



INFO 802
Master Advanced Mechatronics

Luc Marechal

2021

Prerequisites

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



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



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



http://ead-polytech.univ-savoie.fr/pluginfile.php/44312/mod_resource/content/1/How%20to%20Install%20Ubuntu%20on%20VirtualBox.pdf



<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.

Benefits : - Standardisation of communications

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

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



STANFORD



More info - source

<http://www.willowgarage.com>

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>

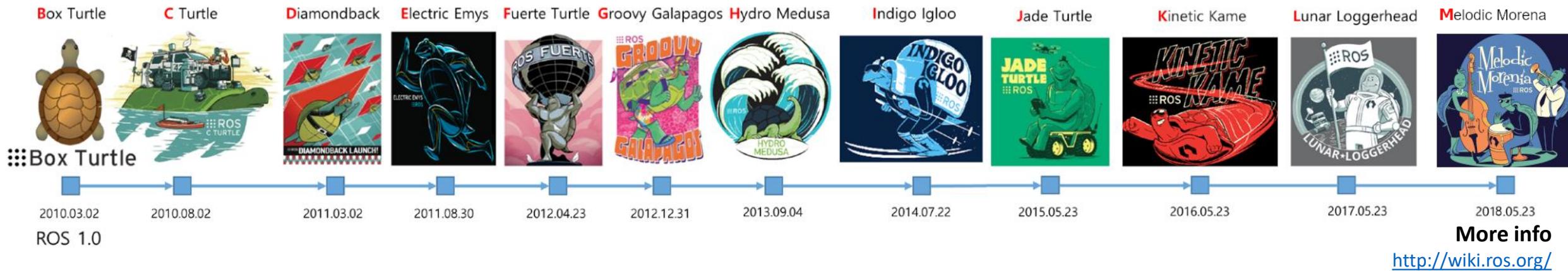
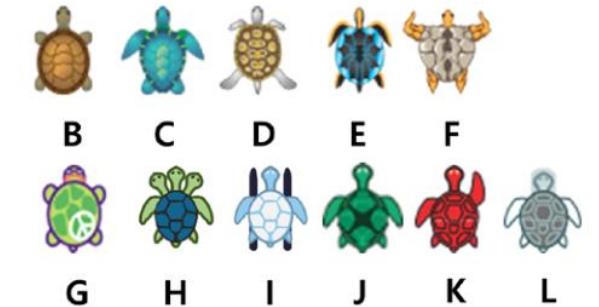
Sensors interfaced with ROS

- 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

Select Your Platform

Supported:



Ubuntu Focal amd64 armhf arm64



Debian Buster amd64 arm64

[Source installation](#)

Experimental:

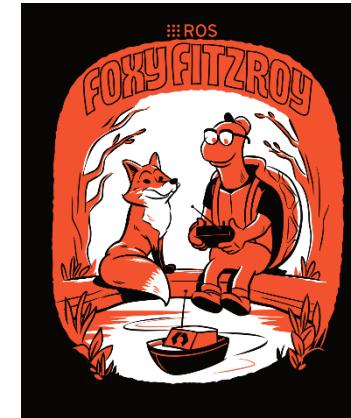


Windows 10 amd64



Arch Linux Any amd64 i686 arm armv6h armv7h aarch64

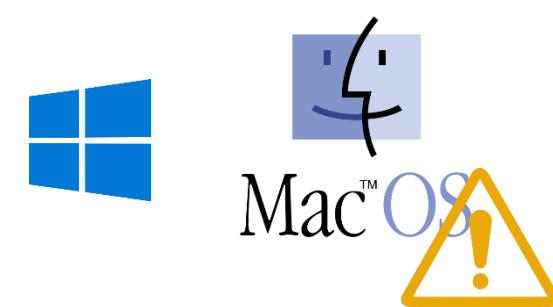
(ROS2)



Foxy Fitzroy
(ROS2)

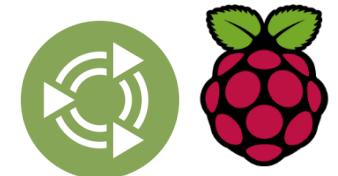
ROS Supported Platforms

- 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)



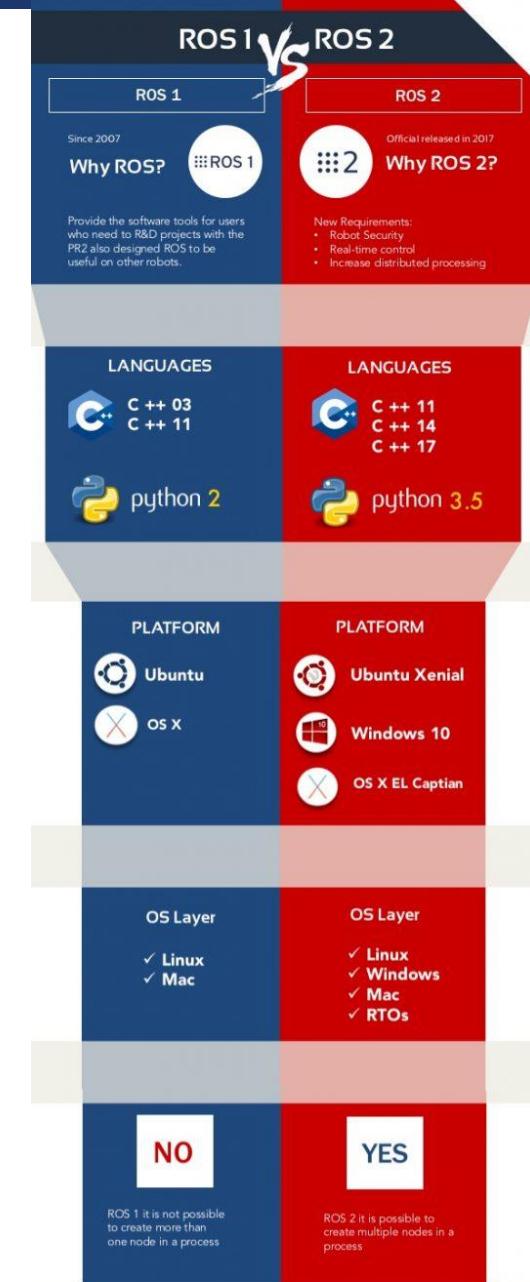
ROS Supported Platforms

- ROS distribution supported is limited to ≤ 3 latest Ubuntu versions
- ROS Kinetic Kame supports the following Ubuntu versions
 - Wily (15.10)
 - Xenial (16.04 LTS)
- ROS Jade supports the following Ubuntu versions
 - Vivid (15.04)
 - Utopic (14.04)
 - Trusty (14.04 LTS)
- Recommended Ubuntu version on Raspberry Pi 3 for ROS
 - Ubuntu Mate

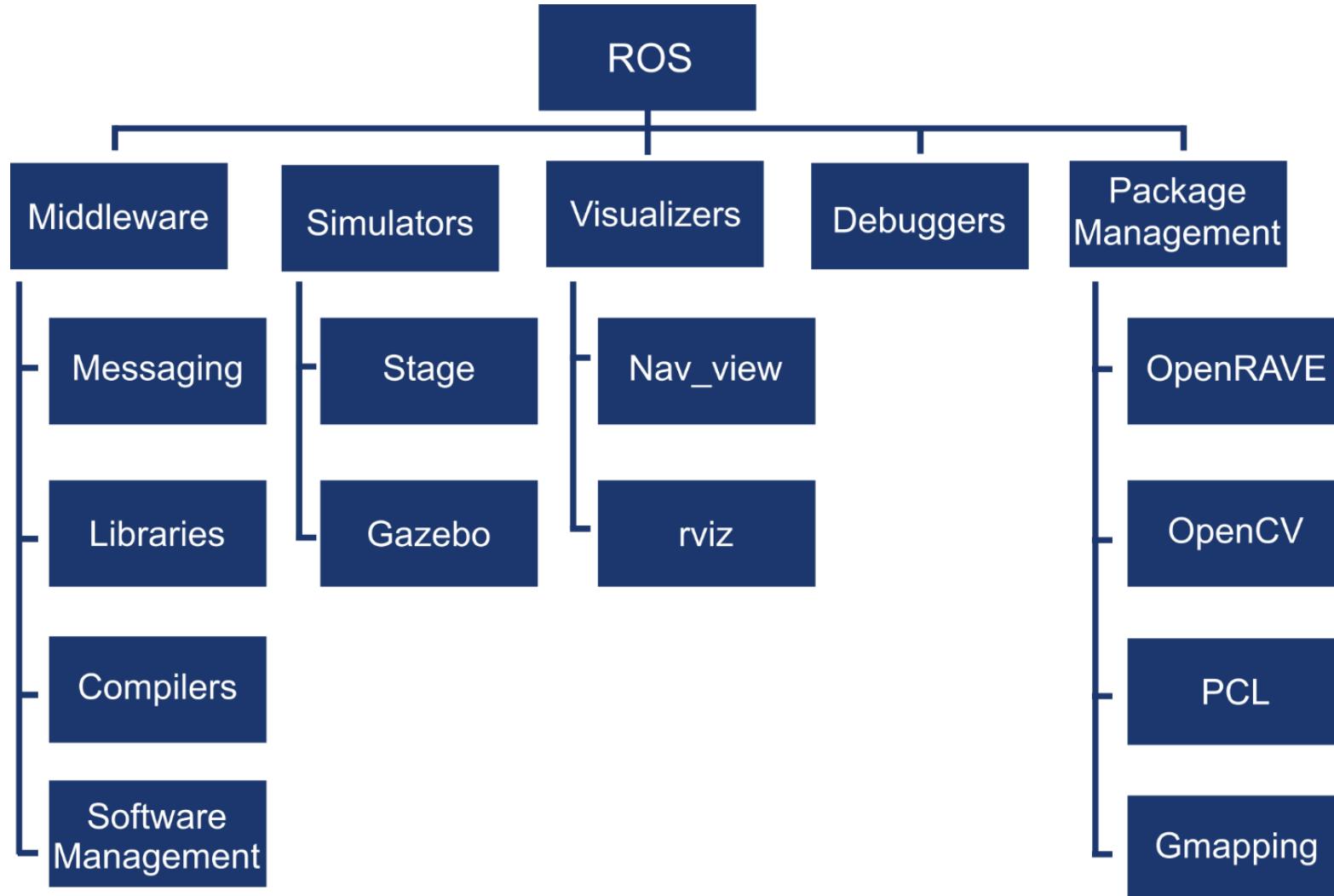


ROS 2

- 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



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!

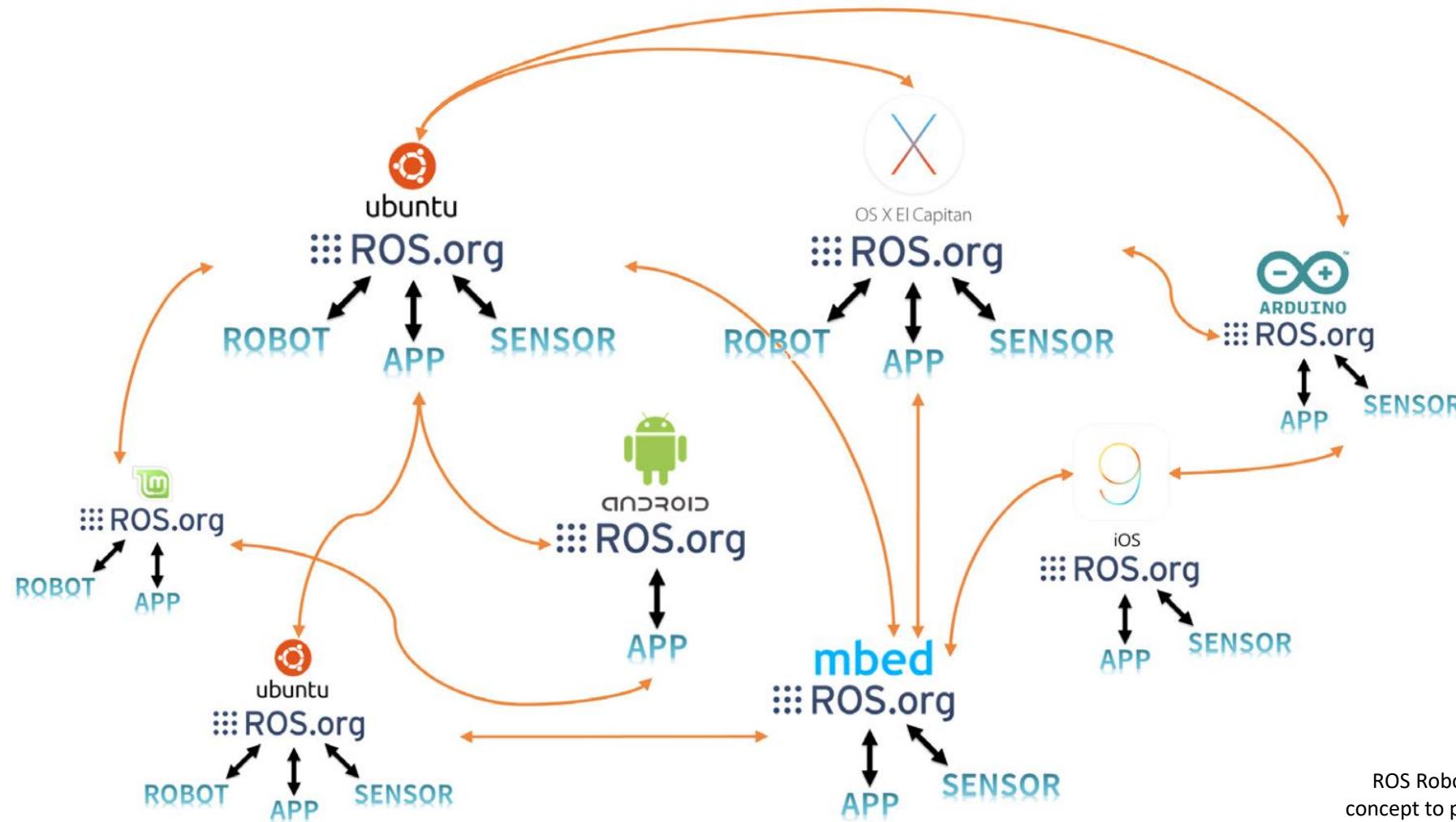
What makes the difference?

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



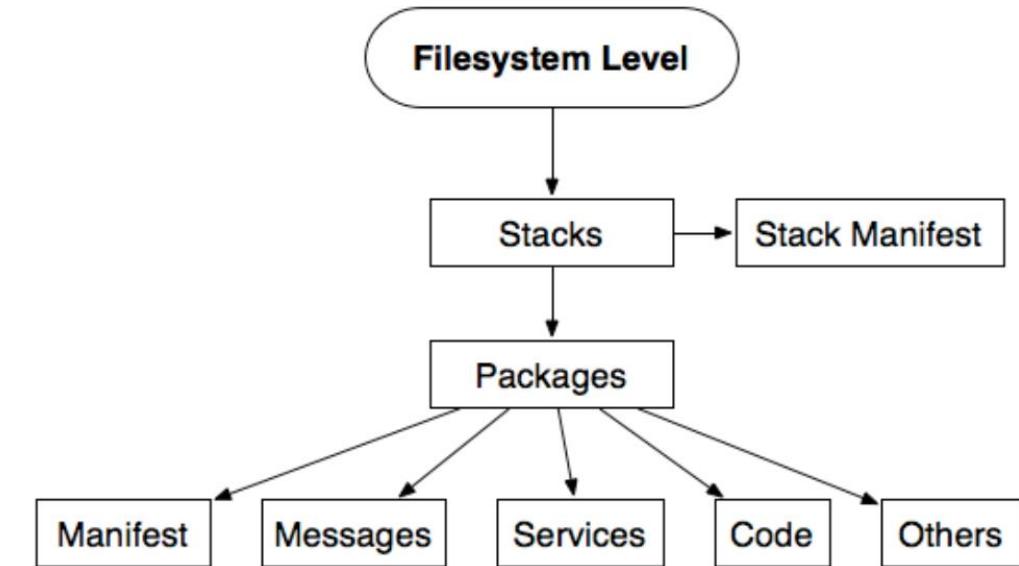
More info - source

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

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
```

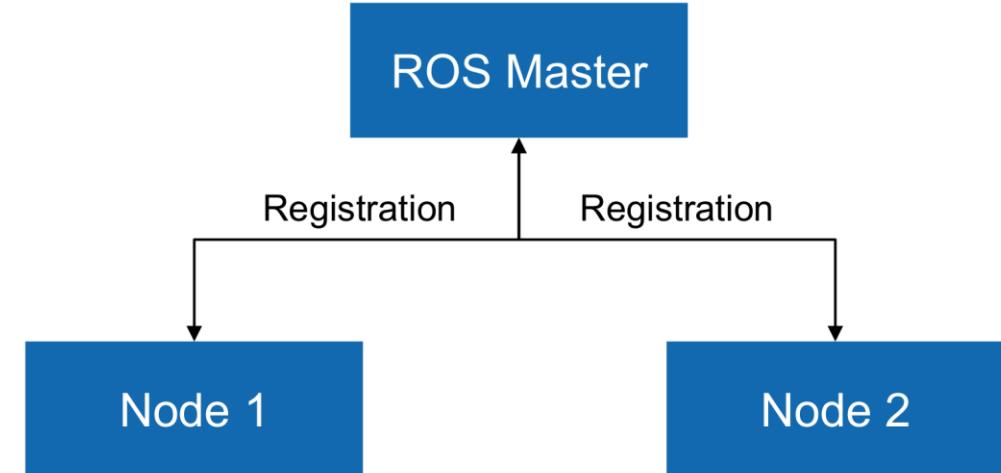
- ROS uses socket communication to facilitate networking.
The roscore starts on http://my_computer:11311
- roscore will start up:
 - a ROS Master
 - a ROS Parameter Server
 - a rosout logging node

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

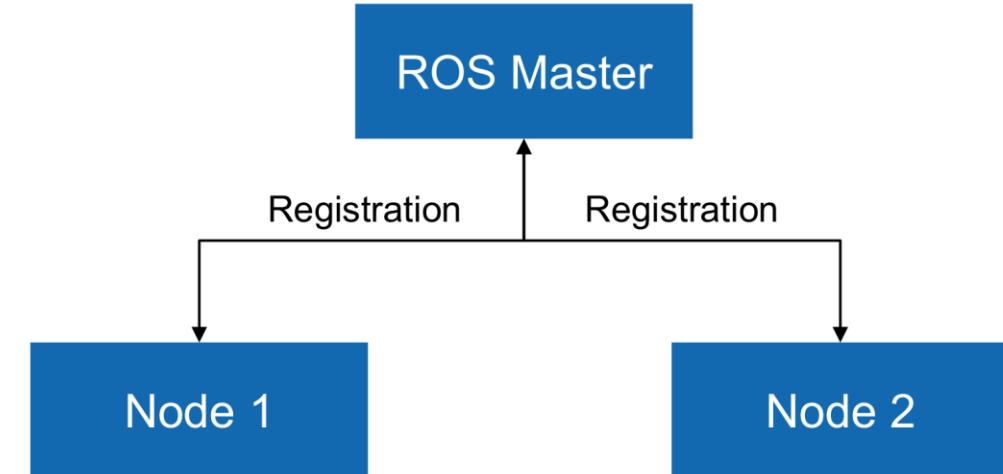
```
> rosrun [package_name] [node_name]
```

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

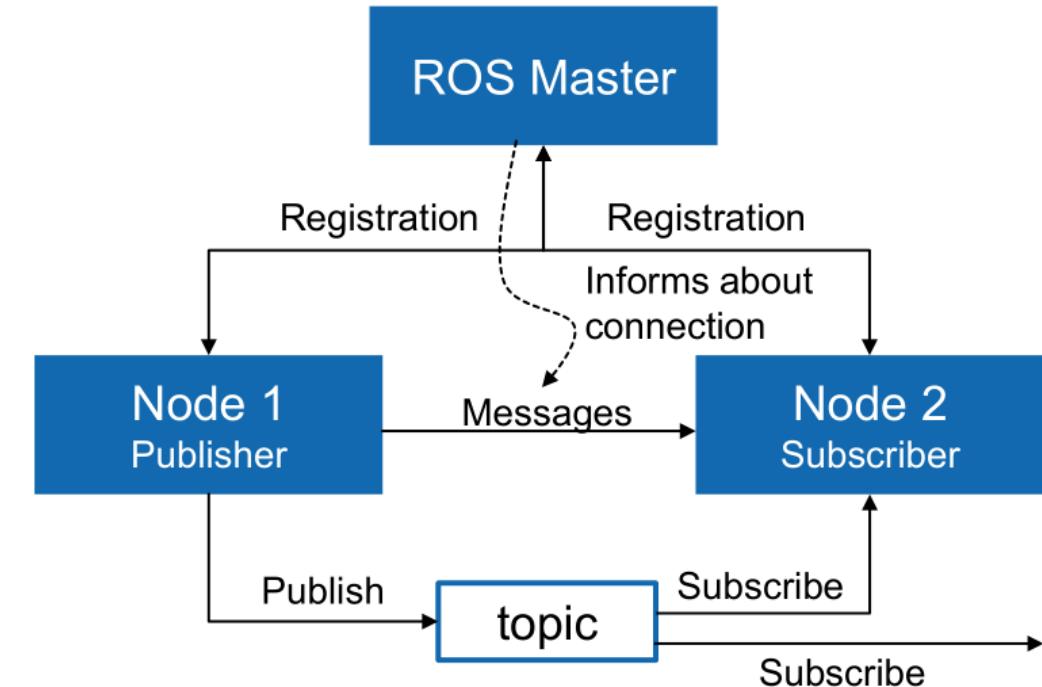
```
> rostopic list
```

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

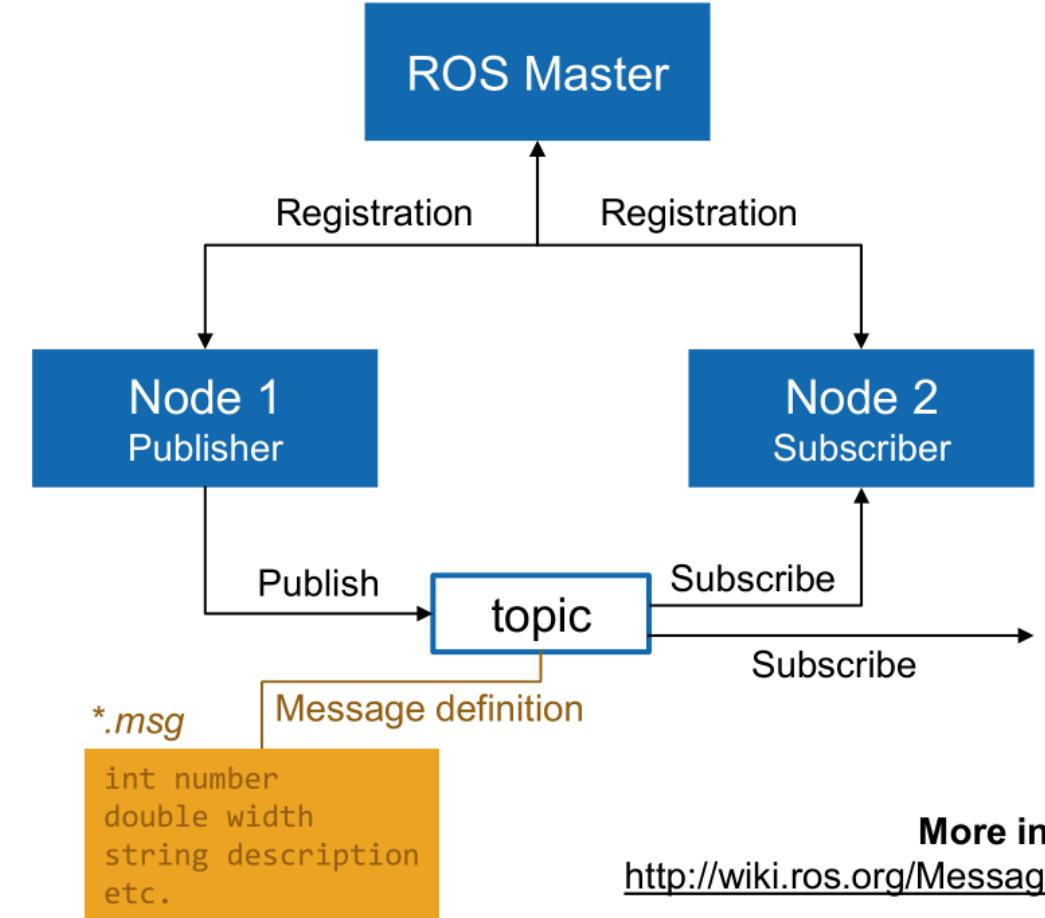
- Data structure defining the *type* of a topic
- Comprised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in **.msg* files

See the type of a topic

```
> rostopic type /topic
```

Publish a message to a topic

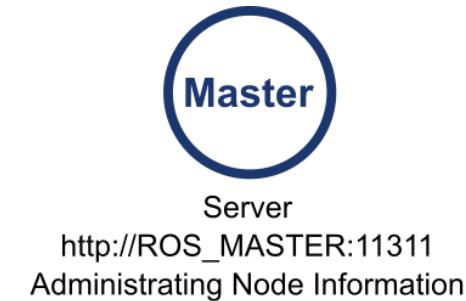
```
> rostopic pub /topic type data
```



Messages Communication Flow

Running the Master

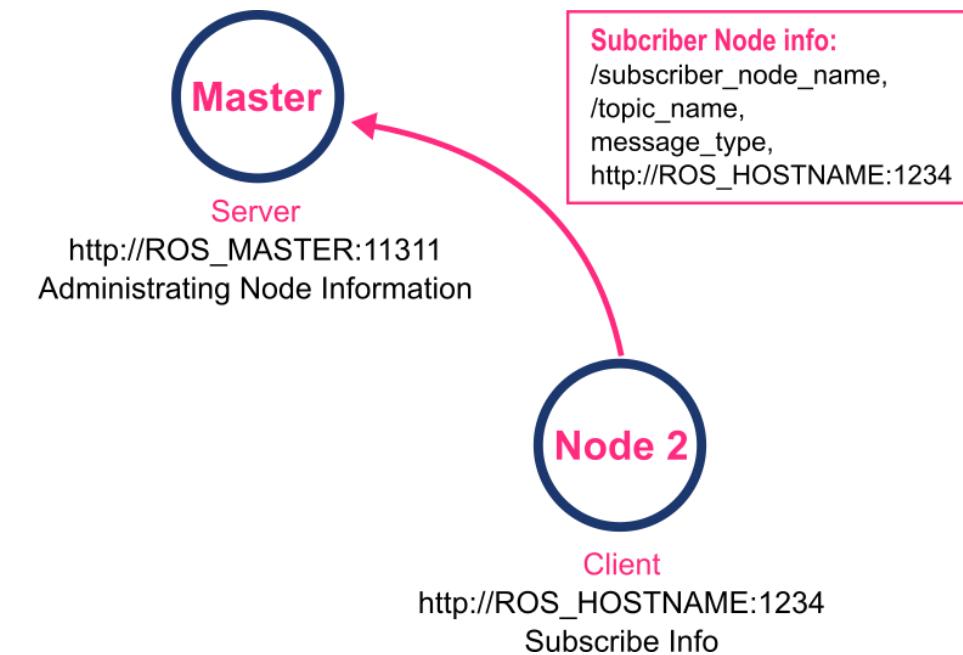
- The master registers the name of nodes, topics, services, action, message types, URI addresses



Messages Communication Flow

Running the Subscriber Node

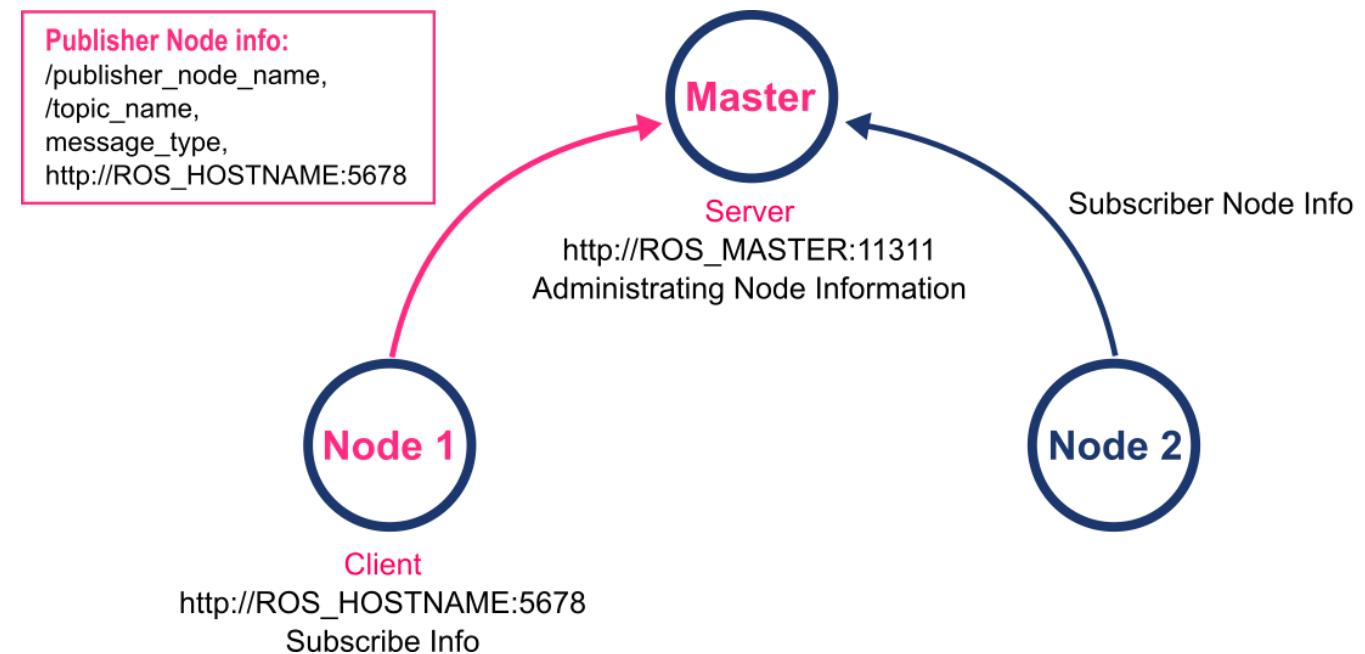
- The subscriber node registers its node name, topic name, message type, URI address, and port with the master



Messages Communication Flow

Running the Publisher Node

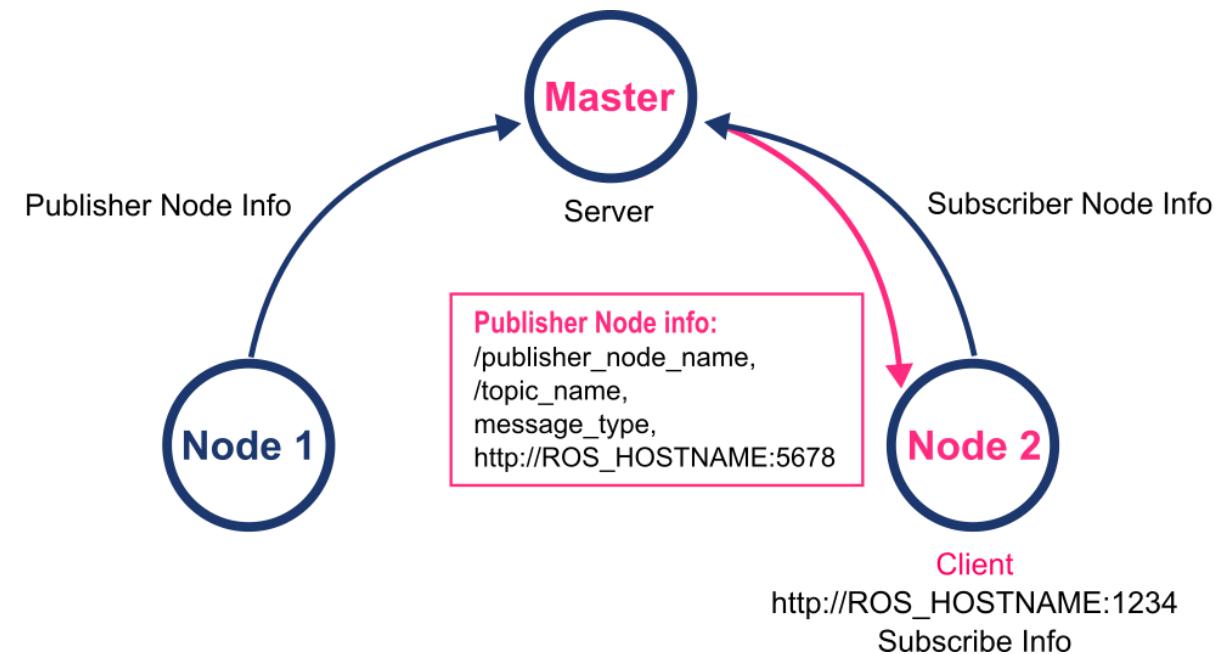
- The publisher node registers its node name, topic name, message type, URI address and port with the master.



Messages Communication Flow

Providing Publisher Information

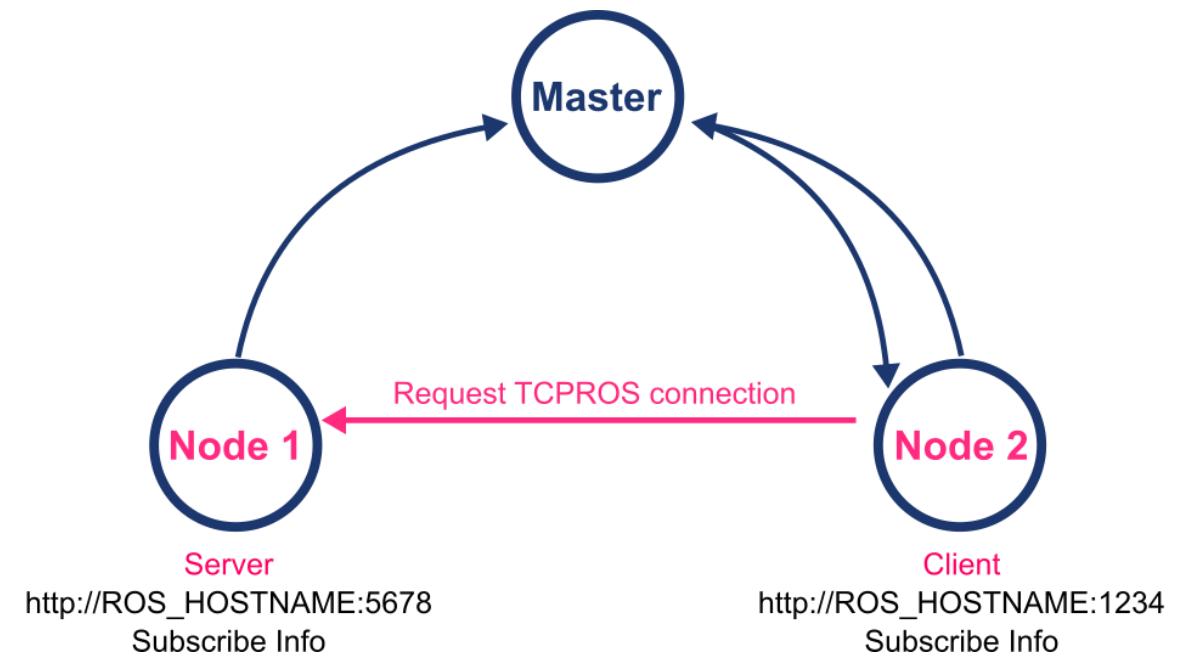
- The master distributes information such as the publisher's name, topic name, message type, URI address and port number of the publisher to subscribers that want to connect to the publisher node



Messages Communication Flow

Connection Request from the Subscriber Node

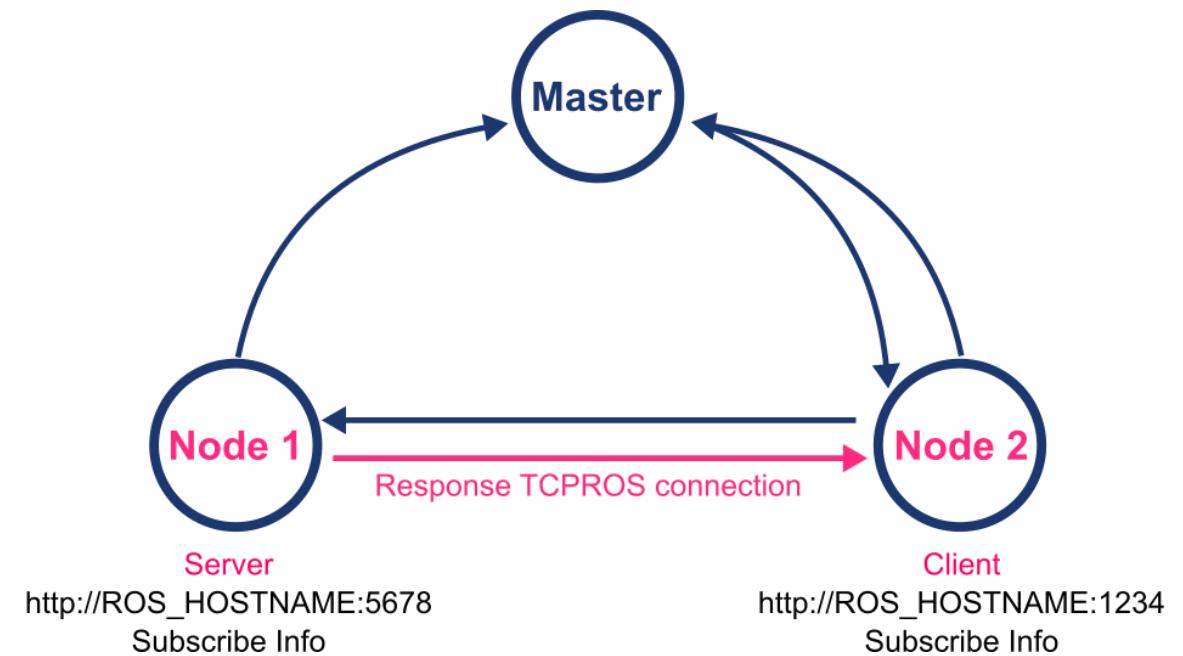
- The subscriber node requests a direct connection to the publisher node based on the publisher information received from the master.



Messages Communication Flow

Connection Response from the Publisher Node

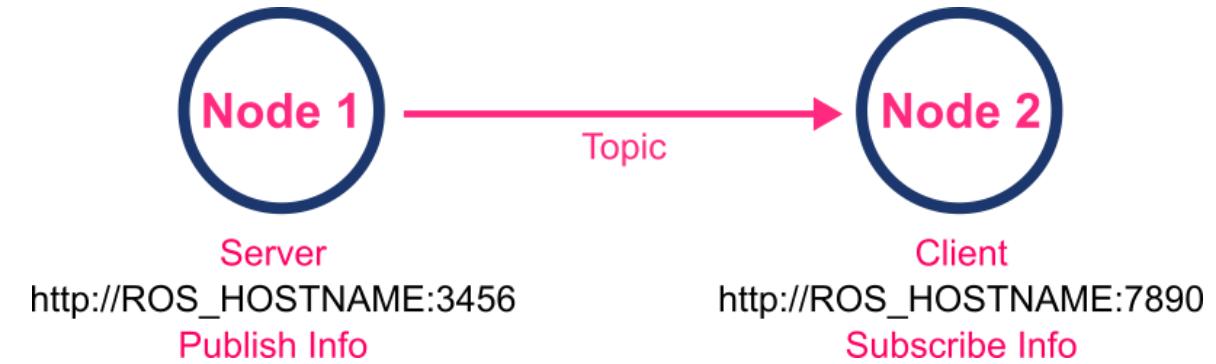
- The publisher node sends the URI address and port number of its TCP server in response to the connection request from the subscriber node.



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

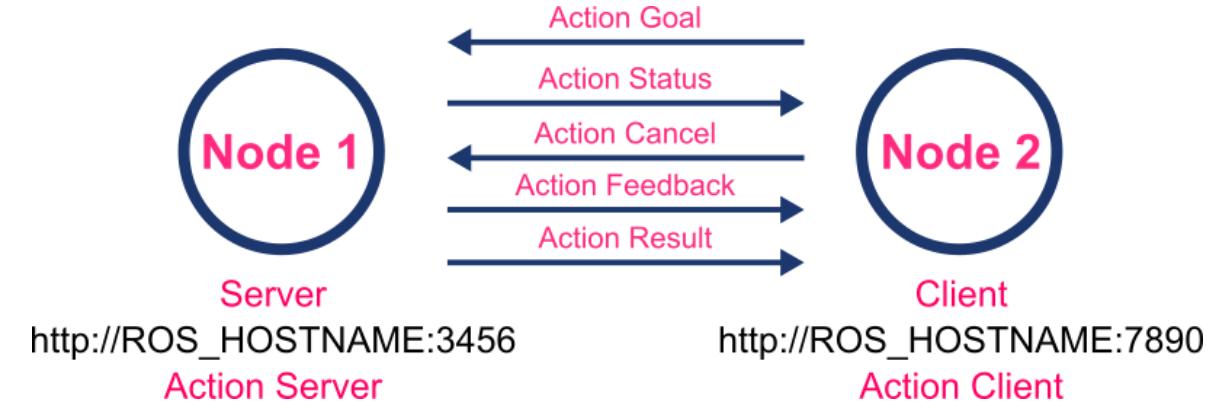
- Synchronous
- Bi-directional
- Used when request processing requests and responds current states
- Unlike the topic, the service is one-time message communication. Therefore, when the request and response of the service are completed, the connection between two nodes will be disconnected.



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.



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

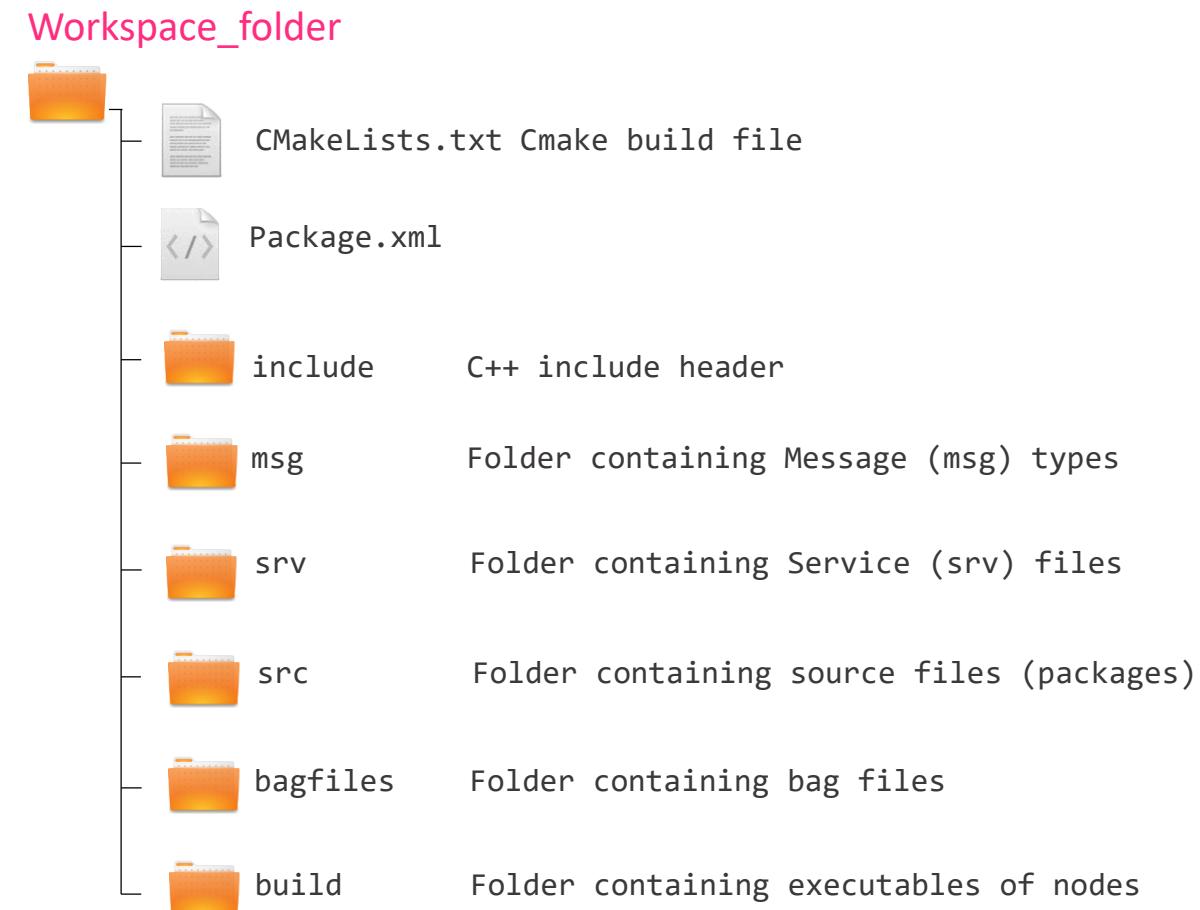
```
> source /opt/ros/kinetic/setup.bash
```

Overlay your catkin workspace with

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

Check your workspace with

```
> echo $ROS_PACKAGE_PATH
```



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

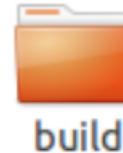
The catkin workspace contains the following spaces

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>

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**



```
PS C:\> $PSVersionTable
Name          Value
----          -----
PSVersion     5.1.15063.786
PSEdition    Desktop
PSCompatibleVersions {1.0, 2.0, 3.0, 4.0...}
BuildVersion  10.0.15063.786
CLRVersion   4.0.30319.42000
WSManStackVersion 3.0
PSRemotingProtocolVersion 2.3
SerializationVersion 1.1.0.1
```

A screenshot of a Windows PowerShell window. The title bar says "Windows PowerShell". The command "\$PSVersionTable" is run, displaying a table of the current PowerShell version and its compatible versions.

an example of CLI shell for Windows is PowerShell

```
mark@linux-desktop: /tmp/tutorial
File Edit View Search Terminal Help
Setting up tree (1.7.0-5) ...
Processing triggers for man-db (2.8.3-2) ...
mark@linux-desktop:/tmp/tutorial$ tree
.
├── another
├── combined.txt
├── dir1
└── dir2
    ├── dir3
    │   ├── test_1.txt
    │   ├── test_2.txt
    │   └── test_3.txt
    └── dir4
        └── dir5
            └── dir6
        └── folder
    └── output.txt

8 directories, 5 files
mark@linux-desktop:/tmp/tutorial$
```

A screenshot of a Linux terminal window titled "mark@linux-desktop: /tmp/tutorial". The window shows the output of the "tree" command, which lists a directory structure with files and sub-directories. The terminal has a standard Xfce-style window title bar.

.bashrc

- `.bashrc` is a script file hidden in 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

A screenshot of a terminal window titled "ros@masterpc: ~". The window shows the command "nano 4.8 /home/ros/.bashrc" and its output. The file contains several lines of shell code:

```
GNU nano 4.8 /home/ros/.bashrc
source /opt/ros/noetic/setup.bash
#source ~/catkin_ws/devel/setup.bash

export ROS_MASTER_URI=http://192.168.1.77:11311
export ROS_HOSTNAME=192.168.1.77
export TURTLEBOT3_MODEL=burger

export TURTLEBOT3BURGER1=192.168.0.229
export TURTLEBOT3BURGER2=192.168.0.230
export TURTLEBOT3WAFFLE=192.168.0.231
```

The terminal window includes standard Linux-style keyboard shortcuts at the bottom.

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 current shell
- Any changes in /home/<username>/.bashrc file will only be taken into account after sourcing



Sourcing a file

```
> source file_name.sh
```

Or

```
> . file_name.sh
```

```
ros@masterpc:~$ nano ~/.bashrc
ros@masterpc:~$ source ~/.bashrc
ros@masterpc:~$
```

sourcing the bashrc file

ROS Environment

Set up the environment

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

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



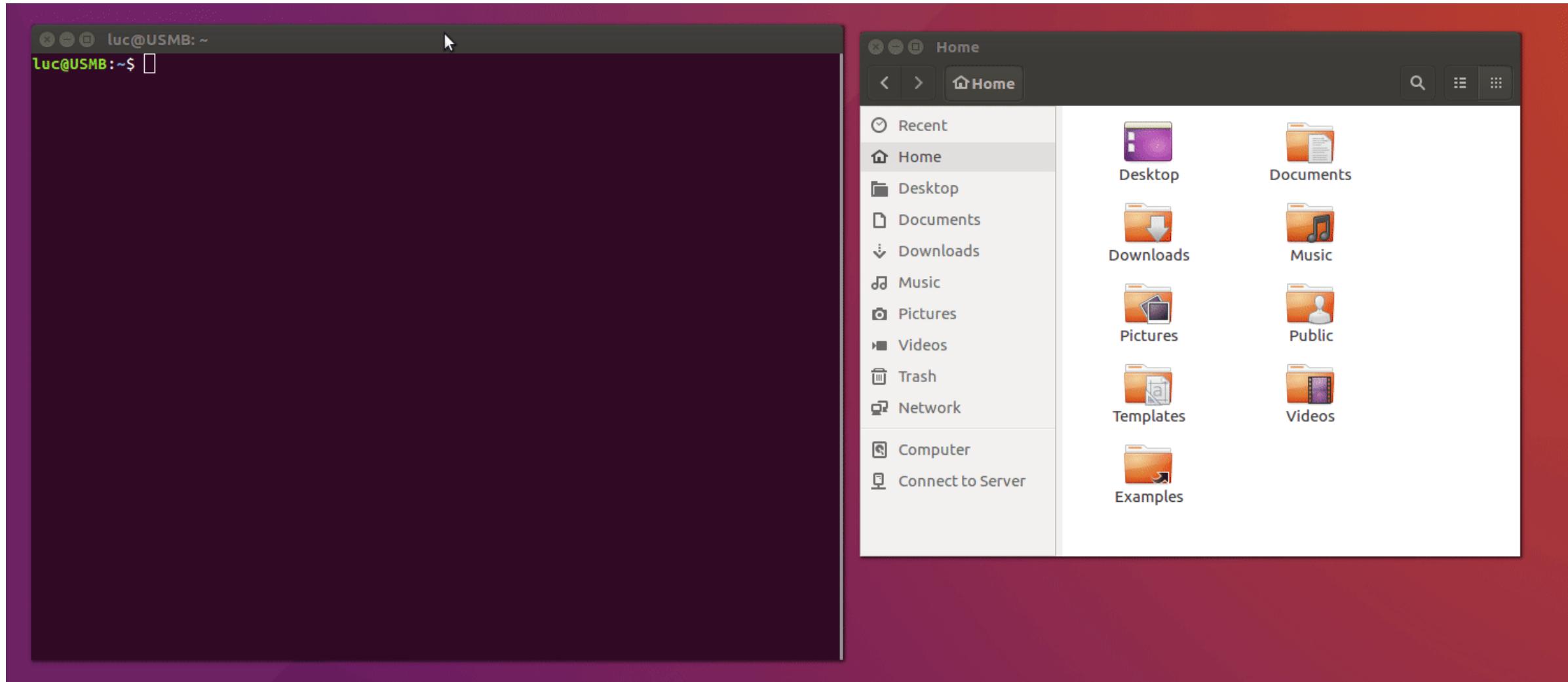
```
\s*//;s/[;&]\s*alert\$/'\\''")'  
  
# Alias definitions.  
# You may want to put all your additions into a separate file like  
# ~/.bash_aliases, instead of adding them here directly.  
# See /usr/share/doc/bash-doc/examples in the bash-doc package.  
  
if [ -f ~/.bash_aliases ]; then  
    . ~/.bash_aliases  
fi  
  
# enable programmable completion features (you don't need to enable  
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile  
# sources /etc/bash.bashrc).  
if ! shopt -oq posix; then  
    if [ -f /usr/share/bash-completion/bash_completion ]; then  
        . /usr/share/bash-completion/bash_completion  
    elif [ -f /etc/bash_completion ]; then  
        . /etc/bash_completion  
    fi  
fi  
  
#source /opt/ros/kinetic/setup.bash  
source ~/catkin_ws/devel/setup.bash
```

Creating a Catkin Workspace

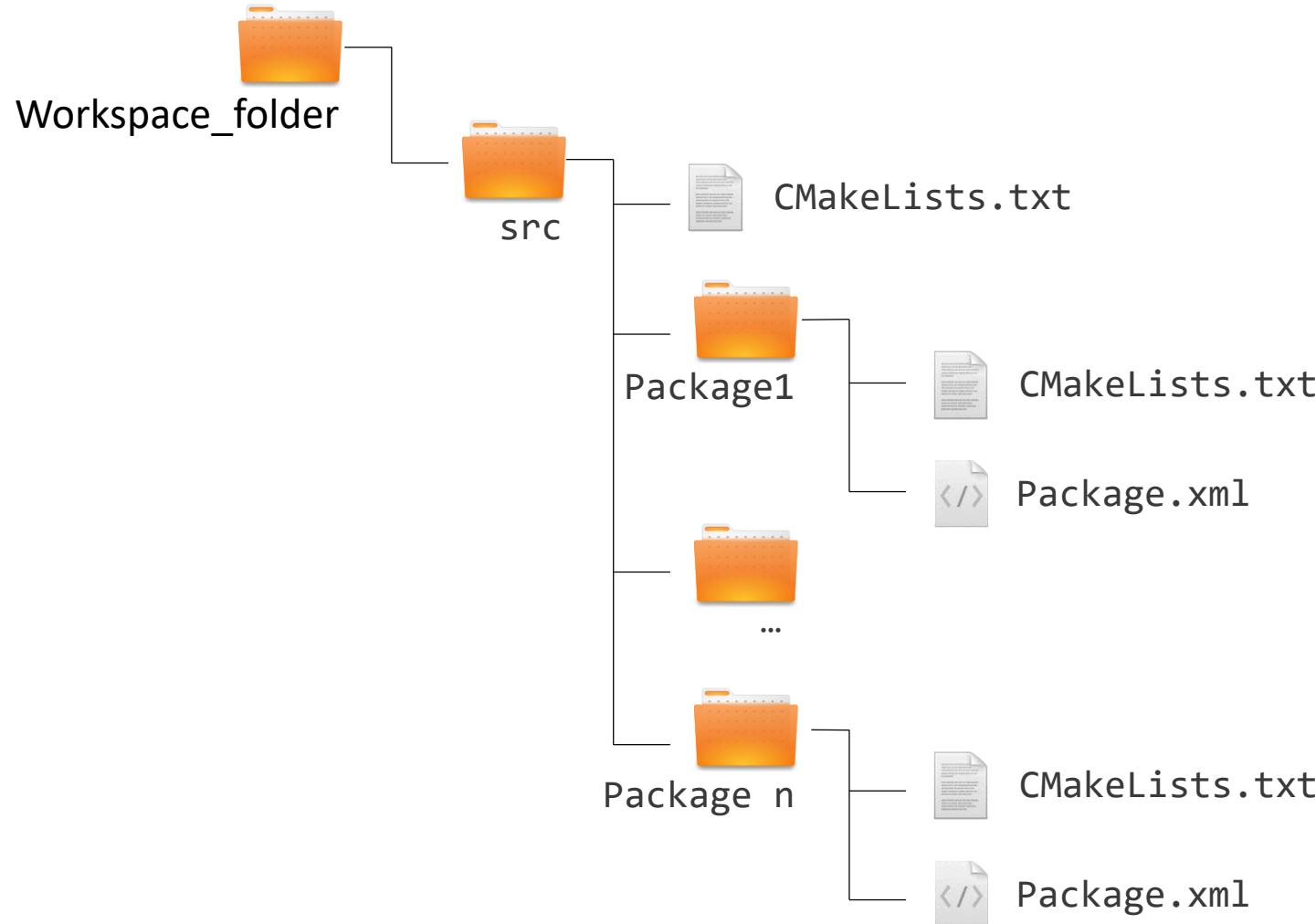
- Catkin_make

```
mkdir -p ~/catkin_ws/src
> cd ~/catkin_ws/src
> catkin_init_workspace
> cd ~/catkin_ws
> catkin_make
> source devel/setup.bash
```

Creating a Catkin Workspace: *Catkin_ws*

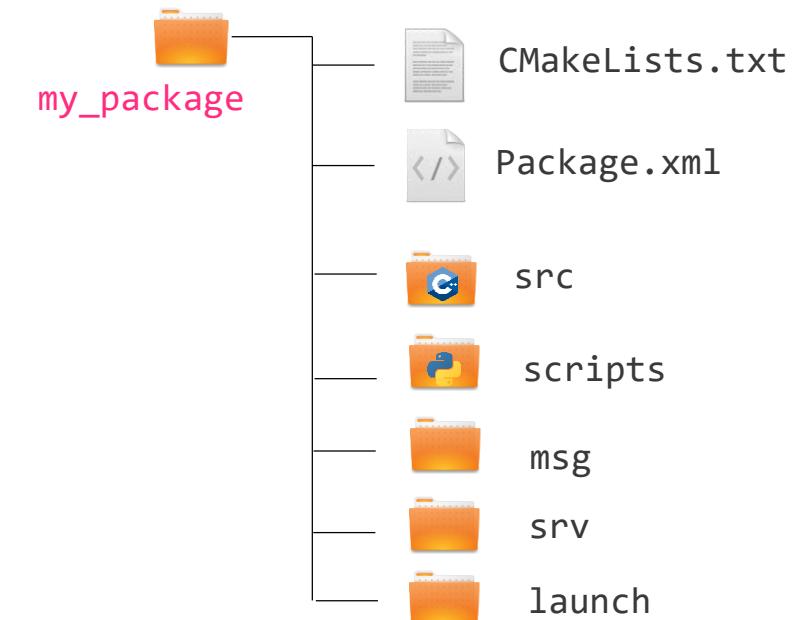
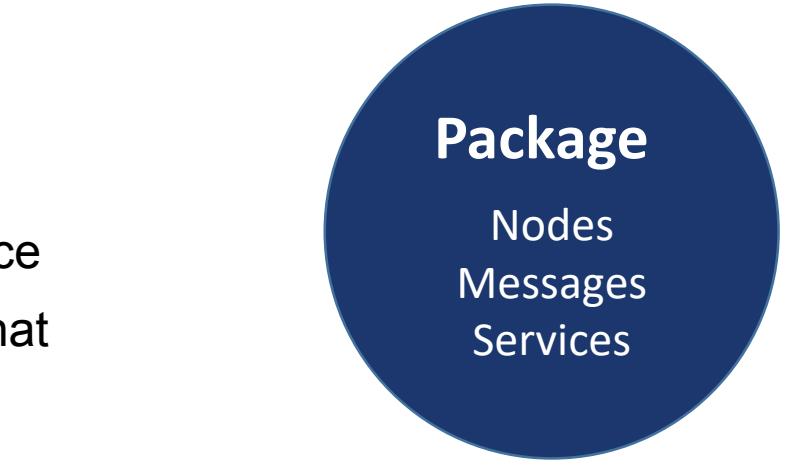


ROS Workspace Environment



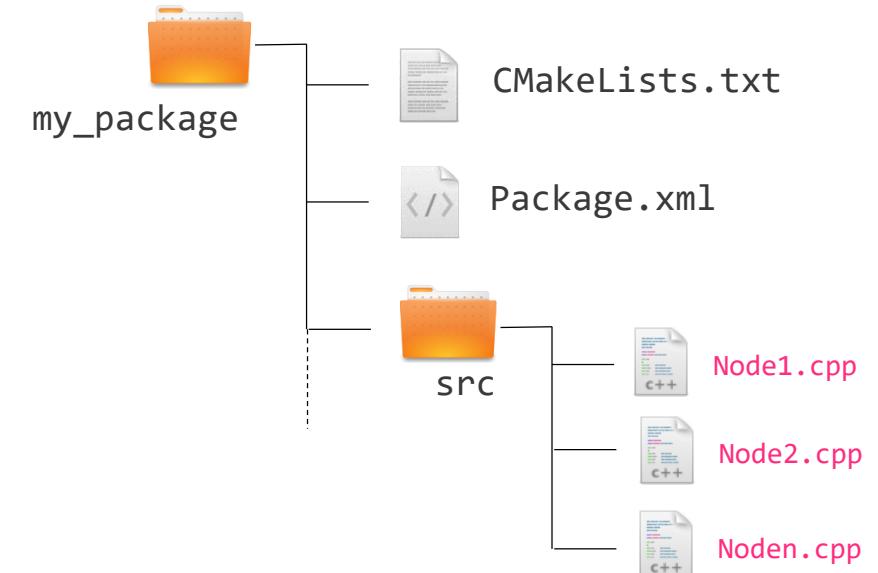
ROS Package

- 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



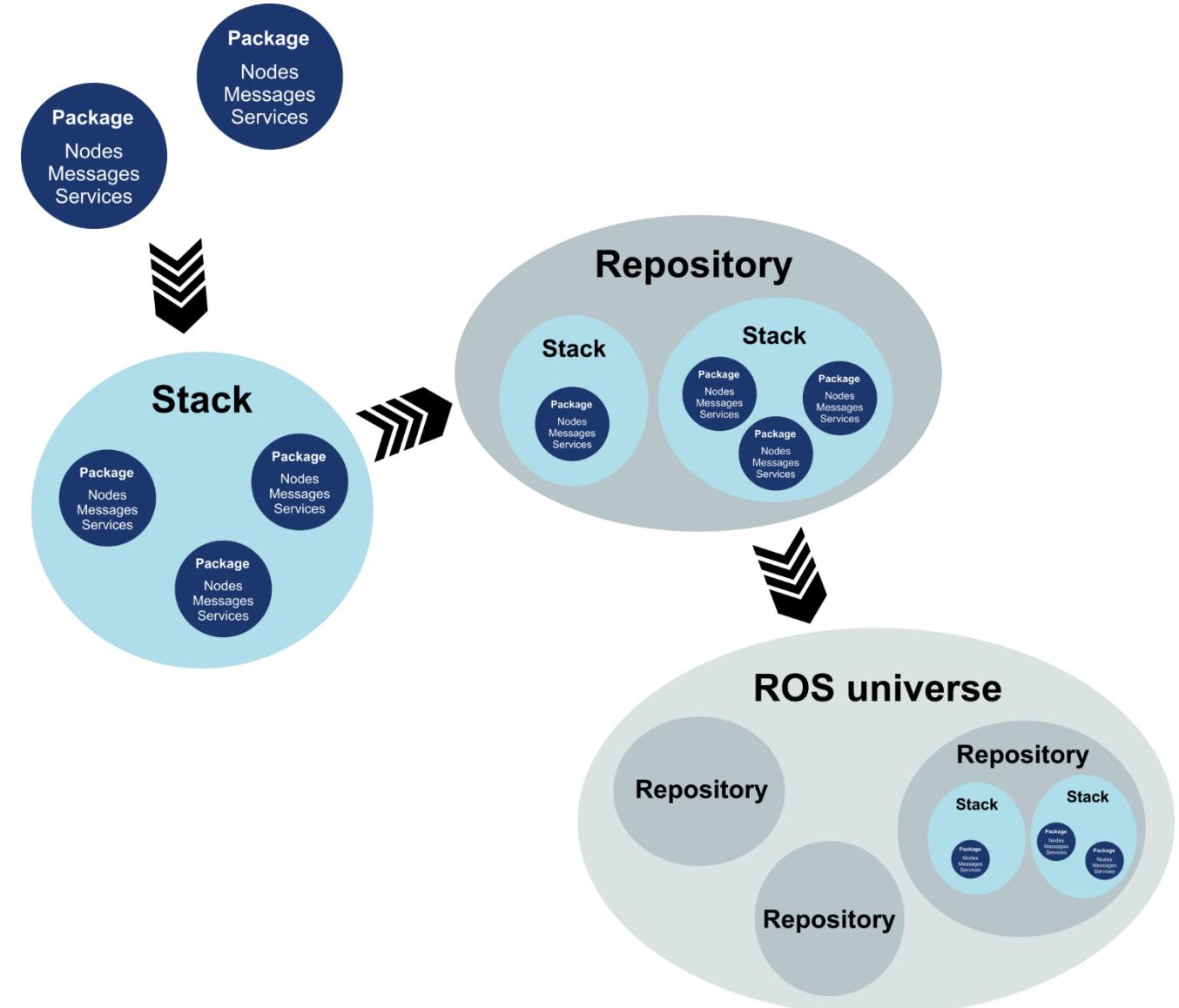
ROS Node

- File located in the source folder *src* of the package
- The file contains the code of the executable



ROS universe

- Most of ROS packages are hosted in GitHub



ROS System File

Commands

Get information on packages

```
> rospack find [package_name]
```

Change directory (cd) directly to a package or a stack

```
> roscd [Location_name[/subdir]]
```

/s directly in a package by name rather than by absolute path

```
> rosls [Location_name[/subdir]]
```

More info

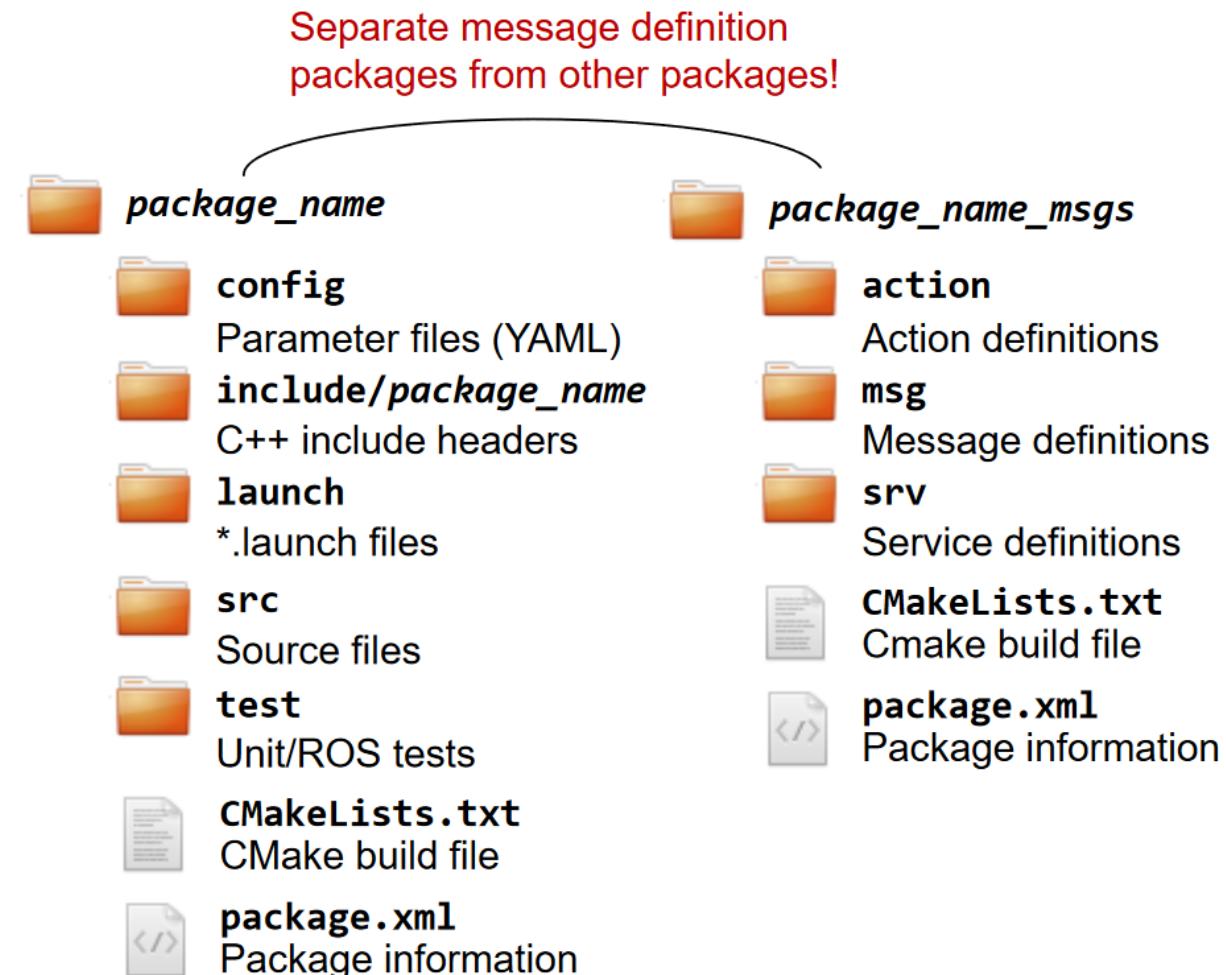
<http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem>

ROS Package

- 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

```
> cd ~/catkin_ws/src  
> catkin_create_pkg package_name {dependencies}
```



More info

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

ROS Package

package.xml

- The package.xml defines properties of the package
 - <name> Package name
 - <version> Version numbers
 - <description> Description of the content
 - <maintainer> Person maintaining the package
 - <licence> Type of licence (usually BSD)
 - Dependencies on other catkin packages**
The dependencies are split into :
build_depend, buildtool_depend, exec_depend, test_depend
 - and more

```
<?xml version="1.0"?>
<package format="2">
<name>ros_package_template</name>
<version>0.1.0</version>
<description>A template for ROS packages.</description>
<maintainer email="luc.marechal@univ-smb.fr">Luc</maintainer>
<license>BSD</license>

<url type="website">https://github.com/my_project/ros_...</url>
<author email="luc.marechal@univ-smb.fr">Luc Mare</author>

<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>

<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>

<run_depend>roscpp</run_depend>
<run_depend>rospy</run_depend>
<run_depend>std_msgs</run_depend>
</package>
```

ROS Package

CMakeLists.txt

The `CMakeLists.txt` is the input to the CMakebuild system

1. Required CMake Version (`cmake_minimum_required`)
2. Package Name (`project()`)
3. Find other CMake/Catkin packages needed for build (`find_package()`)
4. Message/Service/Action Generators (`add_message_files()`, `add_service_files()`, `add_action_files()`)
5. Invoke message/service/action generation (`generate_messages()`)
6. Specify package build info export (`catkin_package()`)
7. Libraries/Executables to build
(`add_library()`/`add_executable()`/`target_link_libraries()`)
8. Tests to build (`catkin_add_gtest()`)
9. Install rules (`install()`)

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_package_template)

## Use C++11
add_definitions(--std=c++11)

## Find catkin macros and libraries
find_package(catkin REQUIRED
COMPONENTS
    roscpp
    sensor_msgs
)
...
```

More info

<http://wiki.ros.org/catkin/CMakeLists.txt>

ROS Package

CMakeLists.txt Example

```
cmake_minimum_required(VERSION 2.8.3)
project(husky_highlevel_controller)
add_definitions(--std=c++11)

find_package(catkin REQUIRED
    COMPONENTS roscpp sensor_msgs
)
```

```
catkin_package(
    INCLUDE_DIRS include
    # LIBRARIES
    CATKIN_DEPENDS roscpp sensor_msgs
    # DEPENDS
)
```

```
include_directories(include ${catkin_INCLUDE_DIRS})
```

```
add_executable(${PROJECT_NAME} src/${PROJECT_NAME}_node.cpp
src/HuskyHighlevelController.cpp)
```

```
target_link_libraries(${PROJECT_NAME} ${catkin_LIBRARIES})
```

Use the same name as in the package.xml

We use C++11 by default

List the packages that your package requires to build (have to be listed in package.xml)

Specify build export information

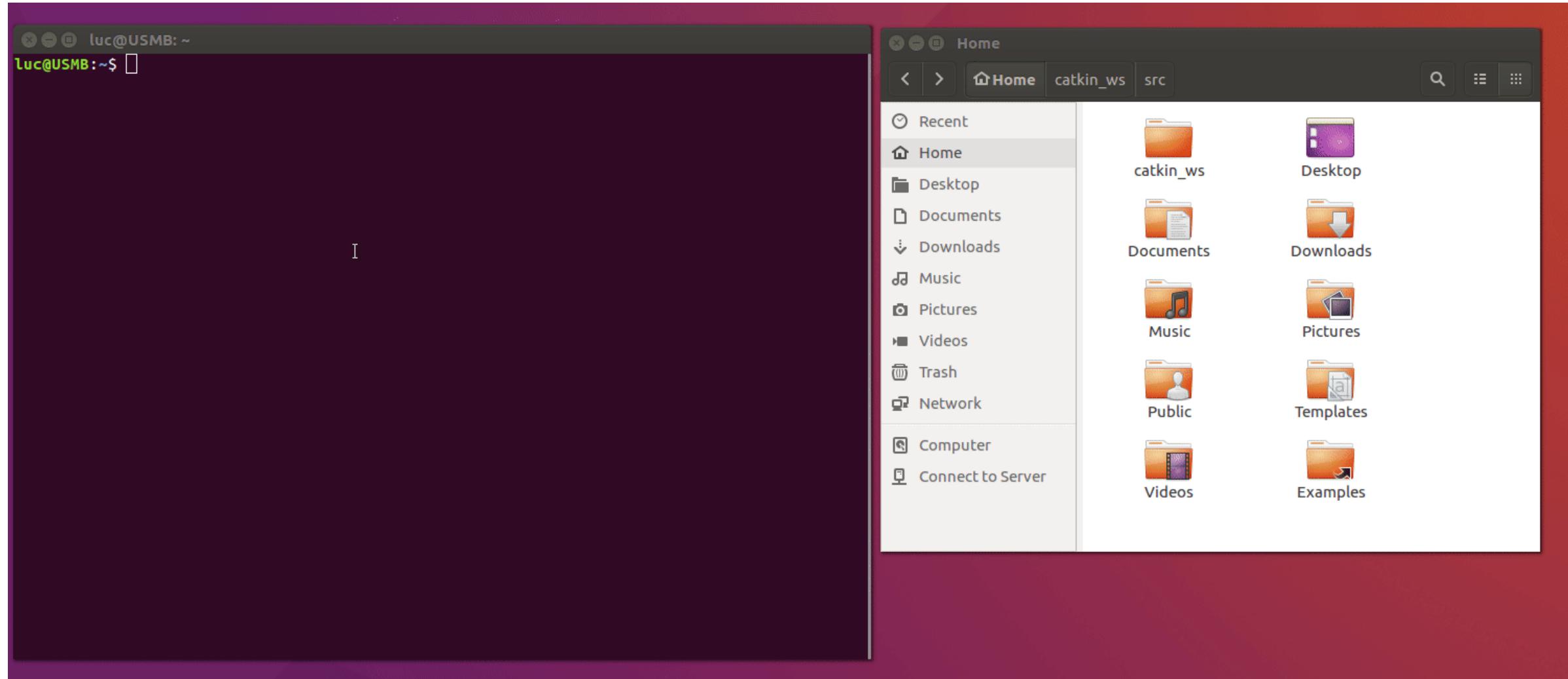
- INCLUDE_DIRS: Directories with header files
- LIBRARIES: Libraries created in this project
- CATKIN_DEPENDS: Packages dependent projects also need
- DEPENDS: System dependencies dependent projects also need (have to be listed in package.xml)

Specify locations of header files

Declare a C++ executable

Specify libraries to link the executable against

Creating a ROS package : *beginner_tutorials*



Exercice 1 - Bash

1 – Edit your bashrc file and add *your country name* in the following environment variable to your system:

MY_COUNTRY

>

2 – Source your environment

>

3 – Check that your variable exist with the command: *echo*

>

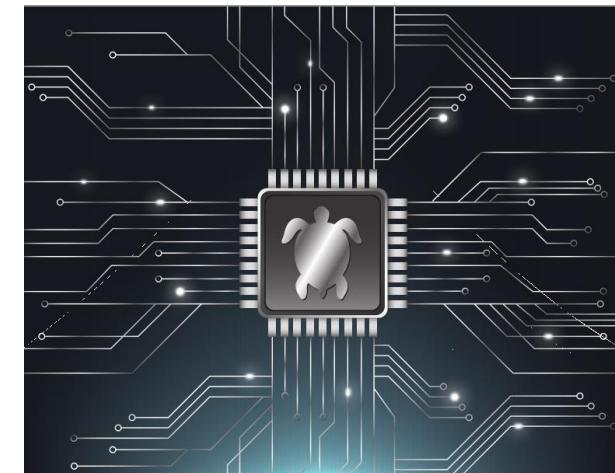
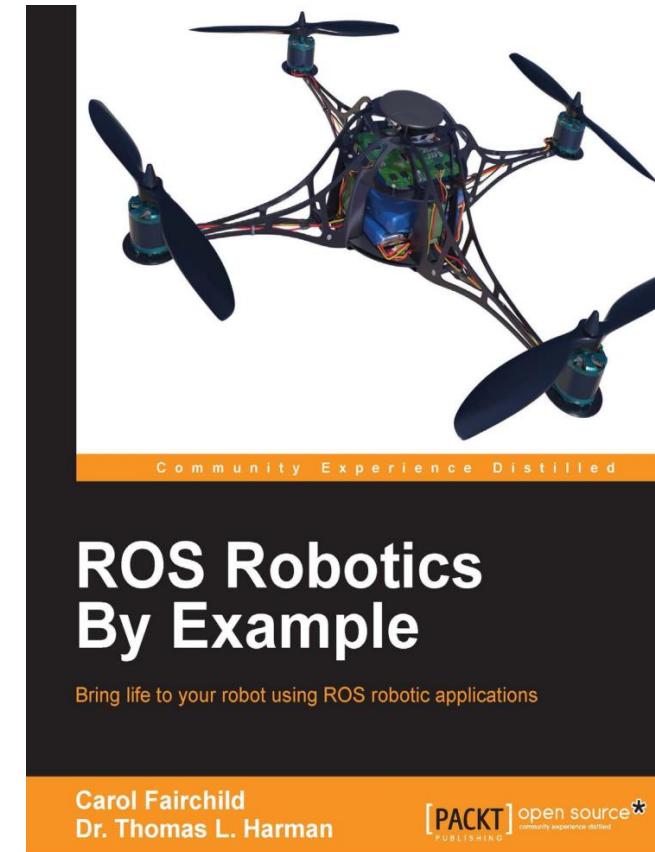
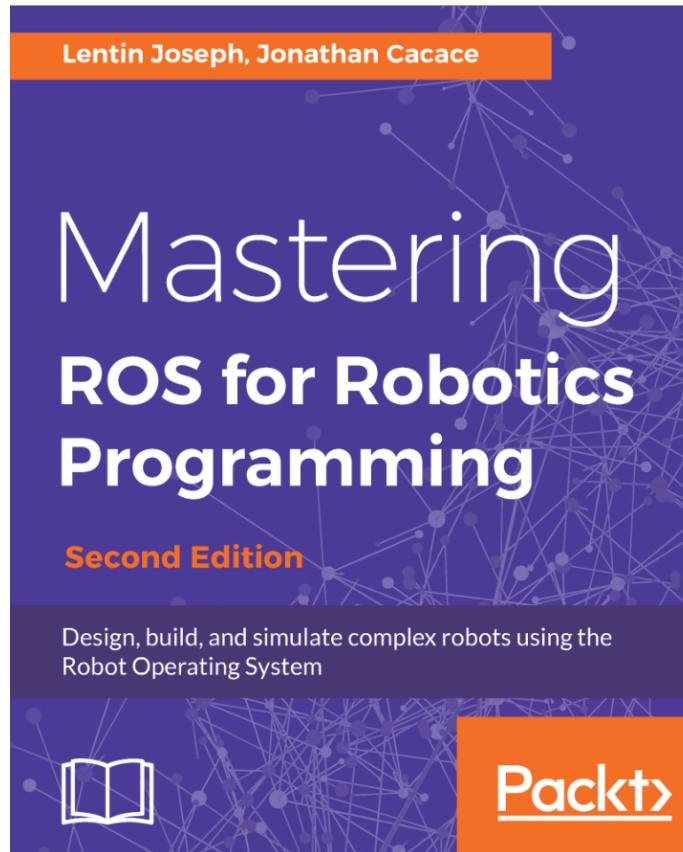
Exercice 2 – Creat Bash

1 – Create a catkin ROS Workspace named : *test_ws*

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 Chambéry
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