Introduction to Robot Operating System (ROS)

Mayank Mittal

May 22, 2018

Outline

- What is ROS?
- ROS Communication Layer
  - ROS Master
  - ROS Nodes
  - Topics, Services, Actions
- ROS Ecosystem
  - ROS Packages
  - *Catkin* build system
- Libraries/Tools in ROS
  - Point Cloud (PCL Library)
  - Coordinate Transformation (Tf Library)

Camera

IMU

Laser scanner

Motor and Encoder

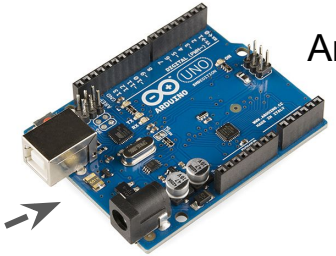
Robot

How to integrate sensors and actuators in your robot software suite?

GPS

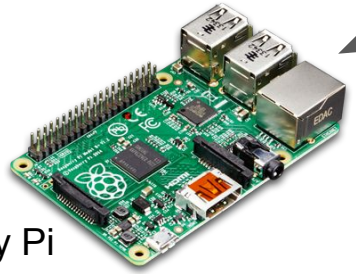
Intel NUC

Arduino

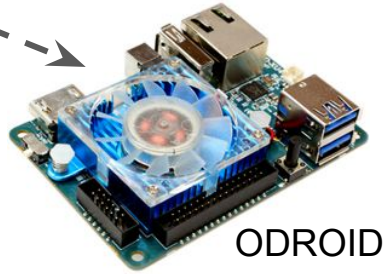


Robot

Raspberry Pi



How to interface the hardware using microprocessors and microcontrollers?



ODROID XU4

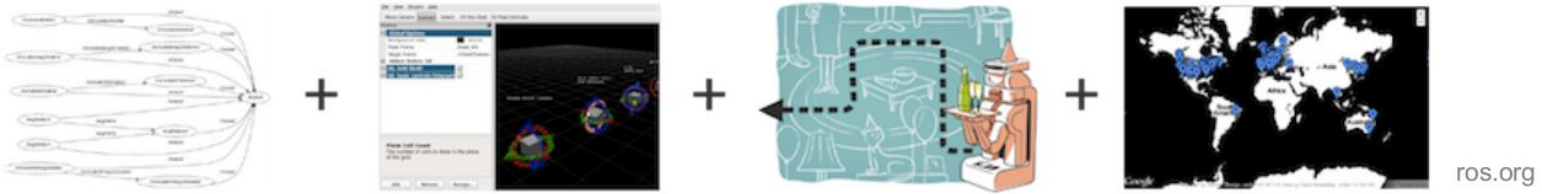
What is ROS?

- A “meta” operating system for robots
- A collection of packaging, software building tools
- An architecture for distributed interprocess/ inter-machine communication and configuration
- Development tools for system runtime and data analysis
- A language-independent architecture (c++, python, lisp, java, and more)

Slide Credit: Lorenz Mösenlechner, TU Munich

What is ROS?

ROS = Robot Operating System



Plumbing	Tools	Capabilities	Ecosystem
▪ Process management ▪ Inter-process communication ▪ Device drivers	▪ Simulation ▪ Visualization ▪ Graphical user interface ▪ Data logging	▪ Control ▪ Planning ▪ Perception ▪ Mapping ▪ Manipulation	▪ Package organization ▪ Software distribution ▪ Documentation ▪ Tutorials

Slide Credit: Marco Hutter, ETH Zurich

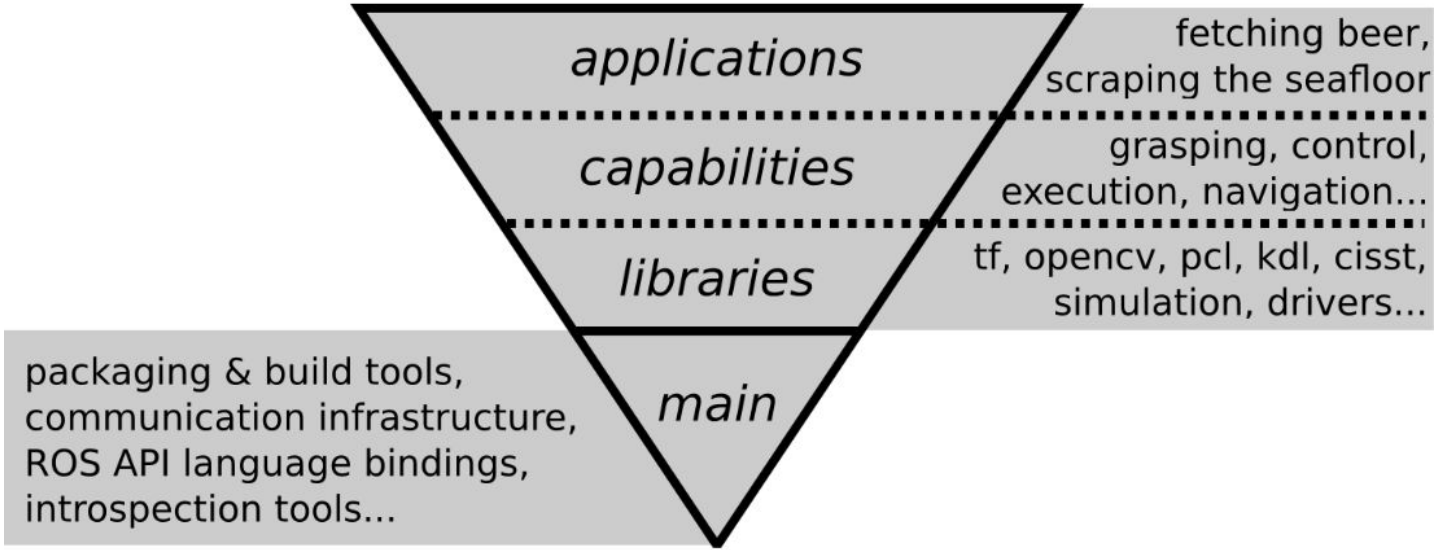
What is ROS not?

- An actual operating system
- A programming language
- A programming environment / IDE
- A hard real-time architecture

Slide Credit: Lorenz Mösenlechner, TU Munich

What does ROS get you?

All levels of development



Slide Credit: Lorenz Mösenlechner, TU Munich

ROS Communication Layer : ROS Core

- **ROS Master**
  - Centralized Communication Server based on XML and RPC
  - Negotiates the communication connections
  - Registers and looks up names for ROS graph resources
- **Parameter Server**
  - Stores persistent configuration parameters and other arbitrary data.
- **`*rosout*`**
  - Network based `*stdout*` for human readable messages.

Slide Credit: Lorenz Mösenlechner, TU Munich

ROS Communication Layer : Graph Resources

- **Nodes**
  - Processes distributed over the network.
  - Serves as source and sink for the data sent over the network
- **Parameters**
  - Persistent data such as configuration and initialization settings, i.e the data stored on the parameter server. e.g camera configuration
- **Topics**
  - Asynchronous many-to-many communication stream
- **Services**
  - Synchronous one-to-many network based functions

Slide Credit: Lorenz Mösenlechner, TU Munich

ROS Communication Protocols: Connecting Nodes

- **ROS Topics**
  - Asynchronous “stream-like” communication
  - Strongly-typed (ROS .msg spec)
  - Can have one or more publishers
  - Can have one or more subscribers
- **ROS Services**
  - Synchronous “function-call-like” communication
  - Strongly-typed (ROS .srv spec)
  - Can have only one server
  - Can have one or more clients
- **Actions**
  - Built on top of topics
  - Long running processes
  - Cancellation

Slide Credit: Lorenz Mösenlechner, TU Munich

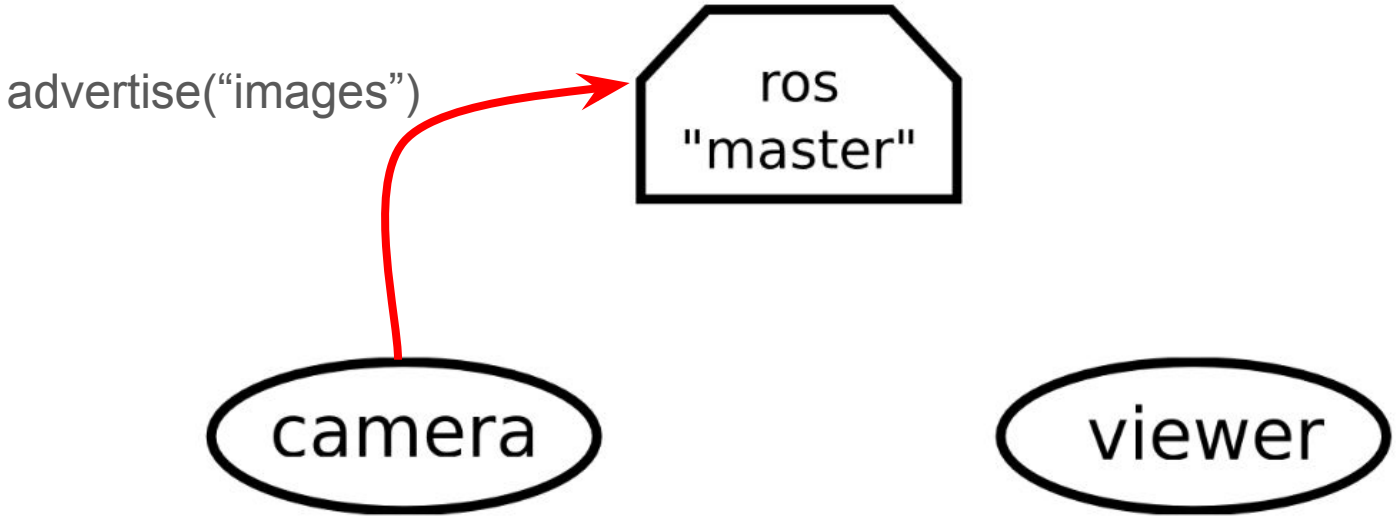
Asynchronous Distributed Communication

Interfaces with the camera hardware and reads the data transmitted by the sensor

Used to display images

Image Courtesy: Lorenz Mösenlechner, TU Munich

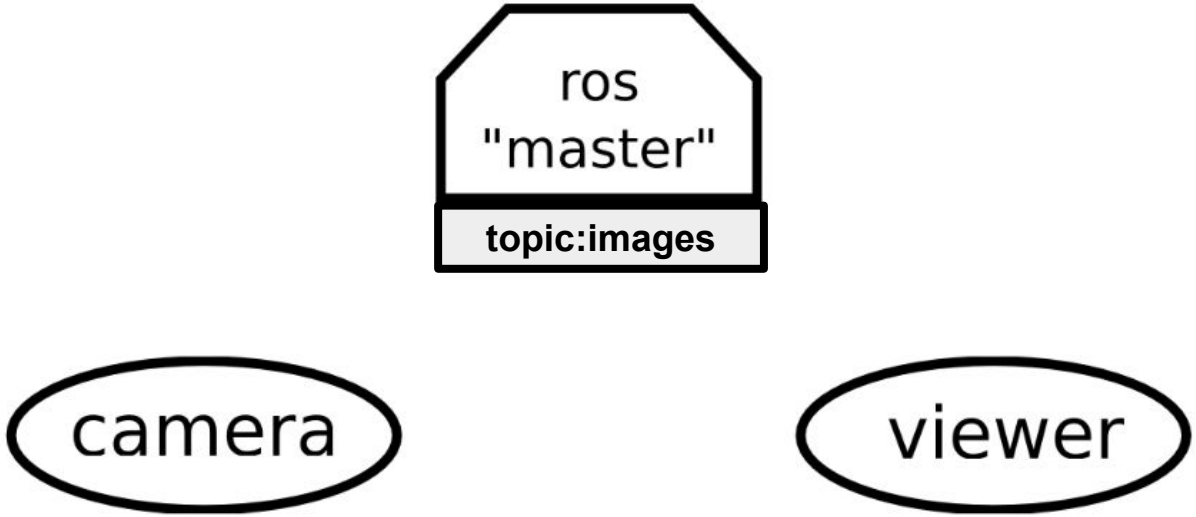
Asynchronous Distributed Communication



camera node is run. It starts advertising the data it has received

Image Courtesy: Lorenz Mösenlechner, TU Munich

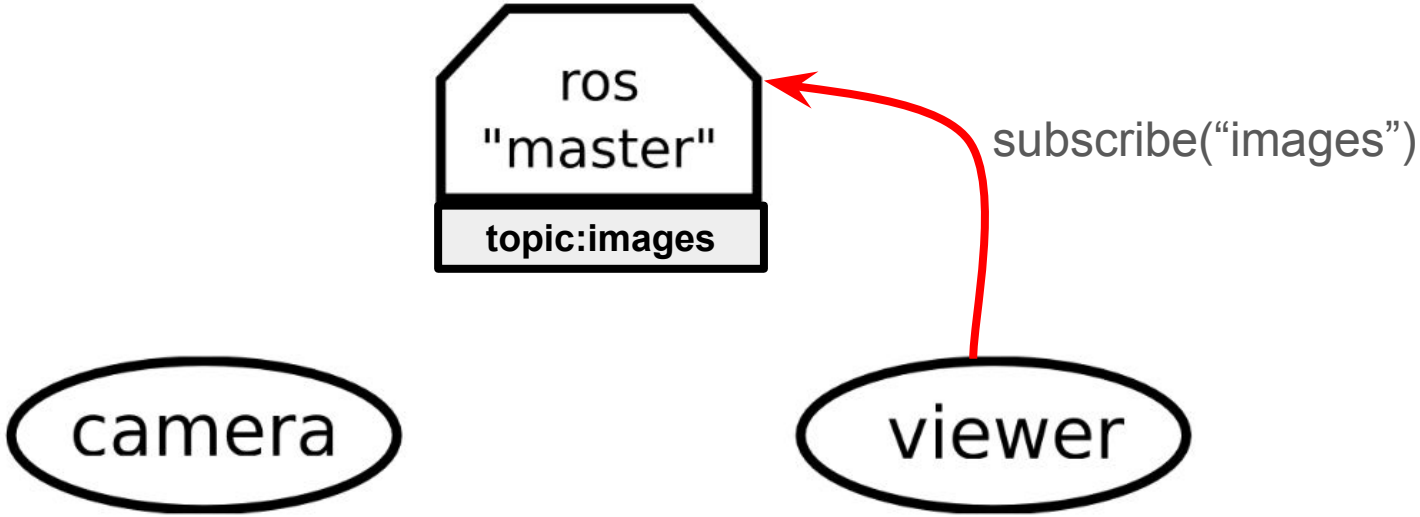
Asynchronous Distributed Communication



master registers the topic with name **images**

Image Courtesy: Lorenz Mösenlechner, TU Munich

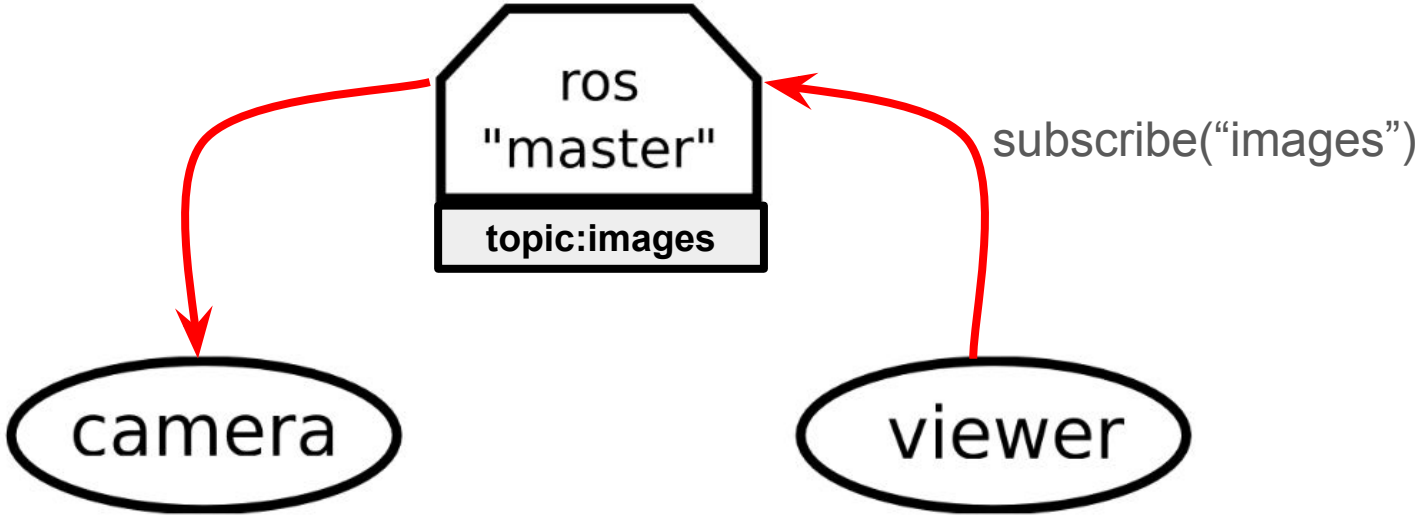
Asynchronous Distributed Communication



viewer node is run. It asks for data being published in topic with name **images**

Image Courtesy: Lorenz Mösenlechner, TU Munich

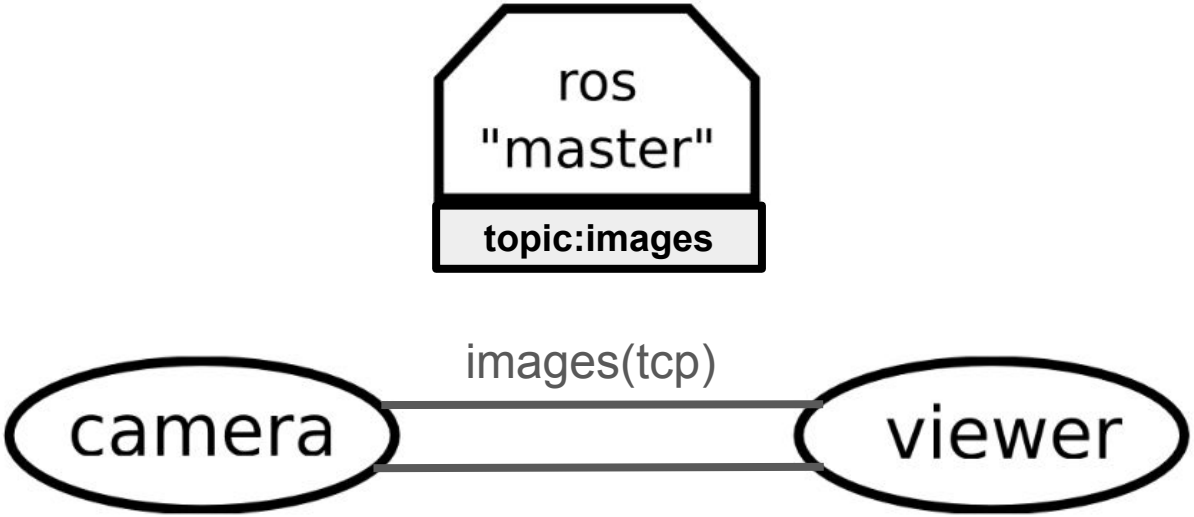
Asynchronous Distributed Communication



master ‘connects’ the *viewer* node to the *camera* node.

Image Courtesy: Lorenz Mösenlechner, TU Munich

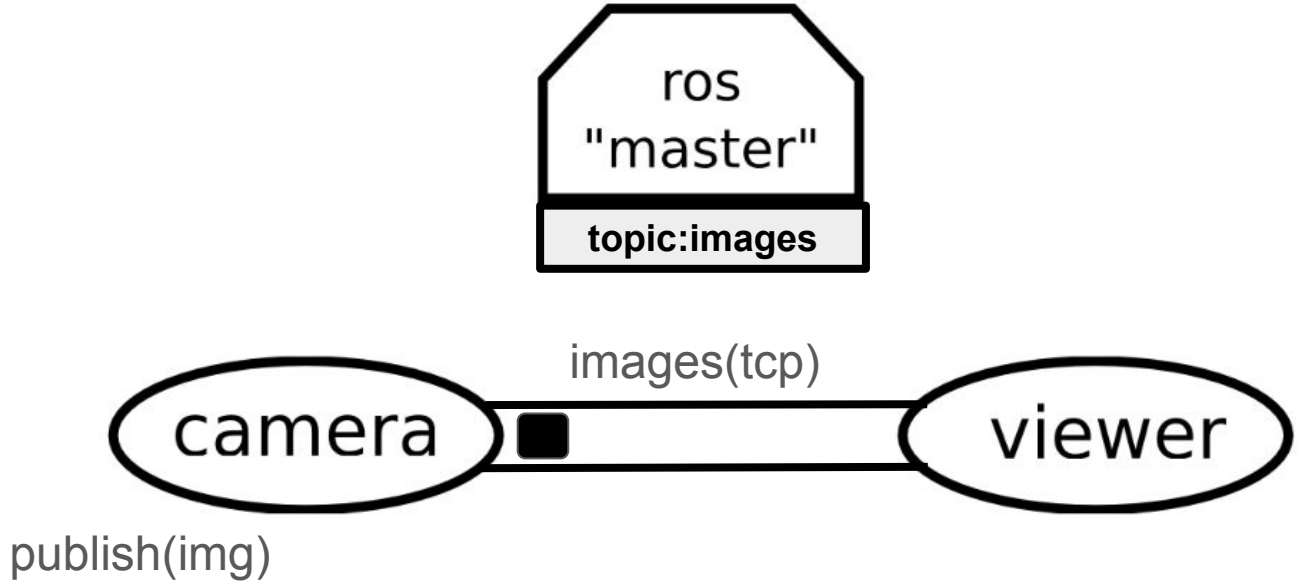
Asynchronous Distributed Communication



master 'connects' the *viewer* node to the *camera* node.

Image Courtesy: Lorenz Mösenlechner, TU Munich

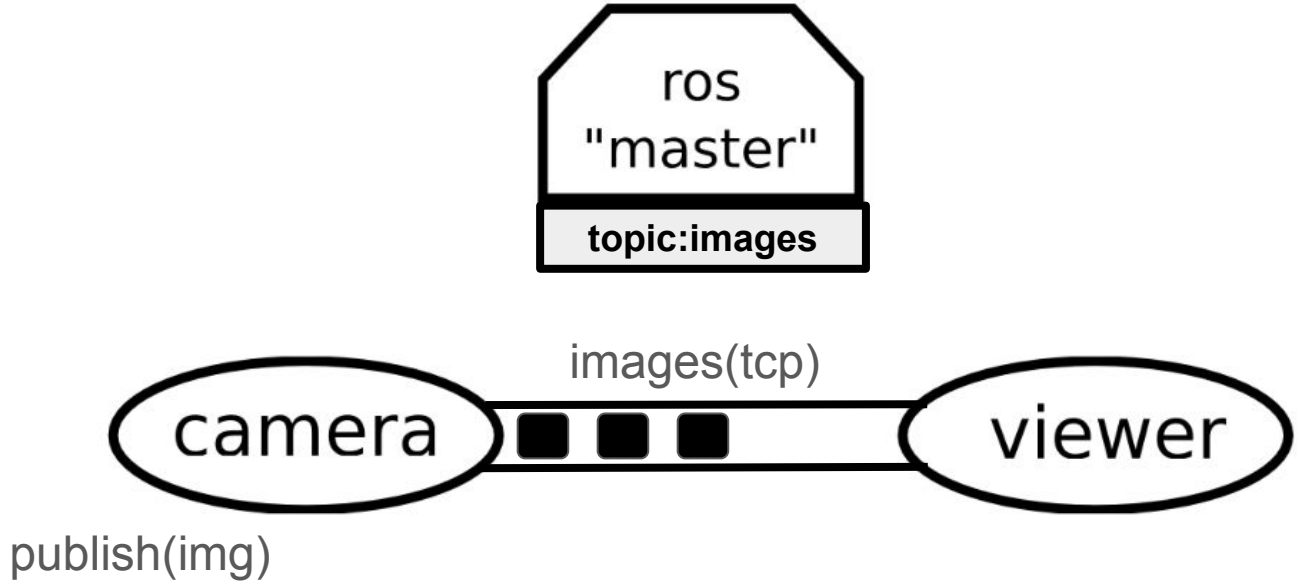
Asynchronous Distributed Communication



camera node sends the data to the *viewer* node using TCP/IP based protocol

Image Courtesy: Lorenz Mösenlechner, TU Munich

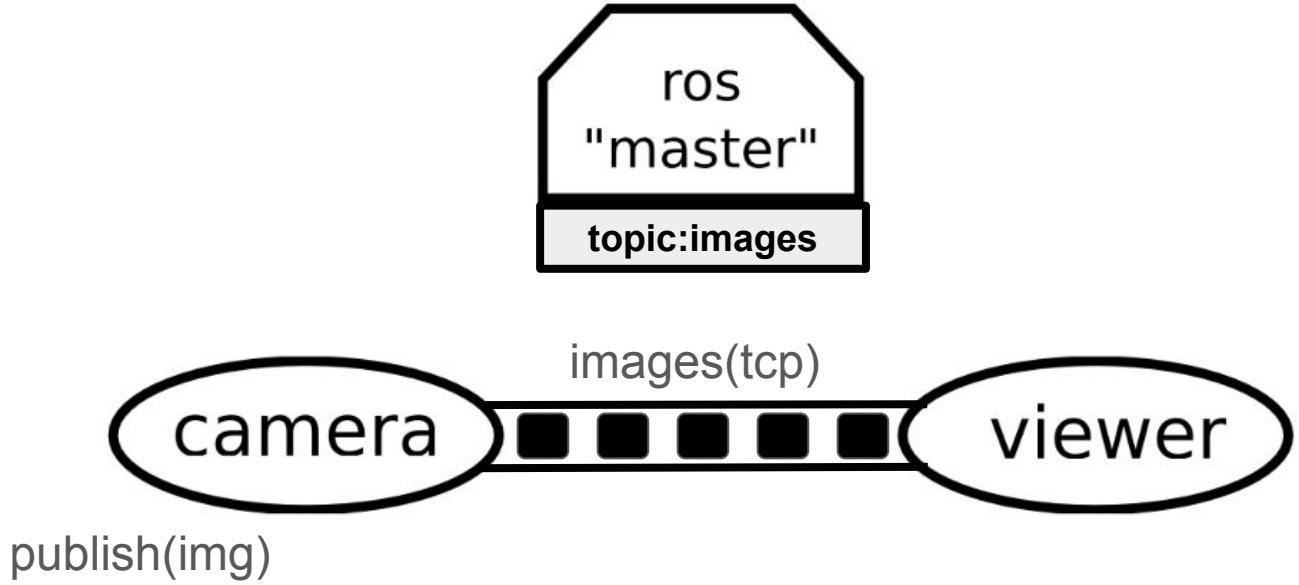
Asynchronous Distributed Communication



camera node sends the data to the *viewer* node using TCP/IP based protocol

Image Courtesy: Lorenz Mösenlechner, TU Munich

Asynchronous Distributed Communication



camera node sends the data to the *viewer* node using TCP/IP based protocol

Image Courtesy: Lorenz Mösenlechner, TU Munich

ROS Master

- Manages the communication between nodes
- Every node registers at startup with the master

Start a master with

```
$ roscore
```

More info:
http://wiki.ros.org/Master

Slide Credit: Marco Hutter, ETH Zurich

ROS Nodes

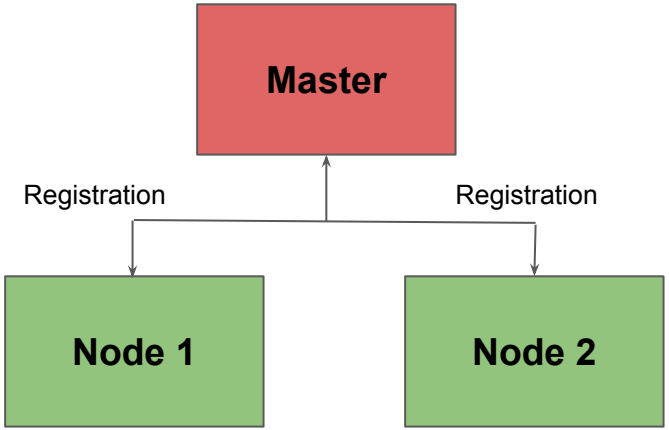
- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in packages

Run a node with

```
$ rosrun package_name node_name
```

See active nodes with

```
$ rosnode list
```

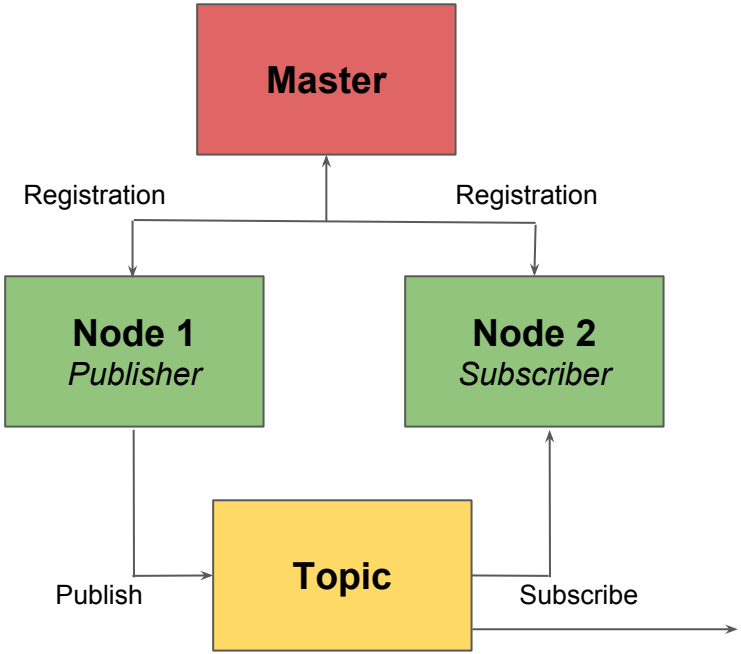


More info:
http://wiki.ros.org/rosnode

Slide Credit: Marco Hutter, ETH Zurich

ROS Topics

- Nodes communicate over topics
  - Nodes can publish or subscribe to a topic
  - Typically, 1 publisher and n subscribers
- Topic is name for stream of messages



See active topics with

```
$ rostopic list
```

Subscribe and print the contents of a topic with

```
$ rostopic echo /topic
```

More info:
http://wiki.ros.org/rostopic

Slide Credit: Marco Hutter, ETH Zurich

ROS Messages

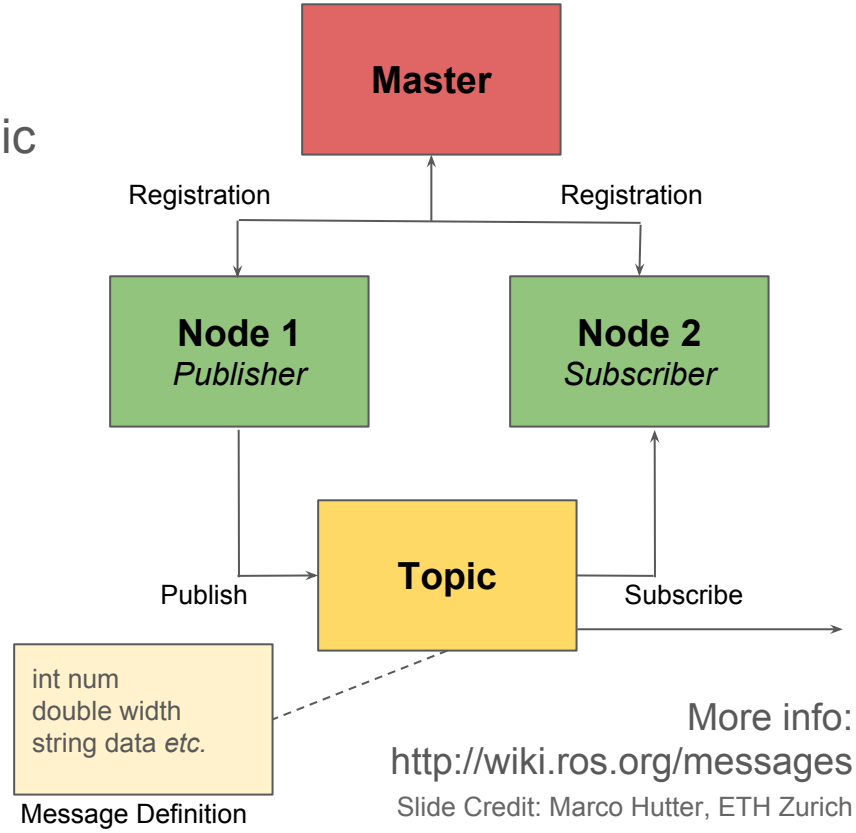
- Data structure defining the type of a topic
  - Comprised of a nested structure of integers, floats, strings etc. and arrays of objects
- Defined in *.msg files

See the type of a topic

```
$ rostopic type /topic
```

Publish a message to a topic

```
$ rostopic pub /topic type args
```



More info:
http://wiki.ros.org/messages

Slide Credit: Marco Hutter, ETH Zurich

ROS Messages

geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

sensor_msgs/Image.msg

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
 uint32 seq
 time stamp
 string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
     float64 x
     float64 y
     float64 z
   geometry_msgs/Quaternion
orientation
     float64 x
     float64 y
     float64 z
     float64 w
```

More info:
http://wiki.ros.org/std_msgs

Slide Credit: Marco Hutter, ETH Zurich

ROS Services

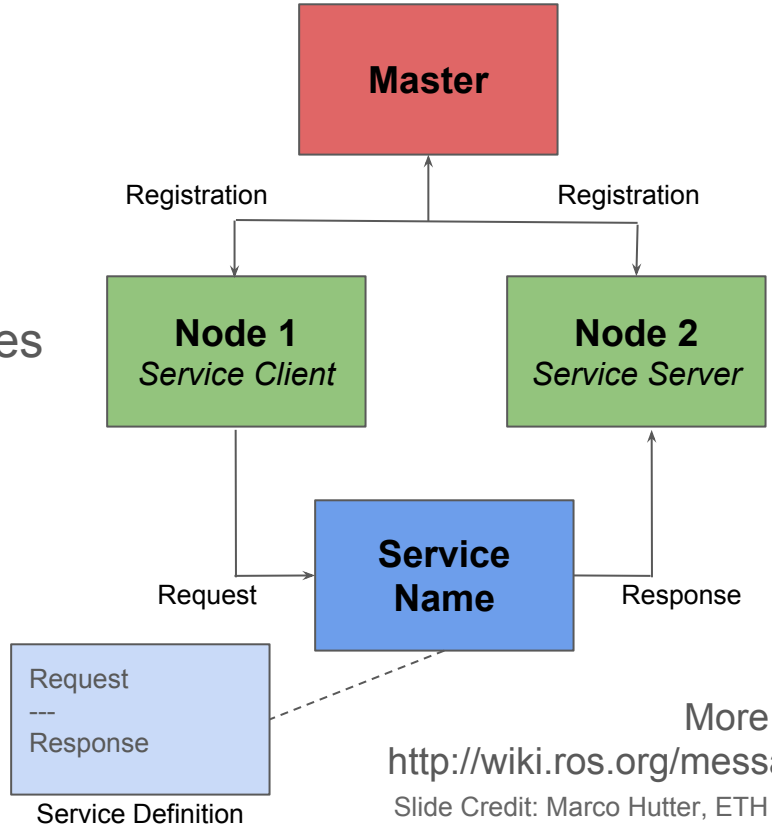
- Request/response communication between nodes is realized with services
  - The service server advertises the service
  - The service client accesses this service
- Similar in structure to messages, services are defined in *.srv files

List available services with

```
$ rosservice list
```

Show the type of a service

```
$ rosservice type /service_name
```

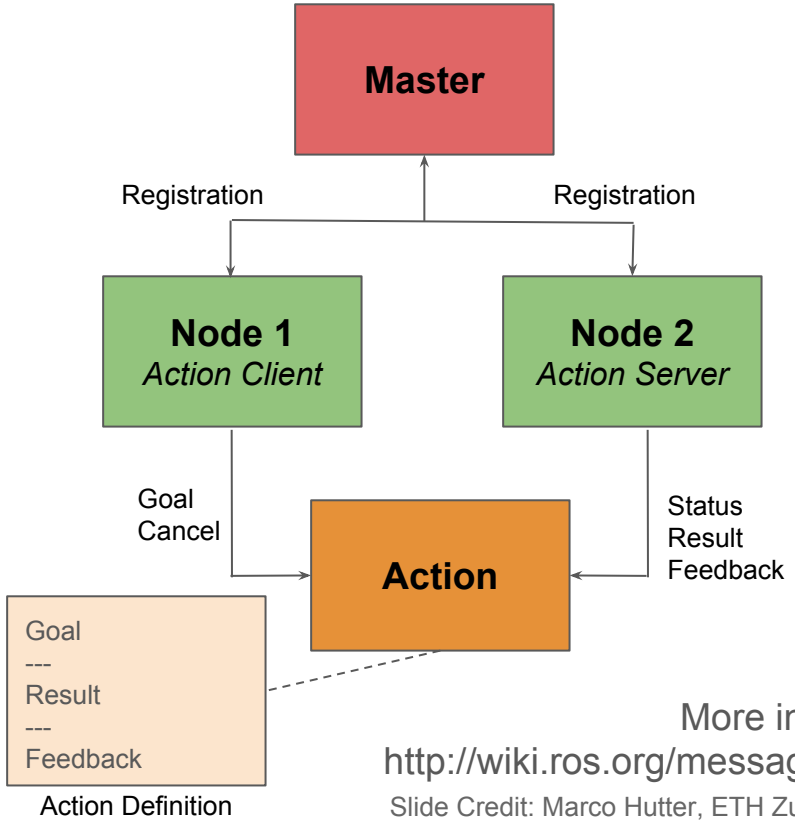


More info:
http://wiki.ros.org/messages

Slide Credit: Marco Hutter, ETH Zurich

ROS Action

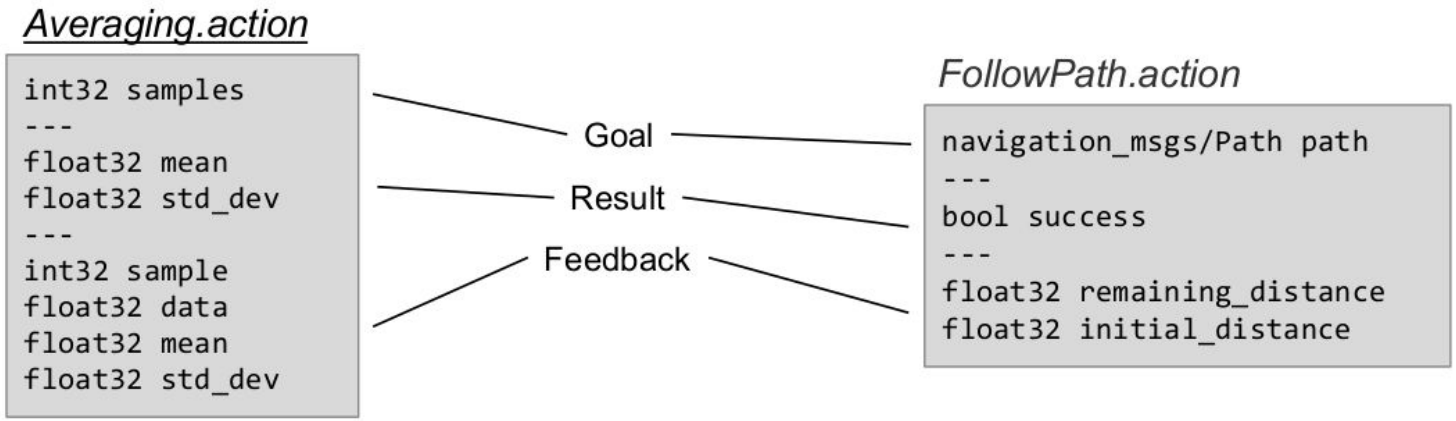
- Similar to service calls, but provide possibility to
  - Cancel the task (preempt)
  - Receive feedback on the progress
- Best way to implement interfaces to time- extended, goal-oriented behaviors
- Similar in structure to services, action are defined in *.action files
- Internally, actions are implemented with a set of topics



More info:
http://wiki.ros.org/messages

Slide Credit: Marco Hutter, ETH Zurich

ROS Action



More info:
http://wiki.ros.org/messages

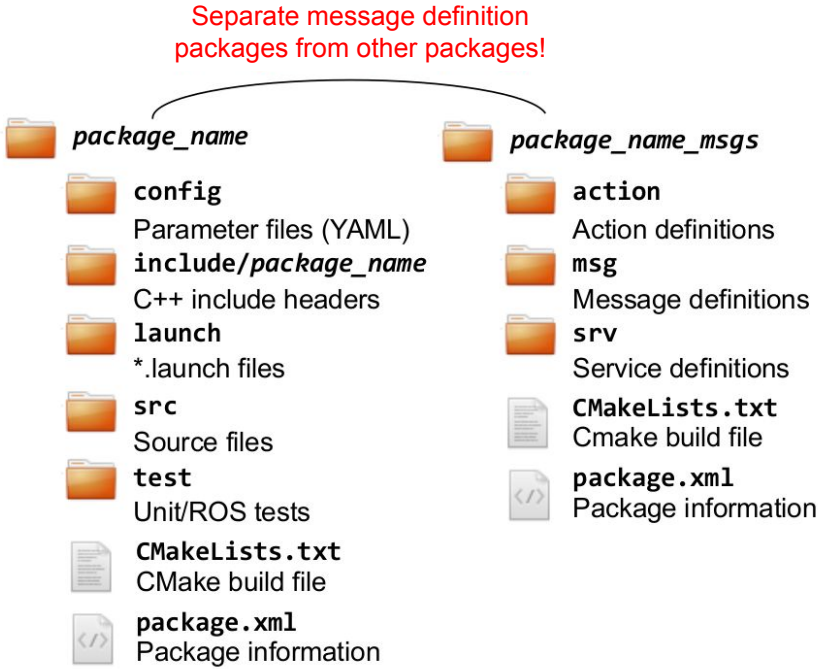
Slide Credit: Marco Hutter, ETH Zurich

ROS Packages

- ROS software is organized into packages, which can contain source code, launch files,configuration files, message definitions, data, and documentation
- A package that builds up on/requires other packages (e.g. message definitions), declares these as dependencies

To create a new package, use:

```
$ catkin_create_pkg package_name {dependencies}
```



More info:
http://wiki.ros.org/Packages

Slide Credit: Marco Hutter, ETH Zurich

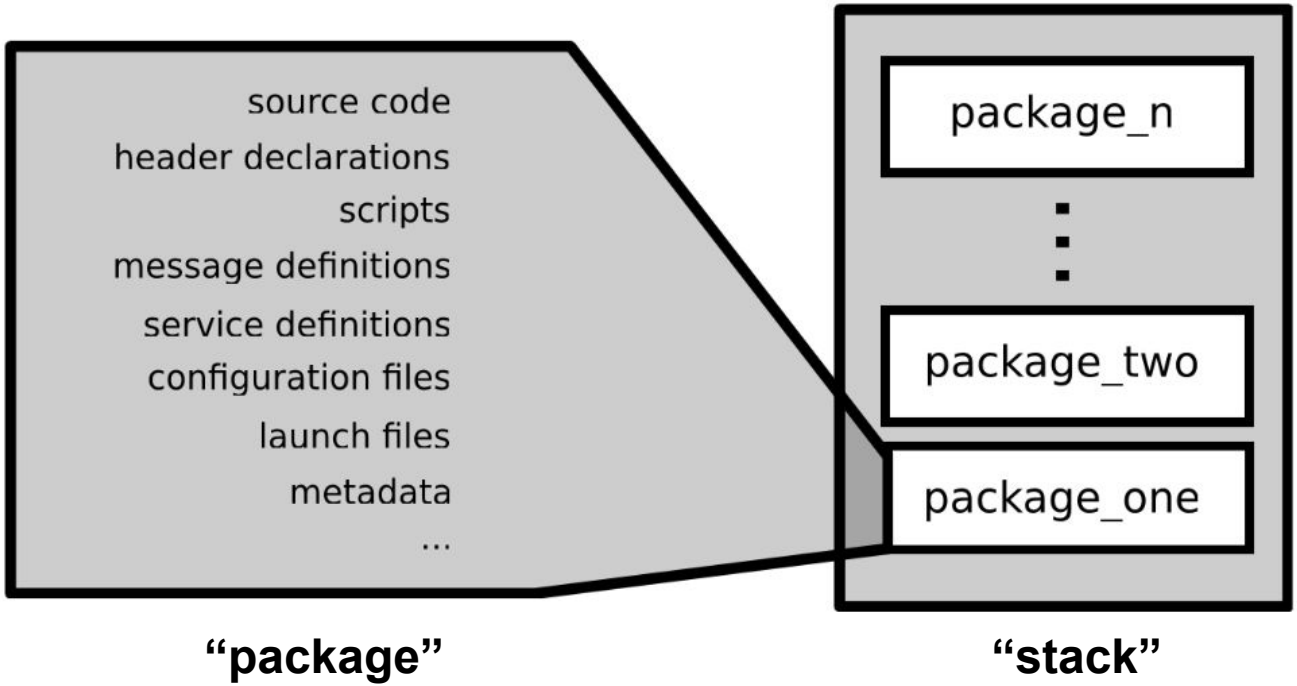
How to organize code in a ROS ecosystem?

ROS code is grouped at two different levels:

- **Packages:**
  - A named collection of software that is built and treated as an atomic dependency in the ROS build system.
- **Stacks:**
  - A named collection of packages for distribution.

Slide Credit: Lorenz Mösenlechner, TU Munich

How to organize code in a ROS ecosystem?



catkin Build System

- *catkin* is the ROS build system to generate executables, libraries, and interfaces
- The *catkin* command line tools are pre-installed in the provided installation.

Navigate to your catkin workspace with

```
$ cd ~/catkin_ws
```

Build a package with

```
$ catkin_make --package package_name
```

Whenever you build a new package, update your environment

```
$ source devel/setup.bash
```

Slide Credit: Lorenz Mösenlechner, TU Munich

catkin Build System

The catkin workspace contains the following spaces

Work here

The source space contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

Don't touch

The build space is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

Don't touch

The development (devel) space is where built targets are placed (prior to being installed).

Slide Credit: Marco Hutter, ETH Zurich

ROS Launch

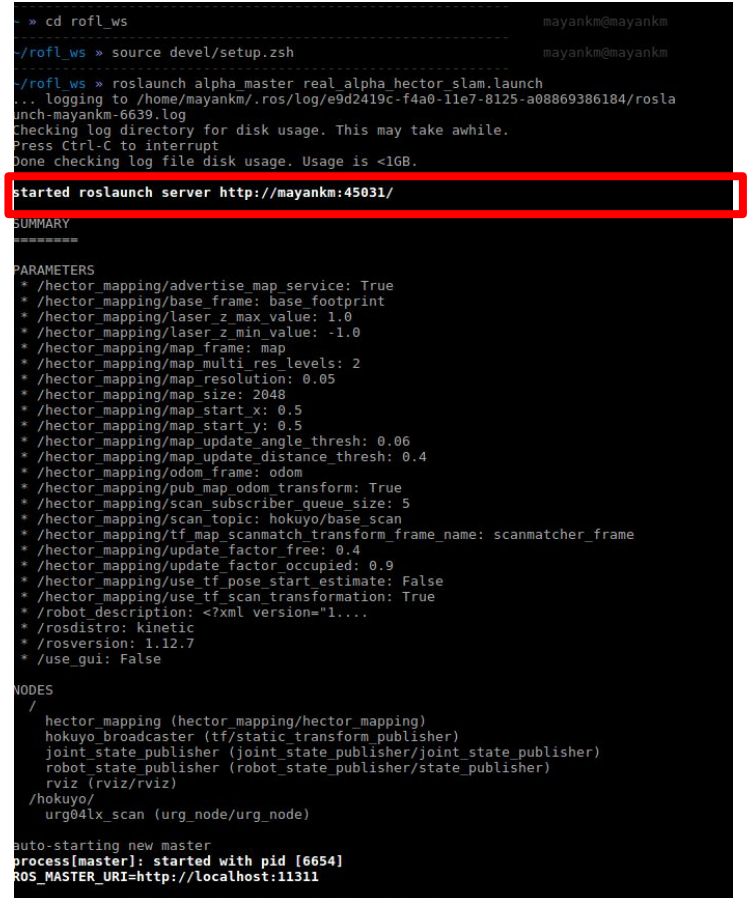
- launch is a tool for launching multiple nodes (as well as setting parameters)
- Are written in XML as *.launch files
- If not yet running, launch automatically starts a roscore

Start a launch file from a package with

```
$ roslaunch package_name file_name.launch
```

More info:
http://wiki.ros.org/roslaunch

Slide Credit: Marco Hutter, ETH Zurich



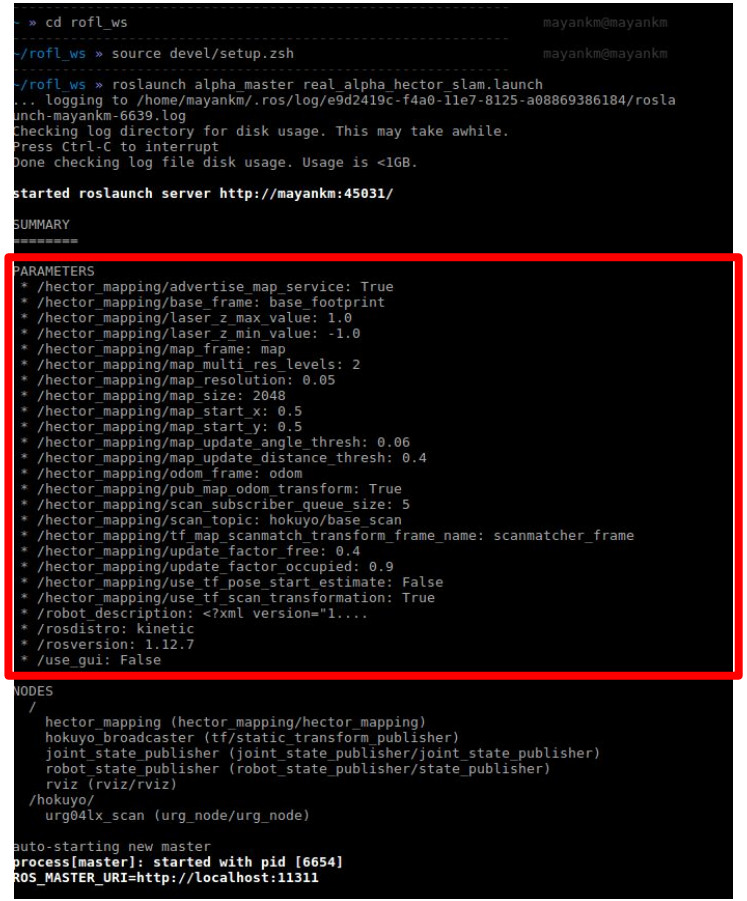
ROS Parameter Server

- Nodes use the parameter server to store and retrieve parameters at runtime
- Best used for static data such as configuration parameters
- Parameters can be defined in launch files or separate YAML files

List all parameters with

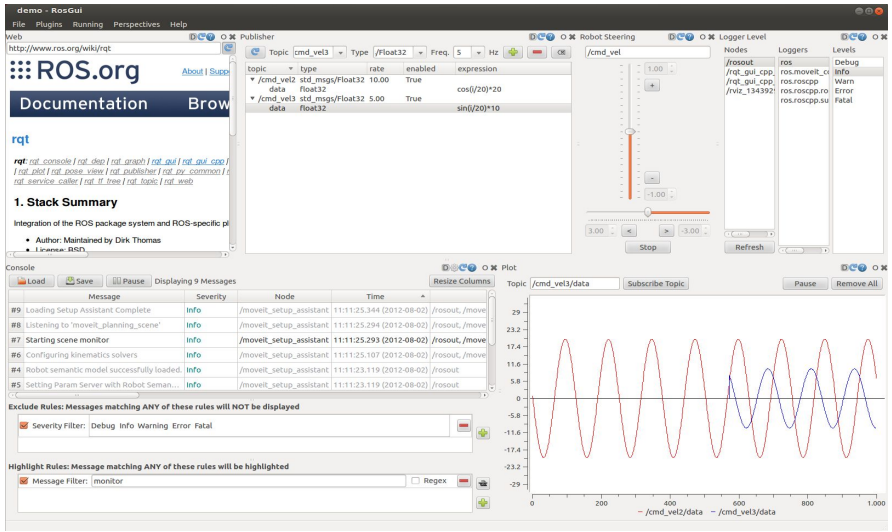
```
$ rosparam list
```

More info:
http://wiki.ros.org/rosparam

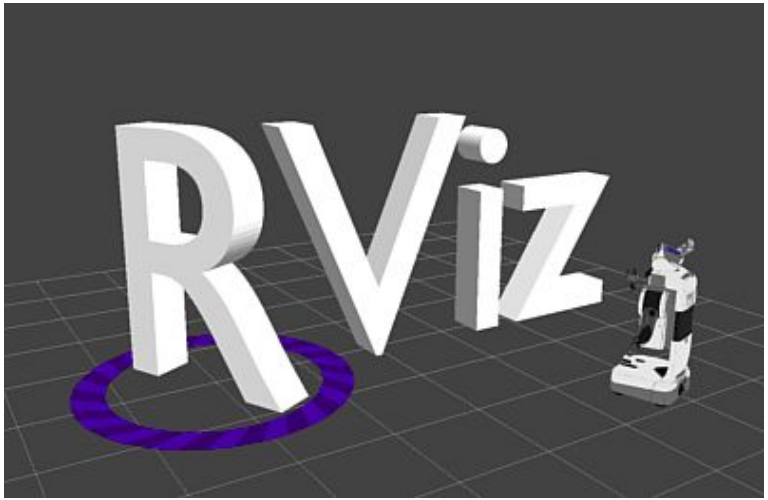


ROS GUI Tools

rqt : A QT based GUI developed for ROS



rviz : Powerful tool for 3D Visualization



(demo in next class)

More info:
http://wiki.ros.org/rqt

ROS Time

- Normally, ROS uses the PC's system clock as time source (wall time)
- For simulations or playback of logged data, it is convenient to work with a simulated time (pause, slow-down etc.)
- To work with a simulated clock:
  - Set the `/use_sim_time` parameter

```
$ rosparam set use_sim_time true
```

  - Publish the time on the topic /clock from
    - Gazebo (enabled by default)
    - ROS bag (use option --clock)

- To take advantage of the simulated time, you should always use the ROS Time APIs:
  - **ros::Time**

```
ros::Time begin = ros::Time::now();
double secs = begin.toSec();
```

  - **ros::Duration**

```
ros::Duration duration(0.5); // 0.5s
```

More info:
http://wiki.ros.org/Clock

Slide Credit: Marco Hutter, ETH Zurich

ROS Bags

- A bag is a format for storing message data
- Binary format with file extension *.bag
- Suited for logging and recording datasets for later visualization and analysis

Record all topics in a bag

```
$ rosbag record --all
```

Record given topics

```
$ rosbag record topic_1 topic_2 topic_3
```

Show information about a bag

```
$ rosbag info bag_name.bag
```

Record given topics

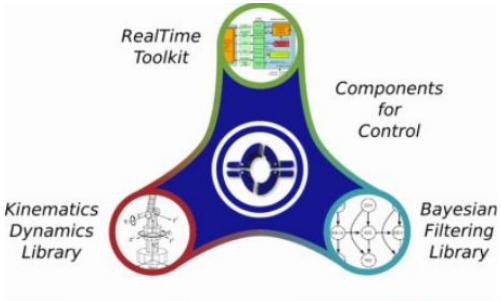
```
$ rosbag play [options] bag_name.bag
```

--rate=factor	Publish rate factor
--clock	Publish the clock time (set param use_sim_time to true)
--loop	Loop playback

More info:
http://wiki.ros.org/Clock

Slide Credit: Marco Hutter, ETH Zurich

Libraries/Tools available with ROS



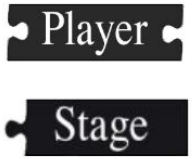


Image Courtesy: Open Source Robotics Foundation

Homework

- Install ROS Kinetic on your laptop (Ubuntu 16.04LTS)
  - Instructions: http://wiki.ros.org/kinetic/Installation/Ubuntu
  - Alternate Option:
    - Download Shell Script (available here)
    - Run on terminal: `./install_ROS kinetic`
- Attempt tutorials on Robot Operating System (available online)

References

- Slides from lectures on ‘Programming for Robotics’ by ETH Zurich
- A Gentle Introduction to ROS, Jason M. O'Kane. Oct 2013 (available online)
- Berger, E., Conley, K., Faust, J., Foote, T., Gerkey, B.P., Leibs, J., Ng, A.Y., Quigley, M., & Wheeler, R. (2009). “**ROS: an open-source Robot Operating System”.**