64-613
Rechnerarchitekturen und Mikrosystemtechnik

http://tams.informatik.uni-hamburg.de/lectures/2011ws/vorlesung/ram

Andreas Mäder

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik

Technische Aspekte Multimodaler Systeme

Wintersemester 2011/2012

Organisation

Zeit	8:30 – 10:00	1. Teil
	10:00 – 10:30	Pause
	10:30 – 12:00	2. Teil

Übungen im Block (jeweils komplette Termine)
genauer Termin wird in der Vorlesung bekanntgegeben

Info auf den Web-Seiten / über STiNE

http://tams.informatik.uni-hamburg.de/lectures/2011ws/vorlesung/ram

Organisation

Zeit	8:30 – 10:00	1. Teil
	10:00 – 10:30	Pause
	10:30 – 12:00	2. Teil

Übungen im Block (jeweils komplette Termine)
genauer Termin wird in der Vorlesung bekanntgegeben

Info auf den Web-Seiten / über STiNE

http://tams.informatik.uni-hamburg.de/lectures/2011ws/vorlesung/ram

Organisation

Zeit	8:30 – 10:00	1. Teil
	10:00 – 10:30	Pause
	10:30 – 12:00	2. Teil

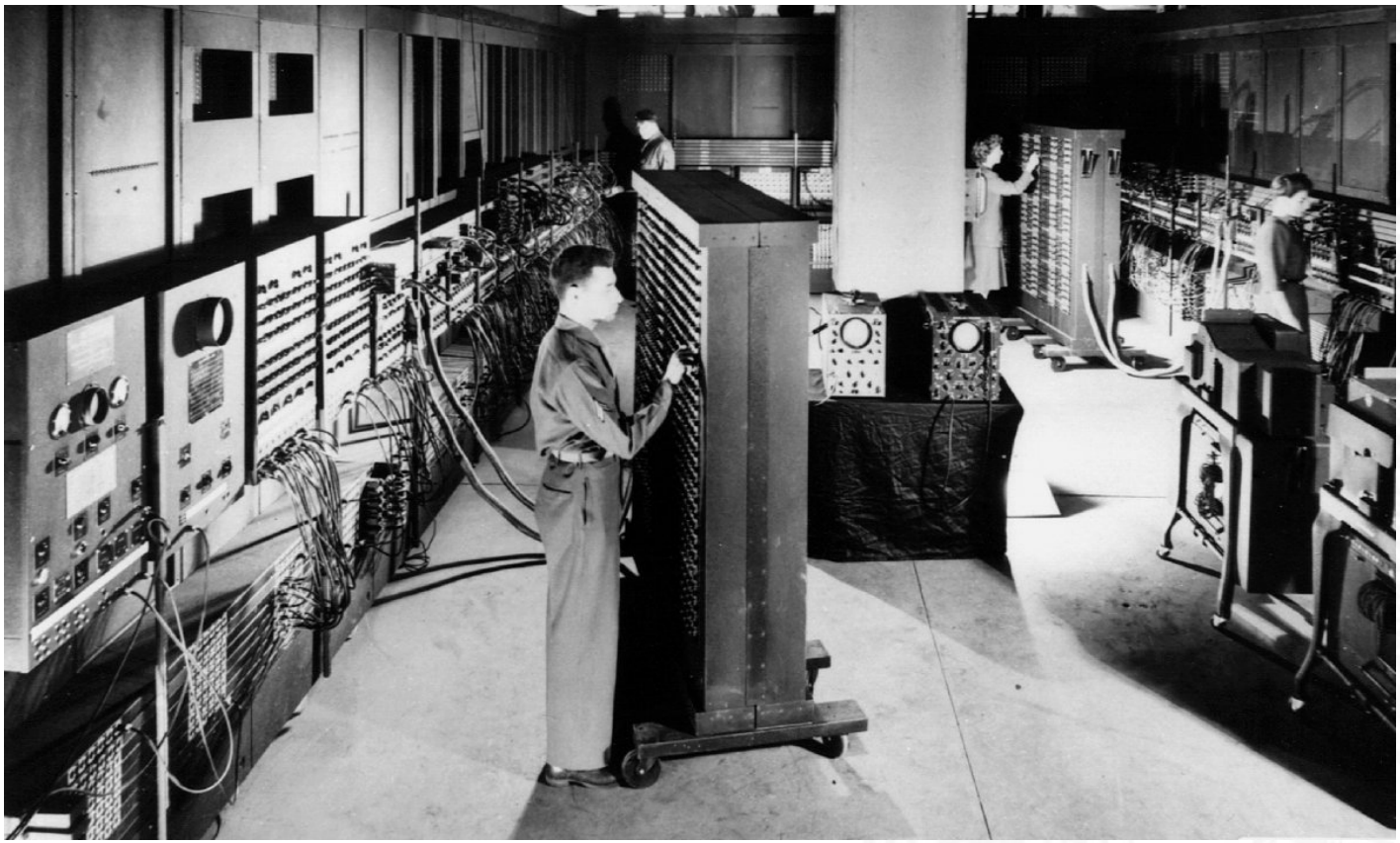
Übungen im Block (jeweils komplette Termine)
genauer Termin wird in der Vorlesung bekanntgegeben

Info auf den Web-Seiten / über STiNE

http://tams.informatik.uni-hamburg.de/lectures/2011ws/vorlesung/ram

Was soll die Vorlesung vermitteln?

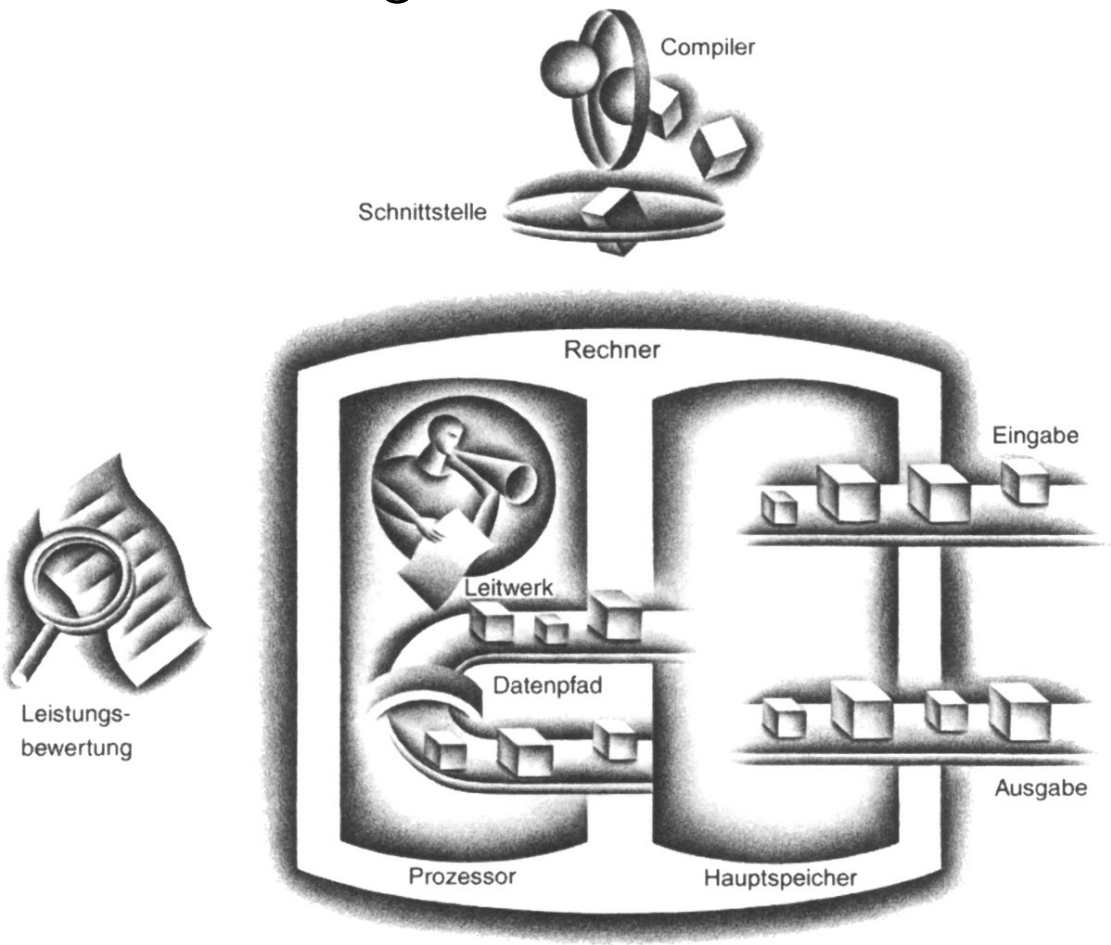
Wie Computer funktionieren...



ENIAC: Electronic Numerical Integrator and Computer, '42

Was soll die Vorlesung vermitteln?

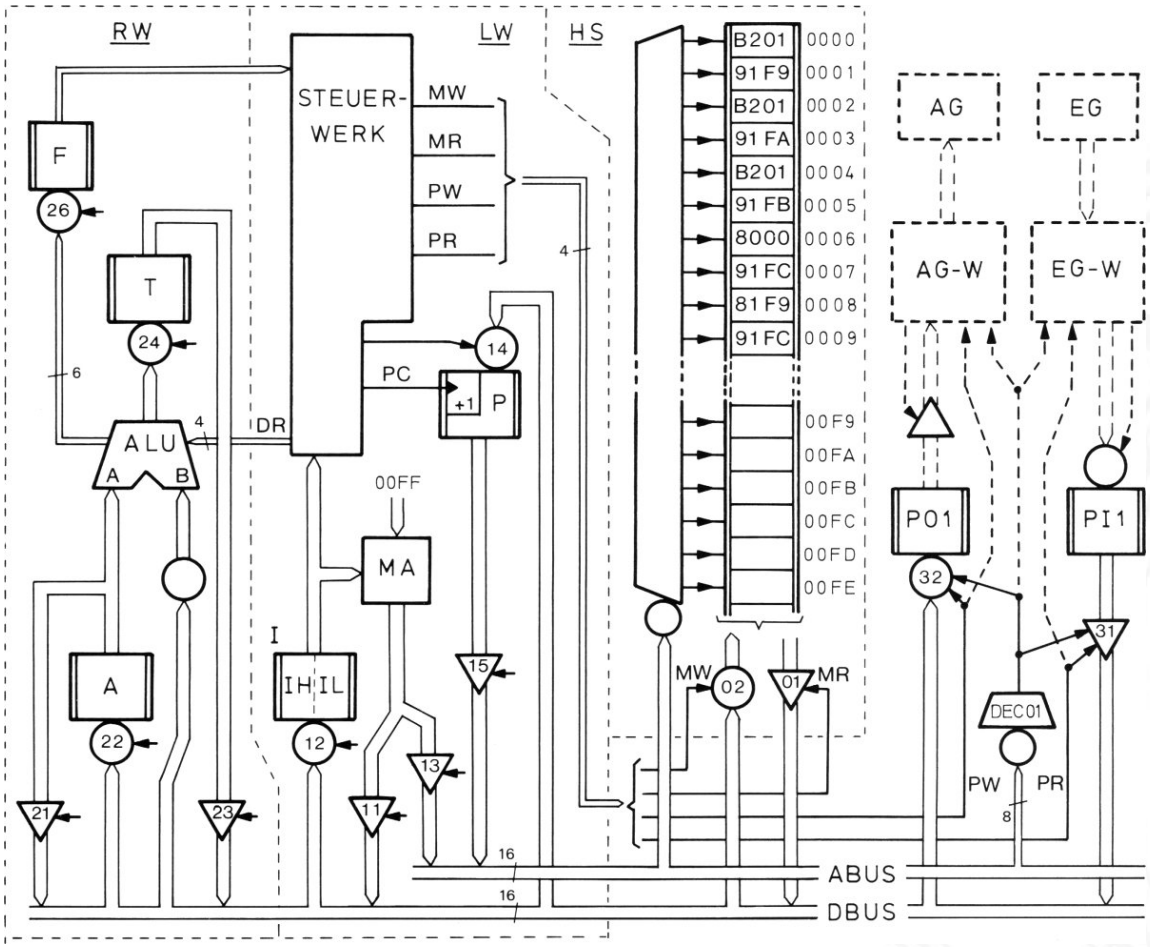
Wie Befehle abgearbeitet werden



Von-Neumann-Architektur [PH05]

Was soll die Vorlesung vermitteln?

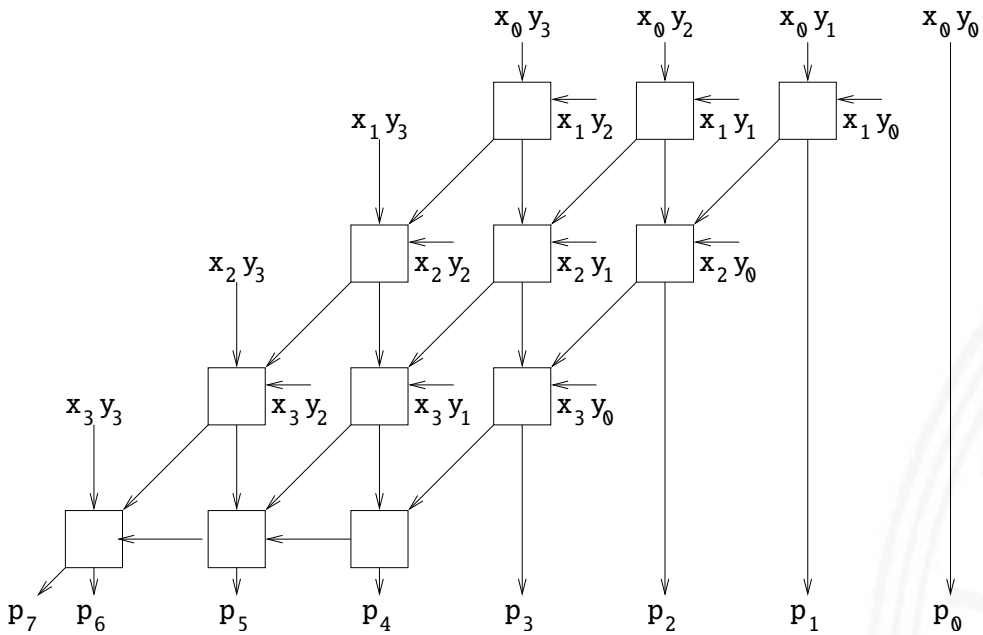
Wie dazu die interne Hardwarearchitektur aussieht



Democom-1 [Lag87]

Was soll die Vorlesung vermitteln?

Wie die Komponenten intern aufgebaut sind



HA

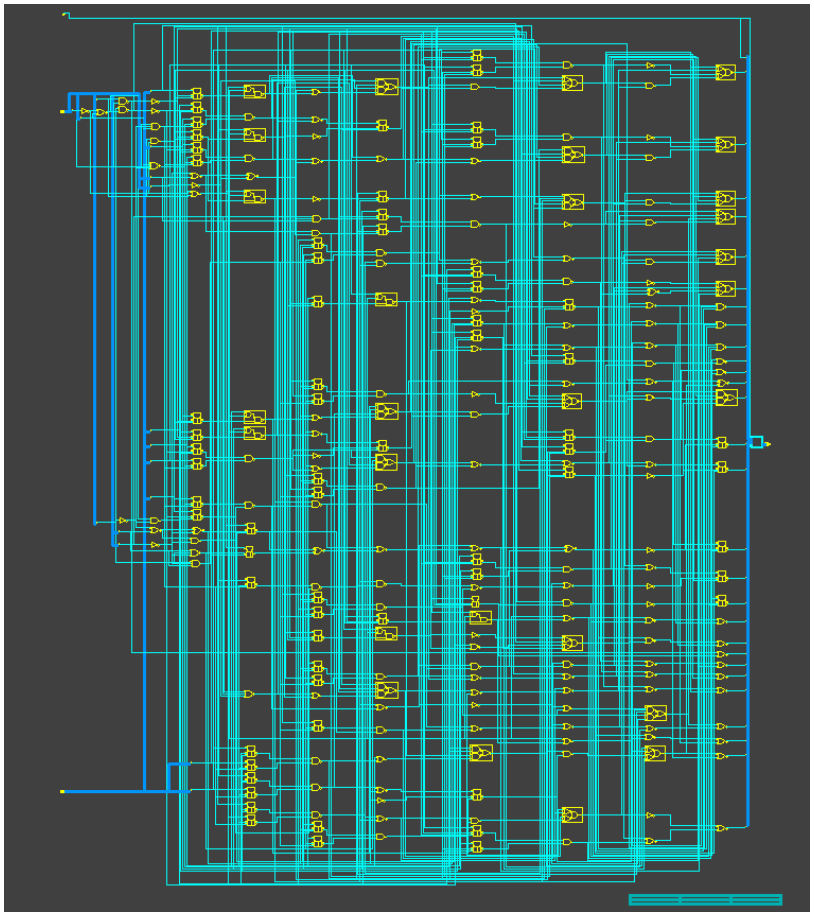
VA

CPA

Blockschaltbild

Was soll die Vorlesung vermitteln?

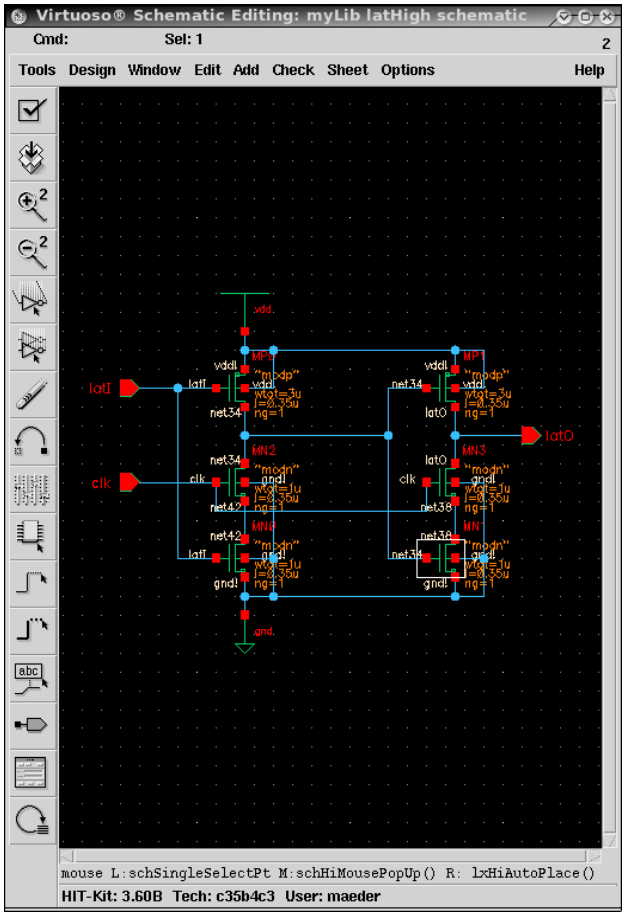
Wie die Komponenten intern aufgebaut sind



Netzliste

Was soll die Vorlesung vermitteln?

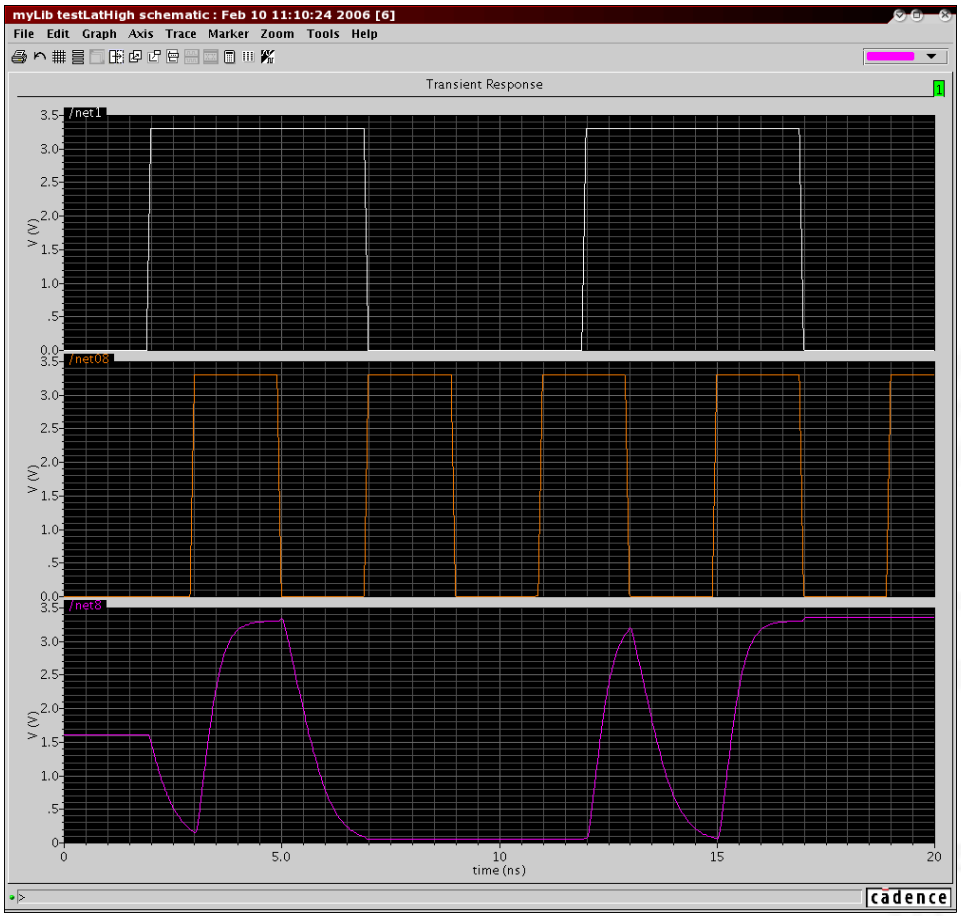
Wie die Schaltungen funktionieren



Transistornetzliste eines Latches

Was soll die Vorlesung vermitteln?

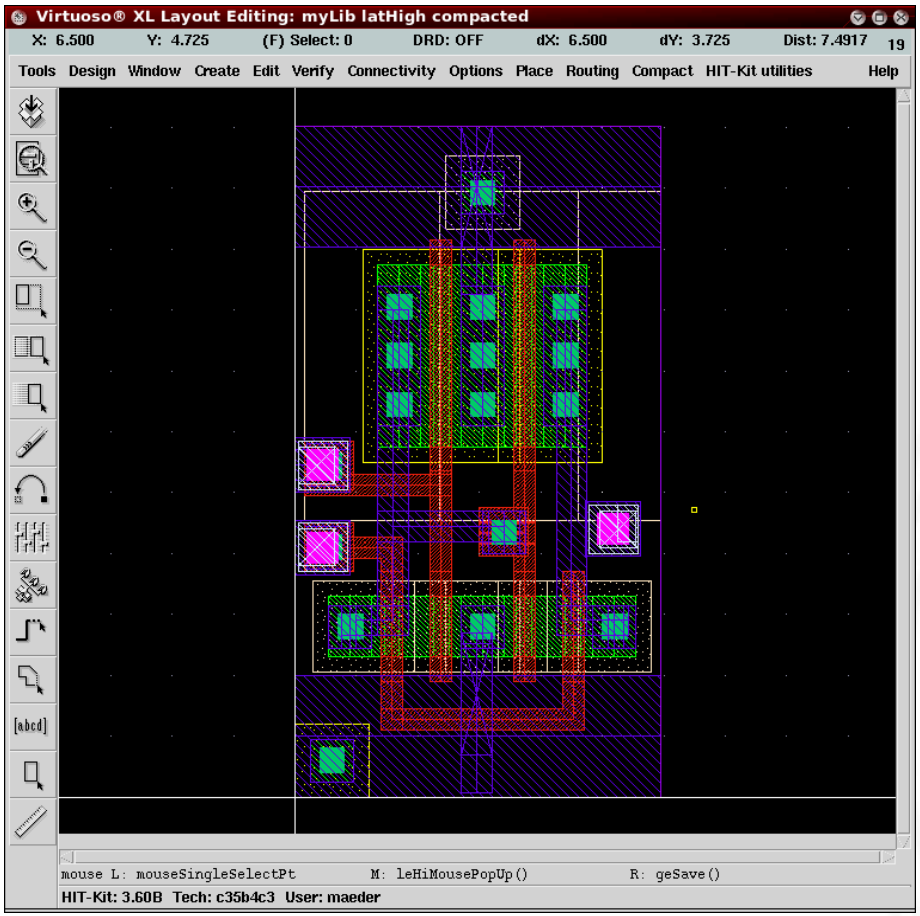
Wie die Schaltungen funktionieren



Impulsdiagramm der elektrischen Simulation

Was soll die Vorlesung vermitteln?

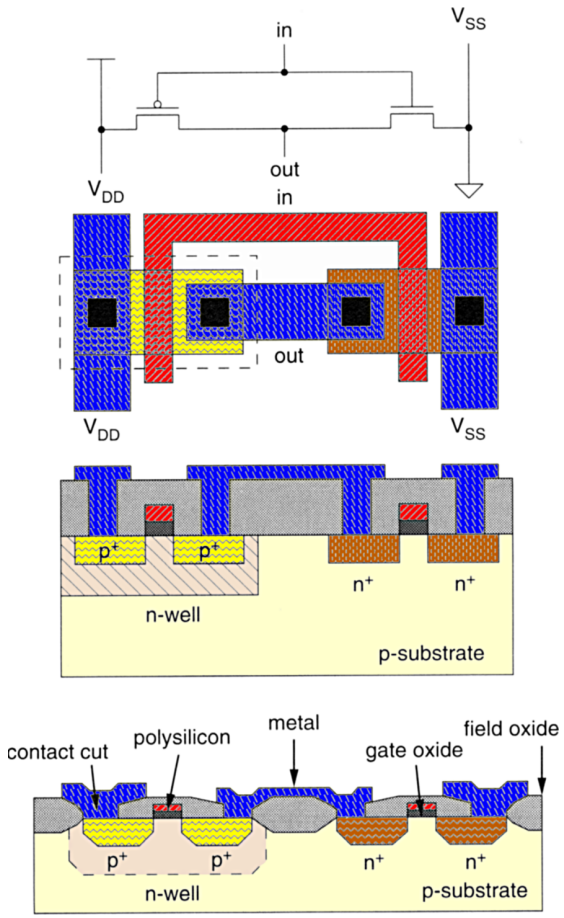
Wie die Schaltungen funktionieren



Layout des Latches

Was soll die Vorlesung vermitteln?

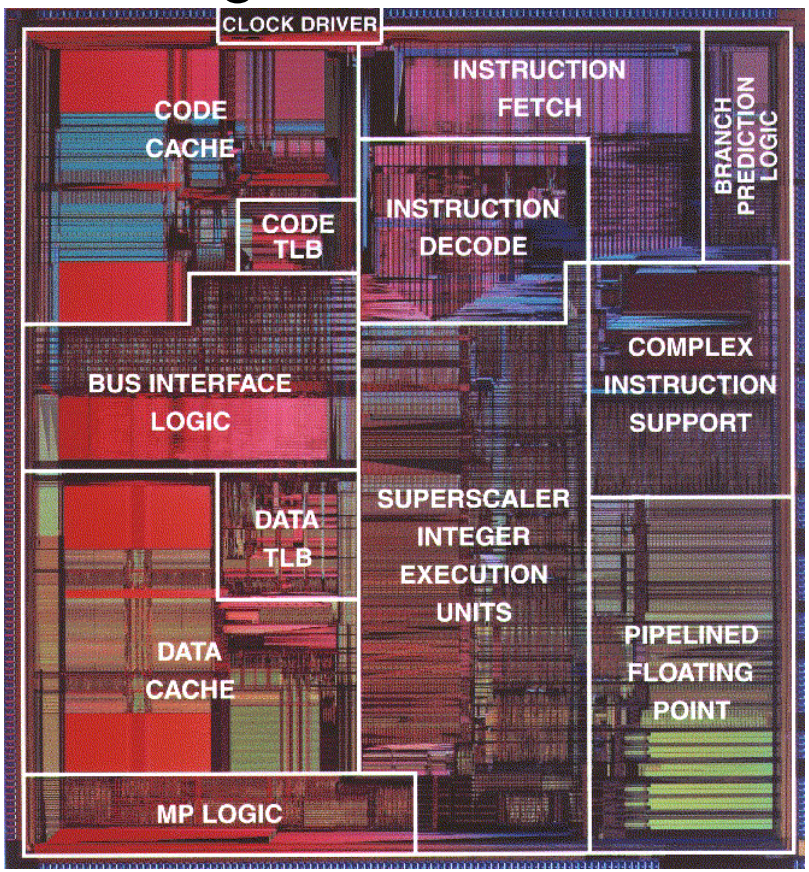
Wie Mikrochips und -Systeme gefertigt werden



CMOS Inverter [WE94]

Was soll die Vorlesung vermitteln?

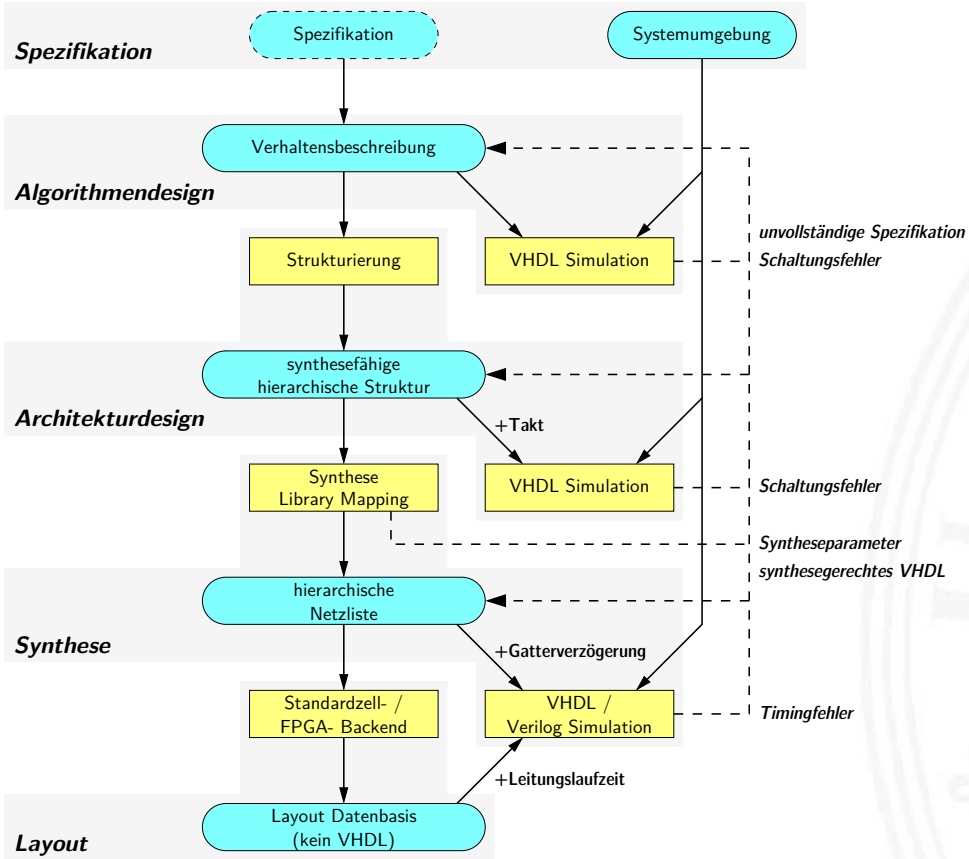
Mit welchen Werkzeugen und nach welchen Prinzipien man eine Schaltung mit mehreren Millionen Transistoren entwirft



Pentium

Was soll die Vorlesung vermitteln?

Mit welchen Werkzeugen und nach welchen Prinzipien man eine Schaltung mit mehreren Millionen Transistoren entwirft



Entwurfsablauf

Gliederung

1. Mikroelektronik
2. Mikrosysteme
3. VLSI- und Systementwurf
4. Rechnerarchitektur

Gliederung

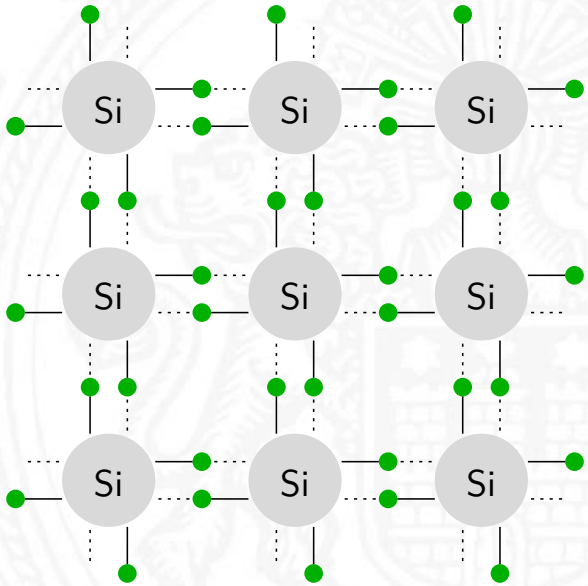
Halbleiter

Halbleiter stehen zwischen *Leitern* (z.B.: Metalle) und *Isolatoren*.

- bei Raumtemperatur quasi nicht-leitend
- Leitfähigkeit steigt mit der Temperatur $\Rightarrow$ Heißleiter
- physikalische Erklärung über Bändermodell
  siehe `http://de.wikipedia.org/wiki/Halbleiter`

Kristallstruktur aus 4-wertigen Atomen

- elementare Halbleiter: Ge, Si
- Verbindungshalbleiter: GaAs, InSb

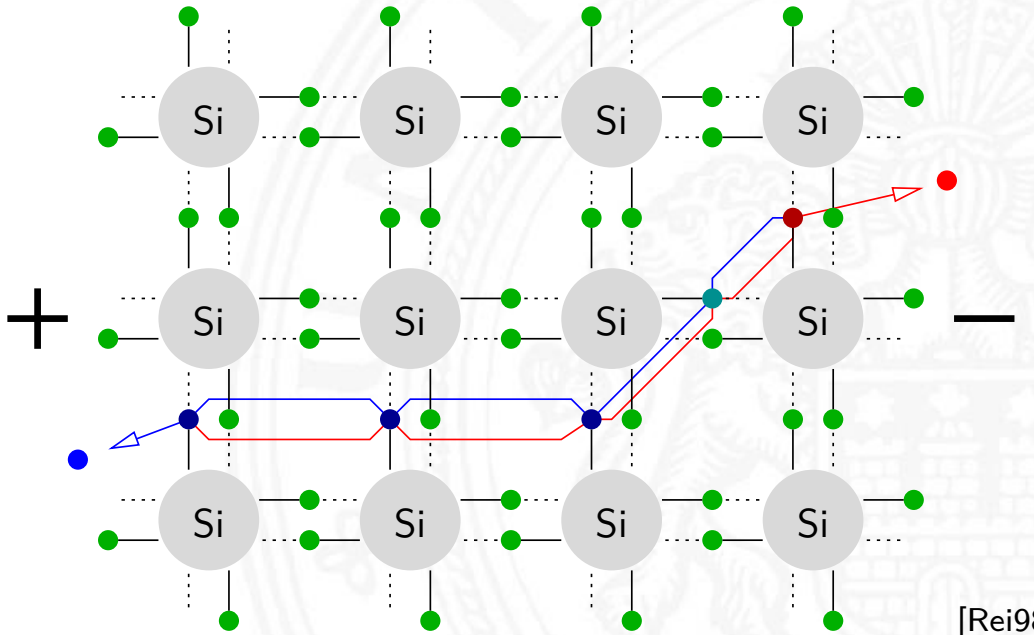


Leitung im undotierten Kristall

- Paarentstehung: Elektronen lösen sich aus Gitterverband Paar aus Elektron und „Loch“ entsteht.
- Rekombination: Elektronen und Löcher verbinden sich quasistatischer Prozess
- Eigenleitungsdichte n_i: temperatur- und materialabhängig

  Si : $1,2 \cdot 10^{10}\ cm^{-3}$
  Ge : $2,5 \cdot 10^{13}\ cm^{-3}$
  GaAs : $1,8 \cdot 10^{6}\ cm^{-3}$
  bei $300°K \approx 20°C$

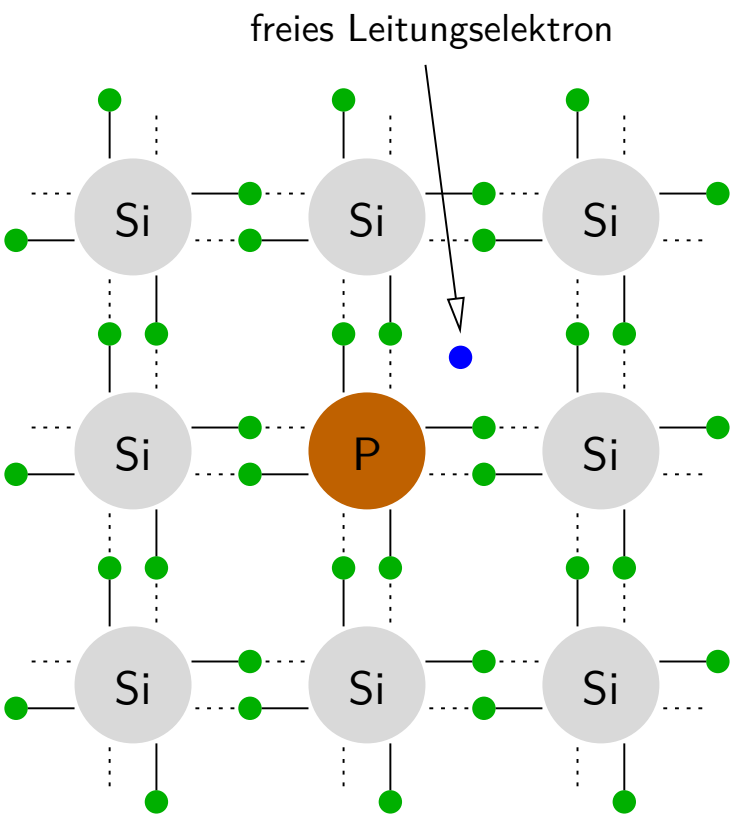
  Atomdichte
  Si : $5 \cdot 10^{22}\ cm^{-3}$
- es gilt: $n_i^2 = n_n \cdot n_p$

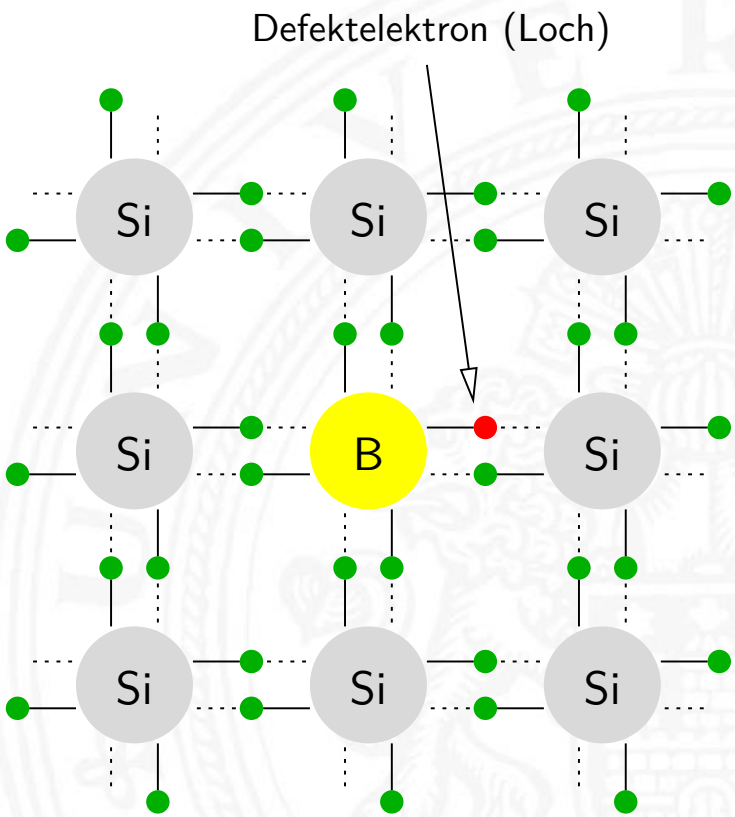


[Rei98]

Dotierung mit Fremdatomen

Ein kleiner Teil der vierwertigen Atome wird durch fünf- oder dreiwertige Atome ersetzt.





Dotierung mit Fremdatomen (cont.)

- Donatoren, Elektronenspender: Phosphor, Arsen, Antimon
- Akzeptoren: Bor, Aluminium, Gallium, Indium
- Dotierungsdichten

Stärke		Fremdatome $[cm^{-3}]$
schwach	n^-, p^-	$10^{15} \ldots 10^{16}$
mittel	n, p	$10^{16} \ldots 10^{19}$
stark	n^+, p^+	$10^{19} \ldots$

- Beweglichkeit μ: materialspezifische Größe

$T = 300°K$		Si	Ge	GaAs $[cm^2/(Vs)]$
Elektronen	μ_n	1500	3900	8500
Löcher	μ_p	450	1500	400

- Leitfähigkeit: ergibt sich aus Material, Beweglichkeit und Ladungsträgerdichte(n)
  $K = e(n_n\mu_n + n_p\mu_p)$

Dotierung mit Fremdatomen (cont.)

- selbst bei hoher Dotierung ist die Leitfähigkeit um Größenordnungen geringer als bei Metallen

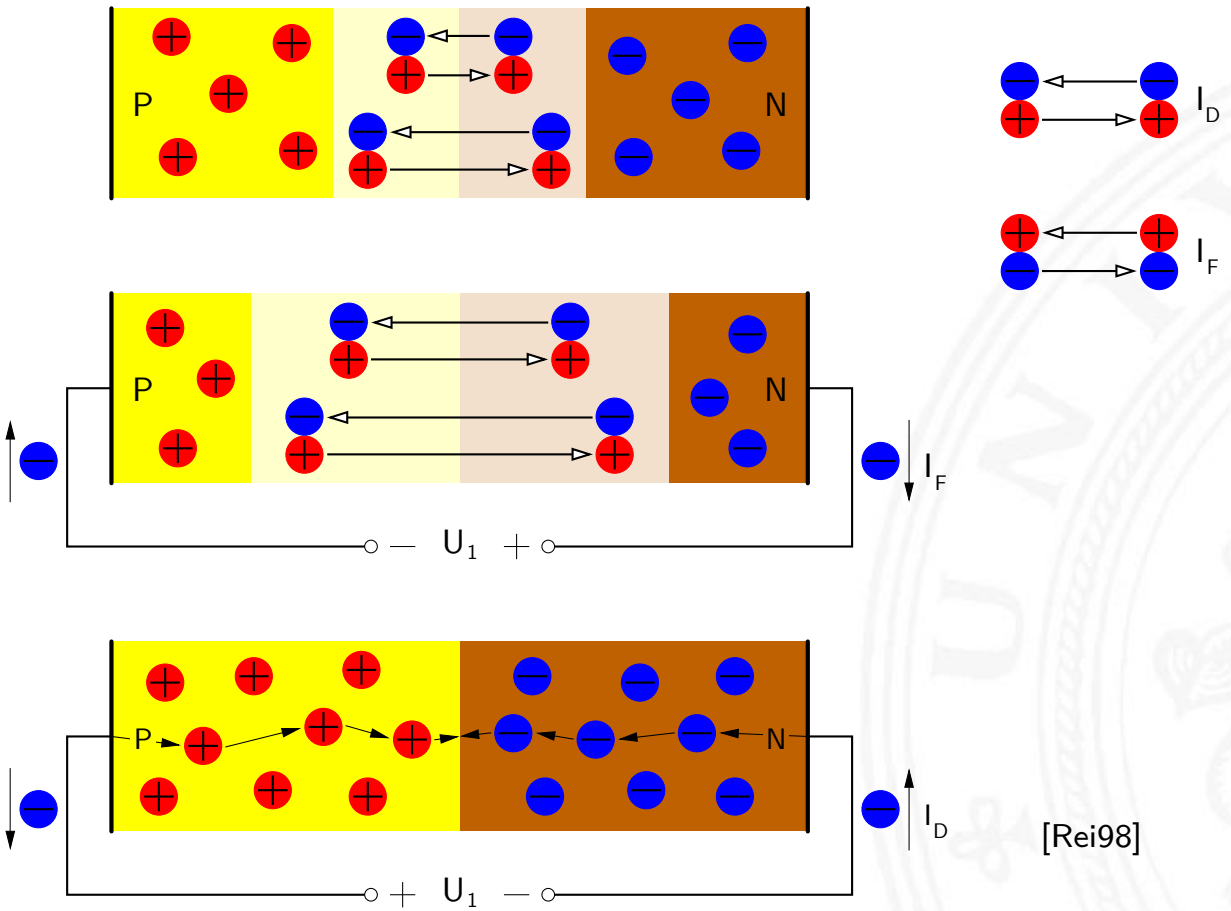
  Si 1 freier Ladungsträger pro 500 Atome ($10^{19}/5 \cdot 10^{22}$)

  Met mindestens 1 Ladungsträger pro Atom

- Majoritätsträger: Ladungsträger in Überzahl (i.d.R. Dotierung)
  Minoritätsträger: Ladungsträger in Unterzahl
  $n_i^2 = n_n \cdot n_p$

Dioden

Ein P/N-Übergang bildet eine Diode



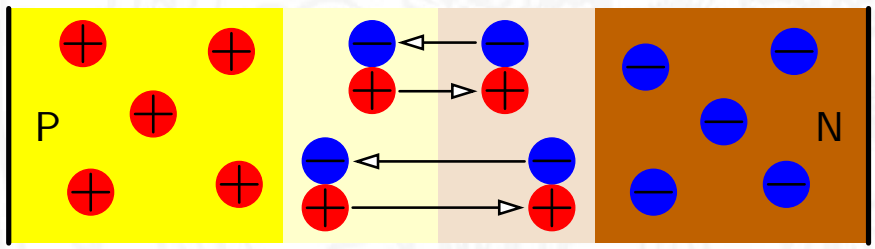
[Rei98]

Dioden (cont.)

- Diffusionsstrom I_D: Ladungsträger diffundieren in das anders dotierte Gebiet
- Feldstom I_F: Löcher und Elektronen verbinden sich, Ladungsträger fehlen
- Raumladung: dadurch entsteht ein elektrisches Feld, es wirkt der Diffusion entgegen und ein Gleichgewicht stellt sich ein $I_D - I_F = 0$
- Diffusionsspannung U_D: material-, temperatur- und dotierungsabhängig

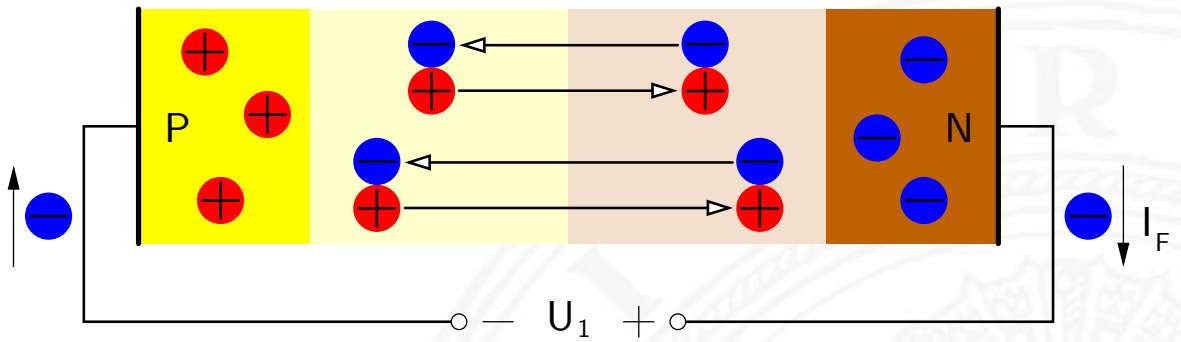
$$U_D = \frac{kT}{e_0} \cdot ln \frac{n_A n_D}{n_i^2}$$

Si : $\approx 0,7$ V
Ge : $\approx 0,3$ V



Dioden (cont.)

- Sperrschicht: die Grenzschicht ohne Ladungsträger
- Sperrrichtung



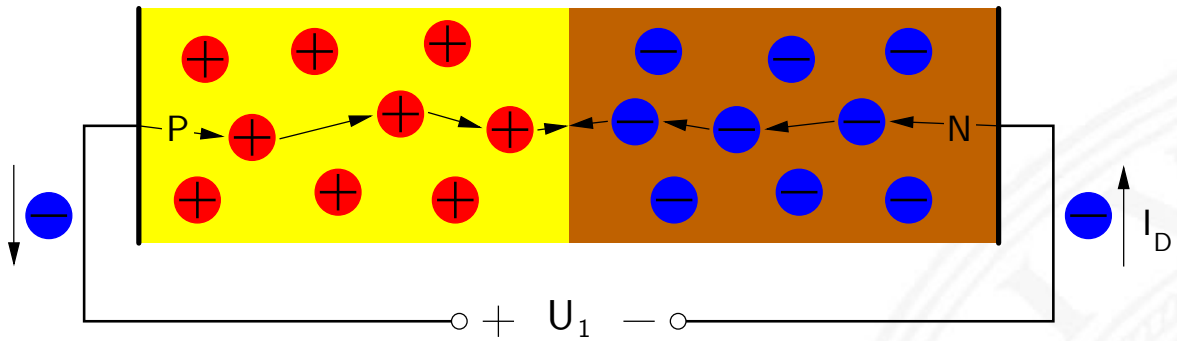
Externe und Diffusionsspannung sind gleich ausgerichtet:
+-Pol an n-Gebiet / −-Pol an p-Gebiet

$\Rightarrow$ Sperrschicht wird breiter

$\Rightarrow$ Energie: $e_0(U_D + U)$

Dioden (cont.)

- Durchlassrichtung



Externe und Diffusionsspannung sind unterschiedlich ausgerichtet: +-Pol an p-Gebiet / −-Pol an n-Gebiet

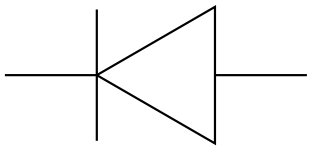
$\Rightarrow$ Sperrschicht verschwindet

$\Rightarrow$ ein Strom kann fließen

Dioden (cont.)

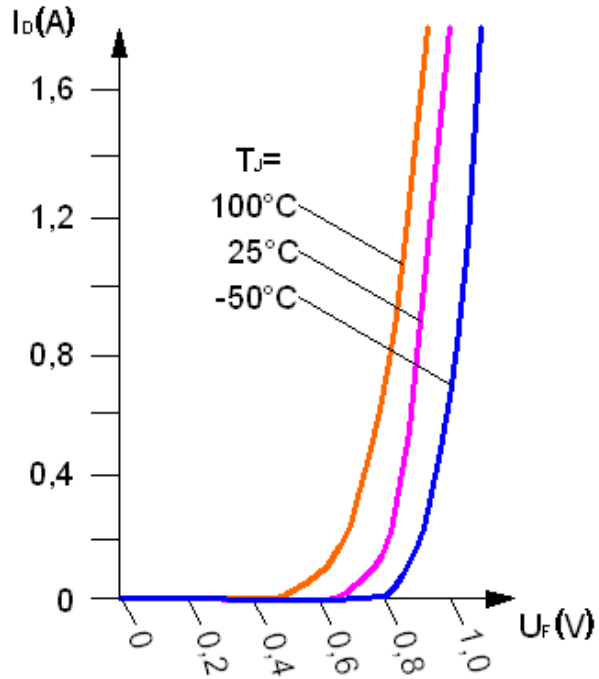
- Diodenkennlinie

$$I = I_D - I_F$$
$$I = I_F(e^{\frac{e_0 U}{kT}} - 1)$$



Kathode
n-dotiert

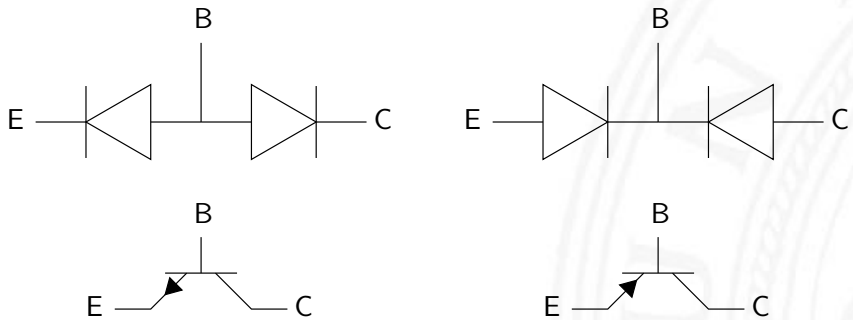
Anode
p-dotiert



Bipolar-Transistoren

Entsprechen zwei gegeneinander geschaltete Dioden

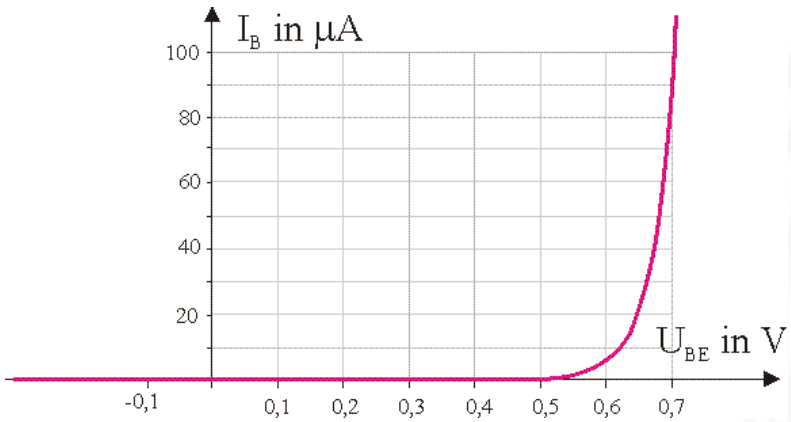
- Anschlüsse: **E**mitter emittiert Ladungsträger in Basis
  Basis gemeinsamer Anschluß
  Collector sammelt Ladungsträger aus Basis
- Typen: NPN-Transistor PNP-Transistor



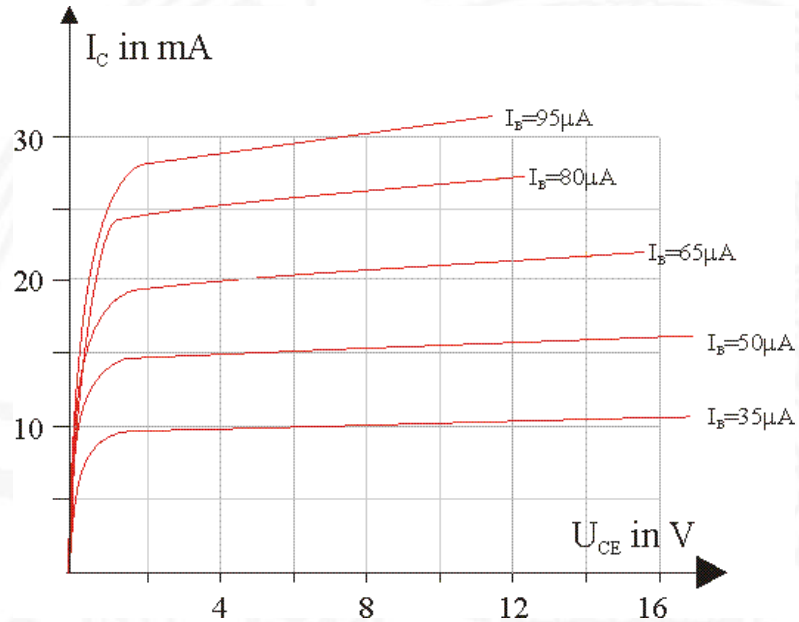
- Im Normalbetrieb sind die Emitter-Basis Diode in Durchlass- und die Collector-Basis Diode in Sperrrichtung gepolt

Bipolar-Transistoren (cont.)

- Funktionsweise: durch die leitende Emitter-Basis Diode wird in der Basis die Minoritätsträgerdichte erhöht. Sind sehr viele Ladungsträger in der Basis, so können sie die Sperrschicht zum Collector durchdringen.



Eingangskennlinie

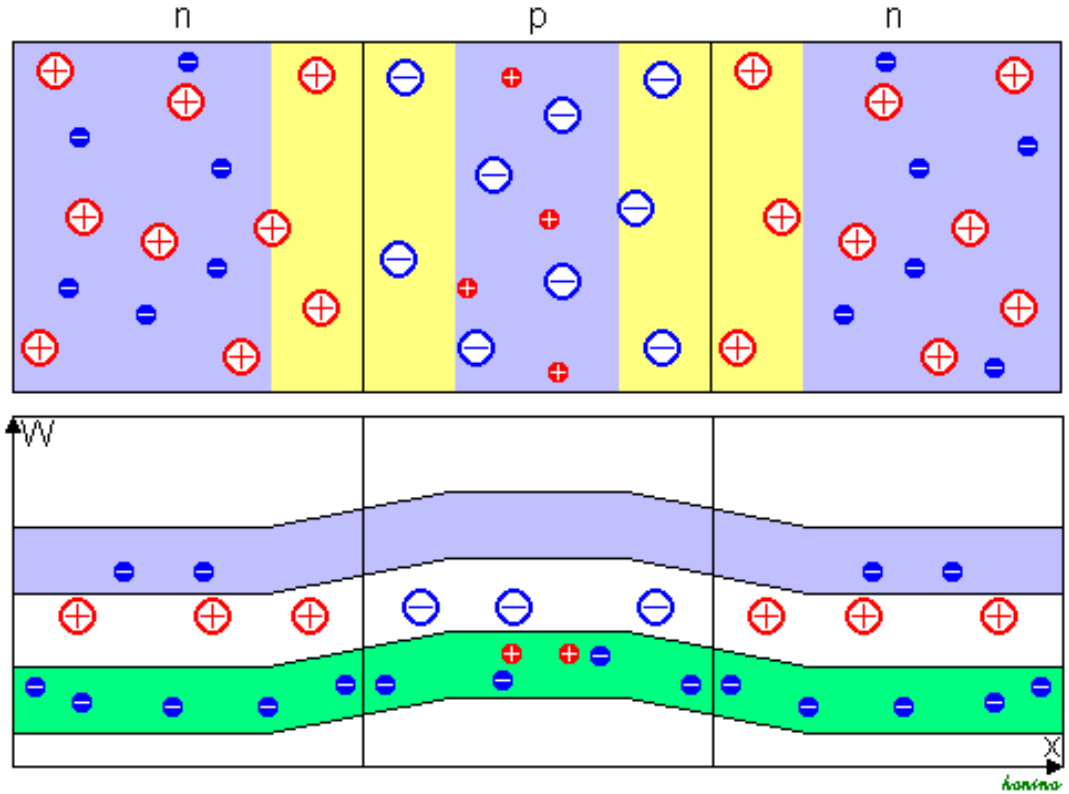


Ausgangskennlinienfeld

Bipolar-Transistoren (cont.)

Erklärung nach dem Bändermodell, siehe

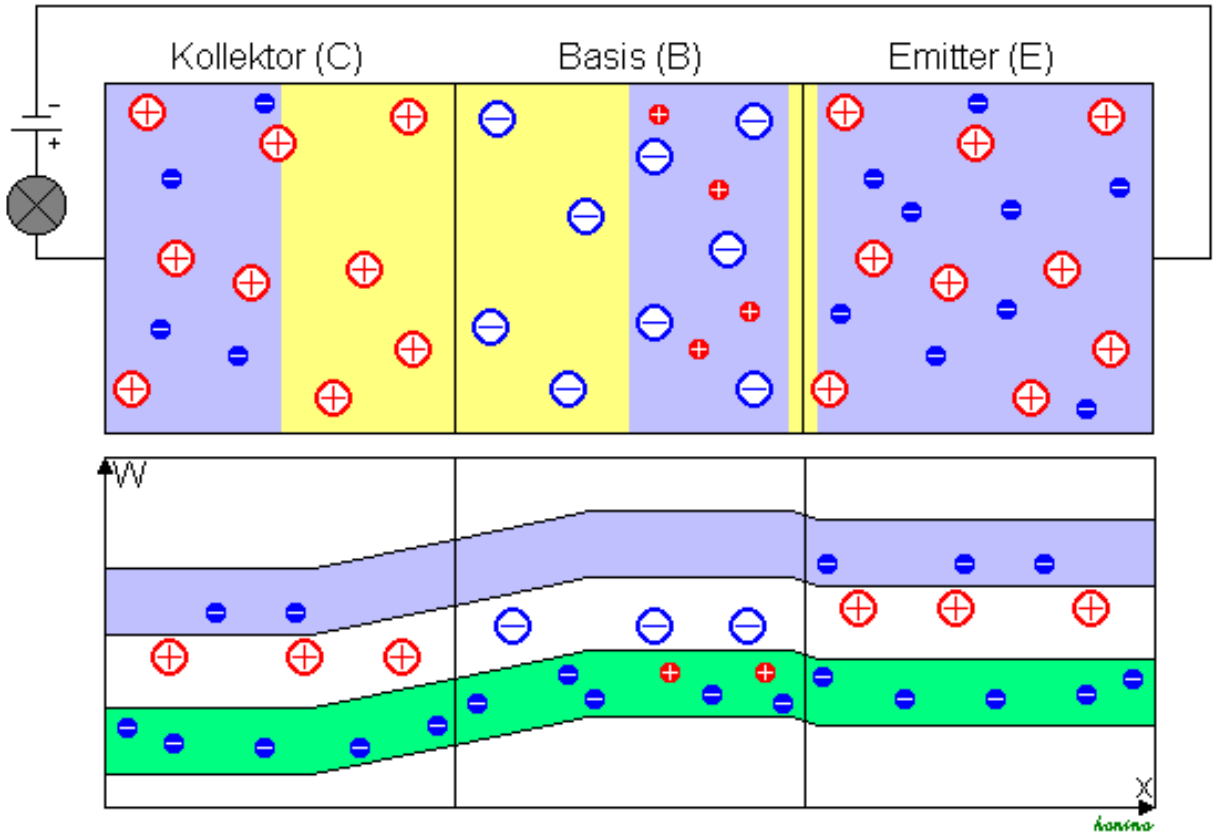
`http://de.wikipedia.org/wiki/Bipolartransistor`



unbeschaltet

Bipolar-Transistoren (cont.)

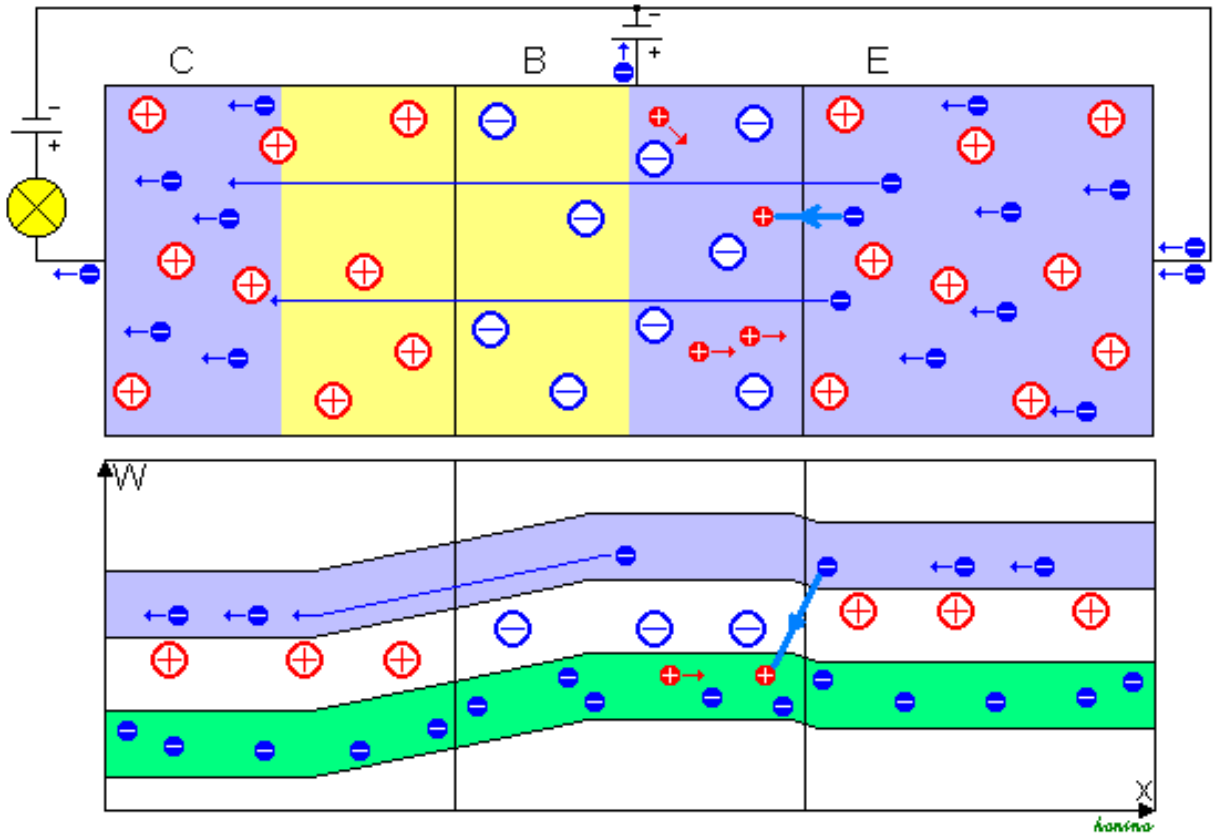
Anlegen einer externen Spannung



U_{CE}

Bipolar-Transistoren (cont.)

Der leitende Transistor



U_{CE} und U_{BE}

Bipolar-Transistoren (cont.)

- Stromverstärkungsfaktor β: mit einem kleinen Emitter-Basis Strom wird so ein großer Emitter-Collector Strom gesteuert. $I_C = \beta \cdot I_B$, typische Werte: $10 < \beta < 10000$
- obige Überlegungen basieren auf der *Emitterschaltung*, die zur Leistungsverstärkung benutzt wird.

  Andere Schaltungsvarianten sowie die unterschiedlichen Betriebsmodi sollen hier nicht weiter diskutiert werden. . . [1]

[1] eine kurze Übersicht findet sich z.B. in `http://de.wikipedia.org/wiki/Transistorgrundschaltungen`, genaueres in Standardwerken der Elektrotechnik, wie beispielsweise [TS09]

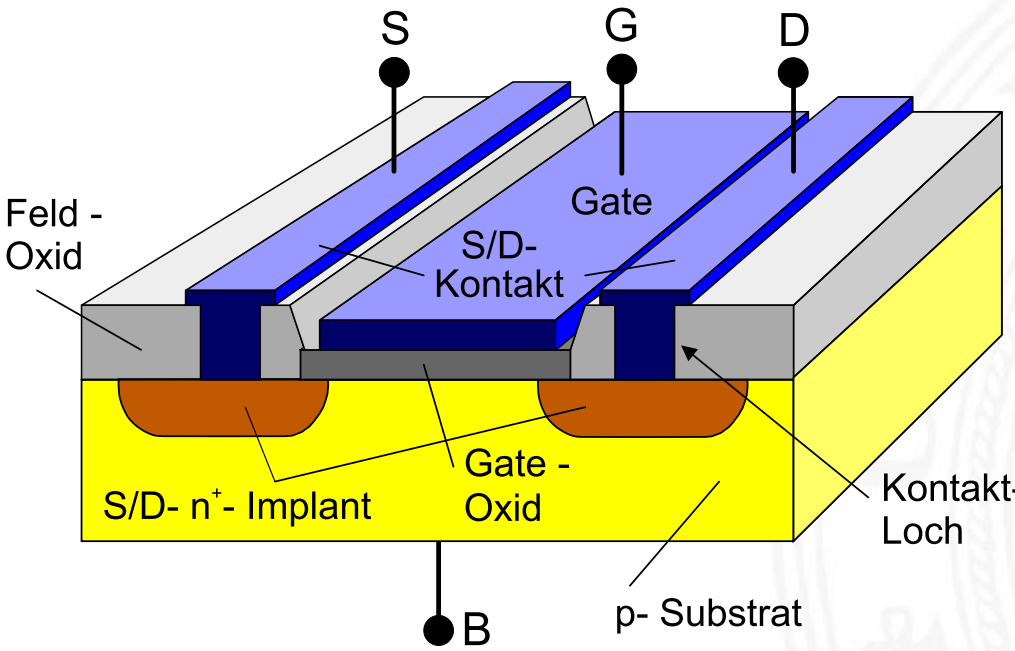
MOS-Transistoren

- MOS: Metal Oxide Semiconductor
  FET : Feldeffekttransistor
  - `http://olli.informatik.uni-oldenburg.de/weTEiS/weteis/tutorium.htm`
  - `http://de.wikipedia.org/wiki/Feldeffekttransistor`
  - `http://de.wikipedia.org/wiki/MOSFET`

  Literatur: [TS09, WE94]
- unipolarer Transistor: nur eine Art von Ladungsträgern, die Majoritätsträger, ist am Stromfluss beteiligt

MOS-Transistoren (cont.)

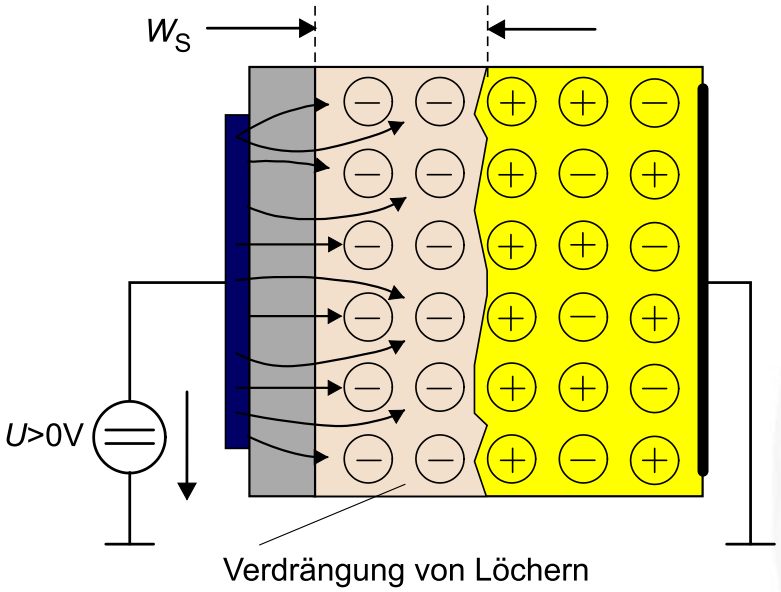
- Anschlüsse:
  - **S**ource — Quelle der Ladungsträger
  - **G**ate — steuert den Stromfluss
  - **D**rain — Senke der Ladungsträger
  - **B**ulk — siehe „Herstellungstechnik“



NMOS Transistor [Rei98]

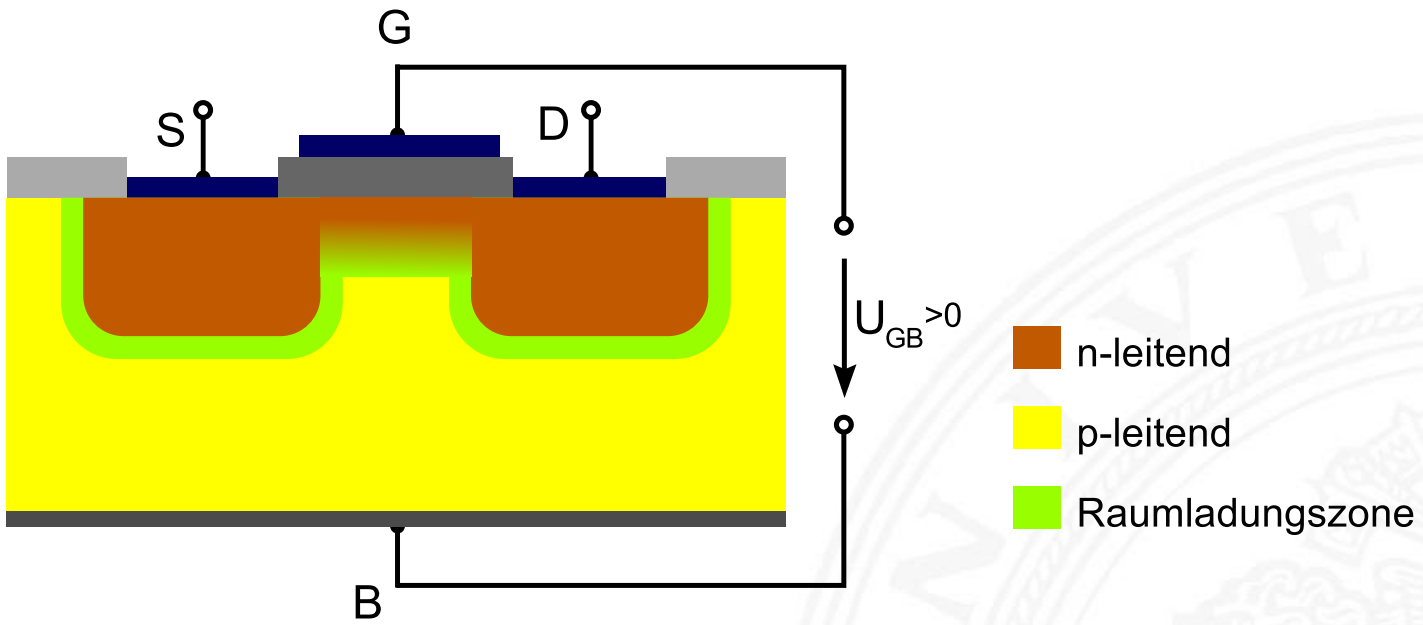
MOS-Transistoren (cont.)

- Funktionsweise: die Ladung des Gates erzeugt ein elektrisches Feld. Durch Inversion werden Ladungsträger unterhalb des Gates verdrängt und ein leitender Kanal zwischen Source und Drain entsteht.



MOS-Kondensator [Rei98]

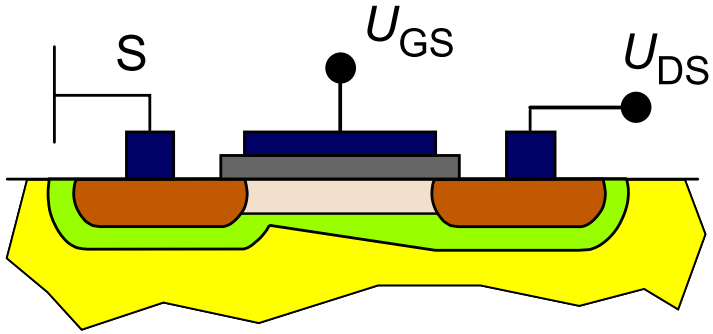
MOS-Transistoren (cont.)



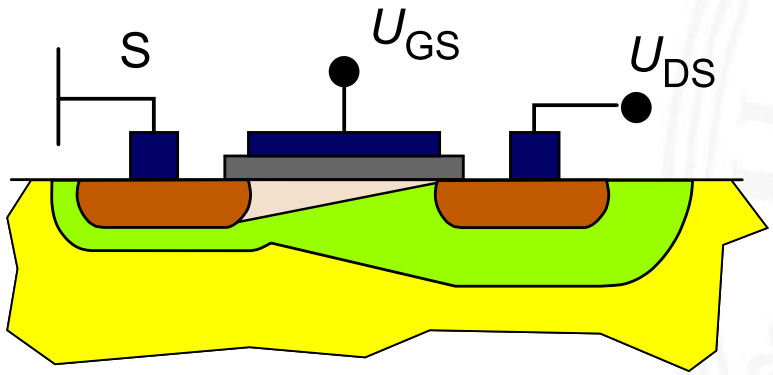
- Schwellspannung U_P: abhängig von der Dotierungsdichte, den Parametern des MOS-Kondensators (Dicke und Material der Gate-Isolationsschicht)...
  U_P möglichst klein: 0,3...0,8 V früher: deutlich mehr

MOS-Transistoren (cont.)

- $U_{DS} \ll U_{GS} - U_P$ normaler Betrieb (Triodenbereich)

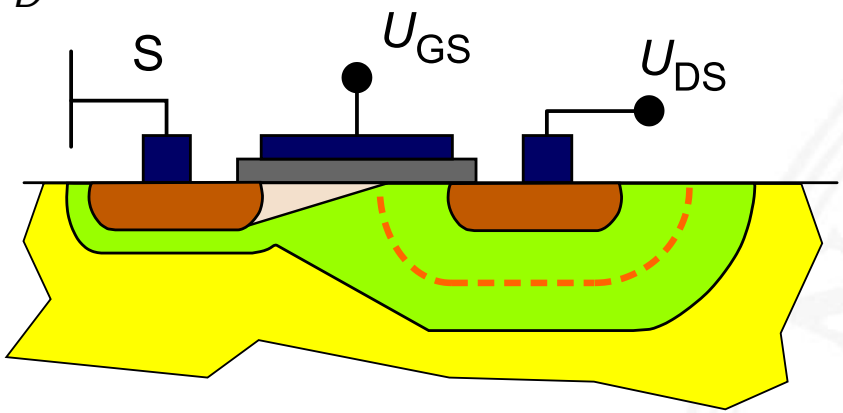


- $U_{DS} = U_{GS} - U_P$ Kanalabschnürung
  Spannungsabfall zwischen S und D durch den Kanalwiderstand



MOS-Transistoren (cont.)

- $U_{DS} > U_{GS} - U_P$ Kanalverkürzung (Sättigungsbereich)
  Der Kanal wird weiter verkürzt, die Spannung U_{DS} bewirkt ein virtuell größeres Drain durch Inversion.
  I_D wächst nur noch minimal.

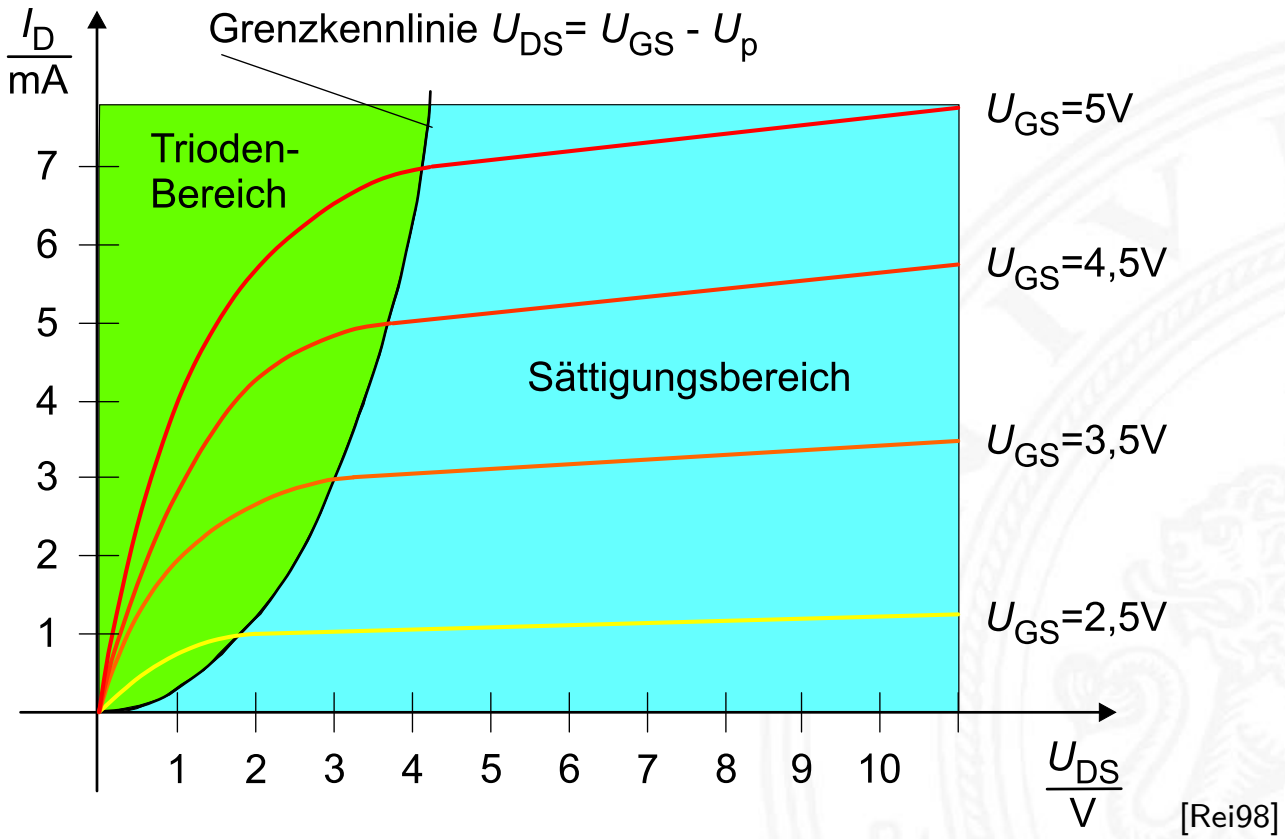


$\Rightarrow$ kurze Kanäle aktueller Submikronprozesse können allein durch hohe Spannungen U_{DS} leitend werden (Durchgreifbetrieb)

$\Rightarrow$ einer der Gründe für sinkende Versorgungsspannungen

MOS-Transistoren (cont.)

- Kennlinienfeld

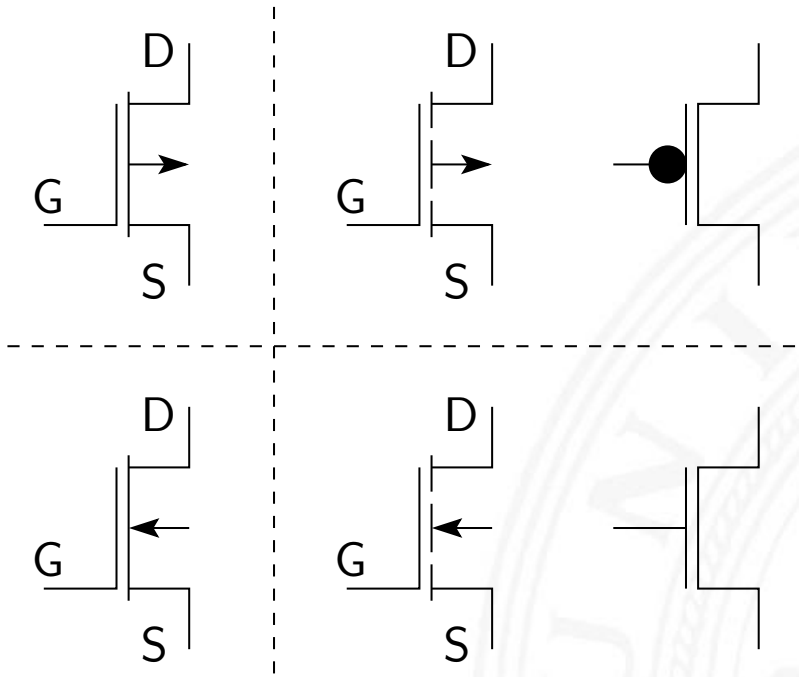


[Rei98]

MOS-Transistoren (cont.)

- Typen

Kennlinienfeld oben: N-Kanal, selbstsperrend

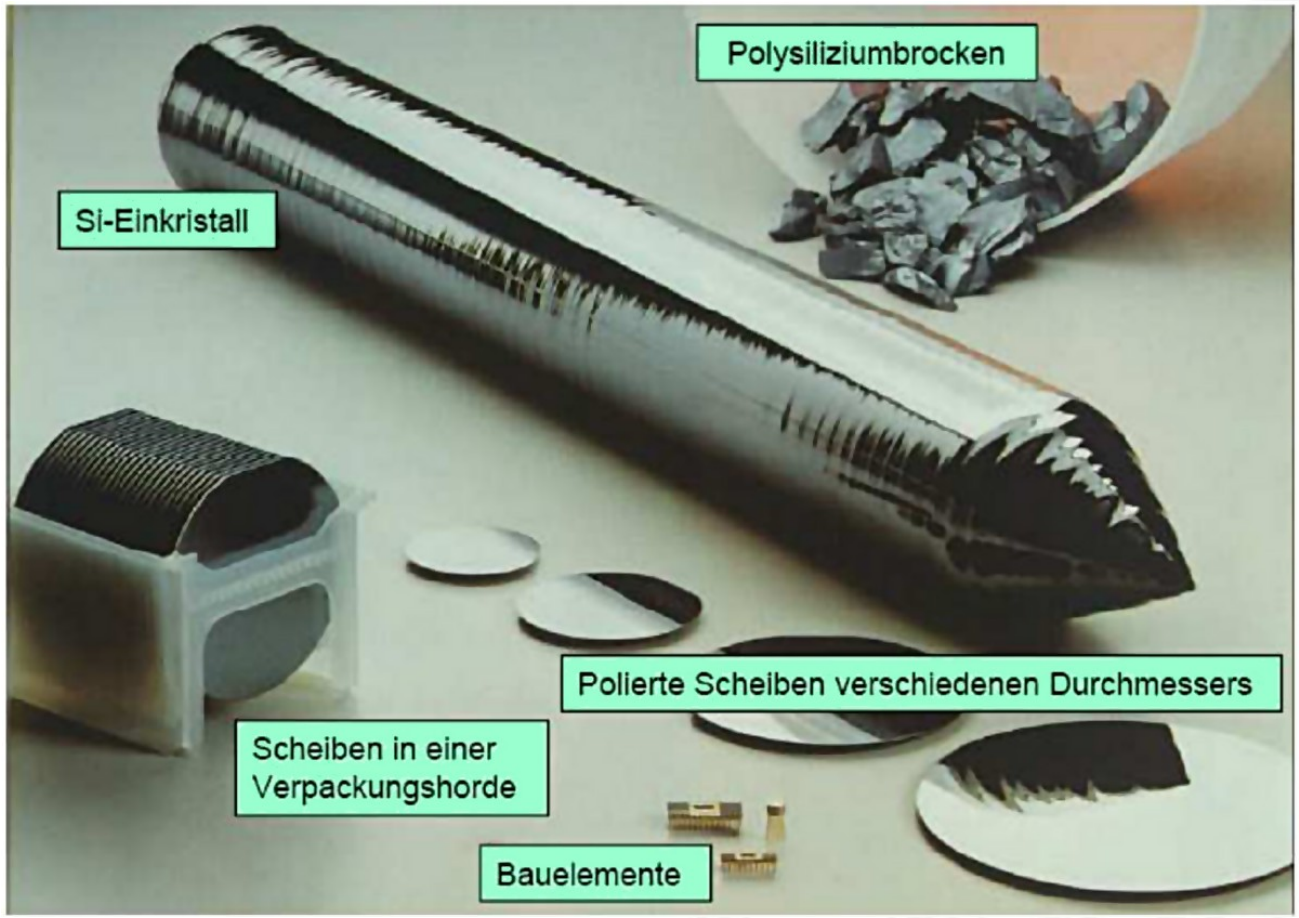


selbstleitend — Verarmungs-

selbstsperrend — Anreicherungstyp

Herstellung von Halbleitermaterial

Übersicht in: `http://de.wikipedia.org/wiki/Silicium`



Rohsilizium

- Siliziumoxid (SiO_2): Sand, Kies...
  ca. 20 % der Erdkruste
- Herstellung im Lichtbogenofen: Siliziumoxid + Koks
  $SiO_2 + 2C \rightarrow Si + 2CO$
- amorphe Struktur, polykristallin
- noch ca. 2 % Verunreinigungen (Fe, Al...)

Solarsilizium

Ziel: Fremdatome aus dem Silizium entfernen

1. Chemische Bindung des Siliziums
   $Si + 3HCl \rightarrow SiHCl_3 + H_2$
   Reaktion mit Salzsäure erzeugt

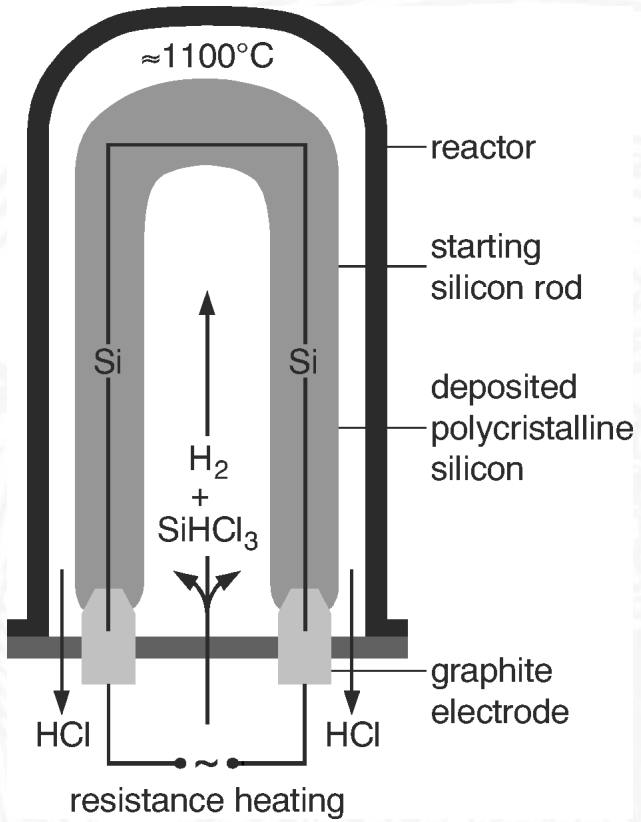
$SiHCl_3$	Trichlorsilan
$SiCl_4$	Siliziumchlorid (10%)
SiH_2Cl_4	div. andere Chlorsilane/Silane
$FeCl_2$, $AlCl_3$	div. Metallchloride

2. Verschiedene Kondensations- und Destillationsschritte trennen Fremdverbindungen ab, hochreines Trichlorsilan entsteht
   $<$ 1ppm Verunreinigungen

Solarsilizium (cont.)

3. CVD (Chemical Vapour Deposition) zur Abscheidung des Trichlorsilans zu elementarem Silizium
   $4SiHCl_3 \rightarrow Si + 3SiCl_4 + 2H_2$

$\Rightarrow$ polykristallines Silizium
$<$ 0,1ppm Verunreinigungen



Siliziumeinkristall

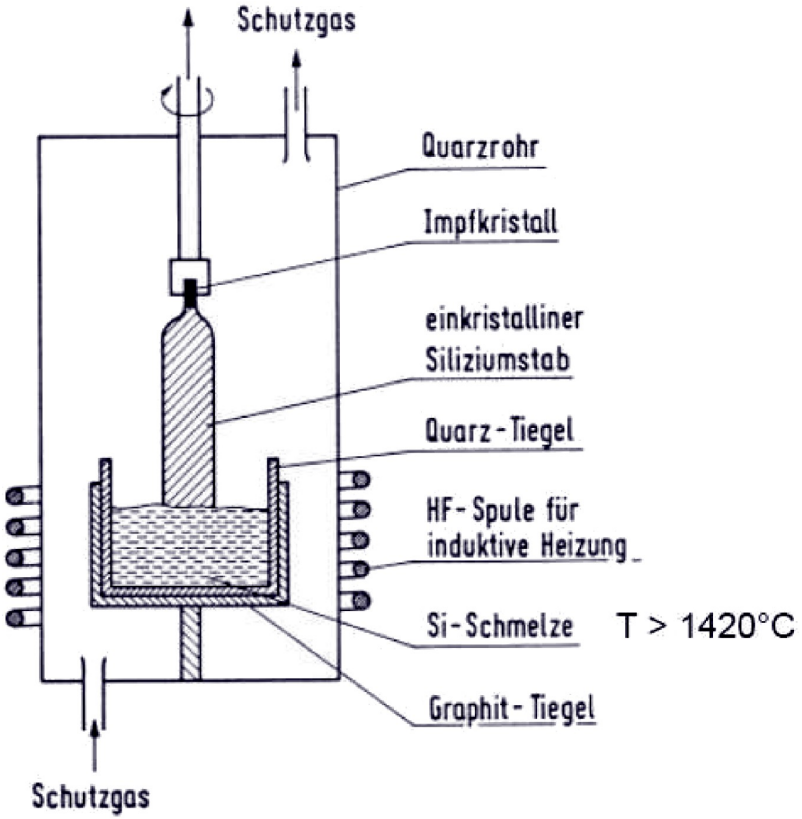
Weitere Ziele

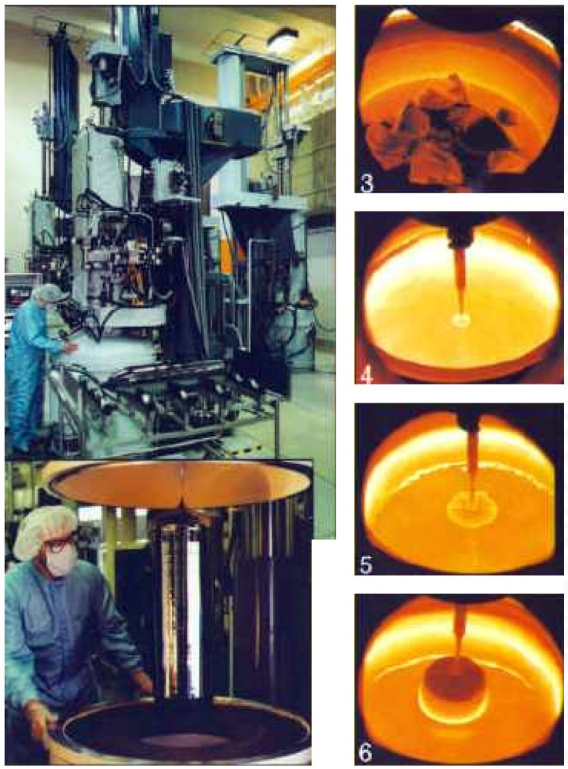
- Einkristalline Struktur erzeugen
- Reinheit für Halbleiterherstellung erhöhen
  $<$, $\ll$ 1ppb
- ggf. Dotierung durch Fremdatome einbringen

Es gibt dazu mehrere technische Verfahren, bei denen das polykristalline Silizium geschmolzen wird und sich monokristallin an einen Impfkristall anlagert.

Siliziumeinkristall (cont.)

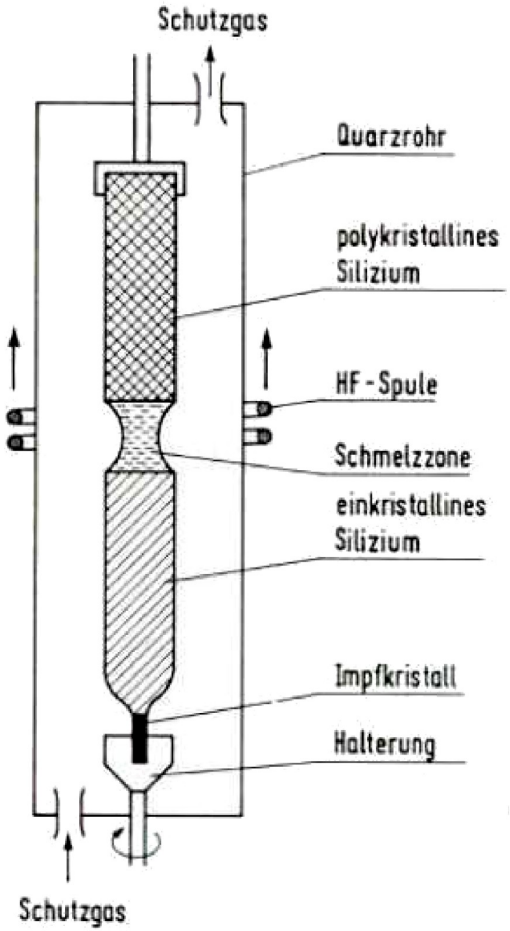
Czochralski-Verfahren (Tiegelziehverfahren)





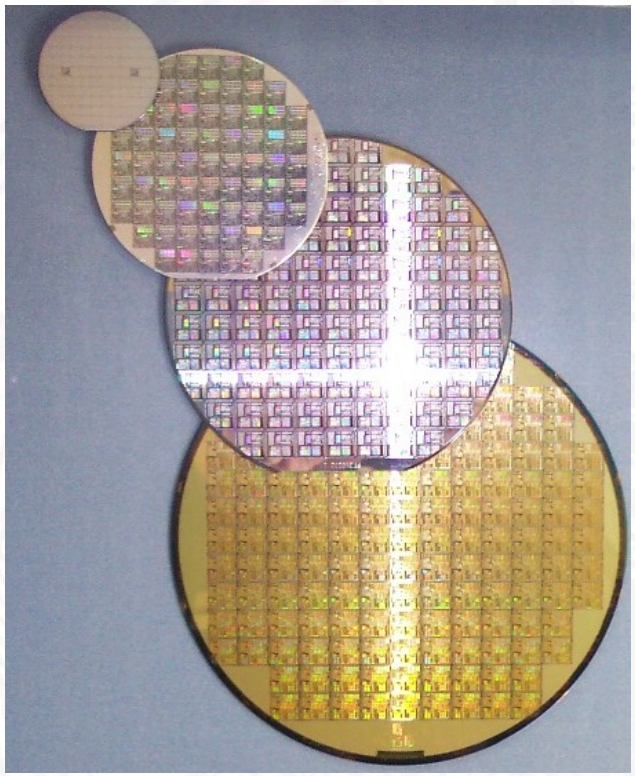
Siliziumeinkristall (cont.)

Zonenschmelz- / Zonenziehverfahren



Wafer

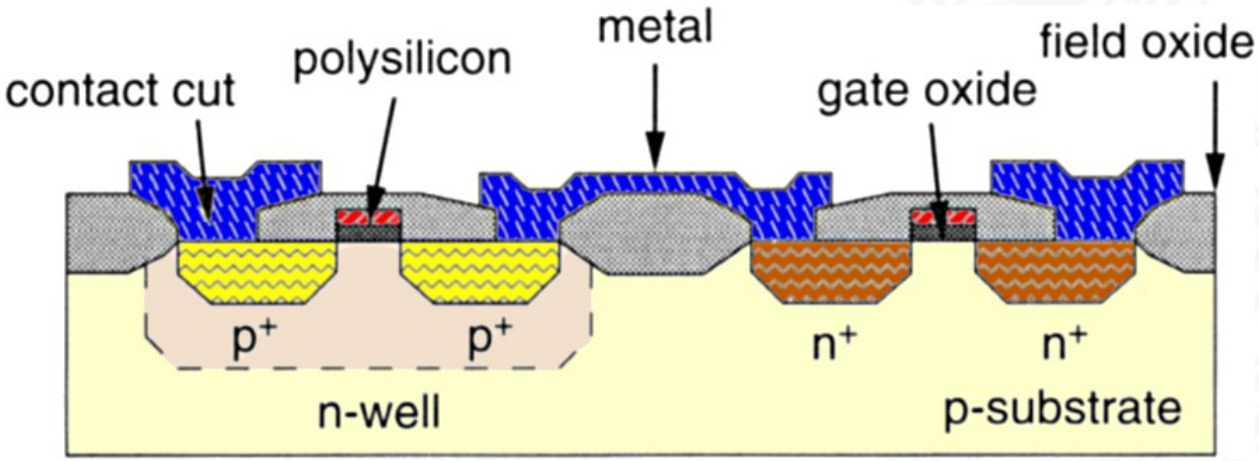
- weitere Bearbeitungsschritte:
  zersägen, schleifen, läppen, ätzen, polieren
- Durchmesser bis 30 cm
  2014: 45 cm [ITRS09]
  Dicke < 1mm
  Rauhigkeit $\approx$ nm
- Markieren: Kerben, Lasercodes. . .
  früher „flats“



Technologien

Technologien zur Erstellung von Halbleiterstrukturen

- Epitaxie: Aufwachsen von Schichten
- Oxidation von Siliziumoberflächen: SiO_2 als Isolator
- Strukturerzeugung durch Lithografie
- Dotierung des Kristalls durch Ionenimplantation oder Diffusion
- Ätzprozesse: Abtragen von Schichten



[WE94]

Technologien (cont.)

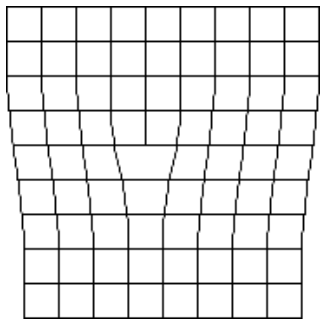
Links

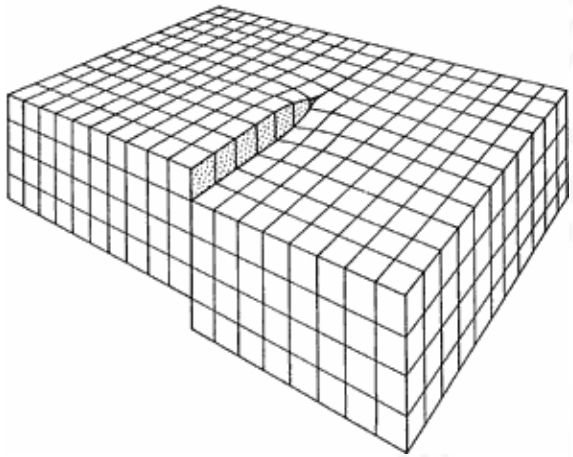
- http://www.halbleiter.org
- http://www.siliconfareast.com
- http://www2.renesas.com/fab/en
- http://en.wikipedia.org/wiki/Semiconductor_device_fabrication
- http://de.wikipedia.org/wiki/Halbleitertechnik

Epitaxie

Aufwachsen von Schichten an der Oberfläche des Einkristalls durch chemische oder physikalische Prozesse

- Homoepitaxie: Schicht und Substrat aus gleichem Material
- Heteroepitaxie: unterschiedliche Materialien





Spannungen und Versetzungen im Kristall durch verschiedene Gitterkonstanten

Epitaxie (cont.)

Unterschiedliche Verfahren – Quellen und Transportmechanismen

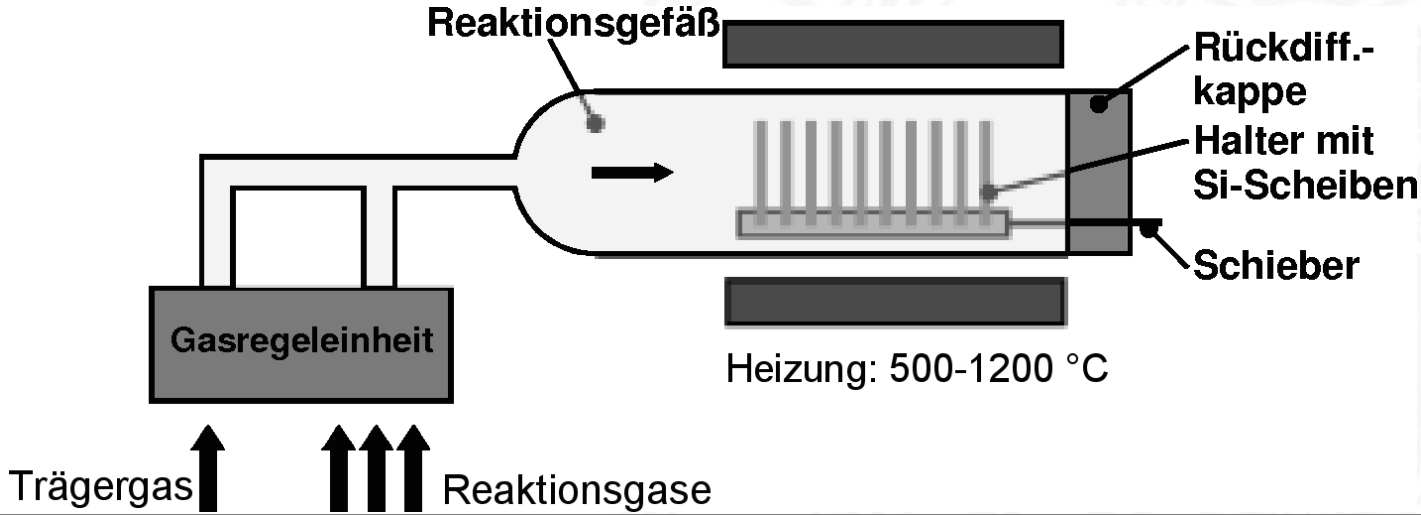
- als Homoepitaxie: Aufwachsen von schwach dotiertem Silizium (p^-, n^-) auf dem Wafer

- Flüssigphasenepitaxie / LPE (Liquid-Phase Epitaxy): Abscheidung aus einer Flüssigkeit, einer Schmelze

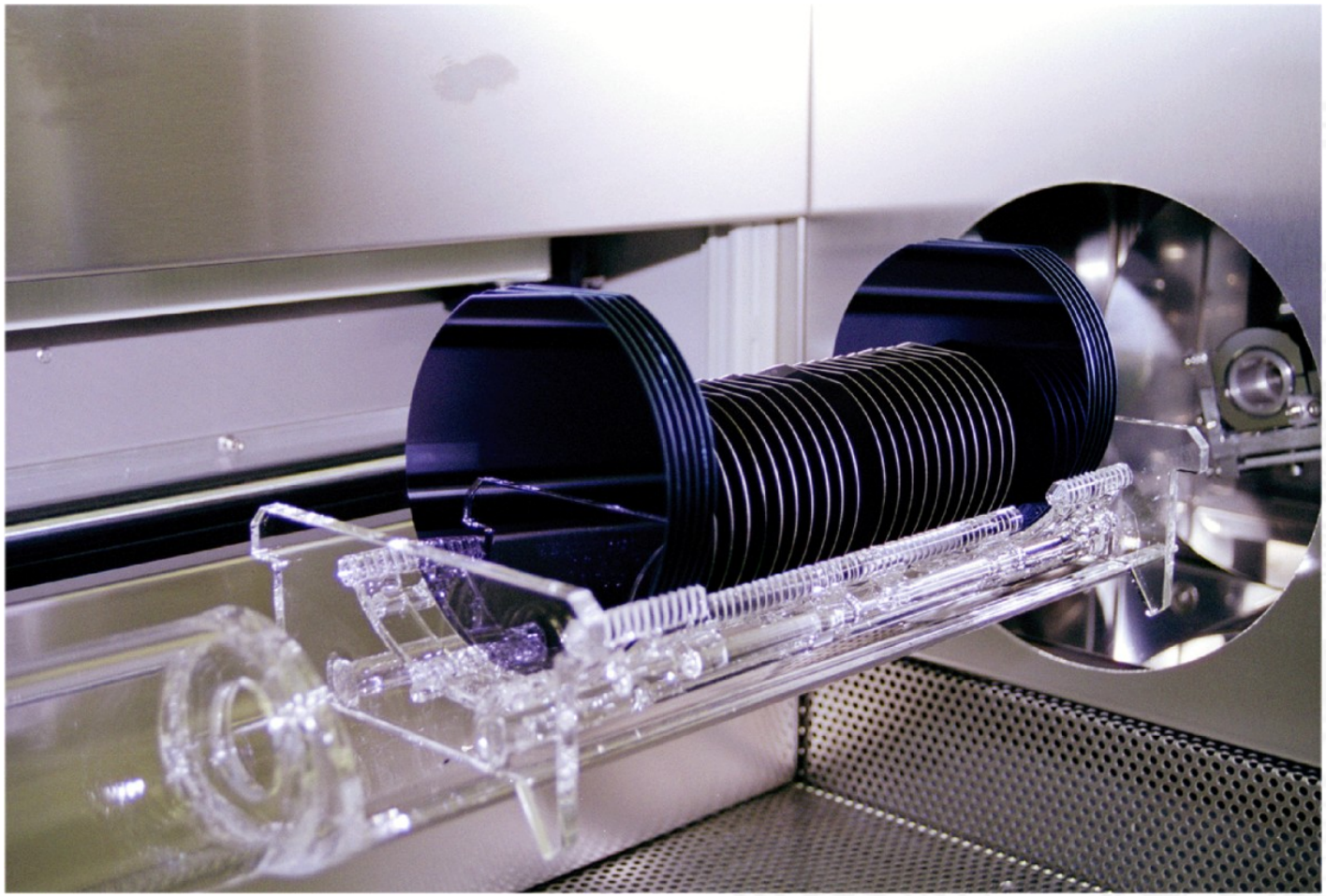
Epitaxie (cont.)

- Chemische Gasphasenabscheidung / CVD (Chemical Vapor Deposition): Chemische Reaktion mit, bzw. Abscheidung aus einem Gas

  Chipherstellung
  - Siliziumdioxid
  - polykristallines Si
  - Siliziumnitrid



Epitaxie (cont.)



Epitaxie (cont.)

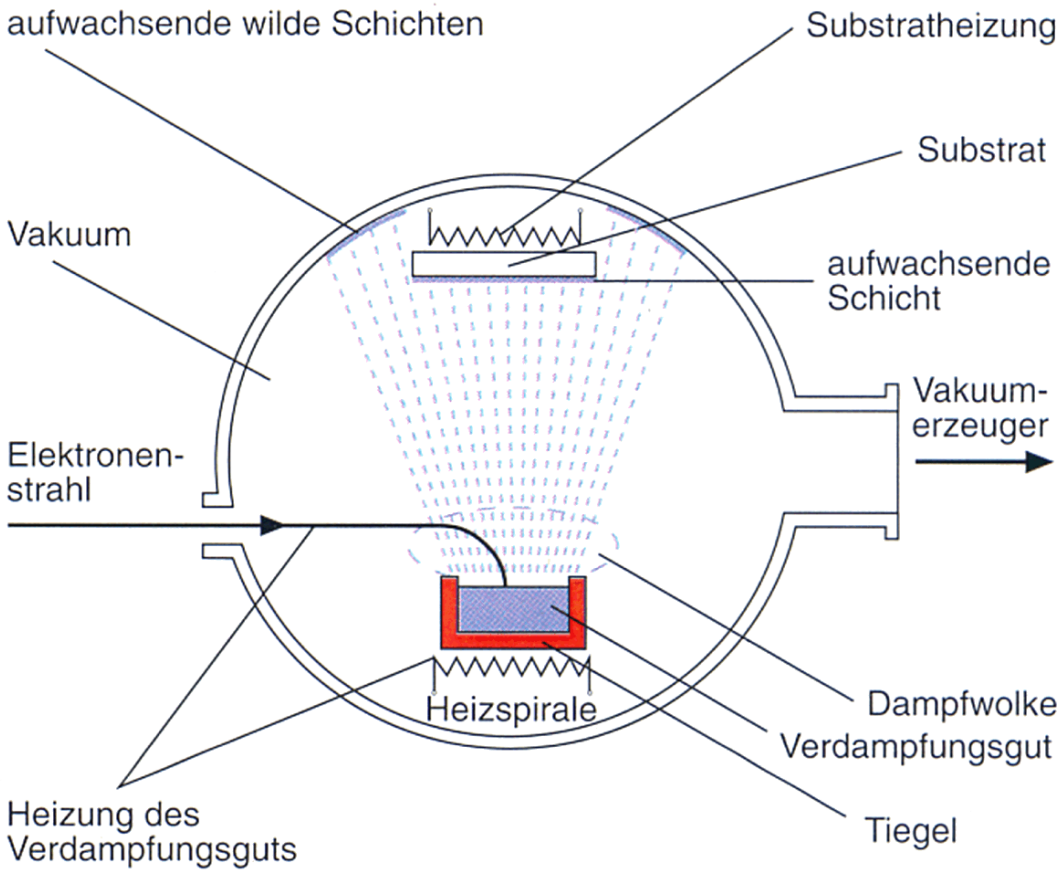
- Physikalische Gasphasenabscheidung / PVD (Physical Vapor Deposition): Kondensation von Materialdampf

  Chipherstellung
  - Metallisierung mit *Al*, *Cu*

  Oberbegriff für unterschiedlichste Verfahren je nach Art der Verdampfung, des Materialtransports...
  - thermisches Verdampfen
  - Elektronenstrahlverdampfen
  - Sputtern (Zerstäuben)
  - Molekularstrahlepitaxie

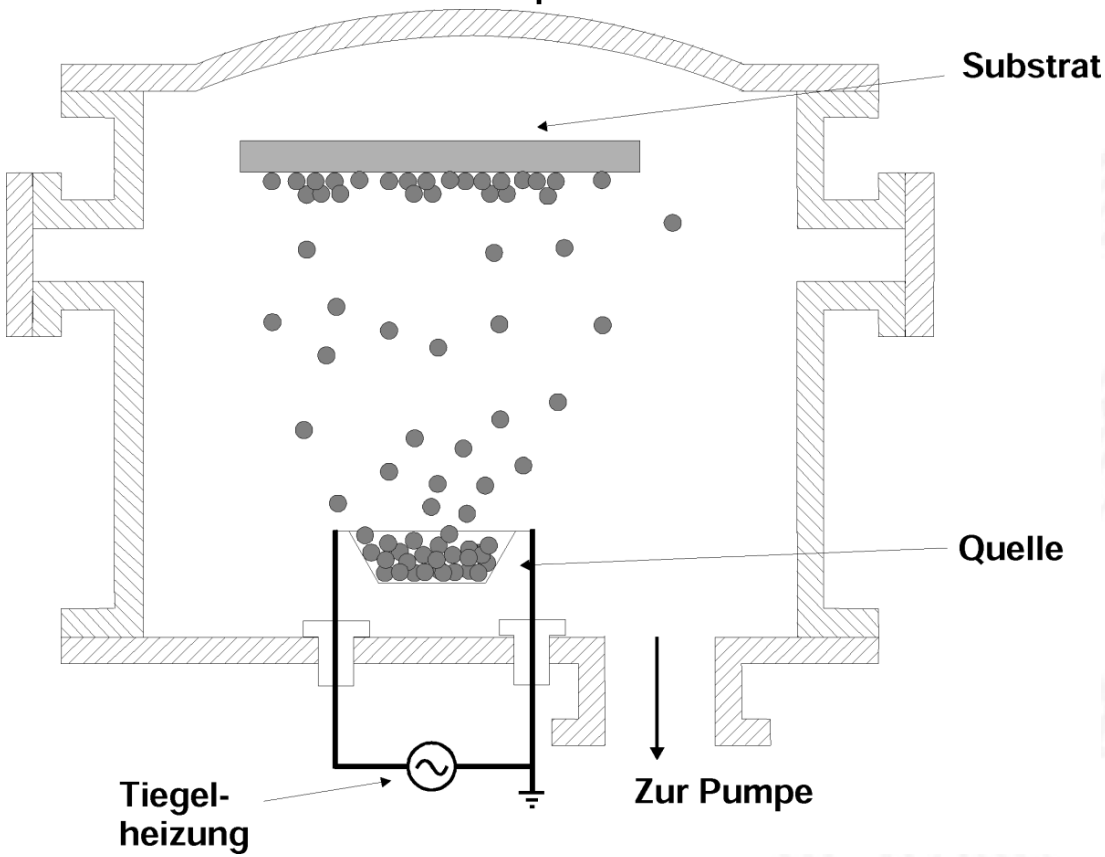
Epitaxie (cont.)

Elektronenstrahlverdampfen für die Metallisierung



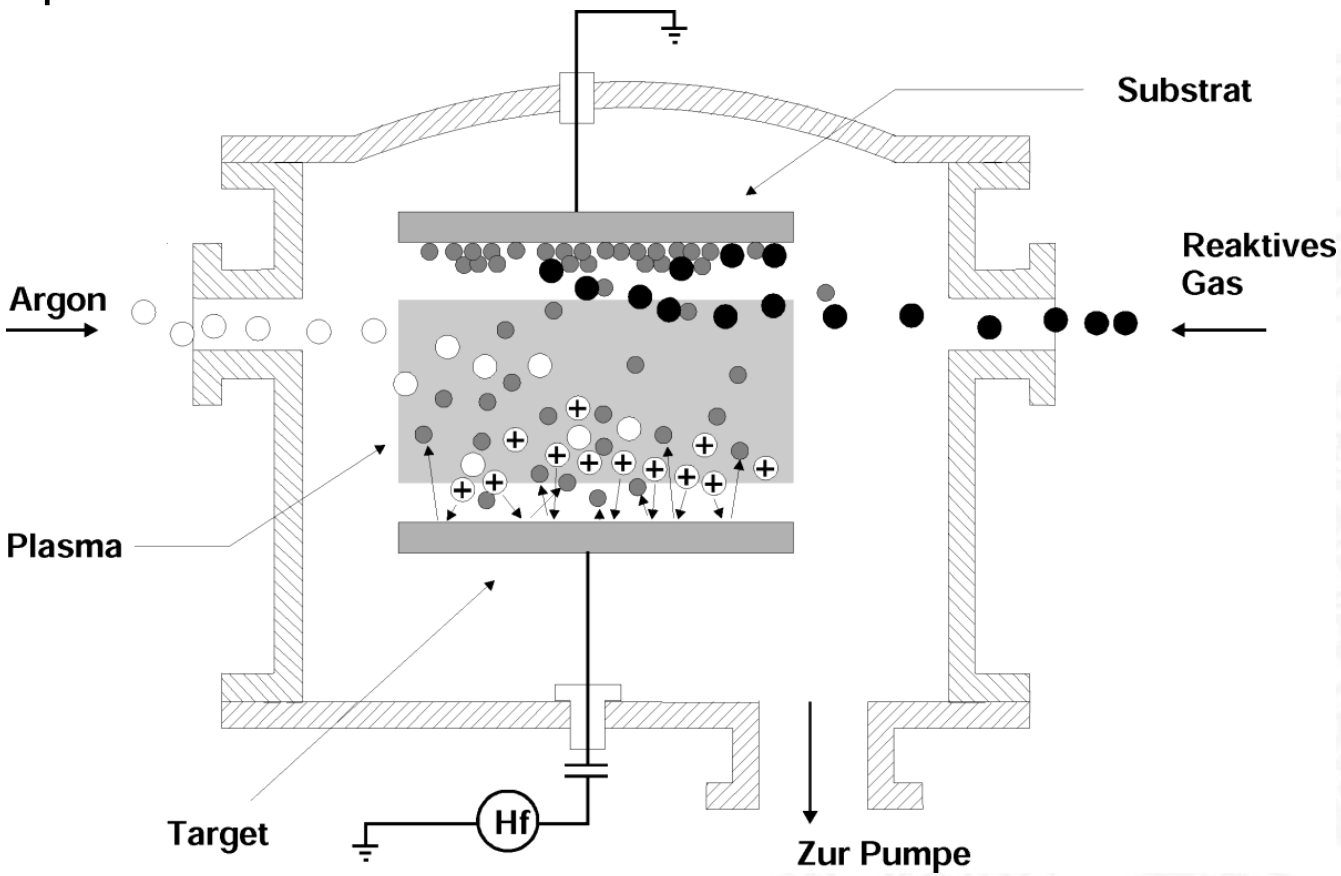
Epitaxie (cont.)

Thermisches Verdampfen



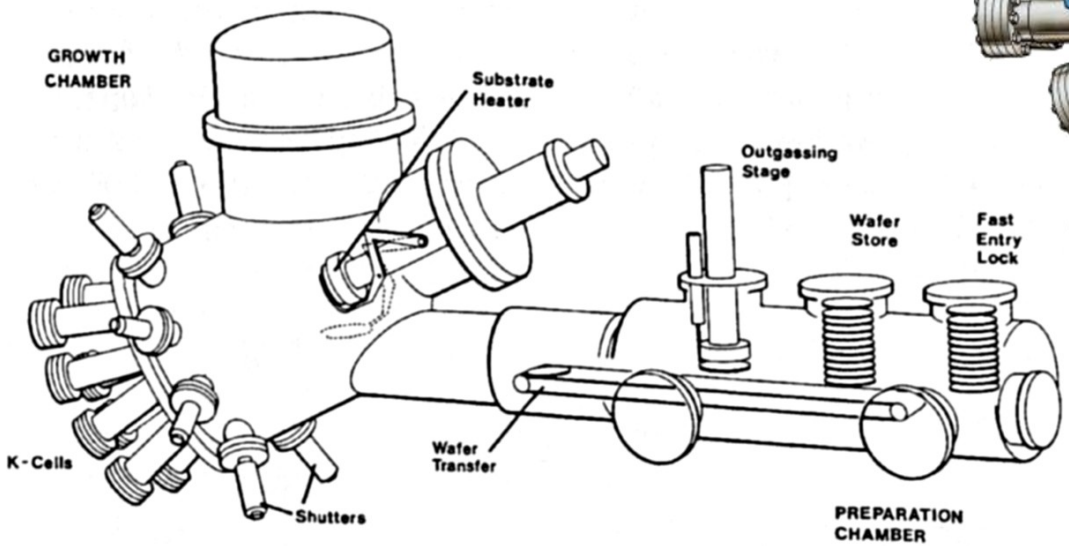
Epitaxie (cont.)

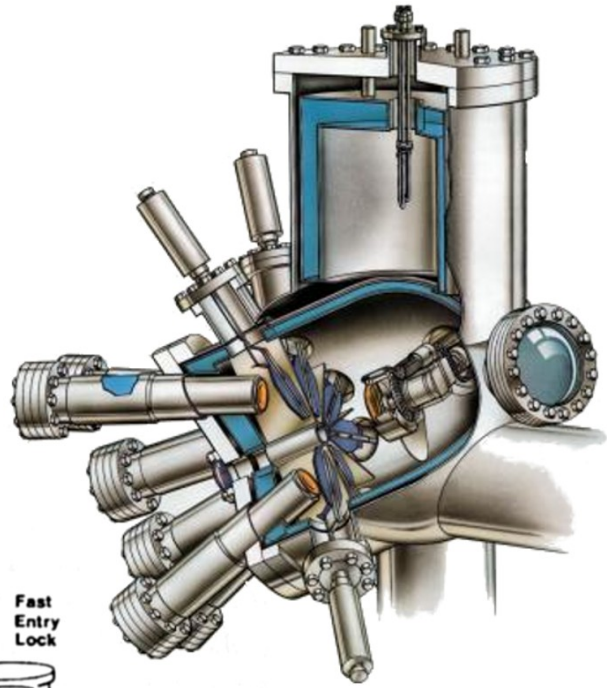
Sputtern



Epitaxie (cont.)

Molekularstrahlepitaxie (MBE Molecular Beam Epitaxie)





Oxidation

- Thermische Oxidation
  - Verfahren

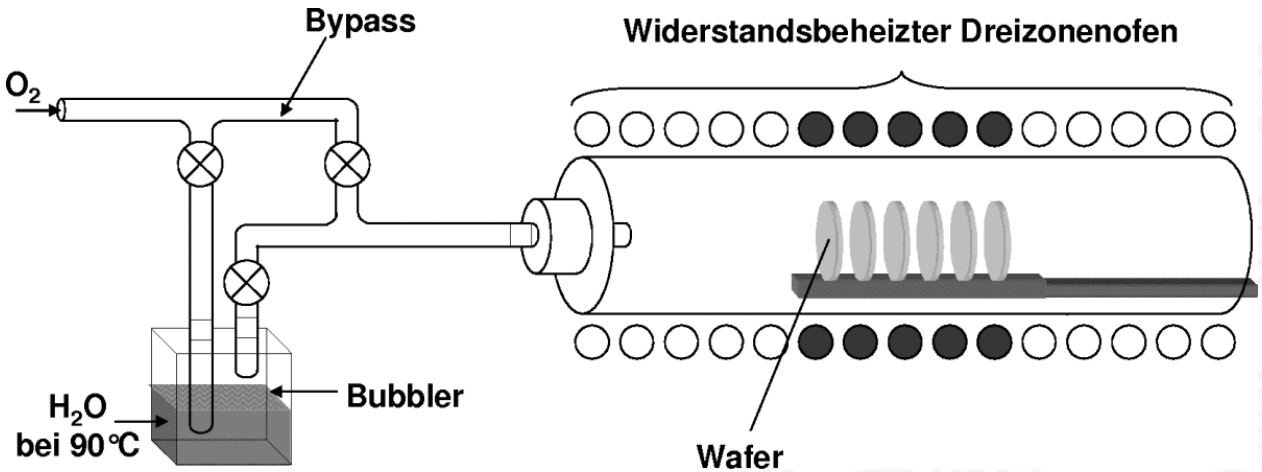
trocken	nass	
$Si + O_2 \rightarrow SiO_2$	$Si + 2H_2O \rightarrow SiO_2 + 2H_2$	
1000-1200	900-1100	Temperatur [°C]
40-210	240-900	Oxidation [nm/h]
hoch	niedrig	Dichte
wenige	vermehrt	Oxidladungen
$\Rightarrow$ Gateoxid	$\Rightarrow$ Feldoxid (Dickoxid)	

    H_2O_2Verbrennung: hohes Schichtwachstum, wenig Oxidladungen

  - verbraucht Silizium aus dem Substrat

Oxidation (cont.)

- Ofen für die thermische Oxidation

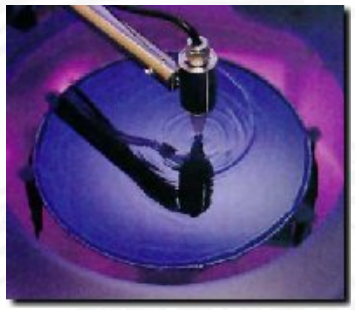


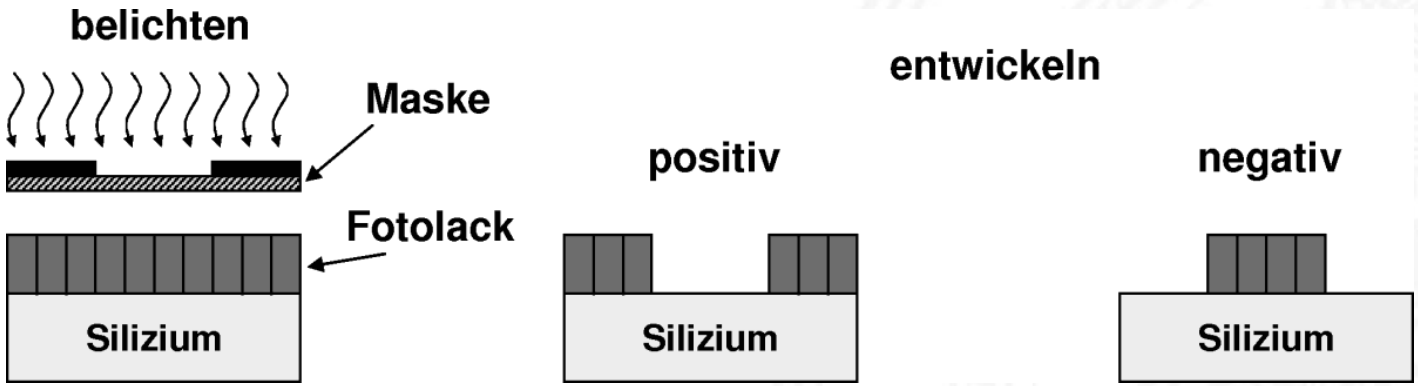
- Abscheideverfahren
  - Sputter-Beschichtung, Silan-Pyrolyse,...
  - dem Prozess muss Silizium zugeführt werden

Lithografie

Übertragung von Strukturen durch einen Belichtungsprozess

1. Lack Auftragen (Aufschleudern)
   - Positivlacke: hohe Auflösung ⇒ MOS
   - Negativlacke: robust, thermisch stabil





Lithografie (cont.)

2. „Belichten“
   - Maskenverfahren: 1:1 Belichtung, Step-Verfahren
     UV-Lichtquelle
   - Struktur direkt schreiben: Elektronen- / Ionenstrahl
   - andere Verfahren: Röntgenstrahl- / EUV-Lithografie
3. Entwickeln, Härten, Lack entfernen
   - je nach Lack verschiedene chemische Reaktionsschritte
   - Härtung durch Temperatur

. . . weitere Schritte des Planarprozess

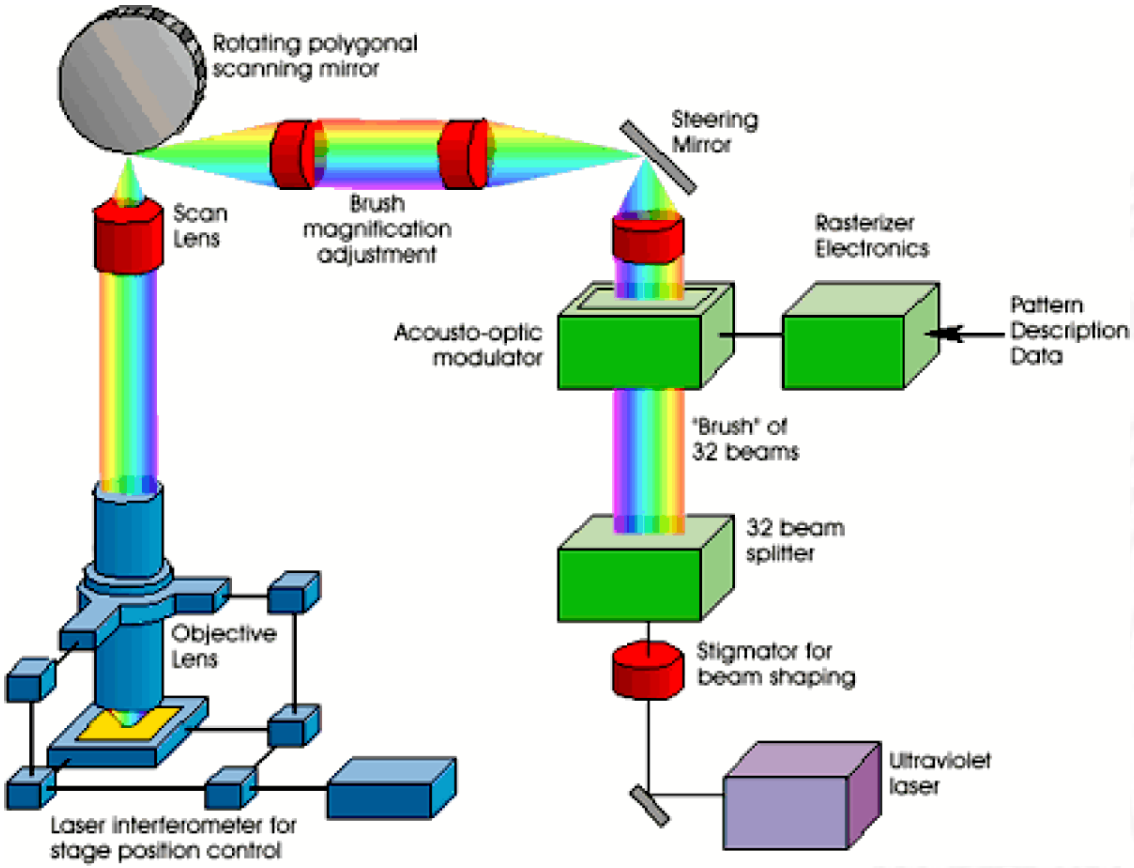
Lithografie (cont.)

Herstellung der Masken: fotochemischer Prozess, der Chromstrukturen auf einer Quarzplatte erzeugt

- Pattern-Generator belichtet Quarzplatte
  - Abbildung mechanischer Blenden
  - Licht aus Laser-Blitzquellen
    direkte Belichtung mit Elektronenstrahl
  - Ergebnis: Maske eines Chips im Maßstab 5:1 – „Reticle“
- Step-and-Repeat Verfahren erzeugen die Wafermaske
  - wiederholte Abbildung und Verkleinerung
  - Ergebnis: Maske eines Wafers

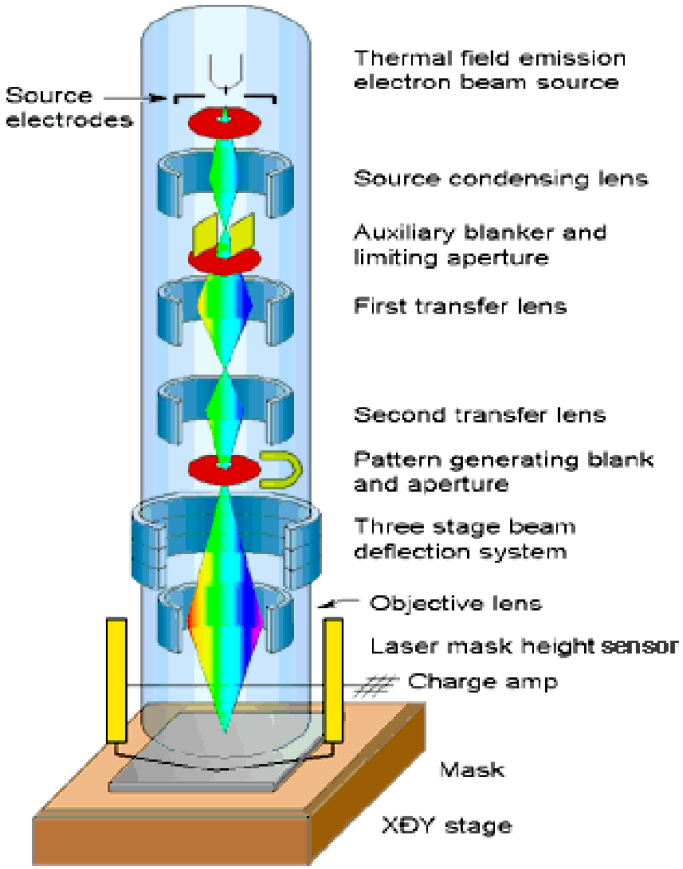
Lithografie (cont.)

Laser Patterngenerator



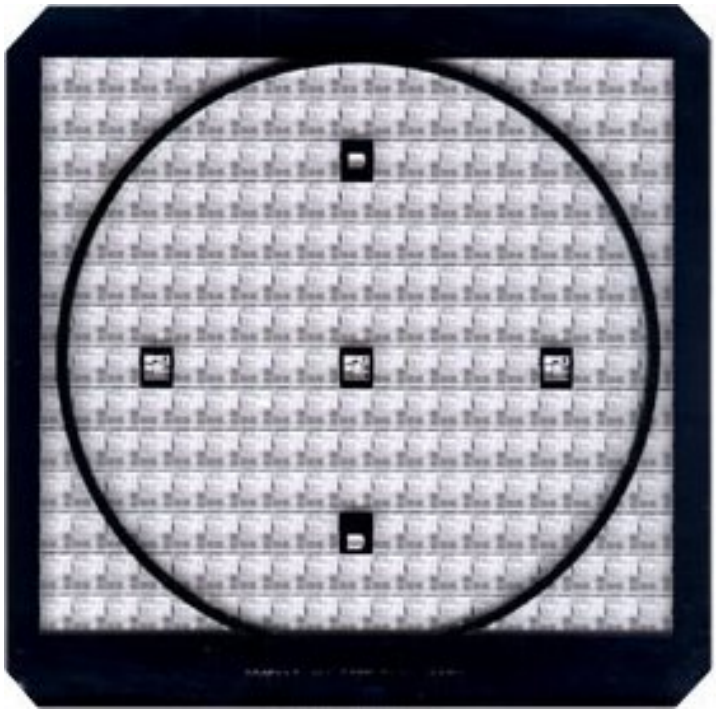
Lithografie (cont.)

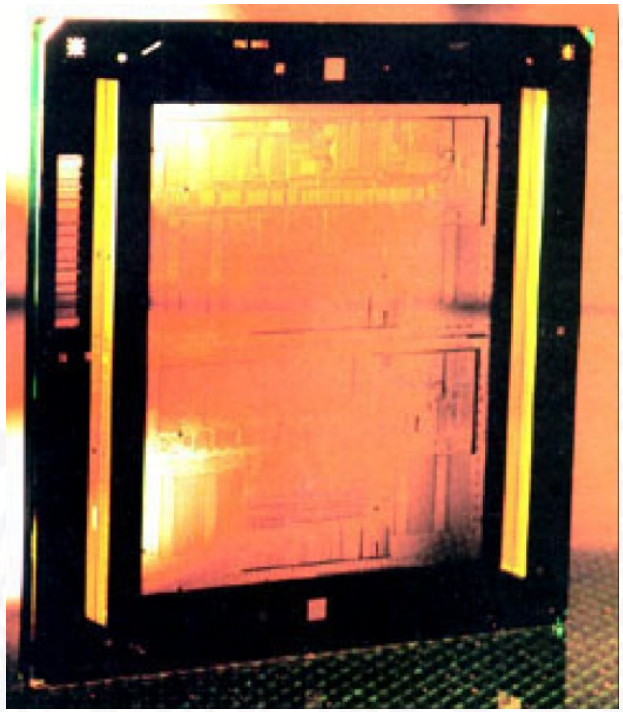
Elektronenstrahlschreiber



Lithografie (cont.)

Maske: Substrat mit Chromstruktur

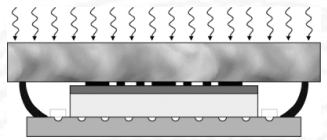


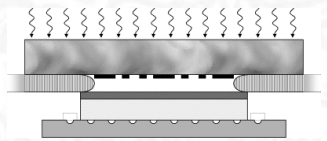


Lithografie (cont.)

optische Lithografieverfahren

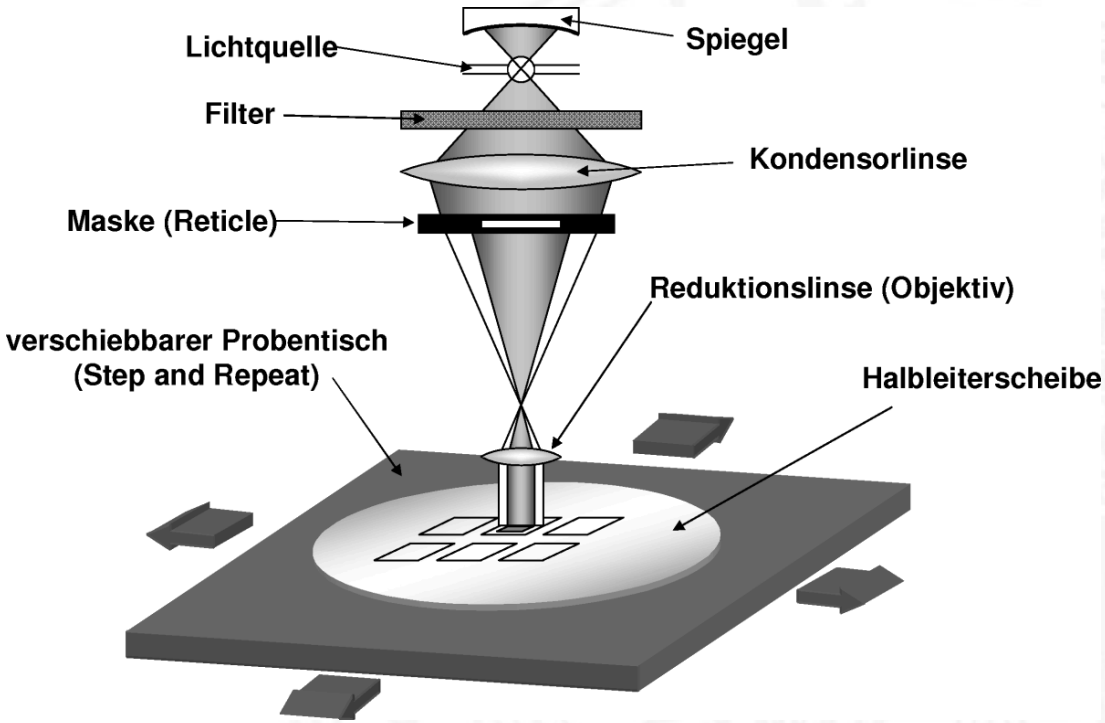
- 1:1 Belichtung: Maskeninhalt entspricht Wafer
  - \+ kostengünstig, hoher Durchsatz
  - – Justierprobleme bei Scheibenverzug
  - Vakuumkontakt
    - Auflösung: $< 0{,}6\ \mu$m
    - Defektdichte: hoch, mechanische Beschädigung
  - Abstandsbelichtung
    - Abstand: 10-30 μm
    - Auflösung: 2-3 μm
    - Defektdichte: gering
  - Projektionsbelichtung: direkte Abbildung oder über Wafer-Scan
    - Auflösung: $< 0{,}8\ \mu$m
    - Defektdichte: gering





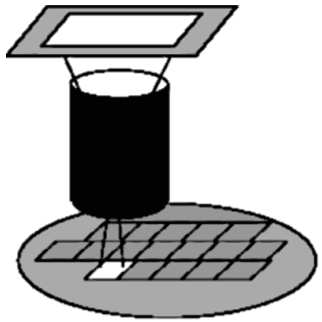
Lithografie (cont.)

- Step-Verfahren: Maskeninhalt entspricht Chip
  - \+ Auflösung, Justierung
  - – Durchsatz

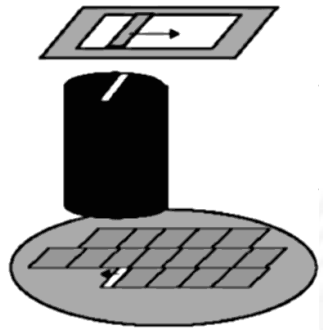


Lithografie (cont.)

- Step-and-Repeat



Step-Scan



- Abbildungsmaßstab: 1:1 bis 1:5
  - Auflösung: 0,6 μm bis $<$ 0,1 μm (1:5)
  - Defektdichte: sehr gering
- extrem aufwändige Anlagen
  - Positioniergenauigkeit
  - Temperaturausgleich
  - Kosten: mehrere Mio.

Lithografie (cont.)

Waferstepper

ASML

Lithografie (cont.)

Probleme der optischen Lithografie

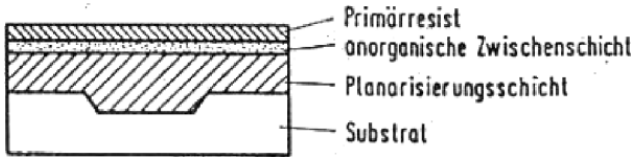
- Auflösung abhängig von der Wellenlänge
  $a = \frac{k_1 \lambda}{NA}$ λ : Wellenlänge
  k_1 : f(Kohärenzgrad des Lichts, optische Eigenschaften)
  NA : numerische Apertur
  $a \approx 0,5 \mu m \quad (\lambda = 400 nm)$
- Tiefenschärfe abhängig von der Wellenlänge
  $DOF = \pm \frac{k_2 \lambda}{NA^2}$
  $DOF \approx \pm 800 nm \quad (\lambda = 400 nm)$

$\Rightarrow$ kleinere Wellenlänge = höhere Auflösung
geringere Tiefenschärfe

- Beugungs- / Reflexionseffekte

Lithografie (cont.)

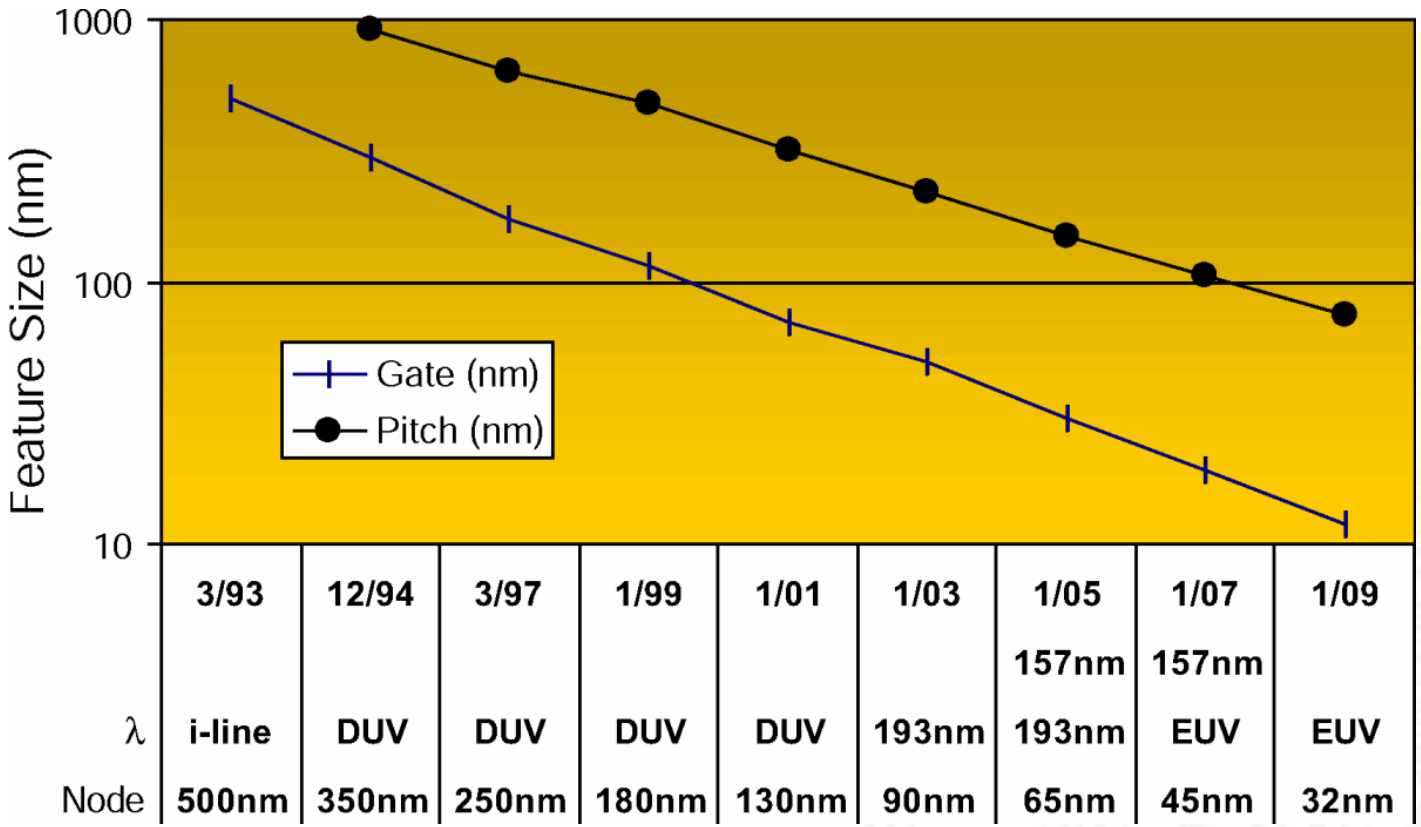
- Optimierungen
  - kurzwelliges Licht, bis 193 nm
  - Phaseneffekte ausnutzen: „Phase-Shifting“ Masken
    $\Rightarrow$ Strukturgrößen kleiner als Lichtwellenlänge
  - nichtaxiale Belichtung
  - Mehrlagen-Resist: mehrlagiger Aufbau der Fotoschicht
    - „Glättung“ der Oberfläche
    - reflexionsvermindernde Schichten



$$a = \frac{k_1 \lambda}{NA}$$

- Mehrfachbelichtung $k_1 < 0,25$
- Immersionslithografie $NA > 1$

Lithografie (cont.)

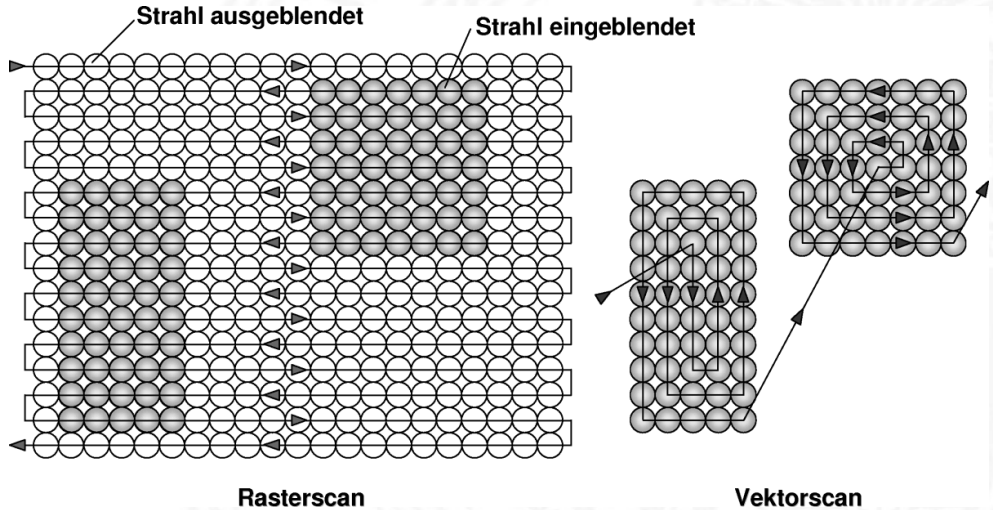


Intel: Lithography Roadmap (2002)

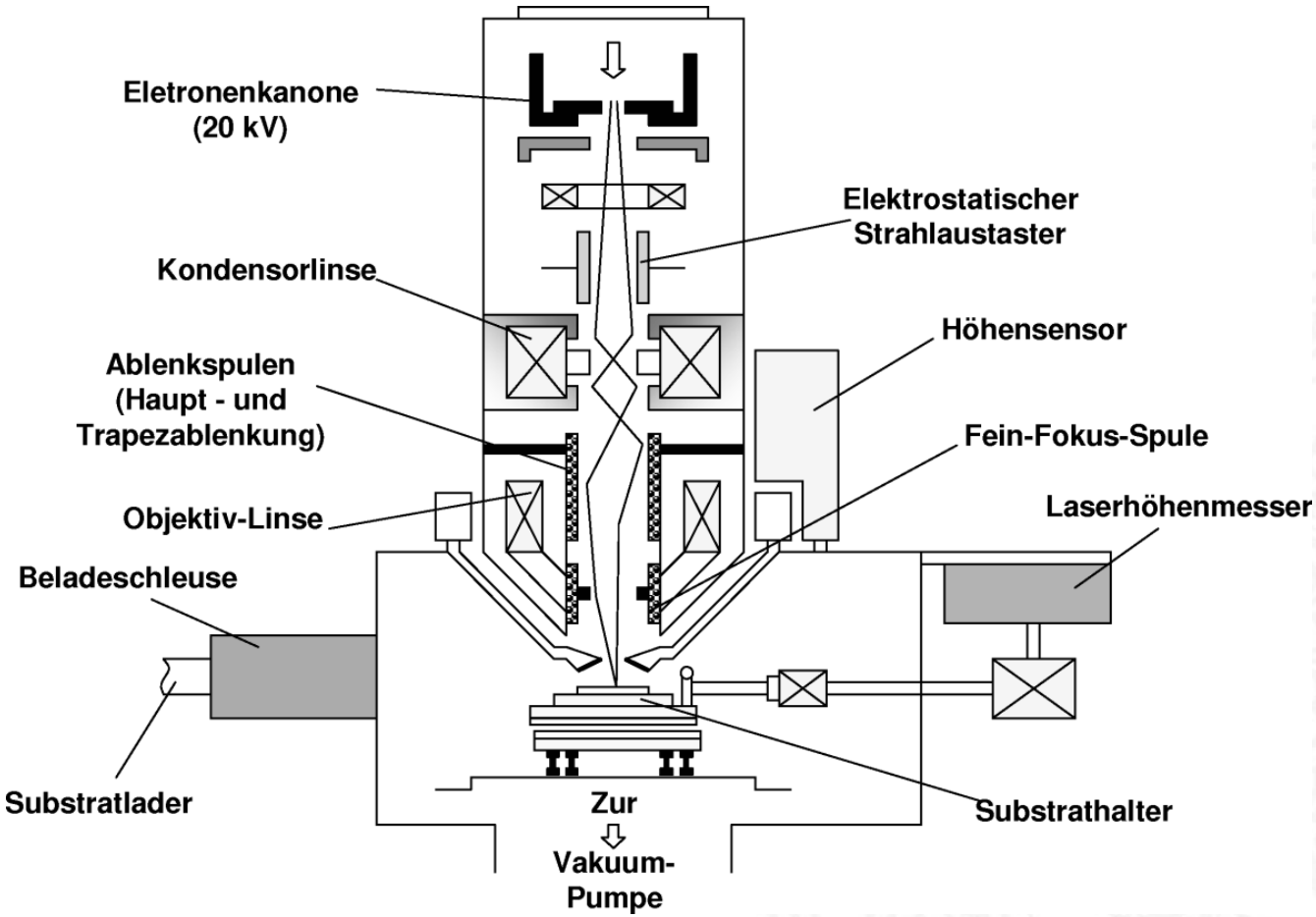
Lithografie (cont.)

weitere Lithografieverfahren

- Röntgenstrahlbelichtung
  - \+ Auflösung
  - – Strahlungsquelle (Synchrotronstrahlung), Maskenmaterial (Gold)
- Elektronenstrahlschreiben
  - für Kleinstserien: Prototypenfertigung
  - \+ Auflösung, keine Masken
  - – Durchsatz

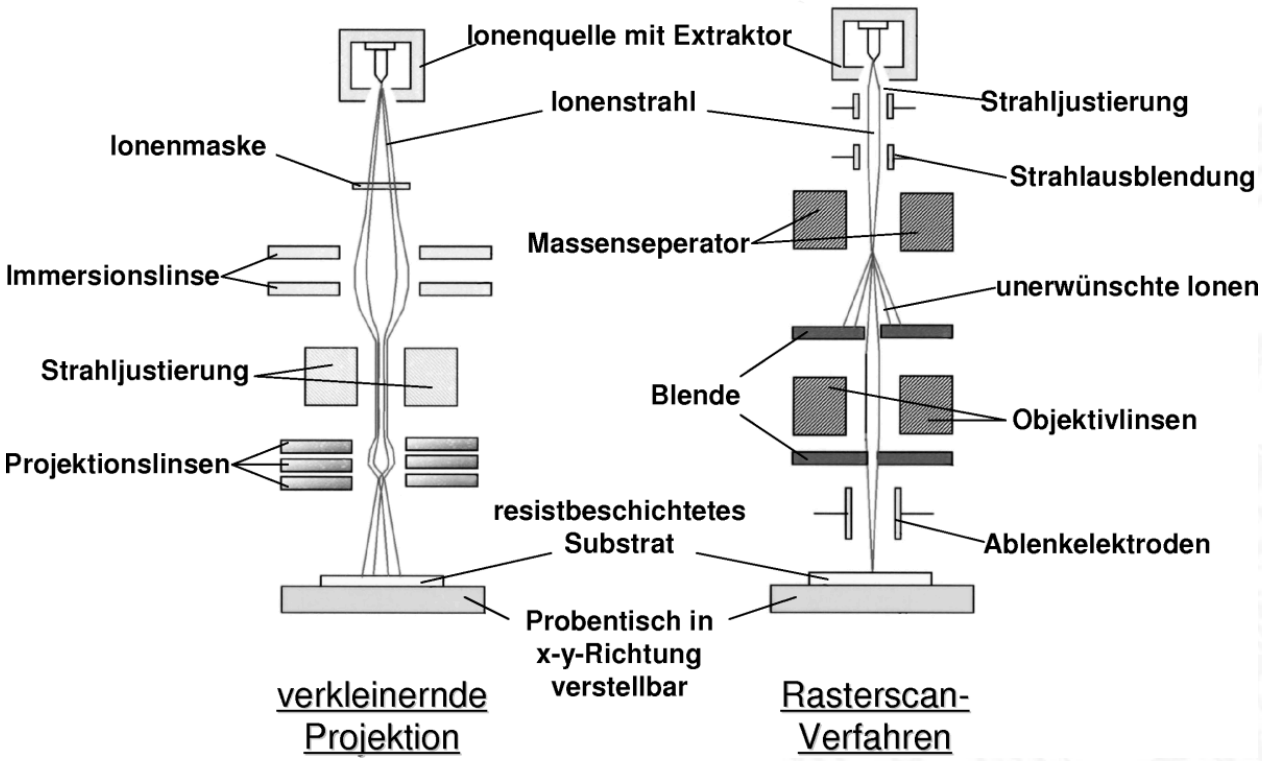


Lithografie (cont.)



Lithografie (cont.)

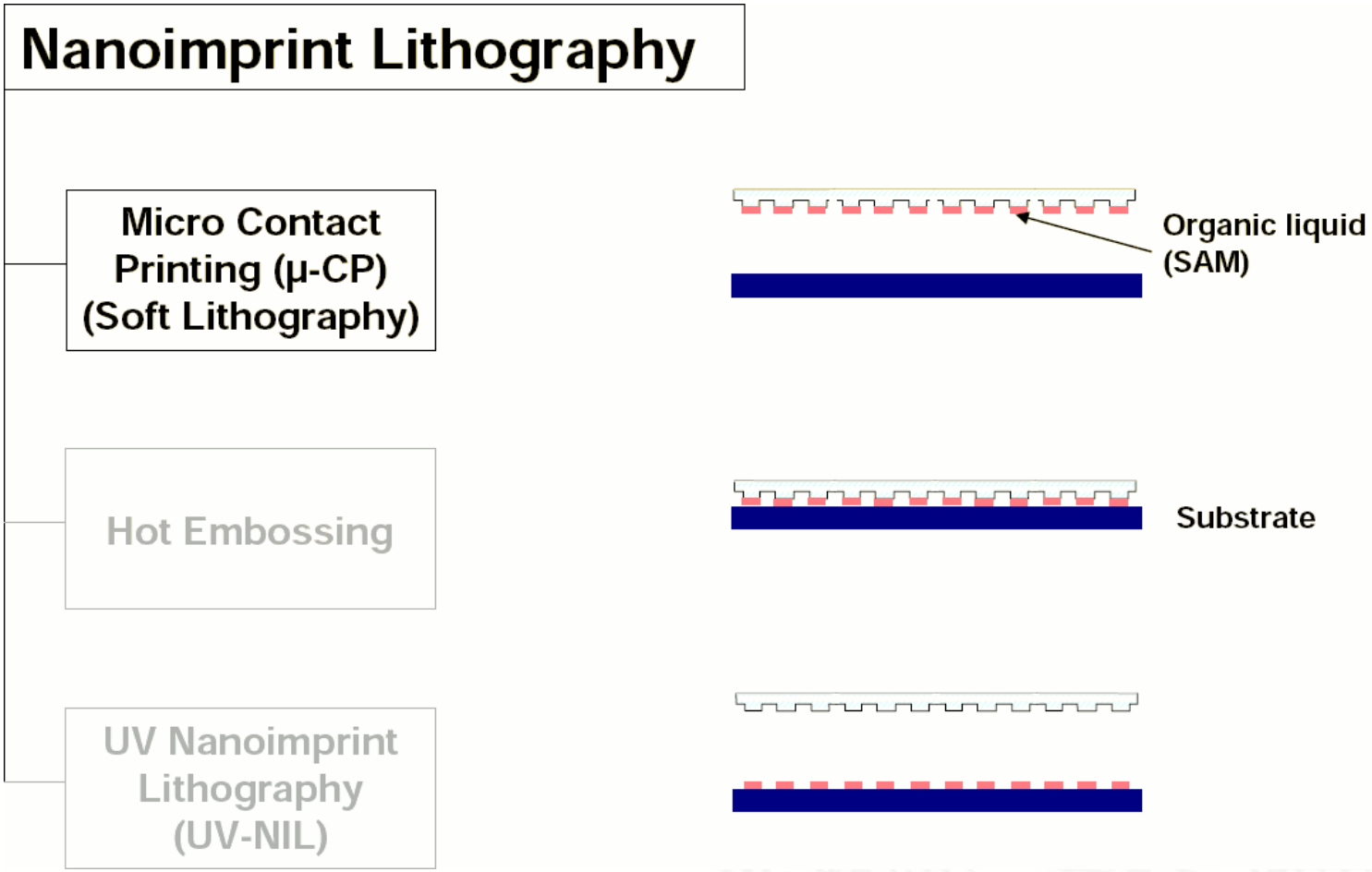
- Ionenstrahlbelichtung
  - \+ Auflösung, keine Masken
  - – Durchsatz



Lithografie (cont.)

- Interferenz Lithografie
  - Interferenzeffekte von UV-Laserstrahlen
  - Auflösungen $< 50\,\text{nm}$
- Nanoimprint Lithografie (NIL)
  - Auflösungen $< 10\,\text{nm}$
  - verschiedene Verfahrensweisen: thermoplastisch, Kontaktübertragung, elektrochemisch, Photo-/UV-basiert
  - verschiedene Materialien: Quarz, Si, Ni, PDMS (organ. Polymer)

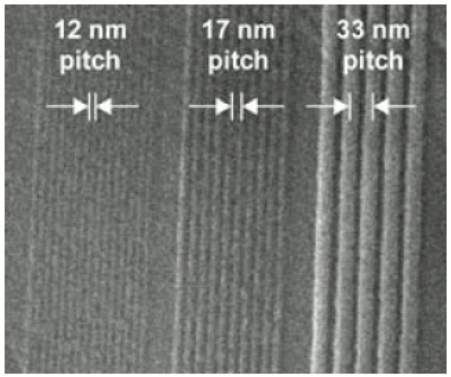
Lithografie (cont.)



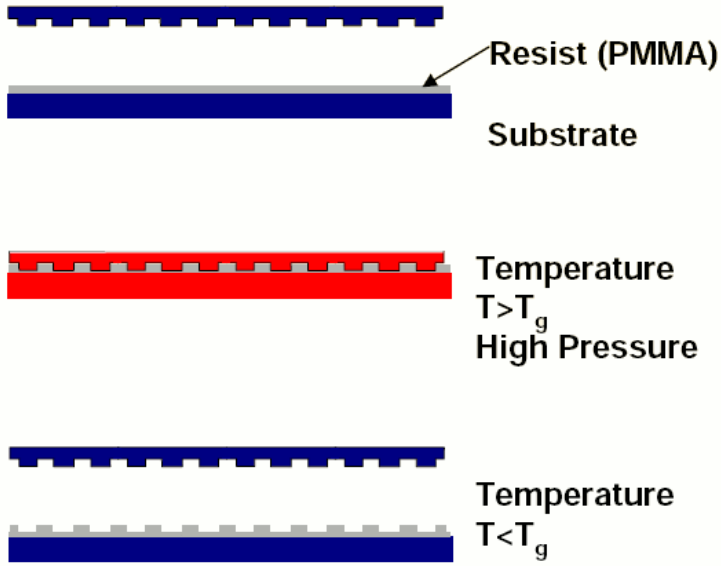
Lithografie (cont.)

Nanoimprint Lithography

- Micro Contact Printing (µ-CP) (Soft Lithography)
- **Hot Embossing**
- UV Nanoimprint Lithography (UV-NIL)



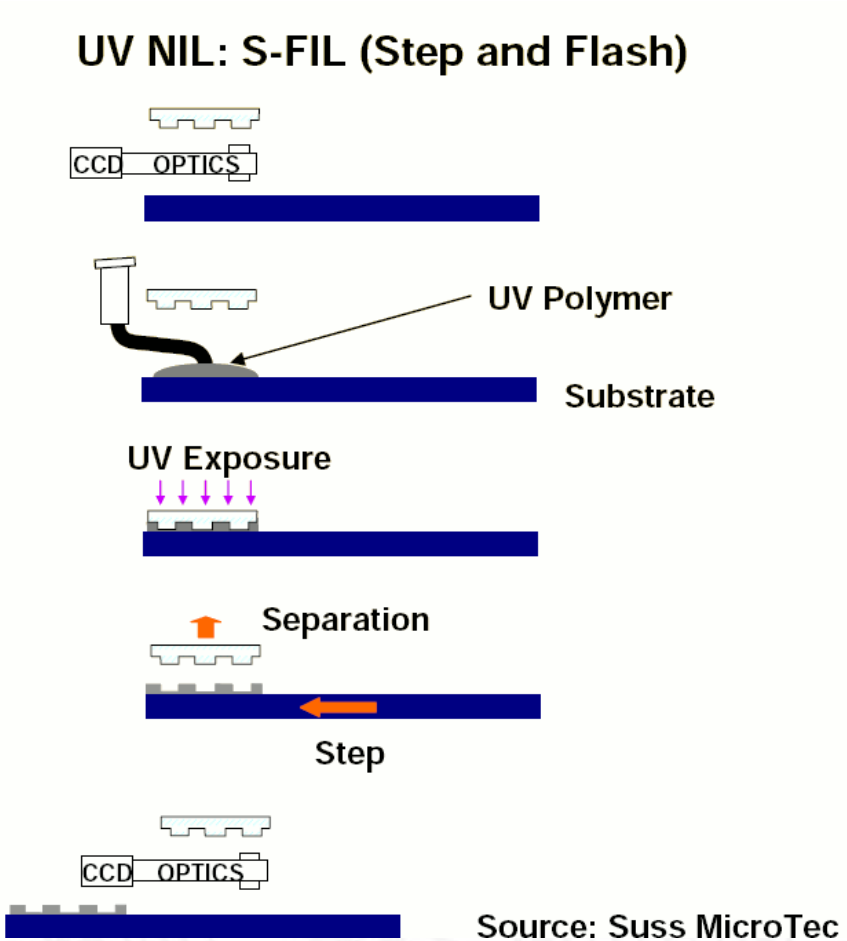
Source: Nanonex



Lithografie (cont.)

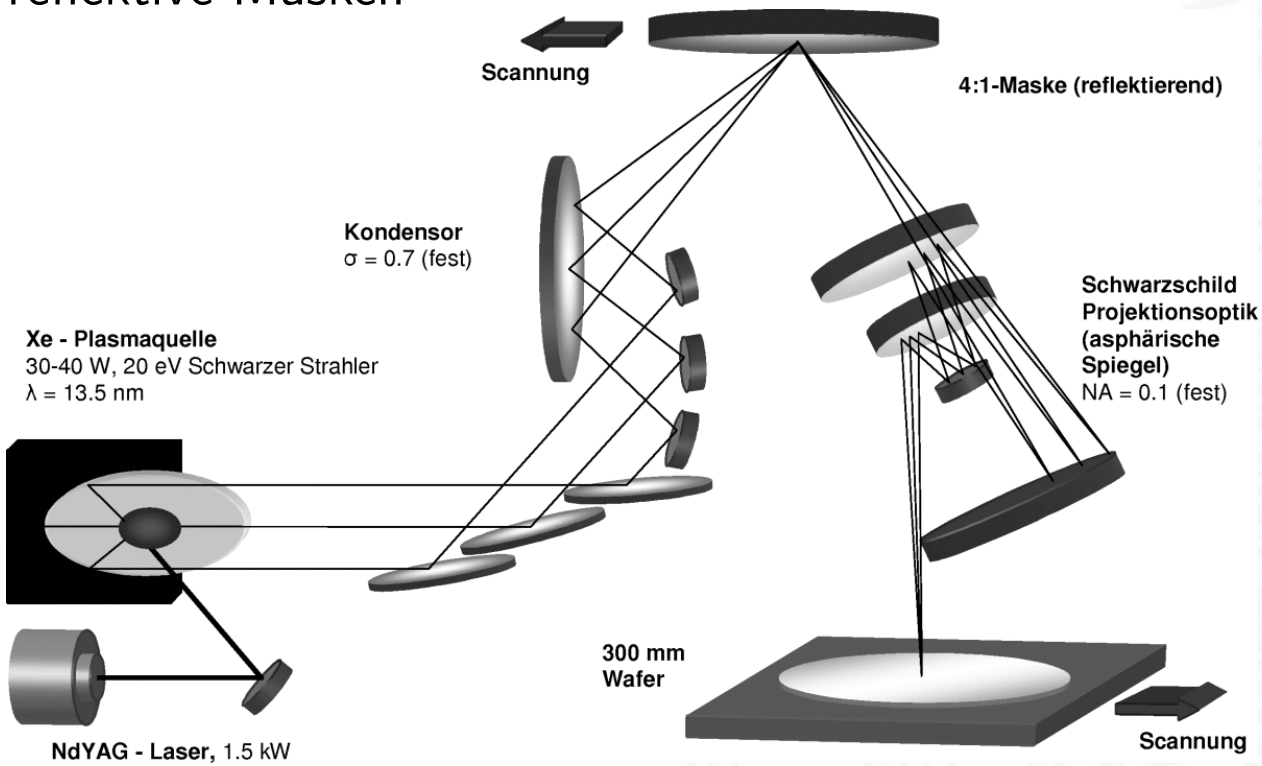
Nanoimprint Lithography

- Micro Contact Printing (µ-CP) (Soft Lithography)
- Hot Embossing
- **UV Nanoimprint Lithography (UV-NIL)**

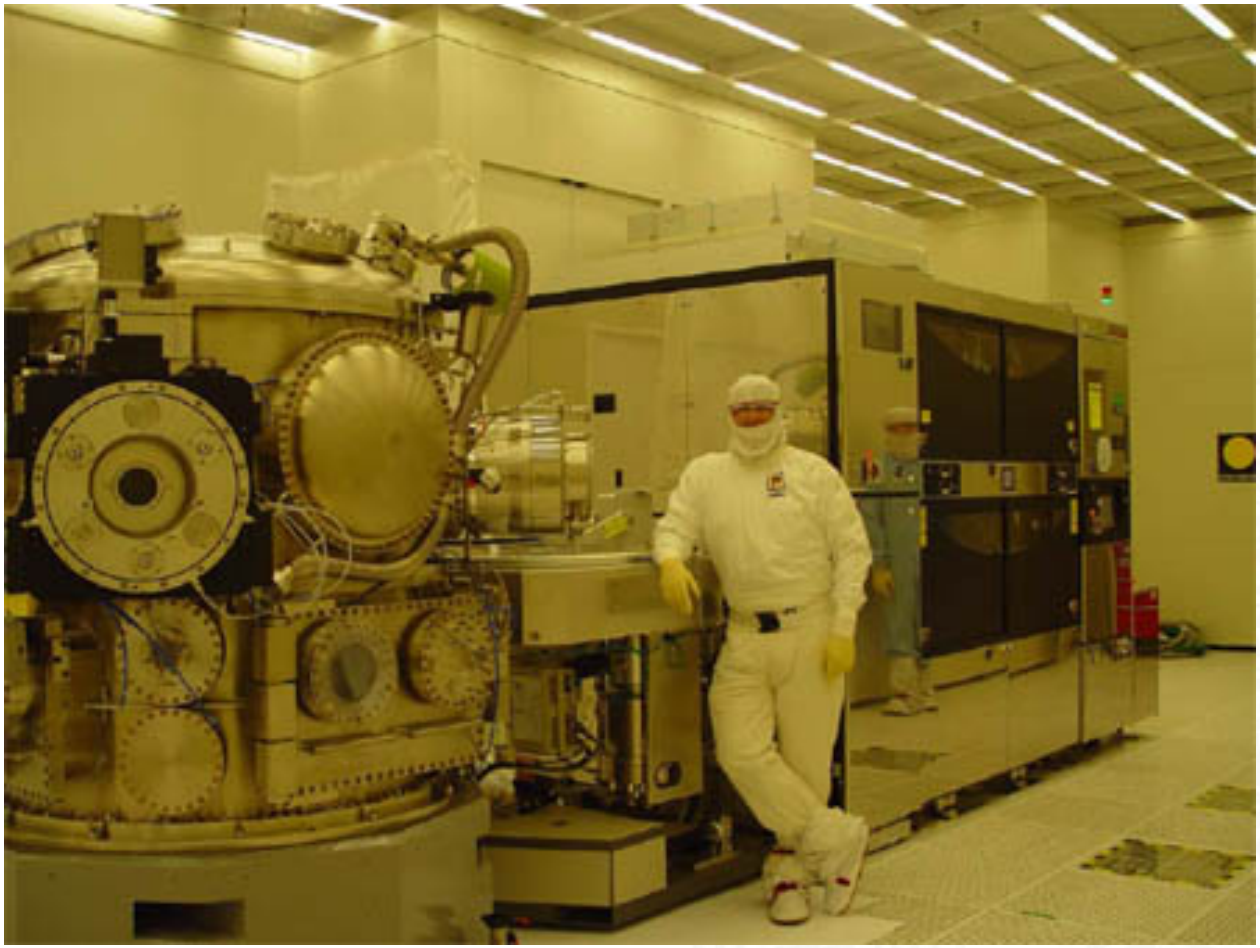


Lithografie (cont.)

- Extrem UV Scanner (EUV)
  - reflektive Masken



Lithografie (cont.)



Intel EUV-Testanlage

Lithografie (cont.)

- zukünftige Technologie, erste Testanlagen in Betrieb (Planung Intel: 32nm in 2009)
- vielfältige Probleme
  - Leistung der EUV-Quelle
  - Maskenmaterial
  - Genauigkeiten, thermische Ausdehnungen...

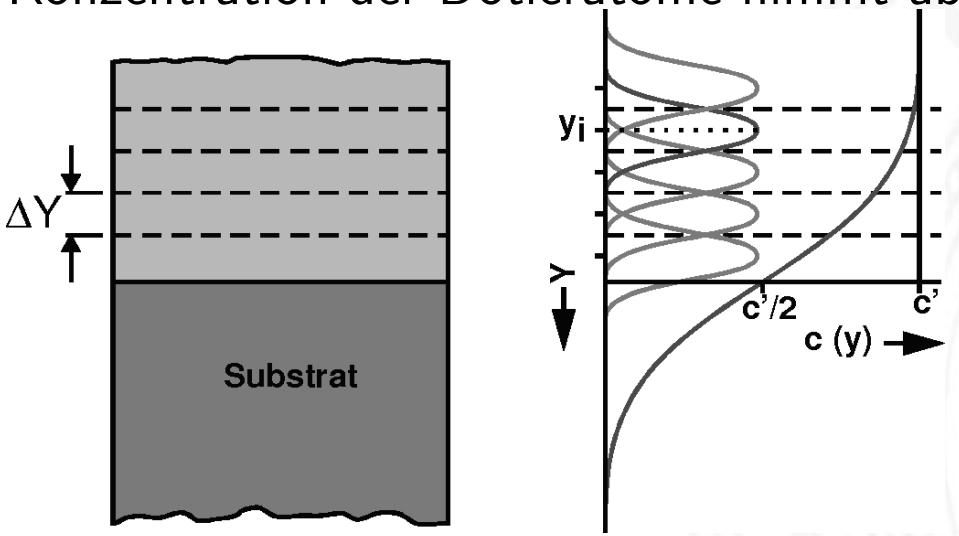
  Beispiel: Oberflächen

Dotierung

Fremdatome in den Siliziumkristall einbringen

- Diffusion
  - Diffusionsofen ähnlich CVS-Reaktor
  - gaußförmiges Dotierungsprofil

    Konzentration der Dotieratome nimmt ab

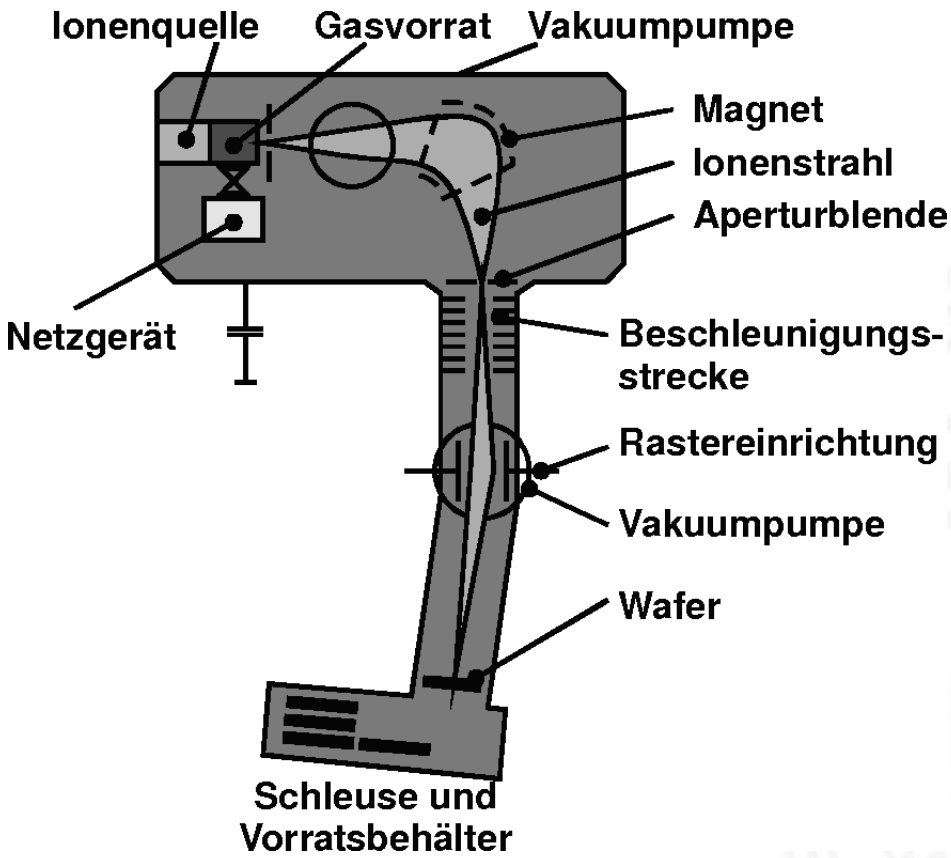


Dotierung (cont.)

- Ionenimplantation
  - „Beschuss“ mit Ionen
  - Beschleunigung der Ionen im elektrischen Feld
  - Über die Energie der Ionen kann die Eindringtiefe sehr genau eingestellt werden
  - „Temperung“ notwendig: Erhitzen des Einkristalls zur Neuorganisation des Kristallgitters

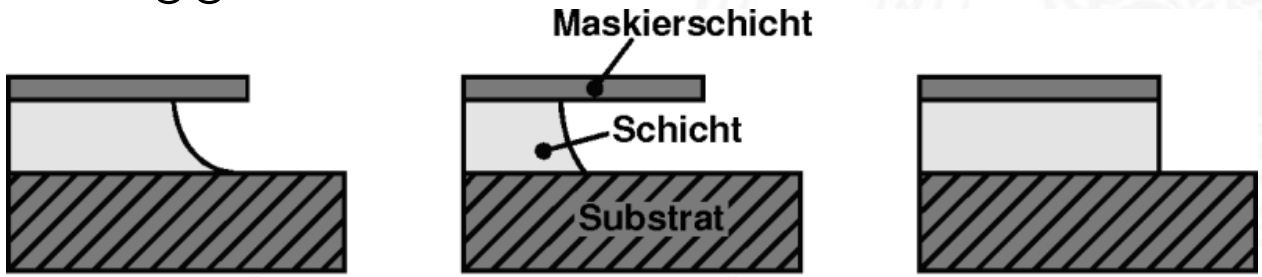
Dotierung (cont.)

Ionenimplantation



Ätztechnik

- Verfahren
  - nasschemisch: mit flüssiger Säure
  - Plasma- / Barrelätzen: reaktive Radikale durch Gasentladung
  - Reaktives Ionenätzen
  - Sputterätzen: rein physikalisches Verfahren
- Ätzprofile: $A = 1 - \frac{v_l}{v_v}$ lateral/vertikal
  - isotrop: gleichmäßig in alle Richtungen $\Rightarrow$ Mikroelektronik
  - anisotrop: mit Vorzugsrichtung $\Rightarrow$ Mikromechanik
  - Abhängig von: Verfahren, Ätzmittel, Kristallorientierung...



Ätztechnik (cont.)

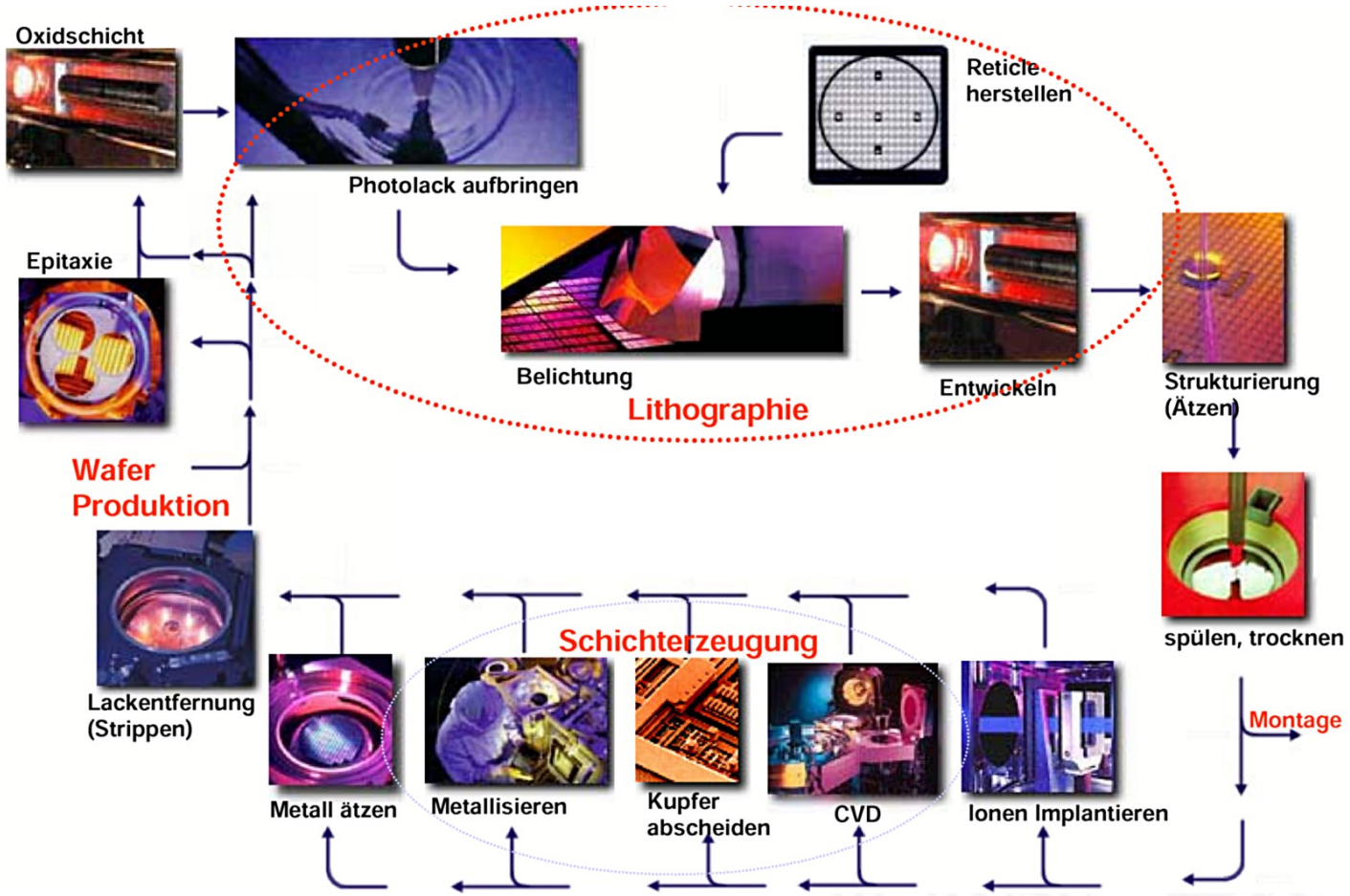
Ätzlösung für dünne Schichten (Mikroelektronik)

Material	**Ätzlösung**	**Ätzrate [nm/min]**
SiO_2	*gepufferte Flusssäure*: 113g NH_4F, 28ml HF, 170ml H_2O	100 – 250 bei 25 °C
SiO_2, PSG	1ml gepufferte Flusssäure 7ml H_2O	80
Si_3N_4	*heisse Phosphorsäure*: H_3PO_4	10 bei 180 °C
Poly-Si	26ml HNO_3, 1ml HF, 33ml CH_3COOH	150
Al	4ml H_3PO, 4ml CH_3COOH, 1ml HNO_3, 1ml H_2O	35
Au	*KI-I_2-Ätze*: 4g KI, 1g I_2, 40ml H_2O	500 – 1000
Cr	1ml (1g NaOH, 2ml H_2O) 3ml (1g $K_3Fe(CN)_6$, 3ml H_2O)	25 – 100

Planarprozess

- Der zentrale Ablauf bei der Herstellung von Mikroelektronik
- Ermöglicht die gleichzeitige Fertigung aller Komponenten auf dem Wafer
- Schritte
  1. Vorbereiten / Beschichten des Wafers:
     Oxidation, CVD, Aufdampfen, Sputtern...
  2. Strukturieren durch Lithografie
  3. Übertragen der Strukturen durch Ätzprozesse
  4. Modifikation des Materials: Dotierung, Oxidation
  5. Vorbereitung für die nächsten Prozessschritte...

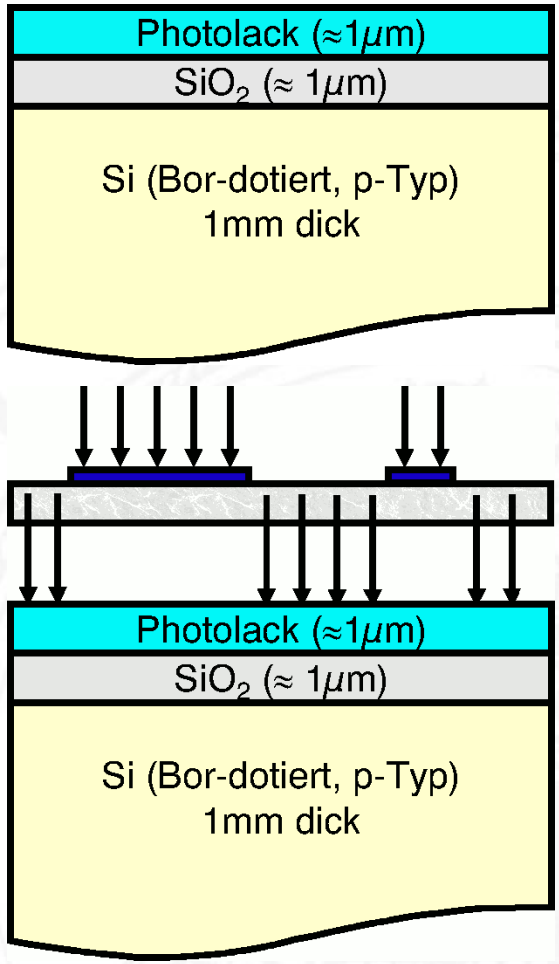
Planarprozess (cont.)



Planarprozess (cont.)

n^+-Dotierung eines n-Kanal Transistors

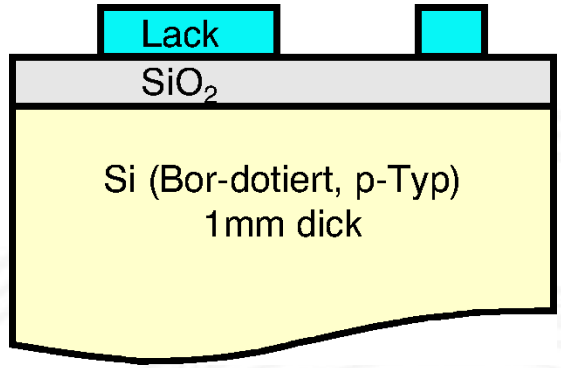
1. p-dotiertes Substrat oxidieren
2. Fotolack aufschleudern



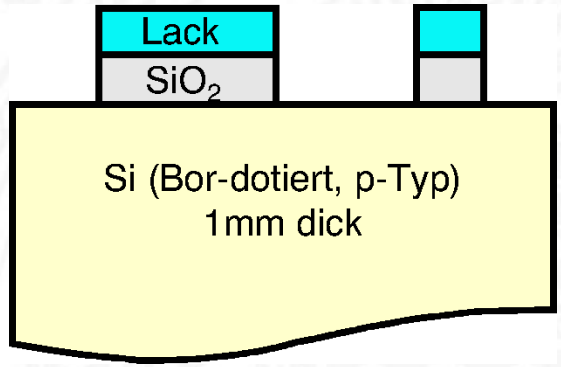
3. Fotolithografie
   Struktur der Maske übertragen

Planarprozess (cont.)

4. Entwickeln
5. Fotolack entfernen

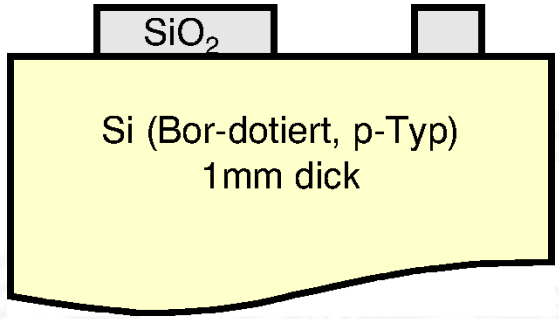


6. Ätzen des nicht geschützten SiO_2 z.B. mit gepufferter Flusssäure

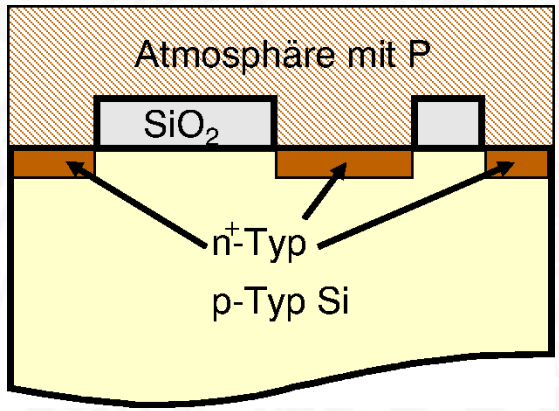


Planarprozess (cont.)

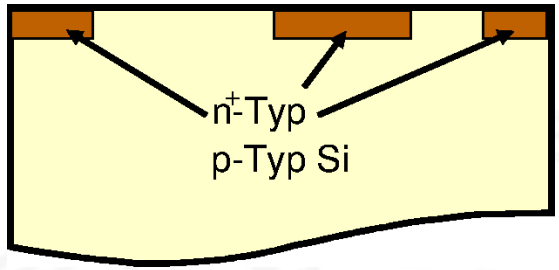
7. Entfernen des Lacks
   Die Struktur ist in SiO_2 übertragen



8. Dotierungsschritt: Diffusion von Phosphor in nicht durch SiO_2 abgedecktes Substrat



Planarprozess (cont.)



8. Maskieroxid entfernen

$\Rightarrow$ fertig...

Je nach Art des MOS-Prozesses und der verwendeten Technologie wird der oben skizzierte Planar-Prozess bis zu 20 mal durchlaufen:

- für Dotierungen
- für Metallisierungen
- für Kontakte

Schaltungstechniken / Logikfamilien

Abhängig von der Schaltungstechnik werden unterschiedliche Logikfamilien unterschieden.

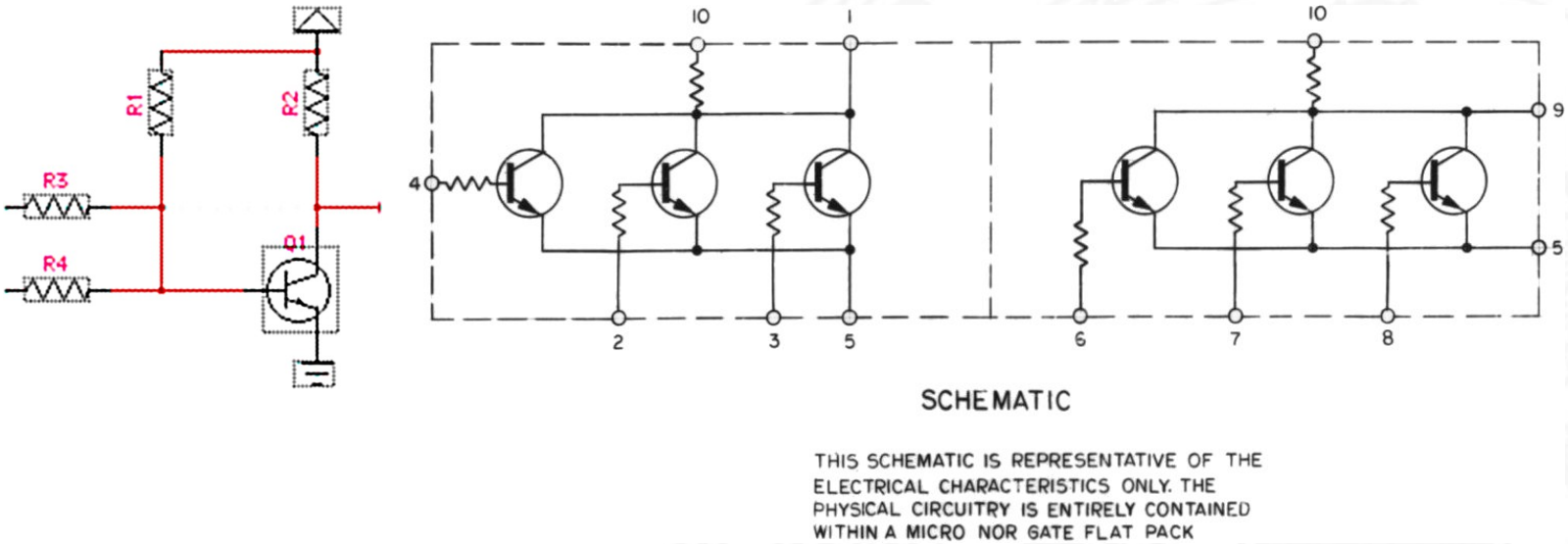
- Bipolar-Schaltungen: RTL, DTL, TTL, I^2L, ECL
  - $\Rightarrow$ nur noch für spezielle Anwendungszwecke: Schnittstellen...
  - stromgesteuert $\Rightarrow$ höhere Leistungsaufnahme
    kaum taktabhängig
- MOS-Schaltungen: NMOS, CMOS, BiCMOS
  - $\Rightarrow$ Rechnertechnologie: CMOS
  - spannungsgesteuert $\Rightarrow$ geringer statischer Verbrauch
  - Gate-Kapazitäten umladen $\Rightarrow$ taktabhängig !!!
  - geringer Flächenbedarf

Aktuelle MOS-Schaltungstechniken werden später detailliert beschrieben – hier kurz die Bipolar-Logikfamilien (historisch)

Schaltungstechniken / Logikfamilien (cont.)

RTL Resistor-Transistor Logic: Inverter, NAND, NOR

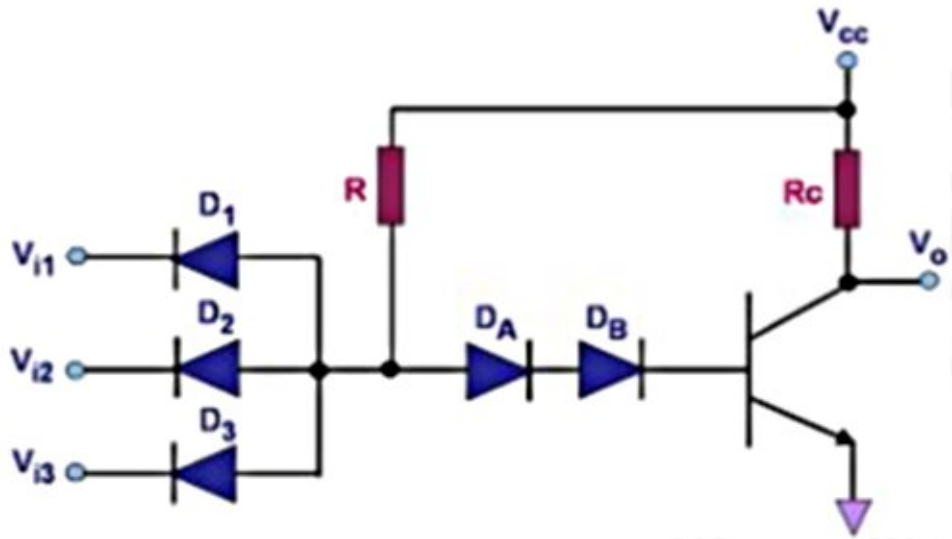
- Widerstände verknüpfen, Verstärkung mit Transistoren
- in den 50er Jahren von TI entwickelt
- erste Apollo Technik



Schaltungstechniken / Logikfamilien (cont.)

DTL Diode-Transistor Logic: Inverter, NAND

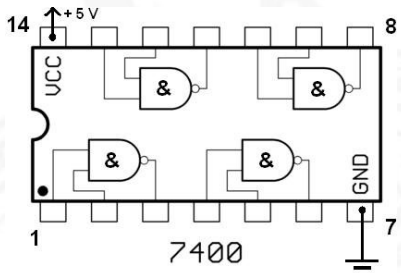
- Dioden verknüpfen, Verstärkung mit Transistoren
- Anfang der 60er Jahre

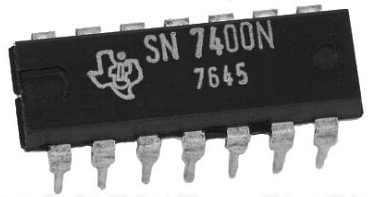


Schaltungstechniken / Logikfamilien (cont.)

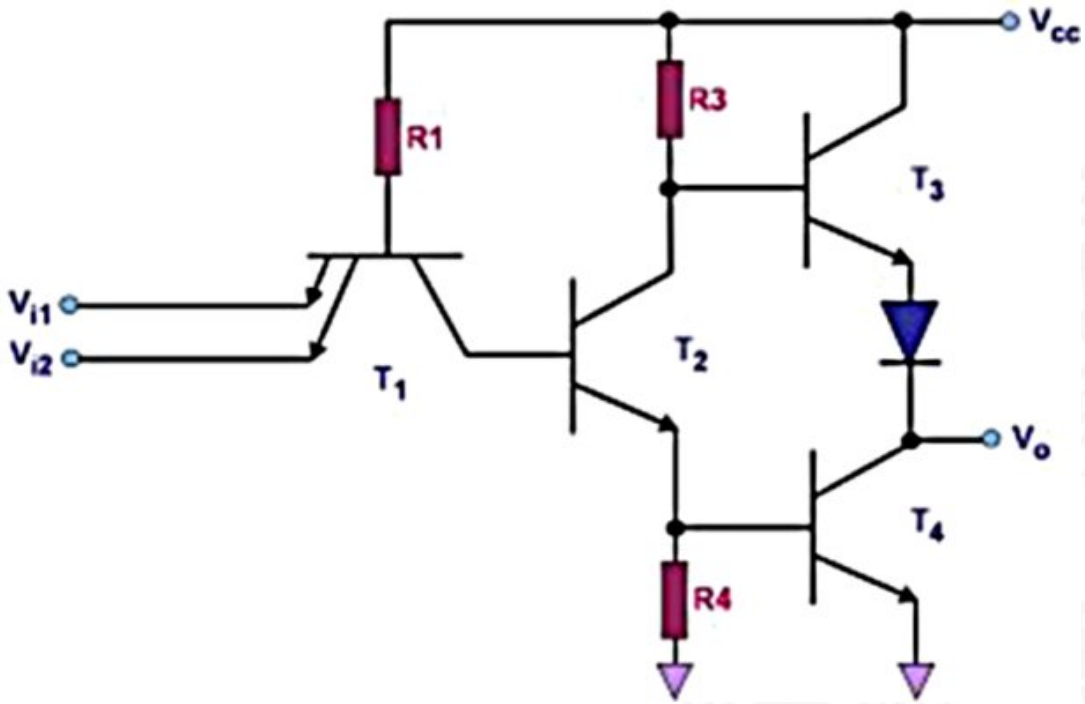
TTL Transistor-Transistor Logic: Inverter, NAND

- DTL Nachfolger: multi-Emitter Transistoren statt Dioden
- ab '62, Boom bis '90er Jahre
- erste Serien Bauteile: 7400-Serie TI „quasi"-Standard
- Anwendungen: glue-Logic, Mini- und Mikrocomputer (DEC, Data General, HP)





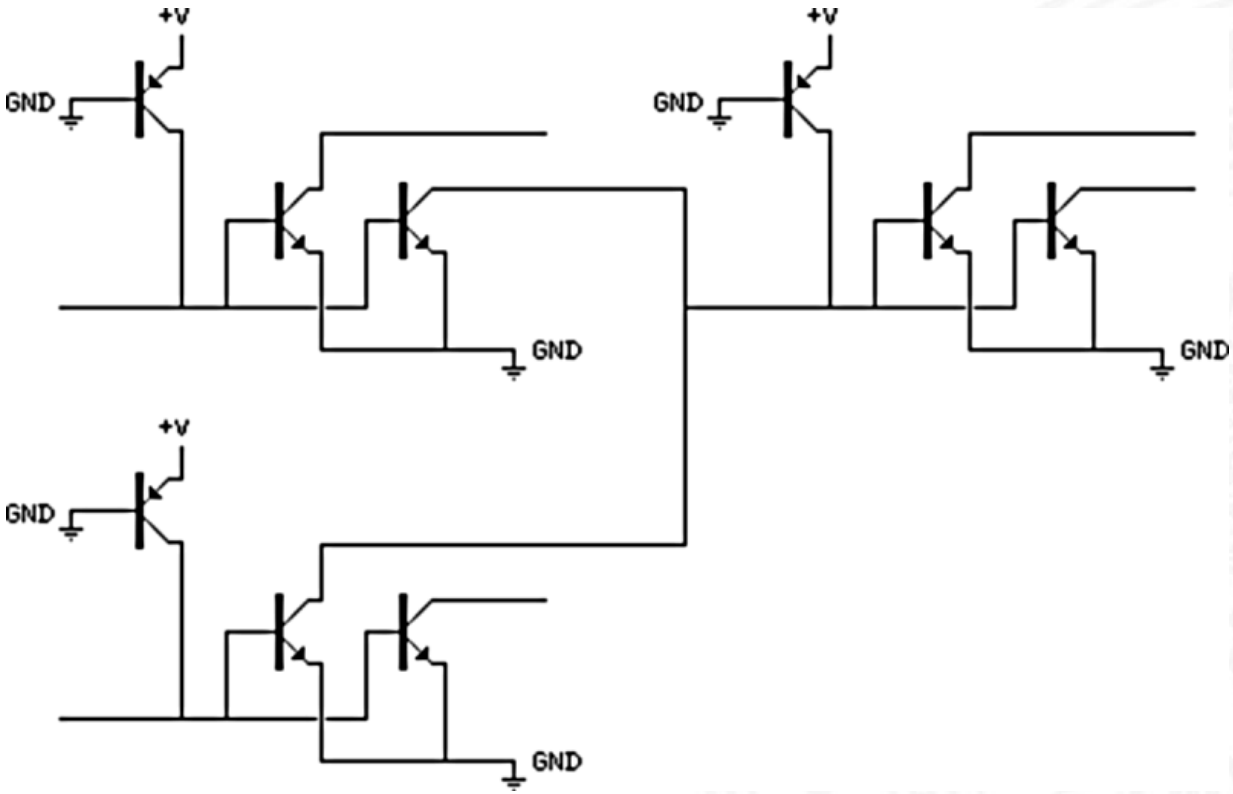
Schaltungstechniken / Logikfamilien (cont.)



Schaltungstechniken / Logikfamilien (cont.)

I^2L Integrated Injection Logic: NAND ($\neg a \vee \neg b$)

- Logik ensteht durch *wired-or* mehrerer Ausgänge

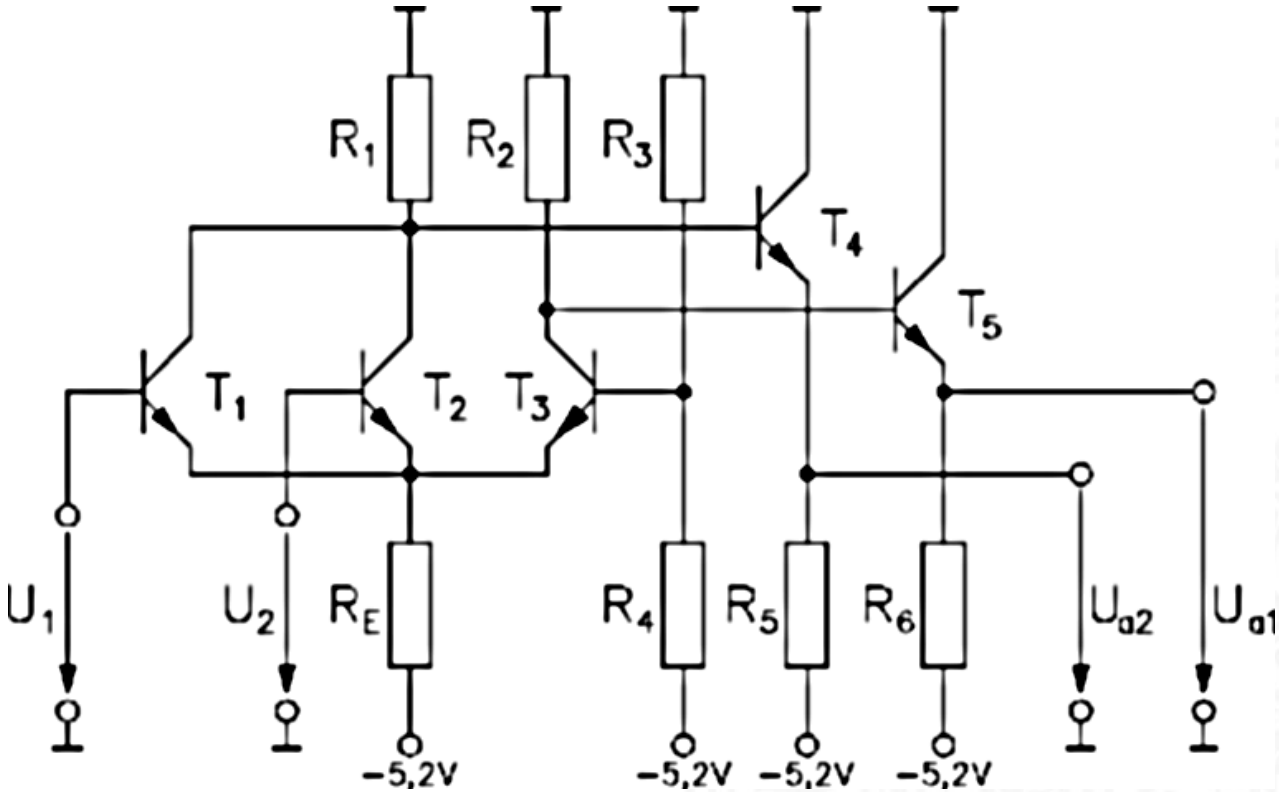


Schaltungstechniken / Logikfamilien (cont.)

ECL Emitter Coupled Logic: Inv. + Buf., OR + NOR

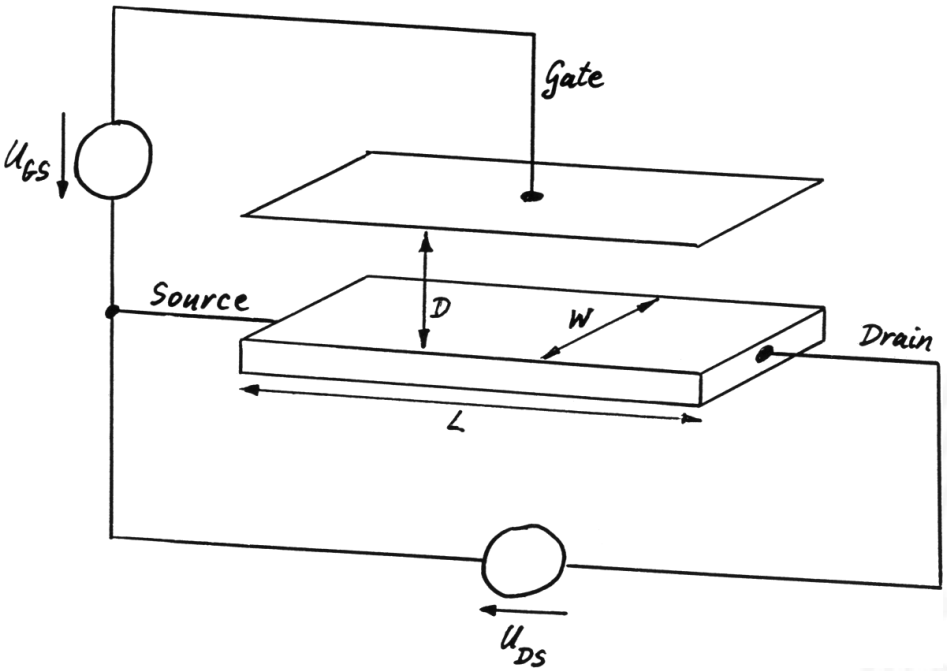
- Differenzverstärker, konstanter Stromfluss
  $\Rightarrow$ Leistungsaufnahme
- Transistoren nicht im Sättigungsbereich
  $\Rightarrow$ Geschwindigkeit
- Ausgänge normal und negiert möglich
- früher: ALUs von Großrechnern

Schaltungstechniken / Logikfamilien (cont.)



MOS-Transistoren

Theorie der MOS-Transistoren



- $U_{GS} < U_P; 0 \leq U_{DS}$
  $I_D = 0$

Sperren

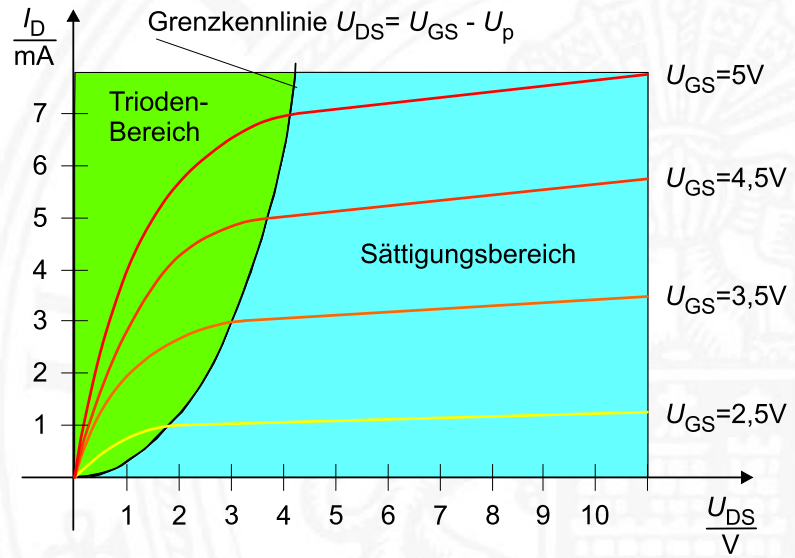
MOS-Transistoren (cont.)

- $U_{GS} \geq U_P$; $0 \leq U_{DS} \leq U_{GS} - U_P$ Leiten, normaler Kanal

  $I_D = \varepsilon_0 \varepsilon^* \mu \frac{W}{LD}[(U_{GS} - U_P)U_{DS} - \frac{1}{2}U_{DS}^2]$

- $U_{GS} > U_P$; $U_{DS} \geq U_{GS} - U_P$ Leiten, Kanalabschnürung

  $I_D = \varepsilon_0 \varepsilon^* \mu \frac{W}{LD} \frac{1}{2}(U_{GS} - U_P)^2$

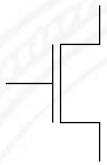


Kennlinienfeld

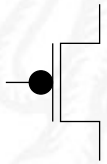
MOS-Transistoren (cont.)

Abstraktion für digitale Schaltungen

- n-Kanal Transistor
  - Elektronenleitung
  - $U_{GS} > U_P$: Source an *Gnd*, positive Spannungen
Gate		Transistor
Gnd	0	sperrt
Vdd	1	leitet

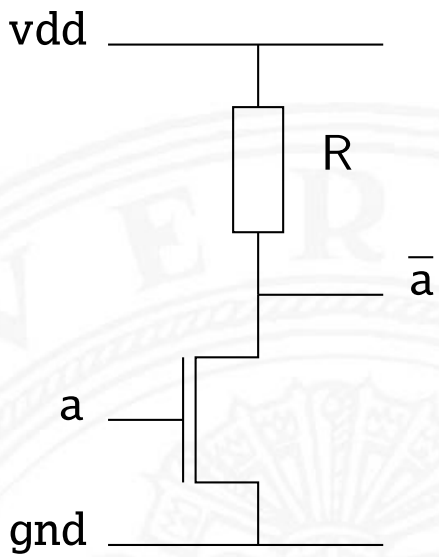


- p-Kanal Transistor
  - Löcherleitung
  - $U_{GS} < -U_P$: Source an *Vdd*, negative Spannungen
Gate		Transistor
Gnd	0	leitet
Vdd	1	sperrt



NMOS-Schaltungen

Beispiel: Inverter



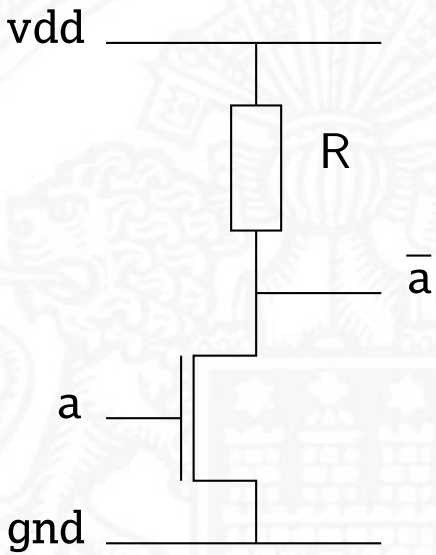
Funktionsweise

- Hochohmiger Widerstand R
  $R_{T.on} \ll R_R \ll R_{T.off}$
- Eingang Transistor Ausgang

$a = 0 \rightarrow$ sperrt $\rightarrow$ über R mit Vdd verbunden $= 1$
$a = 1 \rightarrow$ leitet $\rightarrow$ über T mit Gnd verbunden $= 0$

NMOS-Schaltungen (cont.)

- Ausgang treibt Gate-Anschlüsse nachfolgender Gatter
  kapazitive Last $\Rightarrow$ schnelles Entladen über $R_{T.on}$
  langsames Laden über R_R
- Querstrom bei eingeschaltetem Gatter
  $I = \frac{Vdd}{R_{T.on}+R_R}$



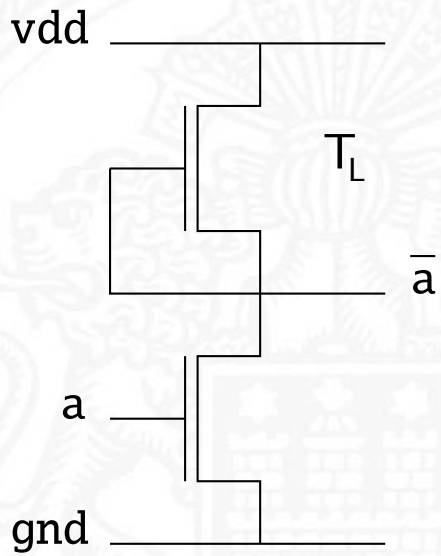
NMOS-Prozesse

Technische Realisierung

- polykristallines Silizium (Polysilizium) als Leiter
  - Gates der Transistoren
  - (kurze) Verbindungen
- Realisierung der Widerstände

Material	$\approx$ Widerstand [$\Omega/\square$]
Metall	$\approx$ 0,03
Diffusion (n^+, p^+)	$\approx$ 10-100
Polysilizium	$\approx$ 15-100
Transistor T_{on}	$\approx$ 1 k
–"– T_{off}	$\approx$ 10 M

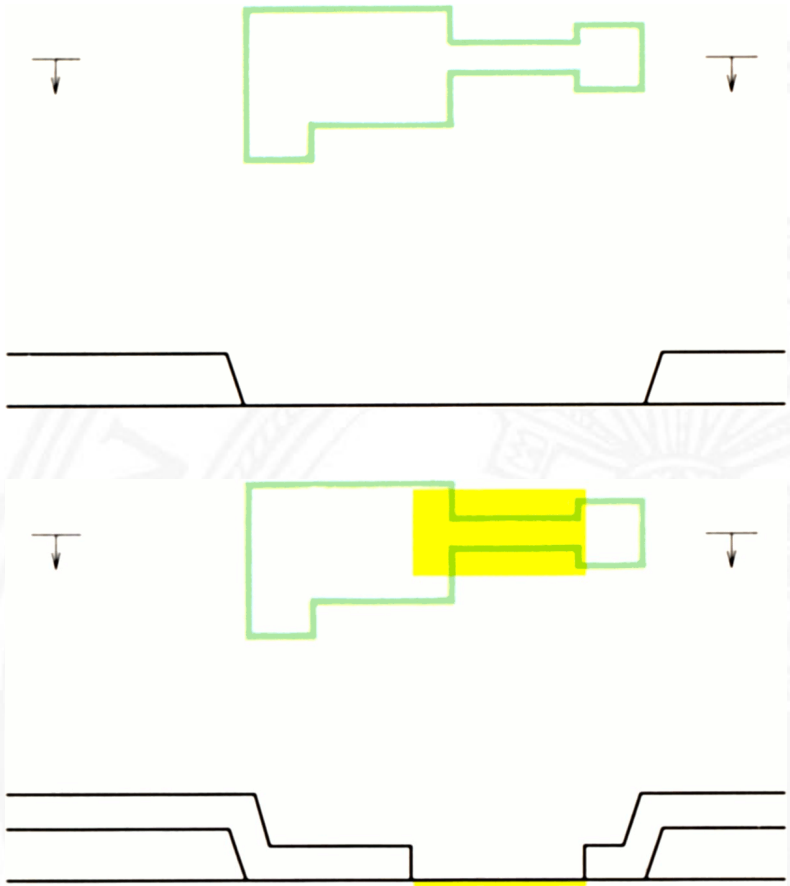
$\Rightarrow$ Lasttransistor, selbstleitend!



NMOS-Prozesse (cont.)

Entwurf eines NMOS Inverters

[MC80]

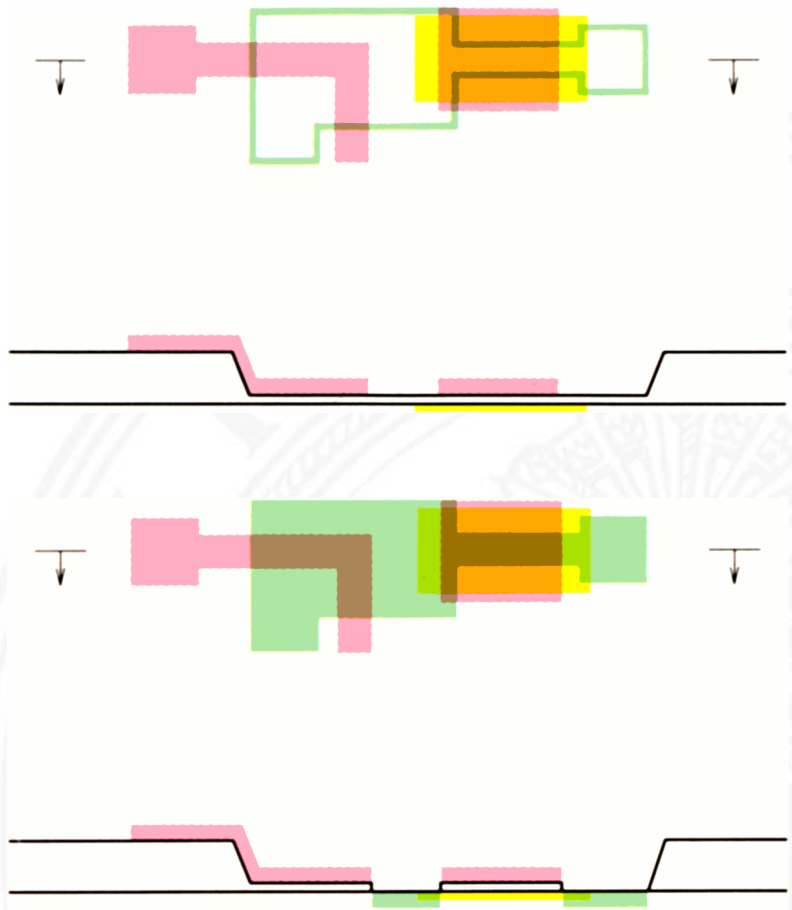


1. aktive Fläche definieren

2. Ionenimplantation für Lasttransistor
   - Selbstleitung

NMOS-Prozesse (cont.)

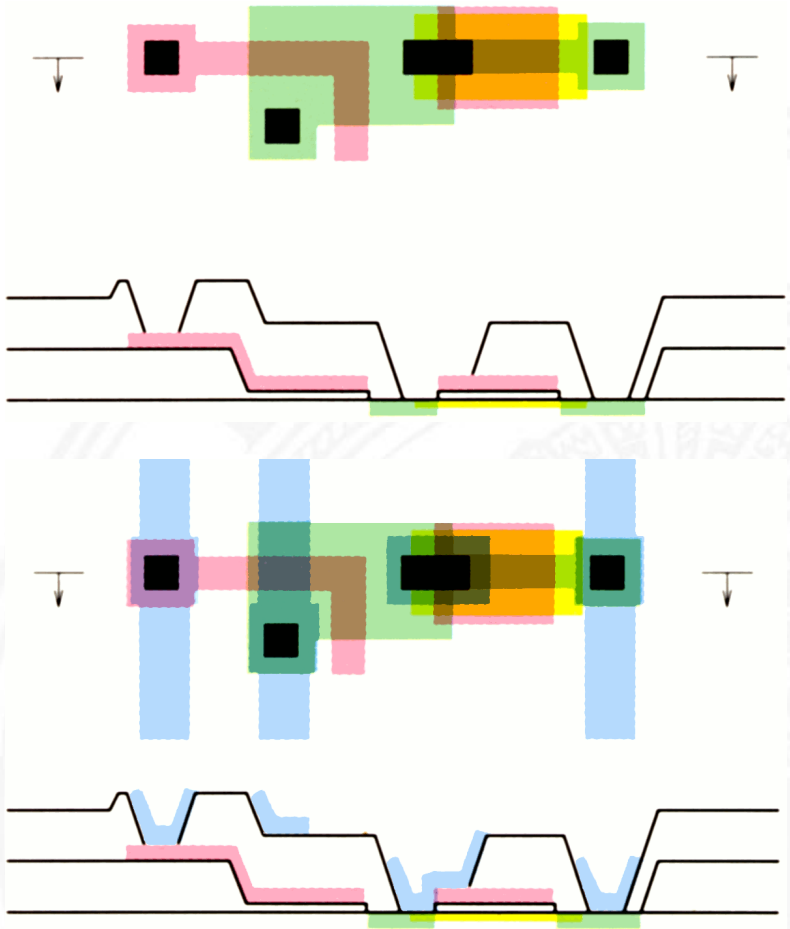
3. Polysilizium / Gate



- n^+-Diffusion
  - aktive Fläche $\wedge\neg$ Polysilizium
  - Selbstjustierung
    self-alignment

NMOS-Prozesse (cont.)

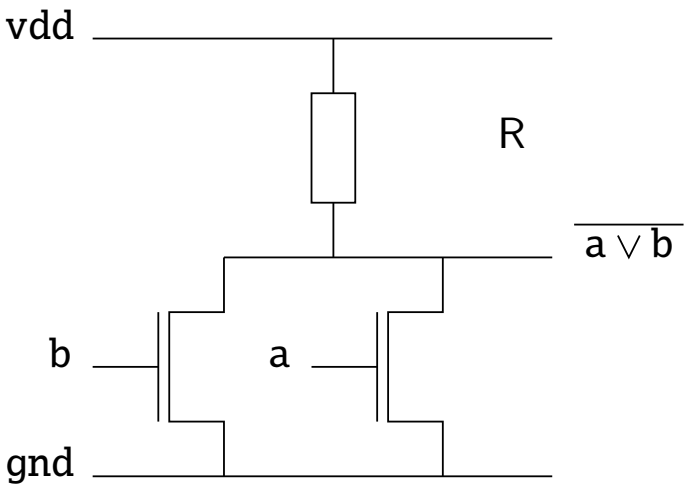
4. Kontaktlöcher



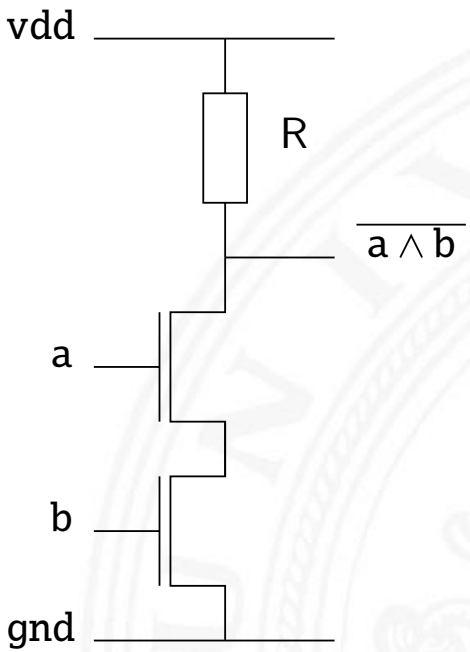
5. Metallisierung

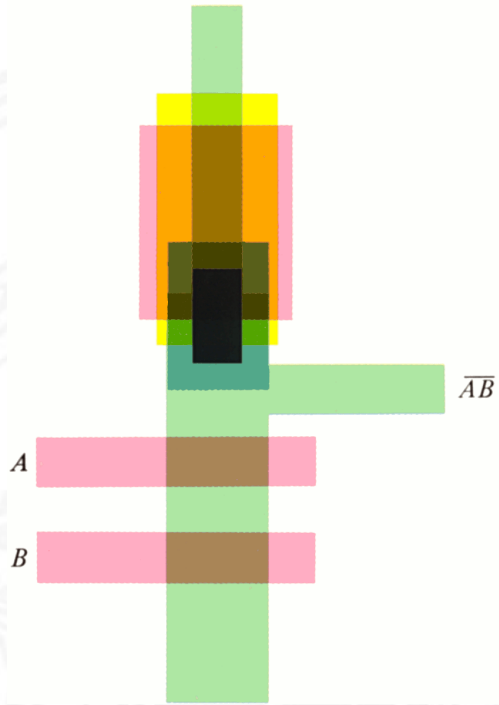
NMOS-Gatter

NOR



NAND





NMOS-Technik

+ einfache Fertigung: min. 5 Masken
– Fläche für Lasttransistor
– Dimensionierung des Lasttransistors
(Parallelschaltung mehrerer Eingänge, z.B. bei NOR)
– Statischer Stromverbrauch: $f(\langle NMOS \rangle) = 0$
– langsam, asymmetrisches Schaltverhalten
– asymmetrische Spannungspegel, Spannungsteiler R_R und $R_{T.on}$

Selbstjustierung

Technik bei der Herstellung von MOS-Transistoren, bei der das Gate die Source- und Drain-Gebiete der Transistoren mit festlegt

- Diffusionsmasken trennen Source und Drain der Transistoren nicht, sie beinhalten auch die Fläche unter dem Gate
  Maske: „*active area*“
- Vor der Diffusion wird das (Poly-) Gate erzeugt
- Diffusion findet im Gebiet der Diff.-Maske statt, die nicht vom Gate bedeckt ist

$\Rightarrow$ Trennung von Source und Drain

$+$ kleine Verschiebungen der Gate-Maske ohne Auswirkung

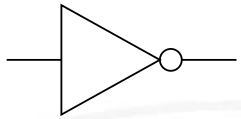
CMOS-Schaltungen

CMOS – **C**omplementary **M**etal **O**xide **S**ilicon

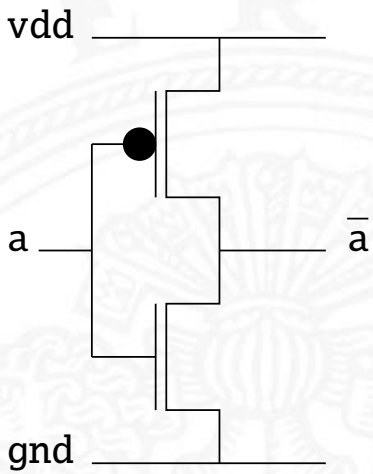
- Standard VLSI Systemtechnologie
- \+ geringer statischer Stromverbrauch
- \+ weniger Flächenbedarf als andere Technologien
- \+ (prozesstechnisch) gut skalierbar
- \+ sehr schnelle Schaltungen möglich
- \+ flexibel einsetzbar
  - realisierbare Logik: NAND, NOR, Komplexgatter...
  - statische, dynamische Schaltungen
  - „Schaltungstricks“...
- – elektrostatisch empfindliche Eingänge $\Rightarrow$ Schutzschaltungen
- – dynamische Leistungsaufnahme

CMOS-Schaltungen (cont.)

Beispiel: Inverter



Funktionsweise



- selbstsperrende p- und n-Kanal Transistoren
- komplementär beschaltet
- Ausgang: Pfad über p-Transistoren zu *Vdd*
  –"– n-Transistoren zu *Gnd*
- genau *einer* der Pfade leitet
- Eingang Trans$_P$ Trans$_N$ Ausgang

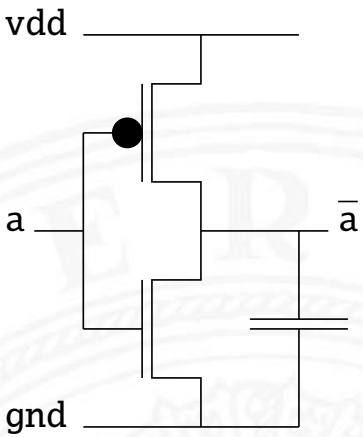
  $a = 0 \rightarrow$ leitet / sperrt $\rightarrow$ über T$_P$ mit *Vdd* verbunden $= 1$

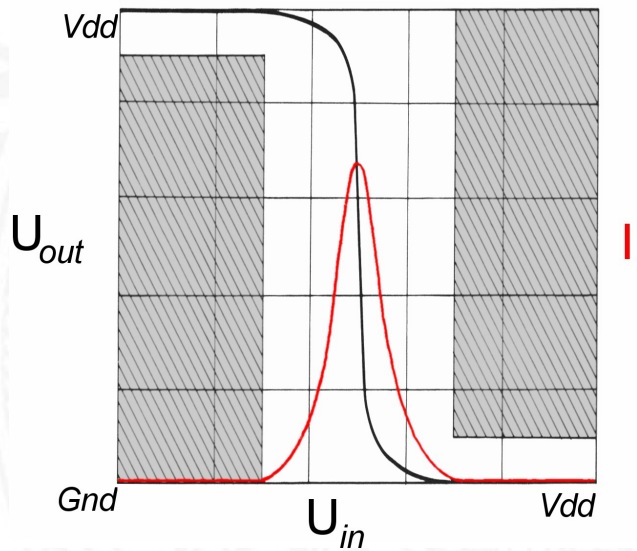
  $a = 1 \rightarrow$ sperrt / leitet $\rightarrow$ über T$_N$ mit *Gnd* verbunden $= 0$

CMOS-Schaltungen (cont.)

Leistungsaufnahme

1. $U_{in} = 0$, bzw. Vdd: Sperrstrom, nur μA
   $\Rightarrow$ niedrige statische Leistungsaufnahme
2. Querstrom beim Umschalten:
   kurzfristig leiten beide Transistoren
   $\Rightarrow$ Forderung nach steilen Flanken
3. Kapazitive Last: Fanout-Gates
   Energie auf Gate(s): $W = \frac{1}{2} C_T Vdd^2$
   Verlustleistung$_{(0/1/0)}$: $P = C_T Vdd^2 \cdot f$





Transfercharakteristik

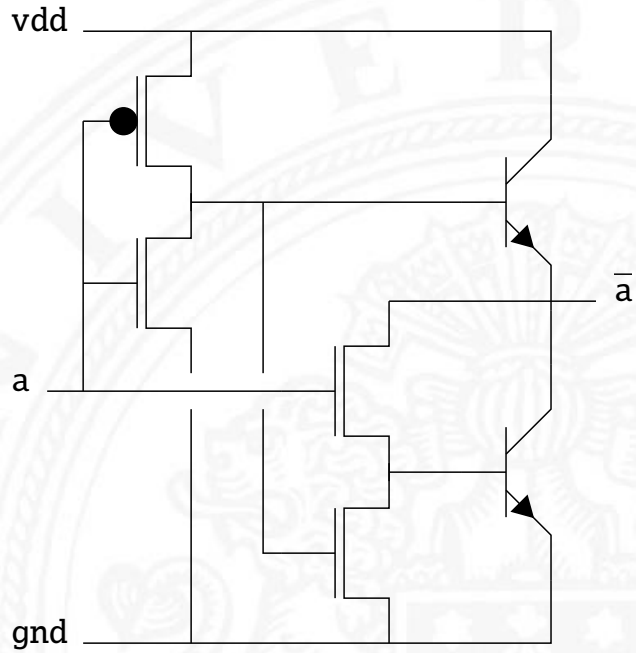
CMOS-Prozesse

Realisierungsmöglichkeiten

- n-Wannen Prozess
- p-Wannen Prozess
- Zwei-Wannen Prozess
- SOI – **S**ilicon **O**n **I**nsulator

BiCMOS – **Bi**polar **CMOS**

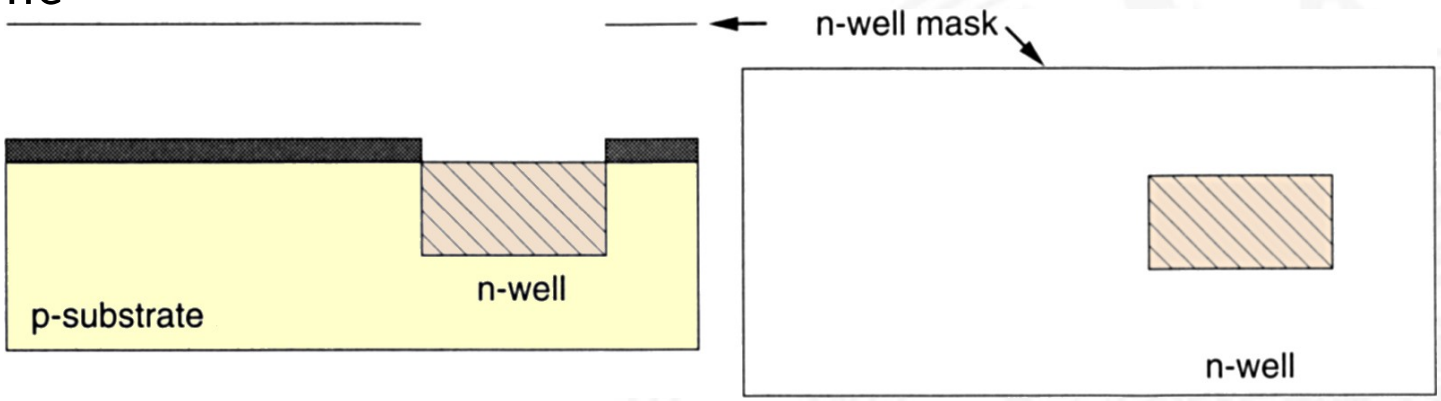
- CMOS Schaltung realisiert Logik
- NPN-Ausgangsstufe
  liefert hohe Ströme



CMOS-Prozesse (cont.)

Ein n-Wannen Prozesses [WE94]

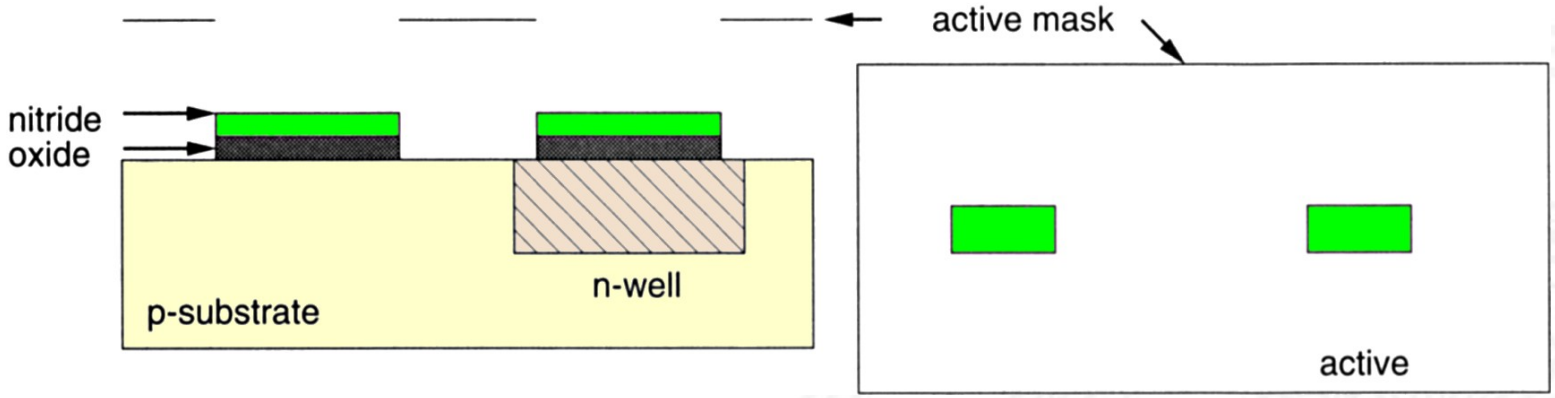
1. Ausgangsmaterial: p-dotiertes Substrat
2. n-Wanne



- Dotierung für p-Kanal Transistoren
- Herstellung: Ionenimplantation oder Diffusion

CMOS-Prozesse (cont.)

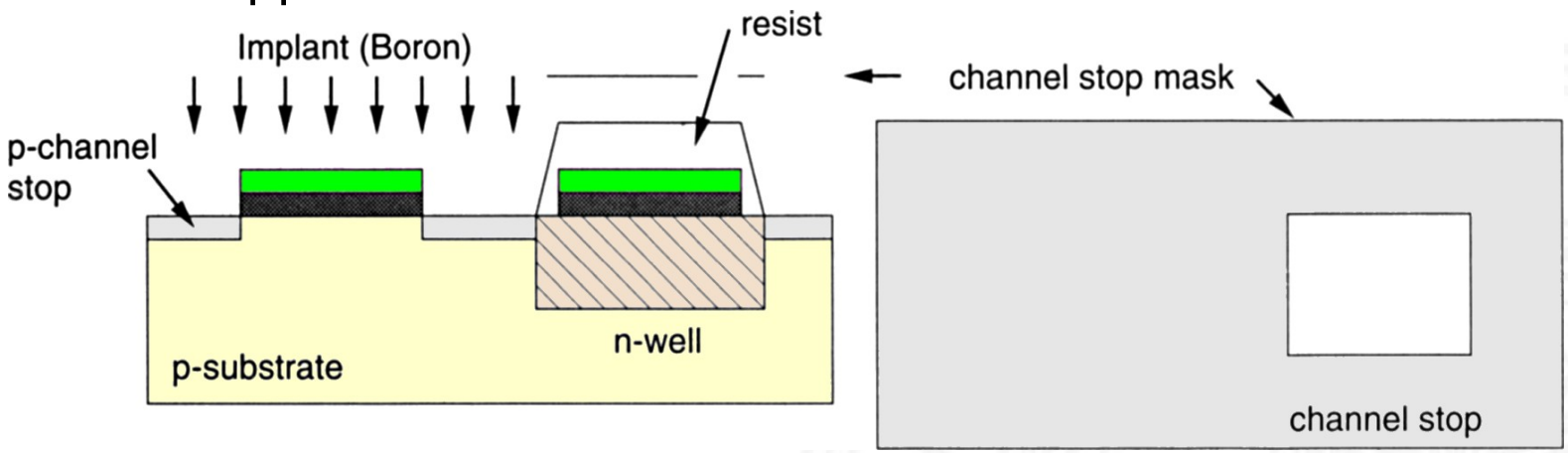
3. „aktive“ Fläche / Dünnoxid



- Spätere Gates und p^+-/n^+-Gebiete
- Herstellung: Epitaxie SiO_2 und Abdeckung mit Si_3N_4

CMOS-Prozesse (cont.)

4. p-Kanalstopp

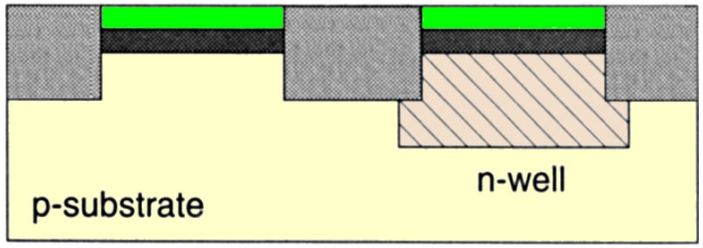


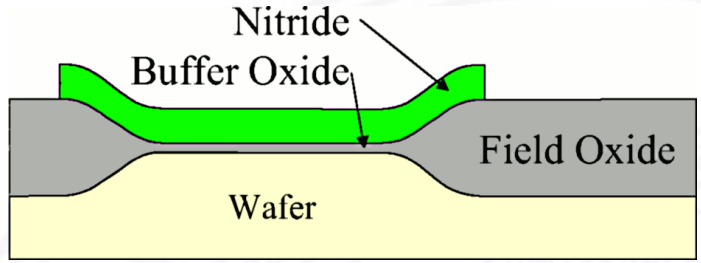
- Begrenzt n-Kanal Transistoren
- p-Wannen Maske, bzw. ¬ n-Wanne
- Maskiert durch Resist und Si_3N_4
- Substratbereiche in denen keine n-Transistoren sind
- Herstellung: p^+-Implant (Bor)

- n-Kanalstopp aktueller Prozesse: analog dazu

CMOS-Prozesse (cont.)

5. Resist entfernen
6. Feldoxid aufwachsen – SiO_2





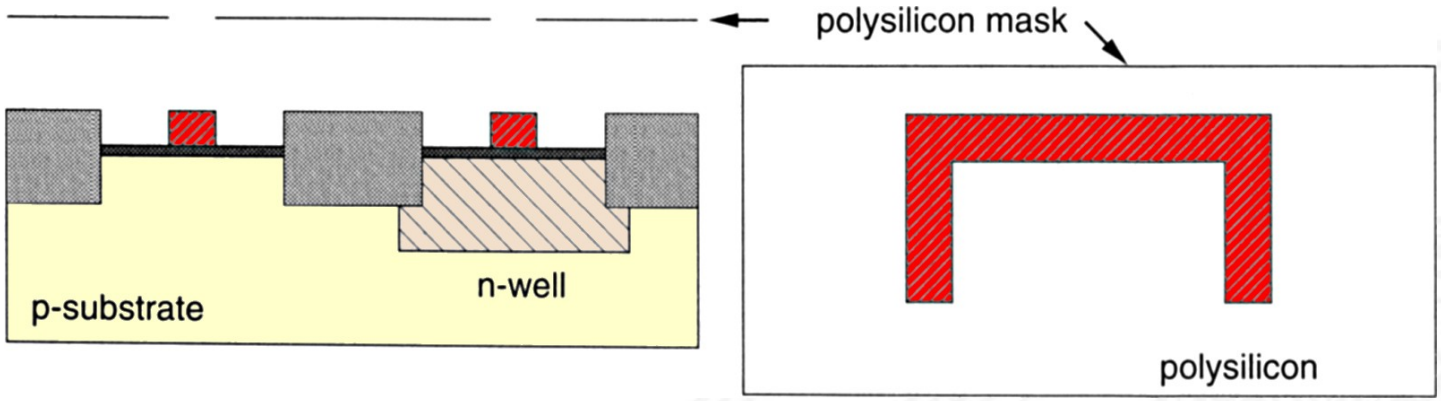
- LOCOS: **Loc**al **O**xidation of **S**ilicon
- Maskiert durch Si_3N_4
- Wächst auch lateral unter Si_3N_4/SiO_2 (aktive) Bereiche engl. *bird's beak*
- Der aktive Bereich wird kleiner als vorher maskiert
- Herstellung: Epitaxie und Oxidation
- Problem: nicht plane Oberfläche

CMOS-Prozesse (cont.)

7. Si_3N_4 entfernen, Gateoxid bleibt SiO_2
8. Transistor Schwellspannungen „justieren“
   - Meist wird das Polysilizium zusätzlich n^+ dotiert
     Grund: bessere Leitfähigkeit
   - Problem: $U_D(T_N) \approx \quad 0{,}5 \ldots \quad 0{,}7\,\mathrm{V}$
     $U_D(T_P) \approx -1{,}5 \ldots -2{,}0\,\mathrm{V}$
   - Maske: n-Wanne, bzw. p-Wanne
   - Herstellung: Epitaxie einer leicht negativ geladenen Schicht an der Substratoberfläche

CMOS-Prozesse (cont.)

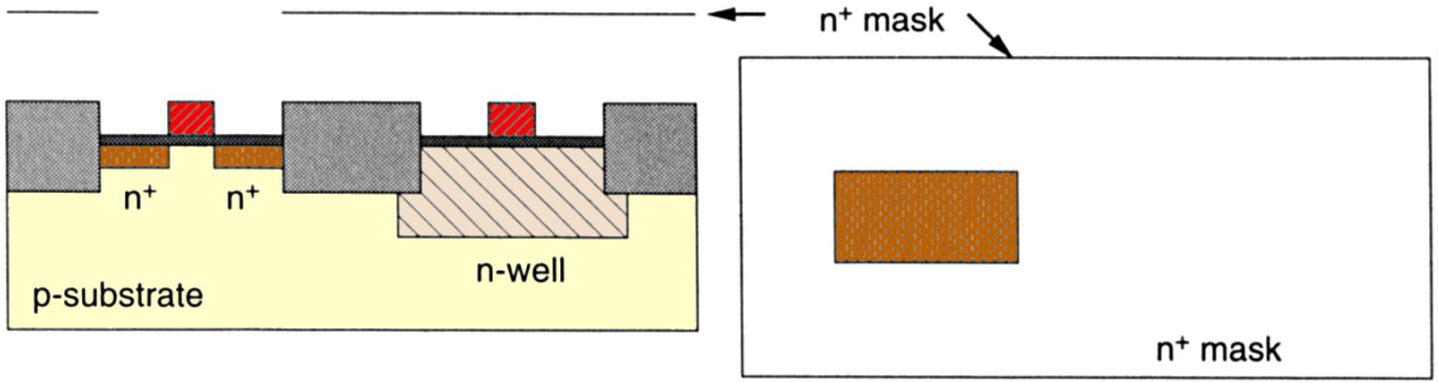
9. Polysilizium Gate



- Herstellung: Epitaxie von Polysilizium, Ätzen nach Planarprozess

CMOS-Prozesse (cont.)

10. n^+-Diffusion

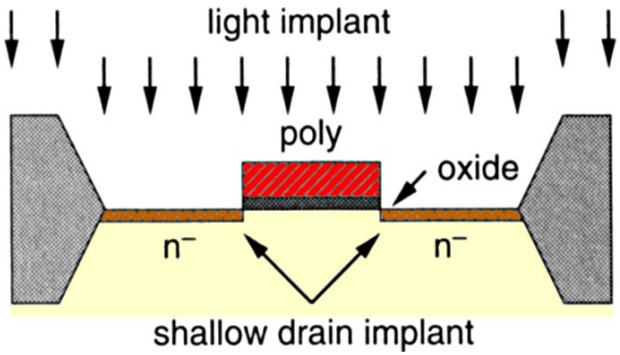


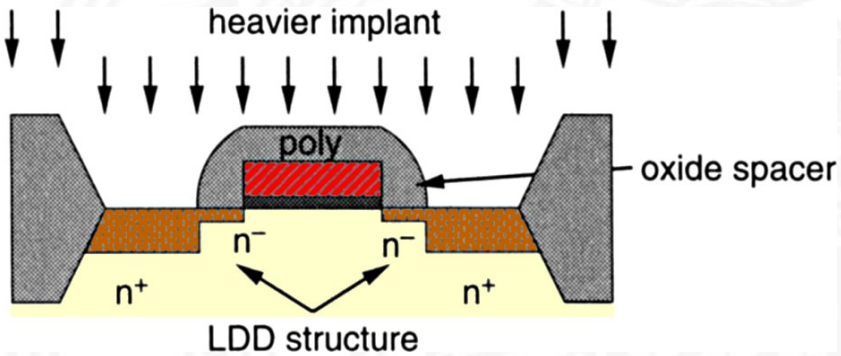
- Erzeugt Source und Drain der n-Kanal Transistoren
- Maskiert durch aktiven Bereich, n^+-Maske und Polysilizium
  $\Rightarrow$ Selbstjustierung
- Dotiert auch das Polysilizium Gate leicht (s.o.)
- Herstellung: Ionenimplantation, durchdringt Gateoxid

CMOS-Prozesse (cont.)

Zusätzliche Schritte bei der Source/Drain Herstellung

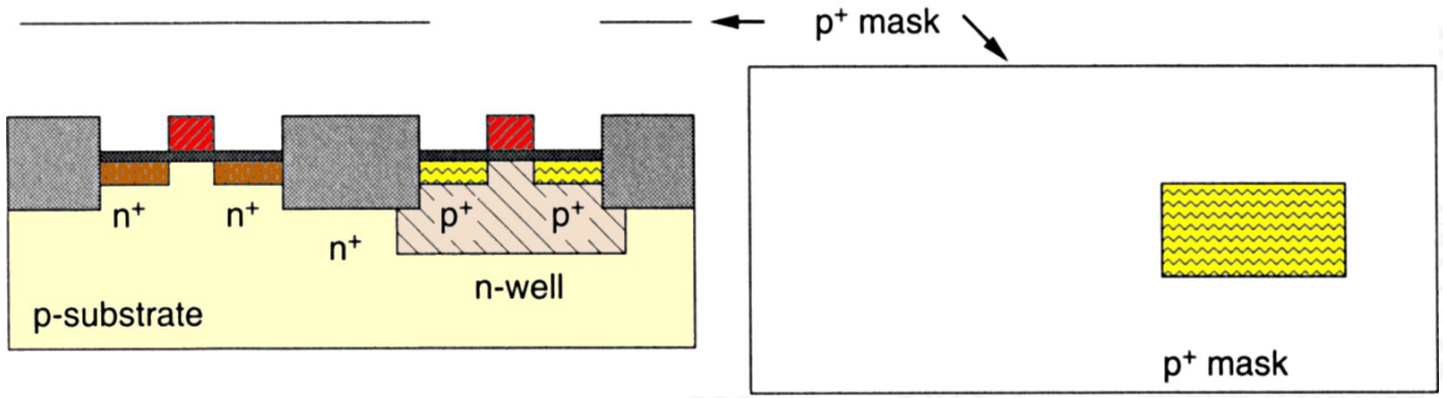
- Problem „Hot-Carrier“ Effekte (schnelle Ladungsträger): Stoßionisation, Gateoxid wird durchdrungen...
- Lösung: z.B. LDD (**L**ightly **D**oped **D**rain)
  a. „flaches“ n-LDD Implant
  b. zusätzliches SiO_2 über Gate aufbringen (*spacer*)
  c. „normales“ n^+-Implant
  d. Spacer SiO_2 entfernen





CMOS-Prozesse (cont.)

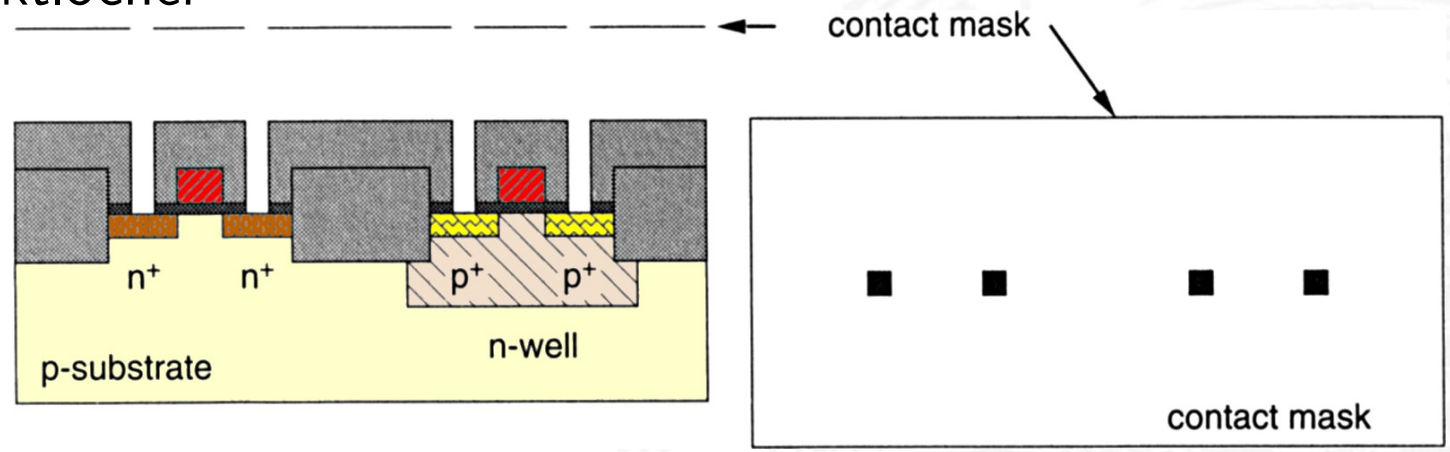
11. p^+-Diffusion



- Erzeugt Source und Drain der p-Kanal Transistoren
- Maskiert durch aktiven Bereich, p^+-Maske und Polysilizium
  $\Rightarrow$ Selbstjustierung
- teilweise implizite p^+-Maske $= \neg$ n^+-Maske
- wenig schnelle Ladungsträger (Löcher), meist keine LDD-Schritte
- Herstellung: Ionenimplantation, durchdringt Gateoxid

CMOS-Prozesse (cont.)

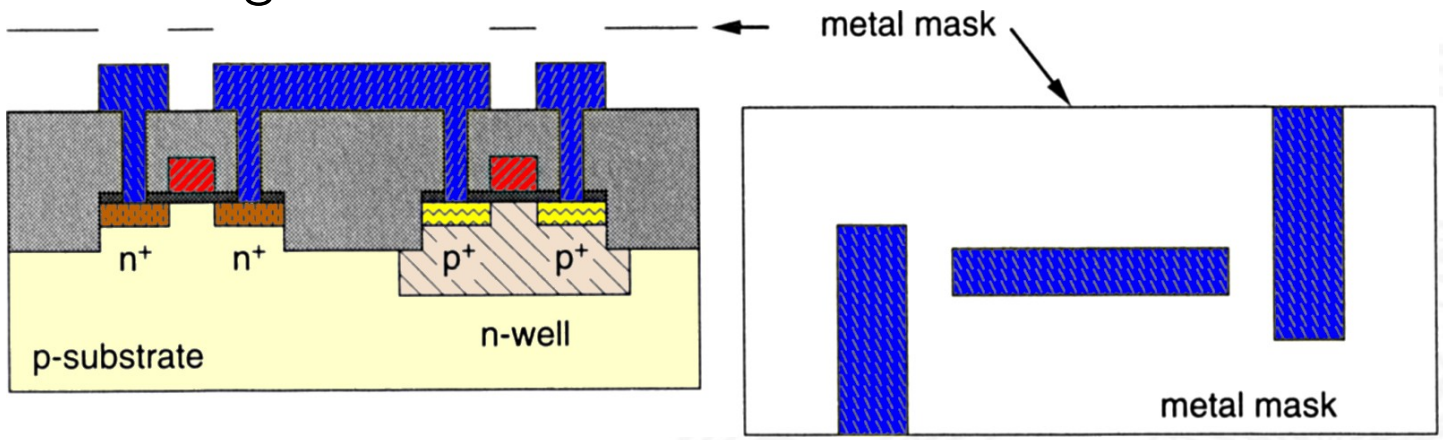
12. SiO_2 aufbringen, Feldoxid
    - Strukturen isolieren
    - Herstellung: Epitaxie
13. Kontaktlöcher



- Verbindet (spätere) Metallisierung mit Polysilizium oder Diffusion
- Anschlüsse der Transistoren: Gate, Source, Drain
- Herstellung: Ätzprozess

CMOS-Prozesse (cont.)

14. Metallverbindung



- Erzeugt Anschlüsse im Bereich der Kontaktlöcher
- Herstellung: Metall aufdampfen, Ätzen nach Planarprozess

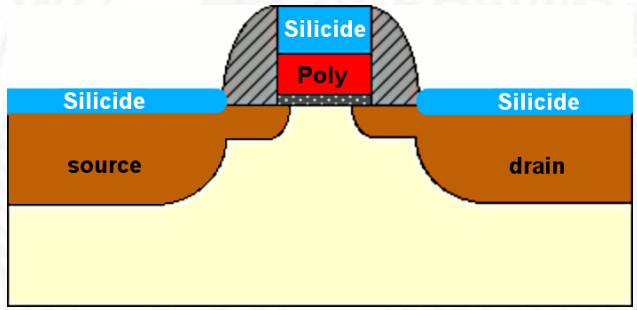
15. weitere Metalllagen
    - Weitere Metallisierungen, bis zu 7 $\times$ Metall
    - Schritte: 12. bis 14. wiederholen

CMOS-Prozesse (cont.)

16. Passivierung
    - Chipoberfläche abdecken, Plasmanitridschicht
17. Pad-Kontakte öffnen

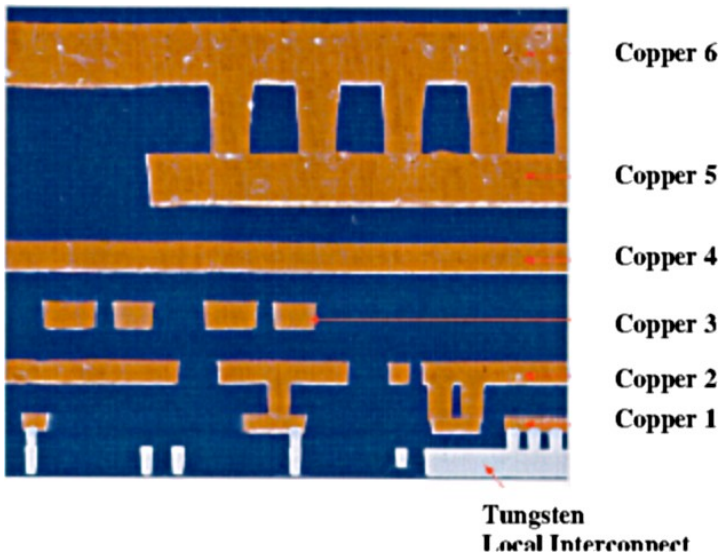
Zahlreiche Erweiterungen für Submikron CMOS-Prozesse

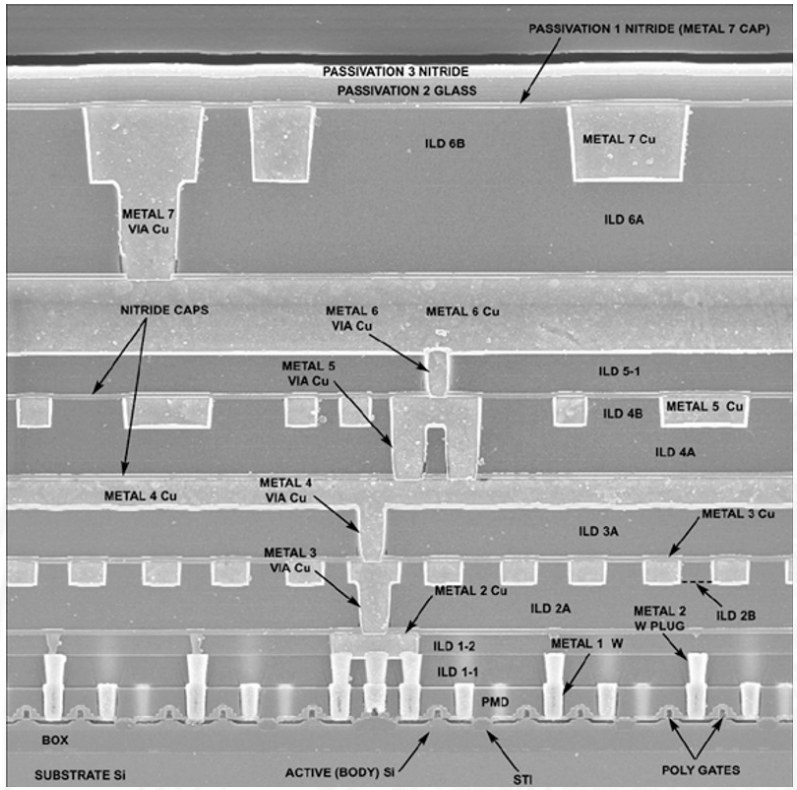
- „vergrabene“ Layer
  - verbessern elektrische Eigenschaften
  - Bipolar-Transistoren
  - Analog-Schaltungen
- Gate Spacer, seitlich SiO_2
- Silizidoberflächen: verringern Kontaktwiderstand zu Metallisierung



CMOS-Prozesse (cont.)

- Kupfer Metallisierung

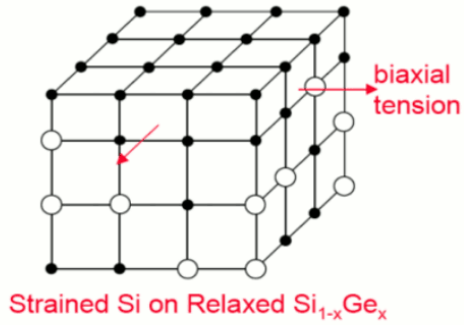


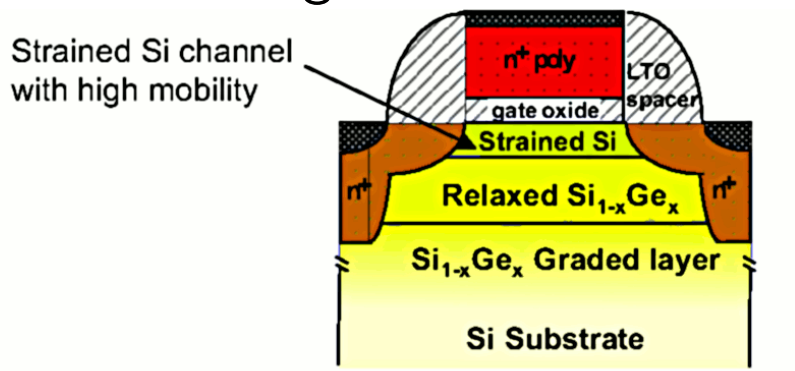


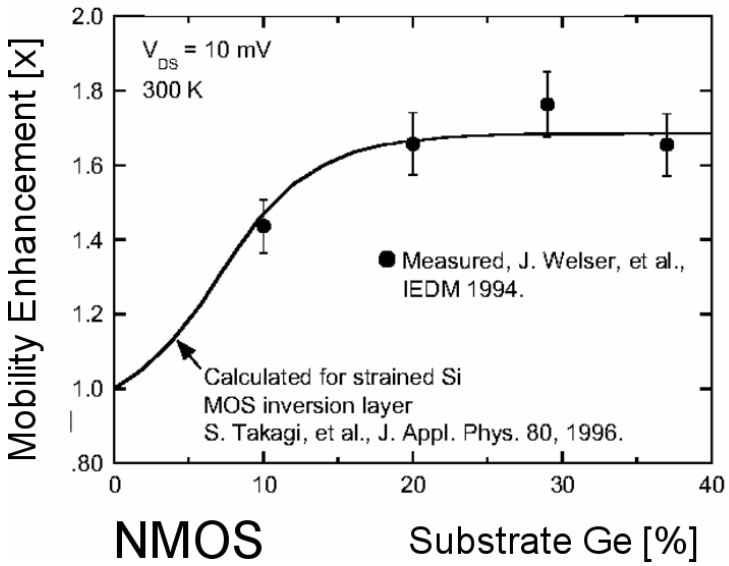
- high-k Dielektrika: Gate-Isolierung dicker, weniger Leckströme

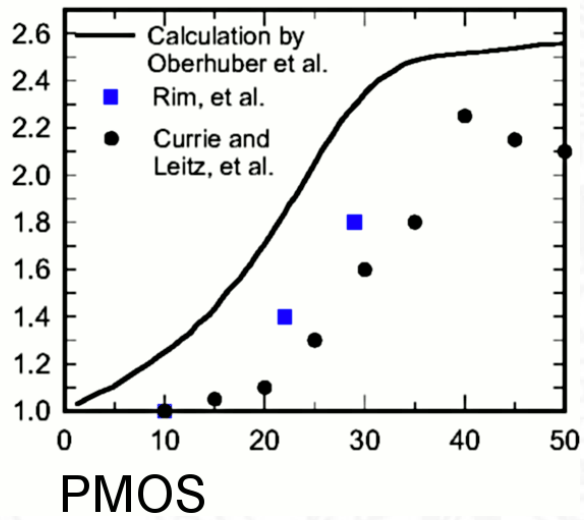
CMOS-Prozesse (cont.)

- „gestrecktes“ Silizium: höhere Beweglichkeit



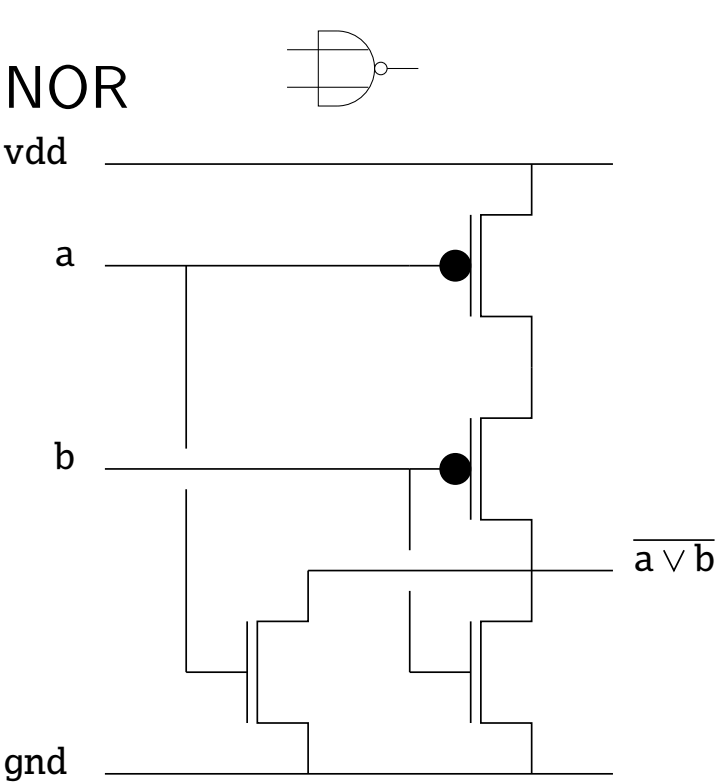




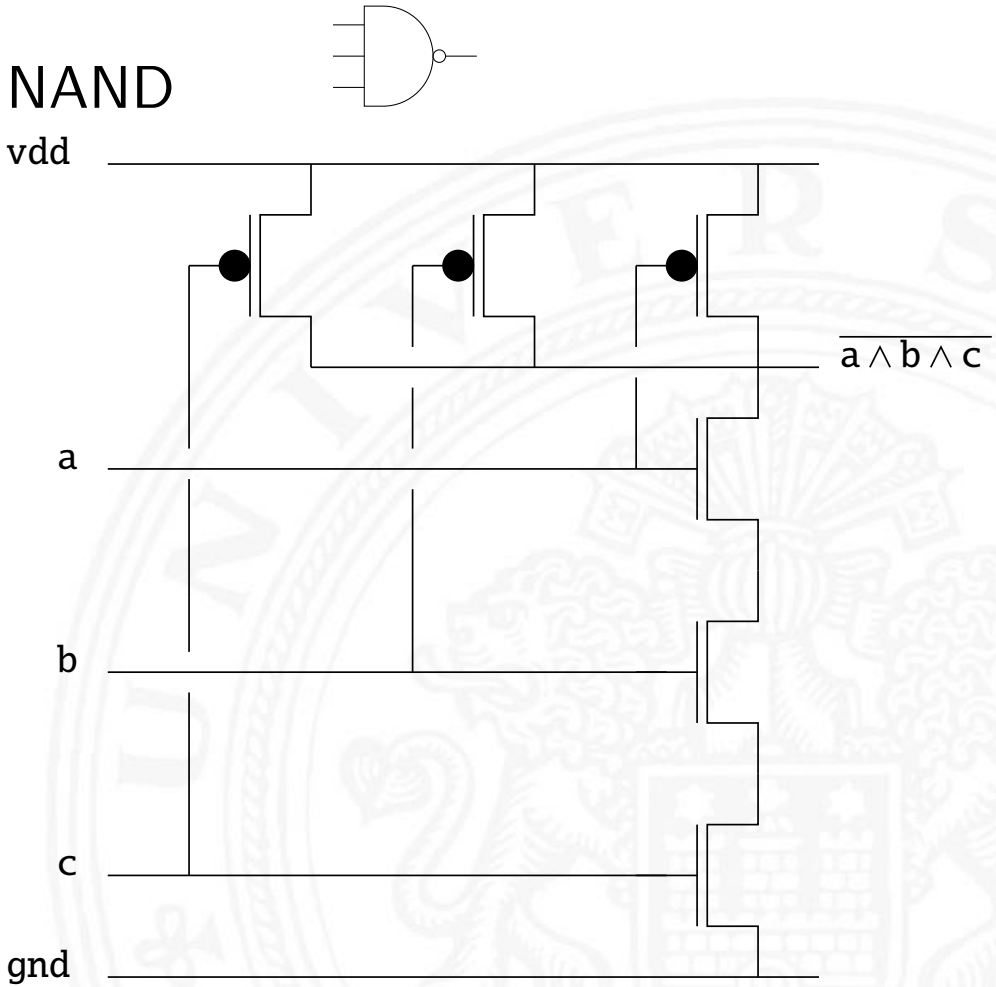


CMOS-Gatter

NOR

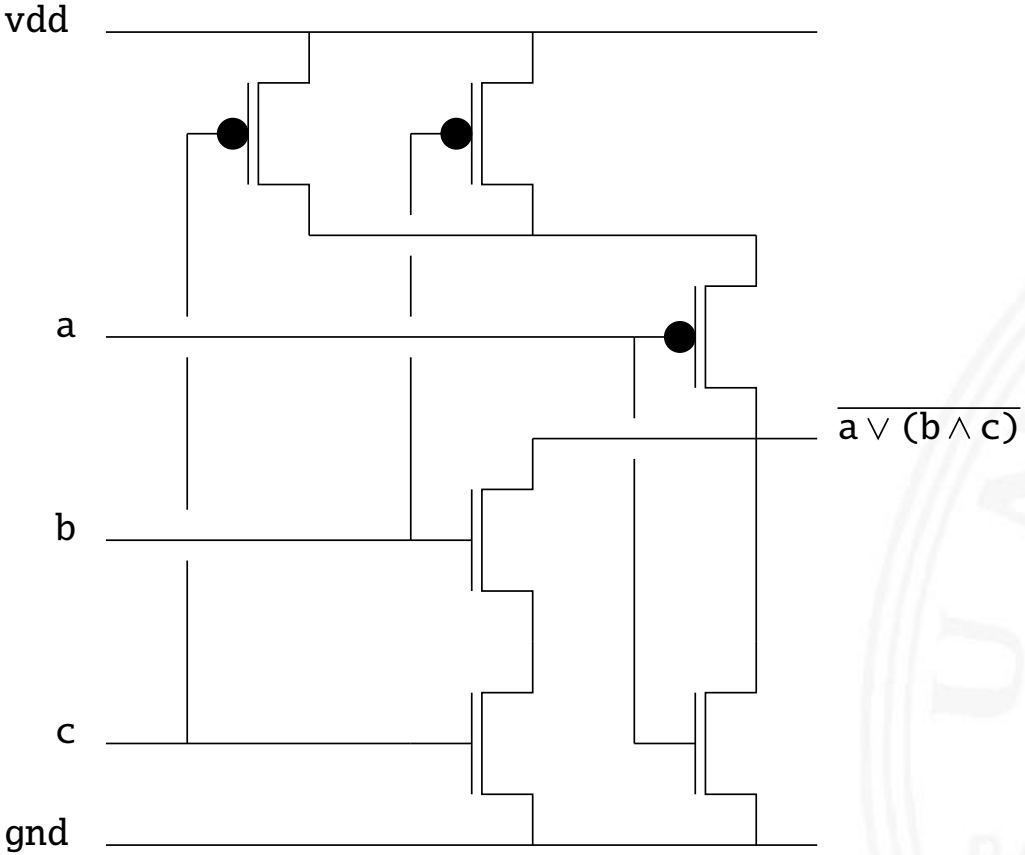


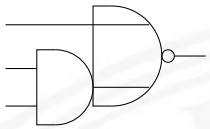
NAND



CMOS-Gatter (cont.)

Komplexgatter





CMOS-Gatter (cont.)

- Schaltungen: *negierte monotone boole'sche Funktionen*
- Beliebiger schaltalgebraischer Ausdruck *ohne Negation*: $\vee, \wedge$
- Negation des gesamten Ausdrucks: Ausgang *immer* negiert
- je Eingang: ein Paar p-/n-Kanal Transistoren
- Dualitätsprinzip: n- und p-Teil des Gatters

n-Teil		p-Teil	Logik, ohne Negation
seriell	$\Leftrightarrow$	parallel	$\equiv \wedge$ / und
parallel	$\Leftrightarrow$	seriell	$\equiv \vee$ / oder

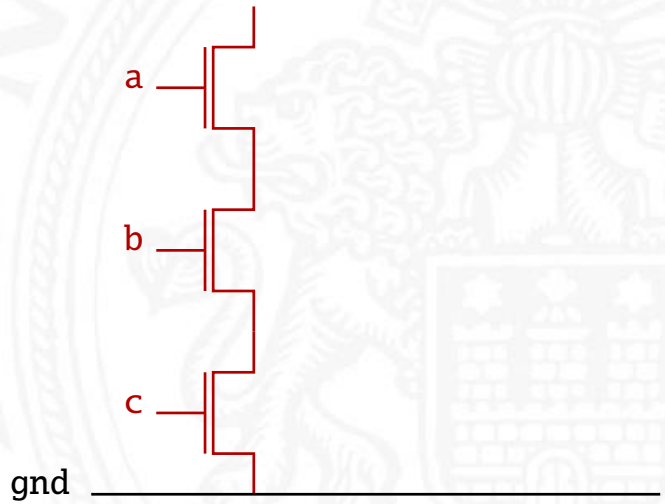
- Konstruktion
  1. n-Teil aus Ausdruck ableiten
  2. p-Teil dual dazu entwickeln

CMOS-Komplexgatter

Beispiel: $\overline{(a \wedge b \wedge c) \vee d \vee (e \wedge f)}$

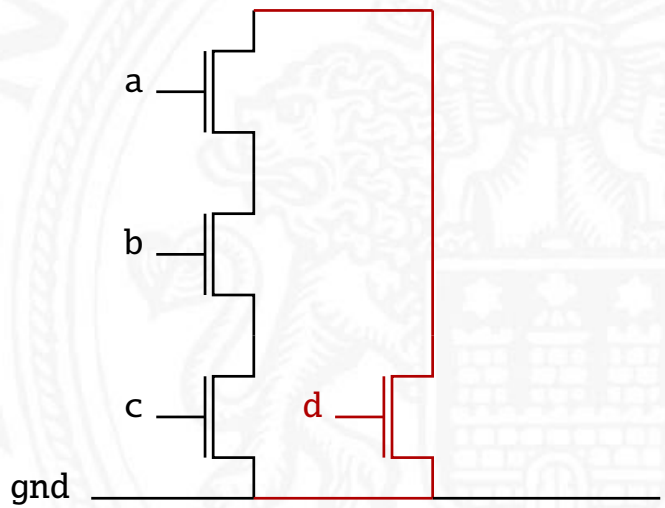
CMOS-Komplexgatter

Beispiel: $\overline{(a \wedge b \wedge c) \vee d \vee (e \wedge f)}$



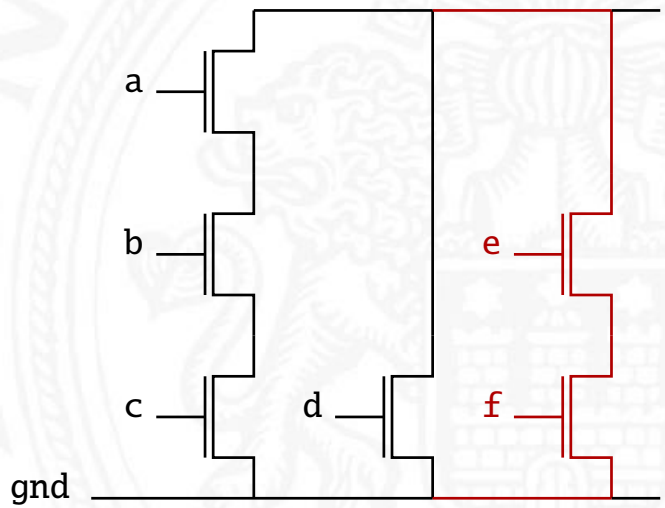
CMOS-Komplexgatter

Beispiel: $\overline{(a \wedge b \wedge c) \vee d \vee (e \wedge f)}$



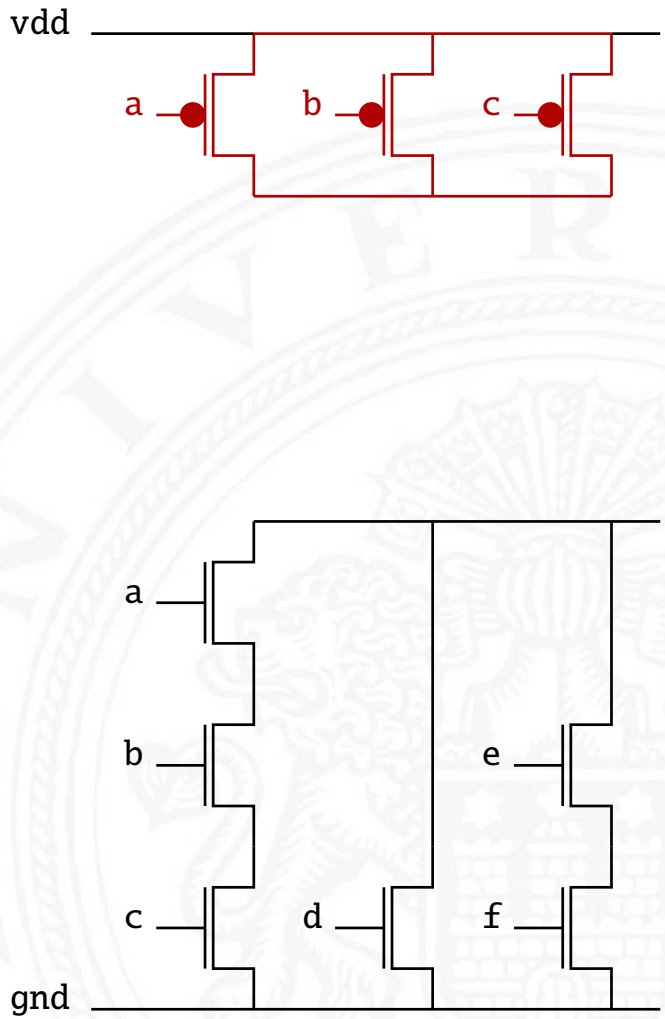
CMOS-Komplexgatter

Beispiel: $\overline{(a \wedge b \wedge c) \vee d \vee (e \wedge f)}$



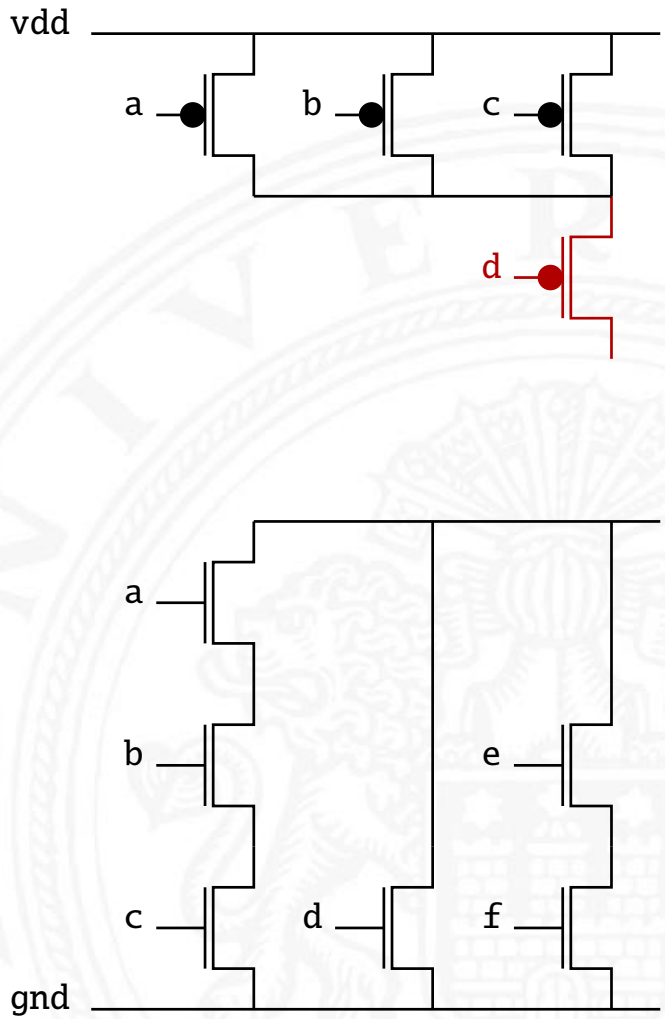
CMOS-Komplexgatter

Beispiel: $\overline{(a \wedge b \wedge c) \vee d \vee (e \wedge f)}$



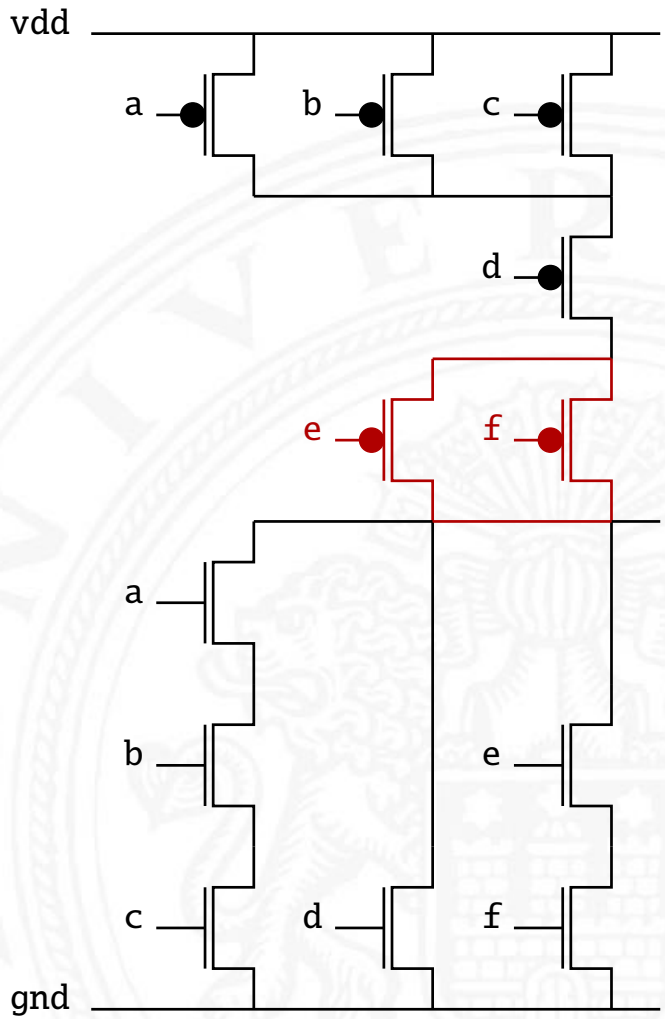
CMOS-Komplexgatter

Beispiel: $\overline{(a \wedge b \wedge c) \vee d \vee (e \wedge f)}$



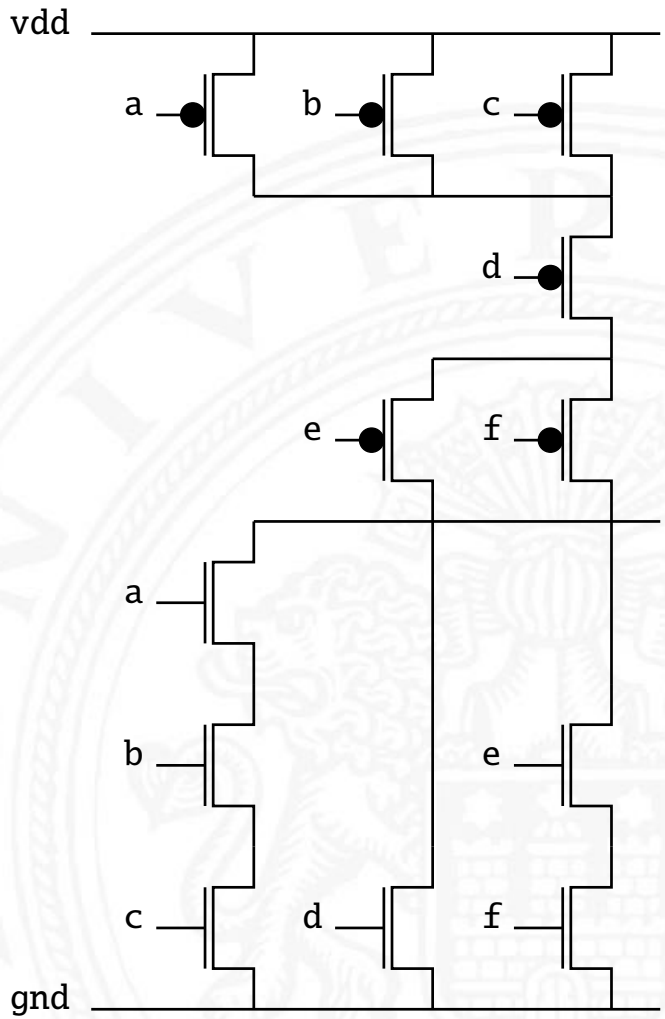
CMOS-Komplexgatter

Beispiel: $\overline{(a \wedge b \wedge c) \vee d \vee (e \wedge f)}$



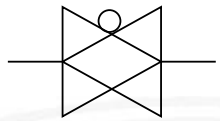
CMOS-Komplexgatter

Beispiel: $\overline{(a \wedge b \wedge c) \vee d \vee (e \wedge f)}$

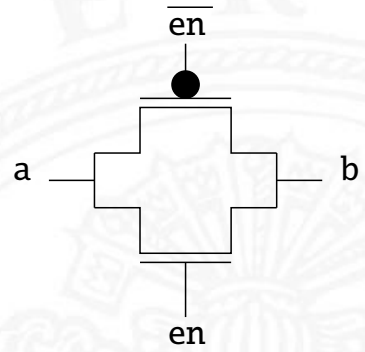


Transmission-Gates

Funktionsweise



- „Schalter“ in einer Leitung
- Ansteuerung mit *Enable*- + $\overline{Enable}$-Leitung
- Paar aus p- und n-Kanal Transistoren

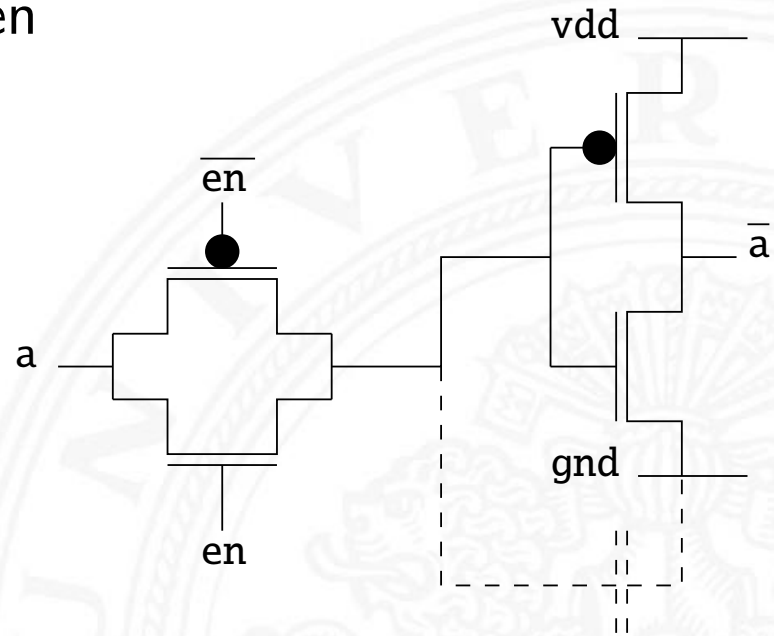


Enable	$Trans_P$	$Trans_N$	Wirkung
$en = 0 \rightarrow$ sperrt	/sperrt $\rightarrow$	Verbindung	getrennt
$en = 1 \rightarrow$ leitet	/leitet $\rightarrow$	–"–	geschlossen
- Äquivalent bei NMOS-Technik: *Pass-Transistor*

Transmission-Gates (cont.)

Schaltungen

- Speicherung auf Gate-Kapazitäten
- „quasi“-statische Latches

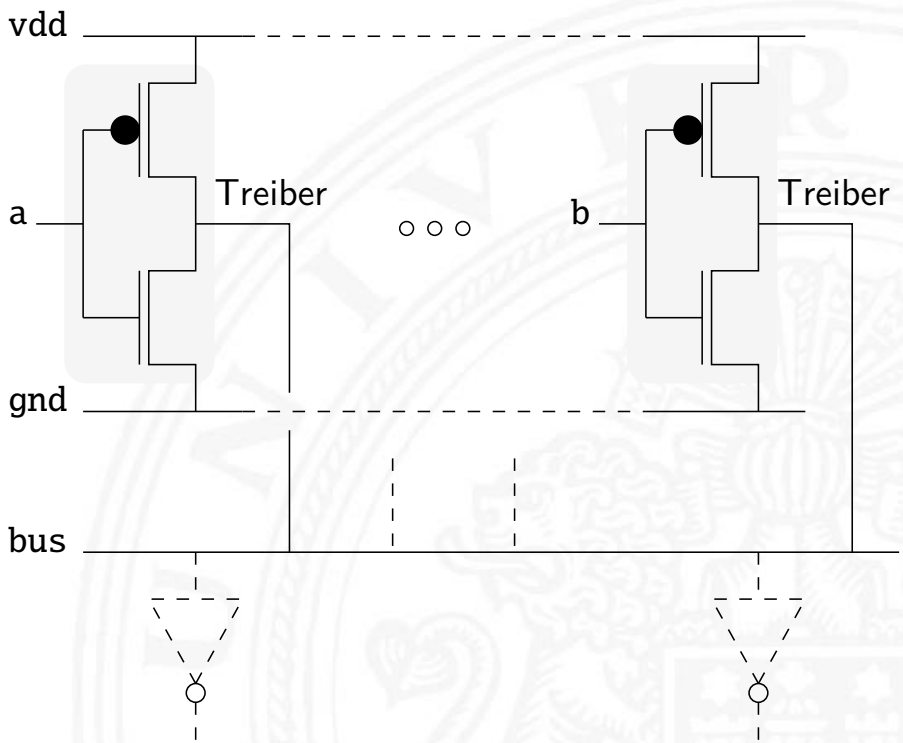


Tristate-Treiber

Bussysteme

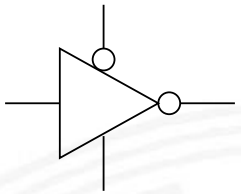
- Quellen: „Bustreiber“
- Senken : Gattereingänge
- Probleme
  - *Kurzschluss*
  - *offene Eingänge*

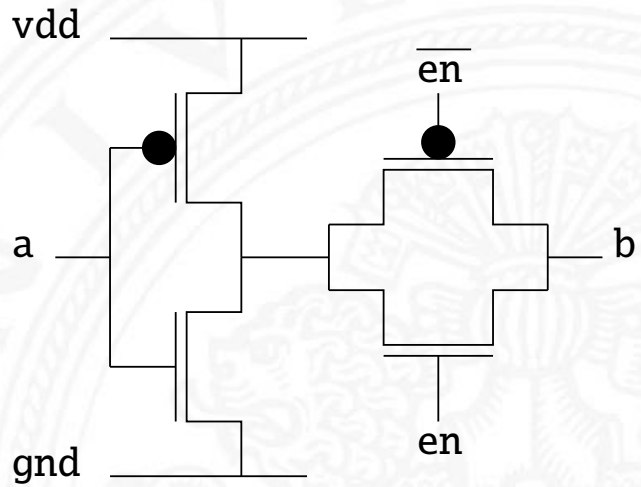
$\Rightarrow$ Tristate



Tristate-Treiber (cont.)

Beispiel: Tristate-Inverter





Funktionsweise

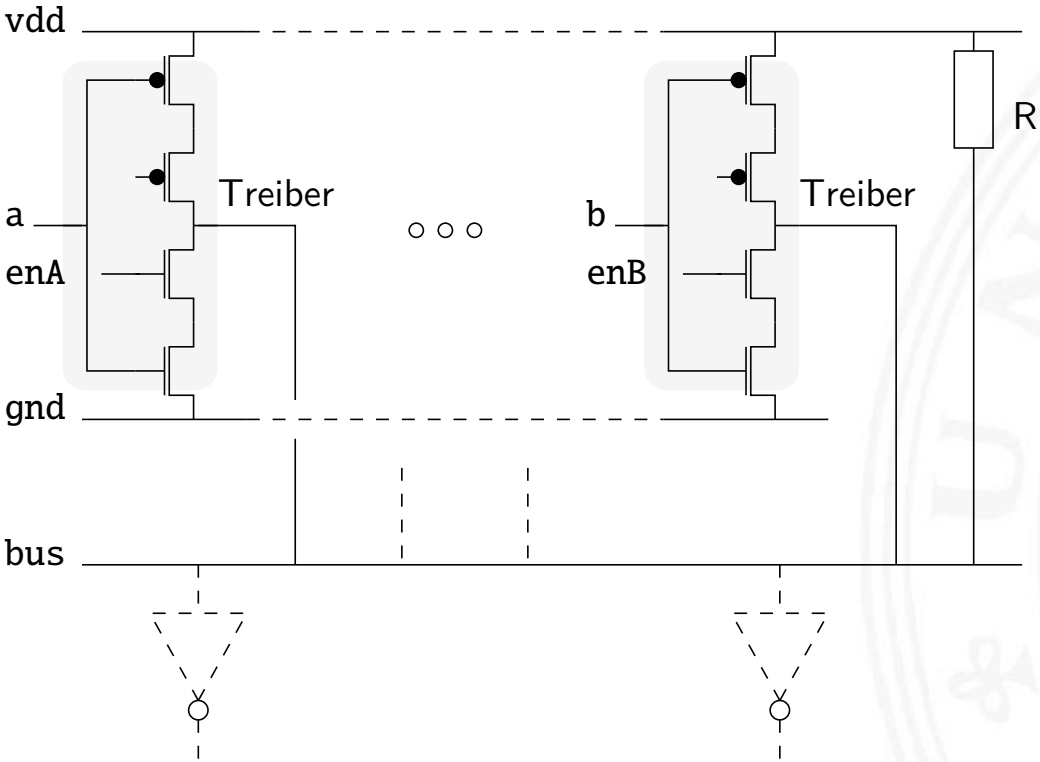
- Ausgang elektrisch trennen
  z.B. mit Transmission-Gate
- 3-Pegel: 0, 1, Z *hochohmig*

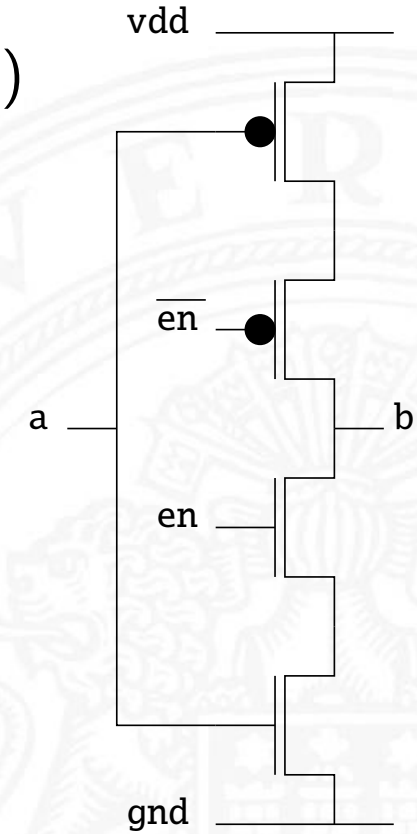
Enable	Verbindung	Ausgang	
$en = 0$	$\rightarrow$ getrennt	$\rightarrow bus = Z$	hochohmig
$en = 1$	$\rightarrow$ geschlossen	$\rightarrow bus = \neg a$	$f(a)$

Tristate-Treiber (cont.)

Tristate-Bussystem

- *pull-up/-down* Widerstand R (offene Eingänge)
- nur genau ein Treiber gleichzeitig aktiv





parasitäre Effekte

Reihenschaltung von Transistoren

- Innenwiderstand leitender Transistoren: $\approx 1\ k\Omega$
- Ausgang schaltet durch Widerstand langsamer um

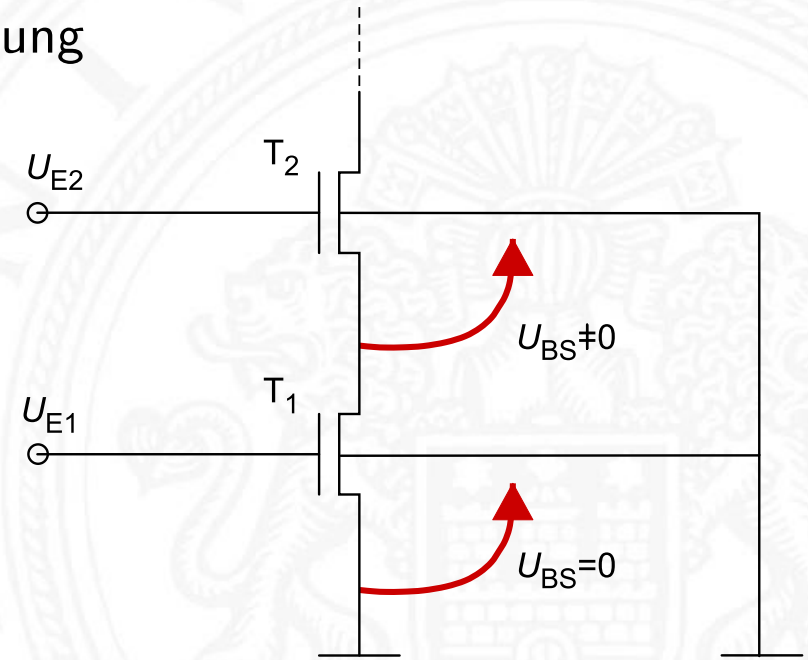
$\Rightarrow$ Reihenschaltung von 3 bis max. 4 Transistoren

$\Rightarrow$ (Komplex-) Gatter mit entsprechend wenig Eingängen
ggf. ist mehrstufige Logik schneller, z.B. 32-bit NOR für $= 0$

parasitäre Effekte (cont.)

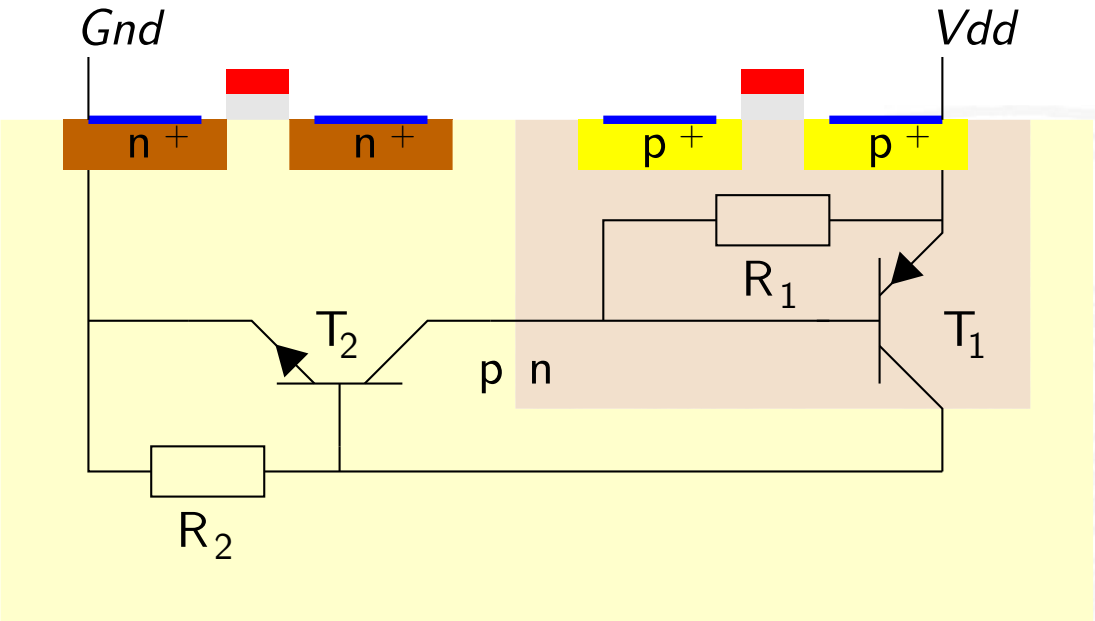
Body Effekt / Substrat Effekt

- Source T_P an Vdd = Substrat $\Rightarrow U_{BS} = 0$
  –"– T_N an Gnd –"–
- gilt *nicht* für
  - „innere“ Transistoren in Reihenschaltung
  - Transmission-Gates. . .
- Auswirkung
  - vergrößerte Raumladungszone: Source-Substrat
  - die Schwellspannung U_P wächst
  - im dynamischen Betrieb wird der Transistor langsamer



parasitäre Effekte (cont.)

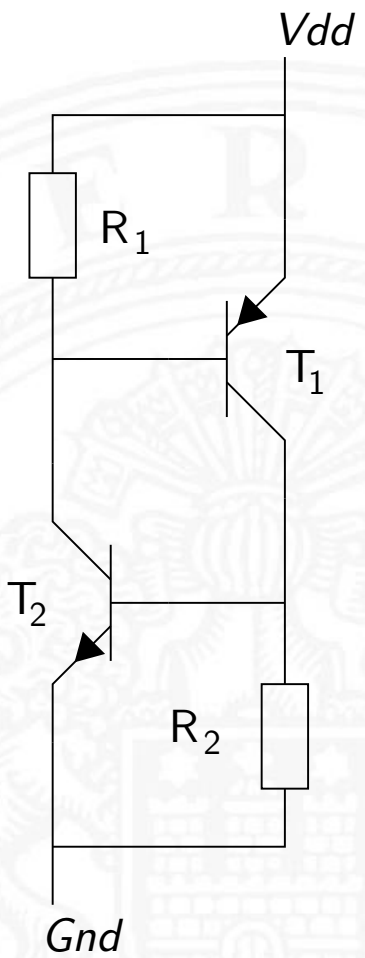
Latch-up Effekt



- parasitäre Bipolar-Transistoren: Diffusion, Substrat, Wanne
- pnp-Transistor T_1 leitet, wenn $U_{BE} \geq 0,6\,V$
  npn-Transistor T_2 –"–

parasitäre Effekte (cont.)

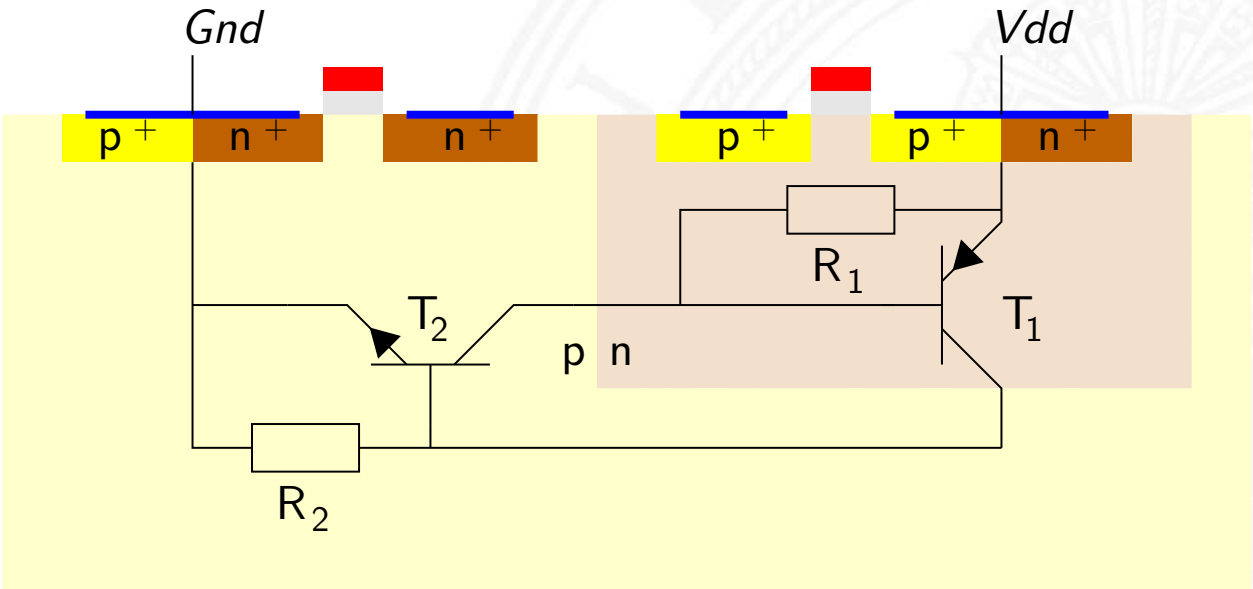
- Latch-up
  1. Spannungsabfall über R_1 oder R_2 schaltet T_1 bzw. T_2 ein
  2. Der leitende Transistor „zündet“ den Zweiten
  3. Beide Transistoren sind leitend
  4. Kurzschluss
- Auslöser: Spannungschwankung



parasitäre Effekte (cont.)

- Gegenmaßnahmen
  - Kleiner Verstärkungsfaktor von T_1, T_2 $\Rightarrow$ Prozessparameter
  - Räumliche Trennung der p- und n-Transistoren $\Rightarrow$ Layout
  - Widerstände R_1 und R_2 möglichst klein machen

  $\Rightarrow$ Wannen- und Substratkontakte
  Ausgangstreiber: komplette Ringe um Transistoren („guard-ring“)



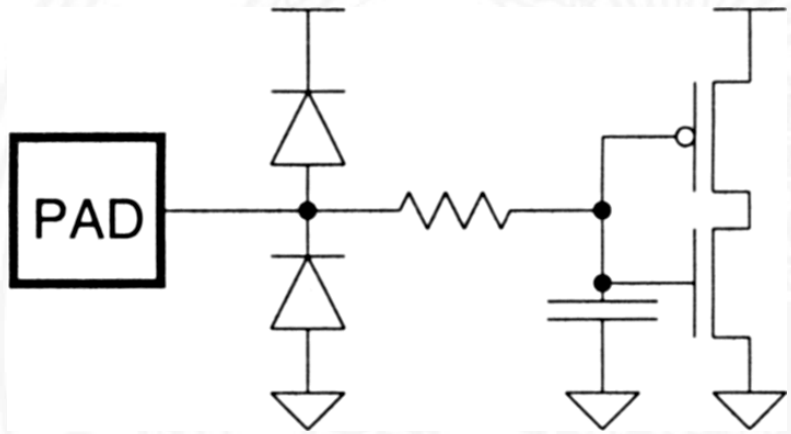
parasitäre Effekte (cont.)

CMOS-Eigenschaften empfindlich gegenüber

1. offenen Eingängen
   - „floating gate“, Potenzial $\approx \frac{1}{2} Vdd$
   - p- und n-Kanal Transistoren leiten

   $\Rightarrow$ richtige Beschaltung
2. elektrostatischen Überspannungen
   ESD (**E**lectro**s**tatic **D**ischarge)

   $\Rightarrow$ Eingangsschutzschaltungen



Entwurfsregeln

1. Geometrische Regeln für das Layout von Schaltungen
   - Minimal- / Maximalgrößen von Strukturen
   - Abstandsregeln
   - meist als Abhängigkeiten über mehrere „Layer“ abgeleitet
2. Elektrische Regeln über die Verschaltung von Elementen
   (keine Entwurfsregeln im engeren Sinne)

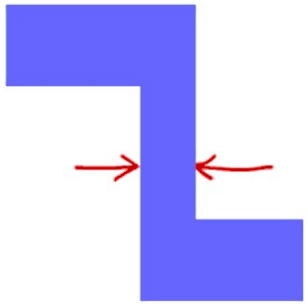
- Prozessspezifisches „Know-How“ der Chiphersteller
- Beschrieben durch Listen und Grafiken
- Überprüfung beim Layout durch Werkzeuge:
  DRC (**D**esign **R**ule **C**heck)

▸ Liste ▸ Grafik

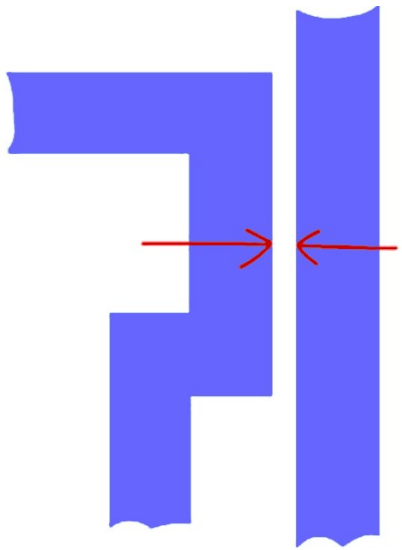
Entwurfsregeln (cont.)

Gründe für Entwurfsregeln

1. Elektrische Randbedingungen bedingen geometrische Merkmale



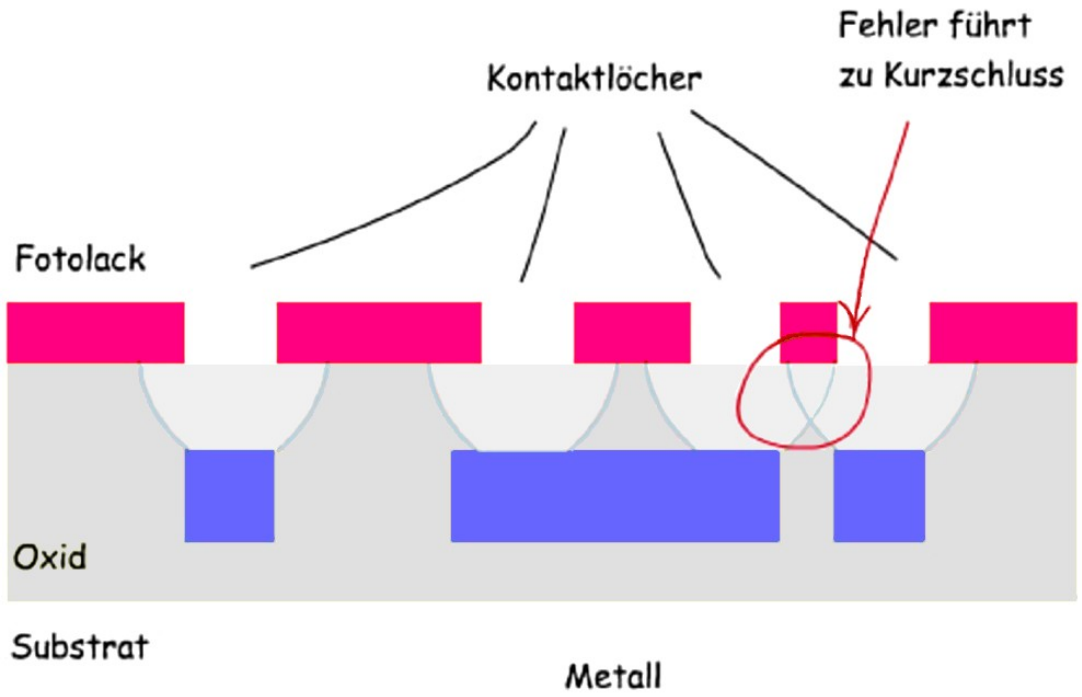
Leitbahnabmessungen zu gering
-> Stromdichteregel verletzt



Mindestabstand unterschritten
-> Übersprechregel verletzt

Entwurfsregeln (cont.)

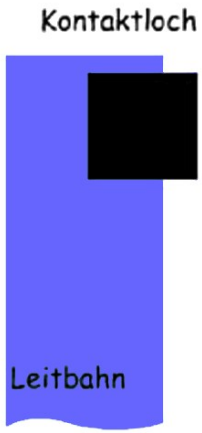
2. (Herstellungs-) Prozessbedingte Regeln



Entwurfsregeln (cont.)

3. Justiertoleranzen von Masken

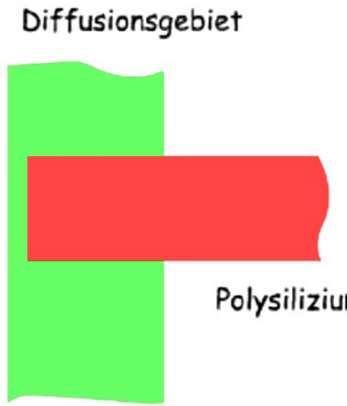
Innenlage von Kontakten



Justiertoleranz:
Max. Verschiebung von Masken

-> Mindestabstand Kontaktloch zu Leitbahnkante muss eingehalten werden.

Mindestüberlappungen



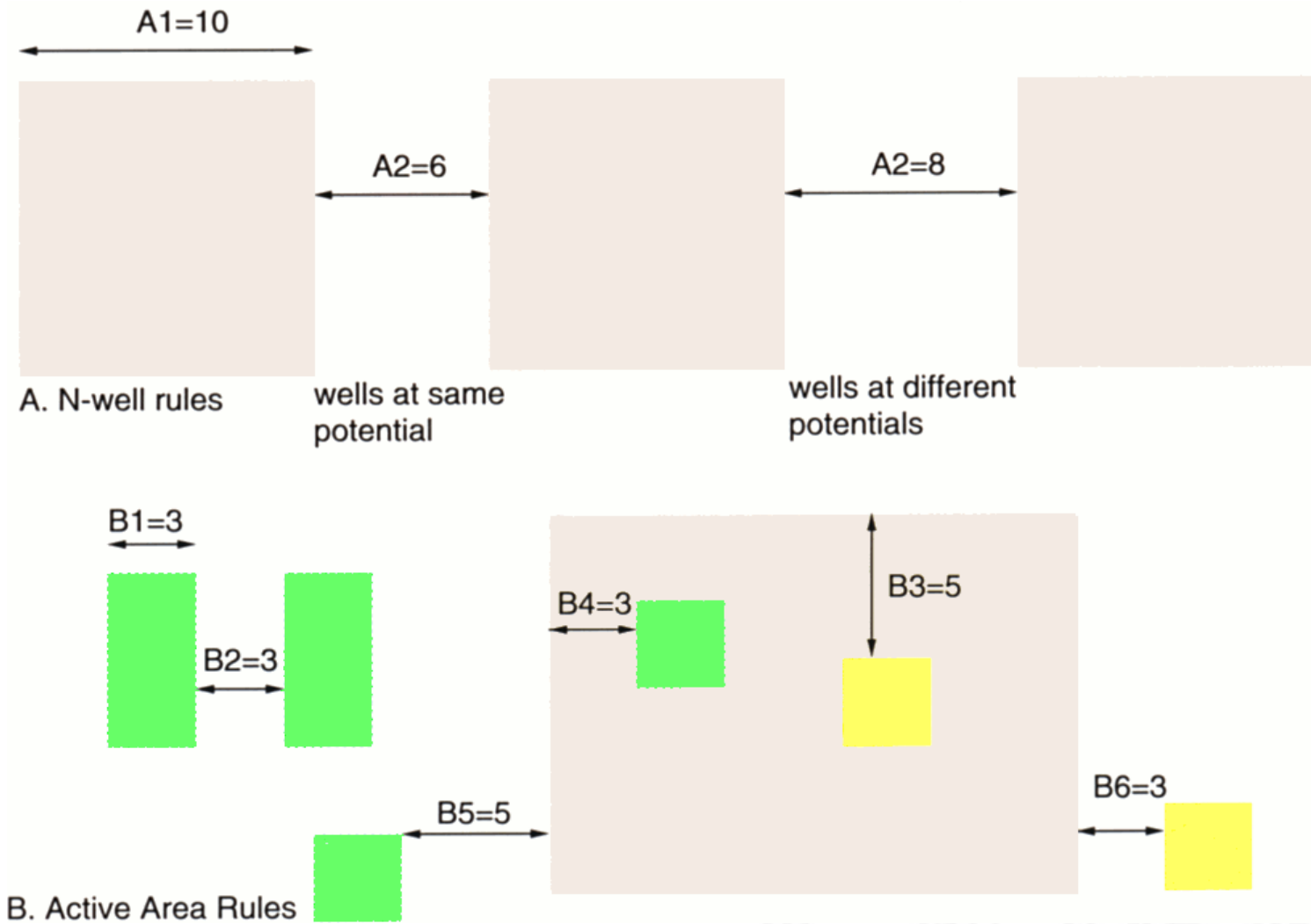
-> Mindestüberlappung Polysilizium zu Diffusion muss eingehalten werden.

TABLE 3.2 CMOS Layout Rules

	λ RULE	λ/μ RULE (0.5μ)	μ RULE
A. N-well layer			
A.1 Minimum size	10λ	5μ	2μ
A.2 Minimum spacing (wells at same potential)	6λ	3μ	2μ
A.3 Minimum spacing (wells at different potentials)	8λ	4μ	2μ
B. Active Area			
B.1 Minimum size	3λ	1.5μ	1μ
B.2 Minimum spacing	3λ	1.5μ	1μ
B.3 N-well overlap of p^+	5λ	2.5μ	1μ
B.4 N-well overlap of n^+	3λ	1.5μ	1μ
B.5 N-well space to n^+	5λ	2.5μ	5μ
B.6 N-well space to p^+	3λ	1.5μ	3μ

zurück

[WE94]


A1=10
A2=6
A2=8
A. N-well rules
wells at same potential
wells at different potentials
B1=3
B2=3
B3=5
B4=3
B5=5
B6=3
B. Active Area Rules

[WE94]

Schaltnetze

alle zuvor skizzierten Schaltungen

- Kombinatorische Logik aus Gattern
- Boole'sche Algebra / Schaltalgebra
- Realisiert Funktionen: $f(X) = Y$
  beliebig kompliziert

– damit kann aber noch nicht „gerechnet“ werden:
  die *Speicherung* von Werten fehlt

Schaltnetze (cont.)

Schaltalgebra

- Boole'scher Verband $(B; \sqcap, \sqcup)$
  - Kommutativgesetze
    $a \sqcap b = b \sqcap a$ $\quad a \sqcup b = b \sqcup a$
  - Distributivgesetze
    $(a \sqcap b) \sqcup c = (a \sqcup c) \sqcap (b \sqcup c)$ $\quad (a \sqcup b) \sqcap c = (a \sqcap c) \sqcup (b \sqcap c)$
  - Null- und Einselement
    $\forall a \in B \; \exists \mathbf{0}, \mathbf{1} \in B :$ $\quad a \sqcup \mathbf{0} = a \quad a \sqcap \mathbf{1} = a$
  - Komplement
    $\forall a \in B \; \exists a' \in B :$ $\quad a \sqcap a' = 0 \quad a \sqcup a' = 1$

Schaltnetze (cont.)

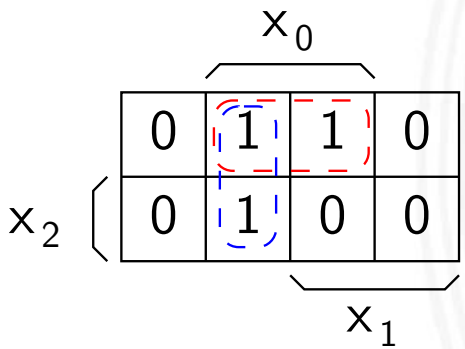
- Minterm: Konjunktion aller n-Variablen
  $m = x_1' \wedge x_2' \ldots \wedge x_n', x_i' \in x_i, \overline{x_i}$
  Maxterm: Disjunktion aller n-Variablen
  $M = x_1' \vee x_2' \ldots \vee x_n', x_i' \in x_i, \overline{x_i}$
- Disjunktive Normalform: Disjunktion von Mintermen
  $f = \bigvee m_u, u \in f^{-1}(\mathbf{1})$
  Konjunktive Normalform: Konjunktion von Maxtermen
  $f = \bigwedge M_u, u \in f^{-1}(\mathbf{0})$
- Beschreibung von Schaltfunktionen
  - Schaltalgebraischer Ausdruck
  - Funktionstabelle
  - KV-Diagramm

Schaltnetze (cont.)

Funktionstabelle

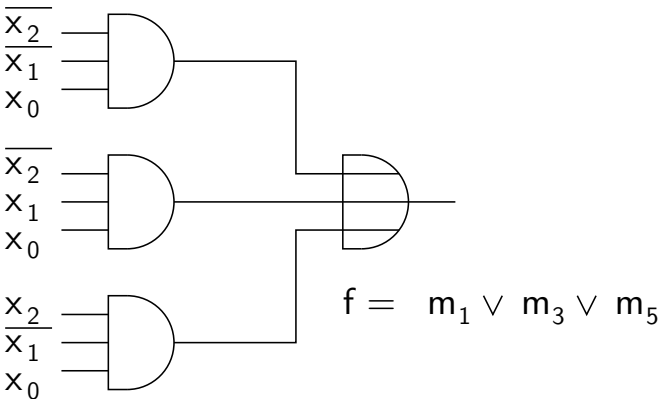
$x_2 x_1 x_0$	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	f
0 0 0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
0 0 1	0	1	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1
0 1 0	0	0	1	0	0	0	0	0	1	1	0	1	1	1	1	1	0
0 1 1	0	0	0	1	0	0	0	0	1	1	1	0	1	1	1	1	1
1 0 0	0	0	0	0	1	0	0	0	1	1	1	1	0	1	1	1	0
1 0 1	0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1
1 1 0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	0	1	0
1 1 1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0

KV-Diagramm



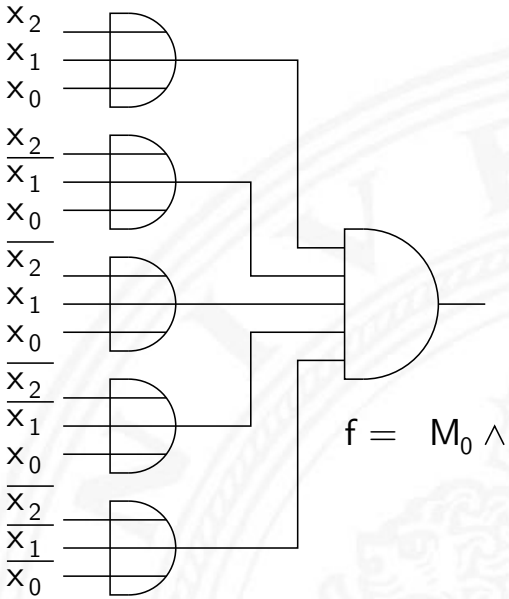
Schaltnetze (cont.)

Disjunktive Normalform



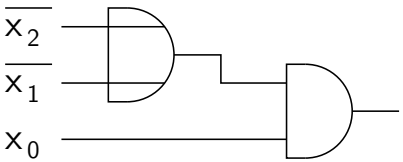
$$f = m_1 \vee m_3 \vee m_5$$

Konjunktive Normalform

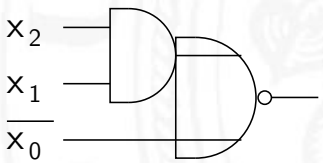


$$f = M_0 \wedge M_2 \wedge M_4 \wedge M_6 \wedge M_7$$

Gatterrealisierungen



$$f = x_0 \wedge (\overline{x_1} \vee \overline{x_2})$$



$$f = \overline{\overline{x_0} \vee (x_1 \wedge x_2)}$$

Schaltwerke

Speichernde Elemente

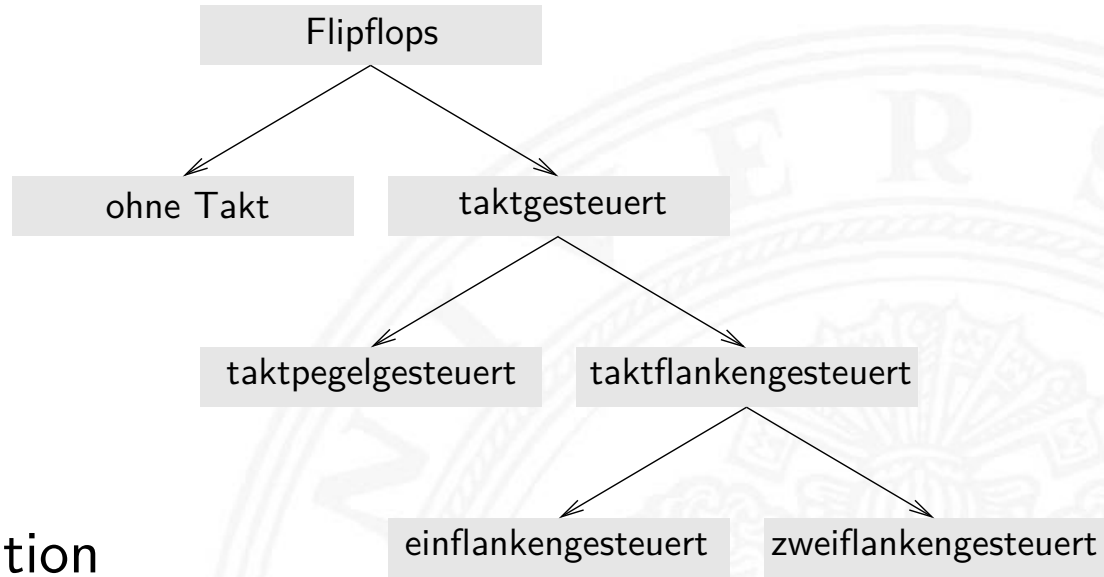
- Register in Steuer- und Operationswerk, Ein-/Ausgabe ...
- Implementation arithmetischer Pipelines
- Speichern interner Zustände von Automaten

siehe dazu

- `http://de.wikipedia.org/wiki/Flipflop`
- `http://tams.informatik.uni-hamburg.de/research/e-learning/applets/hades/webdemos`
- `http://www-ihs.theoinf.tu-ilmenau.de/~sane/projekte/flipflop/flipflop.html`

Schaltwerke (cont.)

Klassifikation anhand



- Funktion ⇒ Typen
- Implementation / Schaltungstechnik

Methoden der Implementation

1. statisch
   - Speicherung: Rückkopplung von Gattern
   + taktunabhängig
   + sicher

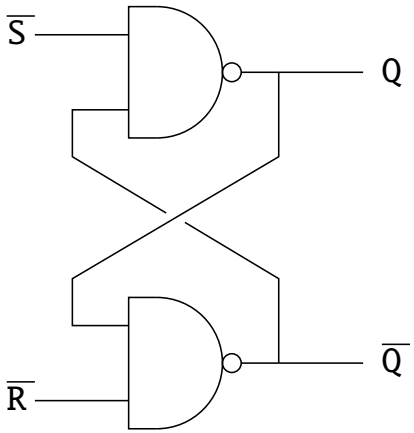
Schaltwerke (cont.)

2. quasi-statisch
   - Speicherung: Rückkopplung von Gattern
   - Transmission-Gates als Multiplexer
   + taktunabhängig
   + kleiner
3. dynamisch
   - Speicherung: Gate-Kapazitäten
   - verschiedene Taktschemata/Schaltungsvarianten
   – muss getaktet werden
   – schwieriger zu Entwerfen (wegen Taktschema)
   + Integration in Datenpfade (arithmetische Pipelines)
   + sehr hohe Taktfrequenzen
   + sehr klein

Latch- / Flipflop-Typen

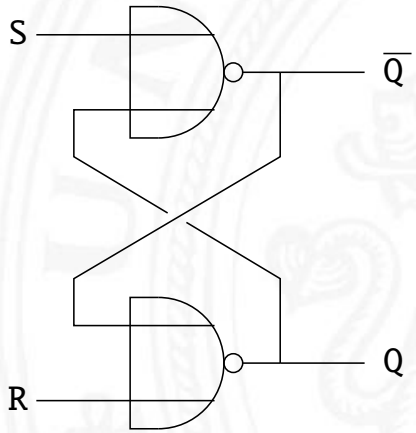
RS-Flipflop, ohne Takt, statisch

- direktes **S**etzen / **R**ücksetzen
- Rückgekoppelte Gatter: Speicherung in allen statischen FFs
- unzulässige Eingangskombinationen



NAND-Form

$\overline{S}$	$\overline{R}$	Q
0	0	verboten
0	1	1
1	0	0
1	1	Q



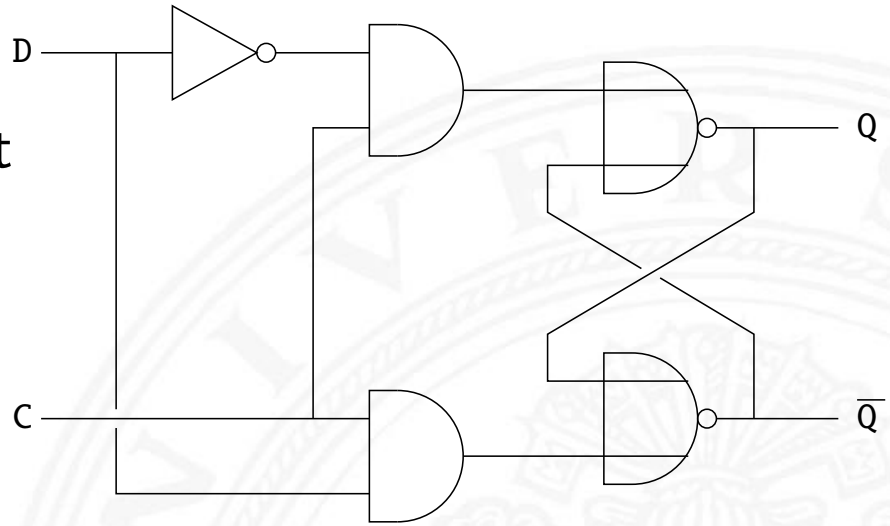
NOR-Form

S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	verboten

Latch- / Flipflop-Typen (cont.)

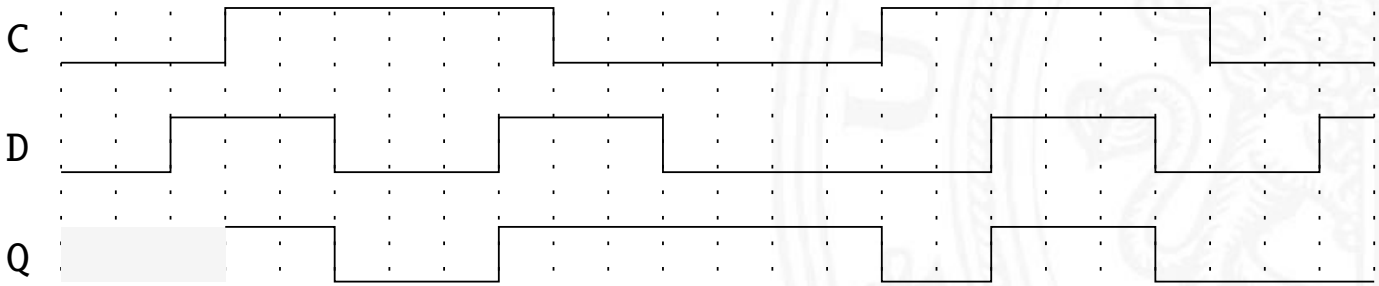
Latch, statisch

- Taktpegelgesteuert
  Transparent bei aktivem Takt
  Speicherung bei inaktivem –"–



C	Q
0	Q
1	D

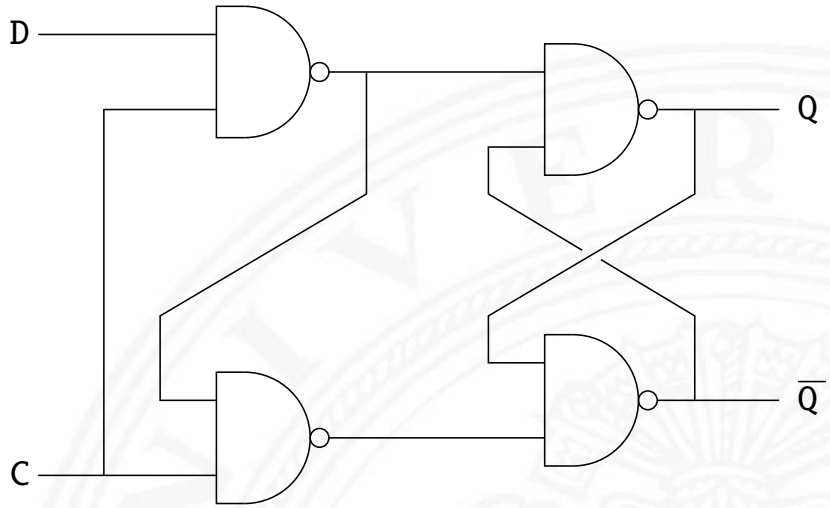
  high-aktiv



Latch- / Flipflop-Typen (cont.)

- Typen: high- / low-aktiv
- viele Schaltungsvarianten

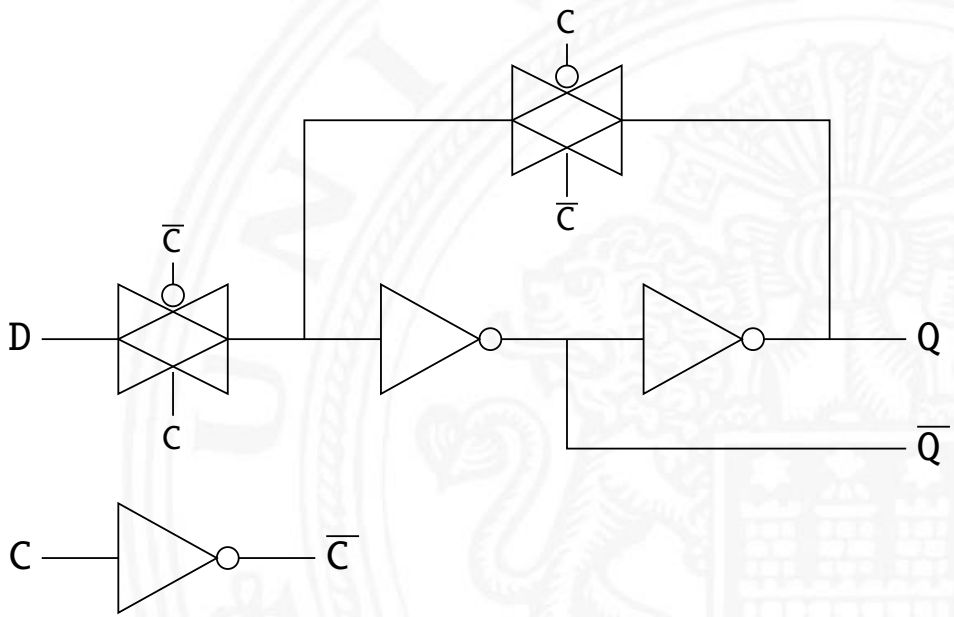
Latch, high-aktiv



Latch- / Flipflop-Typen (cont.)

Latch, quasi-statisch

- Transmission-Gates schalten
  Transparent: Eingang über die Inverter zum Ausgang
  Speicherung: Rückkopplungspfad –"–



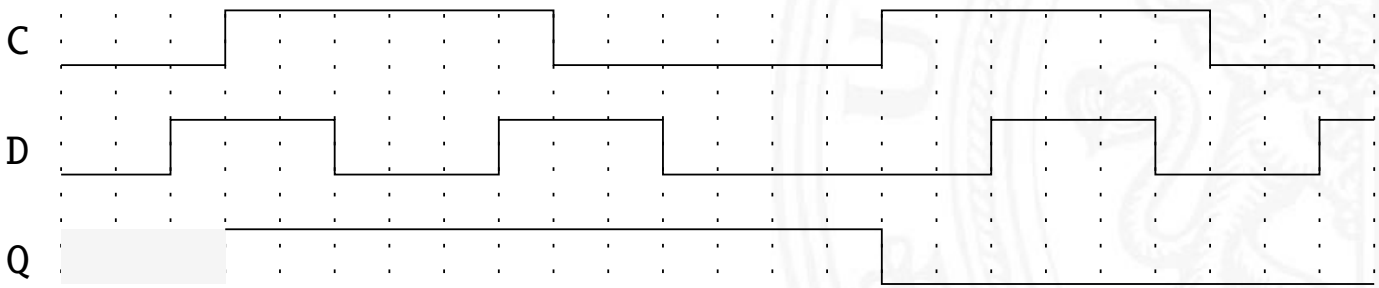
Latch, high-aktiv

Latch- / Flipflop-Typen (cont.)

D-Flipflop, statisch

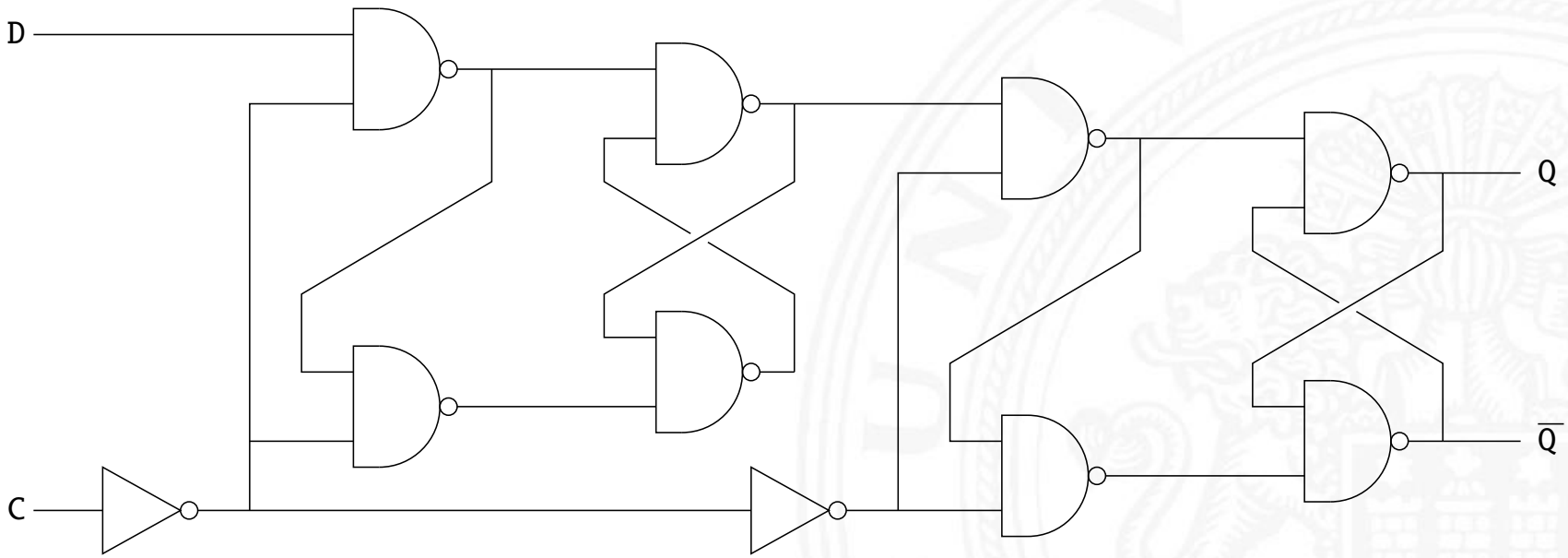
- Taktflankengesteuert: Speichert Eingang bei aktiver Flanke
- Detektion der Flanke
  - low- und high-aktives Latch hintereinandergeschaltet
  - Reihenfolge bestimmt Art der Flanke
- vorderflankengesteuert

C	Q
$0, 1, \downarrow$	Q
$\uparrow$	D

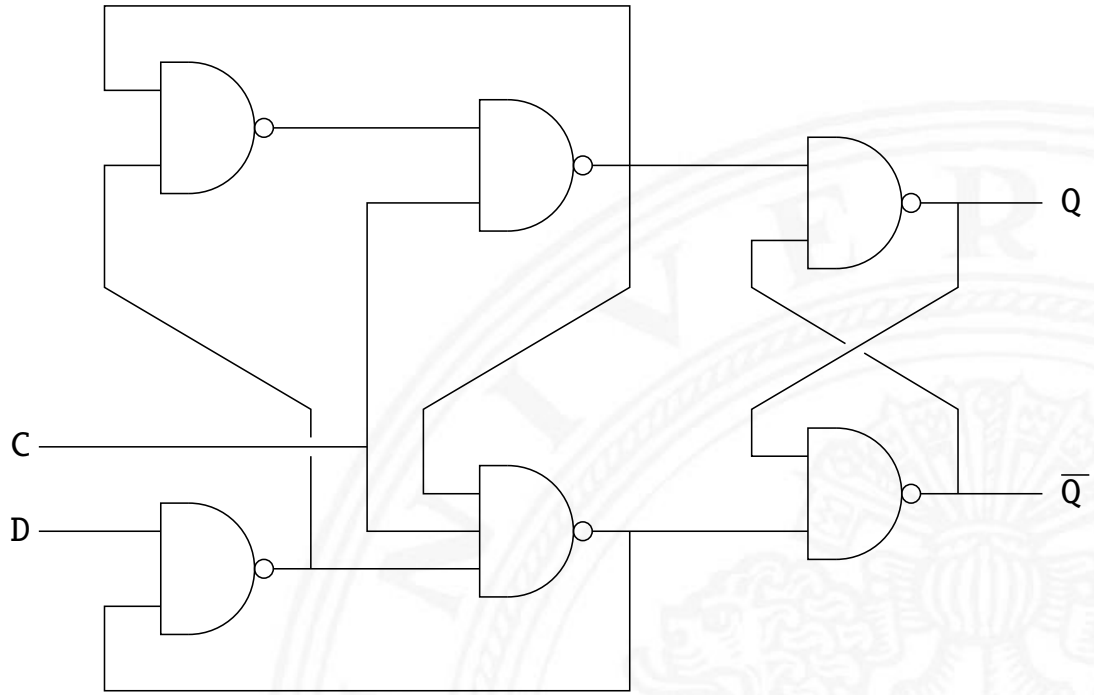


Latch- / Flipflop-Typen (cont.)

- Typen: vorder- / rückflankengesteuert
- viele Schaltungsvarianten



Latch- / Flipflop-Typen (cont.)



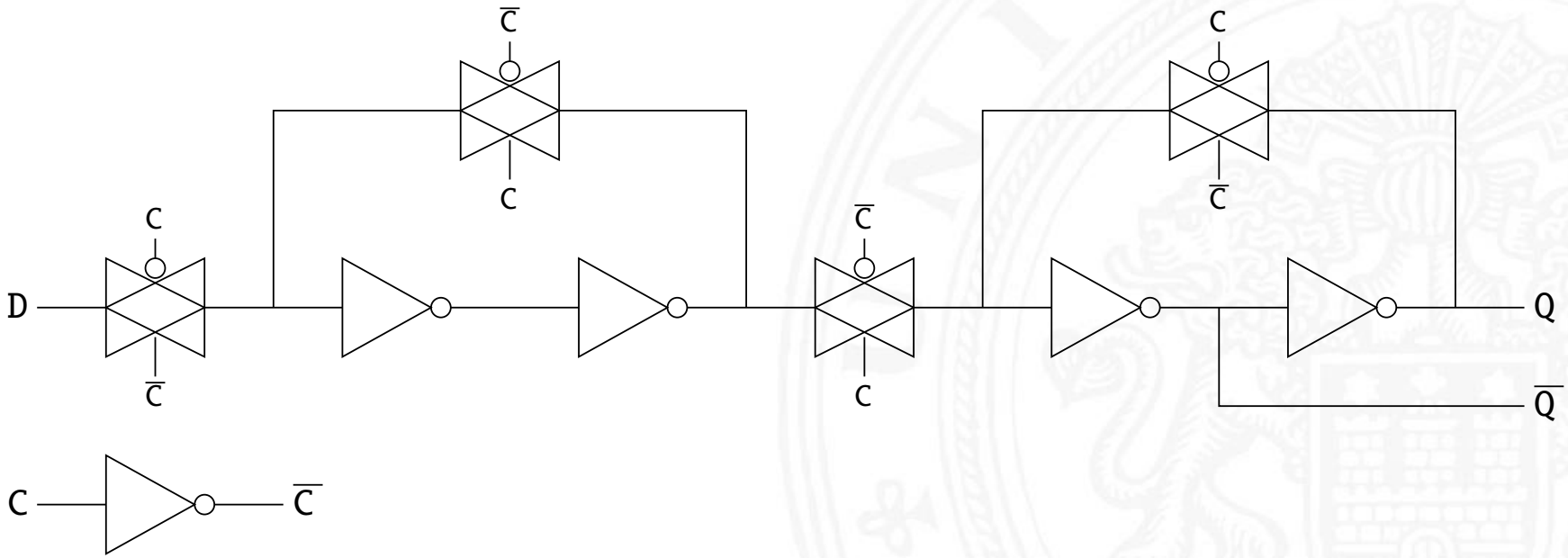
D-FF, vorderflankengest.

Latch- / Flipflop-Typen (cont.)

D-Flipflop, quasi-statisch

- Aufbau aus zwei Latches
- Vorderflanke: low-Transparent + high-Transparent

D-FF, vorderflankengest.



Latch- / Flipflop-Typen (cont.)

Zweiflankensteuerung

- die eine Taktflanke legt Abtastzeitpunkt für D fest
  die andere –"– schaltet den Wert auf den Ausgang Q
- Realisierung: Hintereinanderschaltung von drei Latches

viele weitere Flipfloptypen

- D-Flipflops mit synchronen/asynchronen Setz-/Rücksetz-Eingängen
- T-Flipflop (**T**oggle)

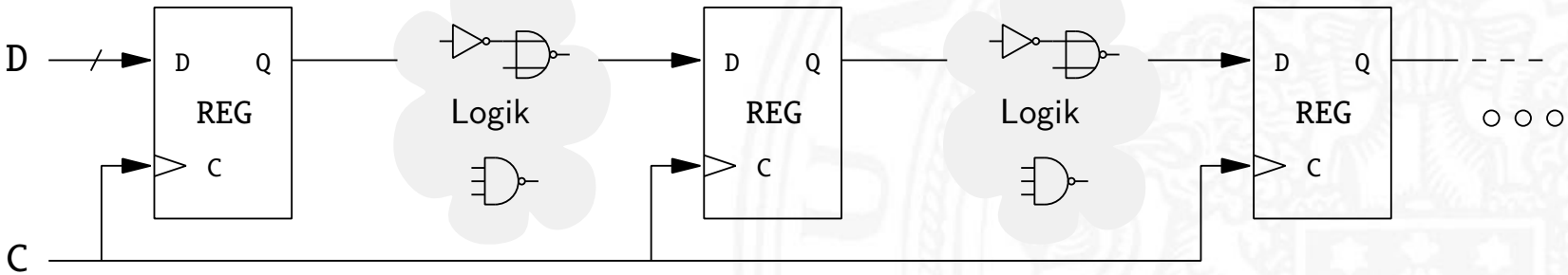
T	Q
0	Q
1	$\overline{Q}$

JK-Flipflop

J	K	Q
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Dynamische Latches / Flipflops

- Information auf Gate-Kapazitäten speichern
- zwei Funktionsprinzipien
  - einfache Speicherung auf Gate-Kapazitäten
  - „Preload“-Verfahren / dynamische Schaltungen
- *muss* getaktet werden (Selbstentladung)
- Integration in Pipeline-Datenpfade



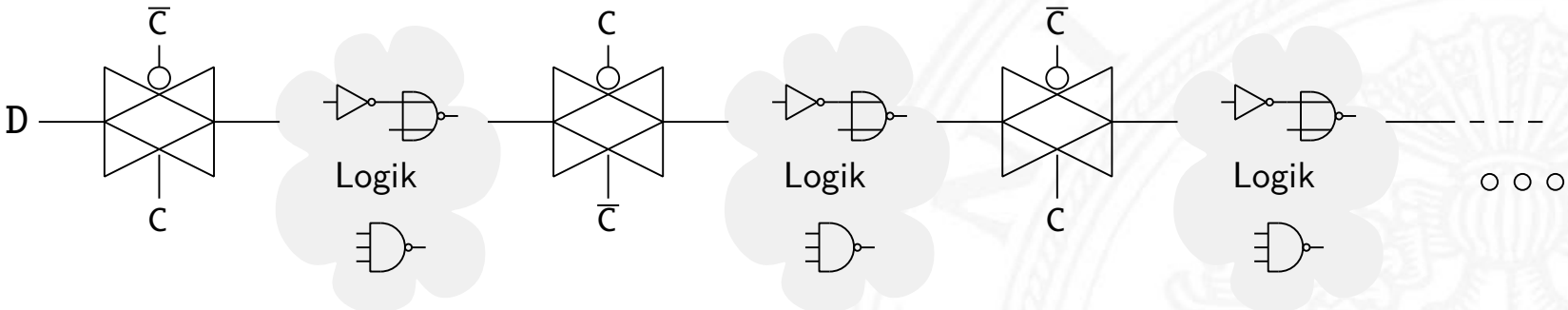
- viele verschiedene Schaltungstechniken

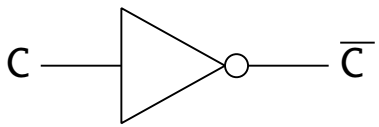
Dynamische Latches / Flipflops (cont.)

Direkte Speicherung auf Gate-Kapazitäten

- Gates mit dem zu speichernden Pegel auf-/entladen
- Eingänge abkoppeln

$\Rightarrow$ Speicherung





Dynamische Latches / Flipflops (cont.)

Probleme

Die folgende Aussagen gelten so oder ähnlich für alle dynamischen und oft auch für statische Schaltungen, sie bezeichnen generelle Anforderung an schnell getaktete Systeme

- Taktgenerierung: C und $\overline{C}$ müssen *gleichzeitig* umschalten
  Gegenseitiges Sperren aufeinanderfolgender Stufen
- Taktflanken müssen sehr steil sein
- Takte dürfen nicht „langsamer“ als die Daten sein
- Takte sollten überall möglichst gleichzeitig ankommen
  Taktverschiebung (*Clock-Skew*) möglichst klein

$\Rightarrow$ Randbedingungen im Layout berücksichtigen!

Dynamische Latches / Flipflops (cont.)

Dynamische Logik

- zwei Betriebs- bzw. Taktphasen
  1. Aufladen: Gates über getaktete Stufen auf-/entladen
  2. Auswerten: Gates entsprechend der Funktion ent-/aufladen
- Eigentlich eine „alte“ Schaltungstechnik, z.B. schnelle Carry-Generierung in Addierern:
  - Annahme: es gibt einen Carry
  - Aufladen: jeweils Carry = 1
  - Auswerten: Prüfung, ob auch wirklich ein Carry erzeugt wird

  lokale Entscheidung:

a	b		c_{out}
0	0	kein Carry möglich	entladen
0	1	warten	$= c_{in}$
1	0	–"–	
1	1	es gibt einen Carry	fertig

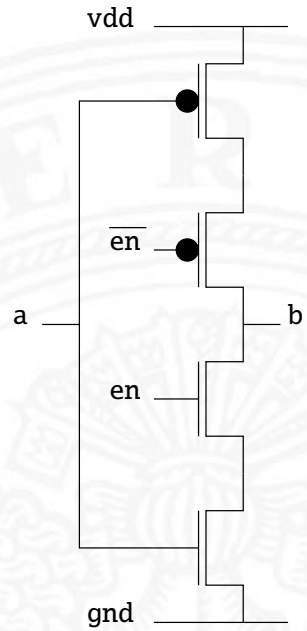
Dynamische Latches / Flipflops (cont.)

Dynamische Speicherschaltungen

- Logik: Funktionsprinzip *ohne* Speicherung
- Erweiterung durch „abschaltbaren Ausgang“
  - Varianten: Transmission Gate
    Tristate-ähnlicher Aufbau
  - Trennt während der Aufladephase den Ausgang ab

  $\Rightarrow$ ein „falscher Wert“ durch das Vorladen wird am Ausgang nicht sichtbar

  $\Rightarrow$ Speicherung während der Aufladephase



Dynamische Latches / Flipflops (cont.)

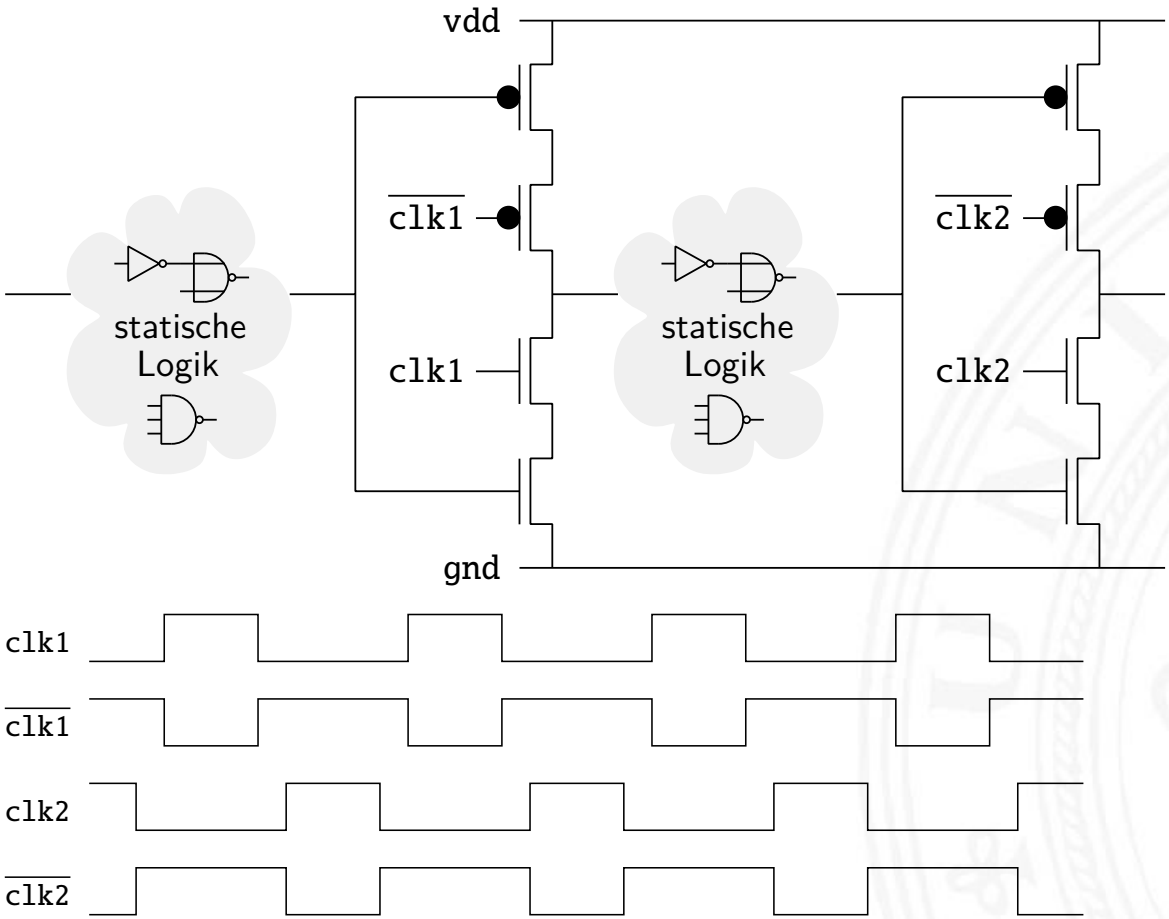
+ erlaubt sehr hohe Taktraten
+ Speicherung erfolgt quasi nebenbei
+ Ausnutzung der gesamten Taktperiode möglich
  - Betriebsphasen auf wahlweise beiden Clockpegeln möglich
  - Varianten:

Aufladephase	Auswertungsphase
Clock = 0	Clock = 1
Clock = 1	Clock = 0

  ⇒ Alternierende Abfolge in mehrstufigen arithmetischen Pipelines

Dynamische Latches / Flipflops (cont.)

C^2MOS Clocked CMOS



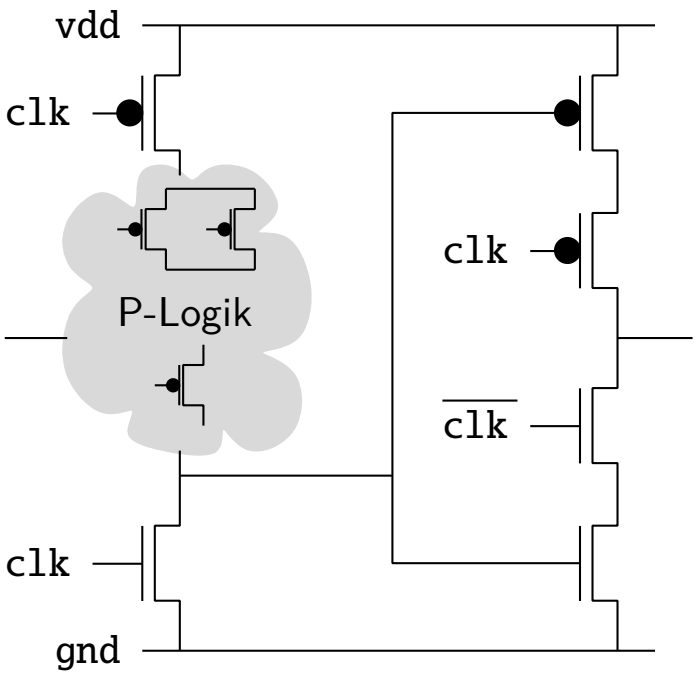
Dynamische Latches / Flipflops (cont.)

- dies ist die „älteste“ Schaltungstechnik.
- Funktionsweise
  - Speicherung auf Gates statisch aufgebauter Logik
  - „geordnetes“ Durchschieben der Werte durch disjunkte Takte
- Nachteile
  - 2 Takte: komplizierter Taktgenerator
  - 4 Taktleitungen: Probleme beim Layout
  - Totzeiten: niedrige Taktraten
- Probleme
  - das Taktschema muss genau eingehalten werden (Clock-Delay)

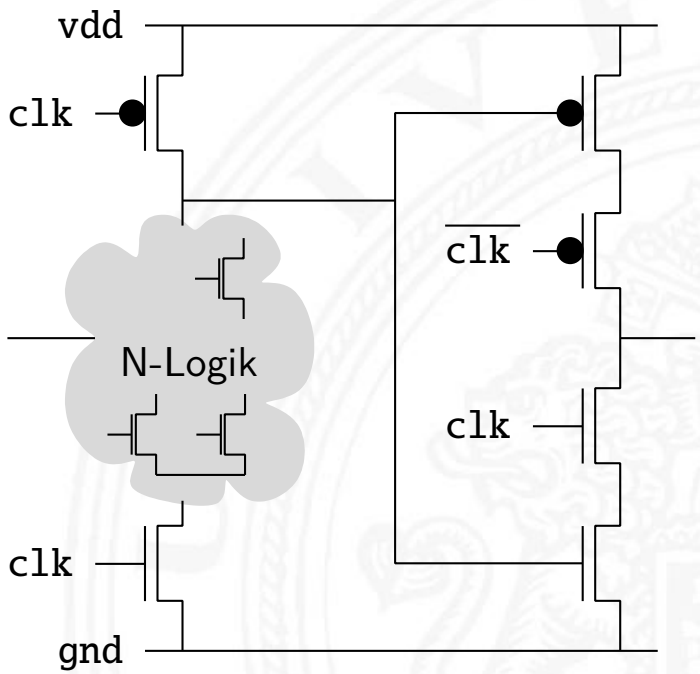
Dynamische Latches / Flipflops (cont.)

NORA **NO RA**ce 2-Phasen Takt

P-Stufe



N-Stufe



Dynamische Latches / Flipflops (cont.)

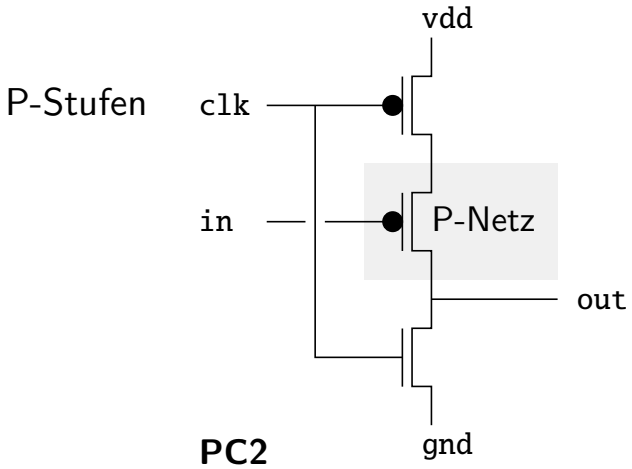
- Funktionsweise
  - Aufladephase
  - Auswertungsphase
  - Speicherelement von C^2MOS mit dynamischer Logik kombiniert
  - zwei Schaltungsvarianten für den Aufbau von Pipelines: P- und N-Stufen
- Vorteile
  - \+ dynamische Technik spart bei größeren Gatterfunktionen Transistoren ein $\Rightarrow$ weniger Fläche
  - \+ höhere Taktraten da keine Totzeiten auftreten
- Nachteile
  - – 2 Taktleitungen: Probleme beim Layout
- Probleme
  - – Vorder- und Rückflanke des Takts müssen „gleichzeitig“ wechseln

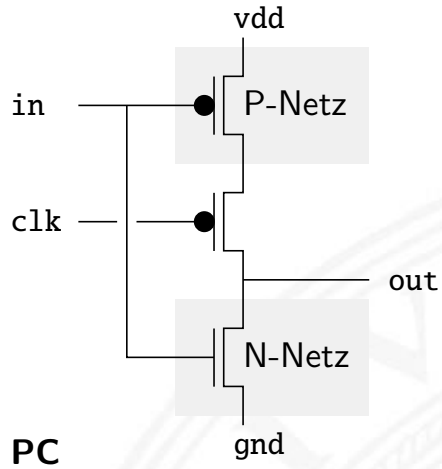
Dynamische Latches / Flipflops (cont.)

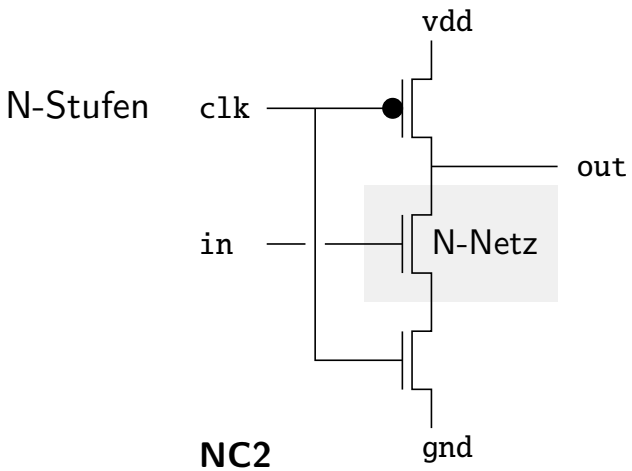
TSPC **T**rue **S**ingle **P**hase **C**lock

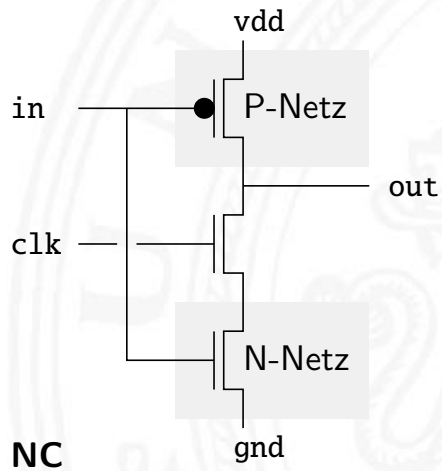
- Schaltungstechnik Anfang der 90er Jahre entwickelt
  Taktgeschwindigkeit $\times$ 4-5 (z.B.: DEC Alpha, PowerPC)
- Funktionsweise ▸ Aufladephase ▸ Auswertungsphase
  - Idee: die Transistoren mit invertiertem Takt können wegfallen
  - 4 Grundelemente für den Aufbau von Pipelines:
    P-Logik / N-Logik $\times$ Precharge-Stufe: 2 getaktete Transistoren / Ausgangsstufe: 1 getakteter Transistor
  - Kombinationsregeln, bzw. abwechselnde Anordnung

Dynamische Latches / Flipflops (cont.)









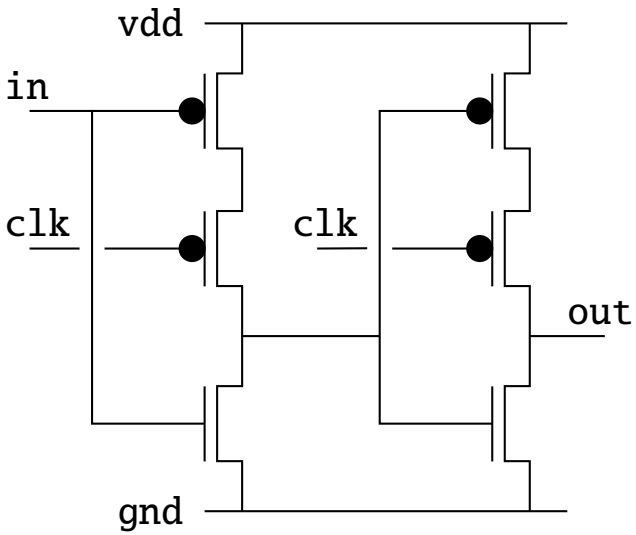
dynamische Logik

Ausgangsstufen

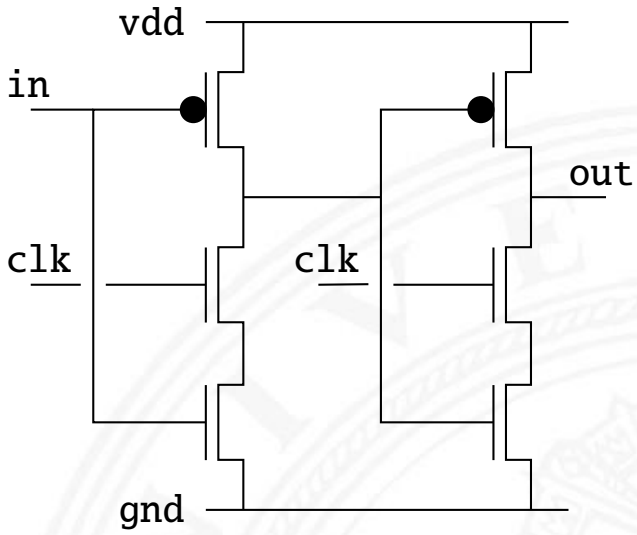
Dynamische Latches / Flipflops (cont.)

- Vorteile
  - \+ P- und N-Logik kann Gatterfunktion implementieren; die hier skizzierten Transistoren entsprechen Inverter
  - \+ keine Totzeiten, sehr hohe Taktraten ähnliche Techniken in allen aktuellen Prozessoren im Datenpfad
  - \+ es gibt nur eine Clock (-Leitung)
- Probleme
  - – sehr steile Flanken der Taktsignale
  - – der Takt muss bei allen getakteten Transistoren „gleichzeitig“ eintreffen (Stichwort: *Clock-Skew*)
  - – Kombinationsregeln in der Abfolge von P- und N-Stufen sind *genau* einzuhalten, insbesondere bei Rückkopplungen

Dynamische Latches / Flipflops (cont.)

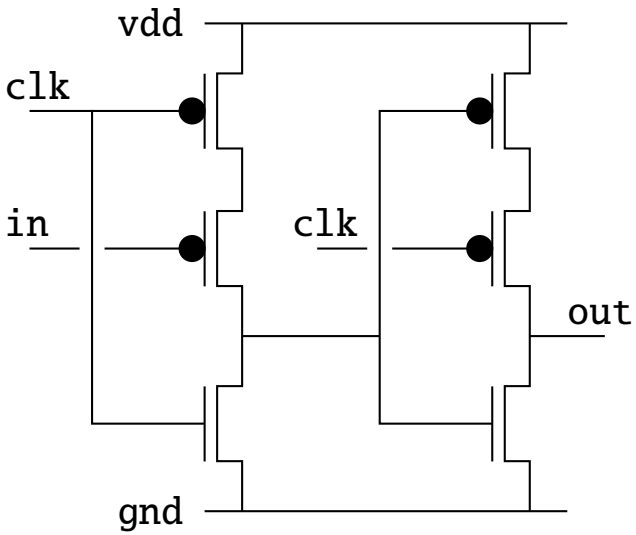


Latch - low transparent **PC-PC**

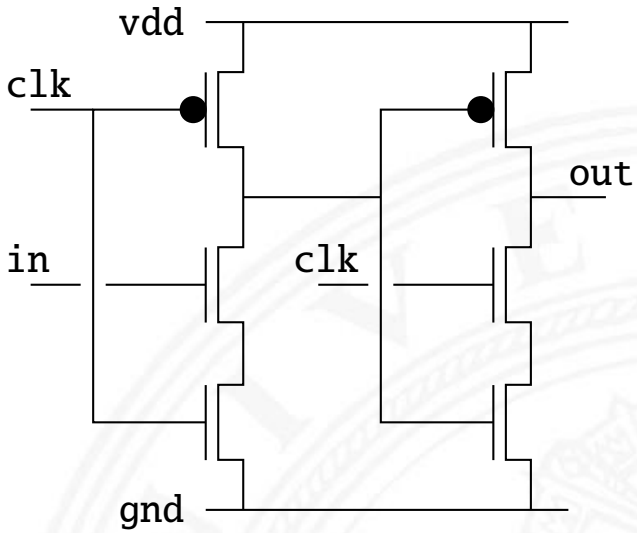


Latch - high transparent **NC-NC**

Dynamische Latches / Flipflops (cont.)

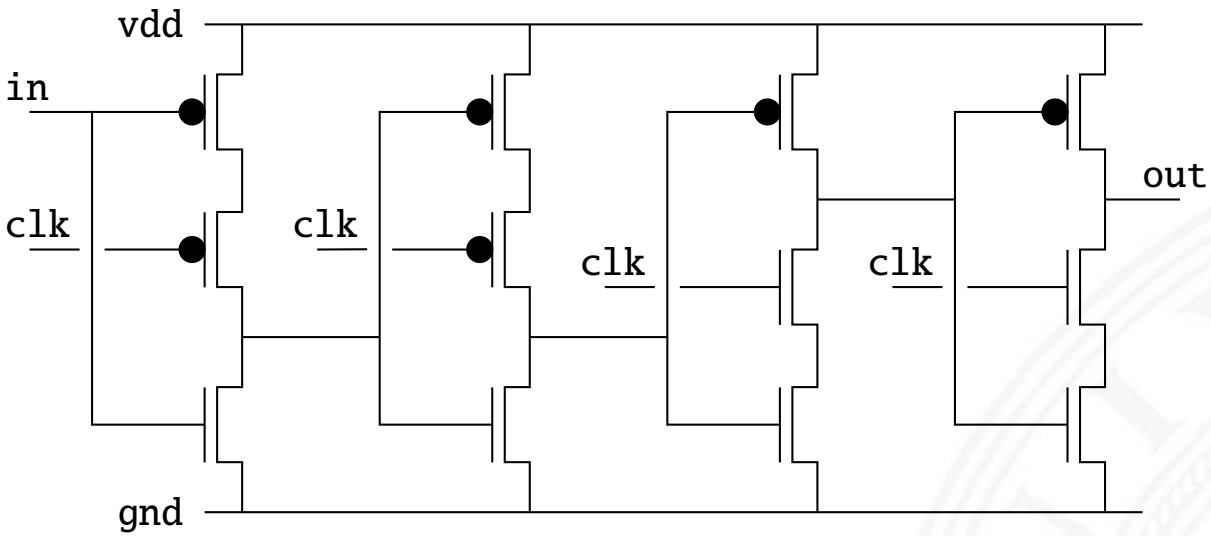


Latch - low transparent **PC2-PC**



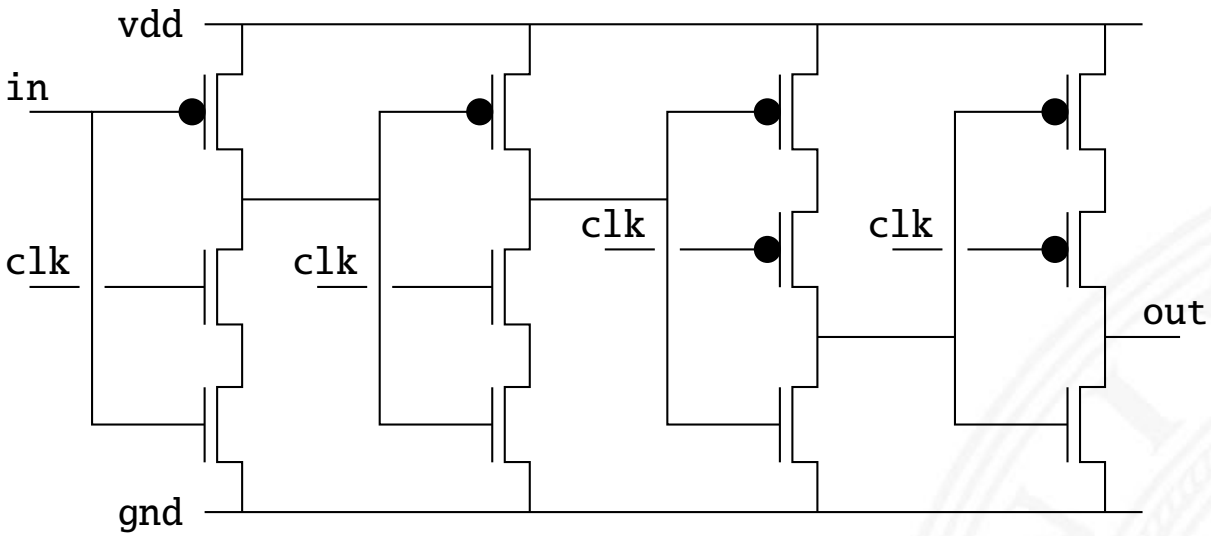
Latch - high transparent **NC2-NC**

Dynamische Latches / Flipflops (cont.)



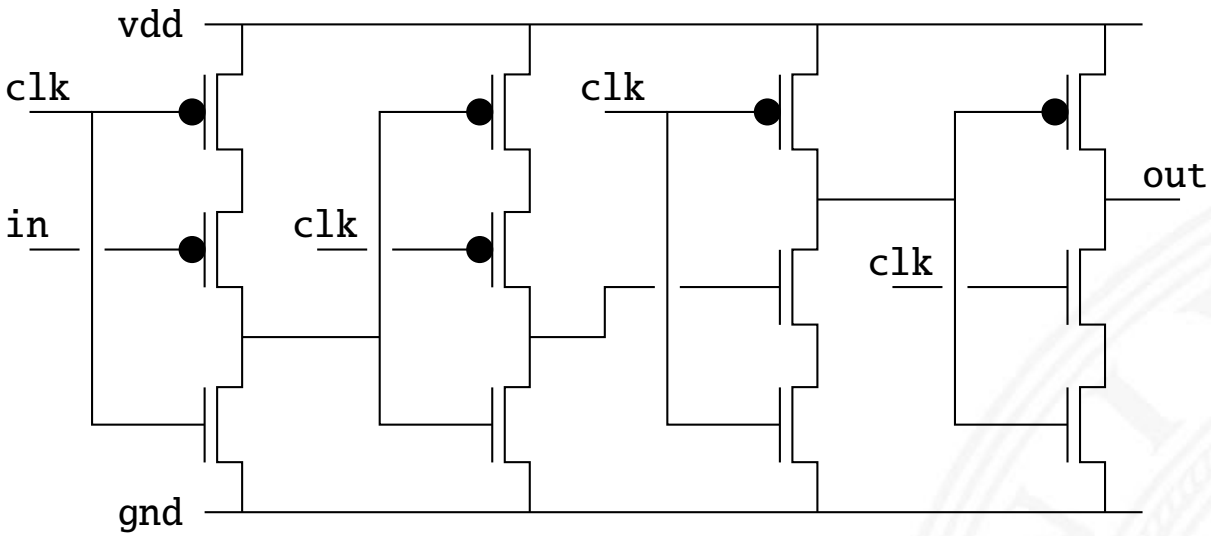
Flipflop - vorderflankengesteuert **PC-PC-NC-NC**

Dynamische Latches / Flipflops (cont.)



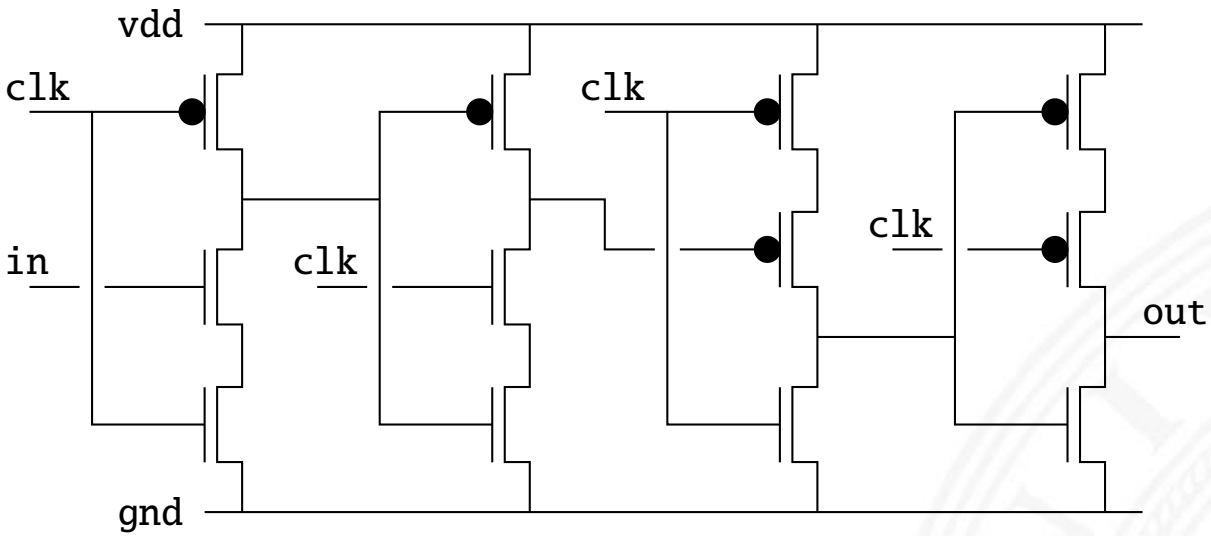
Flipflop - rückflankengesteuert **NC-NC-PC-PC**

Dynamische Latches / Flipflops (cont.)



Flipflop - vorderflankengesteuert **(PC2)-PC-NC2-NC**

Dynamische Latches / Flipflops (cont.)

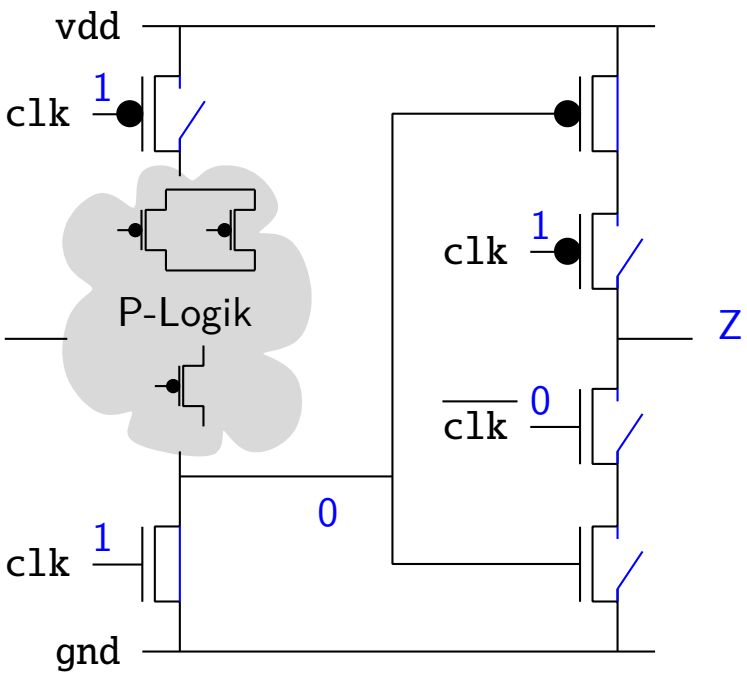


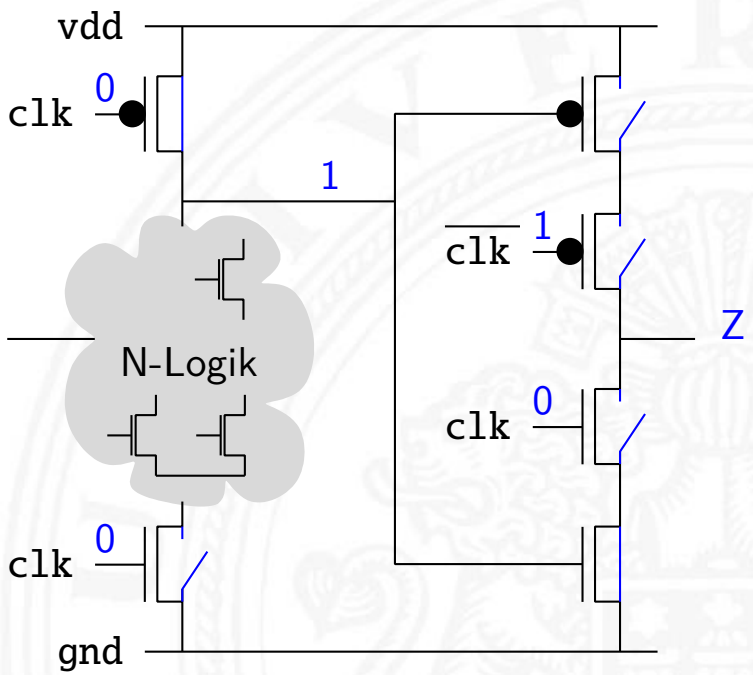
Flipflop - rückflankengesteuert **(NC2)-NC-PC2-PC**

Dynamische Latches / Flipflops

NORA Funktionsprinzip

Aufladephase



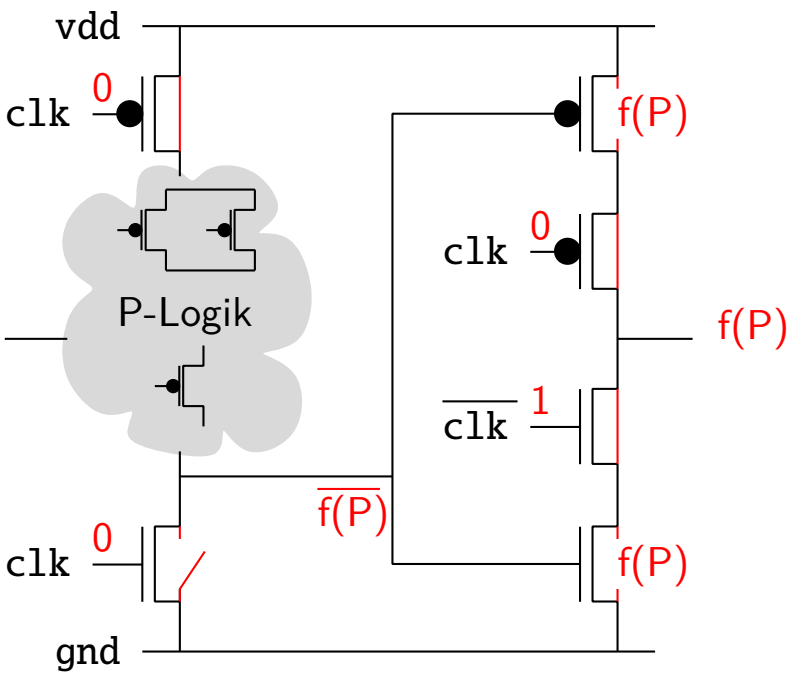


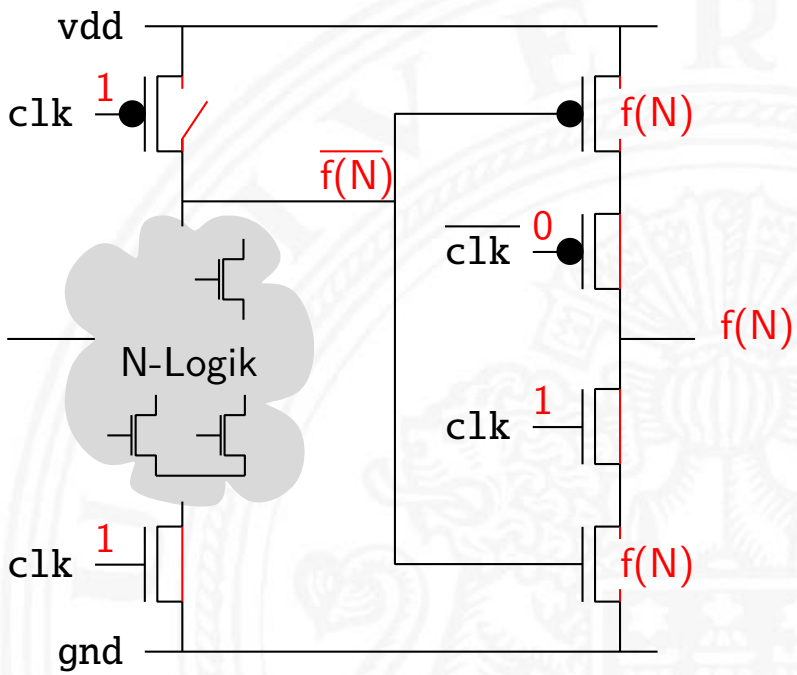
◂ NORA ◂ TSPC

Dynamische Latches / Flipflops

NORA Funktionsprinzip

Auswertungsphase





◂ NORA ◂ TSPC

Endliche Automaten

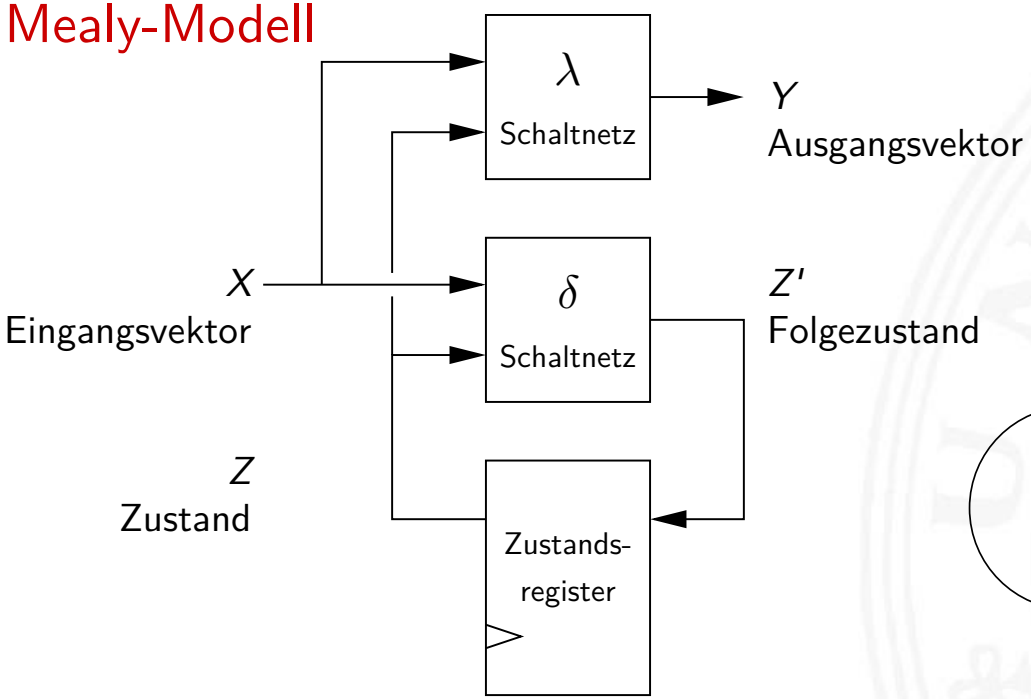
FSM – **F**inite **S**tate **M**achine

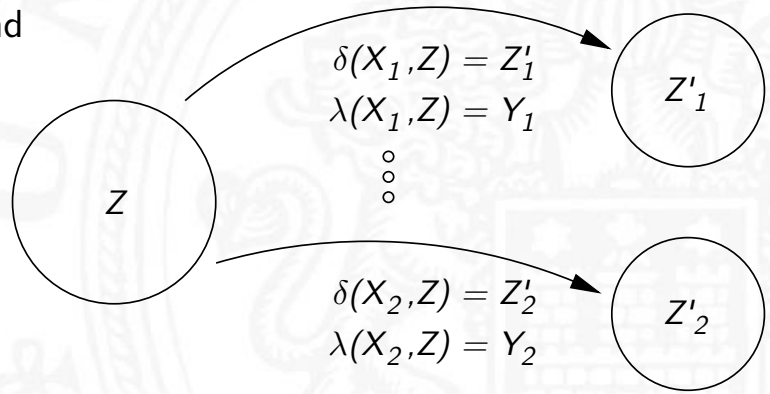
- Deterministischer Endlicher Automat mit Ausgabe
- 2 äquivalente Modelle
  - Mealy: Ausgabe hängt *von Zustand und Eingabe* ab
  - Moore: –"– *nur vom Zustand* ab
- 6-Tupel $(Z, \Sigma, \Delta, \delta, \lambda, z_0)$
  - Z Menge von Zuständen
  - Σ Eingabealphabet
  - Δ Ausgabealphabet
  - δ Übergangsfunktion $\delta : Z \times \Sigma \rightarrow Z$
  - λ Ausgabefunktion $\lambda : Z \times \Sigma \rightarrow \Delta$ Mealy-Modell
    $\lambda : Z \rightarrow \Delta$ Moore- –"–
  - z_0 Startzustand

Endliche Automaten (cont.)

- Beschreibung durch Funktionstabelle, Zustandsübergangsgraph
- siehe `http://de.wikipedia.org/wiki/Endlicher_Automat`
  Einführung in die Automatentheorie,... [HRU02]

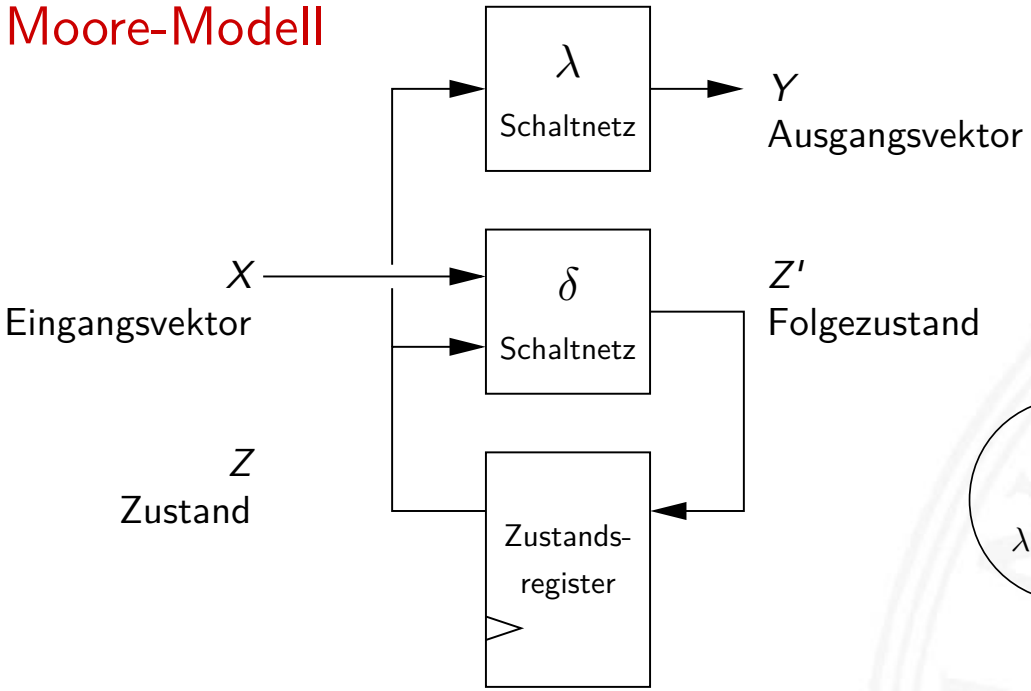
Mealy-Modell

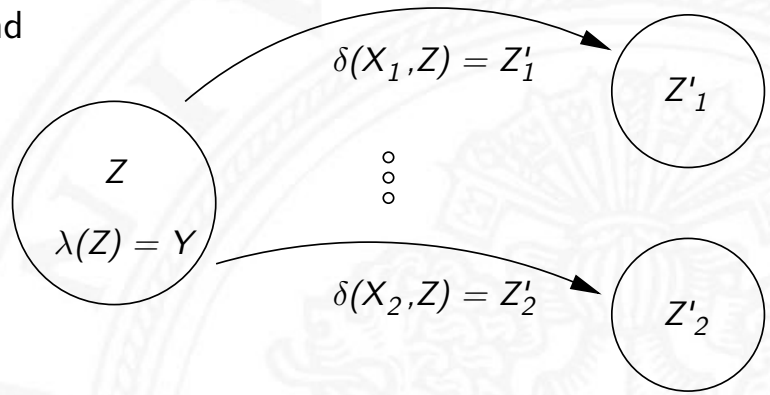




Endliche Automaten (cont.)

Moore-Modell





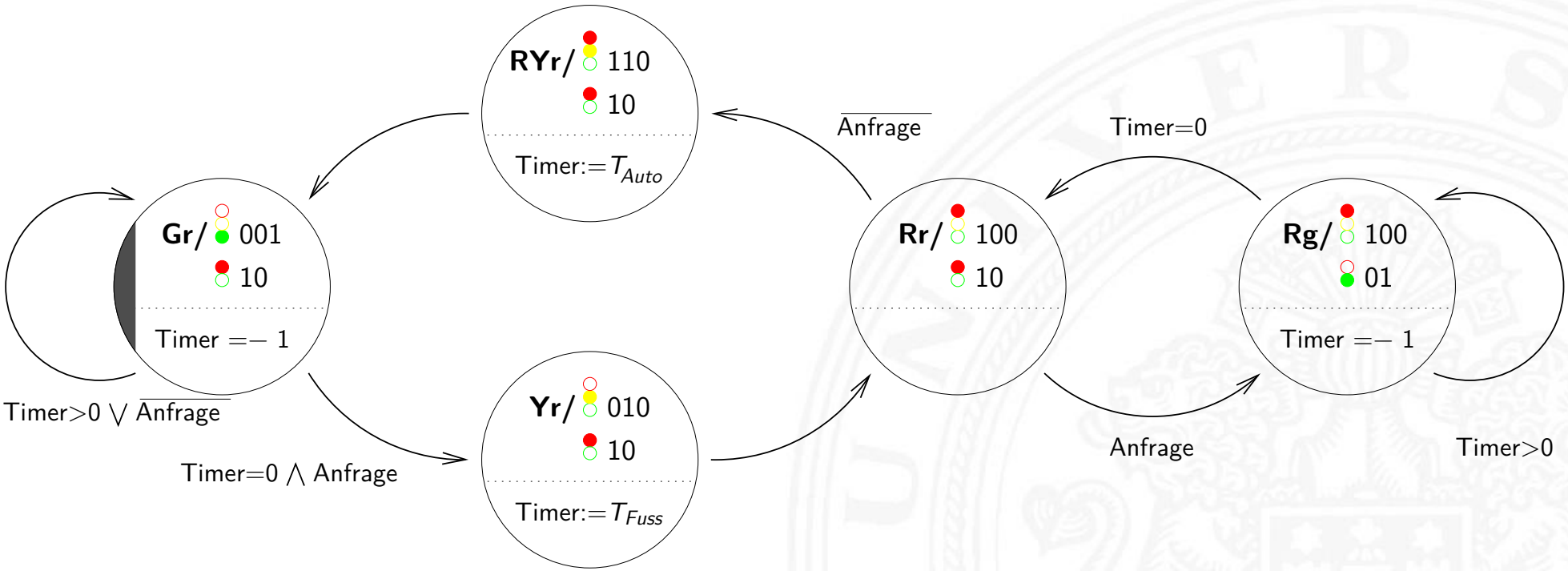
Endliche Automaten (cont.)

Besondere Bedeutung endlicher Automaten im Hardwareentwurf

- Endliche Automaten modellieren alle Strukturen, die eine Art von Rückkopplung enthalten
- Anwendungsgebiete
  - Zähler
  - Steuerungs- und Kontrollaufgaben
  - Regelungen
  - Protokolle
  - ...
- häufig mehrere *gekoppelte* Automaten
  - Rechnermodell: Steuerwerk und Rechenwerk
  - Kontrollautomat(en) und Zähler
  - ...

Endliche Automaten (cont.)

Beispiel: Ampelschaltung



Gliederung

Begriffsbildung

Wikipedia

Die Mikrosystemtechnik (MST) erstellt technische (Sub-)Systeme, deren funktionsbestimmende Strukturen Abmessungen im Mikrometerbereich haben. ...
Die Mikrosystemtechnik kombiniert Methoden der Mikroelektronik, der Mikromechanik, der Mikrofluidik und der Mikrooptik, aber auch Entwicklungen der Informatik, Biotechnologie und Nanotechnologie, indem sie Entwicklungen und Strukturen aus diesen Bereichen zu neuen Systemen vereinigt.

Begriffsbildung (cont.)

MCNC: Micromechanische Systeme (MEMS)

Integrierte Mikrobauteile oder Systeme, die elektrische und mechanische Elemente kombinieren, hergestellt in IC-kompatiblem Batch-Prozessen und in einer Größe von Mikrometer bis Millimeter. Diese Systeme verbinden Berechnung mit Sensorik und Aktuatorik um unsere Wahrnehmung der physikalischen Welt zu ändern.

Begriffsbildung (cont.)

BMBF: Programm "Mikrosystemtechnik 1994 – 1999"

Werden Sensoren, Signalverarbeitung und Aktoren in miniaturisierter Bauform so zu einem Gesamtsystem verknüpft, dass sie „empfinden“, „entscheiden“ und „reagieren“ können, spricht man von einem Mikrosystem. Hierbei ist entscheidend, dass die Funktionen eigenständig erfolgen. Sensoren entsprechen den menschlichen Sinnesorganen, die Signalverarbeitung entspricht dem Gehirn und Aktoren den Gliedmaßen.

Begriffsbildung (cont.)

Europäische Kommission, 4. Rahmenprogramm, Sep. 1996

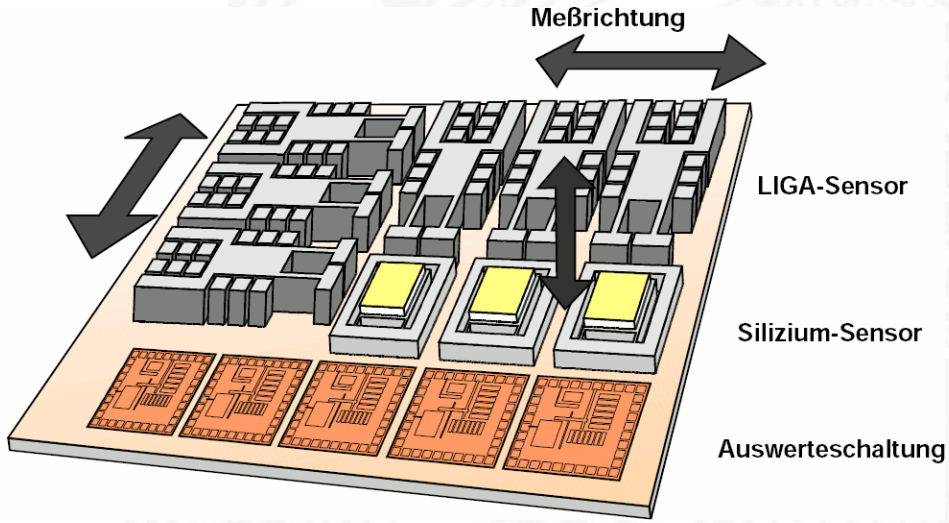
Ein Mikrosystem ist definiert als ein intelligentes miniaturisiertes System, das sensorische, datenverarbeitende und aktuatorische Funktionen beinhaltet. Dies wird normalerweise durch folgende Kombination erreicht: elektrische, mechanische, optische, chemische, biologische, magnetische oder andere Prozesse, integriert auf einem einzigen Chip oder einer Hybridschaltung.

Links / Material

- http://de.wikipedia.org/wiki/Mikrosystemtechnik
- http://en.wikipedia.org/wiki/Microelectromechanical_systems
- http://memscyclopedia.org/introMEMS.html
- http://mems.sandia.gov
- http://www.mstonline.de/mikrosystemtechnik
- http://www.memsnet.org
- http://www.stimesi.org
- http://www.sintef.no/Projectweb/Microbuilder

Mikrosystem

- Strukturgröße: Mikrometer-/Submikrometer-Bereich
- Herstellung durch typische Halbleiter-Technologien: Photolithografie, Filmabscheidung, Ätzprozesse
- Systemkomponenten
  - Sensoren
  - Aktoren
  - Signalverarbeitung



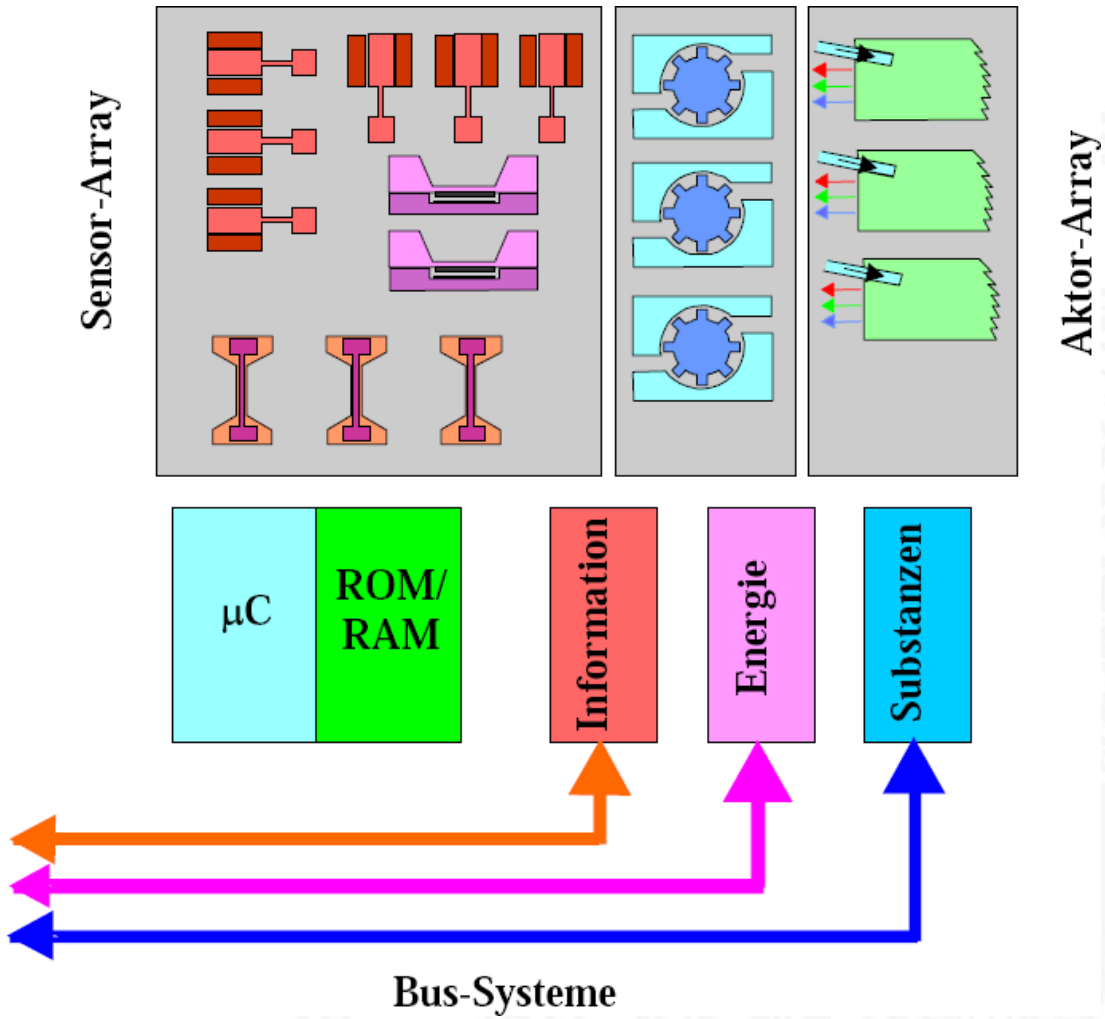
3D- Beschleunigungssensor

Mikrosystem (cont.)

Funktion	Komponente	Beispiel
elektronisch	Mikroelektronik	Prozessoren, Glue-Logik, Speicher...
		A/D-, D/A-Wandler, Operationsverstärker
mechanisch	Mikrosensor	Beschleunigung, Drehmoment, Druck,
		Temperatur, Durchfluss...
	Mikroaktuator	Magnetkopf, Mikroschalter, Pumpe, Ventil...
	Mikrofluidik	Wärmetauscher, Mikroreaktor,
		Dosiereinrichtung, Analysesystem...
	Mikroakustik	Frequenzmessung, Verzögerungsleitung,
		Signalverarbeitung, Filter...
optisch	Mikrooptik	Fiberoptische Komponenten, Sensoren...
	Integrierte Optik	Fiberoptische Komponenten, optische Schalter...
chemisch	Mikrochemie	chemische Analyse
biologisch	Mikrobiologie	biologische Analyse

Mikrosystem (cont.)

Mikrosystem

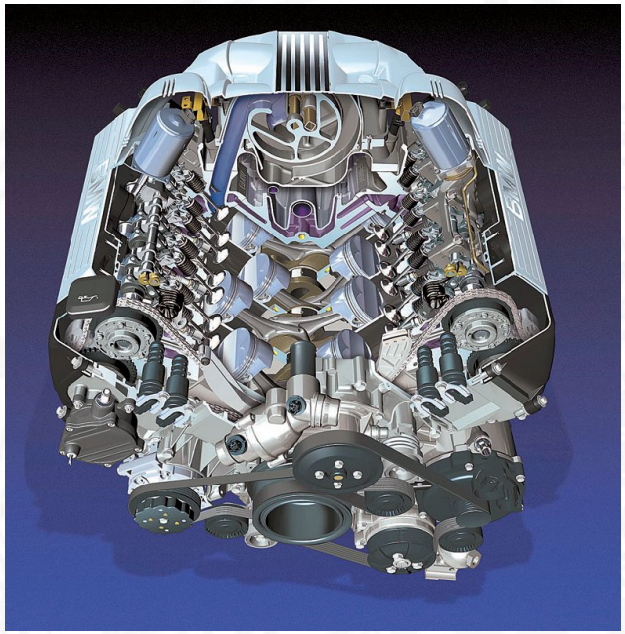


Mikrosystem (cont.)

- Informationsübertragung:
  elektrisch, elektromagnetisch, induktiv, kapazitiv, optisch, akustisch
- Energieeinspeisung:
  elektrisch, elektromagnetisch, induktiv, kapazitiv, optisch, akustisch fluidisch, mechanisch
- Stoffübertragung:
  Spülflüssigkeiten, Medikamente, Analyte (Gase/Flüssigkeiten), biologische Materie

Skalierung physikalischer Prozesse

- Problem: physikalische Größen skalieren unterschiedlich
- einfache Verkleinerung funktioniert nicht



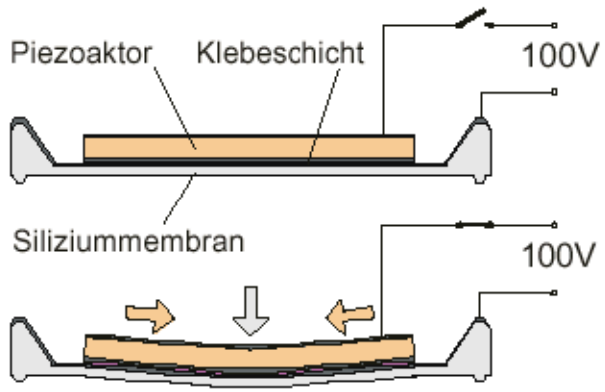
Beispiel: Motor

r^3	Masse, Energiedichte...
r^2	Reibung, Wärmeverlust...
r^1	Drehmoment, Zündspannung...
r^0	Druck, Temperatur...

$\Rightarrow$ elektronik andere physikalische Funktionsprinzipien nutzen

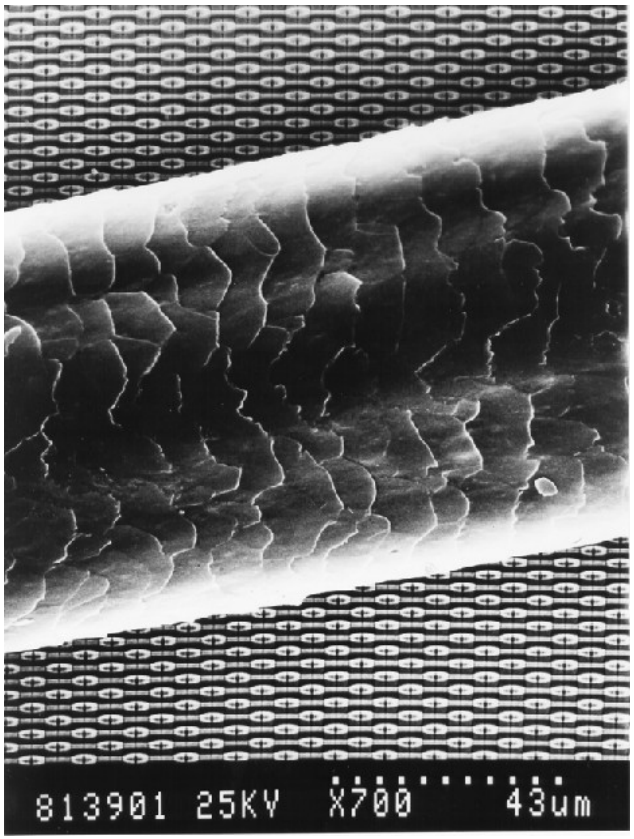
Mikroaktoren

- elektrostatische Kräfte
- Bimetalleffekt
- Piezoeffekt (SiO_2 ist piezoelektrisch)
- Formgedächtniseffekt

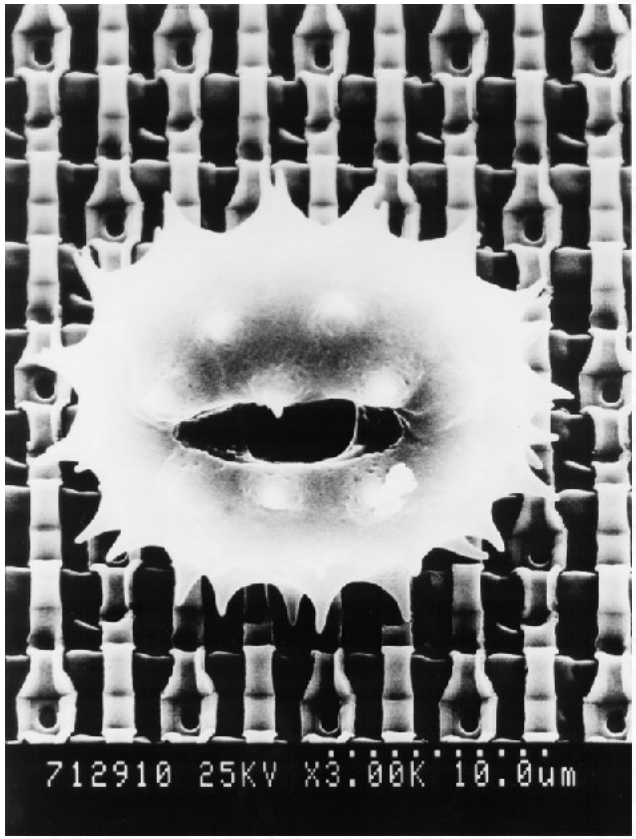


Größe der Strukturen

Haar

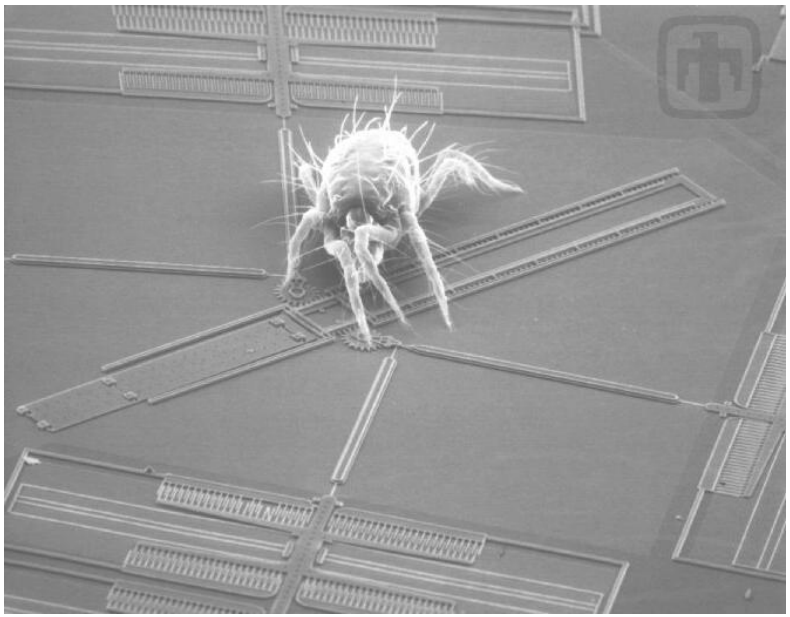


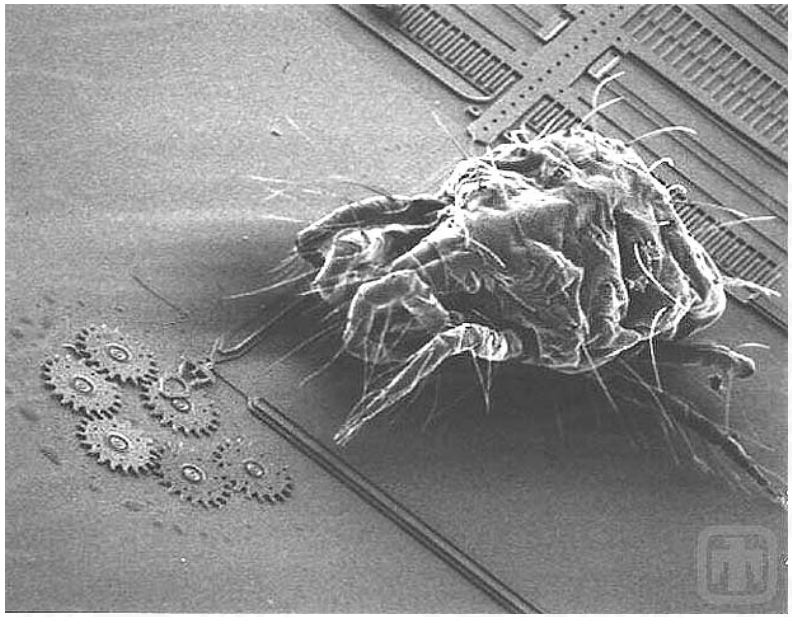
Blütenpolle



Größe der Strukturen (cont.)

Hausstaubmilbe

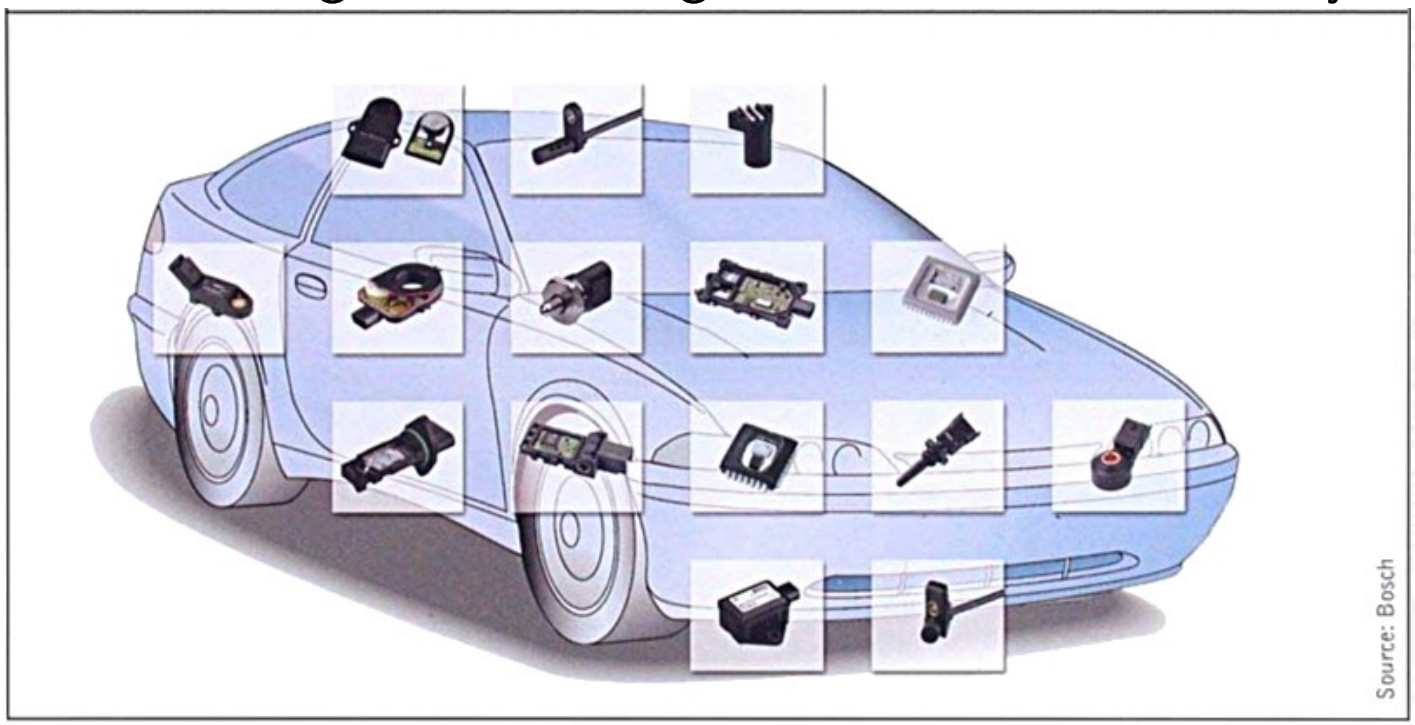




Video 1 Video 2 `http://mems.sandia.gov`

Anwendungen

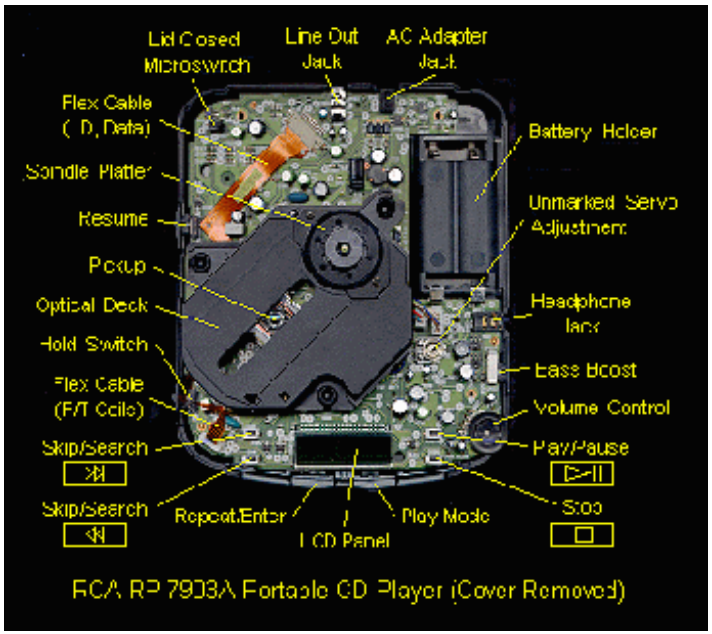
- Kraftfahrzeugtechnik, Navigations- und Verkehrsleitsysteme



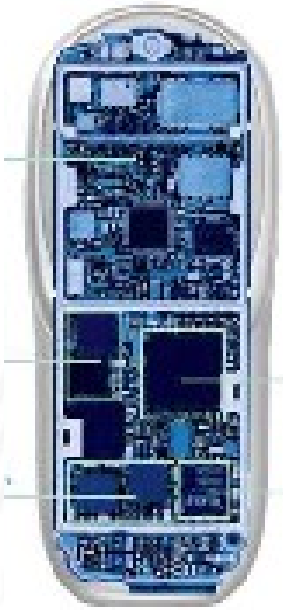
Sensors are the main application field for microsystems in the car. A high-end car may contain between 50 and 100 sensors, many of which are MST-based

Anwendungen (cont.)

- Konsumelektronik

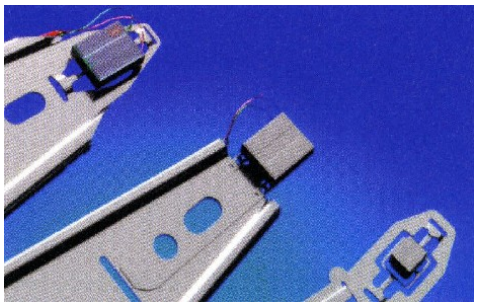


Kommunikationstechnik



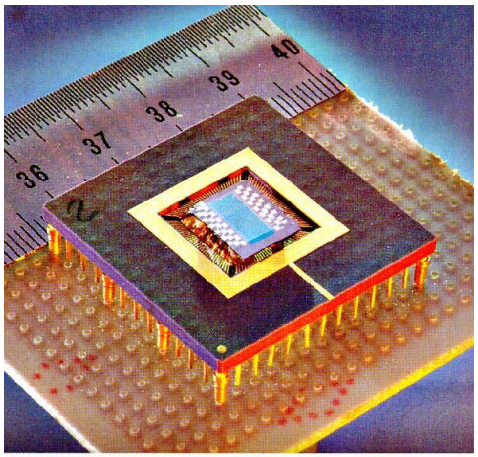
Anwendungen (cont.)

- Computertechnik



Anwendungen (cont.)

- Freizeitmarkt
- Haustechnik (Domotik) und Haushaltsgeräte
- intelligente, dezentrale Prozess- und Regeltechnik
- Luftfahrttechnik und Flugsicherheit
- Umwelttechnik



„Elektronische Nase“

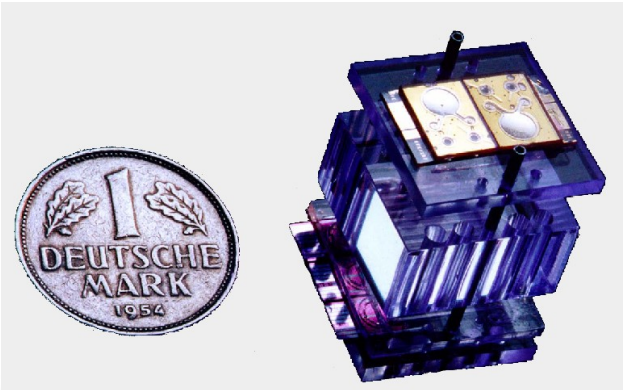
Anwendungen (cont.)

- Medizintechnik

Hörgerät

Implantierbare Medikationspumpe

Bio-/ Gentechnik



Labor auf dem Chip

Silizium

Vergleich von Silizium mit anderen Materialien

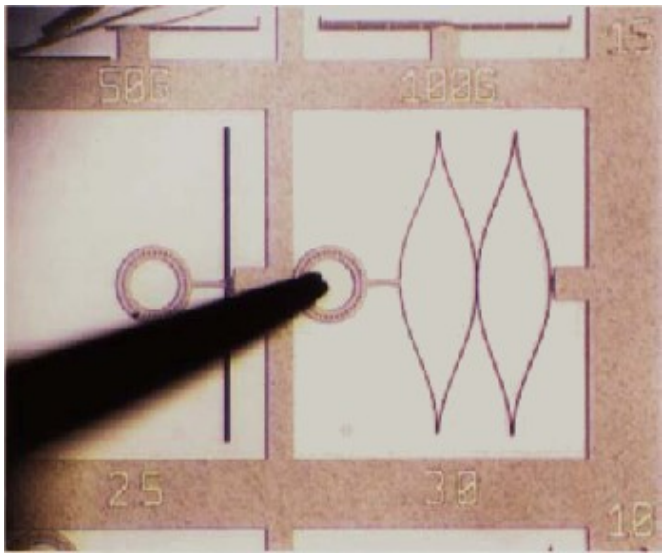
	Diamant	Si_3N_4	Si	Stahl	Al	
Max. Zugbelastung	53,0	14,0	7,0	2,1	0,17	$[10^9 N/m^2]$
Knoop-Härte	70,0	34,9	8,5	6,6	1,3	$[10^9 N/m^2]$
Elastizitätsmodul	1035	385	190	200	70	$[10^9 N/m^2]$
Dichte	3,5	3,1	2,3	7,9	2,7	$[g/cm^3]$
Wärmeleitfähigkeit			157	33		$[W/mK]$
Therm. Ausdehnung	1,0	0,8	2,3	17,3	25,0	$[10^{-6}/K]$

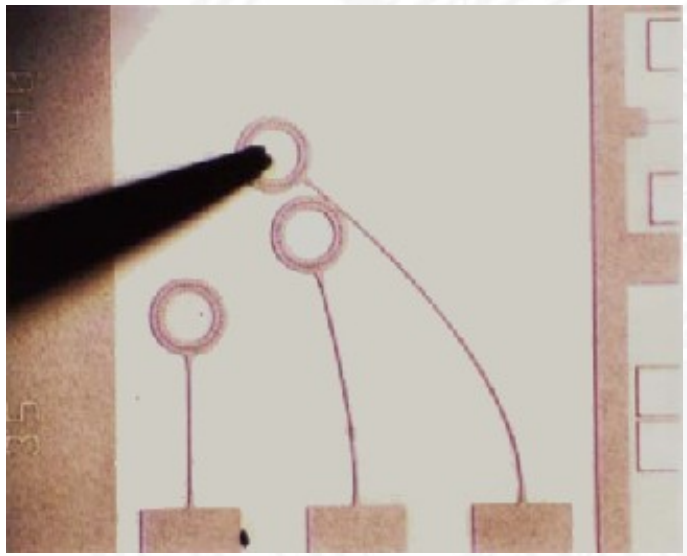
- Härte
- Wärmeleitfähigkeit
  - \+ (Ab-) Wärmetransport
  - – Wärmespeicherung (Messung der Wärmestrahlung)

Silizium (cont.)

- Elastizität
  - Einkristallines Si zeigt keine Ermüdungserscheinungen
  - Einkristallines Si zeigt keine plastische Verformung ($< 600°C$)
  - Streckgrenze = Bruchspannung

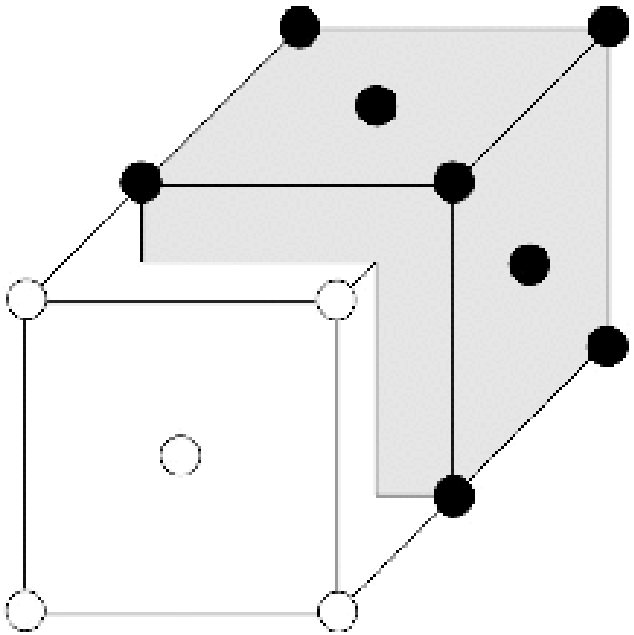
  $\Rightarrow$ Wenn eine Siliziumstruktur gedehnt wird, kehrt sie wieder in ihre Ausgangsform zurück oder sie zerbricht

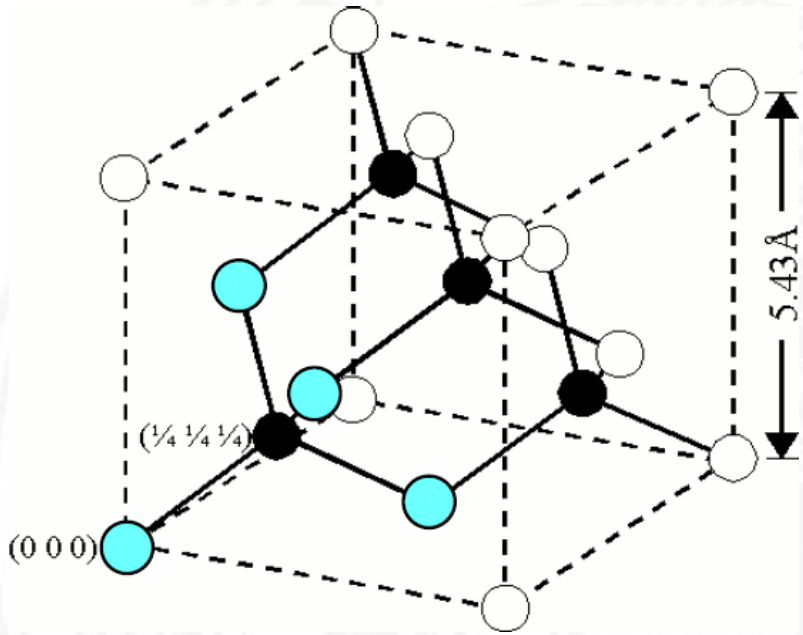




Silizium Einkristall

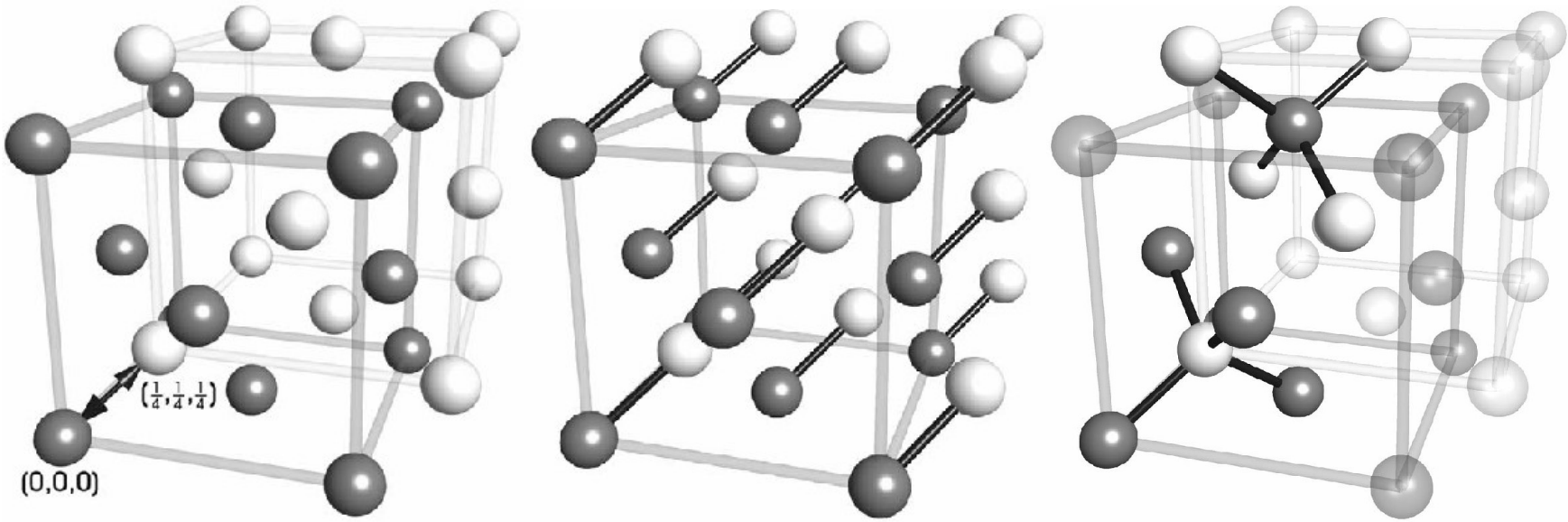
- Kristallstruktur: zwei ineinander gestellte kubisch-flächenzentrierte Gitter
- Verschiebung um $\frac{1}{4}$ der Raumdiagonalen





Silizium Einkristall (cont.)

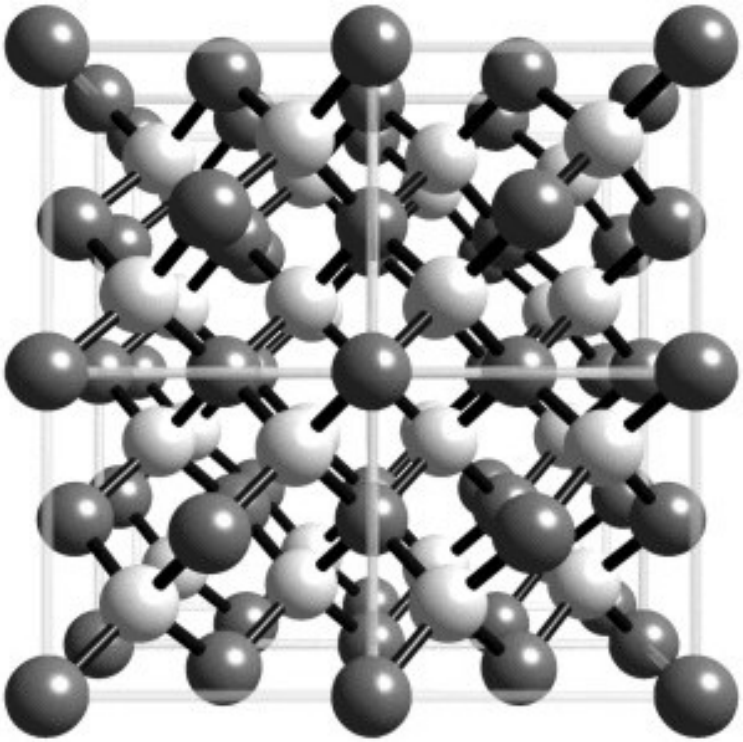
Orientierung der Bindungen

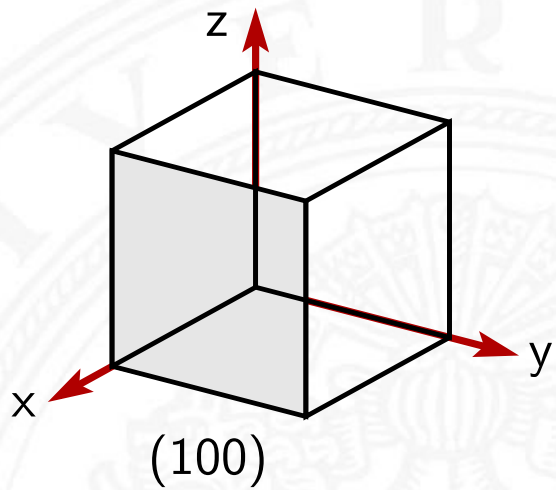


Silizium Einkristall (cont.)

Flächen vom Betrachtungswinkel des Si-Gitters abhängig

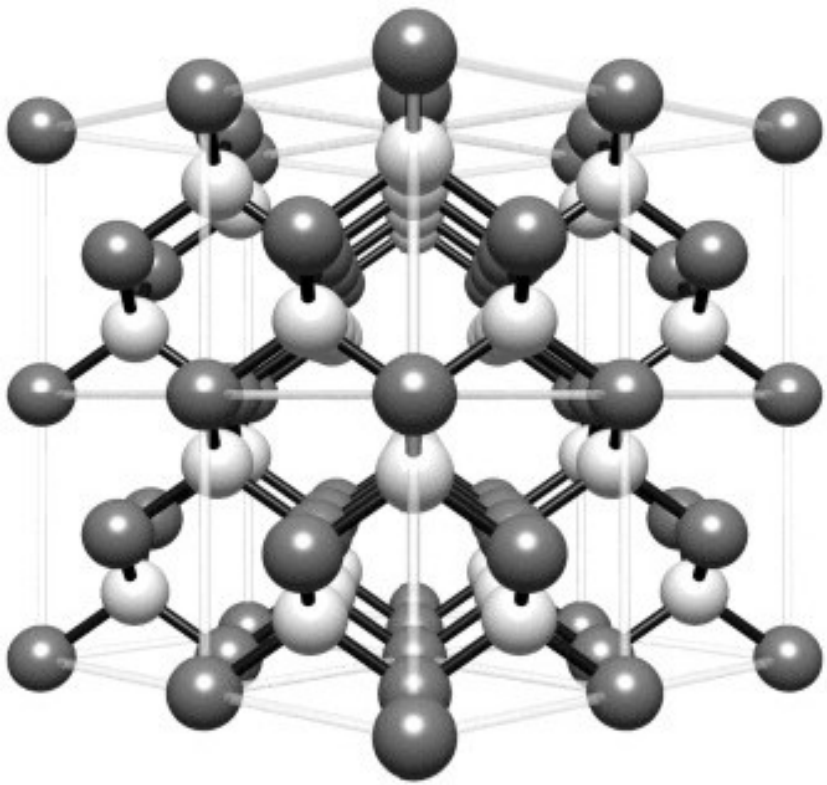
- (100)-Sicht: „Seite des Würfels“

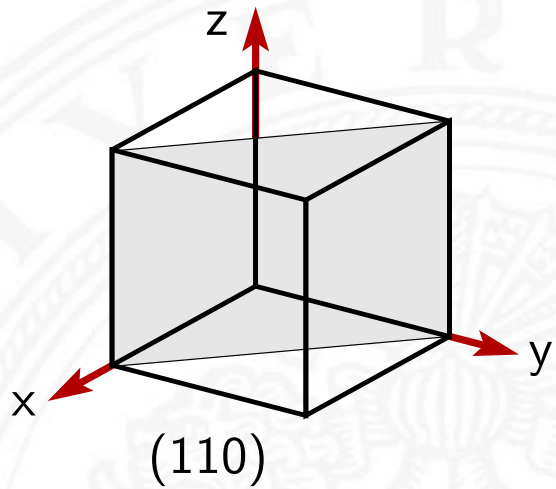




Silizium Einkristall (cont.)

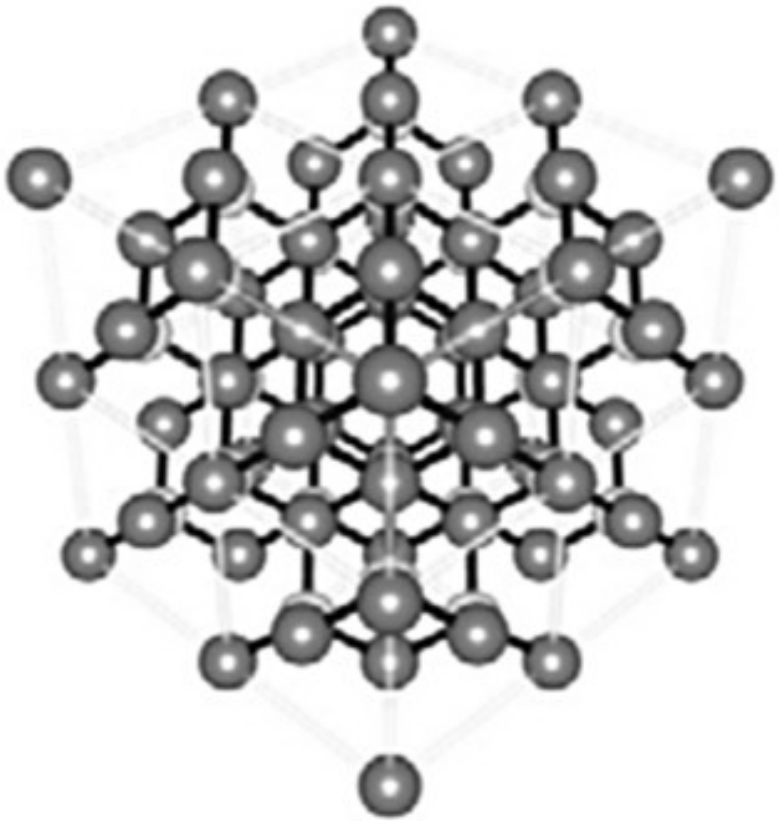
- (110)-Sicht: „Kante des Würfels“

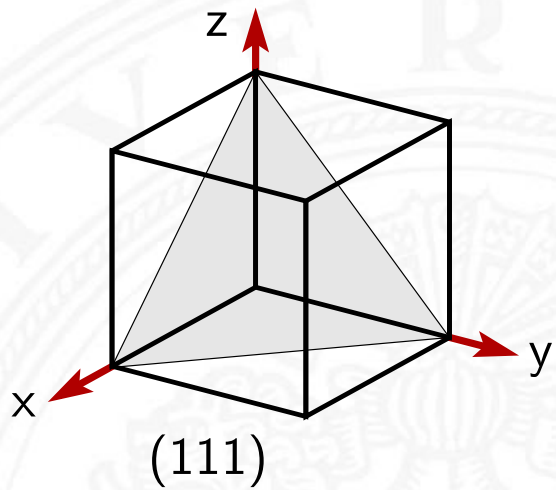




Silizium Einkristall (cont.)

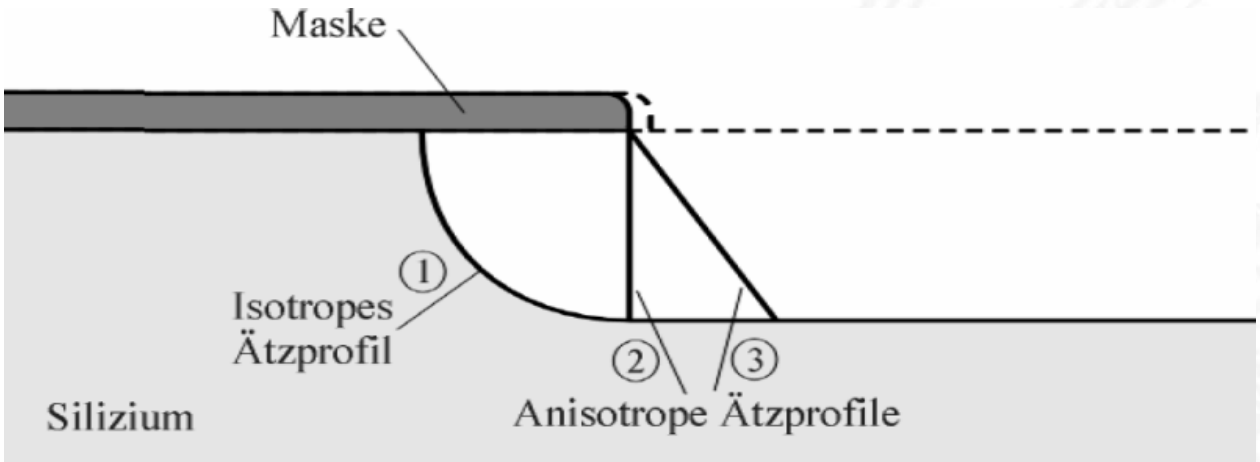
- (111)-Sicht: „Ecke des Würfels“





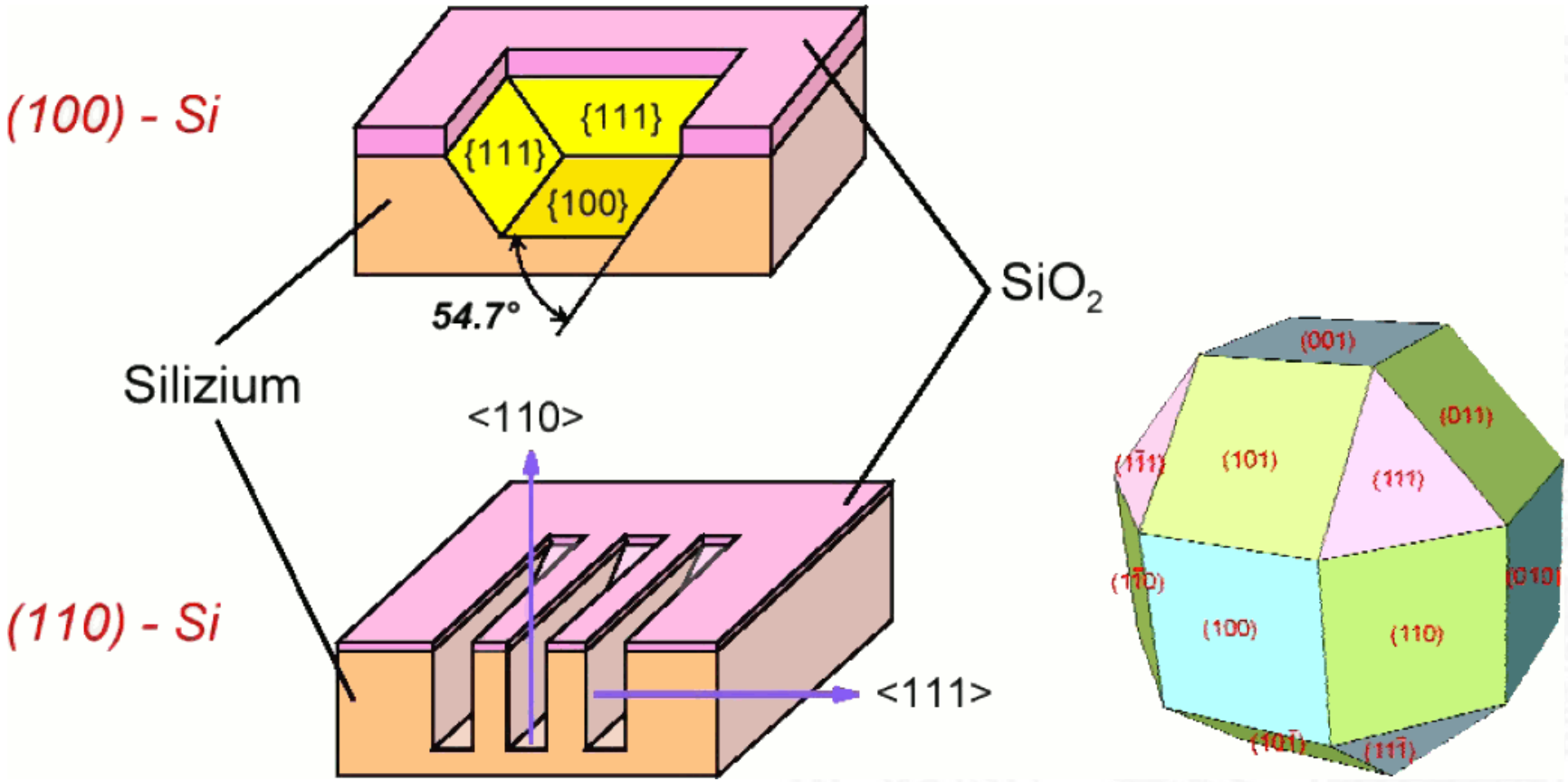
Ätzverfahren

- Halbleitertechnik: SiO_2 gleichmäßiges (isotropes) Ätzen
- Mikrosystemtechnik: vom Verfahren abhängig
  - isotropes Ätzen: Kugelausschnitte
  - anisotropes Ätzen: gerade Kanten
    1. entlang der Kristallstruktur
    2. physikalisch erzeugt



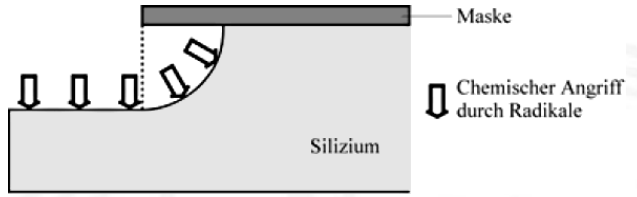
Ätzverfahren (cont.)

Ebenen im Silizium Einkristall

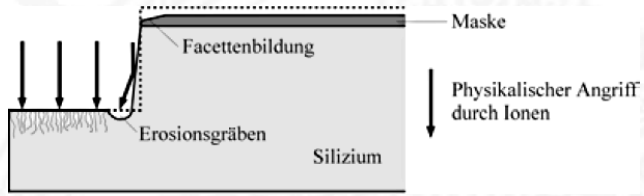


Ätzverfahren (cont.)

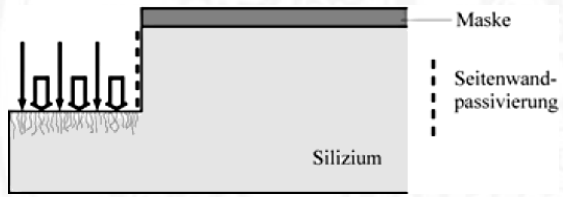
- isotropes Ätzprofil
  - Chemisches Ätzen (Nass)
  - Plasmaätzen (Trocken)



- anisotropes Ätzprofil: senkrecht
  - Sputterätzen (Trocken)

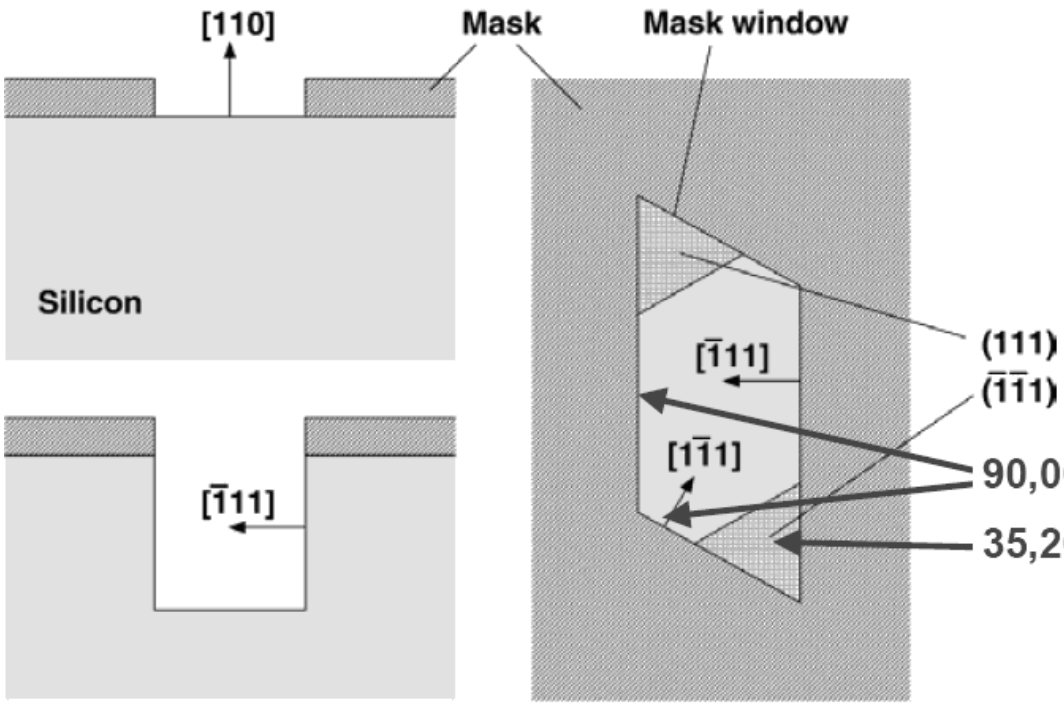


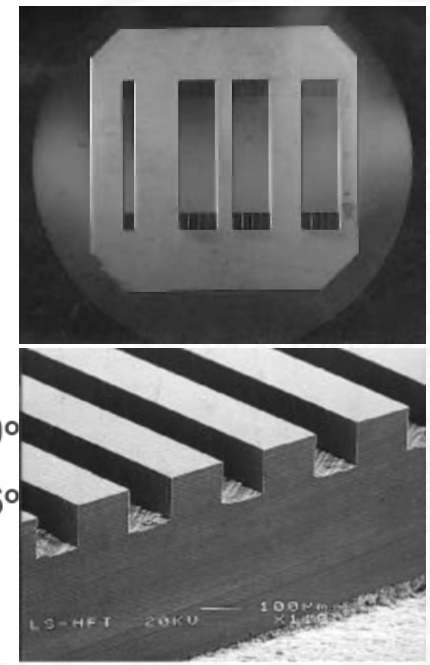
  - Reaktives Ionenätzen (Trocken)



Ätzverfahren (cont.)

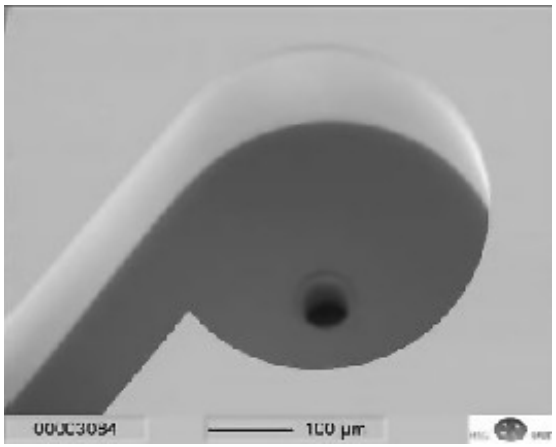
- KOH-Ätzen (110) (Nass) Kaliumhydroxid

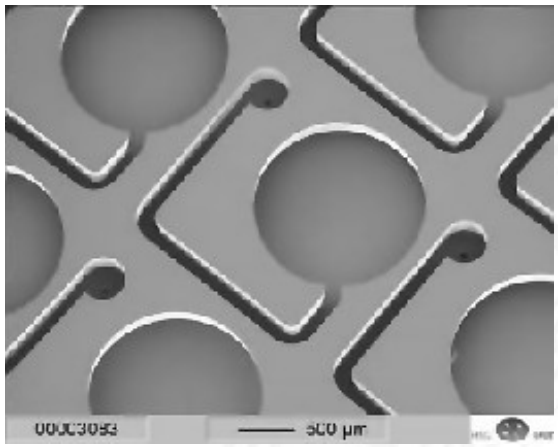




Ätzverfahren (cont.)

Trockengeätzte Strukturen

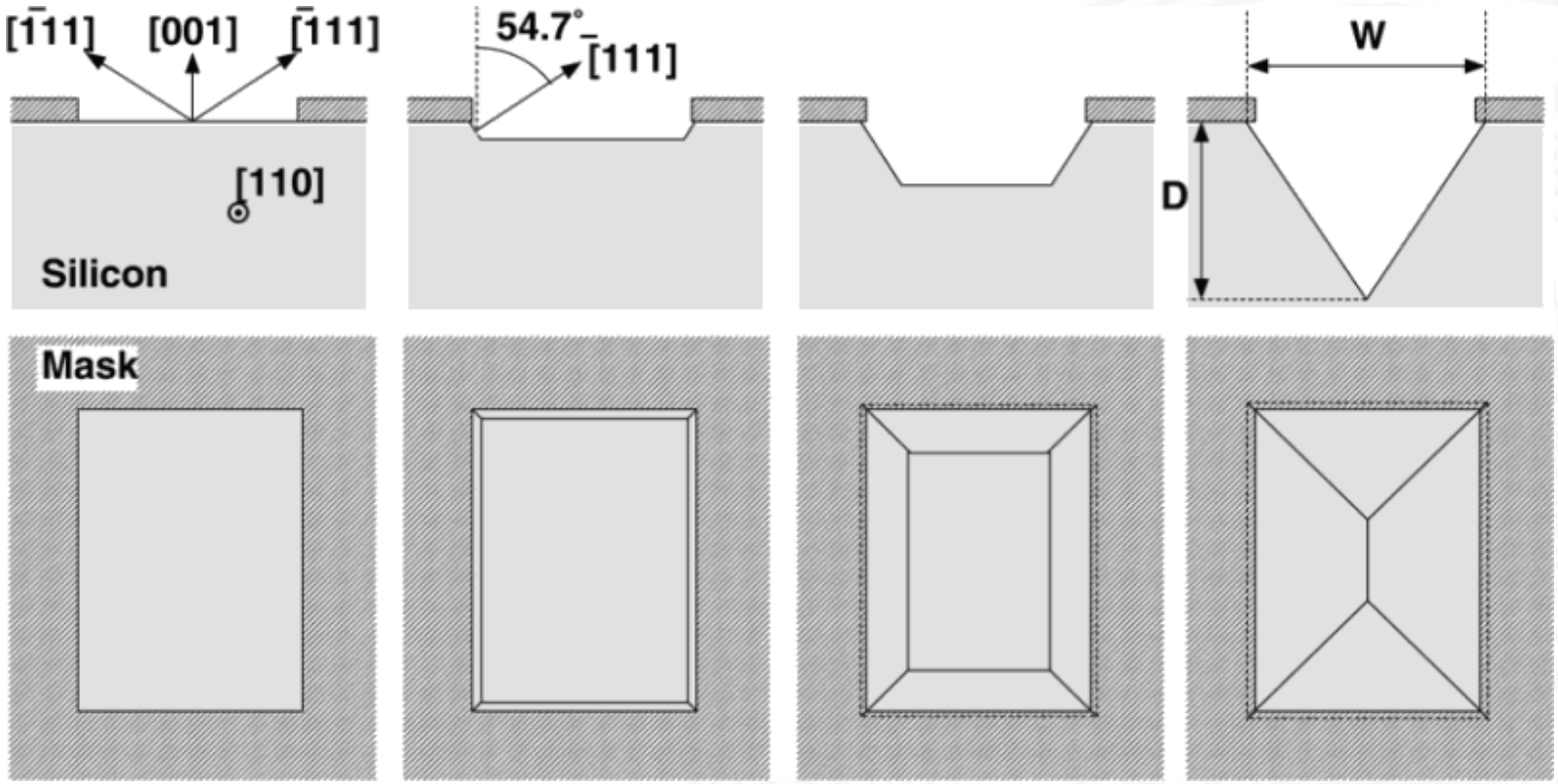




Mikrofluidik Düsen

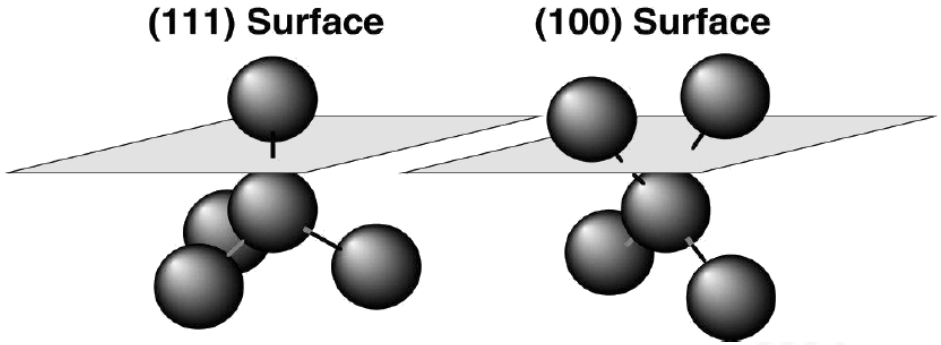
Ätzverfahren (cont.)

- anisotropes Ätzprofil: schräg
  - KOH-Ätzen (100) (Nass) Kaliumhydroxid



Ätzverfahren (cont.)

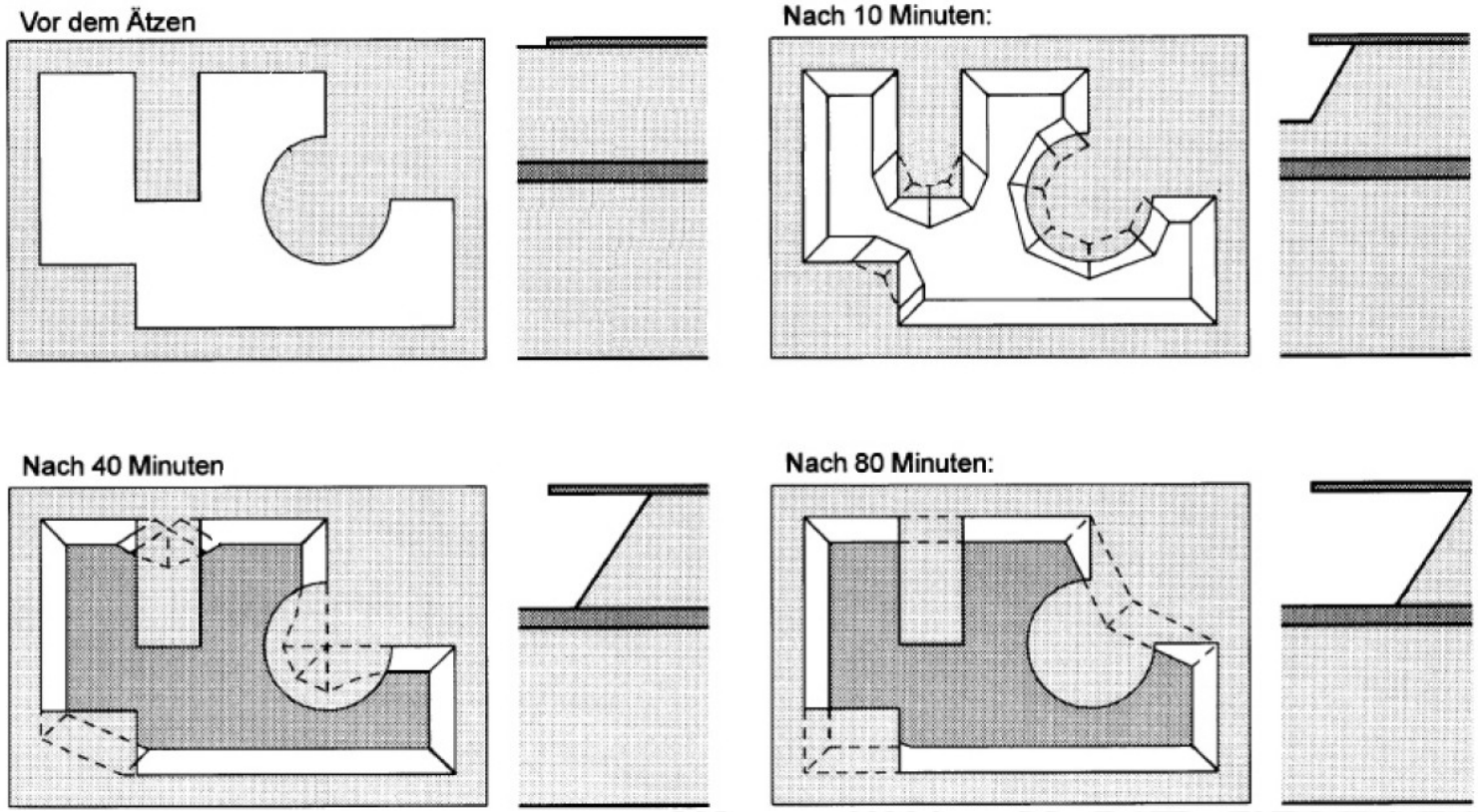
- Ätzgeschwindigkeiten unterschiedlich wegen Bindungen in den Kristall



- (111)-Ebene: am dichtesten gepackt, sehr langsames Ätzen
  (100)-Ebene: 400 $\times$ schneller
- Ätzstopp
  - (111)-Ebenen
  - p^+-Ätzstopp (Bor-Dotierung)
  - elektrochemisch

Ätzverfahren (cont.)

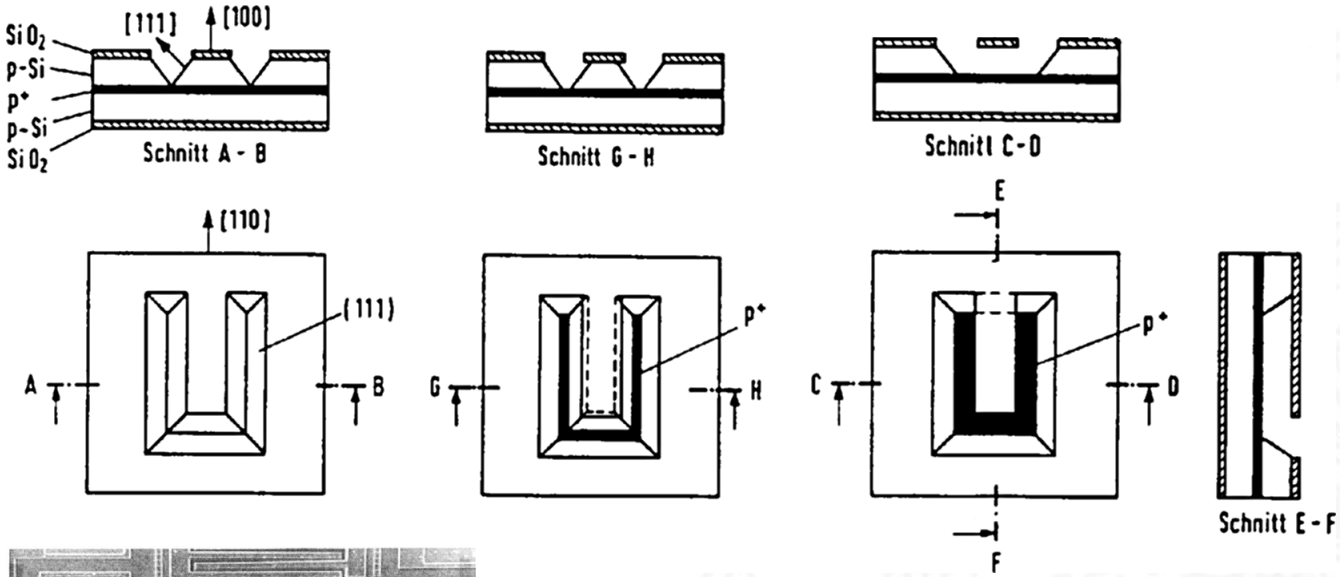
- Einfluss von Zeit und Konzentration beim Ätzen

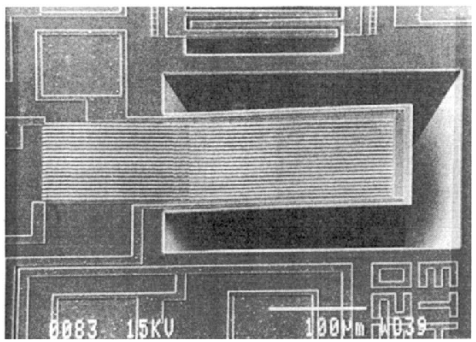


- Kompensationsstrukturen für Ecken

Ätzverfahren (cont.)

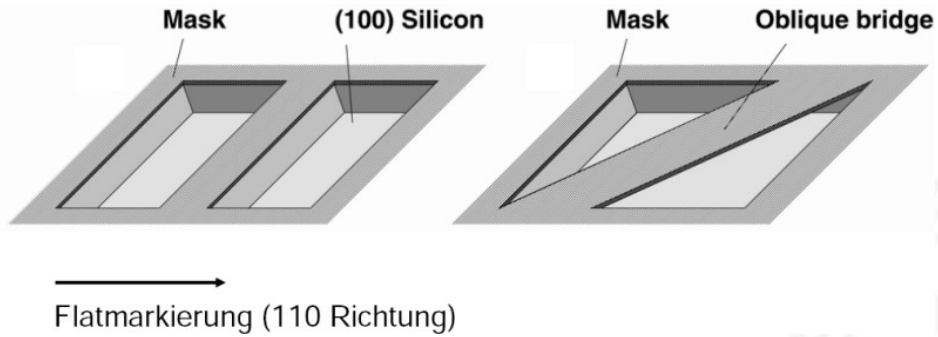
- freistehende Strukturen durch Unterätzen der Maske



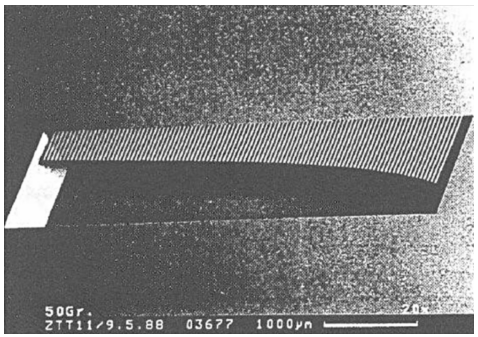


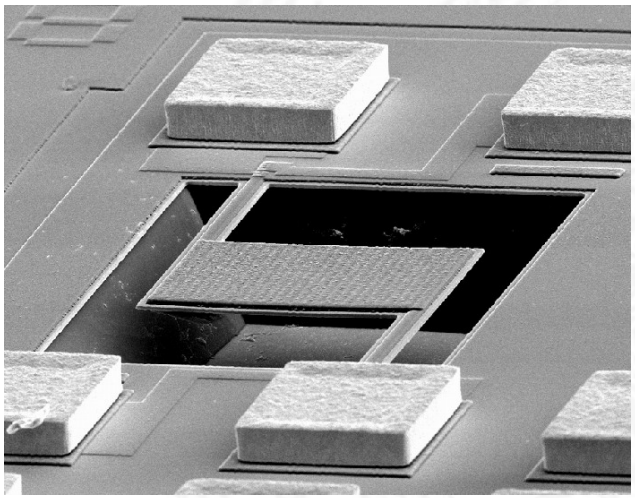
Ätzverfahren (cont.)

Maske nicht zur Kristallstruktur ausgerichtet: Unterätzung



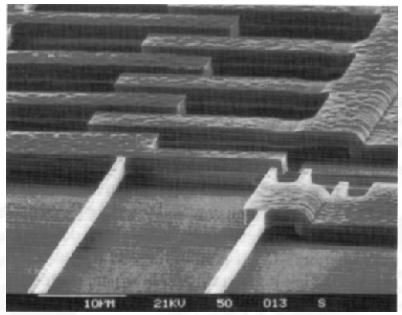
Beispiele

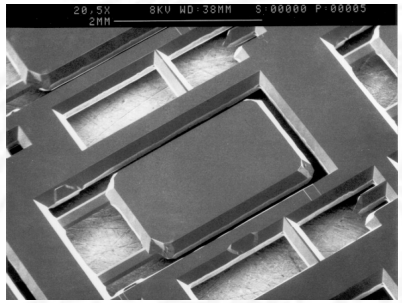


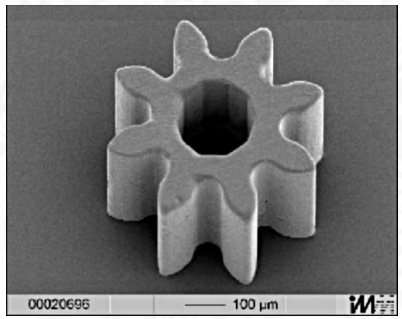


Herstellungsverfahren

- Oberflächenmikromechanik (OMM)
- Bulk-Mikromechanik
- LIGA-Verfahren





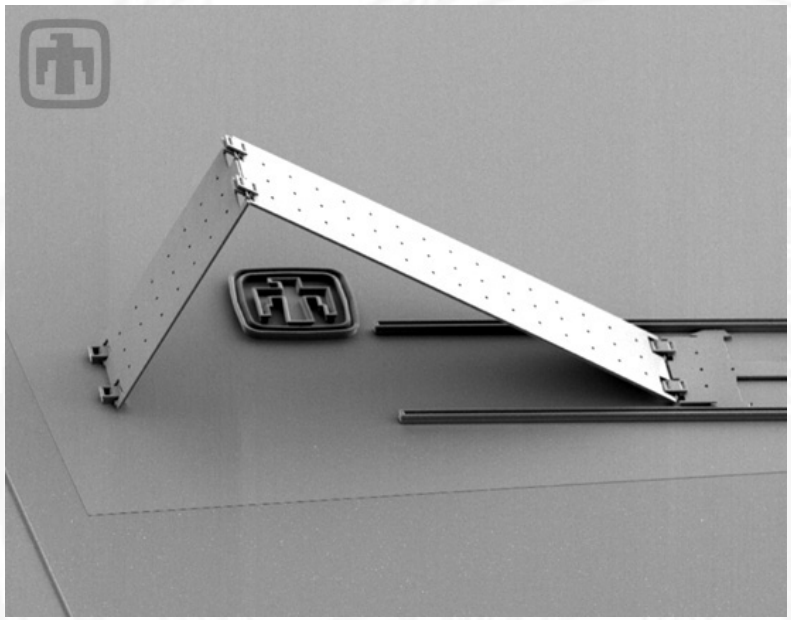


Herstellungsverfahren (cont.)

- weitere Bearbeitungstechniken
  - Laser
  - Funkenerosion
  - Mikrofräsen
  - ...

Herstellung und Funktion

- Mikromechanische Elemente werden an der Oberfläche aufgebracht
- frei bewegliche Strukturen
  - Räder auf Achsen
  - gleitende Stangen
  - Scharniere
  - ...



Herstellung und Funktion (cont.)

1. Abscheidung und Strukturierung der Opferschicht

- dient als Distanzschicht
- definiert dadurch den Abstand zu Substrat
- trägt die Funktionsschicht

Materialien

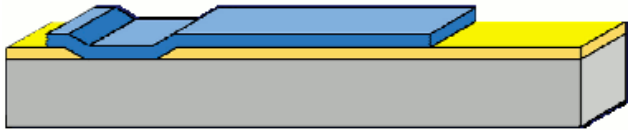
- isotrop ätzbar
- selektiv (verglichen mit Strukturschicht) ätzbar
- Prozesskompatibilität, z.B. CMOS

$\Rightarrow$

Opferschicht	Strukturschicht
SiO_2	Polysilizium
Polymer	Aluminium

Herstellung und Funktion (cont.)

2. Abscheidung und Strukturierung des Polysiliziums

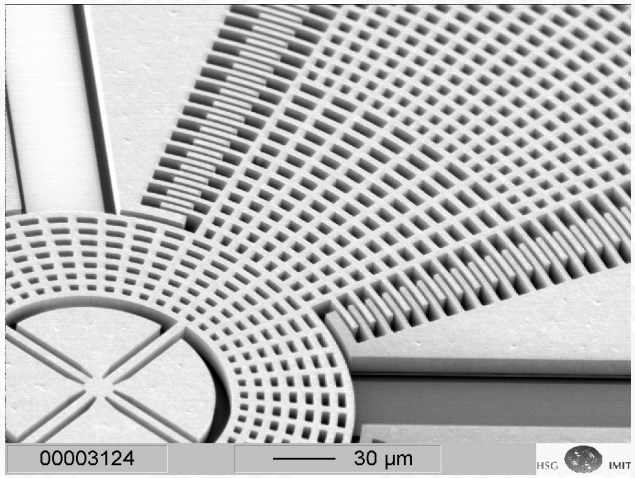


3. Entfernen der Opferschicht

- isotropes Unterätzen
  lange Ätzzeiten für breite Strukturen

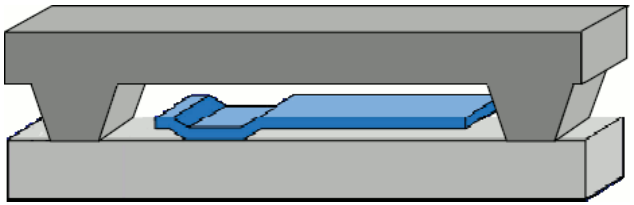
$\Rightarrow$ schmale Strukturen verwenden

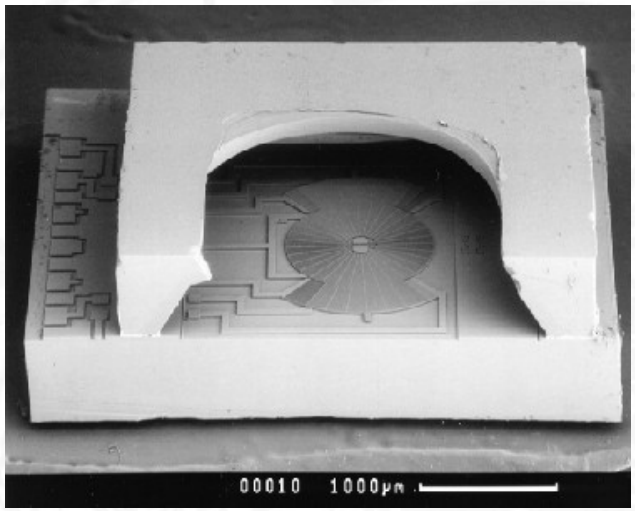
$\Rightarrow$ große Flächen „perforieren“



Herstellung und Funktion (cont.)

4. Aufglasen eines Siliziumdeckels (Schutz der Mikrostruktur)

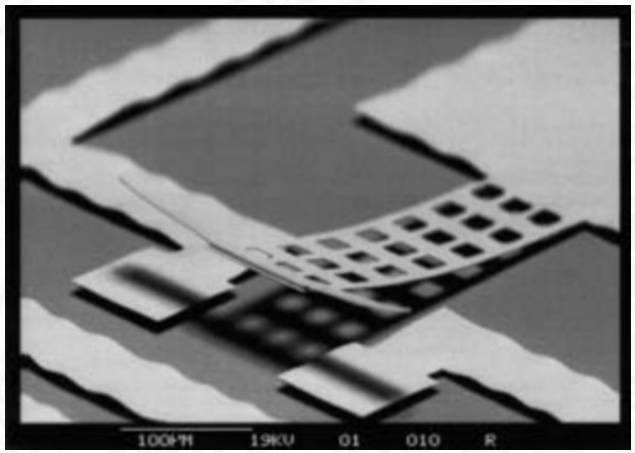




Beispiel: Drehratensensor

OMM-Probleme

- geringe Steifigkeit in Z-Dimension
  Funktionsschicht ist nur wenige μm dick
- Interne Spannungen (Grenzschichten im Silizium)
  - Aspektverhältnis (Länge:Breite) bis zu 1000
  - Beispiel: Biegezungen
    1000 μm Länge
    1 μm Breite
    4 μm Dicke

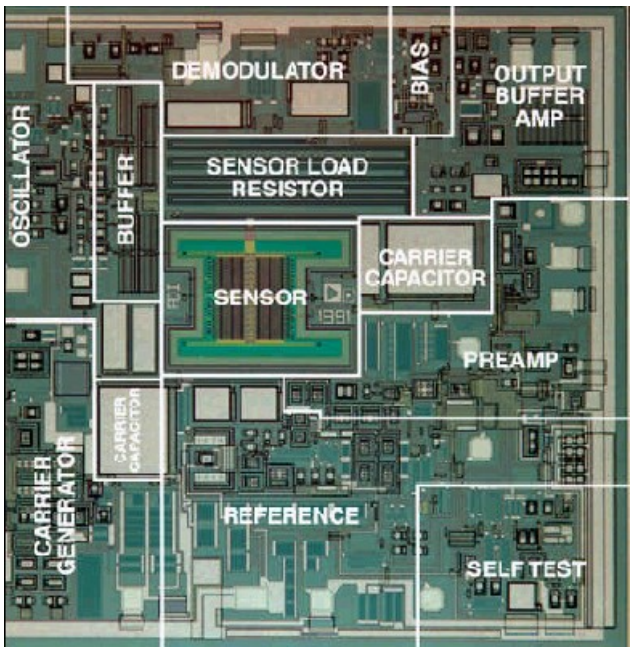


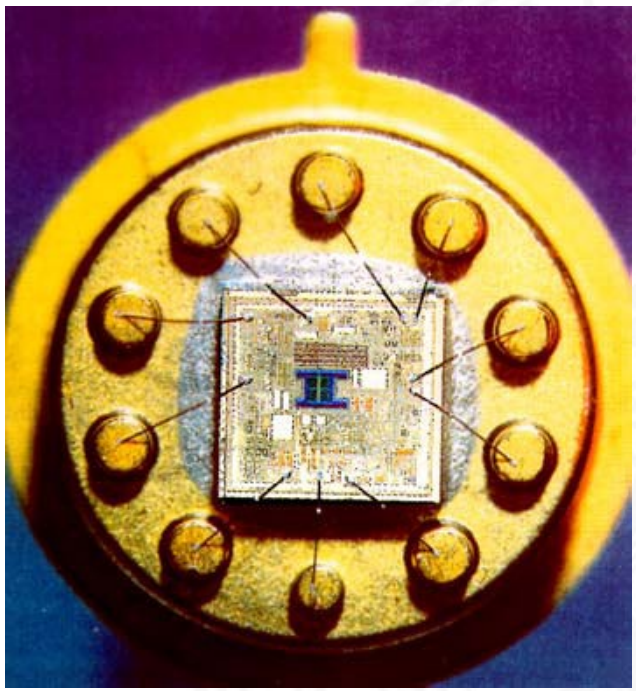
- „Sticking“: Festkleben am Boden durch Adhäsionskräfte
  Problem bei der Herstellung

OMM-Integration

- Prozess ist CMOS-Kompatibel

$\Rightarrow$ Monolithische Integration: MEMS-Komponenten
+ Ansteuerungs- und Auswerteelektronik auf einem Substrat

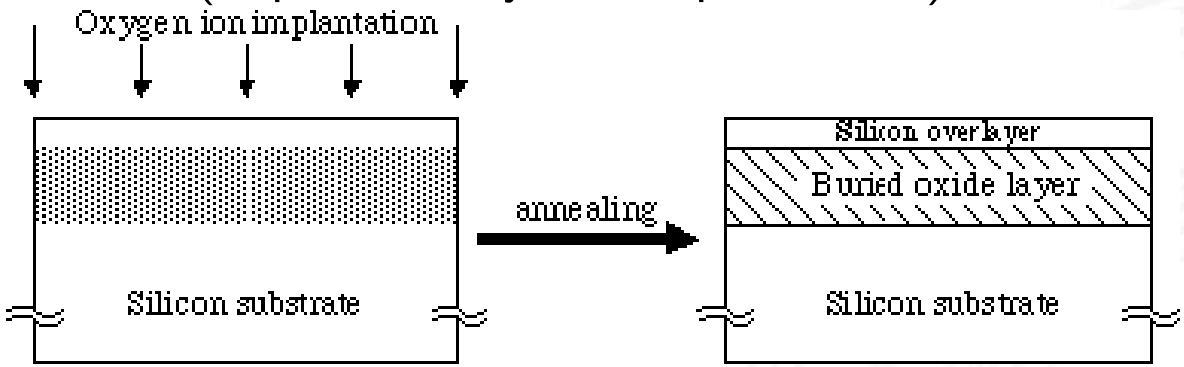




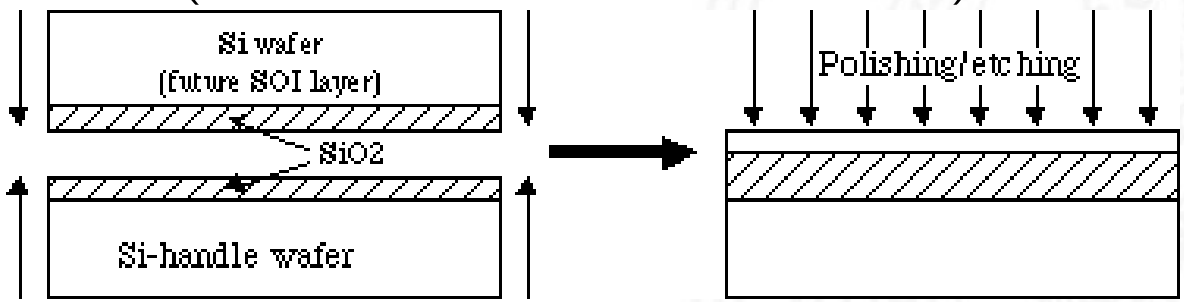
Analog Devices
ADXL-Serie

OMM-Sonderformen

- SOI-Verfahren – **S**ilicon **o**n **I**nsulator
  - SIMOX (Separation by Ion Implantation)



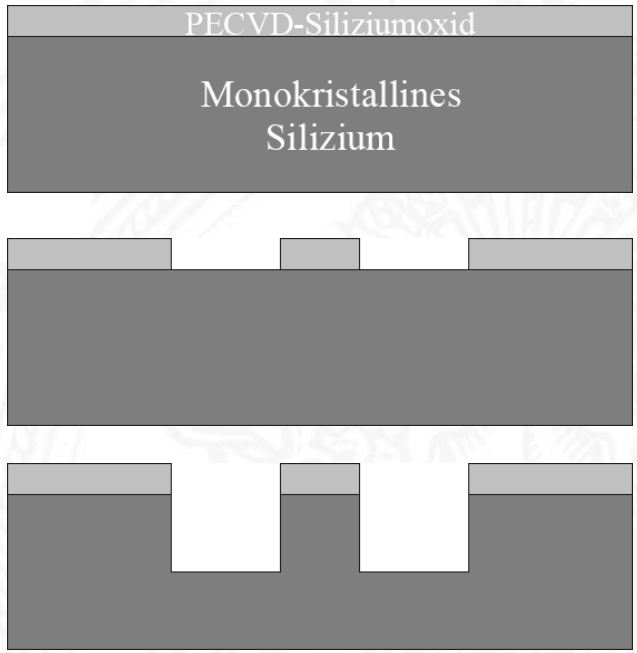
  - BESOI (Back Etched Silicon on Insulator)



OMM-Sonderformen (cont.)

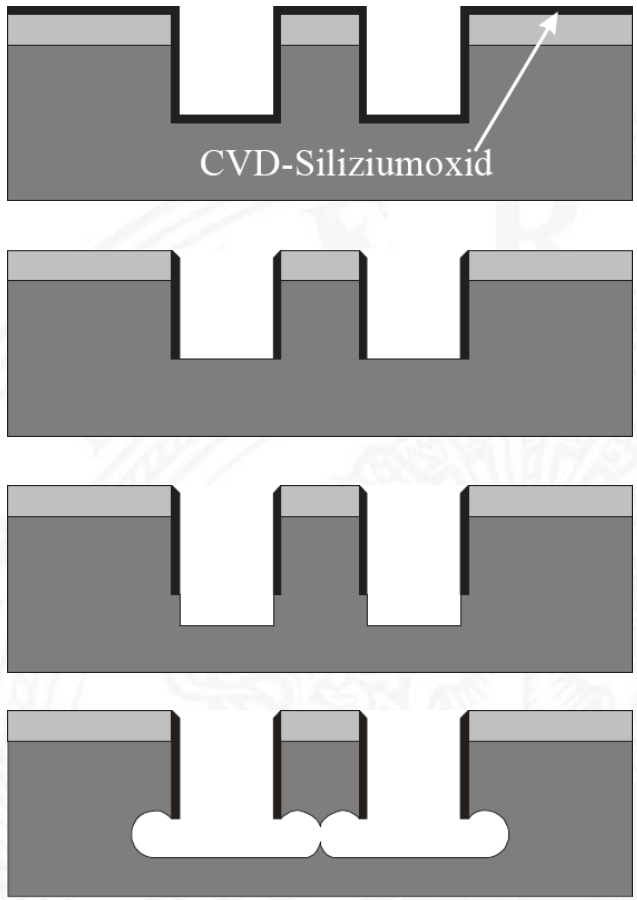
- SCREAM (Single Crystal Reactive Etching and Metallization)
  - ohne Opferschicht
  - Mischung verschiedener Ätzverfahren + Passivierung

1. SiO_2 Abscheiden
2. Strukturieren
3. Reaktives Ionenätzen



OMM-Sonderformen (cont.)

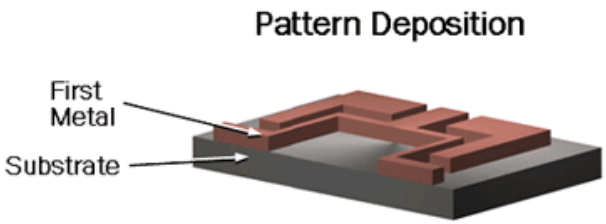
4. Passivierung mit SiO_2
5. Anisotrop horizontal Ätzen
6. Reaktives Ionenätzen
7. Isotropes Ätzen mit Unterätzung



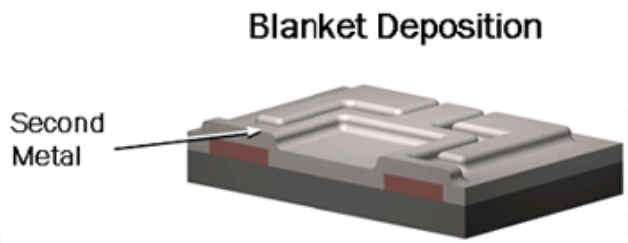
OMM-Sonderformen (cont.)

- 3D-Strukturen „aufeinanderstapeln“
  - mehrfache Abscheidung und Planarisierung von Opfer- und Funktionsschichten
  - Entfernen der Opferschicht am Ende

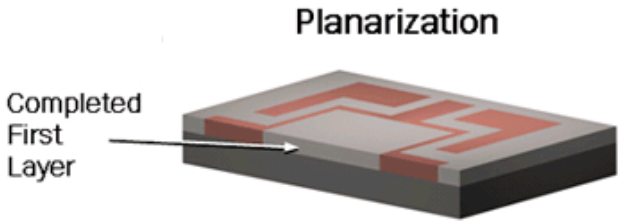
1.



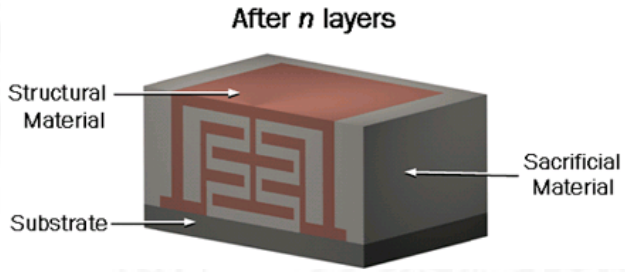
2.



3.

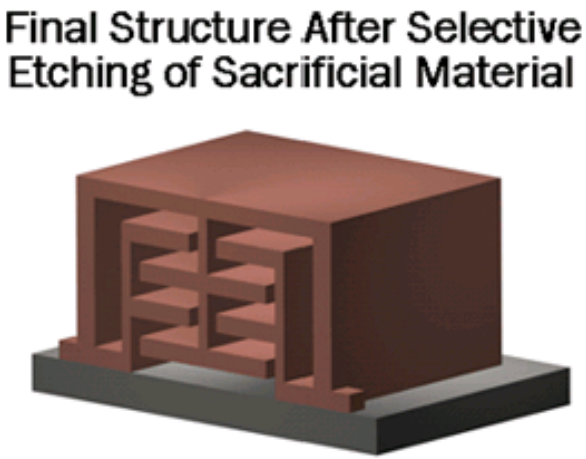


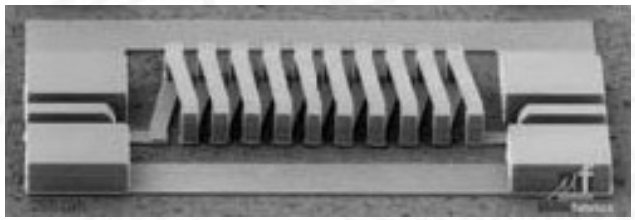
4.

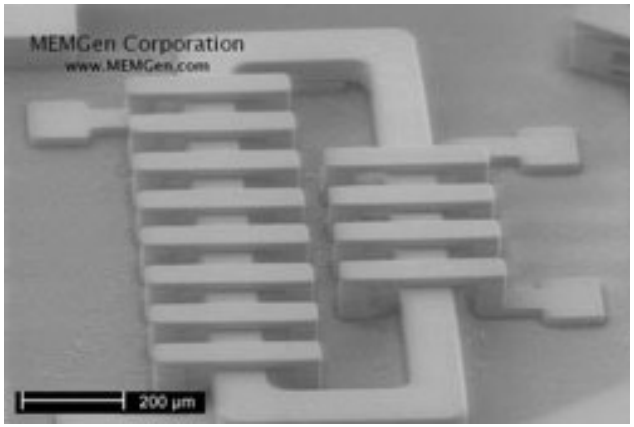


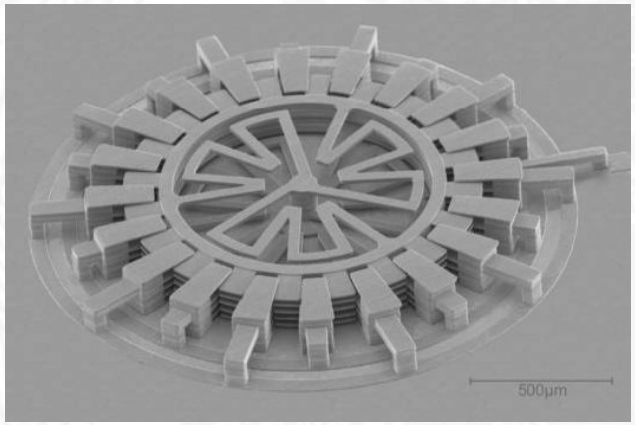
OMM-Sonderformen (cont.)

5.







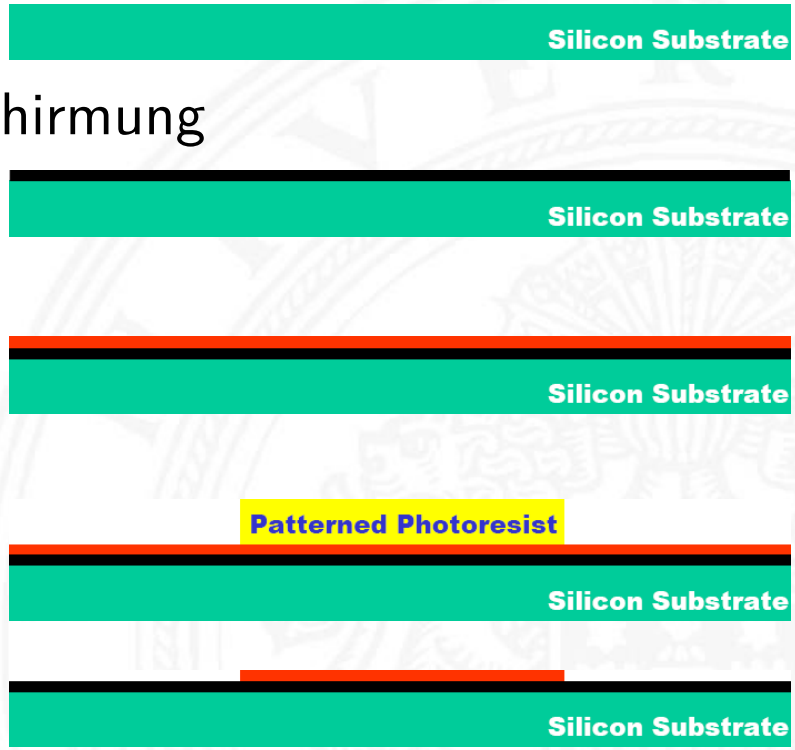


OMM-Beispiele

Elektrostatisch angetriebener Motor

Herstellungsschritte

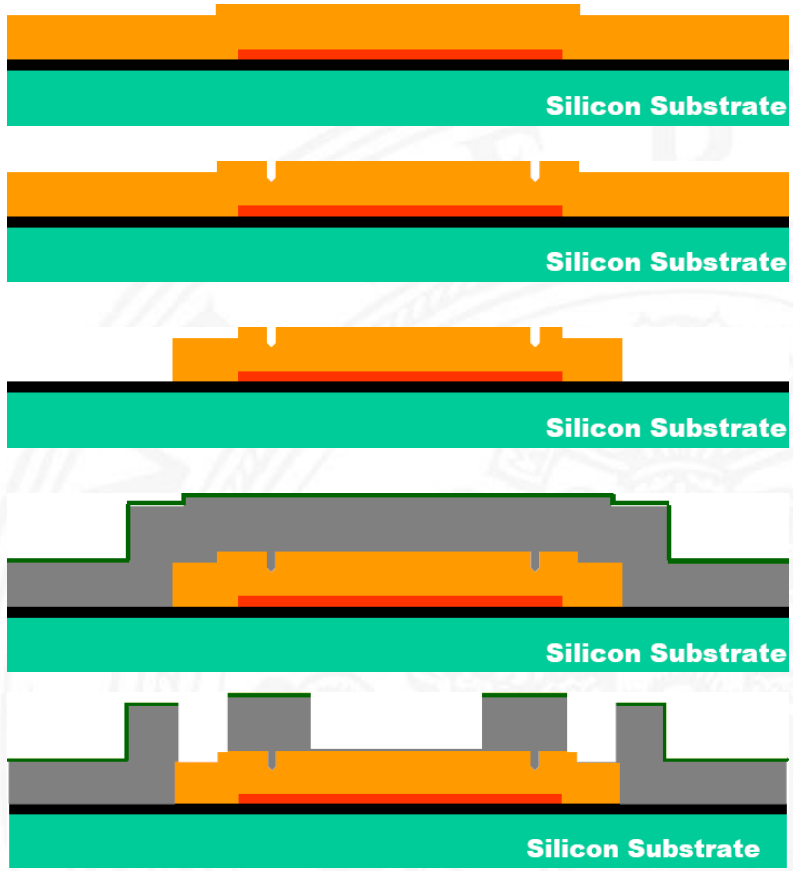
1. n-Substrat, (100)-Orientierung
   p^+-dotiert zur elektrischen Abschirmung
2. Siliziumnitrid (Si_3N_4)
   elektrische Isolation
3. Polysilizium, Poly 0 ($0,5\mu m$)
   elektrische Kontaktierung
4. Lithografie Poly 0
5. Ätzen Poly 0



OMM-Beispiele (cont.)

Elektrostatisch angetriebener Motor

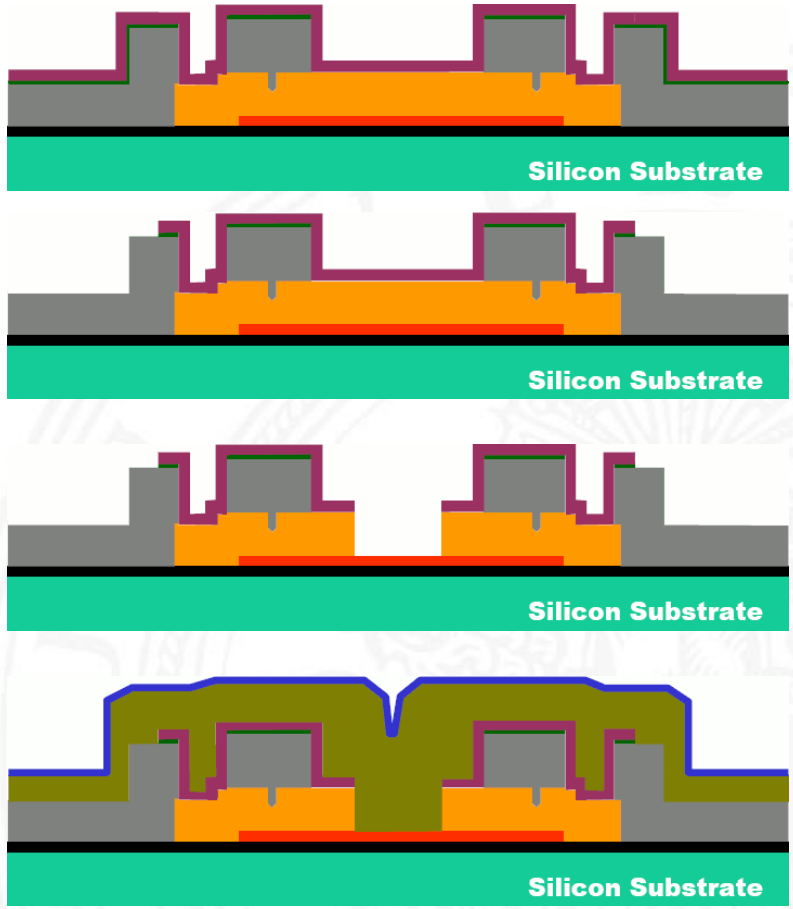
6. SiO_2 Opferschicht $(2,0\mu m)$
7. 1. Strukturierung SiO_2
   Lithografie + Ätzen
8. 2. Strukturierung SiO_2
   Lithografie + Ätzen
9. Polysilizium, Poly 1 $(2,0\mu m)$
   Funktionsschicht - beweglich
10. Strukturierung Poly 1
    Lithografie + Ätzen



OMM-Beispiele (cont.)

Elektrostatisch angetriebener Motor

11. SiO_2 Abscheidung $(0,75\mu m)$
12. Strukturierung SiO_2
    Kontakt zu Poly 1
13. Strukturierung SiO_2
    Kontakt zu Poly 0
14. Polysilizium, Poly 2 $(1,5\mu m)$
    Funktionsschicht - fest



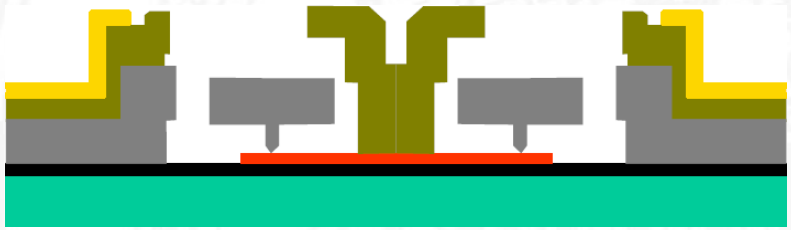
OMM-Beispiele (cont.)

Elektrostatisch angetriebener Motor

15. Strukturierung Poly 2
    Lithografie + Ätzen

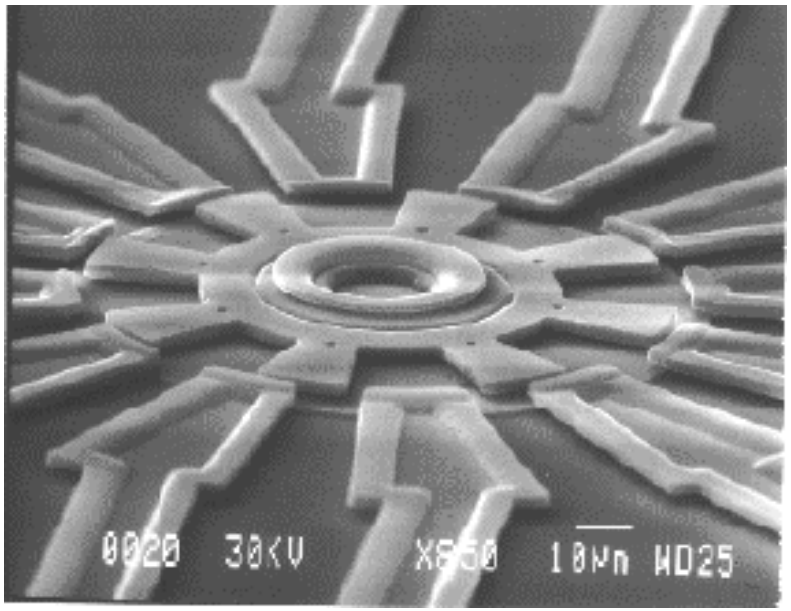
16. Abscheidung und Strukturierung
    Metall ($0,5\mu m$)

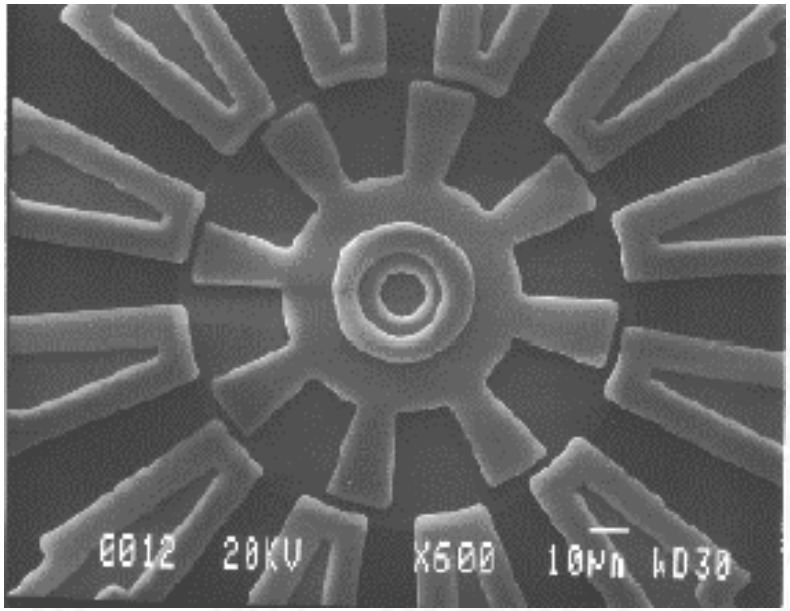
17. SiO_2 Opferschicht entfernen



OMM-Beispiele (cont.)

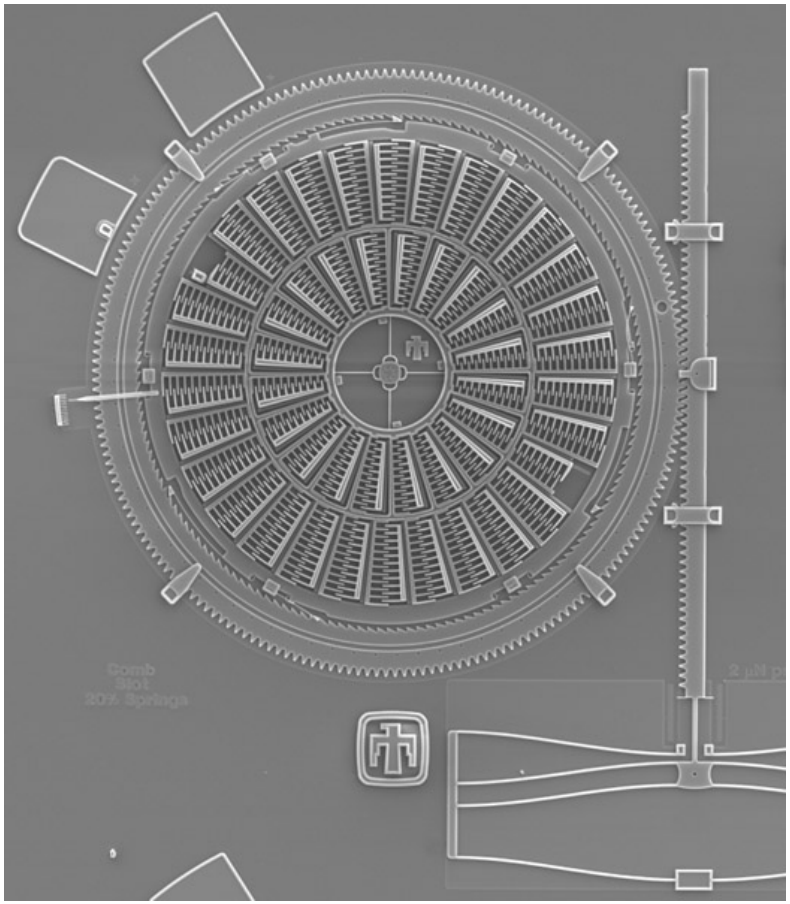
Elektrostatisch angetriebener Motor

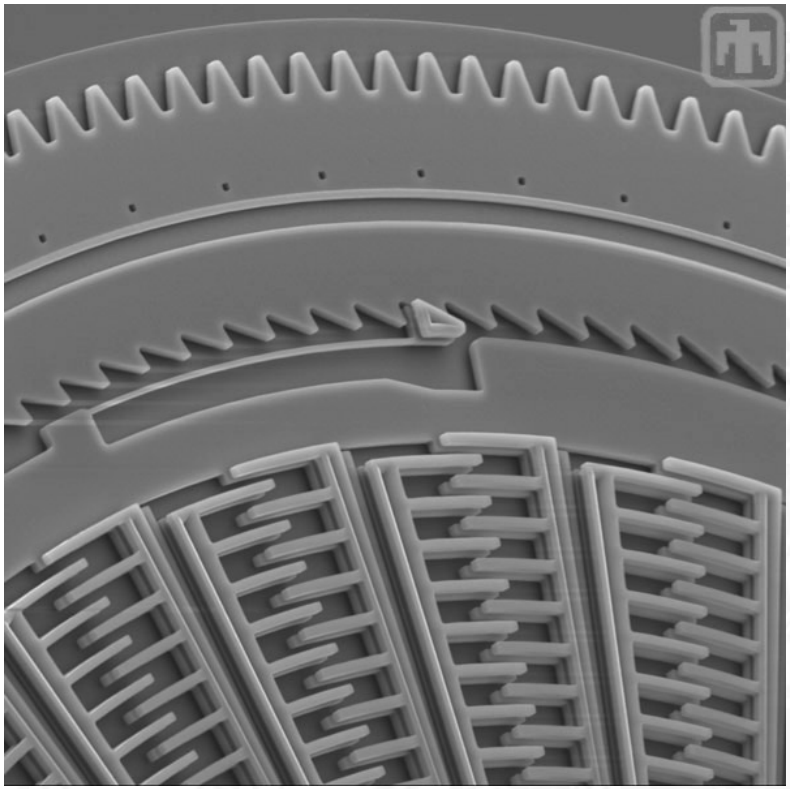




OMM-Beispiele

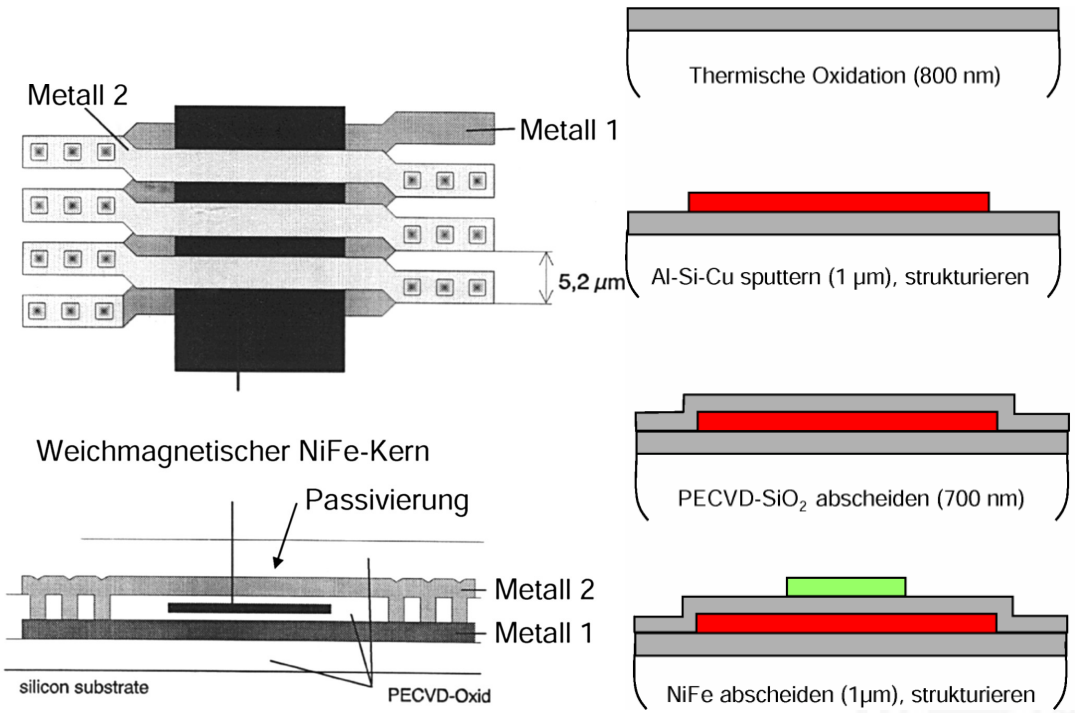
Elektrostatisch oszillierender Motor





OMM-Beispiele

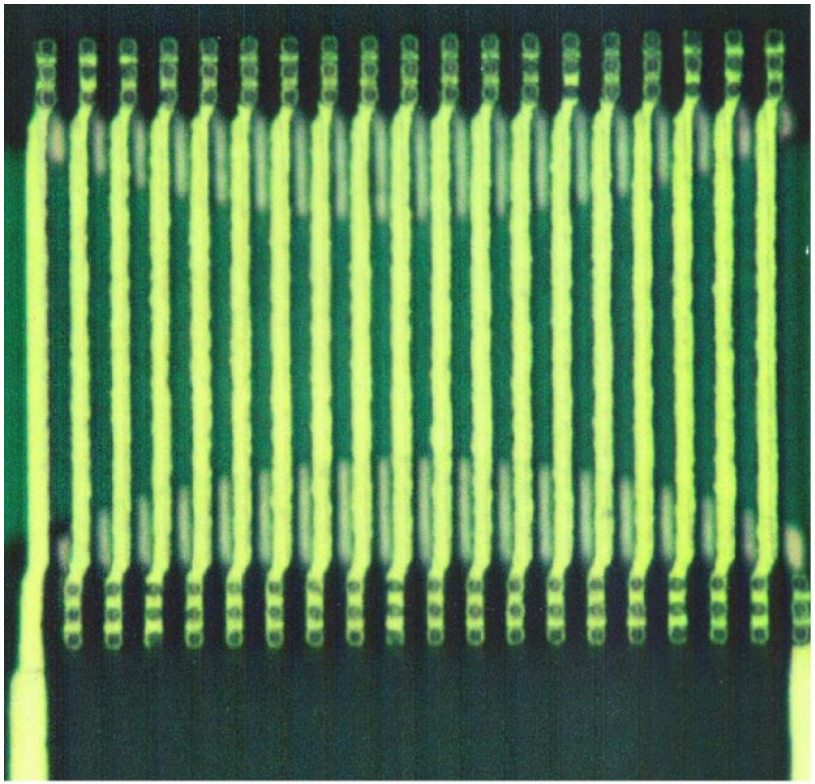
Magnetfeldsensor



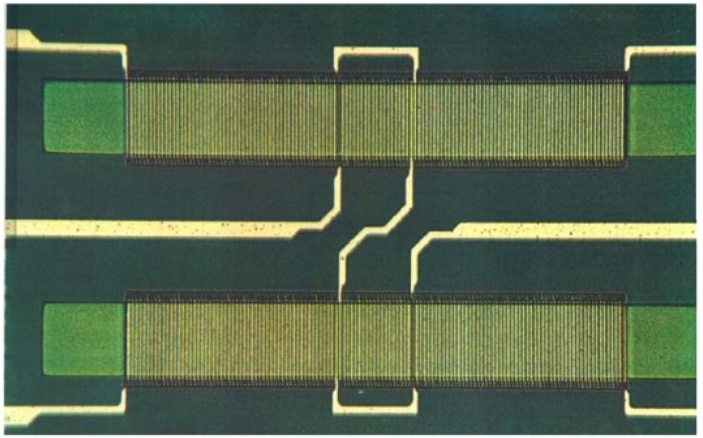
Bis auf die Herstellung des Eisenkerns ist dieser Prozess ähnlich wie die Herstellung einer zweilagigen Verdrahtung für ein Mikrobauteil oder eine Schaltung.

OMM-Beispiele (cont.)

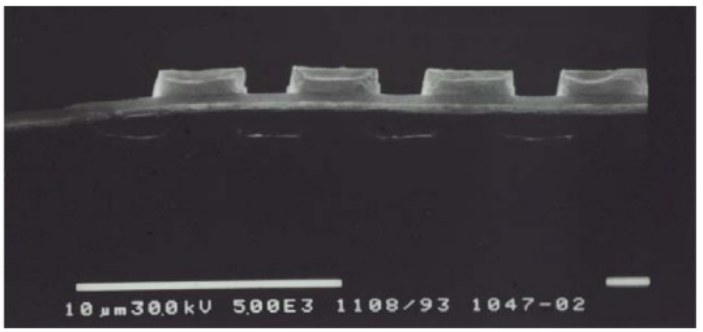
Magnetfeldsensor



Quelle: FhG-IMS



1-Achsen Magnetfeldsensor (Förstersonde)

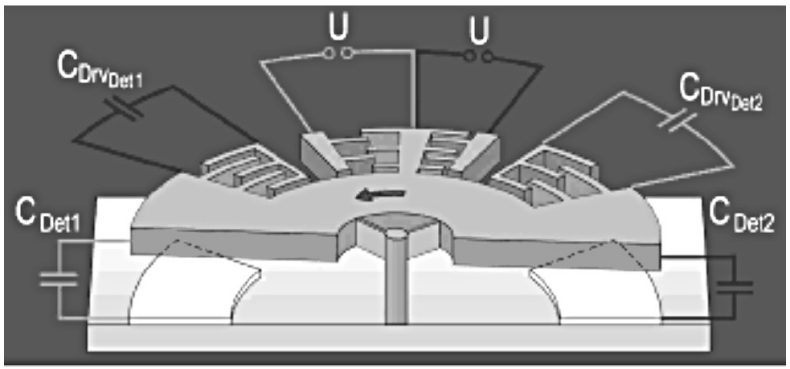


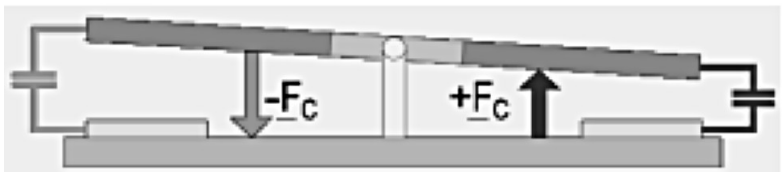
Spulenquerschnitt

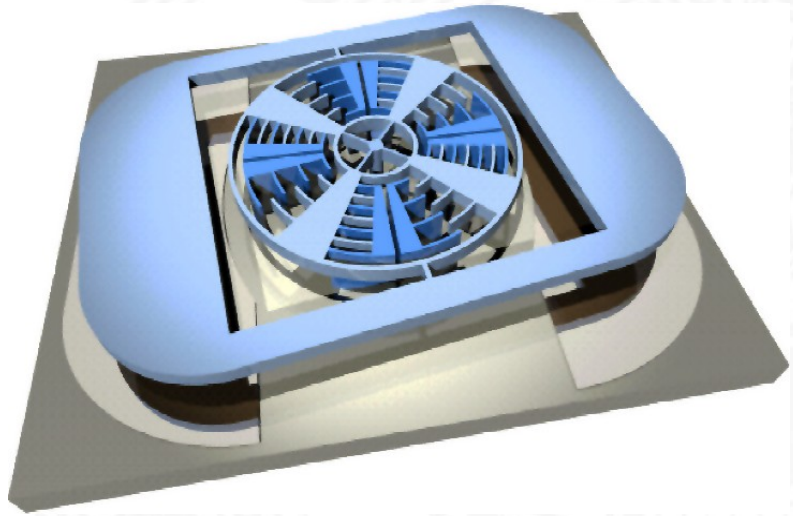
OMM-Beispiele

Drehratensensor

- schwingende Struktur
- Sekundärschwingung durch äußere Drehbewegung
- kapazitive Auswertung

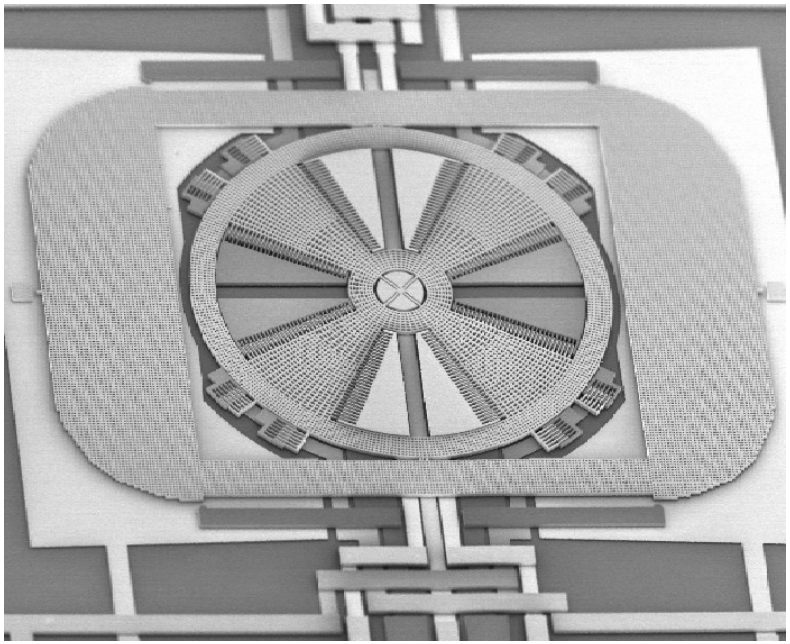


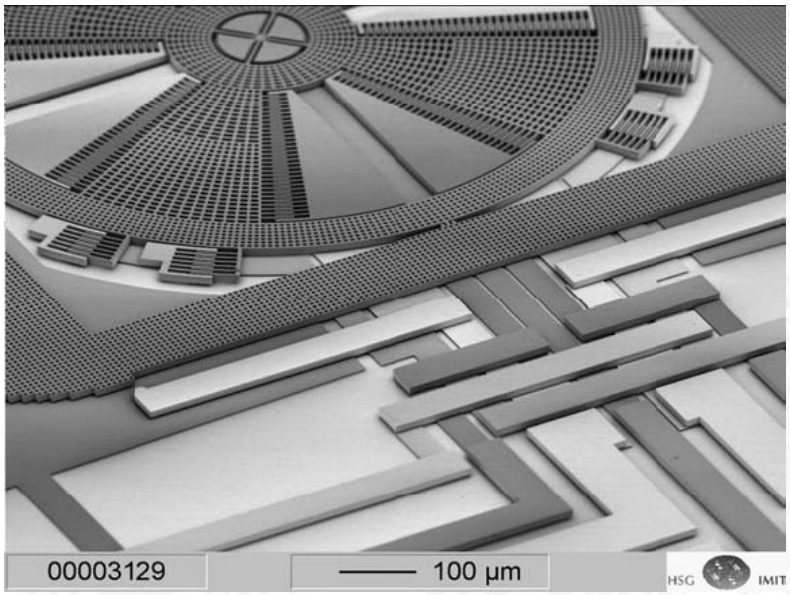




OMM-Beispiele (cont.)

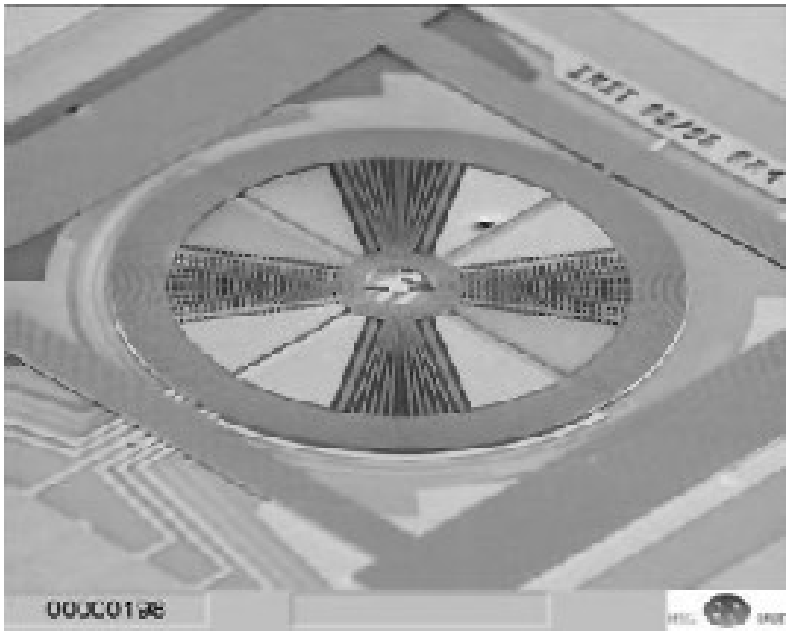
Drehratensensor

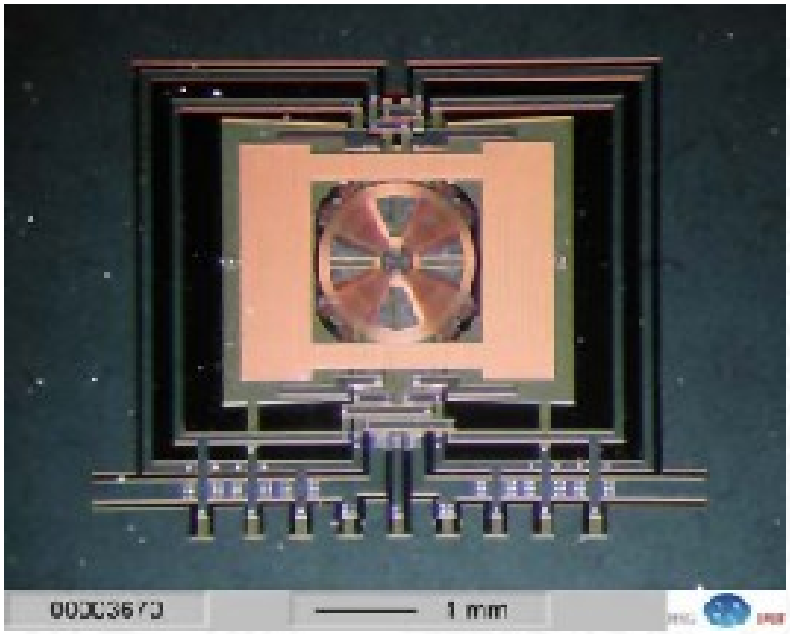




OMM-Beispiele (cont.)

Drehratensensor

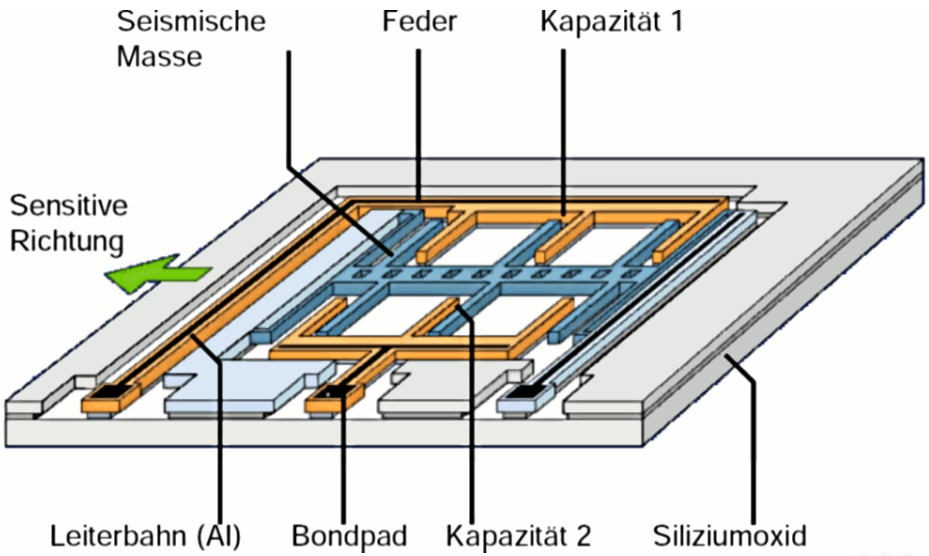


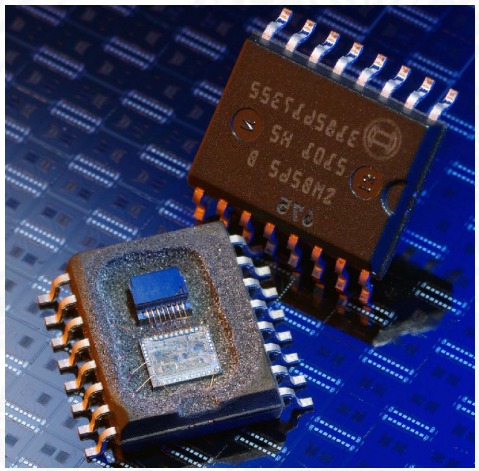


OMM-Beispiele

Beschleunigungssensor 1

- schwingende Struktur
- Überlagerung bei äußerer Beschleunigung
- kapazitive Auswertung $a \approx \frac{C_1 - C_2}{C_1 + C_2}$





OMM-Beispiele (cont.)

Beschleunigungssensor 1

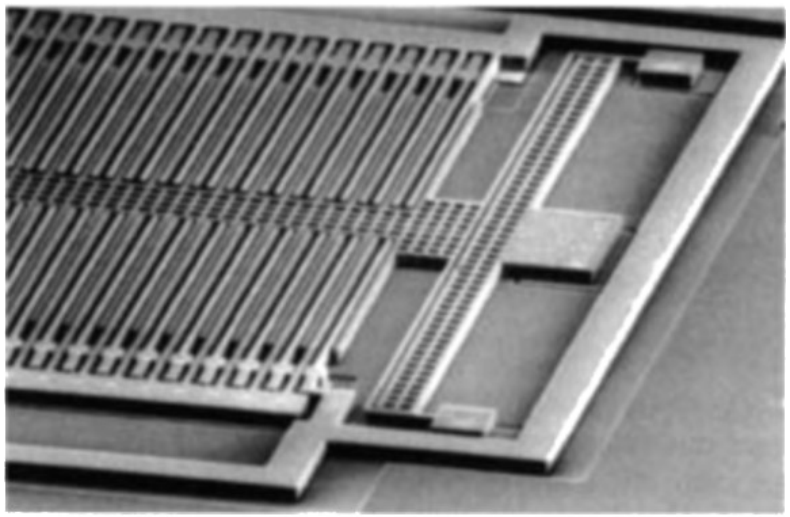
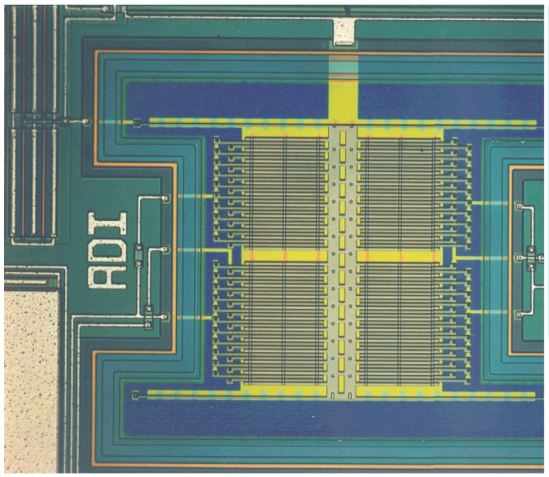
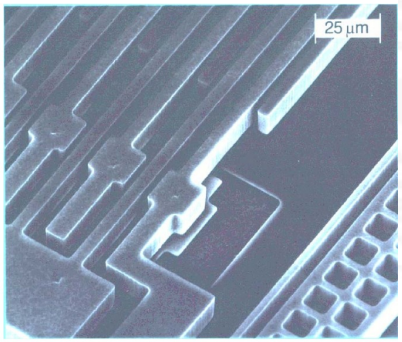


Figure 1: Accelerometer, photo courtesy Bosch

Sensorstruktur



Detail der Elektrodenstruktur



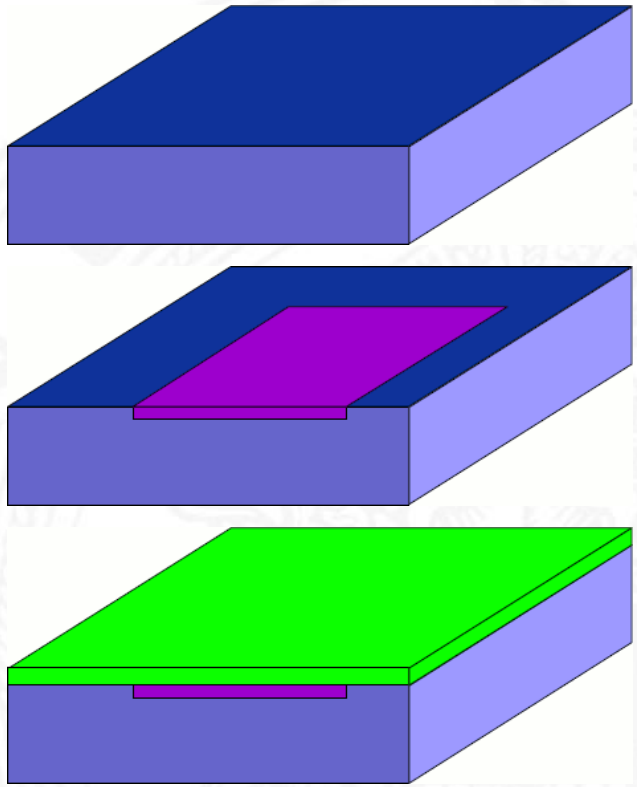
Quelle: Bosch, Motorola

OMM-Beispiele

Beschleunigungssensor 2

Massenträgheit ausnutzen (vergl. Bulk-Mikromechanik)

1. Silizium Wafer
2. Diffusion von Elektroden
3. Opferschicht abscheiden

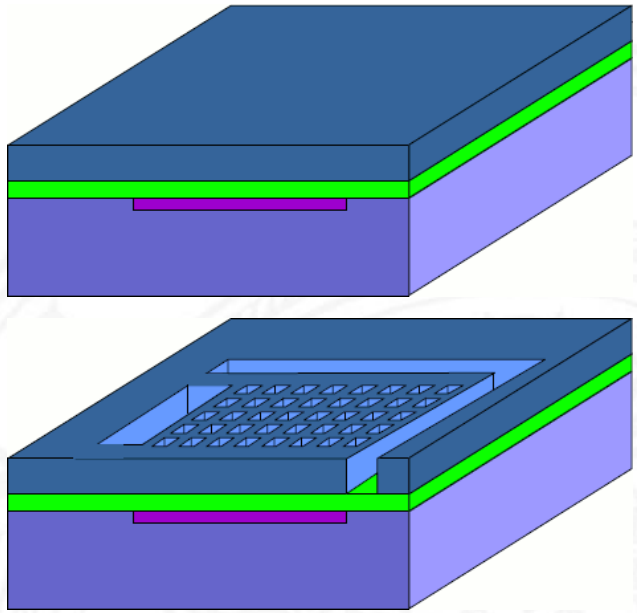


OMM-Beispiele (cont.)

Beschleunigungssensor 2

4. Aktive Schicht abscheiden

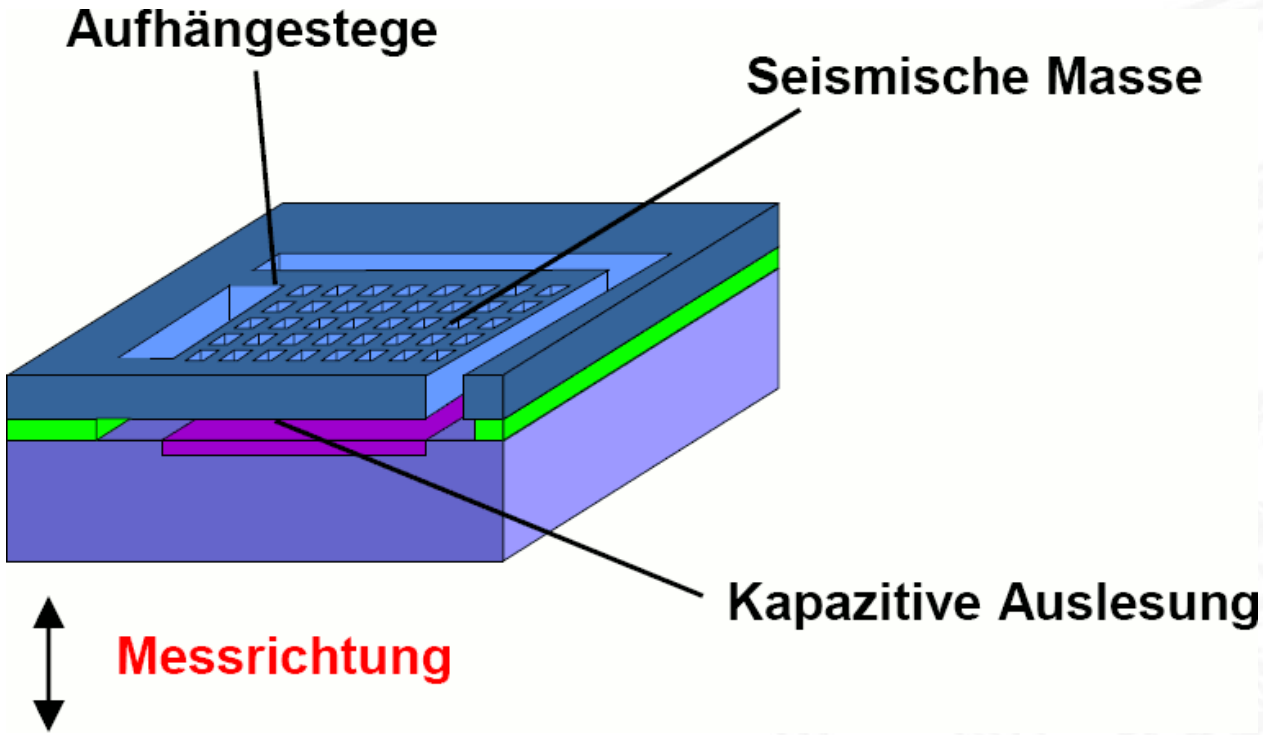
5. Aktive Schicht strukturieren



OMM-Beispiele (cont.)

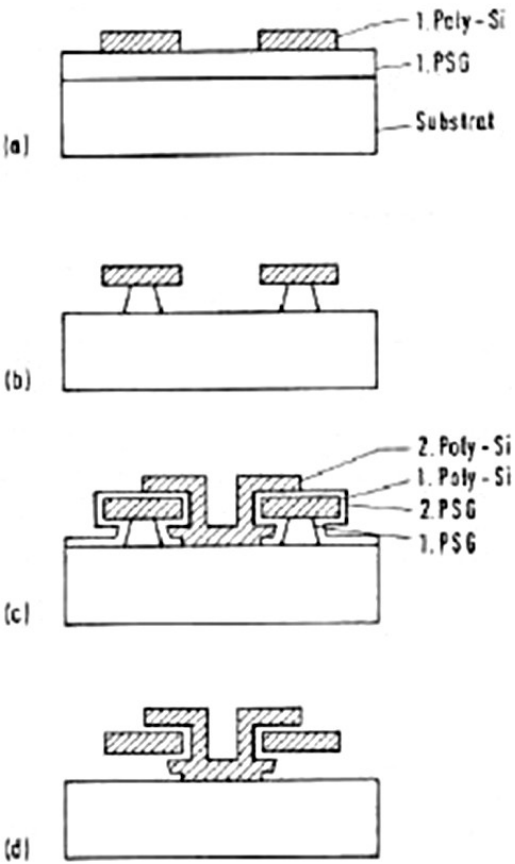
Beschleunigungssensor 2

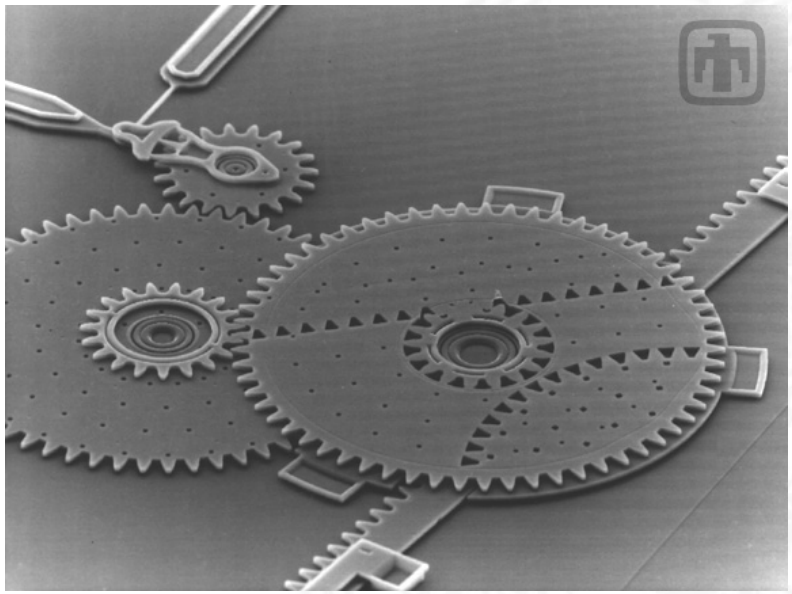
6. Opferschicht selektiv entfernen



OMM-Beispiele

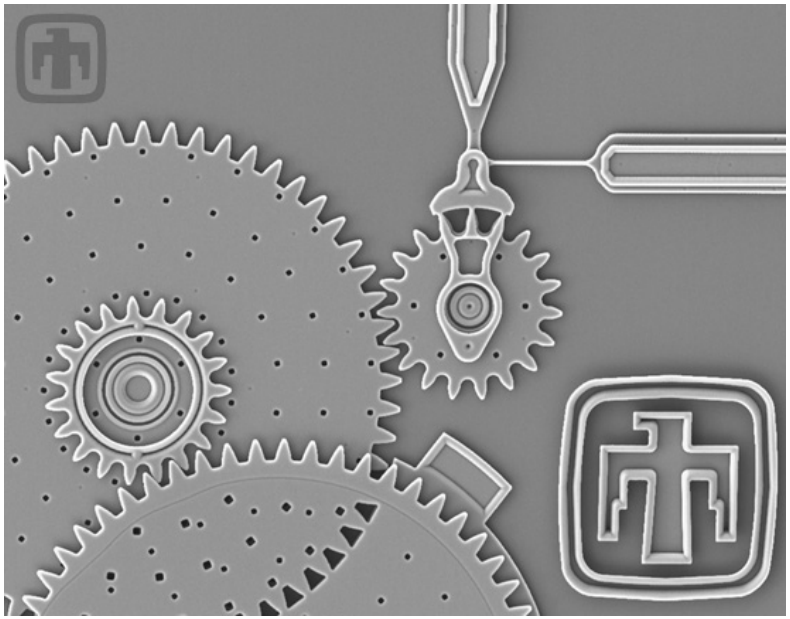
Zahnräder und Getriebe

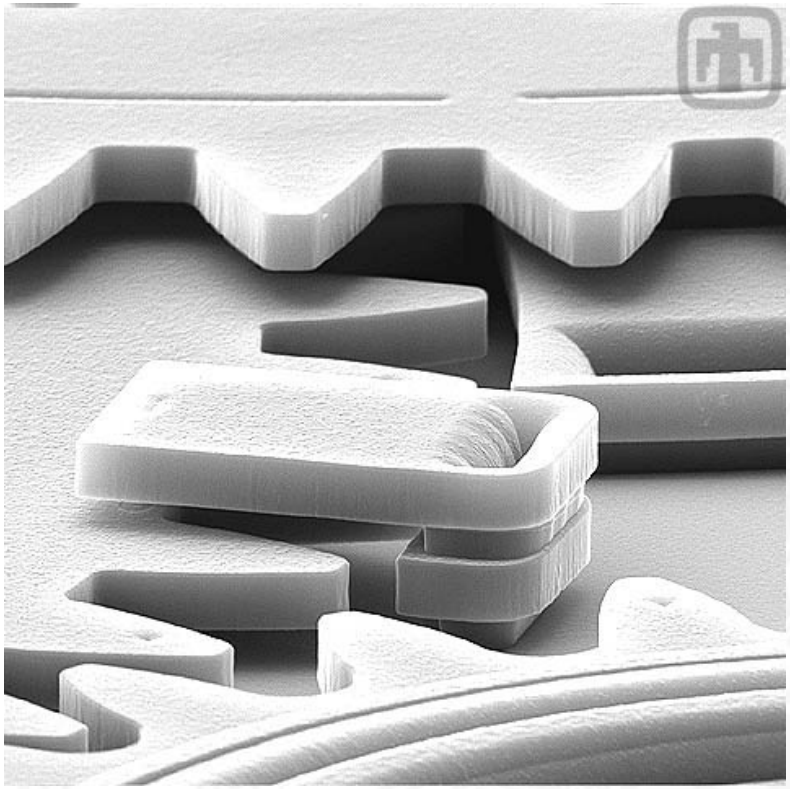




OMM-Beispiele (cont.)

Zahnräder und Getriebe



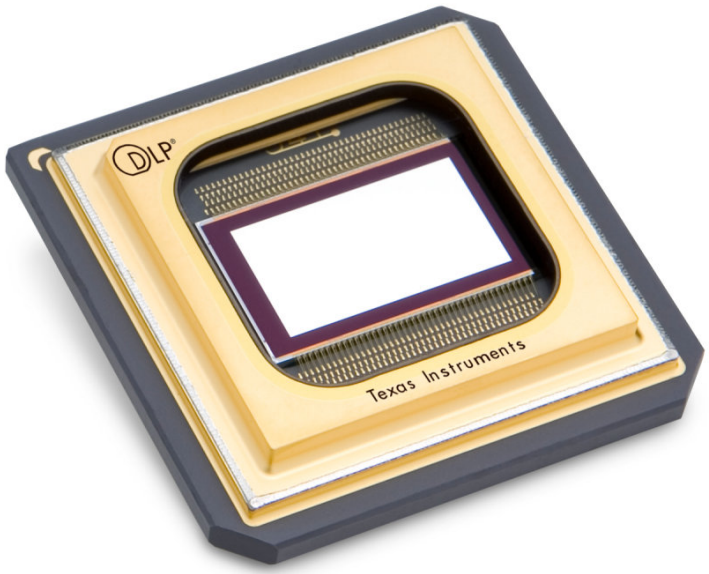


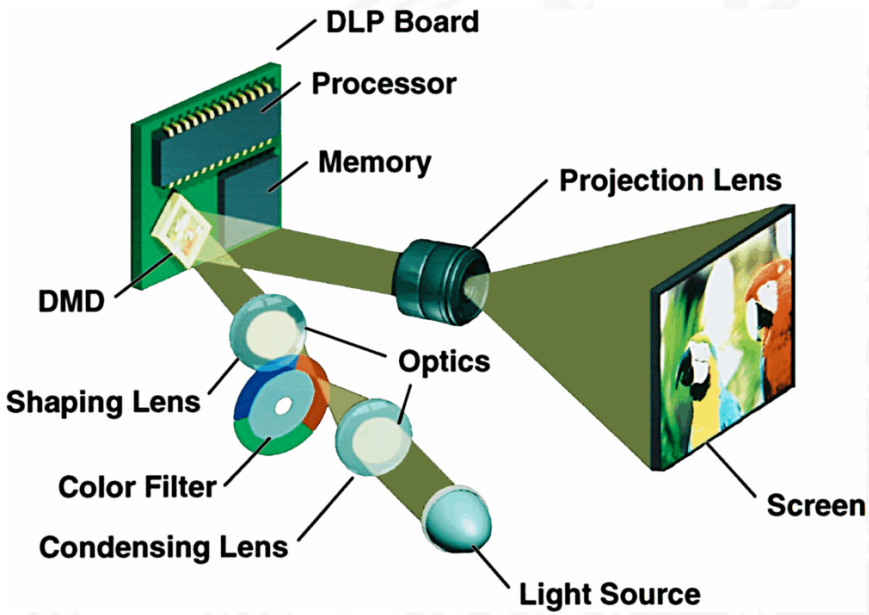
OMM-Beispiele

Digital Mirror Device

- TI-Spiegelchips in Beamern

http://www.dlp.com

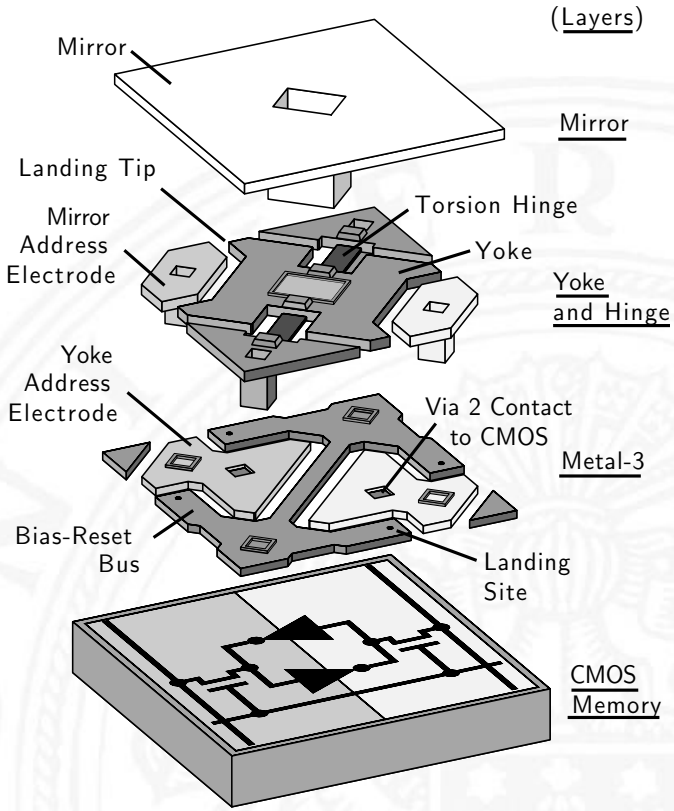




OMM-Beispiele (cont.)

Digital Mirror Device

- 3-lagiger Aufbau
  1. Spiegelebene
  2. Trägerebene mit Torsionsbalken
  3. Elektrodenebene

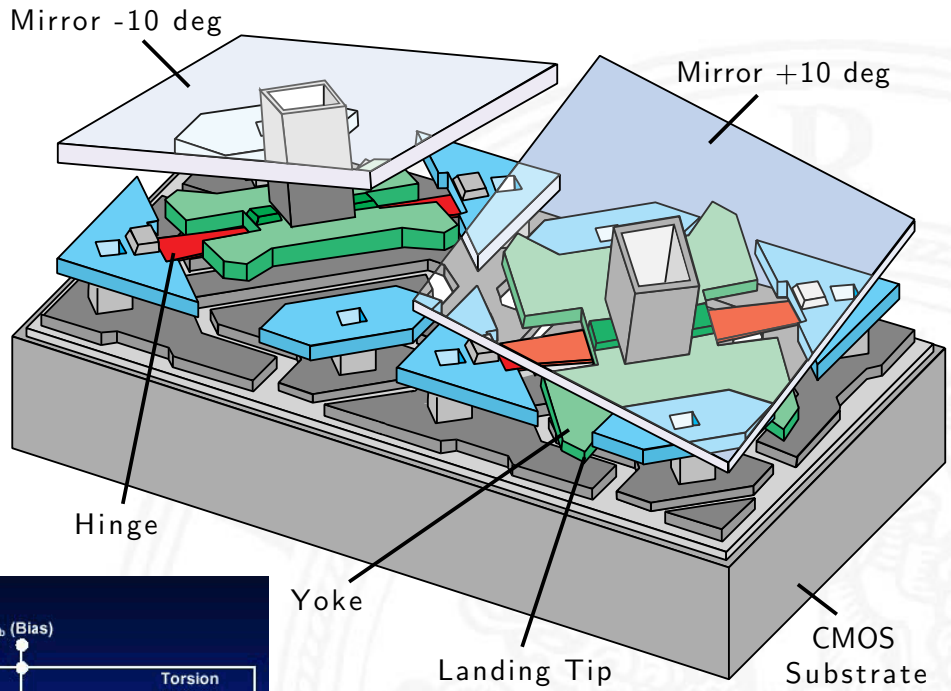


- Ansteuerung als SRAM-Zelle
- Steuerelektronik direkt auf Trägersubstrat der Mikromechanik „Monolithische Integration“

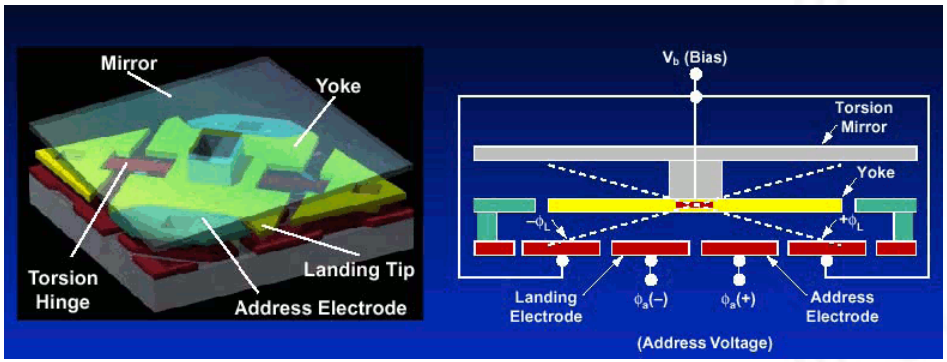
OMM-Beispiele (cont.)

Digital Mirror Device

- Kipp-Bewegung ± 10 deg



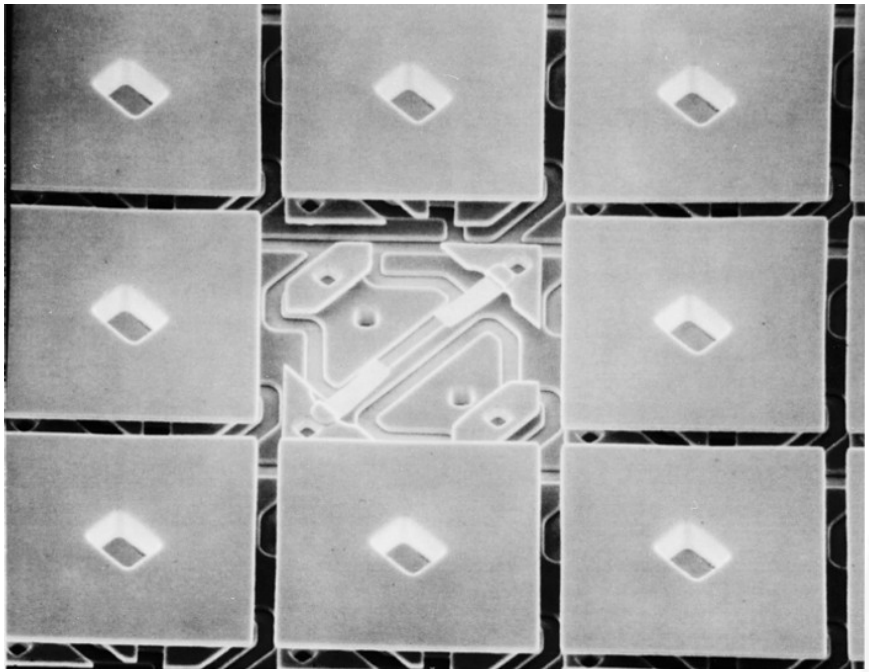
- elektrostatischer Antrieb

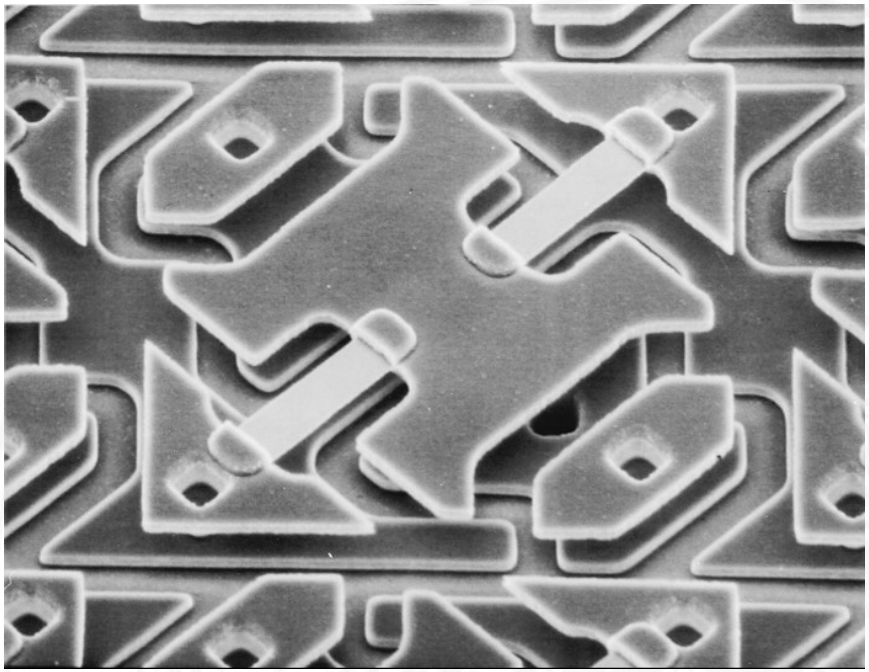


OMM-Beispiele (cont.)

Digital Mirror Device

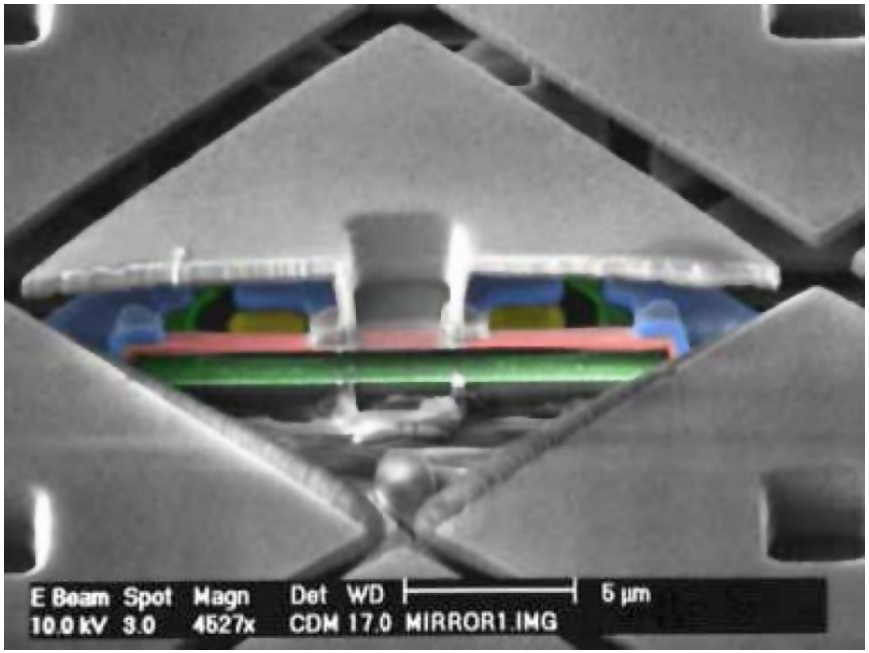
Detailstruktur

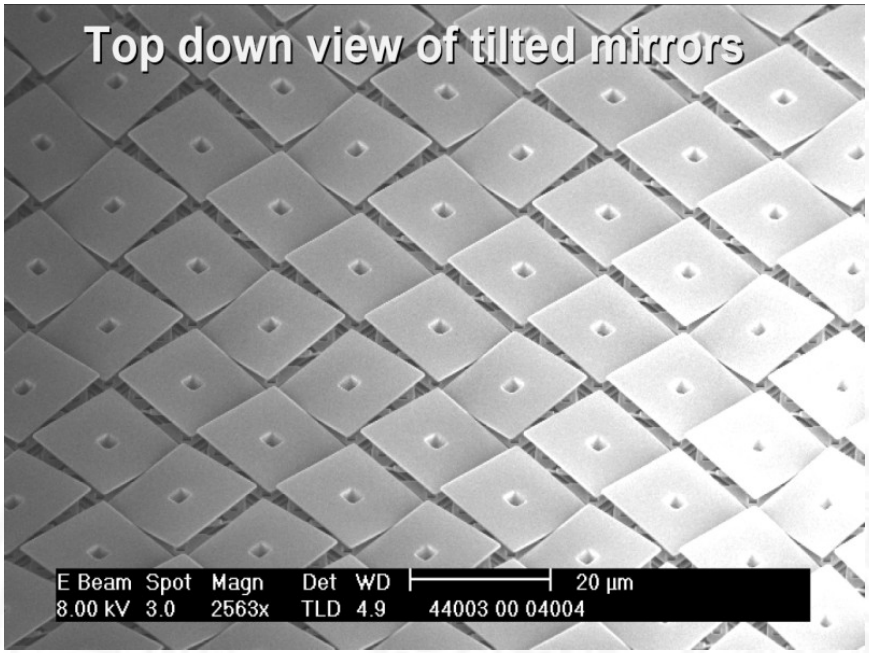




OMM-Beispiele (cont.)

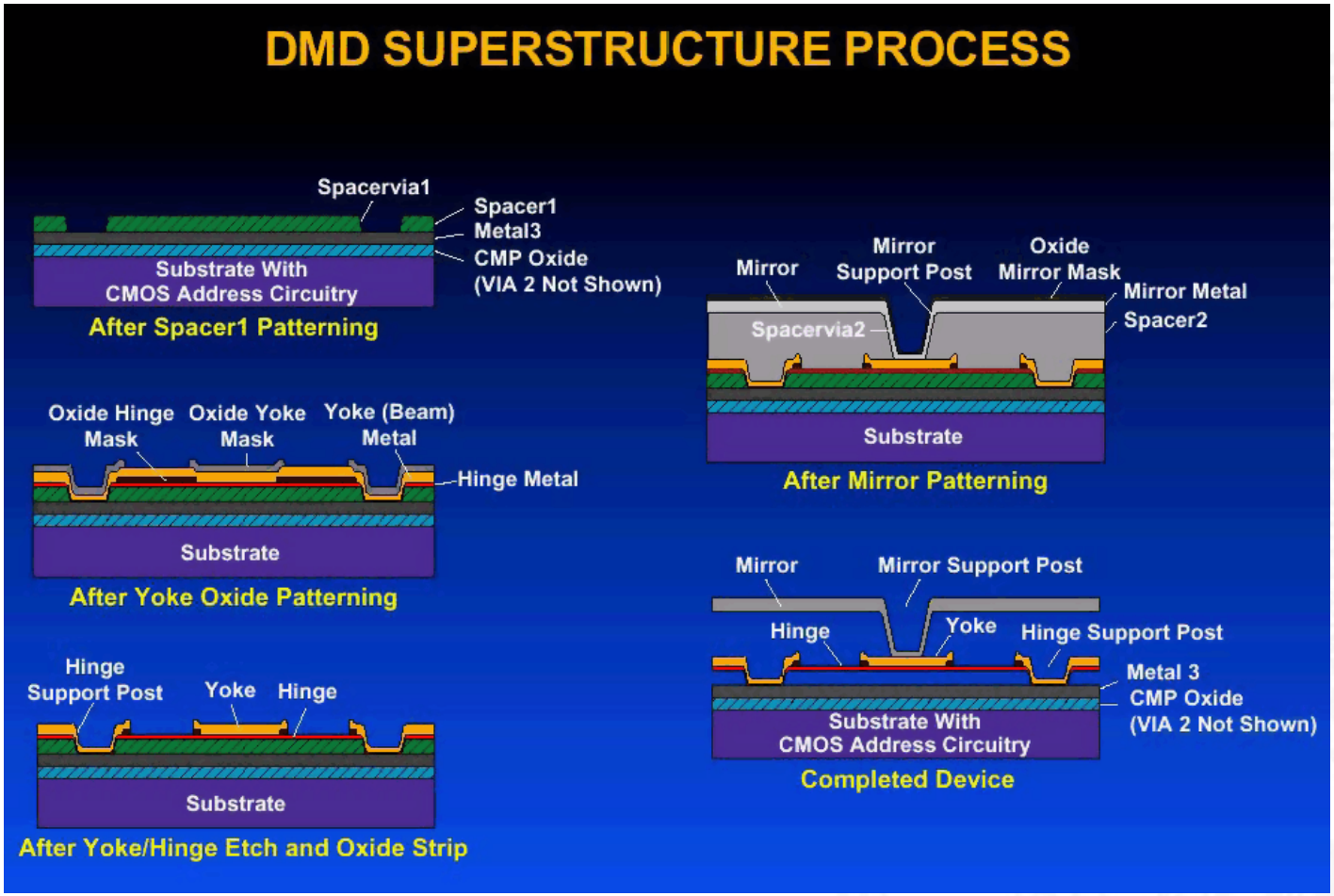
Digital Mirror Device





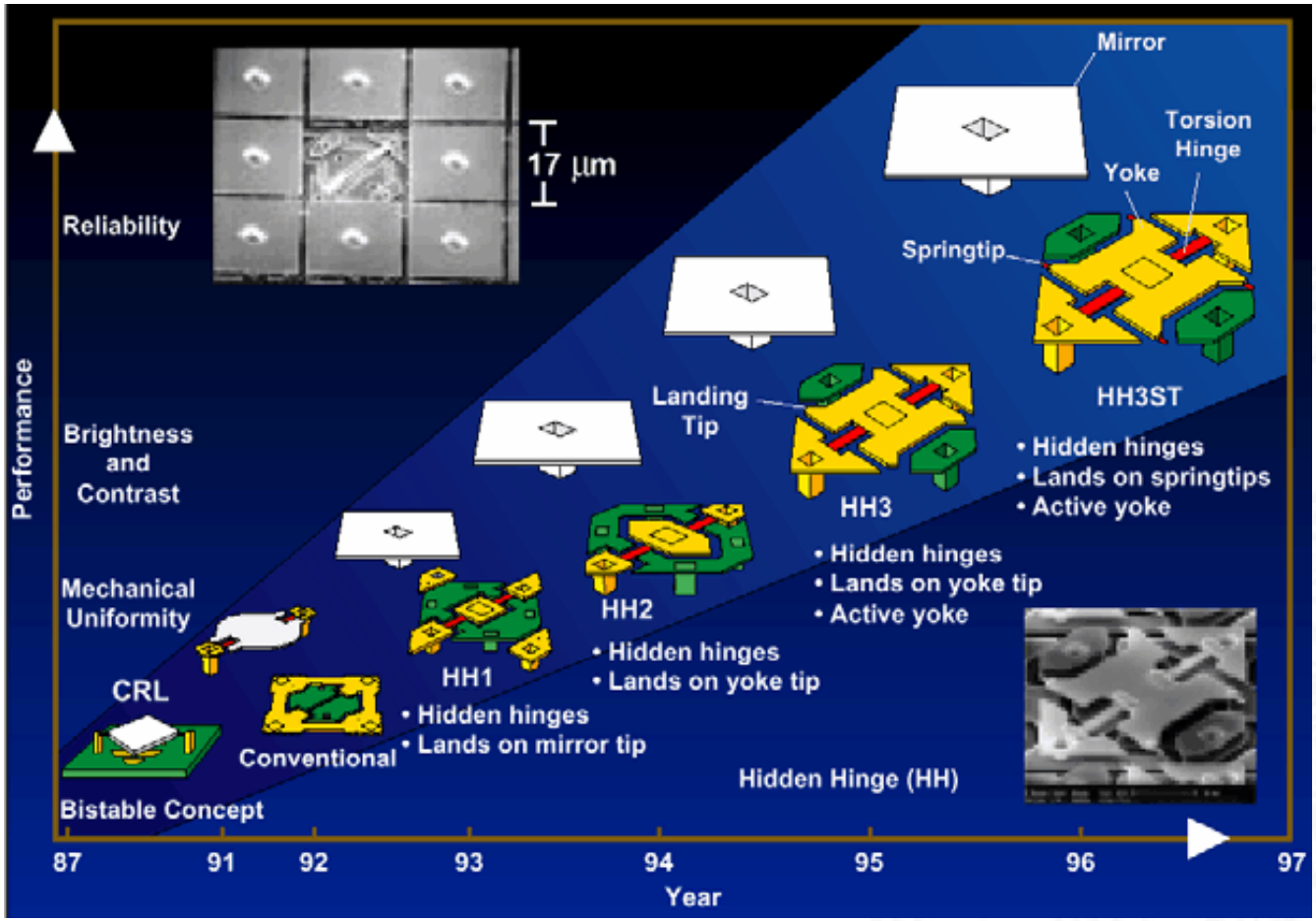
OMM-Beispiele (cont.)

Digital Mirror Device



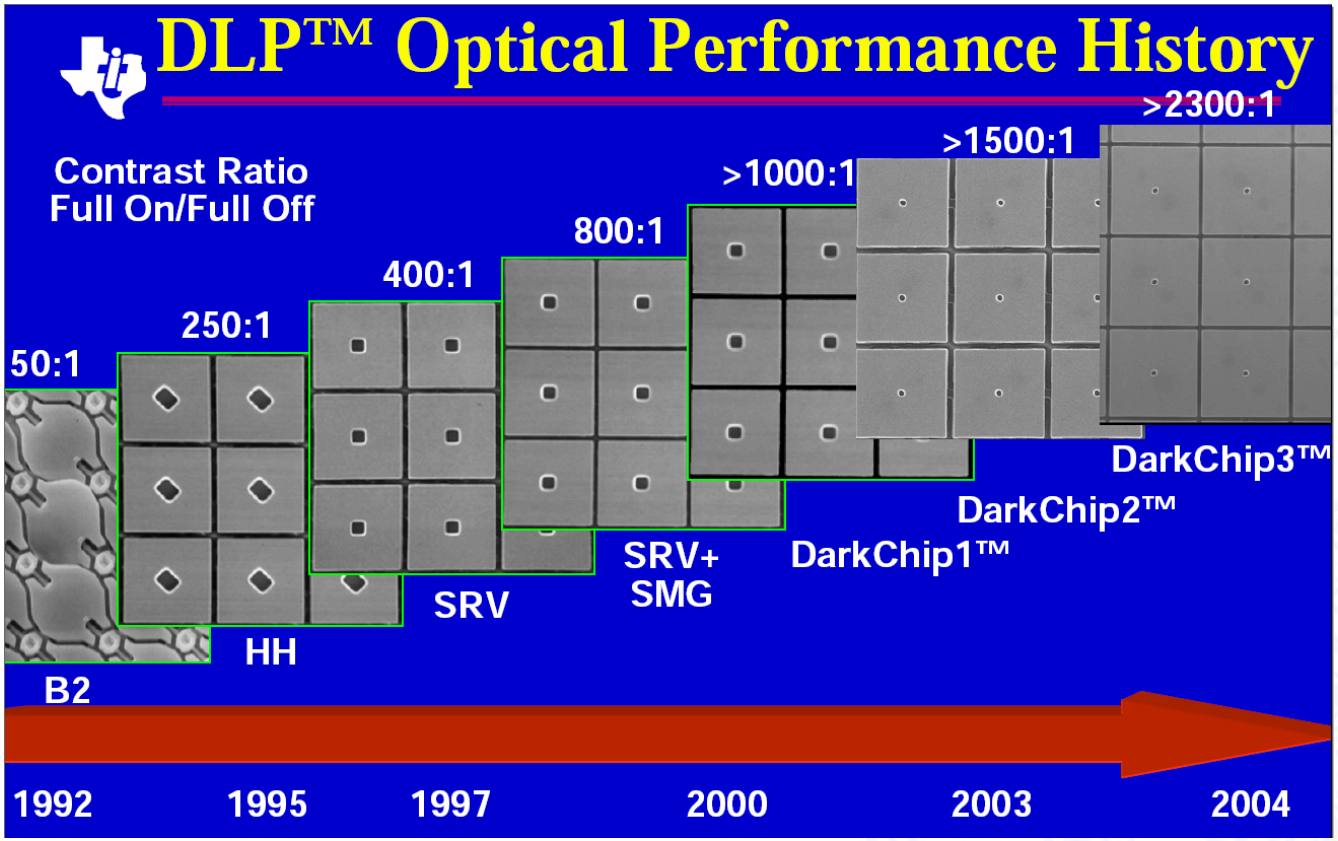
OMM-Beispiele (cont.)

Digital Mirror Device



OMM-Beispiele (cont.)

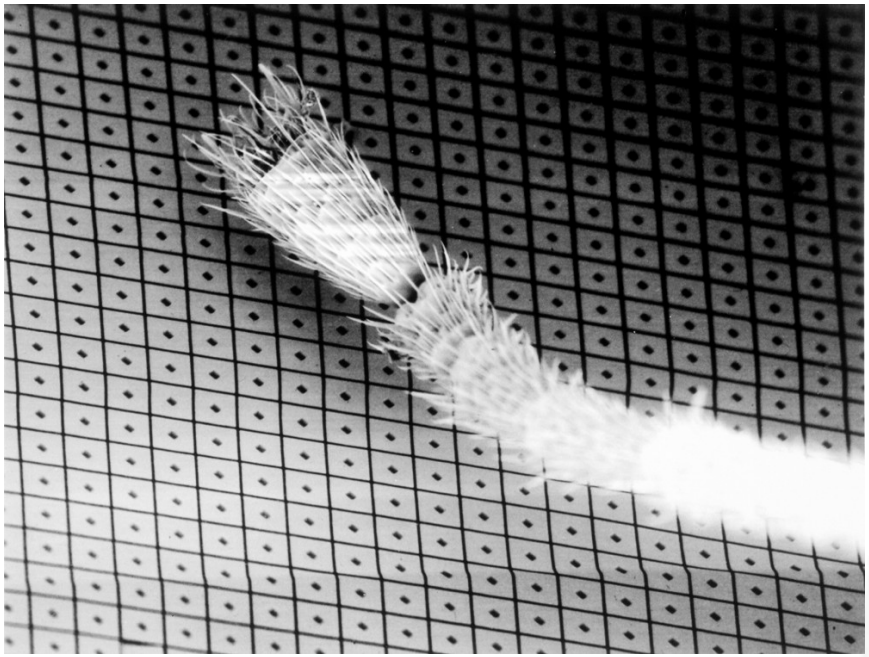
Digital Mirror Device

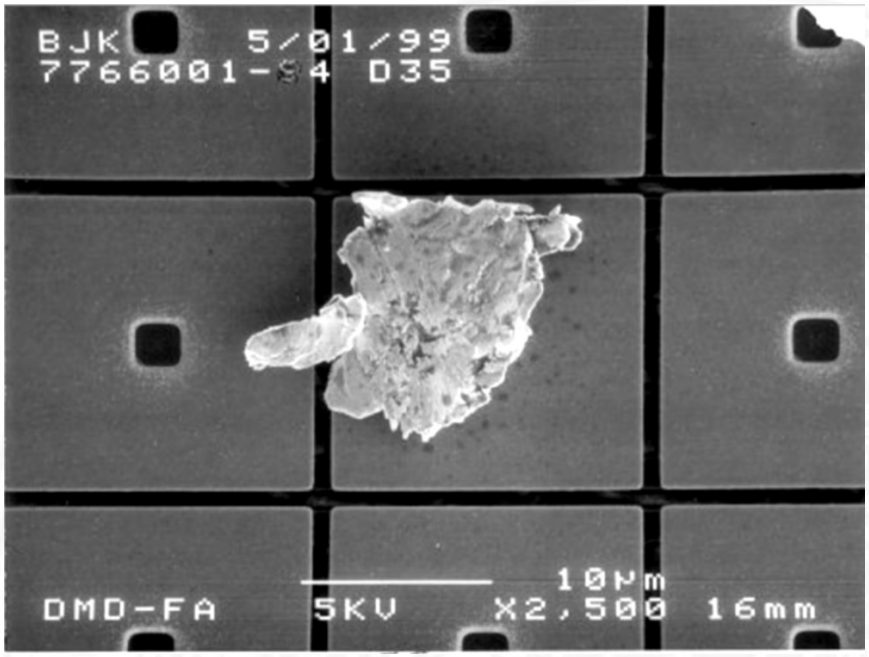


OMM-Beispiele (cont.)

Digital **M**irror **D**evice

Größenvergleich



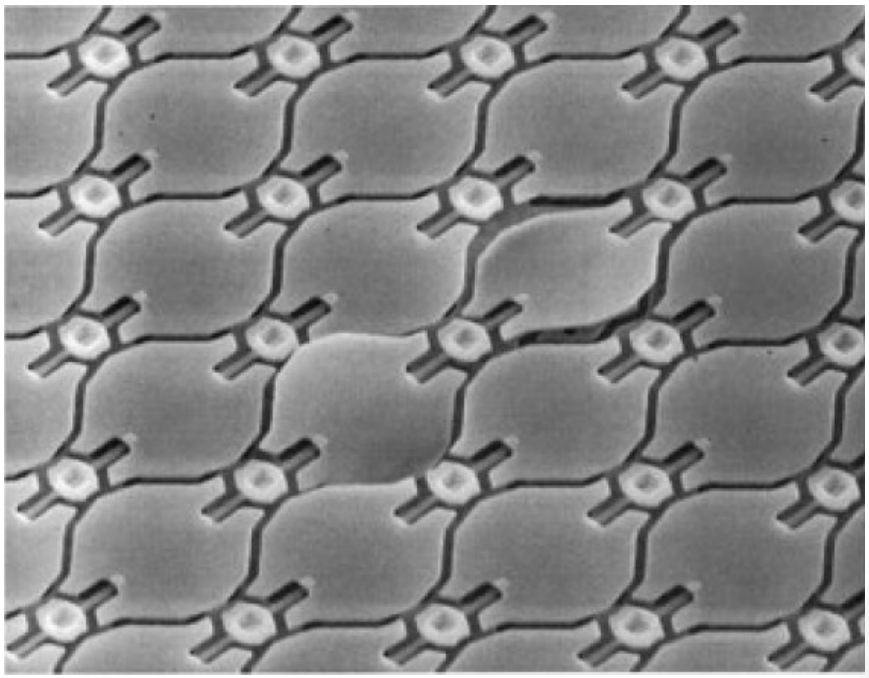


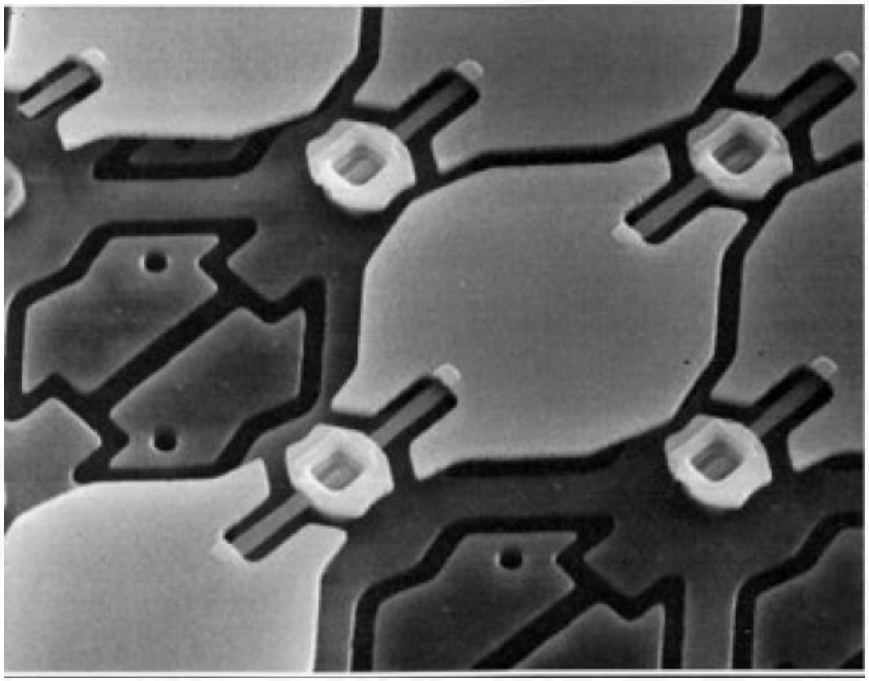
– Problem: Staub

OMM-Beispiele (cont.)

Digital Mirror Device

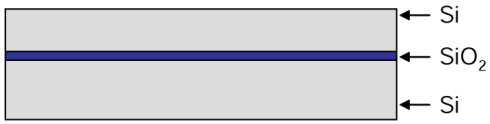
andere Bauformen



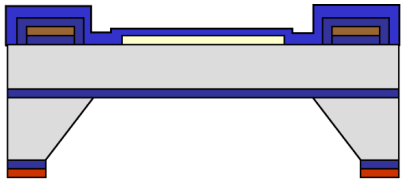


OMM-Beispiele (cont.)

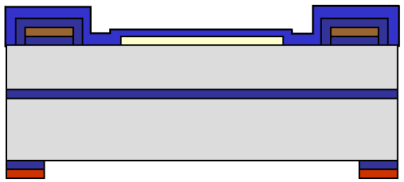
Digital Mirror Device



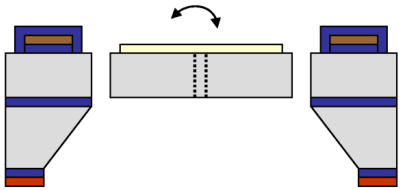
Si-Epitaxie auf einem SIMOX-Wafer (30 µm)



anisotropes nasschemisches Ätzen von Si

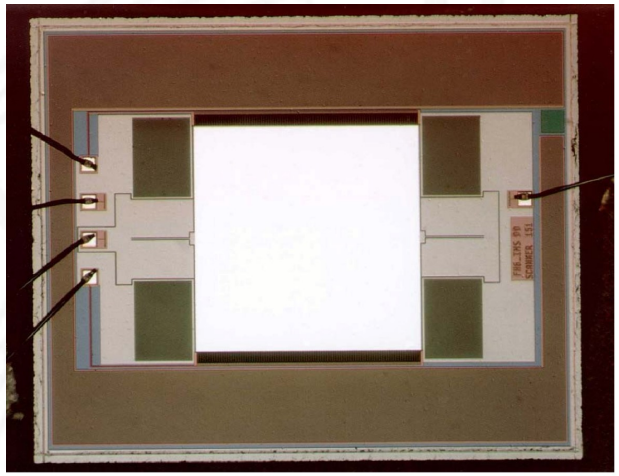


Thermische Oxidation, strukturieren, Al-Si-Cu sputtern, strukturieren, PECVD-SiO_2 abscheiden Vorderseite



PECVD-SiO_2 strukturieren
anisotropes Trockenätzen von Si (RIE)

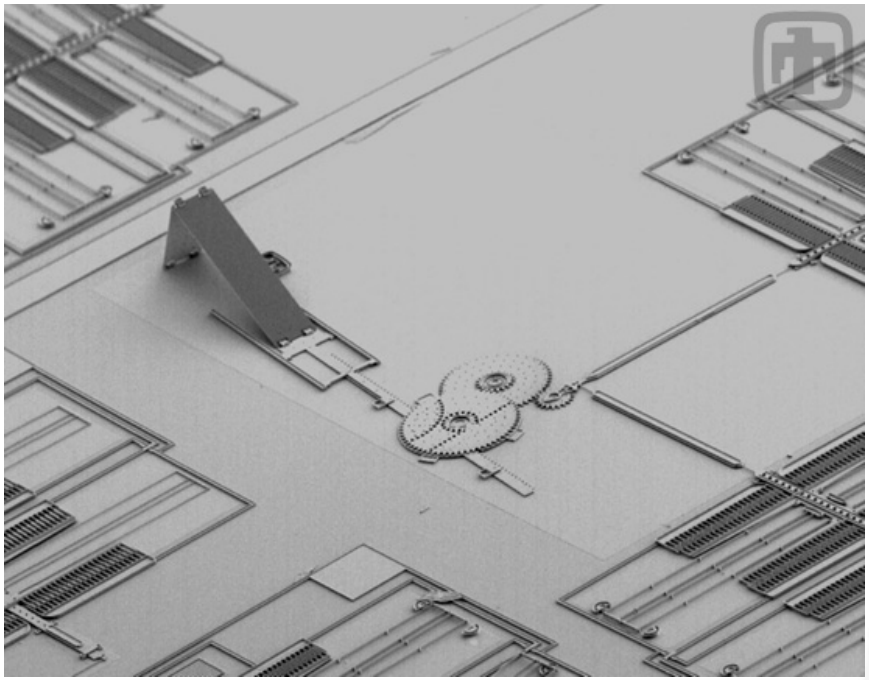
Quelle: FhG-IMS

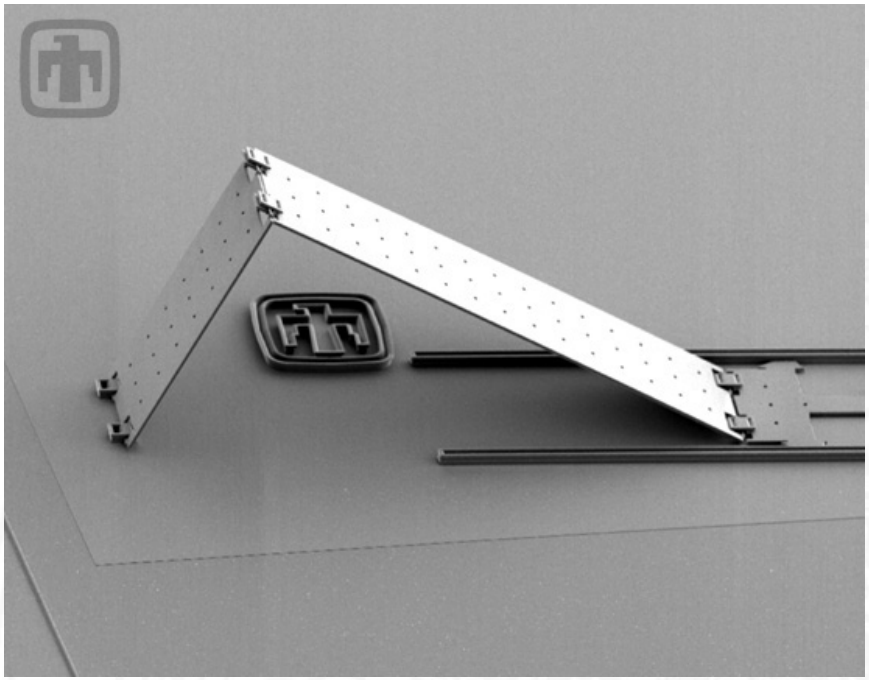


Quelle: FhG-IMS Dresden

OMM-Beispiele (cont.)

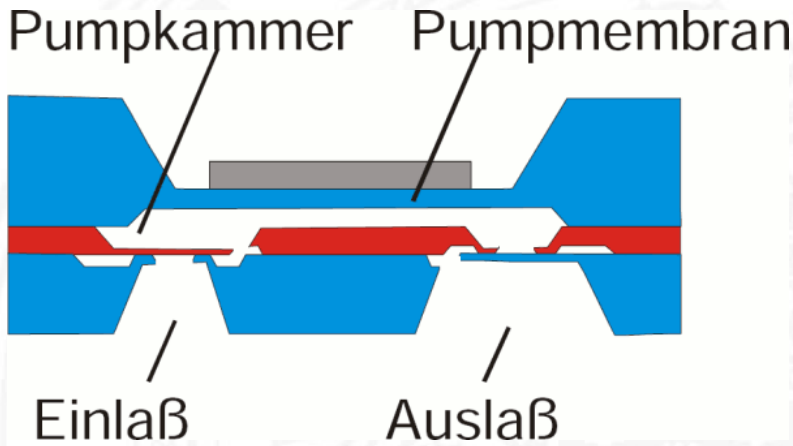
Digital **M**irror **D**evice





Herstellung und Funktion

- Mikromechanische Elemente aus Volumen (Bulk) herausätzen
- Strukturen bestehen aus Einkristall
- maximale Strukturhöhe entspricht Waferdicke (525 μm)
- Direct Wafer Bonding (zwei Wafer miteinander verbinden)



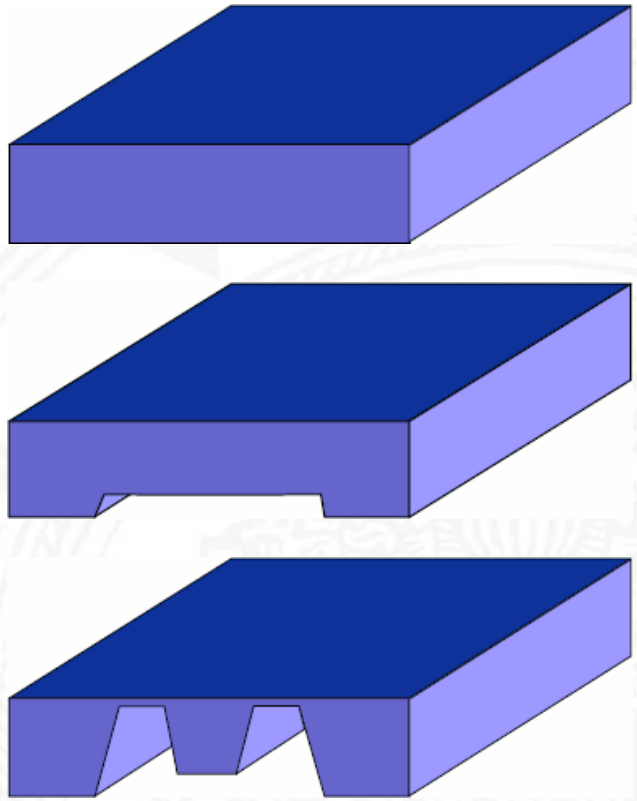
Mikropumpe
elektrostatischer oder piezoelektrischer Antrieb

Herstellungsschritte am Beispiel: Beschleunigungssensor

Bulk-Beispiele

Beschleunigungssensor

1. Wafer
2. Vorätzen der Rückseite
3. Strukturierung der Rückseite

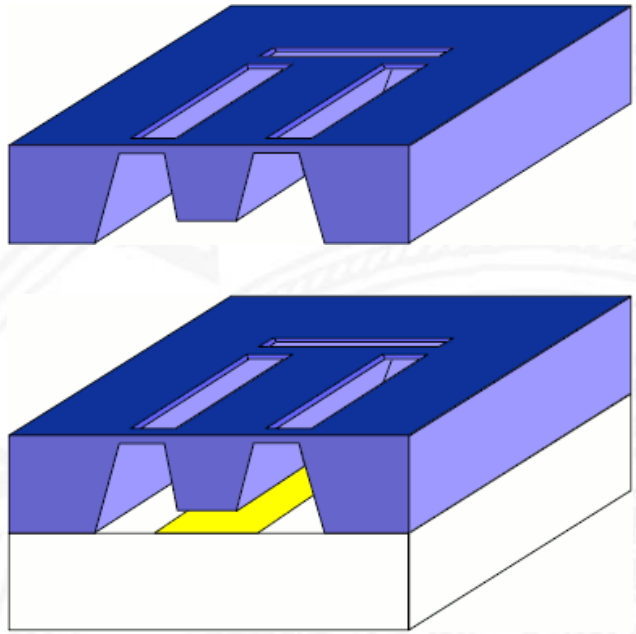


Bulk-Beispiele (cont.)

Beschleunigungssensor

4. Strukturierung der Vorderseite

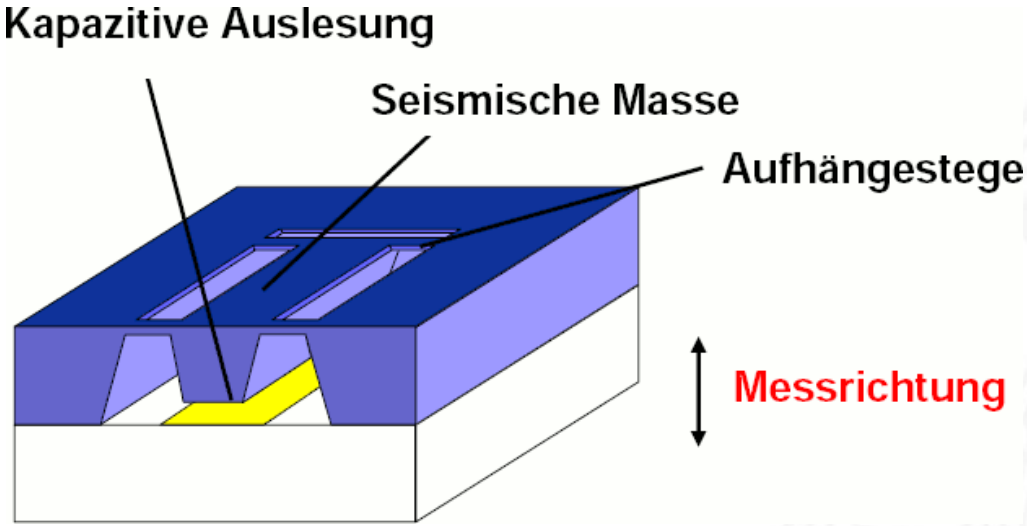
5. Verbinden mit 2. Wafer
   Bonden auf Glas, etc.

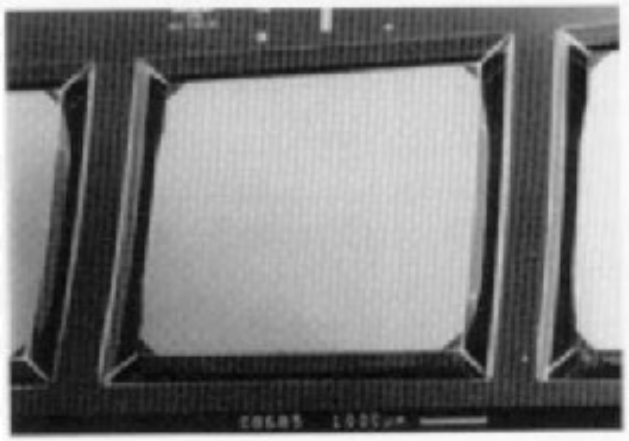


Bulk-Beispiele (cont.)

Beschleunigungssensor

- kapazitiver Beschleunigungssensor

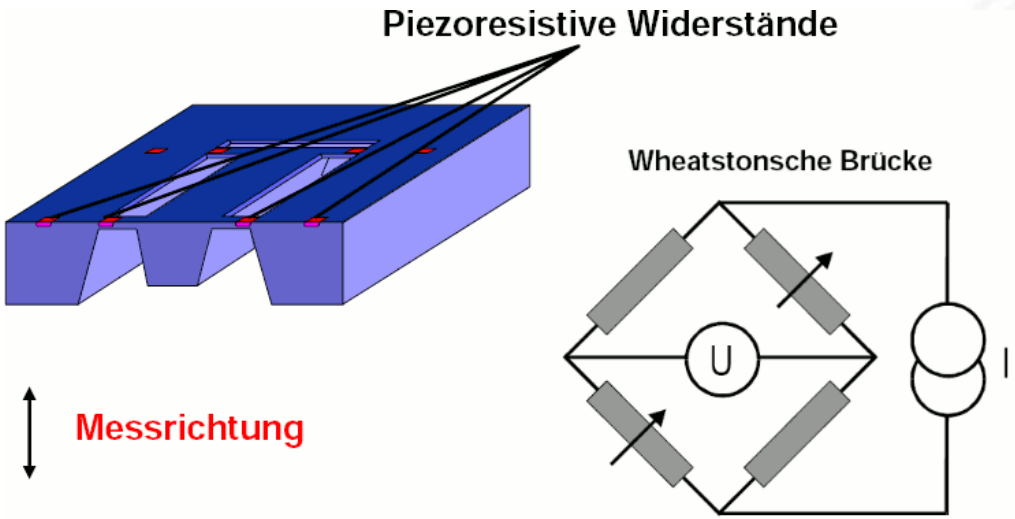


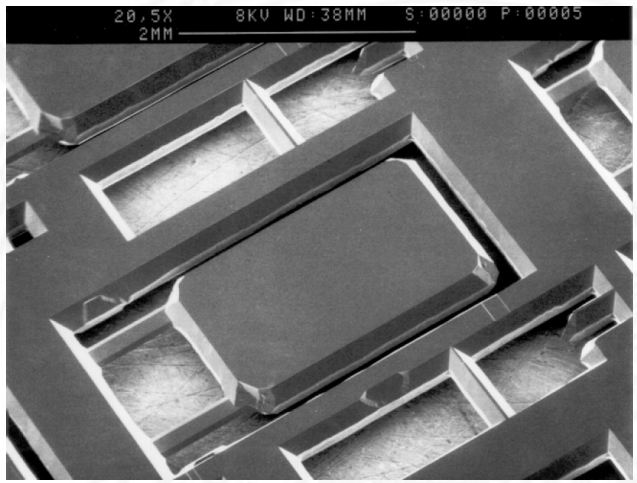


Bulk-Beispiele (cont.)

Beschleunigungssensor

- piezoresistiver Beschleunigungssensor
  Silizium ist nicht piezoelektrisch, aber piezoresistiv:
  mechanische Spannung $\rightarrow$ elektrische Widerstandsänderung

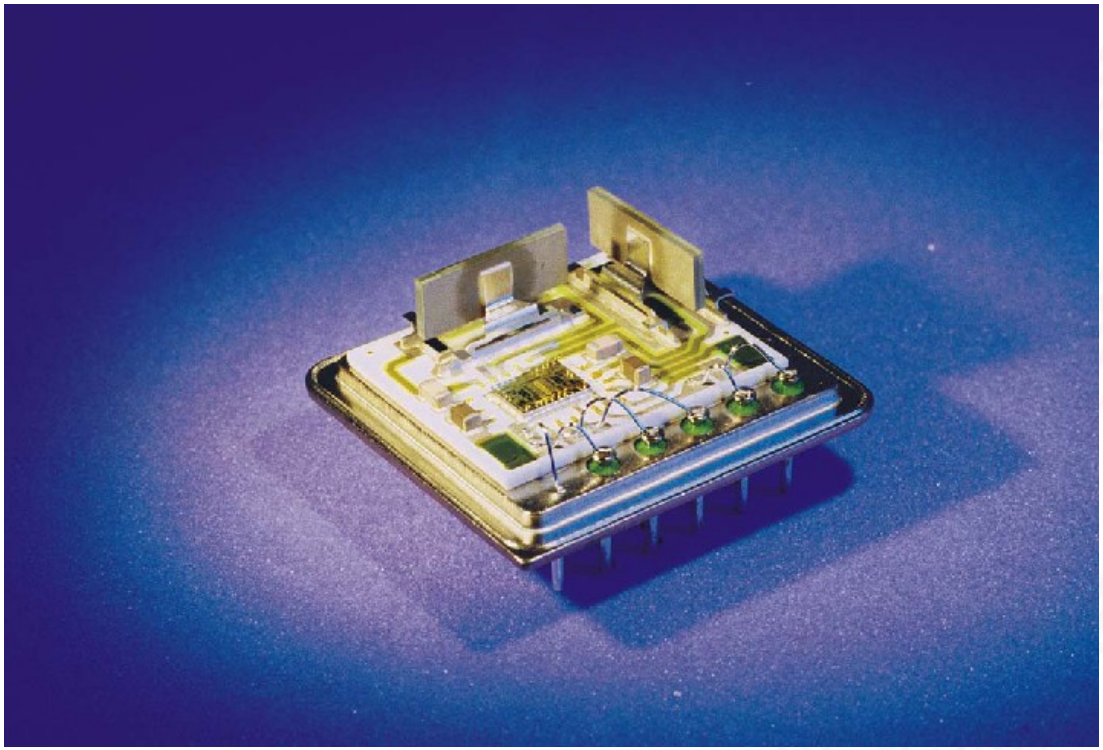




Bulk-Beispiele (cont.)

Beschleunigungssensor

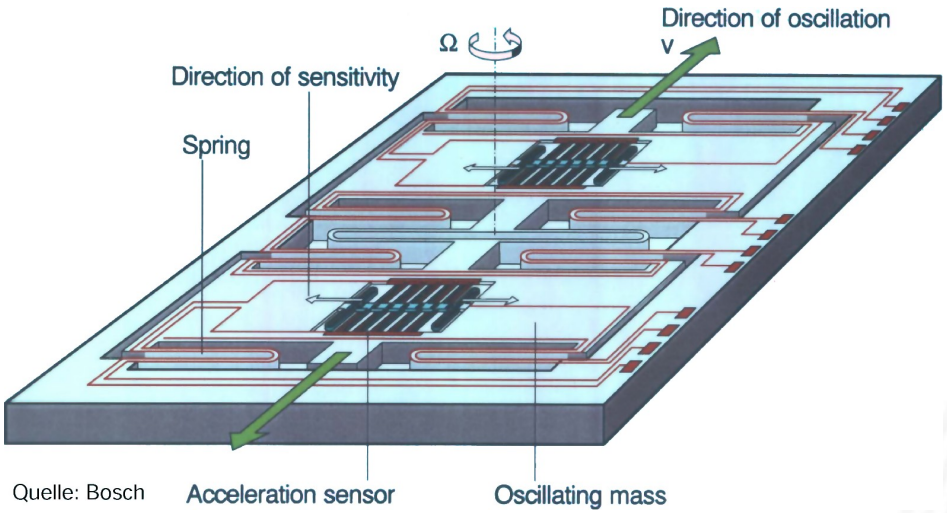
- 3D Beschleunigungssensor



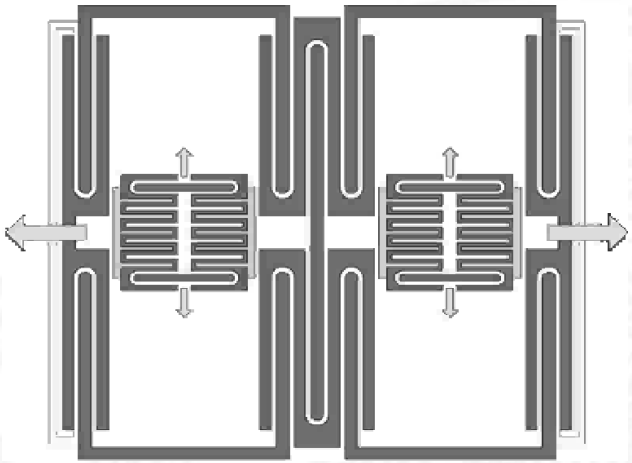
Quelle: Bosch

Bulk-Beispiele

Drehratensensor

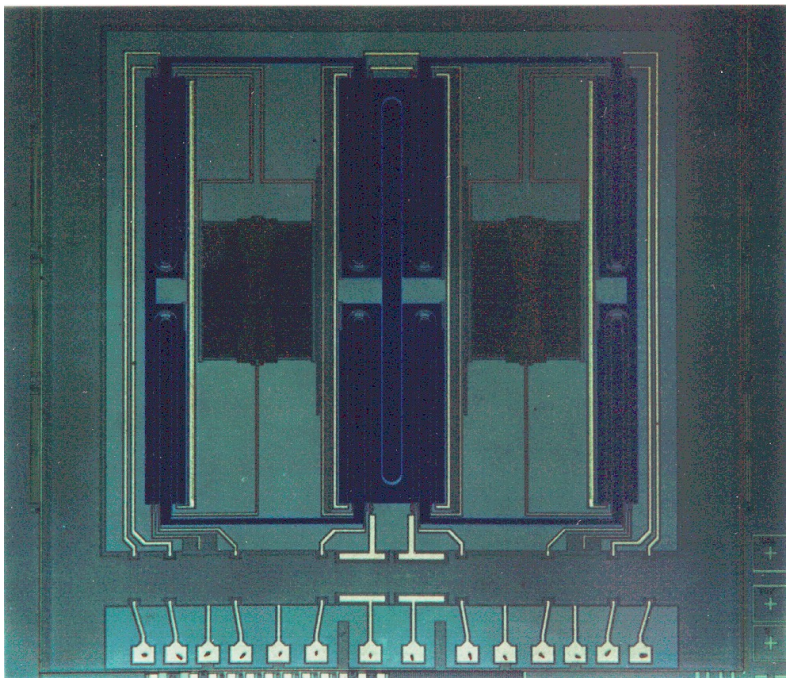


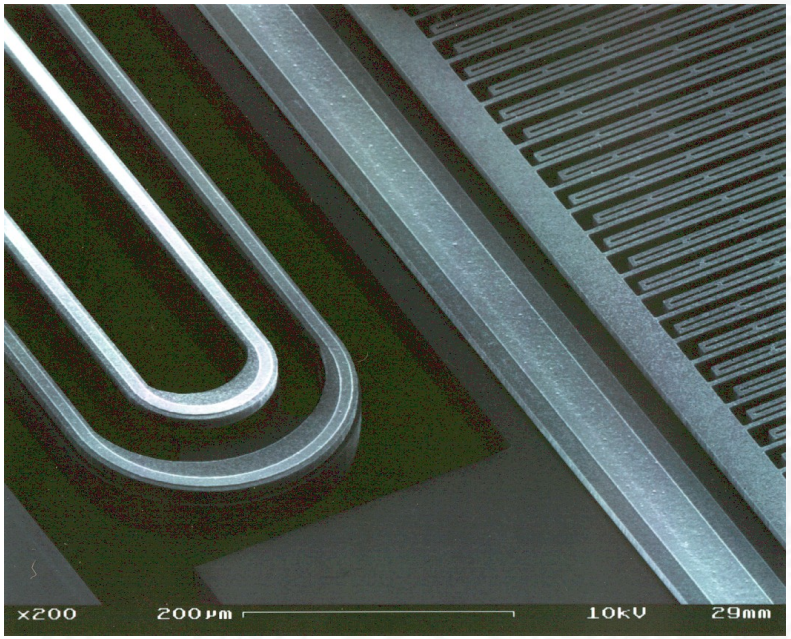
Quelle: Bosch



Bulk-Beispiele (cont.)

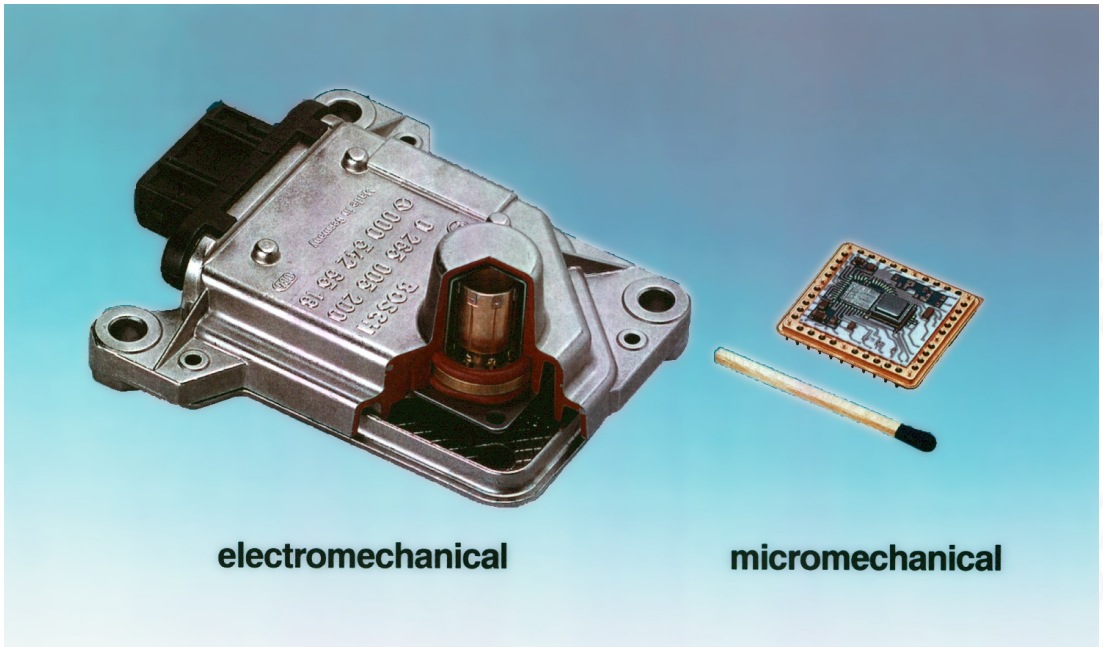
Drehratensensor





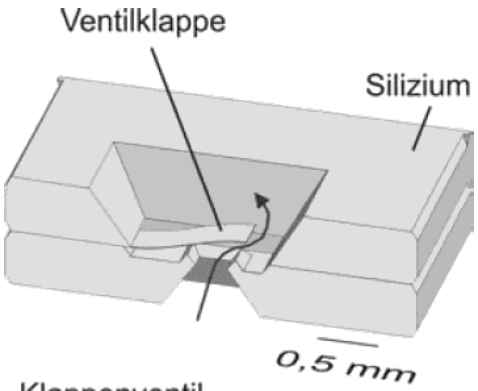
Bulk-Beispiele (cont.)

Drehratensensor

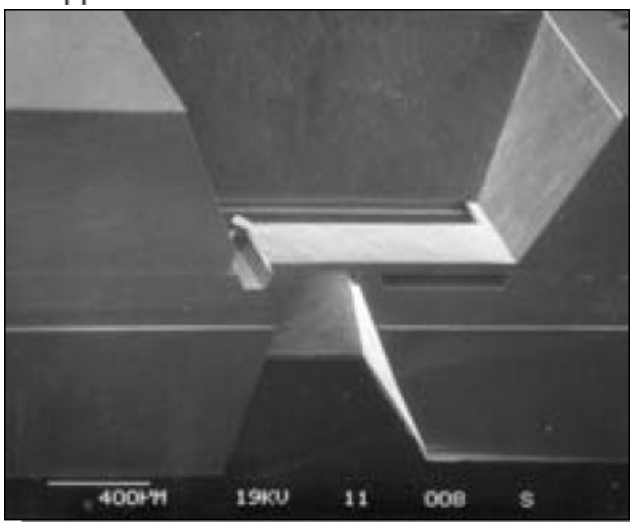


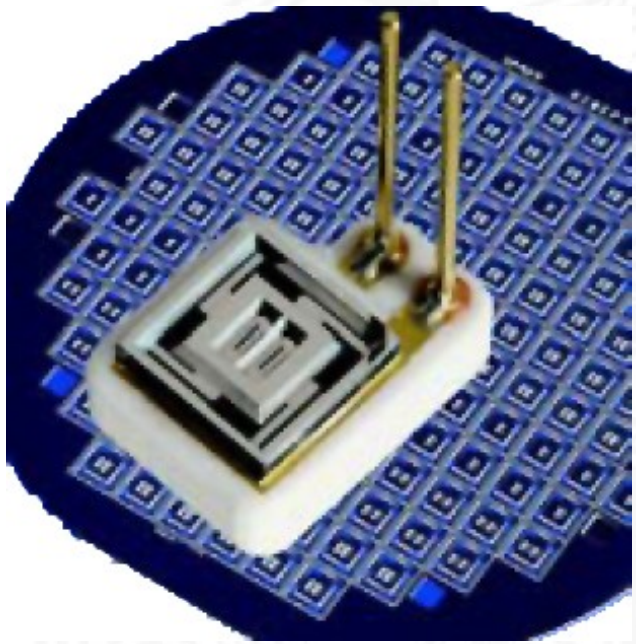
Bulk-Beispiele

Mikroventil



Klappenventil



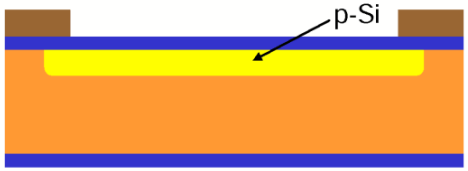


Bulk-Beispiele

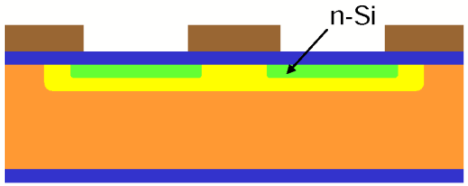
Drucksensor

(ohne BOSS-Struktur und ohne elektronische Schaltung)

thermische Oxidation (100) n⁻-Wafer
(100 nm) (beidseitig poliert)



Fotolackmaske (1µm), Implantation B,
Nachdiffusion (Ätzstopp, Membrandicke)



Fotolackmaske (1µm), Implantation P, Oxidation (1µm)
Temperung (Piezowiderstände)

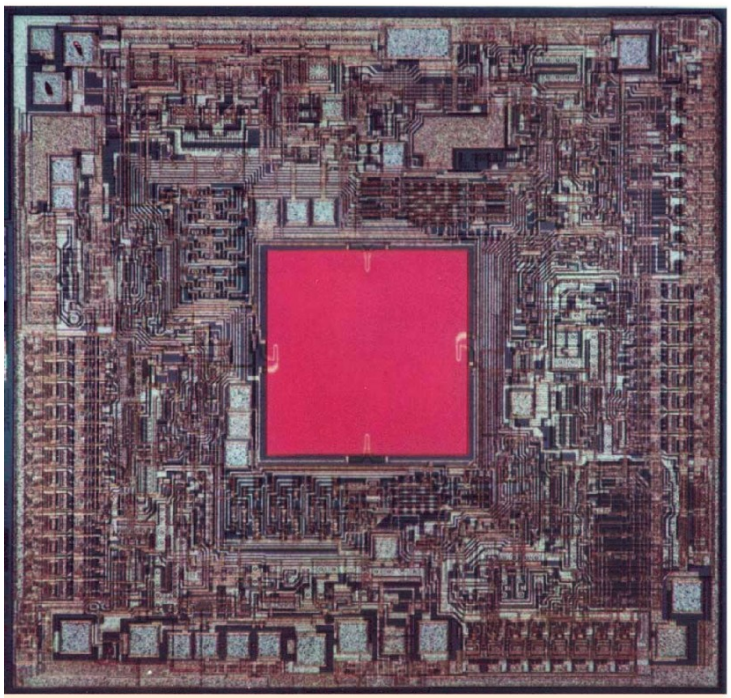
Oxidstrukturieren, Al-Si-Cu abscheiden (1µm),
strukturieren, PSG abscheiden (700nm),
strukturieren

Oxid Waferrückseite strukturieren,
Si anisotrop ätzen

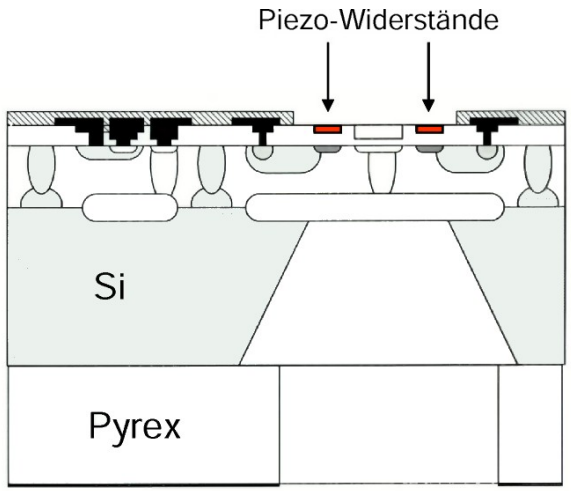
piezoresistiv

Bulk-Beispiele (cont.)

Drucksensor



Quelle: Bosch

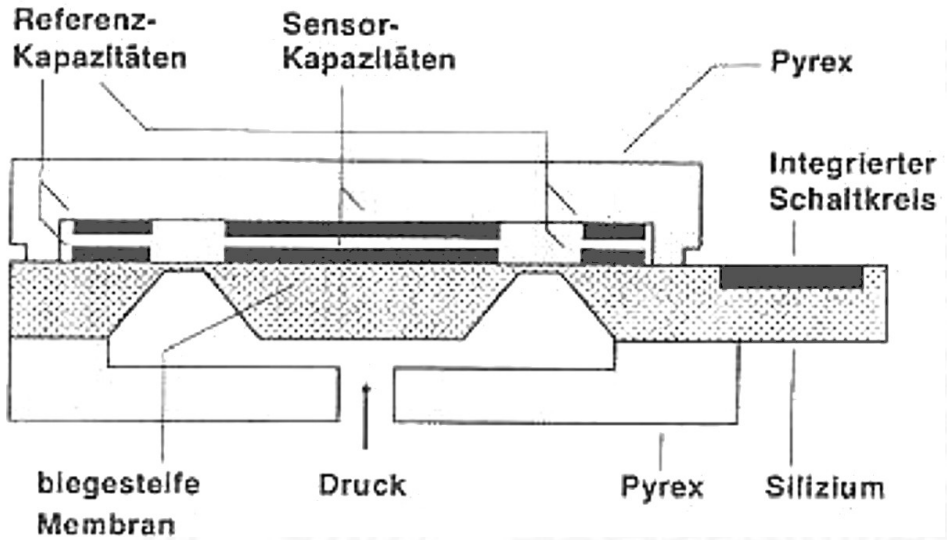


Das Drucksensor Chip hat eine Si-Membran mit Piezo-Widerständen und ist mit der Ausleseelektronik monolith integriert (Bild links). Zur Unterdrückung einer Verbiegung des Chip bei Einleitung hoher Drücke (16bar) ist das Chip durch Verbinden mit einer Pyrex Grundplatte verstärkt (Bild rechts).

Bulk-Beispiele (cont.)

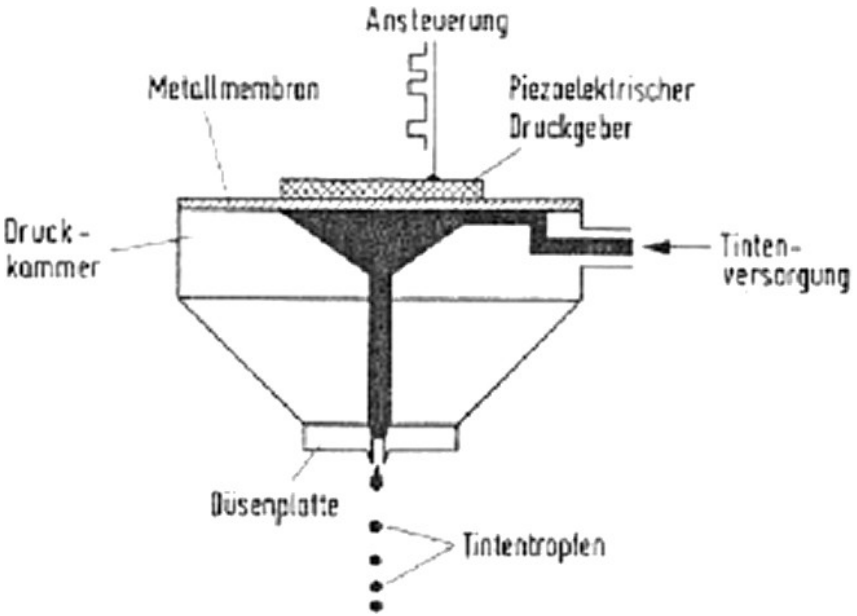
Drucksensor

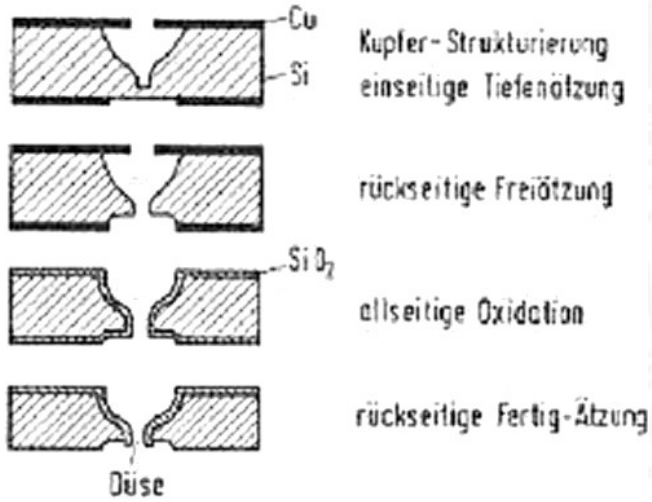
Kapazitive Auswertung



Bulk-Beispiele

Tintenstrahldruckkopf





Technologievergleich

- Bulk-Mikromechanik
  - Strukturhöhe: 20-500 μm
  - Strukturen aus dem Substrat herausätzen
  - Nass- und Trockenätzprozesse
- Oberflächenmikromechanik
  - Strukturhöhe: 2-10 μm
  - Strukturen werden als Schichten auf dem Substrat aufgebaut
  - Trockenätzprozesse
    Beliebige Formen in der Waferebene, senkrechte Wände
- Mikroelektronik
  - Strukturhöhe: 0,1-2 μm

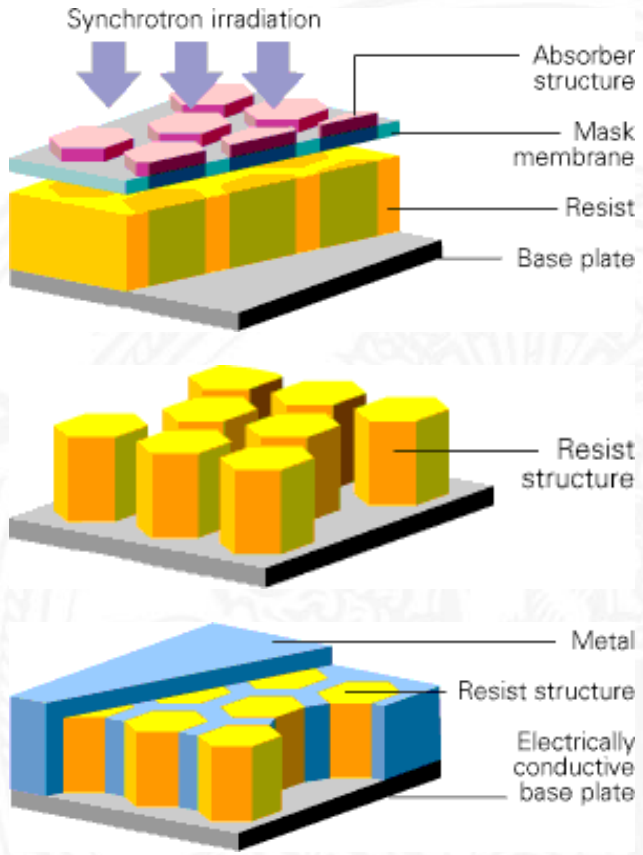
Motivation

- Alternative Materialien: Kunststoff
  - geringe Kosten
  - günstige Fertigungsmöglichkeiten
  - breite Materialpalette
- Alternative Technologien zur Mikrostrukturierung: Abformverfahren
- Kostenreduktion $\Rightarrow$ Massenfertigung
- Anwendungsgebiete: Mikrooptik, Mikrofluidik
- Kunststofffertigung
  - Spritzgiessverfahren
  - Heiß-Prägeverfahren

Herstellung und Funktion

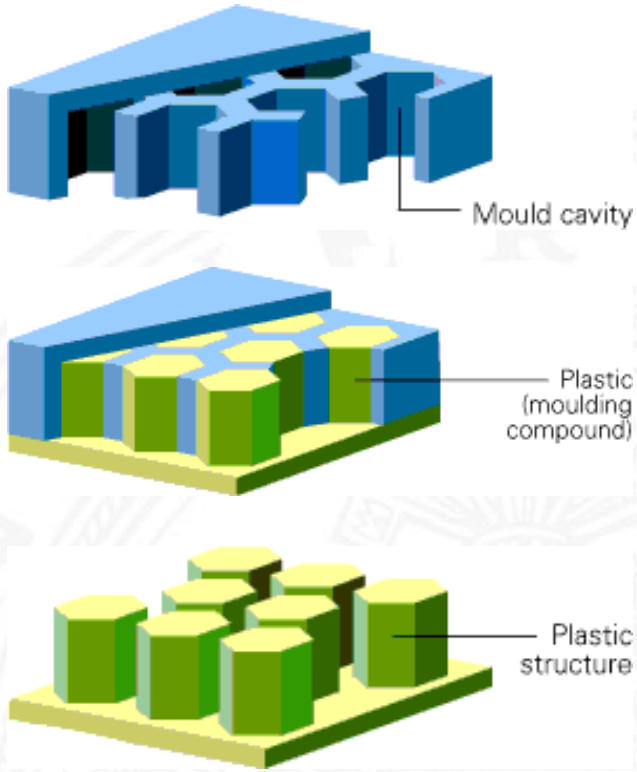
LIGA – **Li**thografie, **G**alvanik und **A**bformung

- Bestrahlung
- Entwicklung
- Galvanik



Herstellung und Funktion (cont.)

- Resist entfernen $\Rightarrow$ Formeinsatz
- Abformung: Formfüllung
- Abformung: Entformung



Herstellung und Funktion (cont.)

Herstellung der Formeinsätze

1. Röntgenlithografie $0,2\,nm < \lambda < 2\,nm$

+ große Eindringtiefe, keine Beugung
+ hohe Lackdicken bis 3mm möglich
  Resist: PMMA (Polymethylmethacrylat, „Plexiglas“)
– Strahlungsquelle: Synchrotronstrahlung
– keine optischen Komponenten verfügbar
– Schattenprojektion (Proximity Belichtung)

Herstellung und Funktion (cont.)

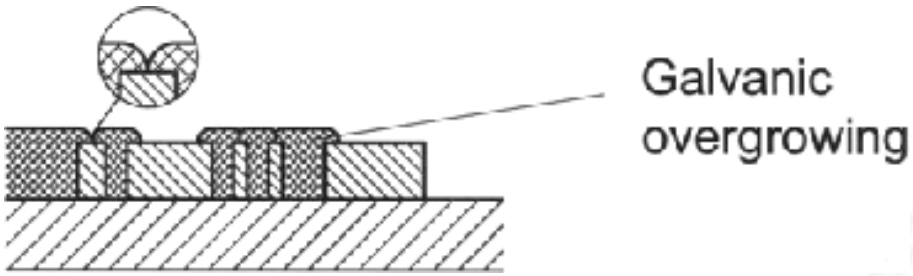
- Absorbermaske aus Gold ($> 10\,\mu m$ Dicke)
  Aufwändige Herstellung wegen Dicke
  - Zwischenmaske mit Elektronenstrahlbelichtung erzeugen
  - LIGA-Maske mit Synchrotronstrahlung belichten
  - Gold galvanisch auftragen
- Trägerfolien
  - Beryllium: giftig
  - Kapton: nicht Strahlungsbeständig
  - Titan: dünne Folie, mechanisch instabil

Alternativen: SU-8 Fotoresist

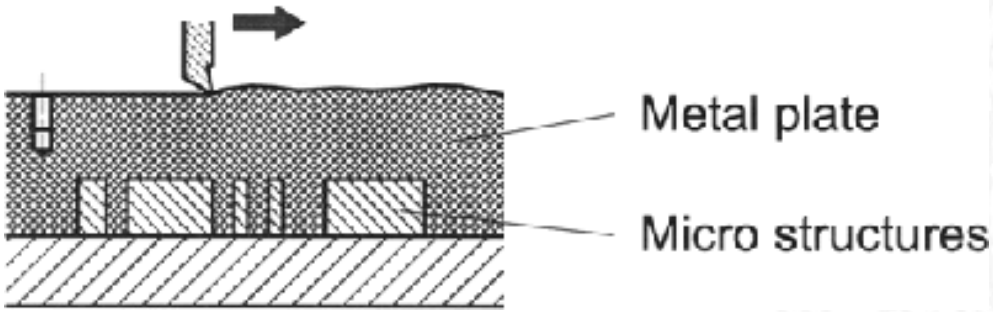
- Strukturhöhe $< 500\,\mu m$ Stapel mehrerer Lagen bis 2mm
- Aspektverhältnis 1:40 (Breite:Höhe) / PMMA $>$ 1:100
- Volumenschwund
- keine großflächigen Strukturen möglich

Herstellung und Funktion (cont.)

2. Galvanische Abscheidung (z.B. Nickel)
   Kunststoffstruktur wird zur (Negativ-) Metallstruktur

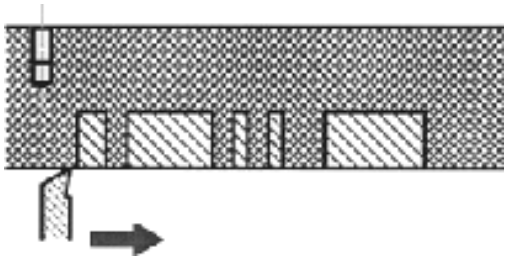


3. Stabile Metallplatte ausbilden (z.B. 5mm), Glätten

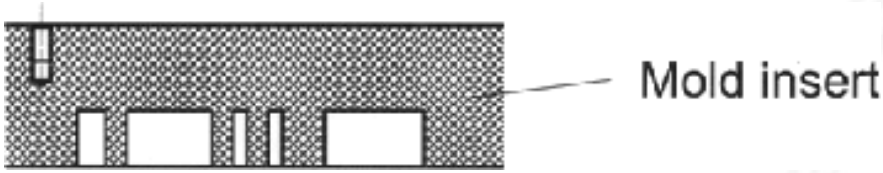


Herstellung und Funktion (cont.)

4. Grundplatte abtrennen, Glätten



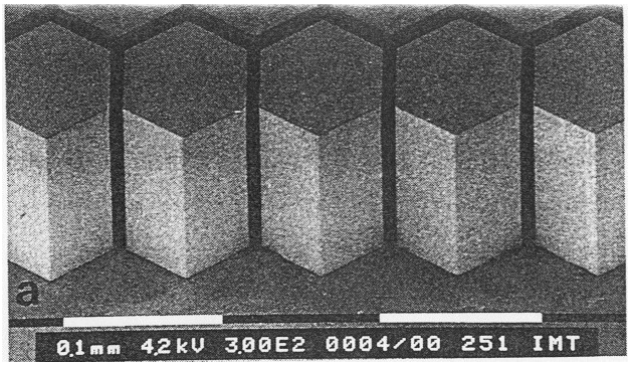
5. Resist entfernen



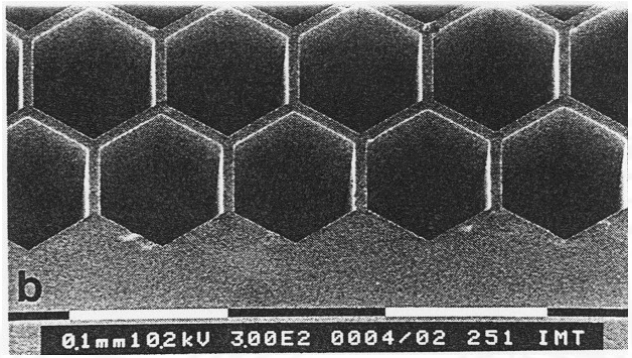
Herstellung und Funktion (cont.)

6. Abformung der Metallstruktur: (Positiv-) Kuststoffstruktur

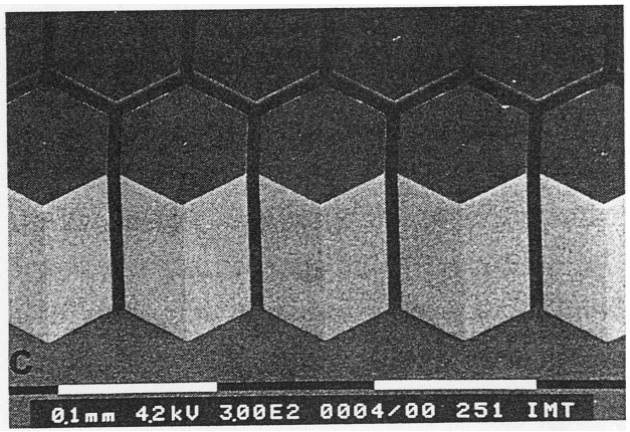
PMMA



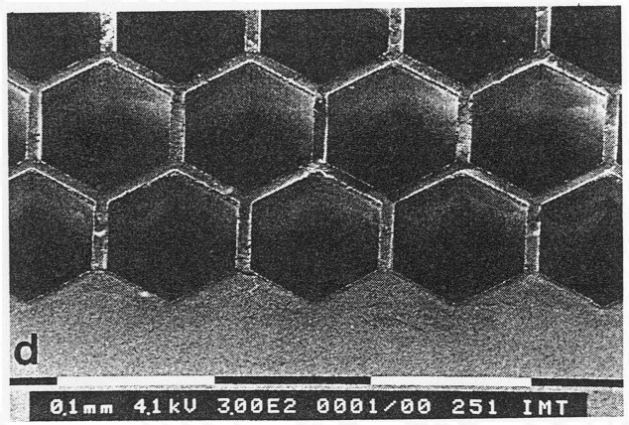
Ni-Formeinsatz



Abgeformte Kunststoffstruktur

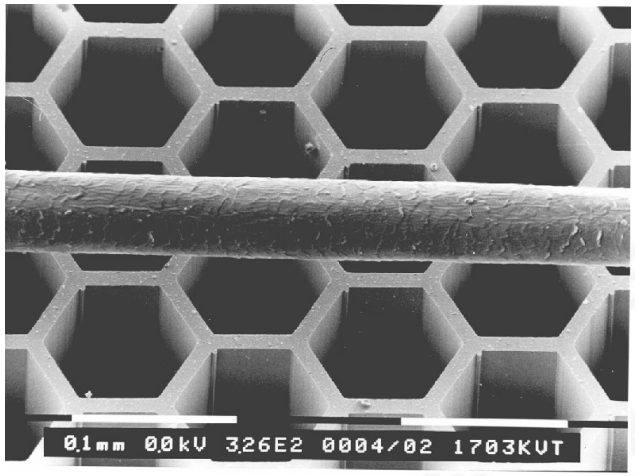


Endprodukt: NI-Wabennetz



Herstellung und Funktion (cont.)

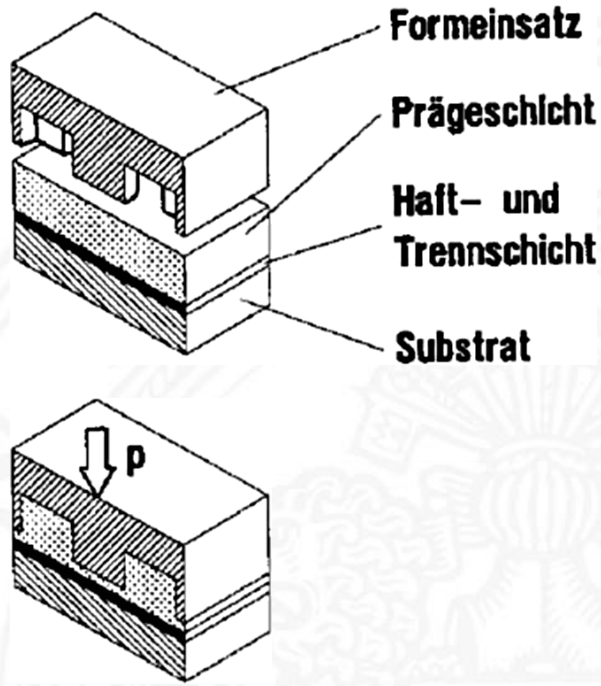
Größenvergleich:



7. Anschließend folgen die eigentlichen Fertigungsverfahren als Mikrospritzguß- oder Prägeverfahren (CD, DVD-Herstellung)

Herstellung und Funktion (cont.)

Heiß-Prägeverfahren

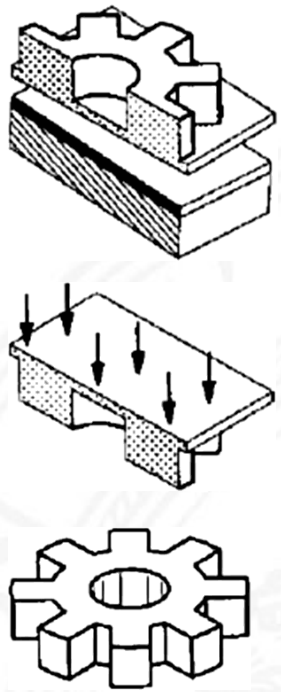


1. Vorbereitung

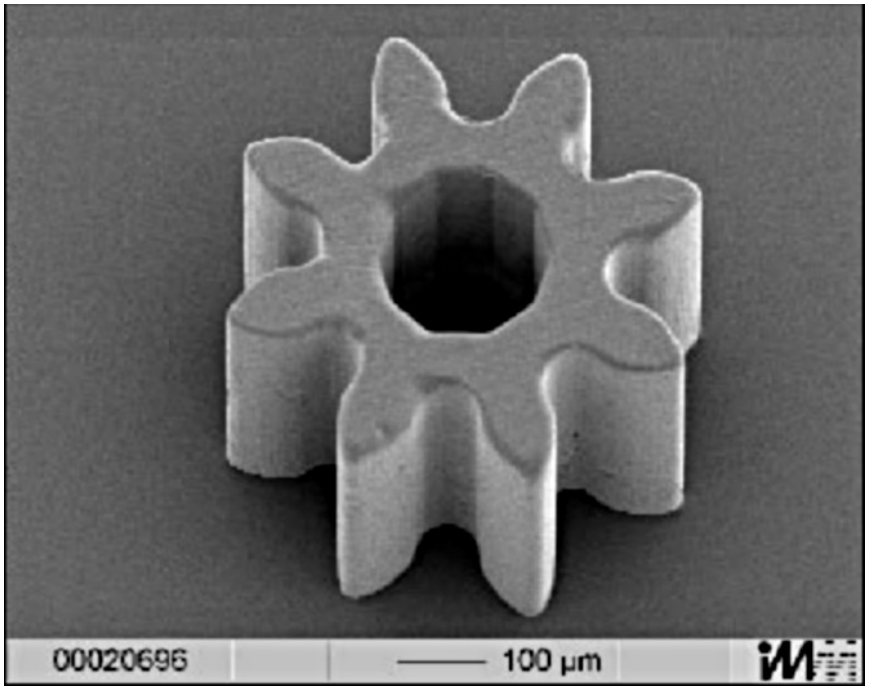
2. Warmumformen
   Drücke mehrere 100 bar

Herstellung und Funktion (cont.)

3. Abtrennen
4. Ätzen der Restschicht (10-50μm)
5. fertige Struktur

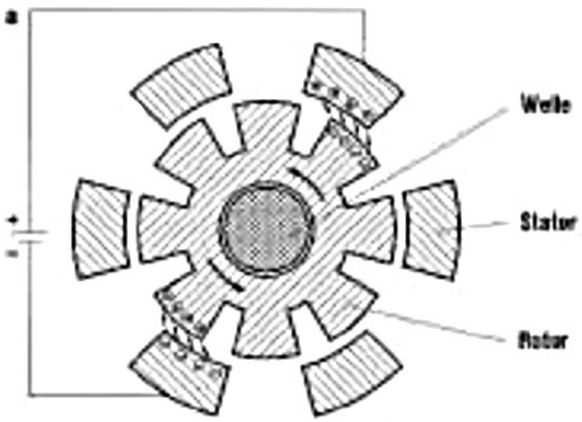


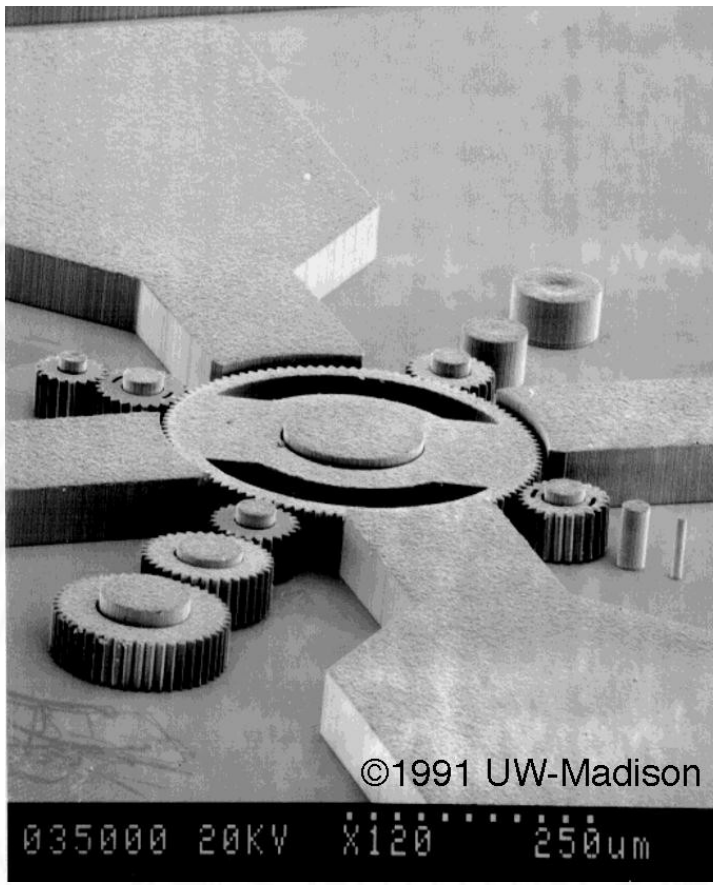
Herstellung und Funktion (cont.)

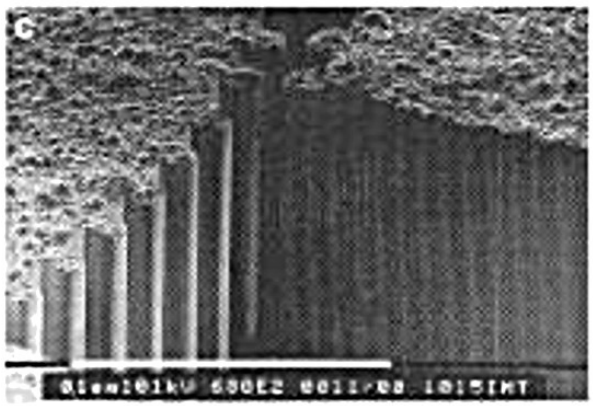


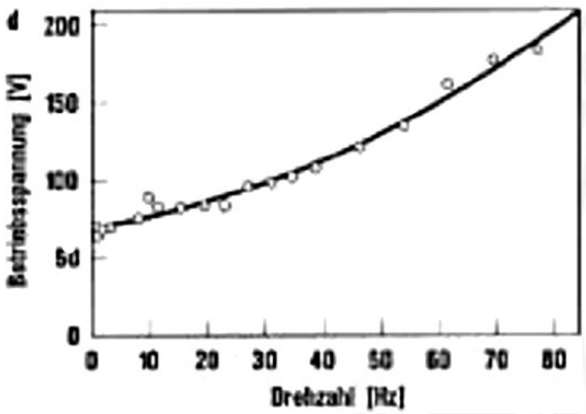
LIGA-Beispiele

Elektrostatisch angetriebener Motor



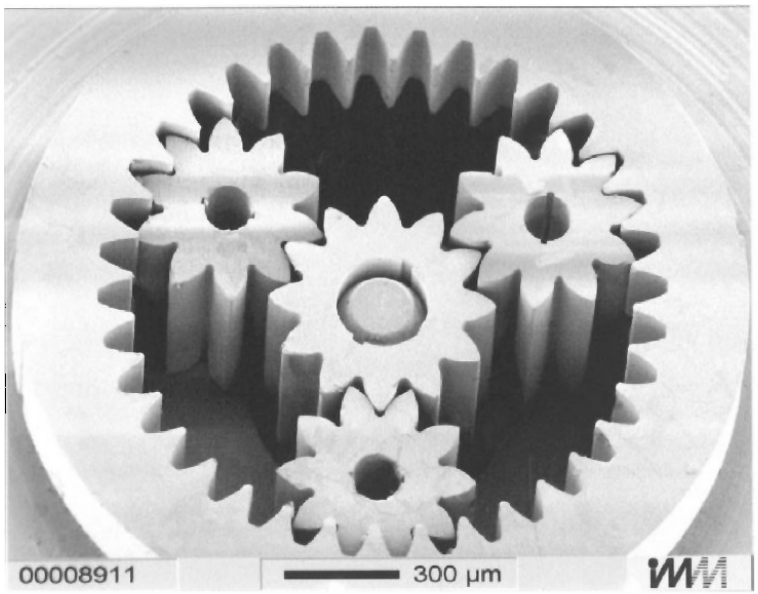






LIGA-Beispiele

Getriebe



LIGA Process

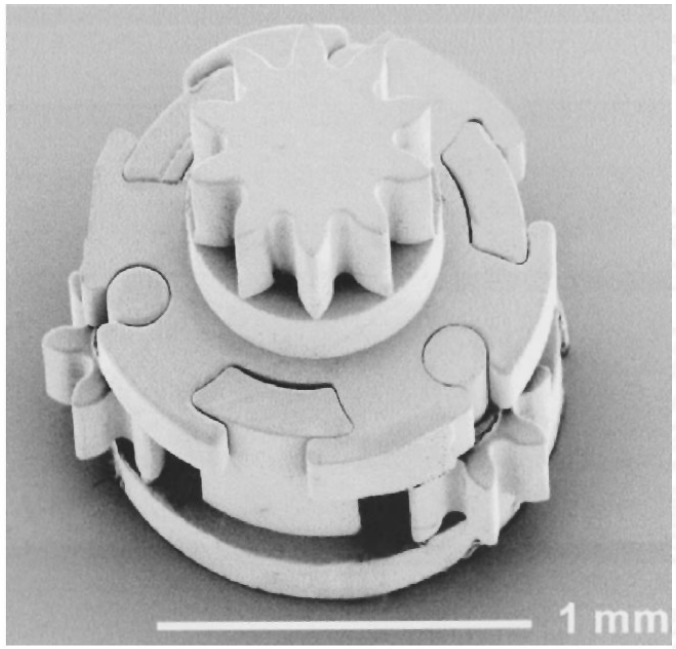
Diameter: 1.8mm

Material: PMMA

Modul: 38 μm

Transmission ratio: 3:1

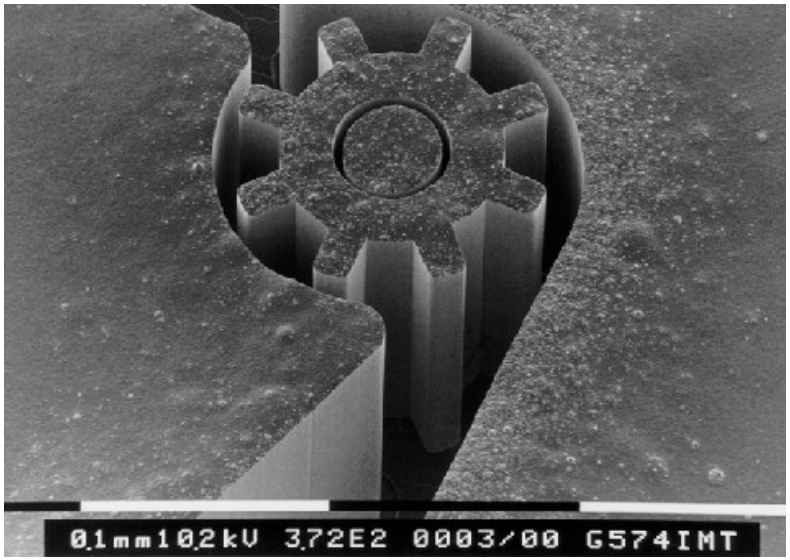
Source: IMM

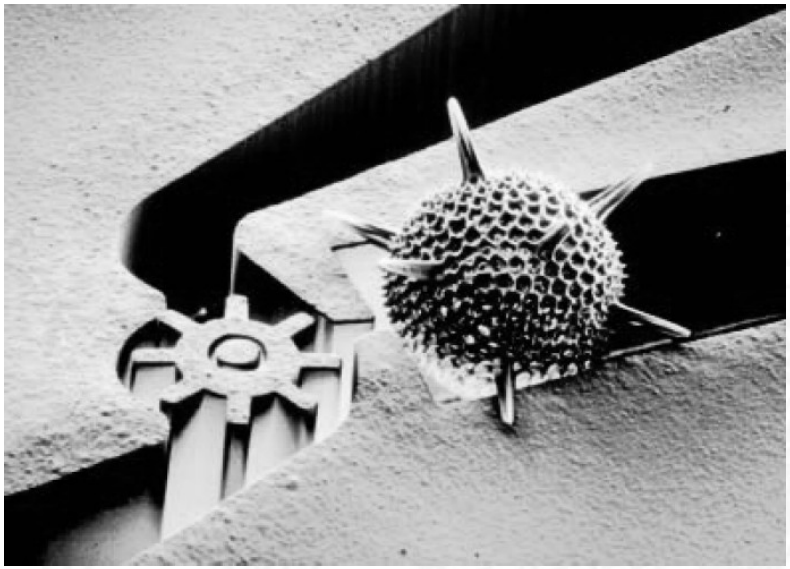


Quelle: IMM, Faulhaber

LIGA-Beispiele

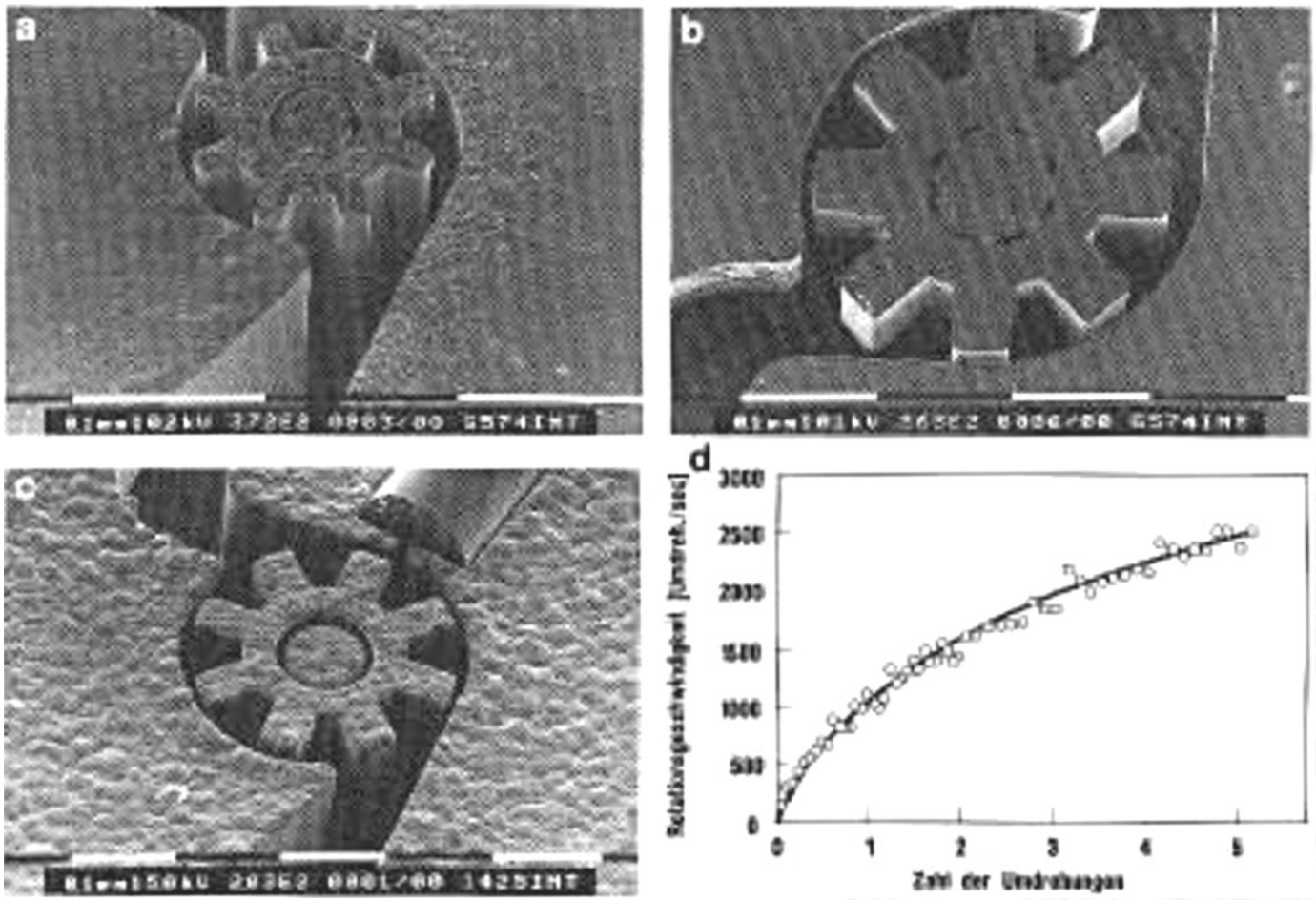
Mikroturbine





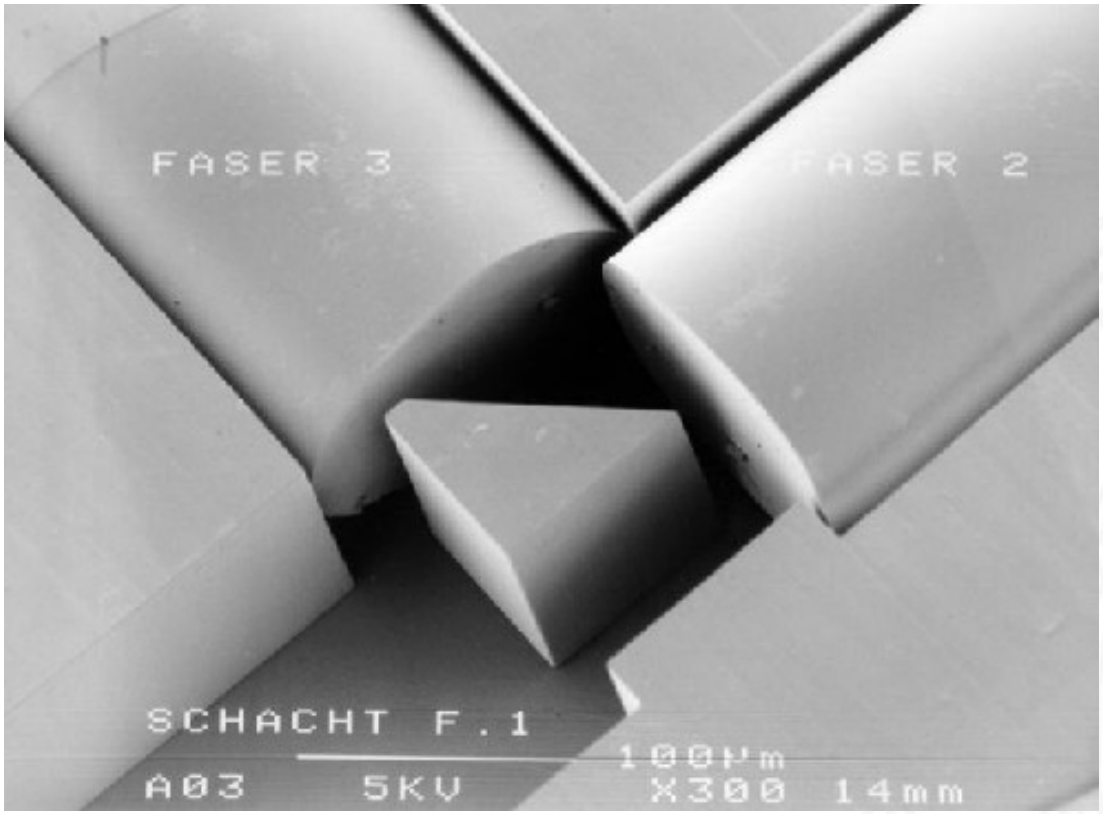
LIGA-Beispiele (cont.)

Mikroturbine



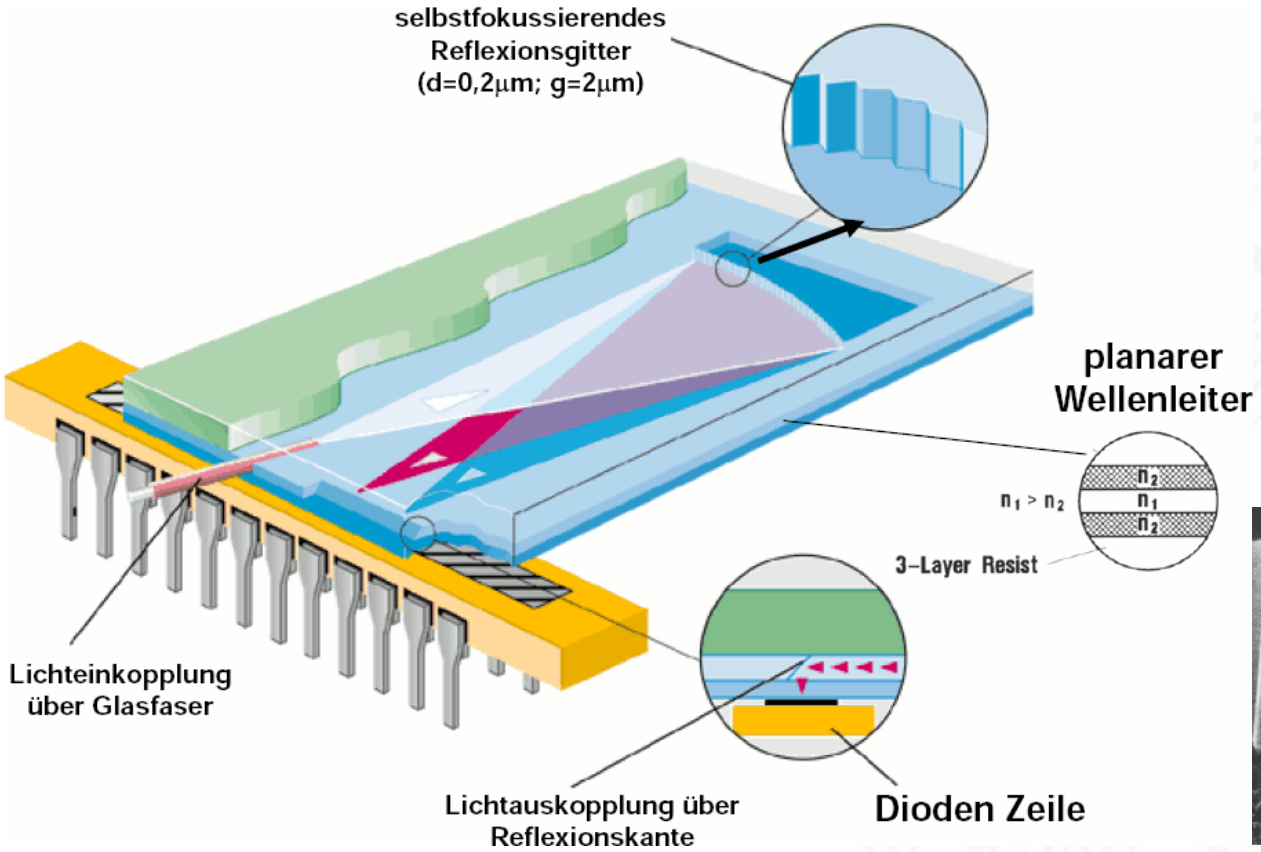
LIGA-Beispiele

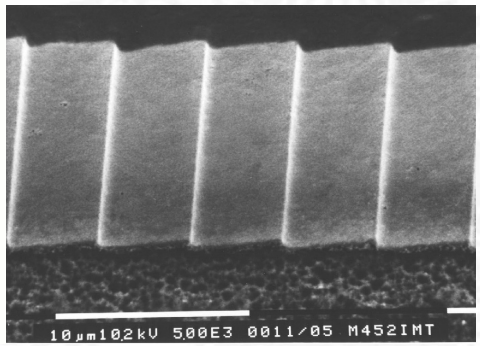
optische Bank



LIGA-Beispiele

Spektrometer



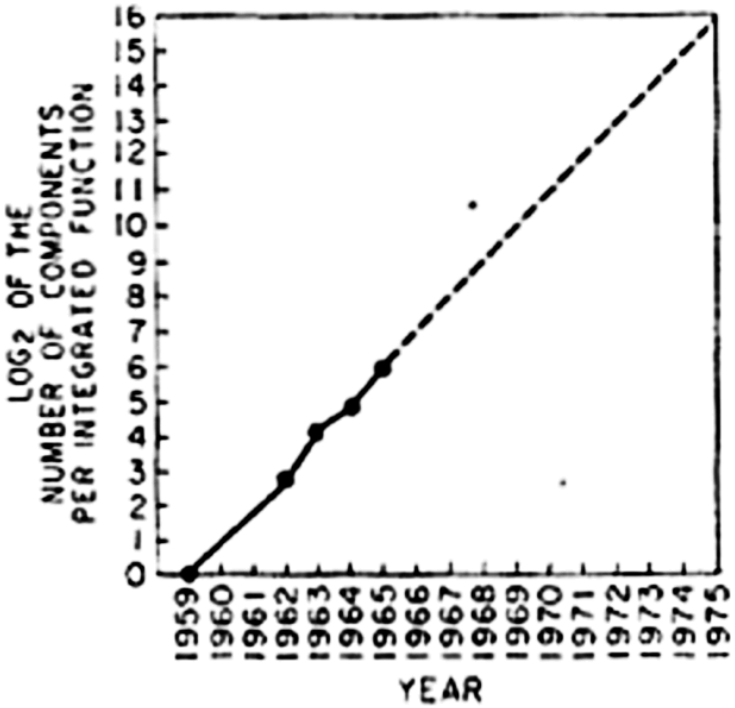


Gliederung

Motivation

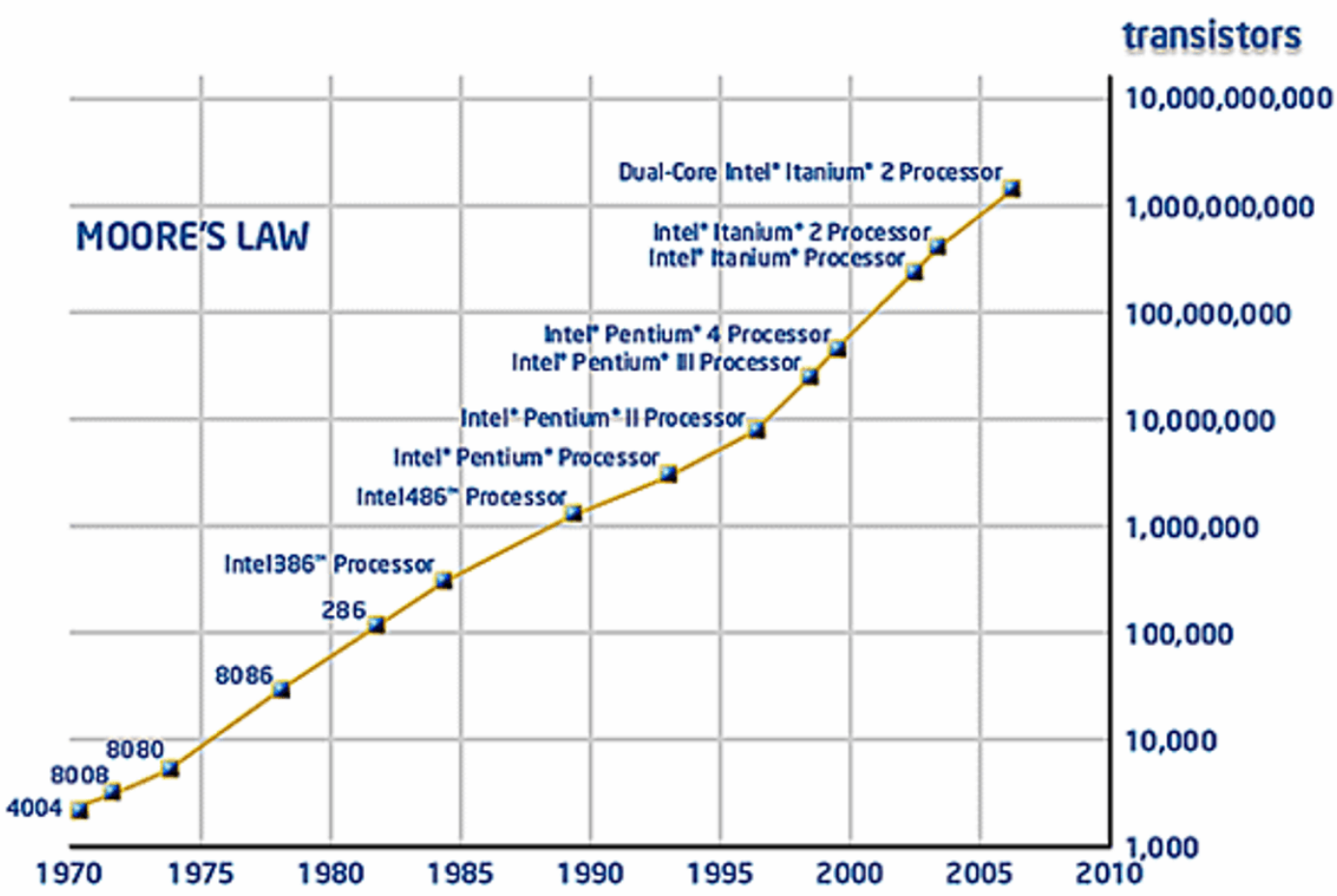
Moore's Law

Die Zahl der Transistoren pro IC verdoppelt sich alle 2 Jahre



Gordon Moore 1965:
„Cramming more components onto integrated circuits“

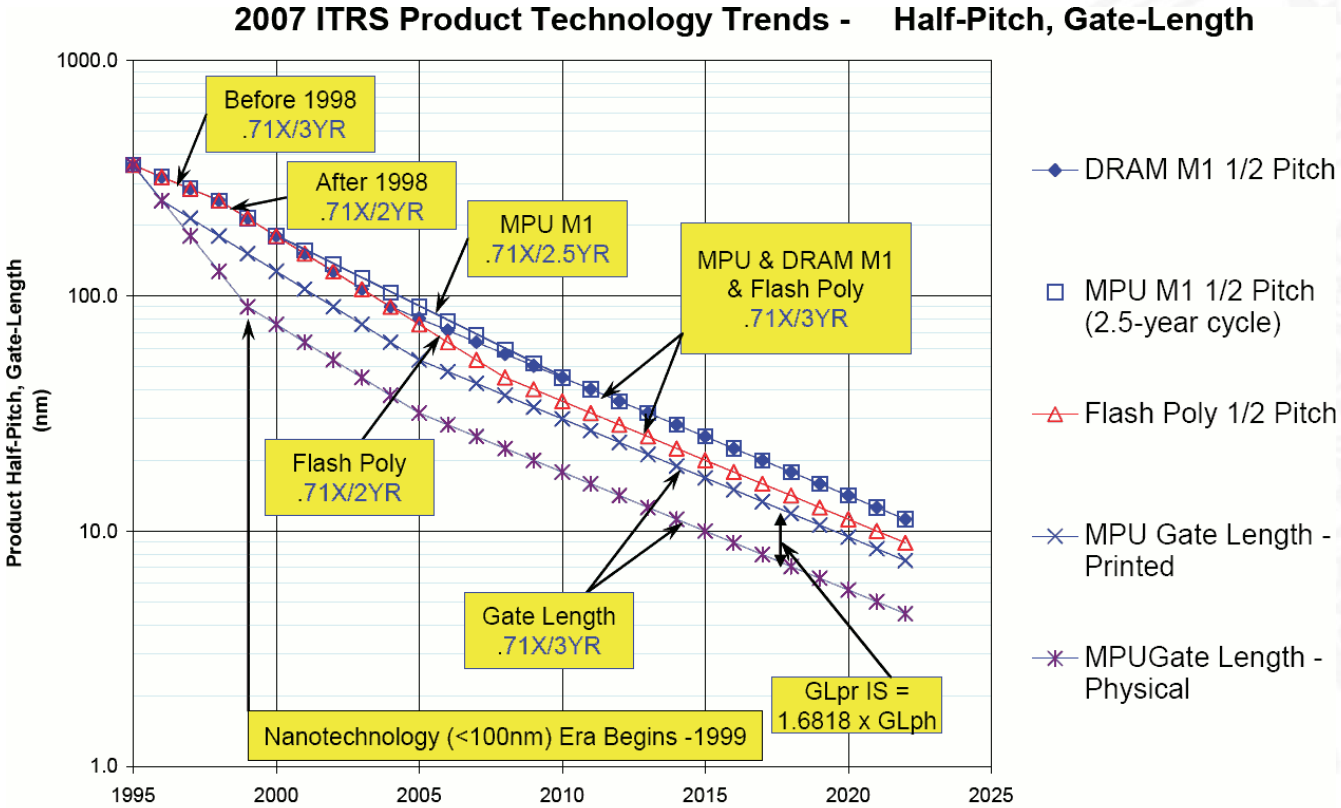
Motivation (cont.)



Intel

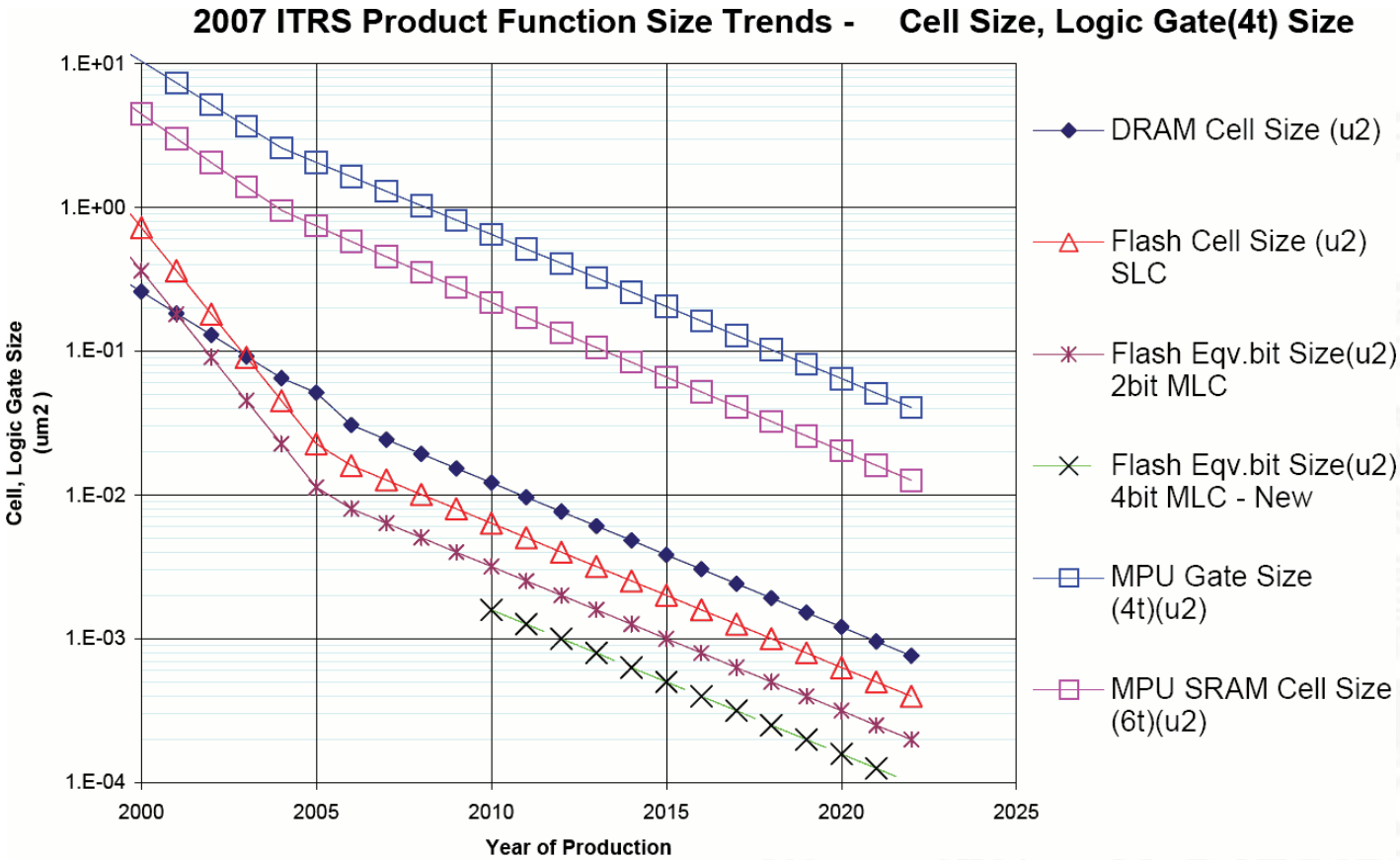
Motivation (cont.)

1. Technologie
   - Verkleinerung der Strukturbreite
   - Höhere Integrationsdichte



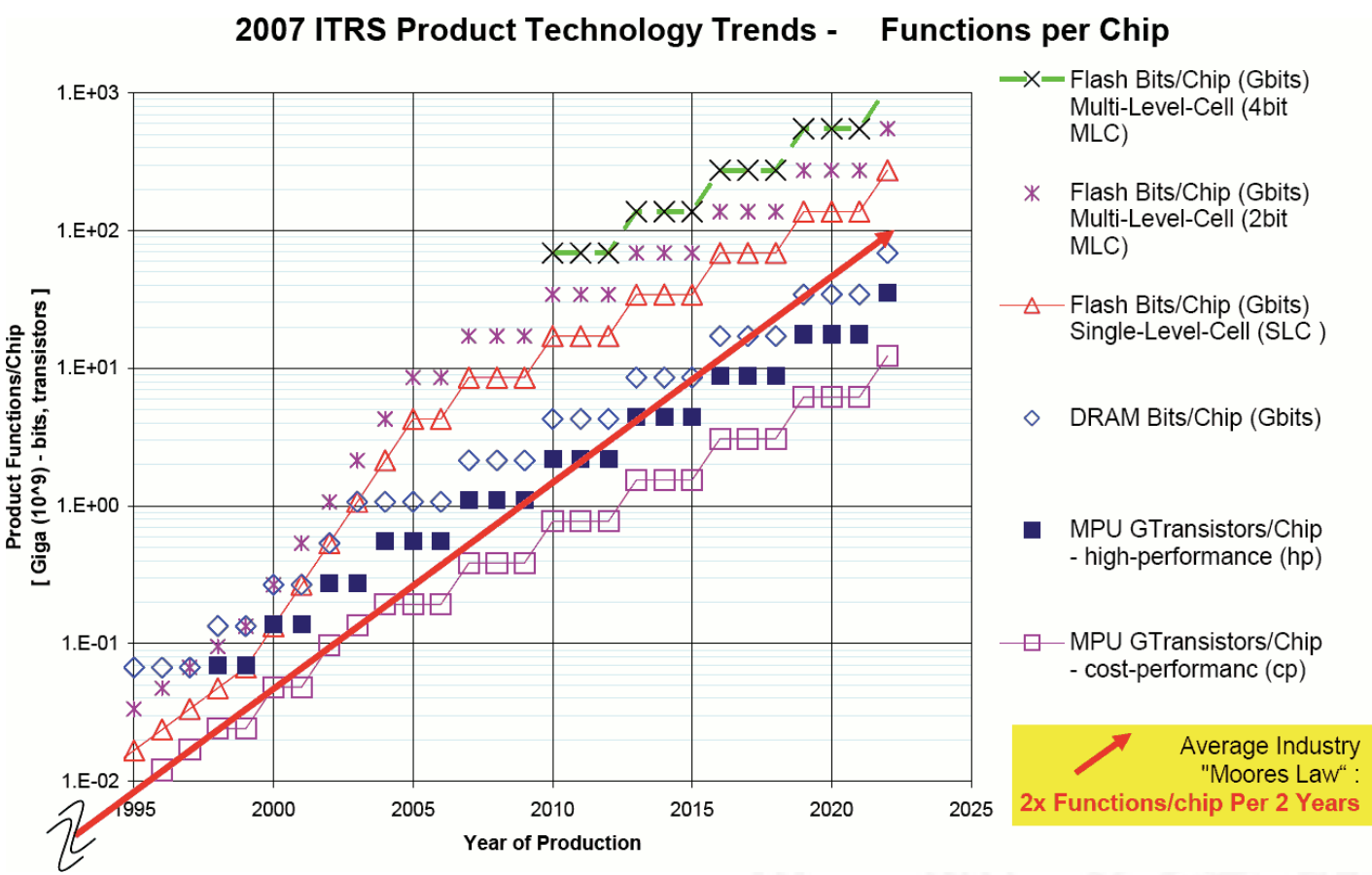
[ITRS07]

Motivation (cont.)



[ITRS07]

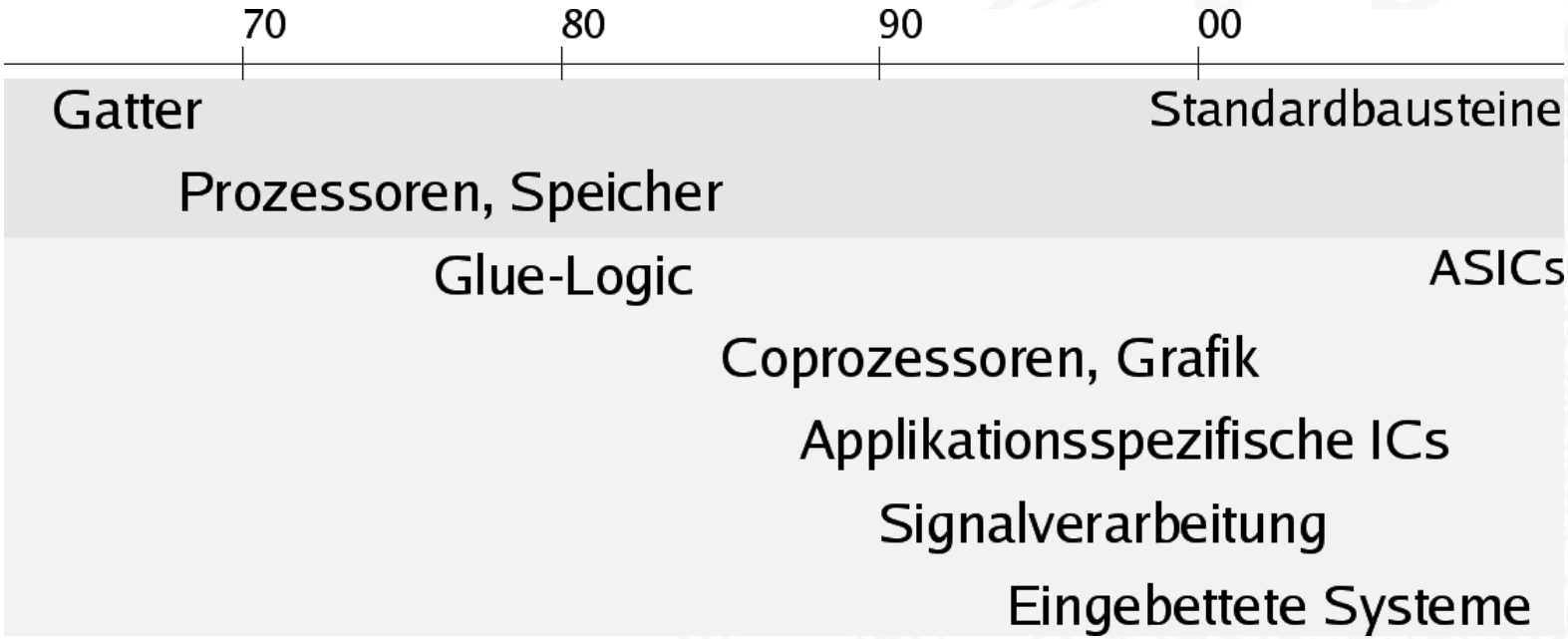
Motivation (cont.)



[ITRS07]

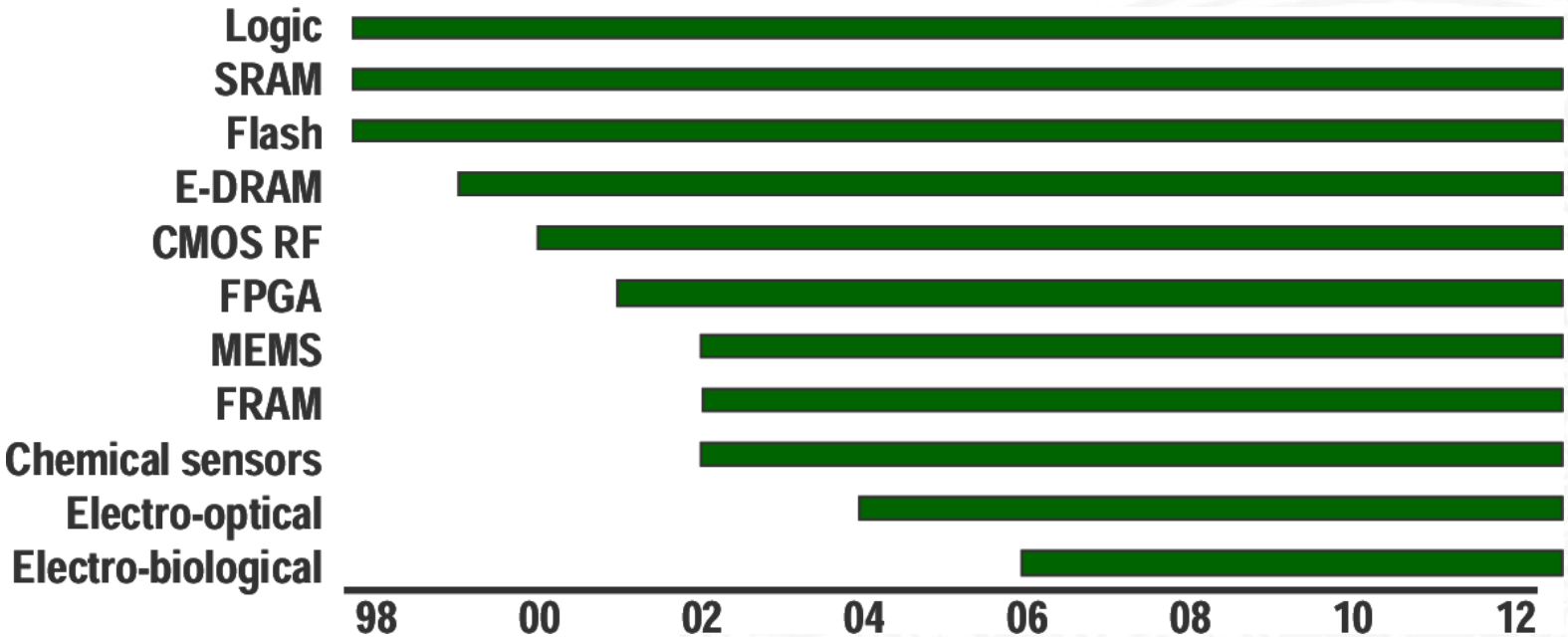
Motivation (cont.)

2. Applikationen
   - von Standardbausteinen zu ASICs und Systemen
   - Digitale Anwendungen



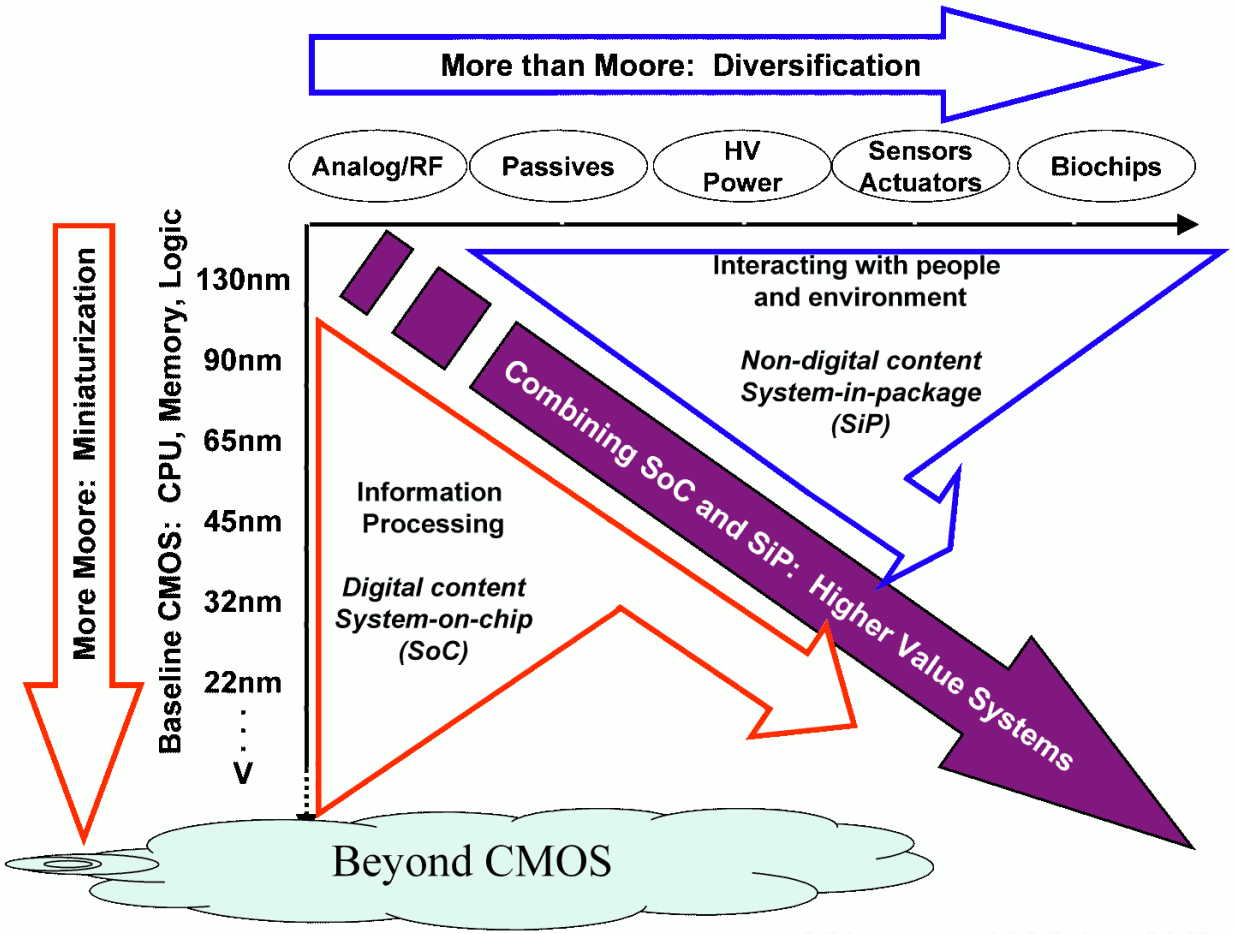
Motivation (cont.)

- + Analoge Komponenten
- + Mikrosysteme



Motivation (cont.)

Übergang zu Systemen



[ITRS07]

Motivation (cont.)

neue Anwendungsfelder

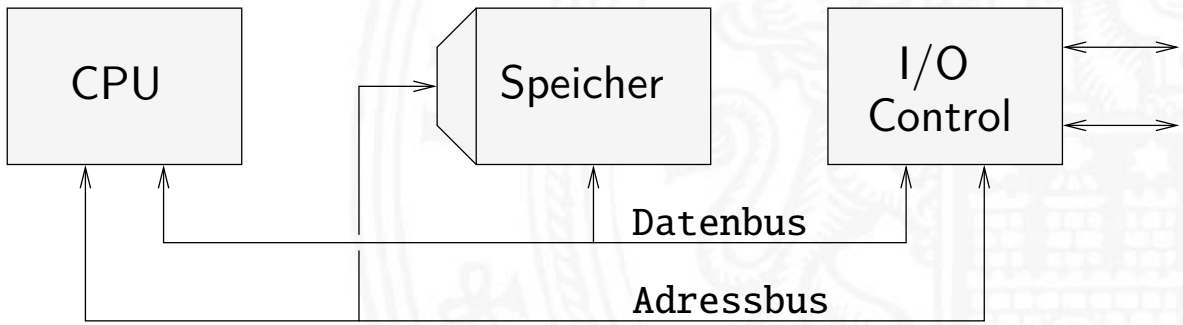
- „Computing“
- „Consumer Products“
- „Automotive“
- „Telecommunication“
- „mobile Applications“

3. Methoden und Werkzeuge im Chipentwurf
   - enges Zusammenwirken mit der technischen Entwicklung und den Anforderungen durch die Applikationen

Abstraktion im VLSI-Entwurf

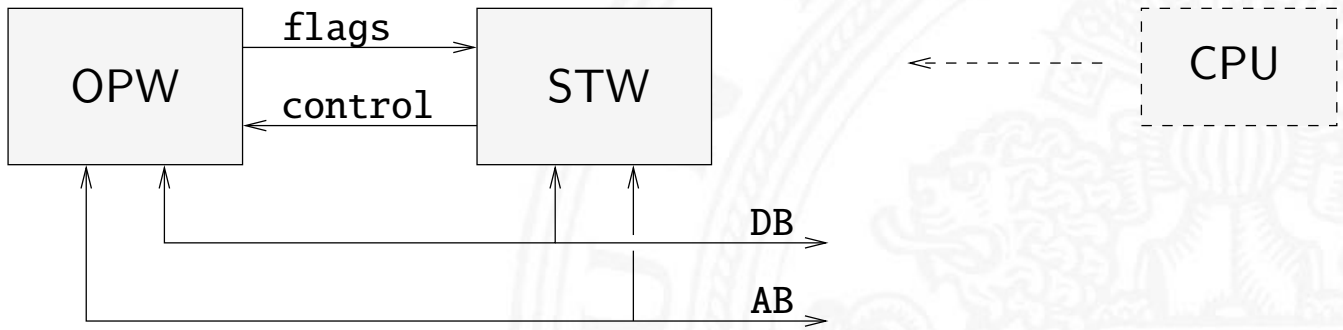
Abstraktionsebenen

- keine einheitliche Bezeichnung in der Literatur
- Architekturebene
  - Funktion/Verhalten Leistungsanforderungen
  - Struktur Netzwerk
    aus Prozessoren, Speicher, Busse, Controller...
  - Nachrichten Programme, Prokolle
  - Geometrie Systempartitionierung



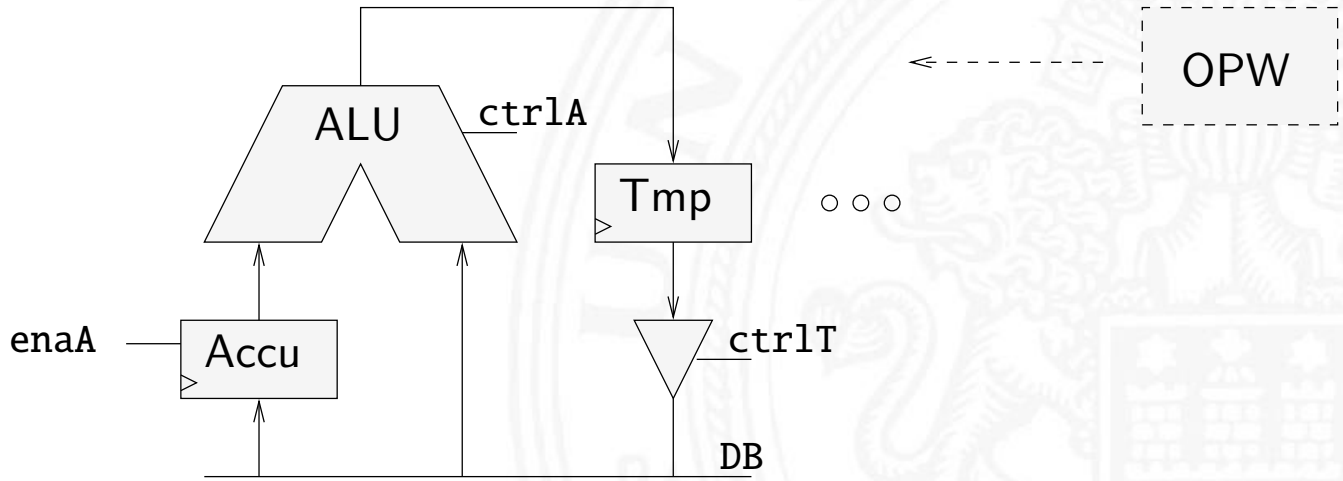
Abstraktion im VLSI-Entwurf (cont.)

- Hauptblockebene (Algorithmenebene, funktionale Ebene)
  - Funktion/Verhalten Algorithmen, formale Funktionsmodelle
  - Struktur Blockschaltbild
    aus Hardwaremodule, Busse...
  - Nachrichten Prokolle
  - Geometrie Cluster



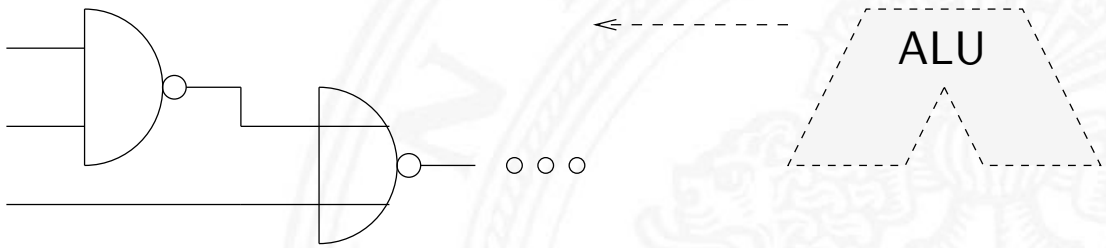
Abstraktion im VLSI-Entwurf (cont.)

- Register-Transfer Ebene
  - Funktion/Verhalten Daten- und Kontrollfluss, Automaten...
  - Struktur RT-Diagramm
    aus Register, Multiplexer, ALUs...
  - Nachrichten Zahlencodierungen, Binärworte...
  - Geometrie Floorplan



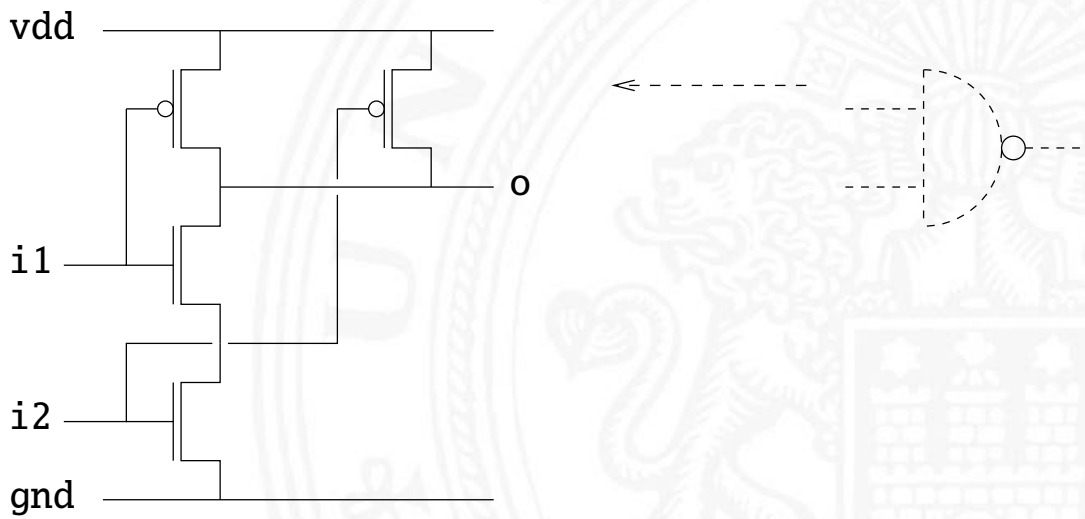
Abstraktion im VLSI-Entwurf (cont.)

- Logikebene (Schaltwerkebene)
  - Funktion/Verhalten Boole'sche Gleichungen
  - Struktur Gatternetzliste, Schematic
    aus Gatter, Flipflops, Latches...
  - Nachrichten Bit
  - Geometrie Moduln



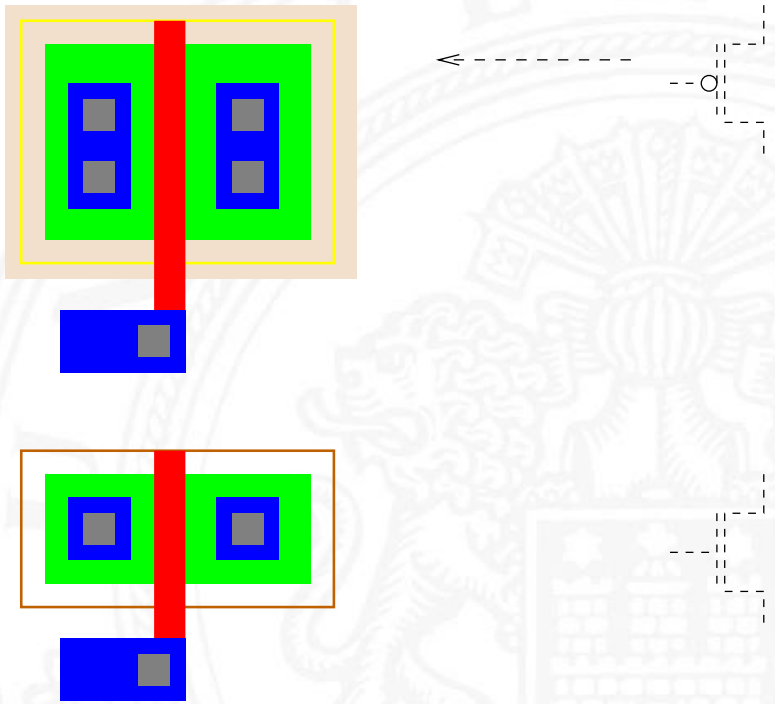
Abstraktion im VLSI-Entwurf (cont.)

- elektrische Ebene (Schaltkreisebene)
  - Funktion/Verhalten Differentialgleichungen
  - Struktur elektrisches Schaltbild
    aus Transistoren, Kondensatoren...
  - Nachrichten Ströme, Spannungen
  - Geometrie Polygone, Layout → physikalische Ebene



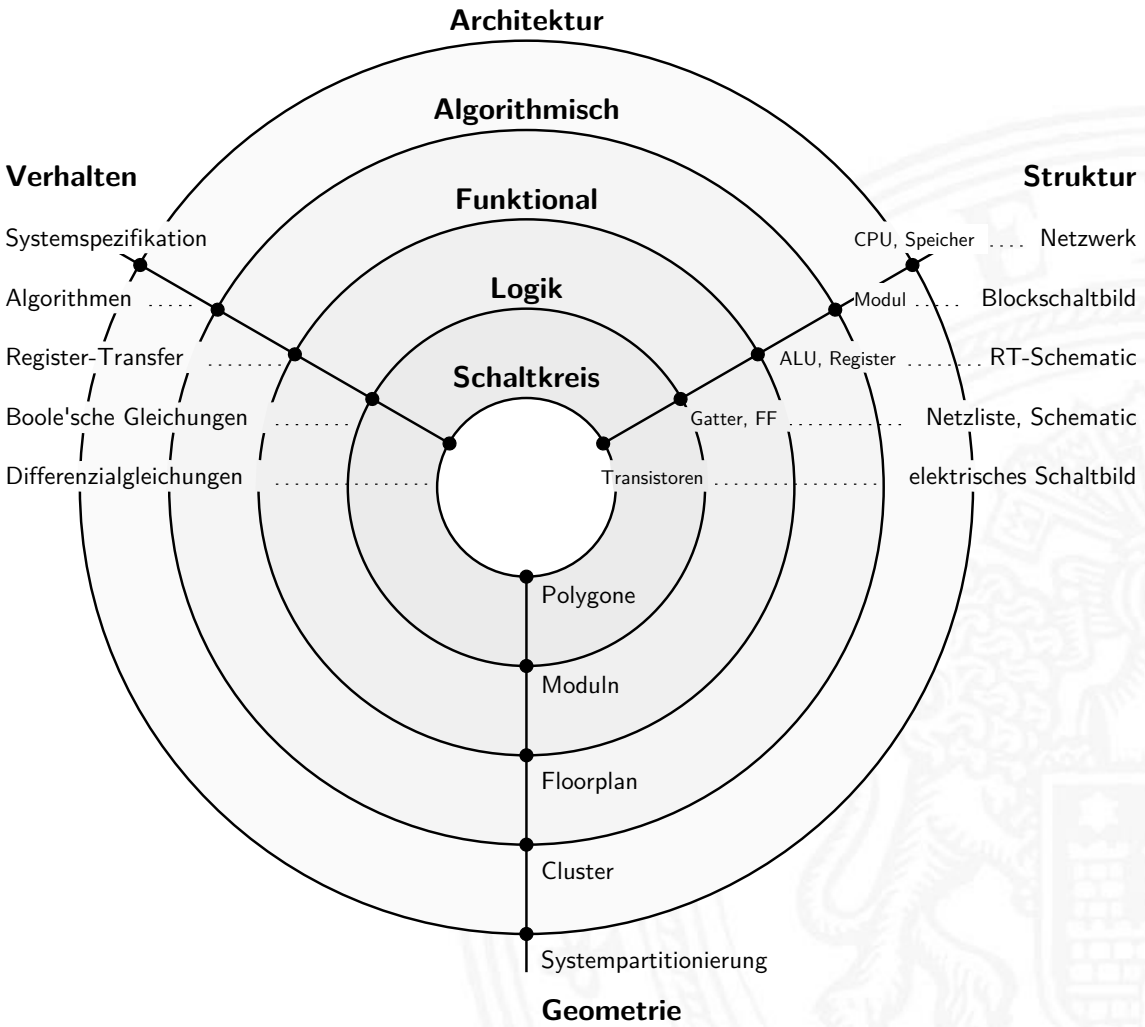
Abstraktion im VLSI-Entwurf (cont.)

- physikalische Ebene (geometrische Ebene)
  - Funktion/Verhalten partielle DGL
  - Struktur Dotierungsprofile



Abstraktion im VLSI-Entwurf (cont.)

Y-Diagramm



D. Gajski, R. Kuhn 1983:
„New VLSI Tools“

Abstraktion im VLSI-Entwurf (cont.)

Y-Diagramm / Gajski-Diagramm

- Visualisiert Abstraktionsebenen
- Sichtweisen
  - Funktion / Verhalten
  - Struktur
  - Geometrie (historisch, inzwischen überholt)

Entwurfsvorgehen

- Unterscheidung von *Struktur* und *Verhalten*
- Auf jeder Abstraktionsebene gibt es *elementare Einheiten* mit definiertem Verhalten
- Entwurfsaufgabe
  - ein gegebenes Verhalten in eine Strukturbeschreibung (aus elementaren Einheiten) der jeweiligen Ebene umzusetzen
  - jede dieser Einheiten ist ihrerseits in der nächstniedrigeren Abstraktionsebene entsprechend zu realisieren

  $\Rightarrow$ hierarchischer Entwurf, top-down

Entwurfsvorgehen (cont.)

⇒ top-down: typisches Entwurfsvorgehen

⇒ bottom-up: Einflüsse auf höhere Abstraktionsebenen

- Zeitverhalten
- Schaltungstechniken
- Arithmetiken
- ...

- Zentrale Bedeutung der Simulation, bzw. der Verifikation
- Entwurf als iterativer Prozess
  - Alternativen: „exploring the design-space“
  - Versionen
  - Teamarbeit

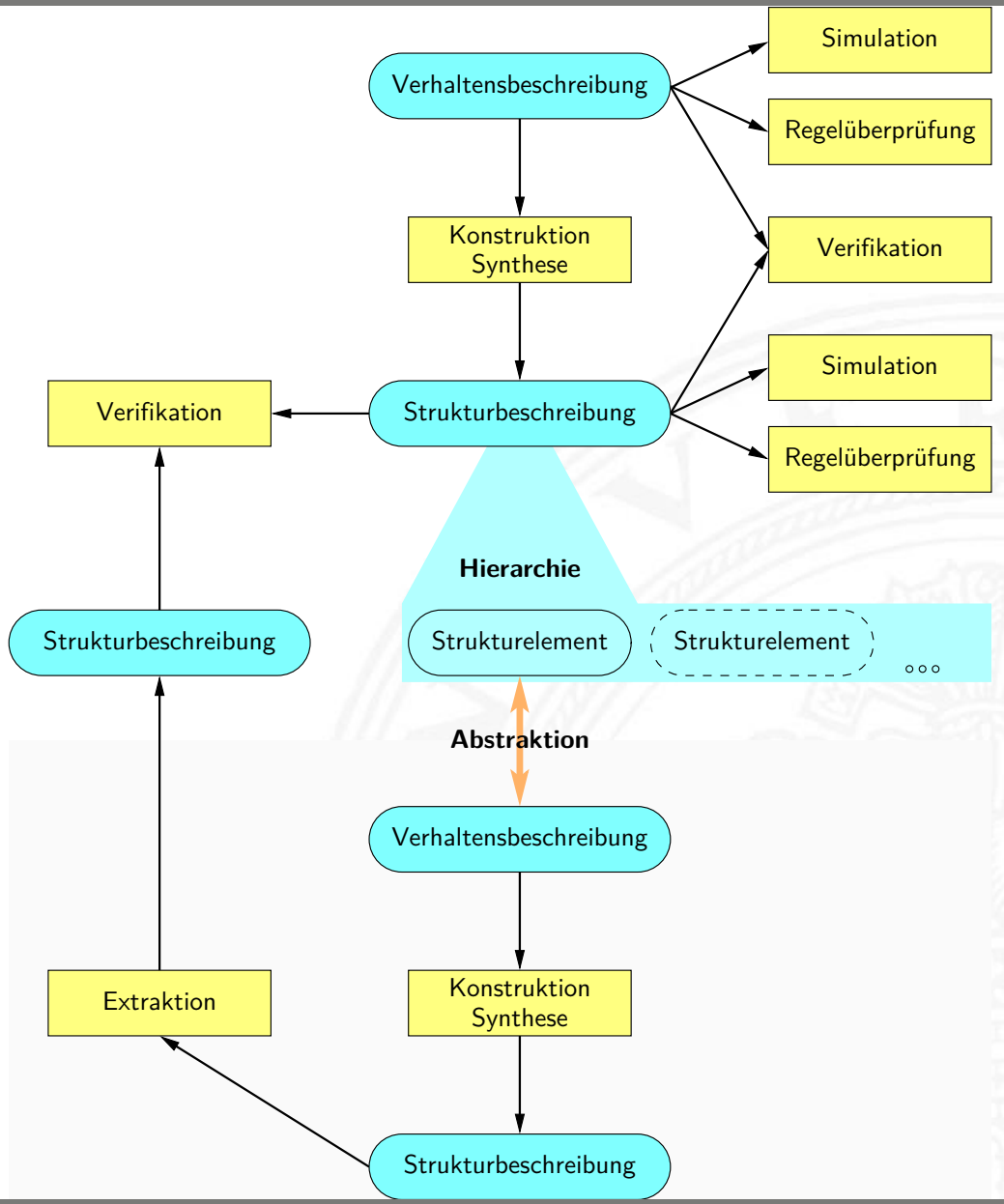
Hierarchischer Entwurf

Nur durch neue Methoden und Werkzeuge konnte die Produktivität beim Chipentwurf während der letzten Jahre mit Moore's Law mithalten

- Änderungen in der Entwurfsmethodik

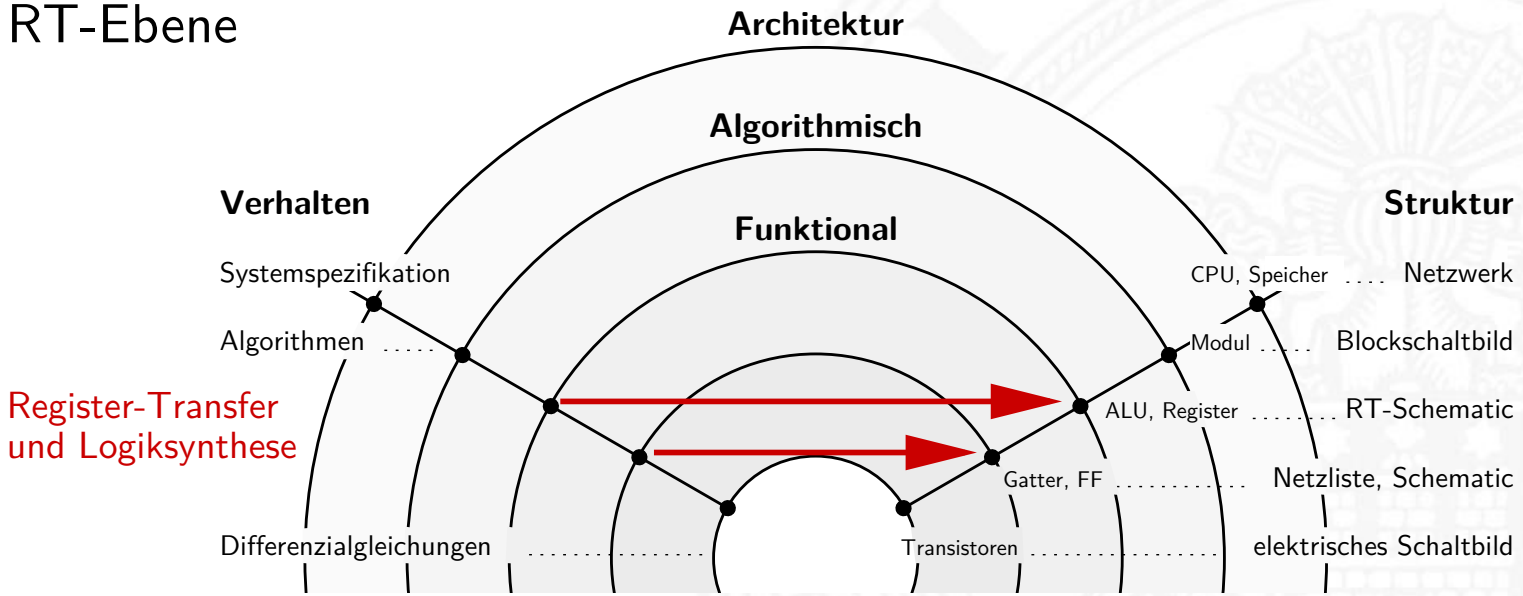
  Struktur $\Rightarrow$ Verhalten
  grafische Eingabe $\Rightarrow$ Hardwarebeschreibungssprachen
- Entwurf auf höheren Abstraktionsebenen
- Automatische Transformationen bis zum Layout
  - Synthese: Register-Transfer, High-Level
  - Datenpfad-/Makrozellgenerierung
  - Zellsynthese
  - Platzierung & Verdrahtung

Entwurfswerkzeuge



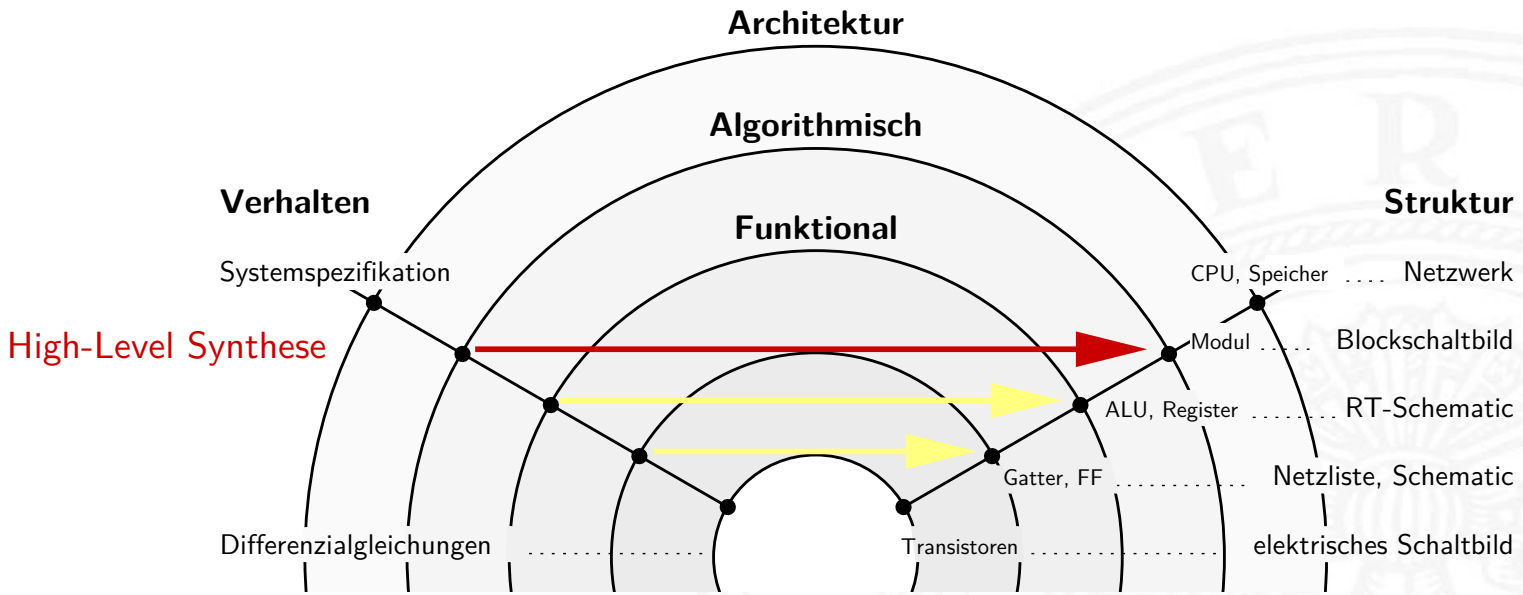
Entwurfswerkzeuge

- Synthese
  - = automatische Generierung von Strukturbeschreibungen aus Verhaltensmodellen
  - Trend: IP-Komponenten (**I**ntellectual **P**roperty) und „behavioral Code“
  - RT-Ebene



Entwurfswerkzeuge (cont.)

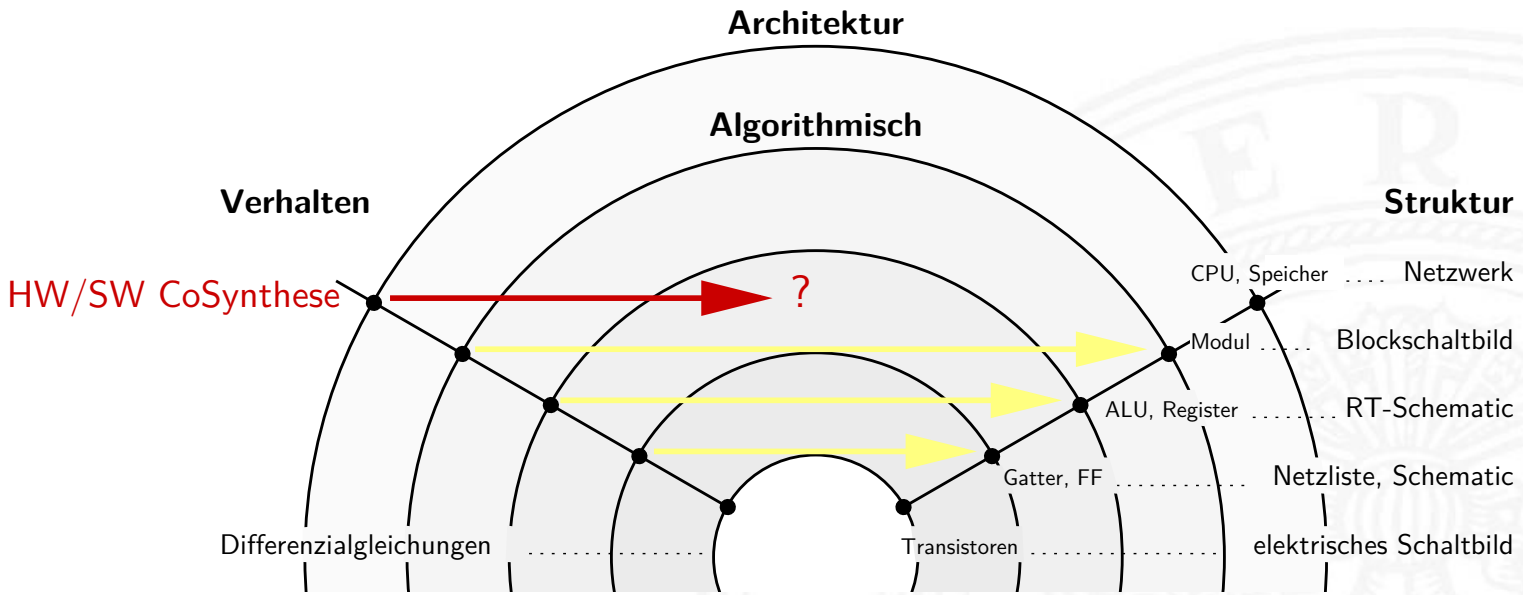
- High-Level Synthese



  - Einschränkung des „Suchraums“
  - spezielle Zielarchitekturen
  - spezielle Anwendungsfelder
  - Datenflussdominiert DSPs
    Kontrollflussdominiert Prozessoren

Entwurfswerkzeuge (cont.)

- CoDesign → CoSynthese



- Partitionierung Hardware / Software ?
- nur manuell möglich

Entwurfswerkzeuge (cont.)

- Simulation
  - Trend: wachsender Aufwand, Systemsimulation
  - Problem der Simulationsauswertung $\Rightarrow$ auch dort Abstraktion
    - Programmiersprachen-Schnittstellen (VHPI, Verilog-PLI...)
      Beispiele: ▸ Signalverarbeitung ▸ Bildverarbeitung
  - Hardwarebeschleunigung
    - Emulation von Gatternetzlisten durch FPGA-Boards
    - Beispiel: Betriebssystem auf Simulationsmodell vom Mikroprozessor booten (Sun Microsystems)
  - gemischte Simulation
    - Hardware- und Software
    - auf verschiedenen Abstraktionsebenen
    - + IP-Modelle
    - + analoge Modelle

Entwurfswerkzeuge (cont.)

- Analysewerkzeuge
  - Leistungsverbrauch
  - Timing
  - jeweils: statisch, geschätzt oder in Verbindung mit Simulation
- Verifikation, wenn möglich

  = Verifikation: Aussagen gelten *für alle möglichen* Eingaben
  Simulation: Beschränkung auf Stimuli
  - formale Methoden, um Eigenschaften zu überprüfen
  - meist Vergleich verschiedener Modelle
    - in Verbindung mit Extraktion
    - Referenzmodell, woher?
  - Ersatz von Simulationen

Entwurfswerkzeuge (cont.)

- Layoutwerkzeuge / Platzierung & Verdrahtung
  - NP-vollständige Probleme

  $\Rightarrow$ Heuristiken

  $\Rightarrow$ sehr starke Spezialisierung, z.B. Routing bei Standardzell Entwürfen:
    1. Verdrahtung der Spannungsversorgung: Power-Routing
    2. Clock-Tree Synthese / -Routing
    3. zeitkritische Netze bearbeiten: „constraint driven“ Routing
    4. normale Verdrahtung
    5. nachträgliche Optimierung: DRC-Fehler, thermische Modelle...
- Test des Entwurfs

  $=$ Testbarkeit: Fertigungsfehler (physikalisch) feststellen

  Simulation: Überprüfung der Funktion
  - Ziel: defekte ICs aussortieren, *vor* Verpackung in Gehäuse

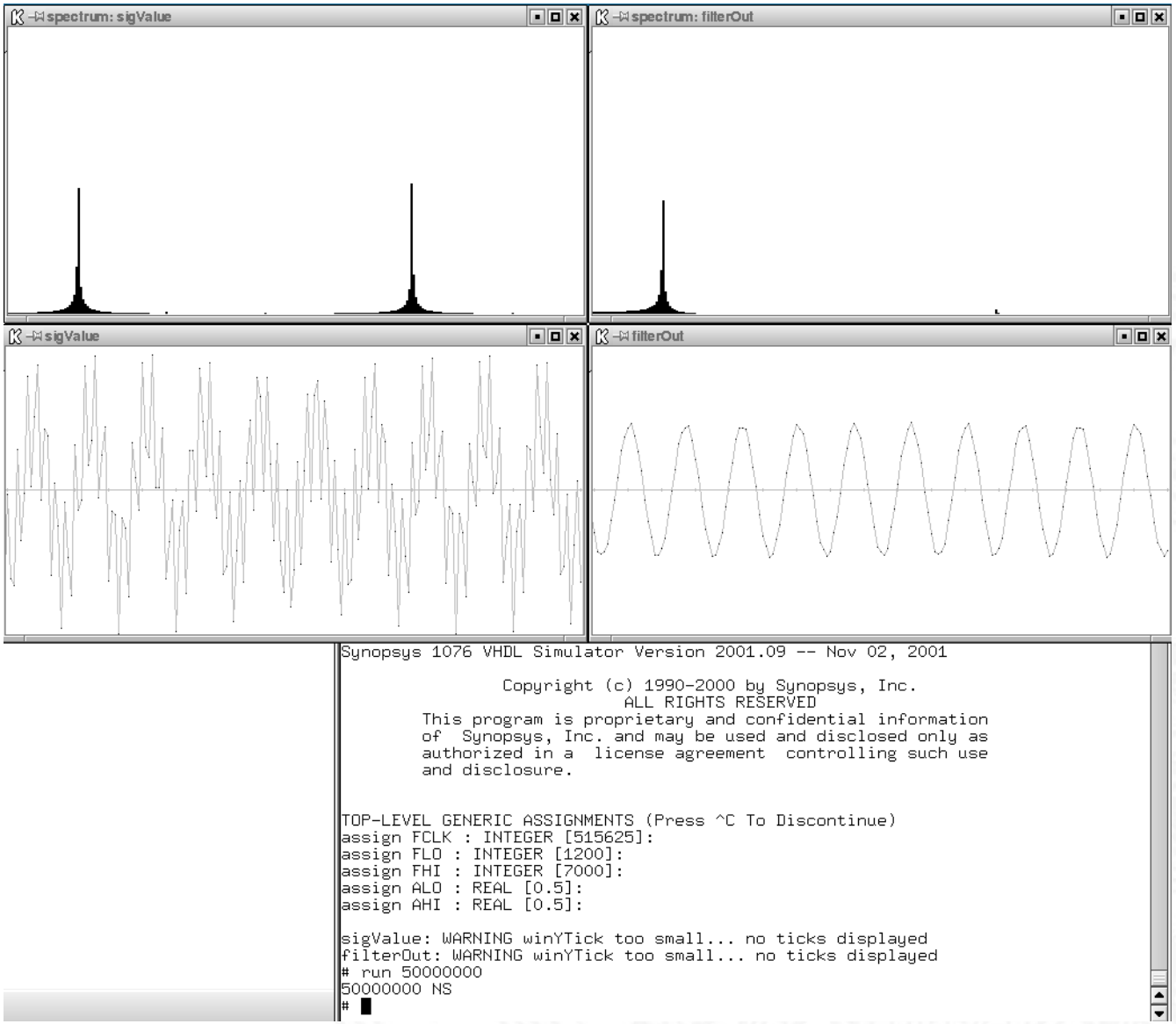
Entwurfswerkzeuge (cont.)

- Problem
  - *alle internen Leitungen/Gatter ansprechen*
  - *nur die Padzellen sind direkt zugänglich*
- Fehlermodelle: „stuck-at“, bridging, open...
- Verfahren um Testbarkeit zu gewährleisten
  - Selbsttest, z.B. BIST (**B**uild **I**n **S**elf **T**est)
  - Scan-Path: Flipflops als Schieberegister
  - ...

  - Dabei wird zusätzliche Logik integriert (bis zu 30%)
  - (teil-)automatisch bei der Synthese
- Fehlersimulation: überprüft die Fehlerüberdeckung
  „Wieviele Fehler können erkannt werden?“
- Testmustergenerierung: erzeugt automatisch Testvektoren

Beispiel

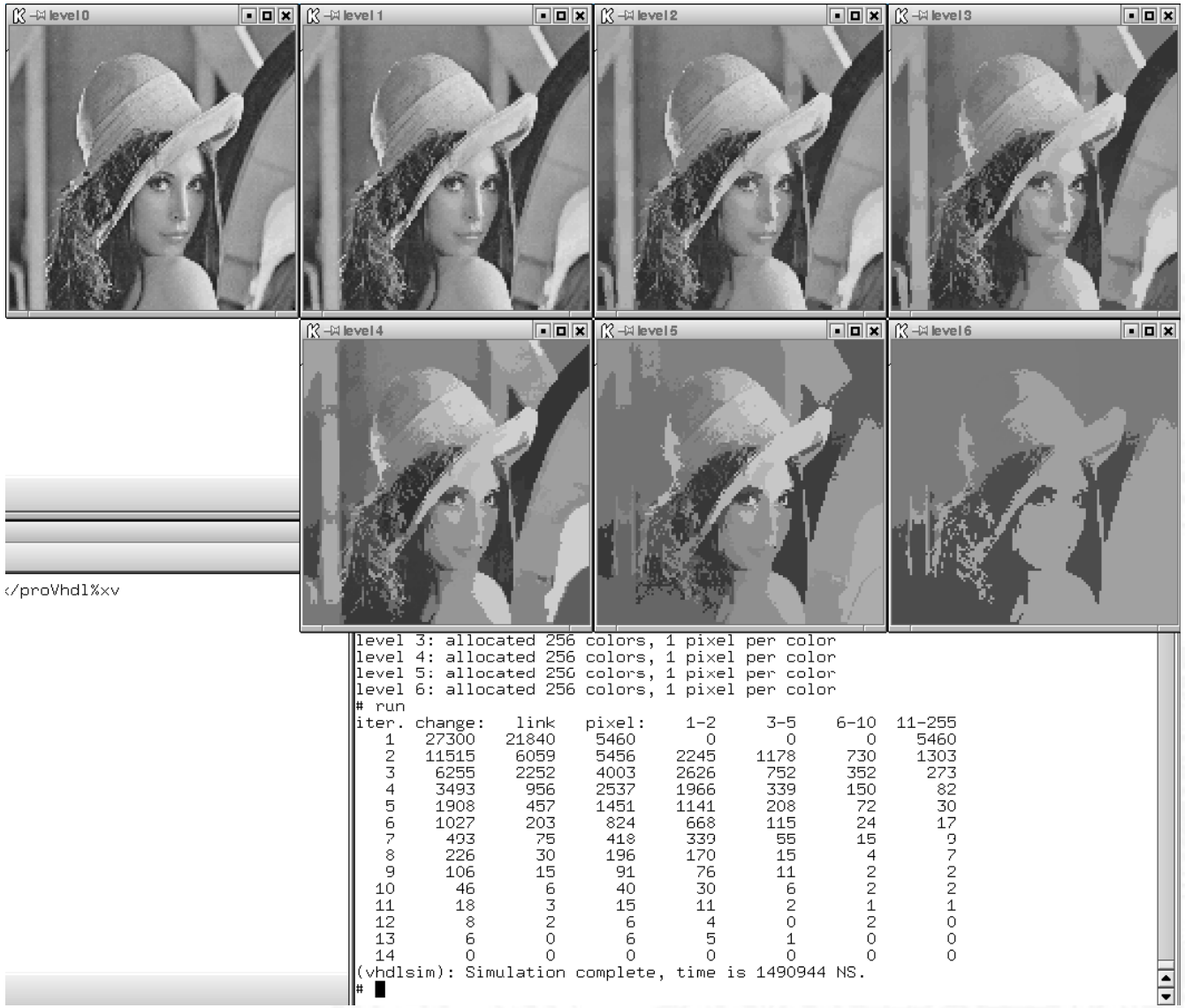
- Signalverarbeitung
- digitales Filter



◂ Simulation

Beispiel

- Bildverarbeitung
- Segmentierung

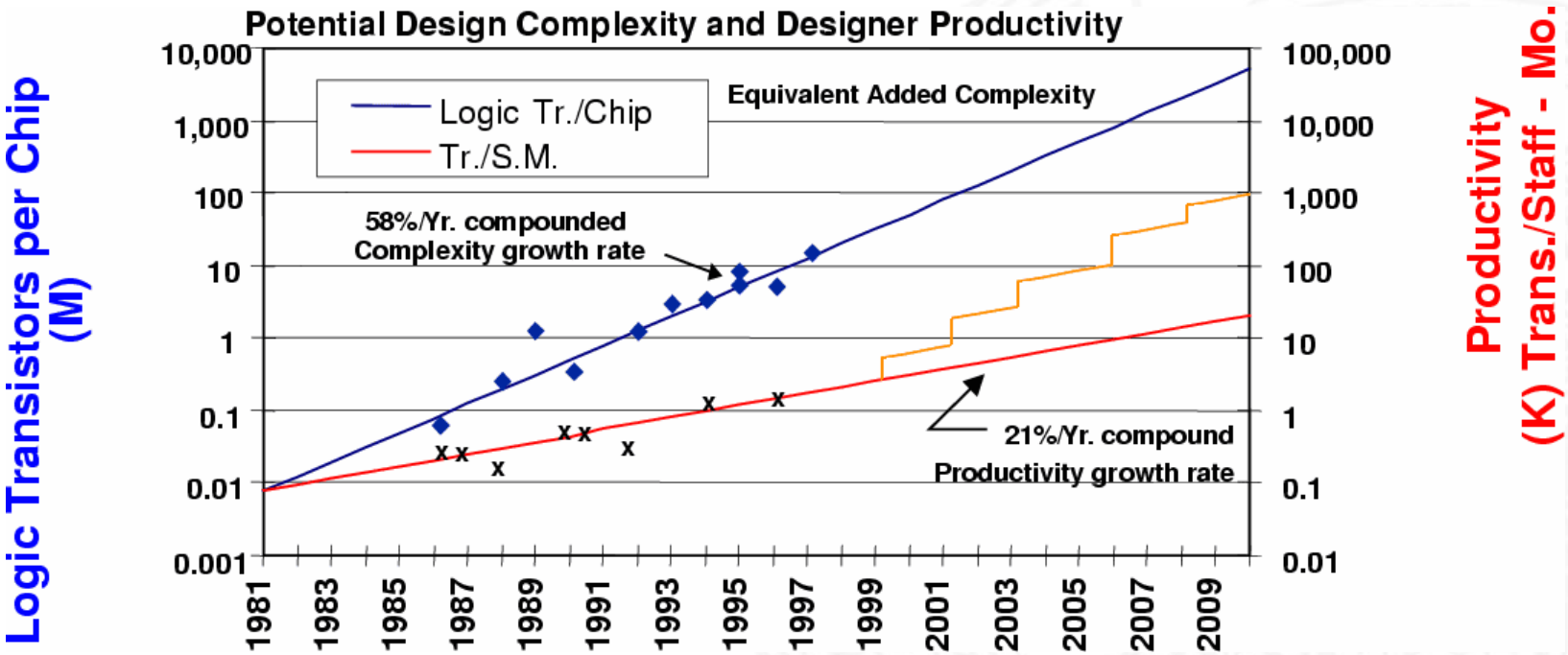


◂ Simulation

Probleme

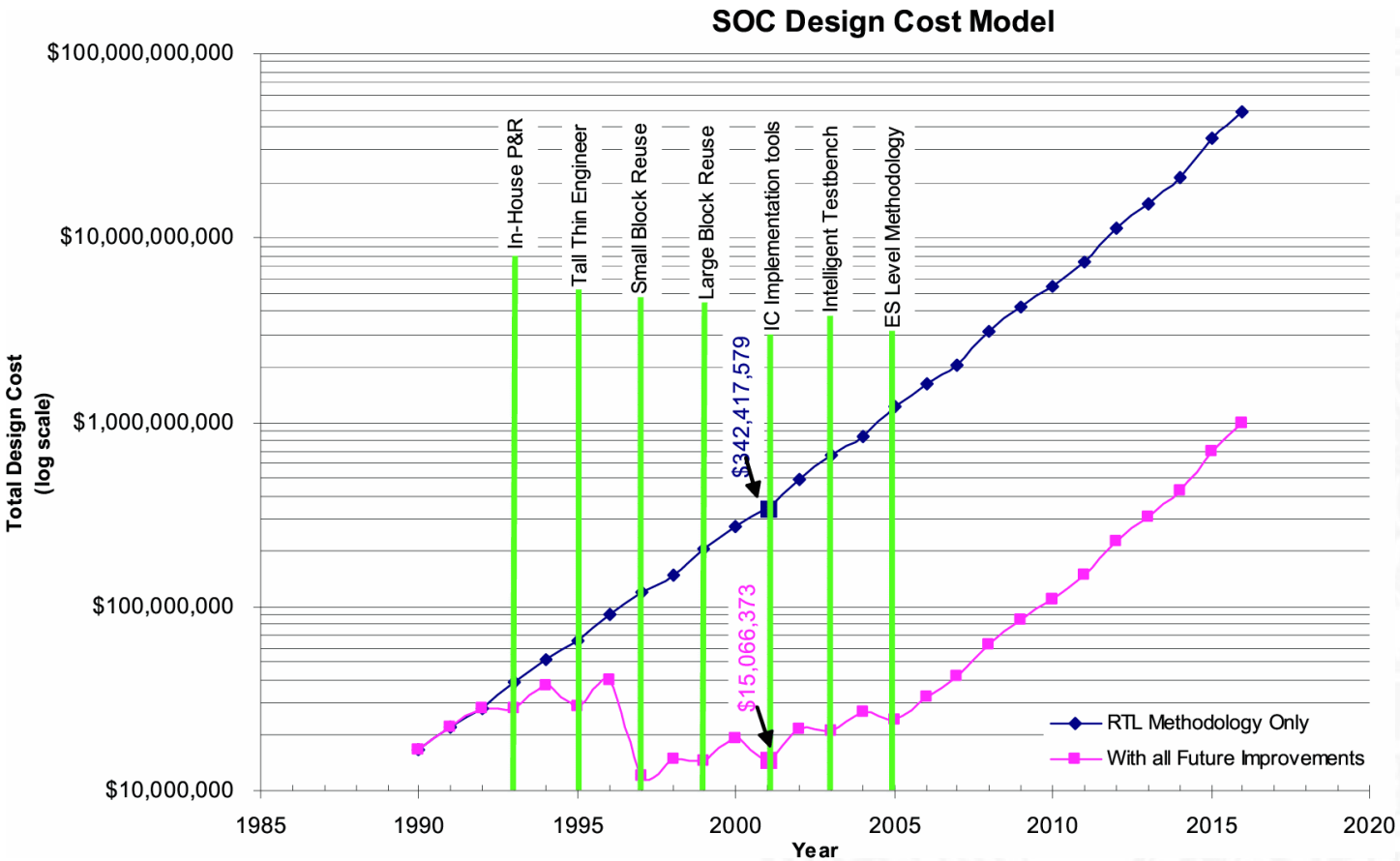
Moore's Law heisst in der Praxis

- Entwurf immer größerer und komplexerer Systeme
- Produktivitätssteigerungen



Probleme (cont.)

– Entwurfskosten



Probleme (cont.)

- Geänderte Systemanforderungen
  - Performance
  - Größe
  - ökonomische Randbedingungen
  - Low-Power: Leistungsaufnahme, Abwärme...
  - Umgebung: EMV, Temperatur, mechanische Eigenschaften...

  – Wie können all diese Anforderungen (formal) spezifiziert werden?

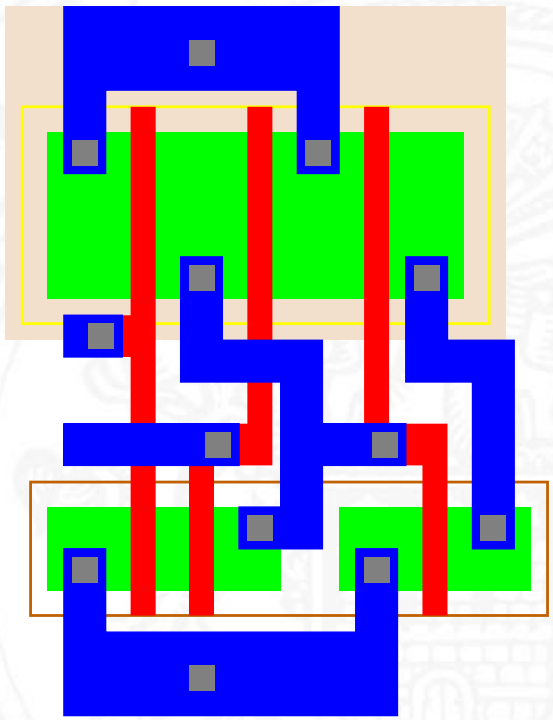
Entwurfsstile

- mehrere Möglichkeiten Schaltungen zu entwerfen
- Unterscheidungsmerkmale
  - Zeitaufwand: Entwurfsdauer, Fertigungszeit
  - Kosten: Fertigung, pro Stück, EDA-Werkzeuge
  - IC-Eigenschaften: Größe, Taktfrequenz, Leistungsaufnahme...
- Entwurfsstile
  - Full-Custom
  - Standardzell
  - Gate-Array
  - FPGA / programmierbare Schaltungen

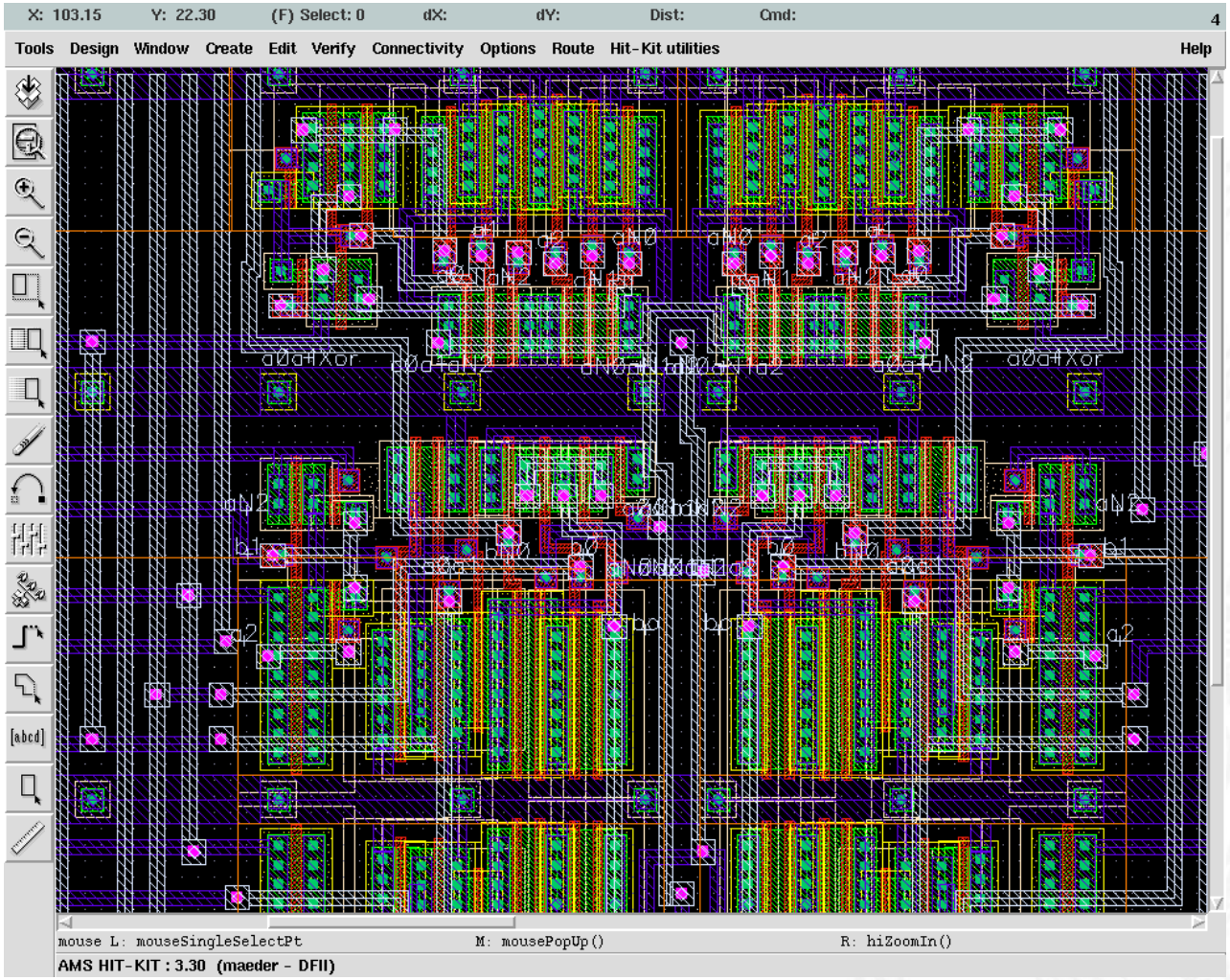
Full-Custom

Vollkundenspezifischer Entwurf / Full-Custom

- Layout aller geometrischer Strukturen
- viel manuelle Arbeit mit Layout-Editoren
- optimal kleine, schnelle Entwürfe
- sehr lange Entwurfsdauer (Effizienz)
- Ausnutzen von Regularität
- Teamarbeit nötig, Schnittstellen
- erfordert erfahrene Entwerfer



Full-Custom (cont.)

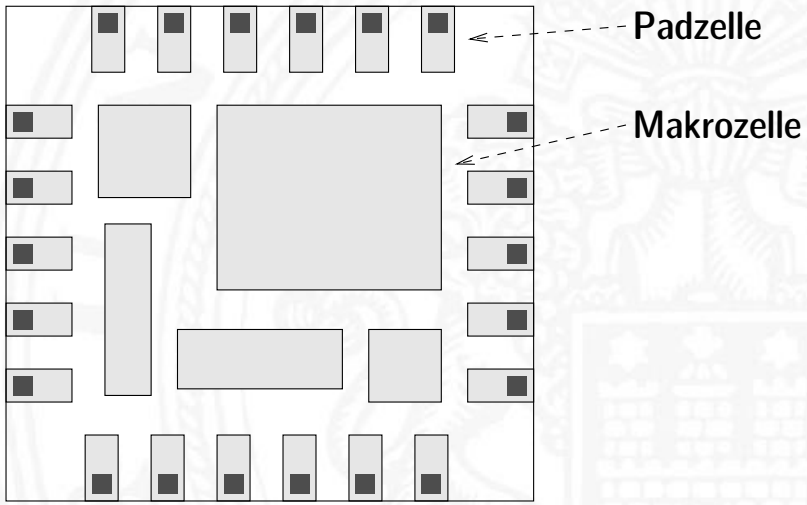


Makrozellentwurf

Makrozellentwurf

- Zellen wie Speicher, ALUs oder Datenpfade werden über Generatoren erzeugt
- Makrozellen in Full-Custom Qualität
- meist in Verbindung mit Standardzellentwurf

Chipgröße	variabel
Zellenanzahl	variabel
Zellengröße	variabel
Anschlusslage	variabel
Leiterbahnkanäle	variabel

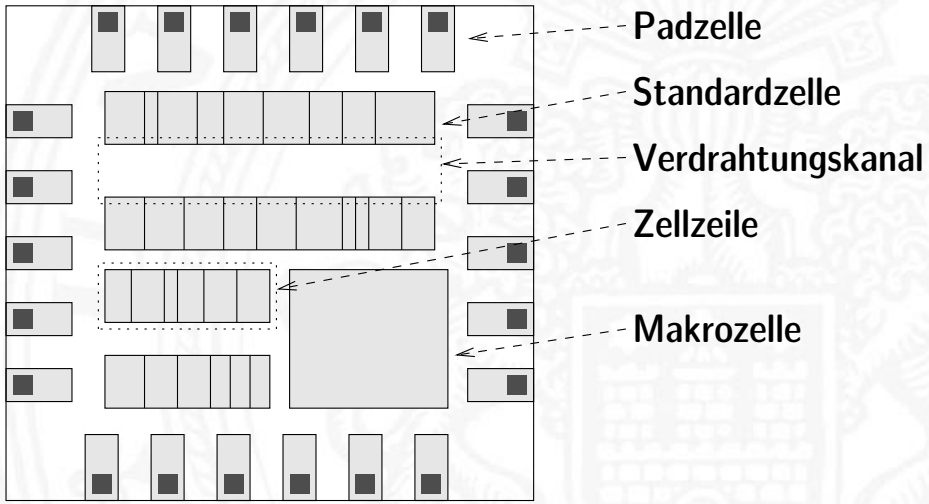


Standardzellentwurf

Standardzellentwurf

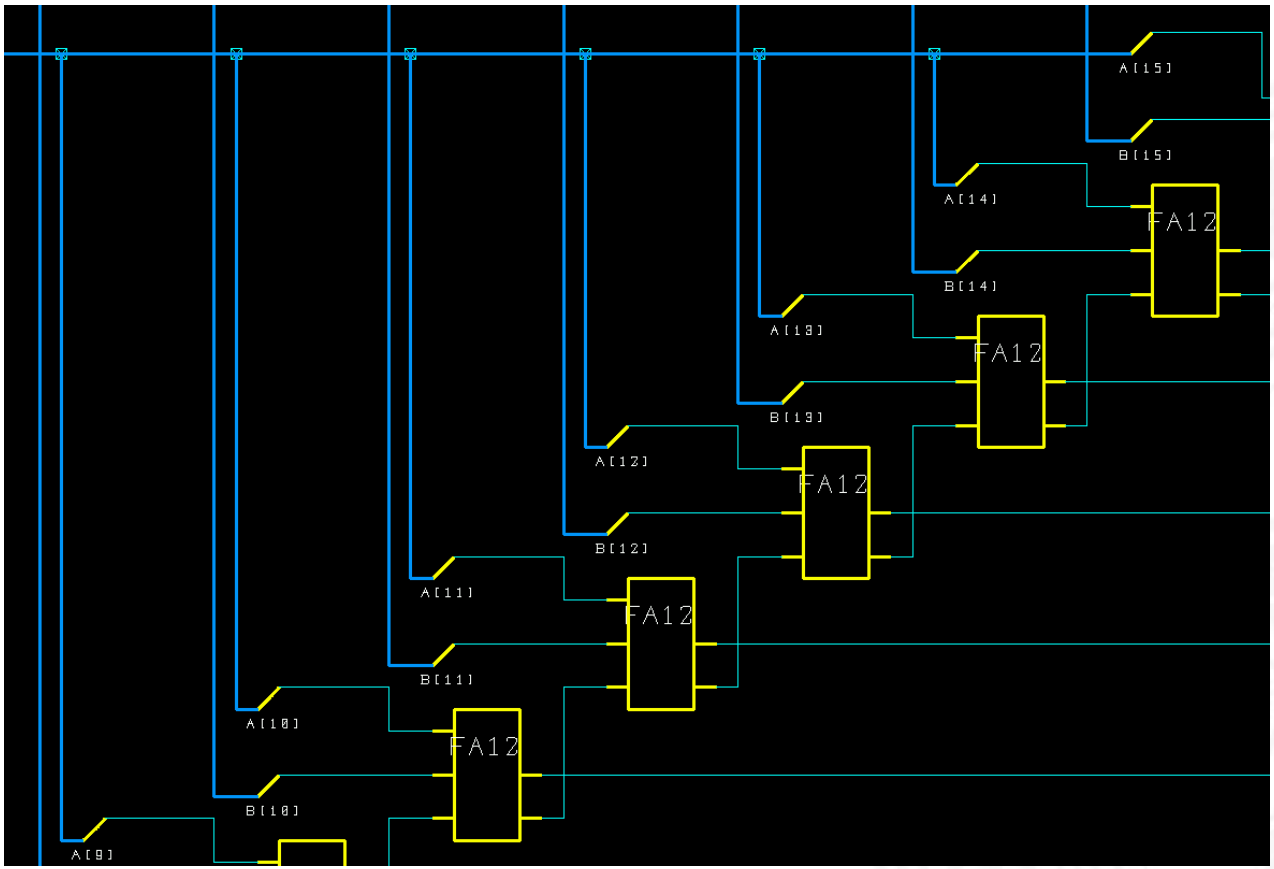
- vorgefertigte Zellen aus Bibliotheken benutzen
- Layout der Standardzellen in Full-Custom Qualität
- schneller flexibler Entwurf
- meist in Verbindung mit Makrozellgeneratoren

Chipgröße	variabel
Zellenanzahl	variabel
Zellenhöhe	fest
Zellenbreite	variabel
Anschlusslage	variabel
Leiterbahnkanäle	variabel

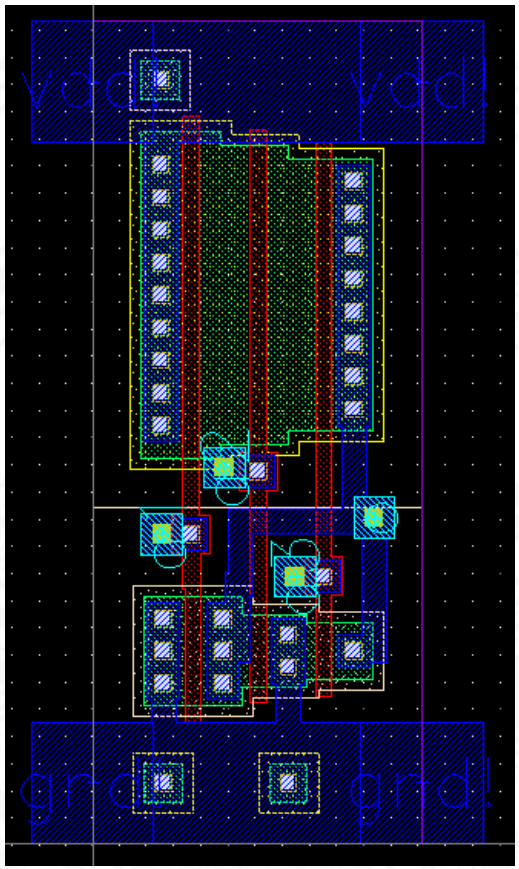


Standardzellentwurf (cont.)

Schematic

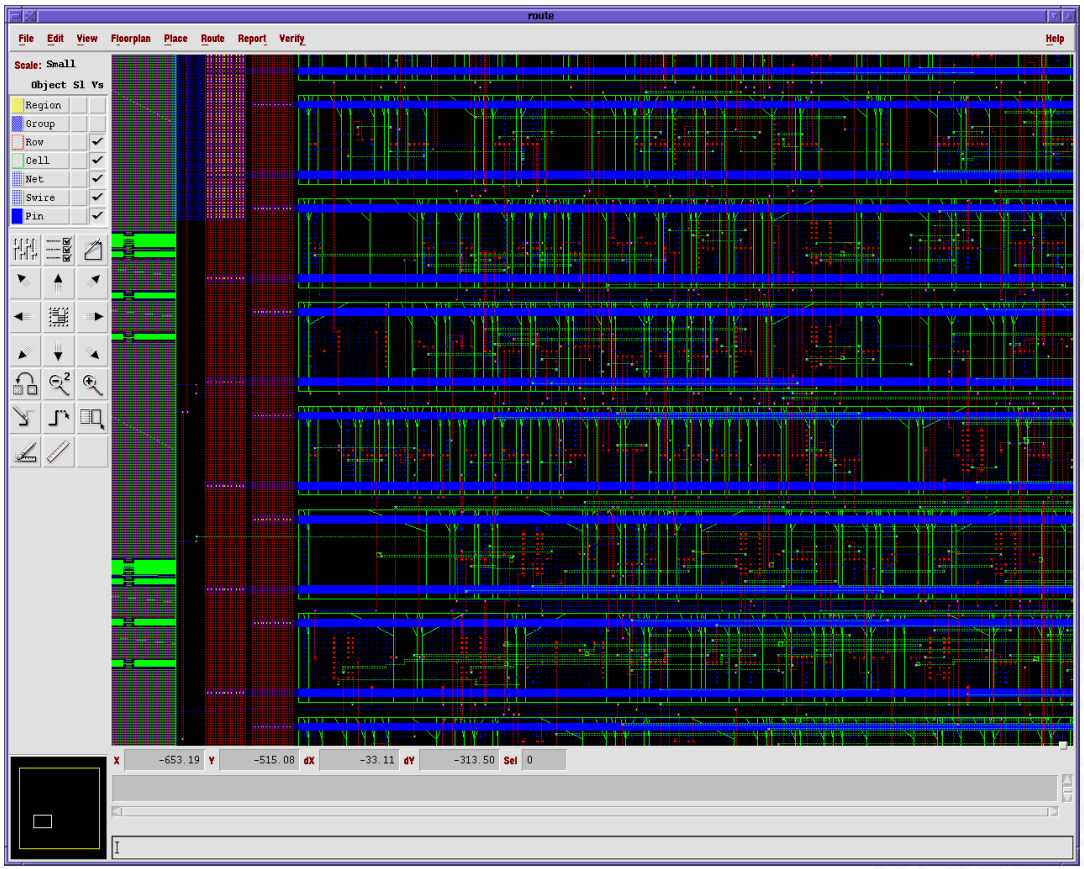


Zell-Layout



Standardzellentwurf (cont.)

Standardzell Layout

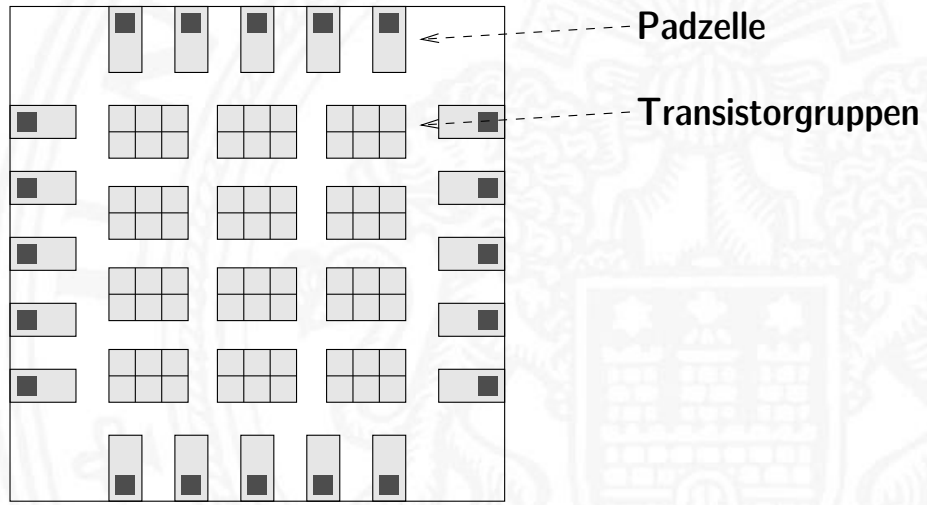


Gate-Array Entwurf

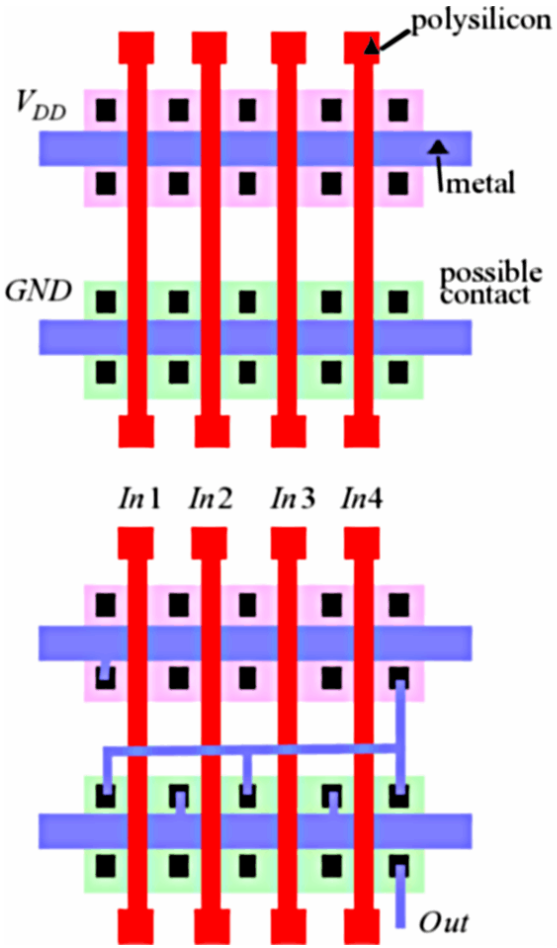
Gate-Array / Sea-of-Gate Entwurf

- vorgefertigte Transistoren
- Layout durch Verbindungsstruktur (Verdrahtung, Kontakte)
- intra-Zell Verdrahtung aus Zellbibliotheken
- vorgegebene Master: Komplexität eingeschränkt, Verschnitt
- schnelle Verfügbarkeit

Chipgröße	fest
Zellenanzahl	fest
Zellengröße	fest
Anschlusslage	fest
Leiterbahnkanäle	fest



Gate-Array Entwurf (cont.)



Uncommited Cell

Committed Cell (4-input NOR)

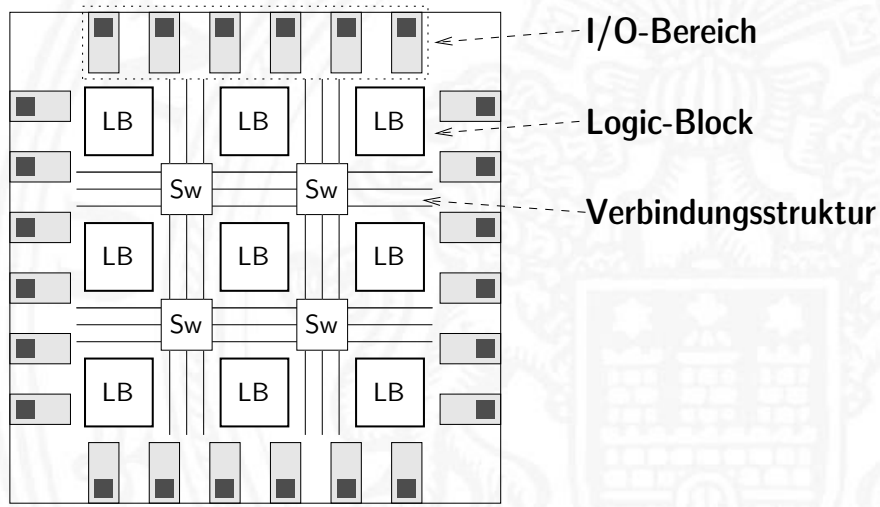
Gate-Array

programmierbare Schaltungen

programmierbare Schaltungen: FPGA, PLD, LCA...

- fertig vorgegebene Schaltung: Logik und Verbindungsstruktur
- Entwurf: Programmierung durch Anwender ⇒ sofort verfügbar
- Einschränkung durch vorgegebene Struktur
- Rekonfiguration möglich
- in-Circuit programmierbar

Chipgröße	fest
Blockanzahl	fest
Anschlusslage	fest
Verbindungsnetz	fest
Blockfunktion	progr.
Verbindungen	progr.

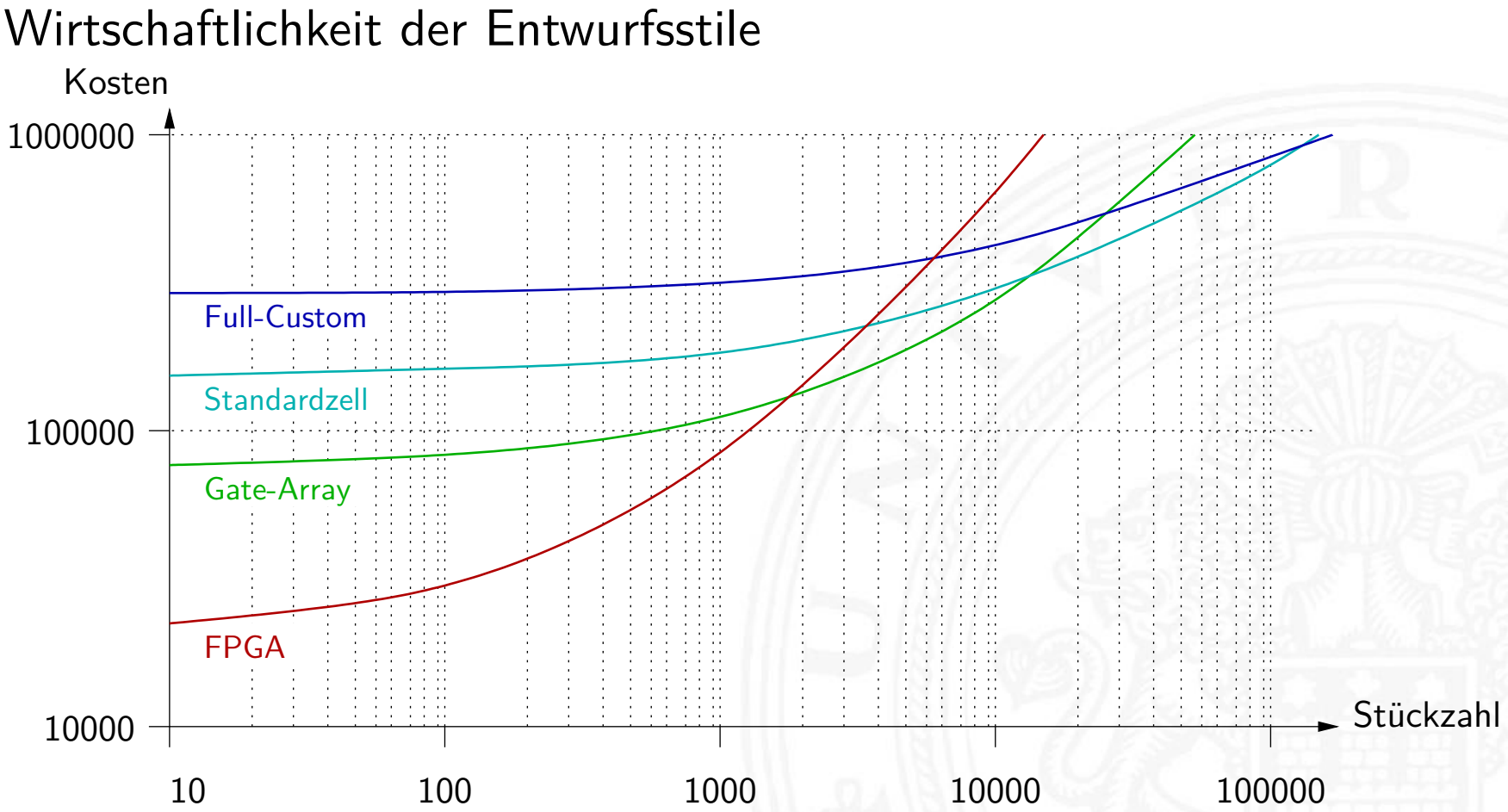


Vergleich der Entwurfsstile

Tabellarische Übersicht

Stil	*Performance*	*Fläche*	*Kosten (IC)*	*Kosten (Design)*	*time-to-Market*	*Prozessschritte*	*Stückzahlen*
Full-Custom	$+++$	$+++$	$+++$	$---$	$---$	voll	10^5
Standard-/Makrozell	$++$	$++$	$++$	$--$	$--$	voll	10^4
Gate-Array	$+$	$\circ$	$+$	$\circ$	$\circ$	4-10	10^3
programmierbare Logik	$-$	$--$	$--$	$++$	$+++$	0	$< 10^3$

Vergleich der Entwurfsstile (cont.)



Vergleich der Entwurfsstile (cont.)

Wahl des Entwurfsstils

- Kostenüberlegungen
- Entwurfsdauer: „time-to-Market“
- technische Randbedingungen, oft als K.O.-Kriterium
  - Fläche
  - Leistungsaufnahme
  - Sicherheitsaspekte
- organisatorische Randbedingungen
  - vorhandene Werkzeuge
  - Know-How
  - „Faktor: Mensch“ (Erfahrungen, Vorlieben)

$\Rightarrow$ vielfältige Wechselwirkungen

VHDL

VHDL

VHSIC **H**ardware **D**escription **L**anguage
Very **H**igh **S**peed **I**ntegrated **C**ircuit

- digitale Systeme
  - Modellierung/Beschreibung
  - Simulation
  - Dokumentation
- Komponenten
  - Standard ICs
  - anwendungsspezifische Schaltungen: ASICs, FPGAs
  - Systemumgebung: Protokolle, Software, ...

VHDL (cont.)

- Abstraktion
  - von der Spezifikation – Algorithmen und Protokolle
  - über die Implementation – Register-Transfer Modelle
  - bis hin zum fertigen Entwurf – Netzliste mit Backannotation

  $\Rightarrow$ VHDL durchgängig einsetzbar

  $\Rightarrow$ Simulation immer möglich

Entwicklung

- 1983 vom DoD initiiert
- 1987 IEEE Standard IEEE 1076
- 2004 IEC Standard IEC 61691-1-1
- regelmäßige Überarbeitungen: VHDL'93, VHDL'02, VHDL'08

VHDL (cont.)

Erweiterungen

- Hardwaremodellierung/Zellbibliotheken — IEC 61691-2, IEC 61691-5
- Hardwaresynthese — IEC 61691-3-3, IEC 62050
- mathematische Typen und Funktionen — IEC 61691-3-2
- analoge Modelle und Simulation — IEC 61691-6

VHDL (cont.)

Links

- http://tams.informatik.uni-hamburg.de/research/vlsi/vhdl
- http://www.vhdl.org
- http://www.vhdl-online.de
- http://en.wikipedia.org/wiki/VHDL
- http://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language
- http://www.asic-world.com/vhdl

VHDL – sequenziell

- Typen, Untertypen, Alias-Deklarationen
  - skalar `integer`, `real`, `character`, `boolean`, `bit`, Aufzählung
  - komplex `line`, `string`, `bit_vector`, Array, Record
  - Datei `text`, File
  - Zeiger Access
  - strikte Typbindung
  - Konvertierungsfunktionen
- Objekte `constant`, `variable`, `file`

VHDL – sequenziell (cont.)

- Operatoren
  1. logisch — `and`, `or`, `nand`, `nor`, `xor`, `xnor`
  2. relational — =, /=, <, <=, >, >=
  3. schiebend — `sll`, `srl`, `sla`, `sra`, `rol`, `ror`
  4. additiv — +, -, &
  5. vorzeichen — +, -
  6. multiplikativ — *, /, `mod`, `rem`
  7. sonstig — **, `abs`, `not`
- Anweisungen
  - Zuweisung — :=, <=
  - Bedingung — `if`, `case`
  - Schleifen — `for`, `while`, `loop`, `exit`, `next`
  - Zusicherung — `assert`, `report`
  - ...

VHDL – sequenziell (cont.)

- Sequenzielle Umgebungen
  - Prozesse `process`
  - Unterprogramme `procedure`, `function`
  - lokale Gültigkeitsbereiche
  - Deklarationsteil: definiert Typen, Objekte, Unterprogramme
    Anweisungsteil: Codeanweisungen sequenziell ausführen

⇒ Imperative sequenzielle Programmiersprache (z.B. Pascal)

- Beliebige Programme *ohne Bezug zum Hardwareentwurf* möglich
- Beispiel: Datei einlesen, verlinkte Liste erzeugen...

```
...
type      LIST_T;
type      LIST_PTR          is  access LIST_T;
type      LIST_T            is  record KEY   : integer;
                                       LINK  : LIST_PTR;
                            end record LIST_T;
```

VHDL – sequenziell (cont.)

```
constant  INPUT_ID          : string   := "inFile.dat";
file      DATA_FILE         : text;
variable  DATA_LINE         : line;
variable  LIST_P, TEMP_P    : LIST_PTR := null;

procedure READ_DATA is     -- Datei einlesen, Liste aufbauen
  variable  KEY_VAL         : integer;
  variable  FLAG            : boolean;
begin
  file_open (DATA_FILE, INPUT_ID, read_mode);
  L1: while not endfile(DATA_FILE) loop
      readline(DATA_FILE, DATA_LINE);
      L2: loop
          read(DATA_LINE, KEY_VAL, FLAG);
          if FLAG  then   TEMP_P := new LIST_T'(KEY_VAL, LIST_P);
                          LIST_P := TEMP_P;
                   else   next L1;
          end if;
      end loop L2;
  end loop L1;
  file_close(DATA_FILE);
end procedure READ_DATA;
...
```

VHDL – konkurrent

- ähnlich ADA'83
- Konkurrenter Code
  - mehrere Prozesse
  - Prozeduraufrufe
  - Signalzuweisung `<=`
    - bedingt `<= ... when ...`
    - selektiv `with ... select ... <= ...`
  - Zusicherung `assert`
  - modelliert gleichzeitige Aktivität der Hardwareelemente
- Synchronisationsmechanismus für Programmlauf / Simulation
  - Objekt signal
  - Signale verbinden konkurrent arbeitende „Teile“ miteinander
  - Entsprechung in Hardware: Leitung

VHDL – Simulation

- Semantik der Simulation im Standard definiert: *Simulationszyklus*
- konkurrent aktive Codefragmente
  - Prozesse + konkurrente Anweisungen + Instanzen (in Hierarchien)
  - durch Signale untereinander verbunden

„Wie werden die Codeteile durch einen sequenziellen Simulationsalgorithmus abgearbeitet?“

VHDL – Simulation (cont.)

- Signaltreiber: Liste aus Wert-Zeit Paaren

S: integer ←

NOW	+5 ns	+12 ns	+15 ns	+21 ns	+27 ns	Zeitpunkt
2	7	3	12	8	-3	Wert

- Simulationsereignis
  - Werteänderung eines Signals
  - (Re-) Aktivierung eines Prozesses nach Wartezeit

⇒ Ereignisgesteuerte Simulation
  - theoretisches Modell
  - veranschaulicht Sematik für den VHDL-Benutzer
  - praktische Implementation durch Simulationsprogramme weicht in der Regel davon ab
  - Optimierungsmöglichkeiten: Datenabhängigkeiten, zyklenbasierte Simulation ...

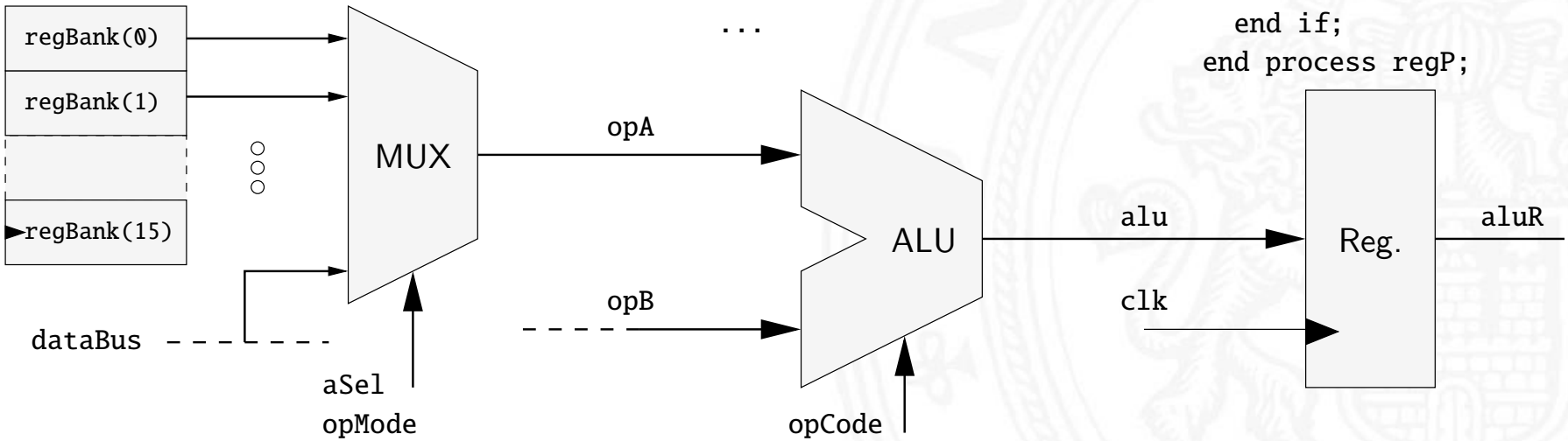
Ereignisgesteuerte Simulation

Beispiel: Datenpfad

```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB when opcAdd,
       opA  -  opB when opcSub,
       opA and opB when opcAnd,
       ...
```

```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation – Zyklus 1

1. Simulationsereignis

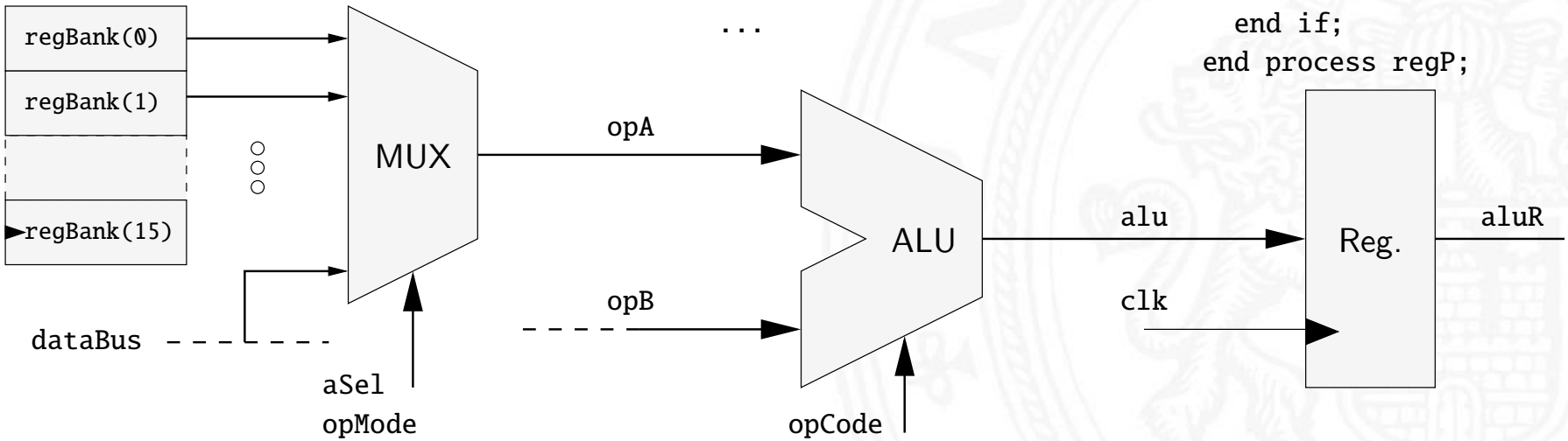
Ereignisliste

NOW	aSel	1
	opMode	regM
	opCode	opcAdd
+10 ns	clk	'1'
...		

```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB when opcAdd,
       opA  -  opB when opcSub,
       opA and opB when opcAnd,
       ...
```

```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



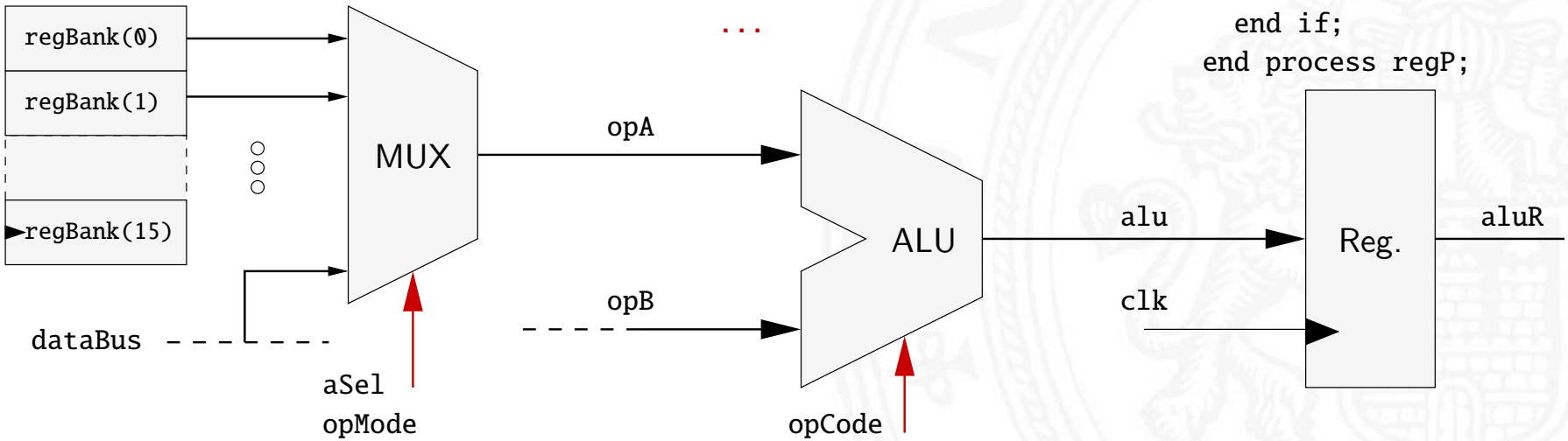
Ereignisgesteuerte Simulation – Zyklus 1

1. Simulationsereignis
2. Prozessaktivierung

```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

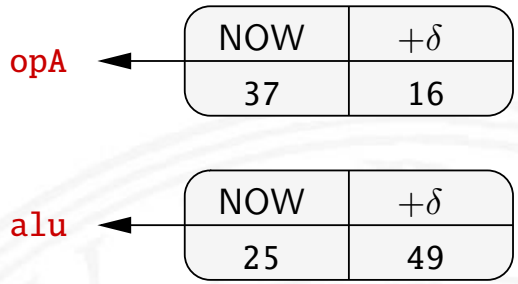
```
with opCode select
alu <= opA  +  opB when opcAdd,
       opA  -  opB when opcSub,
       opA and opB when opcAnd,
       ...
```

```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation – Zyklus 1

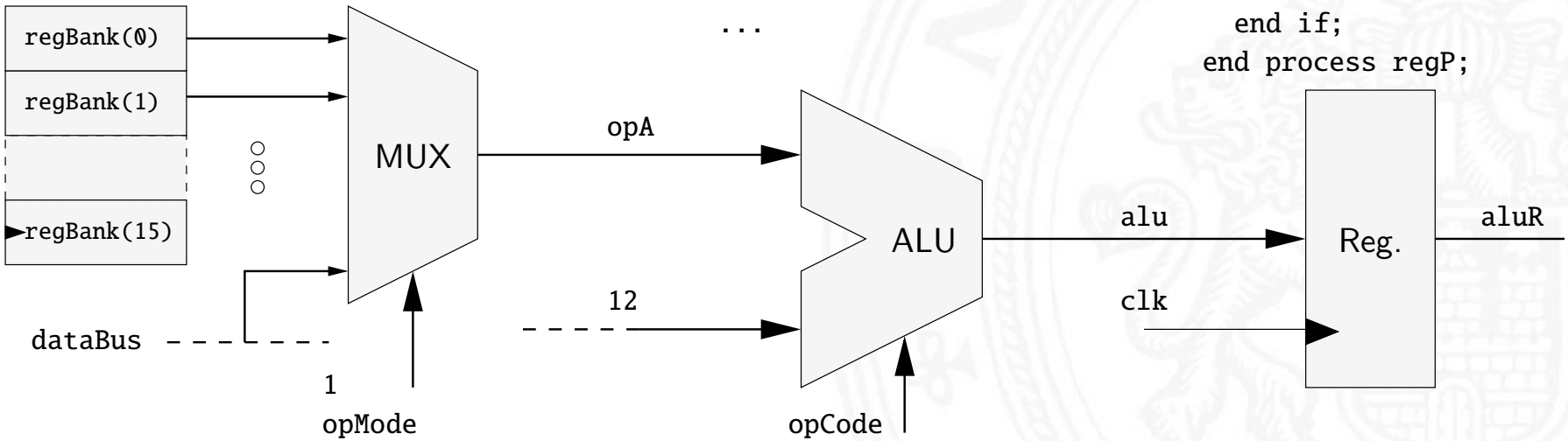
1. Simulationsereignis
2. Prozessaktivierung
3. Aktualisierung der Signaltreiber




```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB when opcAdd,
       opA  -  opB when opcSub,
       opA and opB when opcAnd,
       ...
```

```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation – Zyklus 2

Zeitschritt: $+\delta$

Simulationsereignisse

$+\delta$	opA	16
	alu	49
$+10\,\text{ns}$	clk	'1'
...		

Signaltreiber

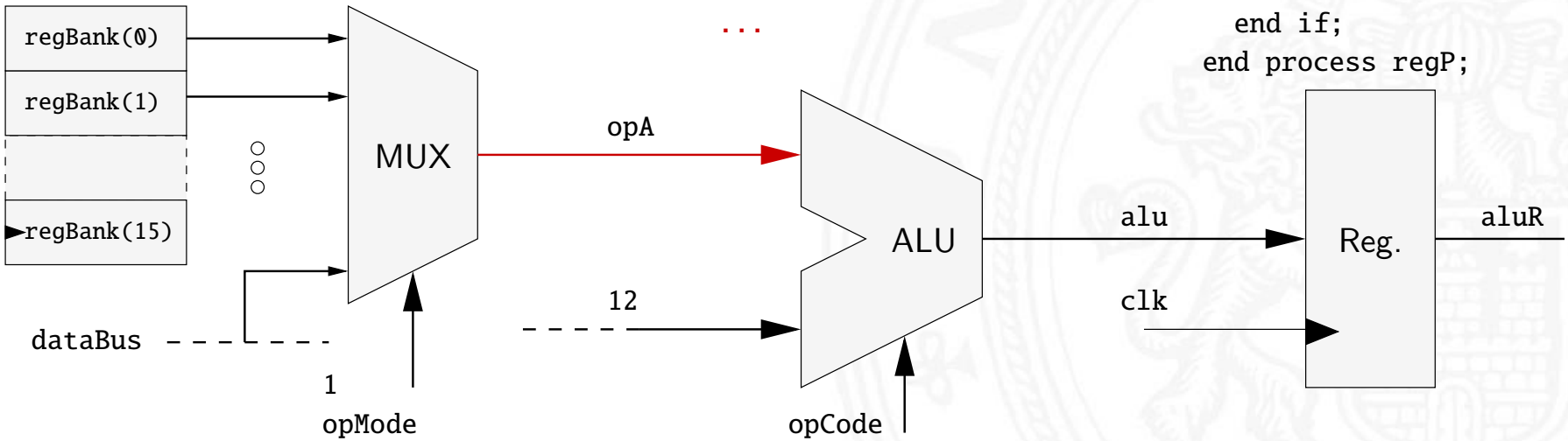
alu

NOW	$+\delta$
49	28

```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB when opcAdd,
       opA  -  opB when opcSub,
       opA and opB when opcAnd,
       ...
```

```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation – Zyklus 3

Zeitschritt: +10 ns

Simulationsereignisse

+10 ns clk '1'

...

Signaltreiber

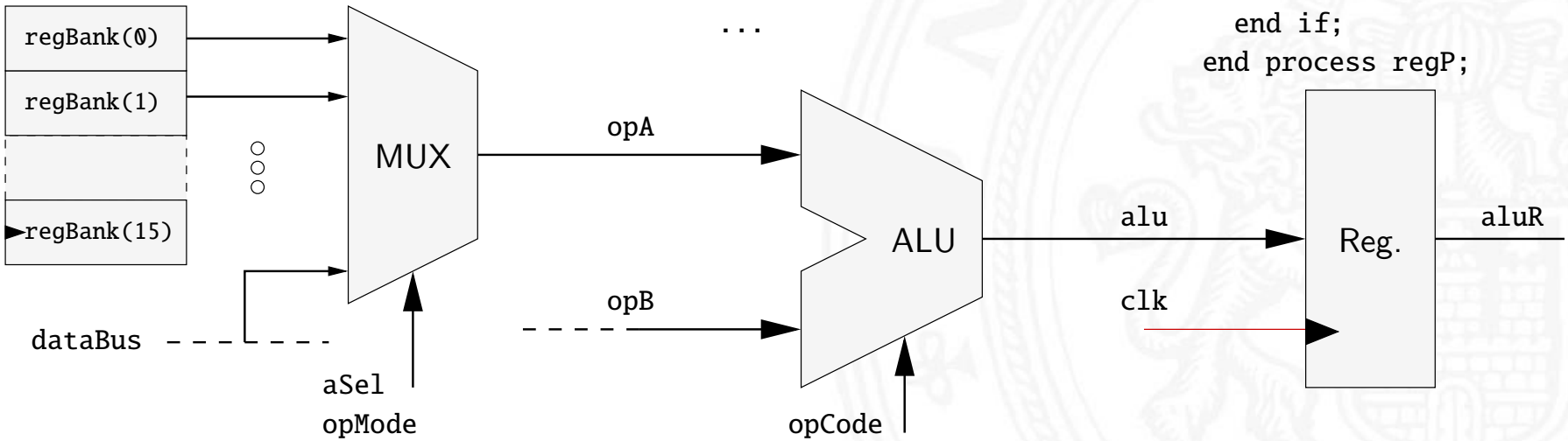
aluR

NOW	$+\delta$
25	28

```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB when opcAdd,
       opA  -  opB when opcSub,
       opA and opB when opcAnd,
       ...
```

```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



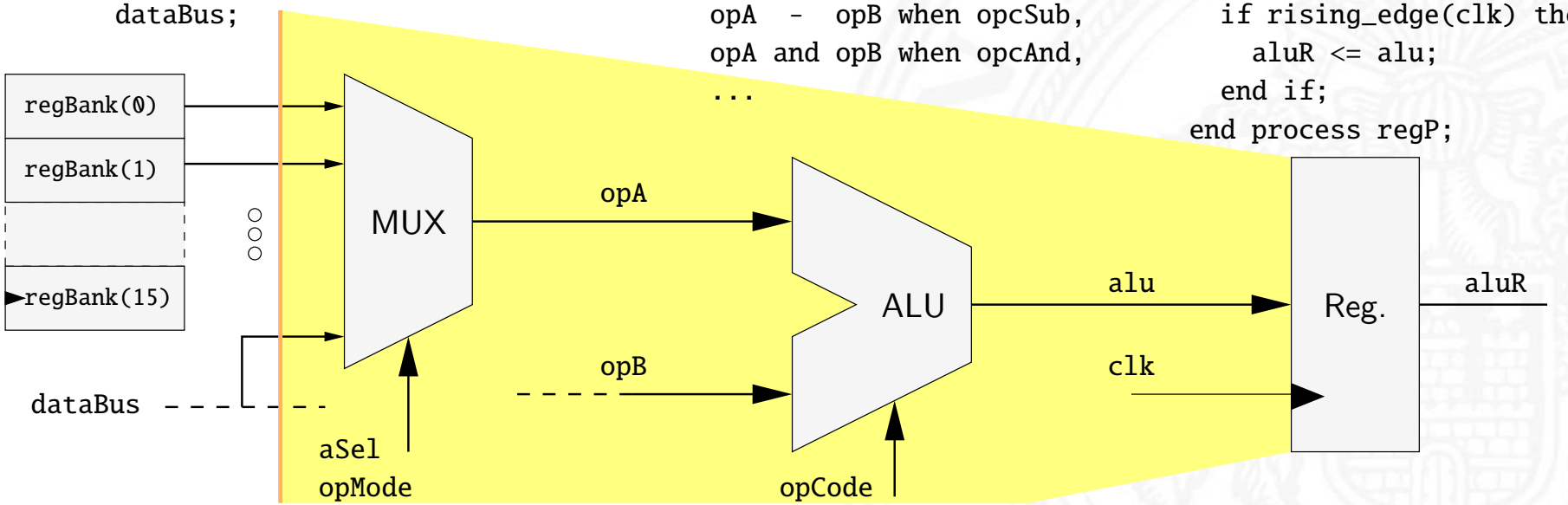
Zyklenbasierte Simulation

- Diskretes Zeitraster: Takt bei Register-Transfer Code
- In jedem Zyklus werden alle Beschreibungen simuliert
- Sequenzialisierung der Berechnung entsprechend den Datenabhängigkeiten

```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB when opcAdd,
       opA  -  opB when opcSub,
       opA and opB when opcAnd,
       ...
```

```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



VHDL – Simulation

Semantik der Simulation

- Kausalität
  1. Simulationsereignis — Simulationszyklus$_n$
  2. Aktivierung des konkurrenten Codes
  3. Signalzuweisungen in der Abarbeitung

  ---

  4. Erneute Signaländerung — Simulationszyklus$_{n+1}$
- Trennung der Zyklen

  $\Rightarrow$ Simulation ist *unabhängig von der sequenziellen Abarbeitungsreihenfolge* durch den Simulator

  $\Rightarrow$ auch bei *mehreren Events* in einem Zyklus, bzw. bei *mehrfachen Codeaktivierungen* pro Event

VHDL – Simulation (cont.)

Prozesse / Umgebungen von sequenziellem Code

- ständig aktiv ⇒ Endlosschleife

1. Sensitiv zu Signalen
   - Aktivierung, bei Ereignis eines Signals
   - Abarbeitung aller Anweisungen bis zum Prozessende

```
ALU_P: process (A, B, ADD_SUB) is
begin
  if ADD_SUB  then  X <= A + B;
              else  X <= A - B;
  end if;
end process ALU_P;
```

2. explizite `wait`-Anweisungen
   - Warten bis Bedingung erfüllt ist
   - Abarbeitung aller Anweisungen bis zum nächsten `wait`
   - Prozessende wird „umlaufen“ (Ende einer Schleife)

VHDL – Simulation (cont.)

Beispiel: Erzeuger / Verbraucher

```
...
signal C_READY, P_READY : boolean        -- Semaphore
signal CHANNEL          : ...            -- Kanal

PRODUCER_P: process is                   -- Erzeuger
begin
  P_READY   <= false;
  wait until C_READY;
  CHANNEL   <= ...                       -- generiert Werte
  P_READY   <= true;
  wait until not C_READY;
end process PRODUCER_P;

CONSUMER_P: process is                   -- Verbraucher
begin
  C_READY   <= true;
  wait until P_READY;
  C_READY   <= false;
       ... <= CHANNEL;                   -- verarbeitet Werte
  wait until not P_READY;
end process CONSUMER_P;
```

VHDL – Simulation (cont.)

Signalzuweisungen im sequenziellen Kontext

- sequenzieller Code wird nach der Aktivierung bis zum Prozessende / zum `wait` abgearbeitet
- Signalzuweisungen werden erst in folgenden Simulationszyklen wirksam, frühestens im nächsten δ-Zyklus

$\Rightarrow$ eigene Zuweisungen sind im sequenziellen Kontext des Prozesses nicht sichtbar

```
process ...
  ...
  if SWAP = '1' then   -- Werte tauschen
    B <= A;            -- B = 'altes' A
    A <= B;            -- A = 'altes' B
  end if;
```

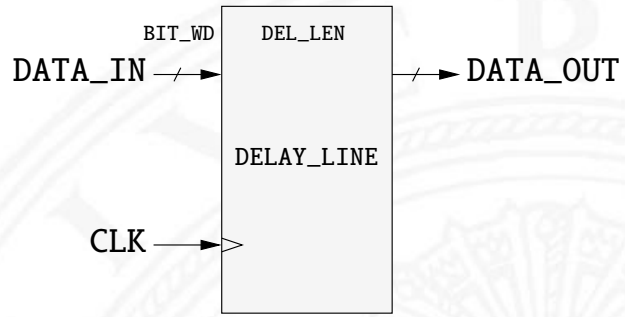
```
process ...
  ...
  NUM <= 5;            -- Zuweisung
  ...
  if NUM > 0 then      -- ggf. /= 5 !!!
    ...
```

Entwurfsspezifische Eigenschaften

- Strukturbeschreibungen / Hierarchie
  - Instanzen `component` `configuration`
  - Schnittstellen `entity`
  - Versionen und Alternativen (*exploring the design-space*)
    `architecture` `configuration`
- Management von Entwürfen
  - Bibliotheken `library`
  - Code-Reuse `package`
  - VHDL-Erweiterungen: Datentypen, Funktionen...
  - Gatterbibliotheken
  - spezifisch für EDA-Tools (**E**lectronic **D**esign **A**utomation)
  - eigene Erweiterungen, firmeninterne Standards...

VHDL – Entity

- Beschreibung der Schnittstelle „black-box“
  - mit Parametern `generic`
  - mit Ein- / Ausgängen `port`



parametrisierbare Verzögerungsleitung

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity DELAY_LINE is
generic(BIT_WD          : integer range 2 to 64 := 16;
        DEL_LEN         : integer range 2 to 16 := 16);
port (  CLK             : in  std_logic;
        DATA_IN         : in  signed(BIT_WD-1 downto 0);
        DATA_OUT        : out signed(BIT_WD-1 downto 0));
end entity DELAY_LINE;
```

VHDL – Architecture

- Implementation einer Entity
- mehrere Architekturen möglich ⇒ Alternativen
- Deklarationsteil: Typen, Signale, Unterprogramme, Komponenten, Konfigurationen
  Anweisungsteil: Prozesse, konkurrente Anweisungen, Instanzen

```
architecture BEHAVIOR of DELAY_LINE is
  type    DEL_ARRAY_TY is array (1 to DEL_LEN) of signed(BIT_WD-1 downto 0);
  signal  DEL_ARRAY     : DEL_ARRAY_TY;
begin
  DATA_OUT      <= DEL_ARRAY(DEL_LEN);
  REG_P: process (CLK) is
  begin
    if rising_edge(CLK) then
      DEL_ARRAY <= DATA_IN & DEL_ARRAY(1 to DEL_LEN-1);
    end if;
  end process REG_P;
end architecture BEHAVIOR;
```

VHDL – strukturell

- Hierarchie
  - funktionale Gliederung des Entwurfs
  - repräsentiert Abstraktion
- Instanziierung von Komponenten
  1. Komponentendeklaration `component`
     - im lokalen Kontext
     - in externen Packages
  2. Instanz im Code vervenden
     - im Anweisungsteil der Architecture
     - Mapping von Ein- und Ausgängen / Generic-Parametern
  3. Bindung: Komponente $\Leftrightarrow$ Entity+Architecture
     - lokal im Deklarationsteil `for ... use ...`
     - als eigene VHDL-Einheit `configuration`

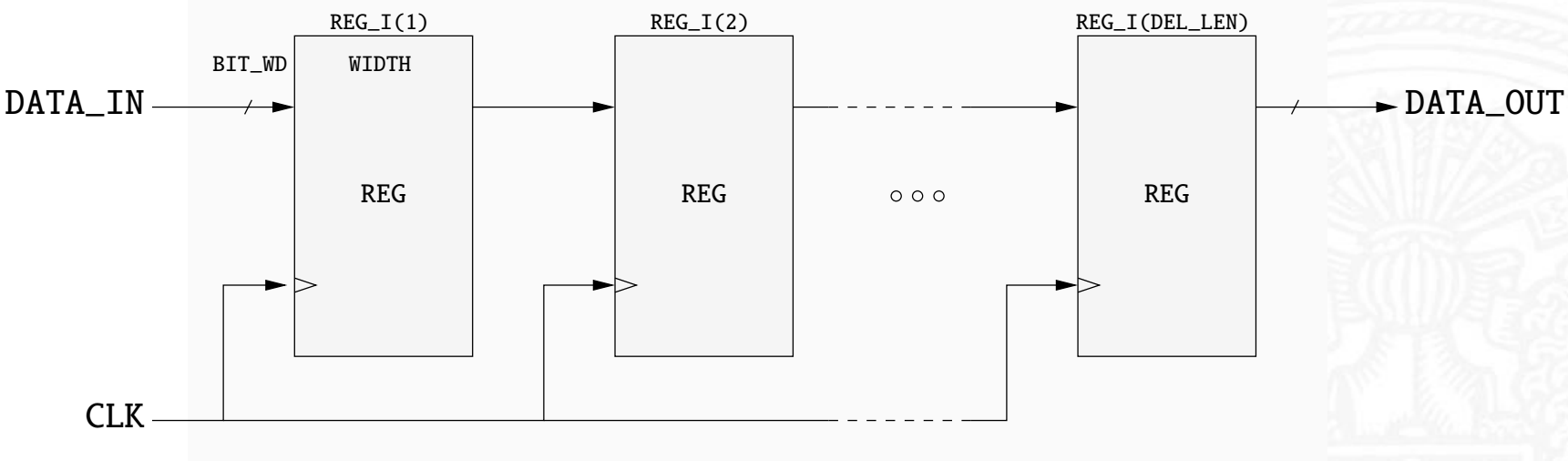
VHDL – strukturell (cont.)

- Komponente: lokale Zwischenstufe im Bindungsprozess
  - andere Bezeichner, Schnittstellen (Ports und Generics)
  - bei Bibliothekselementen wichtig
  - 2-stufige Abbildung
    1. Instanz in Architektur $\Leftrightarrow$ Komponente
    2. Komponente $\Leftrightarrow$ Entity+Architecture
  - „Default“-Konfiguration
    - gleiche Bezeichner und Deklaration
    - zuletzt (zeitlich) analysierte Architektur
- strukturierende Anweisungen
  - Gruppierung `block`
  - konkurrenten Code (Prozesse, Anweisungen, Instanzen...)
    - bedingt ausführen `if ... generate ...`
    - wiederholt ausführen `for ... generate ...`

VHDL – strukturell (cont.)

Beispiel als Strukturbeschreibung von Registern: `REG`

- Die Register sind als eigene VHDL-Entities / -Achitecturen woanders definiert



VHDL – strukturell (cont.)

```
architecture STRUCTURE of DELAY_LINE is
  component REG is                                  -- 1. Komponentendeklaration
  generic( WIDTH        : integer range 2 to 64);
  port   ( CLK          : in std_logic;
           D_IN         : in  signed(WIDTH-1 downto 0);
           D_OUT        : out signed(WIDTH-1 downto 0));
  end component REG;
  type    DEL_REG_TY is array (0 to DEL_LEN) of signed(BIT_WD-1 downto 0);
  signal  DEL_REG       : DEL_REG_TY;
begin
  DEL_REG(0)    <= DATA_IN;
  DATA_OUT      <= DEL_REG(DEL_LEN);
  GEN_I: for I in 1 to DEL_LEN generate
    REG_I: REG generic map (WIDTH => BIT_WD)     -- 2. Instanziierung
                 port map (CLK, DEL_REG(I-1), DEL_REG(I));
  end generate GEN_I;
end architecture STRUCTURE;
```

VHDL – strukturell (cont.)

Konfigurationen

1. Auswahl der Architektur durch eindeutigen Bezeichner

```
configuration DL_BEHAVIOR of DELAY_LINE is
for BEHAVIOR
end for;
end configuration DL_BEHAVIOR;
```

2. Bindung von Instanzen in der Hierarchie

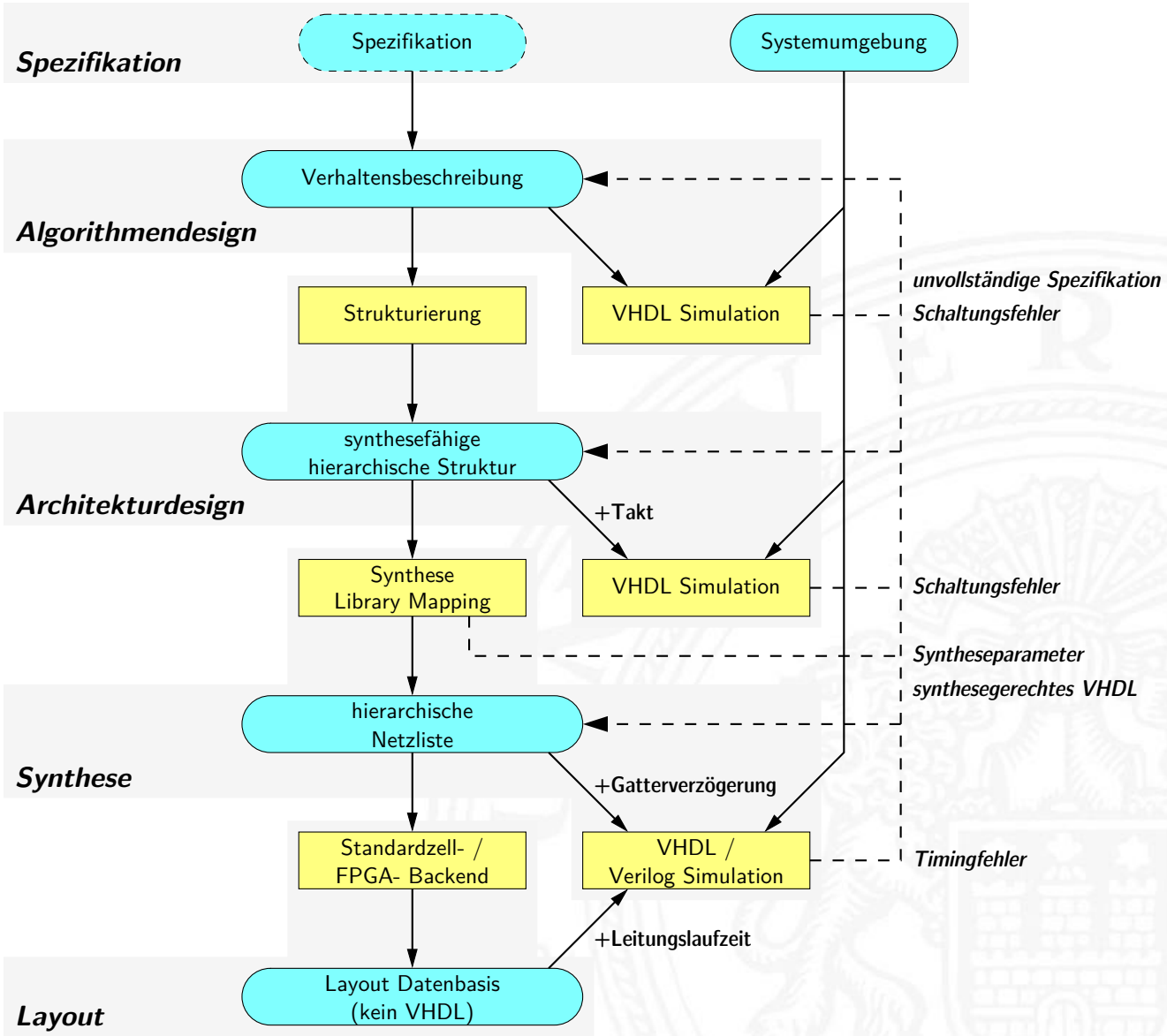
```
configuration DL_STRUCTURE of DELAY_LINE is
for STRUCTURE
  for GEN_I
    for all: REG use entity work.REG(BEHAVIOR);
    end for;
  end for;
end for;
end configuration DL_STRUCTURE;
```

VHDL – Entwurfsmethodik

- VHDL Abstraktion: von Algorithmen- bis zur Gatterebene
  $\Rightarrow$ eine Sprache als Ein- und Ausgabe der Synthese
- Synthese: ab der RT-Ebene
  - Abbildung von Register-Transfer Beschreibungen auf Gatternetzlisten
  - erzeugt neue Architektur, Entity bleibt

  High-Level Synthese
  - eingeschränkter „Suchraum“, feste Zielarchitektur
  - spezielle Anwendungsfälle
- Simulation
  - System-/Testumgebung als VHDL-Verhaltensmodell
  - Simulation der Netzliste durch Austausch der Architektur

VHDL-basierter Entwurfsablauf



VHDL – Entwurfsmethodik (cont.)

- Modellierung der Systemumgebung
- Simulation auf allen Abstraktionebenen

Algorithmenebene	Auswahl verschiedener Algorithmen keine Zeitmodelle
RT-Ebene	Datenabhängigkeiten: Ein-/Ausgabe (Protokolle) Taktbasierte Simulation
Gatterebene	+ Gatterverzögerungen
Layout	+ Leitungslaufzeiten

- Synthese ab der Register-Transfer Ebene

VHDL-AMS

VHDL-AMS – **A**nalog and **M**ixed **S**ignal

- Analoge VHDL Erweiterung IEC 61691-6
- VHDL Obermenge: digitale Systeme werden wie in „normalem“ VHDL modelliert
- „mixed-signal“: gemischte analog-digitale Systeme
  „multi-domain“: elektrische + nicht-elektrische Systeme
- analoge Modellierung: Differentialgleichungssysteme deren freie Variablen durch einen Lösungsalgorithmus („analog Solver“) bestimmt werden
- analoge Erweiterung der Simutionssemantik
  - Anpassung der Paradigmen (diskret $\leftrightarrow$ kontinuierlich) für das Zeit- und Wertemodell
  - Wann wird der Solver aufgerufen?

VHDL-AMS (cont.)

Anwendungen

- analoge Systeme
- mikromechanische Systeme
- mechanische Komponenten
- Modellierung der Schnittstellen zu mechanischen Komponenten
- Beispiel: Ansteuerung eines Motors, Simulation von
  - analogem Treiber
  - elektomagnetischem Verhalten des Motors
  - Massenträgheit und Last
  - ...

VHDL-AMS (cont.)

Erweiterungen

- Datentypen
  - `nature`: zur Modellierung verschiedener Domänen

```
subtype VOLTAGE is real tolerance "TOL_VOLTAGE";
subtype CURRENT is real tolerance "TOL_CURRENT";

nature ELECTRICAL is    VOLTAGE across
                        CURRENT through
                        GROUND  reference;
```

- Objekte/Ports
  - `terminal`: Referenzpunkt
    Knoten eines elektrischen Netzes, Ort im mechanischen System
  - `quantity`: Werte die (zeitkontinuierlich) berechnet werden
    Variablen des Gleichungssystems

VHDL-AMS (cont.)

- Anweisungen
  - simultane Anweisung: Beschreibung einer Gleichung, partiell definierte Systeme (if, case-Fälle) sind möglich
  - `break`: steuert die Zusammenstellung der Gleichungssysteme des „analog Solvers“ beispielsweise bei Diskontinuitäten
  - `procedural`: analoges Äquivalent zum `process`

Beispiele

- Diode: charakteristische Gleichungen

$$i_d = is \cdot (e^{(v_d - rs \cdot i_d)/n \cdot vt} - 1)$$

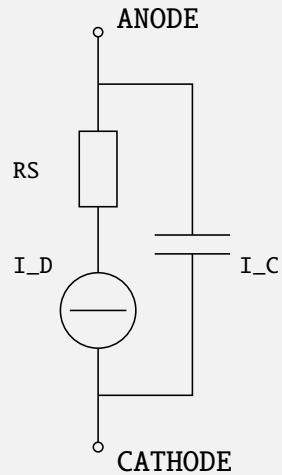
$$i_c = \frac{d}{dt}(tt \cdot i_d - 2 \cdot cj \cdot \sqrt{vj^2 - vj \cdot v_d})$$

VHDL-AMS (cont.)

```
library ieee, AMS_LIB;
use ieee.math_real.all;
use AMS_LIB.ELEC_ENV.all;

entity DIODE is
generic (   ISS                     : real  := 1.0e-14;
            N                       : real  := 1.0;
            TT, CJ0, VJ, RS         : real  := 0.0);
port    (   terminal ANODE, CATHODE : ELECTRICAL);
end entity DIODE;

architecture LEVEL0 of DIODE is
  quantity  V_D across  I_D, I_C through  ANODE to CATHODE;
  quantity  Q_C : CHARGE;
  constant  VT  : real    := 0.0258;
begin
  I_D == ISS * (exp((V_D-RS*I_D)/(N*VT)) - 1.0);
  Q_C == (TT * I_D) - (2.0 * CJ0 * sqrt(VJ**2 - VJ*V_D));
  I_C == Q_C'dot;
end architecture LEVEL0;
```



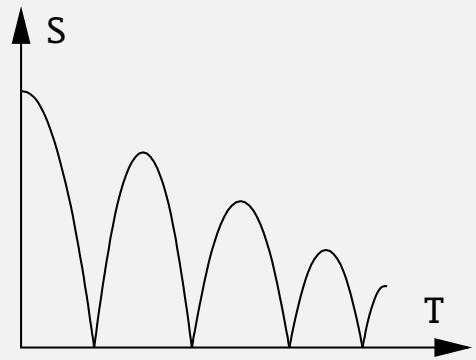
VHDL-AMS (cont.)

- hüpfender Ball

```
library ieee, AMS_LIB;
use ieee.math_real.all;
use AMS_LIB.MECH_ENV.all;

entity BOUNCER is
generic (   S_INI    : DISPLACEMENT  := 10.0;
            V_INI    : VELOCITY      :=  0.0);
end entity BOUNCER;

architecture BALL of BOUNCER is
  quantity  S       : DISPLACEMENT  := S_INI;
  quantity  V       : VELOCITY      := V_INI;
  constant  G       : REAL          := 9.81;
  constant  AIR_RES : REAL          := 0.1;
begin
  break V => -V     when not S'above(0.0);
  S'dot == V;
  if V > 0.0  use   V'dot == -G - V**2 * AIR_RES;
              else  V'dot == -G + V**2 * AIR_RES;
  end use;
end architecture BALL;
```



Verilog / SystemVerilog

- Hardwarebeschreibungssprache
  - Verhaltensbeschreibung (auf Gatter- und RT-Ebene)
    SystemVerilog auch auf höheren Ebenen
  - Strukturbeschreibung, Hierarchien
- Entwicklung
  - 1985 ursprünglich proprietäre Sprache / Simulator
  - 1995 IEEE/IEC Standard, regelmäßige Überarbeitungen IEEE 1364
  - 2005 aktuelle Erweiterung: SystemVerilog IEEE 1800

Verilog / SystemVerilog (cont.)

Links

- http://www.systemverilog.org
- http://www.verilog.org
- http://en.wikipedia.org/wiki/SystemVerilog
- http://en.wikipedia.org/wiki/Verilog
- http://www.asic-world.com/systemverilog
- http://www.asic-world.com/verilog
- http://electrosofts.com/systemverilog
- http://electrosofts.com/verilog

Verilog / SystemVerilog (cont.)

Vorteile

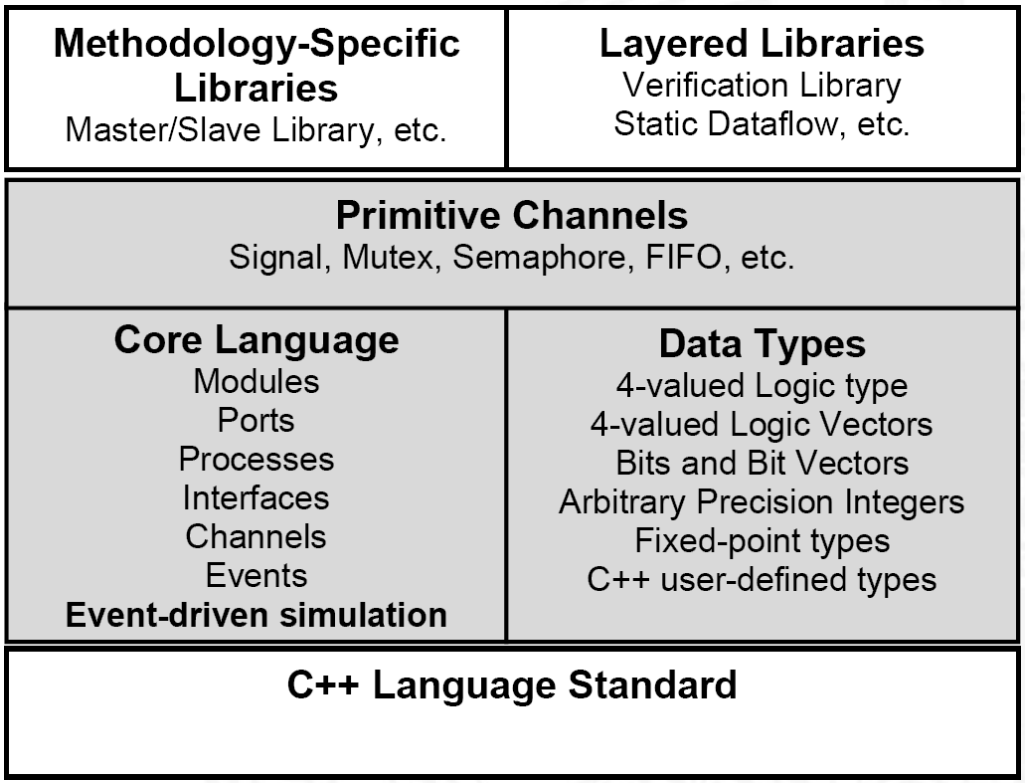
- sehr kompakter Code, wird gerne als Netzlistenformat genutzt – häufig geäußerte Kritik an VHDL: „deklarativer Overhead“ –
- Simulation
  - sehr schnell: älter als VHDL → Algorithmen besser optimiert
  - „golden Simulation“: finale Simulation(en) der Netzliste mit extrahierten Leitungslaufzeiten vor der Fertigung

(System)Verilog oder VHDL

- kein Unterschied bei den Modellierungsmöglichkeiten
- oft werden Komponenten beider HDLs gemeinsam eingesetzt
- EDA-Werkzeuge: Synthese, Simulation
  - ein internes Datenformat
  - unterschiedliche „frond-Ends“

SystemC

- C++ basiert: Klassenbibliotheken mit
  - hardware-nahen Datentypen
  - Simulatorkern



SystemC (cont.)

- Semantische Erweiterungen für den Hardwareentwurf
  - Konkurrente Modelle: Prozesse
  - Zeitbegriff: Clocks
  - Reaktivität (Ereignisse in VHDL): Waiting, Watching...
  - Kommunikationsmechanismen: Signale, Protokolle
  - Hierarchie: Module, Ports...
  - ...
- Entwicklung
  - 2000 OSCI – **O**pen **S**ystem**C** **I**nitiative
  - 2005 IEEE/IEC Standard IEC 61691-7
  - Schrittweise Entwicklung

SystemC (cont.)

Links

- `http://www.systemc.org`
- `http://en.wikipedia.org/wiki/SystemC`
- `http://www.asic-world.com/systemc`
- `http://www-ti.informatik.uni-tuebingen.de/~systemc`

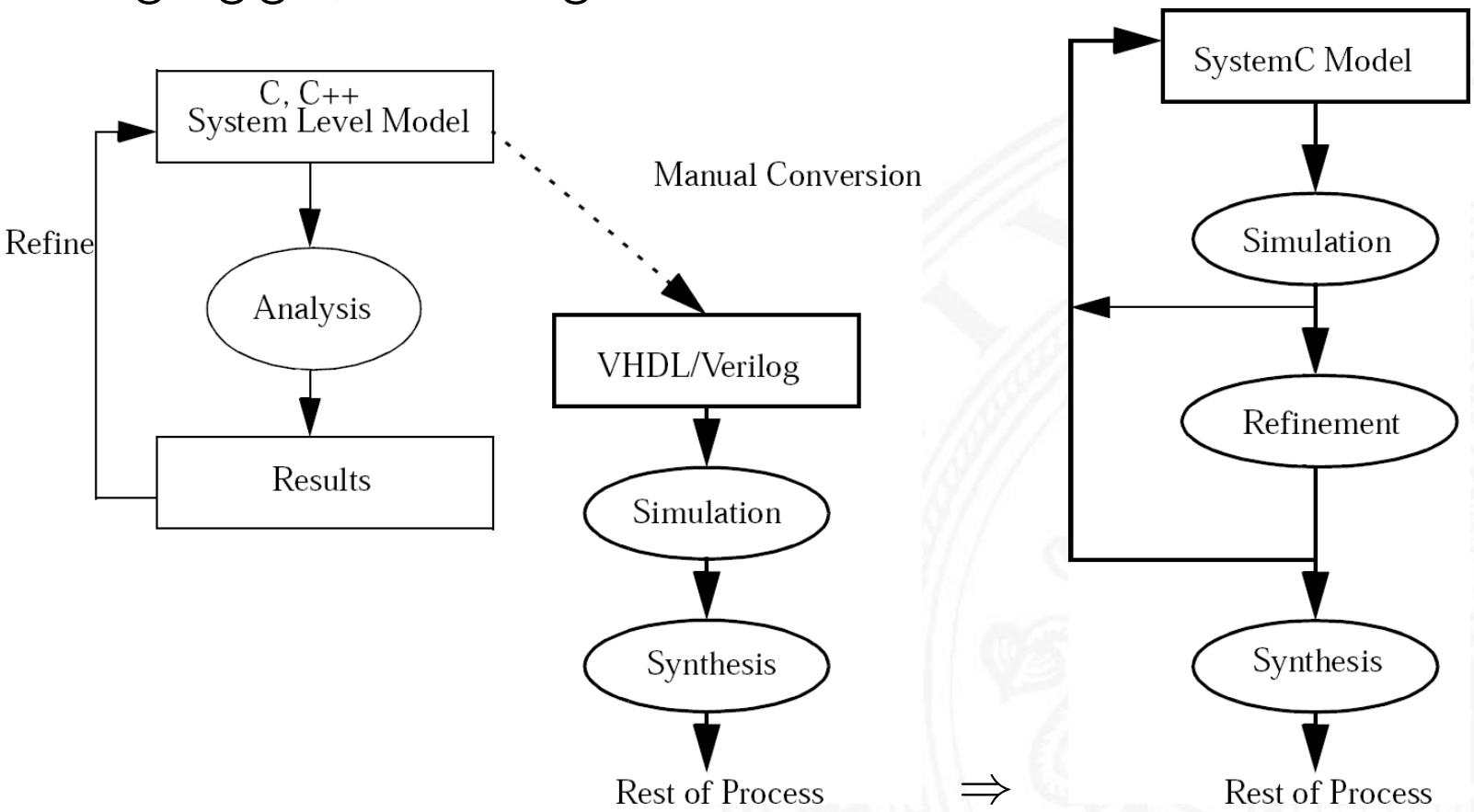
Idee/Vorteile

- Hard- und Software gemeinsam beschreiben: *Hardware-Software Co-Design*

$\Rightarrow$ höhere Abstraktionsebenen

SystemC (cont.)

⇒ durchgängiger, Werkzeug-unterstützter Entwurfsablauf



SystemC (cont.)

- C/C++ Infrastruktur nutzen: Compiler, Debugger...
- Know-How nutzen:
  jeder Softwareentwerfer kann damit auch „Hardware machen“
- Einfache Integration von eigenem Code und Erweiterungen

Praxis

+ Ersatz für (proprietäre) High-Level Simulation
– macht den C++ Programmierer nicht zum Hardwaredesigner
– noch mehr Deklarationsoverhead als bei VHDL
– Unterstützung duch EDA-Werkzeuge

Kunst des Hardwareentwurfs

Guten, synthesefähigen Code zu erstellen ...gilt für alle HDLs

Beispiele

D-Flipflop mit asynchronem Reset

- VHDL: `dff.vhd`

```
library ieee;
use ieee.std_logic_1164.all;

entity DFF is
port (  CLOCK   : in  std_logic;
        RESET   : in  std_logic;
        DIN     : in  std_logic;
        DOUT    : out std_logic);
end entity DFF;

architecture BEHAV of DFF is
begin
  DFF_P: process (RESET, CLOCK) is
  begin
    if RESET = '1' then
        DOUT <= '0';
    elsif rising_edge(CLOCK) then
        DOUT <= DIN;
    end if;
  end process DFF_P;
end architecture BEHAV;
```

Beispiele (cont.)

- Verilog: `dff.v`

```
module dff (clock, reset, din, dout);
input clock, reset, din;
output dout;

reg dout;

  always @(posedge clock or reset)
  begin
    if (reset)
      dout = 1'b0;
    else
      dout = din;
    end
endmodule
```

Beispiele (cont.)

- SystemC: `dff.h`

```
#include "systemc.h"

SC_MODULE(dff)
{ sc_in<bool>   clock;
  sc_in<bool>   reset;
  sc_in<bool>   din;
  sc_out<bool>  dout;

  void do_ff()
  { if (reset)
      dout = false;
    else if (clock.event())
      dout = din;
  };

  SC_CTOR(dff)
  { SC_METHOD(do_ff);
    sensitive(reset);
    sensitive_pos(clock);
  }
};
```

Beispiele (cont.)

8-bit Zähler, synchron rücksetz- und ladbar

- VHDL: `counter.vhd`

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity COUNTER is
port (  CLOCK   : in  std_logic;
        LOAD    : in  std_logic;
        CLEAR   : in  std_logic;
        DIN     : in  unsigned (7 downto 0);
        DOUT    : out unsigned (7 downto 0));
end entity COUNTER;
```

Beispiele (cont.)

```
architecture BEHAV of COUNTER is
  signal CNTVAL : unsigned (7 downto 0);
begin
  CNT_P: process (CLOCK) is
  begin
    if rising_edge(CLOCK) then
      if CLEAR = '1' then
        CNTVAL <= (others=>'0');
      elsif load = '1' then
        CNTVAL <= DIN;
      else
        CNTVAL <= CNTVAL + 1;
      end if;
    end if;
  end process CNT_P;

  DOUT <= CNTVAL;
end architecture BEHAV;
```

Beispiele (cont.)

- Verilog: `counter.v`

```
module counter(clock, load, clear, din, dout);
input           clock, load, clear;
input  [0:7]    din;
output [0:7]    dout;

wire [0:7]      dout;
reg  [0:7]      cntval;

  assign dout = cntval;

  always @(posedge clock)
  begin
    if (clear)
      cntval = 0;
    else if (load)
      cntval = din;
    else
      cntval++;
    end
endmodule
```

Beispiele (cont.)

- SystemC: `counter.h counter.cc`

```
#include "systemc.h"

SC_MODULE(counter)
{ sc_in<bool>           clock;
  sc_in<bool>           load;
  sc_in<bool>           clear;
  sc_in<sc_uint<8> >    din;
  sc_out<sc_uint<8> >   dout;

  int cntval;
  void onetwothree();

  SC_CTOR(counter)
  { SC_METHOD(onetwothree);
    sensitive_pos (clock);
  }
};
```

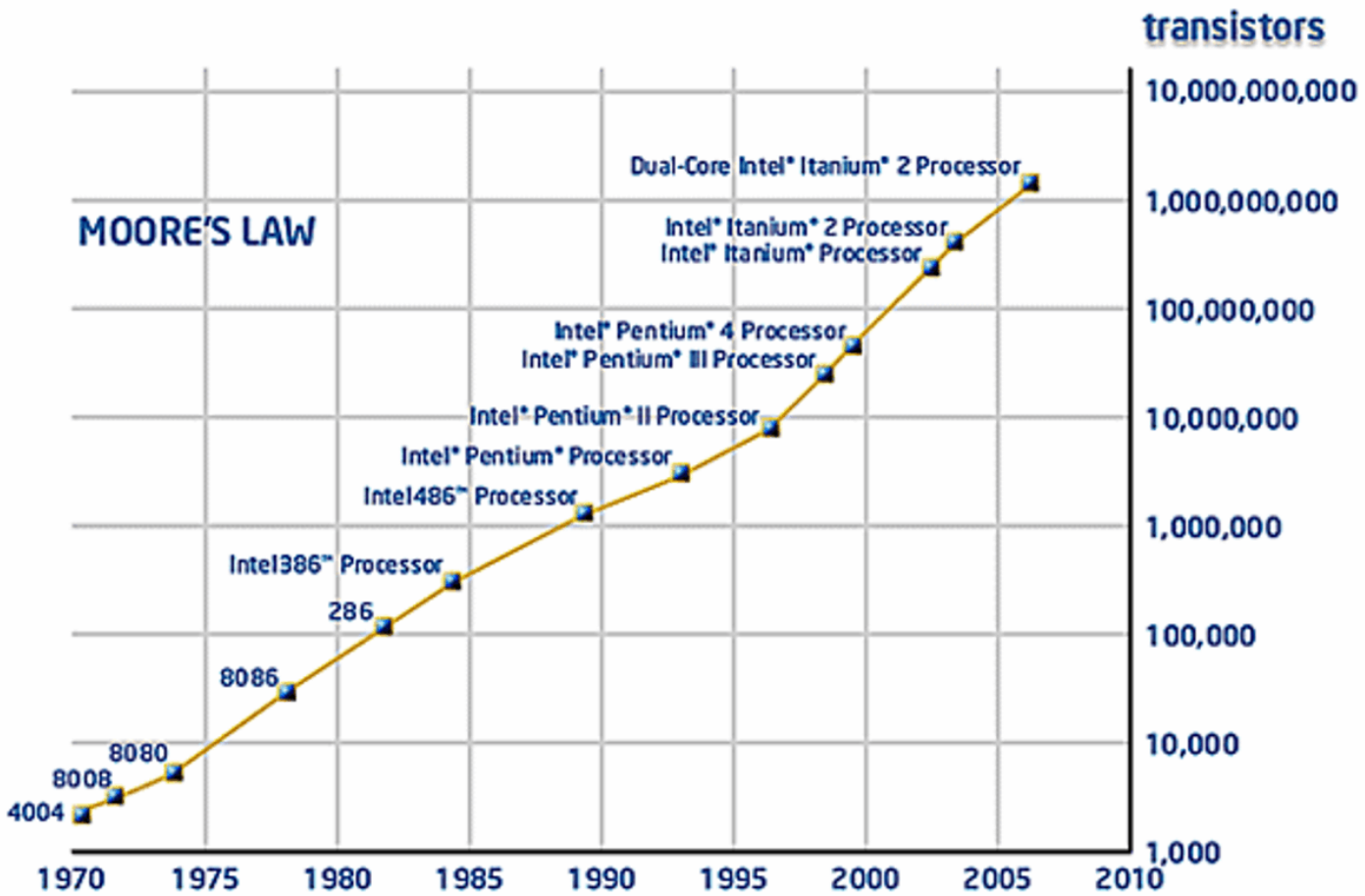
Beispiele (cont.)

```
#include "counter.h"

void counter::onetwothree()
{ if (clear)
    cntval = 0;
  else if (load)
    cntval = din.read();        // read for type conversion from input port
  else
    cntval++;
  }
  dout = cntval;
}
```

Motivation

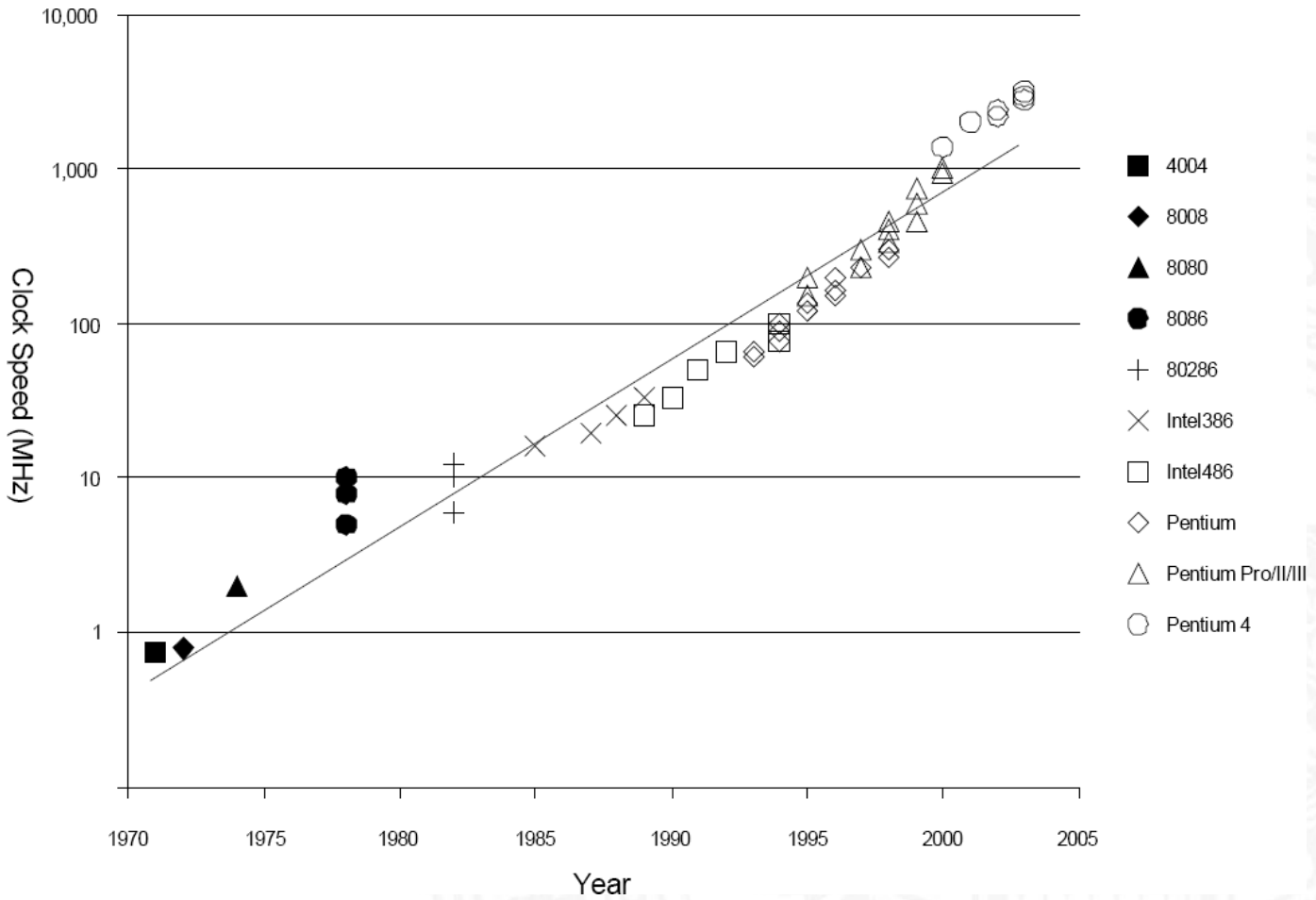
Komplexität



Intel

Motivation (cont.)

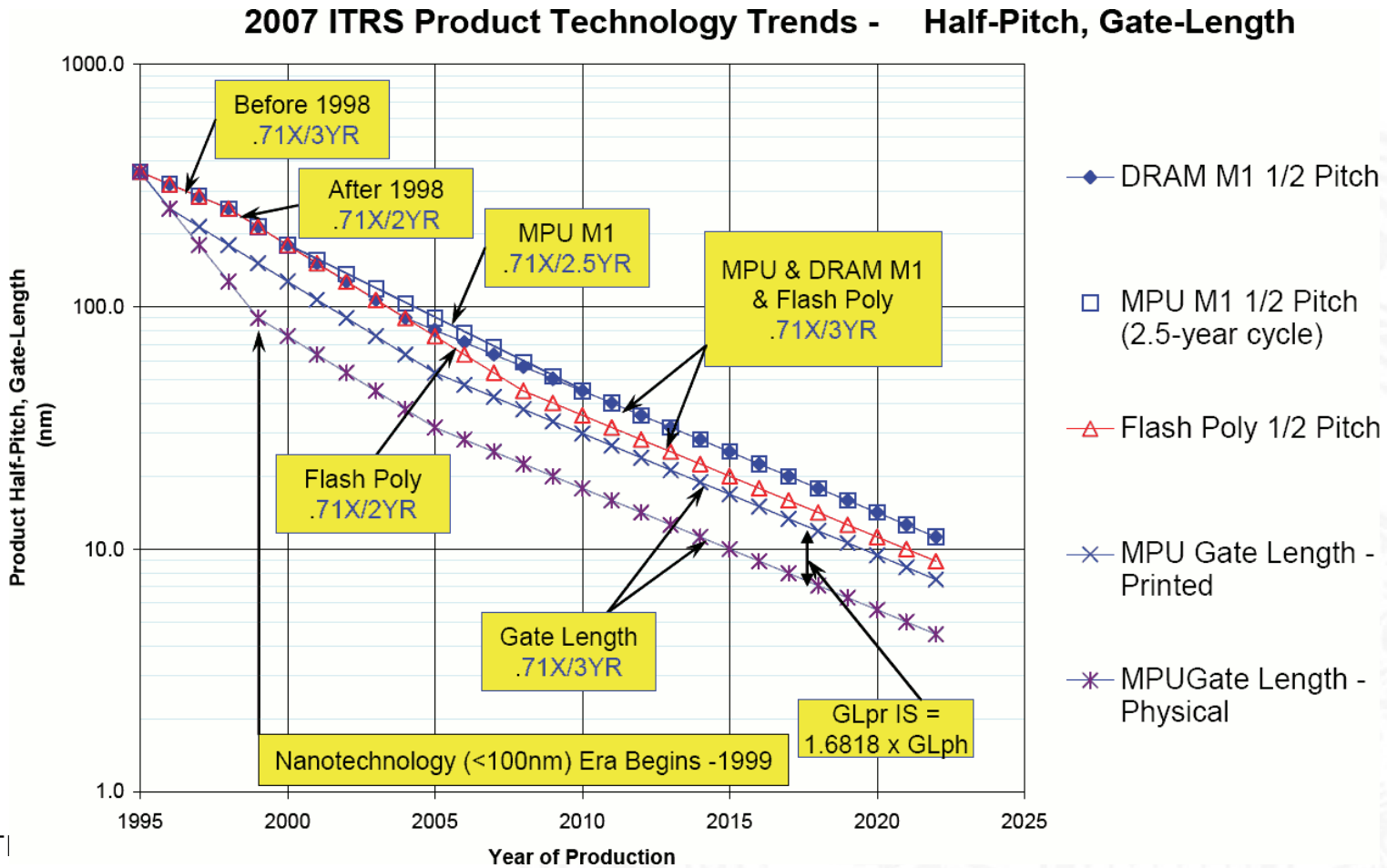
Taktfrequenz



Intel

Motivation (cont.)

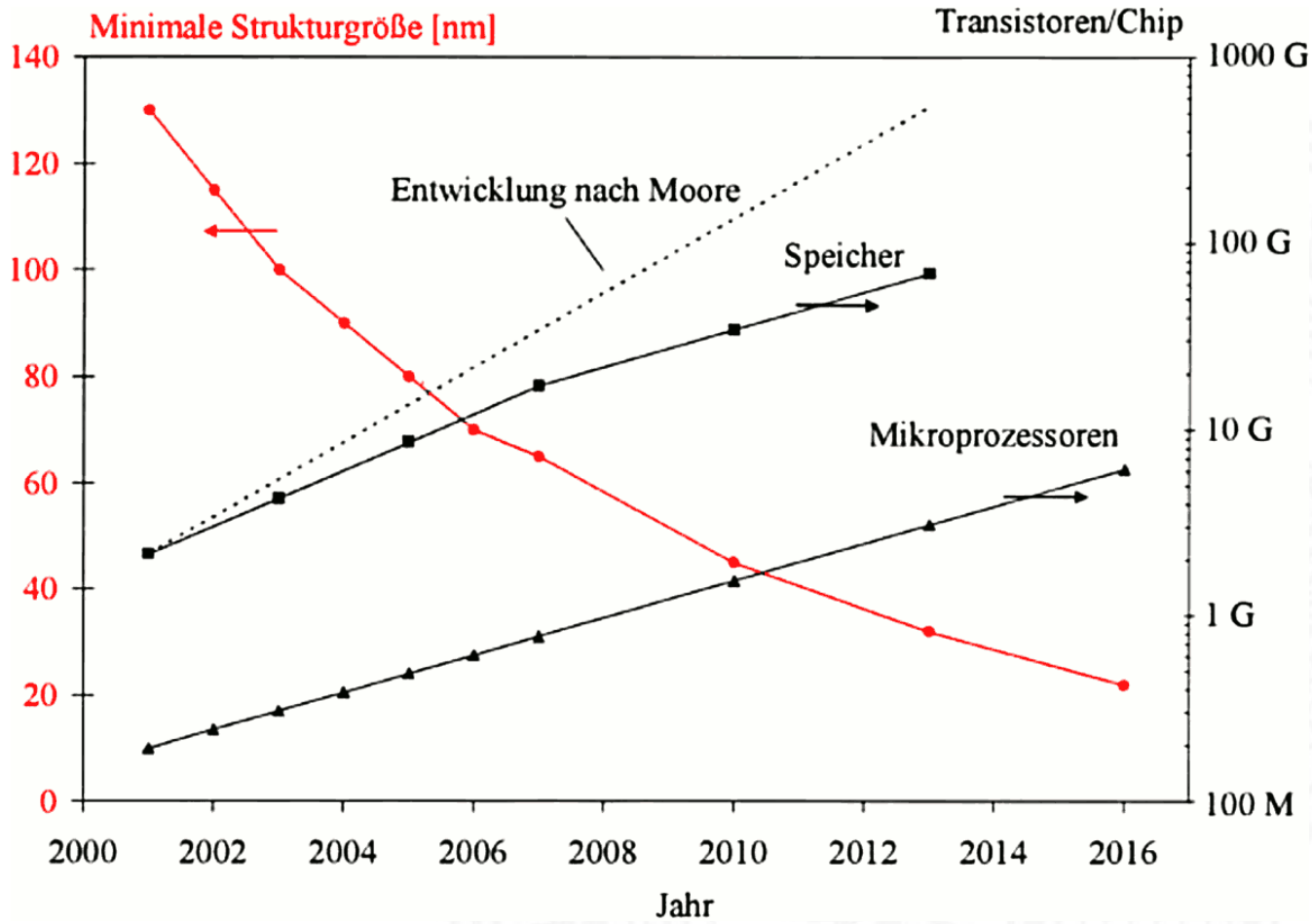
Strukturen



[IT

Motivation (cont.)

Strukturen



Motivation (cont.)

Treibende Faktoren

- Entwicklung der Technologie
- Anwendungen und Einsatzbereiche
- EDA-Entwurfsmethoden und -Werkzeuge

Frage

Wird Moore's Law auch in Zukunft noch gültig sein?

Links

- `http://www.intel.com/technology/mooreslaw`
- `http://www.itrs.net/reports.html`
  International Technology Roadmap for Semiconductors

ITRS Zukunftsprognosen

Chipgrößen

Table 1i High-Performance MPU and ASIC Product Generations and Chip Size Model—Near-term Years

Year of Production	*2007*	*2008*	*2009*	*2010*	*2011*	*2012*	*2013*	*2014*	*2015*
DRAM ½ Pitch (nm) (contacted)	*65*	*57*	*50*	*45*	*40*	*36*	*32*	*28*	*25*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*68*	*59*	*52*	*45*	*40*	*36*	*32*	*28*	*25*
MPU Physical Gate Length (nm)	*25*	*23*	*20*	*18*	*16*	*14*	*13*	*11*	*10*
***Logic** (Low-volume Microprocessor) High-performance ‡*									
Generation at Introduction	**p10h**	**p10h**	**p13h**	**p13h**	**p13h**	**p16h**	**p16h**	**p16h**	**p19h**
Functions per chip at introduction (million transistors)	**2212**	**2212**	**4424**	**4424**	**4424**	**8848**	**8848**	**8848**	**17696**
Chip size at introduction (mm^2)	**620**	**492**	**391**	**620**	**492**	**391**	**620**	**492**	**391**
*Generation at production ***	**p07h**	**p07h**	**p07h**	**p10h**	**p10h**	**p10h**	**p13h**	**p13h**	**p13h**
Functions per chip at production (million transistors)	**1106**	**1106**	**1106**	**2212**	**2212**	**2212**	**4424**	**4424**	**4424**
Chip size at production (mm^2) §§	**310**	**246**	**195**	**310**	**246**	**195**	**310**	**246**	**195**
High-performance MPU Mtransistors/cm^2 at introduction and production (including on-chip SRAM) ‡	**357**	**449**	**566**	**714**	**899**	**1133**	**1427**	**1798**	**2265**
ASIC									
ASIC usable Mtransistors/cm^2 (auto layout)	**357**	**449**	**566**	**714**	**899**	**1133**	**1427**	**1798**	**2265**
ASIC max chip size at production (mm^2) (maximum lithographic field size)	***858***	***858***	***858***	***858***	***858***	***858***	***858***	***858***	***858***
ASIC maximum functions per chip at production (Mtransistors/chip) (fit in maximum lithographic field size)	**3,061**	**3,857**	**4,859**	**6,122**	**7,713**	**9,718**	**12,244**	**15,427**	**19,436**

ITRS Zukunftsprognosen (cont.)

Chipgrößen

Table 1j High-Performance MPU and ASIC Product Generations and Chip Size Model—Long-term Years

Year of Production	*2016*	*2017*	*2018*	*2019*	*2020*	*2021*	*2022*
DRAM ½ Pitch (nm) (contacted)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
MPU Physical Gate Length (nm)	*9*	*8*	*7*	*6.3*	*5.6*	*5.0*	*4.5*
***Logic** (Low-volume Microprocessor) High-performance ‡*							
Generation at Introduction	**p19h**	**p19h**	**p22h**	**p22h**	**p22h**	**p25h**	**p25h**
Functions per chip at introduction (million transistors)	**17696**	**17696**	**35391**	**35391**	**35391**	**70782**	**70782**
Chip size at introduction (mm^2)	**620**	**492**	**391**	**620**	**492**	**391**	**620**
*Generation at production ***	**p16h**	**p16h**	**p16h**	**p19h**	**p19h**	**p19h**	**p22h**
Functions per chip at production (million transistors)	**8848**	**8848**	**8848**	**17696**	**17696**	**17696**	**35391**
Chip size at production (mm^2) §§	**310**	**246**	**195**	**310**	**246**	**195**	**310**
High-performance MPU Mtransistors/cm^2 at introduction and production (including on-chip SRAM) ‡	**2854**	**3596**	**4531**	**5708**	**7192**	**9061**	**11416**
ASIC							
ASIC usable Mtransistors/cm^2 (auto layout)	**2854**	**3596**	**4531**	**5708**	**7192**	**9061**	**11416**
ASIC max chip size at production (mm^2) (maximum lithographic field size)	**858**	**858**	**858**	**858**	**858**	**858**	**1716**
ASIC maximum functions per chip at production (Mtransistors/chip) (fit in maximum lithographic field size)	**24,488**	**30,853**	**38,873**	**48,977**	**61,707**	**77,746**	**195,906**

ITRS Zukunftsprognosen

Wafergrößen

Table 2a Lithographic-Field and Wafer-Size Trends—Near-term Years

Year of Production	*2007*	*2008*	*2009*	*2010*	*2011*	*2012*	*2013*	*2014*	*2015*
DRAM ½ Pitch (nm) (contacted)	*65*	*57*	*50*	*45*	*40*	*36*	*32*	*28*	*25*
Flash ½ Pitch (nm) (un-contacted Poly)(f)	*54*	*45*	*40*	*36*	*32*	*28*	*25*	*22*	*20*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*68*	*59*	*52*	*45*	*40*	*36*	*32*	*28*	*25*
MPU Physical Gate Length (nm)	*25*	*23*	*20*	*18*	*16*	*14*	*13*	*11*	*10*
Lithography Field Size									
Maximum Lithography Field Size—area (mm^2)	**858**	**858**	**858**	**858**	**858**	**858**	**858**	**858**	**858**
Maximum Lithography Field Size—length (mm)	**33**	**33**	**33**	**33**	**33**	**33**	**33**	**33**	**33**
Maximum Lithography Field Size—width (mm)	**26**	**26**	**26**	**26**	**26**	**26**	**26**	**26**	**26**
***Maximum Substrate Diameter** (mm)—High-volume Production (>20K wafer starts per month)*									
Bulk or epitaxial or SOI wafer	**300**	**300**	**300**	**300**	**300**	**300 or 450**	**300 or 450**	**300 or 450**	**300 or 450**

ITRS Zukunftsprognosen (cont.)

Wafergrößen

Table 2b Lithographic-Field and Wafer Size Trends—Long-term Years

Year of Production	*2016*	*2017*	*2018*	*2019*	*2020*	*2021*	*2022*
DRAM ½ Pitch (nm) (contacted)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
Flash ½ Pitch (nm) (un-contacted Poly)(f)	*18*	*16*	*14*	*13*	*11*	*10*	*9*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
MPU Physical Gate Length (nm)	*9*	*8*	*7*	*6*	*6*	*5*	*4*
Lithography Field Size							
Maximum Lithography Field Size—area (mm^2)	**858**	**858**	**858**	**858**	**858**	**858**	**858**
Maximum Lithography Field Size—length (mm)	**33**	**33**	**33**	**33**	**33**	**33**	**33**
Maximum Lithography Field Size—width (mm)	**26**	**26**	**26**	**26**	**26**	**26**	**26**
***Maximum Substrate Diameter** (mm)—High-volume Production (>20K wafer starts per month)*							
Bulk or epitaxial or SOI wafer	**450**	**450**	**450**	**450**	**450**	**450**	**450**

ITRS Zukunftsprognosen

Padzellen und Pins

Table 3a Performance of Packaged Chips: Number of Pads and Pins—Near-term Years

Year of Production	*2007*	*2008*	*2009*	*2010*	*2011*	*2012*	*2013*	*2014*	*2015*
DRAM ½ Pitch (nm) (contacted)	*65*	*57*	*50*	*45*	*40*	*36*	*32*	*28*	*25*
Flash ½ Pitch (nm) (un-contacted Poly)(f)	*54*	*45*	*40*	*36*	*32*	*28*	*25*	*22*	*20*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*68*	*59*	*52*	*45*	*40*	*36*	*32*	*28*	*25*
MPU Physical Gate Length (nm)	*25*	*23*	*20*	*18*	*16*	*14*	*13*	*11*	*10*
***Number of Chip I/Os** (Number of Total Chip Pads)—Maximum*									
Total pads—MPU unchanged	**3,072**	**3,072**	**3,072**	**3,072**	**3,072**	**3,072**	**3,072**	**3,072**	**3,072**
Signal I/O—MPU (% of total pads)	**33.3%**	**33.3%**	**33.3%**	**33.3%**	**33.3%**	**33.3%**	**33.3%**	**33.3%**	**33.3%**
Power and ground pads—MPU (% of total pads)	**66.7%**	**66.7%**	**66.7%**	**66.7%**	**66.7%**	**66.7%**	**66.7%**	**66.7%**	**66.7%**
IS Total pads—ASIC High Performance unchanged	**4,400**	**4,400**	**4,600**	**4,800**	**4,800**	**5,000**	**5,400**	**5,400**	**5,600**
Signal I/O pads— ASIC high-performance (% of total pads)	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**
Power and ground pads— ASIC high-performance (% of total pads)	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**
***Number of Total Package Pins**—Maximum [1]*									
Microprocessor/controller, cost-performance	**600–2140**	**600–2400**	**660–2801**	**660–2783**	**720–3061**	**720–3367**	**800–3704**	**800–4075**	**880–4482**
Microprocessor/controller, high-performance	**4000**	**4400**	**4620**	**4851**	**5094**	**5348**	**5616**	**5896**	**6191**
ASIC (high-performance)	**4000**	**4400**	**4620**	**4851**	**5094**	**5348**	**5616**	**5896**	**6191**

ITRS Zukunftsprognosen (cont.)

Padzellen und Pins

Table 3b Performance of Packaged Chips: Number of Pads and Pins—Long-term Years

Year of Production	*2016*	*2017*	*2018*	*2019*	*2020*	*2021*	*2022*
DRAM ½ Pitch (nm) (contacted)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
Flash ½ Pitch (nm) (un-contacted Poly)(f)	*18*	*16*	*14*	*13*	*11*	*10*	*9*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
MPU Physical Gate Length (nm)	*9*	*8*	*7*	*6*	*6*	*5*	*4*
***Number of Chip I/Os** (Number of Total Chip Pads)—Maximum*							
Total pads—MPU unchanged	**3,072**	**3,072**	**3,072**	**3,072**	**3,072**	**3,072**	**3,072**
Signal I/O—MPU (% of total pads)	**33.3%**	**33.3%**	**33.3%**	**33.3%**	**33.3%**	**33.3%**	**33.3%**
Power and ground pads—MPU (% of total pads)	**66.7%**	**66.7%**	**66.7%**	**66.7%**	**66.7%**	**66.7%**	**66.7%**
IS Total pads—ASIC High Performance unchanged	**6,000**	**6,000**	**6,200**	**6,200**	**6,200**	**6,840**	**6,840**
Signal I/O pads— ASIC high-performance (% of total pads)	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**
Power and ground pads— ASIC high-performance (% of total pads)	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**	**50.0%**
***Number of Total Package Pins**—Maximum [1]*							
Microprocessor/controller, cost-performance	**880–4930**	**960–5423**	**960–5966**	**1050–6562**	**1050–7218**	**1155–7940**	**1155–8337**
Microprocessor/controller, high-performance	**6501**	**6826**	**7167**	**7525**	**7902**	**8297**	**8712**
ASIC (high-performance)	**6501**	**6826**	**7167**	**7525**	**7902**	**8297**	**8712**

ITRS Zukunftsprognosen

Taktfrequenzen

Table 4c Performance and Package Chips: Frequency On-chip Wiring Levels—Near-term Years

Year of Production	*2007*	*2008*	*2009*	*2010*	*2011*	*2012*	*2013*	*2014*	*2015*
DRAM ½ Pitch (nm) (contacted)	*65*	*57*	*50*	*45*	*40*	*36*	*32*	*28*	*25*
Flash ½ Pitch (nm) (un-contacted Poly)(f)	*54*	*45*	*40*	*36*	*32*	*28*	*25*	*22*	*20*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*68*	*59*	*52*	*45*	*40*	*36*	*32*	*28*	*25*
MPU Physical Gate Length (nm)	*25*	*23*	*20*	*18*	*16*	*14*	*13*	*11*	*10*
***Chip Frequency** (MHz)*									
On-chip local clock [1]	**4.700**	**5.063**	**5.454**	**5.875**	**6.329**	**6.817**	**7.344**	**7.911**	**8.522**
Maximum number wiring levels [3] ***[**]***	**11**	**12**	**12**	**12**	**12**	**12**	**13**	**13**	**13**

ITRS Zukunftsprognosen (cont.)

Taktfrequenzen

Table 4d Performance and Package Chips: Frequency On-chip Wiring Levels—Long-term Years

Year of Production	*2016*	*2017*	*2018*	*2019*	*2020*	*2021*	*2022*
DRAM ½ Pitch (nm) (contacted)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
Flash ½ Pitch (nm) (un-contacted Poly)(f)	*18*	*16*	*14*	*13*	*11*	*10*	*9*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
MPU Physical Gate Length (nm)	*9*	*8*	*7*	*6*	*6*	*5*	*4*
***Chip Frequency** (MHz)*							
On-chip local clock [1]	**9.180**	**9.889**	**10.652**	**11.475**	**12.361**	**13.315**	**14.343**
Maximum number wiring levels [3] ***[**]***	**13**	**14**	**14**	**14**	**14**	**15**	**15**

ITRS Zukunftsprognosen

Leistungsaufnahme

Table 6a Power Supply and Power Dissipation—Near-term Years

[1] Power will be limited more by system level cooling and test constraints than packaging

Year of Production	*2007*	*2008*	*2009*	*2010*	*2011*	*2012*	*2013*	*2014*	*2015*
DRAM ½ Pitch (nm) (contacted)	*65*	*57*	*50*	*45*	*40*	*36*	*32*	*28*	*25*
Flash ½ Pitch (nm) (un-contacted Poly)(f)	*54*	*45*	*40*	*36*	*32*	*28*	*25*	*22*	*20*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*68*	*59*	*52*	*45*	*40*	*36*	*32*	*28*	*25*
MPU Physical Gate Length (nm)	*25*	*23*	*20*	*18*	*16*	*14*	*13*	*11*	*10*
***Power Supply Voltage** (V)*									
V_{dd} (high-performance)	**1.1**	**1.0**	**1.0**	**1.0**	**0.95**	**0.90**	**0.90**	**0.90**	**0.80**
V_{dd} (Low Operating Power, high V_{dd} transistors)	**0.80**	**0.80**	**0.80**	**0.70**	**0.70**	**0.70**	**0.60**	**0.60**	**0.60**
***Allowable Maximum Power** [1]*									
High-performance with heatsink (W)	**189**	**198**	**198**	**198**	**198**	**198**	**198**	**198**	**198**
Maximum Affordable Chip Size Target for High-performance MPU Maximum Power Calculation	**310**	**310**	**310**	**310**	**310**	**310**	**310**	**310**	**310**
Maximum High-performance MPU Maximum Power Density for Maximum Power Calculation	**0.61**	**0.64**	**0.64**	**0.64**	**0.64**	**0.64**	**0.64**	**0.64**	**0.64**
Cost-performance (W)	**104**	**111**	**116**	**119**	**119**	**125**	**137**	**137**	**137**
Maximum Affordable Chip Size Target for Cost-performance MPU Maximum Power Calculation	**140**	**140**	**140**	**140**	**140**	**140**	**140**	**140**	**140**
Maximum Cost-performance MPU Maximum Power Density for Maximum Power Calculation	**0.74**	**0.79**	**0.83**	**0.85**	**0.85**	**0.89**	**0.98**	**0.98**	**0.98**
Battery (W)—(low-cost/hand-held)	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**

ITRS Zukunftsprognosen (cont.)

Leistungsaufnahme

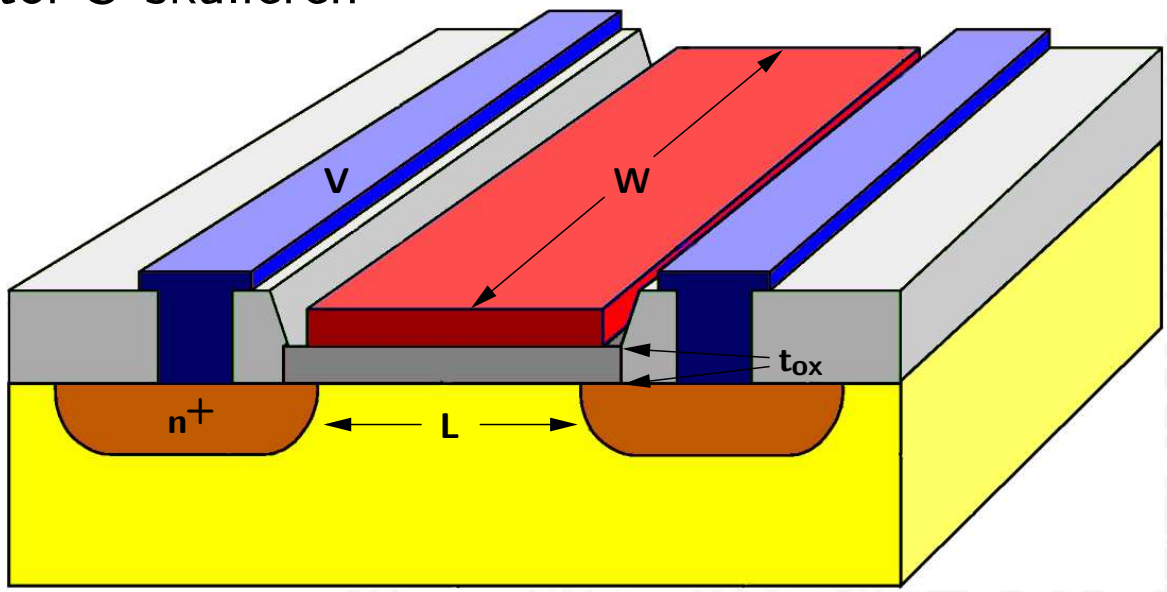
Table 6b Power Supply and Power Dissipation—Long-term Years

[1] Power will be limited more by system level cooling and test constraints than packaging

Year of Production	*2016*	*2017*	*2018*	*2019*	*2020*	*2021*	*2022*
DRAM ½ Pitch (nm) (contacted)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
Flash ½ Pitch (nm) (un-contacted Poly)(f)	*18*	*16*	*14*	*13*	*11*	*10*	*9*
MPU/ASIC Metal 1 (M1) ½ Pitch (nm) (f)	*22*	*20*	*18*	*16*	*14*	*13*	*11*
MPU Physical Gate Length (nm)	*9*	*8*	*7*	*6.3*	*5.6*	*5.0*	*4.5*
***Power Supply Voltage** (V)*							
V_{dd} (high-performance)	**0.80**	**0.70**	**0.70**	**0.70**	**0.65**	**0.65**	**0.65**
V_{dd} (Low Operating Power, high V_{dd} transistors)	**0.50**	**0.50**	**0.50**	**0.50**	**0.50**	**0.45**	**0.45**
***Allowable Maximum Power** [1]*							
High-performance with heatsink (W)	**198**	**198**	**198**	**198**	**198**	**198**	**198**
Maximum Affordable Chip Size Target for High-performance MPU Maximum Power Calculation	**310**	**310**	**310**	**310**	**310**	**310**	**310**
Maximum High-performance MPU Maximum Power Density for Maximum Power Calculation	**0.64**	**0.64**	**0.64**	**0.64**	**0.64**	**0.64**	**0.64**
Cost-performance (W)	**151**	**151**	**151**	**151**	**151**	**151**	**151**
Maximum Affordable Chip Size Target for Cost-performance MPU Maximum Power Calculation	**140**	**140**	**140**	**140**	**140**	**140**	**140**
Maximum Cost-performance MPU Maximum Power Density for Maximum Power Calculation	**1.08**	**1.08**	**1.08**	**1.08**	**1.08**	**1.08**	**1.08**
Battery (W)—(low-cost/hand-held)	**3**	**3**	**3**	**3**	**3**	**3**	**3**

Skalierung

Transistoren mit Faktor S skalieren



- *Constant voltage scaling*
  - bis Anfang der 90er Jahre
  - \+ Gatterverzögerungszeiten skalieren: $1/S^2$
  - – Leistungsdichte steigt: S^3

Skalierung (cont.)

- *Constant field scaling*
  - konstante Feldstärke $\Rightarrow$ sinkende Versorgungsspannung
  - Skalierung
    - alle Dimensionen: x, y, z = W, L, t_{ox}
    - Versorgungsspannung: V_{DD}
    - Dotierung: n^+
  - Gatterverzögerungszeiten skalieren: $1/S$
  - Leistungsverbrauch pro Transistor sinkt: $1/S^2$
  - Leistungsdichte, bleibt konstant 1
  - zusätzliche Effekte bei aktuellen Submikron-Prozessen
- *Lateral scaling*
  - nur Skalierung der Gate-Länge: L
- Skalierung der Chipfläche („Die size“) $D_C \approx 1{,}1$

Skalierung (cont.)

Parameter	Sensitivity	Constant Field	Lateral
Scaling Parameters			
Length: L		$1/S$	$1/S$
Width: W		$1/S$	1
Gate oxide thickness: t_{ox}		$1/S$	1
Supply voltage: V_{DD}		$1/S$	1
Threshold voltage: V_t		$1/S$	1
Substrate doping: N_A		S	1
Device Characteristics			
β	$\frac{W}{L}\frac{1}{t_{ox}}$	S	S
Current: I_{DS}	$\beta(V_{DD}-V_t)^2$	$1/S$	S
Resistance: R	$\frac{V_{DD}}{I_{DS}}$	1	$1/S$
Gate capacitance: C	$\frac{WL}{t_{ox}}$	$1/S$	$1/S$
Gate delay: τ	RC	$1/S$	$1/S^2$
Clock frequency: f	$1/\tau$	S	S^2
Dynamic power (per Gate): P	CV^2f	$1/S^2$	S
Chip Area: A		$1/S^2$	$1/S < A < 1$
Power density	P/A	1	S
Current density	I_{DS}/A	S	S

Skalierung (cont.)

Leitungen skalieren

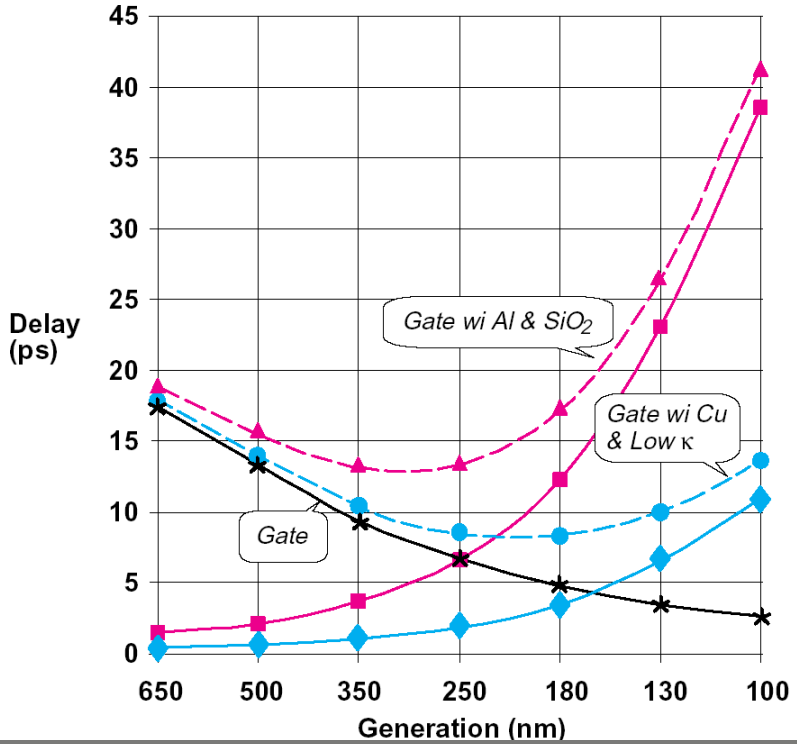
Parameter	Sensitivity	Reduced thickness	Constant thickness
Scaling Parameters			
Width: w		$1/S$	$1/S$
Spacing: s		$1/S$	$1/S$
Thickness: t		$1/S$	1
Interlayer oxide height: h		$1/S$	$1/S$
Characteristics per unit length			
Wire resistance: R_w	$\frac{1}{wt}$	S^2	S
Fringing capacitance: C_{wf}	$\frac{t}{s}$	1	S
Parallel plate capacitance: C_{wp}	$\frac{w}{h}$	1	1
Total wire capacitance: C_w	$C_{wf} + C_{wp}$	1	$1 \leq C_w \leq S$
Unrepeated RC constant: t_{wu}	$R_w C_w$	S^2	$S \leq t_{wu} \leq S^2$
Repeated wire RC delay: t_{wr}	$\sqrt{RCR_w C_w}$	$\sqrt{S}$	$1 \leq t_{wr} \leq \sqrt{S}$
Crosstalk noise	$\frac{t}{s}$	1	S

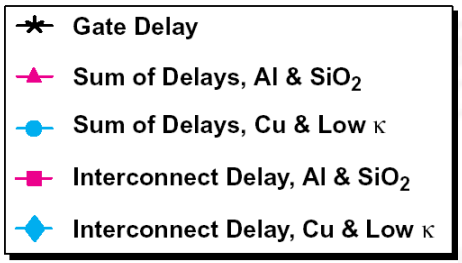
Skalierung (cont.)

Parameter	Sensitivity	Reduced thickness	Constant thickness
Scaling Parameters			
Width: w		$1/S$	$1/S$
Spacing: s		$1/S$	$1/S$
Thickness: t		$1/S$	1
Interlayer oxide height: h		$1/S$	$1/S$
Local/Scaled Interconnect Characteristics			
Length: l		$1/S$	$1/S$
Unrepeated wire RC delay	$l^2 t_{wu}$	1	$1/S \leq t_w \leq 1$
Repeated wire delay	$l t_{wr}$	$\sqrt{1/S}$	$1/S \leq t_w \leq \sqrt{1/S}$
Global Interconnect Characteristics			
Length: l		D_C	D_C
Unrepeated wire RC delay	$l^2 t_{wu}$	$S^2 D_C^2$	$S D_C^2 \leq t_w \leq S^2 D_C^2$
Repeated wire delay	$l t_{wr}$	$D_C \sqrt{S}$	$D_C \leq t_w \leq D_C \sqrt{S}$

Probleme

- Leitungslaufzeiten
  - lokale Leitungen: schneller
  - globale Leitungen: langsamer
  - nicht mehr alle Punkte des Chips in einem Taktzyklus erreichbar



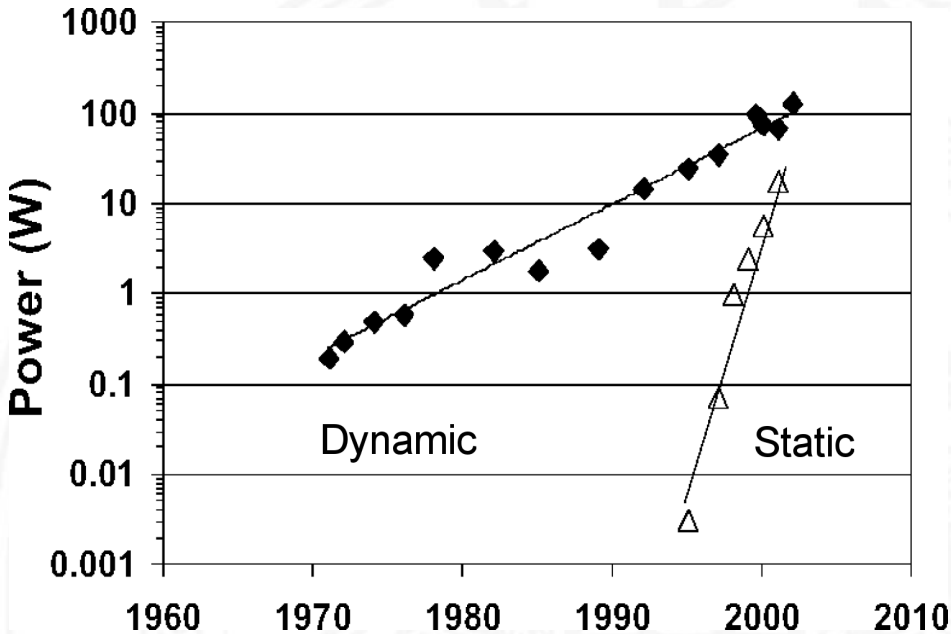


Al	*3.0 μΩ –cm*
Cu	*1.7 μΩ –cm*
SiO_2	*κ = 4.0*
Low κ	*κ = 2.0*
Al & Cu	*.8 μ Thick*
Al & Cu Line	*43 μ Long*

SIA'97
`http://www.sia-online.org`

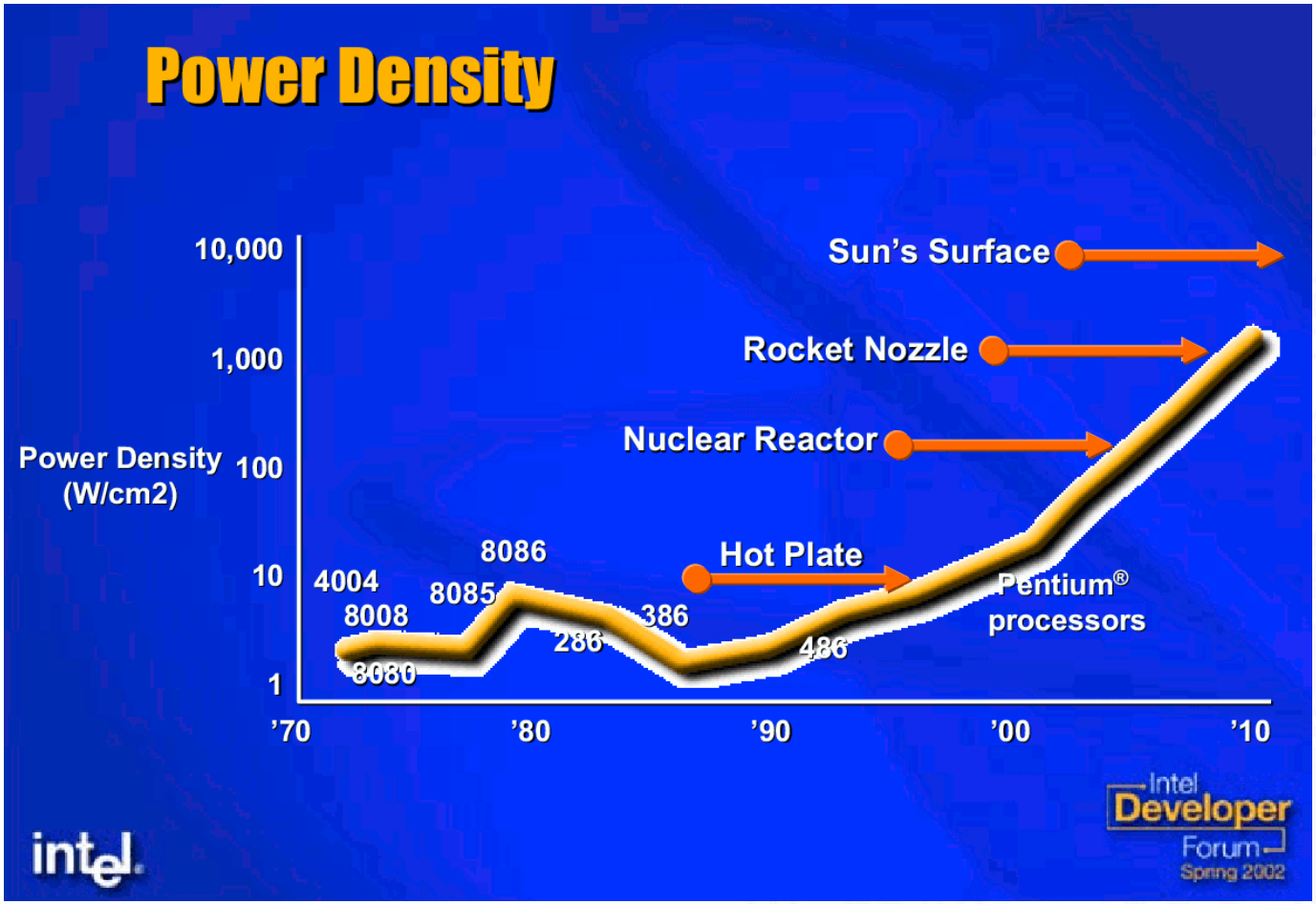
Probleme (cont.)

- Leistungsdichte
  - V_{DD} *muss kleiner werden* wegen
    - dynamischem Leistungsverbrauch
    - dünnerem Gateoxid: t_{ox}
    - kürzeren Kanälen: L



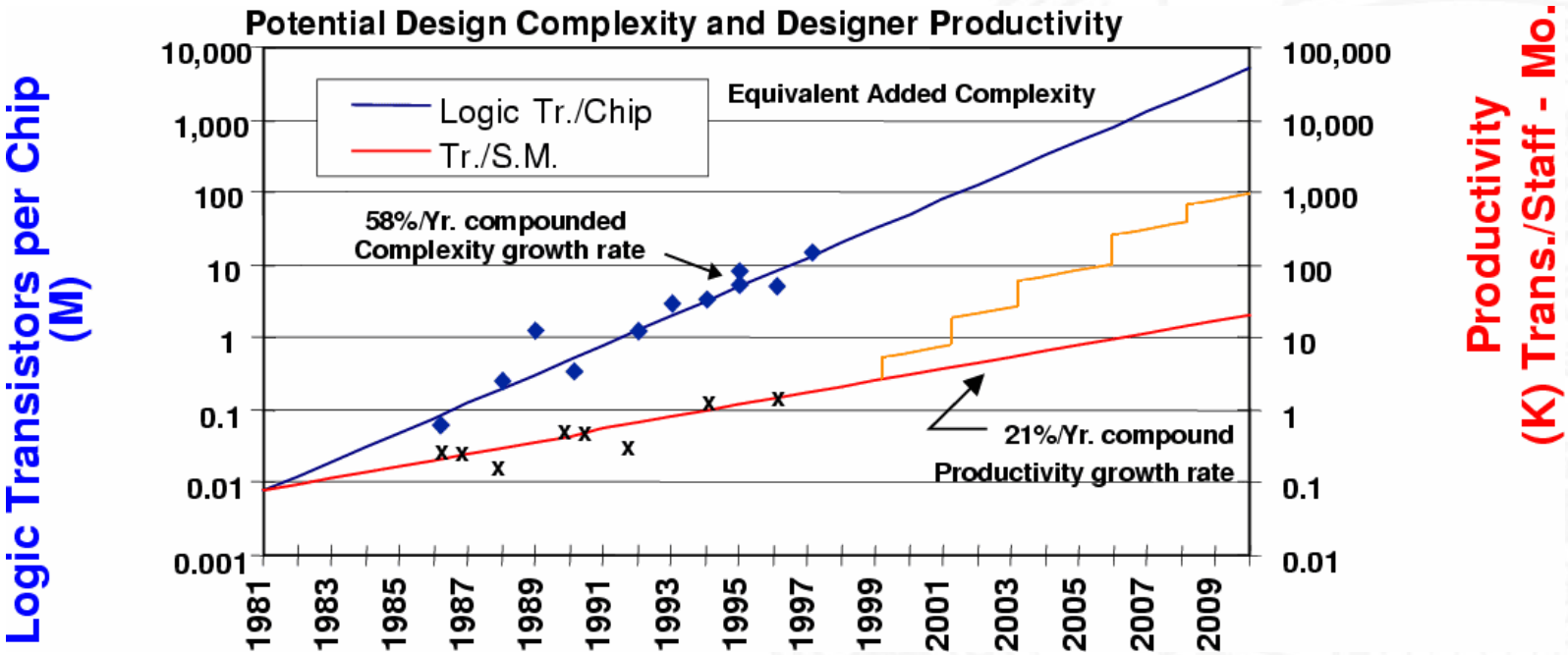
$\Rightarrow$ Schwellspannung U_P senken
$\Rightarrow$ Transistoren sperren schlechter
$\Rightarrow$ Leckströme steigen
– statischer Leistungsverbrauch steigt drastisch

Probleme (cont.)



Probleme (cont.)

- Produktivität
  - Anzahl der Transistoren wächst schneller als Produktivität
  - immer größere Teams (>500 Designer bei Prozessorentwurf)
  - Kosten



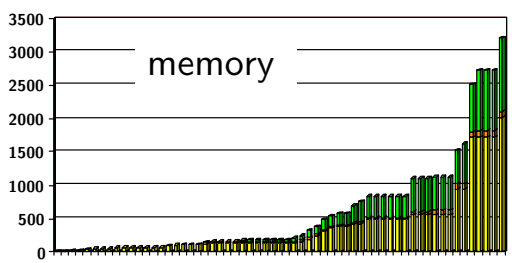
Probleme (cont.)

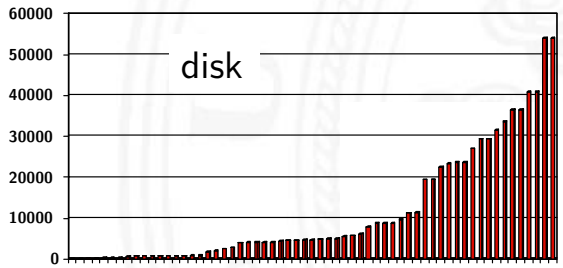
- Entwurfswerkzeuge / Hardwareaufwand
  - EDA-Werkzeuge + Hardware

Quelle: NVIDIA
DAC'02

The Investment

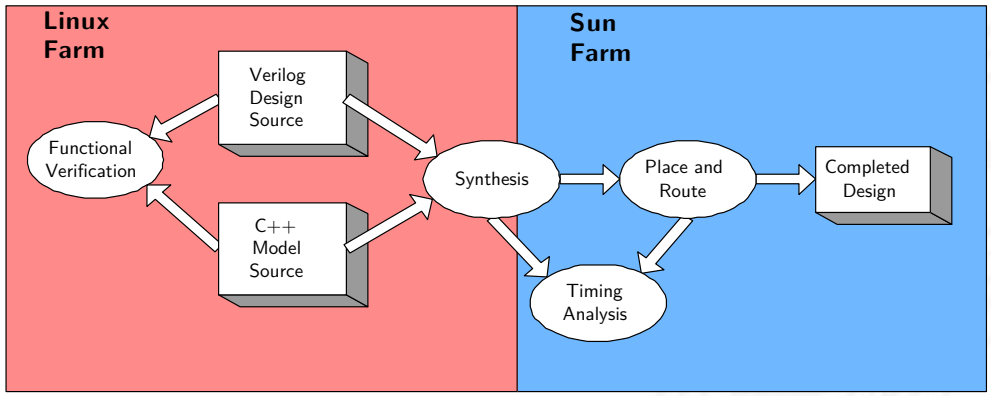
- ~$160M of CAD tools installed and online
- ~$40M Emulation installation
- Engineering Compute resources
  - Desktops: 225 Suns (Solaris), 5000 x86 PCs (Linux/NT)
  - Server CPUs: 450 Sparc (Solaris), 2700 x86 (Linux/NT)
  - 3.2 Terabytes RAM, 55 Terabytes of storage

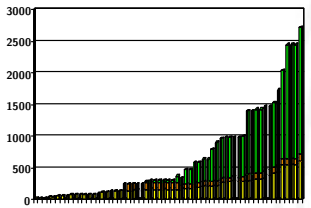




Probleme (cont.)

Compute Resource Allocation





Probleme (cont.)

- Hardware Emulation $>$ 100 Mio. \$ DAC'06

Probleme (cont.)

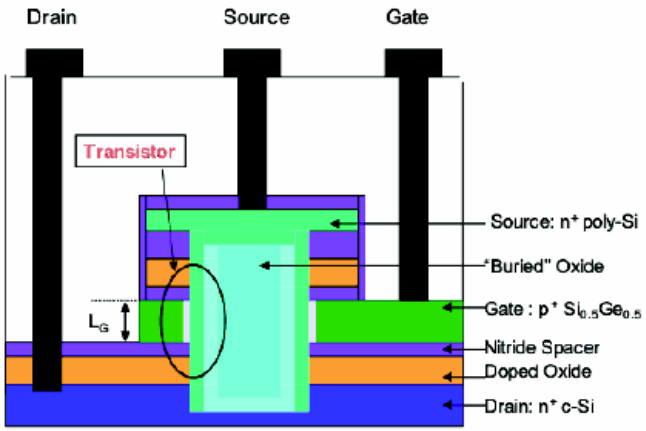
- Technologische Probleme
  - Skalierung der Dotierung
    - Dotierung als statistischer Prozess
    - bei 50 nm Kanallänge: $\approx$ 1000 Dotieratome/μm Weite, $\pm$ 30
  - Lithografie
    - Strukturen kleiner als Wellenlänge
    - Tiefenschärfe im Resist
  - Kurzkanaleffekte
    - Hot-Elektron Effekt
    - Kanallängenmodulation
    - Drain-Source Durchbruch
  - Tunnelstrom durch Gateoxid
  - wachsende Kontaktwiderstände
  - ...

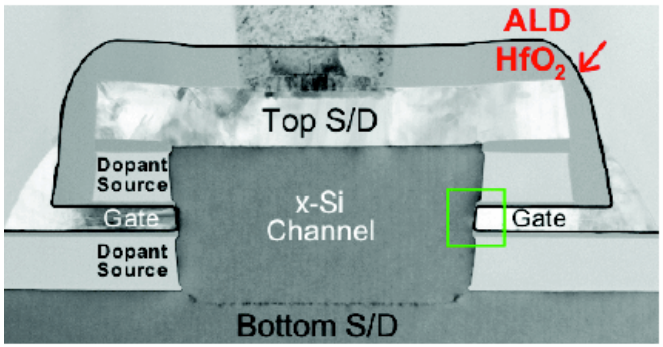
Neue Technologien, Materialien und Verfahren

- EUV-Lithografie
- „gestrecktes“ Silizium, erhöhte Leitfähigkeit im Kanal
- SOI – **S**ilicon **O**n **I**nsulator
- high-k Dielektrikum
- Kupfer Metallisierung
- Metall-Gates
- neue Transistoren: vertikal, multi-Gate ...
- Nanotechnologien
- ...

Neue Technologien, Materialien und Verfahren (cont.)

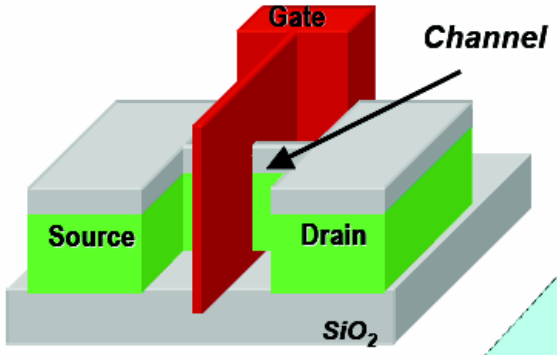
Vertical FET

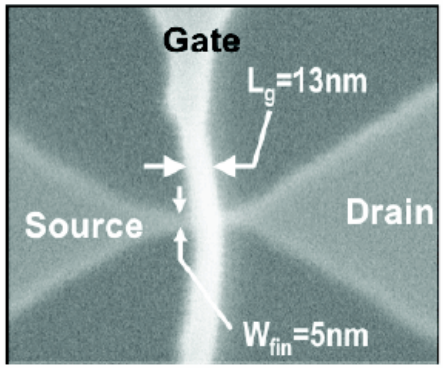




Stanford, AT&T

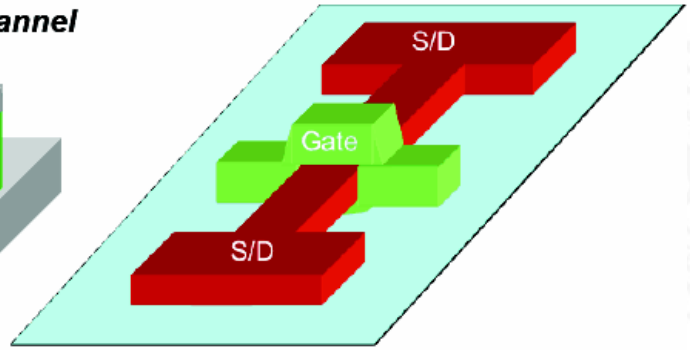
Double Gate FinFET

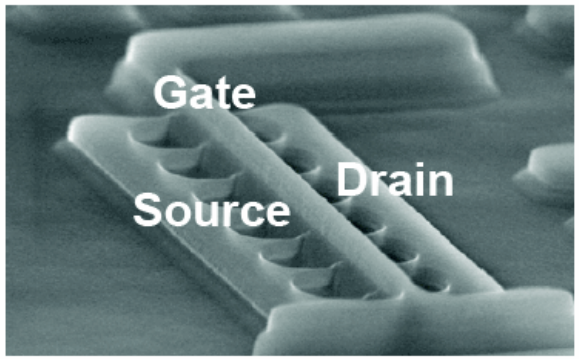




UC Berkeley

Tri Gate FET





Intel

physikalische Limits

„Harte“ Grenzen, sofern keine grundsätzlich andere Technologie (Photonik, Spintronik etc.), bzw. Funktionsprinzipien (Quantencomputer) zum Einsatz kommen

- Tunnelstrom durch den Kanal und der geschaltete Strom müssen unterscheidbar sein
- Thermisches Rauschen und die Gate-Ladungen der Transistoren müssen unterscheidbar sein

$\Rightarrow$ Kanallänge $\approx 10\,\text{nm}$

$\Rightarrow$ Spannungspegel $\approx 0{,}1\,\text{V}$

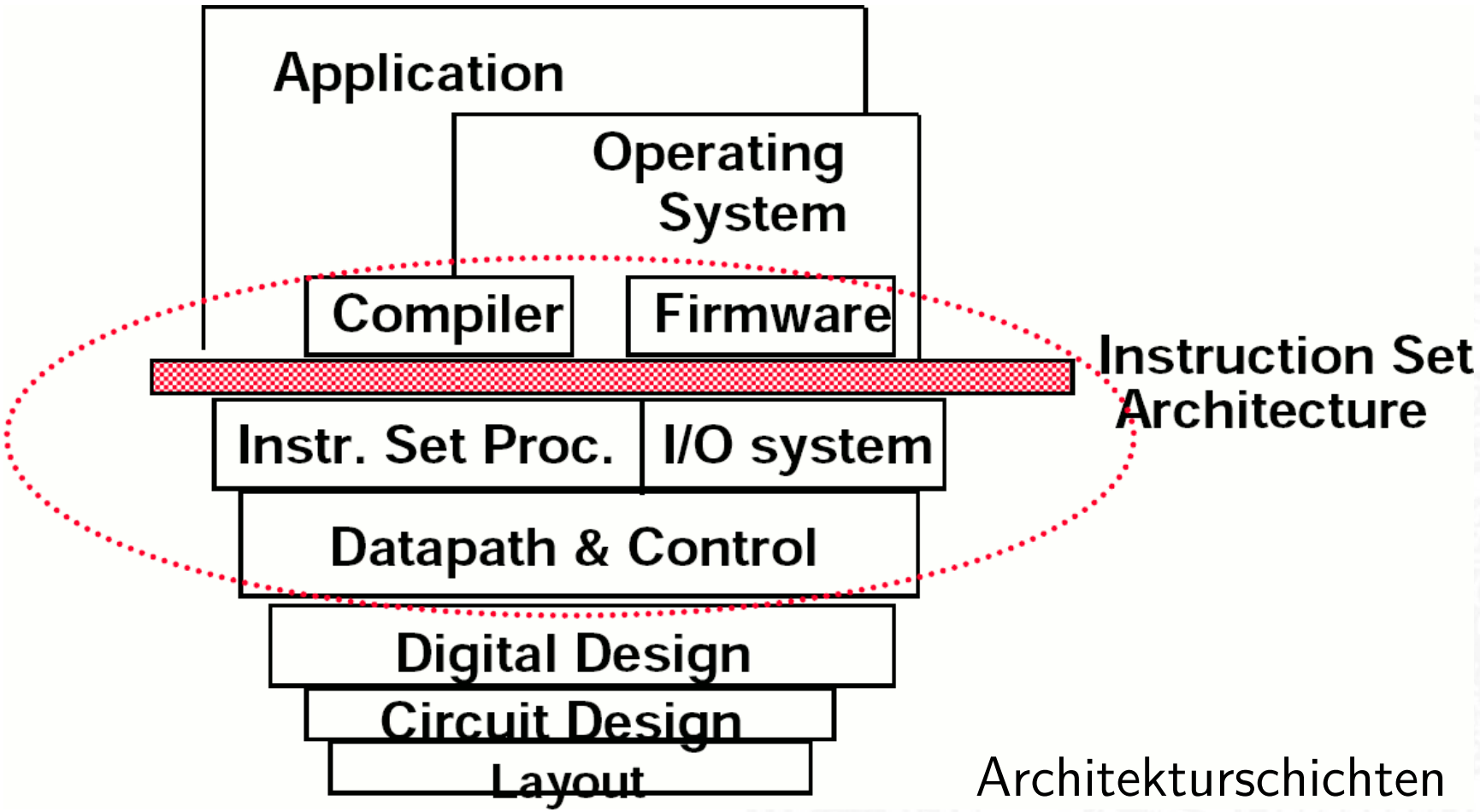
$\Rightarrow$ geschaltete Energie $\approx 1{,}5\,\text{eV}$

$\Rightarrow$ Gate Kapazität $\approx 1\,\text{aF}$

$\Rightarrow$ 5 Elektronen im Gate um zu schalten

Gliederung

Schnittstellen zu anderen Inhalten des Studiums



Architekturschichten

Schnittstellen zu anderen Inhalten des Studiums (cont.)

Sehr starke Wechselwirkungen der Rechnerarchitektur mit anderen Bereichen der Informatik

- Rechnerorganisation
- Betriebssysteme
- Compilerbau
- Warteschlangentheorie

Der Bereich der Rechnerarchitektur liefert genug Stoff für mehrere eigene Vorlesungen, deshalb kann hier nur einführende Übersicht folgen

Schnittstellen zu anderen Inhalten des Studiums (cont.)

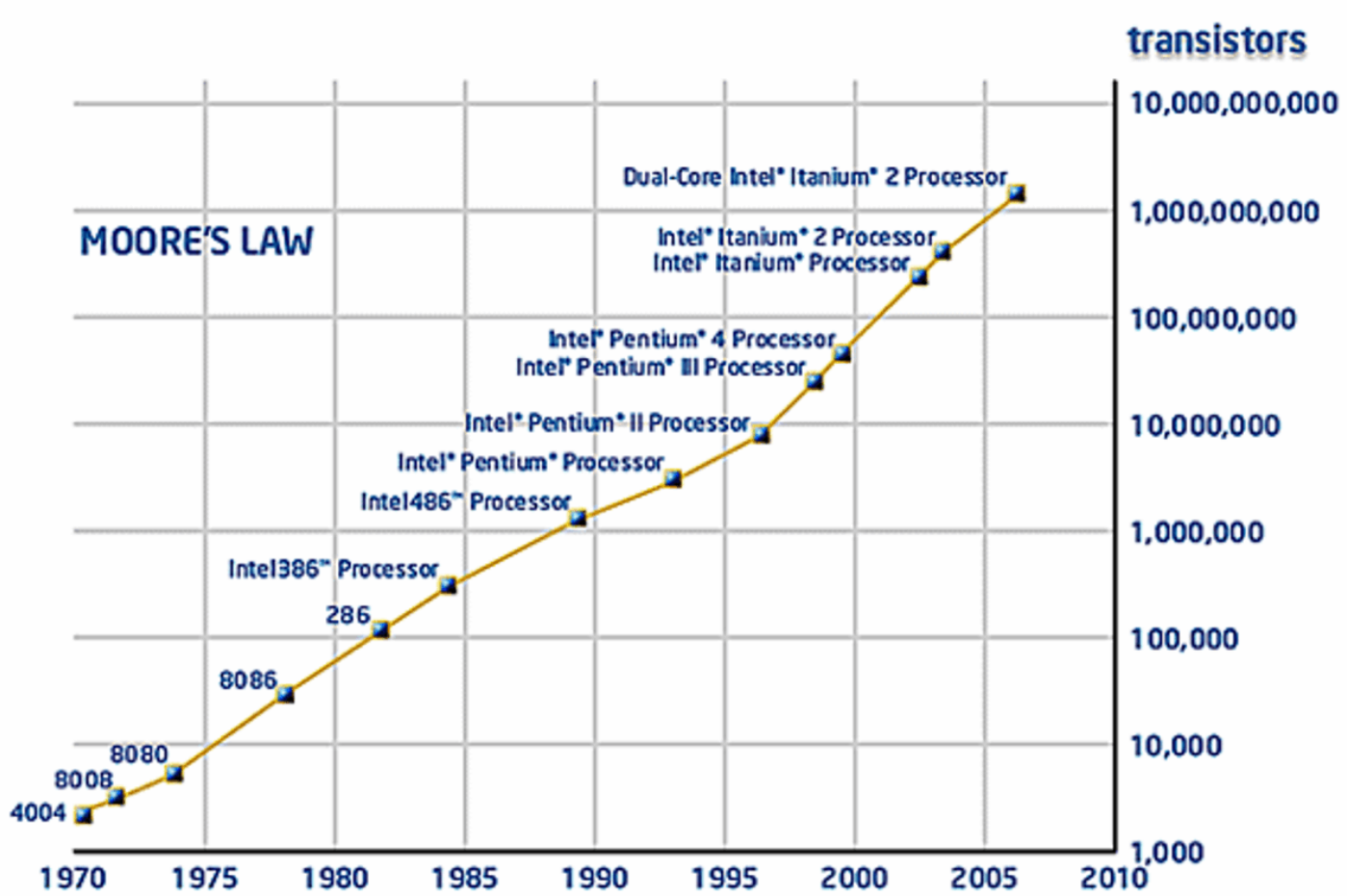
Literatur – besonders diese

- D. Patterson, J. Hennessy: *Rechnerorganisation und -entwurf; Die Hardware/Software-Schnittstelle* [PH05]
- D. Patterson, J. Hennessy: *Computer Organization and Design; The Hardware/Software Interface* [PH09]
- J. Hennessy, D. Patterson: *Rechnerarchitektur; Analyse, Entwurf, Implementierung, Bewertung* [HP94]
- J. Hennessy, D. Patterson: *Computer architecture; A quantitative approach* [HP07]
- A. Tanenbaum: *Computerarchitektur; Strukturen, Konzepte, Grundlagen* [Tan06]

Schnittstellen zu anderen Inhalten des Studiums (cont.)

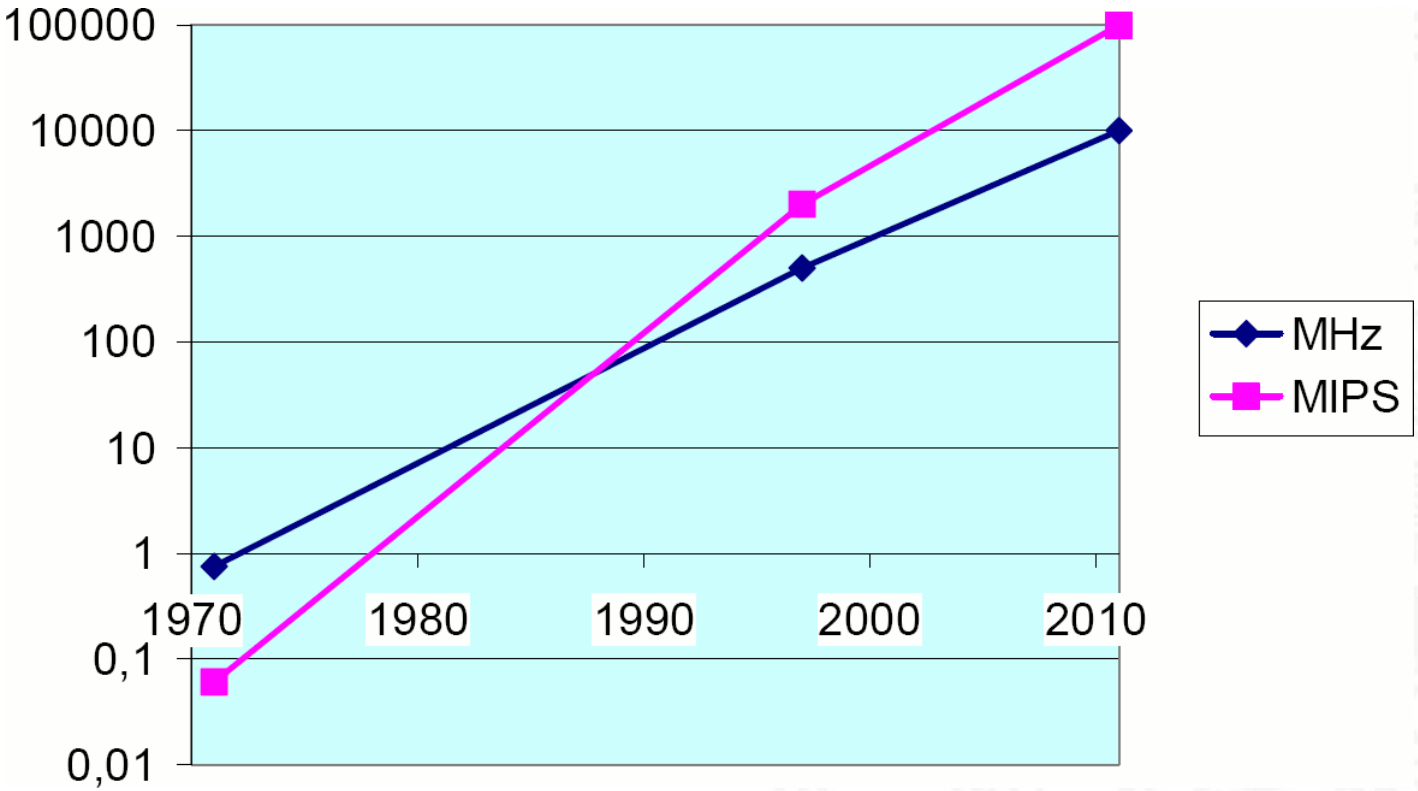
- `http://de.wikipedia.org` und `http://en.wikipedia.org`
- C. Märtin: *Einführung in die Rechnerarchitektur: Prozessoren und Systeme* [Mär03]
- D. Comer: *Essentials of Computer Architecture* [Com05]
- W. Oberschelp, G. Vossen: *Rechneraufbau und Rechnerstrukturen* [OV06]
- M. Dal Cin: *Rechnerarchitektur* [DC96]

Was tun mit Moore's Law?



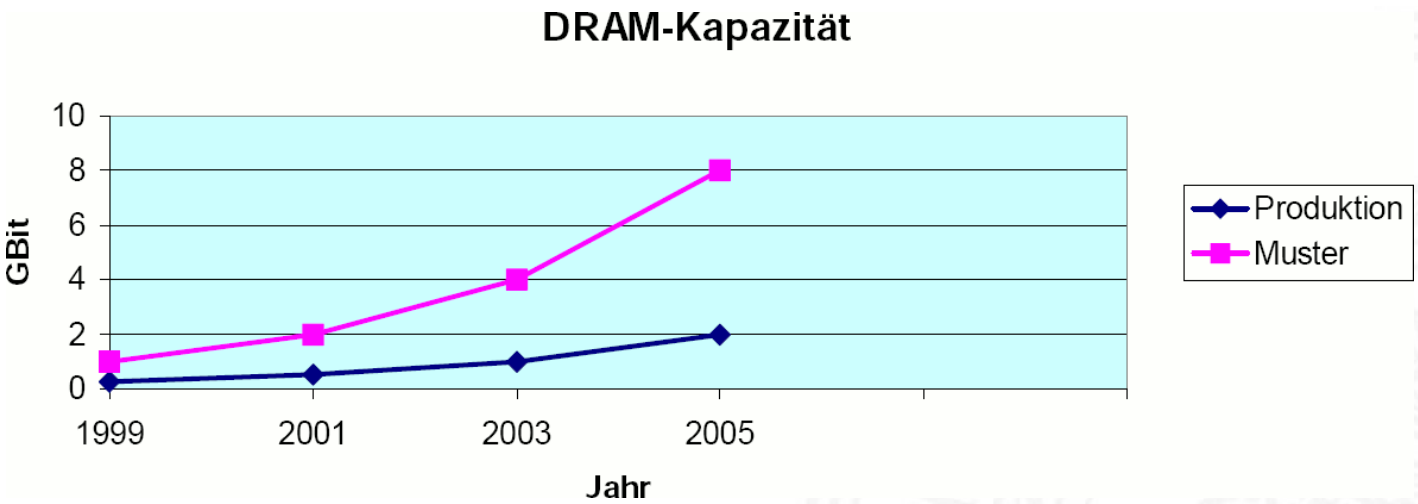
Was tun mit Moore's Law? (cont.)

- Universalrechner: CPUs, Mikrocontroller...



Was tun mit Moore's Law? (cont.)

- reguläre Strukturen: alle Arten von RAM



- . . . im Folgenden geht es um die Frage, nach welchen Prinzipien Prozessoren aufgebaut sind und damit: wie man sie entwirft

Was ist Rechnerarchitektur?

Definitionen

1. *The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behaviour, as distinct from the organization and data flow and control, the logical and the physical implementation.* [Amdahl, Blaauw, Brooks]

2. *The study of computer architecture is the study of the organization and interconnection of components of computer systems. Computer architects construct computers from basic building blocks such as memories, arithmetic units and buses.*

Was ist Rechnerarchitektur? (cont.)

From these building blocks the computer architect can construct anyone of a number of different types of computers, ranging from the smallest hand-held pocket-calculator to the largest ultra-fast super computer. The functional behaviour of the components of one computer are similar to that of any other computer, whether it be ultra-small or ultra-fast.

By this we mean that a memory performs the storage function, an adder does addition, and an input/output interface passes data from a processor to the outside world, regardless of the nature of the computer in which they are embedded. The major differences between computers lie in the way of the modules are connected together, and the way the computer system is controlled by the programs. In short, computer architecture is the discipline devoted to the design of highly specific and individual computers from a collection of common building blocks. [Stone]

Was ist Rechnerarchitektur? (cont.)

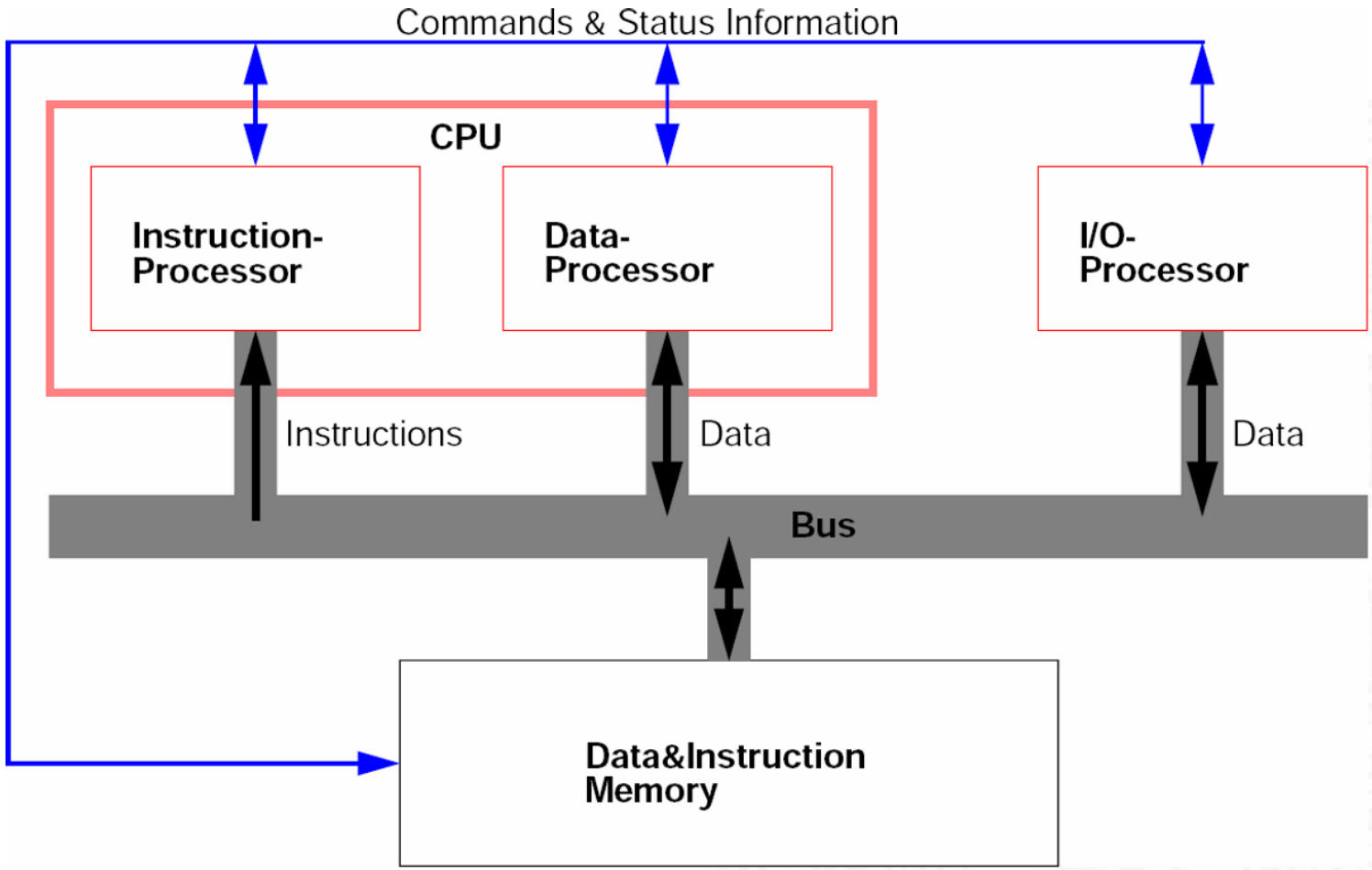
Zwei Aspekte der Rechnerarchitektur

1. Operationsprinzip: das funktionelle Verhalten der Architektur
   - = Programmierschnittstelle
   - = ISA – **I**nstruction **S**et **A**rchitecture
2. Hardwarestruktur: beschrieben durch Art und Anzahl der Hardware-Betriebsmittel sowie die sie verbindenden Kommunikationseinrichtungen
   - = Mikroarchitektur, beispielsweise „von-Neumann“ Architektur

von-Neumann Architektur

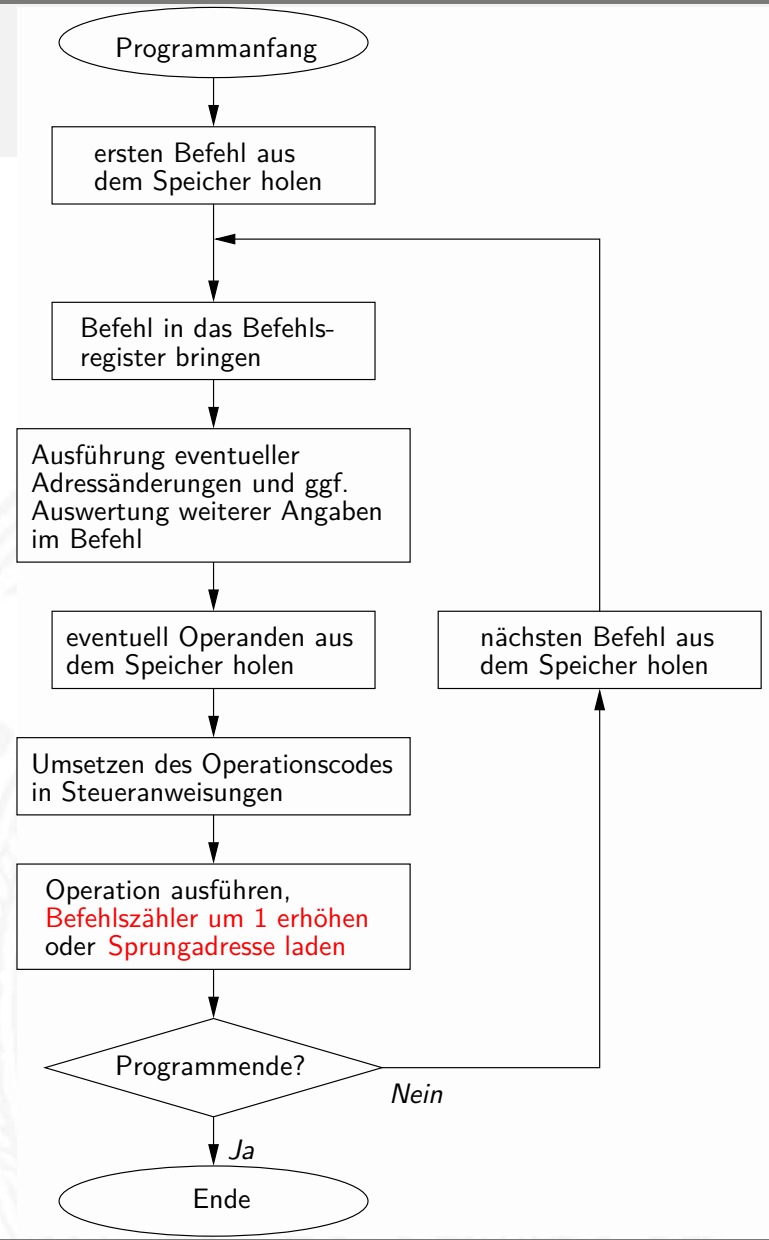
- Historie: 1945 entwickelt (John von Neumann)
- Abstrakte Maschine mit minimalem Hardwareaufwand
- Komponenten
  - zentralen Recheneinheit: CPU
    - logisch unterteilt in
    1. Datenprozessor / Rechenwerk / Operationswerk
    2. Befehlsprozessor / Leitwerk / Steuerwerk
  - Speicher für Daten und Befehle, fortlaufend adressiert
  - Ein/Ausgabe-Einheit zur Anbindung peripherer Geräte
  - Bussystem(e) verbinden diese Komponenten
  - die Struktur ist unabhängig von dem Problem, das Problem wird durch austauschbaren Speicherinhalt (Programm) beschrieben

von-Neumann Architektur (cont.)



von-Neumann Architektur (cont.)

- Paradigma der Programmverarbeitung
  - Programm als Sequenz elementarer Anweisungen (Befehle)
  - als Bitvektoren im Speicher codiert
  - Interpretation (Operanden, Befehle und Adressen) ergibt sich aus dem Kontext (der Adresse)
  - zeitsequenzielle Ausführung der Instruktionen



von-Neumann Architektur (cont.)

– „von-Neumann Flaschenhals“: Zugriff auf Speicher

... vieles davon gilt Heute noch, außer

- *parallele*, statt sequentieller Befehlsabarbeitung
  Stichwort: superskalare Prozessoren
- dynamisch veränderte Abarbeitungsreihenfolge
  Stichwort: *„out-of-order execution“*
- häufig getrennte Daten- und Instruktionsspeicher
  Stichwort: *Harvard-Architektur*
- *Speicherhierarchie*, Caches etc.

Kriterien beim Entwurf

- Architekt sucht beste Lösung im Suchraum möglicher Entwürfe
- Kriterien „guter“ Architekturen:
  - hohe Rechenleistung
  - zuverlässig, robust
  - modular, skalierbar
  - einfach handhabbar, programmierbar
  - orthogonal
  - ausgewogen
  - wirtschaftlich, adäquat
  - ...

Kriterien beim Entwurf (cont.)

Skalierbarkeit Hinzufügen weiterer Module (ohne zusätzliche Änderungen) verbessert das System
⇒ Erweiterbarkeit, Wirtschaftlichkeit

Orthogonalität Jedes Modul hat eine definierte Funktionalität; keine zwei Module bieten die gleiche Funktionalität
⇒ Wartbarkeit, Kosten, Handhabbarkeit

Adäquatheit Die Kosten eines Moduls sind adäquat zur Funktion
⇒ Performance, Kosten

Virtualität Elimination physikalischer Grenzen: virtueller Prozessor, virtueller Speicher, virtuelle Kanäle...
⇒ skalierbar, ausbaubar, einfache Programmierung

Kriterien beim Entwurf (cont.)

Transparenz unwichtige Details werden verborgen
⇒ einfache Programmierung

Performance-Transparenz Änderung der System-Konfiguration ohne die Funktionalität zu beeinflussen
⇒ Skalierbarkeit, Wartbarkeit, Zuverlässigkeit

Größentransparenz Erweiterung des System, so dass sich die Performance verbessert
⇒ Skalierbarkeit, Kosten

Fehlertransparenz System verbirgt, maskiert oder toleriert Fehler
⇒ Zuverlässigkeit

- Die Punkte sind hier für die *Mikroarchitektur* formuliert, gelten aber gleichermaßen für die *ISA*

Kriterien zur Architekturbewertung

Kenngrößen zur Bewertung

- Taktfrequenz
- Werte die sich aus Eigenschaften der Architektur ergeben
- Ausführungszeiten von Programmen
- Durchsätze
- statistische Größen
- ...

Die Wahl der Kenngrößen hängt entscheidend von der jeweiligen Zielsetzung ab

Kriterien zur Architekturbewertung (cont.)

Verfahren zur Bestimmung der Kenngrößen

- *Benchmarking*: Laufzeitmessung bestehender Programme
  - Standard Benchmarks
    SPEC Standard Performance Evaluation Corporation
    `http://www.spec.org`
    TPC Transaction Processing Performance Council
    `http://www.tpc.org`
  - profilspezifische Benchmarks: SysMark, PCmark, Winbench etc.
  - benutzereigene Anwendungsszenarien
- *Monitoring*: Messungen während des Betriebs
- Modelltheoretische Verfahren: Analytische Modelle, Simulation...

Kenngrößen

Taktfrequenz

- In den letzten Jahren erfolgreich beworben!

$\Rightarrow$ *für die Leistungsbewertung aber völlig ungeeignet*

theoretische Werte

- MIPS – **M**illion **I**nstructions **P**er **S**econd
- MFLOPS – **M**illion **Fl**oating Point **O**perations **P**er **S**econd
  - keine Angabe über die Art der Instruktionen und deren Ausführungszeit
  - nicht direkt vergleichbar
- innerhalb einer Prozessorfamilie sinnvoll

Kenngrößen (cont.)

Ausführungszeit

- Benutzer: *Wie lange braucht mein Programm?*
- Gesamtzeit: Rechenzeit + Ein-/Ausgabe, Platten- und Speicherzugriffe...
- CPU-Zeit: Unterteilung in System- und Benutzer-Zeit

Unix `time`-Befehl: `597.07u 0.15s 9:57.61 99.9%`

`597.07`	user CPU time [sec.]
`0.15`	system CPU time
`9:57.61`	elapsed time
`99.9`	CPU/elapsed [%]

Kenngrößen (cont.)

Theoretische Berechnung der CPU-Zeit (user CPU time)

- CPU-Zeit $= IC \cdot CPI \cdot T$

IC	Anzahl auszuführender Instruktionen	**I**nstruction **C**ount
CPI	mittlere Anzahl Takte pro Instruktionen **C**ycles **p**er **I**nstruction	
T	Taktperiode	

- IC kleiner: weniger Instruktionen
  - bessere Algorithmen
  - bessere Compiler
  - mächtigerer Befehlssatz

Kenngrößen (cont.)

- *CPI* kleiner: weniger Takte pro Instruktion
  - parallel Befehle ausführen: VLIW...
  - parallel Teile der Befehle bearbeiten: Pipelining, Superskalar...
- *T* kleiner: höhere Taktfrequenz
  - Technologie
- genauere Untersuchung wenn CPI über die Haufigkeiten und Zyklenanzahl einzelner Befehle berechnet wird
- so lassen sich beispielsweise alternative Befehlssätze miteinander vergleichen

Kenngrößen (cont.)

CPU-Durchsatz

- RZ-Betreiber
  - *Wie viele Aufträge kann die Maschine gleichzeitig verarbeiten?*
  - *Wie lange braucht ein Job im Mittel?*
  - *Wie viel Arbeit kann so pro Tag erledigt werden?*

$\Rightarrow$ Latenzzeit: *Wie lange dauert es, bis mein Job bearbeitet wird?*

$\Rightarrow$ Antwortzeit: *Wie lange rechnet mein Job?*

- Modellierung durch Warteschlangentheorie: Markov-Ketten, stochastische Petri-Netze...

Kenngrößen (cont.)

statistische Werte zur Zuverlässigkeit

- Betriebssicherheit des Systems: „Quality of Service“
- Fehlerrate: Fehlerursachen pro Zeiteinheit
  Ausfallrate: Ausfälle pro Zeiteinheit
  - *Fault:* Fehlerursache
  - *Error:* fehlerhafter Zustand
  - *Failure:* ein Ausfall ist aufgetreten
- MTTF Mean Time To Failure
  MTBF Mean Time Between Failures
  MTTR Mean Time To Repair
  MTBR Mean Time Between Repairs

Kenngrößen (cont.)

- Überlebenswahrscheinlichkeit, *Reliability*
  - Maß für kontinuierlichen korrekten Service
  - Wahrscheinlichkeit, dass System nach Zeit t noch funktioniert, wenn es zum Zeitpunkt t_0 funktioniert hat:

    $R(t) = e^{-\lambda t}$ zeitunabhängige Ausfallrate: λ
- Verfügbarkeit, *Availability*
  - Maß für korrekten Service, u.U. mit Unterbrechungen
  - Wahrscheinlichkeit, dass System zum Zeitpunkt t funktioniert:

    $A(t) =$ konstant
- Sicherheit, *Safety*
  - Maß für Service ohne katastrophale Fehler
  - Wahrscheinlichkeit, dass System trotz internem Fehler keinen (katastrophalen) Ausfall hat

Kenngrößen (cont.)

- Robustheit, *Robustness*
  - Maß für das korrekte Systemverhalten trotz fehlerhafter Eingaben
  - Wahrscheinlichkeit, dass System trotz internem Fehler nicht ausfällt und keine falschen Ausgaben produziert
- Sicherheit, *Security*
  - Maß für den Aufwand um unerlaubten Zugriff auf Informationen zu verhindern
- Wartbarkeit, *Maintainability*
  - Maß für den Aufwand das System funktionsfähig zu halten

Bewertung von Maßnahmen

Wie wirken sich Verbesserungen der Rechnerarchitektur aus?

- Speed-Up: Verhältnis von Ausführungsszeiten *vor* und *nach* der Verbesserung

  Speed-Up $= T_{\mathbf{v}orVerbesserung} / T_{\mathbf{n}achVerbesserung}$
- werden nur Teile der Berechnung beschleunigt, Faktor F:

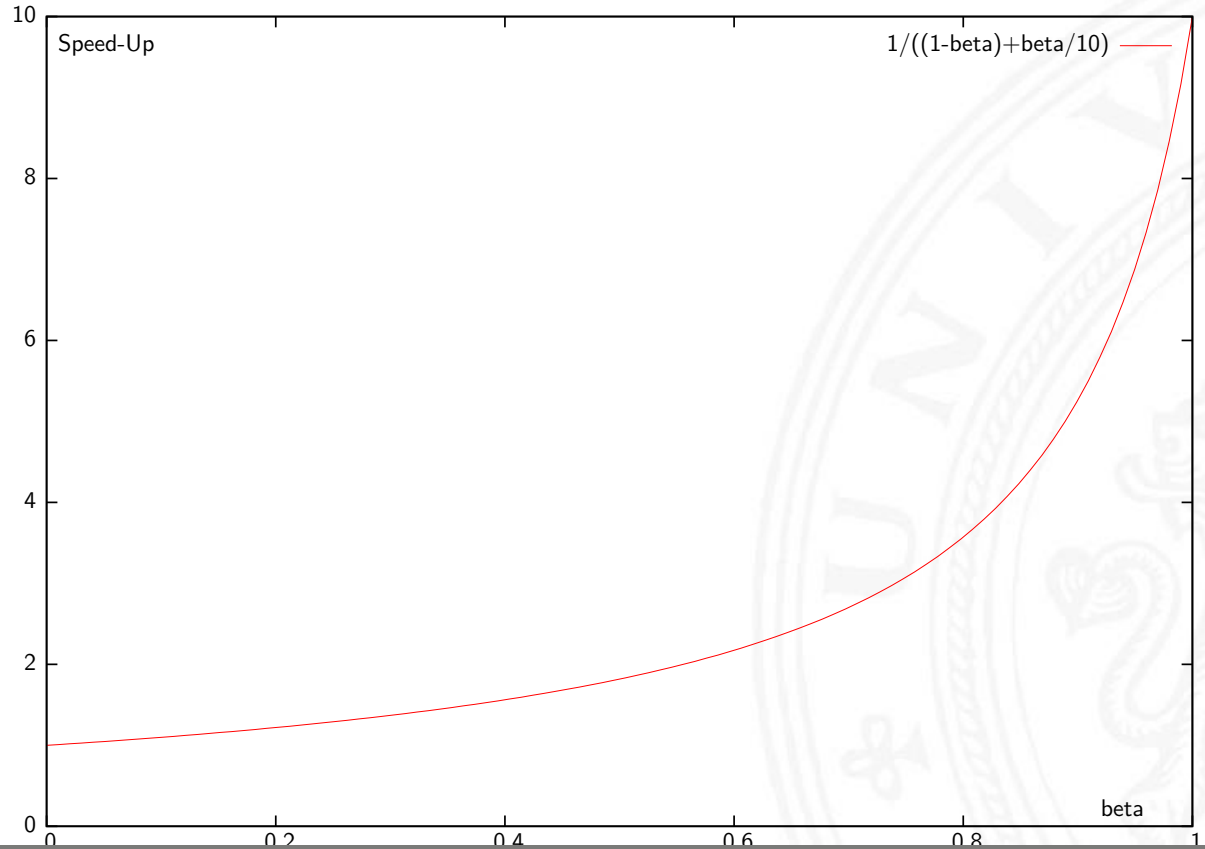
$$\begin{aligned} T &= T_{\mathbf{o}hneEffekt} + T_{\mathbf{m}itEffekt} \\ T_n &= T_{v,o} + T_{v,m}/F_{Verbesserung} \end{aligned}$$

$\Rightarrow$ den „Normalfall“, den häufigsten Fall beschleunigen, um den größten Speed-Up zu erreichen

Bewertung von Maßnahmen (cont.)

- Amdahlsches Gesetz (s.u.)

$$\text{Speed-Up} = \frac{1}{(1-\beta)+\beta/F_{Verbesserung}} \text{ mit } \beta = T_{v,m}/T_v$$



$F_{Verbesserung} = 10$

Bewertung von Maßnahmen (cont.)

Amdahlsches Gesetz

- Beschleunigung der Bearbeitung durch Parallelausführung mit N Prozessoren (Gene Amdahl '67)
- Speed-Up $= \frac{1}{(1-\beta)+f_k(N)+\beta/N} \leq \frac{1}{(1-\beta)}$

N	# Prozessoren als Verbesserungsfaktor
β	Anteil parallelisierbarer Berechnung
$1-\beta$	Anteil nicht parallelisierbarer Berechnung
$f_k()$	Kommunikationsoverhead zwischen den Prozessoren

  - Aufgaben verteilen
  - Arbeit koordinieren
  - Ergebnisse zusammensammeln

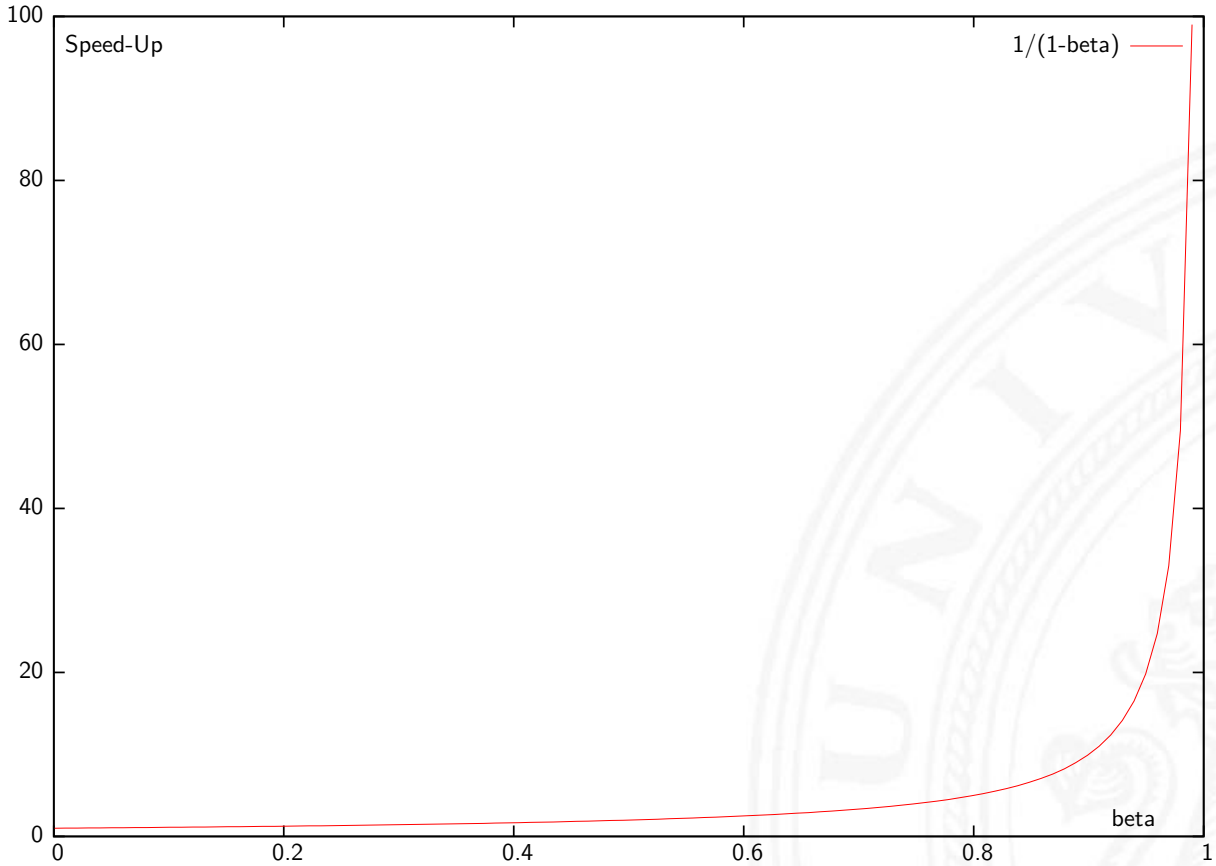
Bewertung von Maßnahmen (cont.)

- *Welche Auswirkungen hat das in der Praxis?*

N	β	Speed-up
2	0,40	1,25
4	0,40	1,43
4536	0,80	5,00
9072	0,99	98,92

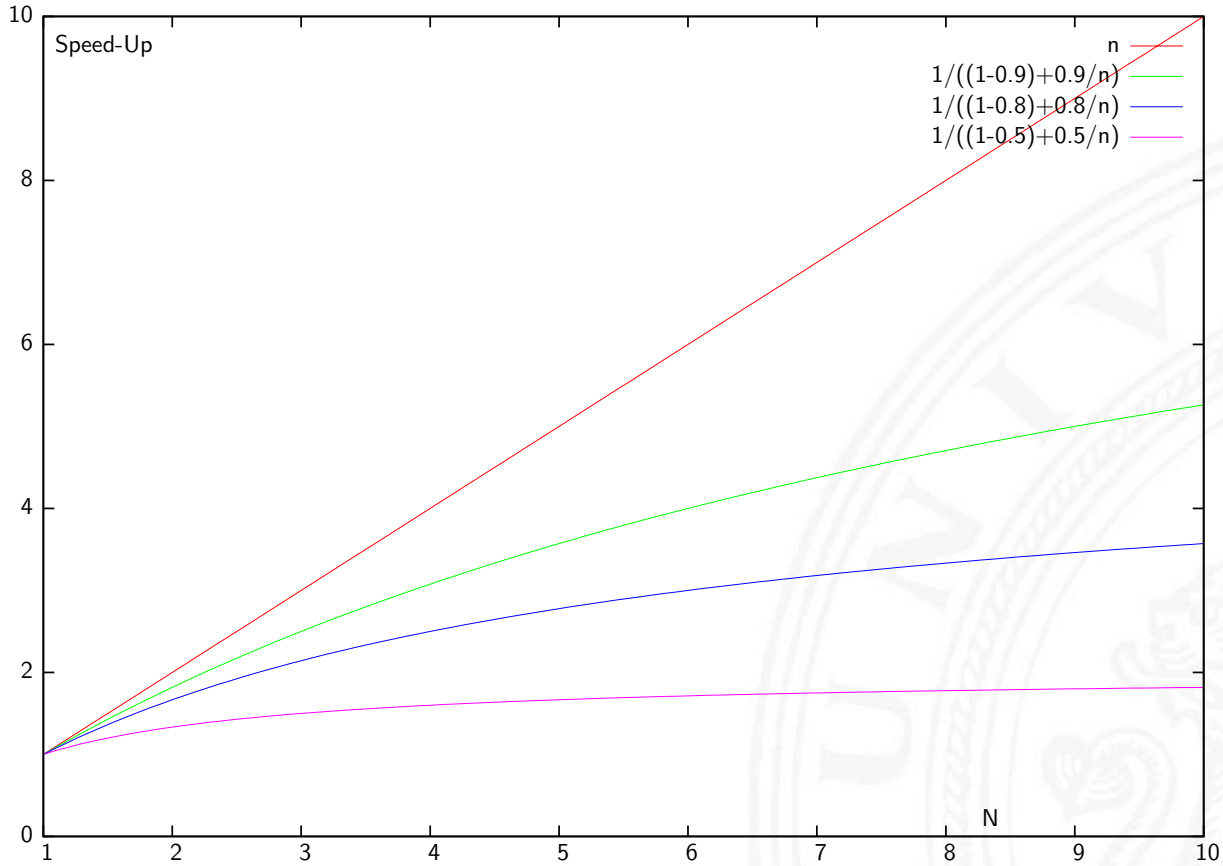
  - Enttäuschend: zwei Prozessoren sind *nicht* doppelt so schnell
  - . . . immerhin bei Multitasking und mehreren Prozessen kommt man auf große β

Bewertung von Maßnahmen (cont.)



$F_{Verbesserung} = \infty$

Bewertung von Maßnahmen (cont.)



Bewertung von Maßnahmen (cont.)

Gustafsonsches Gesetz	Abhängigkeit von N
Umkehrung von Amdahl's Gesetz	–"– β

- Speed-Up eines Parallelrechners gegenüber einem ein-Prozessor Rechner

$$T_n = T_{v,o} + N \cdot T'_{v,m}$$

$$\text{Speed-Up} = \frac{T_{v,o} + N \cdot T'_{v,m}}{T_{v,o} + T'_{v,m}}$$

$$= 1 + (N-1)\beta' \quad \text{mit } \beta' = \frac{T'_{v,m}}{T_{v,o} + T'_{v,m}}$$

Klassifikation

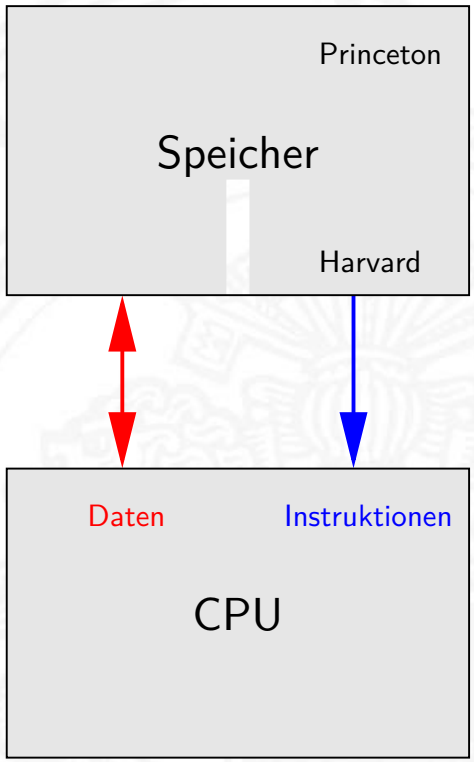
- Klassifikation nach Flynn
  Unterscheidung nach Kontroll- und Datenfluss
  - SI **S**ingle **I**nstruction
  - MI **M**ultiple **I**nstruction
  - SD **S**ingle **D**ata
  - MD **M**ultiple **D**ata
  - $\Rightarrow$ Klassen: SISD, SIMD, MIMD

Klassifikation (cont.)

SISD **S**ingle **I**nstruction Stream, **S**ingle **D**ata Stream

Abhängig von der Speicheranbindung

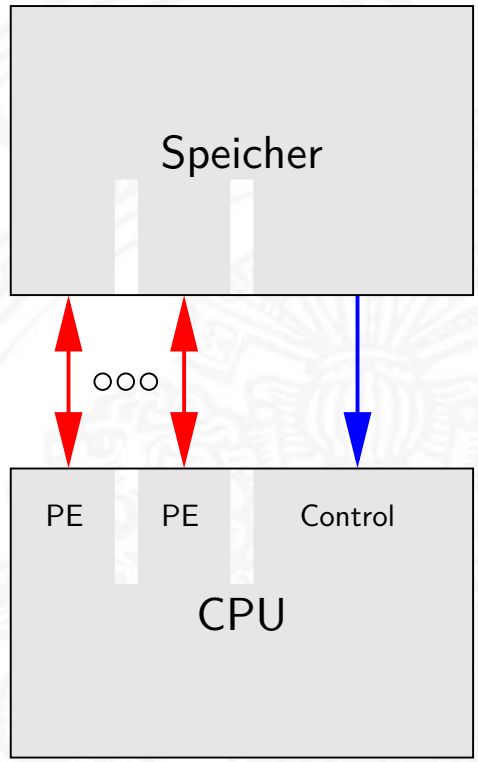
- von-Neumann / Princeton-Architektur
  ein gemeinsamer Bus für Instruktionen und Daten
- Harvard-Architektur
  Instruktionen und Daten haben *getrennte Busse*



Klassifikation (cont.)

SIMD **S**ingle **I**nstruction Stream, **M**ultiple **D**ata Streams

- Feldrechner
- Vektorrechner



Klassifikation (cont.)

Beispiele

- Mathematik: Vektor-Rechenoperationen
- Grafik: Pixel-Operationen
- Intel Streaming SIMD Extensions

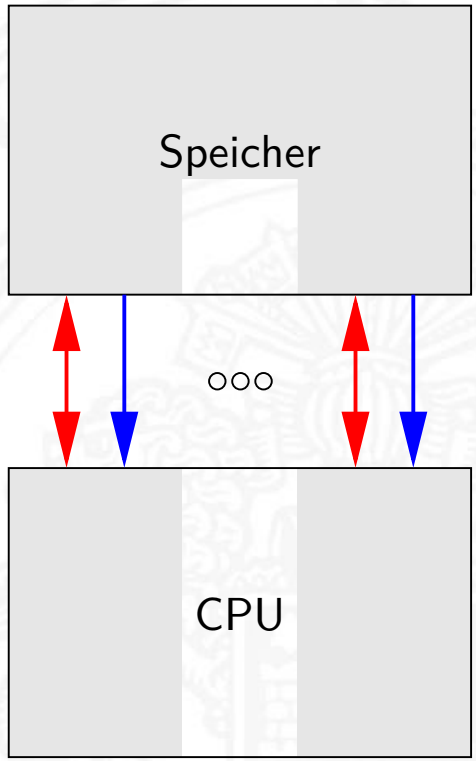
ADDSUBPS xmm1,xmm2/m128

xmm1	X4	X3	X2	X1
xmm2/ m128	Y4	Y3	Y2	Y1
xmm1	X4 + Y4	X3 - Y3	X2 + Y2	X1 - Y1

Klassifikation (cont.)

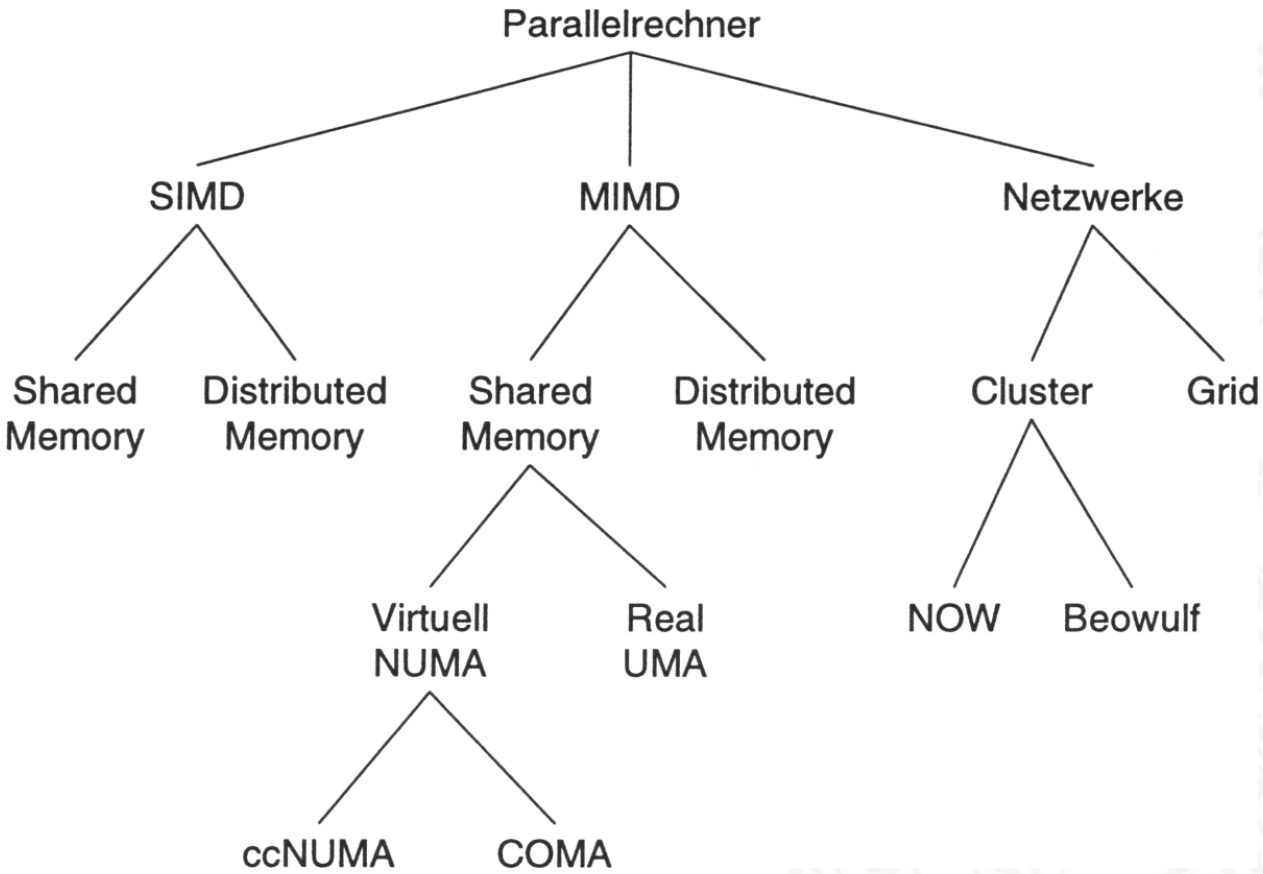
MIMD **M**ultiple **I**nstruction Streams, **M**ultiple **D**ata Streams

- Parallelrechner:
  CPUs/Prozessorelemente als Knoten
- Rechner-Cluster:
  Computer als Knoten



Klassifikation (cont.)

genauere Unterteilung folgt im Abschnitt über „Parallelrechner“



Klassifikation (cont.)

Allgemeine Klassifikationskriterien

- Operationsprinzip
- Struktur
- Leistung
- Einsatzgebiet
- Rechner- / Prozessorfamilien
- ...

Instruction Set Architecture

ISA – **I**nstruction **S**et **A**rchitecture

- Schnittstelle zur Hardware für Programmierer/Compiler
- Definiert durch
  - konzeptionelle Struktur
  - funktionales Verhalten

Compilerschnittstelle

- legt fest, was Compiler ausdrücken kann
- „Wortschatz des Compilers“
- definiert was Hardware ausführen können muss

Instruction Set Architecture (cont.)

Programmierer-Sicht des Computers

- Wie werden Daten und Datenstrukturen repräsentiert?
- Wo können Daten gespeichert werden?
- Wie kann auf die Daten zugegriffen werden?
- Welche Operationen können auf den Daten ausgeführt werden?
- Wie sind die Instruktionen kodiert?
- Wie ist die Organisation der (sichtbaren) Register?
- ...

Instruction Set Architecture (cont.)

Anwendungsprogrammierer / User-Level

- direkte I/O-Befehle nicht erlaubt
- kein Zugriff auf Speicherverwaltung (Segmentierung / Paging)
- kein Zugriff auf Interrupt-, Exception- und System-Call-Handling
- stattdessen *System-Calls*

(Betriebs-) Systemprogrammierer / System-Level

- direkte I/O-Befehle erlaubt
- Zugriff auf Segmentierung / Paging
- Zugriff auf Interrupt-, Exception- und System-Call-Handling
- System-Calls nicht möglich

Instruction Set Architecture (cont.)

Kompatibilität von ISA

- ▶ Erweiterung bestehender ISA (Bildung von Obermengen)
- \+ Abwärtskompatibilität
  Prozessorfamilien: Intel, Sparc, MIPS...
- \+ lange Software Lebensdauer
- – neue (bessere) Architekturen sind schwierig einzuführen

Inhalte einer ISA

- Syntax
  - Argumente der Befehle und Addressierungsarten
  - Codierung
- Semantik
  - Definition der Auswirkung bei Ausführung
- Daten
  - Binärdarstellung
  - Interpretation
  - Speicherung
- Befehle
  - Formate der Befehle
  - Befehlsklassen
  - Adressierungsarten

ISA – Daten

Datenworte

- Architekturabhängige Speicherwortlängen
-

Bit	0, 1	
Nibble	4 Bits	
Byte	8 Bits	
Half Word		16 Bits
Word	16 Bits	32 Bits
Long Word	32 Bits	64 Bits
Quad Word	64 Bits	
...		

ISA – Daten (cont.)

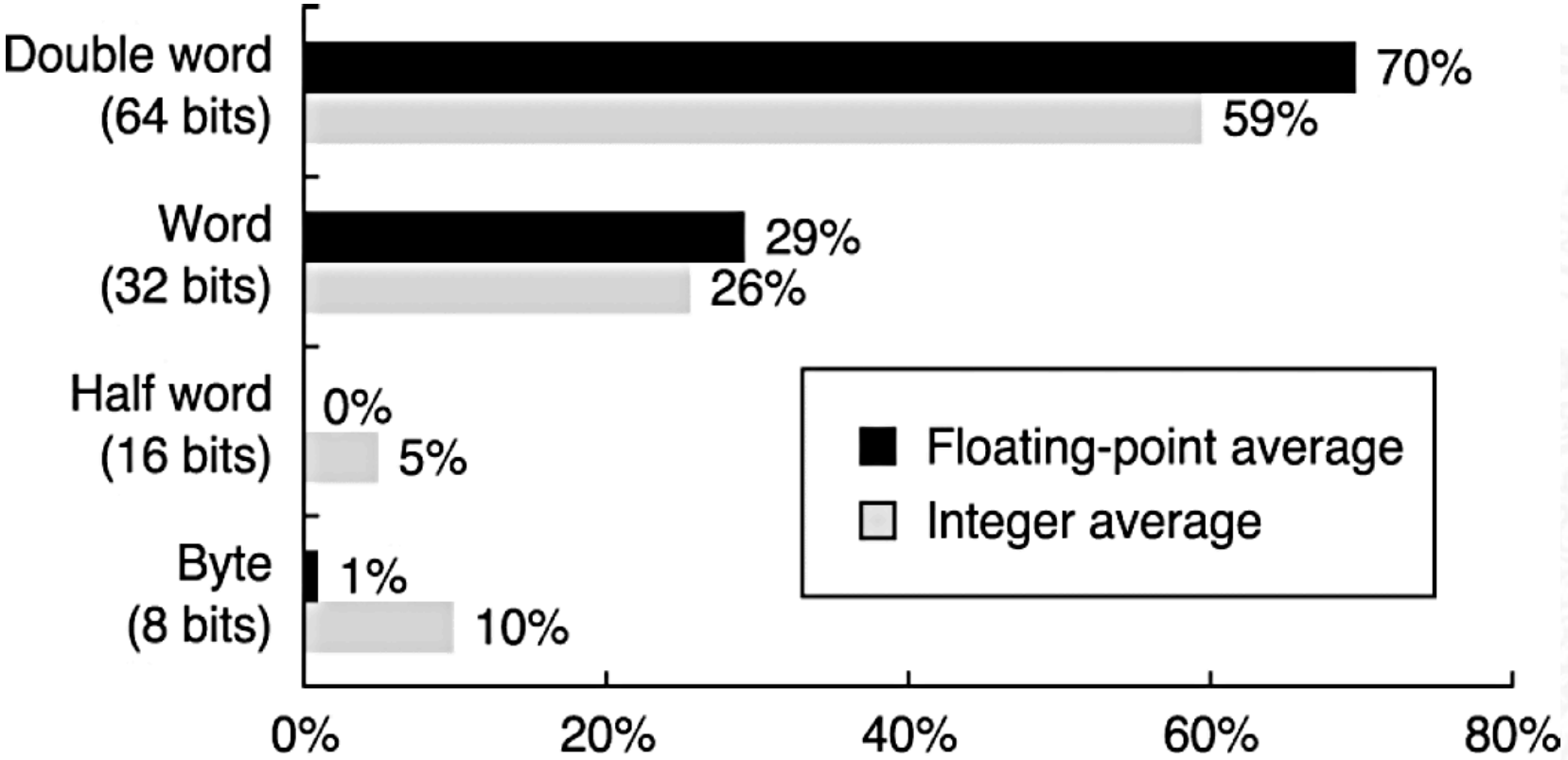
Datentypen

- Interpretation der Worte
Character	ASCII 7 Bit Code. . .
Decimal	BCD-Ziffern: paarweise in 8 Bits. . .
Integer	2er-Komplement. . .
Floating-Point	IEEE 754: Fliesskommastandard
	einfache, doppelte, erweiterte Genauigkeit. . .

  . . .

ISA – Daten (cont.)

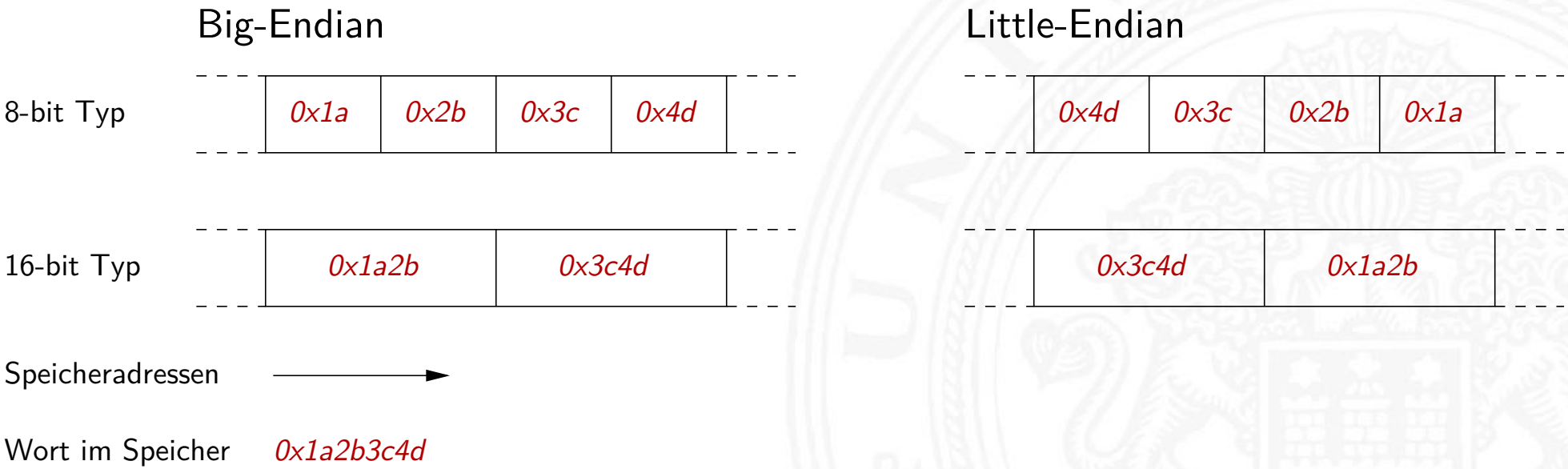
- Verteilung von Wortlängen und Datentypen



ISA – Daten (cont.)

Zuordnung der Datentypen zu Speicheradressen

- Hauptspeicher meist Byte-adressiert
  aber größere Wortbreite (z.B. 32-bit)
- Byte Reihenfolge / „*Endian*“



ISA – Befehle

Befehlsformate / Anzahl der Speicherworte

- variabel: befehlsabhängige Anzahl von Worten
  - CISC-Befehlssätze
  - \+ mächtige Befehle möglich
  - – komplexe Kontrollstruktur
  - – Pipelining schwierig
- fest: einheitliche Formatgröße
  - RISC-Befehlssätze
  - \+ einheitliches Konzept
  - \+ Befehlsabarbeitung sehr gut in Pipeline integrierbar

ISA – Befehle (cont.)

Befehlsklassen

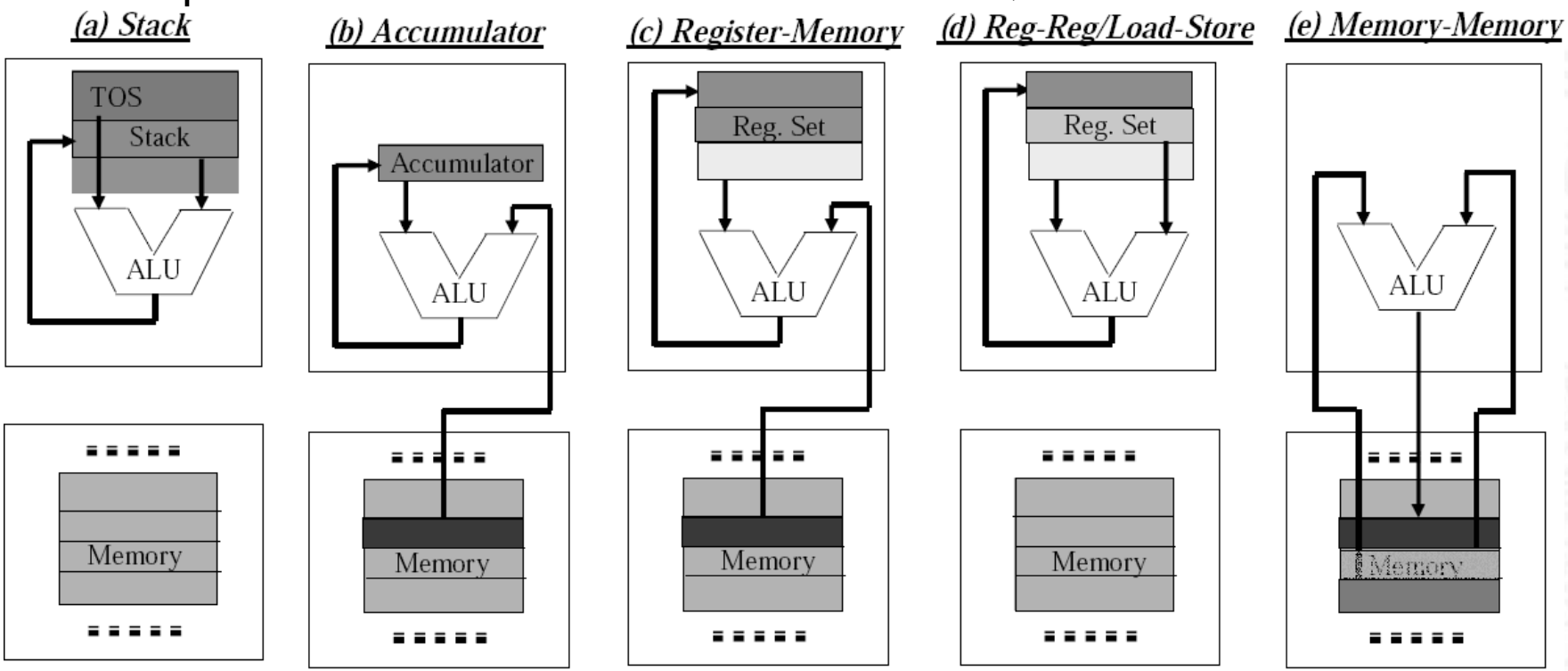
- Datentransfer: Speicher $\leftrightarrow$ Register, Ein-/Ausgabe
- arithmetische und logische Operationen
- Ablaufsteuerung: unbedingte/bedingte Sprünge, Unterprogrammaufrufe
- Systembefehle

Anzahl der Adressen im Befehl

- 0 Stack Operationen
  1 definiertes Akkumulator-Register und $\langle op1 \rangle$
  2 zwei Operanden: $\langle op1 \rangle$, $\langle op2 \rangle$
  3 drei Operanden: $\langle op1 \rangle$, $\langle op2 \rangle$, $\langle op3 \rangle$
- Operanden $\langle op \rangle$ jeweils Register oder Speicher

ISA – Befehle (cont.)

- Beispiel: 0- bis 3-Adress Befehle $C \leftarrow A + B$




```
Push A
Push B
Add
Pop C
```

```
Load  A
Add   B
Store C
```

```
Load  R1,A
Add   R1,B
Store R1,C
```

```
Load  R1,A
Load  R2,B
Add   R3,R1,R2
Store R3,C
```

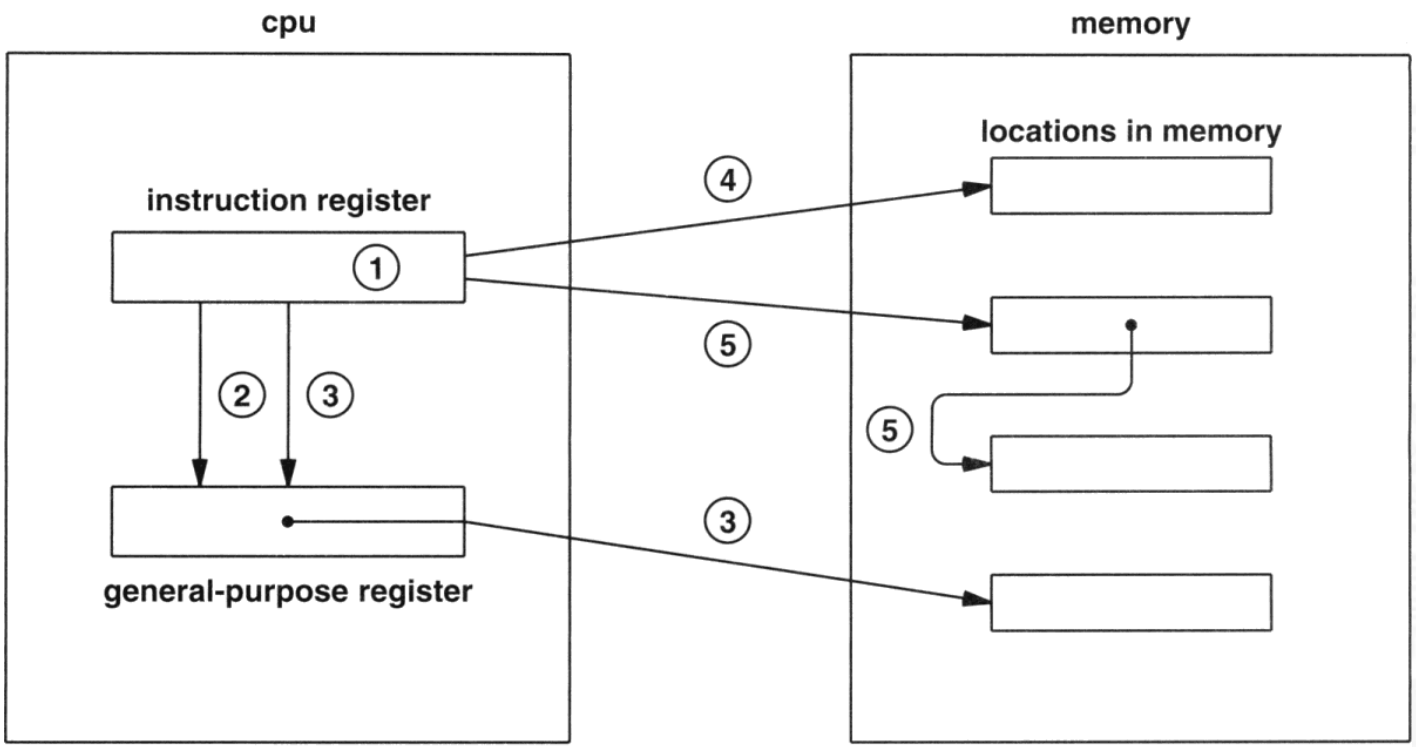
```
Add C,A,B
or Add A,B
```

ISA – Befehle (cont.)

Addressierungsarten

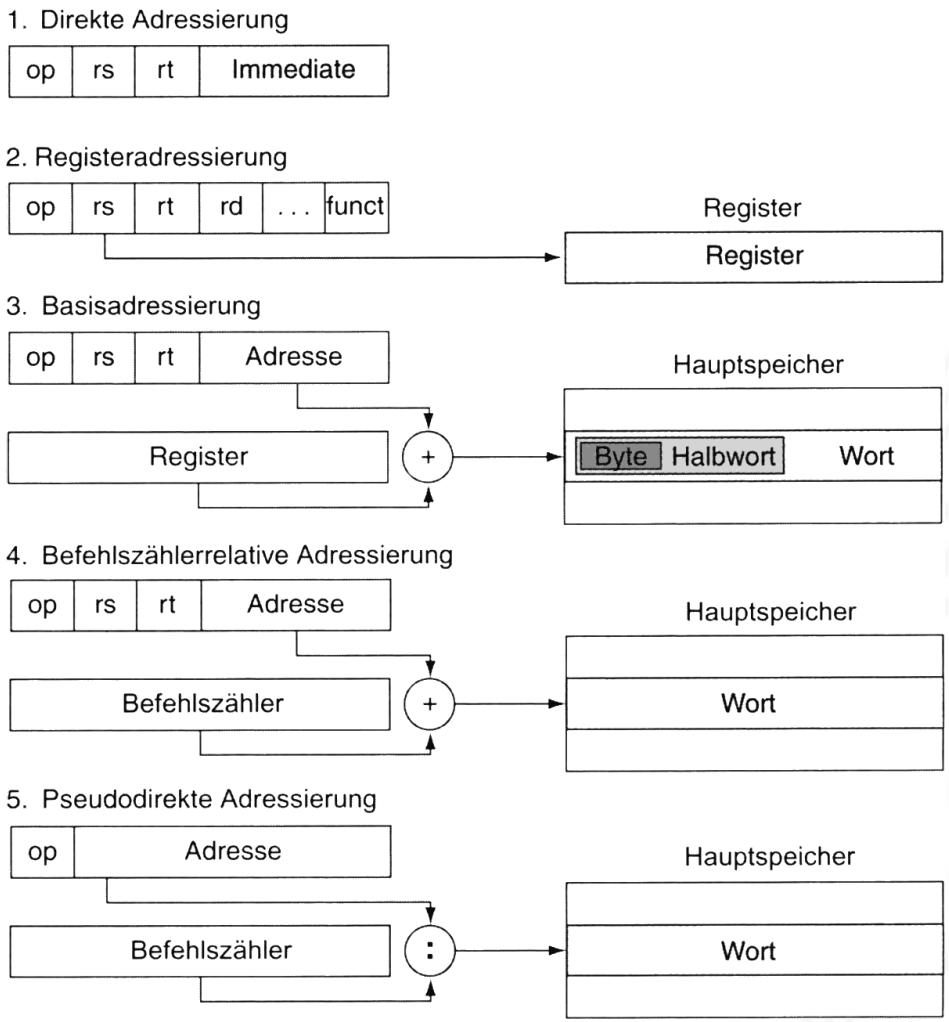
- Speicheradressierung
  - \+ entspricht Programmiermodell
  - – langsam
  - – Operandenlänge
- Registeradressierung
  - \+ Zugriffsgeschwindigkeit
  - – begrenzte Anzahl
  - – Verwaltung
- von grundlegender Architektur abhängig: RISC/CISC
- verschiedene Varianten
  - immediate Operand
  - Memory direct
  - Register direct
  - Indirect
  - Indirect with Offset / with Scale Factor
  - ...

ISA – Befehle (cont.)



1 Immediate value (in the instruction)
2 Direct register reference
3 Indirect through a register
4 Direct memory reference
5 Indirect memory reference

ISA – Befehle (cont.)



ISA – Befehle (cont.)

Ein-/Ausgabe

- direkter Zugriff auf E/A-Kanäle
  - \+ getrennte Adressräume für Speicher und E/A
  - – spezielle Instruktionen notwendig
- *Memory-mapped* Mechanismen
  - \+ Zugriff über „normale“ Load-/Store-Befehle
  - – gemeinsamer Adressraum

Bewertung der ISA

Kriterien für einen *guten* Befehlssatz

- vollständig: alle notwendigen Instruktionen verfügbar
- orthogonal: keine zwei Instruktionen leisten das Gleiche
- symmetrisch: z.B. Addition $\Leftrightarrow$ Subtraktion
- adäquat: technischer Aufwand entsprechend zum Nutzen
- effizient: kurze Ausführungszeiten

Statistiken zeigen: Dominanz der einfachen Instruktionen

Bewertung der ISA (cont.)

- x86-Prozessor

	Anweisung	Ausführungshäufigkeit %
1.	load	22 %
2.	conditional branch	20 %
3.	compare	16 %
4.	store	12 %
5.	add	8 %
6.	and	6 %
7.	sub	5 %
8.	move reg-reg	4 %
9.	call	1 %
10.	return	1 %
Total		96 %

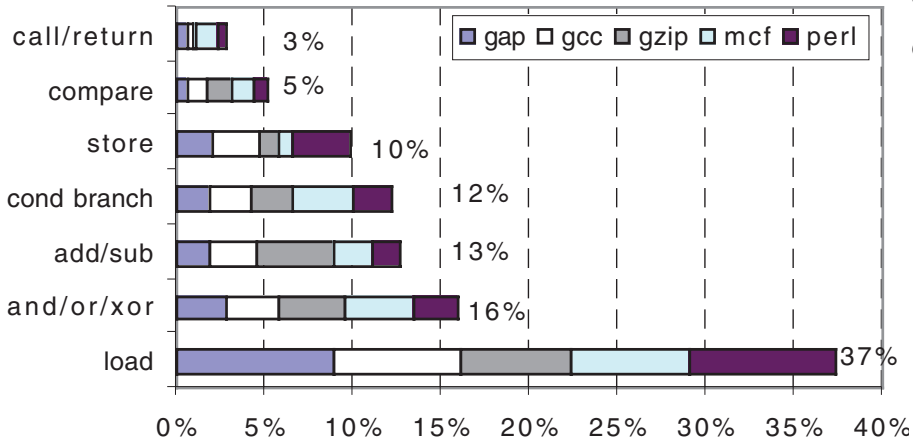
Bewertung der ISA (cont.)

Instruction	compress	eqntott	espresso	gcc (cc1)	li	Int. average
load	20.8%	18.5%	21.9%	24.9%	23.3%	22%
store	13.8%	3.2%	8.3%	16.6%	18.7%	12%
add	10.3%	8.8%	8.15%	7.6%	6.1%	8%
sub	7.0%	10.6%	3.5%	2.9%	3.6%	5%
mul				0.1%		0%
div						0%
compare	8.2%	27.7%	15.3%	13.5%	7.7%	16%
mov reg-reg	7.9%	0.6%	5.0%	4.2%	7.8%	4%
load imm	0.5%	0.2%	0.6%	0.4%		0%
cond. branch	15.5%	28.6%	18.9%	17.4%	15.4%	20%
uncond. branch	1.2%	0.2%	0.9%	2.2%	2.2%	1%
call	0.5%	0.4%	0.7%	1.5%	3.2%	1%
return, jmp indirect	0.5%	0.4%	0.7%	1.5%	3.2%	1%
shift	3.8%		2.5%	1.7%		1%
and	8.4%	1.0%	8.7%	4.5%	8.4%	6%
or	0.6%		2.7%	0.4%	0.4%	1%
other (xor, not, . . .)	0.9%		2.2%	0.1%		1%
load FP						0%
store FP						0%
add FP						0%
sub FP						0%
mul FP						0%
div FP						0%
compare FP						0%
mov reg-reg FP						0%
other (abs, sqrt, . . .)						0%

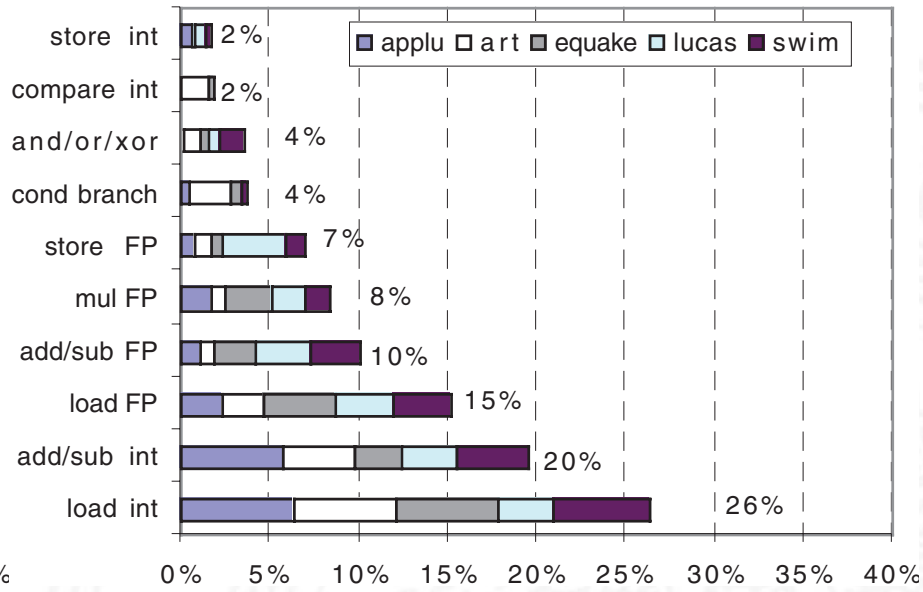
Figure D.15 80x86 instruction mix for five SPECint92 programs.

Bewertung der ISA (cont.)

- MIPS-Prozessor



SPECint2000 (96%)



SPECfp2000 (97%)

Bewertung der ISA (cont.)

- ca. 80 % der Berechnungen eines typischen Programms verwenden nur ca. 20 % der Instruktionen einer CPU
- am häufigsten gebrauchten Instruktionen sind einfache Instruktionen: load, store, add...

$\Rightarrow$ Motivation für RISC

CISC

CISC – **C**omplex **I**nstruction **S**et **C**omputer

- Motivation
  - aus der Zeit der ersten Großrechner, 60er Jahre
  - Programmierung auf Assemblerebene
  - Komplexität durch sehr viele (mächtige) Befehle umgehen
- Eigenschaften
  - Instruktionssätze mit mehreren hundert Befehlen (> 300)
  - sehr viele Adressierungsarten, -Kombinationen
  - verschiedene Instruktionsformate
  - viele Befehle können auf den Speicher zugreifen

CISC (cont.)

- Konsequenzen
  - \+ nah an der Programmiersprache, einfacher Assembler
  - \+ kompakter Code: weniger Befehle holen, kleiner I-Cache
  - – Pipelining schwierig
  - – Ausführungszeit abhängig von: Befehl, Adressmodi...
  - – Instruktion holen schwierig, da variables Instruktionsformat
  - – Speicherhierarchie schwer handhabbar: Adressmodi
- Mikroprogrammierung
  - ein Befehl kann nicht in einem Takt abgearbeitet werden

  $\Rightarrow$ Unterteilung in Mikroinstruktionen ($\varnothing$ 5...7)
  - Ablaufsteuerung durch endlichen Automaten
  - meist als ROM (RAM) implementiert, das *Mikroprogrammworte* beinhaltet

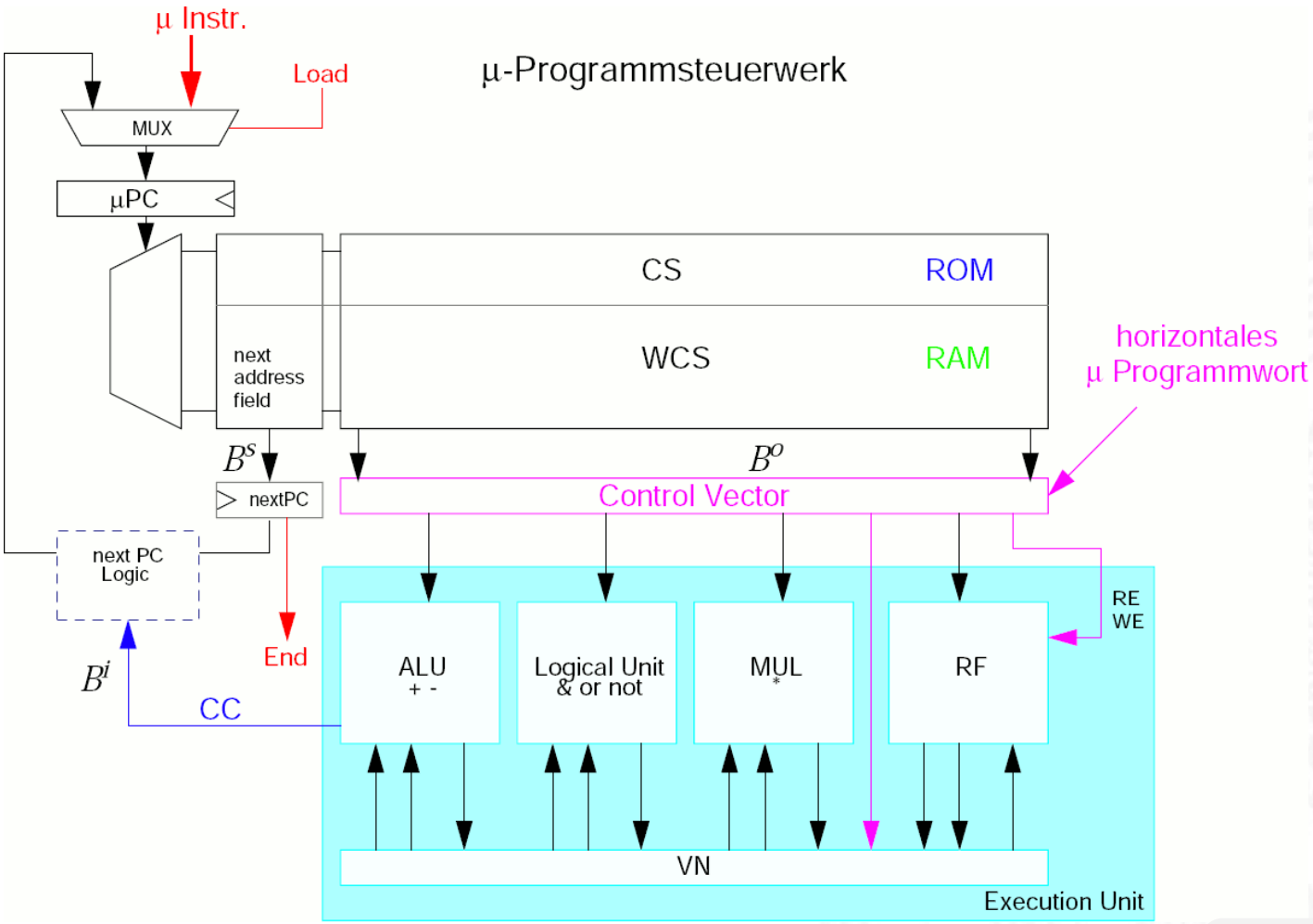
CISC (cont.)

1. horizontale Mikroprogrammierung
   - langes Mikroprogrammwort (ROM-Zeile)
   - steuert direkt alle Operationen
   - Spalten entsprechen: Kontrollleitungen und Folgeadressen
2. vertikale Mikroprogrammierung
   - kurze Mikroprogrammworte
   - Spalten enthalten Mikrooperationscode
   - mehrstufige Decodierung für Kontrollleitungen

\+ CISC-Befehlssatz mit wenigen Mikrobefehlen realisieren
\+ bei RAM: Mikrobefehlssatz austauschbar
– (mehrstufige) ROM/RAM Zugriffe: zeitaufwändig

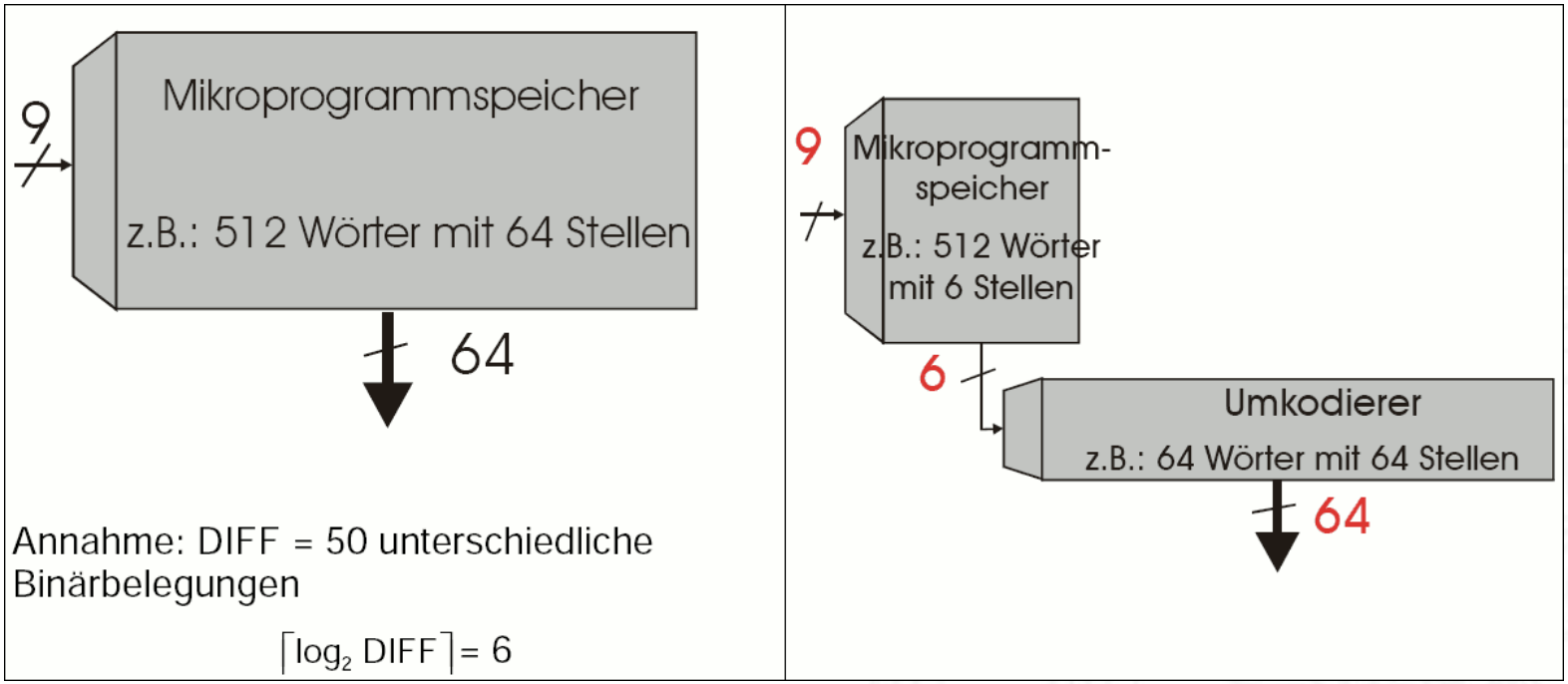
- ▸ horizontale Mikroprog. ▸ vertikale Mikroprog.

horizontale Mikroprogrammierung



◂ Mikroprogrammierung

vertikale Mikroprogrammierung



◂ Mikroprogrammierung

RISC

RISC – **R**educed **I**nstruction **S**et **C**omputer

- Grundidee: Komplexitätsreduktion in der CPU
- Historie
  - seit den 80er Jahren: „RISC-Boom“
  - wegen Hochsprachen und Compilereinsatz besteht kein Bedarf mehr für mächtige Assemblerbefehle
  - wegen schnellem Speicher und der Speicherhierarchie muss nicht mehr „möglichst viel“ lokal in der CPU gerechnet werden (CISC, mikroprogrammiert)
- Eigenschaften
  - reduzierte Anzahl der Instruktionen (z.B. 128)
  - nur Load- und Store-Instruktionen können auf Speicher zugreifen
  - alle anderen Operationen arbeiten auf Registern

RISC (cont.)

- Befehlsformate der meisten RISC ISA
  - 3-Adress-Instruktionen
  - Instruktionen in einem Wort (32 Bits) codiert
- Konsequenzen
  - \+ fest-verdrahtete Logik, kein Mikroprogramm
  - \+ einfache Instruktionen, wenige Adressierungsarten
  - \+ Cycles per Instruction = 1
  - \+ Pipelines
  - – längerer Maschinencode
  - – viele Register notwendig
  - optimierende Compiler nötig / möglich
  - High-performance Speicherhierarchie

Instruction Level Parallelism

ILP – **I**nstruction **L**evel **P**arallelism

- Instruktionen in einer *Pipeline* überlappend ausführen
- Instruktionen auf verschiedene Einheiten verteilen
  - zur Laufzeit: Superskalare Prozessoren
  - statisch durch den Compiler: VLIW Prozessoren

\+ viele Optimierungsmöglichkeiten: Loop unrolling, pipeline scheduling, scoreboarding, register renaming, branch prediction, multiple instruction issue, instruction reordering, software pipelining, trace scheduling, speculative execution...

Einige werden im folgenden Abschnitt über Pipelining ab Folie 508 genauer behandelt

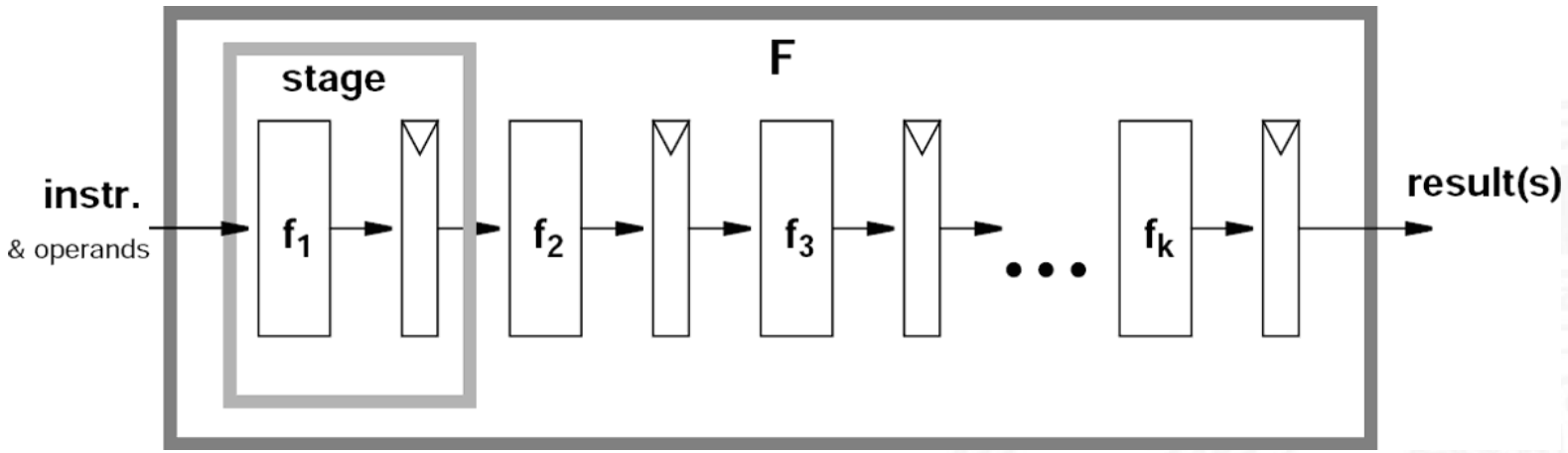
Mikroarchitektur

ISA als *Verhaltensbeschreibung*

Mikroarchitektur als *Strukturbeschreibung*

- funktionale Unterteilung: Daten- und Befehlsprozessor
- Hardwareeinheiten
  - Register und Register-Files
  - Arithmetische Einheiten: Integer-, Fliesskomma-ALUs ...
  - Datenpfade: Busse, Multiplexer, Demultiplexer ...
- Steuerung des Ablaufs = Implementierung der ISA
  - CISC: Mikroprogramm
  - RISC: Steuerung der Pipeline durch Kontrollautomat(en)

Pipelining / Fließbandverarbeitung

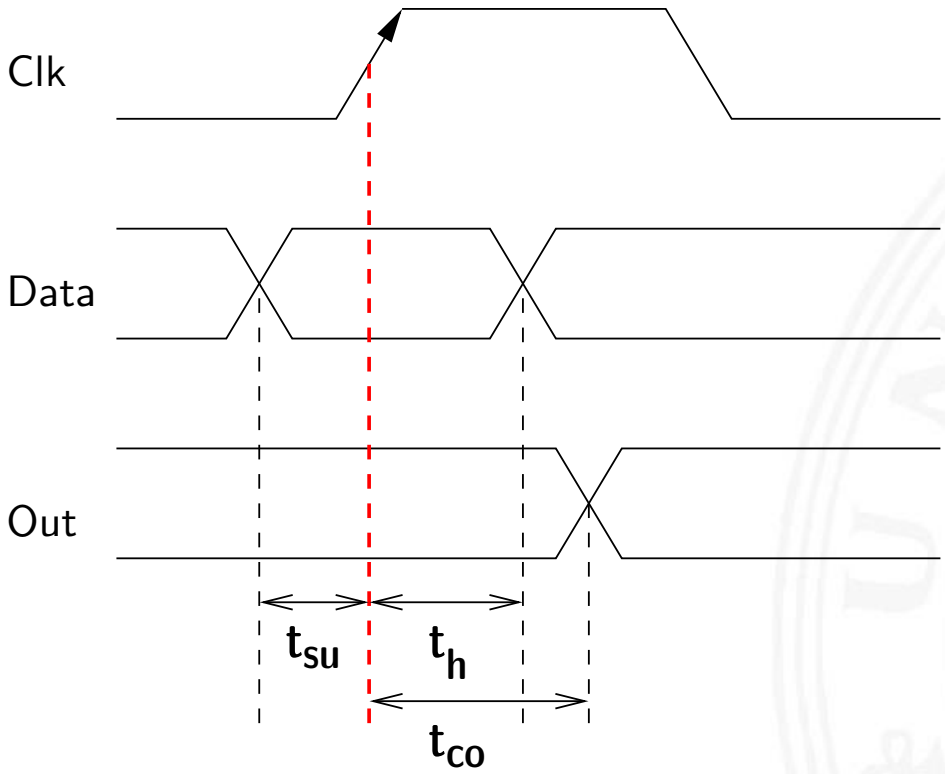


Grundidee

- Operation F kann in Teilschritte zerlegt werden
- jeder Teilschritt f_i braucht ähnlich viel Zeit
- alle Teilschritte f_i können parallel zueinander ausgeführt werden
- Trennung der Pipelinestufen („*stage*“) durch Register

Pipelining / Fließbandverarbeitung (cont.)

- Verglichen mit der Zugriffszeit auf die Register (t_{co}) dauert der Teilschritt f_i lang



Pipelining / Fließbandverarbeitung (cont.)

Arithmetische Pipelines

- Idee: lange Berechnung in Teilschritte zerlegen
  wichtig bei komplizierteren arithmetischen Operationen
  - die sonst sehr lange dauern (weil ein großes Schaltnetz)
  - die als Schaltnetz extrem viel Hardwareaufwand erfordern
  - Beispiele: Multiplikation, Division, Fließkommaoperationen...
- \+ Erhöhung des Durchsatzes, wenn Berechnung mehrfach hintereinander ausgeführt wird

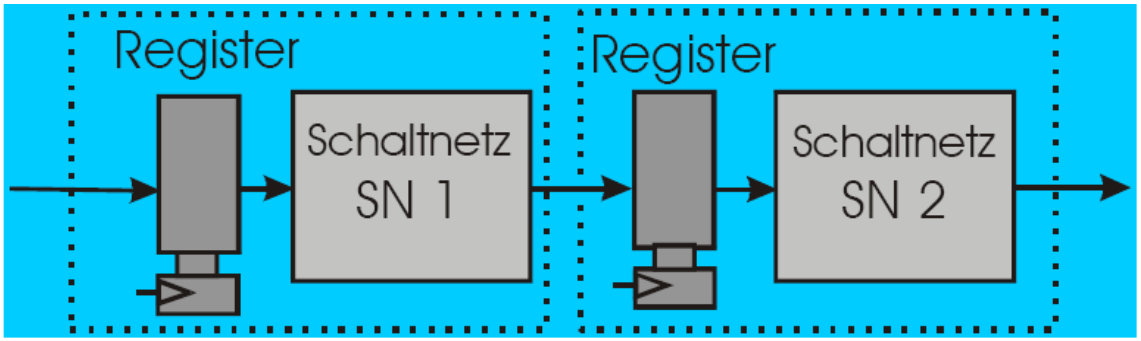
(RISC) Prozessorpipelines

- Idee: die Phasen der von-Neumann Befehlsabarbeitung (Befehl holen, Befehl decodieren ...) in eine Pipeline integrieren
- weiteres folgt ab Folie 523

Analyse von Pipelines

- Reservierungstabelle
  - zweidimensionale Matrix: RT
  - $RT[i,j] = a$ Pipelinestufe i ist zur relativen Taktzeit j mit Teilauftrag a beschäftigt
- Reservierungsmuster
  - Belegung der Reservierungstabelle durch die Teilaufträge
  - Rückführungen: Pipelinestufen i werden zu aufeinanderfolgenden Zeitpunkten j, $j+1$ belegt
- Ablauftabelle
  - zweidimensionale Matrix: AT
  - $AT[i,j] = k, a$ Pipelinestufe i ist zur relativen Taktzeit j mit Teilauftrag a von Auftrag k beschäftigt
  - Belegung der Pipeline durch aufeinanderfolgende Aufträge

Analyse von Pipelines (cont.)



Reservierungstabelle:

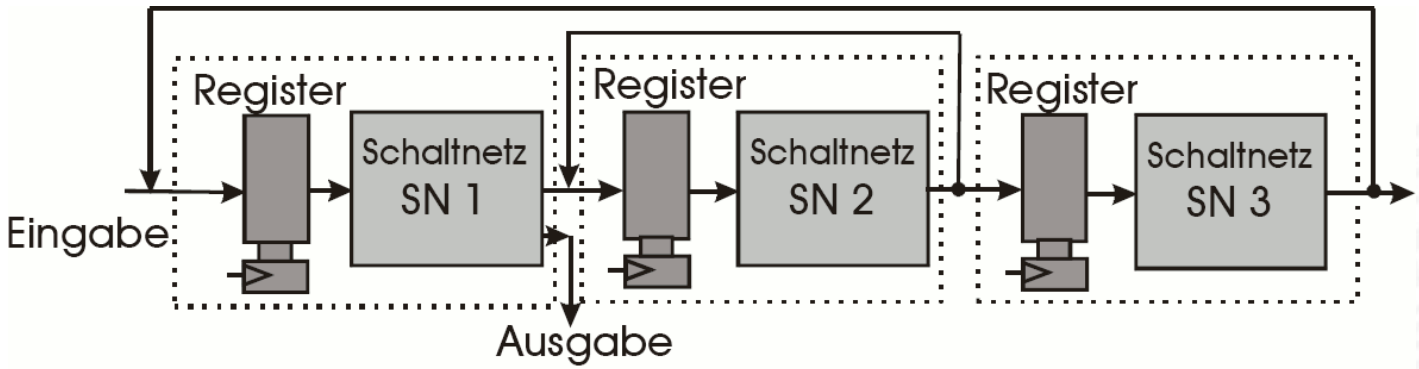
Stufe	Relative Taktzeit 0	1
1	*a*	
2		*b*

Ablauftabelle:

Stufe	Absolute Taktzeit 0	1	2	3	4	5	6
1	*1a*	*2a*	*3a*	*4a*	*5a*	*6a*	*7a*
2		*1b*	*2b*	*3b*	*4b*	*5b*	*6b*

Auftrag 6, Teilauftrag b

Analyse von Pipelines (cont.)



Relative Taktzeit

Stufe	0	1	2	3	4
1	*a*				*e*
2		*b*	*c*		
3				*d*	

Absolute Taktzeit

Stufe	0	1	2	3	4	5	6
1	*1a*		*2a*		*1e*	*3a*	*2e*
2		*1b*	*1c*	*2b*	*2c*		*3b*
3				*1d*		*2d*	

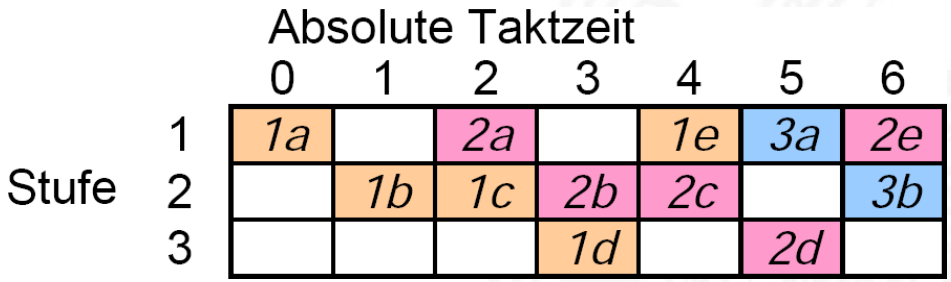
Analyse von Pipelines (cont.)

Wie kann die Pipeline optimal ausgenutzt werden?

= Möglichst gute Füllung der Ablauftabelle

- Latenzzeit: Zeitlicher Abstand zwischen zwei aufeinander folgenden Aufträgen, die konfliktfrei bearbeitet werden können

  Beispiel: Startzeiten 0 2 5 7 10...
  Latenzzeiten 2 3 2 3 2 ...



Füllungsabhängige Latenzzeit
= 2, wenn 1 Auftrag in Pipeline
= 3, wenn 2 Aufträge –"–

Analyse von Pipelines (cont.)

- Latenzfolge: Tupel aus minimalen Latenzzeiten, periodisch
  Latenzzyklus: kleinste Latenzfolge
  Beispiel: Latenzfolge (2, 3, 2, 3...)
  Latenzzyklus (2, 3)
- mittlere Latenzzeit mLZ für einen Latenzzyklus mit m Elementen

$$mLZ = 1/m \sum_{i=1}^{m} lat_i$$

Beispiel: $mLZ = 2,5$

Analyse von Pipelines (cont.)

- Kollisionsvektor: aus Reservierungsmuster ermitteln, welche Abstände Kollision, bzw. keine Kollision bewirken
  - gegeben $RT[i,j]$ mit $1 \leq i \leq s$ Stufen, $0 \leq j \leq m$ Takte
  - gesucht $KV[0 \ldots m] \epsilon \{0,1\}^{m+1}$

$$KV[i] = \begin{cases} 1 & \text{Initialisierungsabstand } i \text{ erzeugt Konflikt} \\ 0 & \text{sonst} \end{cases}$$

  - Initialisierung $KV[0] = 1, KV[j] = 0 \quad \forall j : 1 \leq j \leq m$
  - Berechnung $KV[j_2 - j_1] = 1 \quad \forall j_1, j_2 : 1 \leq j_1 < j_2 \leq m$
    falls $\exists i : 1 \leq i \leq s$ mit $RT[i, j_1] = RT[i, j_2] = 1$

Liste verbotener Abstände: $VL \subseteq 0, 1, \ldots m$

Analyse von Pipelines (cont.)

		Relative Taktzeit					$j_2 - j_1$	
		0	1	2	3	4		
Stufe i	1	*a*				*e*	4 – 0 = 4	KV[4]=1
	2		*b*	*c*			2 – 1 = 1	KV[1]=1
	3				*d*			

Beispiel: Kollisionsvektor $KV = (1\ 1\ 0\ 0\ 1)$

verbotene Abstände $VL = \{0, 1, 4\}$

Initialisierungsabstände $\{2, 3\}$

Analyse von Pipelines (cont.)

- aus dem Kollisionsvektor kann dann ein Verfahren entwickelt werden, wie statischer Fließbänder optimal zu füllen sind
  1. Kollisionsvektor zu der Reservierungstabelle ermitteln
  2. Erstellung eines Übergangsgraphen zwischen Kollisionsvektoren Zyklen des Graphen entsprechen Latenzzyklen
  3. Für alle einfachen Zyklen werden die zugehörigen mittleren Latenzzeiten aus dem Graphen berechnet

  $\Rightarrow$ Der optimale Schedule ergibt sich dann aus dem Zyklus mit der kleinsten mittleren Latenzzeit, bzw. einem Pfad zu diesem Zyklus

. . . aus Zeitgründen nicht weiter beschrieben

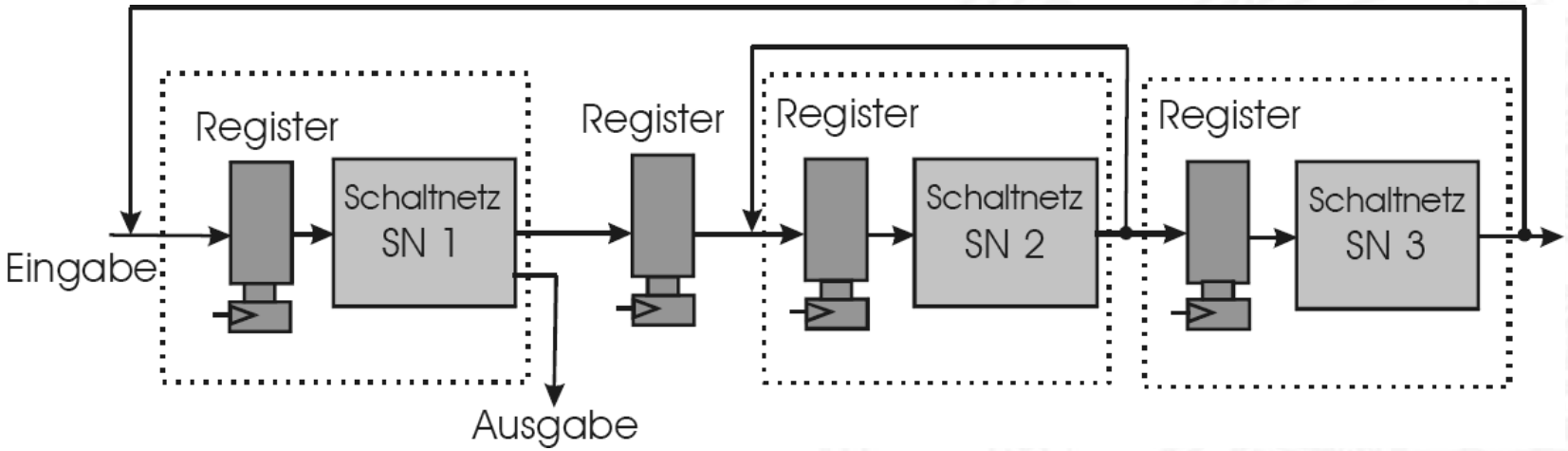
Statische- und dynamische Fließbandverarbeitung

- statische Fließbandverarbeitung:
  es gibt nur eine Sorte von Aufträgen
- dynamische Fließbandverarbeitung:
  mehrere Aufträge / Reservierungsmuster
  - Kollisionen können nicht mehr allgemein bestimmt werden, sondern nur für vorgegebene Auftragsfolgen
  - Liste verbotener Abstände für alle Folgen von Aufträgen (also *zwischen* allen Reservierungsmustern) ermitteln

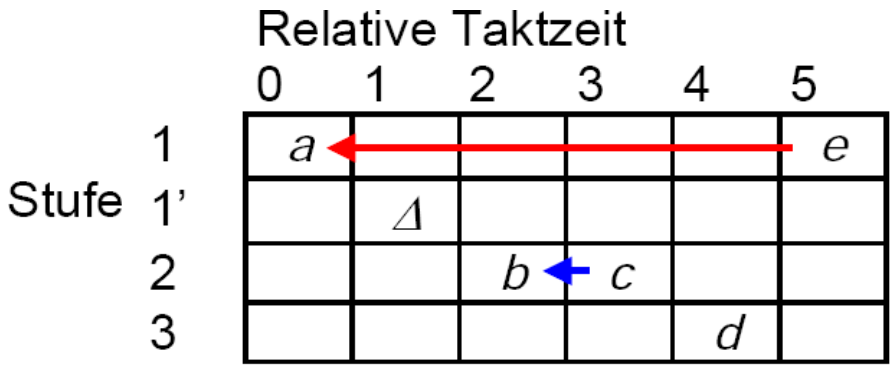
Fließbandoptimierungen

Einfügen zusätzlicher Register

- anderer Schedule
- anderer Kollisionsvektor
- bessere Pipelineauslastung

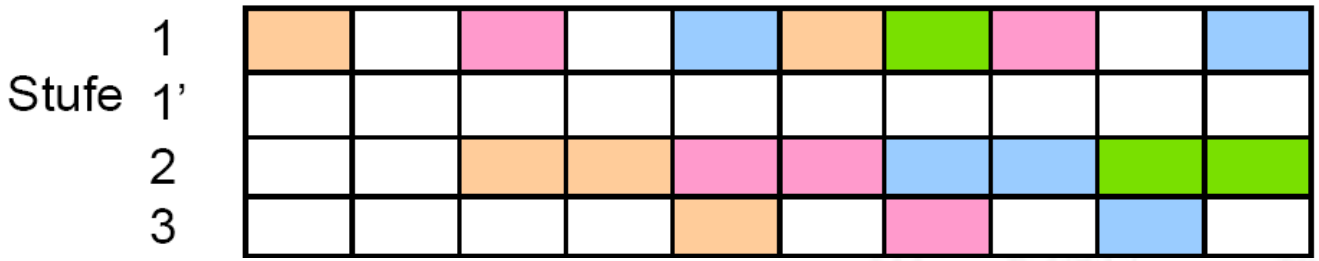


Fließbandoptimierungen (cont.)



Kollisionsvektor:

1 1 0 0 0 1



Fließbandoptimierungen (cont.)

Reservierungstabelle mit Freiheitsgraden

- Operationen können „nach vorne“ oder „nach hinten“ verschoben werden
- anderer Kollisionsvektor ...

Zu allen Optimierungsmethoden gibt es konstruktive Verfahren, die eine bessere Pipelineauslastung erzeugen

... aus Zeitgründen nicht weiter beschrieben

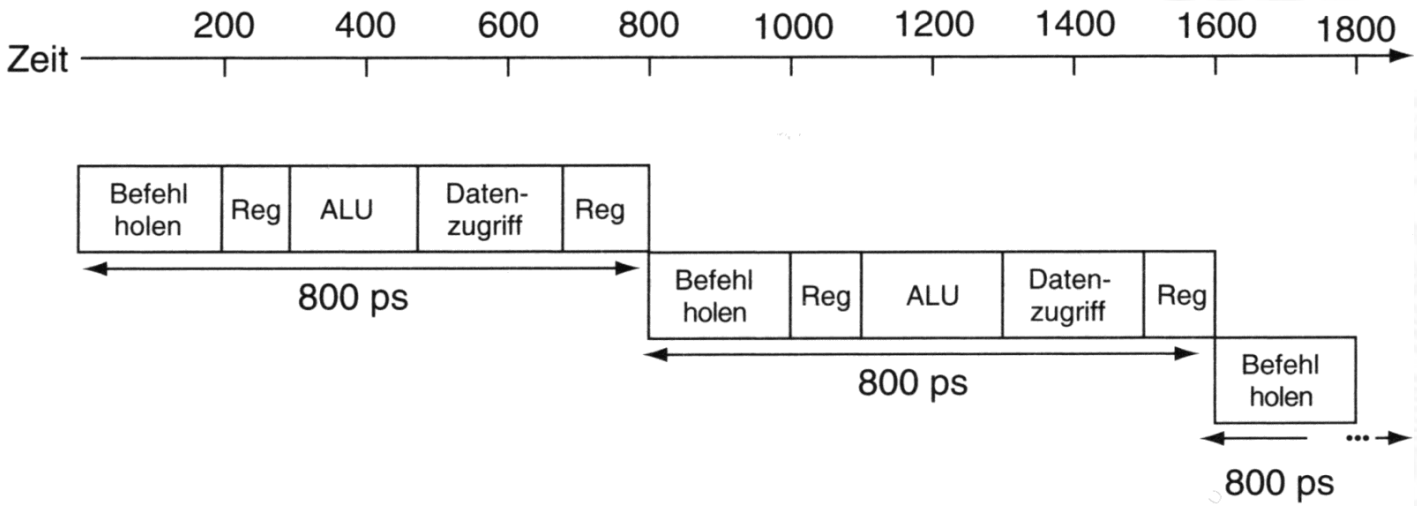
RISC Pipelining

Schritte der RISC Befehlsabarbeitung (von ISA abhängig)

IF	**I**nstruction **F**etch Instruktion holen, in Befehlsregister laden
ID	**I**nstruction **D**ecode Instruktion decodieren
OF	**O**perand **F**etch Operanden aus Registern holen
EX	**Ex**ecute ALU führt Befehl aus
MEM	**Mem**ory access Speicherzugriff bei Load-/Store-Befehlen
WB	**W**rite **B**ack Ergebnisse in Register zurückschreiben

RISC Pipelining (cont.)

- je nach Instruktion sind 3-5 dieser Schritte notwendig
- Beispiel *ohne* Pipelining:

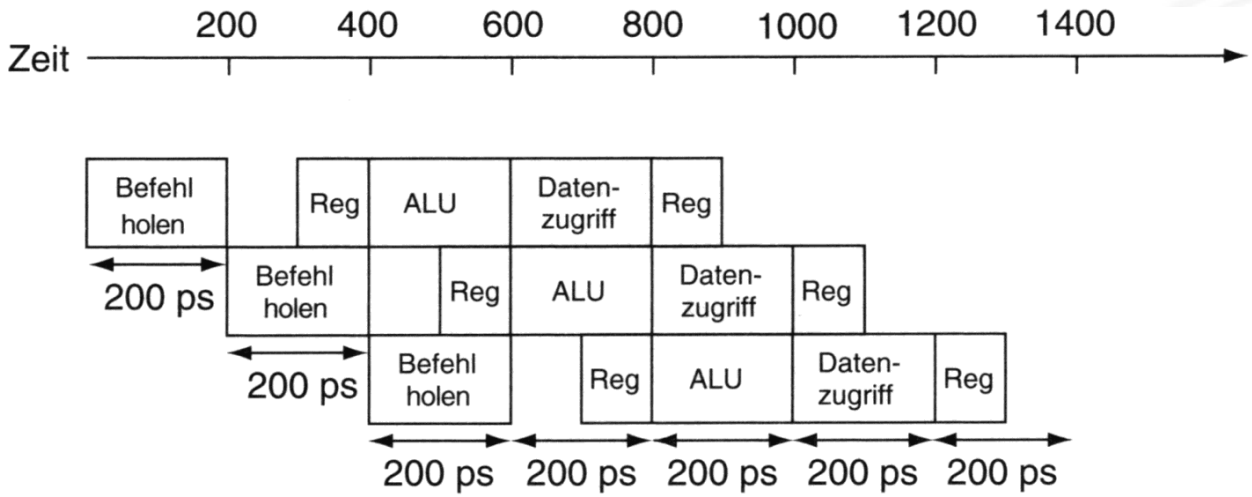


[PH05]

RISC Pipelining (cont.)

Pipelining in Prozessoren

- Beispiel *mit* Pipelining:



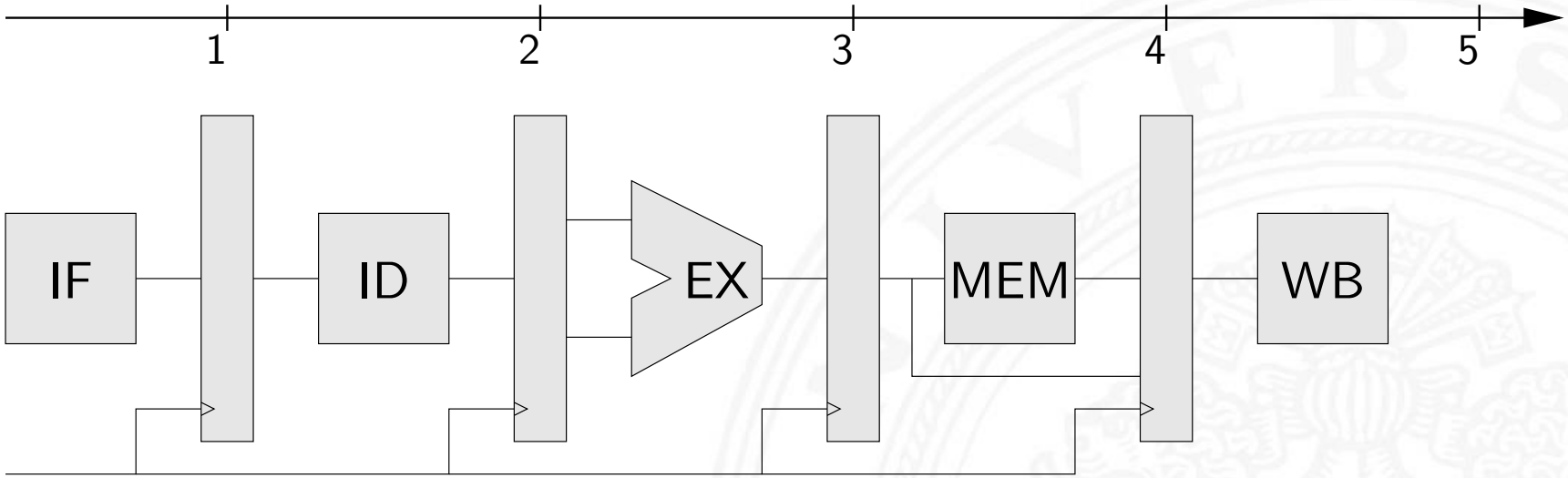
[PH05]

- Befehle überlappend ausführen
- Register trennen Pipelinestufen

RISC Pipelining (cont.)

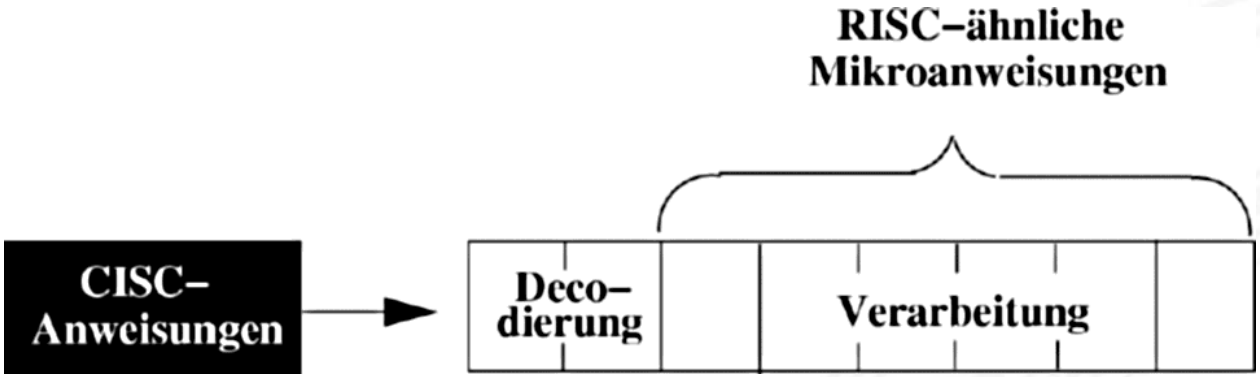
- RISC ISA: Pipelining wird direkt umgesetzt

Pipelinestufen



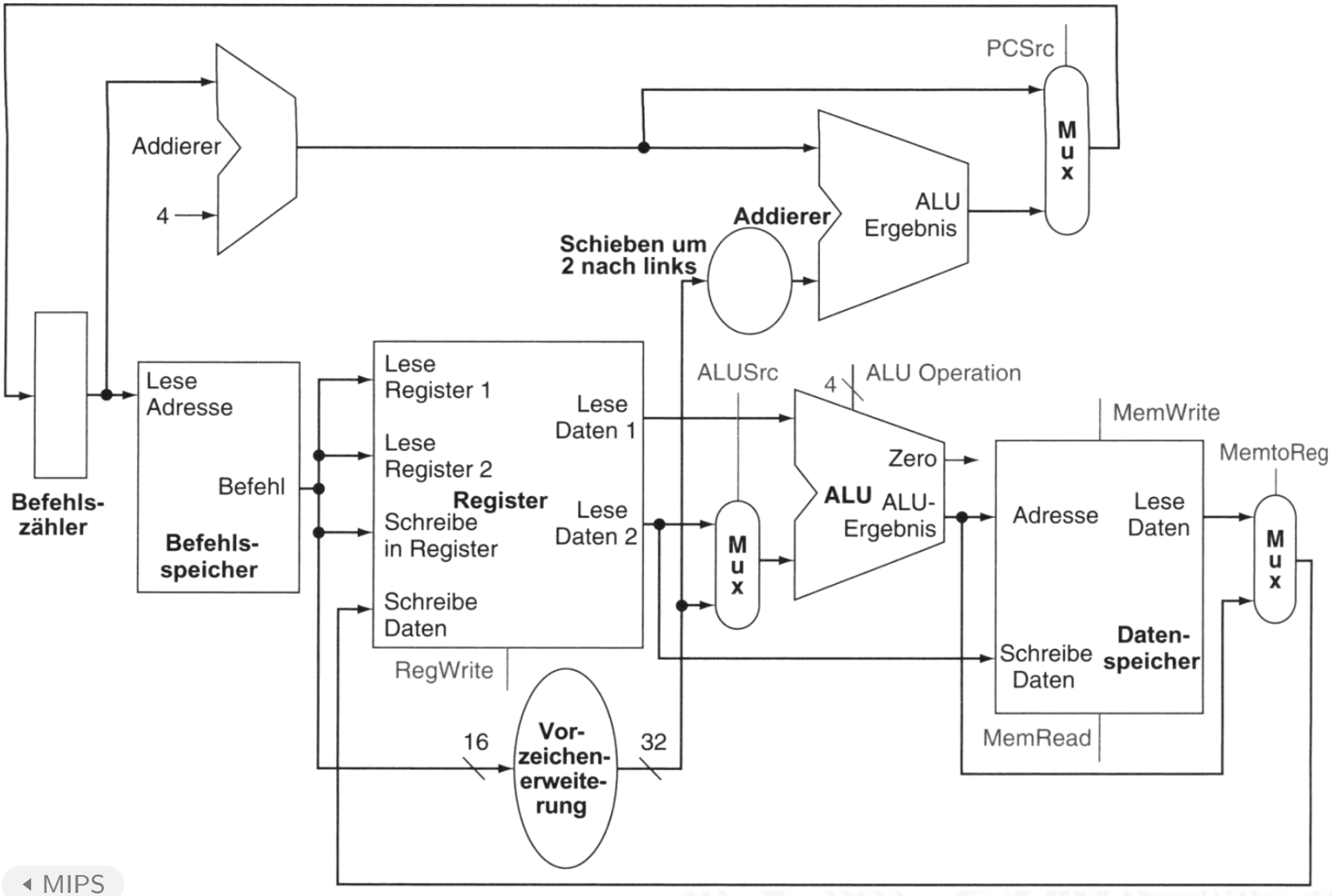
RISC Pipelining (cont.)

- CISC ISA: Umsetzung der CISC Befehle in Folgen RISC-ähnlicher Anweisungen

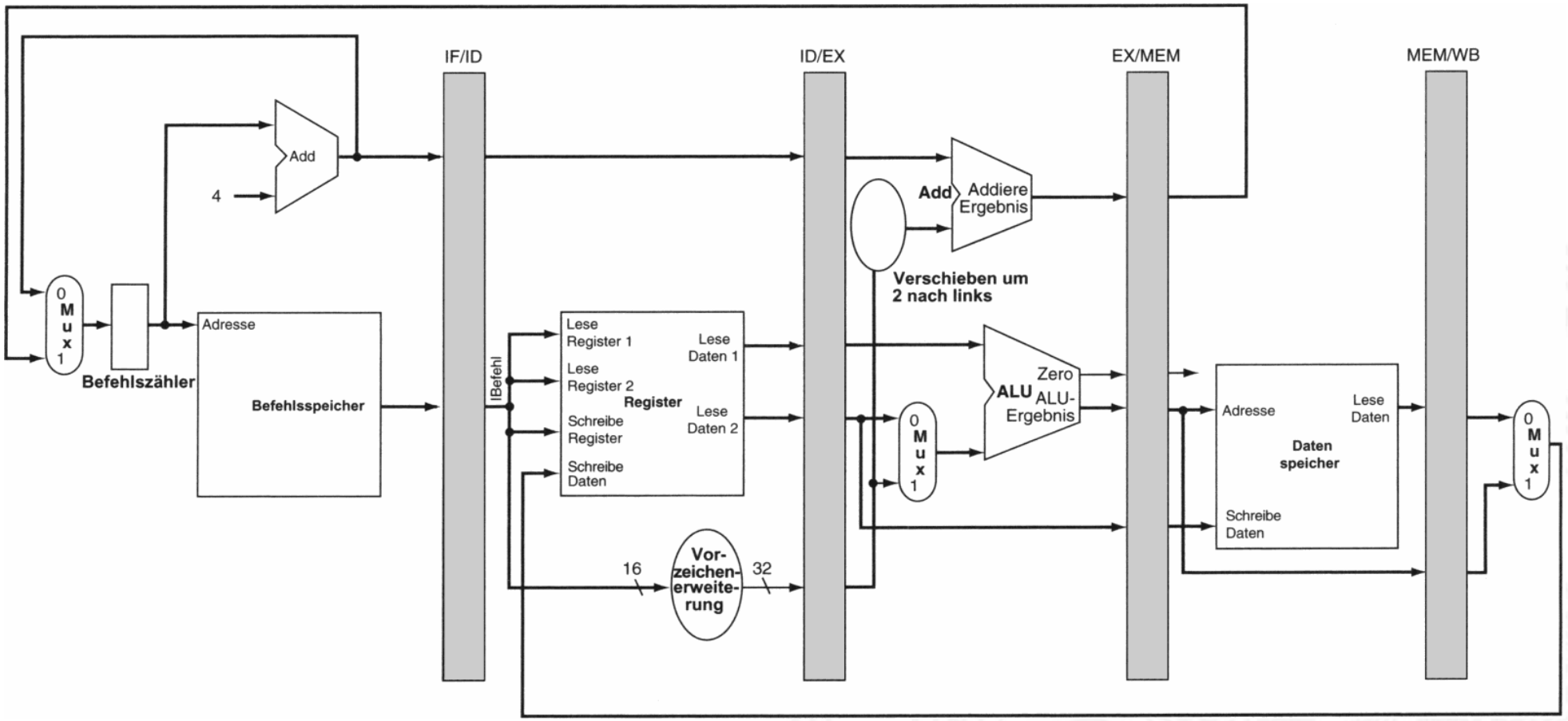


  - \+ CISC-Software bleibt lauffähig
  - \+ Befehlssatz wird um neue RISC Befehle erweitert
- Beispiele: MIPS-Architektur (aus Patterson, Hennessy [PH05])

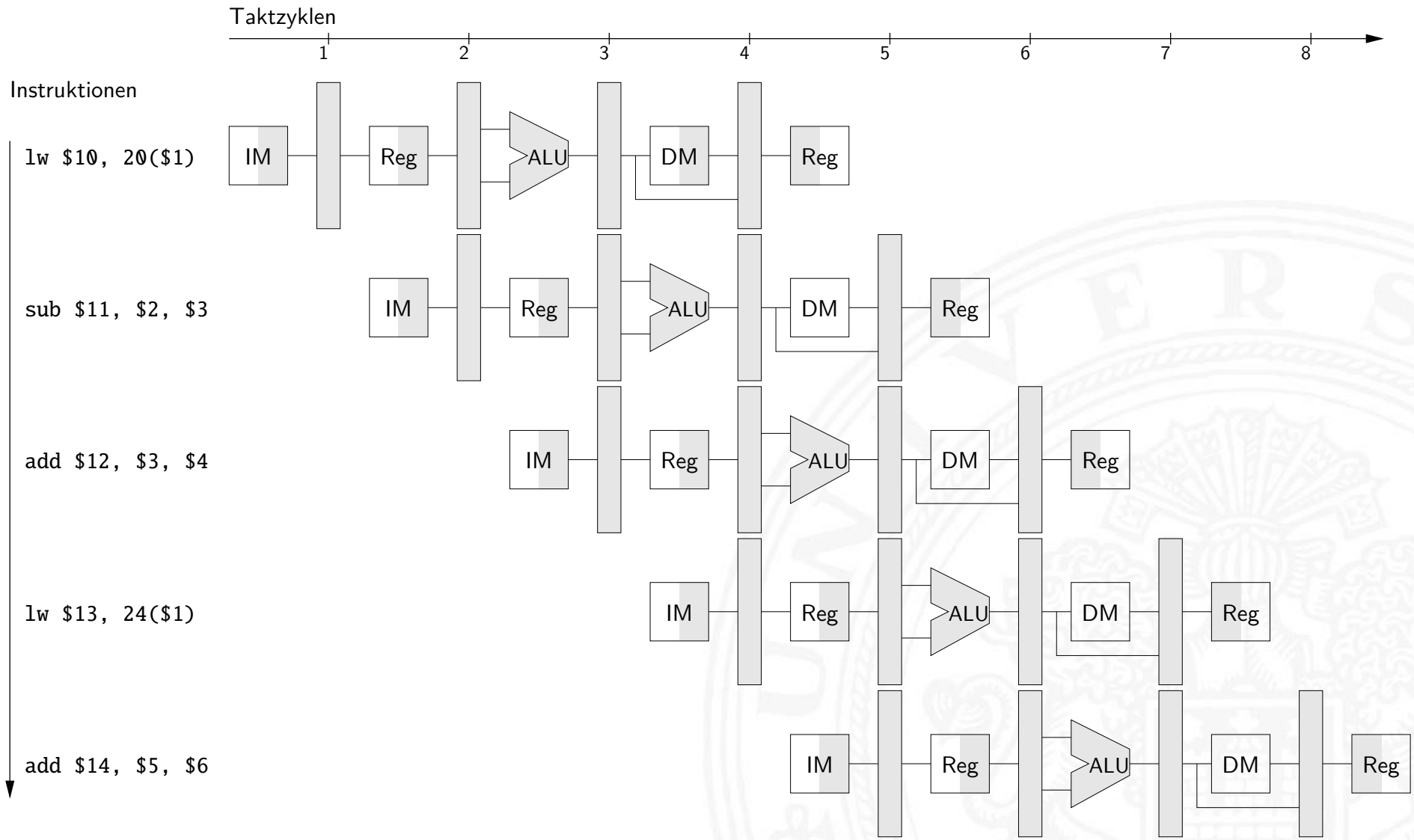
▸ MIPS ohne Pipeline ▸ MIPS Pipeline ▸ Pipeline Schema


PCSrc
Addierer
4
Addierer
Schieben um
2 nach links
ALU
Ergebnis
M
u
x
Befehls-
zähler
Lese
Adresse
Befehl
Befehls-
speicher
Lese
Register 1
Lese
Register 2
Register
Schreibe
in Register
Schreibe
Daten
Lese
Daten 1
Lese
Daten 2
RegWrite
ALUSrc
4
ALU Operation
Zero
ALU
ALU-
Ergebnis
MemWrite
MemtoReg
Adresse
Lese
Daten
Schreibe
Daten
Daten-
speicher
MemRead
16
Vor-
zeichen-
erweite-
rung
32

◂ MIPS


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
Befehlszähler
Adresse
Befehlsspeicher
IBefehl
Lese Register 1
Lese Register 2
Register
Schreibe Register
Schreibe Daten
Lese Daten 1
Lese Daten 2
16
Vor-zeichen-erweite-rung
32
Add
Addiere Ergebnis
Verschieben um 2 nach links
ALU
Zero
ALU-Ergebnis
Adresse
Lese Daten
Daten speicher
Schreibe Daten

◂ MIPS


Taktzyklen
1
2
3
4
5
6
7
8
Instruktionen
lw $10, 20($1)
sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6
IM
Reg
ALU
DM
Reg

◂ MIPS

Prozessorpipeline – Begriffe

Begriffe

- Pipeline-Stage: einzelne Stufe der Pipeline
- Pipeline Machine Cycle: Instruktion kommt einen Schritt in Pipeline weiter
- Durchsatz: Anzahl der Instruktionen, die in jedem Takt abgeschlossen werden
- Latenz: Zeit, die eine Instruktion benötigt, um alle Pipelinestufen zu durchlaufen

Prozessorpipeline – Bewertung

Vor- und Nachteile

+ Pipelining ist für den Programmierer nicht sichtbar!
+ höherer Instruktionsdurchsatz $\Rightarrow$ bessere Performanz
– Latenz wird nicht verbessert, bleibt bestenfalls gleich
– Pipeline Takt limitiert durch langsamste Pipelinestufe
  unausgewogene Pipelinestufen reduzieren den Takt und damit die Performanz
– zusätzliche Zeiten, um Pipeline zu füllen bzw. zu leeren

Prozessorpipeline – Speed-Up

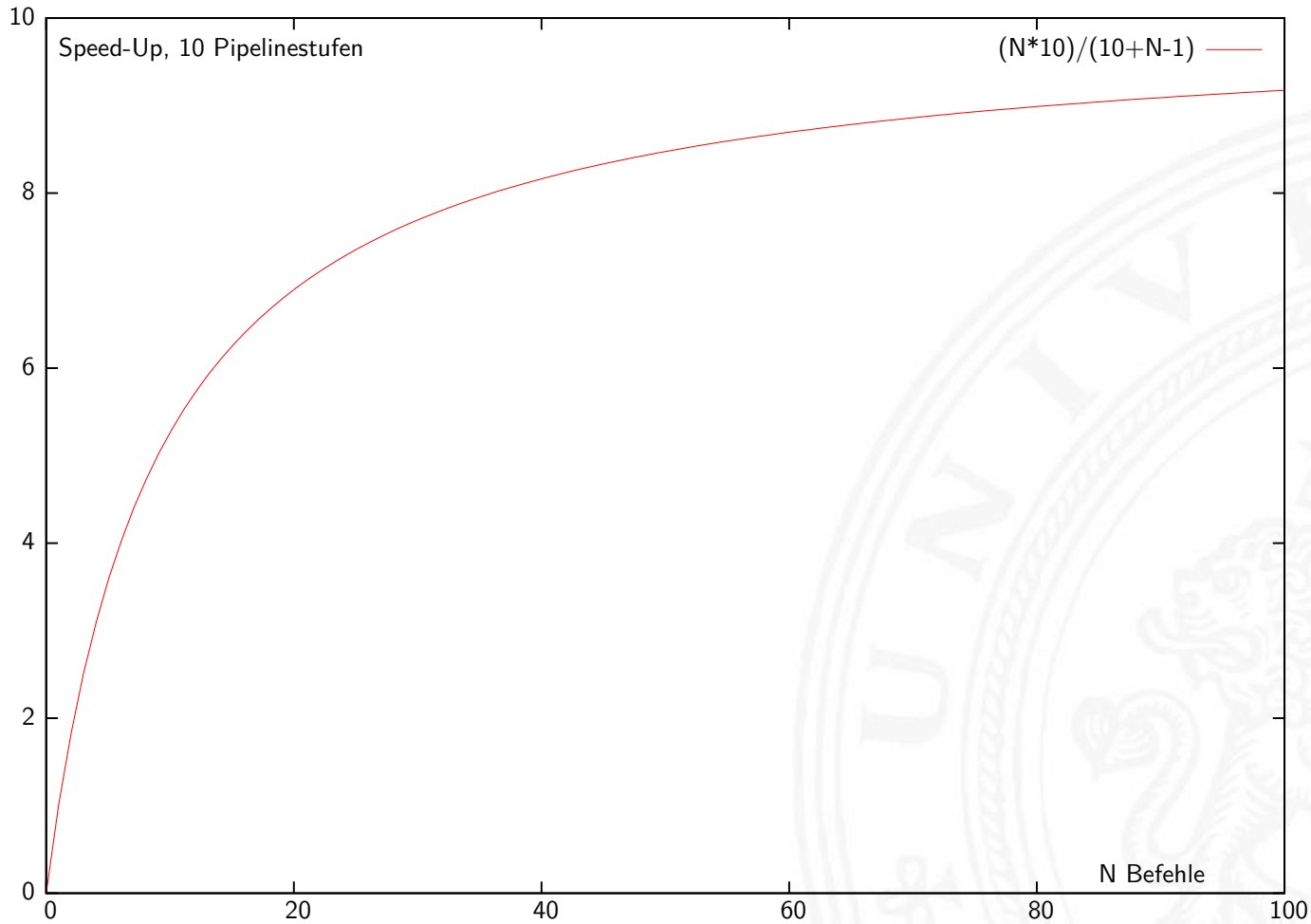
Pipeline Speed-Up

- N Instruktionen; K Pipelinestufen
- ohne Pipeline: $N \cdot K$ Taktzyklen
- mit Pipeline: $K + N - 1$ Taktzyklen
- Speed-Up $= \frac{N \cdot K}{K+N-1}$, $\lim_{N \to \infty} S = K$

$\Rightarrow$ ein großer Speed-Up wird erreicht durch

1. große Pipelinetiefe: K
2. lange Instruktionssequenzen: N

Prozessorpipeline – Speed-Up (cont.)



Prozessorpipeline – Dimensionierung

Dimensionierung der Pipeline

- Längere Pipelines
- Pipelinestufen in den Einheiten / den ALUs (*superskalar*)

$\Rightarrow$ größeres K wirkt sich direkt auf den Durchsatz aus

$\Rightarrow$ weniger Logik zwischen den Registern, höhere Taktfrequenzen

- Beispiele

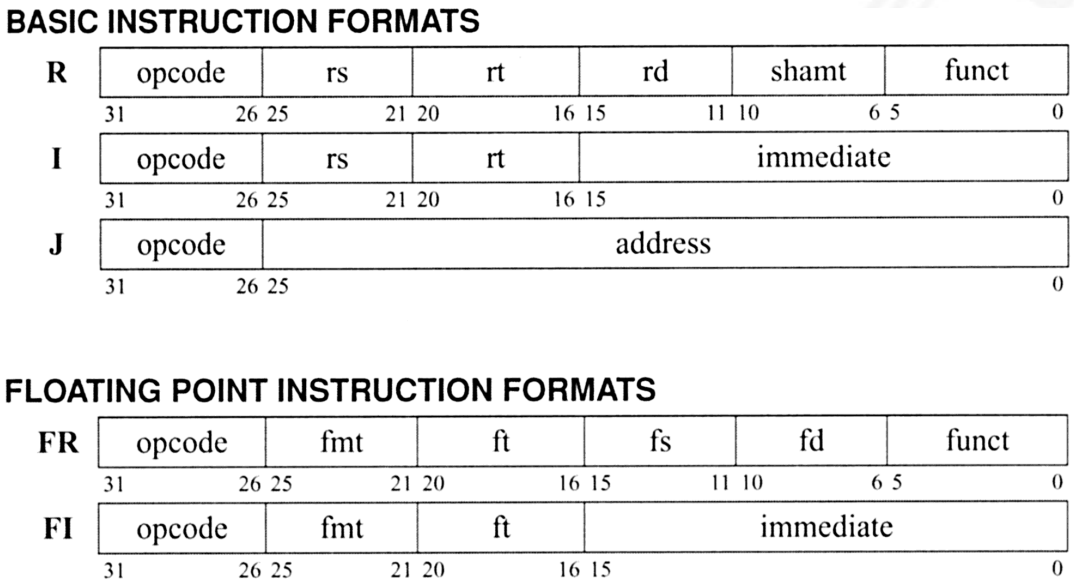
CPU	Pipelinestufen	Taktfrequenz [*MHz*]
Pentium	5	300
Motorola G4	4	500
Motorola G4e	7	1000
Pentium II/III	12	1400
Athlon XP	10/15	2500
Athlon 64, Opteron	12/17	$\leq$ 3000
Pentium 4	20	$\leq$ 5000

Prozessorpipeline – Auswirkungen

Architekturentscheidungen, die sich auf das Pipelining auswirken

gut für Pipelining

- gleiche Instruktionslänge
- wenige Instruktionsformate
- Load/Store Architektur



MIPS-Befehlsformate

Prozessorpipeline – Auswirkungen (cont.)

schlecht für Pipelining: *Pipelinekonflikte / -Hazards*

- Strukturkonflikt: gleichzeitiger Zugriff auf eine Ressource durch mehrere Pipelinestufen
- Datenkonflikt: Ergebnisse von Instruktionen werden innerhalb der Pipeline benötigt
- Steuerkonflikt: Sprungbefehle in der Pipelinesequenz

sehr schlecht für Pipelining

- Unterbrechung des Programmkontexts: Interrupt, System-Call, Exception...
- (Performanz-) Optimierungen mit „Out-of-Order-Execution“ etc.

Pipeline Strukturkonflikte

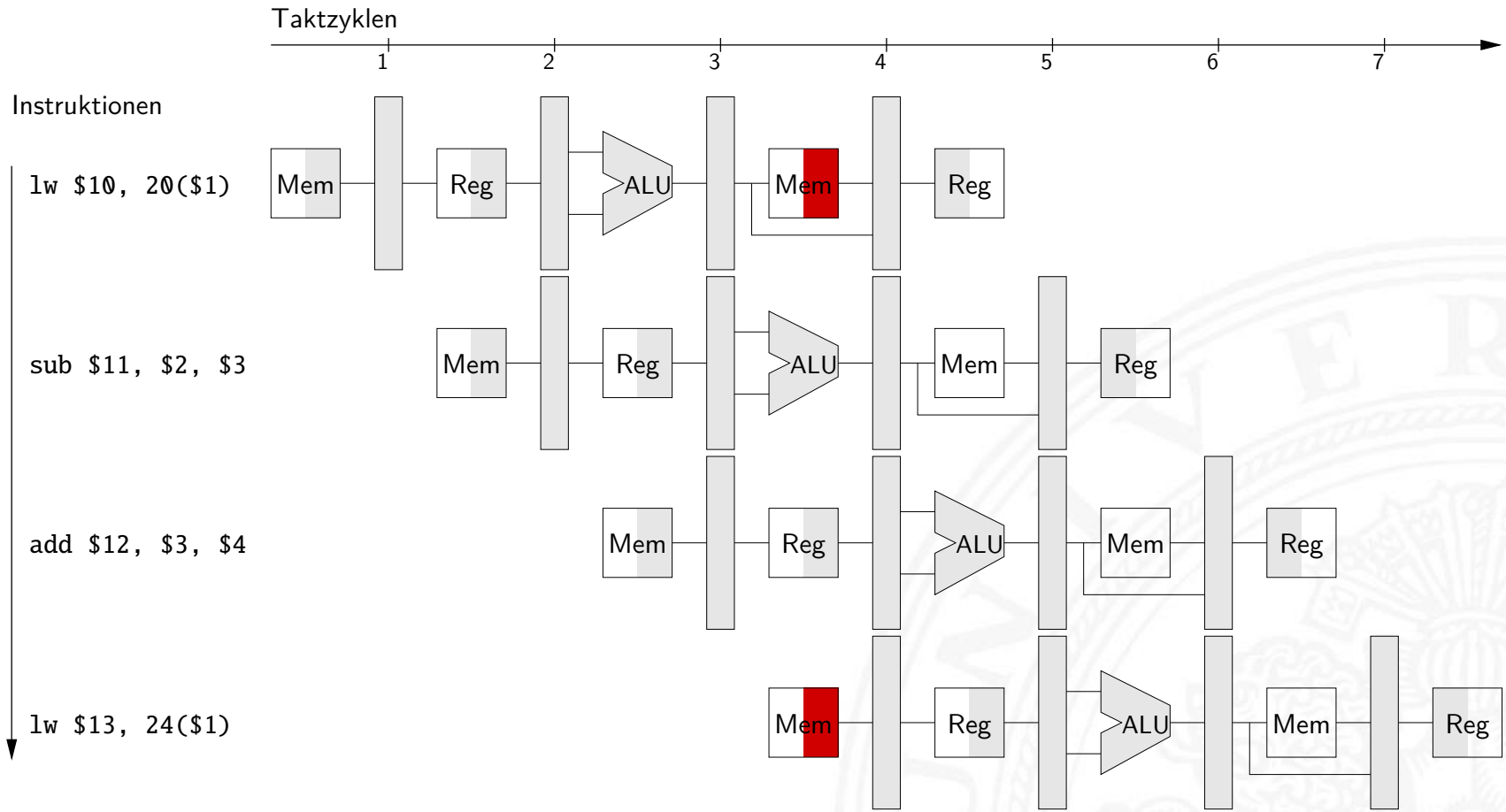
Strukturkonflikt / Structural Hazard

- mehrere Stufen wollen gleichzeitig auf eine Ressource zugreifen
- Beispiel: gleichzeitiger Zugriff auf Speicher

▸ Beispiel

⇒ Mehrfachauslegung der betreffenden Ressourcen

- Harvard-Architektur vermeidet Strukturkonflikt aus Beispiel
- Multi-Port Register
- mehrfach vorhandene Busse und Multiplexer...



◂ Strukturkonflikte

Pipeline Datenkonflikte

Datenkonflikt / Data Hazard

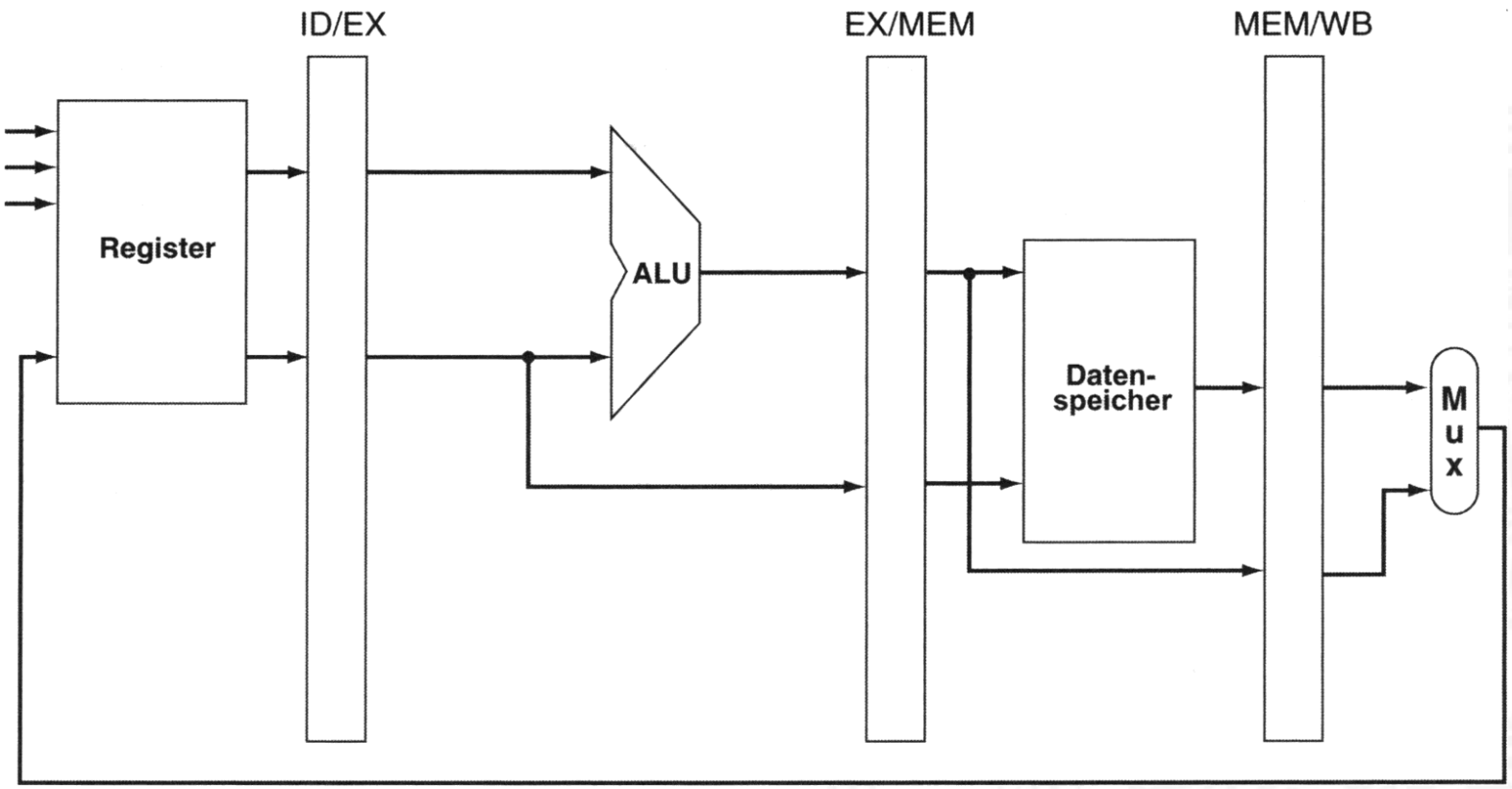
- eine Instruktion braucht die Ergebnisse einer vorhergehenden, diese wird aber noch in der Pipeline bearbeitet
- Datenabhängigkeiten der Stufe „Befehl ausführen“

▸ Beispiel

Forwarding

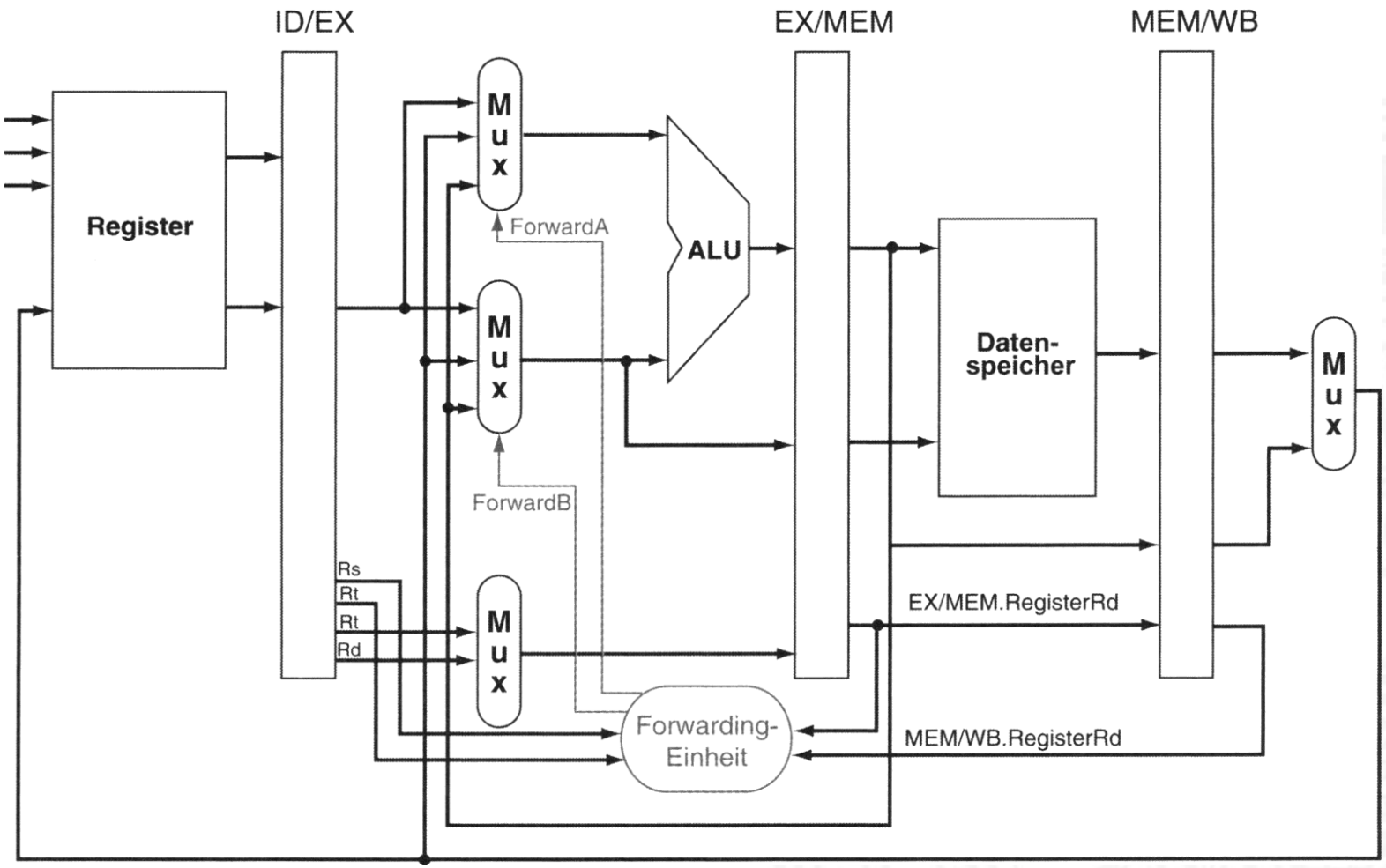
- kann Datenabhängigkeiten auflösen, s. Beispiel
- extra Hardware: „*Forwarding-Unit*“
- Änderungen in der Pipeline Steuerung
- neue Datenpfade und Multiplexer

Pipeline Datenkonflikte (cont.)



MIPS
Datenpfad

Pipeline Datenkonflikte (cont.)



MIPS Forwarding

Pipeline Datenkonflikte (cont.)

Rückwärtsabhängigkeiten

- spezielle Datenabhängigkeit ▸ Beispiel
- Forwarding-Technik funktioniert nicht, da die Daten erst *später* zur Verfügung stehen
  - bei längeren Pipelines
  - bei Load-Instruktionen (s.u.)

Auflösen von Rückwärtsabhängigkeiten

1. Softwarebasiert, durch den Compiler, Reihenfolge der Instruktionen verändern ▸ Beispiel
   - andere Operationen (ohne Datenabhängigkeiten) vorziehen
   - nop-Befehl(e) einfügen

Pipeline Datenkonflikte (cont.)

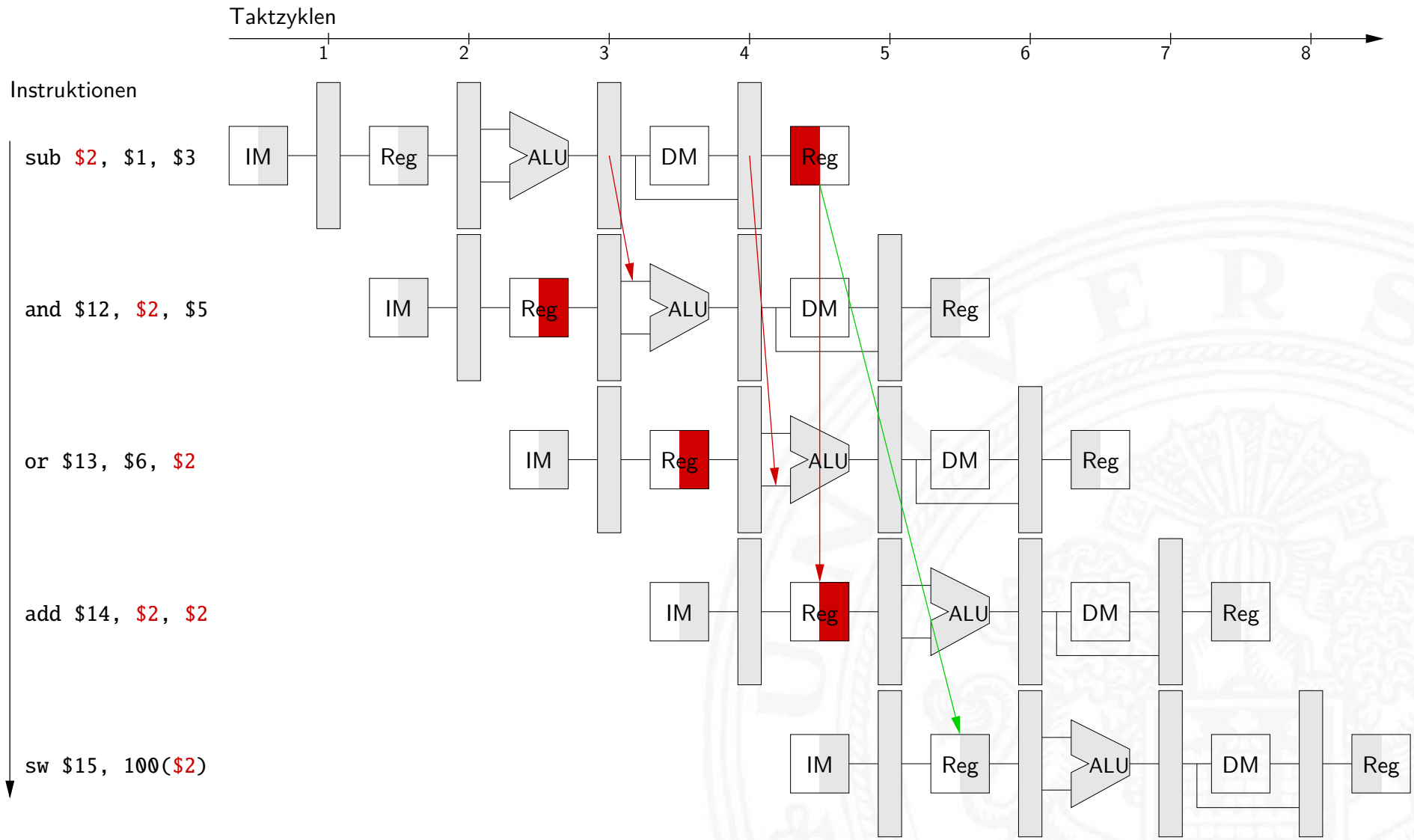
2. *„Interlocking“*

▸ Beispiel

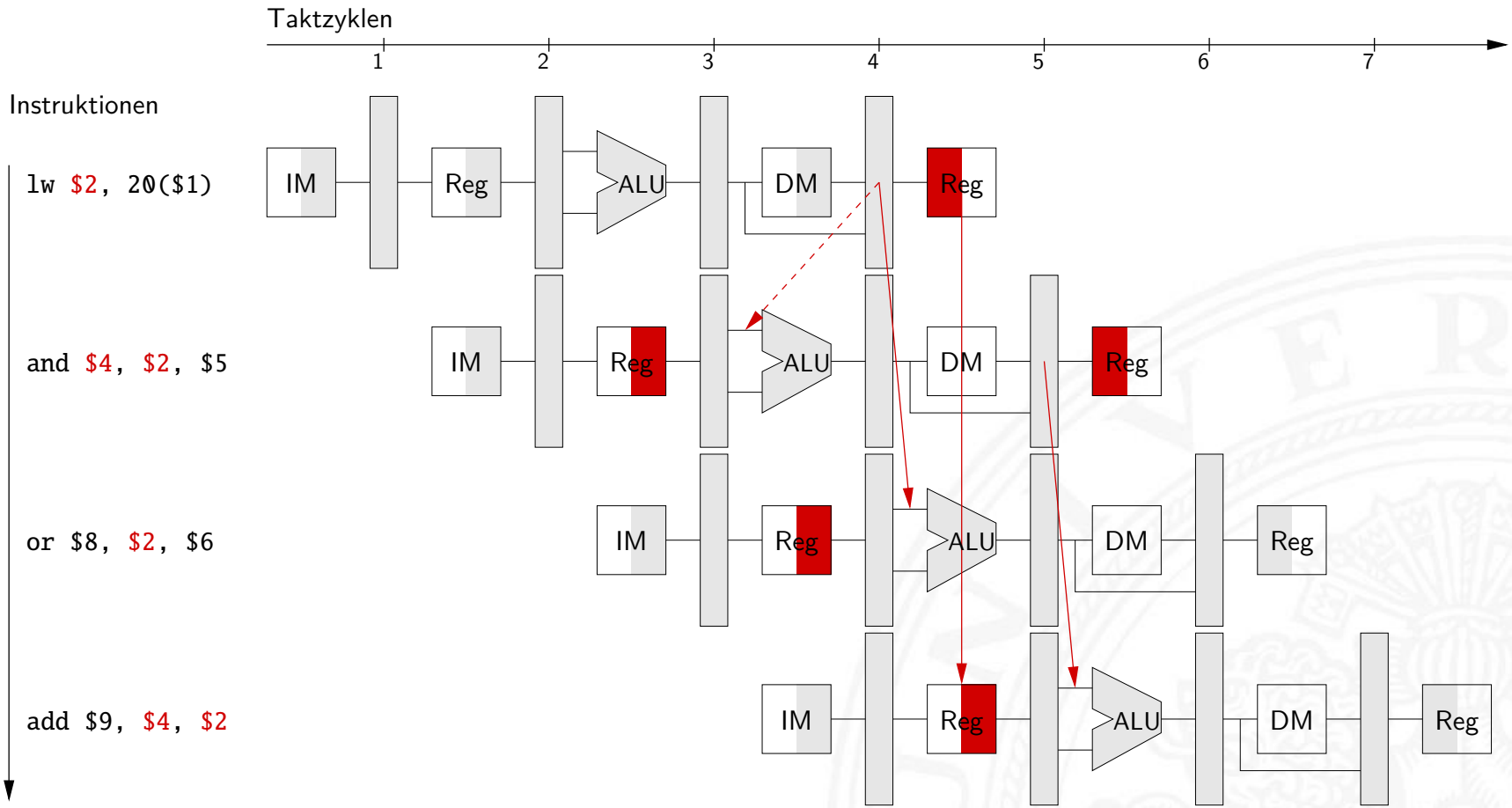
   - zusätzliche (Hardware) Kontrolleinheit
   - verschiedene Strategien
   - in Pipeline werden keine neuen Instruktionen geladen
   - Hardware erzeugt: Pipelineleerlauf / *„pipeline stall“*

„Scoreboard“

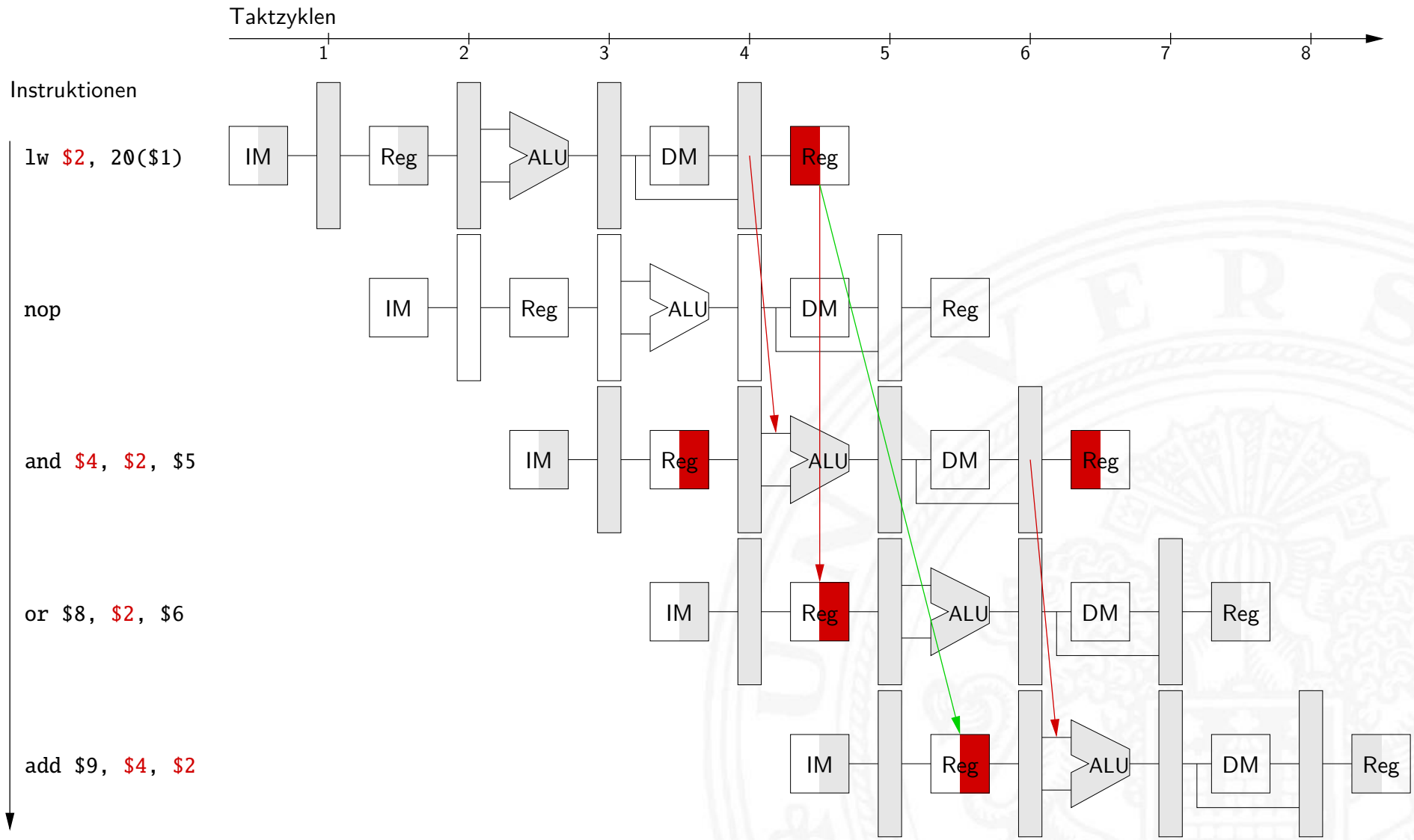
- Hardware Einheit zur zentralen Hazard-Erkennung und -Auflösung
- Verwaltet Instruktionen, benutzte Einheiten und Register der Pipeline



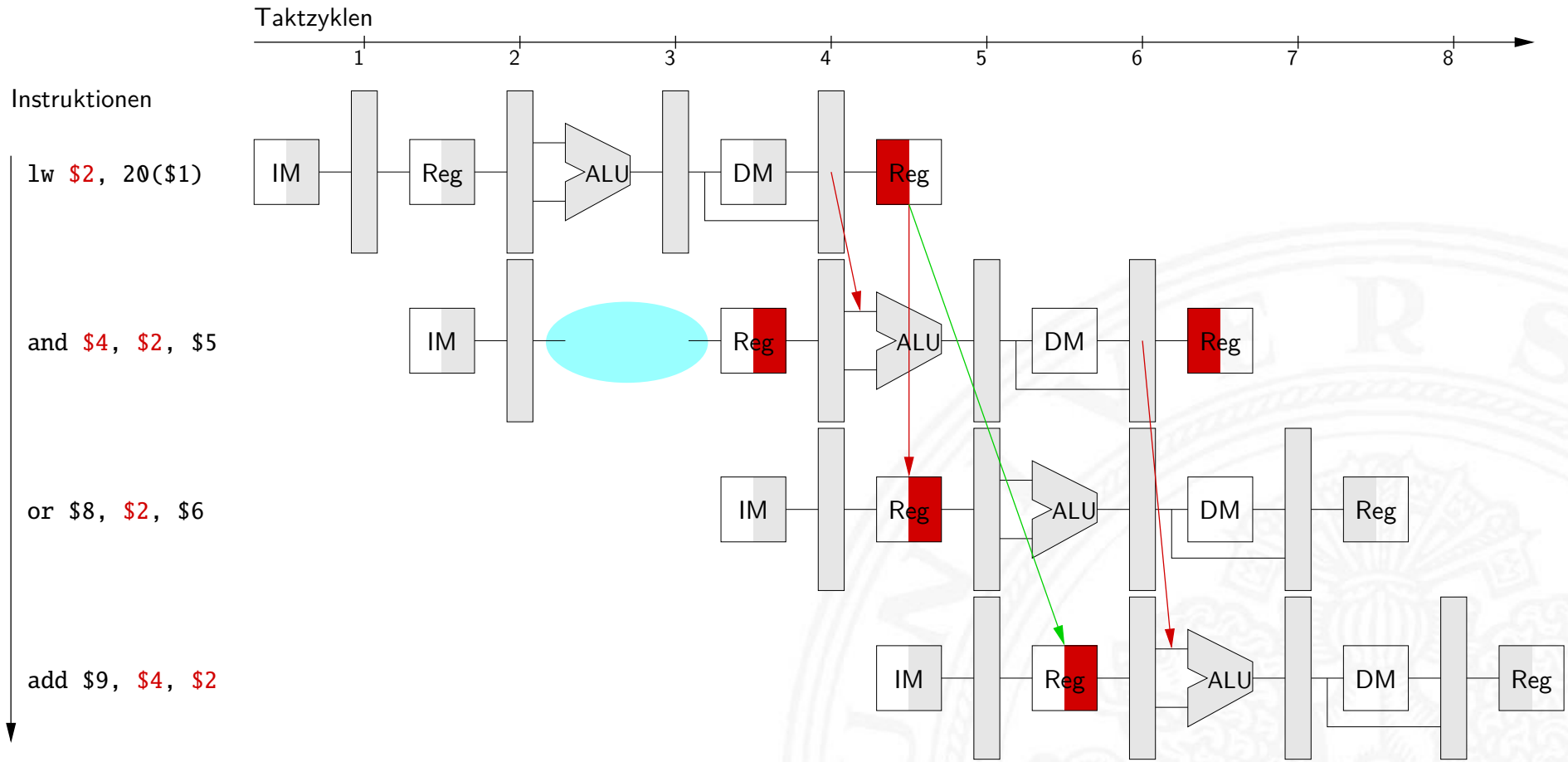
◂ Datenkonflikte



◂ Datenkonflikte



◂ Datenkonflikte



◂ Datenkonflikte

Pipeline Steuerkonflikte

Steuerkonflikt / Control Hazard

- Sprungbefehle unterbrechen den sequenziellen Ablauf der Instruktionen
- Problem: Instruktionen die auf (bedingte) Sprünge folgen, werden in die Pipeline geschoben, bevor bekannt ist, ob verzweigt werden soll
- Beispiel: bedingter Sprung

▸ Beispiel

Pipeline Steuerkonflikte (cont.)

Lösungsmöglichkeiten für Steuerkonflikte

- ad-hoc Lösung: „Interlocking“ erzeugt Pipelineleerlauf
  - – ineffizient: ca. 19 % der Befehle sind Sprünge

1. Annahme: nicht ausgeführter Sprung / „untaken branch“
   - \+ kaum zusätzliche Hardware
   - – im Fehlerfall
     - Pipelineleerlauf
     - Pipeline muss geleert werden / „flush instructions“
2. Sprungentscheidung „vorverlegen“
   - Software: Compiler zieht andere Instruktionen vor
     Verzögerung nach Sprungbefehl / „delay slots“
   - Hardware: Sprungentscheidung durch Zusatz-ALU
     (nur Vergleiche) während Befehlsdecodierung (z.B. MIPS)

Pipeline Steuerkonflikte (cont.)

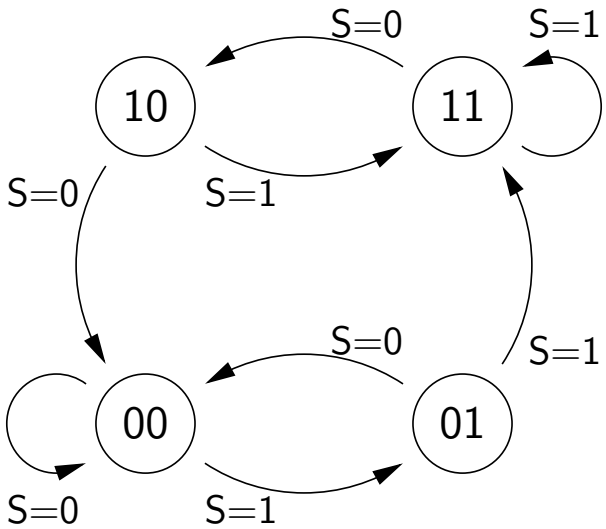
3. Sprungvorhersage / *„branch prediction“*
   - Beobachtung: ein Fall tritt häufiger auf:
     Schleifendurchlauf, Datenstrukturen durchsuchen etc.
   - mehrere Vorhersageverfahren; oft miteinander kombiniert
   + hohe Trefferquote: bis 90 %

   Statische Sprungvorhersage (softwarebasiert)
   - Compiler erzeugt extra Bit in Opcode des Sprungbefehls
   - Methoden: Codeanalyse, Profiling...

   Dynamische Sprungvorhersage (hardwarebasiert)
   - Sprünge durch Laufzeitinformation vorhersagen:
     Wie oft wurde der Sprung in letzter Zeit ausgeführt?
   - viele verschiedene Verfahren:
     History-Bit, 2-Bit Prädiktor, korrelationsbasierte Vorhersage,
     Branch History Table, Branch Target Cache...

Pipeline Steuerkonflikte (cont.)

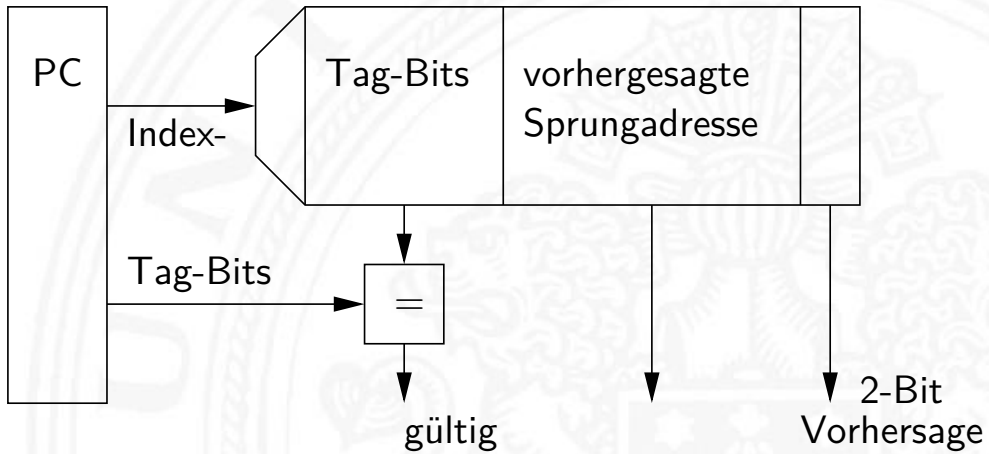
- Beispiel: 2-Bit Sprungvorhersage + Branch Target Cache



VH **V**orhersage bit

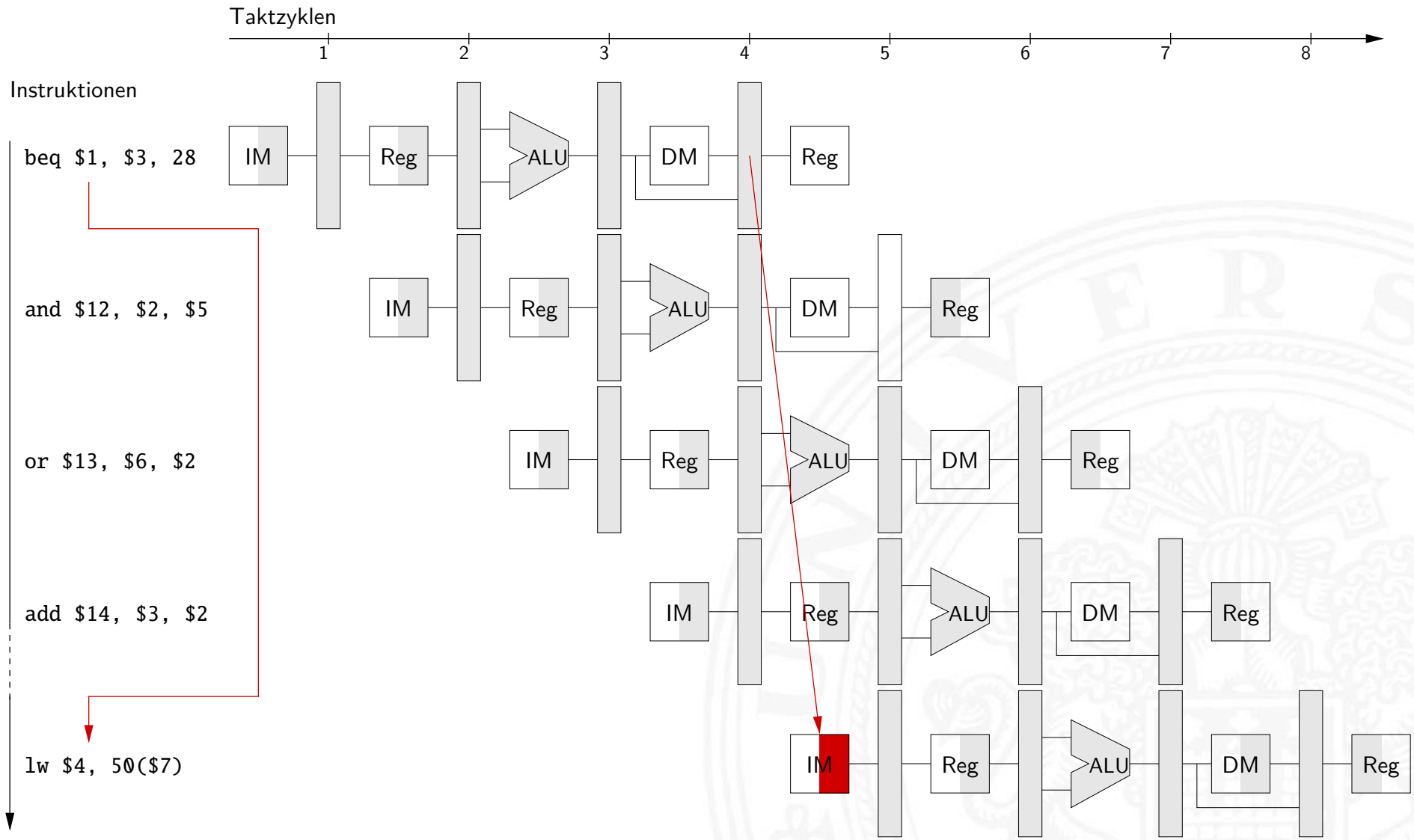
Historie bit

S=0/1 **S**prung ausgeführt



Pipeline Steuerkonflikte (cont.)

- Schleifen abrollen / „*Loop unrolling*“
  - zusätzliche Maßnahme zu allen zuvor skizzierten Verfahren
  - bei statische Schleifenbedingung möglich
  - Compiler iteriert Instruktionen in der Schleife (teilweise)
  - − längerer Code
  - \+ Sprünge und Abfragen entfallen
  - \+ erzeugt sehr lange Codesequenzen ohne Sprünge
    $\Rightarrow$ Pipeline kann optimal ausgenutzt werden



◂ Steuerkonflikte

Superskalare Prozessoren

- Superskalare CPUs besitzen mehrere Recheneinheiten: 4. . . 10
- In jedem Takt werden (dynamisch) mehrere Instruktionen eines konventionell linearen Instruktionsstroms abgearbeitet: $CPI < 1$
  ILP (**I**nstruction **L**evel **P**arallelism) ausnutzen!
- Hardware verteilt initiierte Instruktionen auf Recheneinheiten
- Pro Takt kann *mehr als eine* Instruktion initiiert werden
  Die Anzahl wird dynamisch von der Hardware bestimmt:
  0. . . „*Instruction Issue Bandwidth*“

\+ sehr effizient, alle modernen CPUs sind superskalar

– Abhängigkeiten zwischen Instruktionen sind der Engpass, das Problem der Hazards wird verschärft

Superskalar – Datenabhängigkeiten

Datenabhängigkeiten

- RAW – **R**ead **A**fter **W**rite
  Instruktion I_x darf Datum erst lesen, wenn I_{x-n} geschrieben hat
- WAR – **W**rite **A**fter **R**ead
  Instruktion I_x darf Datum erst schreiben, wenn I_{x-n} gelesen hat
- WAW – **W**rite **A**fter **W**rite
  Instruktion I_x darf Datum erst überschreiben, wenn I_{x-n} geschrieben hat

Superskalar – Datenabhängigkeiten (cont.)

Datenabhängigkeiten superskalarer Prozessoren

- RAW: echte Abhängigkeit; Forwarding ist kaum möglich und in superskalaren Pipelines extrem aufwändig
- WAR, WAW: „*Register Renaming*“ als Lösung

„*Register Renaming*“

- Hardware löst Datenabhängigkeiten innerhalb der Pipeline auf
- Zwei Registersätze sind vorhanden
  1. Architektur-Register: „logische Register“ der ISA
  2. viele Hardware-Register: „Rename Register“
     - dynamische Abbildung von ISA- auf Hardware-Register

Superskalar – Datenabhängigkeiten (cont.)

- Beispiel

Originalcode

```
tmp  = a + b;
res1 = c + tmp;
tmp  = d + e;
res2 = tmp - f;
```

nach Renaming

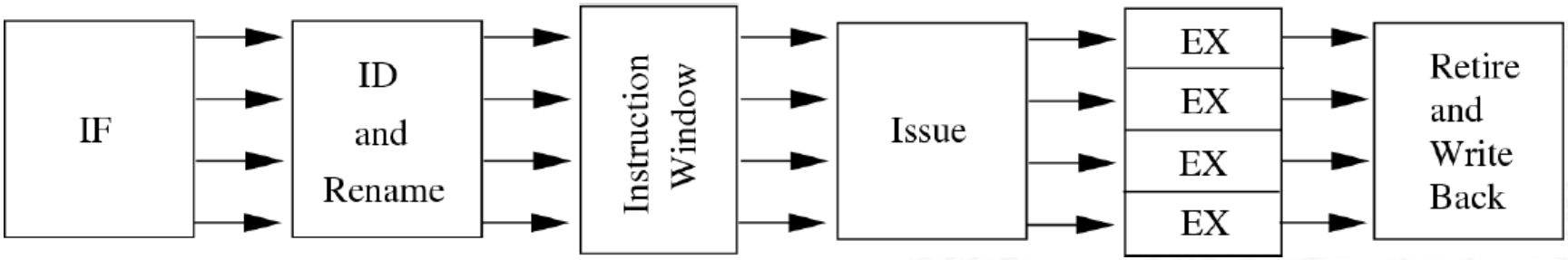
```
tmp1 = a + b;
res1 = c + tmp1;
tmp2 = d + e;
res2 = tmp2 - f;
tmp  = tmp2;
```

Parallelisierung des modifizierten Codes

```
tmp1 = a + b;      tmp2 = d + e;
res1 = c + tmp1;   res2 = tmp2 - f;   tmp = tmp2;
```

Superskalar – Pipeline

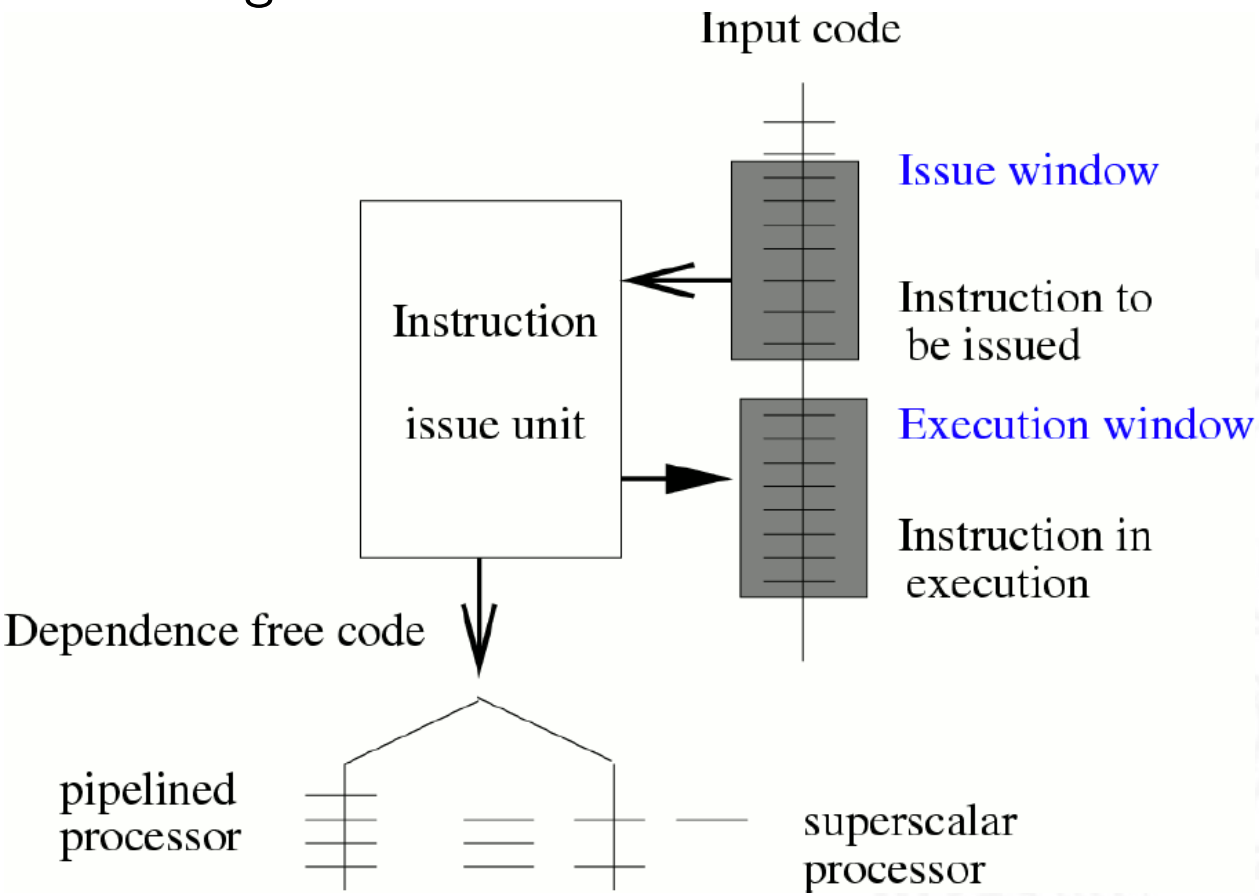
Aufbau der superskalaren Pipeline



- lange Pipelines mit vielen Phasen: Fetch (Prefetch, Predecode), Decode / Register-Renaming, Issue, Dispatch, Execute, Retire (Commit, Complete / Reorder), Write-Back
- je nach Implementation unterschiedlich aufgeteilt
- entscheidend für superskalare Architektur sind die Schritte
  vor den ALUs: Issue, Dispatch ⇒ *out-of-order* Ausführung
  nach -"- : Retire ⇒ *in-order* Ergebnisse

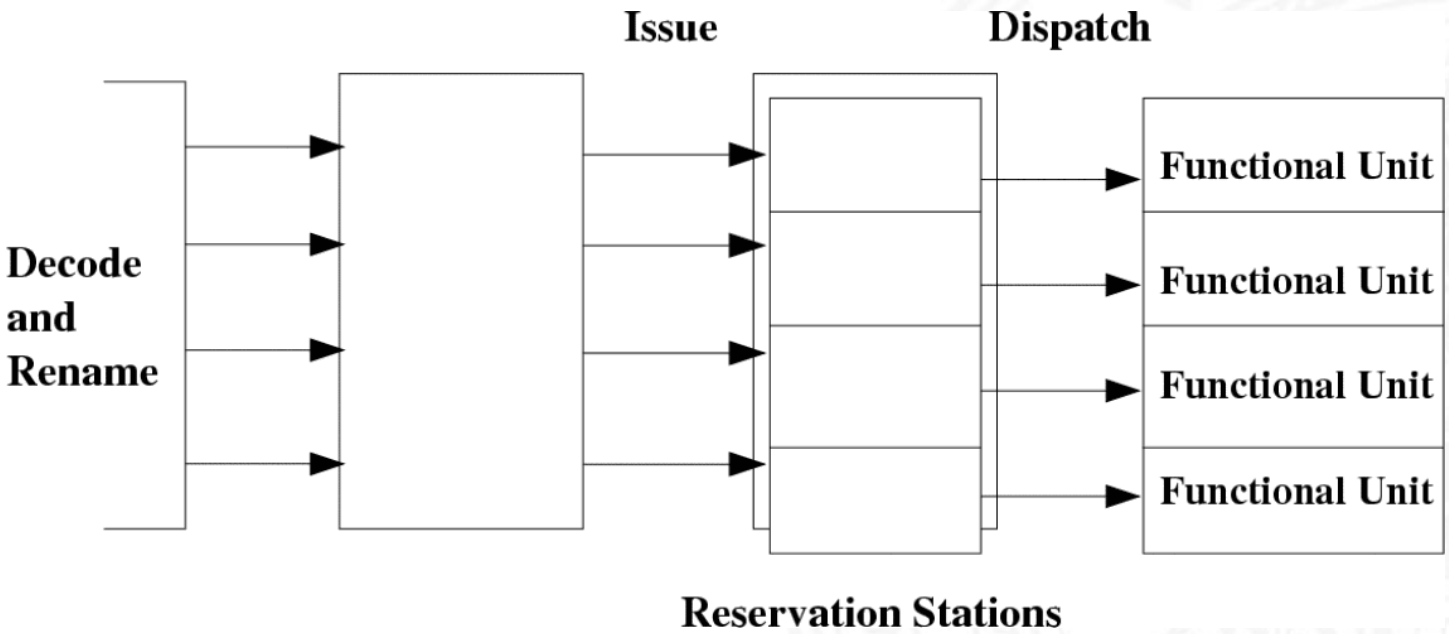
Superskalar – Pipeline (cont.)

Scheduling der Instruktionen



Superskalar – Pipeline (cont.)

- Dynamisches Scheduling erzeugt *out-of-order* Reihenfolge der Instruktionen
- Issue: globale Sicht
  Dispatch: getrennte Ausschnitte in „Reservation Stations“



Superskalar – Pipeline (cont.)

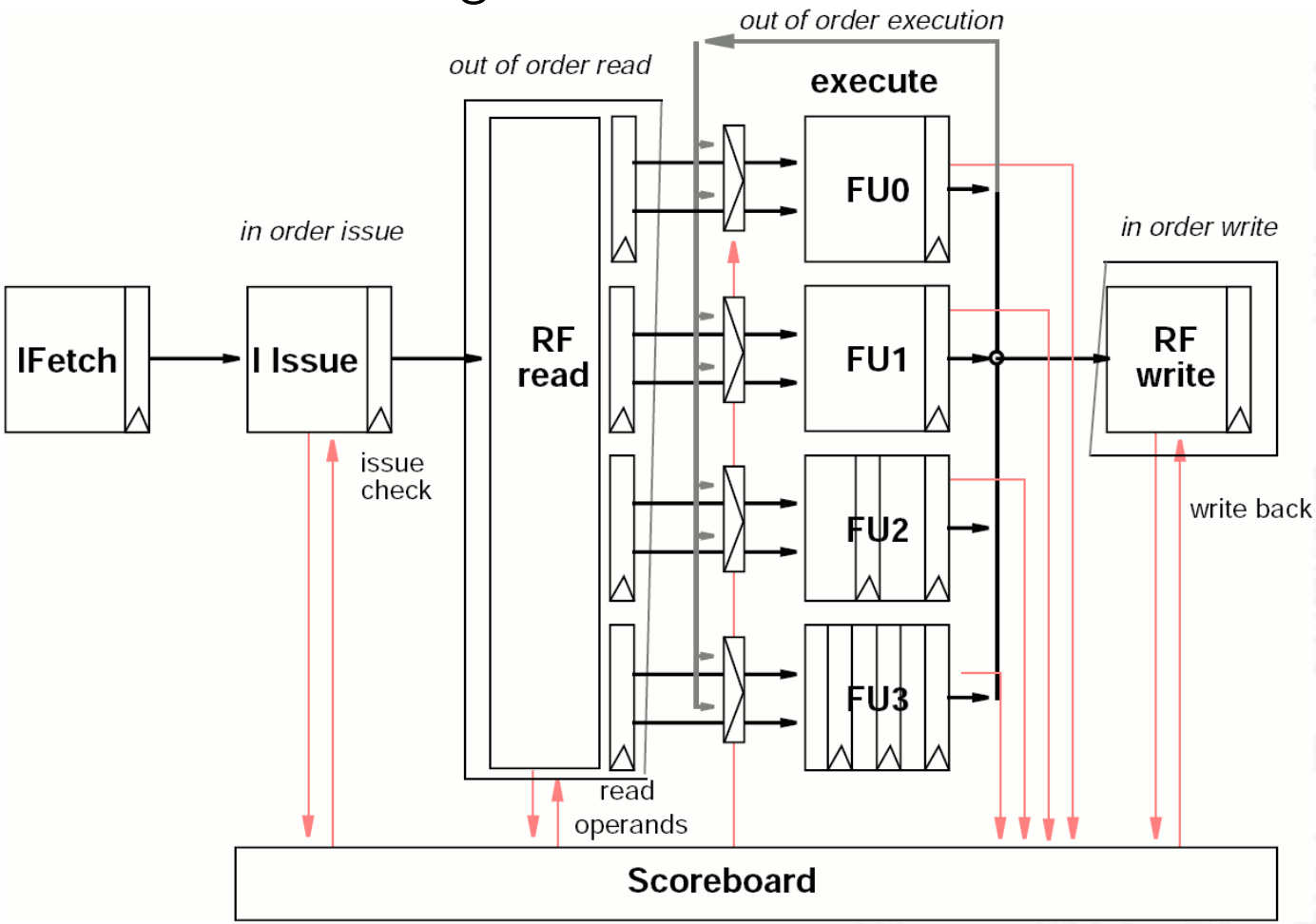
Reservation Station für jede Funktionseinheit

- speichert: initiierte Instruktionen die auf Recheneinheit warten
- –"– zugehörige Operanden
- –"– ggf. Zusatzinformation
- Instruktion bleibt blockiert, bis alle Parameter bekannt sind und wird dann an die zugehörige ALU weitergeleitet

- Dynamisches Scheduling: zuerst '67 in IBM 360 (Robert Tomasulo)
  - Forwarding
  - Registerumbenennung und Reservation Stations

Superskalar – Pipeline (cont.)

Zentrale Verwaltungseinheit: „Scoreboard“



Superskalar – Pipeline (cont.)

Scoreboard erlaubt das Management mehrerer Ausführungseinheiten

- out-of-order Ausführung von Mehrzyklusbefehlen
- Auflösung aller Struktur- und Datenkonflikte: RAW, WAW, WAR

Einschränkungen

- single issue (nicht superskalar)
- in-order issue
- keine Umbenennungen; also Leerzyklen bei WAR- und WAW-Konflikten
- kein Forwarding, daher Zeitverlust bei RAW-Konflikten

Superskalar – Pipeline (cont.)

Retire-Stufe

- erzeugt wieder *in-order* Reihenfolge
- FIFO: Reorder-Buffer
- commit: „richtig ausgeführte“ Instruktionen gültig machen
- abort: Sprungvorhersage falsch
  Instruktionen verwerfen

Superskalar – Pipeline (cont.)

Spezielle Probleme superskalarer Pipelines

- weitere Hazard-Möglichkeiten
  - die verschiedenen ALUs haben unterschiedliche Latenzzeiten
  - Befehle „warten“ in den Reservation Stations

  $\Rightarrow$ Datenabhängigkeiten können sich mit jedem Takt ändern
- Kontrollflussabhängigkeiten: Anzahl der Instruktionen zwischen bedingten Sprüngen limitiert Anzahl parallelisierbarer Instruktion

$\Rightarrow$ „Loop Unrolling“ wichtig

$+$ optimiertes (dynamisches) Scheduling: Faktor 3 möglich

Superskalar – Pipeline (cont.)

Software Pipelining

- Codeoptimierungen beim Compilieren: Ersatz für, bzw. Ergänzend zu der Pipelineunterstützung durch Hardware
- Compiler hat „globalen“ Überblick
  $\Rightarrow$ zusätzliche Optimierungsmöglichkeiten
- symbolisches Loop Unrolling
- Loop Fusion
- ...

Superskalar – Interrupts

Exceptions, Interrupts und System-Calls

- Interruptbehandlung ist wegen der Vielzahl paralleler Aktionen und den Abhängigkeiten innerhalb der Pipelines extrem aufwändig
  - da unter Umständen noch Pipelineaktionen beendet werden müssen, wird *zusätzliche Zeit* bis zur Interruptbehandlung benötigt
  - wegen des Register-Renaming muss sehr viel *mehr Information* gerettet werden als nur die ISA-Register
- Prinzip der Interruptbehandlung
  - keine neuen Instruktionen mehr initiieren
  - warten bis Instruktionen des Reorder-Buffers abgeschlossen sind

Superskalar – Interrupts (cont.)

- Verfahren ist von der „Art“ des Interrupt abhängig
  - Precise-Interrupt: Pipelineaktivitäten komplett Beenden
  - Imprecise-Interrupt: wird als verzögerter Sprung (Delayed-Branching) in Pipeline eingebracht
    Zusätzliche Register speichern Information über Instruktionen die in der Pipeline nicht abgearbeitet werden können (z.B. weil sie den Interrupt ausgelöst haben)
- Definition: Precise-Interrupt
  - Programmzähler (PC) zur Interrupt auslösenden Instruktion ist bekannt
  - Alle Instruktionen bis zur PC-Instruktion wurden vollständig ausgeführt
  - Keine Instruktion nach der PC-Instruktion wurde ausgeführt
  - Ausführungszustand der PC-Instruktion ist bekannt

EPIC

EPIC – **E**xplicit **P**arallelism **I**nstruction **C**omputing

- neues Paradigma, ab Mitte der 90er Jahre
  Beispiel: Intel IA-64
- Idee: bei parallelen Recheneinheiten (z.B. superskalare Pipeline) möglichst viel Optimierung in den Compiler verlagern

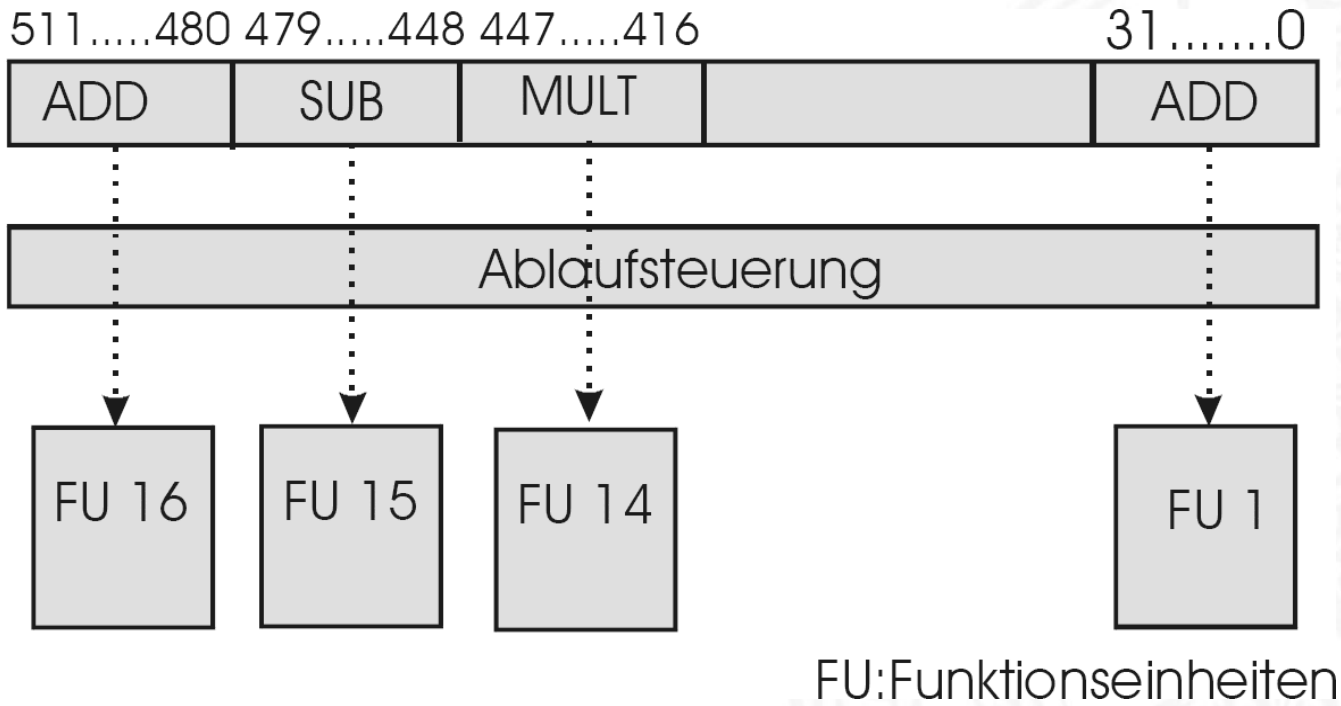
$\Rightarrow$ einfachere Hardware

- VLIW als eine spezielle Klasse

VLIW

VLIW – **V**ery **L**arge **I**nstruction **W**ord

- parallel ausführbare Instruktionen zu *einem einzigen* Befehl zusammenfassen



VLIW (cont.)

- Compiler kann dies „Offline“ erledigen
- Begriff: *Statischer Superskalar Rechner*
- wird häufig verwendet bei:
  Signalprozessoren, eingebettete Prozessoren für hohe Leistung
- Beispiele
  - TMS320C62xx Signalprozessoren (TI)
  - Crusoe / Efficeon Prozessor (Transmeta)

\+ bessere Ergebnisse, mehr Optimierungsmöglichkeiten als „on-the-fly“ superskalar Optimierung

\+ einfachere Hardware, keine Spezialhardware für Verwaltung der Recheneinheiten

VLIW (cont.)

- Einheiten gleichmäßig auslasten
- Bussystem: Ein-/Ausgabemöglichkeiten zu den Einheiten
- Statisches „In Order“ Scheduling ist unflexibel bei externen Ereignissen
- Instruktionsworte lang: „Code Explosion“ bei Leerlauf (nop-Befehle)
- Codekompatibilität nicht gegeben

Multi-Threading / Multi-Core

- Problem superskalarer Architekturen:
  einzelne CPUs lassen sich kaum noch weiter optimieren
  - weitere parallele Ausführungseinheiten sind nur schwer gut auszulasten
  - Parallelisierung über bedingte Sprungbefehle hinweg ist nur schwer möglich oder wenig effizient
  - zwischen den Sprungbefehlen finden sich i.A. nicht genügend Befehle, die sich parallelisieren lassen

  $\Rightarrow$ maximal 5-10 Befehle sind parallel abzuarbeiten
- Lösung: Multi-Threading, Multi-Core
- Parallelität muss explizit vorhanden sein
  - Anwender schreibt parallele Programme
  - verschiedene Prozesse des Betriebssystems

Multi-Threading / Multi-Core (cont.)

- Multi-Threading
  - mehrere Threads (Tasks, Prozesse...) werden parallel bearbeitet
  - Teile der Hardware sind für jeden Thread separat vorhanden, andere Funktionseinheiten (Caches, Pipelinekontrolle etc.) werden gemeinsam verwendet
  - einfachste Realisierung: Registersätze der ISA und Pipelineregister in den Datenpfaden trennen
    Beispiel: Intel *„Hyper-Threading“*

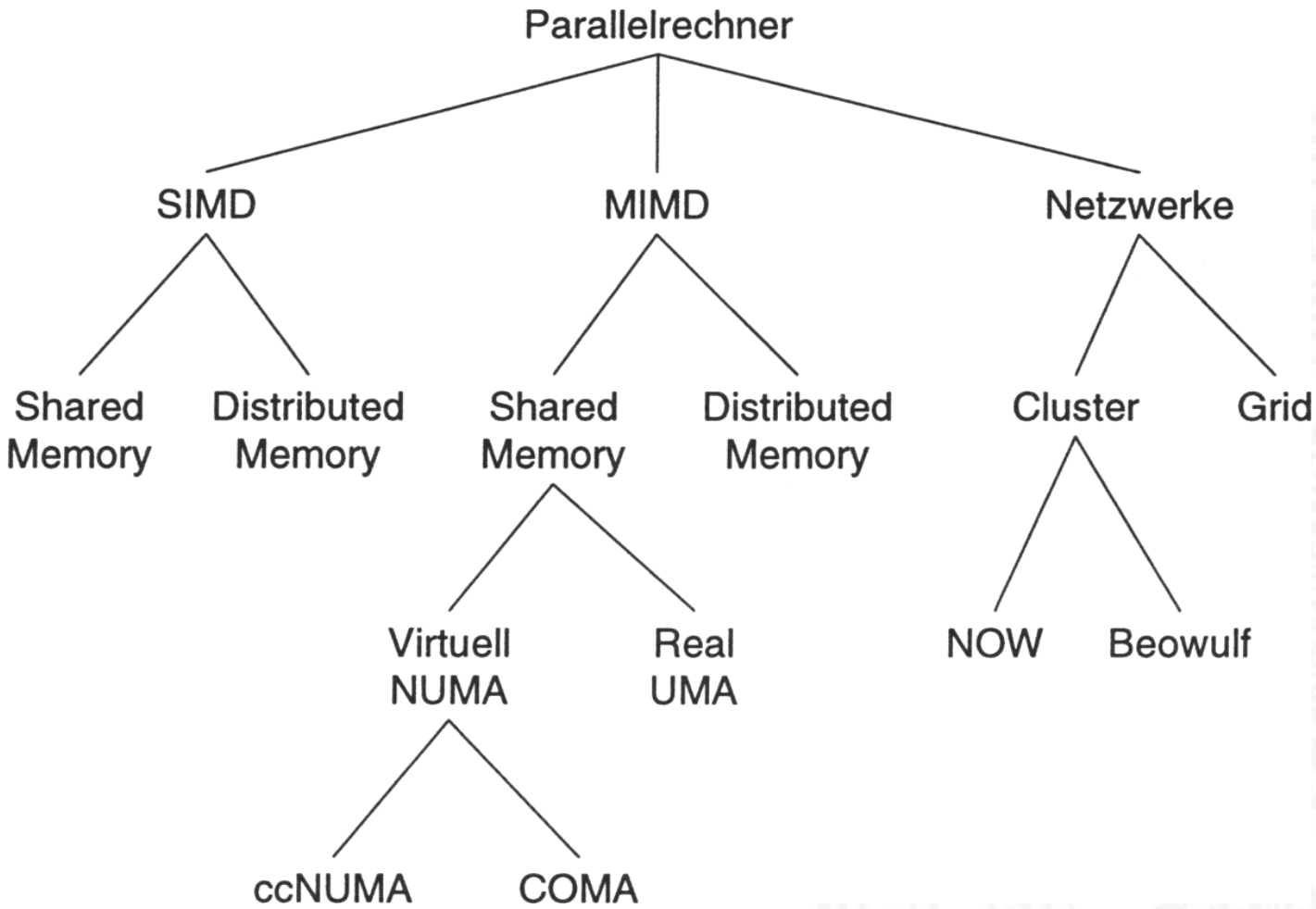
  $\Rightarrow$ Kontextwechsel von einen Takt zum anderen möglich

  $\Rightarrow$ ein einzelner Thread wird nicht schneller abgearbeitet aber der Durchsatz steigt
- Multi-Core
  - mehrere identische Prozessoren auf einem Chip, gemeinsam sind nur 2nd-Level Cache und (einige) Pins

Parallelrechner

- viele Varianten, kein einheitliches Klassifikationsschema
- Unterscheidungskriterien
  - Systemaufbau
  - Kommunikationsmechanismen
  - Speicherarchitektur
  - Kopplung der Knoten
  - Topologie
  - ...

Klassifizierung von Parallelrechnern



[OV06]

Klassifizierung von Parallelrechnern (cont.)

Systemaufbau

- homogenes System: alle Knoten haben die gleiche Architektur, bzw. Funktion
- heterogenes System: Knoten mit unterschiedlichen Prozessoren, Speicher, IO-Kanälen, bzw. Funktionen
  - Anpassung des Problems an heterogene Fähigkeiten der Knoten
  - extra Programme für jede Klasse von Knoten
- Kontrollarchitektur: zentral oder verteilt

Kommunikation zwischen den Knoten/Rechnern

- gemeinsam benutzte Speicherbereiche
- Nachrichtenaustausch

Klassifizierung von Parallelrechnern (cont.)

Möglichkeiten der Speicherarchitektur

- lokaler Speicher je Knoten $\Rightarrow$ Nachrichten zur Kommunikation
- globaler Speicher
  - UMA – **U**niform **M**emory **A**ccess
  - NUMA – **N**on-**U**niform **M**emory **A**ccess
- verschiedene Mischformen

Kopplung der Knoten

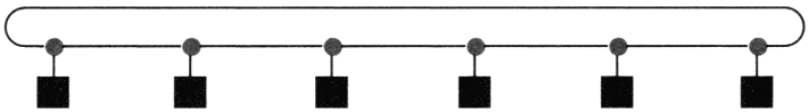
- enge Kopplung: Speicher, Busse, Interruptleitungen...
- lose Kopplung: Netzwerkstrukturen

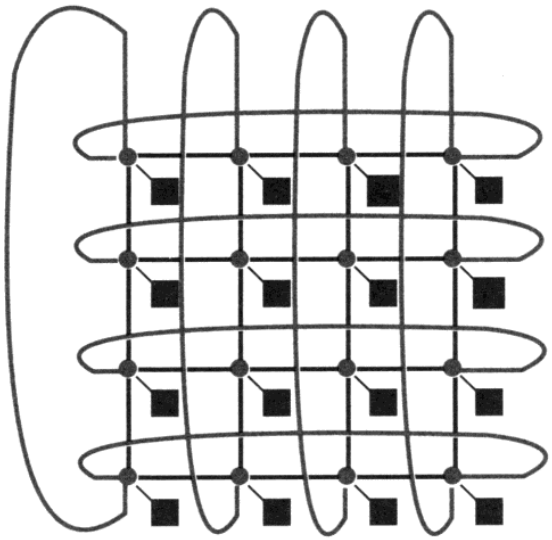
Klassifizierung von Parallelrechnern (cont.)

Topologien

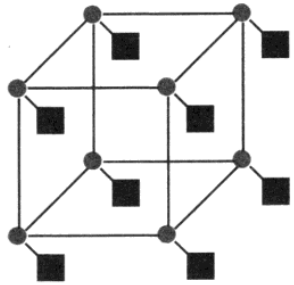
- direkte Verbindung: alle Knoten miteinander (per Switch)
  indirekte Verbindung: nur zu einigen Knoten/Switches
- angepasste Routing Algorithmen; Flusskontrolle
- statisch oder dynamische Netzwerkstruktur
- Anwendungstopologien (Bäume, Arrays etc.) einbetten

Parallelrechner Topologien





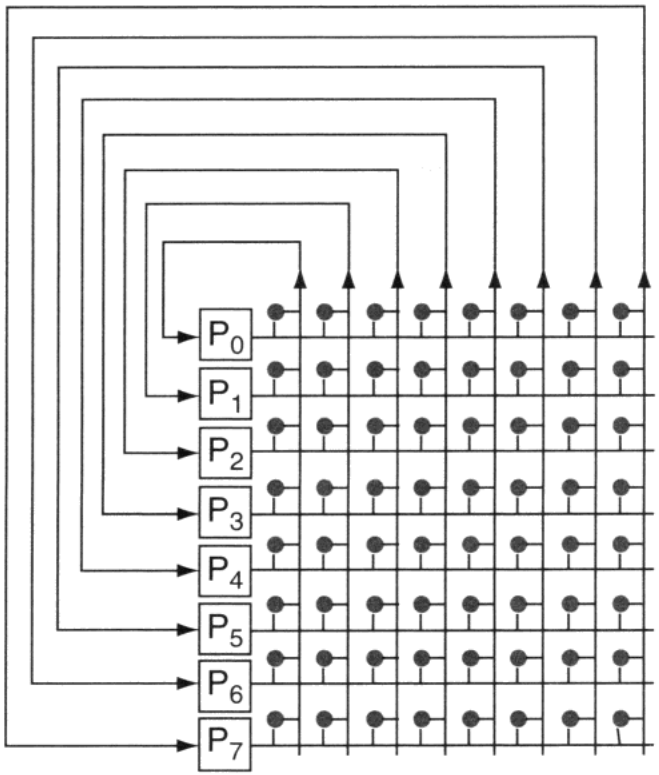
2-D-Gitter oder -Netz aus 16 Knoten



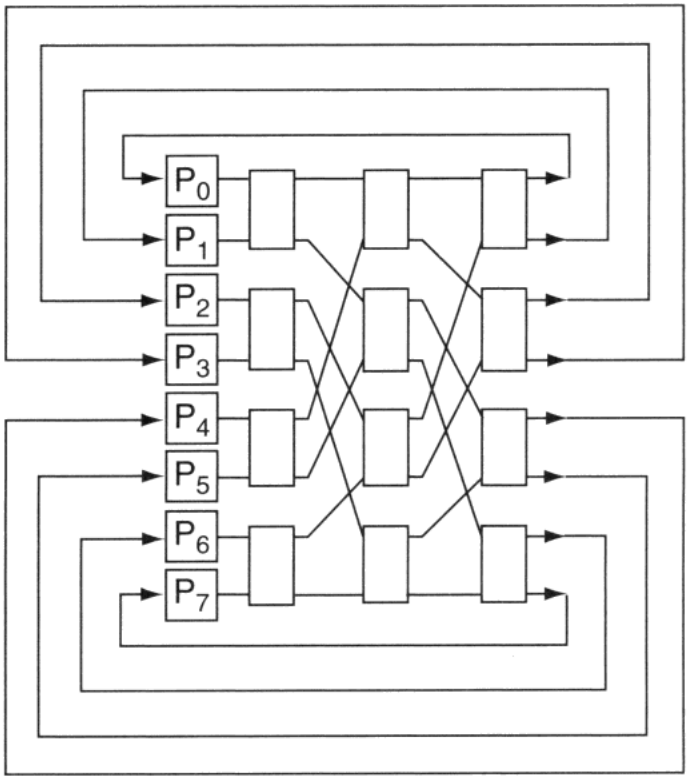
n-Würfel aus 8 Knoten
($8 = 2^3$ damit ist $n = 3$)

[PH05]

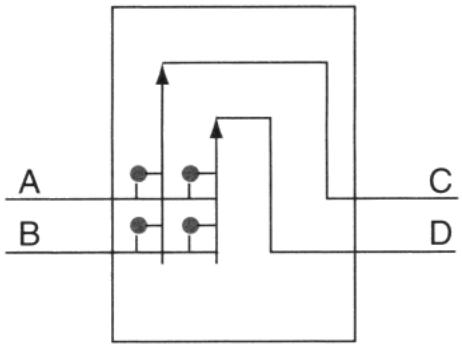
Parallelrechner Topologien (cont.)



Kreuzschiene (Crossbar)



Omega-Netzwerk



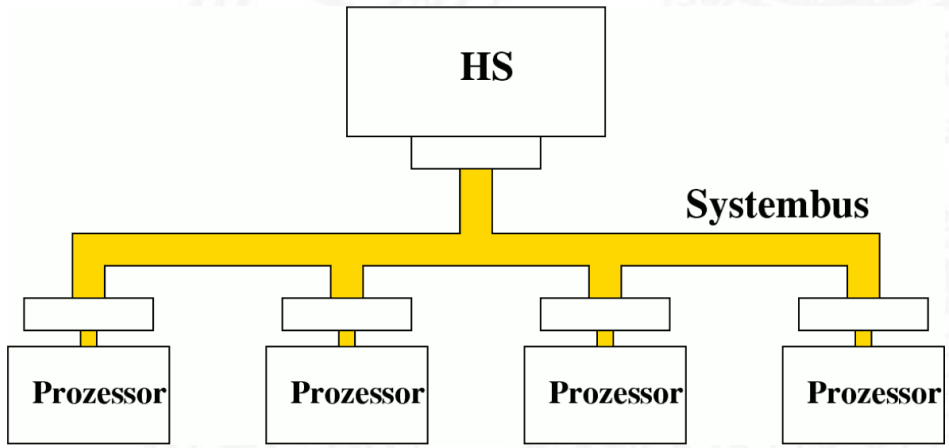
Switch-Box in Omega-Netzwerk

[PH05]

typische Parallelrechner Konfigurationen

- Workstation-Cluster / Grid Computing
  Verteiltes Rechnen über Internet (SETI@home etc.)
  - Kopplung über Ethernet (Internet)
  - Kommunikation langsam

  $\Rightarrow$ nur für parallele Algorithmen, die selten und sehr wenig Informationen untereinander austauschen
- SMP – **S**ymmetric **M**ulti**p**rocessor
  - gleiche CPUs
  - gleichartiger Zugriff auf Speicher und IO



typische Parallelrechner Konfigurationen (cont.)

- Zugriff auf Speicher als “bottleneck„
- Interrupt Behandlung schwierig
  - *Welcher Prozessor soll einen Interrupt bearbeiten?*
  - Die Prozessoren müssen untereinander Interrupts erzeugen
  - ...

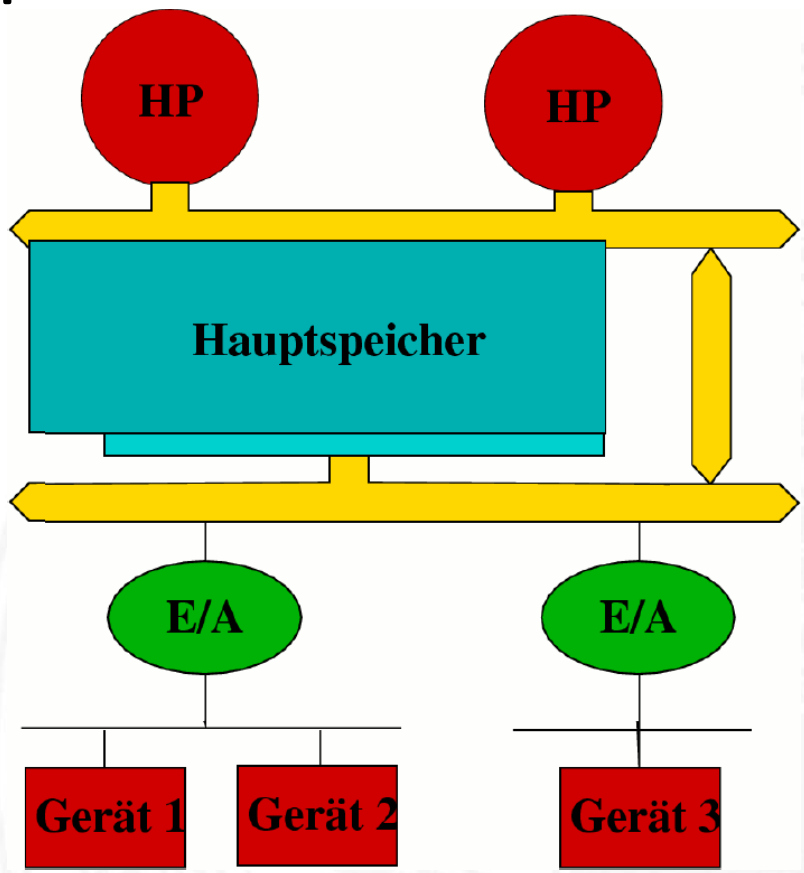
  ⇒ jeder Prozessor erhält einen eigenen Interrupt Controller
- gegenseitige Sicherung von Speicherzugriffen
  - mehrere Prozessoren dürfen nicht gleichzeitig Variablen schreiben und lesen

  ⇒ „*Spin-Locks*“: atomare „read-modify-write“ Befehle, Hardwareunterstützt

  ⇒ „*Bus-Locking*“ Mechanismen

typische Parallelrechner Konfigurationen (cont.)

- NSMP – **N**on-**S**ymmetric **M**ulti**p**rocessor
  - mehrere verschiedene (Haupt-)Prozessoren, Coprozessoren, Ein-/Ausgabeprozessoren
    ...

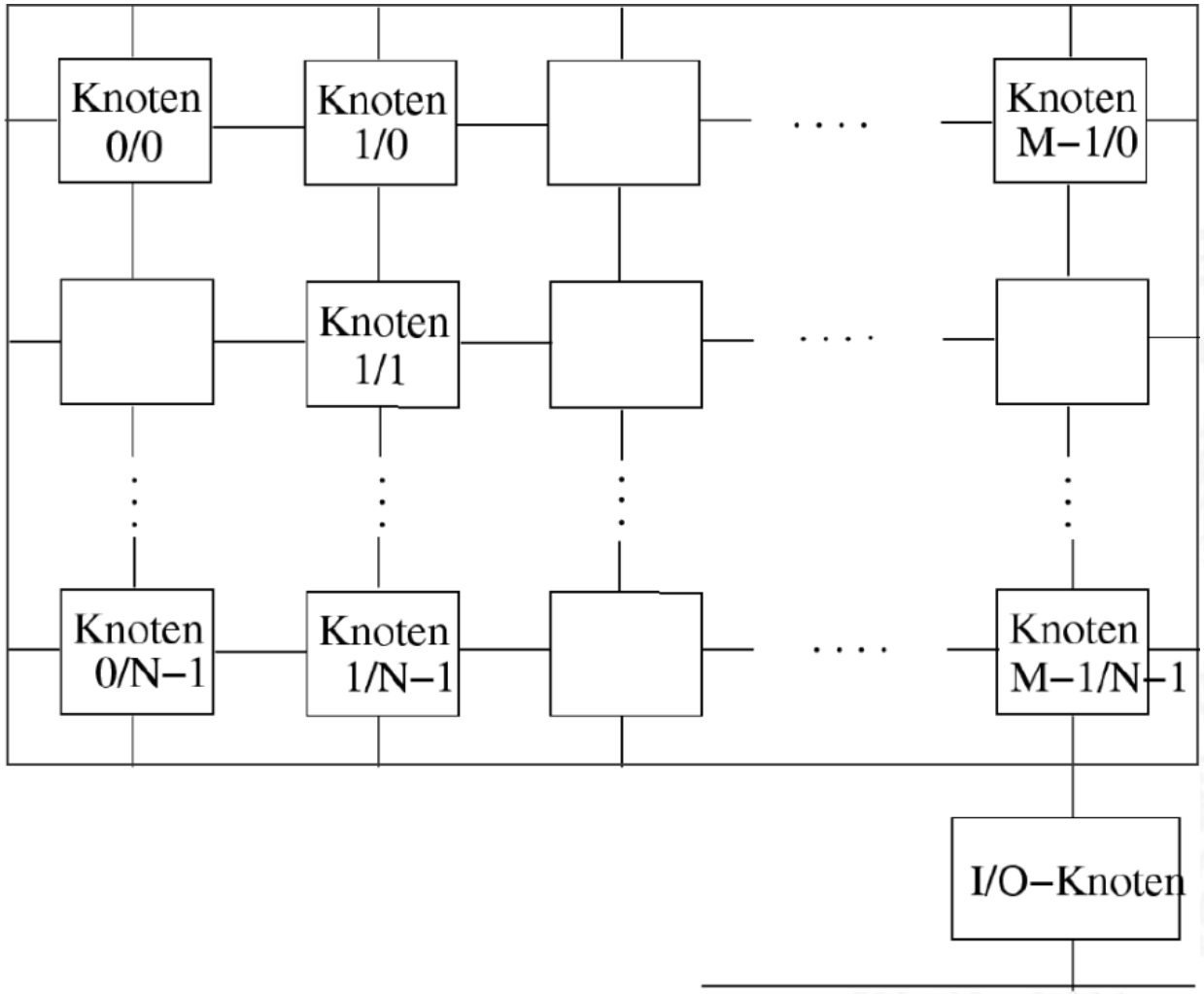


typische Parallelrechner Konfigurationen (cont.)

- Massiv parallele Prozessornetzwerke
  - Architekturen mit extrem vielen gleichartigen Prozessoren / Prozessorelementen
  - viele verschiedene Topologien: 1-D, 2-D...
  - Programmierung ähnlich Workstation-Cluster
  - Beispiel
    - zweidimensionales Netz von Knoten
    - Knoten mit CPU, lokalem Speicher und Verbindungselementen zu den 4 Nachbarn
    - an den Schnittstellen: gemeinsame Busse, Shared Memory
    - spezielle Ein-/Ausgabeknoten

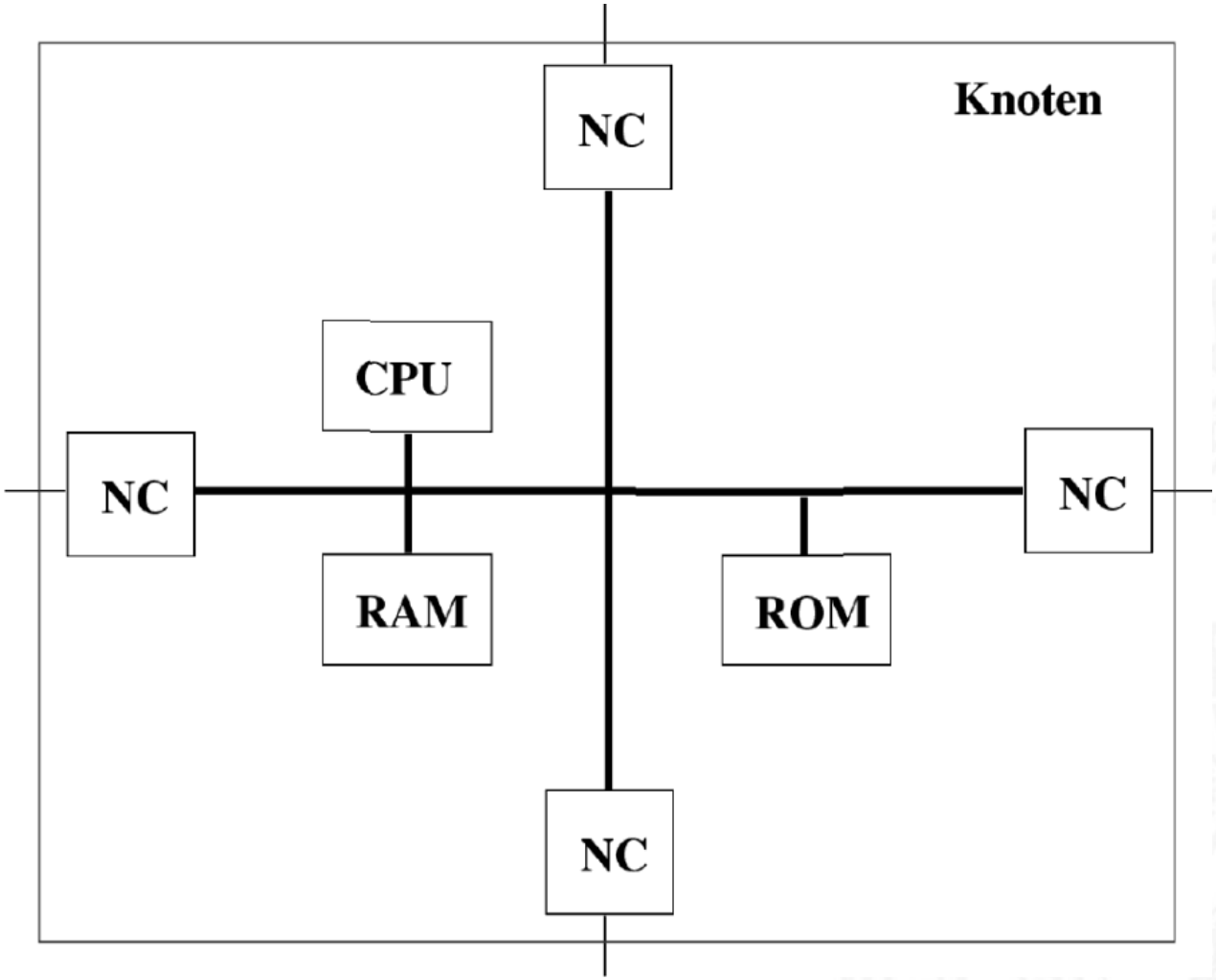
Grafik
Grafik
Grafik
Grafik

Beispiel: Netz



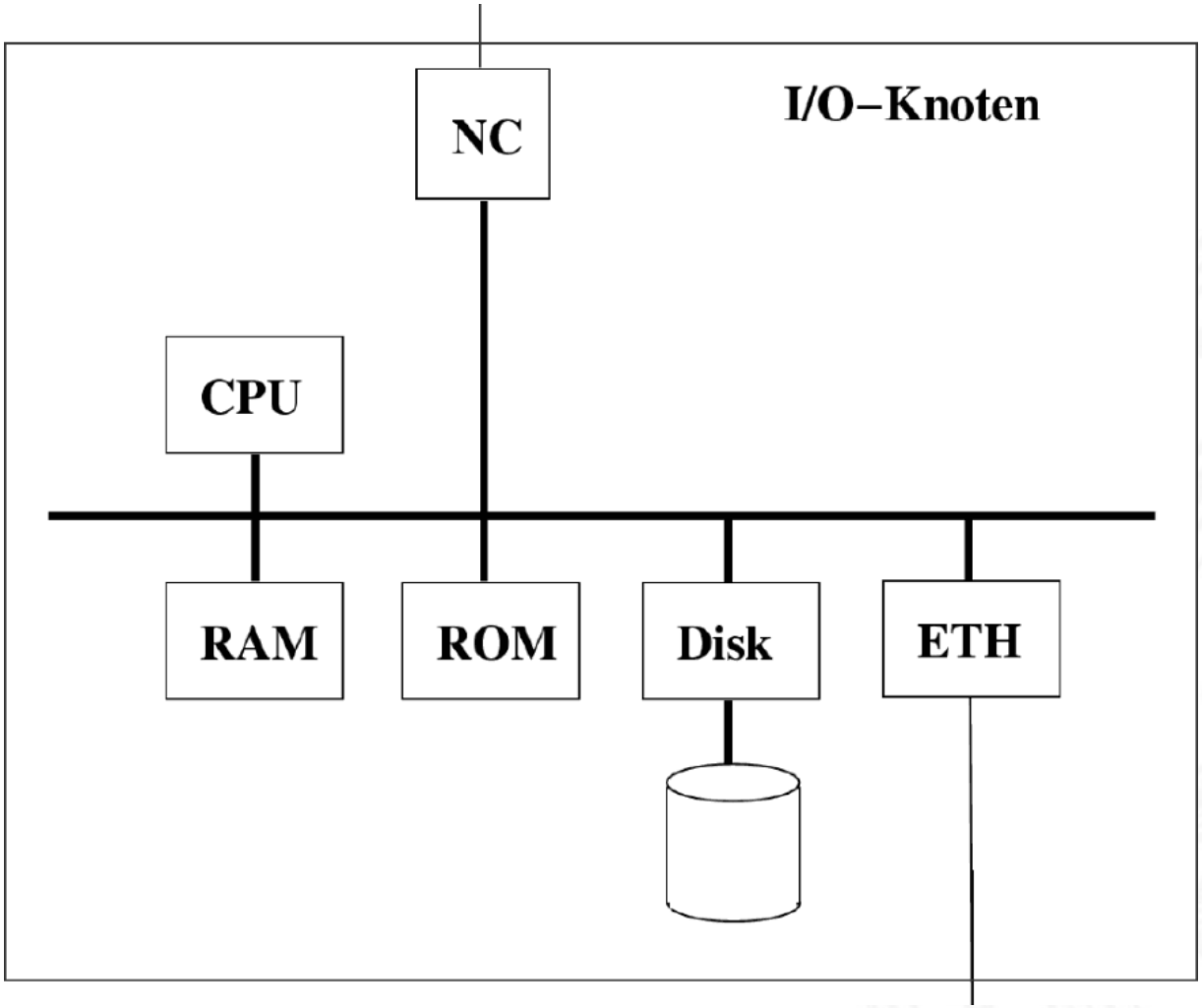
Beispiel

Beispiel: Knoten



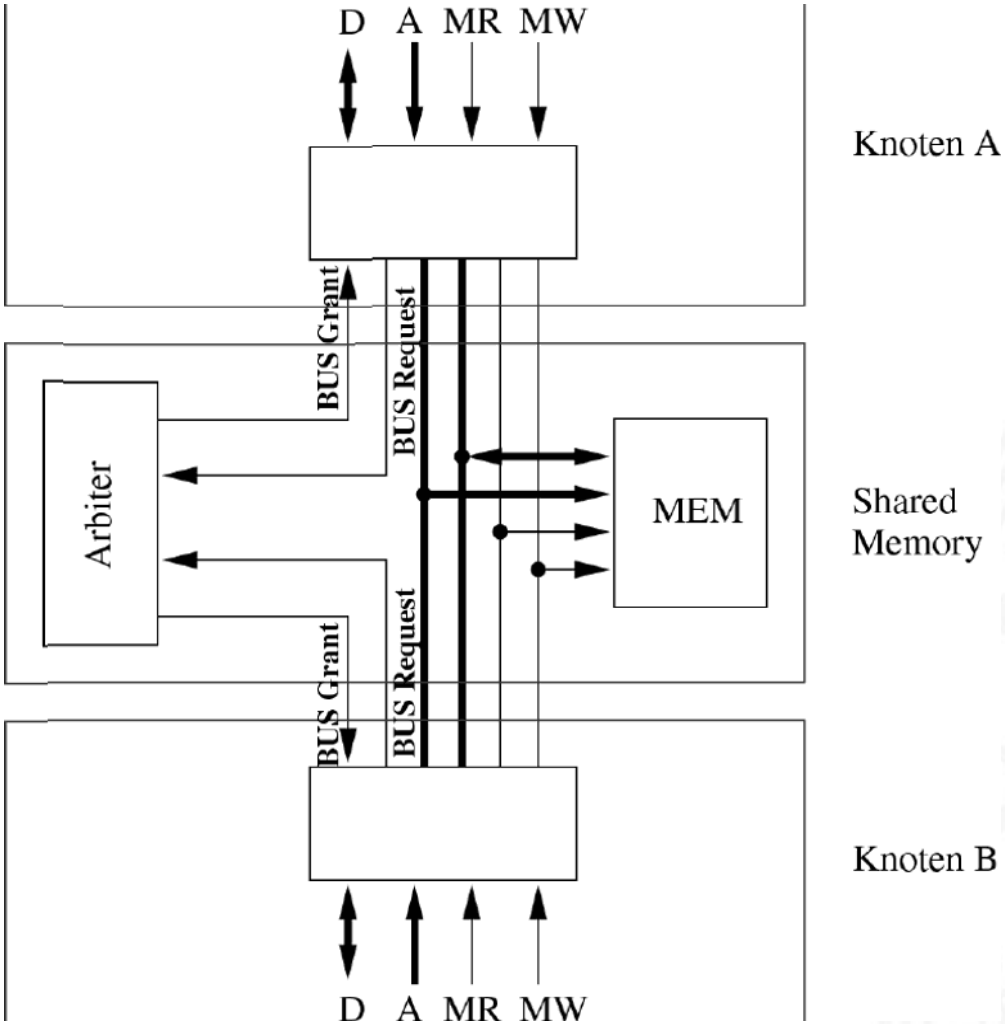
Beispiel

Beispiel: Ein-/Ausgabeknoten

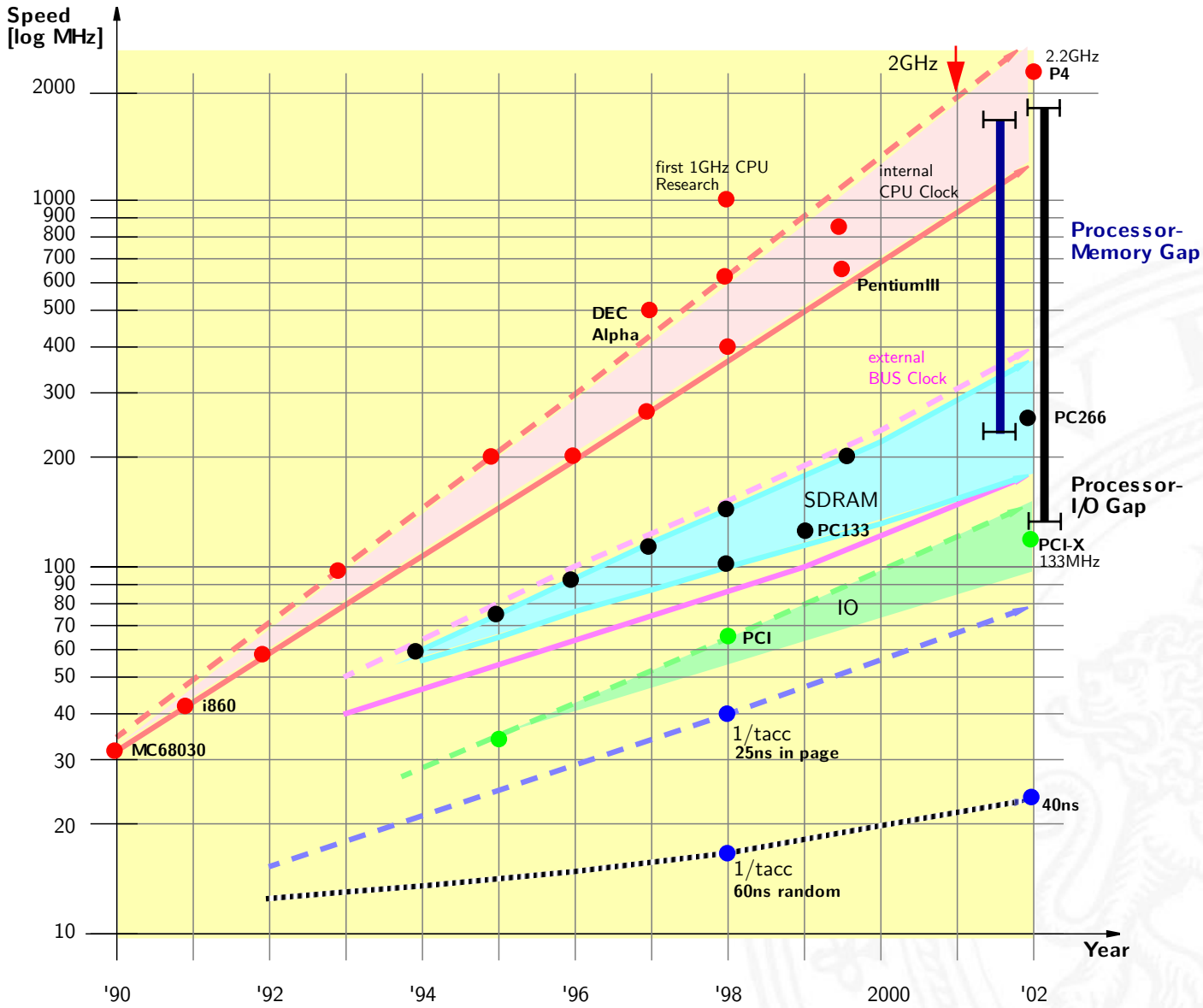


Beispiel

Beispiel: Schnittstelle zwischen Knoten



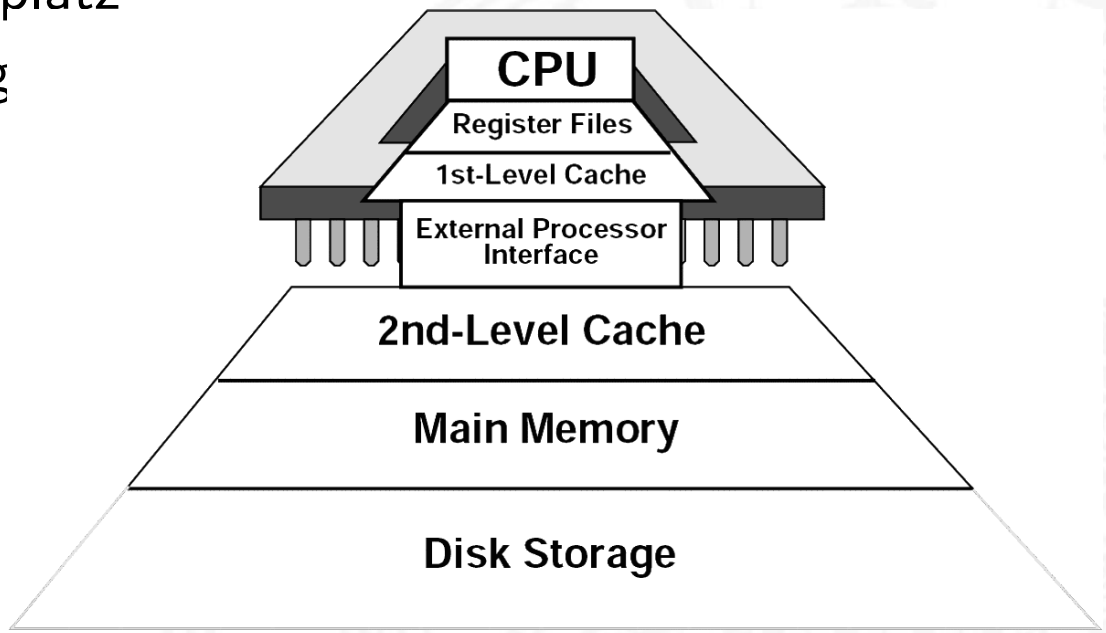
Beispiel


Speed
[log MHz]
2000
1000
900
800
700
600
500
400
300
200
100
90
80
70
60
50
40
30
20
10
2GHz
2.2GHz
P4
first 1GHz CPU
Research
internal
CPU Clock
Processor-
Memory Gap
PentiumIII
DEC
Alpha
external
BUS Clock
PC266
SDRAM
PC133
Processor-
I/O Gap
PCI-X
133MHz
IO
PCI
i860
MC68030
1/tacc
25ns in page
40ns
1/tacc
60ns random
Year
'90
'92
'94
'96
'98
2000
'02

Speicherhierarchie

Motivation

- Geschwindigkeit der Prozessoren
- Kosten für den Speicherplatz
- permanente Speicherung
  - magnetisch
  - optisch
  - mechanisch

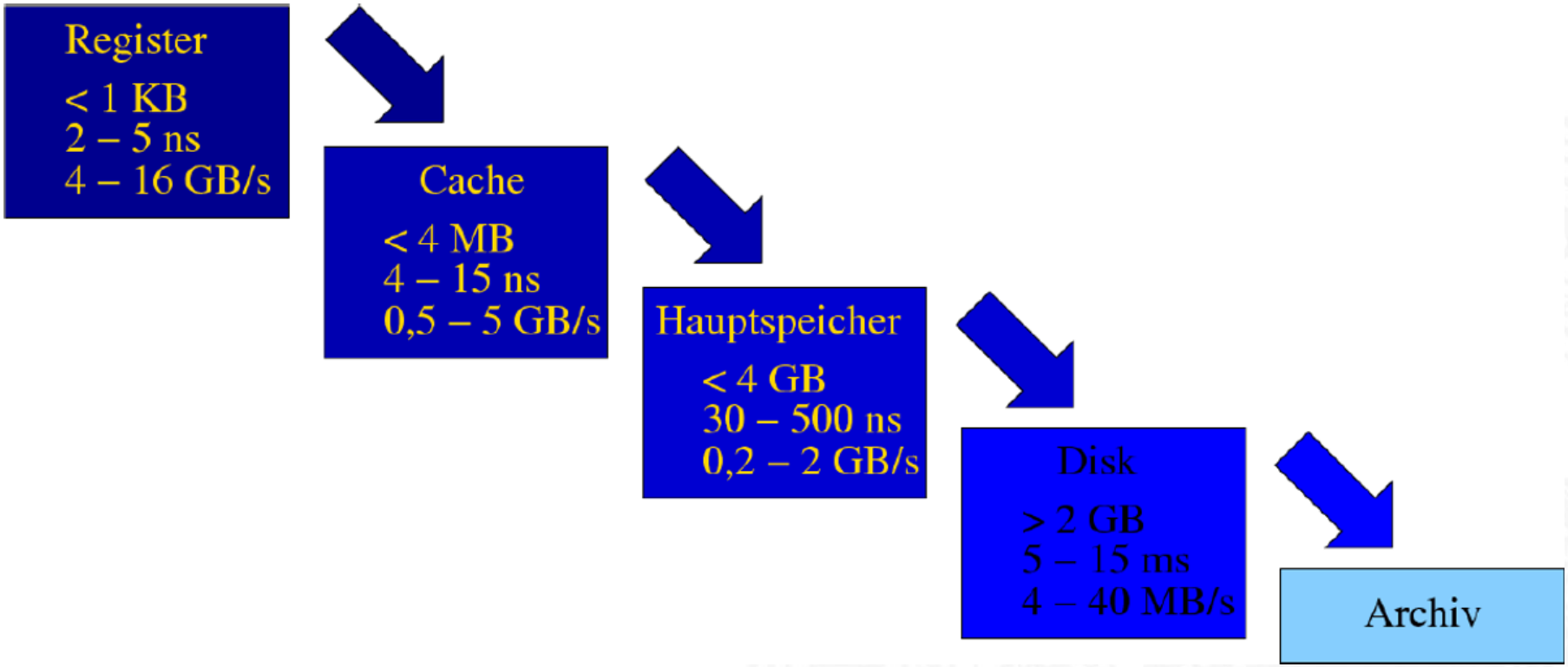


Speicherhierarchie (cont.)

Speichertypen

Speicher	Vorteile	Nachteile
Register	sehr schnell	sehr teuer
SRAM	schnell	teuer, große Chips
DRAM	hohe Integration	Refresh nötig, langsam
Platten	billig, Kapazität	sehr langsam, mechanisch
Beispiel	Hauptspeicher	Festplatte
Latenz	8 ns	4 ms
Bandbreite	$\approx$ 38,4 GByte/sec (triple Channel)	$\approx$ 750 MByte/sec (typisch $<$ 300)
Kosten	1 GByte, 5 €	1 TByte, 40 € (4 Ct./GByte)

Speicherhierarchie (cont.)



Speicherhierarchie (cont.)

Verwaltung der Speicherhierarchie

- Register $\leftrightarrow$ Memory
  - Compiler
  - Assembler-Programmierer
- Cache $\leftrightarrow$ Memory
  - Hardware
- Memory $\leftrightarrow$ Disk
  - Hardware und Betriebssystem (Paging)
  - Programmierer (Files)

Register

- Speicherung von Werten in der CPU
- Aufgaben
  - „*General-purpose*“ Register: für den (Assembler-) Programmierer sichtbare Register für Operanden und Speicheradressen
  - „*Flag*“ Register: Statusanzeigen im Datenprozessor (Status nach ALU Operationen) und Befehlsprozessor (Programm-, Interruptstatus. . .)
  - Befehlszähler, spezielle Adressierungsregister im Befehlsprozessor
  - Pipelineregister
  - . . .

Register (cont.)

- Zugriff mit *voller* Prozessorgeschwindigkeit
  . . . also möglichst viele Register
  - Kontextwechsel sind zeitaufwändig
  - mehr Bits für Adressierung

$\Rightarrow$ virtuelle Register

- beschränkte Anzahl von Registern (pro Kontext)
- Umschalten zwischen Registersätzen

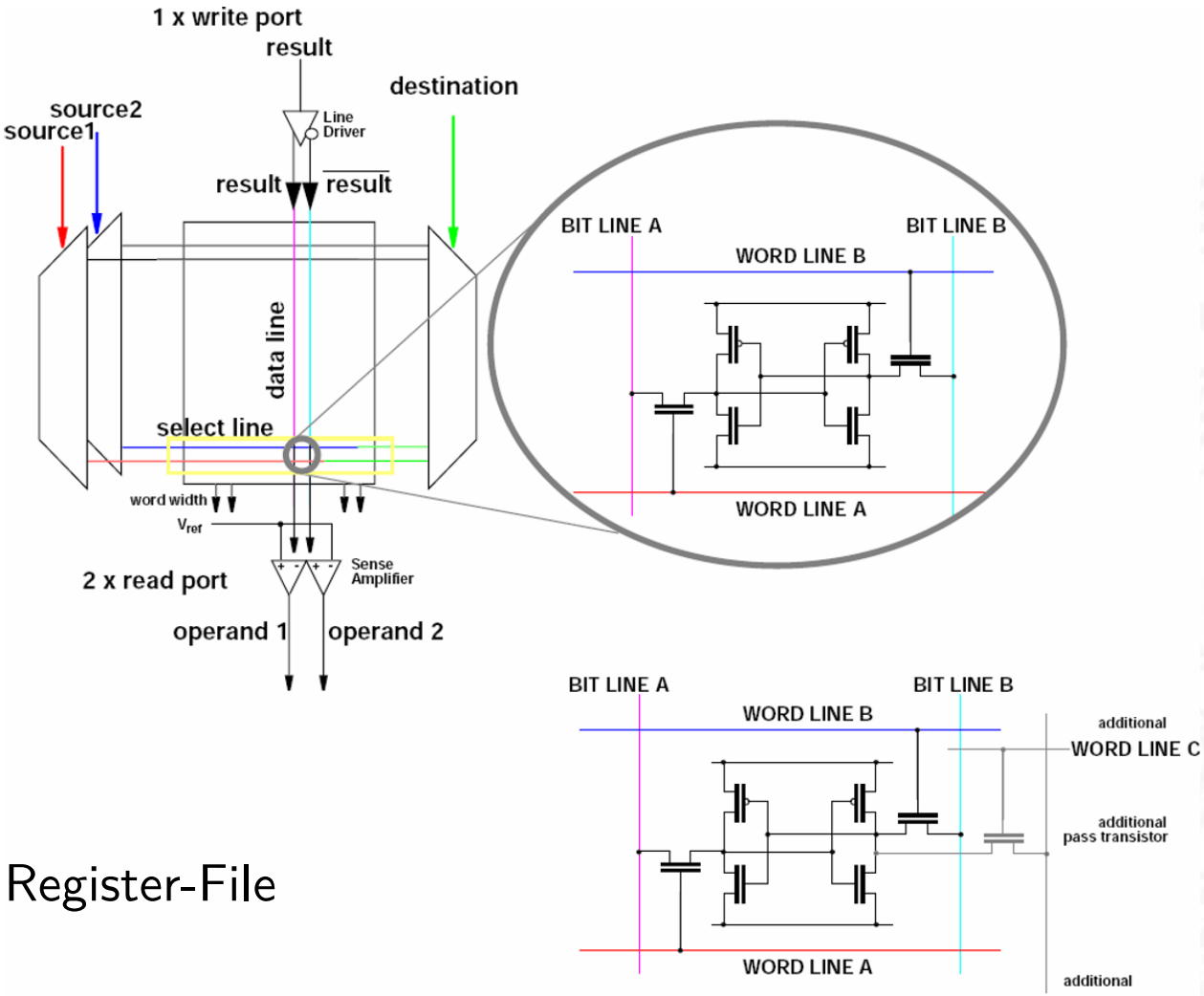
$\Rightarrow$ Registerfenster

- virtuelle Register dürfen sich überlappen
- effiziente Unterprogrammaufrufe: Parameterübergabe über solche Bereiche (sonst über Speicher)

Registerbänke

- (große) Arrays von einzeln adressierbaren Registern
- meist mit mehreren Ein- und Ausgabeports
- Realisierung
  - Register aus Flipflops und Multiplexerlogik
  - SRAM Realisierung große „*Register Files*“

Registerbänke (cont.)



Register-File

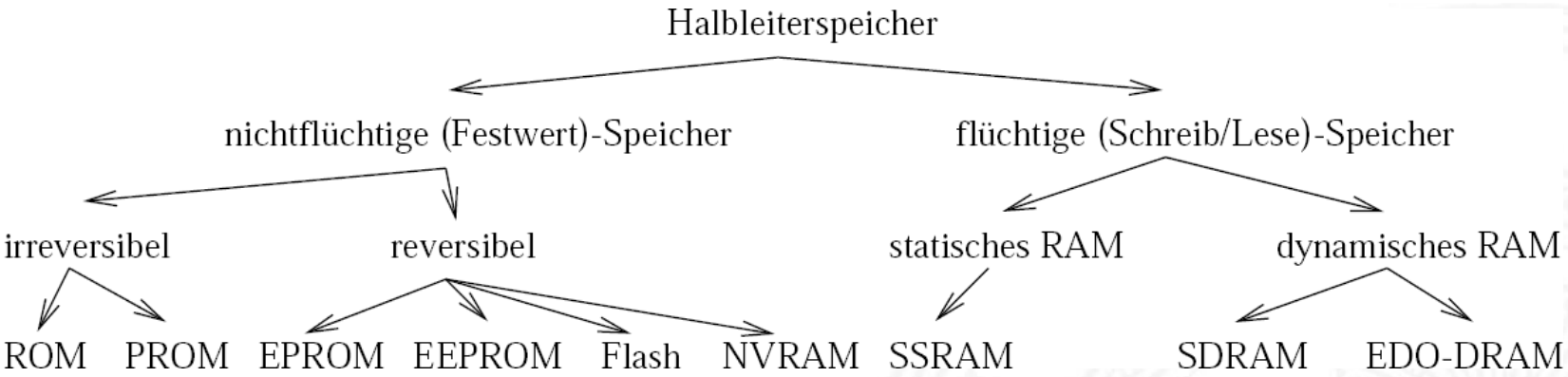
Hauptspeicher

Speichertypen

	SRAM	DRAM
Zugriffszeit	5...50 ns	60...100 ns t_{rac}
		20...300 ns t_{cac}
		110...180 ns t_{cyc}
Leistungsaufnahme	200...1300 mW	300...600 mW
Speicherkapazität	$<$ 72 Mbit	$<$ 4 Gbit
Preis	10 €/Mbit	0,2 Ct./Mbit
	$\Rightarrow$ Cache	$\Rightarrow$ Hauptspeicher

Hauptspeicher (cont.)

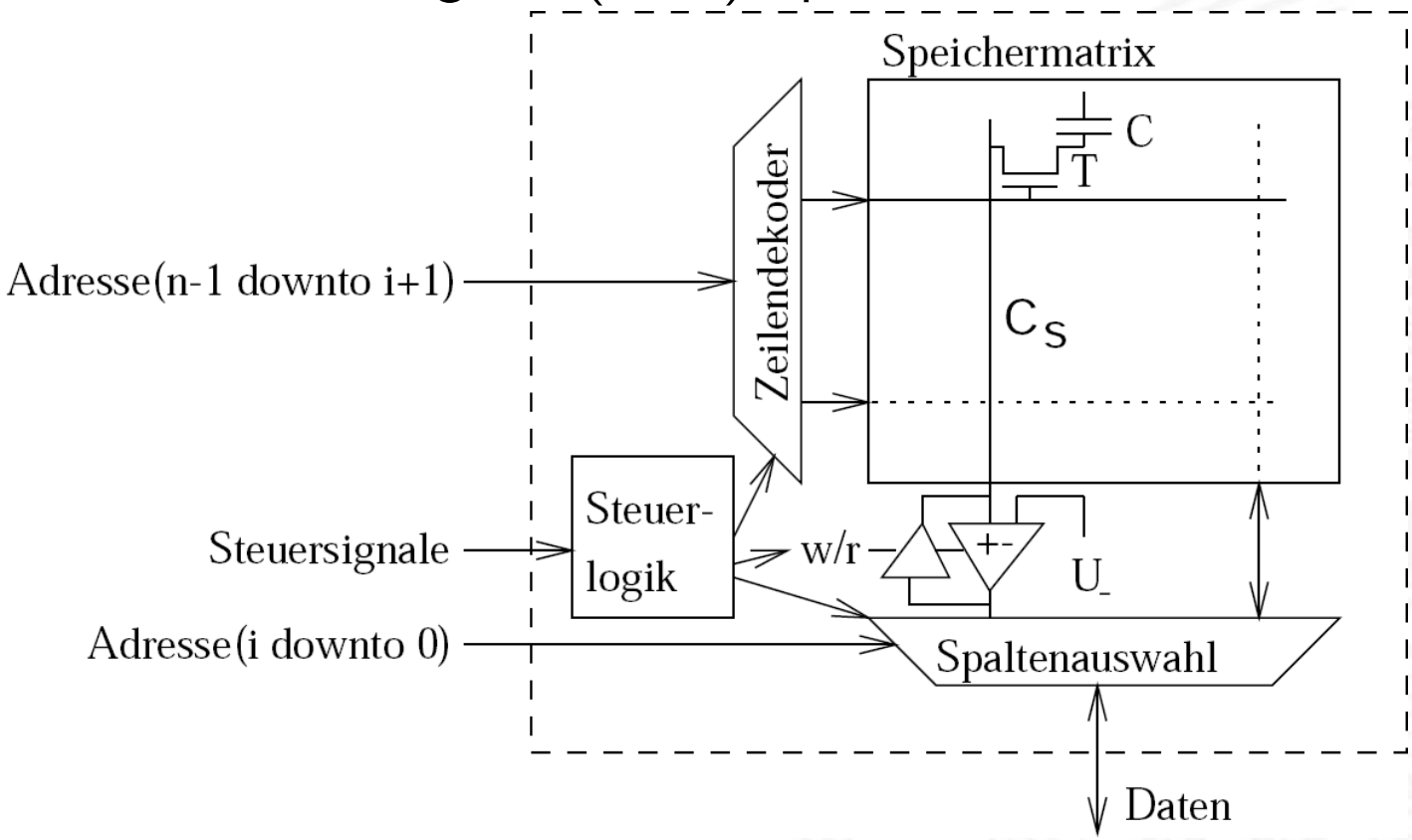
Arten von Halbleiterspeicher



Hauptspeicher: DRAM

Dynamische RAMs

- Matrix-Anordnung der (1-bit) Speicherzellen

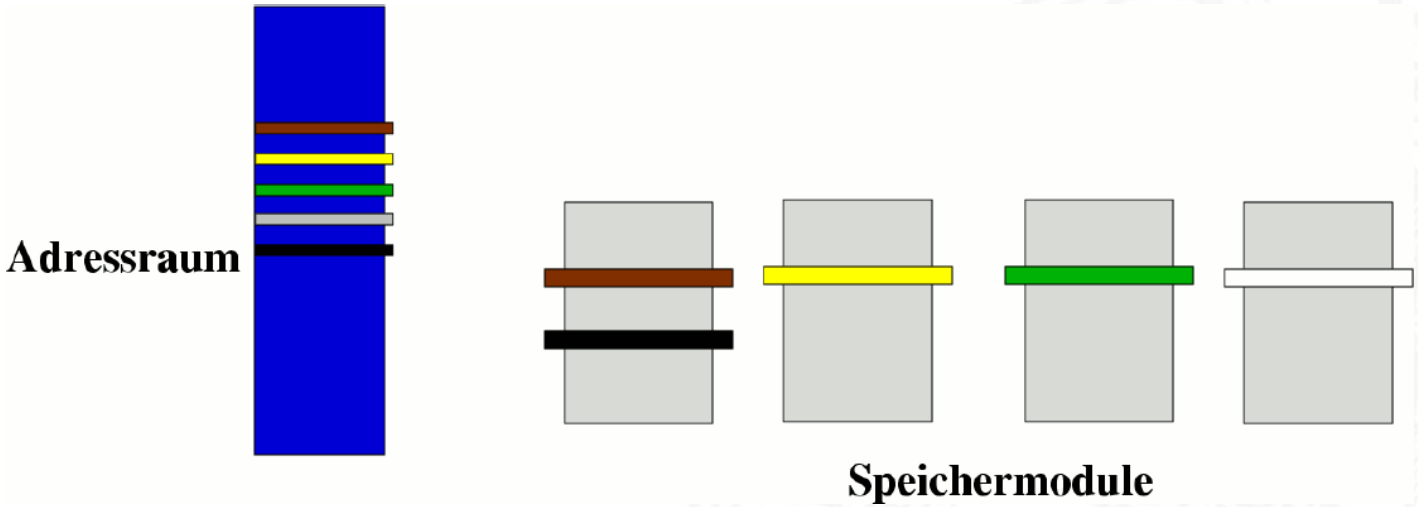


Hauptspeicher: DRAM (cont.)

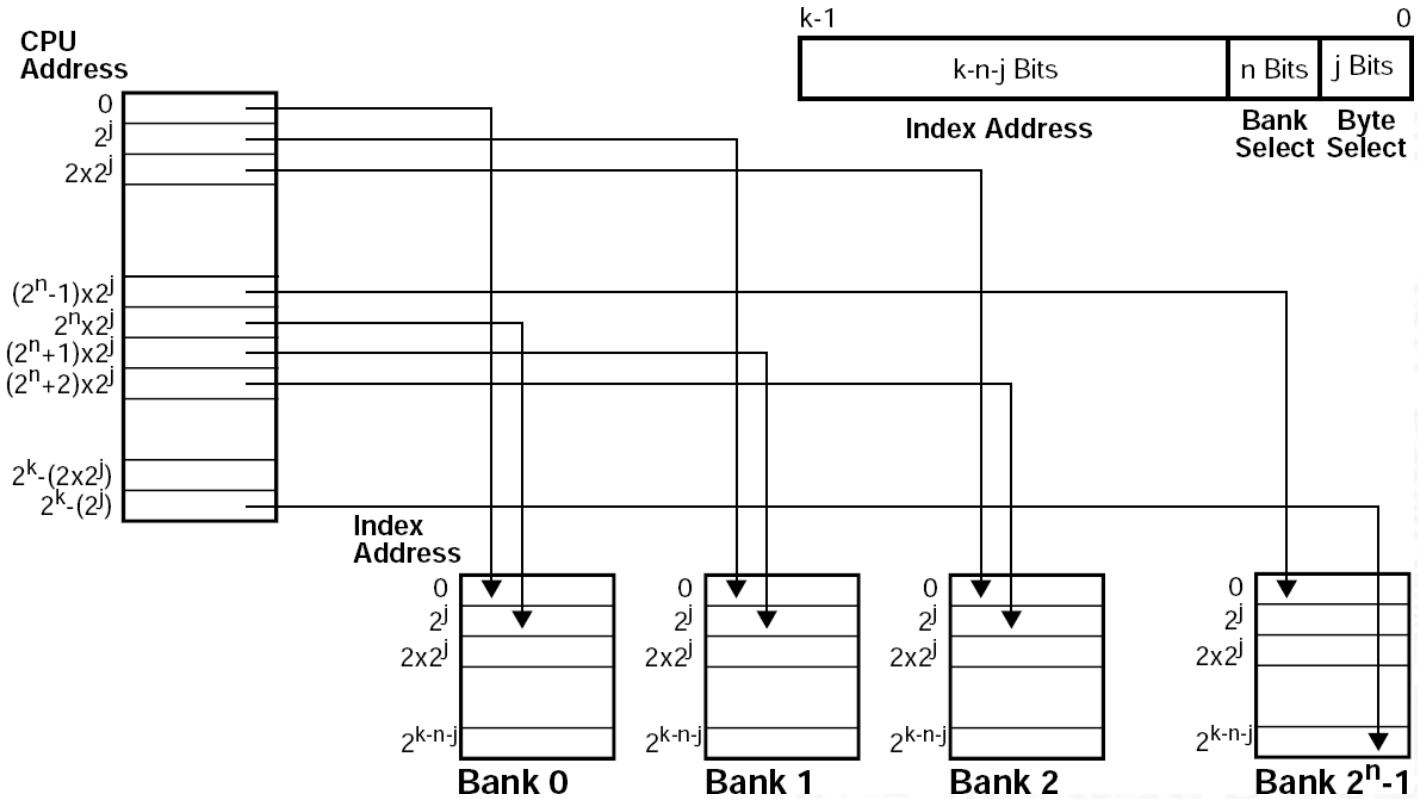
- serielle Zeilen- und Spaltenadressierung
  - \+ weniger Adressleitungen
- Speicher benötigen „Refresh“ nach Zugriff
- Zugriffszeit: Adresse und Daten senden
  Zykluszeit: minimale Zeit vom Beginn eines Zugriffs bis zum nächsten Speicherzugriff

Hauptspeicher: DRAM (cont.)

- „*Interleaving*“: Speichermodule verschränkt Adressieren
  - \+ bei Lokalität der Speicheradressen kann so die Zykluszeit verborgen werden



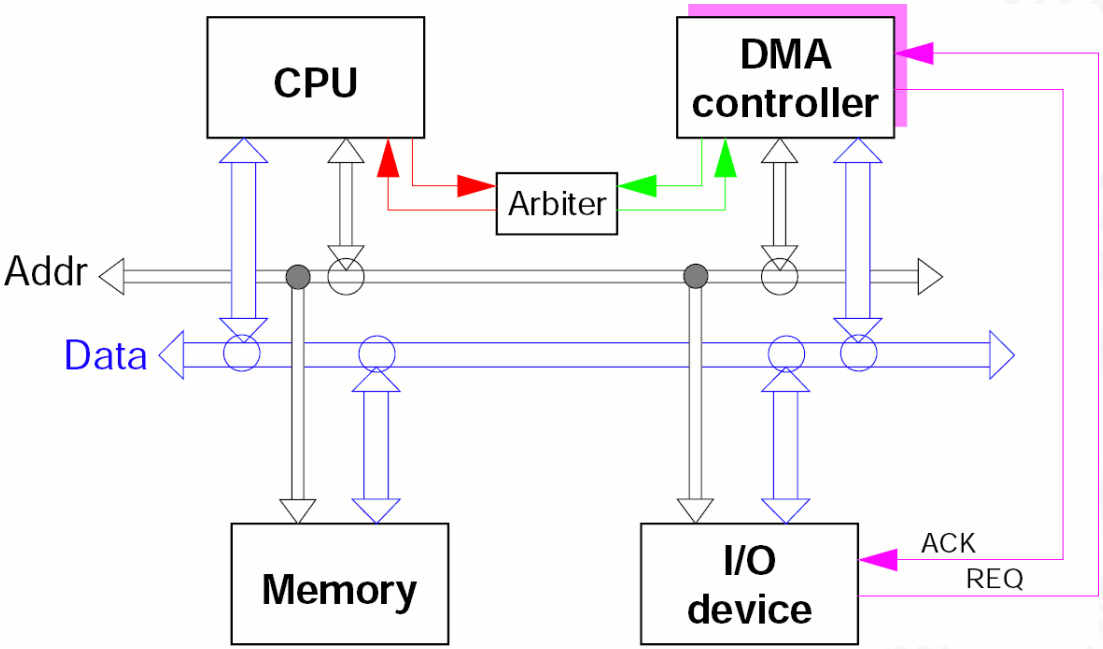
Hauptspeicher: DRAM (cont.)



Hauptspeicher: DMA

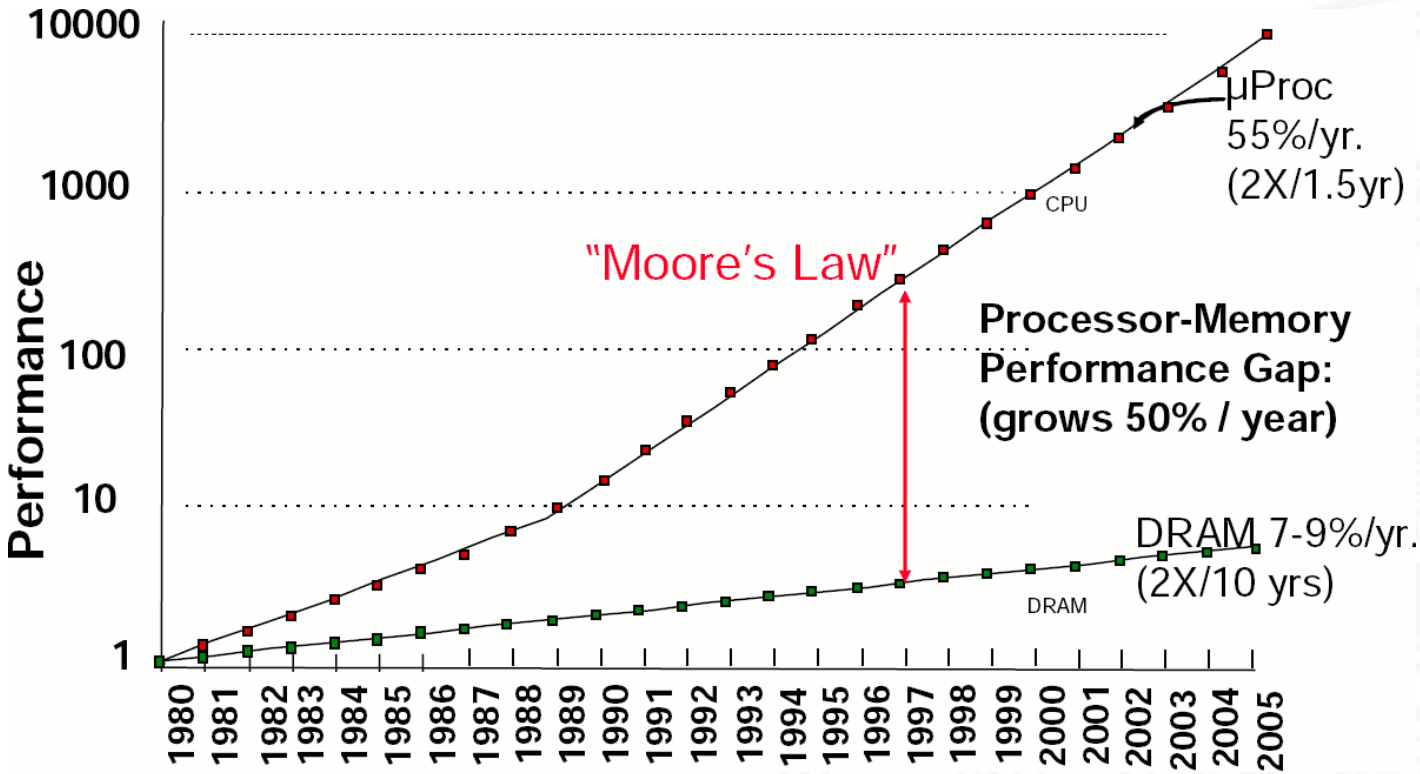
DMA – **D**irect **M**emory **A**ccess

- eigener Controller zum Datentransfer
- \+ Speicherzugriffe unabhängig von der CPU
- \+ CPU kann lokal (Register und Cache) weiterrechnen



Cache

- „*Memory Wall*“: DRAM zu langsam für CPU



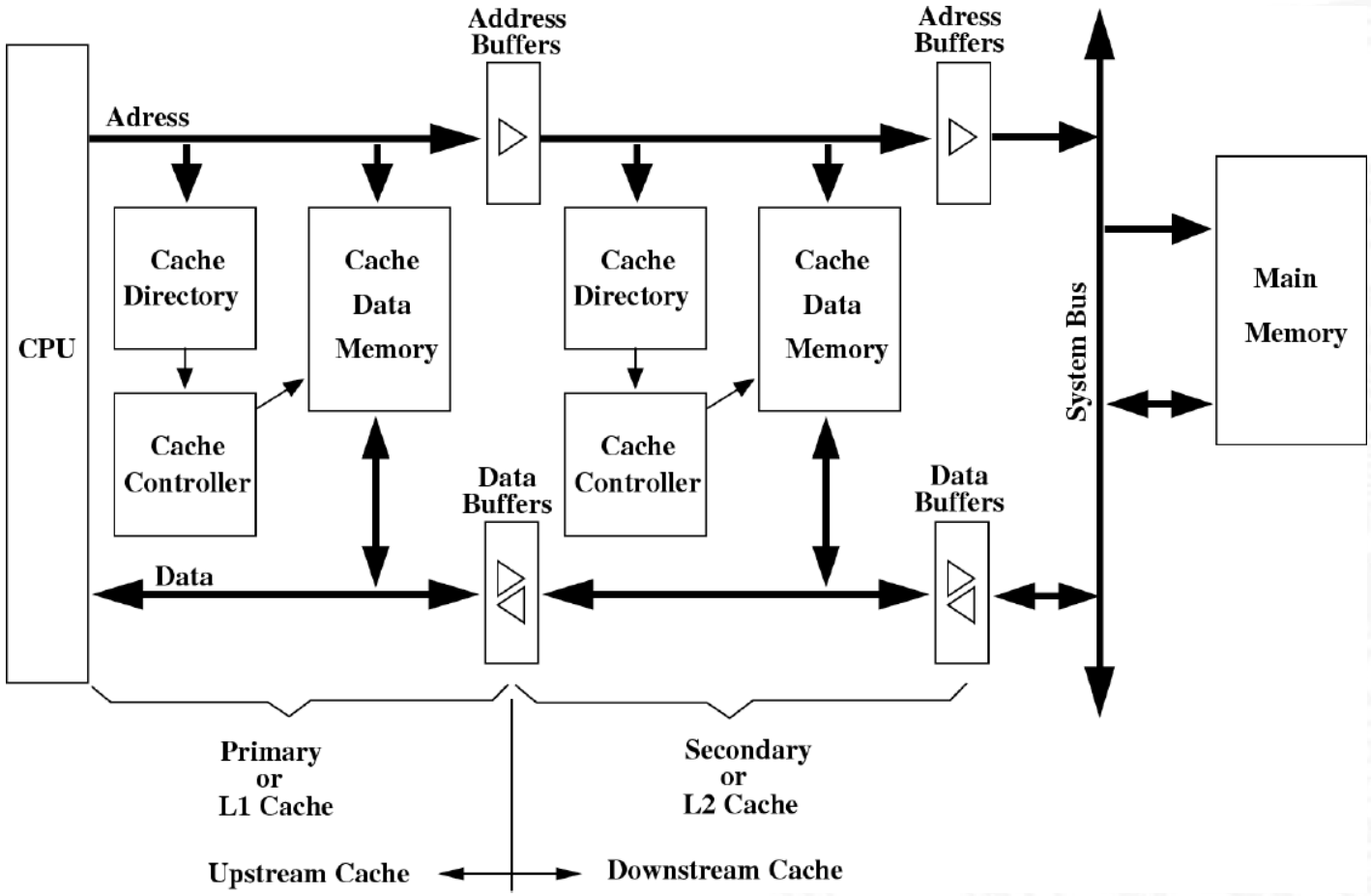
Cache (cont.)

⇒ Cache als schneller Zwischenspeicher zum Hauptspeicher

- technische Realisierung: SRAM
- transparenter Speicher:
  Cache ist für den Programmierer nicht sichtbar!
- `http://de.wikipedia.org/wiki/Cache`
  `http://en.wikipedia.org/wiki/Cache`

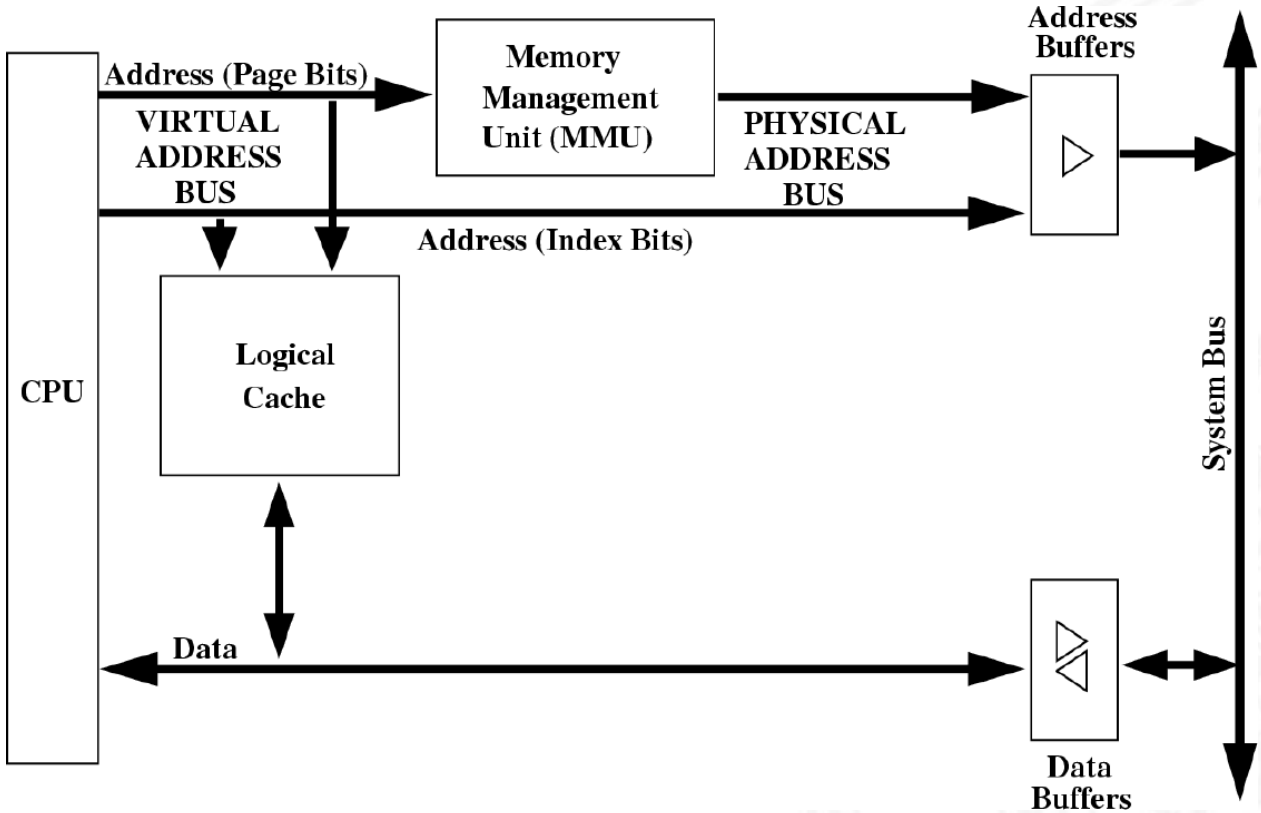
Cache – Position

- First- und Second-Level Cache



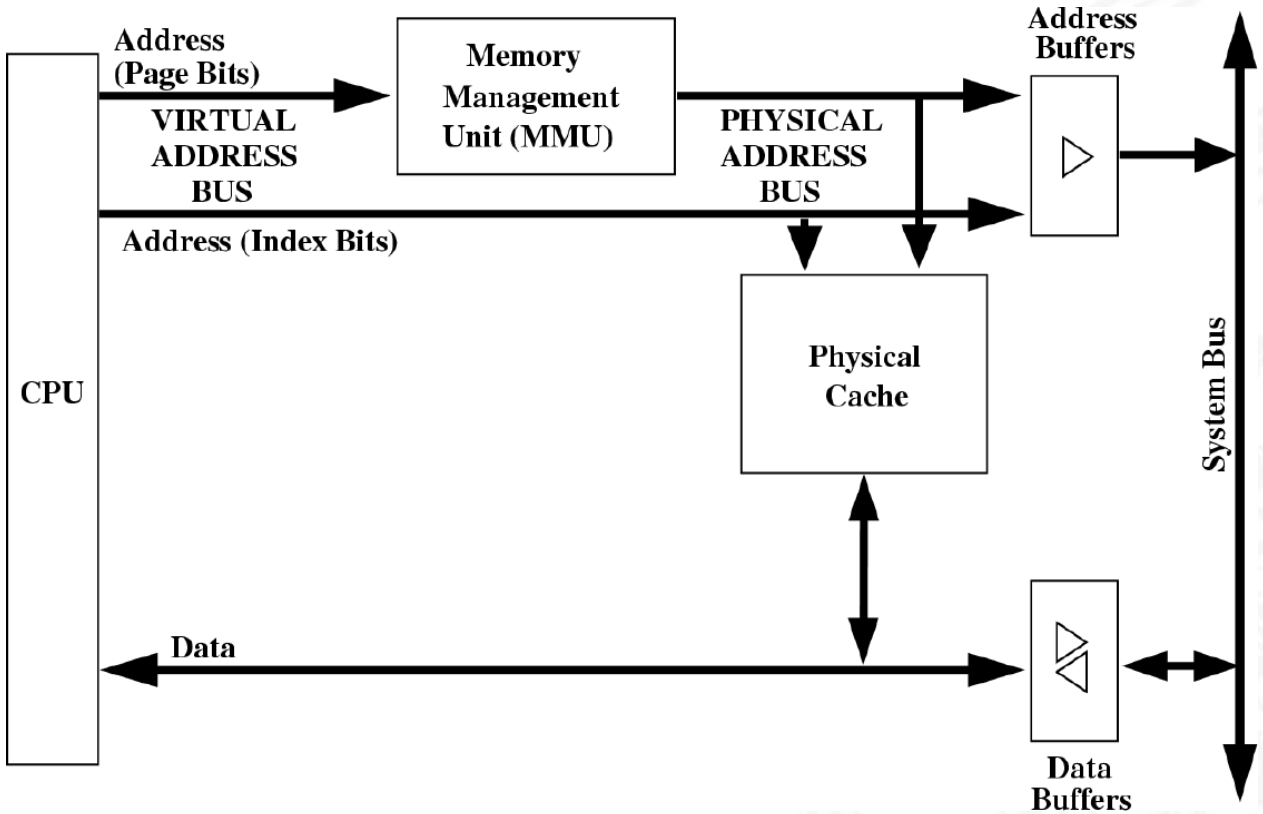
Cache – Position (cont.)

- Virtueller Cache
  - \+ Adressumrechnung durch MMU oft nicht nötig
  - – Cache leeren bei Kontextwechseln



Cache – Position (cont.)

- Physikalischer Cache
  - \+ Cache muss nie geleert werden
  - – Adressumrechnung durch MMU immer nötig



Cache – Strategie

Cachestrategie: *Welche Daten sollen in den Cache?*
Diejenigen, die bald wieder benötigt werden!

- *temporale Lokalität*:
  die Daten, die zuletzt häufig gebraucht wurden
- *räumliche Lokalität*:
  die Daten, die nahe den zuletzt gebrauchten liegen
- verschiedene Platzierungs-, Ersetzungs- und Rückschreibestategien für den Cache

Cache – Performanz

Cacheperformanz

- Begriffe

Treffer (Hit)		Zugriff auf Datum, ist bereits im Cache
Fehler (Miss)		–"– ist nicht –"–
Treffer-Rate	R_{Hit}	Wahrscheinlichkeit, Datum ist im Cache
Fehler-Rate	R_{Miss}	$1 - R_{Hit}$
Hit-Time	T_{Hit}	Zeit, bis Datum bei Treffer geliefert wird
Miss-Penalty	T_{Miss}	zusätzlich benötigte Zeit bei Fehler

- Mittlere Speicherzugriffszeit $= T_{Hit} + R_{Miss} \cdot T_{Miss}$
- Beispiel

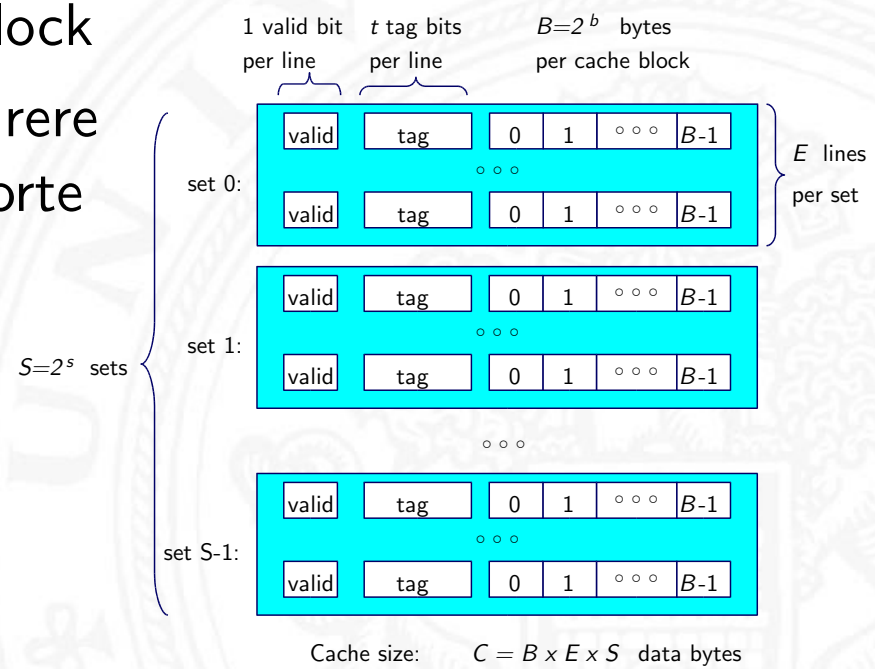
  $T_{Hit} = 1\,Takt,\ T_{Miss} = 20\,Takte,\ R_{Miss} = 5\,\%$

  Mittlere Speicherzugriffszeit $= 2\,Takte$

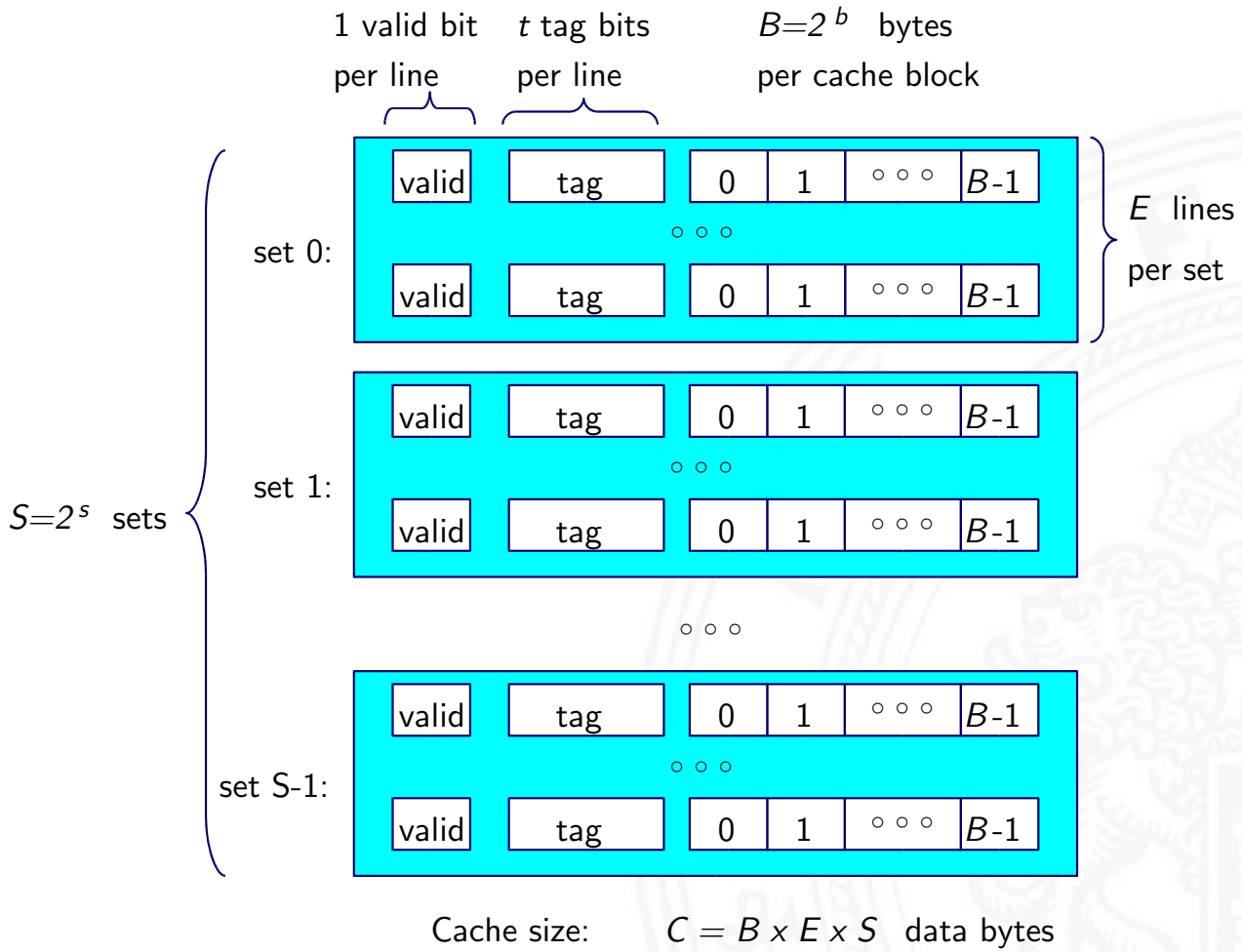
Cache – Organisation

Allgemeine Organisation eines Cache Speichers

- Cache ist ein Array von Speicher-Bereichen („*sets*“)
- Jeder Bereich enthält eine oder mehrere Cachezeilen
- Jede Zeile enthält einen Datenblock
- Jeder Datenblock beinhaltet mehrere (aufeinanderfolgende) Speicherworte



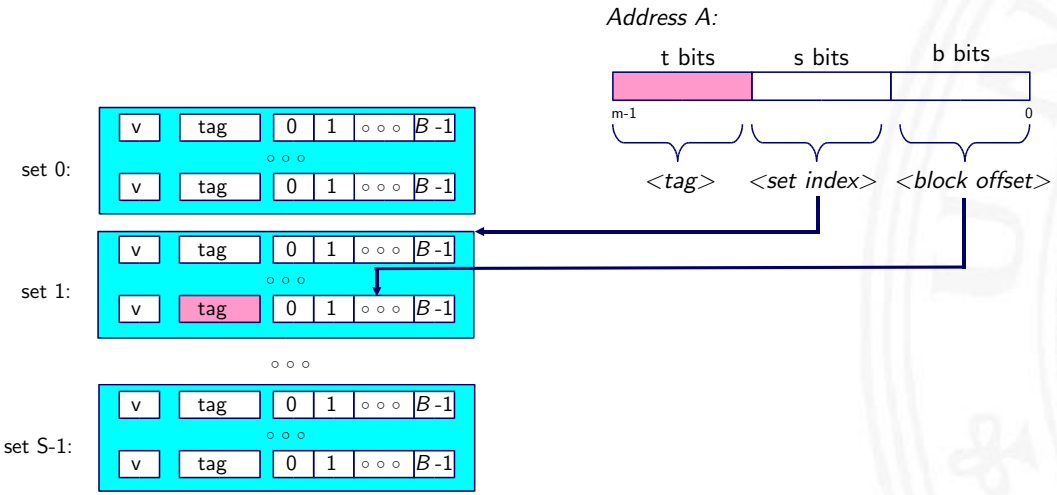
Cache – Organisation (cont.)



Cache – Organisation (cont.)

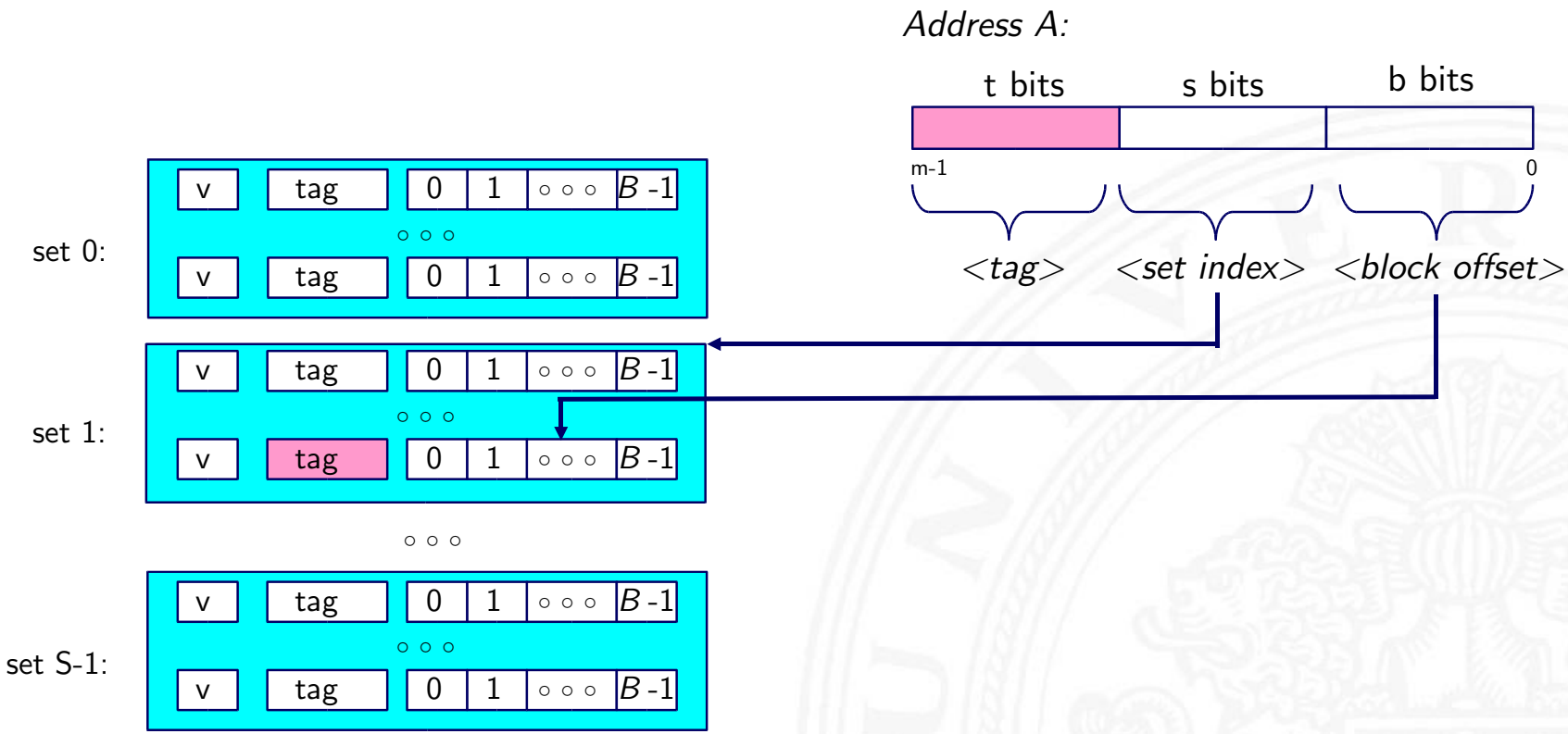
Cache Zugriff

- Das Wort der Adresse A liegt im Cache, wenn die „*tag*“ *Bits* in einer gültigen Zeile $\langle valid \rangle$ im aktuellen Bereich $\langle set\ index \rangle$ mit dem $\langle tag \rangle$ der Adresse A übereinstimmen
- Das gesuchte Wort steht im Cache an der Stelle $\langle block\ offset \rangle$ hinter dem Anfang des Blocks



◂ Direct Mapped
◂ Set Associative

Cache – Organisation (cont.)



Cache – Adressierung

- *Welchen Platz im Cache belegt ein Datum des Hauptspeichers?*
- drei Verfahren

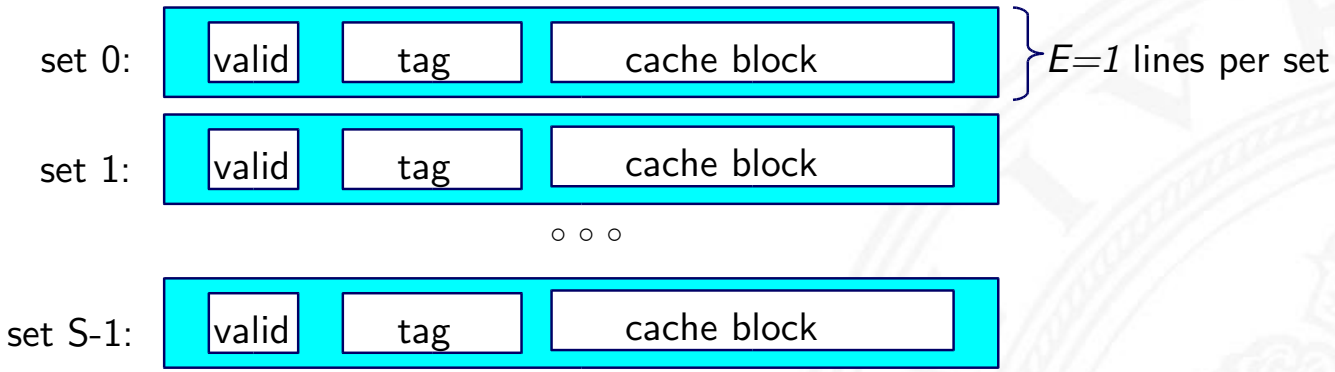
  Direkt abgebildet / direct mapped jeder Speicheradresse ist genau eine Cache-Speicherzelle zugeordnet

  n-fach Bereichsassoziativ / set associative jeder Speicheradresse ist eine von E möglichen Cache-Speicherzellen zugeordnet

  Voll-Assoziativ jeder Speicheradresse kann jede beliebige Cache-Speicherzelle zugeordnet werden

Cache: direkt abgebildet

- Jeder Adresse ist genau eine Speicherzelle im Cache zugeordnet
- Verfügt über genau 1 Zeile pro Bereich S Bereiche (**S**ets)



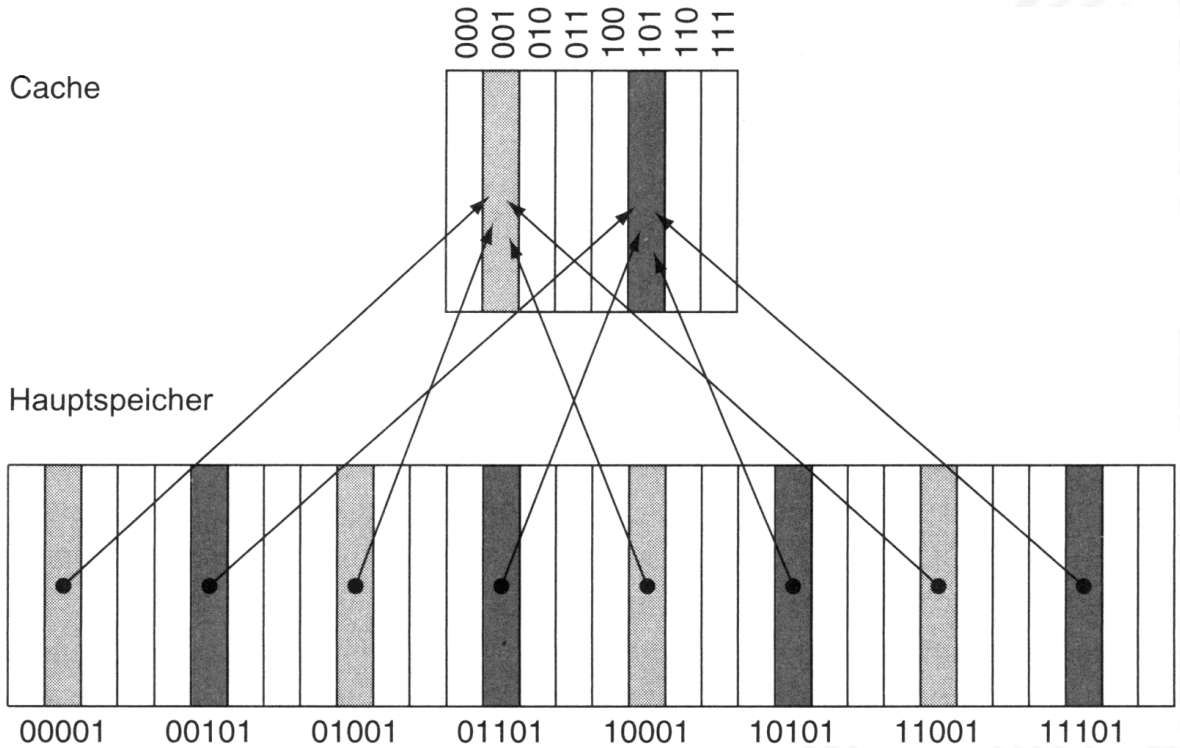
+ einfachste Cache-Art
+ große Caches möglich
– Effizienz, z.B. Zugriffe auf A, $A + n \cdot S \ldots$

Cache: direkt abgebildet (cont.)

Adressabbildung über Bereichsausschnitte

▸ Cache Zugriff

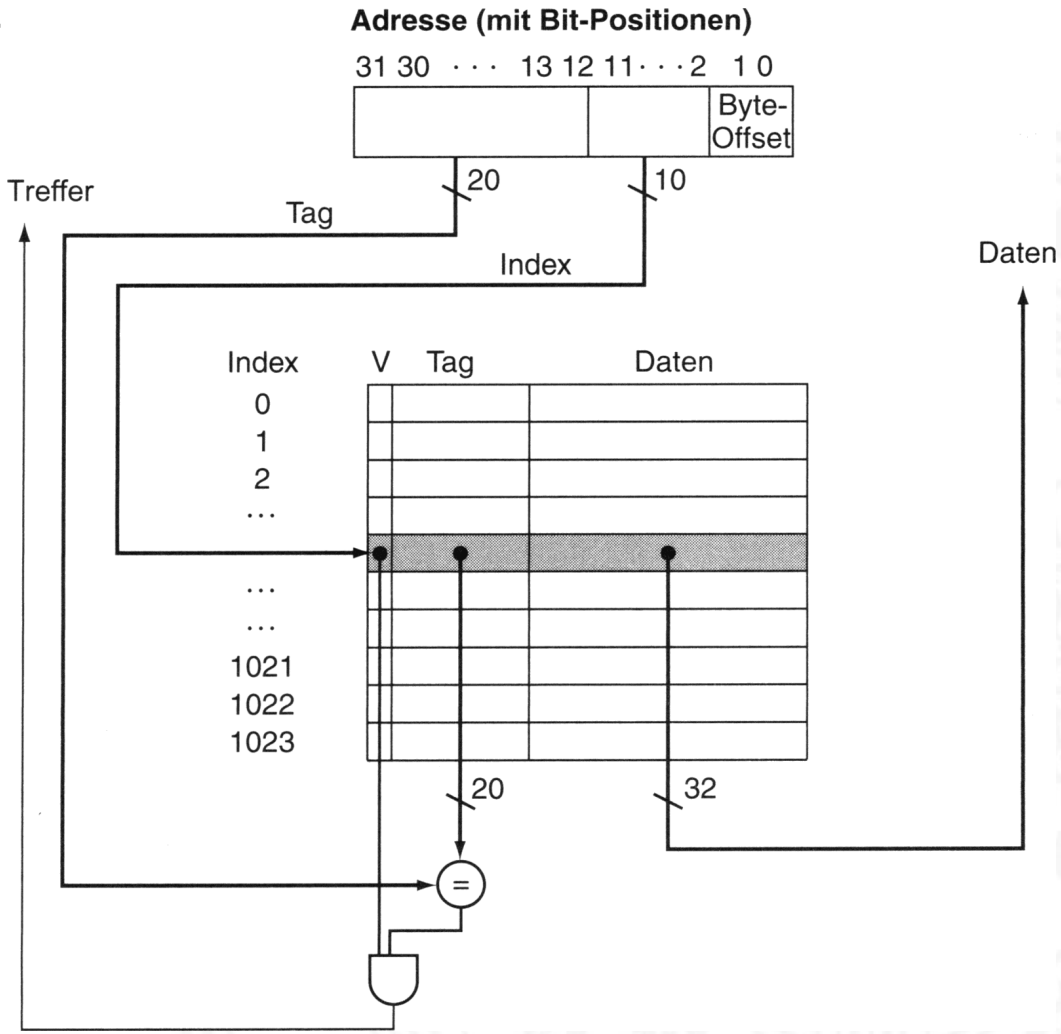
- $A_{Cache} = A_{Memory} \bmod S$, S Cache-Einträge
  $Tag = A_{Memory} \;/\; S$



Cache: direkt abgebildet (cont.)

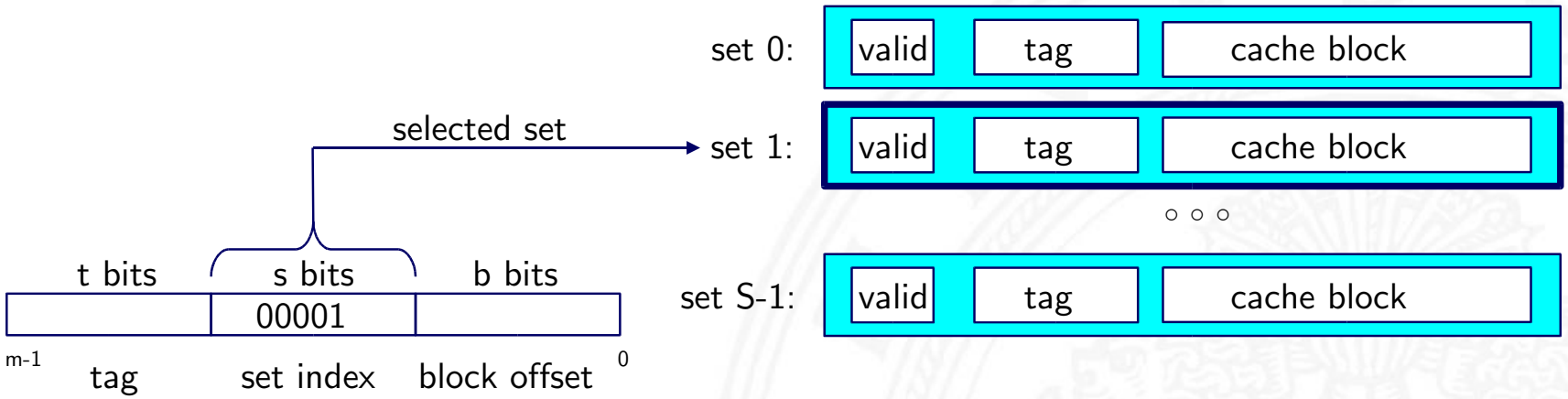
Direct Mapped Cachezugriff

- Rückabbildung:
  A_{Memory} aus *Tag* und
  A_{Cache} bestimmen



Cache: direkt abgebildet (cont.)

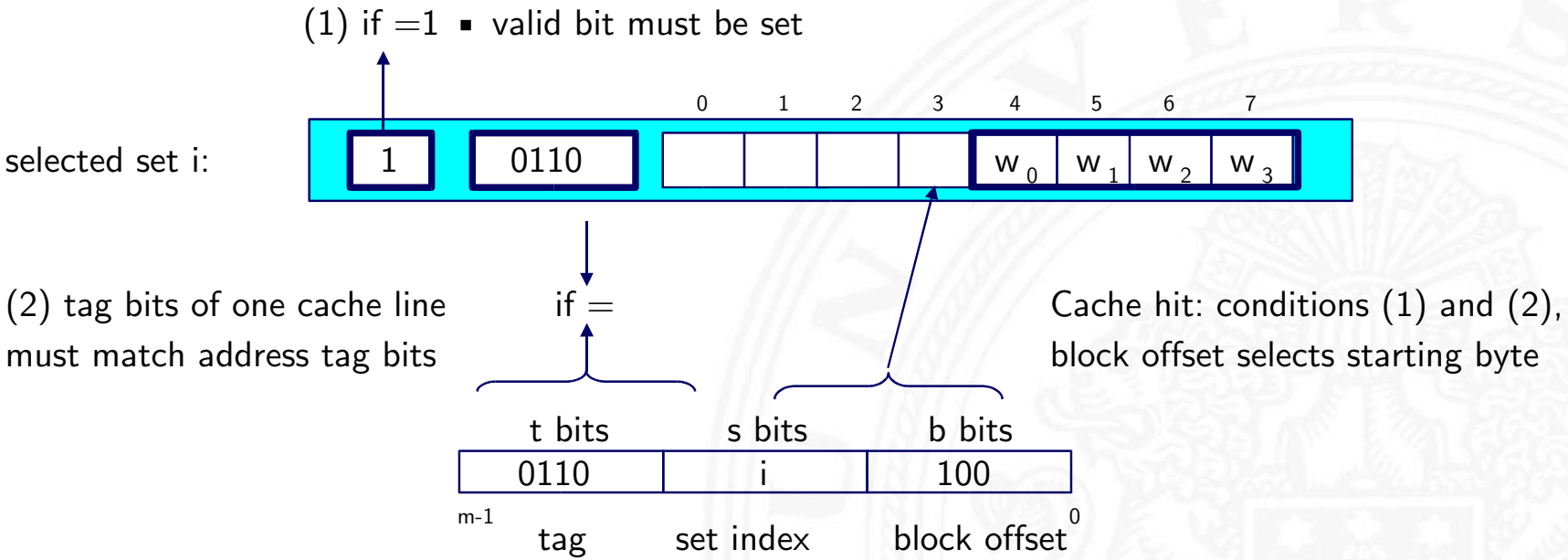
1. Bereichsauswahl
   - Bereichsindex-Bits $\langle set\ index \rangle$ legen den Bereich (die Cachezeile A_{Cache}) fest



Cache: direkt abgebildet (cont.)

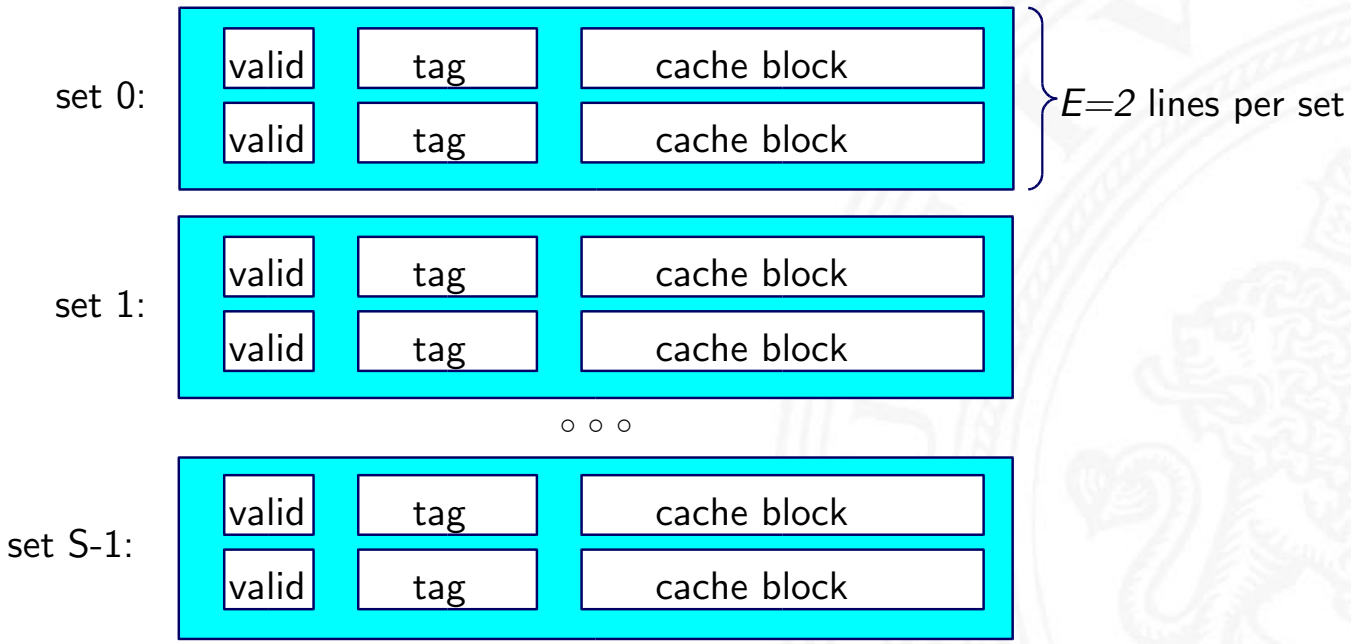
2. „Line matching“ und Wortselektion

- „Line matching“: gültige Zeile bei übereinstimmendem $\langle tag \rangle$
- Wortselektion: dann extrahiere das Wort $\langle block\ offset \rangle$



Cache: bereichsassoziativ

- Jeder Adresse des Speichers ist ein Cachebereich S mit E Speicherzellen zugeordnet
- Verfügen über mehr als eine Zeile pro Bereich (z.B.: $E = 2$)



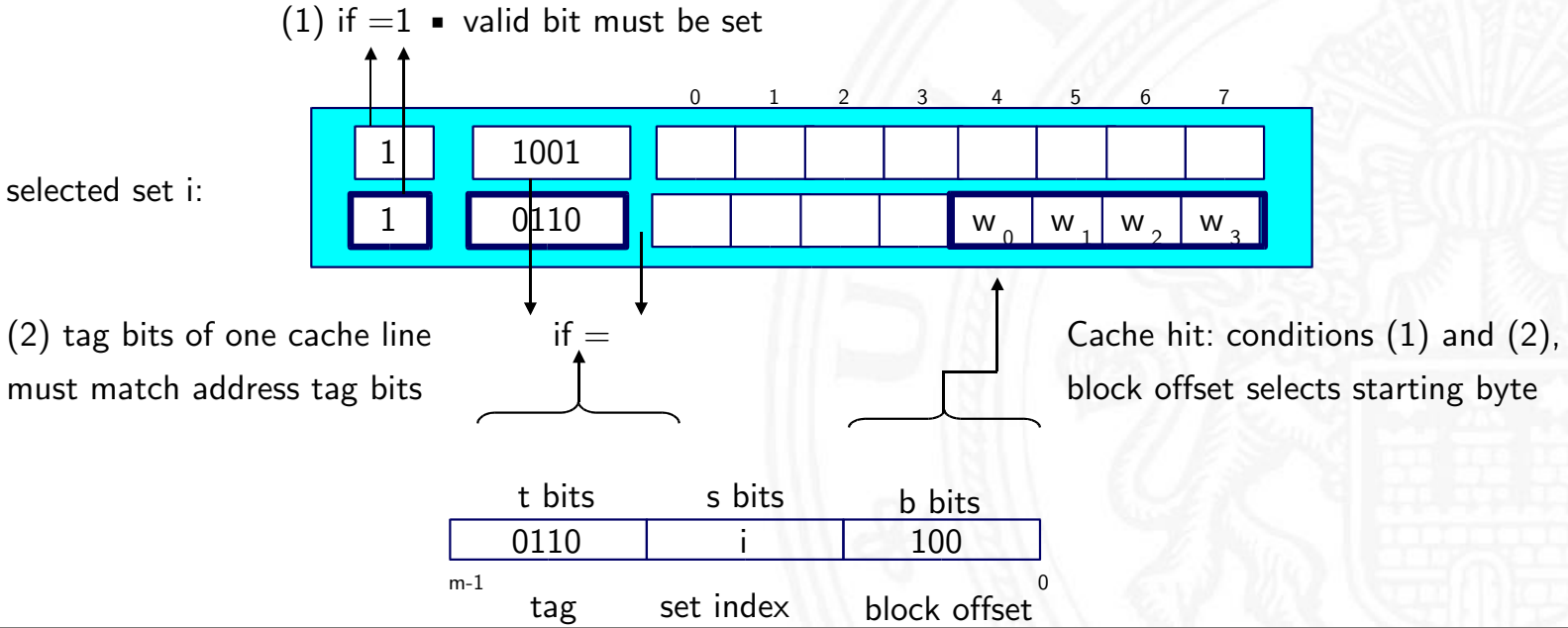
Cache: bereichsassoziativ (cont.)

Adressabbildung über Bereichsausschnitte

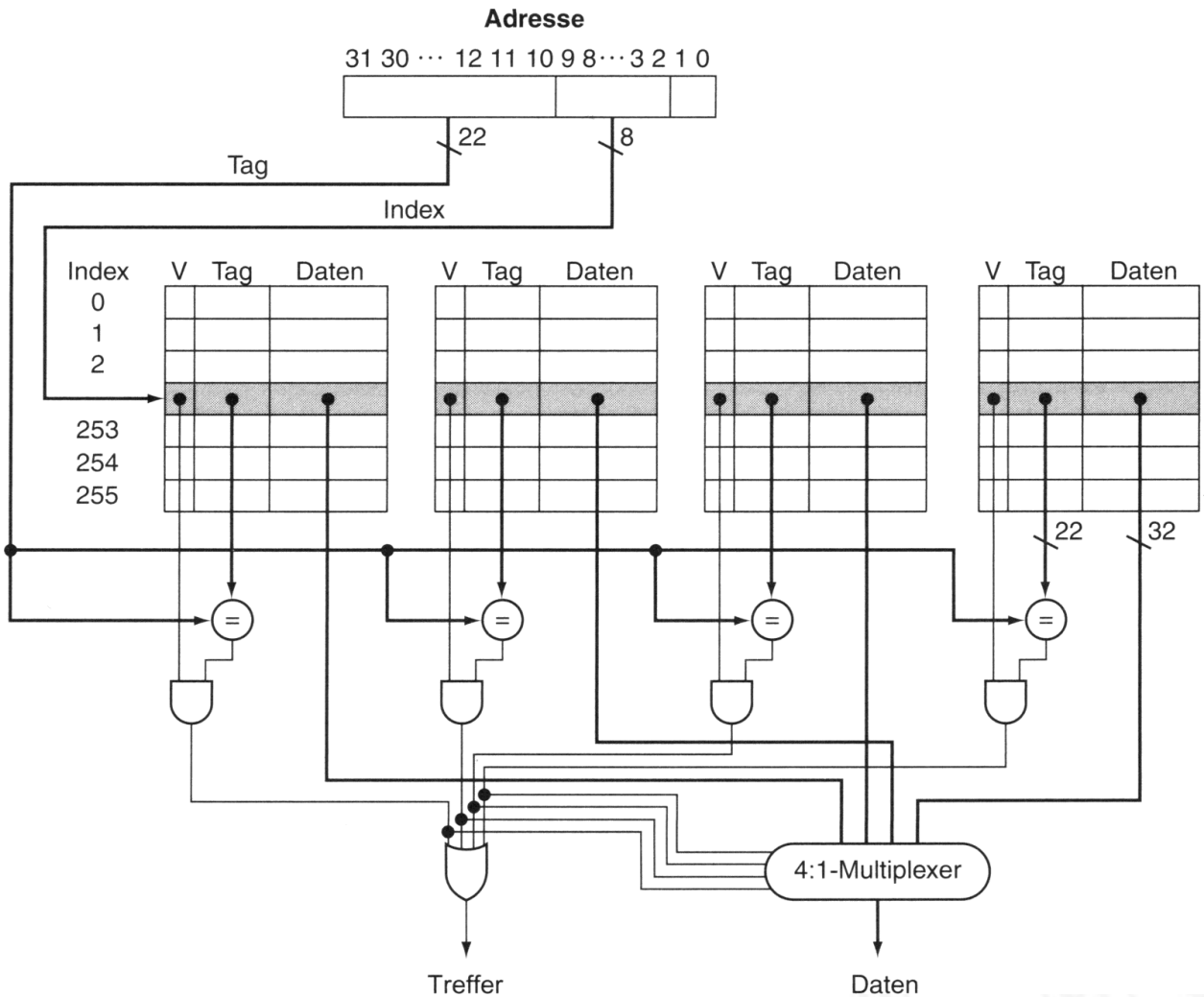
▸ Cache Zugriff

Set Associative Cachezugriff

1. Bereichsauswahl, wie bei „direkt abgebildetem Cache“
2. „Line matching“ und Wortselektion
   - mehrere Tags vergleichen: erhöhter Hardwareaufwand



Cache: bereichsassoziativ (cont.)

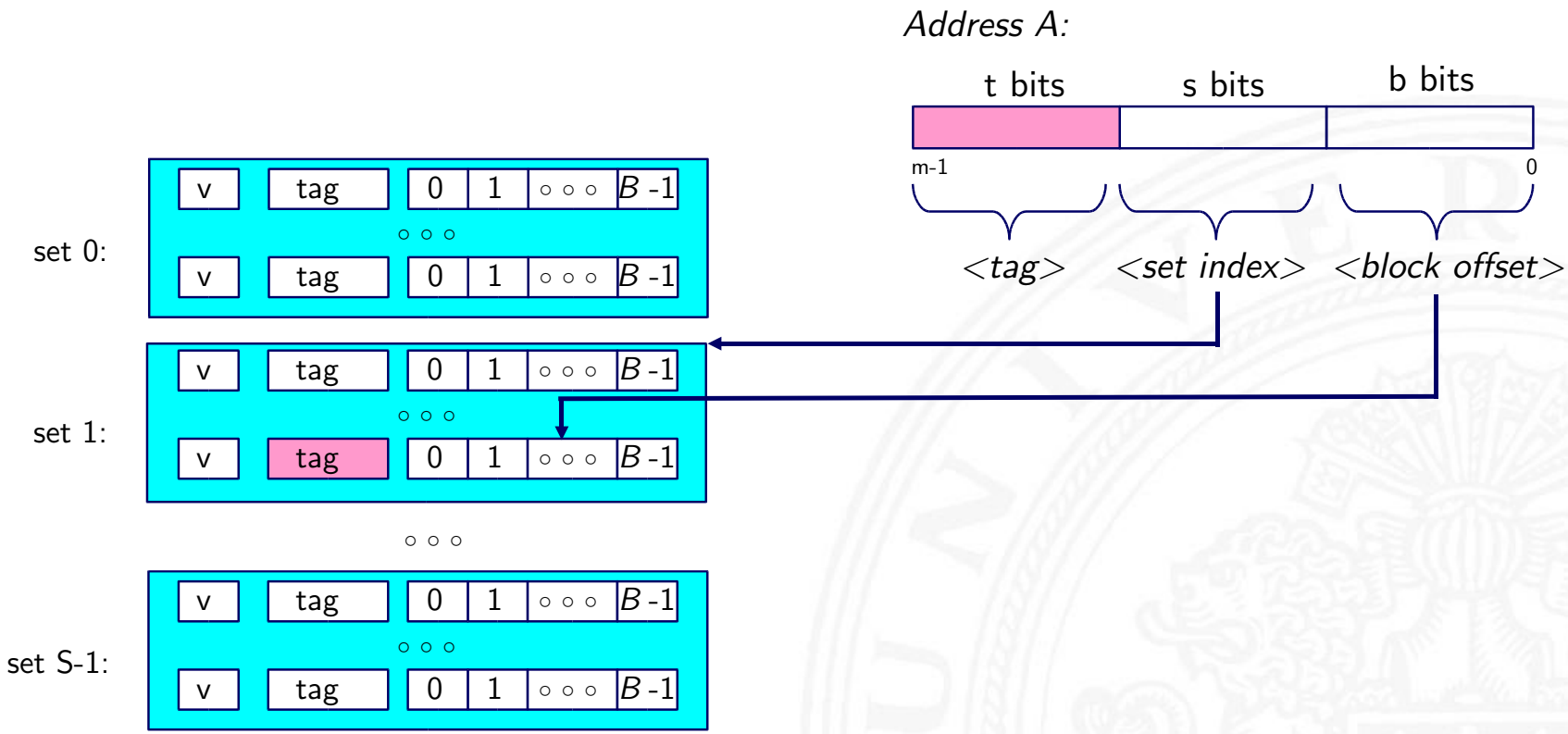


Cache: voll-assoziativ

- Jeder Adresse des Speichers kann jede beliebige Speicherzelle des Caches zugeordnet werden
- Spezialfall: nur ein Cachebereich S

– benötigt E-Vergleicher
– nur für sehr kleine Caches realisierbar

Cache – Dimensionierung



- Parameter: S, B, E

Cache – Dimensionierung (cont.)

- Cache speichert immer größere Blöcke / „*Cache-Line*“
- Wortauswahl durch $\langle$*block offset*$\rangle$ in Adresse

\+ nutzt räumliche Lokalität aus

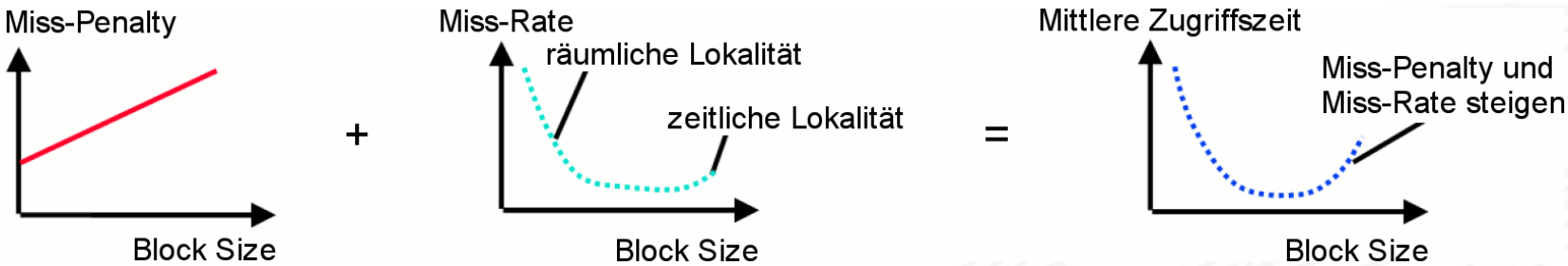
\+ Breite externe Datenbusse

\+ nutzt Burst-Adressierung des Speichers: Adresse nur für erstes Wort vorgeben, dann automatisches Inkrement

\+ kürzere interne Cache-Adressen

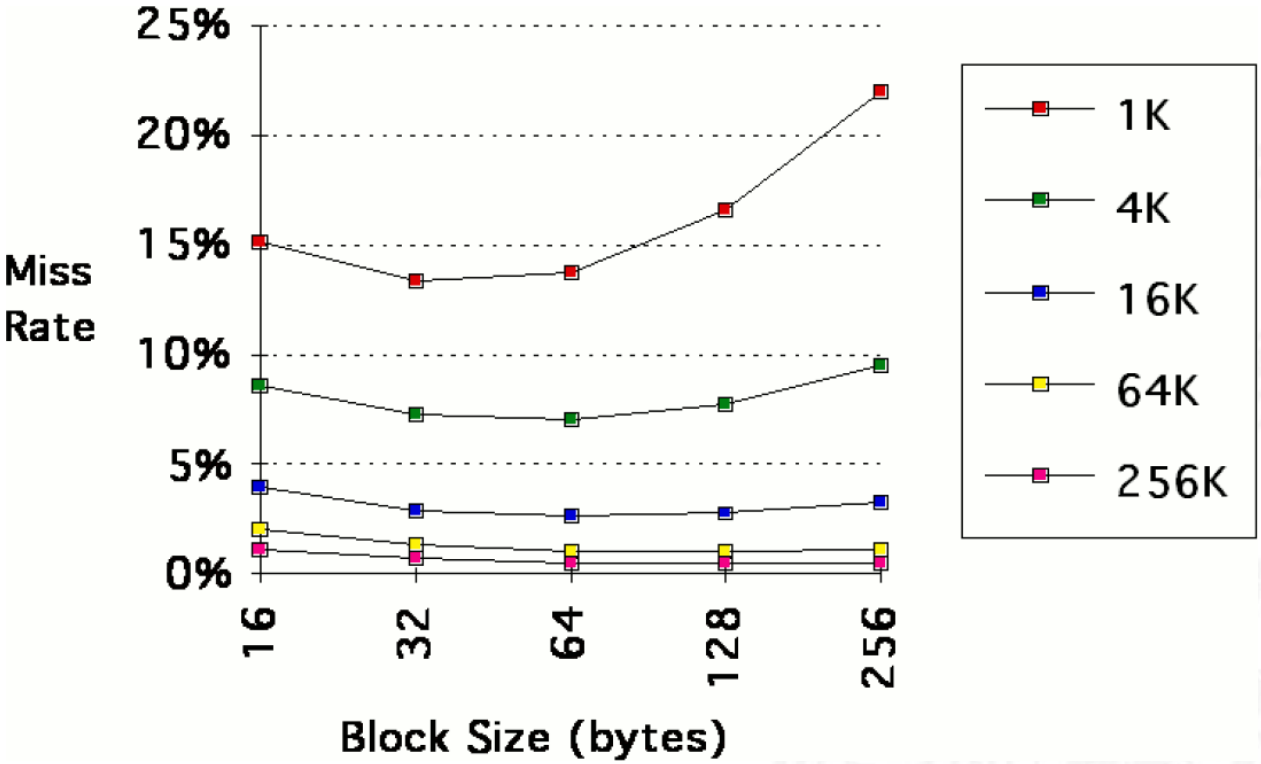
Cache – Dimensionierung (cont.)

Cache- und Block-Dimensionierung



- Blockgröße klein, viele Blöcke
  - \+ kleinere Miss-Penalty
  - \+ temporale Lokalität
  - – räumliche Lokalität
- Blockgröße groß, wenig Blöcke
  - – größere Miss-Penalty
  - – temporale Lokalität
  - \+ räumliche Lokalität

Cache – Dimensionierung (cont.)



- Block-Size: 32...128 Byte
  L1-Cache: 4...256 KByte
  L2-Cache: 256...4096 KByte

Cache – Dimensionierung (cont.)

- zusätzliche Software-Optimierungen
  - Ziel: Cache Misses reduzieren
  - Schleifen umsortieren, verschmelzen...
  - Datenstrukturen zusammenfassen...

Cache Ersetzungsstrategie

Wenn der Cache gefüllt ist, welches Datum wird entfernt?

- zufällige Auswahl
- LRU (**L**east **R**ecently **U**sed):
  der „älteste“ nicht benutzte Cache Eintrag
  - echtes LRU als Warteschlange realisiert
  - Pseudo LRU mit baumartiger Verwaltungsstruktur:
    Zugriff wird paarweise mit einem Bit markiert,
    die Paare wieder zusammengefasst usw.
- LFU (**L**east **F**requently **U**sed):
  der am seltensten benutzte Cache Eintrag
  - durch Zugriffszähler implementiert

Cache Schreibstrategie

Wann werden modifizierte Daten des Cache zurückgeschrieben?

- Write-Through: beim Schreiben werden Daten sofort im Cache und im Hauptspeicher modifiziert
  - \+ andere Bus-Master sehen immer den „richtigen“ Speicherinhalt: *Cache-Kohärenz*
  - – Werte werden unnötig oft in Speicher zurückgeschrieben
- Write-Back: erst in den Speicher schreiben, wenn Datum des Cache ersetzt werden würde
  - \+ häufig genutzte Werte (z.B. lokale Variablen) werden nur im Cache modifiziert
  - – Cache-Kohärenz ist nicht gegeben
  - ⇒ spezielle Befehle für „Cache-Flush“
  - ⇒ „non-cacheable“ Speicherbereiche

Cache Kohärenz

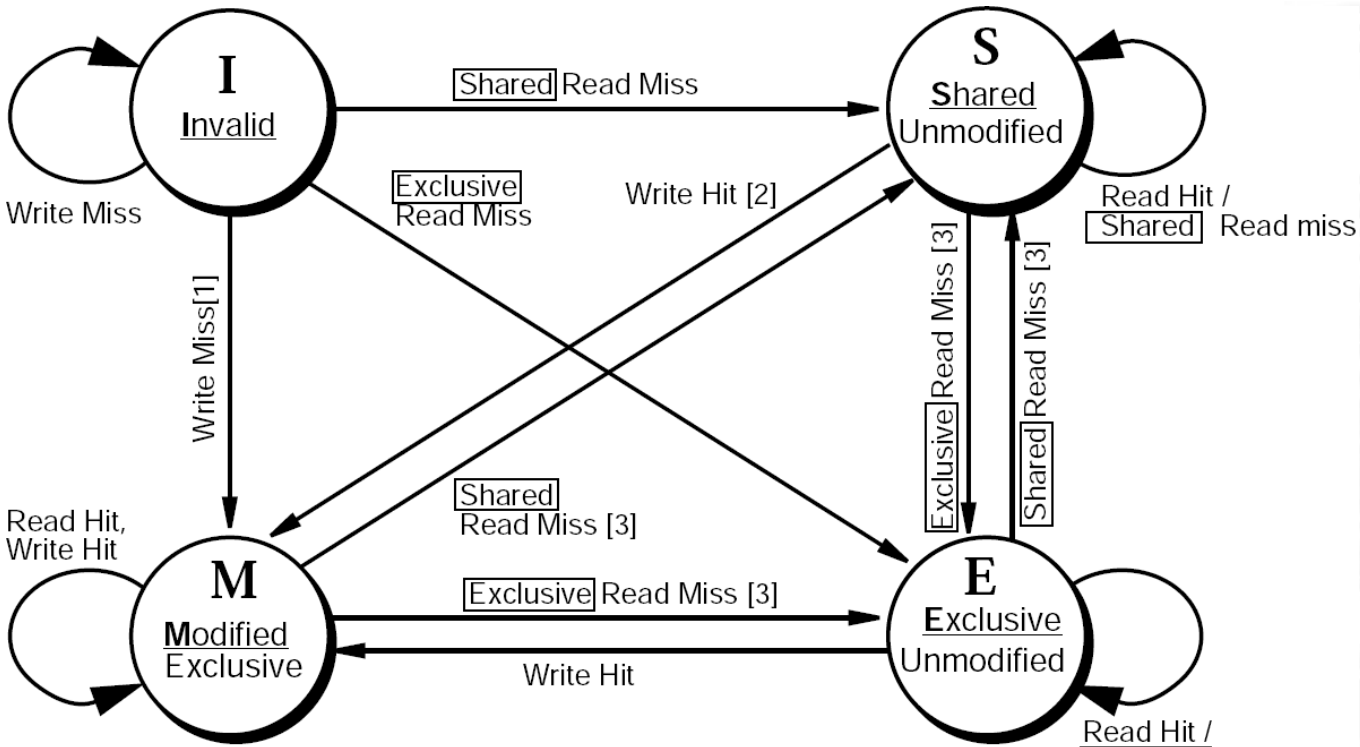
- Daten zwischen Cache und Speicher konsistent halten
- notwendig wenn auch andere Einheiten (Bus-Master) auf Speicher zugreifen können
- Harvard-Architektur hat getrennte Daten- und Instruktions-Speicher: einfacherer Instruktions-Cache
  - Instruktionen sind read-only
  - kein Cache-Kohärenz Problem
- „Snooping“
  - Cache „lauscht“ am Speicherbus
  - externer Schreibzugriff $\Rightarrow$ Cache aktualisieren / ungültig machen
  - externer Lesezugriff $\Rightarrow$ Cache liefert Daten

Cache Kohärenz (cont.)

- MESI Protokoll
  - besitzt zwei Statusbits für die Protokollzustände
  - **M**odified: Inhalte der Cache-Line wurden modifiziert
  - **E**xclusive unmodified: Cache-Line „gehört“ dieser CPU, nicht modifiz.
  - **S**hared unmodified: Inhalte sind in mehreren Caches vorhanden
  - **I**nvalid: ungültiger Inhalt, Initialzustand

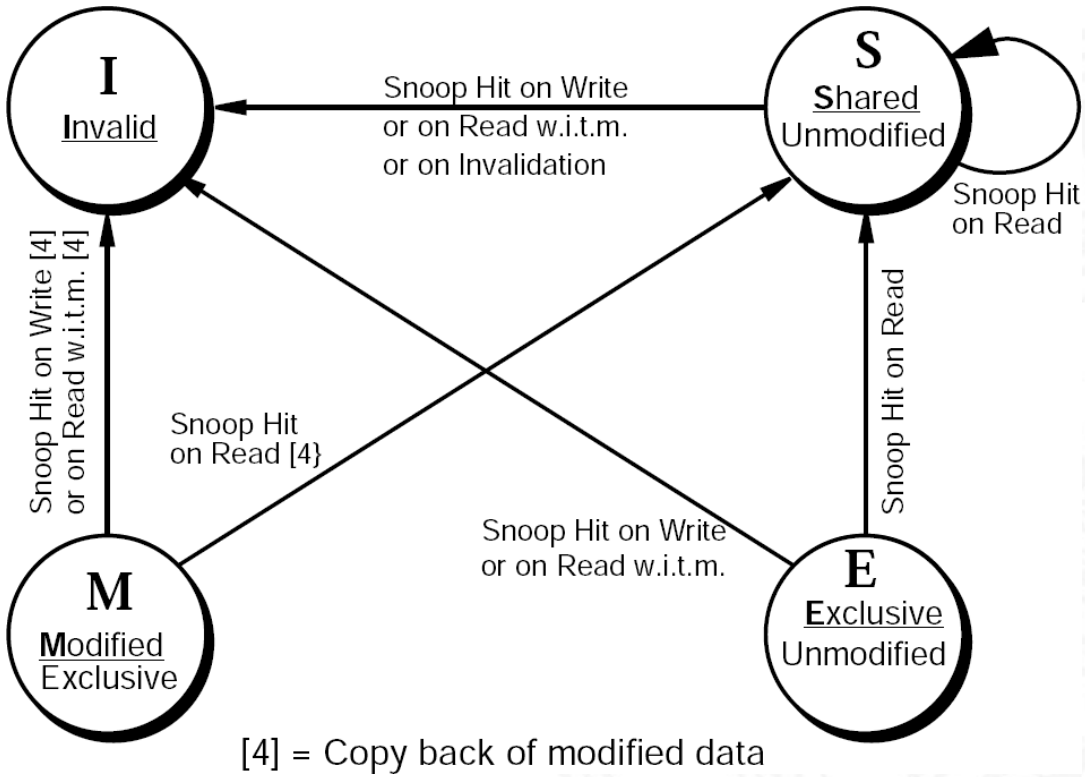
Cache Kohärenz (cont.)

- Zustandsübergänge für „Bus Master“ CPU



Cache Kohärenz (cont.)

- Zustandsübergänge für “Snooping“ CPU



Optimierung der Cachezugriffe

- Mittlere Speicherzugriffszeit $= T_{Hit} + R_{Miss} \cdot T_{Miss}$

$\Rightarrow$ Verbesserung der Cache Performanz durch kleinere T_{Miss} am einfachsten zu realisieren

- mehrere Cache Ebenen
- Critical Word First: bei großen Cache Blöcken (mehrere Worte) gefordertes Wort zuerst holen und gleich weiterleiten
- Read-Miss hat Priorität gegenüber Write-Miss
  $\Rightarrow$ Zwischenspeicher für Schreiboperationen (Write Buffer)
- Merging Write Buffer: aufeinanderfolgende Schreiboperationen zwischenspeichern und zusammenfassen
- Victim Cache: kleiner voll-assoziativer Cache zwischen direct-mapped Cache und nächster Ebene
  „sammelt“ verdrängte Cache Einträge

Optimierung der Cachezugriffe (cont.)

⇒ Verbesserung der Cache Performanz durch kleinere R_{Miss}
- größere Caches (– mehr Hardware)
- höhere Assoziativität (– langsamer)

⇒ weitere Optimierungstechniken
- Software Optimierungen
- Prefetch: Hardware (Stream Buffer)
  Software (Prefetch Operationen)
- Cache Zugriffe in Pipeline verarbeiten
- Trace Cache: im Instruktions-Cache werden keine Speicherinhalte, sondern ausgeführte Sequenzen (*trace*) einschließlich ausgeführter Sprünge gespeichert

  Beispiel: Pentium IV

Virtueller Speicher und Memory Management

Speicher-Paradigmen

- Programmierer
  - ein großer Adressraum
  - linear adressierbar
- Betriebssystem
  - eine Menge laufender Tasks / Prozesse
  - read-only Instruktionen
  - read-write Daten

Virtueller Speicher und Memory Management (cont.)

Aufgaben des Betriebssystems

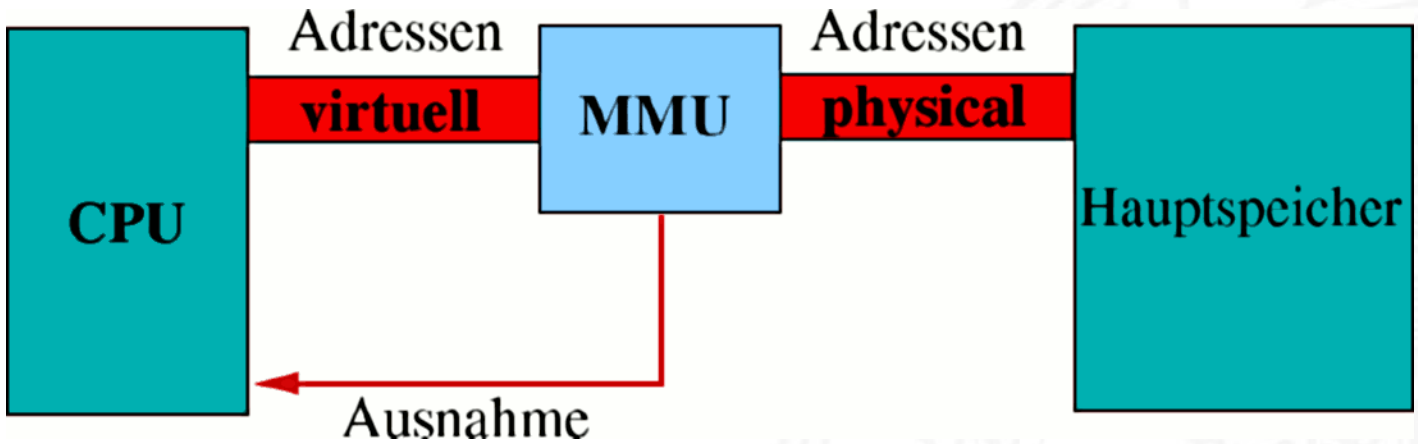
- effiziente Nutzung des Speichers gewährleisten
  - Prozess benötigt oft nicht alle Teile seines Adressraums
  - Speicherbedarf aller Prozesse kann größer als der verfügbare Hauptspeicher sein

  $\Rightarrow$ nicht benötigte Teile des Adressraums auf Hintergrundspeicher auslagern
- Speicherschutz
  - Prozesse haben nur Zugriff auf eigenen Daten

Virtueller Speicher und Memory Management (cont.)

Prinzip des virtuellen Speichers

- jeder Prozess besitzt seinen eigenen virtuellen Adressraum
- MMU – **M**emory **M**anagement **U**nit



- Umsetzung von virtuellen zu physikalischen Adressen, Programm-Relokation

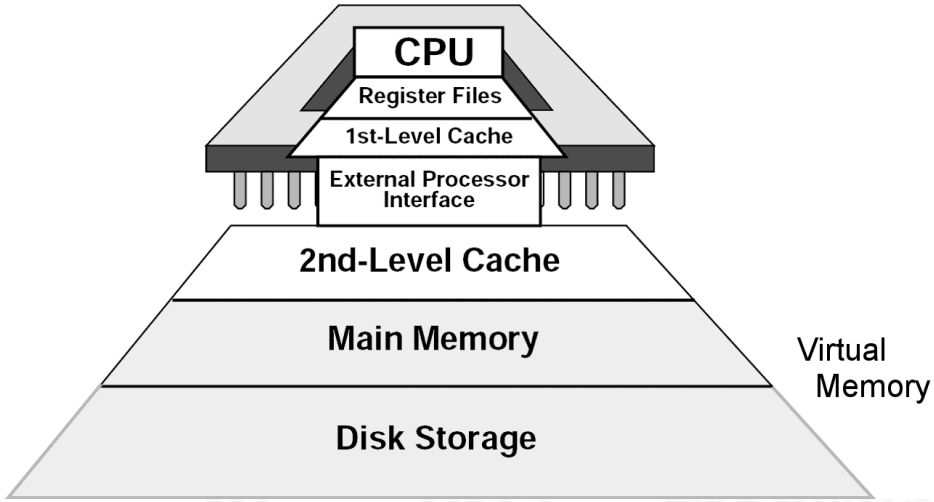
Virtueller Speicher und Memory Management (cont.)

- Umsetzungstabellen werden vom Betriebssystem erzeugt und aktualisiert
- wegen des Speicherbedarfs der Tabellen beziehen sich diese auf größere Speicherblöcke

- Umgesetzt wird nur die Anfangsadresse, der Offset innerhalb des Blocks bleibt unverändert
- Blöcke dieses virtuellen Adressraums können durch Betriebssystem ausgelagert werden
- Methoden zur Implementation virtuellen Speichers
  - *Segmentierung*
  - *Paging*, Speicherzuordnung durch *Seiten*
  - gemischte Verfahren (Standard bei: Desktops, Workstations...)

Virtueller Speicher und Memory Management (cont.)

- Hauptspeicher als Cache für den Plattenspeicher

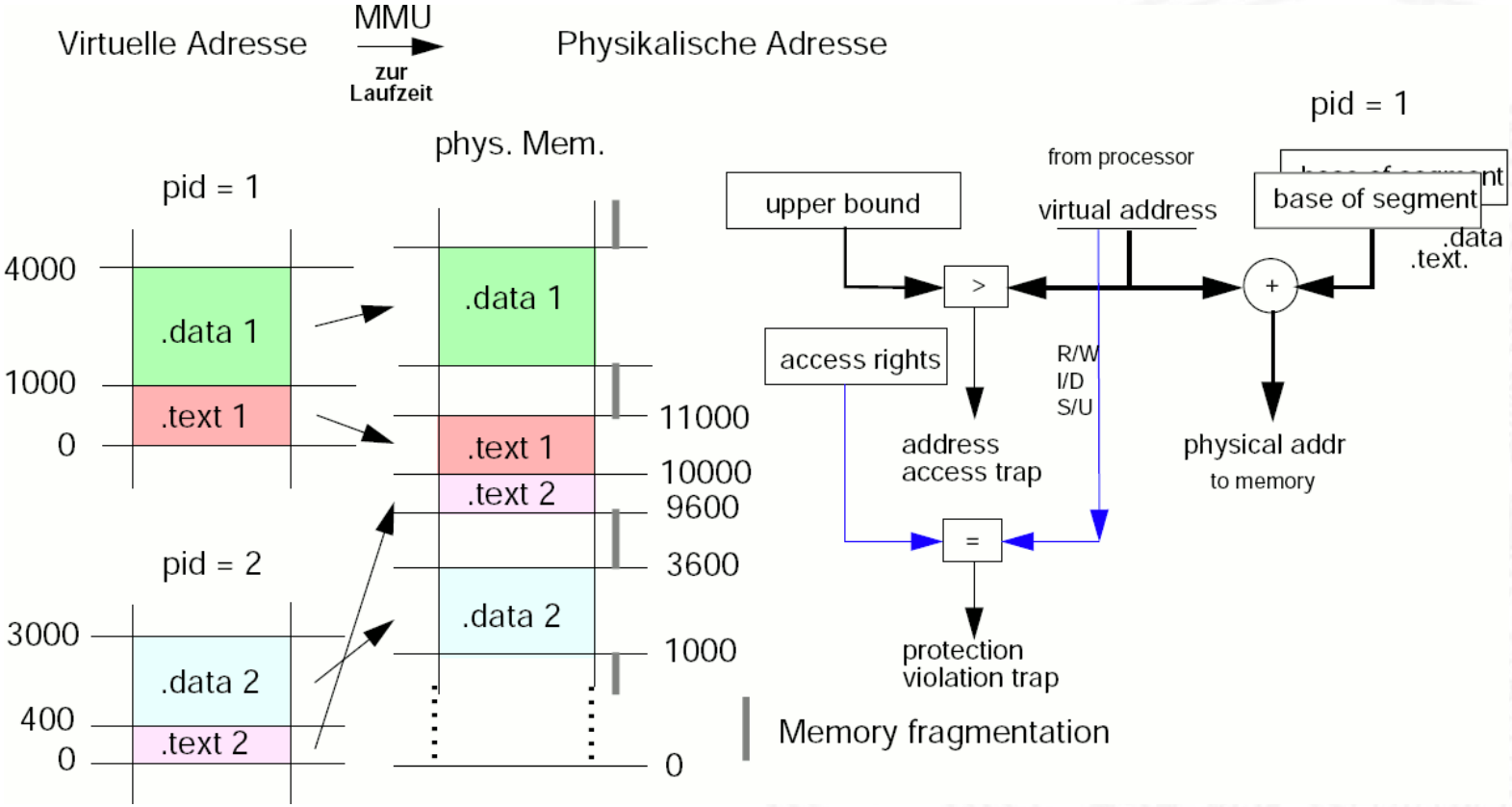


- Parameter der Speicherhierarchie

	1st-Level Cache	virtueller Speicher
Blockgröße	16-128 Byte	4-64 kByte
Hit-Dauer	1-2 Zyklen	40-100 Zyklen
Miss Penalty	8-100 Zyklen	70.000-6.000.000 Zyklen
Miss Rate	0,5-10 %	0,00001-0,001 %
Speichergröße	8-64 kByte	16-8192 MByte

Virtueller Speicher: Segmentierung

- Unterteilung des Adressraums in kontinuierliche Bereiche *variabler* Größe



Virtueller Speicher: Segmentierung (cont.)

- Idee: Trennung von Instruktionen, Daten und Stack

$\Rightarrow$ Abbildung von *Programmen* in den *Hauptspeicher*

$+$ Inhalt der Segmente: logisch zusammengehörige Daten

$+$ getrennte Zugriffsrechte, Speicherschutz

$+$ exakte Prüfung der Segmentgrenzen

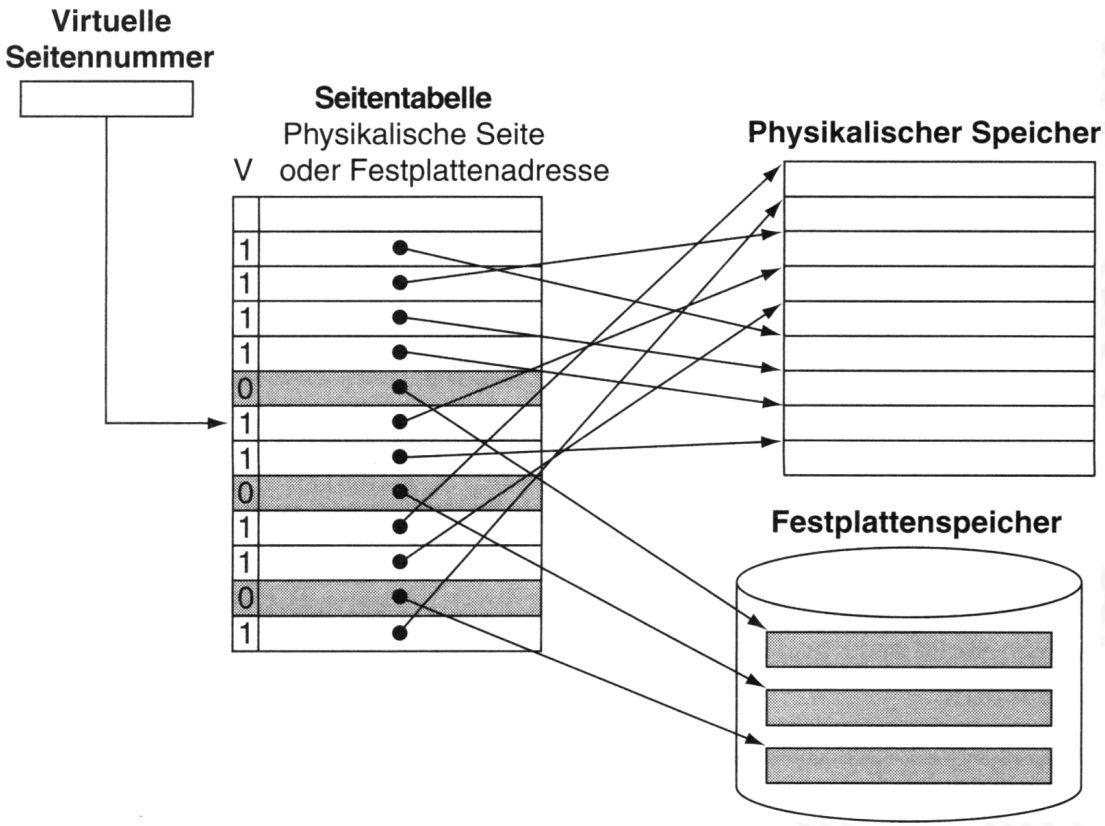
$-$ Segmente könne sehr groß werden

$-$ Ein- und Auslagern von Segmenten kann sehr lange dauern

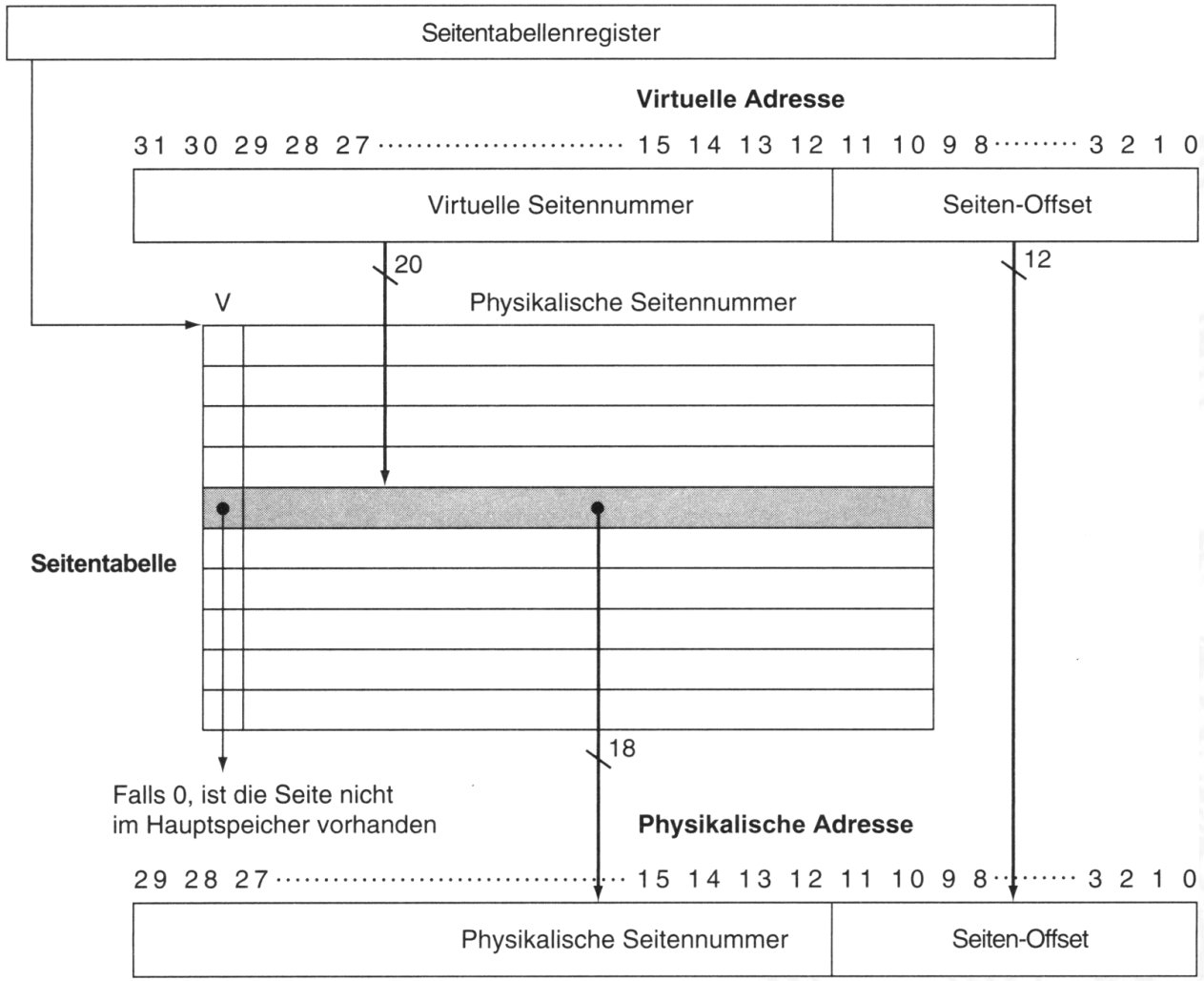
$-$ „Verschnitt“, **Memory Fragmentation**

Virtueller Speicher: Paging / Seitenadressierung

- Unterteilung des Adressraums in Blöcke *fester* Größe = Seiten
  Abbildung auf Hauptspeicherblöcke = Kacheln



Virtueller Speicher: Paging / Seitenadressierung (cont.)



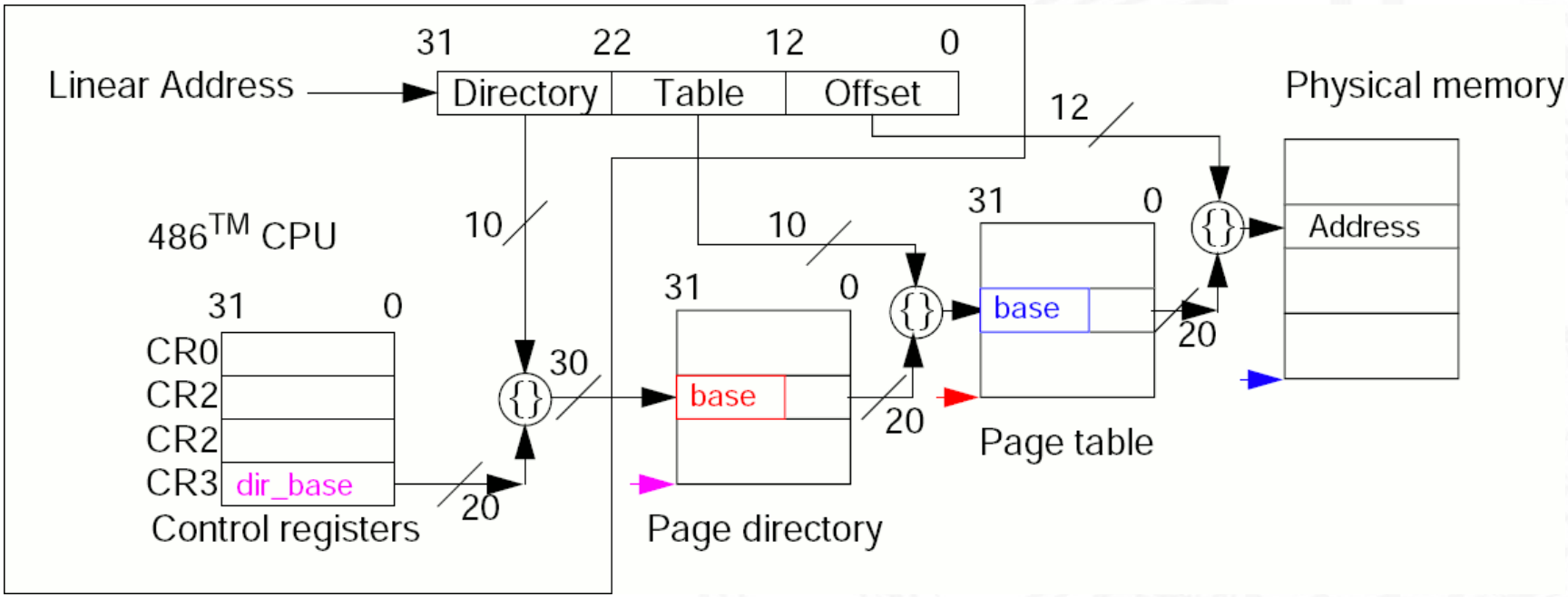
Virtueller Speicher: Paging / Seitenadressierung (cont.)

- ⇒ Abbildung von *Adressen* in den *virtuellen Speicher*
- \+ Programme können größer als der Hauptspeicher sein
- \+ Programme können an beliebige physikalischen Adressen geladen werden, unabhängig von der Aufteilung des physikalischen Speichers
- \+ feste Seitengröße: einfache Verwaltung in Hardware
- \+ Zugriffsrechte für jede Seite (read/write, User/Supervisor)
- ▶ große Miss-Penalty (Nachladen von der Platte)
  ⇒ Seiten sollten relativ groß sein: 4 oder 8 kByte
- – Speicherplatzbedarf der Seitentabelle
  viel virtueller Speicher, 4 kByte Seitengröße
  ⇒ große Pagetable

Virtueller Speicher: Paging / Seitenadressierung (cont.)

⇒ Hash-Verfahren (*inverted page tables*)

⇒ mehrstufige Verfahren

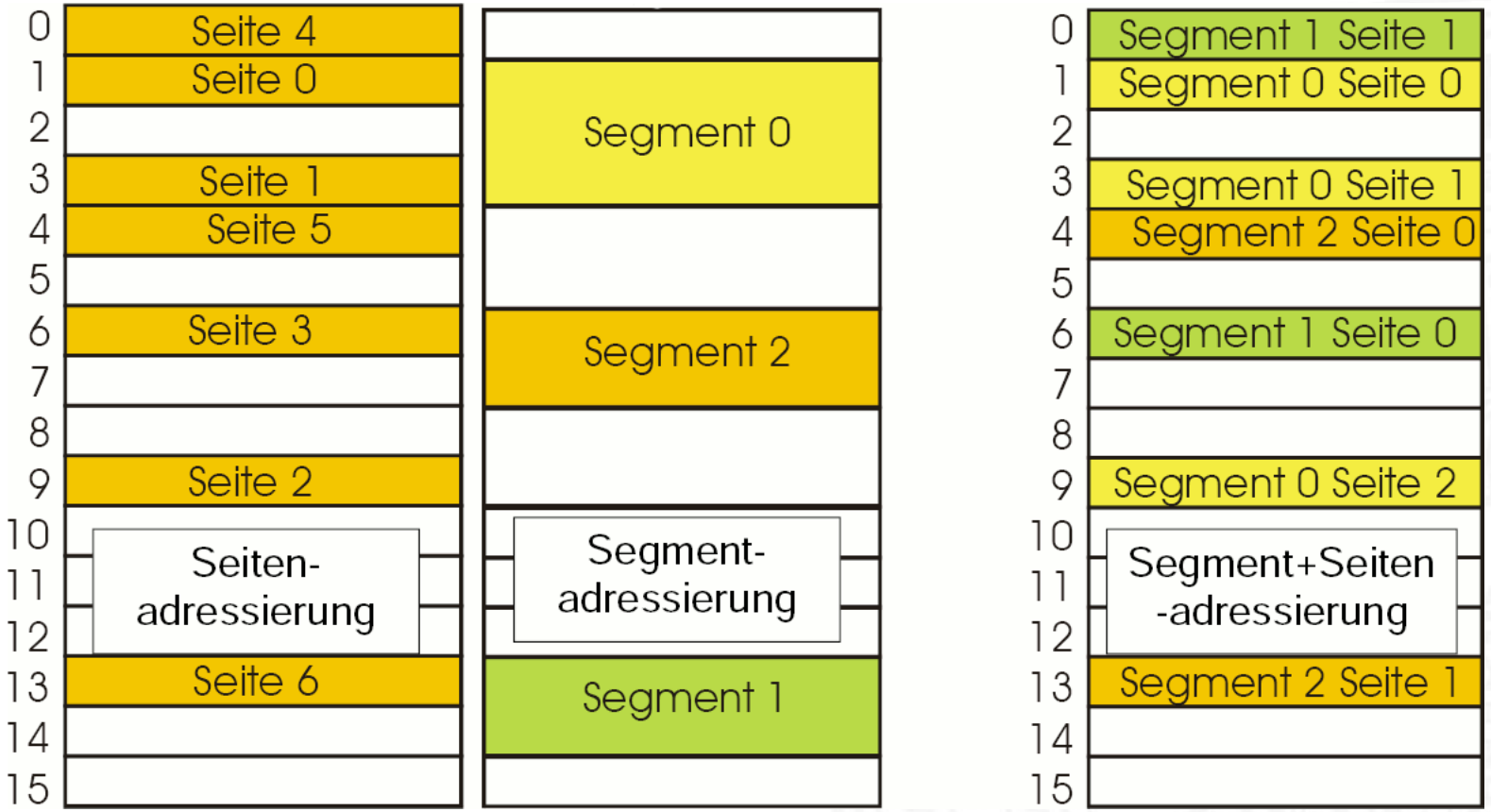


Virtueller Speicher: Paging / Seitenadressierung (cont.)

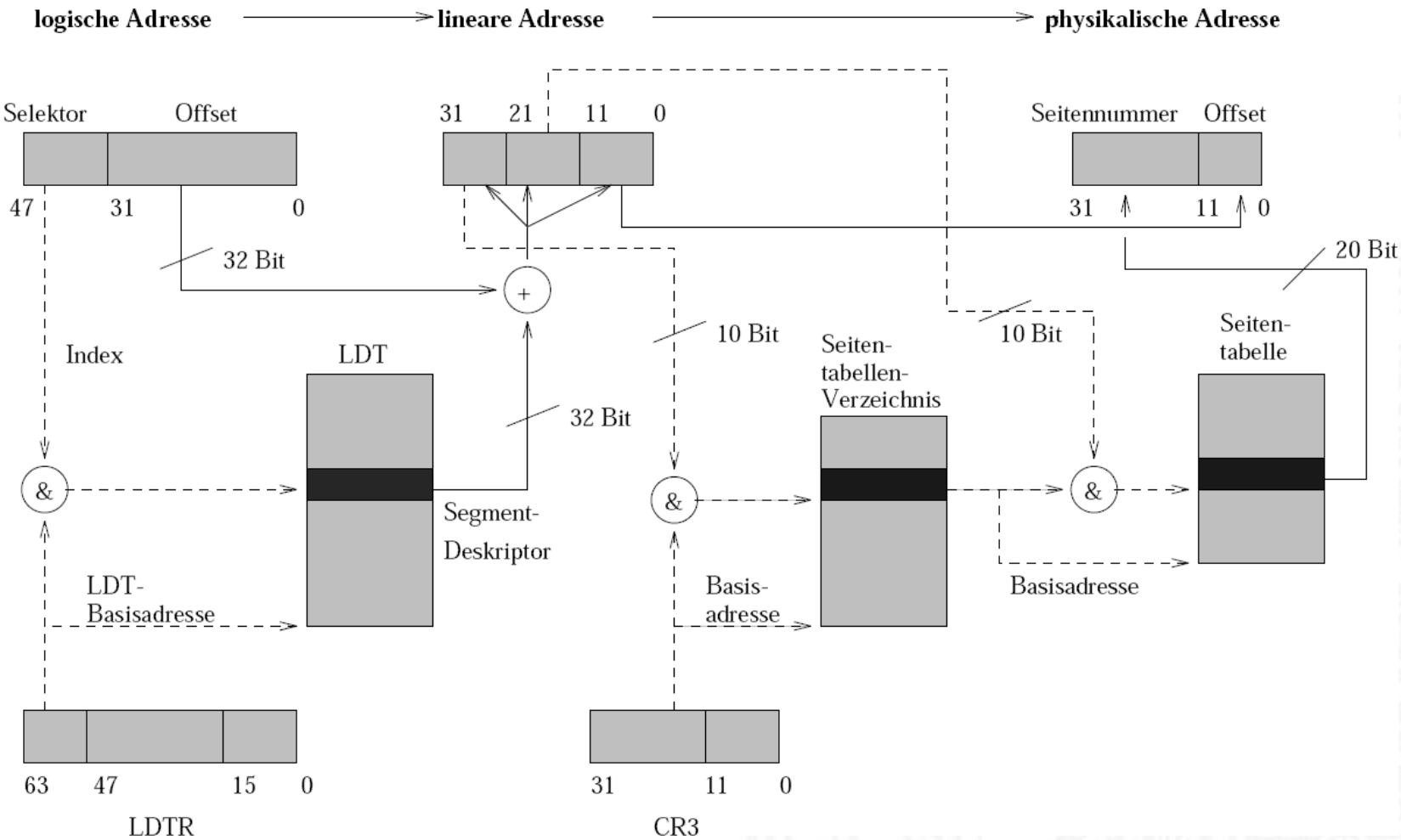
- Schreibstrategie
  - Write-Back (Write-Through wegen Plattenzugriffs zu langsam)
- Ersetzungsstrategie
  - Not recently used
  - Least recently used
  - FIFO...
- Behandlung von Seitenfehlern („*Page-Faults*“) in Software, durch Betriebssystem
  1. Umschalten auf anderen laufbereiten Prozess
  2. Daten in freie Page im Hauptspeicher laden (DMA)
  3. Interrupt wenn DMA fertig ist
  4. Interrupt-Handler: Page-Tabelle des Prozesses updaten
  - beim Zurückschalten auf den ursprünglichen Prozess sind die Daten dann vorhanden

Virtueller Speicher: Segmentierung + Paging

Gemischte Verfahren: Segmentierung und Paging (Beispiel: I386)



Virtueller Speicher: Segmentierung + Paging (cont.)



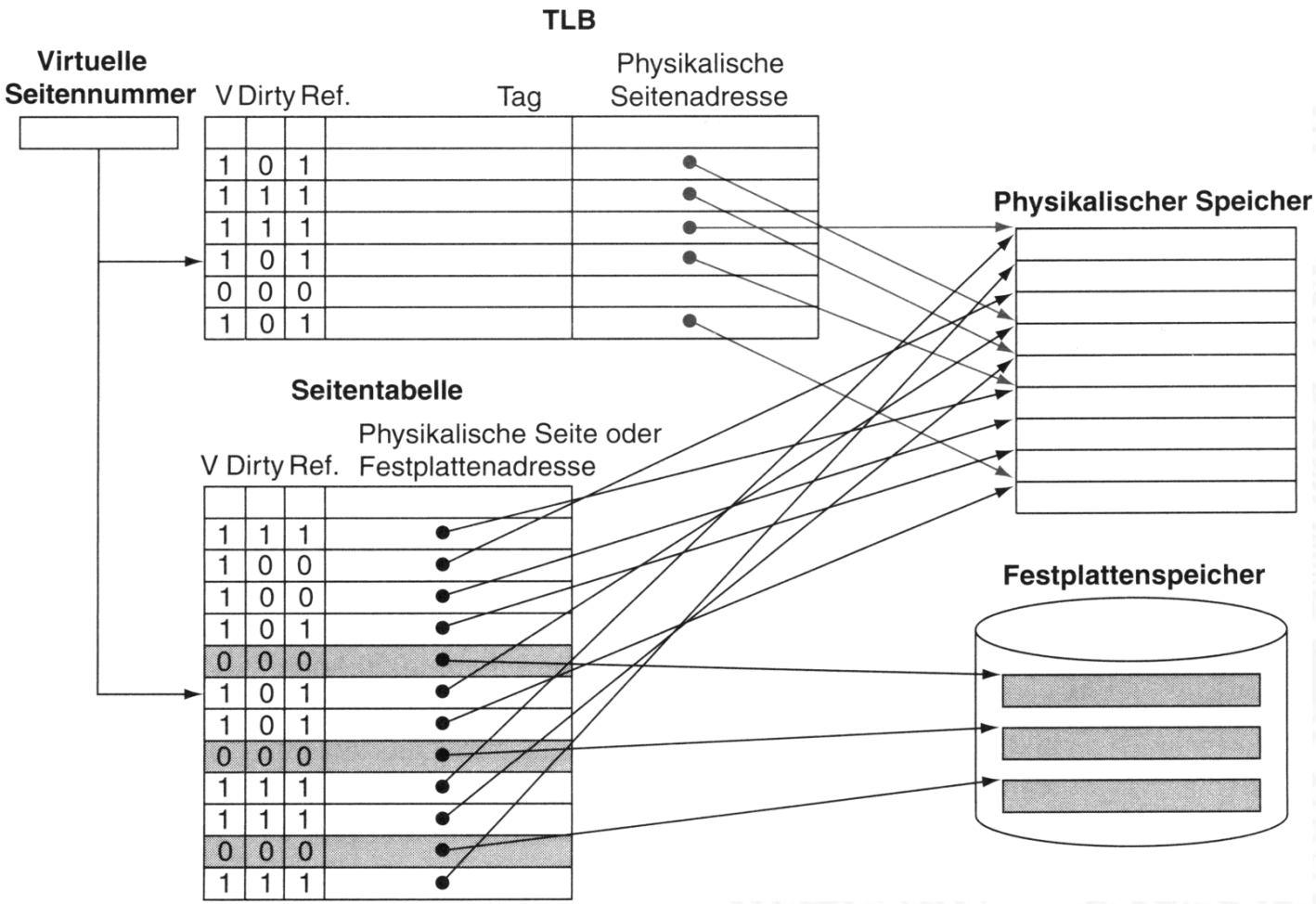
Virtueller Speicher: Segmentierung + Paging (cont.)

- Problem der Adressübersetzung
  - mehrstufige Verfahren
    Beispiel: Segmentierung + 2-stufige Seitenadressierung
  - die jeweiligen Tabellen sind selber im Speicher abgelegt
  - – mehrere Speicherzugriffe

⇒ TLB – **T**ranslation **L**ookaside **B**uffer

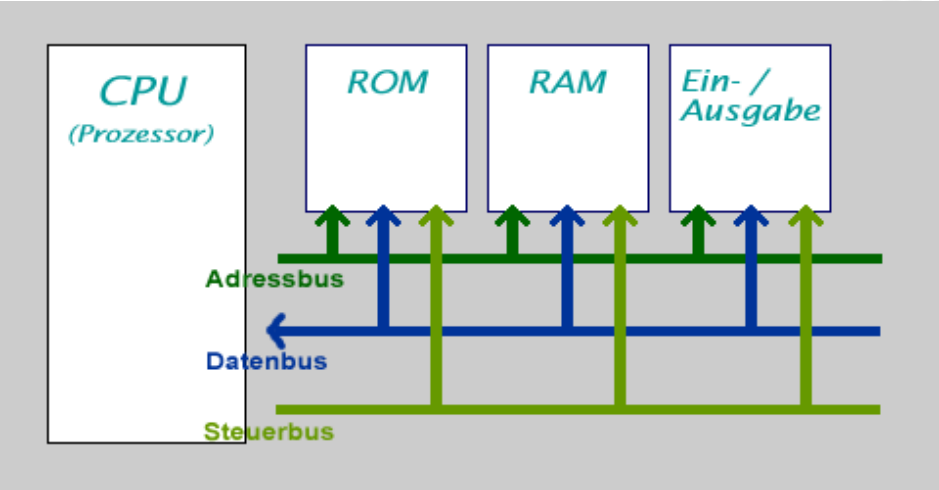
- Cache für Seitentabelle

Virtueller Speicher: Segmentierung + Paging (cont.)



Bussysteme

- Aufgaben
  - Kernkomponenten (CPU, Speicher...) miteinander verbinden
  - Verbindungen zu den Peripherie-Bausteinen
  - Verbindungen zu Systemmonitor-Komponenten
  - Verbindungen zwischen I/O-Controllern und -Geräten
  - ...



Bussysteme (cont.)

- viele unterschiedliche Typen, standardisiert mit sehr unterschiedlichen Anforderungen
  - High-Performance
  - einfaches Protokoll, billige Komponenten
  - Multi-Master-Fähigkeit, zentrale oder dezentrale Arbitrierung
  - Echtzeitfähigkeit, Daten-Streaming
  - wenig Leitungen bis zu Zweidraht-Bussen: I^2C, System-Management-Bus...
  - lange Leitungen: RS232, Ethernet...
  - Funkmedium: WLAN, Bluetooth

Bus-Arbitrierung

- mehrere Komponenten wollen Übertragung initiieren
- der Zugriff muss serialisiert werden

$\Rightarrow$ zentrale Arbitrierung

- Arbiter gewährt Bus-Requests
- Strategien: Priorisierung, Round-Robin...

$\Rightarrow$ dezentrale Arbitrierung

- protokollbasiert
- Beispiel
  - Komponenten sehen ob Bus frei ist
  - beginnen zu senden
  - Kollisionserkennung: gesendete Daten lesen
  - ggf. Übertragung abbrechen
  - „später“ erneut versuchen

Bus-Bandbreite

- Menge an (Nutz-) Daten, die pro Zeiteinheit übertragen werden kann
- zusätzlicher Protokolloverhead $\Rightarrow$ Brutto- / Netto-Datenrate
- RS232 50 Bit/sec ... 460 KBit/sec
  I^2C 100 KBit/sec (Std.)... 3,4 MBit/sec (High Speed)
  USB 1,5 MBit/sec (1.x) ... 5 GBit/sec (3.0)
  ISA 128 MBit/sec
  PCI 1 GBit/sec (2.0) ... 4,3 GBit/sec (3.0)
  AGP 2,1 GBit/sec (1x) ... 16,6 GBit/sec (8x)
  PCIe 250 MByte/sec (1.x) ...1000 MByte/sec (3.0) x1...32
  HyperTransport 12,8 GByte/sec (1.0) ... 51,2 GByte/sec (3.1)

Bus-Bandbreite (cont.)

- Beispiel: Grafikanbindung

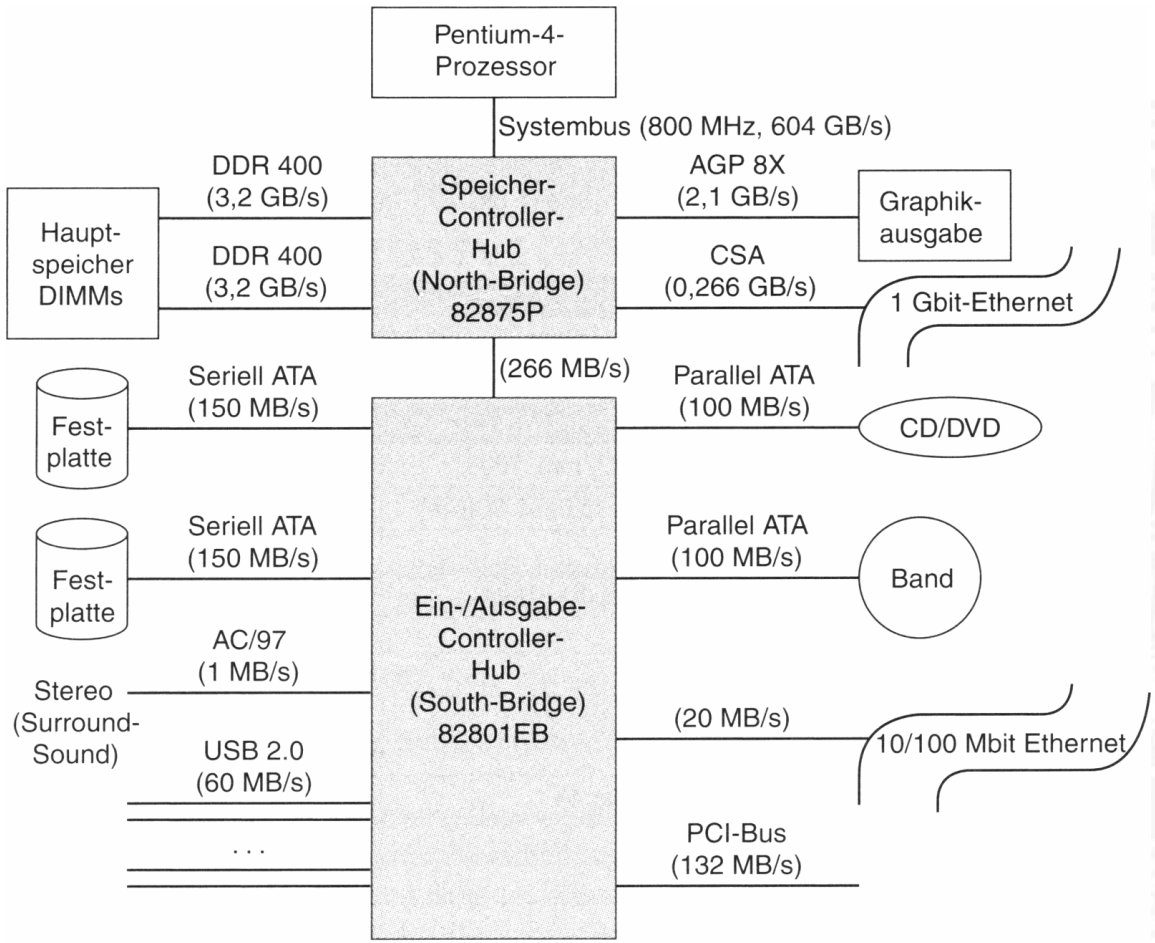
Quelle: c't 01/2009

PCI, PCI-X, PCI Express und AGP

Typ	Signalleitungen/Lanes	Taktfrequenz	Signalspannung	Datentransferrate
PCI: 32 Bit/33 MHz	32 (4 Byte)	33,33 MHz SDR	5 (3,3) Volt	133 MByte/s
PCI: 32 Bit/66 MHz	32 (4 Byte)	66,66 MHz SDR	3,3 Volt	266 MByte/s
PCI: 64 Bit/33 MHz	64 (8 Byte)	33,33 MHz SDR	5 (3,3) Volt	266 MByte/s
PCI: 64 Bit/66 MHz	64 (8 Byte)	66,66 MHz SDR	3,3 Volt	532 MByte/s
PCI-X 66	64 (8 Byte)	66,66 MHz SDR	3,3 Volt	532 MByte/s
PCI-X 100	64 (8 Byte)	100 MHz SDR	3,3 Volt	800 MByte/s
PCI-X 133	64 (8 Byte)	133 MHz SDR	3,3 Volt	1,06 GByte/s
PCI-X 2.0 266	64 (8 Byte)	133 MHz DDR	1,5 Volt	2,1 GByte/s
PCI-X 2.0 533	64 (8 Byte)	133 MHz QDR	1,5 Volt	4,2 GByte/s
AGP 2X	32 (4 Byte)	66,66 MHz DDR	3,3 Volt	532 MByte/s
AGP 4X	32 (4 Byte)	66,66 MHz QDR	1,5 Volt	1,06 GByte/s
AGP 8X (AGP 3.0)	32 (4 Byte)	66,66 MHz ODR	0,8 Volt	2,1 GByte/s
PCI Express 1.1 x1	1 (je Richtung)	1,25 GHz DDR 8b/10b	0,8 $Volt_{P-P}$	250 + 250 MByte/s
PCI Express 1.1 x4	4 (je Richtung)	1,25 GHz DDR 8b/10b	0,8 $Volt_{P-P}$	1 + 1 GByte/s
PCI Express 1.1 x8	8 (je Richtung)	1,25 GHz DDR 8b/10b	0,8 $Volt_{P-P}$	2 + 2 GByte/s
PCI Express 1.1 x16	16 (je Richtung)	1,25 GHz DDR 8b/10b	0,8 $Volt_{P-P}$	4 + 4 GByte/s
PCI Express 2.0 x1	1 (je Richtung)	2,5 GHz DDR 8b/10b	0,8 $Volt_{P-P}$	0,5 + 0,5 GByte/s
PCI Express 2.0 x4	4 (je Richtung)	2,5 GHz DDR 8b/10b	0,8 $Volt_{P-P}$	2 + 2 GByte/s
PCI Express 2.0 x8	8 (je Richtung)	2,5 GHz DDR 8b/10b	0,8 $Volt_{P-P}$	4 + 4 GByte/s
PCI Express 2.0 x16	16 (je Richtung)	2,5 GHz DDR 8b/10b	0,8 $Volt_{P-P}$	8 + 8 GByte/s

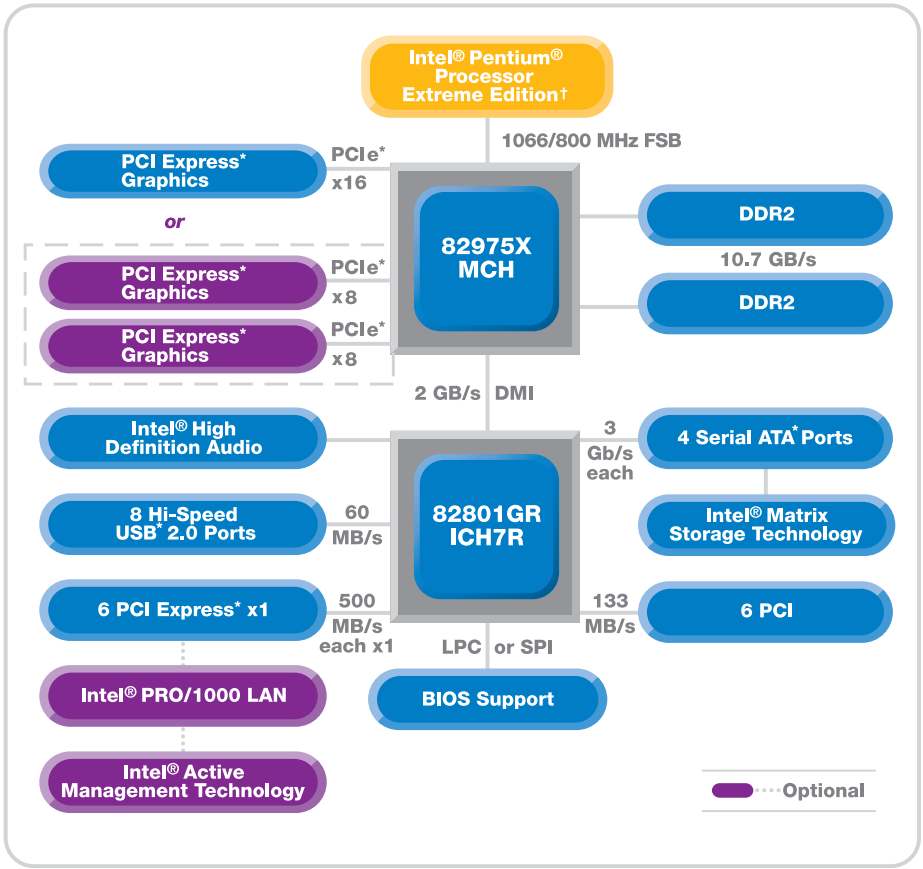
PC Bussysteme

Beispiel:
Intel Chipsätze



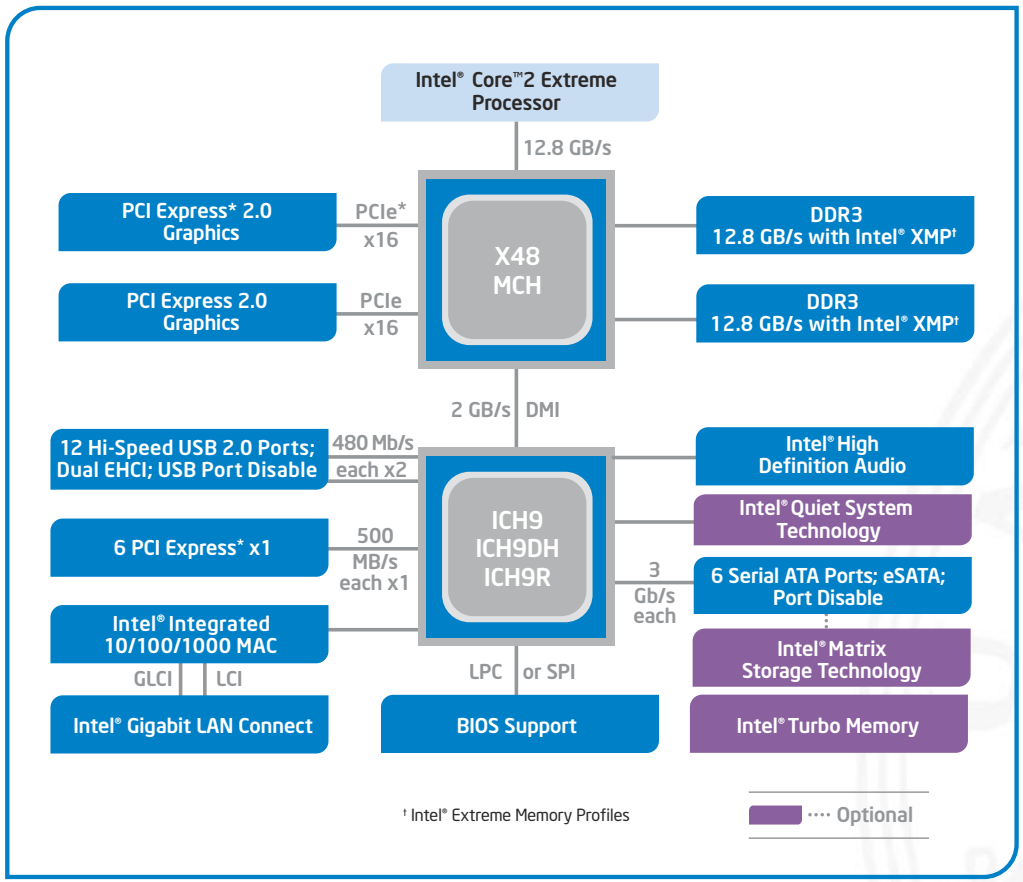
i875

PC Bussysteme (cont.)

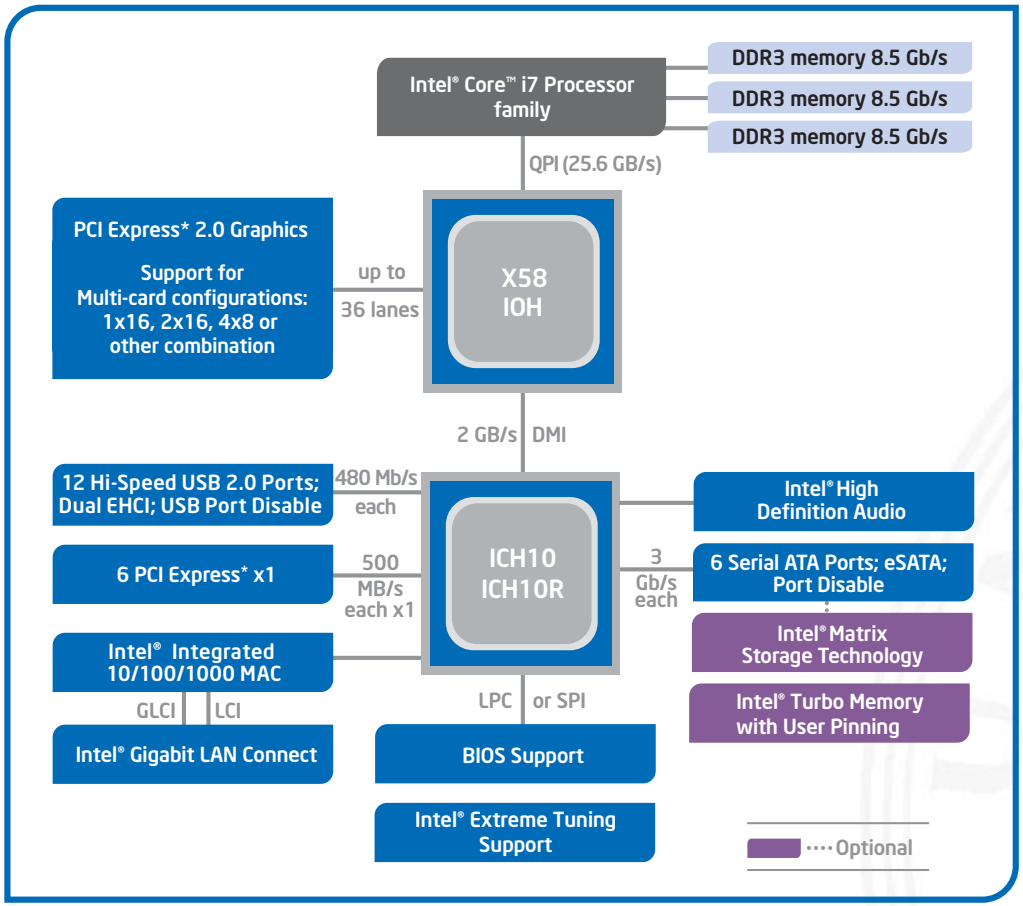


Intel 975X Express Chipset Block Diagram

PC Bussysteme (cont.)



PC Bussysteme (cont.)



Intel® X58 Express Chipset Block Diagram

Literaturliste

[Com05] Douglas E. Comer:
Essentials of Computer Architecture.
Pearson Education; Upper Saddle River, NJ, 2005.
ISBN 0–13–149179–2

[DC96] Mario Dal Cin:
Rechnerarchitektur.
Teubner; Stuttgart, 1996.
ISBN 3–519–02941–3

Literaturliste (cont.)

[HP94] John L. Hennessy, David A. Patterson:
Rechnerarchitektur: Analyse, Entwurf, Implementierung, Bewertung.
Vieweg; Braunschweig, 1994.
ISBN 3–52805–173–6

[HP07] John L. Hennessy, David A. Patterson:
Computer architecture: a quantitative approach.
Morgan Kaufmann Publishers, Inc.;
San Mateo, CA, 2007.
ISBN 978–0–12–370490–0

Literaturliste (cont.)

[HRU02] John E. Hopcroft, Motvani Rajeev, Jeffrey D. Ullman: *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*.
Pearson Studium; München, Boston..., 2002.
ISBN 3–8273–7020–5

[ITRS07] *International Technology Roadmap for Semiconductors – 2007 Edition*.
Semiconductor Industry Association, 2007.
URL `www.itrs.net/Links/2007ITRS/Home2007.htm`

Literaturliste (cont.)

[ITRS09] *International Technology Roadmap for Semiconductors – 2009 Edition*.
Semiconductor Industry Association, 2009.
URL `www.itrs.net/Links/2009ITRS/Home2009.htm`

[Lag87] Klaus Lagemann:
Rechnerstrukturen.
Springer-Verlag; Berlin, 1987.
ISBN 3–540–17618–7

Literaturliste (cont.)

[Mär03] Christian Märtin:
Einführung in die Rechnerarchitektur: Prozessoren und Systeme.
Fachbuchverlag Leipzig im Carl Hanser Verlag;
Leipzig, 2003.
ISBN 3–446–22242–1

[MC80] Carver Mead, Lynn Conway:
Introduction to VLSI systems.
Addison-Wesley; Reading, MA, 1980.
ISBN 0–201–04358–0

Literaturliste (cont.)

[OV06] Walter Oberschelp, Gottfried Vossen:
Rechneraufbau und Rechnerstrukturen.
Oldenbourg; München, 2006.
ISBN 3–486–57849–9

[PH05] David A. Patterson, John L. Hennessy:
Rechnerorganisation und -entwurf: die Hardware/Software-Schnittstelle.
Elsevier / Spektrum Akademischer Verlag; Heidelberg, 2005.
ISBN 3–8274–1595–0

Literaturliste (cont.)

[PH09] David A. Patterson, John L. Hennessy:
Computer Organization and Design: The Hardware/Software Interface.
Morgan Kaufmann Publishers, Inc.; San Mateo, CA, 2009.
ISBN 978–0–12–374493–7

[Rei98] Norbert Reifschneider:
CAE-gestützte IC-Entwurfsmethoden.
Prentice Hall; München, 1998.
ISBN 3–8272–9550–5

Literaturliste (cont.)

[Tan06] Andrew S. Tanenbaum:
Computerarchitektur: Strukturen, Konzepte, Grundlagen.
Pearson Studium; München, Boston. . . , 2006.
ISBN 3–8273–7151–1

[TS09] Ulrich Tietze, Christoph Schenk:
Halbleiter-Schaltungstechnik.
Springer-Verlag; Berlin, 2009.
ISBN 978–3–642–01621–9

Literaturliste (cont.)

[WE94] Neil H. E. Weste, Kamran Eshraghian:
Principles of CMOS VLSI design: a systems perspective.
Addison-Wesley; Reading, MA, 1994.
ISBN 0–201–53376–6

IEEE / IEC Standards

[IEC 61691-1-1] *IEC 61691-1-1 First edition 2004-10; IEEE 1076 – Behavioural languages - Part 1-1: VHDL language reference manual.*
International Electrotechnical Commission; Genf, 2004.
ISBN 2–8318–7691–5

[IEEE 1076] *Standard 1076-2008, IEEE Standard VHDL Language Reference Manual.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 2009.
ISBN 978–0–7381–5801–3

IEEE / IEC Standards (cont.)

[IEEE 1076.1] *Standard 1076.1-2007, IEEE Standard VHDL Analog and Mixed-Signal Extensions.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 2007.
ISBN 0–7381–5627–2

[IEC 61691-6] *IEC 61691-6; IEEE 1076.1-2009 – Behavioural languages - Part 6: VHDL Analog and Mixed-Signal Extensions.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 2009.
ISBN 978–0–7381–6283–6

IEEE / IEC Standards (cont.)

[IEEE 1076.2] *Standard 1076.2-1996, IEEE Standard VHDL Mathematical Packages*.
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 1996.
ISBN 0–7381–0988–6

[IEC 61691-3-2] *IEC 61691-3-2 First edition 2001-06 – Behavioural languages - Part 3-2: Mathematical operation in VHDL*.
International Electrotechnical Commission; Genf, 2001.
ISBN 0–580–39086–1

IEEE / IEC Standards (cont.)

[IEEE 1076.3] *Standard 1076.3-1997, IEEE Standard VHDL Synthesis Packages.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 1997.
ISBN 1–5593–7923–5

[IEC 61691-3-3] *IEC 61691-3-3 First edition 2001-06 – Behavioural languages - Part 3-3: Synthesis in VHDL.*
International Electrotechnical Commission; Genf, 2001.
ISBN 0–580–39087–X

IEEE / IEC Standards (cont.)

[IEEE 1076.4] *Standard 1076.4-2000, IEEE Standard VITAL ASIC (Application Specific Integrated Circuit) Modeling Specification 2001.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 2001.
ISBN 0–7381–2691–0

[IEC 61691-5] *IEC 61691-5 First edition 2004-10; IEEE 1076.4 – Behavioural languages - Part 5: VITAL ASIC (application specific integrated circuit) modeling specification.*
International Electrotechnical Commission; Genf, 2004.
ISBN 2–8318–7684–2

IEEE / IEC Standards (cont.)

[IEEE 1076.6] *Standard 1076.6-1999, IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 1999.
ISBN 0–7381–1819–2

[IEC 62050] *IEC 62050 First edition 2005-07; IEEE 1076.6 – IEEE Standard for VHDL Register Transfer Level (RTL) synthesis.*
International Electrotechnical Commission; Genf, 2004.
ISBN 0–7381–4065–1

IEEE / IEC Standards (cont.)

[IEEE 1164] *Standard 1164-1993, IEEE Standard Multivalue Logic System for VHDL Model Interoperability.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 1993.
ISBN 1–55937–299–0 (withdrawn)

[IEC 61691-2] *IEC 61691-2 First edition 2001-06 – Behavioural languages - Part 2: VHDL multilogic system for model interoperability.*
International Electrotechnical Commission; Genf, 2001.
ISBN 0–580–39266–X

IEEE / IEC Standards (cont.)

[IEC 61691-4] *IEC 61691-4 First edition 2004-10; IEEE 1364 – Behavioural languages - Part 4: Verilog hardware description language*.
International Electrotechnical Commission; Genf, 2004.
ISBN 2–8318–7675–3

[IEEE 1364] *Standard 1364-2005, IEEE Standard for Verilog Hardware Description Language*.
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 2006.
ISBN 0–7381–8450–4

IEEE / IEC Standards (cont.)

[IEEE 1364.1] *Standard 1364.1-2002, IEEE Standard for Verilog Register Transfer Level Synthesis.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 2002.
ISBN 0–7381–3501–1

[IEC 62142] *IEC 62142 First edition 2005-06; IEEE 1364.1 – Verilog register transfer level synthesis.*
International Electrotechnical Commission; Genf, 2005.
ISBN 2–8318–8036–X

IEEE / IEC Standards (cont.)

[IEEE 1666] *Standard 1666-2005, IEEE Standard SystemC Language Reference Manual.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 2005.
ISBN 0–7381–4871–7

[IEC 61691-7] *IEC 61691-7 Edition 1.0 2009-12; IEEE 1666 – Behavioural languages - Part 7: SystemC Language Reference Manual.*
International Electrotechnical Commission; Genf, 2009.
ISBN 978–0–7381–6284–3

IEEE / IEC Standards (cont.)

[IEC 62530] *IEC 62530 Edition 1.0 2007-11; IEEE 1800 – Standard for SystemVerilog - Unified Hardware Design, Specification and Verification Language.*
International Electrotechnical Commission; Genf, 2007.
ISBN 2–8318–9349–6

[IEEE 1800] *IEEE 1800-2009 – Standard for SystemVerilog - Unified Hardware Design, Specification and Verification Language.*
Institute of Electrical and Electronics Engineers, Inc.; New York, NY, 2009.
ISBN 978–0–7381–6129–7