Introducing ROS 2

UNIVERSITY OF
EASTERN FINLAND

Henki
Robotics

George-Cosmin Porusniuc
Co-Founder & Robotics Consultant @ Henki Robotics

During this lecture…

1. Short recap - What is a robot. The general architecture of a robot
2. ROS 2 - What, Why, How?
3. Quick demo with a simulated robot
4. Docker - short introduction
5. ROS 2 cheat sheets and resources

A short recap

What is a robot?

- In **ISO 8373:2021 Robotics**:

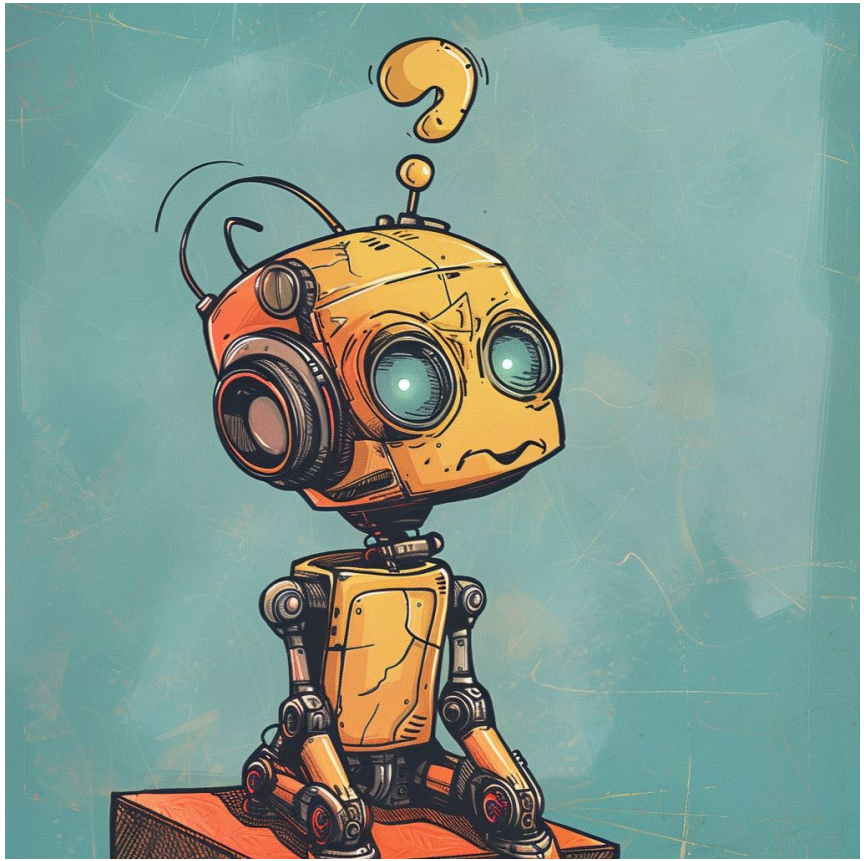
  A robot is a "*programmed actuated mechanism with a certain degree of **autonomy** (3.2) and capable to perform locomotion, manipulation or positioning*".

- Andreas Bihlmaier in the ***Robotics for programmers*** book says that:

  There are 2 requirements for a system to be considered a robot:

  (1) having a significant direct physical interaction with the real world

  (2) has the ability to change its behaviour and tasks according to the program being executed



Definition of robotics systems

Robots shall consist of mechanical, electrical and electronic components

Actuators and sensors

Be physical agents affecting their environment

Capability to perform an action

Have a well-defined task, e.g., to perform locomotion, manipulation, positioning, reconnaissance or delivery

And , potentially, perform more than one singular task.. The direction in modern robotics is towards general purpose robots

Tamas Haidegger, "*Taxonomy and Standards in Robotics*". In Marcelo H. Ang, Oussama Khatib, and Bruno Siciliano (eds), Encyclopedia of Robotics, Springer Nature, 2021

The general architecture of a robot

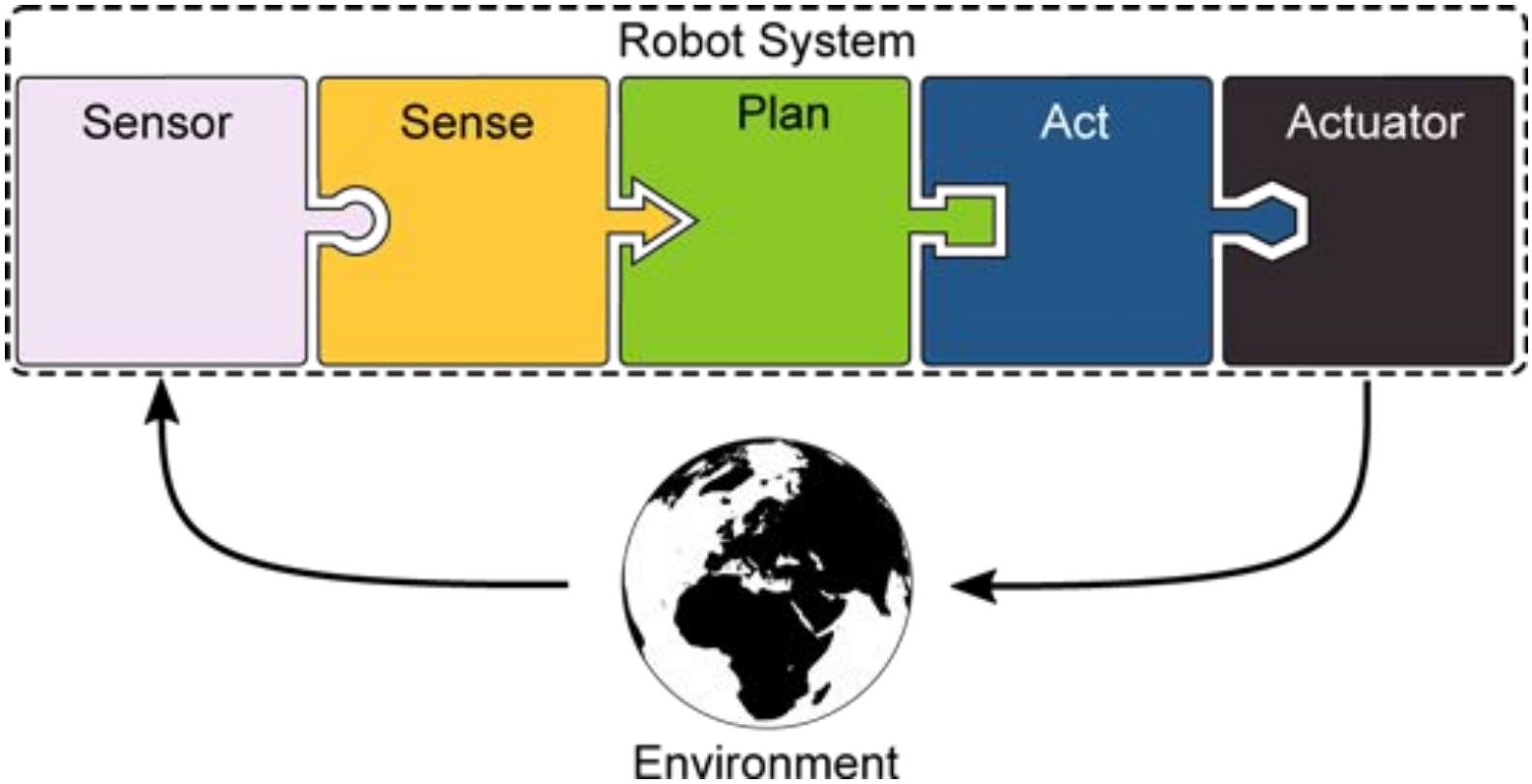
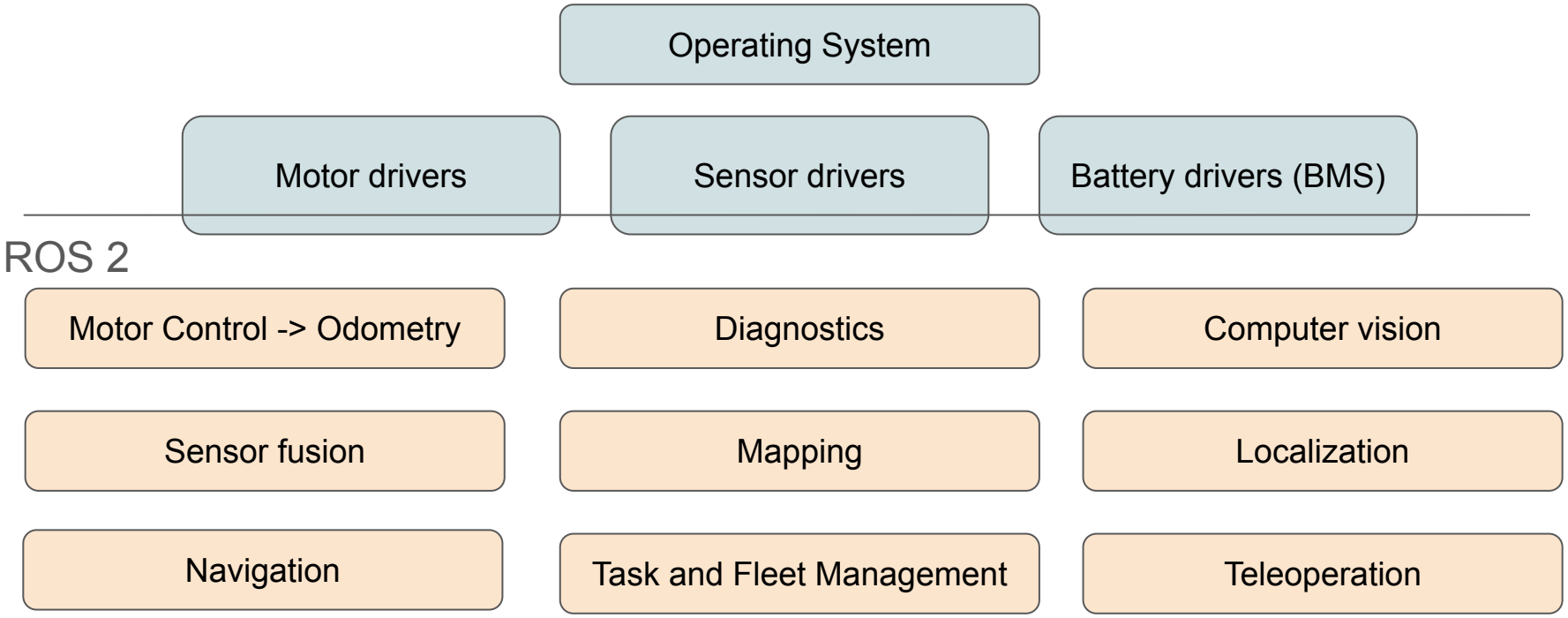


Image source: Robotics for Programmers by Andreas Bihlmaier (MEAP version)

The hardware architecture of a mobile robot

Image source: https://husarion.com/manuals/rosbot/#components

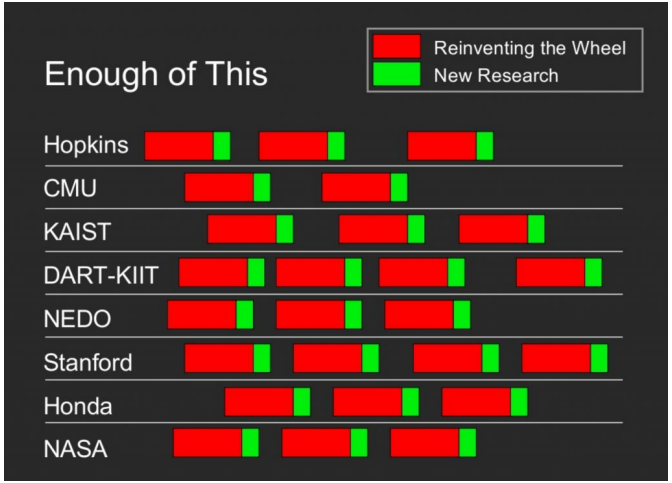
The Software required to build a mobile robot



ROS 2 - What, Why, How?

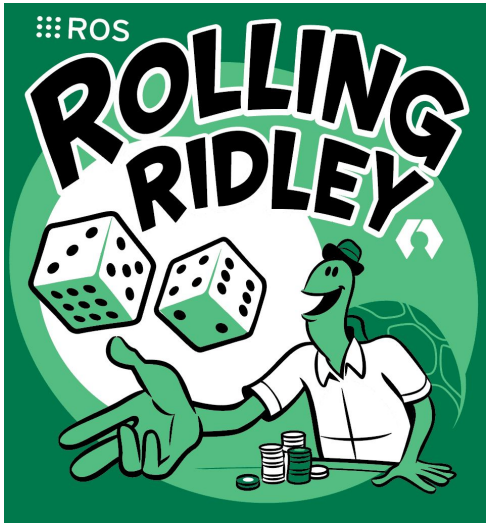
A brief history of ROS

- The ROS initiative started at Stanford University in 2006 with Keenan Wyrobek and Erik Berger
- It came as a response to the big issue in robotics research that everyone was reinventing the wheel, and thus not focusing on actual research
- After Stanford, ROS was developed until 2014 by Willow Garage
- In 2014, it was taken over by OSRF (Open Source Robotics Foundation)
- The last ROS 1 release (Noetic Ninjemys) was released in 2020
- ROS 2 was released in 2017, as an answer to the need of the industry to have a suitable version of ROS
- ROS 1 will reach EOL in 2025, and after that only ROS 2 should be used for new developments



What is ROS?

- ROS (Robot Operating System) is a framework developed to make robotics development easier
- It is a set of libraries, tools and packages
  - Mostly open source and free to use



ROS 2 - overview

- Latest LTS (Long-term support) version: Jazzy Jalisco
- Supported operating systems: **Linux**, Windows, Mac
- Client libraries: rclcpp (C++), rclpy (Python). There are also community maintained ones (Java, Rust)

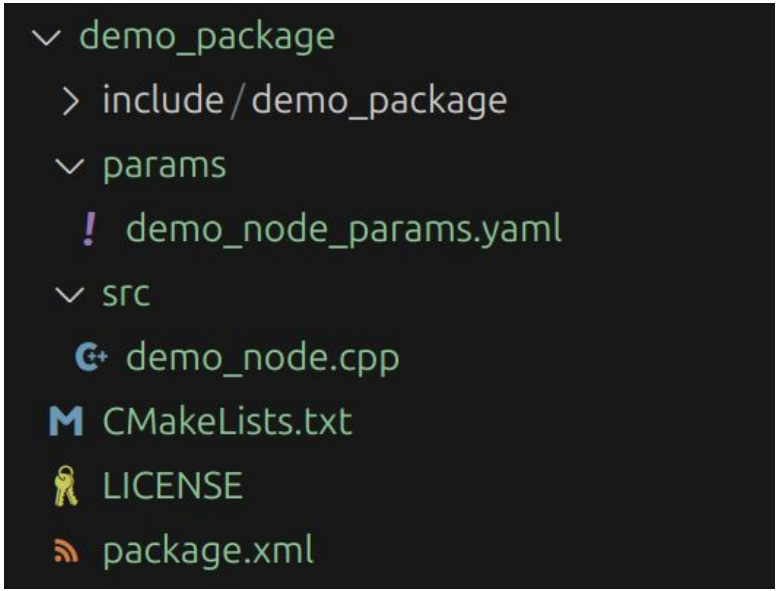
Distro	Release date	Logo	EOL date
Jazzy Jalisco	May 23rd, 2024		May 2029
Iron Irwini	May 23rd, 2023		November 2024
Humble Hawksbill	May 23rd, 2022		May 2027

Why ROS? - main advantages

- Standards and conventions
- Sensors and hardware are plug-and-play, as long as the manufacturers provide ROS drivers
- A lot of already existing packages that do a wide range of things required to start working with a robot - no need to reinvent the wheel!
- Distributed system - works on any computer and allows computers in the same network to talk to each other
- Actively developed by big companies (Nvidia, Sony, Canonical) that are backing up a community of experienced developers
- Active open source community
  - ROS Discourse
  - Robotics Stack Exchange
- It has slowly managed to become the standard framework to use for custom robotics software development

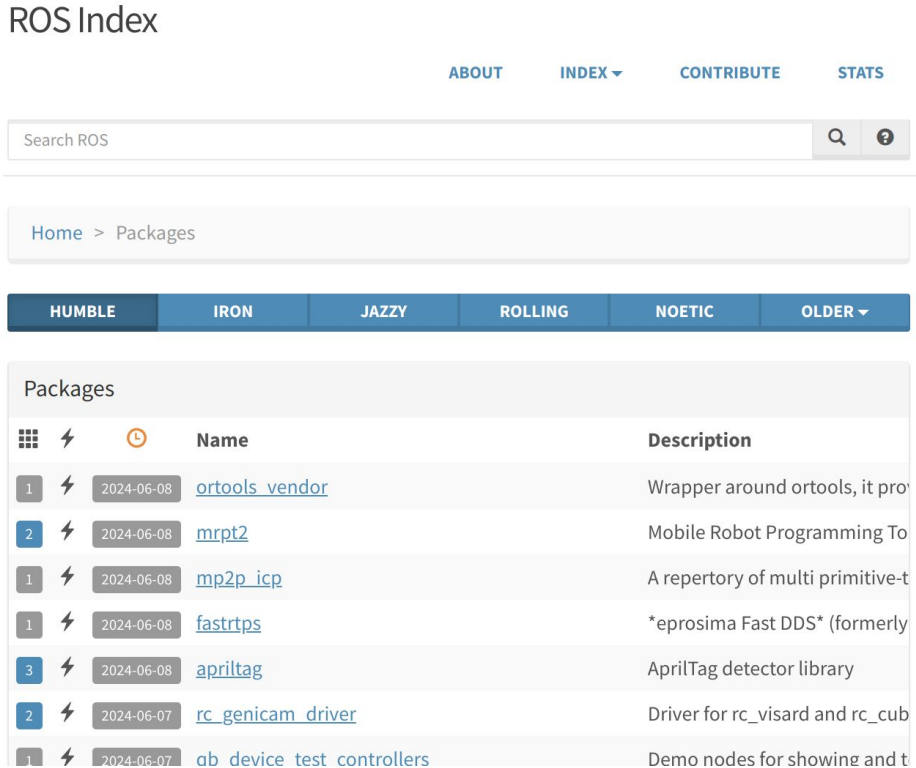
ROS packages - what are they?

- Contains all the code for one specific feature (teleoperation, etc.)
- What it includes
  - Code
  - Parameters
  - Configuration files (dependencies, build instructions)
- Packages make it easy to install the code and share it with others
- To run already existing packages, you only should need to create a new parameter file and pass it for the package when running the code



ROS packages - where do we find them?

- Packages can be found in
  - ROS Index
  - Github
- New package announcements are usually sent in
  - ROS Discourse
  - LinkedIn
- Package documentation is on GitHub or on the package's own website



ROS packages - how do we install them?

- Apt
- Snap
- **Docker**

```
$ sudo apt install ros-humble-slam-toolbox
```

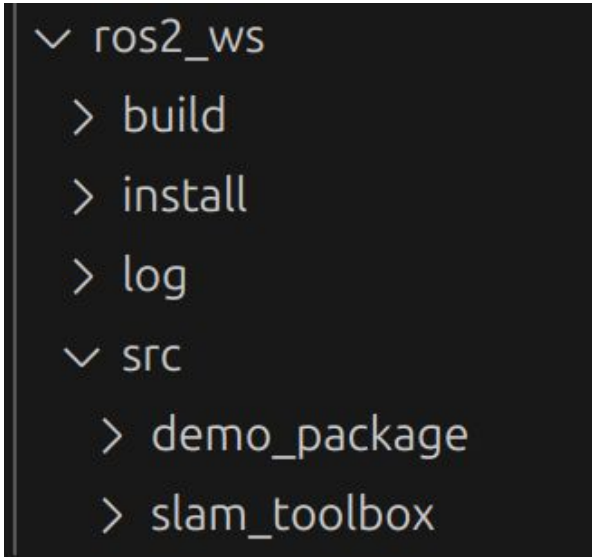
```
$ sudo snap install slam-toolbox --edge
```

```
$ docker pull husarion/slam-toolbox:humble
```

- Build from source code 🙊

ROS workspaces - what is a workspace?

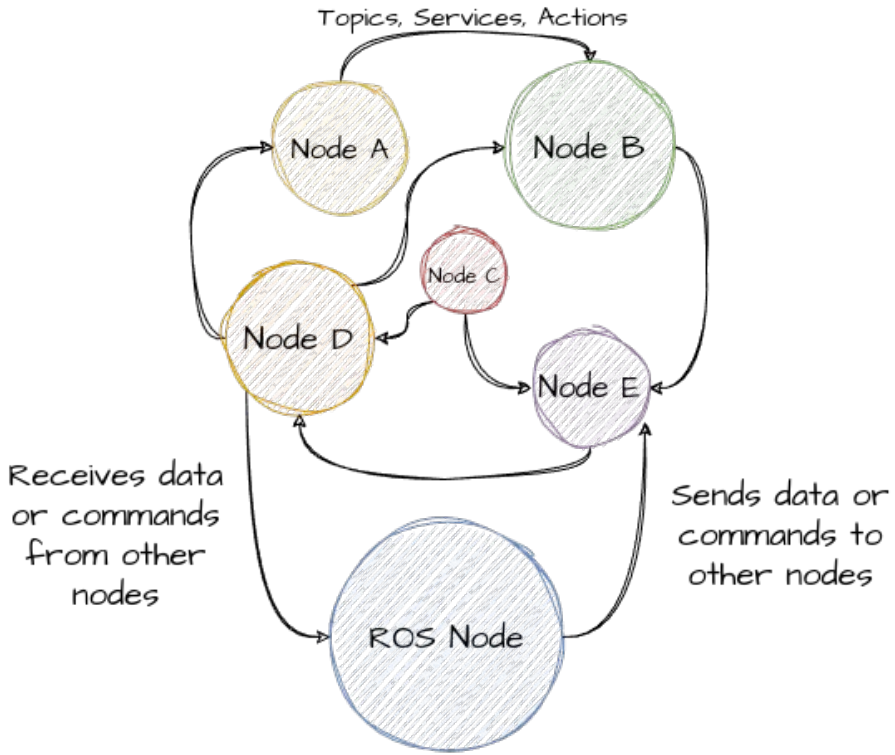
- Directory that contains all the user implemented packages and the packages built from source
- Building an existing package from source
  - Create a workspace (If doesn't exist yet)
  - Clone the package from git
  - Build
  - source




```
mkdir ros2_ws
cd ros2_ws
git clone https://github.com/SteveMacenski/slam_toolbox.git src
colcon build --symlink-install
source install/setup.bash
```

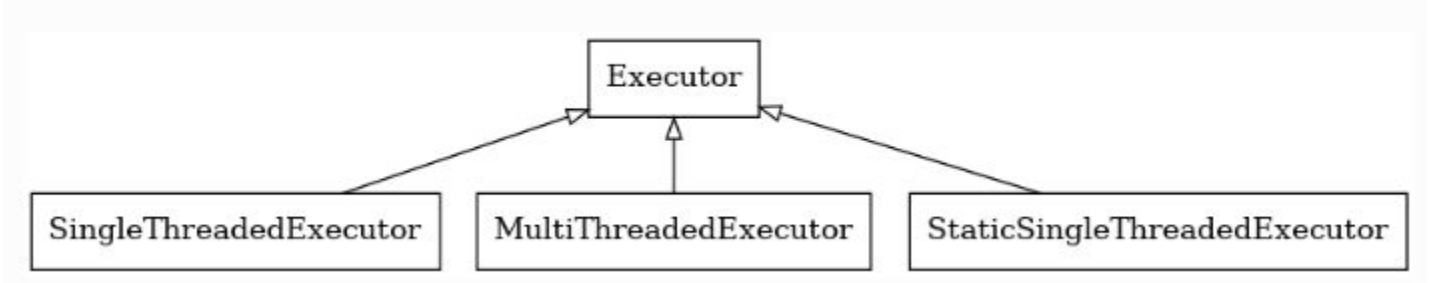
Nodes - what is a node?

- It is the smallest unit of processing in the ROS ecosystem
- It communicates with other nodes in the ROS network by using Topics, Services and Actions
- It contains code that should be responsible only for one thing:
  - Receive data from a sensor
  - Control a motor
  - Calculate a new navigation path



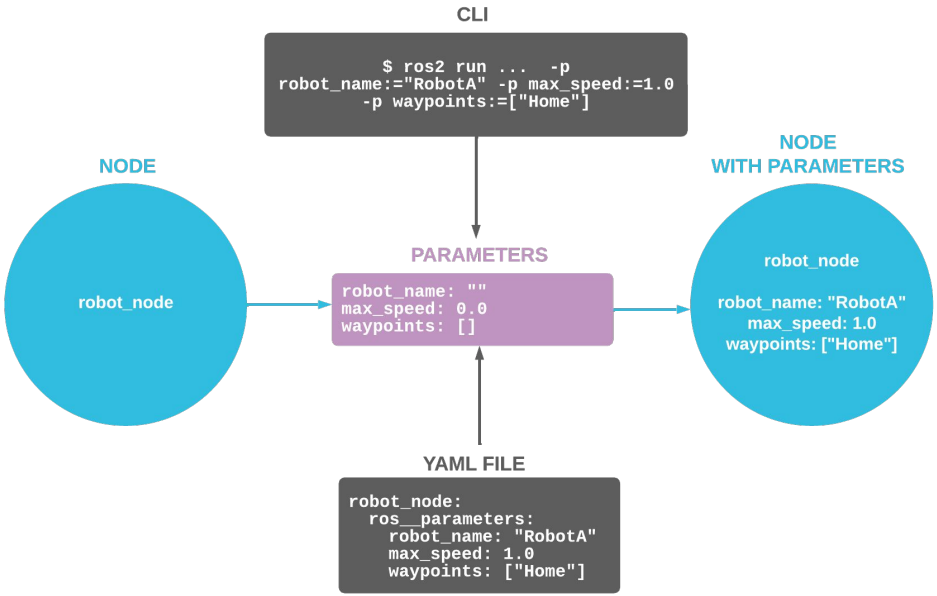
Nodes - Executors. Single-threaded. Multi-threaded.

- The code that sits behind a node needs to spin (execute). This is needed in order to wait for messages and send messages to other nodes.
- The spinning leads to the execution of our code to happen in-order and because of this, if we stay in one part of the code for a longer time, then everything else is also blocked and we cannot receive messages or do any other kind of processing. That is why we need multithreaded executors.



Nodes - parameters

- Parameters can be defined for each node
- They allow the node to run in a specific way, and are usually specific per robot, or sensor.
- These need to be changed between robots and are usually kept in a file that sits in the same package as the node that uses them.



Nodes - launch files

- Ros nodes can be executed in 2 ways: either ran or launched.
- If we want to launch a node, then we need a launch file
- A launch file can be used to launch the node and all the other things required for the node to run properly.
- The node specific parameters can be imported and specified in the launch file.
- Multiple nodes can be launched at once in a launch file, for example to start a complex feature for the robot (navigation, mapping, etc.)
- Python ROS 2 launch file example: https://github.com/ros-navigation/navigation2/blob/main/nav2_bringup/launch/tb3_simulation_launch.py

Topics - what are they?

- They are the main and most simple way in which nodes communicate with each other.
- They work based on the Publisher/Subscriber model
- They are the data exchange medium between publishing and subscribing nodes

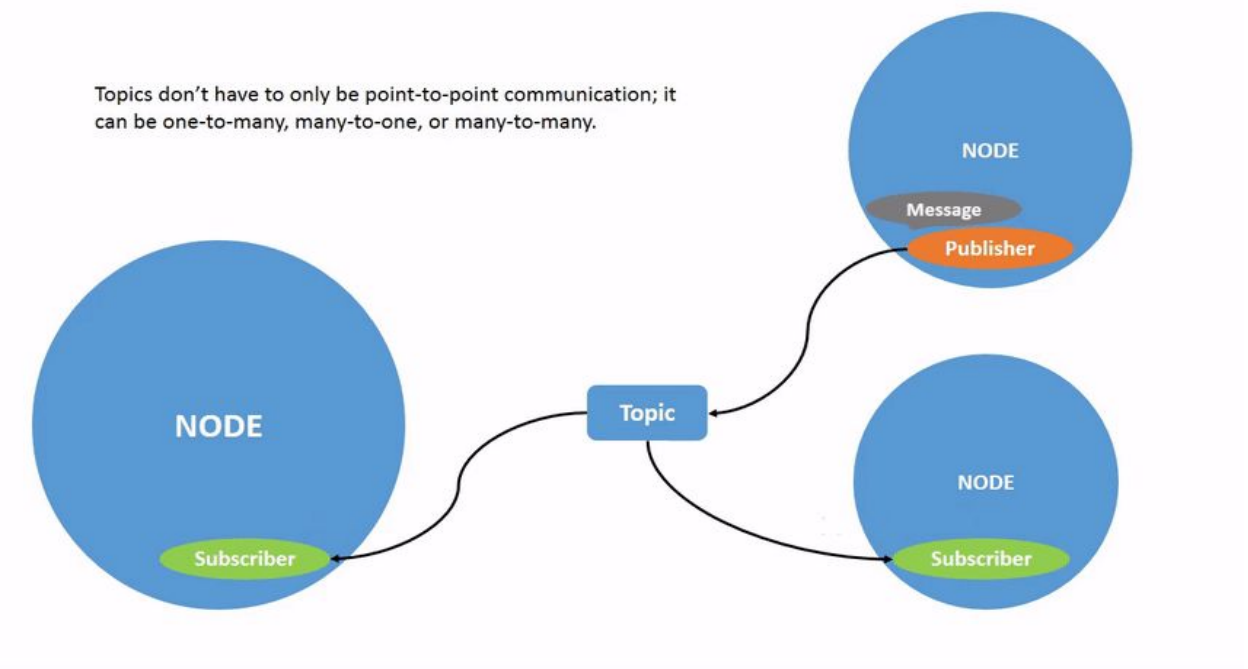


Image source: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

Topics - messages & types

- Topics allow the flow of complex data types through them
- ROS messages are the ones that define what is the data type that can be sent over a topic
- Example: https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

geometry_msgs/Vector3 Message

File: geometry_msgs/Vector3.msg

Raw Message Definition

```
# This represents a vector in free space.
# It is only meant to represent a direction. Therefore, it does not
# make sense to apply a translation to it (e.g., when applying a
# generic rigid transformation to a Vector3, tf2 will only apply the
# rotation). If you want your data to be translatable too, use the
# geometry_msgs/Point message instead.

float64 x
float64 y
float64 z
```

A quick demo with a simulated robot

```
ros2 topic pub -r 1 /turtle1/cmd_vel geometry_msgs/msg/Twist
"{linear: {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0,
z: 0.5}}"
```

Services - what are they?

- Through services, the nodes are exposing their **instantaneous functionalities, e.g:**
  - Turn on an LED.
  - Take a photo from one of the robot's cameras.
- Internally built with topics
- They work based on the **Call-and-response model**

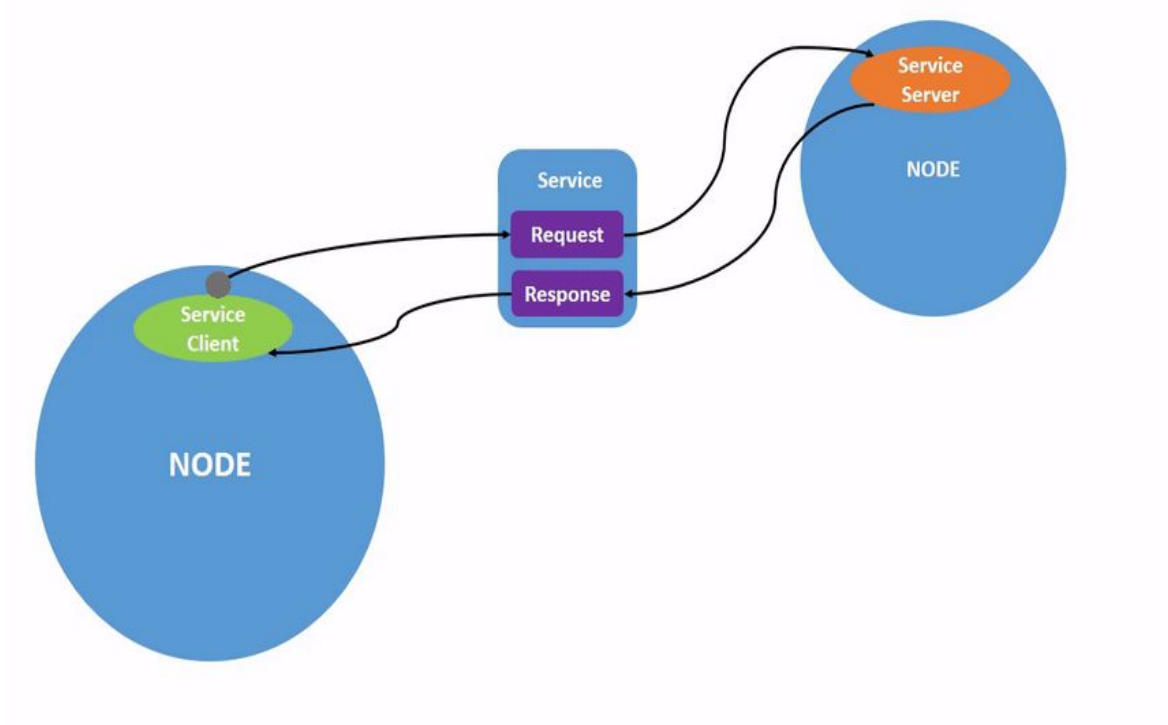


Image source:
https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

Actions - putting everything together

- Through actions, the nodes are exposing their **continuous functionalities, e.g:**
  - Map the room.
  - Autonomously navigate to a new location.
- For each new goal, actions provide feedback from the beginning of the goal and until the end.
- Internally built by combining **topics** and **services**.

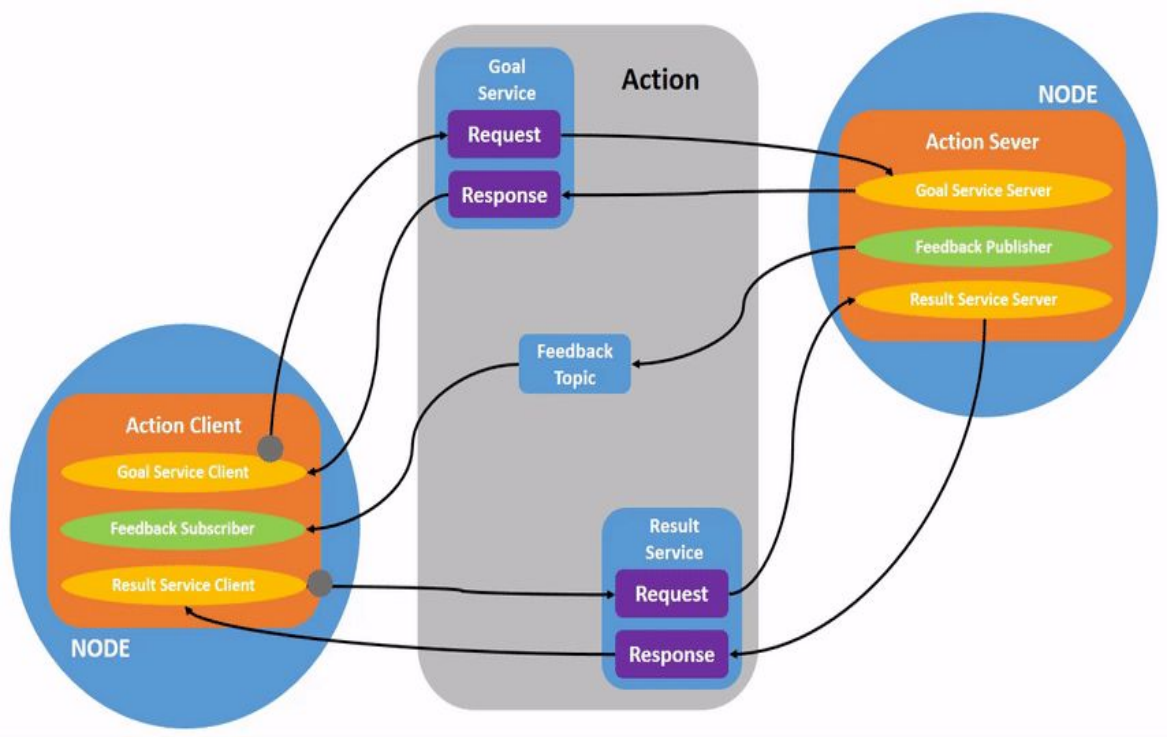


Image source:
https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html

Clock and time-synchronization in ROS 2

- ROS 2 supports simulated time
  - Can progress slower, faster, backwards, and can be paused
  - Every node has a “use_sim_time” parameter
  - Simulated starts from 0, real time is reported as UNIX time: 1718045230 (10.06.2024, 21.47)
- Most of the messages are stamped with the time
  - Important because the messages might not be received in order (asynchronous communication)
- Since ROS is a distributed system, all the computers and sensors should be time synchronized

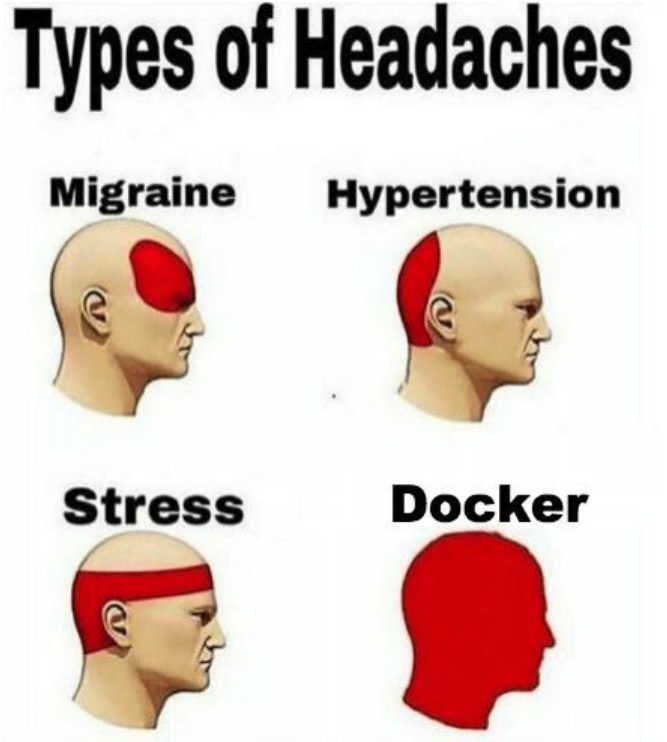
Middleware & DDS

- All the communication between Nodes happens using Middleware
- ROS 2 has 2 main middleware implementations that are based on DDS (Data Distribution Service)
  - Fast DDS
  - Cyclone DDS
- With the default settings, they do not work well over WiFi
- Hard work is currently done by OSRF and ZettaScale to integrate a new middleware, rmw_zenoh, that will hopefully mitigate lots of the issues with DDS

Possible drawbacks of using ROS

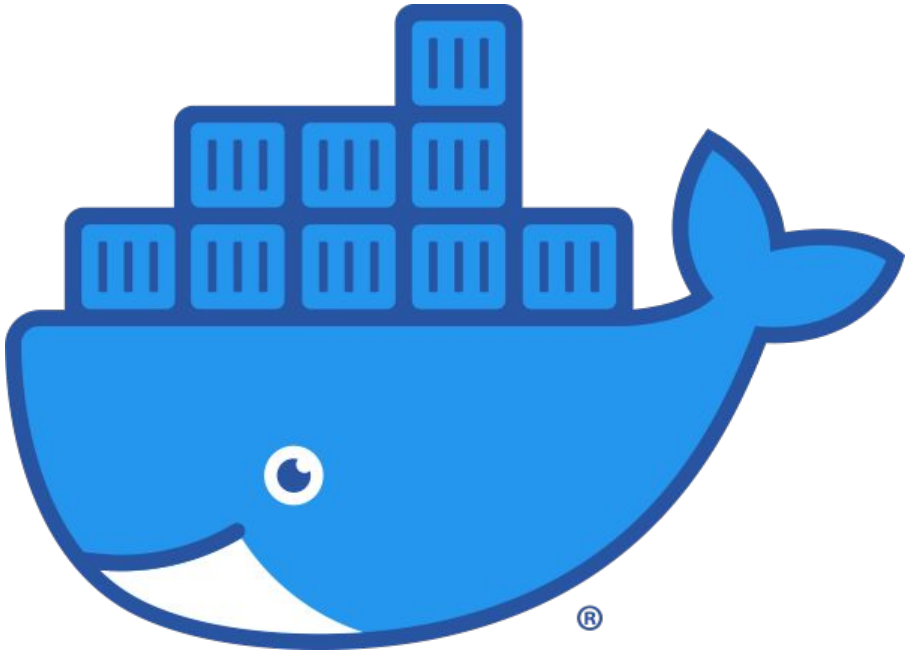
- Open source -> code is sometimes not production-grade -> it often is unstable with: Bugs, issues, etc.
- Complex, high learning curve
- Lacking documentation: Not enough and out-of-date documentation
- Difficult installation. Easy to mess up the whole system -> Use Docker 💡

Docker - A short introduction



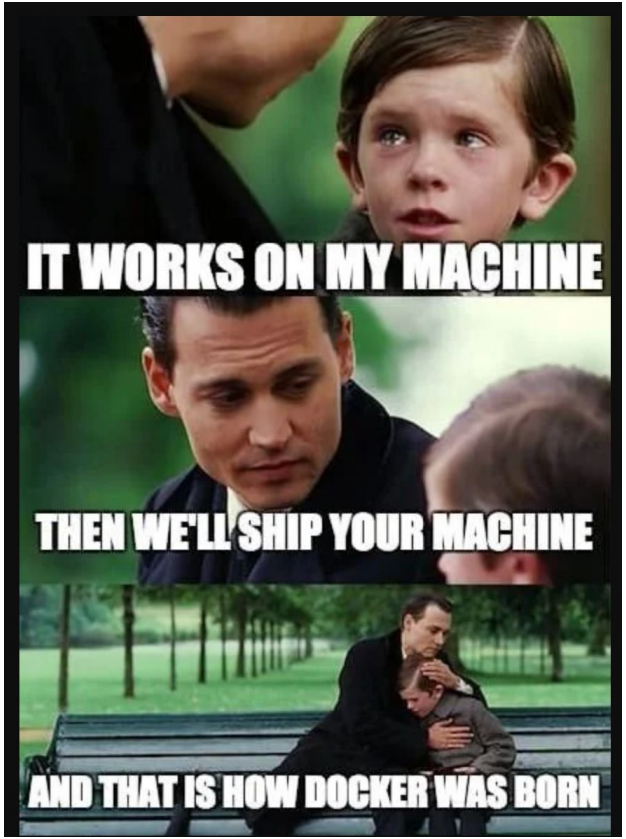
What is Docker?

- Open platform for developing, shipping and running applications
- Provides the ability to package and run an application in an isolated environment called a **container**
- You can think of containers as virtual machines running on top of your OS, and containing all the Software dependencies required by your application



Why would a roboticist want to use Docker?

- Two robots that seem to be the same, are **almost always not the same..**
  - Different versions of the same hardware devices.
  - Other small differences in hardware (motor tightenings, sensor positions, etc)
  - Different software package versions
- Broken dependencies
- Inconsistent versioning in releases
- Many devices with small differences between them to manage (fleets of robots)



Pros and Cons of using Docker

Pros:

- No broken dependencies
- Full control on what software package versions you are using!
- You can build a container, and that same container can be sent to any other computer. Nothing else needs to be installed in order to ship your code!
- Lowers the human-error chance!
- You can try all kinds of risky stuff and mess up any installation inside a container! If you break anything, just start a new container and you are good to go!
    - WARNING: you can still mess stuff up on your own machine if your container has too many rights (mounting critical folders, sudo?)

Cons:

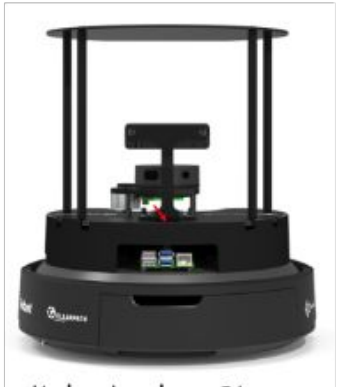
- Hard to master
- Not optimal for applications with complex GUIs
- Working with multiple containers is hard! Solid software architecture knowledge is needed!
- Containers can become hard to manage if we add too much stuff to them!

Some of the most popular robots running on ROS 2 in 2024

SPOT by Boston Dynamics

ANYmal by ANYbotics

Unitree GO1 by Unitree



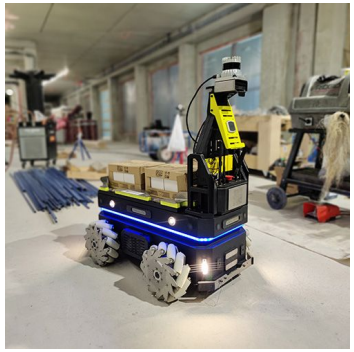
Turtlebot4 by Clearpath Robotics

Industrial use-cases for ROS-powered robots

Site monitoring and data gathering

Warehouse management & automation

Agriculture



Construction

Medicine

Image sources: Karelics, Dexory, Organifarms, Vexev

Cheatsheets & resources

Cheatsheets

- ROS 2
- Docker - https://docs.docker.com/get-started/docker_cheatsheet.pdf
- Linux - https://www.guru99.com/linux-commands-cheat-sheet.html
- There will also be a cheat sheet available with the exercises which we will present on Friday.

ROS 2 resources

ROS 2 Humble official tutorials - https://docs.ros.org/en/humble/Tutorials.html

ROS Index - https://index.ros.org/packages/

ROS Discourse - https://discourse.ros.org/latest

ROS courses (many are free!) - https://app.theconstruct.ai/

Simulated ROSBOT mapping example with Docker

Simulated ROSBOT navigation example with Docker

An interesting overview paper about ROS 2