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

# 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
- 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, 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

# ROS System



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

# 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

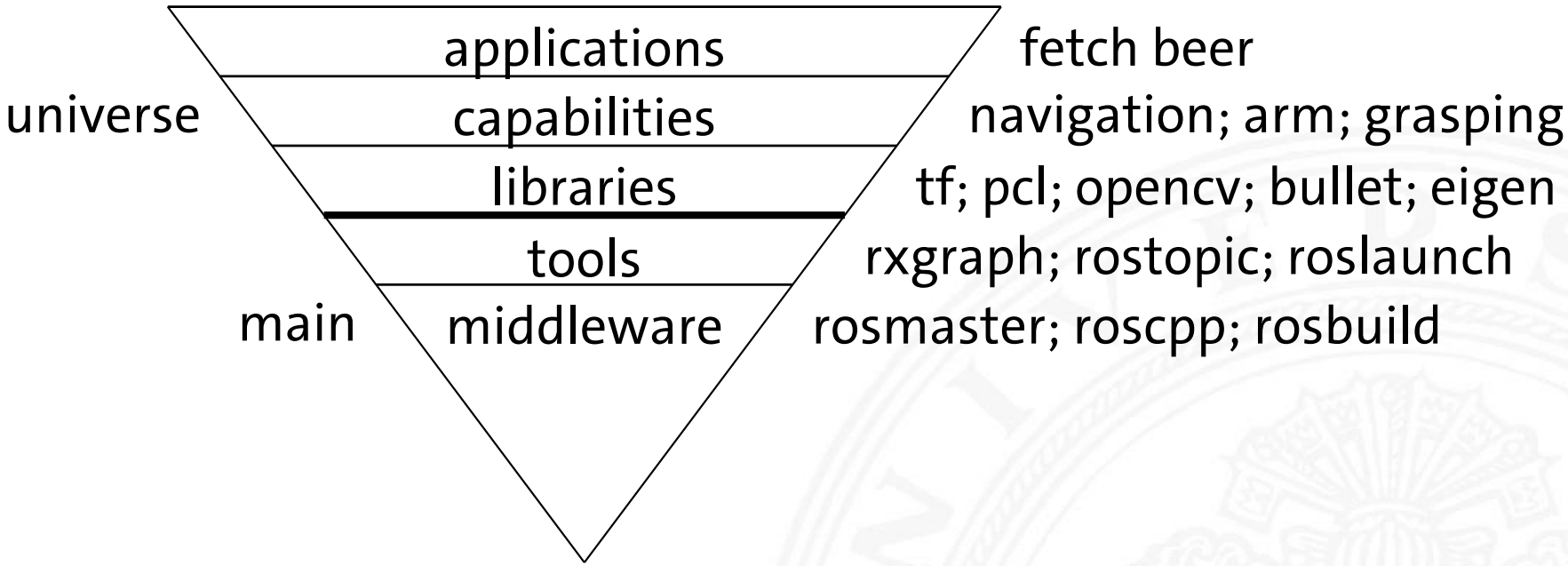
# Sensors



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

# System structure






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

# Outline

# Messages



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

```
$ rosmsg 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

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

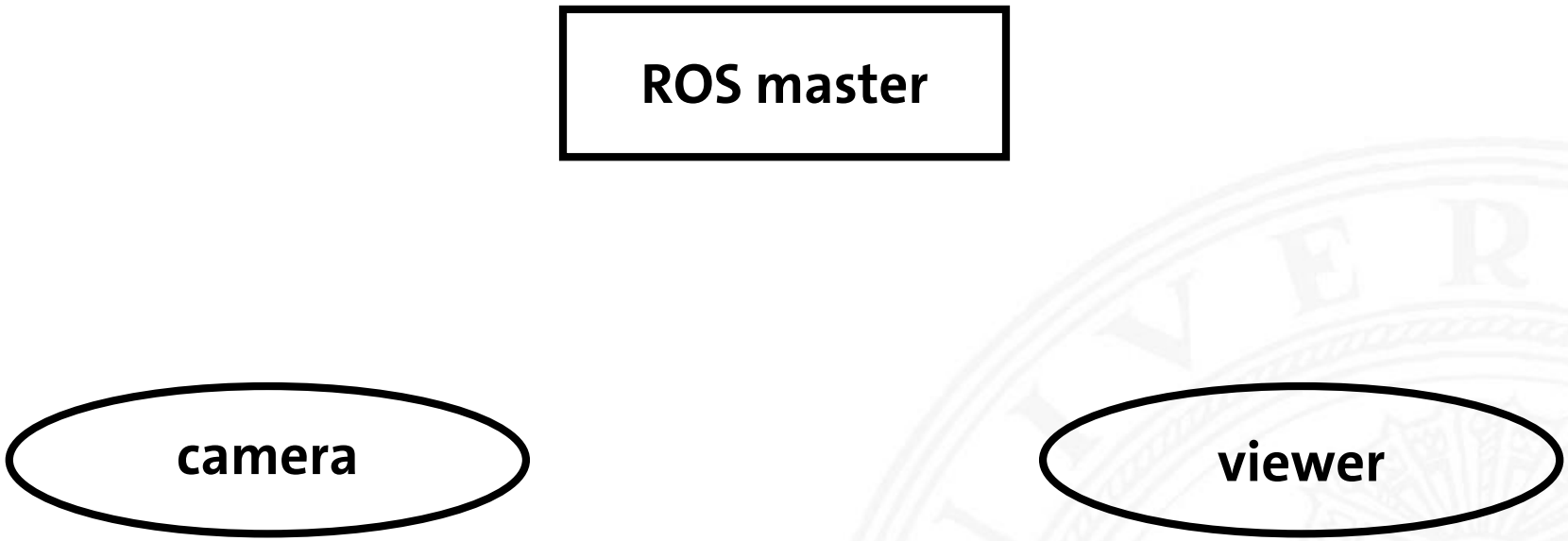
# Topics



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

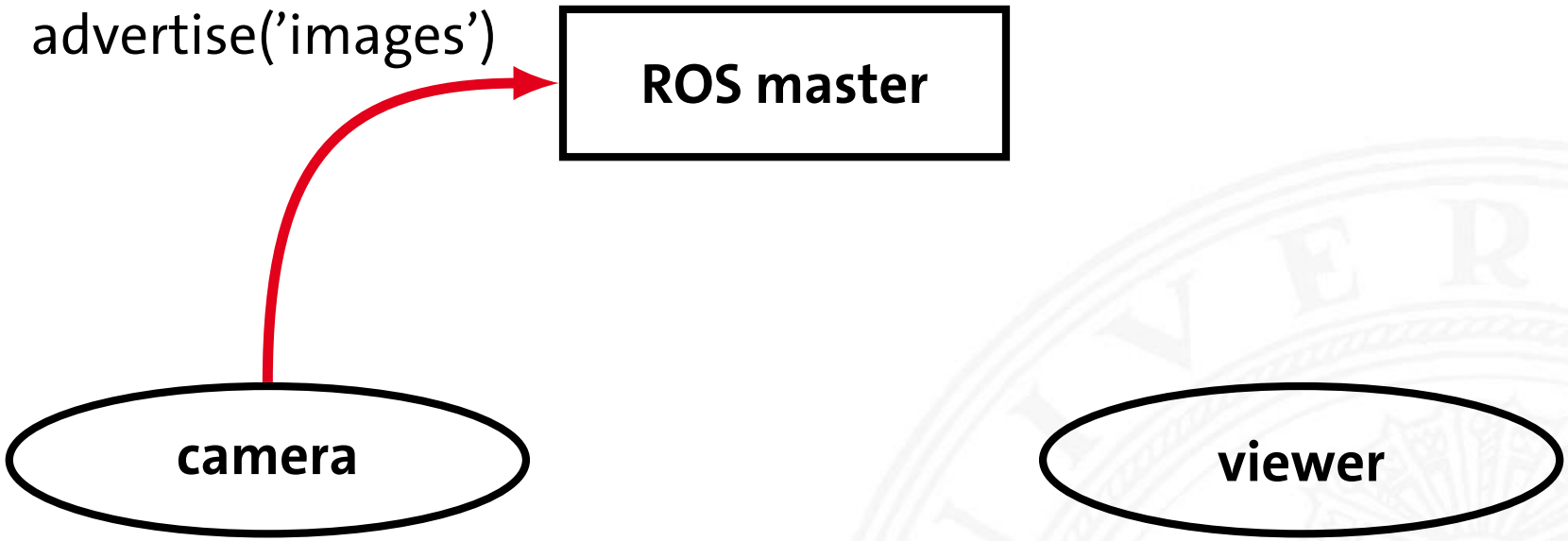
# Communication – Example





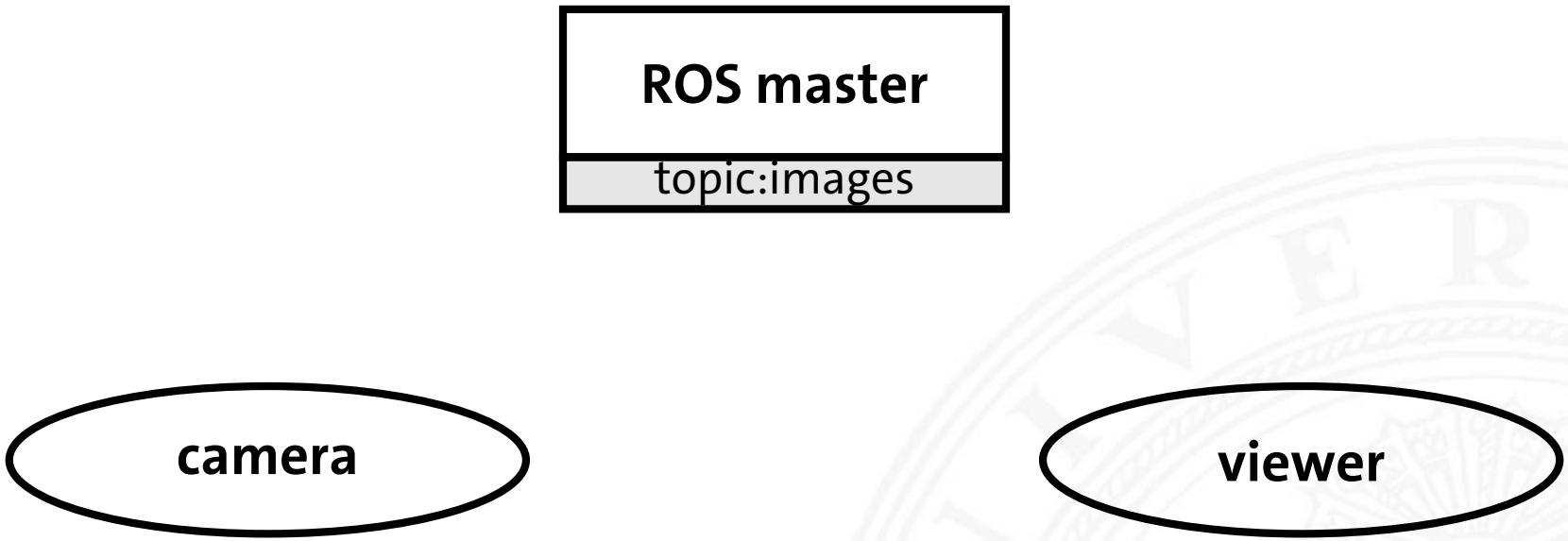
# Communication – Example





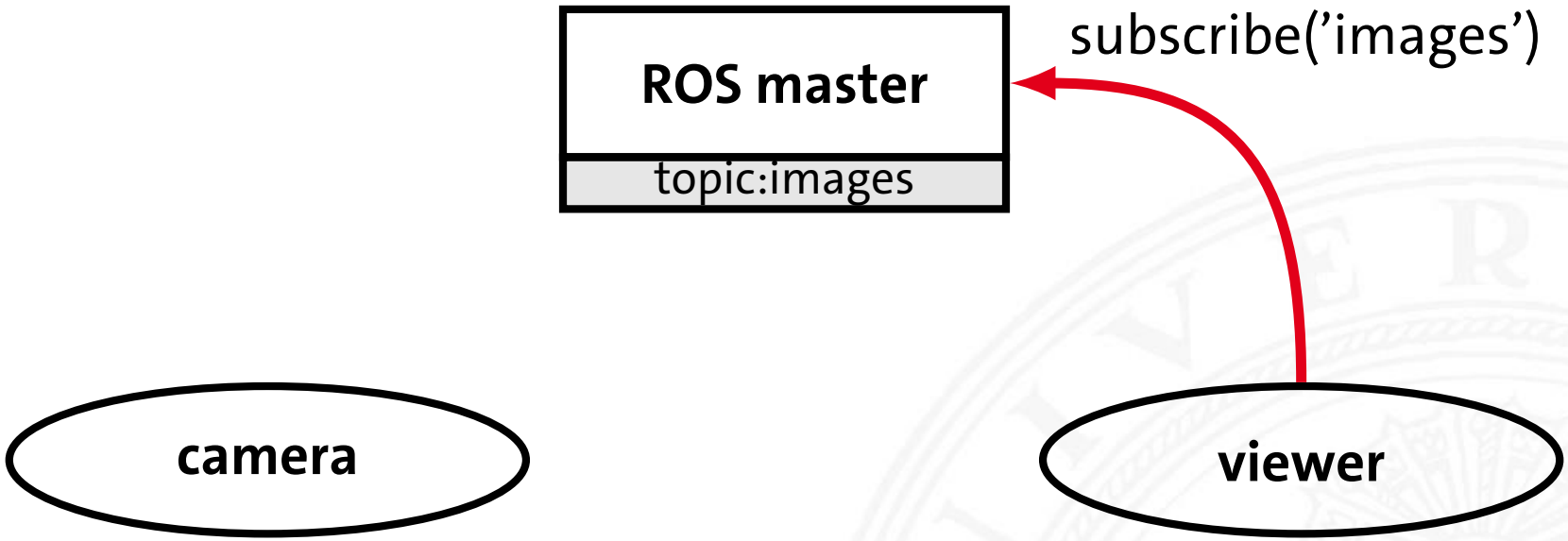
# Communication – Example





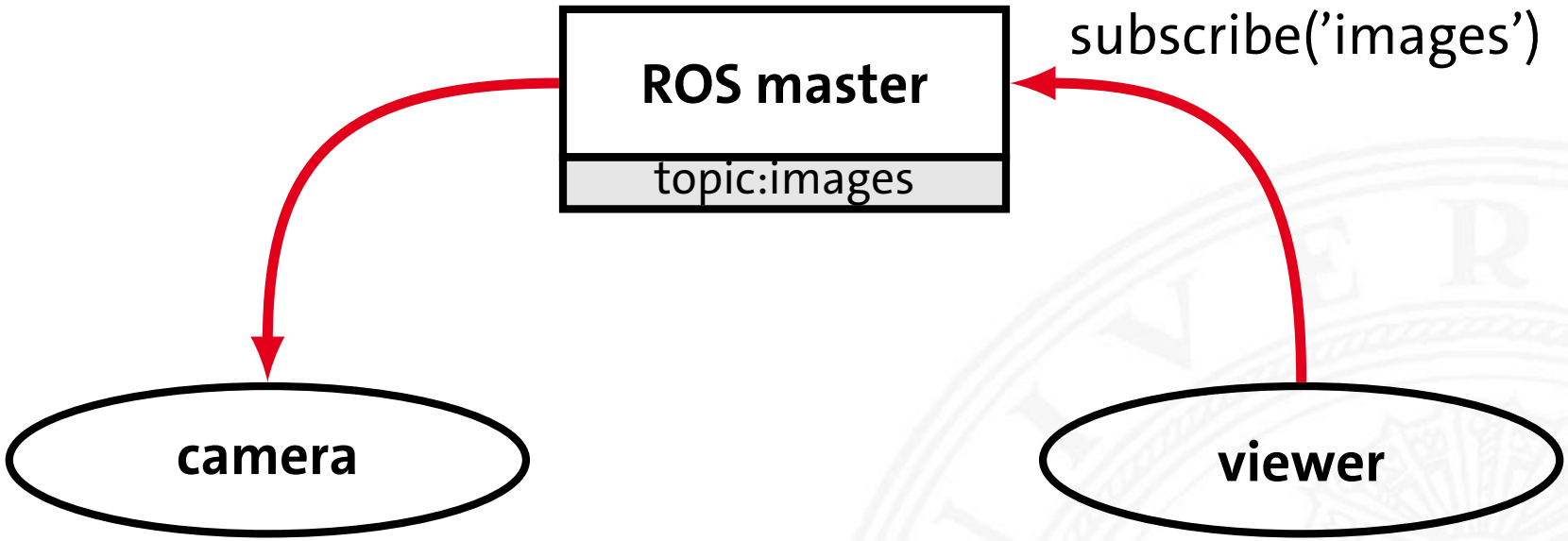
# Communication – Example





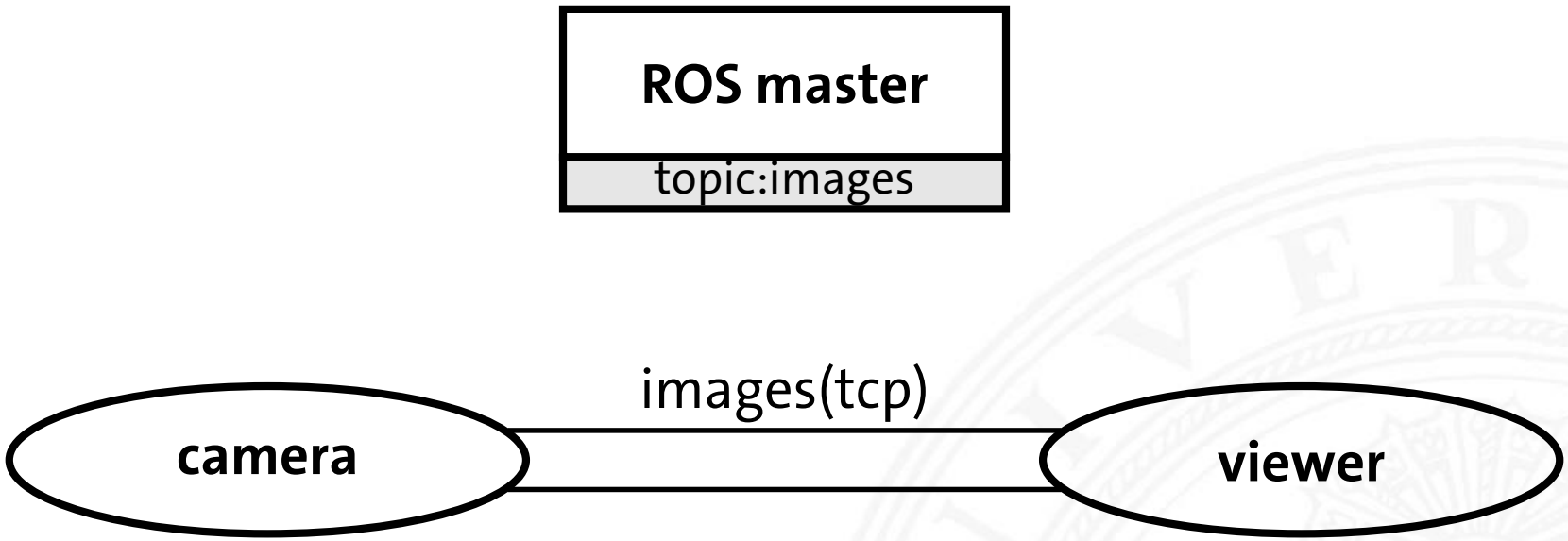
# Communication – Example





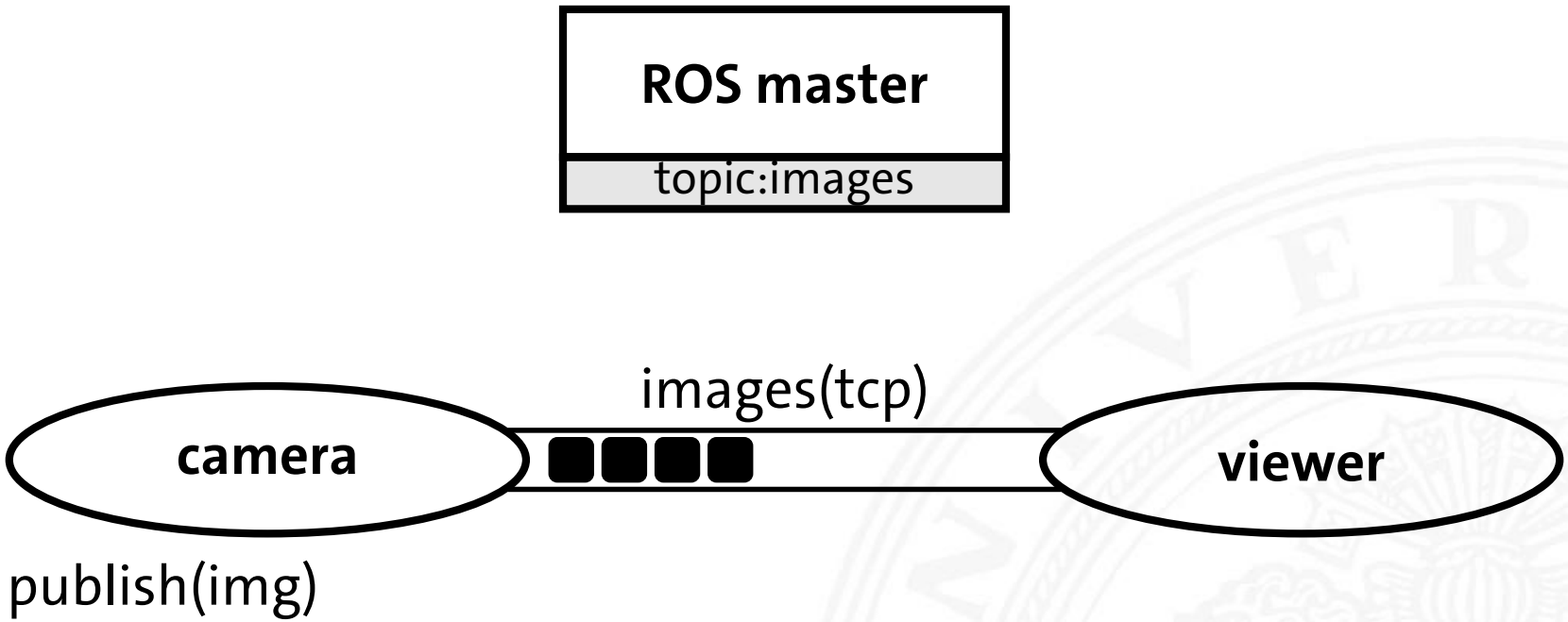
# Communication – Example





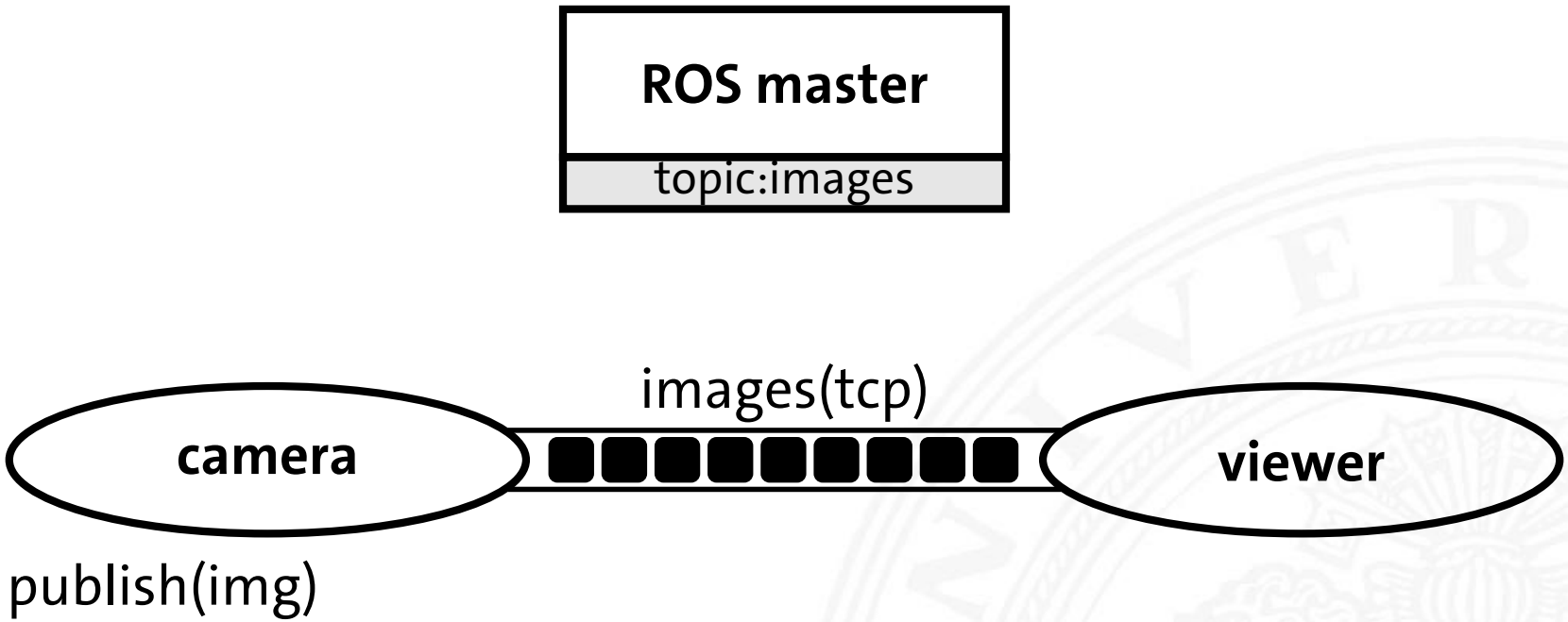
# Communication – Example





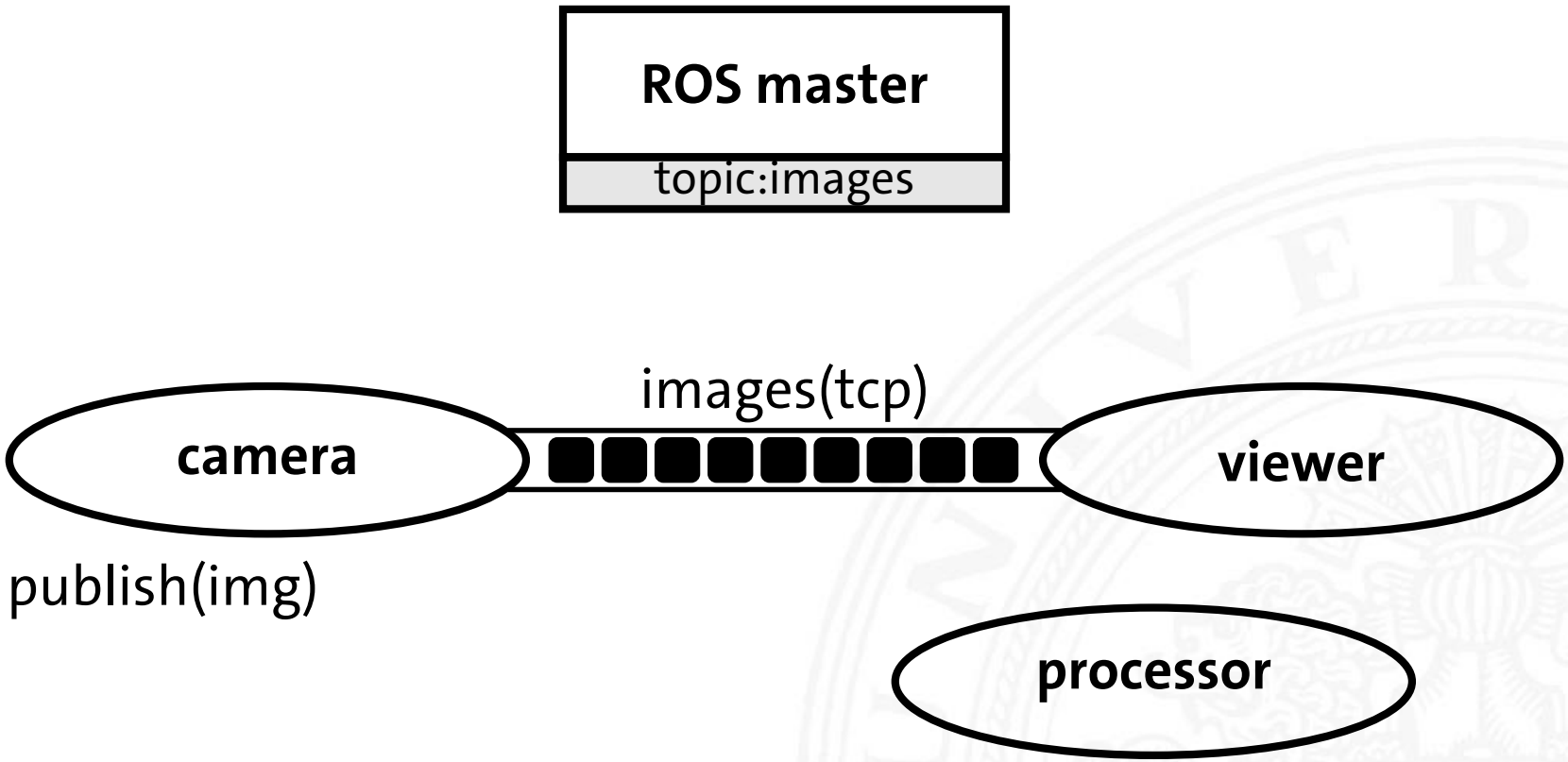
# Communication – Example





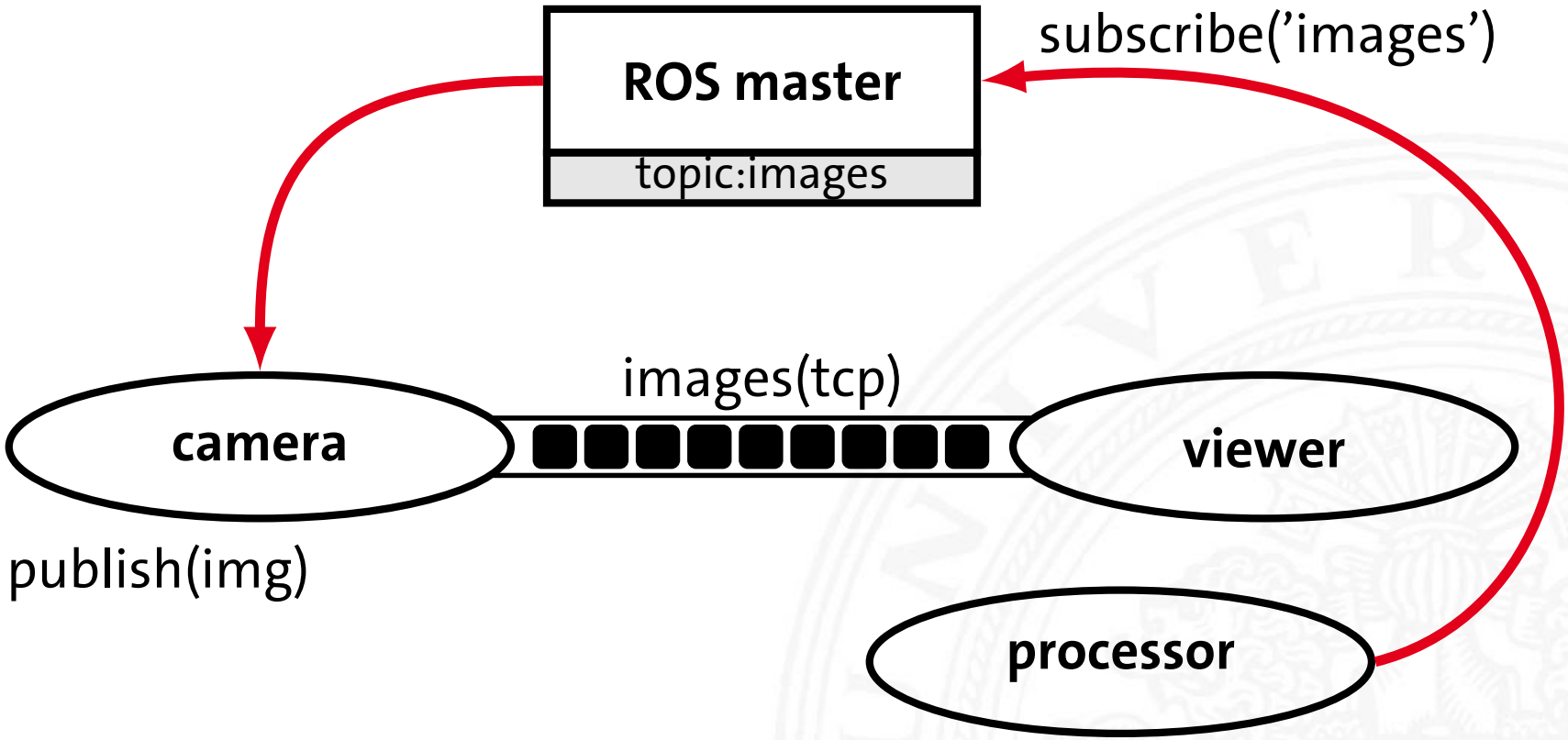
# Communication – Example





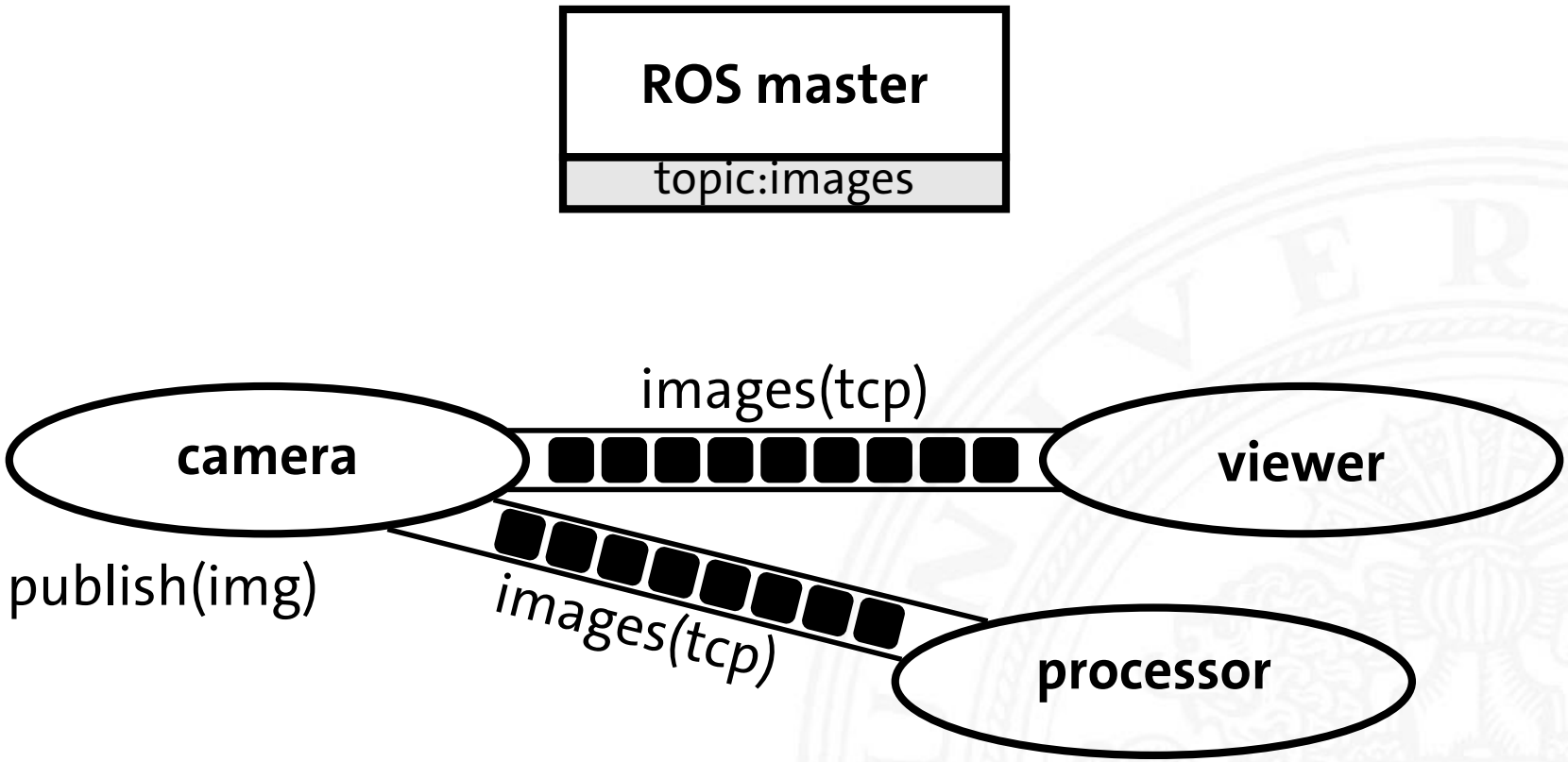
# Communication – Example





# 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

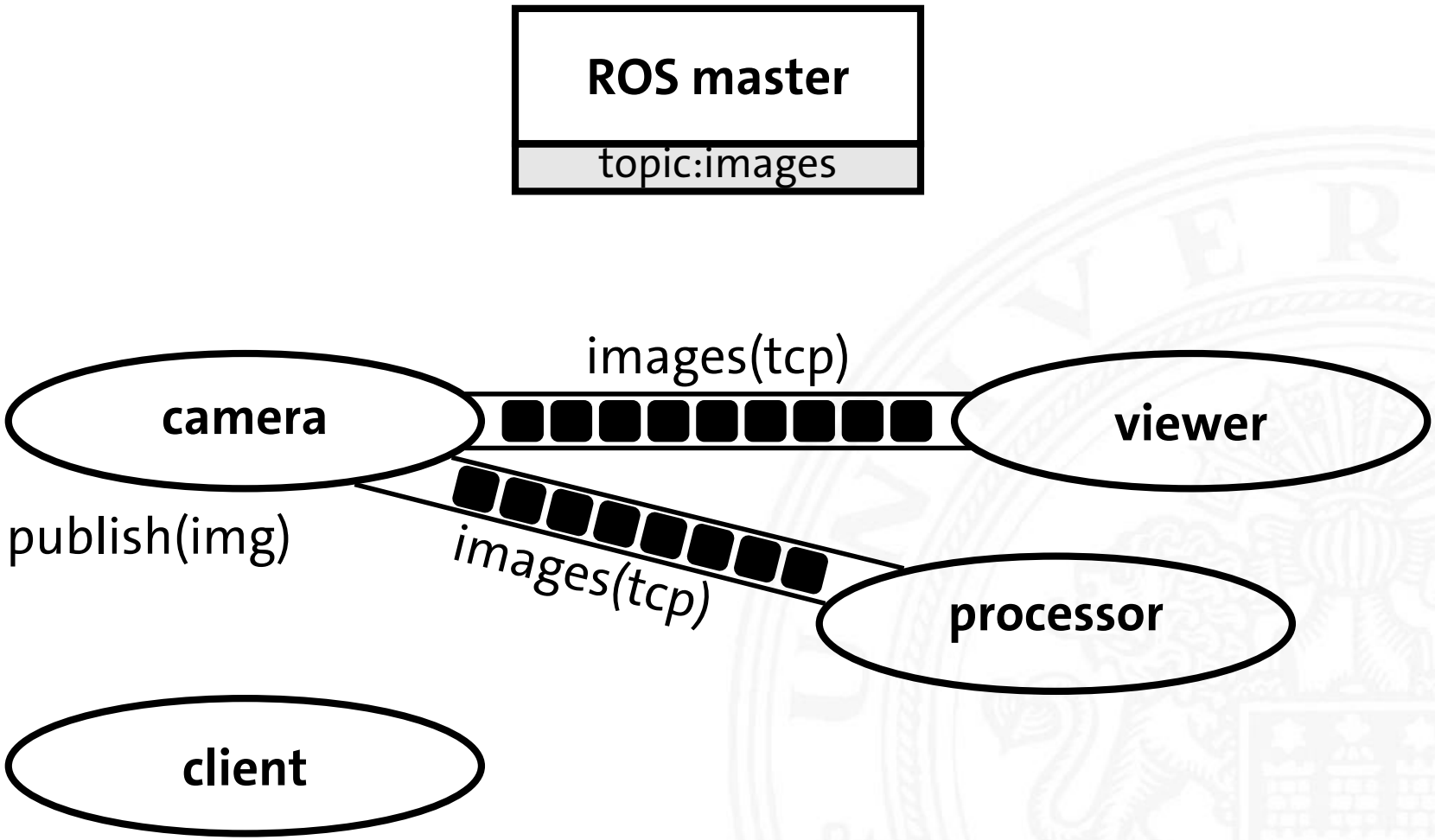


- 2 message types
  - request and response
- Synchronous protocol
  - client sends request
  - client waits for server
  - server replies

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

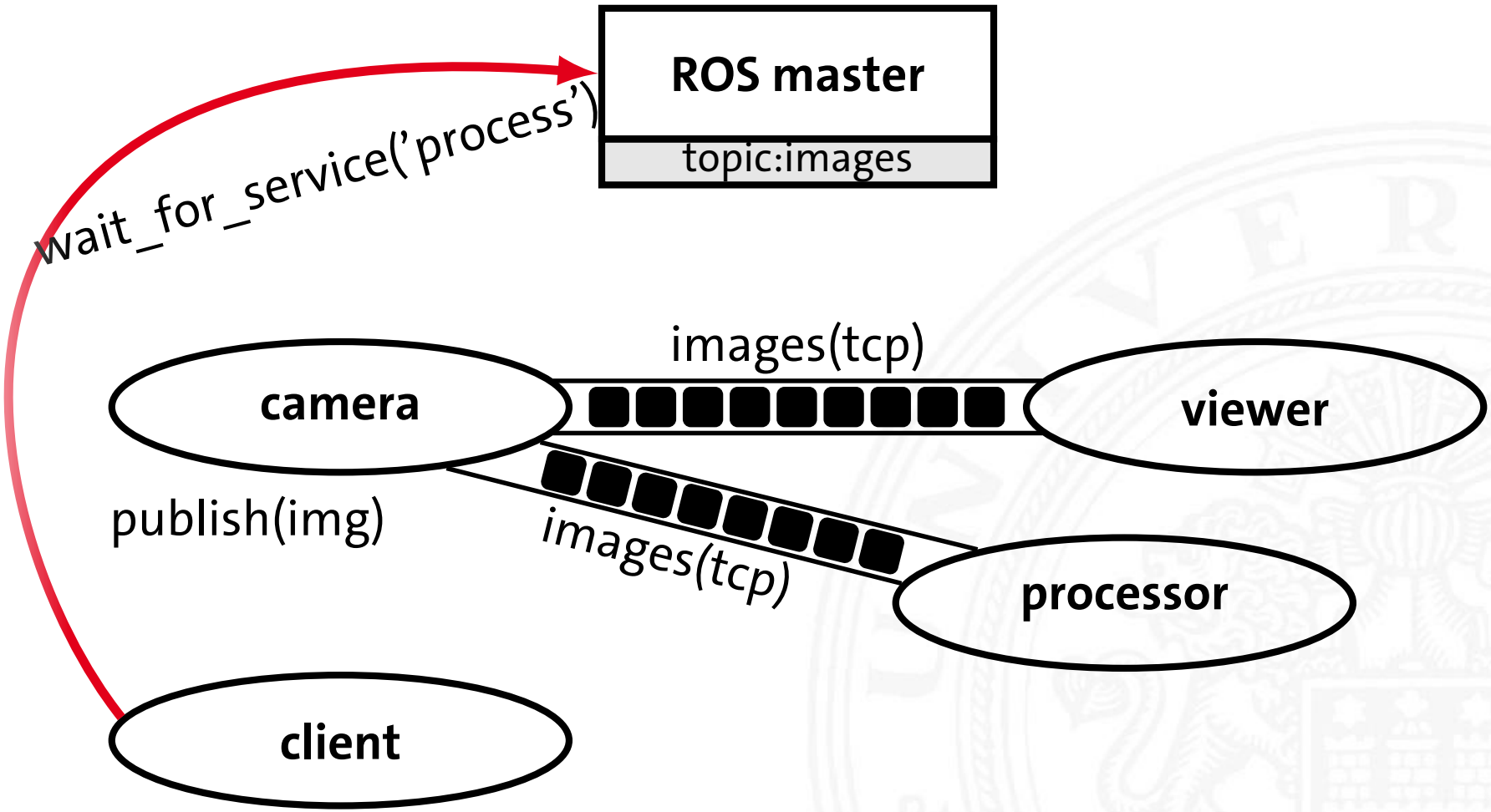
# Communication – Example





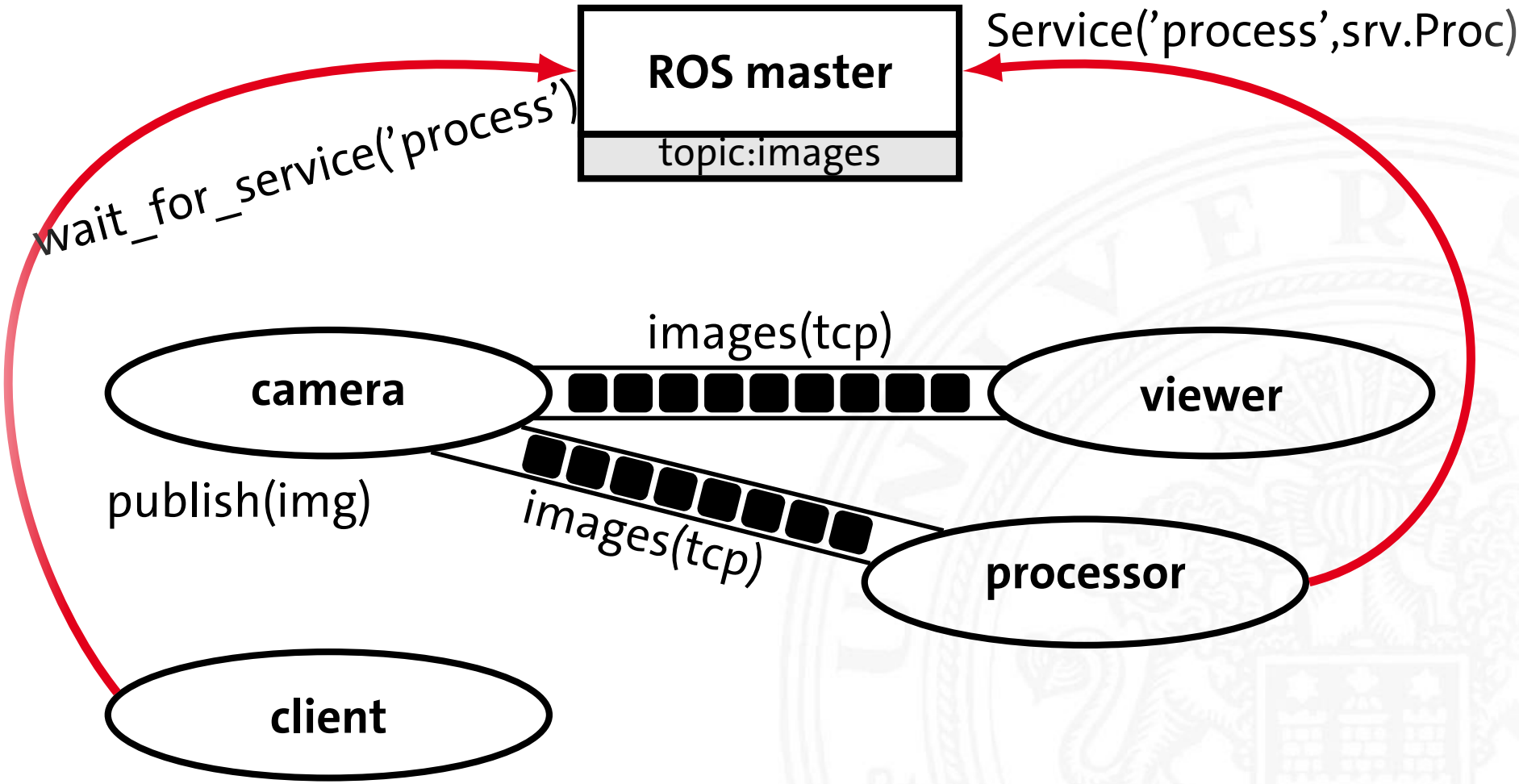
# Communication – Example





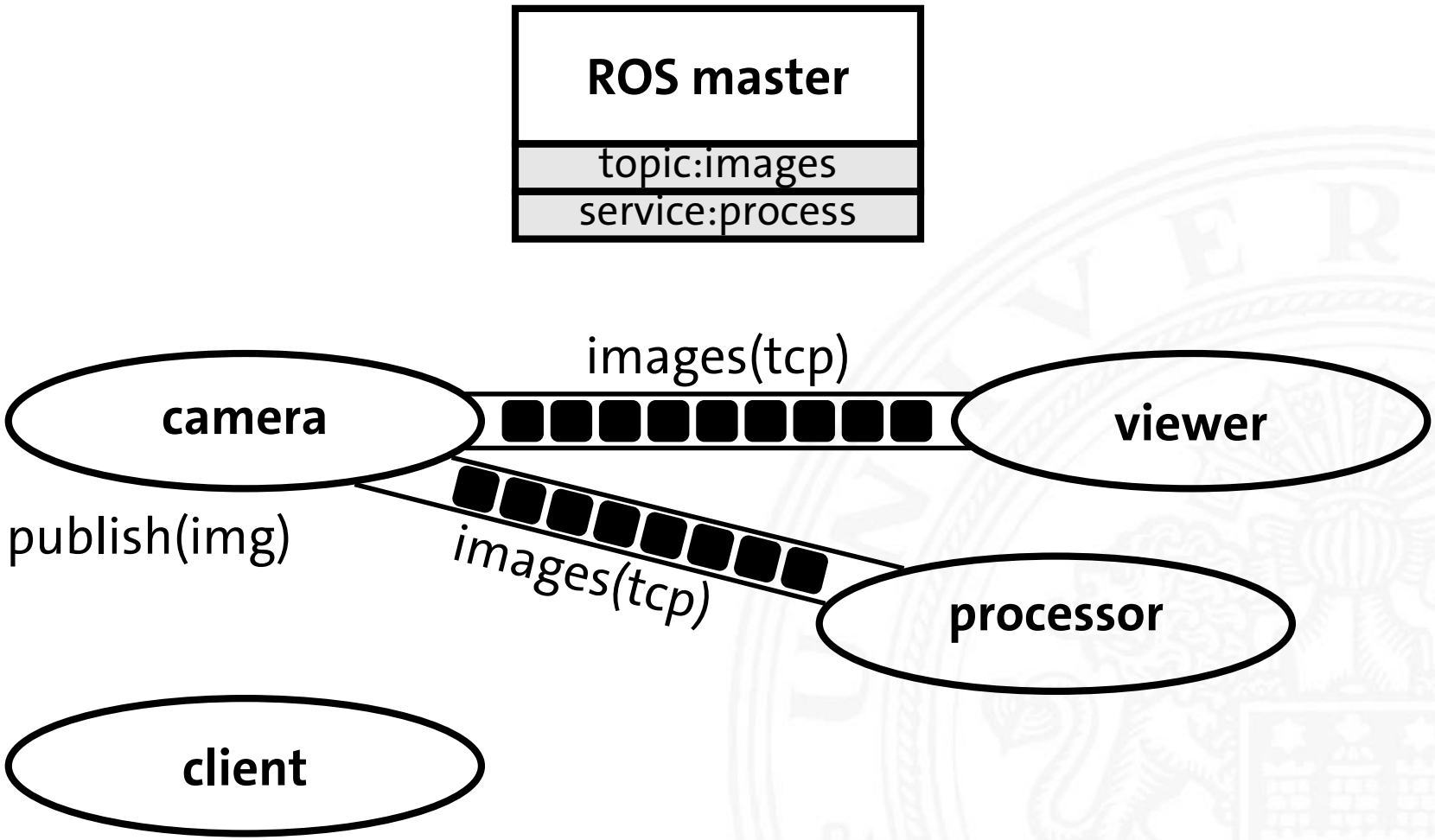
# Communication – Example





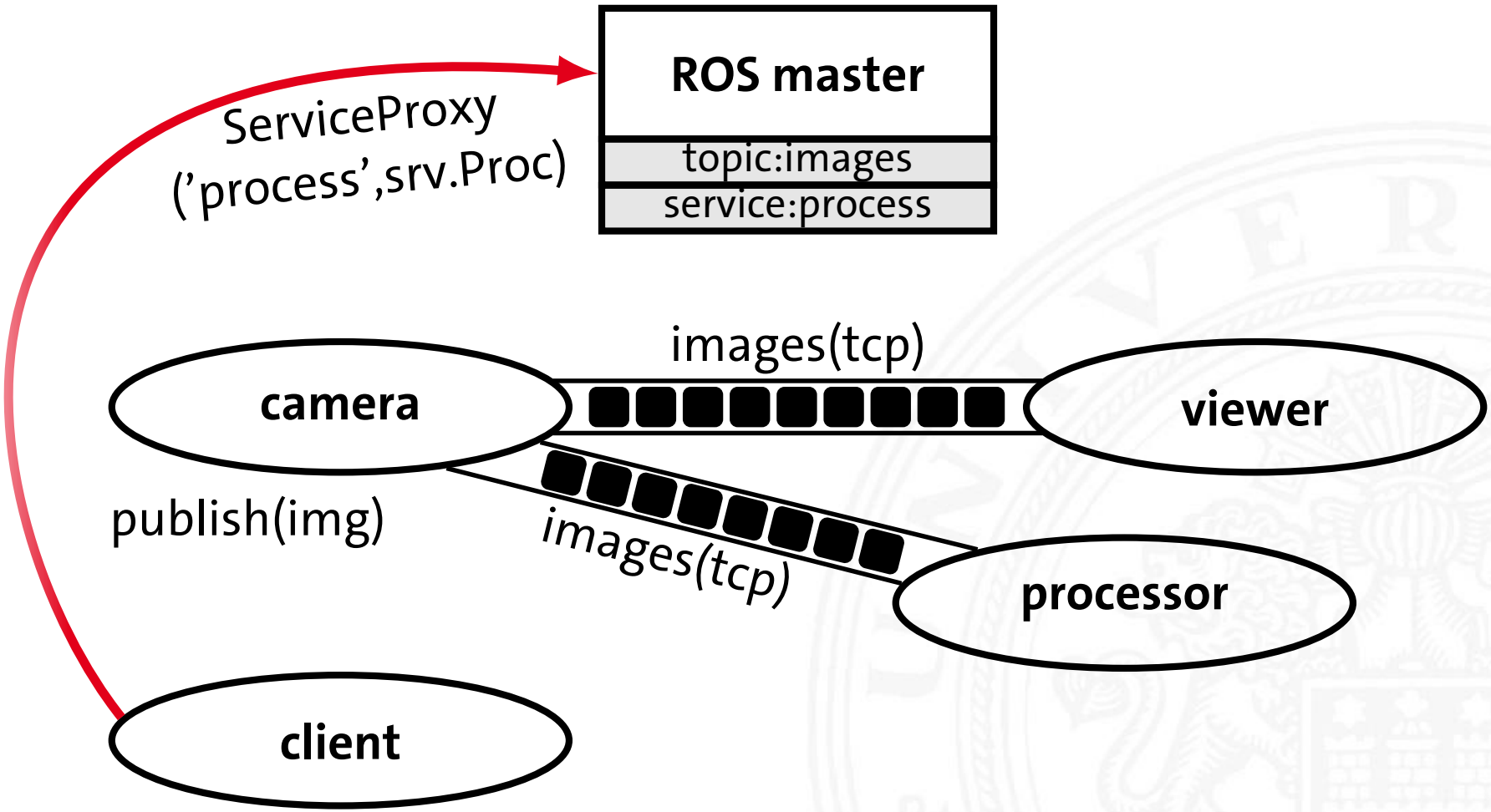
# Communication – Example





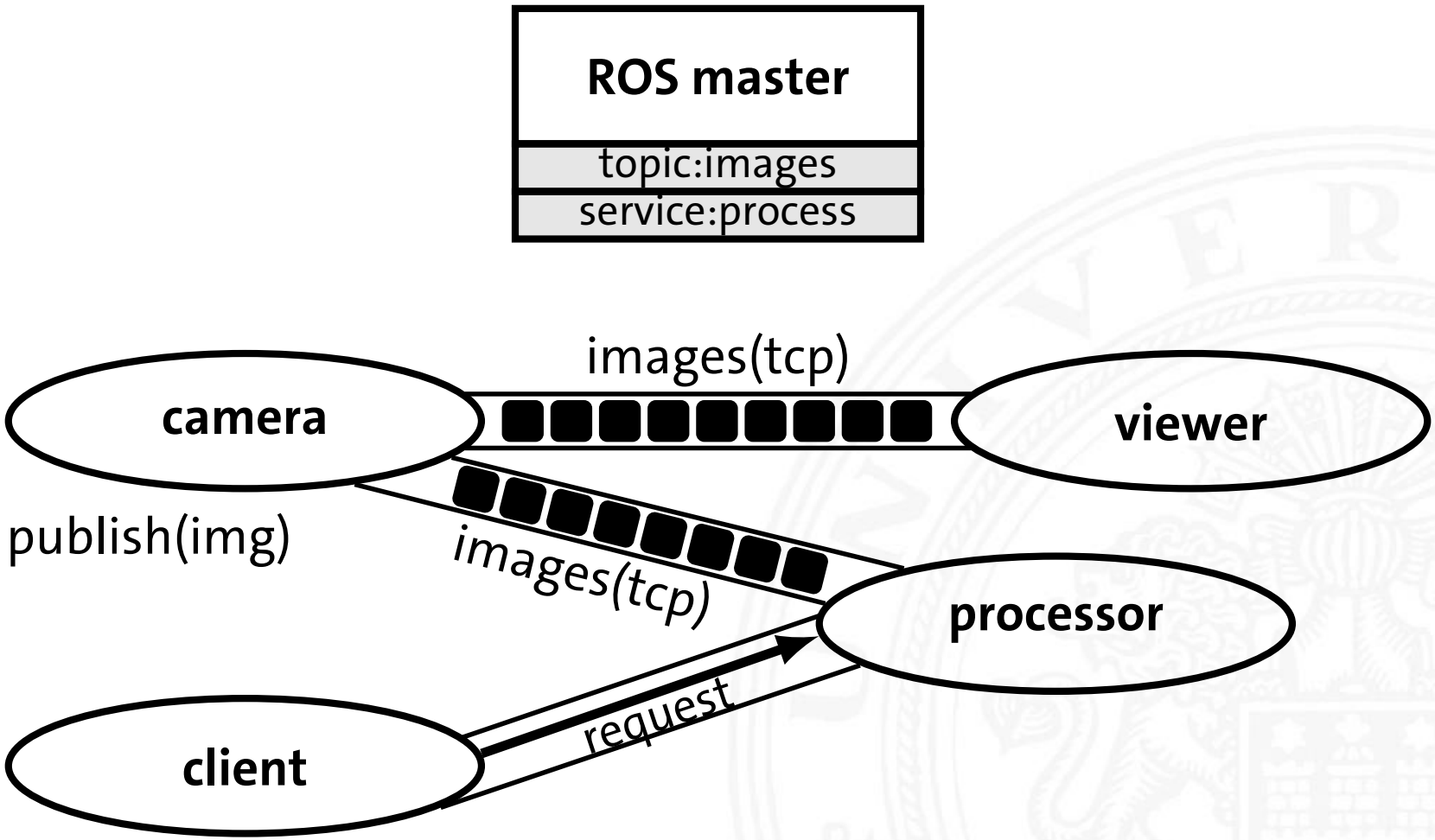
# Communication – Example





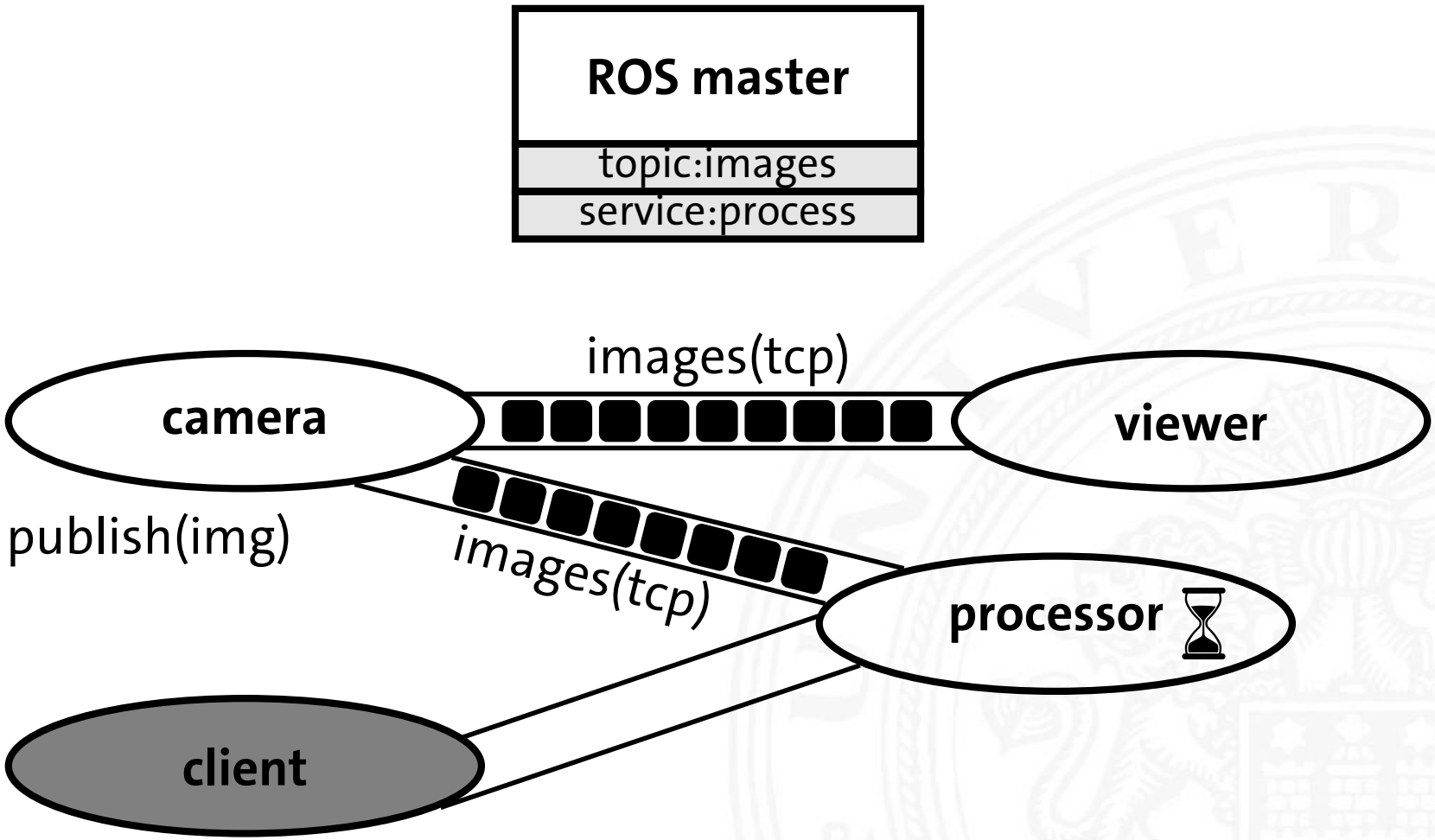
# Communication – Example





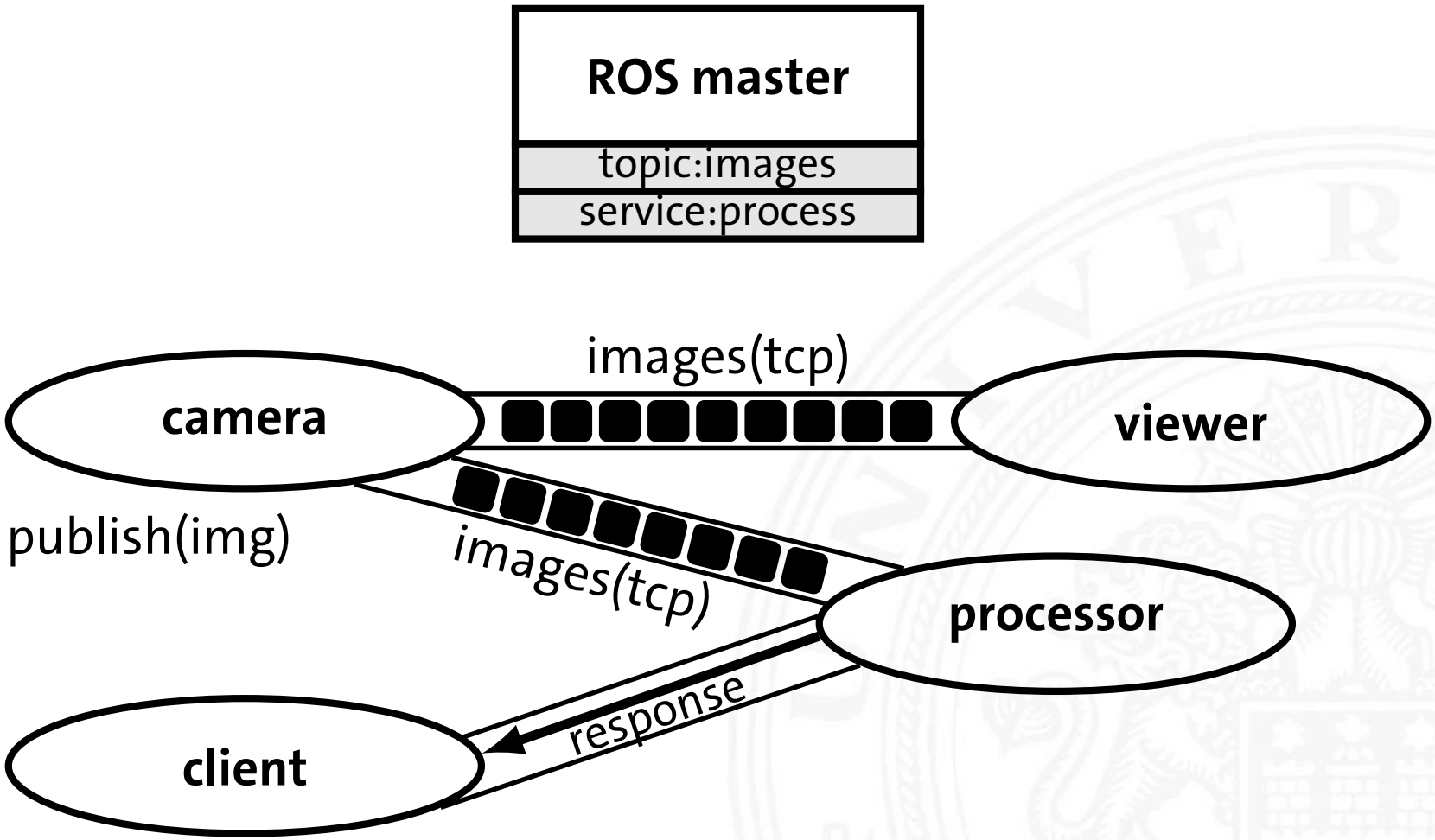
# Communication – Example





# 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

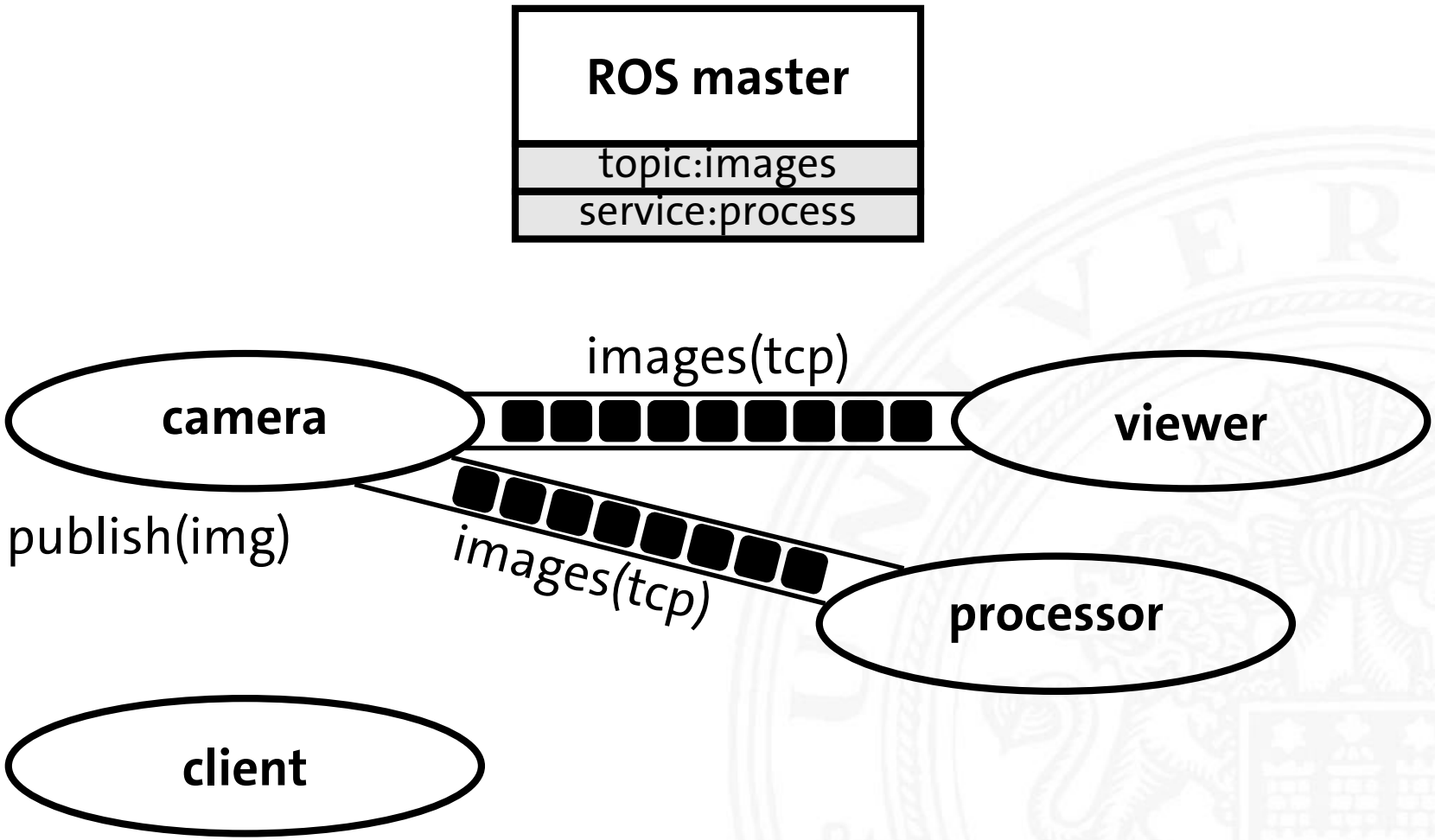


- 3 message types
  - goal and result
  - optional feedback
- Asynchronous protocol
  - client sends goal
  - server may respond with feedback
  - server delivers result
- Interruptible

```
# 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
```

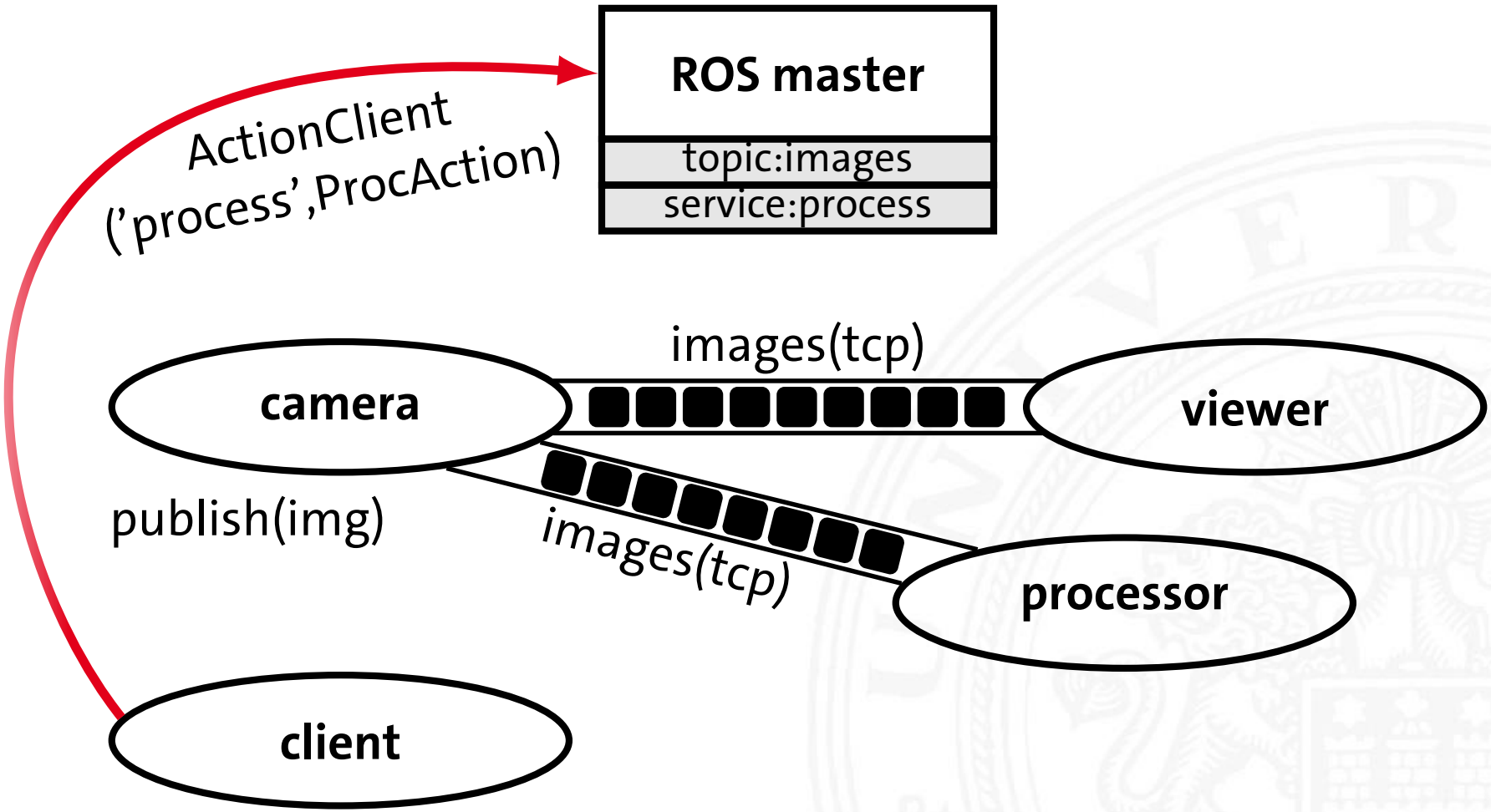
# Communication – Example





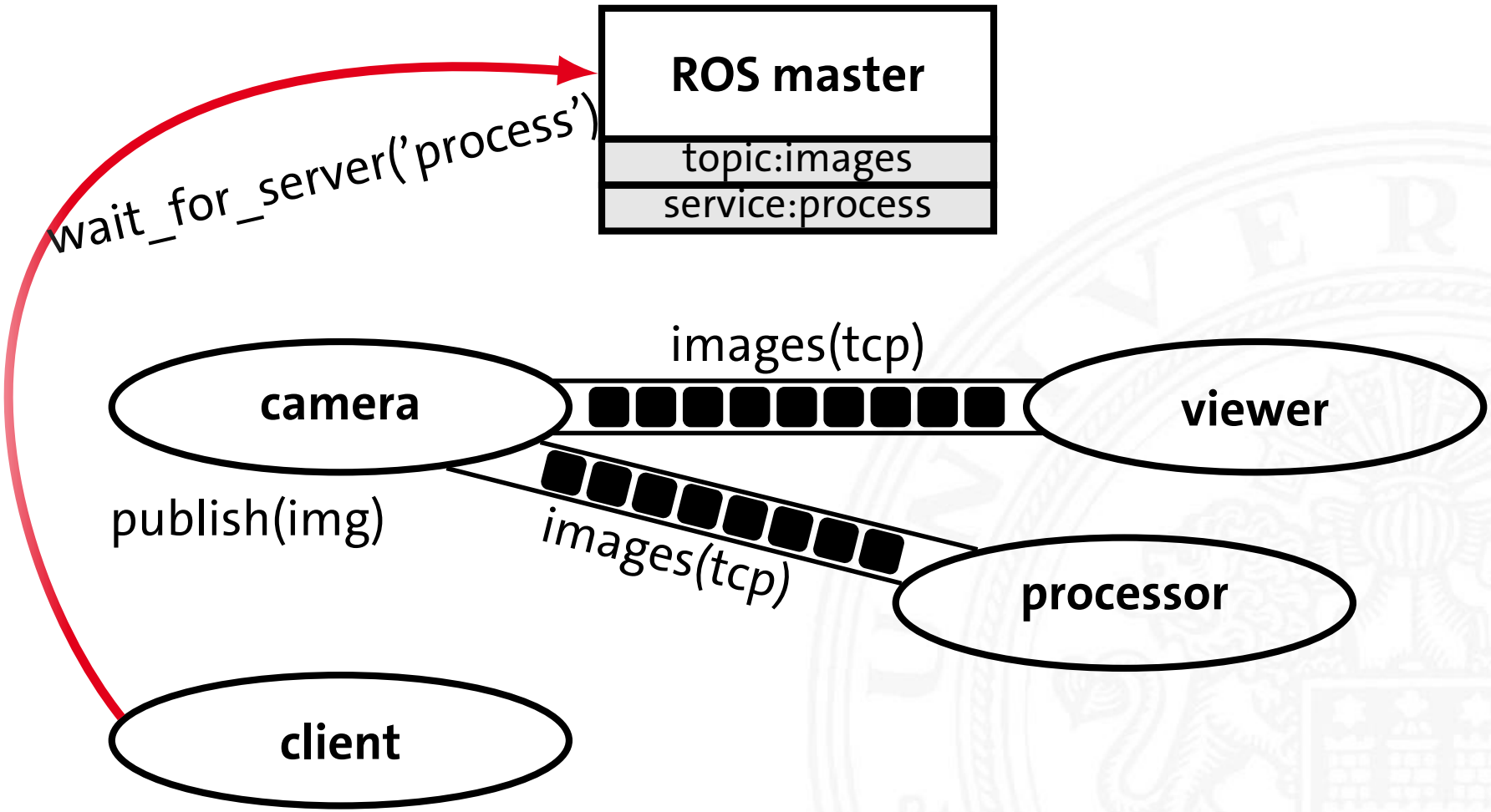
# Communication – Example





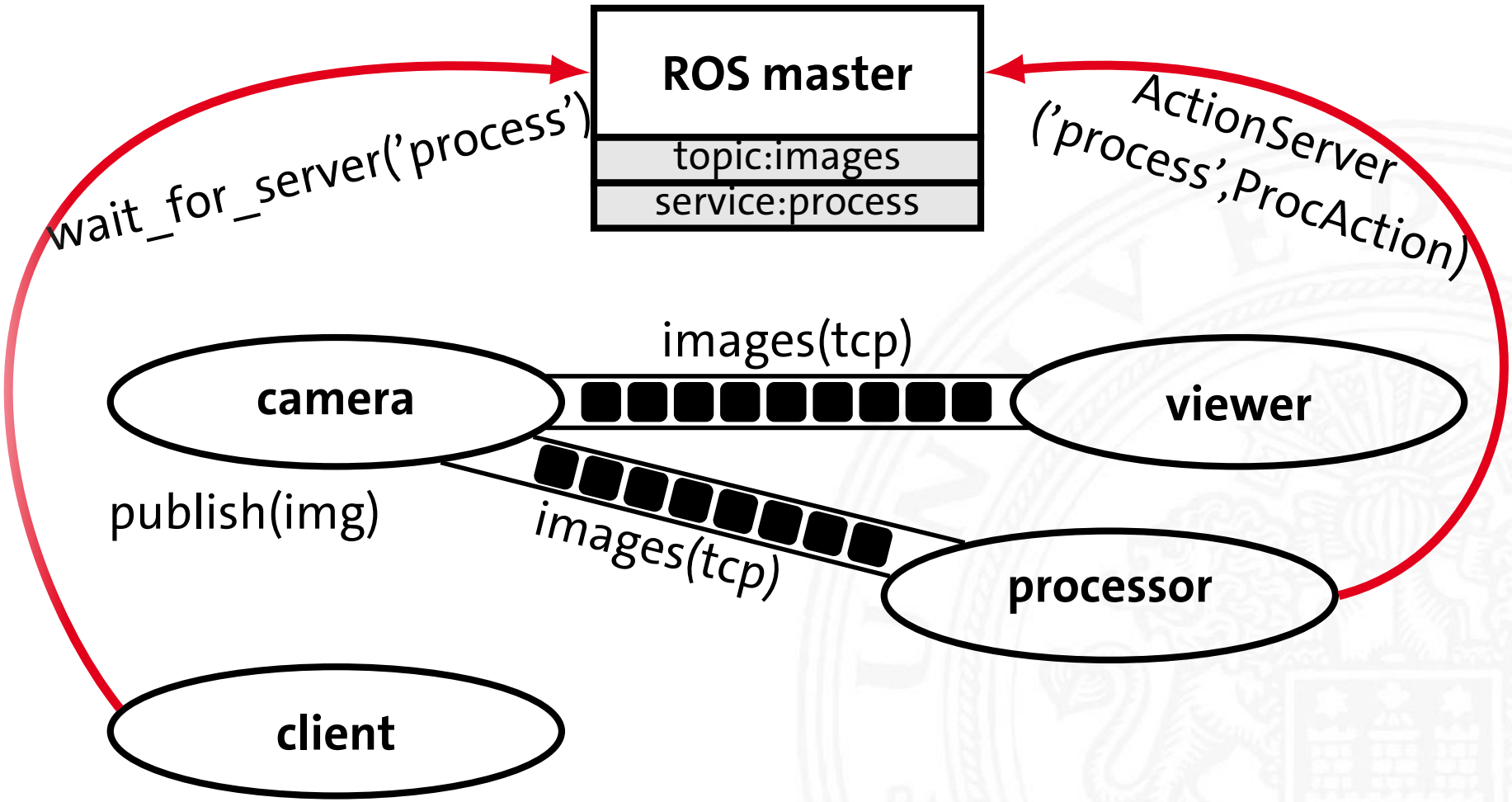
# Communication – Example





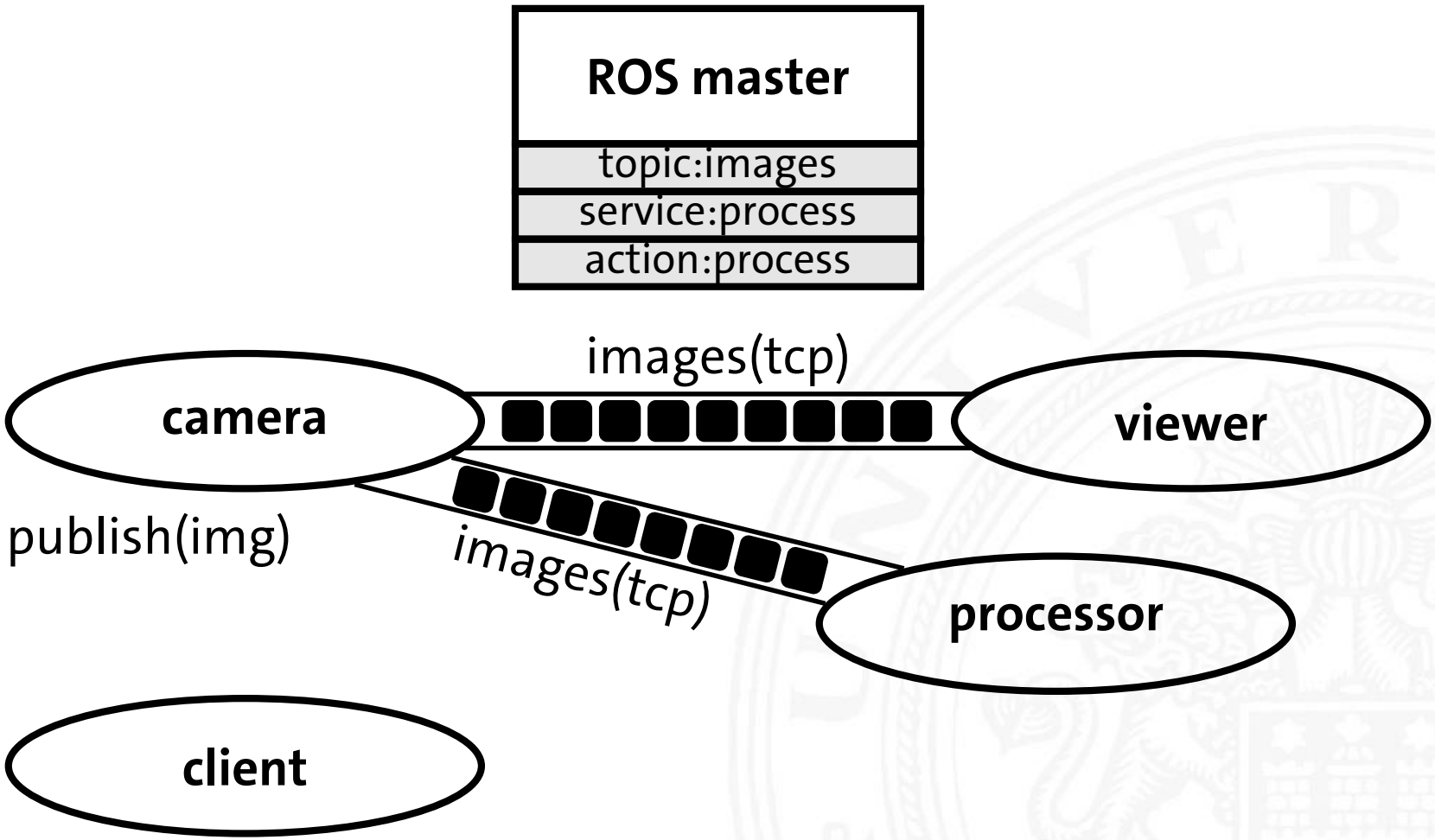
# Communication – Example





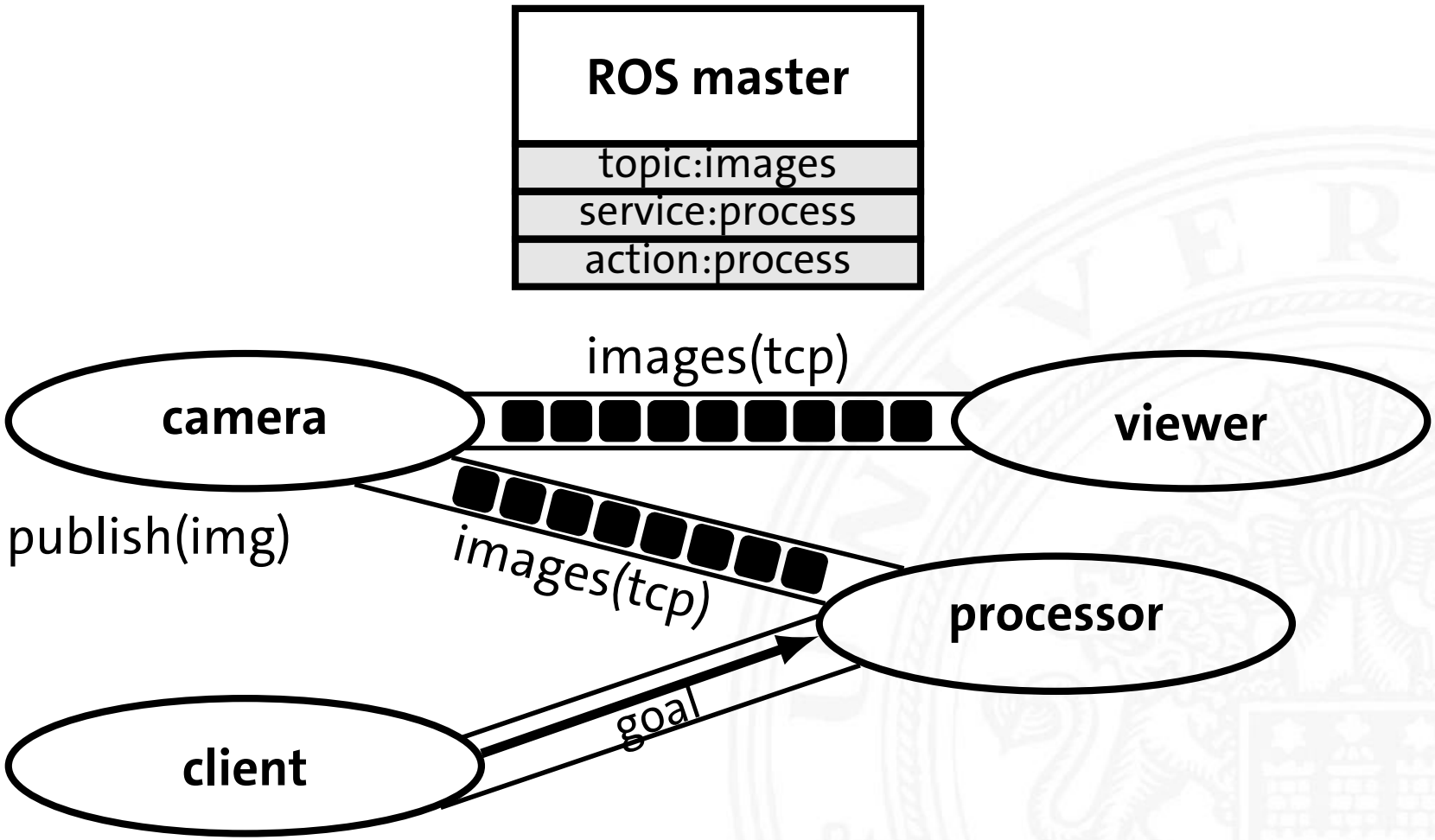
# Communication – Example





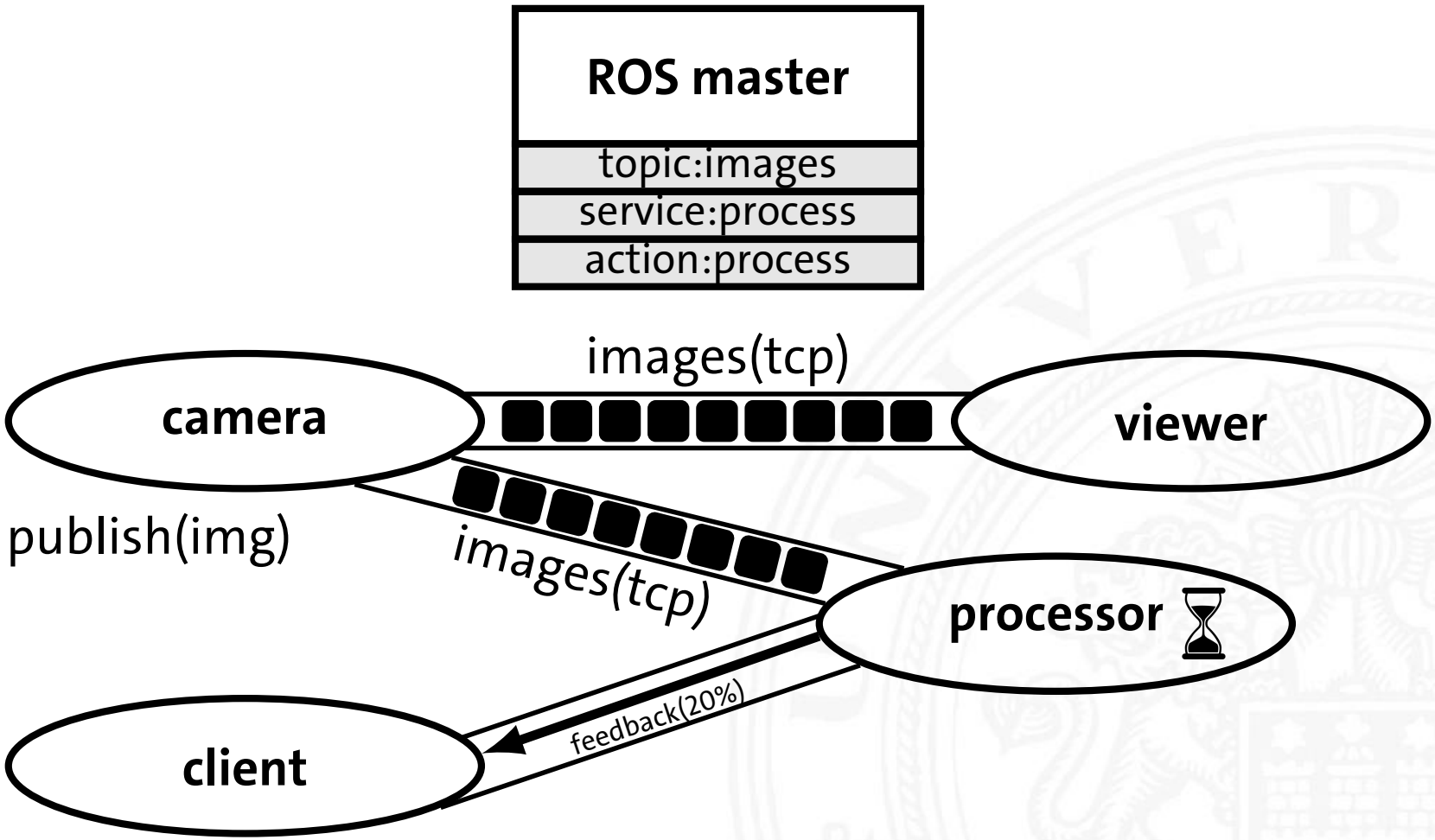
# Communication – Example





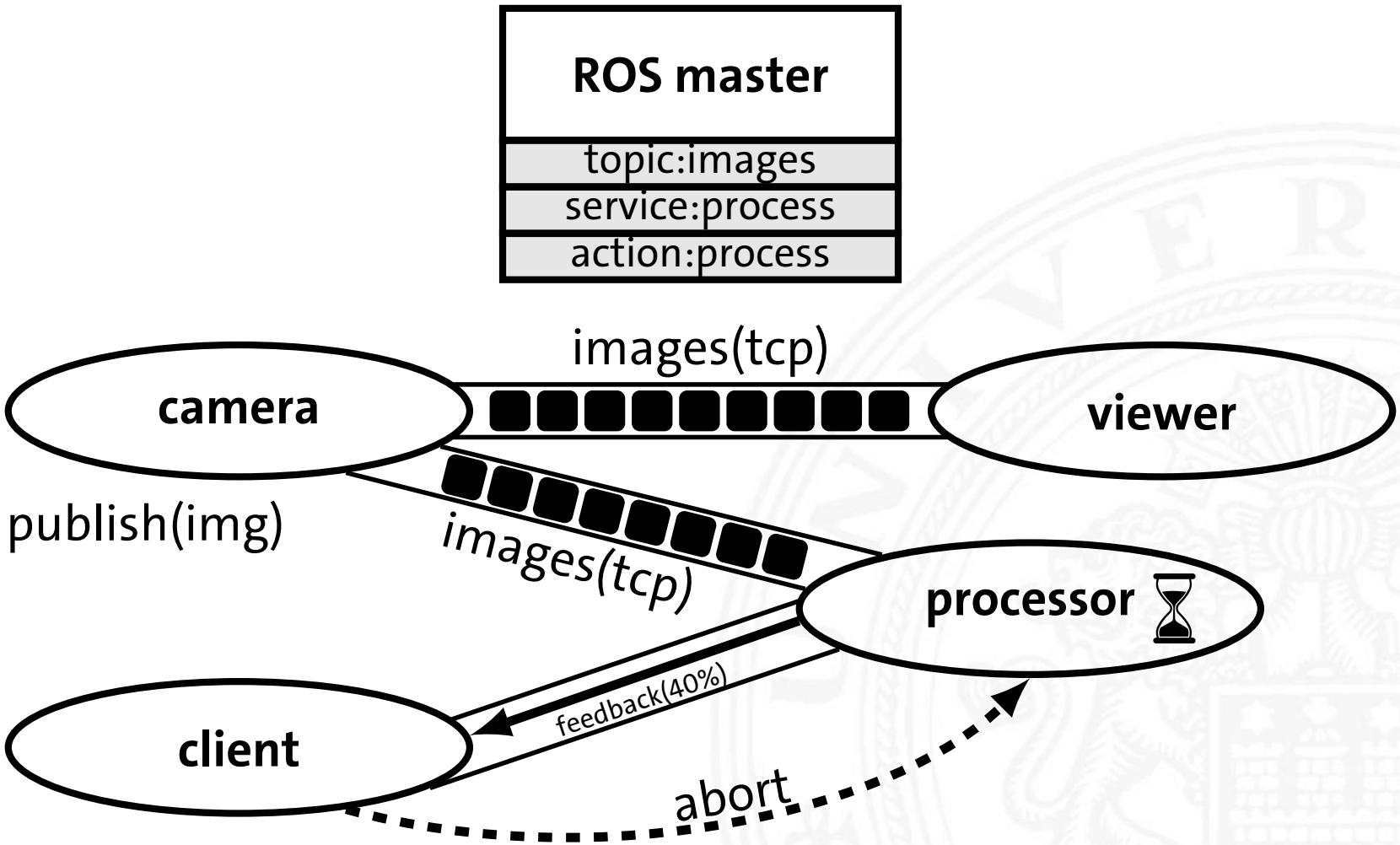
# Communication – Example





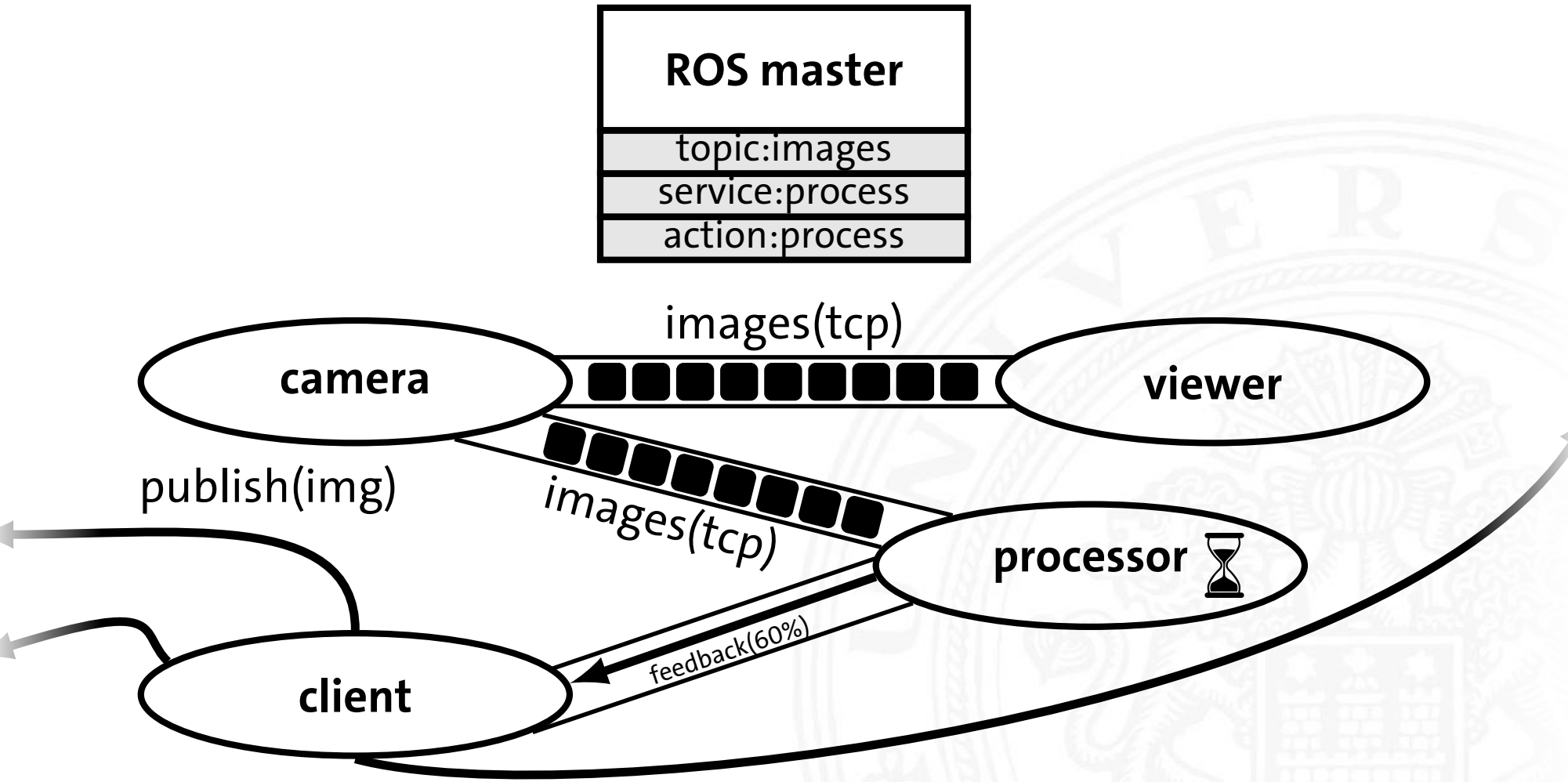
# Communication – Example





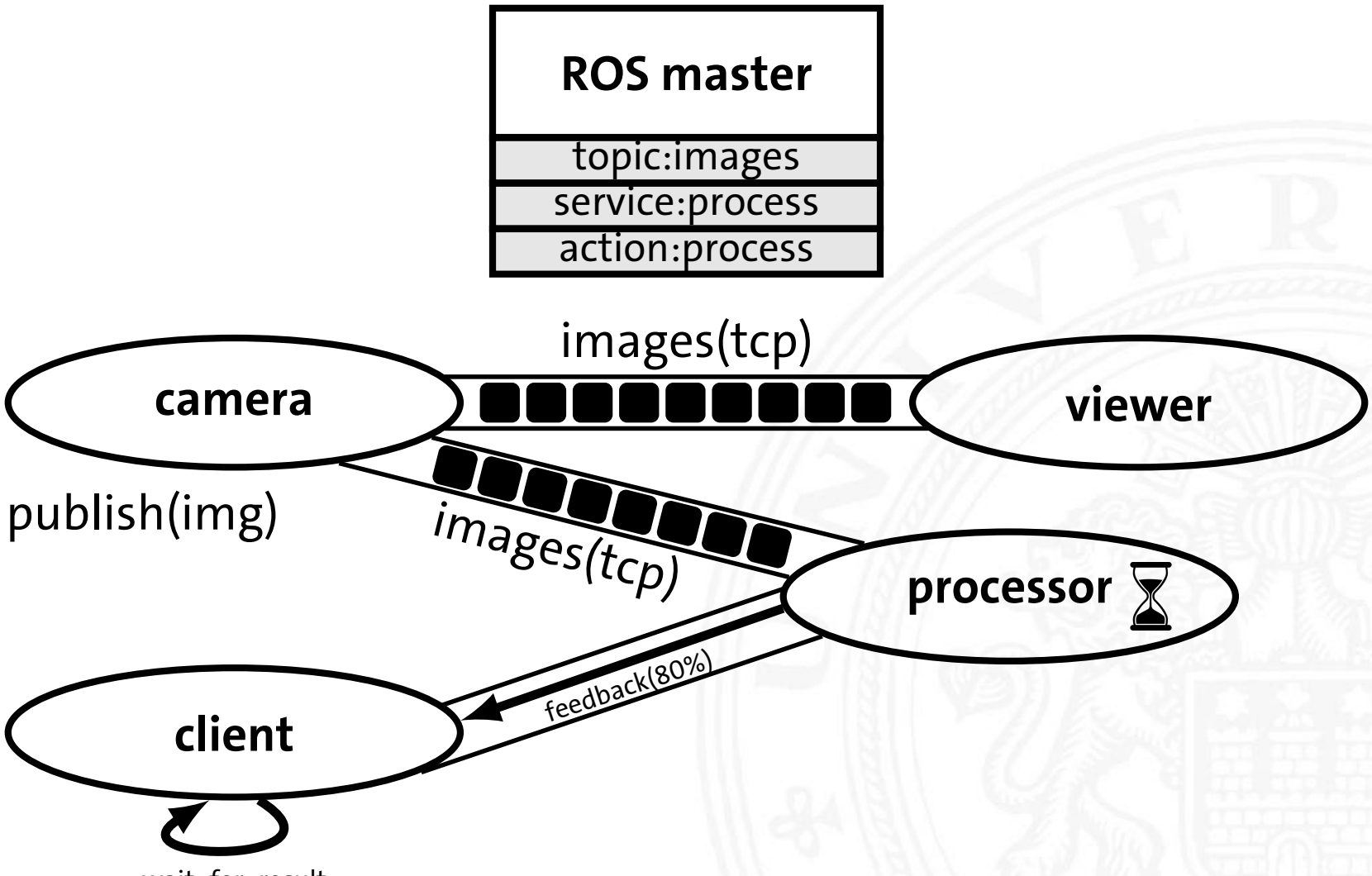
# Communication – Example





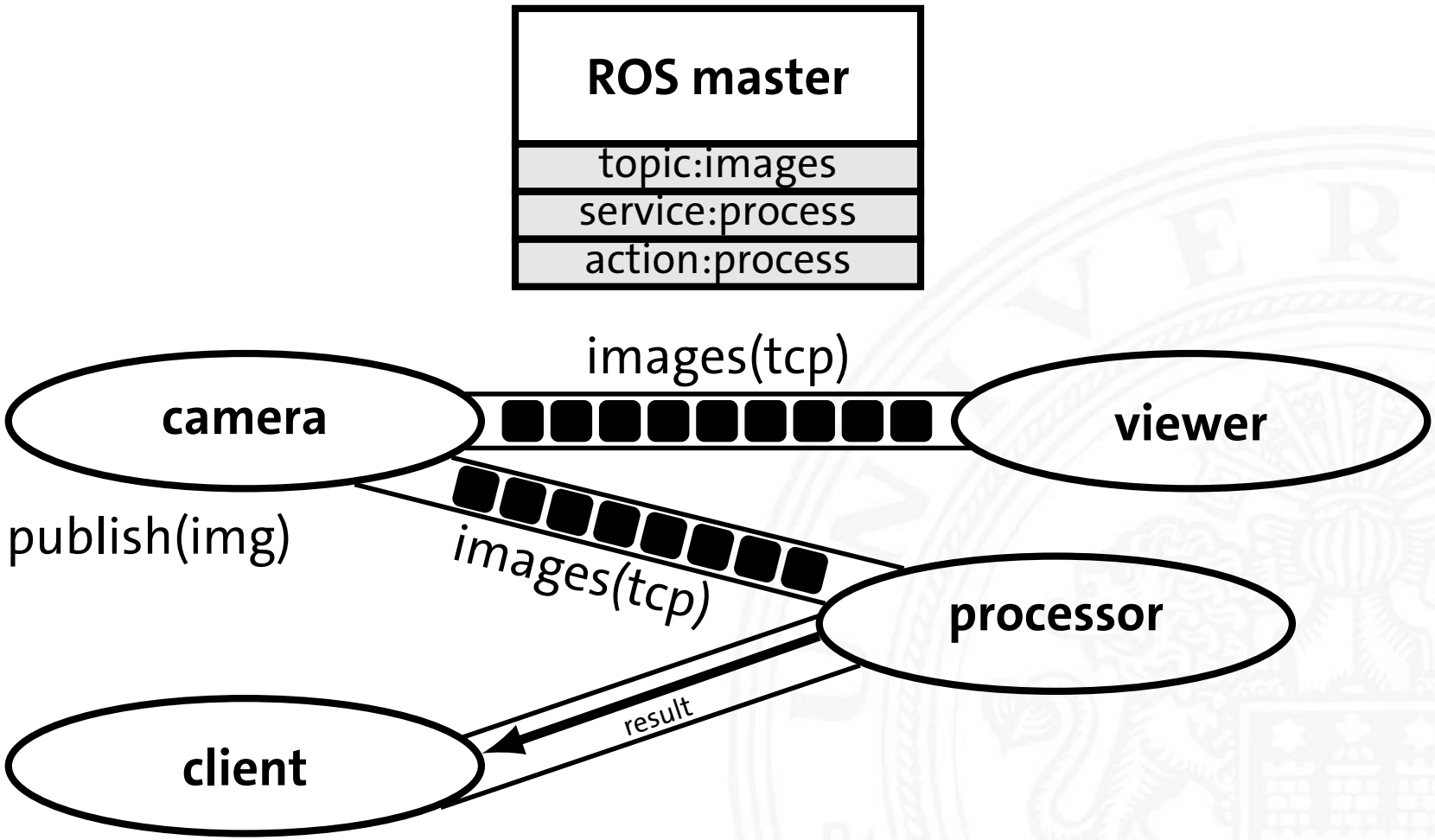
# Communication – Example





# Communication – Example

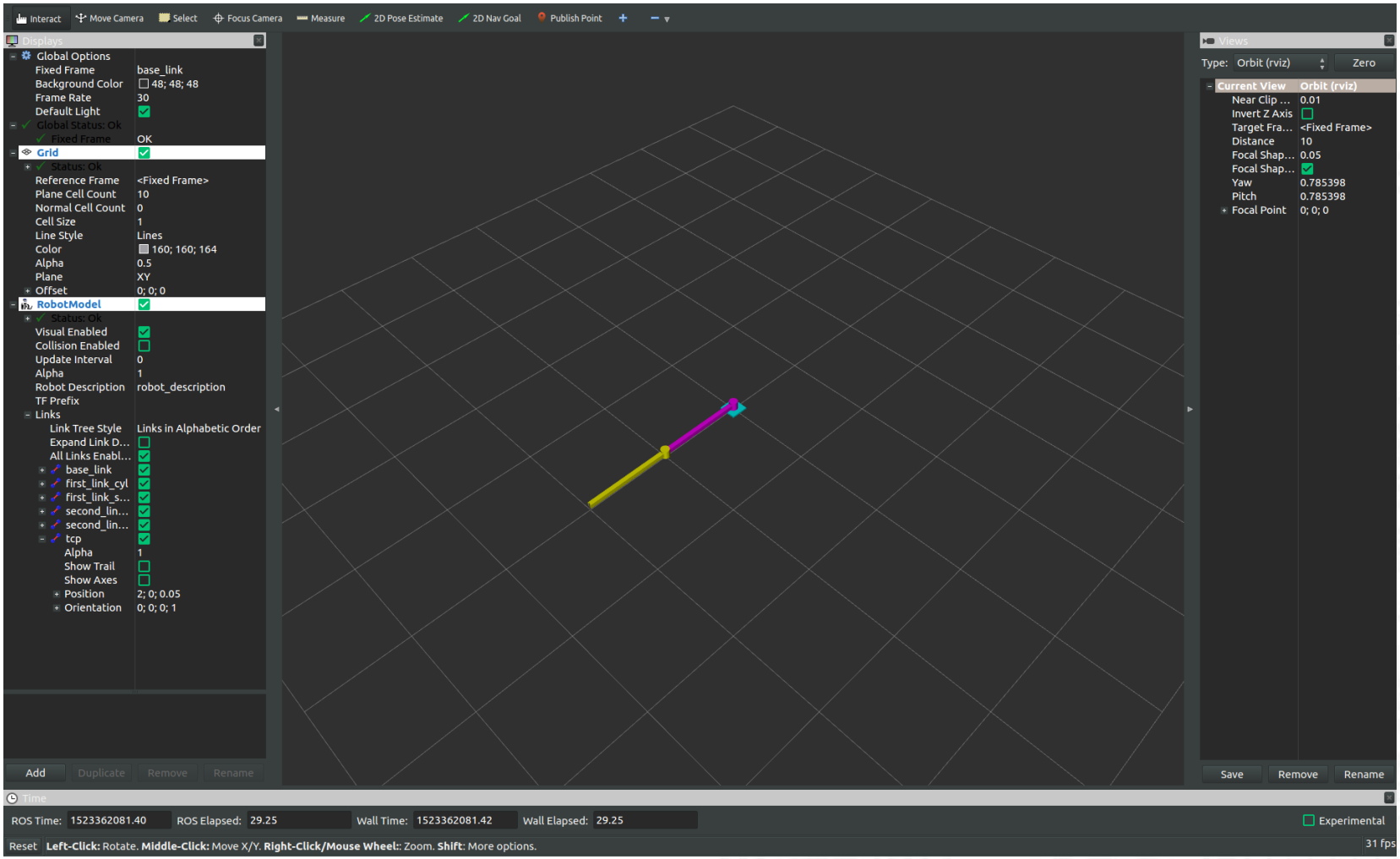




# Outline

# Kinematics Tool

- Visualization for kinematics of robot models



# Properties



- 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