The F1/10 platform

Real-Time Embedded System - The F1tenth autonomous racing

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

High Performance Real Time Lab

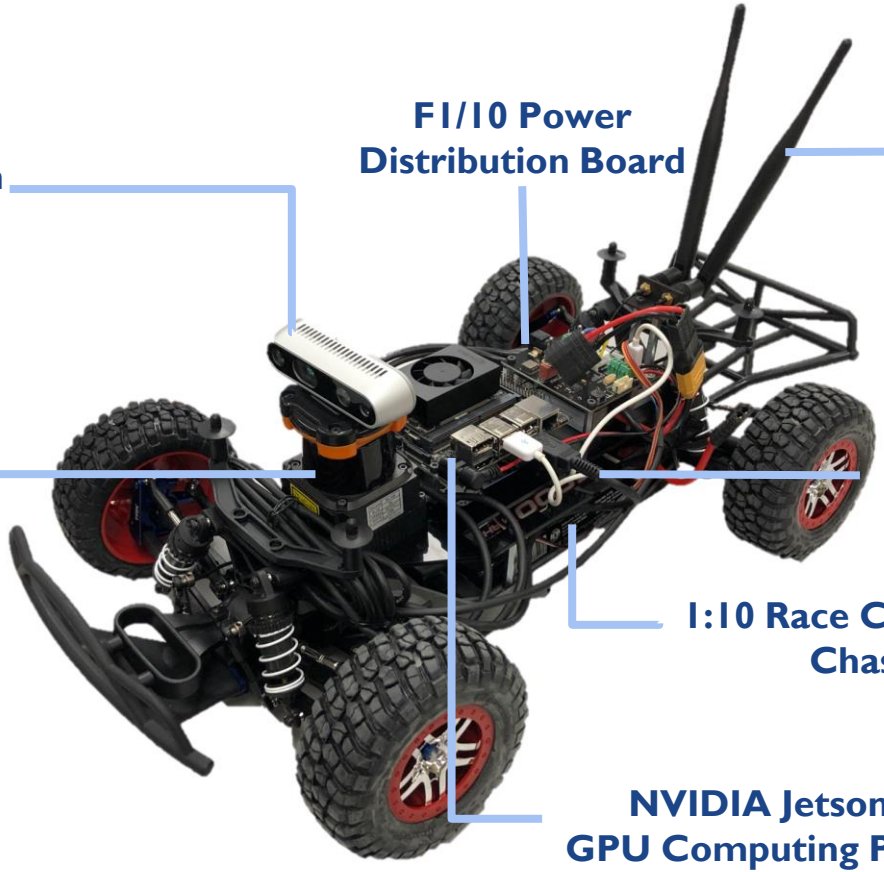
Course outline

- Intro course + basics of AD
- Hardware platform
- ROS2: Installation and profiling
  - Ex: ROS2 to HiL, open a bag
- Navigation: FTG, FTW, Pure pursuit
  - EX: navigation HiL
- Perception: scan matching, PF, LIO?
  - Ex: perception (PF with PThreads)
- Build the car

I do <u>not</u> cover all aspects of AD!!!

- Systems and control theory => Prof. Falcone
- Platforms and algorithms for autonomous systems => Prof. Sanudo & Prof. Falcone
- High-Performance Computing => Prof. Marongiu (FIM)
- Machine Learning => Cucchiara's

The F1/10 platform



Stereovision Camera

F1/10 Power Distribution Board

WiFi Control

Lidar

Single 5000 mAh LiPo Battery

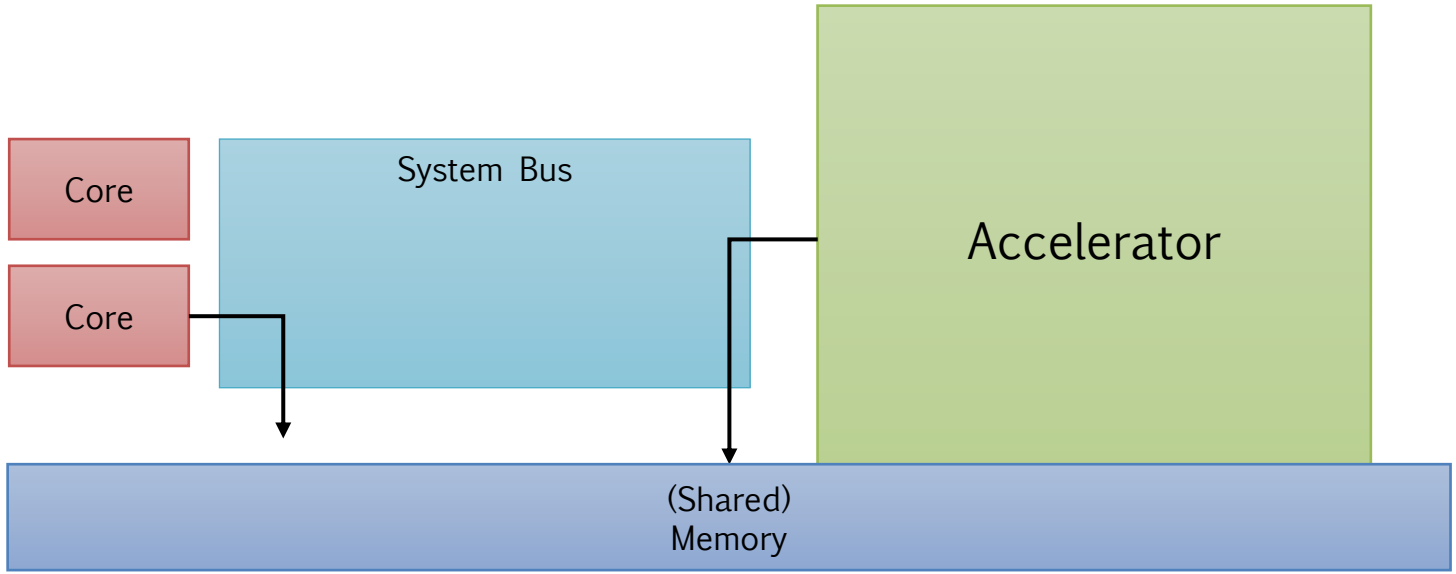
1:10 Race Car Vehicle Chassis

NVIDIA Jetson NX GPU Computing Platform

F1 TENTH

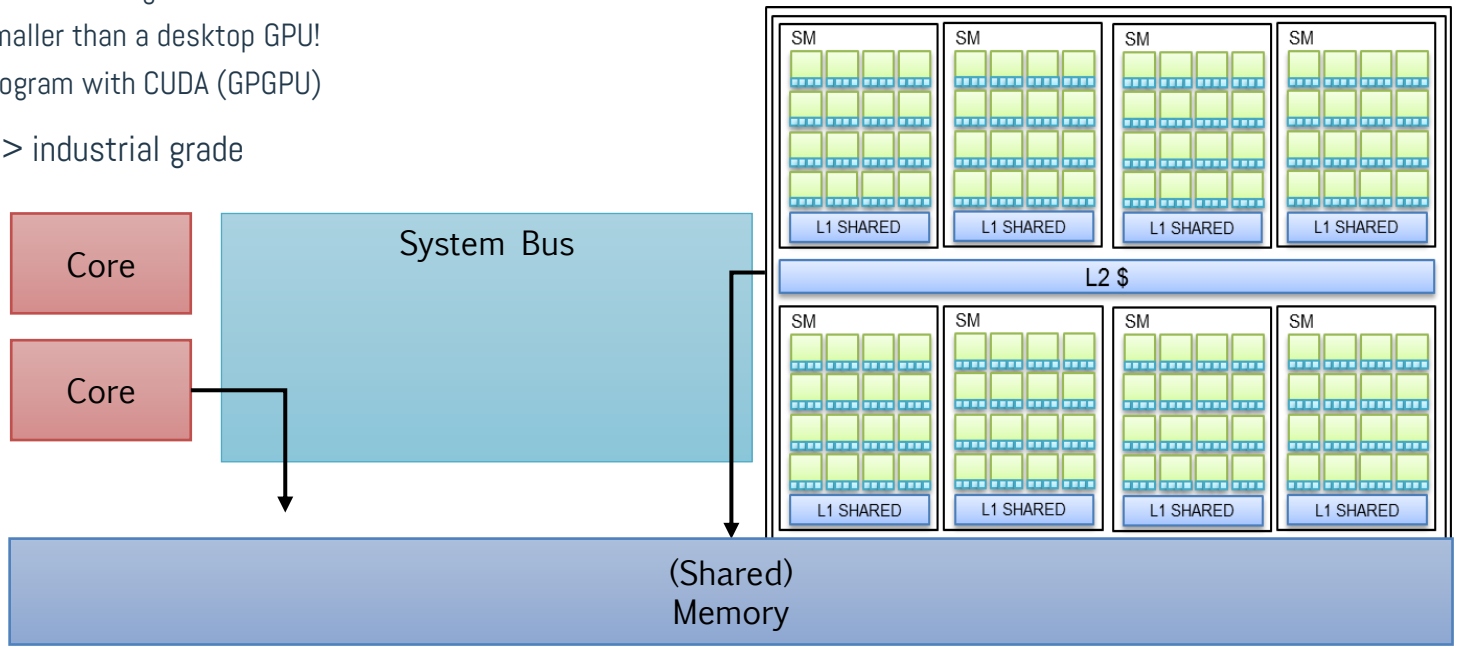
On-board computer - template

- Aka: Domain controller, aka: ECU, aka: …
- Embedded heterogeneous platform
  - Multi-core + data cruncher accelerator
  - Shall employ safety core (Aurix?) to enable automotive ISO26262/ASIL-D compliancy



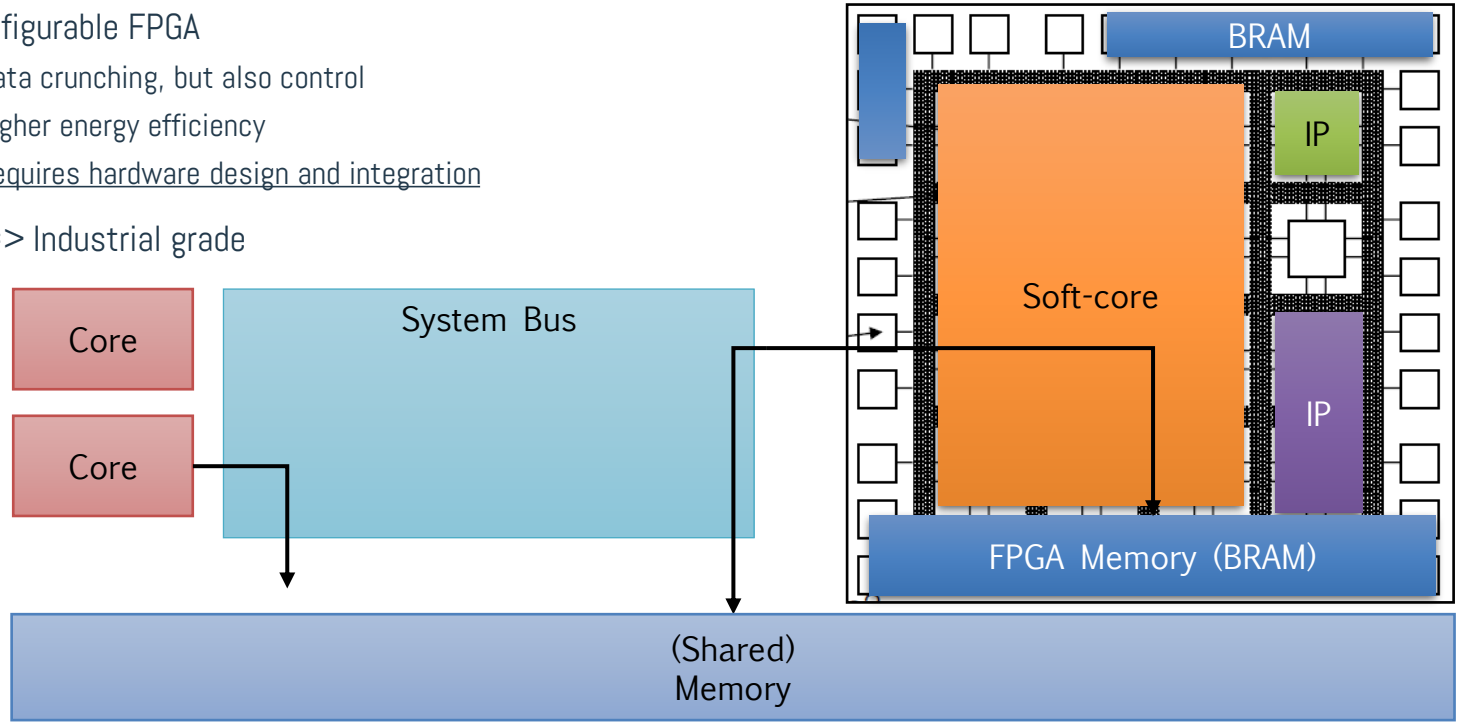
NVIDIA Jetson Orin/NX/Nano

- 6/8 core ARM host
  - For control-based workload
- Embedded GPU
  - Data crunching
  - Smaller than a desktop GPU!
  - Program with CUDA (GPGPU)
- Orin => industrial grade



Xilinx UC+ / Kria

- 6 core ARM host
  - For control-based workload
- Reconfigurable FPGA
  - Data crunching, but also control
  - Higher energy efficiency
  - Requires hardware design and integration
- Kria => Industrial grade



Our F1/10 cars

With GPGPUs

› **Bud** => NVIDIA Orin

› **Kitt** => NVIDIA Xavier NX

With FPGA

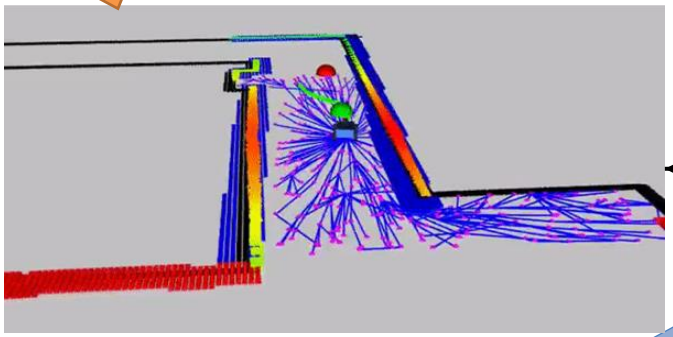
› **Thundershot** F1/10 => Xilinx Kria

› **Frankenstein**/Frankie

- Whatever we want to test
- Currently, cybersecurity

Software ecosystem – Sil vs. HiL

F1tenth gym



AD stack

Perc.

Loc.

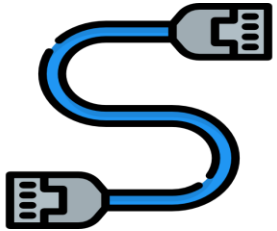
Ctrl

Plan

a

On simulator computer

Software-in-the-Loop



b

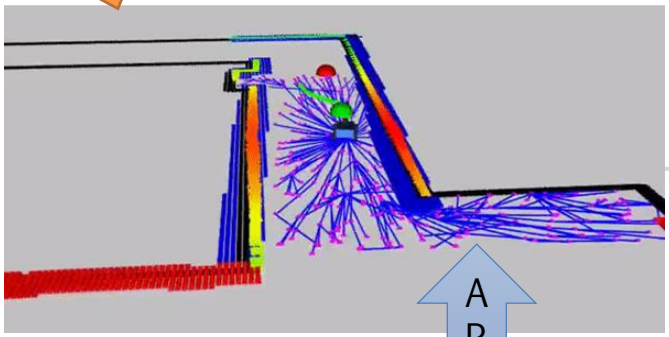
On-board computer

Hardware-in-the-Loop

Same **worst-case** timing behavior

Communication middleware

F1tenth gym



AD stack

Perc.

Loc.

Ctrl

Plan

API

API

Middleware

OS

Computer

OS

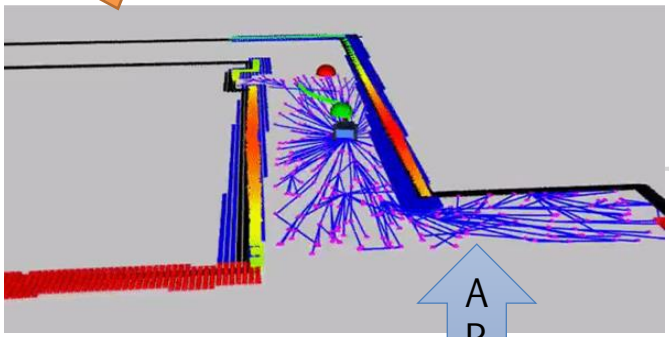
Computer

OS

Computer

Communication middleware (rev'd)

F1tenth gym



AD stack

Perc.

Loc.

Ctrl

Plan

API

API

API

Middleware

OS

Computer

OS

Computer

OS

Computer

The Thundershot project – A real example

Maserati Quattroporte

Carla SIM

AD stack

Perc.

Loc.

Ctrl

Plan

CAN

API

API

API

Middleware

NO OS

STM-something

GNU/Linux

Xilinx ZCU 102

OS

Intel Nuvoo

Robot Operating System

- Peer to peer
- Distributed
- Multi-lingual
- Light-weight
- Free and open-source

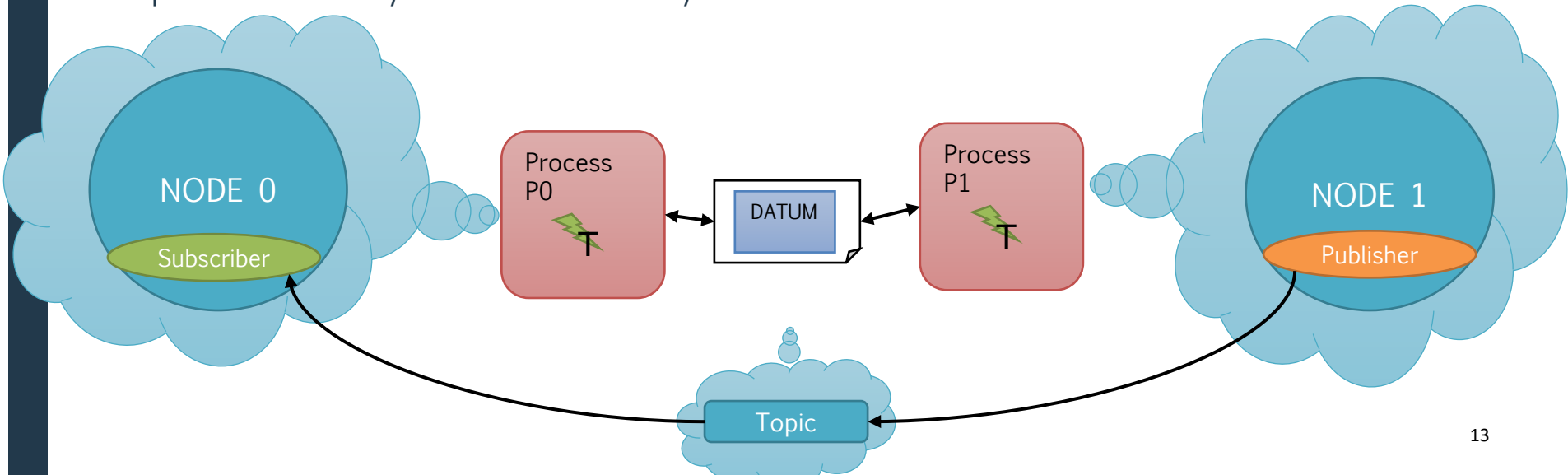
Recap: message-passing vs. shared memory

ROS is a pub-sub middleware

› ROS nodes are processes

› Communication happens via messages called topics

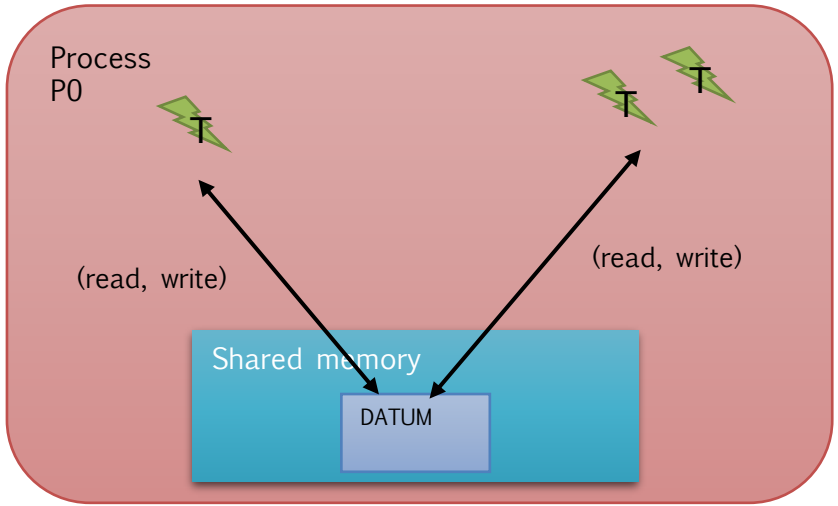
› Implements a *competitive* concurrency

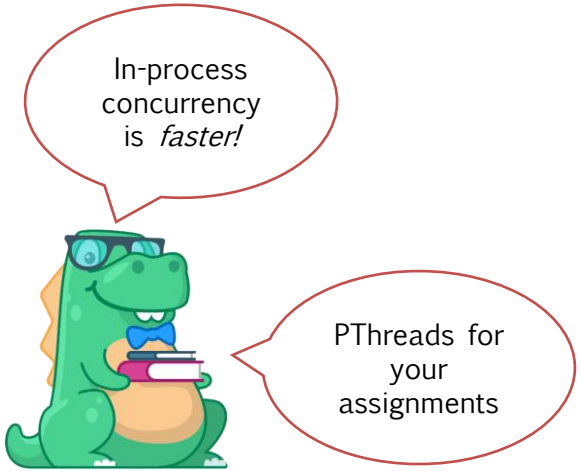


Recap: message-passing vs. shared memory

PThreads (OpenMP, etc..) are shared memory APIs

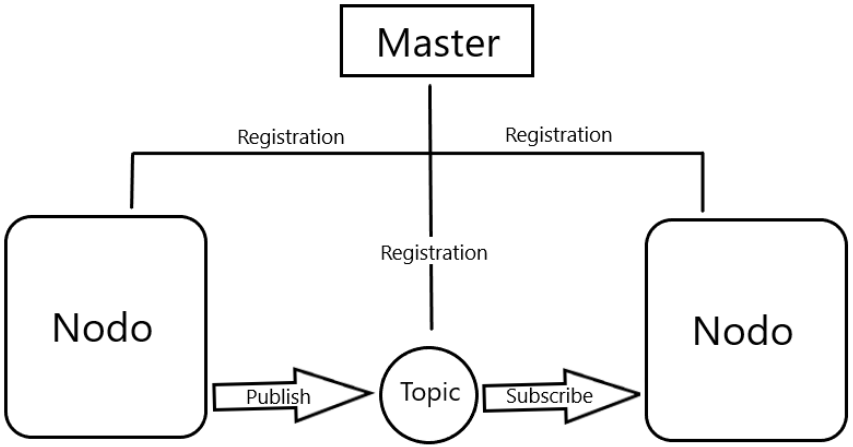
- Identify tasks, assigned to threads
- Communication happens via shared memory (issues with **data races**…)
- Implements *cooperative* concurrency





ROS 1 structure

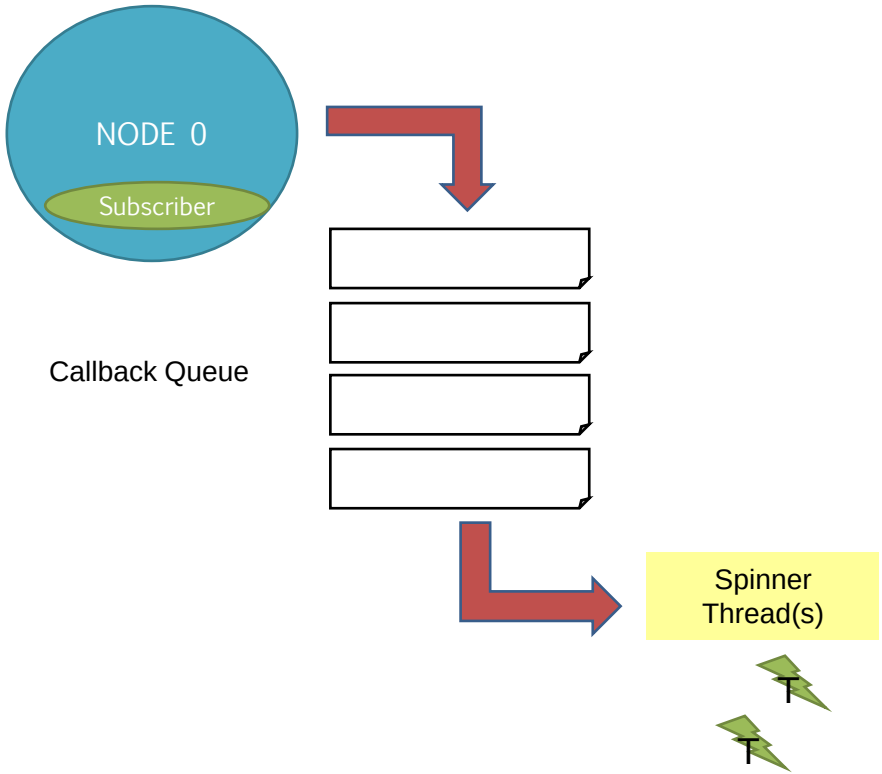
- Modular, based on the concept of nodes
- Message passing (topics)
  - Asynchronous, callbacks
- Package management + rich ROS Client Libraries (RCL)



Communication in ROS

ROS1 is NOT REAL TIME!!!

- No scheduling, no resource mgmt.
- "Simple" FIFO Pub-sub



ROS2 re-engineering process

Libreries

- ROS Client Libraries (RCLs)
- Rigid scheduling

DDS - Data Distribution Services

- ✓Data-Centric Publish-Subsctibe
- ✓Real-Time Publish-Subscribe for Transport
- ✓Distributed (*no master node*)

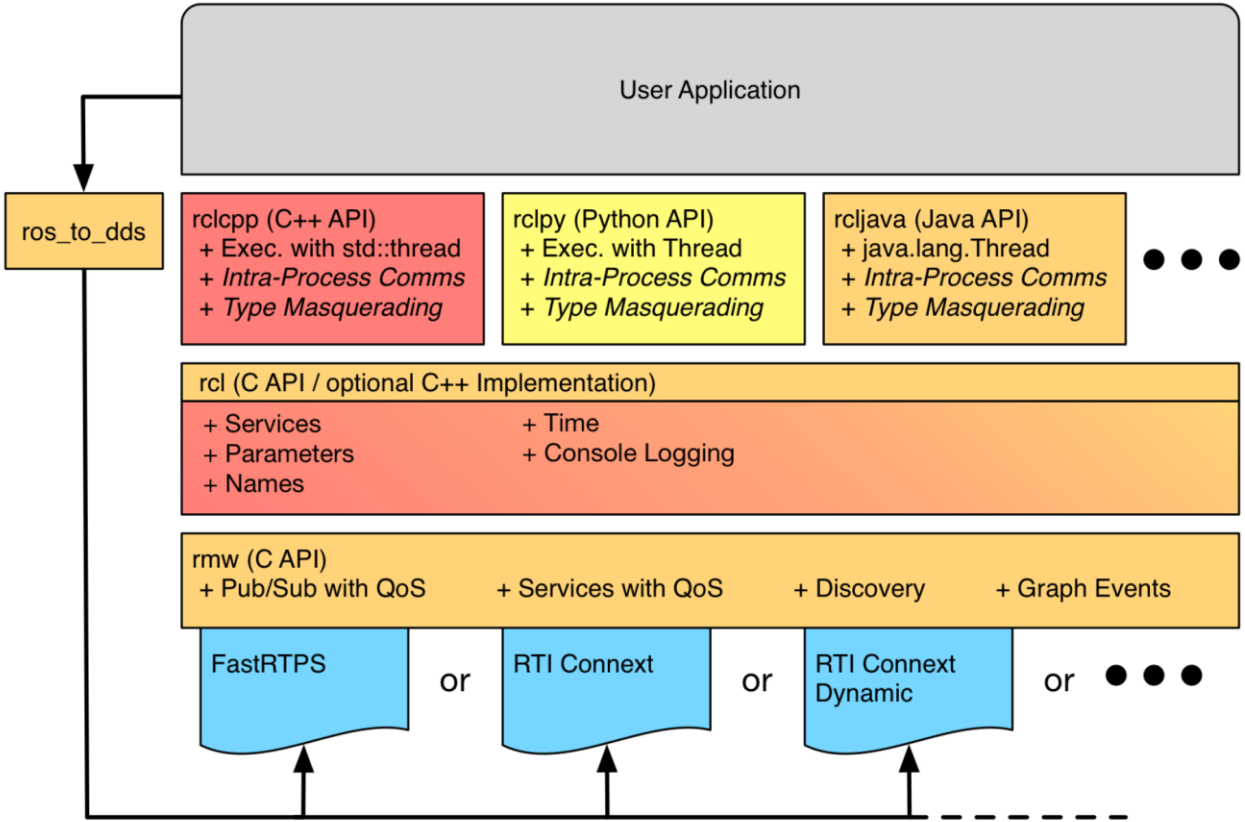
Quality of Service (QoS)

- ✓At the single entity
- ✓Suitable for Real-Time systems ☺

Recap: ROS1 v.s. ROS2

- Language standards: at least C++11 and Python3 for ROS2
  - Tighter Python integration
- Using off the shelf middleware. Now supports discovery, transport and serialization over DDS (Data Distribution Services)
  - Enables communication with non-ROS SW!!!!
- Distributed vs. centralized architecture
  - ROS 1 has a “core” node
- Real time capabilities!!!!
- **API change**

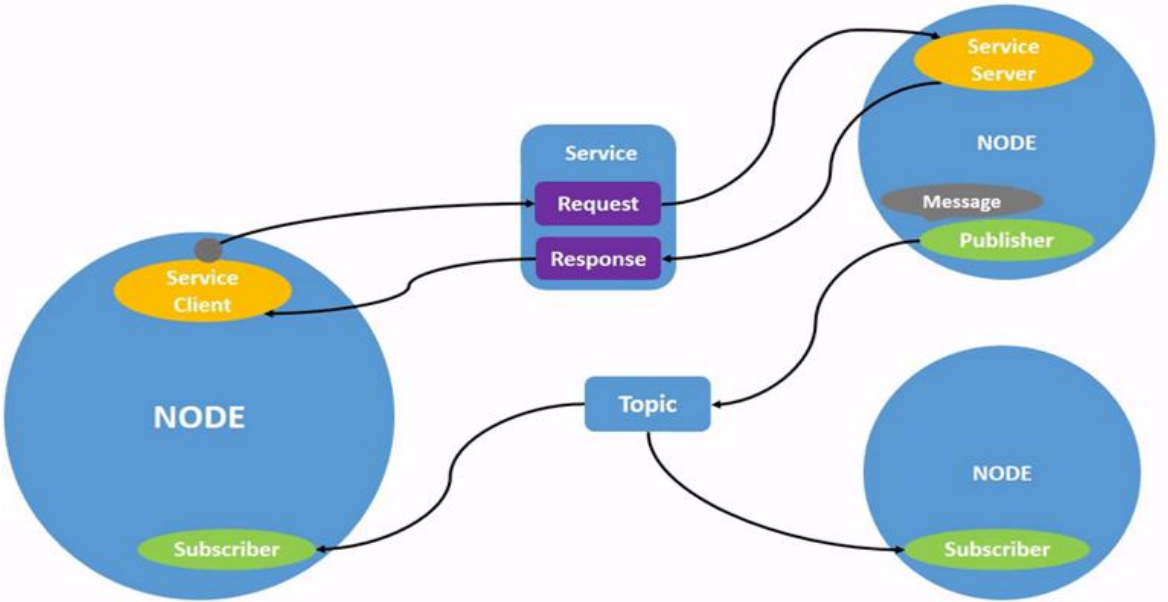
ROS2 stack



* *Intra-Process Comms* and *Type Masquerading* could be implemented in the client library, but may not currently exist.

ROS graph

The ROS graph is a network of ROS 2 elements processing data together at one time. It encompasses all executables and the connections between them if you were to map them all out and visualize them.



ROS Nodes

Each node in ROS should be responsible for a single, module purpose (e.g. one node for controlling wheel motors, one node for controlling a laser range-finder, etc)

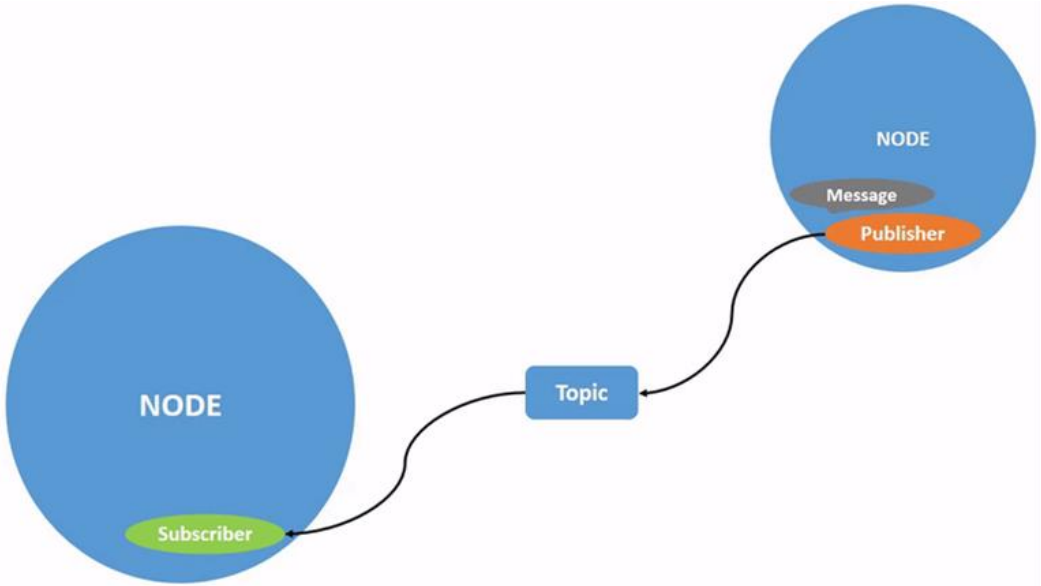
Each node can send and receive data to other nodes via topics, services, actions, or parameters.

Related command line commands

- `ros2 run <package_name> <executable_name>`
- `ros2 node list`
- `ros2 node info <node_name>`

Topics

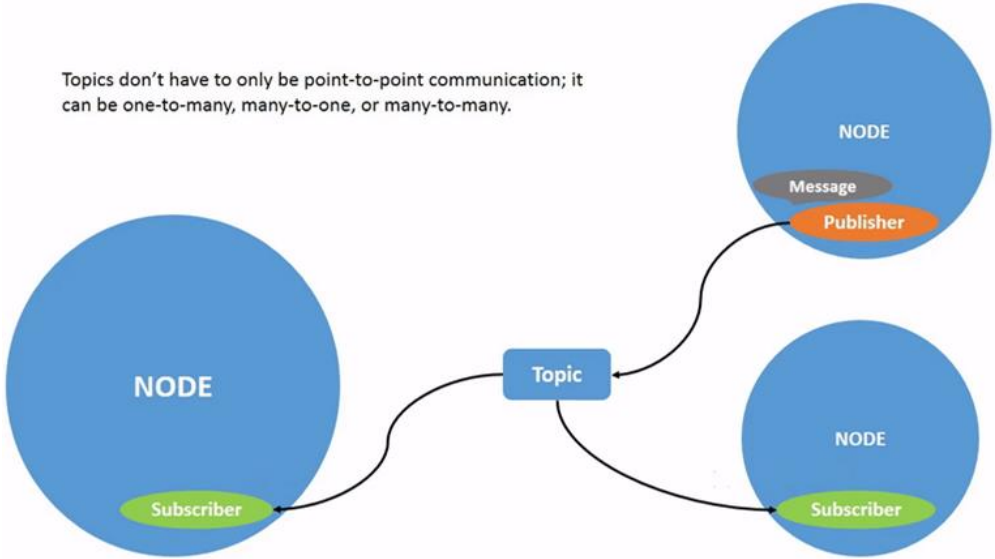
ROS 2 breaks complex systems down into many modular nodes. Topics are a vital element of the ROS graph that act as a bus for nodes to exchange messages.



Topics

A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.

Topics are one of the main ways in which data is moved between nodes and therefore between different parts of the system.



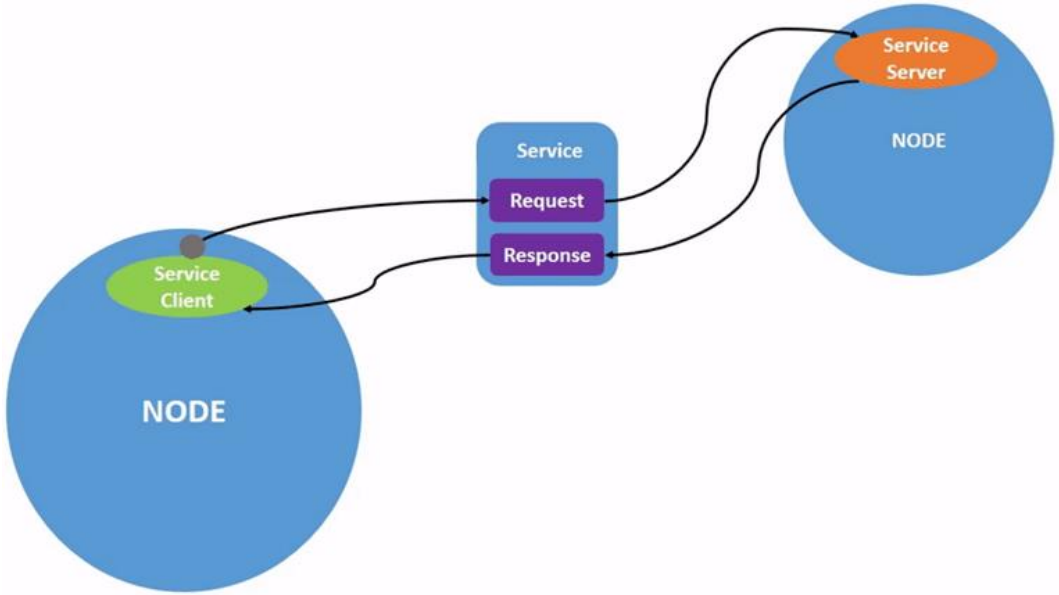
Topics

Related command line commands

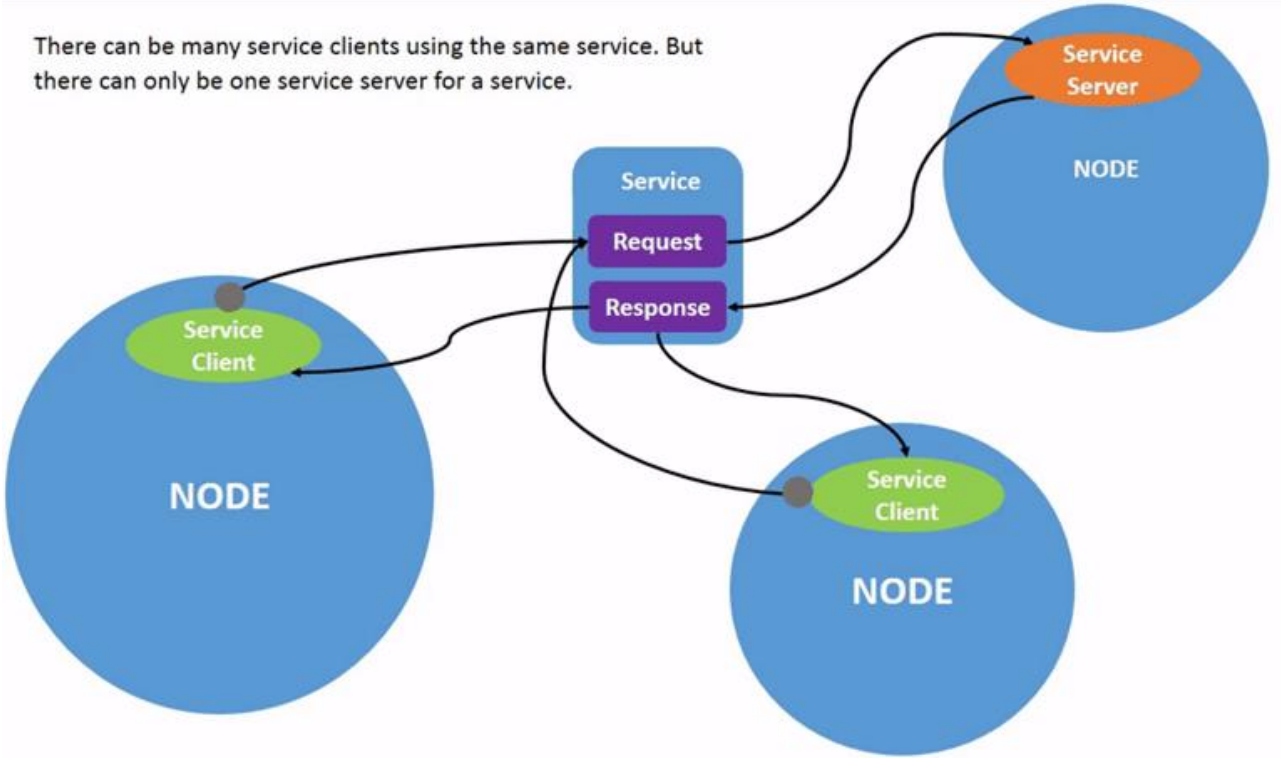
- `rqt_graph`
- `ros2 topic list`
- `ros2 topic list -t`
- `ros2 topic echo <topic_name>`
- `ros2 topic info <topic_name>`
- `ros2 interface show <msg_type>`
- `ros2 topic pub <topic_name> <msg_type> '<args>'`
- `ros2 topic hz <topic_name>`

Services

Services are another method of communication for nodes in the ROS graph. Services are based on a call-and-response model, versus topics' publisher-subscriber model. While topics allow nodes to subscribe to data streams and get continual updates, services only provide data when they are specifically called by a client.



Services


There can be many service clients using the same service. But there can only be one service server for a service.
Service Server
NODE
Service
Request
Response
Service Client
NODE
Service Client
NODE

Services

Related command line commands

- `ros2 service list`
- `ros2 service type <service_name>`
- `ros2 service list -t`
- `ros2 service find <type_name>`
- `ros2 interface show <type_name>.srv`
- `ros2 service call <service_name> <service_type> <arguments>`

Parameters

- A parameter is a configuration value of a node
- Can be integers, floats, booleans, strings and lists
- All parameters are dynamically reconfigurable, and built off of ROS 2 services.

Related command line commands

- `ros2 param list`
- `ros2 param get <node_name> <parameter_name>`
- `ros2 param set <node_name> <parameter_name> <value>`

Could also use an yaml file to define the parameters.

Parameters

- Parameters could also be set in either a launch file or a yaml file.
- Useful to have all parameters in one place while tuning.
- Set parameters for multiple nodes in one file.
- For a full length tutorial: https://roboticsbackend.com/ros2-yaml-params/

Parameters

Getting ROS parameters programmatically:

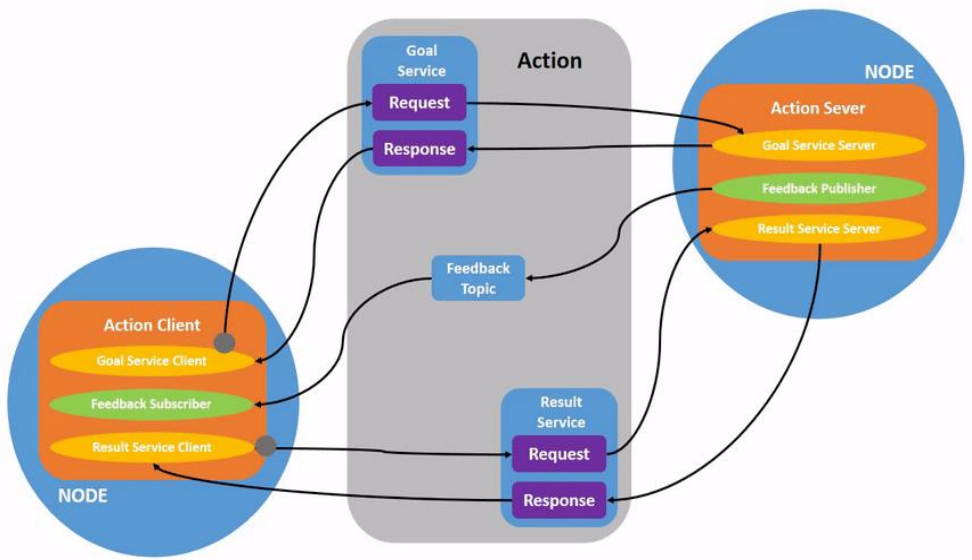
- In a node, declare a parameter first
  - `self.declare_parameter('my_param')`
- Then getting a parameter
  - `self.get_parameter('my_param')`
- You could also get multiple parameters at once
- Similar in C++
- For full tutorials on parameters see:
  - https://roboticsbackend.com/rclpy-params-tutorial-get-set-ros2-params-with-python/
  - https://roboticsbackend.com/rclcpp-params-tutorial-get-set-ros2-params-with-cpp/

Actions

Actions are one of the communication types in ROS 2 and are intended for long running tasks. They consist of three parts: a goal, feedback, and a result.

Actions are built on topics and services. Their functionality is similar to services, except actions are preemptable (you can cancel them while executing). They also provide steady feedback, as opposed to services which return a single response.

Actions use a client-server model, similar to the publisher-subscriber model (described in the topics tutorial). An "action client" node sends a goal to an "action server" node that acknowledges the goal and returns a stream of feedback and a result.



Quality of Service in ROS

ROS 1 uses TCP as the underlying transport

› “No good” (cit.) for lossy networks such as wireless links

ROS 2 uses DDS => UDP

› Better control over the level of reliability a node can expect and act accordingly

› Specify for: Topic, DataReader, DataWriter, Publisher and Subscriber

› Request vs Offerer Model. Publications and Subscriptions will match if the QoS are compatible

QoS policies in ROS2

- History
  - *Keep last* (only store up to N samples) vs. *Keep all*
- Depth
  - *Queue size* (if "history" was set to *Keep last*)
- Reliability
  - *Best effort* (lose samples if the network is not robust) vs. *Reliable* (might retry)
- Durability
  - *Transient local*: the publisher becomes responsible for persisting samples for "late-joining" subscriptions.
  - *Volatile*: no attempt is made to persist samples.

QoS policies in ROS2 – cont'd

- Deadline
  - *Duration*: the expected maximum amount of time between subsequent messages being published to a topic
- Lifespan
  - *Duration*: the maximum amount of time between the publishing and the reception of a message without the message being considered stale or expired (then dropped)
- Liveliness
  - *Automatic*: the system will consider all of the node's publishers to be alive for another "lease duration" when any one of its publishers has published a message.
  - *Manual by topic*: the system will consider the publisher to be alive for another "lease duration" if it manually asserts that it is still alive (via a call to the publisher API).
- Lease Duration
  - *Duration*: the maximum period of time a publisher has to indicate that it is alive before the system considers it to have lost liveliness (losing liveliness could be an indication of a failure).

QoS profiles in ROS2

Defines a set of policies that are expected to go well together for a particular use case.

› Remember that they must be compatible at both ends!

Default QoS settings for publishers and subscriptions

› *Keep last* for history with a *queue size* of 10, *reliable* for reliability, *volatile* for durability, and *system default* for liveliness. Deadline, lifespan, and lease durations are also all set to "*default*"

Services

› Services shall use *volatile* durability, as otherwise service servers that re-start may receive outdated requests

Sensor data

› More important to receive readings in a timely fashion, rather than ensuring that all of them arrive

› *Best effort* reliability and a smaller queue size

Parameters

› Parameters in ROS 2 are based on services, but with a larger queue depth

System default

› This uses the RMW (*Ros MiddleWare* runtime) implementation's default values

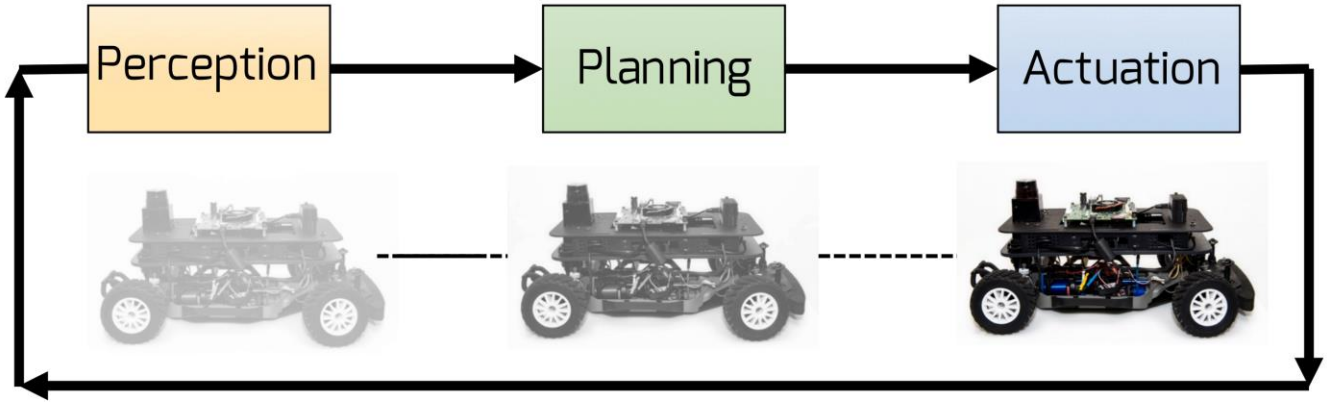
ROS2 and F1/10

Real-Time Embedded System - The F1tenth autonomous racing

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

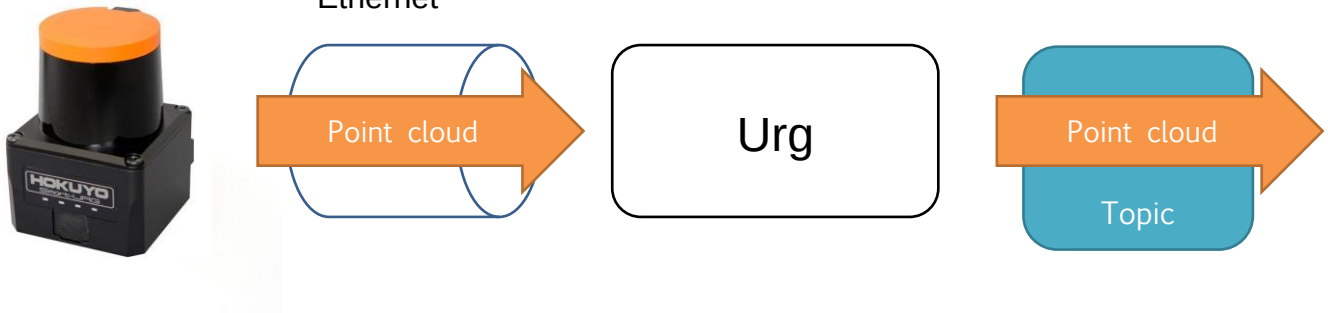
High Performance Real Time Lab

Our simplest stack



Urg → Localize (?) → Follow The Gap → Teensy

Urg – Data from LiDAR



Perception/Localization

Is this really necessary?

› YES! But not always…

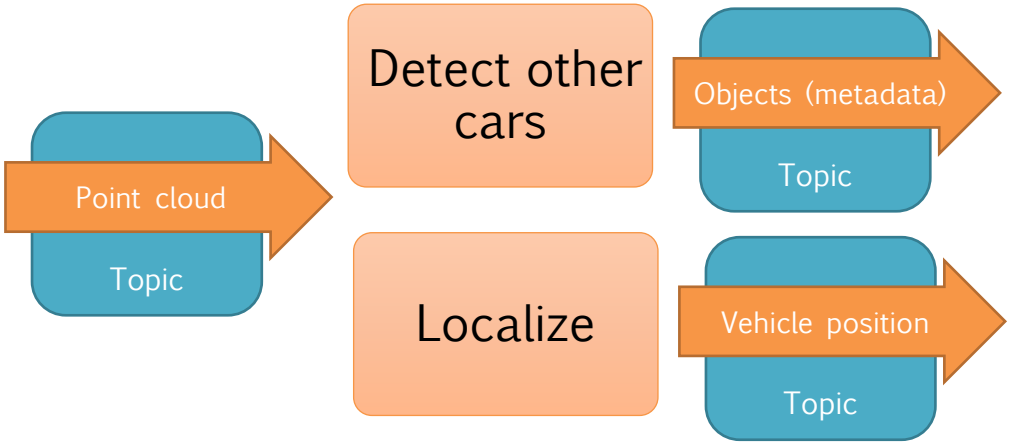
› Remember, you have limited computing power…

In a time attack race, no overtaking/obstacle avoidance

› We use only (faster) local planners

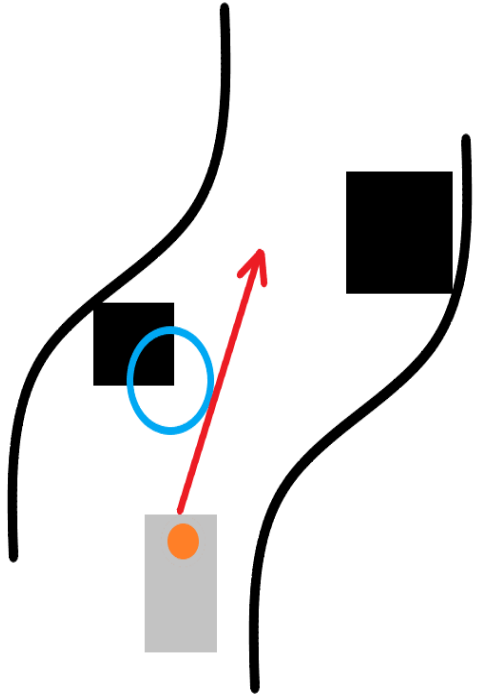
Within a simulator, you can directly fetch vehicle position

› Why is this useful?



Follow The Gap

- Local planner
- Fast, reactive
- Non-optimal trajectories
- Suitable for overtaking!



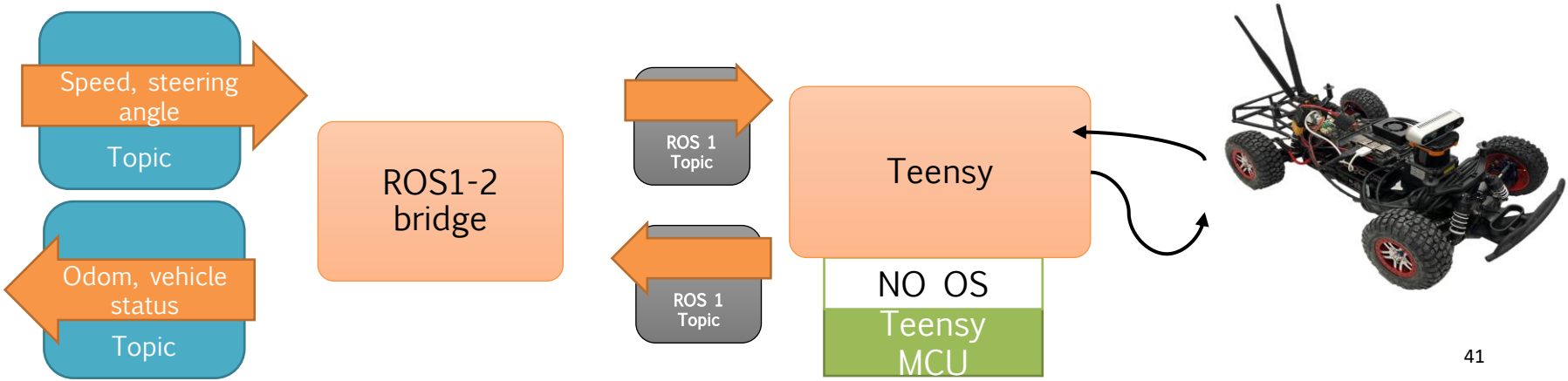
«Teensy» ROS node

Teensy is the microcontroller that controls the brushless engine

› Typical scenario, also real cars have legacy actuator ECUs!

Note written in ROS1 (teensy has no OS!)

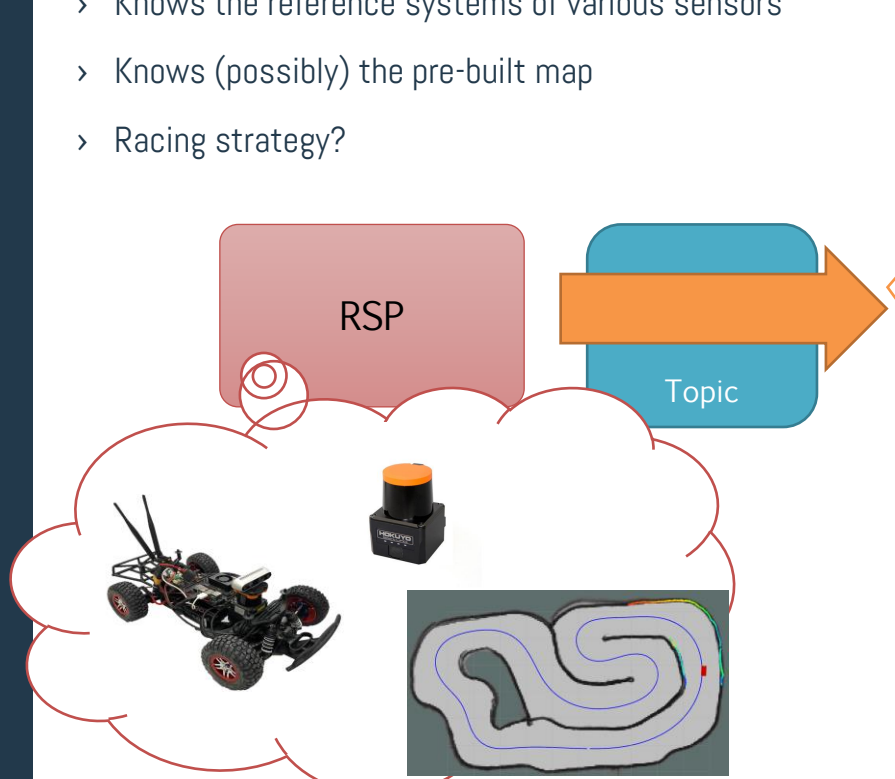
› ROS1-to-ROS2 bridge

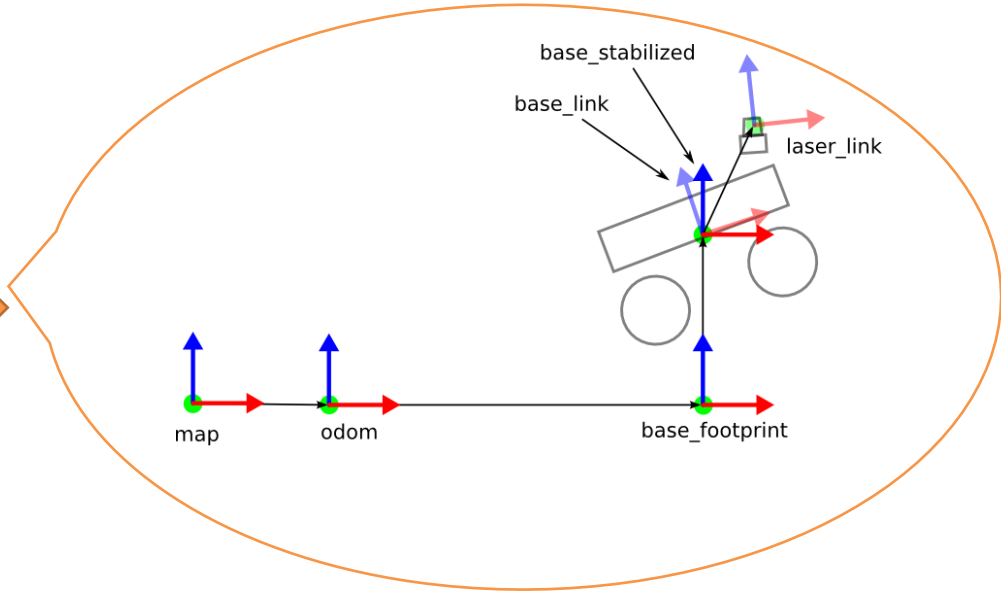


Notable nodes - Robot State Publisher

Required for ROS2 components to work – stores “static” information

- Knows the reference systems of various sensors
- Knows (possibly) the pre-built map
- Racing strategy?





References

Course website

- http://personale.unimore.it/rubrica/contenutiad/markober/2023/71846/N0/N0/10005
- https://github.com/HiPeRT/F1tenth-RTES
  - Online resources/preview

My contacts

- paolo.burgio@unimore.it
- http://hipert.mat.unimore.it/people/paolob/

Resources

- https://docs.ros.org/en/foxy/Tutorials.html
- https://roboticsbackend.com/category/ros2/
- https://it.mathworks.com/help/ros/ug/manage-quality-of-service-policies-in-ros2.html
- A "small blog»
  - http://www.google.com