

DTU

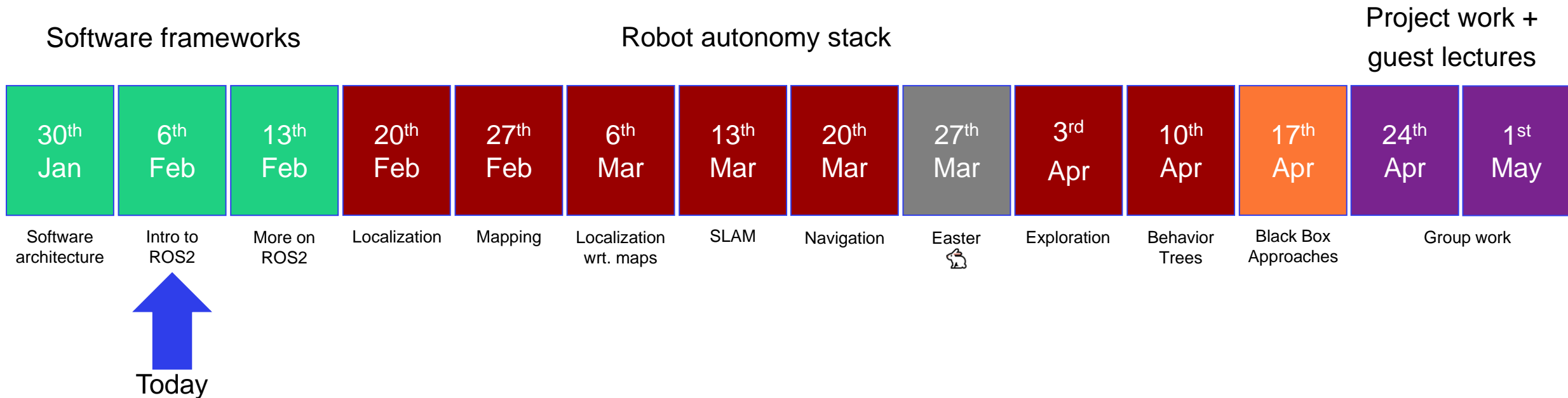


Rasmus Eckholdt Andersen
34761 – Robot Autonomy

Introduction to ROS2

Overview of 34761 – Robot Autonomy

- 3 lectures on software frameworks
- 7 lectures on building your own autonomy stack for a mobile robot
- 1 lecture on DL/RL – an overview of black-box approaches to what you have done
- 2 lectures of project work before hand in + guest lectures



Recalling from last lecture..

- What did we talk about?
 - Which components are necessary to have robot autonomy?
 - Robot localization
 - Mapping
 - Navigation
 - Exploration
 - Behavior definitions
 - NNE/DTU RoboTech Challenge
 - Operation paradigms and software architectures
 - Hardware abstraction / middleware



Calling all DTU students!
Come and test your skills in NNE's robot competition!

Interested in robotics, automation and the future of work? Do you have an interest in pharmaceutical manufacturing? Then join the **DTU RoboTech Challenge** hosted by the Fill Finish Automation team at NNE.

The event will take place on **1st of March 2024**.

Sign-up is open from
JANUARY 30th till FEBRUARY 7th.

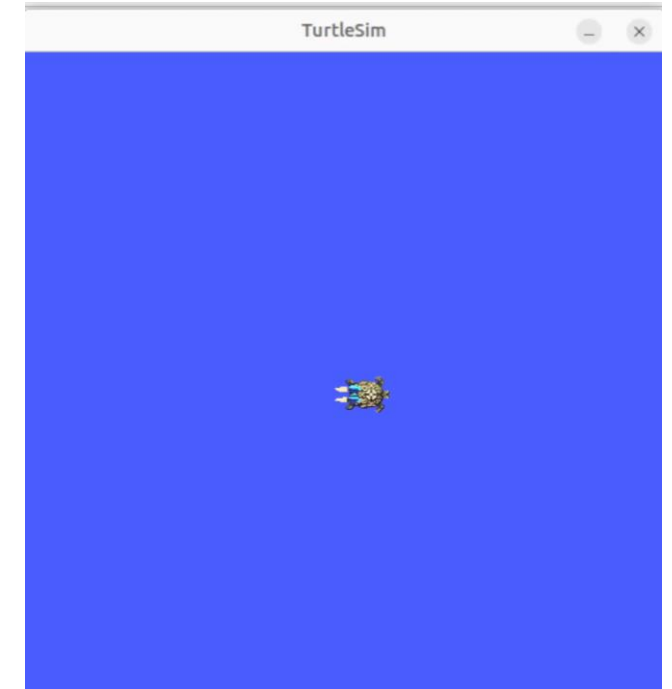


Outline

- The ROS2 framework
 - Nodes
 - Topics
 - Services
 - Actions
- ROS2 tools
- A simple turtle simulation

TODAY

- ROS2 workspace
- ROS2 packages
 - Simple publisher and subscriber
- A turtlebot simulation in Gazebo
 - Your environment for the remainder of the course



Hardware Abstraction Layers #2

- A class of technologies in order to handle the complexity of distributed systems

● ● ● ROS 2TM
● ● ●
● ● ●

User

Human Machine Interaction
(HMI)

Application

Abstract Task Planning

Middleware

Operating System

Thread Scheduling
Real Time?

Hardware

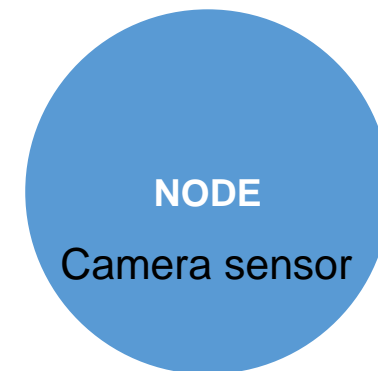
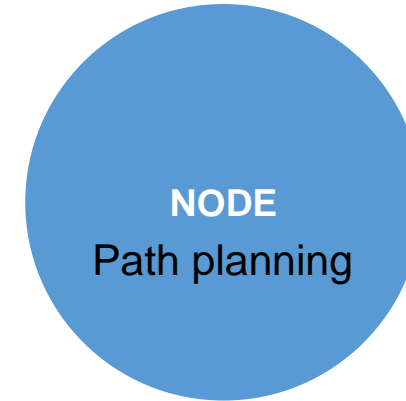
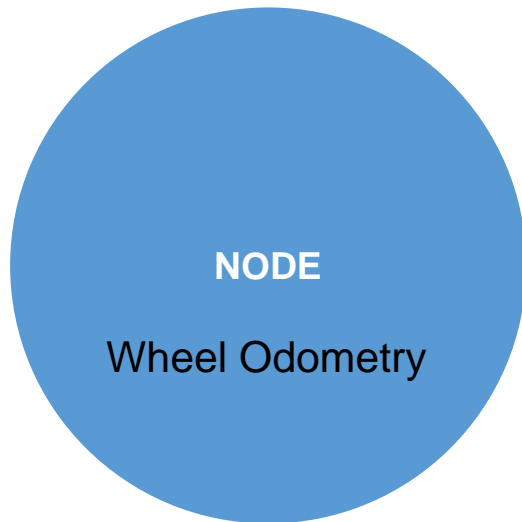
Sensors
Actuators

ROS2 NODES

- **Node in ROS2:**
 - Participant in the ROS2 graph using a client library for communication
 - Executes computations, communicating within the same/different process or on a different machine
 - Are typically anonymous – mostly identified by the topics they publish/subscribe to
- **Communication Abilities:**
 - Publishes to named topics, subscribes to receive data, acts as service client/server, or as an action client/server
 - Nodes can provide configurable parameters for dynamic behavior changes
- **Node Complexity and Connections:**
 - Nodes often combine roles like publishers, subscribers, service servers, service clients, action servers, and action clients
 - Connections between nodes are established through a distributed discovery process

ROS2 NODES

- **ROS2 Nodes**
- ROS2 Topics
- ROS2 Services
- ROS2 Actions



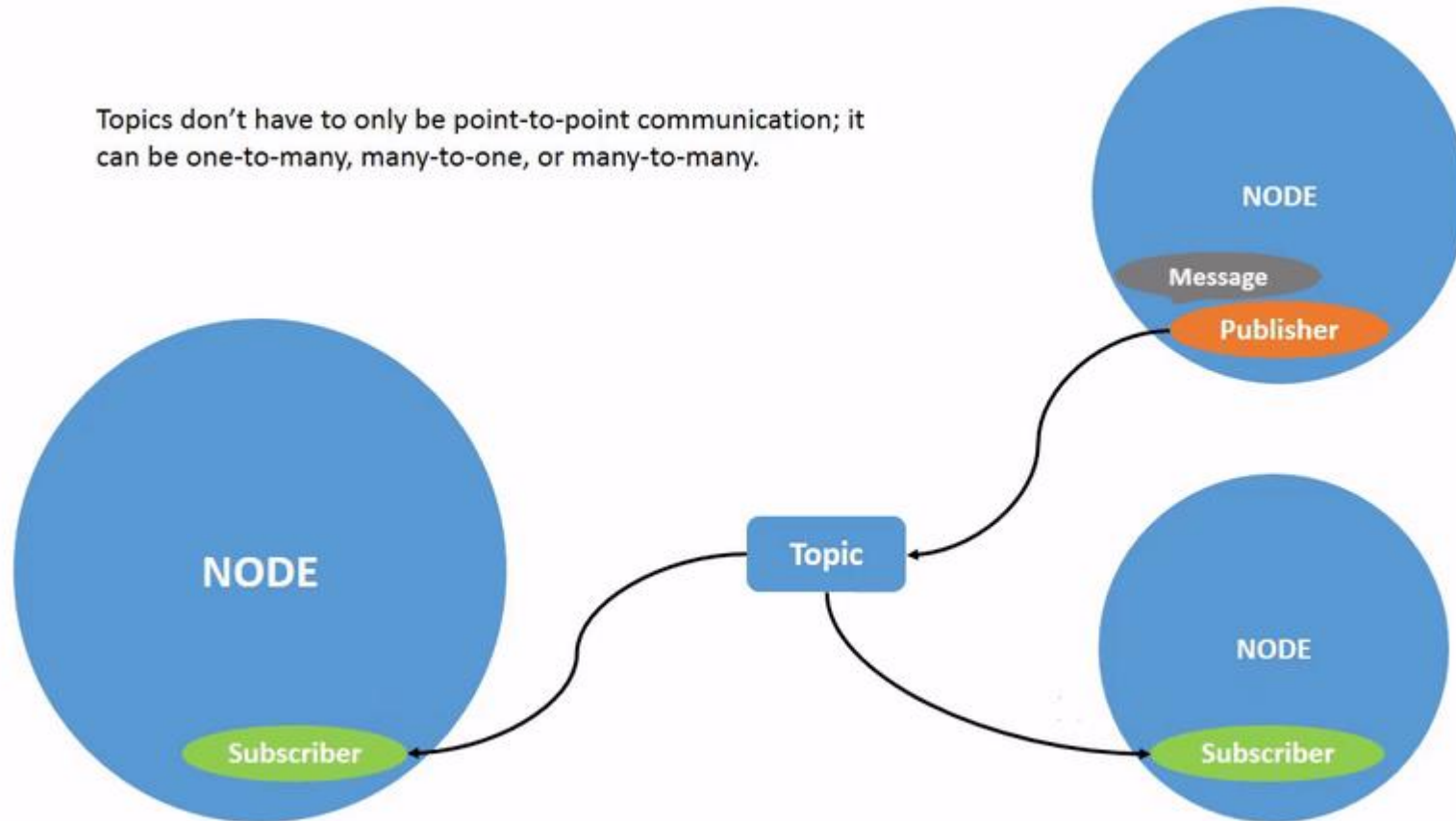
ROS2 TOPICS

- Should be used for continuous data streams (sensor data, robot state, ...).
- Are for continuous data flow. Data might be published and subscribed at any time independent of any senders/receivers
- Many to many connection. Callbacks receive data once it is available. The publisher decides when data is sent.
- **Analogy:** Me giving you this lecture
 - I'm the only one speaking (publishing) and all of you are listening (subscribed)
 - You can subscribe to me speaking by staying in the room or you can unsubscribe by leaving the room

ROS2 TOPICS

- ROS2 Nodes
- **ROS2 Topics**
- ROS2 Services
- ROS2 Actions

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



<https://docs.ros.org/>

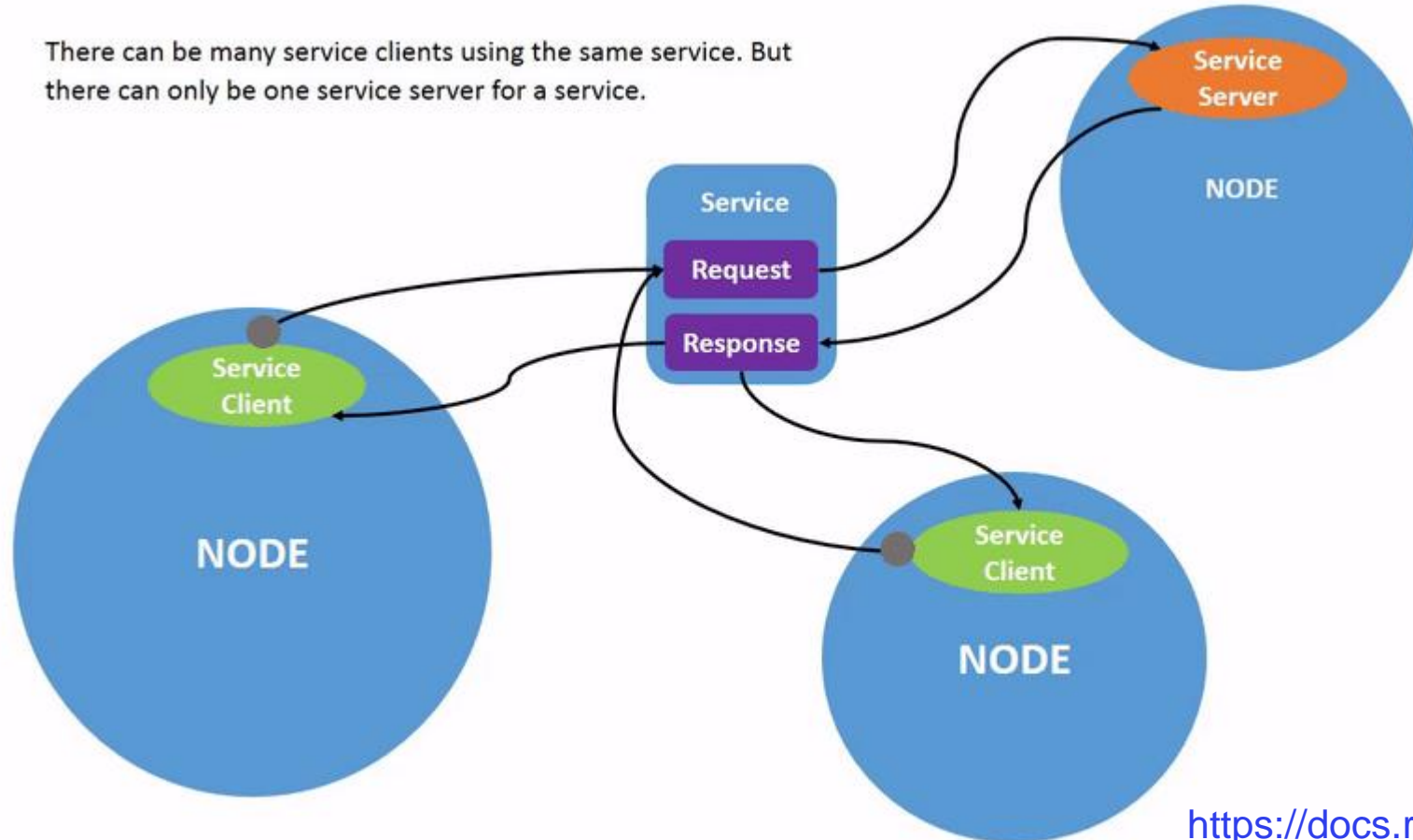
ROS2 SERVICES

- Semantically similar to function calls
 - Blocks the caller, so services should return quickly
 - Used for changing the state of a node (flipping a switch) or fast computations (like IK)
- Don't use for processes that might require to exit preemptively if exceptional situations occur
 - If the service takes too long to process, the state might change, and we can't react to it!
- **Analogy:** Me asking you a question
 - I ask (request) you to answer a question -> "which room are you in?"
 - You answer (respond) -> "Room XX"
 - Taking too long to respond might mean you no longer are in room XX by the time I hear about it

ROS2 SERVICES

- ROS2 Nodes
- ROS2 Topics
- **ROS2 Services**
- ROS2 Actions

There can be many service clients using the same service. But there can only be one service server for a service.



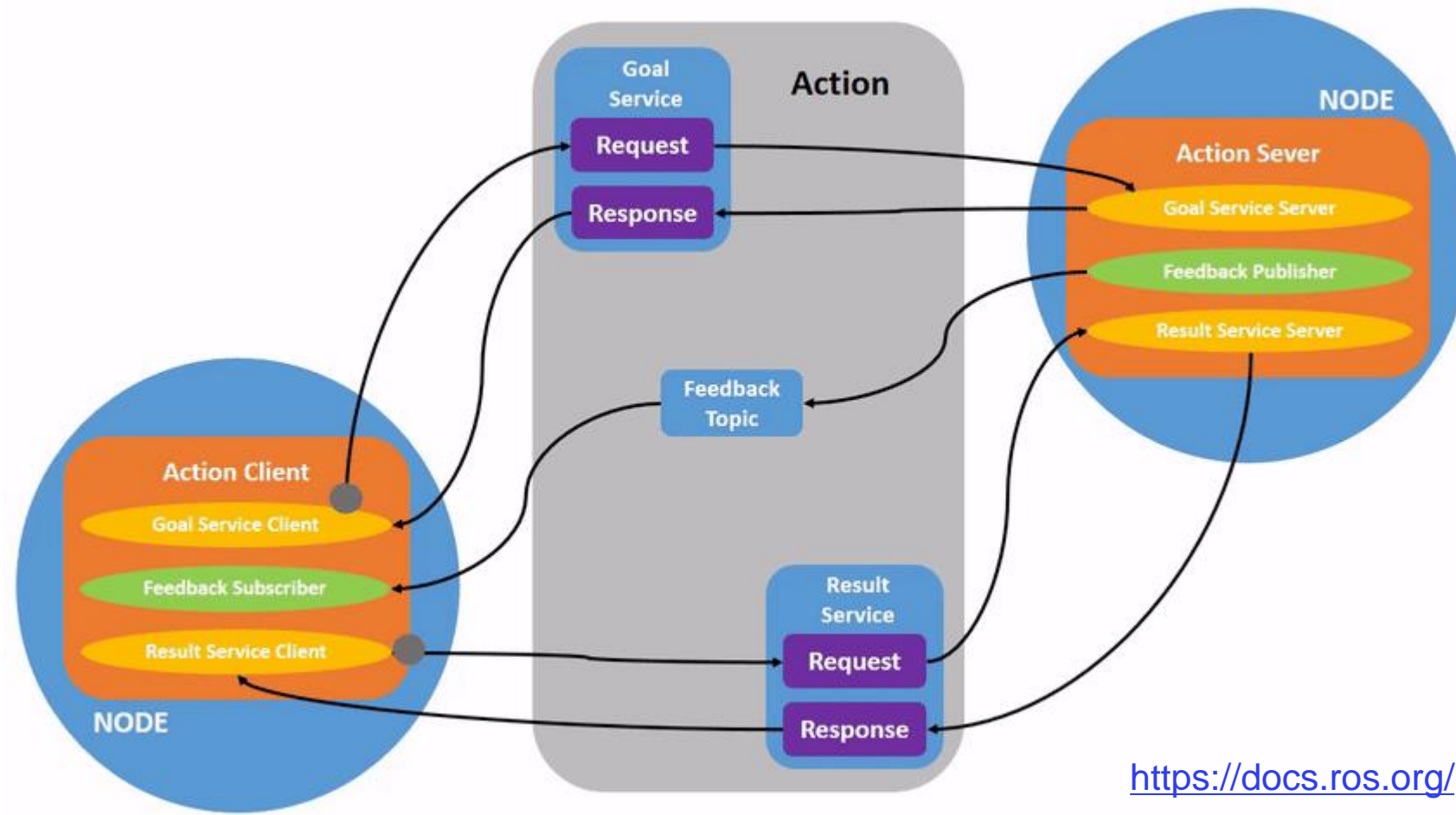
<https://docs.ros.org/>

ROS2 ACTIONS

- Should be used for any discrete behavior that moves a robot or that runs for a longer time
- Provides feedback during execution
- The most important property of actions is that they can be preempted and preemption should always be implemented cleanly by action servers.
- Slow perception routines which take several seconds to terminate or initiating a lower-level control mode are good use cases for actions
- **Analogy:** Me asking you to do an exercise
 - I ask you to raise your left hand
 - You continuously update me on the progress of you raising your hand
 - You tell me when you finished raising your hand

ROS2 ACTIONS

- ROS2 Nodes
- ROS2 Topics
- ROS2 Services
- **ROS2 Actions**



<https://docs.ros.org/>

Message structures

- There are pre-defined topic/service/action messages for most common applications
 - A message type is defined in a message file with the extension *.msg, *.srv, *.action
 - standard messages: https://docs.ros.org/en/melodic/api/std_msgs/html/index-msg.html
 - geometry messages: https://docs.ros.org/en/melodic/api/geometry_msgs/html/index-msg.html
 - Sensor messages: https://docs.ros2.org/latest/api/sensor_msgs/index-msg.html
 - And more ...
 - Structure of the file depends on if it's a topic message, a service message, or action message
 - So lets have a look at a simple topic message...

Topic message structures

File: `std_msgs/msg/ColorRGBA.msg`

Raw Message Definition

```
float32 r  
float32 g  
float32 b  
float32 a
```

- https://docs.ros2.org/foxy/api/std_msgs/msg/ColorRGBA.html

Topic message struct

File: `std_msgs/msg/ColorRGBA`

Raw Message Definition

```
float32 r
float32 g
float32 b
float32 a
```

Type name	C++	Python	DDS type
bool	bool	builtins.bool	boolean
byte	uint8_t	builtins.bytes*	octet
char	char	builtins.str*	char
float32	float	builtins.float*	float
float64	double	builtins.float*	double
int8	int8_t	builtins.int*	octet
uint8	uint8_t	builtins.int*	octet
int16	int16_t	builtins.int*	short
uint16	uint16_t	builtins.int*	unsigned short
int32	int32_t	builtins.int*	long
uint32	uint32_t	builtins.int*	unsigned long
int64	int64_t	builtins.int*	long long
uint64	uint64_t	builtins.int*	unsigned long long
string	std::string	builtins.str	string
wstring	std::u16string	builtins.str	wstring

Message structures

- There are pre-defined topic/service/action messages for most common applications
 - A message type is defined in a message file with the extension *.msg, *.srv, *.action
 - standard messages: https://docs.ros.org/en/melodic/api/std_msgs/html/index-msg.html
 - geometry messages: https://docs.ros.org/en/melodic/api/geometry_msgs/html/index-msg.html
 - Sensor messages: https://docs.ros2.org/latest/api/sensor_msgs/index-msg.html
 - And more ...
 - Structure of the file depends on if it's a topic message, a service message, or action message
 - So lets have a look at a simple topic message...
 - And a simple service message...

Service message structures

File: `std_srvs/srv/SetBool.msg`

Raw Message Definition

```
bool data # e.g. for hardware enabling / disabling  
  
---  
bool success # indicate successful run of triggered service  
  
string message # informational, e.g. for error messages
```

- https://docs.ros2.org/foxy/api/std_srvs/srv/SetBool.html

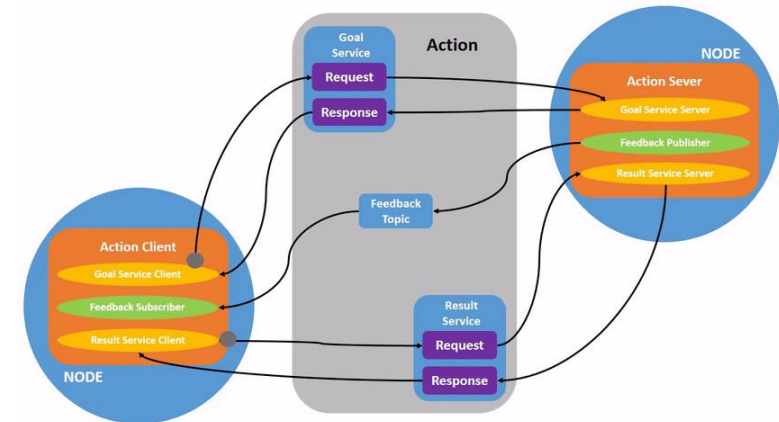
Message structures

- There are pre-defined topic/service/action messages for most common applications
 - A message type is defined in a message file with the extension *.msg, *.srv, *.action
 - standard messages: https://docs.ros.org/en/melodic/api/std_msgs/html/index-msg.html
 - geometry messages: https://docs.ros.org/en/melodic/api/geometry_msgs/html/index-msg.html
 - Sensor messages: https://docs.ros2.org/latest/api/sensor_msgs/index-msg.html
 - And more ...
 - Structure of the file depends on if it's a topic message, service message, or action message
 - So lets have a look at a simple topic message...
 - And a simple service message...
 - And a simple action message...

Action message structures

- The message structure is similar to a concatenation of a service and a topic message
 - A goal (like the request part of a service)
 - A result (like the response part of a service)
 - Intermediate results (like a topic stream of intermediate results)

```
1 int32 order
2 ---
3 int32[] sequence
4 ---
5 int32[] partial_sequence
6 /
```



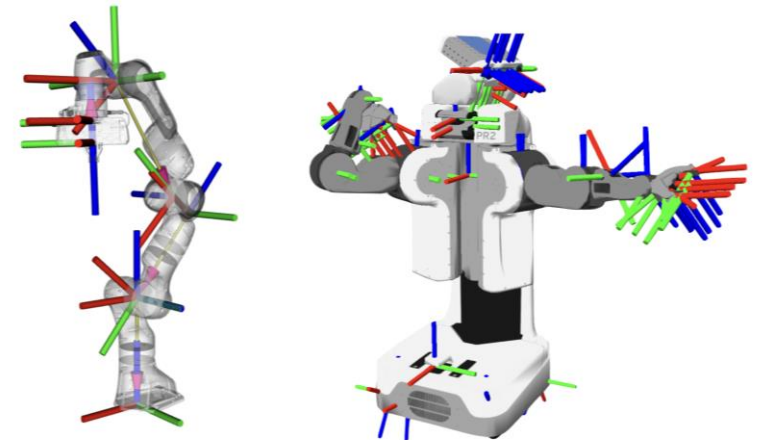
- Remember that the action server can accept/reject your action request (so the above analogy only holds for the message definition and not implementation)

Message structures

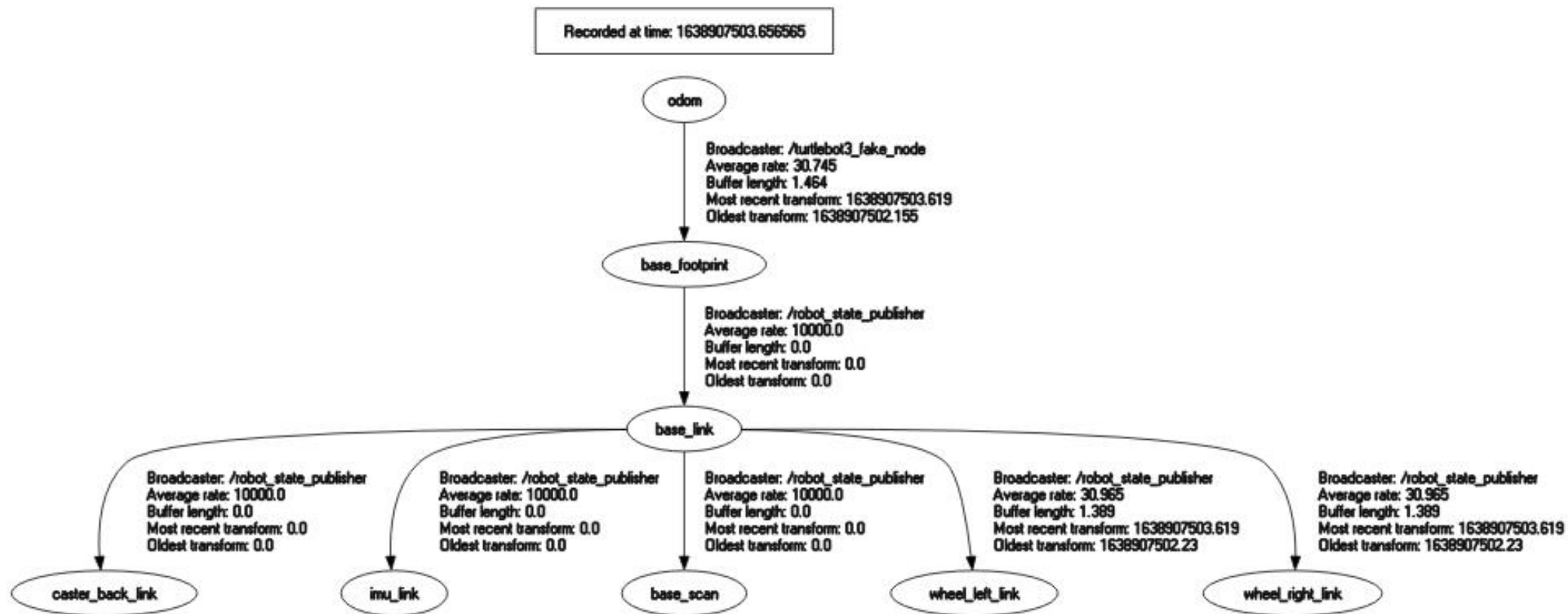
- There are pre-defined topic/service messages for most common applications
 - A message type is defined in a message file with the extension *.msg, *.srv, *.action
 - standard messages: https://docs.ros.org/en/melodic/api/std_msgs/html/index-msg.html
 - geometry messages: https://docs.ros.org/en/melodic/api/geometry_msgs/html/index-msg.html
 - Sensor messages: https://docs.ros2.org/latest/api/sensor_msgs/index-msg.html
 - And more ...
 - Structure of the file depends on if it's a topic message, service message, or action message
 - So lets have a look at a simple topic message...
 - And a simple service message...
 - And a simple action message...
 - Messages can even be nested!!!
 - Include a header message in all other messages (see sensor_msgs/Image message)
 - It's possible to create our own message definitions as well, but **make sure there are no standard message that can be used for what you need before doing so!**
 - Exception: you will most likely have to create your own action definition ☹

Transforms

- Transforms/frames are intrinsic to knowing where things are at what time
- ROS2 (of course) has an implementation of transforms/frames we can/will use
- Can be either static or dynamic
 - Static frames are (often) published at fixed framerates and don't move w.r.t. their parent frame
 - Used for environment and fixed dimensions
 - Dynamic frames are published potentially irregularly and can move between w.r.t. their parent frame
 - Used for the parts of the robot that move (joints etc.)














Transforms



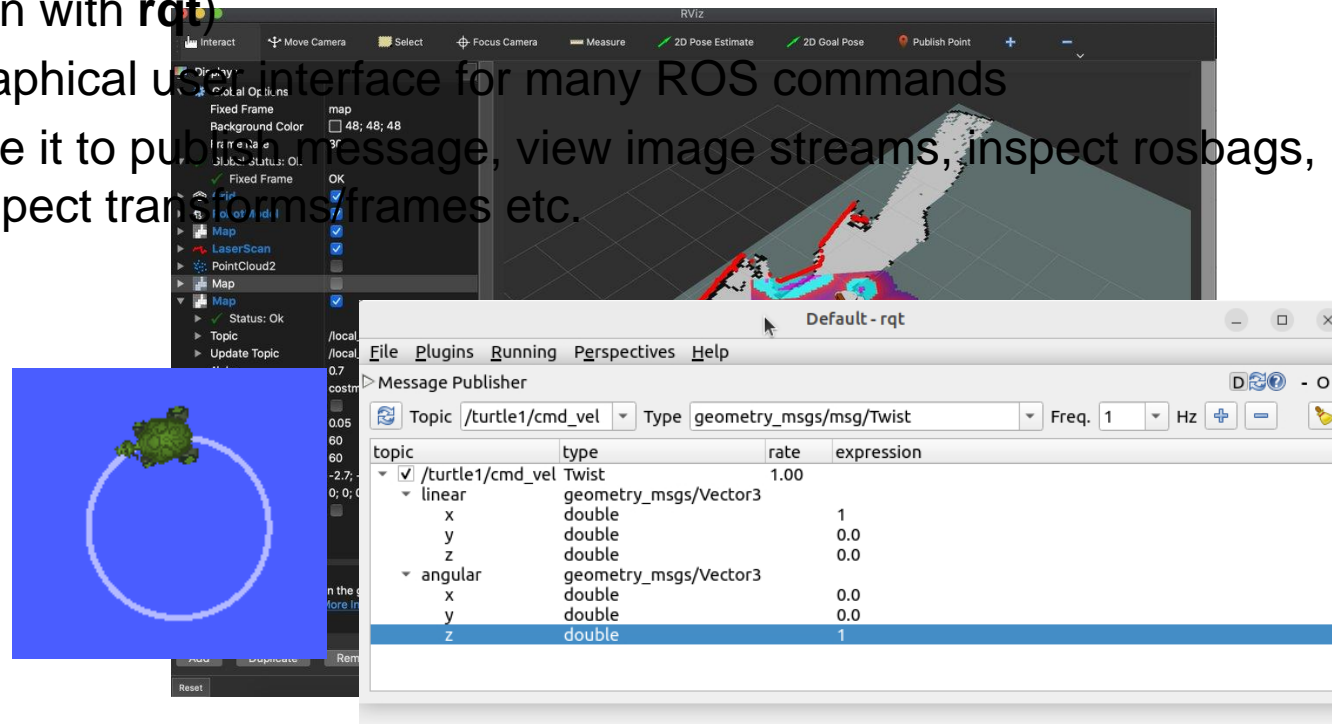
Tools - rosbags

- Often, it's useful to record the robot sensor data and replay it later
 - Extremely useful during development when you don't have access to the robot or environment
- The **ros2 bag** is a tool that does exactly this!
 - Record all topics that are being published into a *rosbag*
 - The bag can be played back, and messages will be published in “realtime”

 rosbags_2023-06-08-11-12-47.bag	91.33 GB	BAG File	06/08/2023 11:19:23
 rosbags_2023-06-08-11-25-07.bag	32.11 GB	BAG File	06/08/2023 11:27:20
 rosbags_2023-06-08-11-28-55.bag	25.5 GB	BAG File	06/08/2023 11:31:05
 rosbags_2023-06-08-11-31-40.bag	7.68 GB	BAG File	06/08/2023 11:32:42
 rosbags_2023-06-08-11-46-02.bag	11.44 GB	BAG File	11/07/2023 12:05:21
 rosbags_2023-06-08-12-12-15.bag	26.8 GB	BAG File	11/07/2023 15:35:50
 rosbags_2023-06-08-12-16-45.bag	18.5 GB	BAG File	06/08/2023 12:18:01
 rosbags_2023-06-08-12-18-18.bag	28.49 GB	BAG File	06/08/2023 12:20:33
 rosbags_2023-06-08-12-20-45.bag	16.61 GB	BAG File	06/08/2023 12:22:01
 rosbags_2023-06-08-12-27-01.bag	12.13 GB	BAG File	06/08/2023 12:27:51
 rosbags_2023-06-08-12-28-18.bag	22.49 GB	BAG File	06/08/2023 12:30:09

Other tools

- Rviz2
 - 3D Visualization tool
 - Visualize transforms/frames, robot models, sensor data (LiDAR etc.), odometry, paths, maps...
- Rqt (run with **rqt**)
 - A graphical user interface for many ROS commands
 - Use it to publish message, view image streams, inspect rosbags, inspect transforms/frames etc.



- nav2_rviz_plugins
 - ParticleCloud
- rviz_common
 - Group
- rviz_default_plugins
 - AccelStamped
 - Axes
 - Camera
 - DepthCloud
 - Effort
 - FluidPressure
 - Grid
 - GridCells
 - Illuminance
 - Image
 - InteractiveMarkers
 - LaserScan
 - Map
 - Marker
 - MarkerArray
 - Odometry
 - Path
 - PointCloud
 - PointCloud2
 - PointStamped
 - Polygon
 - Pose
 - PoseArray
 - PoseWithCovariance
 - Range
 - RelativeHumidity
 - RobotModel
 - TF
 - Temperature
 - TwistStamped
 - Wrench

Exercises

- Run turtlesim (hint: `ros2 run ...`)
 1. Kill the turtle already in the simulator
 2. Spawn a new turtle at position (6,3), remember to give it a unique name
 3. Control the turtle you spawned with the keyboards
 4. Record a rosbag of you moving the turtle around
 5. Replay the rosbag and see the turtle replicate the movement you recorded
 6. Use an action message to control the turtle