Intro to ROS2

(*Robot Operating System*)

Ros2
What ,When,Why

What is ROS2, When to use it, and Why ?

ROS2
Robot Operating System

- Basics
- Core functionalities

➔ Program a new robot in no time !

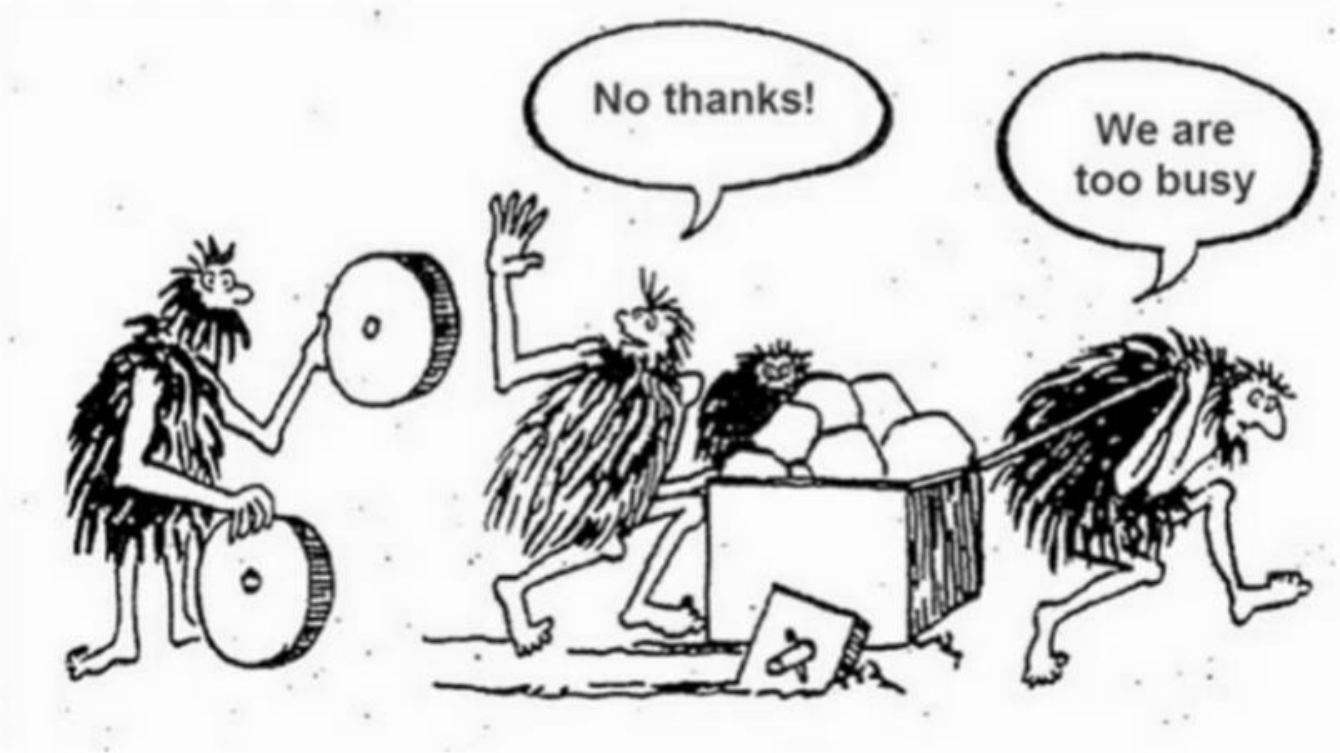
What is ROS2, When to use it, and Why ?

ROS goals :

➜ Provide a **standard** for robotic applications

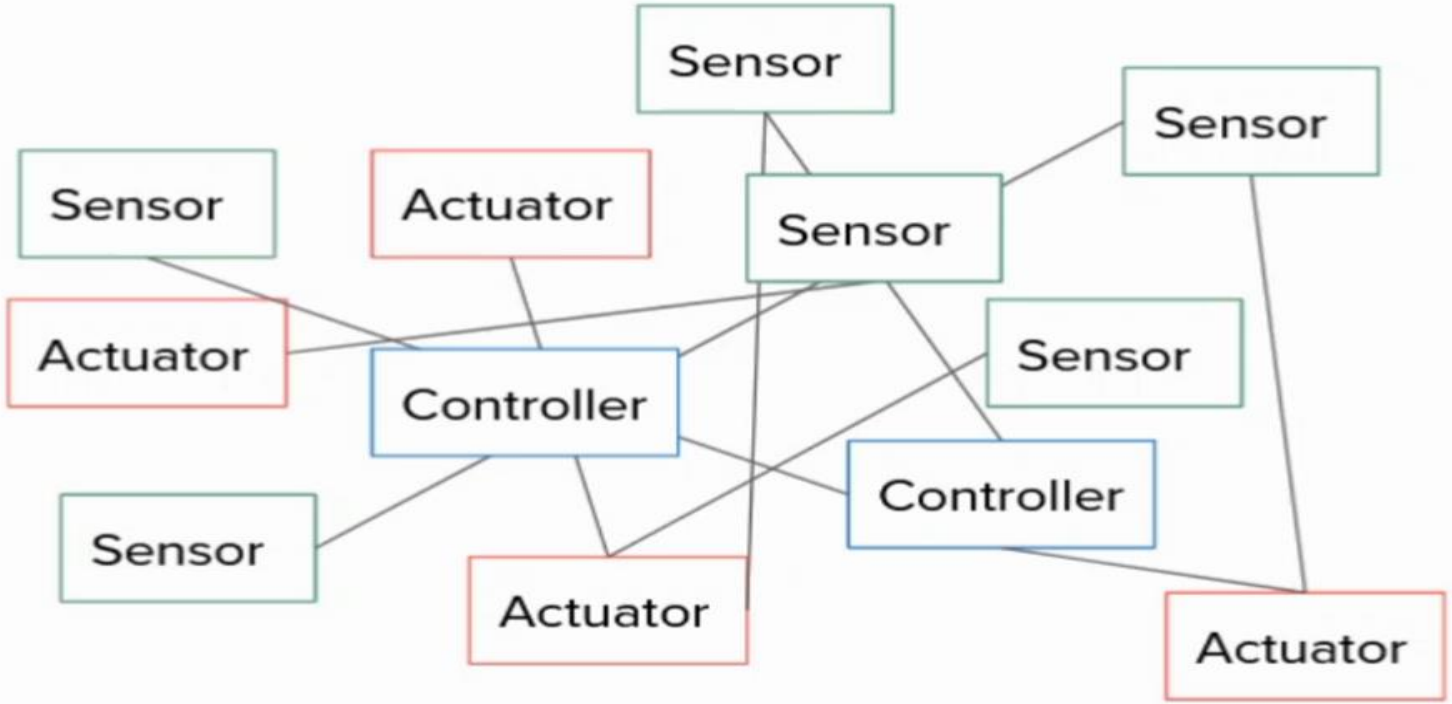
➜ Use on **any robot**

➜ **Don't reinvent the wheel !**



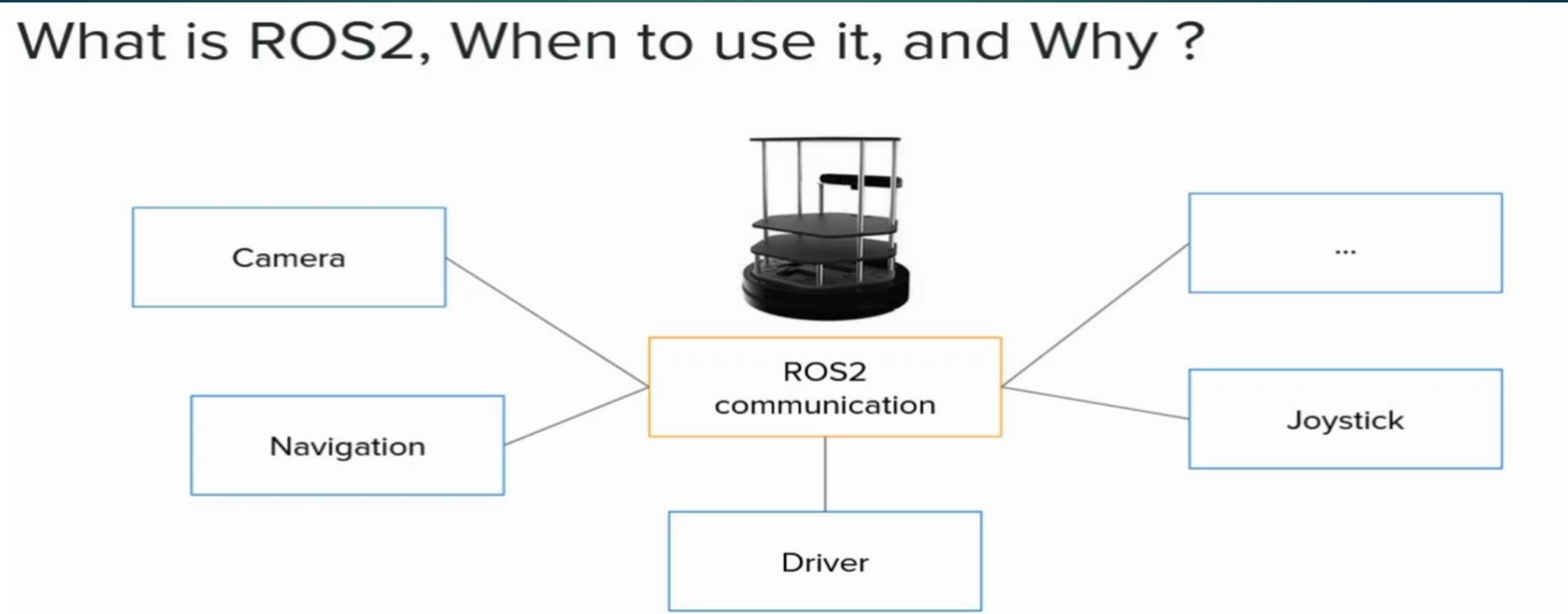
What is ROS2, When to use it, and Why ?

When to use ROS2 ?



What is ROS2

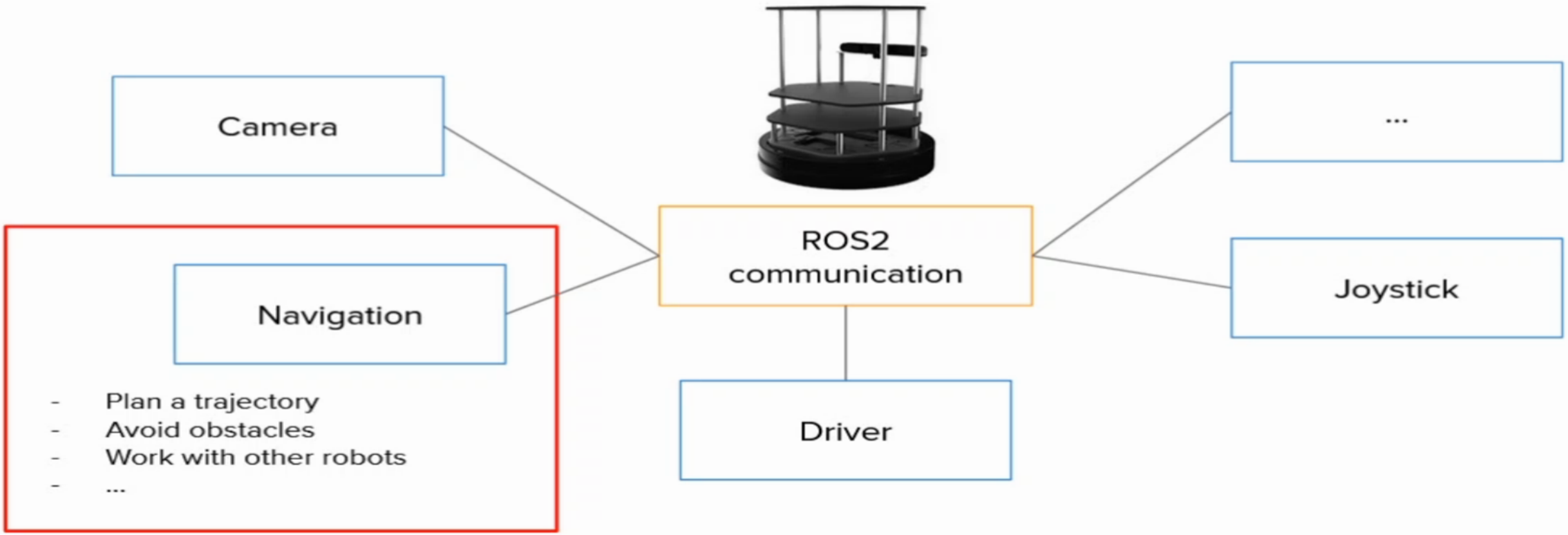
- code separation and communication tool



What is ROS2

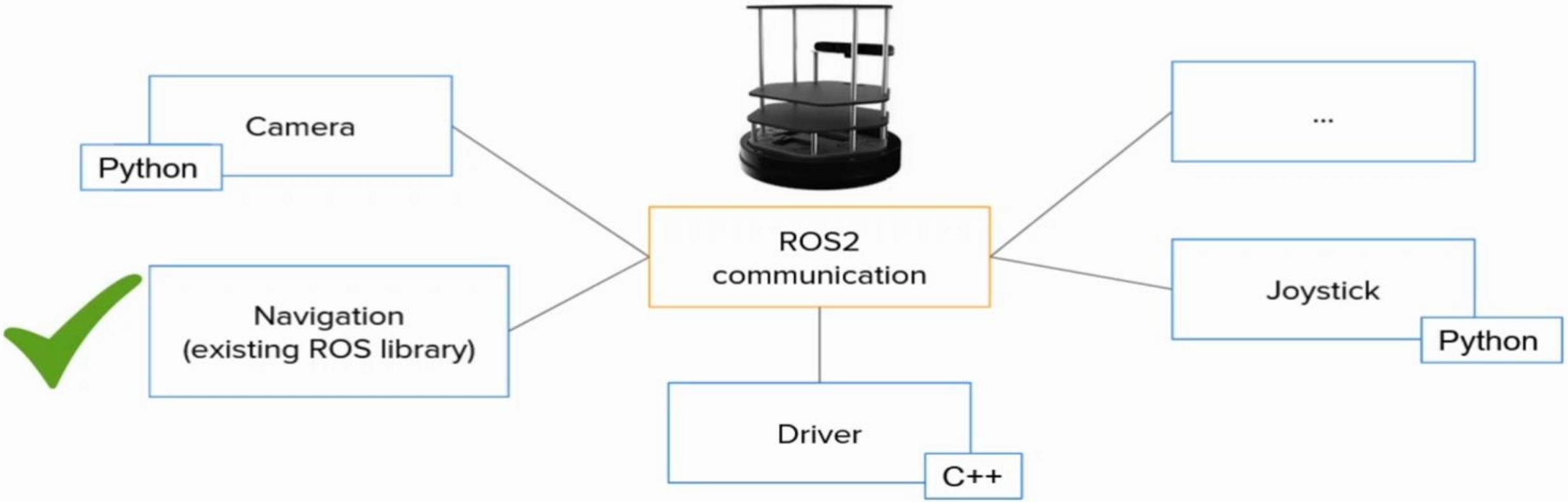
Ros2 is tool and plug&play libraries.

What is ROS2, When to use it, and Why ?



ROS2 is language agnostic

What is ROS2, When to use it, and Why ?



Ros2 Index

https://index.ros.org/ (For Self Study)

ROS Resources: Documentation | Support | Discussion Forum | Service Status | Q&A answers.ros.org

ROS Index BETA

ABOUT INDEX DOC CONTRIBUTE STATS Search ROS

PACKAGE LIST REPOSITORY LIST

Search ROS

Welcome to ROS Index

ROS Index is the entry point for searching ROS and ROS 2 resources, including packages, repositories, system dependencies and documentation.

You can enter keywords and phrases in the search bar and then filter results by resource type, or you can browse the complete package, repository and system dependency lists under the **Index** tab.

Under the **Doc** tab, you'll find the official ROS 2 documentation, including installation instructions, tutorials, distribution features and summaries, contributing guides, and more.

To go directly to the installation pages, click here.
To go directly to the tutorials, click here.

Active Distributions

ROS Melodic

ROS Noetic

ROS 2 Dashing

ROS 2 Foxy

Development Distribution

ROS 2 Rolling

More resources

ROS Discourse

ROS Answers

ROS2 Distribution

https://docs.ros.org/en/foxy/Releases.html

ROS 2 Documentation: Foxy

Search docs

- Installation
- Tutorials
- Guides
- Concepts
- Contributing
- Contact
- ROSCon Content
- Distributions
  - ROS 2 alpha releases (Aug 2015 - Oct 2016)
  - Beta 1 (codename 'Asphalt'; December 2016)
  - Beta 2 (codename 'r2b2'; July 2017)
  - Beta 3 (codename 'r2b3'; September 2017)
  - ROS 2 Ardent Apalone (codename 'ardent'; December 2017)
  - ROS 2 Bouncy Bolson (codename 'bouncy'; June 2018)
  - ROS 2 Crystal Clemmys (codename 'crystal'; December 2018)
  - ROS 2 Dashing Diademata (codename 'dashing'; May 31st, 2019)
  - ROS 2 Eloquent Elusor (codename

Other Versions v: foxy

» Distributions

Edit on GitHub

Distributions

What is a Distribution?

A ROS distribution is a versioned set of ROS packages. These are akin to Linux distributions (e.g. Ubuntu). The purpose of the ROS distributions is to let developers work against a relatively stable codebase until they are ready to roll everything forward. Therefore once a distribution is released, we try to limit changes to bug fixes and non-breaking improvements for the core packages (every thing under ros-desktop-full). That generally applies to the whole community, but for "higher" level packages, the rules are less strict, and so it falls to the maintainers of a given package to avoid breaking changes.

List of Distributions

Below is a list of current and historic ROS 2 distributions. Rows in the table marked in green are the currently supported distributions.

Distro	Release date	Logo	EOL Date
Foxy Fitzroy	June 5, 2020		May 2023
Eloquent Elusor	Nov 22nd, 2019		November 2020

Supported Platforms For ROS2

https://docs.ros.org/en/foxy/Releases/Release-Foxy-Fitzroy.html#installation

Foxy Fitzroy is the sixth release of ROS 2.

Supported Platforms

Foxy Fitzroy is primarily supported on the following platforms:

Tier 1 platforms:

- Ubuntu 20.04 (Focal): `amd64` and `arm64`
- Mac macOS 10.14 (Mojave)
- Windows 10 (Visual Studio 2019)

Tier 3 platforms:

- Ubuntu 20.04 (Focal): `arm32`
- Debian Buster (10): `amd64`, `arm64` and `arm32`
- OpenEmbedded Thud (2.6) / webOS OSE: `arm32` and `x86`

For more information about RMW implementations, compiler / interpreter versions, and system dependency versions see REP 2000.

Installation

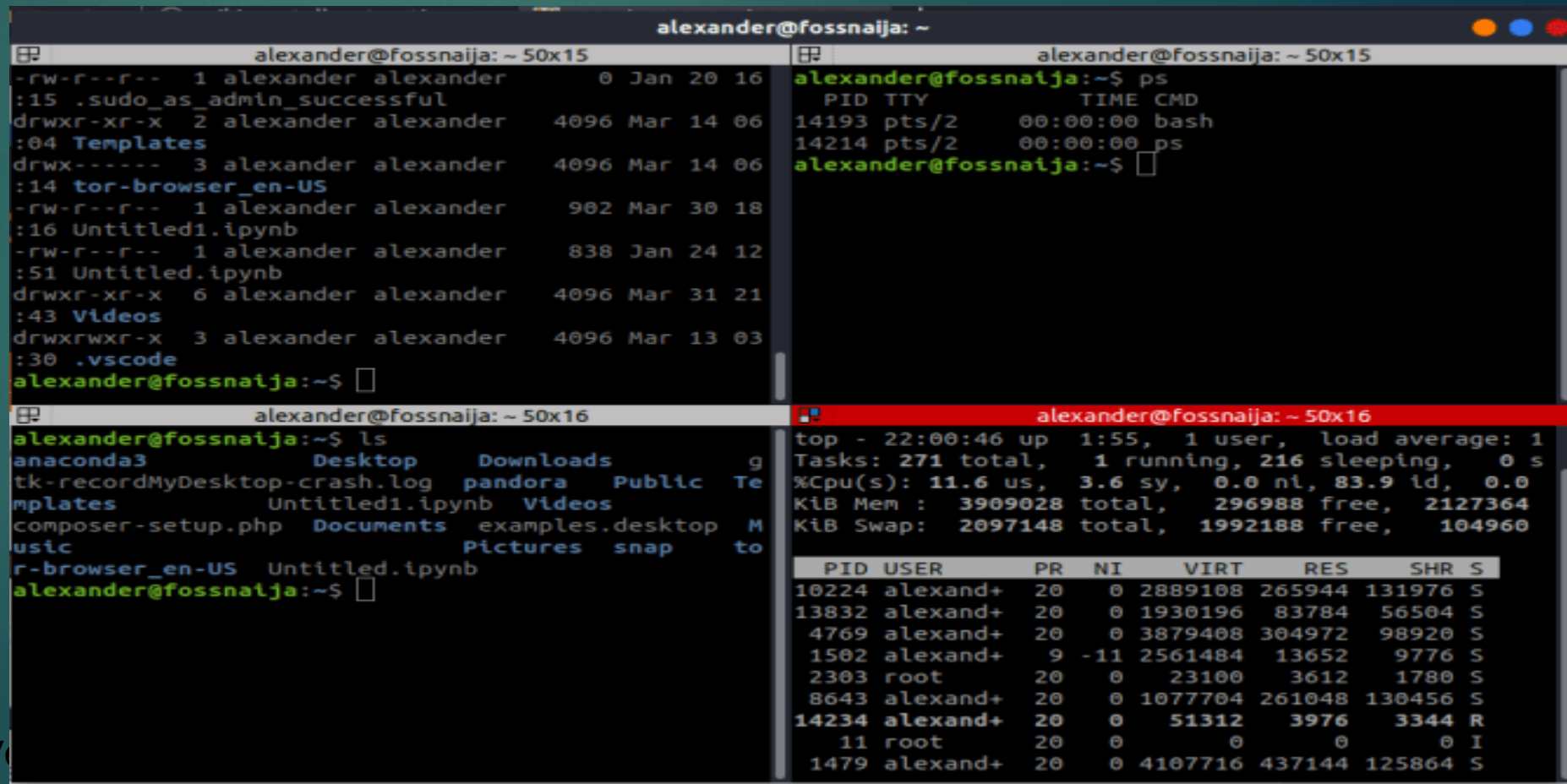
Install Foxy Fitzroy

Terminator

- installation commands:-
- sudo add-apt-repository ppa:gnome-terminator

- sudo apt-get update

- sudo apt-get install terminator



tekomoro

Terminator Shortcuts

https://cheatography.com/svschannak/cheat-sheets/terminator-ubuntu/pdf_bw/

Cheatography

Terminator Ubuntu Keyboard Shortcuts

by svschannak via cheatography.com/25158/cs/6681/

Splitting	
Ctrl+Shift+O	Split terminals Horizontally.
Ctrl+Shift+E	Split terminals Vertically.
Ctrl+Shift+W	Close the current terminal.

Moving	
Alt+Right	Move to the right terminal.
Alt+Left	Move to the left terminal.
Alt+Up	Move to the upper terminal.
Alt+Down	Move to the bottom terminal.

General Control	
Ctrl+Shift+X	Toggle between showing all terminals and only showing the current one.
Ctrl+Shift+Right	Move parent dragbar Right.
Ctrl+Shift+Left	Move parent dragbar Left.
Ctrl+Shift+Up	Move parent dragbar Up.
Ctrl+Shift+Down	Move parent dragbar Down.

Tabs	
Ctrl+Shift+T	Open new tab
Ctrl+PageDown	Move to next Tab
Ctrl+PageUp	Move to previous Tab

Font Size	
Ctrl+Plus (+)	Increase font size.
Ctrl+Minus (-)	Decrease font size.
Ctrl+Zero (0)	Restore font size to original setting.

Features Status

- supported platforms (Ubuntu 18.04, macOS 10.12.x, Windows 10)
- DDS(Data Distribution Services) implementations (eProsima Fast RTPS, RTI Connext and ADLINK Opensplice)
- Quality of Services
- Real time
- ROS 1 - ROS 2 communication bridge
- Compatibility

As for now ROS is not very popular in the industry, and lacks some of the most important requirements, such as real-time, safety, certification, security. One of the goals for ROS2 is to make it compatible with industrial applications.

- ROS Noetic's EOL (End of Life) is scheduled for 2025. After that, no more ROS1!

ROS1 vs ROS2: writing your nodes

- **The ROS API – rclcpp, rclpy**

- In ROS1, for Cpp you use roscpp, and for Python, rospy. Both libraries are completely independent and built from scratch. It means that the API is not necessarily the same between roscpp and rospy, and some features are developed for one, and not the other.
- You won't use the rcl library directly in your programs. You'll use another client library built on top of rcl. For example: rclcpp for Cpp, rclpy for Python

Installing ROS 2

https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html

- Set Locals

```
sudo apt update && sudo apt install locales
 sudo locale-gen en_US en_US.UTF-8
 sudo update-locale LC_ALL=en_US.UTF-8  LANG=en_US.UTF-8
export  LANG=en_US.UTF-8
```

- Setup Sources

```
sudo apt update && sudo apt install curl gnupg2 lsb-release
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc  |  sudo apt-key add –
sudo sh -c 'echo "deb [arch=$(dpkg --print-architecture)] http://packages.ros.org/ros2/ubuntu
$(lsb_release -cs) main" > /etc/apt/sources.list.d/ros2-latest.list'
```

- Install ROS2 packages

```
sudo apt update
sudo apt install ros-foxy-desktop
sudo apt install -y python3-pip
pip3 install -U argcomplete
```

Source the setup files

- You will need to run this command on every new shell you open to have access to the ROS 2 commands
- Command to source the setup files : `source /opt/ros/<distro>/setup.bash`
- If you don't want to have to source the setup file every time you open a new shell, you can add the command to your shell startup script: `echo "source /opt/ros/<distro>/setup.bash" >> ~/.bashrc`

ROS2 packages

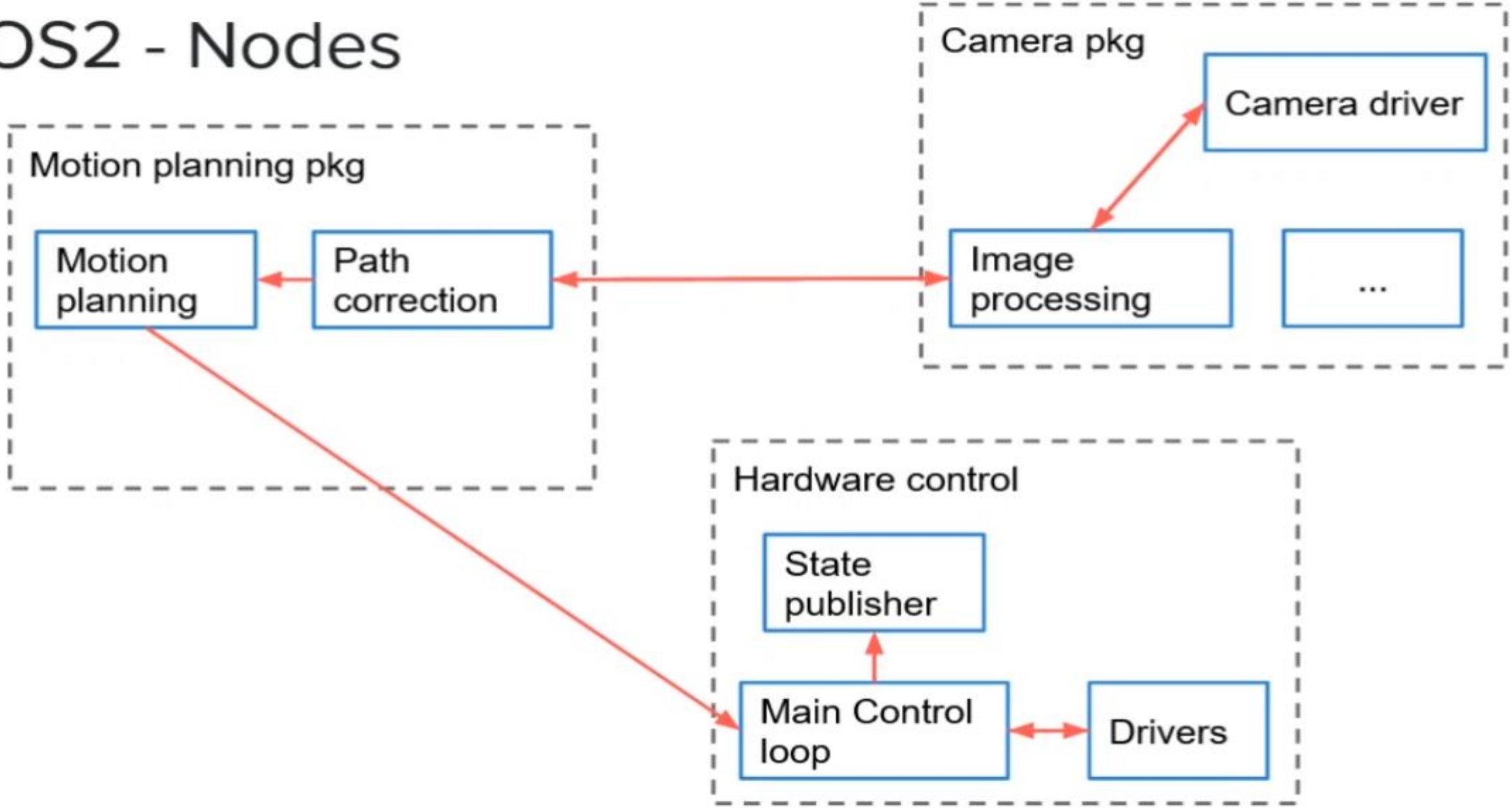
ROS2 - Packages

Motion planning pkg

Camera pkg

Hardware control

ROS2 - Nodes


Camera pkg
Camera driver
Image processing
...
Motion planning pkg
Motion planning
Path correction
Hardware control
State publisher
Main Control loop
Drivers

ROS2 - Nodes

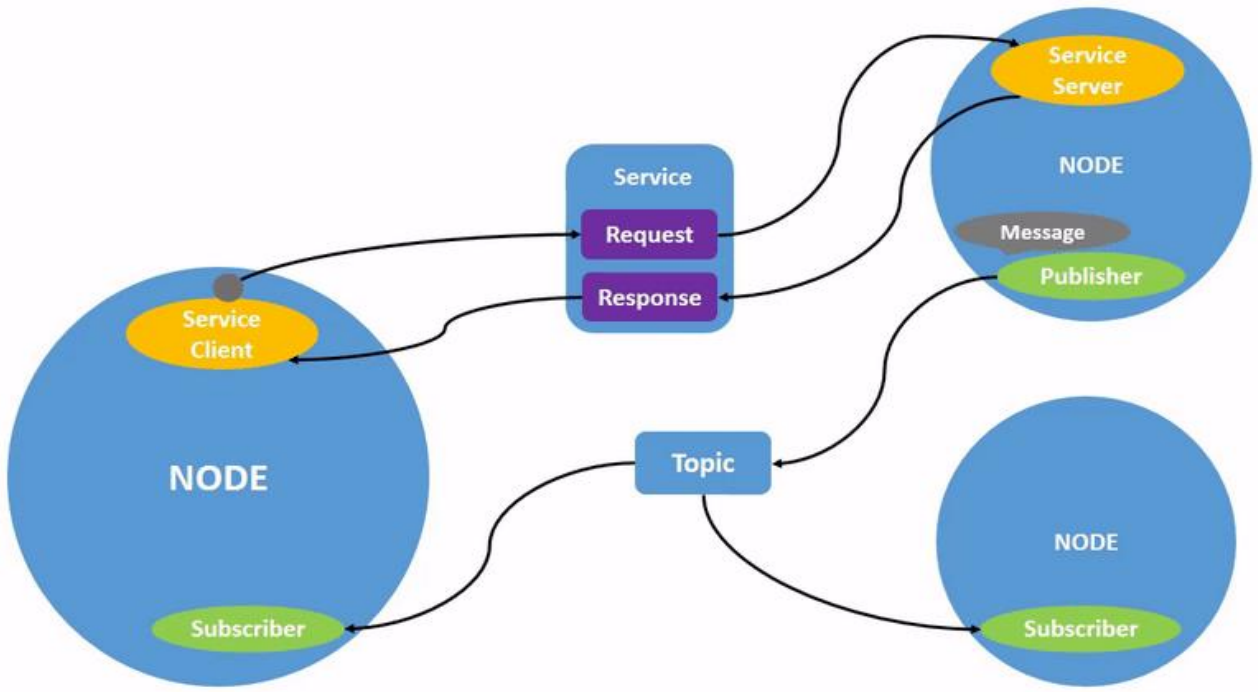
- Subprograms in your application, responsible for only one thing
- Combined into a graph
- Communicate with each other through topics, services, and parameters

Benefits :

- Reduce code complexity
- Fault tolerance
- Can be written in Python, C++, ...

Understanding ROS 2 nodes

- Each node in ROS should be responsible for a single, module purpose
- Each node can send and receive data to other nodes via topics, services, actions, or parameters



Create workspace

first make a directory with workspace you need using the following command :-

```
mkdir  -p <directory name>/src
```

then build your Workspace using colcon build tool

```
colcon build
```

you will find the following files in your WS_directory:-

build - src - log - install

build Tool

- colcon is an iteration on the ROS build tools catkin_make, catkin_make_isolated, catkin_tools and ament_tools.
- to install colcon :-

```
sudo apt install python3-colcon-common-extensions
```

- colcon autocomplete

  add the following command to bashrc

```
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
```

Create workspace

ahmedhp@ahmedhp-HP-EliteBook-8560p: ~/tekomoro_ws

ahmedhp@ahmedhp-HP-EliteBook-8560p: ~/tekomoro_ws 80x24

```
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws$ ls
build  install  log  src
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws$
```

ROS2_WS

https://docs.ros.org/en/foxy/Tutorials/Workspace/Creating-A-Workspace.html

```
. install/local_setup.bash
```

Note

Sourcing the `local_setup` of the overlay will only add the packages available in the overlay to your environment. `setup` sources the overlay as well as the underlay it was created in, allowing you to utilize both workspaces.

So, sourcing your main ROS 2 installation's `setup` and then the `dev_ws` overlay's `local_setup`, like you just did, is the same as just sourcing `dev_ws`'s `setup`, because that includes the environment of the underlay it was created in.

setup.bash vs local_setup.bash

Overlay : your WS
Underlay : ROS 2 environment

```
ahmedhp@ahmedhp-HP-EliteBook-8560p: ~/tekomoro_ws/install
ahmedhp@ahmedhp-HP-EliteBook-8560p: ~/tekomoro_ws/install 80x24
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws/install$ ls
COLCON_IGNORE      local_setup.sh              local_setup.zsh      setup.ps1
local_setup.bash   _local_setup_util_ps1.py    robot_localization   setup.sh
local_setup.ps1    _local_setup_util_sh.py     setup.bash           setup.zsh
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws/install$
```

ROS2 - Packages

Camera pkg

Motion planning pkg

Hardware control

Create python Package

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

ahmedhp@ahmedhp-HP-EliteBook-8560p: ~/tekomoro_ws/src

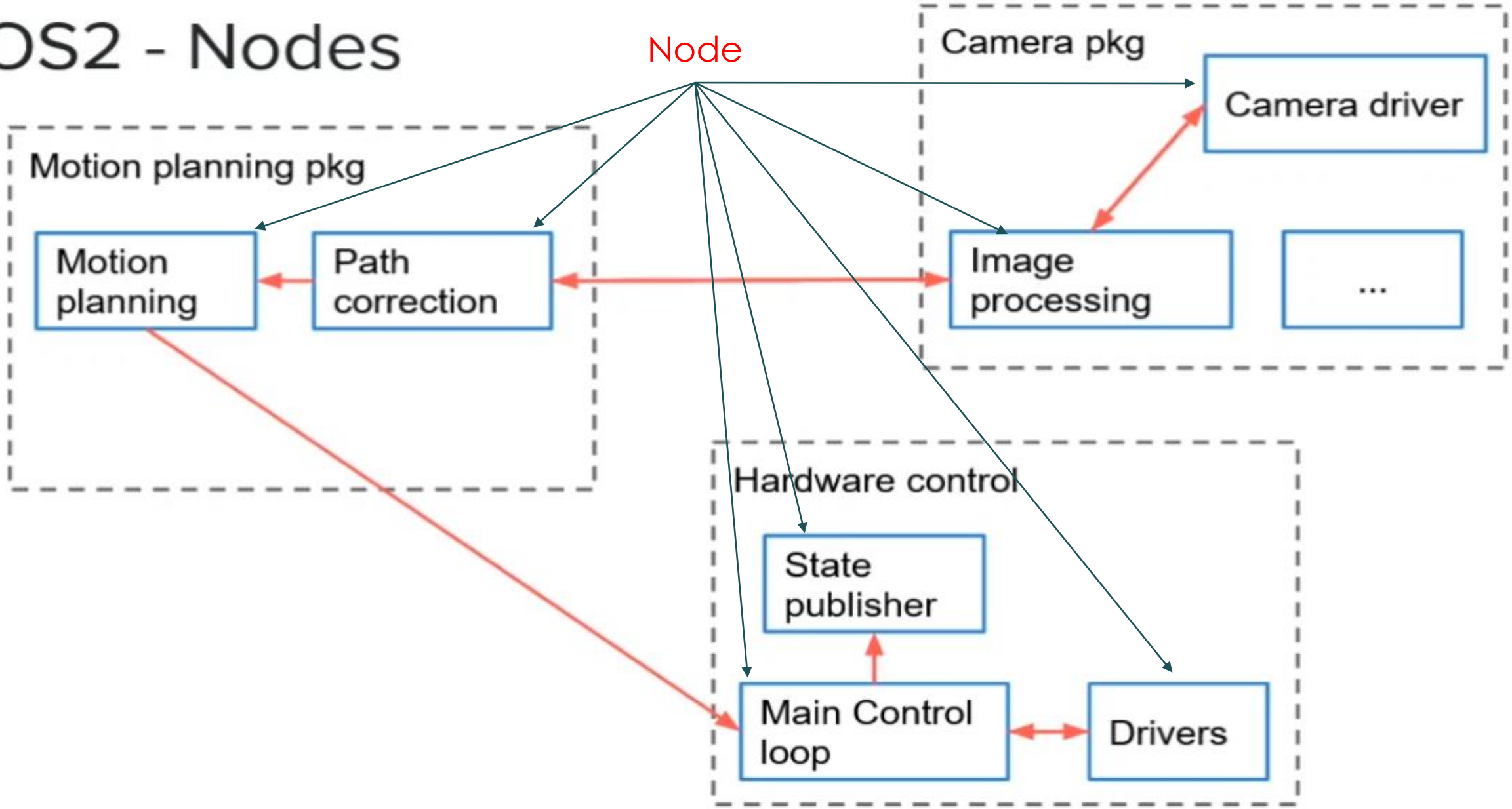
ahmedhp@ahmedhp-HP-EliteBook-8560p: ~/tekomoro_ws/src 118x28

```
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws/src$ ros2 pkg create py_pkg --build-type ament_python
going to create a new package
package name: py_pkg
destination directory: /home/ahmedhp/tekomoro_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['ahmedhp <ahmedhp@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
creating folder ./py_pkg
creating ./py_pkg/package.xml
creating source folder
creating folder ./py_pkg/py_pkg
creating ./py_pkg/setup.py
creating ./py_pkg/setup.cfg
creating folder ./py_pkg/resource
creating ./py_pkg/resource/py_pkg
creating ./py_pkg/py_pkg/__init__.py
creating folder ./py_pkg/test
creating ./py_pkg/test/test_copyright.py
creating ./py_pkg/test/test_flake8.py
creating ./py_pkg/test/test_pep257.py
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws/src$ ls
custom_navigation  py_pkg  robot_localization  rplidar_ros  test_pkg  zed-ros2-wrapper
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws/src$
```

ROS2 python Package

```
ahmedhp@ahmedhp-HP-EliteBook-8560p: ~/tekomoro_ws/src/py_pkg/py_pkg
ahmedhp@ahmedhp-HP-EliteBook-8560p: ~/tekomoro_ws/src/py_pkg/py_pkg 118x28
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws/src/py_pkg$ ls
package.xml  py_pkg  resource  setup.cfg  setup.py  test
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws/src/py_pkg$ cd py_pkg/
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws/src/py_pkg/py_pkg$ ls
__init__.py
ahmedhp@ahmedhp-HP-EliteBook-8560p:~/tekomoro_ws/src/py_pkg/py_pkg$
```

ROS2 - Nodes


Node
Camera pkg
Camera driver
Motion planning pkg
Motion planning
Path correction
Image processing
...
Hardware control
State publisher
Main Control loop
Drivers

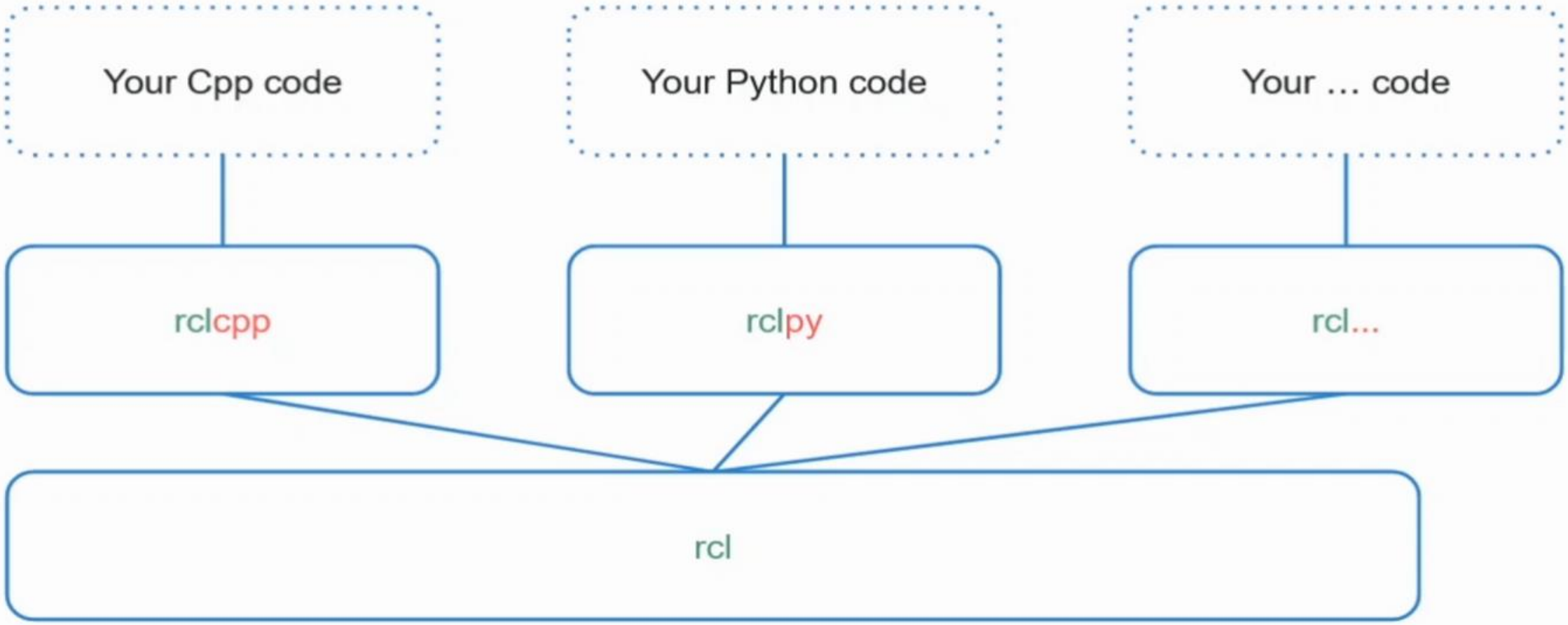
ROS2 - Nodes

- Subprograms in your application, responsible for only one thing
- Combined into a graph
- Communicate with each other through topics, services, and parameters

Benefits :

- Reduce code complexity
- Fault tolerance
- Can be written in Python, C++, ...

ROS2 - Language Libraries



Write your first Node

Install Node

Setup.py

"simple_node=pkg.exec:main"

OOP Pub Node

Create Your Own Node

tekomoro

Make Your Node with your Name

ACTIVITY

Ros2 Node Tools (debug)

ros2 run -h

ros2 node (list,info)

ros2 node -h

Don't make 2 node with same name

source .bashrc

Remap : ros2 run pkg exec --ros-args --remap __node:=new_name

Colcon (pkg-select,symlink)

How to use Colcon

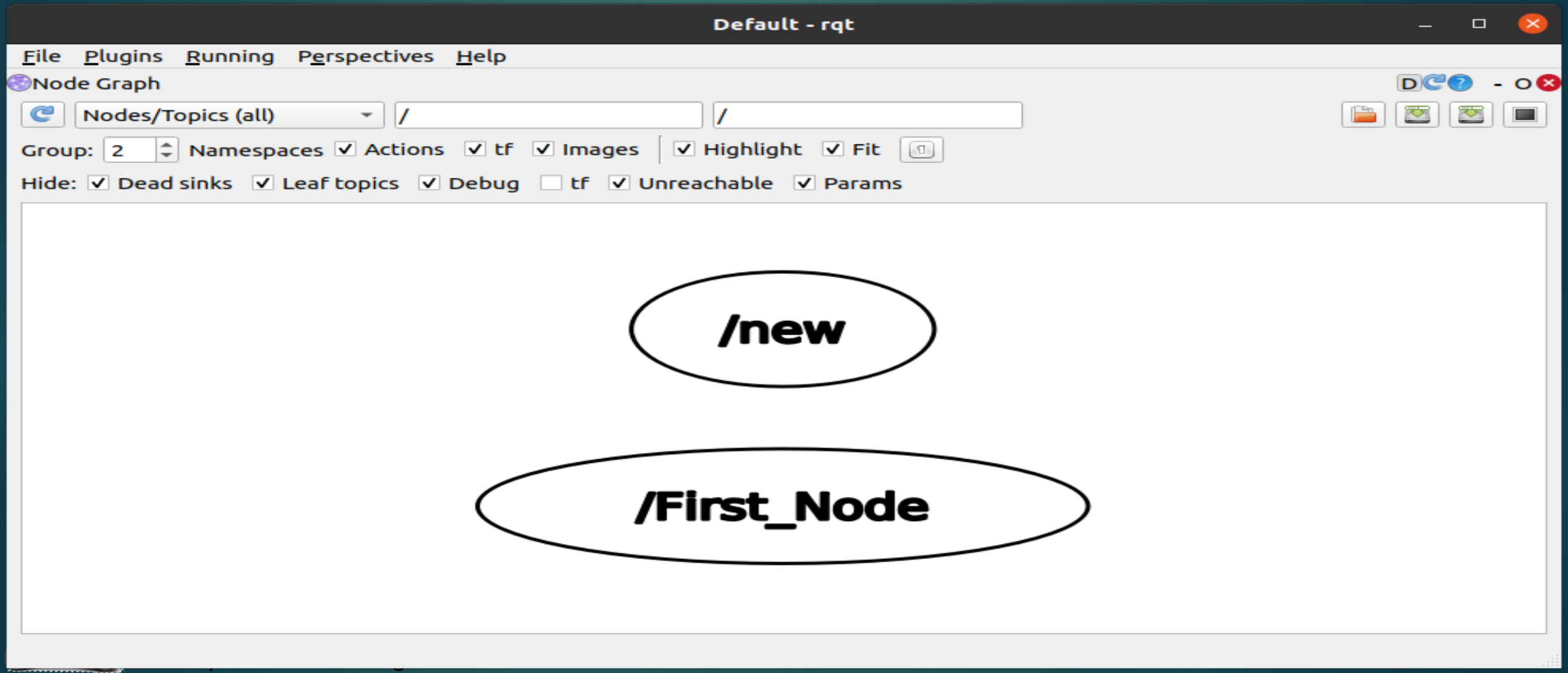
to build Workspace use the Following Command

```
colcon build
```

you can ma symlink to your pkg use the following command

```
colcon build --packages-select <pkg_name> --symlink-install
```

rqt_graph


Default - rqt
File
Plugins
Running
Perspectives
Help
Node Graph
Nodes/Topics (all)
/
/
Group:
2
Namespaces
Actions
tf
Images
Highlight
Fit
Hide:
Dead sinks
Leaf topics
Debug
tf
Unreachable
Params
/new
/First_Node

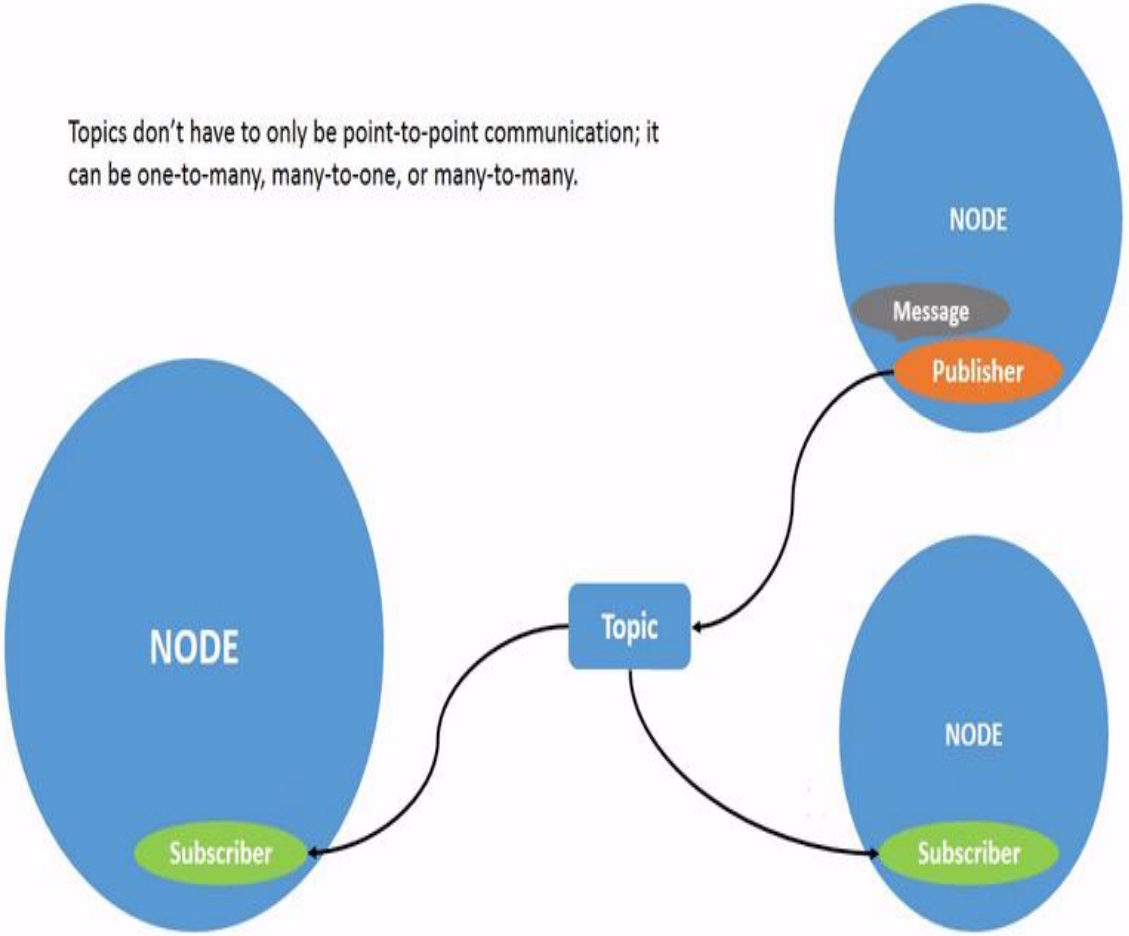
tekomoro

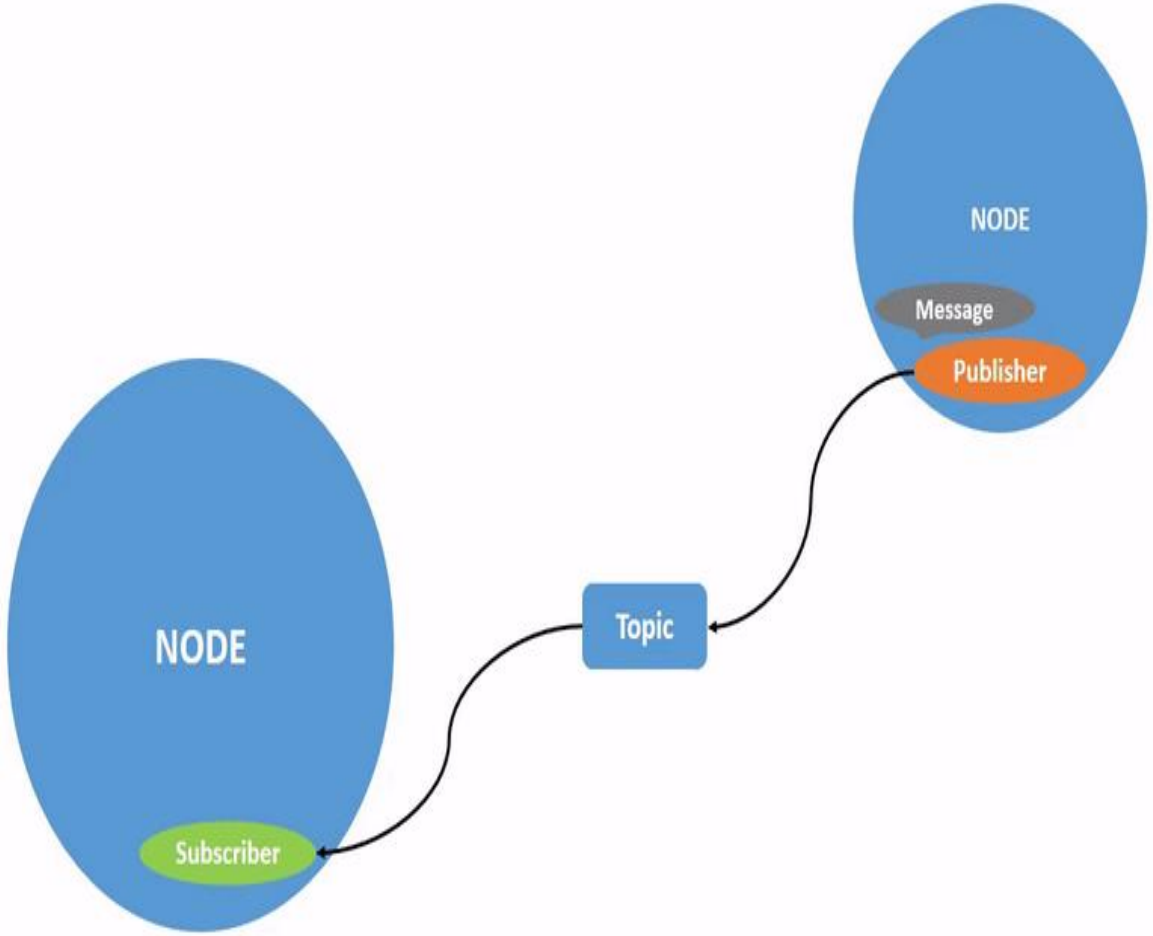
Make 3 different Node with different name from same

ACTIVITY

Topic

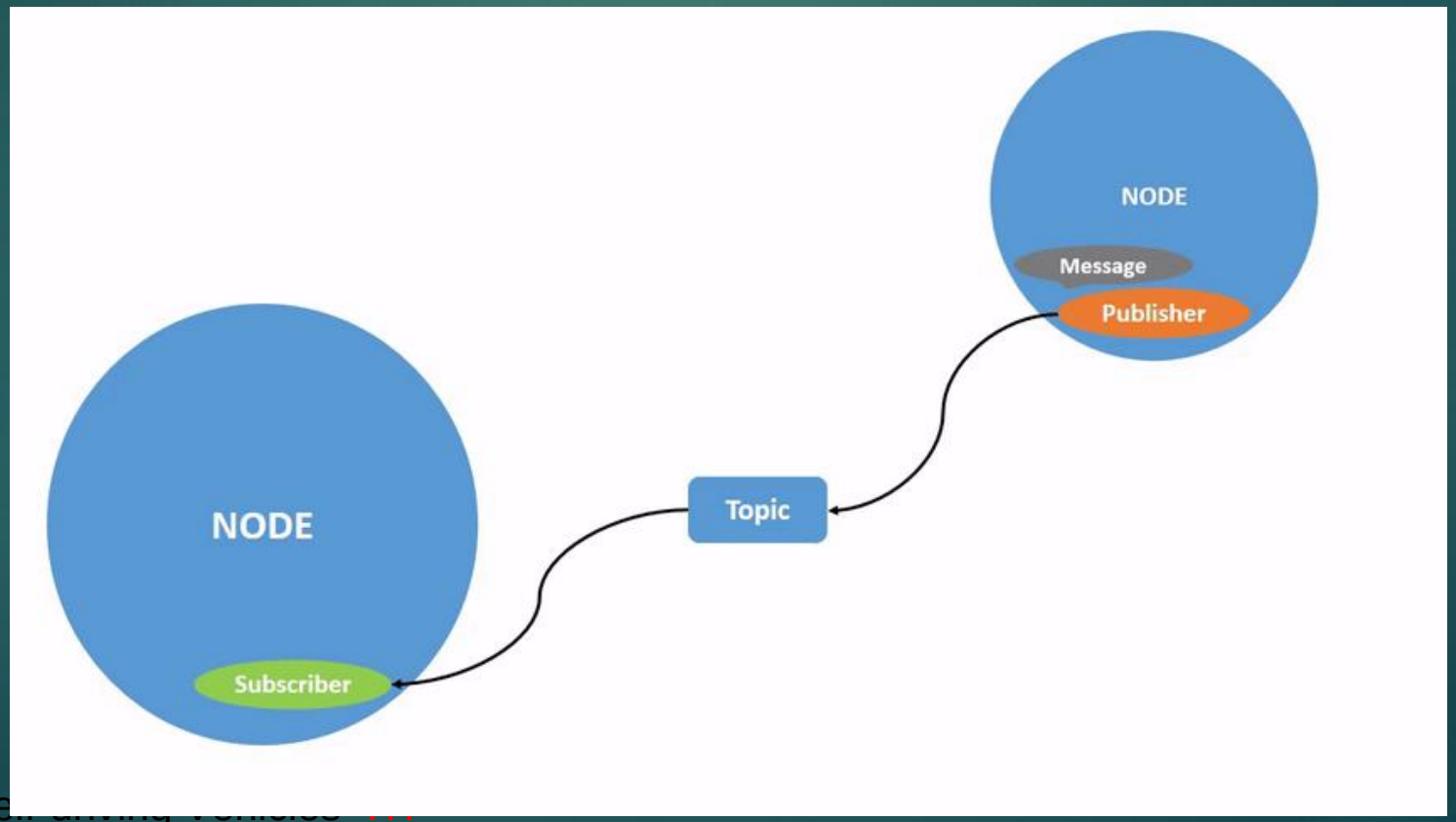
Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.



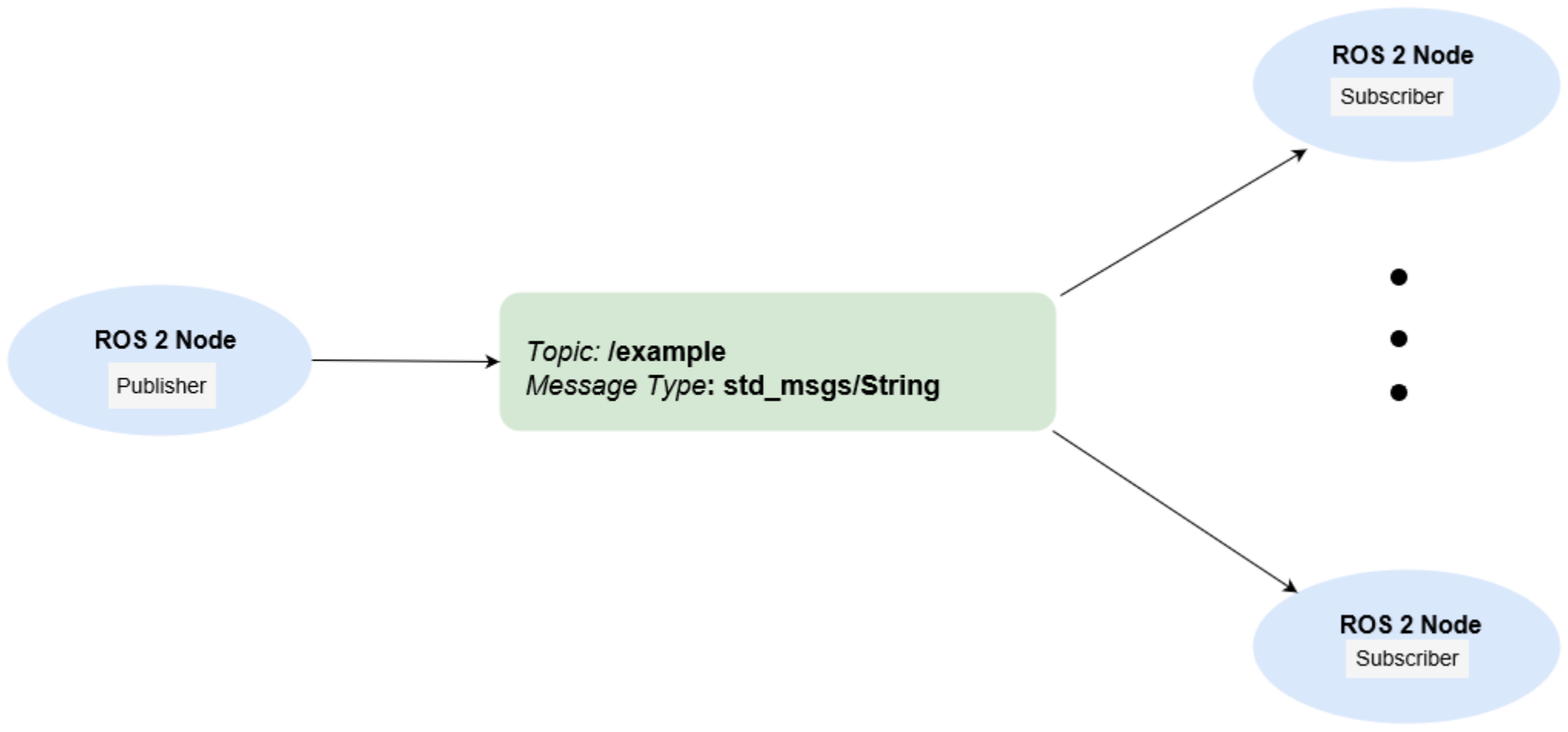


Understanding ROS 2 topics

- Topics are a vital element of the ROS graph that act as a bus for nodes to exchange messages.



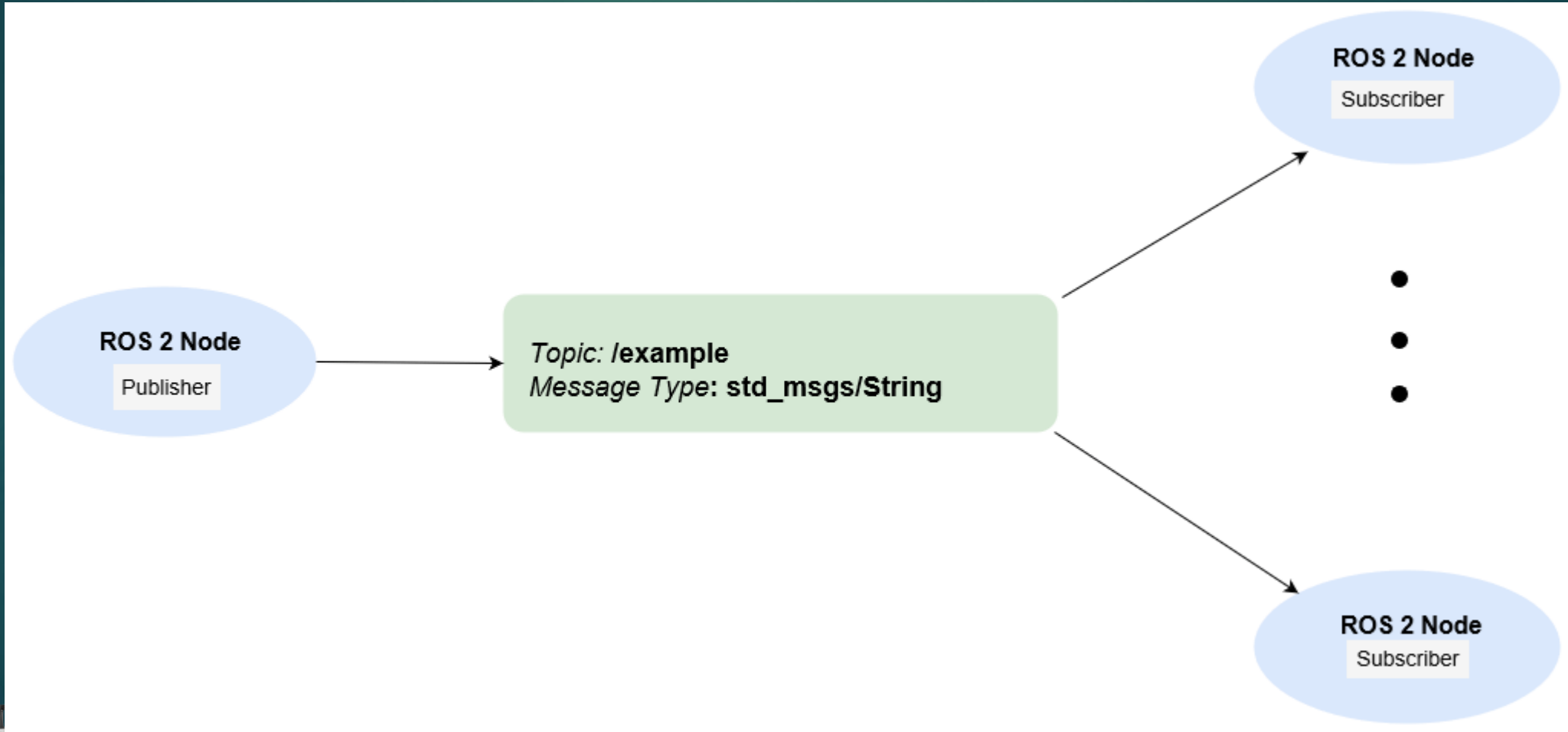
Publisher


ROS 2 Node
Publisher
Topic: /example
Message Type: std_msgs/String
ROS 2 Node
Subscriber
ROS 2 Node
Subscriber

Topic Tools

```
ros2 topic list
ros2 topic echo   (subscriber in terminal)
ros2 topic info
ros2 topic hz
ros2 topic pub
ros2 topic pub /topic example_interfaces/msg/String "{data: "msg"}"
ros2 topic pub -r 5 /my_topic example_interfaces/msg/String "{data: "msg"}"
```

Subscriber


ROS 2 Node
Publisher
Topic: /example
Message Type: std_msgs/String
ROS 2 Node
Subscriber
ROS 2 Node
Subscriber

Remap Topic

```
ros2 run my_1 pub --ros-args --remap __node:=new_one --remap my_topic:=new_topic
```

tekomoro

Make 3 different subscriber node from same topic with different name from same exec

ACTIVITY

How to Run Ros2 Node

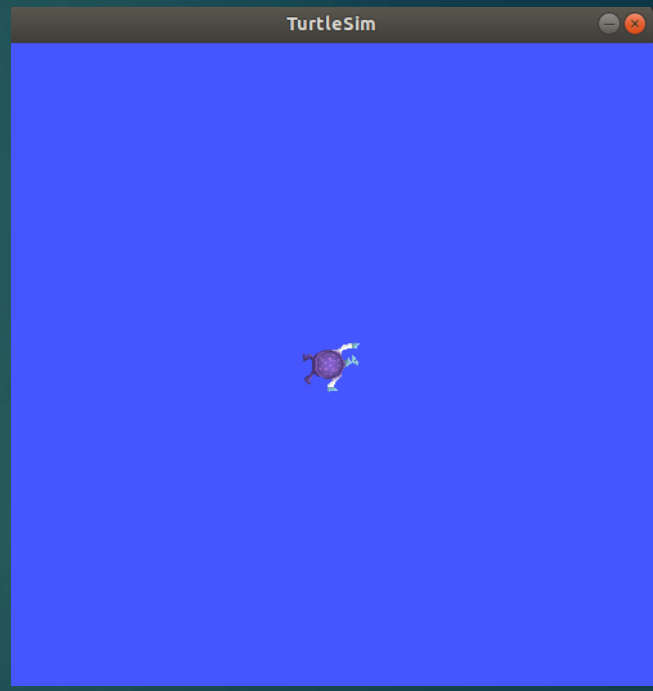
- Ros2 Run

```
ros2 run <package_name> <executable_name>
```

- To run turtlesim, open a new terminal, and enter the following command:

```
ros2 run turtlesim turtlesim_node
```

- The turtlesim window will open



- Here, the package name is turtlesim and the executable name is turtlesim_node

How to know running Node names

- ros2 node list

```
ros2 node list
```

- The terminal will return the node name: /turtlesim
- Open another new terminal and start the teleop node with the command

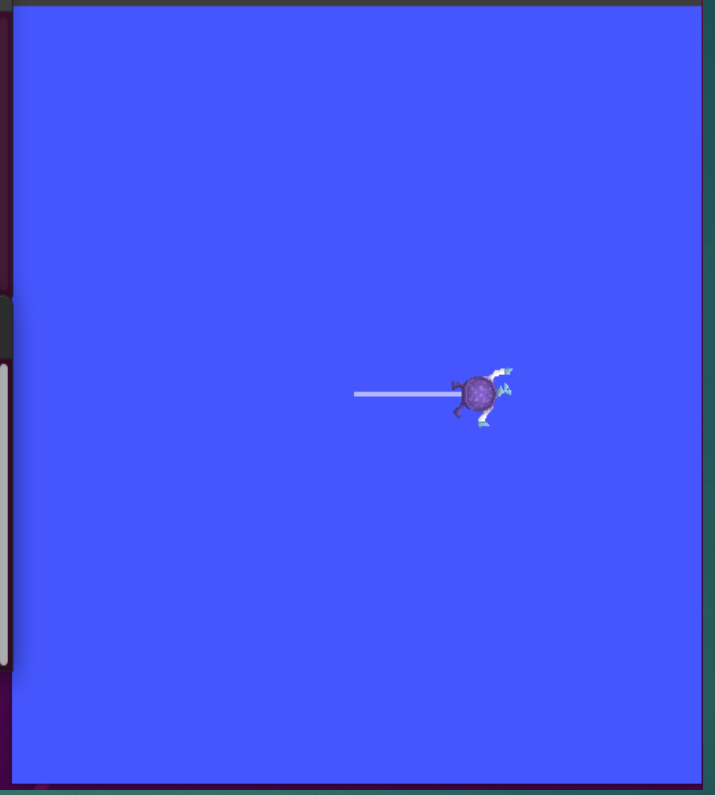
```
ros2 run turtlesim turtle_teleop_key
```

- Here, we are searching the turtlesim package again, this time for the executable named turtle_teleop_key.
- Return to the terminal where you ran ros2 node list and run it again. You will now see the names of two active nodes:/turtlesim, /teleop_turtle

After running previous command

```
ahmed@ahmed-VirtualBox:~$ ros2 run  turtlesim turtlesim_node
[INFO] [1606933202.324425739] [turtlesim]: Starting turtlesim with node name /tu
rtlesim
[INFO] [1606933202.348482946] [turtlesim]: Spawning turtle [turtle1] at x=[5.544
445], y=[5.544445], theta=[0.000000]
```

```
ahmed@ahmed-VirtualBox:~$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rot
ation.
'Q' to quit.
```



- Type ros2 node list ,terminal will return the following:

```
ahmed@ahmed-VirtualBox:~$ ros2 node list
/teleop_turtle
/turtlesim
ahmed@ahmed-VirtualBox:~$
```

Ros2 node info

- you can access more information about the nodes using ros2 node info command :

```
ros2 node info <node_name>
```

- To examine your latest node, turtlesim, run the following command:

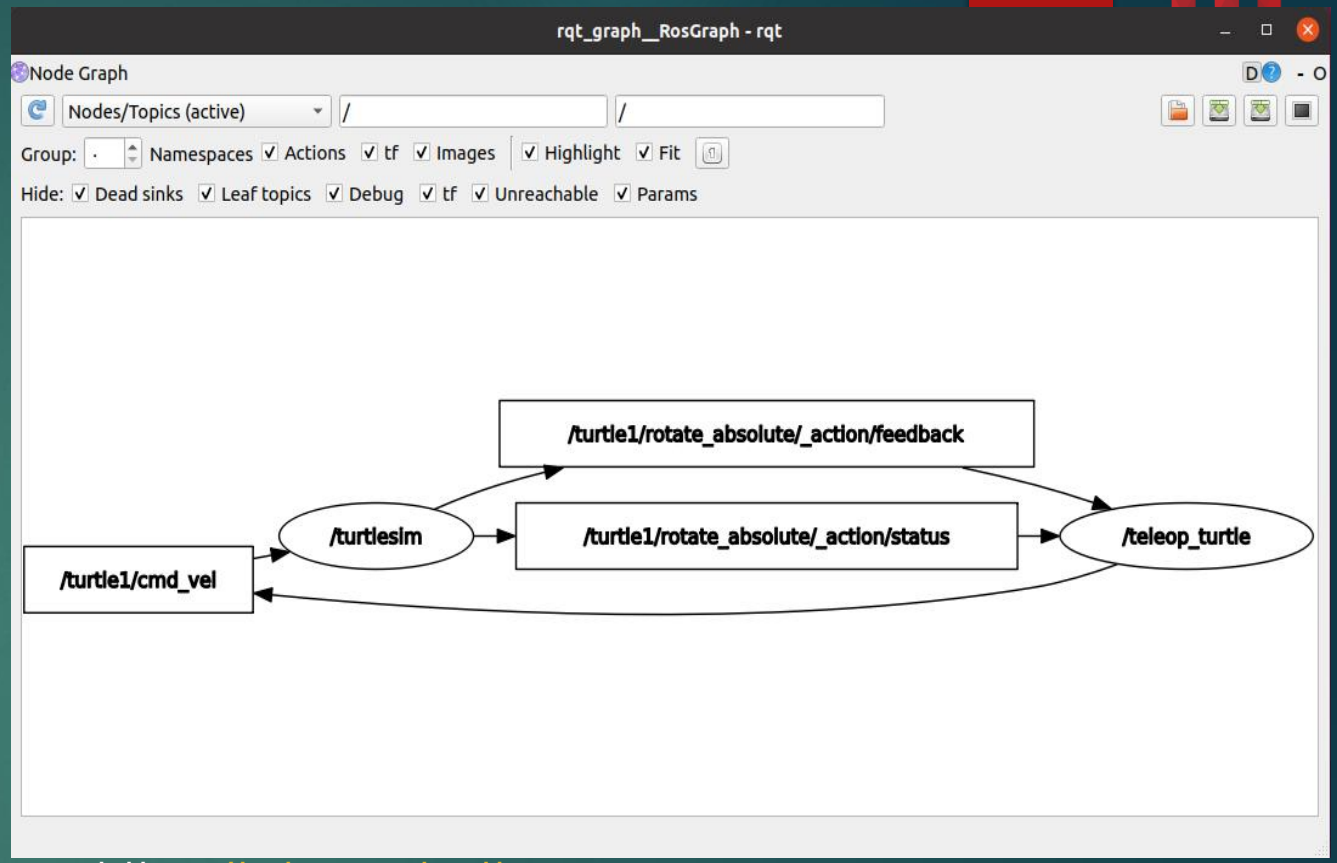
```
ros2 node info /turtlesim
```

- ros2 node info returns a list of subscribers, publishers, services, and actions (the ROS graph connections) that interact with that node. The output should look like this:

```
ahmed@ahmed-VirtualBox:~$ 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/SetParametersAtomic
ally
  Service Clients:

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

Rqt_graph



- Open a new terminal and run:

```
ros2 run turtlesim turtlesim_node
```

- Open another terminal and run:

```
ros2 run turtlesim turtle_teleop_key
```

- Now run rqt_grapgh in another terminal

```
rqt_graph
```

- You should see the above nodes and topic.
- The graph is depicting how the /turtlesim node and the /teleop_turtle node are communicating with each other over a topic. The /teleop_turtle node is publishing data (the keystrokes you enter to move the turtle around) to the /turtle1/cmd_vel topic, and the /turtlesim node is subscribed to that topic to receive the data.

C++ Publisher/ subscriber

- Create a ROS package

```
ros2 pkg create --build-type ament_cmake cpp_pubsub --dependencies rclcpp std_msgs
```

- Create nodes

```
touch publisher.cpp
```

- Add executable to CMakeLists.txt

```
add_executable(talker src/publisher.cpp)
ament_target_dependencies(talker rclcpp std_msgs)
install(TARGETS
  talker
  DESTINATION lib/${PROJECT_NAME})
```

- Make sure dependencies are correct in package.xml
- Build and run

```
colcon build --packages-select cpp_pubsub
ros2 run cpp_pubsub talker
```