robotics&automation society

# Intro to ROS

Micah Nye
Autonomous Racing Lead

## Agenda

robotics&automation society

- What is ROS
- ROS Fundamentals
- Tutorial using ROS CLI
- Tutorial writing ROS
- Advanced Demo
- Takeaways & Q&A

Questions always!

## Key Objectives

- Understanding of what ROS is and why we use it
- Good foundation built for ROS fundamentals
- Understanding of ROS CLI
- Entry-level comfort with writing ROS nodes

robotics&automation society

# What is ROS?

## What is ROS?

- The Robot Operating System (ROS) is a set of **software libraries and tools** for building robot applications

## What is ROS NOT?

- An operating system
- A language
- The holy grail

## What is ROS?

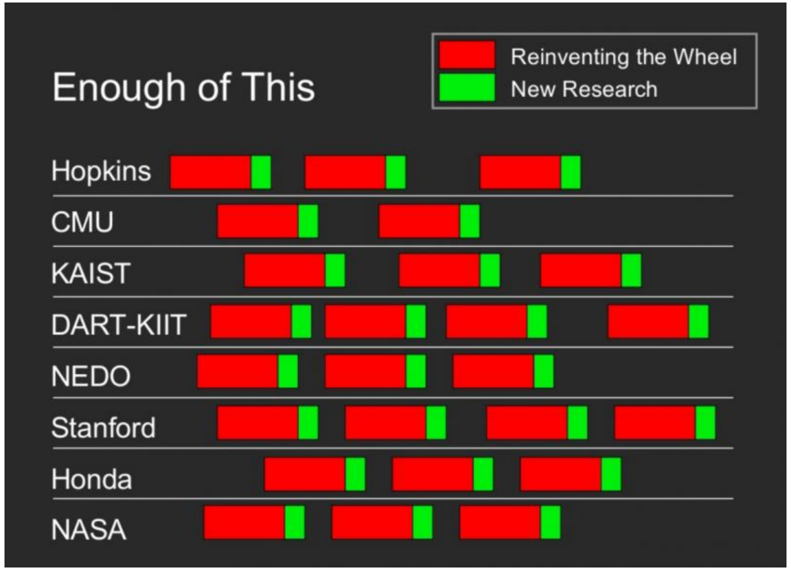
- A build and installation system
- A communication network
- A development environment manager
- A single or multi-machine launch system
- A single or multi-machine process “manager”
- A parameter managing service
- A community of open-source packages

robotics&automation society

## Why ROS?

# ROS History

- Initially ideated by Roboticists at Stanford, Keenan Wyrobeck and Eric Berger
  - Everyone **kept reinvented the wheel** and this stunted progress
  - Created to have a **unified, universal set of tools** for all researchers and students to use so they don't need to redo the grunt work every time
- Development taken on by Willow Garage in 2008
- Open Robotics takes on development after in 2013

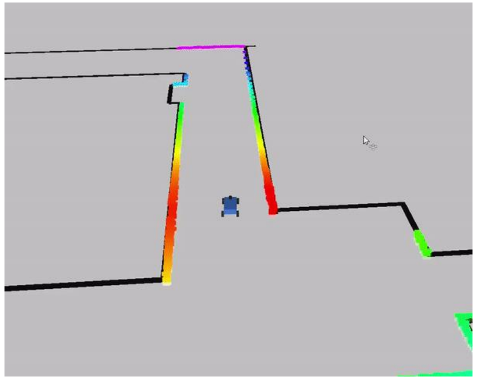
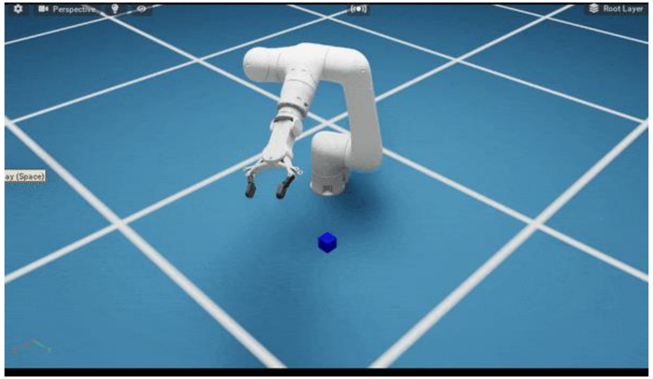


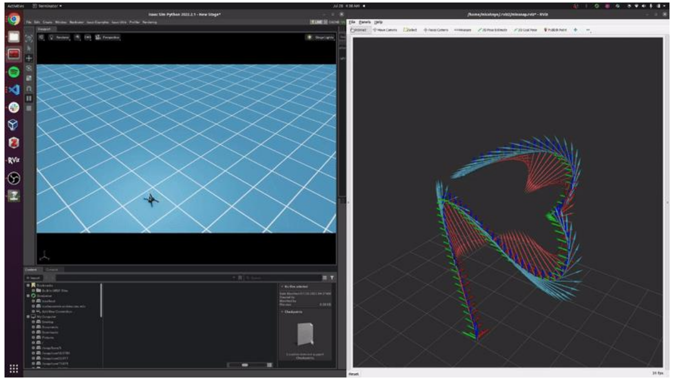
Taken from original slide deck of Wyrobeck and Berger
https://www.slideshare.net/KeenanWyrobek/personal-robotics-program-fund-fundraising-deck-from-2006

## Why ROS?

- Learning about robotics with many easy-to-use tools
- Unified interface for communication
- Fast and easy development for projects, research, industry*
- Global community with massive bank of knowledge
- Useful on many platforms – Linux, Windows, MacOS, Microcontrollers, even Android!
- Useful for many languages – C++, Python, C, Matlab, Java, etc.
- Open source

## Why ROS?

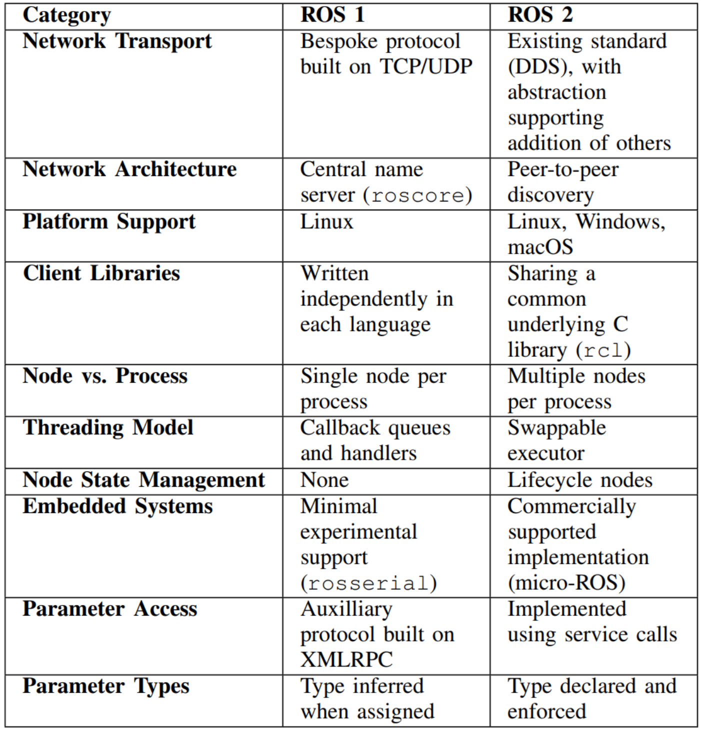





Top Left to Bottom Right:
MIT-PITT-RW, F1-tenth,
IsaacROS, Turtlebot3,
IsaacSim Quadrotor Sim,
CMU AirLab
TartanDrive/SARA Project

# ROS Versions – *Which ROS to use?*

- What ROS Version? ROS1 or ROS2?
  - **ROS2** *Strongly* Recommend
  - Why?
    - Active development
    - More platform support
    - Better networking (transport and architecture)
    - Better threading and process management
    - Better parameter management
  - See [paper from ROS devs]
  - See [article]



| Category | ROS 1 | ROS 2 |
|---|---|---|
| **Network Transport** | Bespoke protocol built on TCP/UDP | Existing standard (DDS), with abstraction supporting addition of others |
| **Network Architecture** | Central name server (`roscore`) | Peer-to-peer discovery |
| **Platform Support** | Linux | Linux, Windows, macOS |
| **Client Libraries** | Written independently in each language | Sharing a common underlying C library (`rcl`) |
| **Node vs. Process** | Single node per process | Multiple nodes per process |
| **Threading Model** | Callback queues and handlers | Swappable executor |
| **Node State Management** | None | Lifecycle nodes |
| **Embedded Systems** | Minimal experimental support (`rosserial`) | Commercially supported implementation (micro-ROS) |
| **Parameter Access** | Auxilliary protocol built on XMLRPC | Implemented using service calls |
| **Parameter Types** | Type inferred when assigned | Type declared and enforced |

TABLE I: Summary of ROS 2 features compared to ROS 1

Steven Macenski *et al.*, Robot Operating System 2: Design, architecture, and uses in the wild.*Sci. Robot.*7,eabm6074(2022).DOI:10.1126/scirobotics.abm6074

# ROS Versions – *Which Distribution (Distro)?*

| | "v9" | "v8" | "v7" | "v6" | |
|---|---|---|---|---|---|
| Distro | | | | | ■ ■ ■ |
| Platforms | Ubuntu 22.04<br>Windows 10<br>MacOS | Ubuntu 22.04<br>Windows 10<br>Ubuntu 20.04<br>MacOS | Ubuntu 20.04<br>Windows 10 | Ubuntu 20.04<br>Windows 10<br>MacOS | |
| EOL | Nov 2024 | May 2027 | EOL | EOL | |

## ROS Versions – *Which Distribution (Distro)?*

| | "v9" | "v8" | "v7" | "v6" | |
|---|---|---|---|---|---|
| Distro | | | | | . . . |
| Platforms | Ubuntu 22.04<br>Windows 10<br>MacOS | Ubuntu 22.04<br>Windows 10<br>Ubuntu 20.04<br>MacOS | Ubuntu 20.04<br>Windows 10 | Ubuntu 20.04<br>Windows 10<br>MacOS | |
| EOL | Nov 2024 | May 2027 | EOL | EOL | |

robotics&automation society

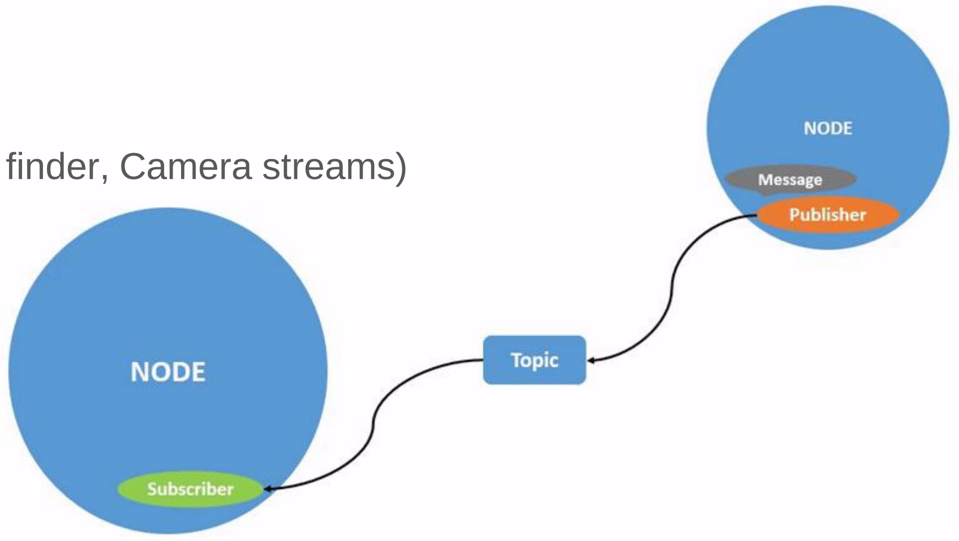
## How ROS works

## Overview

- Nodes
- Messages
- Topics
- Parameters
- Services
- Actions

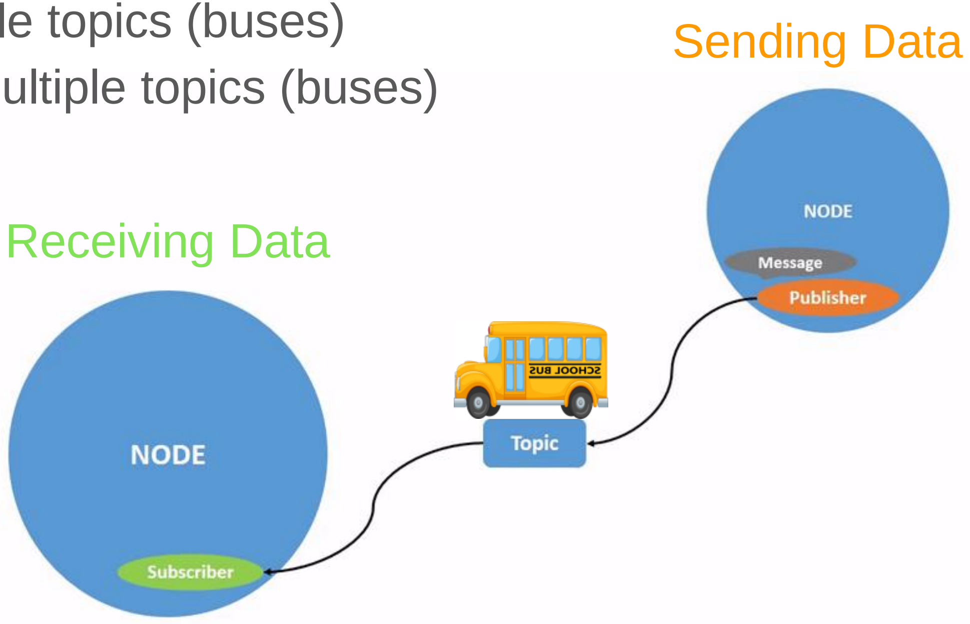
## Nodes

- The primary building block for robot software with ROS
- Executable processes that communicate over the ROS graph
- Examples of nodes:
  - Motor controller
  - Path follower
  - Sensor data receiver (Laser range finder, Camera streams)
  - Actuator or Sensor Drivers



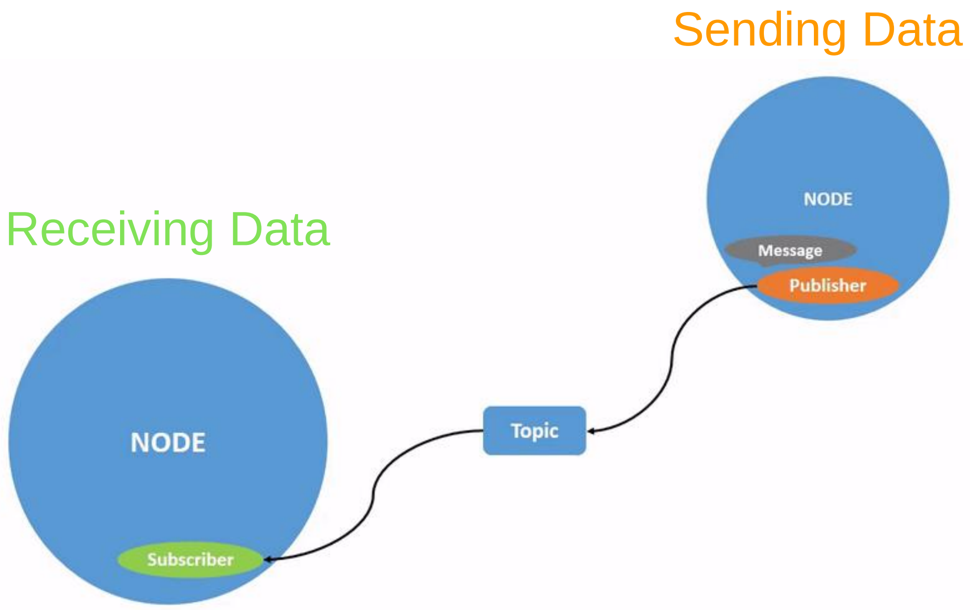
## Topics

- A “*bus*” that exchanges information (*data*) between nodes
- Nodes can send data on multiple topics (buses)
- Nodes can receive data from multiple topics (buses)
- *How is our data represented?*



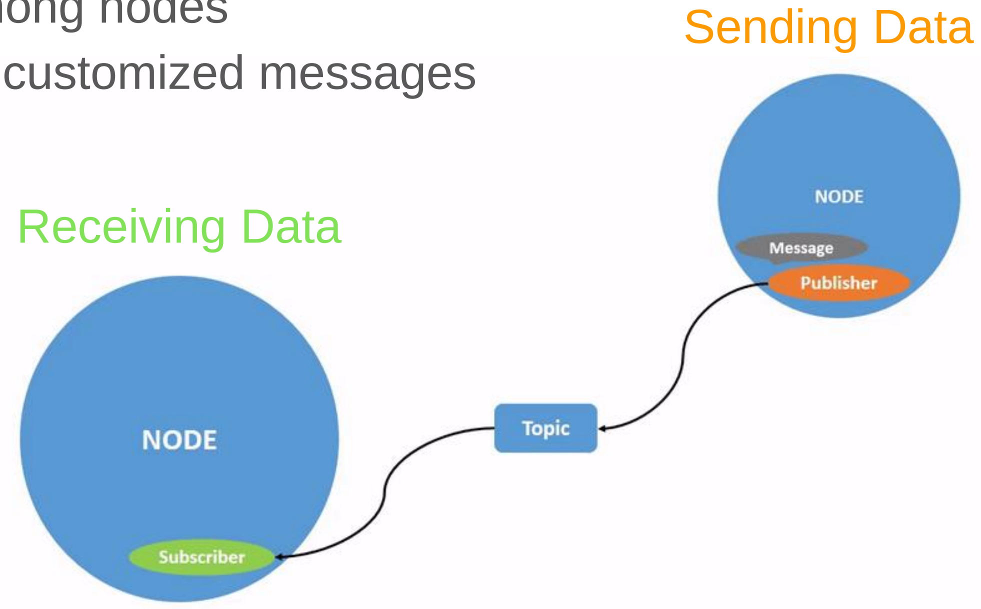
## Topics

- A topic name: `”/my_topic_name”`
- *How is our data represented?*



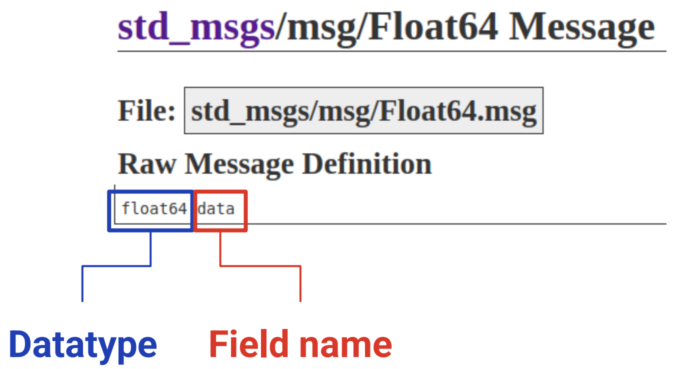
## Messages

- A data structure for node communication
- A container to transfer data among nodes
- A building block to make more customized messages



# Messages

Simple Message


std_msgs/msg/Float64 Message

File: std_msgs/msg/Float64.msg

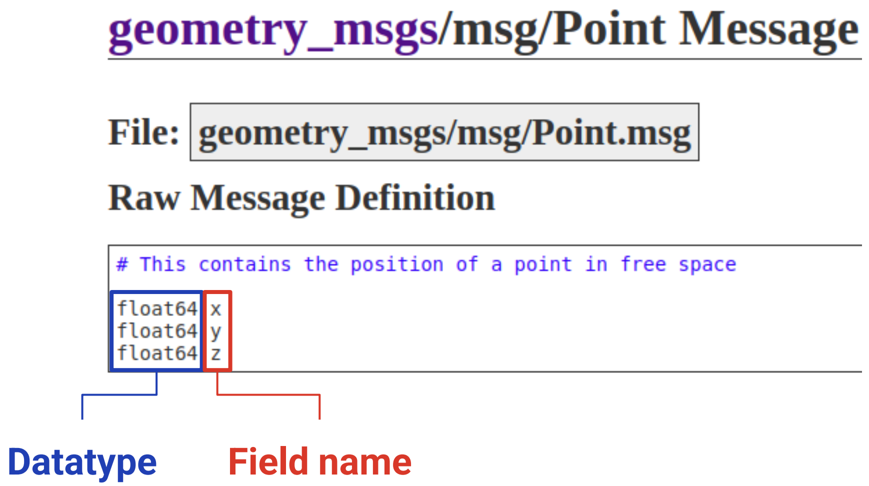
Raw Message Definition

```
float64 data
```

How do I unpack the data? `my_data = msg.data`

## Messages

Less Simple Message



**geometry_msgs/msg/Point Message**

**File:** geometry_msgs/msg/Point.msg

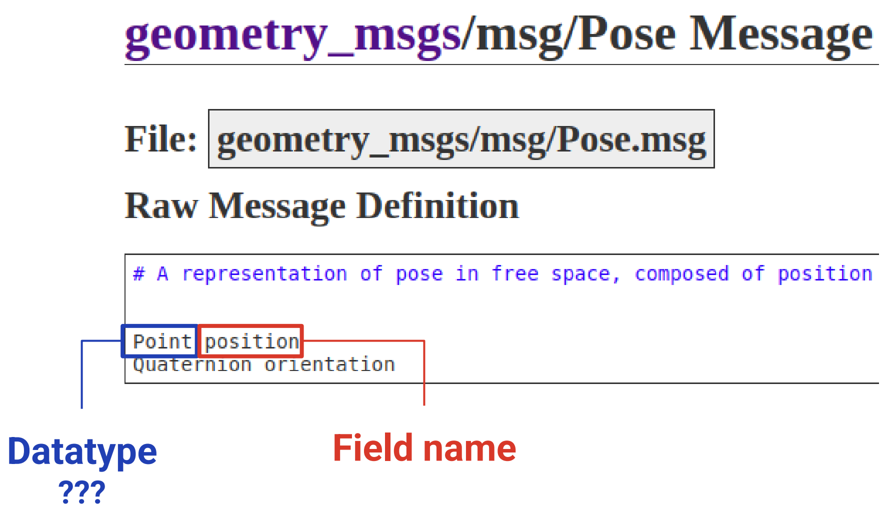
**Raw Message Definition**

```
# This contains the position of a point in free space

float64 x
float64 y
float64 z
```

How do I unpack the Point data? `x_value = msg.x`

## Messages

More Complex Message



### geometry_msgs/msg/Pose Message

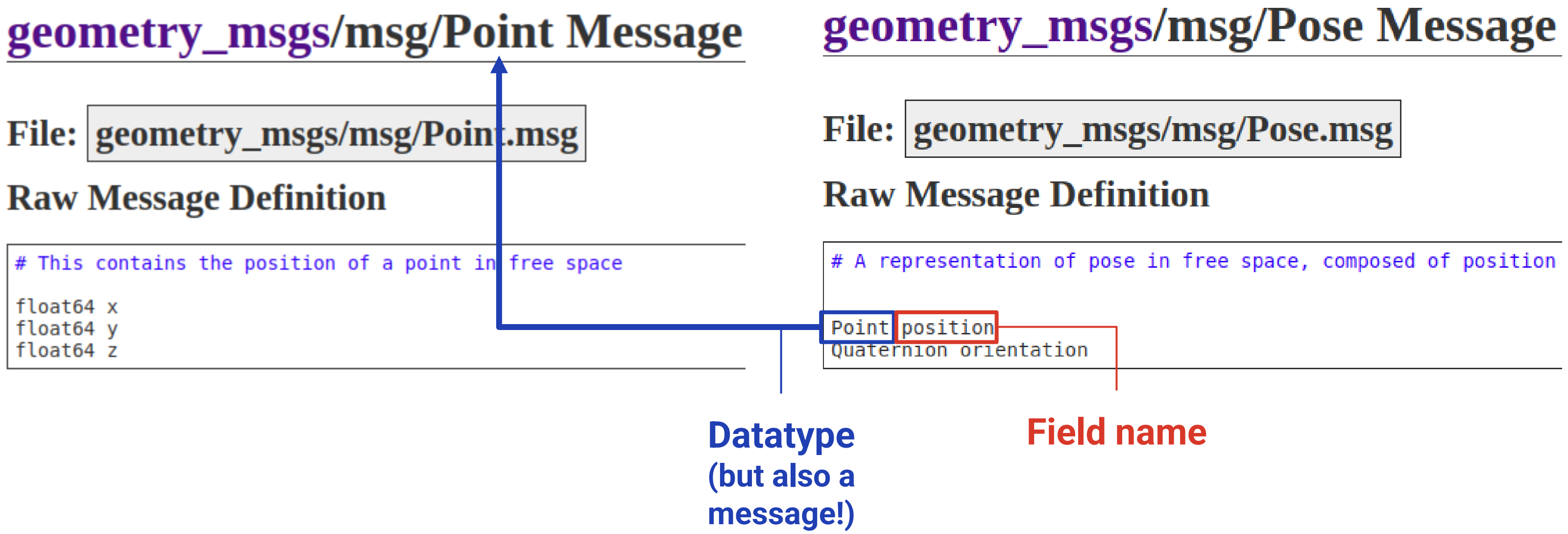
**File:** `geometry_msgs/msg/Pose.msg`

**Raw Message Definition**

```
# A representation of pose in free space, composed of position

Point position
Quaternion orientation
```

## Messages

### More Complex Message



geometry_msgs/msg/Point Message

File: geometry_msgs/msg/Point.msg

Raw Message Definition

```
# This contains the position of a point in free space

float64 x
float64 y
float64 z
```

geometry_msgs/msg/Pose Message

File: geometry_msgs/msg/Pose.msg

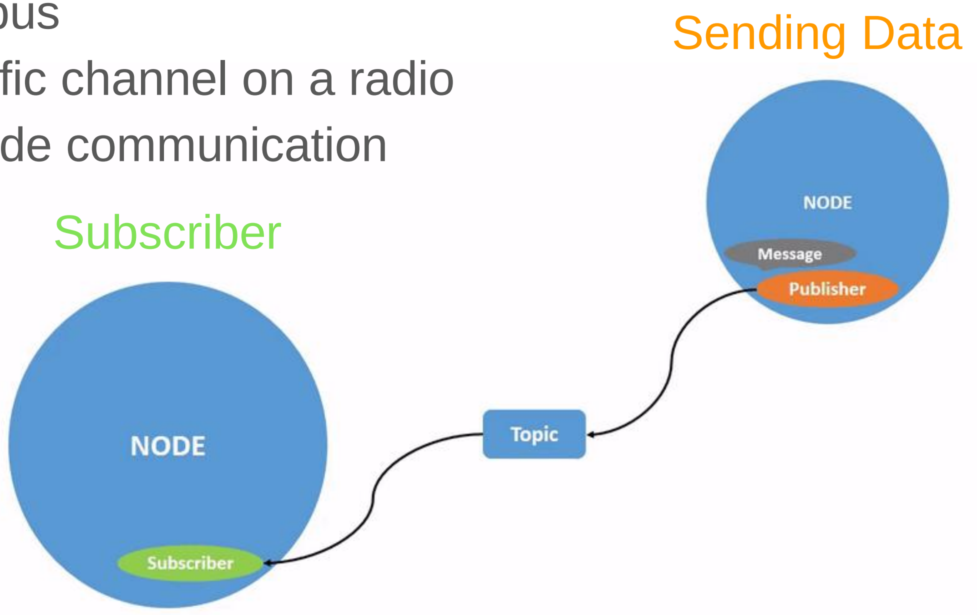
Raw Message Definition

```
# A representation of pose in free space, composed of position

Point position
Quaternion orientation
```

How do I unpack the Pose data? `x_value = msg.position.x`

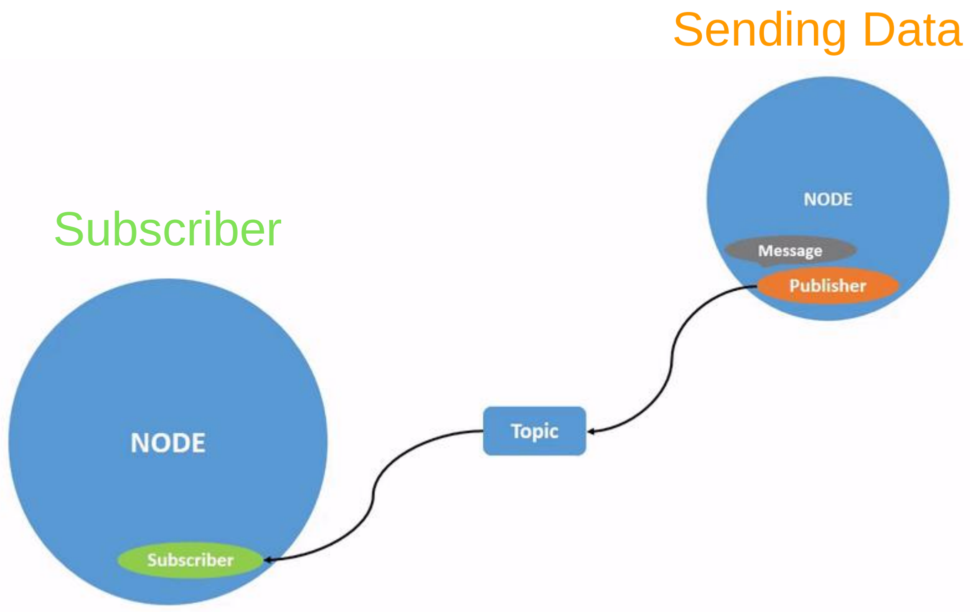
## Subscriber

- A component listening for a specific topic
  - The house waiting for the bus
  - A receiver tuned to a specific channel on a radio
- 1 of 2 major components for node communication



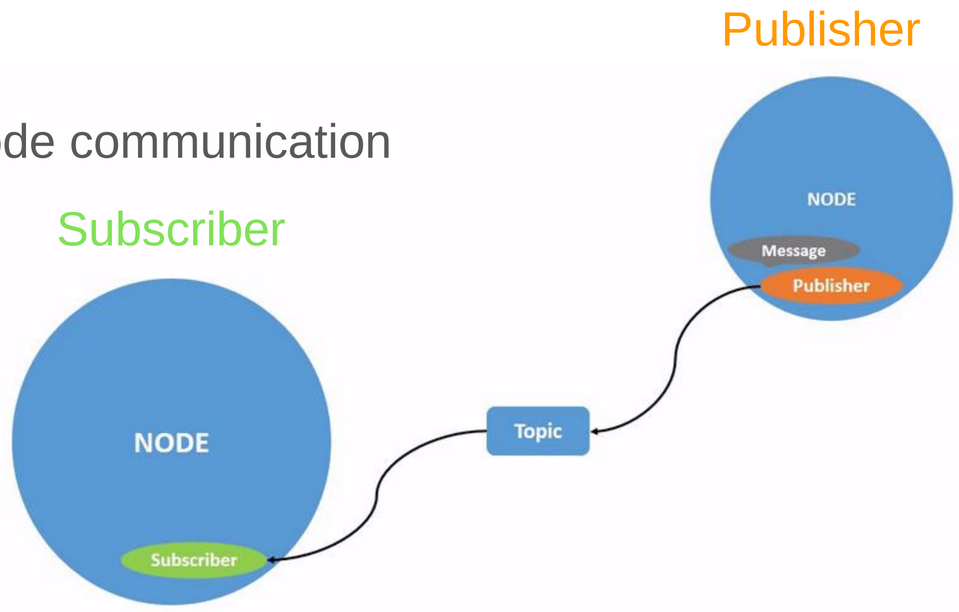
## Subscriber

- What are we listening to?
  - A specific topic name
- How do we receive the data?
  - From a callback function
- How do we unpack the data?
  - Read the message



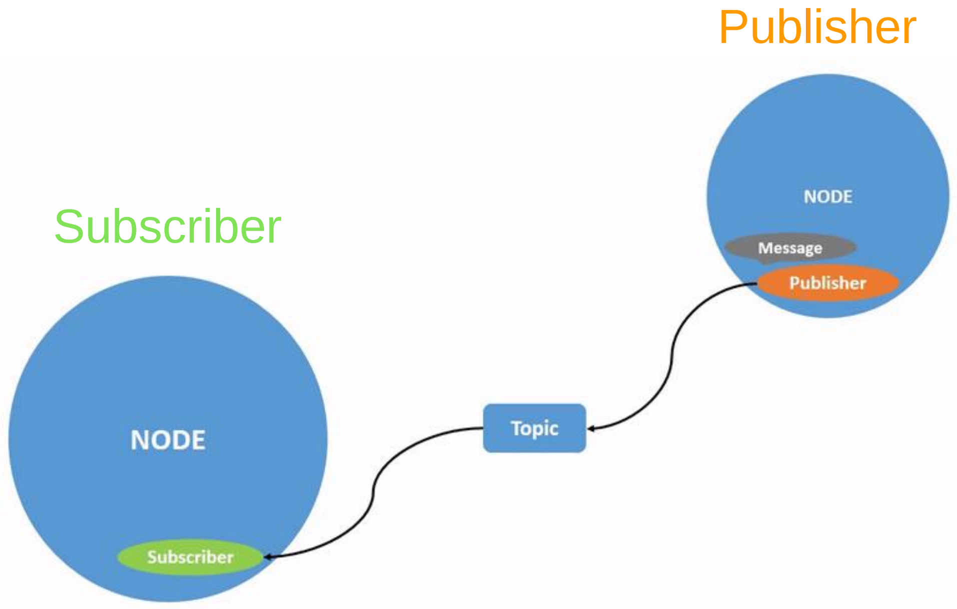
## Publisher

- A component sending data on that specific topic
  - A transmitter
  - A broadcaster on a radio
- 1 of 2 major components for node communication



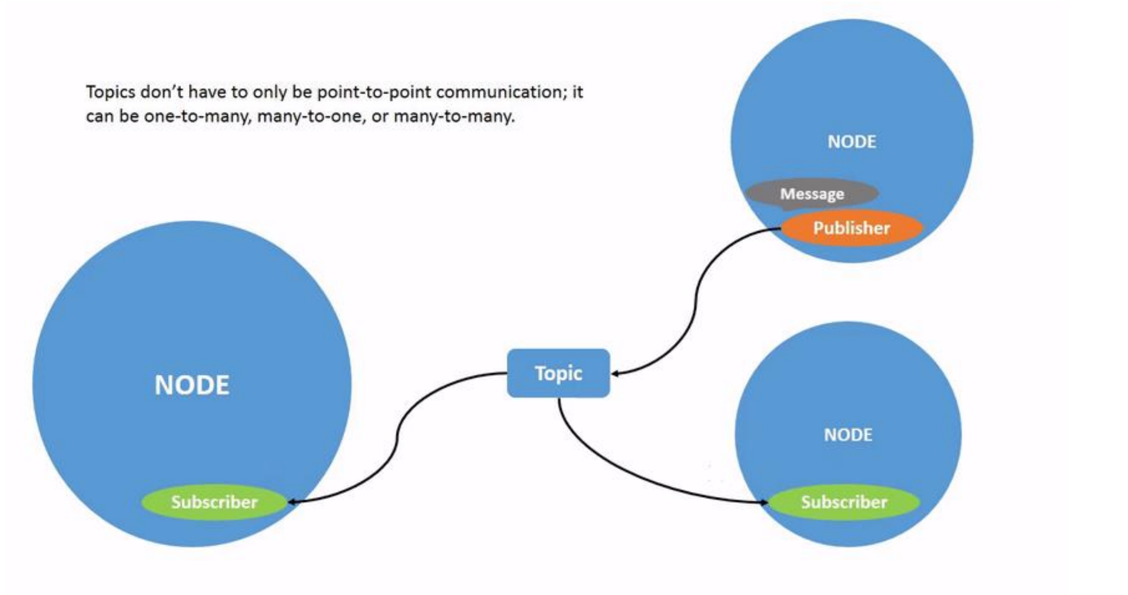
## Publisher

- What are we sending data on?
  - A specific topic name
- How do we send the data?
  - From a publisher object
- How do we pack the data?
  - Create and populate a message



## Multi-node graphs

- Many nodes can listen and publish to the same topics
- This builds complex robotic architectures

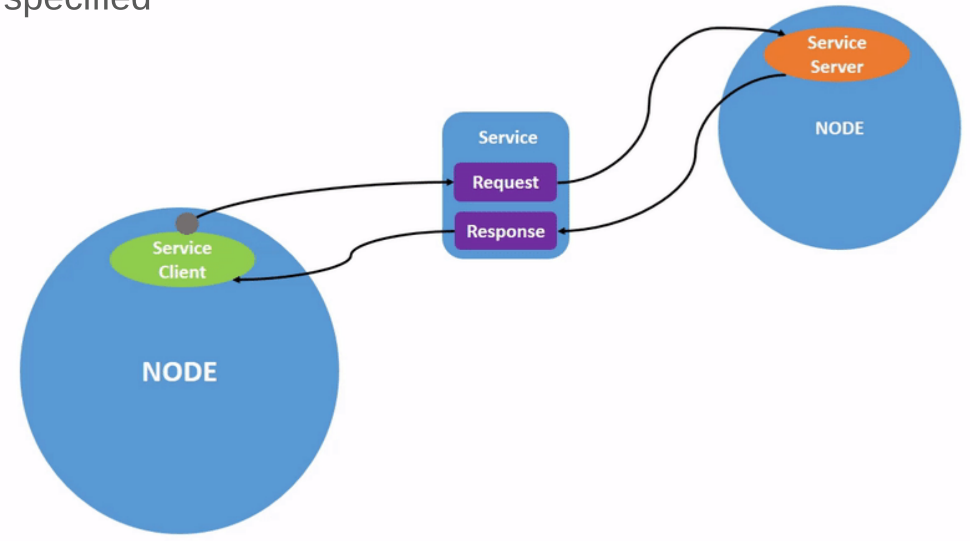

Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.
NODE
Message
Publisher
NODE
Subscriber
Topic
NODE
Subscriber

## Parameters

- “Node settings” – A configurable value for a node
- A way to update numbers/settings in real-time without having to rebuild your package

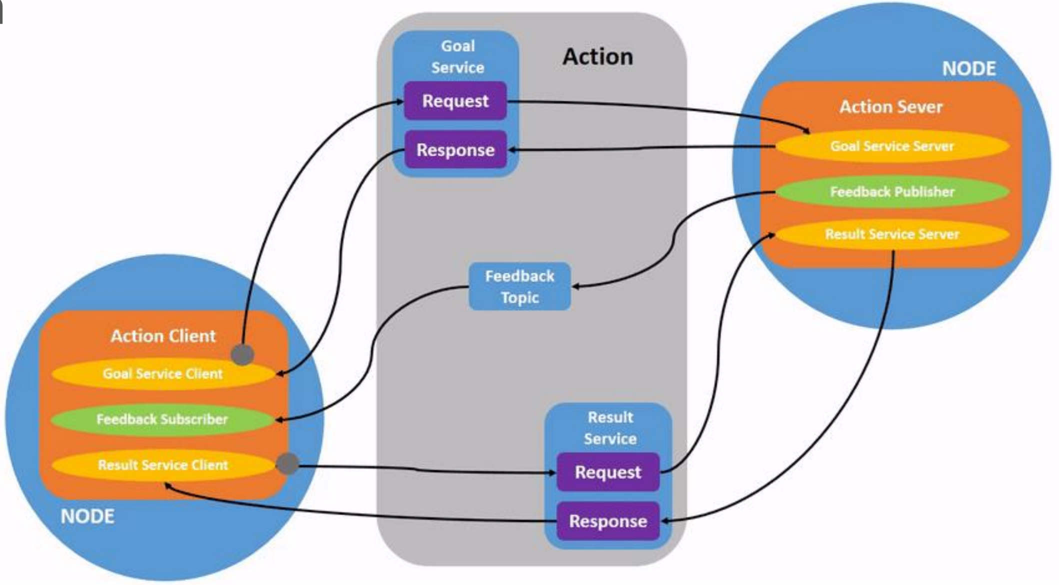
## Services

- An alternative method for communication on the ROS graph
- Call and response
    - Only receive/provide data when specified
    - Not continually updated



# Actions

- Another communication type on the ROS graph
- Useful for long-running tasks, more intricate
- Goal-oriented communication
- Goal, Feedback, Result
  - **Goal** – Desired outcome/task action server should accomplish. Send by client node
  - **Feedback** – Continuous updates on progress of action from server
  - **Result** – Final message sent from server to alert status (success/fail)



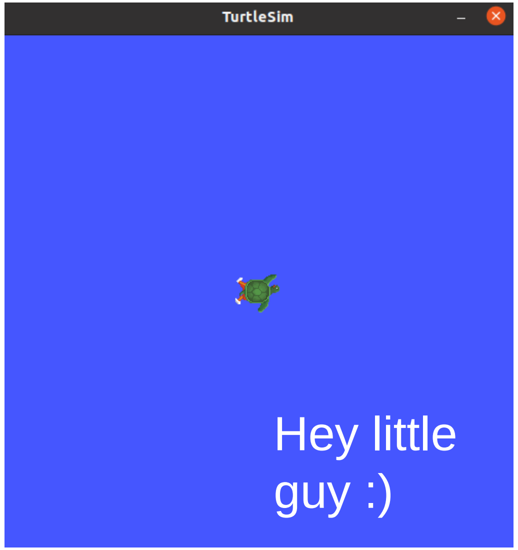
robotics&automation society

## Command Line

Turtlesim Playground Tutorial

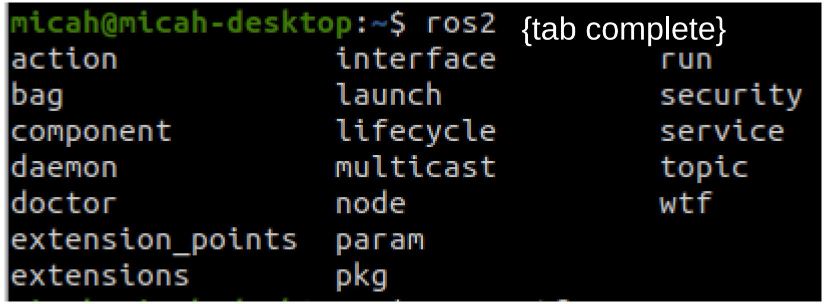
## ROS Command-line interface (CLI)

- We interface with ROS nodes and anything ROS through the CLI
- Every ROS command begins with `ros2` …
- Run a node (start the communication and do the things!)
  - `ros2 run turtlesim turtlesim_node`

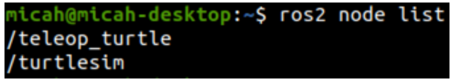


# Exploring ROS CLI – General Information

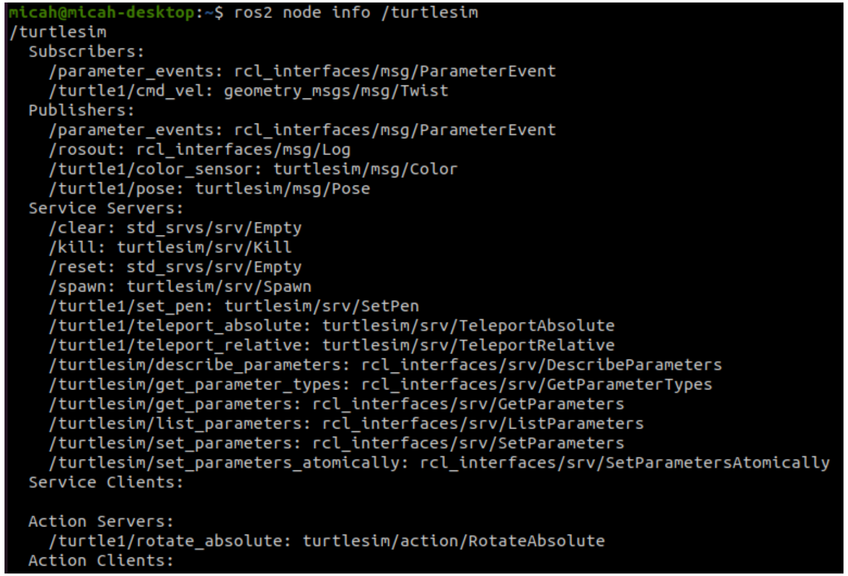
- **Tab complete** is your best friend
  - Shows viable subcommands you can execute or flags/arguments to pass
- You can generally do:
  - `ros2 {node/topic/param/service} list`
  - `ros2 {node/topic/param/service} info`



```
micah@micah-desktop:~$ ros2  {tab complete}
action             interface         run
bag                launch            security
component          lifecycle         service
daemon             multicast         topic
doctor             node              wtf
extension_points   param
extensions         pkg
```



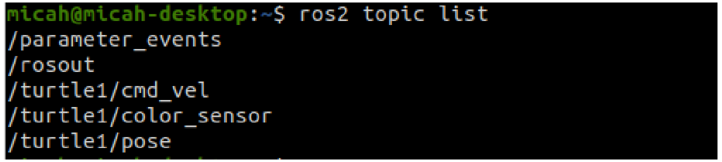
```
micah@micah-desktop:~$ ros2 node list
/teleop_turtle
/turtlesim
```



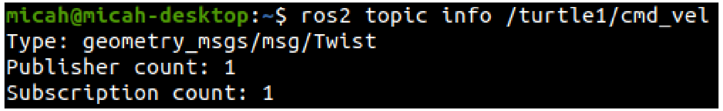
```
micah@micah-desktop:~$ ros2 node info /turtlesim
/turtlesim
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /turtle1/cmd_vel: geometry_msgs/msg/Twist
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /turtle1/color_sensor: turtlesim/msg/Color
    /turtle1/pose: turtlesim/msg/Pose
  Service Servers:
    /clear: std_srvs/srv/Empty
    /kill: turtlesim/srv/Kill
    /reset: std_srvs/srv/Empty
    /spawn: turtlesim/srv/Spawn
    /turtle1/set_pen: turtlesim/srv/SetPen
    /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
    /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
    /turtlesim/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /turtlesim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /turtlesim/get_parameters: rcl_interfaces/srv/GetParameters
    /turtlesim/list_parameters: rcl_interfaces/srv/ListParameters
    /turtlesim/set_parameters: rcl_interfaces/srv/SetParameters
    /turtlesim/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
  Action Clients:
```

## Exploring ROS CLI – Data

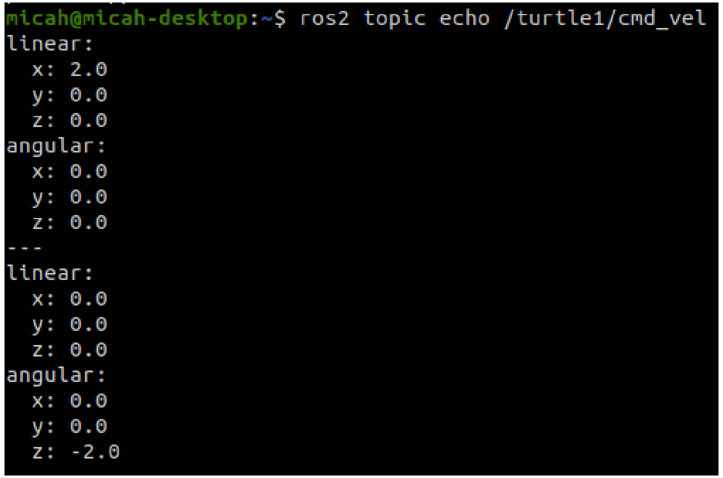
- How is my turtle moving? Let’s find out!
  - *Remember, most commands are sent over topics so that’s a good place to start*



```
micah@micah-desktop:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```



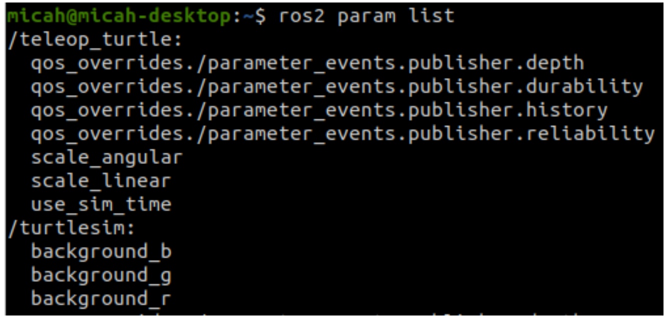
```
micah@micah-desktop:~$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 1
```



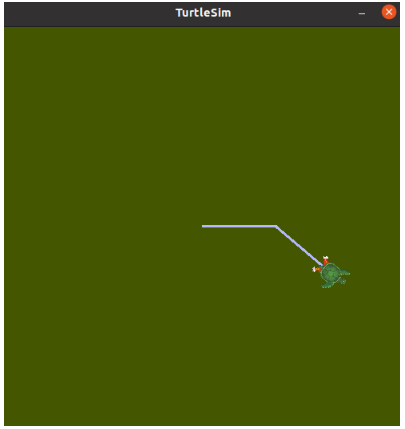
```
micah@micah-desktop:~$ ros2 topic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -2.0
```

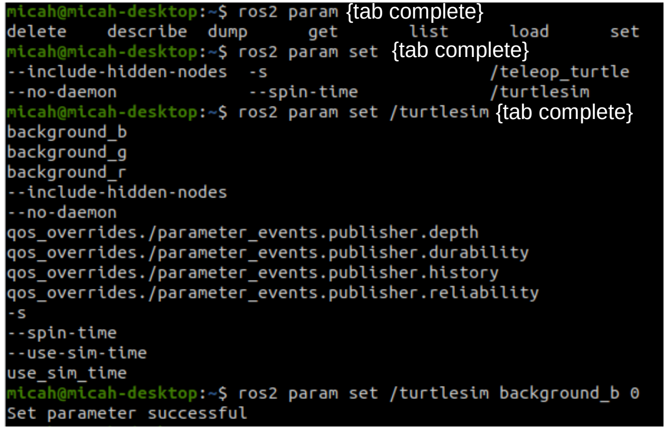
## Exploring ROS CLI – Parameters

- Let's see what we can do with parameters
- Tab completing shows us what we can do and what it changes



```
micah@micah-desktop:~$ ros2 param list
/teleop_turtle:
  qos_overrides./parameter_events.publisher.depth
  qos_overrides./parameter_events.publisher.durability
  qos_overrides./parameter_events.publisher.history
  qos_overrides./parameter_events.publisher.reliability
  scale_angular
  scale_linear
  use_sim_time
/turtlesim:
  background_b
  background_g
  background_r
```





```
micah@micah-desktop:~$ ros2 param {tab complete}
delete     describe  dump       get        list       load       set
micah@micah-desktop:~$ ros2 param set {tab complete}
--include-hidden-nodes   -s                          /teleop_turtle
--no-daemon              --spin-time                 /turtlesim
micah@micah-desktop:~$ ros2 param set /turtlesim {tab complete}
background_b
background_g
background_r
--include-hidden-nodes
--no-daemon
qos_overrides./parameter_events.publisher.depth
qos_overrides./parameter_events.publisher.durability
qos_overrides./parameter_events.publisher.history
qos_overrides./parameter_events.publisher.reliability
-s
--spin-time
--use-sim-time
use_sim_time
micah@micah-desktop:~$ ros2 param set /turtlesim background_b 0
Set parameter successful
```

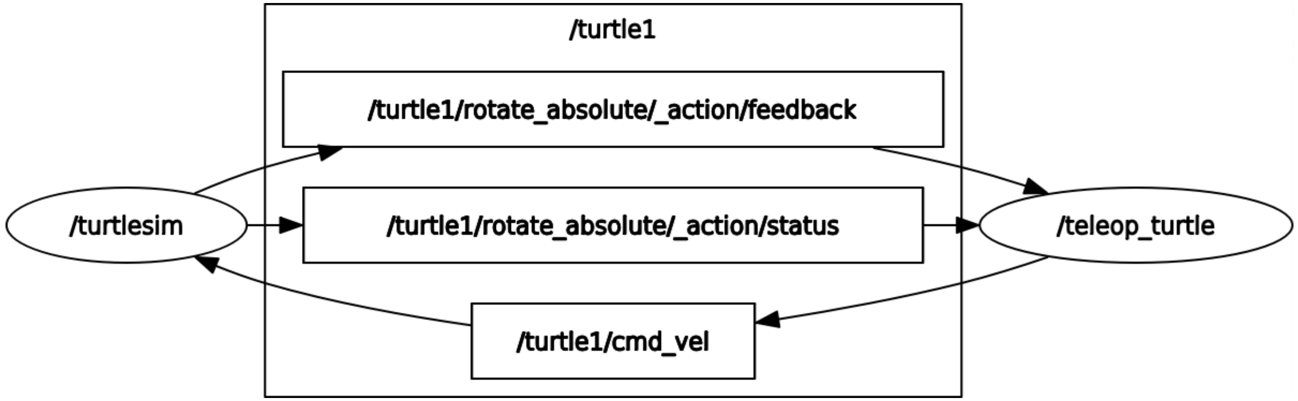
We changed a ROS node setting!

## More Common CLI Commands

- `ros2 launch`
- `ros2 bag`
- `ros2 msg show`
- `ros2 interface`

## rqt_graph

- Visualizing the ROS Graph
- `rqt_graph`

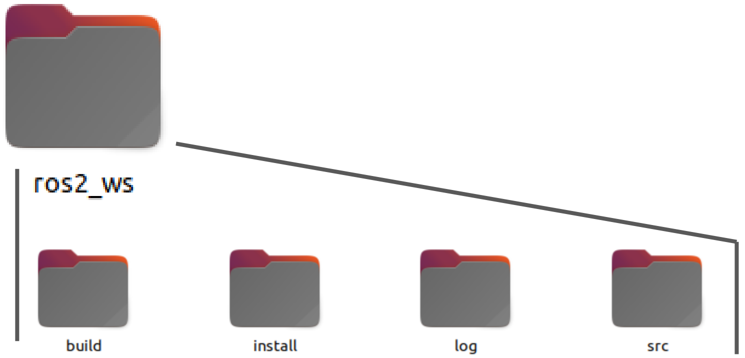

/turtle1
/turtle1/rotate_absolute/_action/feedback
/turtlesim
/turtle1/rotate_absolute/_action/status
/teleop_turtle
/turtle1/cmd_vel

## Developer Environment

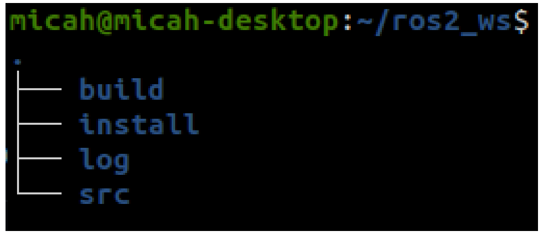
## Workspaces

- A directory with a specific structure
- Inside is where our ROS packages reside
- An environment overlay to be sourced

File Explorer



Terminal



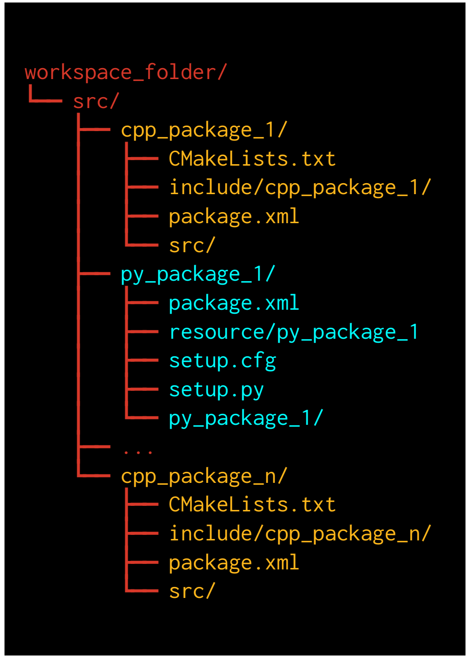
```
micah@micah-desktop:~/ros2_ws$
.
├── build
├── install
├── log
└── src
```

## Packages

- An organizational unit for ROS code and nodes
- C++ Contents:
  - `Package.xml` – Meta information about the package (Version, Maintainer, Dependencies, Licenses, etc.)
  - `CMakeLists.txt` – Recipe for compiler to follow to build your code (Flags, executables, include directories, packages, etc.)
  - `src` – Folder (directory) containing our source code – the code that actually does stuff and has nodes
- Python Contents
  - `Package.xml` – Meta information about the package (Version, Maintainer, Dependencies, Licenses, etc.)
  - `setup.cfg` – File to let ROS find our executable python files
  - `setup.py` – Instructions for how to install package
  - `__init__.py` – Helps ROS find your package

# Packages

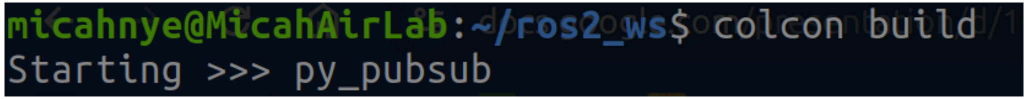
- Like much of ROS, packages are modular!
  - They can be used across workspaces (if installed or relocated and dependencies are met)
  - You can even mix C++ and Python packages in the same workspace!
    - Beauty of ROS backbone communication



```
workspace_folder/
└── src/
    ├── cpp_package_1/
    │   ├── CMakeLists.txt
    │   ├── include/cpp_package_1/
    │   ├── package.xml
    │   └── src/
    ├── py_package_1/
    │   ├── package.xml
    │   ├── resource/py_package_1
    │   ├── setup.cfg
    │   ├── setup.py
    │   └── py_package_1/
    ├── ...
    └── cpp_package_n/
        ├── CMakeLists.txt
        ├── include/cpp_package_n/
        ├── package.xml
        └── src/
```

## Building your package

- Colcon is our build tool
- This creates
  - /build directory – Intermediate files are auto-generated and stored (CMake)
  - /install directory – Package gets installed into here
  - /log directory – Logging information from build gets stored here
- You can set build flags to fit the build process to your needs
- You **need** to rebuild your package to update the installation and build with your new changes



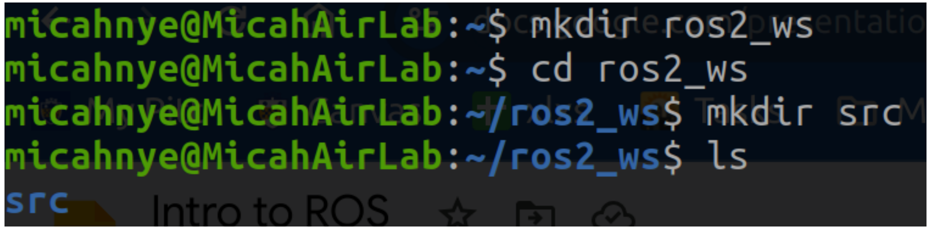
```
micahnye@MicahAirLab:~/ros2_ws$ colcon build
Starting >>> py_pubsub
```

# Creating ROS Nodes

Talker and Listener Tutorial in Python

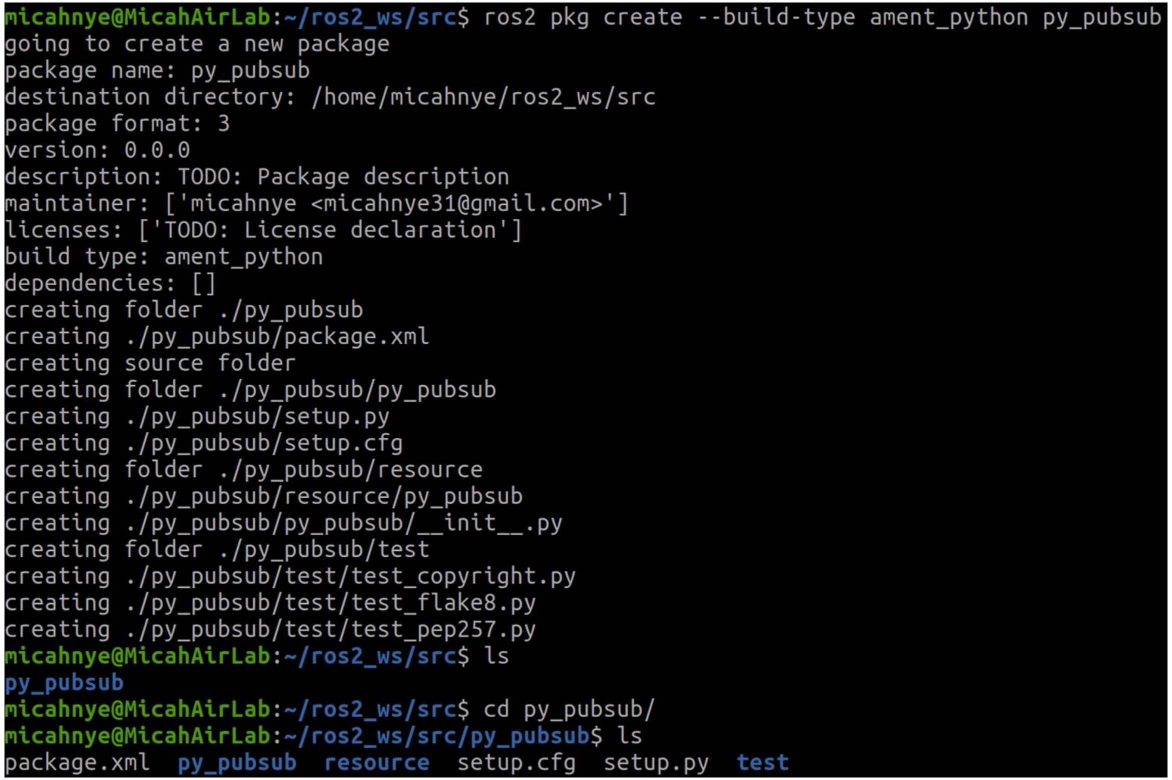
No ROS on Laptop? No worries! Follow along on https://rosonweb.io/

## Creating your workspace



```
micahnye@MicahAirLab:~$ mkdir ros2_ws
micahnye@MicahAirLab:~$ cd ros2_ws
micahnye@MicahAirLab:~/ros2_ws$ mkdir src
micahnye@MicahAirLab:~/ros2_ws$ ls
src
```

## Creating your package



```
micahnye@MicahAirLab:~/ros2_ws/src$ ros2 pkg create --build-type ament_python py_pubsub
going to create a new package
package name: py_pubsub
destination directory: /home/micahnye/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['micahnye <micahnye31@gmail.com>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
creating folder ./py_pubsub
creating ./py_pubsub/package.xml
creating source folder
creating folder ./py_pubsub/py_pubsub
creating ./py_pubsub/setup.py
creating ./py_pubsub/setup.cfg
creating folder ./py_pubsub/resource
creating ./py_pubsub/resource/py_pubsub
creating ./py_pubsub/py_pubsub/__init__.py
creating folder ./py_pubsub/test
creating ./py_pubsub/test/test_copyright.py
creating ./py_pubsub/test/test_flake8.py
creating ./py_pubsub/test/test_pep257.py
micahnye@MicahAirLab:~/ros2_ws/src$ ls
py_pubsub
micahnye@MicahAirLab:~/ros2_ws/src$ cd py_pubsub/
micahnye@MicahAirLab:~/ros2_ws/src/py_pubsub$ ls
package.xml  py_pubsub  resource  setup.cfg  setup.py  test
```

# Creating your Publisher

```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

...
```

```python
...

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()
```

https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html#write-the-publisher-node

# Creating your Subscriber

robotics&automation society

```
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()
```

https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html#write-the-subscriber-node

# Update Package.xml and setup.py

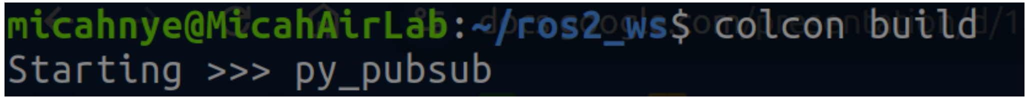
```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
 <name>py_pubsub</name>
 <version>0.0.0</version>
 <description>TODO: Package description</description>
 <maintainer email="micahnye31@gmail.com">micahnye</maintainer>
 <license>TODO: License declaration</license>

 <test_depend>ament_copyright</test_depend>
 <test_depend>ament_flake8</test_depend>
 <test_depend>ament_pep257</test_depend>
 <test_depend>python3-pytest</test_depend>

 <exec_depend>rclpy</exec_depend>
 <exec_depend>std_msgs</exec_depend>

 <export>
   <build_type>ament_python</build_type>
 </export>
</package>
```

```python
from setuptools import setup

package_name = 'py_pubsub'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='micahnye',
    maintainer_email='micahnye31@gmail.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'talker = py_pubsub.publisher_member_function:main',
            'listener = py_pubsub.subscriber_member_function:main',
        ],
    },
)
```

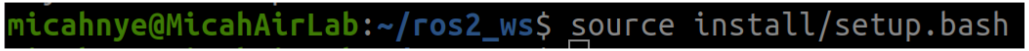
## Building the package

- Inside workspace, `colcon build`
- Don't forget to source the overlay of your workspace!

  `source install/setup.bash`

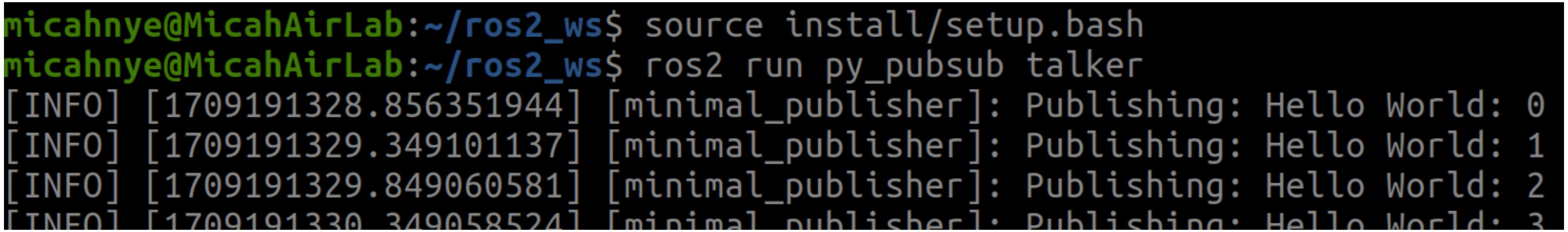


```
micahnye@MicahAirLab:~/ros2_ws$ colcon build
Starting >>> py_pubsub
```

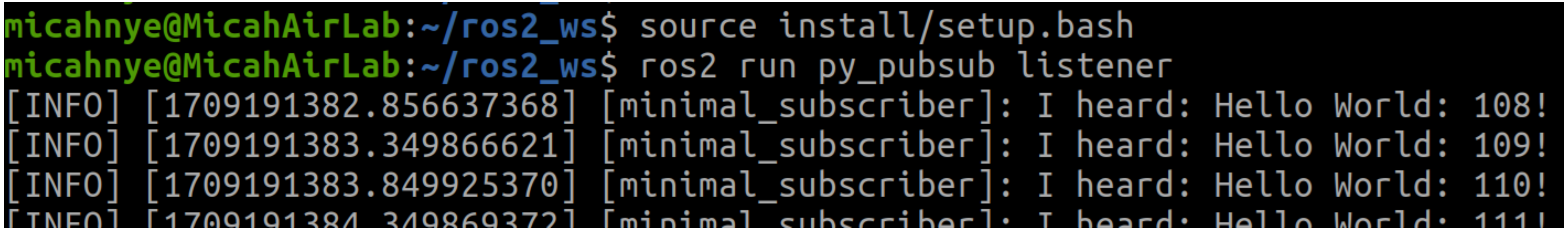


```
micahnye@MicahAirLab:~/ros2_ws$ source install/setup.bash
```

## Testing our nodes



```
micahnye@MicahAirLab:~/ros2_ws$ source install/setup.bash
micahnye@MicahAirLab:~/ros2_ws$ ros2 run py_pubsub talker
[INFO] [1709191328.856351944] [minimal_publisher]: Publishing: Hello World: 0
[INFO] [1709191329.349101137] [minimal_publisher]: Publishing: Hello World: 1
[INFO] [1709191329.849060581] [minimal_publisher]: Publishing: Hello World: 2
[INFO] [1709191330.349058524] [minimal_publisher]: Publishing: Hello World: 3
```

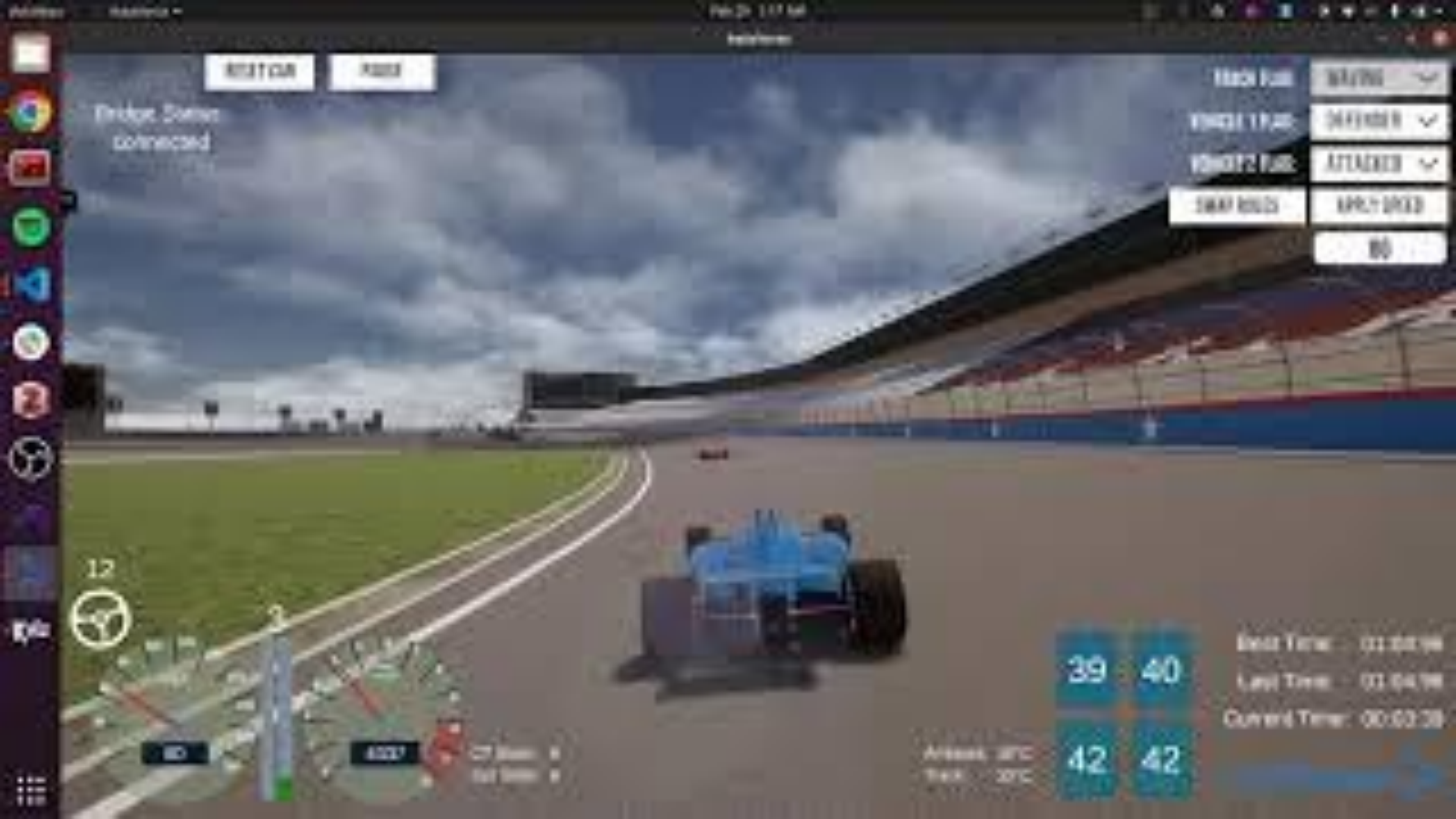


```
micahnye@MicahAirLab:~/ros2_ws$ source install/setup.bash
micahnye@MicahAirLab:~/ros2_ws$ ros2 run py_pubsub listener
[INFO] [1709191382.856637368] [minimal_subscriber]: I heard: Hello World: 108!
[INFO] [1709191383.349866621] [minimal_subscriber]: I heard: Hello World: 109!
[INFO] [1709191383.849925370] [minimal_subscriber]: I heard: Hello World: 110!
[INFO] [1709191384.349869372] [minimal_subscriber]: I heard: Hello World: 111!
```

robotics&automation society

# Autonomous Racing with ROS!

Complex System Demo



Track Flag: WAVING
Vehicle 1 Flag: DEFENDER
Vehicle 2 Flag: ATTACKER

Best Time: 01:[illegible]
Last Time: 01:[illegible]
Current Time: 00:00:38

Ambient: [illegible]°C
Track: [illegible]°C

## Final Takeaways

- ROS is a powerful tool for projects, research, and industry*, but it is not the only option
- ROS is a modular framework that builds complex systems using Nodes on the ROS graph
- Topics are a primary form of communication between Nodes
- ROS CLI is used for running code and useful for debugging and visualization into what’s going on behind the scenes
- Never be afraid to poke around and explore ROS!

## What’s Next?

- Bagging
- Rviz & Visualization
- Plotjuggler
- Custom Messages
- Quality of Service for Subscribers and Publishers
- ROS_DOMAIN_IDs
- Specified Executors
- Containers & Components
- Parameter Configurations
- Callback groups
- RMW and DDS
- …

robotics&automation society

# Thank you :)

Micah Nye