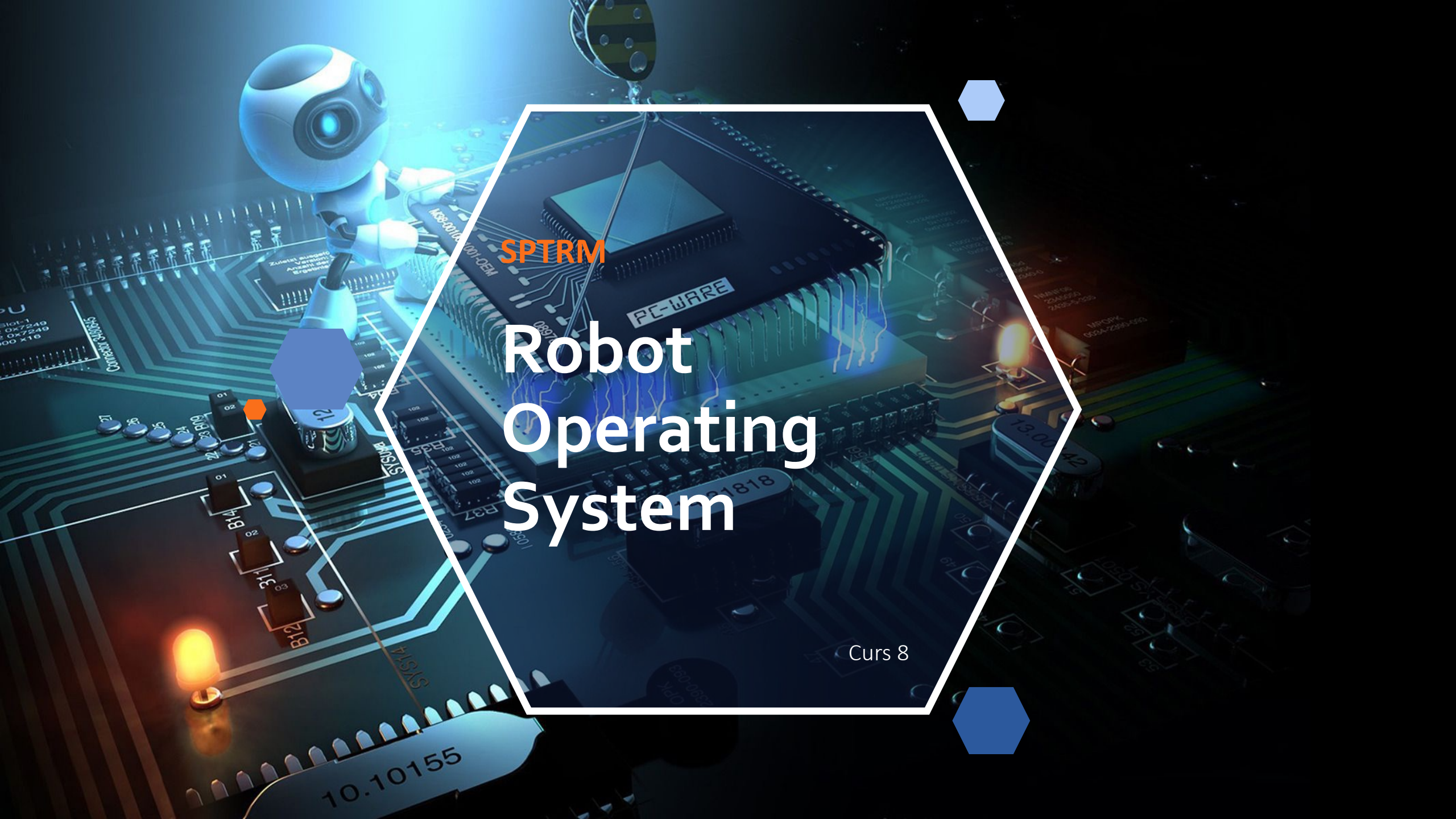


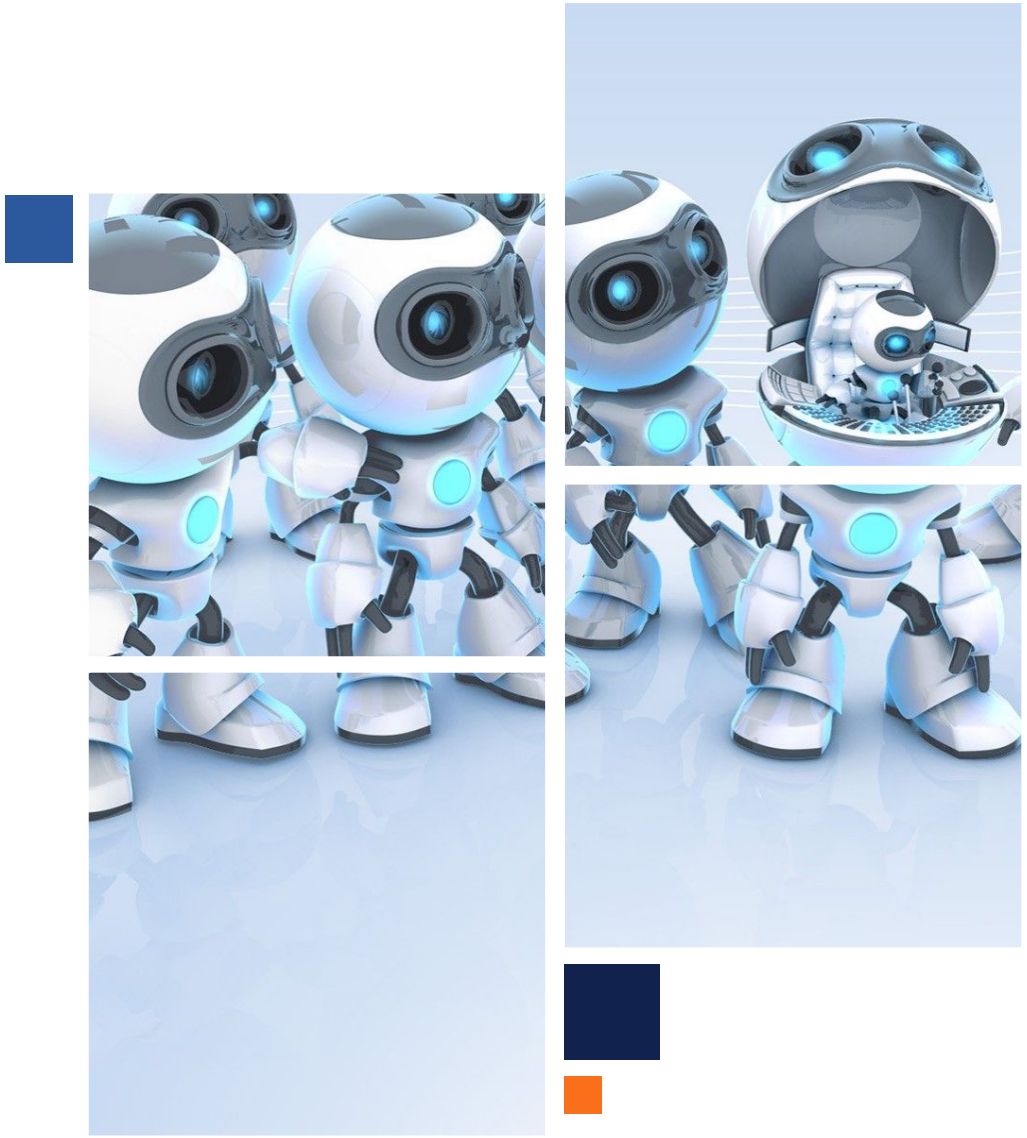
SPTRM

Robot Operating System

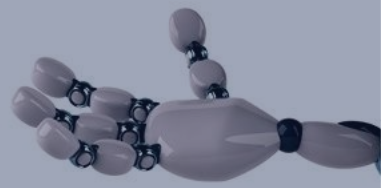
Curs 8

Agenda

- Recap
- Real-time
  - Definition
  - Types
  - Real-time communication
- RT in ROS
  - RT in ROS
  - ROS2

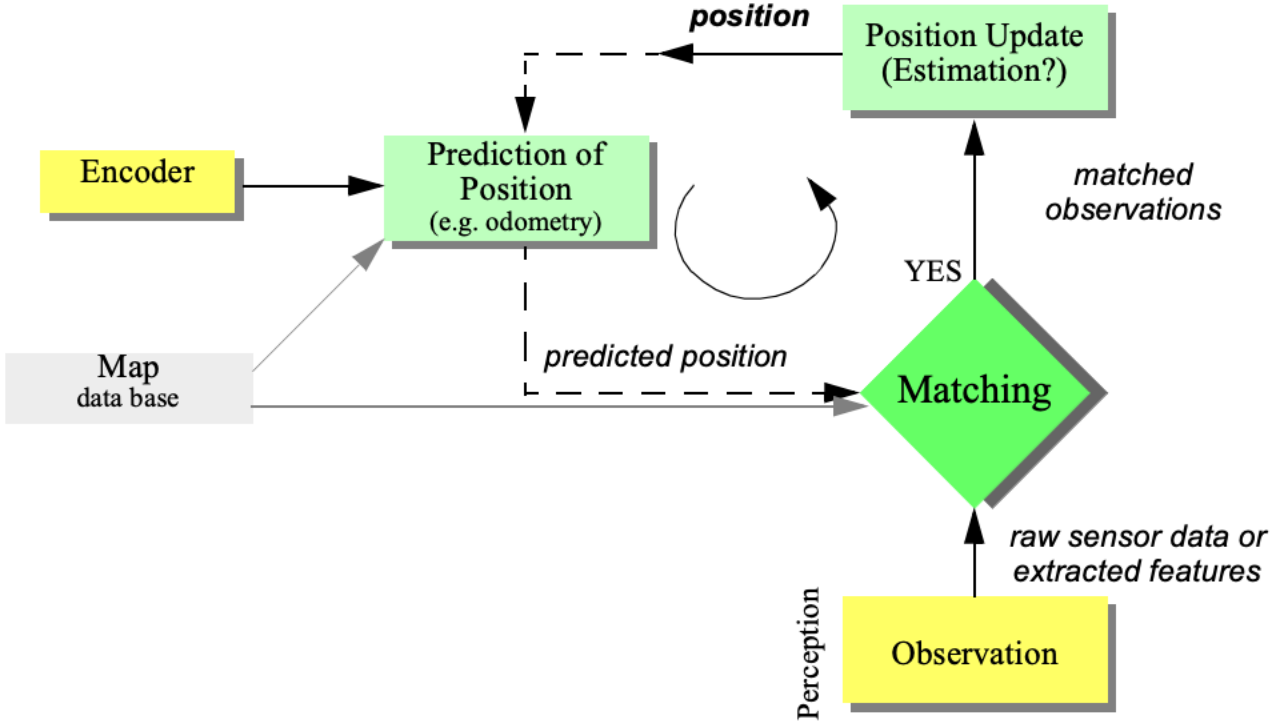


Recap



From the last episode...

Robot Localization



SLAM

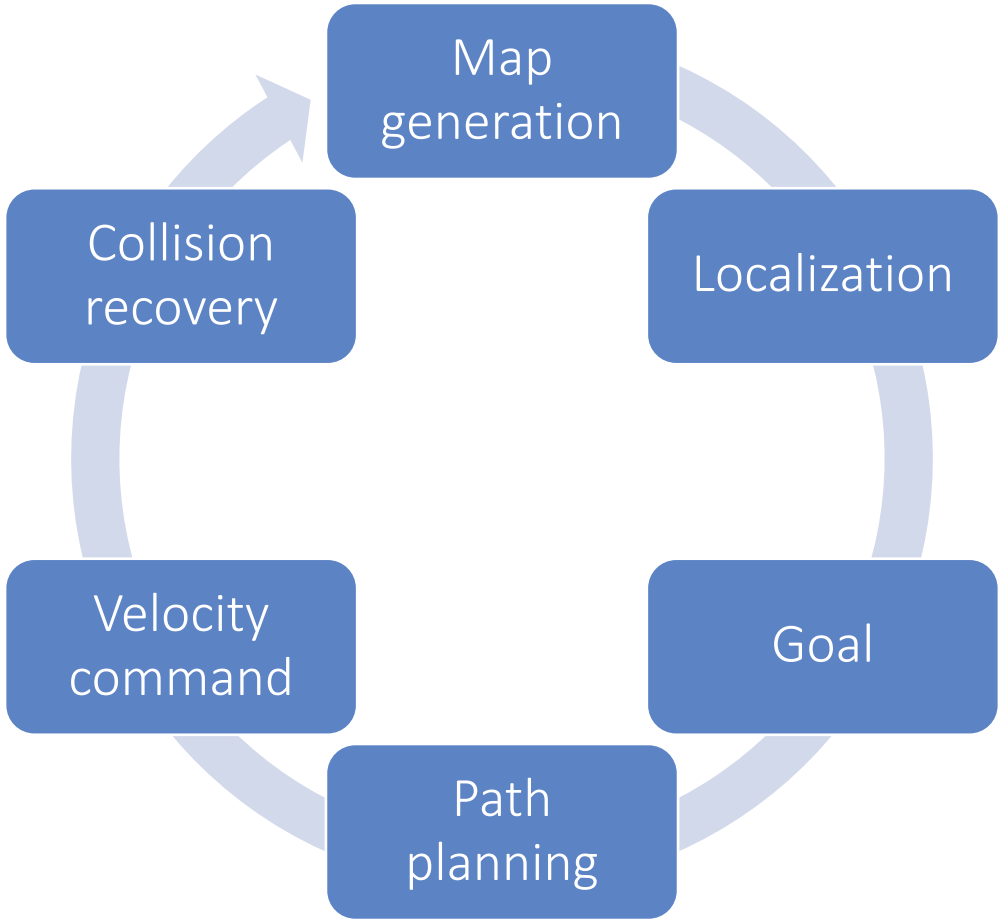
- **Simultaneous localization and mapping (SLAM)** is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it

- Fundamental problem for robots to become truly autonomous

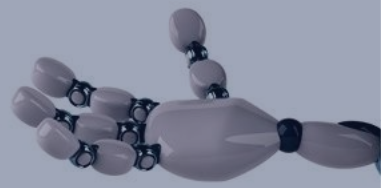
- Inputs:
  - control data
  - observations (sensor data)

- Outputs:
  - agent location
  - map of the environment

Navigation


Map generation
Localization
Goal
Path planning
Velocity command
Collision recovery

Real Time



Real-time systems. Real-time communcation

Real-time systems

- A **real-time system** is any information processing system which has to respond to externally generated input stimuli within a finite and specified period
- It's about **determinism**, not **performance**
- The correctness depends not only on the logical result but also the time it was delivered (**Correct computation** delivered at the **correct time**)
- **Failure to respond** is as bad as the **wrong response**!
- Robotic systems need to be responsive
- Mission critical applications

Definitions

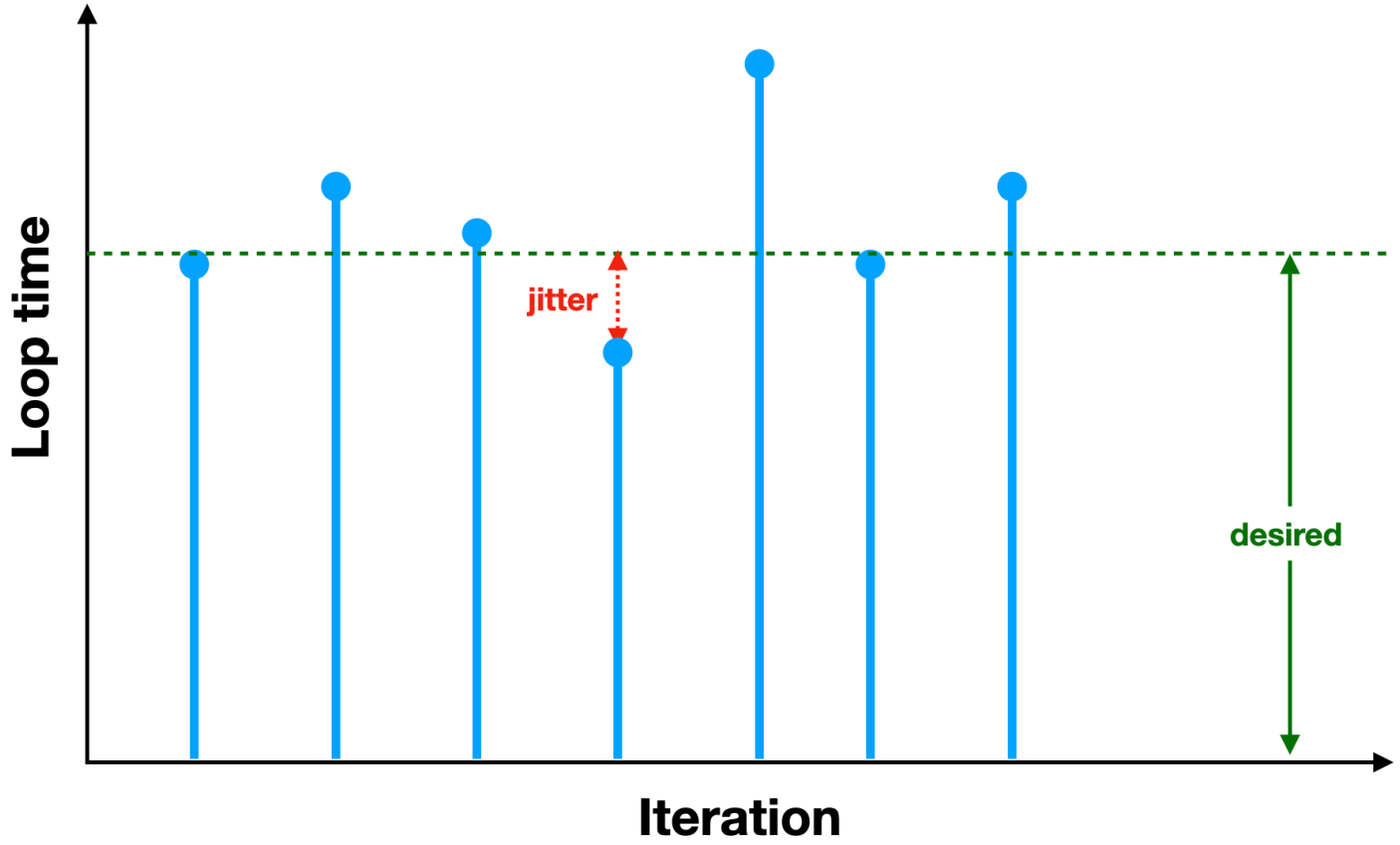
- **Determinism**: A system is deterministic if it always produces the same output for a known input. The output of a nondeterministic system will have random variations.

- **Deadline**: A deadline is the finite window of time in which a certain task must be completed.

- **Quality of Service**: The overall performance of a network. Includes factors such as bandwidth, throughput, availability, jitter, latency, and error rates.

Real-time constraints

- **Jitter**: The variation in the periodicity of a signal or periodic event from its target frequency



Types

- Real-time systems guarantees correct computation at the correct time.
  - Hard real-time systems
  - Soft real-time systems
  - Firm real-time systems

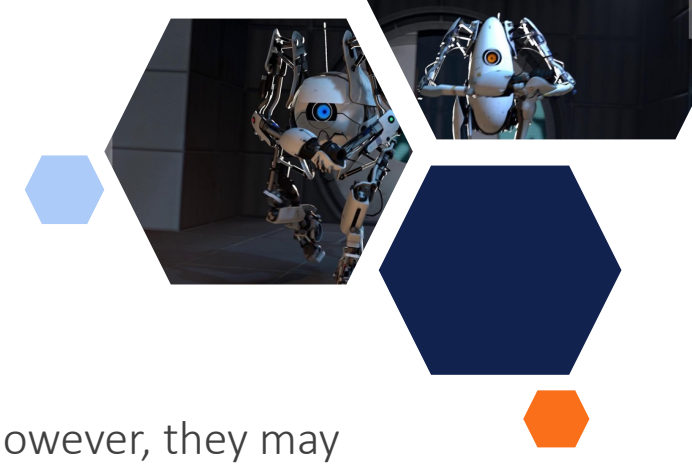
Hard real-time systems

- **Hard real-time systems** have a set of strict deadlines, and missing a deadline is considered a system failure.
- Examples: airplane sensor and autopilot systems, spacecrafts and planetary rovers.
- **Strongly hard real-time systems** = must meet all of their deadlines
- **Weakly hard real-time systems** = the distribution of its met and missed deadlines during a window of time is precisely bounded

Weakly hard real-time systems

- **Missed deadline** can be associated with:
  - **Delayed completion**: The task invocation runs until completion even though it finishes after the deadline.
  - **Abortion**: The task invocation is terminated before finishing its computation either because it will not end its computation by the deadline or because another (more important) task needs the resource.
  - **Rejection**: The task invocation is not accepted into the system.
  - **Skip**: The task invocation is not released and the whole invocation is not executed.

Soft real-time systems



- **Soft real-time systems** try to reach deadlines but do not fail if a deadline is missed. However, they may degrade their quality of service in such an event to improve responsiveness.

- Examples: audio and video delivery software for entertainment (lag is undesirable but not catastrophic).

Firm real-time systems

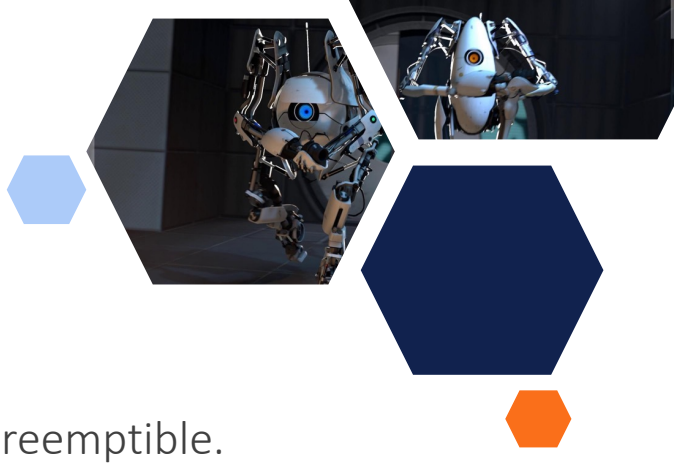
- **Firm real-time systems** treat information delivered/computations made after a deadline as invalid. Like soft real-time systems, they do not fail after a missed deadline, and they may degrade QoS if a deadline is missed.

- Examples: financial forecast systems, robotic assembly lines.

Scheduling

- Task = Running / Ready / Blocked
- Algorithms
  - Cooperative scheduling
  - Preemptive scheduling
    - Rate-monotonic scheduling
    - Round-robin scheduling

Examples

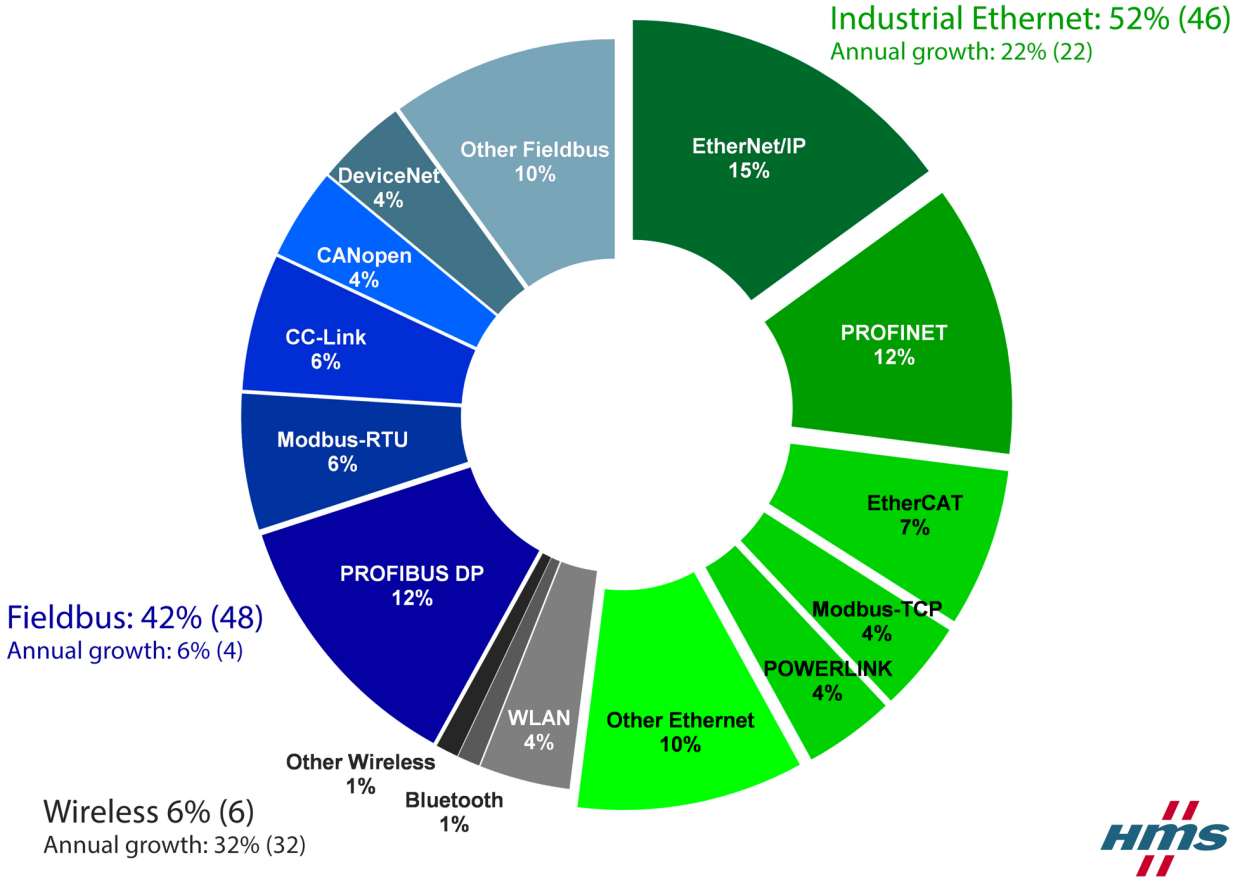


- The **RT_PREEMPT** Linux kernel patch, which modifies the Linux scheduler to be fully preemptible.
- **Xenomai**, a POSIX-compliant co-kernel (or hypervisor) that provides a real-time kernel cooperating with the Linux kernel. The Linux kernel is treated as the idle task of the real-time kernel's scheduler (the lowest priority task).
- **RTAI**, an alternative co-kernel solution.
- **QNX Neutrino**, a POSIX-compliant real-time operating system for mission-critical systems.
- **RTOS** -for embedded devices

Real-Time Communication

- Communication between two systems without transmission delays
- Nearly instant with minimal latency
- Peer-to-peer communication
- Half-duplex = data can be transmitted in both directions but not at the same time
- Full-duplex = data can be transmitted in both directions simultaneously

RTC protocols



https://www.anybus.com/about-us/news/2018/02/16/industrial-ethernet-is-now-bigger-than-fieldbuses

Industrial Ethernet (IE)

- Fieldbus - family of industrial computer network protocols used for real-time distributed control (IEC 61158 standard)
- Modified Media Access Control (MAC) layer - low latency and determinism
- Requirements: high availability, predictable performance and maintainability, security
- Challenges: harsh environments, electrical noise, electrical isolation, high temperatures
- Examples: EtherNet/IP, EtherCAT, PROFINET, POWERLINK, SERCOS III, CC-Link IE, Modbus TCP

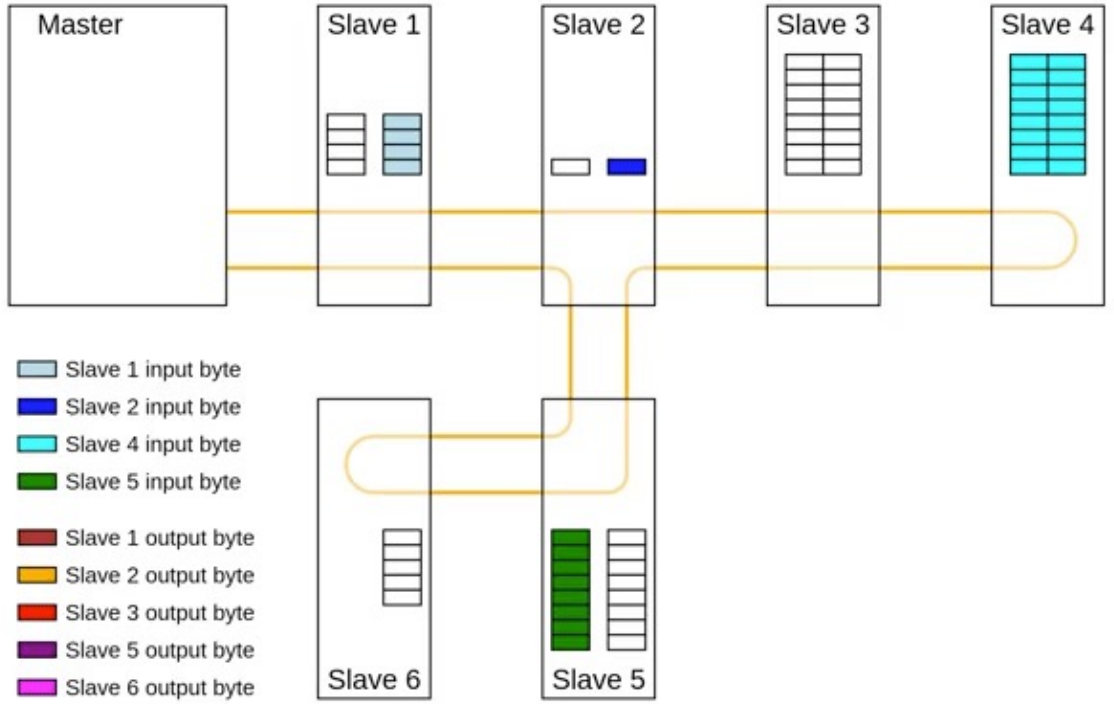
EtherNet/IP

- application layer protocol within a TCP/IP Packet
- based on the Common Industrial Protocol (CIP)
- supports various topologies: star (traditional Ethernet infrastructure) or device level ring (DLR)
- hot-swap devices
- managed by Open DeviceNet Vendors Association

EtherCAT

- EtherCAT = Ethernet for Control Automation Technology
- Supports both hard and soft real-time computing requirements
- “on the fly” data processing:
  - the slave devices read the data addressed to them while the telegram passes through the device
  - input data are inserted while the telegram passes through
  - a frame is not completely received before being processed
- Entire network is addressed in one frame

EtherCAT



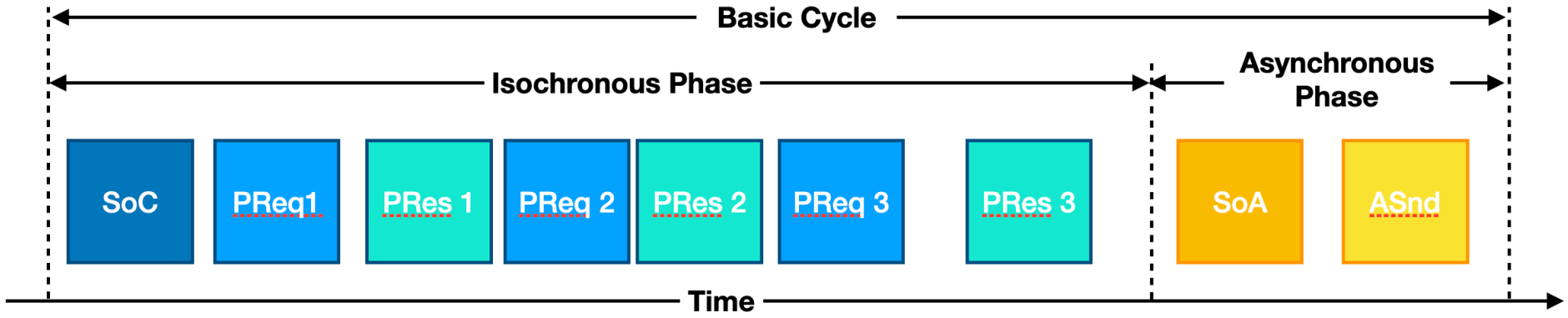
https://upload.wikimedia.org/wikipedia/commons/transcoded/8/8f/EthercatOperatingPrinciple.webm/EthercatOperatingPrinciple.webm.480p.vp9.webm

Ethernet Powerlink

- open-source real-time protocol for standard Ethernet
- managed by Ethernet POWERLINK Standardization Group
- introduces mixed polling and timeslicing mechanisms to Ethernet
- https://www.ethernet-powerlink.org/powerlink/technology

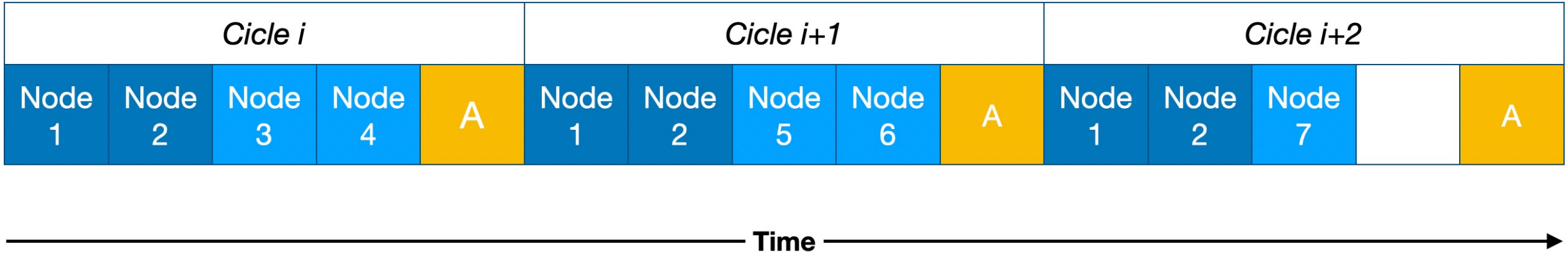
Ethernet Powerlink

- Cycle based communication controlled by a Managing Node (MN)
- Each cycle consists of 2 phases:
  - Isochronous Phase
    - MN is sending a synchronization message to all nodes called the Start of Cycle (SoC) frame
    - MN calls each node to transfer time-critical data Poll Request (PReq) frame
    - Each node answers with the Poll Response (PRes) frame
    - All nodes listen to the data => producer - consumer relationship
  - Asynchronous Phase
    - MN grants one node the right for sending ad-hoc data by sending out a Start of Asynchronous (SoA) frame
    - The adressed node responds with an Asynchronous Send (ASnd) frame



Ethernet Powerlink

- Bandwidth optimization can be achieved by multiplexing less important nodes
- Example: Nodes 1 and 2 are sending data every cycle while nodes 5, 6 and 7 are sharing transfer slots



PROFINET

- PROFINET = Process Field Net

- Device types:
  - IO-Controller - controls the automation task
  - IO-Device - field device, monitored by an IO-controller
  - IO-Supervisor - software used to parameter configuration and diagnosis

- https://profinetuniversity.com/category/profinet-basics/

PROFINET Classes

- PROFINET Class A (CC-A) devices
  - Real-Time (RT) channel - standard cyclic data transfers
- PROFINET Class B (CC-B) devices
  - support for Simple Network Management Protocol (SNMP)
  - support for Link Layer Discovery Protocol (LLDP)
- PROFINET Class C (CC-C) devices
  - Isochronous Real Time (IRT) channel - high speed channel used for Motion Control applications

Isochronous Real-Time

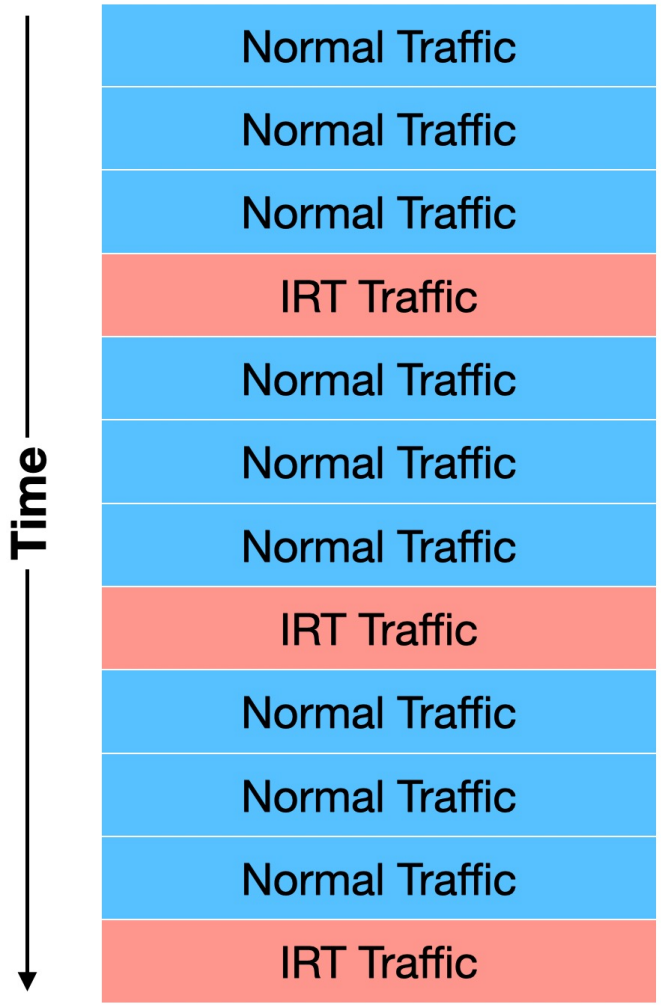
- Synchronising the open network

- Carrier Sense Multiple Access – Collision Detect (CSMA-CD)
  - each node is responsible for detecting a collision and retransmitting their data if one occurs
  - standard in Ethernet

- Time Division Multiple Access (TDMA):
  - MAC-layer extension
  - uses time slices for deterministic behaviour and strictly sequenced data

Isochronous Real-Time

- 75% of time networks operates normally
- 25% of time only IRT traffic is allowed while any other traffic is buffered
- IRT time slices use a quarter of total network bandwidth



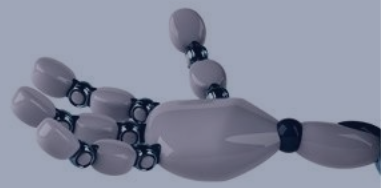
Isochronous Real-Time

- Easy to implement with 2 conditions:
  - A shared and extremely accurate clock to determine when the switches need to enter the IRT timeslice and when to return to normal Ethernet operation.
  - Additional circuitry in the network switch to buffer and hold any stray Ethernet traffic that is received on other ports during the IRT timeslice.
- IEEE 1588v2 - standard for shared clock - Precision Time Protocol (PTP)

Precision Transparent Clock Protocol (PTCP)

- PROFINET wrapper for Precision Time Protocol (PTP)
- Allows network switches to share a common real-time clock across the network
- Is also used to compute the delays inherent in both the network switches and the cabling between them down to the nanosecond

ROS2



Real-time in ROS. ROS2

Real-time in ROS

- no native support directly in the framework
- can be achieved by using third party software integrated in ROS

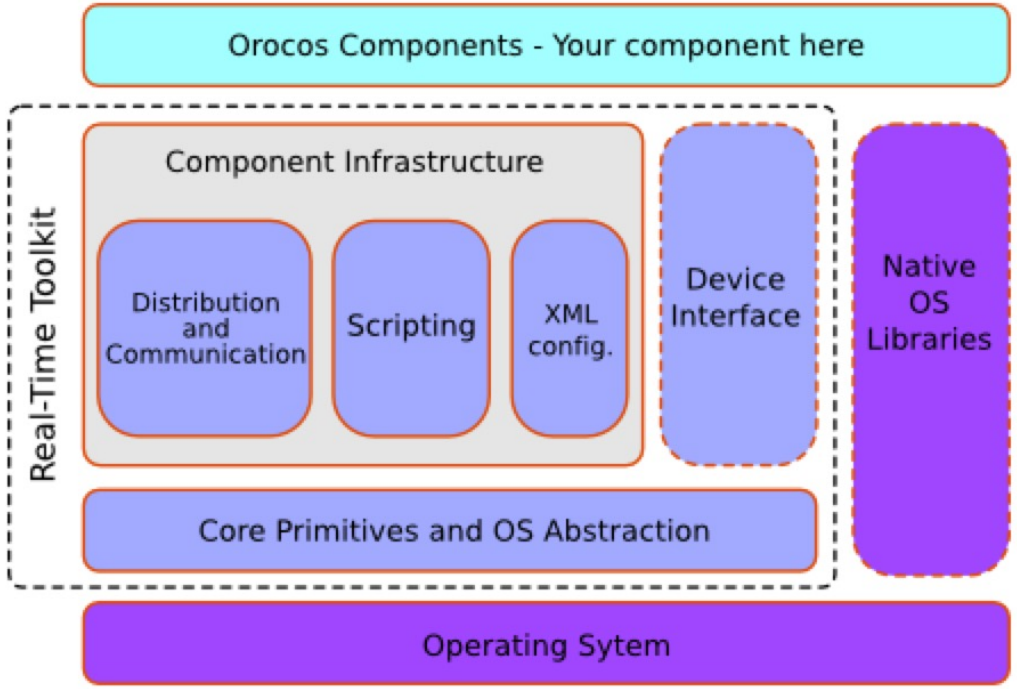
- **OpenRTM** - software platform based on RT Middleware standard
- all robotic technological elements (actuators and sensors) are RT-components (RTC)
- each RTC has 4 states **CREATED**, **INACTIVE**, **ACTIVE**, and **ERROR** with event-handlers attached to them
- several ROS packages: **openrtm**, **openrtm_ros_bridge**, **openrtm_aist**

https://openrtm.org/openrtm/ja/node/834

Real-time support

- **OROCOS Real-Time Toolkit (RTT)** - C++ framework, or "runtime", targeting the implementation of control systems
- allows application designers to build highly configurable and interactive component-based real-time control applications
- ROS package: **orocos_toolchain**

https://www.orocos.org/rtt



Real-time support

- YARP (Yet Another Robot Platform) - a set of libraries, protocols, and tools to keep modules and devices cleanly decoupled
- soft real-time parallel computation cluster
- can be integrated to ROS at different levels (topics, services, nameserver and node interfaces)

https://www.yarp.it/index.html

Introducing ROS2

- Real-time from day 1
- Standard node lifecycle state machine
- Best practices from existing frameworks:
  - microblx
  - OpenRTM
  - Orocos RTT
  - ros_control

https://design.ros2.org/

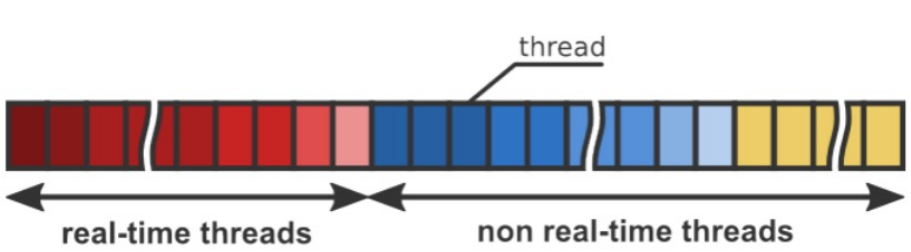
http://docs.ros2.org/eloquent/developer_overview.html#

Requirements

- Use an OS able to deliver the required determinism

OS	real-time	max latency (µs)
Linux	no	10^4
RT PREEMPT	soft	10^1 - 10^2
Xenomai	hard	10^1

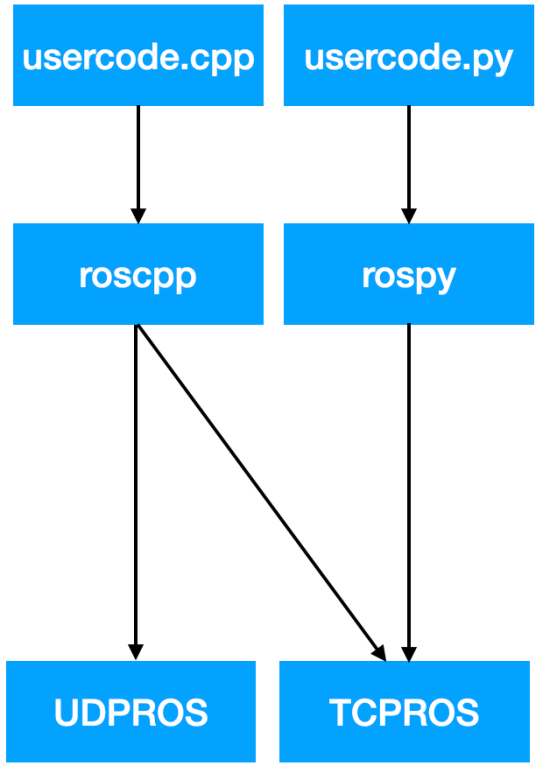
- Prioritize real-time threads in the scheduling policy



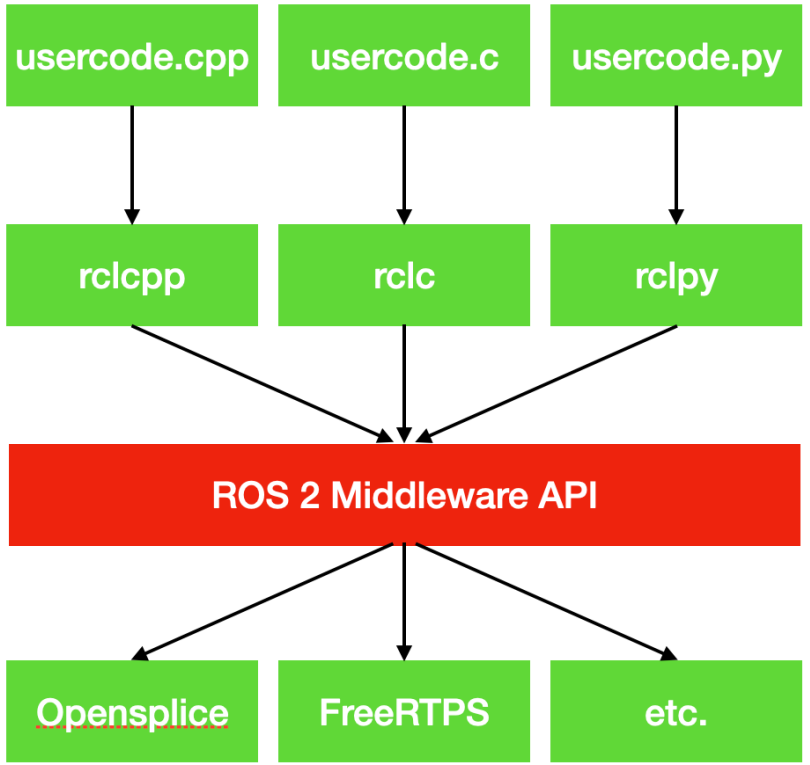
Best practices

- Avoid sources of non-determinism in real-time code
  - Memory allocation and management (e.g.: malloc, new)
  - Blocking synchronization primitives (e.g.: mutex)
  - Printing, logging (e.g.: printf, cout)
  - Network access (e.g.: TCP/IP)
  - Non real-time device drivers
  - Accessing the hard disk
  - Page faults

ROS1 vs ROS2



ROS 1
No native support for RT



ROS2
Native support for RT

ROS1 vs ROS2

	ROS 1	ROS 2
Officially Supported OS	ubuntu	ubuntu osx Win 10
C++ standard	C++ 3	C++ 11 C++ 14
Python version	2	3.5
Interfaces	Custom serialization Custom transport protocol Custom discovery	Abstract middleware interface (DDS)
Build systems	CMake	CMake Plain python packages Other build systems
Build	Non-isolated build n packages - 1 build context	Isolated build 1 package - 1 build context

	ROS 1	ROS 2
Topic namespace	Supported	Not supported due to DDS
ROS graph notifications	Pulled from master	Published by master
Actions	Supported	Not supported (yet)
Threading model	Single/Multi-threaded execution	More granular execution models
Launch files	XML	Python
Real-time support	External frameworks	Native

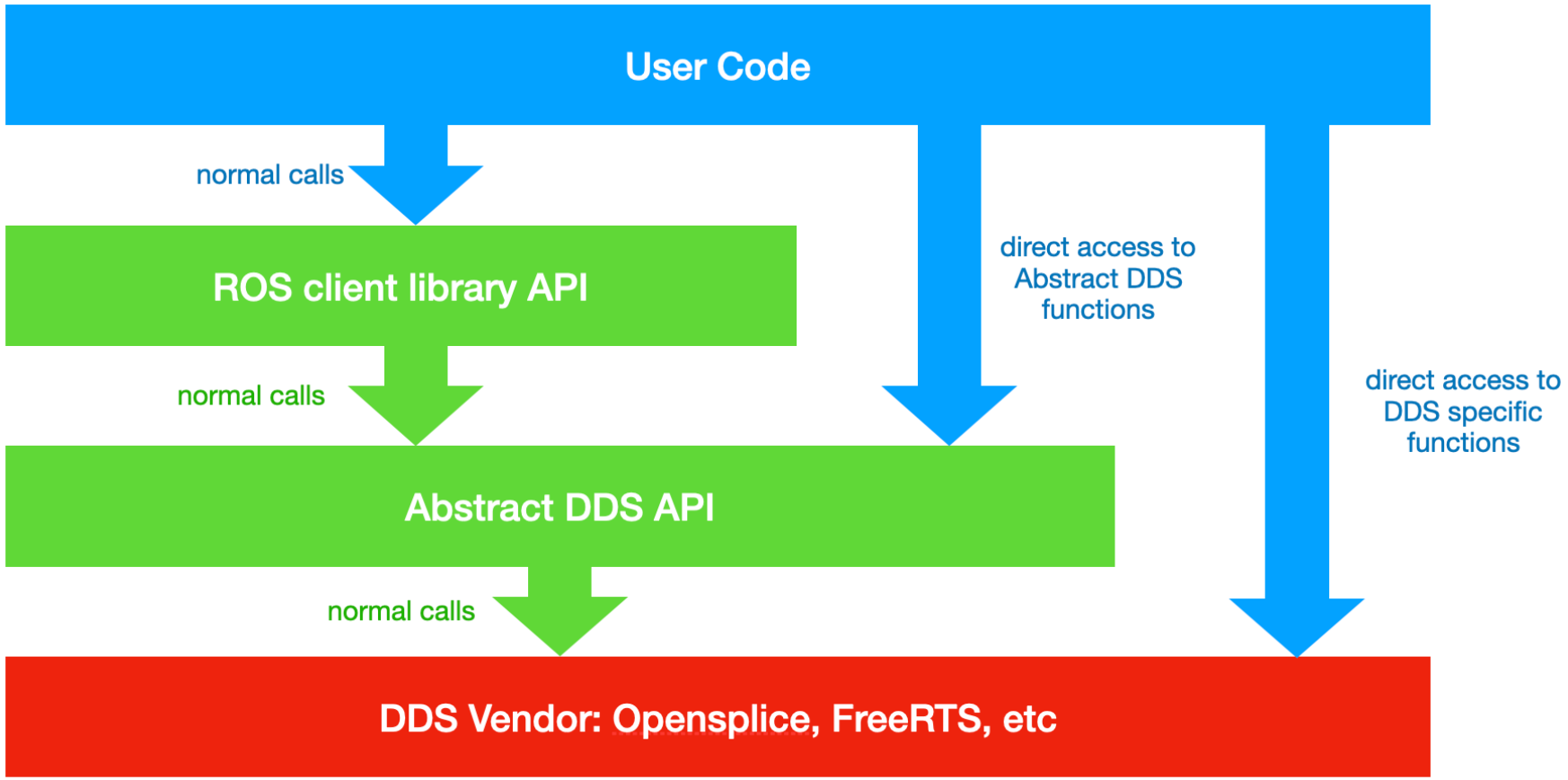
Data Distribution Service

- Data Distribution Service (DDS) for real-time systems is an Object Management Group (OMG) machine-to-machine (sometimes called middleware or connectivity framework) standard that aims to enable dependable, high-performance, interoperable, real-time, scalable data exchanges using a publish–subscribe pattern (DDS-RPC).

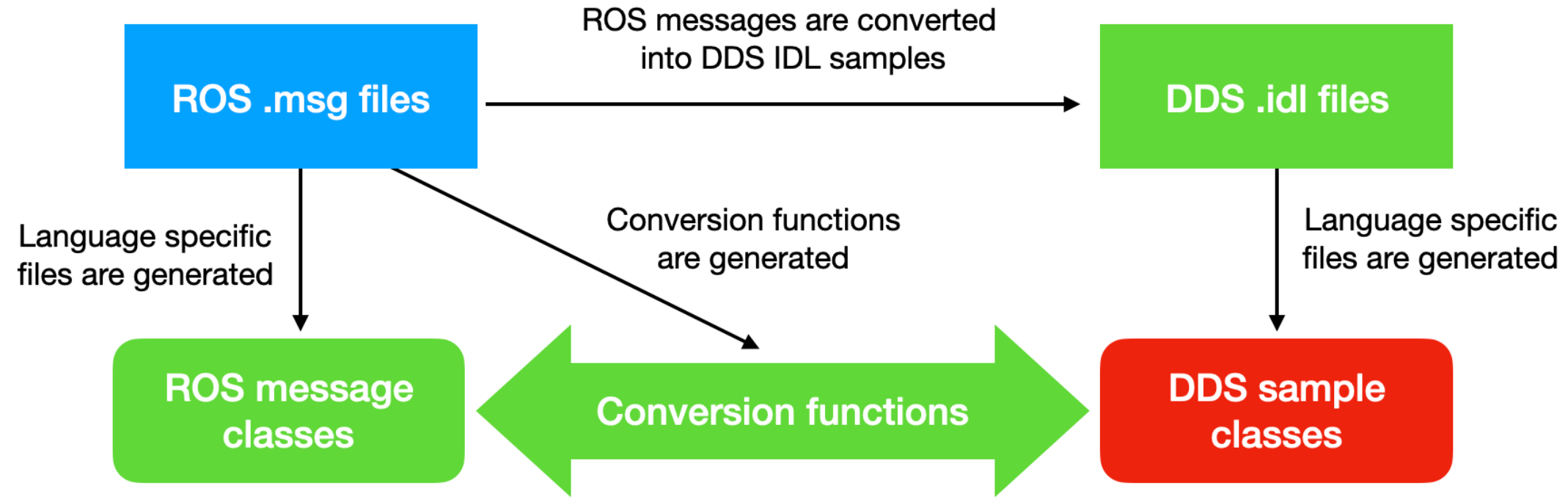
- uses Interface Description Language (IDL)

- has a distributed discovery system which allows any two nodes to communicate with each over without master intervention

ROS 2 DDS model



Messages


ROS .msg files
ROS messages are converted into DDS IDL samples
DDS .idl files
Language specific files are generated
Conversion functions are generated
Language specific files are generated
ROS message classes
Conversion functions
DDS sample classes

ROS2 RT implementation

initialization — **non real-time**

```
preallocate memory
....
```

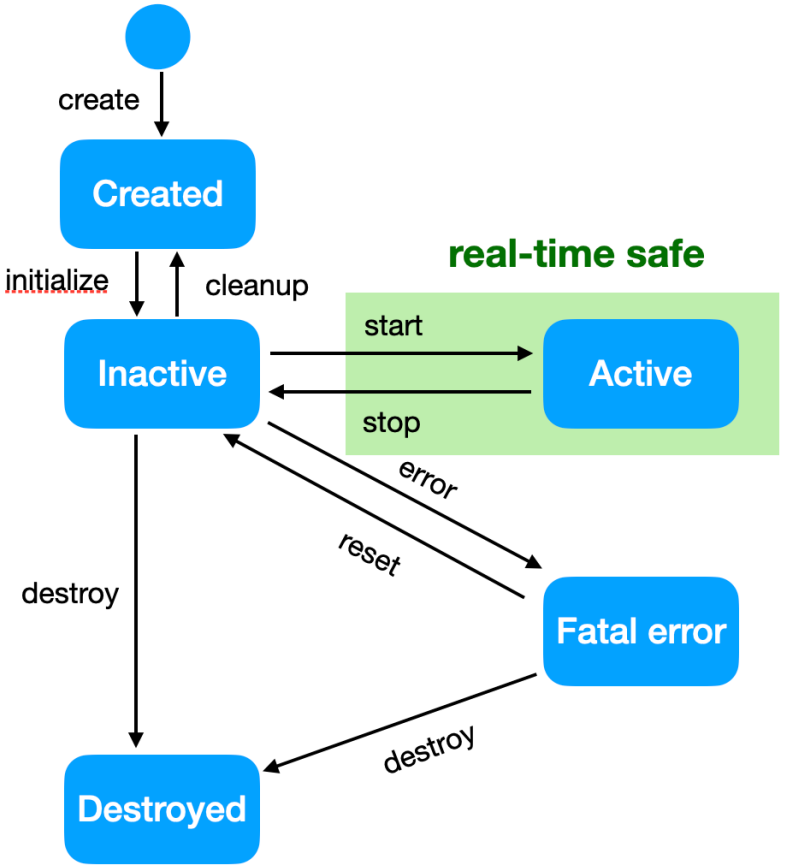
spin — **real-time**

```
rmw_wait(timeout)
{
    pass conditions to waitset
    wait (in DDS)
    wake-up if timed-out
}
do work if it came in
```

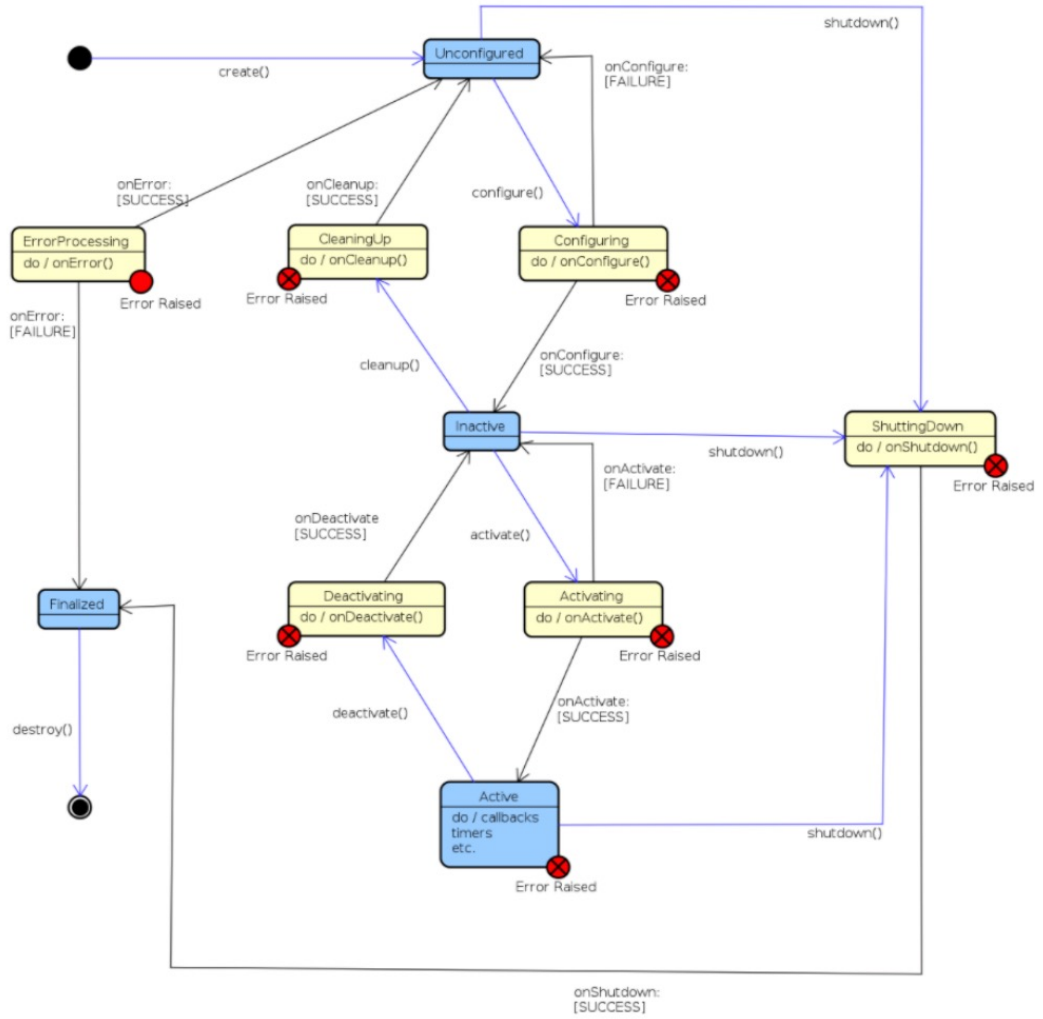
cleanup — **non real-time**

```
deallocate memory
....
```

Node life cycle



ROS2 Life cycle



https://design.ros2.org/articles/node_lifecycle.html

Benefits of managed life cycle

- Clear separation of real-time code path
- Greater control of ROS network
  - Ensure correct launch sequence
  - Online node restart / replace
- Better monitoring and supervision

ROS 2 Quality of Service

- QoS is defined is a QoS configuration struct

- Base QoS profile:
  - **History -** if DDS should store all or some messages in the message queue
    - **Keep all**: tells the DDS to keep all messages
    - **Keep last**: tells the DDS that only some last messages are stored
  - **Depth** - how many messages should be stored (Keep last)
  - **Reliability** - how reliable the DDS should be
    - **Best effort**: attempt to deliver all, but may lose some
    - **Reliable**: guarantee that all messages are delivered (may retry)
  - **Durability** - message persistence
    - **Transient local**: the publisher becomes responsible for persisting messages for “late-joining” subscribers
    - **Volatile**: no attempt is made to persist messages

ROS 2 Quality of Service

- Extended QoS profile:
  - **Deadline**: contract for the amount of time allowed between receiving (subscribers) or sending (publishers) messages
  - **Liveliness**: how nodes report that they are alive:
    - LIVELINESS_AUTOMATIC - uses the ROS layer
    - LIVELINESS_MANUAL_BY_NODE - publishing ***one message*** *on* ***one topic*** marks that node is alive for ***all subscribers***
    - LIVELINESS_MANUAL_BY_TOPIC - publishing ***one message*** on ***one topic*** marks that node is alive ***only to the topic's subscribers***
  - **Lifespan**: how long a message remains valid. Only valid messages are received (subscribers) or maintained in the topic history (subscribers)

ROS 2 Security

- ROS2 introduces several security properties, including encryption, authentication, and authorization (based on DDS-Security specifications)
- profiles are used based on a policy schema
- privileges supported at different levels:
  - *actions* - **call**, **execute**
  - *services* - **reply**, **request**
  - *topics* - **publish**, **subscribe**

https://design.ros2.org/articles/ros2_dds_security.html

https://design.ros2.org/articles/ros2_access_control_policies.html

Thank you

Question Time