
# INFO 802

## Master Advanced Mechatronics

**Luc Marechal**

2021

ROS

## Introduction
## Lecture 1

UNIVERSITÉ SAVOIE MONT BLANC

POLYTECH ANNECY-CHAMBÉRY

# Prerequisites

- Ubuntu installation
  - Ubuntu on a Windows Virtual machine
  - Or Ubuntu **mate** on a Raspberry Pi
  - Or Ubuntu on a a Linux machine

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

- Installation of ROS Noetic Ninjemys


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

# What is ROS ?

- ROS (**R**obot **O**perating **S**ystem) 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

**Not an operating system !**

**More info**
https://roboticsbackend.com/what-is-ros/

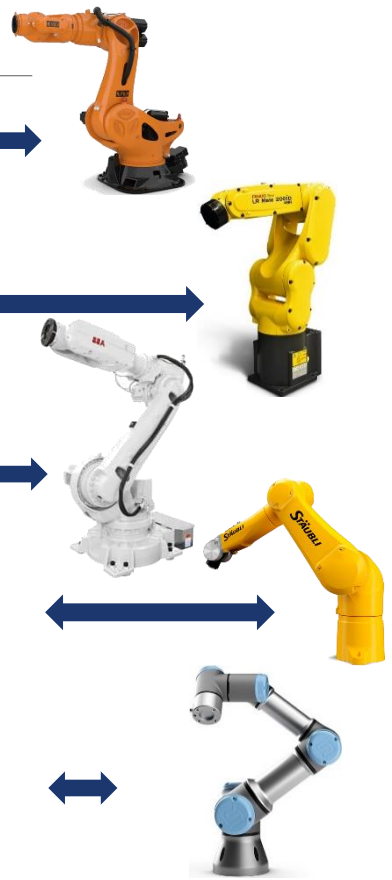
# What is a Middleware ?


- 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.

# Why using ROS ?

- 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 |



ROS

**Standardisation** of communications

Coders can develop applications in any environment without worrying about future connectivity with the rest of the system.

# 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

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

STANFORD

Willow Garage

Open Source Robotics Foundation

**More info - source**

http://www.willowgarage.com

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

# 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

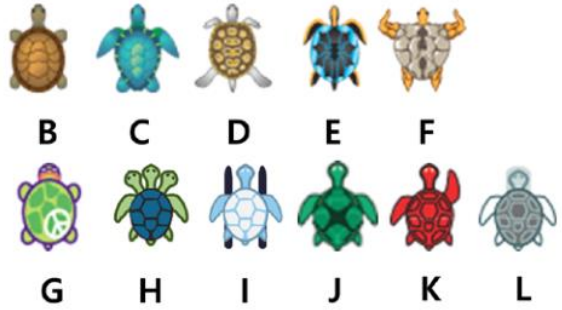


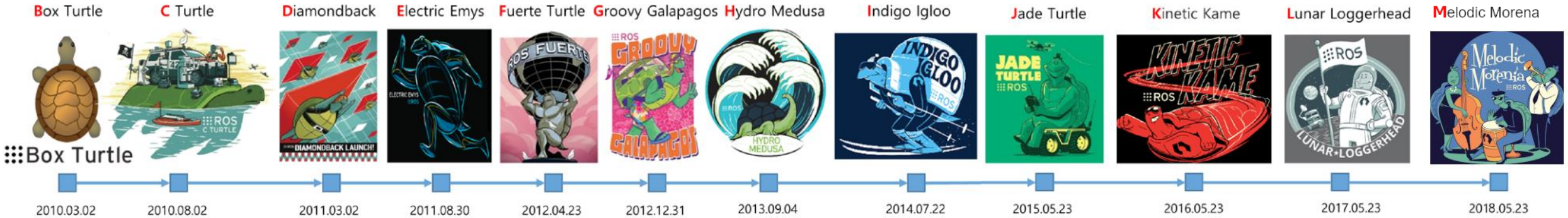




# Release Schedule

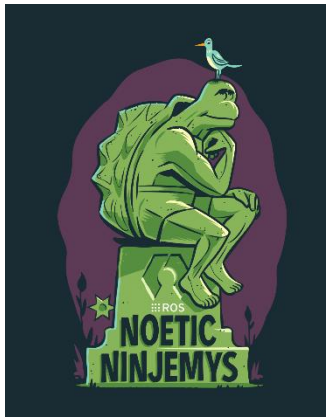
- There is a ROS release every year in May.
  - Releases on even numbered years are LTS release, supported for 5 years.
  - Releases on odd numbered years are normal ROS releases, supported for 2 years.
  - ROS releases will drop support for EOL Ubuntu distributions, even if the ROS release is still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




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

# ROS Latest Releases




**Noetic Ninjemys**

## Select Your Platform

**Supported:**

Ubuntu Focal amd64 armhf arm64

Debian Buster amd64 arm64

Source installation

**Experimental:**

Windows 10 amd64

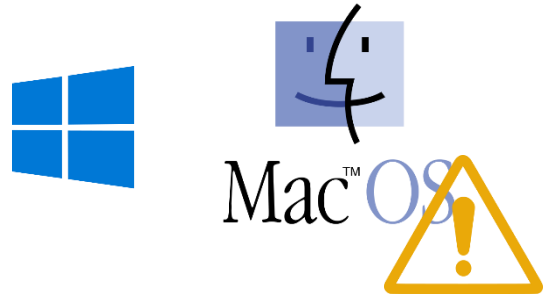
Arch Linux Any amd64 i686 arm armv6h armv7h aarch64

(ROS2)


**Foxy Fitzroy (ROS2)**

# ROS Supported Platforms

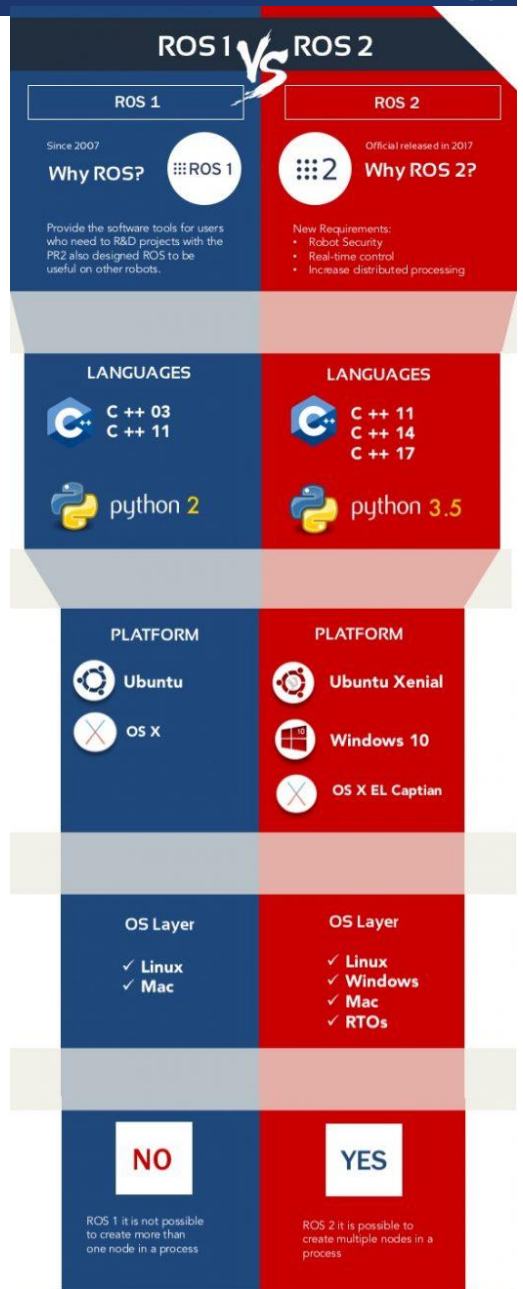
- ROS is currently supported only on **Ubuntu**
  - Fedora
  - Gentoo
  - Arch Linux
  - other variants such as Windows and Mac OS X are considered experimental (will be supported on ROS 2.0)



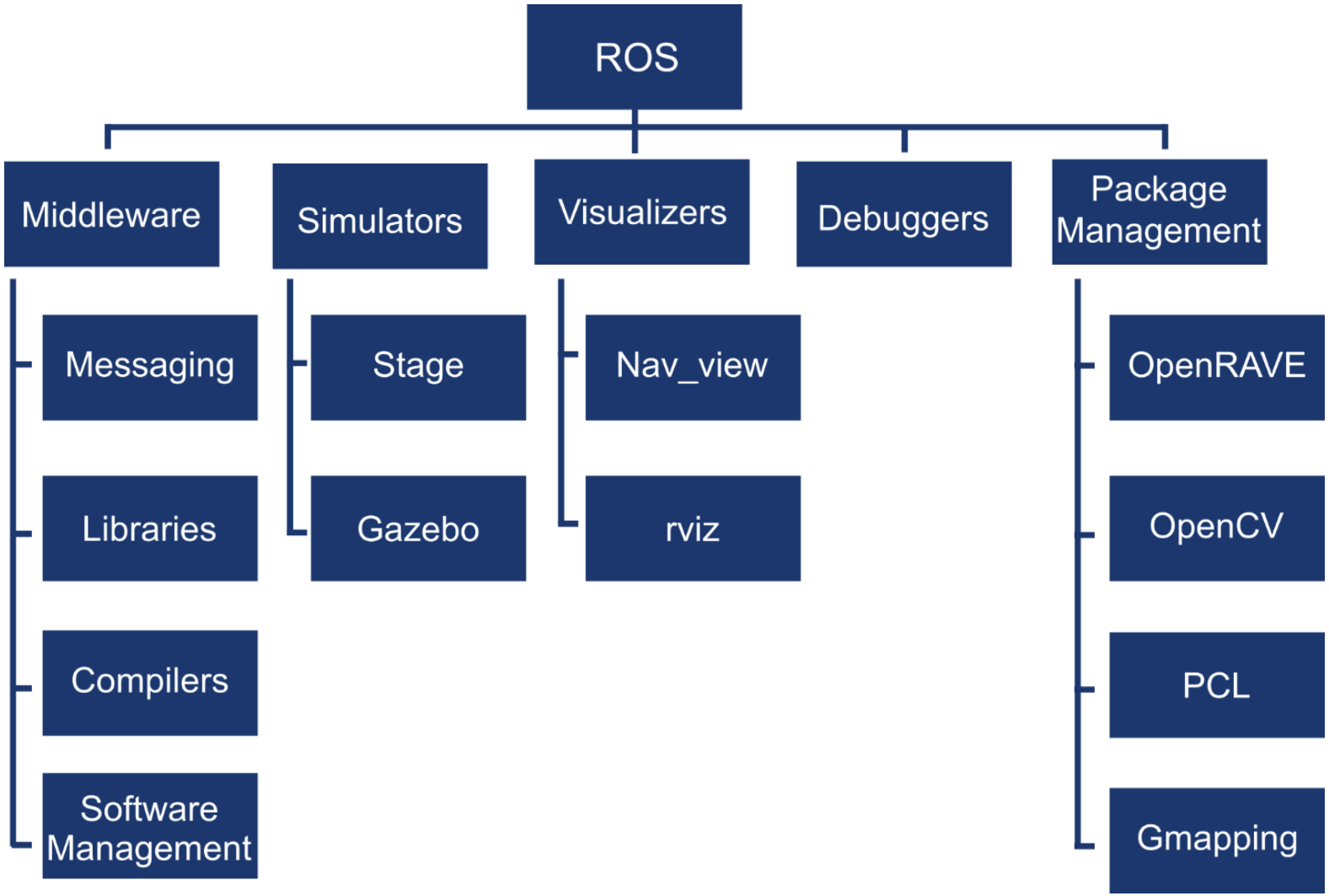
# ROS 2

- Since ROS was started in 2007, a lot has changed in the robotics and ROS community.

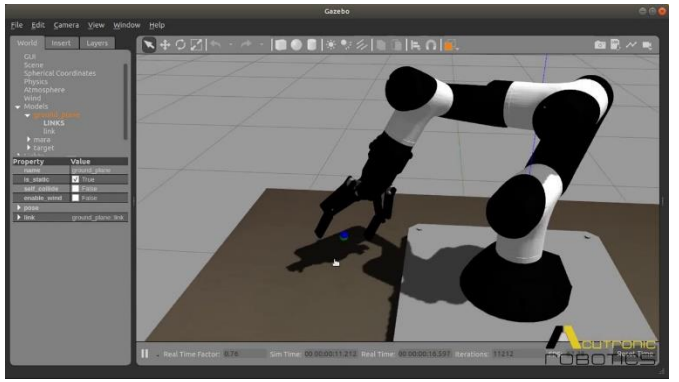
- The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn't.



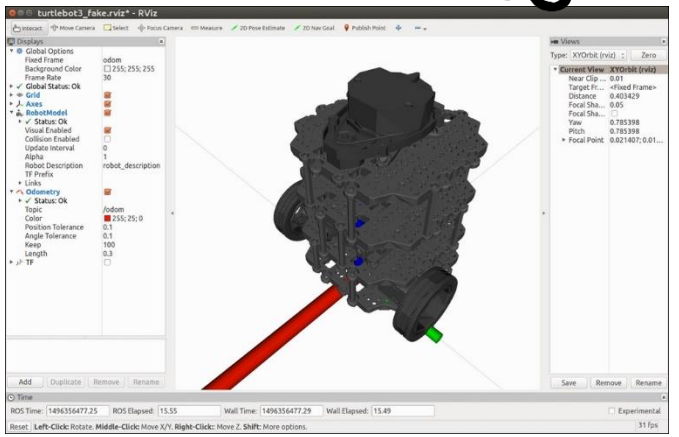
# ROS Components




GAZEBO



RViz



# 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 |
| Propriety/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.).

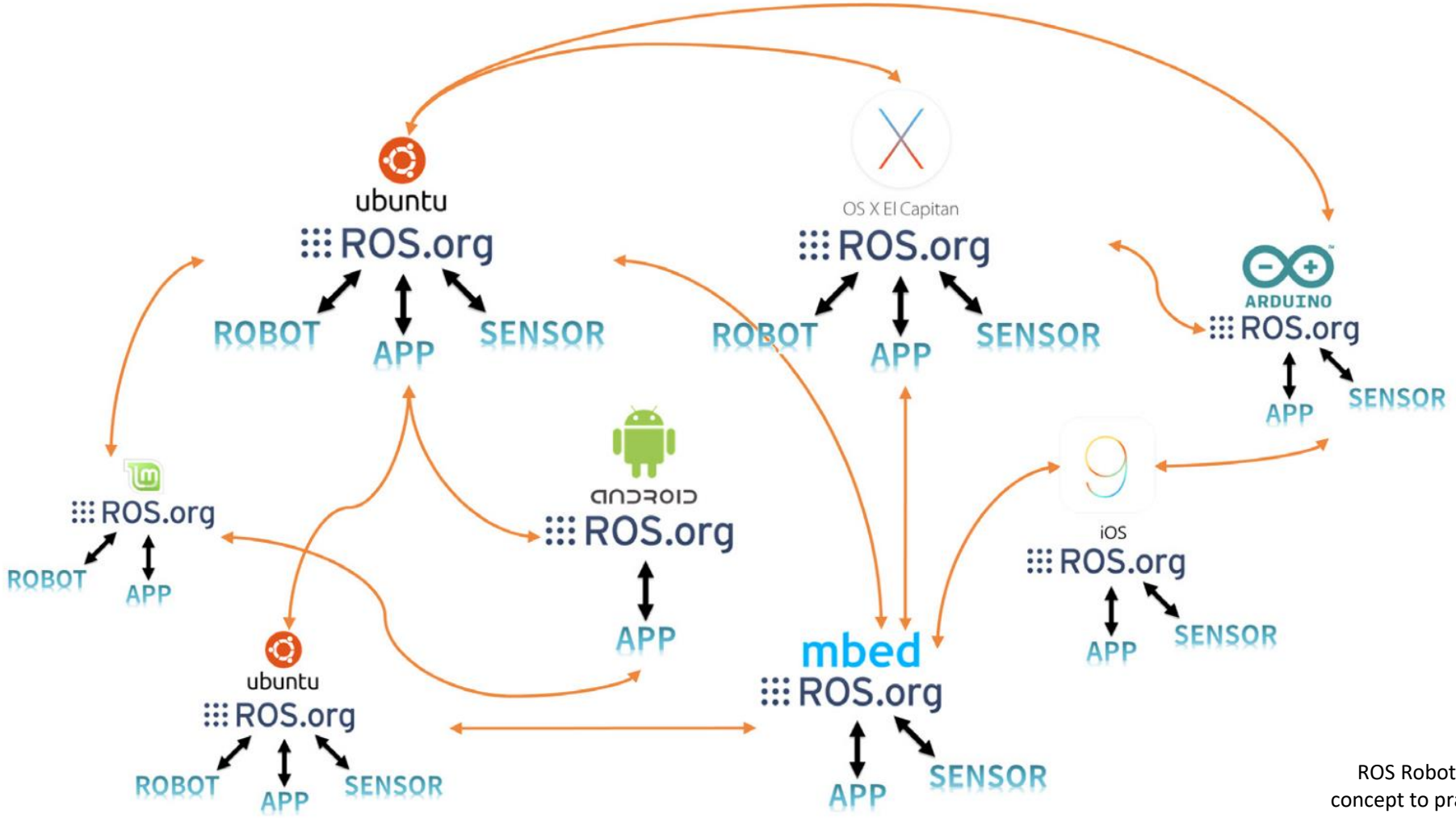
- **Thin**

  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



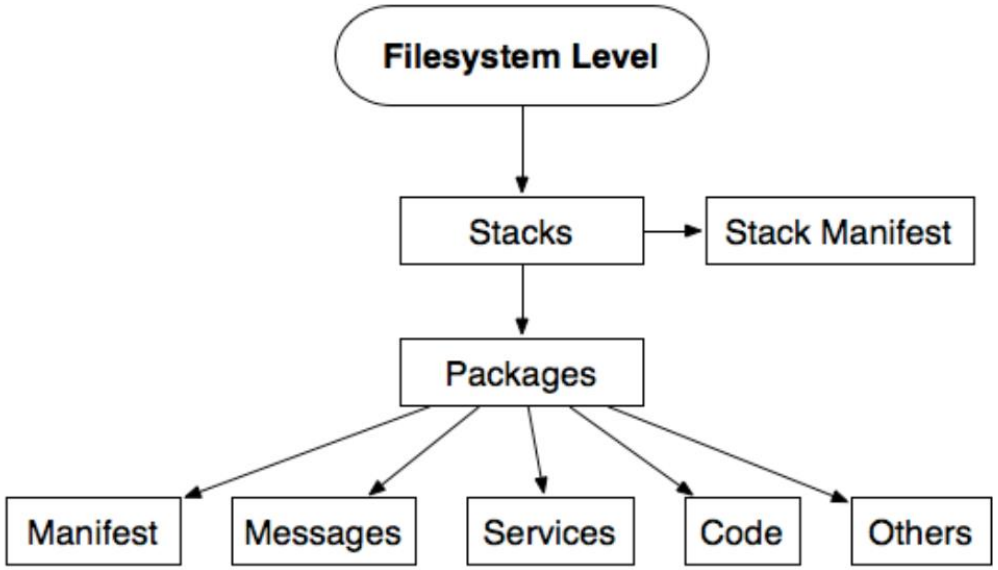

**More info - source**

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

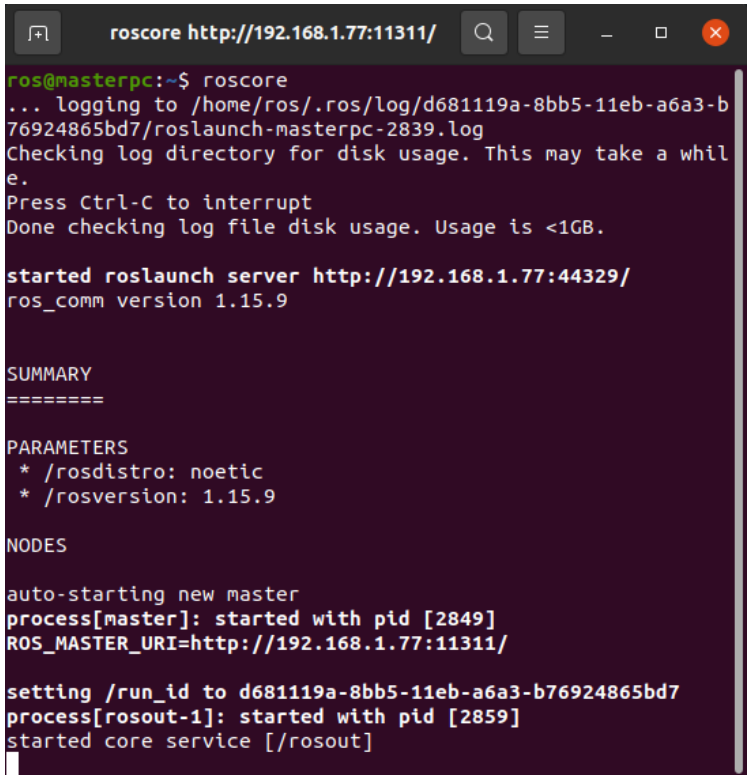
- **ROS Topics**
  - Asynchronous 'stream-like' communication
  - TCP/IP or UDP Transport
  - Strongly-typed (ROS.msg spec)
  - Can have one or more publishers
  - Can have one or more subscribers

- **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

# ROS Overview

- Launch roscore
- Launch Node 1
- Launch Node 2

```
> roscore
```



# ROS Master

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


Start a master with

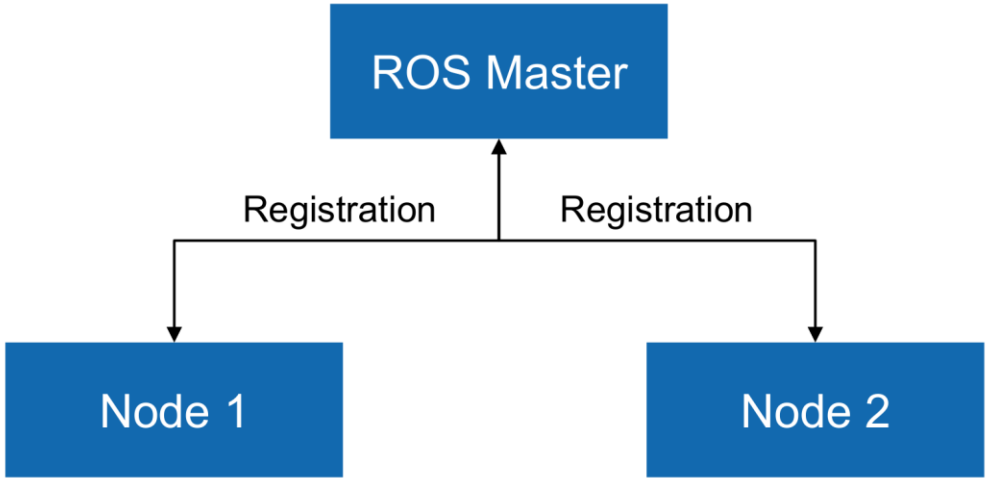
```
> roscore
```

- 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

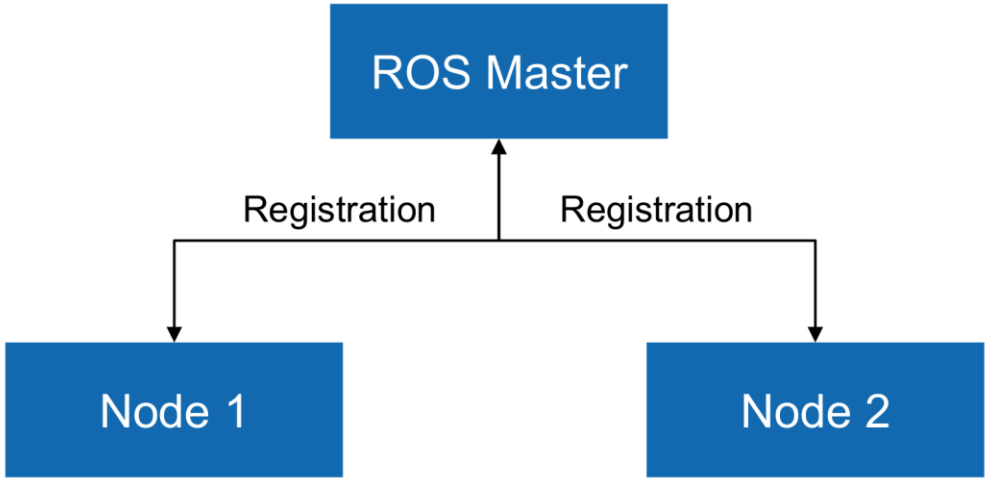
```
> rosrun [package_name] [node_name]
```

See active node with

```
> rosnode list
```

Retrieve information about a node with

```
> rosnode info [node_name]
```




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

# 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

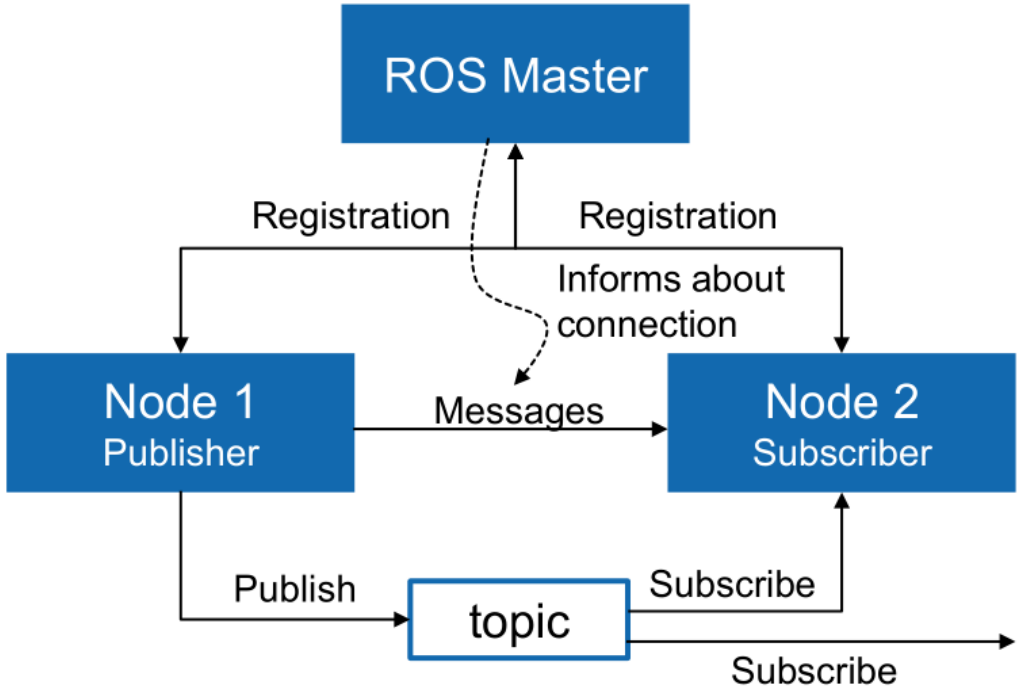
```
> rostopic list
```

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

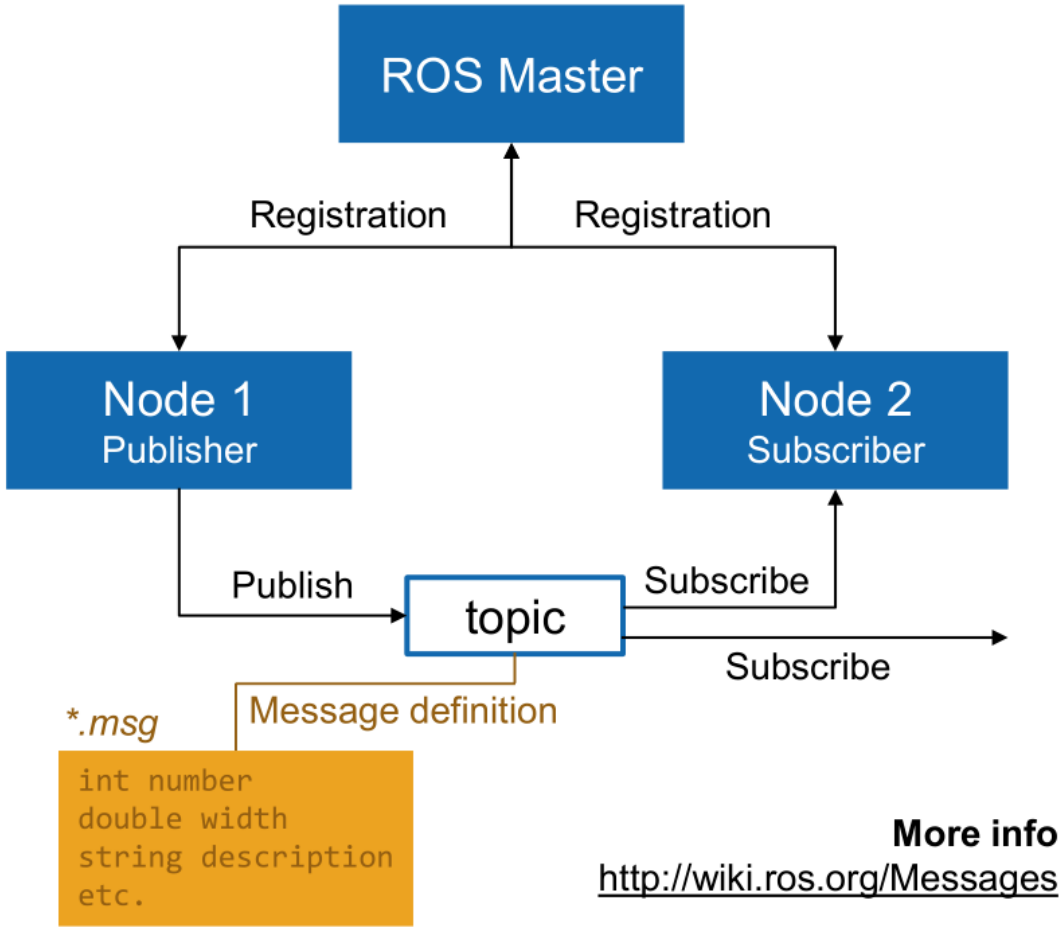
- Data structure defining the *type* of a topic
- Comprised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in **.msg* files

See the type of a topic

```
> rostopic type /topic
```

Publish a message to a topic

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




**More info**
http://wiki.ros.org/Messages

# Messages Communication Flow

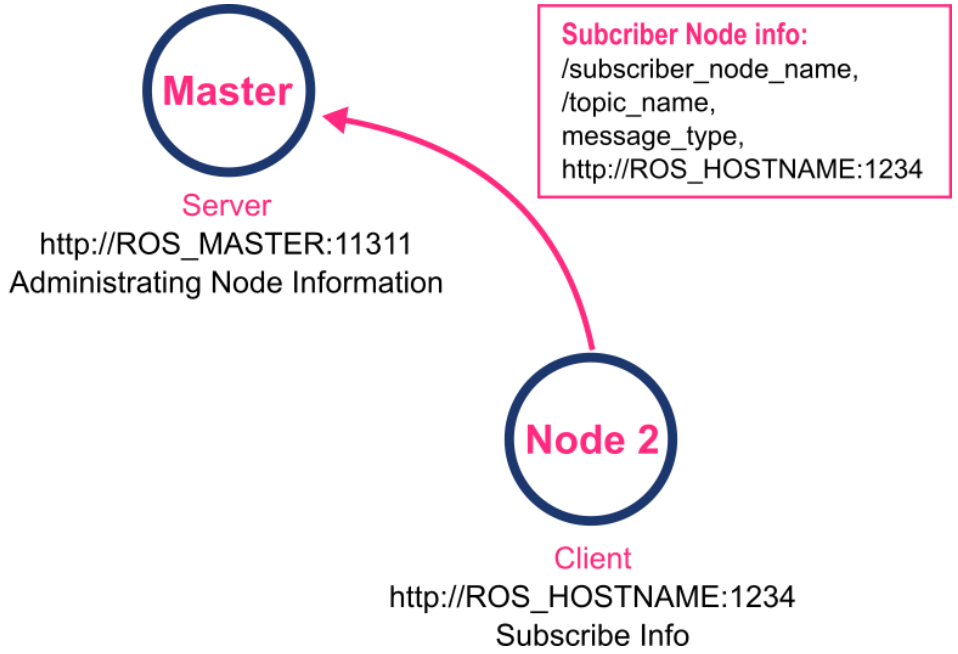
## Running the Master

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

# Messages Communication Flow

## Running the Subscriber Node

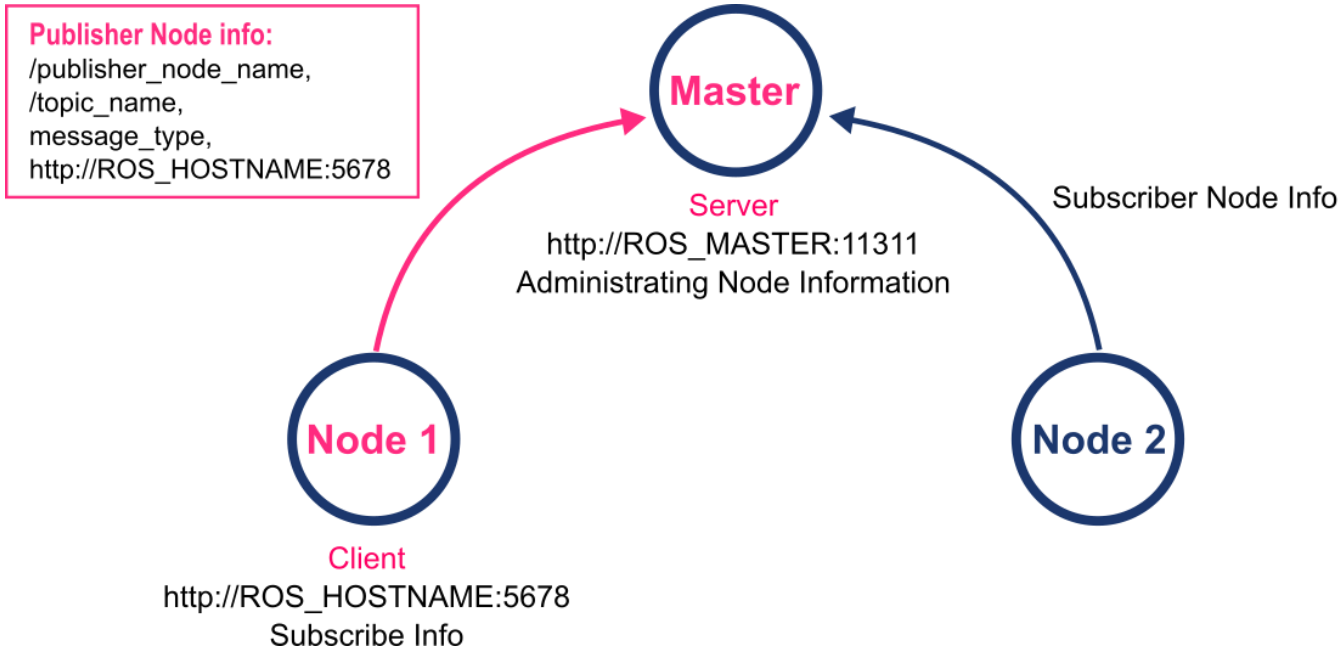
- The subscriber node registers its node name, topic name, message type, URI address, and port with the master



# Messages Communication Flow

## Running the Publisher Node

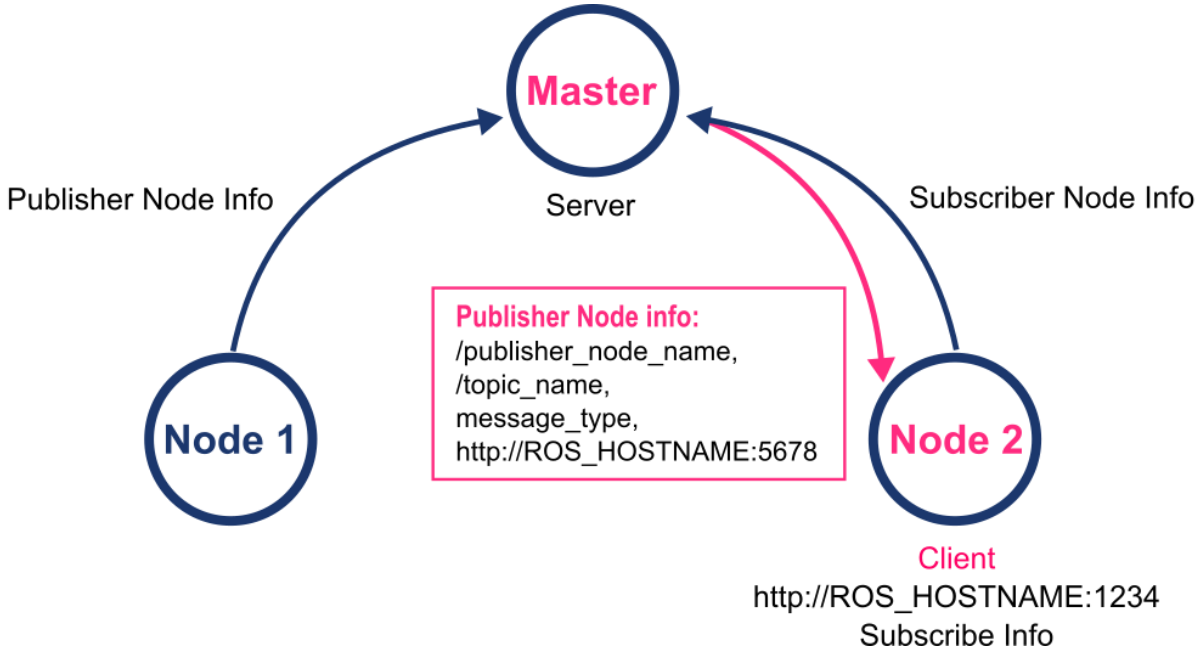
- The publisher node registers its node name, topic name, message type, URI address and port with the master.



# Messages Communication Flow

## Providing Publisher Information

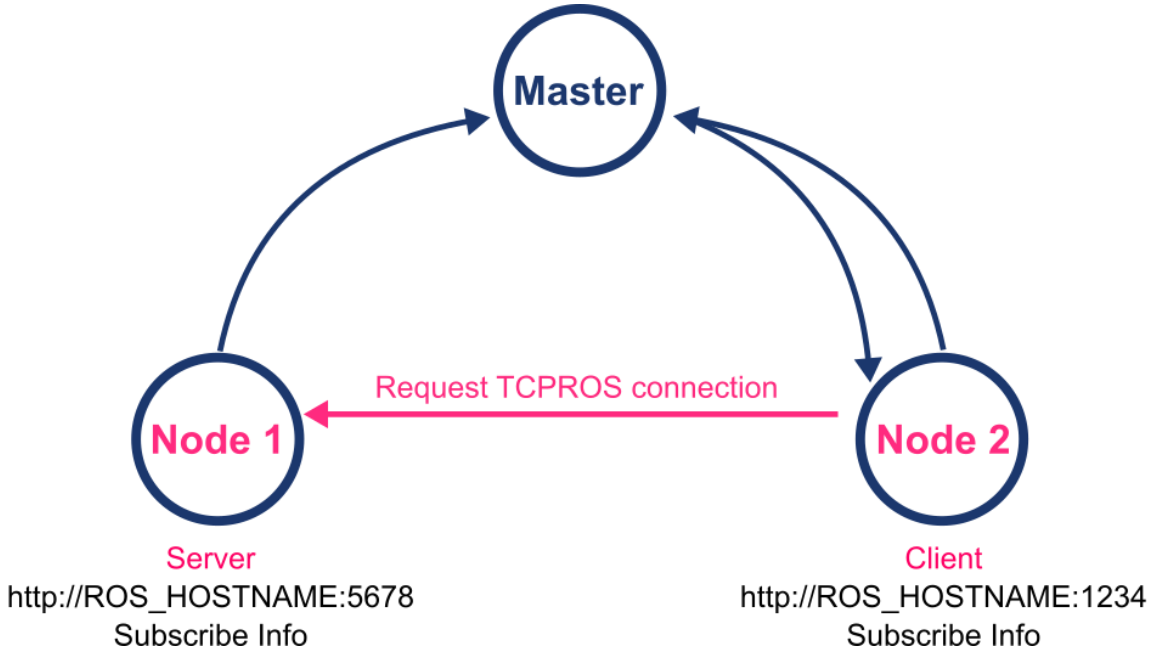
- The master distributes information such as the publisher's name, topic name, message type, URI address and port number of the publisher to subscribers that want to connect to the publisher node



# Messages Communication Flow

## Connection Request from the Subscriber Node

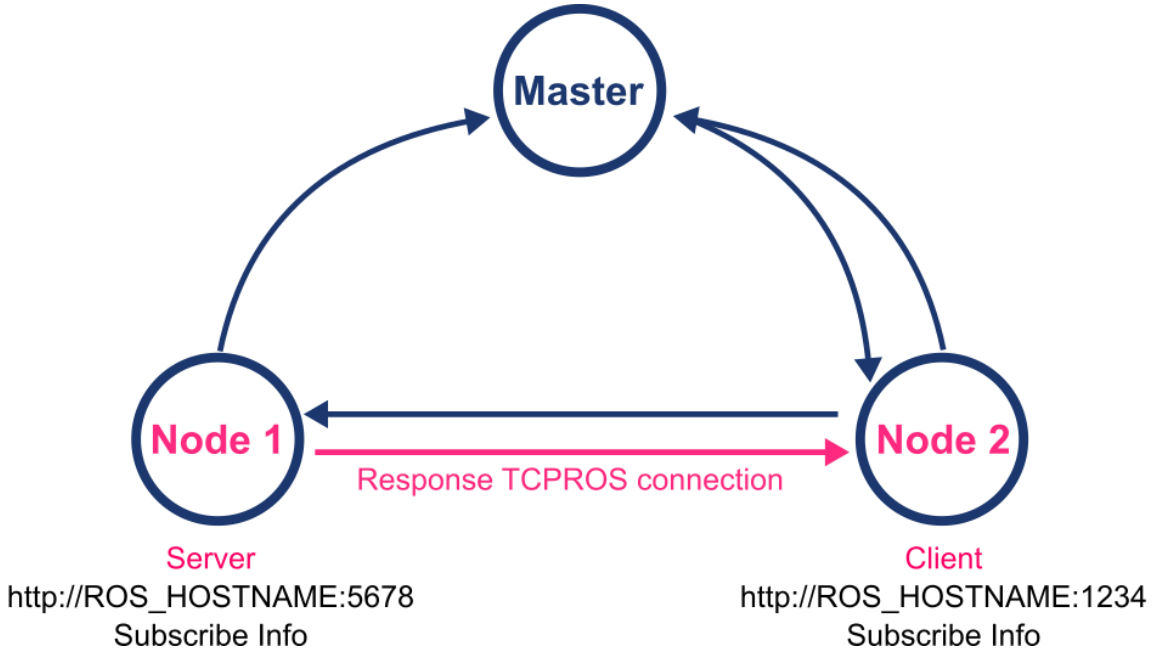
- The subscriber node requests a direct connection to the publisher node based on the publisher information received from the master.



# Messages Communication Flow

## Connection Response from the Publisher Node

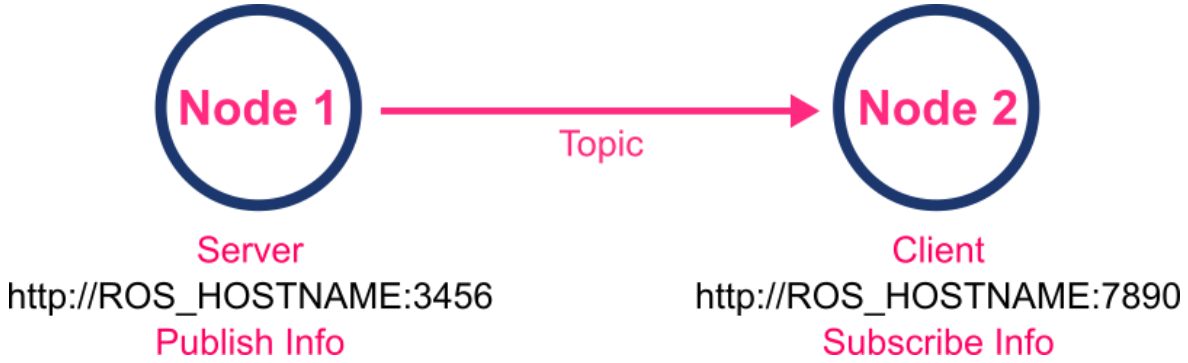
- The publisher node sends the URI address and port number of its TCP server in response to the connection request from the subscriber node.



# 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

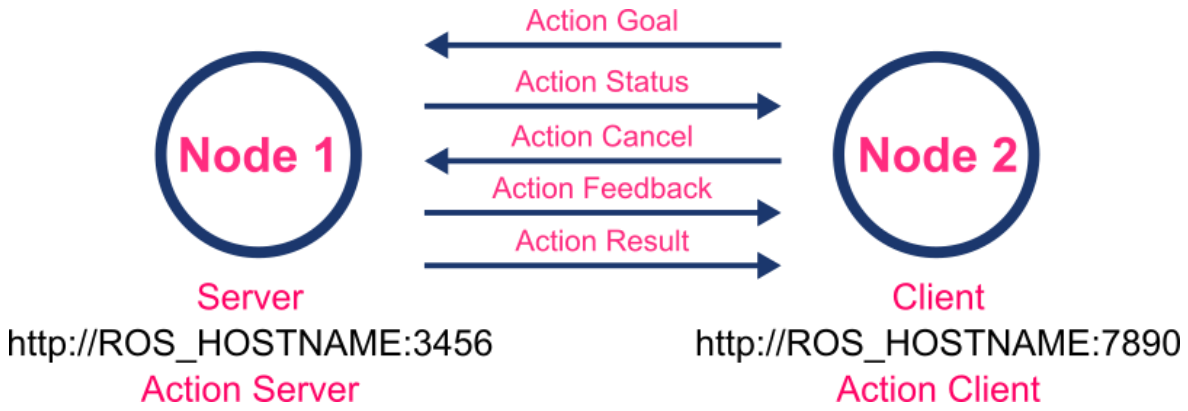
- Synchronous
- Bi-didirectional
- Used when request processing requests and responds current states


- 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.



# How to decide what to use ?

- **Topics** : especially for stream of data
- **Services** : execution of fast tasks
- **Actions** : execution of fast tasks that need to be tracked and should be preempted in some cases

# 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
```

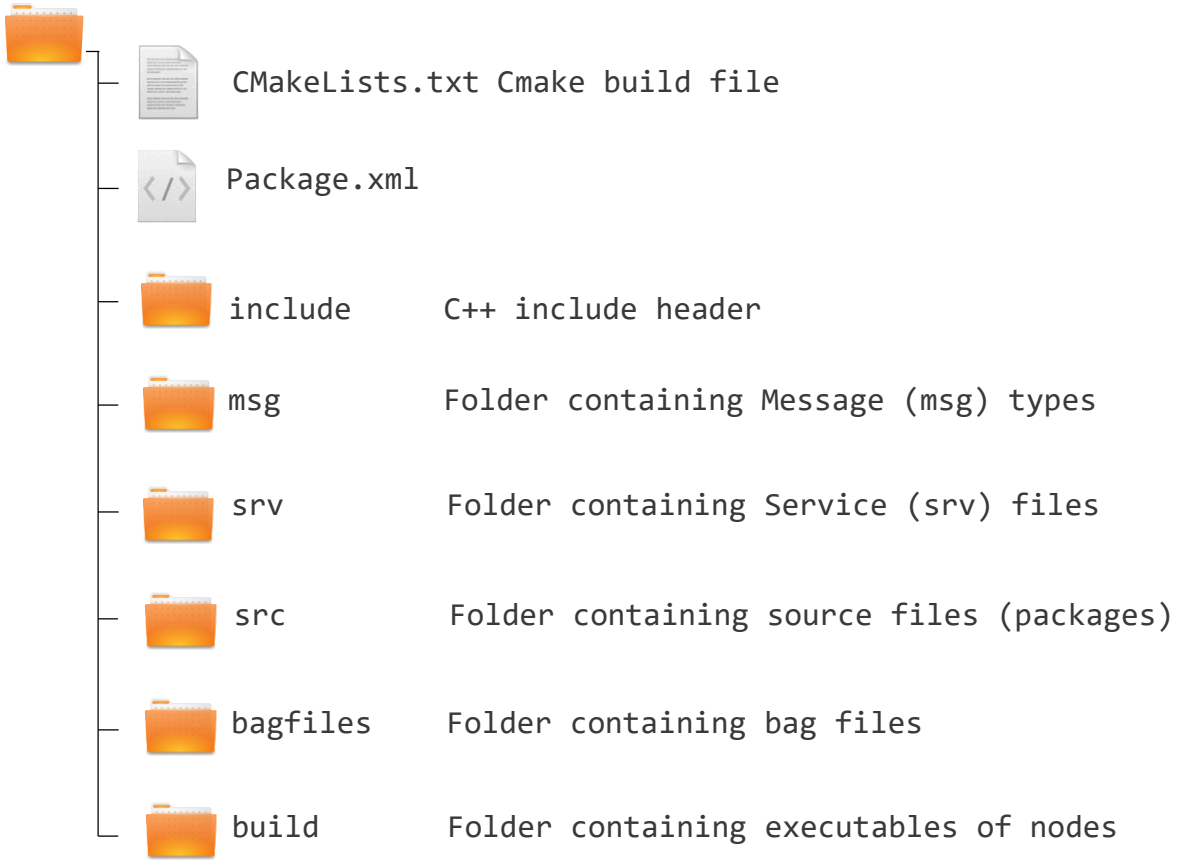
Overlay your catkin workspace with

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

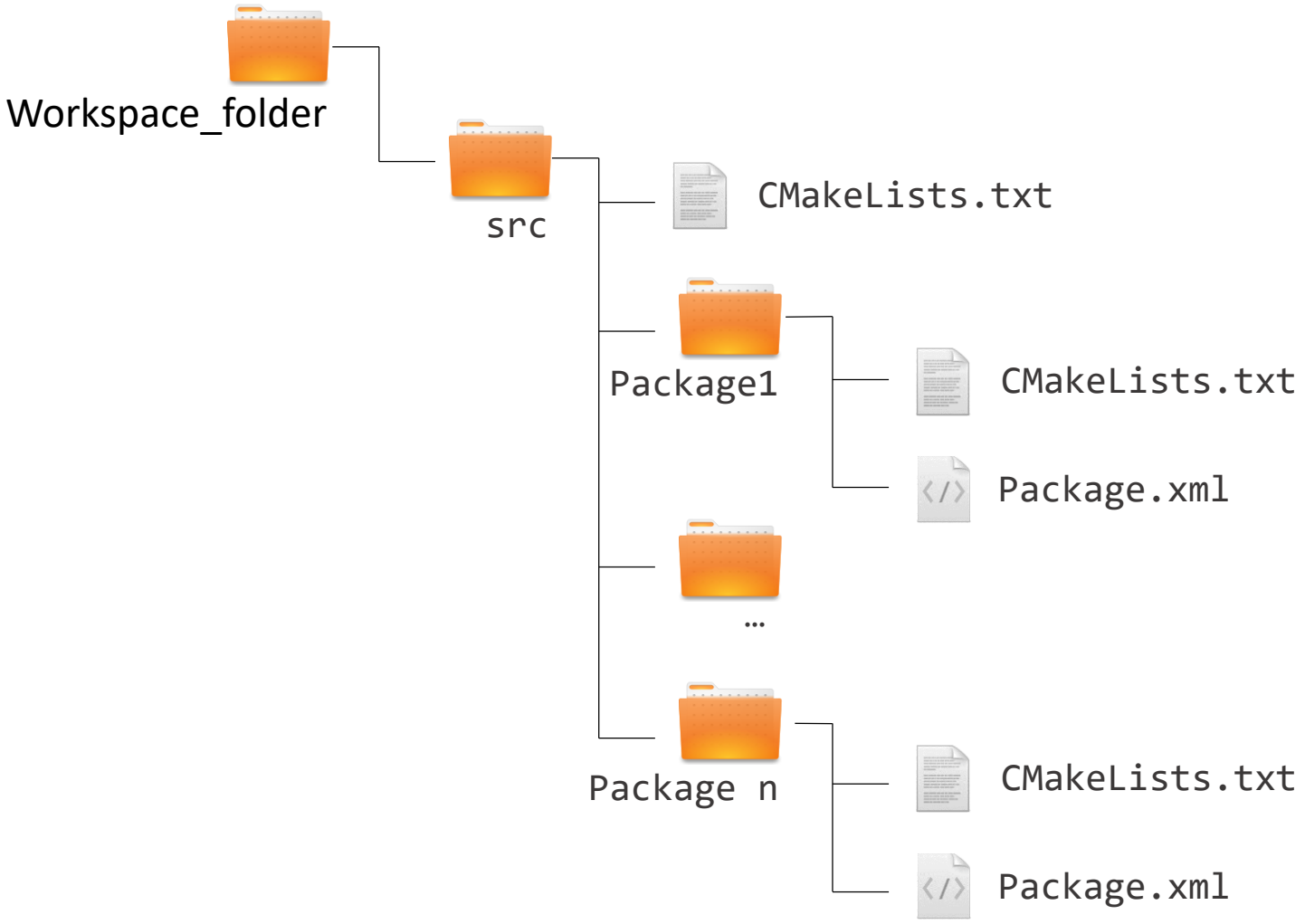
Check your workspace with

```
> echo $ROS_PACKAGE_PATH
```

Workspace_folder



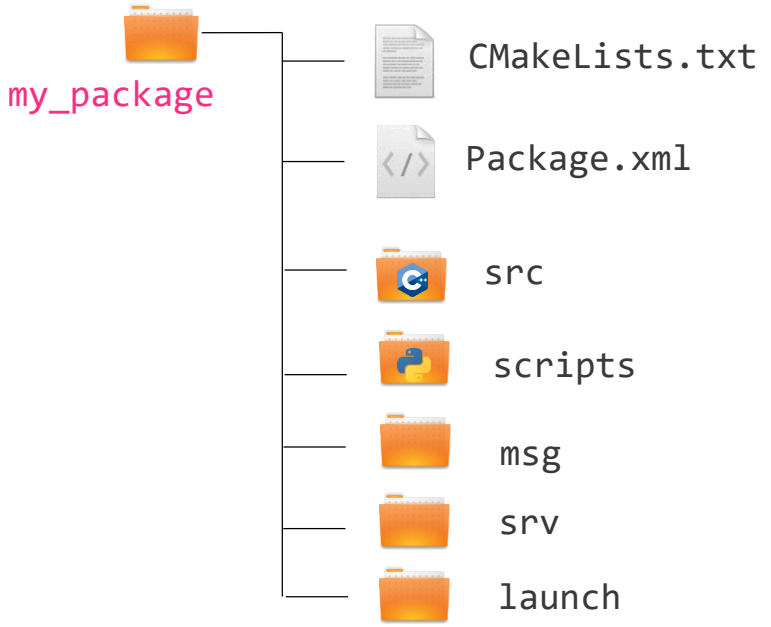
# ROS Workspace Environment



# ROS Package

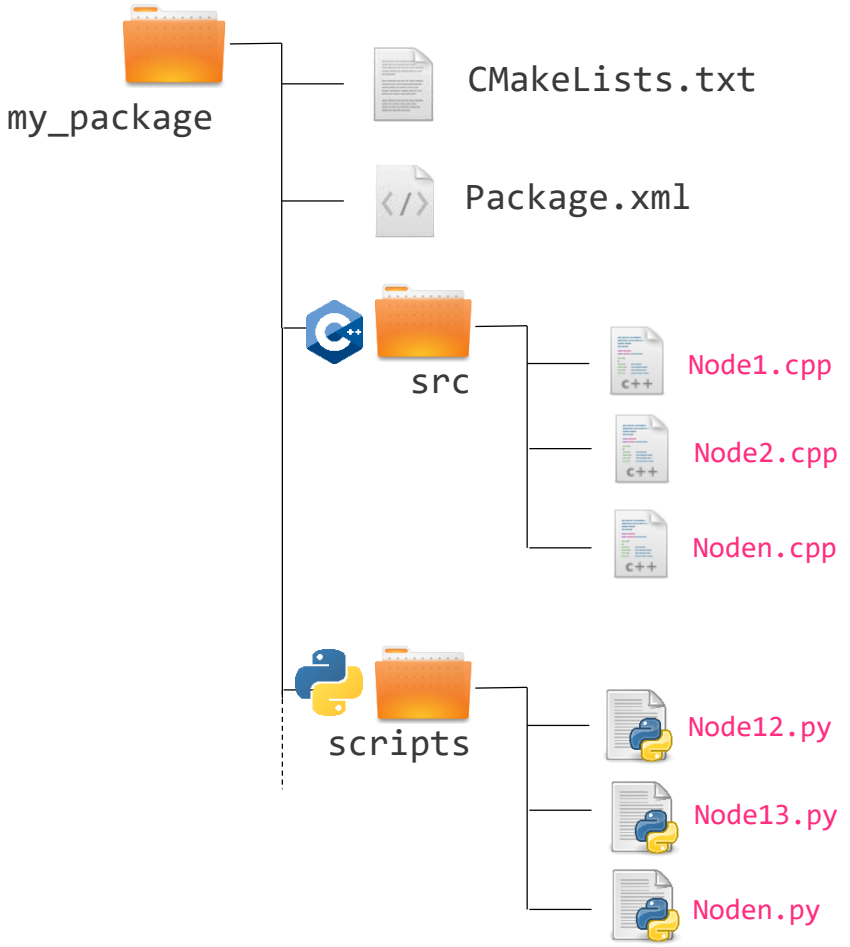
- Software in ROS is organized in *Packages*.
- A package contains one or more *Nodes* and provides a ROS interface
- A ROS package is simply a directory inside a catkin workspace that has a package.xml file in it


- 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



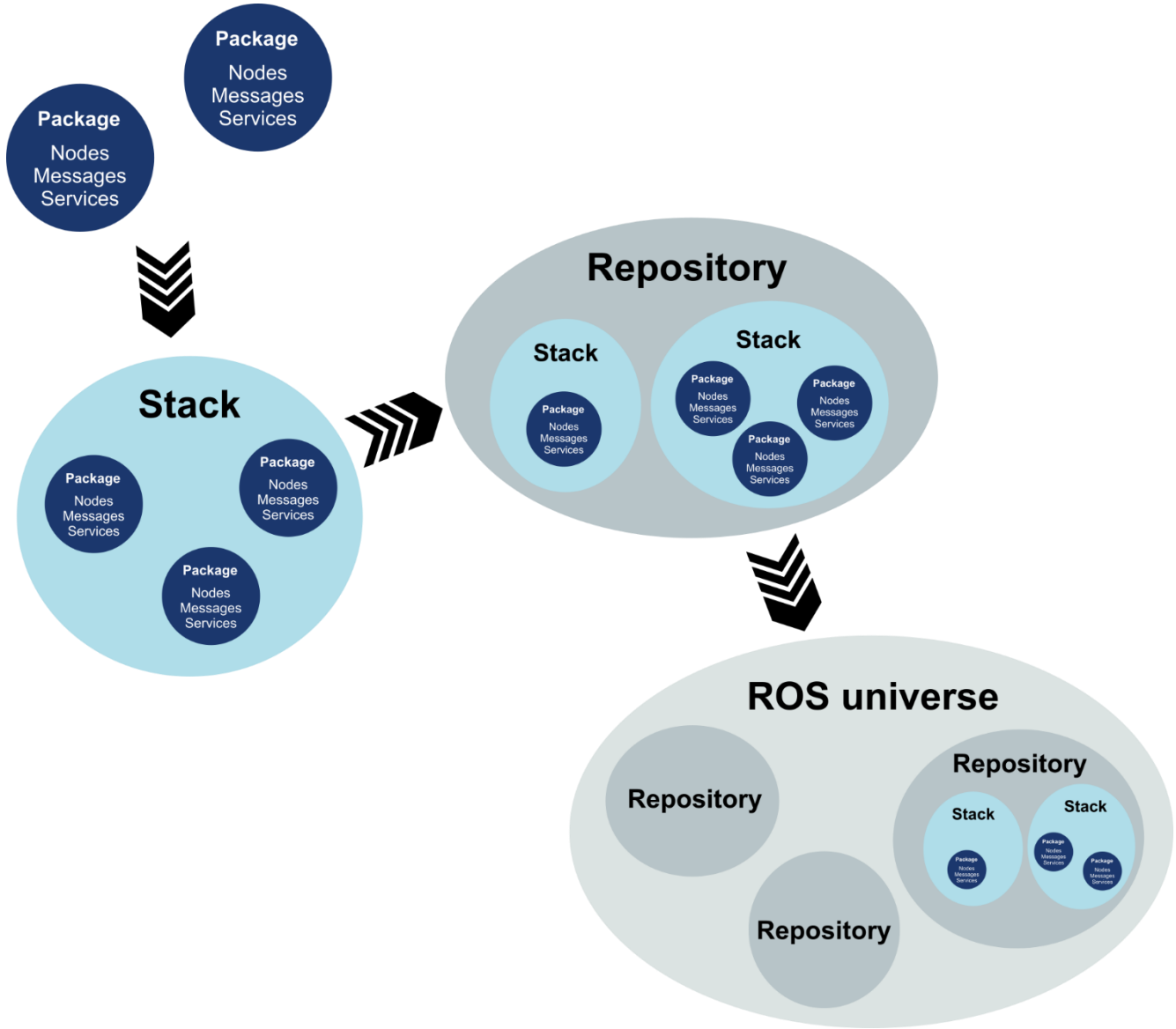
# ROS Node

- 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 *srcipts* 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

# ROS Environment

## Set up the environment

- In order to work properly, ROS uses the **setup.bash** and **setup.sh** files
- It is located in in the following directory: */opt/ros/noetic/*

- The main function of these files is to set environment variables used by ROS and other apps.
- During the installation of ROS, you will see that you are prompted to source one of several setup.*sh files, or even add this 'sourcing' to your shell startup script

- If you are ever having problems finding or using your ROS packages make sure that you have your environment properly setup. Sourcing these setup.*sh files might help sometimes.

# ROS Environment

## Set up the environment:


- You will need to run source /opt/ros/noetic/setup.bash on every new shell you open to have access to the ROS commands, unless you add this line to your bash startup file (~/.bashrc)

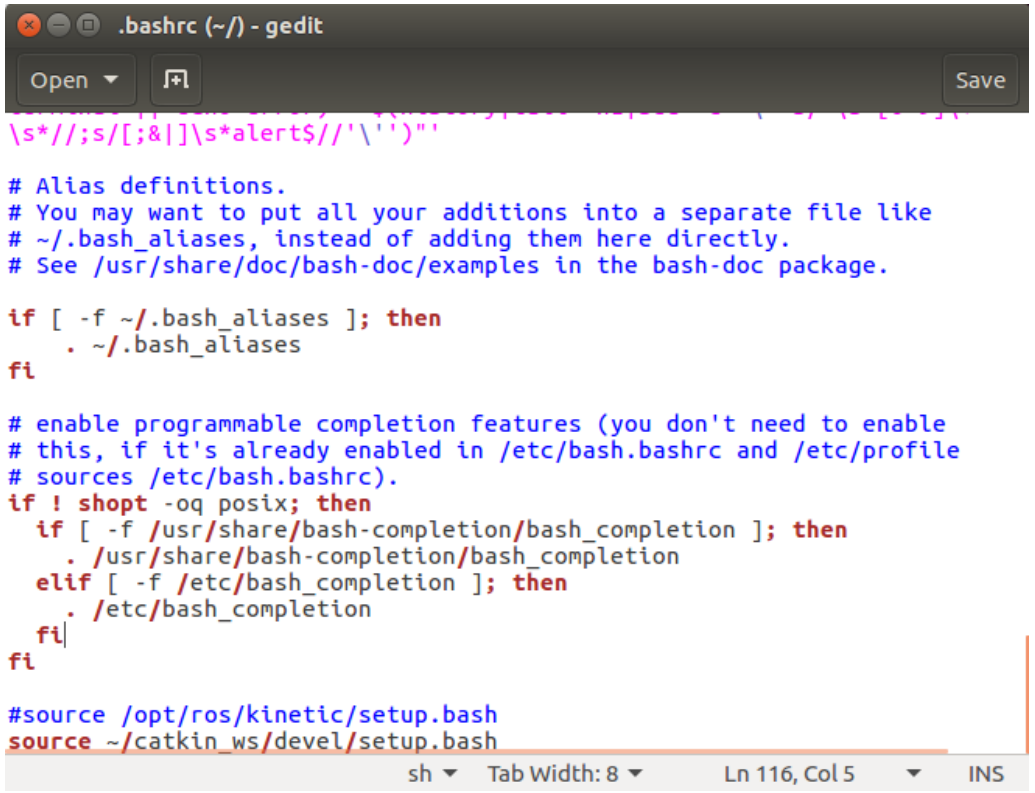
- This will allow you to run roscore from any directory in your terminal window. To do so, we will modify the .bashrc.

Edit *.bashrc* file

```
> gedit ~/.bashrc
```

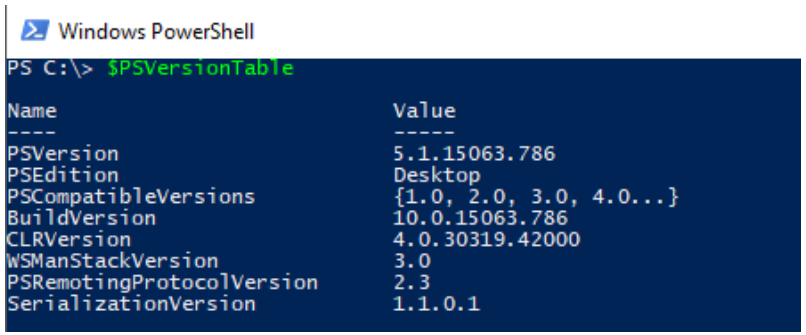
add the the line at the bottom

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

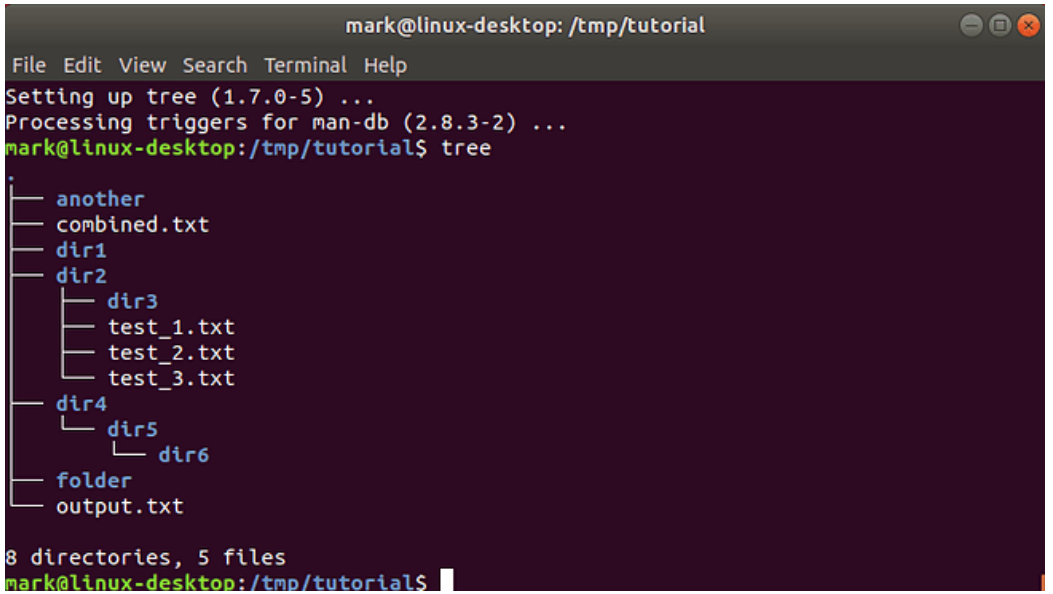


# Bash (Unix shell)


- Bash is the Unix basic **shell** used in the terminal. (the $ character is the default prompt.)
- The shell is an interface between the user and the operating system.
- It uses either a **command-line interface** (CLI) or graphical user interface (GUI) to control the computer.
- The CLI used in Ubuntu is **Terminal**




an example of CLI shell for Windows is PowerShell



# .bashrc


- *.bashrc* is a script file hidden is the /home /<username> directory.
- When you open a new terminal window by pressing CTRL + ALT + T or simply to open a new terminal tab, bash reads and executes commands from *~/.bashrc*, if that file exists.
- In particular, it reads the environment variables that are in the file.

Edit the file

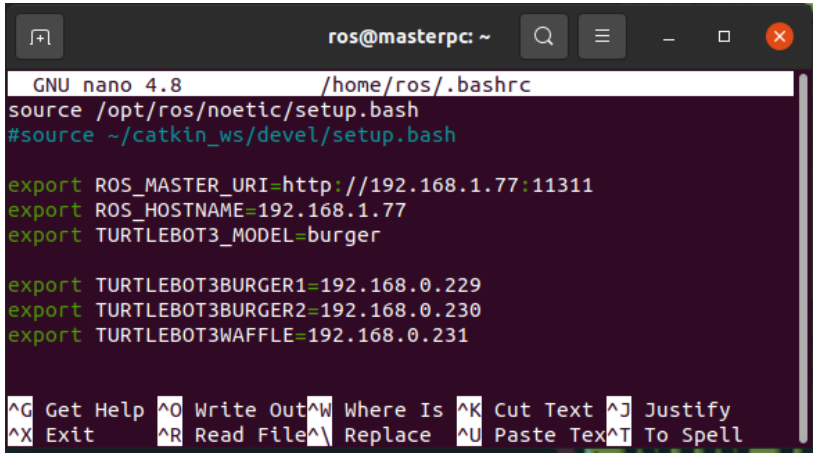
```
> nano ~/.bashrc
```

Remarks

`nano` is an easy to use command line text editor for Unix OS

`~` is a shortcut for the /home/<username> directory

`.file` means the file is hidden




In this ~/.bashrc file, the variable:
- TURTLEBOT3_MODEL has the value “burger”
- TURTLEBOT3BURGER1 has the value “192.168.0.229”

# Linux Command : *Source*

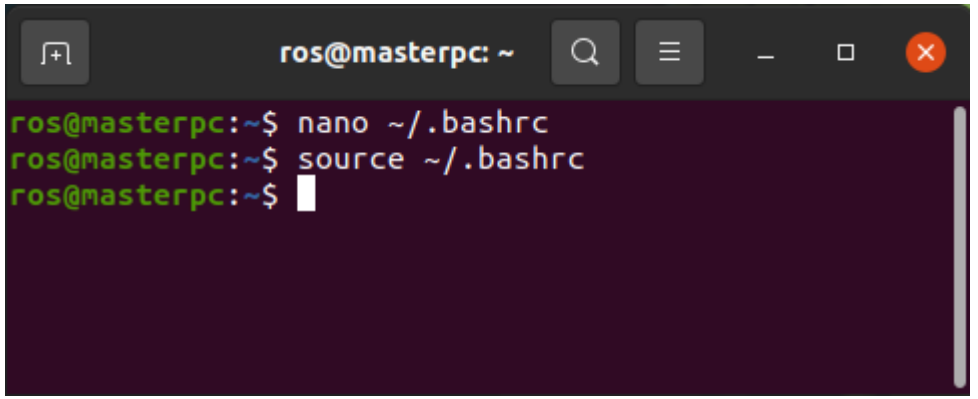

- When a file is sourced, the lines of code in the file are executed as if they were printed at the command line.
- It updates functions and variables in the file for the curent shell
- Any changes in /home/<username>/.bashrc file will only be taken into acount after sourcing

Sourcing a file

```
> source file_name.sh
```

Or

```
> . file_name.sh
```

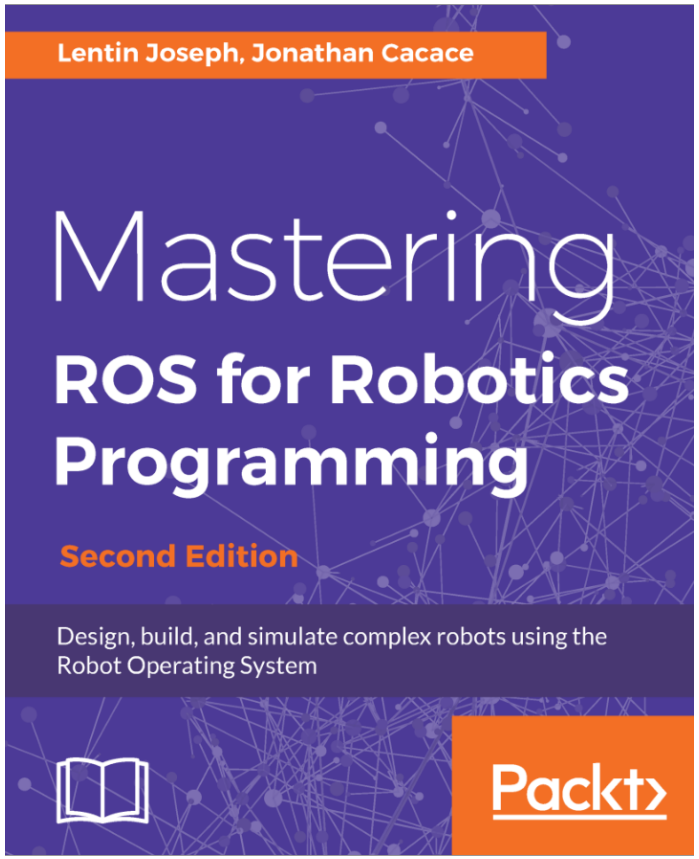


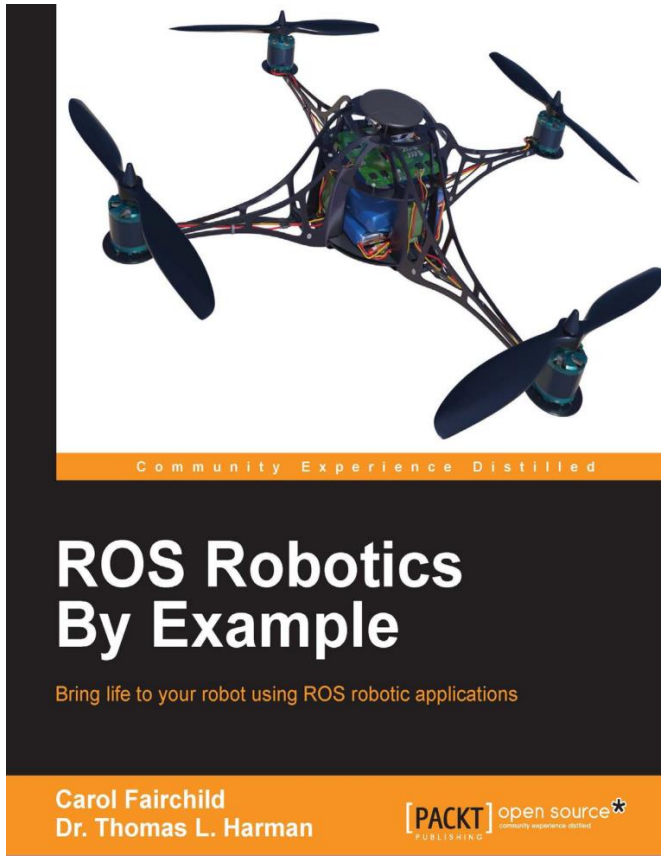

sourcing the bashrc file

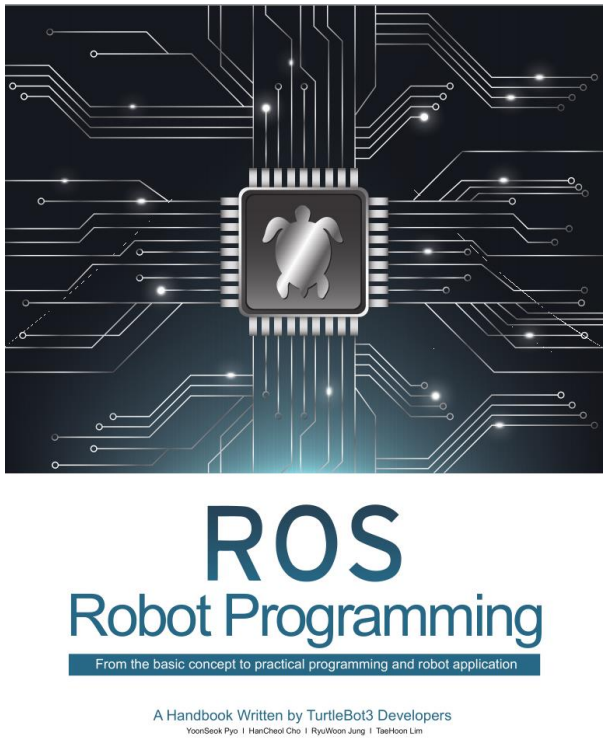
# Further References

- 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

**Université 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

UNIVERSITÉ SAVOIE MONT BLANC
POLYTECH ANNECY-CHAMBÉRY