3rd December
2025

ROS2 WORKSHOP

Ignacio Dassori W.

WHAT WE'LL SEE

01. What is ROS?

02. ROS vs ROS2

03. Nodes & Topics

04. Messages

05. CLI Tools

07. Services & Actions

07. Packages

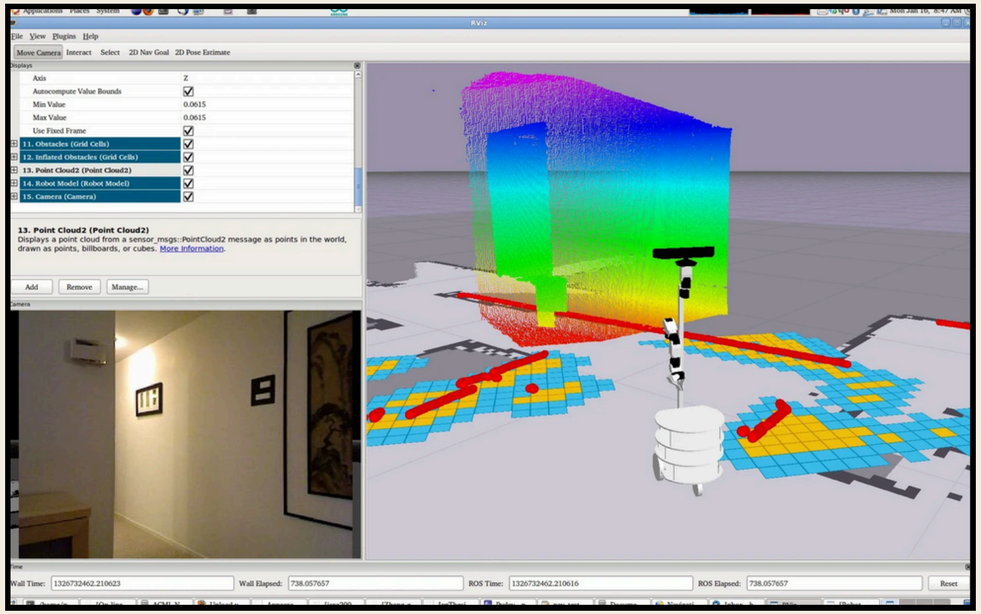
08. Launch files

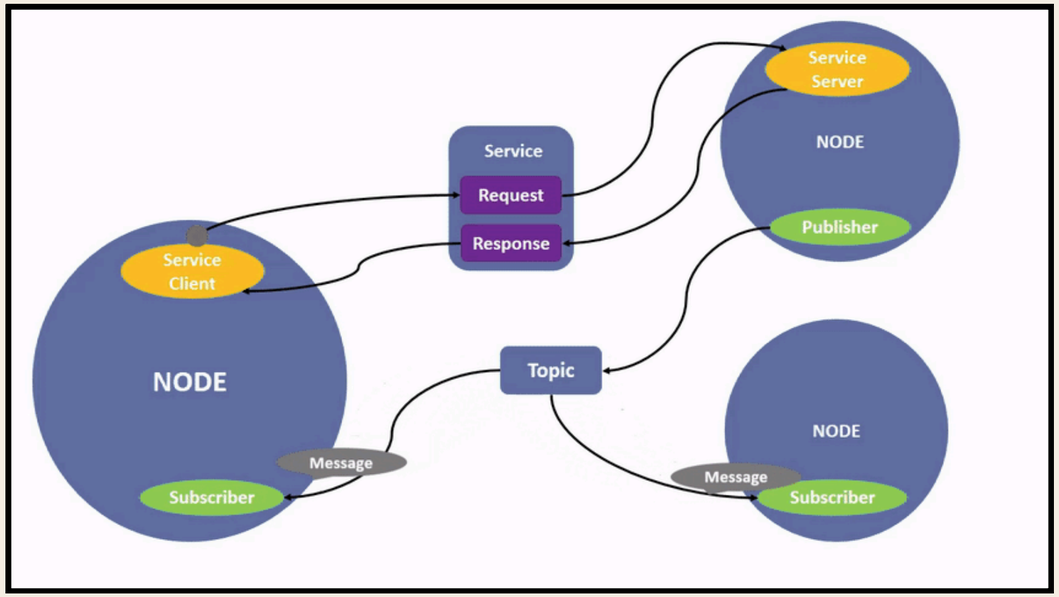
09. Rviz2

10. Gazebo example

WHAT IS ROS

- The **Robot Operating System (ROS):** Set of software libraries and tools for building robotics applications.
- **NOT** an operating system (OS). Open source project, provides standardized way of developing and distributing robotic software.
- First developed by PhD students at **Stanford University**.





ROS provides the user with many features:

1) Communication through graph of nodes

2) Package management

3) Hardware abstraction

4) Simulation tools

5) Visualization tools

6) Huge community!

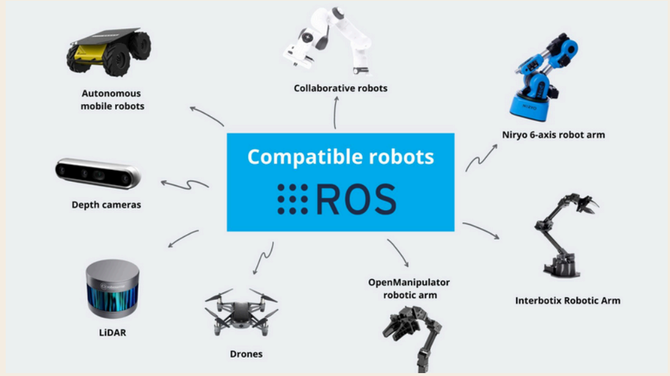
7) Many implemented algorithms / drivers

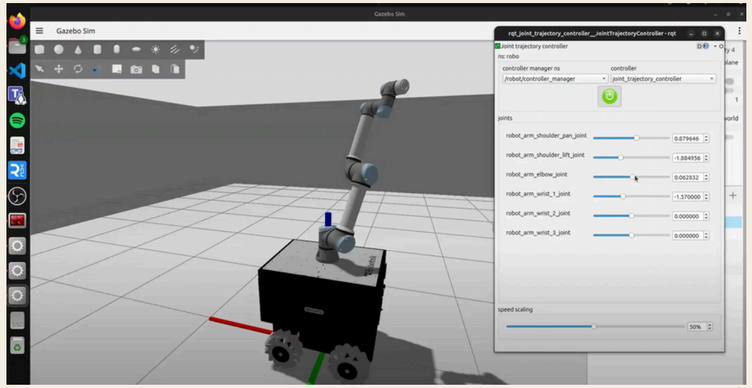
ROS DISTROS

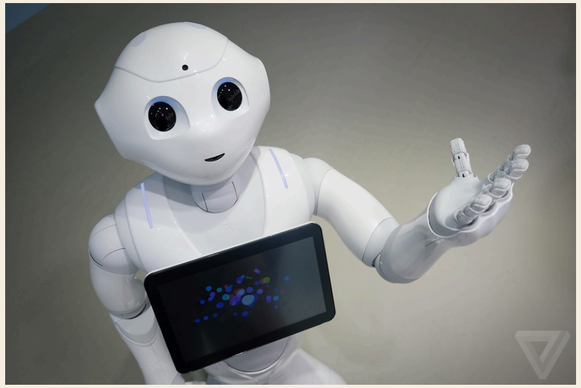
UBUNTU 22.04

UBUNTU 24.04

WHERE CAN IT BE USED

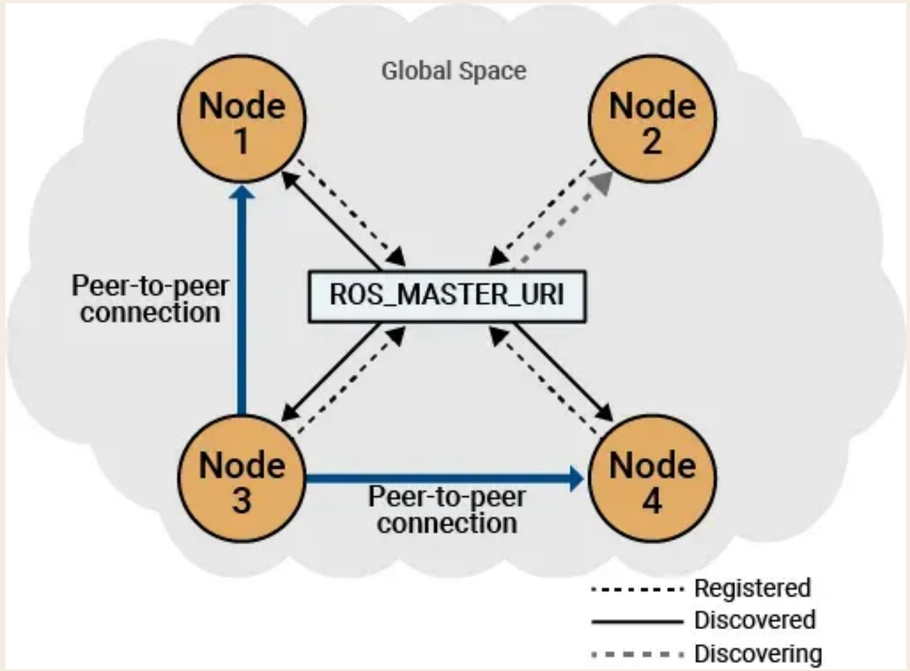




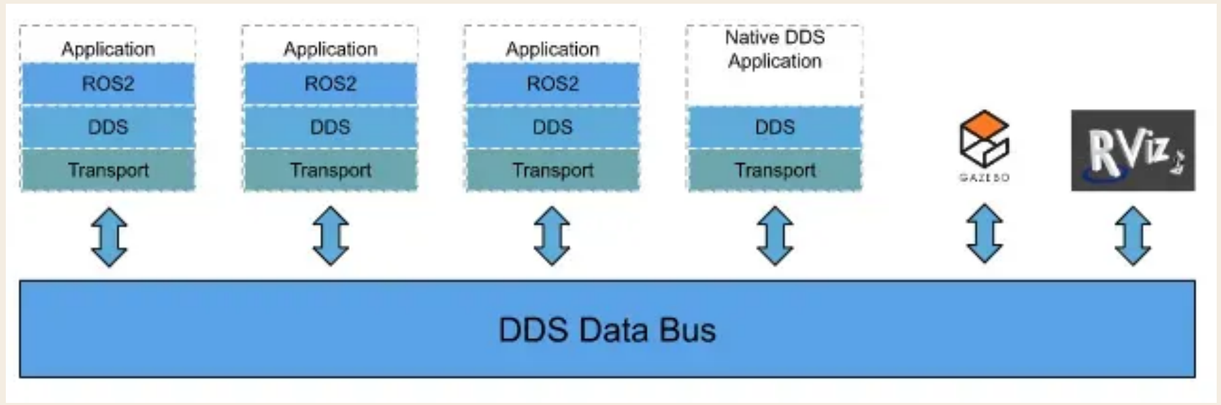


ROS VS ROS2

Issue with ROS: Not made for real-time systems and is centralized!



ROS uses Master



ROS2 uses DDS (Data Distribution Server)

ROS2 is crossplatform: ROS mainly functions on Linux (easiest to set up on Ubuntu). ROS2 can run on Linux, Windows, macOS, RTOS, and even microcontrollers, giving more flexibility in choosing robot hardware.

ROS VS ROS2

1. Communication Layer:

- **ROS:** Based on **TCP/UDP** + a custom protocol, with limited capabilities.
- **ROS2:** Uses industrial-grade **DDS**, which supports configurable QoS to meet low-latency, high reliability, encryption, and other scenarios.

2. Build System:

- **ROS:** Used **rosbuild / catkin**, which could “break” in big projects or with complex dependencies.
- **ROS2:** Uses **colcon + ament**, providing more elegant multi-package management and dependency handling.

3. Command Line:

- **ROS:** Commands like **rosrun, rostopic, rosservice** are spread out.
- **ROS2:** Unified under the **ros2** main command, with more and extensible subcommands.

WORKSPACE

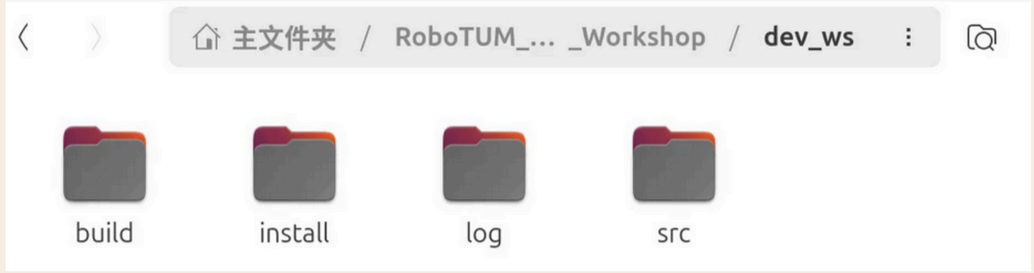
Home base of your ROS2 project.

- Source code, dependencies, build artifacts, and executables: All live here!
- Packages are separated in subdirectories.

Typical Layout:

```
bash

dev_ws/
  ├── src/        # Source code, scripts, packages
  ├── build/      # Build artifacts
  ├── install/    # Compiled executables, libraries
  └── log/        # Build and runtime logs
```



- **src/:** Packages (source code)
- **build/:** Processing center
- **install/:** Finished products
- **log/:** Production logs

Easy to create! Just like any folder:

$ mkdir -p ros2_ws/src

PACKAGES

Package = Organizational unit of code. Nice and tidy package of code, easy to distribute and share.

rclpy

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

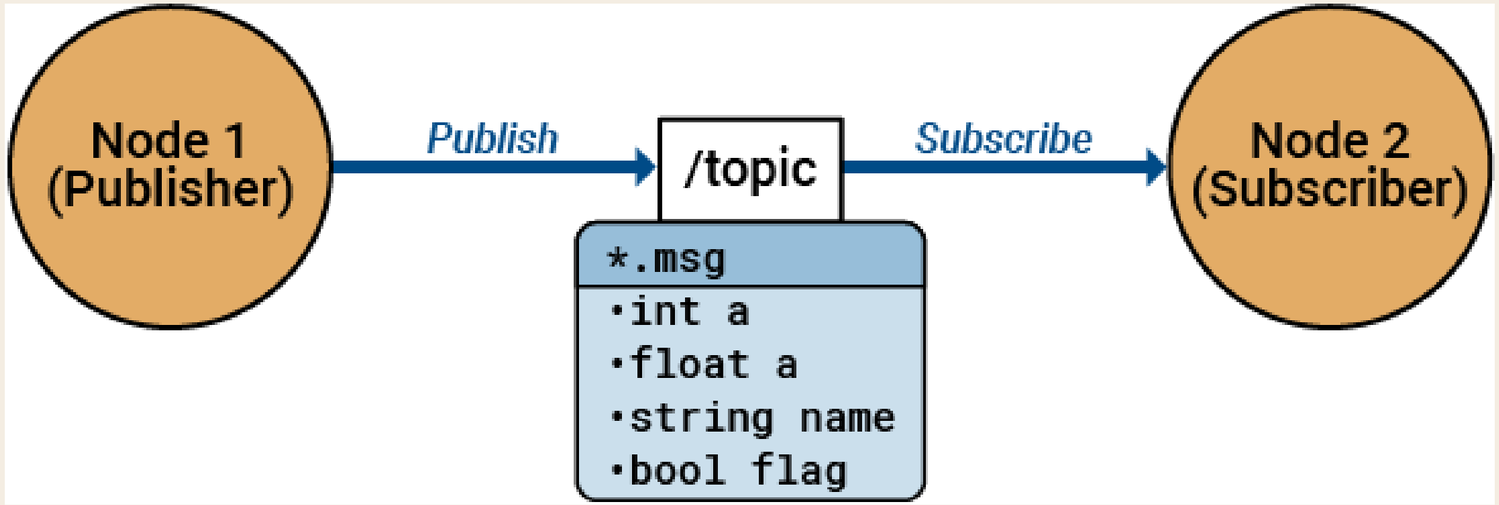
```
my_package/
      package.xml
      resource/my_package
      setup.cfg
      setup.py
      my_package/
```

rclcpp

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

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

NODES AND TOPICS



Node: Executable process that communicate through the ROS2 graph.

Topic: Communication channel through which nodes talk to each other.

Nodes are run using

$ ros2 run <package_name> <node_name>

NODES AND TOPICS

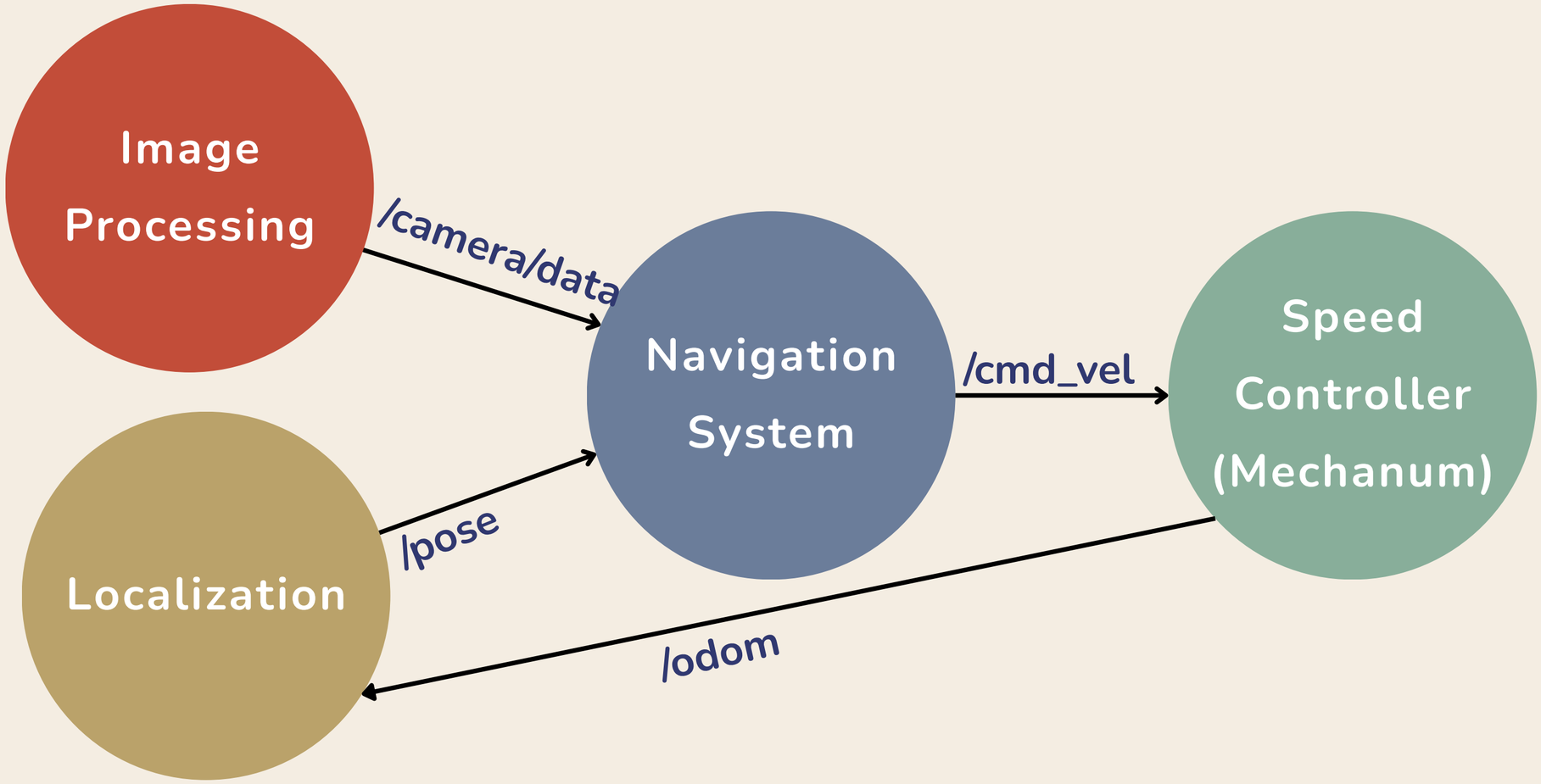
Image Processing

Navigation System

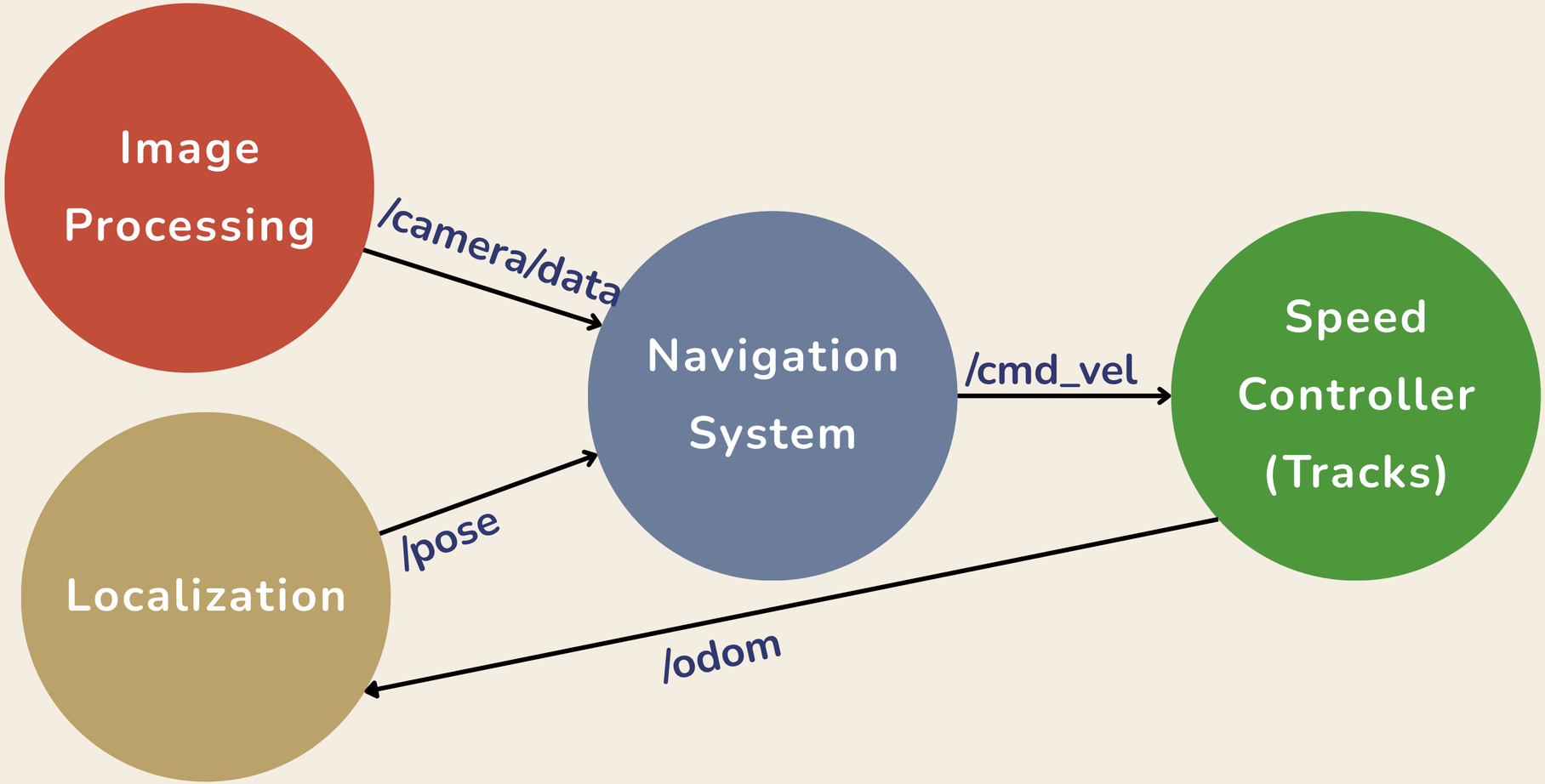
Localization

Speed Controller (Mechanum)

NODES AND TOPICS



NODES AND TOPICS



MESSAGES:

Messages are described by **.msg** text files. They define the type of information that a Node sends or receives through a topic.

Composed of two parts: **Fields** and **constants.**

For example:

```
uint8 x 42
int16 y -2000
string full_name "John Doe"
int32[] samples [-200, -100, 0, 100, 200]
```

Built-in-types currently supported:

Type name	C++	Python	DDS type
bool	bool	builtins.bool	boolean
byte	uint8_t	builtins.bytes*	octet
char	char	builtins.str*	char
float32	float	builtins.float*	float
float64	double	builtins.float*	double
int8	int8_t	builtins.int*	octet
uint8	uint8_t	builtins.int*	octet
int16	int16_t	builtins.int*	short
uint16	uint16_t	builtins.int*	unsigned short
int32	int32_t	builtins.int*	long
uint32	uint32_t	builtins.int*	unsigned long
int64	int64_t	builtins.int*	long long
uint64	uint64_t	builtins.int*	unsigned long long
string	std::string	builtins.str	string
wstring	std::u16string	builtins.str	wstring

HANDS ON:

1. **Initialization**
   - In Python: `rclpy.init()`; in C++: `rclcpp::init()`
   - Allocates resources needed by the node.
2. **Create the Node Object**
   - You specify a node name, finishing internal setup.
3. **Run Main Logic**
   - The primary work—often in a loop or via callbacks.
4. **Node Destruction**
   - When finished or on manual exit: `rclpy.shutdown()` frees resources.

BUILDING PACKAGES

Packages are built from the workspace directory using colcon. Additional arguments can be given to personalize the build process:

- **$ colcon build**
- **$ colcon build --packages-select <package_name_1> <package_name_2> ...**
- **$ colcon build --symlink-install**
- **$ colcon build --cmake-args -DCMAKE_EXPORT_COMPILE_COMMANDS=ON**
- **$ colcon build --parallel-workers <number_of_workers>**

After a package is built, it needs to be **sourced** on every new terminal:

$ source install/setup.bash

CLI TOOLS

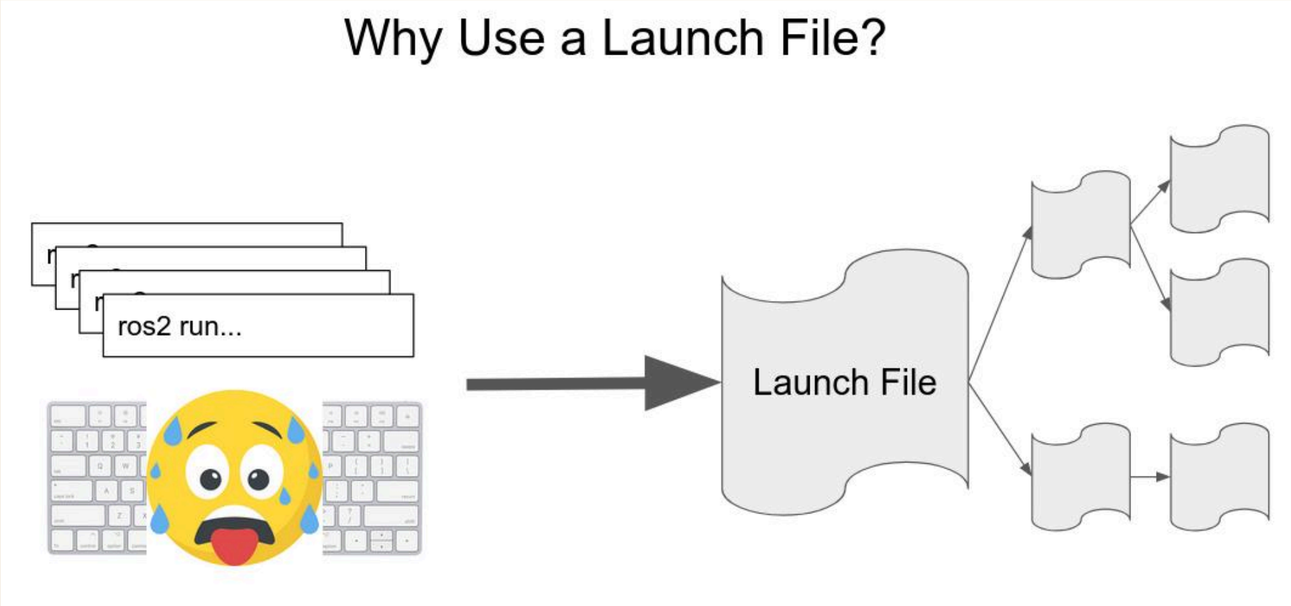
There are plenty of tools that can be used from the command line interface:

1. **ros2 run <package_name> <node_name>**
2. **ros2 topic list**
3. **ros2 node list**
4. **ros2 topic info <topic_name>**
5. **ros2 node info <node_name>**
6. **ros2 topic echo <topic_name>** (can also add --once to only print one time)
7. **rqt_graph**
8. **ros2 topic pub <topic_name> <msg_type> <message_contents>**

LAUNCH FILES

Issue: Do we want to open up however many terminals when we need to run multiple nodes?

Launch files allow for starting multiple nodes at once, manage and configure their parameters!

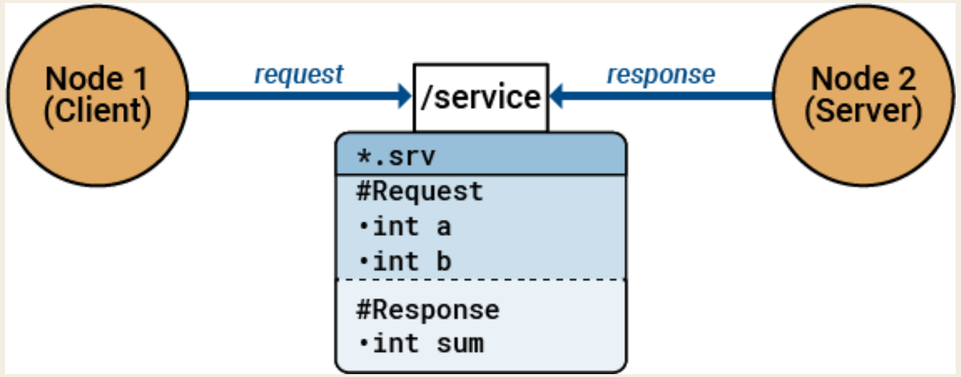


COMMUNICATION INTERFACES

Besides **messages** (.msg), the other two main communication interfaces are **services** (.srv) and **actions** (.action).

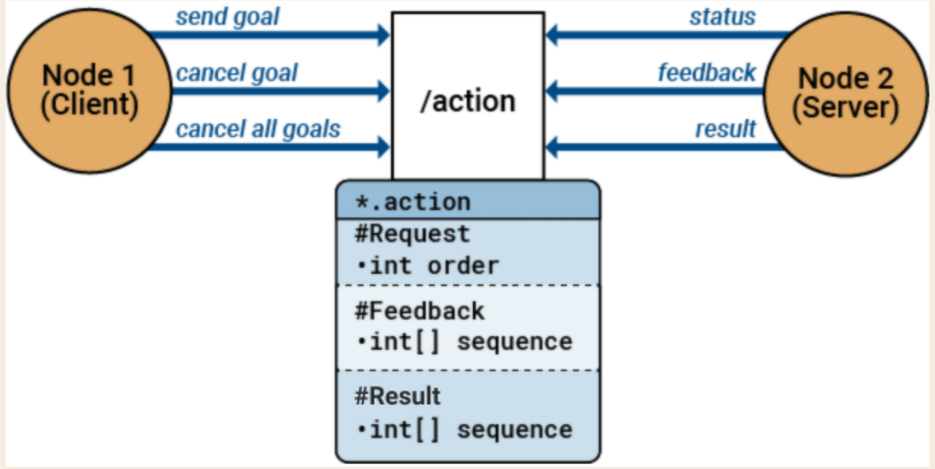
Services

- Request-Response model
- Client send a request
- Server calculates and responds

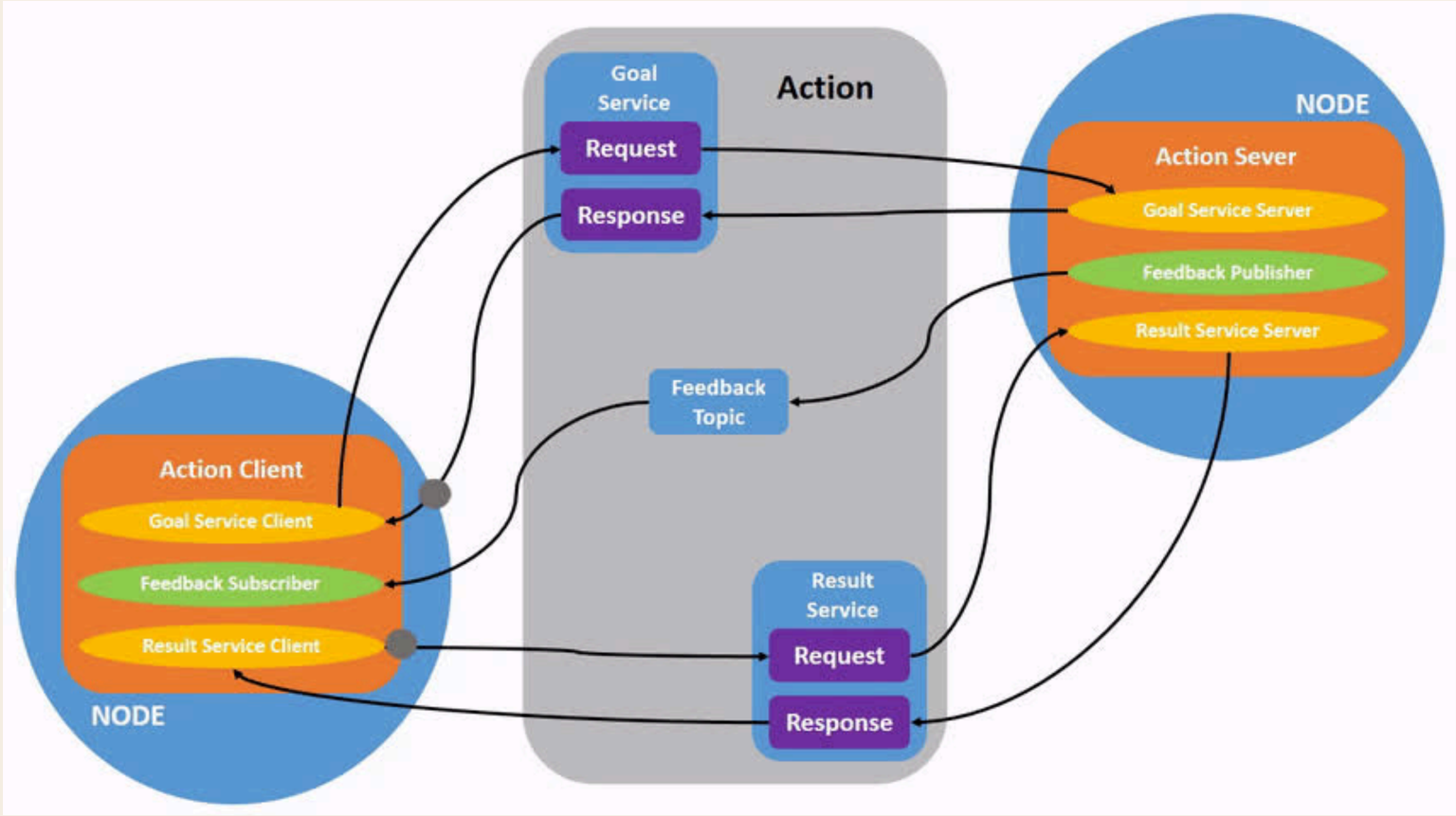


Actions

- Long-running operation
- Periodic Feedback

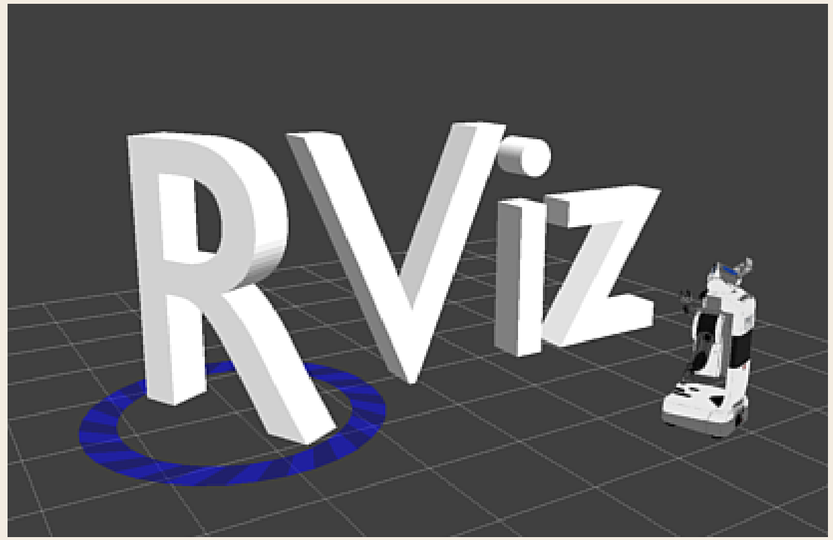


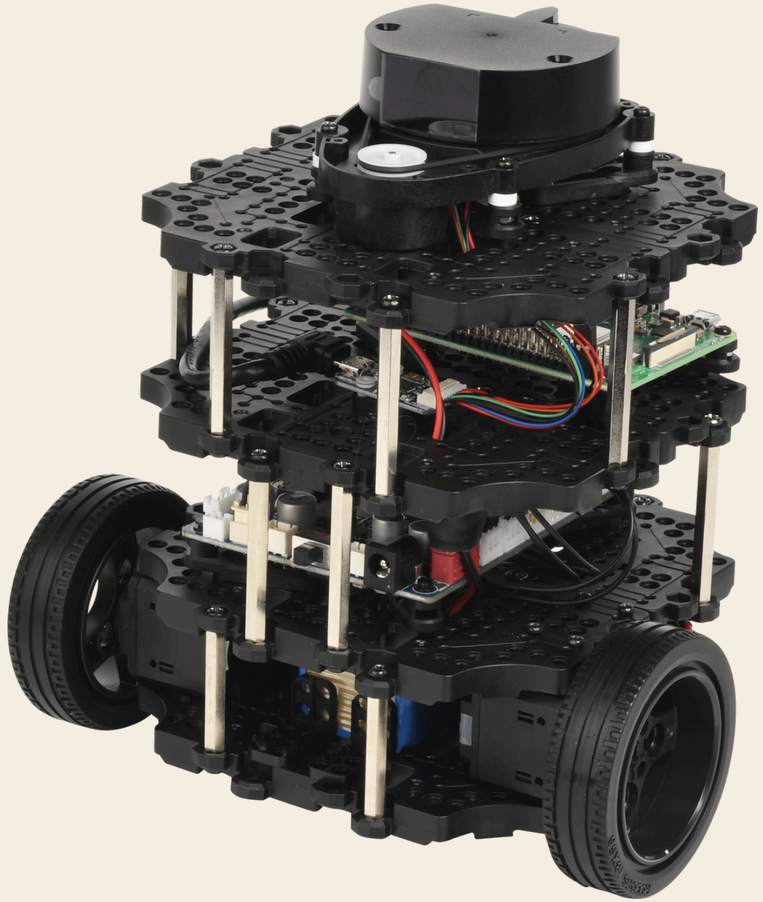
ACTION UNDER THE HOOD



GAZEBO EXAMPLE

GAZEBO

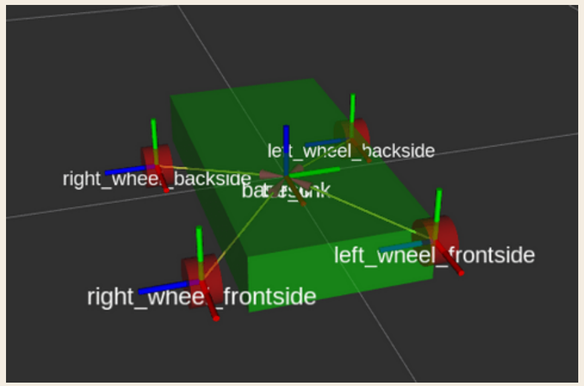


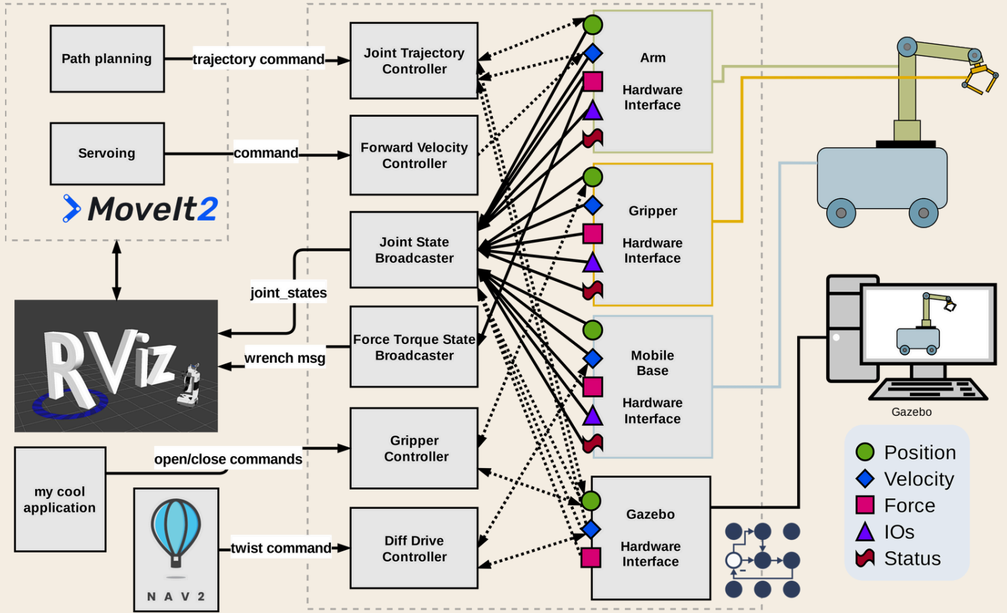


WHAT ELSE IS THERE?

A lot! Many things were not covered in this short workshop:

- Actions
- URDFs
- TF2
- rosbags
- Hardware Interfaces
- Control Manager
- Nav2
- MoveIt
- Real-Time Systems
- Sensor Drivers





N A V 2

Powered By

OPEN NAVIGATION

MoveIt2

3rd December
2025

ROS2 WORKSHOP

Ignacio Dassori W.