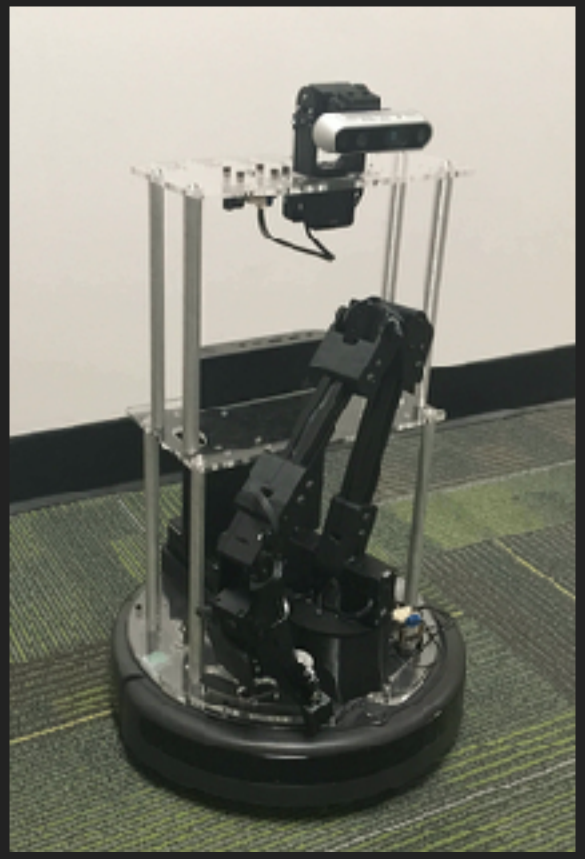
MICHAEL FERGUSON

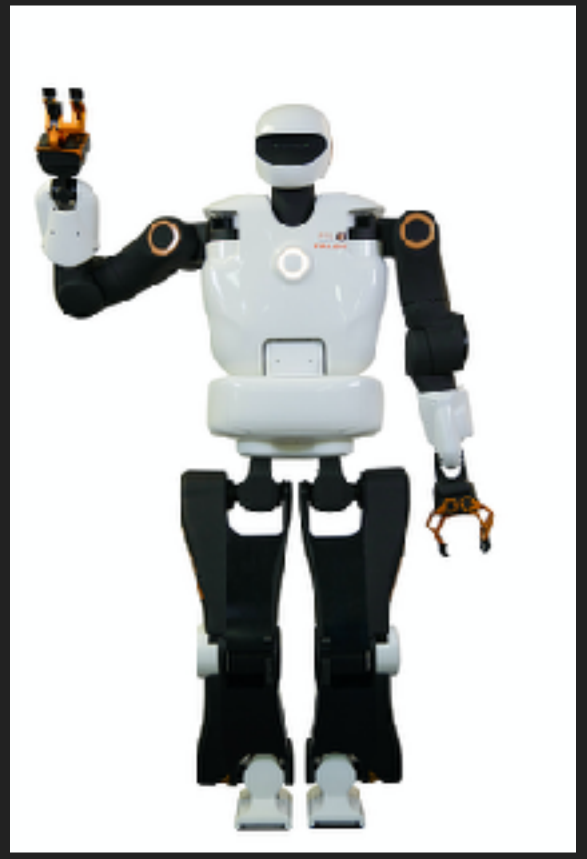
# GETTING STARTED WITH ROS2

## WHO AM I?

## WHY ROS?

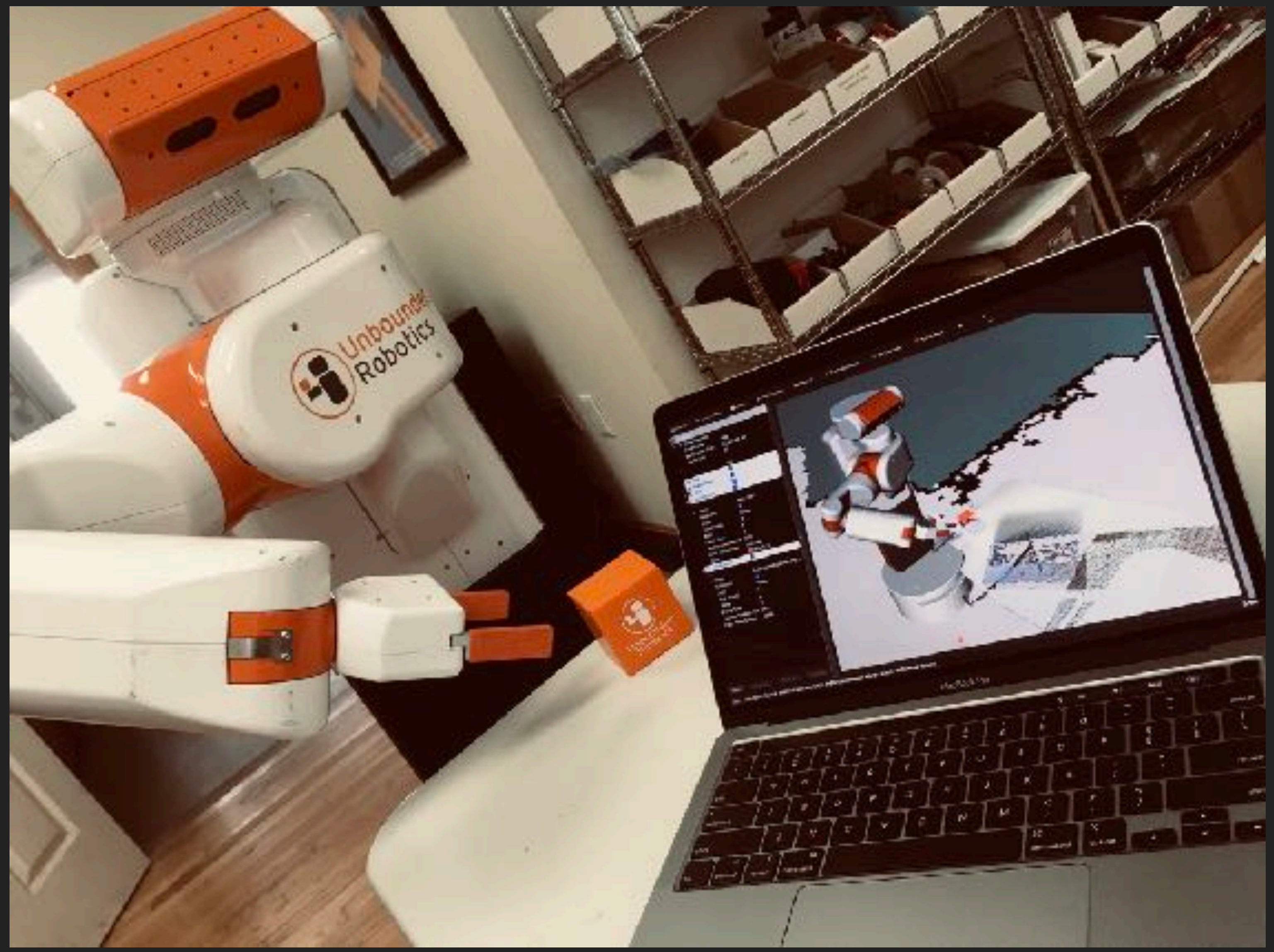
- Message passing + build system, leads to:
- Code re-use and sharing, leads to:
- Large community and robot-related code base



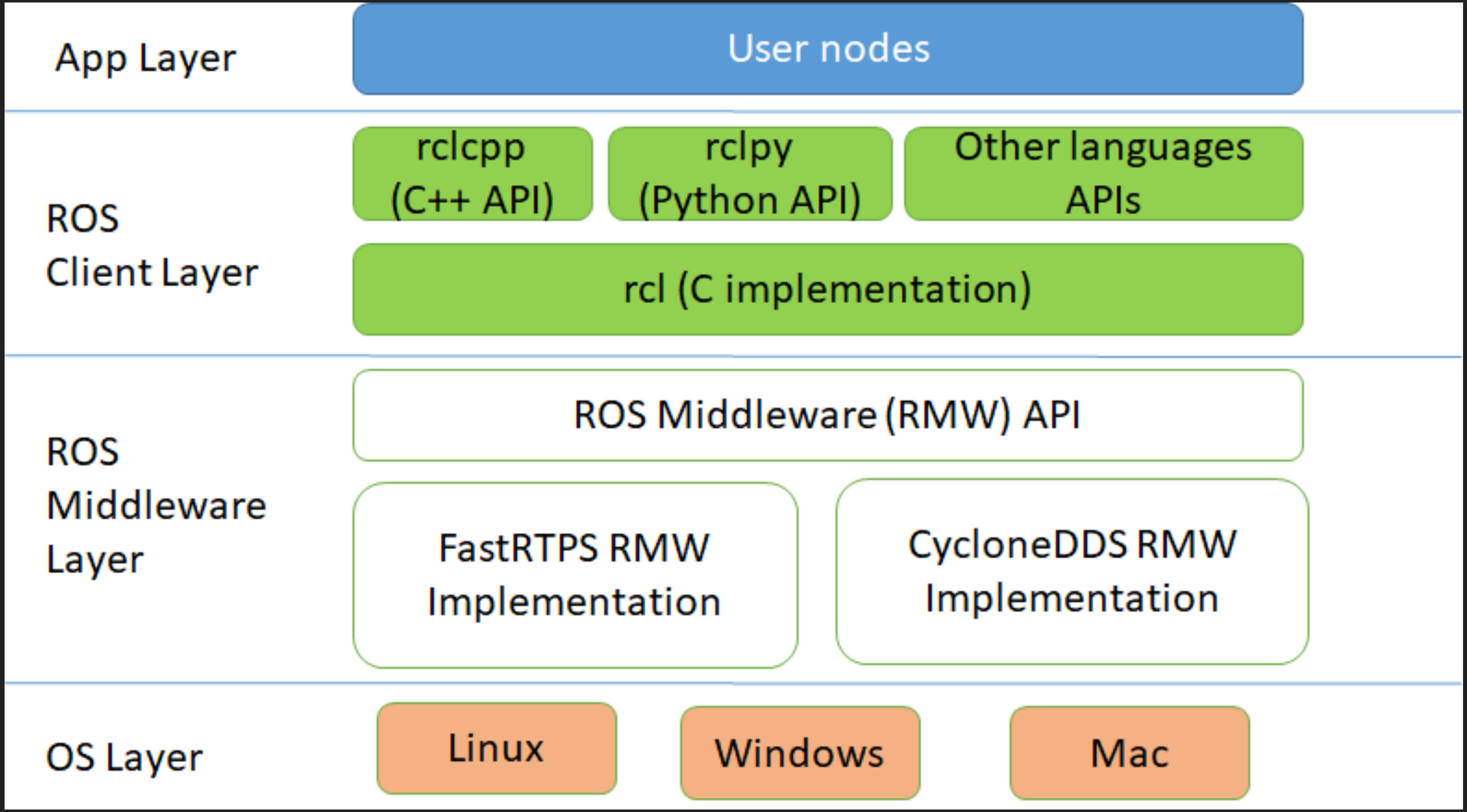


## WHY ROS2?

- Replace custom, aging messaging infrastructure
  - Greater ability to tune for reliability, performance
  - Industry standard rather than custom (DDS)
- Cross platform support (Linux, MacOSX, Windows)
  - Modern C++, very little Boost
  - Ubuntu is still best supported environment



## ROS2 ARCHITECTURE

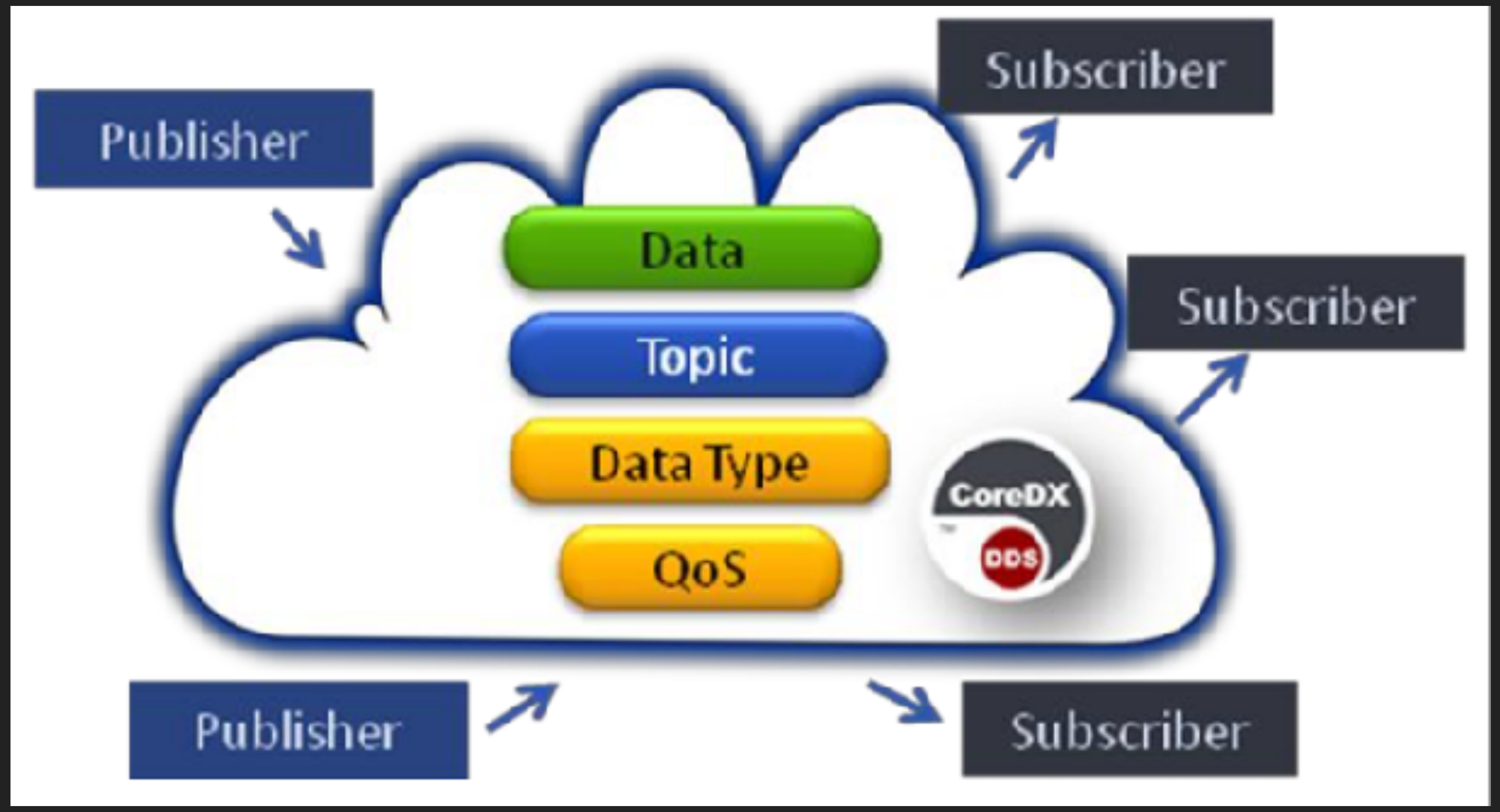


## DOUBLE EDGE SWORDS: ROS2 VS ROS1

- Documentation: not as complete, but also not out of date after 10+ years
- Ability to tune: sometimes out-of-box experience is challenging
- It’s new: cool features, but also buggy/incomplete

## WHAT IS DDS?

- Message passing architecture
- Widely used
- No rosmaster
- Shared memory transport
- Quality of Service (QoS)



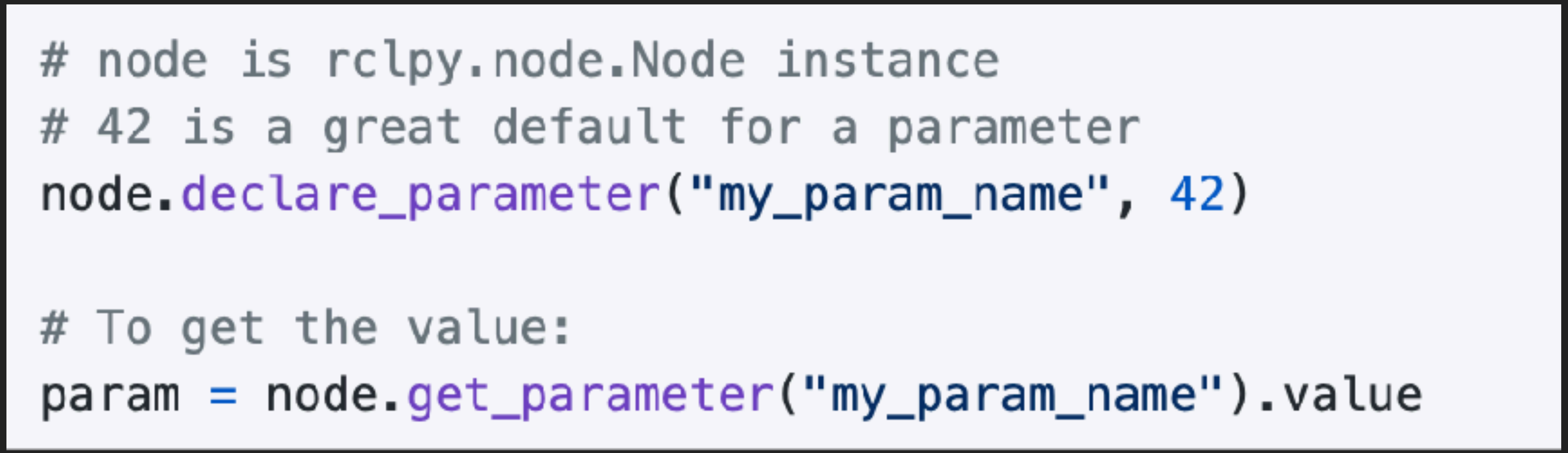
## WHAT'S DIFFERENT?

- No rosmaster required
  - There is a ros2 daemon - but it is just for efficiency
  - Also means no common parameter server
    - Changes to robot_description (it's a topic)

https://design.ros2.org/articles/changes.html

## WHAT'S DIFFERENT?

- All parameters are dynamic
  - No more dynamic_reconfigure
  - Parameter services and events
  - All parameters must be declared - but allows introspection



```
# node is rclpy.node.Node instance
# 42 is a great default for a parameter
node.declare_parameter("my_param_name", 42)

# To get the value:
param = node.get_parameter("my_param_name").value
```

https://design.ros2.org/articles/ros_parameters.html

## WHAT'S DIFFERENT?

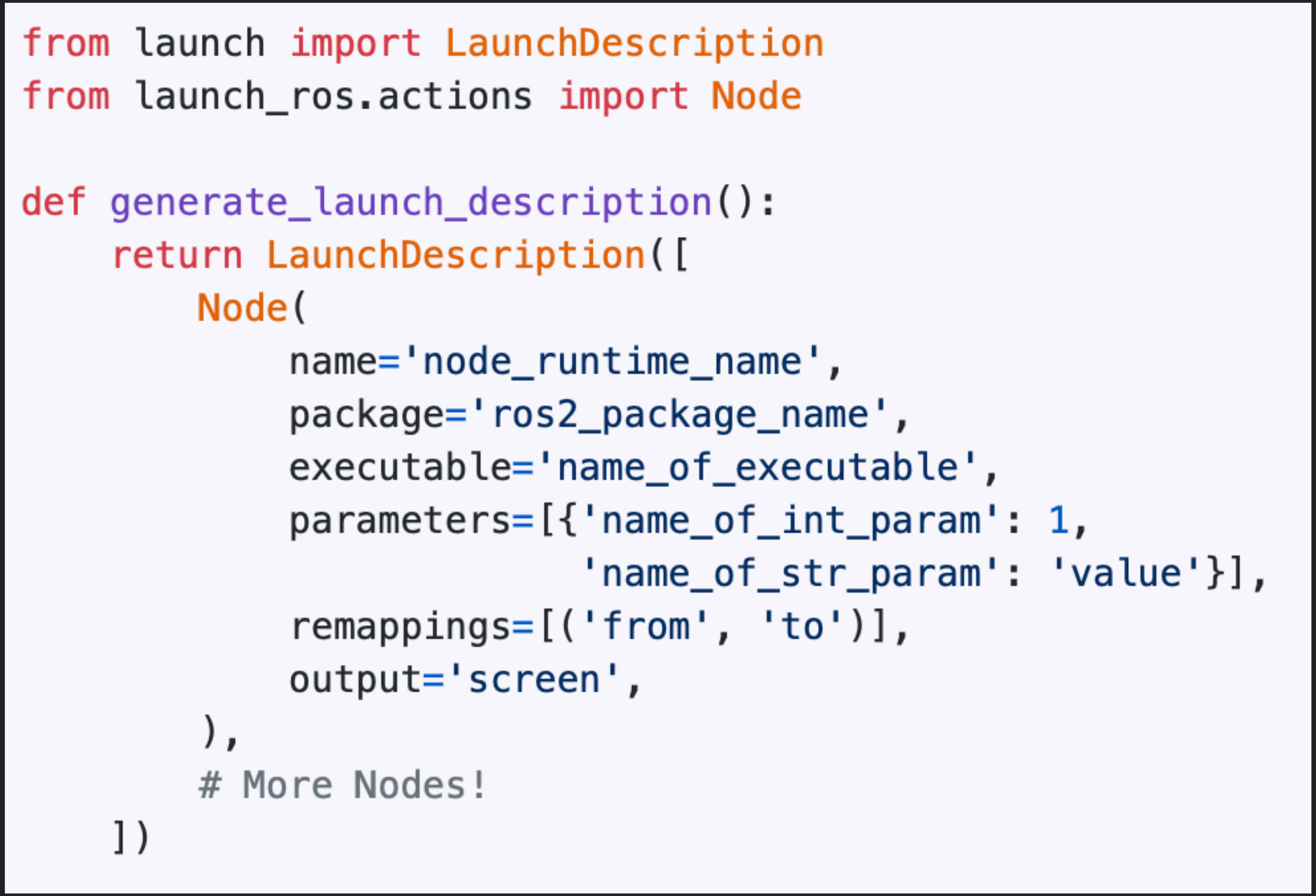
- Catkin is replaced by Ament/colcon
  - Even closer to pure CMake
  - No devel workspace - colcon build –symlink-install
  - Need multiple find_package()
  - No ${ament_LIBRARIES} - use ament_target_dependencies()
  - Support for pure Python packages (no CMake needed!)

## WHAT'S DIFFERENT?

- Nodelets become components
  - Can still put multiple nodes in a single executable at runtime
  - But unlike ROS1, introspection on individual nodes works!
  - https://github.com/mikeferguson/ros2_cookbook/blob/main/rclcpp/nodes.md#creating-a-component

## WHAT'S DIFFERENT?

- roslaunch2 supports XML, Python and YAML.



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

def generate_launch_description():
    return LaunchDescription([
        Node(
            name='node_runtime_name',
            package='ros2_package_name',
            executable='name_of_executable',
            parameters=[{'name_of_int_param': 1,
                         'name_of_str_param': 'value'}],
            remappings=[('from', 'to')],
            output='screen',
        ),
        # More Nodes!
    ])
```

## WHAT'S THE SAME?

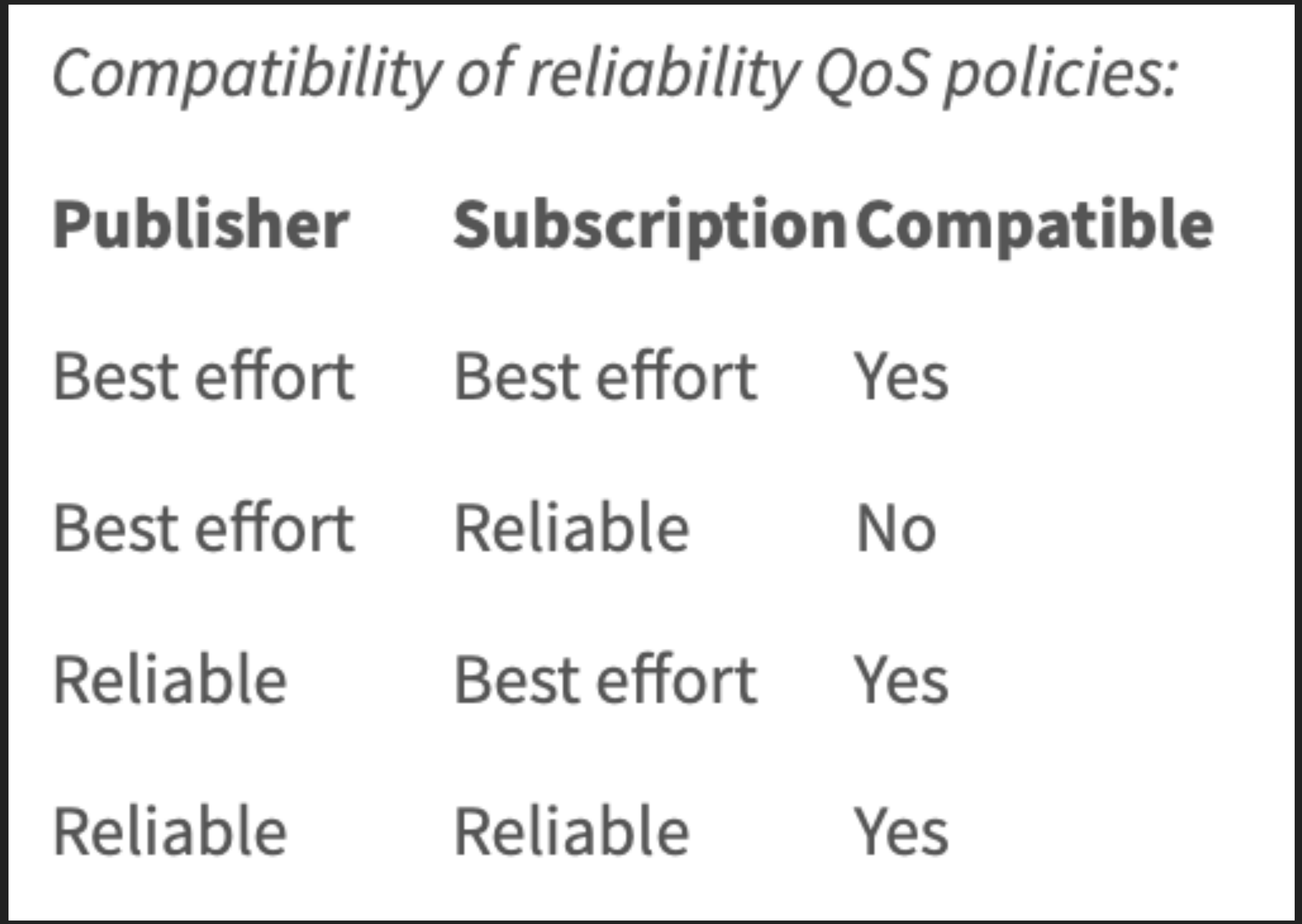
- Still mostly C++/Python
  - But, better feature parity between rclpy and rclcpp since both use rcl
- Command line tools
  - ros2 <node, topic, launch> <info, list>

## QUALITY OF SERVICE

- History/Depth: how many messages to keep (similar to ROS1 queue_size)
- Reliability: *best effort* or *reliable*
- Durability: *volatile* or *transient local* (similar to ROS1 latching)
- Also: Deadline, Lifespan, Liveliness, Lease Duration
- Profiles
  - Default: reliable, keep last 10, volatile
  - Sensor Data: best effort, smaller queue size

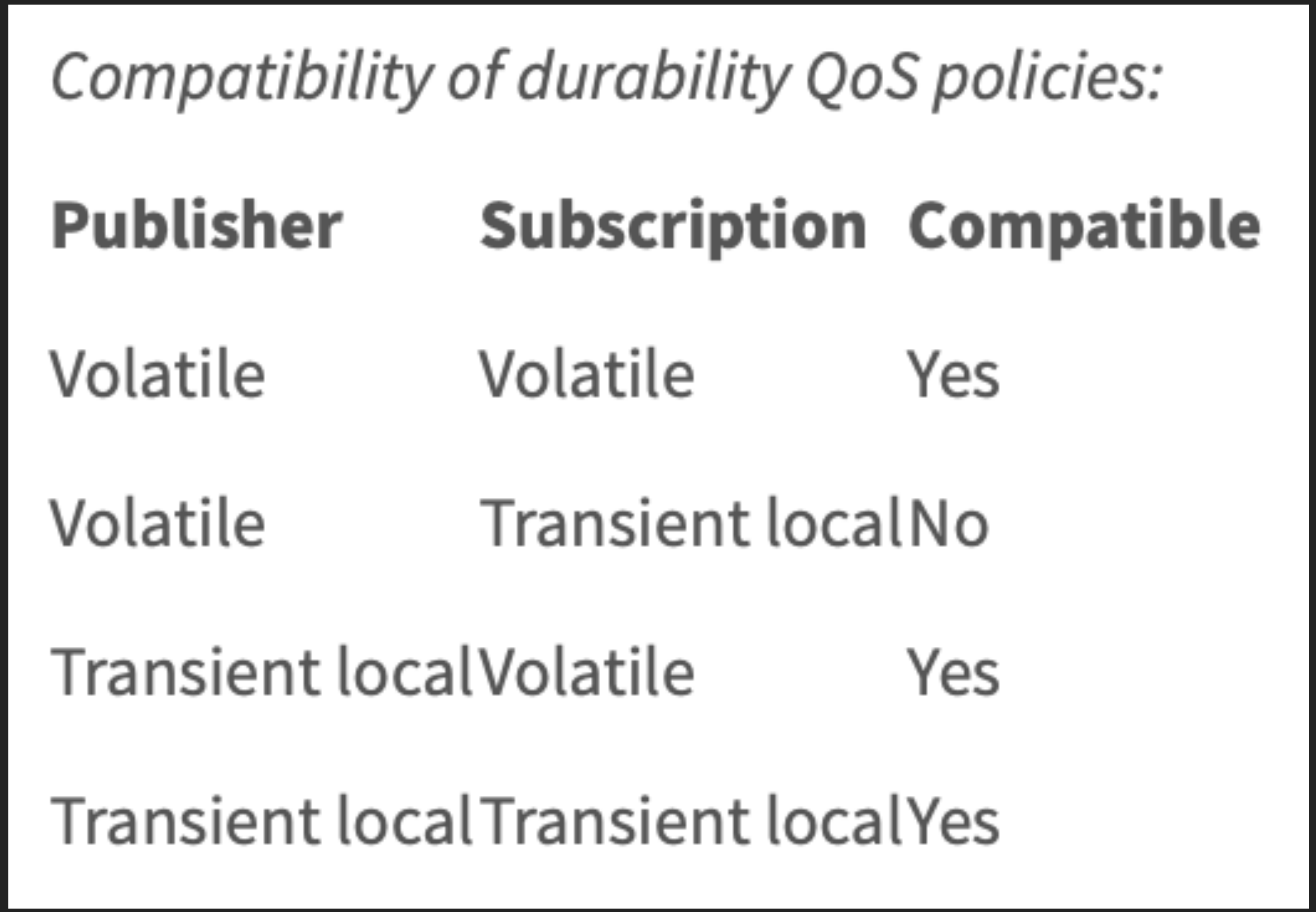
https://index.ros.org/doc/ros2/Concepts/About-Quality-of-Service-Settings/

## QUALITY OF SERVICE



*Compatibility of reliability QoS policies:*

| **Publisher** | **Subscription** | **Compatible** |
|---|---|---|
| Best effort | Best effort | Yes |
| Best effort | Reliable | No |
| Reliable | Best effort | Yes |
| Reliable | Reliable | Yes |



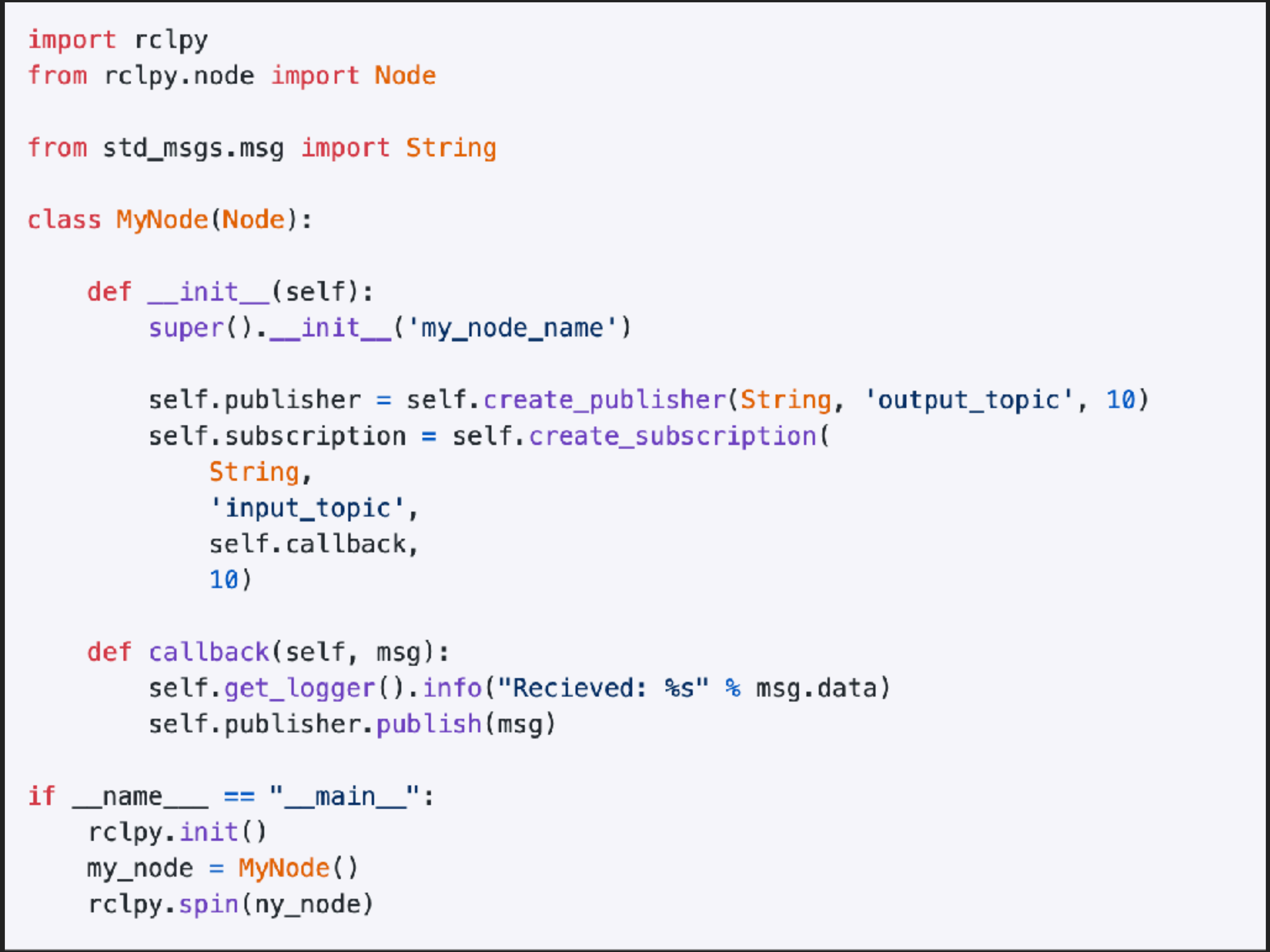
*Compatibility of durability QoS policies:*

| **Publisher** | **Subscription** | **Compatible** |
|---|---|---|
| Volatile | Volatile | Yes |
| Volatile | Transient local | No |
| Transient local | Volatile | Yes |
| Transient local | Transient local | Yes |

- And now: a demo with RVIZ2 on MacOSX

## RCLPY: CREATE A NODE

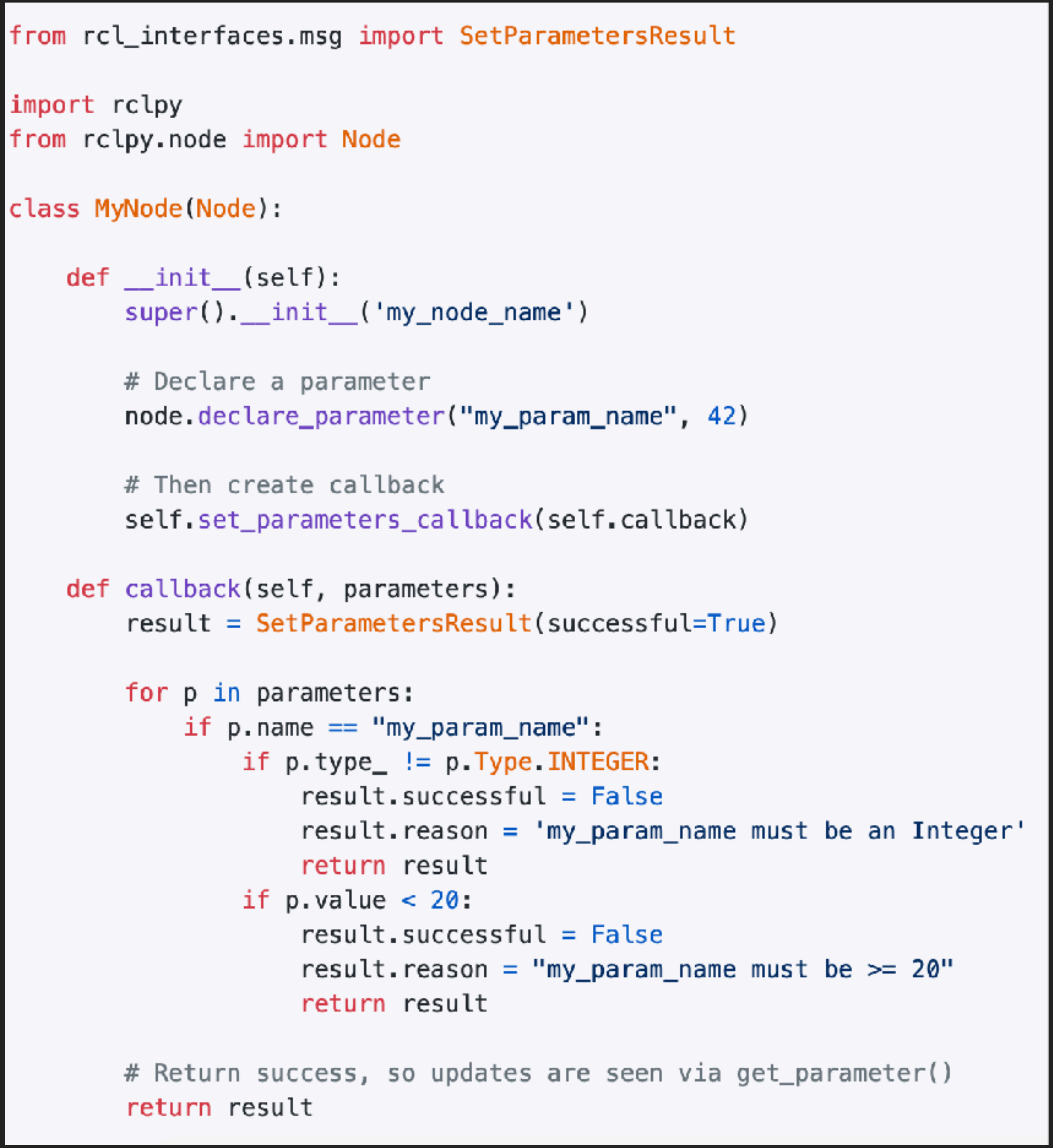
- Python3 only!
- Derive from Node
- Spin on the node
- Threading is very different from ROS1



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

from std_msgs.msg import String

class MyNode(Node):

    def __init__(self):
        super().__init__('my_node_name')

        self.publisher = self.create_publisher(String, 'output_topic', 10)
        self.subscription = self.create_subscription(
            String,
            'input_topic',
            self.callback,
            10)

    def callback(self, msg):
        self.get_logger().info("Recieved: %s" % msg.data)
        self.publisher.publish(msg)

if __name___ == "__main__":
    rclpy.init()
    my_node = MyNode()
    rclpy.spin(ny_node)
```

## RCLPY: PARAMETERS

- Must be declared
- A set_parameters callback allows you to limit changes.



```python
from rcl_interfaces.msg import SetParametersResult

import rclpy
from rclpy.node import Node

class MyNode(Node):

    def __init__(self):
        super().__init__('my_node_name')

        # Declare a parameter
        node.declare_parameter("my_param_name", 42)

        # Then create callback
        self.set_parameters_callback(self.callback)

    def callback(self, parameters):
        result = SetParametersResult(successful=True)

        for p in parameters:
            if p.name == "my_param_name":
                if p.type_ != p.Type.INTEGER:
                    result.successful = False
                    result.reason = 'my_param_name must be an Integer'
                    return result
                if p.value < 20:
                    result.successful = False
                    result.reason = "my_param_name must be >= 20"
                    return result

        # Return success, so updates are seen via get_parameter()
        return result
```

## ROSPY TO RCLPY

- rospy.time.now() => NodeInstance.get_clock().now()
- msg.header.stamp = t => msg.header.stamp = t.to_msg()

## OTHER FEATURES / CHANGES

- Lifecycle nodes:
- rosbag2:
  - Uses sqlite (bag is a directory, not a file)
  - Automatic QoS in Foxy
- Security

## DEMOS

- UBR-1 building a map with slam_toolbox
- UBR-1 running Navigation2
  - Better control of recovery behaviors through behavior trees
- UBR-1 running MoveIt2

## RMW IMPLEMENTATIONS

- Multiple vendors have implemented DDS
- Default is FastRTPS (FastDDS)
- CycloneDDS is up and coming
- Set RMW_IMPLEMENTATION environment variable
- CYCLONEDDS_URI may be required (esp. if multiple interfaces)
  - https://www.robotandchisel.com/2020/08/12/cyclonedds/

## NOTES ON NETWORKING

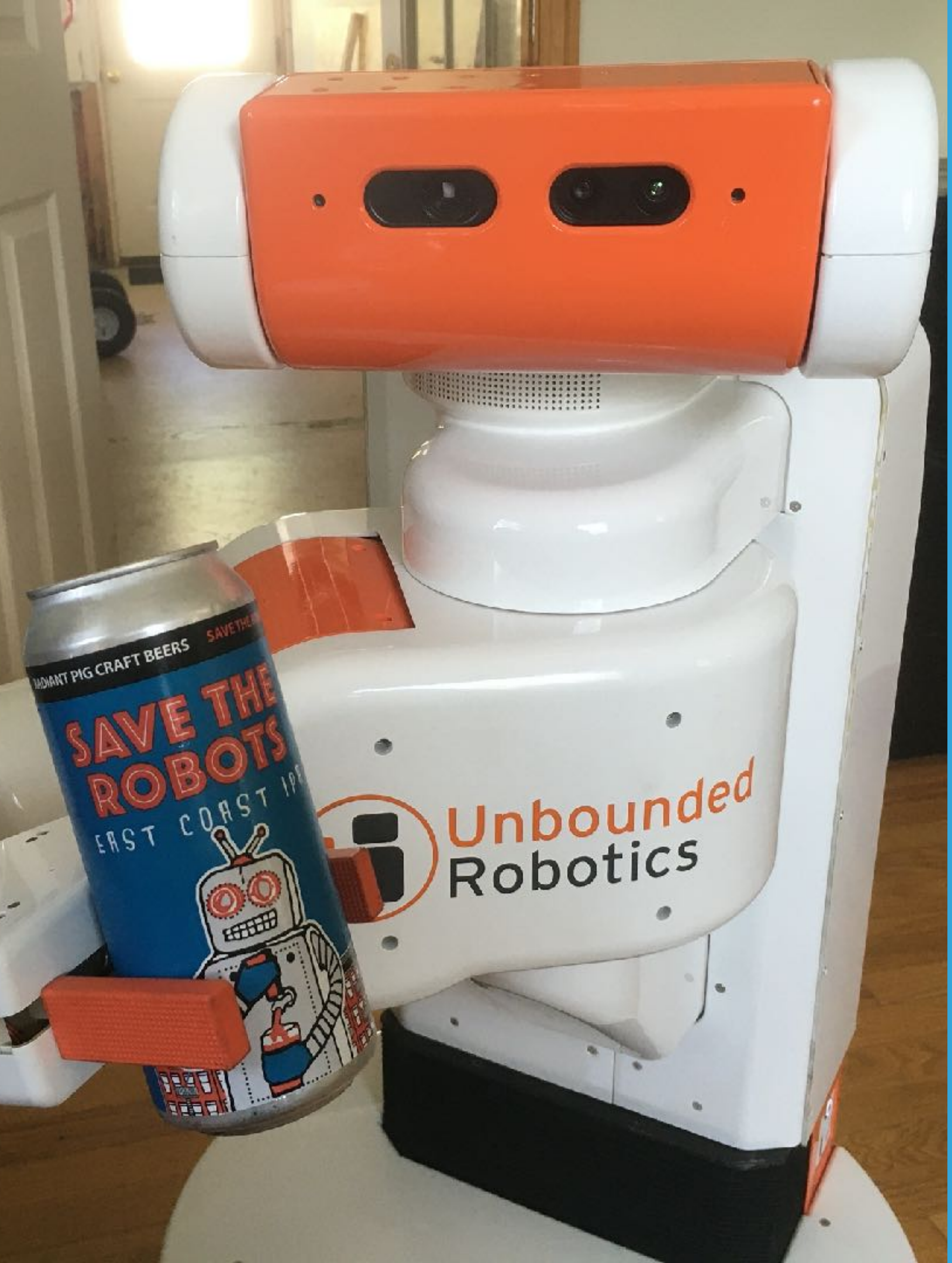
- export ROS_DOMAIN_ID=0
- export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
- export CYCLONEDDS_URI="/path/to/file"
- export CYCLONEDDS_URI="<xml>"

## NOTES ON CYCLONEDDS_URI

```xml
<CycloneDDS>
  <Domain>
    <General>
      <!-- Explicitly set network interface -->
      <NetworkInterfaceAddress>wlp2s0</NetworkInterfaceAddress>
      <!-- Use multicast for discovery only -->
      <AllowMulticast>spdp</AllowMulticast>
    </General>
    <Discovery>
      <!-- This tag has to be the same on each machine -->
      <Tag>my_robot_name</Tag>
    </Discovery>
  </Domain>
</CycloneDDS>
```

## STATE OF ROS2

- Message passing is generally solid
- What’s missing:
  - Lazy subscribers
  - Many drivers have not been ported and/or merged
  - Mid-level utilities are buggy but improving (image_pipeline, etc)
  - High-level apps (Nav, MoveIt) are under significant development



ROBOTANDCHISEL.COM

@TheRealFergs