USI/SUPSI IDSIA

Università della Svizzera italiana

Part 2

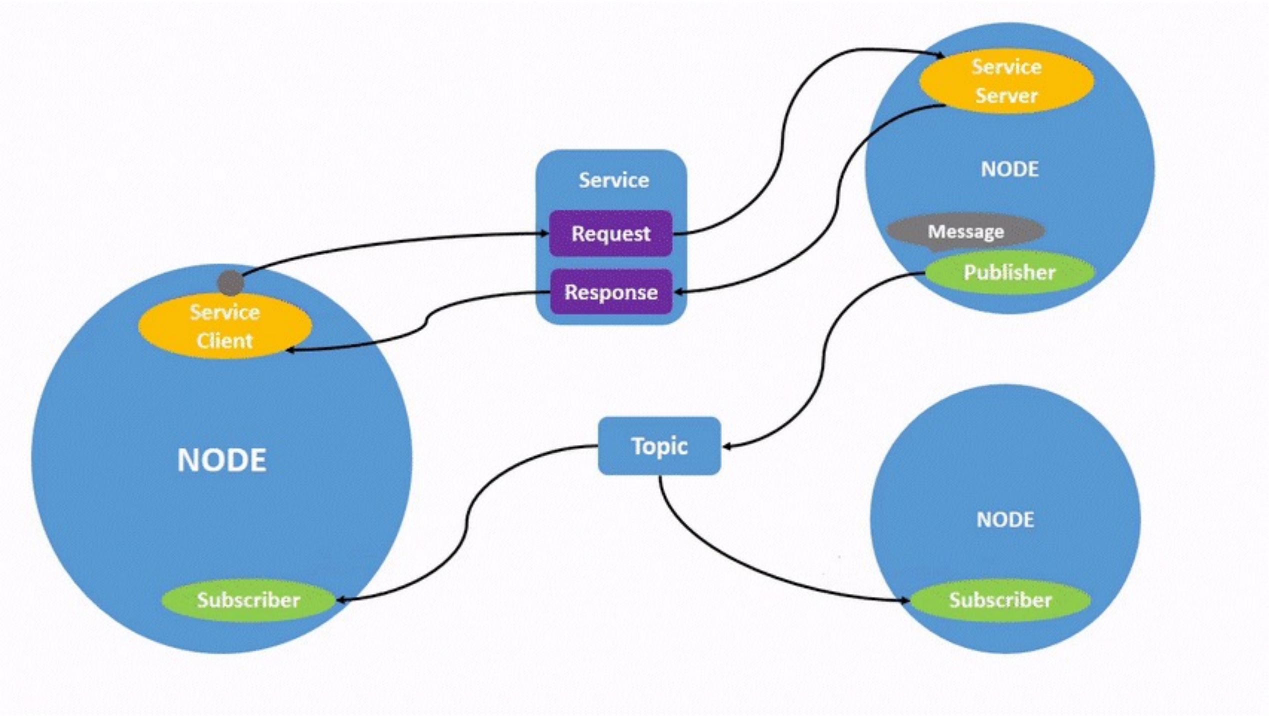
Introduction to ROS

Elia Cereda, Simone Arreghini

Introduction to ROS - Part 2

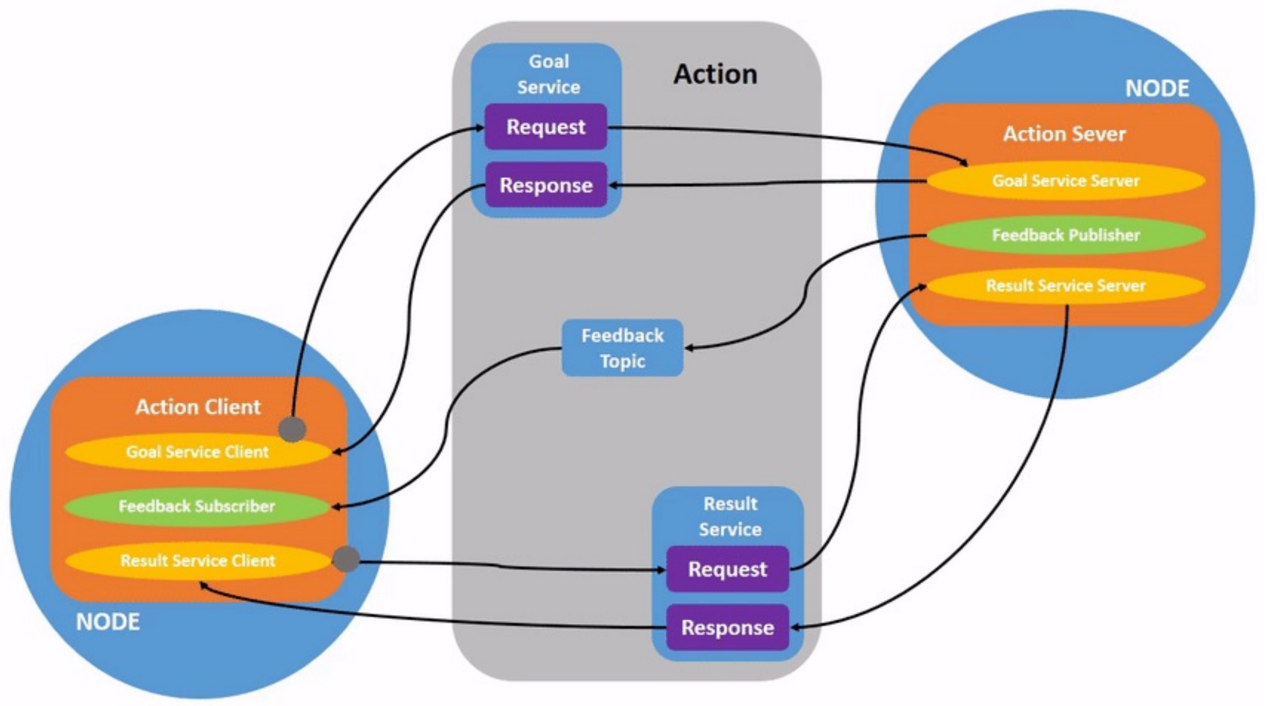
Recap

- **Computation Graph:** a peer-to-peer network of nodes that can communicate via Topics, Services and Actions.
- **Node**: Single-purpose executable program
  - Organized into packages
  - Focused on a single purpose (e.g. control a laser sensor)
  - Can act as client or server of actions and services



Recap

- **All communications are typed**
- **Message**: Typed data structure that contains data
- **Topic**: Communication channel for a stream of typed messages
  - Publish to (output) a topic
  - Subscribe to (input) a topic
- **Service**: Callable service function (request/reply model) taking messages as input and producing messages as output
- **Action**: Complex communication system based on Topic and Services used for long term operations



Teaser Homework 1

- On April 8 at 23:59 you will have to submit the first HW.
- You will use turtlesim. More info can be found here
- Most of what we will discuss today will be essential for the HW so pay attention.

Runtime tools

- `ros2 run <package> <executable_file>` — runs an executable file of a package
- `ros2 launch <package> <launch_file>` — launches multiple nodes from a single batch Python file
- `ros2 node <sub—command>` — displays information about nodes
  - `list` print all nodes
  - `info` print information about a specific node
- `ros2 topic <sub—command>` — displays information about topics
  - `list` print all active topics
  - `echo` print messages to screen
  - `pub` publish data to topic
  - `info` print information about a specific active topic
  - `hz` display publishing rate of topic
- `ros2 <interface> <sub—command>` — General ros2 CLI structure for communication interfaces
  - Interfaces:
    - `topic`
    - `service`
    - `action`

Introduction to ROS - Part 2

Runtime tools

- **`rqt_graph`** — displays a graphical representation of nodes in the ROS2 network
- **`rqt_plot`** — plots time series from numerical topics’ fields

Introduction to ROS - Part 2

ROS2 Filesystem

- **Package** container used to organize software in ROS2. Can contain nodes, libraries, datasets, configs, …
- **Workspace** the base directory in which to store packages, develop, build and install them (throughout this course it will be **`~/dev_ws/`**)

Introduction to ROS - Part 2

Packages

- Unit container that can be built for ROS2 code
- This is the correct way to share your code with others by organizing it in packages.
- ROS2 uses `colcon` as build tool
- The package can be created either using CMake or Python (other methods are also supported)
- Every package needs a set of files depending on the language chosen (see https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html)
- A package may contain datasets, configuration files, message and service definitions, or anything else required to build and run the package

Introduction to ROS - Part 2

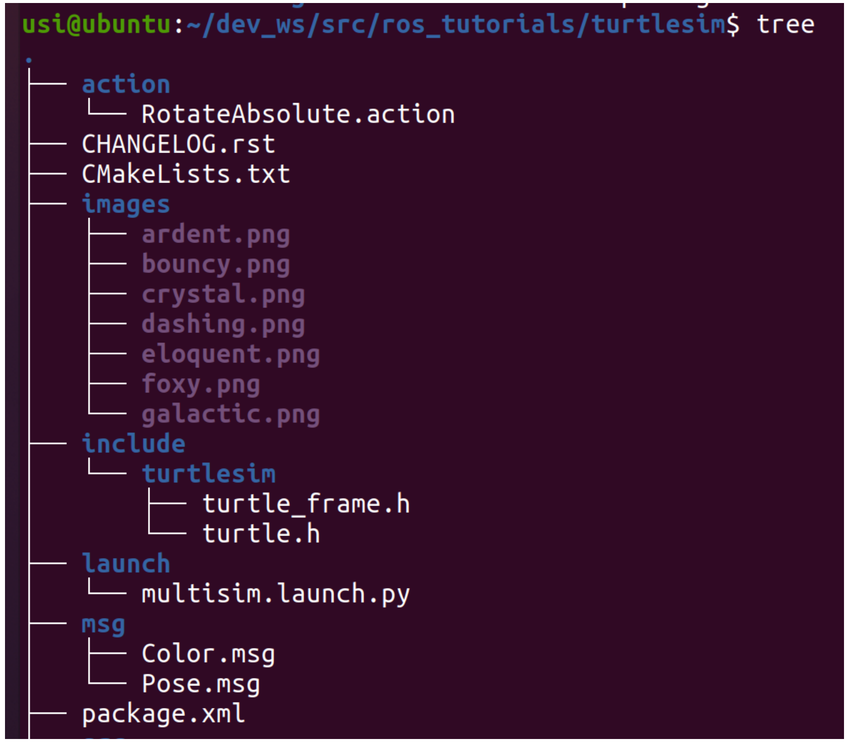
Packages

- Packages can contain
  - Source code
  - Launch files
  - Configuration files
  - Message and service definitions
  - Documentation
  - …
- A package may be built upon other packages (referred to as dependencies), e.g., other code, message definitions

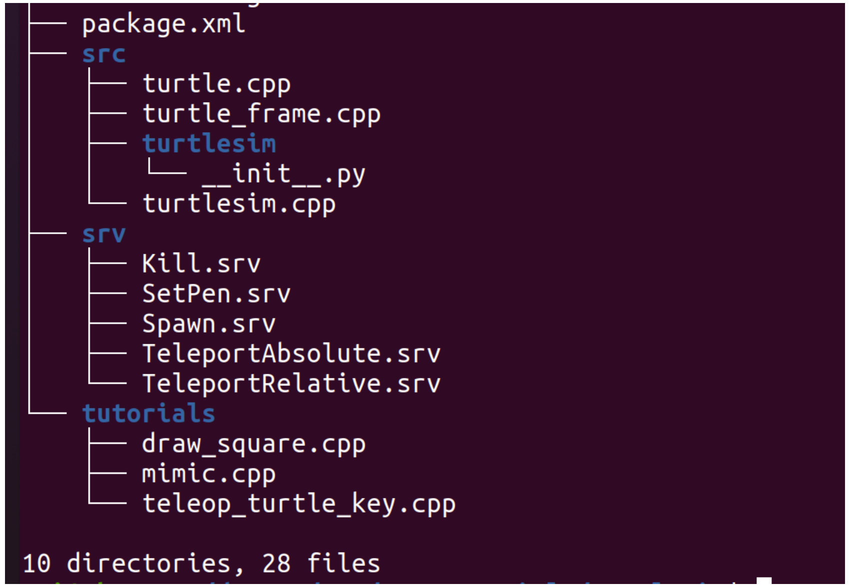
Introduction to ROS - Part 2

Example of a package

- This is an example of a package that now is part of ROS2 (it’s not in your dev_ws folder).



```
usi@ubuntu:~/dev_ws/src/ros_tutorials/turtlesim$ tree
.
├── action
│   └── RotateAbsolute.action
├── CHANGELOG.rst
├── CMakeLists.txt
├── images
│   ├── ardent.png
│   ├── bouncy.png
│   ├── crystal.png
│   ├── dashing.png
│   ├── eloquent.png
│   ├── foxy.png
│   └── galactic.png
├── include
│   └── turtlesim
│       ├── turtle_frame.h
│       └── turtle.h
├── launch
│   └── multisim.launch.py
├── msg
│   ├── Color.msg
│   └── Pose.msg
├── package.xml
```



```
├── package.xml
├── src
│   ├── turtle.cpp
│   ├── turtle_frame.cpp
│   ├── turtlesim
│   │   └── __init__.py
│   └── turtlesim.cpp
├── srv
│   ├── Kill.srv
│   ├── SetPen.srv
│   ├── Spawn.srv
│   ├── TeleportAbsolute.srv
│   └── TeleportRelative.srv
└── tutorials
    ├── draw_square.cpp
    ├── mimic.cpp
    └── teleop_turtle_key.cpp

10 directories, 28 files
```

Workspaces

- A directory containing different ROS2 **packages**
- Can be combined as layers of **overlay** workspaces
- Underlying workspaces must contain all **dependencies** of the overlays. Duplicated packages in the **overlay** will override the underlying workspaces.
- Workspaces must be built and sourced

VM instructions

- By default, we source only the base ROS2 installation.
- Any additional **workspace** must be sourced through their **`setup.bash`** file. It will act as an **overlay,** a secondary workspace over the default one.
- The main workspace where you will work is **`~/dev_ws`**

Package command-line tools

- **`ros2 pkg list`** — print all available packages and their locations

Create a new package

- **`ros2 pkg create --build-type ament_python <package_name>`**
  create a new Python package in the current directory. Optional arguments:
  - **`--dependencies <dep_name1> <dep_name2>`** … add your package's dependencies
  - **`--node-name <node_name>`** also create an example node inside the new package

Be sure you are in `~/dev_ws/src` when creating packages

Package command-line tools

Build the workspace (after every change to your packages)

- If you were working on a package, remember to `cd ~/dev_ws`
- `colcon build` build all the packages in the workspace
  - The built packages can be found in `~/dev_ws/install`

Be sure to be in `~/dev_ws` (the root of your workspace) when building!

Using your packages (after every change to your packages)

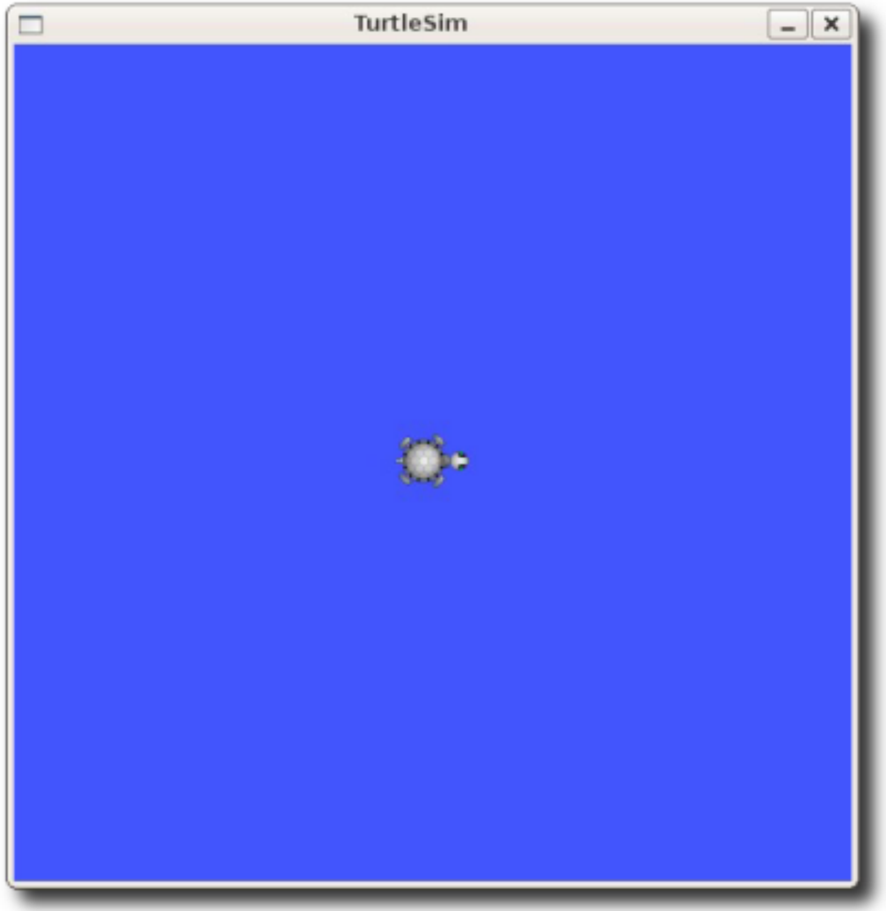
- `source ~/dev_ws/install/setup.bash` source your workspace, so that its packages become available for use

Use different terminals for building and executing packages

you cannot build a workspace while it is currently sourced in that terminal

Introduction to ROS - Part 2

Exercise 1: TurtleSim



Introduction to ROS - Part 2

Exercise 1: TurtleSim

- Open **a terminal** and type `ros2 run turtlesim turtlesim_node`
  - Notice that a new window has opened with a turtle in the middle of it
- Open **another terminal** and type `ros2 node list`
  - Notice that there's now a new node called "turtlesim"
- Type `ros2 node info /turtlesim` to see more details on the node's topics, services, and connected nodes
- Type `ros2 topic info /turtle1/cmd_vel` to see the topic's info
  - This topic is used to send control commands to the robots, or turtles
  - Type `ros2 interface show geometry_msgs/msg/Twist` to see the data fields of this topic (linear and angular velocities)

Exercise 1: TurtleSim

- Let's move the turtle! Type in the terminal

```
ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "linear:
  x: 0.1
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0" -r 1
```

- Wow! The turtle is now moving!!! Have fun experimenting with different linear and angular velocities

Introduction to ROS - Part 2

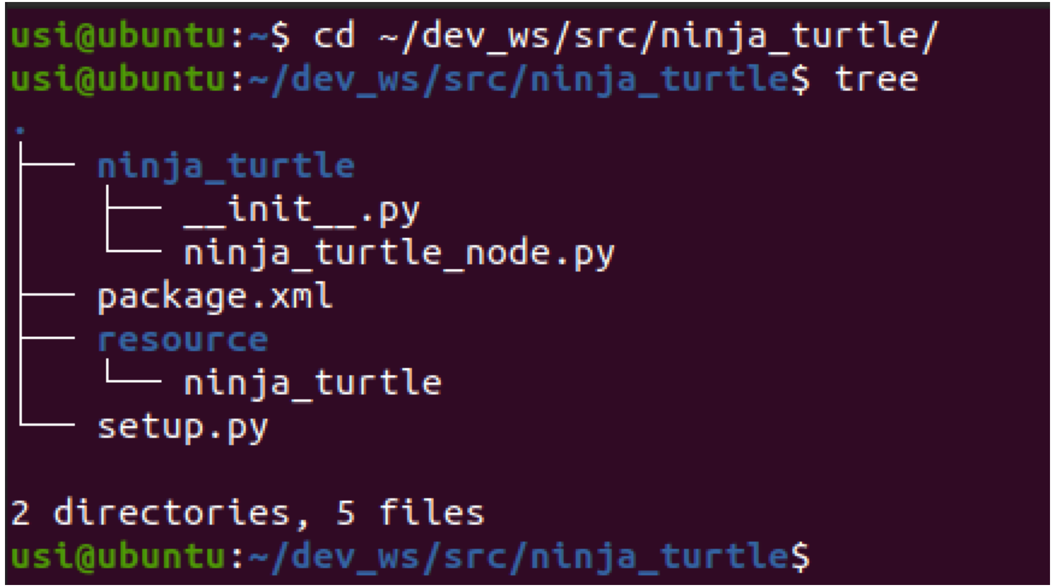
Exercise 2: Ninja Turtles

Exercise 2: Ninja Turtles

- Clone this repository https://github.com/EliaCereda/ninja_turtle in your workspace with:

```
cd ~/dev_ws/src
git clone https://github.com/EliaCereda/ninja_turtle.git
```

- You should have something like the following (use the command `tree` to display the folder structure)



```
usi@ubuntu:~$ cd ~/dev_ws/src/ninja_turtle/
usi@ubuntu:~/dev_ws/src/ninja_turtle$ tree
.
├── ninja_turtle
│   ├── __init__.py
│   └── ninja_turtle_node.py
├── package.xml
├── resource
│   └── ninja_turtle
└── setup.py

2 directories, 5 files
usi@ubuntu:~/dev_ws/src/ninja_turtle$
```

Exercise 2: Ninja Turtles

- Inside **`~/dev_ws/src/ninja_turtle/ninja_turtle`** there's a node (Python script) that connects to the topic **`/turtle/cmd_vel`** and moves the turtle
- We will use this script in a bit, but first we need to build it:
  - Run **`colcon build`** from the workspace root ***`~/dev_ws`***
  - If everything goes to plan, it will successfully compile all packages (it may take a while)

Exercise 2: Ninja Turtles

- **Open two new terminals** and source the workspace
  - `source ~/dev_ws/install/setup.bash`
  - Run it **after every colcon build** or you will not see the latest changes

Remember: one terminal for building packages and one (or more) for executing nodes

- Check that the environment is set up correctly
  - Type `ros2 pkg list | grep ninja`, you should see a package named *ninja_turtle*
- Now let's run the script inside the *ninja_turtle* package
  - In the first terminal, type `ros2 run turtlesim turtlesim_node`
  - In the second terminal, run `ros2 run ninja_turtle ninja_turtle_node`
  - The turtle should be moving by now!

Exercise 2: Ninja Turtles

- TurtleSim is versatile and supports **multiple turtles**
- Take a look at `ros2 service list`
  - You should see a `/spawn` service, which asks TurtleSim to spawn a new turtle
- **What is the command to invoke this service?**
- Now that multiple turtles are on the screen, we must choose which one to move
  - Let's have a look at our `ninja_turtle` package. The node script is in file `~/dev_ws/src/ninja_turtle/ninja_turtle/ninja_turtle_node.py`
  - Try to change it to control the other turtle

The first homework will consist in **controlling two turtles using a script,** similar to `ninja_turtle_node.py`

Let’s use the Python

tiger-in-the-flightdeck

The lack of context here is thrilling

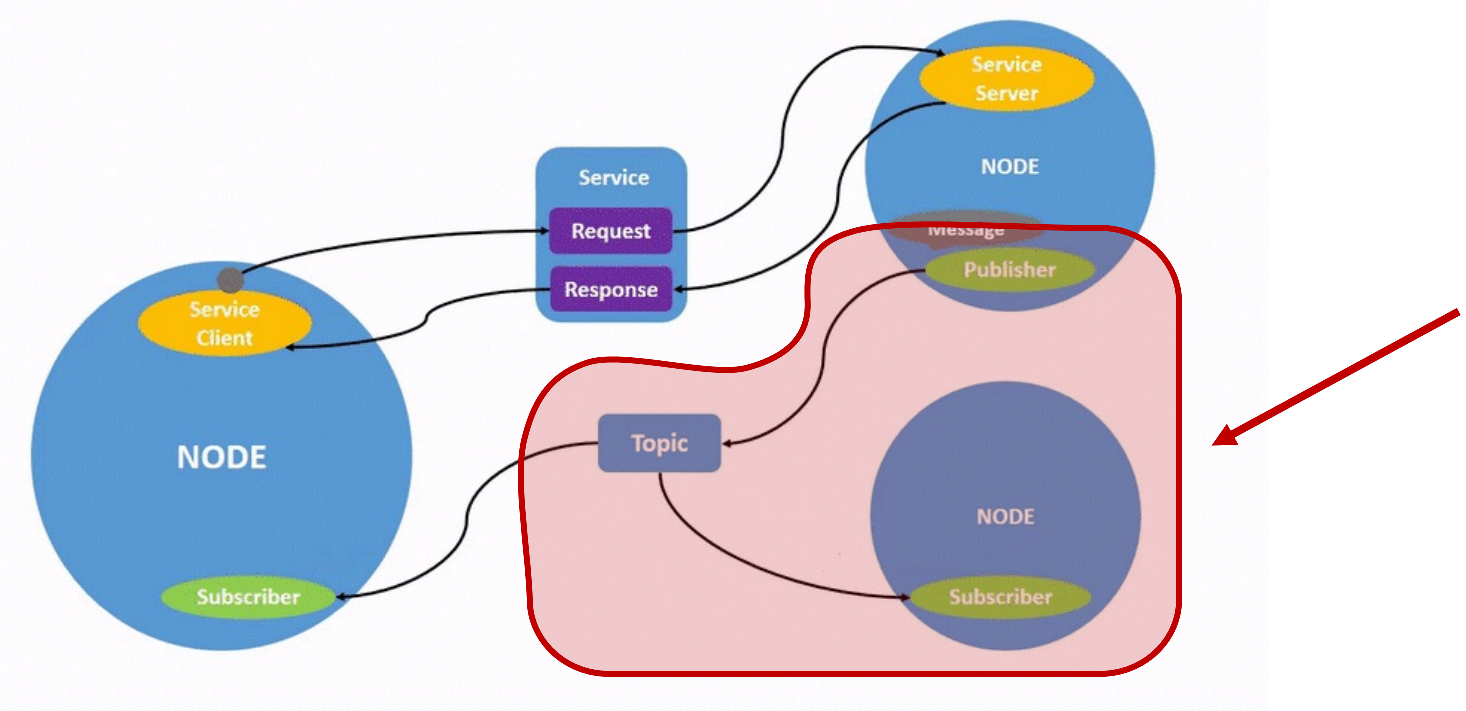
mark-watney-spacepirate

introductory python programming course

Introduction to ROS - Part 2

Exercise 3: Resources

- Writing a publisher and subscriber https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html



rclpy API Reference

We built the documentation for the ROS2 Python Client Library since the online version is outdated.

https://www.icorsi.ch/mod/resource/view.php?id=1040432

This is a zip of the whole documentation. To use it, open the index.html file

NB: This will be useful even for the HWs and Project

Exercise 3: ROS HelloWorld

- The goal of this exercise is to get you acquainted with the ROS Python API and the publisher/subscriber architecture
- Remember our first exercise? Publishing our name and displaying it to console
- Now we will replicate it with a full-fledged ROS package in Python

Exercise 3: Your turn

- Navigate to the *workspace* source folder with `cd ~/dev_ws/src`
- Create a new package called *ros_helloworld* with

```
ros2 pkg create --build-type ament_python ros_helloworld
```

- Move into the package folder by `cd ros_helloworld`
  - Display the package structure with the command `tree`
- Download the two Python files from here

https://www.icorsi.ch/mod/folder/view.php?id=1040459

- Copy them into the package's code folder, `ros_helloworld`, and make them executable:

```
chmod +x ros_helloworld/*.py
```

- Remember to do `colcon build` and source again the workspace
- Complete the missing parts in the two *.py files and show us your results

Introduction to ROS - Part 2

Homework 1

- Today we are giving you the first homework
- Link: https://www.icorsi.ch/mod/assign/view.php?id=1040461
- You must work alone; **no groups allowed**
- The deadline is at **23:59 of the 8th April**
- The homework will be about using turtlesim

Calendar (tentative)

26 March	Homework assistance — half lecture
8 April@23:59	HW1 first deadline
9 April	HW1 solution, ROS Lab Part 3 lecture, HW2 give out
19 April	Homework assistance — half lecture
22 April@23:59	HW2 deadline
23 April	HW2 solution, final projects give out
30 April	Project pitch presentations: validate and refine your project idea in class
7 May (tentative)	Project half-way presentations: present the first iteration to gather feedback
31 May	Project deadline: 3/5 min presentation with final results + short written report