Agimus winter school

Getting started with ROS 2

Ilenia Perrella, Thomas Peyrucain, David ter Kuile

Banyuls-sur-Mer, 2023

Table of contents

Introduction - PAL

Company

Founded in 2004

Located in Barcelona

+20 nationalities

~100 people

80% Engineers | 10% Ph.D.

Robots sales +35 countries

Our robots

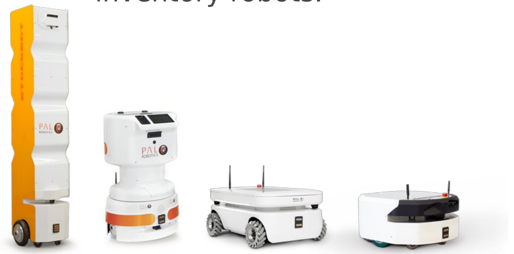
2004 — 2023

Business units

Intralogistics

INDUSTRY | RETAIL | HEALTHCARE

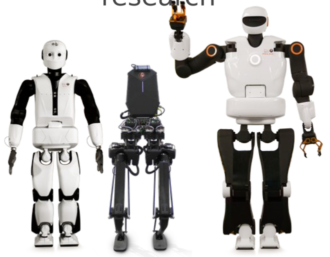
Platforms for automating transportation of goods, inventory robots.



Legged

RESEARCH | UNIVERSITIES

Humanoid service Platforms for state-of-the-art research



Mobile Interaction

RESEARCH | INDUSTRY | HEALTHCARE

ARI & TIAGo products and services for industry & research.

Introduction - TIAGo

Custom modular Mobile Manipulation
TIAGo Family

TIAGo
The Mobile Manipulator

Measurements
110cm - 145cm height

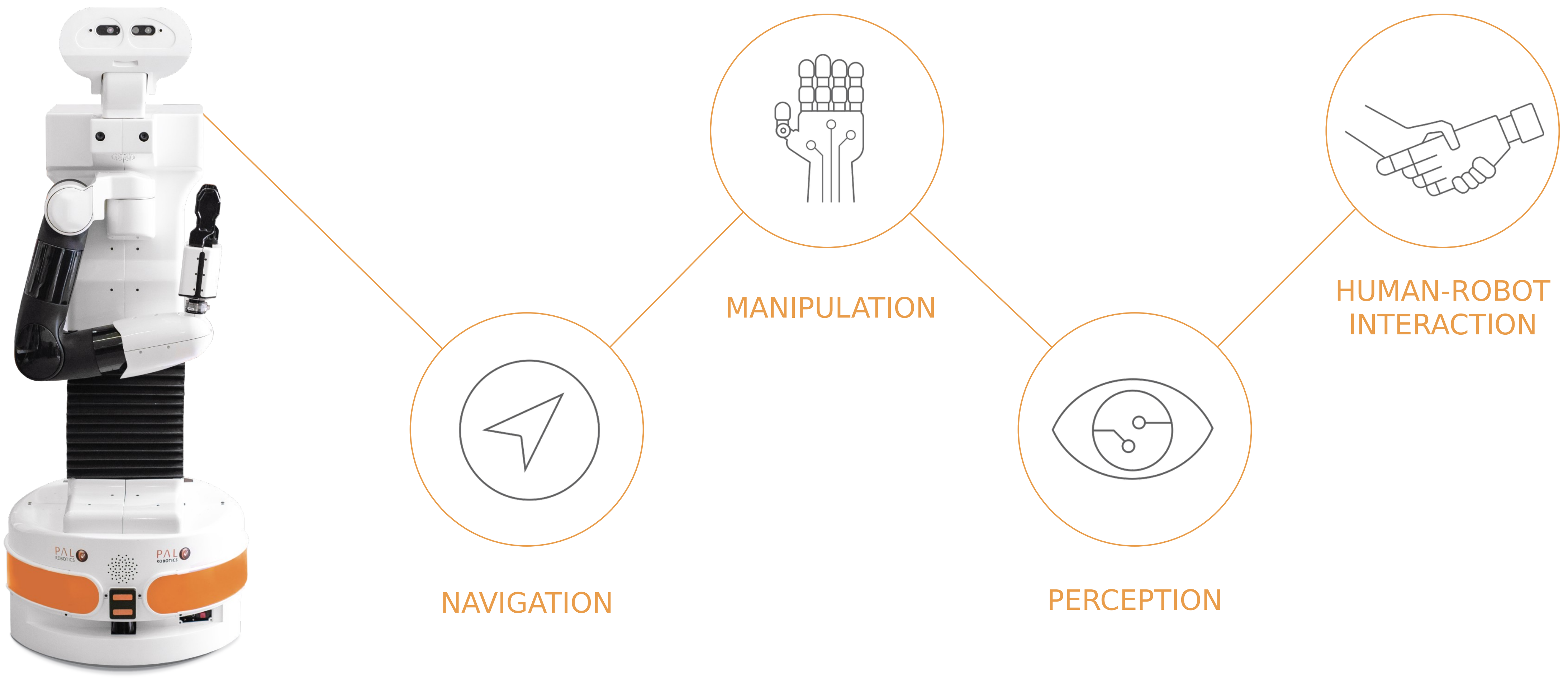
Torso
Expandable

Operating system
100% ROS integrated

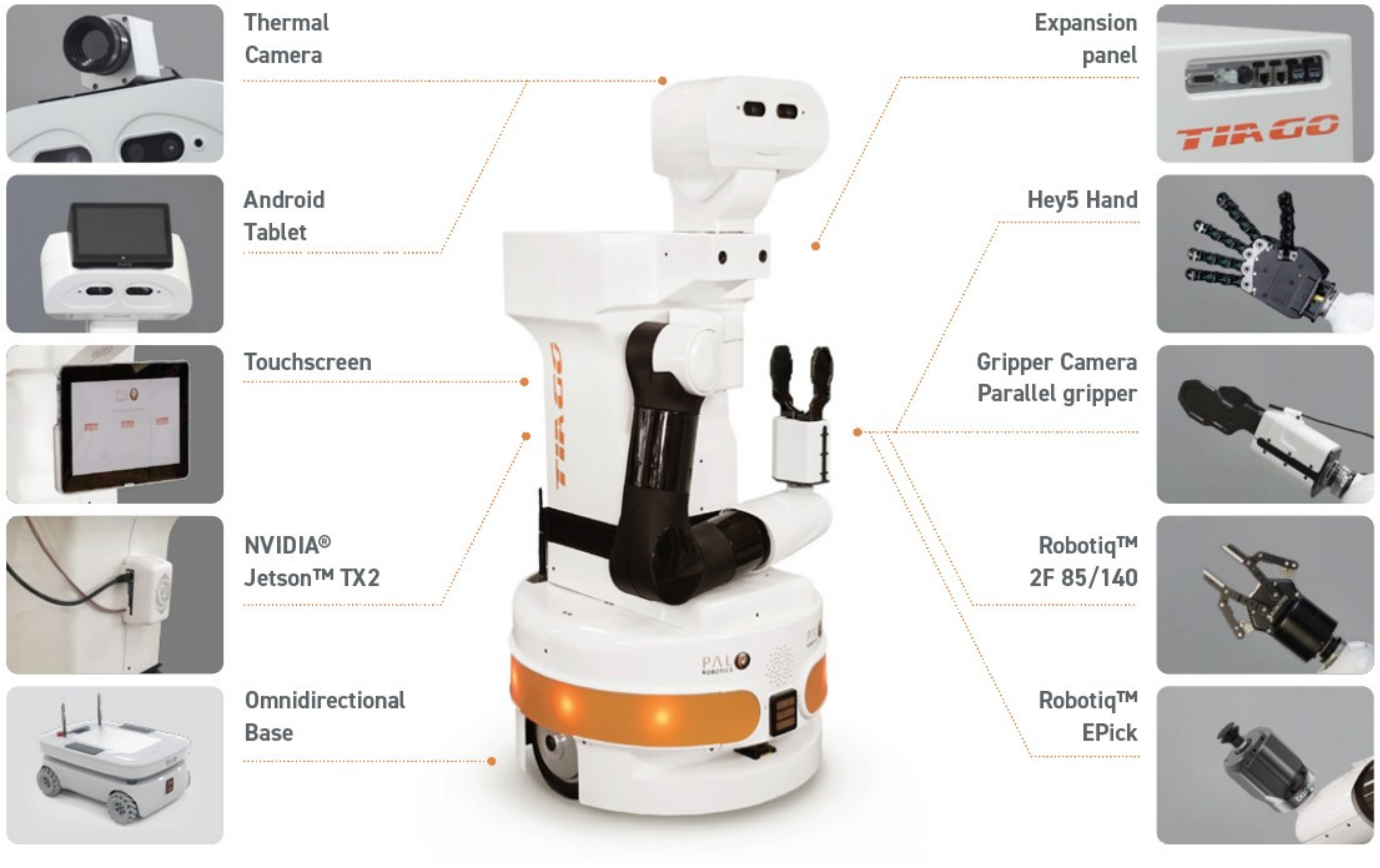
Tutorials and simulations
Free and available online

Sectors
Research | Industry | Ambient Assisted Living

Demonstration activities at Poznan University of Medical Science within Enrichme EU funded project.


NAVIGATION
MANIPULATION
PERCEPTION
HUMAN-ROBOT INTERACTION

TIAGo
Accessories



The robot that adapts to your research needs, not the other way around

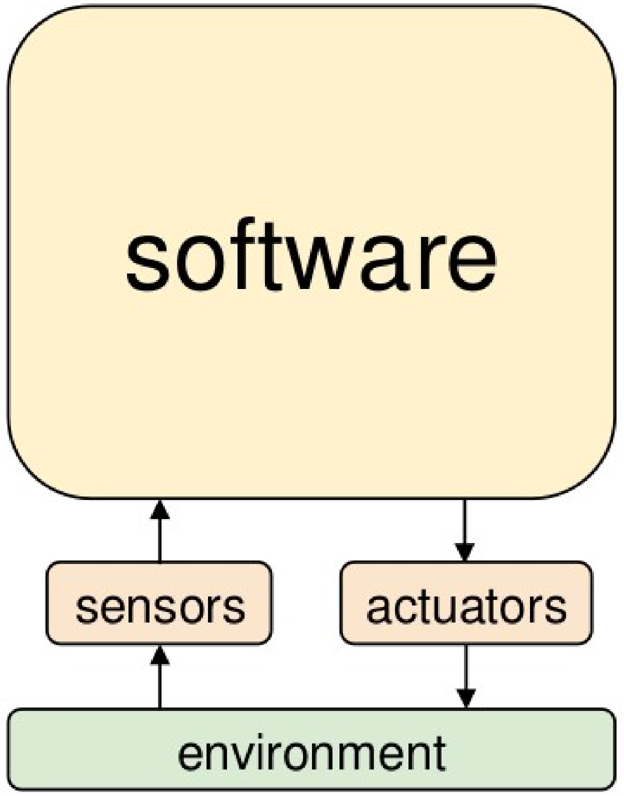
Introduction - ROS

ROS

Robotic standard

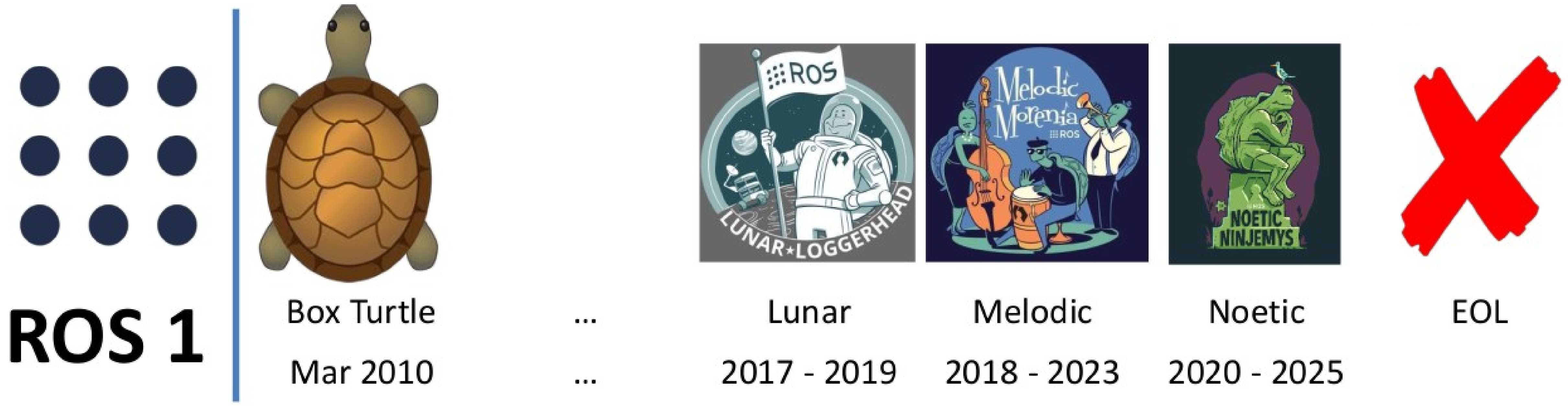
(Image from ROS Industrial Training Documentation)

Robotic standard



(Image from ROS Industrial Training [Documentation])

ROS 1 Lifecycle



(Image from ROS Industrial Training Documentation)

ROS 1 vs ROS 2

ROS 1:

- Research oriented
- Build on custom TCP/IP middleware
- Supports one robot per ROS network
- Centralised architecture
- Developed for Linux

ROS 1 vs ROS 2

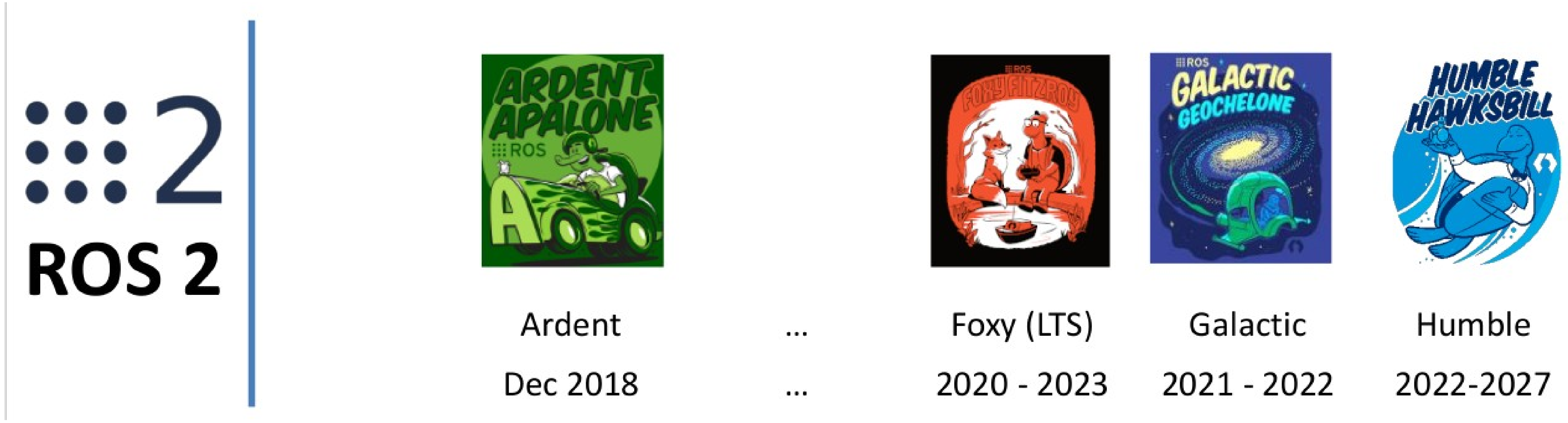
ROS 1:

- Research oriented
- Build on custom TCP/IP middleware
- Supports one robot per ROS network
- Centralised architecture
- Developed for Linux

ROS 2:

- Research and industrial oriented
- Build on industry proven DDS middleware
- Supports multiple robots per ROS network
- Decentralised architecture
- Supports Linux, MacOS and Windows

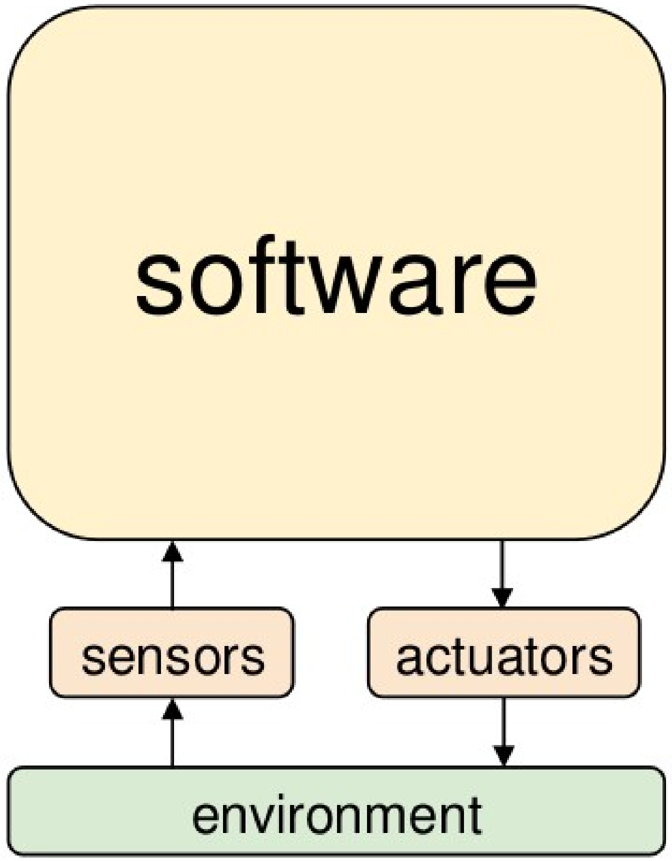
ROS 2 Lifecycle



(Image from ROS Industrial Training [Documentation]())

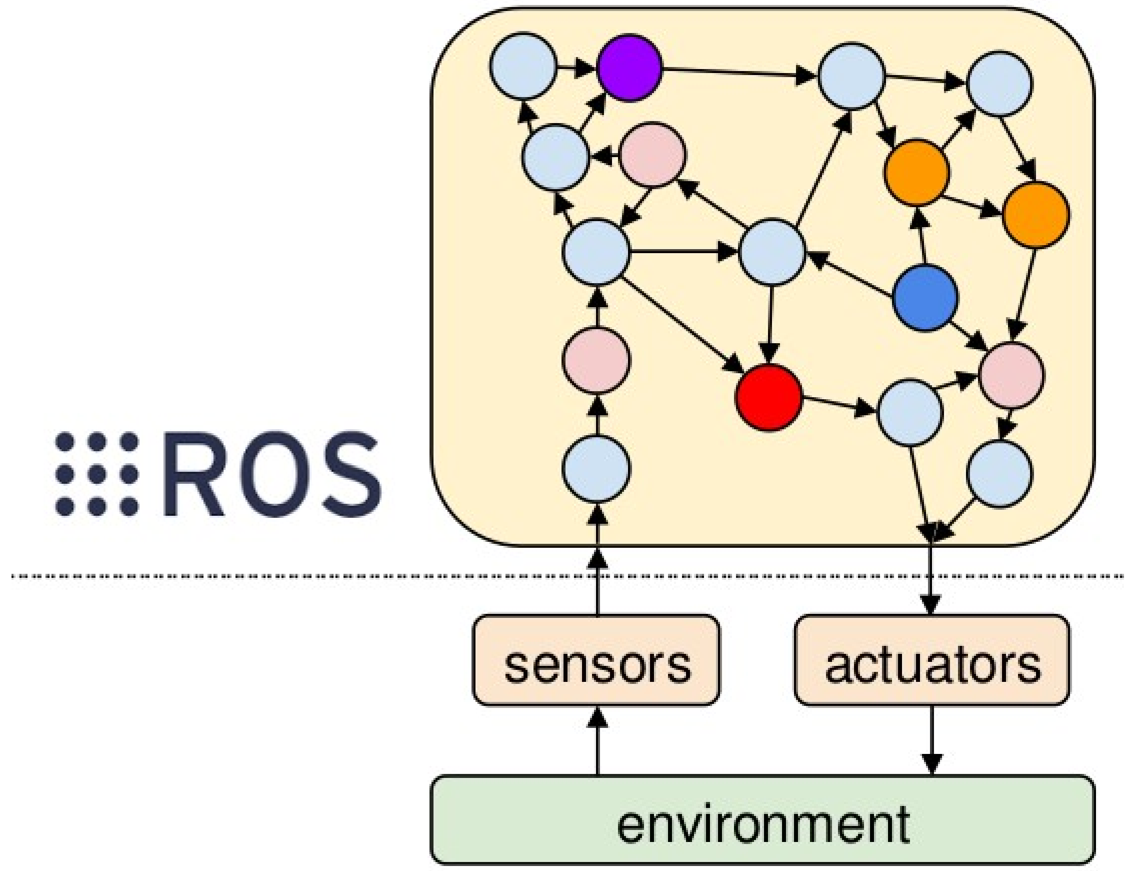
ROS 2 - Basics

ROS 2 - Nodes



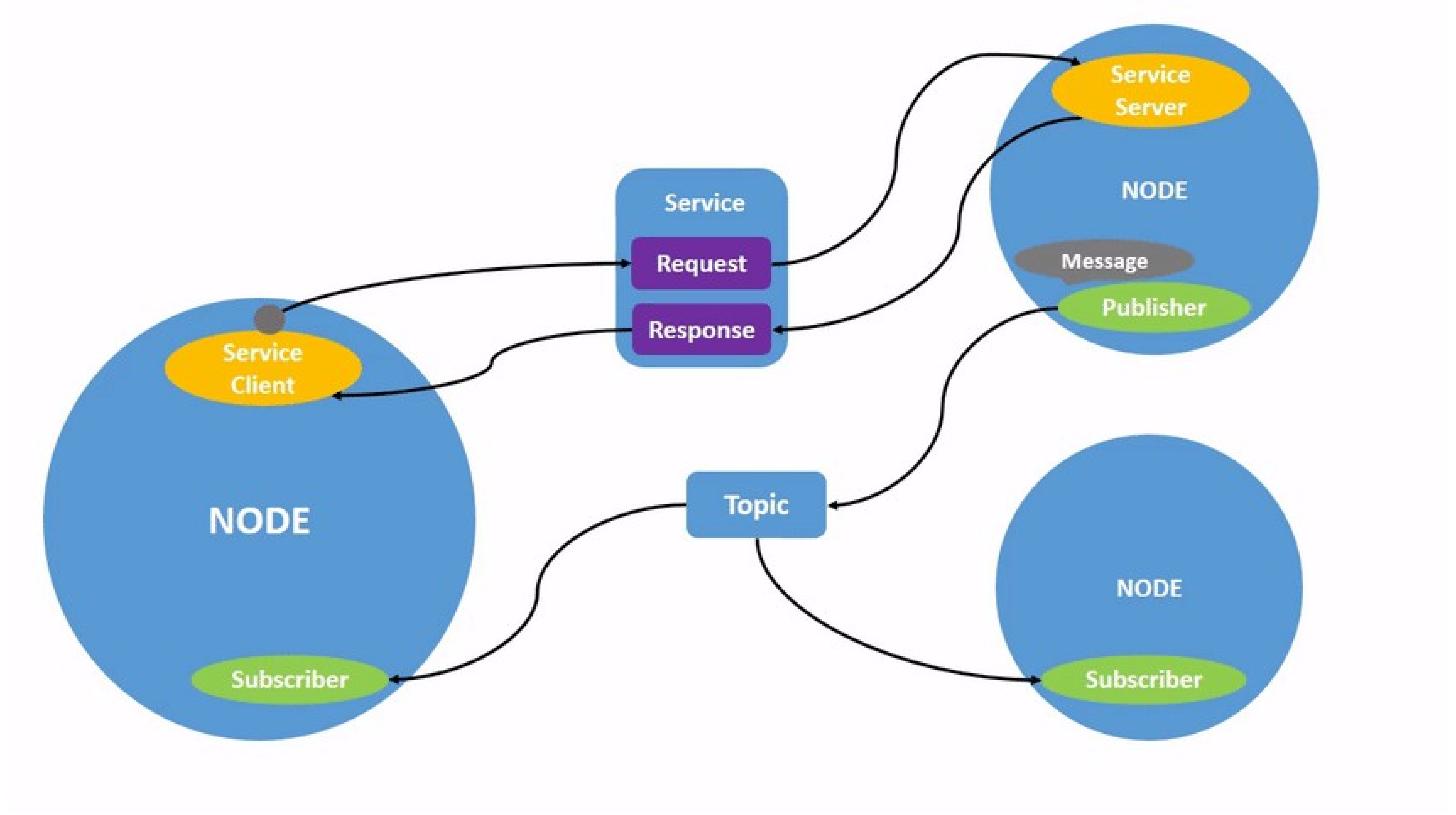
(Image from ROS Industrial Training Documentation)

ROS 2 - Nodes



(Image from ROS Industrial Training Documentation)

ROS 2 - Nodes



(Image from ROS Tutorials)

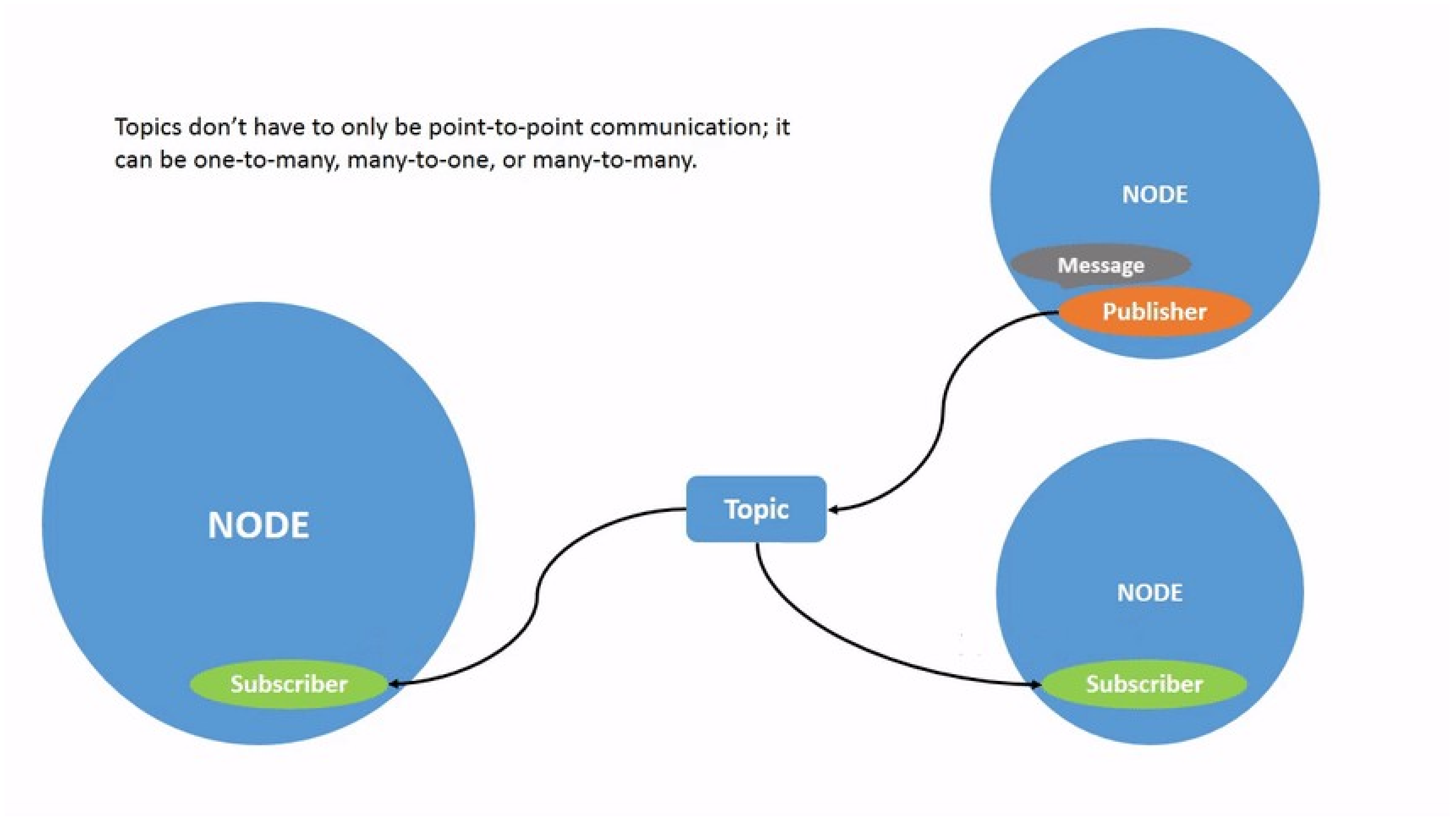
ROS 2 - Interfaces

- Messages
- Services
- Actions

ROS 2 - Messages (Topics)

- Continuous data stream
- Require no response
- Examples: Camera images, joint states etc.

ROS 2 - Publisher/Subscriber

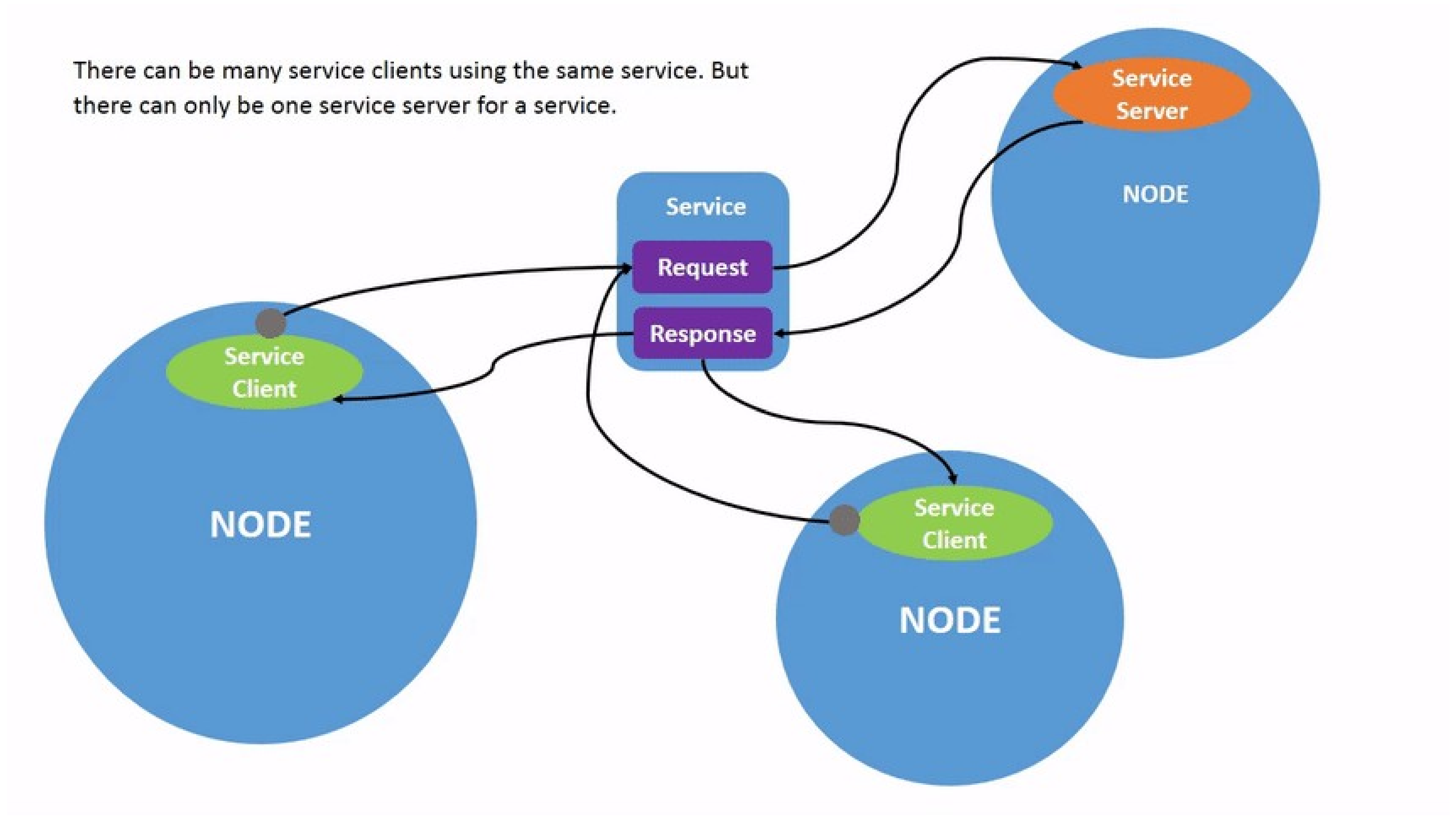

Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.
NODE
Message
Publisher
NODE
Subscriber
Topic
NODE
Subscriber

(Image from ROS Tutorials)

ROS 2 - Services

- Call and response
- Synchronous
- Examples: Change map, list controllers

ROS 2 - Service Client/Server

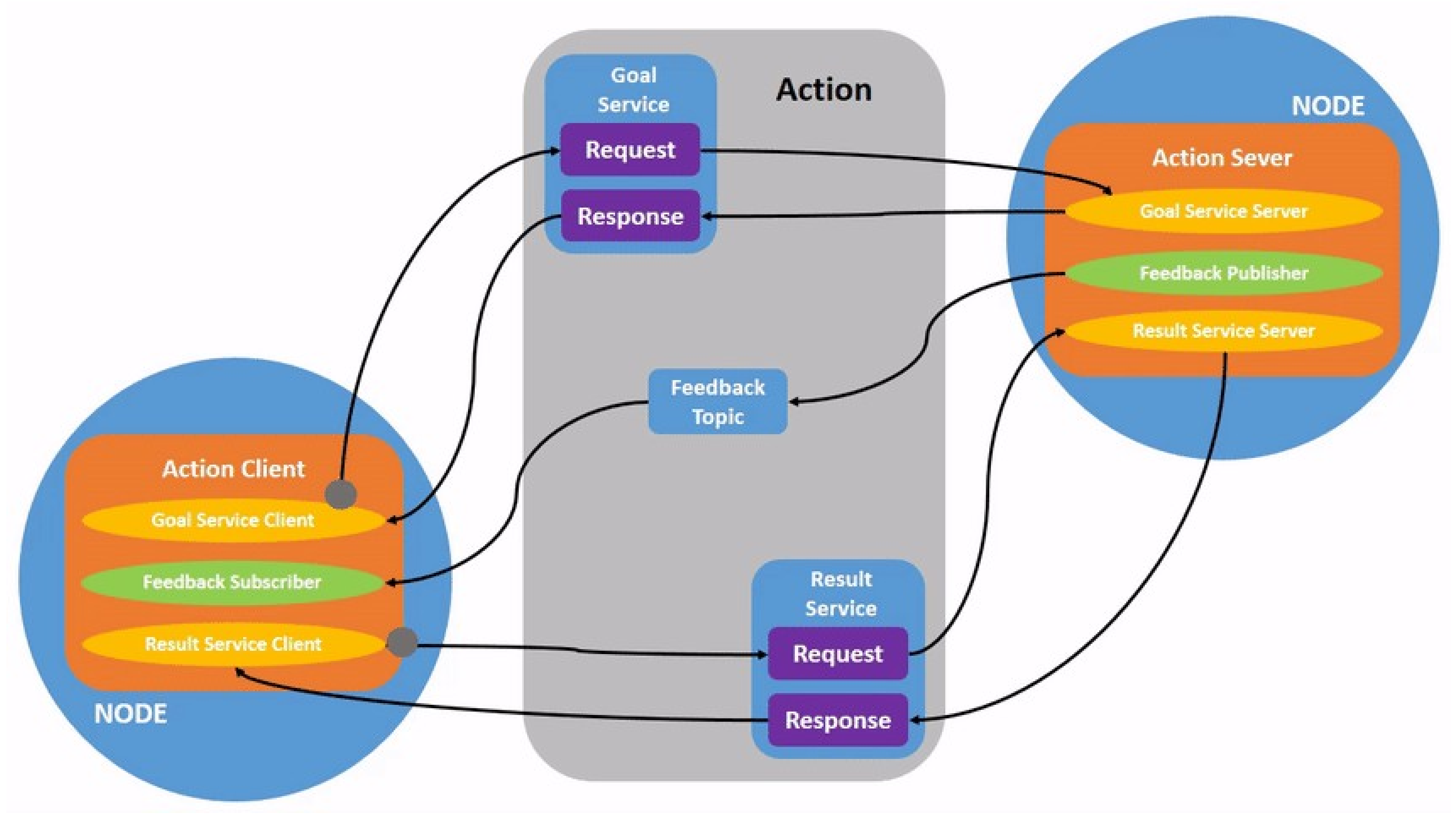

There can be many service clients using the same service. But there can only be one service server for a service.
Service
Request
Response
Service Server
NODE
Service Client
NODE
Service Client
NODE

(Image from [ROS Tutorials])

ROS 2 - Actions

- Execute long running tasks
- Examples: Navigate, play motions
- Goal, feedback and result
- Asynchronous
- Possibility of cancellation

ROS 2 - Action Client/Server



(Image from ROS Tutorials)

ROS 2 - Launch files

- Start multiple nodes at once
- Pass on parameters to nodes
- Include other launch files

ROS 2 - Launch files

```
10:57:39
~$ d
```

ROS 2 - Launch files

```
0:57:39
$ d
```

```
11:49:03
~$
```

```
11:48:58
~$
```

```
11:49:07
~$
```

davidterkuile@heemstede: ~ 49x27 | davidterkuile@heemstede: ~ 49x27 | davidterkuile@heemstede: ~ 49x27 | davidterkuile@heemstede: ~ 50x27

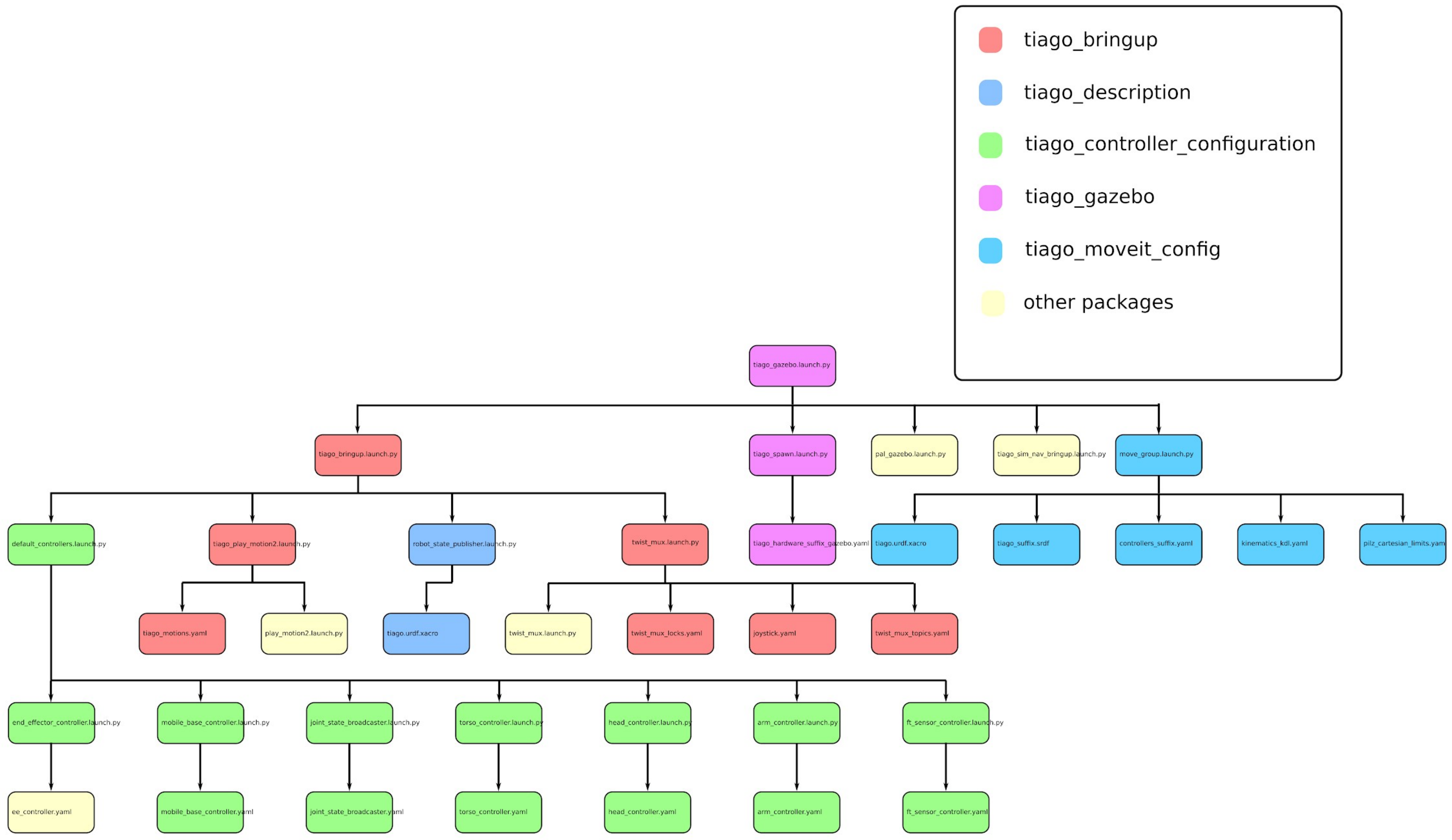
```
1:49:02
$
```

```
11:49:08
~$
```

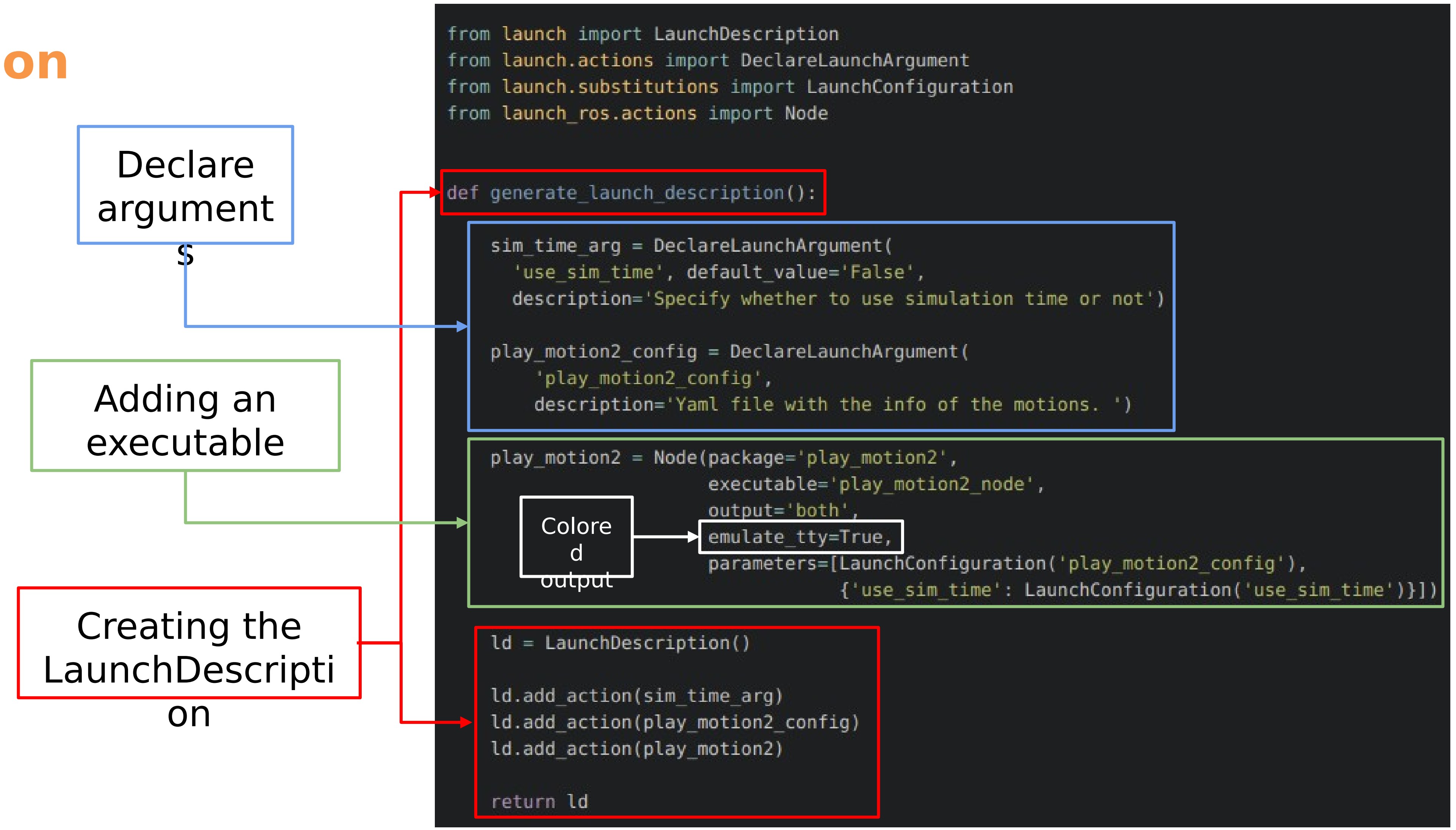
```
11:48:59
~$
```

```
11:49:07
~$
```

ROS 2 - Launch files



Python



```python
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node


def generate_launch_description():

    sim_time_arg = DeclareLaunchArgument(
      'use_sim_time', default_value='False',
      description='Specify whether to use simulation time or not')

    play_motion2_config = DeclareLaunchArgument(
        'play_motion2_config',
        description='Yaml file with the info of the motions. ')

    play_motion2 = Node(package='play_motion2',
                        executable='play_motion2_node',
                        output='both',
                        emulate_tty=True,
                        parameters=[LaunchConfiguration('play_motion2_config'),
                                    {'use_sim_time': LaunchConfiguration('use_sim_time')}])

    ld = LaunchDescription()

    ld.add_action(sim_time_arg)
    ld.add_action(play_motion2_config)
    ld.add_action(play_motion2)

    return ld
```

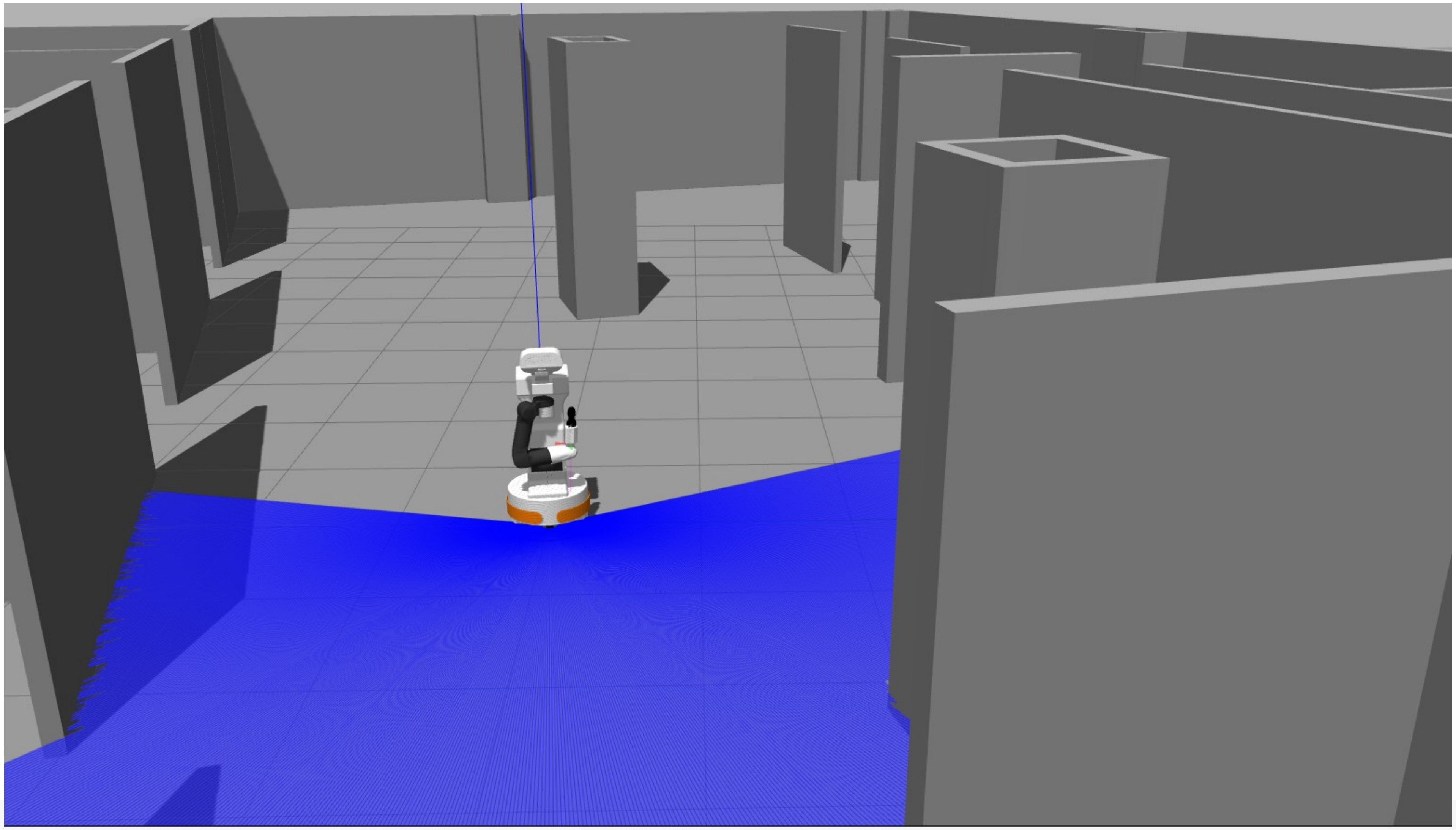
```
ros2 launch pal_tts tts.launch.py
```

ROS 2 - Simulation and visualization

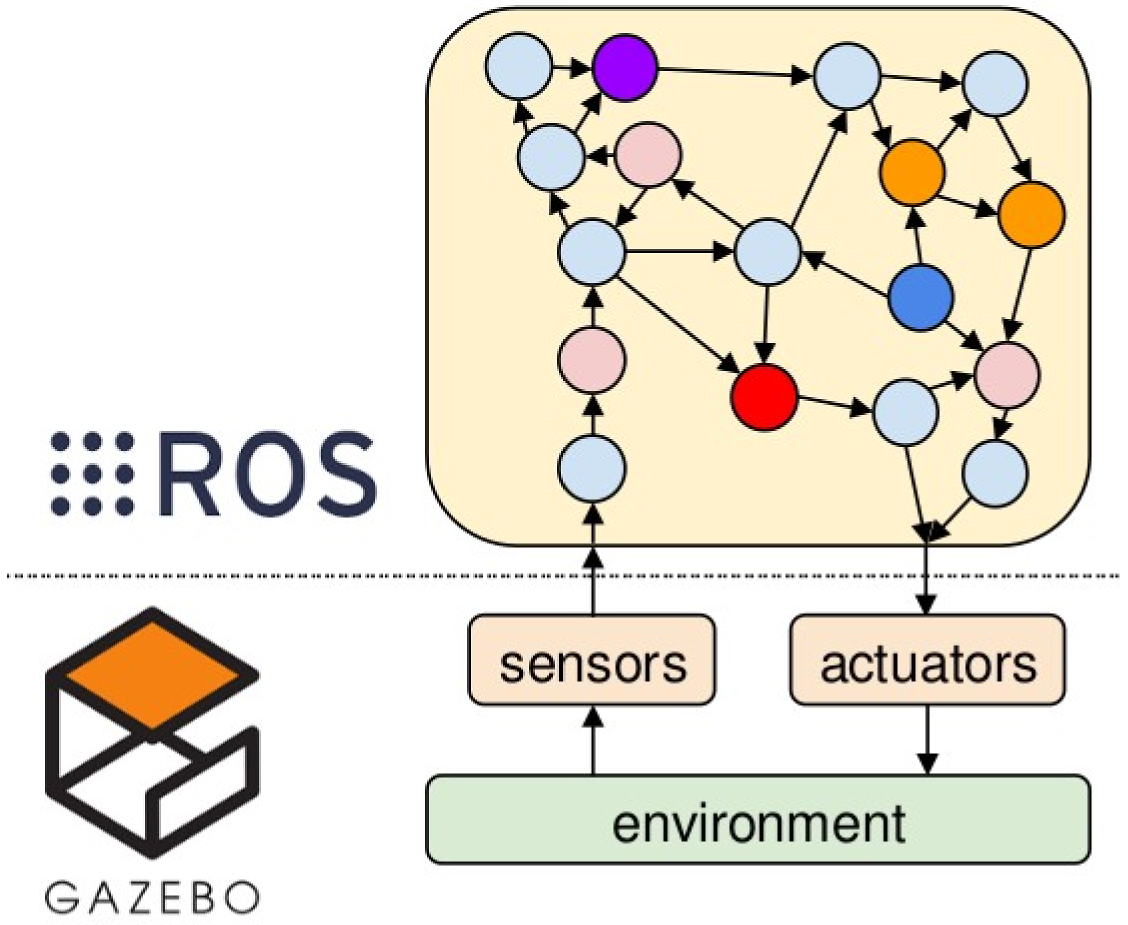
ROS 2 - Simulation

GAZEBO

ROS 2 - Simulation

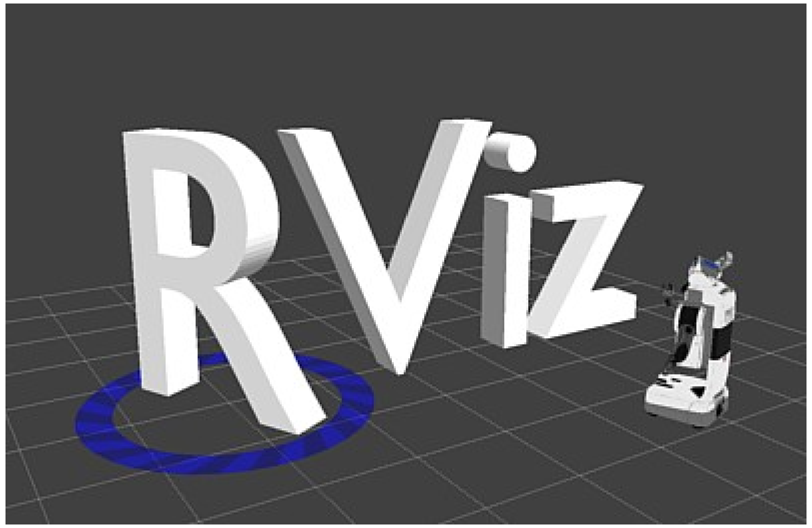


ROS 2 - Simulation

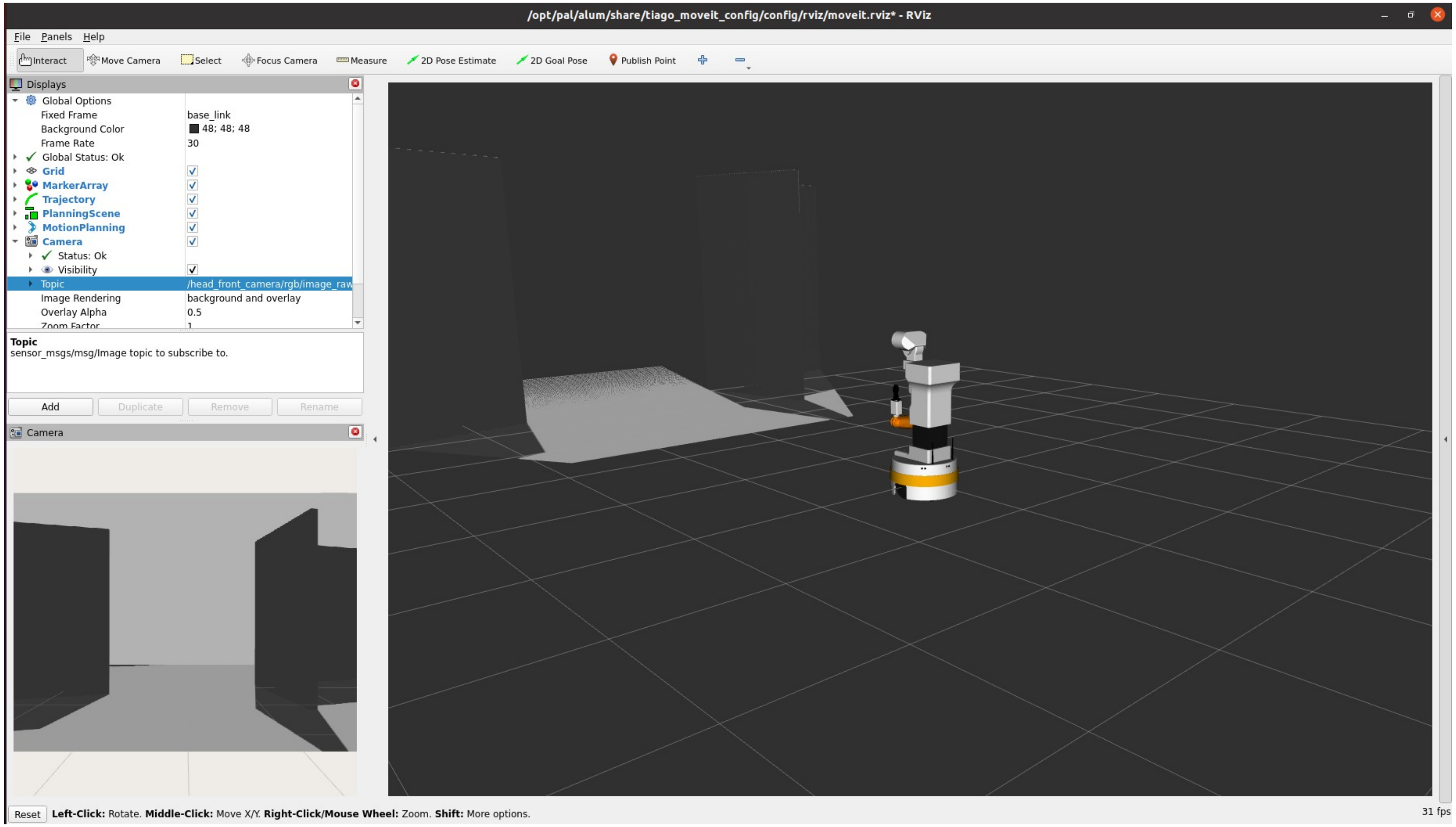


(Image from ROS Industrial Training Documentation)

RViz



RViz



/opt/pal/alum/share/tiago_moveit_config/config/rviz/moveit.rviz* - RViz

Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: Ok	
Grid	✓
MarkerArray	✓
Trajectory	✓
PlanningScene	✓
MotionPlanning	✓
Camera	✓
Status: Ok	
Visibility	✓
Topic	/head_front_camera/rgb/image_raw
Image Rendering	background and overlay
Overlay Alpha	0.5
Zoom Factor	1

Topic
sensor_msgs/msg/Image topic to subscribe to.

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options.

ROS 2 - Development

ROS 2 - Environment configuration

~/agimus_ws/
custom_package_A

/opt/pal/alum/
custom_package_A

/opt/ros/humble
custom_package_A

- lib/
  - Contains the installed code of the packages
- setup.bash
  - The file to source for the bash shell to find the ros related commands
- share/
  - Contains the configuration folders used in the different packages

ROS 2 - Domain ID

- All ROS 2 nodes use domain ID 0 by default.
- To avoid interference between different groups of computers running ROS 2 on the same network, a different domain ID should be set for each group.
- The domain ID is used by DDS to compute the UDP ports that will be used for discovery and communication.
- Topics, Actions, services, etc.. cannot be seen from 2 machines with 2 different Domain ID
- The highest domain ID that can possibly be assigned is 232

ROS 2 - Workspace

- A full workspace looks like this:
- agimus_ws/
  - build/
  - install/
    - setup.bash
  - log/
  - src/
    - my_package1
    - my_package2
    - .....

ROS 2 - Colcon

- Inside your workspace:
  - source /opt/pal/alum/setup.bash
  - colcon build

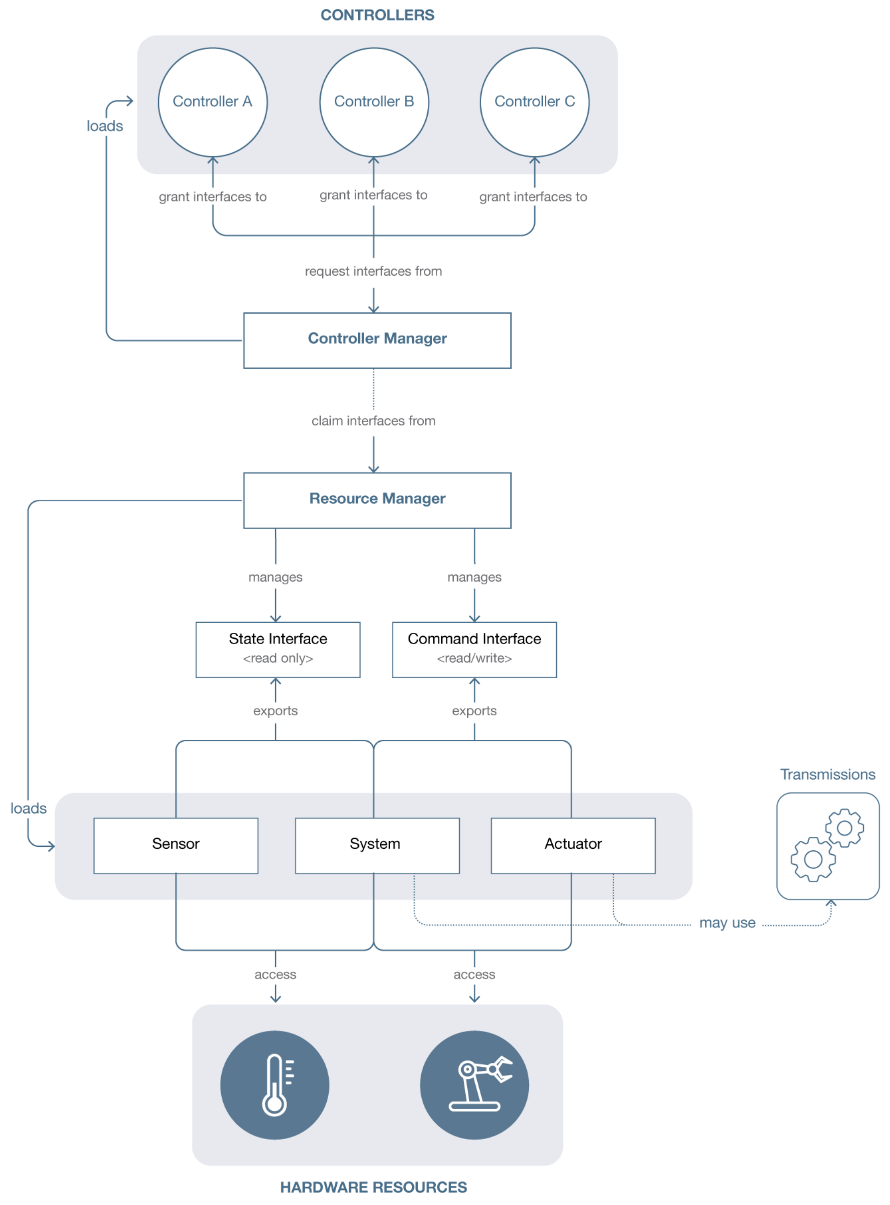
- Colcon will use the CMake instructions to install each packages

- The install directory contains you workspace’s setup files, which you can use to source your overlay. => source install/setup.bash

- echo $COLCON_PREFIX_PATH

- To clean the workspace: rm -rf build/ install/ log/

ROS 2 - Control


CONTROLLERS
Controller A
Controller B
Controller C
loads
grant interfaces to
grant interfaces to
grant interfaces to
request interfaces from
Controller Manager
claim interfaces from
Resource Manager
manages
manages
State Interface
<read only>
Command Interface
<read/write>
exports
exports
loads
Sensor
System
Actuator
Transmissions
may use
access
access
HARDWARE RESOURCES

Hardware Components

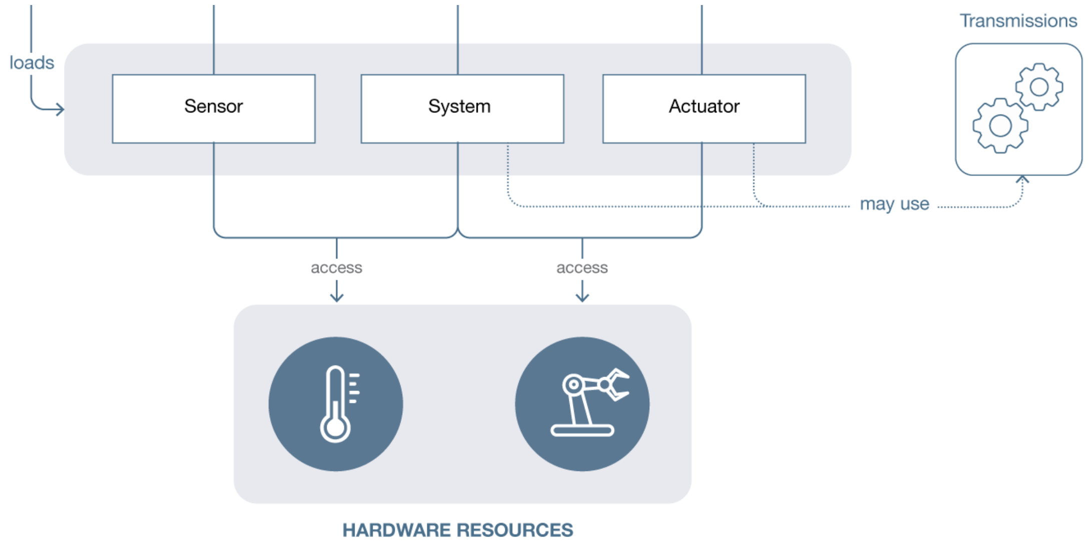
- Provides an abstraction from ros2_control to realize the communication with the physical hardware
- The components are exported as plugins
- Resource manager is responsible for loading them and maintaining their lifecycle

Types

Sensor

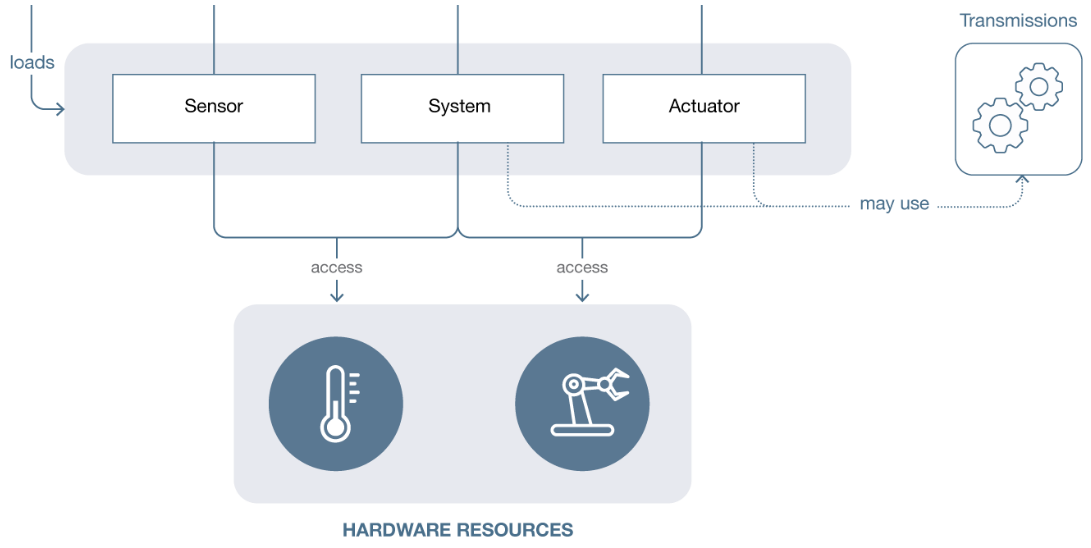
Actuator

System



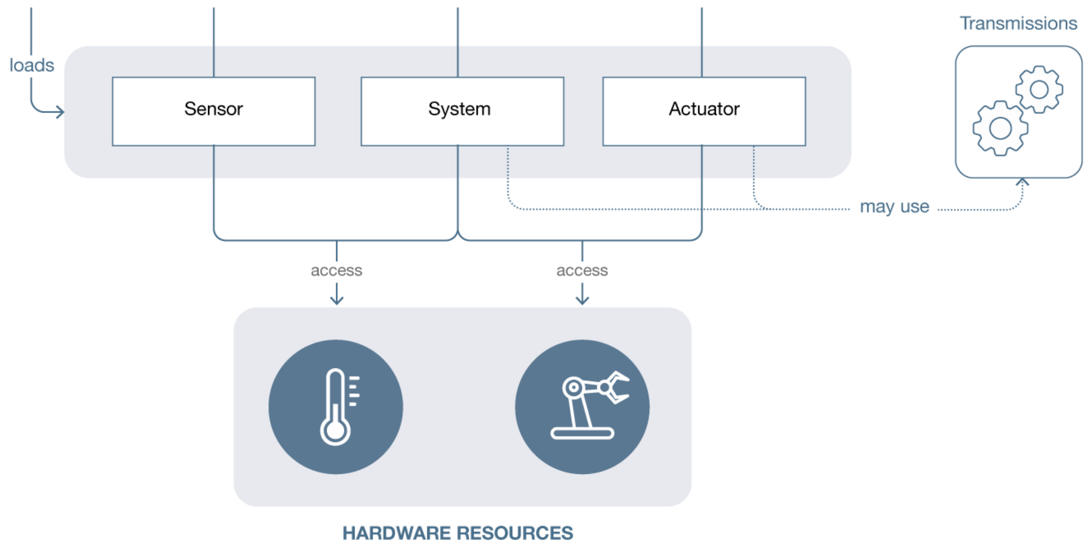
Sensor Actuator System

```
<ros2_control name="ForceSensor" type="sensor">
 <hardware>
   <plugin>vendor_specific/ForceSensorHardware</plugin>
   <param name="foo">0.43</param>
 </hardware>
 <sensor name="my_force_sensor">
   <state_interface name="force"/>
   <param name="frame_id">rrbot_tcp</param>
   <param name="force_limit">100</param>
 </sensor>
</ros2_control>
```



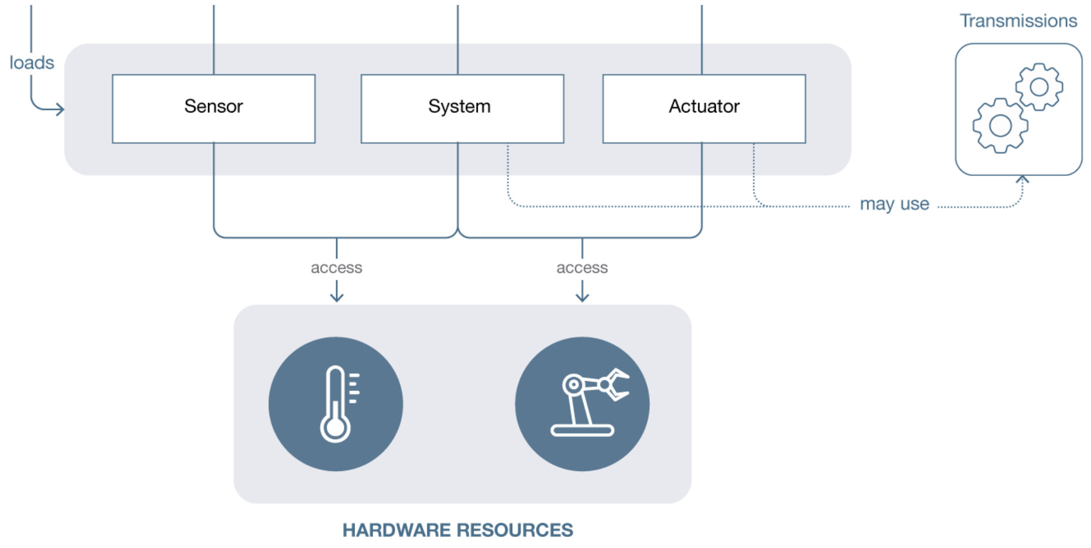
Sensor **Actuator** System

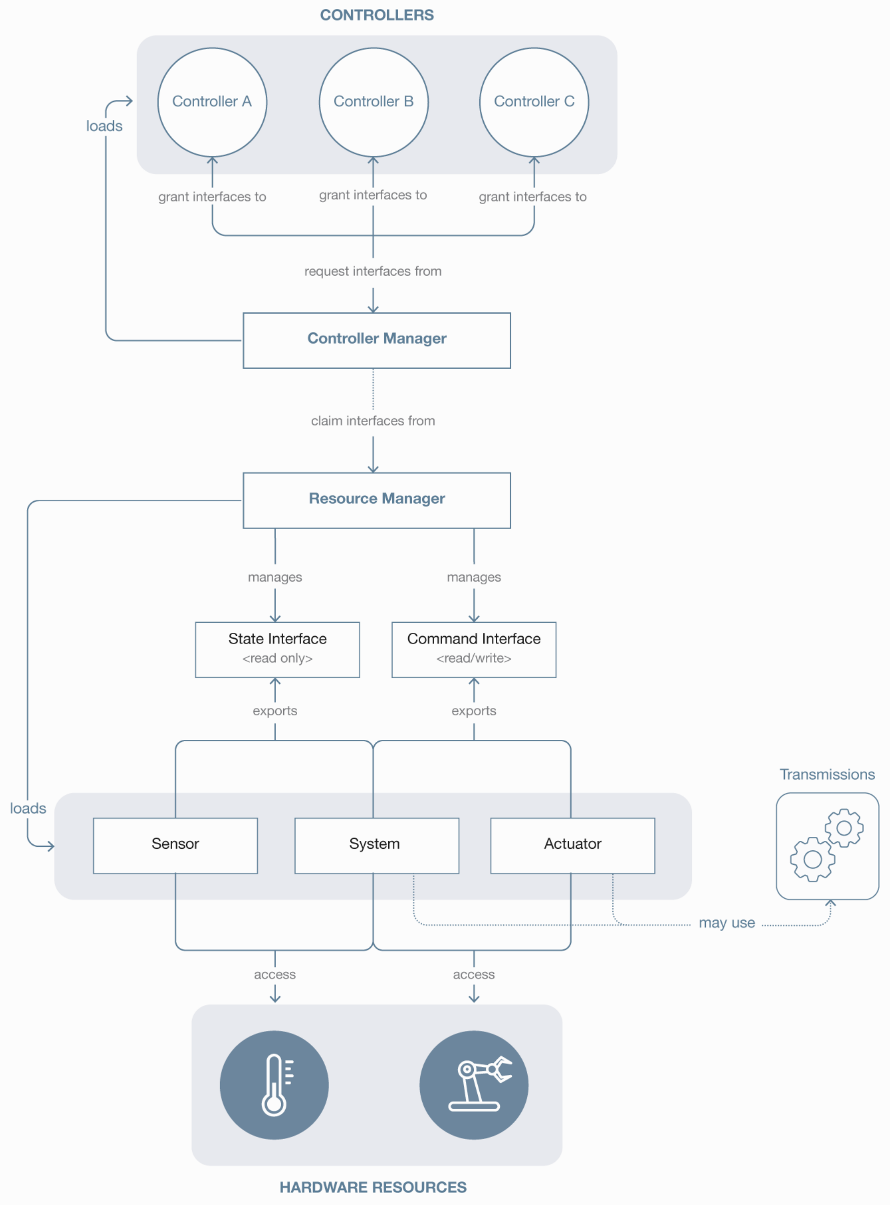
```xml
<ros2_control name="Gripper" type="actuator">
 <hardware>
   <plugin>vendor_specific/PositionActuatorHardware</plugin>
   <param name="foo">1.23</param>
   <param name="bar">3</param>
 </hardware>
 <joint name="gripper_joint">
   <command_interface name="position">
     <param name="min">0</param>
     <param name="max">50</param>
   </command_interface>
   <state_interface name="position"/>
   <state_interface name="velocity"/>
 </joint>
</ros2_control>
```

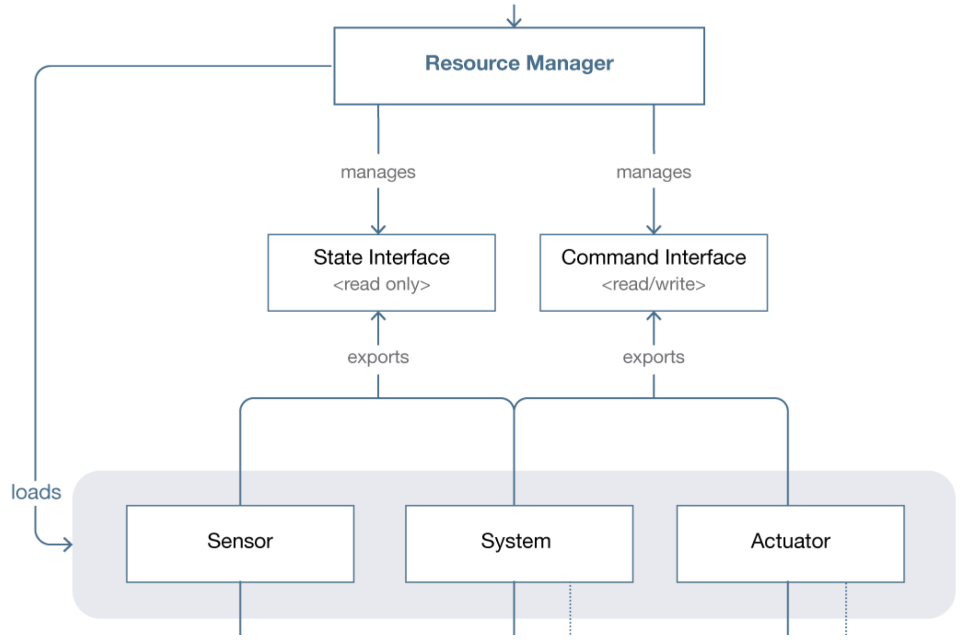


Sensor   Actuator   **System**

```
<ros2_control name="MyRobotArm" type="system">
 <hardware>
   <plugin>vendor_specific/PositionOnlyHardware</plugin>
   <param name="foo">2</param>
   <param name="bar">2</param>
 </hardware>
 <joint name="joint1">
   <command_interface name="position"/>
   <state_interface name="position"/>
 </joint>
 <joint name="joint2">
   <command_interface name="position"/>
   <state_interface name="position"/>
 </joint>
</ros2_control>
```

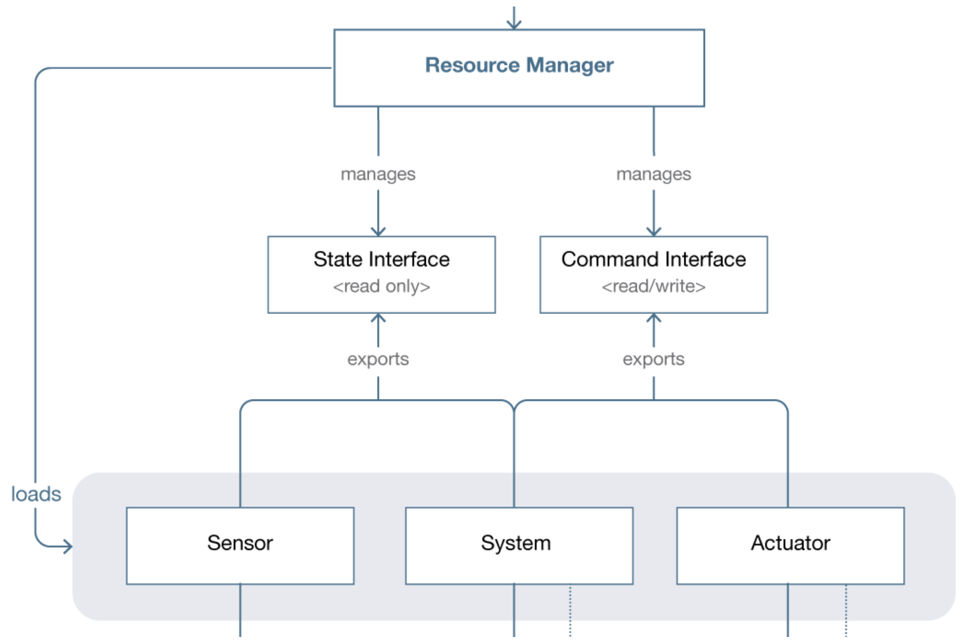



CONTROLLERS
Controller A
Controller B
Controller C
loads
grant interfaces to
grant interfaces to
grant interfaces to
request interfaces from
Controller Manager
claim interfaces from
Resource Manager
manages
manages
State Interface
<read only>
Command Interface
<read/write>
exports
exports
loads
Sensor
System
Actuator
Transmissions
may use
access
access
HARDWARE RESOURCES


Resource Manager
manages
manages
State Interface
<read only>
Command Interface
<read/write>
exports
exports
loads
Sensor
System
Actuator

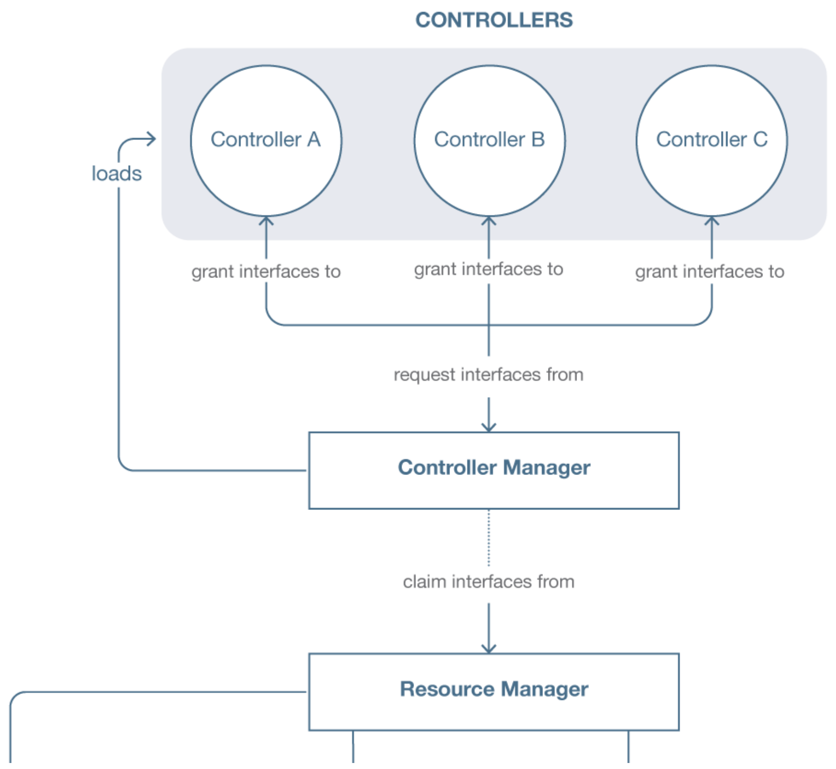
Resource Manager

- Abstracts physical hardware and its drivers (called hardware components)
- The loaded components are plugin based
- Responsible for loading them, maintaining their lifecycle, and components' state and command interfaces
- Why this level of abstraction?
  - Reuse of implemented hardware components
  - Flexible hardware applications for both state and command interfaces



Controller Manager

- An entry point for users via ROS services
- A node without an executor
- Connects controllers and the hardware-abstraction layer
- Manages Loading, Configuring, Activation, Deactivation, and Unloading of the controllers.
- Responsible for granting controllers access to the hardware via interfaces when enabled
- Manages the access to the hardware interfaces



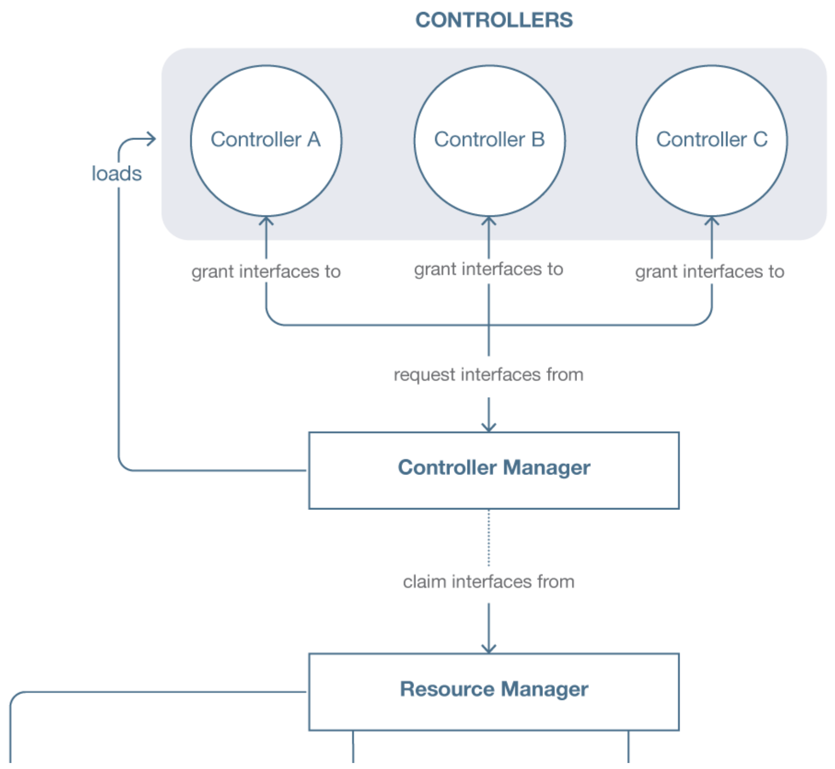
Controller Manager

read()

update()

write()

- Reads data from the hardware and updates the interfaces
- Handle hardware read errors



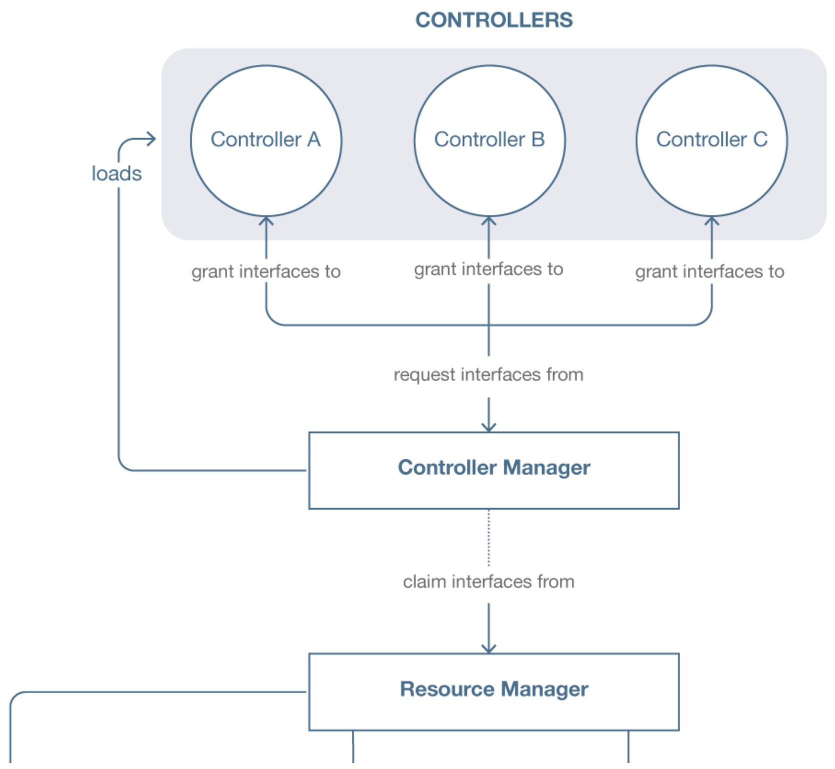
Controller Manager

read()

update()

write()

- Run the controller update cycle
- Maintain the controller update rate
- Manages the controller switching
- Outcome depending on the controllers update result



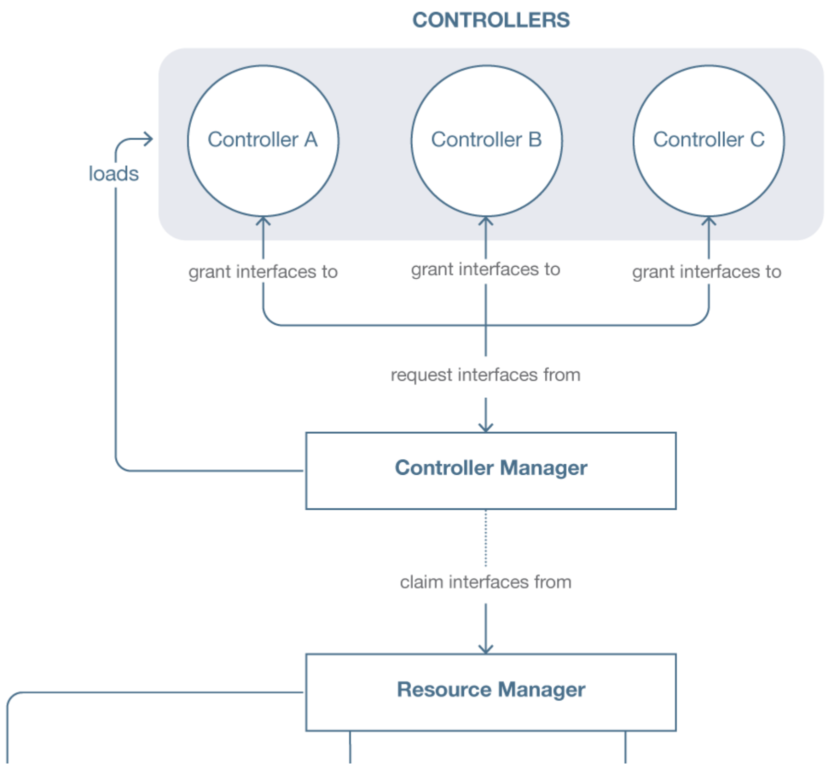
Controller Manager

read()

update()

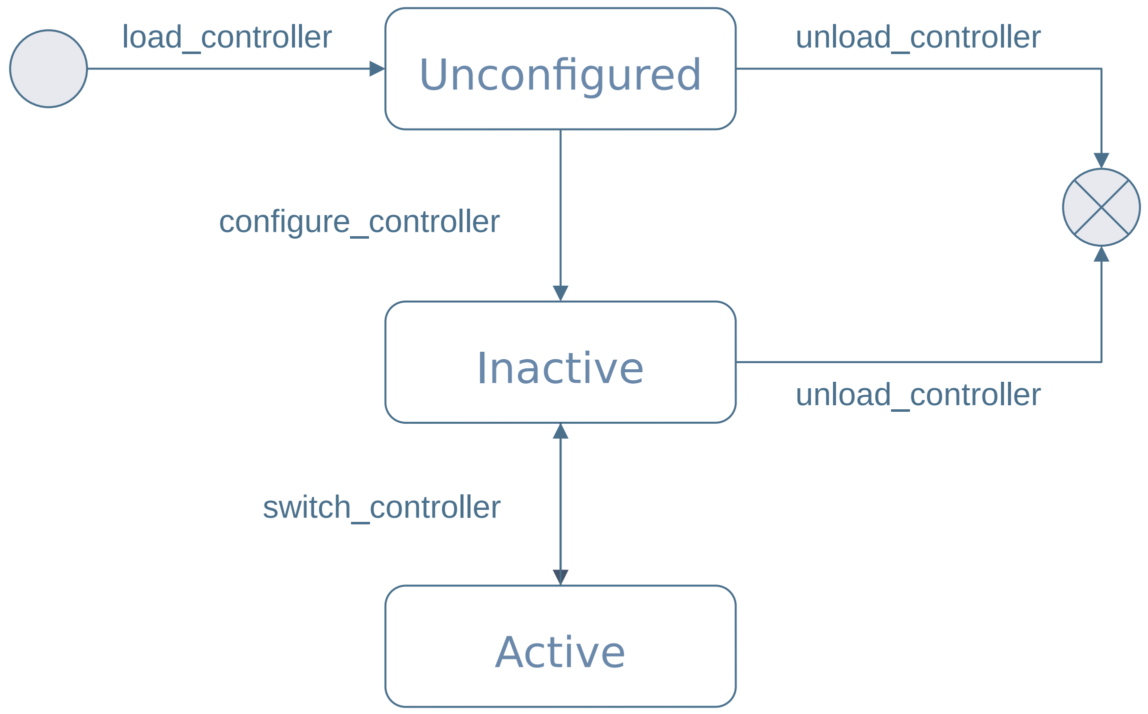
write()

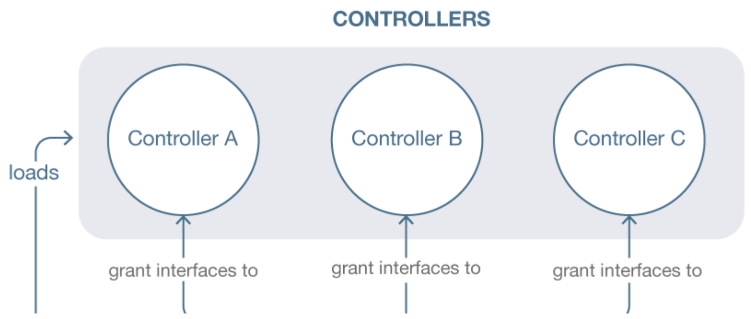
- Writes data to the hardware from the interfaces updated by the controller
- Handle hardware write errors



ROS 2 Controllers

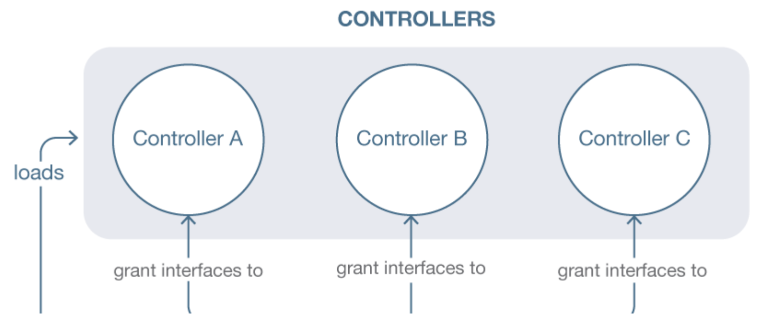
- Similar job of a ROS Controller but better with lifecycle

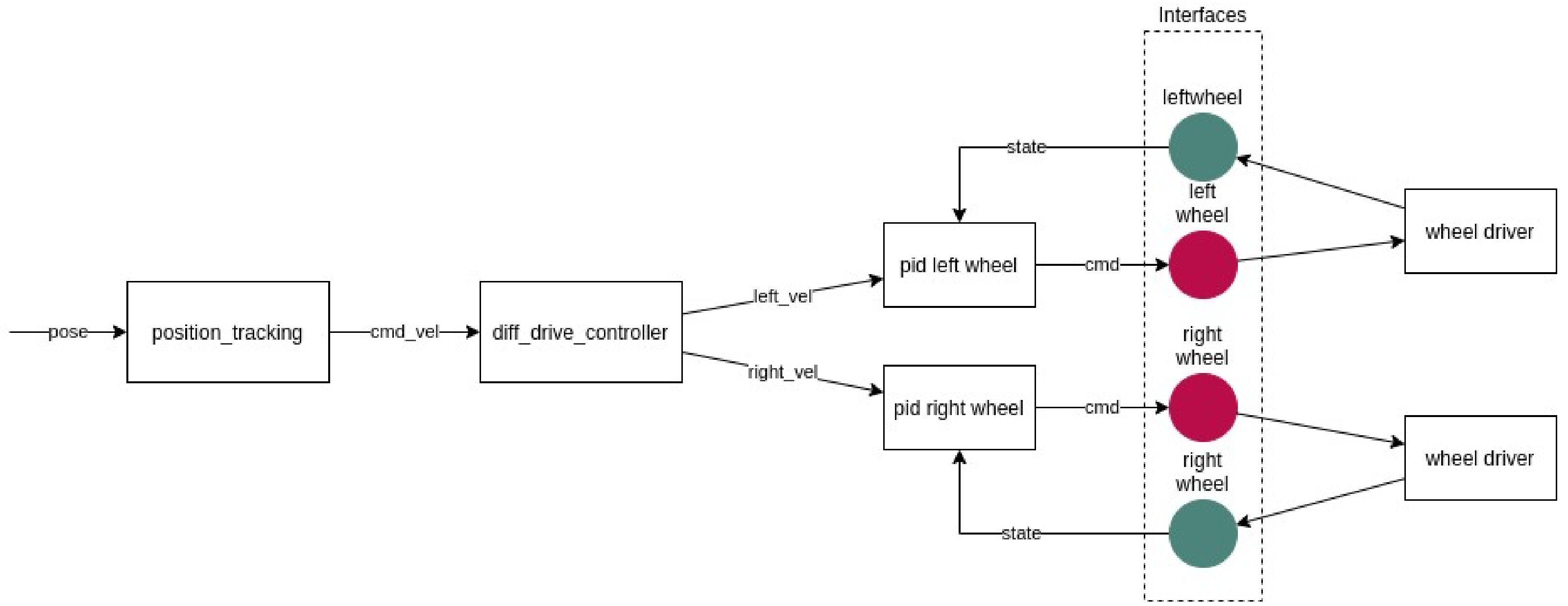




ROS 2 Controllers

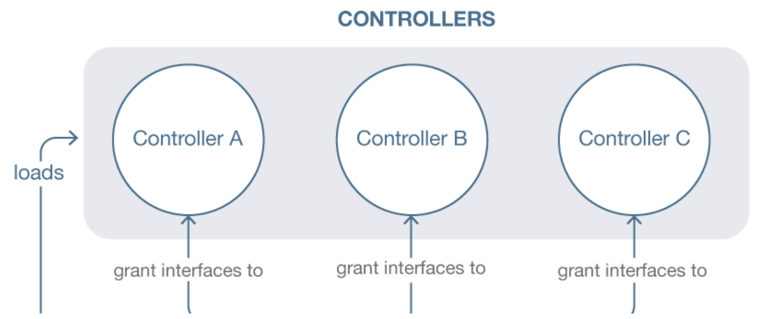
- Similar job of a ROS Controller but better with lifecycle
- Loaning of state and command interfaces
- Ability to chain with other controllers
- Ability to update synchronously and asynchronously
- Ability to have different update rates w.r.t controller manager





ROS 2 Controllers

- Similar job of a ROS Controller but better with lifecycle
- Loaning of state and command interfaces
- Ability to chain with other controllers
- Ability to update synchronously and asynchronously
- Ability to have different update rates w.r.t controller manager



joint_trajectory_controller

diff_drive_controller

gripper_controllers

joint_state_broadcaster

ackermann_steering_controll er

force_torque_sensor_broadca ster

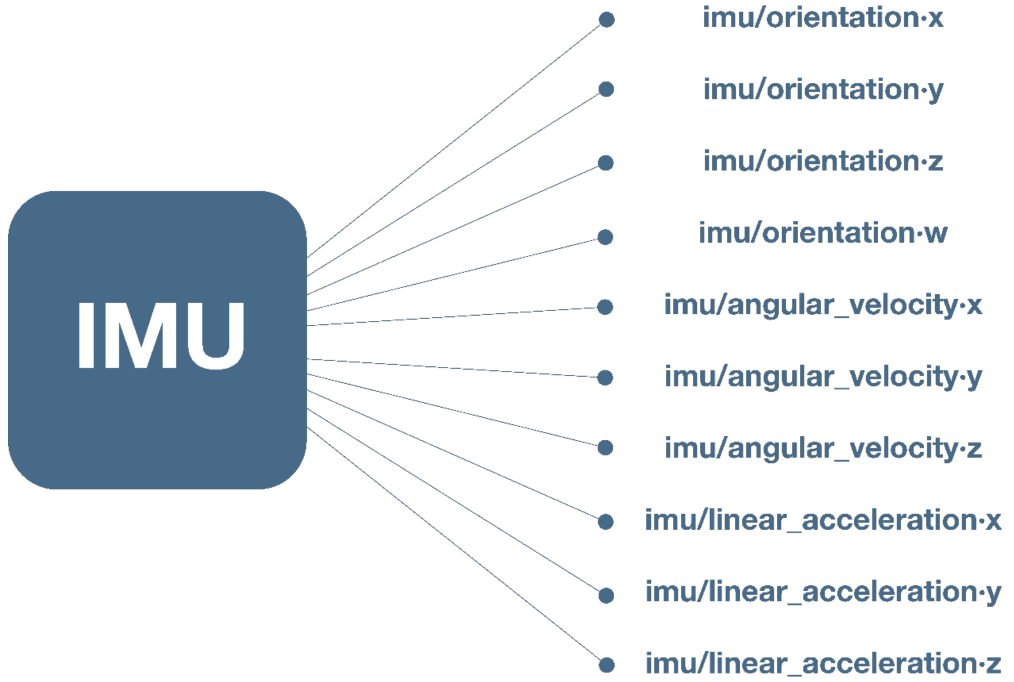
admittance_controller

bicycle_steering_controller

imu_sensor_broadcaster

Semantic Components

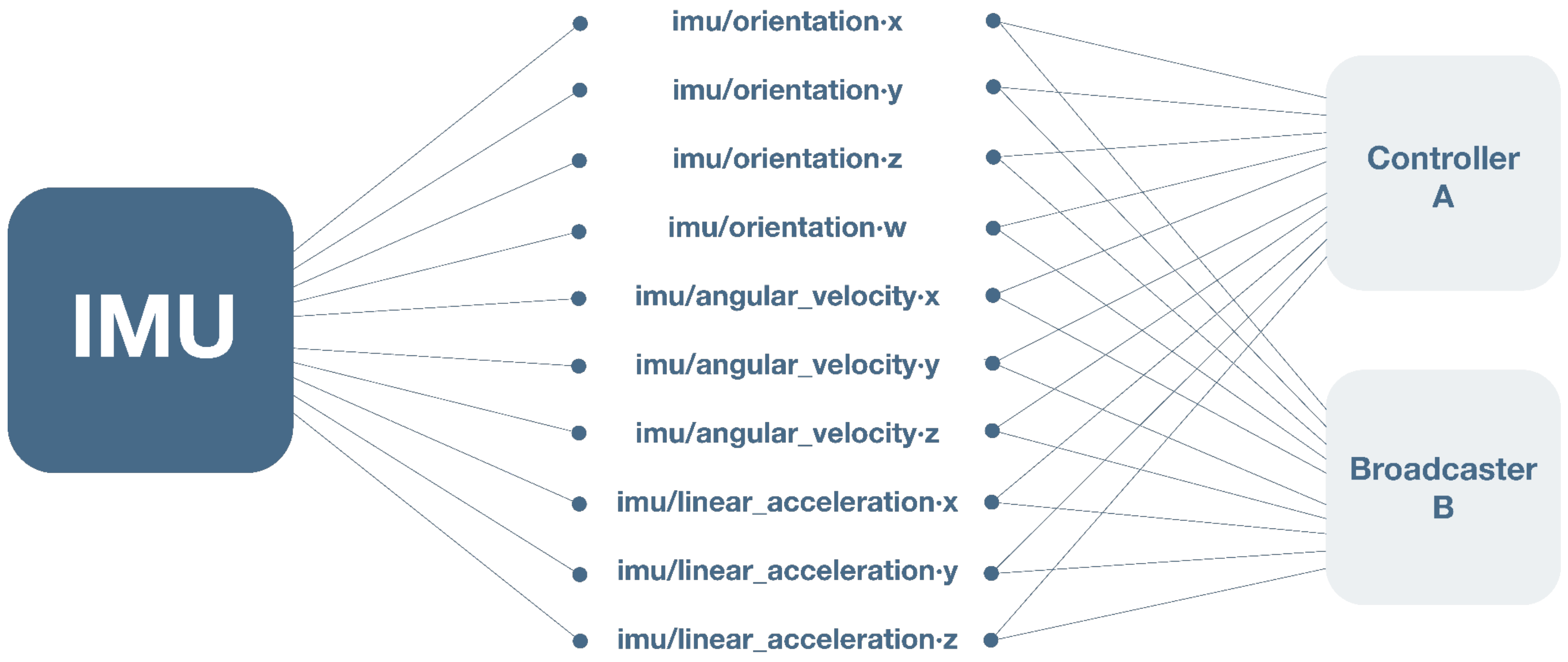
- Why do we need it?



Semantic Components

- Why do we need it?

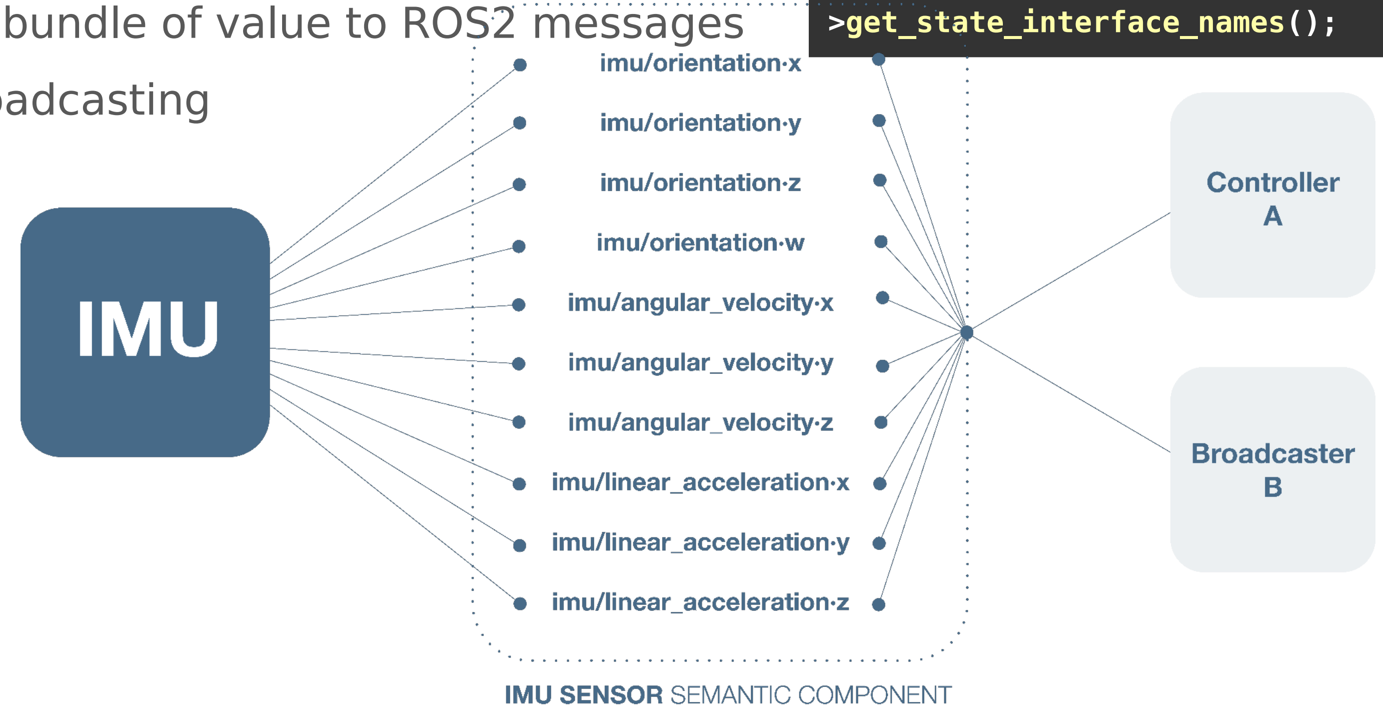
```
using namespace controller_interface;
InterfaceConfiguration interfaces_config;
interfaces_config.type =
interface_configuration_type::INDIVIDUAL;
interfaces_config.names = {"imu/orientation.x",
"imu/orientation.y", "imu/orientation.z", "imu/orientation.w"};
```



Semantic Components

- Loaning of the state interfaces
- Abstract all interfaces with semantic meaning into one component
- API to convert from bundle of value to ROS2 messages directly for easy broadcasting

```
imu_sensor_ =
std::make_unique<semantic_components::IMUSensor>(
            semantic_components::IMUSensor("imu"));
...
using namespace controller_interface;
InterfaceConfiguration state_interfaces;
state_interfaces.type =
interface_configuration_type::INDIVIDUAL;
state_interfaces.names = imu_sensor_-
>get_state_interface_names();
```



Why ros2_control is better than ros_control?

- ros2_control surpasses ros_control in flexibility
- Reuse of already existing hardware components
- Controller chaining
- Ability to run controllers with different update rate
- Ability to choose the components to run asynchronously
- Semantic components to wrap data with semantic meaning

PAL ROBOTICS

ROS 2 - Practical session

Practical session - Overview

- **Tutorial 0: Docker tutorial**
  - Create and launch Agimus container
  - Setup ROS 2 environment
- **Tutorial 1: Joint subscriber**
  - Create a subscriber
  - Print joint states
- **Tutorial 2: Play motion2 client**
  - Create a action client
  - Create a custom motion
- **Tutorial 3: Change controller service client**
  - Create a service client
  - Change controllers of TIAGo

Questions?