TOWARDS ASYNCHRONOUS CONTROL SYSTEMS

An asyncio implementation of OPC UA using TANGO green modes

Emilio Morales - 25.09.2025

Overview

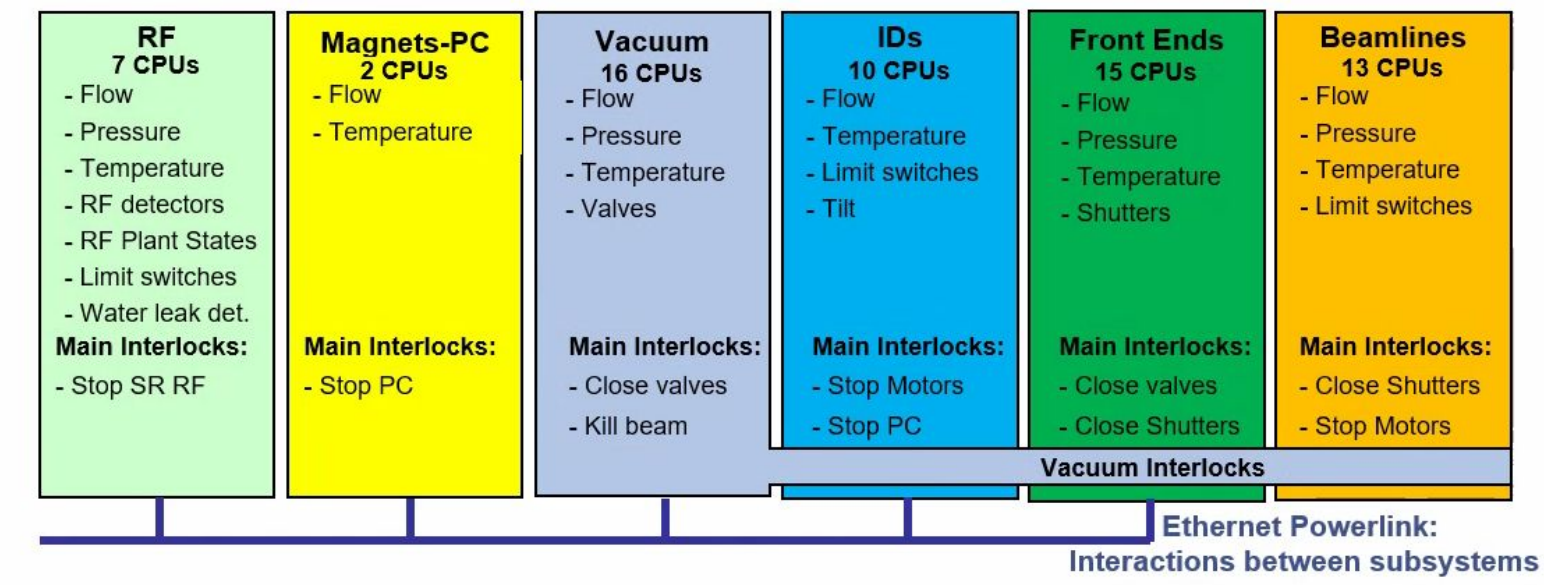
- About ALBA
- ALBA Equipment Protection System
- Technologies involved
- Implementation and benchmarks
- Conclusions

ALBA Synchrotron

- 3rd generation synchrotron light source, based in Barcelona (Spain) in operation since 2012.
- It is foreseen to upgrade to 4th generation light source.
- Most of the controls infrastructure will be upgraded, including the Equipment Protection System.

ALBA Equipment Protection System

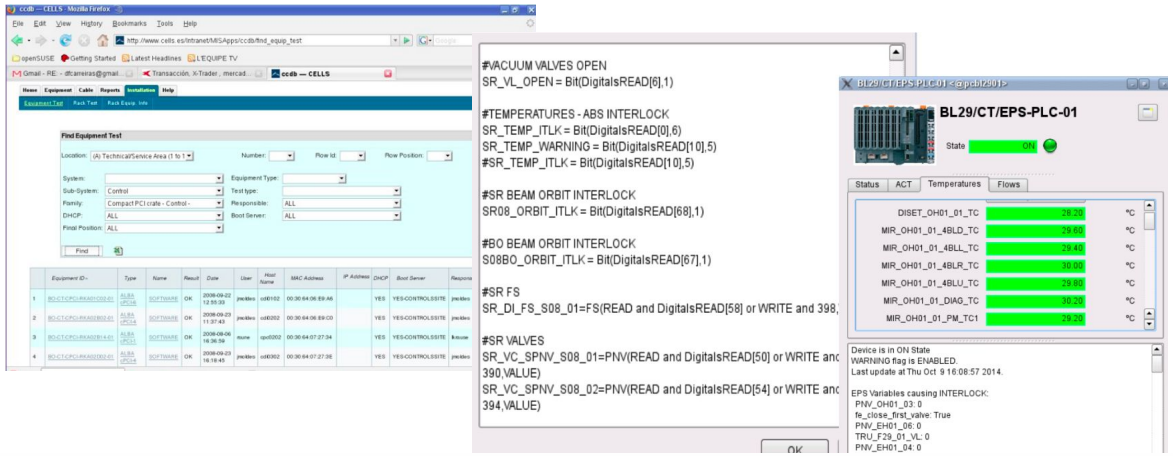
- 6 different subsystems
- 63 PLC CPUs
- ~ 400 nodes - 1500 parameters
- Powerlink Architecture
- Interlock machine in 50 ms
- 50 PLC in main Powerlink network

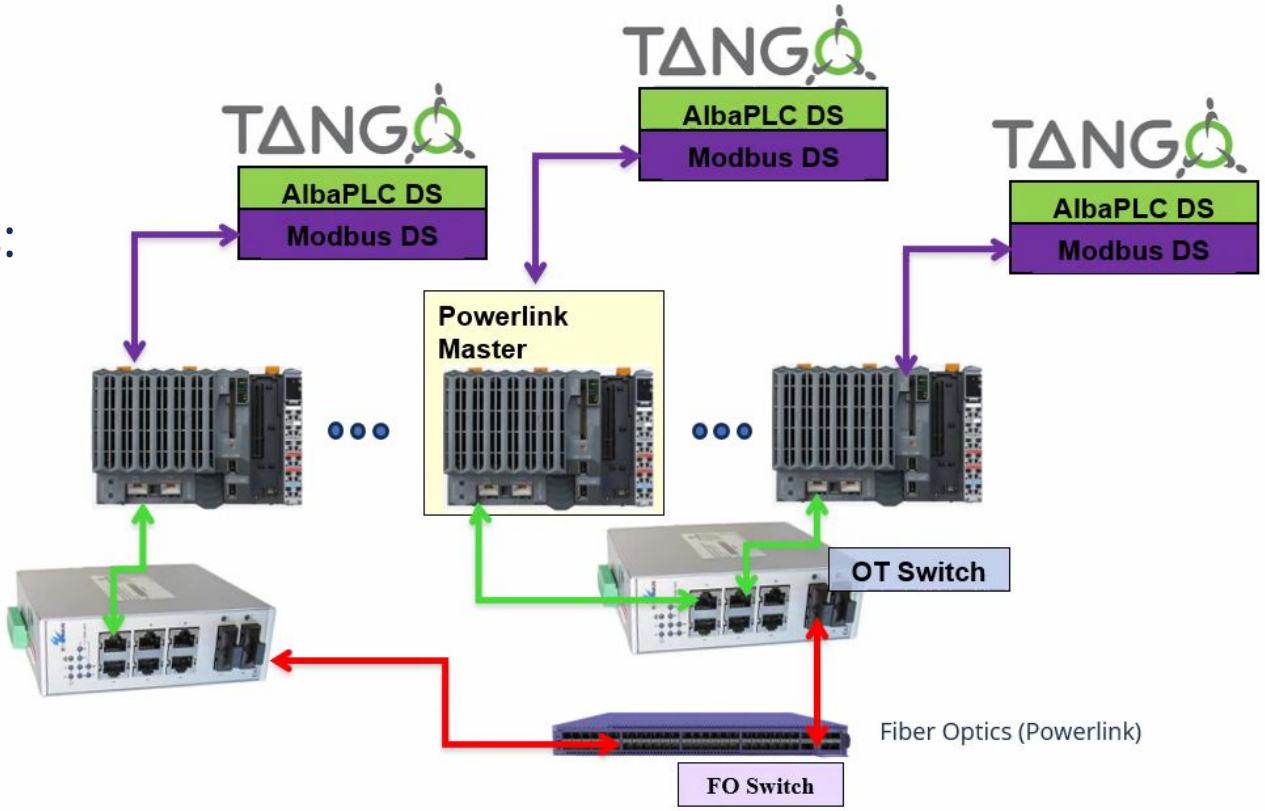


Current EPS System

1. Cabling database ⟷ Autogeneration
2. CSV file: Modbus ⟷ Variable mapping
3. EPS_LIBRARY: 16 bit Status Words
4. Modbus TCP communication
5. Integration with Tango using 2 devices:

   PyPLC (generic) & **AlbaPLC** (specific).





Modbus TCP

Open and widely adopted industrial protocol, valued for its simplicity and broad compatibility, but limited when used at large scale or in modern demanding systems.

Strengths

- *Simplicity*: Lightweight client–server protocol.
- *Compatibility*: Open standard, supported by most PLCs and SCADA systems.
- *Ethernet Integration*: Runs on TCP/IP, reusing existing infrastructure.

Weaknesses

- *Limited Data Model*: 16-bit registers, no explicit data typing or semantics.
- *No Native Security*: Lacks encryption, authentication and data address validation.
- *Scalability*: Polling-based, limited concurrent connections and increasing latency at large scale.

OPC UA

A secure, event-driven standard for industrial systems that enables scalability and advanced data modelling, but achieving performance requires careful setup and tuning, depending on the PLC CPU model.

Strengths

- *Interoperability*: Connects devices from different vendors, simplifying integration.
- *Robust Security*: Built-in encryption, authentication, authorization, and digital signatures.
- *Self-Descriptive & Scalable*: Exports PLC data structures directly and supports client–server and PubSub models for flexible deployments.

Weaknesses

- *Implementation Complexity*: Careful setup and specialized knowledge.
- *Concurrency Challenges*: Asynchronous operation may lead to race conditions.
- *Subscription Limits*: Hard limits depending on vendor, constraints on update rates and number of nodes.

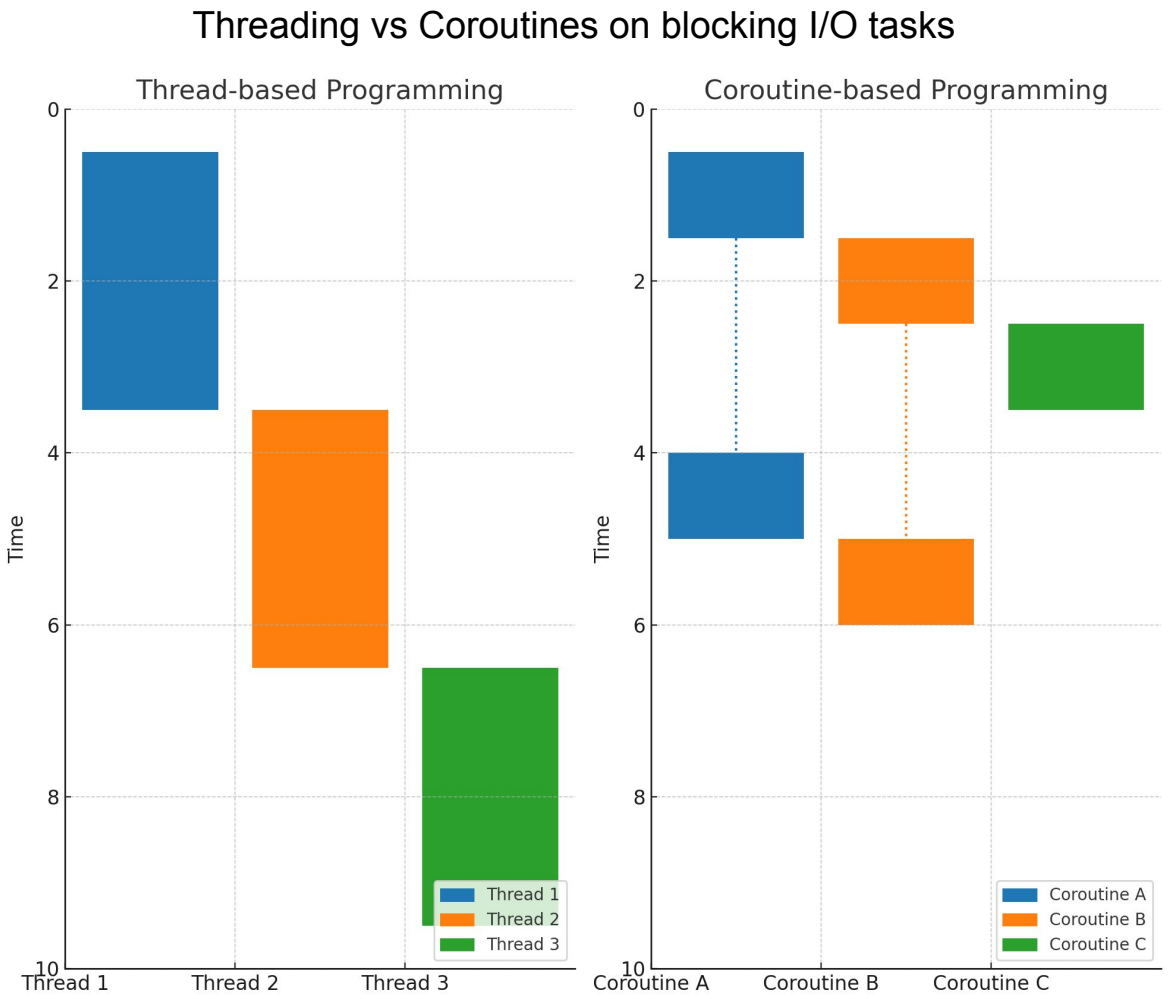
Asynchronous programming

Synchronous / Single-threaded

- Code waits for each task to finish: easy to debug, inefficient for I/O-bound tasks
- When using multiple threads or processes, increasing complexity and resources.

Asynchronous / Coroutine-based

- Allows multiple tasks (coroutines) to progress concurrently without blocking
- Lightweight frameworks
- Useful for certain applications : Networking, High-latency, Event-driven.



asyncio & asyncua

asyncio

Python native library for asynchronous programming.

Pros

- *Efficiency*: Reduces idle time in I/O-bound.
- *Scalability*: Thousands of concurrent connections with low overhead.

Cons

- *Complexity*: Learning async/await patterns and event loop concepts.
- *Debugging*: Concurrency issues (race conditions, deadlocks) are hard to trace.

asyncua

Open-source Python library for implementing OPC UA clients and servers.

Features

- Support for core OPC UA functionalities.
- Asynchronous design ideal for large subscriptions and high-frequency updates.
- Seamless integration with modern Python applications.

FreeOpcUA Project:
https://github.com/FreeOpcUa/opcua-asyncio

TANGO, PyTango and Green Modes

PyTango

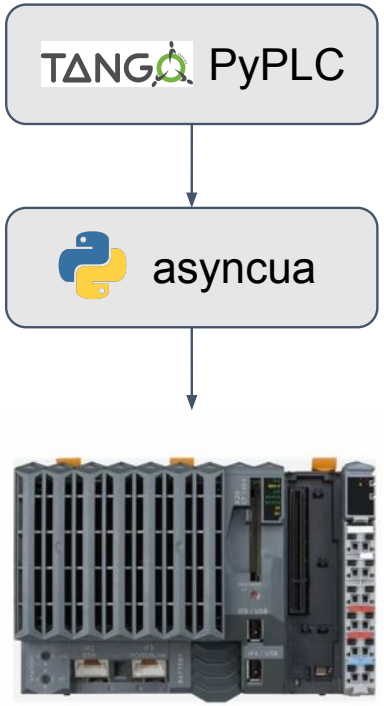
- Tango Python binding to enable rapid development of device servers and client applications.
- Widely adopted across the Tango community.
- **www.tango-controls.org**

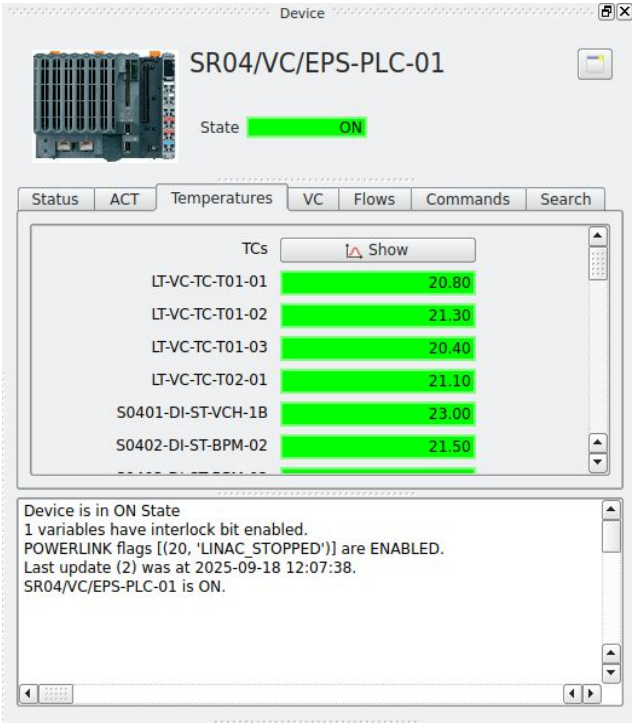
Green Modes

- This enables asynchronous Python frameworks in Tango (**asyncio**, **gevent**).
- Avoids explicit thread management:
  - Lower overhead
  - Simpler code
  - Improved scalability
- Effective for I/O-bound operations and client applications refresh.

PyPLCua- Generic OPC UA Comms. Layer

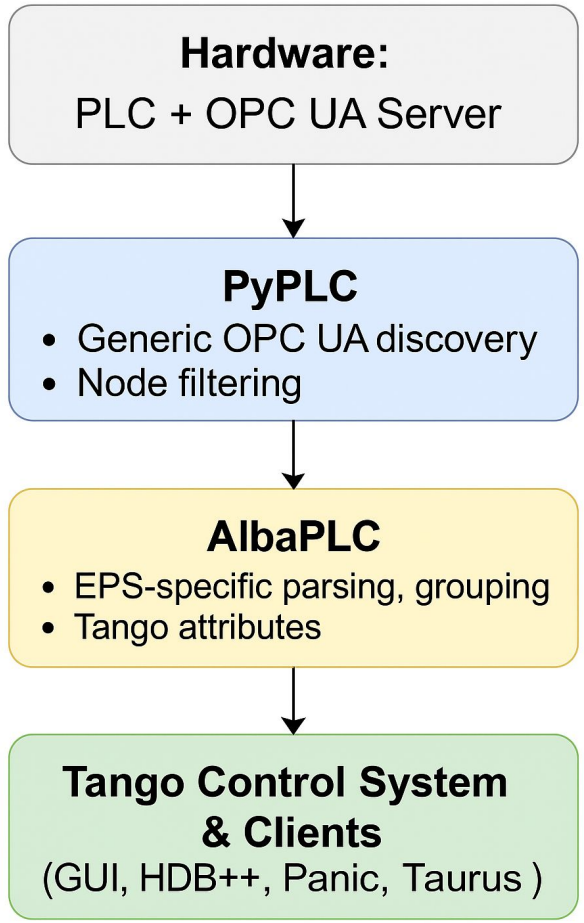
- Python Tango Device Server for **vendor-independent** PLC integration.
- We use client-server approach.
- Exploits *OPC UA*'s self-descriptive model. Dynamically discovers nodes at startup and creates attributes.
- Uses regular expression filters stored in *Tango DB* to select which nodes are read or subscribed.
- Uses *fandango.DynamicDS* to provide flexibility to customize attribute values with Python expressions.





AlbaPLC - EPS-Specific Implementation

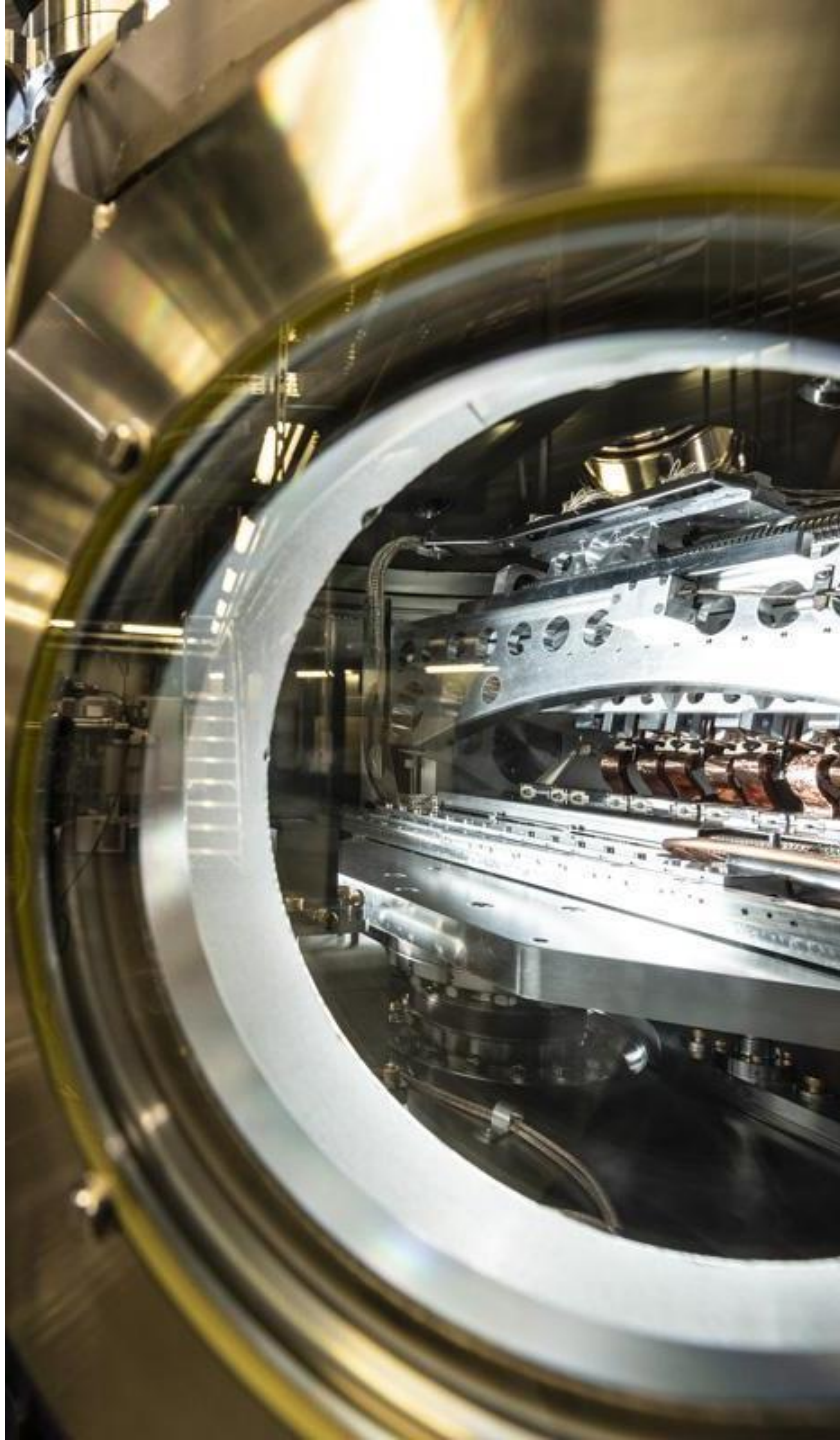
- Subclass of PyPLC tailored to ALBA's Equipment Protection System (EPS).
- Parses PLC EPS Library structures (timestamps, alarms, status bits, ranges).
- Transforms raw OPC UA data into Tango attributes with full quality and diagnostics.
- Groups multi-sensor/actuator devices (valves, shutters, masks) into Tango attributes.



Benchmark comparison

- Benchmark tests using **B&R X20CP1584 CPU**
- Subscription update period of 50 ms
- Modbus registers are read in bunches of 112 with 20 ms wait time

Operation	Nodes	OPC UA	Modbus
Read 1 node	1	**4.14 ms**	60 ms
Read all nodes	1194	11562 ms	24720 ms
Values batch reading	206	18 ms	342 ms
Setting batch reading	782	**325 ms**	626 ms
Values subscription	412	**74 ms**	–
Cache refresh period	412	112 ms	342 ms



ALBA

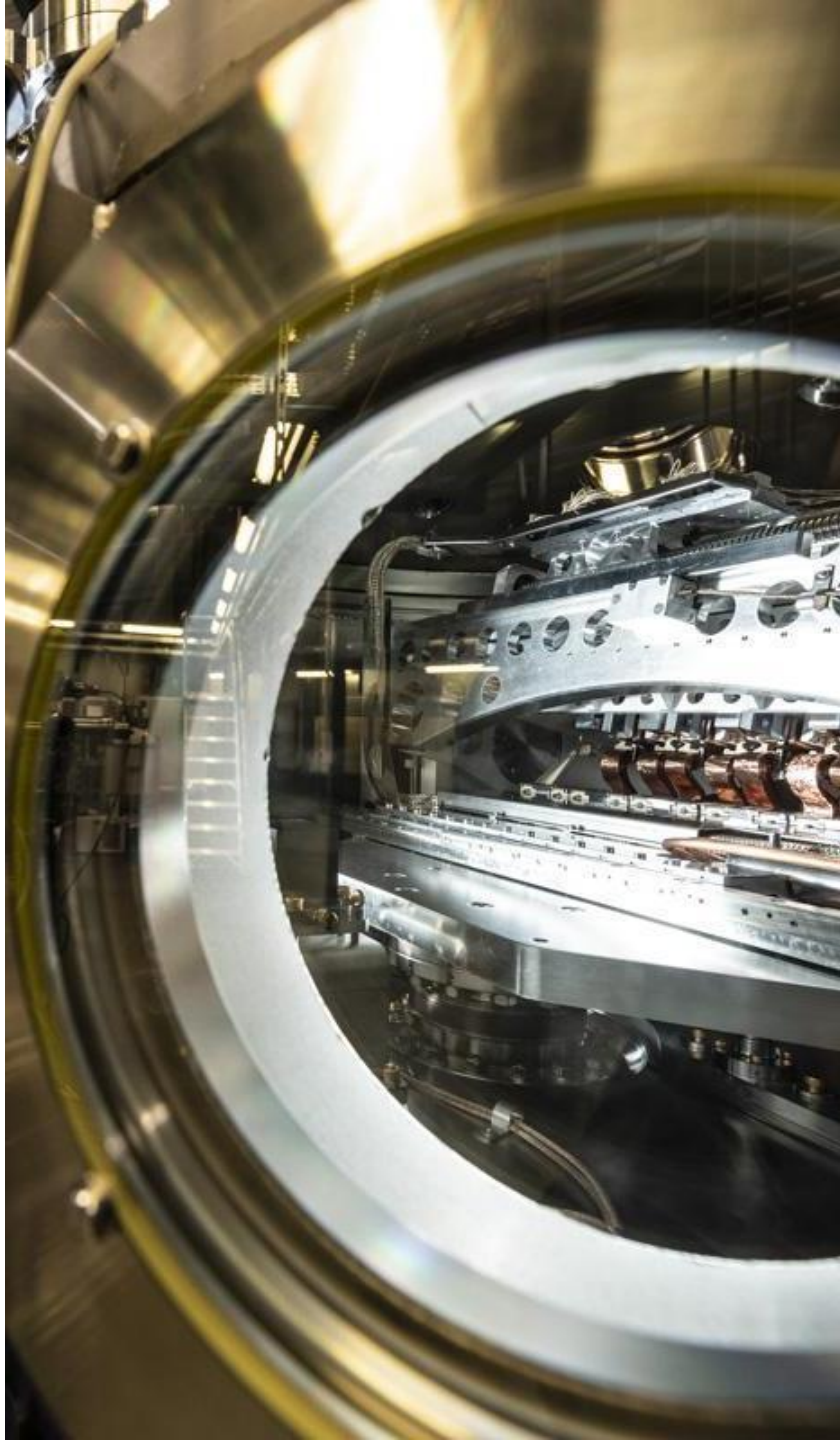
Benchmark comparison

Tests of full movement cycle of a valve:

Send Command → Initiate movement → Movement finished→ Client update

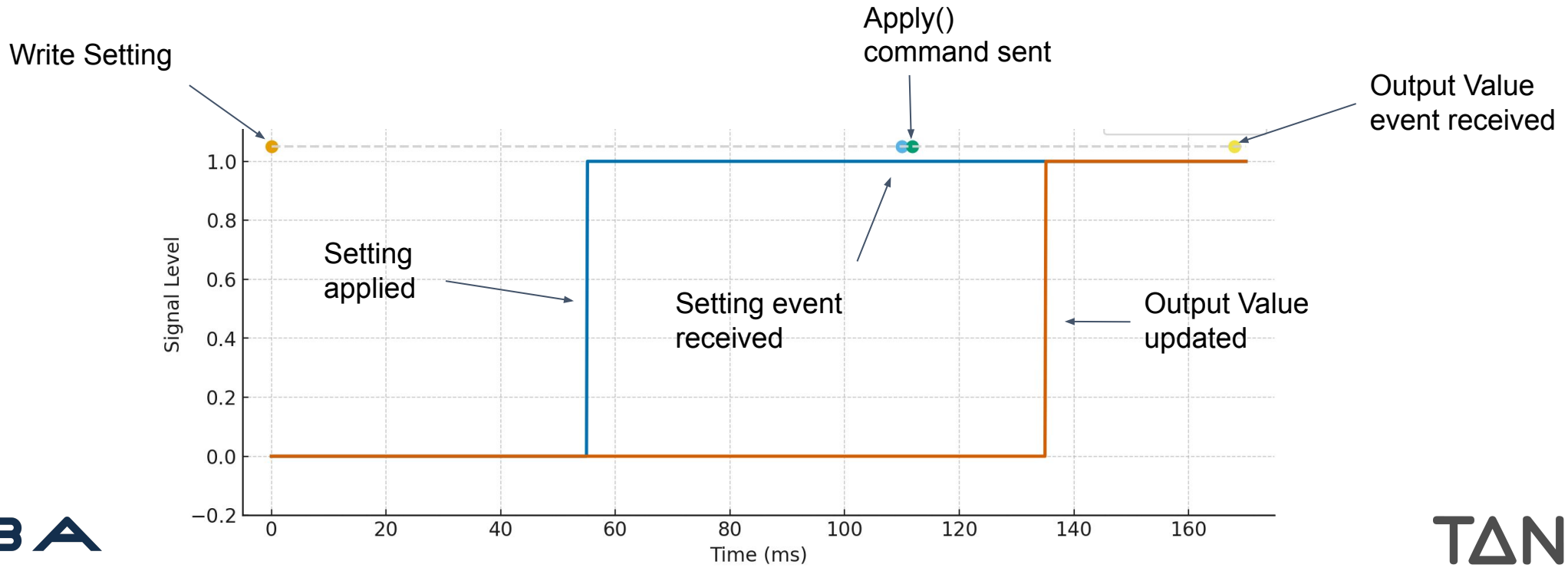
Protocol	min-max refresh time	average
OPC UA	90 - 270 ms	168 ms
Modbus TCP	170 - 1000 ms	499 ms

- OPC UA reacted at 168 ms on average, although with certain variability in response times .
- Modbus exhibited generally higher and more irregular response times, 499 ms on average.
- Modbus performance would be still significantly better for a number of registers below 100.



Concurrency vs Serialization

- Asynchronous execution of commands does not guarantee the order at which they are completed.
- Under certain race conditions, when writing a setting value prior to an Apply() command, **command may be executed against the previous setting value!**
- AlbaPLC manages ongoing operations by using the MOVING state bit and the CHANGING quality, ensuring proper control and coordination of active processes.



Conclusions

- We have benchmarked our Modbus and OPC UA implementations
  - OPC UA shows **better performance**.
  - OPC UA **self-descriptive** nature eliminates the need for intermediate configuration files.
  - For small setups, Modbus is still a performant solution.
- We have developed a Python Tango device class to integrate OPC UA hardware.
  - Uses **asynchronous** programming.
  - It is **open-source** and available on GitLab. →
- We have tested with B&R PLC, let us know if you test it with another vendors!

ALBA Controls and PLC Automation

Zbigniew Reszela
Controls Section Head
zreszela@cells.es

Sergi Rubio
Controls Group Head
srubio@cells.es

Emilio José Morales
Controls, SW developer
emorales@cells.es

Jose Ramos
Controls, SW developer
jramos@cells.es

Jorge Villanueva
Controls, PLC SW & HW
jvillanueva@cells.es

Nil Serra
Controls, PLC SW & HW
nserra@cells.es

Xavier Mercadal
Controls, PLC SW & HW
xmercadal@cells.es

Alberto Rubio
Controls, PLC SW & HW
arubio@cells.es

ALBA

TANGO

www.cells.es

THANK YOU