AsynCA: Controlling Large-Scale Experiments with Tcl

Christian Gollwitzer

EuroTcl 2017

AsynCA & EPICS

- AsynCA is a Tcl package to interface with EPICS in an asynchronous way
- EPICS (Experimental Physics and Industrial Control System) is a widely used distributed control system for large scale experiments

Q: When we have already dbus, CORBA, why we need another middleware? What's so special about EPICS?

http://www.github.com/auriocus/AsynCA

EPICS Overview

EPICS…

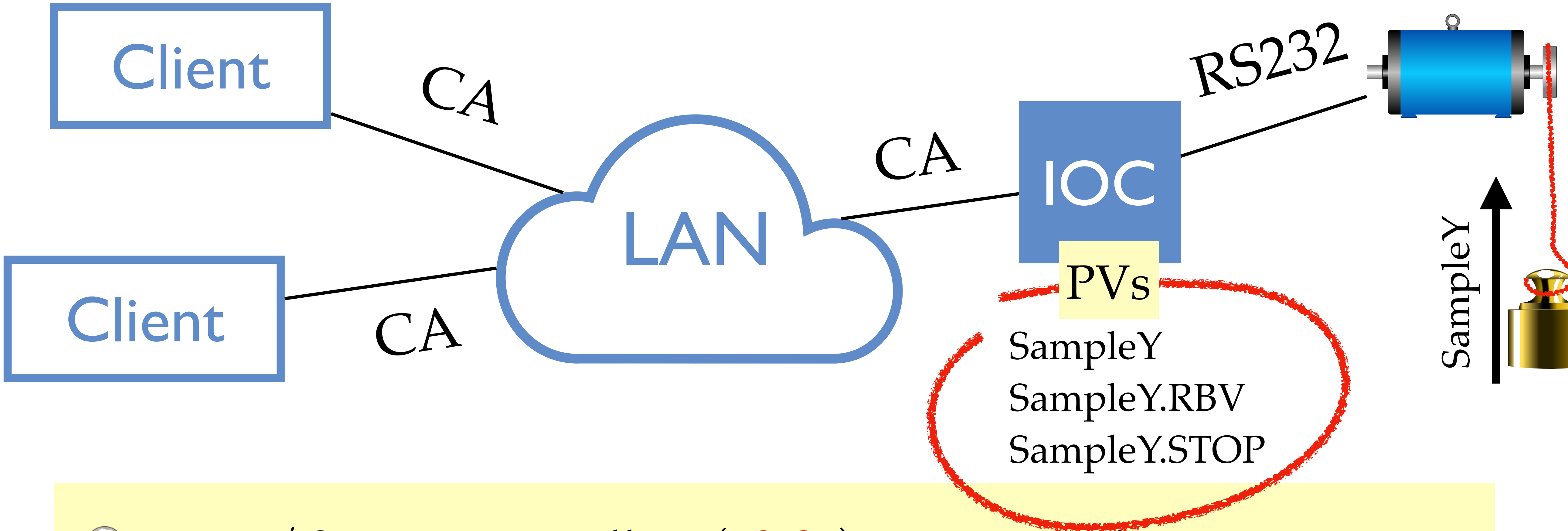
- …is event-based (asynchronous I/O)
- …is distributed (and highly scalable)
- …requires no configuration on the client
- …and no central server either

→ What means large scale?

EPICS Overview

- ~60 motors, ~30 detector devices
- ~8000 process variables
- 1 of ~45 beam lines

EPICS Overview



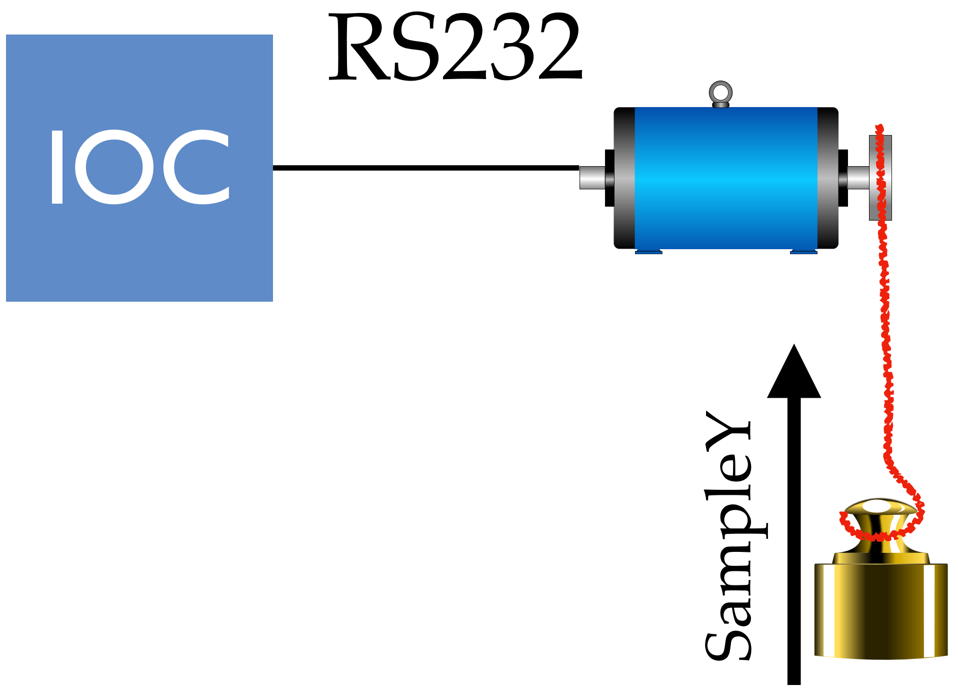
- Input/Output controllers (IOCs) connect to devices (via USB, RS232, GPIB, …)
- Process Variables (PVs) such as *setpoint* or *current position* are exported via the Channel Access (CA) protocol
- There can be multiple IOCs and clients connected, no central server

EPICS Overview

Typical motor record

SampleY
SampleY.RBV
SampleY.STOP
… (~120 PVs)

- Writing to SampleY moves the motor
- Reading SampleY.RBV shows the current position
- Writing to SampleY.STOP halts the motor



Short demonstration

Simulation courtesy to Mika Pflüger

EPICS Client C API

- `ca_create_channel()` connects to a PV
- `ca_put()` / `ca_put_callback()` write to PV
- `ca_get()` / `ca_get_callback()` read from a PV
- `ca_create_subscription()` invokes a callback on each PV update

The EPICS library takes care of…

- finding the server in the network, maintaining a TCP connection
- converting between data types and efficient transport (binary protocol)
- running callbacks upon events
- clients connect to PVs using only the PV name

EPICS Client C API

- `ca_create_channel()` connects to a PV
- `ca_put() / ca_put_callback()` write to PV
- `ca_get() / ca_get_callback()` read from a PV
- `ca_create_subscription()` invokes a callback on each PV update

The EPICS library takes care of…

- finding the server in the network, maintaining a TCP c tion
- converting between data types and efficient transpo (binary protocol)
- running callbacks upon events
- clients connect to PVs using only the P name

AsynCA Client API

- `AsynCA::connect -command cb` connects to a PV
- `$pv put value ?-command cb?` writes to PV
- `$pv get -command cb` requests a read from a PV
- `$pv monitor -command cb` invokes a callback on each PV update

AsynCA takes care of:

- converting the EPICS data types to Tcl values (Tcl_Obj)
- mapping the callbacks to events in the Tcl event loop

AsynCA Client API

- `AsynCA::connect -command cb` connects to a PV
- `$pv put value ?-command cb?` writes to PV
- `$pv get -command cb` requests a read from a PV
- `$pv monitor -command cb` invokes a callback on each PV update

AsynCA takes care of:

- converting the EPICS data typ
- mapping the callbacks to event ent loop

Short demonstration

AsynCA Client API

All network I/O in EPICS happens asynchronously:

- `$pv get` does not return the value; delivered in the callback
- `AsynCA::connect` returns a PV, but it is connected only after the connect calls fires
- `$pv put` notifies you when the command is processed (motor has arrived, …)

+ Nice: event based system, short response times

– Complicated programming model

→ Additional Synchronous API functions

AsynCA Synchronous API

- `AsynCA::connectwait` connects multiple PV
- `AsynCA::read?multiple? $pv` reads a / multiple PV
- `AsynCA::putwait` writes to multiple PVs

More complicated than it seems…

- Callbacks can come in any order
- Cancelled callbacks from previous calls can ring back
- AsynCA uses dicts and `vwait` - nesting wait problem

→ Transfer to coroutine mechanism wanted

AsynCA Server API

- `AsynCA::server` creates a server object
- `$s createPV name ?type ?count??` creates a new PV
- `$pv write value` modifies the stored value
- `$pv read` returns the stored value

→ Very few lines of code to create PVs

AsynCA Server API

- `$pv writecommand callback` changes the PV to asynchronous write
- Upon writing, the callback receives a request object
- `$request return` signals the completion of the request
- `$request destroy` signals a failure
- For `readcommand`, the request object accepts the return value

→

- Simple setup of asynchronous PVs
- No other EPICS Tcl library does provide the server API

Conclusion

- EPICS is a sophisticated distributed control system
- AsynCA wraps both the client and server libraries and provides a low-level Tclish interface, mapping callbacks to Tcl events
- Asynchronous programming is facilitated by a few synchronous support routines
- A standard way to do asynchronous I/O would be most welcome (Python has asyncio….)