# EPICS Training

HZB EPICS Summer School 2024


Advancing humanity.
Engineering remarkable.

**Žiga Oven**

ziga.oven@cosylab.com

COSYLAB

# What is EPICS?

COSYLAB

# Contents

- What is EPICS?
- EPICS vocabulary
- Control system architecture
- What does it do?
- How and where does it do it?

COSYLAB

# What is EPICS?

- A Collaboration
  - A world wide collaboration that shares designs, software tools, and expertise for implementing large-scale control systems
- A Control System Architecture
  - A client/server model with an efficient communication protocol (Channel Access) for passing data
  - A distributed real-time database of machine values
- A Software Toolkit
  - A collection of software tools collaboratively developed which can be integrated to provide a comprehensive and scalable control system

# Collaboration

## Major Collaborators:

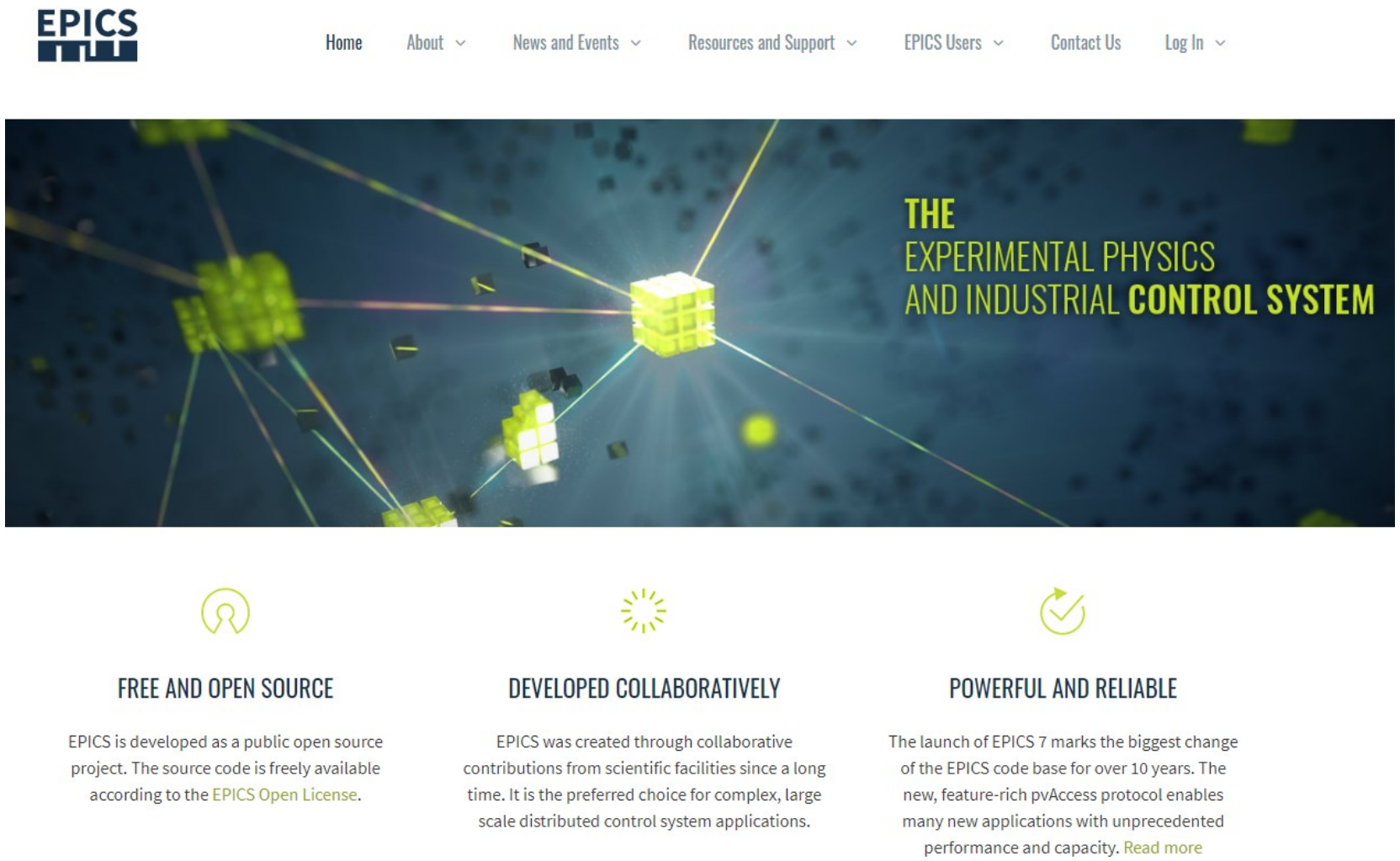
- ANL (USA)
- LANL (USA)
- ORNL (SNS) (USA)
- SLAC (USA)
  - SSRL
  - LCLS
- JLAB (USA)
  - CEBAF
- DESY (Germany)
- HZB (Germany)
- PSI (Switzerland)
  - SLS
  - SwissFEL
- Gemini (USA, Chile)
- Keck (USA)
- KEK (Japan)
- Diamond (UK)
- AS (Australia)
- NSSRC (China)
- BNL (USA)
  - NSLS2
- ASKAP (Australia)
- ITER (France)
- ESS (Sweden)

- And of course…
  - Cosylab!

# Collaboration

- Began in 1989 between LANL/GTA & ANL/APS (Bob Dalesio & Marty Kraimer)
- Over 150 license agreements were signed before EPICS became “open source” in February 2004
- Regular collaboration meetings
  - 2 meetings per year
  - 100-200 participants
  - Over 70 presentations (3+ days) for single meetings
- Home page with links to manuals, presentations, software
  - https://epics-controls.org
- List server; tech-talk: the collaboration in action
- Collaborative efforts vary
  - Assist in finding bugs
  - Share tools, schemes, and advice

# EPICS Home Page



# EPICS Documentation

## EPICS 7.0.8

The EPICS 7.0.8 release contained a number of bug fixes and new features since 7.0.7. The changes and additions were be spread across the different modules, and are described in the appropriate release notes for the sub-modules.

The EPICS 7.0.8.1 release fixes several bugs in the EPICS Core since 7.0.8 along with some minor enhancements and documentation updates. Read the Release Notes linked below for details.

### Documentation

The following documents cover EPICS version 7.0.8 and subsequent patch releases. Additional documents can be found in the documentation subdirectory of the distribution source code, and in the html subdirectory that gets created at build-time.

NOTE: These documents may be revised at any time without generation of a new source release. The core-talk and tech-talk mailing lists are probably the best sources of up-to-date information about the 7.0 series.

Installation Instructions (Readme)

### Release Notes

Base 7.0.8.1

pvData 8.0.6

pvAccess 7.1.7

pvaClient 4.8.0

# Essential EPICS Vocabulary

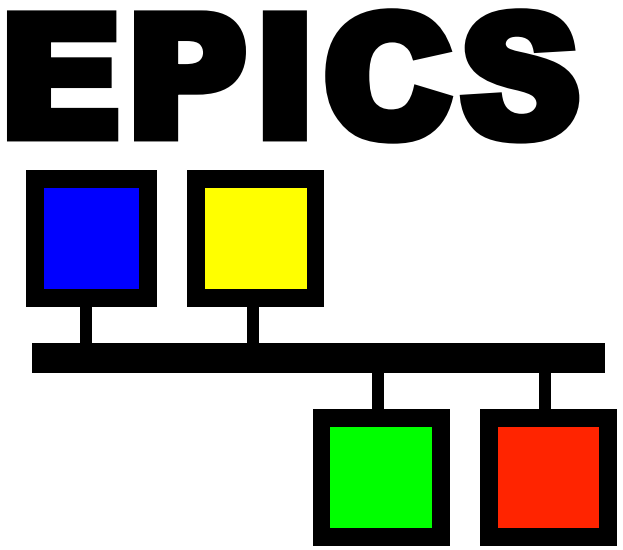
- **EPICS**
  - Experimental Physics and Industrial Control System
- **Channel Access (CA)**
  - The communication protocol used by EPICS
- **Process Variable (PV)**
  - A piece of named data referred to by its PV name
  - The primary object of the Channel Access Protocol
- **Channel**
  - A synonym for Process Variable
- **Channel Access Server**
  - Software that provides access to a Process Variable using the Channel Access Protocol
- **Channel Access Client**
  - Software that requests access to a Process Variable using the Channel Access Protocol

# Essential EPICS Vocabulary

- **IOC – Input Output Controller**
  - A computer running *iocCore,* a set of EPICS routines used to define process variables and implement real-time control algorithms
  - *iocCore* uses database records to define process variables and their behavior
- **Soft IOC**
  - An instance of *iocCore* running as a process on a “non-dedicated” computer (i.e. a computer that is performing other functions as well)
- **Record**
  - The mechanism by which a Process Variable is defined in an IOC (using *iocCore*)
  - Dozens of record types exist, each with its own attributes and processing routine that describe its functionality

# Control System Architecture

- Network-based “client/server” model (hence the EPICS logo).
- EPICS is a distributed control system where every client can see all the servers.




EPICS

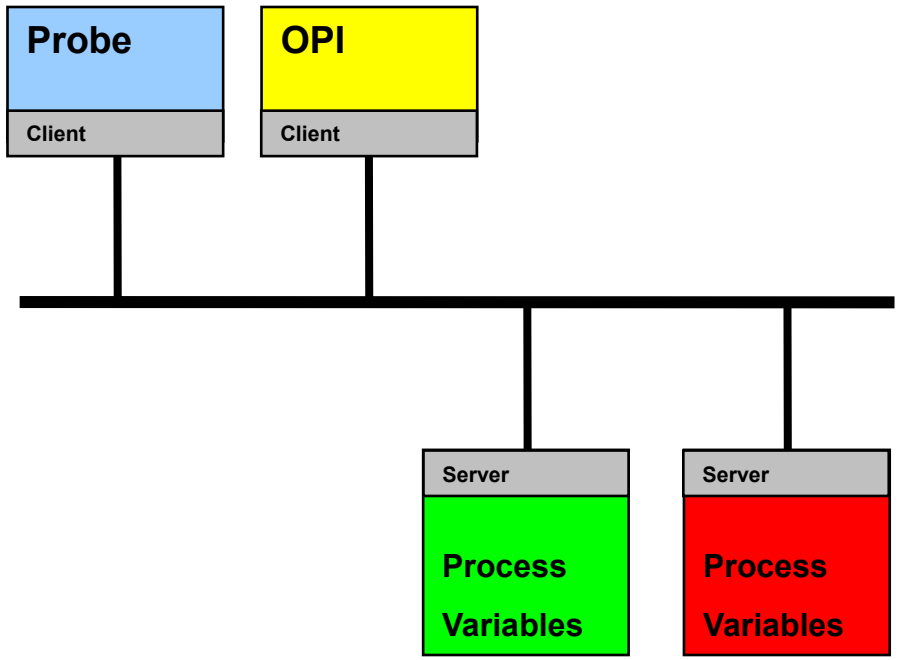
# Control System Architecture

- ***Clients*** are programs that require access to ***Process Variables*** to carry out their purpose.

EPICS

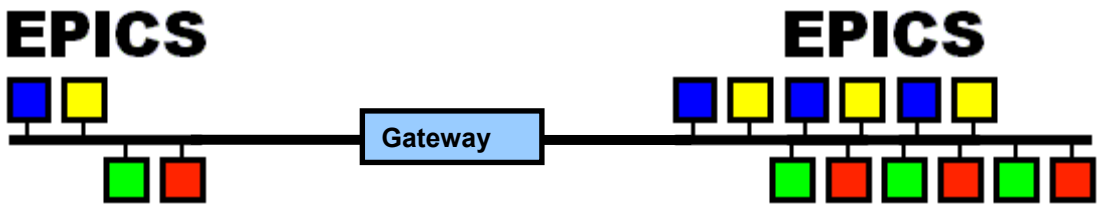
- The “service” that a ***server*** provides is access to a ***Process Variable***.

- A ***Process Variable*** (PV) is a named piece of data.



# Control System Architecture

- By default, PV traffic is constrained to a single subnet, but configuration options can direct traffic elsewhere.
- Physical hierarchies can be implemented using switches, routers, and gateways.
- EPICS gateways can be applied to limit access to Process Variables.



# Control System Architecture

- **Channel Access** is EPICS original protocol for communication
  - **PV Access** is an upgrade (included in EPICS 7)
  - Channel Access still fully supported
    - We will be using Channel Access as a reference in presentations
- Detailed operation of either not important as they both work out of the box
- Both protocols created to transfer data over the network (in form of **Process Variable**)
- Both protocols have libraries (APIs) for multiple languages:
  - Java, C/C++, Python, …

# Control System Architecture

- Process Variable
  - A ***Process Variable*** (PV) is a named piece of data associated with the machine (e.g. status, readback, setpoint, parameter).
  - Examples of PV names and values:
    - S1:VAC:reading `3.2e-08 torr`
    - LINAC:BPM4:xPosition `-0.323 mm`
    - BOOSTER:gateValvePosition `'OPEN'`
    - S3:DIPOLE:PS:setPoint `123.4 Amps`
    - APS:Mode `'Stored Beam'`
    - BL3:HISTOGRAM `{3, 8, 1, 2, 56, 44, 32, 43, 3, 5, 1}`

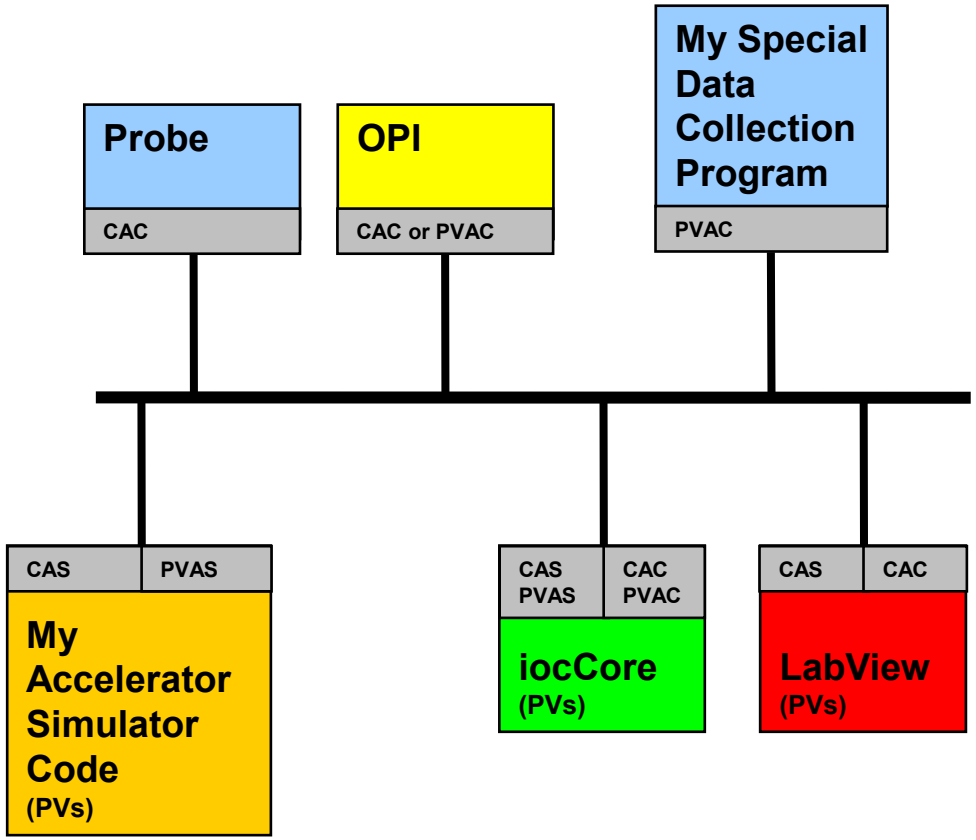
# Control System Architecture

- Process Variable
  - A ***Process Variable*** is a named piece of data with a set of attributes.
  - Examples of Attributes:
    - Alarm Severity (`NO_ALARM, MINOR, MAJOR, INVALID`)
    - Alarm Status (e.g. `LOW, HI, LOLO, HIHI, READ_error`…)
    - Timestamp
    - Number of elements (array)
    - Normal Operating Range
    - Control Limits
    - Engineering Unit Designation (e.g. degrees, mm, MW)

# Control System Toolkit

- Any tool/program/application that abides by the Channel Access (or PV Access) protocol could be described as “EPICS Compliant”.

- EPICS can be viewed as a “toolkit” of EPICS compliant programs.
  - One can select the appropriate tool for their need or develop their own.

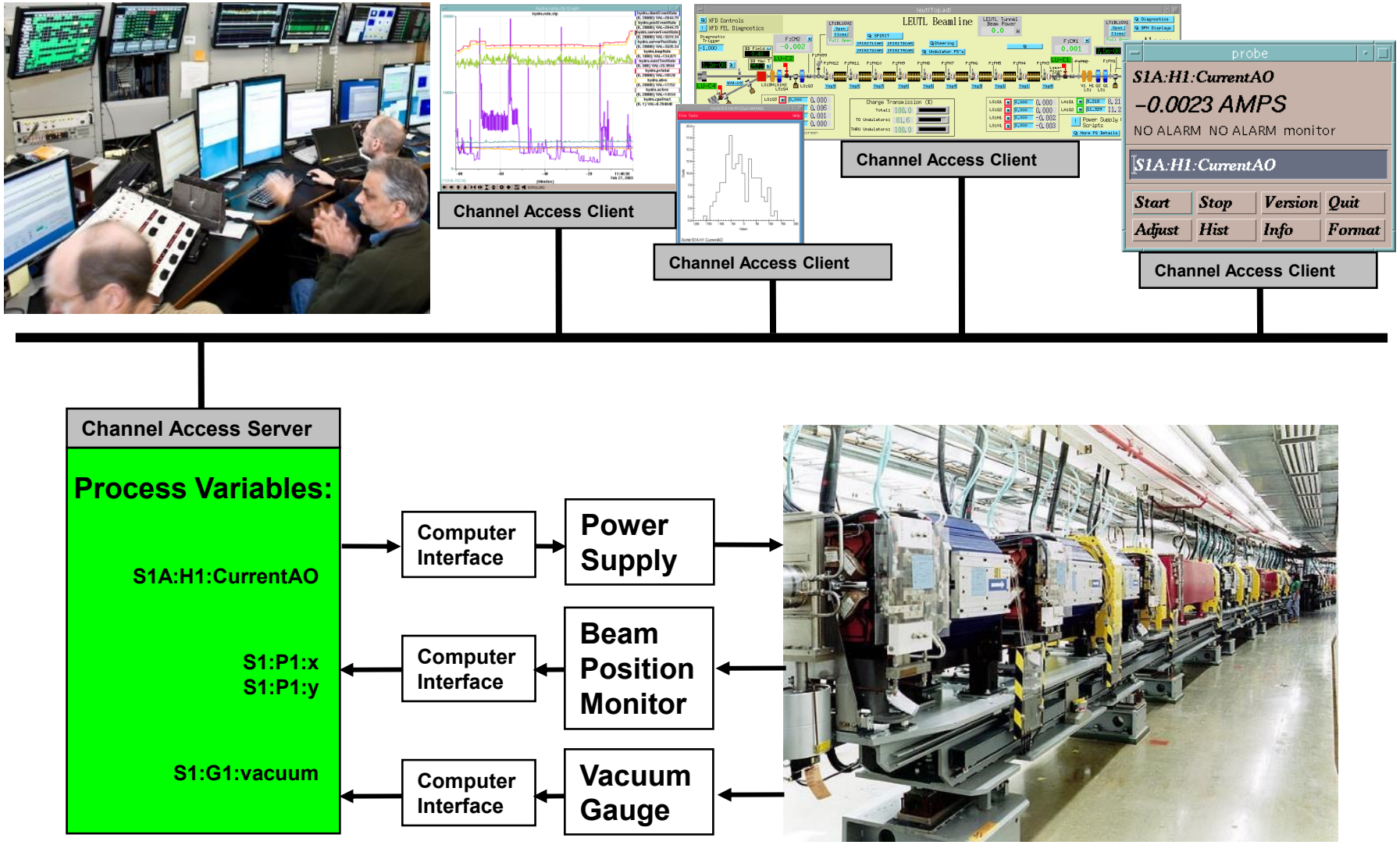


# So what does EPICS do?

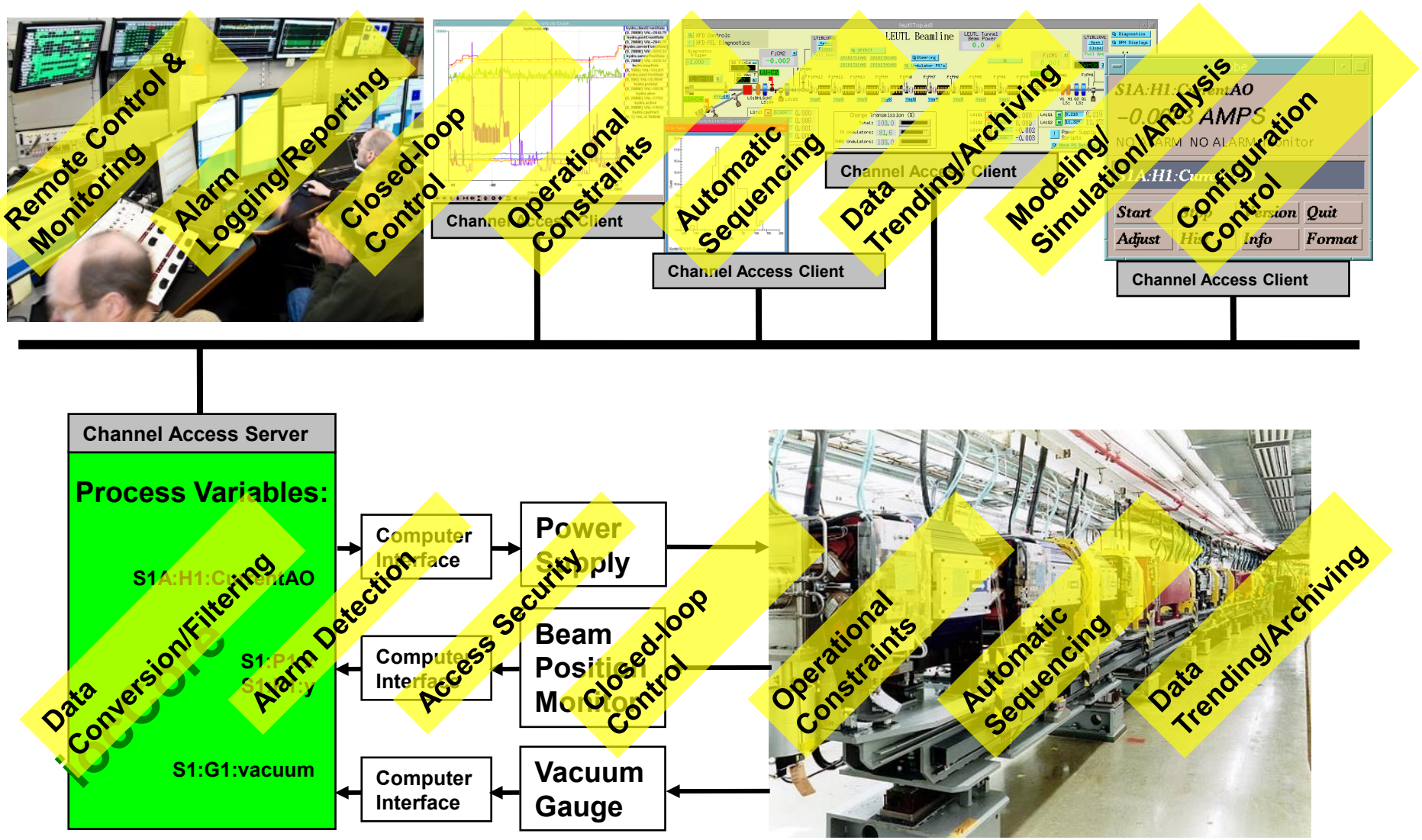
EPICS tools are available to accomplish almost any typical Distributed Control System (DCS) functionality, such as:

- Remote Control & Monitoring of Technical Equipment
- Data Conversion/Filtering
- Closed Loop Control
- Access Security
- Equipment Operation Constraints
- Alarm Detection/Reporting/Logging
- Data Trending/Archiving/Retrieval/Plotting
- Automatic Sequencing
- Mode & Facility Configuration Control (save/restore)
- Modelling/Simulation
- Data Acquisition
- Data Analysis

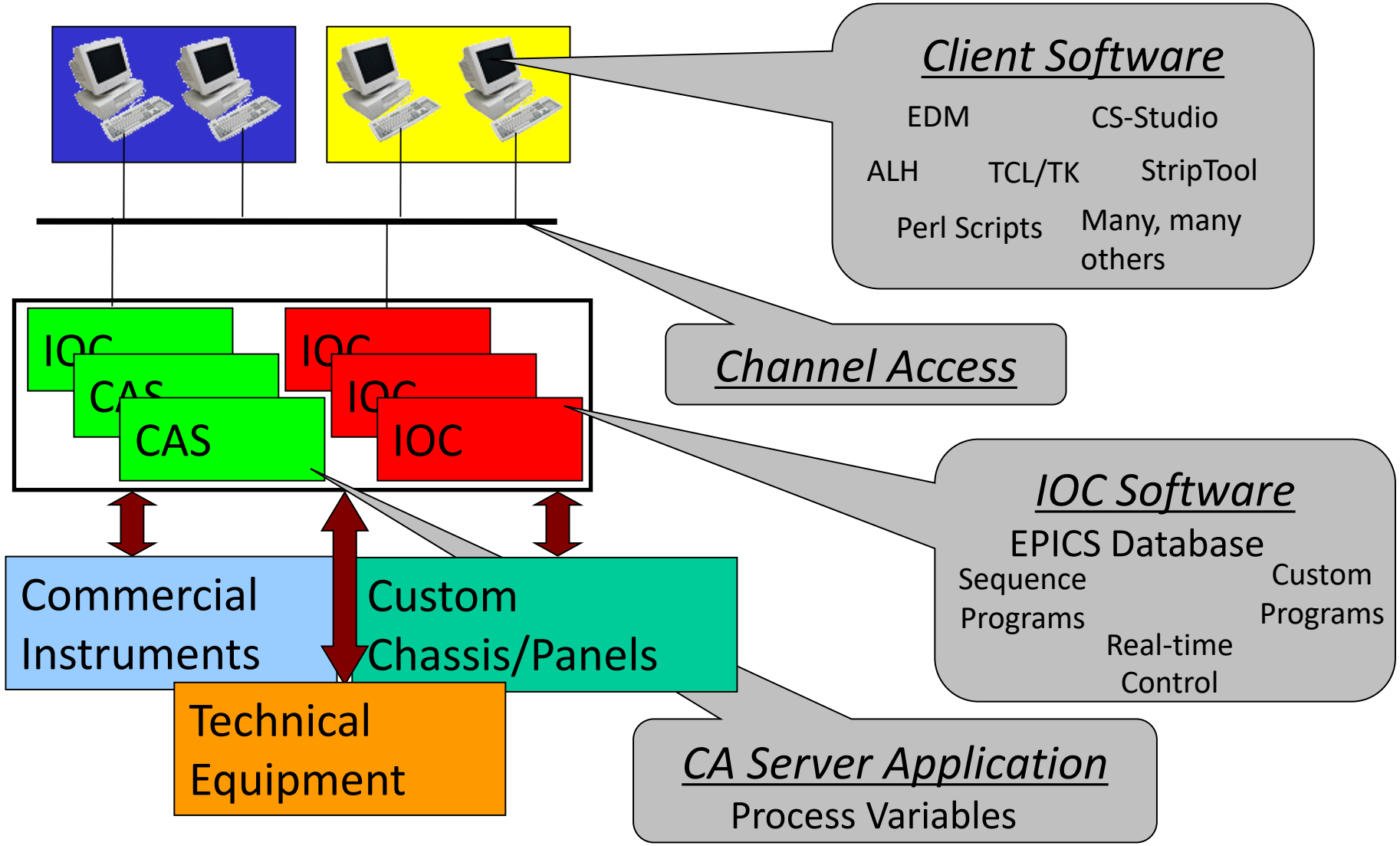
# How does EPICS do it?



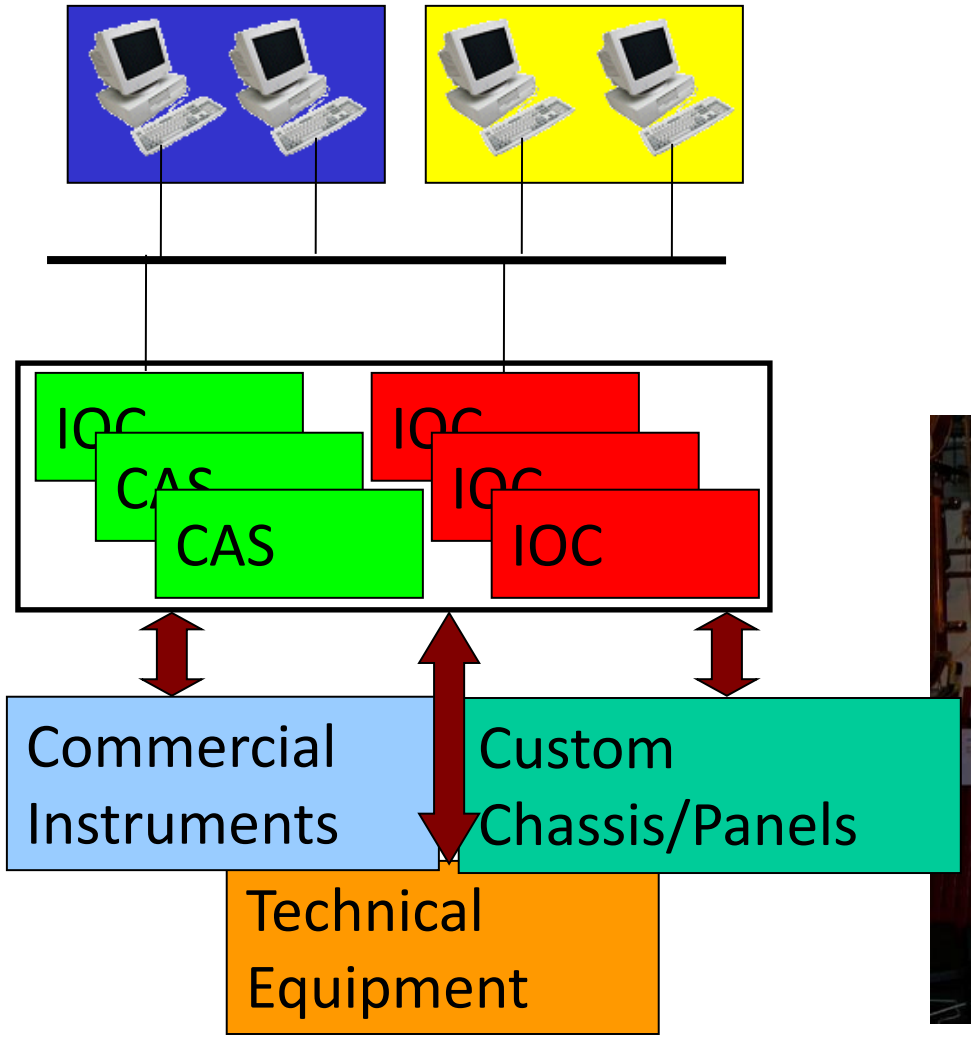
# Where does EPICS do it?

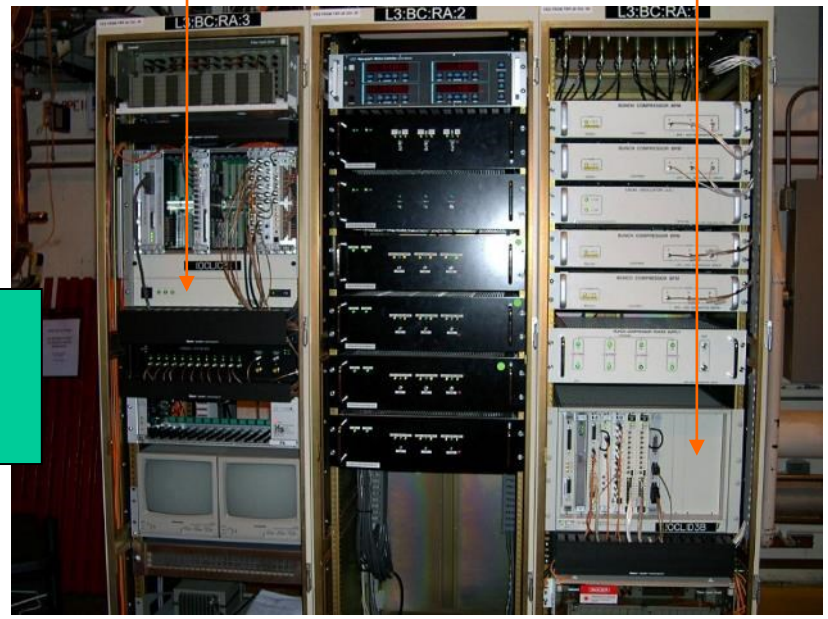


# Canonical Form of an EPICS Control System

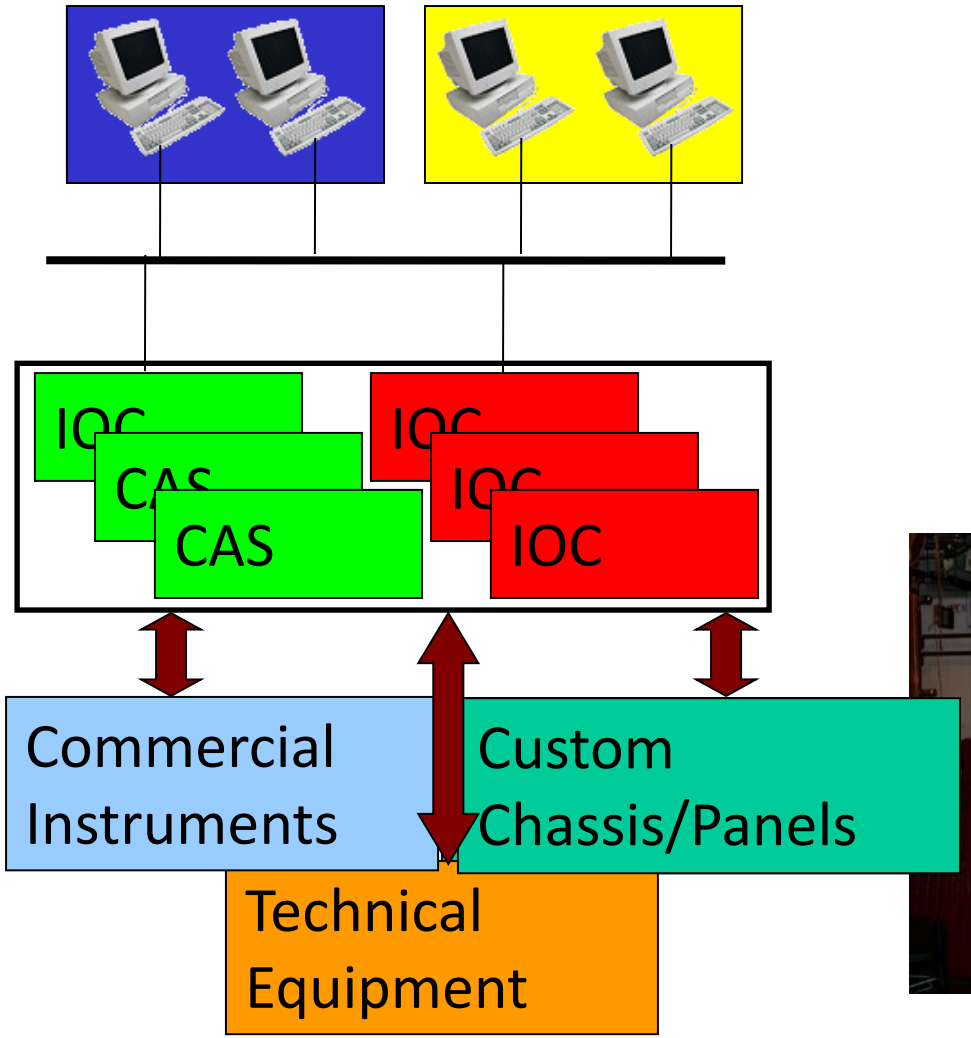


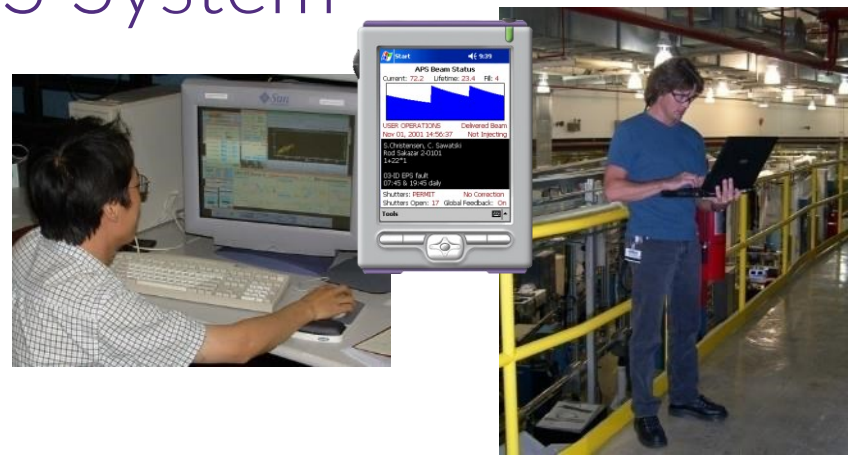
# Historical Realizations of an EPICS System

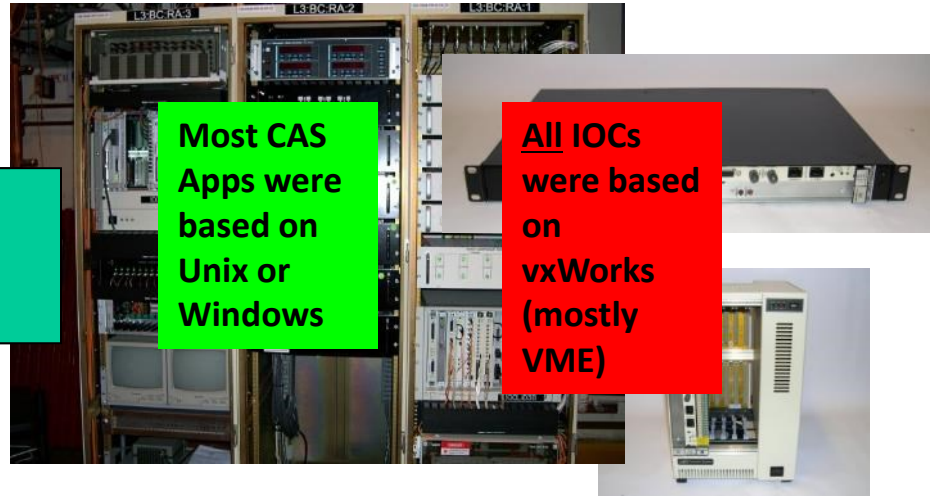


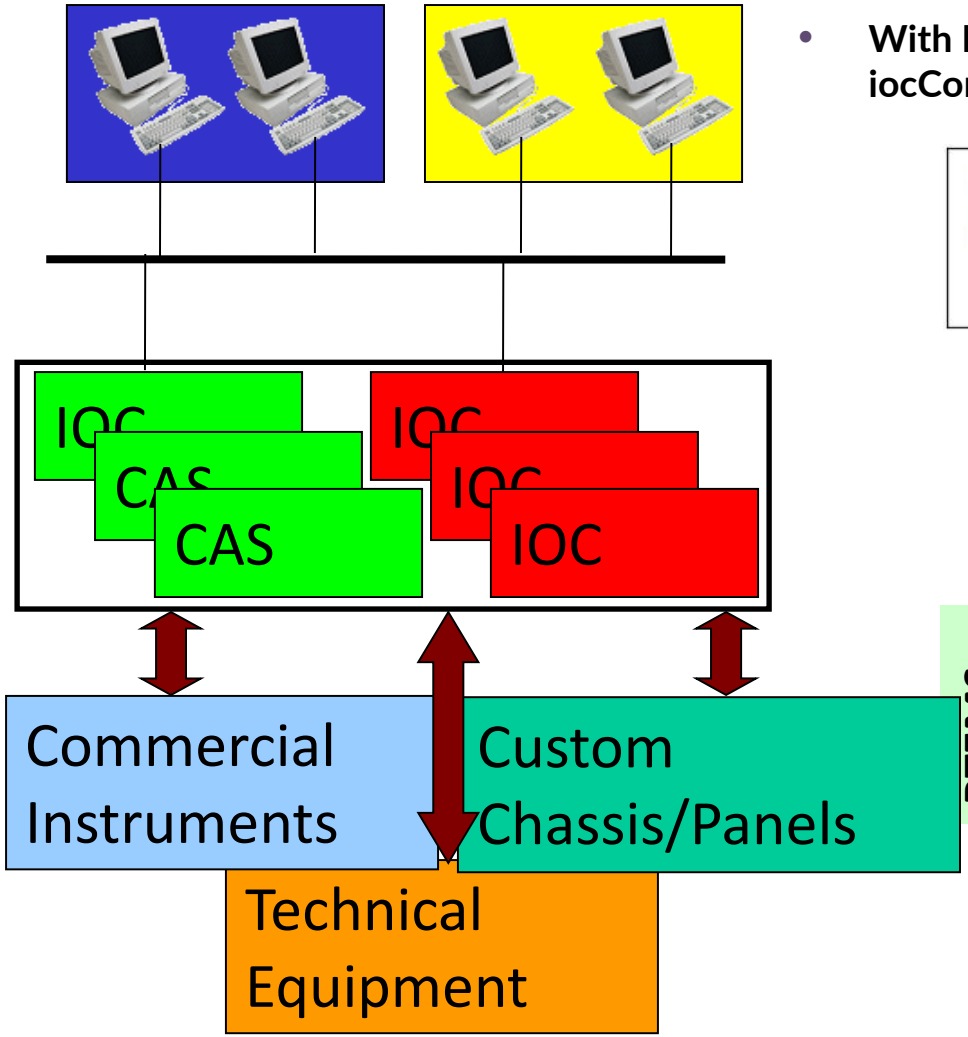
# Historical Realizations of an EPICS System



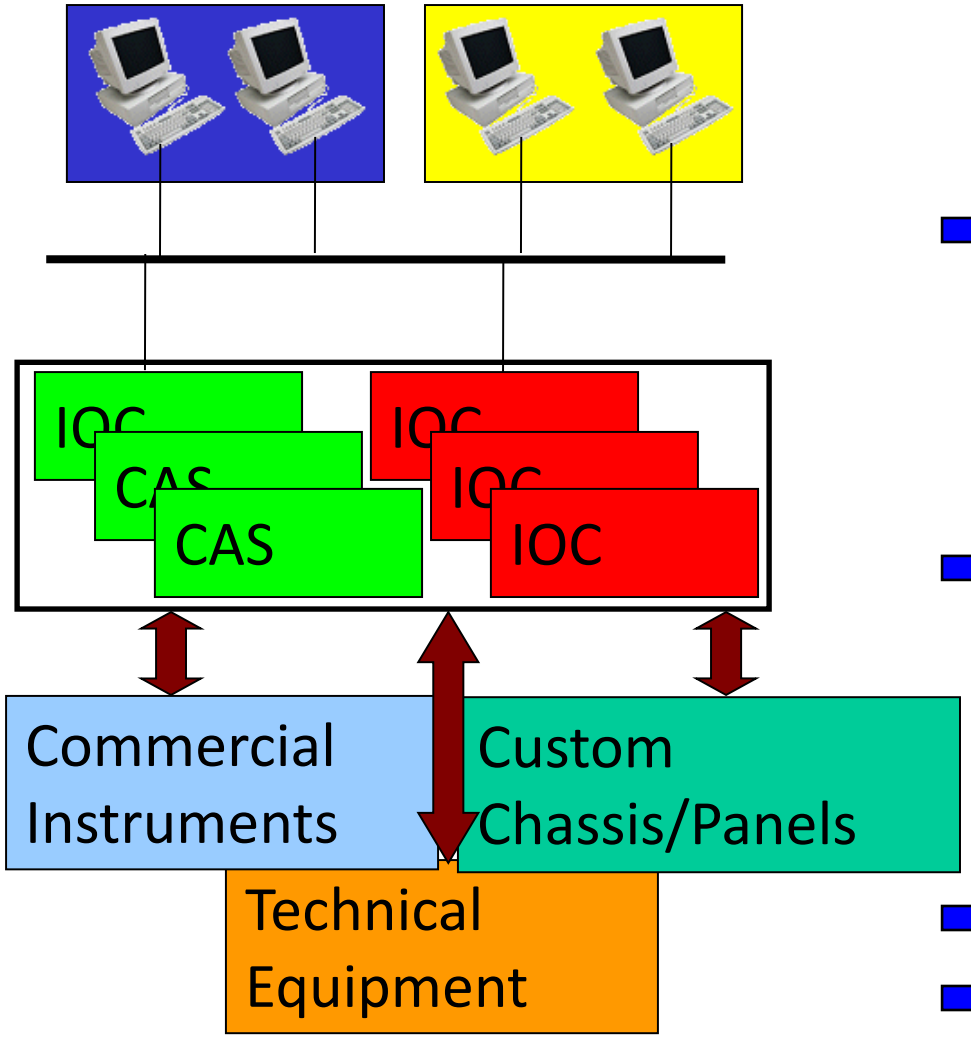


# Historical Realizations of an EPICS System




- **With Release 3.14, the operating system limitations for iocCore have been removed.**

# Typical Realizations of an EPICS System

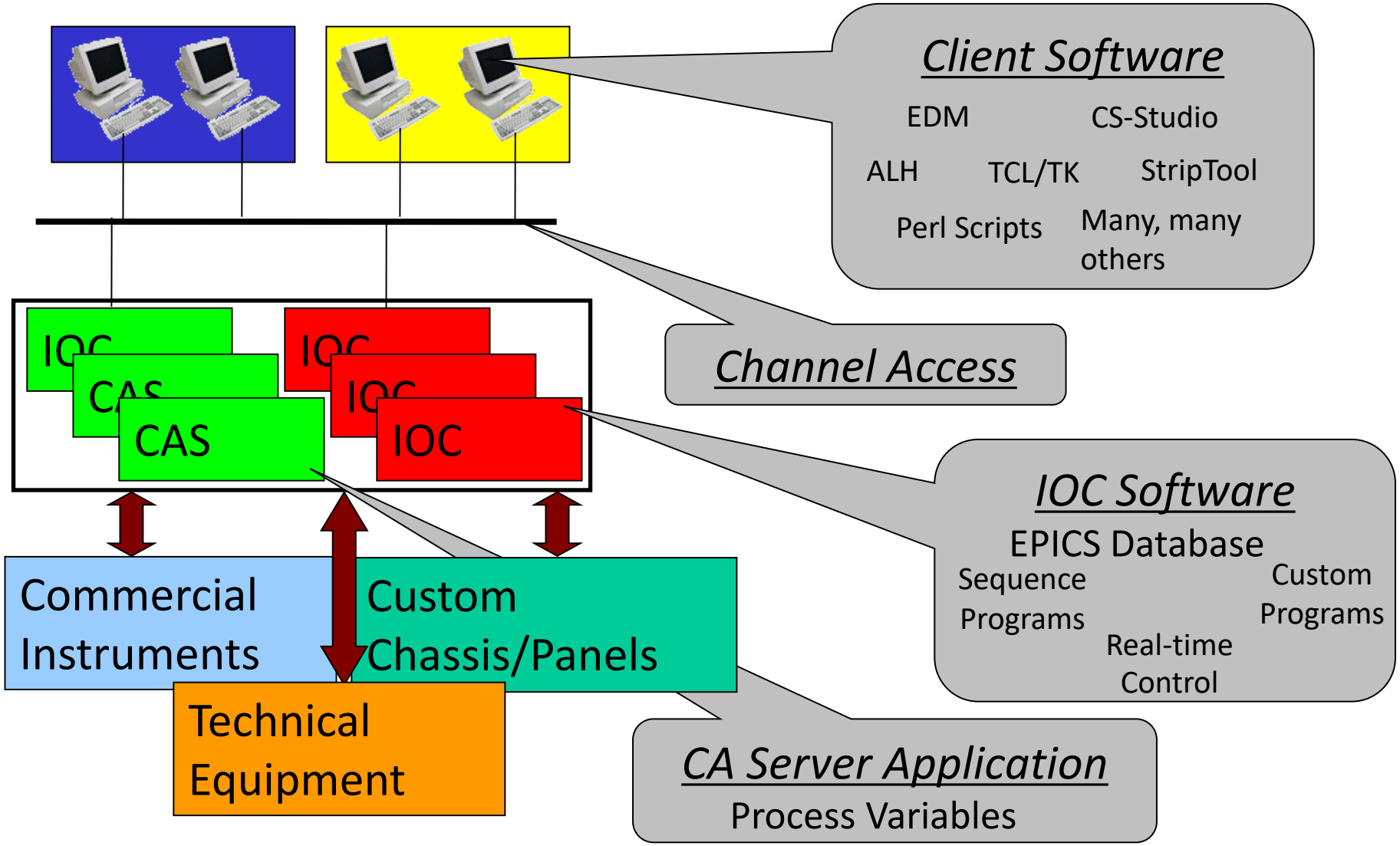



**Driving a motor with EPICS**

circa 1995

circa 2002

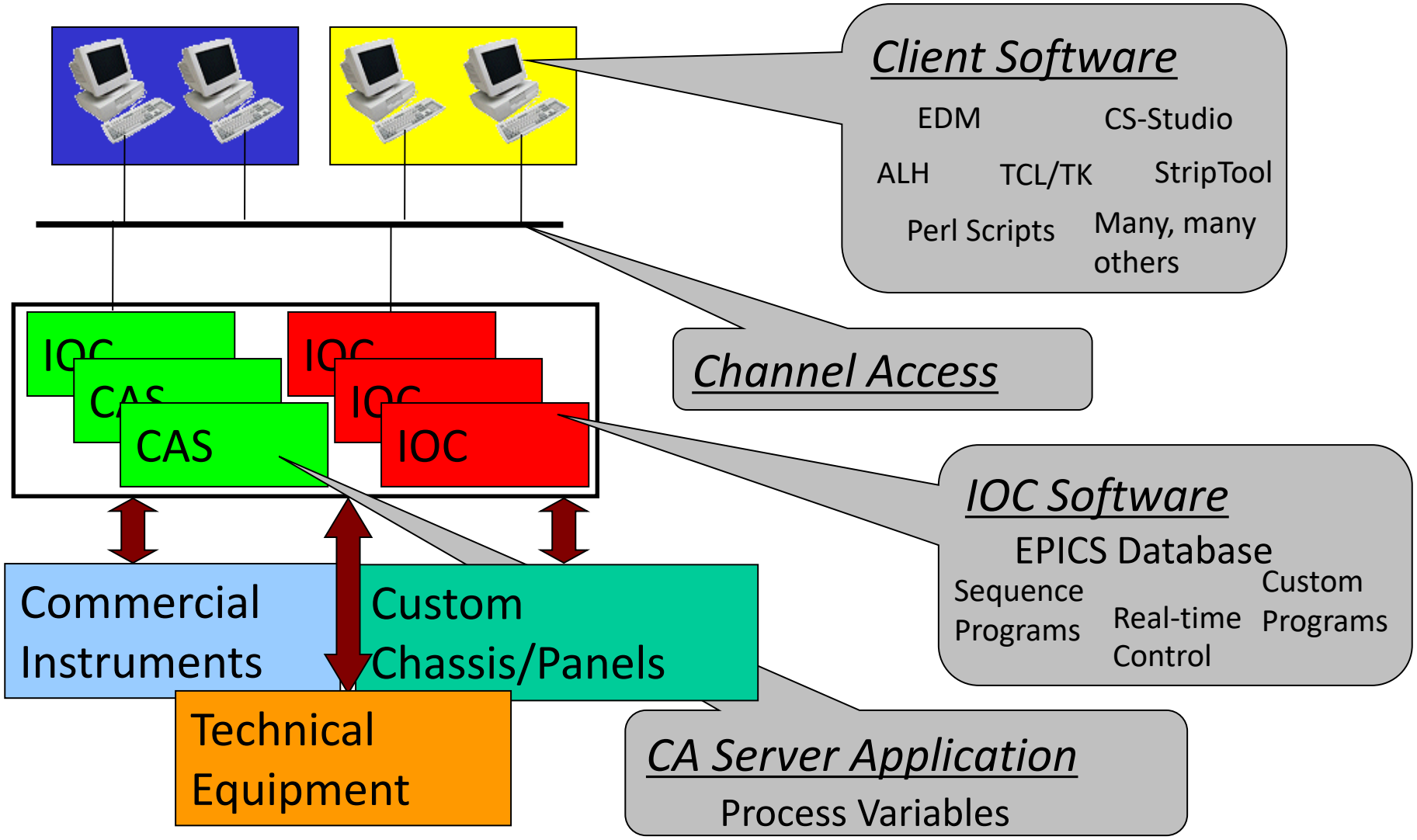
# Canonical Form of an EPICS Control System



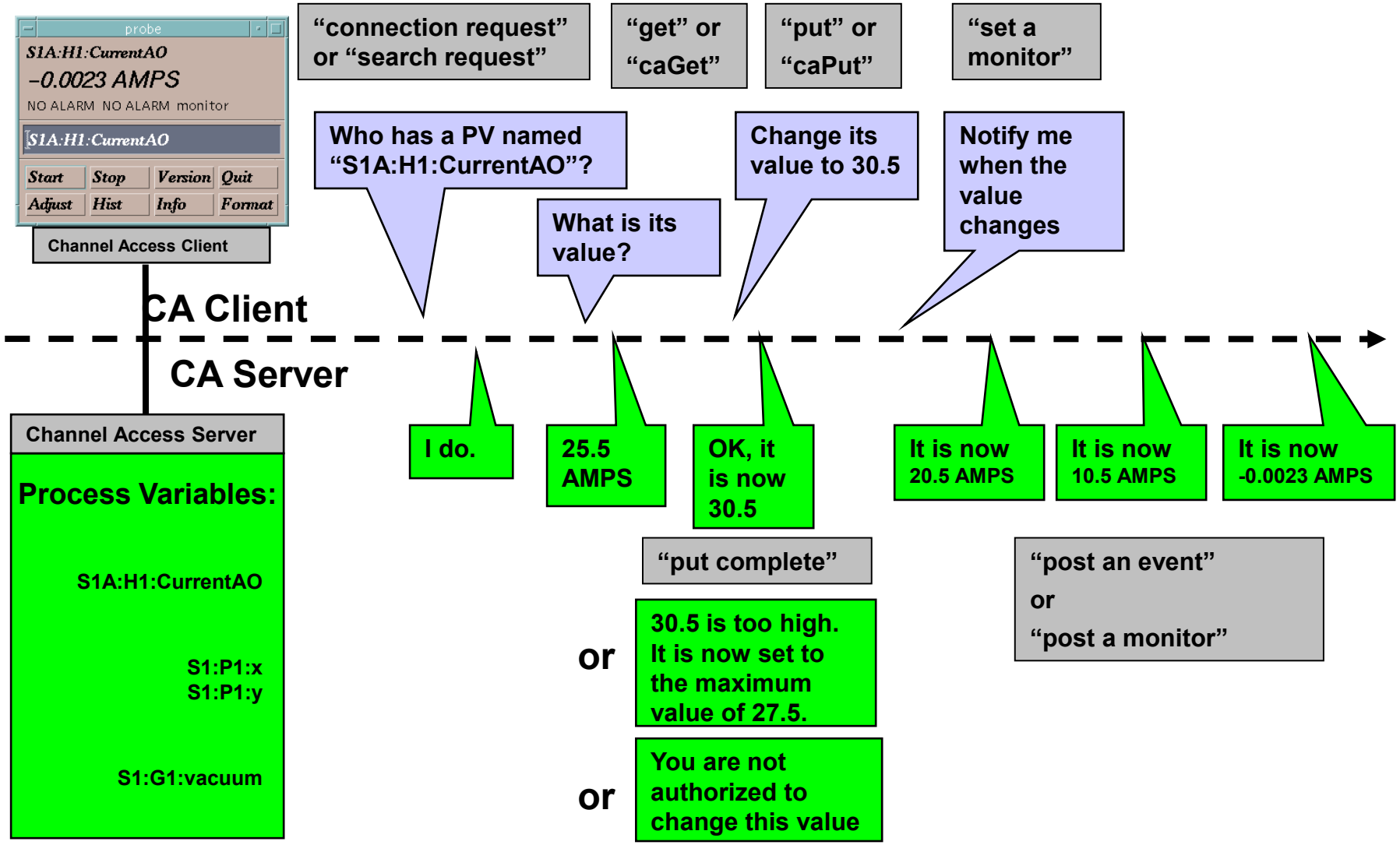
# Standalone CA Clients (from EPICS Website)

- ALH: Alarm Handler
- Archiver Appliance
- CAPod: Channel Access projects for Apple iOS devices (SF)
- caQtDM: An MEDM replacement based on Qt (PSI)
- CASR: Host-based Save/Restore
- CSS: Control System Studio (SF)
- EDM: Extensible Display Manager (ORNL)
- Kryten: Tool to run commands on PV changes (GitHub)
- MEDM: Motif Editor and Display Manager
- Probe: Motif Channel Monitoring program
- StripTool: Strip-chart plotting tool
- Others: link

# Canonical Form of an EPICS Control System



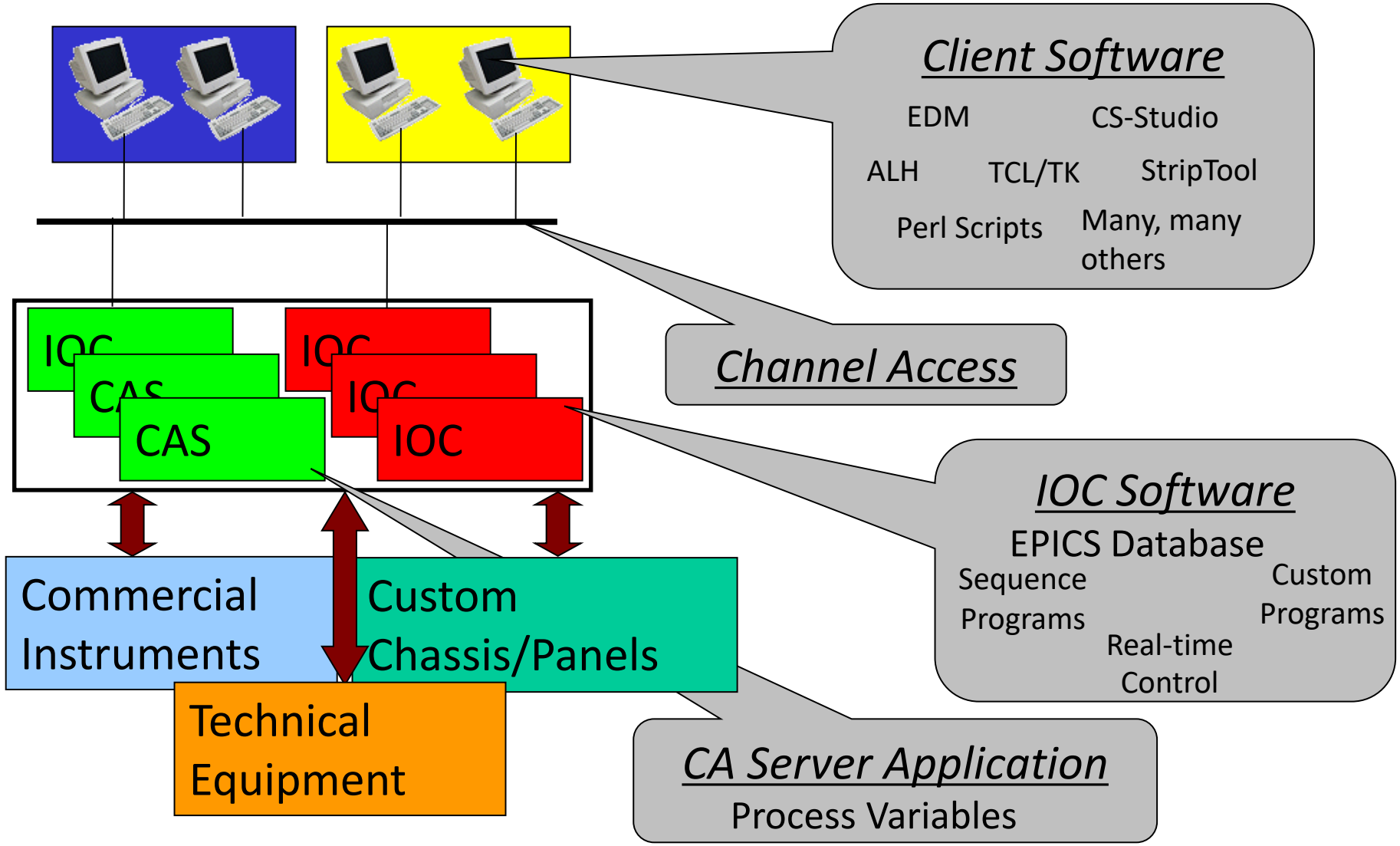
# Channel Access in One Slide



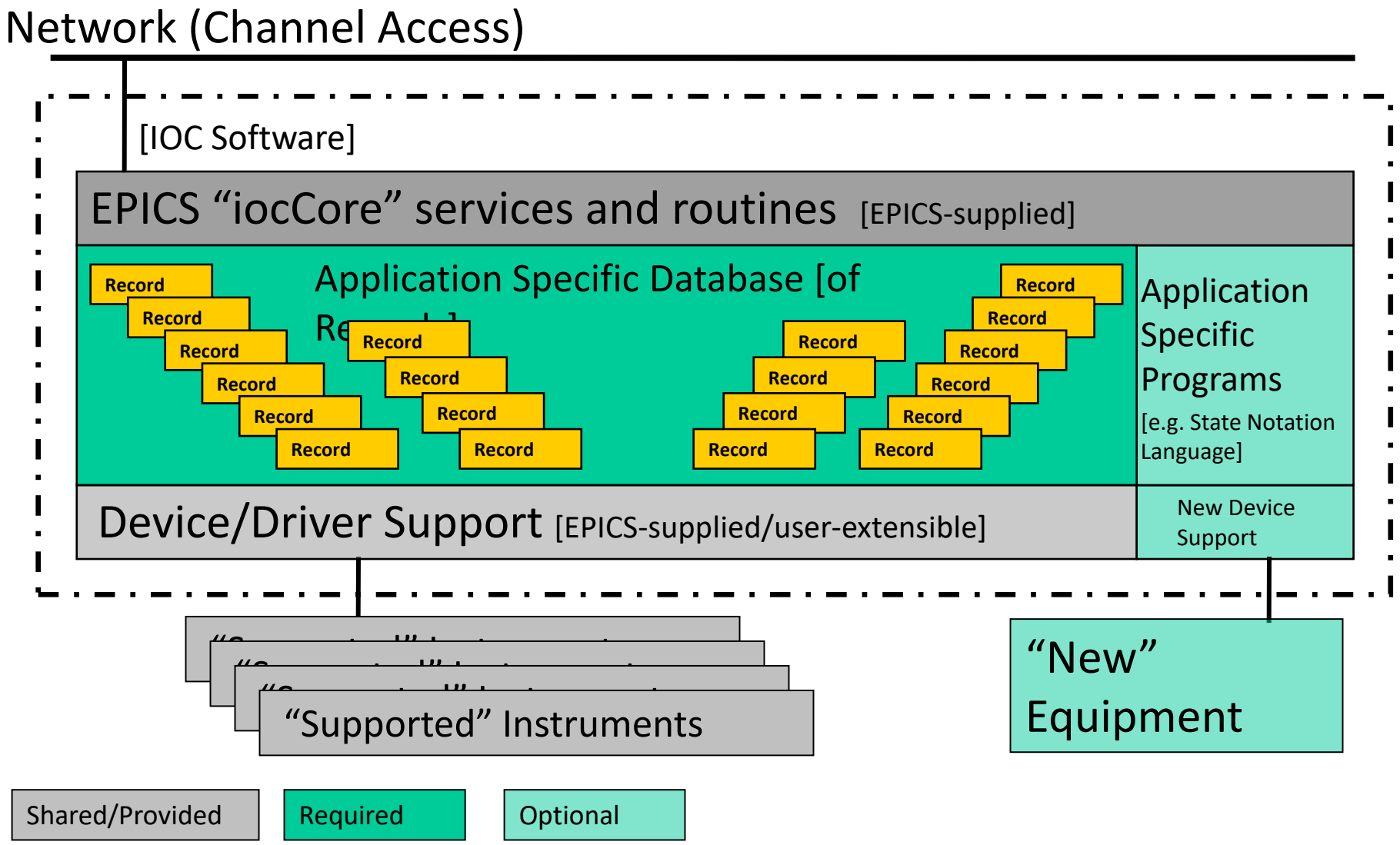
# Key Features of Channel Access

- Clients broadcast PV names to find the server in which they exist.
- Channel Access Security can be applied to limit access to Process Variables.
- Clients can wait until a '*put request*' is completed before proceeding.
- Clients can '*set monitors*' on PVs and will then be notified when the value changes.

# Canonical Form of an EPICS Control System



# IOC Software in One Slide

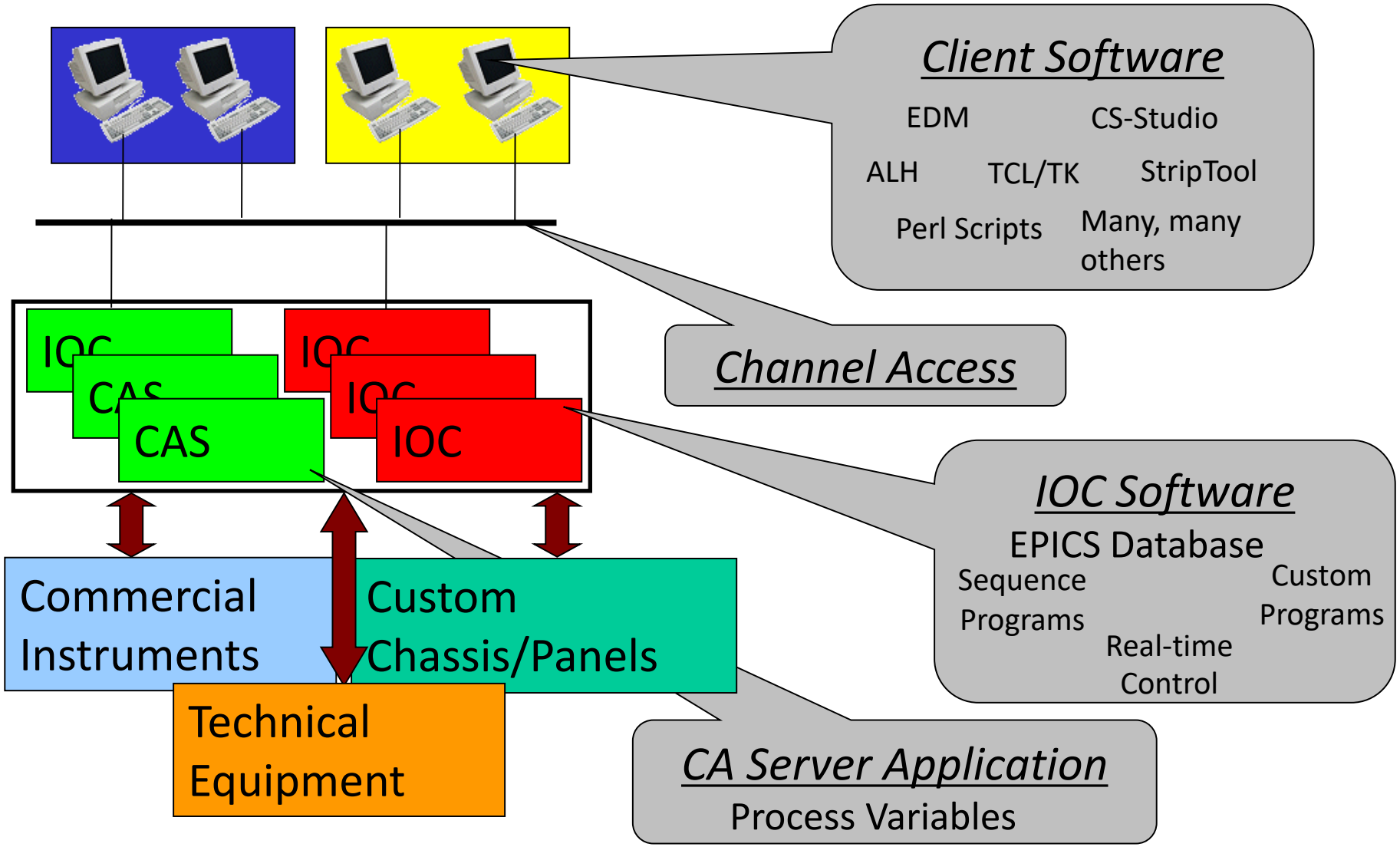


COSYLAB

# Key Features of IOC software

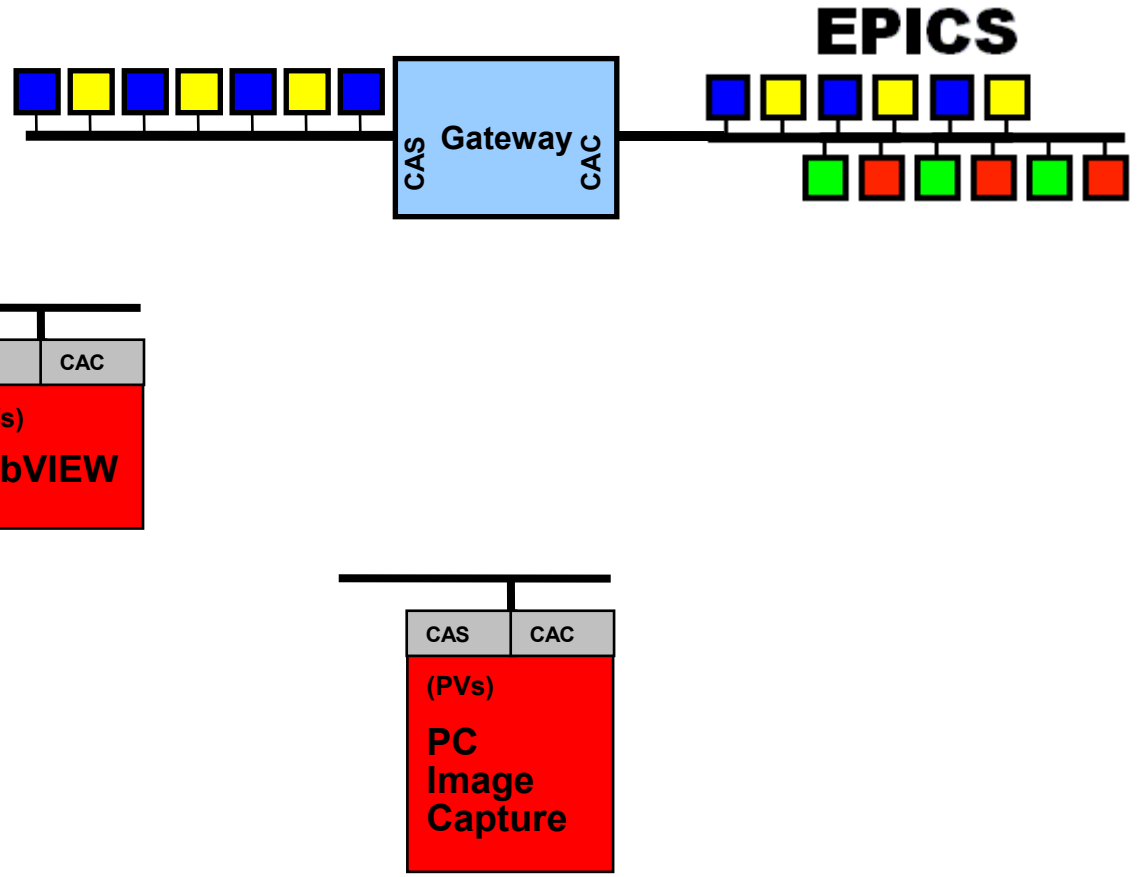
- Two primary application specific components:
  - The real-time database of records (required)
  - State Notation Language programs used to implement state-oriented programs (finite-state machine)
- Machine status, information and control parameters are defined as “records” in the application specific database.
- The data within a record is accessible via Process Variables.
- Records have some functionality associated with them (scaling, filtering, alarm detection, calculations, etc.). Different record types have different functions and uses.
- Records are frequently associated with I/O equipment that requires unique “device support” for that instrument.

# Canonical Form of an EPICS Control System



# Popular CA Server Applications

- IOC Core
- PV Gateway
- CA Server Interface to LabVIEW
- CA Server Interface to PC Image Acquisition Systems



COSYLAB

# Ten really neat things about EPICS

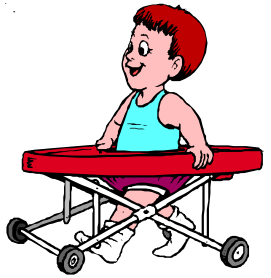
1. It's free.
2. It's Open Source.
3. There are lots of users.
4. All a client needs to know to access data is a PV name.
5. You can pick the best tools out there …
6. … or build your own.
7. The boring stuff is already done.
8. There is a lot of expertise available close by.
9. A good contribution becomes internationally known.
10. By following a few simple rules, you get a lot for free.

COSYLAB

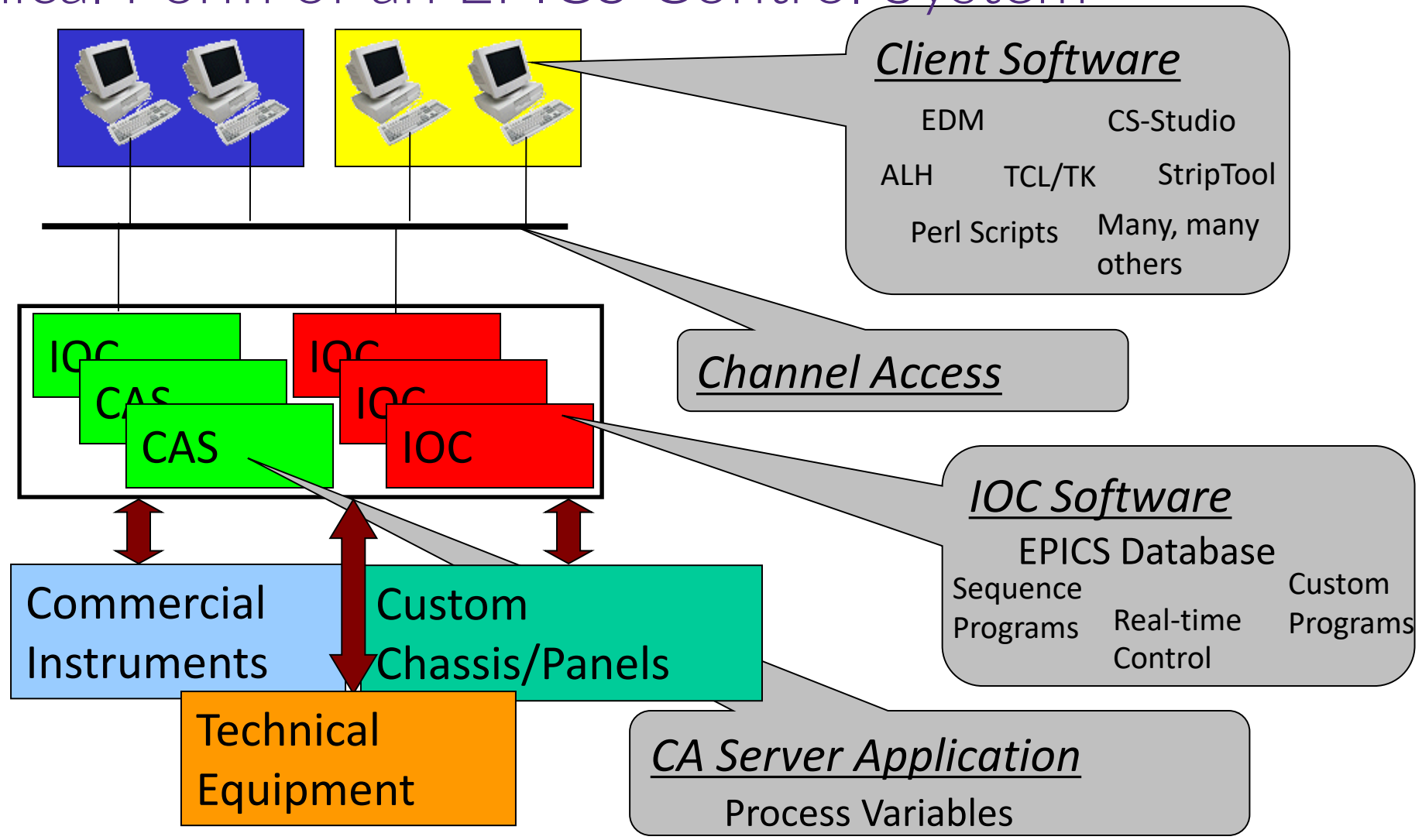
# EPICS Foundations

# Overview

- Lay the foundation for understanding an EPICS control system
- Introduce IOCs
  - Channel Access (CA)
  - Database
  - Sequencer
  - Device Support
- Choosing the correct tools for the job
  - When to use a database
  - The sequencer, what is it good for?
  - Why write your own CA client program?
- How fast is EPICS?
- How to find more information
  - Website walk through



# Canonical Form of an EPICS Control System

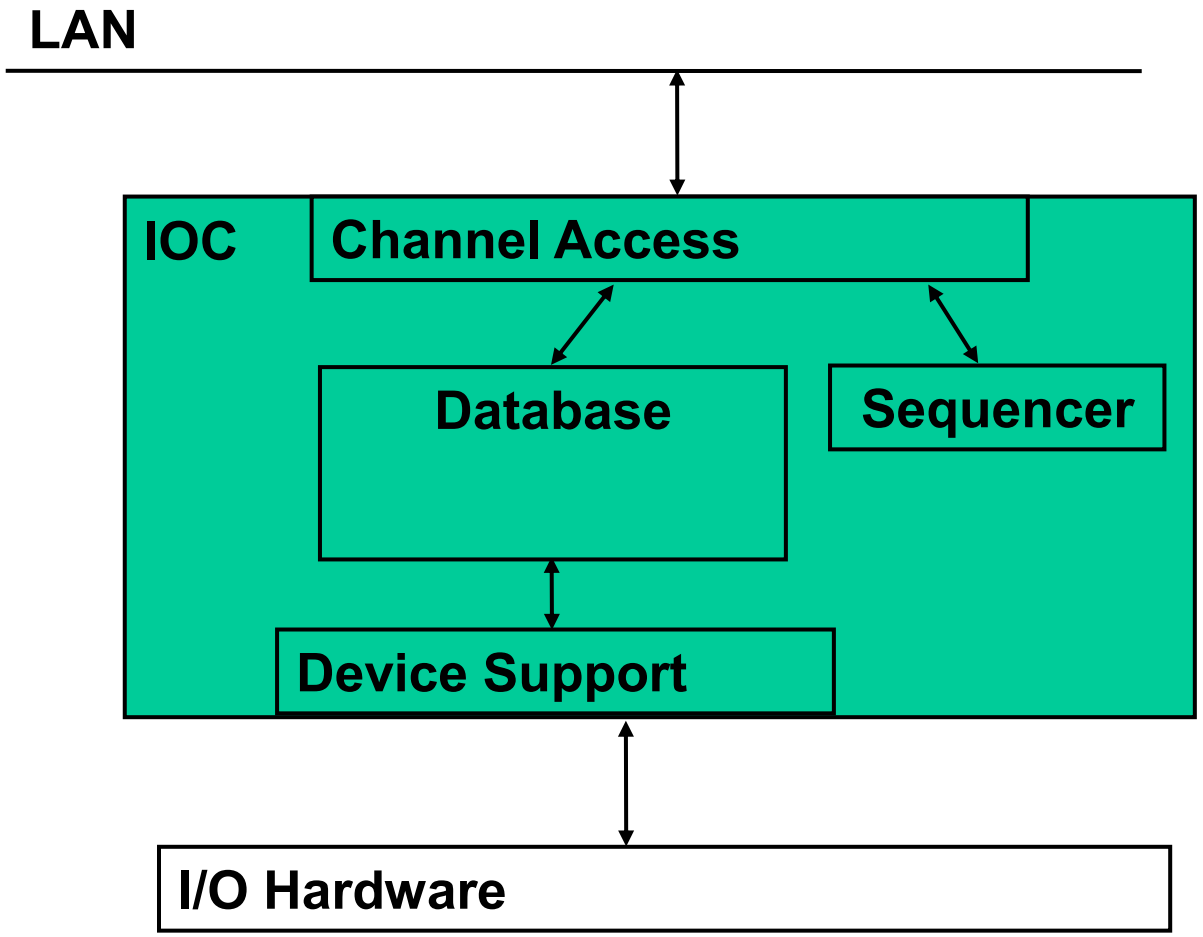


# Introducing the IOC

- Input Output Controller
- A computer running software called “IOC Core”
- The computer can be:
  - VME based, running VxWorks (only choice until Release 3.14) or RTEMS
  - PC running Windows, Linux, RTEMS
  - Apple running OSX
  - UNIX Workstation running Solaris
- Usually has Input and/or Output devices attached
- An EPICS control system must consist of at least one Channel Access Server (usually an IOC)
- An IOC has one or more databases loaded.
  - The database tells it what to do

# Inside an IOC

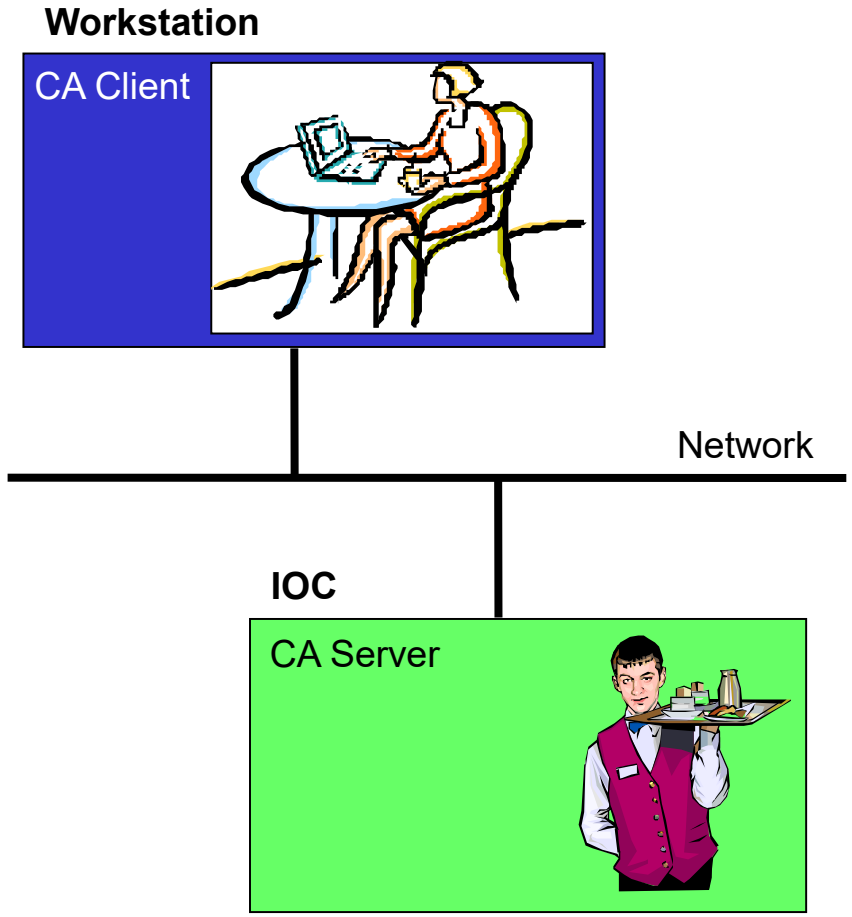
- The major software components of an IOC (IOC Core)



# Channel Access

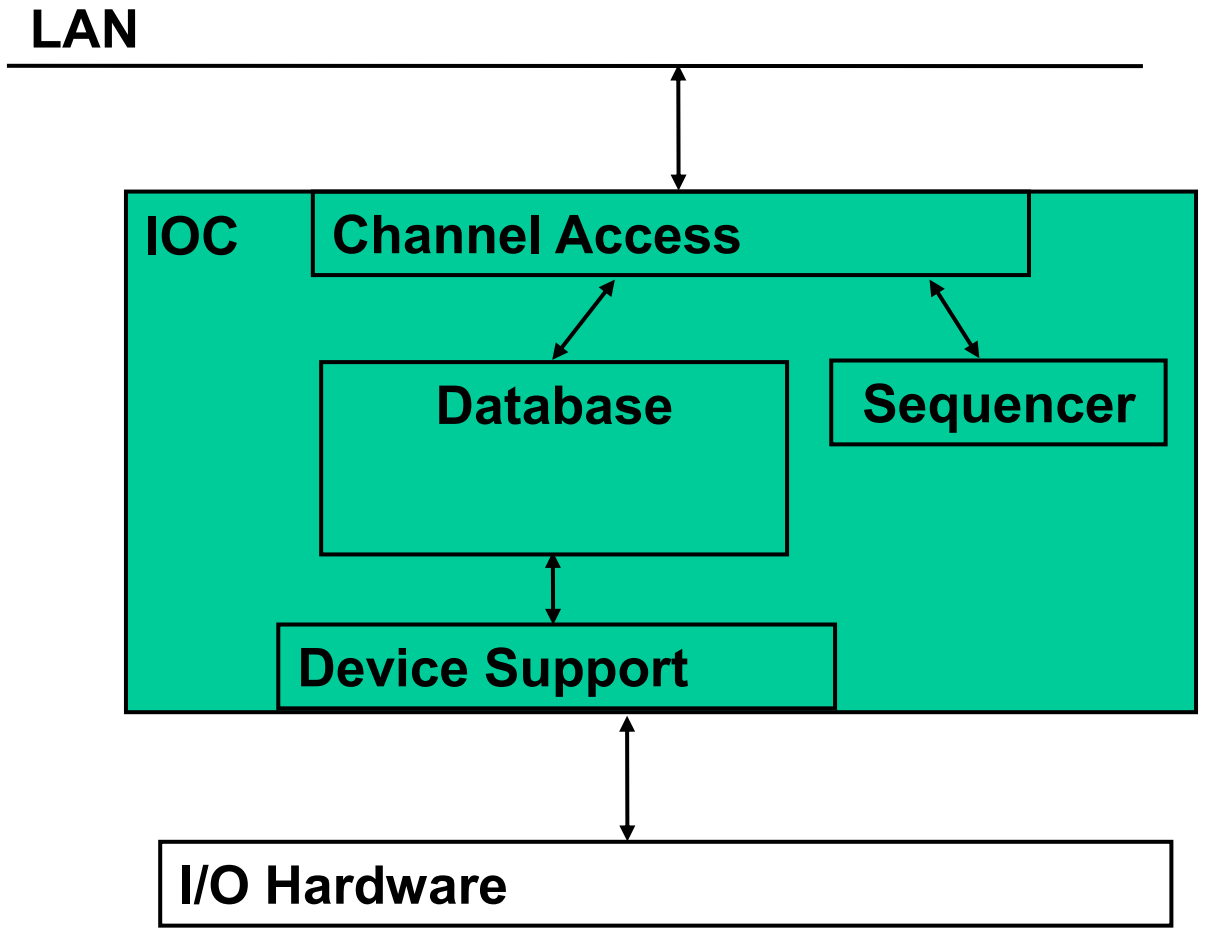
- Allows other programs (CA Clients) to see and change values of Process Variables in an IOC (CA Server)
- CA Clients may
  - Put (write)
  - Get (read)
  - Monitor

  data of Process Variables
- IOCs are both CA clients and CA servers. They can interact with data in other IOCs
- A CA Client can connect to many servers
- A CA Server may serve many clients
- A very efficient and reliable protocol



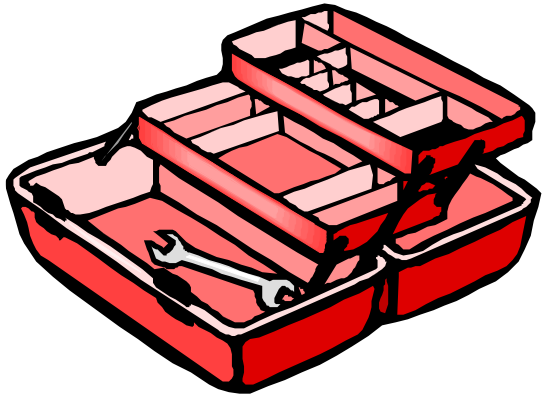
# Inside an IOC

- The major software components of an IOC (IOC Core)



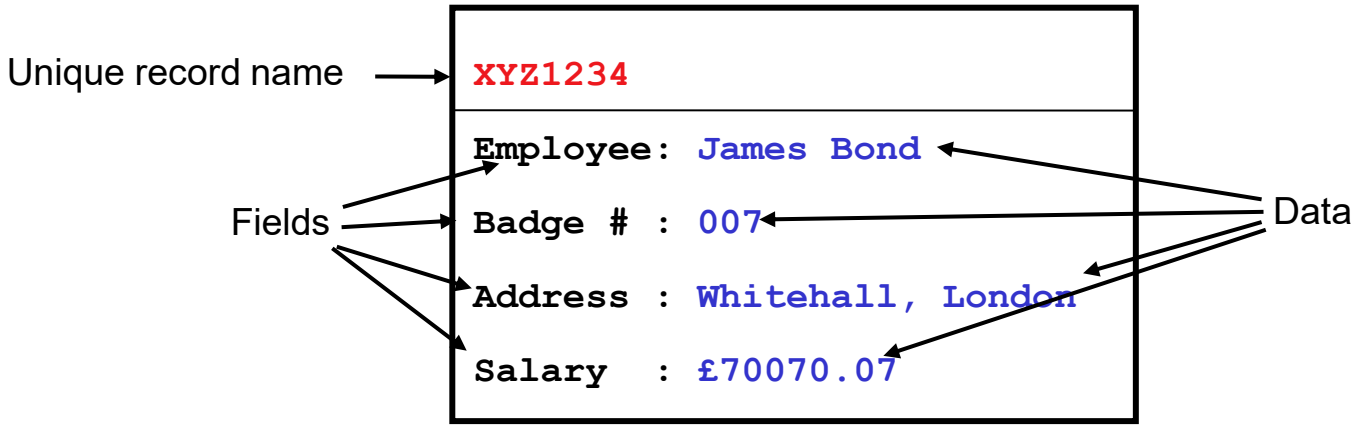
# EPICS Databases – What are they for?

- Interface to process instrumentation
- Distribute processing
- Provide external access to all process information
- Use common, proven, objects (records) to collect, process and distribute data
- Provide a common toolkit for creating applications



# What are records?

- A record is an object with
  - A unique name
  - Properties (fields) that contain information (data)
  - The ability to perform actions on that data
- For example, a personnel record in a relational database has a name, and fields containing data.

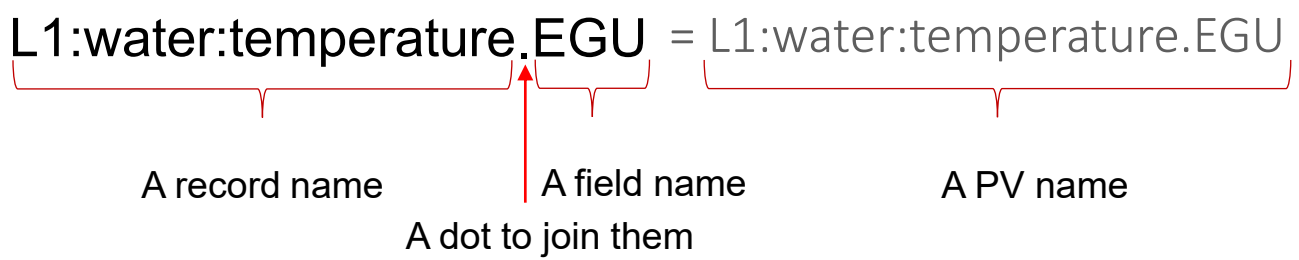


# What are EPICS records?

- A record is an object with…
  - A unique name e.g. **`S28:waterPressure`**.
  - Controllable properties (fields) e.g. **`EGU`**.
  - A behavior - defined by its record type.
  - Optional associated hardware I/O (device support).
  - Links to other records.
- Each field can be accessed individually by name.
- A record name and field name combined give the name of a process variable (PV).
- A Process Variable name is what Channel Access needs to access data.

# A Process Variable Name

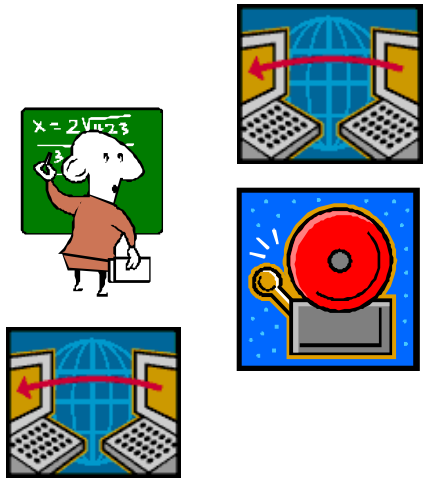
- A PV name is comprised of two parts
  - The record name, and
  - the name of a field belonging to that record
- For example…




- **Note**: If no field name is given, Channel Access will default to using the `.VAL` field
  - i.e. to CA, "`L1:water:temperature`" = "`L1:water:temperature.VAL`"

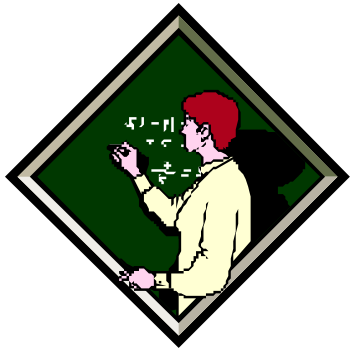
# What do records do?

- Records are active, they do things
  - Get data from other records or from hardware
  - Perform calculations
  - Check values are in range and raise alarms
  - Put data into other records or into hardware
  - Activate or disable other records
  - Wait for hardware signals (interrupts)
- What a record does depends upon its type and the values in its fields.
- A wide range of records have already been created.
- New record types can be added to a new application as needed.
- A record does nothing until it is *processed*.



# Record Types

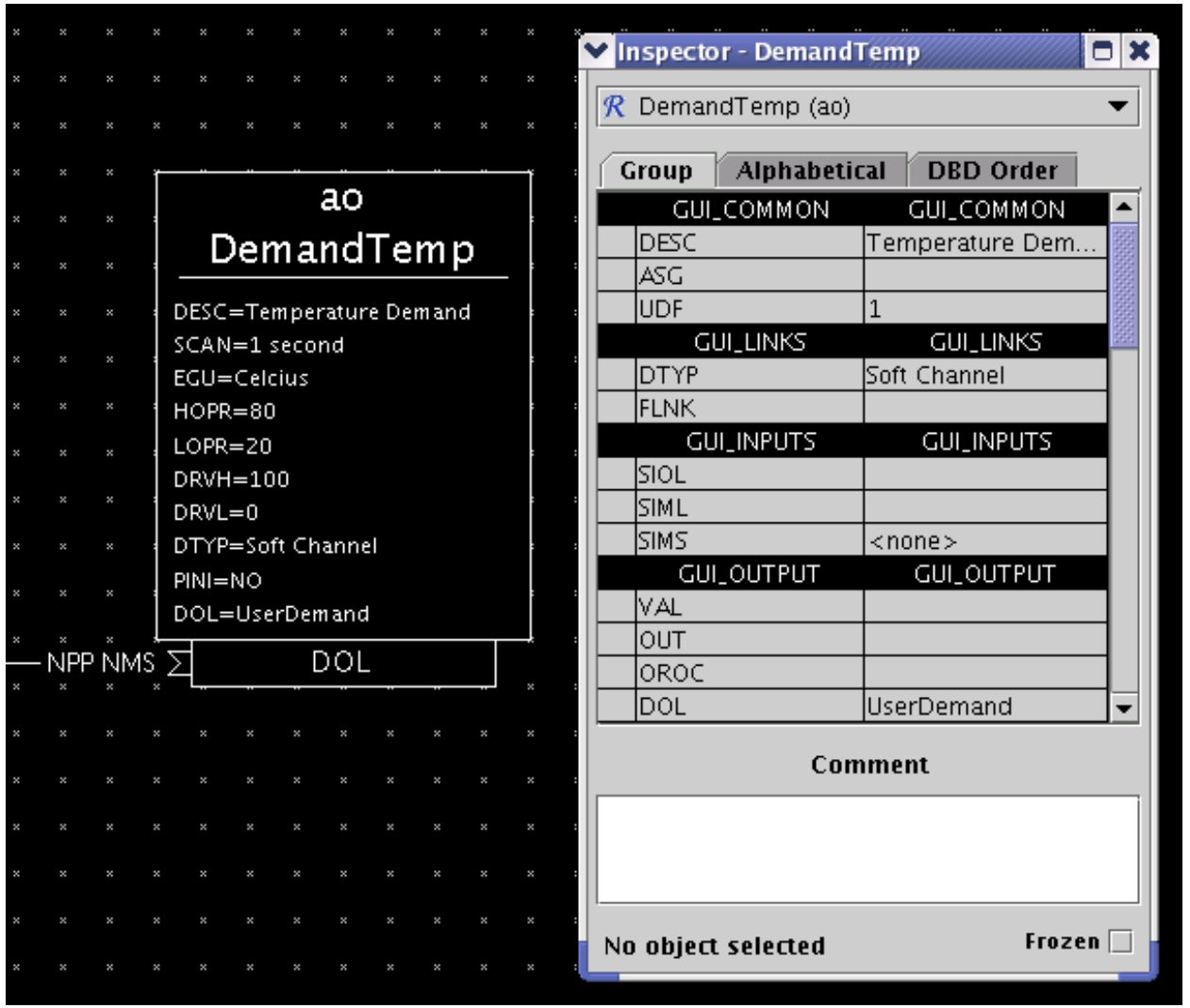
- Classified into four general types
- Input, e.g.
  - Analog In (AI)
  - Binary In (BI)
  - String In (SI)
- Algorithm/control, e.g.
  - Calculation (CALC)
  - Subroutine (SUB)
- Output, e.g.
  - Analog Out (AO)
  - Binary Out (BO)
- Custom, e.g.
  - Beam Position Monitor
  - Multi-Channel Analyzer



# Some Record Types

- Analog in
- Analog out
- Binary in
- Binary out
- Calculation
- Calculation out
- Compression
- Data fanout
- Event
- Fanout
- Histogram
- Motor
- Multi bit binary input
- Multi bit binary output
- PID control
- Pulse counter
- Pulse delay
- Scan
- Select
- Sequence
- String in
- String out
- Subarray
- Subroutine
- Waveform

# Graphical View of a Record



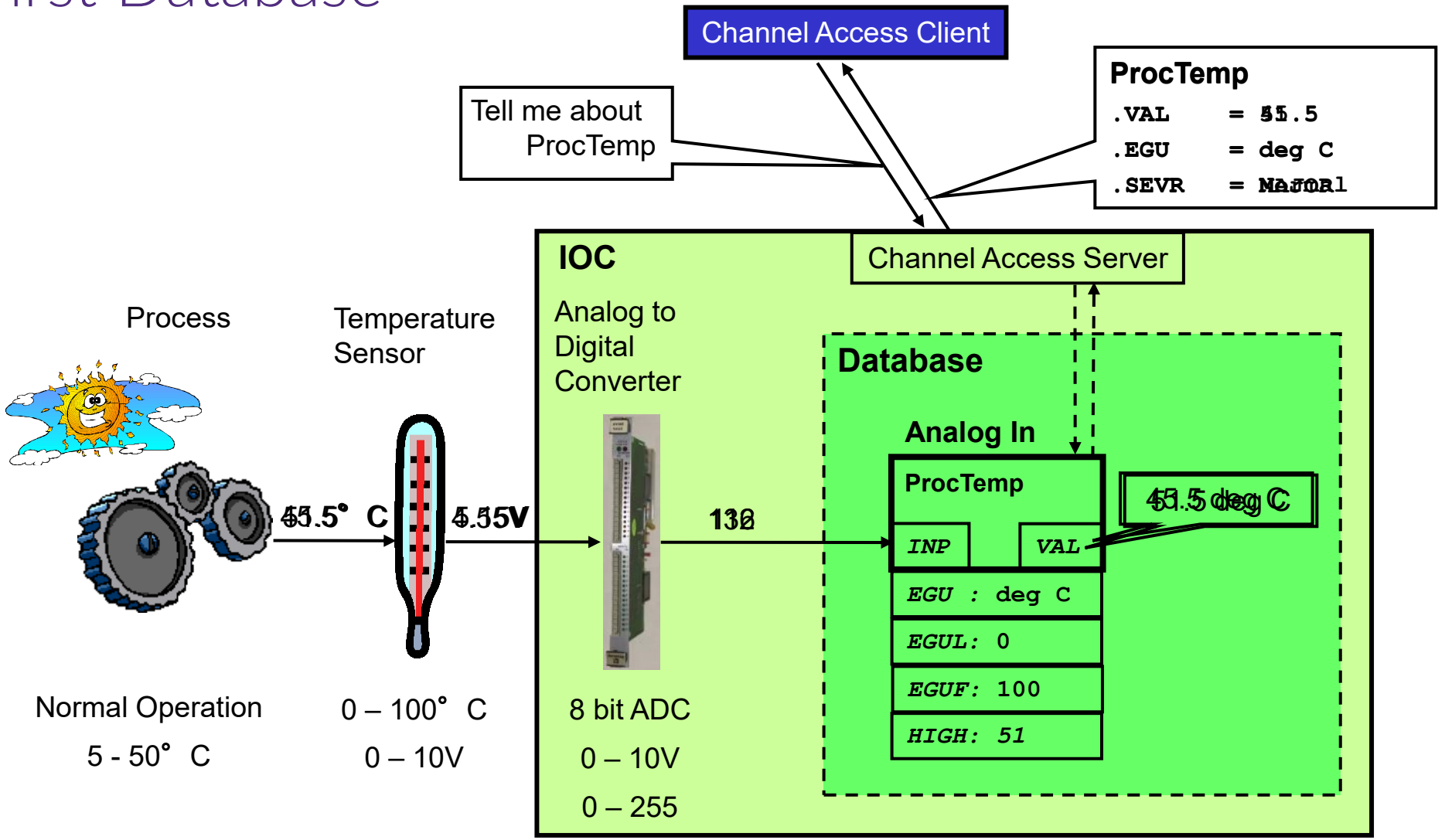
# IOC View of a Record

```
record(ao,"DemandTemp") {
    field(DESC,"Temperature")
    field(ASG,"")
    field(SCAN,"Passive")
    field(PINI,"NO")
    field(PHAS,"0")
    field(EVNT,"0")
    field(DTYP,"VMIC 4100")
    field(DISV,"1")
    field(SDIS,"")
    field(DISS,"NO_ALARM")
    field(PRIO,"LOW")
    field(FLNK,"")
    field(OUT,"#C0 S0")
    field(OROC,"0.0e+00")
    field(DOL,"")
    field(OMSL,"supervisory")
    field(OIF,"Full")
    field(PREC,"1")
    field(LINR,"NO CONVERSION")
    field(EGUF,"100")
    field(EGUL,"0")
    field(EGU,"Celsius")
    field(DRVH,"100")
    field(DRVL,"0")
    field(HOPR,"80")
    field(LOPR,"10")
    field(HIHI,"0.0e+00")
    field(LOLO,"0.0e+00")
    field(HIGH,"0.0e+00")
    field(LOW,"0.0e+00")
    field(HHSV,"NO_ALARM")
    field(LLSV,"NO_ALARM")
    field(HSV,"NO_ALARM")
    field(LSV,"NO_ALARM")
    field(HYST,"0.0e+00")
    field(ADEL,"0.0e+00")
    field(MDEL,"0.0e+00")
    field(SIOL,"")
    field(SIML,"")
    field(SIMS,"NO_ALARM")
    field(IVOA,"Continue
  normally")
    field(IVOV,"0.0e+00")
}
```

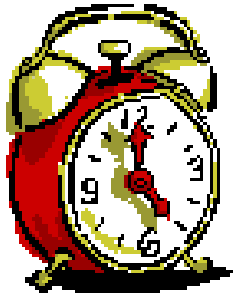
# EPICS Databases – What are they?

- A collection of one or more EPICS *records* of various types.
- Records can be interconnected and are used as building blocks to create applications.
- A data file that's loaded into IOC memory at boot time.
- Channel access talks to the IOC memory copy of the database.

# Our First Database



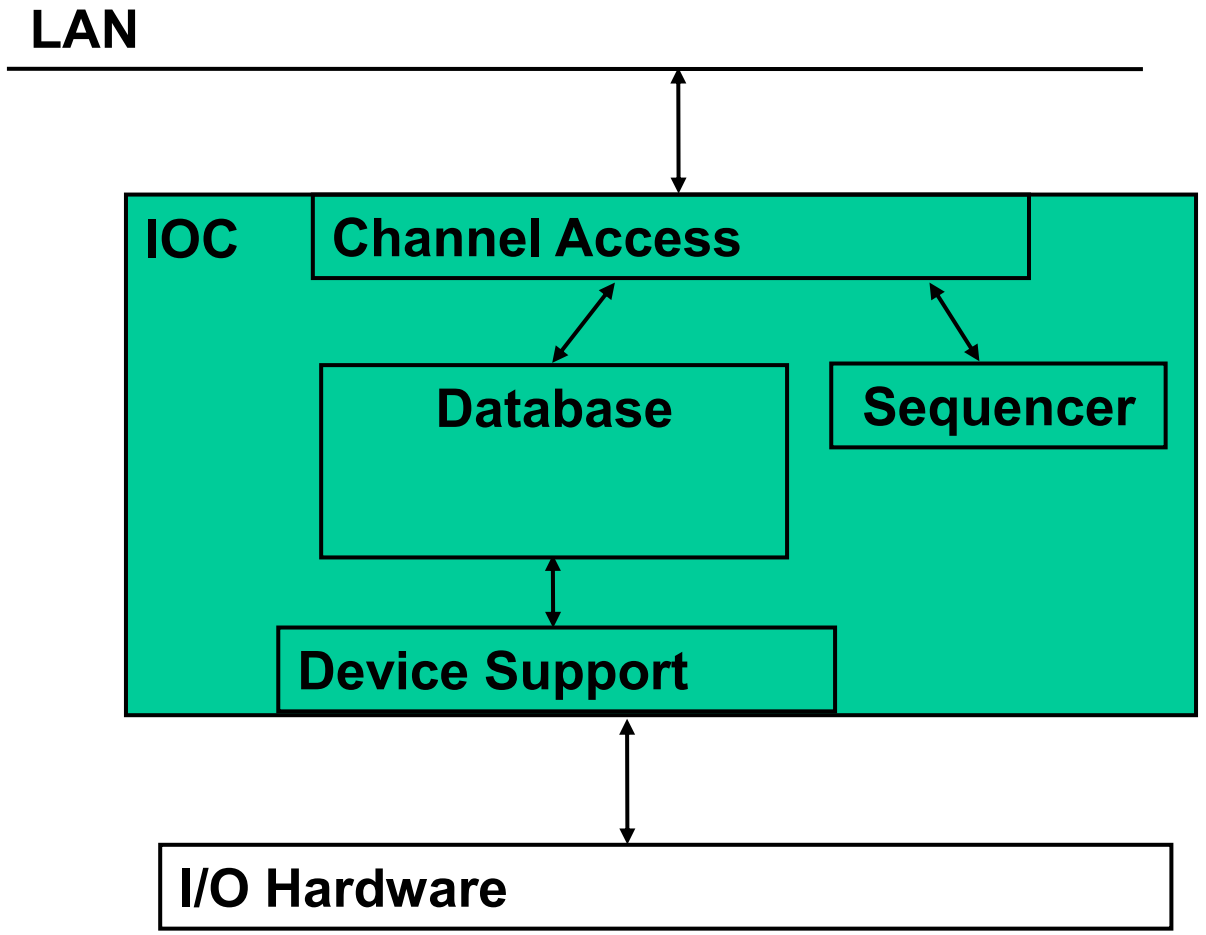
# Record Processing



- Record processing can be periodic, or event driven.
- For *periodic* record processing, standard scan rates are:
  - 10, 5, 2, 1, 0.5, 0.2 and 0.1 seconds
  - Custom scan rates can be configured up to speeds allowed by operating system and hardware.
- For *event driven* record processing, events include:
  - Hardware interrupts
  - Request from another record via links
  - EPICS Events
  - Channel Access Puts

# Inside an IOC

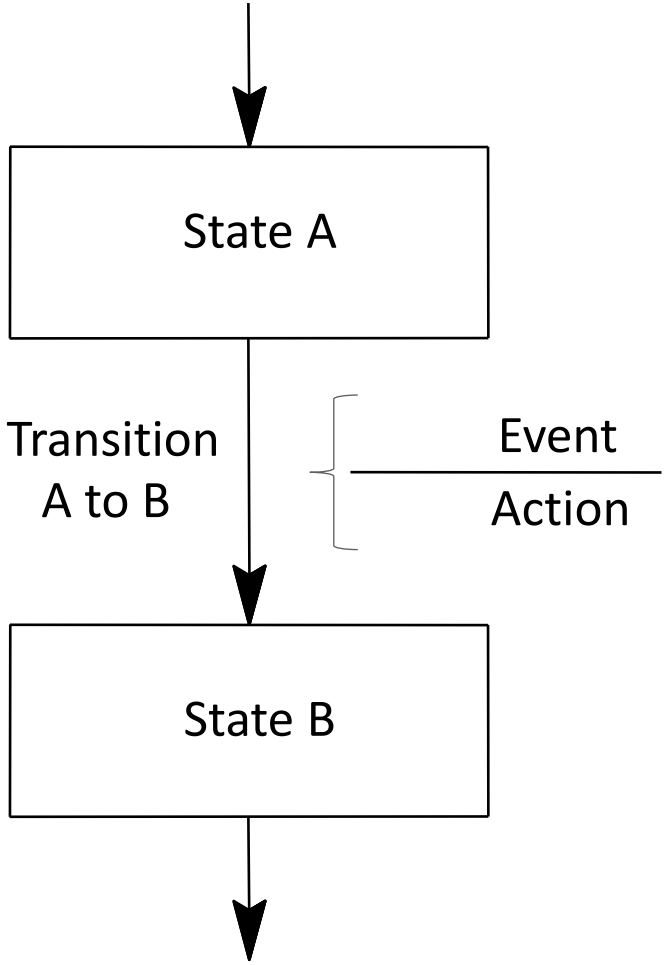
- The major software components of an IOC (IOC Core).



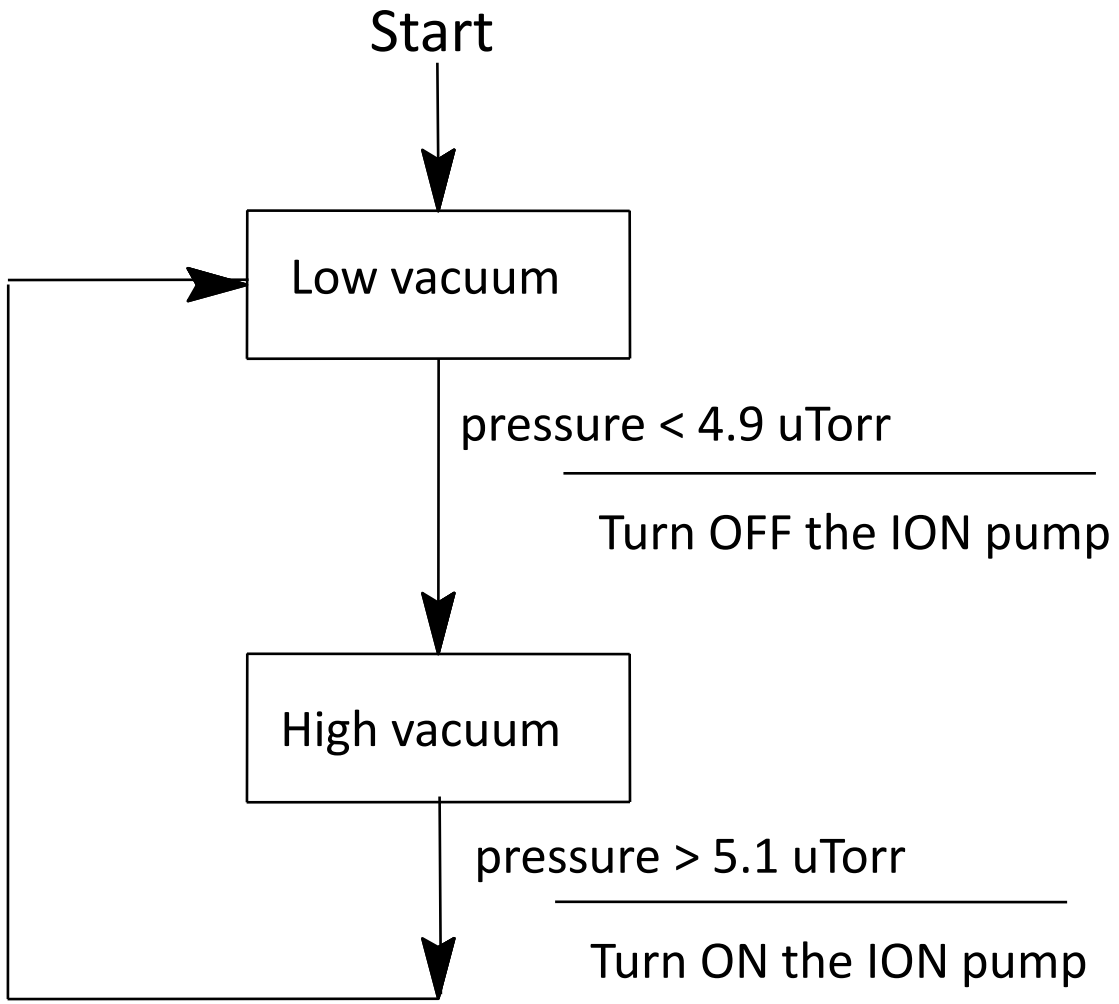
# The Sequencer

- Runs programs written in State Notation Language (SNL).
- SNL is a 'C' like language to facilitate programming of sequential operations.
- Fast execution - compiled code.
- Programming interface to extend EPICS in the real-time environment.
- Common uses:
  - Provide automated start-up sequences like vacuum or RF where subsystems need coordination.
  - Provide fault recovery or transition to a safe state.
  - Provide automatic calibration of equipment.

# SNL implements State Transition Diagrams

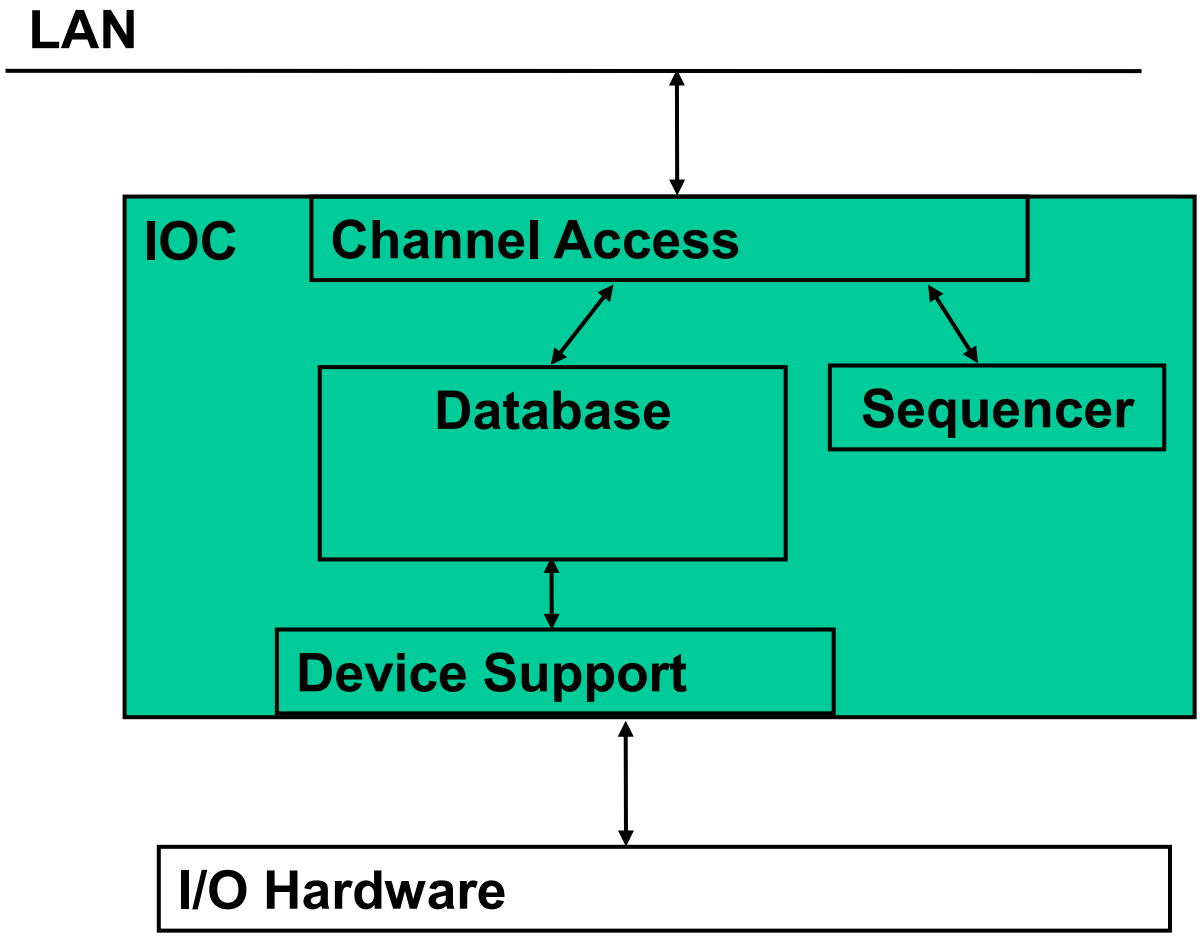


# STD Example


Start
Low vacuum
pressure < 4.9 uTorr
Turn OFF the ION pump
High vacuum
pressure > 5.1 uTorr
Turn ON the ION pump

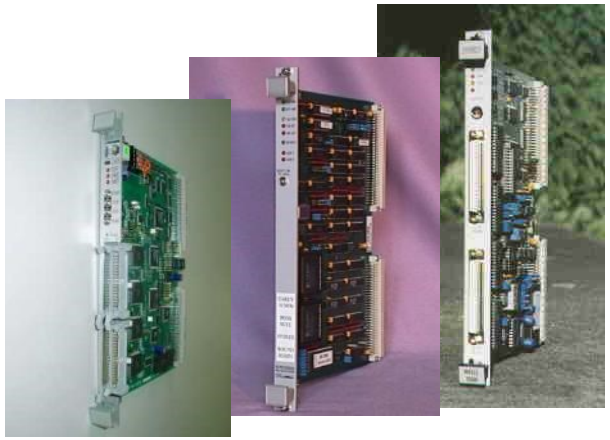
# Inside an IOC

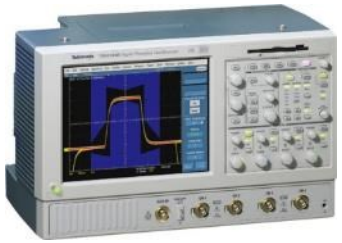
- The major software components of an IOC (IOC Core)



# Device Support

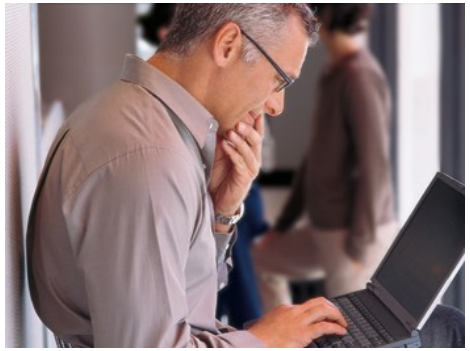
- Device and driver support interface hardware to the database
- Examples of devices:
  - VME cards: ADC, DAC, Binary I/O etc.
  - Motor controllers
  - Oscilloscopes
  - PLCs





# Device Support

- Usually has to be written for ‘new’ hardware.
- Good news – someone, somewhere has usually written support for your device, or a very similar one.
- See the EPICS web site for available support.
- Or ask the EPICS community.



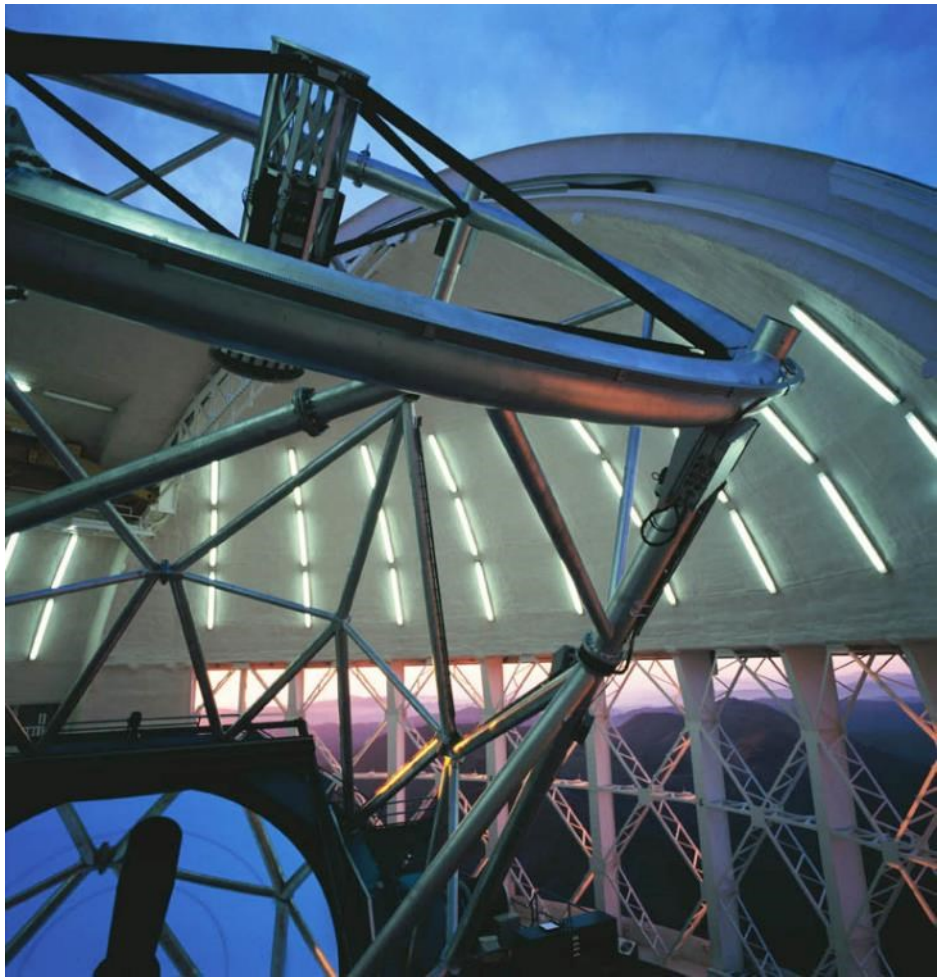
# When to use databases

- Hardware connection
- Real time performance – no network latencies
- Whenever a database is good enough

| Advantages | Disadvantages |
|---|---|
| Simplify hardware connection | If you have device support |
| Configuring not programming. | You need to understand database use |
| Database is easily understood by other EPICS developers | |
| Speed - All processing (often) in same machine | |

# When to use the sequencer

- For sequencing complex events
  - e.g. parking and unparking a telescope mirror



Photograph courtesy of the Gemini Telescopes project

# When to use clients

- To interact with the control system
- Many already exist – CSS, ALH, Strip Tool, archiver, EDM etc.
- For data analysis or visualization
- Supervisory control
  - e.g. to manage an accelerator

# Command Line Clients

| Functionality | Channel Access | PV Access |
|---|---|---|
| Read PV | `caget <PV_name>` | `pvget <PV_name>` |

# How fast is EPICS?

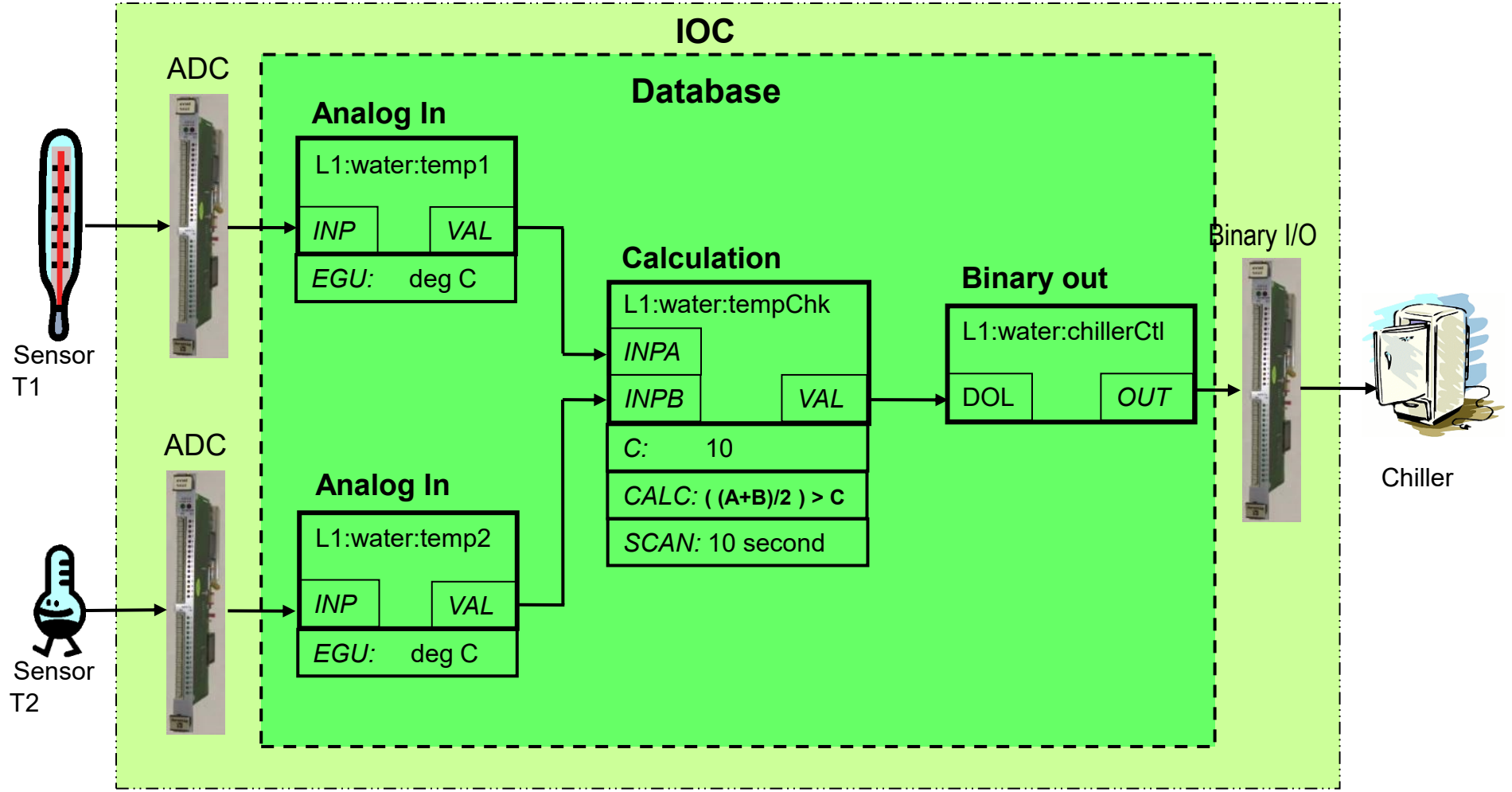
- Can be fast or slow, it depends how you use it!
- Use the correct tool for the job; Database, sequencer, custom code (ioc) or custom code (client).
- Ultimately speed depends upon hardware.
- Some (a bit old) benchmarks*:

| Machine | OS | CPU | Speed | Rec/sec | %CPU |
|---|---|---|---|---|---|
| MVME167 | vxWorks | 68040 | 33MHz | 6000 | 50 |
| MVME 2306 | vxWorks | PPC604 | 300MHz | 10000 | 10 |
| MVME5100 | vxWorks | PPC750 | 450MHz | 40000** | 10** |
| PC | Linux | PII | 233MHz | 10000 | 27 |
| PC | Linux | P4 | 2.4GHz | 50000 | 9 |

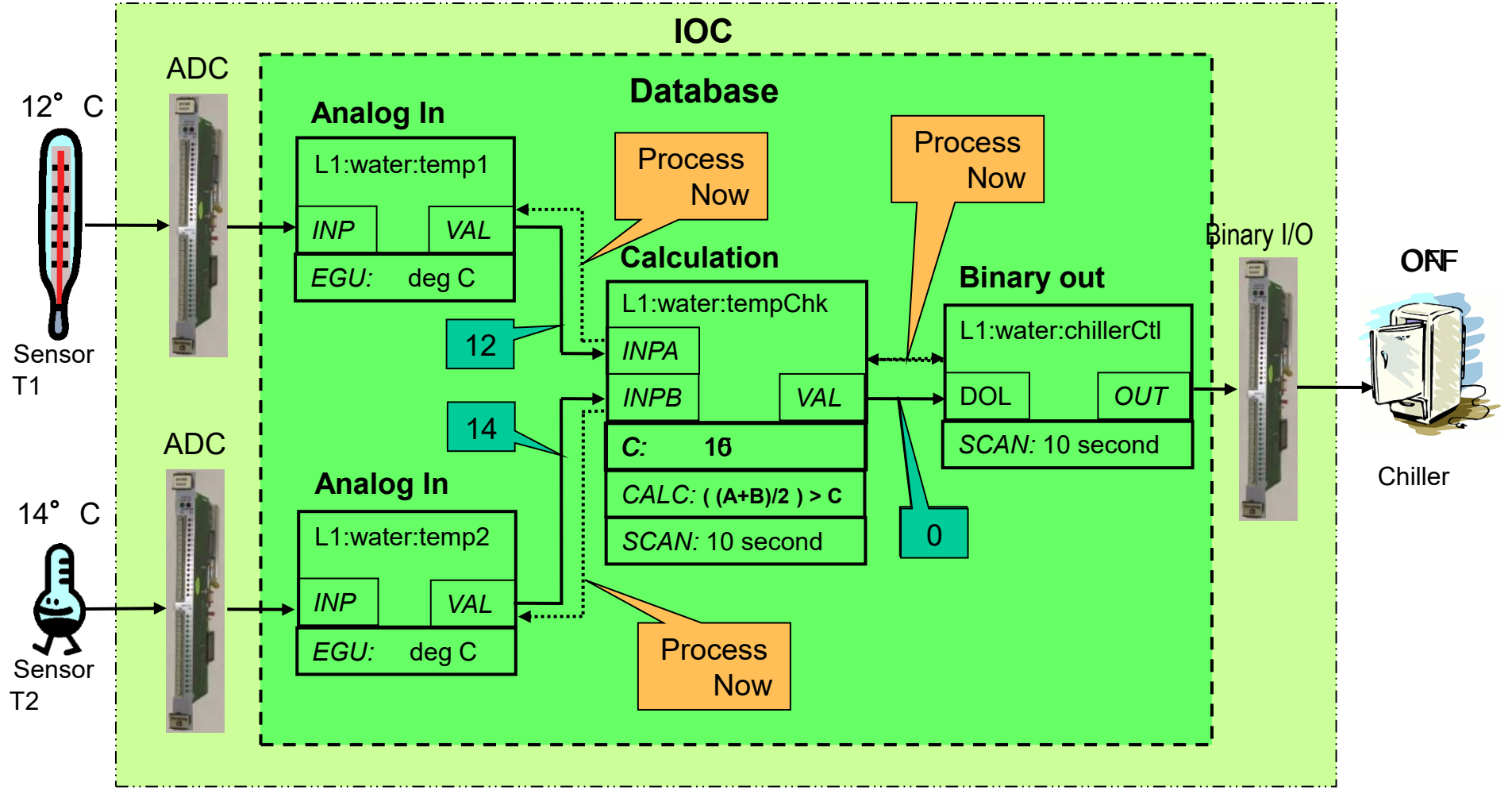
*Benchmark figures courtesy of Steve Hunt (PSI)
**Extrapolated from performance figures provided by L.Hoff, BNL

- Database design and periodic scanning affect *apparent* system speed

# Database Processing



# Apparent Performance



# The EPICS Web Site

- The central site for EPICS information
- Documentation
  - Application Developer's Guide
  - Component Reference Manual
- CA Clients
- Device support
- Tech-talk
- https://epics-controls.org

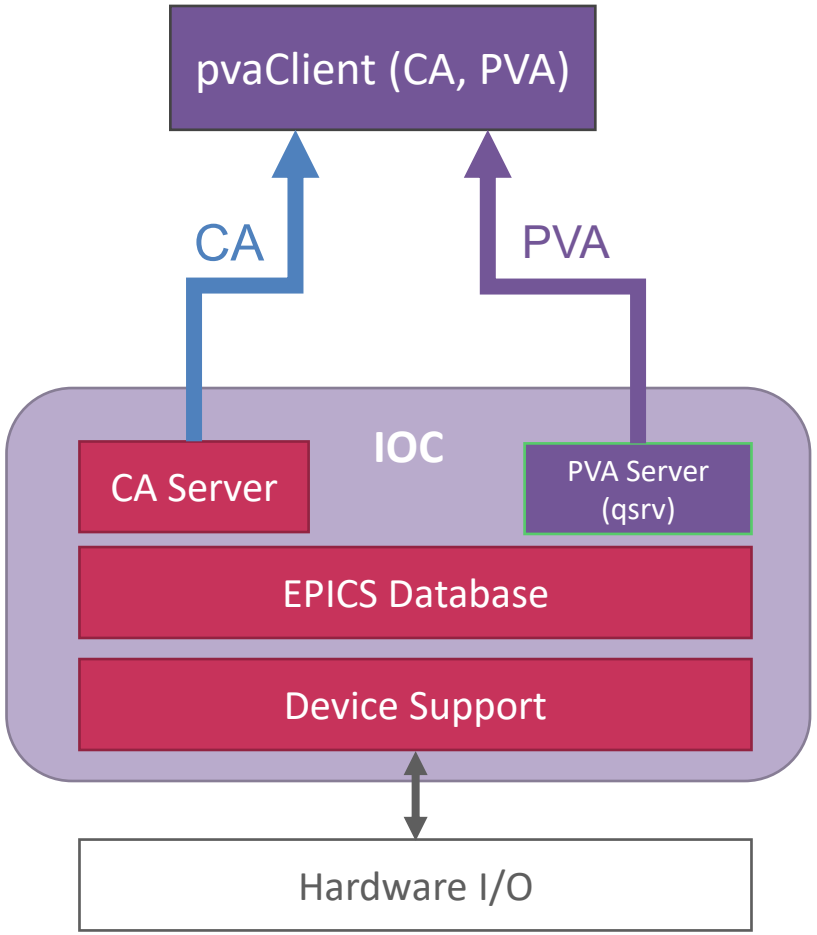
COSYLAB

# Improvements in EPICS 7

# What is EPICS 7? Why "7"?

- EPICS 7 is a merge of EPICS V3 and V4 ( 3 + 4 = 7 )
  - First released in December 2017
- Adds structured data on top of EPICS V3
  - From EPICS V3
    - core: build system, libCom, tools
    - database: db access and processing, record types, device support, filters
    - ca: channel access client and server
  - From EPICS V4
    - pvData: structured data, helper libraries
    - pvAccess: network protocol for pvData, client and server
    - normativeTypes: standard data containers
    - pvaClient: simplified user-level libraries
    - examples: example code
- EPICS 7 does not require upgrading any of existing applications
  - But allows gradual enhancement
- EPICS 7 is the active development branch
  - EPICS 3.15 is a "long-term support" classic version
  - All fixes and new features go to EPICS 7 first and are selectively back-ported to 3.15 (when appropriate).

Purple: new in EPICS 7
Red: classic EPICS



# Why EPICS 7

- EPICS 3 was designed and intended for efficient **control**
  - Simple data, mostly scalar values
  - Efficient passing of small items
  - Control loop at low level more important than data transport
  - No or limited buffering of data values
    - For control loops, historical values do not matter.
- Several developments have happened:
  - EPICS has expanded into data acquisition
    - Loss of data cannot be tolerated
  - Higher level of abstraction is often required
    - Individual scalar values are not sufficient
- A lot of work goes into working around the deficiencies to cover the new use cases
  - Multiplication of effort. Loss of attractivity.

- Needs of data processing and measurements
  - Data depends on the surrounding conditions
    - Stream of bytes does not make a picture without knowing what the bytes mean.
      - How many bytes make a pixel?
      - How many pixels make a row?
      - How many rows does the picture have?
      - What is the mapping of a pixel to colors?
    - Beam positions change really fast.
      - If X is from a different pulse than Y, we do not really know where the beam was.
      - And if signal intensity was low, we know even less…
  - Loss of data is not acceptable
    - If you get a waveform with half old and half new data, it is not much worth
    - If points of a (fast) scan are lost, the scan has failed

# Short highlights of EPICS 7

- Structured data access over network (pvAccess)
  - Consistent handling of correlated data items and metadata (e.g. images, spectra, correlated values)
    - Data quality, synchronization, atomic actions
  - Better `ARRAY` and `STRING` handling
- Efficient data transfer over network
  - Allows for high performance archiving and image transfers
  - Send only items that have changed (subscriptions, aka monitors)
- Communication patterns: read, write, subscribe, RPC (read with parameters)
  - Service oriented architecture: archiver, snapshots, database backends
- Complex computing
  - Communicating with devices (groups of PVs on an IOC) in an always-consistent, transaction type way
- Extends the scope of EPICS usefulness
  - Data acquisition and processing in addition to device control

COSYLAB

# Other improvements in EPICS 7

## Overcome shortcomings of Channel Access

- Support for 64bit integers
  - 1/8/16/32/64bit integers, signed and unsigned
- Better support for arrays
  - No element_count upper limit (fixed and bounded arrays possible)
  - Clear distinction between arrays of size 1 and scalars
  - Axis metadata is transmitted with the array
- Better support for strings
  - Arbitrary size
  - No fixed limit or need for long string workaround
- Much better support for arrays of strings
  - Handles arbitrary number of arbitrary length strings

# EPICS 7 data handling
## Structured Data

- EPICS 7 works with pvData structures
- Structures include “payload” data and metadata
  - Timestamps
  - Alarm status
  - Payload = “Value” can be of different types and structures
    - Scalar
    - Array
    - Structure of scalars
    - Structure of arrays
    - Etc.
- Structures could be almost “anything”
  - To allow applications to interoperate, a set of Normative Types has been defined.
    - Display an image PV in a GUI screen, with correct number of pixels, bits per pixel, etc.
    - Plot sampled data with the correct time axis.
    - Perform an FFT, all required data being in the same structure (e.g., sampling frequency).

```
NTScalar
    double value
    alarm_t alarm
        int severity
        int status
        string message
    time_t timeStamp
        long secondsPastEpoch
        int nanoseconds
        int userTag
```

# Improvements in EPICS 7

## Network transport and RPC type services

- pvAccess only sends deltas on the wire
  - In transport only changed values are set and  a bitset that indicates which fields have changed value

- RPC type services can generate different structures for every call and for different operations (put – request, get – response)
  - pvD

```
NTScalar                                    NTScalar
    double value                 8.0            double value                 8.1
    alarm_t alarm                               alarm_t alarm
        int severity               2                int severity               2
        int status                 3                int status                 3
        string message      HIHI_ALARM              string message      HIHI_ALARM
    time_t timeStamp                            time_t timeStamp
        long secondsPastEpoch 1460589144            long secondsPastEpoch 1460589145
        int nanoseconds       553824112             int nanoseconds       588698520
        int userTag                0                int userTag                0
```

*Based on a presentation by :*
*E. Norum (APS)*

COSYLAB

# Getting Started with EPICS

## IOC Overview

COSYLAB

# IOC Overview

- What is an EPICS Input/Output Controller (IOC)
- How to create a new IOC application.
- How to build an IOC application.
- How to run an IOC application on various platforms.
- Console interaction with an IOC application (`iocsh`).

# What is an Input/Output Controller?

- Some definitions from the first lectures:
  - A computer running *iocCore*, a set of EPICS routines used to define process variables and implement real-time control algorithms.
  - *iocCore* uses database records to define process variables and their behavior.

# What does an Input/Output Controller do?

- As its name implies, an IOC often performs input/output operations to attached hardware devices.
- An IOC associates the values of EPICS process variables with the results of these input/output operations.
- An IOC can perform sequencing operations, closed-loop control and other computations.

# ‘Host-based’ and ‘Target’ IOCs

- ‘Host-based’ IOC
  - Runs in the same environment as which it was compiled
  - ‘Native’ software development tools (compilers, linkers)
  - Sometimes called a ‘Soft’ IOC
  - IOC is a program like any other on the machine
  - Possible to have many IOCs on a single machine
- ‘Target’ IOC
  - Runs in a different environment than where compiled
  - ‘Cross’ software development tools
  - Linux, VxWorks, RTEMS
  - IOC boots from some medium (usually network)
  - IOC is the only program running on the machine

# IOC Software Development Tools

- EPICS uses the GNU version of make
  - Almost every directory from the `{TOP}` on down contains a `Makefile`
  - Make recursively descends through the directory tree
    - Determines what needs to be [re]built
    - Invokes compilers and other tools as instructed in Makefile
  - GNU C/C++ compilers or vendor compilers can be used
- No fancy ‘integrated development environment’ yet…

# IOC Application Development Examples

- The following slides provide step-by-step examples of how to:
  - Create, build, run the example IOC application on a 'host' machine (Linux)

- Each example begins with the use of `makeBaseApp.pl`

# The **`makeBaseApp.pl`** Script

- Part of EPICS base distribution.
- Populates a new, or adds files to an existing, `{TOP}` area.
- Requires that your environment contain a valid `EPICS_HOST_ARCH`
  - EPICS base contains scripts which can set this as part of your login sequence
    - `linux-x86(_64)`, `darwin-ppc`, `solaris-sparc`, `win32-x86`
- Creates different directory structures based on a selection of different templates.
- Commonly-used templates include
  - ioc - Generic IOC application skeleton
  - example - Example IOC application

# Creating and initializing a new {TOP}

- Create a new directory and run `makeBaseApp.pl` from within that directory:

```
1. mkdir <dirName>
2. cd <dirName>
3. makeBaseApp.pl -t example ex1
```

  - The template is specified with the `-t` argument
  - The application name (`ex1App`) is specified with the argument `ex1`

    (created directory gets "`App`" appended to this name)

# {TOP} directory structure

- The makeBaseApp.pl creates the following directory structure in {TOP} (<dirName>):
  - configure/ - Configuration files
  - ex1App/ - Files associated with the 'ex1App' application
    - Db/ - Databases, templates, substitutions
    - src/ - Source code

- Every directory also contains a Makefile

# {TOP}/configure files

- Some may be modified as needed
  - CONFIG
    - Specify make variables (e.g. to build for a particular target):
    - CROSS_COMPILER_TARGET_ARCHS = vxWorks-68040
  - RELEASE
    - Specify location of other {TOP} areas used by applications in this {TOP} area.
- Others are part of the (complex!) build system and should be left alone.

# Create a host-based IOC boot directory

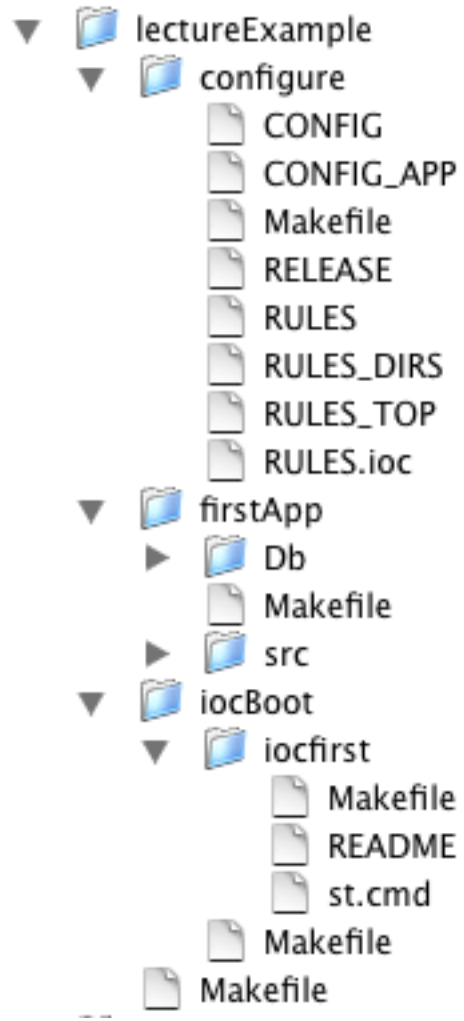
- Run `makeBaseApp.pl` from the `{TOP}` directory
- `-t example`: to specify template
- `-i`: to show that IOC boot directory is to be created
- name of IOC

```
4. makeBaseApp.pl -i -t example ex1
```

# {TOP} directory structure

- The command from the previous slide creates an additional directory in {TOP}:
  - iocBoot/ - Directory containing per-IOC boot directories
    - iocex1/ - Boot directory for ‘iocex1’ IOC

# {TOP} directory structure



COSYLAB

# Build the application

- Run the GNU make program
  - `make` on Darwin, Linux, Windows
  - `gnumake` on Solaris

- Or to clean things up completely
  - `make clean uninstall`

# {TOP} directory structure after running make

- These additional directories are now present in {TOP}
  - bin/ - Directory containing per-architecture directories
    - linux-x86_64/ - Object files and executables for this architecture
  - lib/ - Directory containing per-architecture directories
    - linux-x86_64/ - Object libraries for this architecture
  - dbd/ - Database definition files
  - db/ - Database files (record instances, templates)
- There may be other directories under bin/ and lib/.

# IOC Startup

- IOCs read commands from a startup script
  - Typically `st.cmd` in the `{TOP}/iocBoot/<iocname>/` directory
- IOCs read these scripts with the `iocsh` shell
- Command syntax can be similar but `iocsh` allows a more familiar form too.
- Script was created with the `makeBaseApp.pl -i` command.
- For a ‘real’ IOC you’d likely add commands to configure hardware modules, start sequence programs, update log files, etc.

# Example application startup script

```
1  #!../../bin/linux-x86_64/ex1
2
3  ## You may have to change ex1 to something else
4  ## everywhere it appears in this file
5
6  < envPaths
7
8  cd ${TOP}
9
10 ## Register all support components
11 dbLoadDatabase("dbd/ex1.dbd")
12 ex1_registerRecordDeviceDriver(pdbbase)
13
14 ## Load record instances
15 dbLoadRecords("db/dbExample1.db","user=epics")
16 dbLoadRecords("db/dbExample2.db","user=epics,no=1,scan=1 second")
17 dbLoadRecords("db/dbExample2.db","user=epics,no=2,scan=2 second")
18 dbLoadRecords("db/dbExample2.db","user=epics,no=3,scan=5 second")
19 dbLoadRecords("db/dbSubExample.db","user=epics")
20
21 cd ${TOP}/iocBoot/${IOC}
22 iocInit()
23
24 ## Start any sequence programs
25 #seq sncExample,"user=epics”
```

# Example application startup script

```
1  #!../../bin/linux-x86_64/ex1
```

- This allows a host-based IOC application to be started by simply executing the `st.cmd` script.
- If you are running this on a different architecture, the `linux-x86_64` will be that of the architecture you are using.
- `ex1` refers to the IOC name that you created with the '`makeBaseApp.pl -i`' command. For our example, it is `ex1`.
- Remaining lines beginning with a `#` character are comments.

# Example application startup script

```
1  #!../../bin/linux-x86_64/ex1
2
3  ## You may have to change ex1 to something else
4  ## everywhere it appears in this file
5
6  < envPaths
7
8  cd ${TOP}
9
10 ## Register all support components
11 dbLoadDatabase("dbd/ex1.dbd")
12 ex1_registerRecordDeviceDriver(pdbbase)
13
14 ## Load record instances
15 dbLoadRecords("db/dbExample1.db","user=epics")
16 dbLoadRecords("db/dbExample2.db","user=epics,no=1,scan=1 second")
17 dbLoadRecords("db/dbExample2.db","user=epics,no=2,scan=2 second")
18 dbLoadRecords("db/dbExample2.db","user=epics,no=3,scan=5 second")
19 dbLoadRecords("db/dbSubExample.db","user=epics")
20
21 cd ${TOP}/iocBoot/${IOC}
22 iocInit()
23
24 ## Start any sequence programs
25 #seq sncExample,"user=epics”
```

# Example application startup script

```
6  < envPaths
```

- The application reads commands from the `envPaths` file created by `makeBaseApp -i` and `make`
- The envPaths file contains commands to set up environment variables for the application:
  - Architecture
  - IOC name
  - `{TOP}` directory
  - `{TOP}` directory of each component named in `configure/RELEASE`
- These values can then be used by subsequent commands:
  - `epicsEnvSet(IOC,"iocex1")`
  - `epicsEnvSet(TOP,"/home/$USER/ex1")`
  - `epicsEnvSet(EPICS_BASE,"/opt/epics/base")`

# Example application startup script

```
1  #!../../bin/linux-x86_64/ex1
2
3  ## You may have to change ex1 to something else
4  ## everywhere it appears in this file
5
6  < envPaths
7
8  cd ${TOP}
9
10 ## Register all support components
11 dbLoadDatabase("dbd/ex1.dbd")
12 ex1_registerRecordDeviceDriver(pdbbase)
13
14 ## Load record instances
15 dbLoadRecords("db/dbExample1.db","user=epics")
16 dbLoadRecords("db/dbExample2.db","user=epics,no=1,scan=1 second")
17 dbLoadRecords("db/dbExample2.db","user=epics,no=2,scan=2 second")
18 dbLoadRecords("db/dbExample2.db","user=epics,no=3,scan=5 second")
19 dbLoadRecords("db/dbSubExample.db","user=epics")
20
21 cd ${TOP}/iocBoot/${IOC}
22 iocInit()
23
24 ## Start any sequence programs
25 #seq sncExample,"user=epics”
```

# Example application startup script

```
8   cd ${TOP}
```

- The working directory is set to the value of the `{TOP}` environment variable (as set by the commands in `envPaths`).
- Allows use of relative path names in subsequent commands.

```
11  dbLoadDatabase("dbd/ex1.dbd")
```

- Loads the database definition file for this application.
- Describes record layout, menus, drivers.

```
12  ex1_registerRecordDeviceDriver(pdbbase)
```

- Registers the information read from the database definition files.

# Example application startup script

```
1  #!../../bin/linux-x86_64/ex1
2
3  ## You may have to change ex1 to something else
4  ## everywhere it appears in this file
5
6  < envPaths
7
8  cd ${TOP}
9
10 ## Register all support components
11 dbLoadDatabase("dbd/ex1.dbd")
12 ex1_registerRecordDeviceDriver(pdbbase)
13
14 ## Load record instances
15 dbLoadRecords("db/dbExample1.db","user=epics")
16 dbLoadRecords("db/dbExample2.db","user=epics,no=1,scan=1 second")
17 dbLoadRecords("db/dbExample2.db","user=epics,no=2,scan=2 second")
18 dbLoadRecords("db/dbExample2.db","user=epics,no=3,scan=5 second")
19 dbLoadRecords("db/dbSubExample.db","user=epics")
20
21 cd ${TOP}/iocBoot/${IOC}
22 iocInit()
23
24 ## Start any sequence programs
25 #seq sncExample,"user=epics”
```

# Example application startup script

```
15 dbLoadRecords("db/dbExample1.db","user=epics")
16 dbLoadRecords("db/dbExample2.db","user=epics, no=1, scan=1 second")
17 dbLoadRecords("db/dbExample2.db","user=epics, no=2, scan=2 second")
18 dbLoadRecords("db/dbExample2.db","user=epics, no=3, scan=5 second")
19 dbLoadRecords("db/dbSubExample.db","user=epics")
```

- Read the application database files
  - These define the records which this IOC will maintain.
  - A given file can be read more than once (with different macro definitions).

# Example application startup script

```
1  #!../../bin/linux-x86_64/ex1
2
3  ## You may have to change ex1 to something else
4  ## everywhere it appears in this file
5
6  < envPaths
7
8  cd ${TOP}
9
10 ## Register all support components
11 dbLoadDatabase("dbd/ex1.dbd")
12 ex1_registerRecordDeviceDriver(pdbbase)
13
14 ## Load record instances
15 dbLoadRecords("db/dbExample1.db","user=epics")
16 dbLoadRecords("db/dbExample2.db","user=epics,no=1,scan=1 second")
17 dbLoadRecords("db/dbExample2.db","user=epics,no=2,scan=2 second")
18 dbLoadRecords("db/dbExample2.db","user=epics,no=3,scan=5 second")
19 dbLoadRecords("db/dbSubExample.db","user=epics")
20
21 cd ${TOP}/iocBoot/${IOC}
22 iocInit()
23
24 ## Start any sequence programs
25 #seq sncExample,"user=epics”
```

# Example application startup script

```
21  cd ${TOP}/iocBoot/${IOC}
```

- The working directory is set to the per-IOC startup directory

```
22  iocInit()
```

- Activates everything
- After reading the last line of the `st.cmd` script the IOC continues reading commands from the console
  - Diagnostic commands
  - Configuration changes

# Running a host-based IOC

- Change to IOC startup directory (the one containing the st.cmd script)

```
cd iocBoot/iocex1
```

- Run the IOC executable with the startup script as the only argument

```
../../bin/linux-x86/ex1 st.cmd
```

- The startup script commands will be displayed as they are read and executed
- When all the startup script commands are finished the `iocsh` will display an `epics>` prompt and wait for commands to be typed.

```
iocInit()
############################################################################
###  EPICS IOC CORE built on Jun 23 2004
###  EPICS R3.14.6 $R3-14-6$ $2004/05/28 19:27:47$
############################################################################
Starting iocInit
## Start any sequence programs
#seq sncExample,"user=epics"
iocInit: All initialization complete
epics>
```

# Some Useful `iocsh` Commands

- Display list of records maintained by this IOC:

```
epics> dbl
epics:aiExample
epics:aiExample1
epics:aiExample2
epics:aiExample3
epics:calcExample
epics:calcExample1
epics:calcExample2
epics:calcExample3
epics:compressExample
epics:subExample
epics:xxxExample
```

- Caution – some IOCs have many records.

# Some Useful `iocsh` Commands

- Display a record:

```
epics> dbpr epics:aiExample
ASG:                DESC: Analog input  DISA: 0             DISP: 0
DISV: 1             NAME: epics:aiExample               RVAL: 0
SEVR: MAJOR         STAT: HIHI          SVAL: 0             TPRO: 0
VAL: 9
epics> dbpr epics:aiExample
ASG:                DESC: Analog input  DISA: 0             DISP: 0
DISV: 1             NAME: epics:aiExample               RVAL: 0
SEVR: MINOR         STAT: LOW           SVAL: 0             TPRO: 0
VAL: 4
```

- **`dbpr <recordname>`** 1 prints more fields
- **`dbpr <recordname>`** 2 prints even more fields, and so on

# Some Useful `iocsh` Commands

- Show list of attached clients:

```
epics> casr
Channel Access Server V4.11
No clients connected.
```

- **`casr 1`** prints more information
- **`casr 2`** prints even more information

# Some Useful `iocsh` Commands

- Do a ‘put’ to a field:

```
epics> dbpf epics:calcExample.SCAN "2 second"
DBR_STRING:           2 second
```

- Arguments with spaces must be enclosed in quotes

COSYLAB

# Some Useful `iocsh` Commands

- The `help` command, with no arguments, displays a list of all `iocsh` commands
  - 90 or so, plus commands for additional drivers
- With arguments it displays usage information for each command listed:

```
epics> help dbl dbpr dbpf
dbl 'record type' fields
dbpr 'record name' 'interest level'
dbpf 'record name' value
```

# Terminating a host-based IOC

- Type **`exit`** at the `iocsh` prompt.
- Type your ‘interrupt’ character (usually `Ctrl-C`).
- Kill the process from another terminal/window.

# Command-Line Tools

- These are client-side tools.
- The tools we will cover are:
  - `caget` – gets the value of one or more process variables
  - `caput` – sets the value of one process variables
  - `camonitor` – monitors the value changes of one or more process variables
  - `cainfo` – gets information about one or more process variables
- All accept `-h` to display usage and options.

# **caget** Example

- Get the values of two process variables:

```
caget S35DCCT:currentCC S:SRlifeTimeHrsCC
```

- Returns:

```
S35DCCT:currentCC    102.037
S:SRlifeTimeHrsCC    7.46514
```

# **caput** Example

- Set the value of a process variable:

```
caput Xorbit:S1A:H1:CurrentAO 1.2
```

- Returns:

```
Old : Xorbit:S1A:H1:CurrentAO    0
New : Xorbit:S1A:H1:CurrentAO    1.2
```

# **camonitor** Example

- Monitor two process variables:

```
camonitor evans:calc evans:bo01
```

- Returns:

```
evans:calc    1970-08-05 17:23:04.623245 1
evans:bo01    1970-08-05 17:23:04.623245 On
evans:calc    1970-08-05 17:23:05.123245 2
evans:bo01    1970-08-05 17:23:05.123245 Off
evans:calc    1970-08-05 17:23:05.623245 3
evans:calc    1970-08-05 17:23:06.123245 4
evans:calc    1970-08-05 17:23:06.623233 5
evans:calc    1970-08-05 17:23:07.123183 6
```

- Use `Ctrl-C` to stop monitoring.

# **cainfo** Example

- Get information about a process variable:

```
cainfo S35DCCT:currentCC
```

- Returns:

```
State:     connected
Host:      ctlapps41188:5064
Access:    read, no write
Data type: DBR_DOUBLE (native: DBF_DOUBLE)
Element count: 1
```

COSYLAB

# Review

- IOC applications can be host-based or target-based.
- The `makeBaseApp.pl` script is used to create IOC application modules and IOC startup directories.
- `{TOP}/configure/RELEASE` contents specify location of other `{TOP}` areas used by this `{TOP}` area.
- `{TOP}/iocBoot/<iocname>/st.cmd` is the startup script for IOC applications.
- The EPICS build system requires the use of GNU `make`.
- The EPICS Application Developer's Guide contains a wealth of information.

Based on a presentation by :
A. Johnson (APS)

COSYLAB

# Getting started with EPICS

## Database Concepts

# Contents

- Records
- Fields and field types
- Record Scanning
- Input and Output record types
- Links, link address types
- Connecting records together
- Protection mechanisms
- Alarms, deadbands, simulation and security

# Database = Records + Fields + Links

- A control system using EPICS will contain one or more IOCs.
- Each IOC loads one or more Databases telling it what to do.
- A Database is a collection of Records of various types.
- A Record is an object with:
  - A unique name
  - A behavior defined by its record type (class)
  - Controllable properties (fields)
  - Optional associated hardware I/O (device support)
  - Links to other records

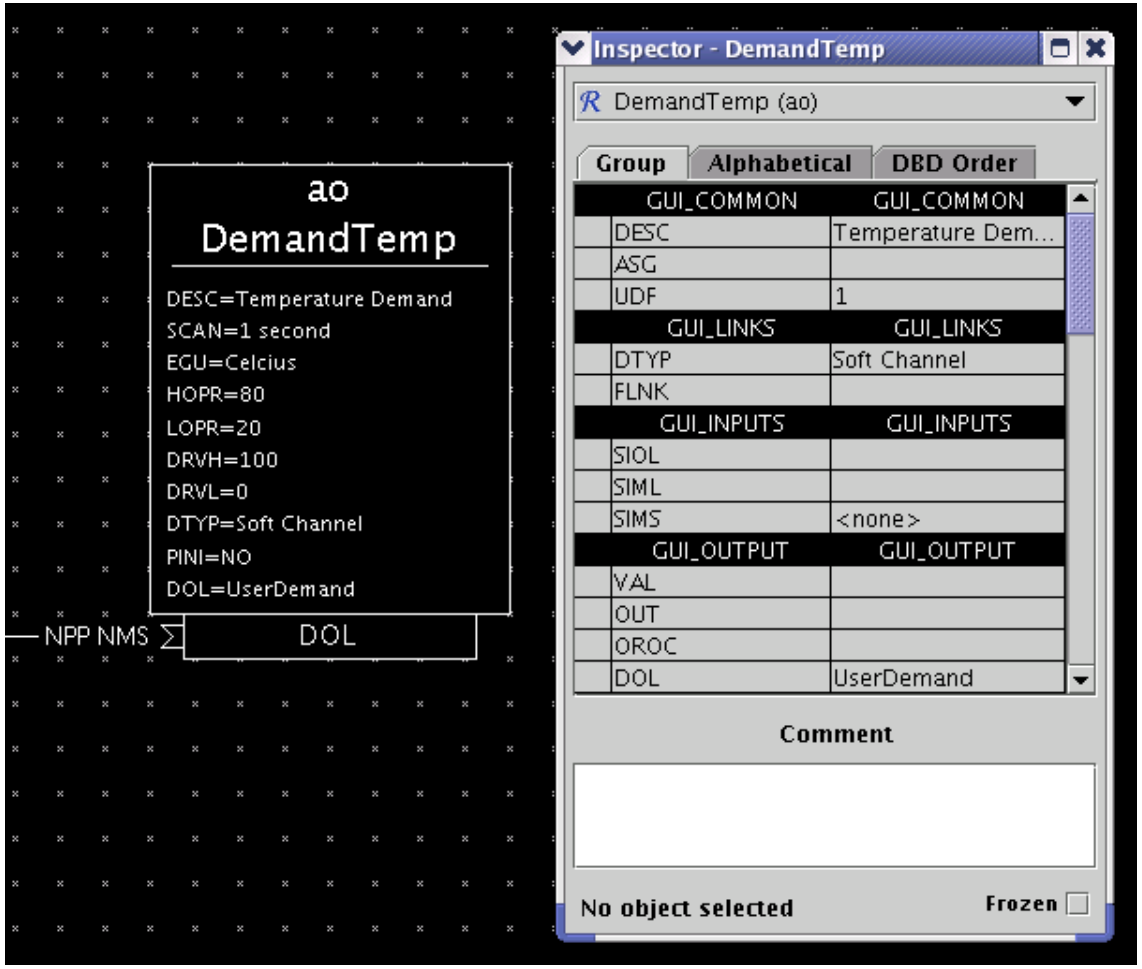
# Record Activity

- Records are active — they can do things:
  - Get data from other records or from hardware
  - Perform calculations
  - Check values are in range & raise alarms
  - Put data to other records or to hardware
  - Activate or disable other records
  - Wait for hardware signals (interrupts)
- What a record does depends upon its record type and the settings of its fields.
- No action occurs unless a record is processed.

# How is a record implemented?

- A ‘C’ structure with both data storage and pointers to record type information.
- A record definition within a database provides:
  - Record name
  - The record’s type
  - Values for each design field
- A record type provides:
  - Definitions of all the fields
  - Code which implements the record behavior
- New record types can be added to an application as needed.

# A Graphical View of a Record



# The IOC's View

- The full `.db` file entry for an Analogue Output Record

```
record(ao,"DemandTemp") {
  field(DESC,"Temperature")
  field(ASG,"")
  field(SCAN,"Passive")
  field(PINI,"NO")
  field(PHAS,"0")
  field(EVNT,"0")
  field(DTYP,"VMIC 4100")
  field(DISV,"1")
  field(SDIS,"")
  field(DISS,"NO_ALARM")
  field(PRIO,"LOW")
  field(FLNK,"")
  field(OUT,"#C0 S0")
  field(OROC,"0.0e+00")
  field(DOL,"")
  field(OMSL,"supervisory")
  field(OIF,"Full")
  field(PREC,"1")
  field(LINR,"NO CONVERSION")
  field(EGUF,"100")
  field(EGUL,"0")
  field(EGU,"Celsius")
  field(DRVH,"100")
  field(DRVL,"0")
  field(HOPR,"80")
  field(LOPR,"10")
  field(HIHI,"0.0e+00")
  field(LOLO,"0.0e+00")
  field(HIGH,"0.0e+00")
  field(LOW,"0.0e+00")
  field(HHSV,"NO_ALARM")
  field(LLSV,"NO_ALARM")
  field(HSV,"NO_ALARM")
  field(LSV,"NO_ALARM")
  field(HYST,"0.0e+00")
  field(ADEL,"0.0e+00")
  field(MDEL,"0.0e+00")
  field(SIOL,"")
  field(SIML,"")
  field(SIMS,"NO_ALARM")
  field(IVOA,"Continue normally")
  field(IVOV,"0.0e+00")
}
```

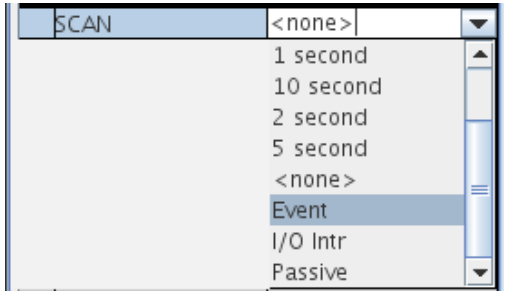
This shows only the design fields, there are other fields which are used only at run-time

# Fields are for…

- Defining
  - What causes a record to process
  - Where to get/put data from/to
  - How to turn raw I/O data into a numeric engineering value
  - Limits indicating when to report an alarm
  - When to notify value changes to a client monitoring the record
  - A Processing algorithm
  - Anything else which needs to be set for each record of a given type
- Holding run-time data
  - Input or output values
  - Alarm status, severity and acknowledgements
  - Processing timestamp
  - Other data for internal use

# Field Types

- Fields can contain
  - Integers `[1, 2, … ]`
    - char, short or long
    - signed or unsigned
  - Floating-point numbers `[ 0.1, 3.2, … ]`
    - float or double
  - Strings `["this is a string"]`
    - maximum useful length is 40 characters
  - Menu choices
    - select one from up to 16 strings
    - stored as a short integer
  - Links
    - to other records in this or other IOCs
    - to hardware signals (device support)
    - provide a means of getting or putting a value



# All Records Have These Fields

- Design fields
  - NAME 60 Character unique name (using >40 characters can cause problems)
  - DESC 40 Character description
  - ASG Access security group
  - SCAN Scan mechanism
  - PHAS Scan order (phase)
  - PINI Process at IOC initialization?
  - PRIO Scheduling priority
  - SDIS Scan disable input link
  - DISV Scan disable value
  - DISS Disabled severity
  - FLNK Forward link
- Run-time fields
  - PROC Force processing
  - PACT Process active
  - STAT Alarm status
  - SEVR Alarm severity
  - TPRO Trace processing
  - UDF Set if record value undefined
  - TIME Time when last processed

# Other Interesting Fields

- Input/Output
  - `INP` Input link
  - `OUT` Output link
  - `DOL` Desired output location
  - `RVAL` Raw value
- Conversion
  - `EGU` Engineering unit string
  - `LINR` Unit conversion control
  - `EGUL` Low engineering value
  - `EGUF` High engineering value
  - `ESLO` Unit conversion slope
  - `EOFF` Unit conversion offset
- Limits
  - `HOPR` High operating range
  - `LOPR` Low operating range
  - `DRVH` Drive high
  - `DRVL` Drive low
- Calculus
  - `CALC` Calculation
- Alarms
  - `HIGH` High alarm limit
  - `LOW` Low alarm limit
  - `HIHI` HiHi alarm limit
  - `LOLO` LoLo alarm limit
  - `HSV` High alarm severity
  - `LSV` Low alarm severity
  - `HHSV` HiHi alarm severity
  - `LLSV` LoLo alarm severity
  - `HYST` Alarm deadband
- Monitors
  - `ADEL` Archive deadband
  - `MDEL` Monitor deadband
- Runtime data
  - `ORAW` Old raw value
  - `PVAL` Previous value
  - `ORBV` Old readback value
  - `LALM` Last Alarm Monitor Trigger Value
  - `ALST` Last Archiver Monitor Trigger Value

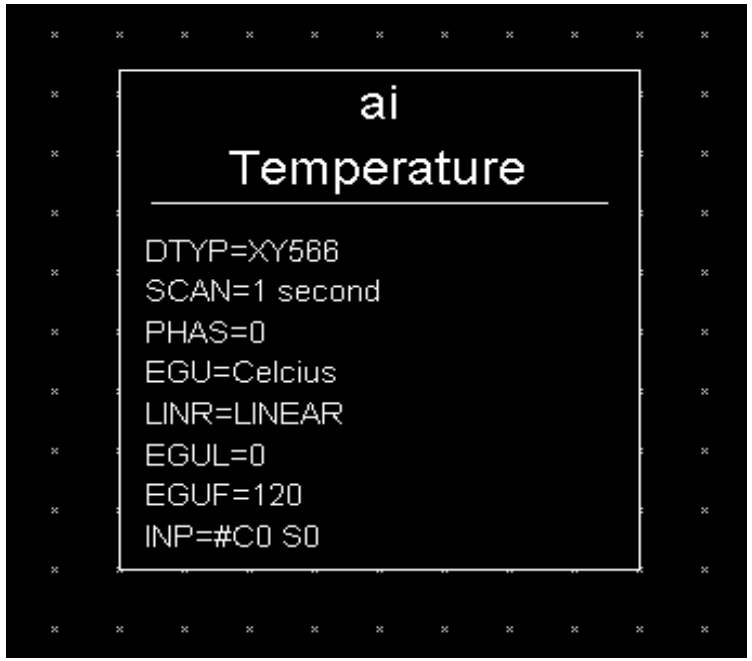
# Device Support

- Records do not access hardware directly.
- The Device Support layer performs I/O operations on request.
- A particular device support provides I/O for a single record type.
- The `DTYP` field determines which device support to use.
- The device support selected determines the format of the link (`INP` or `OUT` field) containing device address information.
- Adding new device support does not require change to the record software.
- Device support may call other software to do work for it (Driver Support).

# Record Scanning

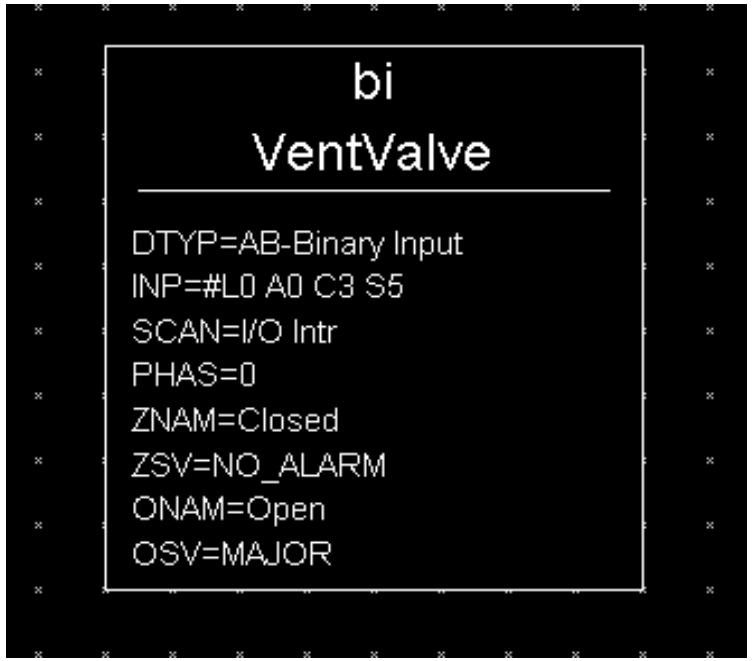
- The **SCAN** field is a menu choice from:
  - Periodic — 0.1 seconds .. 10 seconds
  - I/O Interrupt (if device supports this)
  - Soft event — `EVNT` field
  - Passive (default)
- The number in the **PHAS** field allows the processing order to be set within a scan.
  - Records with `PHAS=0` are processed first.
  - Then those with `PHAS=1`, `PHAS=2` etc.
- Records with **PINI**=`YES` are processed once at start-up.
- **PRIO** field selects Low/Medium/High priority for Soft event and I/O Interrupts.
- A record is also processed whenever any value is written to its **PROC** field.

# Periodically Scanned Analog Input



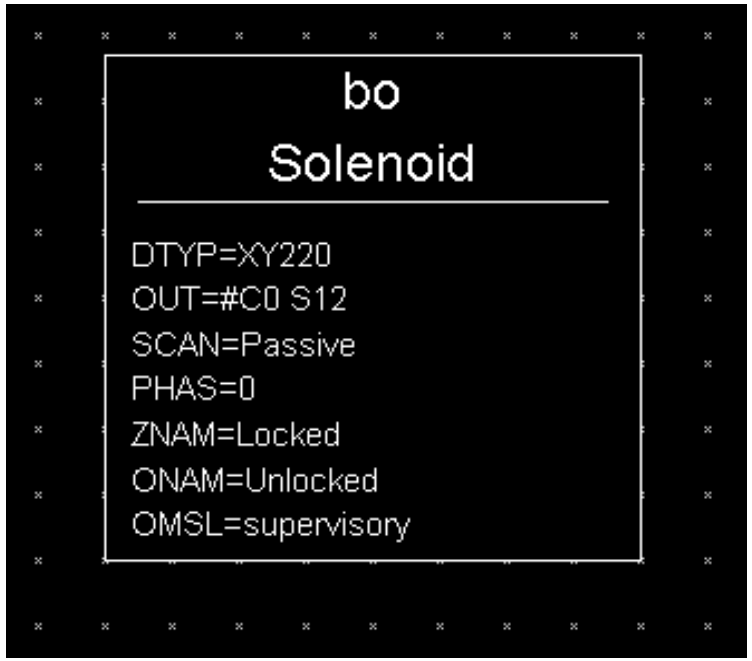

- Analogue Input “Temperature”
- Reads from the Xycom XY566 ADC Card 0 Signal 0
- Gets a new value every second
- Data is converted from ADC range to 0..120 Celsius

# Interrupt Scanned Binary Input




- Binary Input “VentValve”
- Reads from Allen-Bradley TTL I/O Link 0, Adaptor 0, Card 3, Signal 5
- Processed whenever value changes
- 0 = “Closed”, 1 = “Open”
- Major alarm when valve open

# Passive Binary Output




- Binary Output “Solenoid”
- Controls Xycom XY220 Digital output Card 0 Signal 12
- Record is only processed by:
  - Channel Access ‘put’ to a `PP` field (e.g. .`VAL`)
  - Another record writes to this one using `PP` flag
  - Forward Link from another record
  - Another record reads from this one using `PP` flag

# Links

- A link is a type of field, and is one of:
  - Input link
    - Fetches data
  - Output link
    - Writes data
  - Forward link
    - Points to the record to be processed once this record finishes processing.

# Input and Output links may be…

- Constant numeric value, e.g.:
  - `0`
  - `3.1415926536`
  - `1.6e-19`
- Hardware link
  - A hardware I/O signal selector, the format of which **depends** on **the device support layer**
- Process Variable link — the name of a record, which at run-time is resolved into:
  - Database link
    - Named record is in this IOC
  - Channel Access link
    - Named record not found in this IOC

# Hardware Links

- VME_IO
  - `#Cn Sn @parm`
  - Card, Signal
- INST_IO
  - `@parm`
- CAMAC_IO
  - `#Bn Cn Nn An Fn @parm`
  - Branch, Crate, Node, Address, Function
- AB_IO
  - `#Ln An Cn Sn @parm`
  - Or `#Ln Pn Cn Sn Fn @parm`
  - Link, Adaptor, Card, Signal, Flag
- GPIB_IO
  - `#Ln An @parm`
  - Link, Address
- BITBUS_IO
  - `#Ln Nn Pn Sn @parm`
  - Link, Node, Port, Signal
- BBGPIB_IO
  - `#Ln Bn Gn @parm`
  - Link, Bitbus Address, GPIB Address
- VXI_IO
  - `#Vn Cn Sn @parm`
  - or `#Vn Sn @parm`
  - Frame, Slot, Signal

# Database Links

- These comprise:
  - The name of a record in this IOC
    - `myDb:myRecord`
  - An optional field name
    - `.VAL` default
  - Process Passive flag
    - `NPP` default, no processing action
    - `PP` in case of **INPUT** links, request the target to process before fetching data, in case of **OUTPUT** links, request the target to process after writing data
  - Maximize Severity flag
    - `NMS` default, no change in record severity
    - `MS` maximize severity, propagate alarm severity from source to destination
    - `MSS` maximize severity and status
    - `MSI` maximize severity but only if invalid
- Example:

```
field(INP, "M1:current.RBV NPP MS")
```

- **Note:** An input database link with `PP` set that is pointing to an asynchronous input record will not wait for the new value from that record.

# Channel Access Links

- Specified like a database link
- Name specifies a record not found in this IOC
- Use Channel Access protocol to communicate with remote IOC
- May include a field name (default `.VAL`)
- `PP` Link flags are ignored:
  - Input links are always `NPP`
  - Output links follow `PP` attribute of destination field
  - This behavior is identical to all other CA clients
- `MS` Link flags apply to Input links:
  - Input links honors a given `NMS` (default) or `MS` flag
  - Output links are always `NMS`
- Additional flags for CA links
  - `CA` Forces a “local” link to use CA
  - `CP` On input link, process this record on CA monitor event
  - `CPP` Like `CP` but only process if `SCAN` is Process Passive

# pvAccess Links

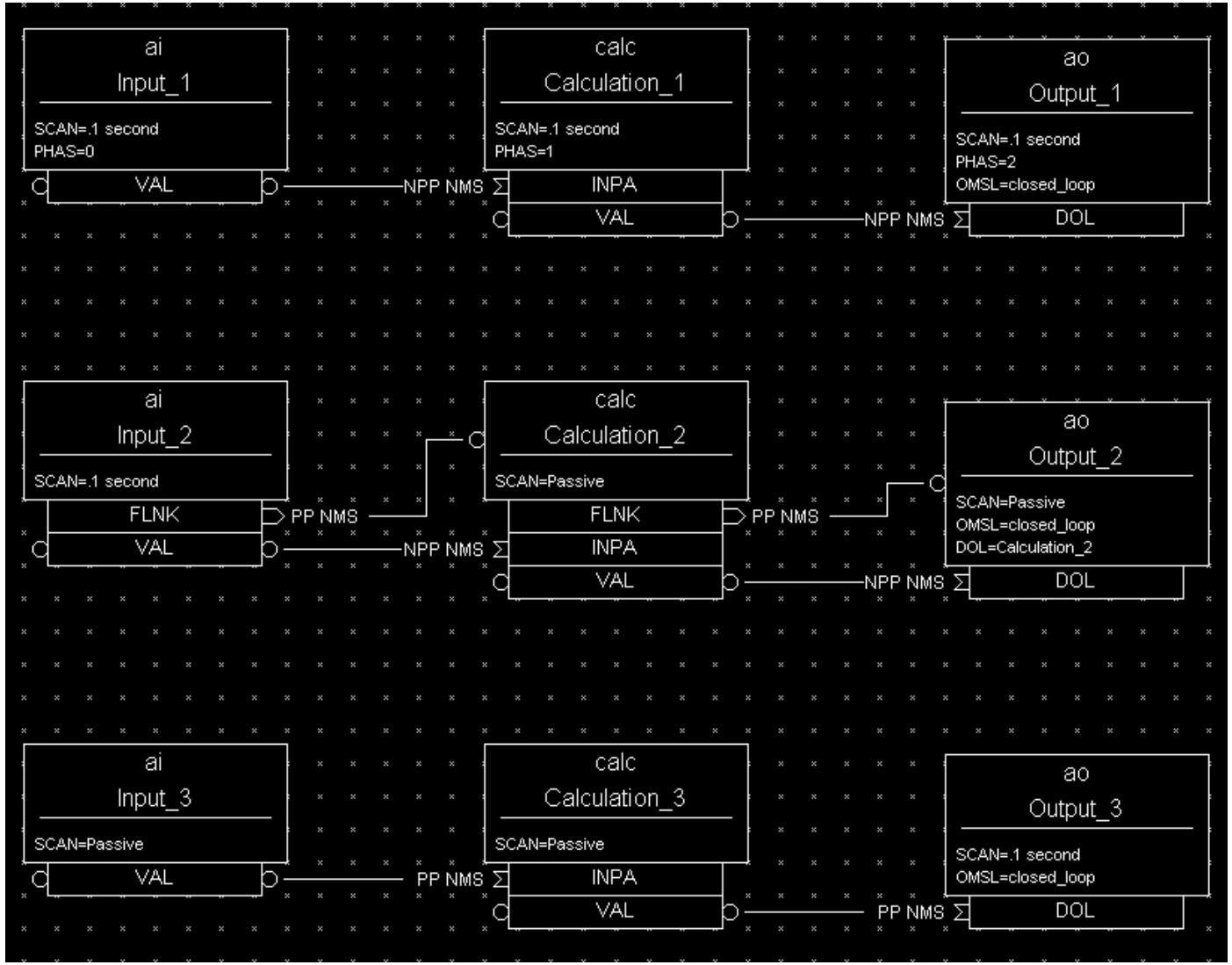
- Possible to link records over PVAccess
  - Data and control flow between records
  - Similar to CA links
- Links can be internal or external
  - Pointing to records in the same IOC or a different IOC
- Propagation of
  - Record processing
  - Alarm severity
    - Linked record inherits alarm severity from link source
- Data queue/buffer control
  - Q, pipeline

```
record(longin, “<PV>") {
  field(INP, {pva:{
    pv:“<PV_name>",
    field:"",   # may be a sub-field
    local:false,# Require local PV
    Q:4,        # monitor queue depth
    pipeline:false, # require that server uses monitor flow control protocol
    proc:none,  # Request record processing (side-effects).
    sevr:false, # Maximize severity.
    time:false, # set record time during getValue
    monorder:0, # Order of record processing as a result of CP and CPP
    retry:false,# allow Put while disconnected
    always:false,# CP/CPP input link process even when .value field hasn't changed
    defer:false # Defer put
  }})
}
```

# Forward Links

- Usually a Database link, referring to a record in the same IOC.
- Forward linking via Channel Access is possible but must explicitly name the `PROC` field of the remote record.
- No flags (`PP`, `NMS` etc.).
- Destination record is only processed if it has:
  - `SCAN = Passive`
- Does not pass a value, just causes subsequent processing.

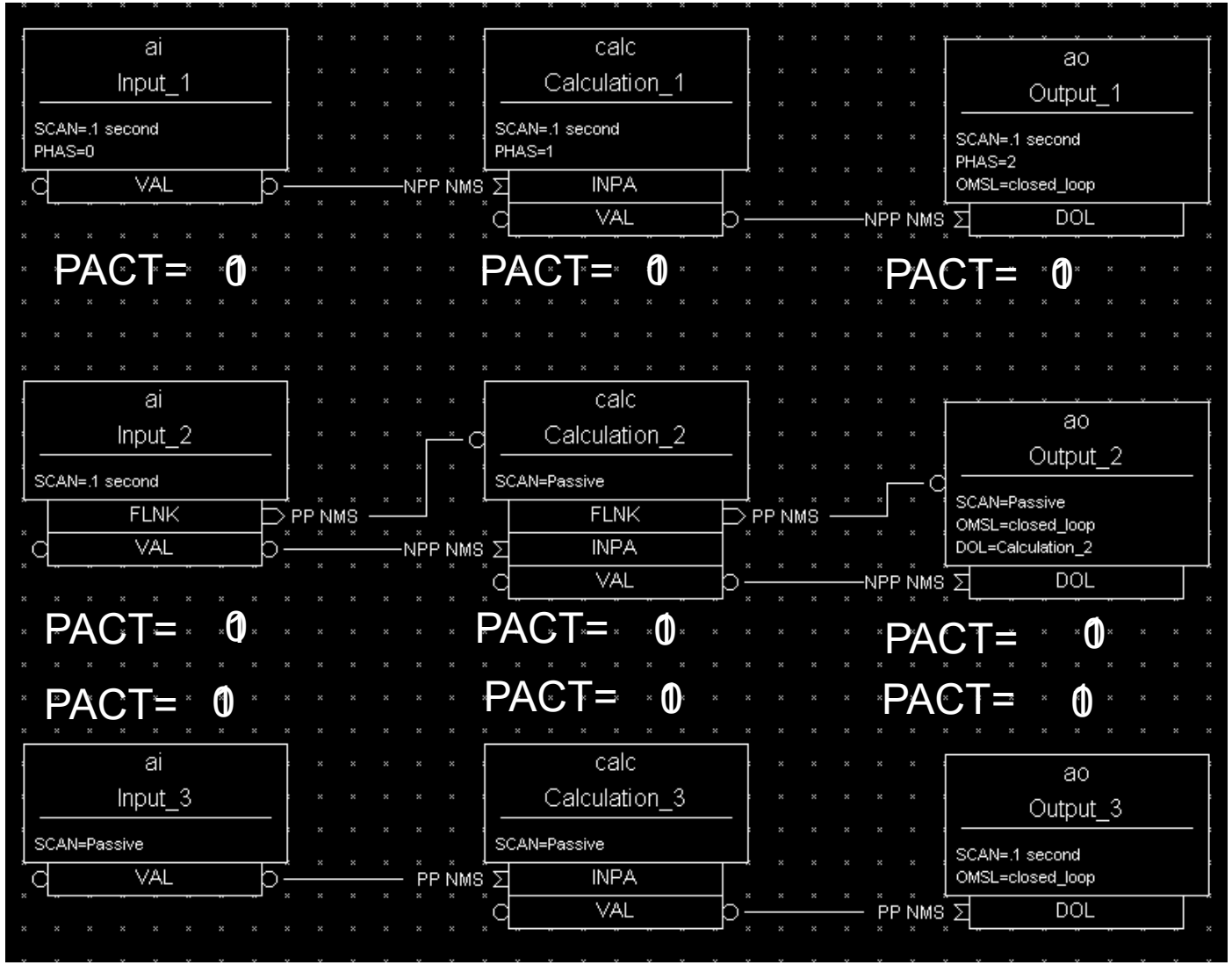
# Processing Chains



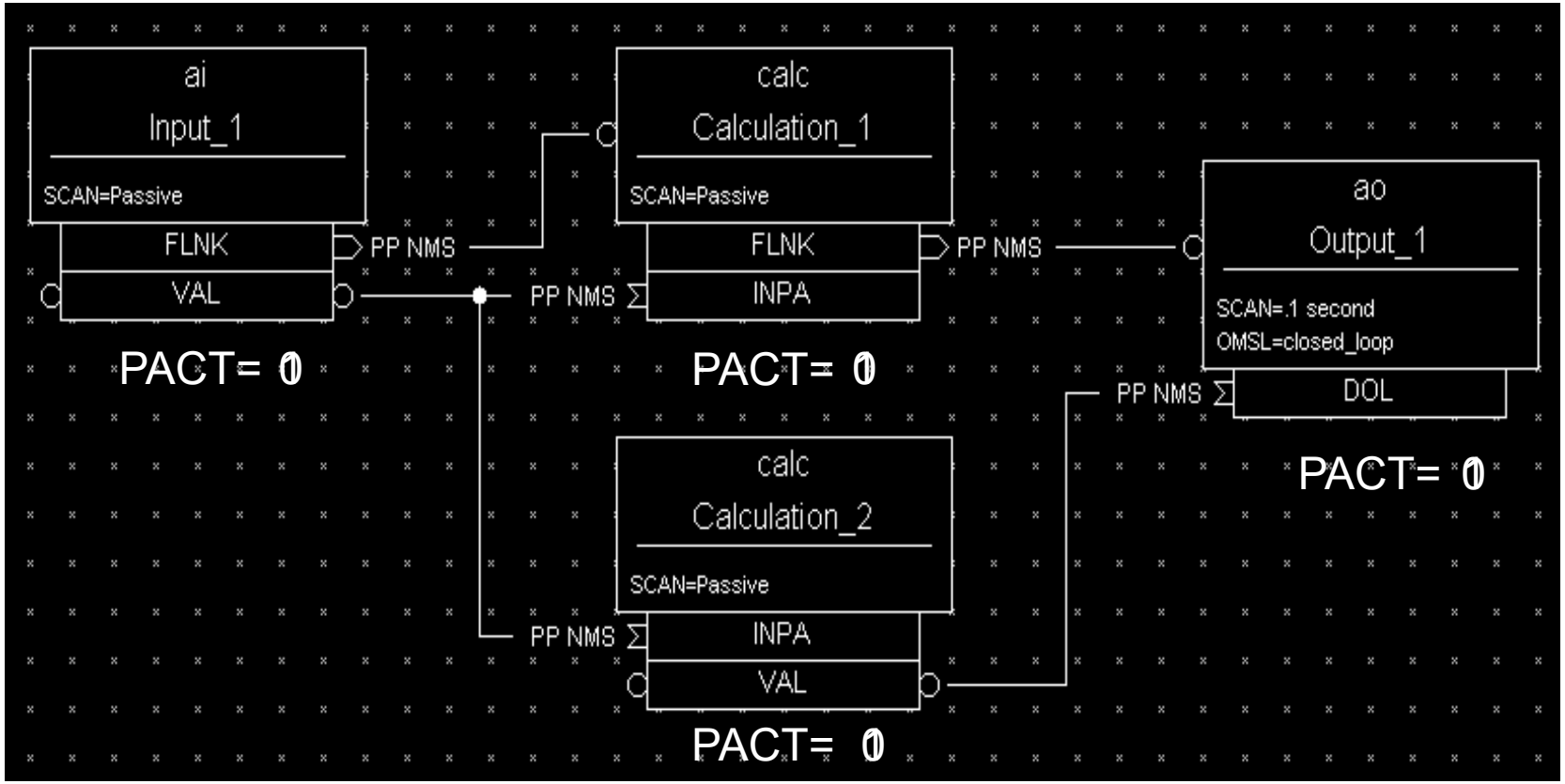
# The PACT Field

- Every record has a Boolean run-time field called `PACT` (Process Active)
- `PACT` breaks loops of linked records
- It is set to ‘true’ early in the act of processing the record
  - `PACT` is true whenever a link in that record is used to get/put a value
- `PACT` is set to false after record I/O and forward link processing are finished
- A `PP` link can never make a record process if it has `PACT` true
  - Input links take the current value
  - Output links just put their value

# Processing Chains



# What happens here?



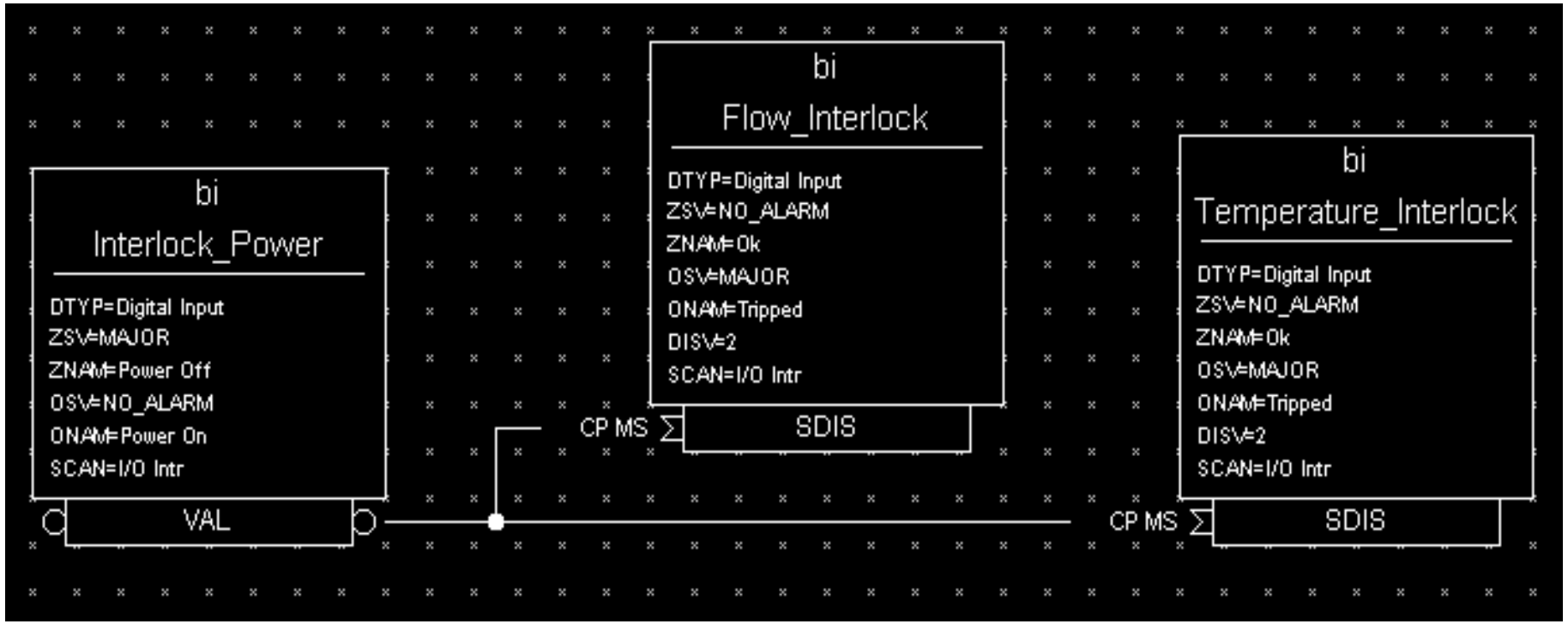
# Disable Processing

- It is useful to be able to stop an individual record from processing on some condition
- Before record-specific processing is called, a value is read through the `SDIS` input link into `DISA`
- If `DISA=DISV`, the record will not be processed
- A disabled record may be put into an alarm by giving the desired severity in the `DISS` field

- The `FLNK` of a disabled record is never triggered

# Database Example

- `Temp_Interlock` and `Flow_Interlock` get their values from the hardware. If the interlock power is `OFF`, both interlock records are disabled (their values do not change).




- How can we give the two interlock records an INVALID alarm severity when the Interlock Power is OFF?
- Are there any mistakes in the DB?

`SDIS` field

# How do records allocate CPU time?

- Several IOC tasks are used:
  - `callback` (3 priorities) — I/O Interrupt
  - `scanEvent` — Soft Event
  - `scanPeriod` — Periodic
    - A separate task is used for each scan period
    - Faster scan rates are given a higher task priority (if supported by the IOC's Operating System)

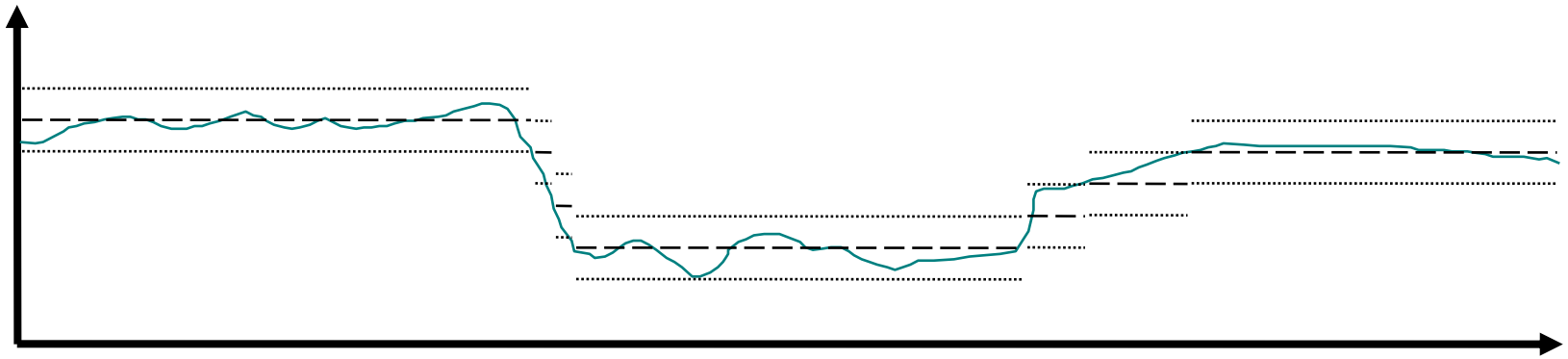
- Channel Access tasks use lower priority than record processing
  - If a CPU spends all its time doing I/O and record processing, you may be unable to control or monitor the IOC via the network

# Alarms

- Every record has the fields
  - SEVR Alarm Severity
    - NONE, MINOR, MAJOR, INVALID
  - STAT Alarm Status (reason)
    - READ, WRITE, UDF, HIGH, LOW, STATE, COS, CALC, DISABLE, etc.
- Most numeric records check VAL against HIHI, HIGH, LOW and LOLO fields after the value has been determined
- The HYST field prevents alarm chattering
- A separate severity can be set for each numeric limit (HHSV, HSV, LSV, LLSV)
- Discrete (binary) records can raise alarms on entering a particular state, or on a change of state (COS)

# Change notification: Monitor deadbands

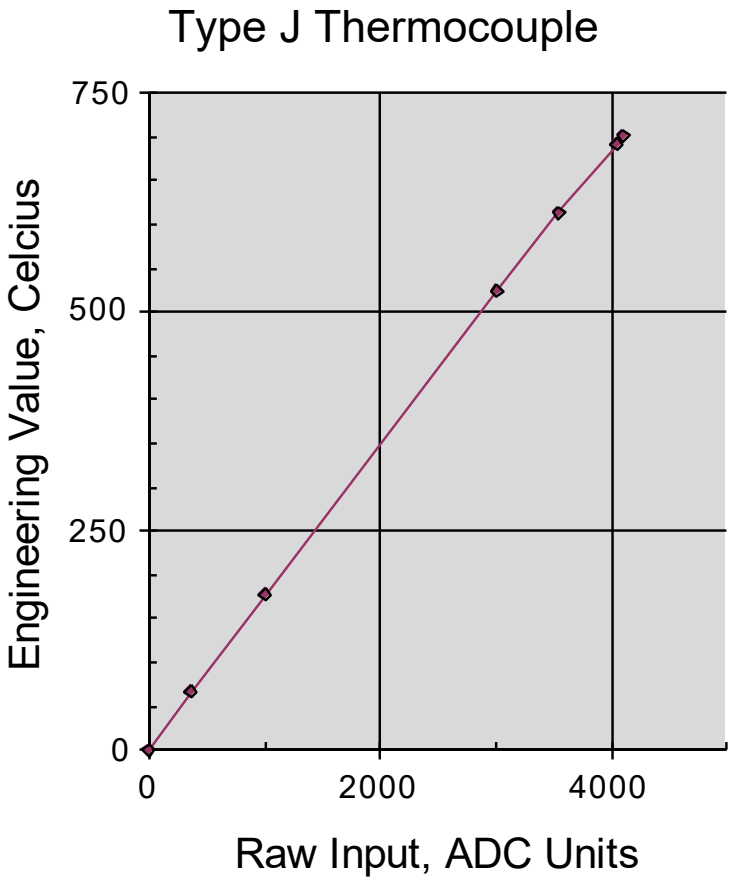
- Channel Access notifies clients which are monitoring a numeric record when
  - VAL changes by more than the value in field:
    - MDEL Value monitors
    - ADEL Archive monitors
  - Record's Alarm Status changes
    - HYST Alarm hysteresis
  - Analogue Input record provides smoothing filter to reduce input noise (SMOO)



# Breakpoint Tables

- Analogue Input and Output records can do non-linear conversions from/to the raw hardware value
- Breakpoint tables interpolate values from a given table
- To use, set the record's `LINR` field to the name of the breakpoint table you want to use (e.g. `typeJDegC`)
- Example breakpoint table (in some loaded .dbd file)

```
breaktable(typeJDegC) {
        0.000000   0.000000
      299.268700  74.000000
      660.752744 163.000000
     1104.793671 274.000000
     1702.338802 418.000000
     2902.787322 703.000000
     3427.599045 831.000000
     ...
}
```



# Simulation

- Input and output record types often allow simulation of hardware interfaces
  - `SIML` Simulation mode link
  - `SIMM` Simulation mode value
  - `SIOL` Simulation input link
  - `SIMS` Simulation alarm severity
- Before using its device support, a record reads `SIMM` through the `SIML` link
- If `SIMM=YES`, device support is ignored; record I/O uses the `SIOL` link instead
- An alarm severity can be set whenever simulating, given by the `SIMS` field.

# Access Security

- A networked control system must have the ability to enforce security rules
  - Who can do what from where, and when?
- In EPICS, security is enforced by the CA server (typically the IOC).
- A record is placed in the Access Security Group named in its ASG field
  - DEFAULT is used if no group name is given
- Rules for each group determine whether a CA client can read or write to records in the group, based on
  - Client user ID
  - Client IP address
  - Access Security Level of the field addressed
  - Values read from the database

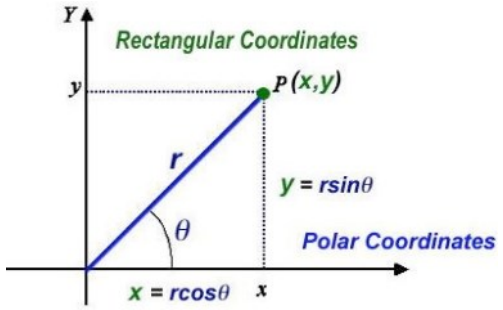
# Access Security Configuration File

- Security rules are loaded from an Access Security Configuration File, for example:

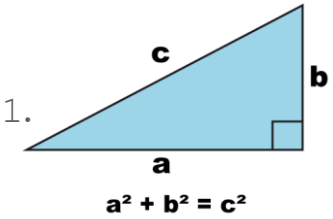
```
UAG(users) {user1, user2}
HAG(hosts) {host1, host2}
ASG(DEFAULT) {
    RULE(1, READ)
    RULE(1, WRITE) {
        UAG(users)
        HAG(hosts)
    }
}
```

- If no security file is loaded, Security will be turned off and nothing refused
- For more details and the rule syntax, see Chapter 8 of the IOC Application Developers Guide.

# EPICS 7 enhancement
## Atomic access




- A simple semi-realistic example
  - Rotating unit vector
    - Polar vs. Cartesian coordinates
  - Internally incrementing angle PV
  - Cartesian coordinate PVs
    - circle:X
    - circle:Y
  - Test on client side: use Pythagoras' theorem on received values. Radius `r` should always be `1`.




- Using CA: two independent channels.
  - Correctness depends on simultaneity ("mostly" OK but not guaranteed)
  - May appear to work if run on one host, or on a simple network.
- Using PVA: calculated and transported as a single unit.
  - Correctness guaranteed, regardless of network or IOC load.
- Using `info` tags in the EPICS database, we can create group PVs
  - Combine data from different records
  - group PVs are served by the PVA server. Addressed by the group name.

# EPICS 7 enhancement
## Groups

- Use of `info` tags to configure groups
  - Added to “V3” EPICS records; no other configuration needed.
  - Would be a no-op in a V3 IOC
  - Use `trigger` keyword to process the group (send monitors)
- Example on right: creates a group `circle`
  - NTTable normative type (version 1.0)
  - `Value` consists of two scalars, `X` and `Y`

Using pvget we see:

```
$ pvget circle
circle epics:nt/NTTable:1.0
    structure record
        structure _options
            uint queueSize 0
            boolean atomic true
    double angle 16
    alarm_t alarm
        int severity 0
        int status 0
        string message NO_ALARM
    time_t timeStamp 2019-06-24
16:08:17.546
        long secondsPastEpoch 1561385297
        int nanoseconds 546217000
        int userTag 0
    structure value
        double X 0.961262
        double Y 0.275637
```

```
record(calc, "circle:angle") {
  field(PINI, "RUNNING") # bootstrap
  field(INPA, "circle:angle NPP")
  field(INPB, "circle:step NPP")
  field(INPD, "360")
  field(CALC, "C:=A+B;(C>=D)?C-D:C")
  info(Q:group, {
        "circle":{ +id:"epics:nt/NTTable:1.0",
            "angle":{+type:"plain",
                                +channel:"VAL"}}
  })
}
record(calc, "circle:x") {
  field(INPA, "circle:angle NPP")
  field(CALC, "cos(A*PI/180)")
  field(TSEL, "circle:angle.TIME")
  field(FLNK, "circle:y")
  field(PREC, "3")
  info(Q:group, {
        "circle":{ "":{+type:"meta", +channel:"VAL"},
            "value.X":{+type:"plain",
                                +channel:"VAL",
                                +trigger:"*"} }
  })
}
record(calc, "circle:y") { <idem> } (except “trigger” keyword)
```

Based on a presentation by :
T. Mooney (APS)

COSYLAB

# Getting Started with EPICS

## Record Types and Examples

# Scope

- This lecture:
  - Existing record types and what they can do
  - Record-type documentation
  - Where to look for record types
- Related topics not covered in this lecture:
  - What is a record?
    - Database – Concepts and linking
  - How do I connect a record instance to a device?
    - set the link field (Database– Concepts and linking)
  - How do I connect a record type to a device?
    - Finding and deploying I/O support -- or, if not found...
    - Writing device support
  - How do I write a new record type?
    - Writing Record Support

# EPICS Record Types

- Where do record types come from?
  - EPICS Base (`<base>/modules/database/src/std/rec`)
    - General purpose record types
    - No record-type specific operator displays or databases
    - Documentation in EPICS Component Reference Manual
  - EPICS collaboration
    - General purpose, and application-specific, record types
    - Some are supported for use by collaborators (some are NOT)
    - Some come with record-type specific displays, databases
  - Custom record types can be written by an EPICS developer and added to an EPICS application.
    - Not in the scope of this lecture

# Component Reference Manual

- Where is it?
  - Software > EPICS Base > EPICS 7 > <release> > Other Links > IOC Component Reference Documentation
  - https://epics.anl.gov/base/R7-0/8-docs/ComponentReference.html
- What is in it?
  - Database Concepts (good review)
  - Fields common to all records
  - Fields common to many records
  - Record Types – provides a description of the record processing routines for most of the record types in the base.
- When would I use it?
  - Skim through before writing any databases
  - Read through before writing any records
  - Otherwise, use as reference

# Component Reference Manual (cont.)

- Introduction to EPICS, Process Database Concepts
  - Note special meaning of  the words scan, process, address, link, and monitor
- Record references
  - Descriptions of record fields, processing, and useful info for writing device support
  - Contains lots of tables like the following:

| Field | Summary | Type | DCT | Default | Read | Write | CA PP |
|---|---|---|---|---|---|---|---|
| EGU | Engineering Units | STRING [16] | Yes | null | Yes | Yes | No |
| HOPR | High Operating Range | FLOAT | Yes | 0 | Yes | Yes | No |
| LOPR | Low Operating Range | FLOAT | Yes | 0 | Yes | Yes | No |
| PREC | Display Precision | SHORT | Yes | 0 | Yes | Yes | No |
| NAME | Record Name | STRING [29] | Yes | Null | Yes | No | No |
| DESC | Description | STRING [29] | Yes | Null | Yes | Yes | No |

# Collaboration Supported Records

- Where are they found?
  - Soft-support list (search for record)
    - https://epics-controls.org/resources-and-support/modules/soft-support/
  - The tech-talk email list: tech-talk@aps.anl.gov
  - The soft-support list contains entries like this (among entries for other kinds of soft support):

| Class | Name | Description | Contact | Link |
|---|---|---|---|---|
| record | epid | Enhanced PID record | Mark Rivers | CARS:epidRecord |
| record | genSub | Multi-I/O subroutine, handles arrays | Andy Foster | OSL:epics |
| ... | ... | ... | ... | ... |
| record | table | Control an optical table | Tim Mooney | APS:synApps/optics |

# Record Types

# Input Records

- `ai` - Analog input [BASE]
  - Read analog value, convert to engineering units, four alarm levels, simulation mode
- `bi` - Binary input [BASE]
  - Single bit, two states, assign strings to each state, alarm on either state or change of state, simulation mode
- `mbbi` - Multi-bit binary input [BASE]
  - Multiple bit, 16 states, assign input value for each state, assign strings to each state, assign alarm level to each state, simulation mode
- `mbbiDirect` – mbbi variant [BASE]
  - Read an unsigned short and map each bit to a field (32 `bi` records in one)

# Input Records (cont.)

- `stringin` - String input [BASE]
  - 40 character (max) ascii string,  simulation mode
- `longin` - Long integer input [BASE]
  - Long integer, four alarm levels, simulation mode
- `int64in` – 64bit integer input [BASE]
  - 64bit integer, four alarm levels, simulation mode
- `waveform` – array input [BASE]
  - Configurable data type and array length

# Output Records

- `ao` - Analog output [BASE]
  - Write analog value, convert from engineering units, four alarm levels, closed_loop mode, drive limits, output rate-of-change limit, INVALID alarm action, simulation mode
- `bo` - Binary output [BASE]
  - Single bit, two states, assign strings to each state, alarm on either state or change of state, closed_loop mode, momentary 'HIGH', INVALID alarm action, simulation mode
- `longout` [BASE]
  - Write long integer value, four alarm levels, closed_loop mode, INVALID alarm action, simulation mode
- `int64out` [BASE]
  - Write 64bit integer value, four alarm levels, closed_loop mode, INVALID alarm action, simulation mode

# Output Records (cont.)

- `mbbo` - Multi-bit binary output [BASE]
  - Multiple bit, 16 states, assign output value for each state, assign strings to each state, assign alarm level to each state, closed_loop mode, INVALID alarm action, simulation mode
- `mbboDirect` - mbbo variant [BASE]
  - 32 settable bit fields that get written as a short integer to the hardware, closed_loop mode, INVALID alarm action, sim. mode
- `motor` [synApps]
  - Controls stepper and servo motors
- `stringout` [BASE]
  - Write a character string (40 max), closed_loop mode, INVALID alarm action, simulation mode

# Algorithms/Control Records
## *Calc*

- `calc` - run-time expression evaluation [BASE]
  - 12 input links, user specified “calc expression” (algebraic, trig, relational, Boolean, Logical, “?”), four alarm levels
  - Sample expressions:
    - `0 read: “<calc_record>.VAL = 0”`
    - `A note ‘A’ refers to <calc_record>.A`
    - `A+B`
    - `sin(a)`
    - `(A+B)<(C+D)?E:F+L+10`
- `calcout` – calc variant [BASE]
  - Conditional output link, separate output CALC expression (`.OCAL`), output delay, and output event
  - Output-link options : `Every Time`, `On Change`, `When Zero`, `When Non-zero`, `Transition To Zero`, `Transition To Non-zero`

# Algorithms/Control Records
## *Calc*

- `sCalcout` – calcout variant [synApps]
  - Has both numeric fields (A,B,..L) and string fields (AA,BB,..LL)
  - Supports both numeric and string expressions. E.g.,
    - `A+DBL("value is 3.456") -> 3.456`
    - `printf("SET:VOLT:%.2lf", A+4) -> "SET:VOLT:5.00"`
  - Additional output-link option: "Never"
- `transform` – calc/seq variant [synApps]
  - Like 16 calcout records (but outlinks are not conditional)
  - Expressions read all variables but write to just one.
  - Uses sCalcout record's calculation engine
  - Example expressions:
    - `A: 2 read: "<transform>.A = 2"`
    - `B: A+1+C uses new value of 'A', old value of 'C'`

# Algorithms/Control Records
## *Subroutine*

- Goal: Connect subroutine (C code) to a record

- `sub` – Subroutine [BASE]
  - 12 input links, user provided subroutine
- `aSub` – Array subroutine [BASE]
  - Type of data could be selected
  - Up to 21 inputs and outputs
- Fields:
  - `INAM` - Initialization Subroutine Name
  - `SNAM` - Subroutine Name
  - `SUBL` - Subroutine Link [`aSub`]
- Processing
  - Synchronous
  - Asynchronous

# Algorithms/Control Records
## *Subroutine (impl.)*

- C subroutine example (synchronous).

```
long subInit(subRecord *psub) {
    printf("subInit was called\n");
    return(0);
}

long subProcess(subRecord *psub) {
    psub->val++;
    return(0);
}
```

# Algorithms/Control Records
## *Processing*

- `dfanout` – Data fanout [BASE]
  - Writes a single value to eight output links
- `fanout` [BASE]
  - Forward links to 16 other records.
  - Selection mask
- `sel` - Select [BASE]
  - 12 input links, four select options [specified, highest, lowest, median], four alarm levels
- `seq` - Sequence [BASE]
  - 16 “Input link/Value/Output link” sets: [in-link, delay, value, out-link]
  - Selection mask

# Algorithms/Control Records
## *Analysis*

- `subArray` [BASE]
  - Extracts a sub-array from a waveform.
- `compress` [BASE]
  - The data compression record is used to collect and compress data from arrays.
  - Input link can be scalar or an array.
  - Algorithms include N to 1 compression (highest, lowest, or average), circular buffer of scalar input.
- `histogram` [BASE]
  - Accumulates histogram of the values of a scalar PV

COSYLAB

# Examples of Custom Records

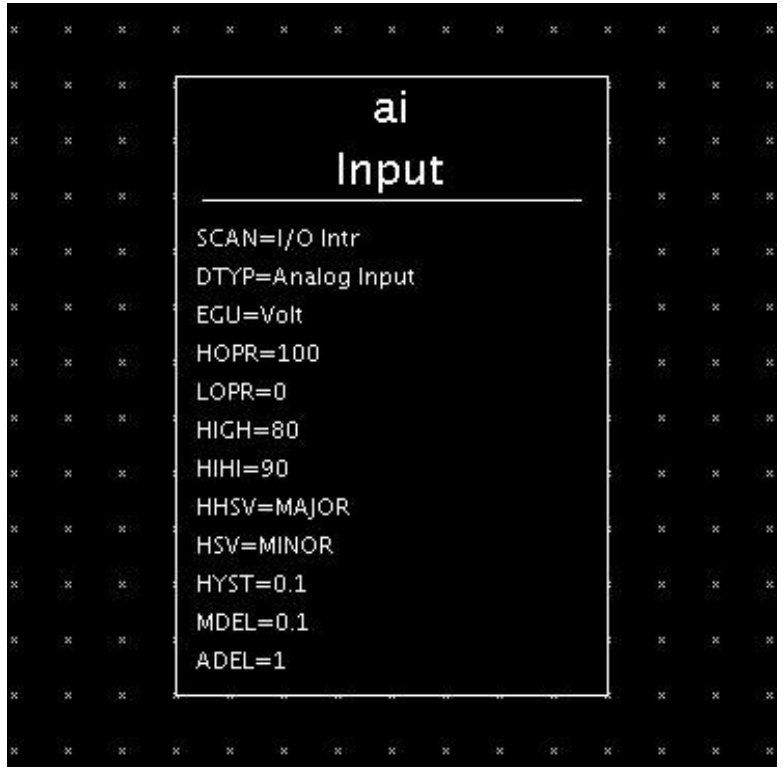
- `rf` - RF Amplitude Measurements [ANL]
  - Sample time, measurement in watts and db, waveform acquired through sweeping sample time
- `bpm` - Beam Position Monitor [ANL]
  - Four voltage inputs, numerous calibration constants, X-Y-I outputs, waveforms for each input
- Many others that are site-specific

# Examples

# Database Example
## *Analog Input (AI)*

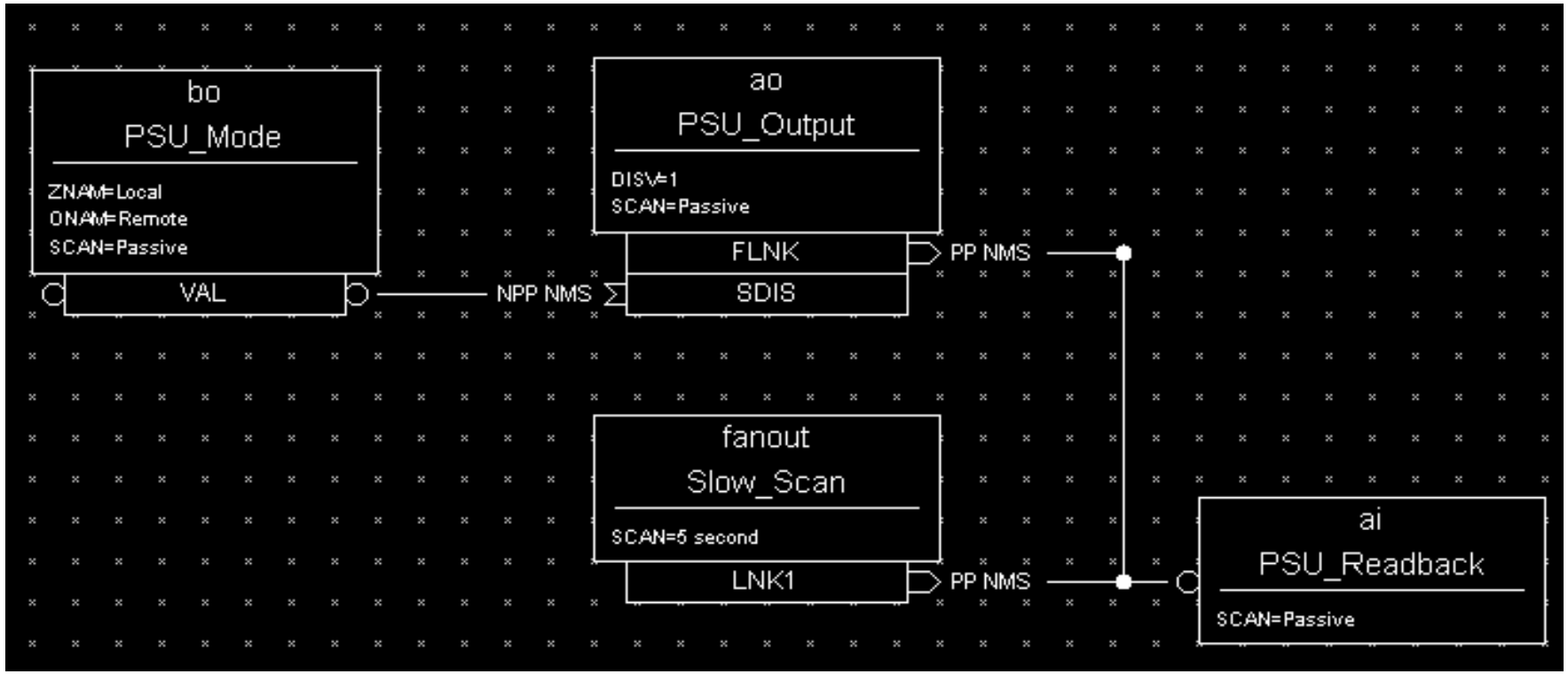
- An analog input record
  - Reading a voltage in Volts
  - Operating range from 0 V to 100 V.
  - Limit for a minor alarm is 80 V
  - Limit for a major alarm is 90 V

There is a hysteresis associated with the alarm limits and a deadband for reporting value changes to monitors and archivers.



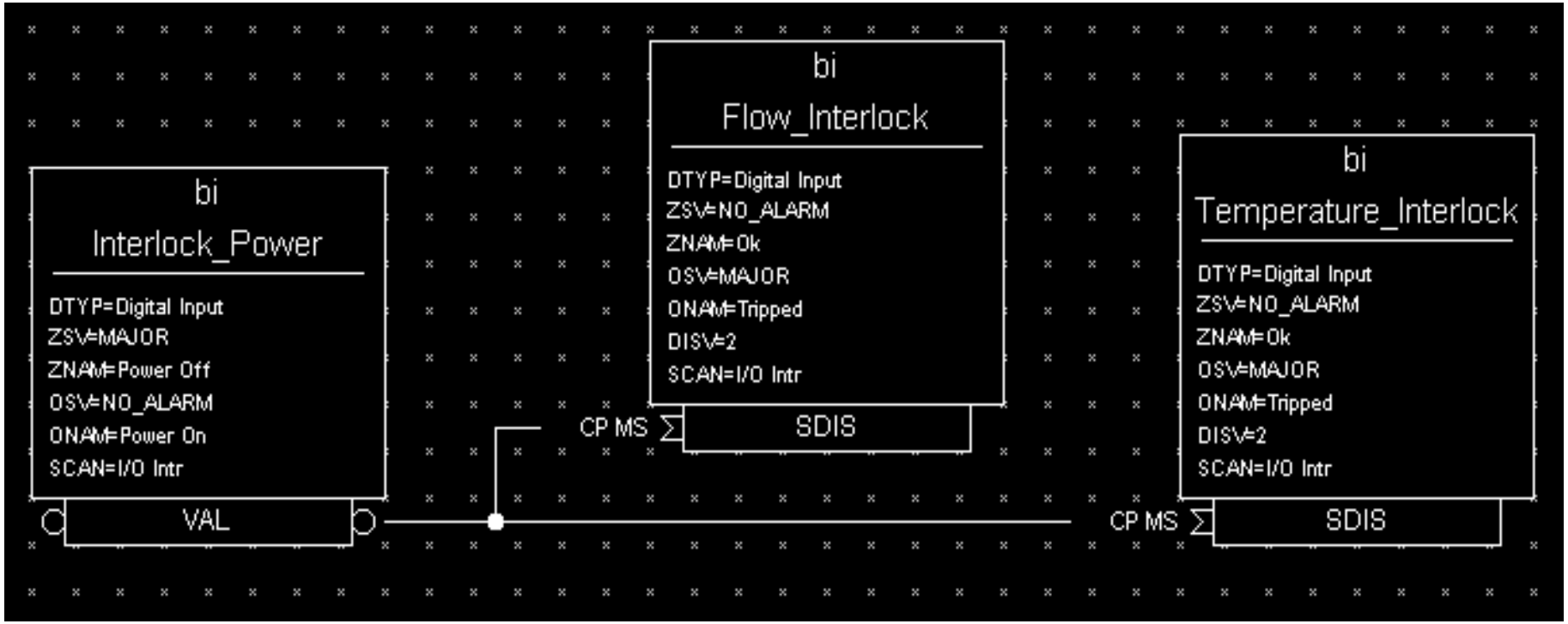
# Database Example
## *Processing*

- Slow Periodic Scan with Fast Change Response



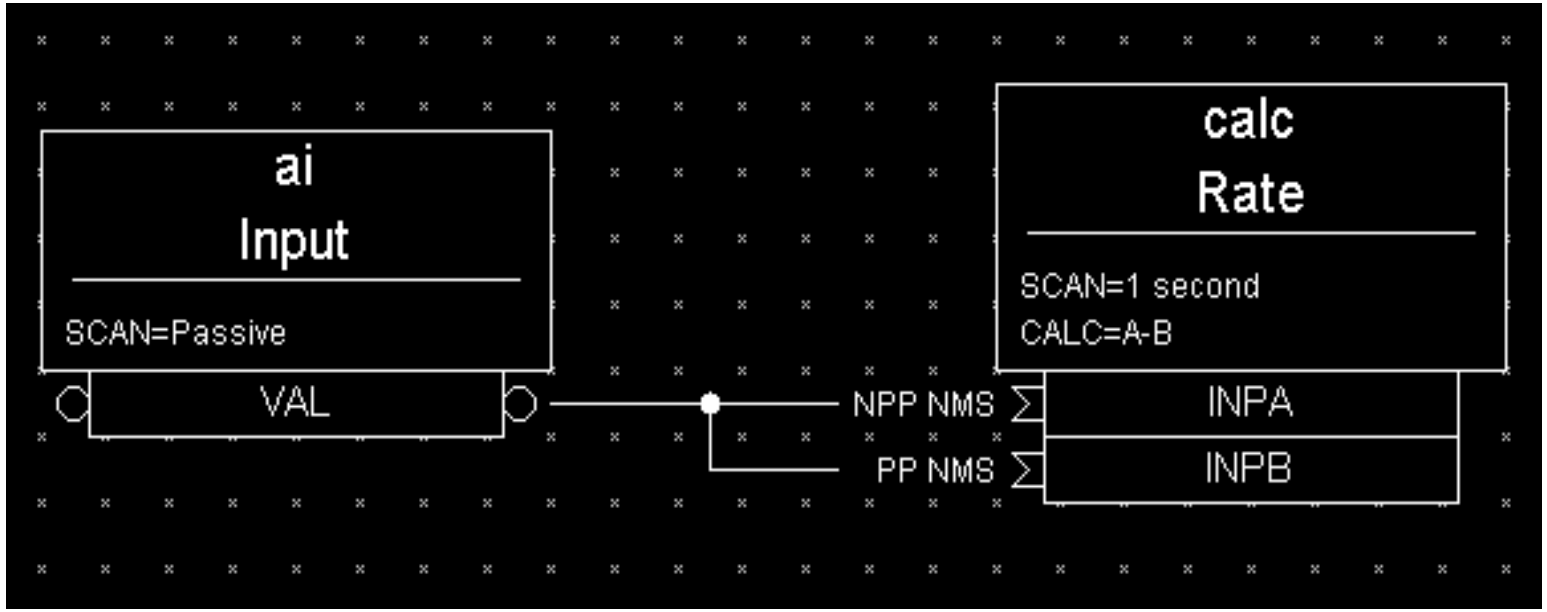

- The `ai` record gets processed every 5 s AND when the associated `ao` record is changed. This provides an immediate response to a change even though the desired scan rate is very slow. Changes to the power supply settings are inhibited by the `bo` record, which could represent a Local/Remote switch.

# Database Example *(Process Control)*



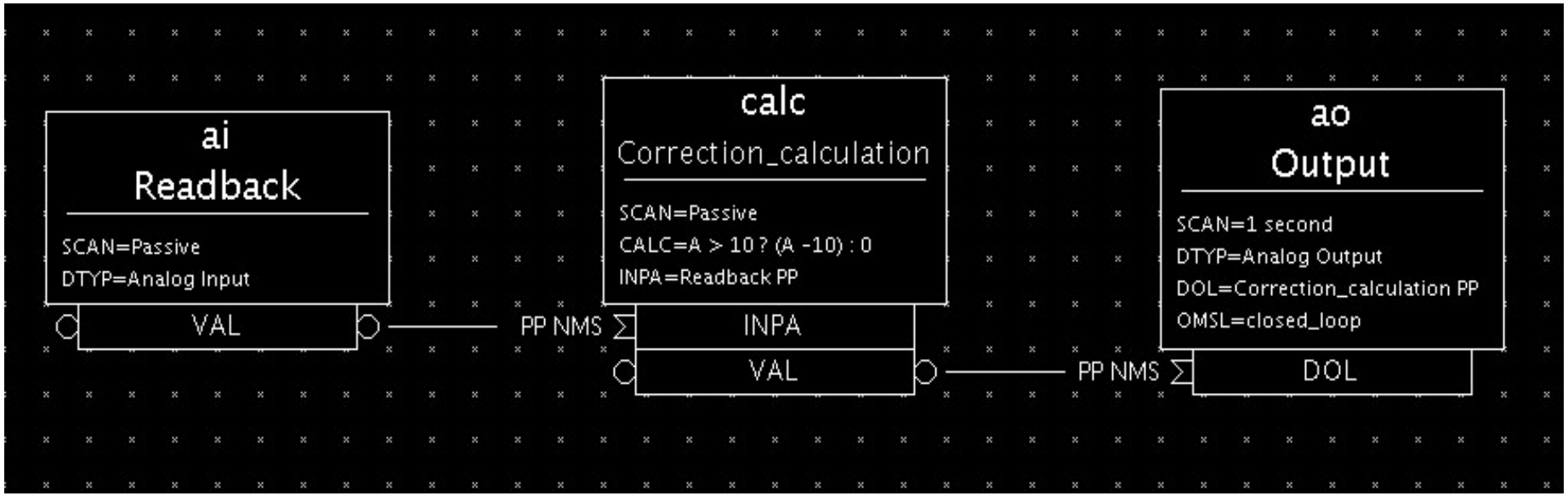

- `Temp_Interlock` and `Flow_Interlock` get their values from the hardware. Both trigger a `MAJOR` alarm if their value is `1`. If the interlock power is `OFF`, both interlock records are disabled (their values do not change).
- How can we give the two interlock records an `INVALID` alarm severity when the Interlock Power is off? Are there any mistakes in the DB?

# Database Example
## *Calc ("Rate-of-Change" of Input)*



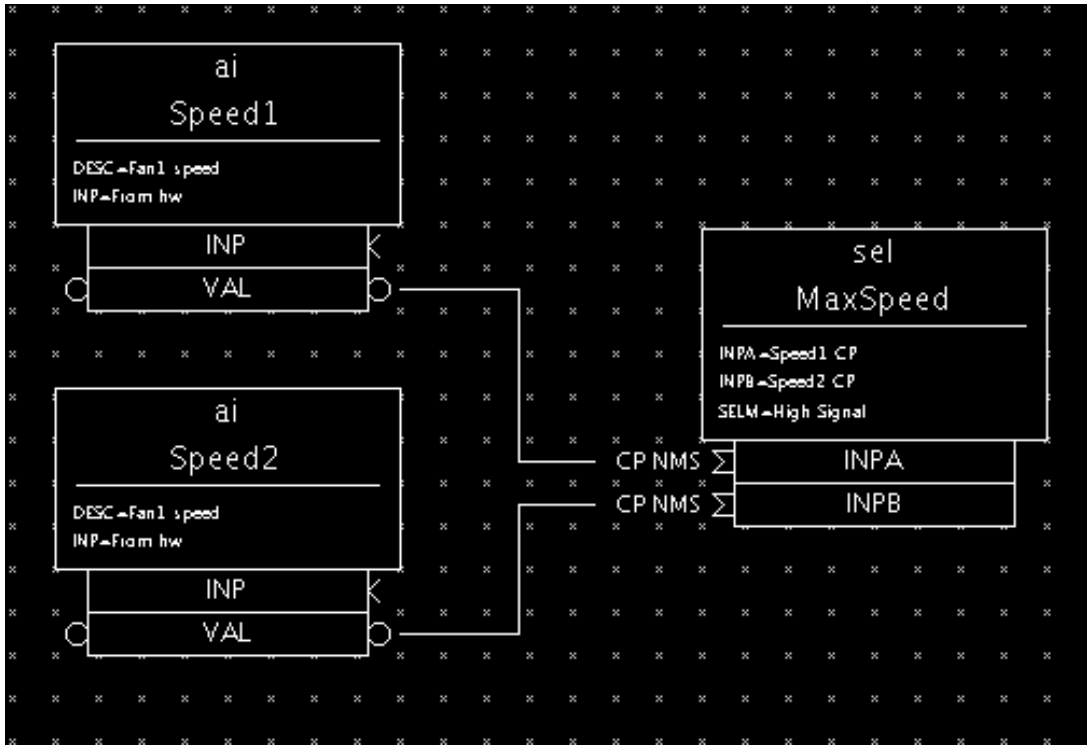

- `INPA` fetches data that is 1 s old because it does not request processing of the `ai` record. `INPB` fetches current data because it requests the `ai` record to process. The subtraction of these two values reflects the 'rate of change' (difference/sec) of the reading.

# Database Example
## *Simple Control*



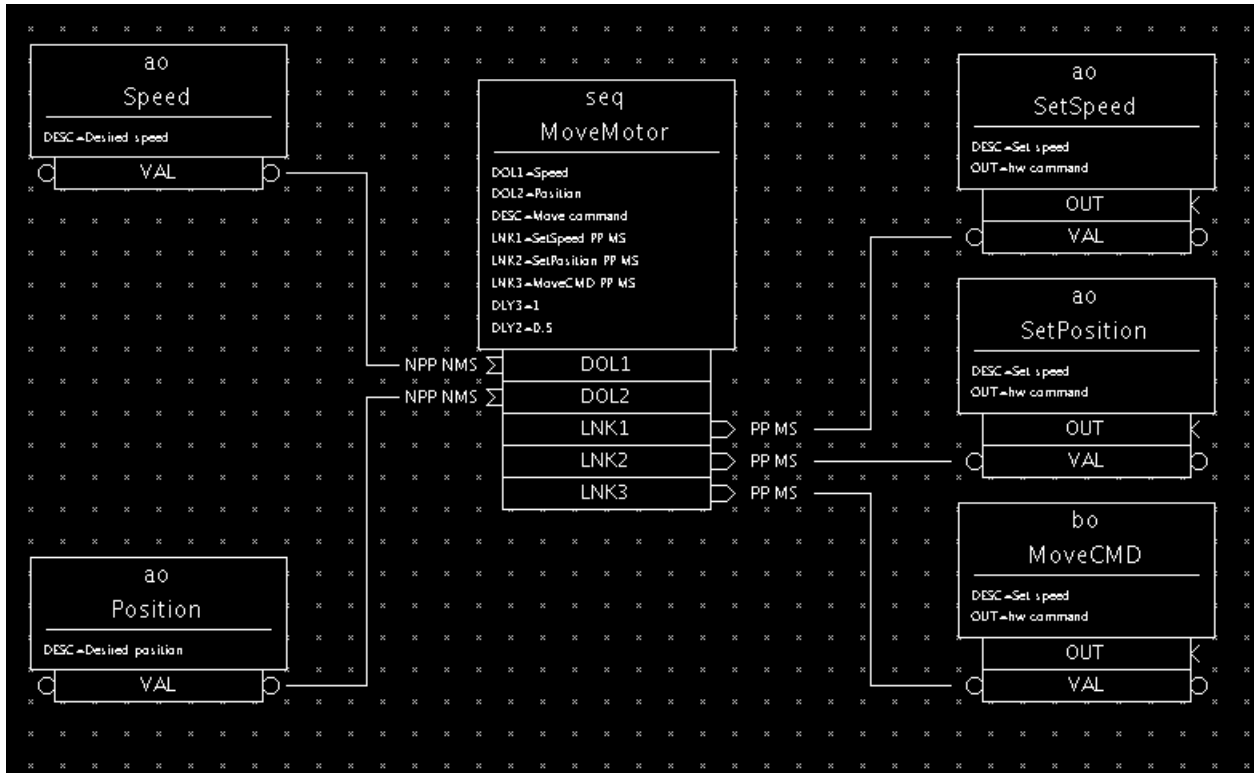

- An `ao` record triggers every second and requests the correction calculation from the `calc` record. The calc record requests the readback value from `ai` record, calculates the correction and the `ao` finishes its processing and outputs the correction.

# Database Example
## *Select*



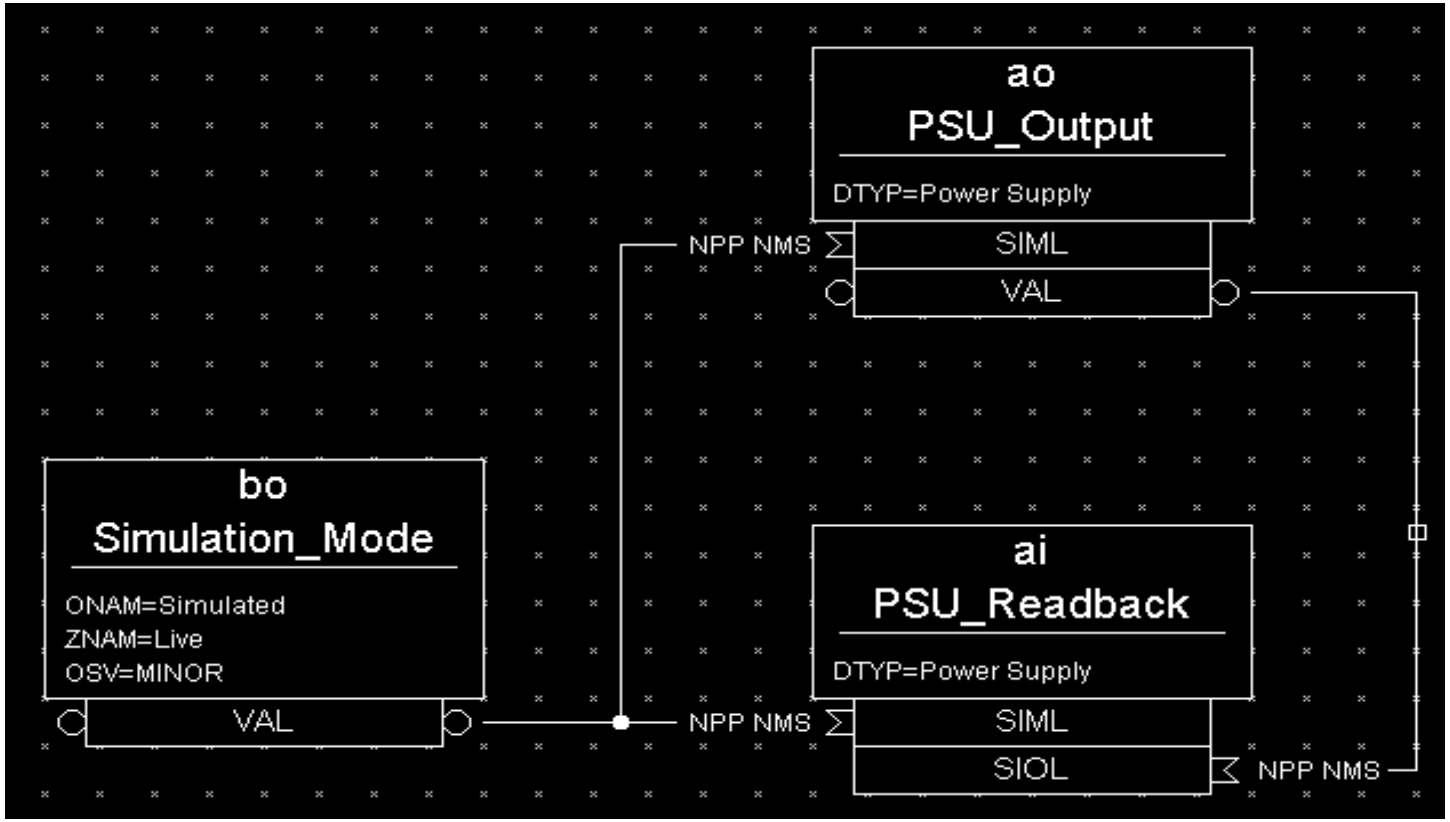

- The speeds of two fans are read from the hardware. The `select` record monitors their values and sets its value to the highest speed.

# Database Example
## *seq*



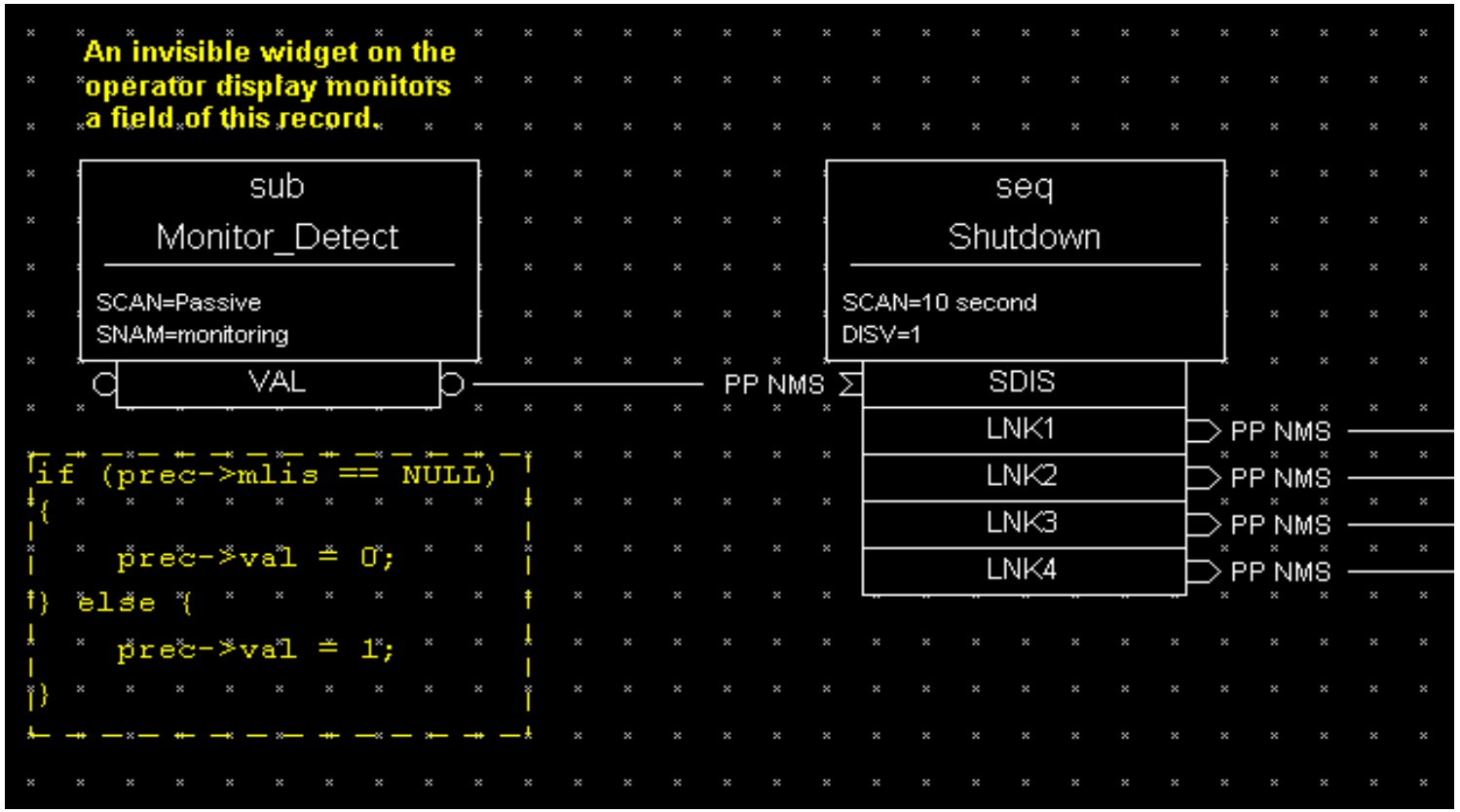

- Users set motor speed and desired position with the `ao` records. When an actual move is desired, the `seq` record fetches data from the `ao` records and sends it to the hardware. When the speed and position are set, the move command is executed.

# Database Example
## *Simulation Mode*




- When in simulation mode, the `ao` record does not call device support and the `ai` record fetches its input from the `ao` record.

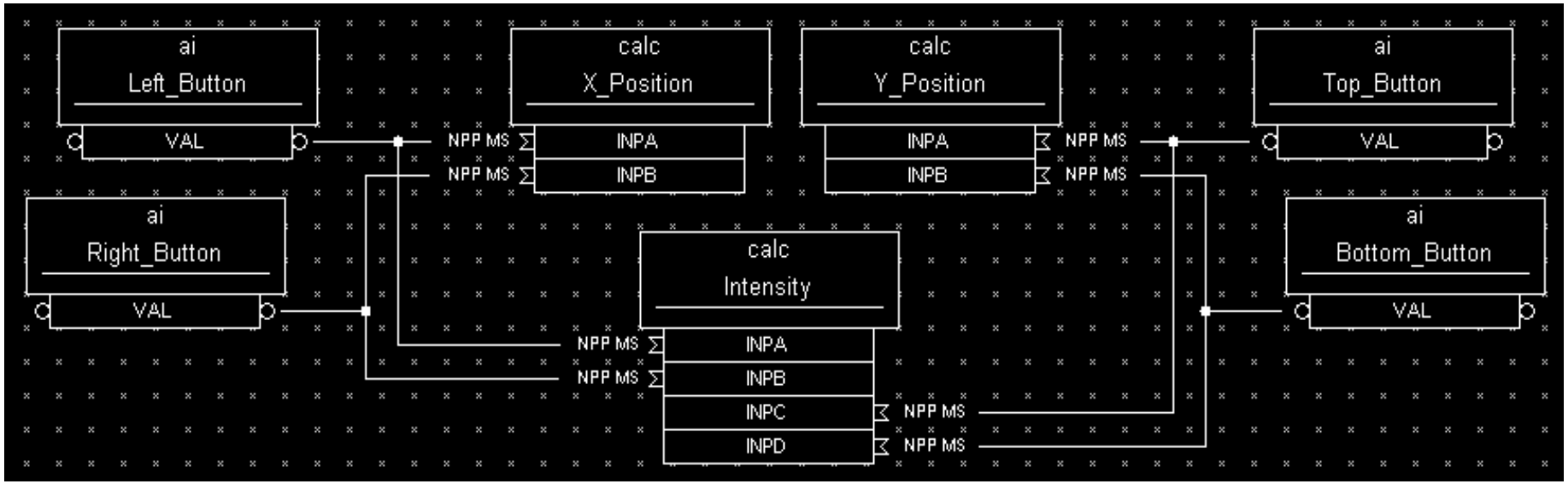
# Database Example
## *Automatic Shutdown on Logout*



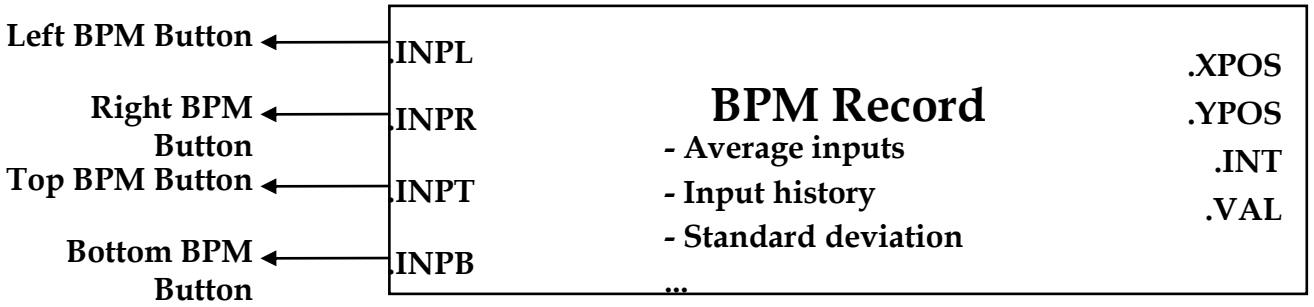

- If no CA monitor exists on the `sub` record (i.e. the operator logs out), `MLIS` will be `NULL`. The subroutine will then set the `VAL` field to `0`, causing the sequence record to process.

# Database Examples

- Quick Prototyping with Standard Records




- Custom Record Definition



# RECORDS

Summary

# Which record is right for …

- There are different ways to do things, but there are also some guidelines.
- “operator entered” soft parameters
  - `ao` has DRVH, DRVL, OROC, closed loop
  - `mbbo` provides enumerated options which can be converted to constants (`DTYP = Raw Soft Channel`)
  - Normally one does not use input records for this purpose
- Multiple output actions
  - `seq` record can have a different data source for each output link
  - `dfanout` record “fans out” a single source to multiple links
- Different output actions based on an operator selection
  - `calcout` records that conditionally process sequence records
  - `mbbo` (`DTYP = Raw Soft Channel`) forward linked to a single sequence record in “masked” mode. Mask is provided in MBBO for each state.

# Creating Database Files

- Since the database file is a simple ASCII file, it can be generated by numerous applications… as long as the syntax is correct.
  - Text editor
  - Script
  - Relational Database Tool
  - EPICS-aware Database Configuration Tools:
    - VDCT
- An EPICS-aware tool will read the `.dbd` file (library provided) and provide menu selections of enumerated fields. It may also detect database errors prior to the boot process
- A graphical tool can be helpful for complex databases.

COSYLAB

# Macro Substitutions

- EPICS features simple string substitution macros
  - `$(macro)` can be used in `.db` files
  - This allows `db` files to function as templates (e.g. use the same db file for all vacuum sectors, just with different names (and possibly other parameters))
- Database with `$(macro)` cannot be loaded – all macros need to be expanded
- This can be done in `st.cmd` (as in Exercise 1) or by means of a separate substitutions file
  - Creating a new EPICS application using example template will provide an example substitution file
- For more complex macro handling, there is an EPICS extension called msi.

# Defining the Database

- How does an IOC know what record types and device support options are available?
  - Record types, device support options, enumerated menus, and other configuration options are defined in “database definition files” (`.dbd`)
  - During the IOC booting process, one or more `.dbd` files are loaded
  - `.dbd` files are created on the workstation to include the desired information for that IOC.
- How does an IOC know about record instances (the user’s database) ?
  - Record instances are describe in “database files” (`.db`)
  - During the IOC booting process, one or more `.db` files are loaded
  - `.db` files are created on the workstation to include the desired information for that IOC.

# Database Definition
## *File Formats*

- Typical content of a database definition file (`.dbd`)

```
menu(menuPriority) {
  choice(menuPriorityLOW,    "LOW")
  choice(menuPriorityMEDIUM, "MEDIUM")
  choice(menuPriorityHIGH,   "HIGH")
}
menu(menuScan) {
  choice(menuScanPassive,   "Passive")
  choice(menuScanEvent,     "Event")
  choice(menuScanI_O_Intr,  "I/O Intr")
  choice(menuScan10_second, "10 second")
  choice(menuScan5_second,  "5 second")
  choice(menuScan2_second,  "2 second")
  choice(menuScan1_second,  "1 second")
  choice(menuScan_5_second, ".5 second")
  choice(menuScan_2_second, ".2 second")
  choice(menuScan_1_second, ".1 second")
}
```

```
device(ai,CONSTANT,devAiSoftRaw,
            "Raw Soft Channel")
device(ai,BITBUS_IO,devAiIObug,
            "Bitbus Device")
device(ao,CONSTANT,devAoSoftRaw,
            "Raw Soft Channel")
device(ao,VME_IO,devAoAt5Vxi,
            "VXI-AT5-AO")
device(bi,VME_IO,devBiAvme9440,
            "AVME9440 I")
device(bi,AB_IO,devBiAb,
            "AB-Binary Input")
driver(drvVxi)
driver(drvMxi)
driver(drvGpib)
driver(drvBitBus)
```

# Database Definition
## *File Formats*

- Typical content of database definition file (`.dbd`):

```
recordtype(ai)
{
  include "dbCommon.dbd"
  field(VAL,DBF_DOUBLE)
  {
    prompt("Current EGU Value")
    promptgroup(GUI_INPUTS)
    asl(ASL0)
    pp(TRUE)
  }
…
  field(PREC,DBF_SHORT)
  {
    prompt("Display Precision")
    promptgroup(GUI_DISPLAY)
    interest(1)
  }
}
```

```
menu(scalerCNT)
{
  choice(scalerCNT_Done,"Done")
  choice(scalerCNT_Count,"Count")
}
…
field(CNT,DBF_MENU)
{
  prompt("Count")
  special(SPC_MOD)
  menu(scalerCNT)
  pp(TRUE)
  interest(1)
}

device(ao,CONSTANT,devAoSoftRaw,
            "Raw Soft Channel")

driver(drvVxi)
```

# Database File Formats

- A typical database file (`.db`)

```
record(calc,"$(user):rampM") {
    field(CALC,"A>6.27?0:A+.1")
    field(SCAN,"1 second")
    field(INPA,"$(user):rampM.VAL NPP NMS")
}
record(calc,"$(user):cathodeTempM") {
    field(DESC,"Measured Temp")
    field(SCAN,"1 second")
    field(CALC,"C+(A*7)+(SIN(B)*3.5)")
    field(INPA,"$(user):cathodeCurrentC.OVAL NPP NMS")
    field(INPB,"$(user):rampM.VAL NPP NMS")
    field(INPC,"70")
    field(EGU,"degF")
    field(PREC,"1")
    field(HOPR,"200")
    field(LOPR,"")
    field(HIHI,"180")
    field(LOLO,"130")
    field(HIGH,"160")
    field(LOW,"140")
    field(HHSV,"MAJOR")
    field(HSV,"MINOR")
    field(LLSV,"MAJOR")
    field(LSV,"MINOR")
}
```

# Loading Database Files into the IOC

```
dbLoadDatabase("../../dbd/linacApp.dbd")

dbLoadRecords("../../db/xxLinacSim.db","user=studnt1")

iocInit           /* starts ioc software */
```

- Part of a typical startup script (`st.cmd`)
- One or more database definition files (`.dbd`) must be loaded first.
- Any record type specified in the database files must have been defined in the definition file
- Macros (variables) within the database files (e.g. `$(user)`) can be specified at boot time. This allows the same database to be loaded with different names or channel assignments.

*Based on a presentation by :*
*Andrew Johnson (APS)*

COSYLAB

# Getting Started with EPICS

## SNL – State Notation Language

# Outline

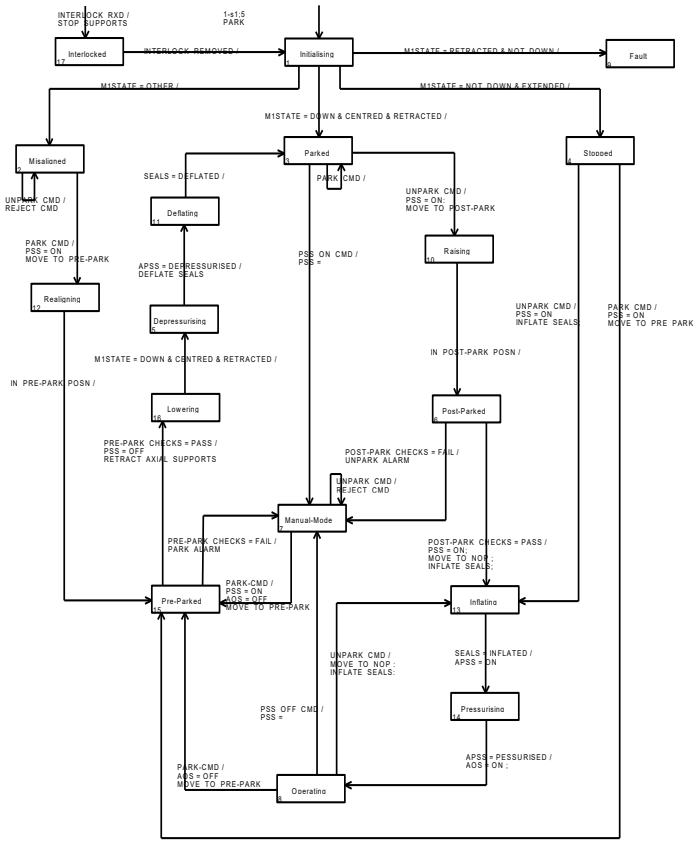
- What is State Notation Language (SNL)
- When to use it
- Where it fits in the EPICS toolkit
- Components of a state notation program
- Some notes on the Sequencer runtime
- Building, running and debugging a state notation program
- Additional Features
- This talk does not cover all the features of SNL and the sequencer. Consult the manual for more information:
  - https://github.com/epics-modules/sequencer/

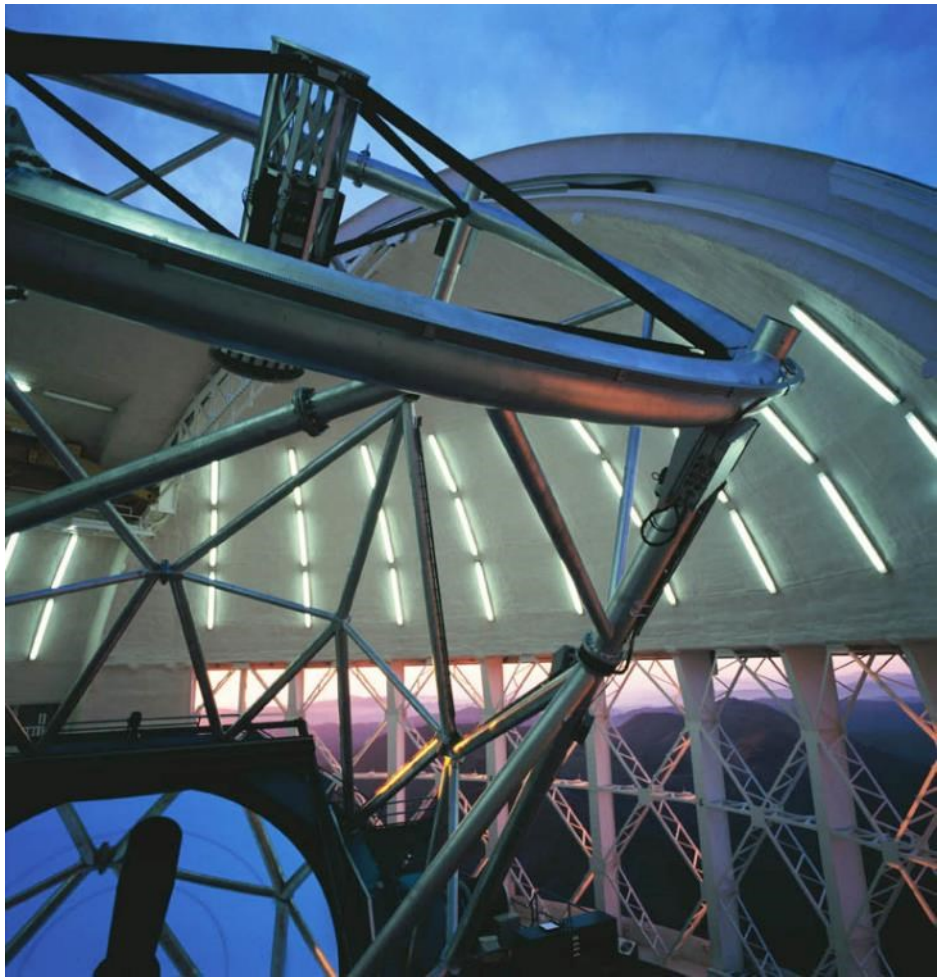
# SNL and the Sequencer

- The sequencer runs programs written in State Notation Language (SNL)
- SNL is a ‘C’ like language to facilitate programming of sequential operations
- Fast execution - compiled code
- Programming interface to extend EPICS in the real-time environment
- Common uses
  - Provide automated start-up sequences like vacuum or RF where subsystems need coordination
  - Provide fault recovery or transition to a safe state
  - Provide automatic calibration of equipment

# When to use the sequencer

- For sequencing complex events
  - e.g. parking and unparking a telescope mirror

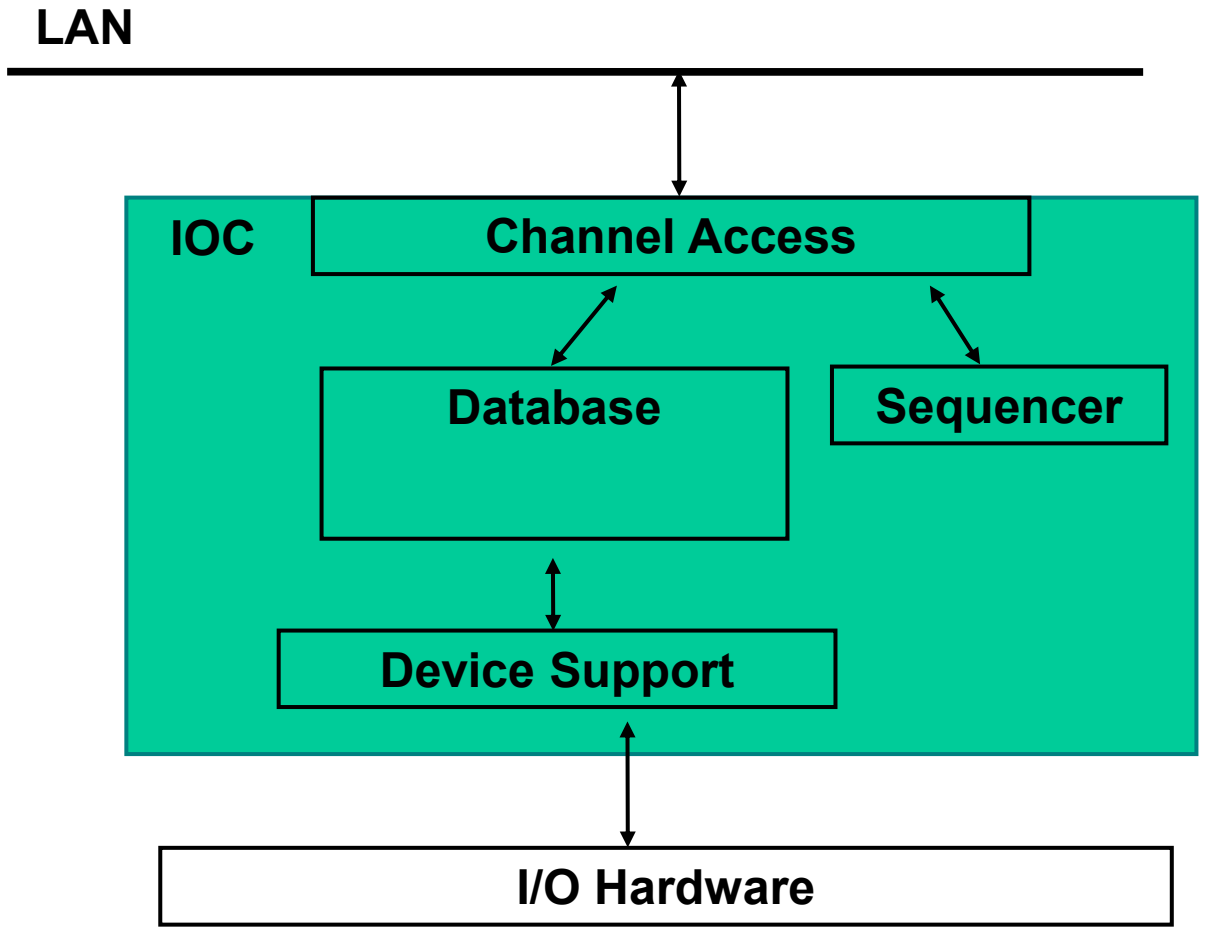


Photograph courtesy of the Gemini Telescopes project

# Where's the Sequencer?

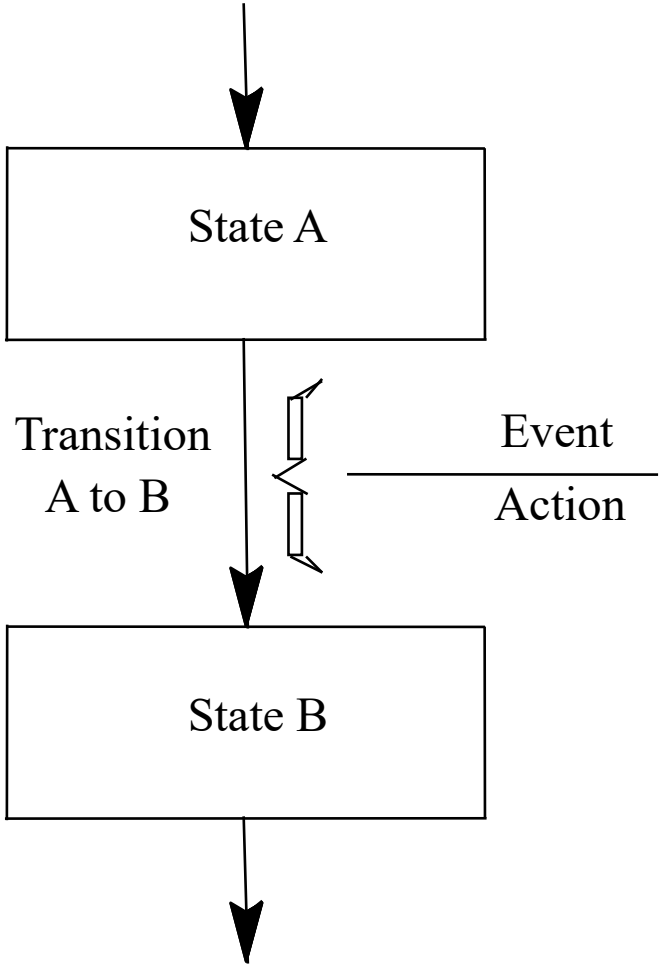
- The major software components of an IOC (IOC Core)



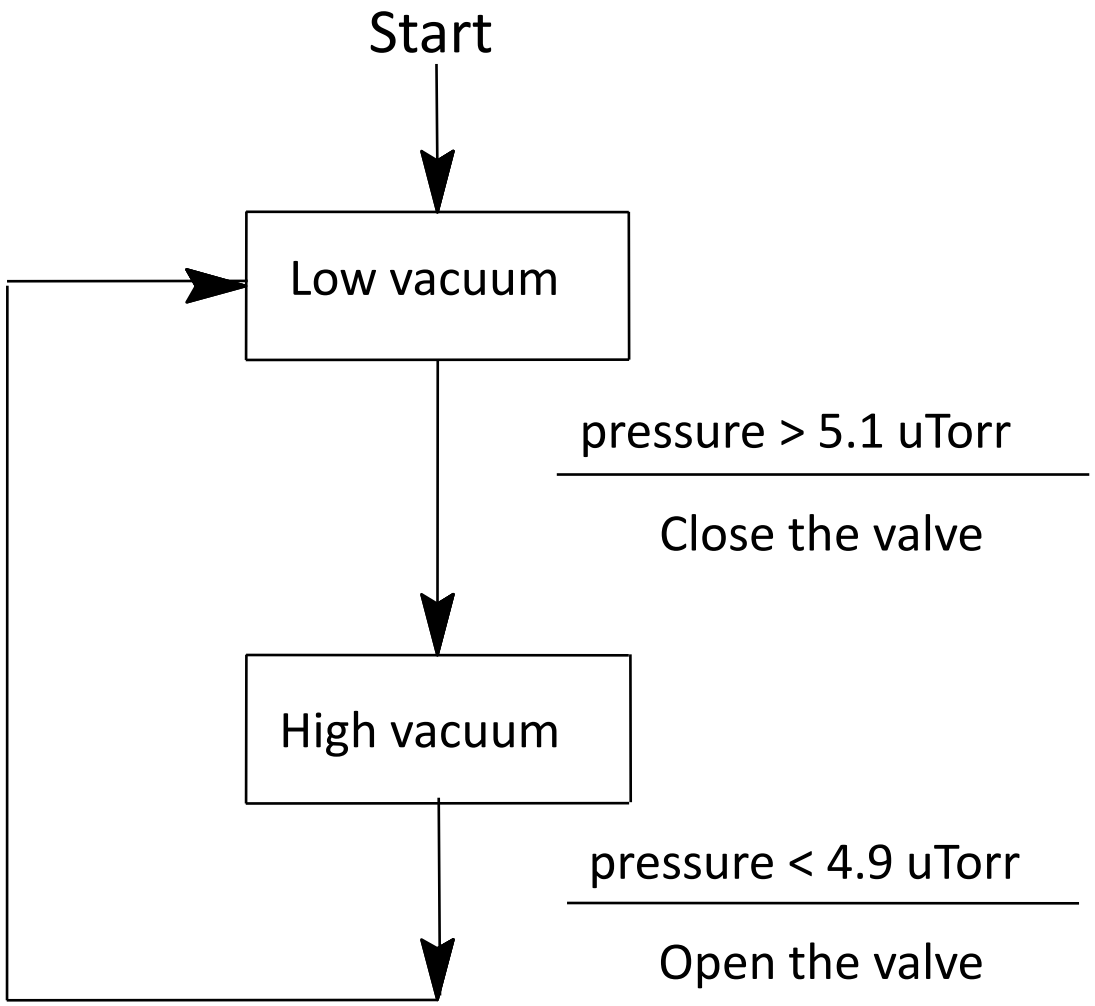
# The Best Place for the Sequencer

- Sequencer can run either on an IOC or as a standalone program on a workstation
- Traditionally, sequencers run on the IOC
- Locating them within the IOC they control makes them easier to manage
- Running them on a workstation can make testing and debugging easier
- On a workstation, SNL provides an easy way to write simple CA client programs

# SNL implements State Machines



# SM Example


Start
Low vacuum
pressure > 5.1 uTorr
Close the valve
High vacuum
pressure < 4.9 uTorr
Open the valve

# Some Definitions

- SNL : State Notation Language
- SNC : State Notation Compiler
- Sequencer : The tool that executes the compiled SNL code
- Program : A complete SNL application consisting of declarations and one or more state sets
- State Set : A set of states that make a complete finite state machine
- State : A particular mode of the state set in which it remains until one of its transition conditions is evaluated to be `TRUE`

# SNL: General Structure and Syntax

```
program program_name
declarations

ss state_set_name {

  state state_name {
    entry {
      entry action statements
    }

    when (event) {
      action statements
    } state next_state_name

    when (event) {
…
    } state next_state_name

    exit{
      exit action statements
    }
  }

  state state_name {
…
  }
}
```

| | |
|---|---|
| `program` *name* | A program may contain multiple state sets. The program name is used as a handle to the sequencer manager for state programs. |
| `ss` *name* `{` | A state set becomes a task. |
| `state` *name* `{` | A state is an area where the task waits for events. The related task waits until one of the events occurs and then checks to see which it should execute. The first state defined in a state set is the initial state. |
| `option` *flag*`;` | A state specific option |
| `when (`*event*`) {` | Defines the events for which this state waits. |
| `} state` *next* | Specifies the following state after the actions complete. |
| `entry {`*actions*`}` | Actions to do on entry to this state from another state. With `option -e;` these actions will trigger even if it re-enters from the same state. |
| `exit {`*actions*`}` | Actions to do before exiting this state to another state. With `option -x;` these actions will trigger even if it exits to the same state. |

# Declarations – Variables

- Appear before a state set and have a scope of the entire program.
- Scalar variables

```
int             var_name;
short           var_name;
long            var_name;
char            var_name;
float           var_name;
double          var_name;
string          var_name;    /* 40 characters */
```

- Array variables: 1 or 2 dimensions, no strings

```
int             var_name[num_elements];
short           var_name[num_elements];
long            var_name[num_elements];
char            var_name[num_elements];
float           var_name[num_elements];
double          var_name[num_elements];
```

# Declarations – Assignments

- Assignment connects a variable to a channel access PV name

```
float  pressure;
assign pressure to “CouplerPressureRB1”;
double pressures[3];
assign pressures to {“CouplerPressureRB1”, ”CouplerPressureRB2”, ” CouplerPressureRB3”};
```

- To use these channels in when clauses, they must be monitored

```
monitor pressure;
monitor pressures;
```

- Use preprocessor macros to aid readability:

```
#define varMon(t,n,c) t n; assign n to c; monitor n;
varMon(float, pressure, “CouplerPressureRB1”)
```

# Declarations – Event Flags

- Event flags are used to communicate between state sets, or to receive explicit event notifications from Channel Access
- Declare like this:

```
evflag      event_flag_name;
```

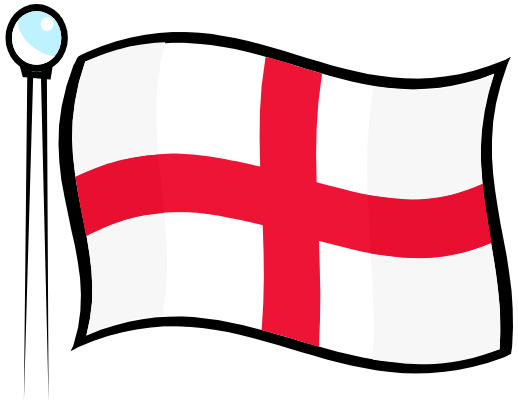
- An event flag can be synchronized with a monitored variable

```
sync var_name event_flag_name;
```

- The flag will then be set when a monitor notification arrives

```
evflag      flag_monitor;

sync pressure flag_monitor;
```



# Events

- Event: The condition on which actions associated with a when are run and a state transition is made.
- Possible events:
  - Change in value of a variable that is being monitored:

    ```
    when (achan < 10.0)
    ```

  - A timer event (not a task delay!):

    ```
    when (delay(1.5))
    ```

    - The delay time is in seconds.
    - A delay is normally reset whenever the state containing it is exited.
    - Use the state specific `option -t;` to stop it from being reset when transitioning to the same state.

# Possible Events (continued)

- The state of an event flag:

```
when (efTestAndClear(myflag))
```

```
when (efTest(myflag))
```

  - `efTest()` does not clear the flag. `efClear()` must be called sometime later to avoid an infinite loop.
  - If the flag is synced to a monitored variable, it will be set when the channel sends a value update
  - The event flag can also be set by any state set in the program using `efSet(event_flag_name)`

- Any change in the channel access connection status:

```
when (pvConnectCount() < pvChannelCount())
```

```
when (pvConnected(mychan))
```

# Action Statements

- Built-in action function, e.g. :

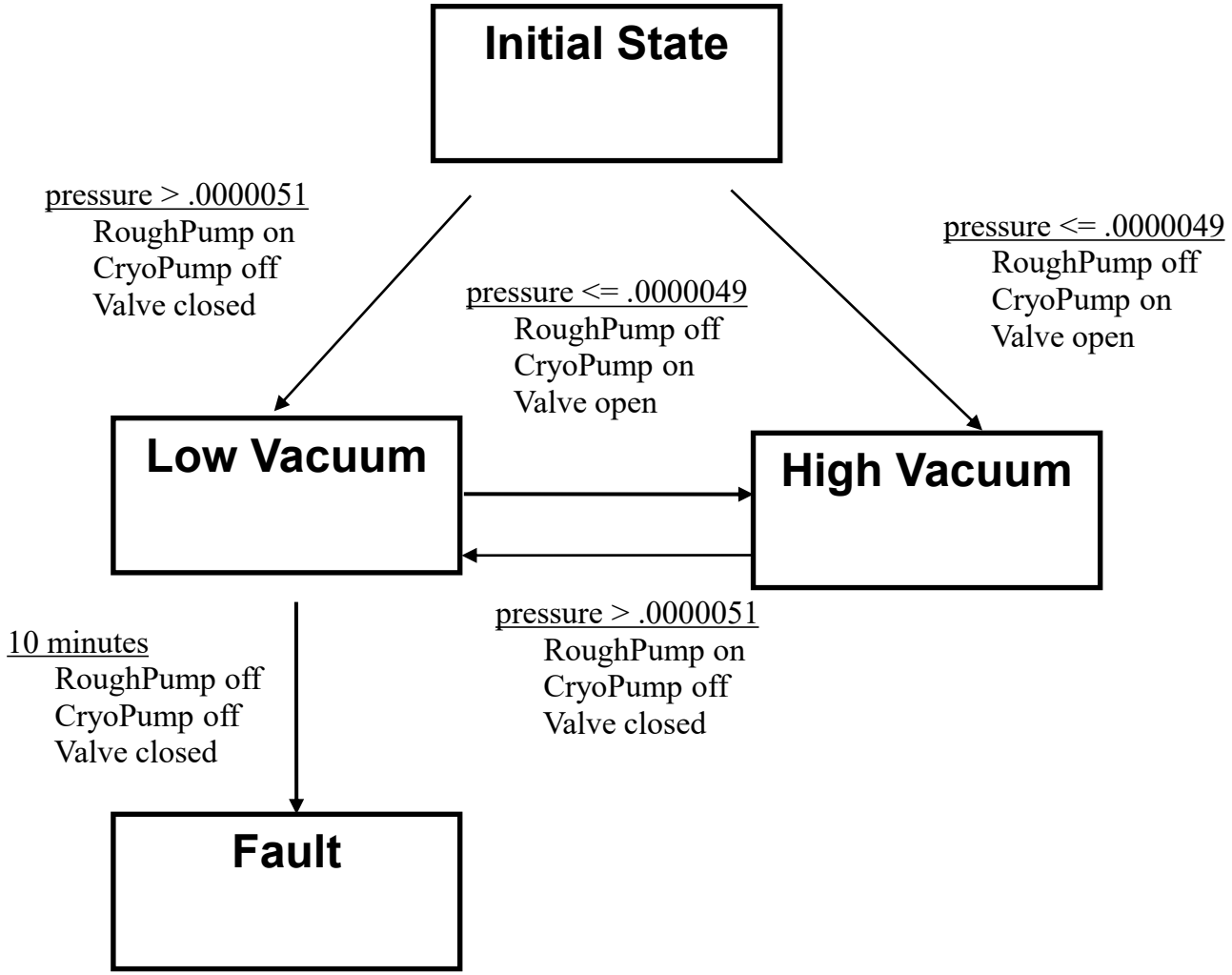
```
pvPut(var_name);
pvGet(var_name);
efSet(event_flag_name);
efClear(event_flag_name);
```


- Almost any valid C statement
  - `switch()` is not implemented and code using it must be escaped.
  - `%%` *escapes one line of C code*
  - `%{`

    *escape any number of lines of C code*

    `}%`

# Example – State Definitions and Transitions



# Example – Declarations and State Transitions (actions omitted)

```
double  pressure;
assign  pressure to
        "Tank1Coupler1PressureRB";
monitor pressure;

short     RoughPump;
assign  RoughPump to
        "Tank1Coupler1RoughPump";

short     CryoPump;
assign  CryoPump to "Tank1Coupler1CryoPump";

short     Valve;
assign  Valve to
        "Tank1Coupler1IsolationValve";

string  CurrentState;
assign  CurrentState to
        "Tank1Coupler1VacuumState";
```

```
program vacuum_control
ss coupler_control
{
  state init{
    when (pressure > .0000051){
    } state low_vacuum
    when (pressure <= .0000049){
    } state high_vacuum
  }
  state high_vacuum{
    when (pressure > .0000051){
    } state low_vacuum
  }
  state low_vacuum{
    when (pressure <= .0000049){
    } state high_vacuum
    when (delay(600.0)){
    } state fault
  }
  state fault {
  }
}
```

# Example – `init` state and `low_vacuum` state

```
state init {
  entry {
    strcpy(CurrentState,"Init");
    pvPut(CurrentState);
  }
  when (pressure > .0000051){
    RoughPump = 1;
    pvPut(RoughPump);
    CryoPump = 0;
    pvPut(CryoPump);
    Valve = 0;
    pvPut(Valve);
  } state low_vacuum
  when (pressure <= .0000049){
    RoughPump = 0;
    pvPut(RoughPump);
    CryoPump = 1;
    pvPut(CryoPump);
    Valve = 1;
    pvPut(Valve);
  } state high_vacuum
}
```

```
state low_vacuum{
  entry {
  strcpy(CurrentState,"Low Vacuum");
  pvPut(CurrentState);
  }
  when (pressure <= .0000049){
    RoughPump = 0;
    pvPut(RoughPump);
    CryoPump = 1;
    pvPut(CryoPump);
    Valve = 1;
    pvPut(Valve);
  } state high_vacuum
  when (delay(600.0)){
  } state fault
}
```

# Example – `high_vacuum` state and `fault` state

```
state high_vacuum{
  entry {
  strcpy(CurrentState,”High Vacuum”);
  pvPut(CurrentState);
  }
  when (pressure > .0000051){
    RoughPump = 1;
    pvPut(RoughPump);
    CryoPump = 0;
    pvPut(CryoPump);
    Valve = 0;
    pvPut(Valve);
  } state low_vacuum
}
```

```
state fault{
  entry{
    strcpy(CurrentState,”Vacuum Fault”);
    pvPut(CurrentState);
  }
}
```

# Building an SNL program

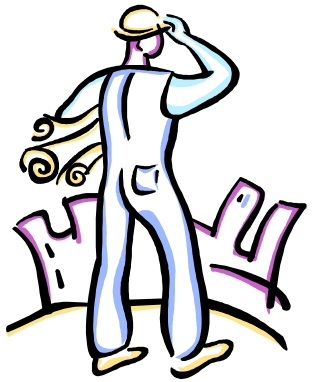
- Use editor to build the source file. File name must end with `.st` or `.stt`, e.g. `example.st`
- `make` automates these steps:
  - Runs the C preprocessor on `.st` files, but not on `.stt` files.
  - Compiles the state program with SNC to produce C code:

    ```
    snc  example.st  ->  example.c
    ```

  - Compiles the resulting C code with the C compiler:

    ```
    cc example.c -> example.o
    ```

  - The object file `example.o` becomes part of the application library, ready to be linked into an IOC binary.
  - The executable file `example` can be created instead.



# Run Time Sequencer

- The sequencer executes the state program
- It is implemented as an event-driven application; no polling is needed
- Each state set becomes an operating system thread
- The sequencer manages connections to database channels through Channel Access
- It provides support for channel access get, put, and monitor operations
- It supports asynchronous execution of delays, event flag, pv put and pv get functions
- Only one copy of the sequencer code is required to run multiple programs

```
options +r; // use macro
```

- Commands are provided to display information about the state programs currently executing

# Executing a State Program

- From an IOC console

```
seq vacuum_control
```

- To stop the program

```
seqStop vacuum_control
```

# Debugging

- Use the sequencer's query commands:

`seqShow`

  - displays information on all running state programs

`seqShow vacuum_control`

  - displays detailed information on program

`seqChanShow vacuum_control`

  - displays information on all channels

`seqChanShow vacuum_control,"-"`

  - displays information on all disconnected channels

# Debugging (continued)

- Use `printf` functions to print to the console

```
printf("Here I am in state xyz \n");
```

- Put strings to pvs

```
sprintf(seqMsg1, "Here I am in state xyz");
pvPut(seqMsg1);
```

- Reload and restart (if running independently of IOC)

```
seqStop vacuum_control
```

  - ... edit, recompile ...
  - `seq vacuum_control`

# Debugging – seqShow

```
epics> seqShow
Program Name      Thread ID  Thread Name       SS Name
stabilizer        ede78      stabilizer        stabilizerSS1
beamTrajectory    db360      beamTrajectory    bpmTrajectorySS
autoControl       ed620      autoControl       autoCtlSS
```

# Debugging – seqShow

```
epics> seqShow stabilizer
State Program: "stabilizer"
  initial thread id = ede78
  thread priority = 50
  number of state sets = 1
  number of syncQ queues = 0
  number of channels = 3
  number of channels assigned = 3
  number of channels connected = 3
  options: async=0, debug=0, newef=1, reent=0, conn=1, main=0

  State Set: "stabilizerSS1"
  thread name = stabilizer;  thread id = 974456 = 0xede78
  First state = "init"
  Current state = "waitForEnable"
  Previous state = "init"
  Elapsed time since state was entered = 88.8 seconds
```

# Debugging – seqChanShow

```
epics> seqChanShow stabilizer
State Program: "stabilizer"
Number of channels=3

#1 of 3:
Channel name: "stabilizerC"
  Unexpanded (assigned) name: "stabilizerC"
  Variable name: "enableButton"
    address = 154120 = 0x25a08
    type = short
    count = 1
  Value = 0
  Monitor flag = 1
    Monitored
  Assigned
  Connected
  Get not completed or no get issued
  Put not completed or no put issued
  Status = 17
  Severity = 3
  Message =
  Time stamp = <undefined>
Next? ( skip count)
```

# Additional Features

- Connection management:

```
when (pvConnectCount() != pvChannelCount())
when (pvConnected(Vin))
```

- Macros:

```
assign Vout to "{unit}:OutputV";
```

  - must use the `+r` compiler options for this if more than one copy of the sequence is running on the same IOC

```
seq example, "unit=HV01"
```

- Some common SNC program options:
  - `+r` make program reentrant (default is `-r`)
  - `-c` don't wait for all channel connections (default is `+c`)
  - `+a` asynchronous pvGet() (default is `-a`)
  - `-w` don't print compiler warnings (default is `+w`)

```
program dynamic
option -c; /* don't wait for db connections */
```

# Additional Features (continued)

- Access to channel alarm status and severity:

```
pvStatus(var_name)
pvSeverity(var_name)
```

- Queued monitors save CA monitor events in a queue in the order they come in, rather than discarding older values when the program is busy

```
syncQ var_name to event_flag_name [queue_length]
pvGetQ(var_name)
```

  - removes oldest value from variables monitor queue. Remains true until queue is empty.

```
pvFreeQ(var_name)
```

# Advantages of SNL

- Can implement complicated algorithms
- Can stop, reload, restart a sequence program without rebooting
- Interact with the operator through `string` records and `mbbo` records
- C code can be embedded as part of the sequence
- All Channel Access details are taken care of
- File access can be implemented as part of the sequence

COSYLAB

# Getting Started with EPICS

Device Support

COSYLAB

# Writing Device Support – Outline

- Introduction
  - What this for?
  - What is ‘Device Support’?
  - The `.dbd` file entry
  - The driver DSET
  - Device addresses
  - Support routines
- Example
- Asynchronous processing
  - Using interrupts
  - Asynchronous input/output
  - Callbacks

# Device Integration Task

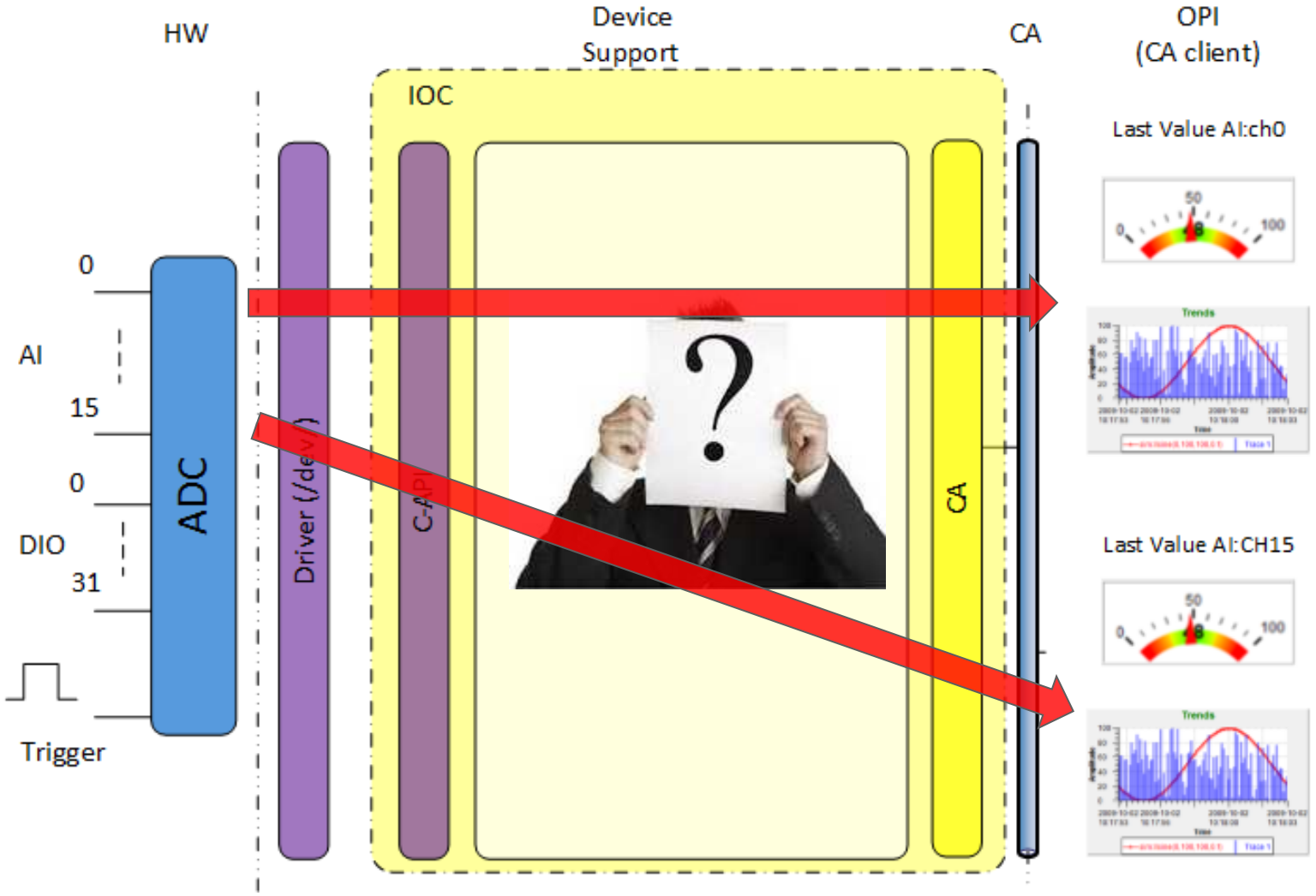
- Steps
  - Use Case collection
  - Requirements specification
  - **Design**
  - **Implementation**
  - Verification

# Writing Device Support – Scope

- An overview of the concepts associated with writing EPICS Device Support routines.
- Examples show the “stone knives and bearskins” approach.

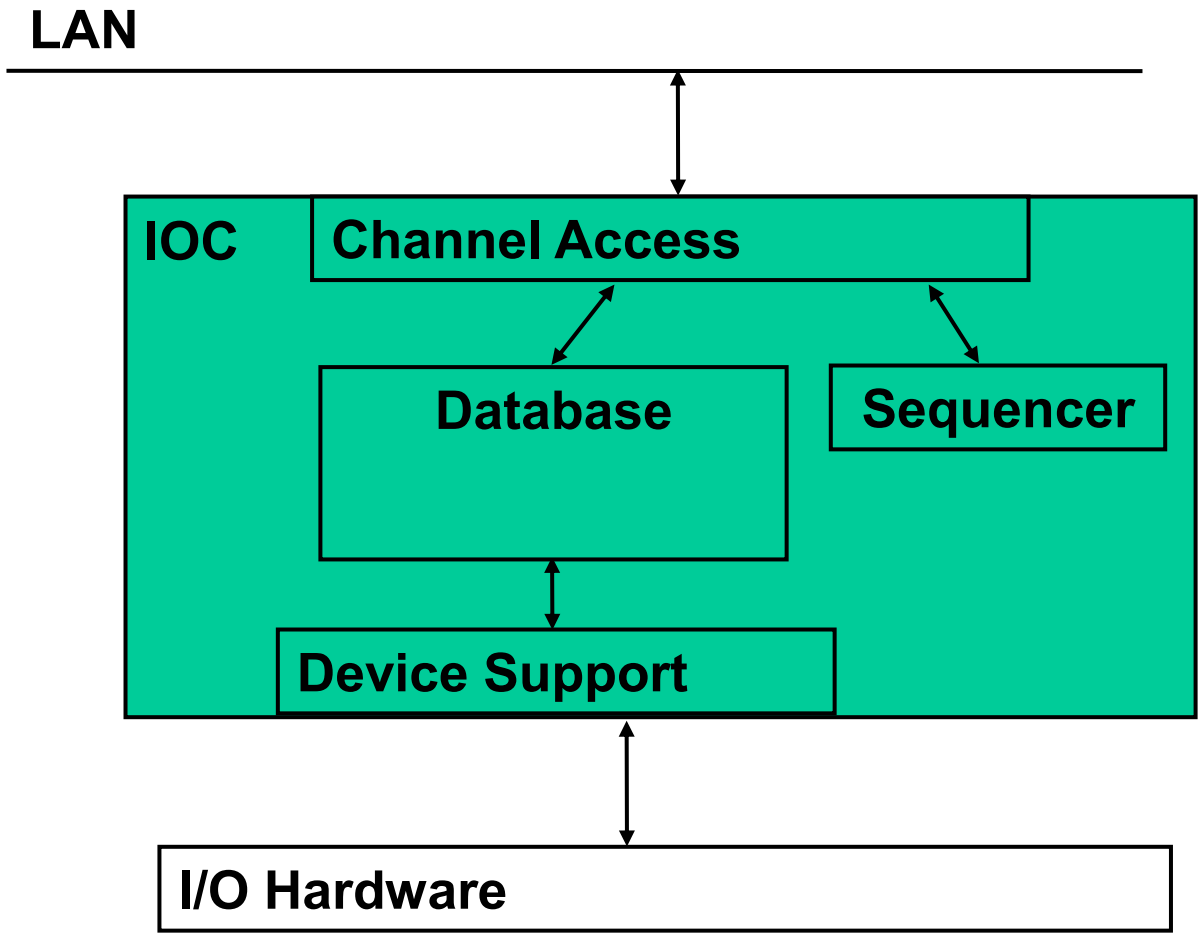
- The ASYN package provides a framework which makes writing device support much easier.
  - The concepts presented here still apply.

# Goal


HW
Device Support
IOC
CA
OPI (CA client)
Last Value AI:ch0
0
AI
15
0
DIO
31
Trigger
ADC
Driver (/dev/)
C-API
CA
Last Value AI:CH15
Trends
Trends

# Inside the IOC

- The major software components of an IOC (IOC Core)



# EPICS Databases – What are they for?

- Interface to process instrumentation
- Distribute processing
- Provide external access to all process information
- Use common, proven, objects (records) to collect, process and distribute data
- Provide a common toolkit for creating applications

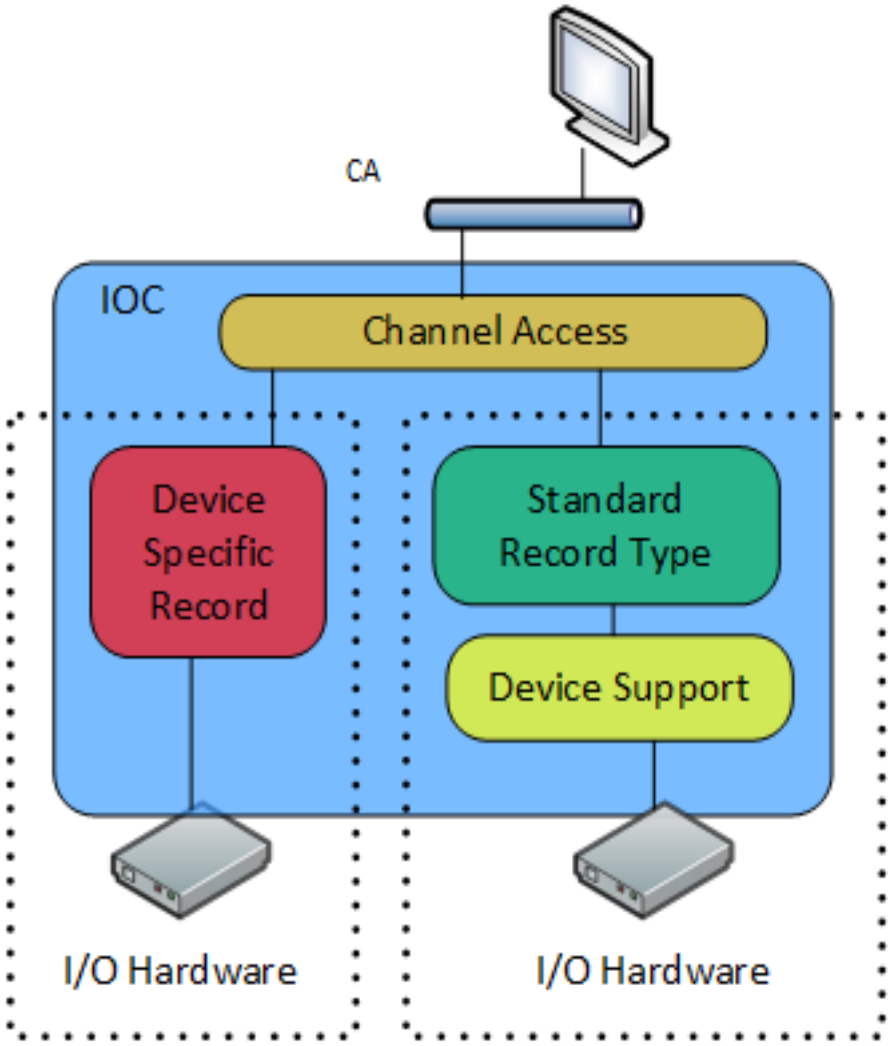
# How is a Record implemented?

- A ‘C’ structure with both data storage and pointers to record type information

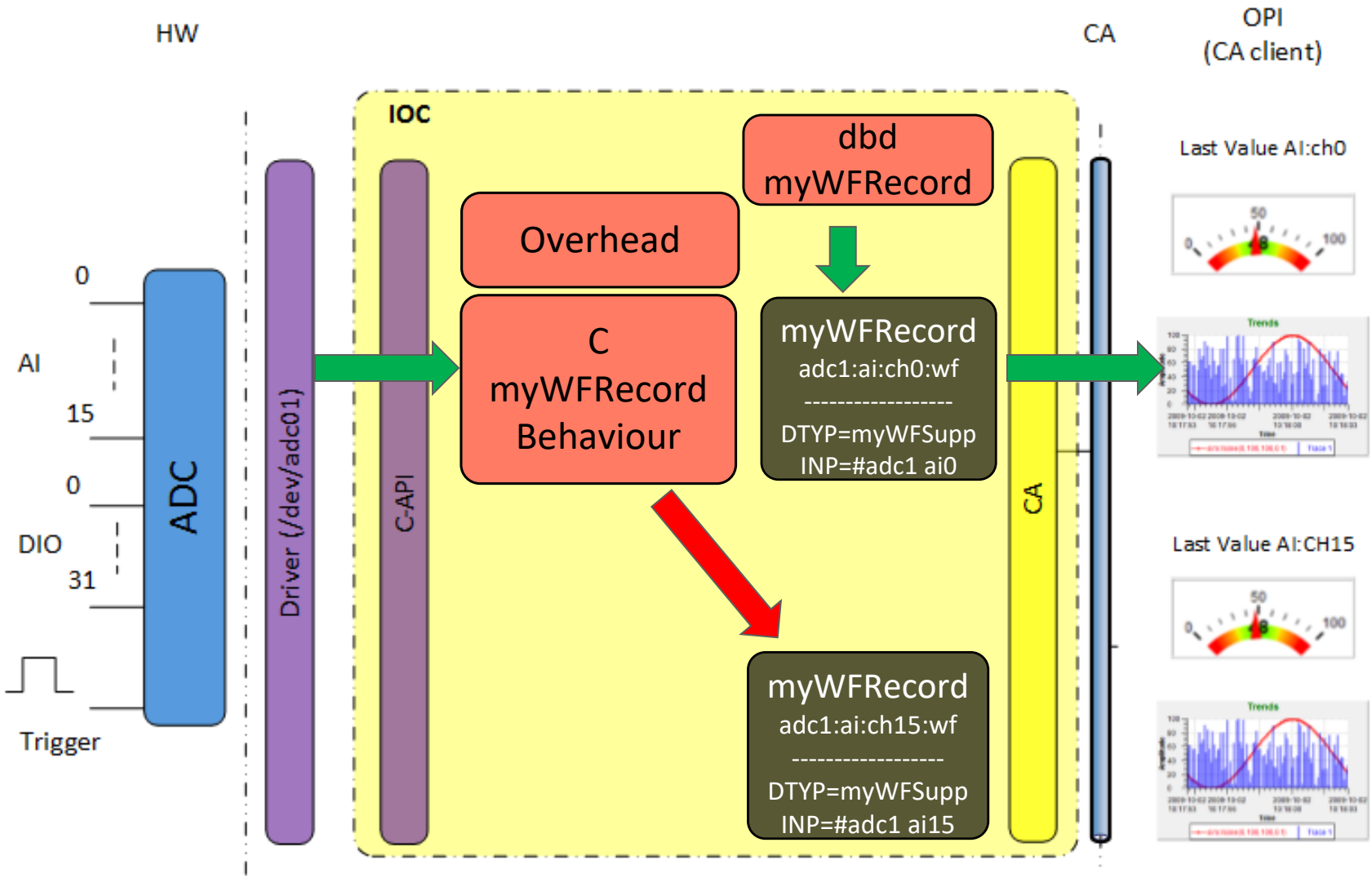
- A record definition within a database provides
  - Record name
  - The record’s type
  - Values for each design field
- A record type provides
  - Definitions of all the fields
  - Code which implements the record behavior

- New record types can be added to an application as needed
  - Check EPICS example application (`xxxRecord`)

# Device support diagram



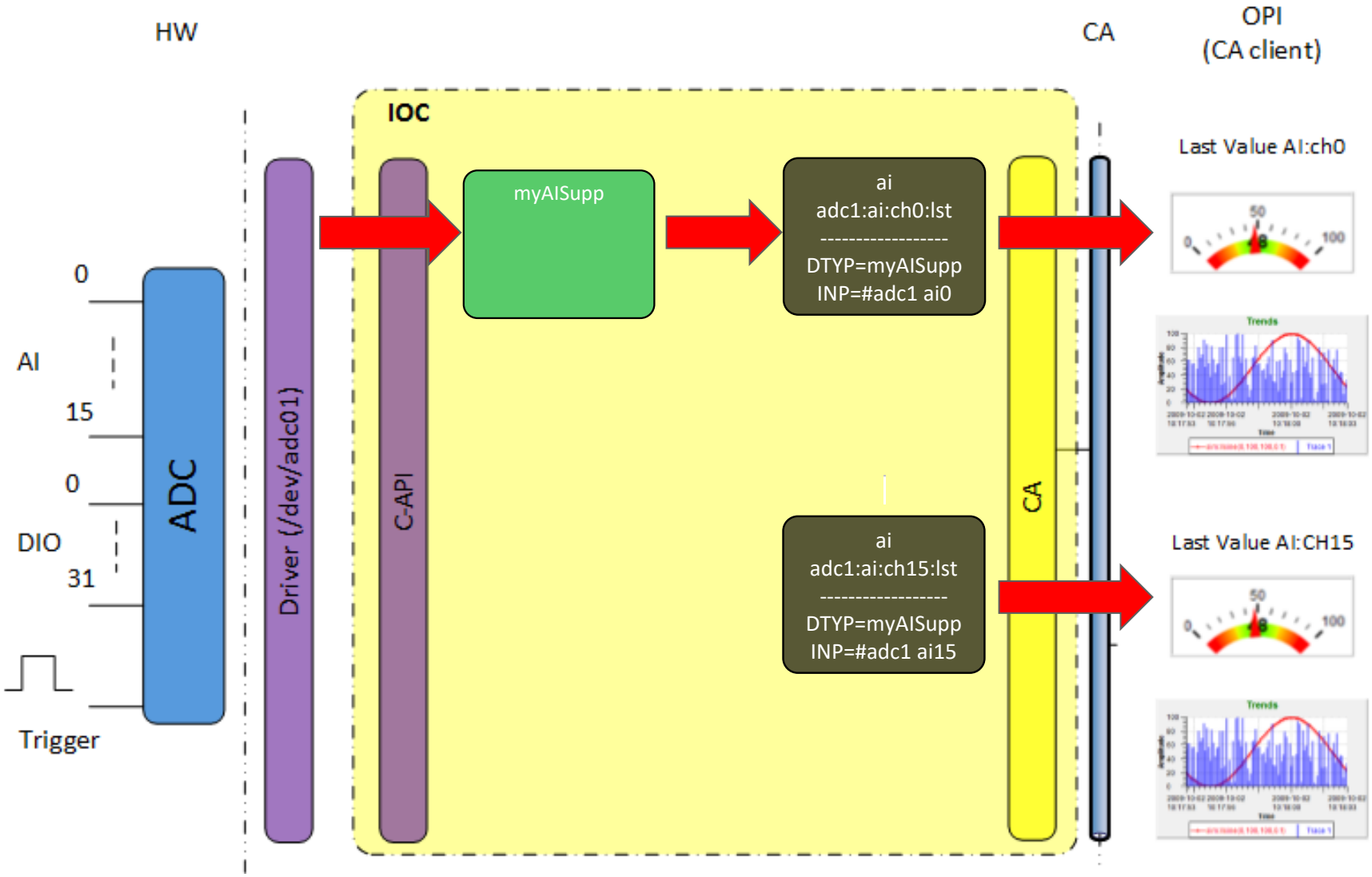
# Specific record implementation



# What is ‘Device Support’?

- Interface between record and hardware
- A set of routines for record support to call
  - The record type determines the required set of routines
  - These routines have full read/write access to any record field
- Determines synchronous/asynchronous nature of record
- Performs record I/O
  - Provides interrupt handling mechanism

# Device Support Architecture



COSYLAB

# Why use device support?

- Could instead make a different record type for each hardware interface, with fields to allow full control over the provided facilities.
- A separate device support level provides several advantages:
  - There is no needs for USERs learn a new record type for each type of device
  - Increases modularity
    - I/O hardware changes are less disruptive
    - Device support is simpler than record support
    - Hardware interface code is isolated from record API
- Custom records are available if really needed.
  - By which I mean “really, really, really needed!”
  - Existing record types are sufficient for most applications.

# Where to look for help?

https://epics.anl.gov/base/R3-16/2-docs/AppDevGuide/AppDevGuide.html

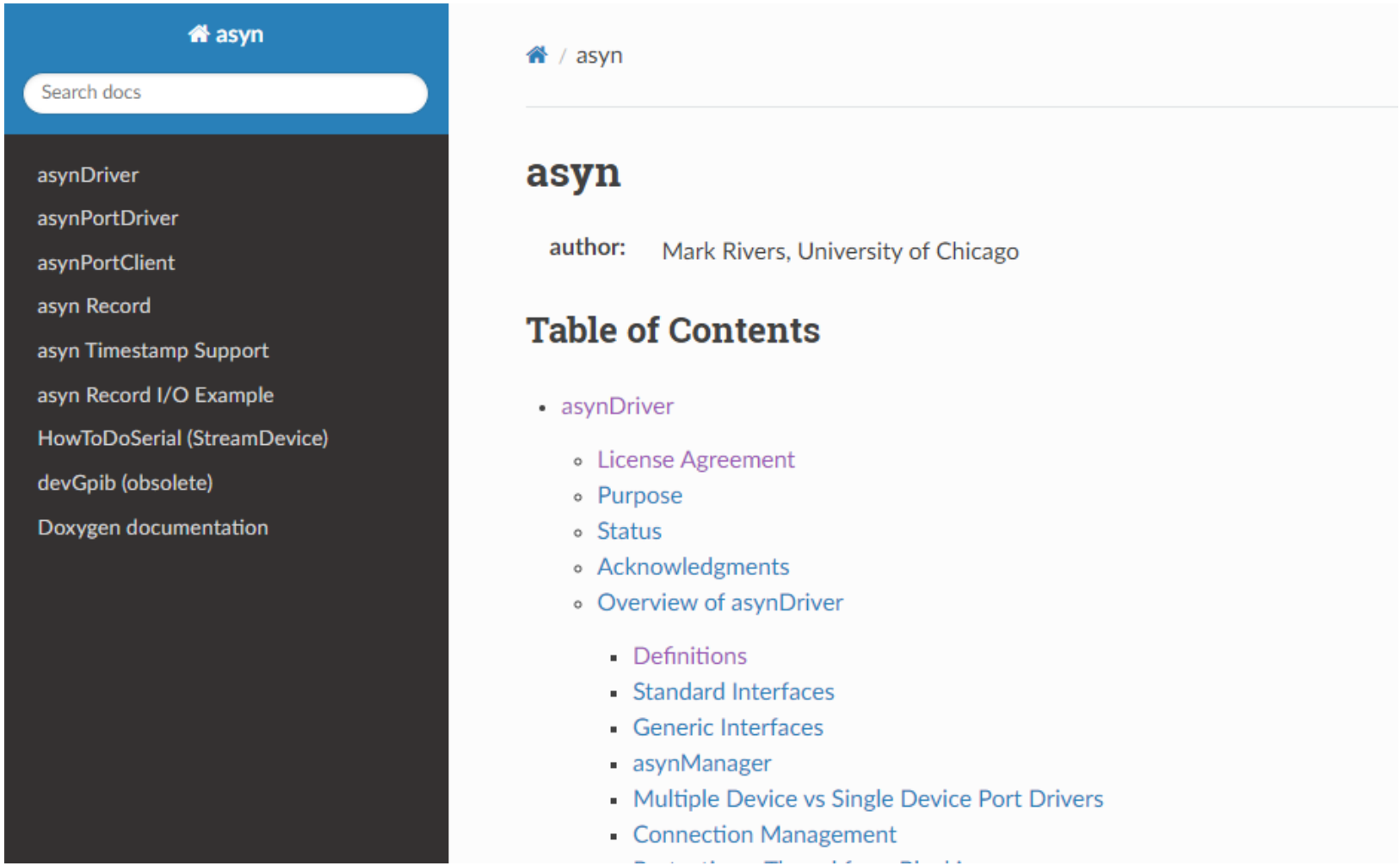
COSYLAB

# EPICS
# Advanced part

## Asyn Driver

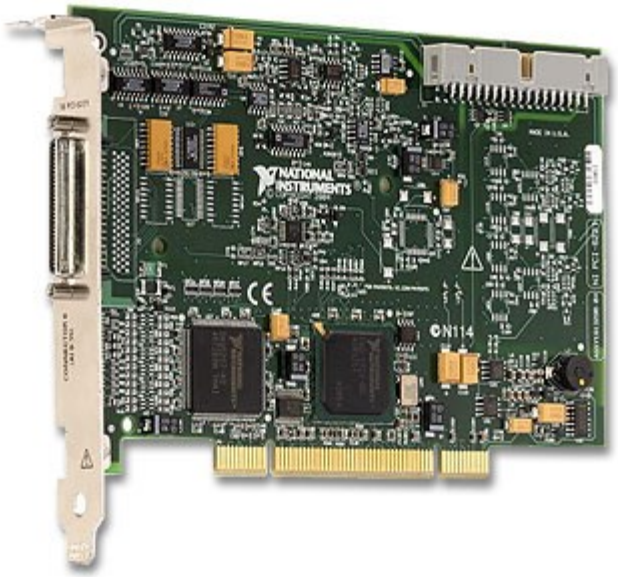
# AsynDriver on the Web

https://epics-modules.github.io/asyn/



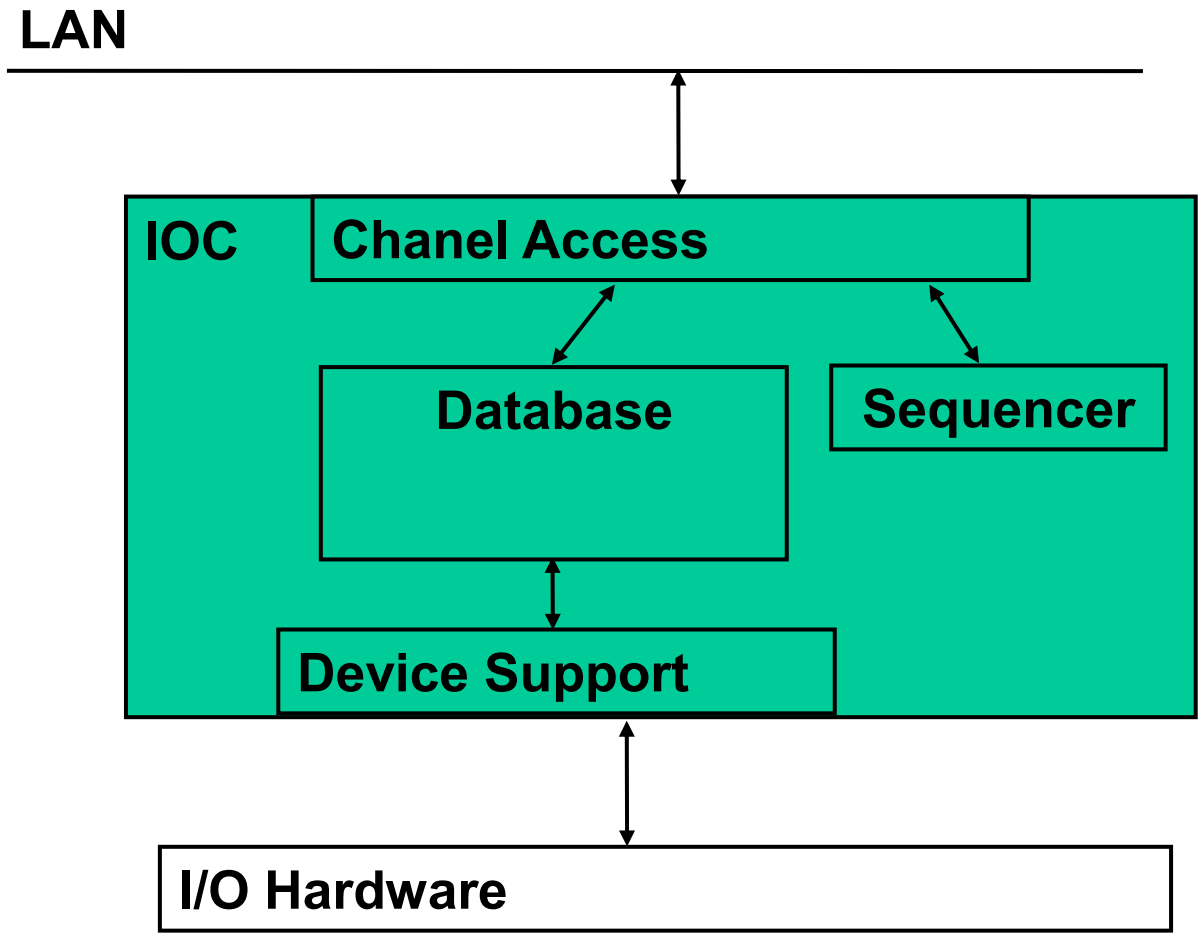
# Goal

- Card
  - Card parameters
- Channel groups
  - Channels
    - Range
    - Offset
    - Gain
    - ???
  - Triggers

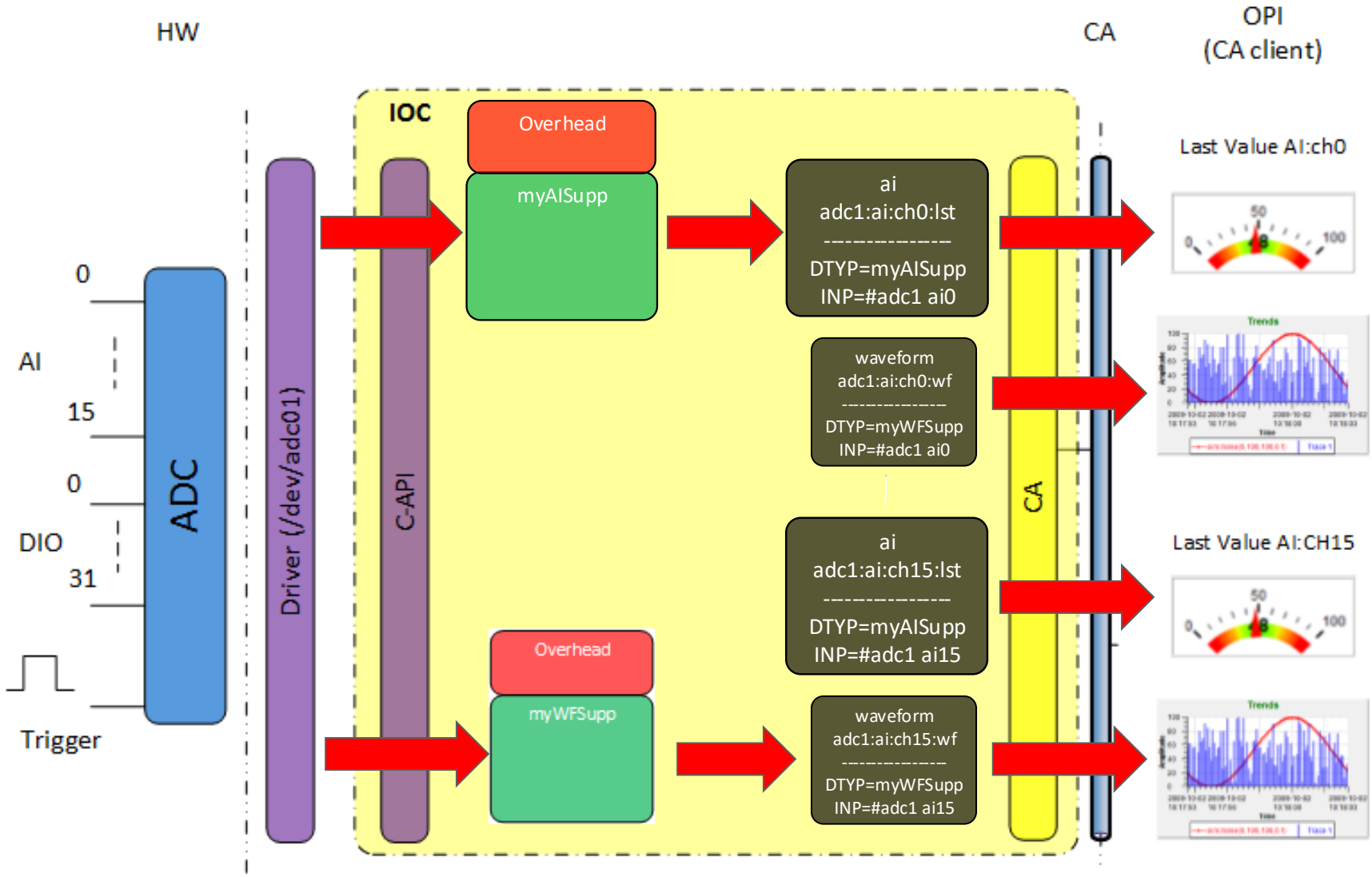


# Inside an IOC

- The major software components of an IOC (IOC Core)



# Device Support Architecture



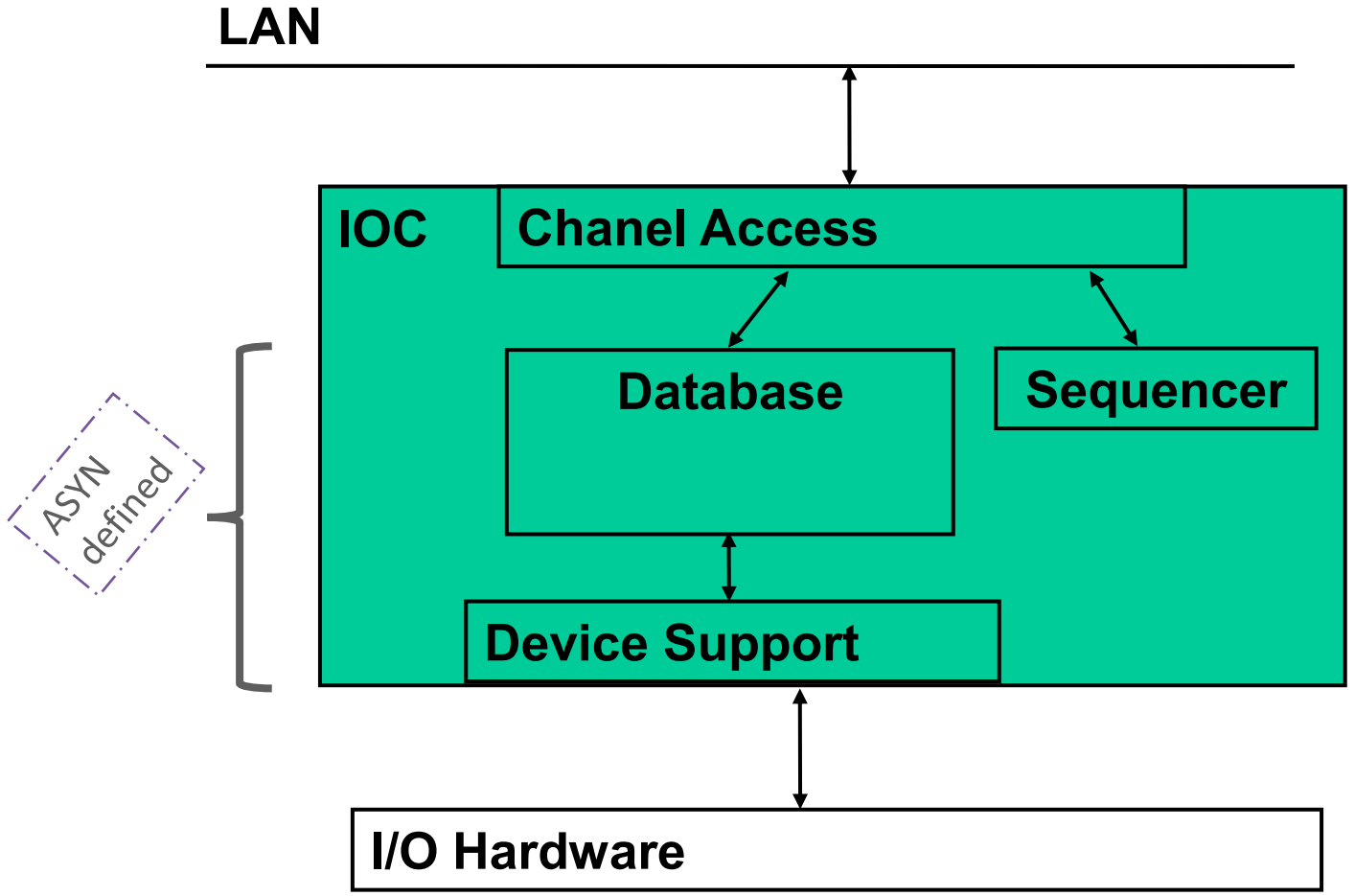
# Device Support Approach

- Device state management
- Device parameters
- Channel parameters

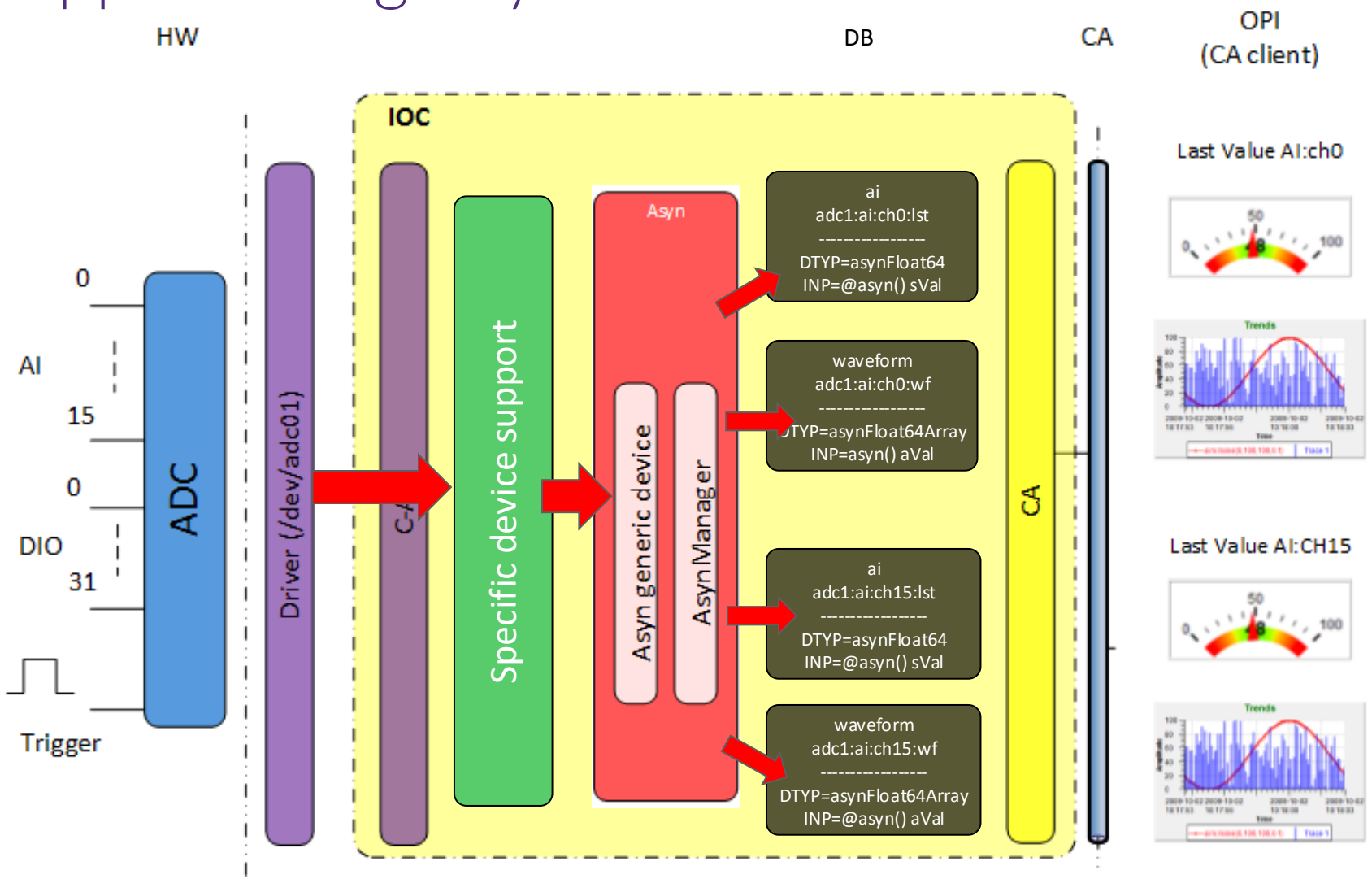
- Each required data type (interface) must be implemented
- Asynchronous processing / Interrupts

# ASYN Definition

- The major software components of an IOC (IOC Core)



# Device Support Using Asyn

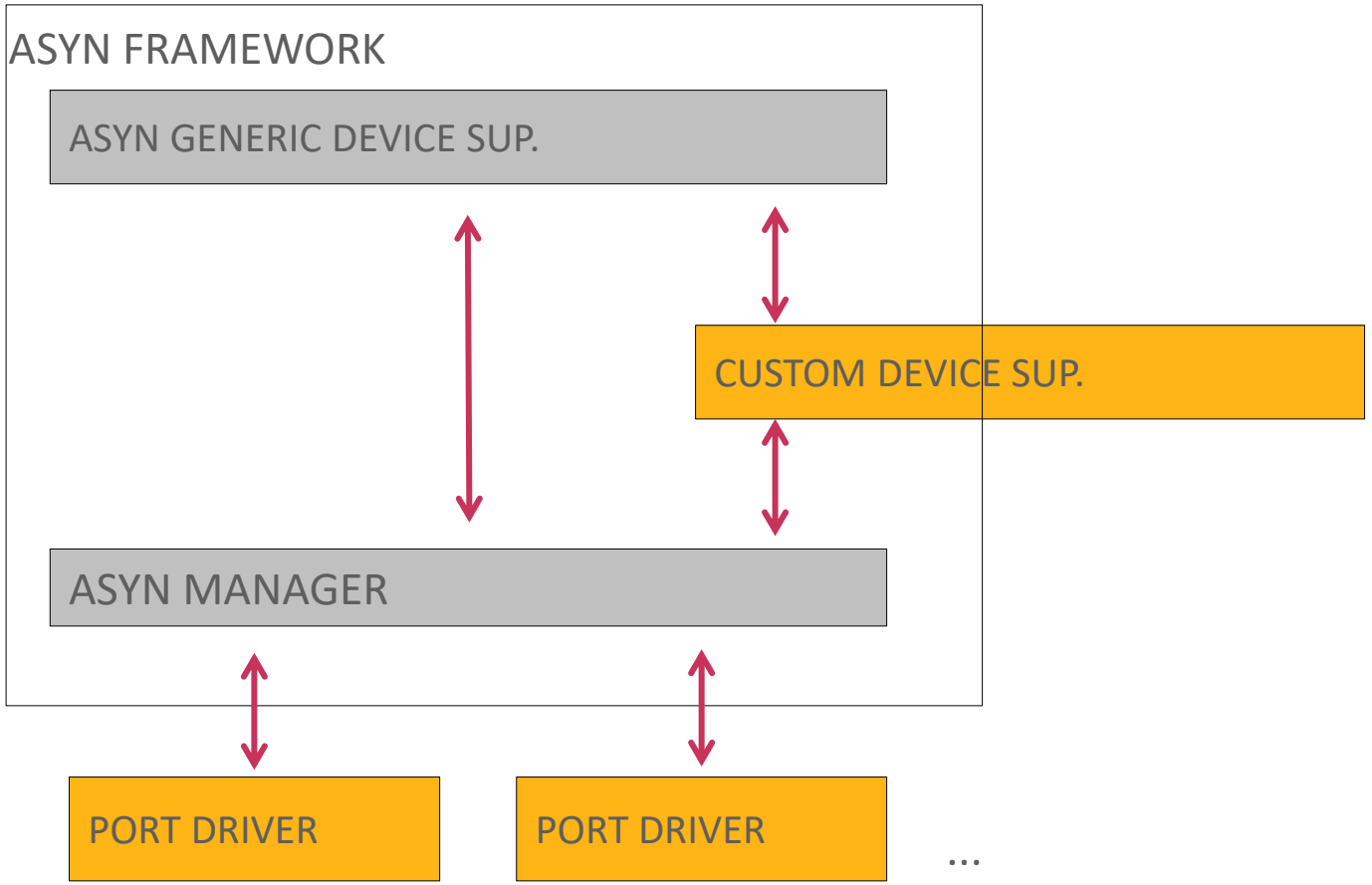


COSYLAB

# Why is this so cool?

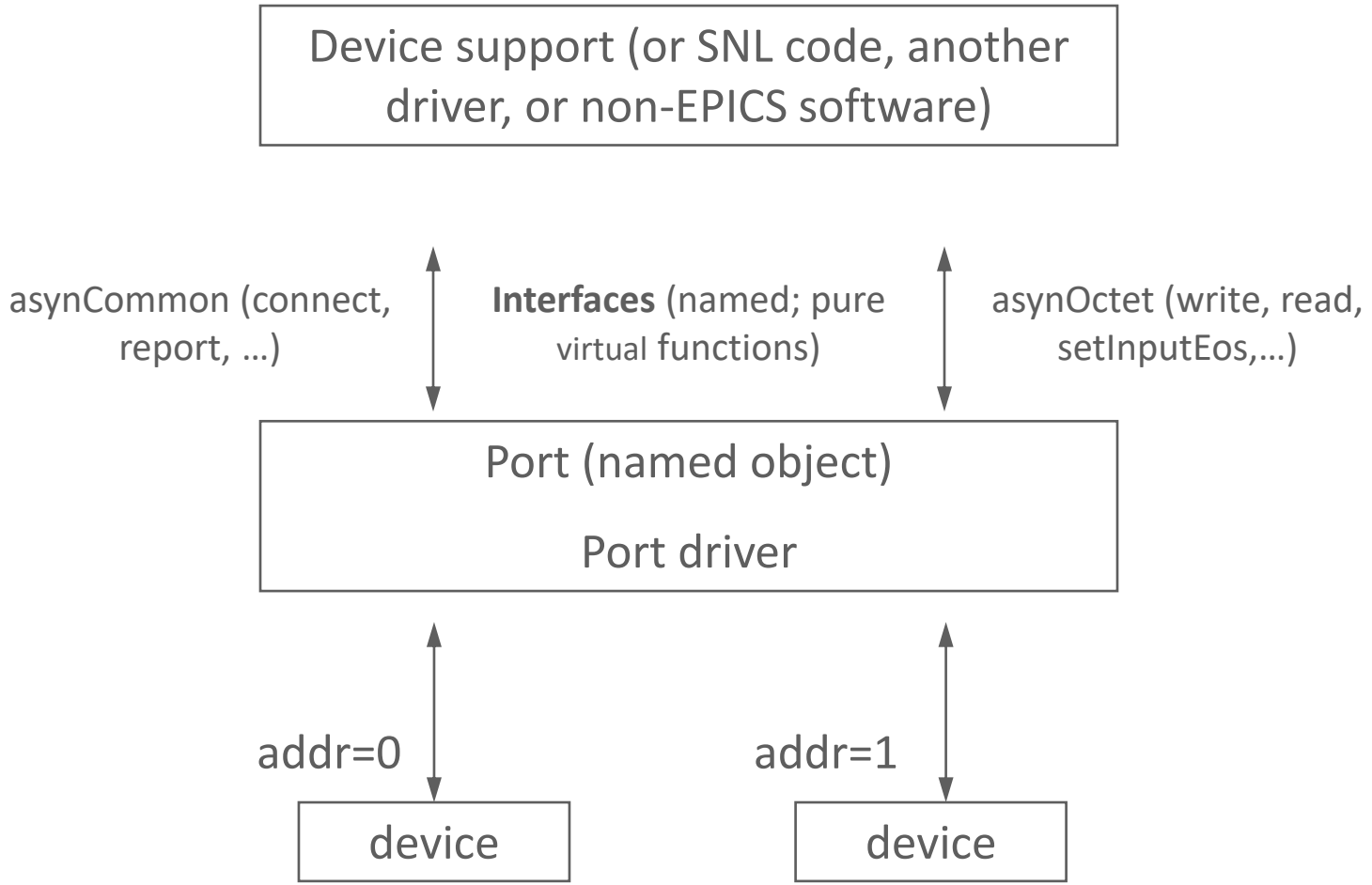
- Handles complex stuff that is common
  - Write/Read operations
  - Asynchronous processing/Interrupts
  - Multiplicity
- Simplify writing of drivers, prevents bad design
- No need for EPICS device support (most cases)
- C/C++ Interfaces

# Asyn Architecture




- Asyn manager has access to asyn port driver

# Asyn Architecture (Cont.)



# Generic Device Support

- asyn includes generic device support for many standard EPICS records and standard asyn interfaces
- Eliminates need to write device support in many cases.  New hardware can be supported by writing just a driver.
- Examples:

  `asynInt32`
    - `ao`, `ai`, `mbbo`, `mbbi`, `longout`, `longin`

  `asynInt32Average`
    - `ai`

  `asynUInt32Digital`, `asynUInt32DigitalInterrupt`
    - `bo`, `bi`, `mbbo`, `mbbi`

  `asynFloat64`
    - `ai`, `ao`

  `asynOctet`
    - `stringin`, `stringout`, `waveform`

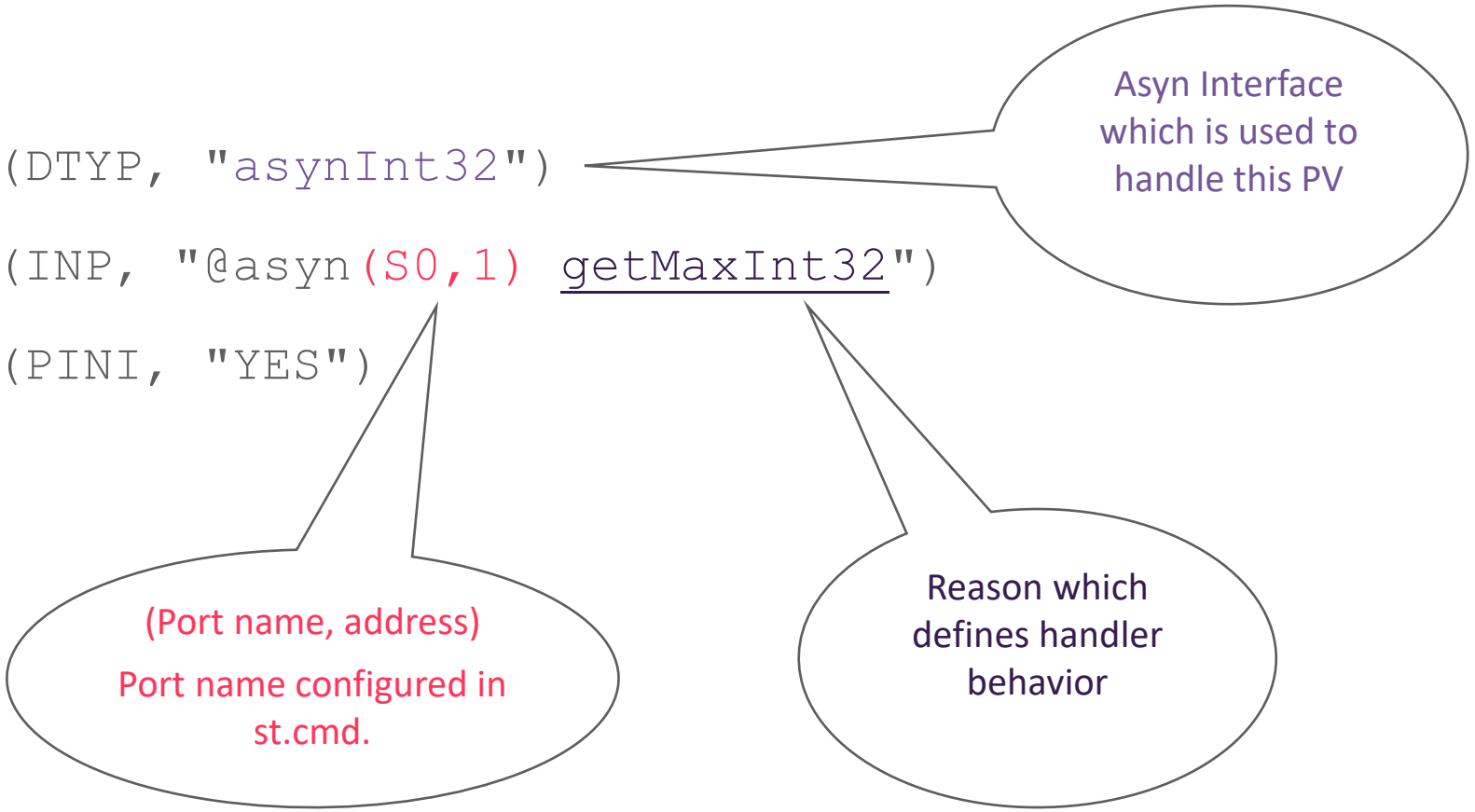
# How is it done?

```
device(ai,      INST_IO, asynAiInt32,  "asynInt32")
device(ai,      INST_IO, asynAiInt32Average, "asynInt32Average")
device(ao,      INST_IO, asynAoInt32,  "asynInt32")
device(bi,      INST_IO, asynBiInt32,  "asynInt32")
device(bo,      INST_IO, asynBoInt32,  "asynInt32")
device(mbbi,    INST_IO, asynMbbiInt32, "asynInt32")
device(mbbo,    INST_IO, asynMbboInt32, "asynInt32")
device(longin,  INST_IO, asynLiInt32,  "asynInt32")
device(longout, INST_IO, asynLoInt32,  "asynInt32")
```

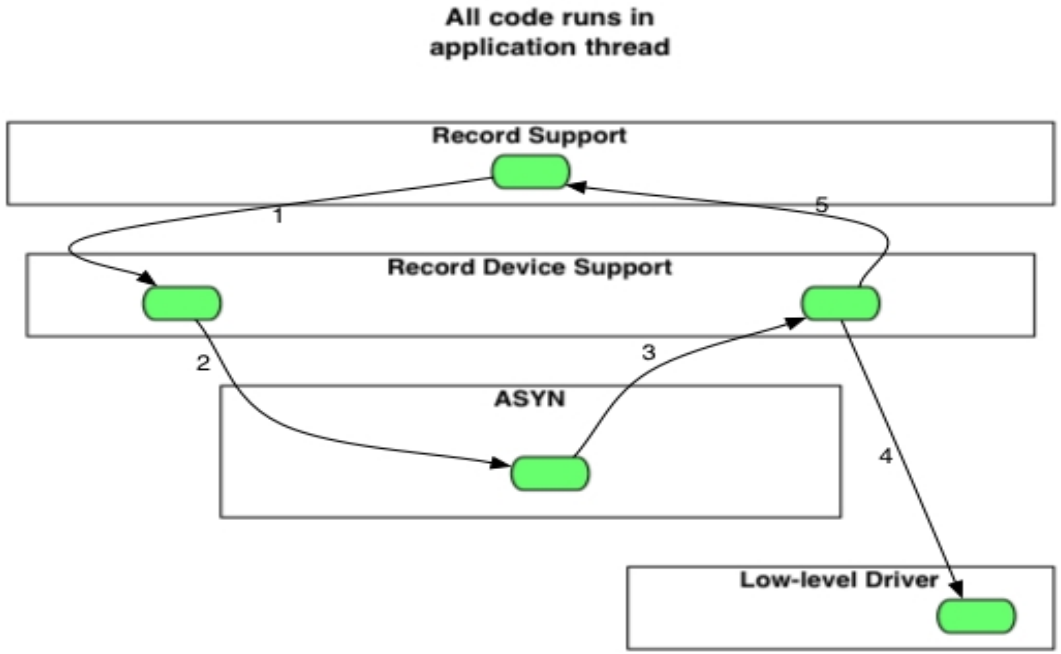
…

# Record Configuration

```
record (longin, "$(app)-LONGIN-MAX-INT32")
{
        field(DTYP, "asynInt32")
        field(INP, "@asyn(S0,1) getMaxInt32")
        field(PINI, "YES")
}
```



# Synchronous Flow




- Sometimes blocking is desired
- We don't write extra stuff for blocking support

# Synchronous Methods

- Read

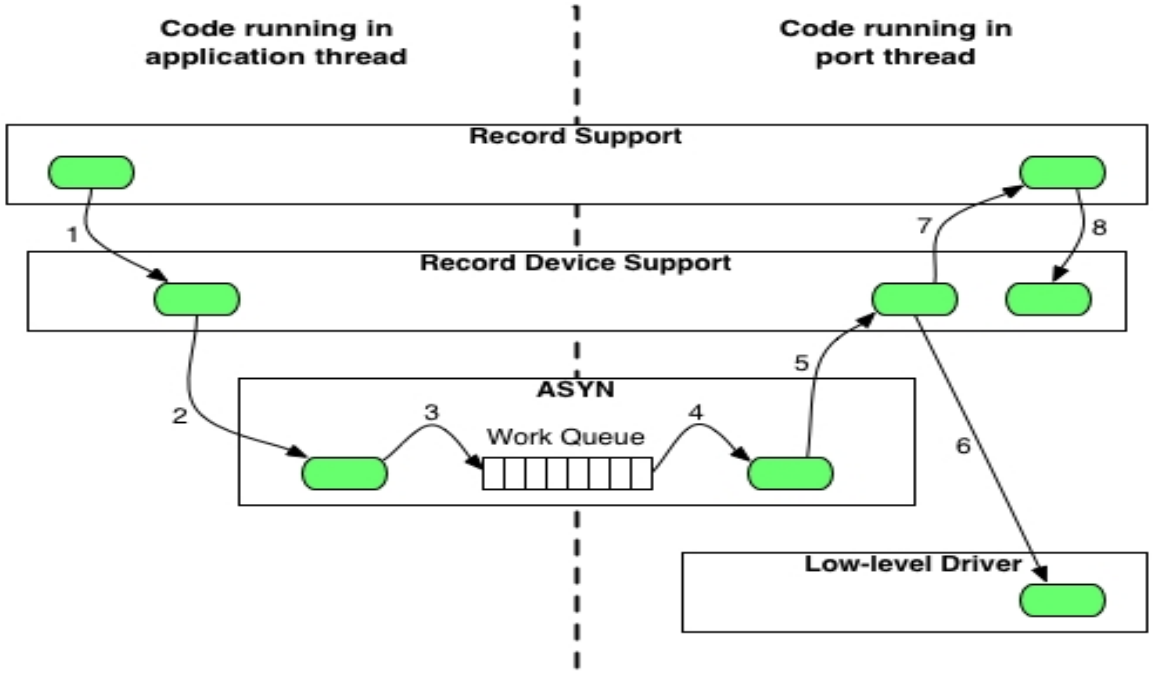
```
asynStatus (*read) (
    asynUser *pasynUser,
    epicsXXX *pvalue,
    size_t nelem,
    size_t *nIn,
    double timeout );
```

- Write

```
asynStatus (*write) (
    asynUser *pasynUser,
    epicsXXX *pvalue,
    size_t nelem,
    double timeout );
```

`asynUser` - is an interface between generic device support and specific driver. It is managed by asynManager.

# Asynchronous Flow




- An `asynUser` is the means by which asynManager  manages multiple requests
- An `asynUser` should be created for each “atomic” access to low level driver

# Support for Interrupts

- Interfaces with interrupt callback support: `asynInt32`, `asynInt32Array`, `asynUInt32Digital`, `asynFloat64` and `asynFloat64Array`

- `registerInterruptUser(…,userFunction, userPrivate, …)`
  - Driver will call `userFunction(userPrivate, pasynUser, data)` whenever an interrupt occurs
  - Callback will not be at interrupt level, so callback is not restricted in what it can do

- Callbacks can be used by device support, other drivers, etc.

# Adding ASYN: Application Conf.

- Register asyn in the config file (`config/RELEASE`)

```
ASYN=$(EPICS_BASE)/../modules/asyn
```

# Adding ASYN: DB Configuration

- DBD:

```
include "asyn.dbd"
```

- Record Configuration:
  - DTYP

```
field(DTYP,"asynXXX")
```

  - INP/OUT

```
field(INP,"@asyn(portName, addr, timeout) reason")
```

    - or

```
field(INP,"@asynMask(portName, addr, mask, timeout) reason")
```

# ASYN Port Implementation

- C version
  - Define/implement common ASYN interfaces

    `asynCommon`

    `asynDrvUser`
  - Define/implement device support interfaces

    `asynInt32`
    - ...

- C++ version
  - Extend asynPortDriver class
    - No need to define separate interfaces everything already defined

- Provide asynPort initialization function.

# ASYN Port Implementation (cont.)

- Define supported reasons

```
asynUser.reason
```

- Register interrupt callback functions
  - For SCAN = I/O Interrupt PVs

# Existing Port Drivers

asynGpib
Port Drivers

- Local Serial Port
- TCP/IP or UDP/IP Port
- TCP/IP or UDP/IP Server
- USB TMC (Test and Measurement Class)
- VXI-11
- Linux-Gpib
- Green Springs IP488
- National Instruments GPIB-1014D
- Additonal Drivers

COSYLAB

# EPICS
# Advanced part

## Stream Device

COSYLAB

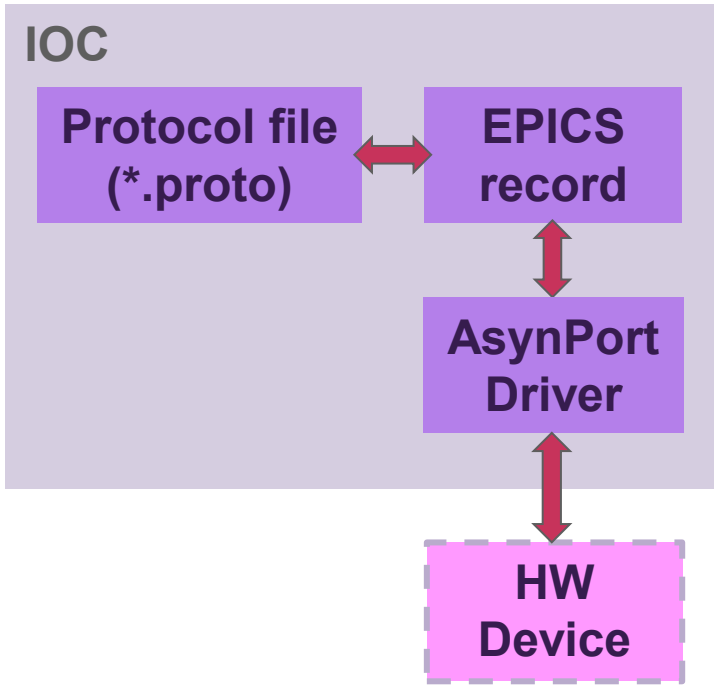
# Overview

- What is Stream Device?
- Architecture
- Build Configuration
- Protocol Files
- Features
- Example Protocol File
- Conclusion

# What is StreamDevice?

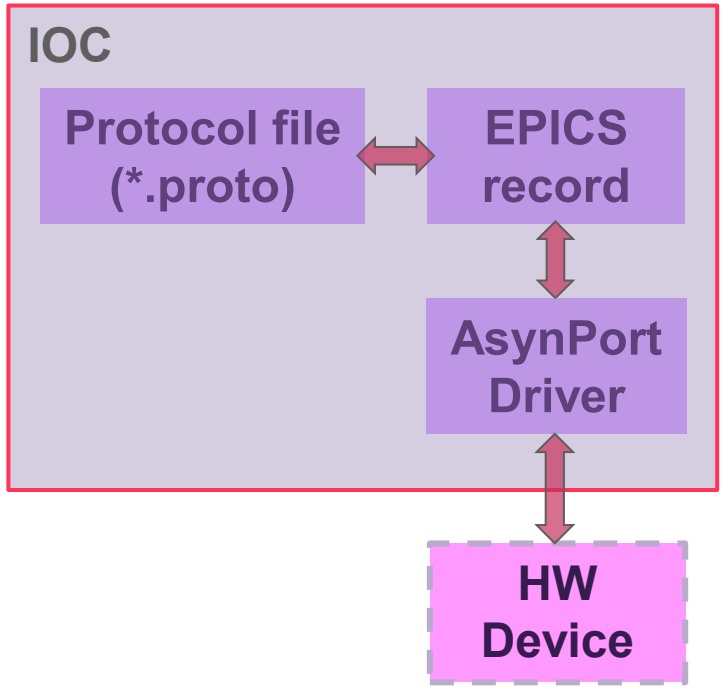
- EPICS module used to define string-based communication protocols between EPICS records and devices
    - E.g. Message sent: `V?`; Message received: `V=2.1`
- Based on AsynDriver, has all of its advantages:
    - Thread handling
    - Proper resource locking
    - Queueing (by priority)
- But yet simple configuration, no programming required
- Based on plain text protocol files
- All documentation: https://paulscherrerinstitute.github.io/StreamDevice/

# Architecture




- Protocol file includes a set of protocols
  - A protocol is a set of “instructions” how to make a particular data exchange with a particular device
- Records may use one protocol to configure how they function
- Records communicate with HW device by exchanging the messages described in their assigned protocol via an AsynPort Driver

# Include Stream Device Support




- Stream Device support must be loaded in EPICS Application
  - In `configure/RELEASE` add the following lines

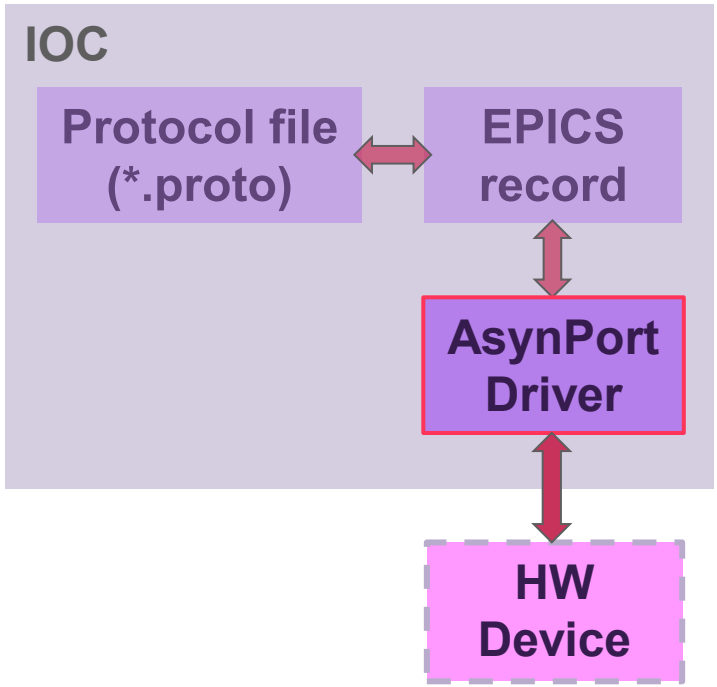
```
STREAM = /opt/epics/stream-<version>/
ASYN = /opt/epics/asyn-<version>/
```

  - In `<AppName>App/src/Makefile`, include:

```
<AppName>_DBD += stream-base.dbd
<AppName>_DBD += asyn.dbd
<AppName>_DBD += drvAsynIPPort.dbd

<AppName>_LIBS += stream
<AppName>_LIBS += asyn
```

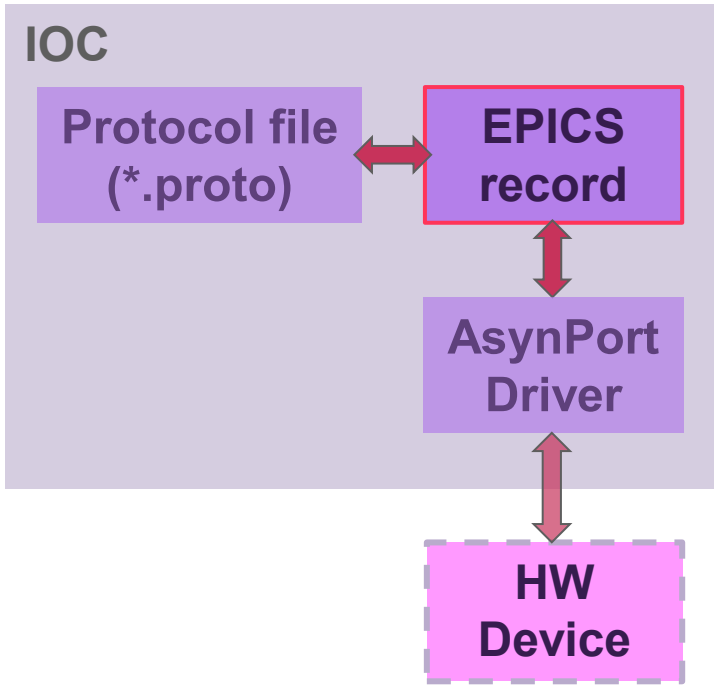
# Asyn Port Driver Configuration




- For debugging, the asyn trace mask can be used to output to shell exactly what is being transmitted:
  `asynSetTraceMask("DEV1",-1,0x9);`
  `asynSetTraceIOMask("DEV1",-1,0x2)`
- Check `asynDriver.h` for details

- AsynPort is initialized in IOC start file `st.cmd`
- IP connection:
  `drvAsynIPPortConfigure ("DEV1", "192.168.1.212:8080")`
  Creates a connection called `DEV1` with a network device reachable at `192.168.1.212` over port `8080`
- Serial connection:
  `drvAsynSerialPortConfigure("DEV2", "/dev/ttyS1")`
  Creates a connection called `DEV2` with a serial device connected to `/dev/ttyS1`
- Optional asyn parameters (line terminators, serial communication parameters…) may be specified with other commands

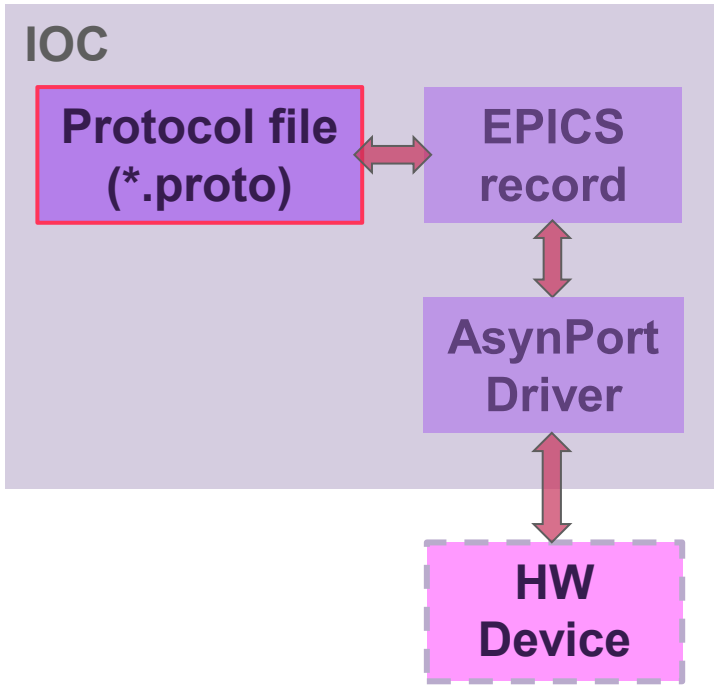
# Record Configuration




- Records that may use Stream Device out of the box:
  `ai, ao, bi, bo, mbbi, mbbo, mbbiDirect, mbboDirect, longin, longout, stringin, stringout, waveform, calcout, scalcout`
- Others may as well, but require support

- Two fields are used to tell a record to use Stream Device
- Device Type (`DTYP`):
  `field (DTYP, "stream")`
- Either Input (`INP`) or Output (`OUT`):
  `field(INP,"@protoFile.proto getFrequency DEV1")`
  `field(OUT,"@protoFile.proto setFrequency DEV1")`
- First example tells input record to acquire its value using protocol `getFrequency`, specified in the file `protoFile.proto`, from device connection `DEV1`
- Second example tells output record to send its value using protocol `setFrequency`, specified in the file `protoFile.proto`, to device connection `DEV1`

# Protocol File Locations




- Records are given the name of a protocol file, but not a path to find it
- Paths to protocol files are declared in `st.cmd`
- Using command:
  `epicsEnvSet("STREAM_PROTOCOL_PATH", "/path1/:/path2/")`
- Multiple paths can be provided using a colon as a separator.

# Protocol Files

- Plain text ASCII files containing a set of protocols
- Each protocol has a name and a sequence of commands
    - `in` *`string`*`;` - reads message that matches “string”
    - `out` *`string`*`;` - sends message that matches “string”
    - others: `wait`, `event`, `exec`, `disconnect`, `connect`
- “Strings” are matched using format converters
    - doubles or floats: `%f`, `%e`, `%E`, `%g`, `%G`
    - shorts, ints or longs: `%d`, `%i`, `%u`, `%o`, `%x`, `%X`
    - strings or chars: `%s`, `%c`
    - enumerations: `%{string0|string1|...}`
    - others: `%[charset]`, `%b`, `%Bzo`, `%r`, `%R`, `%D`, `%<checksum>`, `%/regex/`, `%#/regex/subst/`, `%m`, `%T(timeformat)`
- Optional width and flags can modify behaviour of format converters
    - Width specifies number of characters string should have
    - Flags: `*`(skip), `#`(format), `+`(positive int prefix), `0`(pad), `-`(left justify), `?`(allow fail), `=`(compare), `!`(impose width)

# Features

- **Protocol nesting** – one protocol may call another
- **Redirection** – protocols may read/write from/to other records and/or fields *(in general not a good practice, but possible)*
- **System variables** – preset variables (can be local or global): `LockTimeout`, `WriteTimeout`, `ReplyTimeout`, `ReadTimeout`, `PollPeriod`, `Terminator`, `OutTerminator`, `InTerminator`, `MaxInput`, `Separator`, `ExtraInput`
- **User variables** – free to customize, called with `$` prefix
- **Arguments** – arguments can be passed from the record to the protocol it calls (up to 9, called with `\$1`..`\$9`)
- **Exception handlers** – clauses that are called only when certain conditions are met: `@mismatch`, `@writetimeout`, `@replytimeout`, `@readtimeout`, `@init`

# Example Protocol File

```
Terminator = CR LF;             # sets line terminators for both inputs and outputs to "\r\n"
f = "FREQ";                     # User variable: sets f to "FREQ" (including the quotes)
f1 = $f " %f";                  # User variable: sets f1 to "FREQ %f"
getFrequency {                  # ai record could call this using INP="@protoFile.proto getFrequency DEV1"
    out $f "?";                 # same as: out "FREQ?";
    in $f1;                     # same as: in "FREQ %f";
}
setFrequency {                  # ao record could call this using OUT="@protoFile.proto setFrequency DEV1"
    out $f1;                    # same as: out "FREQ %f";
    @init { getFrequency; }     # Exception handler calling another protocol: initial sync
}
getSwitch {                     # bi record could call this using INP="@protoFile.proto getSwitch DEV1"
    out "SW?";
    in "SW %{OFF|ON}";          # if input is "SW OFF" then VAL=0, if input is "SW ON" then VAL=1
}
setSwitch {                     # bo record could call this using OUT="@protoFile.proto setSwitch DEV1"
    out "SW %{OFF|ON}";         # if VAL=0, send "SW OFF", if VAL=1, send "SW ON"
    @init { getSwitch; }        # Exception handler calling another protocol: initial sync
}
move {                          # longout record could call this using OUT="@protoFile.proto move(X) DEV1"
    out "\$1 GOTO %d";          # Message sent would be "X GOTO %d"
}
debug {                         # Generic command for stringout record: any string written to
    ExtraInput = Ignore;        # ... record's VAL field is sent to device, any reply from device
    out "%s"; in "%39c";        # ... will be written to record's VAL field cropped at 39 chars.
}
```

# Messages don’t get mixed up!

- **Use-case:** Two records that use similar protocols for the same device are triggered to process at (almost) the same time
  - Record A protocol: { `out “V PS1?”; in “%f”;` }
  - Record B protocol: { `out “V PS2?”; in “%f”;` }
- If both output messages are sent to the same device at the same time, how will the responses not get mixed up?
  - AsynDriver sends the first output message it receives (either A or B).
  - Second output message gets buffered in a queue and doesn’t get sent (yet).
  - Only once the first protocol is completed (a proper reply is received within the timeout), the second output message gets sent automatically.
- AsynDriver takes care of everything!

# Conclusion

- Stream Device offers universal support to integrate all message-based devices
- AsynDriver takes care of all the complicated low-level stuff and provides out-of-the-box support for most low-level drivers:
  serial, TCP/IP, VXI-11, IEEE-488, …
- Only configuration, no coding
- Protocols are reloadable → Faster development
- Support for most common record types
- Extendable:
  - Write your own format converters
  - Write support for other record types
  - Use StreamDevice with other non-Asyn low-level drivers
- However:
  - Not a programming language (no *for*s, *if*s, etc.)
  - Everything is a string (no other datatypes, just format converters)
  - Line terminators can cause headaches (use Asyn Trace Mask for debugging!)
  - Timeouts and errors must be handled
  - Although quick to setup, can take time to make fail-safe (80/20 rule!)

COSYLAB

# CS-Studio

# Outline

- What is CS-Studio
- The motivation behind it
- CS-Studio core architecture
- CS-Studio functionalities - tools

COSYLAB

# What is CS-Studio?

- Control System Studio
- An Eclipse RCP application
- Common platform for control system applications
  - Users – single point of control
  - Developers – write portable applications
- Front-end to different control systems

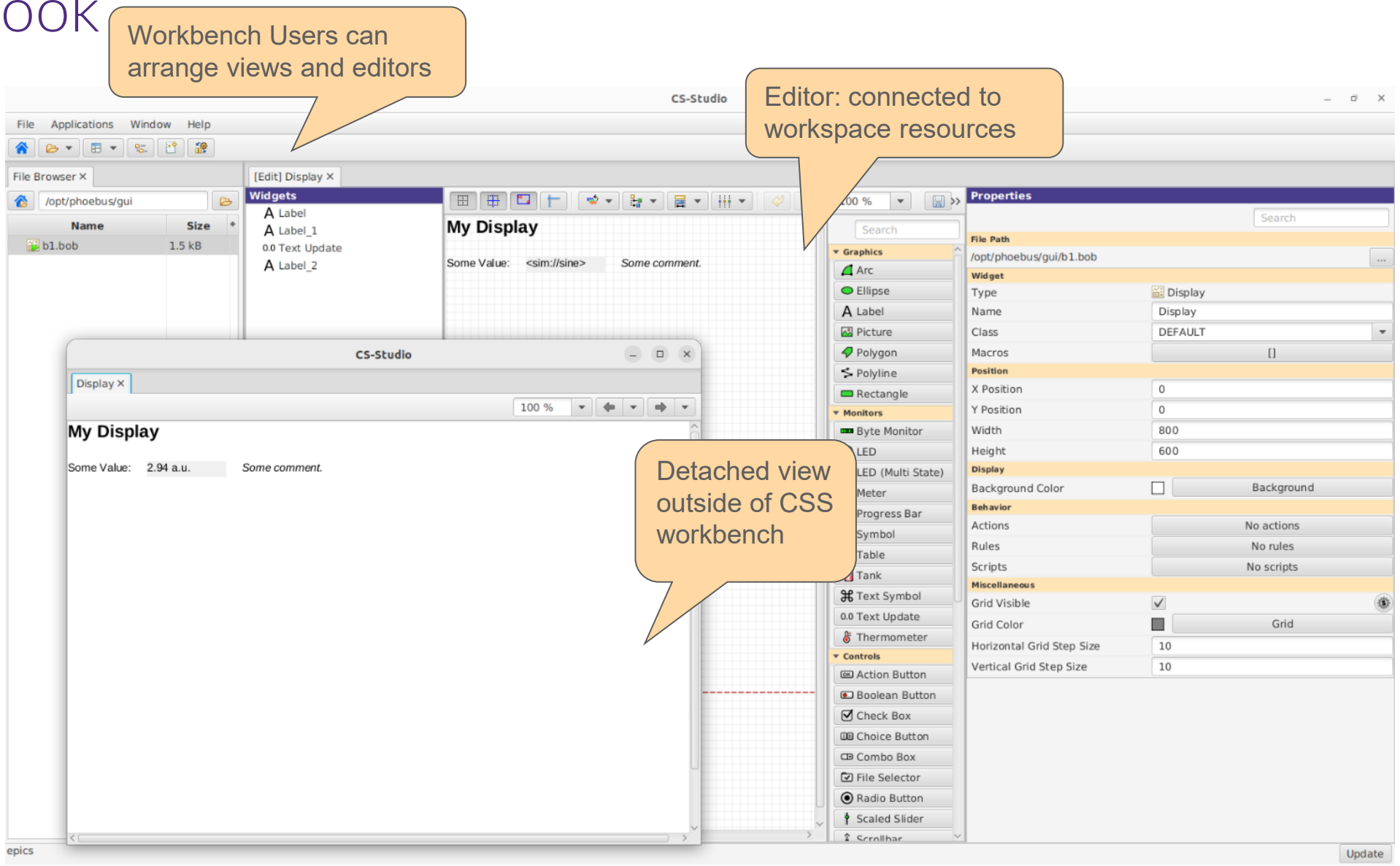
# Motivation

- Too many CS technologies
  - X-Probe - Single-PV inspection
  - StripTool - Plot live data over time
  - Archive Data Viewer - Plot historic data over time
  - Display Manager: EDM, MEDM – HMI display (and editor)
  - …
- Different look and feel of applications
- No simple unified way of usage
- No inherent ability to exchange information between applications
- CS-Studio effectively combines all of those into one application

# CS-Studio Phoebus

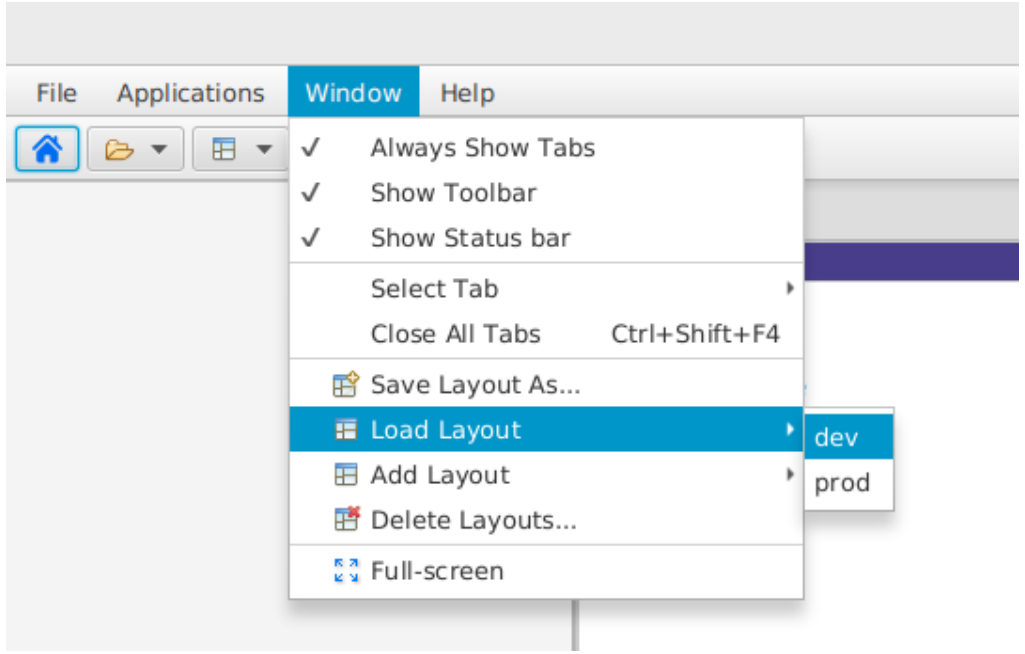
- Part of a wide community of users and developers:
  BNL, DESY, FRIB, SNS, ITER, ESS, …
- Java – OS independent
- Open-source
- CS-Studio is simply a collection of plugins
- Some concepts:
  - Editors (OPI editor, plot editor…)
  - Views – windows that provide a view of data
  - Perspectives – arrangement of views, editors, toolbars…

# CSS Look


Workbench Users can arrange views and editors
Editor: connected to workspace resources
Detached view outside of CSS workbench
CS-Studio
File Applications Window Help
File Browser
/opt/phoebus/gui
Name Size
b1.bob 1.5 kB
[Edit] Display
Widgets
Label
Label_1
0.0 Text Update
Label_2
My Display
Some Value: <sim://sine>
Some comment.
Properties
Search
File Path
/opt/phoebus/gui/b1.bob
Widget
Type Display
Name Display
Class DEFAULT
Macros []
Position
X Position 0
Y Position 0
Width 800
Height 600
Display
Background Color Background
Behavior
Actions No actions
Rules No rules
Scripts No scripts
Miscellaneous
Grid Visible
Grid Color Grid
Horizontal Grid Step Size 10
Vertical Grid Step Size 10
Graphics
Arc
Ellipse
Label
Picture
Polygon
Polyline
Rectangle
Monitors
Byte Monitor
LED
LED (Multi State)
Meter
Progress Bar
Symbol
Table
Tank
Text Symbol
0.0 Text Update
Thermometer
Controls
Action Button
Boolean Button
Check Box
Choice Button
Combo Box
File Selector
Radio Button
Scaled Slider
Scrollbar
CS-Studio
Display
100 %
My Display
Some Value: 2.94 a.u.
Some comment.
epics
Update

# Layouts

- Different layouts can be used for different tasks
  - Development
    - Display Editor – Develop HMIs
    - File Browser – List of all development files
  - Production
    - Alarm - Monitor alarms
    - Data Browser - Plot data trends
    - OPI Runtime - Interact with Control System through HMIs



# Advantages

- CS-Studio allows you to display/set global settings that impact the behavior of all tools
- Tools interact – drag-and-drop, open related displays from alarm tool...
- Perspectives – save personal or global arrangements of panels as different perspectives – alarm, archiving ...
- Create custom CS-Studio – separate product
- In short - integration

# Disadvantages

- Steeper learning curve
- Too many options for unexperienced users
- Lower performance and higher hardware requirements
- That’s about it

COSYLAB

# CS-Studio

GUI editor

# Outline

- What is Display Builder?
- Runtime
- Editor
  - Widgets
  - Widget properties
  - Scripting

# What is Display Builder?

- Development and runtime environment
  - Display Builder
    - 2nd, improved version of dev and run-time environment
    - Still in development
  - Working on OPI files with extension `.bob`
- A Control System Studio tool – integrated with alarms, archiving …
- Maintained by the EPICS / CS-Studio community
- Two Modes:
  - Runtime – Use widgets to interact with PVs
  - Editor – Design OPI by placing and configuring widgets

# Runtime / Editor

- OPIs can be opened in different modes depending on the tool used:
  - Runtime
    - OPI is not editable
    - OPI tries to connect to corresponding PVs (as CA clients)
    - Can also be invoked with Ctrl+G or by clicking on the button: 
  - Editor
    - OPI is editable
    - No interaction with PVs, no CA requests
  - Text Editor (to view OPI source, not practical in most cases)

COSYLAB

# CS-Studio

Run-time

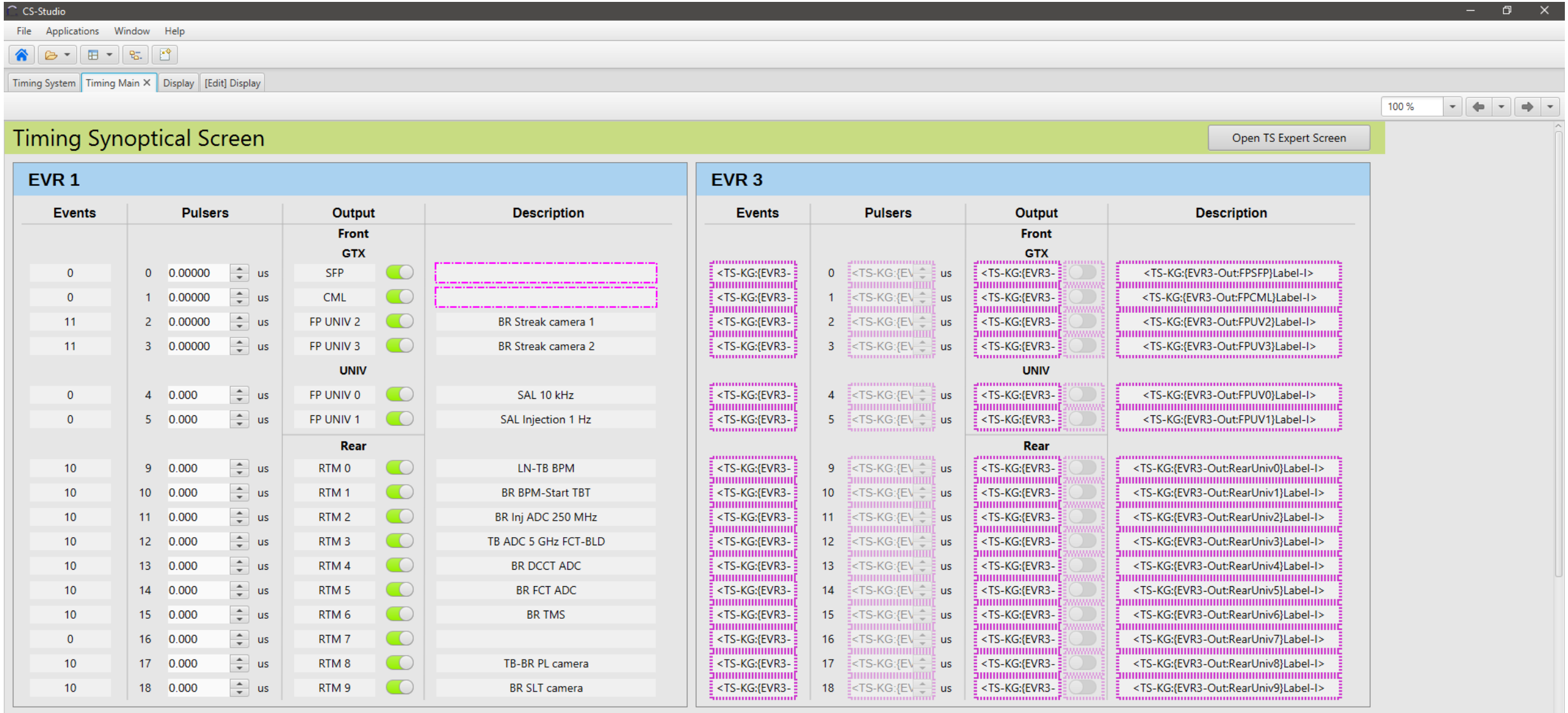
# Runtime

- Displays an already created screen
  - Connect to the CS
  - Draw widgets and update them to PVs
  - Control PV values

- Very simple and straightforward operation

- Browser-like behavior
  - Default – links to other screens open in the same window
  - Possible – new tab or new window

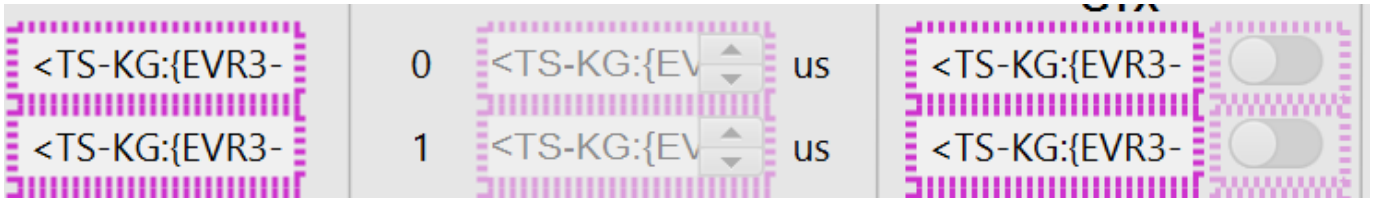
# What Does It Look Like



# PV Connectivity

- Widgets automatically connect / reconnect to PVs
  - If you restart IOC, no need to reopen / refresh OPI

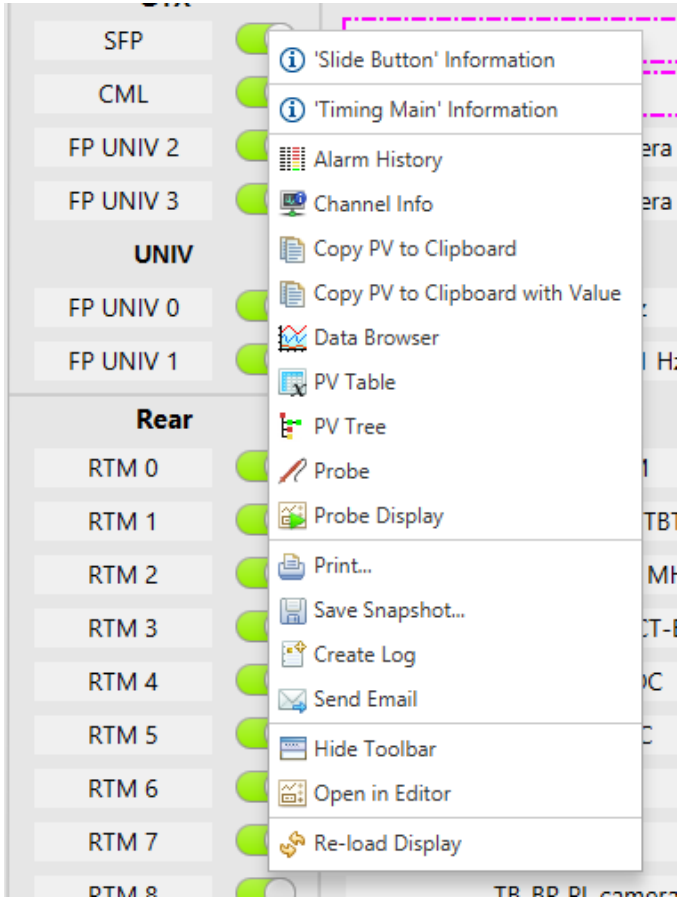
- Common look for disconnected widgets

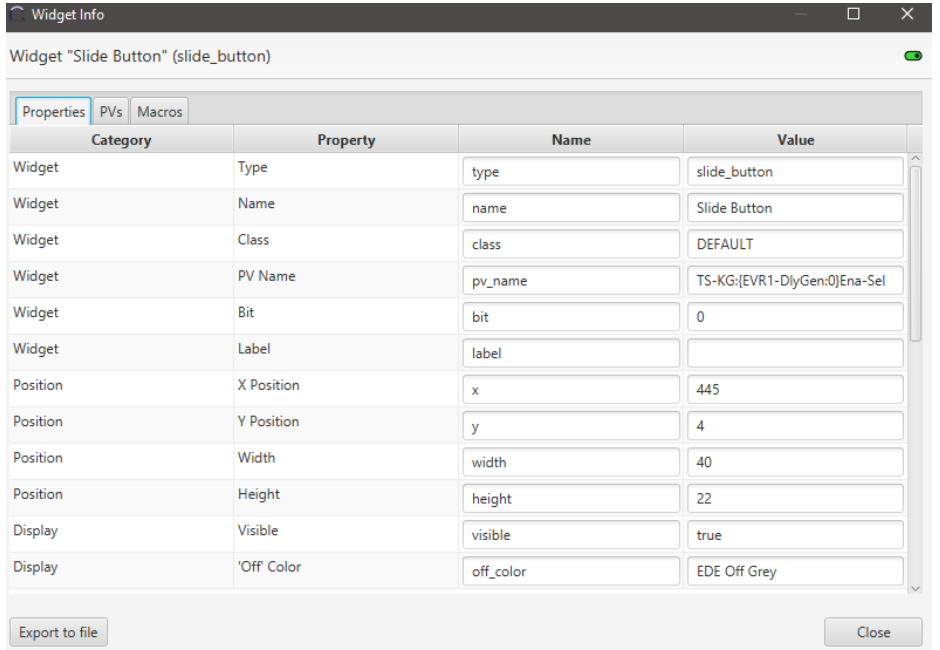


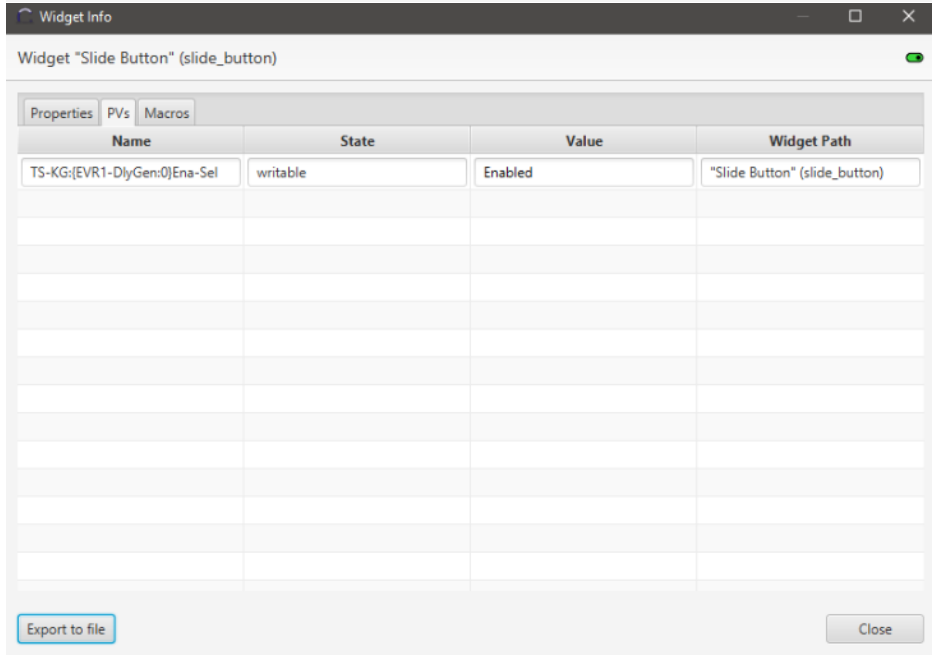

- If PV is write-disabled control widget will be disabled too

# Runtime: Common PV Tools

- Context menu



COSYLAB

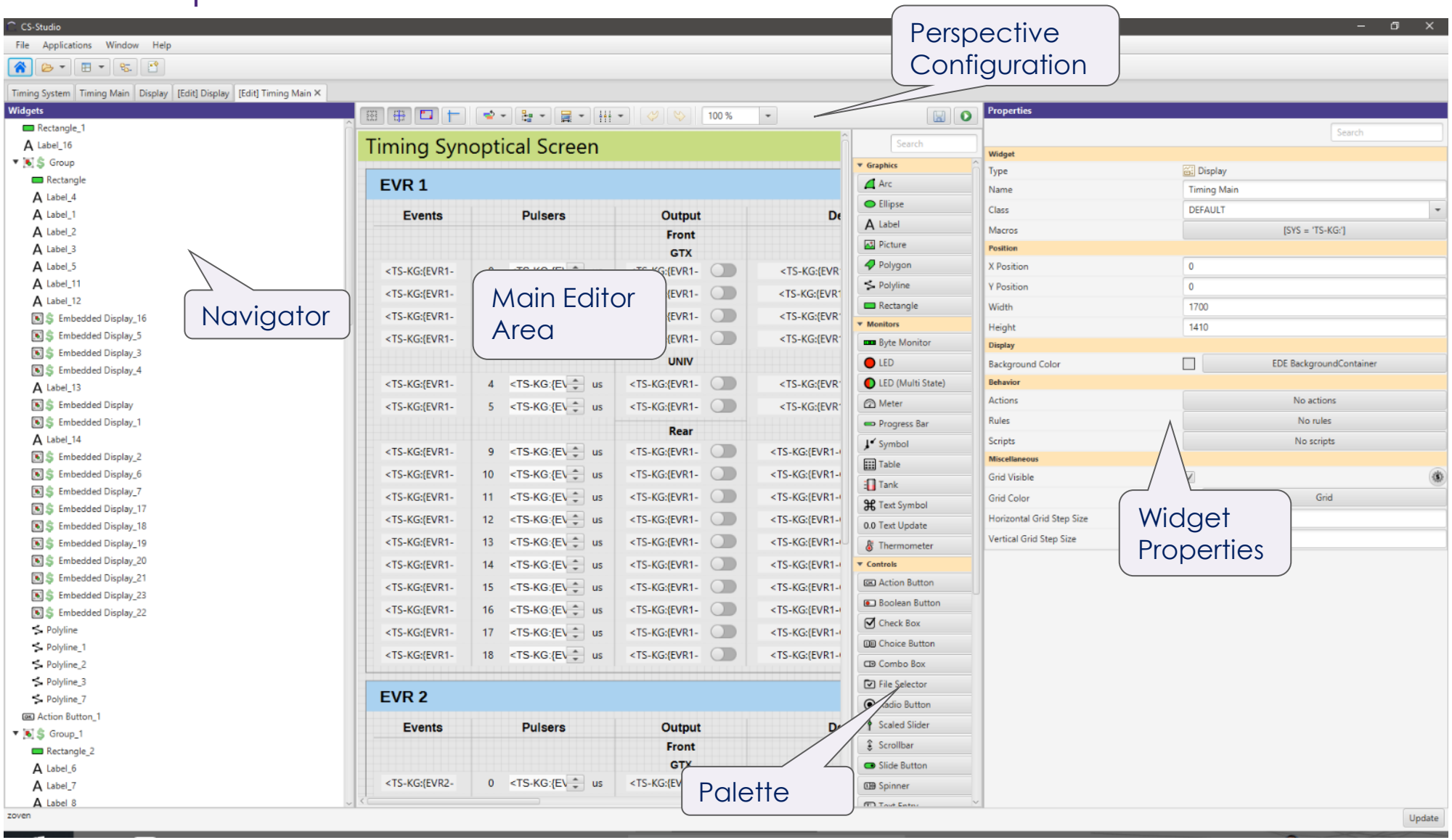
# CS-Studio

Editor

# Editor

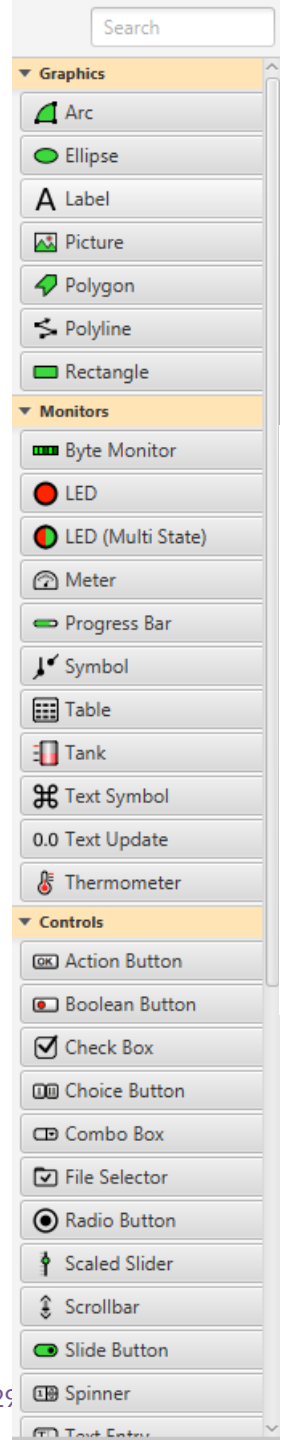
- Simple screen creation + advanced functionality
- WYSIWYG - What You See Is What You Get
- Standard operations
  - Edit multiple widgets at once
  - Copy/paste
  - Move, resize, delete
  - Undo/redo
  - Align widgets with grid, other widgets
  - Zoom
  - ...

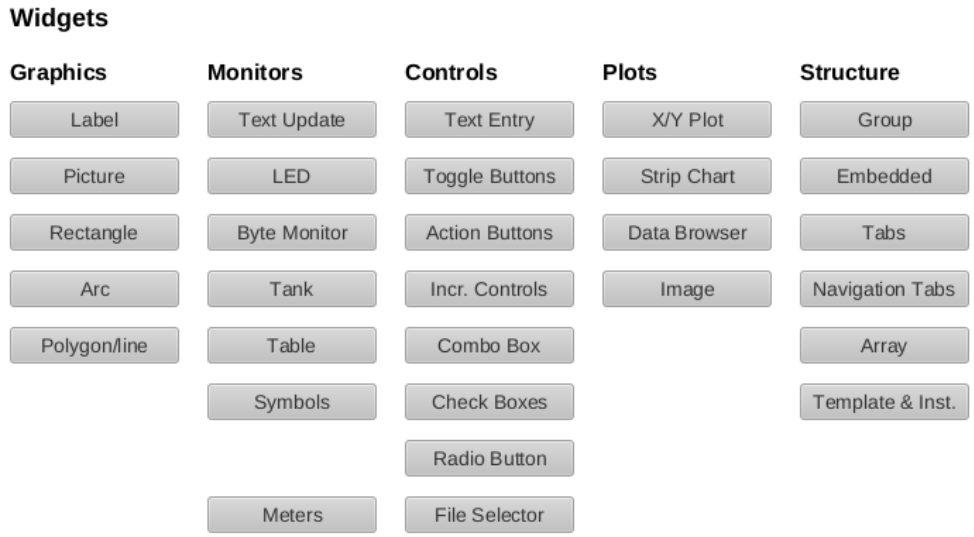
# Editor Perspective



# Widgets

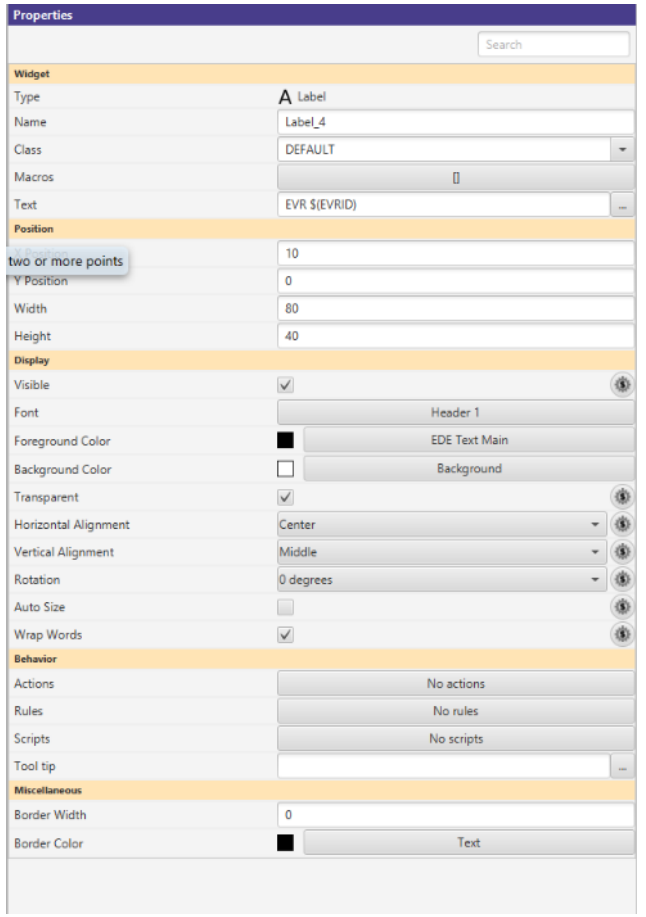
- All available widgets are found in palette
- An OPI consists of a set of widgets laid out on a grid with certain configurable attributes
- Categories of widgets:
  - Graphics – illustrative only, no I&C interaction
  - Monitors – display read-only PV data
  - Controls – editable fields to write PV data
  - Other (containers, groupings…)



# Widget Properties

- Every widget has configurable properties that depend on the widget type.
- Properties are grouped in logical categories, some of which are:
  - Widget (type of widget, name, class, associated PV, …)
  - Position (coordinates, size)
    - Position can also be modified by moving or resizing the widget in the editor, or via Toolbar buttons to align etc.
  - Behavior (actions, rules, scripts, tooltip, alarm sensitivity …)
  - Miscellaneous (color, style,…)

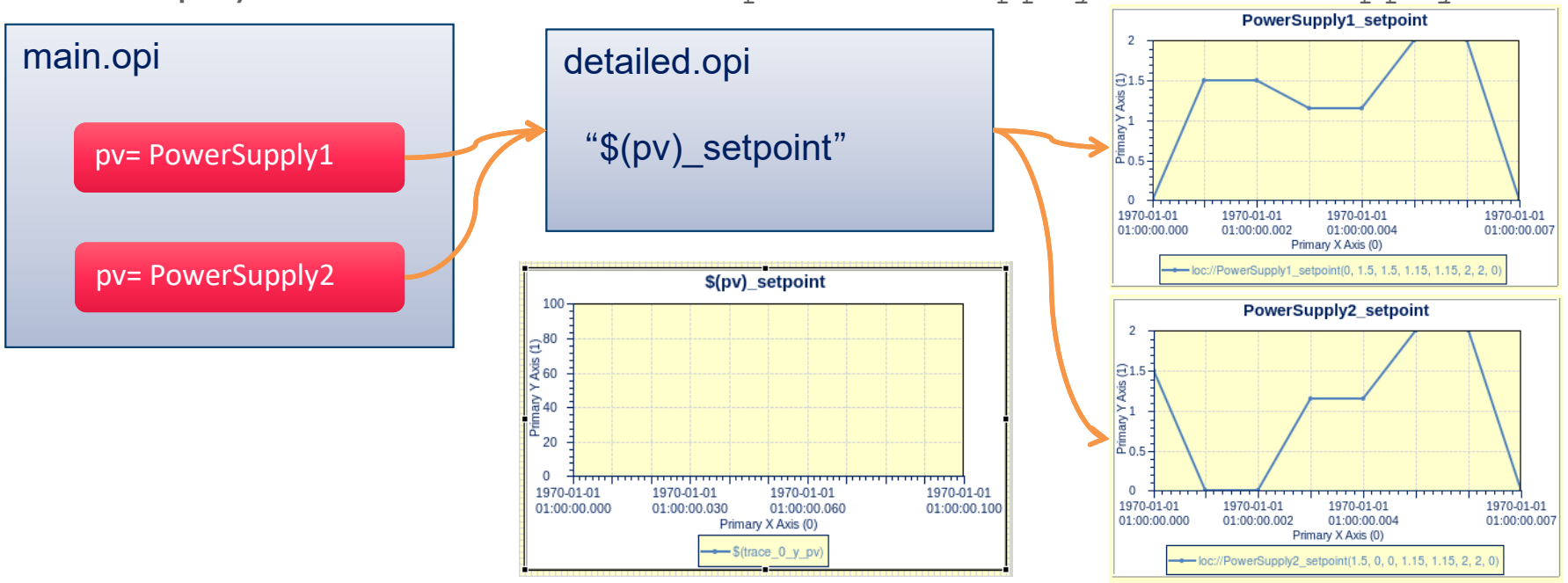


# PV Names

- `ca://some_pv_name`
  - – EPICS Channel Access PV
- `some_pv_name`
  - – Typically same, since “ca://” is the default
- `sim://sine`
  - – Simulated PV. Read online help for details
- `loc://x(4)`
  - – Local PV, initialized to value 4
- `pva://some_pv_name`
  - – EPICS PV Access protocol

# Macros

- Similar OPIs are often used to control similar devices
- No need to create new OPIs for each device of the same type.
- Instead use macros: `$(macro)` or `${macro}`
  - Most often used for partial PV name - `$(pv)_setpoint` or `$(pv)_readback`
  - Such a display can then be invoked with `pv=PowerSupply1` or `PowerSupply2`



COSYLAB

# CS-Studio

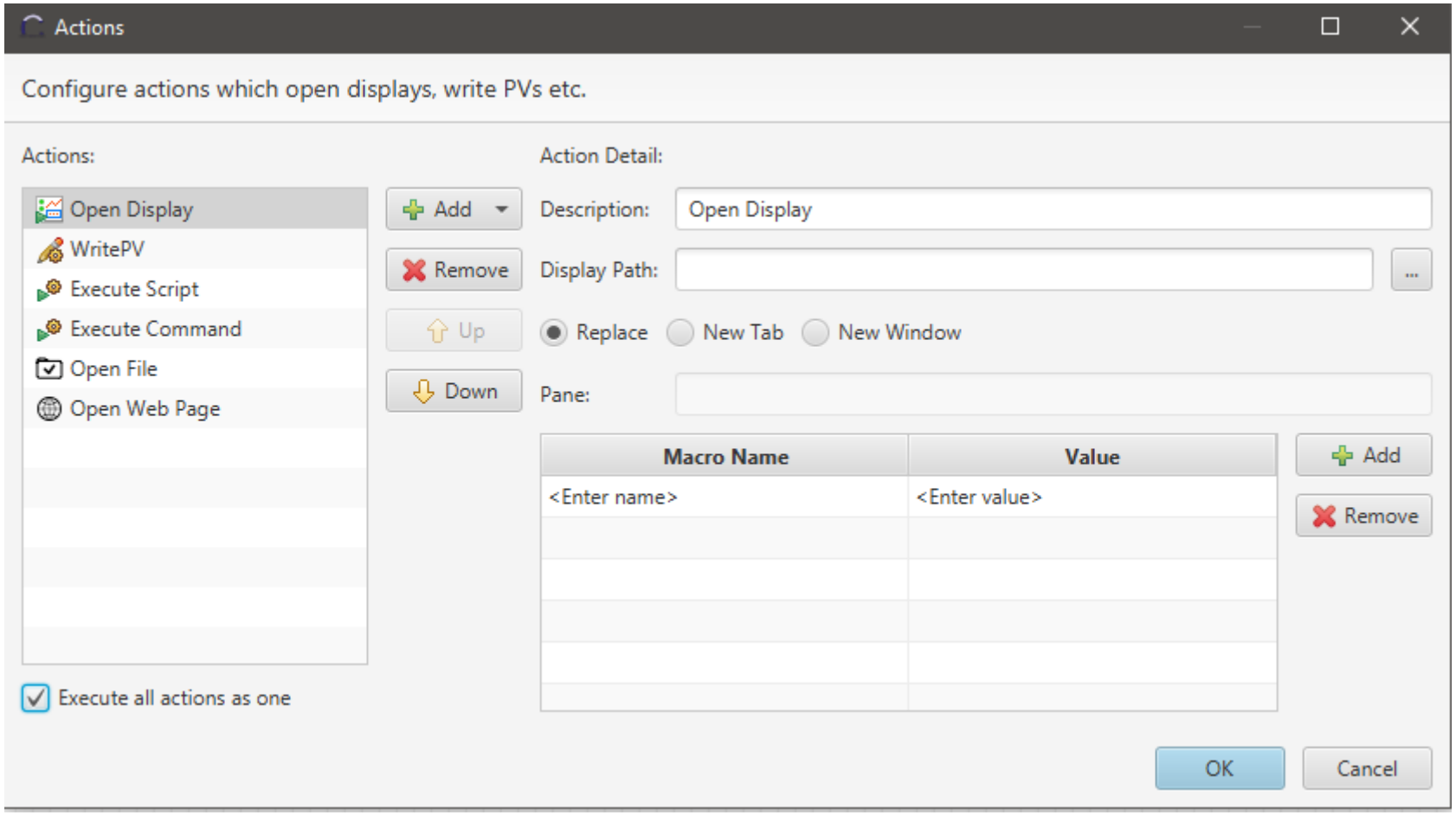
Behavior

# Widgets' Behavior

- Rules
- Scripts
- Actions

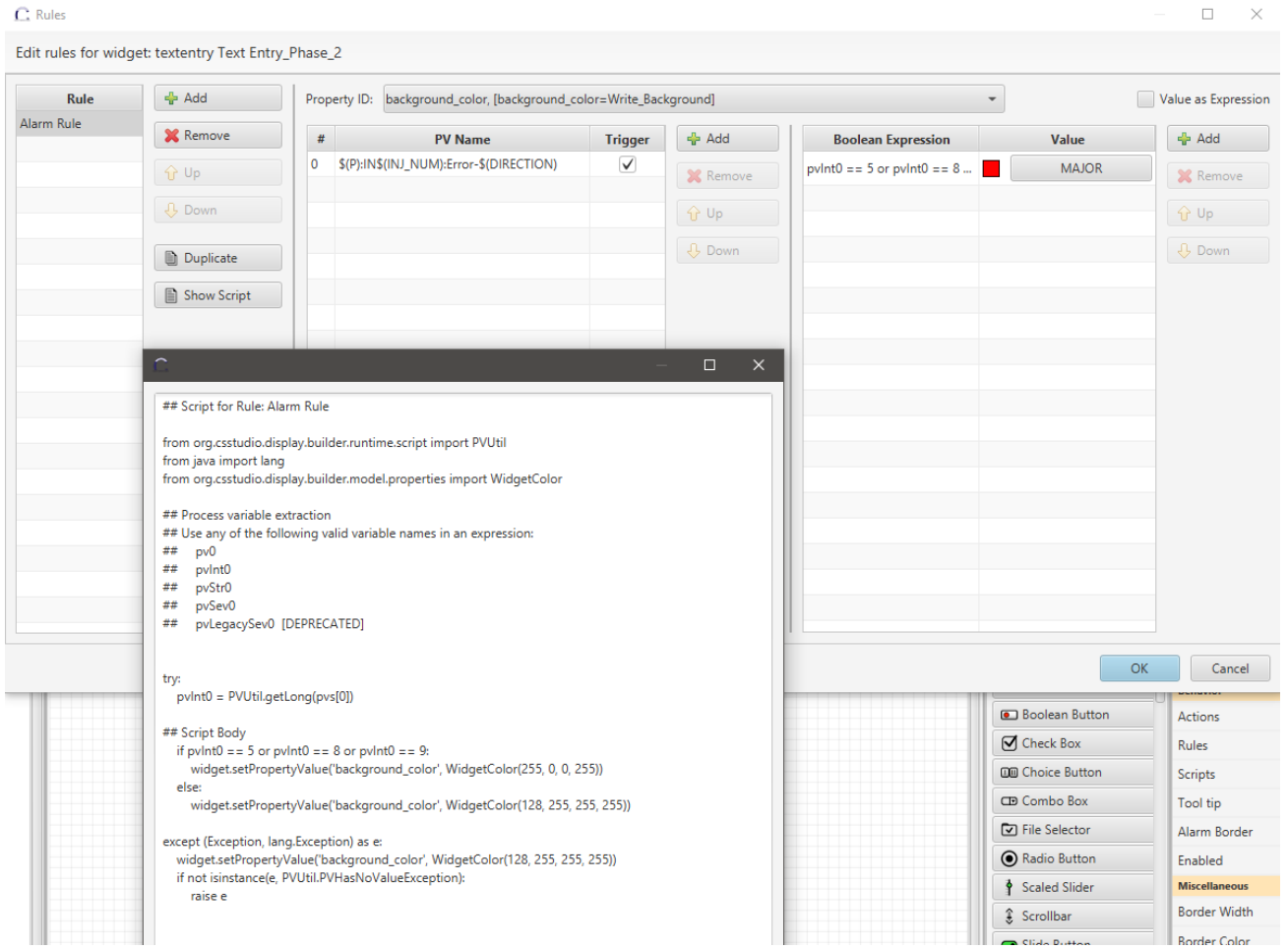
# Actions

- Widgets can have one ore more actions associated
  - Open another OPI
  - Write to PV
  - Execute script
  - Execute command
  - Open file
  - Open webpage



# Rules

- Dynamic property changes – based on PV values
- Rules use Boolean expressions to alter one specific property
  - Implemented as JavaScript (generated automatically)
  - Easier to maintain and control → use scripts only when a rule can't

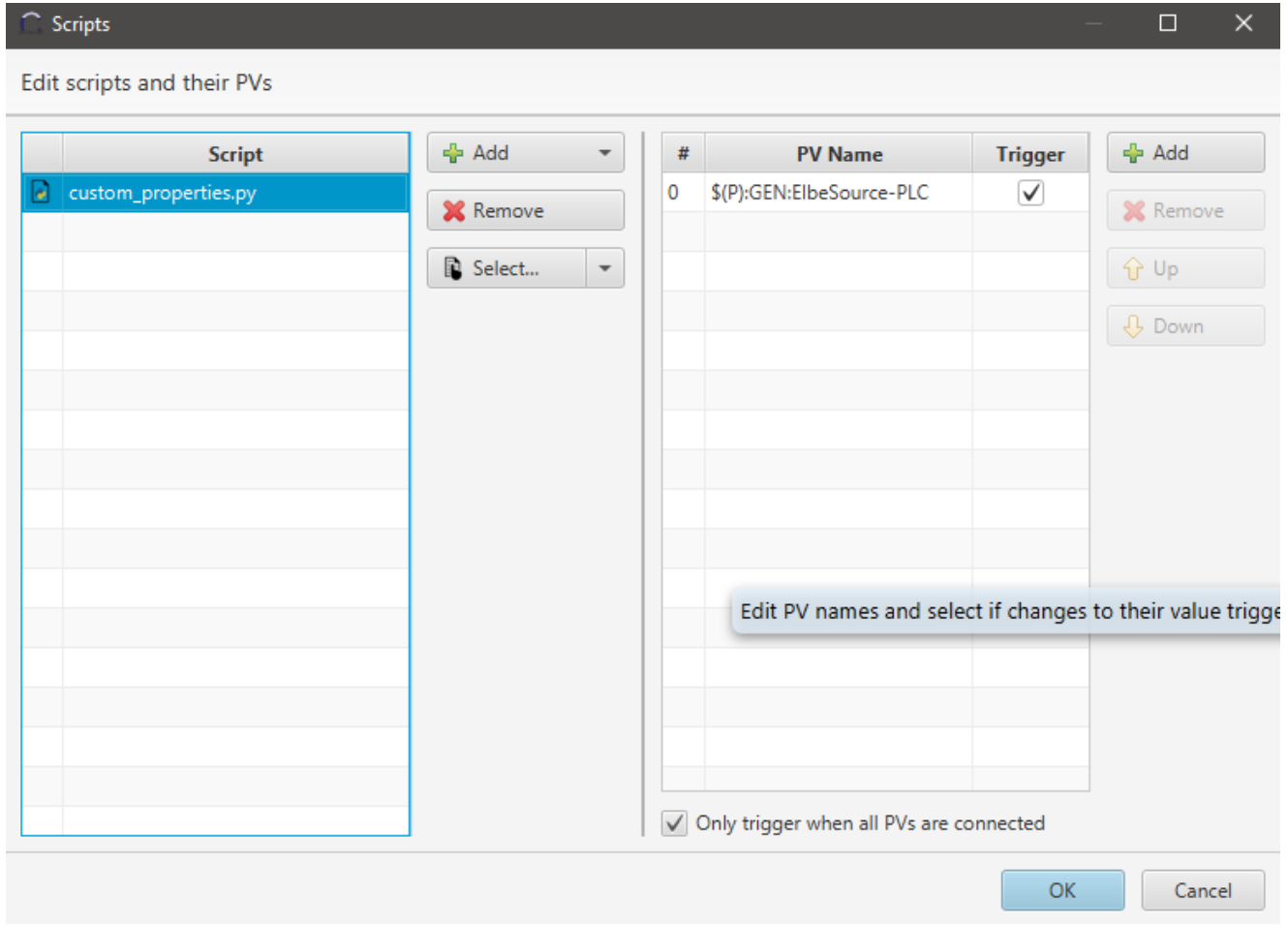


# Rules (cont.)

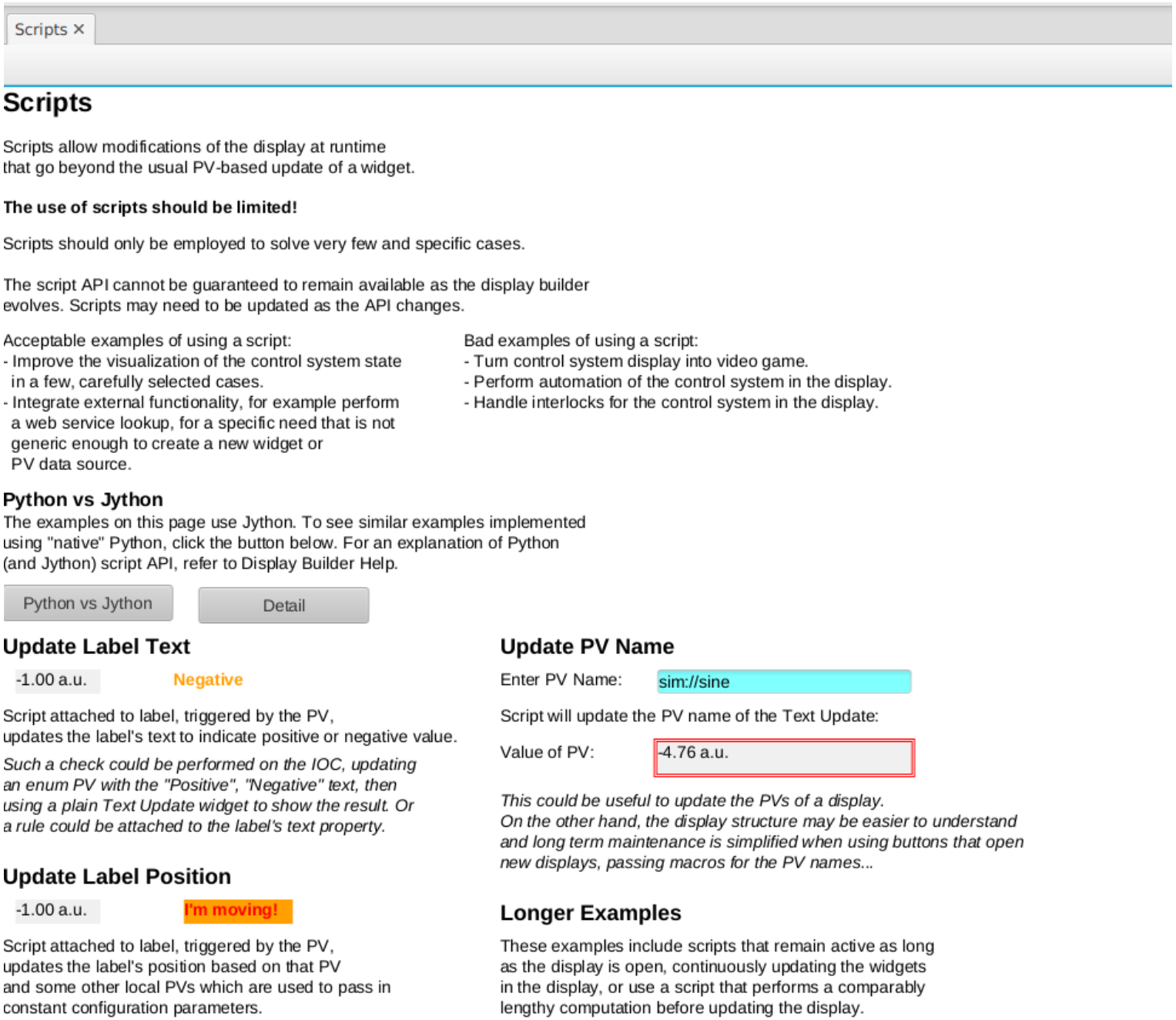
- How to define a rule
  - Behavior → Rules
  - Name new Rule
  - Select property to modify
  - Use PVs as parameters for Boolean expression
  - Write conditions that override property value
  - Rule gets generated as a script

# Scripts

- Complex behavior – attach JavaScript to widget
- Access widget and it's properties
- Access PVs related to the widget
- Script is triggered by input PVs

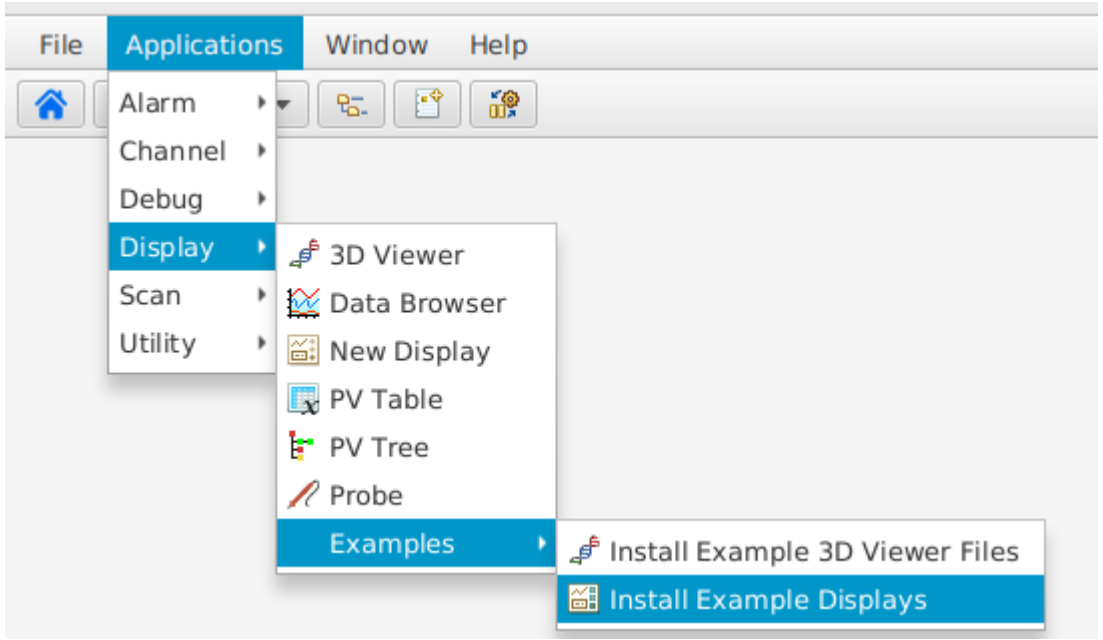


# Scripts (cont.)



# Examples

- Install and check examples!!!




- **Note:** Location of the examples could vary, depends on CS-Studio distribution.

# CS-Studio Tools

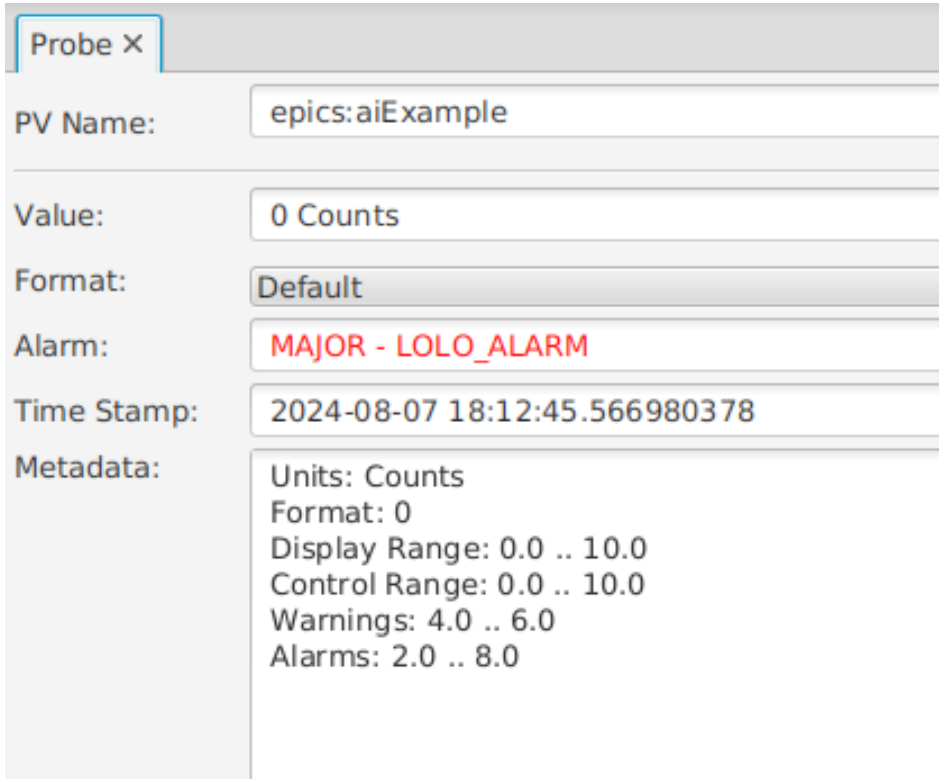
# Included in the CSS

- Probe
- PV Tree
- PV Table
- Data Browser

- Connected to external services
  - Alarms
  - Data Browser
  - Channel Finder
  - Scan Editor/Monitor

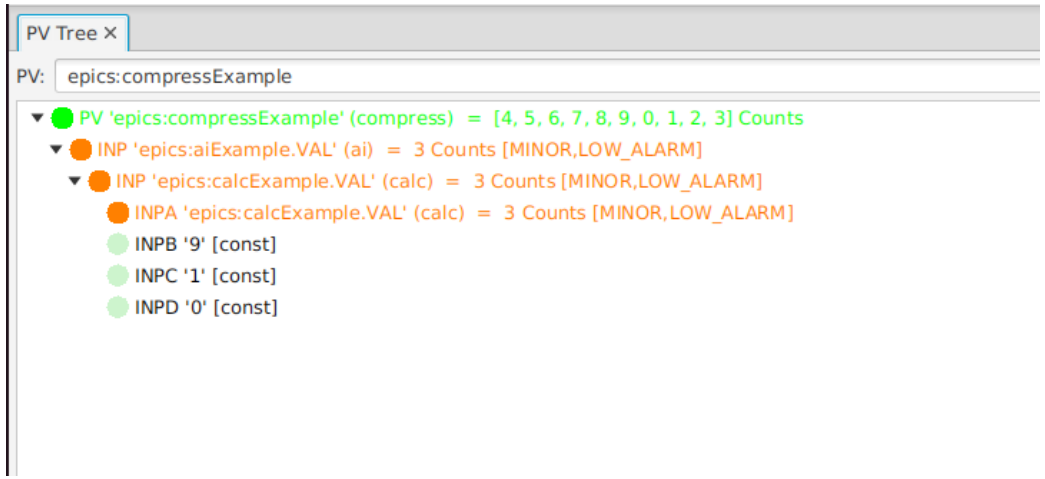
# Probe

- Allows basic reading and writing of PVs
  - Display value, timestamp and alarms
  - Adjust the value and update on IOC



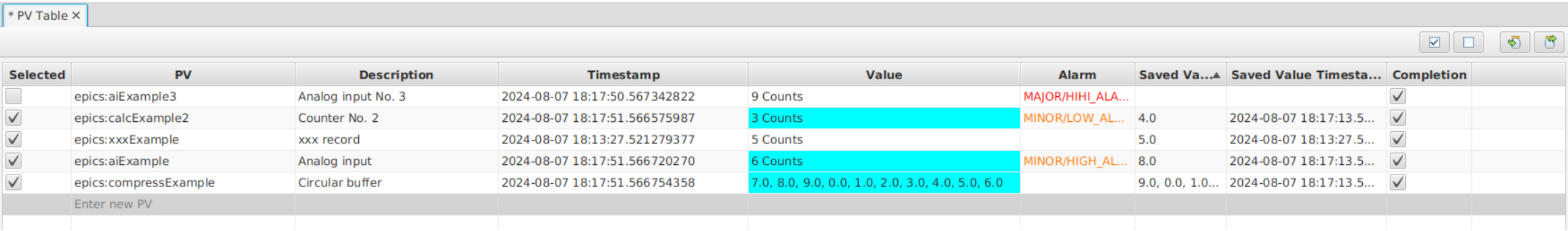
# PV Tree

- Displays hierarchical data flow between EPICS records



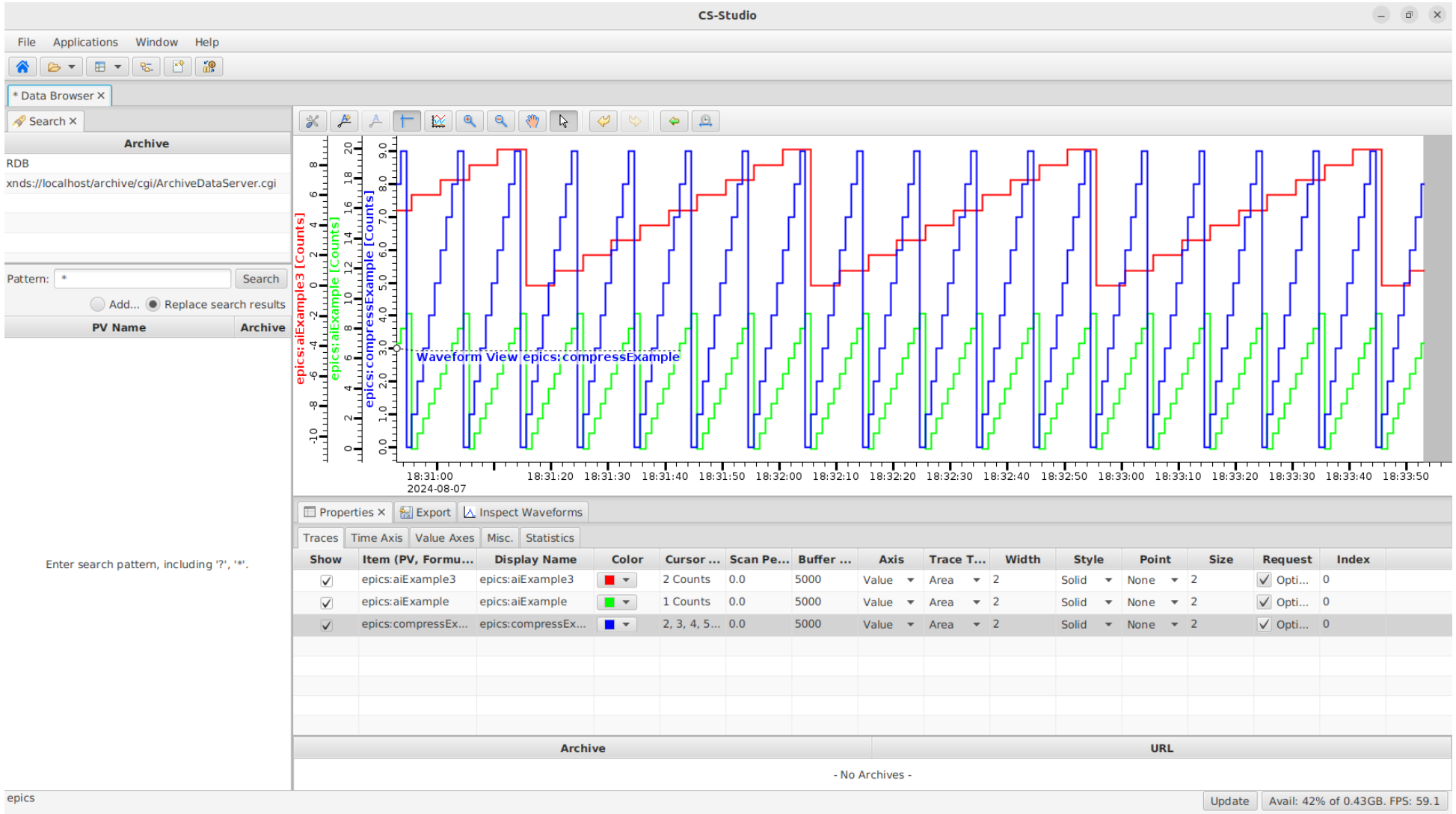
# PV Table

- Tabular view of PV names and their current value
  - Start/stop live update
  - Snapshot of current values
  - Indicating differences between current values and snapshot




| Selected | PV | Description | Timestamp | Value | Alarm | Saved Va... | Saved Value Timesta... | Completion |
|---|---|---|---|---|---|---|---|---|
| ☐ | epics:aiExample3 | Analog input No. 3 | 2024-08-07 18:17:50.567342822 | 9 Counts | MAJOR/HIHI_ALA... | | | ☑ |
| ☑ | epics:calcExample2 | Counter No. 2 | 2024-08-07 18:17:51.566575987 | 3 Counts | MINOR/LOW_AL... | 4.0 | 2024-08-07 18:17:13.5... | ☑ |
| ☑ | epics:xxxExample | xxx record | 2024-08-07 18:13:27.521279377 | 5 Counts | | 5.0 | 2024-08-07 18:13:27.5... | ☑ |
| ☑ | epics:aiExample | Analog input | 2024-08-07 18:17:51.566720270 | 6 Counts | MINOR/HIGH_AL... | 8.0 | 2024-08-07 18:17:13.5... | ☑ |
| ☑ | epics:compressExample | Circular buffer | 2024-08-07 18:17:51.566754358 | 7.0, 8.0, 9.0, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 | | 9.0, 0.0, 1.0... | 2024-08-07 18:17:13.5... | ☑ |
| | Enter new PV | | | | | | | |

# Data Browser



# Thank you.

Advancing humanity. Engineering remarkable.


**Žiga Oven**

ziga.oven@cosylab.com