ROS

ABOUT ME

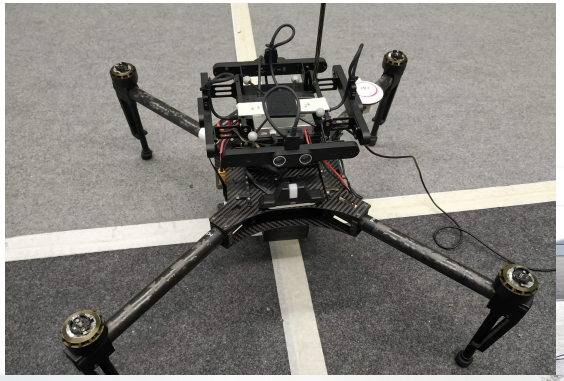
Mentasti Simone, PhD student in CS and Mechanics

Contacts:

simone.mentasti@polimi.it

Research field:

- Autonomous vehicles
- Autonomous driving
- Sensor fusion
- Deep learning



Useful info

Slide and example link:

https://goo.gl/GonArW

Slack channel:

https://join.slack.com/t/robotics2020workspace/shared_invite/zt-crwbnpth-21p0KDHC2WzpHtynLlTaDg

Schedule

L1	Middleware for robotics and ROS Installation Party	L6	ROS on multiple machines, time synchronization, stage
L2	Ros workspace, publisher/subscriber	L7	Robot Navigation (Part I)
L3	Messages, services, parameters,launch file	L8	Robot Navigation (Part II)
L4	Bags, tf, actionlib, rqt_tools	L9	Nodlet, openCV-CV-bridge
L5	Message filters, rospy. First project presentation	L10	Point Cloud Library. Second project presentation

ROBOTIC MIDDLEWARES

ROBOTICS

POLITECNICO
MILANO 1863

MIDDLEWARE ORIGINS

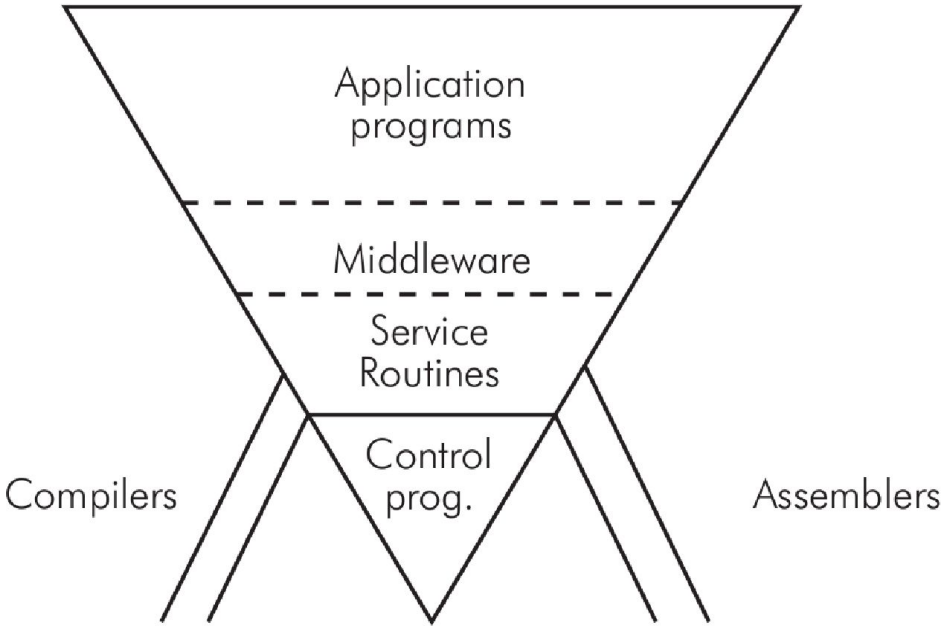
The origins

1968 introduced by d'Agapeyeff

80's wrapper between legacy systems and new applications

Nowadays: widespread in different domain fields (including Robotics)

Some (non robotics) examples: Android, SOAP, Web Services, ...



MIDDLEWARE ORIGINS

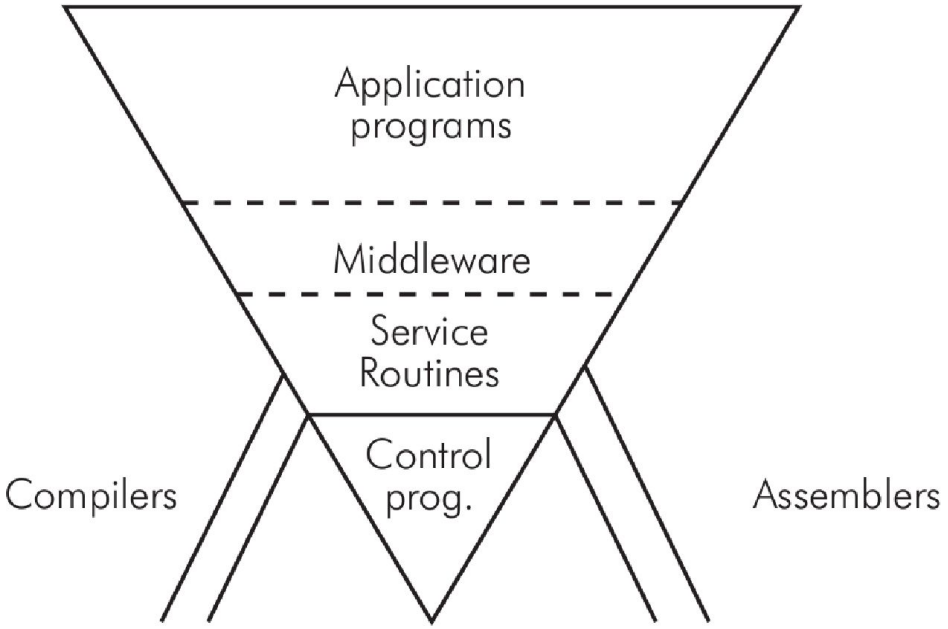
The Middleware idea

Well-known in software engineering

It provides a computational layer

A bridge between the application and the low-level details

It is not a set of API and library



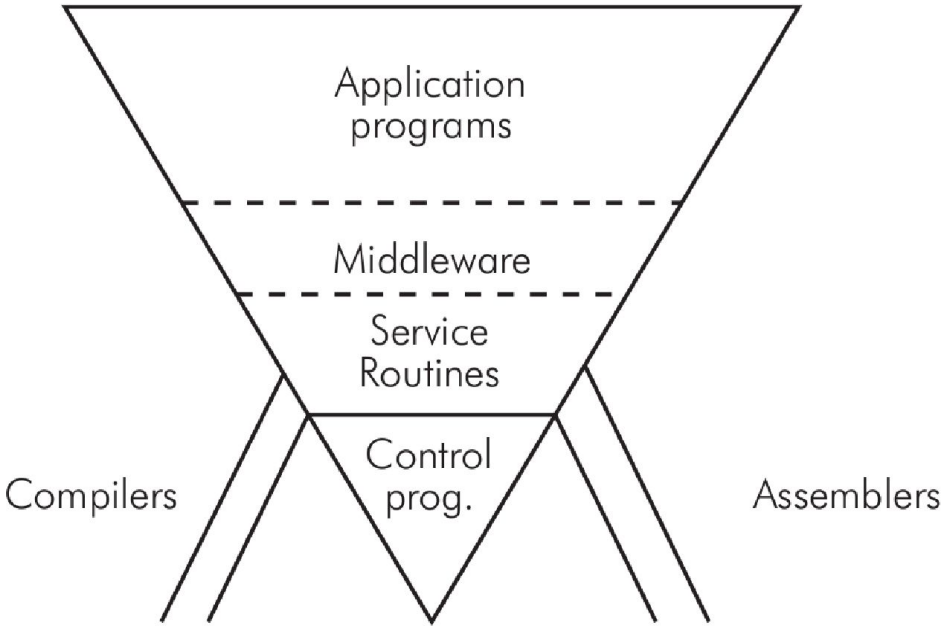
MIDDLEWARE ORIGINS

Issues in developing real robots

Cooperation between hardware and software

Architectural differences in robotics systems

Software reusability and modularity



MIDDLEWARES MAIN FEATURES

Portability*:* provides a common programming model regardless the programming language and the system architecture.

Reliability: middleware are tested independently. They permit to develop robot controllers without considering the low level details and using robust libraries.

Manage the complexity: low-level aspects are handled by libraries and drivers inside the middleware. It (should) reduce(s) the programming error and decrease the development time.

ROBOT MIDDLEWARES: A LIST

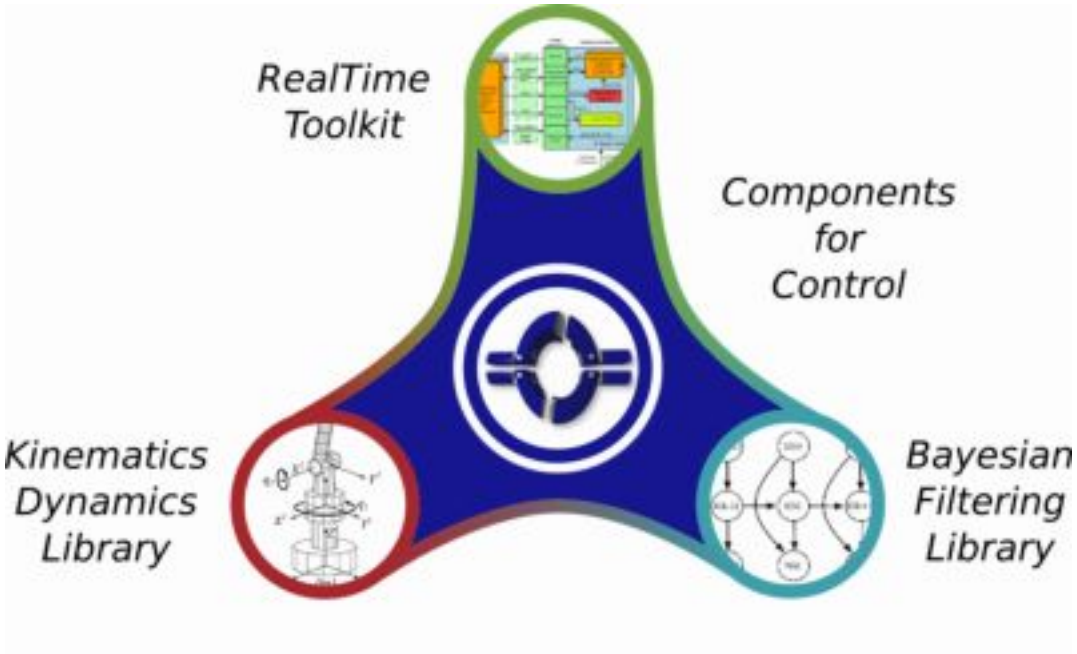
Several middleware have been developed in recent years:

OROCOS	[Europe]
ORCA	[Europe]
YARP	[Europe / Italy]
BRICS	[Europe]
OpenRTM	[Japan]
OpenRave	[US]
ROS	[US]
…	

Let’s see their common features and main differences

OROCOS: OPEN ROBOT CONTROL SOFTWARE

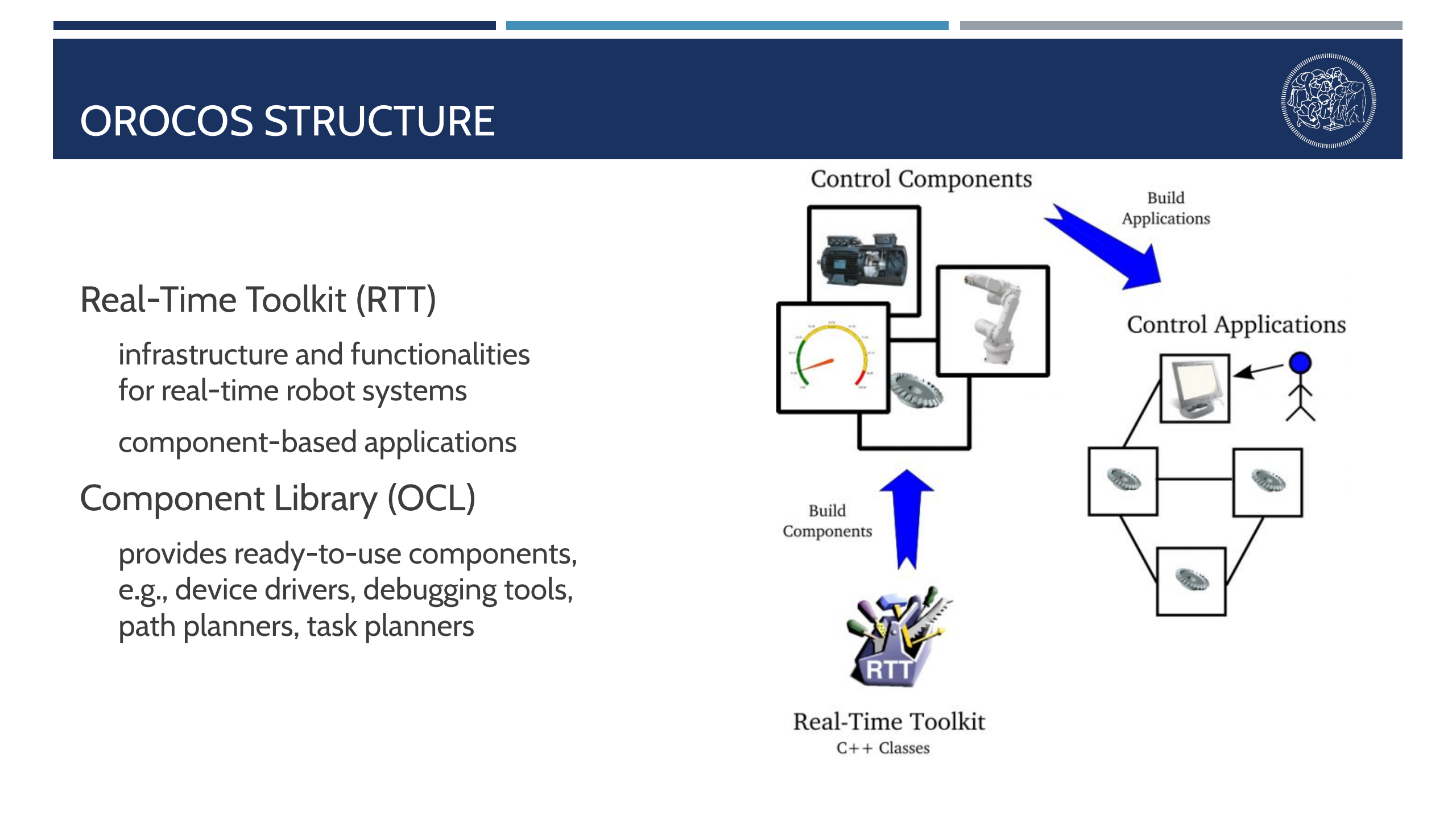
The project started in December 2000 from an initiative of the mailing list EURON then it become an European project with 3 partners: K.U. Leuven (Belgium), LAAS Toulouse (France), KTH Stockholm (Sweden)



OROCOS requirements:

- Open source license
- Modularity and flexibility
- Not related to robot industries
- Working with any kind of device
- Software components for kinematics, dynamics, planning, sensors, controller
- Not related to a unique programming language

OROCOS STRUCTURE



Real-Time Toolkit (RTT)

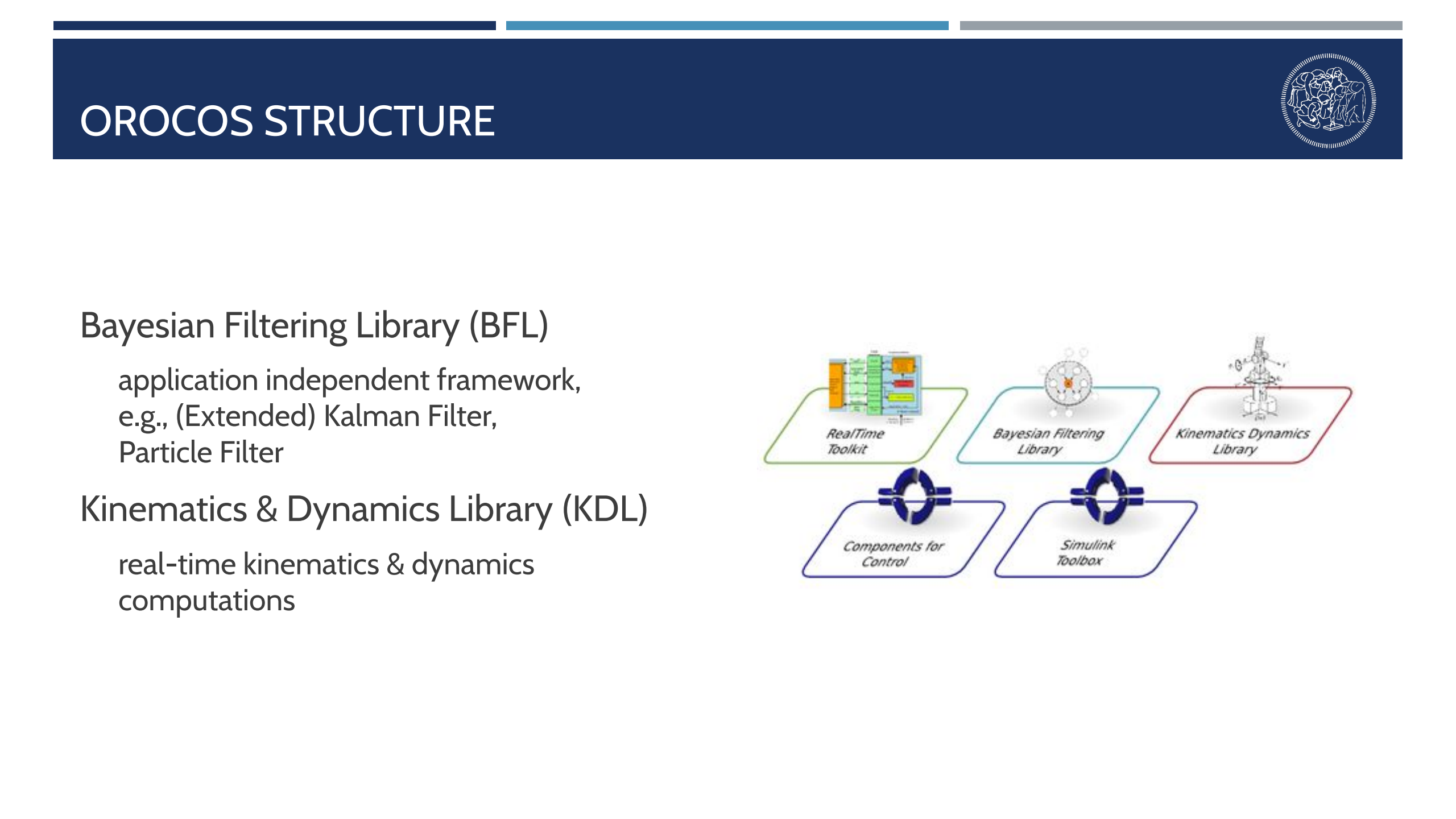
infrastructure and functionalities for real-time robot systems

component-based applications

Component Library (OCL)

provides ready-to-use components, e.g., device drivers, debugging tools, path planners, task planners

OROCOS STRUCTURE



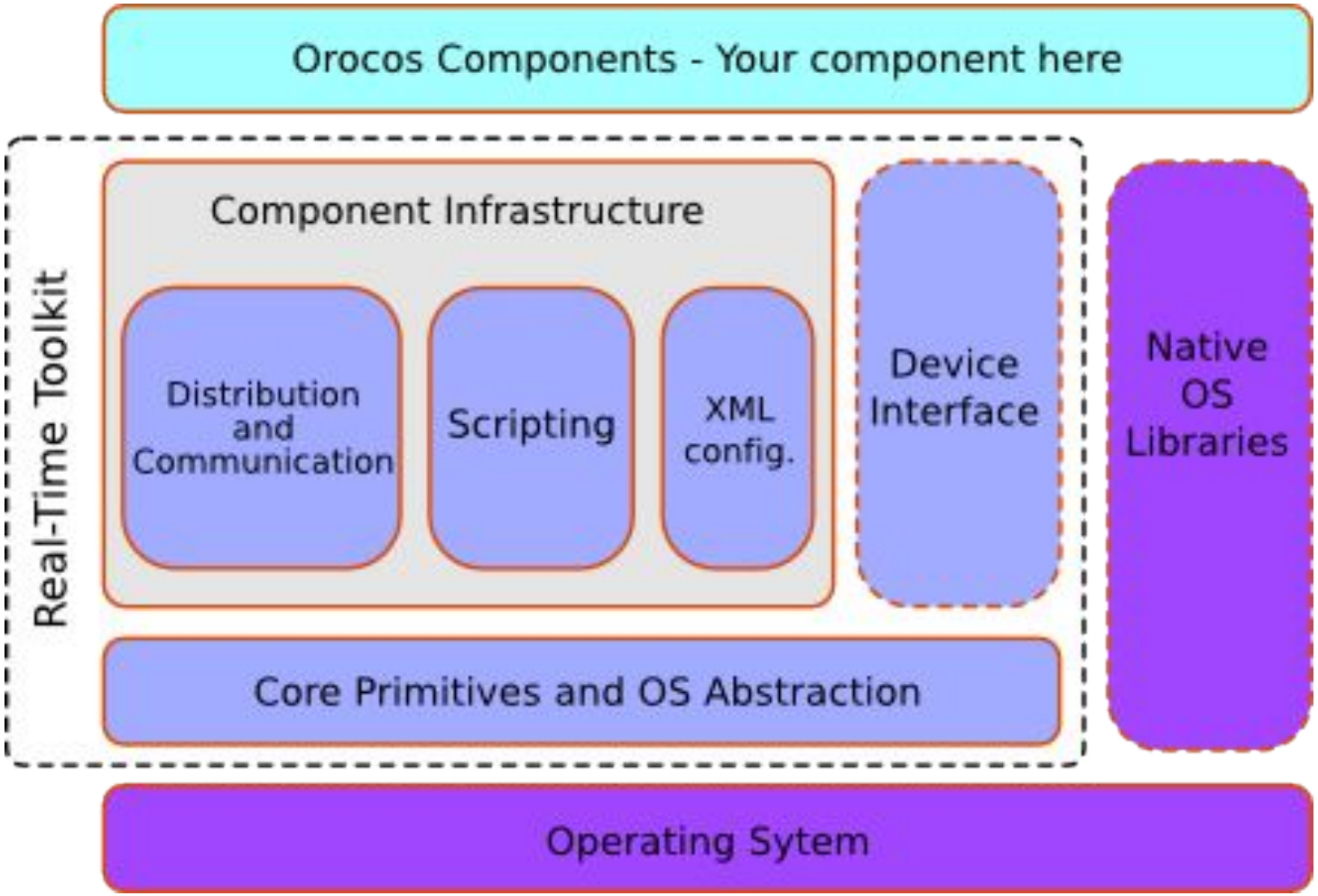
Bayesian Filtering Library (BFL)

application independent framework,
e.g., (Extended) Kalman Filter,
Particle Filter

Kinematics & Dynamics Library (KDL)

real-time kinematics & dynamics
computations

OROCOS RTT FRAMEWORK



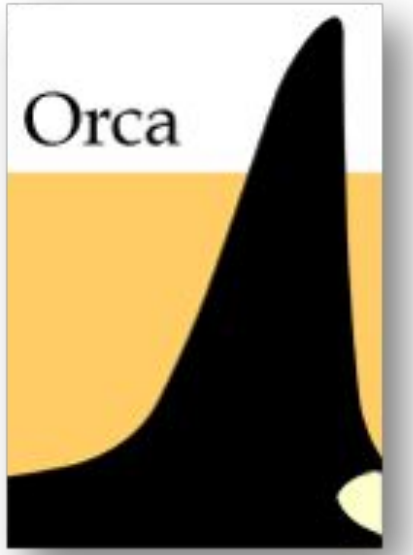
ORCA: COMPONENTS FOR ROBOTICS

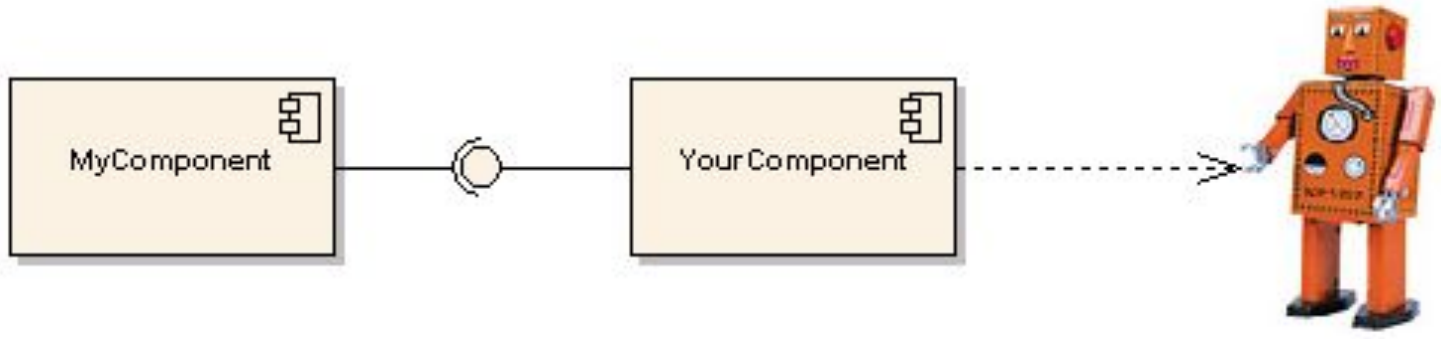
The aim of the project is to focus on software reuse for scientific and industrial applications

Key properties:

- **Enable software reuse** defining commonly-use interfaces
- **Simplify software reuse** providing high-level libraries
- **Encourage software reuse** updated software repositories

ORCA defines itself as “unconstrained component-based system”





ORCA AND ICE

The main difference between OROCOS and ORCA is the communication toolkit; OROCOS uses CORBA while ORCA uses ICE

- ICE is a modern framework developed by ZeroC
- ICE is an open-source commercial communication system
- ICE provides two core services
  - IceGrid registry (Naming service): which provides the logic mapping between different components
  - IceStorm service (Event service): which constitute the publisher and subscriber architecture

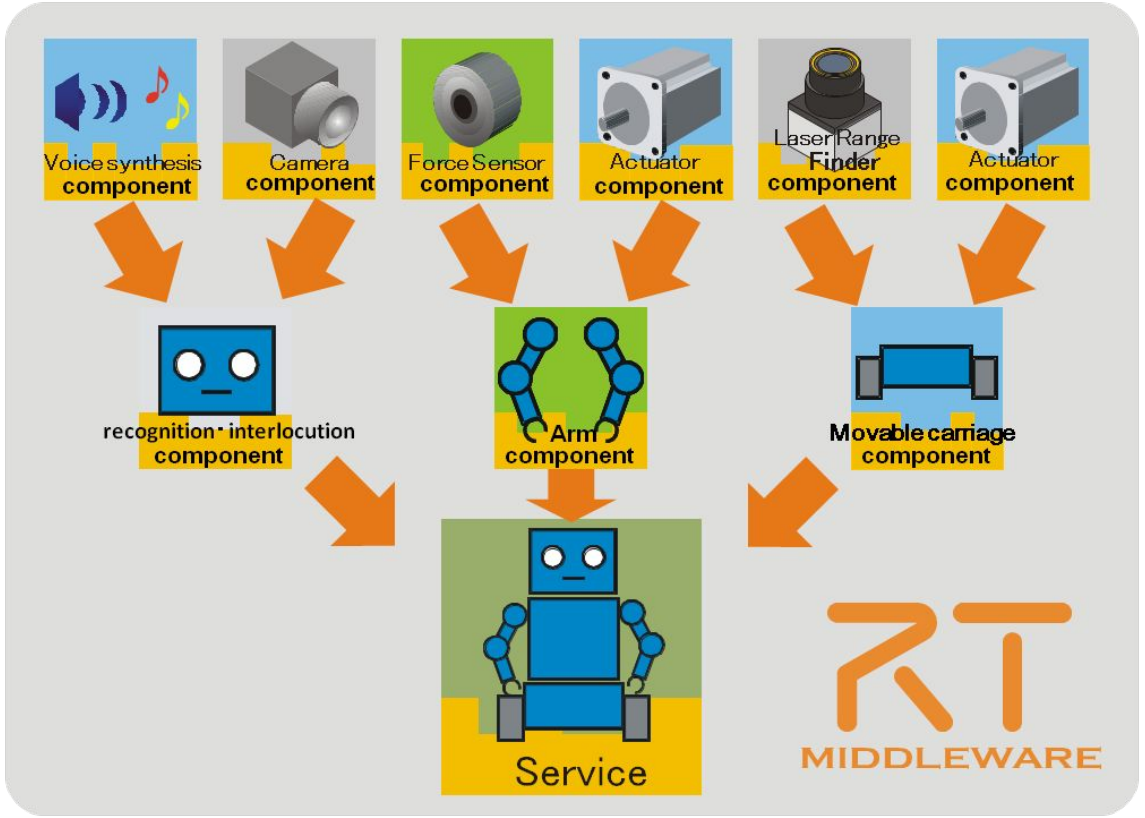
"A component can find the other components through the IceGrid registry and can communicate with them through the IceStorm service."

RT MIDDLEWARE

RT-Middleware (RTM) is a common platform standard to construct the robot system by combining the software modules of the robot functional elements (RTC):

- Camera component
- Stereovision component
- Face recognition component
- Microphone component
- Speech recognition component
- Conversational component
- Head and arm component
- Speech synthesis component

OpenRTM-aist (Advanced Industrial Science & Technology) is based on the CORBA technology to implement RTC extended specification

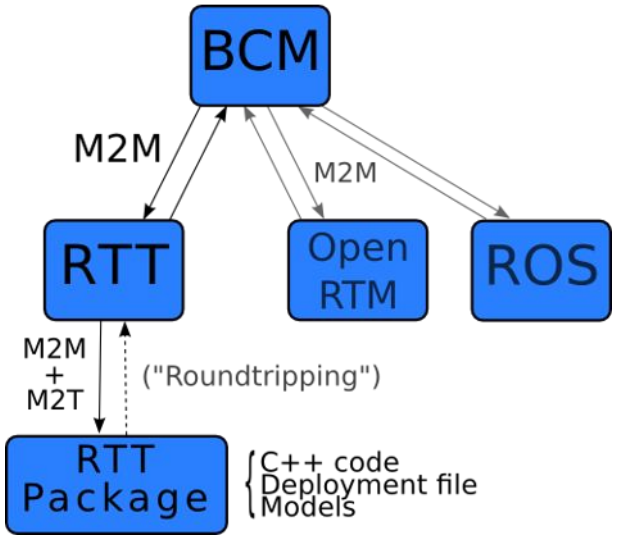


BRICS: BEST PRACTICES IN ROBOTICS

Aimed at find out the "best practices" in the developing of the robotic systems:

- Investigate the weakness of robotic projects
- Investigates the integration between hardware & software
- Promote model driven engineering in robot development
- Design an Integrated Development Environment for robotic projects (BRIDE)
- Define showcases for the evaluation of project robustness with respect to BRICS principles.

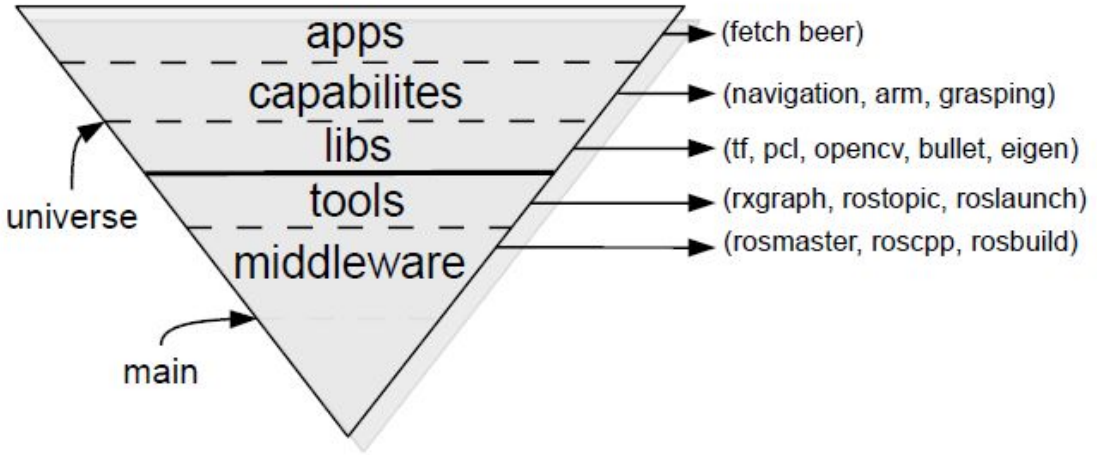
BRICS
BEST PRACTICE IN ROBOTICS



"The prime objective of BRICS is to structure and formalize the robot development process itself and to provide tools, models, and functional libraries, which help accelerating this process significantly."

ROS: ROBOT OPERATING SYSTEM

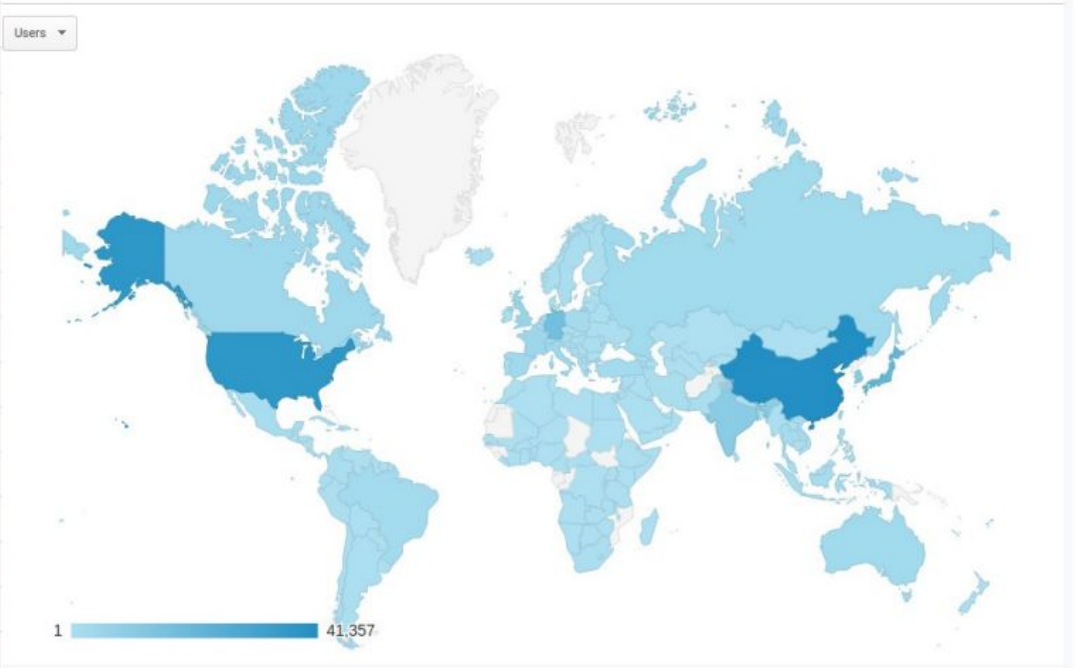
Presented in 2009 by Willow Garage, is a meta-operating system for robotics with a rich ecosystem of tools and programs



WHY ROS?

1.	China	41,357	(19.88%)
2.	United States	36,531	(17.56%)
3.	Japan	19,738	(9.49%)
4.	Germany	15,525	(7.46%)
5.	South Korea	9,382	(4.51%)
6.	India	9,345	(4.49%)
7.	United Kingdom	4,972	(2.39%)
8.	Taiwan	4,856	(2.33%)
9.	France	4,056	(1.95%)
10.	Canada	3,854	(1.85%)
11.	Singapore	3,516	(1.69%)
12.	Italy	3,464	(1.66%)
13.	Russia	3,207	(1.54%)
14.	Australia	3,114	(1.50%)
15.	Spain	3,080	(1.48%)
16.	Hong Kong	2,941	(1.41%)
17.	Brazil	2,548	(1.22%)
18.	Turkey	2,253	(1.08%)
19.	Netherlands	1,822	(0.88%)
20.	Poland	1,820	(0.87%)
21.	Philippines	1,711	(0.82%)
22.	Thailand	1,585	(0.76%)
23.	Switzerland	1,478	(0.71%)
24.	(not set)	1,429	(0.69%)
25.	Indonesia	1,345	(0.65%)

wiki.ros.org visitor locations:



Source: Google Analytics
Site: wiki.ros.org in July 2019

ROS has grown to include a large community of users worldwide

The community of developer is one of the most important characteristics of ROS

A LOT OF RESOURCES

ROS Wiki

- Archive for the existing ROS component
- Installation and configuration guides
- Information about the middleware itself
- Lots of tutorials

ROS Q&A

- For specific problems
- Thousand of already answered questions
- Active community
- Like *Stack Overflow* for ROS

SOME NUMBERS

ROS wiki:

- pages: 17058
- edits: 14,7/day
- views: 44794/day

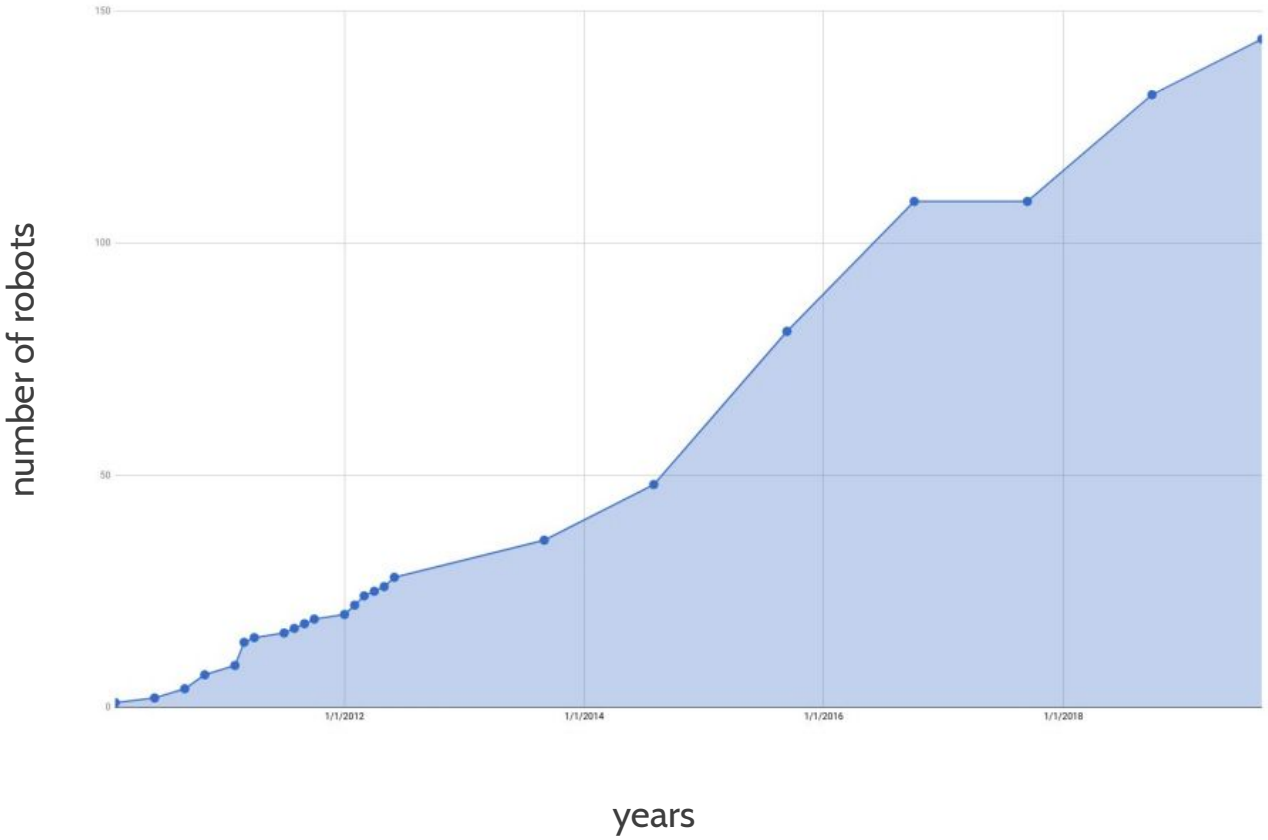
ROS Q&A:

- total Q: 30243
- total A: 21697
- avg Q: 17,2/day

ROS deb:

- total DL: 8441279
- unique DL: 7582
- unique IP: 113345

ROBOT AND RESEARCH



Total number of papers citing *ROS: an open-source Robot Operating System* (Quigley et al., 2009)
5875 (22% increase)
6703 (scholar)

ROS INTRODUCTION

ROBOTICS

POLITECNICO
MILANO 1863

ROS: ROBOT OPERATING SYSTEM

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

ROS

Statistics

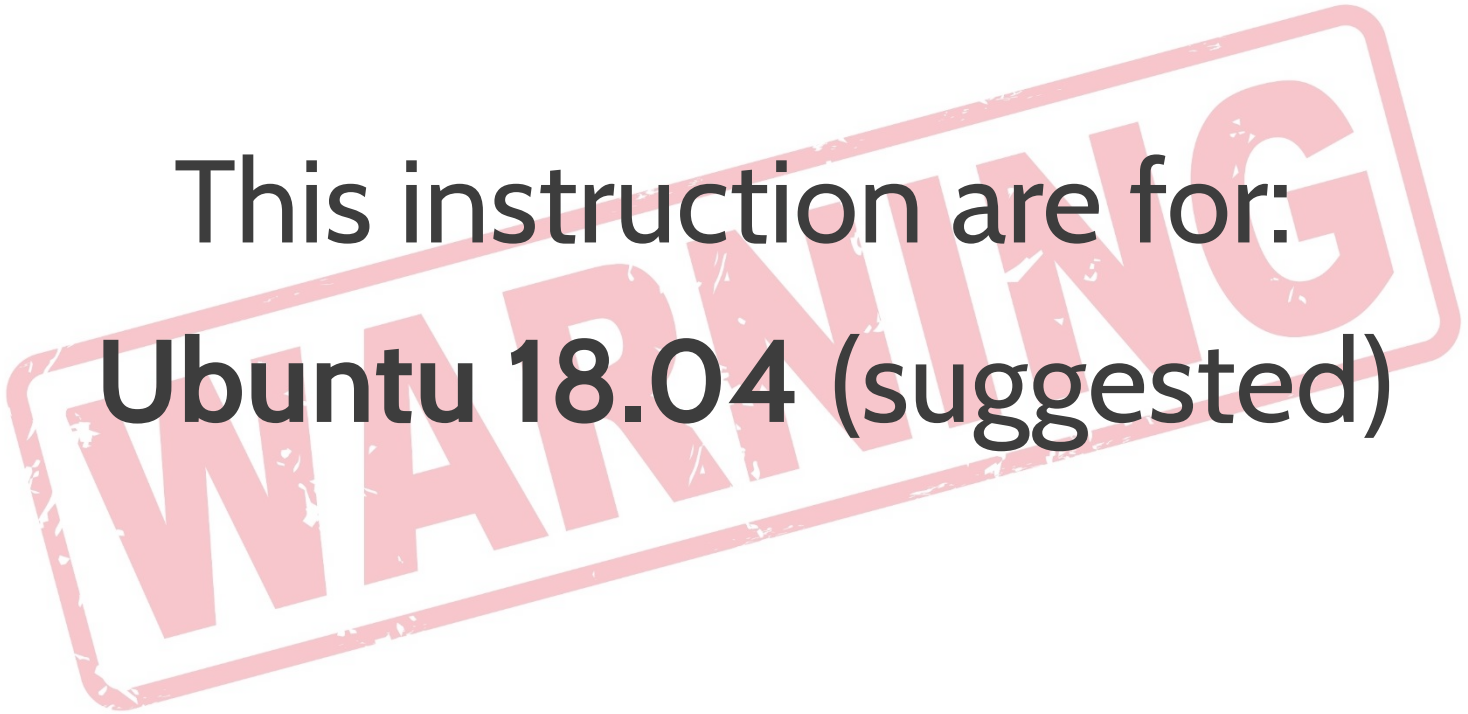
Downloads for ROS distro:

- Indigo: 5.02 % (Long Term Support Release)
- Jade: 0.00 %
- Kinetic: 53.06 % (Long Term Support Release)
- Lunar: 0.85 %
- Melodic: 26.71 % (Long Term Support Release)
- Bouncy and earlier ROS 2: 0.12 %
- Crystal 0.62 %
- Dashing 1.51 % (Long Term Support Release)

INSTALLATION

This instruction are for:

Ubuntu 18.04 (suggested)



INSTALLATION

Check on ROS site for updated commands:
http://wiki.ros.org/melodic/Installation/Ubuntu

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://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

INSTALLATION

Update the packaged index

sudo apt-get update

Choose between the four pre-packaged ROS installation

Desktop-Full Install: sudo apt-get install ros-melodic-desktop-full

Desktop Install: sudo apt-get install ros-melodic-desktop

ROS-Base: sudo apt-get install ros-melodic-ros-base

INSTALL

How to install single packages:

sudo apt-get install ros-melodic-PACKAGE

Example

sudo apt-get install ros-melodic-slam-gmapping

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

apt-cache search ros-melodic

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/melodic/setup.bash

Or put it inside your .bashrc

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

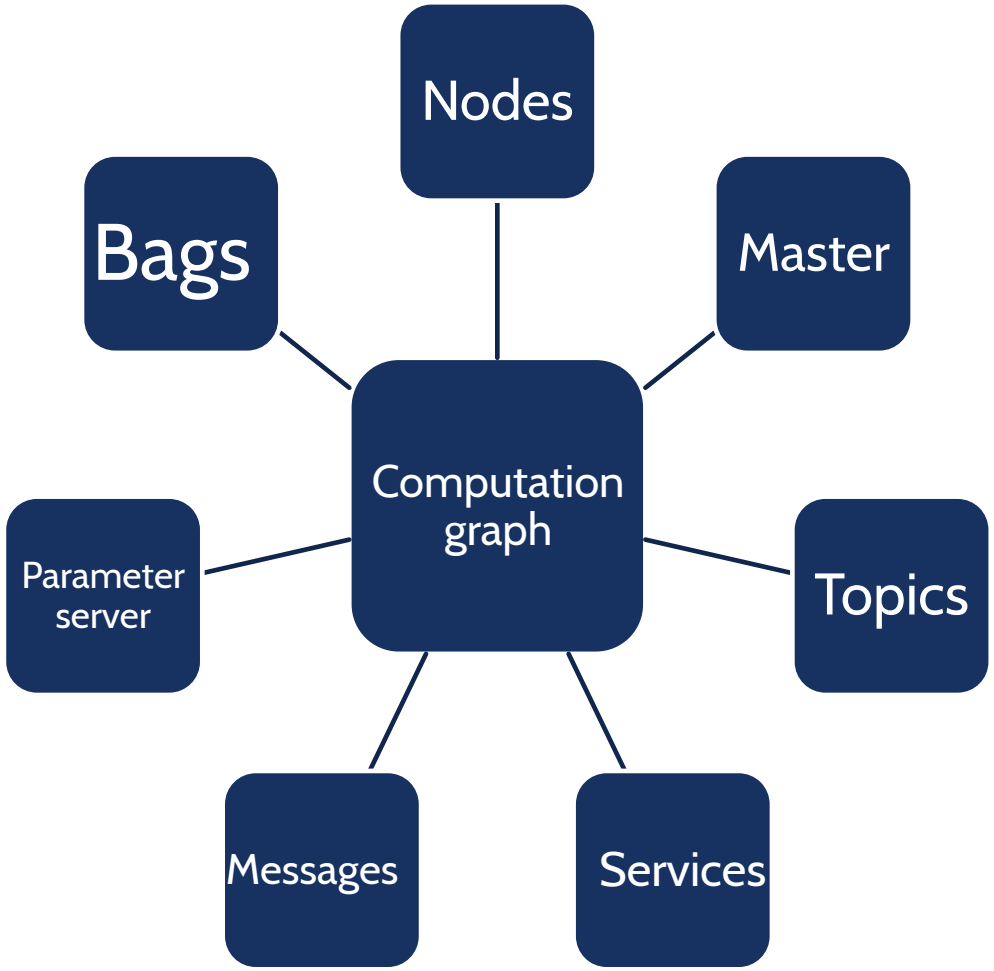
source ~/.bashrc

ROS STRUCTURE

ROBOTICS

POLITECNICO
MILANO 1863

ROS STRUCTURE: COMPUTATIONAL GRAPH



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

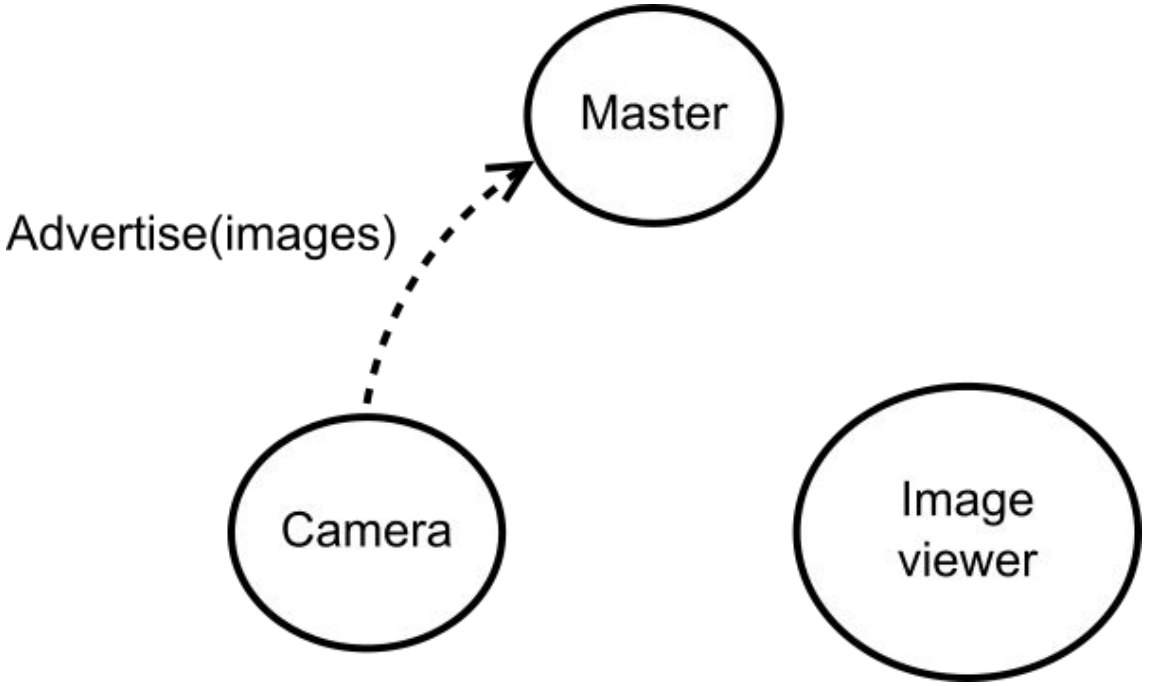
Meant to operate at fine-grained scale

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

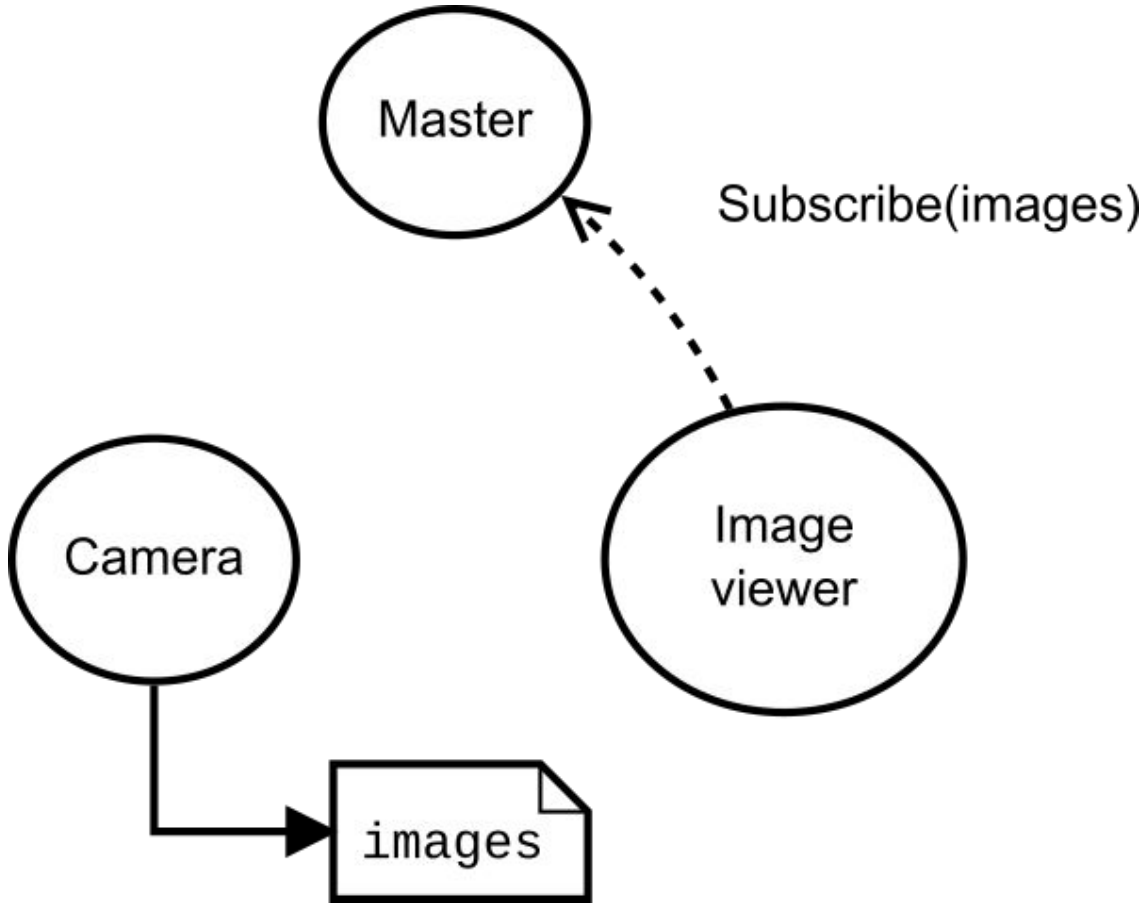
Provides naming and registration services

Essential for nodes interactions

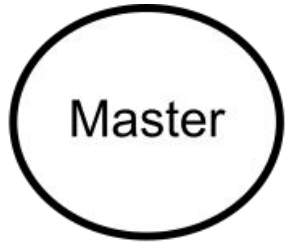
One master for each system, even on distributed architectures

Enables individual ROS nodes to locate one another

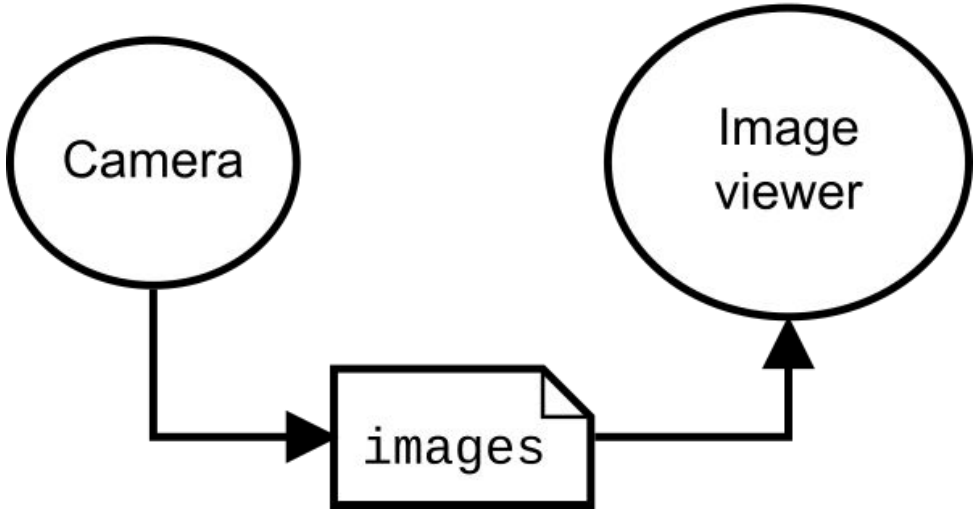
One of the functionalities provided by roscore



MASTER

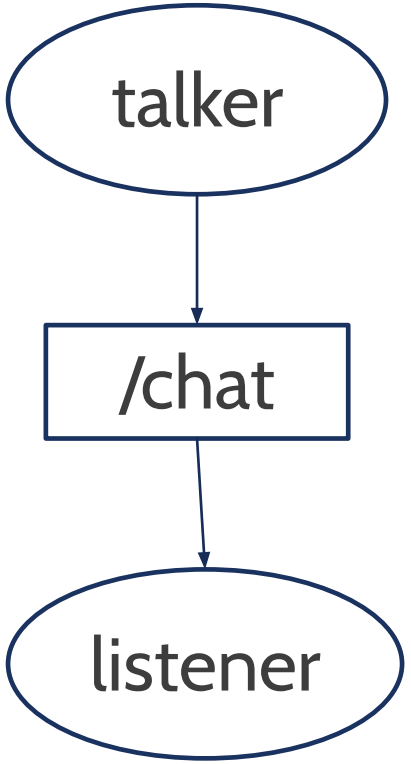


- Provides naming and registration services
- Essential for nodes interactions
- One master for each system, even on distributed architectures
- Enables individual ROS nodes to locate one another
- One of the functionalities provided by roscore



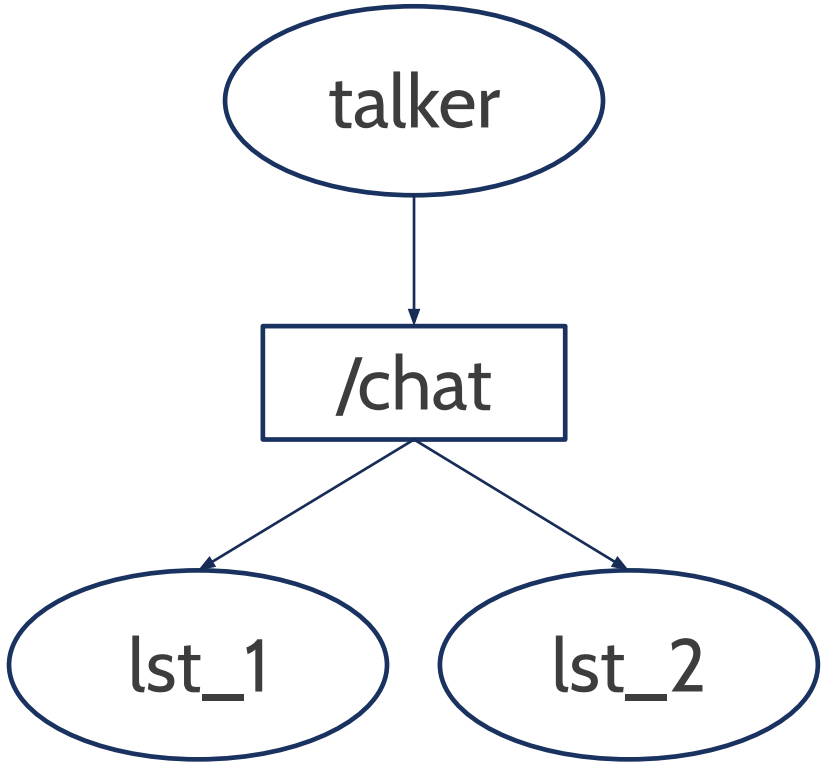
TOPICS

- 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



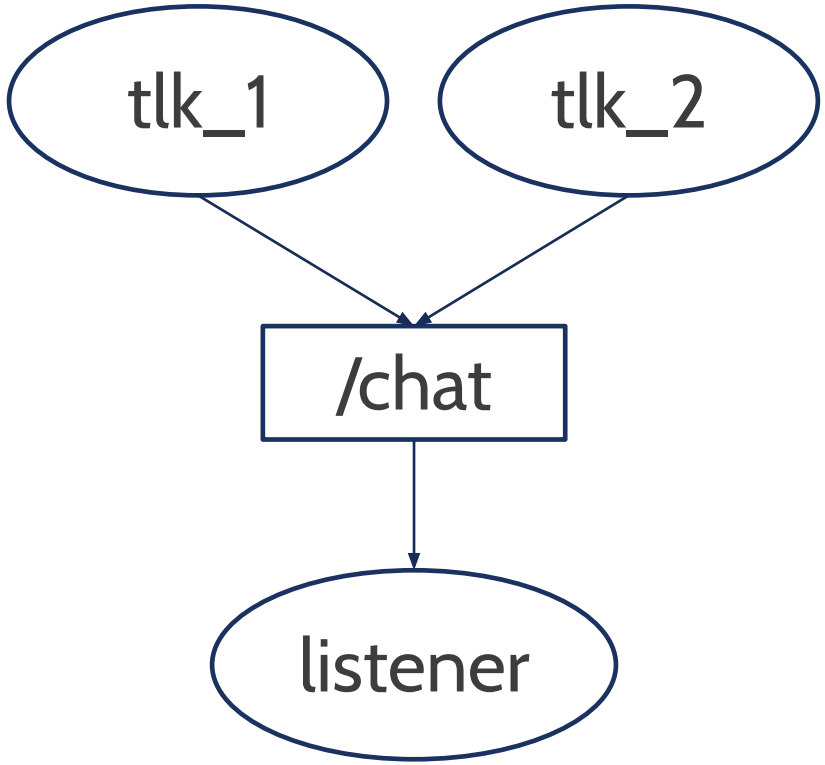
TOPICS

- 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

- 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

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

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

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

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

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

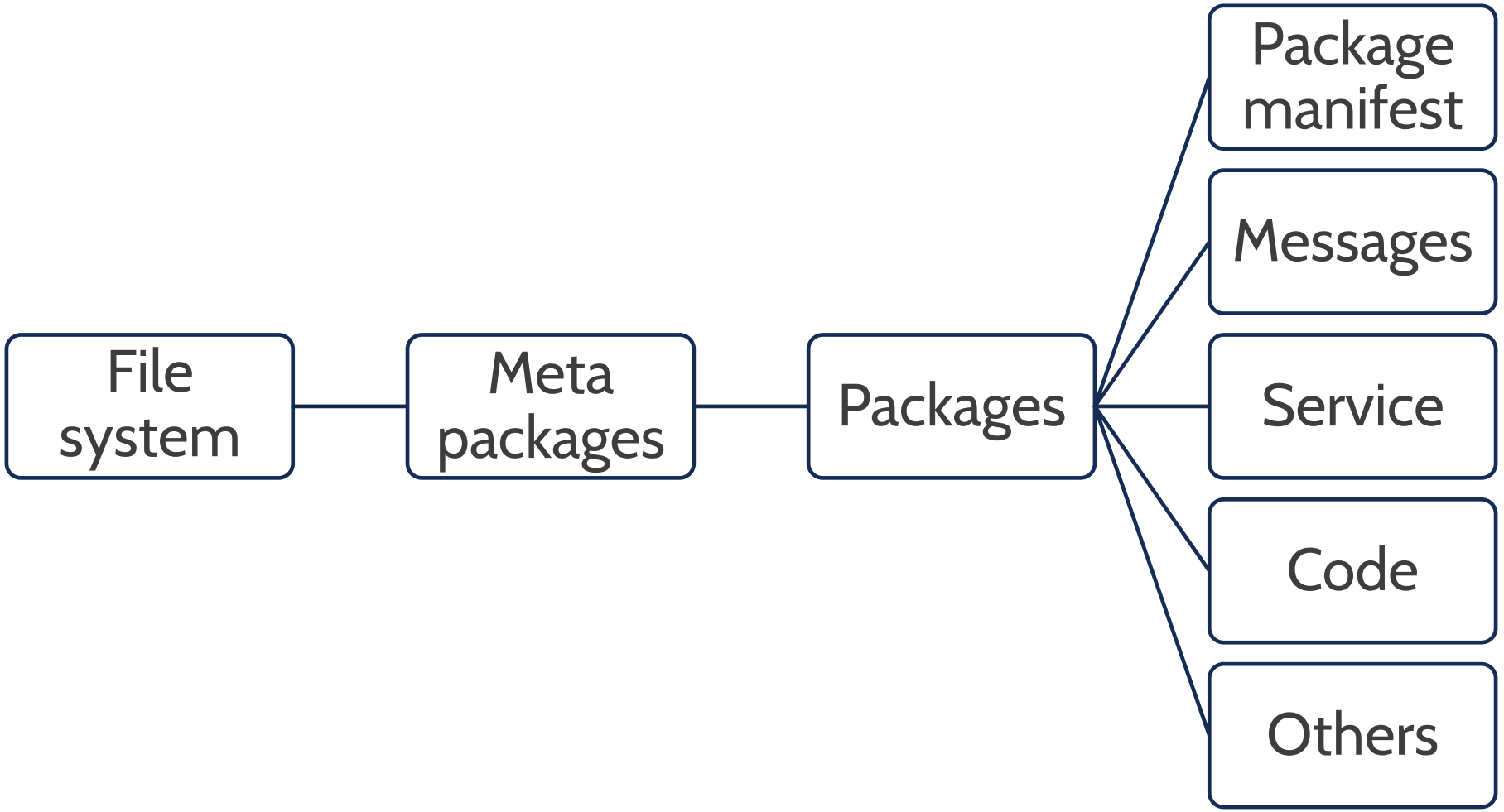
Must be running in order for ROS nodes to communicate

Launched using the roscore command.

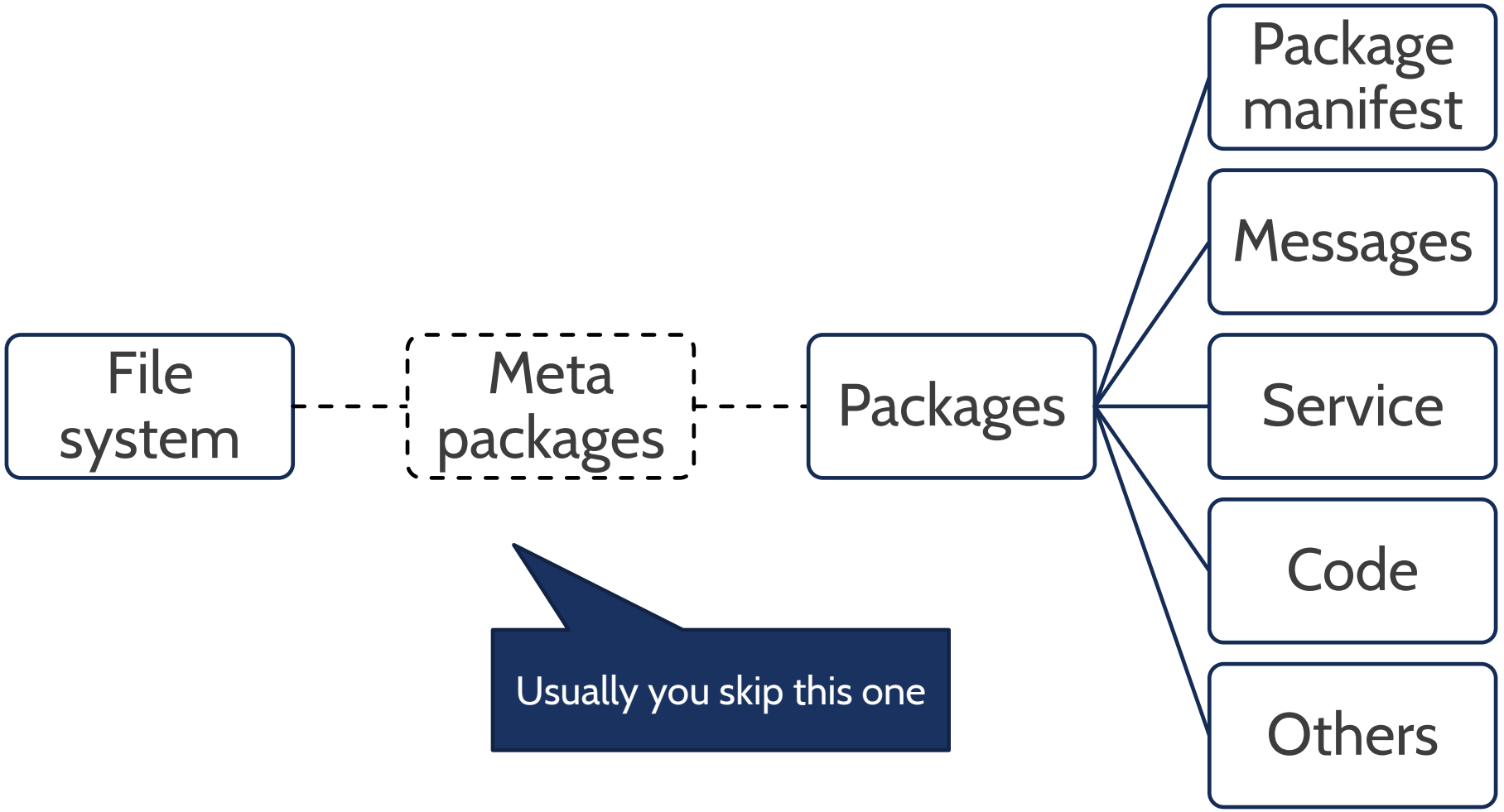
Elements of roscore:

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

ROS FILE SYSTEM



ROS FILE SYSTEM



PACKAGES AND METAPACKAGES

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

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