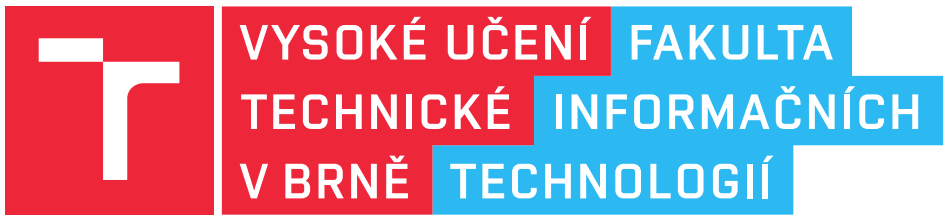
ROS & ROS2: What, why, how...?

Jan Beran

Fakulta informačních technologií Vysokého učení technického v Brně
Božetěchova 1/2. 612 66 Brno – Královo Pole

iberan@fit.vut.cz



November 21, 2024

Overview

ROS

- Basics
- Simple demo
- Commands, tools
- Packages demo
- Turtlebot demo

ROS2

- Differences
- Turtlebot2 demo
- PubSub2 demo

ROS

What is ROS?

According to ChatGPT:

ROS stands for Robot Operating System. It's a flexible framework for writing robot software. Despite its name, ROS is not an actual operating system; rather, it provides services like hardware abstraction, device drivers, communication between processes, and package management. It's widely used in the field of robotics for developing and controlling robots.

What is ROS?

According to ChatGPT:

ROS stands for Robot Operating System. It's a flexible framework for writing robot software. Despite its name, ROS is not an actual operating system; rather, it provides services like hardware abstraction, device drivers, communication between processes, and package management. It's widely used in the field of robotics for developing and controlling robots.

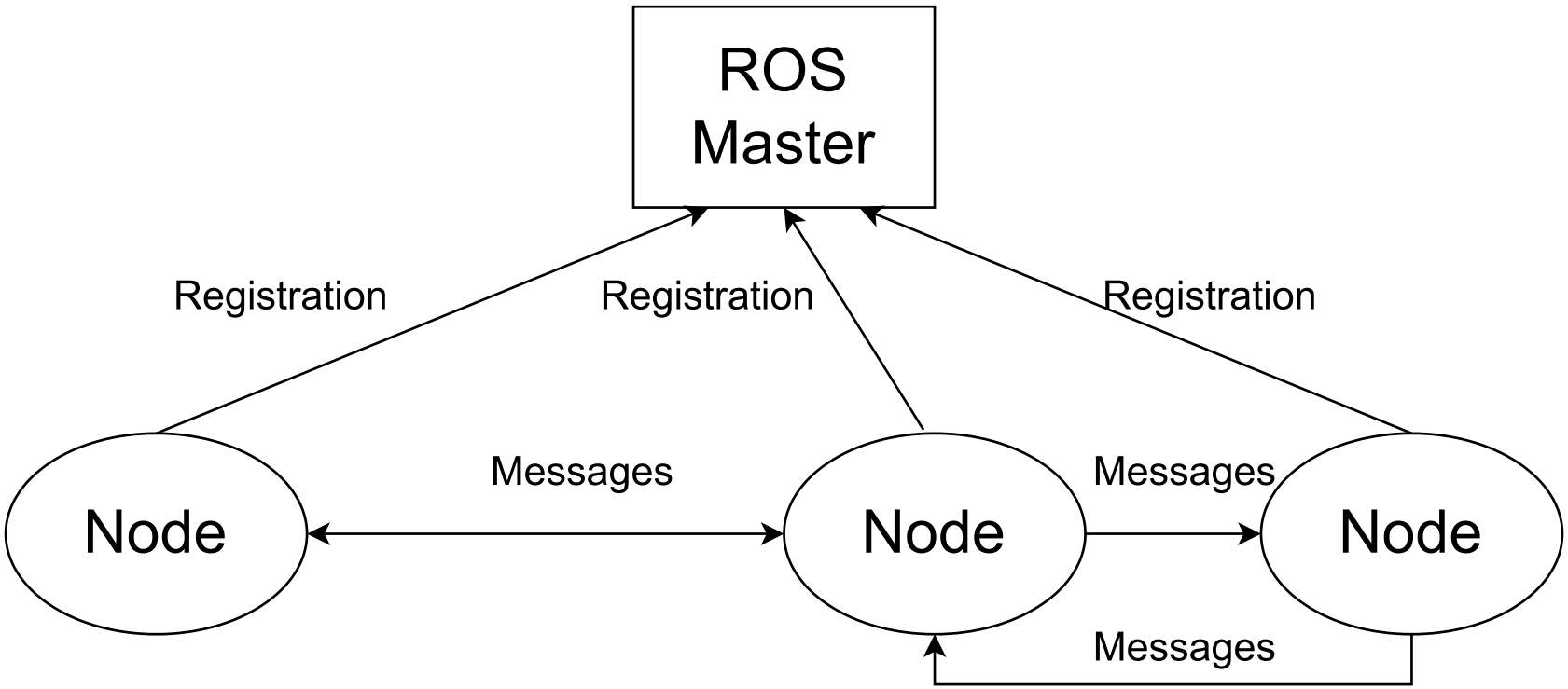
- In other words...
  - (Not only) open–source framework used in robotics.
  - Also tools, libraries, wrappers, adapters and conventions.
  - Node–based, centralised (will be covered later today).
  - Three main ways of how the nodes can exchange the data: publisher–subscriber, service, action. (will be covered).
  - Currently being put aside in favor of ROS2 :((will be covered, incl. why we are talking mainly about ROS 1).

According to ChatGPT:

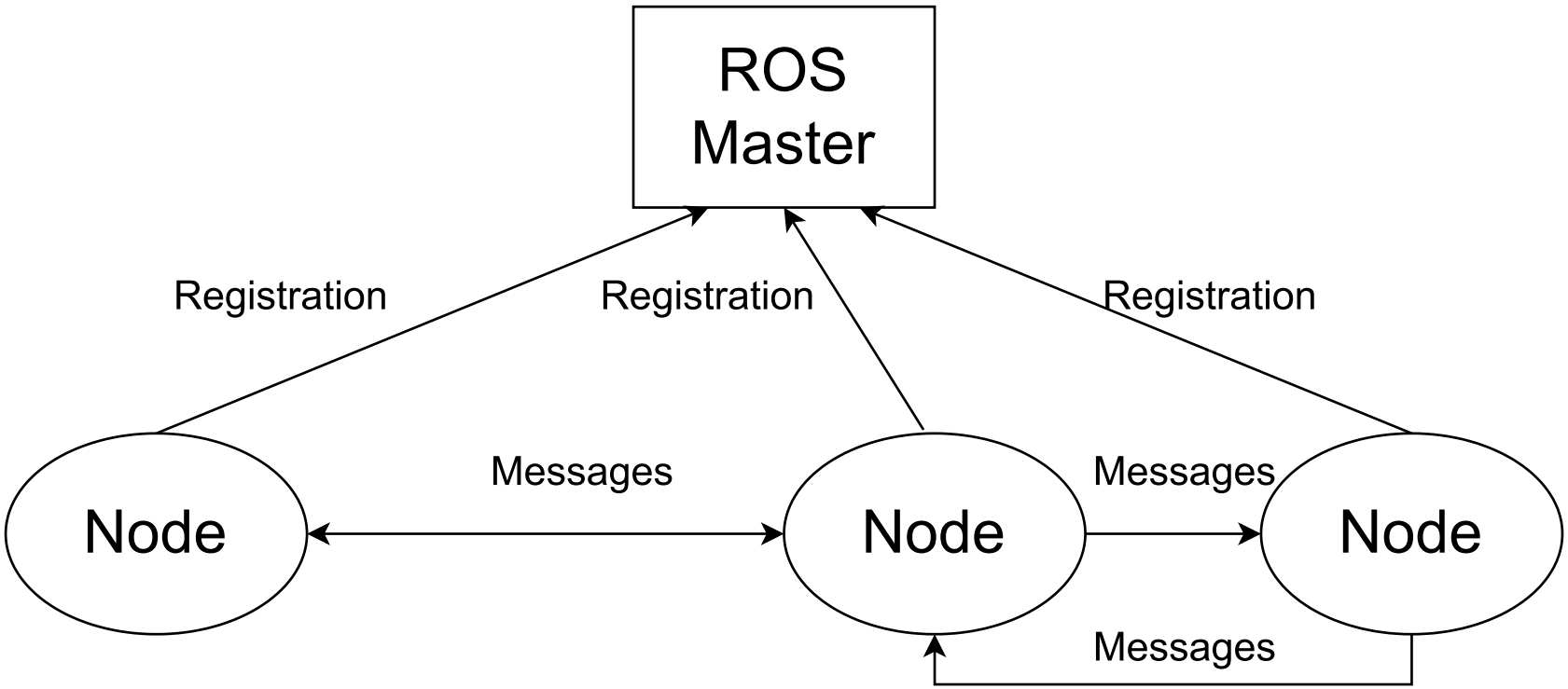
ROS stands for Robot Operating System. It's a flexible framework for writing robot software. Despite its name, ROS is not an actual operating system; rather, it provides services like hardware abstraction, device drivers, communication between processes, and package management. It's widely used in the field of robotics for developing and controlling robots.

- In other words...
  - (Not only) open–source framework used in robotics.
  - Also tools, libraries, wrappers, adapters and conventions.
  - Node–based, centralised (will be covered later today).
  - Three main ways of how the nodes can exchange the data: publisher–subscriber, service, action. (will be covered).
  - Currently being put aside in favor of ROS2 :((will be covered, incl. why we are talking mainly about ROS 1).
  - `http://wiki.ros.org/Documentation`

(Basics of) ROS in one picture



(Basics of) ROS in one picture



Not entirely true... You will see.

Why ROS?

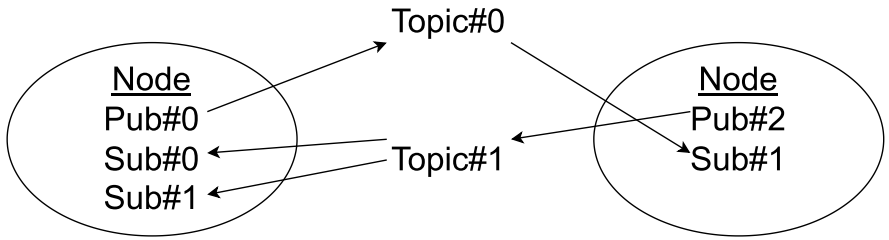
- Open–source
- Very versatile, widely used
- Many algorithms already implemented, clear, debug–friendly structure
- Functionality isolated into nodes and packages
- Used in space!
  - Robonaut2
  - `https://robots.ros.org/robonaut2/`

- **ROS master:** registers other nodes. Enables other nodes to locate one another[1].
  - You are mostly shadowed from it... Just run `roscore` and you are fine.
  - Wanna know more?
    -> `http://wiki.ros.org/ROS/Technical%20Overview`
- **Nodes:** One program in C++/Python which does one thing (moving, translating, reading from sensors...)
- **Topics:** "channels" used for communication. Structured like this: *general/specific/more-specific* (*robot/arm/joints/femur*). Every topic has defined...
- **...Message:** data structure, describing the format of data sent via the topic.

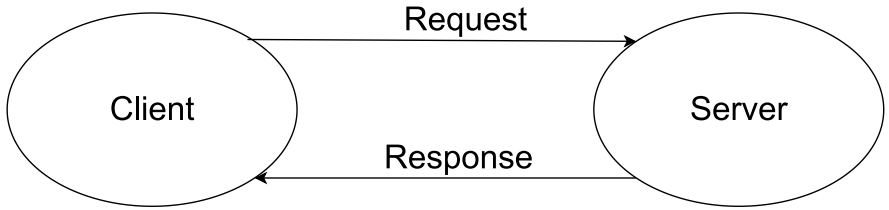
[1] The communication per se is P2P

Three ways of communication

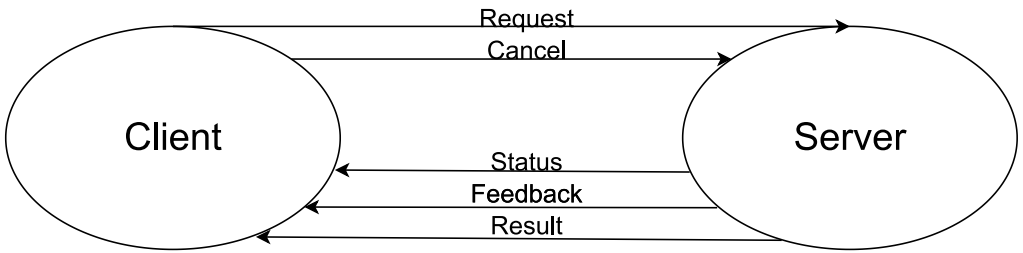
1. **Publisher–subscriber** via **topics** and **messages**
   - `http://wiki.ros.org/Topics`



2. **Services**
   - `http://wiki.ros.org/Services`



3. **(Actions)**
   - `http://wiki.ros.org/actionlib`



- Data structures (almost identical to C–like structures)
- Specify the data sent via specific topics
- **Use standard messages if available:**
  - `rosmsg show std_msgs` or `rosmsg show common_msgs`
  - `http://wiki.ros.org/std_msgs` (basic messages and datatypes)
  - `http://wiki.ros.org/common_msgs` (more complex widely used messages)

geometry_msgs/Pose Message

File: geometry_msgs/Pose.msg

Raw Message Definition

```
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

Compact Message Definition

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

geometry_msgs/Point Message

File: geometry_msgs/Point.msg

Raw Message Definition

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

Compact Message Definition

```
float64 x
float64 y
float64 z
```

Simple demo

Every command in new terminal window:

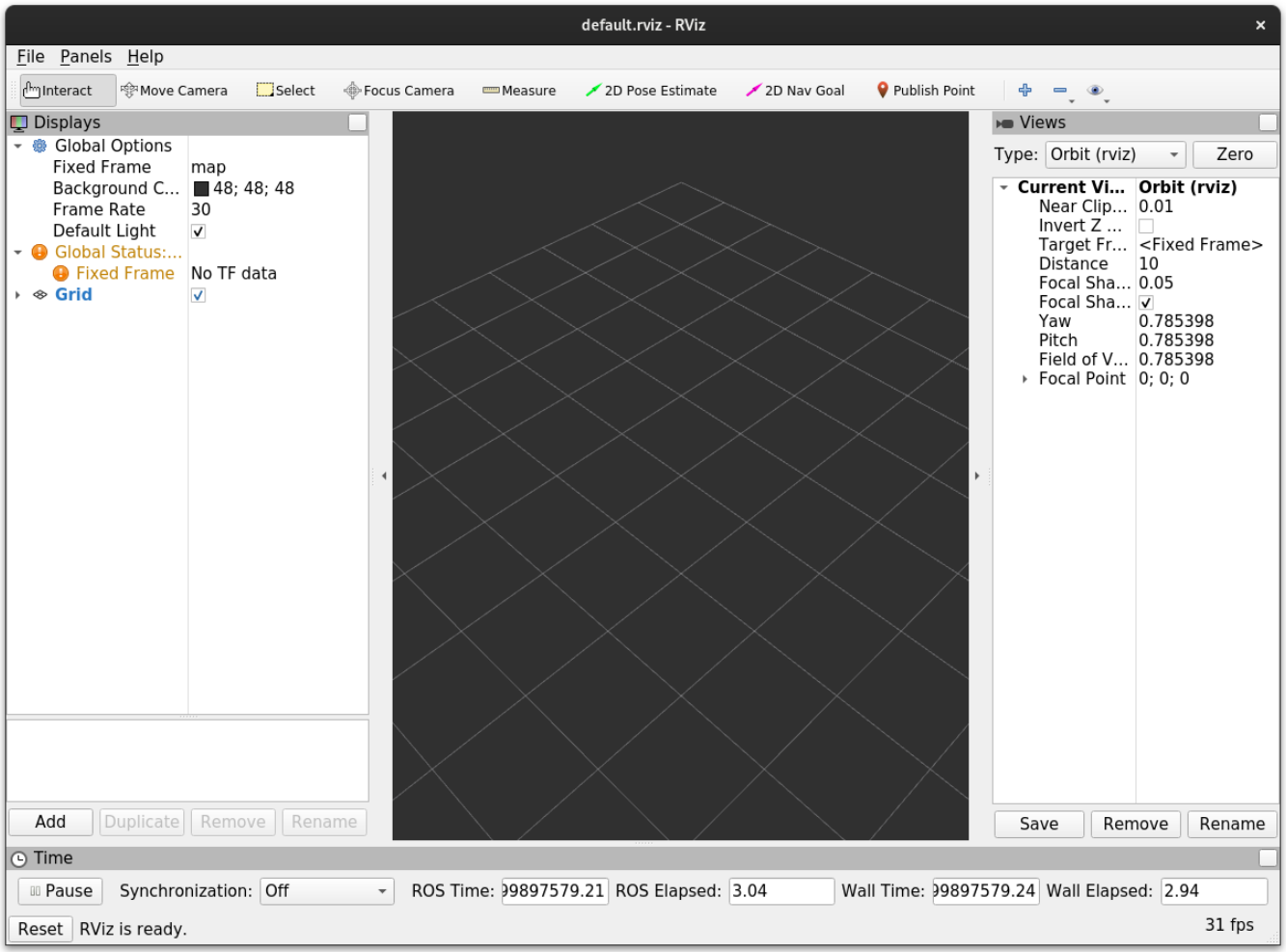
- `source /opt/ros/noetic/setup.bash`
- Run ROS master: `roscore`
- Make periodic publisher:
  `rostopic pub -r 1 /ROS/demo std_msgs/Int32 "data: 42"`
  - `-r 1` means 1 message per second
- Subscribe: `rostopic echo /ROS/demo`
- Check what topics are available: `rostopic list`
- Show info about some topic: `rostopic info /ROS/demo`
- Visualize the graph: `rqt_graph`

Commands, tools...

Commands

- `roscore`: starts ROS master node – first command you run.
- `rostopic`: work with topics and messages from command line
  - `pub [-r rate] topic message data`: publish messages to `topic`
  - `echo topic`: print messages to stdout from `topic`
  - `list`: list all available topics
  - `info topic`: get info about `topic` (pubs, subs, message type...
- `rosrun package node`: run one `node` from given `package`
- `roslaunch package launchfile`: run pre–made set of nodes (no need to run `roscore` separately).
- `roscd package`: like `cd`, but moves you directly to ROS `package`

- `rqt_graph`: visualizes nodes and topics.
- Other `rqt`–based commands.
- `rviz`
- `https://www.youtube.com/watch?v=34n1tF5OtQU`



Demo with packages and Python

How to make a ROS package

- http://wiki.ros.org/catkin/Tutorials/create_a_workspace
- http://wiki.ros.org/ROS/Tutorials/CreatingPackage
- http://wiki.ros.org/ROS/Tutorials/BuildingPackages
- mkdir -p /catkin_ws/src
- cd /catkin_ws/
- source devel/setup.bash
- catkin_create_pkg <package_name> <dependency1> <dependency2>...
- catkin_make

Demo with packages and Python

- `http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29`
- Using Python, we will mimic:
  - `rostopic pub` with `talker.py`
  - `rostopic echo` with `listener.py`

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String

def talk():
    pub = rospy.Publisher("ROS_demo/talker", String,
                          queue_size=10)
    rospy.init_node("talker_node", anonymous=True)
    rate = rospy.Rate(2)
    cnt = 0
    while not rospy.is_shutdown():
        pub.publish("Hello World! {}".format(cnt))
        cnt += 1
        rate.sleep()

if __name__ == "__main__":
    try:
        talk()
    except rospy.ROSInterruptException: pass
```

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String


def callback(data):
    print("I received this: {}".format(data.data))


def listen():
    rospy.Subscriber("ROS_demo/talker", String, callback)
    rospy.init_node("listener_node", anonymous=True)
    rospy.spin()


if __name__ == "__main__":
    listen()
```

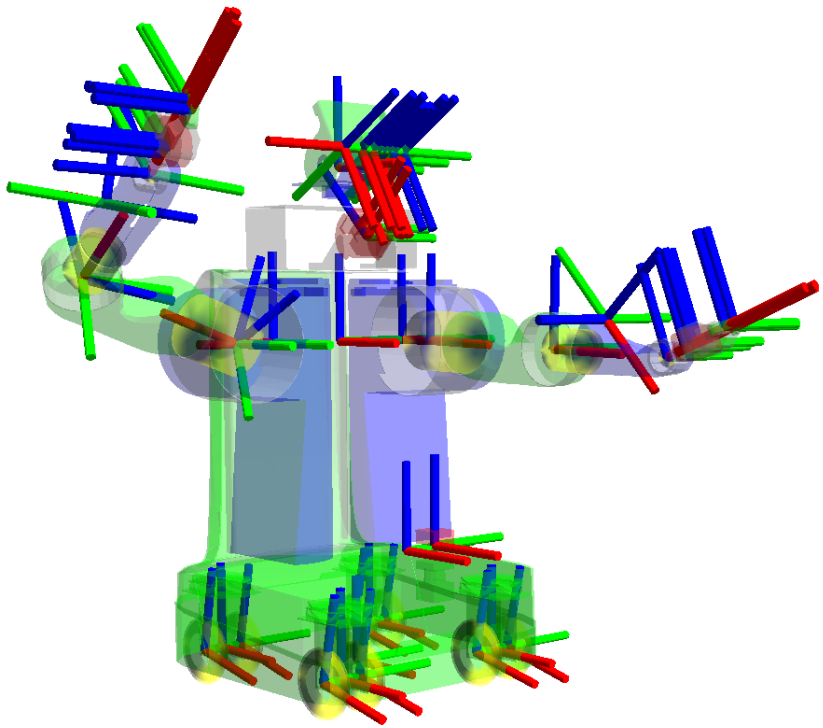
Turtlebot demo

- Beginner–friendly simulator
- (`roscore`)
- `rosrun turtlesim turtlesim_node`: start basic node
- `rosrun turtlesim turtle_teleop_key`: start teleoperation (move by keyboard)

- Robots needs to move themselves and their parts
- Every part has its own coordinate frame (start of coordinates)
- How to translate and bound them together?

- Robots needs to move themselves and their parts
- Every part has its own coordinate frame (start of coordinates)
- How to translate and bound them together?
- `tf`!

- Robots needs to move themselves and their parts
- Every part has its own coordinate frame (start of coordinates)
- How to translate and bound them together?
- `tf`!
- = library that represents and translates coordinate frames of every parts of the robot



- `sudo apt-get install ros-$ROS_DISTRO-rosserial-arduino`
- `sudo apt-get install ros-$ROS_DISTRO-rosserial`
- install Arduino IDE from source
- `cd ~/Arduino/libraries; rosrun rosserial_arduino make_libraries.py .`

ROS2

Why new ROS?

- ROS 1 started as a research project (and almost as a hobby).
- Some decisions locked ROS in previous decade:
  - Centralization
  - Custom protocols TCPROS and UDPROS
  - Poor network performance
- ROS2 fixes most of these things.

- Completely new communication paradigm: **DDS**.
  - Decentralized
  - Standardized
  - Several implementations(!)
- New paradigm, commands, workflow... :(

DDS in ROS2

- ROS2 using its own middleware over DDS: rmw.
- DDS itself can be of different implementations
- Distributed by implementation
- Nodes organized into domains (areas of communication)

Note: So why to use ROS1?

- TLDR: Don’t. :)

Note: So why to use ROS1?

- TLDR: Don’t. :)
- **BUT…**

Note: So why to use ROS1?

- TLDR: Don’t. :)
- **BUT...**
- ROS 1 is still widely used in existing projects.
- Some technologies do not work in ROS 2 (e.g. 8–bit MCUs, such as Arduino UNO/Mega)
- ROS 2 is still not as widely adopted as ROS 1 (something like Windows XP problem :))

Note: So why to use ROS1?

- TLDR: Don’t. :)
- **BUT…**
- ROS 1 is still widely used in existing projects.
- Some technologies do not work in ROS 2 (e.g. 8–bit MCUs, such as Arduino UNO/Mega)
- ROS 2 is still not as widely adopted as ROS 1 (something like Windows XP problem :))
- **–> Do not use ROS 1 in new project, but learn it because of existing ones! There are plenty of them – and they will be here for years and decades!**

Turtlebot2 demo

- `export ROS_DOMAIN_ID=<your_domain_id>`
- If you do not want to use ROS2 on multiple machines: `export ROS_LOCALHOST_ONLY=1`
- `ros2 run turtlesim turtlesim_node`
- `ros2 run turtlesim turtle_teleop_key`
- `/opt/ros/foxy/bin/rqt`[2]
  - Plugins –> Services –> Service caller
  - /spawn (change X,Y, name – to turtle42)
- `ros2 run turtlesim turtle_teleop_key -ros-args -remap turtle1/cmd_vel:=turtle42/cmd_vel`

[2] Because I have two ROS instances, `rqt` does not work.

PubSub2 demo

PubSub2 demo

- (close terminal and disable sourcing ROS1 in .bashrc)
- `https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html`
- `https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html#new-directory`
- `mkdir -p ~/ros2_ws/src`
- `cd ~/ros2_ws/src`
- `ros2 pkg create –build-type ament_python py_pubsub`
- *do some magic...*[3]
- `sudo apt install python3-colcon-common-extensions`
- `source install/setup.bash`
- `ros2 run py_pubsub talker`
- `ros2 run py_pubsub listener`

[3] Actually create the publisher and subscriber according to the tutorials above.

Questions?