



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

MIN Faculty
Department of Informatics



Introduction to ROS

Lasse Einig, Dennis Krupke, Florens Wasserfall



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

April 12, 2019



Outline

Foundation

Structure

Communication

Kinematics Tool

Foundation

Structure

Communication

Kinematics Tool



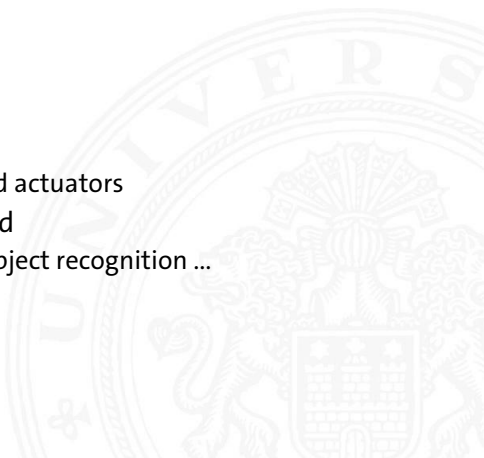


- ▶ Heterogeneity vs. Homogeneity
 - ▶ sensor types, actuators, ...
 - ▶ sensor model, kinematic chain, ...
- ▶ Abstraction
- ▶ Algorithm re-usability
 - ▶ 2D laser data mapping
 - ▶ object recognition
- ▶ Debugging
 - ▶ simulation, data visualization, ...



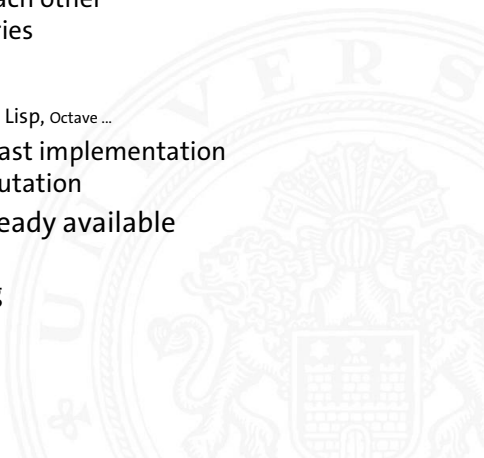


- ▶ Robot Operating System
- ▶ Meta operating system
- ▶ Open source
- ▶ Hardware abstraction
 - ▶ portability
 - ▶ simplification of sensors and actuators
- ▶ Recurring tasks already solved
 - ▶ Navigation, data filtering, object recognition ...





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





Outline

Foundation

Structure

Communication

Kinematics Tool

Foundation

Structure

Communication

Kinematics Tool



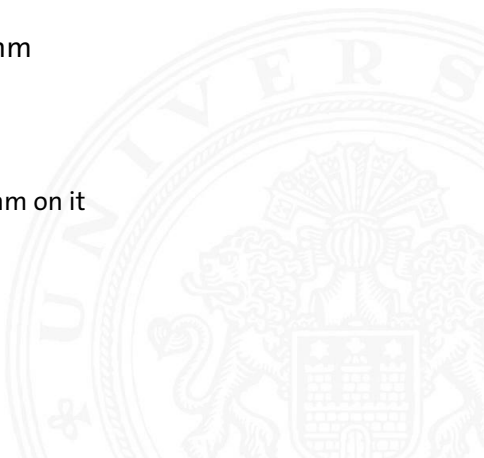


- ▶ ROS nodes
 - ▶ sensors
 - ▶ actuators
 - ▶ logic
- ▶ ROS core
- ▶ Communication





- ▶ 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





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





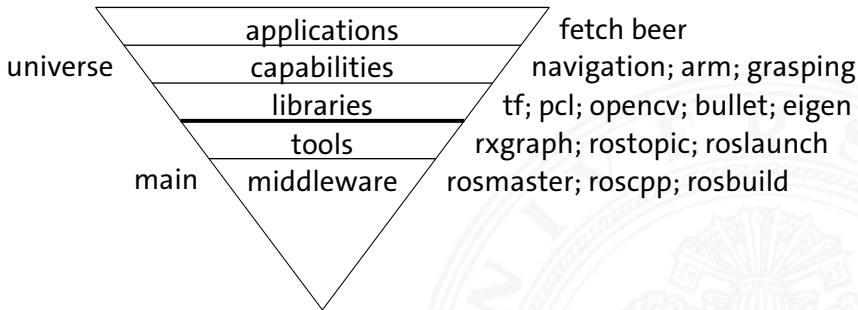
- ▶ Messages
 - ▶ standardized data types
- ▶ Topics
 - ▶ n:n communication
- ▶ Services and Actions
 - ▶ 1:1 communication





- ▶ Exploration
- ▶ Localization
- ▶ Detection
- ▶ One node per sensor
 - ▶ provide data as topic
 - ▶ abstract from hardware





- ▶ universe → robot centric, developed by community
- ▶ main → general tools, maintained by OSRF



Outline

Foundation

Structure

Communication

Kinematics Tool

Foundation

Structure

Communication

- Messages

- Topics

- Services

- Actions

Kinematics Tool





- ▶ 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
```



- ▶ 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
```



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





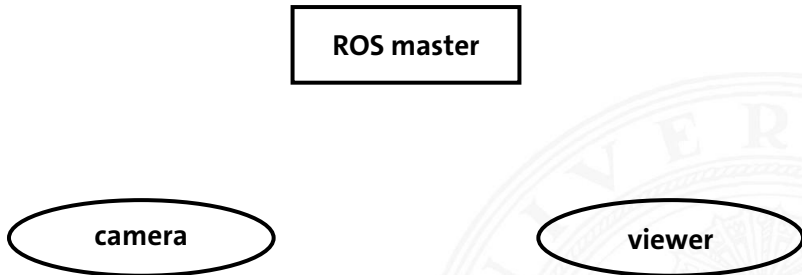
Communication – Example

Foundation

Structure

Communication

Kinematics Tool





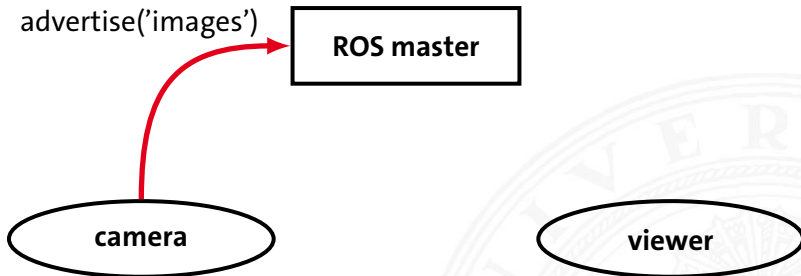
Communication – Example

Foundation

Structure

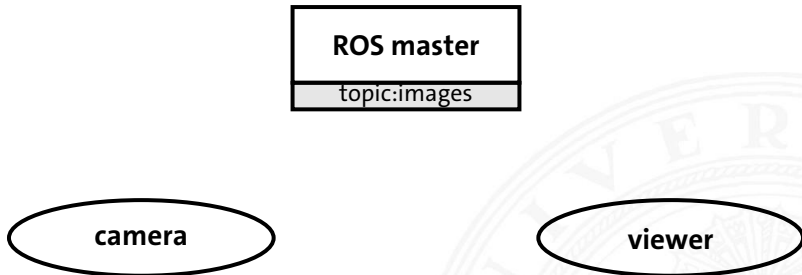
Communication

Kinematics Tool

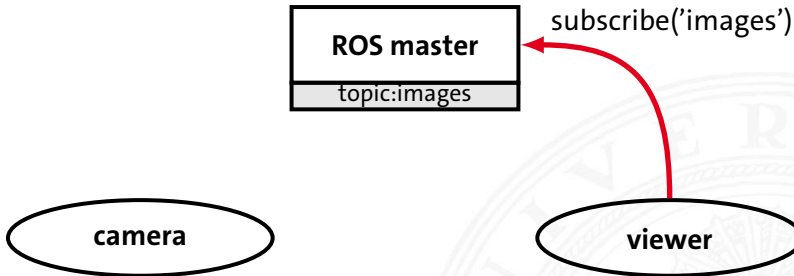




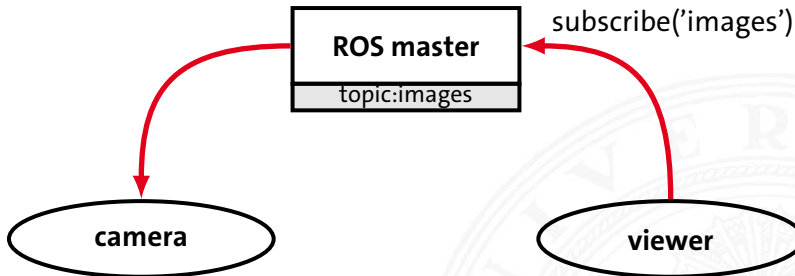
Communication – Example



Communication – Example

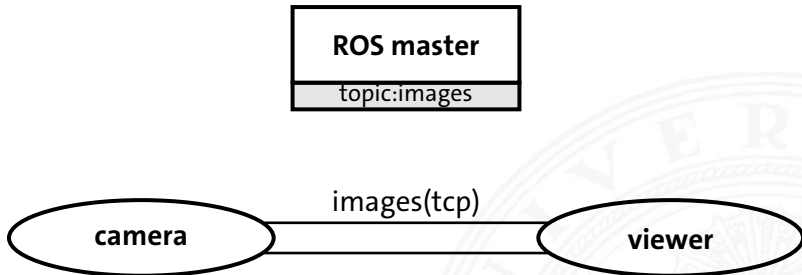


Communication – Example

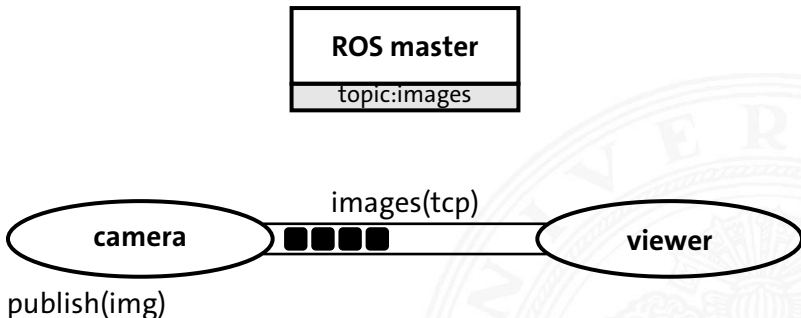




Communication – Example

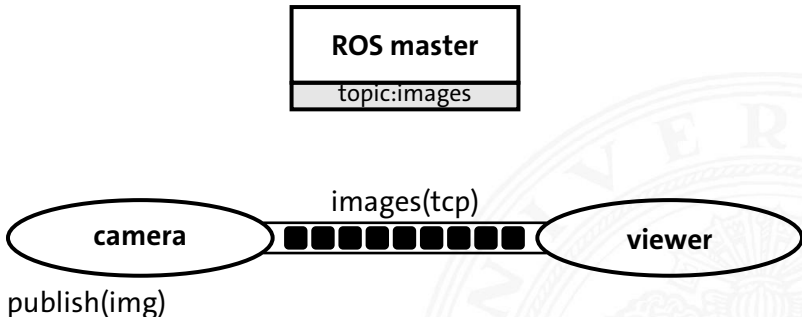


Communication – Example

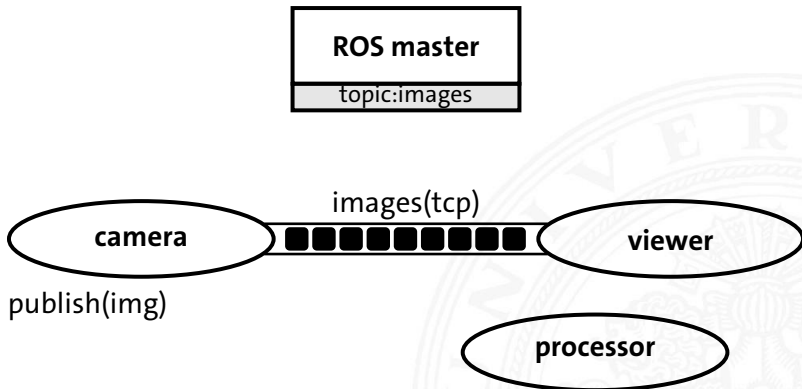




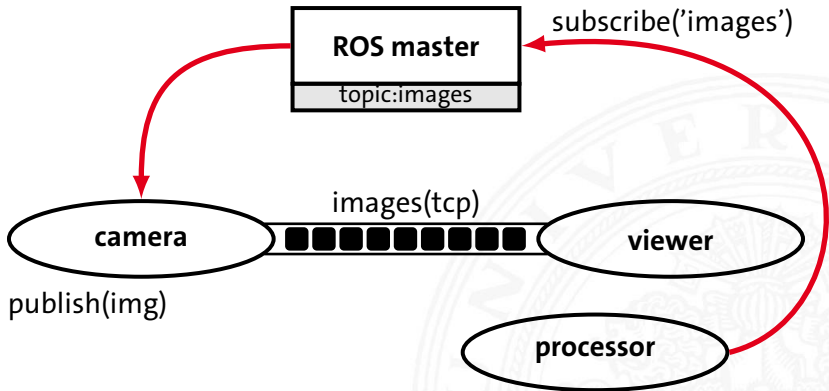
Communication – Example



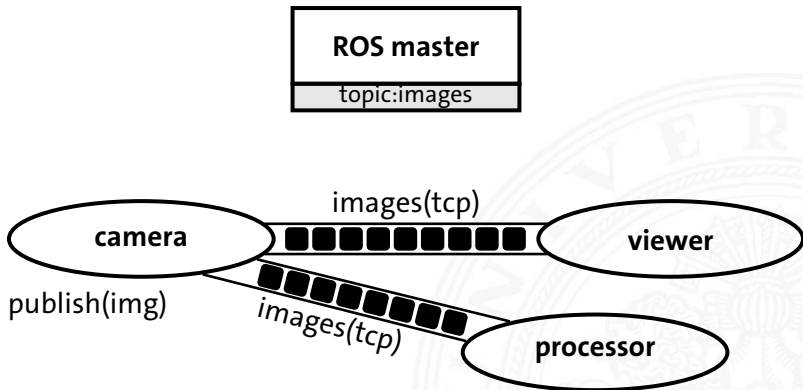
Communication – Example



Communication – Example



Communication – Example





- ▶ 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
```

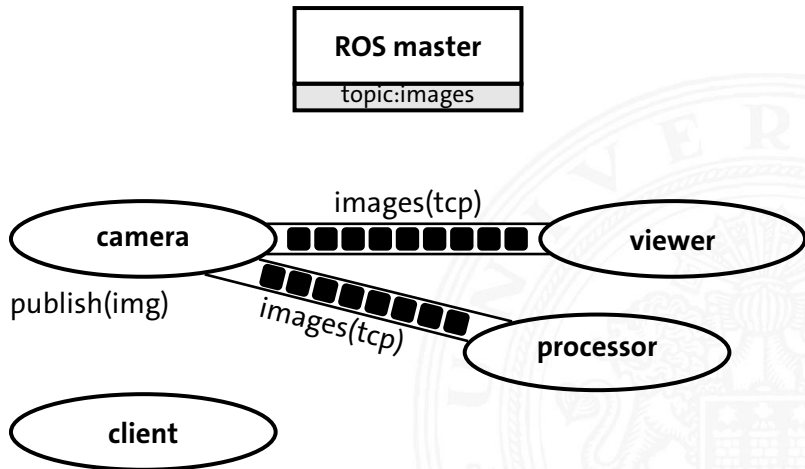




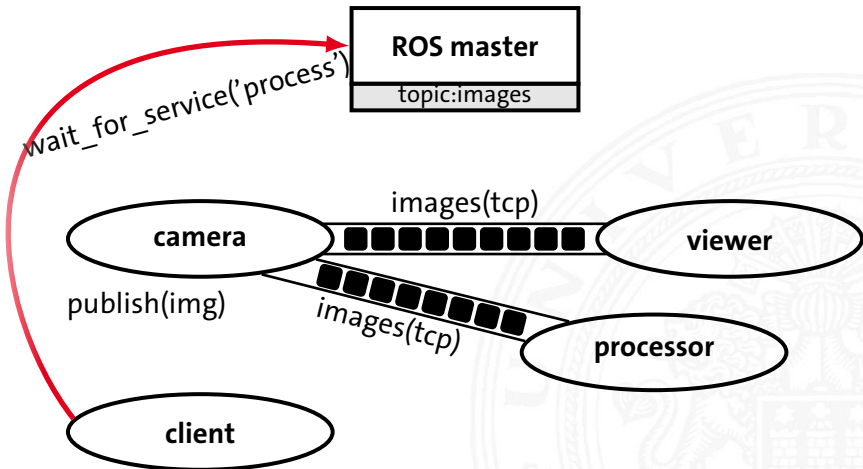
- ▶ 2 message types
 - ▶ request and response
- ▶ Synchronous protocol
 - ▶ client sends request
 - ▶ client waits for server
 - ▶ server replies

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

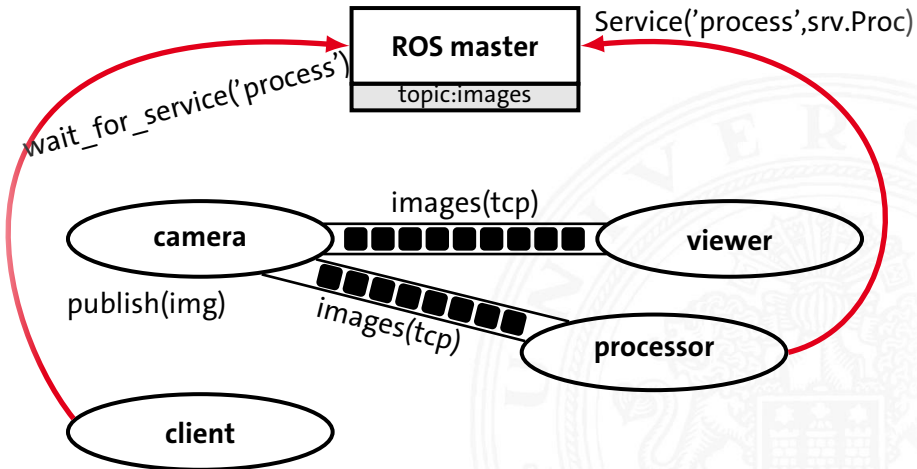
Communication – Example



Communication – Example

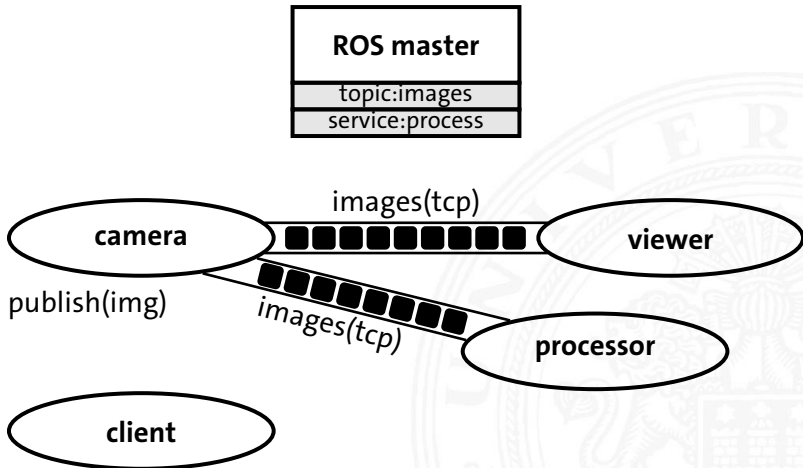


Communication – Example

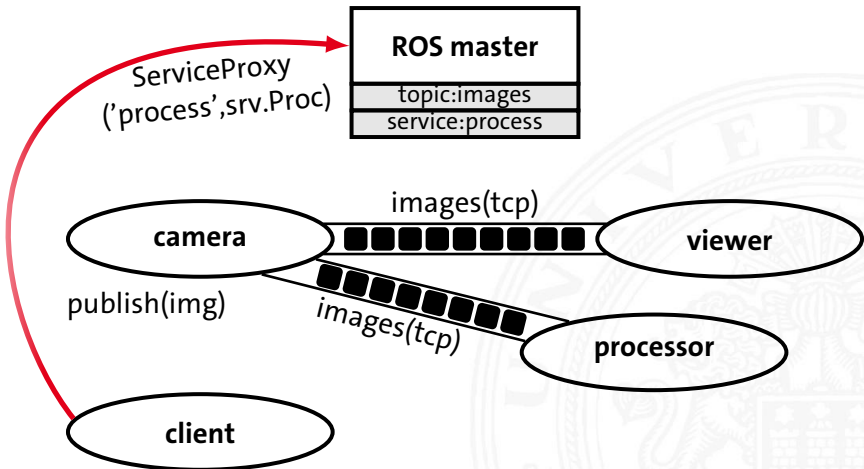




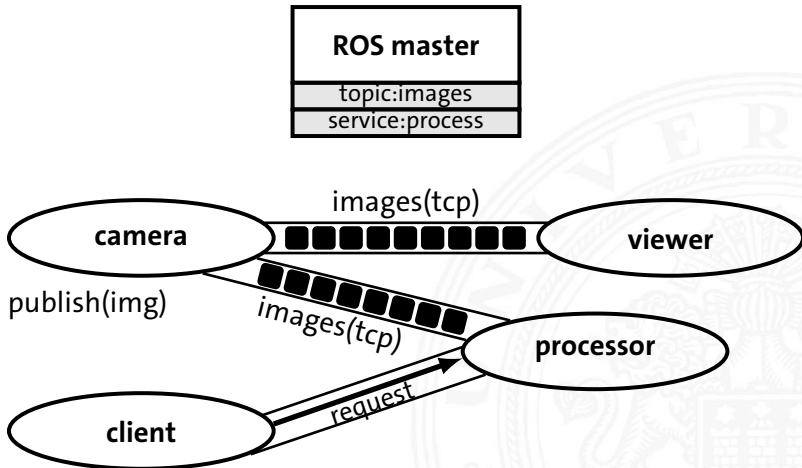
Communication – Example



Communication – Example

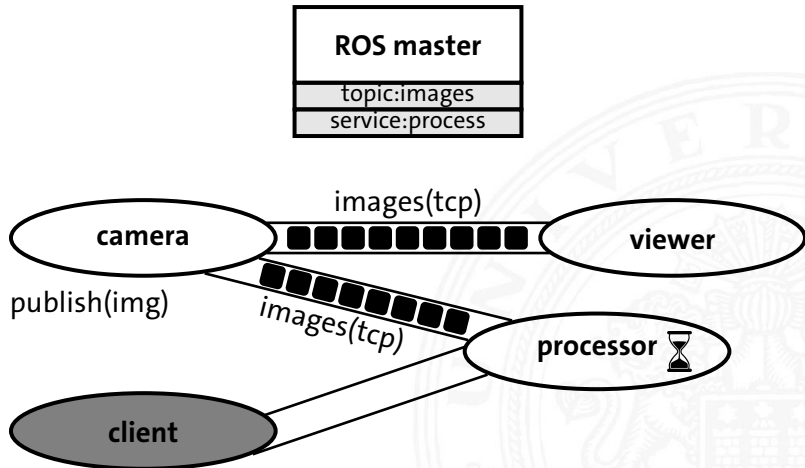


Communication – Example



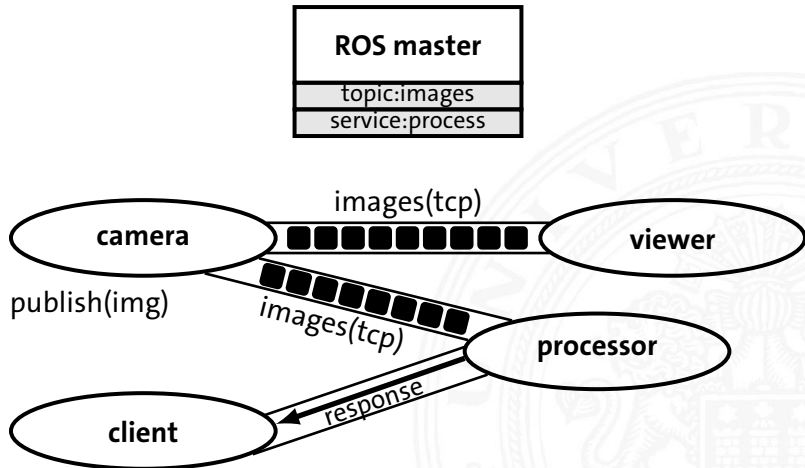


Communication – Example





Communication – Example





- ▶ 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
```

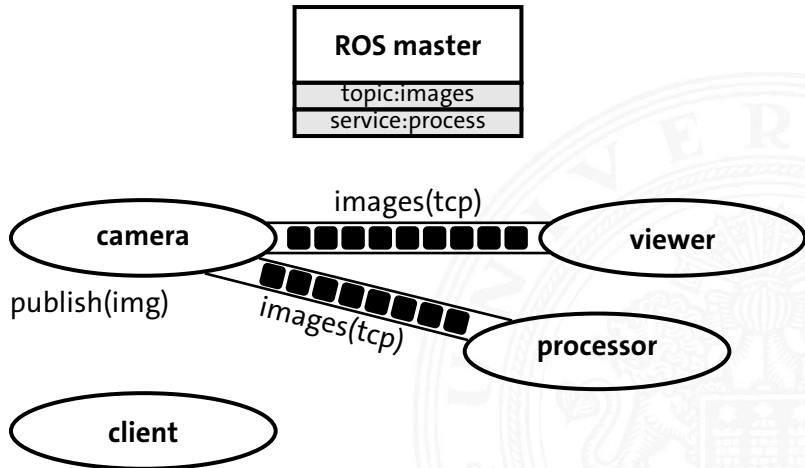


- ▶ 3 message types
 - ▶ goal and result
 - ▶ optional feedback
- ▶ Asynchronous protocol
 - ▶ client sends goal
 - ▶ server may respond with feedback
 - ▶ server delivers result
- ▶ Interruptible

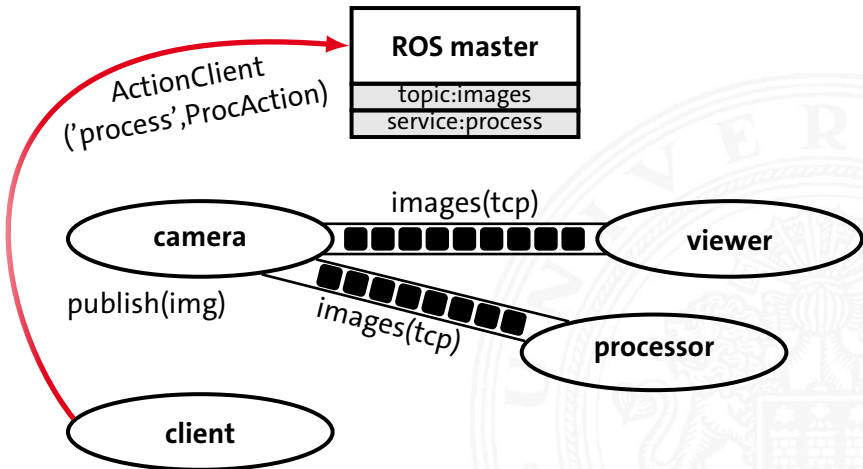
```
# Define the goal
uint32 dishwasher_id      # Specify which dishwasher we want to use
- - -
# Define the result
uint32 total_dishes_cleaned
- - -
# Define a feedback message
float32 percent_complete
```



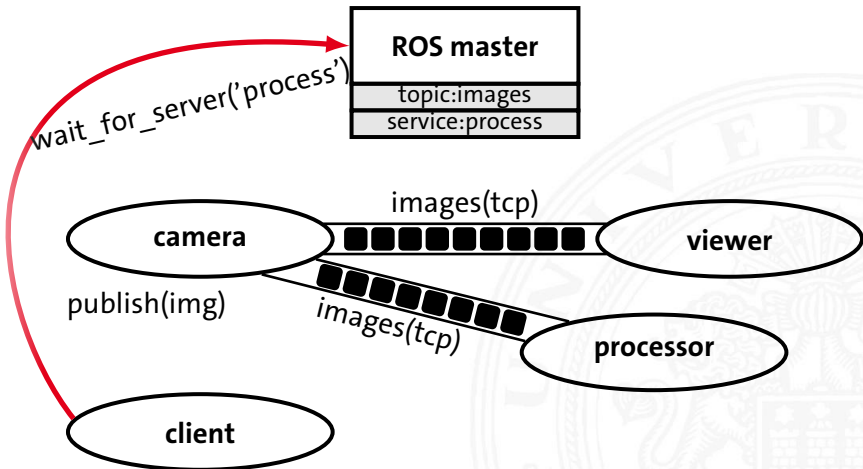
Communication – Example



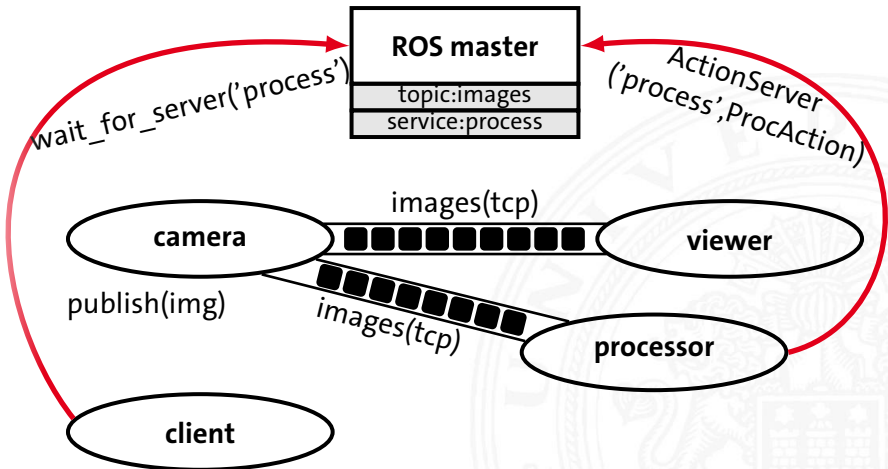
Communication – Example



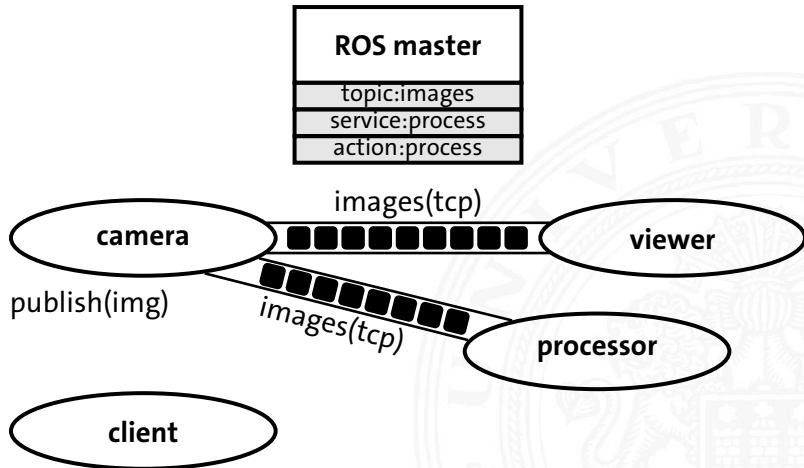
Communication – Example



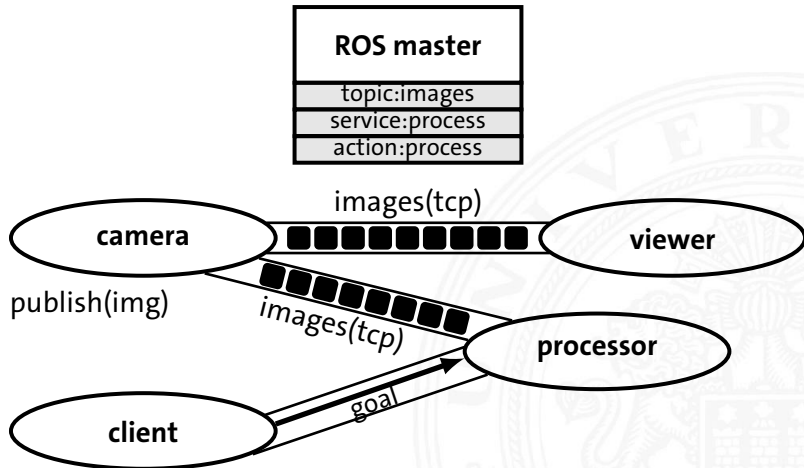
Communication – Example



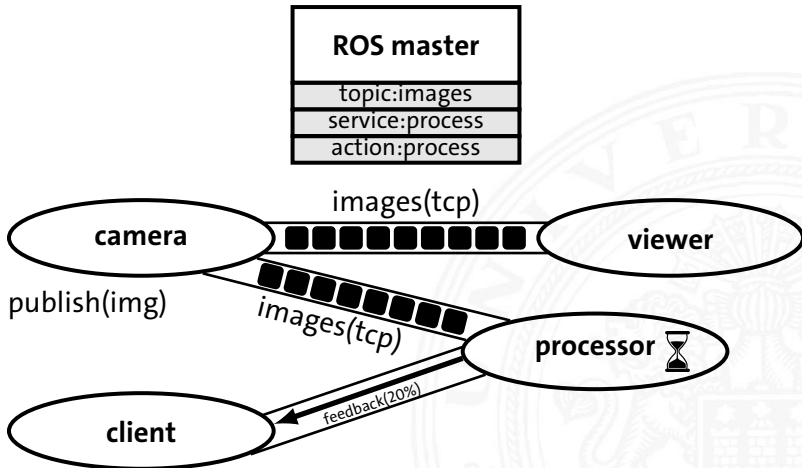
Communication – Example



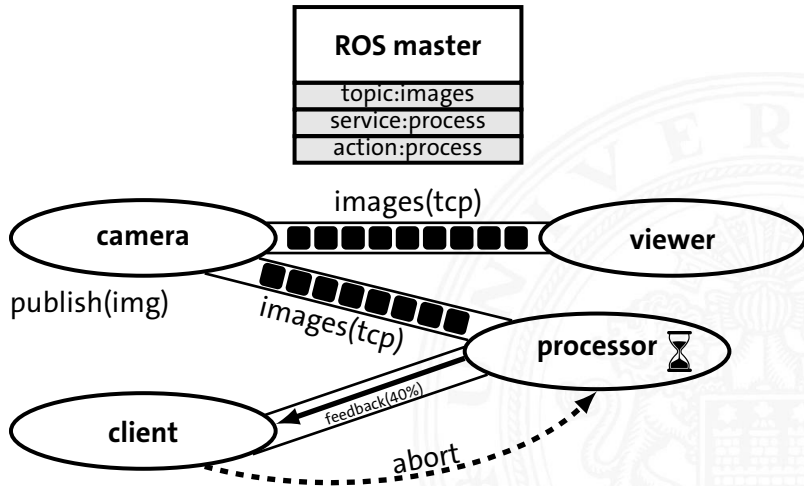
Communication – Example



Communication – Example

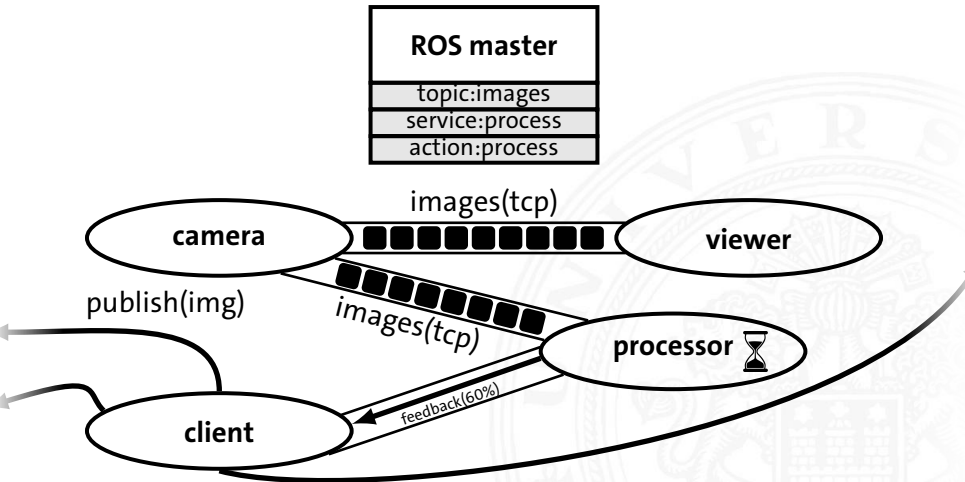


Communication – Example

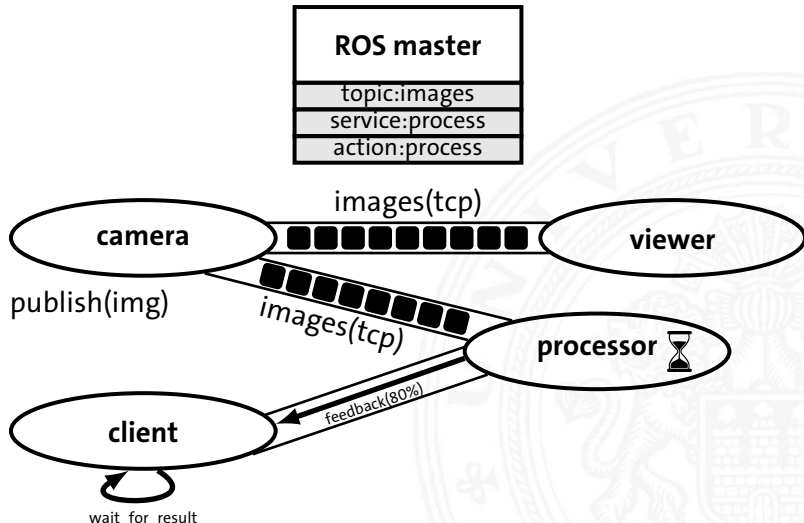




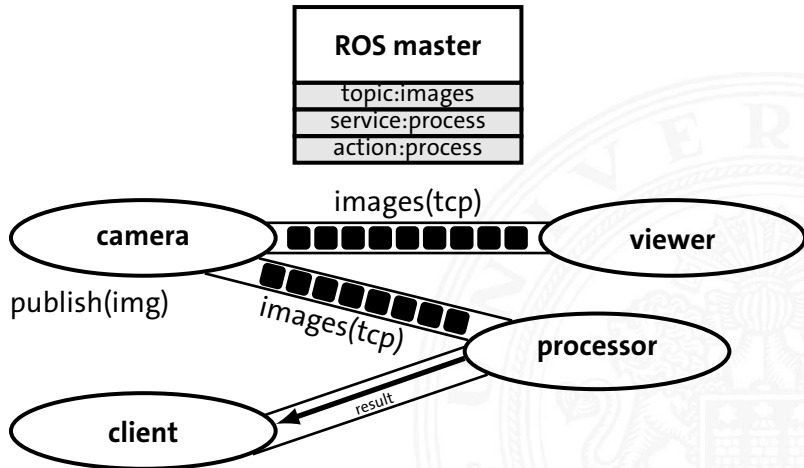
Communication – Example



Communication – Example



Communication – Example





Outline

Foundation

Structure

Communication

Kinematics Tool

Foundation

Structure

Communication

Kinematics Tool



► Visualization for kinematics of robot models

The screenshot displays the Kinematics Tool interface. The left sidebar contains the following configuration options:

- Global Options**
 - Fixed Frame: base_link
 - Background Color: 48; 48; 48
 - Frame Rate: 30
 - Default Light:
 - Global Mouse: OK
 - Fixed Frame: OK
 - Grid:
- Reference Frame** (Fixed Frame)
 - Plane Cell Count: 10
 - Normal Cell Count: 0
 - Cell Size: Lines
 - Line Style: 160; 160; 164
 - Color:
 - Alpha: 0.5
 - Plane: XY
 - Offset: 0; 0; 0
- RobotModel** (checked)
 - Visual Enabled:
 - Collision Enabled:
 - Update Interval: 0
 - Alpha: 1
 - Robot Description: robot_description
 - TF Prefix:
 - Links**
 - Link Tree Style: Links in alphabetic Order
 - Expand Link D...:
 - All Links Enabl...:
 - base_link:
 - first_link_cyl:
 - first_link_c...:
 - second_lin...:
 - top:
 - Alpha: 1
 - Show Trail:
 - Show Axes:
 - Position: 2; 0; 0.05
 - Orientation: 0; 0; 0; 1

The right sidebar shows camera settings for Orbit (vixl):

- Type: Orbit (vixl) | Zero
- Current View: Orbit (vixl)
- Near Clip: 0.01
- Invert Z Axis:
- Target Fra...: Fixed Frame
- Distance: 10
- Focal Snap: 0.05
- Focal Snap...:
- Yaw: 0.785398
- Pitch: 0.785398
- Focal Point: 0; 0; 0

The bottom status bar displays:

- ROS Time: 1523362081.40
- ROS Elapsed: 29.25
- Wall Time: 1523362081.42
- Wall Elapsed: 29.25
- Experimental:
- 31 fps



- ▶ Obeys physical constraints of the robot model
- ▶ Based on 100 Hz control loop
- ▶ Forward control by sending joint target messages
- ▶ Offers IK solving service
- ▶ Simple setup
- ▶ Programmable in Python and C++
 - ▶ Samples only in Python

