Container Virtualization for EPICS

PANDA Collaboration Meeting 19/3

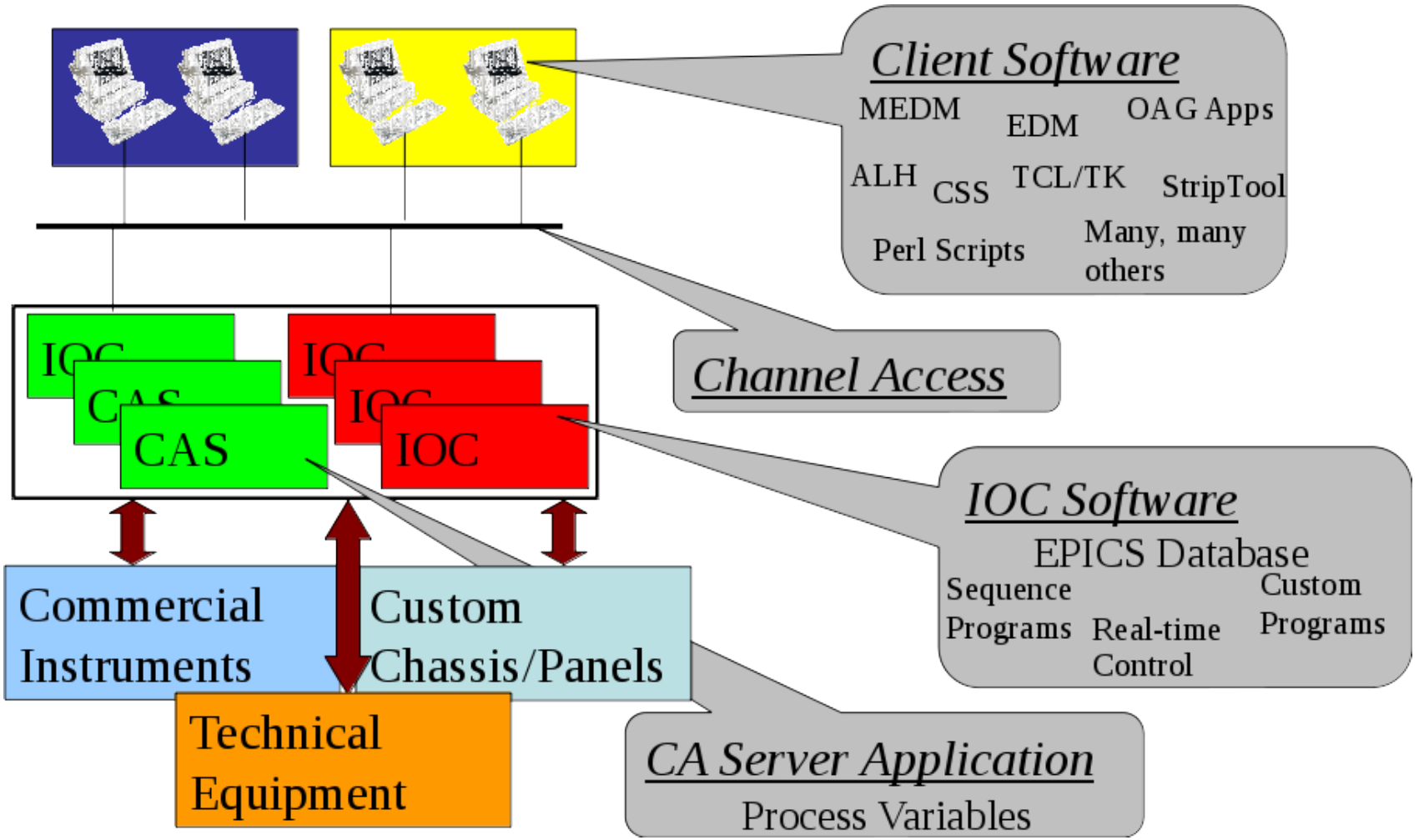
Florian Feldbauer

Ruhr-Universität Bochum - Experimentalphysik I AG

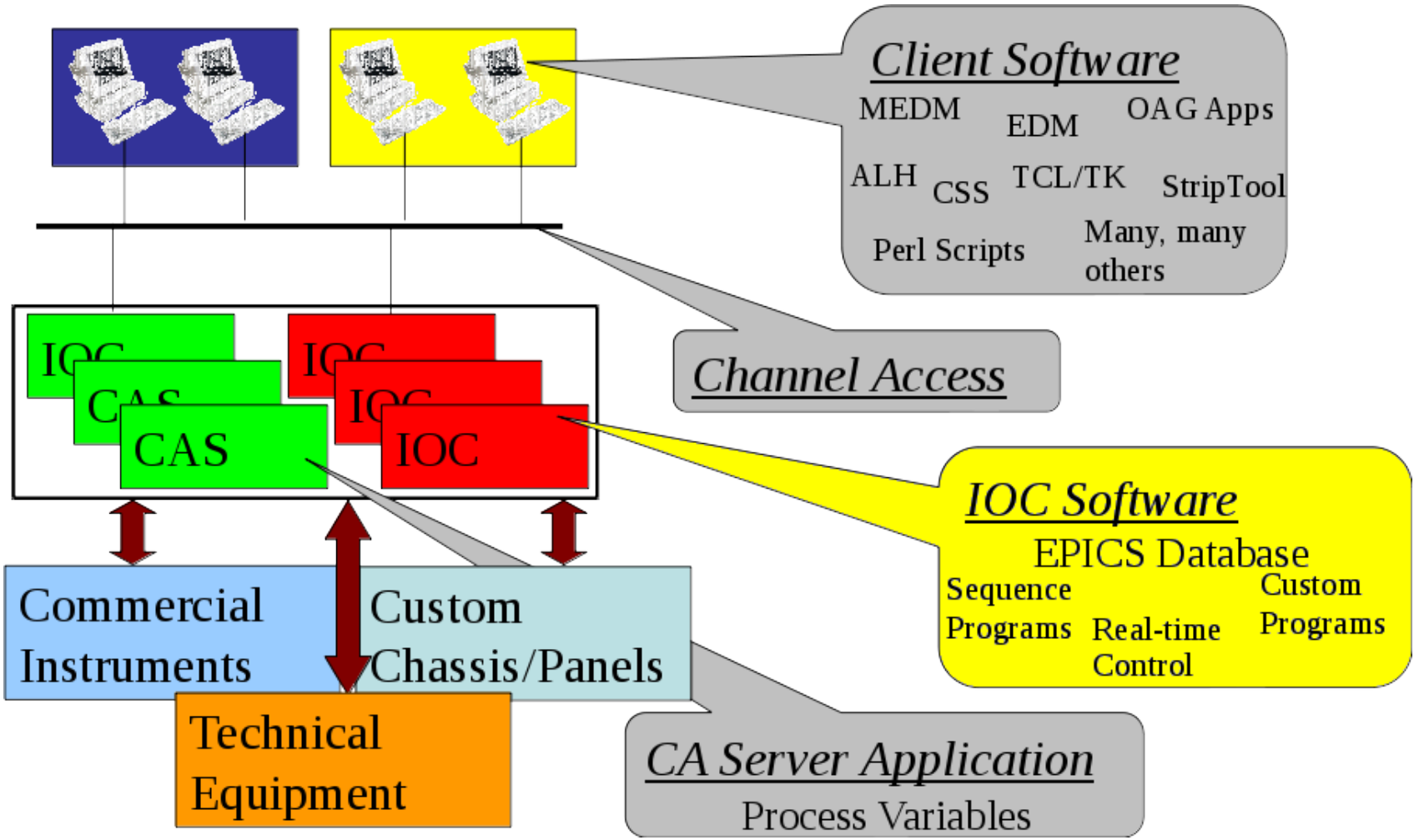
Experimental Physics and Industrial Control System

- EPICS is a decentralized control system architecture
- Network based client/server model (publich/subscribe)
- Channel Acces (CA) protocol connects clients and servers
  UDP and TCP based
- PVaccess protocol (new development introduced in EPICS v4)
  Extension of CA

Canonical Form of an EPICS Control System



Canonical Form of an EPICS Control System

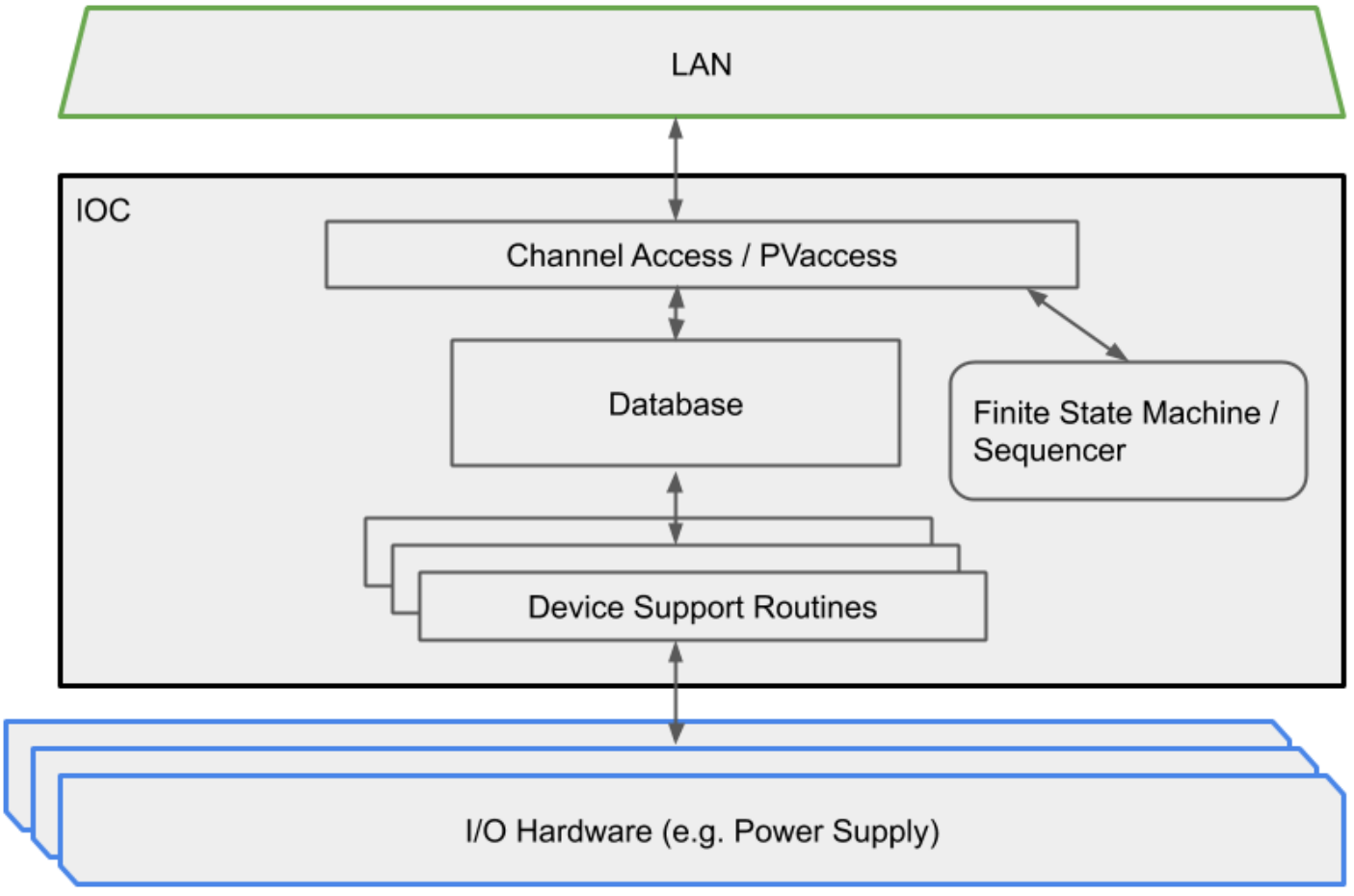


Key Feautures of IOC software

- 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" (PV).

Inside an IOC

The major software components of an IOC (IOC Core)



What are EPICS Records?

- A record is an object with. . .
  - A *unique* name e.g. "PANDA:LMD:MUPIX:HV:P0:H1:G2:Vmom"
  - 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)
  e.g. "PANDA:LMD:MUPIX:HV:P0:H1:G2:Vmom.EGU"
- A Process Variable is a named piece of data (with attributes)
  CA needs the name to access data

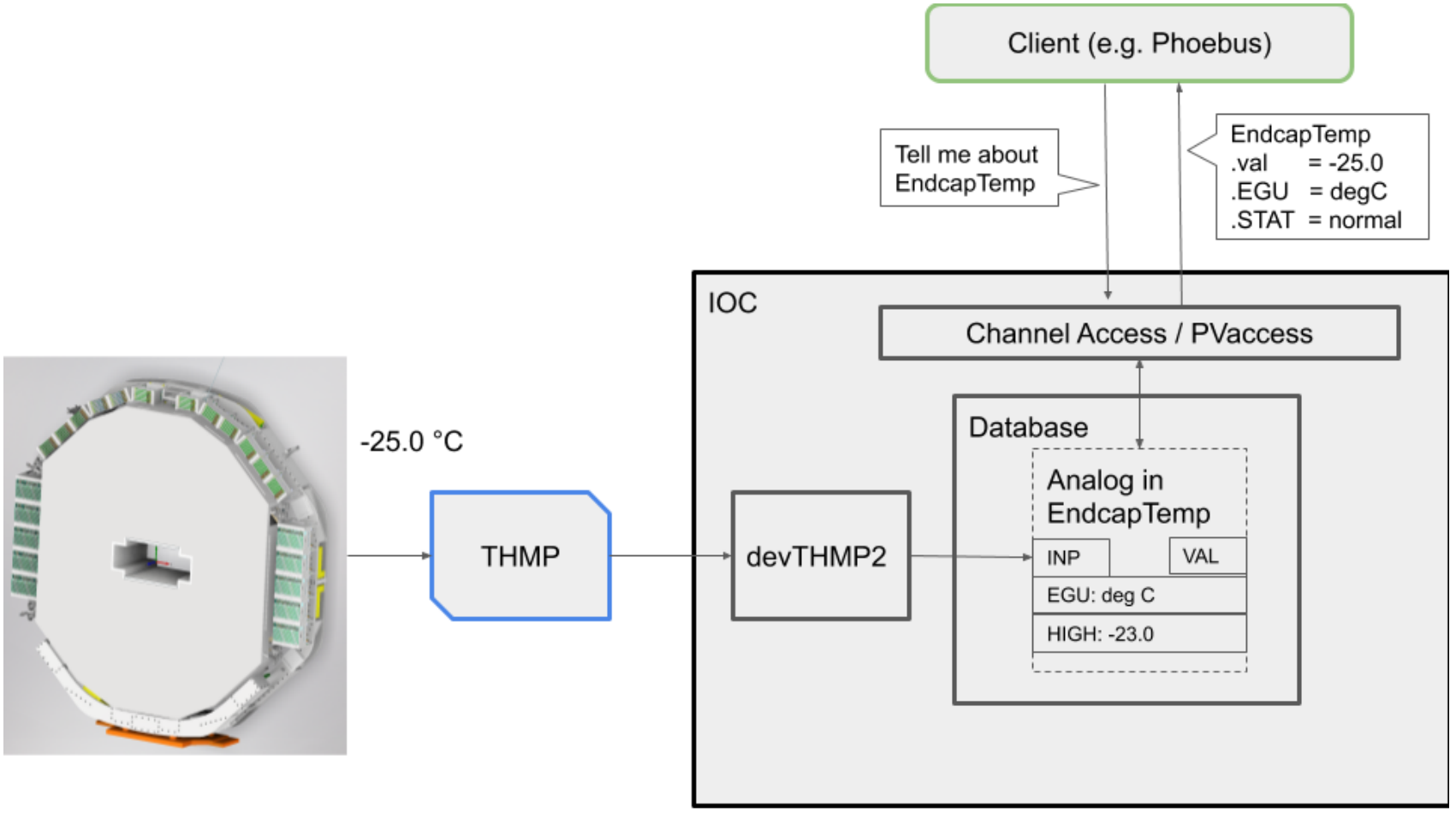
What do records do?

- 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 to other records or to 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 record does nothing until it is *processed*!
- Records can be processed periodically or event driven (Hardware interrupt, CAput, . . .)

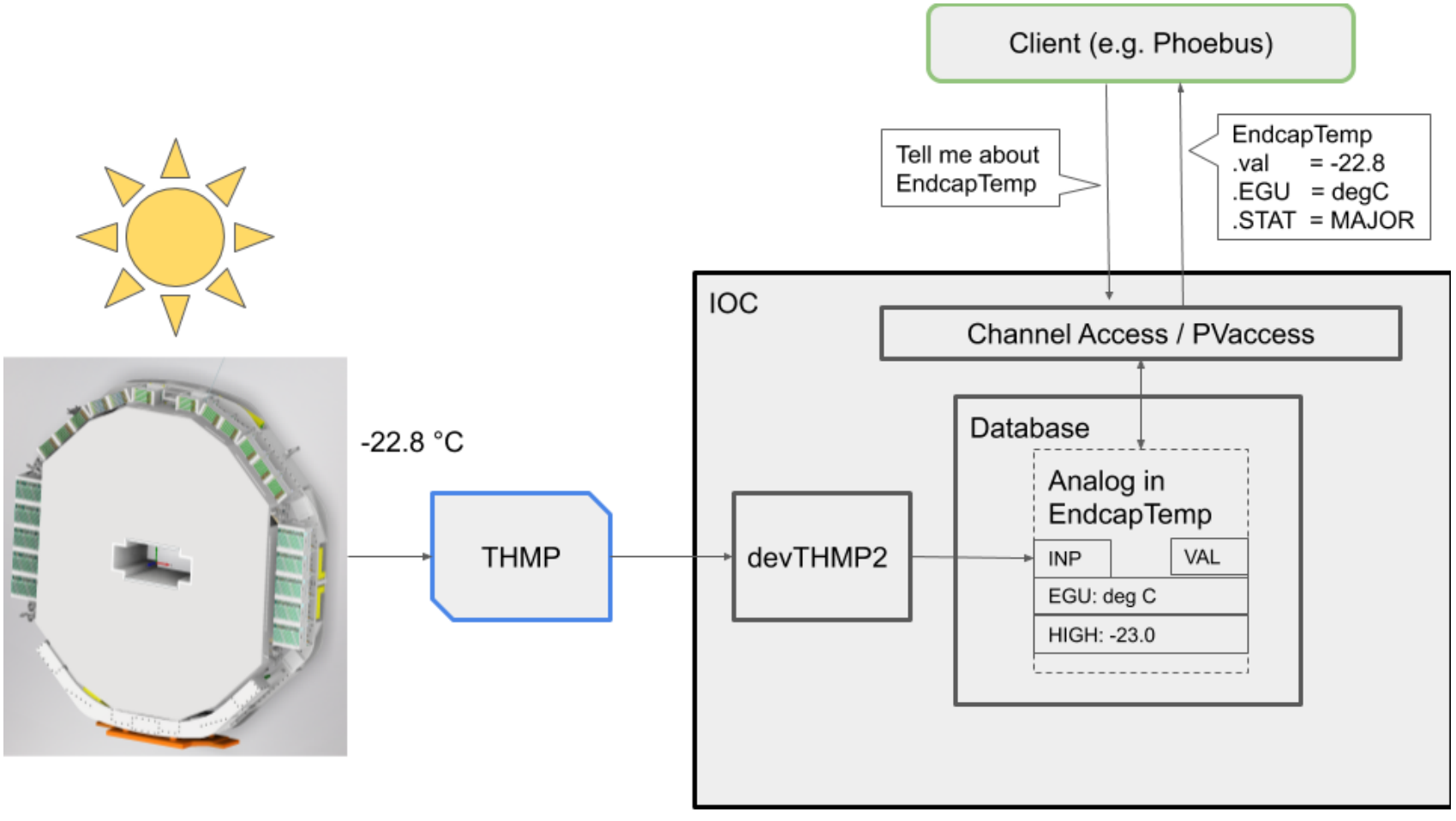
EPICS Database

- 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

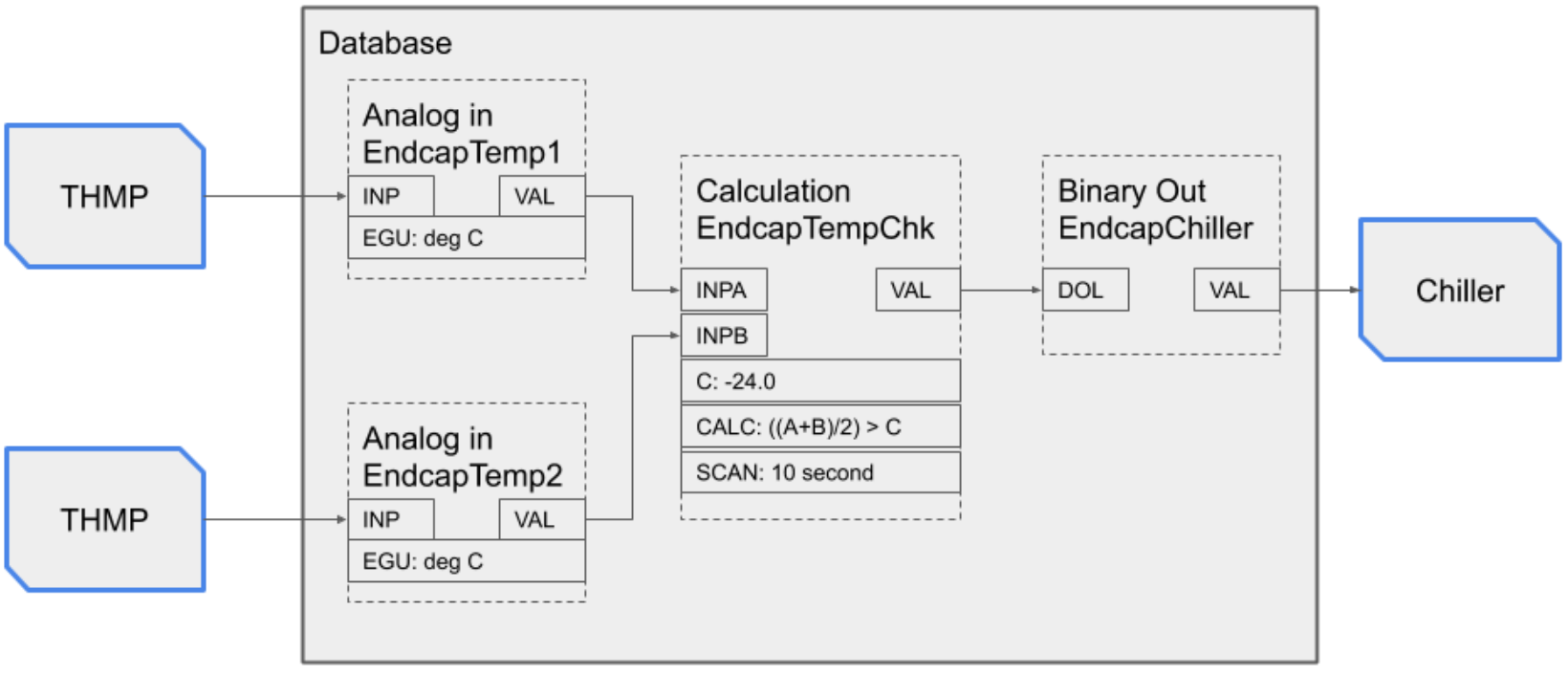
A Simple Database



A Simple Database



Database Processing



Device Support

- Interface between records and hardware
- Tells a record how to read/write data from/to hardware
- For most common protocols, field busses already there (e.g. Seriel communication, SNMP, ModBus, . . .)

- Example:
  - asyn: “low level” driver for serial interfaces (and TCP/IP)
  - streamDevice: “high level” driver for string based communication

Configuration of an IOC

What do we need:

- Database file defining the records
- Eventually protocol file for StreamDevice defining the protocol
- A substitution file
- IOC Start up script

Databases and protocol files are already available for many devices used in PANDA:
`https://panda-repo.gsi.de/pandadcs/epics-files`

Example: Controlling a Hameg HMP4040

Hameg HMP4040 is a 4 channel programmable LV power supply

- Remote control via USB (serial interface) or via TCP/IP
- String based communication

$\Rightarrow$ Control via asyn + StreamDevice

Digression: Docker

"Docker is a set of platform-as-a-service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines."
– Wikipedia

Image: Template containing the software and all its dependencies
Container: runtime version of an image including runtime specific data (configuration, payload data), executing a specific command

Digression: Docker

Get the latest (stable) version of the panda-ioc image:

```
~ > docker pull paluma.rub.de/panda-ioc
```

Start container to see its contents:

```
~ > docker run \    # use image to create and start a container
      --rm     \    # delete container after finishing
      -it      \    # start container with interactive console
      paluma.rub.de/panda-ioc \    # name of image
      /bin/sh      # command to execute inside container
```

Substitution File

This file tells EPICS to load a database template and replace macros with given values.
You can use the same template multiple time (e.g. multiple channels of a power supply)

```
file "<DATABASE TEMPLATE>"
{
  pattern { <COMMA SEPARATED LIST OF MACROS> }
    { <COMMA SEPARATED LIST OF VAULES> }
    { <COMMA SEPARATED LIST OF VAULES> }
    [...]
}
```

Substitution File for Hameg HMP4040

Hameg HMP4040 has 4 channels. Database template defines 1 channel

```
file "/databases/hmp4040.db"
{
  pattern { CHAN, PORT }
    { 0, hmp_1 }
    { 1, hmp_1 }
    { 2, hmp_1 }
    { 3, hmp_1 }
}
```

“/databases/hmp4040.db” will be loaded 4 times, the macro `$(CHAN)` is replaced by 0, 1, 2, and 3, respectively the macro `$(PORT)` is replaced by “hmp_1”
EPICS will not create records with identical name

IOC Start up Script

Template IOC Start up script for our Docker container:

```
#!/pandaIoc

## Set Environment Variables
epicsEnvSet( "STREAM_PROTOCOL_PATH", "/protocols" )

## Register all support components
dbLoadDatabase( "/dbd/pandaIoc.dbd", 0, 0 )
pandaIoc_registerRecordDeviceDriver( pdbbase )

## Define Interfaces
[...]

## Load record instances
dbLoadTemplate( "<SUB FILE>" )

iocInit()
```

Defining Interfaces: Serial Port

Configuring a serial port in EPICS via ASYN:

```
drvAsynSerialPortConfigure( "<NAME>", "<DEVICE>", 0, 0, 0 )
asynSetOption( "<NAME>", 0, "baud", "<50|75|110|134|150|200|300|600|...>" )
asynSetOption( "<NAME>", 0, "bits", "<5|6|7|8>" )
asynSetOption( "<NAME>", 0, "parity", "<none|odd|even>" )
asynSetOption( "<NAME>", 0, "stop", "<1|2>" )
asynSetOption( "<NAME>", 0, "clocal", "<Y|N>" )
asynSetOption( "<NAME>", 0, "crtscts", "<Y|N>" )
asynSetOption( "<NAME>", 0, "ixon", "<Y|N>" )
asynSetOption( "<NAME>", 0, "ixoff", "<Y|N>" )
asynSetOption( "<NAME>", 0, "ixany", "<Y|N>" )
asynSetOption( "<NAME>", 0, "rs485_enable", "<Y|N>" )
asynSetOption( "<NAME>", 0, "rs485_rts_on_send", "<Y|N>" )
asynSetOption( "<NAME>", 0, "rs485_rts_after_send", "<Y|N>" )
asynSetOption( "<NAME>", 0, "rs485_delay_rts_before_send", "<msec_dly>" )
asynSetOption( "<NAME>", 0, "rs485_delay_rts_after_send", "<msec_dly>" )
```

Defining Interfaces

Configuring a TCP/IP or UDP/IP port in EPICS via ASYN:

```
drvAsynIPPortConfigure( "<NAME>", "<HOST INFO>", 0, 0, 0 )
# HOST INFO is: <host>:<port>[:localport] [protocol]
# e.g. "192.168.0.5:5025 TCP"
```

Configuring an I2C port

```
drvAsynI2CConfigure( "<NAME>", "<I2Cbus>", 1 )
```

Configuring THMP/LedPulser

```
THMPConnect( "<CAN INTERFACE>", "<CAN ID IN HEX>" )
LedPulserConnect( "<CAN INTERFACE>", "<CAN ID IN HEX>" )
```

Digression: Docker

Starting our panda ioc

```
~ > docker run \   # use image to create and start a container
     -d       \   # detach from current prompt (run in background)
     -it      \   # start container with interactive console
     -v ~/epics-tut:/config \   # mount ~/epics-tut (host) at /config (container)
     --device /dev/ttyAMA0 \    # make serial port accessable
     --group-add 20 \           # add user to group dialout (GID 20) from host
     paluma.rub.de/panda-ioc \  # name of image
     ./st.cmd     # command to execute inside container
```

Adding new Device to EPICS

How to add control for a new device?

- What kind of interface/protocol does the device provide?
  - $\Rightarrow$ Which device support routine is needed?
- Needed device support already in panda-ioc?
  - Y: Write a database (and protocol) file
  - N: Ask EPICS Community if a device support is available or write your own, then write a database file

Device Support

Currently we use these Device Support Modules:

- *asyn*: “low level” interface to serial ports, and network sockets (TCP, UDP)
- *modbus*: “high level” interface for modbus protocols (based on asyn)
- *devSnmp*: interface for SNMP
- *StreamDevice*: “high level” interface for string based communications
- *drvAsynI2C*: “low level” interface to I2C bus
- *devGpio*: Interface to GPIOs (e.g. RasPi or BeagleBone Black)
- *devThmp2*: Interface for the THMP
- *devLedPulser2*: Interface for the PANDA-EMC Led Pulser

Usefull links

- Information about Docker images:
  `https://paluma.ruhr-uni-bochum.de/wiki/index.php/Container_Virtualization_for_PANDA_DCS`
- Record Reference Manual:
  `https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14`
- Collection of some device support modules:
  `https://github.com/epics-modules`