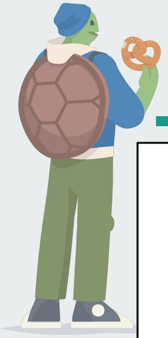
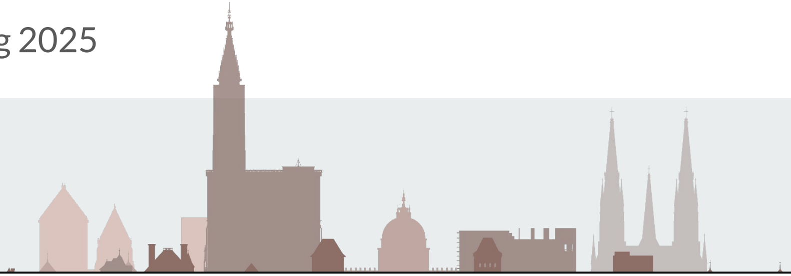
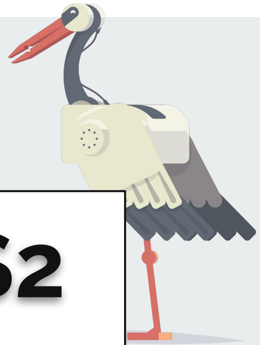






Introduction to the EtherCAT ROS2 driver and EtherCAT safety with Synapticon drives

Maciej BEDNARCZYK (Asterion Robotics)
Philippe ZANNE (ICube/IRIS/CNRS)
Manuel YGUEL (ICube/IRIS/Unistra)

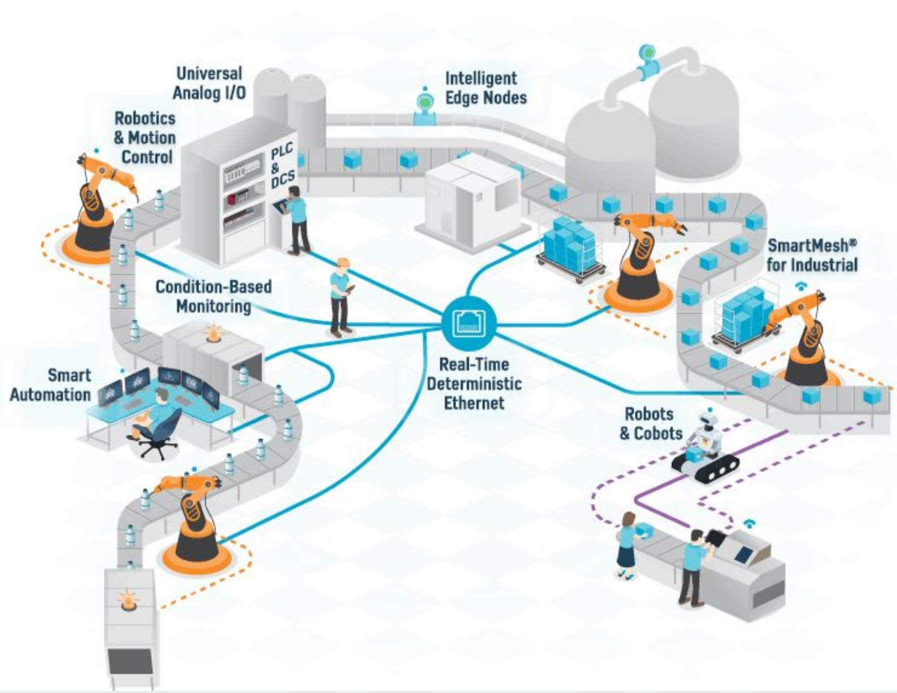
Florian WEIßHARDT (Synapticon)
Abhilash INGALE (Synapticon)

iCUBE cnrs Université de Strasbourg

asterion robotics

synapticon™ INTEGRATED MOTION

What is EtherCAT, and why using it for Robots ?


Universal Analog I/O
Intelligent Edge Nodes
Robotics & Motion Control
PLC & DCS
SmartMesh® for Industrial
Condition-Based Monitoring
Smart Automation
Real-Time Deterministic Ethernet
Robots & Cobots

EtherCAT : the industrial fieldbus

- **Goal**: interconnecting multiple systems with a single type of interface

EtherCAT : the industrial fieldbus

- ♺ **Goal**: interconnecting multiple systems with a single type of interface
  - ▻ interoperability

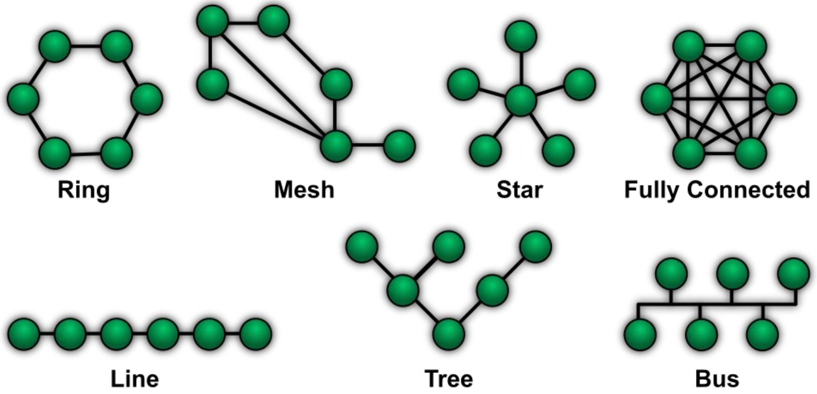
EtherCAT : the industrial fieldbus

♺ Goal: interconnecting multiple systems with a single type of interface

- ▻ interoperability
- ▻ reducing cost and time to production (installation/maintenance)

EtherCAT : the industrial fieldbus

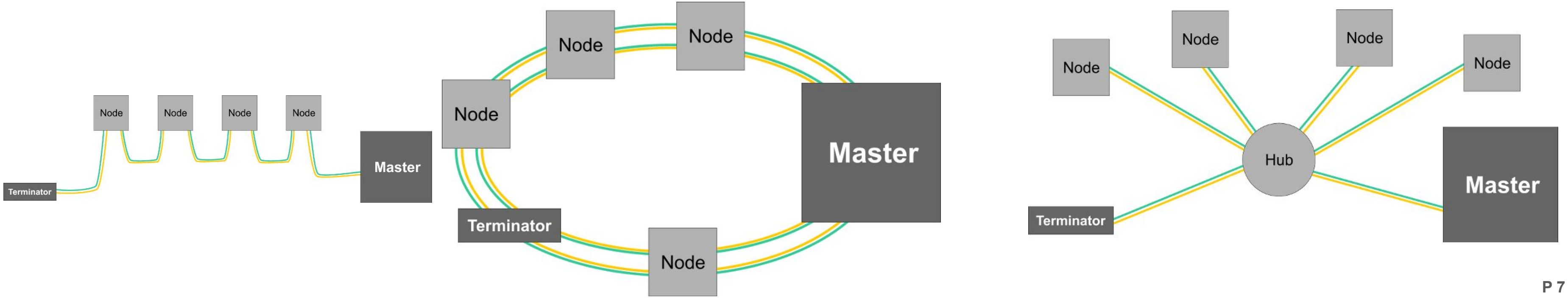
- **Goal**: interconnecting multiple systems with a single type of interface
  - interoperability
  - reducing cost and time to production (installation/maintenance)
- **User constraints**:
  - connectivity depends upon mechanical constraints



The CAN bus alternative

- **Goal**: interconnecting multiple systems with a single type of interface

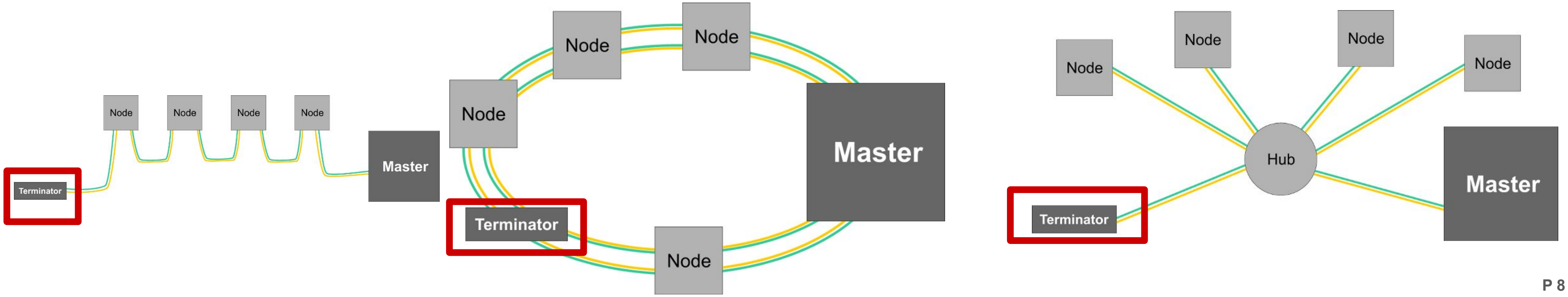
- Industry standard alternative: **the CAN bus** (Controler Area Network bus):



The CAN bus alternative

- **Goal**: interconnecting multiple systems with a single type of interface

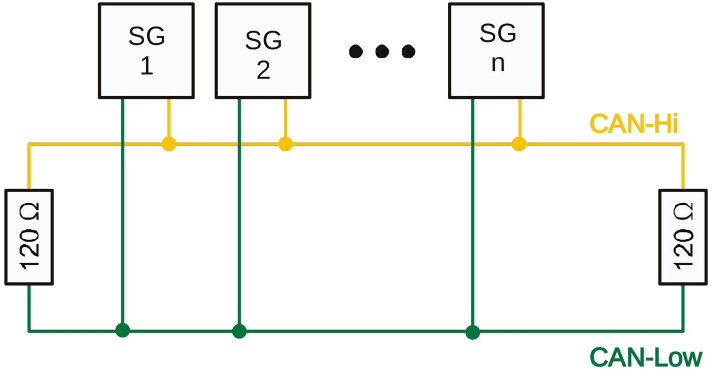
- Industry standard alternative: **the CAN bus** (Controler Area Network bus):



The CAN bus alternative

- **Goal**: interconnecting multiple systems with a single type of interface

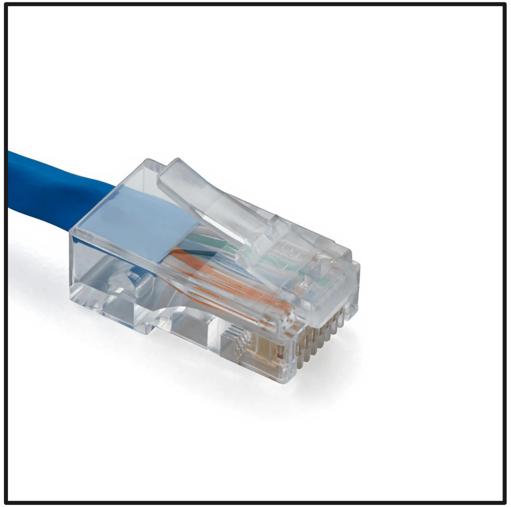
- Industry standard alternative: **the CAN bus** (Controler Area Network bus):



EtherCAT a better fieldbus solution

- ♺ **Goal**: interconnecting multiple systems with a single type of interface

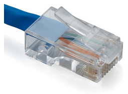
- ♺ **Ethercat is a field bus**:
  1. a physical interface
     - with standard connectors



EtherCAT a better fieldbus solution

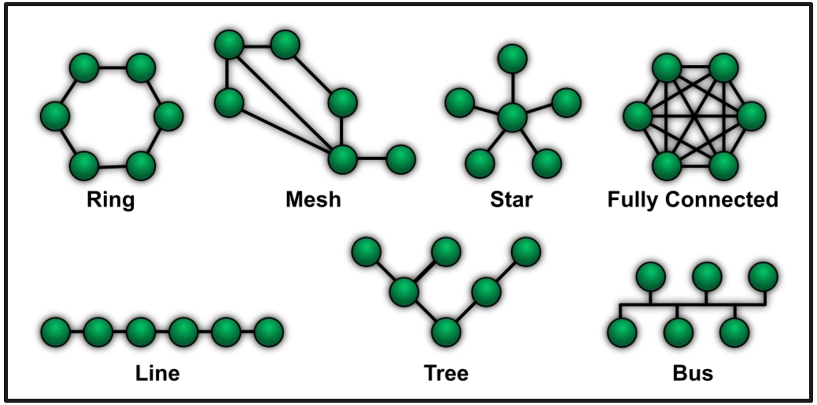
EtherCAT®

♺ **Goal**: interconnecting multiple systems with a single type of interface



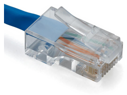
♺ **Ethercat is a field bus**:

1. a physical interface
   - with standard connectors
   - any topology

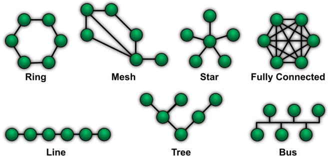


EtherCAT a better fieldbus solution

- **Goal**: interconnecting multiple systems with a single type of interface

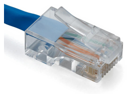


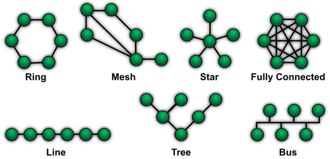
- **Ethercat is a field bus**:
  1. a physical interface
     - with standard connectors
     - any topology
     - allowing cable redundancy



EtherCAT a better fieldbus solution

- **Goal**: interconnecting multiple systems with a single type of interface

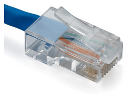




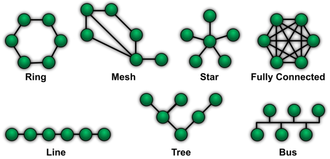
- **Ethercat is a field bus**:
  1. a physical interface
     - with standard connectors
     - any topology
     - allowing cable redundancy
     - up to 100m of cable length

EtherCAT a better fieldbus solution

- **Goal**: interconnecting multiple systems with a single type of interface



- **Ethercat is a field bus**:
  1. a physical interface
     - with standard connectors
     - any topology
     - allowing cable redundancy
     - up to 100m of cable length,
     - 100Mbit/s, Full Duplex



SIMPLE, PRACTICAL, FAST

EtherCAT a better fieldbus solution

EtherCAT®

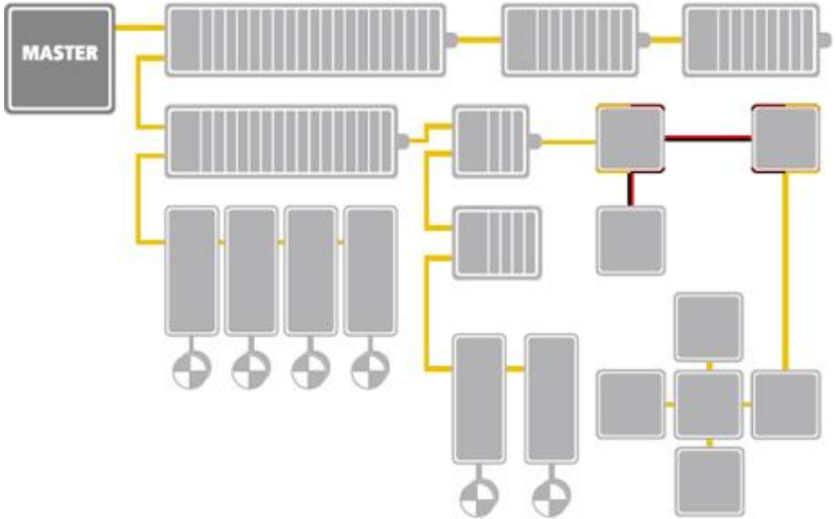
♺ **Goal**: interconnecting multiple systems with
a single type of interface

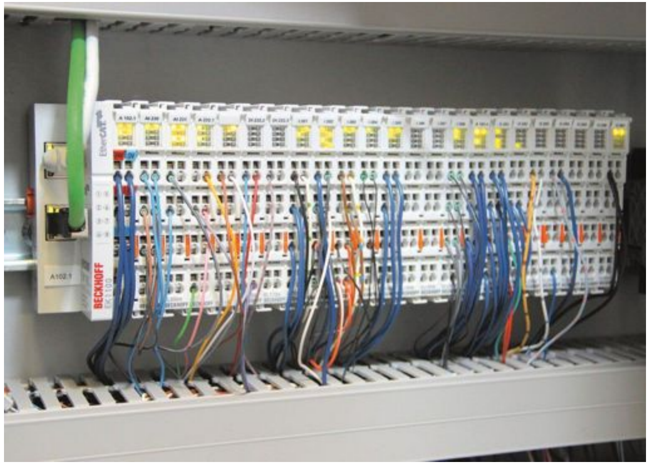
♺ **Ethercat is a field bus**:

1. a physical interface,
2. a link interface
   - ■ Standard IEEE 802.3, CEI 61158
   - ■ Master/Slave system
   - ■ Cycle time of 100µs and jitter < 1 µs
   - ■ Distributed clock system for synchronization
   - ■ can encapsulate ethernet com.

SIMPLE, PRACTICAL, FAST

EtherCAT : Wiring / Installation

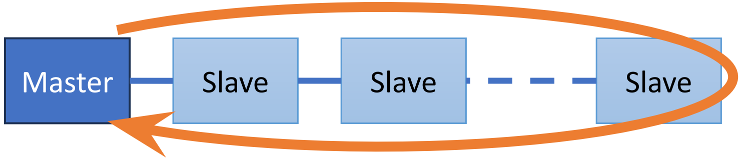




- Flexible topology: line, star, daisy-chain, tree, …
- Possibility to connect up to 65535 modules on a segment
- Connexion with 100 BASE-TX cables up to 100m and above using optical fiber (100BASE-FX)

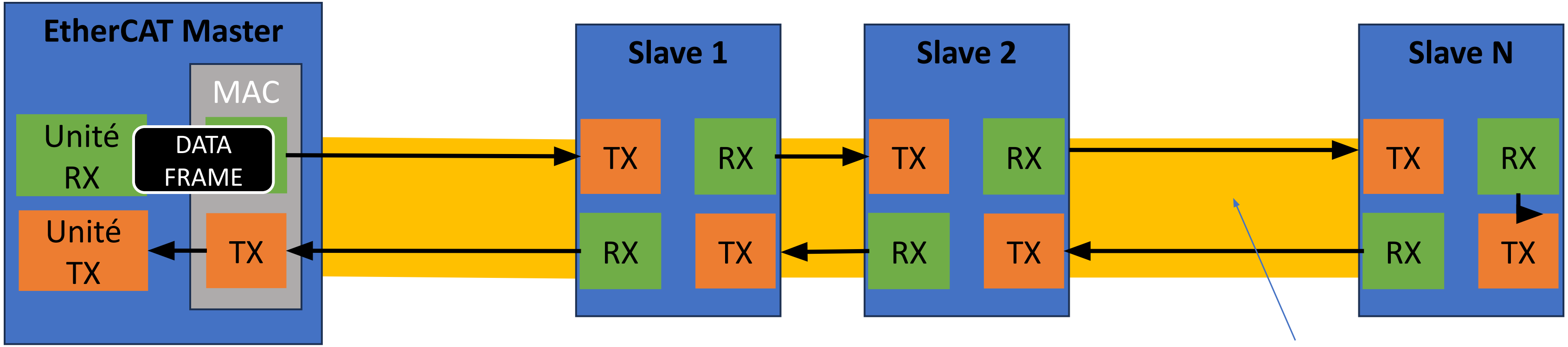
EtherCAT : Principles

- ❖ The EtherCAT protocol operates on the basis of **on-the-fly data exchange**
- ❖ The master send a datagram that traverses all slave modules **in a virtual ring**.



Up to 65535 slaves

Data flow

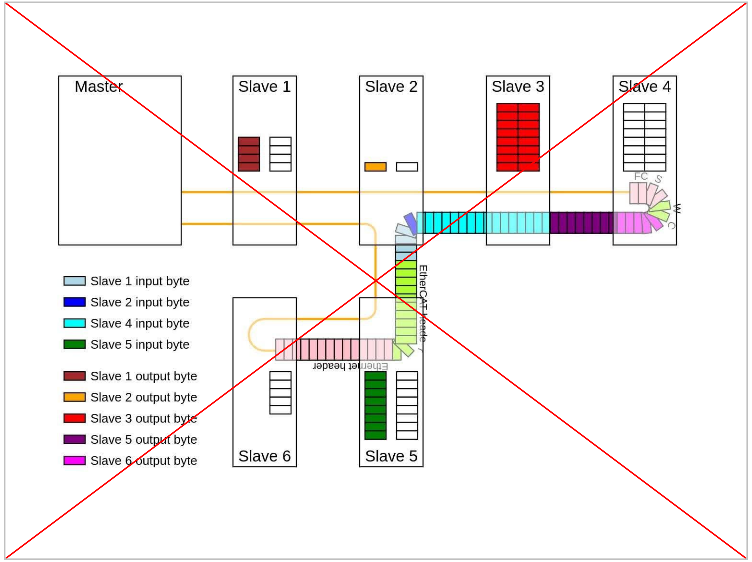


Cat5 cables

EtherCAT : Principles

Ethernet for **C**ontrol **A**utomation **T**echnology (Beckhoff)

ISO/OSI Layer		EtherCAT	
Host layers	7. Application	–	● Cyclic Data Exchange ● Mailbox Acyclic Data Access
	6. Presentation	–	
	5. Session	–	
	4. Transport	TCP	
Media layers	3. Network	IP	
	2. Data link	● Mailbox/Buffer Handling ● Process Data Mapping ● Extreme Fast Auto-Forwarder	
		Ethernet MAC	
	1. Physical	100BASE-TX, 100BASE-FX	



credits: Wikipedia

EtherCAT : Principles

Cyclic Data Exchange vs Mailbox Acyclic Data Access

📦 PDO (Cyclic Data Exchange)

🔹 **PDO — Process Data Objects**

- **Cyclic**, real-time communication
- **Fast**, sent every EtherCAT cycle
- **No protocol overhead** → mapped directly into frames
- Ideal for **sensor readings, motor commands, I/O data**
- **Master** ↔ **Slave exchange** happens *automatically*
- Not suitable for large or occasional data

📫 SDO — Service Data Objects

- **Acyclic**, on-demand communication
- Used for **configuration**, **parameters**, and **diagnostics**
- Higher overhead (CoE mailbox protocol)
- Much **slower**, not deterministic
- Ideal for setting **control gains, limits, device parameters**
- Not used in the fast control loop

🧭 Quick Summary

- **PDO = fast, cyclic, real-time I/O**
- **SDO = slow, acyclic, configuration & diagnostics**

Interacting with the EtherCAT Bus (CLI Tools)

Interacting with the bus using command-line utilities

- Etherlab provides several `ethercat` tools
- Useful for diagnostics, configuration, and inspection
- Commands operate directly on the **active EtherCAT master**

Check Which Master Is Running

Check active EtherCAT master

```
ethercat master
```

- Shows which master is currently in control
- Useful when multiple masters or services are installed

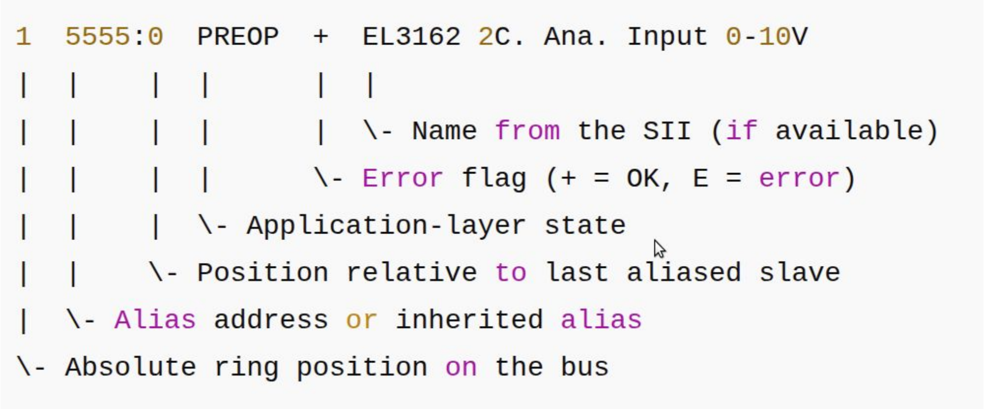
Interacting with the EtherCAT Bus (CLI Tools)

Get Slave Information

List slaves with detailed info

```
ethercat slaves --verbose
```

Interpreting the address & position fields



```
1  5555:0  PREOP  +  EL3162 2C. Ana. Input 0-10V
|  |    |  |      |  |
|  |    |  |      |  \- Name from the SII (if available)
|  |    |  |      \- Error flag (+ = OK, E = error)
|  |    |  \- Application-layer state
|  |    \- Position relative to last aliased slave
|  \- Alias address or inherited alias
\- Absolute ring position on the bus
```

Set an Alias for a Slave

Assign alias to slave at position 1

```
ethercat alias --position=1 17
```

- Alias range: **1–65535**
- `0` means *no alias*
- Used for persistent addressing rather than relying on physical ring order

Interacting with the EtherCAT Bus (CLI Tools)

EtherCAT®

Read ESI Equivalent from the Device (SII)

Read SII (Slave Information Interface)

```
ethercat sii_read -v --position 1
```

- Shows detailed hardware information
- Equivalent to inspecting the ESI XML stored inside the device
- (Ref: Etherlab docs, page 76)

Interacting with the EtherCAT Bus (CLI Tools)

List Available PDOs

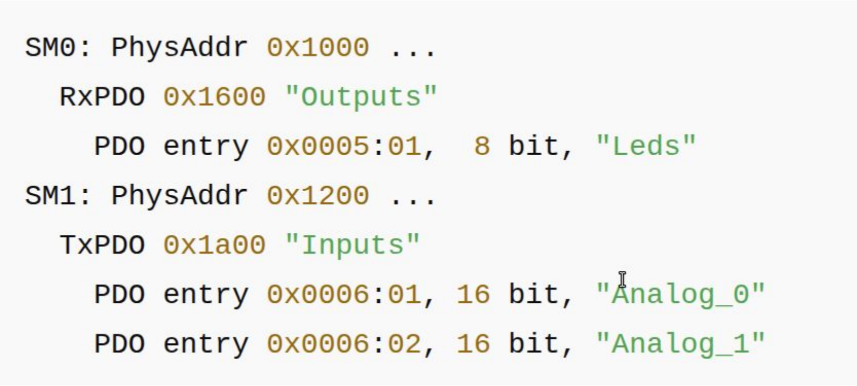
Get PDO mapping of a slave

```
ethercat pdos --position 0
```

or

```
ethercat pdos -p 0
```

Example Output:



```
SM0: PhysAddr 0x1000 ...
  RxPDO 0x1600 "Outputs"
    PDO entry 0x0005:01,  8 bit, "Leds"
SM1: PhysAddr 0x1200 ...
  TxPDO 0x1a00 "Inputs"
    PDO entry 0x0006:01, 16 bit, "Analog_0"
    PDO entry 0x0006:02, 16 bit, "Analog_1"
```

- Useful to configure PDO mappings in applications
- Equivalent to looking inside the ESI's `<RxPdo>` and `<TxPdo>` sections

Interacting with the EtherCAT Bus (CLI Tools)

Read an SDO Value

Upload an SDO

```
ethercat upload --position 0 --type int32 0x6063 0
```

- Reads object **0x6063:00** (int32) from slave 0
- Accepts decimal, octal (`0`), or hex (`0x`) inputs
- For configuration or diagnostics (non-cyclic communication)

Interacting with the EtherCAT Bus (CLI Tools)

Write an SDO Value (Download an SDO)

```
ethercat download --position 0 --type int8 0x6060 0 8
```

- Writes **mode of operation = 8** into **0x6060:00** (int8)
- Same numeric formatting rules (decimal / octal / hex)
- Used for configuration, not for cyclic real-time updates

Writing a *negative* value (important syntax!)

When writing negative integers, you **must** insert `--` to prevent the negative value from being interpreted as a command-line option.

Example: write −3 to SDO 0x3000:01

```
ethercat download --position 0 --type int16 0x3000 1 -- -3
```

Interacting with the EtherCAT Bus (CLI Tools)

Write an SDO Value (Download an SDO)

```
ethercat download --position 0 --type int8 0x6060 0 8
```

- Writes **mode of operation = 8** into **0x6060:00** (int8)
- Same numeric formatting rules (decimal / octal / hex)
- Used for configuration, not for cyclic real-time updates

Writing a *negative* value (important syntax!)

When writing negative integers, you **must** insert `--`
to prevent the negative value from being interpreted as a command-line option.

Example: write −3 to SDO 0x3000:01

```
ethercat download --position 0 --type int16 0x3000 1 -- -3
```

Why `--` is required?

- Without it, `-3` would be parsed as an **invalid option** (`-3`)
- With `--`, everything after is treated as a **literal argument** (the value)

Interacting with the EtherCAT Bus (CLI Tools)

Display the Current EtherCAT Configuration

Show full bus configuration

```
ethercat cstruct
```

Generate C code for a specific slave

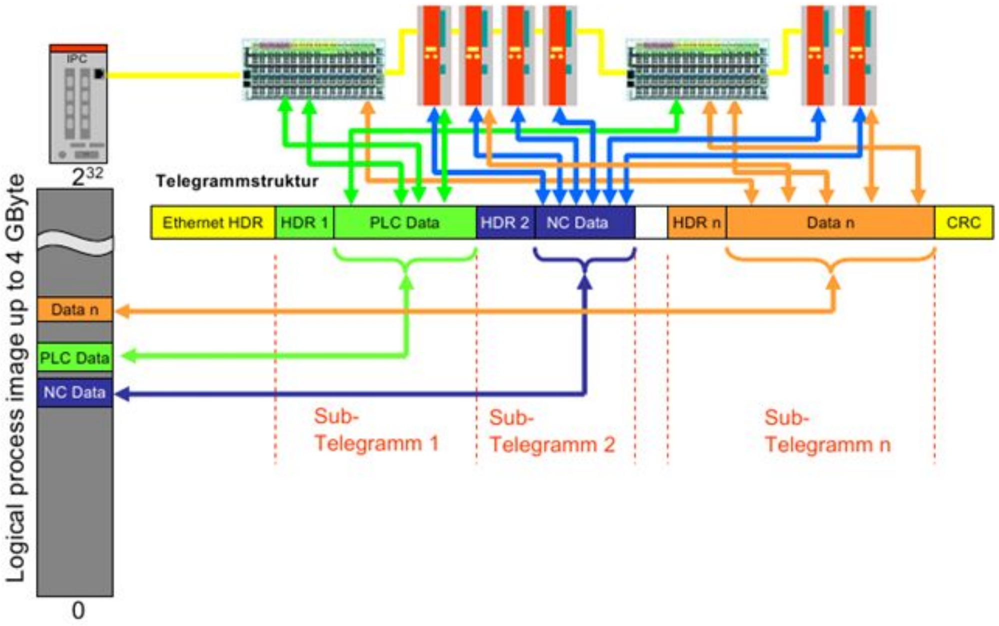
```
ethercat cstruct --position <pos>
```

- Produces C structures for use with `ecrt_slave_config_pdos()`
- Helpful when integrating EtherCAT into a custom C/C++ master application

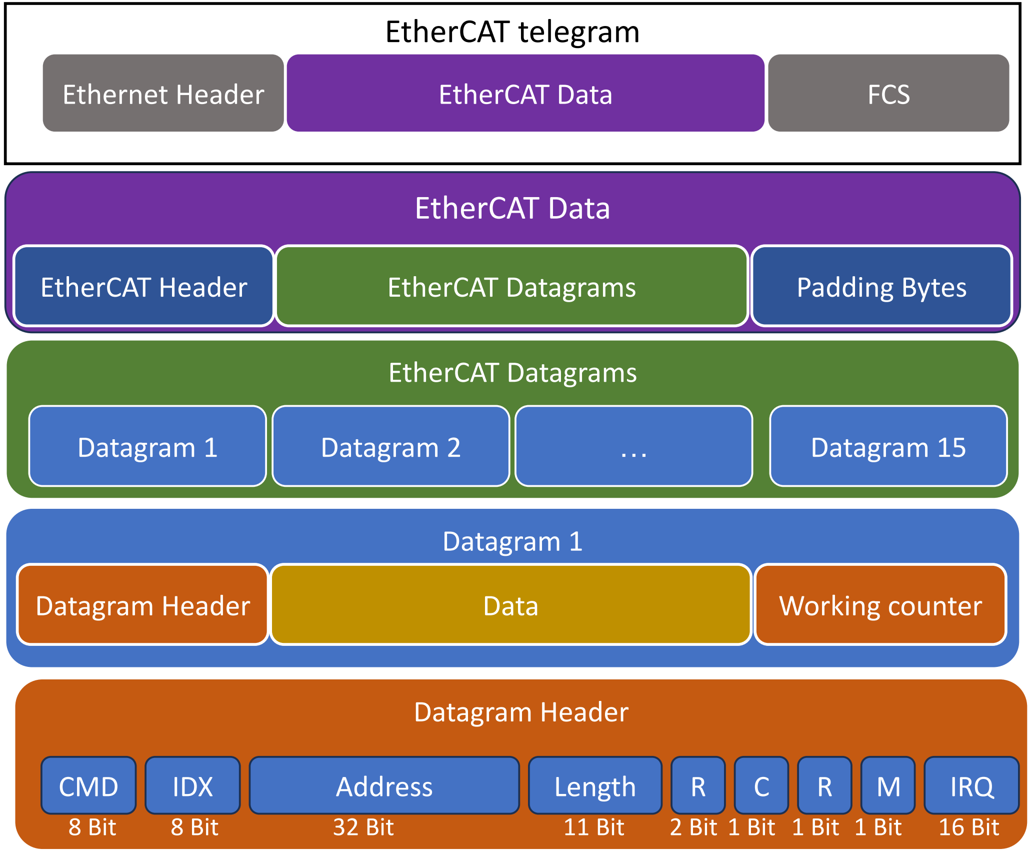
EtherCAT : Cyclic Data Exchange Principles

Operation

- The master sends a telegram across the entire network, passing through each slave.
- Each slave takes the data ‘**on the fly**’
- Then the telegram is forwarded to the next slave
- The last node or slave completes the frame and sends it back to the master through the network
- The exchange of input and output data is performed within a few nanoseconds by a specialized circuit (EtherCAT ASIC)
- The processing of the telegrams is carried out within the hardware and is therefore independent of the microprocessor response times



EtherCAT Frames



EtherCAT telegram

Ethernet Header | EtherCAT Data | FCS

EtherCAT Data

EtherCAT Header | EtherCAT Datagrams | Padding Bytes

EtherCAT Datagrams

Datagram 1 | Datagram 2 | … | Datagram 15

Datagram 1

Datagram Header | Data | Working counter

Datagram Header

CMD	IDX	Address	Length	R	C	R	M	IRQ
8 Bit	8 Bit	32 Bit	11 Bit	2 Bit	1 Bit	1 Bit	1 Bit	16 Bit

The EtherCAT frame (or telegram) is composed of an Ethernet header, EtherCAT data, and a Frame Check Sequence (validation tool)

The EtherCAT Data block is composed of an EtherCAT header, the EtherCAT datagrams, and padding bytes (used to fill the frame if the Ethernet frame is smaller than 64 bytes)

The EtherCAT datagram block is composed of up to 15 datagrams

A datagram is composed of a header, data that can be read or written, and a working counter

EtherCAT : the Working Counter

The **Working Counter (WKC)** is a key mechanism used to ensure **data integrity, correct addressing, and proper execution of read/write operations** during each EtherCAT cycle.

✅ **How it works**

- Each EtherCAT datagram contains commands such as ***read***, ***write***, or ***read-write*** requests.
- As the frame travels “on the fly” through the slave devices, **each slave increments the Working Counter** when it successfully processes the command addressed to it.
- The master then checks the final WKC value when the frame returns.

Operation	Increment
Read from a slave	+1
Write to a slave	+1
Read-Write operation	+2

The Working Counter helps the master **verify** that:

- The **correct number** of slaves participated in the operation.
- All addressed slaves **received** and **handled** their portion of the datagram.
- No **communication errors** occurred (e.g., lost frames, wiring faults, wrong configuration)

EtherCAT : the working counter

Example:

Suppose a datagram targets **5 slaves** and performs a **read-write** operation on each one.

- **Read-write** increments WKC by **2** per slave
  (1 for read, 1 for write)

So the **expected WKC** is:

```
WKC_expected = 5 slaves × 2 = 10
```

If the master receives:

- **WKC = 10** → All slaves responded correctly.
- **WKC = 8** → One slave did not process the request.
- **WKC = 0** → No device responded (cable unplugged, wrong address, etc.).

EtherCAT : the industrial fieldbus

EtherCAT®

- **Goal**: interconnecting multiple systems with
  a single type of interface

- **Ethercat is a field bus**:
  1. a physical interface, **SIMPLE, PRACTICAL, FAST**
  2. a link interface, **Open Standard, robust, real time**

EtherCAT : the industrial fieldbus

♺ Goal: interconnecting multiple systems with
a single type of interface

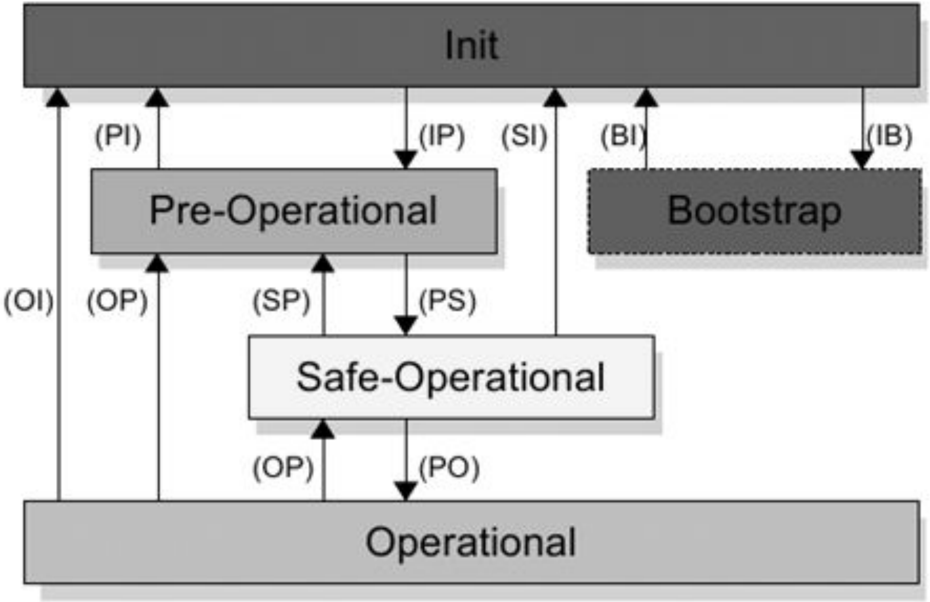
♺ Ethercat is a field bus:

1. a physical interface,
2. a link interface,
3. an application interface
   - Reuse CANOpen and Sercos application profils
   - Festo, OMRON, SMC, MAXON, KUKA, etc.

SIMPLE, PRACTICAL, FAST

Open Standard, robust, real time

EtherCAT has an application layer



EtherCAT Slave State Machine (ESM)

- **Each EtherCAT slave** implements the **EtherCAT State Machine (ESM)**
  → The **current state** determines which functions are available.

- **Defined slave states:**
  - **Init**
  - **Pre-Operational**
  - **Safe-Operational**
  - **Operational**
  - **Bootstrap** *(optional)*

- **State transitions require master action:**
  → For **each state change**, the **EtherCAT master** must send a **specific sequence of commands** to the slaves.

EtherCAT : the industrial fieldbus

- **Goal**: interconnecting multiple systems with a single type of interface

- **Ethercat is a field bus**:
  1. a physical interface,
  2. a link interface and
  3. an application interface

SIMPLE, PRACTICAL, FAST

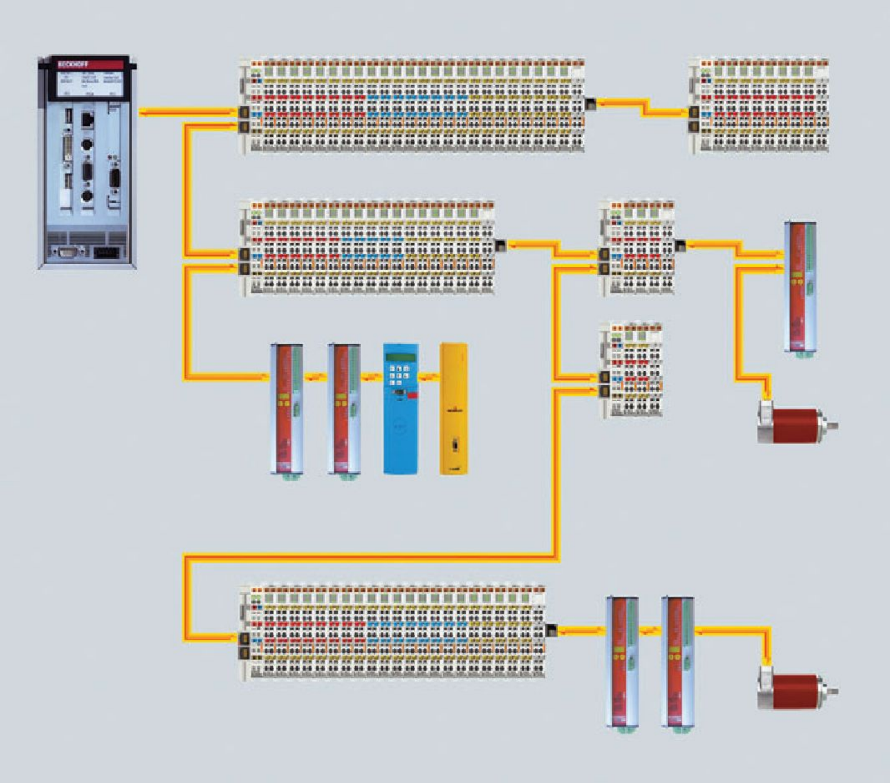
Open Standard, robust, real time

Optimized for automation and robotics

Developed by Beckhoff Automation since 2003

Maintained by the EtherCAT Technology Group since 2006

EtherCAT – Synchronisation

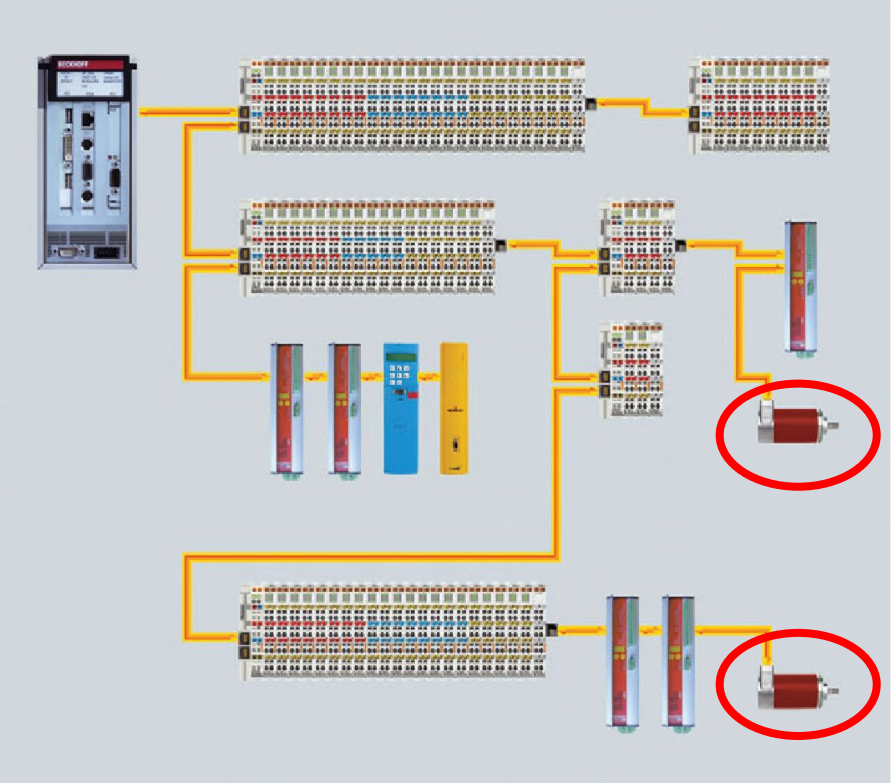


Hard requirements

- **Determinism** : the network must always react the same way to the same event.
- **Real-time control** : time synchronization must be ensured. The response delays of any part of the system must be bounded.

(Servoing, Medical, Parallel, …)

EtherCAT – Synchronisation



✨ Needs for Precise Synchronization

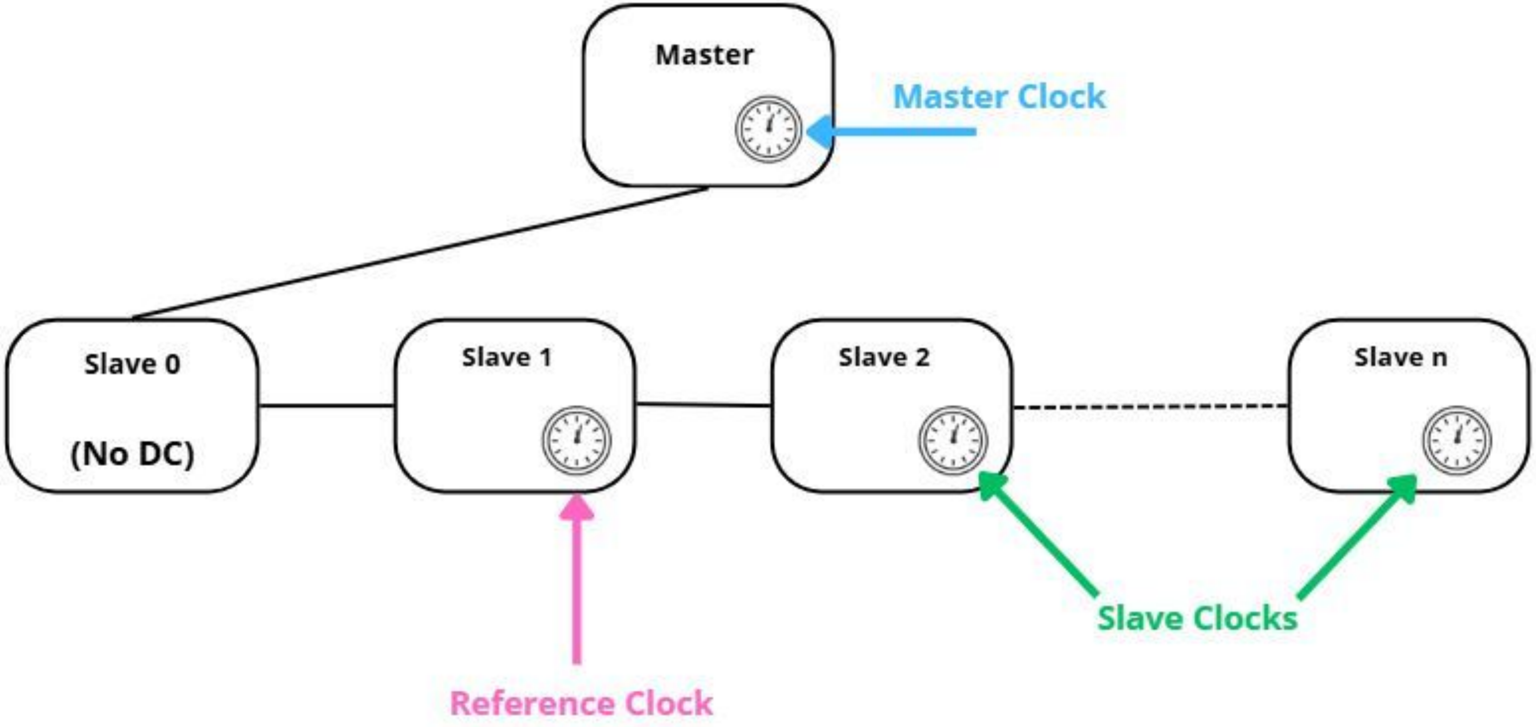
🌍 **Processes distributed in space**
→ **Require simultaneous and coordinated actions**
Very important for distributed systems

⏱️ **Precise and reliable synchronization**
→ Critical for correct system behavior

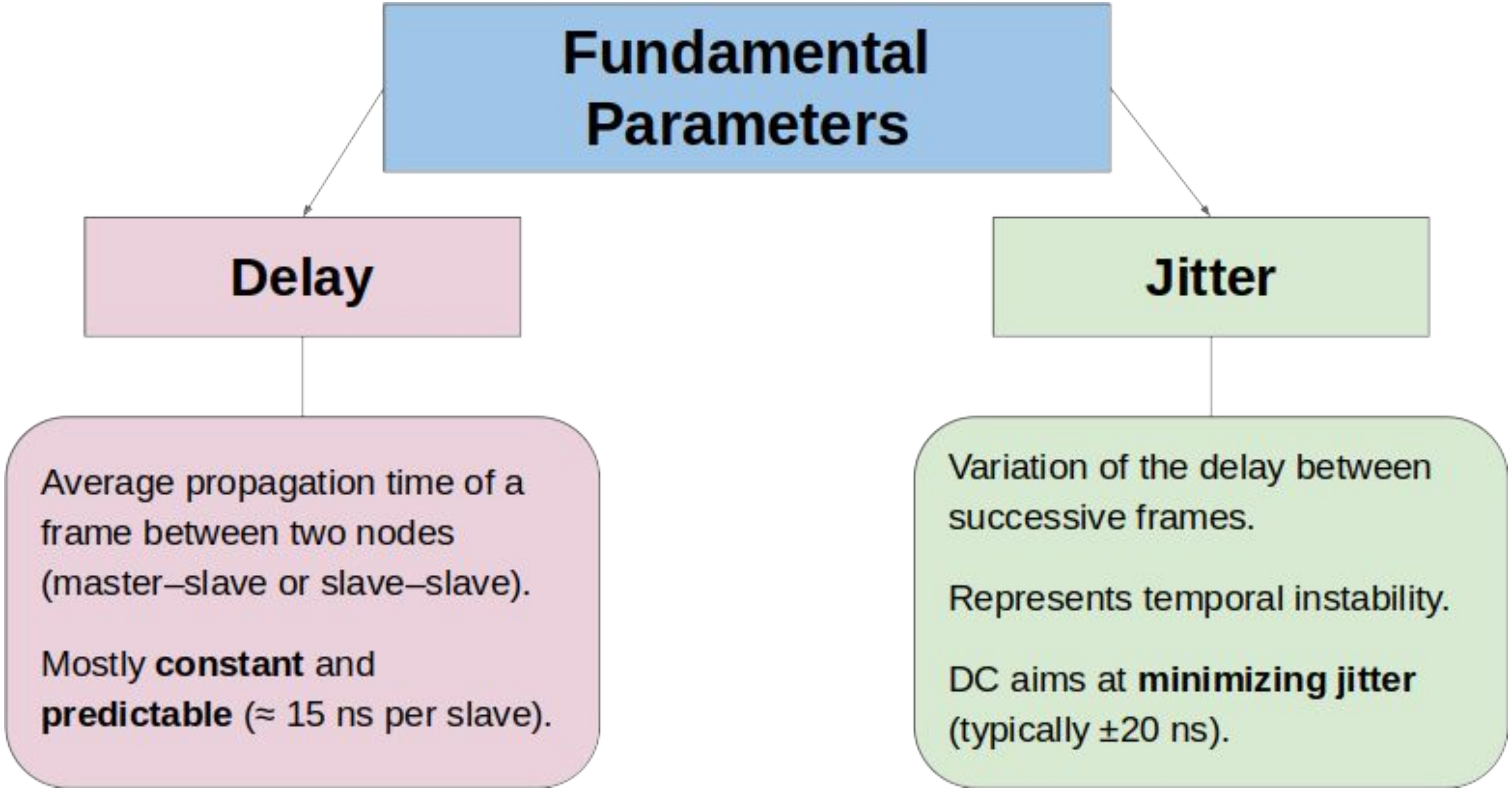
🤖 **Example: Robotic systems**

- Multiple **motorized axes**
- Must perform **coordinated movements**
- Accurate timing ensures **smooth, precise motion**

Distributed Clocks (DC) - Concept



Synchronization parameters
Delay and Jitter



Fundamental Parameters

Delay

Average propagation time of a frame between two nodes (master–slave or slave–slave).

Mostly **constant** and **predictable** (≈ 15 ns per slave).

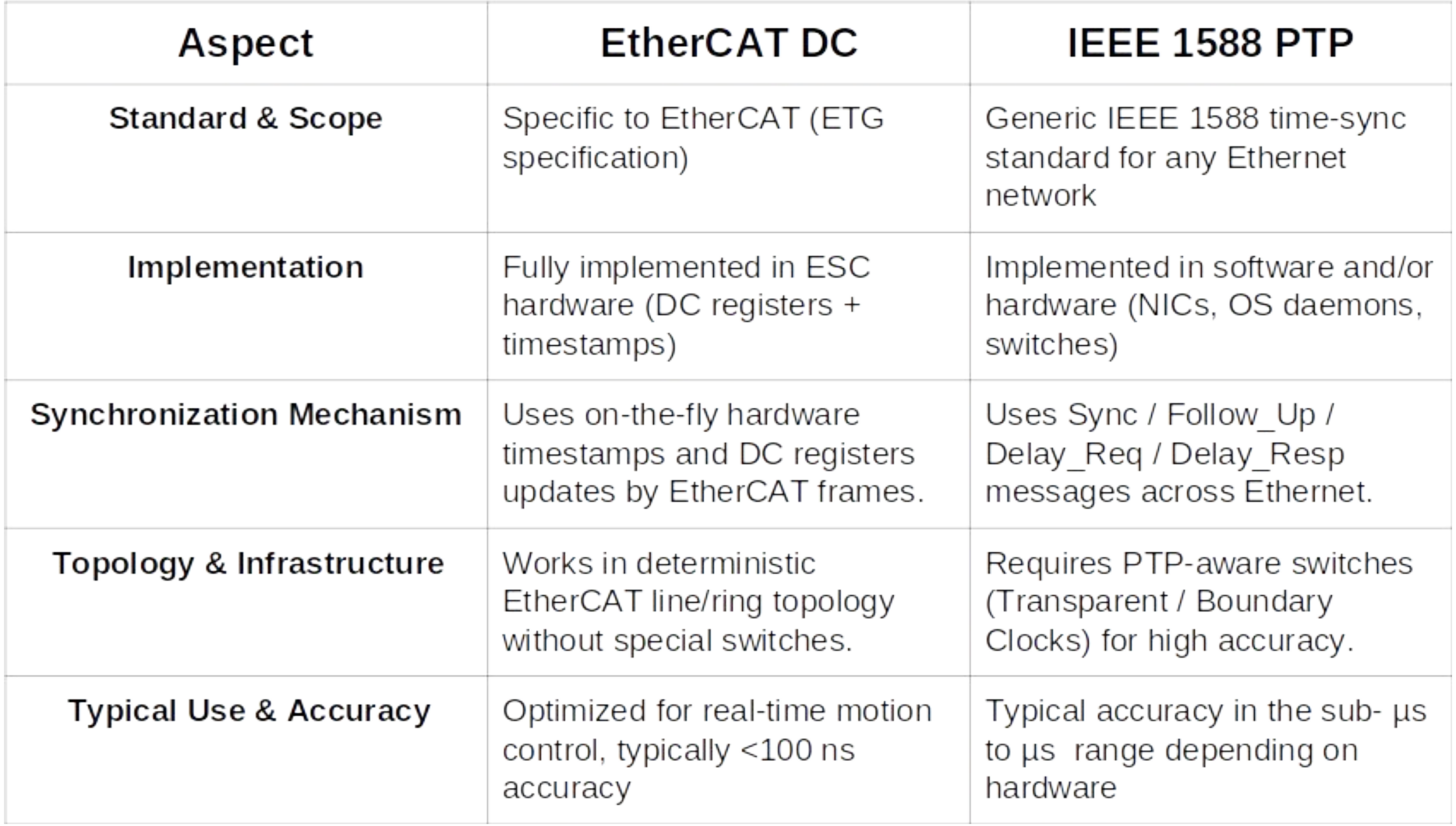
Jitter

Variation of the delay between successive frames.

Represents temporal instability.

DC aims at **minimizing jitter** (typically ±20 ns).

Synchronization parameters EtherCAT vs IEEE 1588 PTP



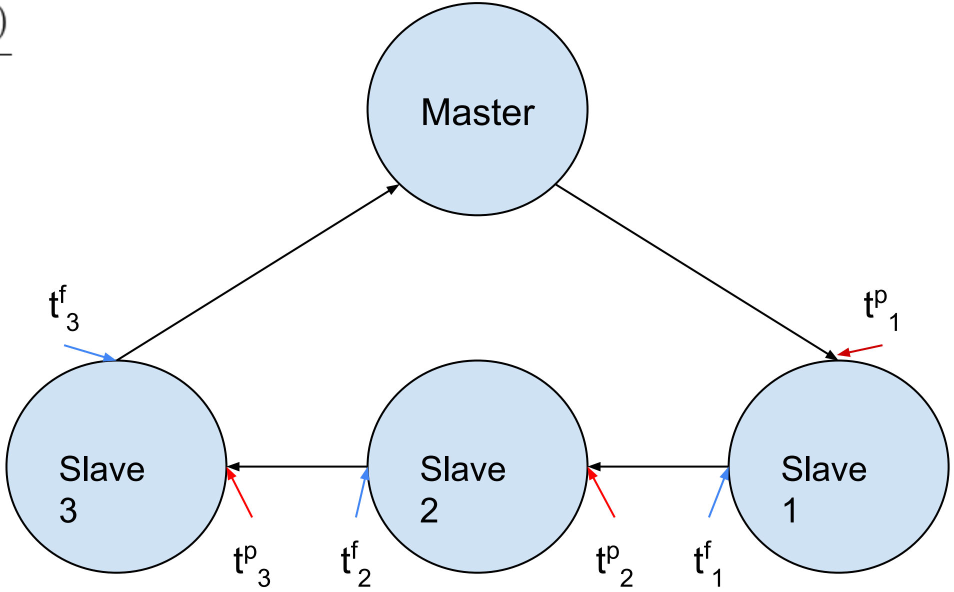
Aspect	EtherCAT DC	IEEE 1588 PTP
Standard & Scope	Specific to EtherCAT (ETG specification)	Generic IEEE 1588 time-sync standard for any Ethernet network
Implementation	Fully implemented in ESC hardware (DC registers + timestamps)	Implemented in software and/or hardware (NICs, OS daemons, switches)
Synchronization Mechanism	Uses on-the-fly hardware timestamps and DC registers updates by EtherCAT frames.	Uses Sync / Follow_Up / Delay_Req / Delay_Resp messages across Ethernet.
Topology & Infrastructure	Works in deterministic EtherCAT line/ring topology without special switches.	Requires PTP-aware switches (Transparent / Boundary Clocks) for high accuracy.
Typical Use & Accuracy	Optimized for real-time motion control, typically <100 ns accuracy	Typical accuracy in the sub- µs to µs range depending on hardware

DC mechanisms: delay measurement

- **<u>Phase 1 :</u>** Propagation delay measurement
  - Each slave writes timestamp in *Receive_Time_Port*
  - Master reads and collects each *Receive_Time_Port*
  - $t_{propagation\_i} = \frac{(t_i^p - t_{ref}^p) - (t_i^f - t_{ref}^f)}{2}$

t^p_i= Processing time

t^f_i= Forwarding time



DC mechanisms: Offset Calculation

- **<u>Phase 2 :</u>** Offset compensation

    1. Master reads the local time of each slave
    2. Computation : $t_{offset} = t_{sys\_time\_i} - (t_{sys\_ref} - t_{propogation\_i})$
    3. Master writes the value in *System_Time_Offset* register

⇒ This offset precisely quantifies the time-phase difference between the hardware clocks.

The master then transmits a positive or negative offset correction to each slave, which is directly applied to the ESC’s System Time register.

DC mechanisms: Dynamic Compensation Digital PLL & Quartz Drift

- **<u>Phase 3 :</u>** Drift Compensation

Local time can be modelled as : $t_{slave-i}(t) = \frac{f_i}{f_{ref}} \times (t - t_0) + \Delta_i$

Where:

- fi: actual oscillator frequency of slave *i*
- f_ref : reference clock frequency
- t0 : last synchronization instant
- Δi : applied offset correction

The master periodically estimates the drift, $drift_i = \frac{d\Delta_i}{dt}$

And adjusts the effective frequency of the local System Time to compensate for the drift,

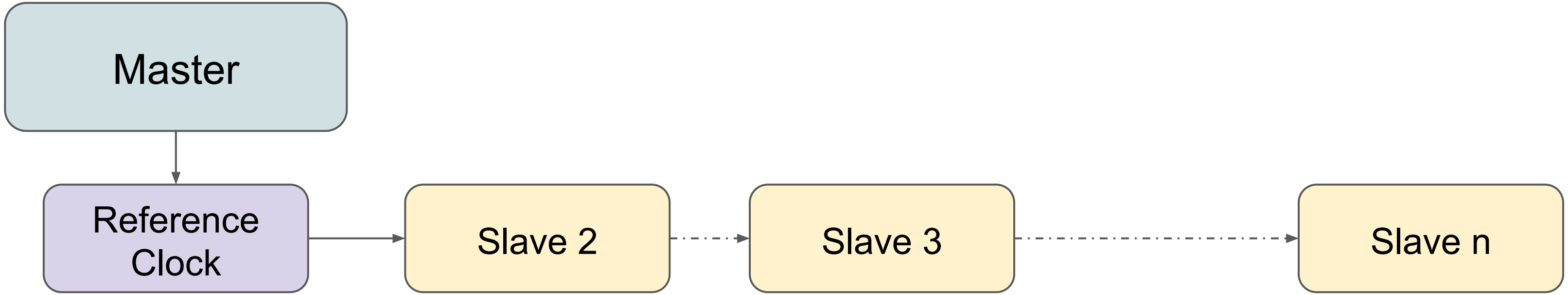
$$f'_i = f_i - K_p \times drift_i$$

with a small proportional gain *Kp* for a time-domain servo-type control.

DC mechanisms: Cascaded Synchronization

⏱️ **EtherCAT Synchronization** uses a **cascaded, hop-by-hop mechanism**.

- The **master first synchronizes the Reference Clock**.
- Each slave then **aligns its local clock** to the **previous slave** in the chain.
- This **local correction at every hop**:
  - minimizes phase error,
  - prevents error accumulation along the network.



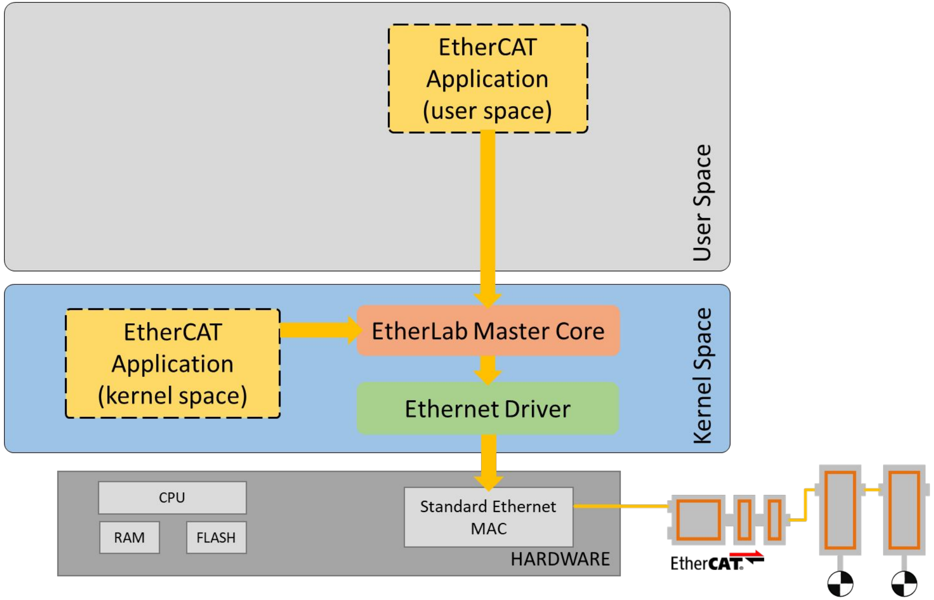
Advantage: non-cumulative local errors.

Result: network-wide synchronization accuracy better than 100 ns, even across hundreds of nodes.

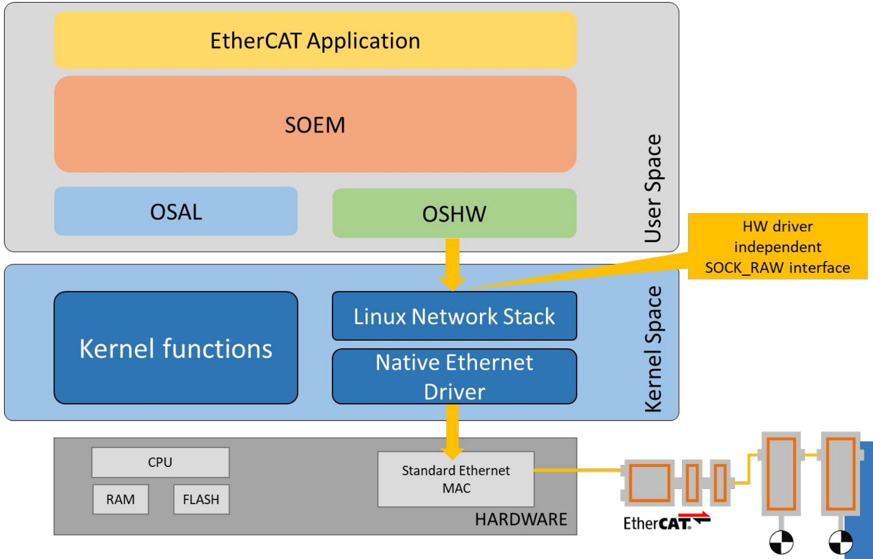
EtherCAT : System vendors

Different Master Architectures

IgH Etherlab



SOEM

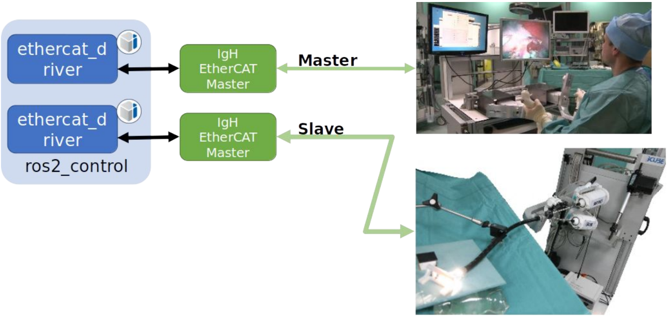


ICube

Development context

Robotics at the ICube Lab.

Lab activities: research in **engineering**, **computer science** and **image processing** with focus on **healthcare technologies**, **environment monitoring and protection** and **sustainable development**.



ethercat_driver

ethercat_driver

ros2_control

IgH EtherCAT Master

IgH EtherCAT Master

Master

Slave

❖Robotics

Research areas

- Laparoscopic and endoscopic surgery,
- Percutaneous interventional imagerie,
- Telemanipulation with force and haptic feedback,
- Visual servoing,
- Robotics system design.

Technical Expertises

- Command and Control systems,
- Software development,
- Mechatronics design,
- Multi-material and silicon additive manufacturing.

EtherCAT at the ICube Robotics Lab.

- More than 15 years of development using **EtherCAT**
- Resources:
  - **3 research engineers** (1 dedicated, 2 experts),
  - 10+ practitioners: researchers, postdocs and PhD students
- All new robotics systems are build using EtherCAT

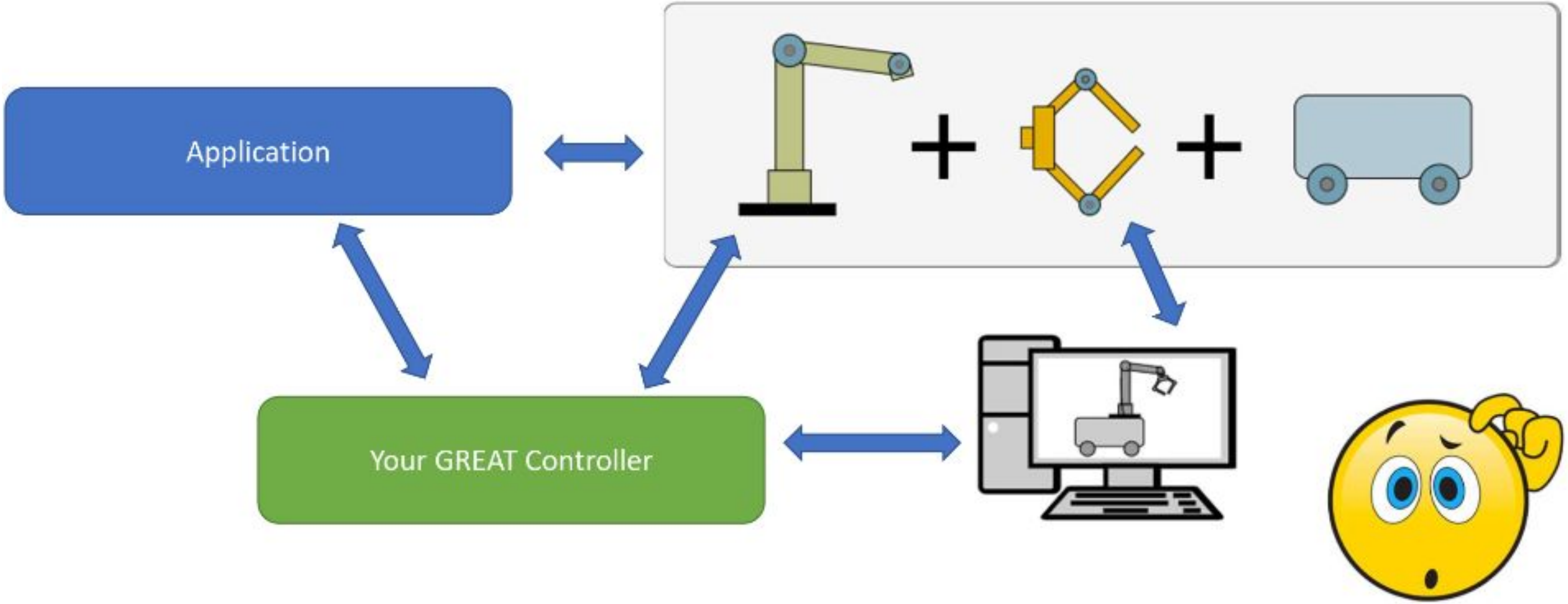
❖Medical Robotics

- **Hard requirements :** provable safety
- Focus research efforts on patient added-value technologies, not system infrastructure

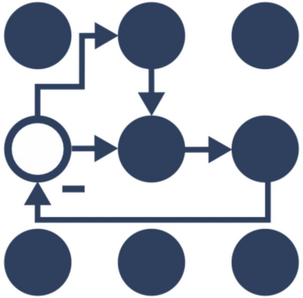
Ethercat_driver_ros2

ROS2 CONTROL

ROS2 Control - Motivation

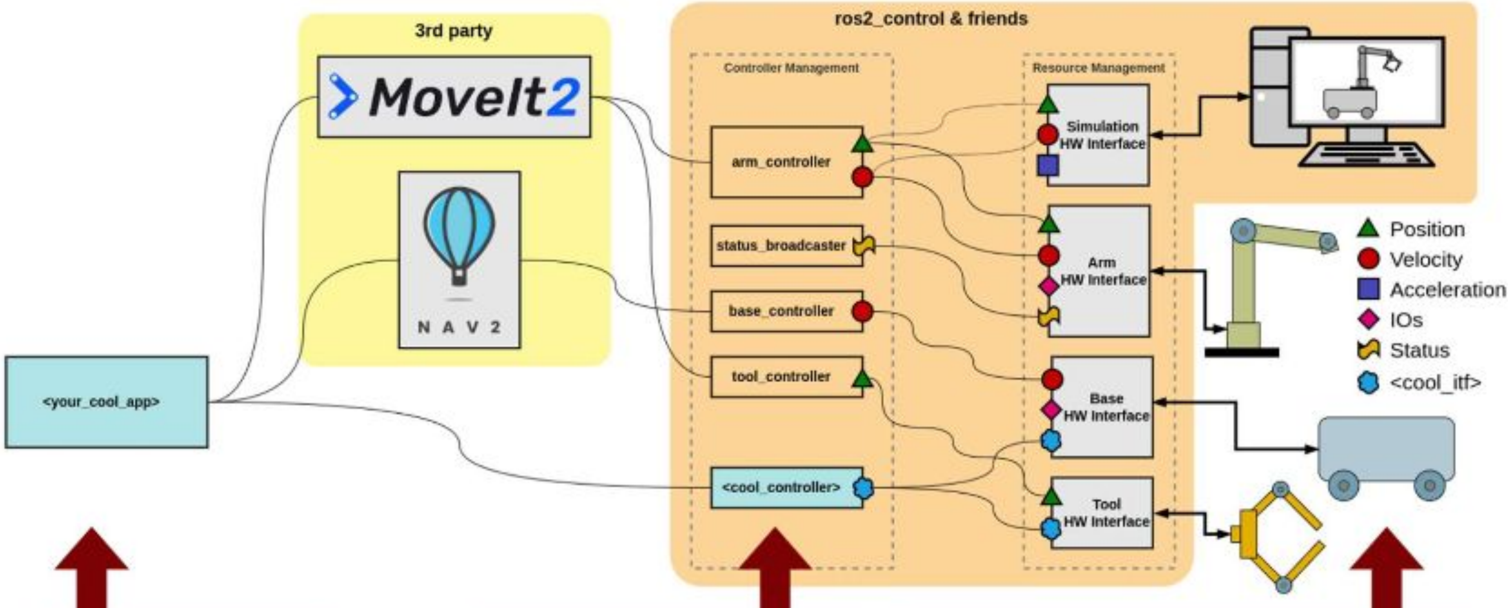


ROS2 Control



- Framework for real-time robot control
- Designed for general robotics applications
- Abstraction Layer for simple integration of hardware and controllers
- Standard Interfaces and Robot Agnostic design
- Modular design and easy configuration using description files
- Focus on applications and reuse of available hardware drivers and controllers
- Easy switch between real robot and simulation
- Use of standard interfaces to integrate 3rd party ROS2 packages

ROS2 Control - Overview



Maximize resources spent on actual **client application**

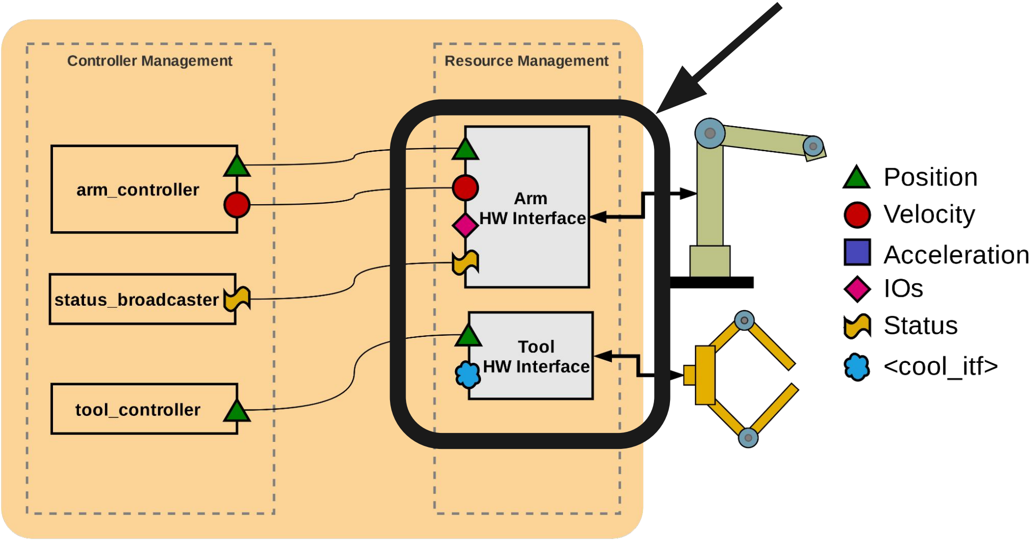
- **Leverage** existing controllers
- **Implement** custom ones, extend existing

- **Leverage** simulation backend
- **Real robot** backend - extend existing ones or create your own

CC-BY: Denis Stogl, Bence Magyar (ros2_control)

An integration for ROS2 control

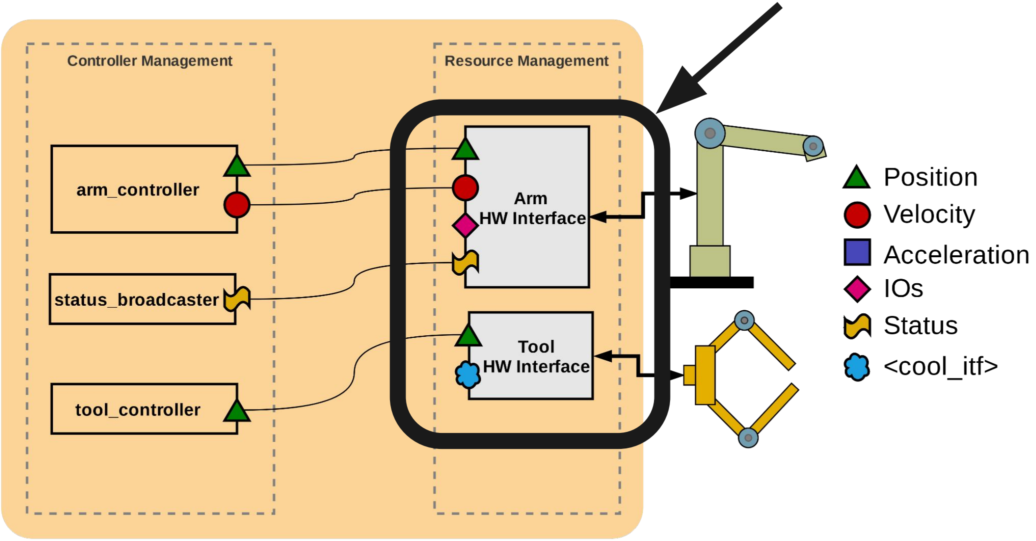
The ethercat_driver_ros2 package provides a generic way to parameterize and assemble ros2_control hardware interfaces of EtherCAT based hardware modules.



CC-BY: Denis Stogl, Bence Magyar (ros2_control)

An integration for ROS2 control

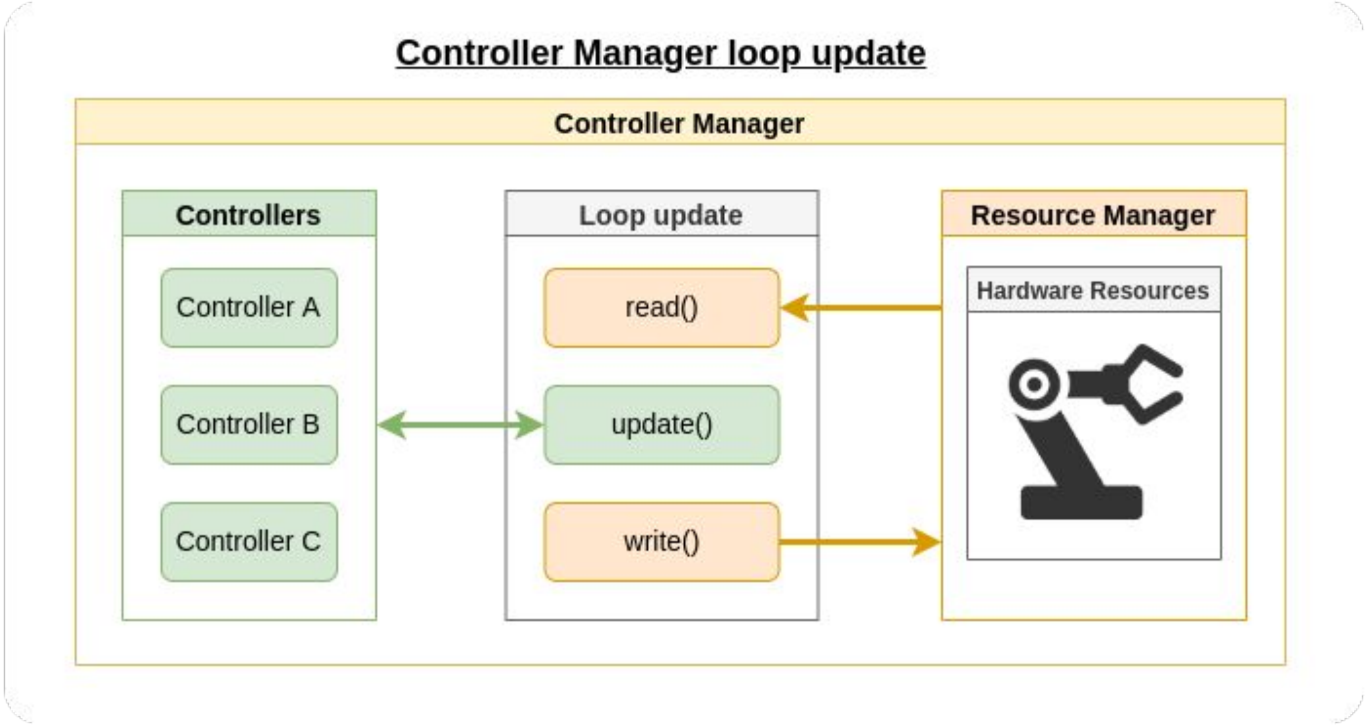
The **ethercat_driver_ros2** package provides a generic way to parameterize and assemble **ros2_control hardware interfaces** of EtherCAT based hardware modules.



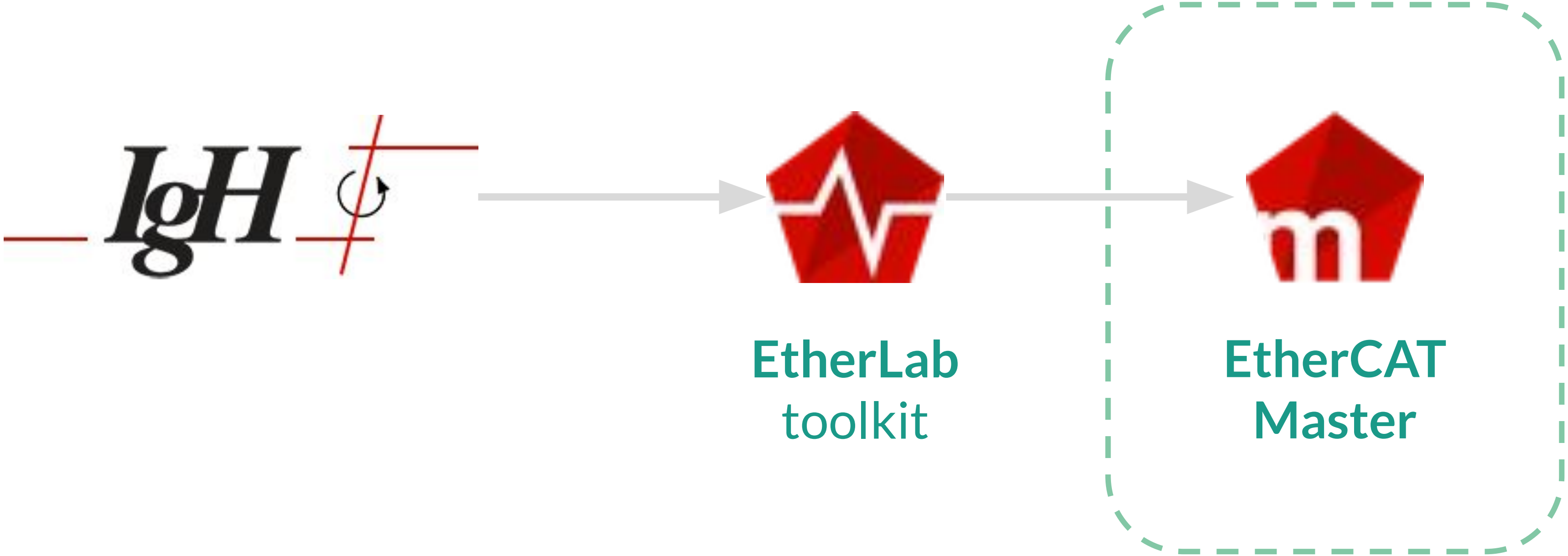
CC-BY: Denis Stogl, Bence Magyar (ros2_control)

Enable real-time control loops based on EtherCAT

ROS2 Control - Controller Manager Loop



Interfaced with the Open Source EtherCAT Master from EtherLab (IgH)



EtherLab toolkit

EtherCAT Master

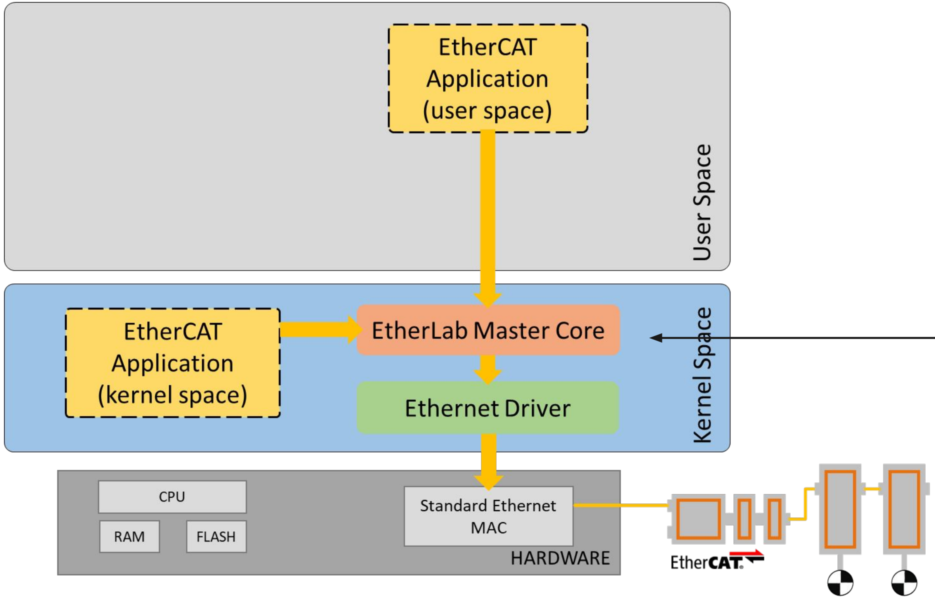
By Florian POSE (IgH)

Ethercat system tools offered by ICube ?

ROS2 CONTROL

System tools for EtherLab (IgH) Master

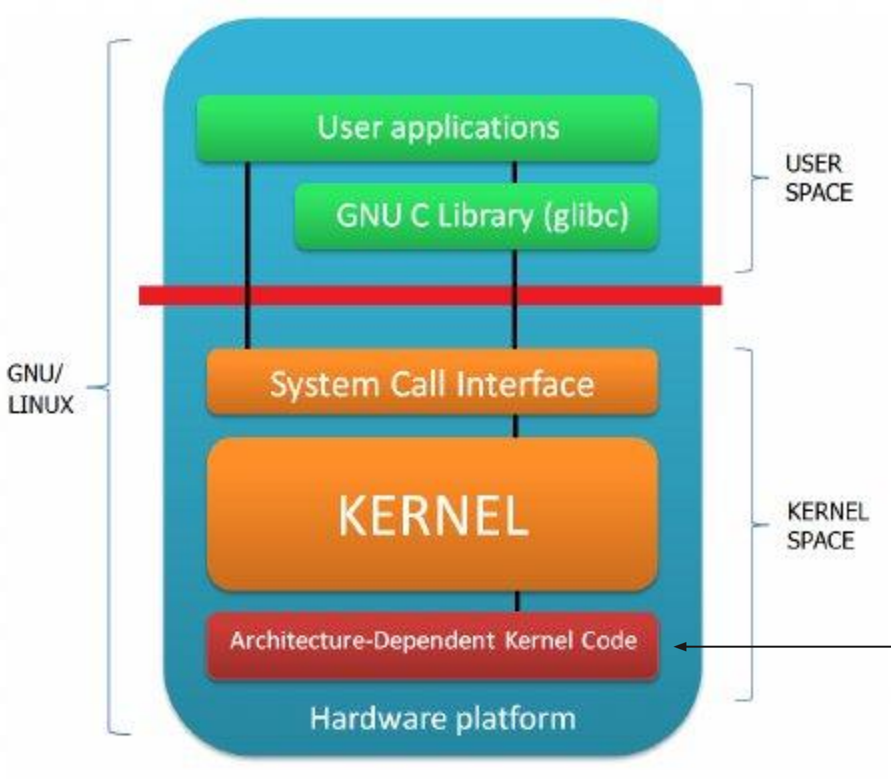
IgH Etherlab



Efficiency / True Real Time capabilities

EtherLab Master Core

System tools EtherLab (IgH) Master



But dev. conundrum: needs to be re-installed for each new version of the Linux Kernel

IgH EtherLab Master Core

System tools EtherLab (IgH) Master

New solution : automatic installer from a custom **debian package.**

For:

- **amd64**
- **arm64 (Raspberry Pi)**

architectures

What brings ethercat_driver_ros2 package ?

EtherCAT®

ROS2 CONTROL

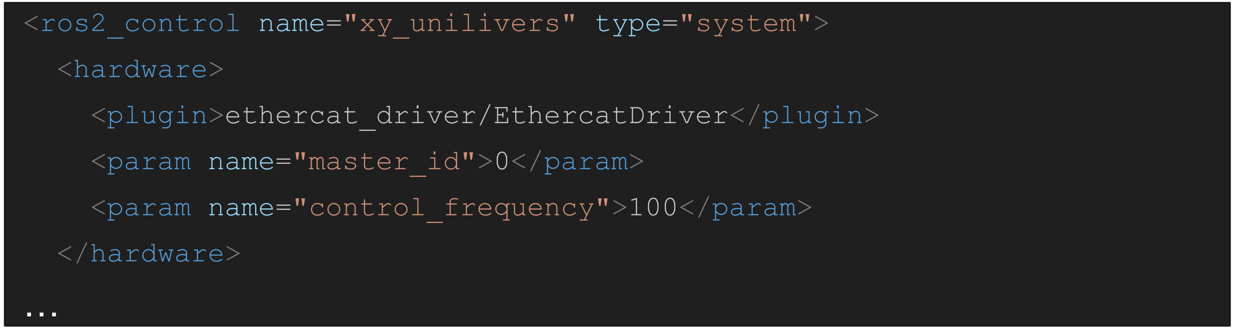
The EtherCAT driver for ROS2

Features

- Integration with **ros2_control**
  - use ros2_control XML/URDF/xacro parameter files
- Generic and friendly slave configuration system
  - use YAML configuration files specific to each EtherCAT slaves
- Available library of slave configurations for popular industrial slaves
- Generic CiA402 Motor Drive Slave System
- Independent plugin-based module design
- hardware interface configuration requires no source code modification

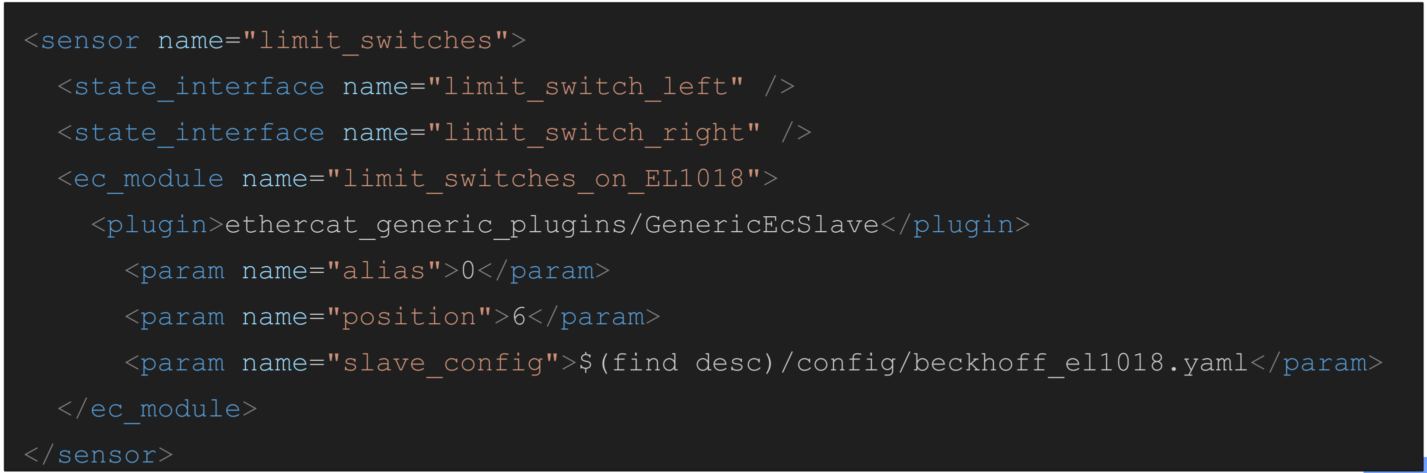
Setup example

- Defining each EtherCAT Master as a Hardware Interface of type ***system***



```
<ros2_control name="xy_unilivers" type="system">
  <hardware>
    <plugin>ethercat_driver/EthercatDriver</plugin>
    <param name="master_id">0</param>
    <param name="control_frequency">100</param>
  </hardware>
...
```

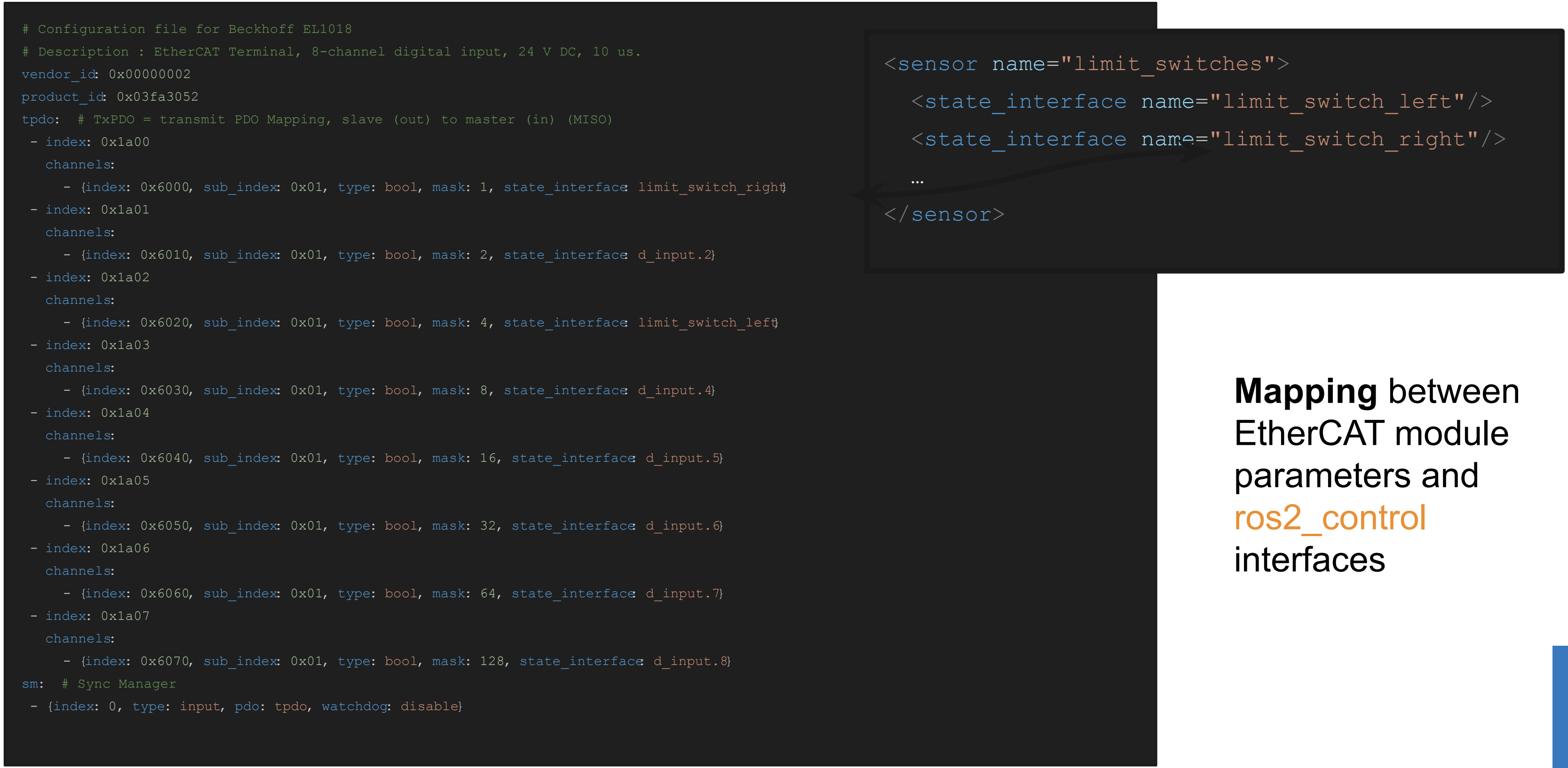
- Defining each EtherCAT slave as an **ecSlave** plugin (here a **GenericEcSlave**)



```
<sensor name="limit_switches">
  <state_interface name="limit_switch_left" />
  <state_interface name="limit_switch_right" />
  <ec_module name="limit_switches_on_EL1018">
    <plugin>ethercat_generic_plugins/GenericEcSlave</plugin>
      <param name="alias">0</param>
      <param name="position">6</param>
      <param name="slave_config">$(find desc)/config/beckhoff_el1018.yaml</param>
  </ec_module>
</sensor>
```

Setup example

- Corresponding generic slave YAML configuration file



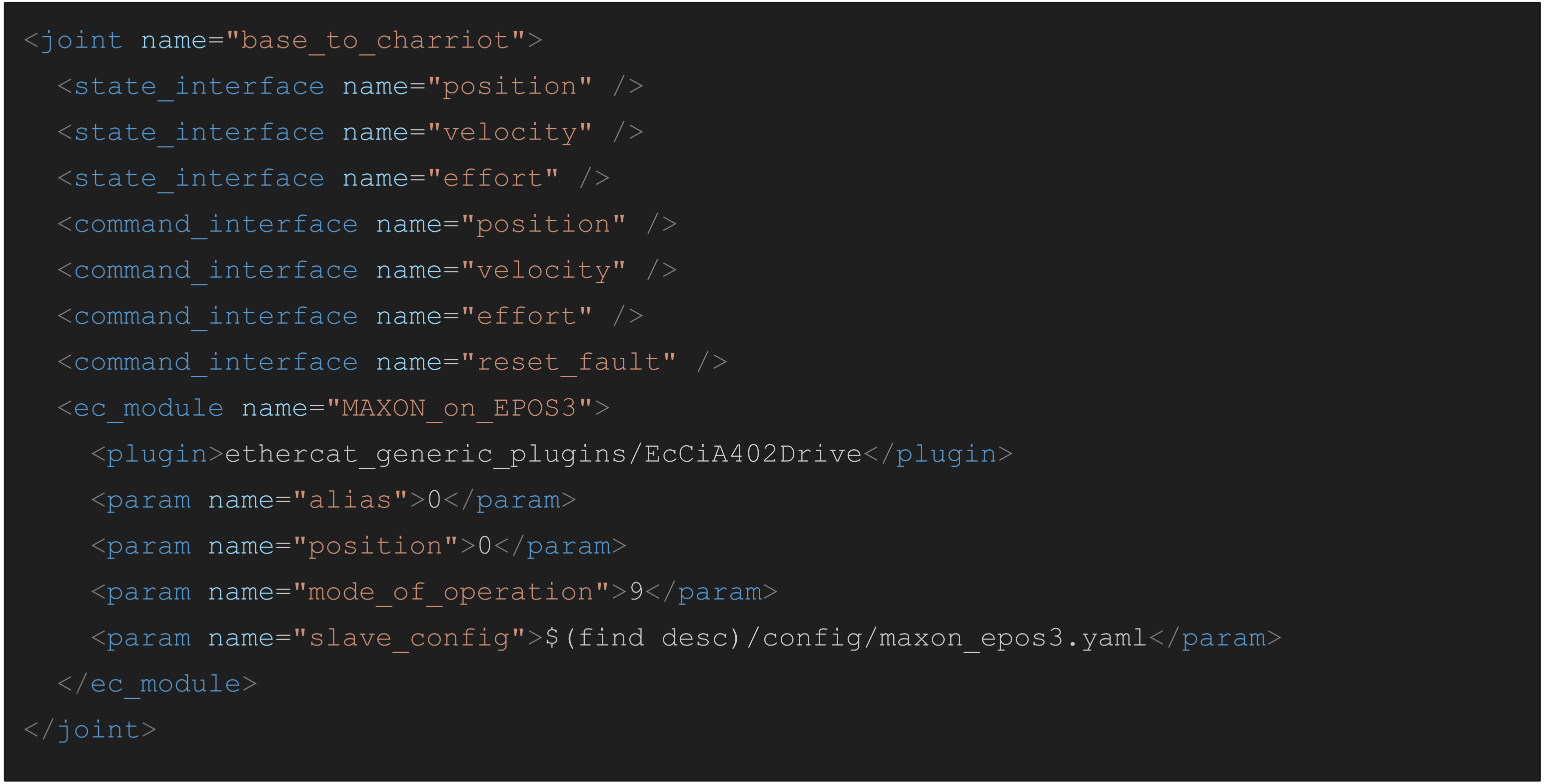
```
# Configuration file for Beckhoff EL1018
# Description : EtherCAT Terminal, 8-channel digital input, 24 V DC, 10 us.
vendor_id: 0x00000002
product_id: 0x03fa3052
tpdo:  # TxPDO = transmit PDO Mapping, slave (out) to master (in) (MISO)
 - index: 0x1a00
   channels:
     - {index: 0x6000, sub_index: 0x01, type: bool, mask: 1, state_interface: limit_switch_right}
 - index: 0x1a01
   channels:
     - {index: 0x6010, sub_index: 0x01, type: bool, mask: 2, state_interface: d_input.2}
 - index: 0x1a02
   channels:
     - {index: 0x6020, sub_index: 0x01, type: bool, mask: 4, state_interface: limit_switch_left}
 - index: 0x1a03
   channels:
     - {index: 0x6030, sub_index: 0x01, type: bool, mask: 8, state_interface: d_input.4}
 - index: 0x1a04
   channels:
     - {index: 0x6040, sub_index: 0x01, type: bool, mask: 16, state_interface: d_input.5}
 - index: 0x1a05
   channels:
     - {index: 0x6050, sub_index: 0x01, type: bool, mask: 32, state_interface: d_input.6}
 - index: 0x1a06
   channels:
     - {index: 0x6060, sub_index: 0x01, type: bool, mask: 64, state_interface: d_input.7}
 - index: 0x1a07
   channels:
     - {index: 0x6070, sub_index: 0x01, type: bool, mask: 128, state_interface: d_input.8}
sm:  # Sync Manager
 - {index: 0, type: input, pdo: tpdo, watchdog: disable}
```

```
<sensor name="limit_switches">
  <state_interface name="limit_switch_left"/>
  <state_interface name="limit_switch_right"/>
  …
</sensor>
```

Mapping between EtherCAT module parameters and ros2_control interfaces

Setup example

- Example for a motor driver for a joint

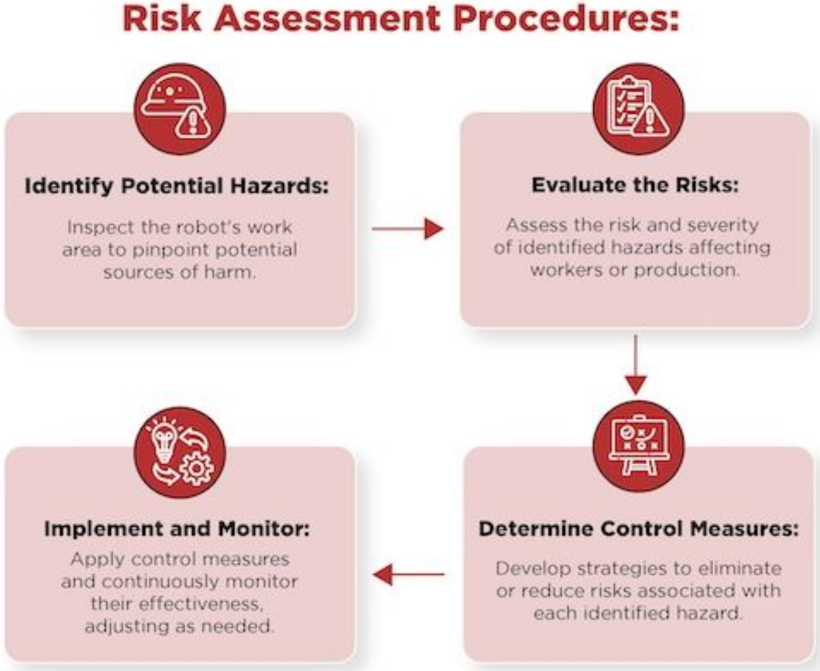


```
<joint name="base_to_charriot">
  <state_interface name="position" />
  <state_interface name="velocity" />
  <state_interface name="effort" />
  <command_interface name="position" />
  <command_interface name="velocity" />
  <command_interface name="effort" />
  <command_interface name="reset_fault" />
  <ec_module name="MAXON_on_EPOS3">
    <plugin>ethercat_generic_plugins/EcCiA402Drive</plugin>
    <param name="alias">0</param>
    <param name="position">0</param>
    <param name="mode_of_operation">9</param>
    <param name="slave_config">$(find desc)/config/maxon_epos3.yaml</param>
  </ec_module>
</joint>
```

FSoE : the new feature in Jazzy release

ROS2 CONTROL

Safety in Robotics



Risk Assessment Procedures:

Identify Potential Hazards:

Inspect the robot's work area to pinpoint potential sources of harm.

Evaluate the Risks:

Assess the risk and severity of identified hazards affecting workers or production.

Determine Control Measures:

Develop strategies to eliminate or reduce risks associated with each identified hazard.

Implement and Monitor:

Apply control measures and continuously monitor their effectiveness, adjusting as needed.

Safety monitoring function

Safety features are provided that make risk assessment easier.

Safe I/O

Supports safe system connection through duplicated safe I/O (8 inputs and 4 outputs)

Position monitoring function

- Monitors robot positions
- Monitors movement into designated areas (8 locations)

Speed monitoring function

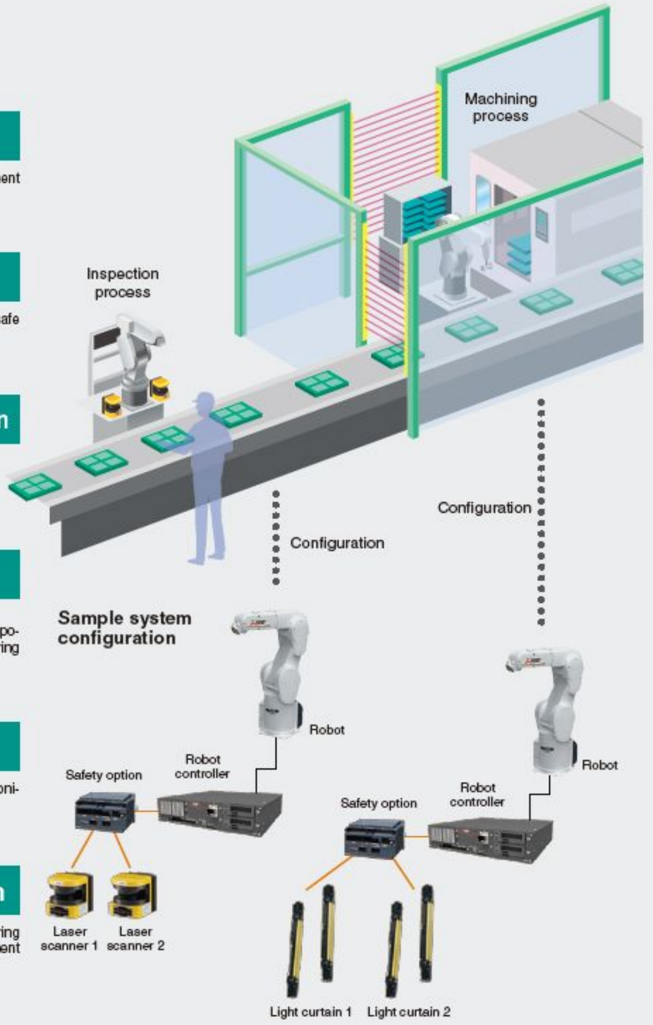
- Monitors robot speeds
- Also capable of monitoring each of the speed components in the X, Y and Z directions for the monitoring point

Safety logic editing

Allows the working parameters (logic) of the safety monitoring function to be defined.

Collision detection function

Detects robot arm collisions as a standard function during teaching or operation. Minimizes damage to equipment such as robot arms, workpieces and grippers.



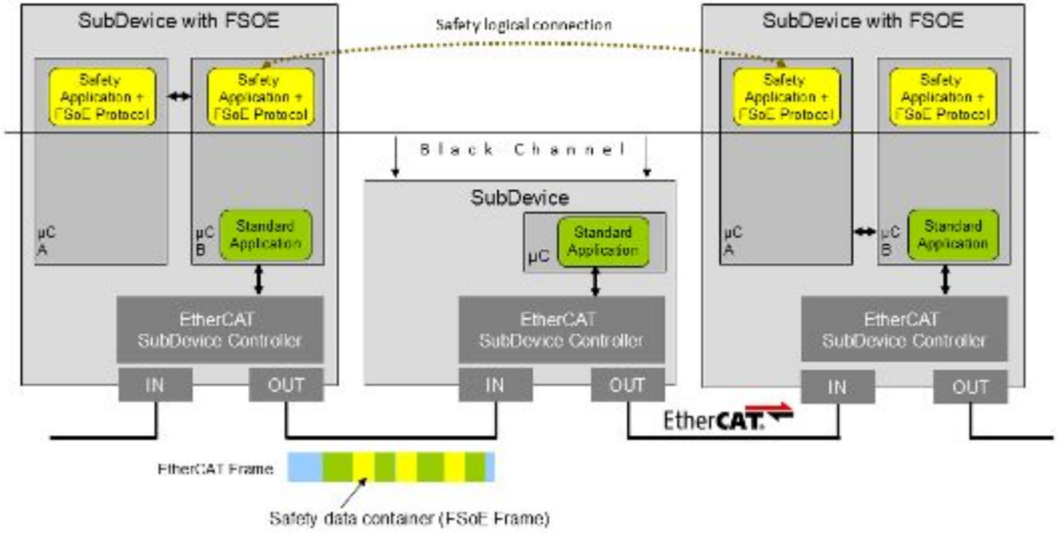
Sample system configuration

What is FSOE

- Functional safety layer integrated into EtherCAT networks
- Allows **safety-critical data** to be transmitted **in parallel** with standard process data on the *same* EtherCAT network
- Based on the **Black Channel principle**: the underlying communication medium does not need to be intrinsically safe; safety is handled at the protocol level.
- TÜV-Certified, developed according to **IEC 61508** and standardized in **IEC 61784-3** -> **SIL 3**.
- Open technology within the EtherCAT Technology Group (ETG).

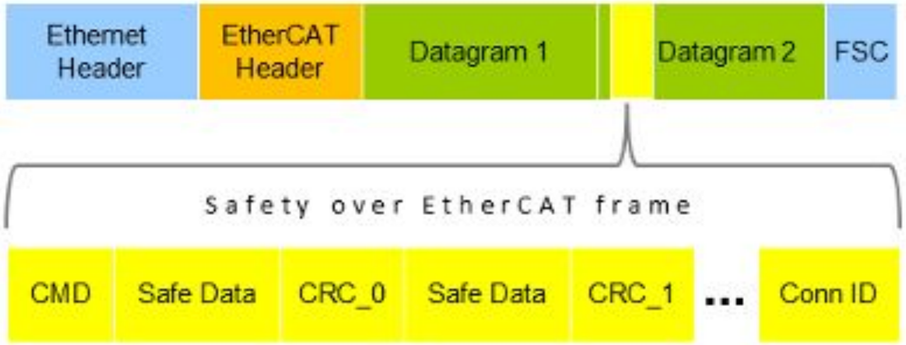
Typical safety Hard- and Software architecture

- Redundant hardware for safety protocol and safety related application
- Thank to Black Channel principle, standard communication interface (1 channel) can be used


SubDevice with FSOE
Safety logical connection
SubDevice with FSOE
Safety Application + FSoE Protocol
Safety Application + FSoE Protocol
Safety Application + FSoE Protocol
Safety Application + FSoE Protocol
Black Channel
SubDevice
µC A
µC B
Standard Application
µC
Standard Application
µC A
µC B
Standard Application
EtherCAT SubDevice Controller
EtherCAT SubDevice Controller
EtherCAT SubDevice Controller
IN
OUT
IN
OUT
IN
OUT
EtherCAT
EtherCAT Frame
Safety data container (FSoE Frame)

FSOE Basics and Frame Structure

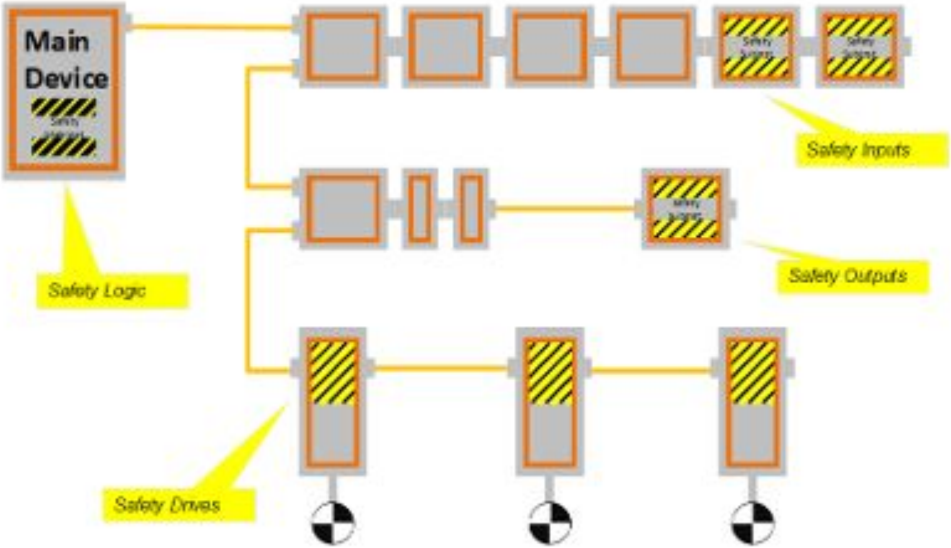
- Safety data is embedded in standard cyclic process data as a “container” with additional integrity information (CRC, watchdog, connection ID)
- Each safety cycle: Checksum (CRC), Connection ID, Watchdog timer for each frame
- Minimum FSoE frame length is 6 bytes, containing: command (CMD), safe data, CRC_0, CRC_1, connection ID
- Each2 Byte Safe Data are checked by a 2 Byte CRC → no strict upper limit on safe-data size



Network Topologies with FSOE

Centralized approach-

- •The FSoE Master (the safety controller) is integrated in the Master EtherCAT device (MainDevice)
- •Simplifies configuration: no need for SubDevice-to-SubDevice communication
- •Drawback: the controller itself must be safety certified

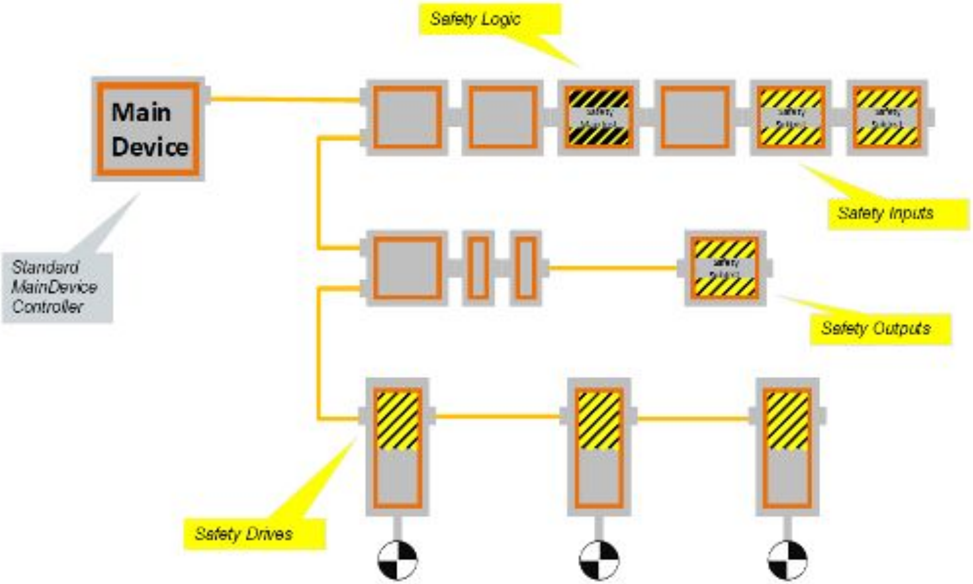


=> Not implemented in ethercat_ros2_driver

Network Topologies with FSoE

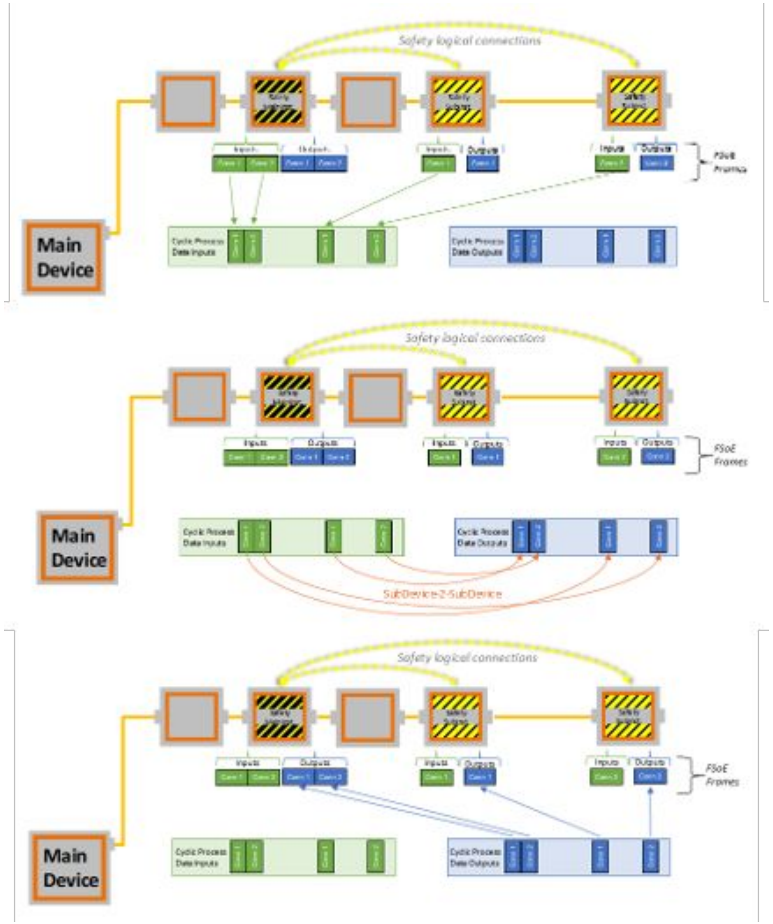
Decentralized Safety Logic

- The FSoE MainInstance runs on a separate EtherCAT SubDevice
- The EtherCAT MainDevice does not need to be safety certified
- Multiple FSOE SubInstances (safe I/O, Drives,… communicate with FSOE MainInstance via the EtherCAT MainDevice



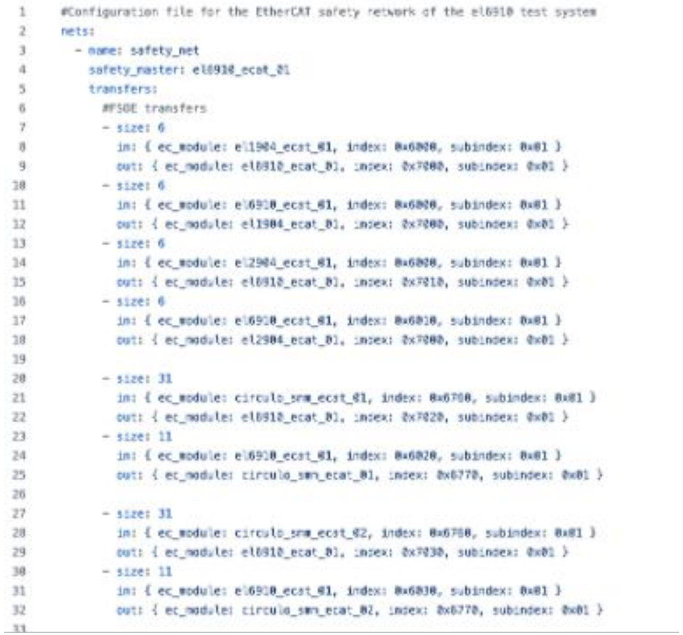
Communication Mechanism

- There is a logical safety connection between the FSOE MainInstance and each FSOE SubInstance
- Safety Frames (FSOE) are exchanged via standard EtherCAT process data
- To enable subdevice to subdevice communication the MainDevice must support copying of data from input to output based on a Copy infos configuration file

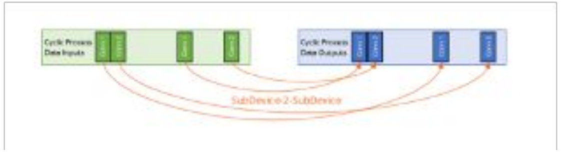

Safety logical connections
Main Device
Inputs
Outputs
FSoE Frames
Cyclic Process Data Inputs
Cyclic Process Data Outputs
Safety logical connections
Main Device
Inputs
Outputs
FSoE Frames
Cyclic Process Data Inputs
Cyclic Process Data Outputs
SubDevice-2-SubDevice
Safety logical connections
Main Device
Inputs
Outputs
FSoE Frames
Cyclic Process Data Inputs
Cyclic Process Data Outputs

Copy infos configuration file

- The ENI (EtherCAT Network Information) file defines which data blocks are copied for Subdevice-to-Subdevice communication.
  - ➢Source
  - ➢Destination
  - ➢Size
- The MainDevice (ethercat_driver_ros2) processes these entries and performs the memory copying

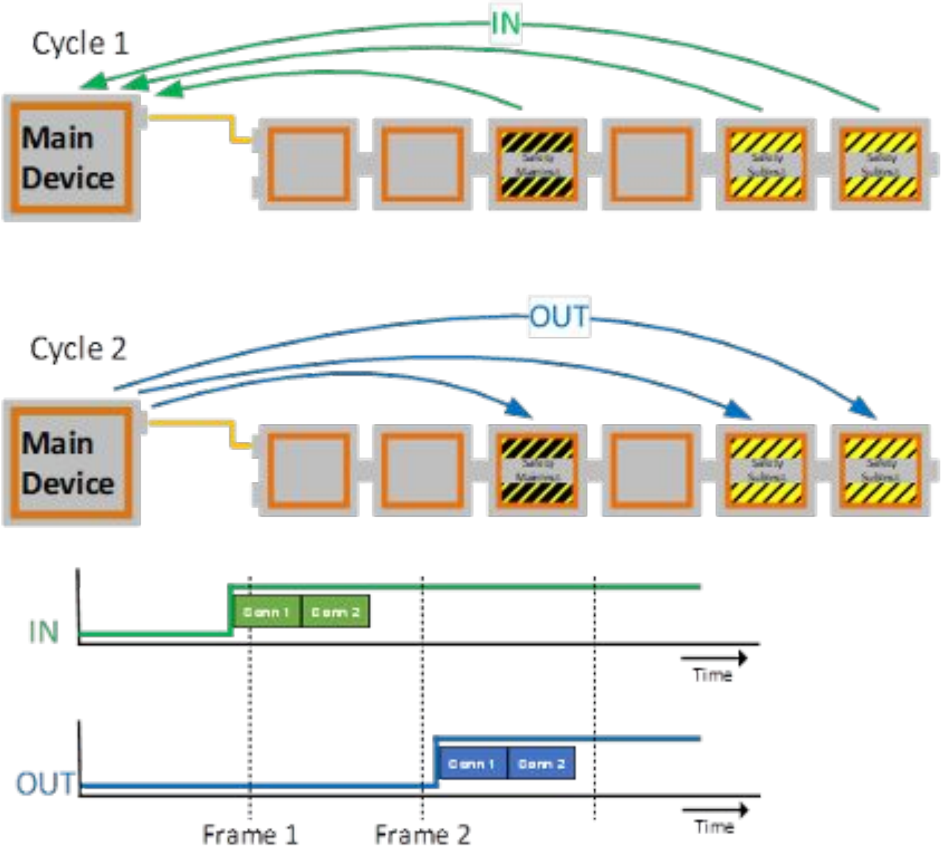


```
#Configuration file for the EtherCAT safety network of the el6910 test system
nets:
  - name: safety_net
    safety_master: el6910_ecat_01
    transfers:
      #FSOE transfers
      - size: 6
        in: { ec_module: el1904_ecat_01, index: 0x6000, subindex: 0x01 }
        out: { ec_module: el6910_ecat_01, index: 0x7000, subindex: 0x01 }
      - size: 6
        in: { ec_module: el6910_ecat_01, index: 0x6000, subindex: 0x01 }
        out: { ec_module: el1904_ecat_01, index: 0x7000, subindex: 0x01 }
      - size: 6
        in: { ec_module: el2904_ecat_01, index: 0x6000, subindex: 0x01 }
        out: { ec_module: el6910_ecat_01, index: 0x7010, subindex: 0x01 }
      - size: 6
        in: { ec_module: el6910_ecat_01, index: 0x6010, subindex: 0x01 }
        out: { ec_module: el2904_ecat_01, index: 0x7000, subindex: 0x01 }

      - size: 31
        in: { ec_module: circulo_smm_ecat_01, index: 0x6700, subindex: 0x01 }
        out: { ec_module: el6910_ecat_01, index: 0x7020, subindex: 0x01 }
      - size: 11
        in: { ec_module: el6910_ecat_01, index: 0x6020, subindex: 0x01 }
        out: { ec_module: circulo_smm_ecat_01, index: 0x6770, subindex: 0x01 }

      - size: 31
        in: { ec_module: circulo_smm_ecat_02, index: 0x6700, subindex: 0x01 }
        out: { ec_module: el6910_ecat_01, index: 0x7030, subindex: 0x01 }
      - size: 11
        in: { ec_module: el6910_ecat_01, index: 0x6030, subindex: 0x01 }
        out: { ec_module: circulo_smm_ecat_02, index: 0x6770, subindex: 0x01 }
```



Role of MainDevice Software (ethercat_driver_ros2)

- The MainDevice (controller) is part of the black channel – it does not need to be safetly-certified
- Ethercat_driver_ros2 supports SubDevice-to-SubDevice communication (safe and unsafe), enabling the safety data to be routed correctly
- The MainDevice reads process-data input, performs the copy per ENI rules, and write to process-data output to deliver safety data

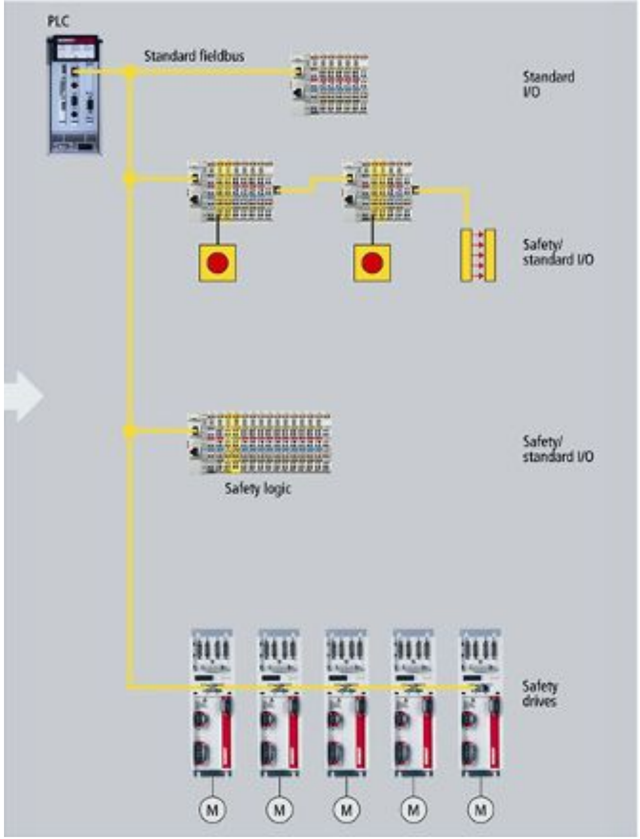


Safety Cycle In Decentralized Mode

- The FSOE cycle : MainInstance sends a safety « MainInstance Frame »; SubInstance replies with its « SubInstance Frame »
- This cycle is longer than the regular EtherCAT cycle, because safety logic imposes additional checks
- Safety checks include CRC, connection Ids and watchdog timers on both sides in each cycle

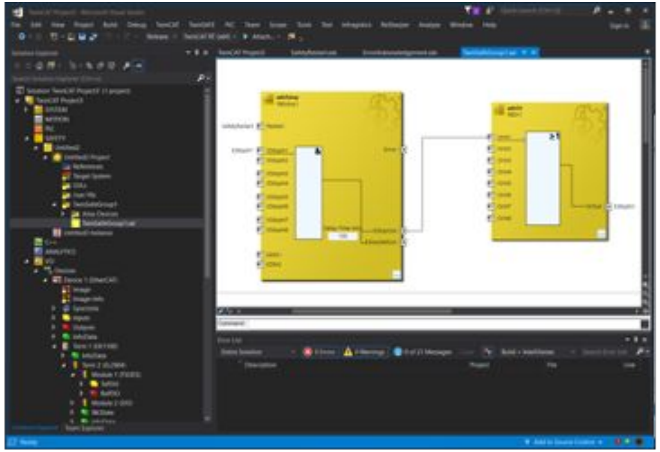
Advantage & Use Cases of Descentralized Approach

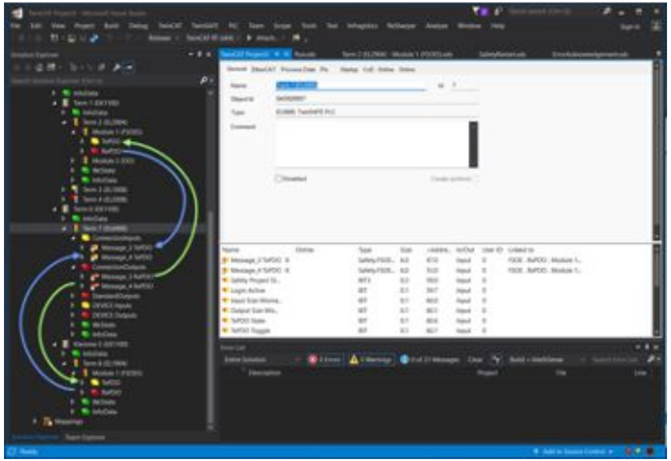
- + : The main controller (MainDevice) can remain non certified, which simplifies design and reduces cost
- + : Safety logic is distributed, which increase modularity and can improve fault isolation
- Use-cases:
  - Safety I/O modules (emergency stop, safety gates)
  - Safe drives / motion control (safe speed, STO, SS1, SS2,…)
  - Modular machines with distributed safety devices



Configuration of FSOE Devices

- Assign FSOE device addresses (16 bit and unique)
  - Via hardware DIP switches on the device
  - Software-base adressing within a safety configuration tool
- Use the manufacturer's safety tool to define the FSOE safety logic (MainInstance) - Beckhoff -> TwinSAFE with TwinCAT
- In tool, you map safety inputs and outputs (e-stop, safety gates, etc...) to logical safety variables
- The safety configuration establishes logical connections between the FSOE MainInstance and all FSOE SubInstances
- SubDevice-to-SubDevice communication is configured





Current limitations

ROS2 CONTROL

Current limitations

- Limitation to only IgH EtherCAT Master
- Not very user friendly for configuration (ROS configuration files limitation)
- No check done to ensure that bus state and provided configuration are consistent
- No multi-master management (for redundancy and security)
- No hot-plug

Roadmap

EtherCAT®

ROS2 CONTROL

Roadmap : more EtherCAT masters supported

1. Integrate/generalize for **SOEM EtherCAT Master**
   Simple **O**pen **E**therCAT **M**aster
   https://github.com/OpenEtherCATsociety/SOEM

   a. no kernel module to compile and install
   b. not good at real-time but …
      - good for prototyping
      - good for starting framework generalization and beginning a design valid for many other EtherCAT Masters

Roadmap : more system tools

1. Provides **docker images** for non real-time app. (mostly dev)
   a. with IgH
   b. with SOEM
2. Provides system images for real-time app.
   a. **Xenomai for arm64(RPi)/amd64** + ROS2
   b. **Ubuntu Preempt-RT** + ROS2
   c. **Xenomai for amd64** + ROS2
   d. **QNX for amd64** + ROS2

Roadmap : more user friendly experience

1. **Create a Config/Plugin library**: a platform to facilitate access and contributions for configuration files or plugins for EtherCAT modules usable for ethercat_driver_ros2
   a. Better documentation of how to create a new configuration file or a new module
2. Provide tools to **facilitate configuration** (maybe GUI to generate configuration files ?)
3. **Error reporting** for inconsistency between bus state and provided configuration
4. **More examples** of complete systems with EtherCAT, ones that users can easily replicate for education

Roadmap : focus on security/predictability

1. Provide **reference installation/systems** with **performance tests** (code+setups) and **test reports**
2. **Support multi-master** redundancy
3. Provide **examples of procedures** for failure recovery/resiliency
4. Provide **doc for pre-certification**

Roadmap : industry 4.0

1. **Hot-plug support** for tool exchange with robotic arms