ROS: Robot Operating System

Michael Bosello

michael.bosello@unibo.it

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Smart Vehicular Systems A.Y. 2021/2022

Acknowledgements

Thanks to Prof. Madhur Behl from University of Virginia

- for some of the slides and illustrations

https://linklab-uva.github.io/autonomousracing/

ROS

Robot Operating System

‘Is a set of software libraries and tools
that help you build robot applications.’

Architecture for process communication
+
Suite of development tools
+
Libraries

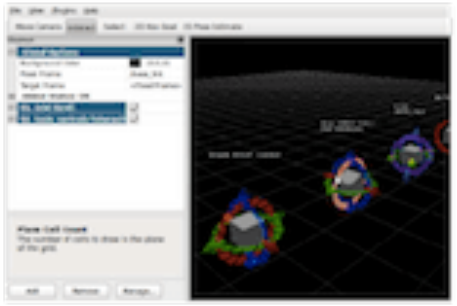
ROS Features

- Modular
  - Programs are peer-to-peer
  - They communicate over defined API
- Ditributed
  - Modules can run on different devices (on-board or over the network)
- Multi-language
  - Modules can be written in any supported language
  - C++, Python, Javascript, Java, MATLAB, LISP
  - Transparent to the user
- Light-weight
- Open-source

- Widely used in both open/commercial robot and in the industry

Plumbing

- Process Management
- Inter-process communication
- Device drivers



Tools

- Simulation
- Visualization
- GUI
- Data logging

Capabilities

- Control
- Planning
- Perception
- Mapping
- Manipulation

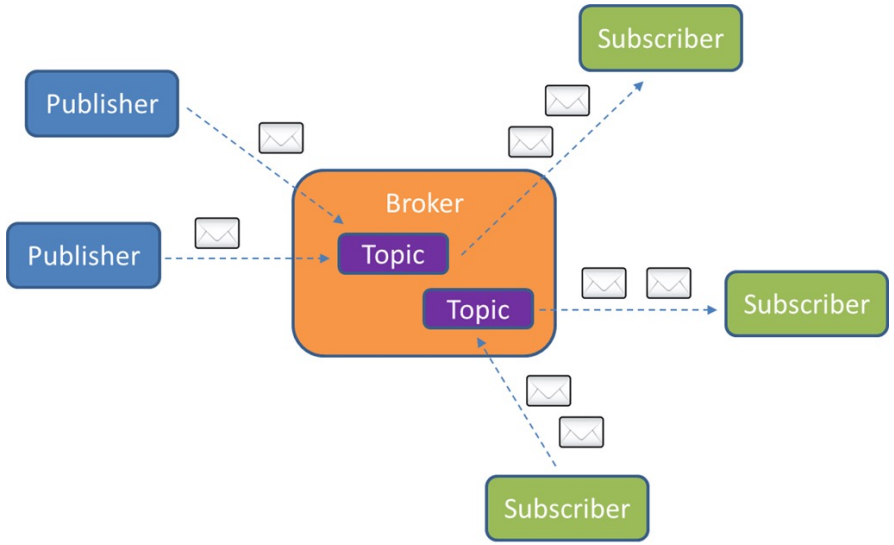
Ecosystem

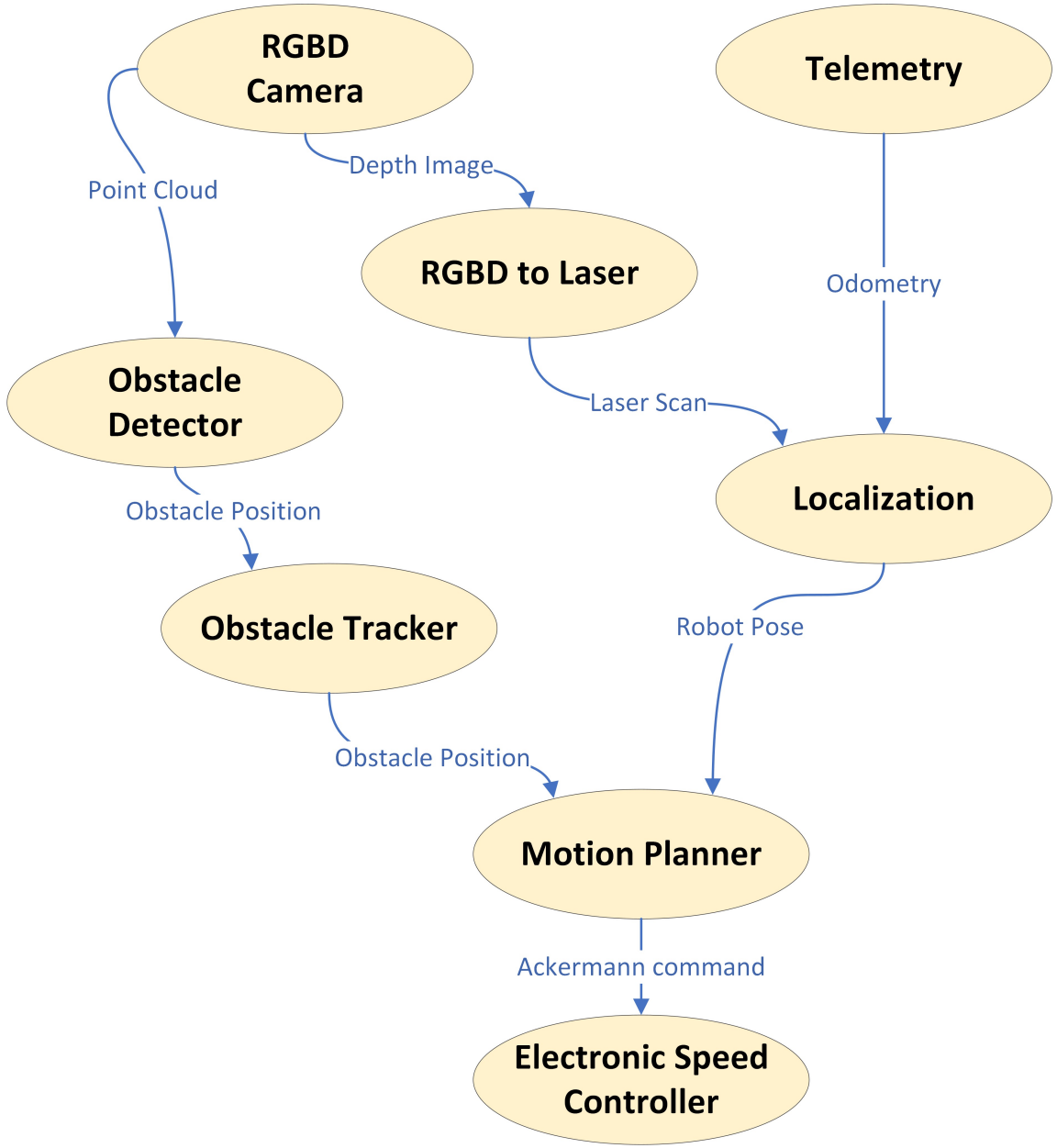
- Package organization
- Software distribution
- Documentation
- Tutorials

Plumbing

- Modular
- Parallel execution
  - Each node executed in its own process

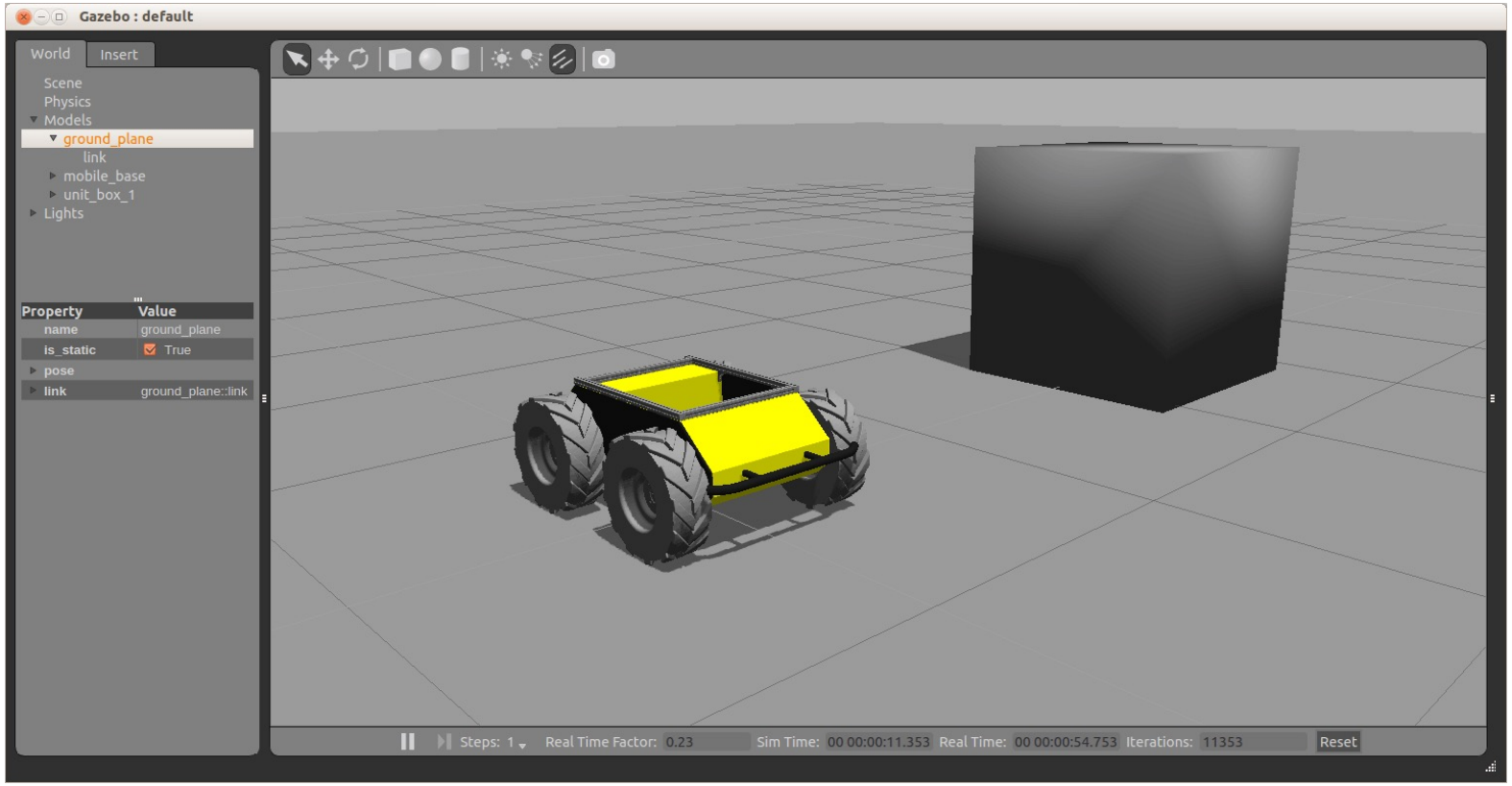
- Publish-subscribe



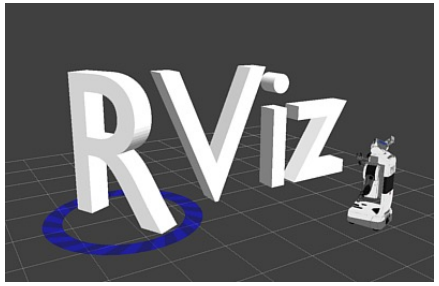


Tools: GAZEBO

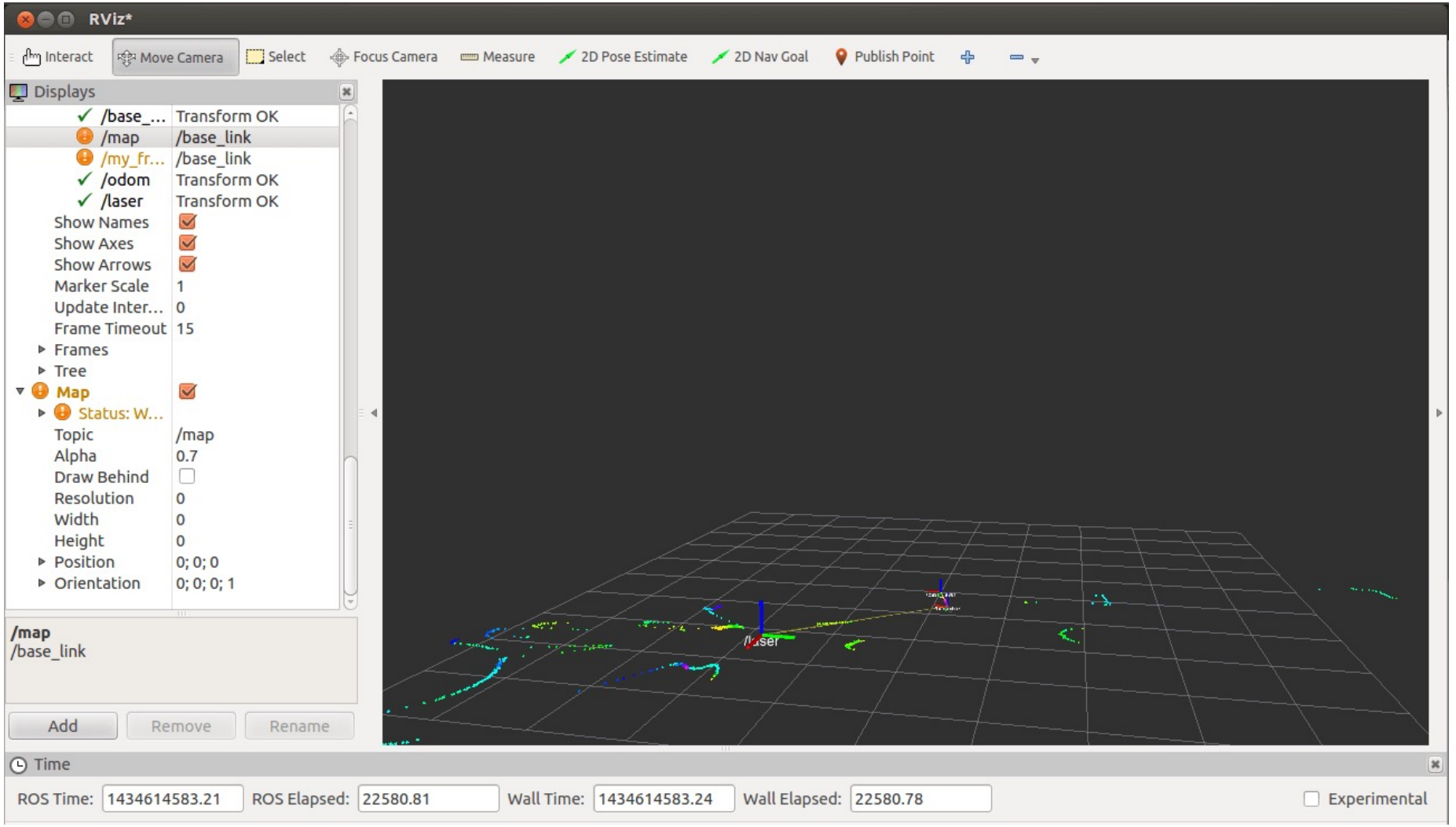
- Simulate 3d rigid-body dynamics
- Simulate sensors
- 3D GUI
- Database of robots/envs



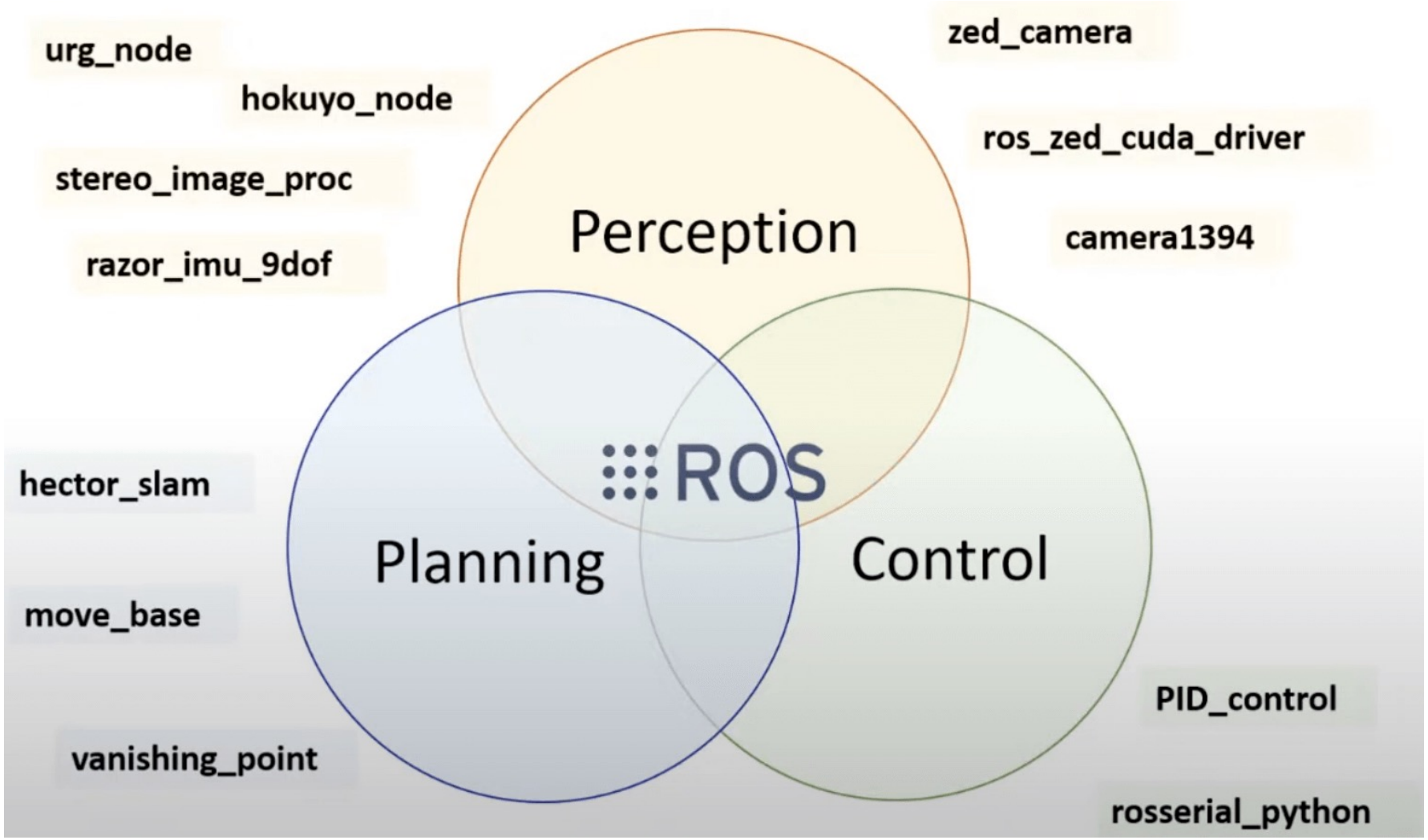
Tools:




```
$ rosrun rviz rviz
```

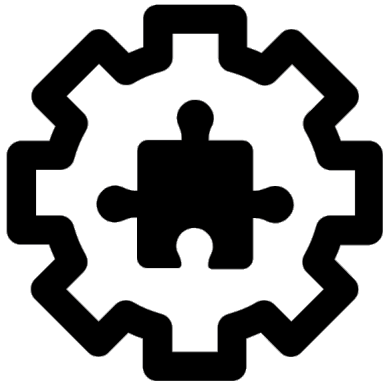


Capabilities



ROS Components

Communication architecture



Master

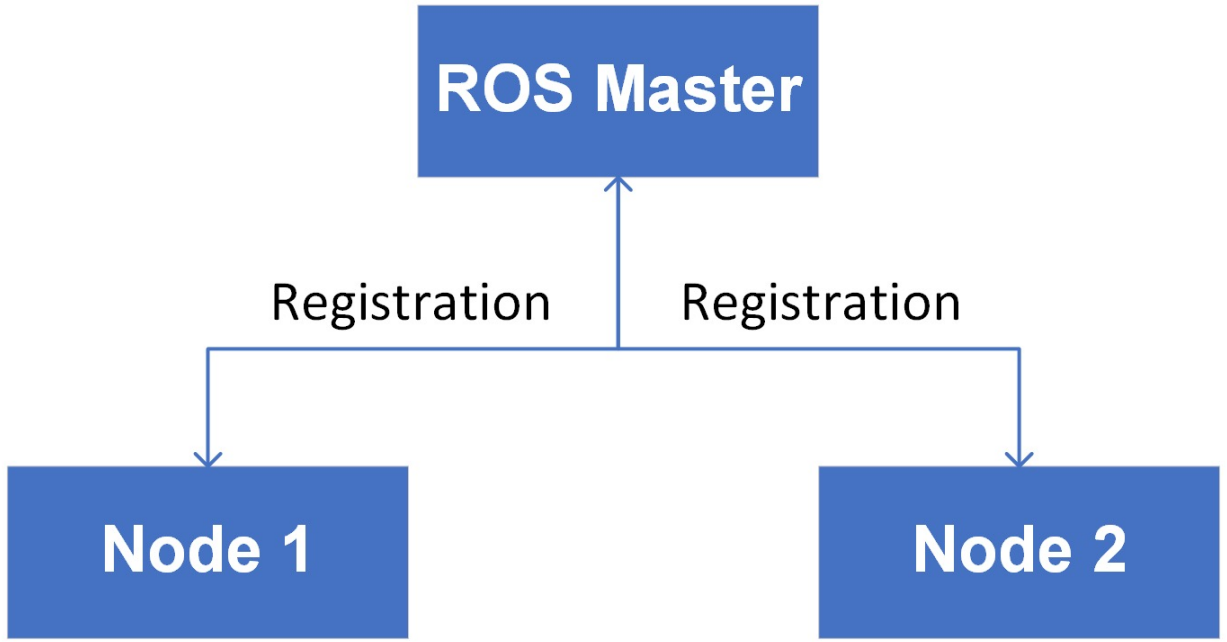
- Manages connection between nodes
- Enables nodes to locate one another

Start Master (and setup the ROS env)

```
$ roscore
```

Node

- Executable program
- **Individually managed/executed**
- Organized in *packages*
- Every node registers with the master at startup



Run a node

```
$ rosrun package_name node_name
```

See active nodes

```
$ rosnode list
```

```
$ rosnode info node_name
$ rosnode kill node_name
$ rosnode ping node_name
```

Topics

- Nodes communicates over topics
- Topics are channels
  - for data streaming
  - Tipically **one-to-many**

List active topics

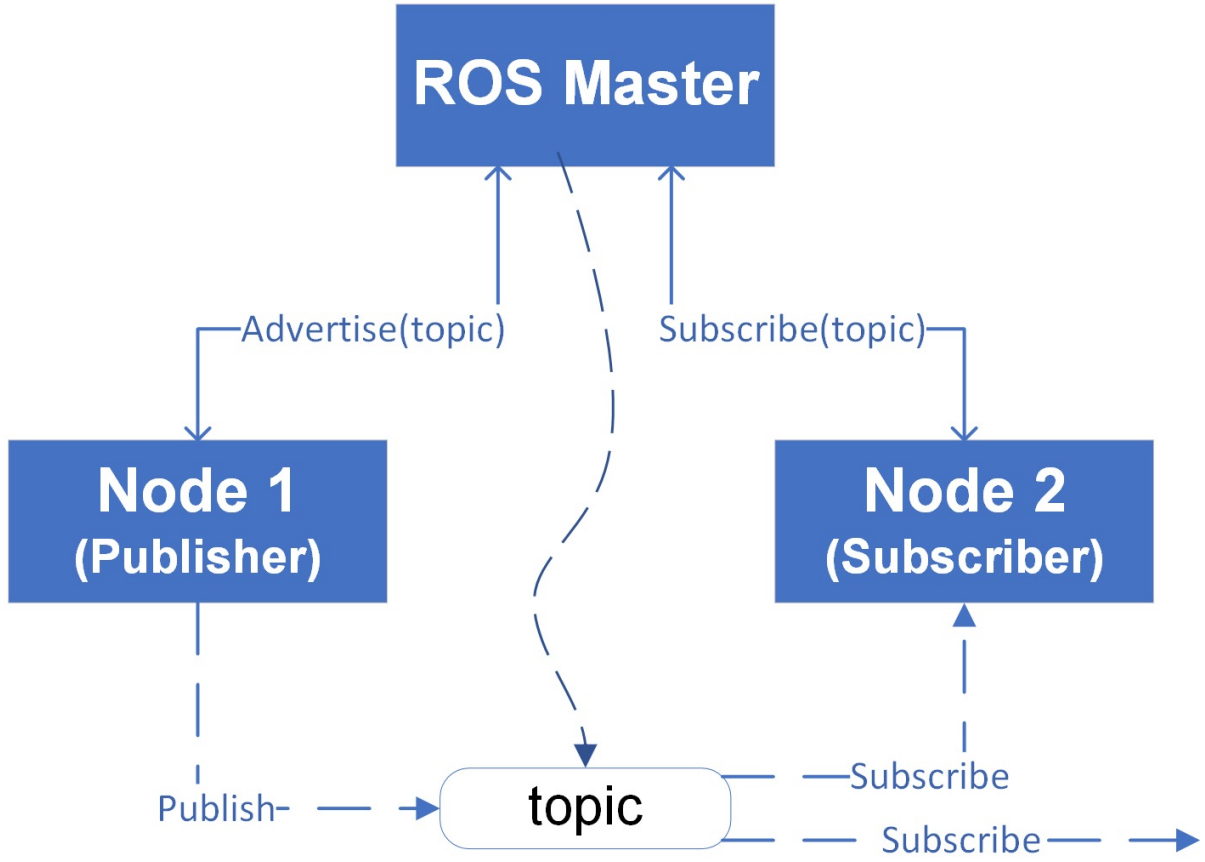
```
$ rostopic list
```

Subscribe and print messages

```
$ rostopic echo /topic
```

Topic info

```
$ rostopic info /topic
```



When a node subscribe to an existing topic, a channel between the publisher and the subscriber is opened

Messages

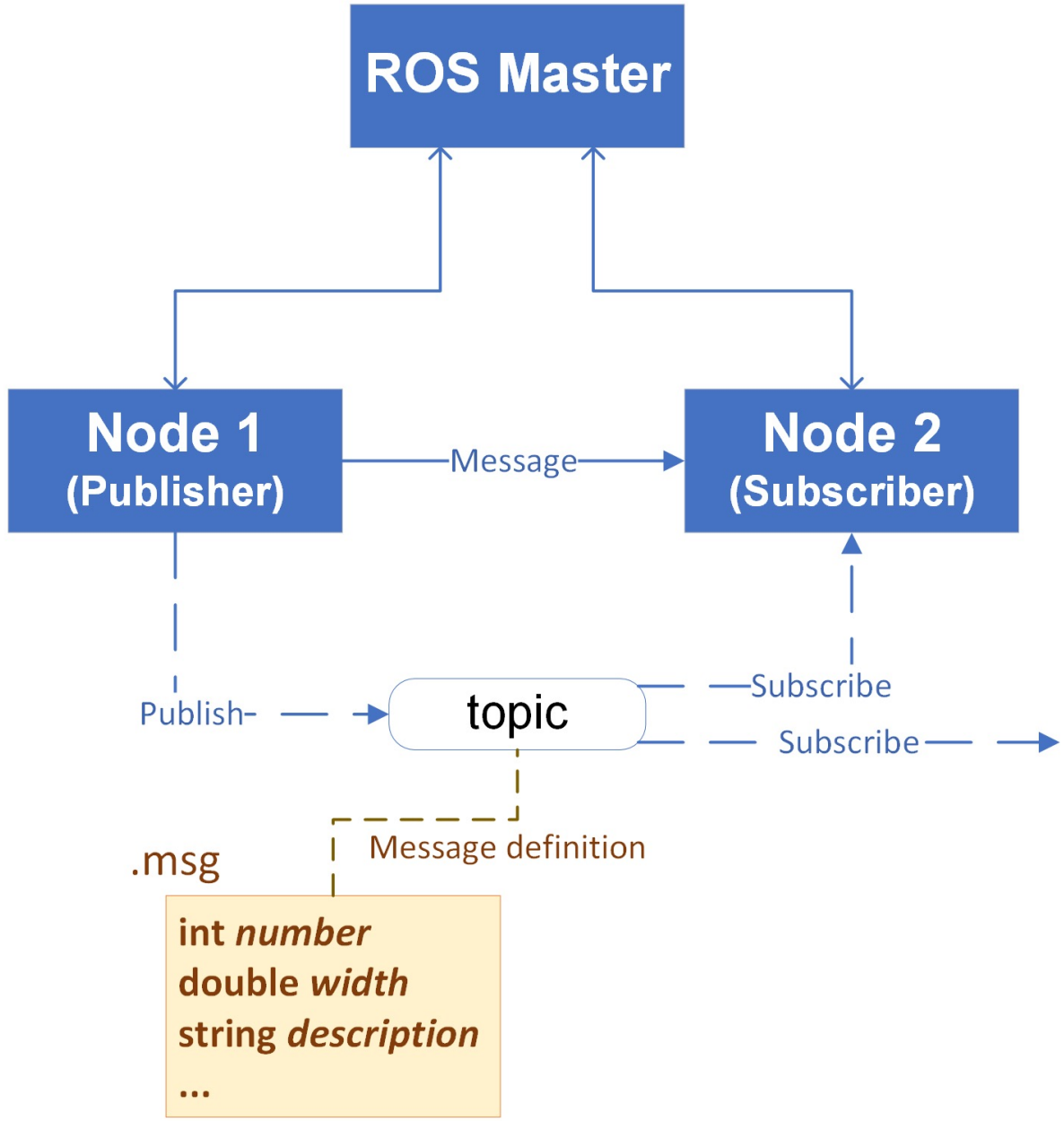
- Strongly-typed **data structure**
- Define the type of a topic
- Defined in .msg files

See the type of a topic

```
$ rostopic type /topic
```

Publish a message to a topic

```
$ rostopic pub /topic type args
```



Communication between nodes

Nodes communicate messages via topics

- Many nodes can pub/sub to the same topic
- Communication is direct node-to-node

Messages are **asynchronous**

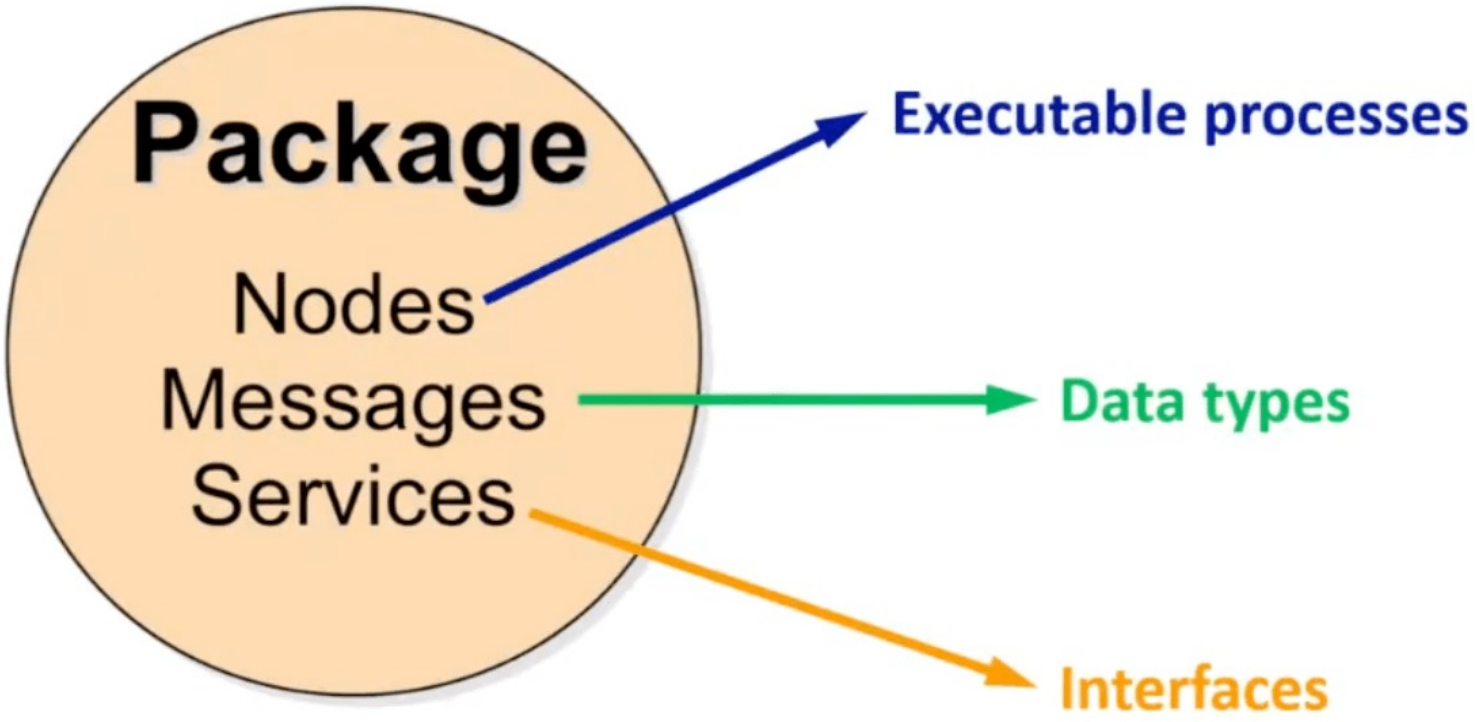
- Publishers don’t know if anyone’s listening
- Messages **may be dropped**
- Subscribers are event-triggered by incoming messages

Topics vs Messages

Topics are **channels**, Messages are **data types**

Different topics can use the same message type

Packages

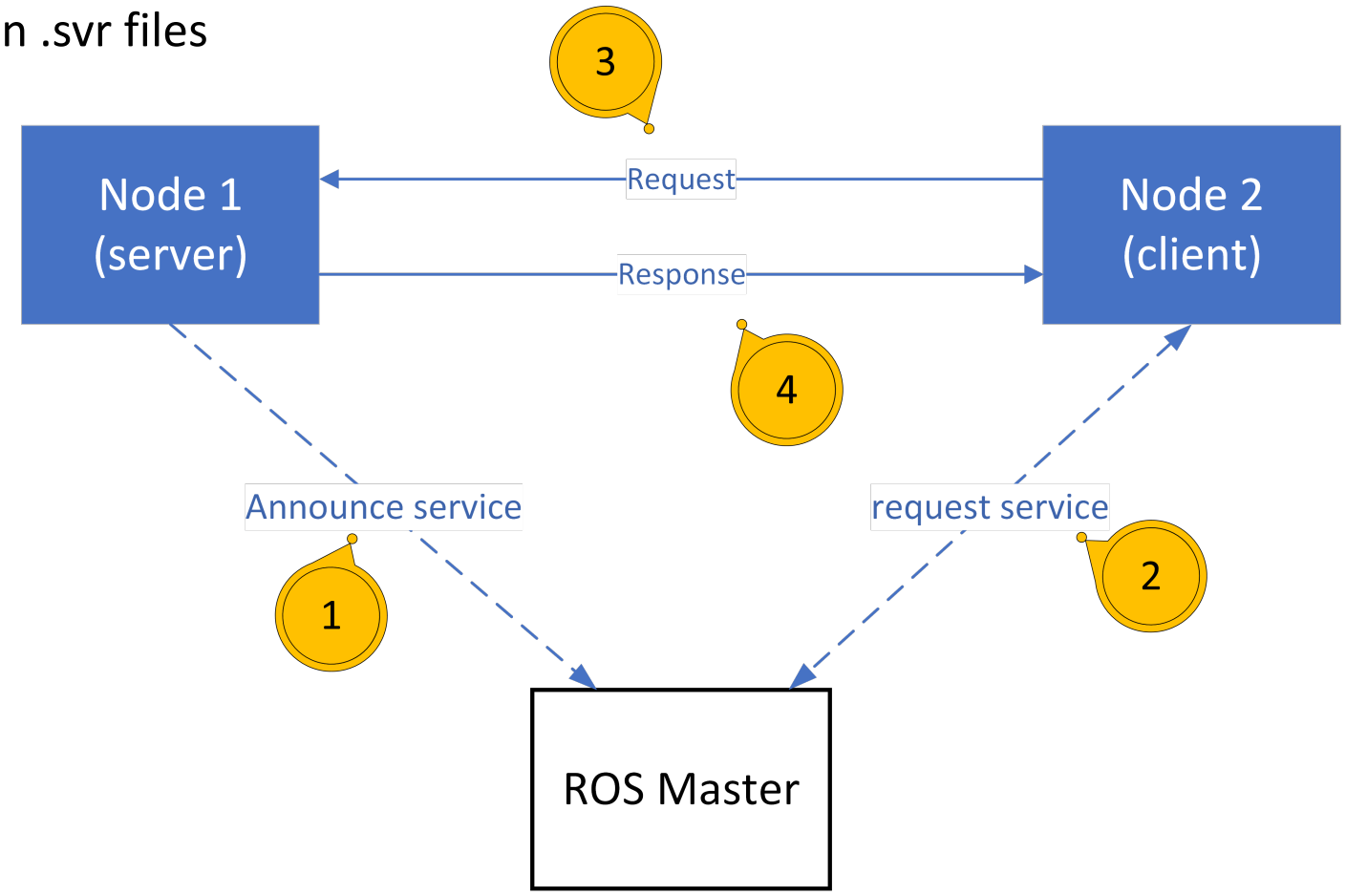


Software in ROS is organized into **packages**

- A package contains one or more nodes

Services

- ROS implementation of request/response
  - Providing node advertise the service by name
  - Requests and responses are defined in .svr files
  - Bi-directional communication
  - **Syncronous**



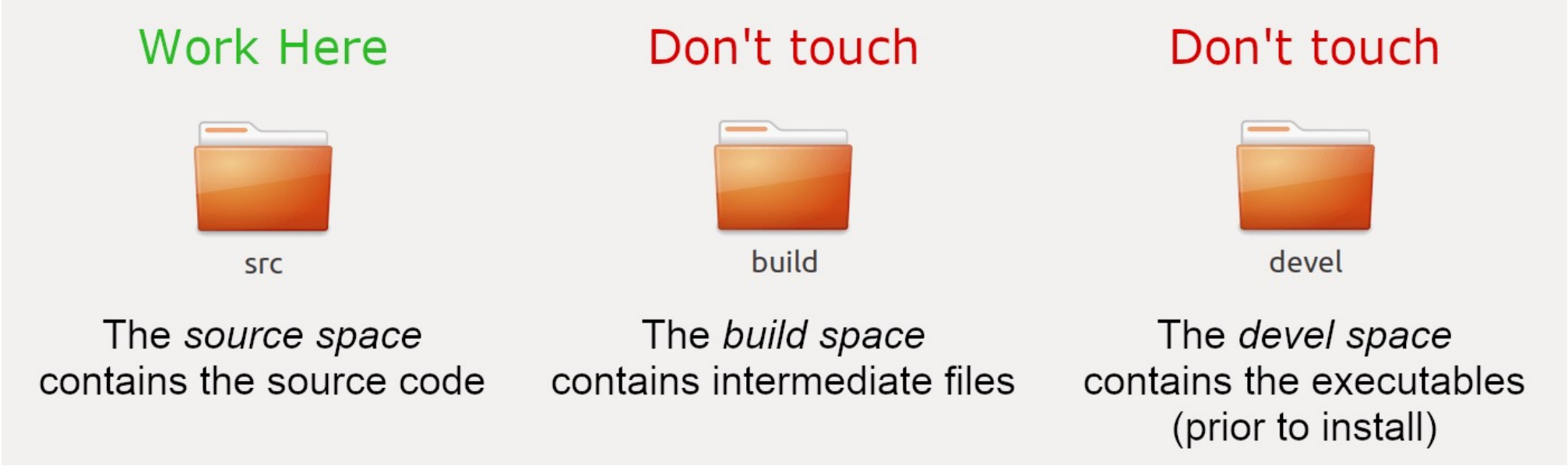
ROS Workspace and Build

Catkin

The ROS build system

- Custom Cmake macros + Python code

Catkin Workspace



Catkin Workspace

- Build multiple packages

Create a workspace

- It will create the CMakeLists

```
$ mkdir ./catkin_ws/src
$ cd ./catkin_ws/src
$ catkin_init_workspace
```

Build the workspace

- It builds all the packages
- Creates build and devel

```
$ cd ../
$ catkin_make
```

Optional install parameter: “catkin_make install”

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

Sourcing

- You need to add the workspace to the ROS env
  - It adds variables to the bash session
  - So ROS can locate the packages

```
$ source catkin_ws/devel/setup.sh
```

- During installation you configured bash to automatically load the ROS installed env at startup

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

Catkin Packages

- A package contains source code, launch files, config files, message definitions, data, documentation
- Dependencies among packages can be declared

A package contains at least:

```
my_package/
  CMakeLists.txt
  package.xml
```

Create a package with dependencies:

- It creates CMakeLists.txt and package.xml

```
$ catkin_create_pkg pkg_name [dependencies]
```

Catkin Packages Files

package.xml

- Contains the **metadata** of a package
  - name, description, version, maintainer(s), license
  - opt. authors, url, dependencies, plugins, etc...

CMakeLists.txt

- **Build rules** for catkin
  - “Read” the package.xml
  - find other catkin packages to access libraries / include directories
  - export items for other packages depending on you

Catkin Packages Files

package.xml

```xml
<?xml version="1.0"?>
<package format="2">
  <name>beginner_tutorials</name>
  <version>0.1.0</version>
  <description>The beginner_tutorials package</description>

  <maintainer email="you@yourdomain.tld">Your Name</maintainer>
  <license>BSD</license>
  <url type="website">http://wiki.ros.org/beginner_tutorials</url>
  <author email="you@yourdomain.tld">Jane Doe</author>

  <buildtool_depend>catkin</buildtool_depend>

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

  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

</package>
```

CMakeLists.txt

- Complex and non-intuitive
  - Designed for machines
- You typically don’t modify it

Navigating Across ROS Packages

- rosbash
  - `roscd` – change directory to a package
    - you can reach a package dir without knowing the path
    - e.g. `roscd rospy`
  - `rosls` – list files of a package
  - …

- Where are the packages?
  - Installed ones: /opt/ros/<distro>
    - Typically without source code
  - User ones: anywhere
    - Found thanks to sourcing

Launch files

- Launch multiple nodes with one command
- roslaunch starts nodes as defined in the launch file
  - Nodes are executed sequentially
  - It starts the Master without having to write it
  - Accept arguments, can perform simple if-then

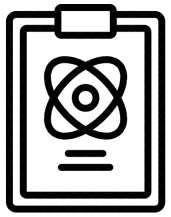
```
$ roslaunch [package] [filename.launch]
```

- To visualize the graph of launched nodes

```
$ rqt_graph
```

```
1  <launch>
2
3    <group ns="turtlesim1">
4      <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
5    </group>
6
7    <group ns="turtlesim2">
8      <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
9    </group>
10
11   <node pkg="turtlesim" name="mimic" type="mimic">
12     <remap from="input" to="turtlesim1/turtle1"/>
13     <remap from="output" to="turtlesim2/turtle1"/>
14   </node>
15
16 </launch>
```

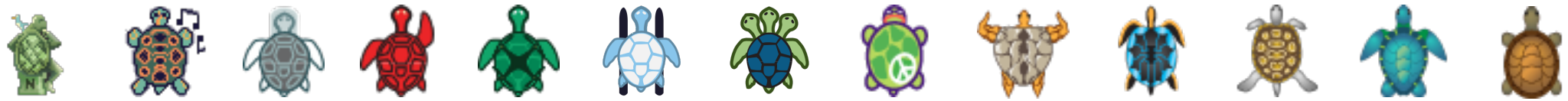
Lab Exercise



- Create a workspace
- Add and build the F1tenth simulator: *https://f1tenth.readthedocs.io/en/stable/going_forward/simulator/sim_install.html#*
  - Pay attention to the building process
  - Inspect package.xml, CMakeLists.txt, .launch files
  - Inspect other files

- Launch the simulator
  - Play with the simulator keys (To select the behavior: *j k b r n*)
  - Lists and analyze the nodes and the topics
  - Vizualize the nodes graph
  - Print the info and the (message) type of the topic */scan /odom /nav*
  - Echo of */scan* (use *–n1* to limit the print to one message)
  - Publish a command to */nav* (args example: *"drive: {speed: 0.5}"*)

ROS Versions



ROS Versions

	Distribution	EOL date	Details
	Melodic	May 2023	Last version for Ubuntu 18.04
	Noetic	May 2025	Last ROS 1 release, Ubuntu 20.04
	Foxy	May 2023	Latest LTS ROS 2 version, Ubuntu 20.04

ROS1 vs ROS2

- Support also for Windows, MacOs, RTOS
- Real-time nodes
- Python3 native
- Quality of Service
- Launch files in Python instead of XML
- Multiple nodes per process
- Data Distributed System (DDS)
- More..