



# Introduction to ROS

Shang-Ching Liu, Niklas Fiedler, Lasse Einig, Dennis Krupke,  
Florens Wasserfall



University of Hamburg  
Faculty of Mathematics, Informatics and Natural Sciences  
Department of Informatics  
**Technical Aspects of Multimodal Systems**

October 16, 2024



# Motivation

- ▶ Heterogeneity vs. Homogeneity
  - ▶ sensor types, actuators, ...
  - ▶ sensor model, kinematic chain, ...
- ▶ Abstraction
- ▶ Algorithm re-usability
  - ▶ 2D laser data mapping
  - ▶ object recognition
- ▶ Debugging
  - ▶ simulation, data visualization, ...



# Idea

- ▶ Robot Operating System
- ▶ Meta operating system
- ▶ Open source
- ▶ Software encapsulation
- ▶ Hardware abstraction
  - ▶ portability
  - ▶ simplification of sensors and actuators
- ▶ Recurring tasks already solved
  - ▶ Navigation, data filtering, object recognition ...



# Current State

- ▶ Multiple versions actively used
  - ▶ may not be compatible to each other
  - ▶ may not provide same libraries
- ▶ Linux (Ubuntu!)
- ▶ Supports C/C++, Python (and others)
  - ▶ Python for high level code/fast implementation
  - ▶ C/C++ for algorithms/computation
- ▶ Many tools, functions and algorithms already available
  - ▶ May be difficult to find
  - ▶ Better than reimplementing



# ROS System

- ▶ ROS nodes
  - ▶ sensors
  - ▶ actuators
  - ▶ logic
- ▶ ROS core
- ▶ Communication
- ▶ Visualization
- ▶ Tools



# ROS Node

- ▶ Discrete part of the system
- ▶ Specialized software/algorithm
- ▶ Many ROS nodes per system
- ▶ Example:
  - ▶ node gets image
  - ▶ runs edge detection algorithm on it
  - ▶ provides found edges



# ROS Core

- ▶ Central unit, also called ROS master
  - ▶ nodes
  - ▶ sensors
  - ▶ communication
- ▶ Coordination of nodes
- ▶ Communication Management
- ▶ Exactly one per system
- ▶ Transparent to the user



# Communication

- ▶ Messages
  - ▶ standardized data types
- ▶ Topics
  - ▶ n:n communication
- ▶ Services and Actions
  - ▶ 1:1 communication





# Messages

- ▶ Fundamental communication concept
- ▶ Description of data set
- ▶ Data types
  - ▶ ROS
  - ▶ general
- ▶ Header
  - ▶ time stamp
  - ▶ identifier

```
$ rosmmsg show -r robot_msgs/Quaternion
# xyz - vector rotation axis, w - scalar term (cos(ang/2))
float64 x
float64 y
float64 z
float64 w
```



# Messages

- ▶ Fundamental communication concept
- ▶ Description of data set
- ▶ Data types
  - ▶ ROS
  - ▶ general
- ▶ Header
  - ▶ time stamp
  - ▶ identifier

```
$ rosmmsg show -r robot_msgs/Quaternion
# xyz - vector rotation axis, w - scalar term (cos(ang/2))
float64 x
float64 y
float64 z
float64 w
```

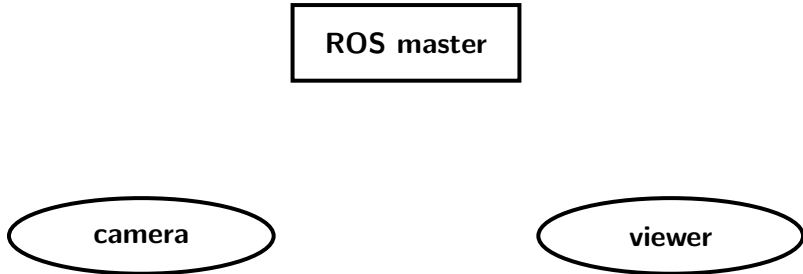


# Topics

- ▶ Published by nodes
- ▶ Unique identifier
- ▶ Anonymity
- ▶ Open subscription

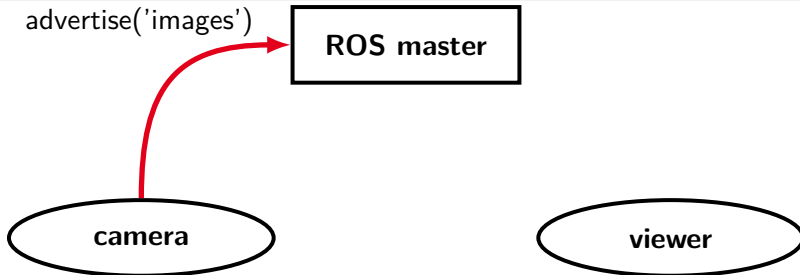


## Communication - Example



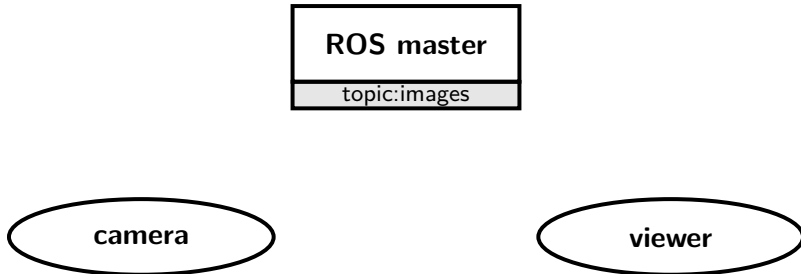


## Communication - Example



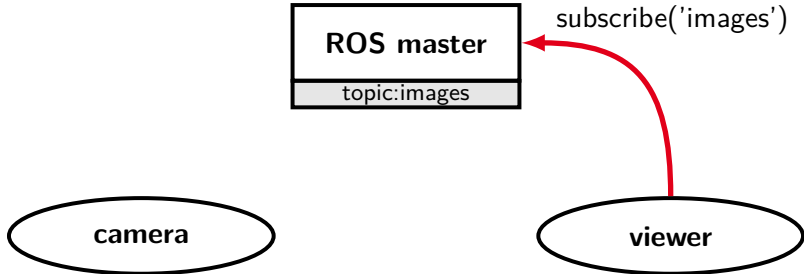


## Communication - Example



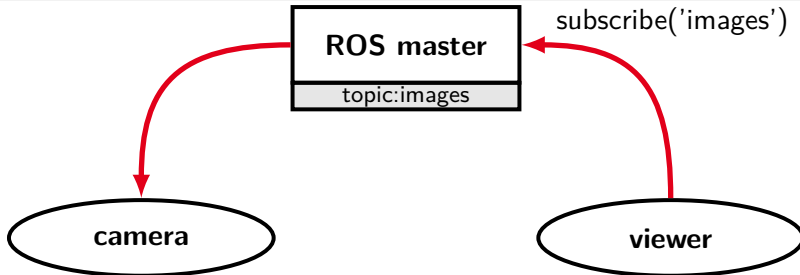


## Communication - Example





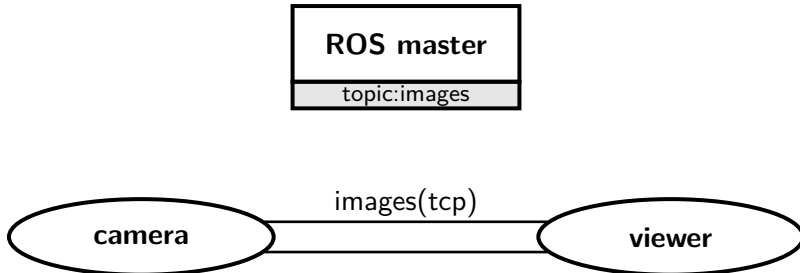
## Communication - Example



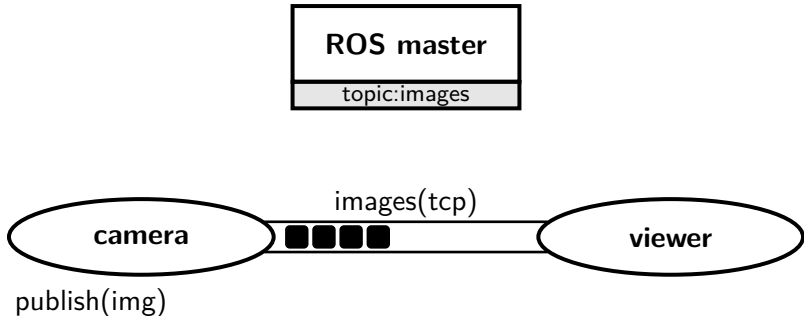




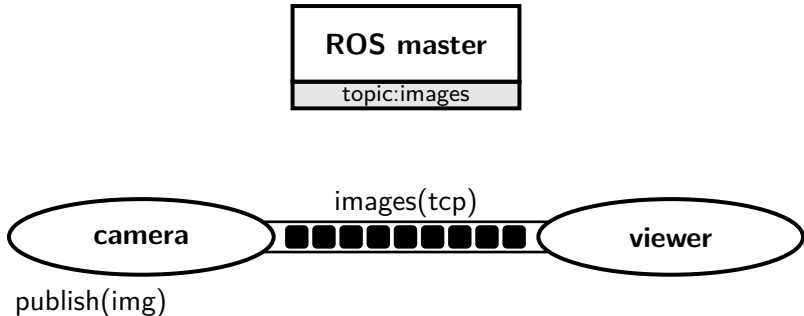
## Communication - Example



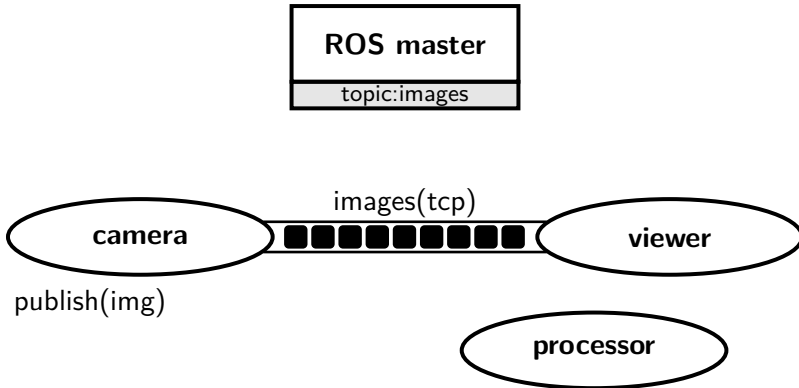
# Communication - Example



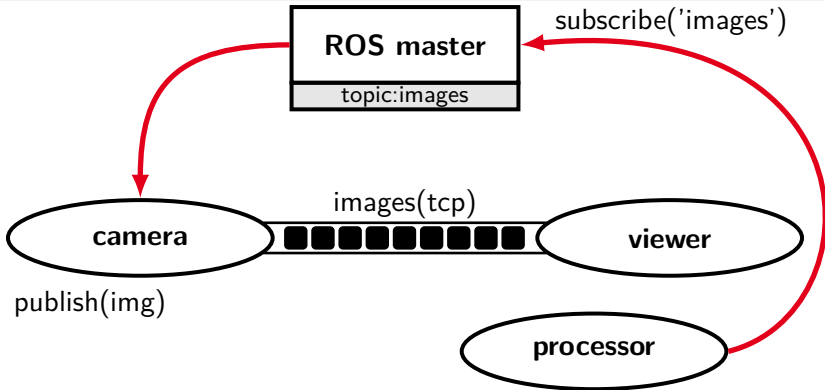
# Communication - Example



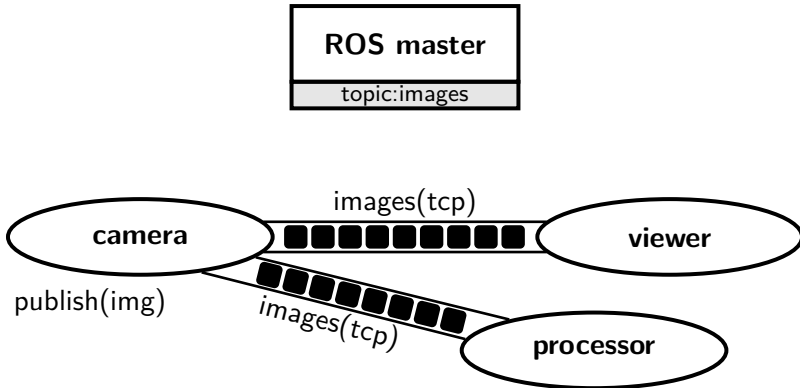
# Communication - Example



# Communication - Example



# Communication - Example





# Services

- ▶ 2 message types
  - ▶ request and response
- ▶ Synchronous protocol
  - ▶ client sends request
  - ▶ client waits for server
  - ▶ server replies

```
$ rosservice type add_two_ints | rossrv show
int64 a
int64 b
- - -
int64 sum
```



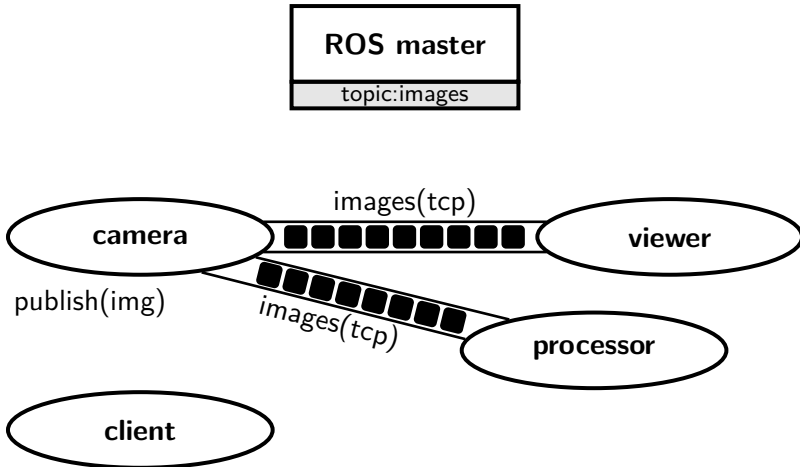
# Services

- ▶ 2 message types
  - ▶ request and response
- ▶ Synchronous protocol
  - ▶ client sends request
  - ▶ client waits for server
  - ▶ server replies

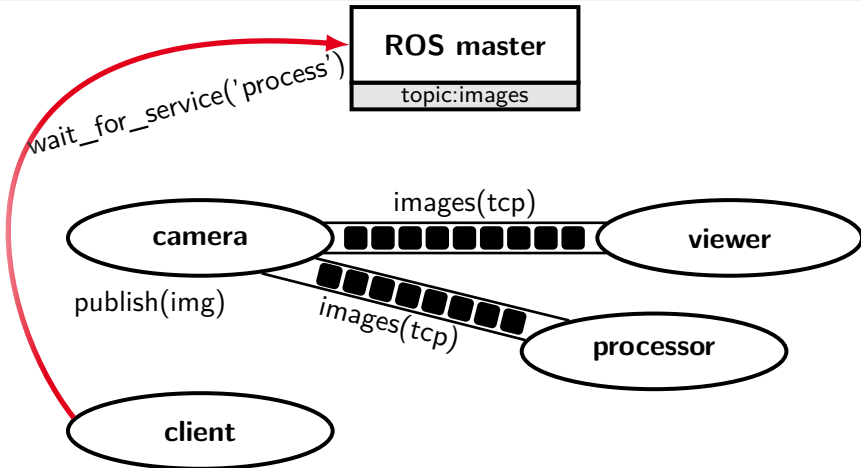
```
$ rosservice type add_two_ints | rossrv show
int64 a
int64 b
- - -
int64 sum
```



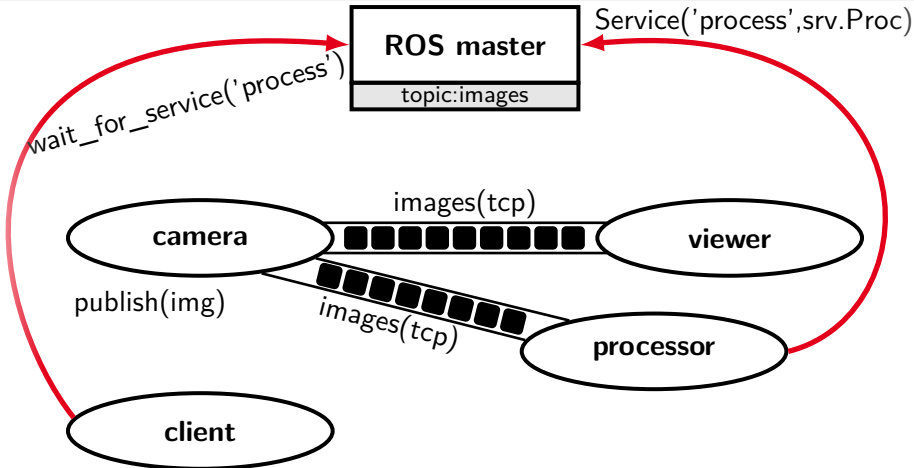
# Communication - Example



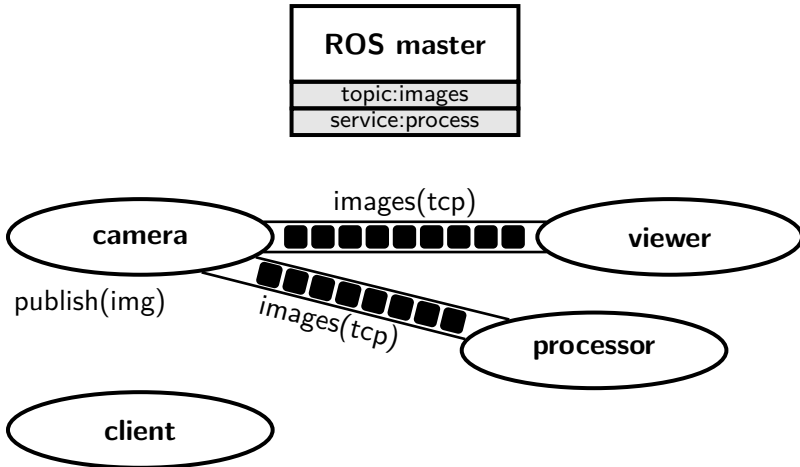
# Communication - Example



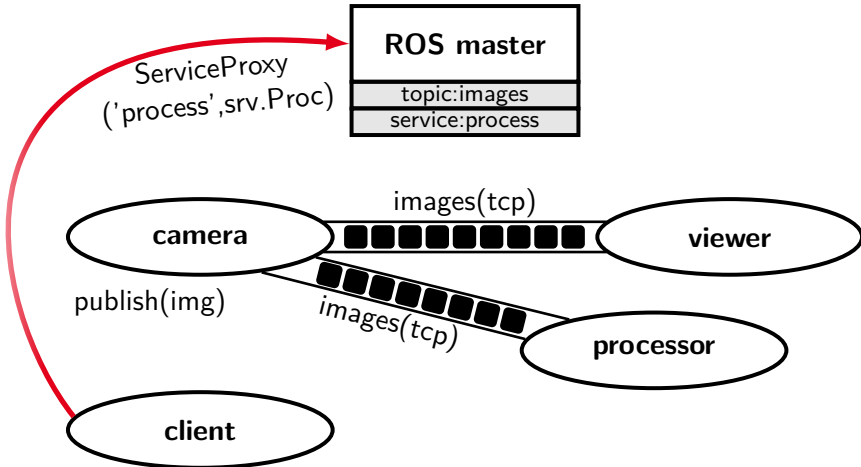
## Communication - Example



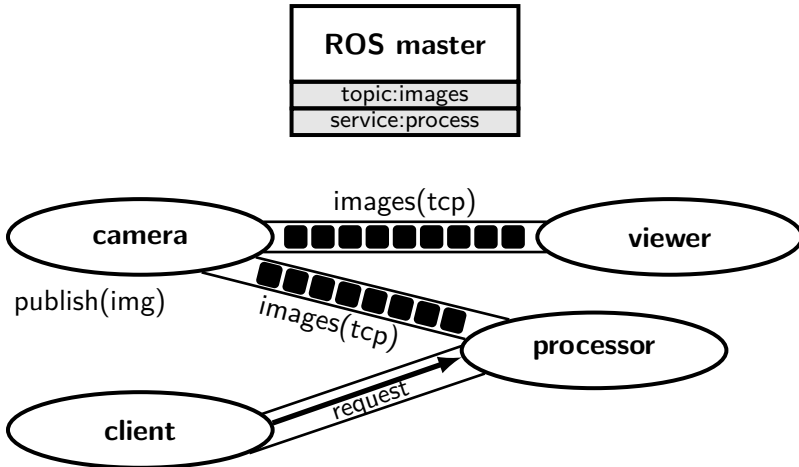
# Communication - Example



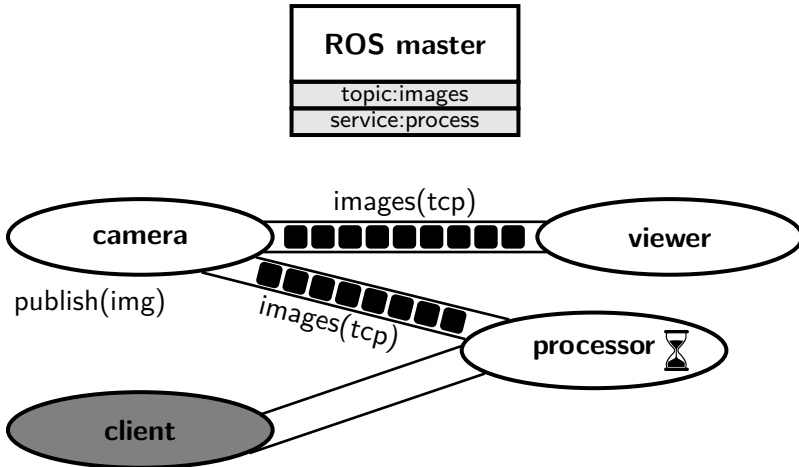
# Communication - Example



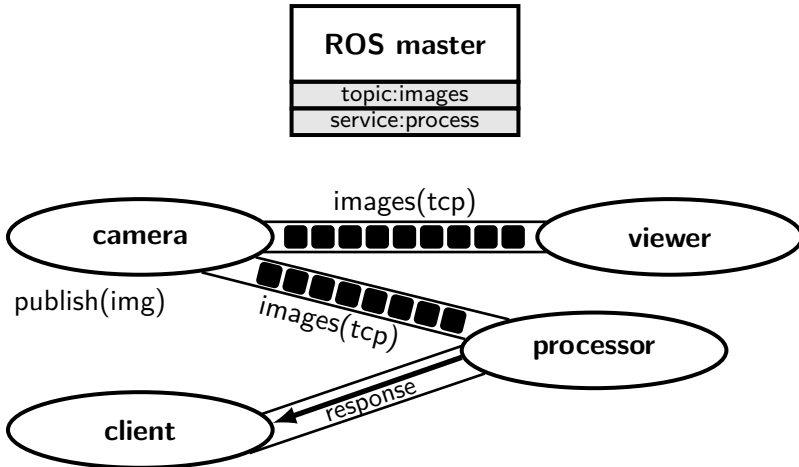
# Communication - Example



# Communication - Example



# Communication - Example







# Actions

- ▶ 3 message types
  - ▶ goal and result
  - ▶ optional feedback
- ▶ Asynchronous protocol
  - ▶ client sends goal
  - ▶ server may respond with feedback
  - ▶ server delivers result
- ▶ Interruptible

```
# Define the goal
uint32 dishwasher_id    # Specify which dishwasher we want to use
- - -
# Define the result
uint32 total_dishes_cleaned
- - -
# Define a feedback message
float32 percent_complete
```

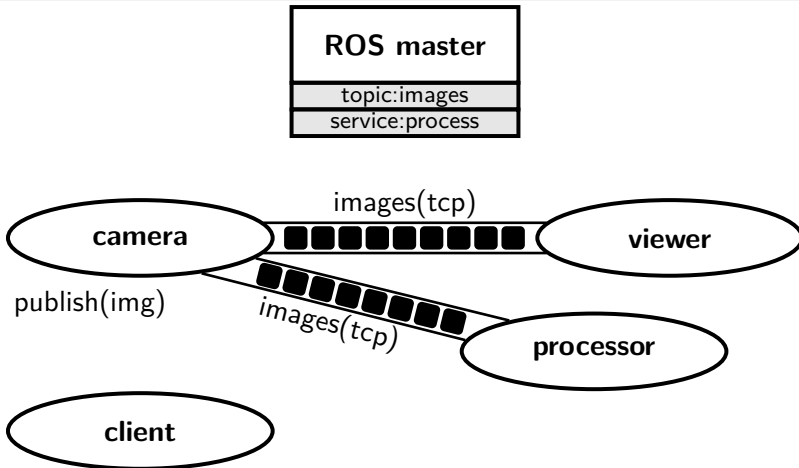


# Actions

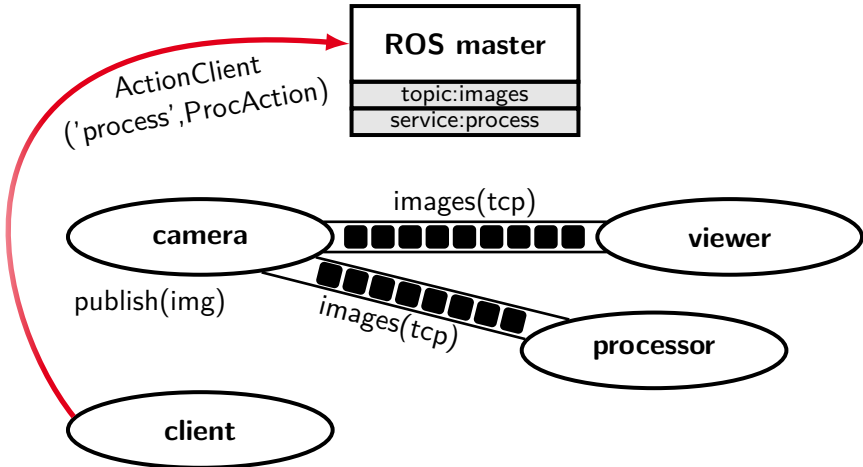
- ▶ 3 message types
  - ▶ goal and result
  - ▶ optional feedback
- ▶ Asynchronous protocol
  - ▶ client sends goal
  - ▶ server may respond with feedback
  - ▶ server delivers result
- ▶ Interruptible

```
# Define the goal
uint32 dishwasher_id    # Specify which dishwasher we want to use
- - -
# Define the result
uint32 total_dishes_cleaned
- - -
# Define a feedback message
float32 percent_complete
```

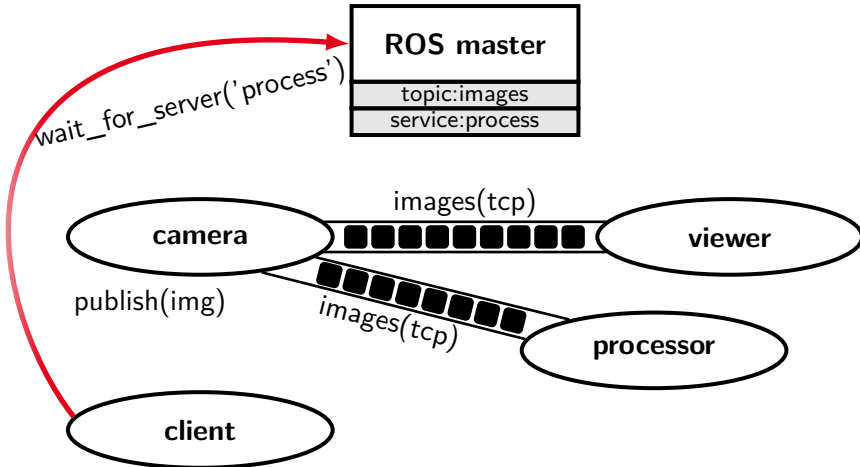
# Communication - Example



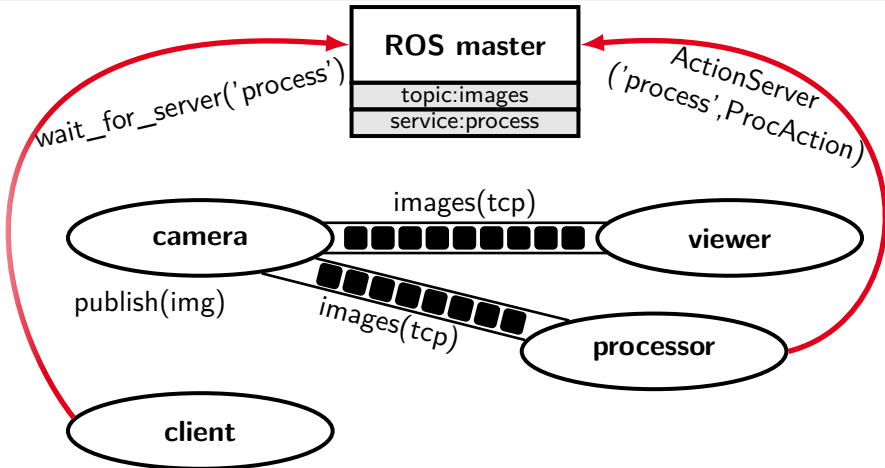
# Communication - Example



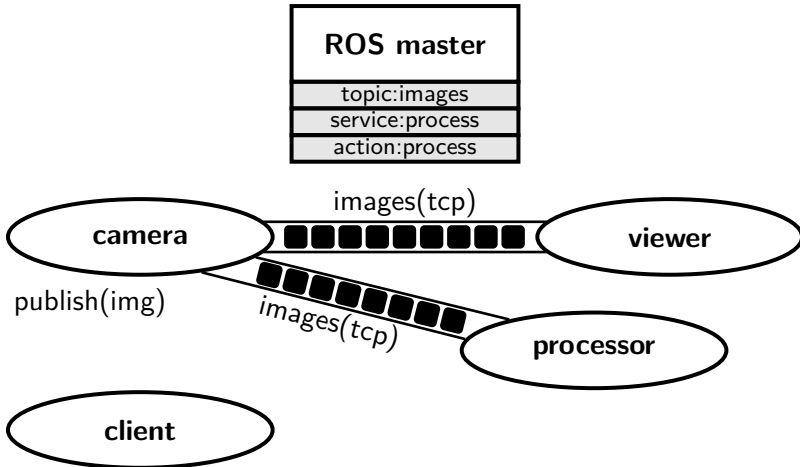
# Communication - Example



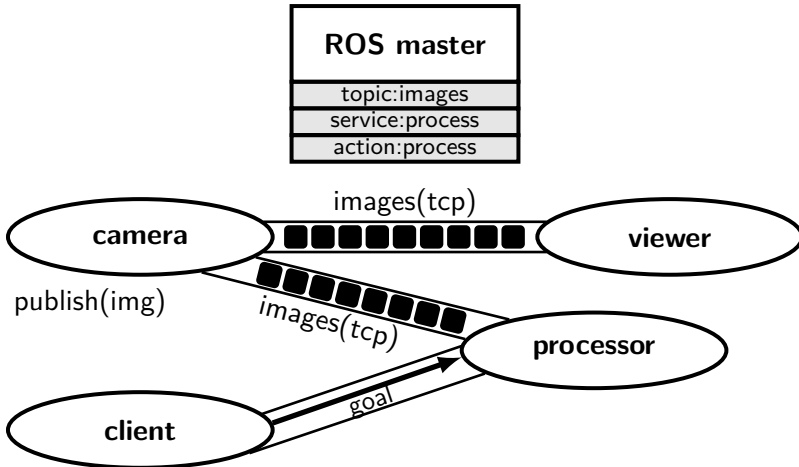
# Communication - Example



# Communication - Example

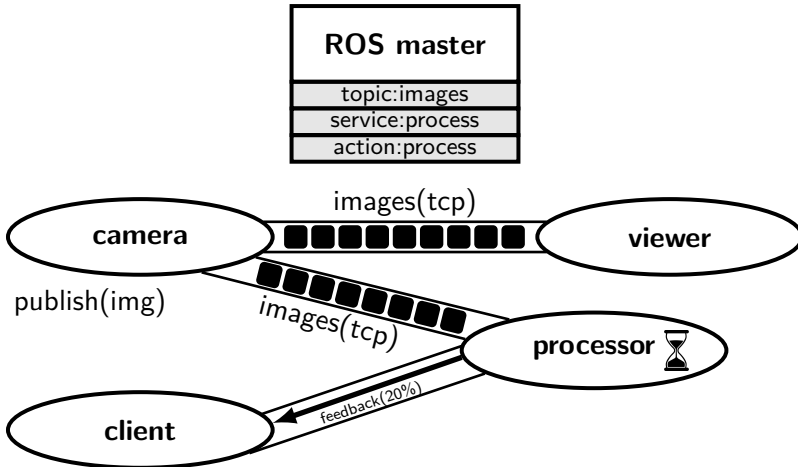


# Communication - Example

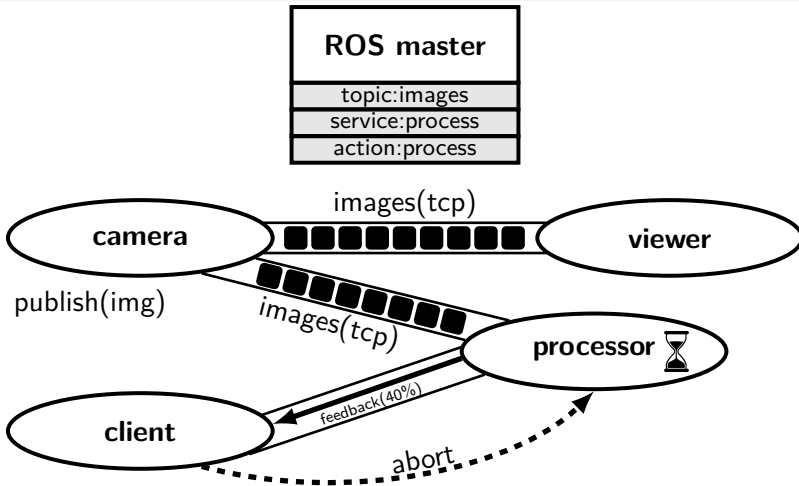




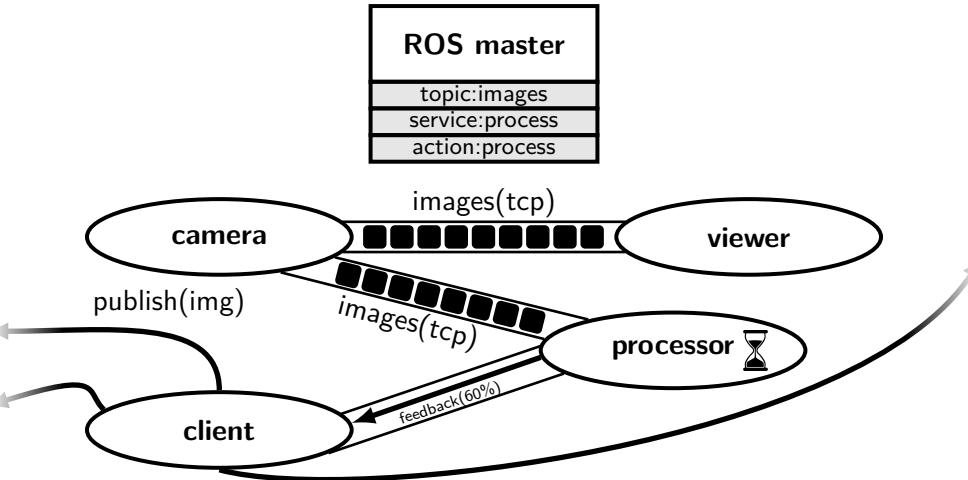
# Communication - Example



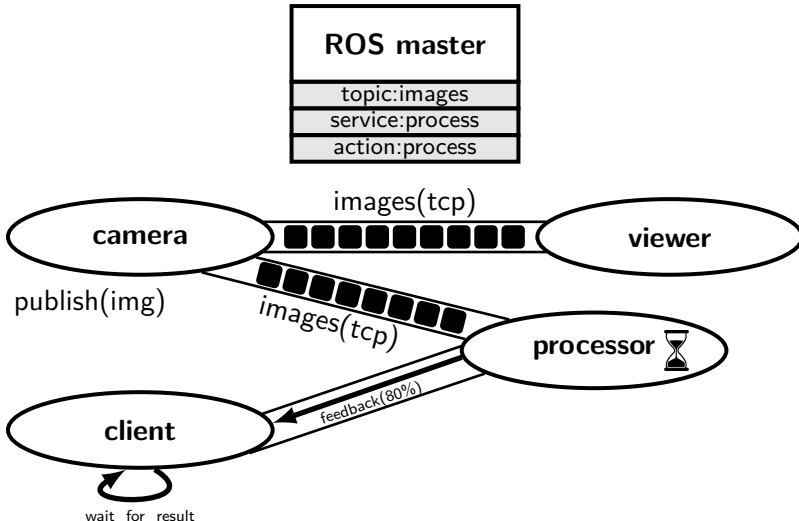
# Communication - Example



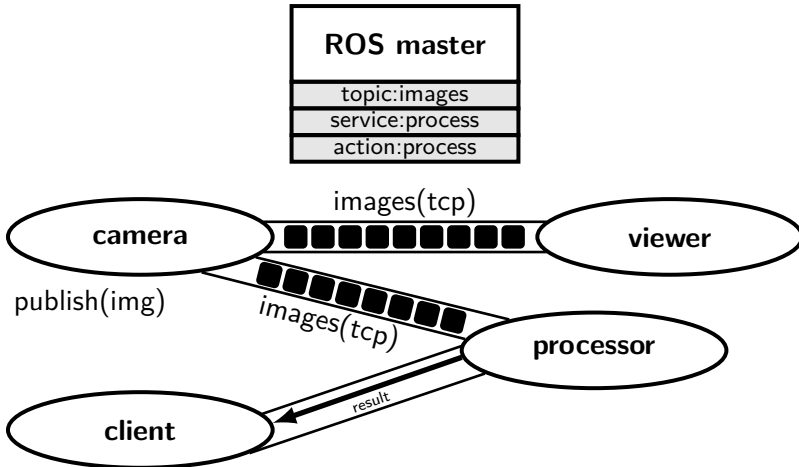
# Communication - Example



# Communication - Example



# Communication - Example





# Tools and Visualization

- ▶ Standardized interfaces allow using tools in various applications
- ▶ ROS-provided tools
  - ▶ ROS Bag
  - ▶ RQT
  - ▶ RViz
- ▶ User-provided tools
  - ▶ PlotJuggler
  - ▶ RQT-Plugins
  - ▶ Teleoperation node



# ROS Bag

- ▶ Collects messages sent over topics
- ▶ Includes time component
- ▶ Allows to capture a situation on the robot and debug nodes independently
- ▶ Provides programming interface

# RQT

- ▶ User interaction framework for the ROS environment
- ▶ Relies on various plugins
- ▶ Standard plugins are provided
- ▶ Custom plugins can be written

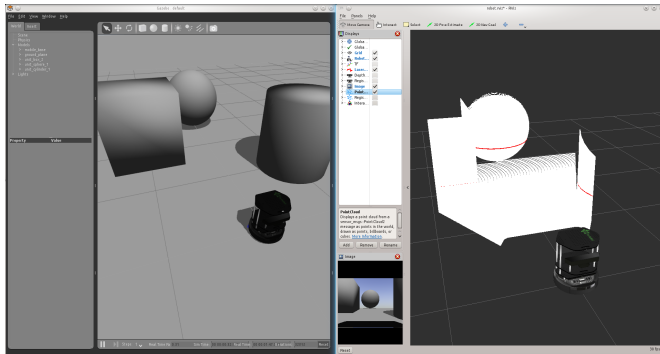
S. Liu

18



# RViz

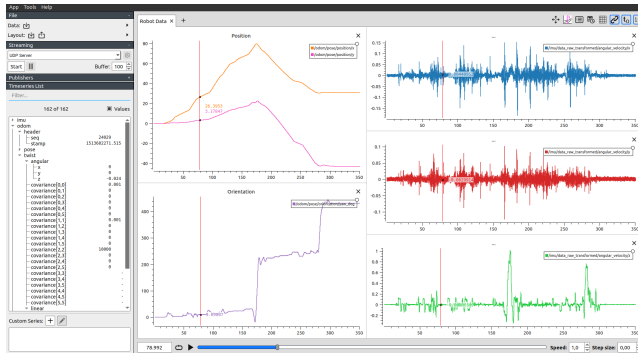
- ▶ 3D visualization environment
- ▶ Different data can be shown
  - ▶ Laser scan data, map, ...



Source: [http://wiki.ros.org/turtlebot\\_gazebo](http://wiki.ros.org/turtlebot_gazebo)

# PlotJuggler

- ▶ Visualization of data over time
- ▶ Different types of data streams can be shown



Source: <https://github.com/facontidavide/PlotJuggler>



# Simulations

- ▶ Important development tool
  - ▶ protects expensive hardware
  - ▶ develop and test without robot
  - ▶ high-level test
- ▶ Simulates sensor data
  - ▶ clean data
- ▶ Turtlesim
  - ▶ ROS learning tool
- ▶ Gazebo
  - ▶ ROS simulator
- ▶ Webots
  - ▶ Robotics simulator



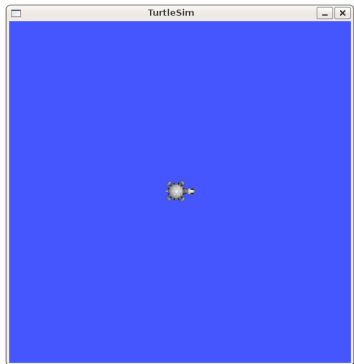
# Simulations

- ▶ Important development tool
  - ▶ protects expensive hardware
  - ▶ develop and test without robot
  - ▶ high-level test
- ▶ Simulates sensor data
  - ▶ clean data
- ▶ Turtlesim
  - ▶ ROS learning tool
- ▶ Gazebo
  - ▶ ROS simulator
- ▶ Webots
  - ▶ Robotics simulator



# Turtle Sim

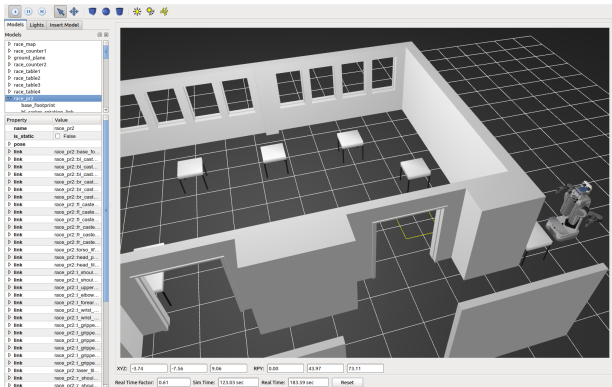
- ▶ Learning platform
- ▶ 2D turtle
  - ▶ move
  - ▶ turn
  - ▶ draw
- ▶ Communication
- ▶ ROS structure



Source: <http://wiki.ros.org/turtlesim>

# Gazebo

- ▶ 3D rigid body simulator
- ▶ Simulates robots, environment and sensor data



Source: Lasse Einig

# Webots

- ▶ 3D rigid body simulator
- ▶ Simulates robots, environment and sensor data



Source: Jonas Hagge