Introduction to ROS 2

Course: Robotic Programming Environments

Michał Drwięga
michal.drwiega@pwr.edu.pl
www.mdrwiega.com/edu/rpe

Department of Cybernetics and Robotics
Wrocław Univeristy of Science and Technology

Wrocław University
of Science and Technology

Content

- ROS: Introduction & History
- ROS[1] vs ROS 2[2]
- Example robots
- Nodes & Topics

[1] Notation: ROS == ROS 1
[2] Notation: ROS2 == ROS 2

ROS: Introduction & History

ROS (Robot Operating System)

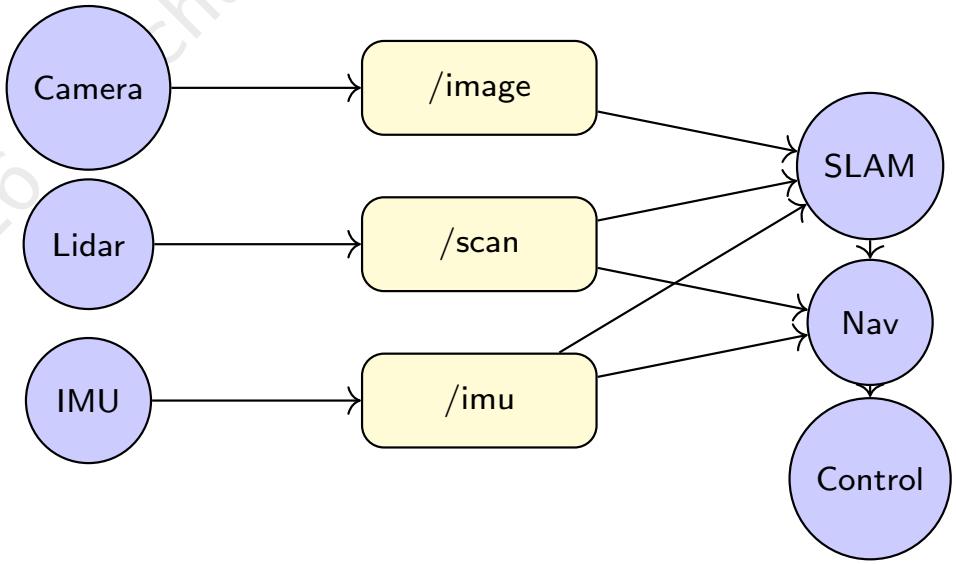
What is ROS?

A flexible framework for writing robot software. Collection of tools, libraries, and conventions that simplify building complex robots.

Core Idea: Break robot software into small, reusable nodes that communicate via messages

Core Features

- ✓ Open-source
- ✓ Language-neutral (C++, Python)
- ✓ Component-based architecture
- ✓ Hardware abstraction



What ROS is NOT

- Not an operating system
- Not a single application
- Not only for robots
- Not real-time by default
- Not a GUI tool

Motivation for ROS Development

Common challenges in robotics:

- Re-implementing software infrastructure is time-consuming, yet essential for developing complex robotic algorithms.
- Limited time available for creating truly intelligent robots.

History of ROS

- (2007) - Started as a personal project of Keenan Wyrobek and Eric Berger at Stanford University. The project was called Stanford Personal Robotics Program
- (2008 - 2013) - In 2008, development moved to Willow Garage, a research center focused on robotics.
- (2013 - now) - Since 2013, ROS has been managed by the Open Source Robotics Foundation (OSRF), which was renamed Open Robotics in 2017. [3]

[3] https://www.theconstructsim.com/history-ros/

History of ROS – ROS 1 Releases

- 2010 - Box Turtle
- 2010 - C-Turtle
- 2011 - Diamond Back
- 2011 - Electric Emys
- 2012 - Fuerte Turtle
- 2012 - Groovy Galapagos
- 2013 - Hydro Medusa
- 2014 - Indigo Igloo
- 2015 - Jade Turtle
- 2016 - Kinetic Kame
- 2017 - Lunar Loggerhead
- 2018 - Melodic Morenia
- 2020 - Noetic Ninjemys

History of ROS/ROS 2

- ROS 2 was announced in 2014 [4]
- Why ROS 2 was planned instead of a new release of the first ROS?

[4] https://www.theconstructsim.com/history-ros/

Motivation for the ROS 2 development

Limitations of ROS 1

- Single point of failure, SPoF (the ROS master)
- Lack of built-in security
- No support for real-time operations

Assumptions regarding ROS 2

- Security
- Operation on embedded systems
- Real-time support
- Multiple platforms

ROS 2 Principles

- **Distributed** - No single point of failure
- **Modular** - Components work independently
- **Asynchronous** - Event-based communication
- **Abstracted** - Hide hardware complexity

ROS 2 Ecosystem

Navigation (Nav2)

- Path planning
- Obstacle avoidance
- Behavior trees
- Recovery behaviors

Manipulation (MoveIt2)

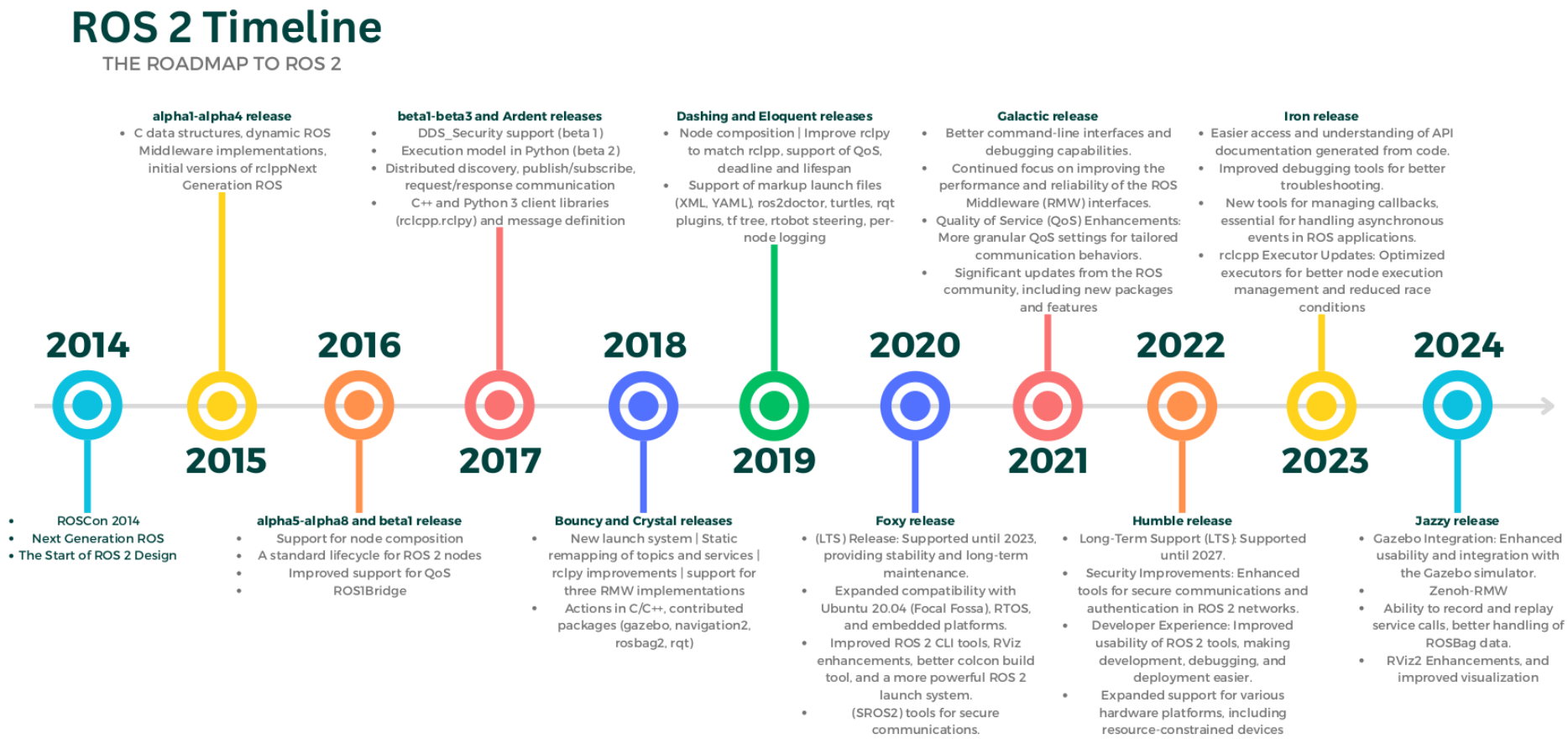
- Kinematics
- Motion planning
- Collision checking
- Pick & place

Simulation

- Gazebo (physics)
- Webots
- Ignition
- NVIDIA Isaac

Community: 1000+ packages, active development, industrial adoption

A history of ROS – ROS 2 Releases



- 2025 - ROS 2 Kilted (targeted for Ubuntu 24.04)

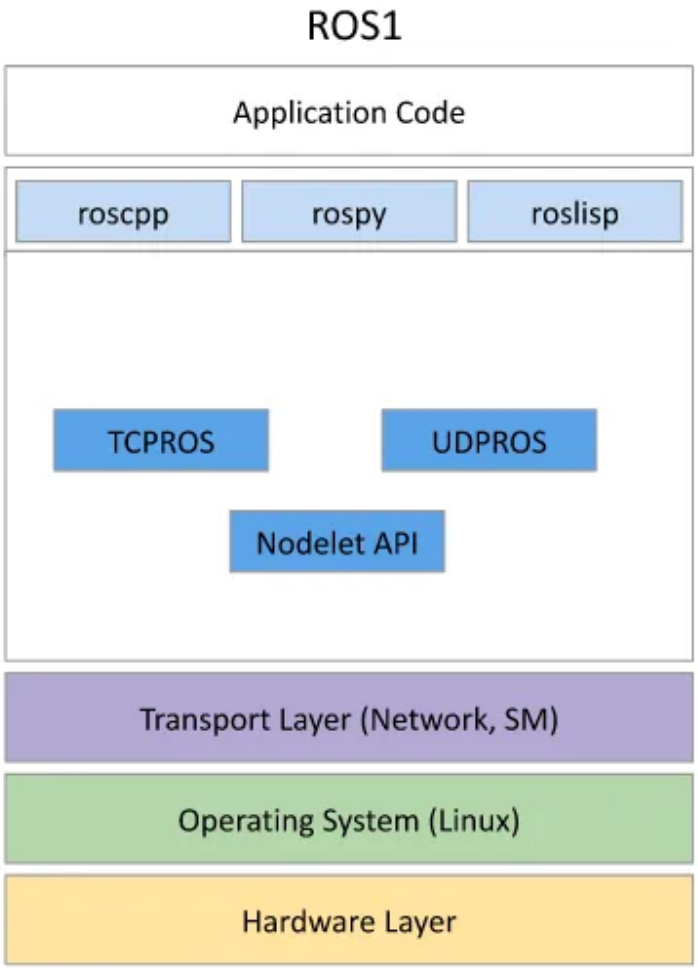
ROS 1 vs ROS 2

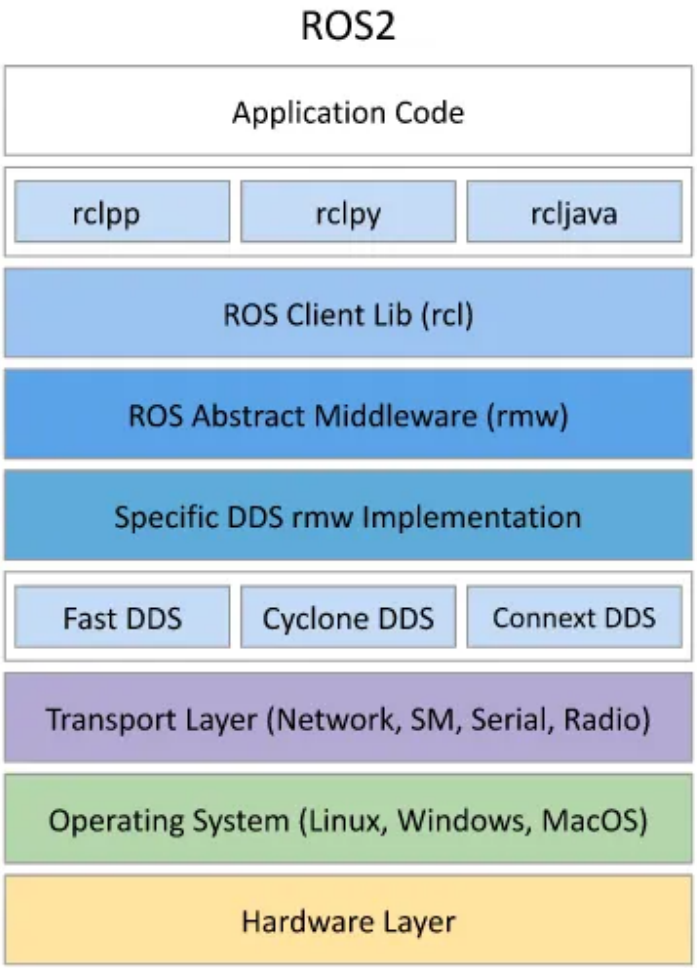
ROS 1 vs ROS 2

Feature	ROS 1	ROS 2
Master	ROS Master (single point)	**Distributed discovery**
Communication	TCPROS/UDPROS	**DDS standard**
Real-time	Limited	**Real-time capable**
Platforms	Linux only	Linux, Windows, macOS
Security	None built-in	**Encryption + auth**
QoS	Fixed	**Configurable**

- ✓ ROS 2 is designed for **production** and **commercial** use
- ✓ ROS 2 supports **multi-robot** systems natively
- ✓ ROS 2 has no central point in the system - no single point of failure

ROS 1 vs ROS 2 Architecture





5

[5] https://medium.com/

ROS - applications

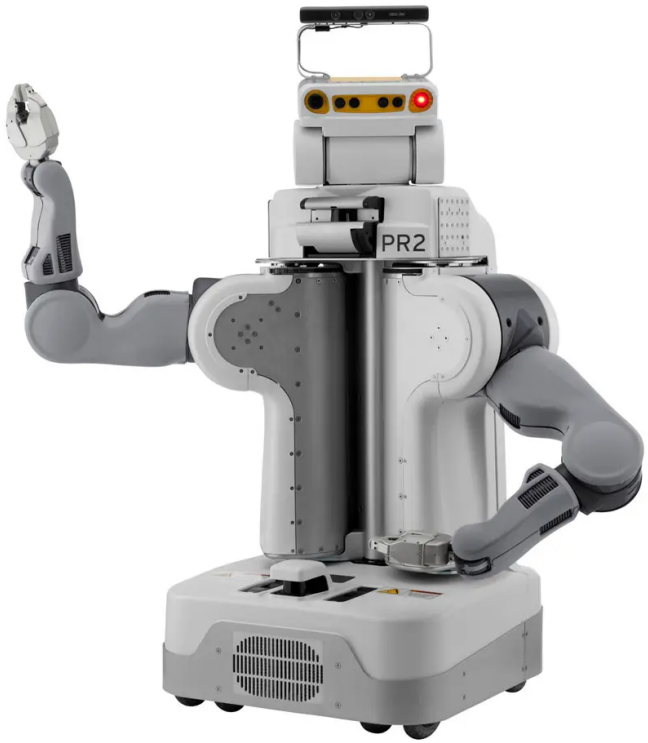
- Mobile robots
- Manipulators
- Mobile Manipulators
- Humanoid robots
- UAVs (Unmanned aerial vehicle)
- UUVs (Unmanned underwater vehicle)
- Autonomous cars
- Many others

6

[6] https://husarion.com/

PR2 (Personal Robot 2) - ROS-based robot

- Developed by Willow Garage in 2010[7]
- An open and robust robot platform created for researchers
- Fully integrated with ROS
- Features: autonomous navigation, grasping, objects manipulation
- Specification: height 165 cm, length 66.8 cm, weight 226 kg, speed 3.6 km/h
- Sensors: cameras, lidars, RGB-D cameras, IMU, fingertip pressure sensors
- Cost: $400 000 (in 2010)



8

[7] https://wiki.ros.org/Robots/PR2
[8] https://robotsguide.com/robots/pr2

Turtlebot 2 - the most famous ROS-based robot

- A low-cost, wheeled mobile robot with open-source software
- An open and robust robot platform created for researchers
- Fully integrated with ROS
- Features: autonomous navigation
- Specification[9]: dimensions: 35.4 x 35.4 x 42 cm, weight 6.3 kg, max speed 0.65 m/s
- Sensors (in basic version): encoders, bumpers, RGB-D camera
- Cost: $< \$2\,000$

10

[9] https://clearpathrobotics.com/turtlebot-2-open-source-robot/
[10] https://www.turtlebot.com/turtlebot2/

Example Sensors Supported by ROS

- **Lidar Sensors**: For 2D/3D mapping and obstacle detection (e.g., Velodyne, RPLidar).
- **Cameras**: RGB, depth, and stereo cameras for visual perception (e.g., Microsoft Kinect, Intel RealSense).
- **Inertial Measurement Units (IMUs)**: Provide orientation and acceleration data (e.g., MPU-6050, Xsens).
- **GPS**: Essential for outdoor localization (e.g., u-blox, Trimble).
- **Ultrasonic Sensors**: Used for proximity sensing and obstacle avoidance (e.g., HC-SR04).
- **Force/Torque Sensors**: Measure forces in robotic manipulation (e.g., ATI Industrial Automation).
- **Temperature and Pressure Sensors**: Important for environmental monitoring.

ROS 2 Nodes & Topics

ROS 2 Graph - Core Concepts

Nodes

Individual processes that perform computation.

- Camera driver node
- Lidar processing node
- Motion control node

Messages

Typed data structures for communication.

- `sensor_msgs/Image`
- `geometry_msgs/Twist`
- `nav_msgs/Odometry`

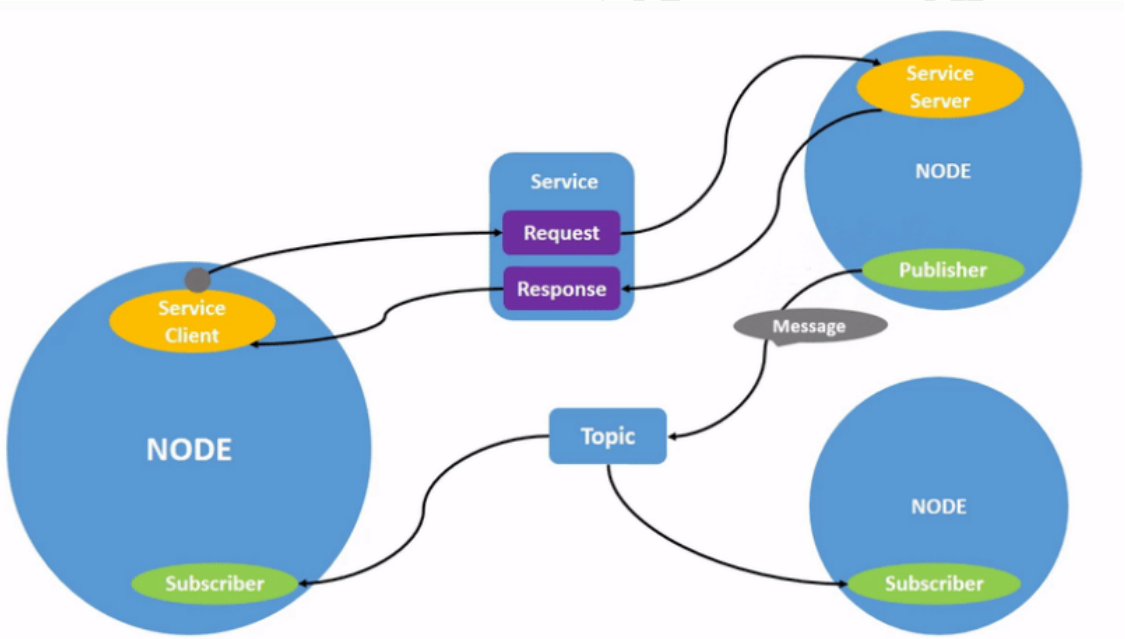
Topics

Named buses for asynchronous data streams.

- `/camera/image` - Images
- `/scan` - Laser scans
- `/cmd_vel` - Velocity commands

ROS 2 Node

- A node is independent programming unit.
- Single purpose, for example path generation.
- Can be written in any supported language (C++, Python, others).

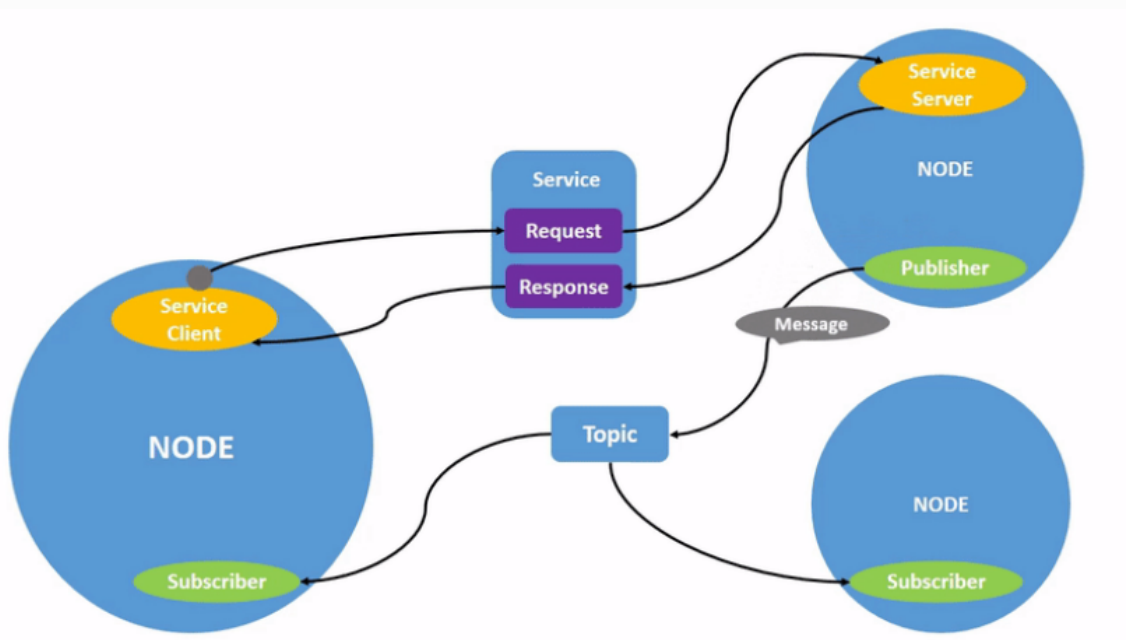


[11]

[11] https://docs.ros.org/

ROS 2 Node

- **Functionality:** Performs some specific tasks in the ROS 2 ecosystem.
- **Communication:** Communicates with other nodes via ROS 2 topics/services/actions.
- **Parameters:** Configurable parameters for customization.



[11]

[11] https://docs.ros.org/

Node - code example (Python)

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

class ExampleNode(Node):
    def __init__(self):
        super().__init__('example_node')
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.
            timer_callback)
        self.i = 0

    def timer_callback(self):
        self.get_logger().info(f'Counter: {self.i}')
        self.i += 1

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

    example_node = ExampleNode()
    rclpy.spin(example_node)
```

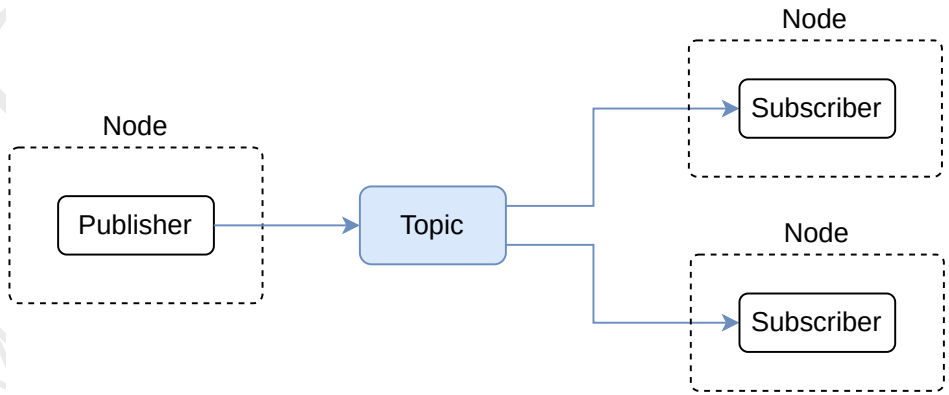
ROS 2 CLI: Node Management

Command	Description
`ros2 run <pkg> <exec>`	Start a node from a package
`ros2 node list`	List all active nodes
`ros2 node info <node>`	Show node details (topics, services, actions)
`ros2 launch <pkg> <file>`	Start multiple nodes via launch file
`ros2 daemon status`	Check the background discovery service

The ROS 2 daemon runs in the background to cache node information for faster queries. Use `ros2 daemon stop` to clear the cache if nodes don't appear.

Communication: Topics

- Based on publish-subscribe model
- Message based
- Pattern: many to many
- Underlaying layer: DDS
- Communication frequency depends on the publisher

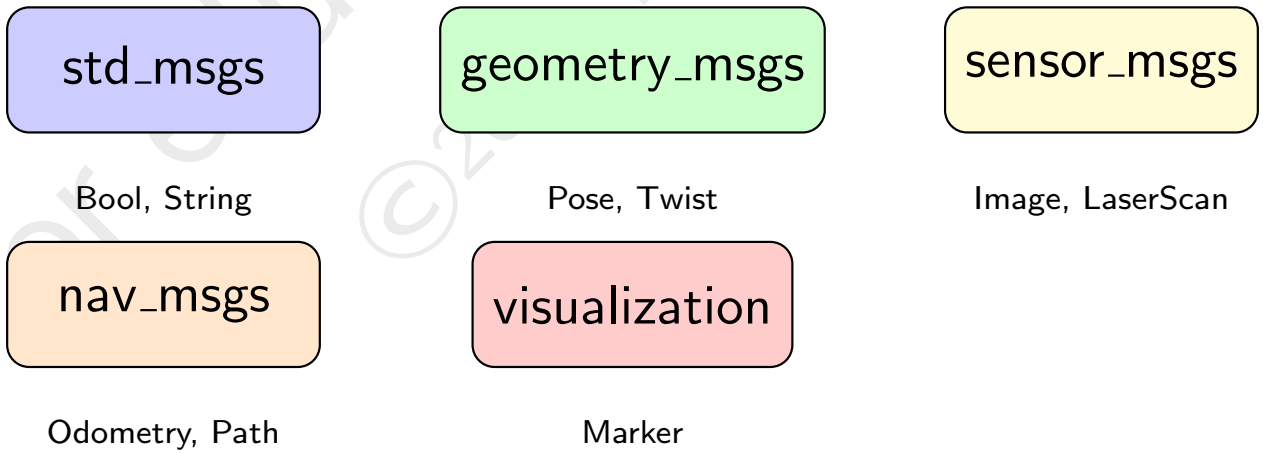


ROS 2 Messages

- **Data structures** used for communication between nodes
- Defined in **.msg** files
- **Standardized** communication
- **Nested** message structures

- **Language** independent
- **Automatic** serialization
- **Backward** compatible
- **Tool** integration (RViz, rqt)

Standard messages



ROS Standard Messages - std_msgs

Primitive Types

Message	C++ equivalent
Bool	bool
Int8	int8_t
Int16	int16_t
Int32	int32_t
Int64	int64_t
UInt8	uint8_t
UInt16	uint16_t
UInt32	uint32_t
UInt64	uint64_t
Float32	float
Float64	double
String	std::string

Special Types

Header Standard metadata

```
1 uint32 seq
2 time stamp
3 string frame_id
```

ColorRGBA RGBA color

```
1 float32 r
2 float32 g
3 float32 b
4 float32 a
```

Empty No data

Message definition example

Pose.msg

```
# A representation of pose in free space
Point position
Quaternion orientation
```

Point.msg

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Quaternion.msg

```
# This represents an orientation in quaternion form.
float64 x 0
float64 y 0
float64 z 0
float64 w 1
```

[12]

[12] github.com/ros2/common_interfaces/blob/rolling/geometry_msgs/msg/Pose.msg

Publisher - code example

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class PublisherNode(Node):
    def __init__(self):
        super().__init__('publisher')
        self.publisher = self.create_publisher(String, 'topic')
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello'
        self.publisher.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
```

[13]

[13] github.com/ros2/common_interfaces/blob/rolling/geometry_msgs/msg/Pose.msg

Publisher - code example

```
1
2 def main(args=None):
3     rclpy.init(args=args)
4     publisher = PublisherNode()
5     rclpy.spin(publisher)
6     rclpy.shutdown()
7
8 if __name__ == '__main__':
9     main()
```

14

[14] https://github.com/ros2/examples/blob/rolling/rclpy/topics/minimal_publisher/

Subscriber - code example

```
1  import rclpy
2  from rclpy.node import Node
3  from std_msgs.msg import String
4
5  class SubscriberNode(Node):
6      def __init__(self):
7          super().__init__('subscriber')
8          self.subscription = self.create_subscription(
9              String,
10             'topic_name',
11             self.listener_callback,
12             10)
13
14     def listener_callback(self, msg):
15         self.get_logger().info('I received: "%s"' % msg.
               data)
```

Subscriber - code example

```
1  def main(args=None):
2      rclpy.init(args=args)
3      subscriber = SubscriberNode()
4      rclpy.spin(subscriber)
5      rclpy.shutdown()
6
7  if __name__ == '__main__':
8      main()
```

ROS 2 CLI: Topic & Node Utilities

Command	Description
`ros2 run my_pkg my_node --ros-args -r old:=new`	Remap topic names at runtime
`ros2 topic echo /topic --once`	Peek at one message
`ros2 topic list -t`	List topics with their types
`ros2 topic hz /topic`	Check publishing frequency
`ros2 topic info /topic`	Show publisher and subscriber count
`ros2 topic bw /topic`	Check bandwidth usage
`rqt_graph`	Visualize the node network

Summary

1. ROS: Introduction & History
2. ROS 1 vs ROS 2
3. ROS 2 Nodes & Topics

What's next?

1. Communication: services, actions
2. Workspaces and packages
3. Nodes parametrization: config files
4. Launch system

The end

Thank you for your attention!