Introduction to ROS/ROS2

Shang-Ching Liu

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

October 14, 2025

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 ...

From ROS to ROS2

- ROS 1 pioneered modular robotics frameworks
  - Global master dependency
  - No native real-time guarantees
  - Difficult multi-robot networking
  - Static parameter configuration
- ROS 2 redesign built on DDS (Data Distribution Service) for reliability and scalability

Current State (ROS/ROS2)

- Multiple versions actively used
  - may not be compatible to each other
  - may not provide same libraries
- Linux (Ubuntu!) $\Rightarrow$ Windows, Mac, Linux
- 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/ROS2 System

- ROS nodes
  - sensors
  - actuators
  - logic
- ROS core $\Rightarrow$ None
- 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

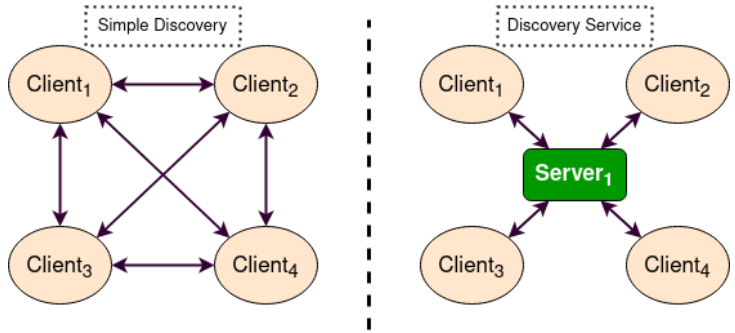
From `ROS Core` to Distributed Discovery in ROS 2

ROS 1: Centralized Control

- Uses `roscore` (Master) for:
  - Node registration and name resolution
  - Topic/service lookup via XML-RPC

ROS 2: Distributed Discovery

- No central master — built on **DDS (Data Distribution Service)**
- Each node is a **DDS Participant** handling its own discovery



DDS Participants auto-discover publishers/subscribers within a domain.

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

```
$ rosmsg show -r robot_msgs/Quaternion
⇒ $ ros2 interface show robot_msgs/Quaternion
# xyz - vector rotation axis, w - scalar term (cos(ang/2))
float64 x
float64 y
float64 z
float64 w
```

Messages

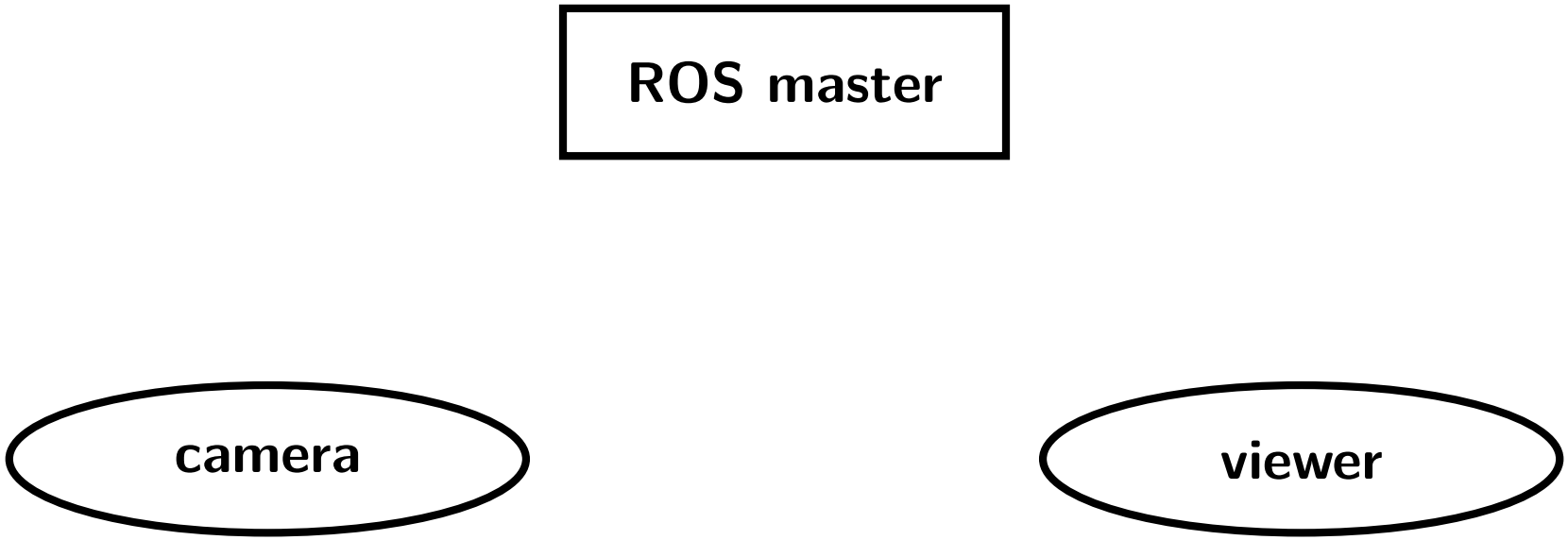
- Fundamental communication concept
- Description of data set
- Data types
  - ROS
  - general
- Header
  - time stamp
  - identifier

```
$ rosmsg show -r robot_msgs/Quaternion
⇒ $ ros2 interface show robot_msgs/Quaternion
# xyz - vector rotation axis, w - scalar term (cos(ang/2))
float64 x
float64 y
float64 z
float64 w
```

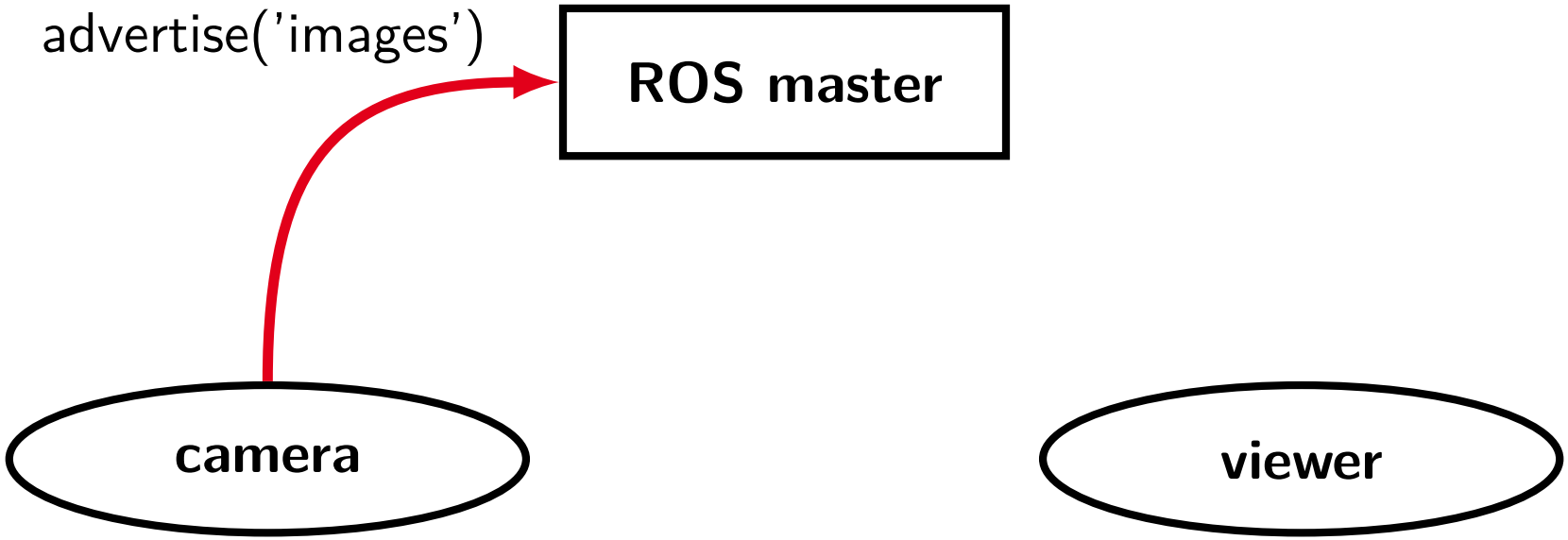
Topics

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

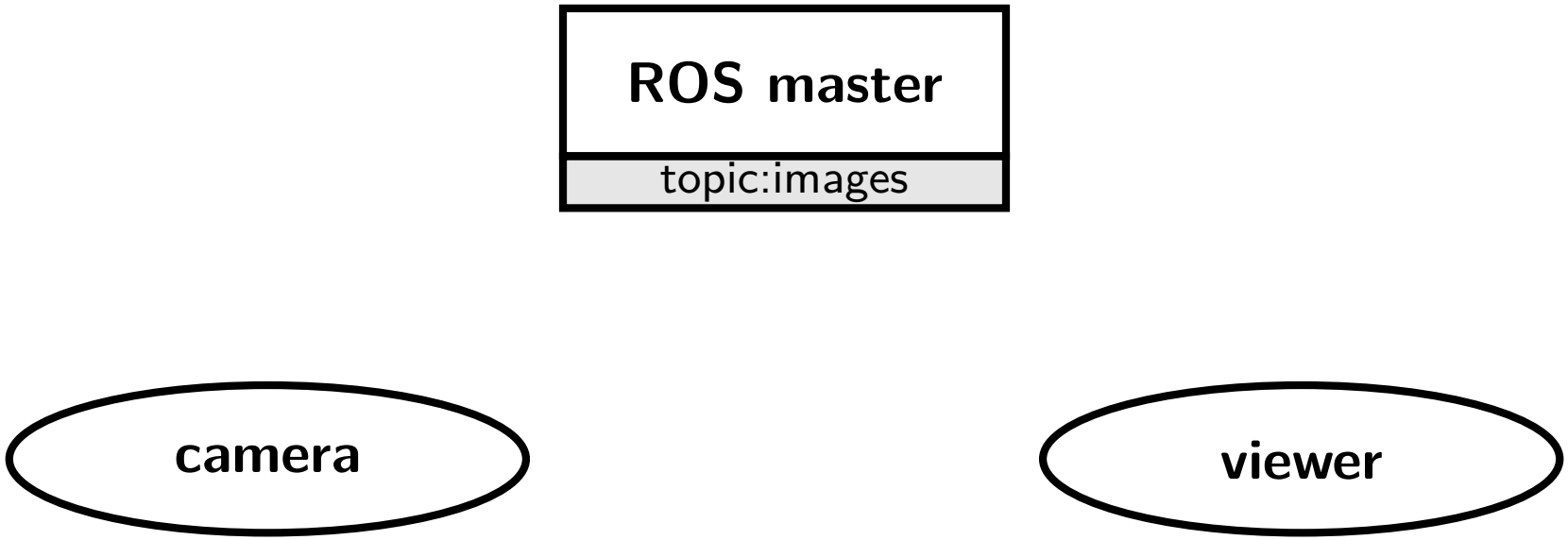
Communication - Example



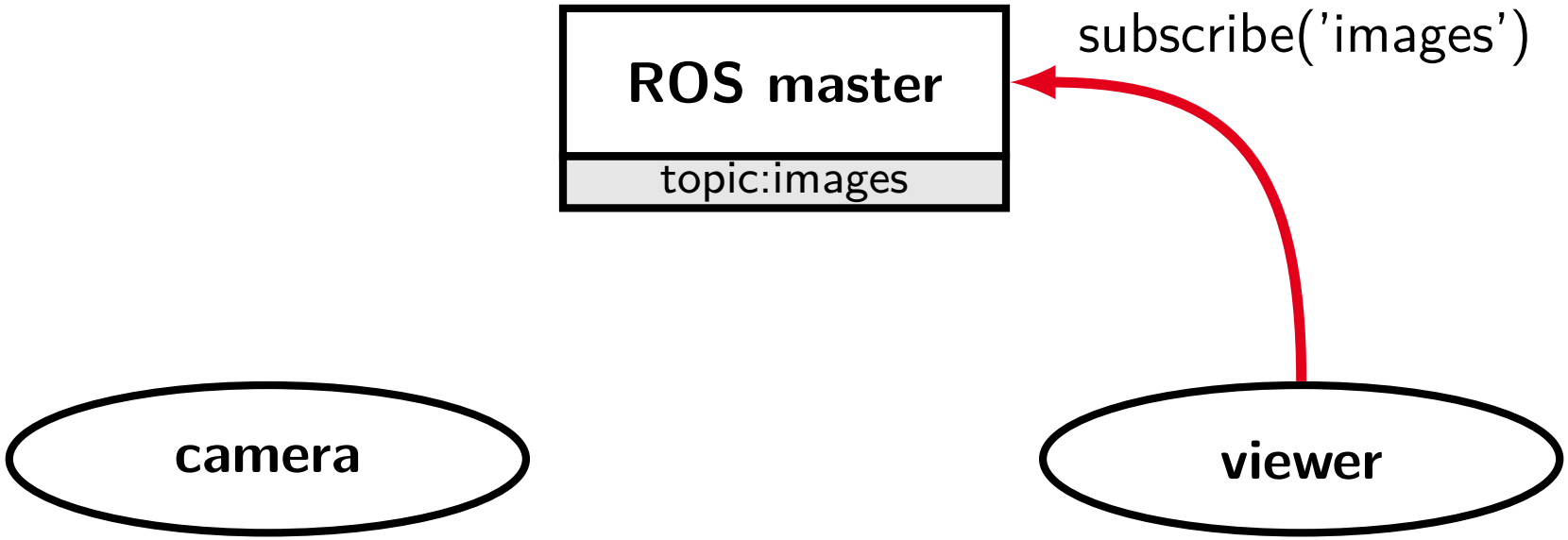
Communication - Example



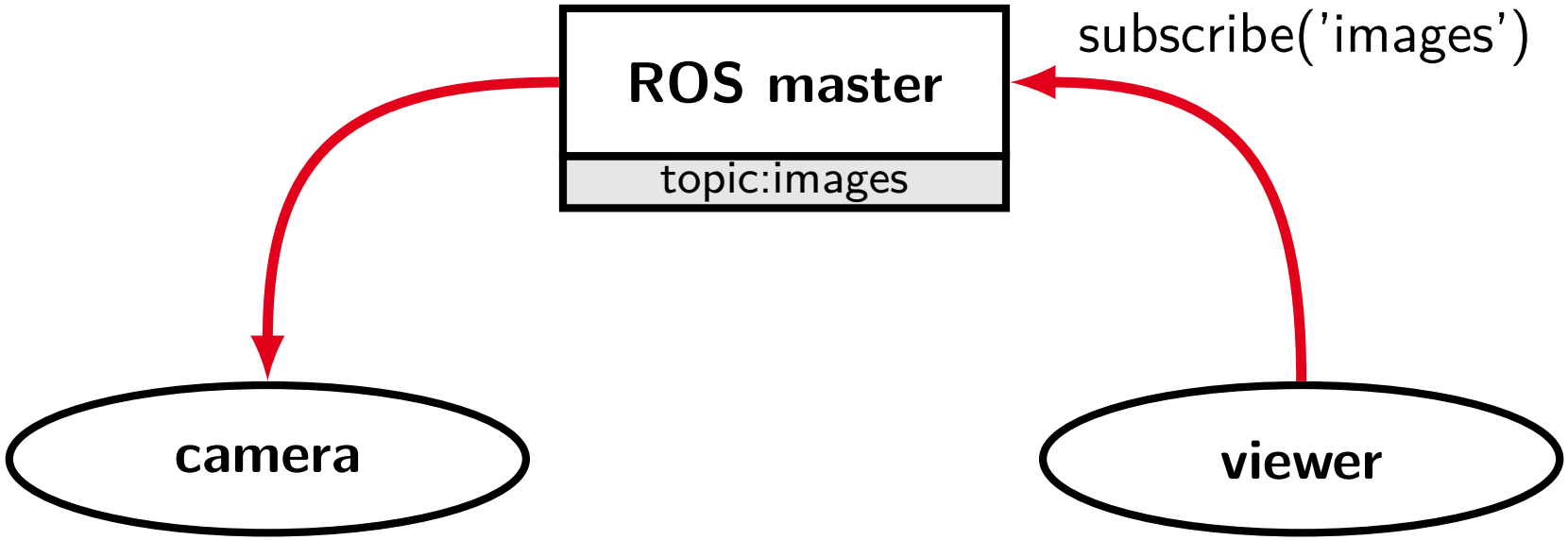
Communication - Example



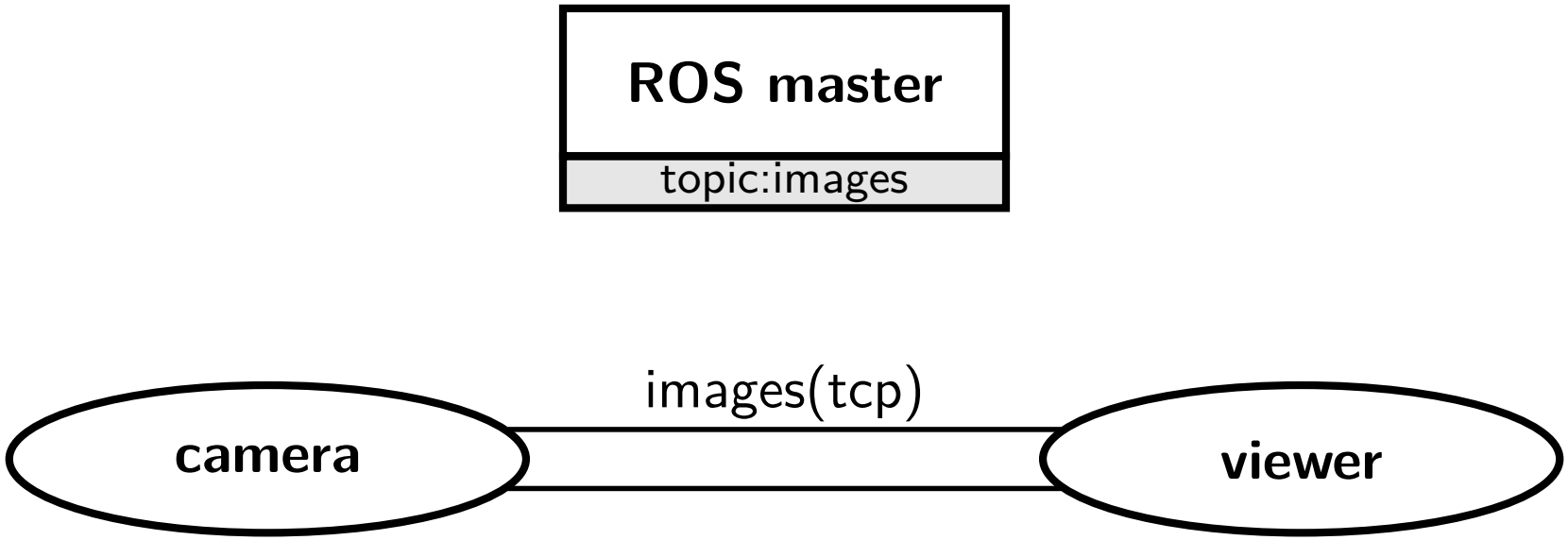
Communication - Example



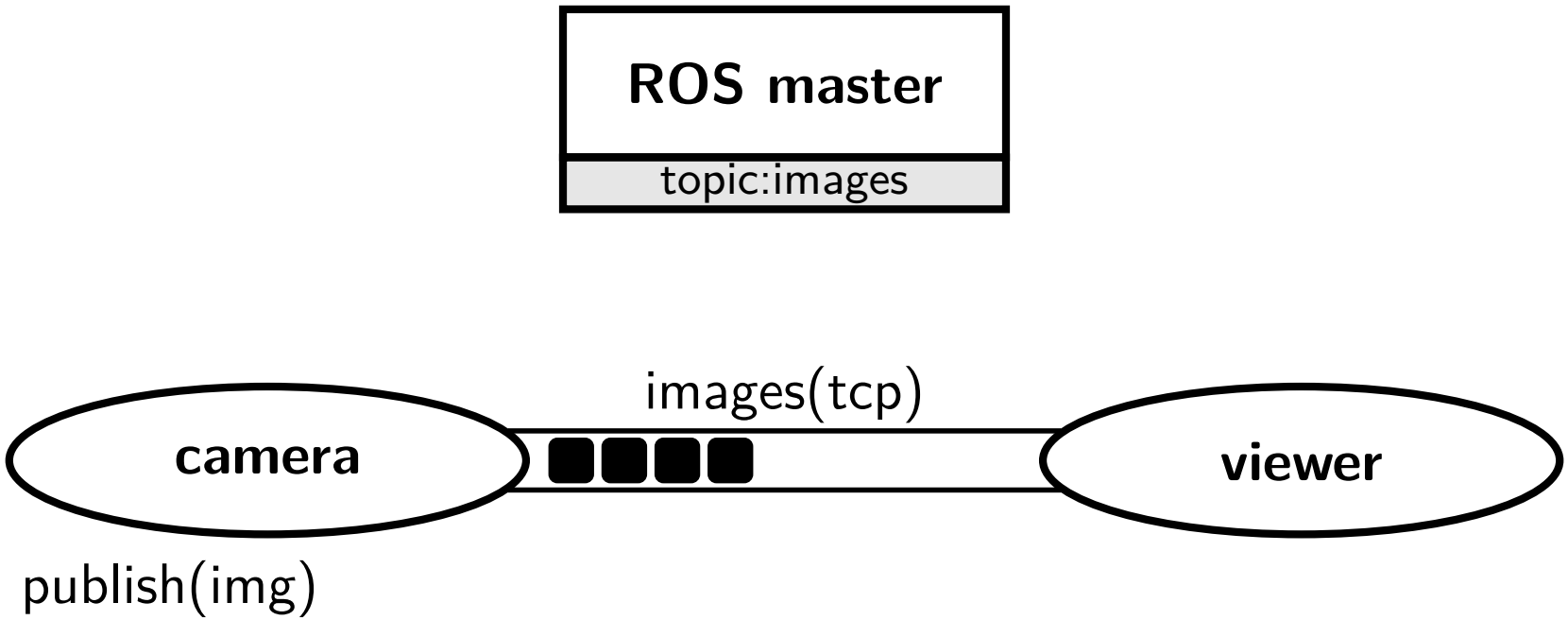
Communication - Example



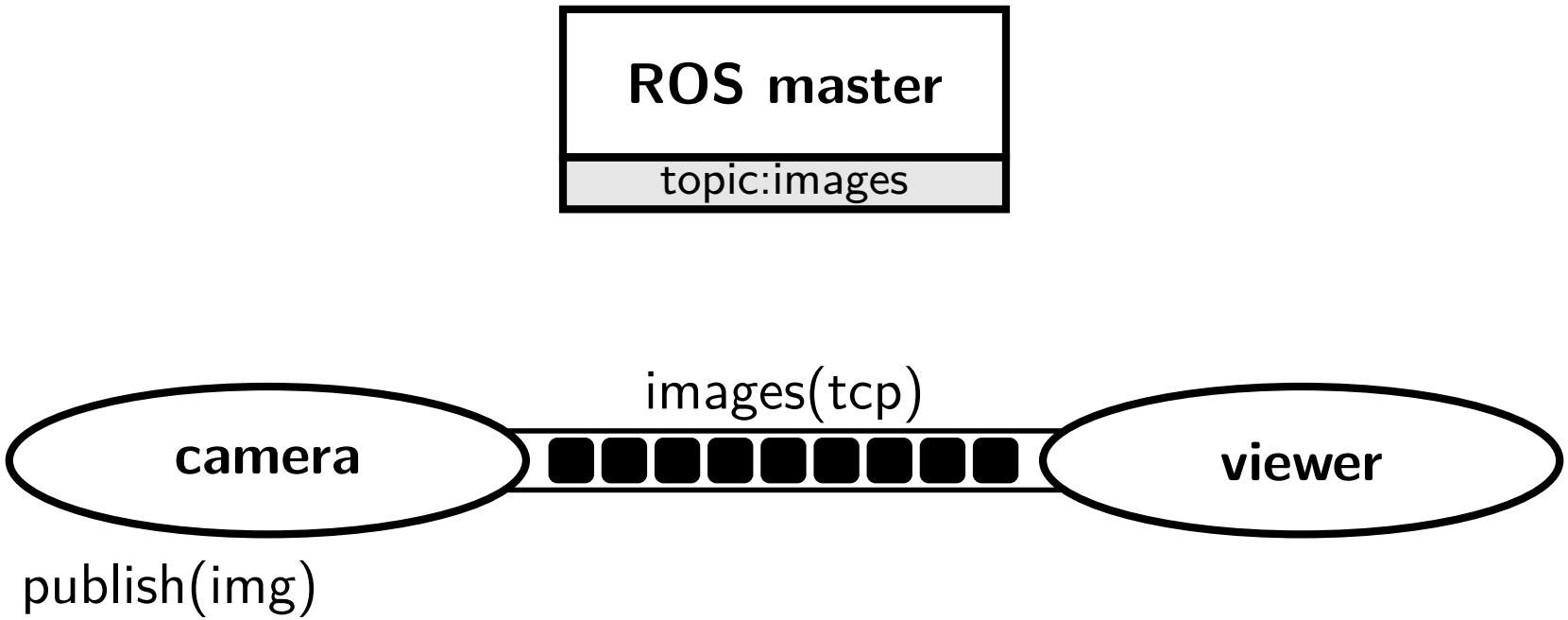
Communication - Example



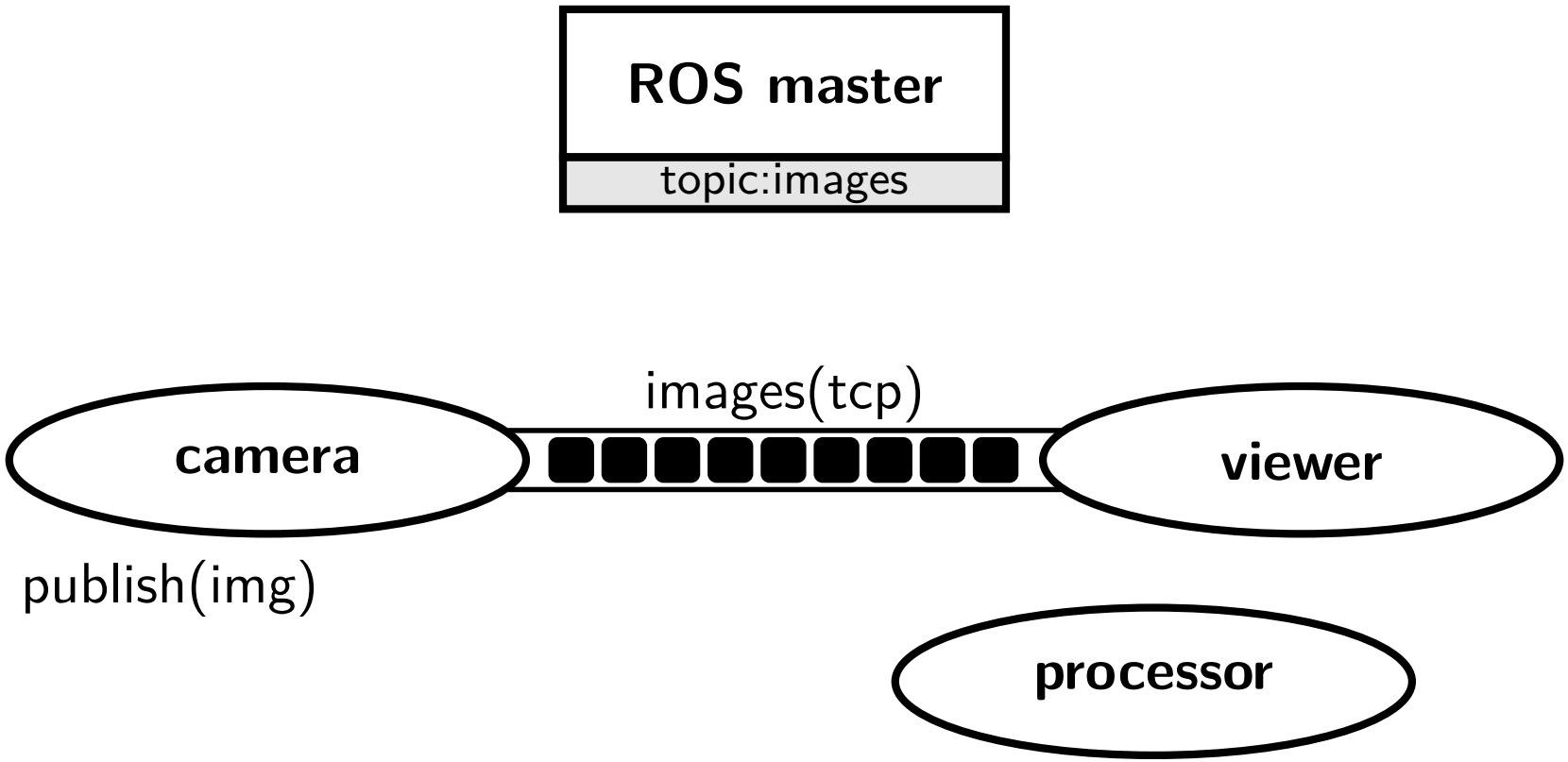
Communication - Example



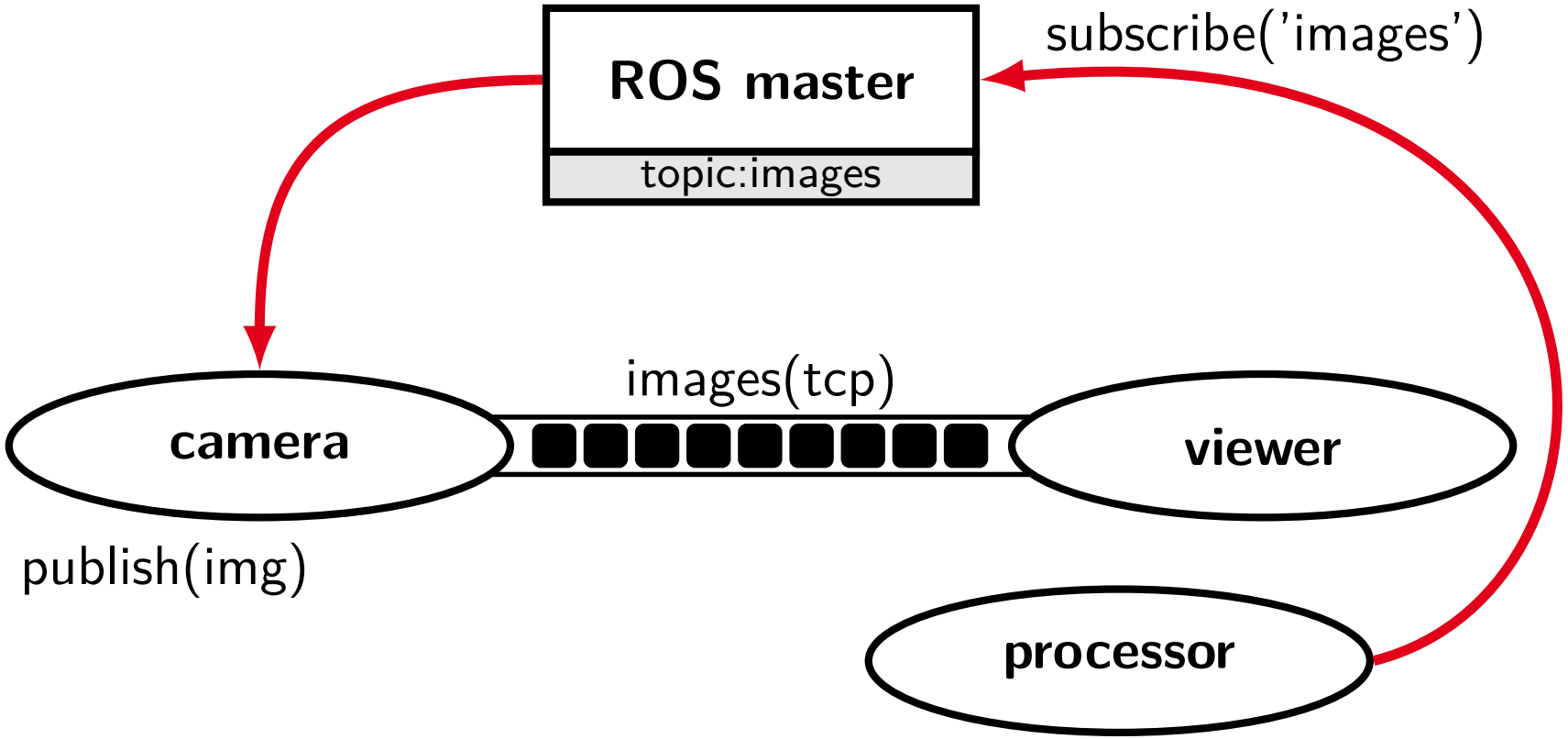
Communication - Example



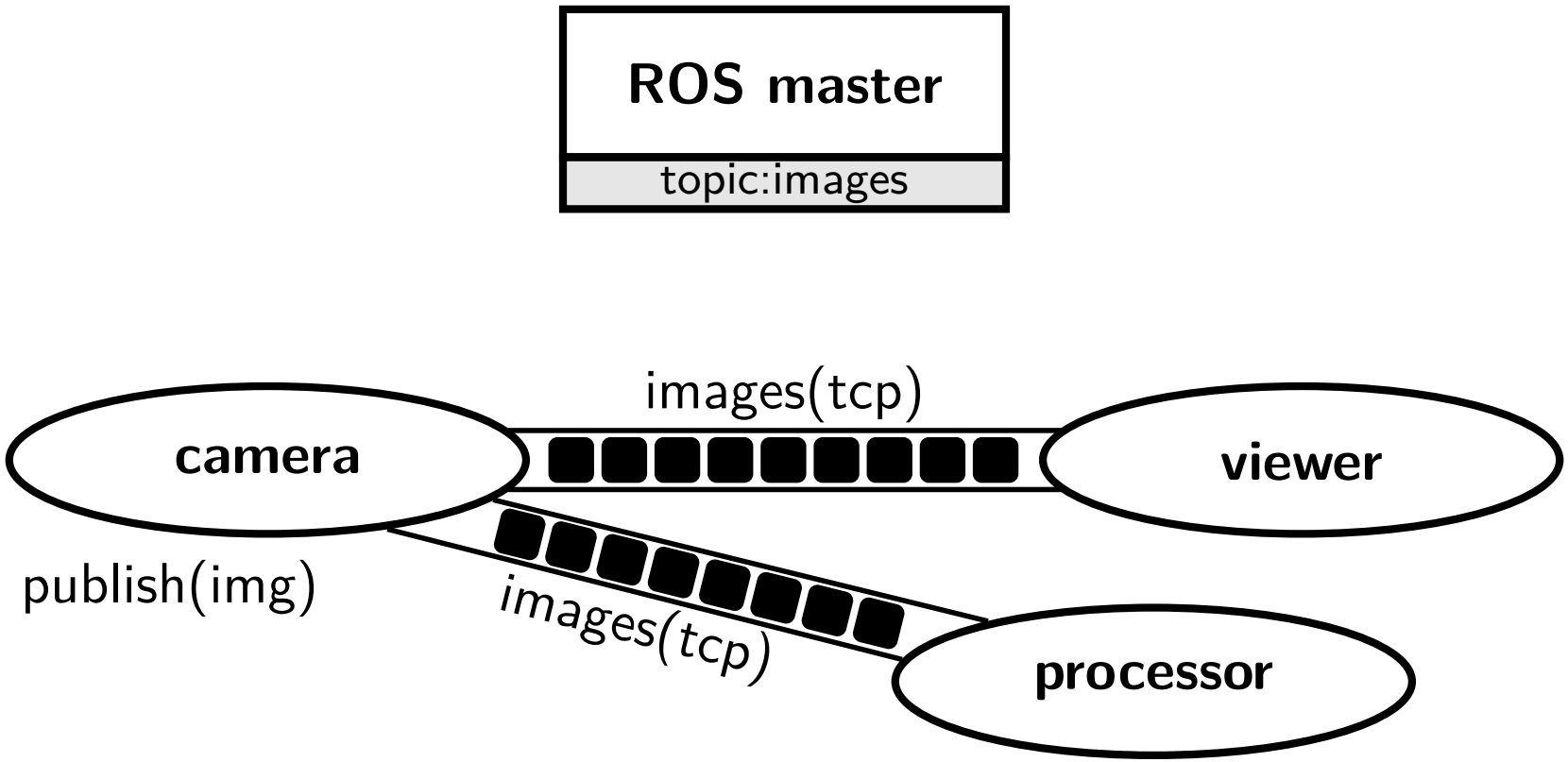
Communication - Example



Communication - Example



Communication - Example



Services

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

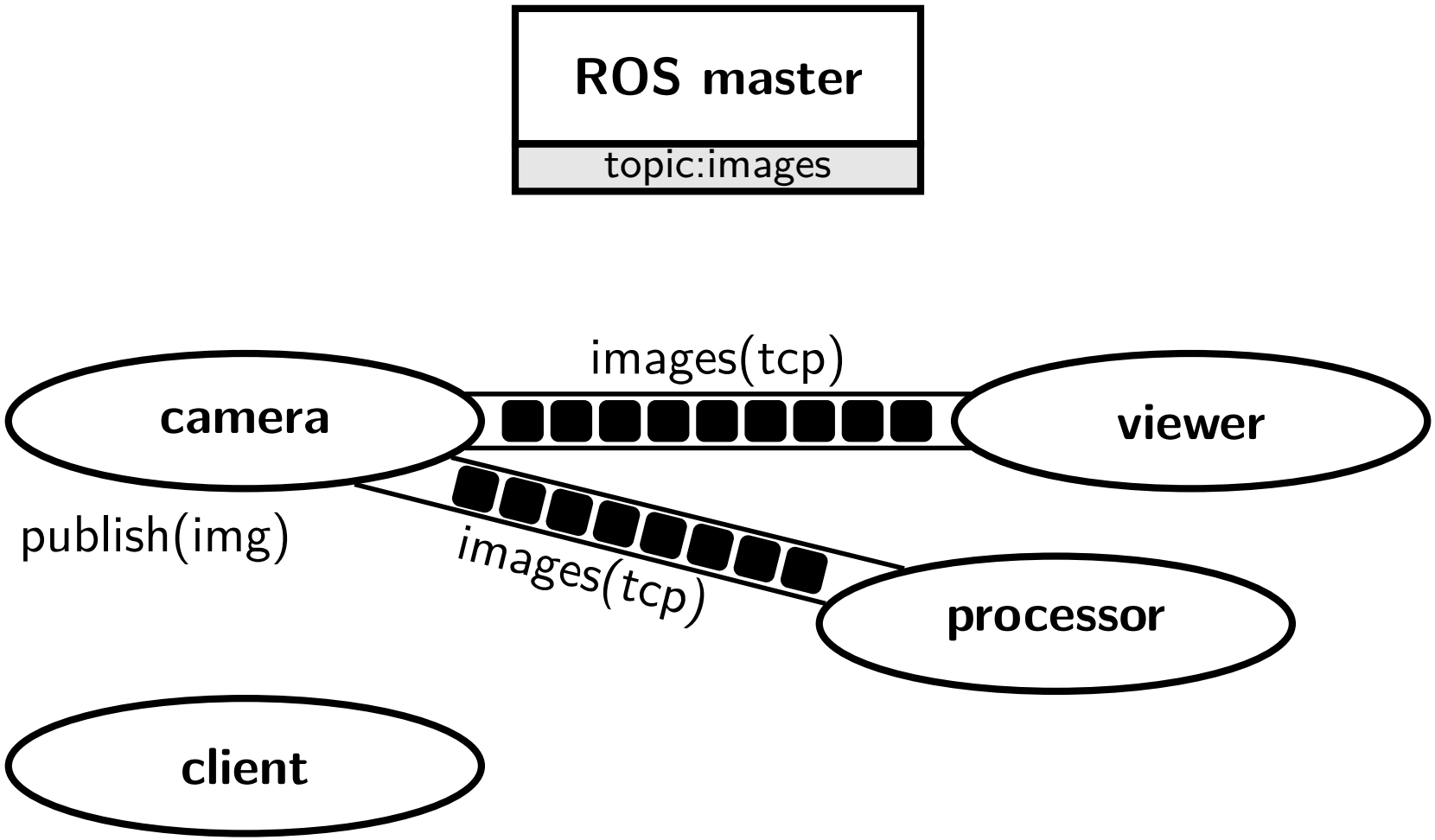
```
$ rosservice type add_two_ints | rossrv show
int64 a
int64 b
- - -
int64 sum
```

Services

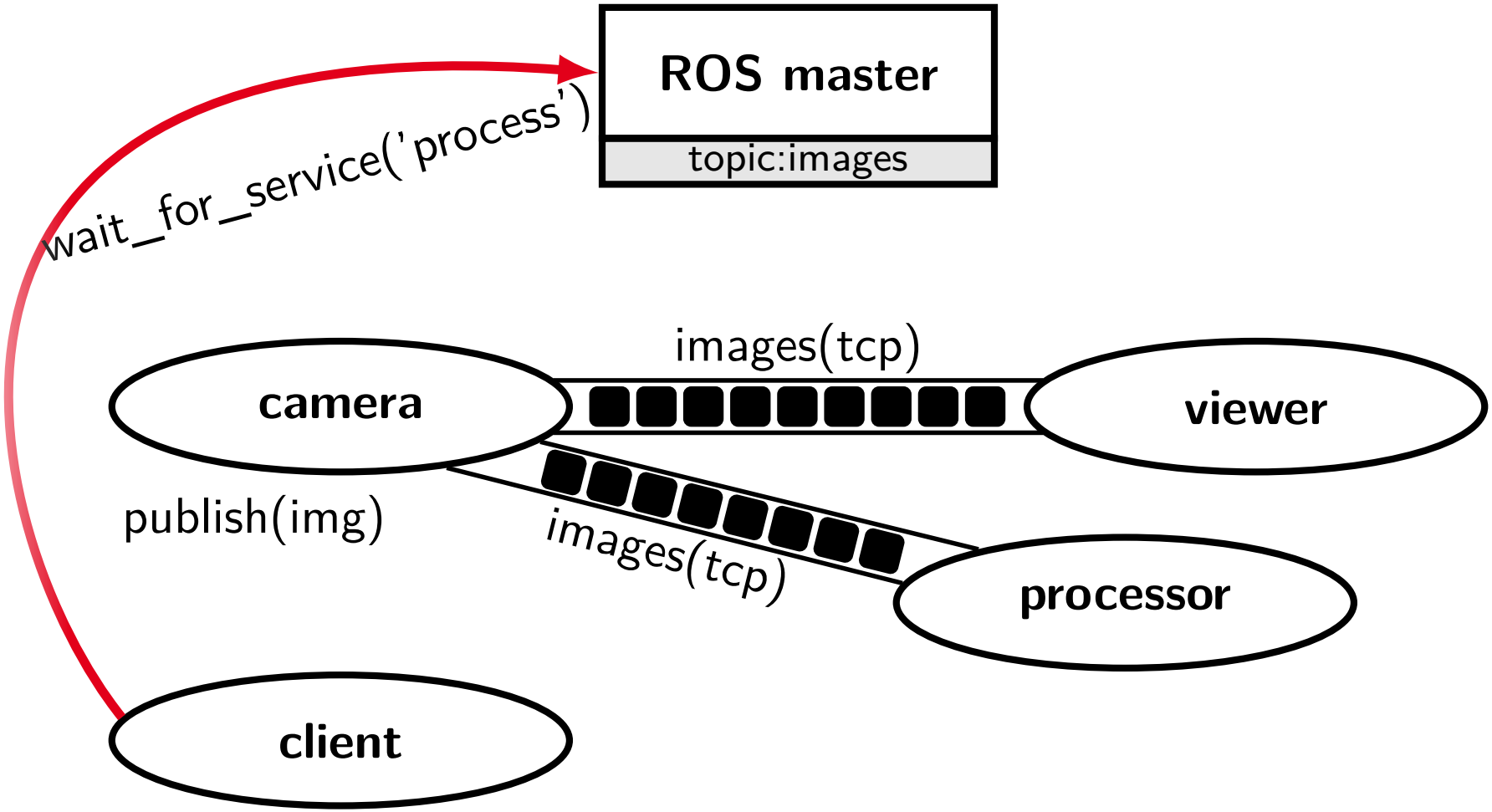
- 2 message types
  - request and response
- Synchronous protocol
  - client sends request
  - client waits for server
  - server replies

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

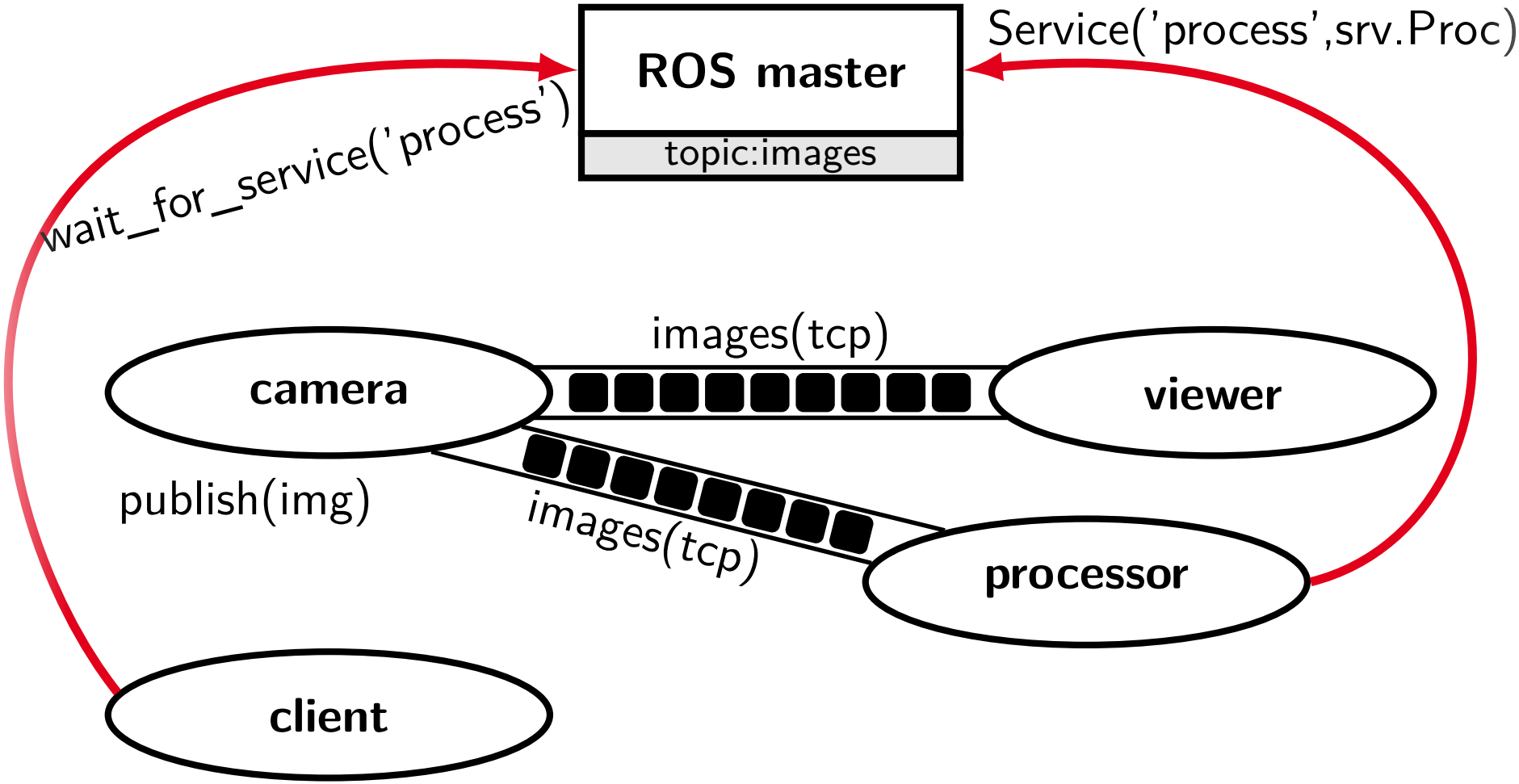
Communication - Example



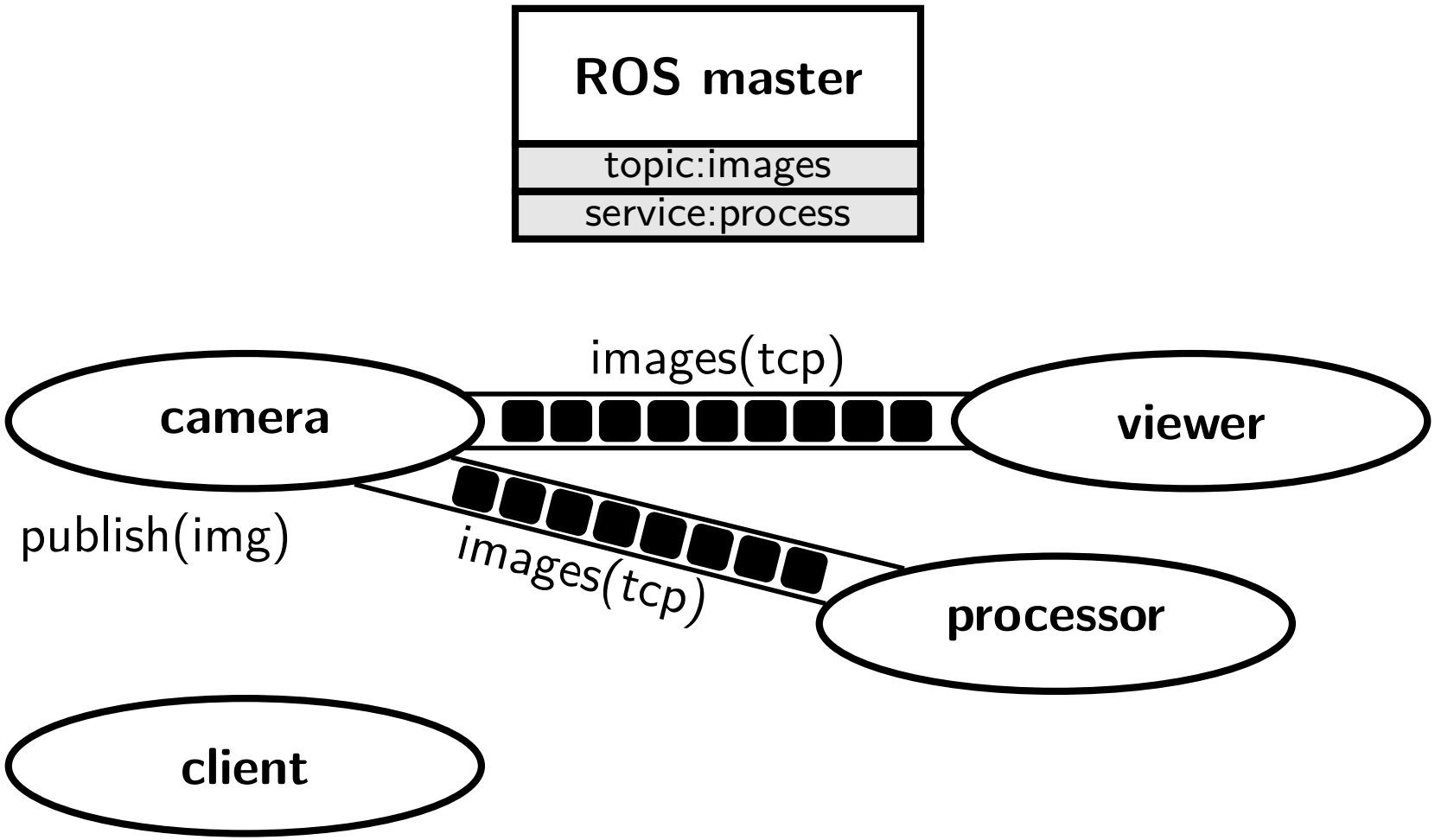
Communication - Example



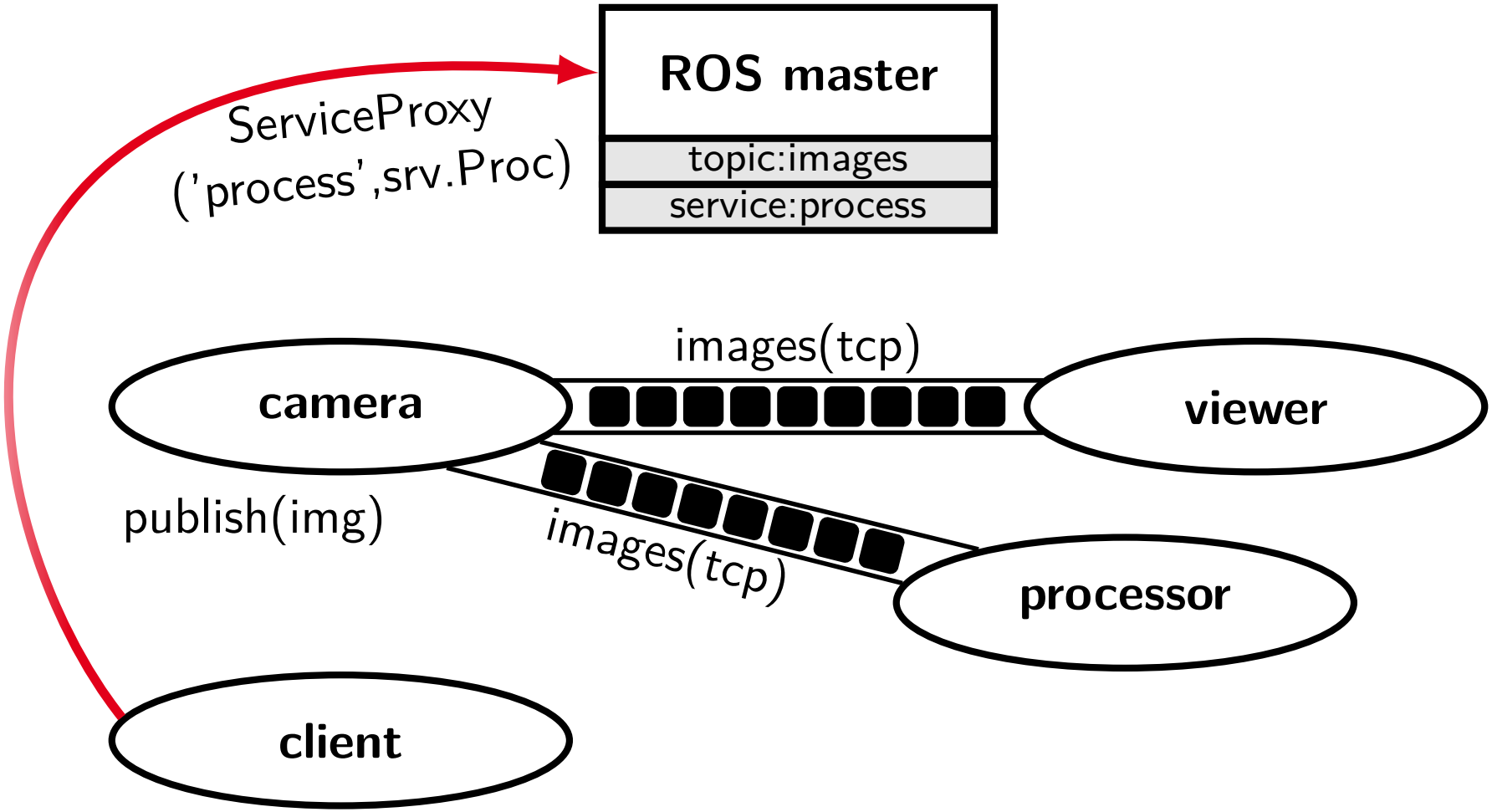
Communication - Example



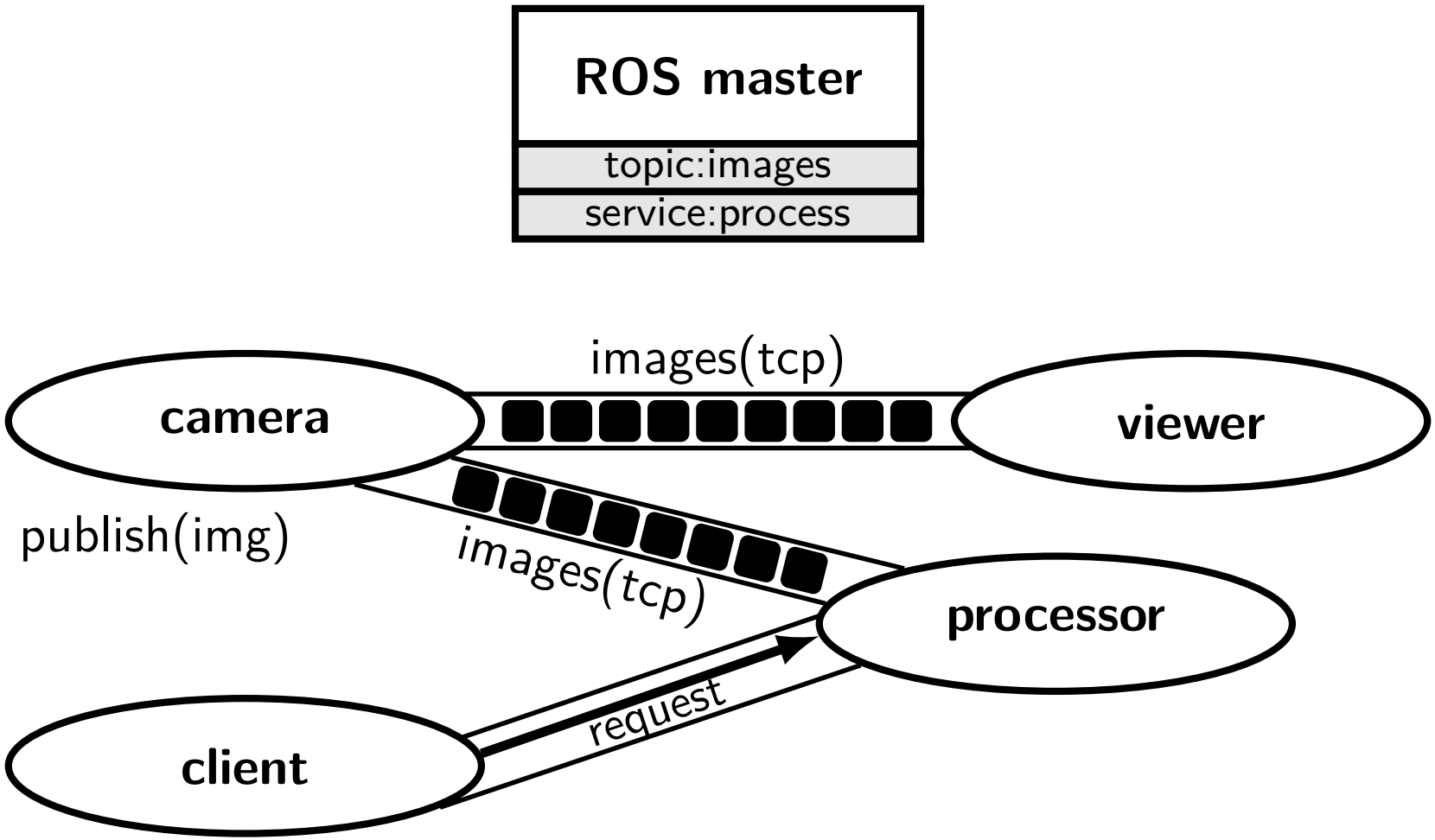
Communication - Example



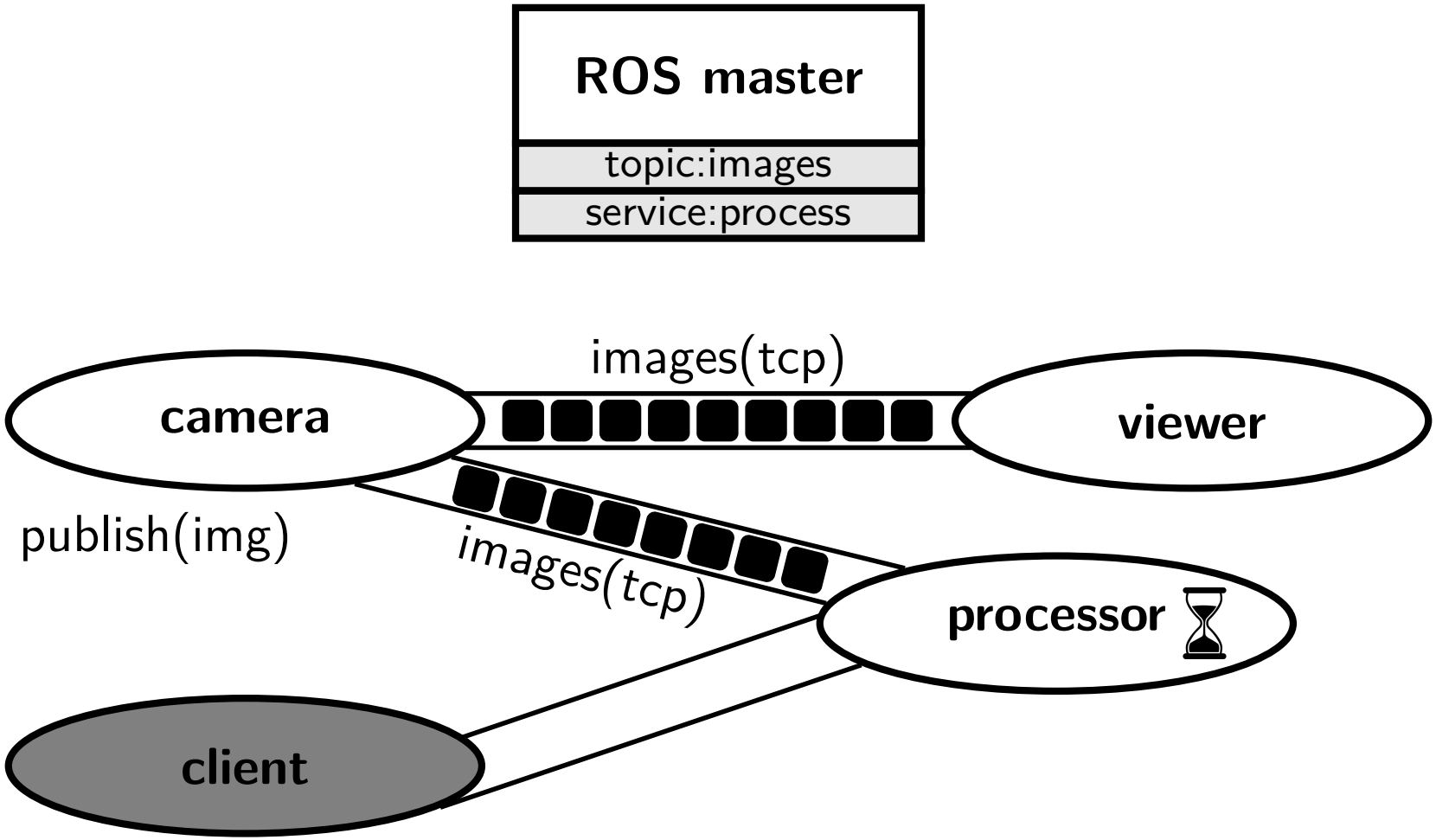
Communication - Example



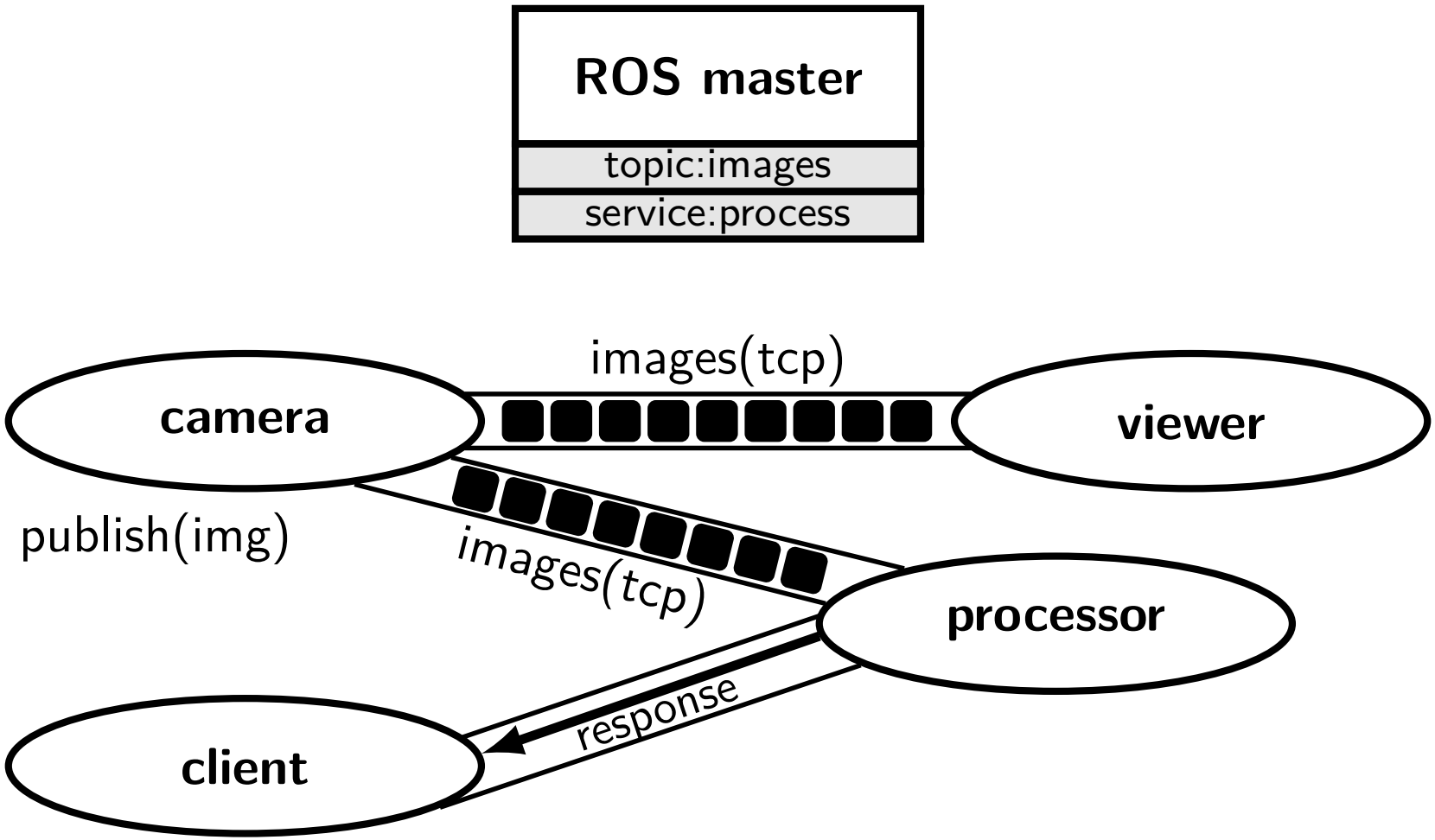
Communication - Example



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

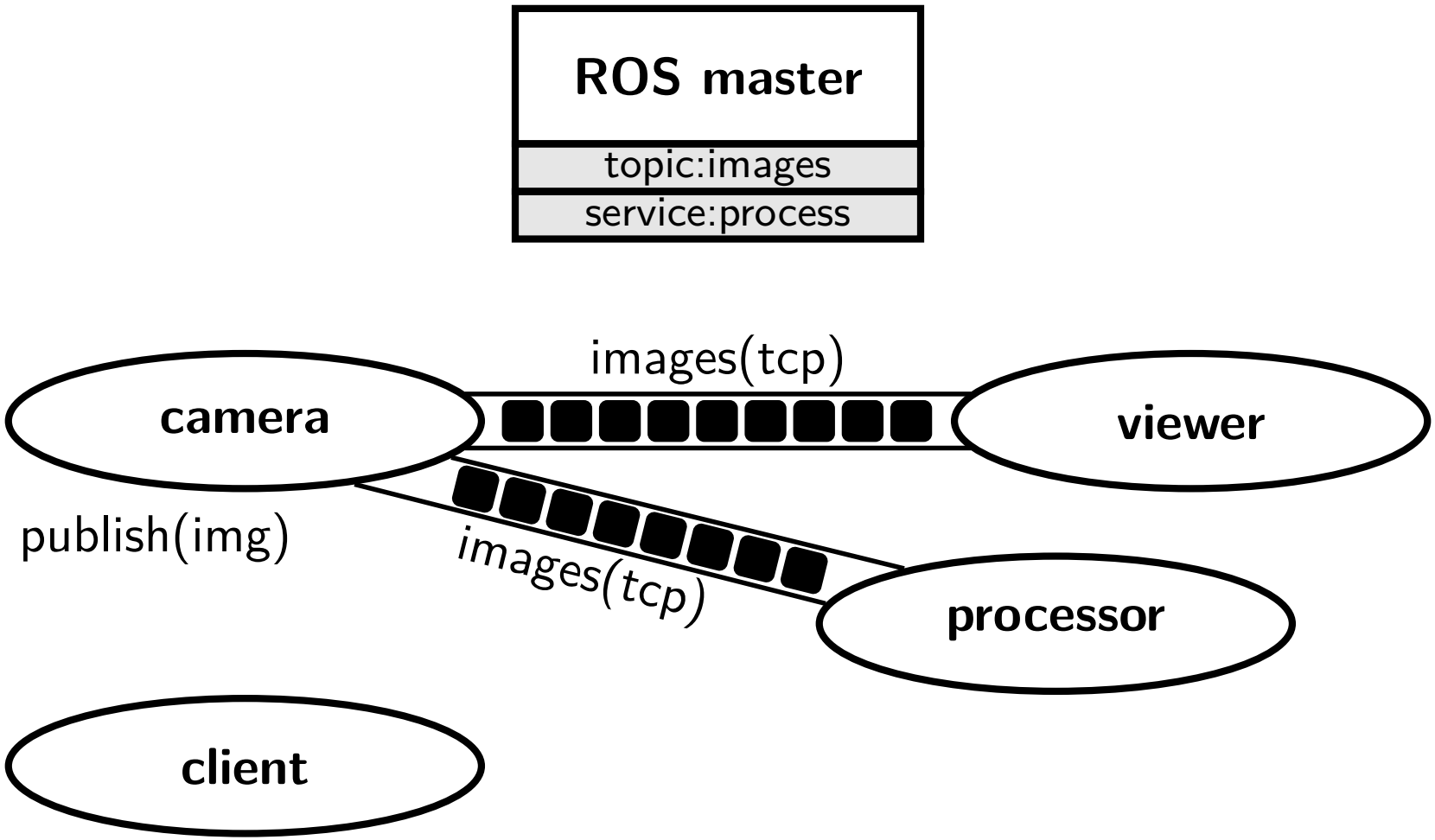
```
# Define the goal
uint32 dishwasher_id    # Specify which dishwasher we want to use
- - -
# Define the result
uint32 total_dishes_cleaned
- - -
# Define a feedback message
float32 percent_complete
```

Actions

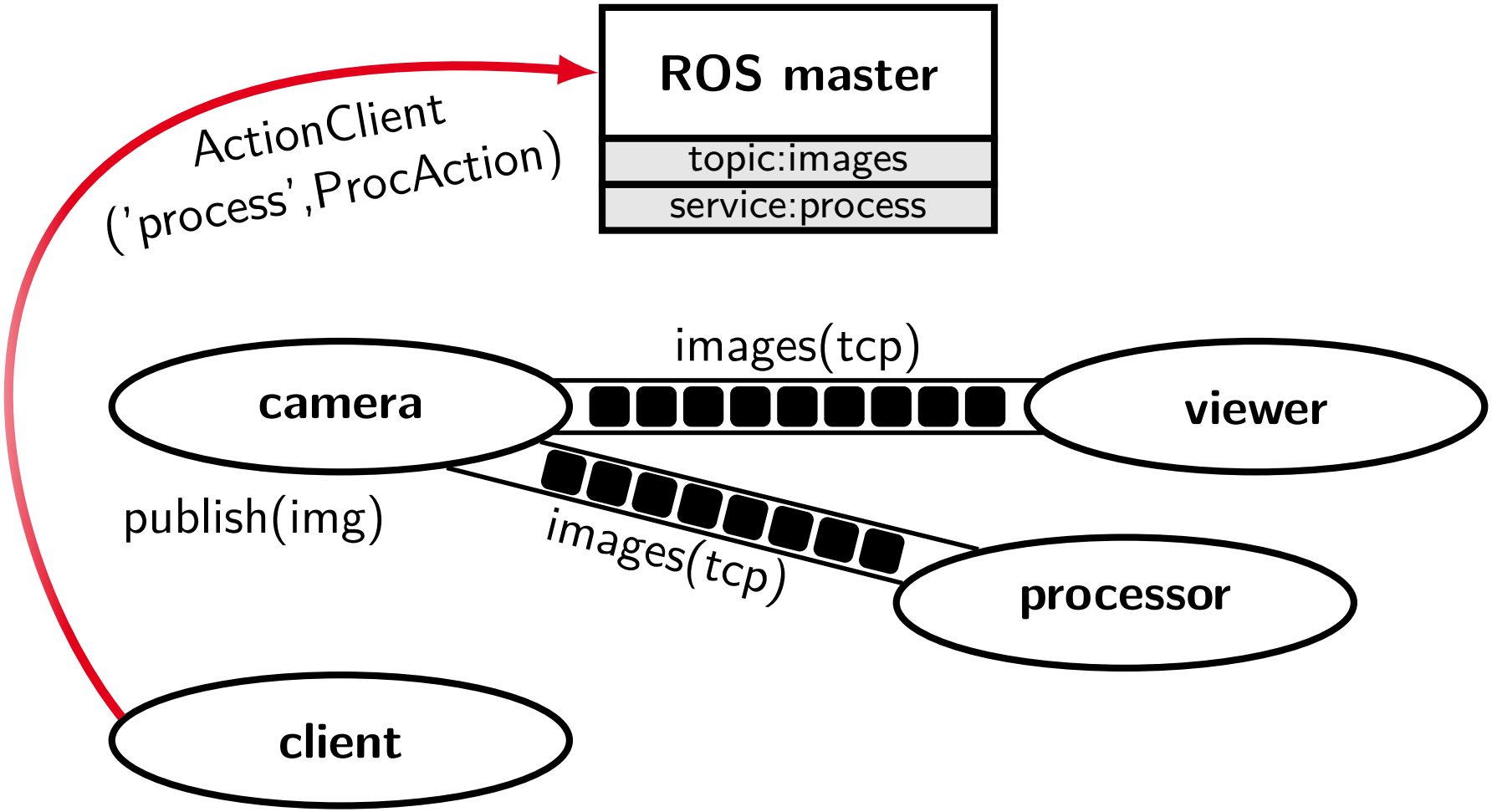
- 3 message types
  - goal and result
  - optional feedback
- Asynchronous protocol
  - client sends goal
  - server may respond with feedback
  - server delivers result
- Interruptible

```
# 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
```

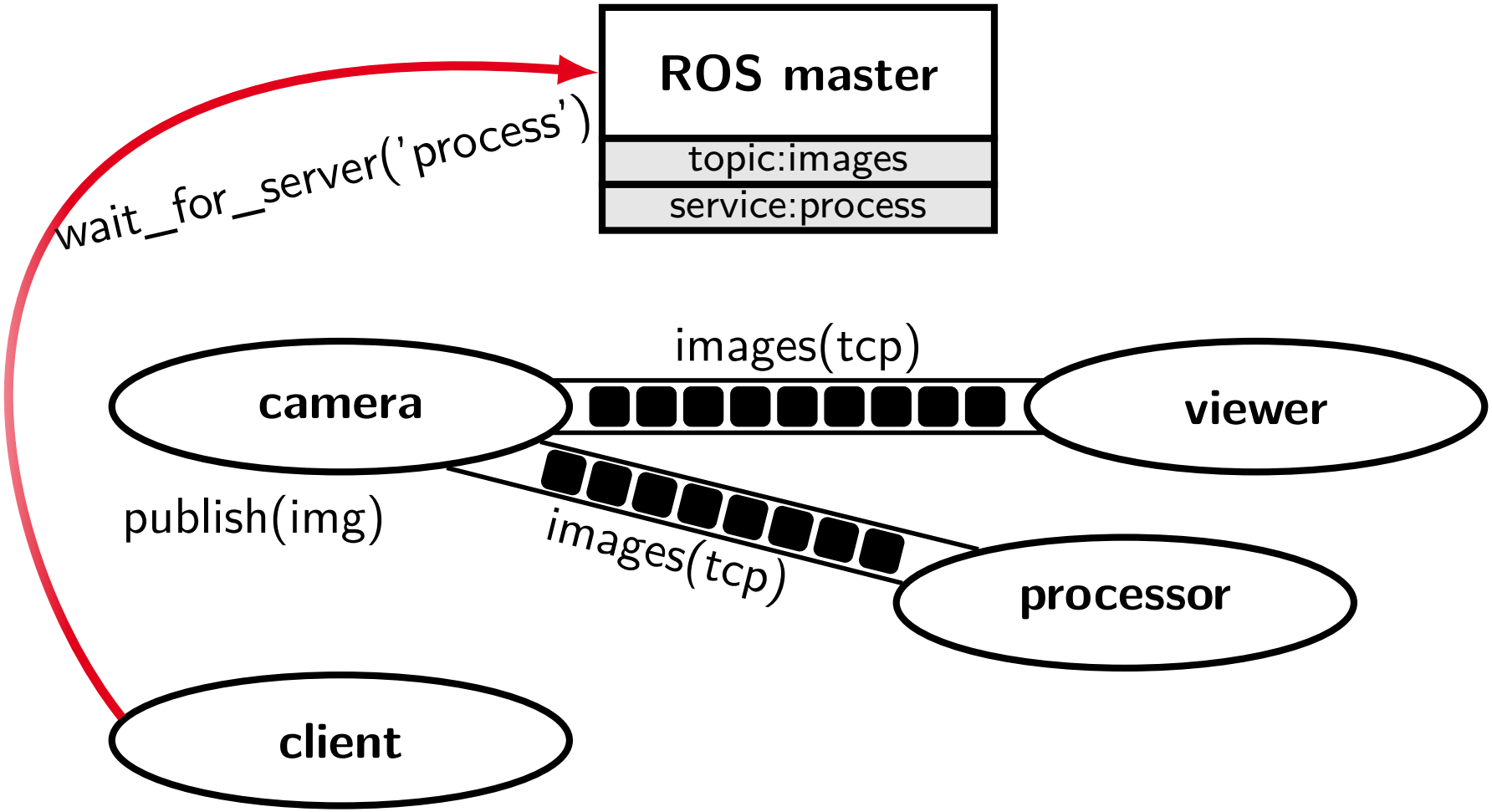
Communication - Example



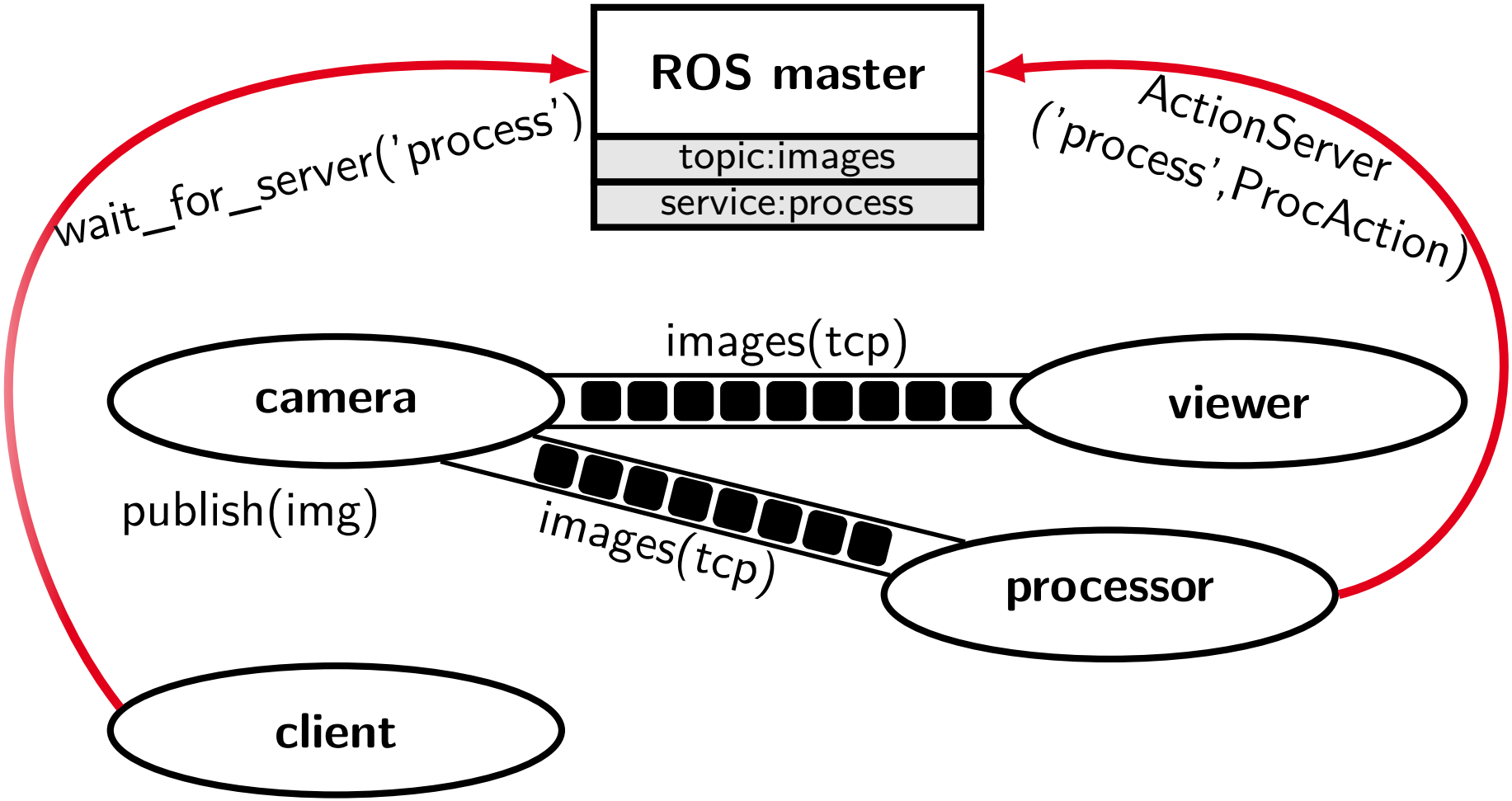
Communication - Example



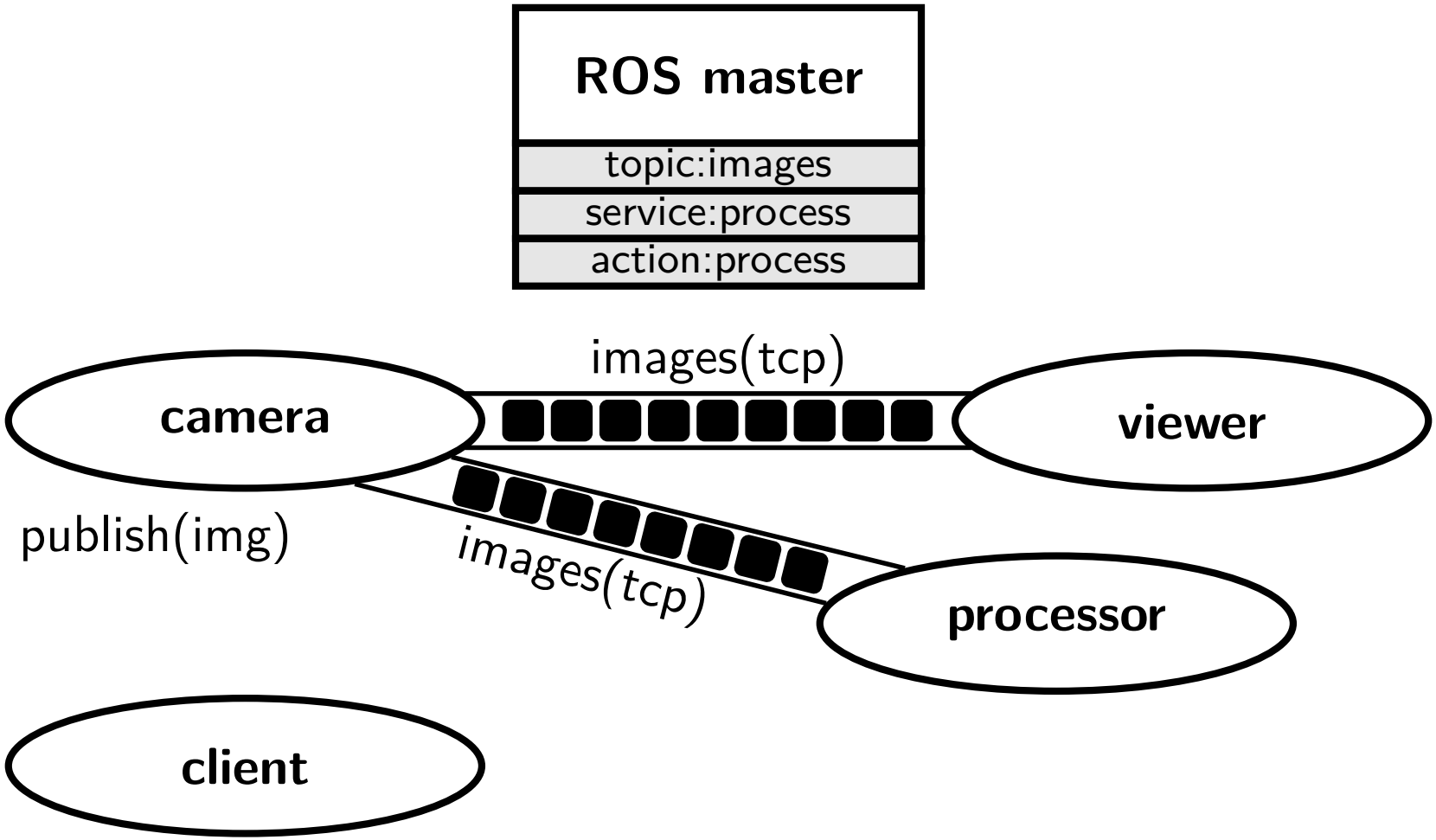
Communication - Example



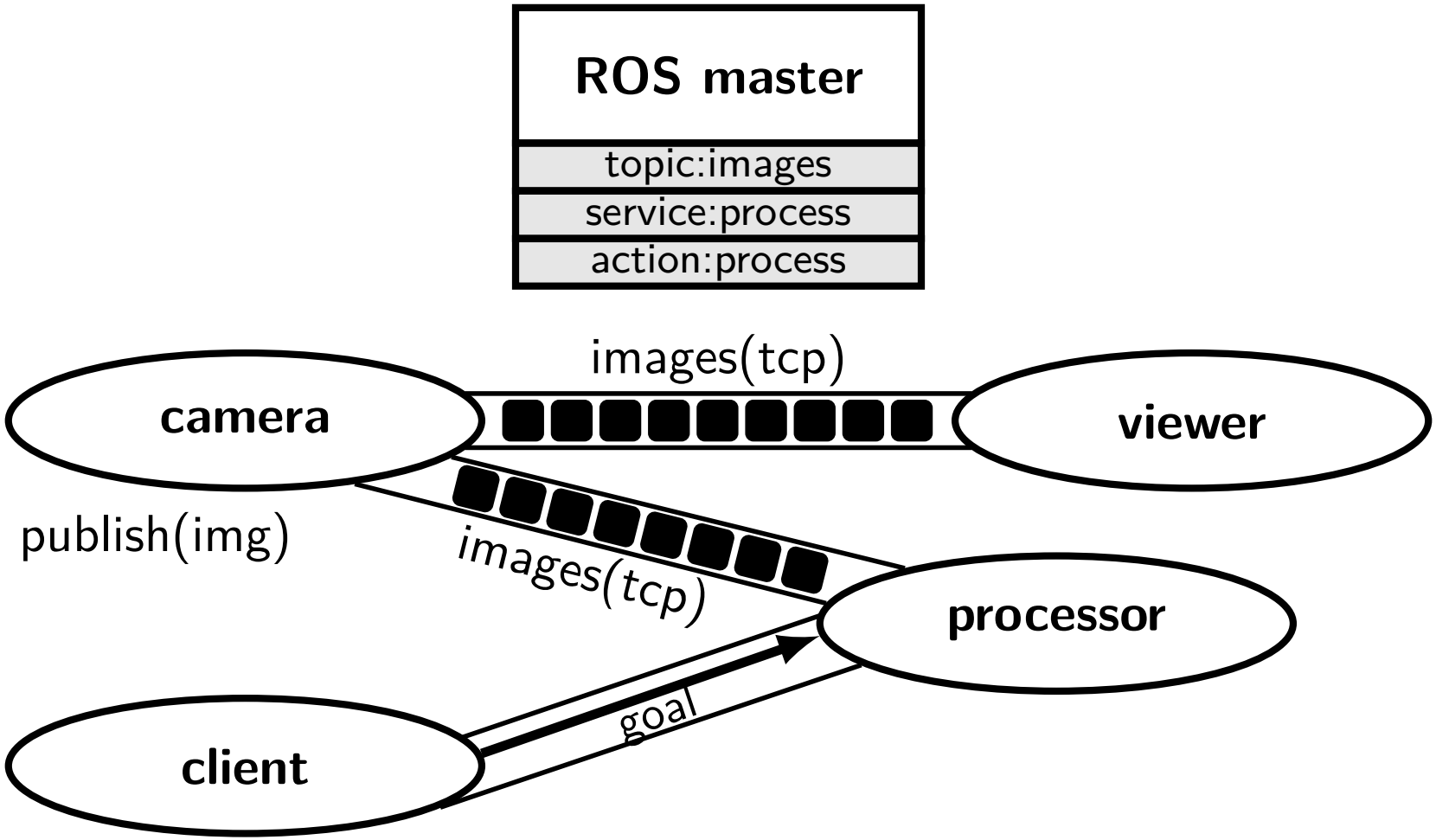
Communication - Example



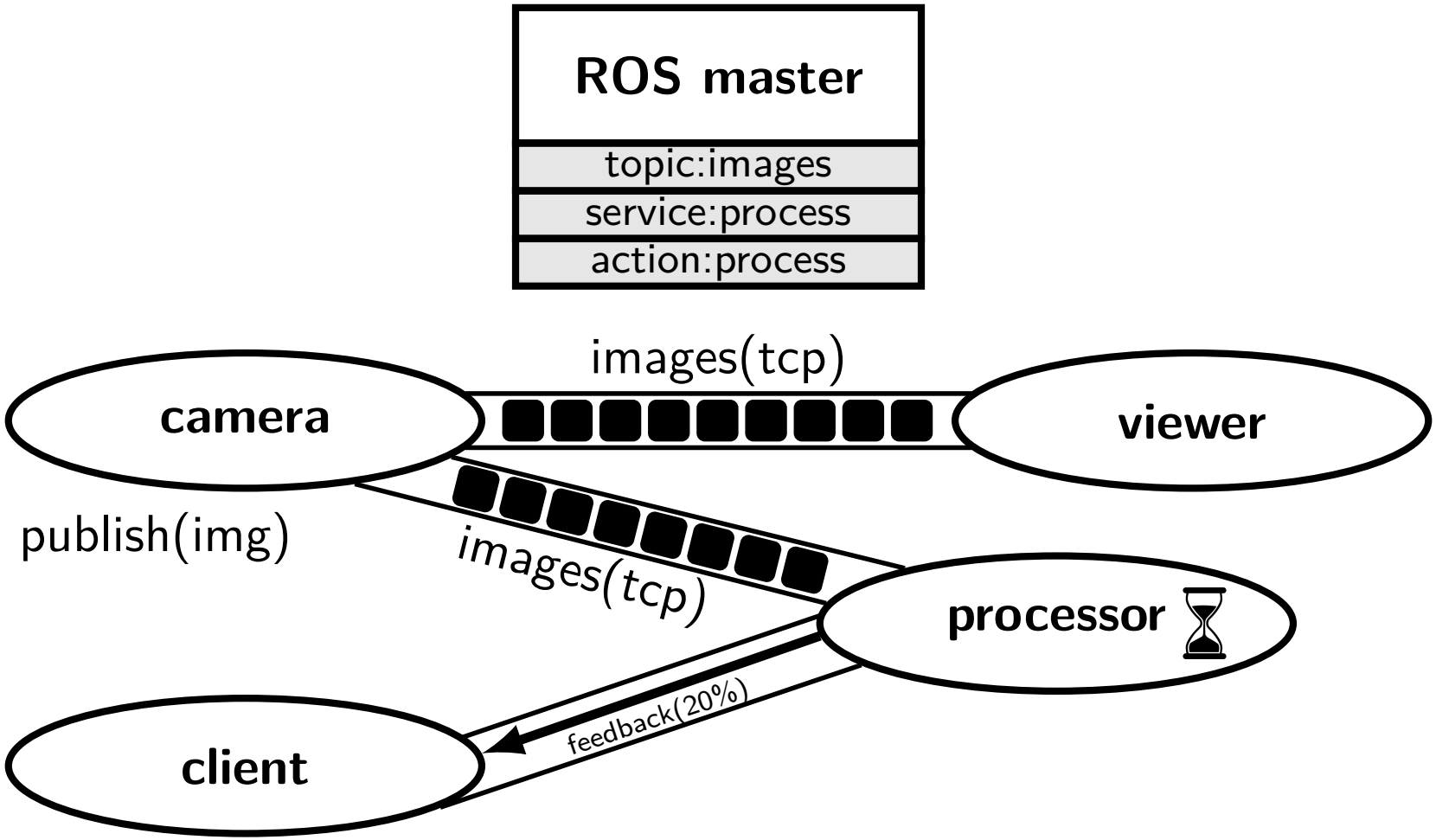
Communication - Example



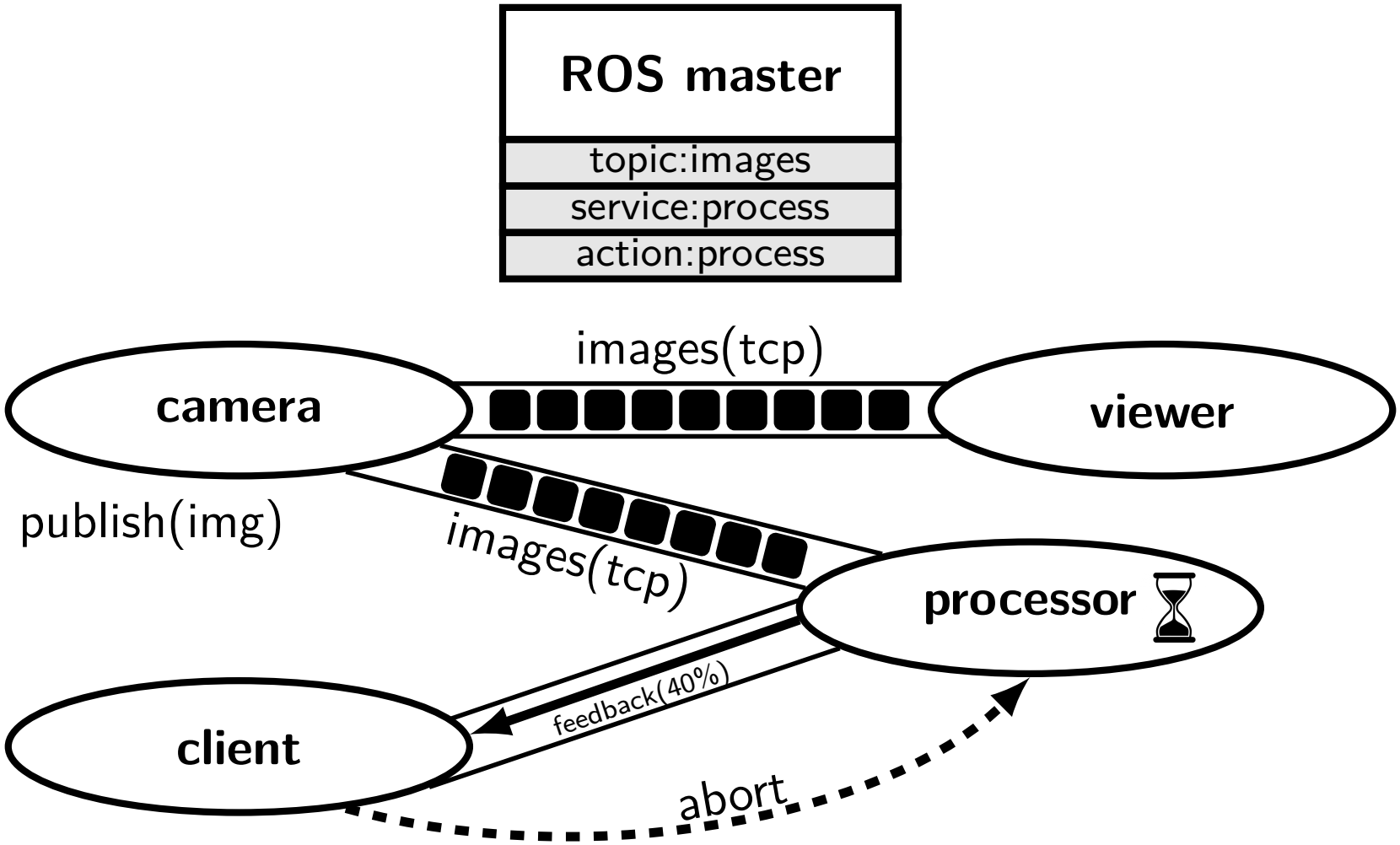
Communication - Example



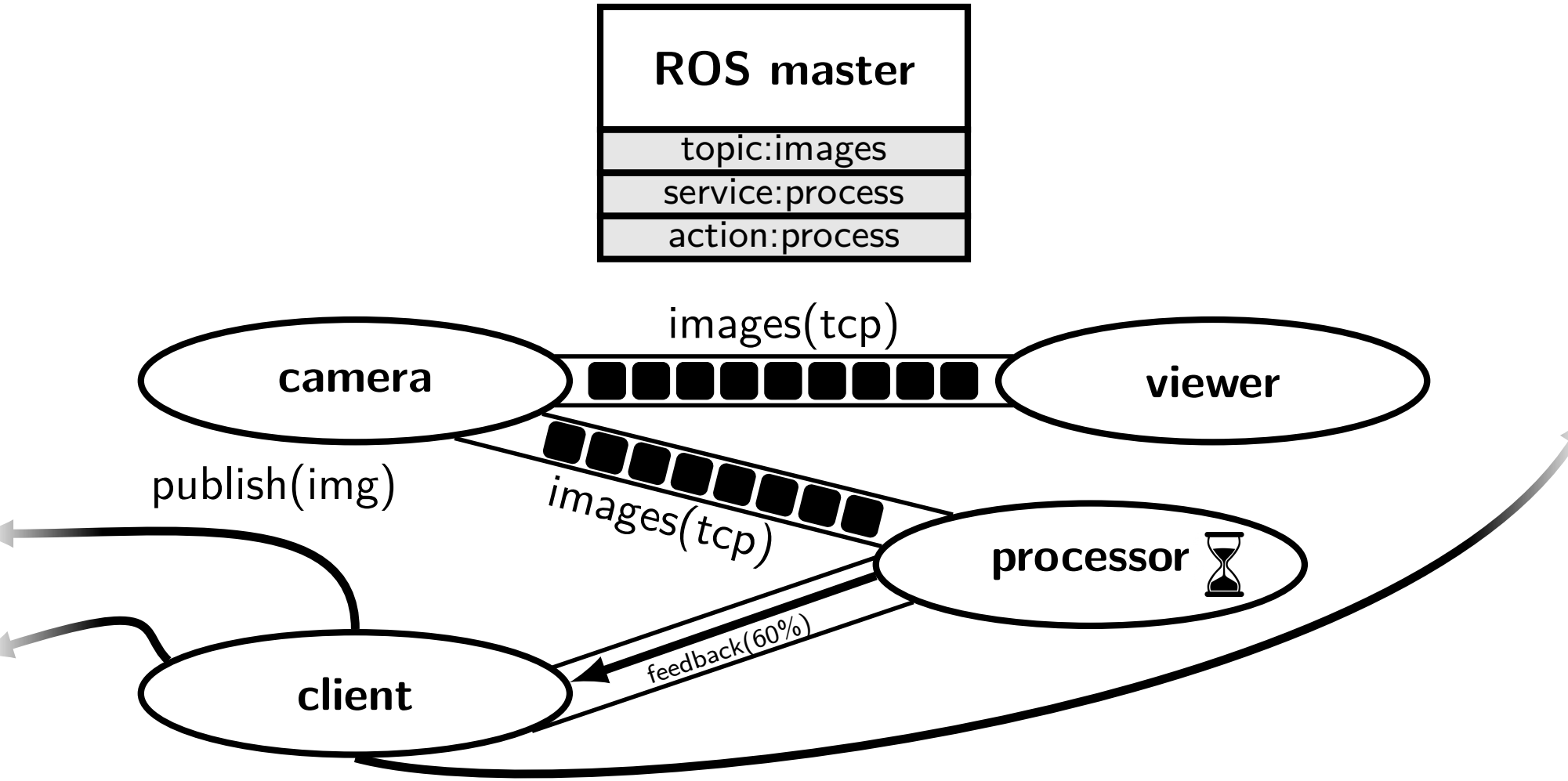
Communication - Example



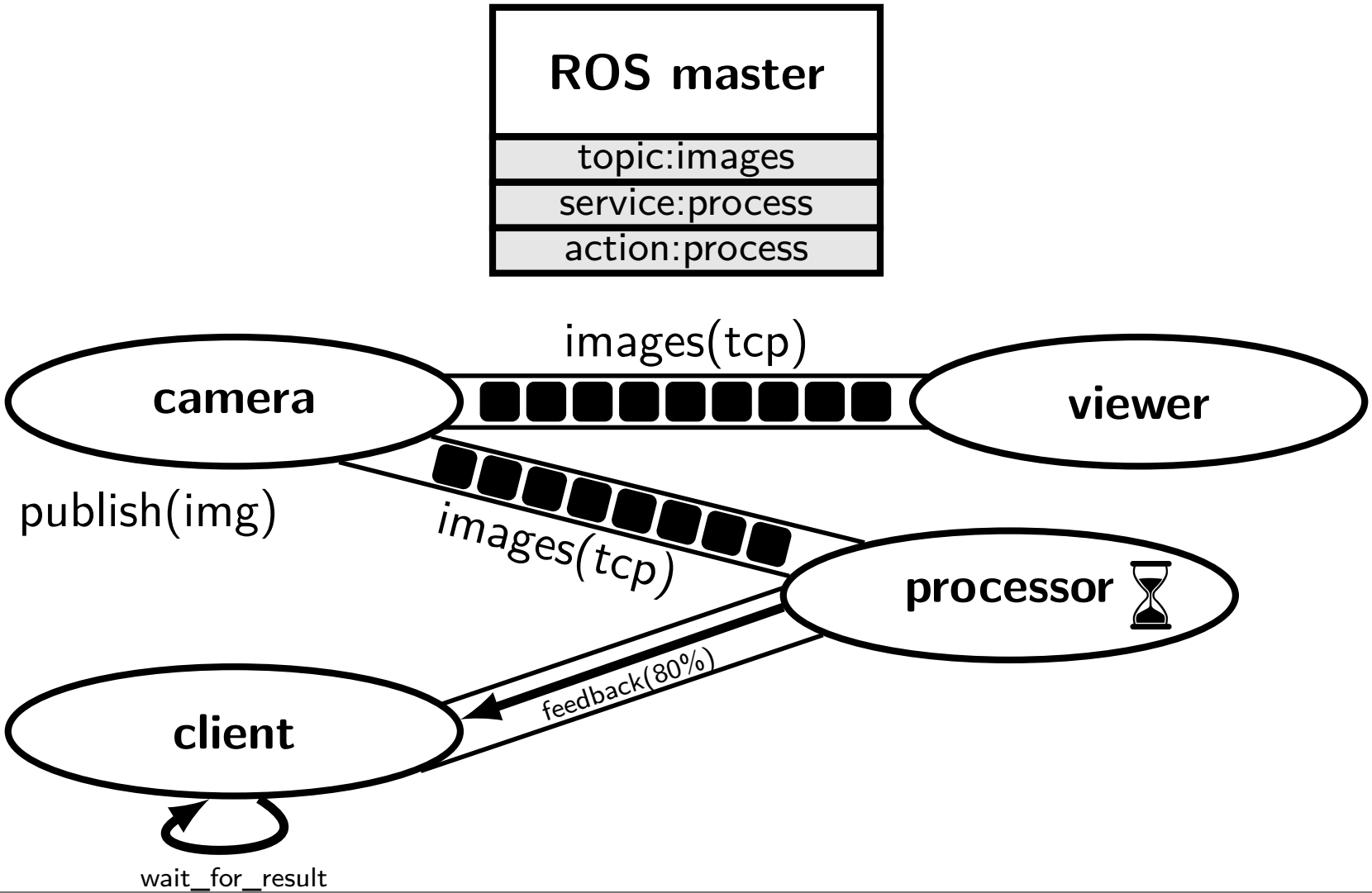
Communication - Example



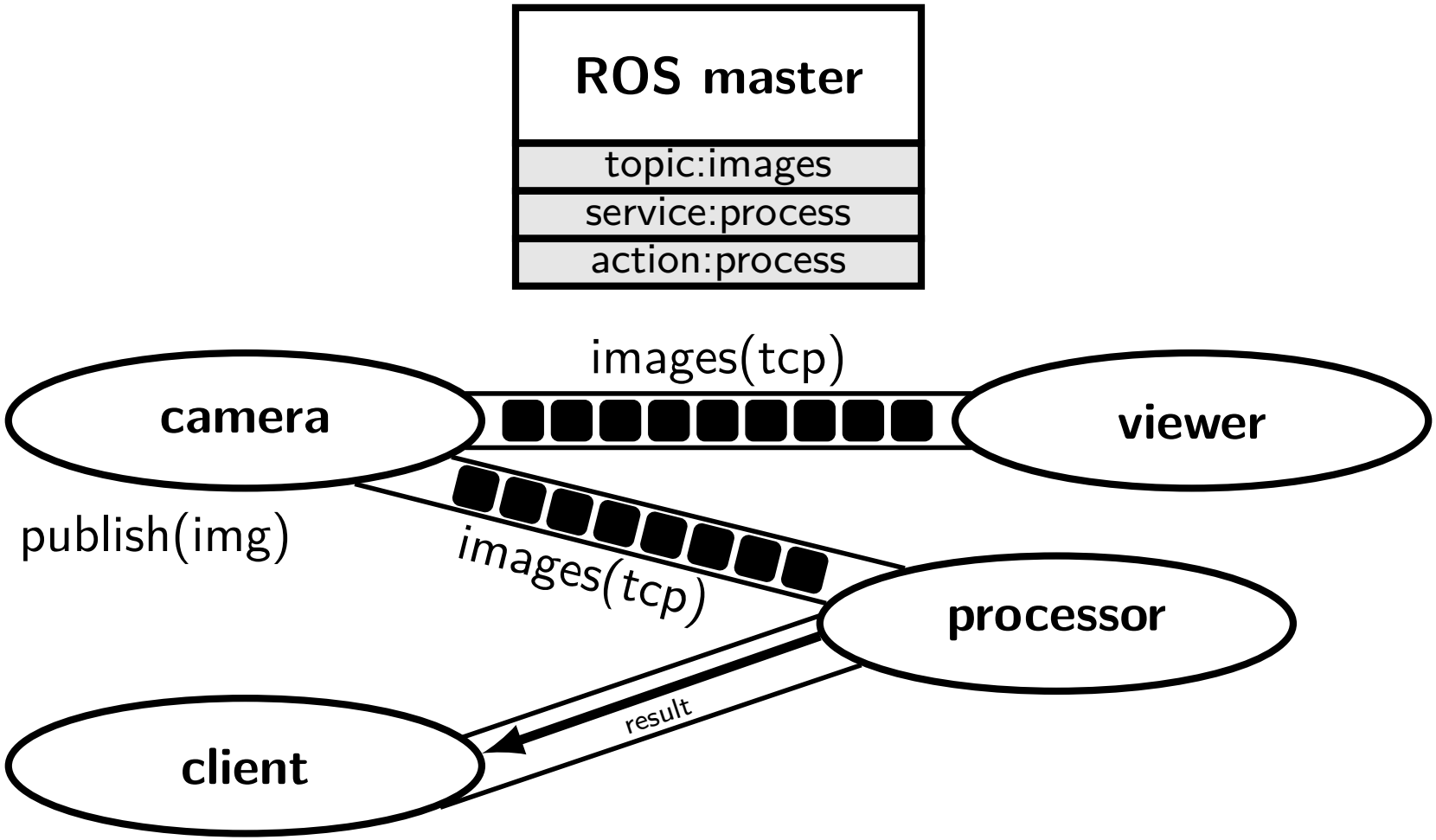
Communication - Example



Communication - Example



Communication - Example



Tools and Visualization

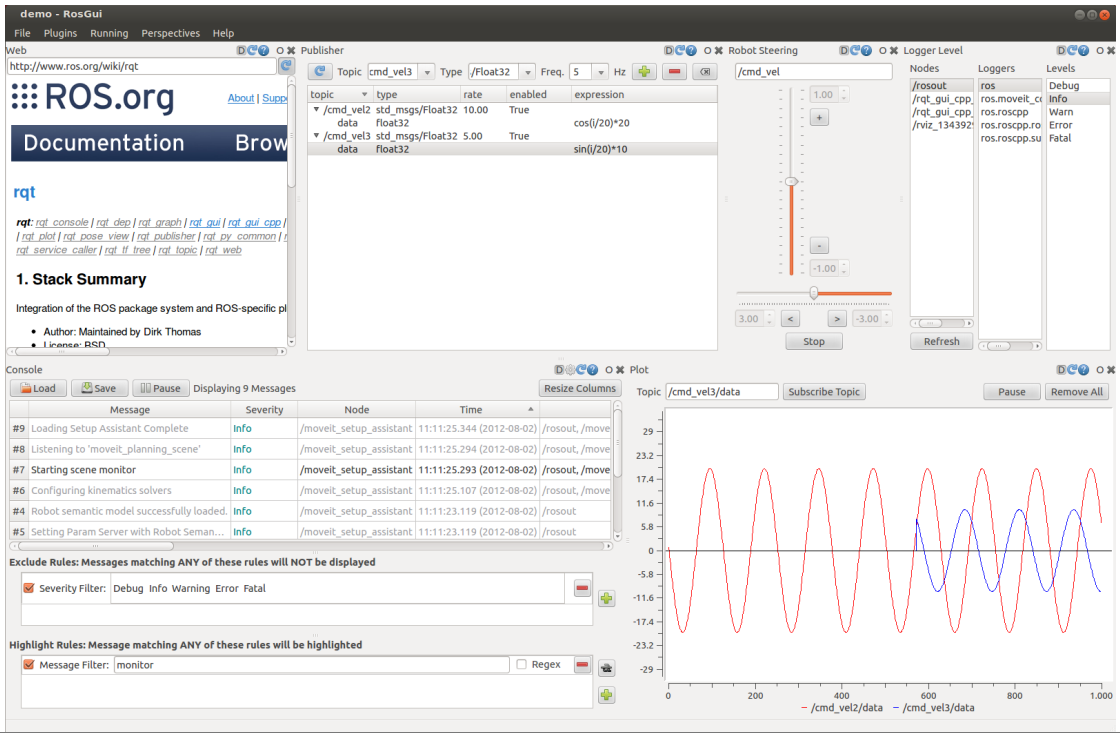
- Standardized interfaces allow using tools in various applications
- ROS-provided tools
  - ROS Bag $\Rightarrow$ ROS2 Bag
  - RQT $\Rightarrow$ RQT2
  - RViz $\Rightarrow$ RViz2
- User-provided tools
  - PlotJuggler
  - RQT-Plugins
  - Teleoperation node

ROS/ROS 2 Bag

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

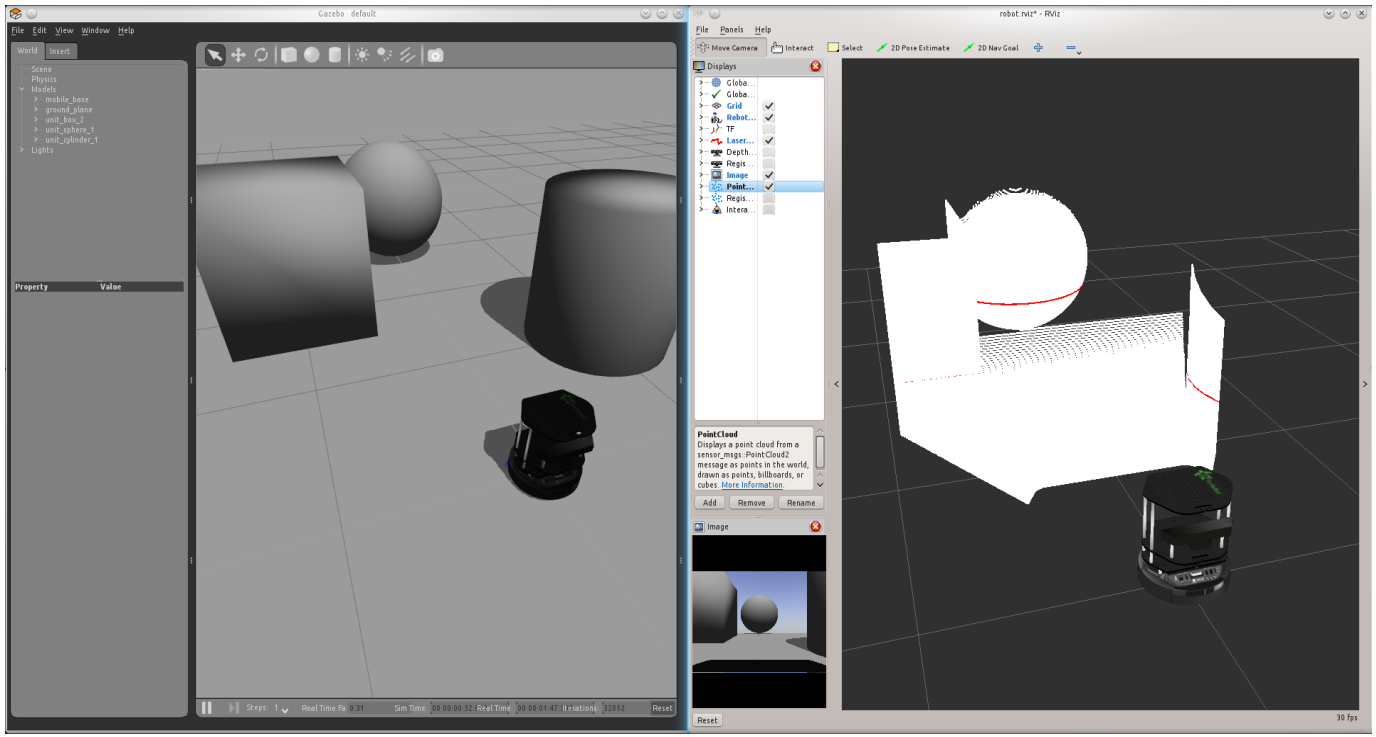
RQT/RQT2

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



RViz/RViz2

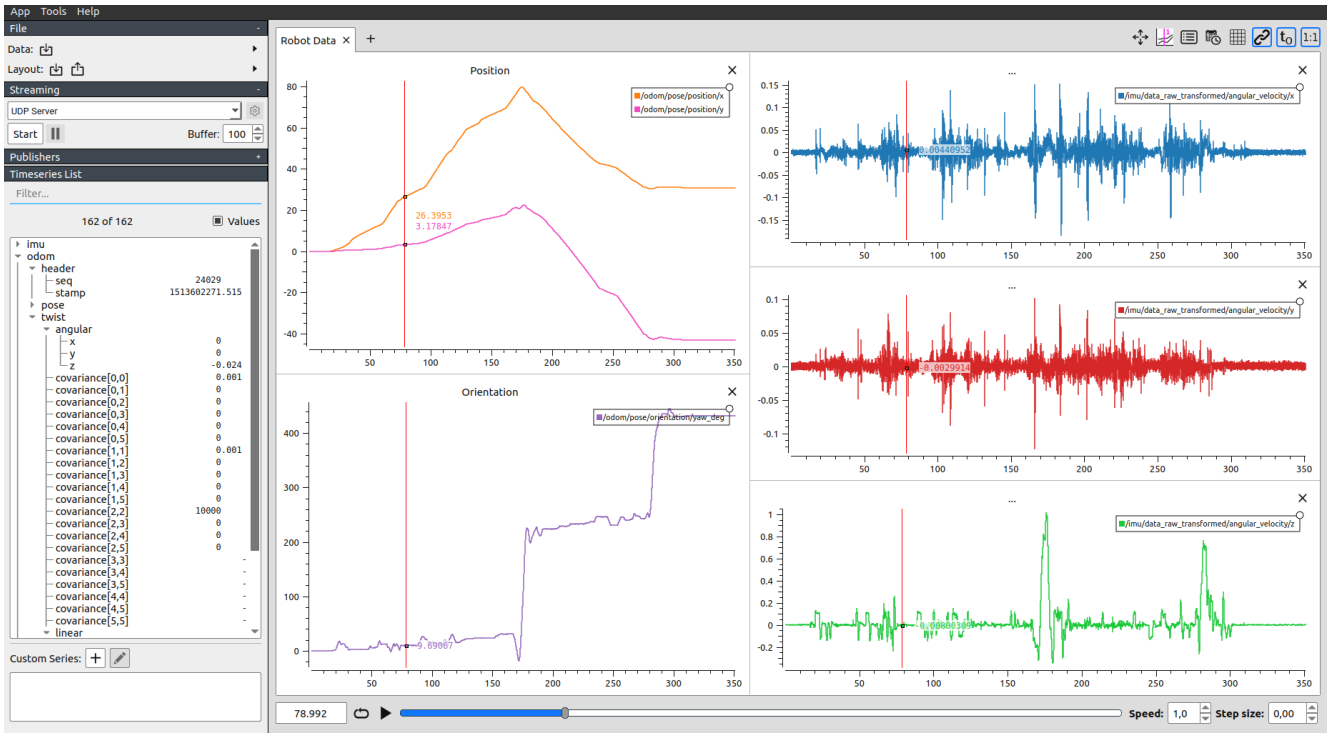
- 3D visualization environment
- Different data can be shown
  - Laser scan data, map, ...



Source: 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

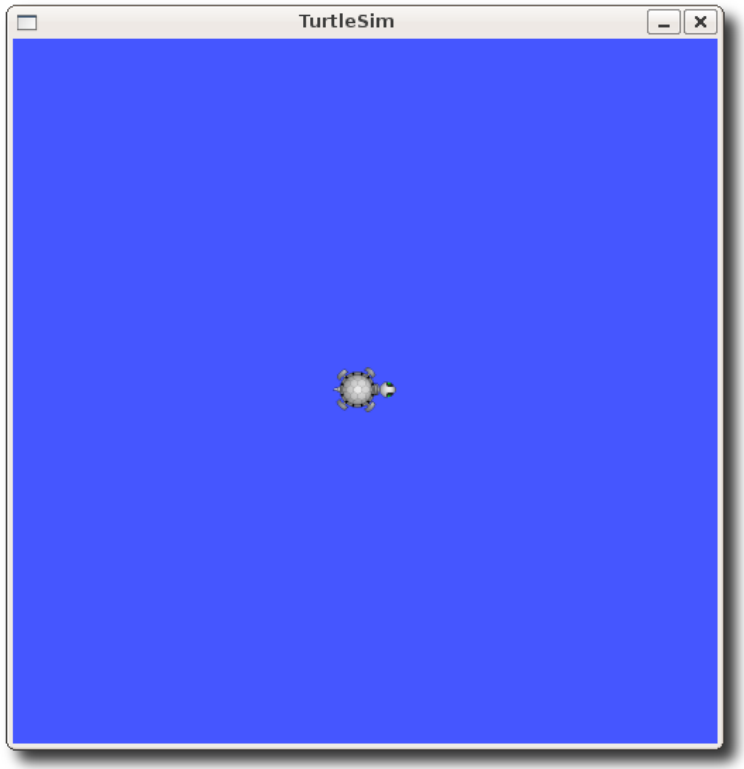
- Turtlesim
  - ROS learning tool
- Gazebo
  - ROS simulator
- Webots
  - Robotics simulator

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

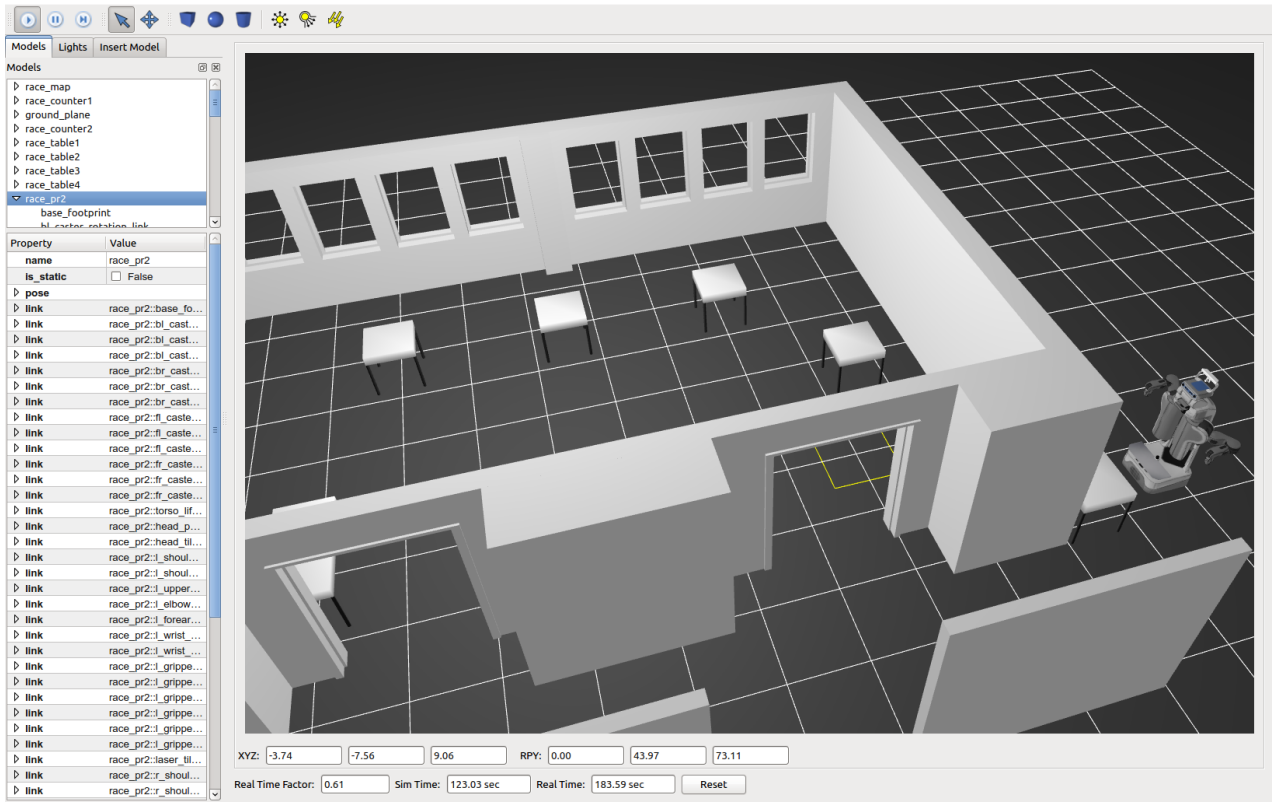
- Learning platform
- 2D turtle
  - move
  - turn
  - draw
- Communication
- ROS structure



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

Gazebo

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



Source: Lasse Einig

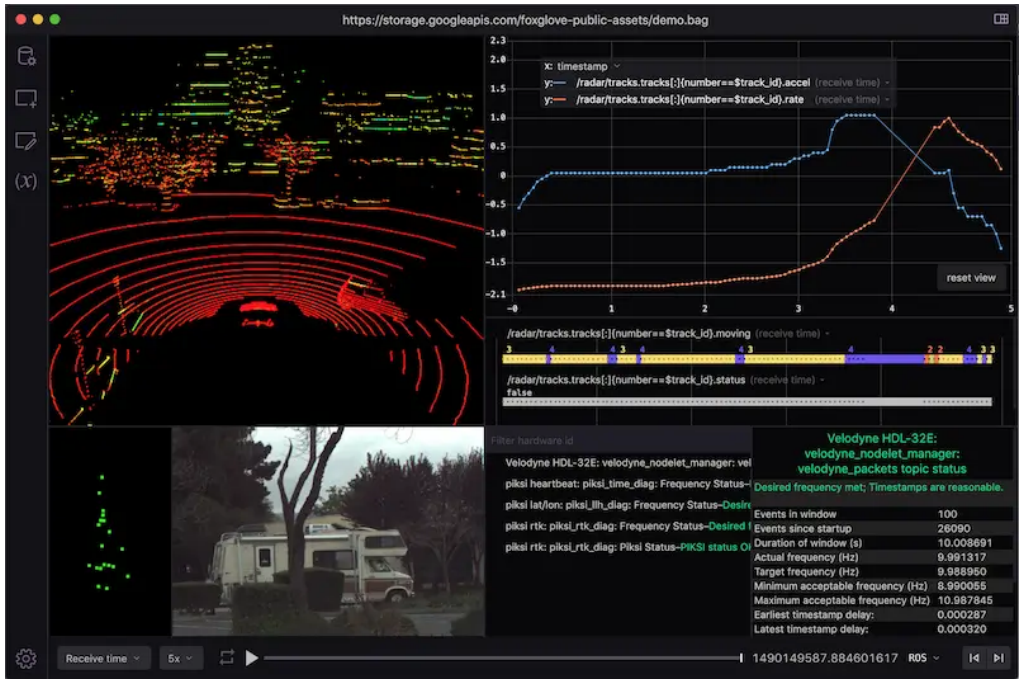
Webots/Webots ROS 2

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

Source: Jonas Hagge

Foxglove Studio (ROS 2)

- Modern visualization and debugging tool for ROS 2 (bags & live topics)
- Dashboards with panels: Time Series, Image, 3D, Map, Tables, Console



Learning Resources (ROS 2)

- ROS 2 Humble Documentation
- TurtleBot Official Site
- The Construct: ROS 2 Courses
- Foxglove Studio Tutorials for ROS 2