ROS

POLITECNICO
MILANO 1863

ABOUT ME

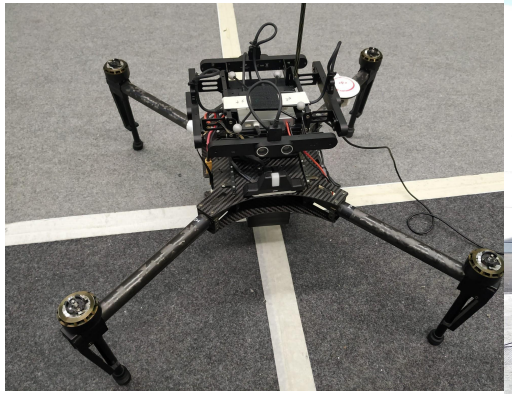
Mentasti Simone, researcher at AIRLab

Contacts:

simone.mentasti@polimi.it

Research field:

- Autonomous vehicles
- Sensor fusion
- Deep learning
- Smart eyewear
- Agriculture robotics
- ...



Useful info

Slide and example link:

https://goo.gl/GonArW

Schedule

L1	Middleware for robotics and ROS What to install and survive the course	L6	rospy, rosbag, message filter
L2	Docker intro Ros workspace, publisher/subscriber, tools	L7	ROS on multiple machines, time synchronization, Actionlib, latched pub,async spinner
L3	Publisher, subscriber, launch file , custom messages	L8	Robot Navigation, Stage, slam toolbox, gmapping
L4	Services, parameters, timers	L9	Robot Navigation (Part II), movebase, amcl
L5	TF, Rviz, node architecture, **first project**	L10	Future of ROS, ROS2, **second project**

Challenges

Alternative to doing **the projects**

Require **more time** than doing the project

Experiment on real robots

Will have soft deadlines (but we have deadlines)

Competitions are after the end of the course, we still evaluate when the course finish (not on the challenge results)

ROS2 based

If interested, have question, etc. write a mail to us:

- Subject “Challenge Robotics 2024 - Leonardo”/“Challenge Robotics 2024 - FRE”
- Attach study plan, specify study course and year
- write a few lines where you explain why you are interested in the challenge

UGV-UAV cooperation (Leonardo drone contest)

Organized by Leonardo

5th edition, 2nd edition with UGV

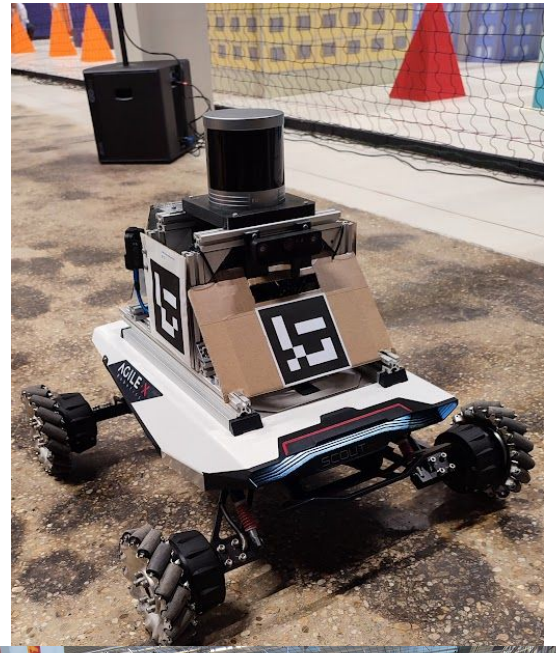
Tasks:

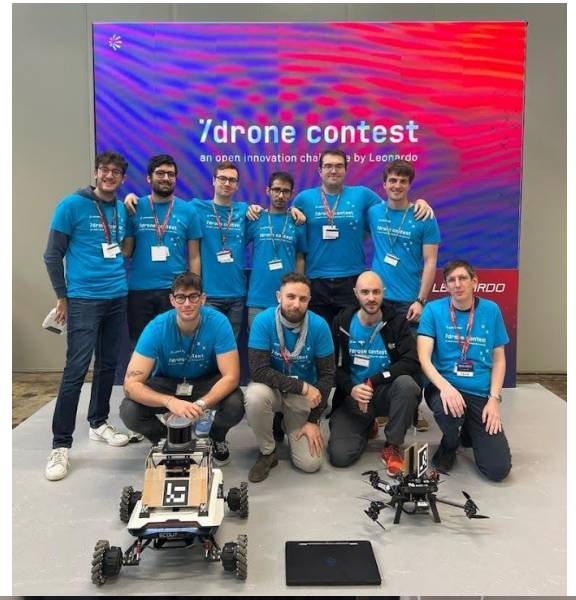
- exploration
- mapping
- detection
- tracking
- cooperation

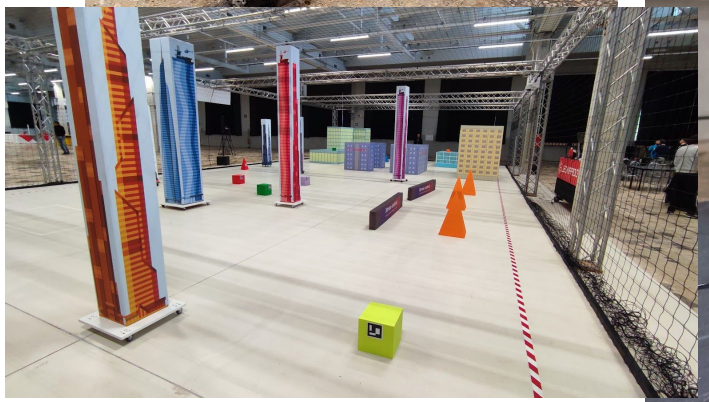
Urban-like environment

Competition in October (Turin)

mail to: simone.mentasti@polimi.it







Agriculture robot (Field Robotics Event)

Agriculture competition

3rd year for our lab

newly built custom robot

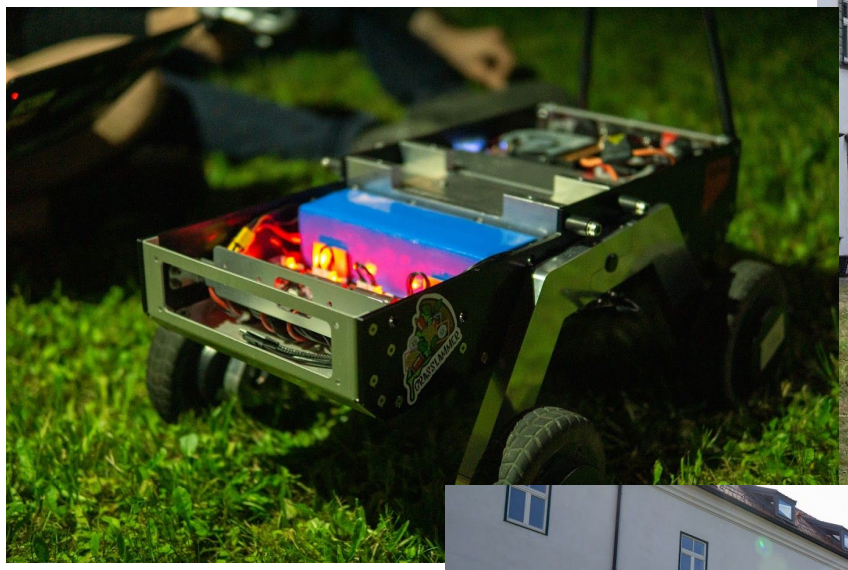
tasks:

- autonomous navigation
- obstacle detection
- weed detection
- autonomous spraying

Competition in June (Germany)

mail to: mirko.usuelli@polimi.it

simone.mentasti@polimi.it



ROBOTIC MIDDLEWARES

ROBOTICS

POLITECNICO
MILANO 1863

MIDDLEWARE ORIGINS

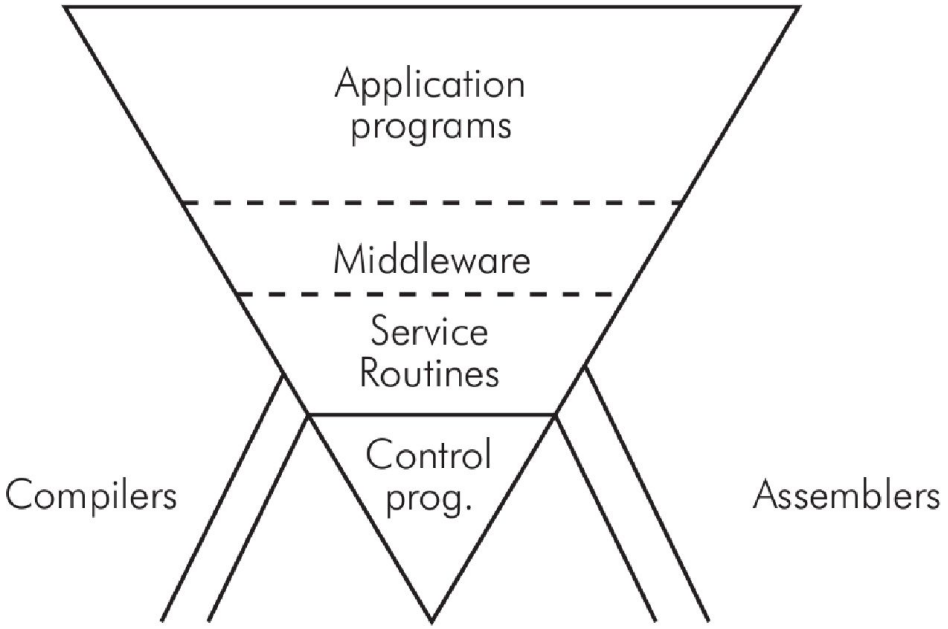
The origins

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

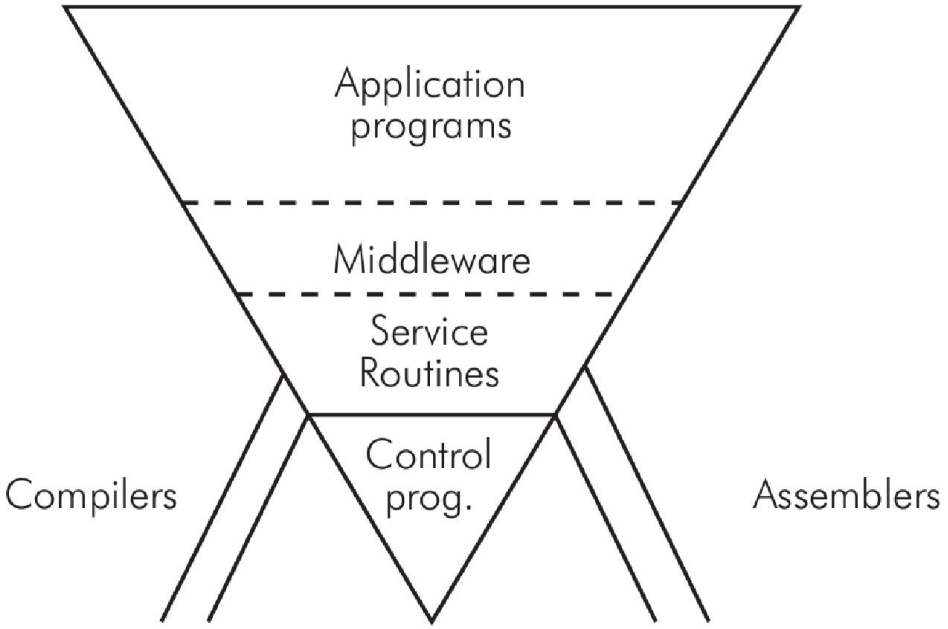
The Middleware idea

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



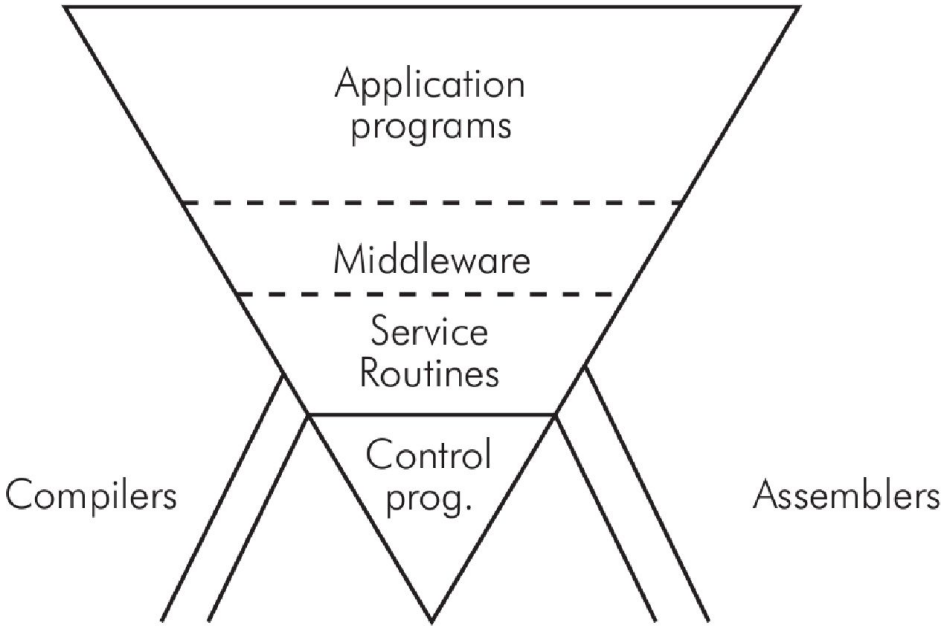
MIDDLEWARE ORIGINS

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

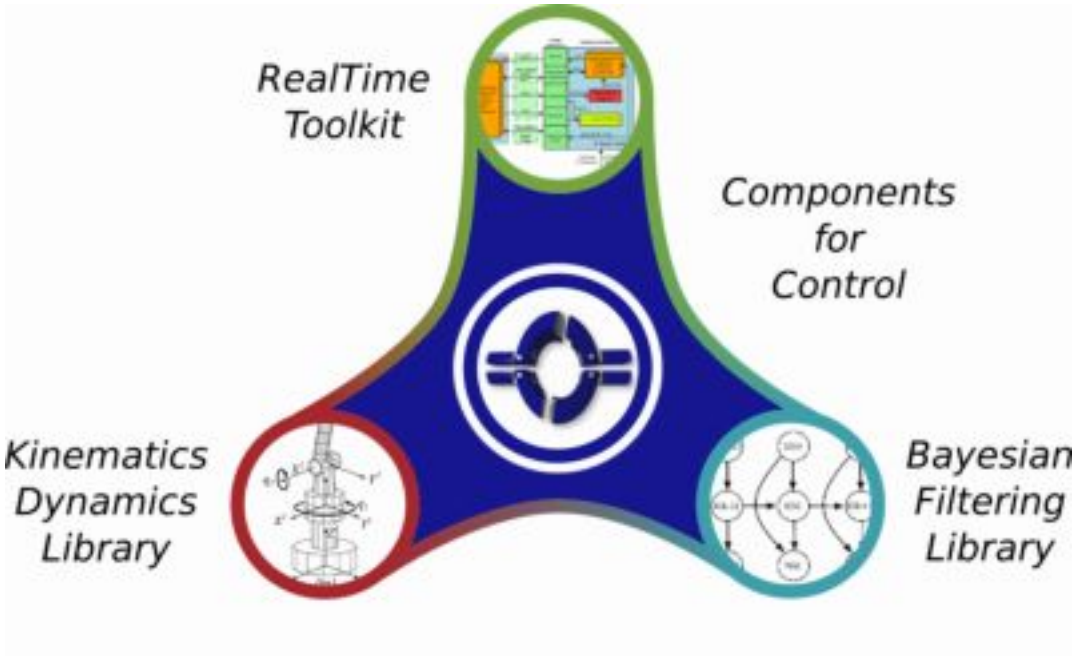
Several middleware have been developed in recent years:

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

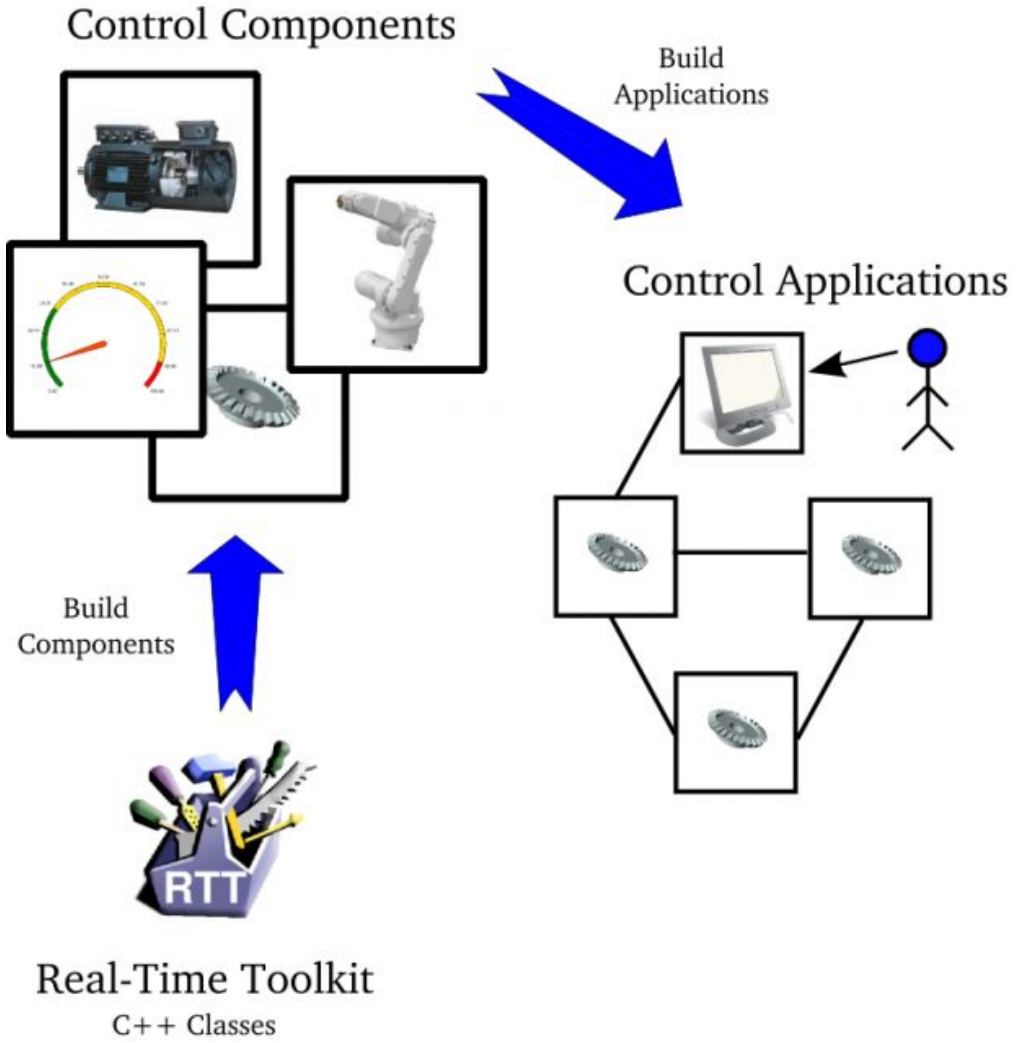
The project started in December 2000 from an initiative of the mailing list EURON then it become an European project with 3 partners: K.U. Leuven (Belgium), LAAS Toulouse (France), KTH Stockholm (Sweden)



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)

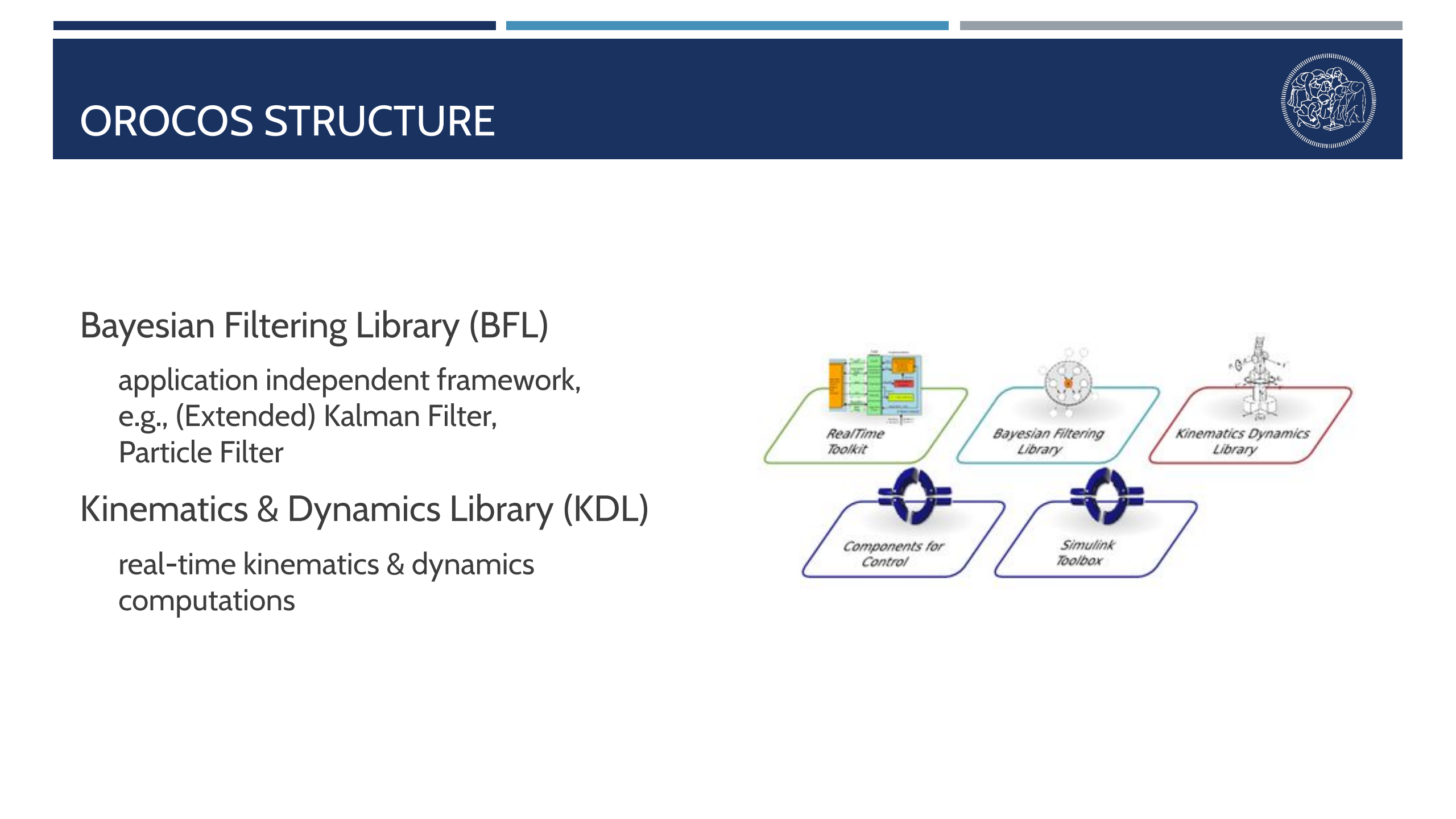
infrastructure and functionalities for real-time robot systems

component-based applications

Component Library (OCL)

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

OROCOS STRUCTURE



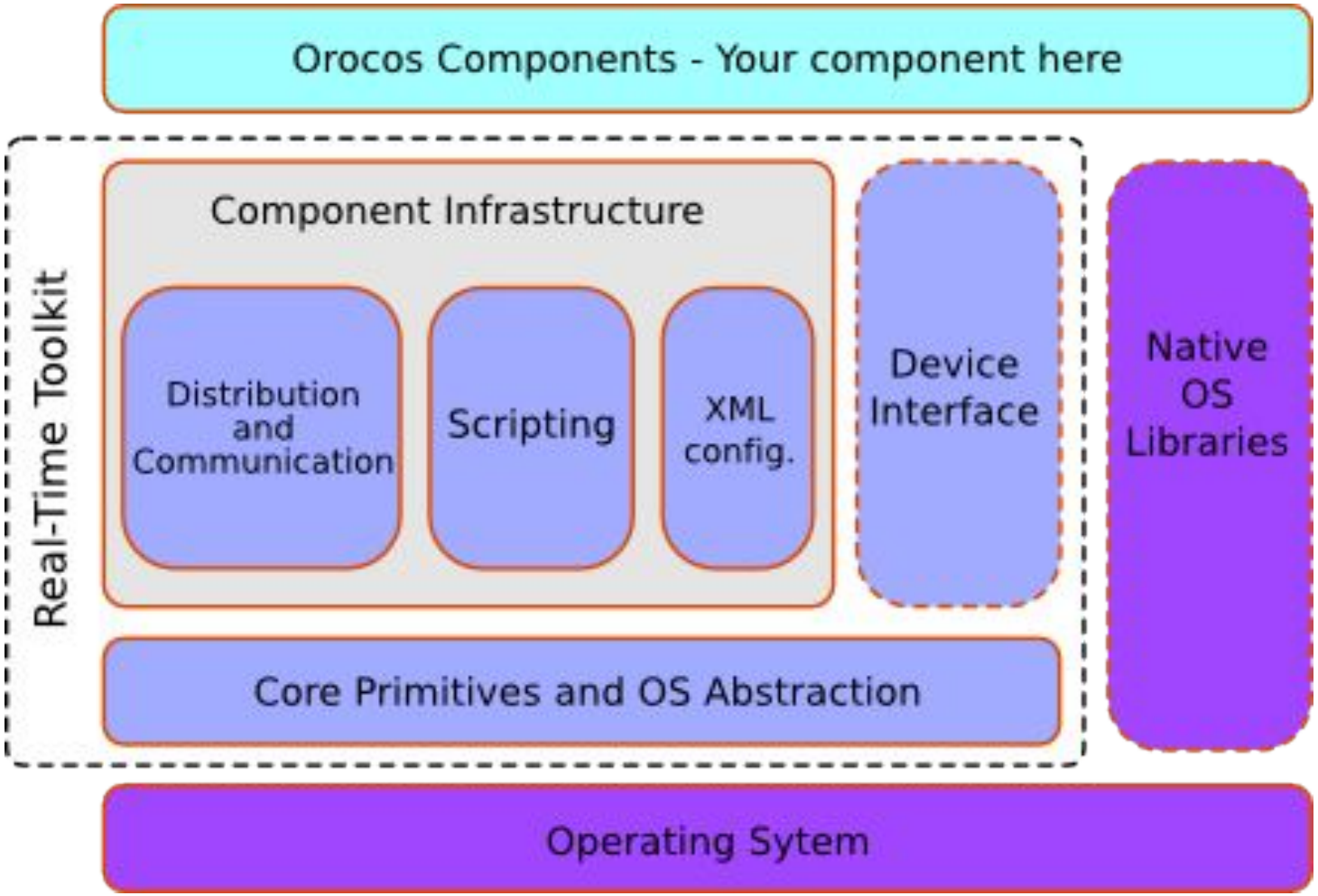
Bayesian Filtering Library (BFL)

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

Kinematics & Dynamics Library (KDL)

real-time kinematics & dynamics
computations

OROCOS RTT FRAMEWORK



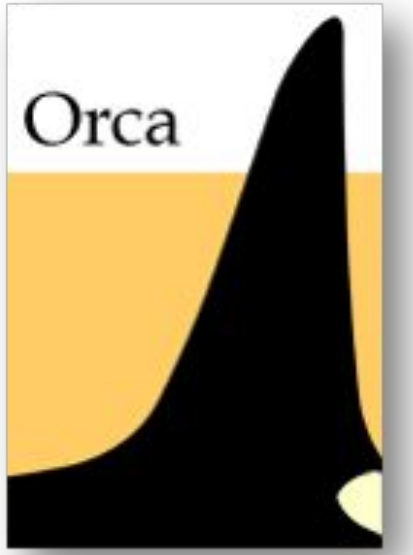
ORCA: COMPONENTS FOR ROBOTICS

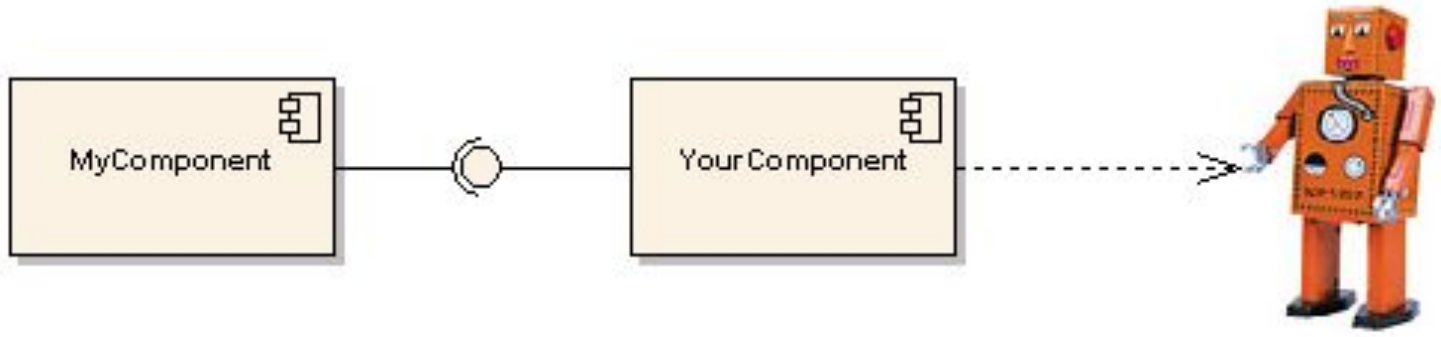
The aim of the project is to focus on software reuse for scientific and industrial applications

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

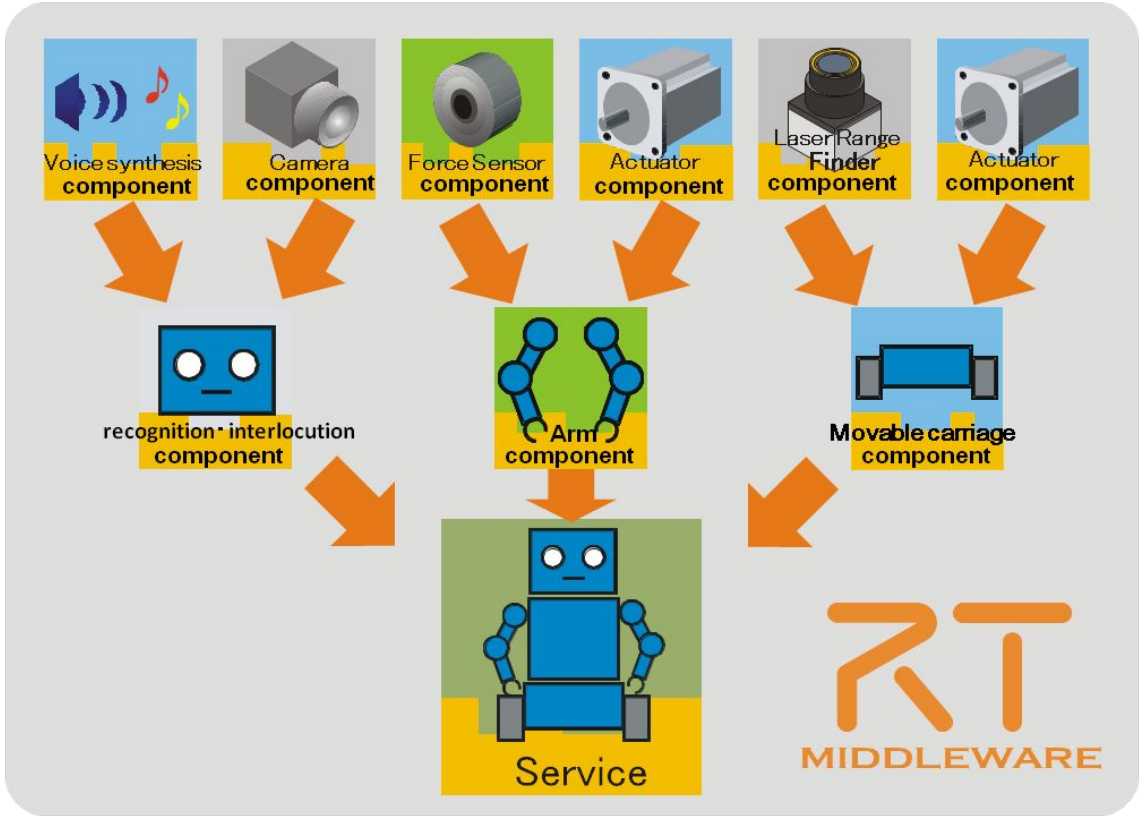
"A component can find the other components through the IceGrid registry and can communicate with them through the IceStorm service."

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

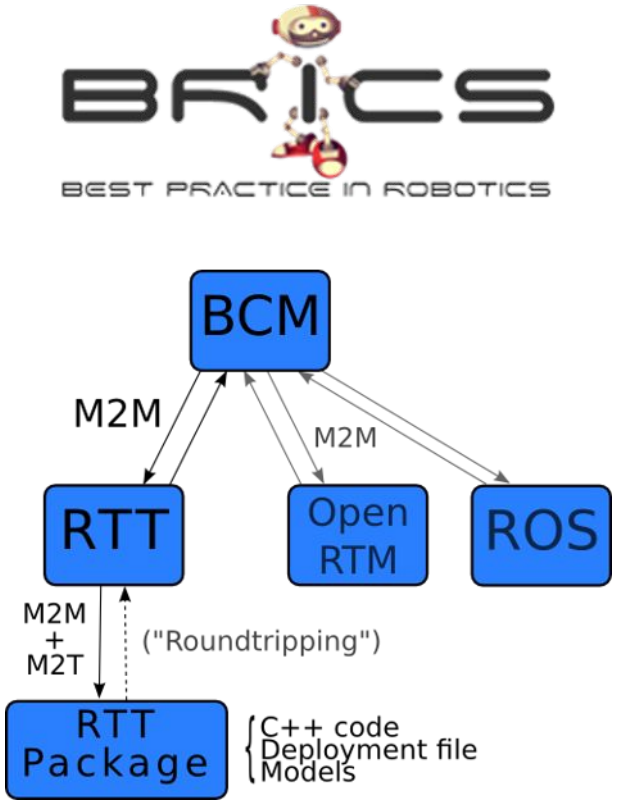
OpenRTM-aist (Advanced Industrial Science & Technology) is based on the CORBA technology to implement RTC extended specification



BRICS: BEST PRACTICES IN ROBOTICS

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

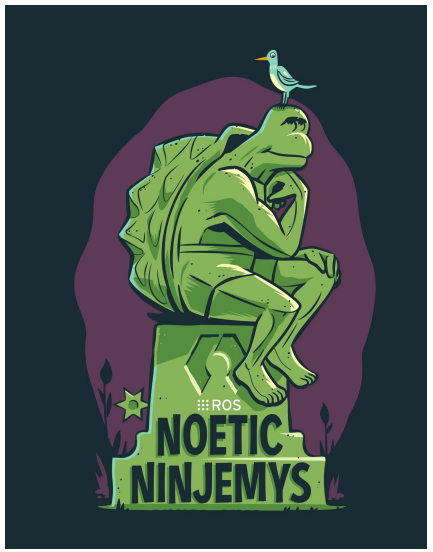
- Investigate the weakness of robotic projects
- Investigates the integration between hardware & software
- Promote model driven engineering in robot development
- Design an Integrated Development Environment for robotic projects (BRIDE)
- Define showcases for the evaluation of project robustness with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

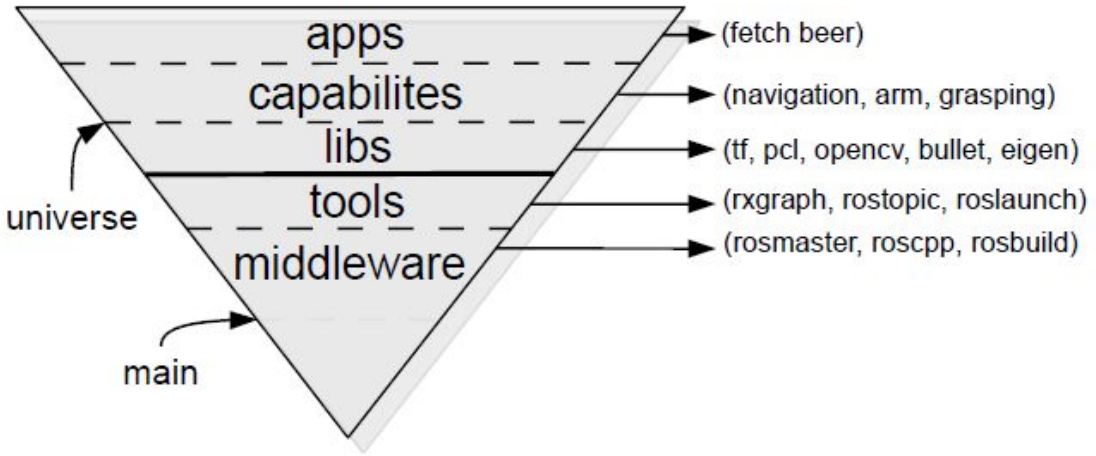


"The prime objective of BRICS is to structure and formalize the robot development process itself and to provide tools, models, and functional libraries, which help accelerating this process significantly."

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?

wiki.ros.org

Traffic volume by country

	Country	Users	New users
		178,380 62.48% of total	102,471 48.37% of total
1	China	29,232	17,705
2	United States	29,045	15,770
3	Japan	14,241	7,171
4	India	11,916	6,782
5	Germany	11,702	6,169
6	South Korea	10,909	5,870
7	United Kingdom	4,953	2,869
8	Hong Kong	4,591	2,043
9	Taiwan	4,237	2,222
10	Singapore	4,048	1,953
11	Italy	3,999	2,184
12	Russia	3,934	2,358
13	France	3,843	2,043
14	Canada	3,746	2,069
15	Türkiye	3,473	2,107
16	Spain	3,075	1,517
17	Australia	2,480	1,240
18	Vietnam	2,353	1,311
19	Brazil	2,055	1,052
20	Netherlands	1,994	994

ROS has grown to include a large community of users worldwide

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

https://discourse.ros.org/uploads/short-url/gXltQJOg3jBEzsJWcGaQ2G8YpNw.pdf

ROS™

A LOT OF RESOURCES

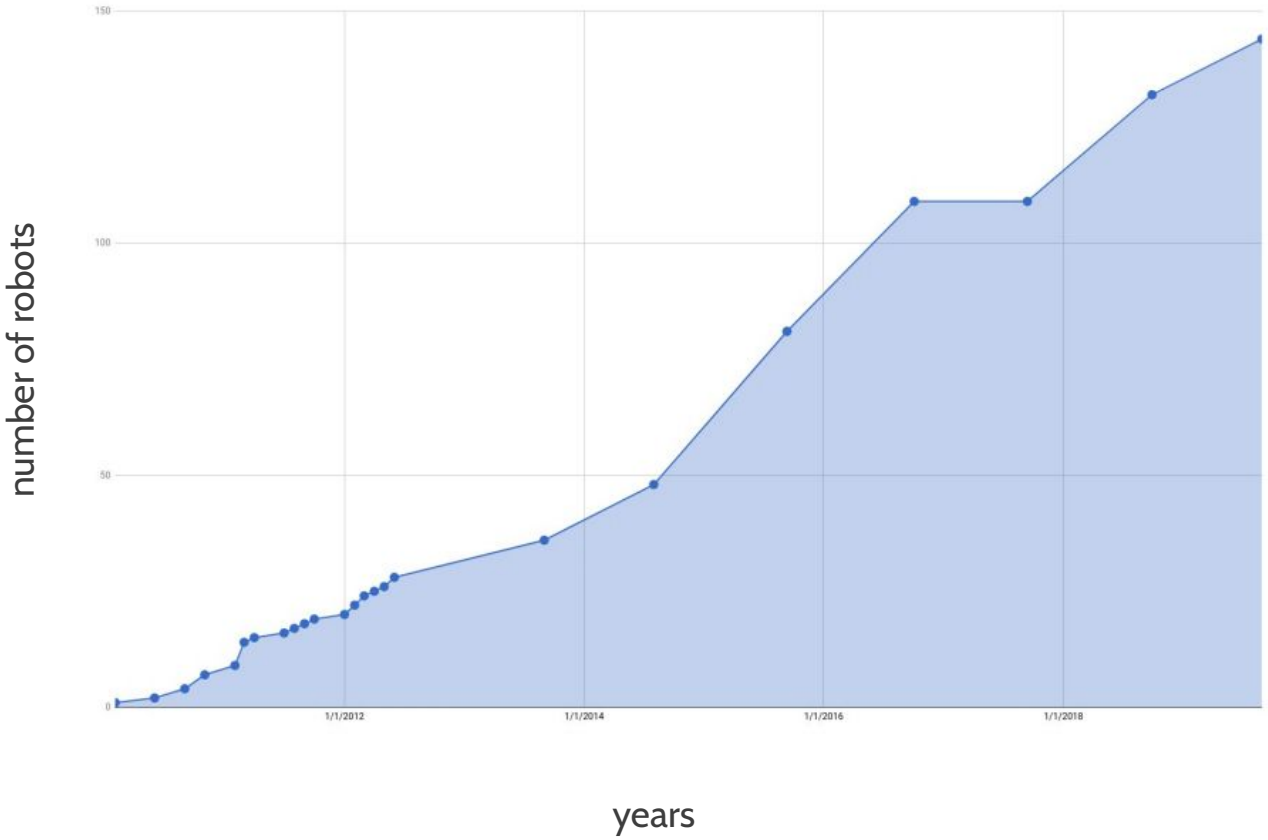
ROS Wiki

- 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

ROBOT AND RESEARCH



Total number of papers citing *ROS: an open-source Robot Operating System* (Quigley et al., 2009)
12115(scholar)

ROS Industrial

ROS Industrial

Open Source Robotics Foundation (OSRF)

Open Source Robotics Corporation (OSRC)

ROS INTRODUCTION

ROBOTICS

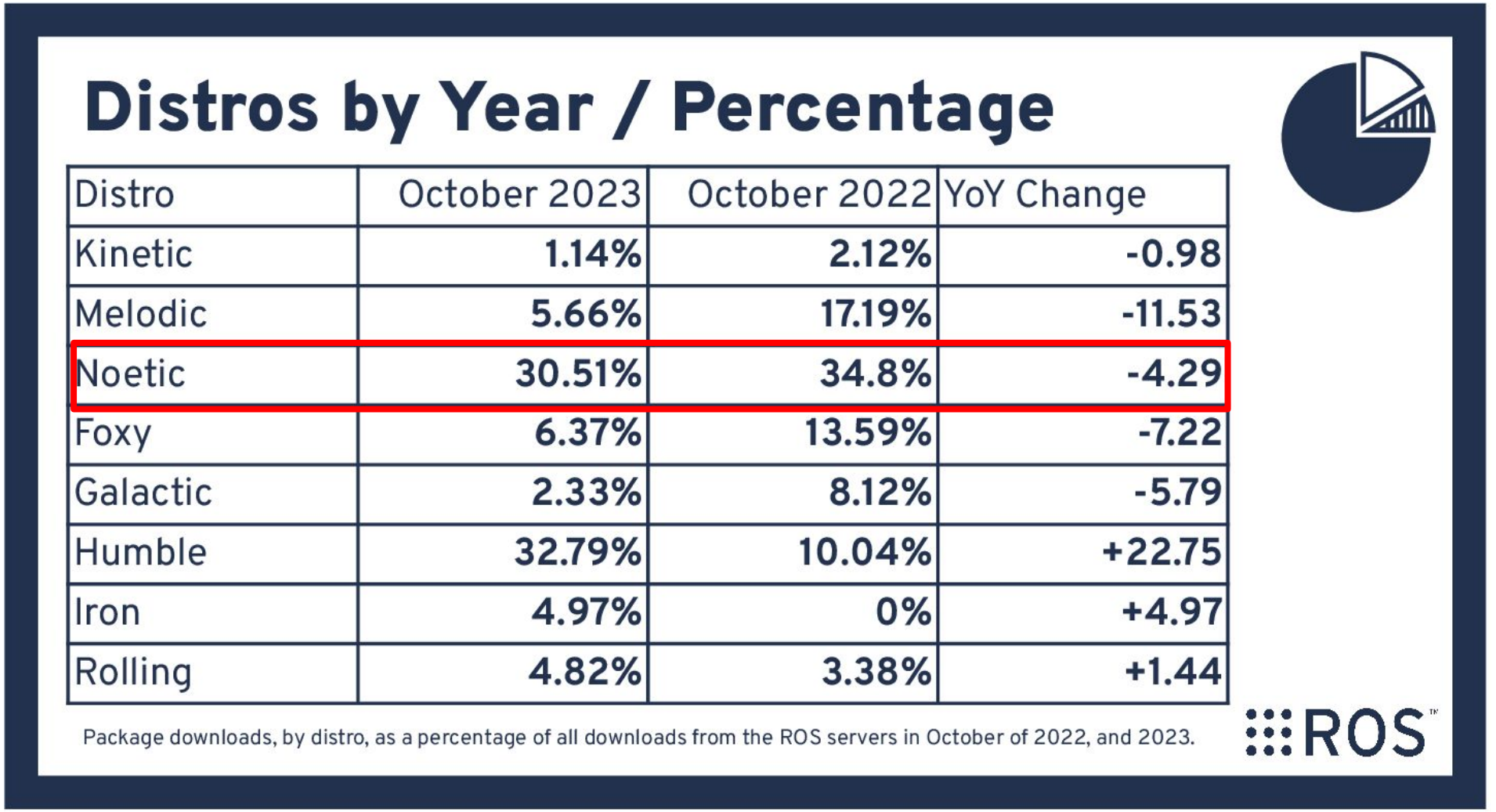
POLITECNICO
MILANO 1863

Lat year...

Statistics

Distro	10/2021	10/2022	YoY Change
ROS 1 Kinetic	4.74%	2.62%	-2.62%
ROS 1 Melodic	36.64%	17.19%	-19.45%
ROS 1 Noetic	24.35%	34.80%	+10.45%
ROS 2 Humble	NA	10.04%	NA
ROS 2 Galactic	7.63%	8.12%	+0.49%
ROS 2 Foxy	14.04%	13.59%	-0.45%
ROS 2 Rolling	1.84%	3.38%	+1.54%
All other distros	2.24%	1.43%	-0.81%

Statistics



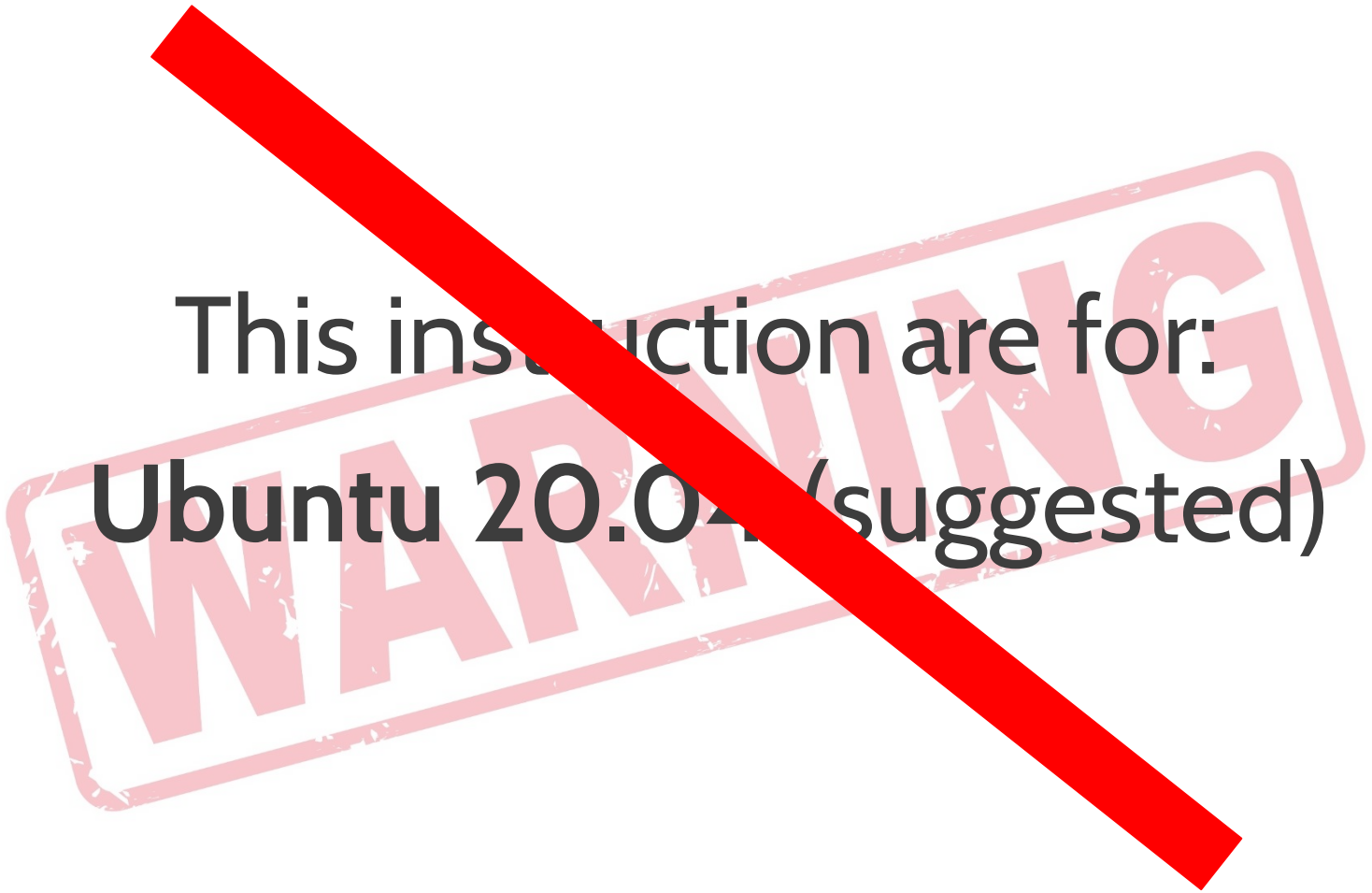
Distros by Year / Percentage

Distro	October 2023	October 2022	YoY Change
Kinetic	1.14%	2.12%	-0.98
Melodic	5.66%	17.19%	-11.53
Noetic	30.51%	34.8%	-4.29
Foxy	6.37%	13.59%	-7.22
Galactic	2.33%	8.12%	-5.79
Humble	32.79%	10.04%	+22.75
Iron	4.97%	0%	+4.97
Rolling	4.82%	3.38%	+1.44

Package downloads, by distro, as a percentage of all downloads from the ROS servers in October of 2022, and 2023.

ROS™

INSTALLATION



This instruction are for:

Ubuntu 20.04 (suggested)

INSTALLATION

- We use docker

- Should work on any OS
- We all use the same docker image
- No dependencies issues
- ROS Image is not small, download it before next lecture

INSTALLATION (Docker)

First install docker:

- https://docs.docker.com/engine/install/

- Check docker is running
- `docker run hello-world`

- If you get permission error:
- `sudo groupadd docker`
- `sudo usermod -aG docker $USER`
- reboot

INSTALLATION (Docker)

Now that you have installed docker, build the robotics image:

- Build an image using a Dockerfile (provided, it should have most of the required packages, run the command in the Dockerfile folder):
- For easy use I provide directly a script which simply run the docker build command
  - `./build.sh`

- This might not work on some os, you can manually run the command:

```
docker build --build-arg USER_ID=$(id -u) --build-arg GROUP_ID=$(id -g) --build-arg
USER_NAME=$(whoami) -t robotics:stable .
```

- On windows most of the command does not make sense, simply run

```
docker build --build-arg USER_NAME="here put your username" -t ros-custom .
```

The dot at the end of the command is important!

INSTALLATION (Docker, how to get a gui?)

Docker usually works without gui, you do most of your work from terminal

In robotics you might need a gui (data visualization, debugging, etc.):

http://wiki.ros.org/docker/Tutorials/GUI

On unix you can use the X server and allow execution of the gui outside docker:

If you need a fast solution: `xhost +local:root`

Then remember to: `xhost -local:root`

On windows you can run a container with a virtual desktop and connect to it (the noVNC approach).

Remember to create the ros network: `docker -> network create ros`

Why are we using docker?

Any OS solution

This year the project will be “automatically” evaluated

Everyone will have the same image/libraries/dependencies, no OS differences

So this year no python

Standard evaluation metrics will be used to evaluate the project

What about IDEs? What is a terminal?

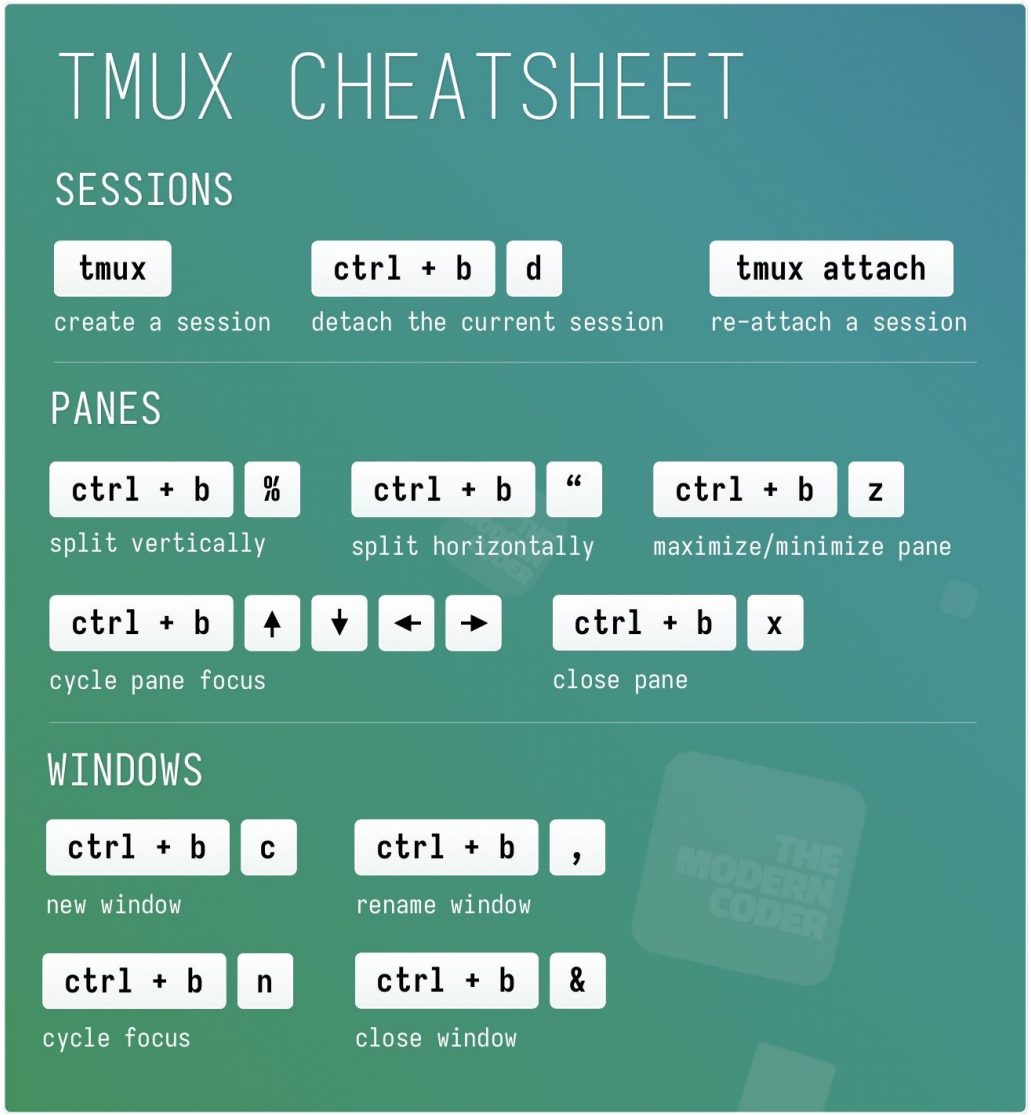
What I will use:
- nano/ gedit
- tmux

You will need to know at least the basic navigation command from terminal

tmux: terminal multiplexer, keeps programs running when you close a terminal, allow creation of multiple terminals. Shortcuts in the next slide

I will not use an IDE, but all mayor IDEs have docker integration

Tmux basic commands

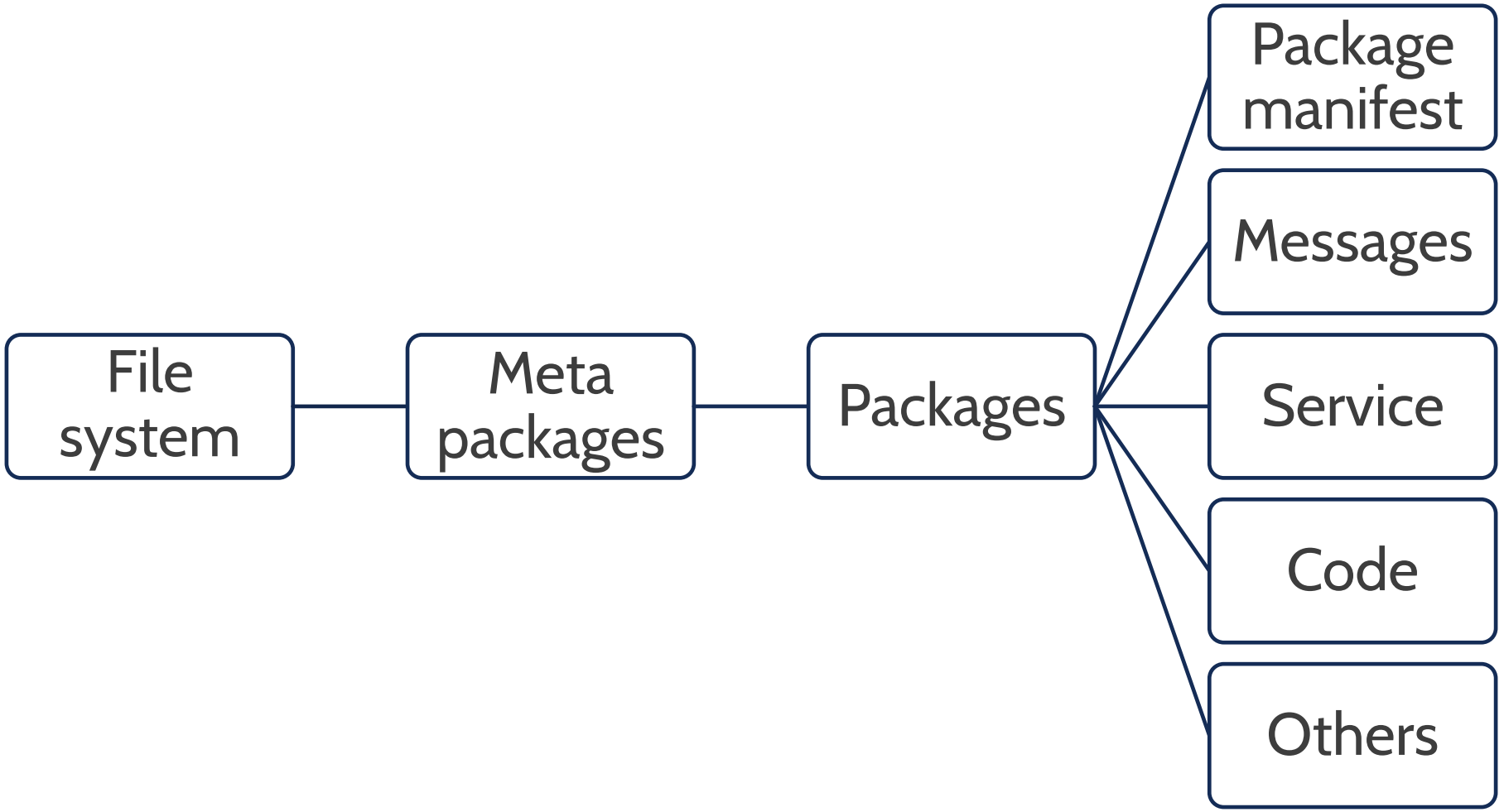


ROS FILESYSTEM

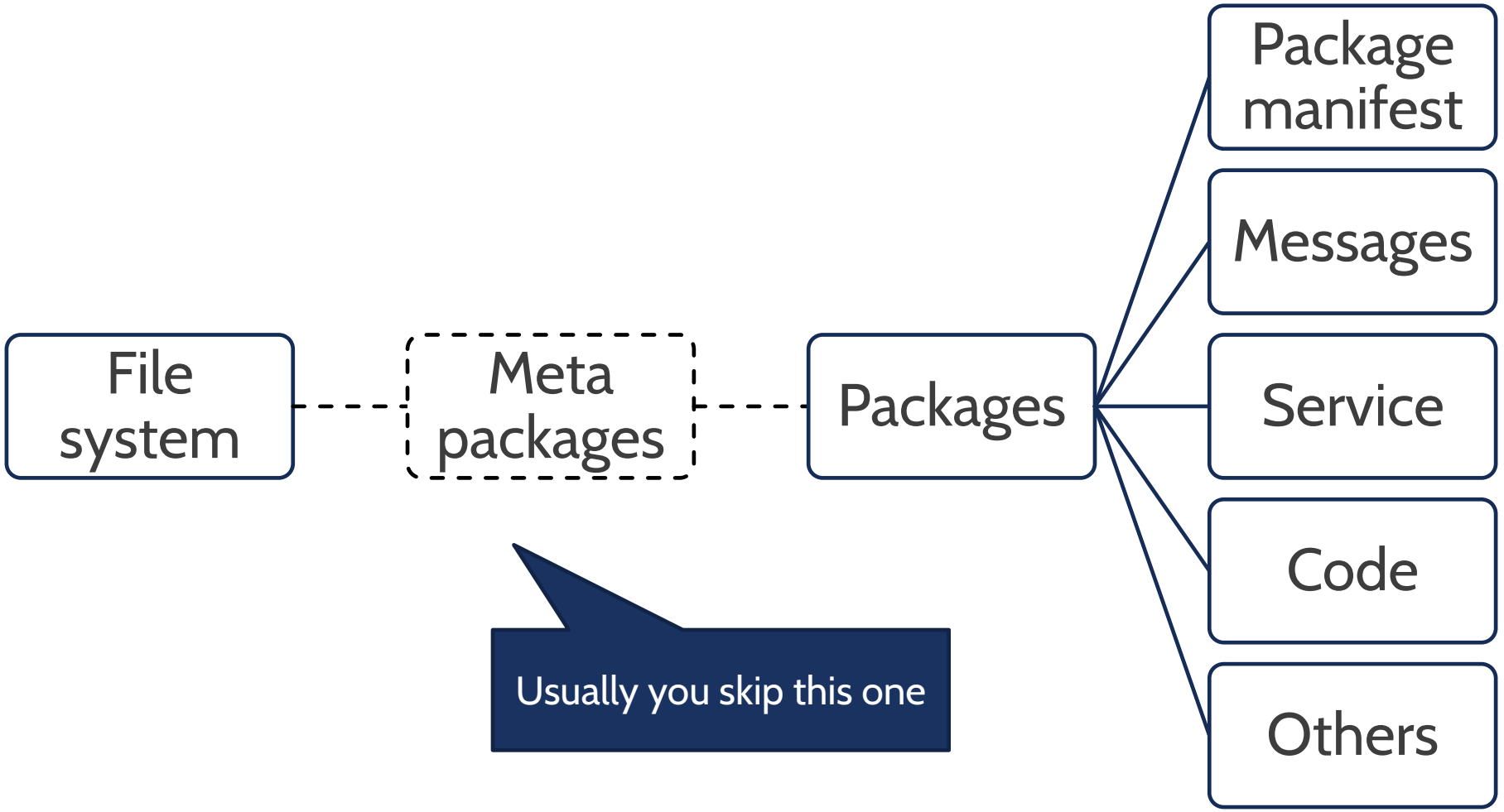
ROBOTICS

POLITECNICO
MILANO 1863

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

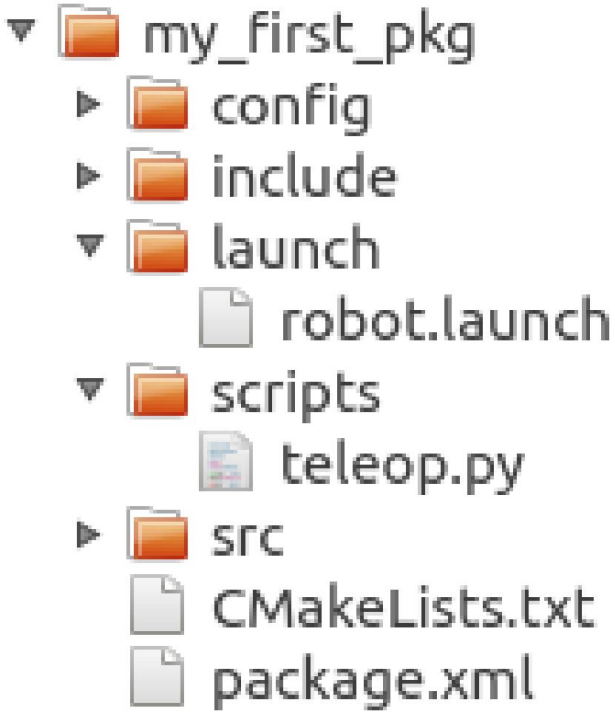
STRUCTURE OF A PACKAGE

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



ROS STRUCTURE

ROBOTICS

POLITECNICO
MILANO 1863

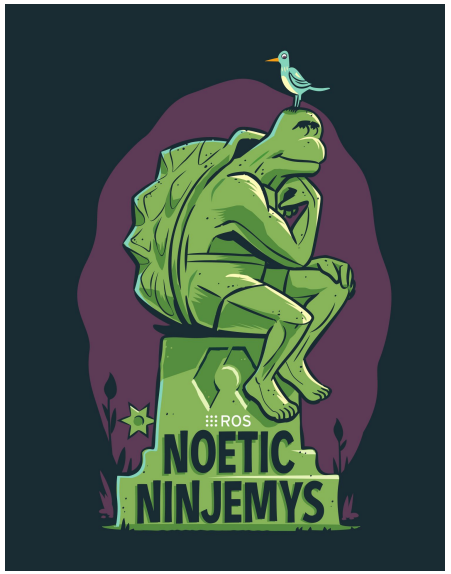
ROS: ROBOT OPERATING SYSTEM

ROS main features:

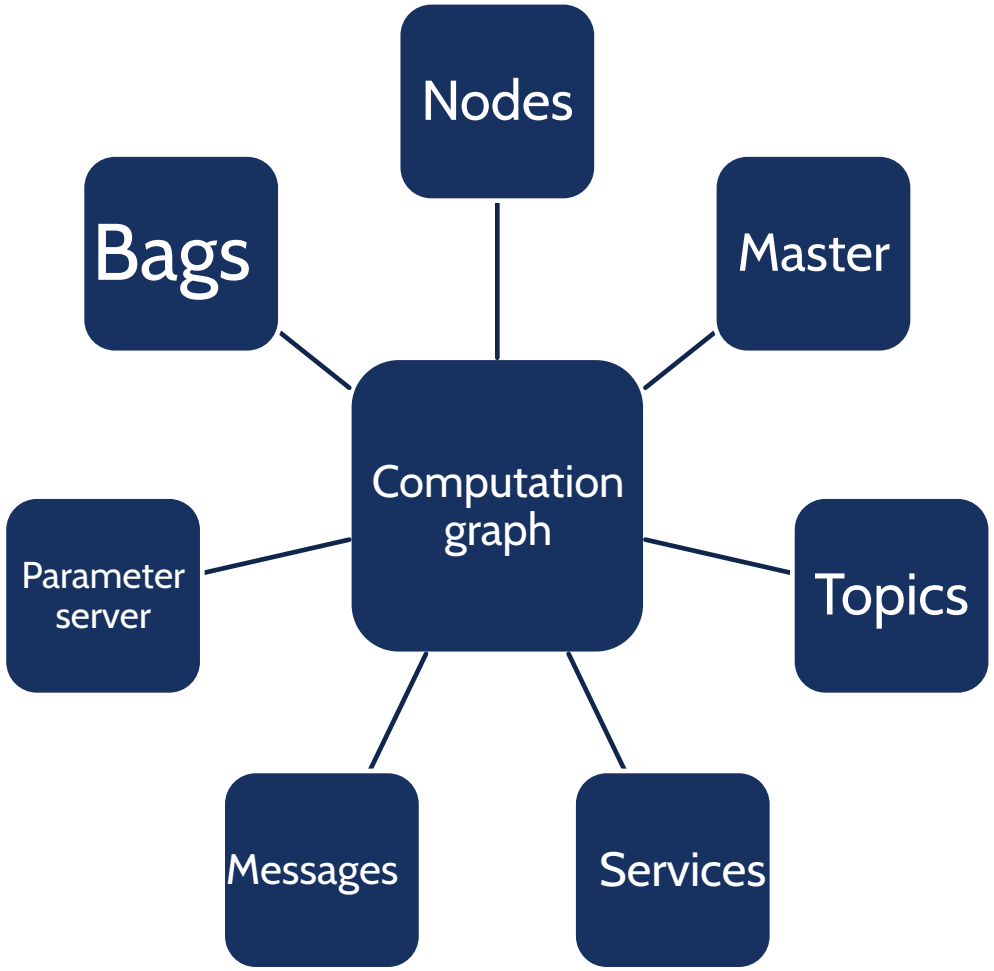
- Distributed framework
- Reuse code
- Language independent
- Easy testing on Real Robot & Simulation
- Scaling

ROS Components

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



ROS 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

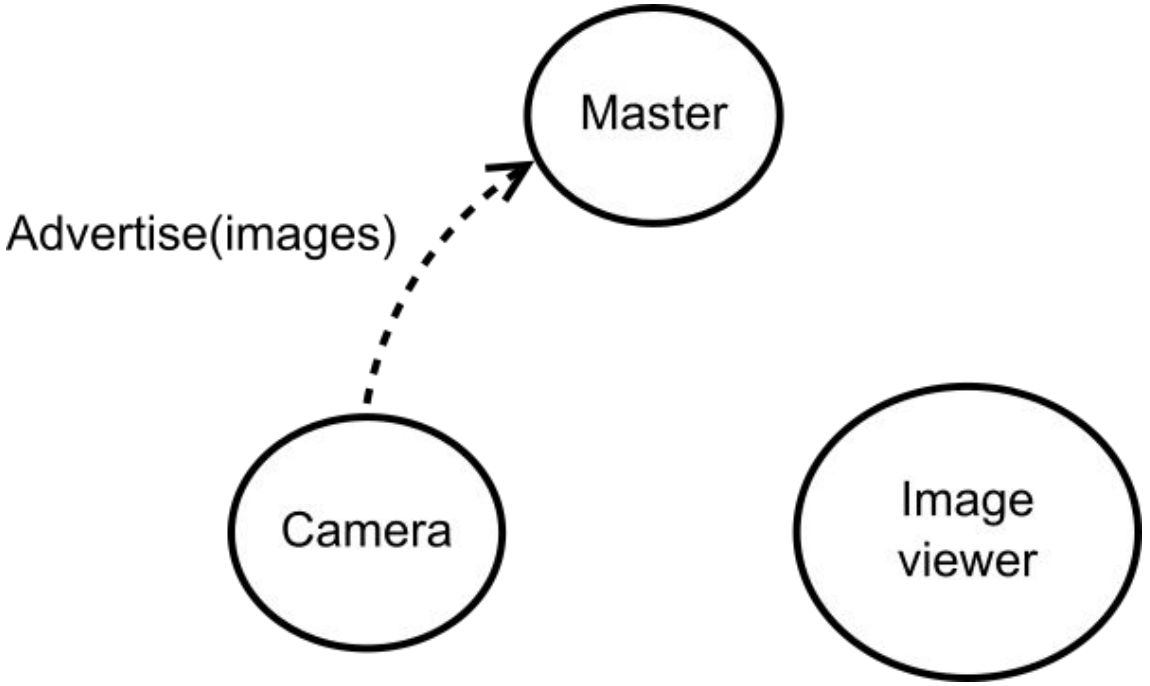
Meant to operate at fine-grained scale

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

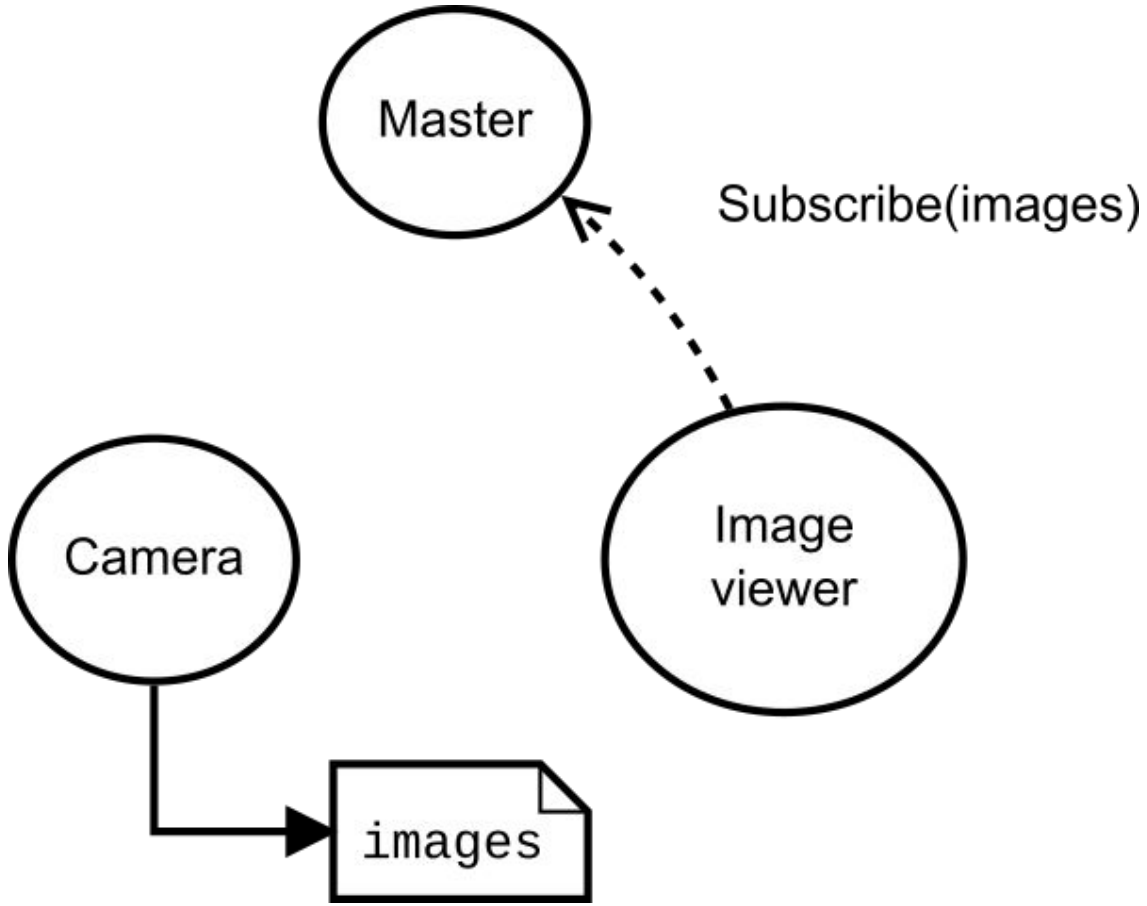
Provides naming and registration services

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

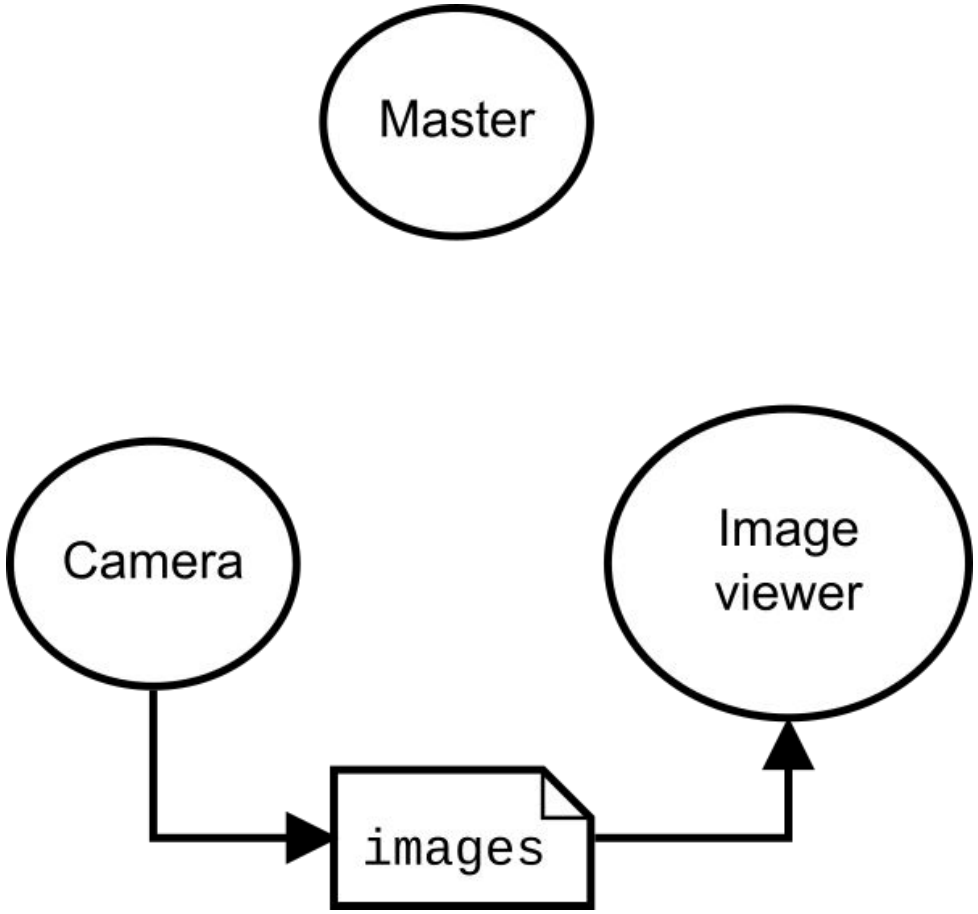
Provides naming and registration services

Essential for nodes interactions

One master for each system, even on distributed architectures

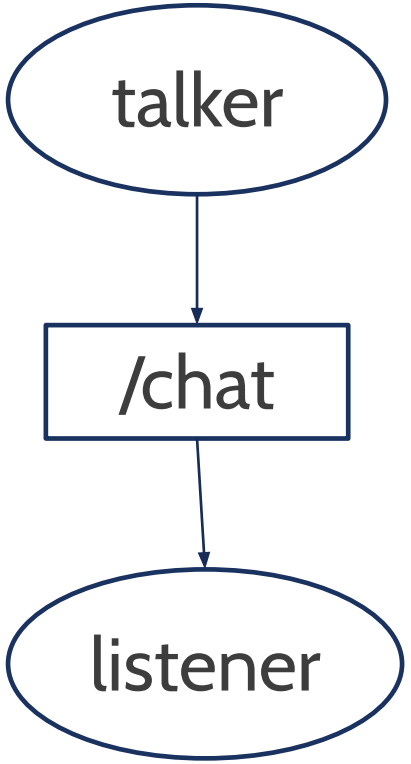
Enables individual ROS nodes to locate one another

One of the functionalities provided by roscore



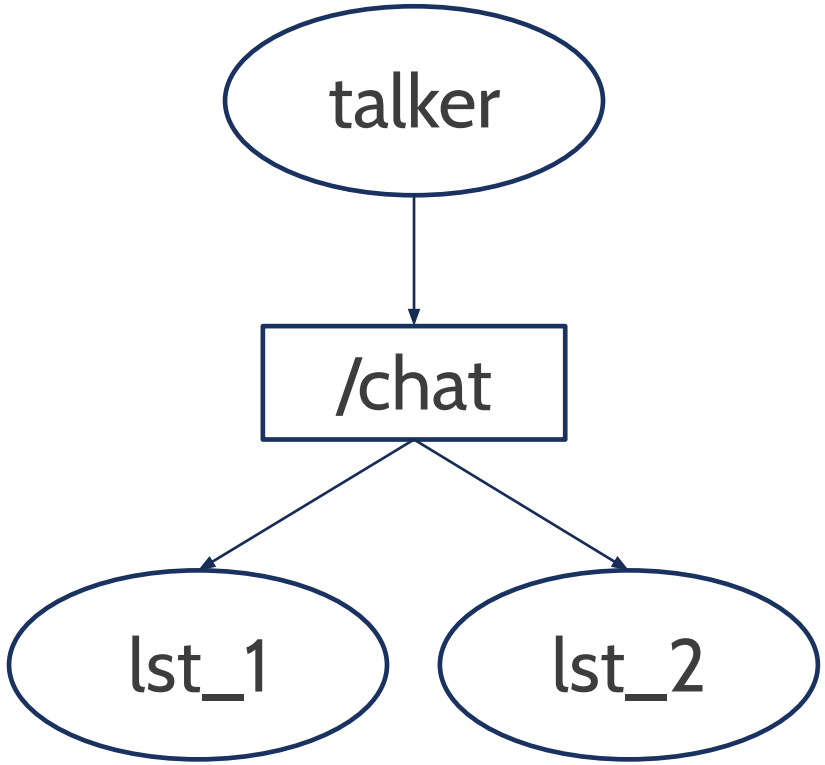
TOPICS

- 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



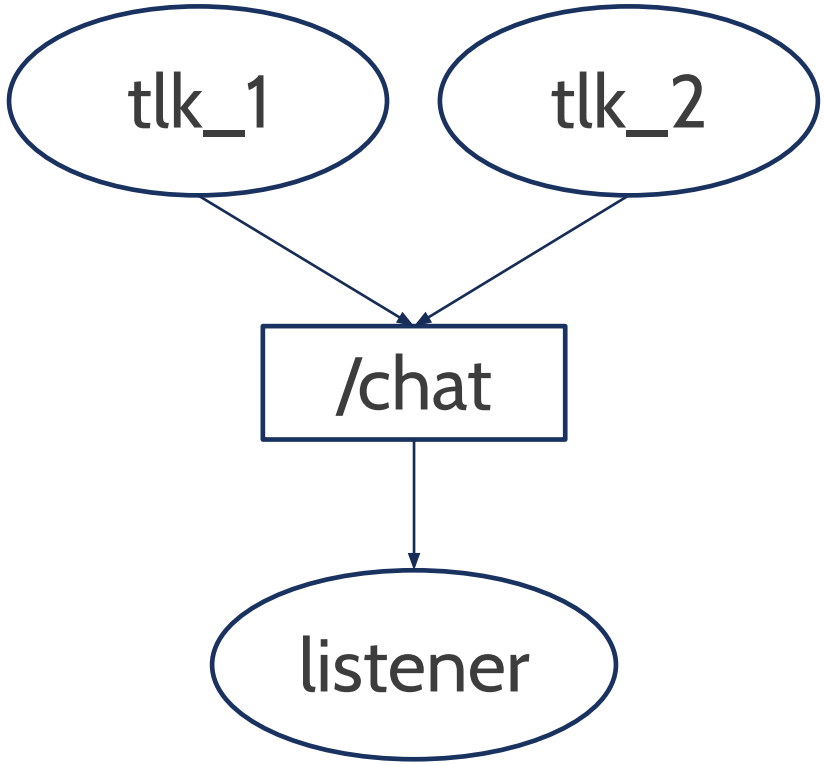
TOPICS

- 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

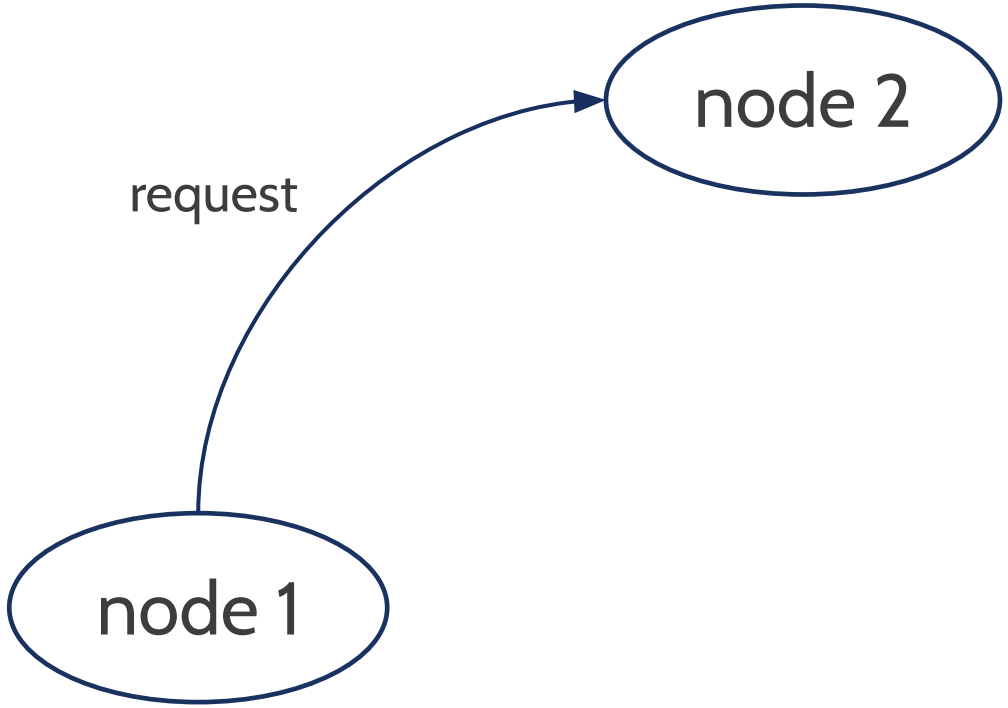
SERVICES

Work like remote function calls

Implement the client/server paradigm

Code waits for service call to complete

Guarantee of execution



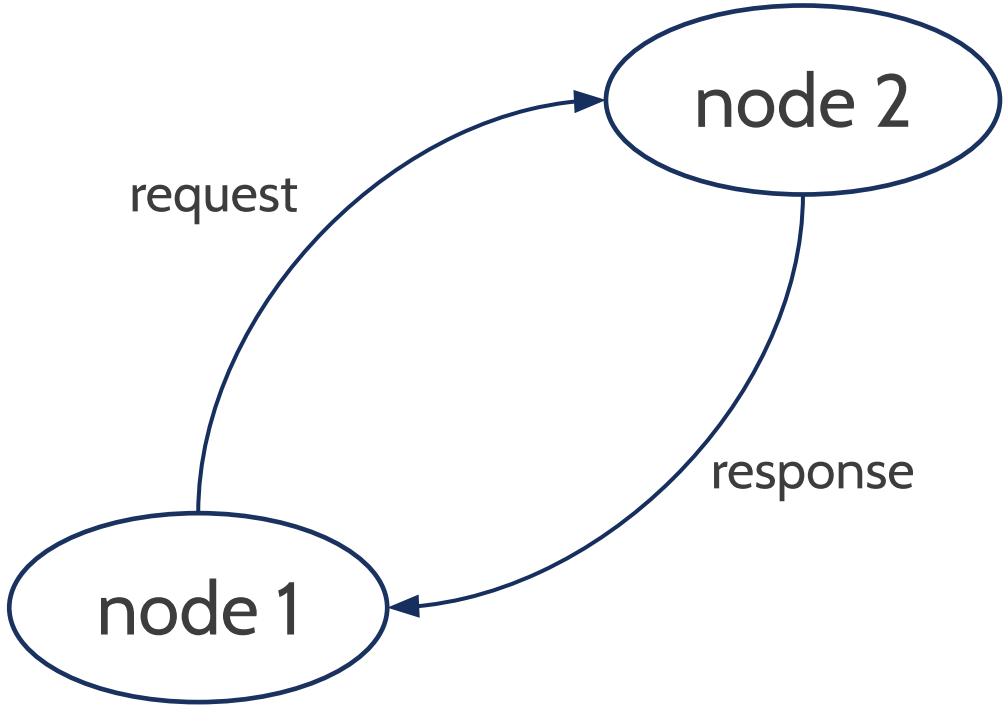
SERVICES

Work like remote function calls

Implement the client/server paradigm

Code waits for service call to complete

Guarantee of execution



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
```

request

response

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

name	value
/gains/P	10.0
/gains/I	1.0
/gains/D	0.1
use_sim_time	True

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