

Linux Systems and Open Source Software

Robot Operating System (ROS)

Chia-Heng Tu
Dept. of Computer Science and Information Engineering
National Cheng Kung University
Fall 2022



Outline

- Robot Operating System (ROS)
  - Publish–Subscribe Model
  - ROS Computation Graph
- Communication Types in ROS
- ROS Files Organization
- Visualization and Debugging
- Rosbridge



Robot Operating System (ROS)

- Is a flexible framework for developing robot applications
- Is a collection of tools, libraries, and conventions
  - that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms

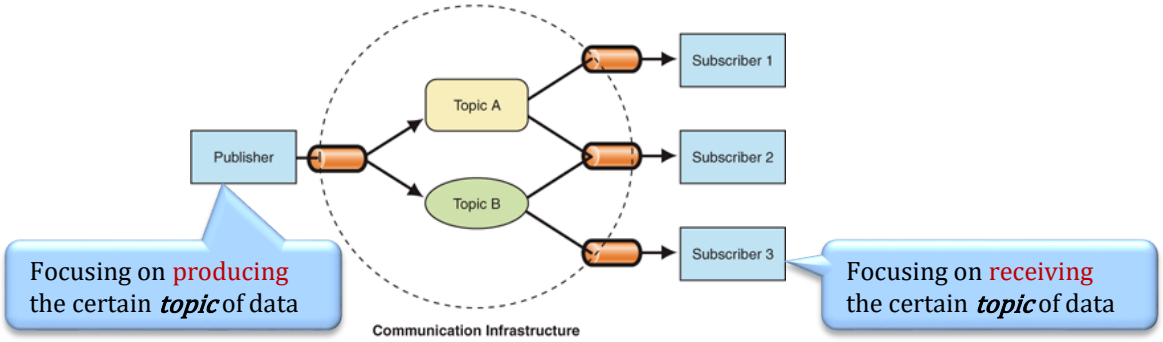
Courtesy of https://www.ros.org/about-ros/



Publish–Subscribe Model

- ***Publish–Subscribe*** is a ***messaging pattern*** from software architecture perspective
- Senders (**publishers**) categorize published messages into classes without knowledge of which receivers (**subscribers**)
  - Publishers do not program the messages to be sent directly to specific receivers
- Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers

An illustration of the publish-subscribe pattern.



Courtesy of https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern
Courtesy of https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649664(v=pandp.10)?redirectedfrom=MSDN#applying-publishsubscribe

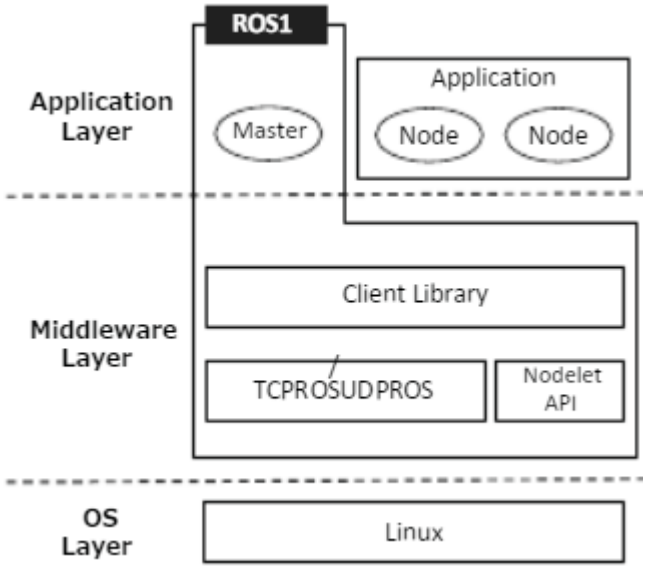


Software Architecture of ROS1

- ROS1 is mainly running on Linux-based systems

- It is a multi-layer software architecture
- **Application layer**
  - for application developments, e.g., navigation
- Middleware layer
  - exposes the ***programming interfaces*** for application development
  - provides infrastructures for ***communications***, e.g., data transport protocols (TCP/ROS, UPD/ROS)
- OS layer
  - provides the basic functionalities for ROS

ROS multi-layer software architecture.



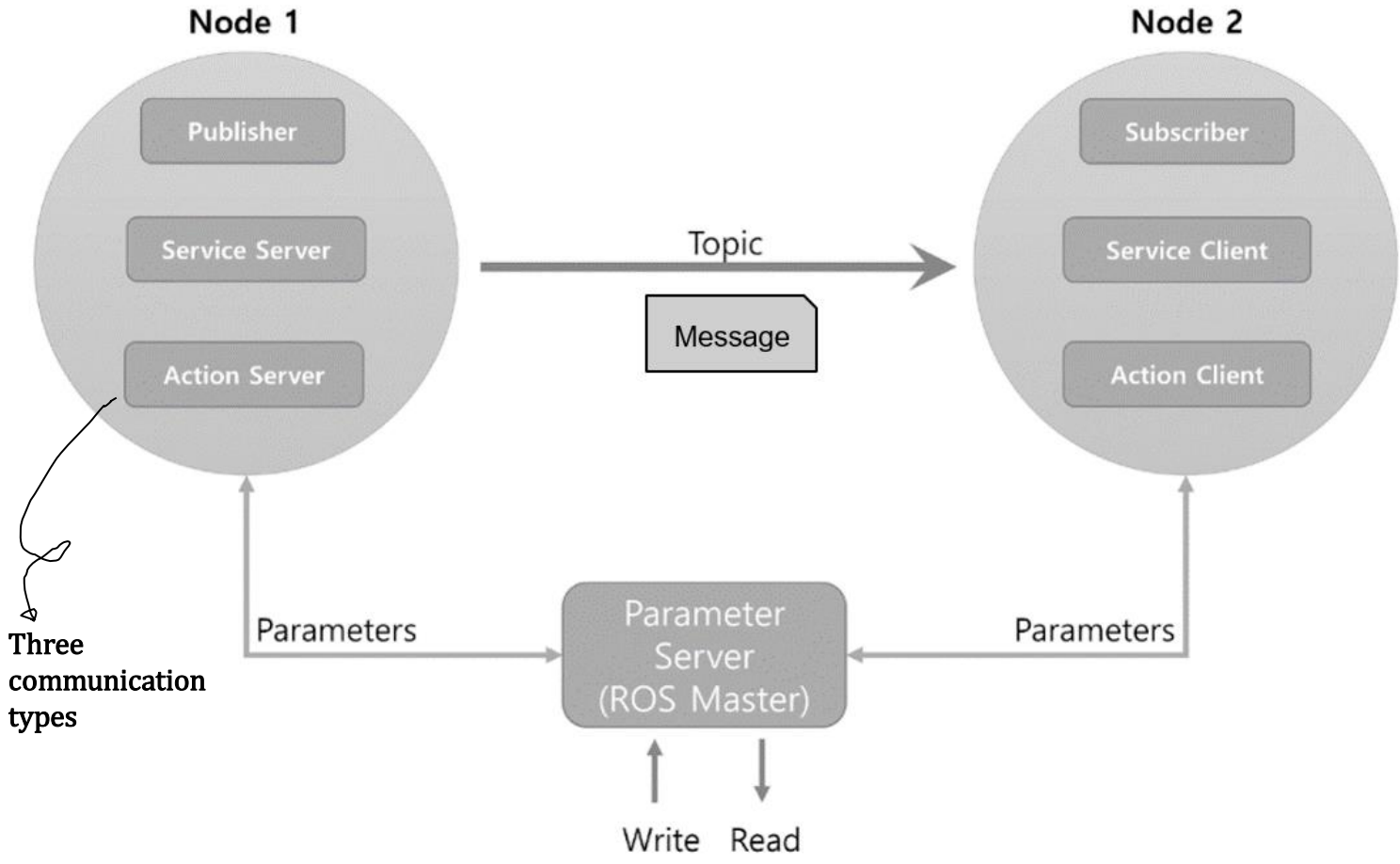


ROS Computation Graph

- Nodes
- Master
  - Parameter Server
- Messages
- Topics
- Bags



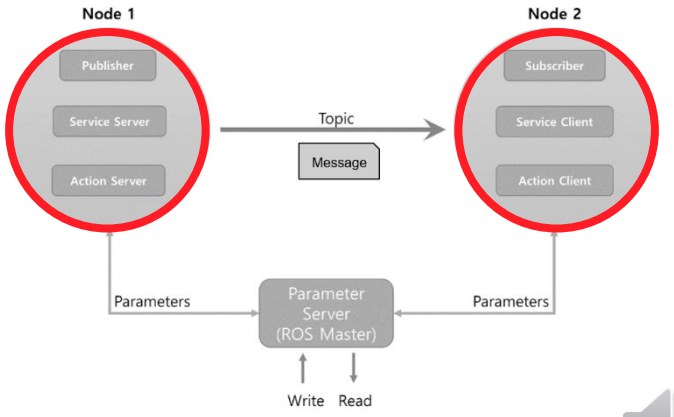
Publish–Subscribe Model in ROS1





Nodes

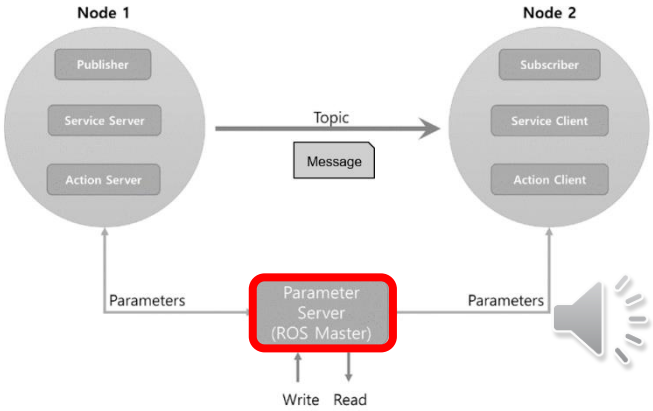
- A *node* is a software **process** performing computations
- Benefits
  - fault tolerance as crashes are isolated to individual nodes
  - Code complexity is reduced in comparison to monolithic systems
- ***Client Library***
  - Python (rospy)
  - C++ (roscpp)





Master

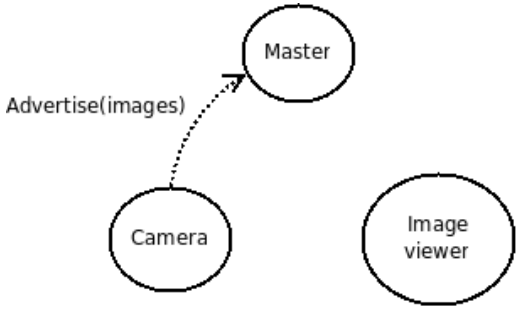
- Is to enable individual ROS nodes to **locate** one another
  - Naming and registration services to the rest of the nodes
  - Tracks publishers and subscribers to topics
  - Provides an XMLRPC-based API (Remote Procedure Call, RPC)
  - Provides the Parameter Server
- Once these nodes have located each other, **peer-to-peer communications** are performed between them



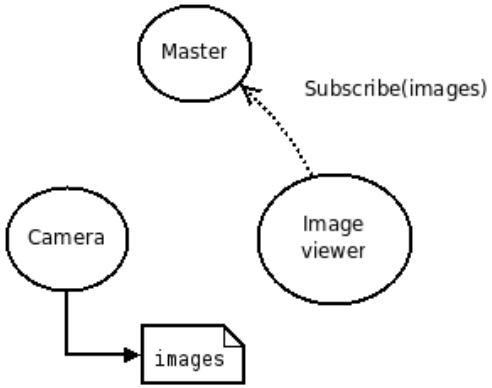
Courtesy of http://wiki.ros.org/Master



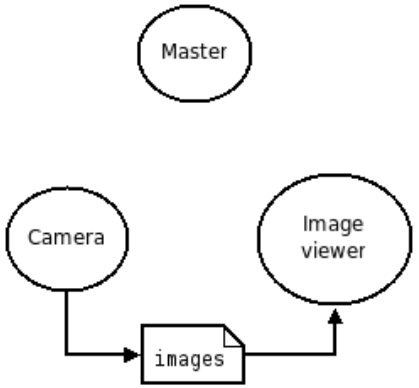
Example: Name Service Offered by Master



1. *Camera* notifies Master that it wants to publish images on the topic "images"



2. *Image viewer* wants to subscribe to the topic "images"



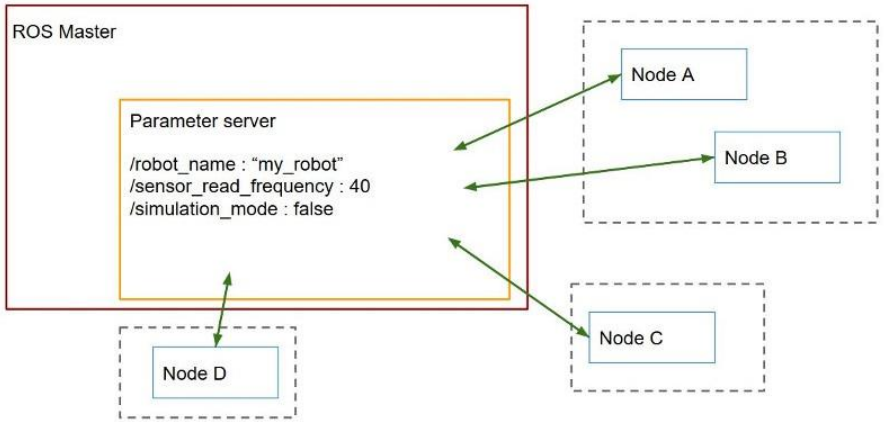
3. Now that the topic "images" has both a publisher and a subscriber, Master notifies *Camera* and *image viewer* about each others existence so that they can start transferring images to one another

Courtesy of http://wiki.ros.org/Master



Parameter Server

- A Parameter Server is created along with the creation of Master
  - It contains a ***dictionary***, accessible globally on the ROS environment
  - A ROS parameter is basically one of the ***shared variable*** stored in the parameter server



Example with 3 parameters:
- Robot name (string)
- Sensor read frequency (integer)
- Simulation mode flag (boolean)

Courtesy of https://roboticsbackend.com/what-is-a-ros-parameter/



Command: roscore

- Is a collection of **nodes** and **programs** that are pre-requisites of a ROS-based system
- **roscore** will bring up:
  - a ROS **Master**
  - a ROS **Parameter Server**
  - a **rosout** logging node
    - logs all messages into **rosout.log** in the ROS log directory

```
ros@ros-VirtualBox:~$ roscore
... logging to /home/ros/.ros/log/5c3f3888-e1d0-11ea-bee7-08002
7d70459/roslaunch-ros-VirtualBox-12588.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ros-VirtualBox:44705/
ros_comm version 1.14.6


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.6

NODES

auto-starting new master
process[master]: started with pid [12604]
ROS_MASTER_URI=http://ros-VirtualBox:11311/
```

Courtesy of http://wiki.ros.org/roscore



Messages

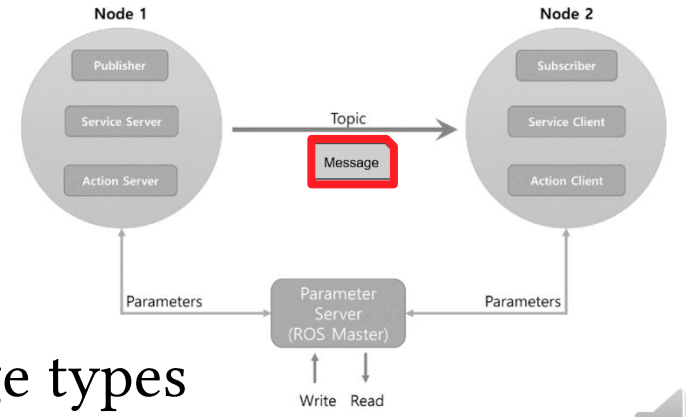
- Nodes communicate with each other by publishing ***messages*** to topics
  - A message is a simple data structure, comprising typed fields
- Message descriptions are stored in **.msg** files
  - Message Description Specification

```
fieldtype1 fieldname1
fieldtype2 fieldname2
```

  - Example:

```
int32 x
int32 y
```

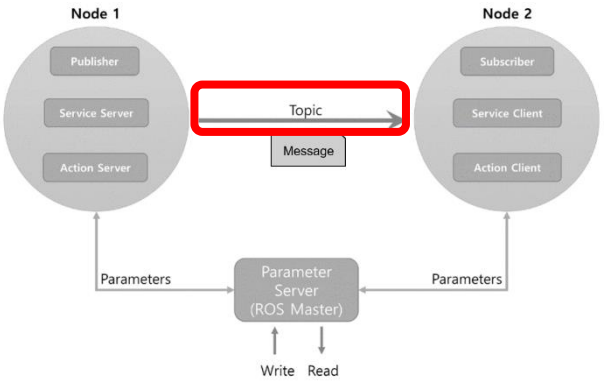
  - **rosmsg** (command line tool) displays information of ROS message types



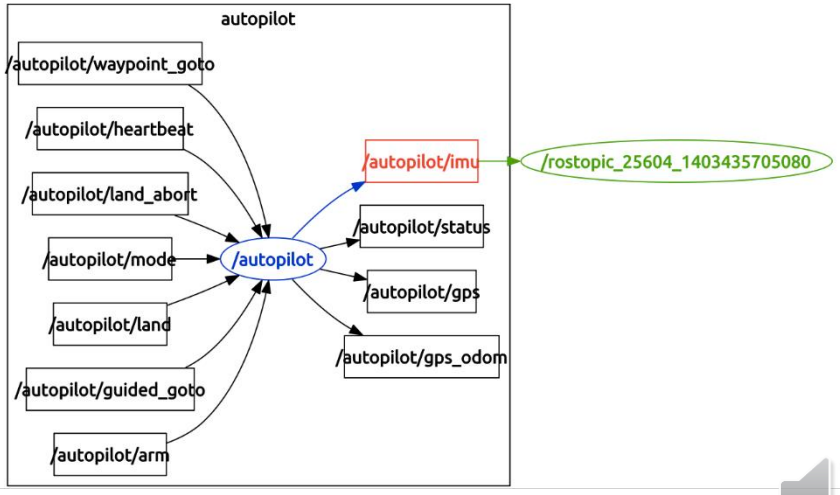


Topics

- Are named buses over which nodes exchange messages
- Are intended for ***unidirectional***, ***streaming*** communications
- There can be multiple publishers and subscribers to a topic



A multiple publishers/subscribers example.

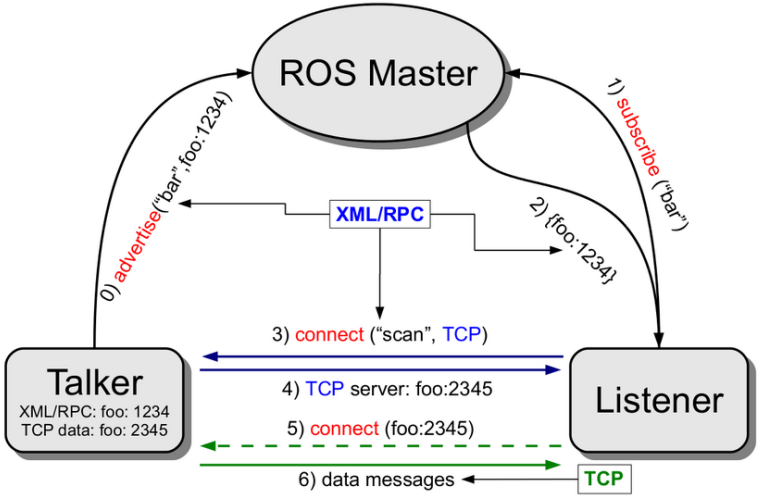


Courtesy of http://wiki.ros.org/Topics
Courtesy of http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics
Courtesy of https://erlerobotics.gitbooks.io/erle-robotics-erle-brain-a-linux-brain-for-drones/content/en/mavlink/ros/autopilot_bridge.html



Topic Transports

- ROS currently supports **TCP/IP-based (TCPROS)** and **UDP-based (UDPROS)** message transports
  - TCP is widely used because it provides a simple, reliable communication stream
  - TCP packets always arrive in order, and lost packets are resent until they arrive
  - UDP is a low-latency, lossy transport, so is best suited for tasks like teleoperation



To emphasize, nodes communicate directly with each other, over an appropriate transport mechanism

- Data does not route through the master
- Data is not sent via XMLRPC
- The XMLRPC system is used only to negotiate connections for data

Courtesy of http://wiki.ros.org/Topics#Topic_Transports
Courtesy of https://wiki.ros.org/Events/CoTeSys-ROS-School?action=AttachFile&do=view&target=ros_tutorial.pdf



Bags

- A **bag** is a file format for storing ROS message data
- Bags are typically created by a tool like **rosbag**
  - which subscribes to one or more ROS topics, and **stores the serialized message data in a file** as it is received
- A bag file is efficient for both **recording and playback**
  - as messages are stored in the same representation used in the network transport layer of ROS
  - A bag file can also be played back in ROS to the **same topics** they were recorded from, or even **remapped** to new topics

```
$ rosbag record -a -O recorded1.bag   # Record all topics to recorded1.bag
$ rosbag play recorded1.bag           # Play back (publish) the contents of the given bags.
```

Courtesy of http://wiki.ros.org/Bags
Courtesy of http://wiki.ros.org/rosbag/Commandline

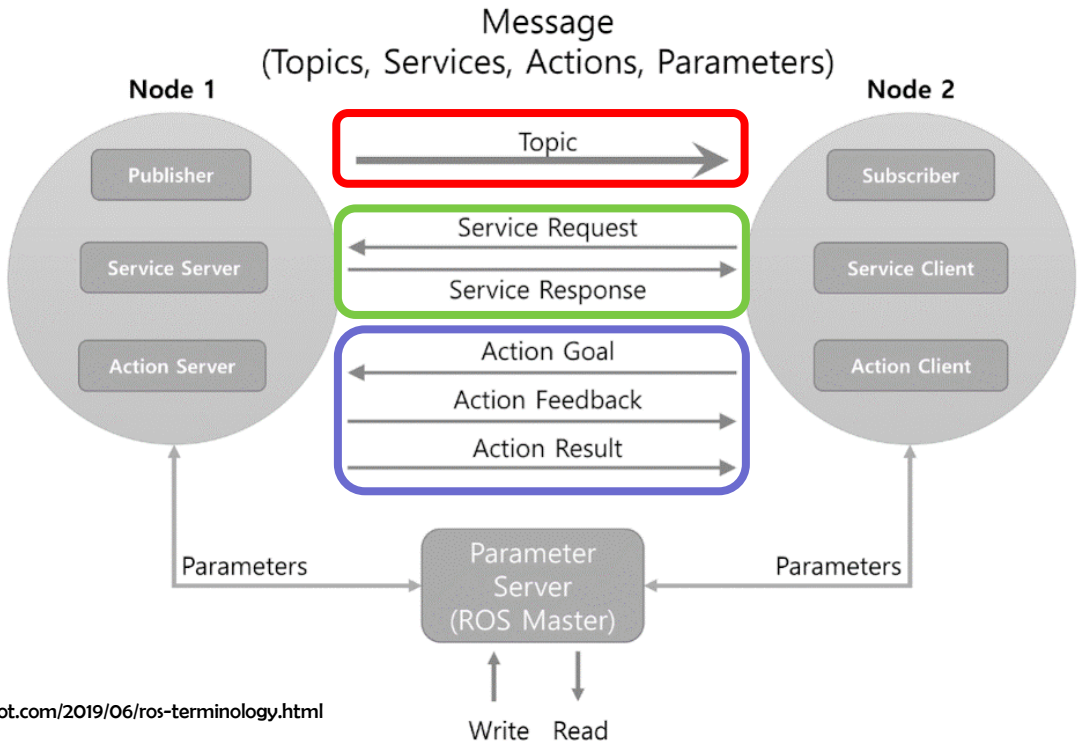


Communication Types in ROS

- Topics (Publisher and Subscriber)
- Services (Client and Server)
- Actions (Client and Server)

Communication Patterns in ROS.

Type	Features		Description
Topic	Asynchronous	Unidirectional	Used when exchanging data continuously
Service	Synchronous	Bi-directional	Used when request processing requests and responds current states
Action	Asynchronous	Bi-directional	Used when it is difficult to use the service due to long response times after the request or when an intermediate feedback value is needed



Courtesy of https://robinrobotic.blogspot.com/2019/06/ros-terminology.html



Services (Client and Server, Synchronous)

- **Request and Reply** is done via a **Service**,
  - which is defined by **a pair of messages**: one for the request and one for the reply
- The service **client** requests a service regarding a particular task
- The service **server** is responsible for responding to requests
  - The publish/subscribe model is a very flexible communication paradigm, but its many-to-many one-way transport is not appropriate for the **request/reply interactions**
  - Example codes are available

Courtesy of https://wiki.ros.org/Services
Courtesy of https://wiki.ros.org/Events/CoTeSys-ROS-School?action=AttachFile&do=view&target=ros_tutorial.pdf

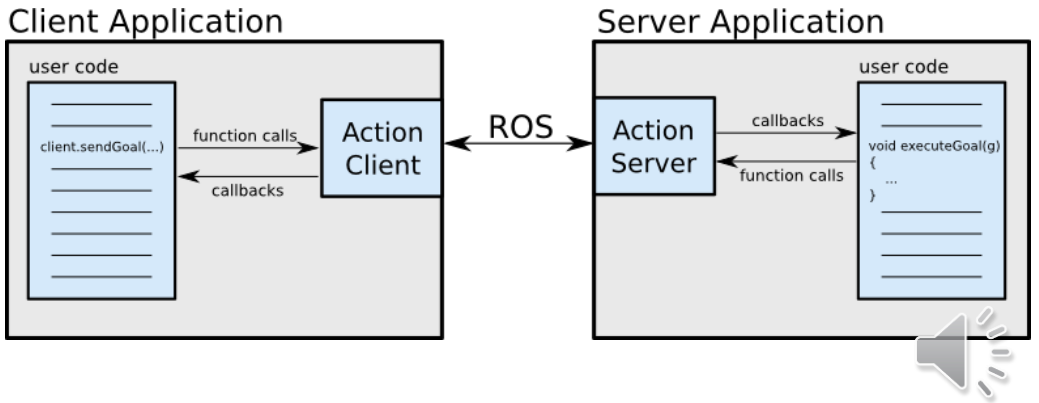


Actions (Client and Server, Asynchronous)

- In some cases, if the **service takes a long time to execute**,
  - the user might want the ability **to cancel the request during execution** or **get periodic feedback** about how the request is progressing
- The actionlib package
  - provides tools to create servers that execute long-running goals that can be preempted
  - provides a client interface in order to send requests to the server

Three *message types* in Actions:

- Goal
  - To accomplish a task using action, a *goal* can be sent to an ActionServer by an ActionClient
- Feedback
  - It provides server implementers a way to tell an ActionClient about the incremental progress of a goal
- Result
  - A result is sent from the ActionServer to the ActionClient upon completion of the goal
  - It is sent only once (and is different from Feedback)



Courtesy of https://wiki.ros.org/actionlib



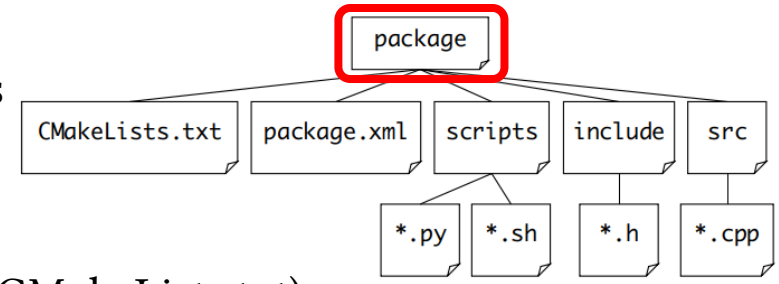
ROS Files Organization

- Packages
- Catkin Workspaces



Packages

- Software in ROS is organized in ***packages***
- A package might contain
  - *ROS nodes*, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module
- Common files and directories
  - **include/package_name**: C++ include headers (make sure to export in the CMakeLists.txt)
  - **msg/**: Folder containing Message (msg) types
  - **src/package_name/**: Source files (C++)
  - **scripts/**: executable scripts (Python)
  - **CMakeLists.txt**: CMake build file (see catkin/CMakeLists.txt)
  - **package.xml**: Package catkin/package.xml

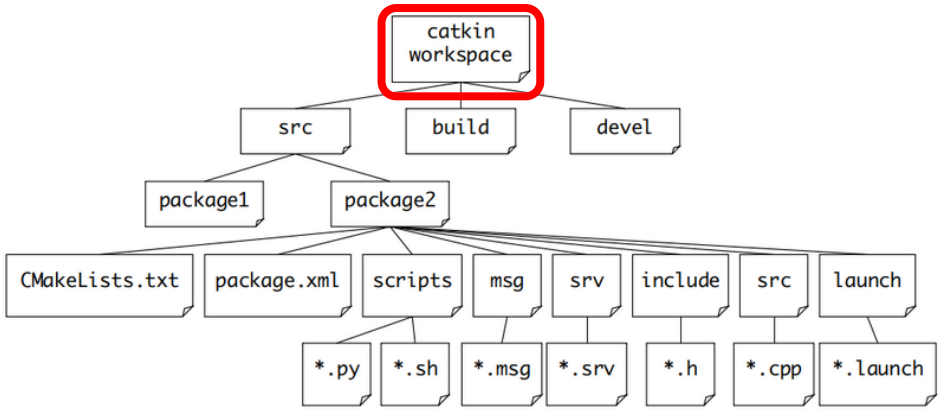


Courtesy of http://wiki.ros.org/Packages
Courtesy of https://segmentfault.com/a/1190000019947692



Catkin Workspaces

- Catkin is low-level build system macros and infrastructure for ROS
- A catkin workspace is a top-level directory
  - where you build, install, and modify ***catkin packages***
- The workspace contains all the packages for your project,
  - along with several other directories for the catkin system to use when building executables and other targets from your source code



Courtesy of http://wiki.ros.org/catkin/workspaces
Courtesy of https://zhuanlan.zhihu.com/p/104773919
Courtesy of https://segmentfault.com/a/1190000019947692



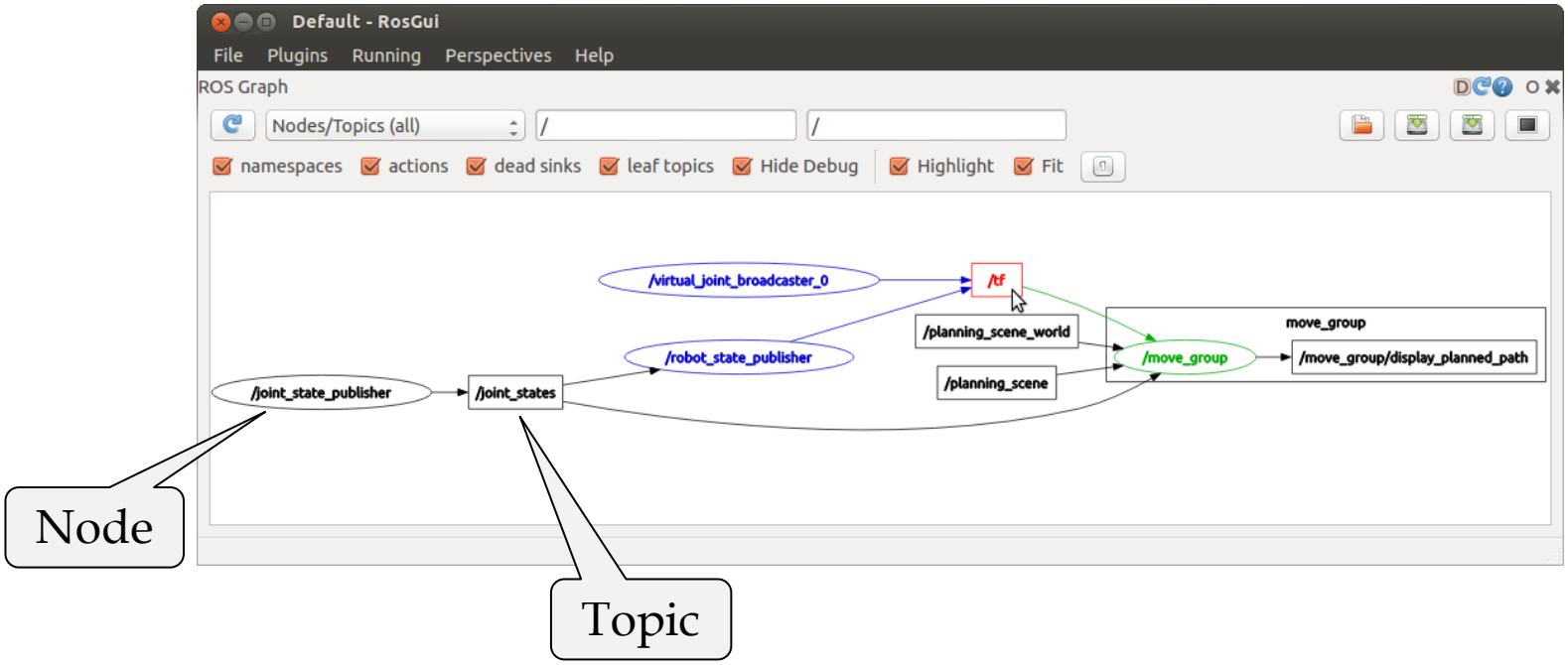
Visualization and Debugging

- rqt_graph
- enable_statistics
- rviz



rqt_graph

- rqt is a Qt-based framework for GUI development for ROS
- **rqt_graph** is a tool providing a GUI plugin for visualizing the ROS computation graph

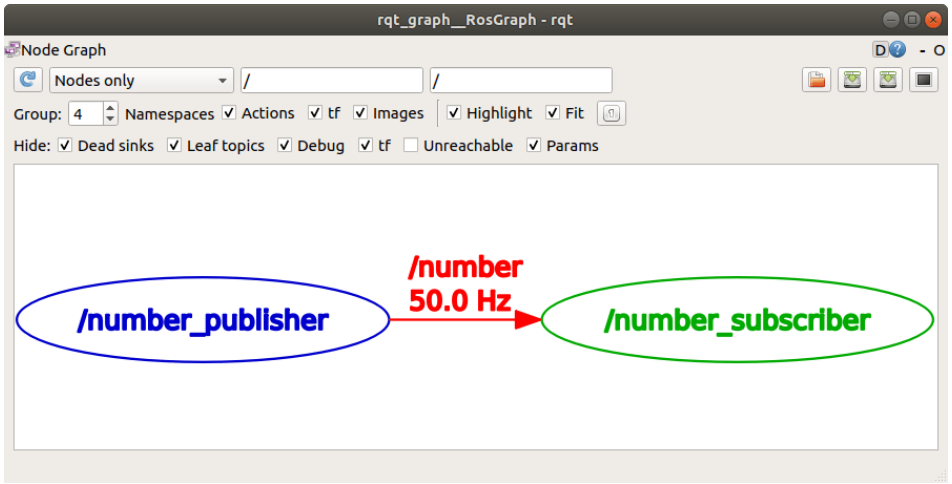




enable_statistics

- To capture runtime statistics of ROS topics
  - you can set the ROS parameter, **enable_statistics**, to see more info on ROS topics with rqt_graph

$ rosparam set enable_statistics true



*One important point: make sure to set this parameter before running any other node, not just **rqt_graph**

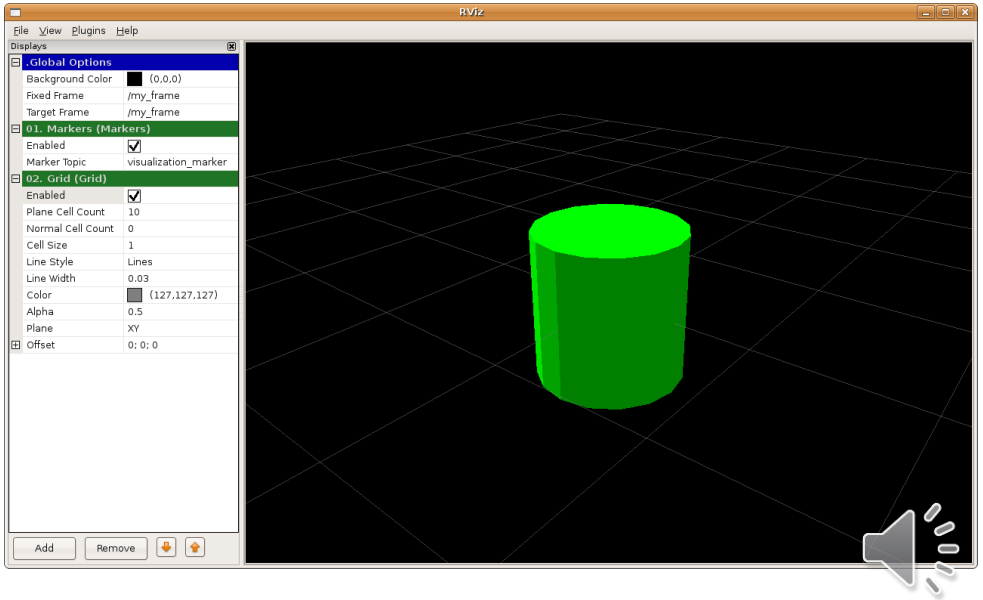
1. Start **roscore**
2. Set the **enable_statistics** parameter
3. Start the publisher and subscriber nodes
4. Start **rqt_graph**

*Courtesy of https://roboticsbackend.com/rqt-graph-visualize-and-debug-your-ros-graph/



rviz

- **rviz** is a **3D visualization tool** for ROS applications
  - Provide a view of your robot model
  - Capture sensor information from robot sensors
  - Replay captured data
  - Display data from camera, lasers, from 3D and 2D devices including pictures and point clouds
- To perform the tasks, **rviz** must be opened and connected to a running simulation job
- See the video for more concrete information

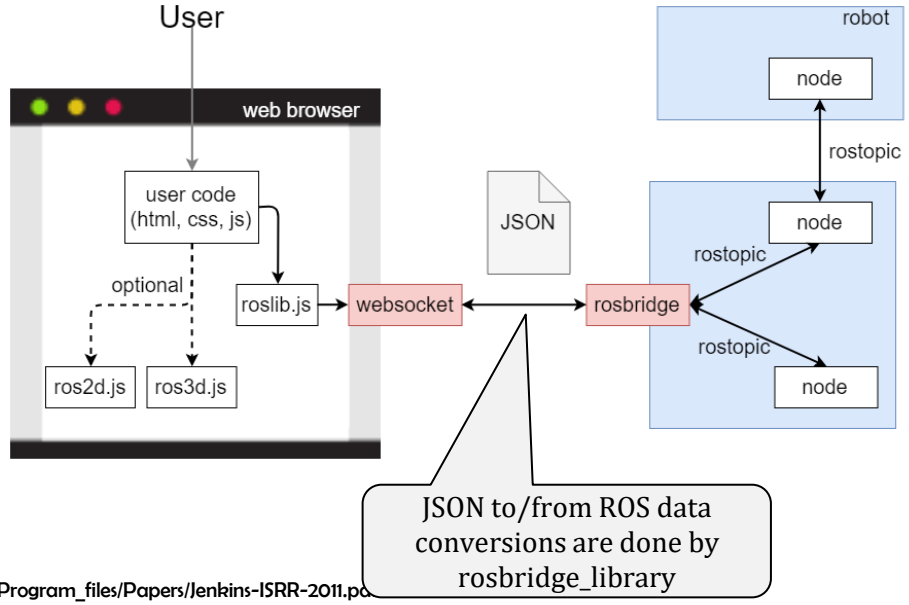


Courtesy of https://docs.aws.amazon.com/en_us/robomaker/latest/dg/simulation-tools-rviz.html



Rosbridge

- Rosbridge provides a **JSON API** to ROS functionality for non-ROS programs
  - Interoperability between ROS nodes and other software modules
- There are a variety of **front-ends** that interface with ***rosbridge***, including a WebSocket server for web browsers to interact with



Courtesy of http://wiki.ros.org/rosbridge_suite 6
Courtesy of http://www.isrr-2011.org/ISRR-2011/Program_files/Papers/Jenkins-ISRR-2011.pd
Courtesy of http://robotwebtools.org/jsdoc/roslibjs/r4/symbols/ROSLIB.Ros.html



References

- ROS Terminology
- The Robotics Back-End
- ROS Tutorial

- https://github.com/qboticslabs/mastering_ros
- https://github.com/ros/cheatsheet/releases/