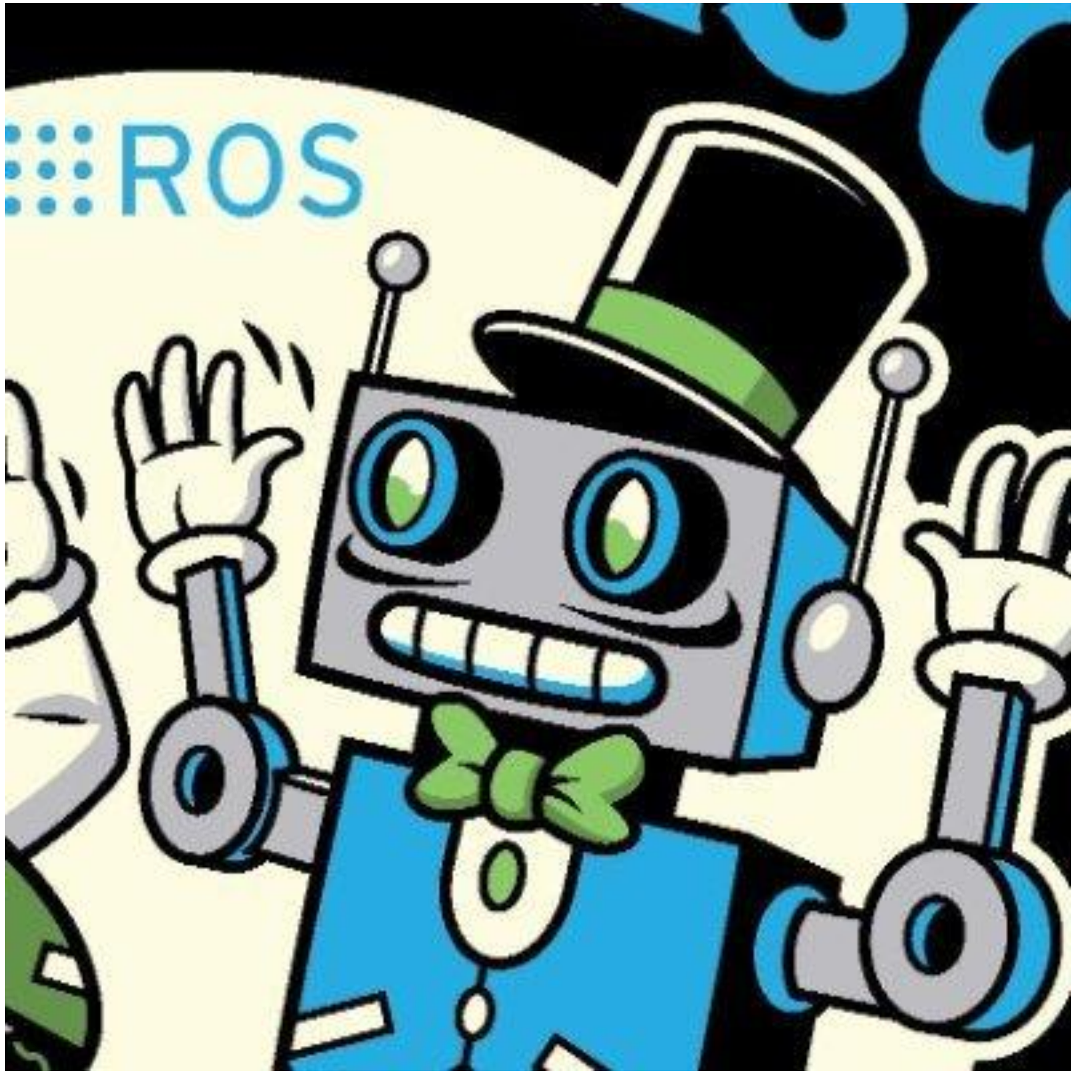
Robotics Operating System

Basic principles, use cases, and examples

DV Autonomous Driving System Dedicated Vehicle
FORMULA STUDENT
DJ RACECARS
Centre for Connected Autonomous Vehicles

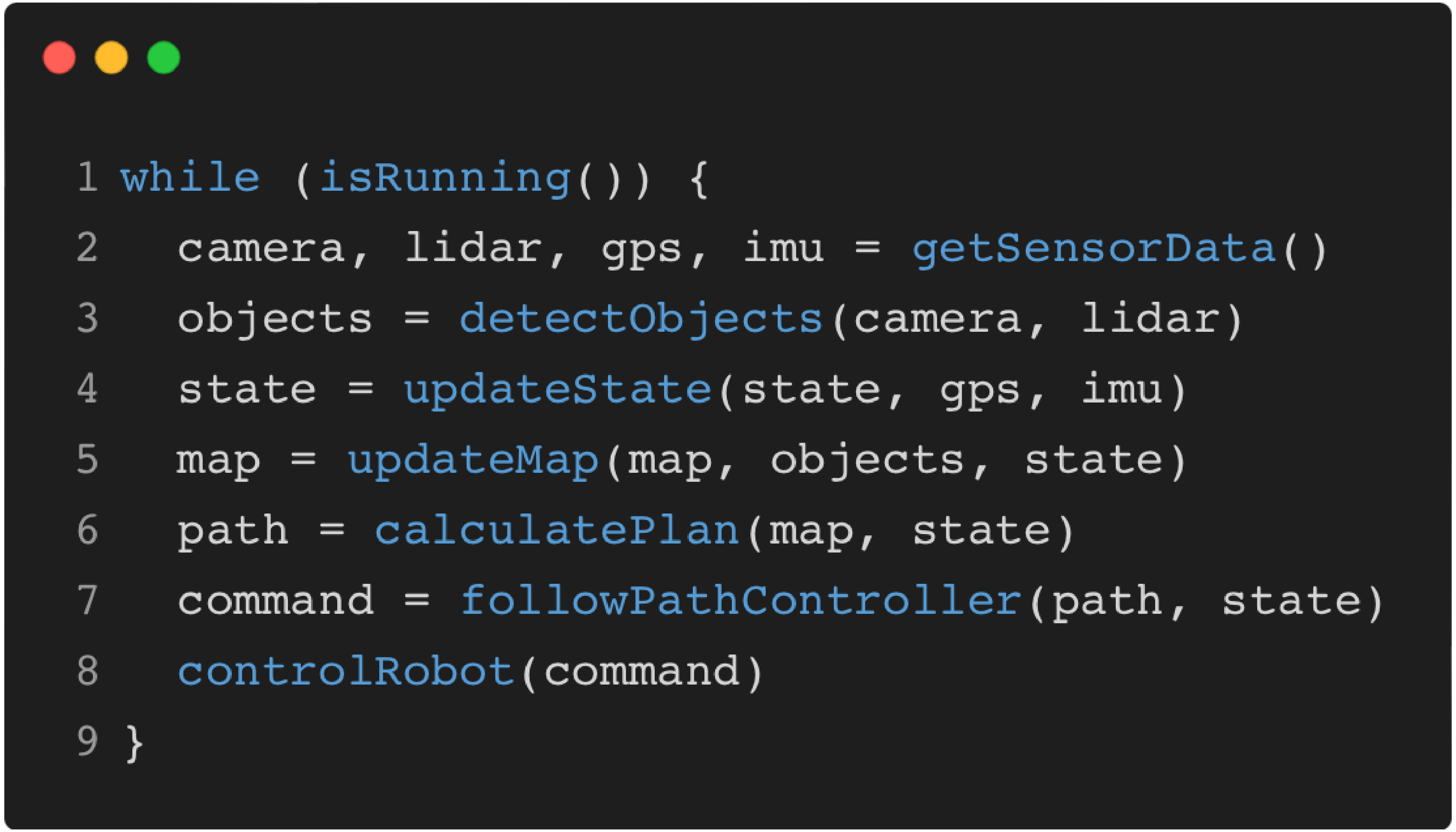
Overview

- Naive robot software
- What is ROS?
- How does ROS2 work?
- Basic features and tools
- Other useful features and tools



Naïve Robot Software

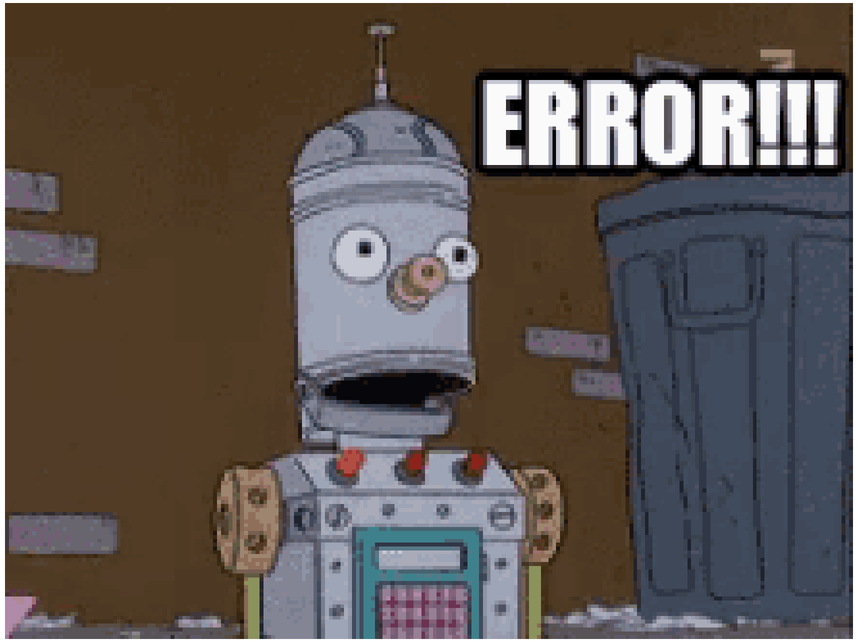
Naïve Robot Software Example



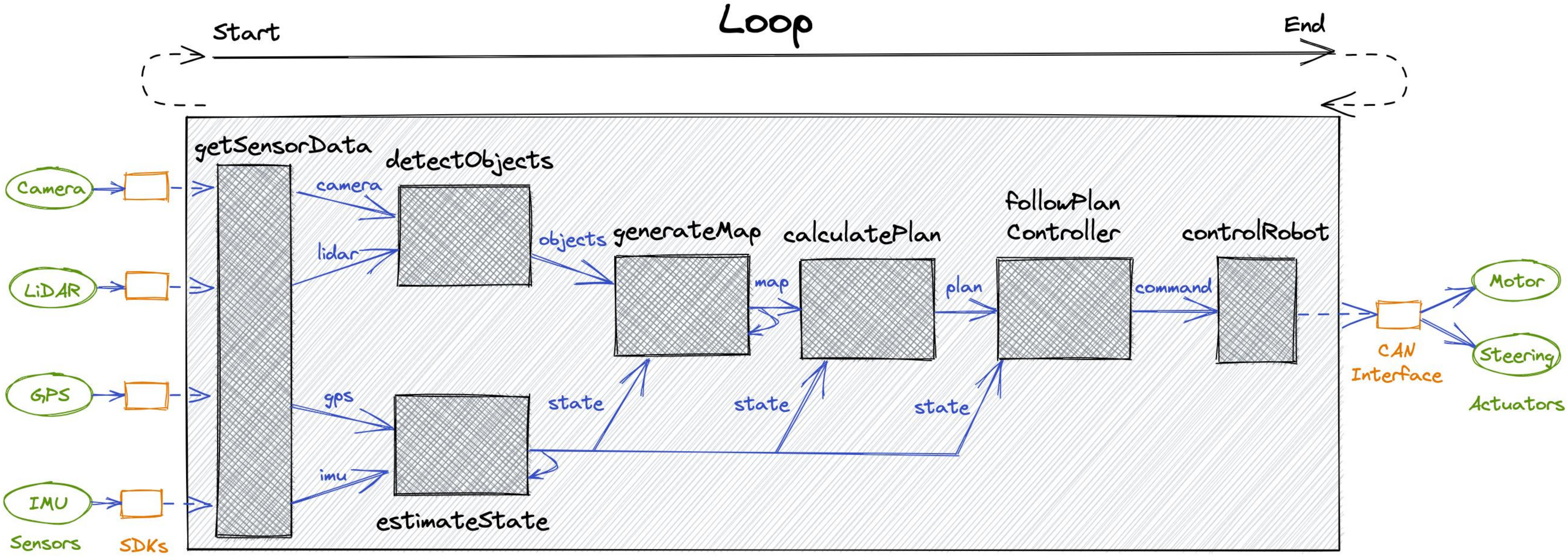
```
while (isRunning()) {
  camera, lidar, gps, imu = getSensorData()
  objects = detectObjects(camera, lidar)
  state = updateState(state, gps, imu)
  map = updateMap(map, objects, state)
  path = calculatePlan(map, state)
  command = followPathController(path, state)
  controlRobot(command)
}
```

Problems/Challenges

- Single process (synchronous)
- Single program
- Difficulty in separation of concerns
- Develop custom tools for:
  - Visualization
  - Simulation
  - Managing configuration
  - Hardware interfaces



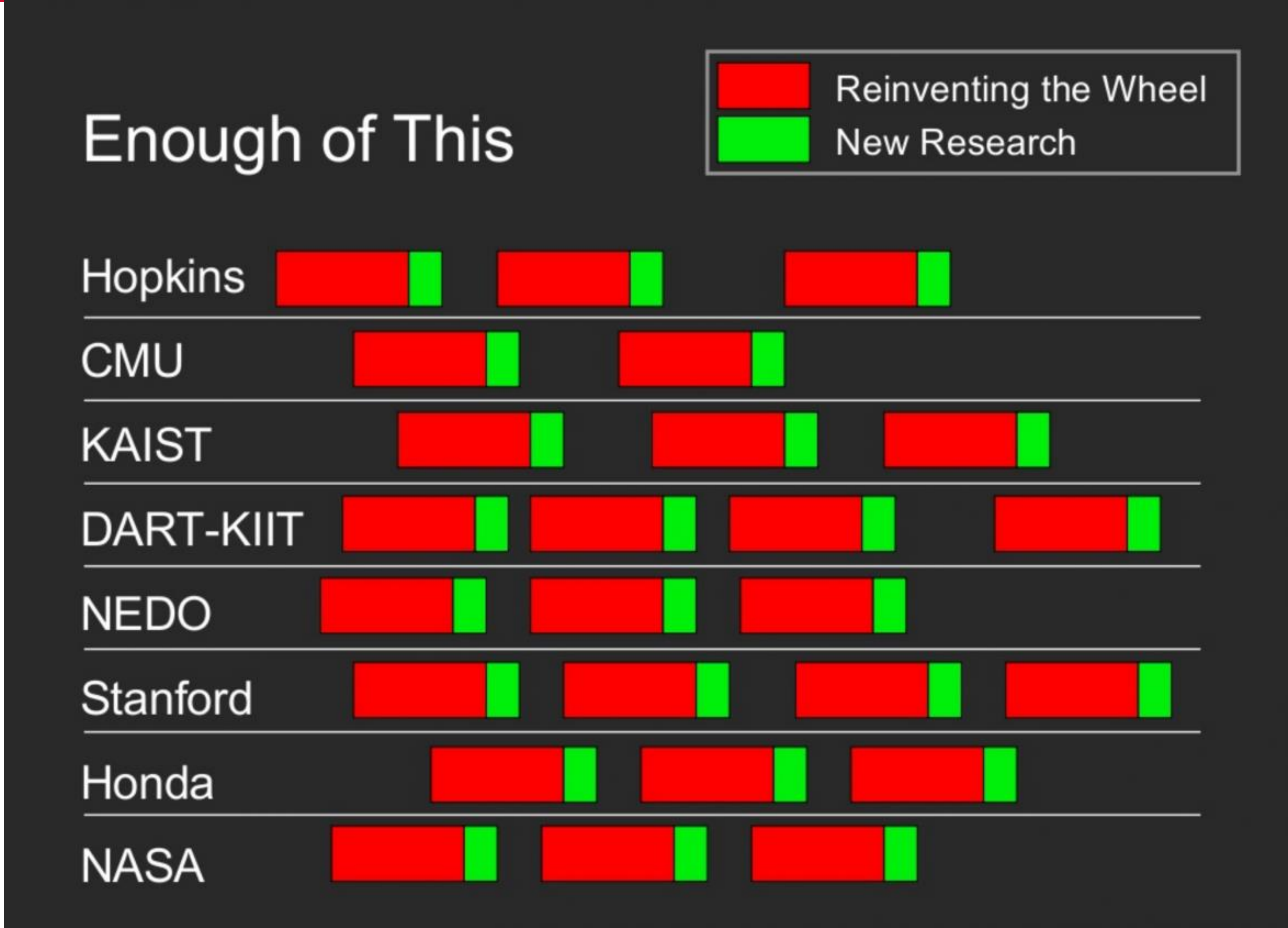
Naïve Robot Software Example



Reinventing the Wheel

Around year 2000

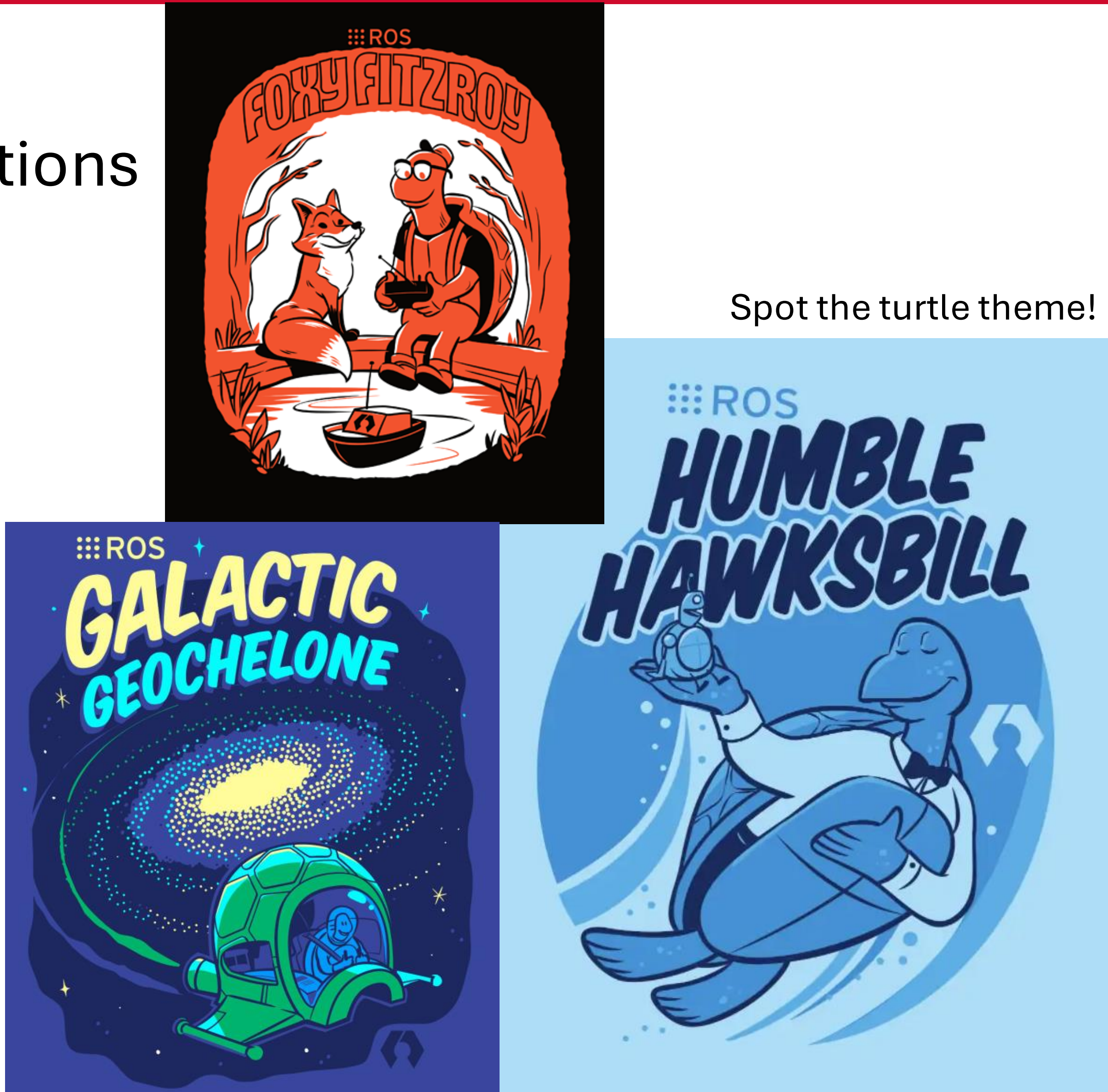
- **Too much time** reinventing the wheel
- **Too little time** new research



What is ROS?

ROS is…

- Infrastructure to build robot applications
- Framework to allow communication between processes
- Many additional features and tools around this
- Open-source
  - Maintained by Open Robotics



Spot the turtle theme!

ROS Solves Problems/Challenges

- Single process (synchronous)
- Single program
- Difficulty in separation of concern
- Develop custom tools for:
  - Visualization
  - Simulation
  - Managing configuration
  - Hardware interfaces

☑ Inter-processes communication

☑ Distributed processing

☑ Encourages clear interfaces

☑ Many existing tools!

- ☑ RViz2
- ☑ Gazebo
- ☑ ROS Parameters
- ☑ ROS Drivers

Community

- **ROS Index**
  - Aims to be the *definitive index* of all ROS Software

7612 Packages

2665 Repositories

(as of 4th Oct 2023)

EUFS

Industry-wide Standard

- Used by many companies
- If you want to get into robotics, ROS (2) is almost mandatory

- **ROS-Industrial**

How Does ROS2 Work?

Middleware

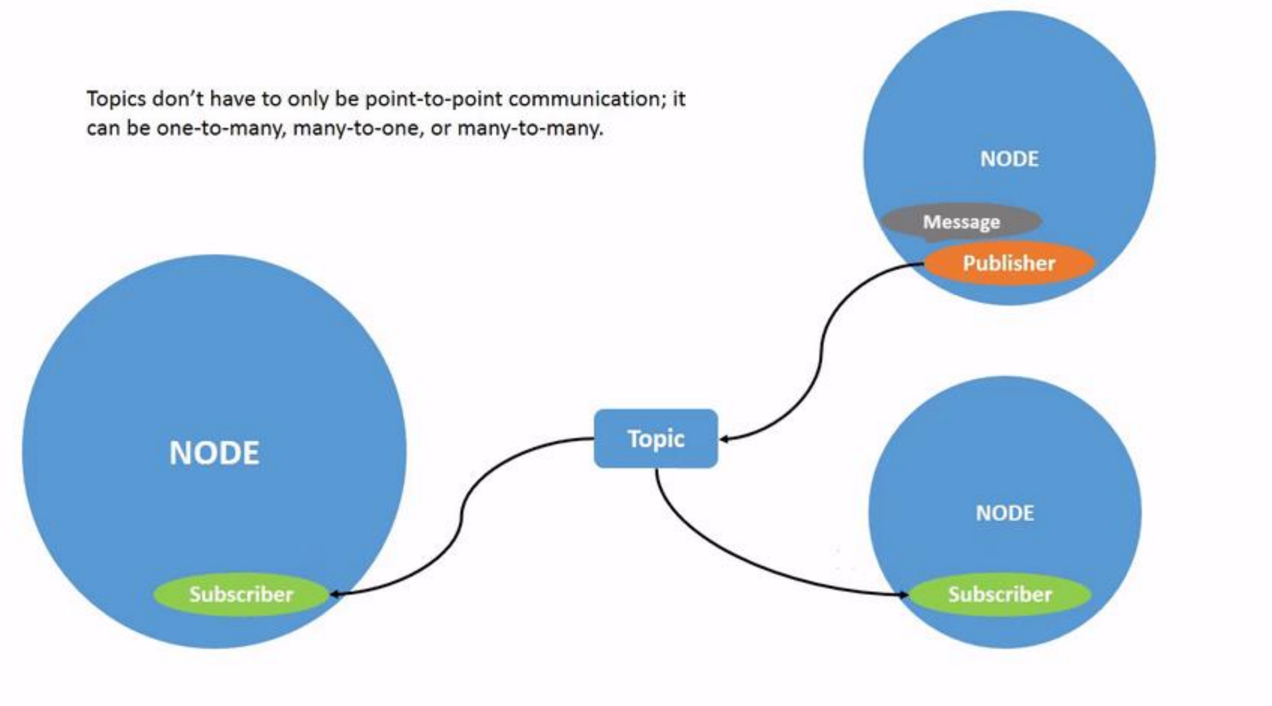
- ROS 2 is a **middleware**
  - Layer between Operating System and Applications
  - “Software glue”

- Uses a publish/subscribe mechanism

Graph Concept

- **ROS Graph**: Network of nodes in a ROS system and the connections between them by which they communicate

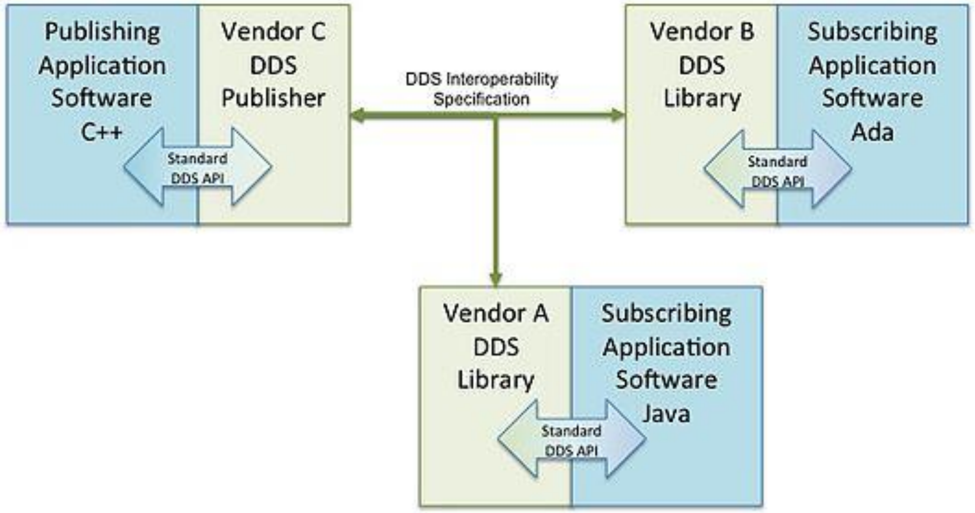
- Concepts
  - **Node**
  - **Message**
  - **Topic**
  - **Discovery**



Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.

Data Distribution Service (DDS)

- Standard enabling data exchanges using the publish-subscribe pattern
  - Dependable
  - High Performance
  - Interoperable
  - Real-time
  - Scalable


Publishing Application Software C++
Vendor C DDS Publisher
Standard DDS API
DDS Interoperability Specification
Vendor B DDS Library
Subscribing Application Software Ada
Standard DDS API
Vendor A DDS Library
Subscribing Application Software Java
Standard DDS API

- Many different implementations

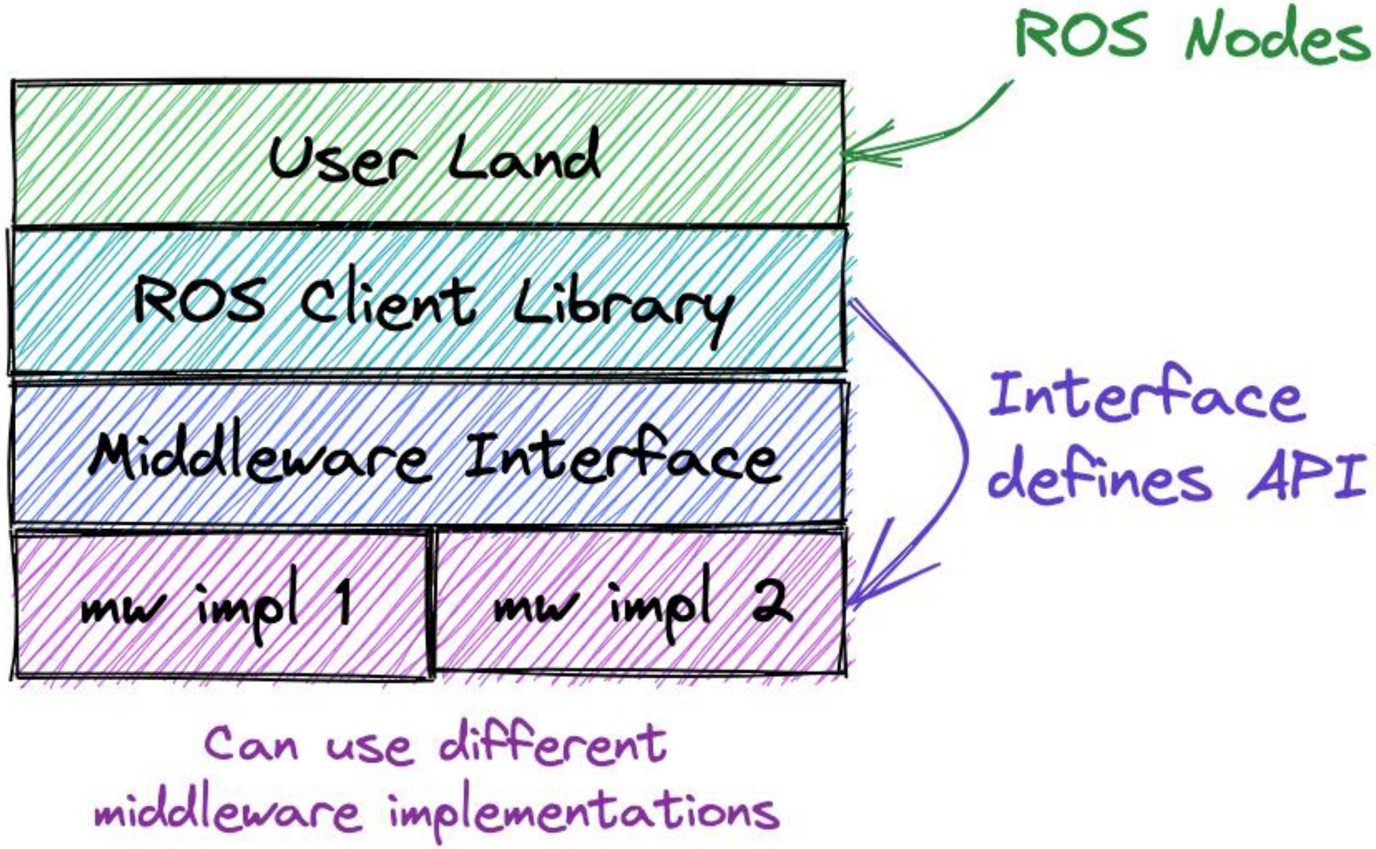
- Maintained by Object Management Group

Client Libraries

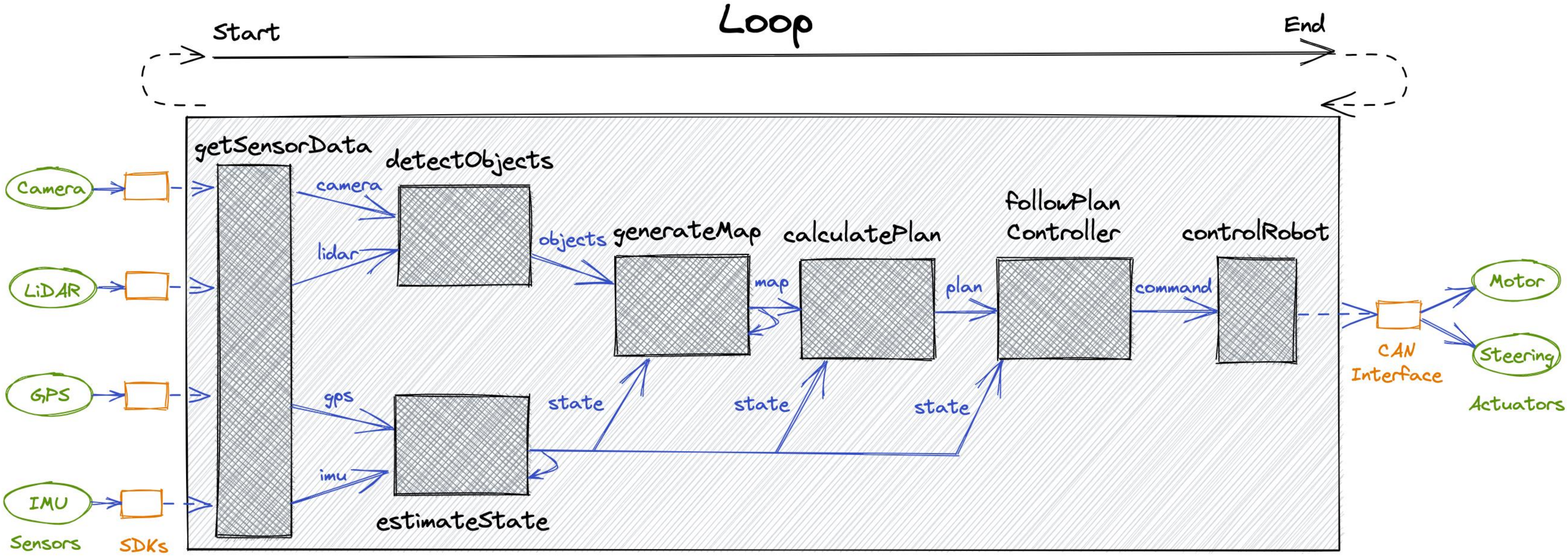
- Allow nodes written in different programming languages to communicate

- 2 client libraries maintained by the ROS 2 team
  - C++ - **rclcpp**
  - Python - **rclpy**
- Community maintained client libraries
  - Ada, C, JVM, .NET, Node.js, and Rust

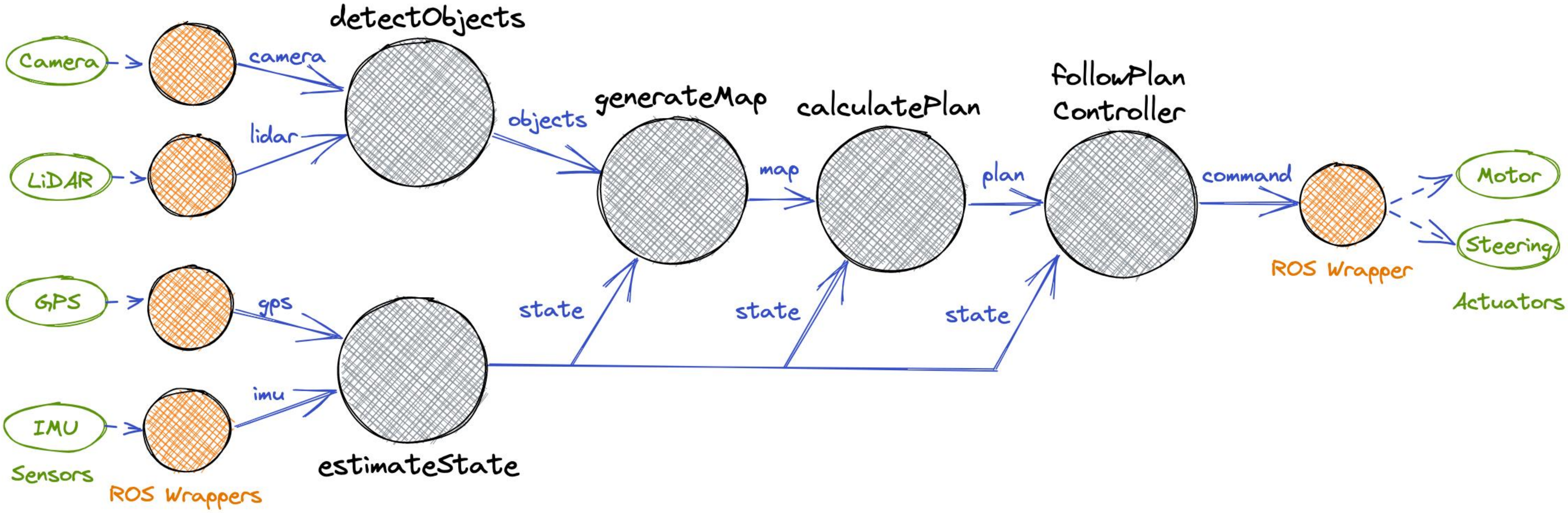
Middleware Details



Naïve Robot Software Example



ROS Robot Software Example



Basic Features - Tools

Workspace

- Directory containing ROS 2 packages

- Sub-directories
  - `build` - output from build
  - `install` - workspace’s setup files
  - `log` - logs from build
  - `src` - packages

```
<workspace_folder>/
    build/
        ...
    install/
        ...
    log/
        ...
    src/
        ...
```

Creating Packages

Python:

```
ros2 pkg create --build-type ament_python <package_name>
```

- With node:
  ```
  ros2 pkg create --build-type ament_python --node-name my_node my_package
  ```

C++:

```
ros2 pkg create --build-type ament_cmake <package_name>
```

- With node:
  ```
  ros2 pkg create --build-type ament_cmake --node-name my_node my_package
  ```

Packages

```
<workspace_folder>/
    src/
        <package_1>/ (Python)
            setup.py
            setup.cfg
            package.xml
            resource/<package_1>
            <package_1>/
                __init__.py
                <node>.py
```

```
        ...
        <package_2>/ (C++)
            CMakeList.txt
            package.xml
            include/<package_2>/
                <node>.hpp
            src/
                <node>.cpp
```

Build System

- `colcon` - command line tool to improve the workflow of building, testing and using multiple software packages
  - `colcon build` - build workspace
  - `colcon test` - run tests in workspace

- Useful flags
  - `--symlink-install` - uses ‘symlinks’ instead of copying files
  - `--continue-on-error` – Continue other packages when package fails
  - `--packages-select` - Build only specific packages

Sourcing

IMPORTANT

- Before using ROS 2 in terminal, **source** your ROS 2 installation workspace

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

- “Overlay” – Secondary workspace with additional packages

```
. install/local_setup.bash
```

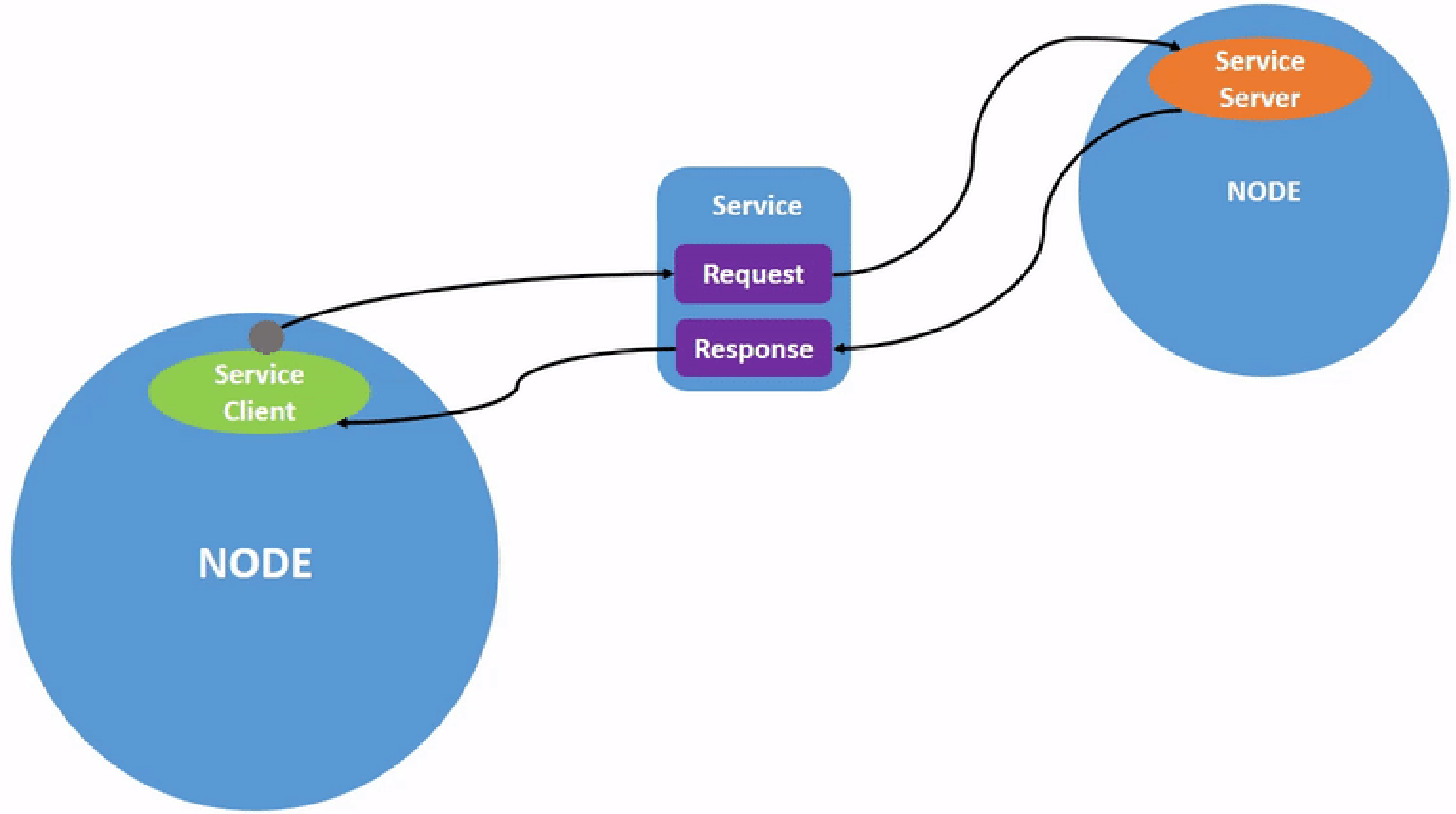
- “Underlay” – Workspace containing dependencies of packages in overlay
- Packages in “overlay” will override packages in the “underlay”

ROS2 CLI

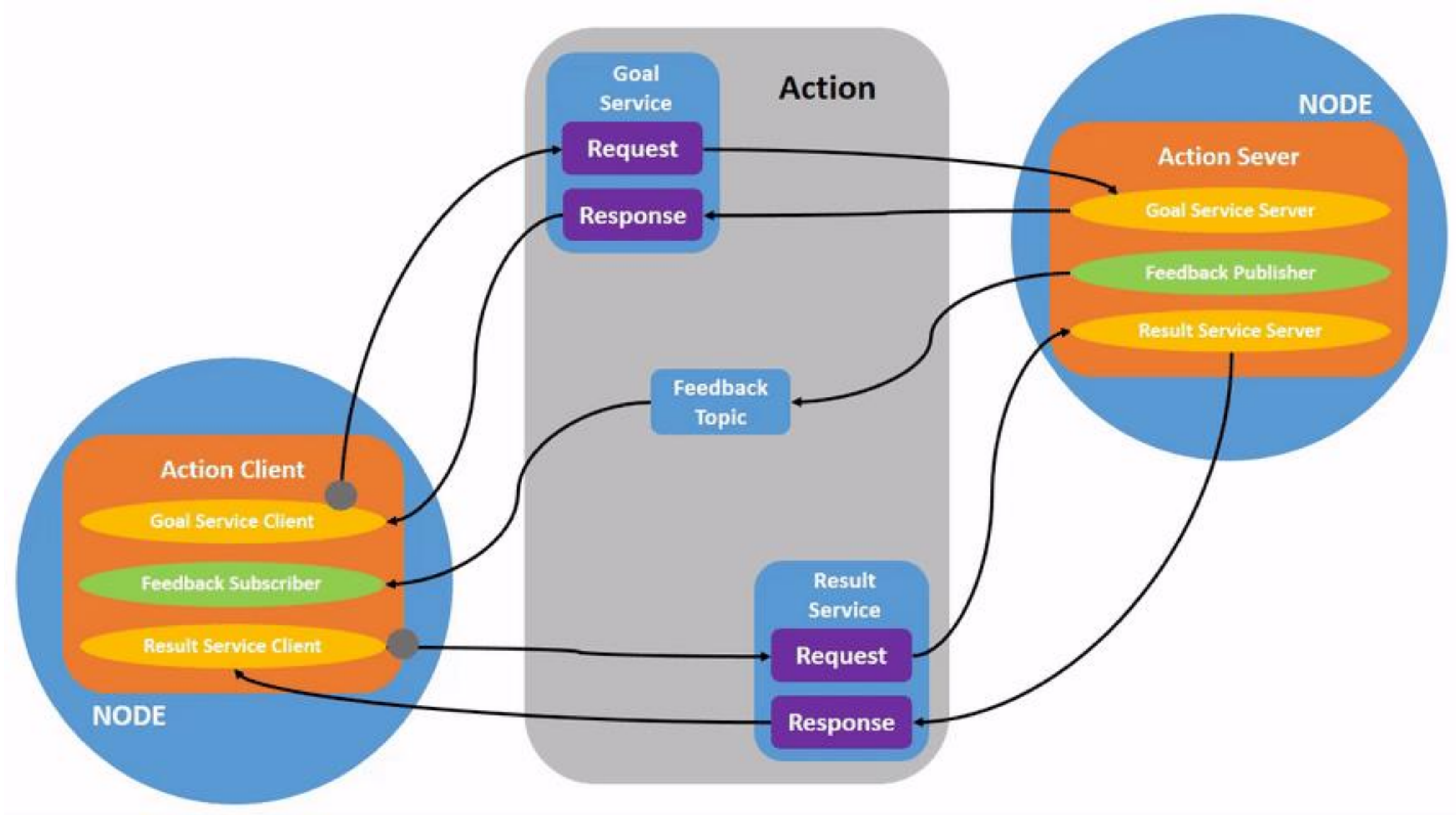
- `ros2`
    - `launch` – Allows running launch files
    - `node` – Display information about node
    - `param` – Allow manipulating parameters
    - `pkg` – Create package or get information about package
    - `run` – Allows running executable
    - `test` – Run launch tests
    - `topic` – Display debug information about topic
    - `wtf` – (where's the fire, alias for `doctor`)
      Check ROS setup and other potential issues

Other Useful Features

Service


Service
Server
NODE
Service
Request
Response
Service
Client
NODE

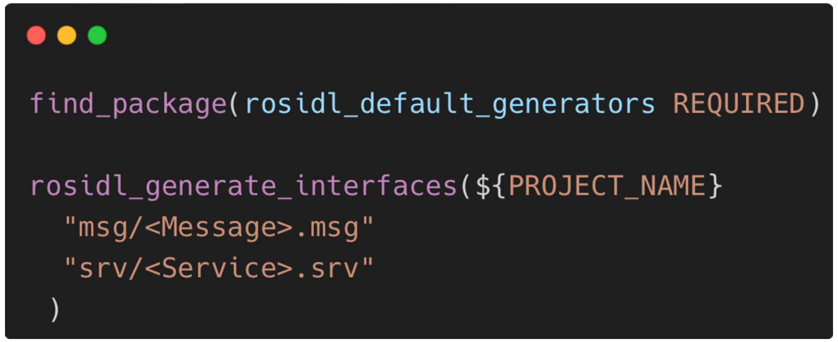
Action


Goal
Service
Request
Response
Action
NODE
Action Sever
Goal Service Server
Feedback Publisher
Result Service Server
Feedback
Topic
Action Client
Goal Service Client
Feedback Subscriber
Result Service Client
NODE
Result
Service
Request
Response

Custom Messages and Services

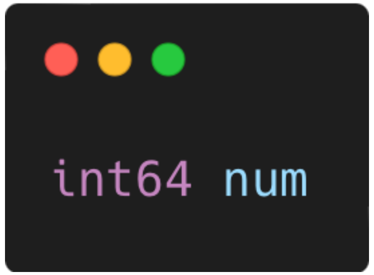
```
<workspace_folder>/
    src/
        <msg_package>/ (C++)
            CMakeList.txt
            package.xml
            msg/
                <Message>.msg
            srv/
                <Service>.msg
```

CMakeList.txt



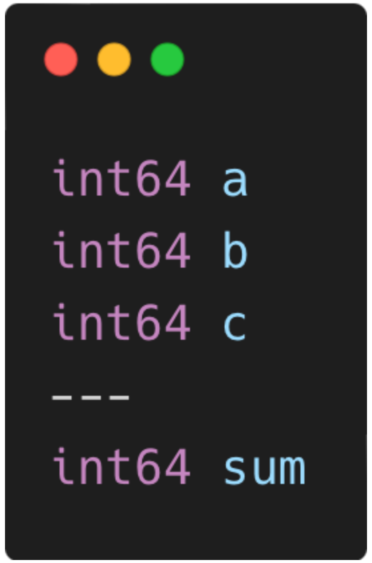
```
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/<Message>.msg"
  "srv/<Service>.srv"
 )
```



```
int64 num
```

<Message>.msg



```
int64 a
int64 b
int64 c
---
int64 sum
```

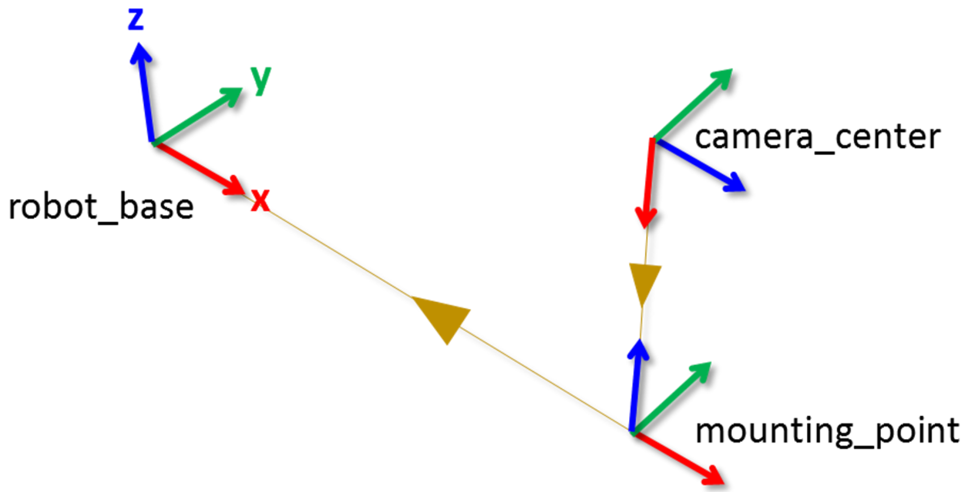
<Service>.msg

TF

- TF maps different coordinate frames
- **Transforms** are stored in a tree structure

- Broadcasters sends transforms
- Listeners receives transforms
  - Stored in buffer
  - Provides functions to perform transformations

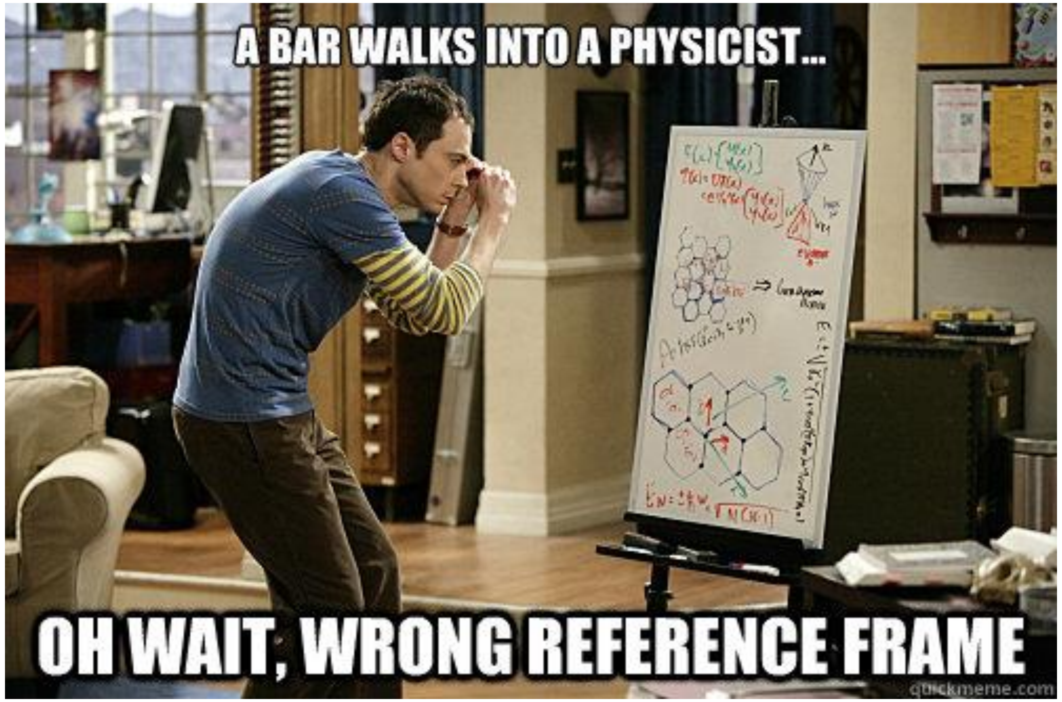
- Can be displayed in RViz2



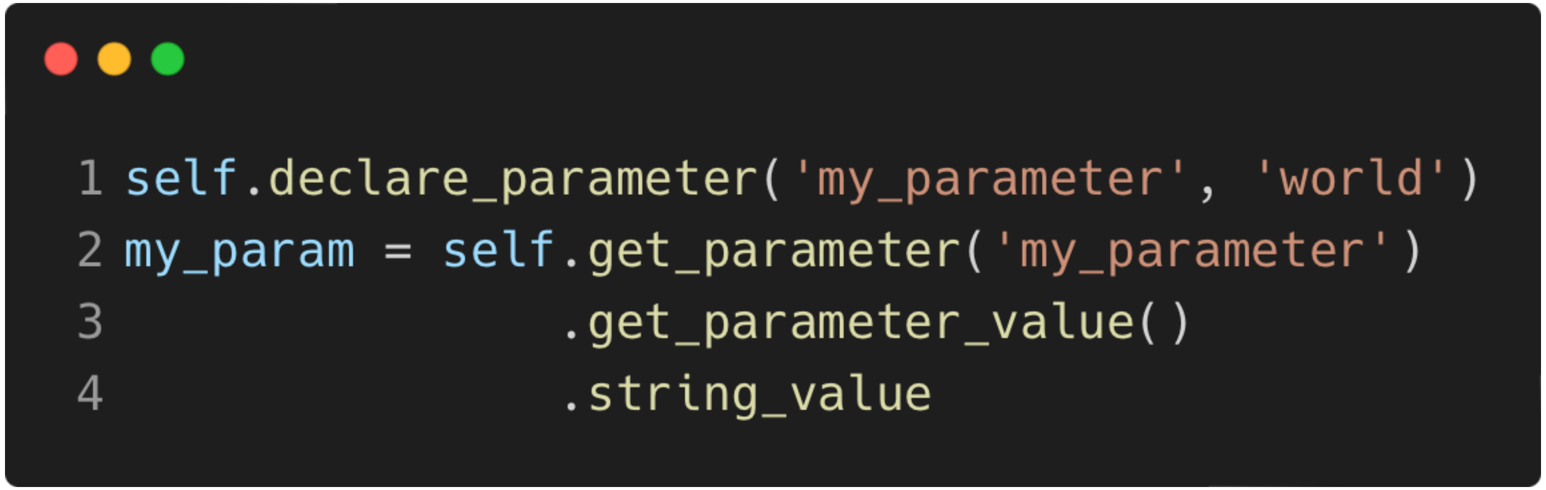
ROS Frames

Transform Conventions

- The *map* to *odom* transform is provided by a positioning system (particle filter, SLAM)
- The *odom* to *base_link* transform by an odometry system.


A BAR WALKS INTO A PHYSICIST...
OH WAIT, WRONG REFERENCE FRAME
quickmeme.com

Parameters



```python
self.declare_parameter('my_parameter', 'world')
my_param = self.get_parameter('my_parameter')
               .get_parameter_value()
               .string_value
```

- Can be set using
  - Command line arguments when starting
  - In launch file
  - Command line while running

Types

`bool`
`int64`
`float64`
`string`
`byte[]`
`bool[]`
`int64[]`
`float64[]`
`string[]`

Remappings

- Parameters could be used to set topic names
- **Alternative**: Remapping topic name for node

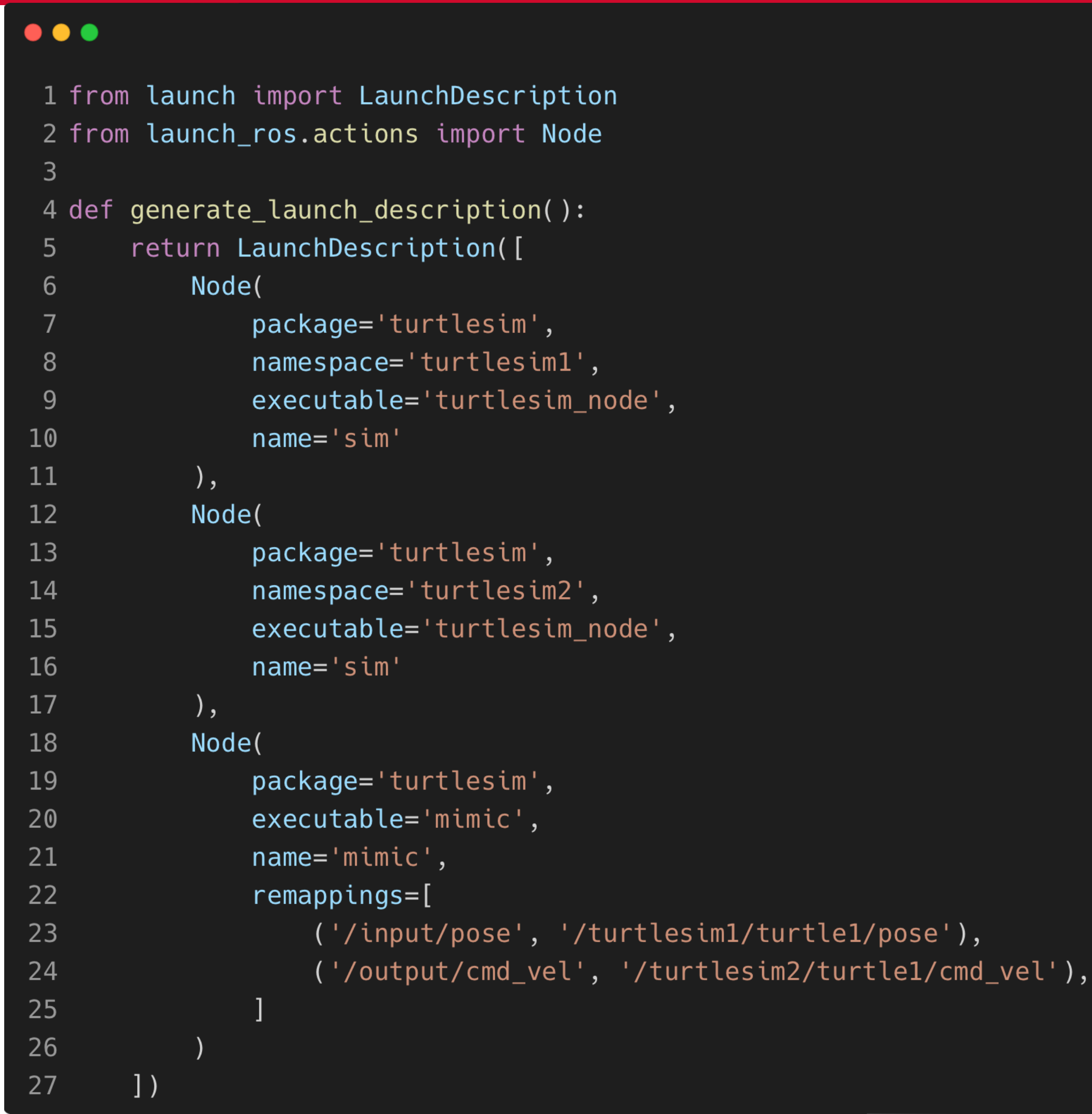
- Some other use cases:
  - Change the default namespace
  - Change the node name
  - Remap topic and service names separately

- Purpose: Allows reuse of same node in different parts

Launch

- Way to start multiple nodes at the same time
- Launch files written in **Python**, XML, or YAML

- `launch_ros` - Provides framework for launch file in different formats
    - Uses the ROS-independent `launch` framework underneath



```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            namespace='turtlesim2',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        )
    ])
```

Lifecycle Nodes

- Failed nodes can be re-configured and relaunched

- Used in Nav2 – more on that later
- Activates nodes in a specific sequence

```
ros2 lifecycle set /lifecycle_node configure
```

Rosbags

- `ros2 bag` - command line tool for recording data published on topics

- Commands
  - `ros2 bag record <topic_name>`
  - `ros2 bag info <bag_file_name>`
  - `ros2 bag play <bag_file_name>`

Other Useful Tools

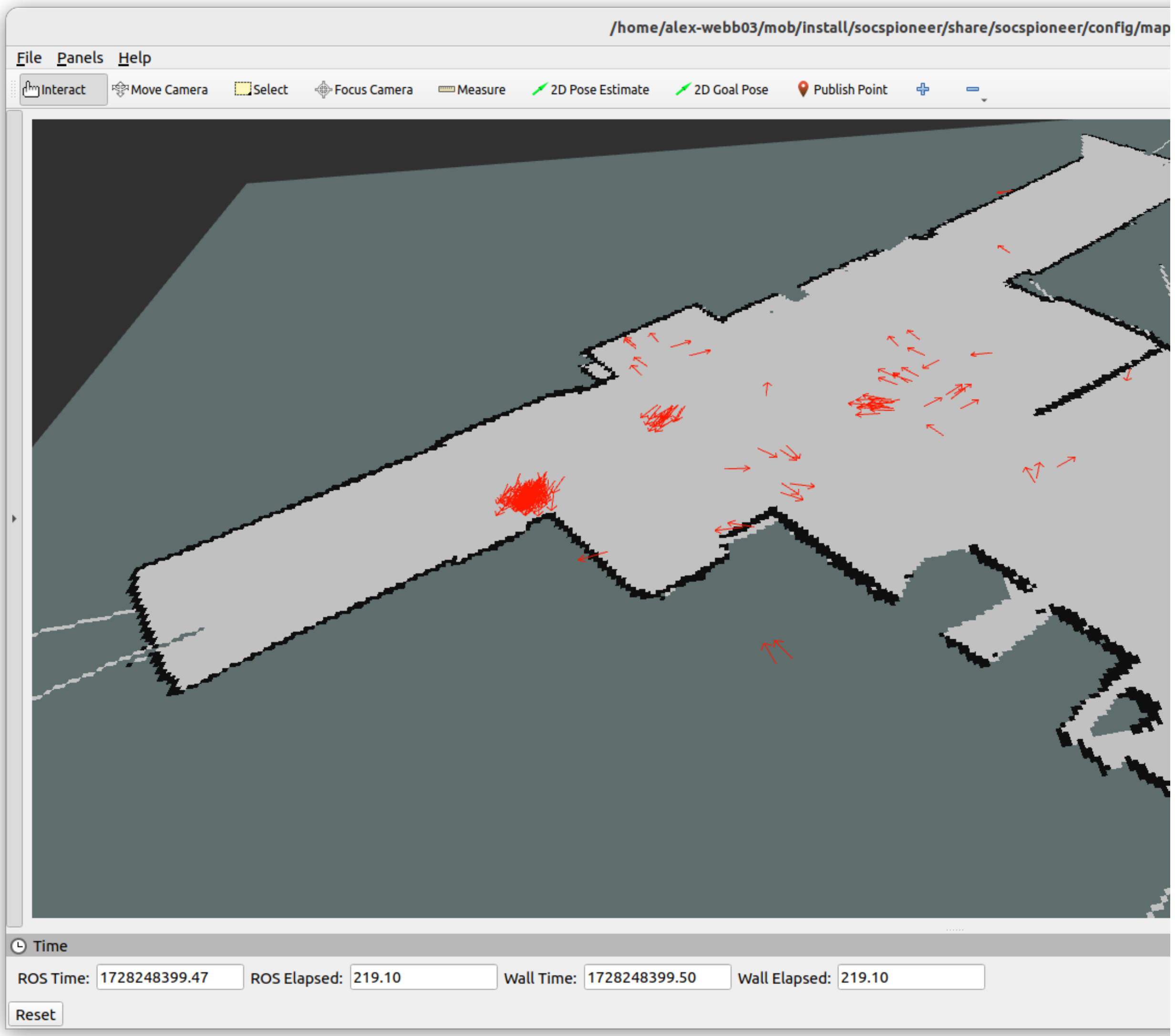
RViz2

Visualization tool

- Select topics to subscribe to
  - Visualizes message contents
  - Only specified message types

- Config files - Save and share layouts

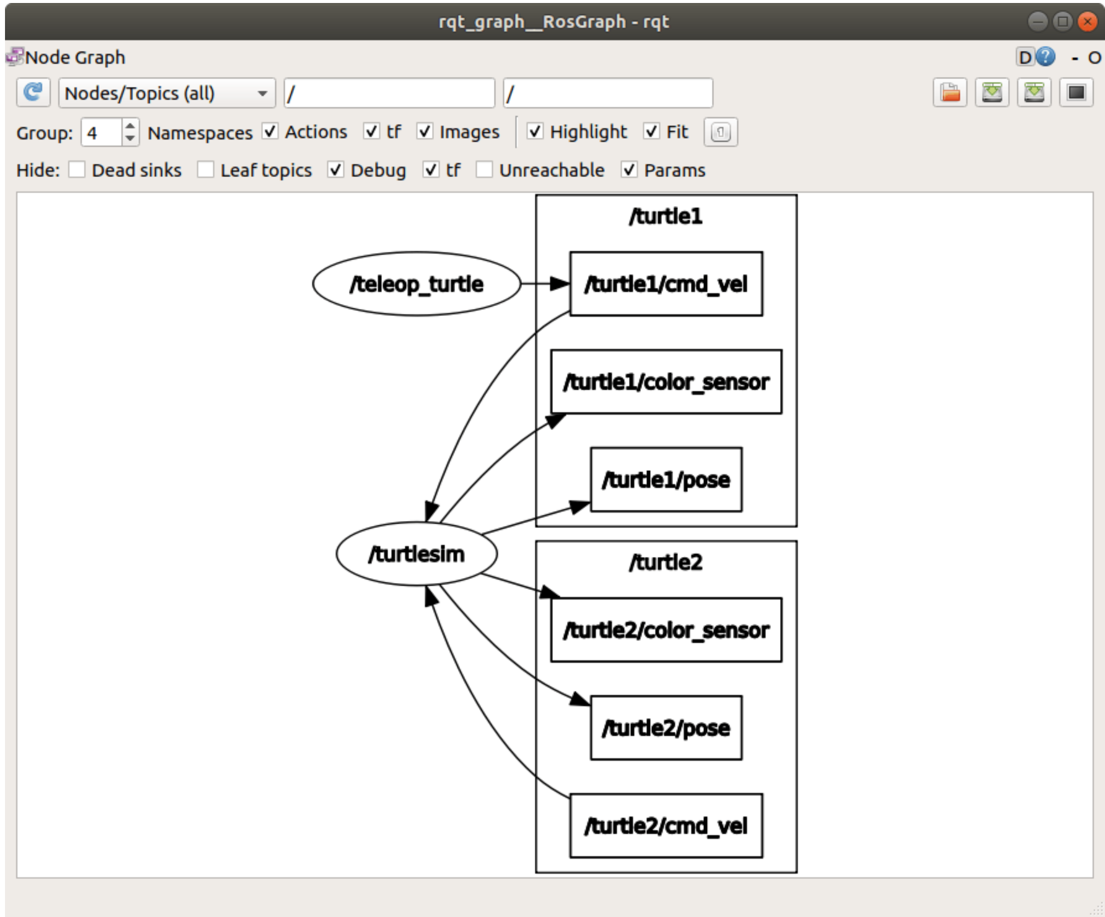
- Plugins - Add custom features
  - Display custom message types


/home/alex-webb03/mob/install/socspioneer/share/socspioneer/config/map

ROS Time: 1728248399.47 ROS Elapsed: 219.10 Wall Time: 1728248399.50 Wall Elapsed: 219.10

RQT

- Graphical User Interface (GUI) framework
  - tools and interfaces in the form of plugins
- RQT is built on top of the open-source QT framework
- Easy to manage many windows in single screen layout
- Many existing GUIs
  - Configuration, logging, topics, services, actions, and introspection



rqt_graph__RosGraph - rqt

Node Graph

Nodes/Topics (all) | / | /

Group: 4 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

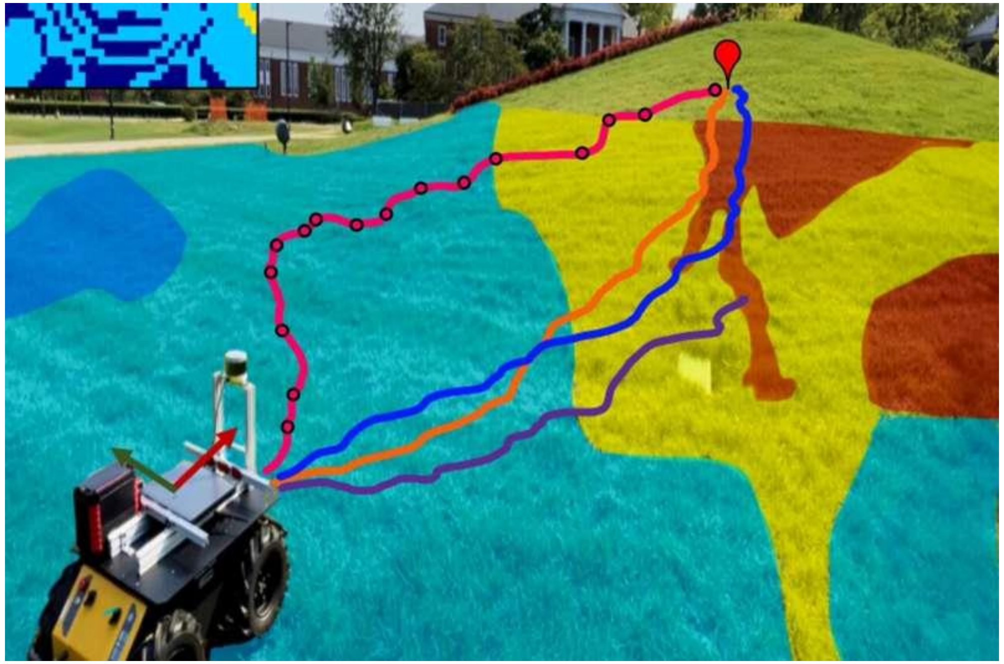
Hide: ☐ Dead sinks ☐ Leaf topics ☑ Debug ☑ tf ☐ Unreachable ☑ Params

NAV2 Stack

Mobile robot navigation tool

- Modular architecture

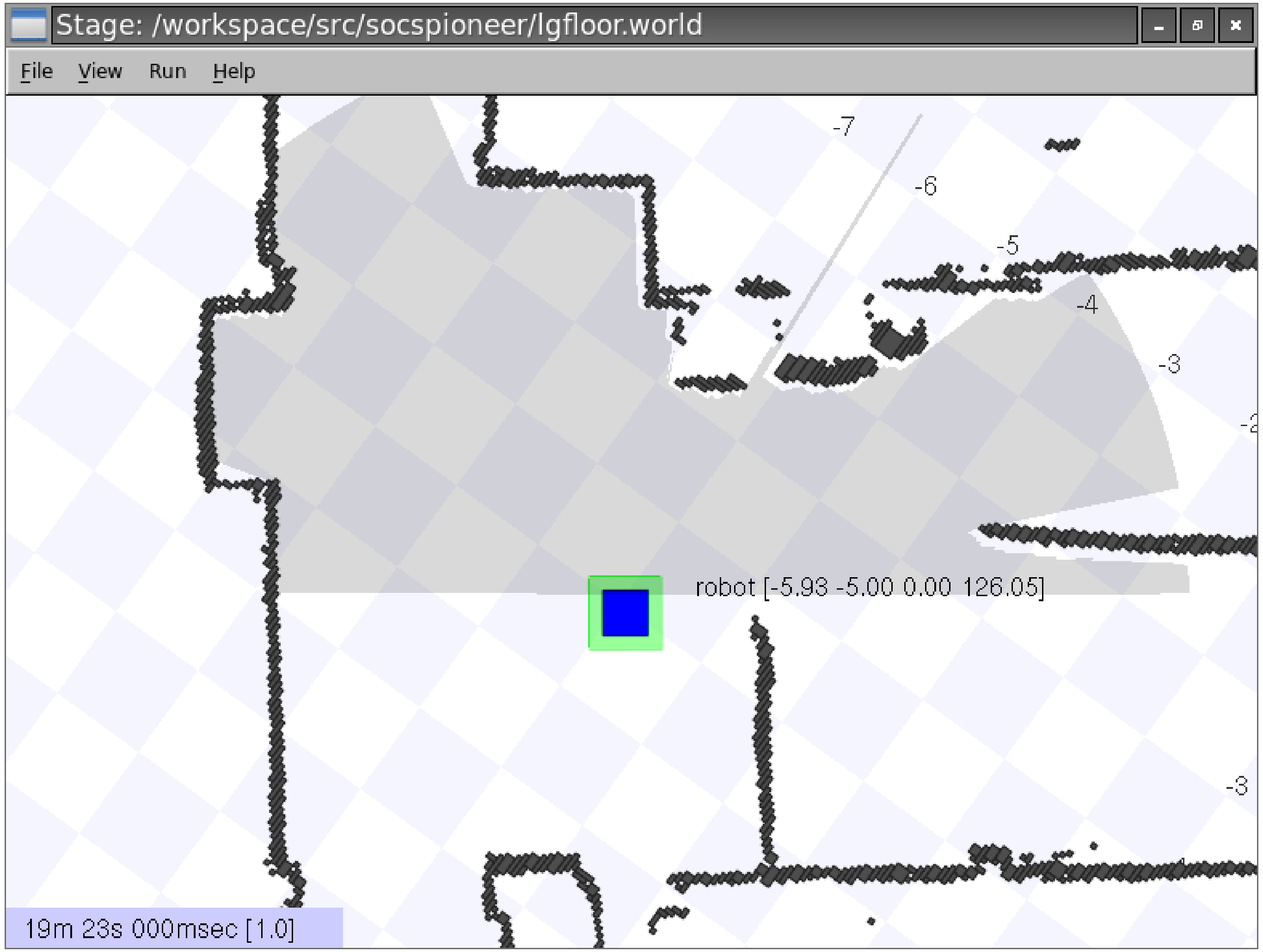
- Map Server reads a map from a file and publishes it
- Adaptive Monte-Carlo Localizer (AMCL) takes map and finds the robot's location in the map
- Lifecycle Manager orders nodes



Stage Simulator

- **Basic 2D simulator**

- Load custom maps
- Move robot around
- See laser scanner


Stage: /workspace/src/socspioneer/lgfloor.world
File
View
Run
Help
-7
-6
-5
-4
-3
-2
-3
robot [-5.93 -5.00 0.00 126.05]
19m 23s 000msec [1.0]

DEMO!

Limitations

- Overly complicated for simple projects

- Some libraries are not mature
- Poor and non-existent documentation
- Awkward API in client libraries

Scalability

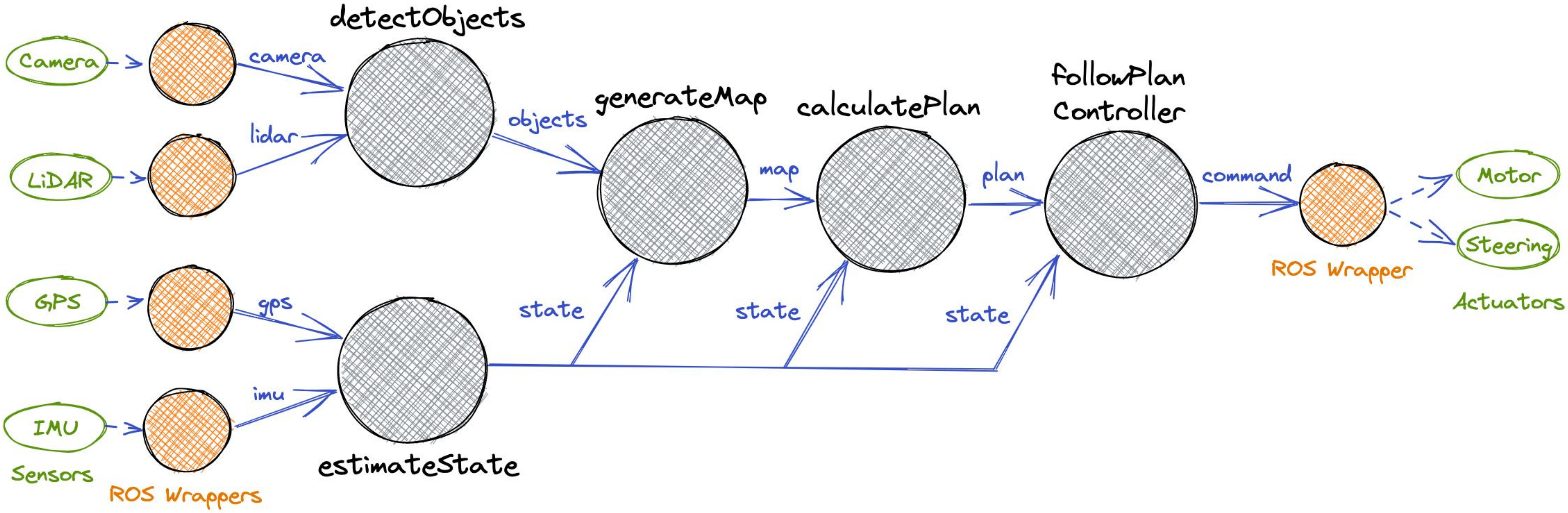
- Difficult to diagnose performance issues

Common Problems

- Serial Lidar processing
- Network bandwidth between computers
- Inter-process communication is up to DDS vendor

Conclusion

Graph Concept



ROS Workflow

1. Create a workspace
2. Create a package
3. Write code for node(s)
4. Write launch file
5. Build the workspace in terminal
6. Run the launch file/node
    1. Open a NEW terminal! (different to terminal code is built)
    2. Use ROS2 CLI

Tips

- **Remember to source ROS 2 workspace!**
- When using a search engine, make sure to type is ROS 2
  - Ensures you do not get ROS 1 documentation
    - http://wiki.ros.org – ROS 1
    - https://docs.ros.org – ROS 2
  - Ensure correct ROS 2 release
    - https://docs.ros.org/en/humble - ROS 2 humble (in English)
- Google message type to get definition
  - For example: “std_msgs header ros 2”

Useful Resources

- ROS 2 Documentation:
  - https://docs.ros.org/en/humble/index.html
- rclpy Documentation:
  - https://docs.ros2.org/humble/api/rclpy/index.html
- rclcpp Documentation:
  - https://docs.ros2.org/humble/api/rclcpp/index.html
- ROS Tutorials Source Code:
  - https://github.com/ros/ros_tutorials
- ROS 2 Cheat sheets (colcon and ROS CLI):
  - https://github.com/ubuntu-robotics/ros2_cheats_sheet