Robot Programming with ROS2

Instructor:
Erfan Riazati

Fall 2025

Iran University of Science and Technology

WEEK 0
Course Setup

ROS2 Installation

Distribution

- ROS2 Humble Hawksbill

Supported OS

- Ubuntu Jammy Jellyfish (22.04)
  Windows 10 (VS2019), macOS, ...

Course Setup

- Native/Dual-Boot Ubuntu
  Virtual Machine (VMWare Player)

for more information:

- ROS2 Humble Documentation Installation
- REP2000

Course Schedule

Grading Scale:

- Presence 40%
- Homework 60%

+60% to obtain certification

	WEEK 1	WEEK 2	WEEK 3	WEEK 4
Session 1	**ROS2 Concepts** ▪ Course Setup ▪ ROS Vs. ROS2 ▪ ROS2 Design ▪ Publishers and Subscribers ▪ Interfaces and Messages	**URDF and Simulation** ▪ Build a Visual Robot Model ▪ URDF and Xacro ▪ Gazebo Simulation ▪ Publish Robot States ▪ Use Existing Models ▪ Changing Worlds	**Robotic Manipulation** ▪ Serial Robot Manipulators ▪ MoveIt2 Introduction → Collisions → Kinematics → APIs	**Behavior Trees** ▪ Real-World Scenarios ▪ Behavior Tree Tools ▪ Factory Example ▪ Final Project
Session 2	**Review and Examples** ▪ Review Basics ▪ Services ▪ Actions ▪ Examples: → Trajectory Control → ...	**Robot Navigation** ▪ Mobile Robots ▪ SLAM Algorithms ▪ Nav2 Introduction → Localization → Planning and Control ▪ Goal Points and Patrolling	**Robot Perception and TF** ▪ Transform Coordinates ▪ Static Transform ▪ TF Broadcasters ▪ ROS Standards ▪ Robot Vision	**Conclusion** ▪ Course Summary ▪ Advanced Topics ▪ Future Directions ▪ Available Resources ▪ Opportunities
	HW1	HW2	HW3	HW4

Week 1

ROS2 Introduction

History of ROS

- originally developed in 2007 at the Stanford Artificial Intelligence Laboratory
- the most common problem in robotics:
  - excessive time dedicated to re-implementing the software infrastructure
  - inadequate time allocated to actually building intelligent robotics programs
- **ROS**; a framework that allows processes to communicate with each other and provides some tools to help create code on top of that.
- since 2013 managed by Open-Source Robotics
- de facto standard for robot programming

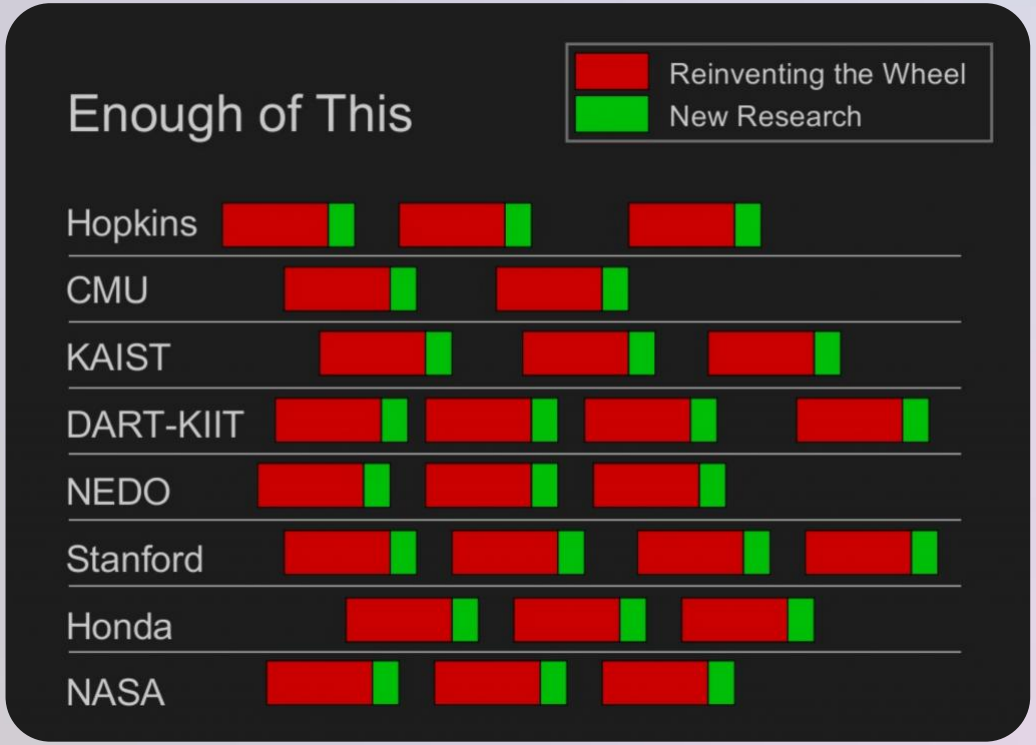


Fig. 1. Eric and Keenan pitch deck.

Typical Robotics Modules

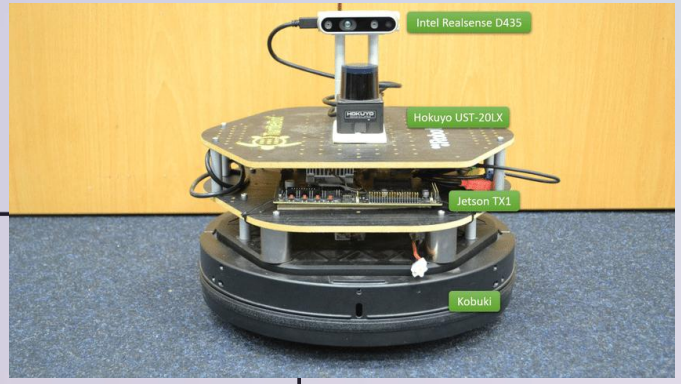


Fig. 1. Kobuki Robot.

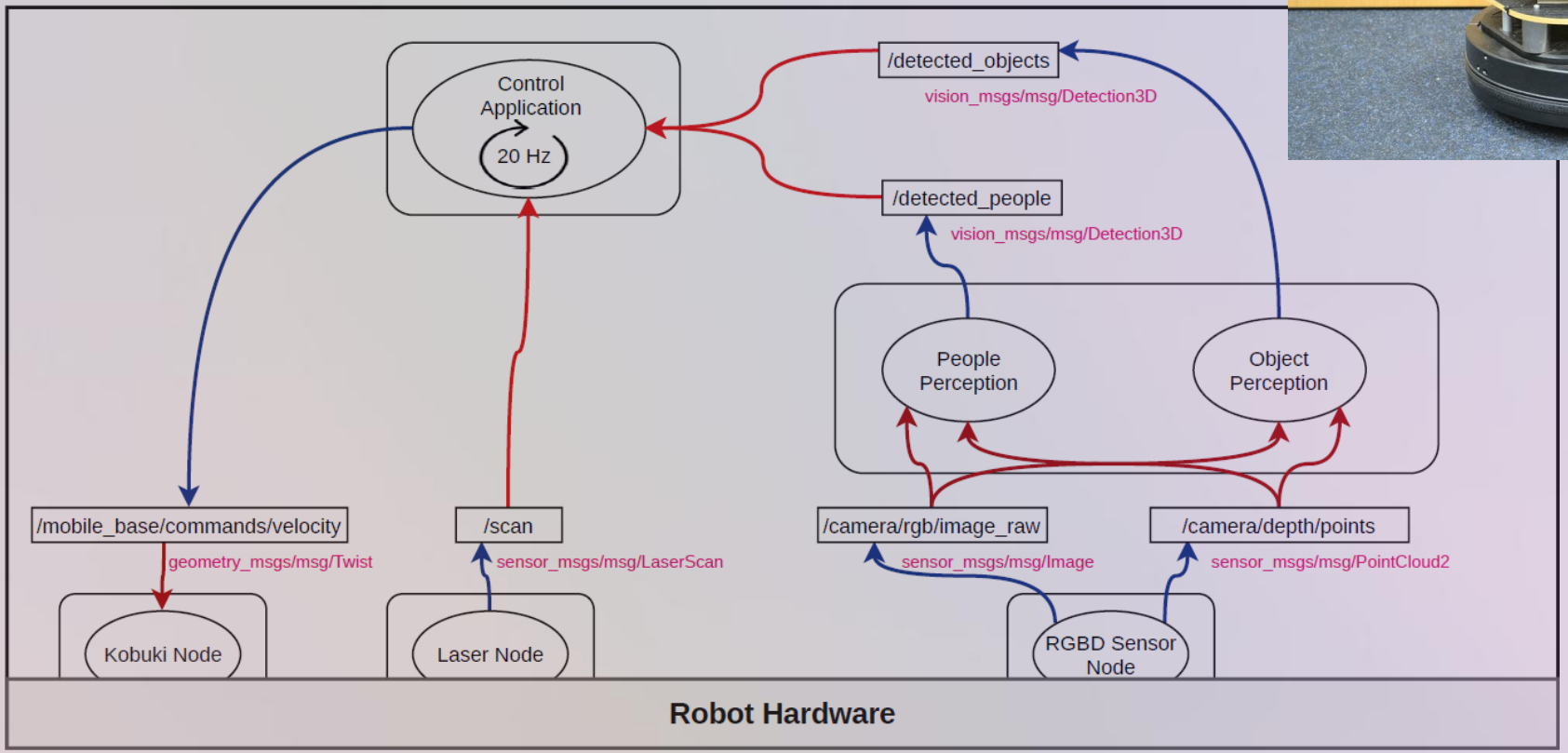


Fig. 1. Representation of software layers in a robot.

Robotics Software Stack

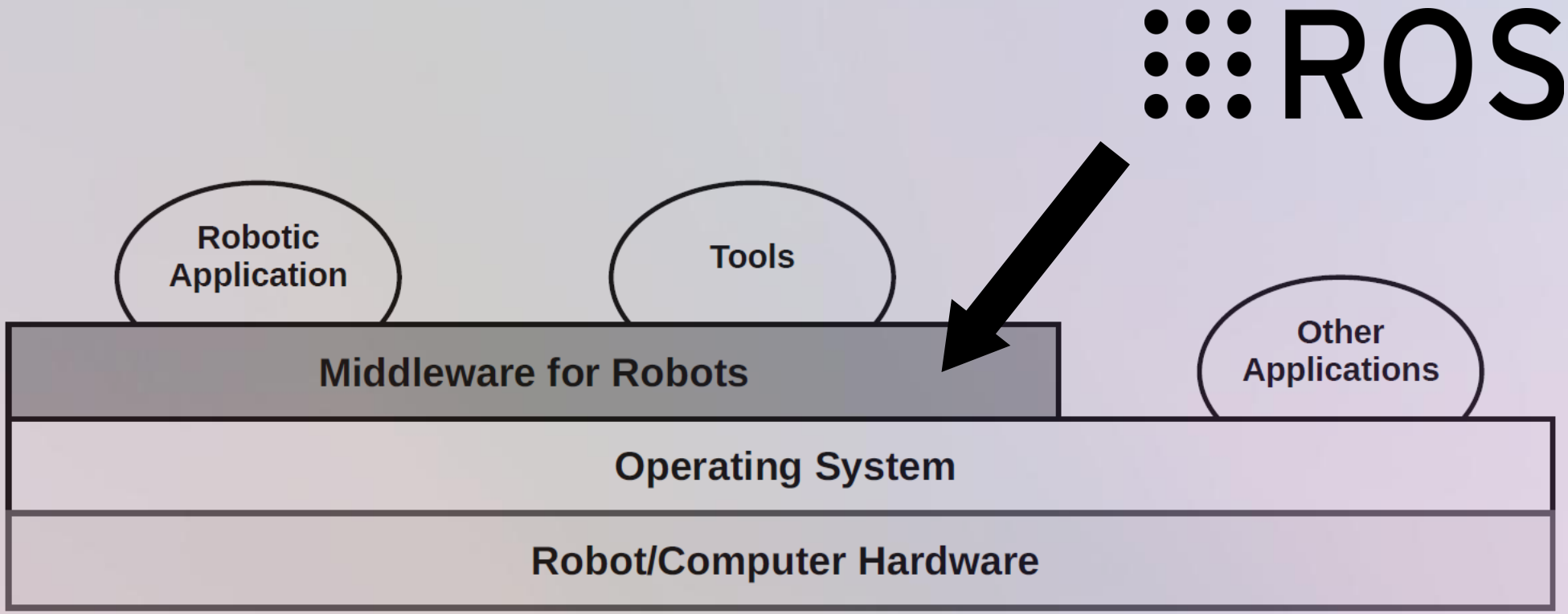


Fig. 1. Representation of software layers in a robot.

ROS1 vs. ROS2

- **Middleware:**
  - ROS1 → TCPROS/UDPROS
  - ROS2 → DDS
- **Real-Time Support**
- **Quality of Service (QoS)**
- **Security Features**
- **Build System:**
  - ROS1 → `catkin`
  - ROS2 → `ament`
- **Maintenance:**
  - "ROS1 is DEAD, and we have killed it!"
- **Many Other Things:**
  - Node Composition, Lifecycle Nodes, ...

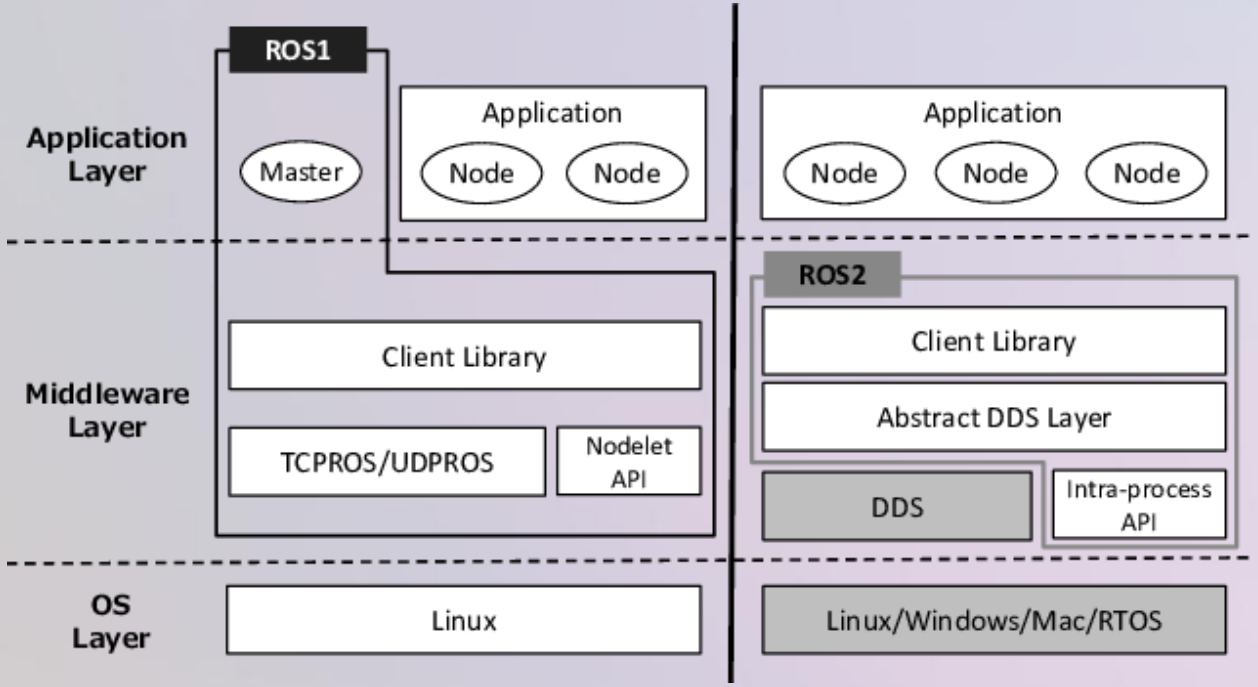


Fig. 1. ROS1/ROS2 Architecture.

ROS2 Design

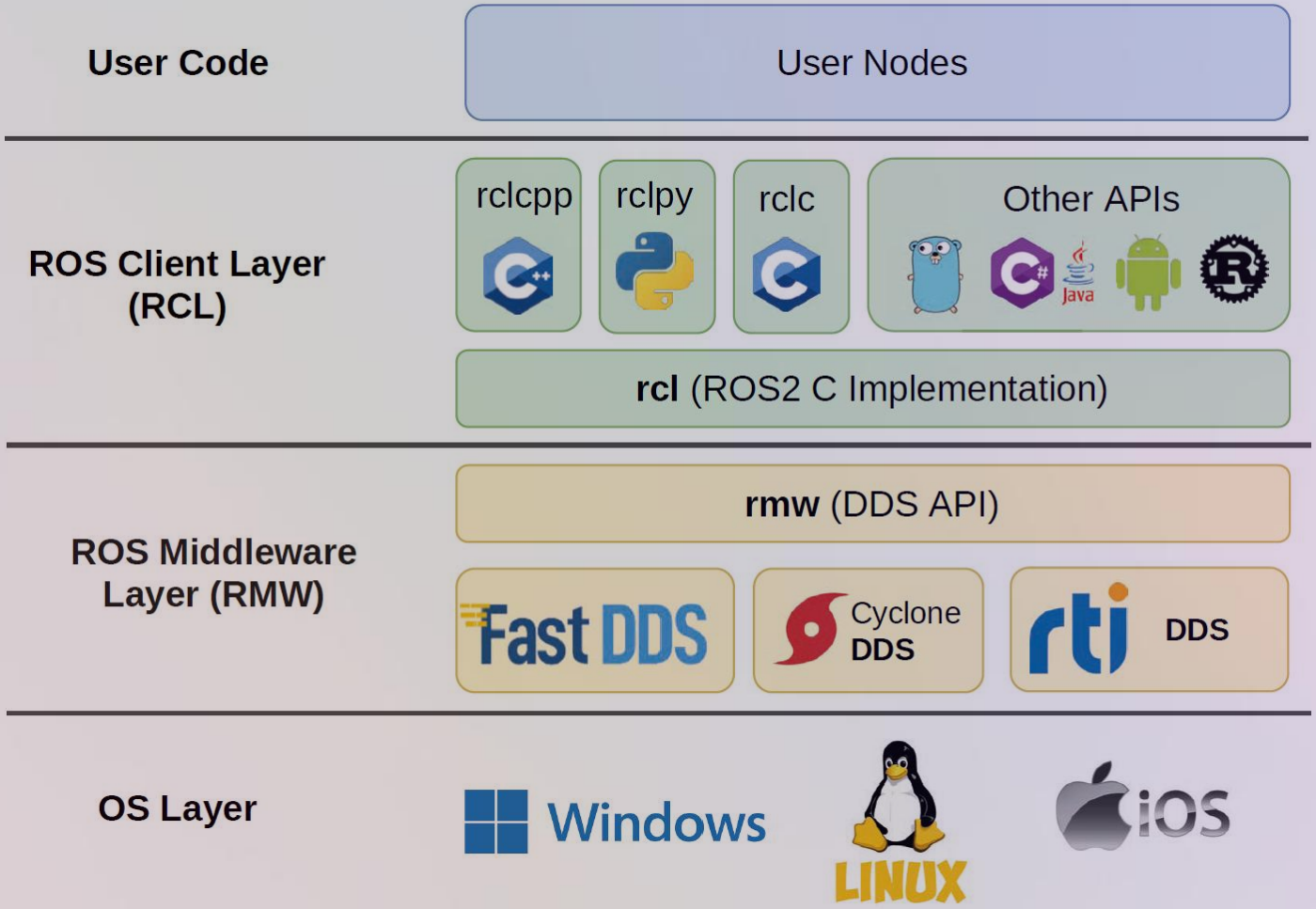


Fig. 2. ROS2 layered Design.

WEEK 2

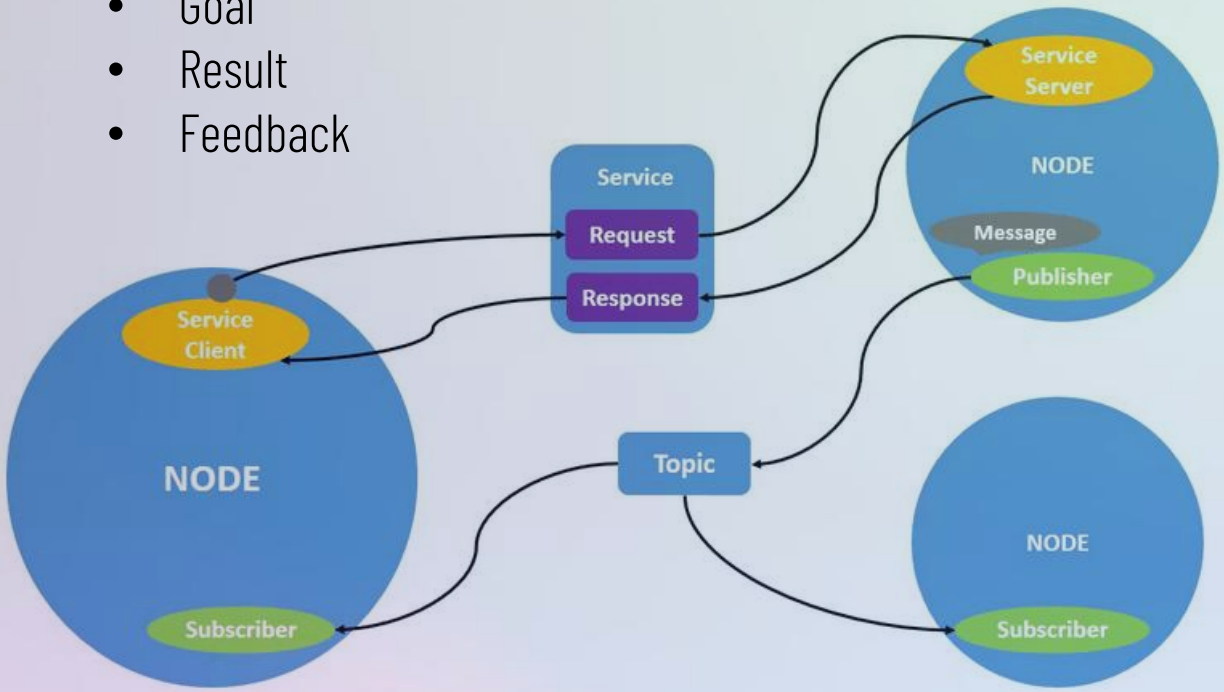
ROS Essentials

ROS Basics

- **Nodes:**
  - Software Processes that do things
  - Usually in C++ and Python
- **Topics:**
  - Transform Information between Nodes
  - Organized as Data Structures
  - Identified by “Name” and “Type”

- **Publishers/Subscribers:**
  - Use Topics to Communicate
  - **Publishers** Generate Information
    - e. g. Sensors and Actuator Drivers
  - **Subscriber** Receive Information
    - e. g. Monitoring Systems

- **Services:**
  - Event-based execution using Request-Response Communication
  - e. g. Taking a Camera Snapshot
- **Actions:**
  - Generalized Request-Response Non-Blocking Execution
  - e. g. Autonomous Navigation
  - Includes:
    - Goal
    - Result
    - Feedback



ROS2 Commands

Setup ROS2 Environment

Before using any ROS 2 command, source the setup file:

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

To make this automatic every time you open a terminal:

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

Install Turtelsim (if not already installed)

```
$ sudo apt update
$ sudo apt install ros-humble-turtlesim
```

ROS Packages Installation Structure

```
$ sudo apt install ros-{$ROS_DISTRO}-{package_name}
```

ROS2 Commands Structure

```
$ ros2 <command> <verb> [<params>|<option>]*
```

Example

```
$ ros2 run turtlesim turtlesim_node
```

Turtlesim Example

In the first terminal run the following command:

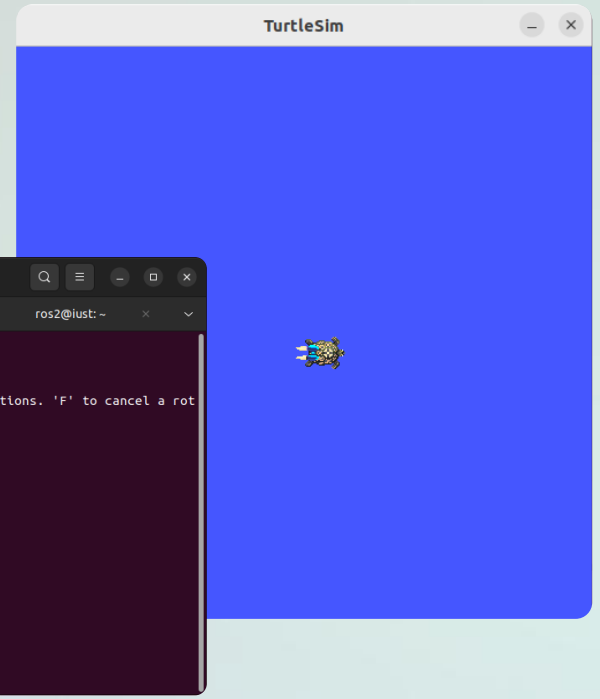
```
$ ros2 run turtlesim turtlesim_node
```

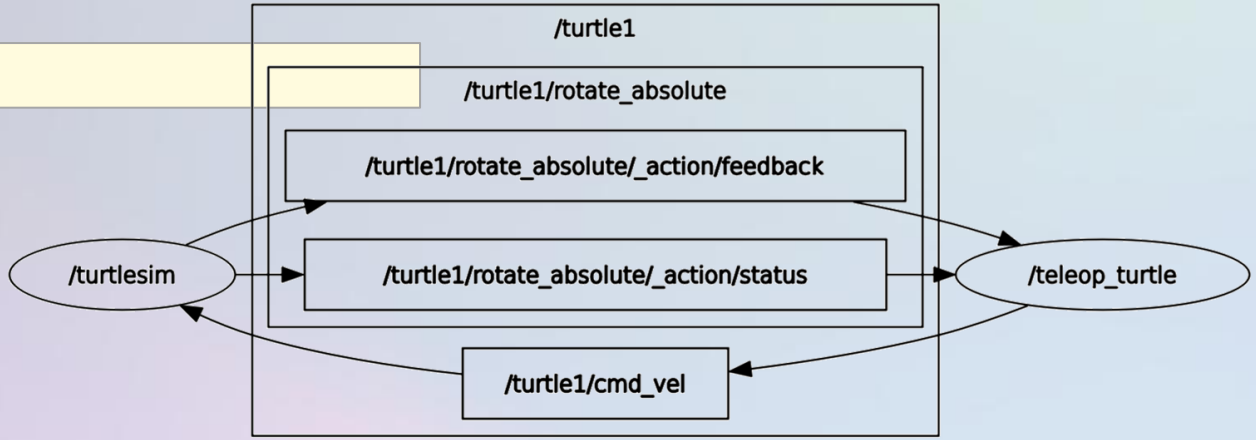
In a new terminal run the following command and use arrow keys to move the turtle:

```
$ ros2 run turtlesim turtle_teleop_key
```

Witness the computing graph of the current ROS application using `rqt`:

```
$ ros2 run rqt_graph rqt_graph
```





ROS2 Commands Cheat Sheet

`$ ros2 node list`	Lists all active ROS nodes
`$ ros2 node info </node_name>`	Information about a specific node
`$ ros2 topic list`	Lists all active ROS topics
`$ ros2 topic list -t`	Lists all active ROS topics with their type
`$ ros2 topic info </topic_name>`	Information about a specific topic
`$ ros2 topic echo </topic_name> <topic_type>`	Returns messages published on a topic
`$ ros2 topic pub </topic_name> <topic_type> <value>`	Publishes a message on a topic
`$ ros2 topic pub </topic_name> <topic_type> <value> -r <rate>`	Publishes a message with a fixed rate
`$ ros2 topic pub </topic_name> <topic_type> <value> -1`	Publishes a message only once
`$ ros2 interface show <message_type>`	Shows inside a message type

Hello World!

Let's write our first publisher and subscriber:

Objective:
Write a ROS Publisher that generate a String message on an arbitrary topic every half a second. On the other side, develop a ROS Subscriber to receive the message

More Information:
[ROS2 Tutorial]

ROS2 Simple Publisher with Python

```python
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalPublisher(Node):

    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1


def main(args=None):
    rclpy.init(args=args)

    minimal_publisher = MinimalPublisher()

    rclpy.spin(minimal_publisher)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_publisher.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

ROS2 Simple Subscriber with Python

```python
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalSubscriber(Node):

    def __init__(self):
        super().__init__('minimal_subscriber')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)


def main(args=None):
    rclpy.init(args=args)

    minimal_subscriber = MinimalSubscriber()

    rclpy.spin(minimal_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

ROS2 Workspace

ROS2 Packages