ROS INTRODUCTION

ROBOTICS

POLITECNICO
MILANO 1863

ROS: ROBOT OPERATING SYSTEM

goo.gl/DBwhhC

ROS main features:

- Distributed framework
- Reuse code
- Language independent
- Easy testing on Real Robot & Simulation
- Scaling

ROS Components

- File system tools
- Building tools
- Packages
- Monitoring and GUIs
- Data Logging

INSTALLATION

goo.gl/DBwhhC

WARNING

This instruction are for:

Ubuntu 16.04.2 (suggested)

and **Ubuntu 15.10** only

INSTALLATION

goo.gl/DBwhhC

Initial setup for sources and keys for downloading the packages

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release
-sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --
recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

INSTALLATION

goo.gl/DBwhhC

Update the packaged index

```
sudo apt-get update
```

Choose between the four pre-packaged ROS installation

Desktop-Full Install:	`sudo apt-get install ros-kinetic-desktop-full`
Desktop Install:	`sudo apt-get install ros-kinetic-desktop`
ROS-Base:	`sudo apt-get install ros-kinetic-ros-base`

INSTALL

goo.gl/DBwhhC

How to install single packages:

```
sudo apt-get install ros-kinetic-PACKAGE
```

Example

```
sudo apt-get install ros-kinetic-slam-gmapping
```

To find the exact name of a package you can use the usual aptitude search:

```
apt-cache search ros-kinetic
```

INITIALIZATION AND SETUP

`rosdep` enables you to easily install system dependencies and it's required by some ROS packages

```
sudo rosdep init
rosdep update
```

To use `catkin` (the compiling environment of ROS) you need to define the location of your ROS installation.

In each new terminal type:

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

Or put it inside your `.bashrc`

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

SUGGESTED TOOL

goo.gl/DBwhhC

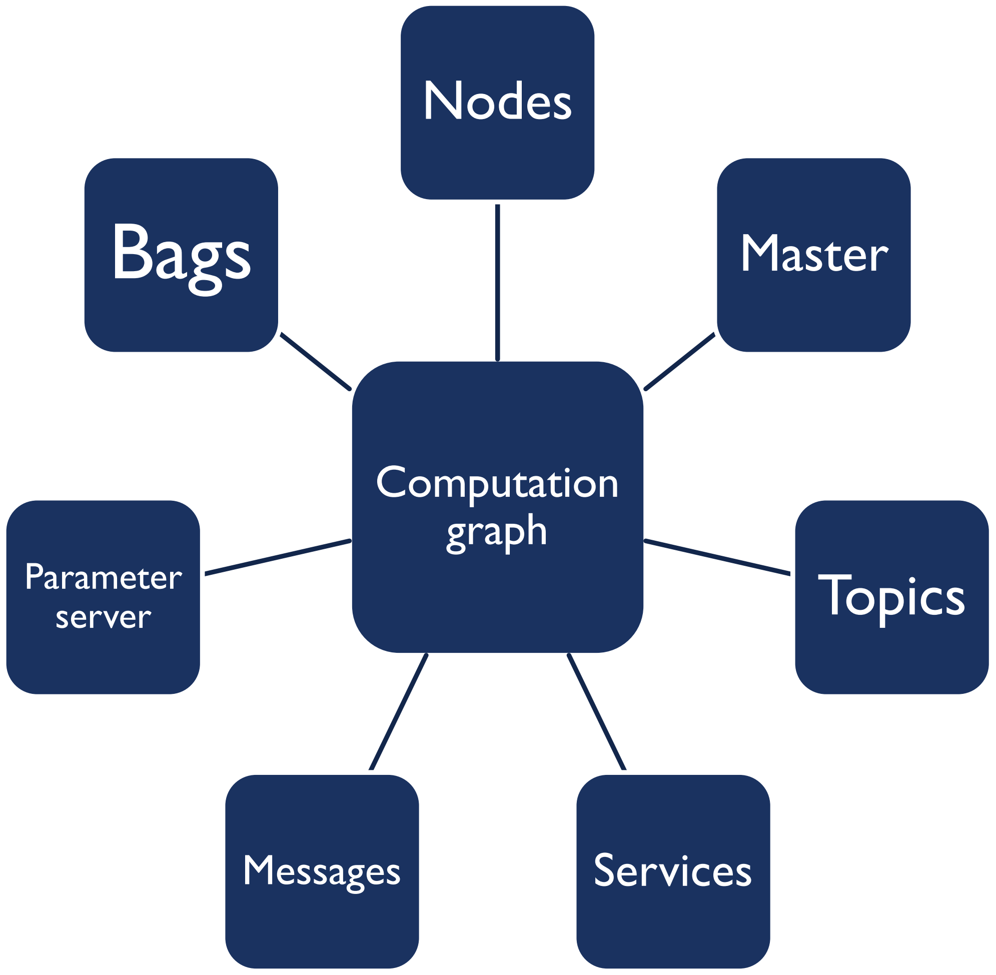
`rosinstall` is a frequently used command-line tool in ROS that is distributed separately. It enables you to easily download many source trees for ROS packages with one command.

To install this tool on Ubuntu, run:

```
sudo apt-get install python-rosinstall
```

ROS STRUCTURE: COMPUTATIONAL GRAPH

goo.gl/DBwhhC



The *Computation Graph* is the peer-to-peer network of ROS processes that are processing data together.

NODES

Executable unit of ROS:

- Scripts for Python
- Compiled source code for C++

Process that performs computation

Nodes exchange information via the graph

Meant to operate at fine-grained scale

A robot system is composed by various nodes

```
rosrun package_name node_name
```

```
rosrun turtlesim turtlesim_node
```

MASTER

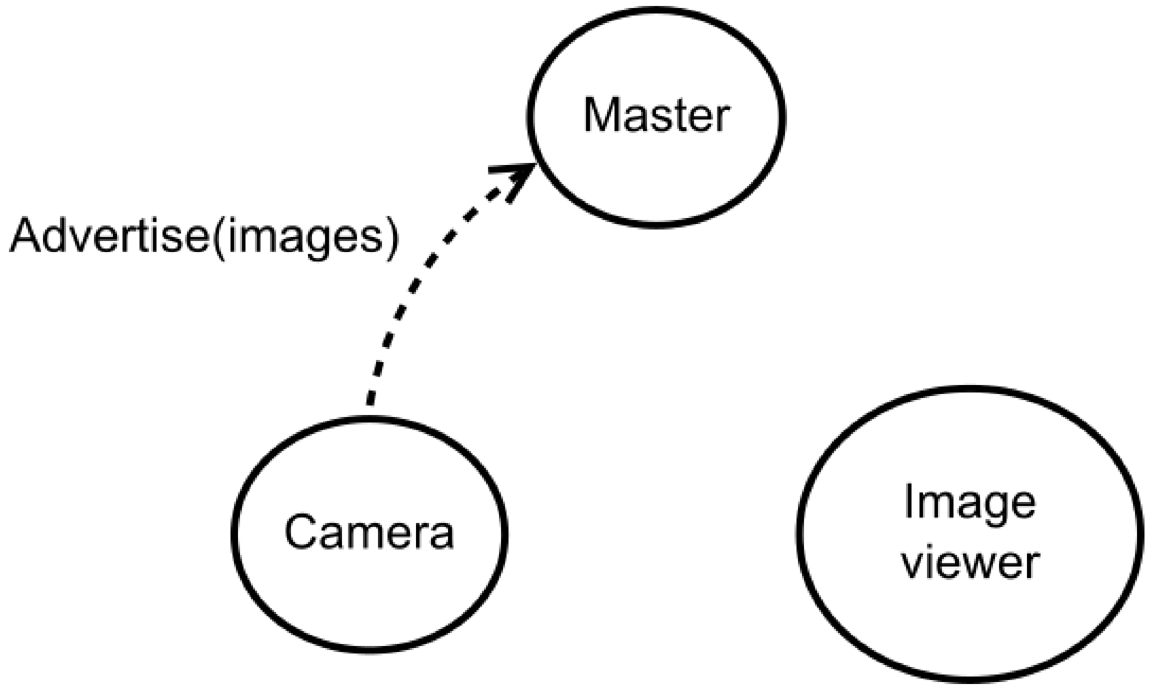
Provides naming and registration services

Essential for nodes interactions

One master for each system, even on distributed architectures

Enables individual ROS nodes to locate one another

One of the functionalities provided by `roscore`



MASTER

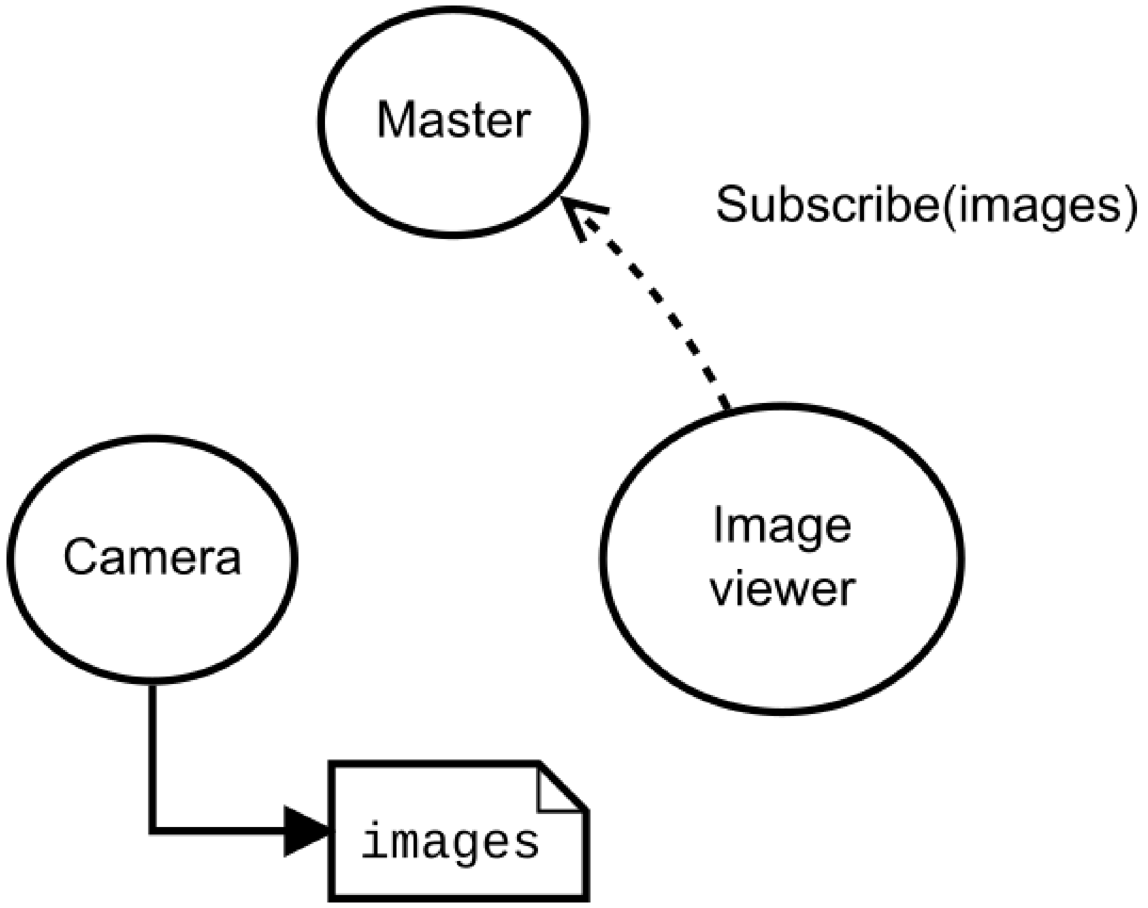
Provides naming and registration services

Essential for nodes interactions

One master for each system, even on distributed architectures

Enables individual ROS nodes to locate one another

One of the functionalities provided by `roscore`



MASTER

goo.gl/DBwhhC

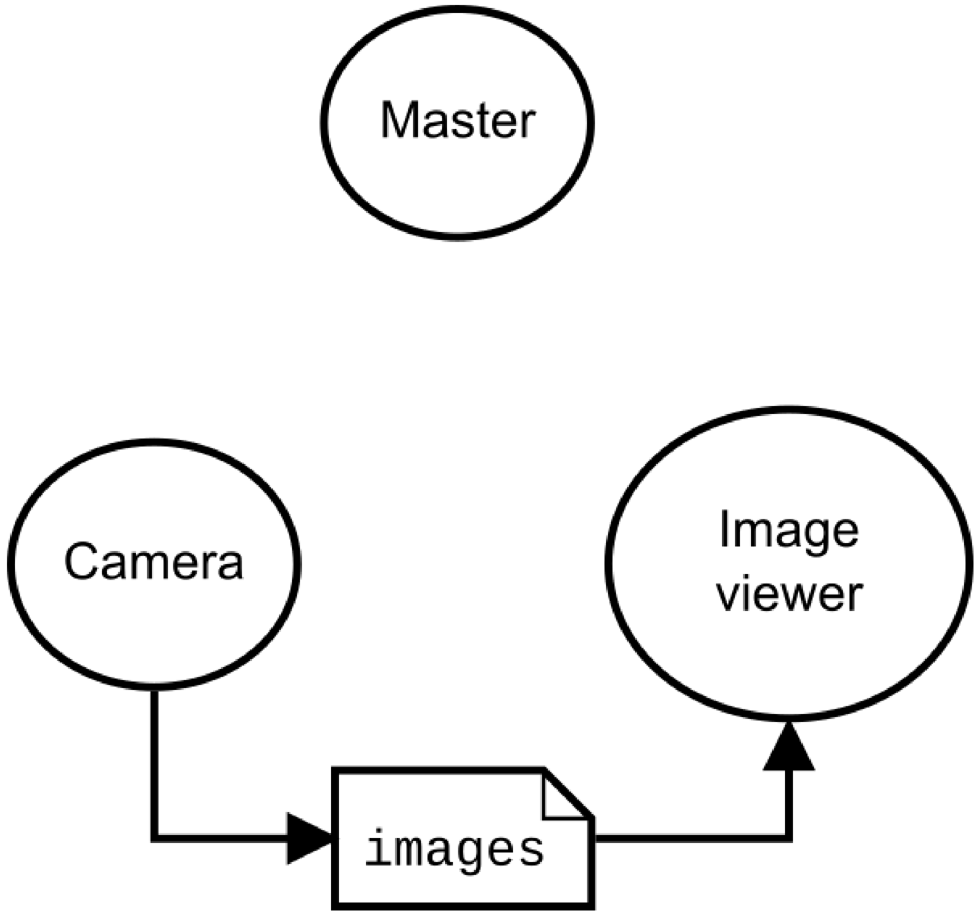
Provides naming and registration services

Essential for nodes interactions

One master for each system, even on distributed architectures

Enables individual ROS nodes to locate one another

One of the functionalities provided by `roscore`



TOPICS

goo.gl/DBwhhC

Named channels for communication

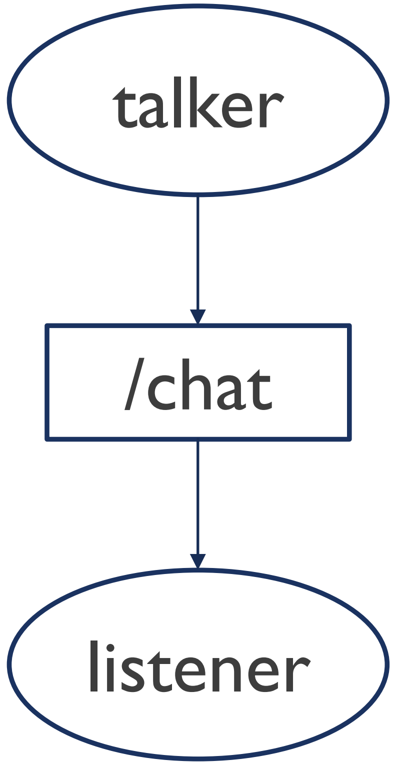
Implement the publish/subscribe paradigm

No guarantee of delivery

Have a specific message type

Multiple nodes can publish messages on a topic

Multiple nodes can read messages from a topic



TOPICS

goo.gl/DBwhhC

Named channels for communication

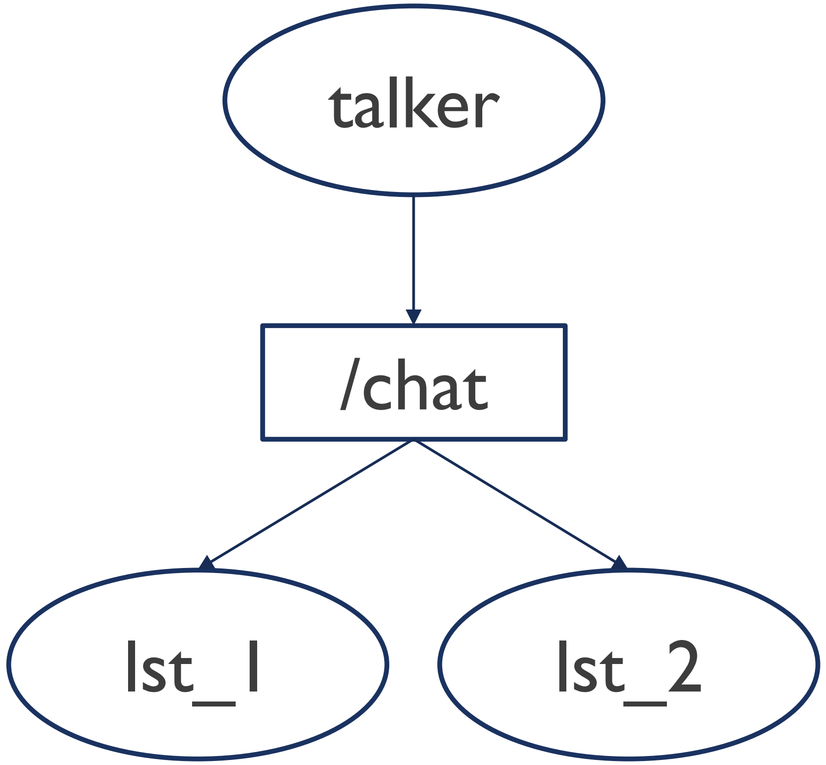
Implement the publish/subscribe paradigm

No guarantee of delivery

Have a specific message type

Multiple nodes can publish messages on a topic

Multiple nodes can read messages from a topic



TOPICS

goo.gl/DBwhhC

Named channels for communication

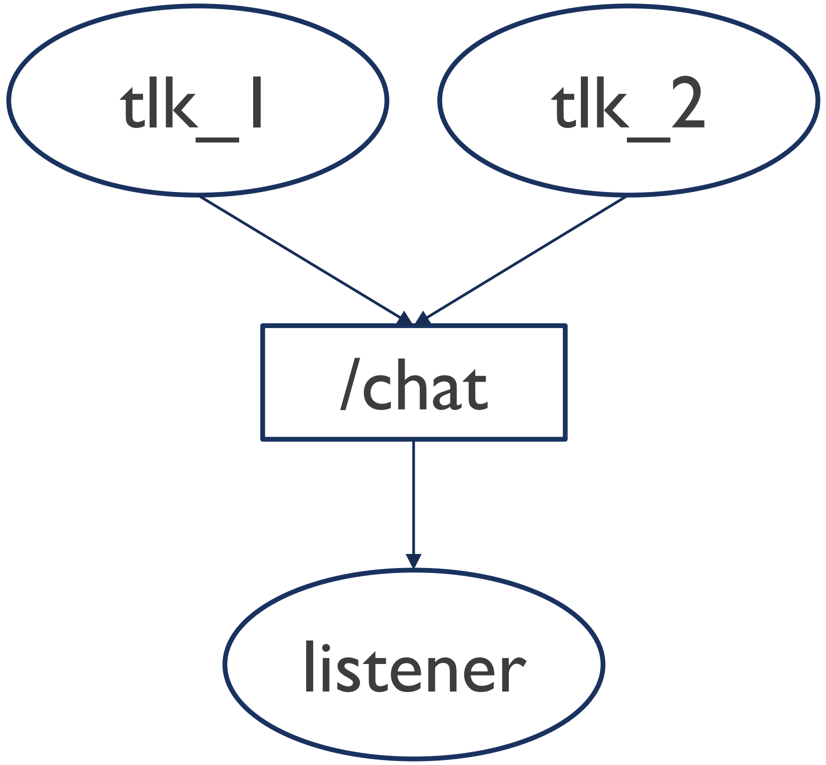
Implement the publish/subscribe paradigm

No guarantee of delivery

Have a specific message type

Multiple nodes can publish messages on a topic

Multiple nodes can read messages from a topic



MESSAGES

goo.gl/DBwhhC

Messages are exchanged on topics

They define the type of the topic

Various already available messages

It is possible to define new messages using a simple language

Existing message types can be used in new messages together with base types

```
std_msgs/Header.mgs

  uint32 seq
  time stamp
  string frame_id

std_msgs/String.msg

  string data

sensor_msgs/Joy.msg

  std_msgs/Header header

  float32[] axes

  int32[] buttons
```

MESSAGES

Messages are exchanged on topics

They define the type of the topic

Various already available messages

It is possible to define new messages using a simple language

Existing message types can be used in new messages together with base types

Quick recap:

14 base types

32 std_msgs

29 geometry_msgs

26 sensor_msgs

...and more

SERVICES

goo.gl/DBwhhC

Work like remote function calls

Implement the client/server paradigm

Code waits for service call to complete

Guarantee of execution

Use of message structures

```
example/AddTwoInt.srv
  int64 A
  int64 B
  ---
  int64 Sum
```

PARAMETER SERVER

goo.gl/DBwhhC

Shared, multivariable dictionary that is accessible via network

Nodes use this server to store and retrieve parameters at runtime

Not designed for performance, not for data exchange

Connected to the master, one of the functionalities provided by `roscore`

```
rosparam [set|get] name value

rosparm set use_sim_time True

rosparam get use_sim_time
  True
```

PARAMETER SERVER

Shared, multivariable dictionary that is accessible via network

Nodes use this server to store and retrieve parameters at runtime

Not designed for performance, not for data exchange

Connected to the master, one of the functionalities provided by `roscore`

Available types:

- 32-bit integers
- Booleans
- Strings
- Doubles
- ISO8601 dates
- Lists
- Base64-encoded binary data

BAGS

goo.gl/DBwhhC

File format (*.*bag*) for storing and playing back messages

Primary mechanism for data logging

Can record anything exchanged on the ROS graph (messages, services, parameters, actions)

Important tool for analyzing, storing, visualizing data and testing algorithms.

```
rosbag record -a
rosbag record /topic1 /topic2

rosbag play ~/bags/fancy_log.bag

rqt_bag ~/bags/fancy_log.bag
```

ROSCORE

goo.gl/DBwhhC

`roscore` is a collection of nodes and programs that are pre-requisites of a ROS-based system

Must be running in order for ROS nodes to communicate

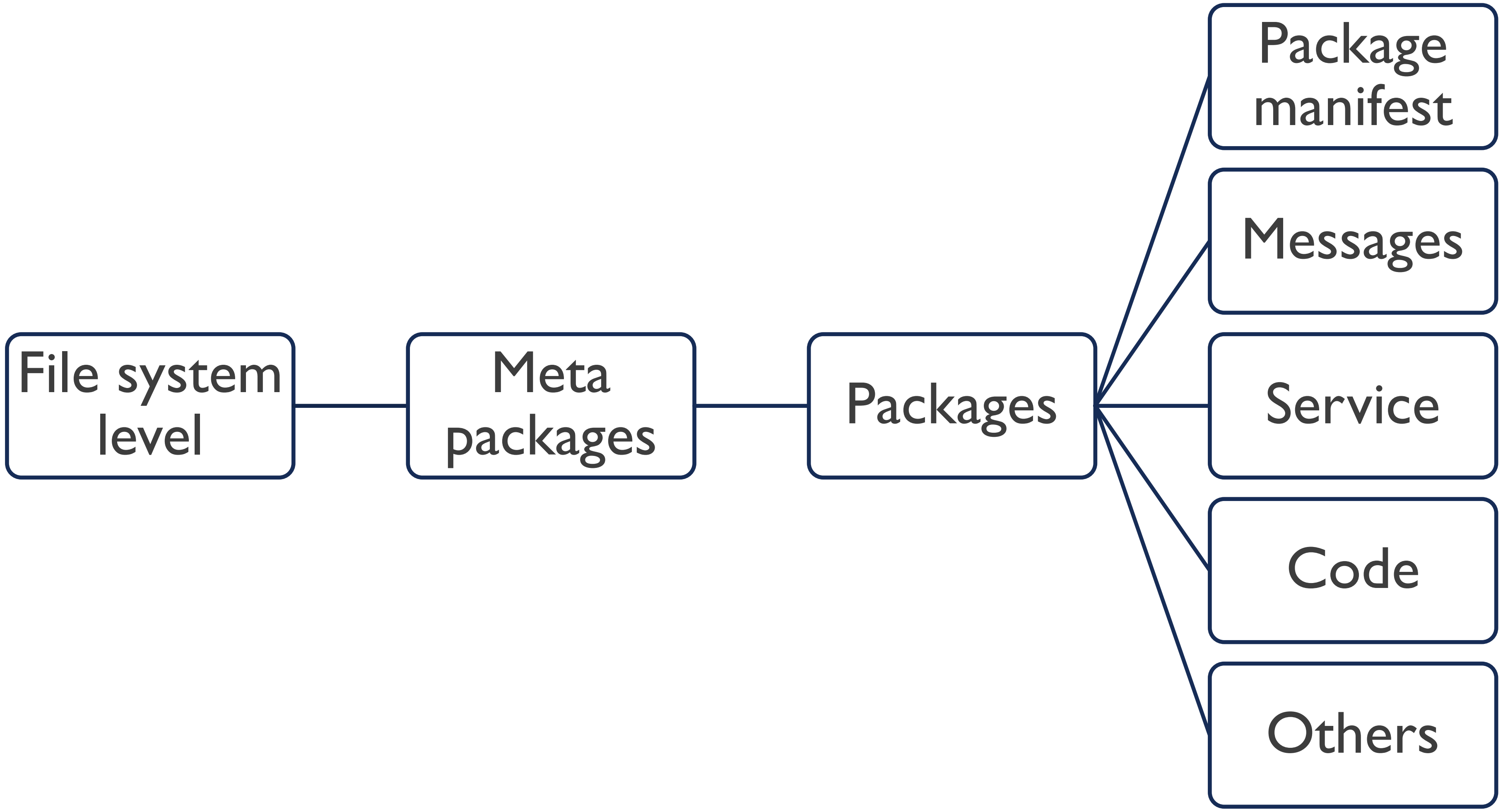
Launched using the `roscore` command.

Elements of `roscore:`

- a ROS Master
- a ROS Parameter Server
- a rosout logging node

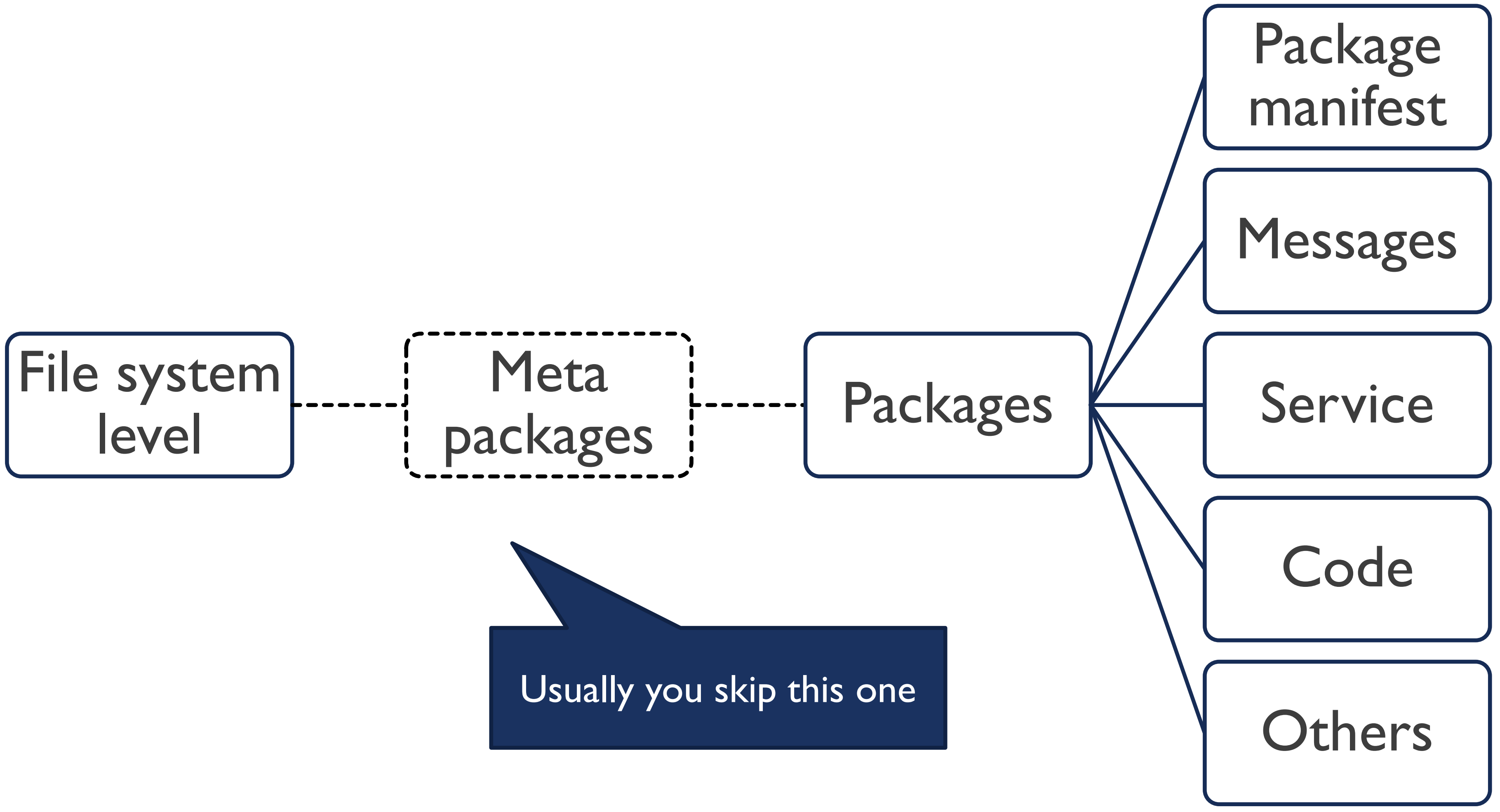
ROS FILE SYSTEM

goo.gl/DBwhhC



ROS FILE SYSTEM

goo.gl/DBwhhC



PACKAGES AND METAPACKAGES

goo.gl/DBwhhC

PACKAGES

Atomic element of ROS file system

Used as a reference for most ROS commands

Contains nodes, messages and services

`package.xml` used to describe the package

Mandatory container

METAPACKAGES

Aggregation of logical related elements

Not used when navigating the ROS file system

Contains other packages

`package.xml` used to describe the package

Not required

STRUCTURE OF A PACKAGE

goo.gl/DBwhhC

Folder structure:

`/src`, `/include`, `/scripts` (coding)

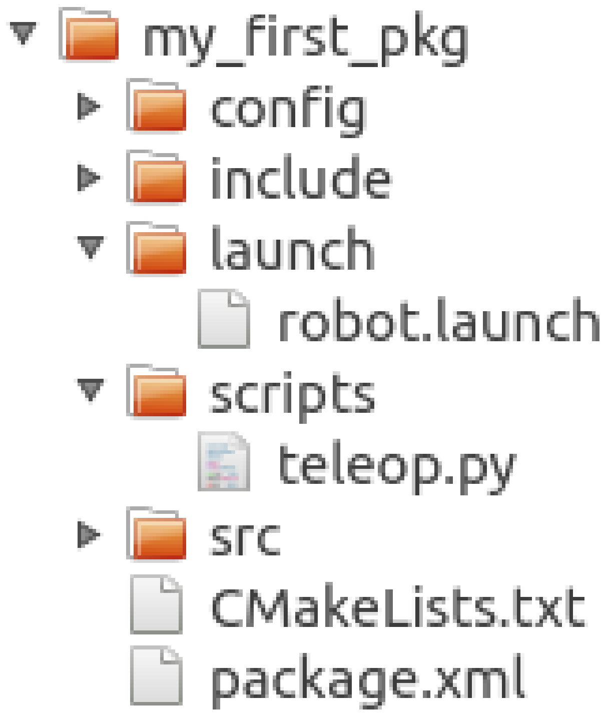
`/launch` (launch files)

`/config` (configuration files)

Required files:

`CMakeList.txt`: Build rules for catkin

`package.xml`: Metadata for ROS



```
▼ my_first_pkg
  ▶ config
  ▶ include
  ▼ launch
      robot.launch
  ▼ scripts
      teleop.py
  ▶ src
    CMakeLists.txt
    package.xml
```