



INFO 802

Master Advanced Mechatronics

Luc Marechal

2022

Prerequisites

- Ubuntu installation
 - **Ubuntu on a Windows Virtual machine**
 - Or Ubuntu **mate** on a Raspberry Pi
 - Or Ubuntu on a Linux machine
- Installation of ROS Noetic Ninjemys



<https://moodle.univ-smb.fr/course/view.php?id=242>



<https://ubuntu-mate.org/raspberry-pi/>



<https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop>



<http://wiki.ros.org/noetic/Installation>

What is ROS ?

- ROS (Robot Operating System) is an open-source, meta-operating system for your robot
 - open-source: all code is public. Most people share their code as to be used with ROS
 - meta-operating system: contains many of the components expected in an OS: hardware abstraction, low-level control, package management
- It is rather a **middleware** (*i.e.* a framework for writing robot software)
 - collection of tools, libraries, and conventions that help to build robot applications working across a wide variety of robotic platforms



More info

<https://roboticsbackend.com/what-is-ros/>

What is a Middleware ?



- Interface allowing the interconnection of several applications or hardware that were not necessarily designed to communicate with each other (not being on the same network, not sharing the same network, the same OS or the same protocols.)
- Software that acts as a bridge between an operating system or database and applications, especially on a network.
- Responsible for handling the communication between programs in a **distributed system**.

What is ROS ?



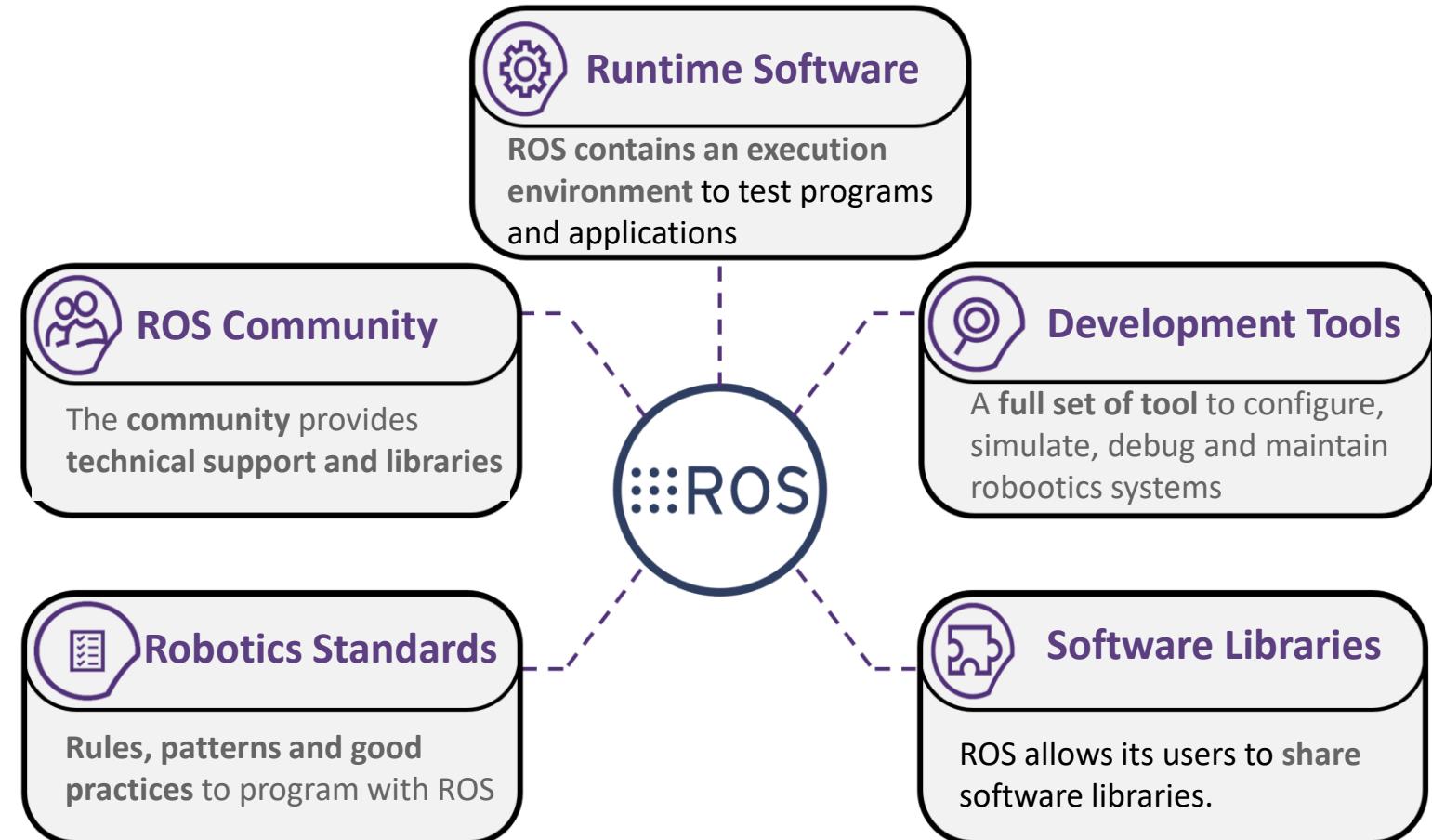
- **Plumbing** : ROS provides a message-passing system, often called “middleware” or “plumbing”
- **Tools** : launch, introspection, debugging, visualization, plotting, logging, and playback ...
- **Capabilities** : From drivers to algorithms, to user interfaces, ROS provides the building blocks
- **Community** : The ROS community is large, diverse, and global.

More info

<https://www.ros.org/blog/ecosystem/>

What is ROS ?

- ROS is mainly composed of 2 things:
 - A core (middleware) with communication tools
 - A set of plug & play libraries



More info

<https://roboticsbackend.com/what-is-ros/>

Programming languages of industrial robot arms

- Almost every robot manufacturer has developed their own proprietary robot programming language → nothing was unified

Manufacturer

Language

KUKA

RAPID



FANUC

KRL



ABB

Karel



STÄUBLI

VAL3



**UNIVERSAL
ROBOTS**

URscript



RAPID

```

1 MODULE Module1
2   PROC Main()
3     IF DI_Spuscene = 1 THEN
4       Path_10;
5       Path_20;
6       Path_30;
7     ELSE
8       Break;
9       MoveAbsJ Home,v300,fine,efektor_4_1\WObj:=wobj0;
10      Reset DO_Zatvor;
11      Reset DO_Otvor;
12      Reset DO_10toc;
13      Reset DO_Otoc;
14      Reset DO_Vysun;
15      Reset DO_Zasun;
16    ENDIF
17  ENDPROC
18  PROC Path_10()

```

VAL3

```

begin
movej(above2,flange,mNomSpeed)

movel(p[0],flange,mNomSpeed)
movel(p[1],flange,mNomSpeed)
movej(above2,flange,mNomSpeed)

movej(above2,flange,mNomSpeed)
waitEndMove()
end

```

KRL

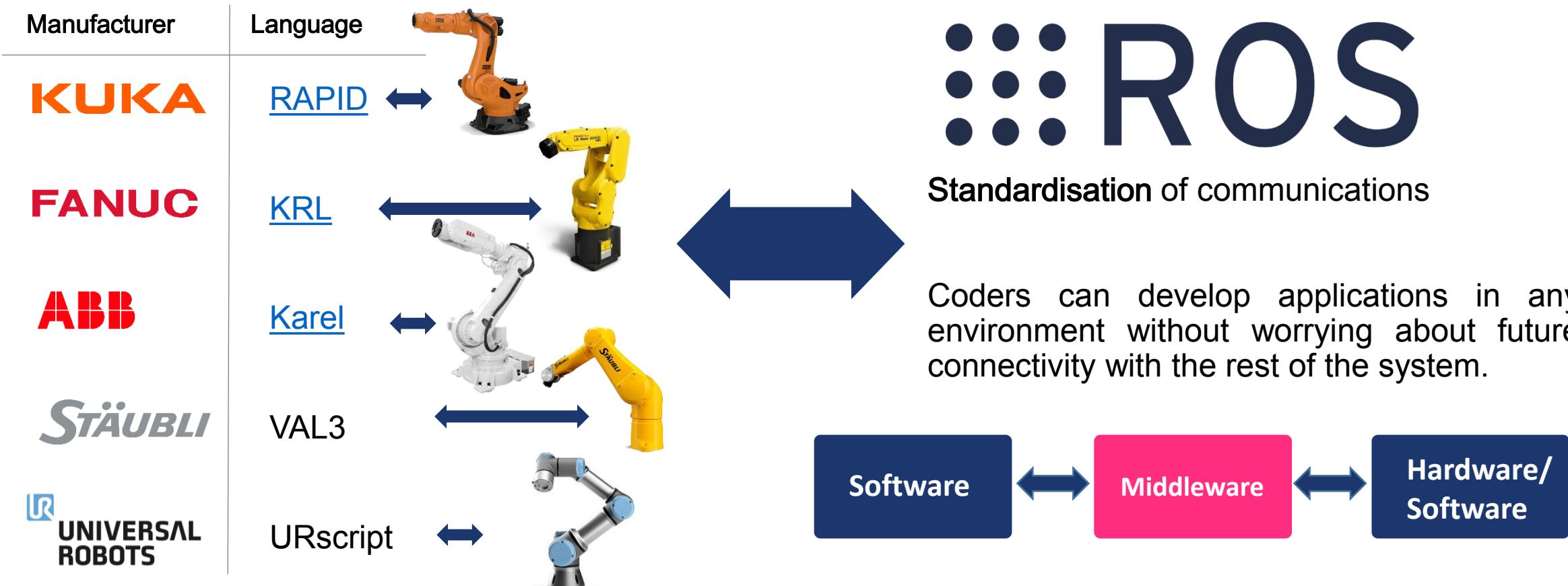
```

0 0
1 DEF RSI_Ethernet( )
2 ; =====
3 ;
4 ; RSI EXAMPLE: ETHERNET communication
5 ; Realtime UDP data exchange with server application
6 ;
7 ; =====
8 ;
9 ; Declaration of KRL variables
10 DECL INT ret ; Return value for RSI commands
11 DECL INT CONTID ; ContainerID
12 INI
13 ;
14 ; Move to start position
15 PTP {A1 0.82, A2 -57.12, A3 94.15, A4 -178, A5 32.92,
16   A6 268.08}
17 ;
18 ; Create RSI Context
19 ret = RSI_CREATE("RSI_Ethernet.rsi",CONTID,TRUE)
20 IF (ret <> RSIOK) THEN
21   HALT
22 ENDIF
23 ;
24 ; Start RSI execution
25 ret = RSI_ON(#RELATIVE)
26 IF (ret <> RSIOK) THEN
27   HALT
28 ENDIF
29 ;
30 RSI_MOVECORR()
31 ; Turn off RSI
32 ret = RSI_OFF()
33 IF (ret <> RSIOK) THEN
34   HALT
35 ENDIF

```

Why using ROS ?

- Almost every robot manufacturer has developed their own proprietary robot programming language → nothing was unified



Why using ROS ?

- **We use ROS to**
 - Interact between different programs (threads) running in parallel
 - Interact with robot hardware
 - Display data in real time
 - Record and replay sensor data
- **Advantages of ROS**
 - Easy way to share and use code from others
 - It hides the complexity to use several computers talking to each other
 - Use of the speed of C++ in some parts and the flexibility of Python in other parts.
 - Nodes in ROS do not have to be on the same system (multiple computers) or even of the same architecture!

Why using ROS ?

- **We use ROS to**
 - Interact between different programs (threads) running in parallel
 - Interact with robot hardware
 - Display data in real time
 - Record and replay sensor data
- **Advantages of ROS**
 - Easy way to share and use code from others
 - It hides the complexity to use several computers talking to each other
 - Use of the speed of C++ in some parts and the flexibility of Python in other parts.
 - Nodes in ROS do not have to be on the same system (multiple computers) or even of the same architecture!

History

- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory
- Development continued at Willow Garage founded by Larry Page (also Google cofounder)
- Since 2013 managed by OSRF (Open Source Robotics Foundation)
- De facto standard for robot programming



More info - source

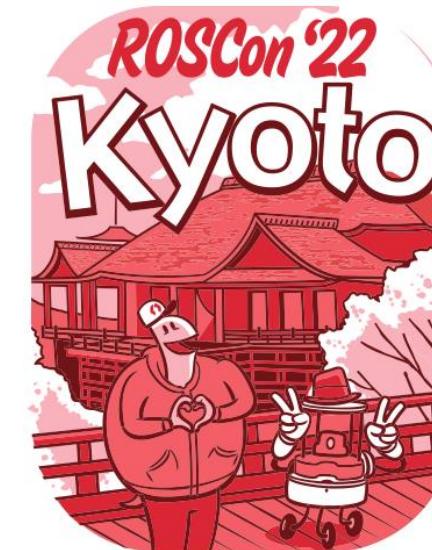
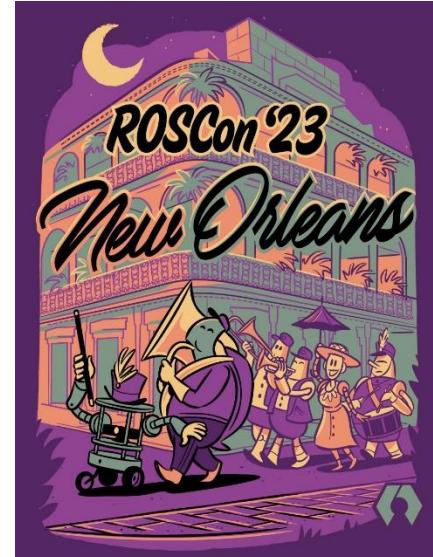
<http://www.willowgarage.com>

<https://www.theconstructsim.com/history-ros/>

The STAIR
(STanford AI
Robot) project
seeks to
develop the
software
needed to put
a general-
purpose robot
in every home

ROScon

- ROSCon is a developers conference.
- Two days learning from and networking with the ROS community.
- Get tips and tricks from experts and meet and share ideas with fellow developers from around the globe.



More info - source
<http://www.willowgarage.com>

Robot using ROS



[Fraunhofer IPA Care-O-bot](#)



[Videre Erratic](#)



[TurtleBot](#)



[Aldebaran Nao](#)



[Lego NXT](#)



[Shadow Hand](#)



[Willow Garage PR2](#)



[iRobot Roomba](#)



[Robotnik Guardian](#)



[Merlin miabotPro](#)



[AscTec Quadrotor](#)



[CoroWare Corobot](#)



[Clearpath Robotics Husky](#)



[Clearpath Robotics Kingfisher](#)



[Festo Didactic Robotino](#)

More info
<http://wiki.ros.org/Robots>

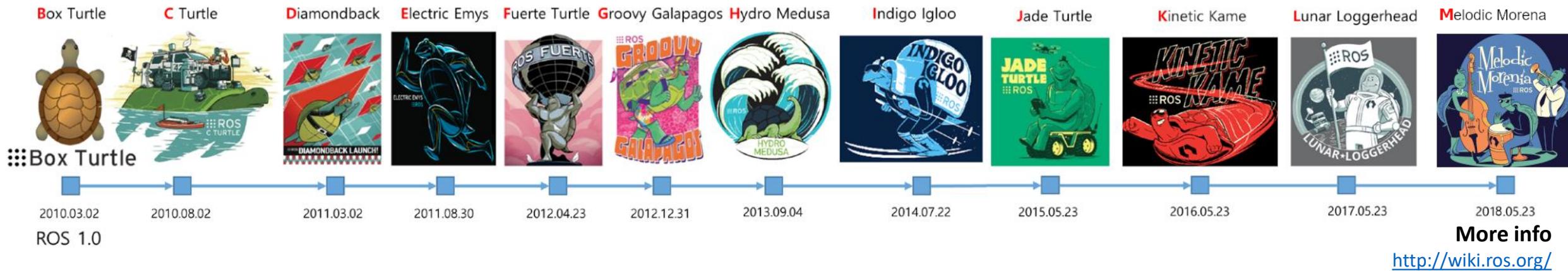
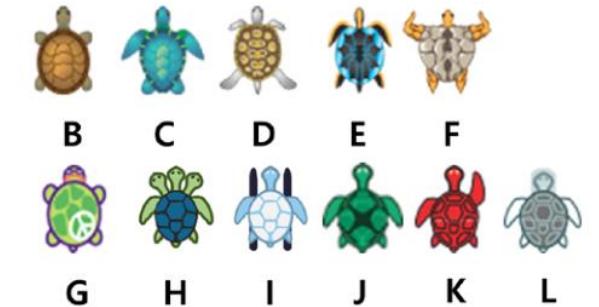
Sensors interfaced with ROS

- 1D/2D/3D Rangefinder.
 - Sharp IR rangefinder / Hokuyo laser scanner / Microsoft Kinect
- Cameras
 - Monocular / stereo / USB / Video streaming (gstreamer)
- Force / torque / touch sensors
- Motion capture systems
- IMU / GPS
- Audio / Speech recognition
- RFID
- Actuators / Interfaces
 - Dynamixel
 - Arduino
 - Lego NXT

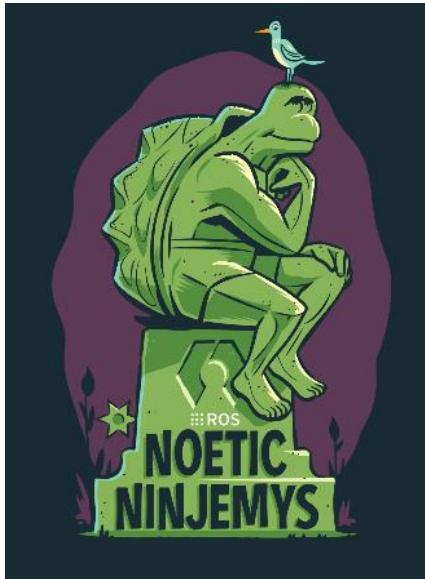


Release Schedule

- There is a ROS release every year in May.
 - Releases on even numbered years are LTS release, supported for 5 years.
 - Releases on odd numbered years are normal ROS releases, supported for 2 years.
 - ROS releases will drop support for EOL Ubuntu distributions, even if the ROS release is still supported.



ROS Latest Releases



**Noetic
Ninjemys**
(2020->2025)

Select Your Platform

Supported:



Ubuntu Focal amd64 armhf arm64



Debian Buster amd64 arm64

[Source installation](#)

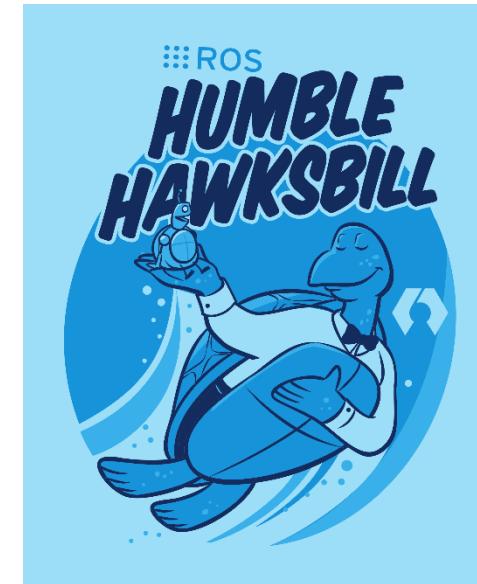
Experimental:

Windows 10 amd64



Arch Linux Any amd64 i686 arm armv6h armv7h aarch64

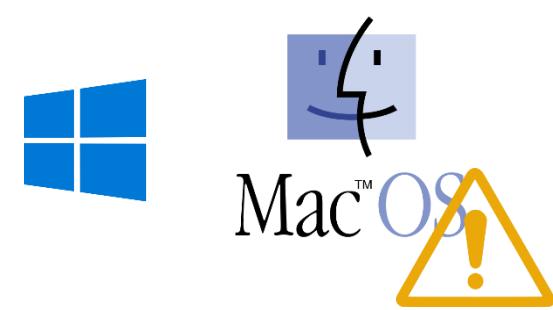
(ROS2)



**Humble
Hawksbill**
(2022 -> 2027)

ROS Supported Platforms

- ROS is currently supported only on **Ubuntu**
 - Fedora
 - Gentoo
 - Arch Linux
 - other variants such as Windows and Mac OS X are considered experimental (will be supported on ROS 2.0)

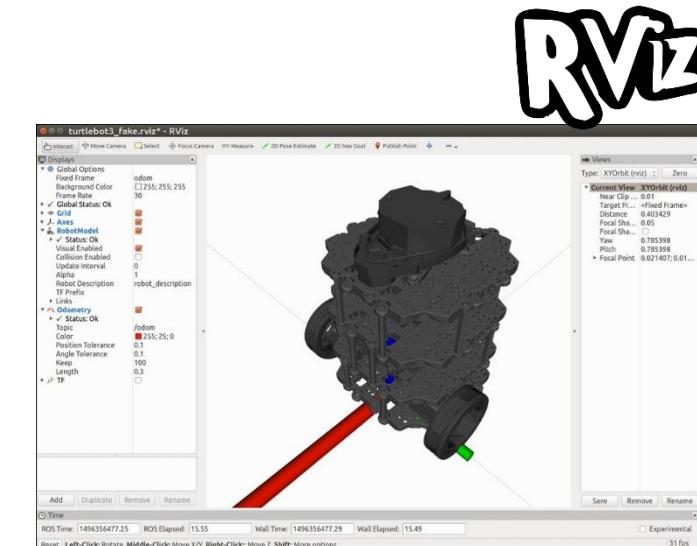
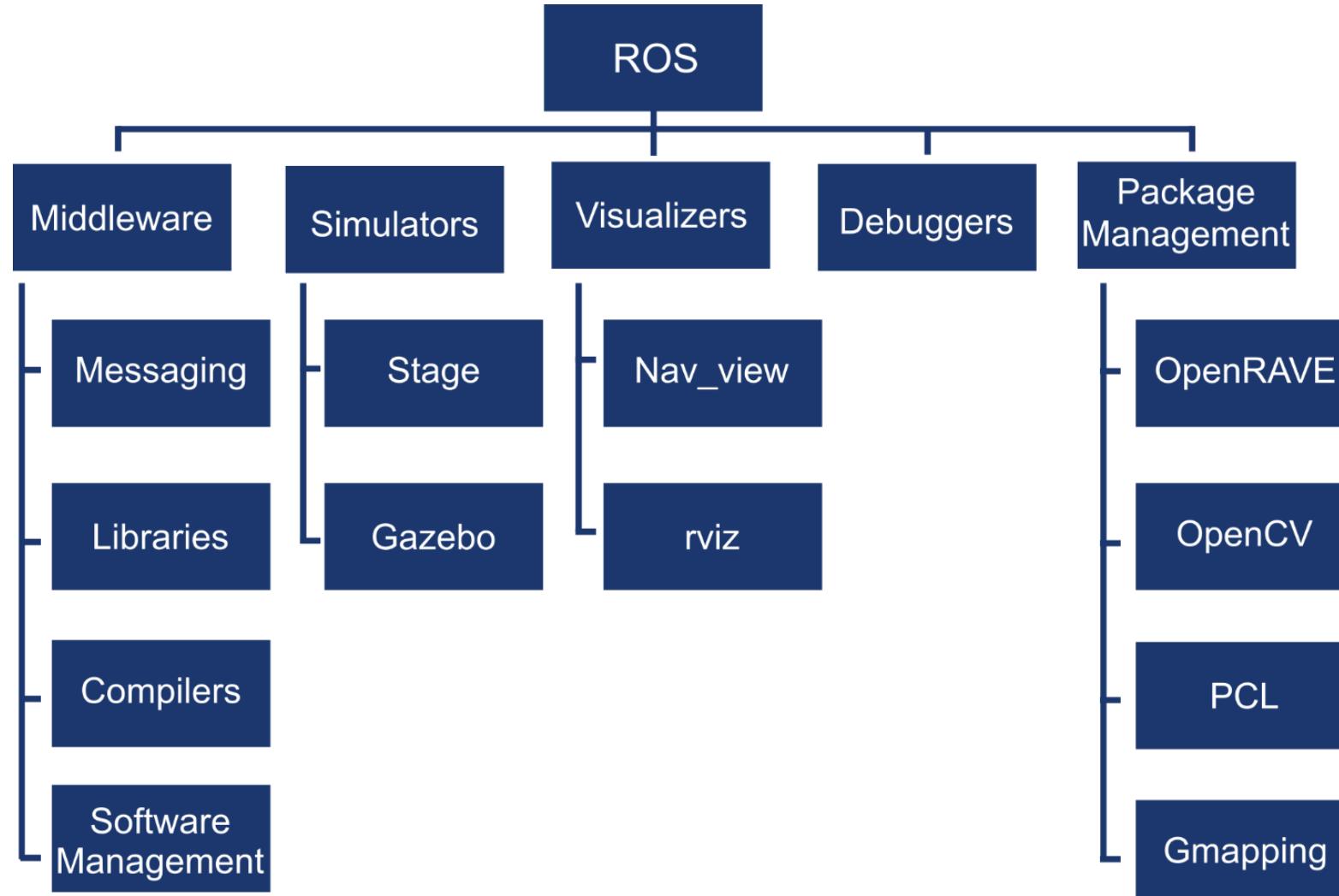


ROS 2

- Since ROS was started in 2007, a lot has changed in the robotics and ROS community.
- The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn't.



ROS Components



What makes the difference?

Conventional OS	ROS
Explicitly a general purpose	Exclusive for Robotic Platform
Native Language Programming	Language-independent architecture
Programming IDE	Software Frameworks
Proprietary/Open-Source	Open-source under BSD license
Programs	Nodes
Communication	Message
Kernel is Included	Kernelless

ROS Philosophy

- **Peer to peer**

Individual programs communicate over defined API (ROS messages, services, etc.)

- **Distributed**

Programs can be run on multiple computers and communicate over the network.

- **Multi-lingual**

ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, LISP, Ruby, etc.).

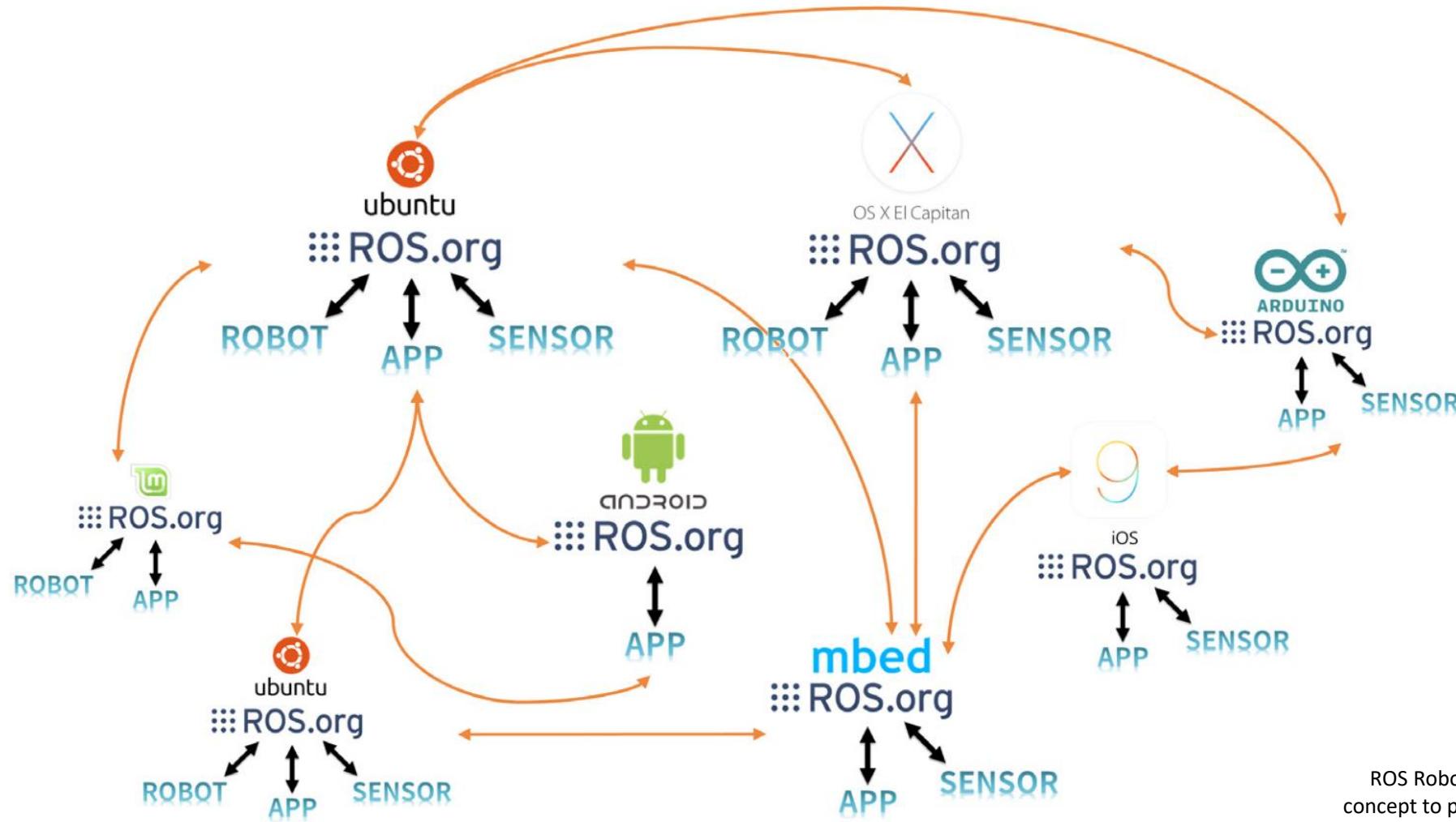
- **Thin**

Stand-alone libraries are wrapped around with a thin ROS layer.

- **Free and open-source**

Most ROS software is open-source and free to use.

Communication between Heterogenous Devices



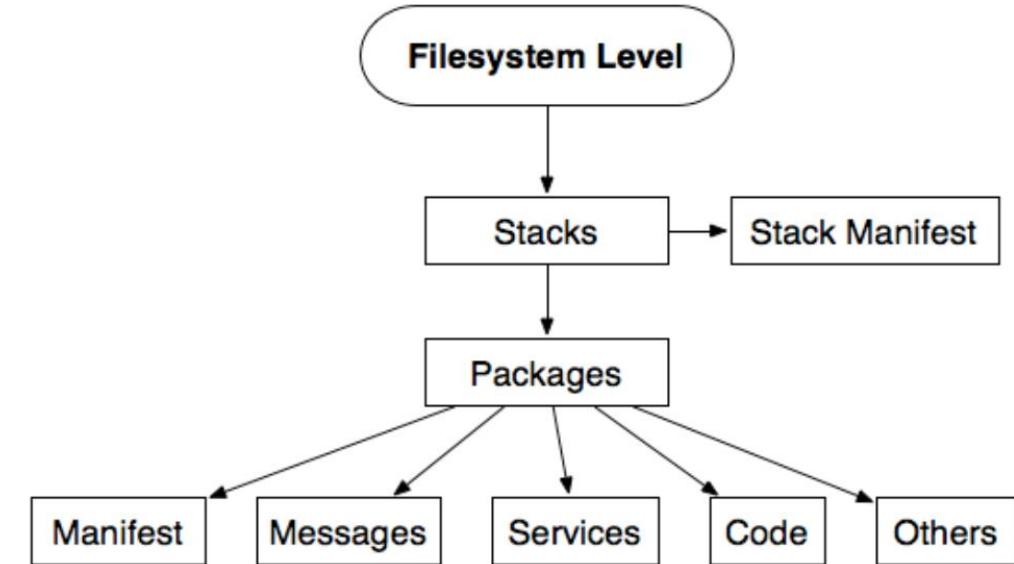
More info - source

ROS Robot Programming - From the basic concept to practical programming and robot application

ROS code hierarchy

- **Repository:** contains all the code from a particular development group
- **Stack:** groups all code on a particular subject / device
- **Packages:** separate modules that provide different services
- **Nodes:** executable that exist in each model

Repository



ROS Communication Protocol

- ROS Communication Protocol helps in Connecting nodes over the network.
These capabilities are built entirely on two high-level communication API's
- **ROS Topics**
 - Asynchronous 'stream-like' communication
 - TCP/IP or UDP Transport
 - Strongly-typed (ROS.msg spec)
 - Can have one or more publishers
 - Can have one or more subscribers
- **ROS Services**
 - Synchronous 'function-call-like' communication
 - TCP/IP or UDP Transport
 - Strongly-typed (ROS.srv spec)
 - Can have only one server
 - Can have one or more clients

How to decide what to use ?

- **Topics:** used mainly for sending **data streams** between nodes.

Example: you're monitoring the temperature of a motor on the robot. The node monitoring this motor will send a data stream with the temperature. Now, any other node can subscribe to this topic and get the data

- **Services:** execution of fast tasks. Allows you to create a simple synchronous client/server communication between nodes. Very useful for changing a setting on your robot, or ask for a specific action: enable freedrive mode, ask for specific data, etc.
- **Actions:** execution of tasks that need to be tracked and should be preempted in some cases.

Also, where the client can send a request that takes a long time (ex: asking to move the robot to a new location). The client can asynchronously monitor the state of the server, and cancel the request anytime.

Source

<https://roboticsbackend.com/what-is-ros/>

ROS Overview

- Launch roscore
- Launch Node 1
- Launch Node 2

```
> roscore

roscore http://192.168.1.77:11311/  Q  -  X

ros@masterpc:~$ roscore
... logging to /home/ros/.ros/log/d681119a-8bb5-11eb-a6a3-b
76924865bd7/roslaunch-masterpc-2839.log
Checking log directory for disk usage. This may take a whil
e.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.1.77:44329/
ros_comm version 1.15.9

SUMMARY
=====

PARAMETERS
* /rosdistro: noetic
* /rosversion: 1.15.9

NODES

auto-starting new master
process[master]: started with pid [2849]
ROS_MASTER_URI=http://192.168.1.77:11311/

setting /run_id to d681119a-8bb5-11eb-a6a3-b76924865bd7
process[rosout-1]: started with pid [2859]
started core service [/rosout]
```

ROS Master

- Manages the communication between nodes
- Every node registers at startup with the master
- Exactly **one** master per system

ROS Master

Start a master with

```
> roscore
```

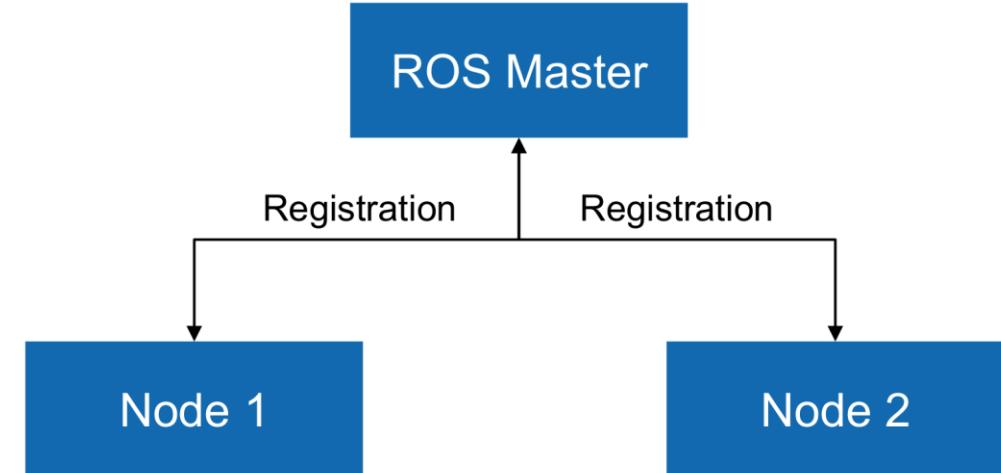
- ROS uses socket communication to facilitate networking.
The roscore starts on http://my_computer:11311
- roscore will start up:
 - a ROS Master
 - a ROS Parameter Server
 - a rosout logging node

More info

<http://wiki.ros.org/Master>

ROS Nodes

- Executable programs that uses ROS to communicate with other nodes
- Individually compiled, executed, and managed
- Organized in *packages*
- Nodes are written using a **ROS client library**
 - roscpp : C++ client library
 - rospy : python client library
- Nodes can publish or subscribe to a Topic
- Nodes can also provide or use a Service



More info

<http://wiki.ros.org/rosnode>

ROS Nodes

Run a node with

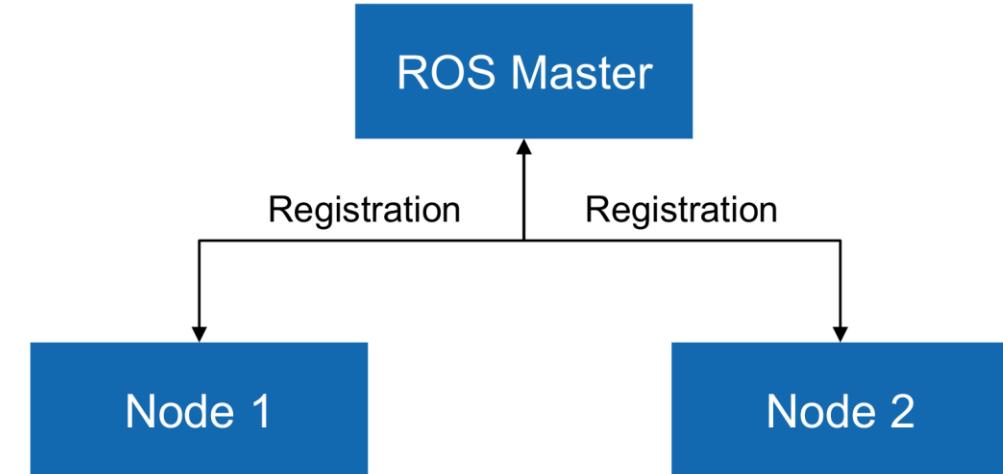
```
> rosrun [package_name] [node_name]
```

See active node with

```
> rosnodes list
```

Retrieve information about a node with

```
> rosnodes info [node_name]
```



More info

<http://wiki.ros.org/rosnodes>

ROS Topics

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

List active topics with

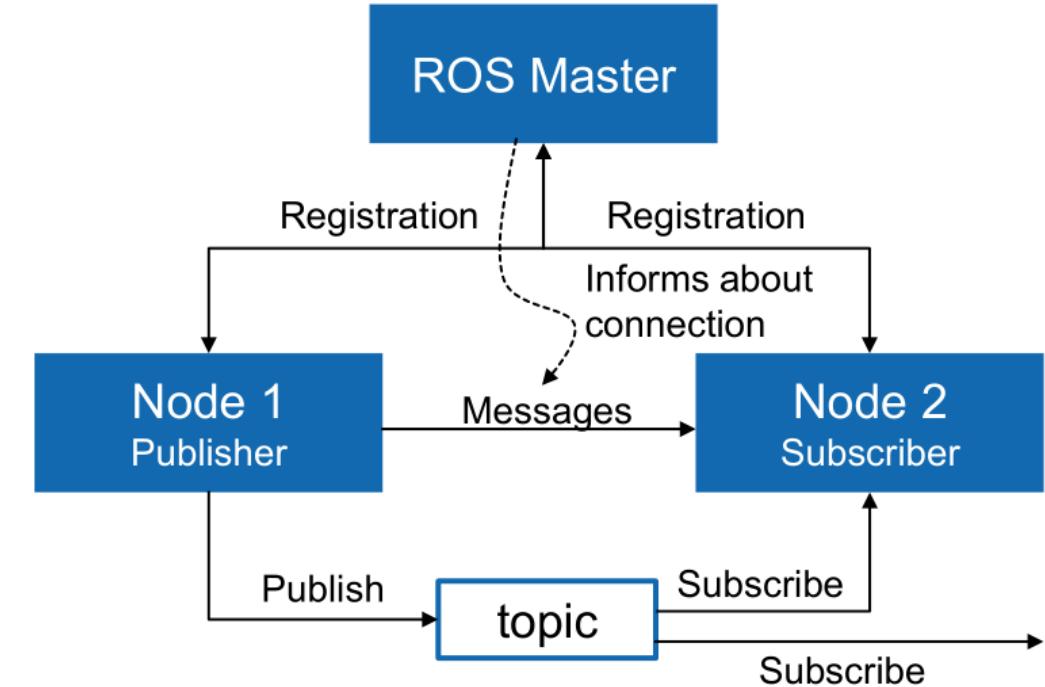
```
> rostopic list
```

Subscribe and print the contents of a topic with

```
> rostopic echo /topic
```

Show information about a topic with

```
> rostopic info /topic
```



More info

<http://wiki.ros.org/rostopic>

ROS Messages

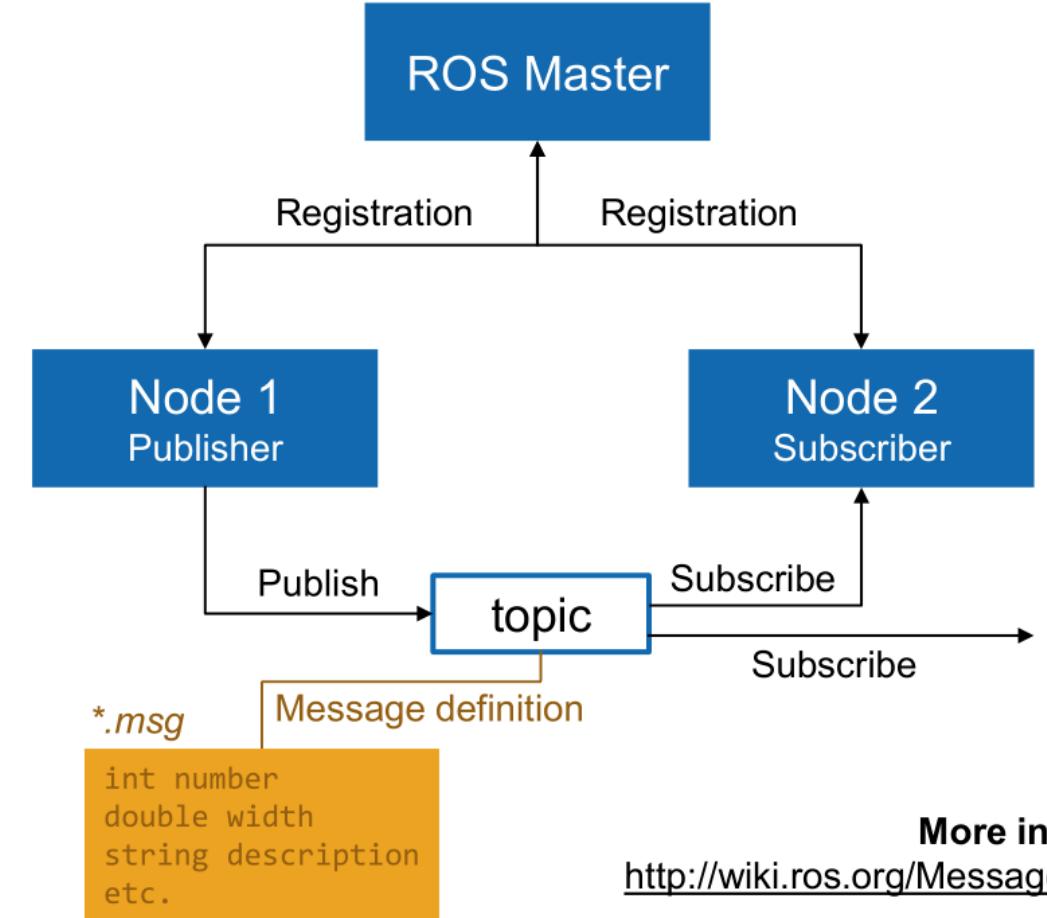
- Data structure defining the *type* of a topic
- Comprised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in **.msg* files

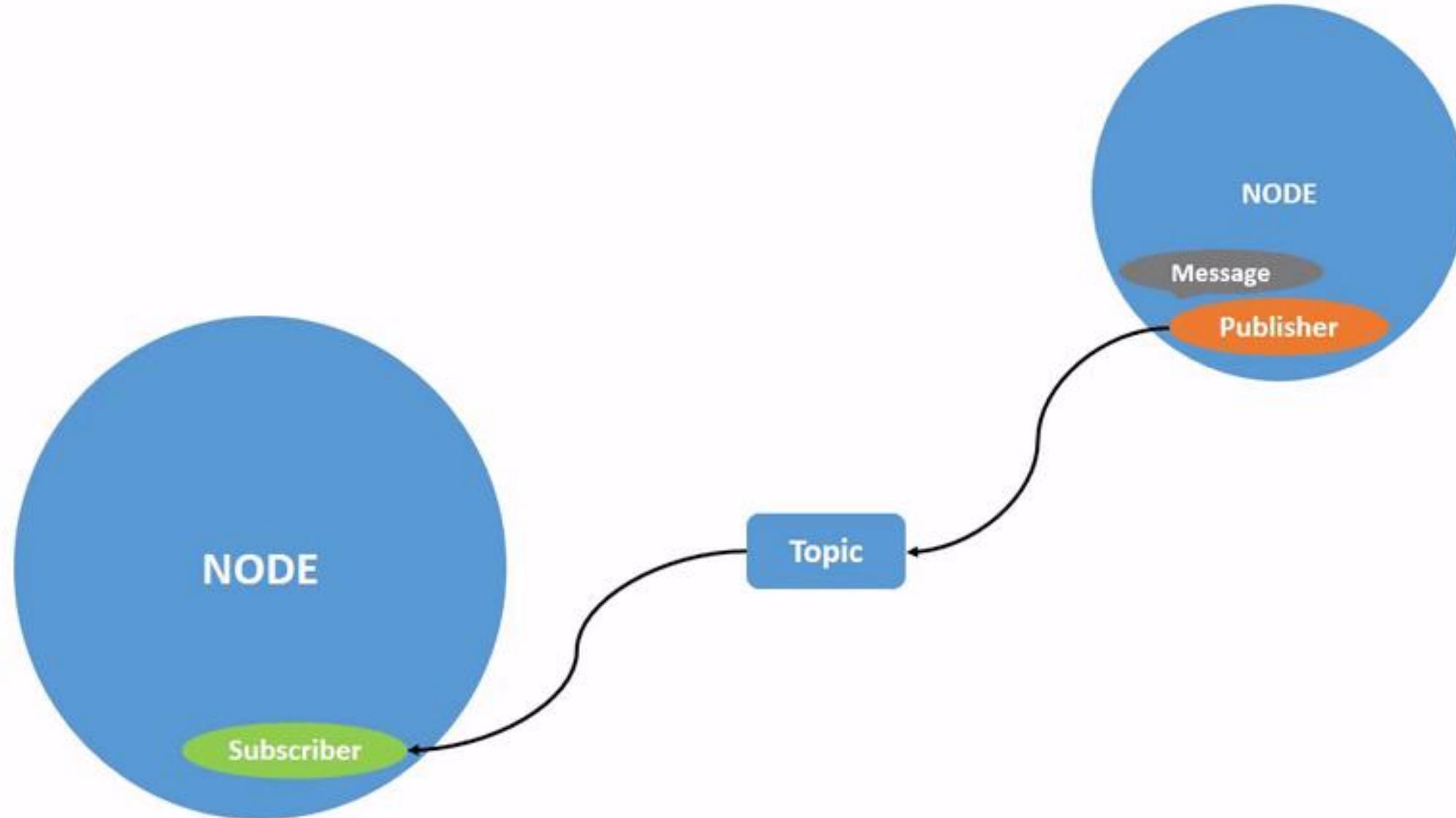
See the type of a topic

```
> rostopic type /topic
```

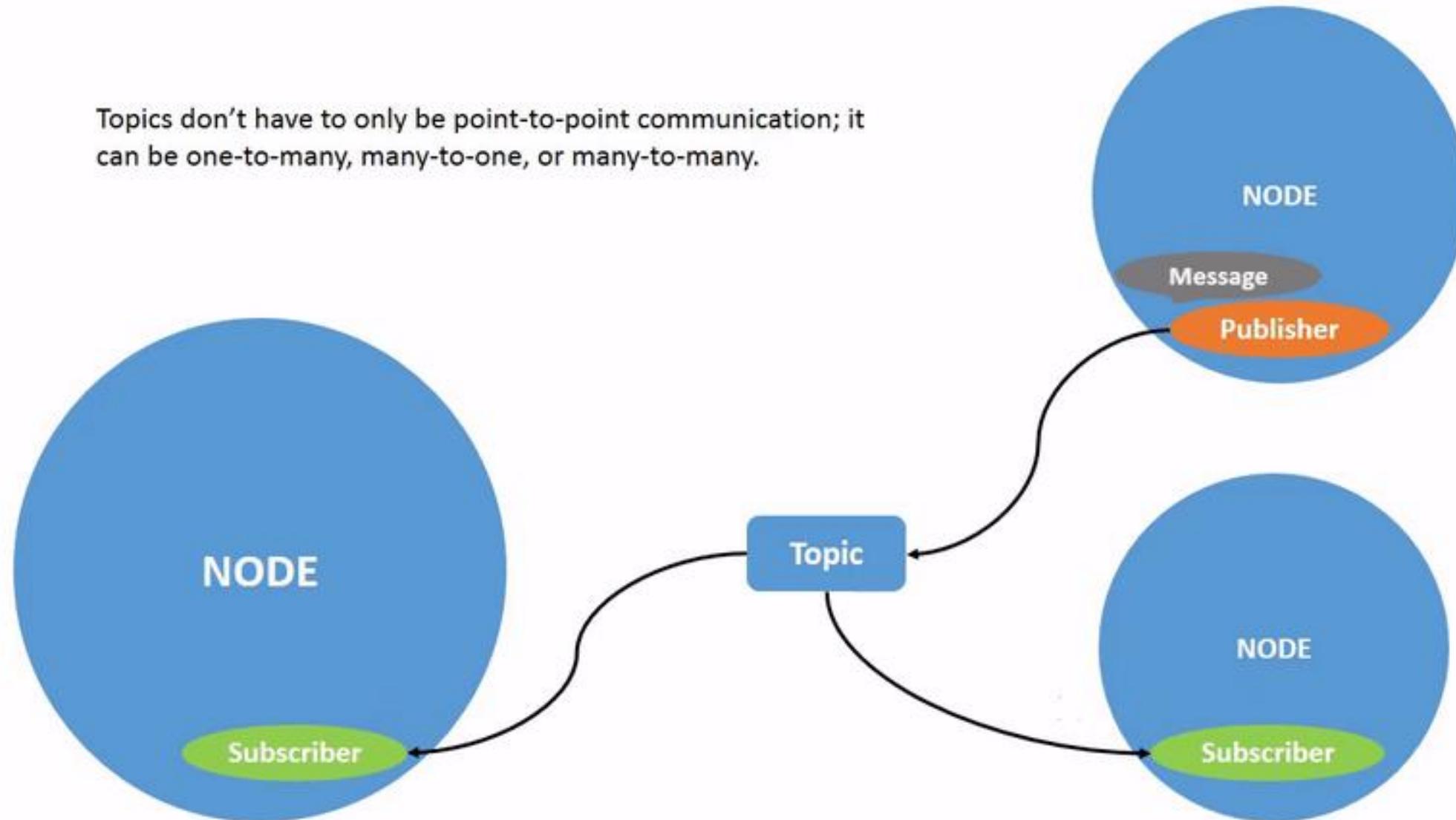
Publish a message to a topic

```
> rostopic pub /topic type data
```





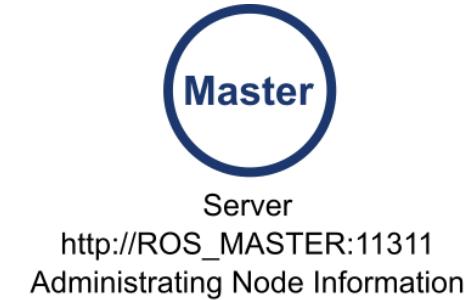
Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.



Messages Communication Flow

Running the Master

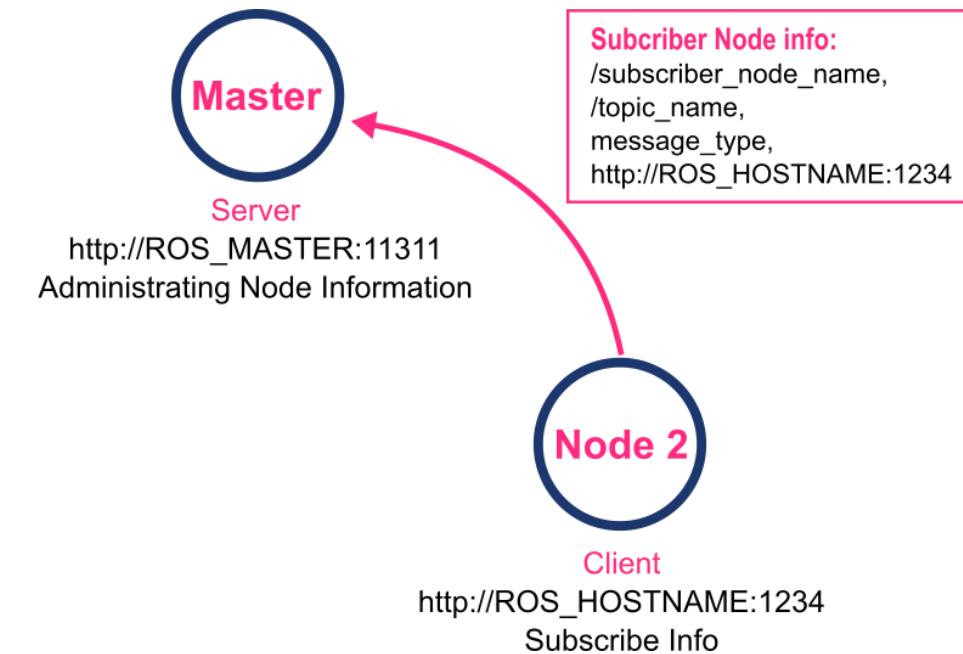
- The master registers the name of nodes, topics, services, action, message types, URI addresses



Messages Communication Flow

Running the Subscriber Node

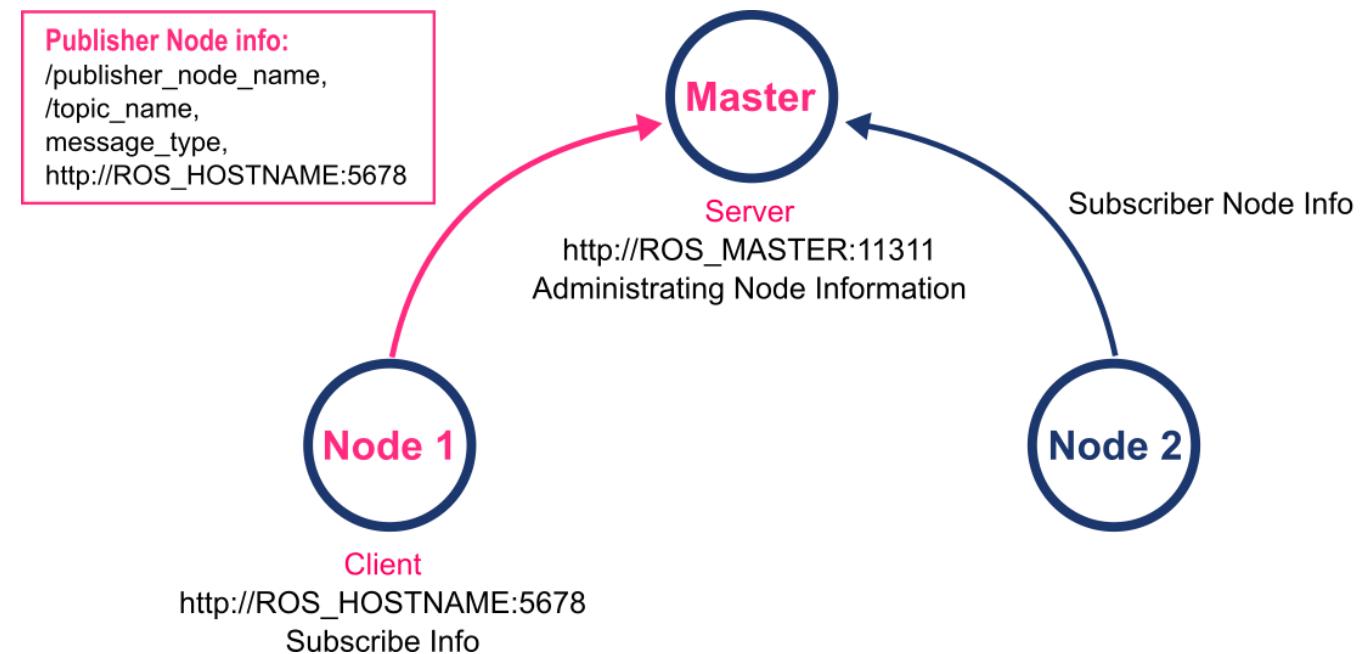
- The subscriber node registers its node name, topic name, message type, URI address, and port with the master



Messages Communication Flow

Running the Publisher Node

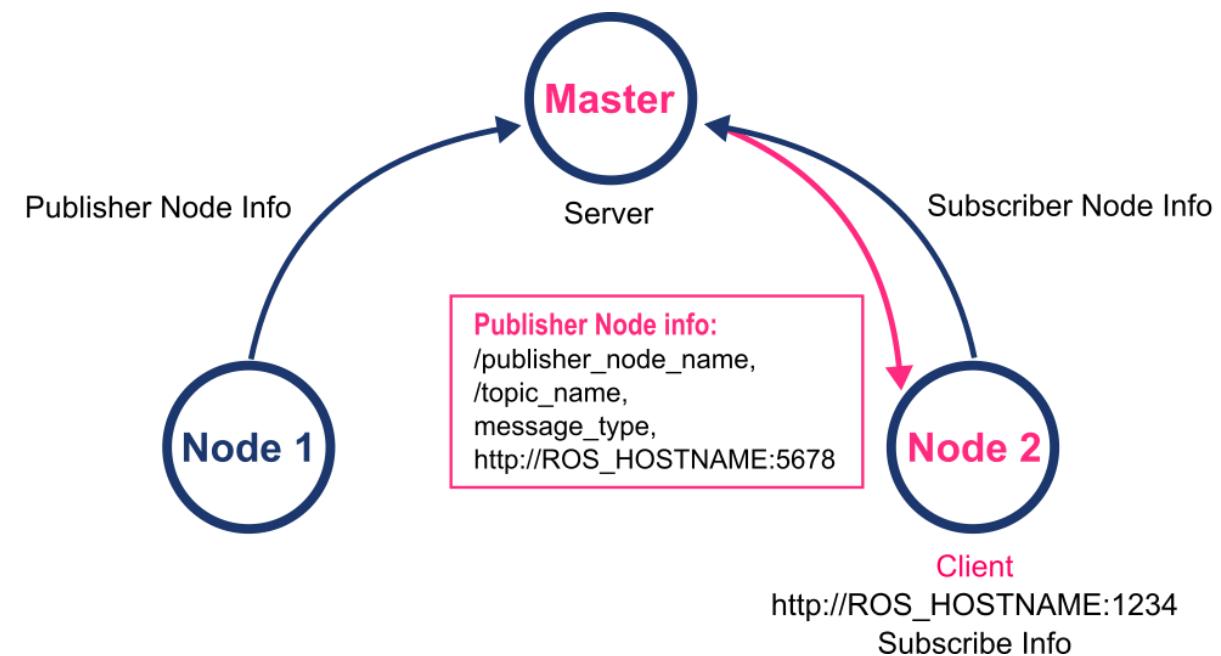
- The publisher node registers its node name, topic name, message type, URI address and port with the master.



Messages Communication Flow

Providing Publisher Information

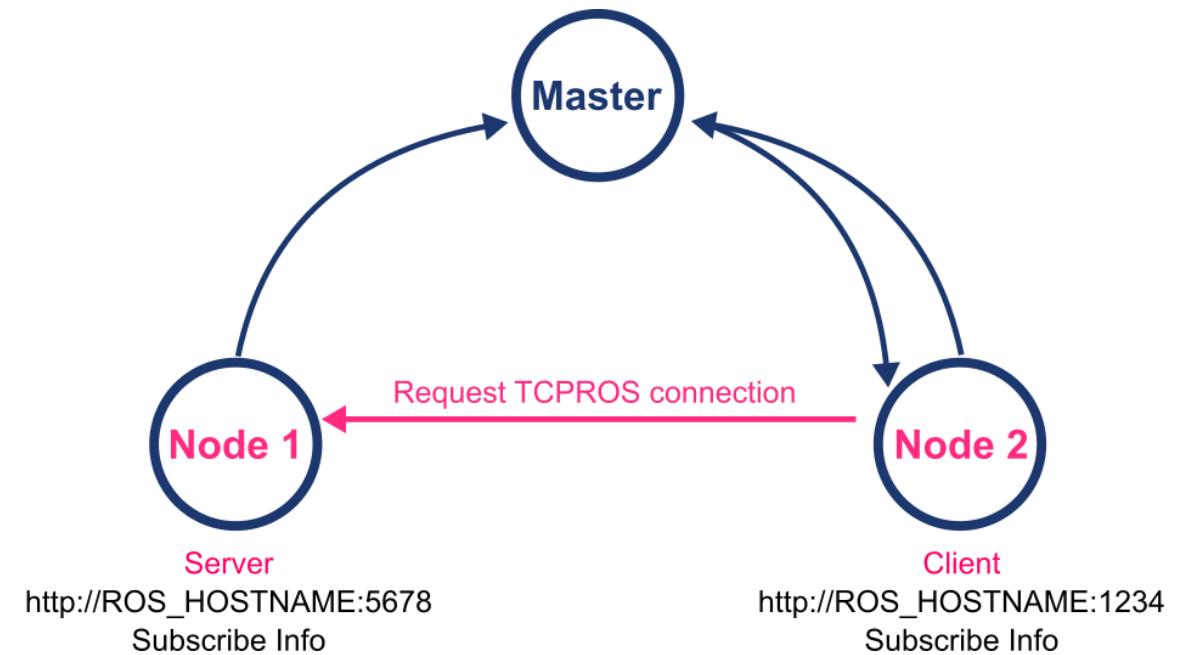
- The master distributes information such as the publisher's name, topic name, message type, URI address and port number of the publisher to subscribers that want to connect to the publisher node



Messages Communication Flow

Connection Request from the Subscriber Node

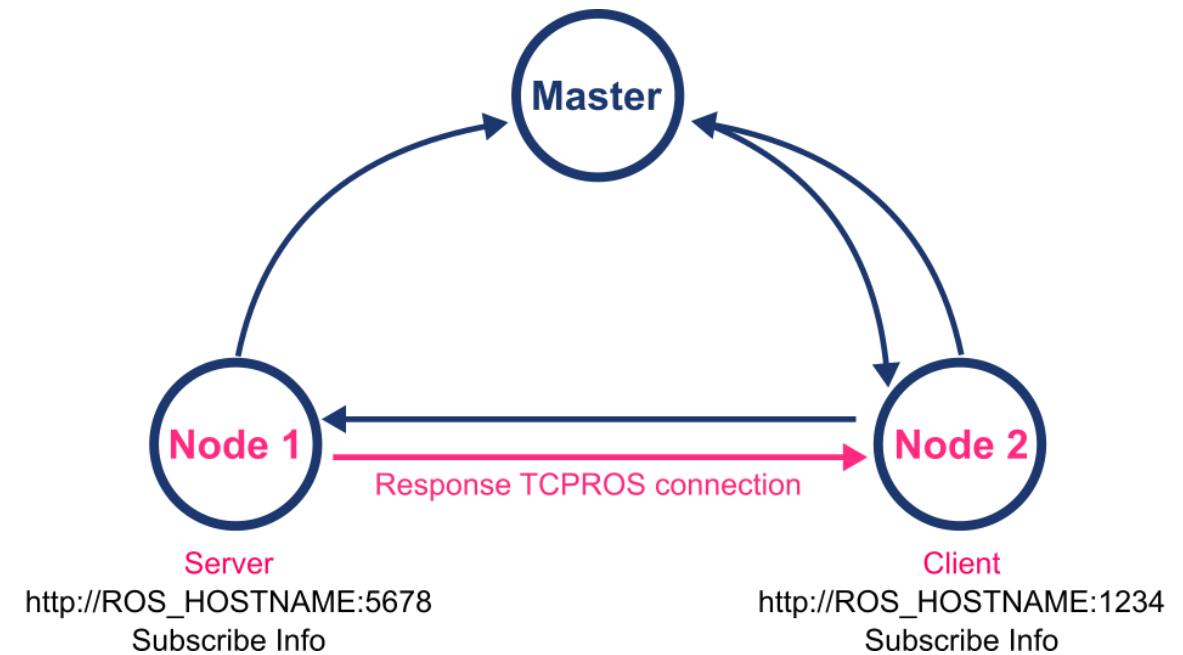
- The subscriber node requests a direct connection to the publisher node based on the publisher information received from the master.



Messages Communication Flow

Connection Response from the Publisher Node

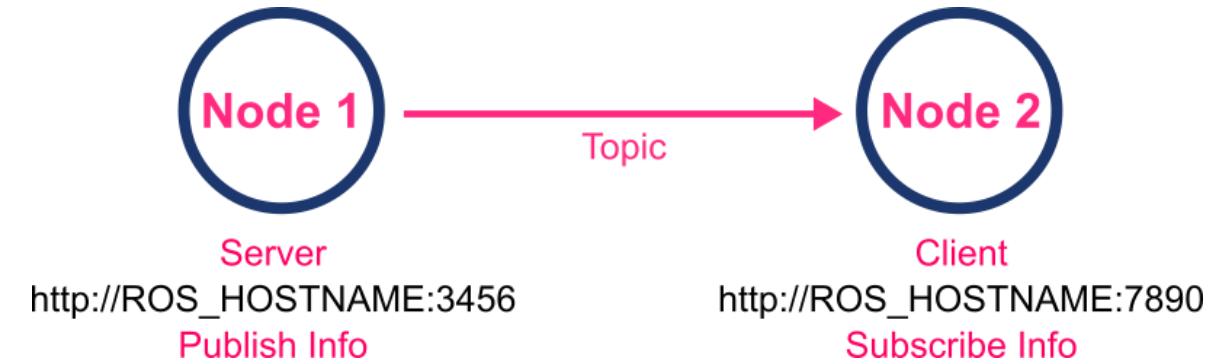
- The publisher node sends the URI address and port number of its TCP server in response to the connection request from the subscriber node.



ROS Communication Protocol

Topics

- Unidirectional
- Used when exchanging data continuously
- uses the same type of message for both publisher and subscriber
- The subscriber node receives the information of publisher node corresponding to the identical topic name registered in the master.



ROS Communication Protocol

Services

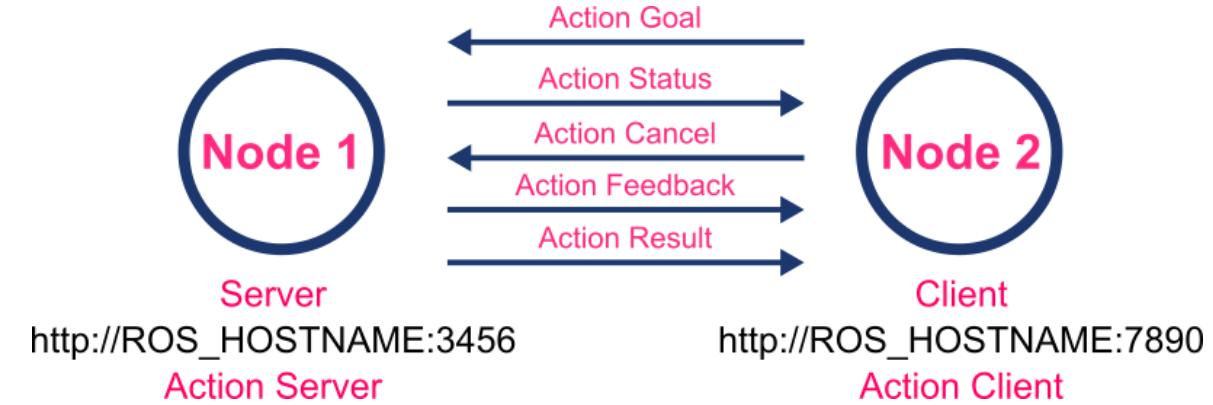
- Synchronous
- Bi-directional
- Used when request processing requests and responds current states
- Unlike the topic, the service is one-time message communication. Therefore, when the request and response of the service are completed, the connection between two nodes will be disconnected.



ROS Communication Protocol

Actions

- Asynchronous
- Bi-directional
- Used when it is difficult to use the service due to long response times after the request or when a feedback value is needed
- Similar to the service where ‘goals’ and ‘results’ correspond to ‘requests’ and ‘responses’ respectively. In addition, the ‘feedback’ is added to report feedbacks to the client periodically when intermediate values are needed.



What is a Workspace?

Any ROS project begins with making a workspace.

In this workspace, you will put all the things related to this particular project.

A workspace is a directory containing these parts:

- Software packages
- Build artifacts
- Logs
- Install artifacts



Catkin is a command line tool to improve the workflow of building, testing and using multiple software packages.

Catkin Workspace Environment

- A set of directories in which a set of related ROS code lives
- Defines context for the current workspace
- Default workspace loaded with

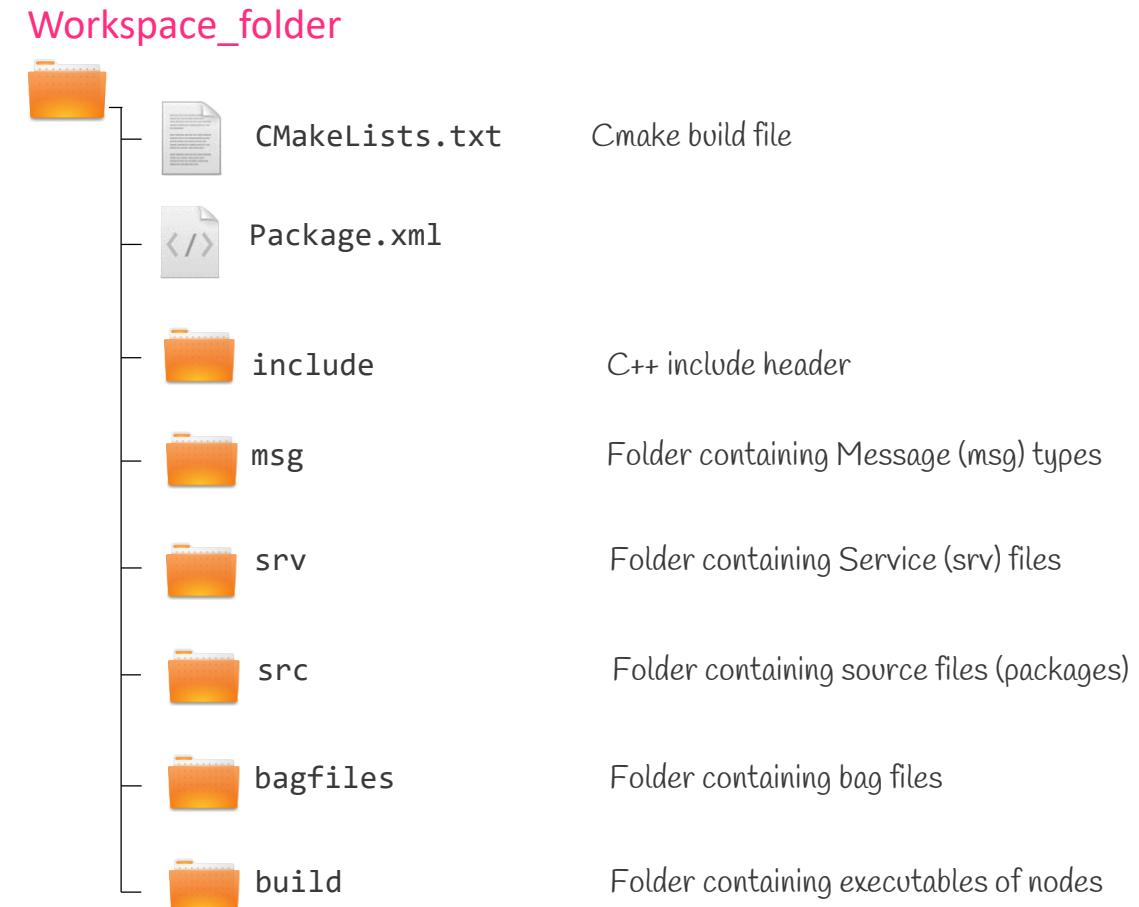
```
> source /opt/ros/kinetic/setup.bash
```

Overlay your catkin workspace with

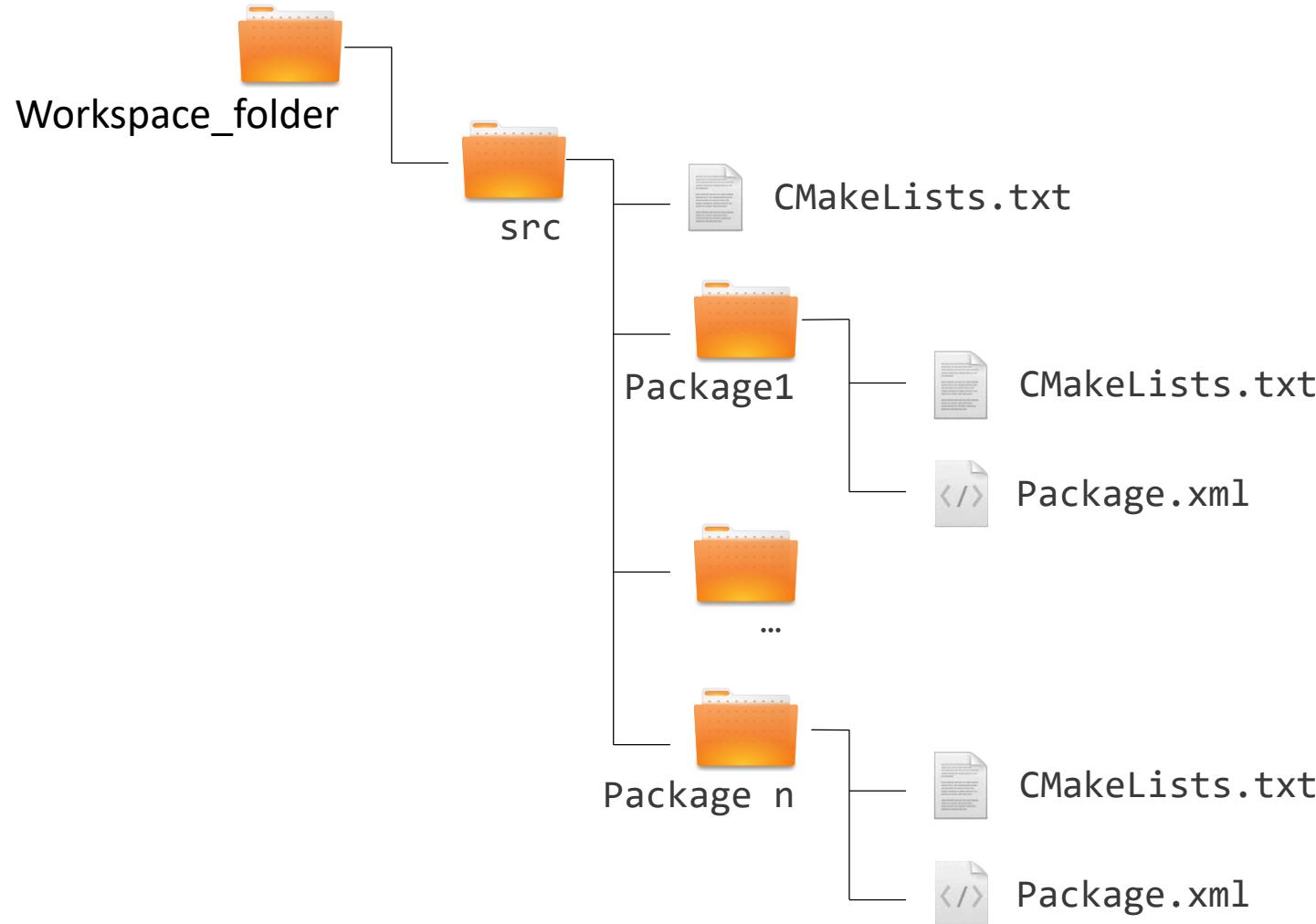
```
> cd ~/catkin_ws  
> source devel/setup.bash
```

Check your workspace with

```
> echo $ROS_PACKAGE_PATH
```



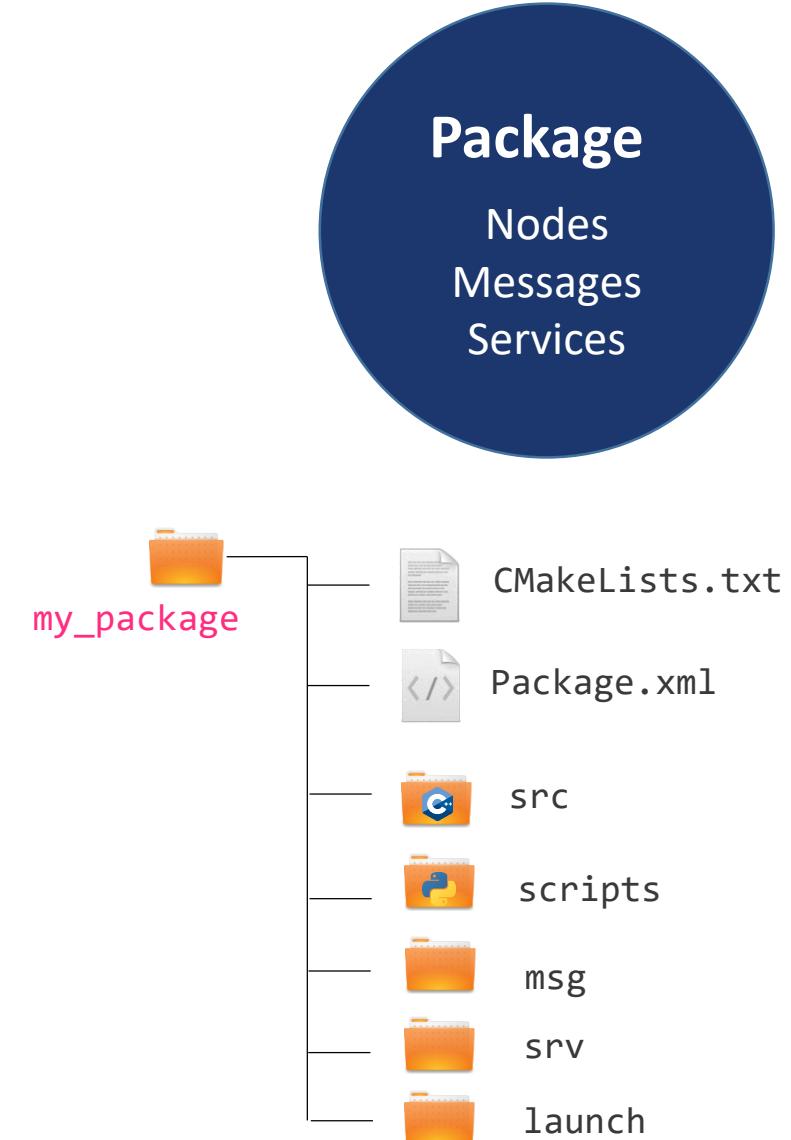
ROS Workspace Environment



ROS Package

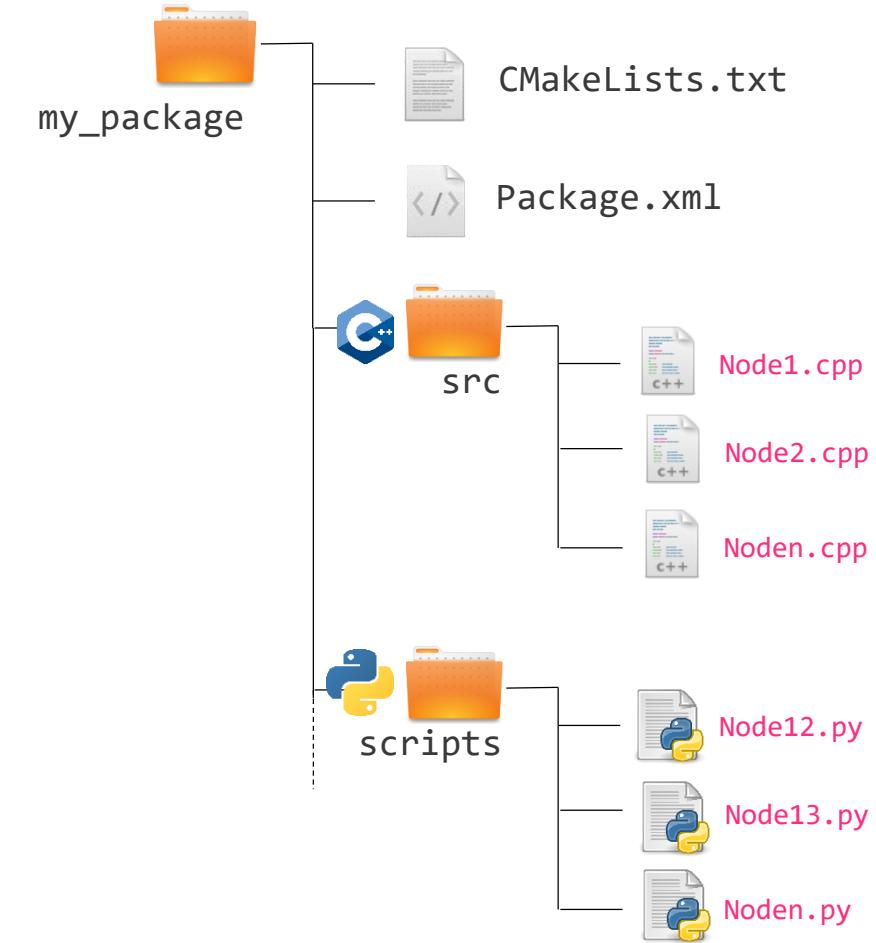
- Software in ROS is organized in *Packages*.
- A package contains one or more *Nodes* and provides a ROS interface
- A ROS package is simply a directory inside a catkin workspace that has a package.xml file in it

- The package must contain a package.xml file.
 - That package.xml file provides meta information about the package.
- The package must contain a CMakeLists.txt which uses catkin.
- Each package must have its own folder



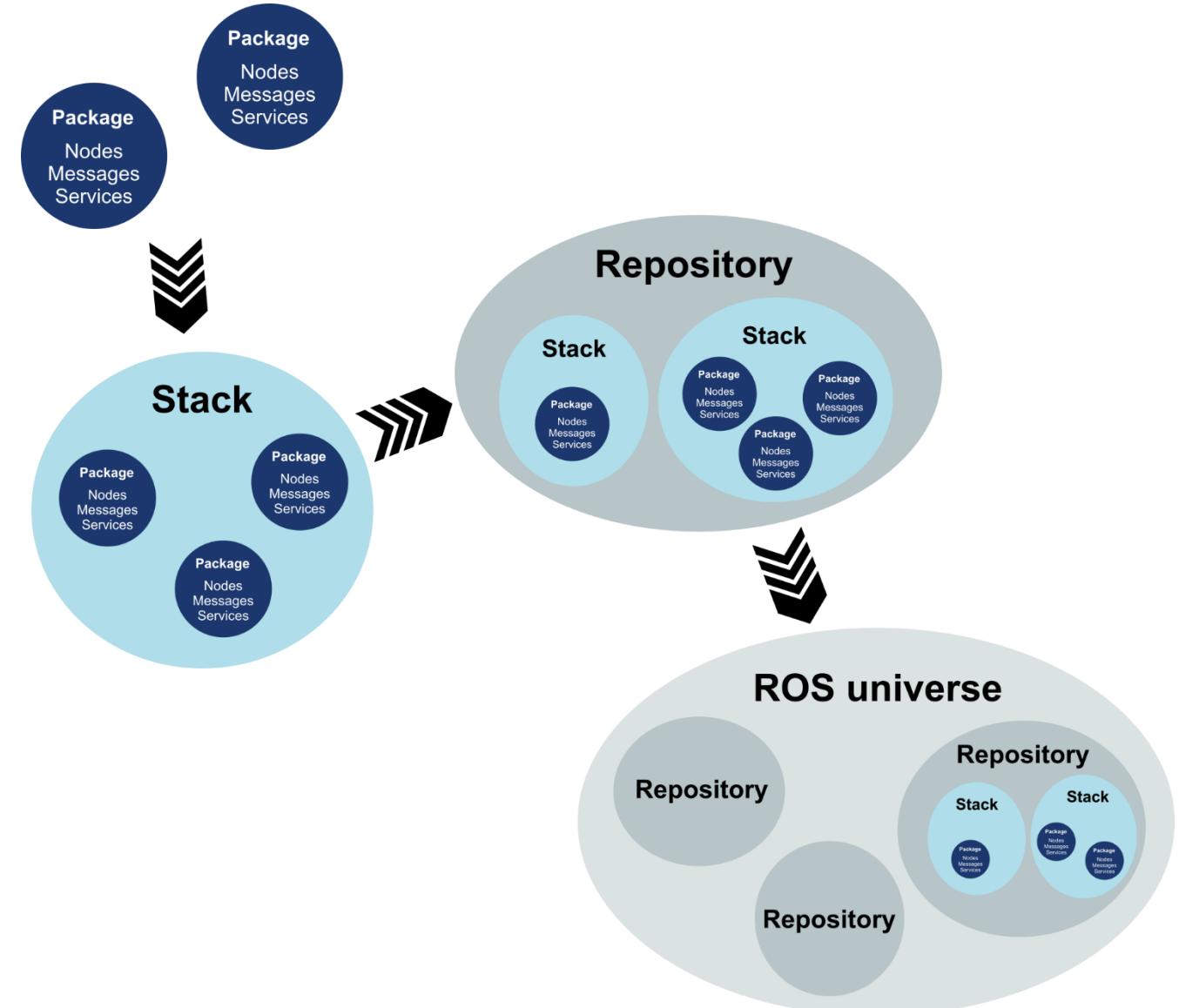
ROS Node

- Node = file containing the code of the executable
- File written in **C++** should be saved in the source folder **src** of the package
- File written in **Python** should be saved in the scripts folder **scripts** of the package



ROS universe

- Most of ROS packages are hosted in GitHub



Catkin Build System

- *catkin* is the ROS build system to generate executables, libraries, and interfaces. (*i.e* make and compile *packages*)
- Collection of CMake macros and Python scripts
- A build system is responsible for generating 'targets' from raw source code that can be used by an end user.
- You can have multiple ROS workspaces, but you can only work in one of them at any one time



Navigate to your catkin workspace with

```
> cd ~/catkin_ws
```

Create a package with

```
> catkin_create_pkg [package_name] [dependencies ...]
```

Whenever you build a new package, update your environment !!!

```
> source ./devel/setup.bash
```

catkin Build System

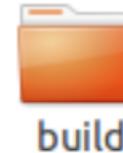
The catkin workspace contains the following spaces

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).

If necessary, clean the entire build and devel space with

```
> catkin clean
```

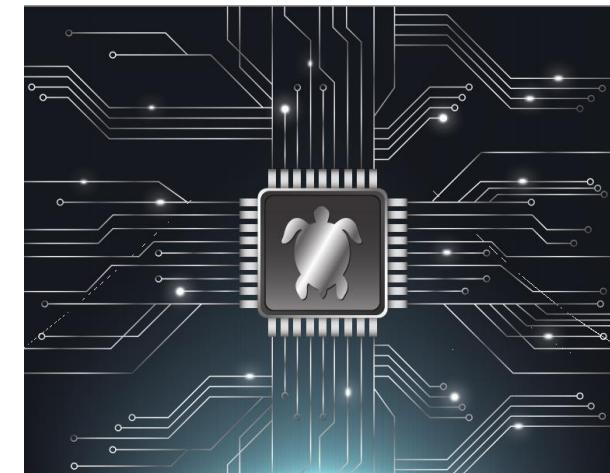
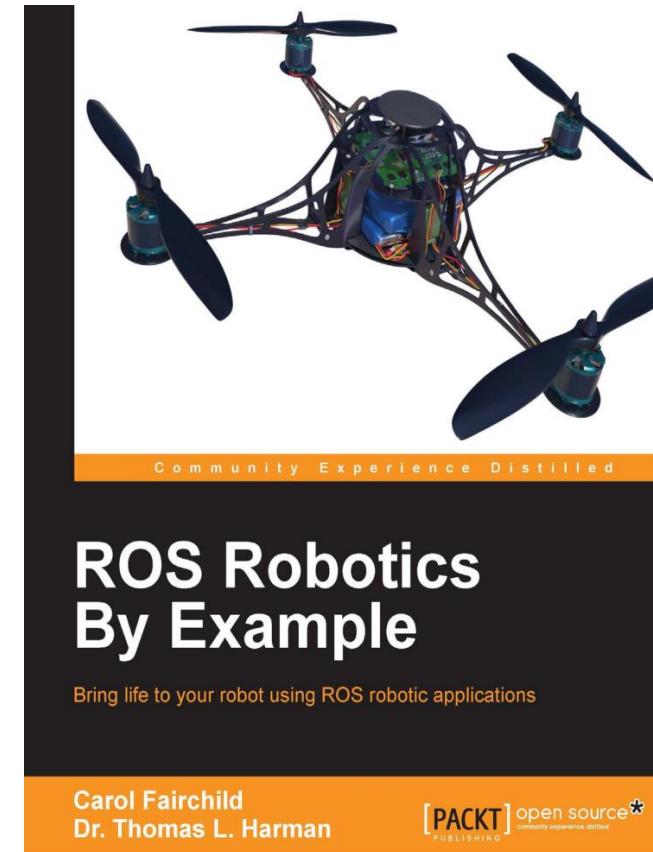
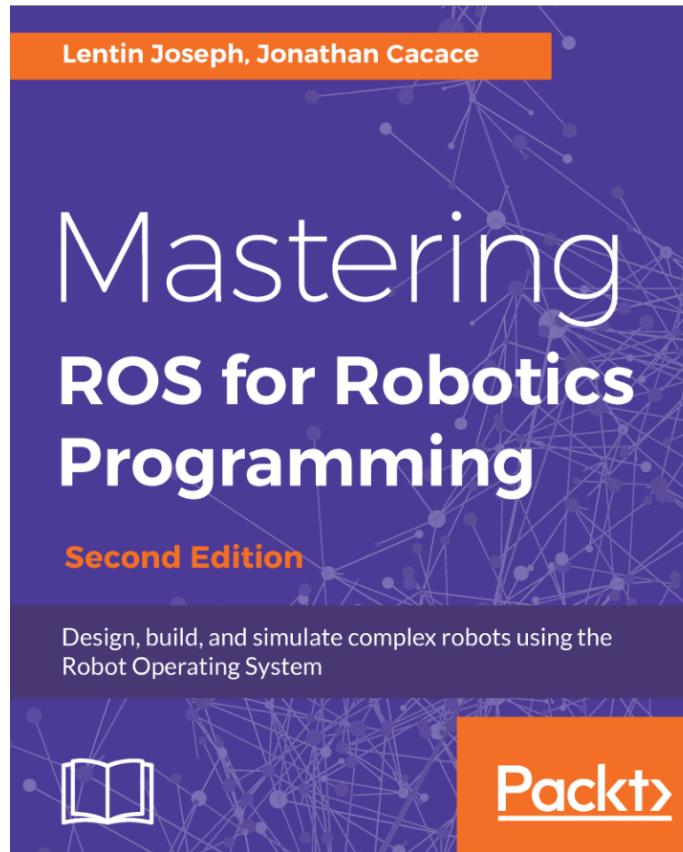
More info

<http://wiki.ros.org/catkin/workspaces>

Further References

- **ROS Wiki**
 - <http://wiki.ros.org/>
- **Installation**
 - <http://wiki.ros.org/ROS/Installation>
- **Tutorials**
 - <http://wiki.ros.org/ROS/Tutorials>
- **Available packages**
 - <http://www.ros.org/browse/>
- **ROS Cheat Sheet**
 - <https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/>
 - https://kapeli.com/cheat_sheets/ROS.docset/
- **ROS Best Practices**
 - https://github.com/leggedrobotics/ros_best_practices/wiki
- **ROS Package Template**
 - https://github.com/leggedrobotics/ros_best_practices/tree/master/ros_package_template

Relevant books



Contact Information

University Savoie Mont Blanc

Polytech' Annecy Chambery
Chemin de Bellevue
74940 Annecy
France

<https://www.polytech.univ-savoie.fr>

Lecturer

Luc Marechal (luc.marechal@univ-smb.fr)
SYMME Lab (Systems and Materials for Mechatronics)



SYMME