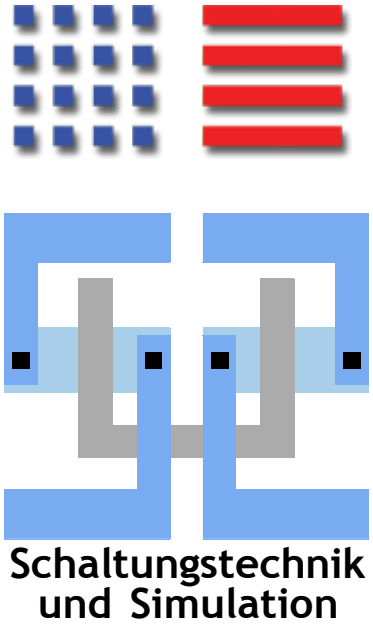
UNIVERSITÄT
HEIDELBERG

# VXD Slow-Control




Michael Ritzert

michael.ritzert@ziti.uni-heidelberg.de

7th Belle II VXD Workshop

Prague

21.01.2015

# Overview

- PXD Slow-Control coordinated by Michael Ritzert.
- SVD Slow-Control coordinated by Szymon Bacher.

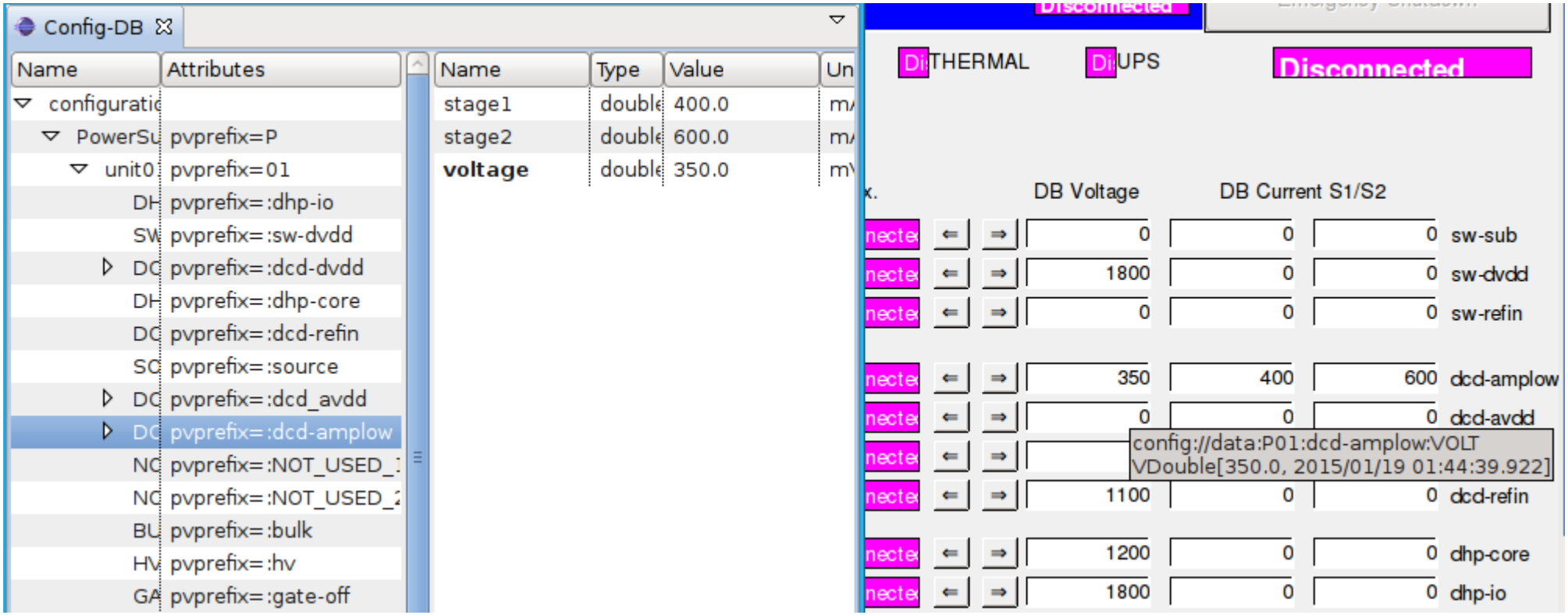
- Shared systems
  - IBBelle $CO_2$ cooling plant
  - FOS environmental sensors

- Coordinated efforts in all areas of the slow-control.
  - EPICS, CSS
  - Databases
  - PV naming / GUI conventions
  - Alarm & interlock systems

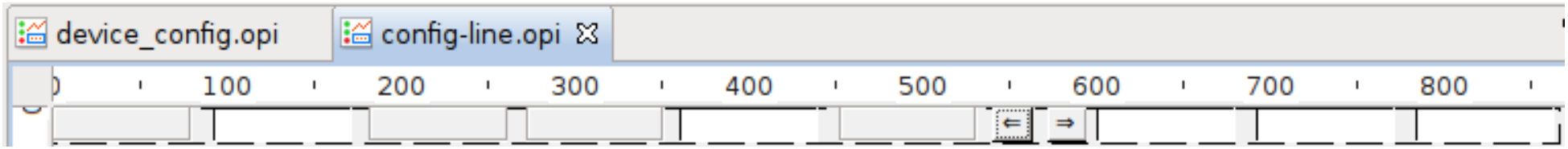
# 2014 in Brief

- DESY testbeam
  Big success with all deployed SC systems.
- UNICOS
  Ready to work with the actual IBBelle UNICOS project.
- EPICS and CSS binary repositories
- SC unification
  “The common GUI for Belle II is CSS.”
- Interface to SuperKEKB EPICS established.
- Productive use of the LMU PS
- Interlock
  Prevents dangerous settings from reaching the PS.
- PXD power-up sequence
  Replaces tedious manual operation.

Configuration visible as PVs to display and modify database entries.




Implementation note: Use “Linking Containers” with macros for repetitive GUI items.

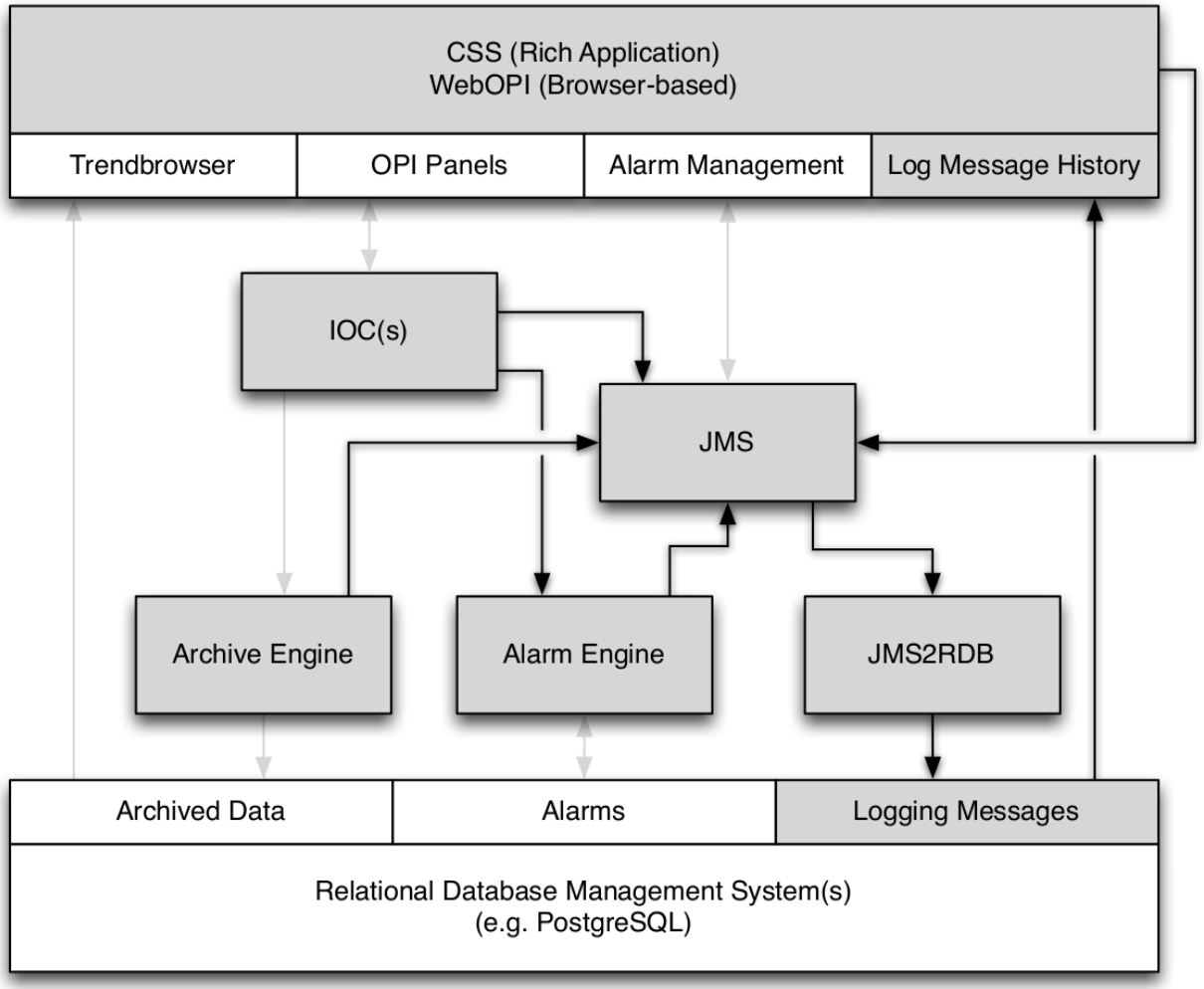


# Physics Configuration

1. The run type is set in the Master RC.
2. We look up the configuration id to load for this run type.
   - Previously configured as run type ⇒ configuration id pairs.
3. The id is stored in the conditions database.
4. The configuration is loaded and used by the power-up sequence.
5. Any deviation from the configured values is raised as an alarm.

# Message Log

- C++ library that connects to the ActiveMQ service in the same way as CSS applications.
  - Can be used from IOCs.
    ⇒ All log messages in one common database accessible from CSS.



# Message Log II

- Showing mixed C++ and Java log messages from the database:

| TIME | TEXT | NAME | SEVERITY | CREATETIME | USER | APPLICATION-ID | CLASS |
|---|---|---|---|---|---|---|---|
| 2015/01/05 | Internal browser is not a | logging | WARNING | 2015-01-05 20:06:39.04 | ritzert | CSS | org.csstudio.logging.PluginLogListener |
| 2015/01/05 | WARNING: Prevented re | logging | SEVERE | 2015-01-05 16:04:41.88 | ritzert | CSS | org.csstudio.logging.PluginLogListener |
| 2015/01/05 | Operation details | logging | SEVERE | 2015-01-05 15:53:53.50 | ritzert | CSS | org.csstudio.logging.PluginLogListener |
| 2015/01/05 | Operation details | logging | SEVERE | 2015-01-05 15:53:53.50 | ritzert | CSS | org.csstudio.logging.PluginLogListener |
| 2015/01/05 | SIGNAL 11 received. | static void SuS::lo | FATAL | 2015-01-05 15:50:59.37 | fecmess | app | logger |
| 2015/01/05 | final | int main() | WARNING | 2015-01-05 15:50:59.37 | fecmess | app | example |
| 2015/01/05 | info: 9 | int main() | INFO | 2015-01-05 15:50:59.37 | fecmess | app | example |
| 2015/01/05 | 9 | int main() | DEBUG | 2015-01-05 15:50:59.37 | fecmess | app | example |

- Features of the C++ logging classes:
  - Output to stdout, log files (automatically rotated), STOMP (JMS protocol)
  - Multi-threaded: application thread continues while the messages are sent.
  - Automatic retry when log server is unavailable.
  - runtime configurable from EPICS IOC shell
  - Logs backtrace on crashes.

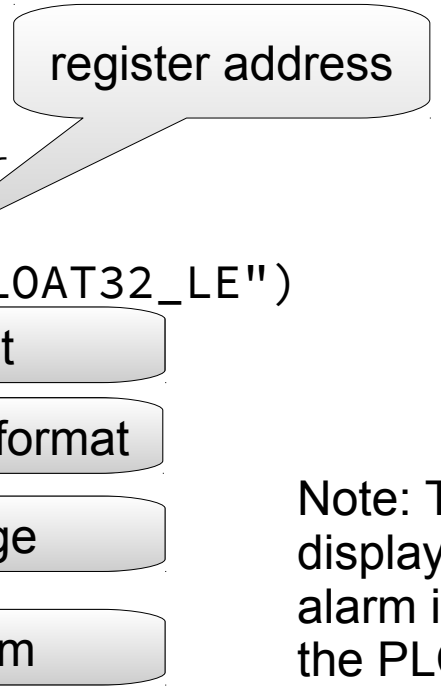
# UNICOS Porting to EPICS

- Reminder: We need to implement the supervision layer (operator interface). The actual plant control logic needs not be changed.
- UNICOS can now create output for EPICS
  - EPICS database for the modbus IOC. (4298 PVs for IBBelle Plant A.)
  - Template OPI with all devices.
- Implementation of the “faceplates” in CSS has progressed well.
- Requires rigorous testing and comparisons with WinCC OA (“PVSS”) behavior!
  - The UNICOS documentation is incomplete in parts.
- Ready to run with the actual hardware!
- Still to do:
  - generate Archiver configuration.
  - generate device tree views
  - same approach as for EPICS database and OPI template output can be used.

# UNICOS ⇒ EPICS DB

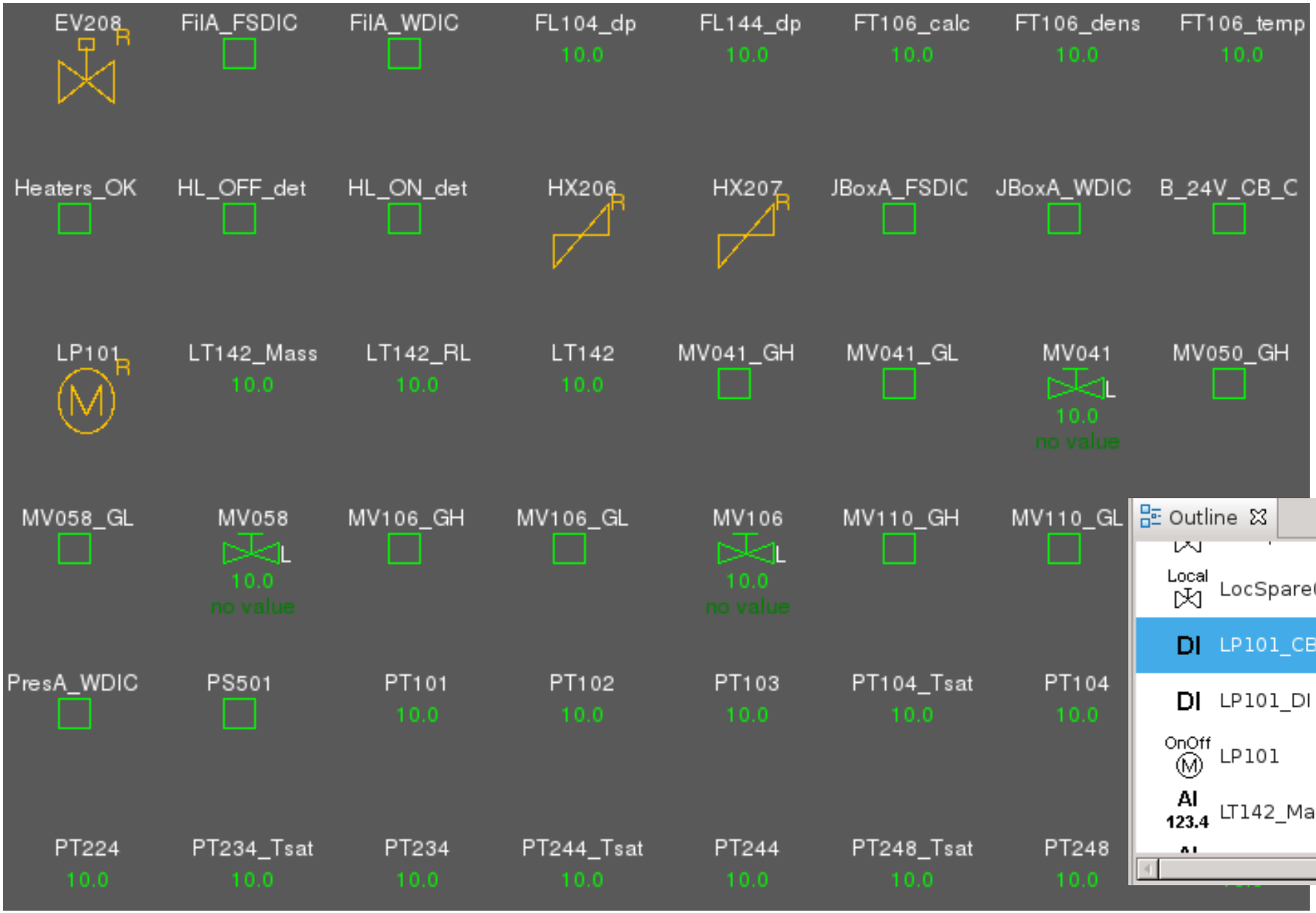
- Python script started (one time) by the UNICOS converter GUI.
- Outputs an EPICS database with hardware addresses to be used by the modbus IOC.
- Tries to convert all metadata
  - Alarm levels
  - Units
  - Number formats
- Sample output:

```
record(ai, "CSMB_WP1_PT142:PosSt") {
    field(DTYP, "asynFloat64")
    field(INP,  "@asyn(block1192 4)FLOAT32_LE")
    field(SCAN, "I/O Intr")
    field(EGU,  "bar")
    field(PREC, "1")
    field(LOPR, "0.0")
    field(HOPR, "110.0")
    field(HIGH, "55.0")
}
```



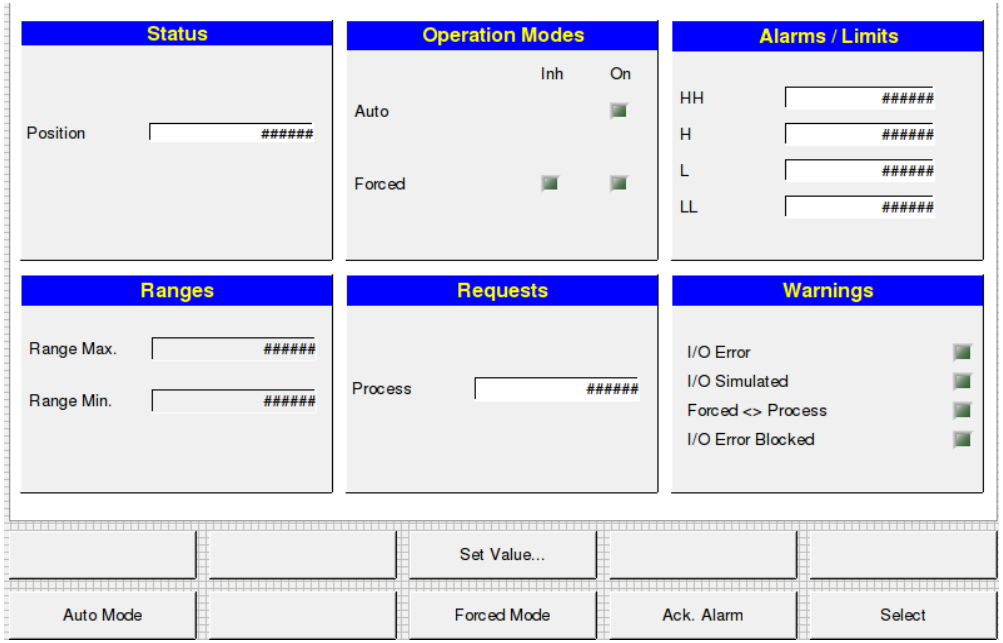

Note: The alarm limit is for display purposes only. The alarm is generated within the PLC.

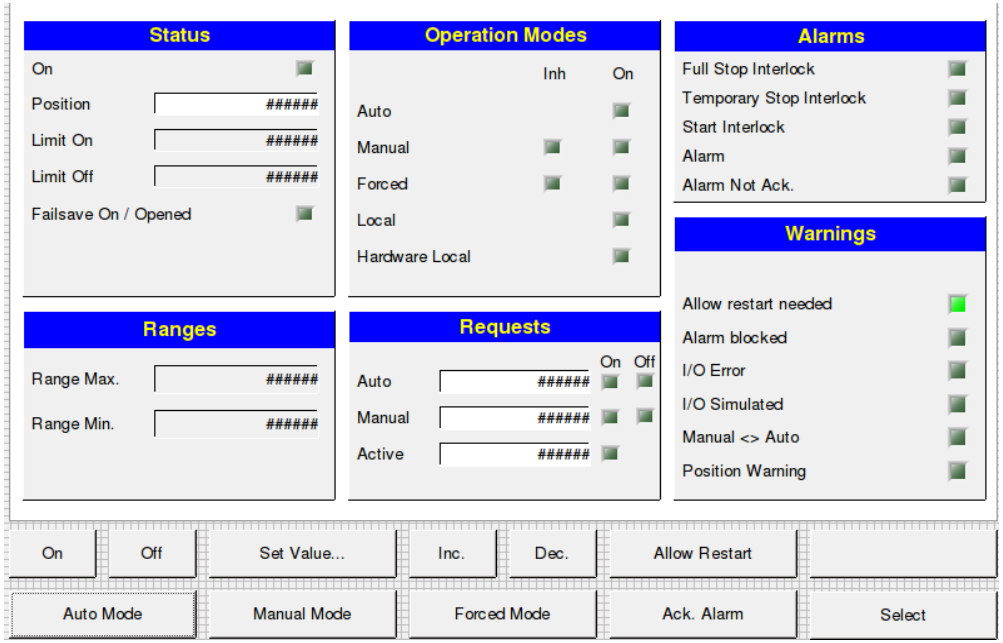
# Generated OPI template: All widgets in one OPI to be copied.

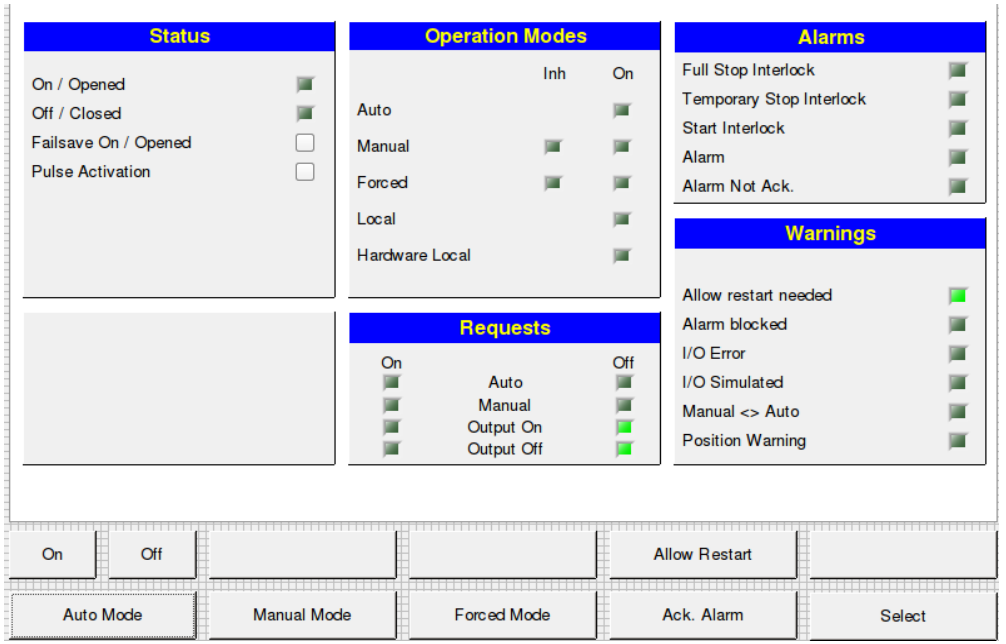


# UNICOS Faceplates

- Dialogs for seeing details about and manipulating the devices.
- Linked from each device widget.



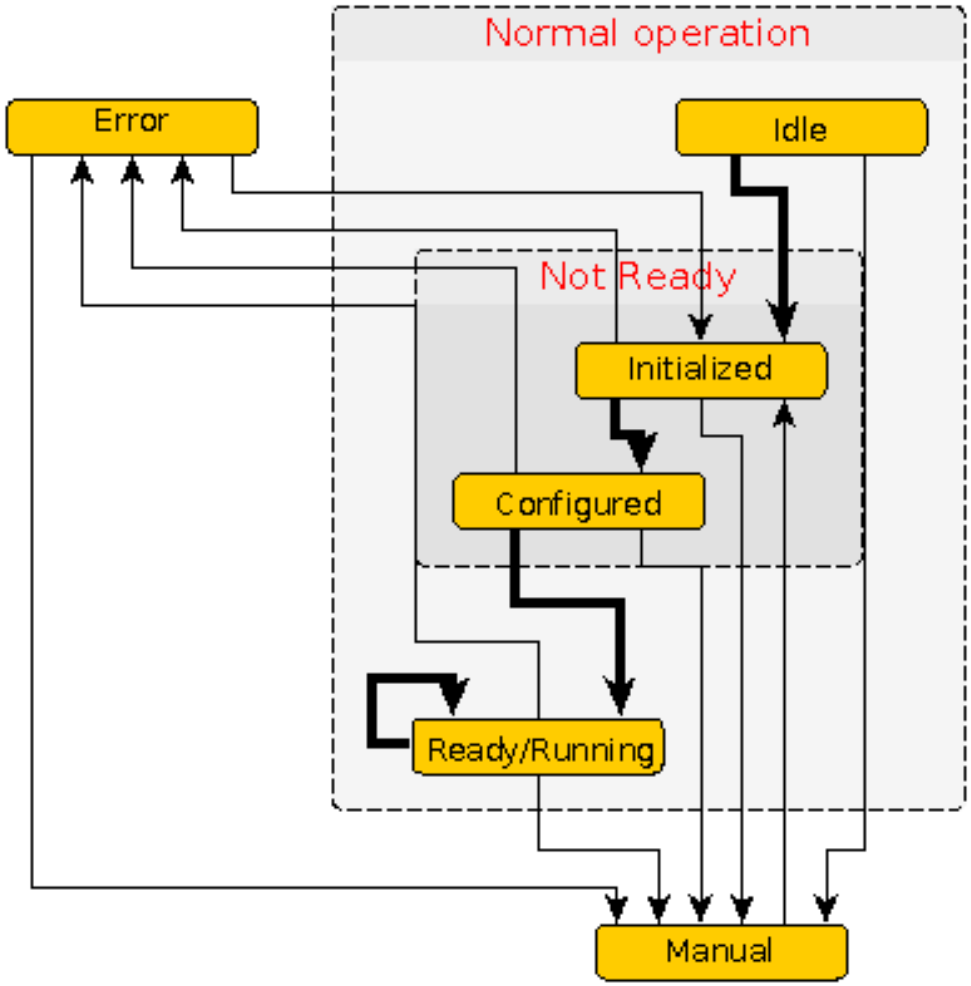



# SVD Slow-Control Recent Activities

- State machines: First version of state machines is being tested.
- Power supplies: IOC for PS has been prepared, and is being documented.
- FADC IOC: C++/EPICS interface on the way. First version of C wrapper was examined.
- Some more data can be found on Slow Control Twiki, and during SVD Software panel.

# State Machine Concept




Startup steps can be divided into two groups:

- Things to do before low voltage is on
- Things to do after low voltage is on

eg: setting up APV must happen after ramping LV.

because of that, Nakao-san's "NotReady" step is subdivided into two steps:

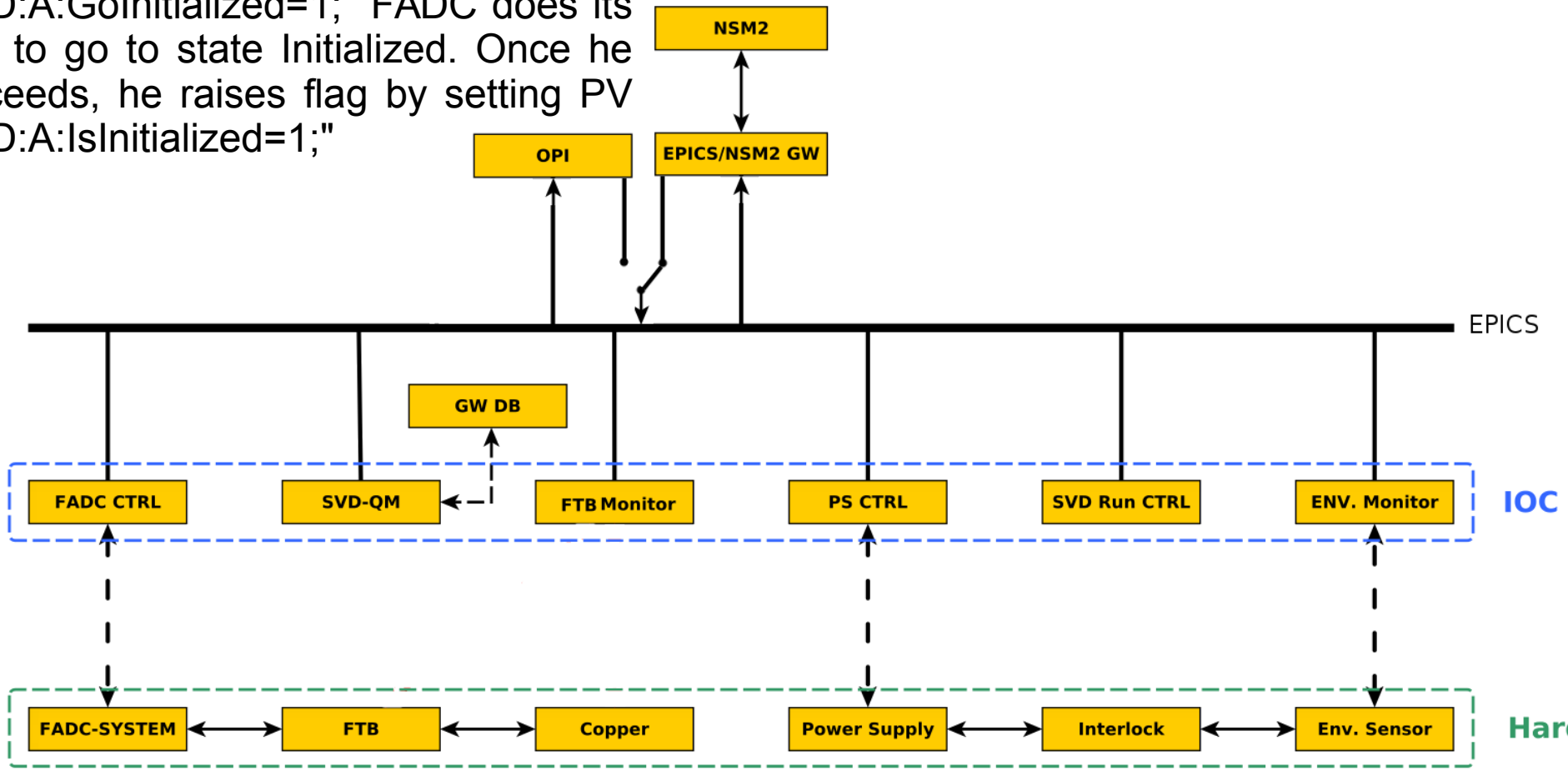
- Initialized
- Configured

Condition for state transition Init->Config is “Power Supply State” is at least “Standby”. If “Off” is issued afterwards, we go to “Error”

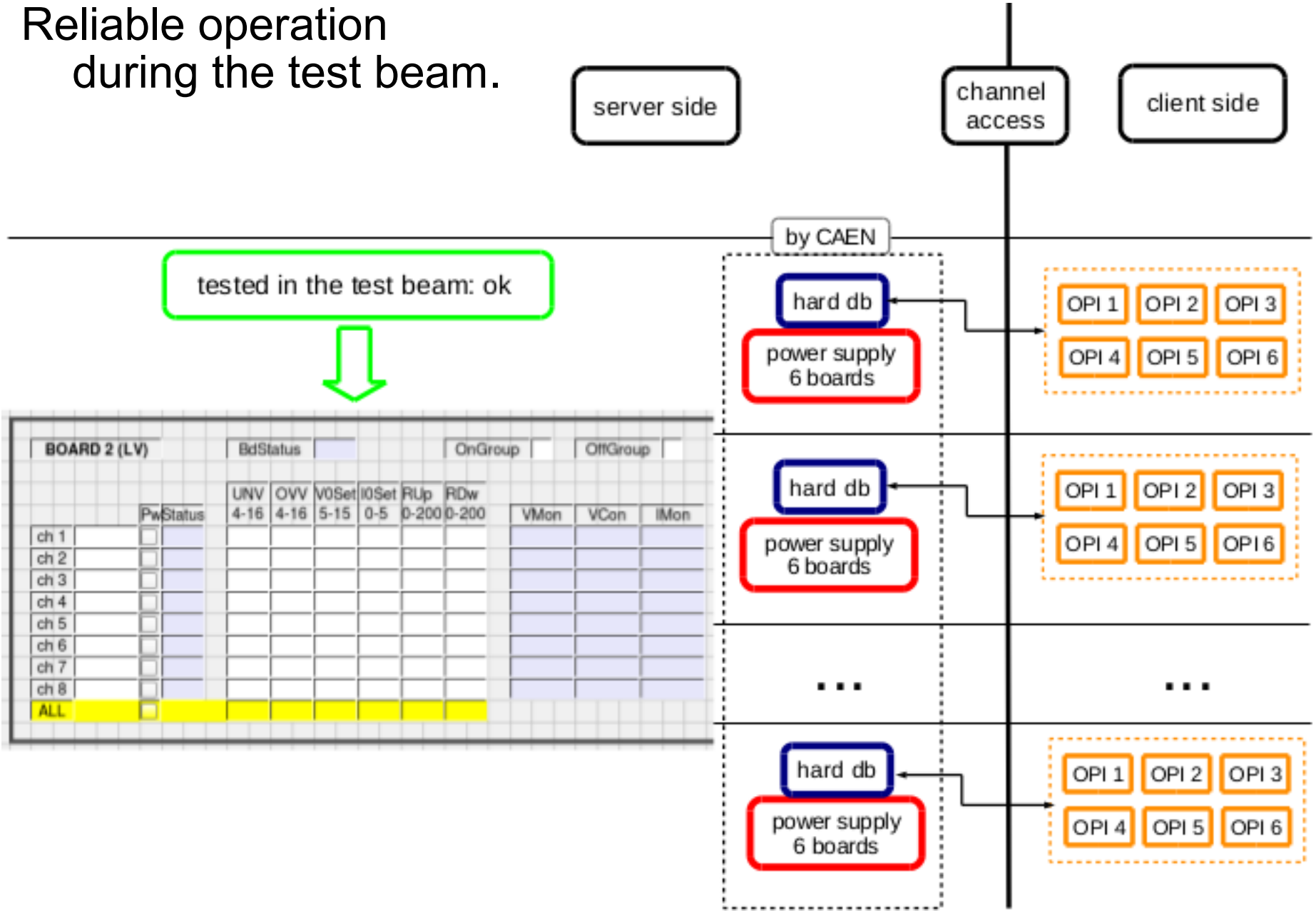
# Hierarchical Approach

All IOCs have their own state machine, with identical states.
For IOCs other then SVDRunCTRL state transitions are dependent of flags raised by SVDRunCtrl.
eg: when SVDRunCtrl does "SVD:A:GoInitialized=1;" FADC does its best to go to state Initialized. Once he succeeds, he raises flag by setting PV "SVD:A:IsInitialized=1;"

If any IOC is stuck as some step (eg. PS waits for “Standby”, whole system will wait. Also, every IOC can issue “GoToError” for others.



- Reliable operation
  during the test beam.

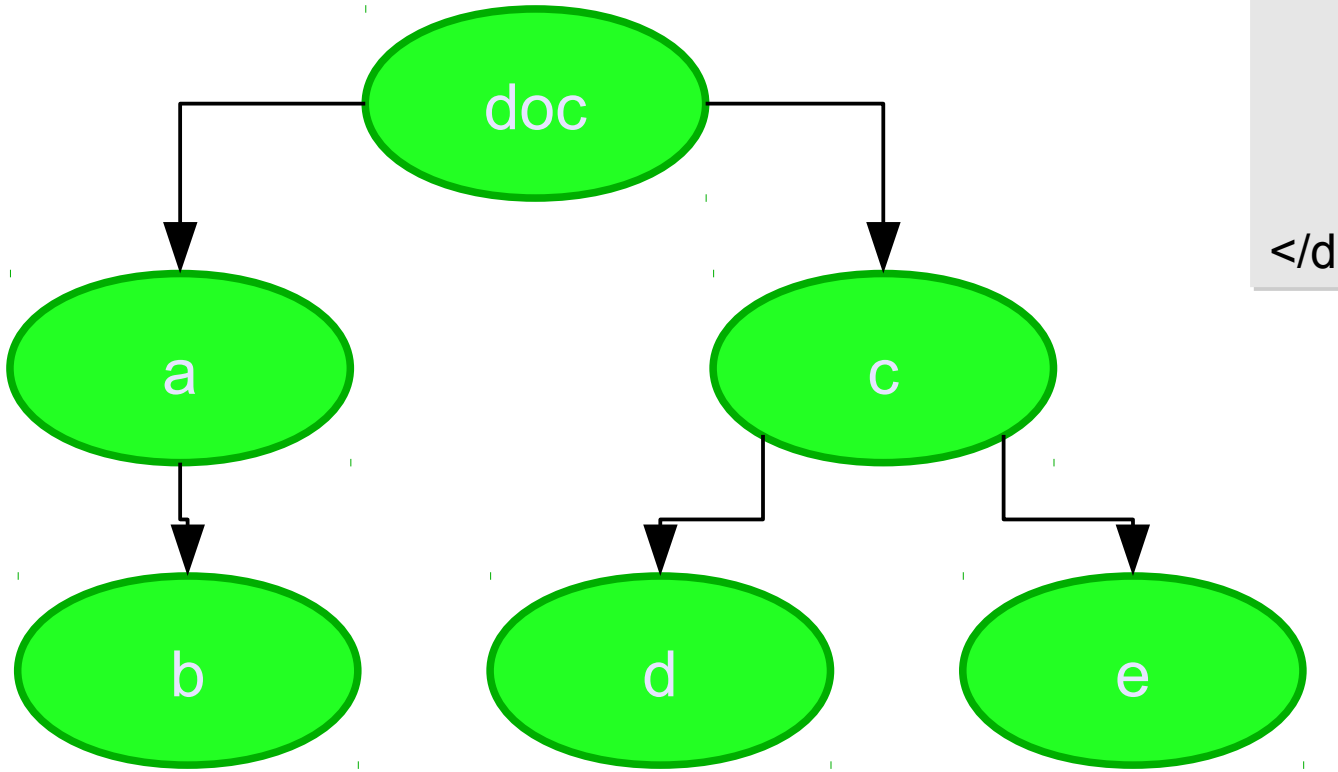


# Next Steps

- Include all environment sensors.
  - Update FOS IOC for the readout with multiplexer. To be tested at DESY for mockup tests.
  - NTC and humidity sensor readout.
  - Check what sensors can provide input to environmental interlocks.
- Working group for GUI (aka OPI) guidelines.
- EPICS on ONSEN and DATCON
  - FPGA: IOC on embedded PowerPC (or IPBus for DATCON)
  - Hardware: IPMI
- Investigate IPMI support of COPPER crates.
- DQM (from ONSEN data).
- Prepare the control rooms (@KEK + remote).

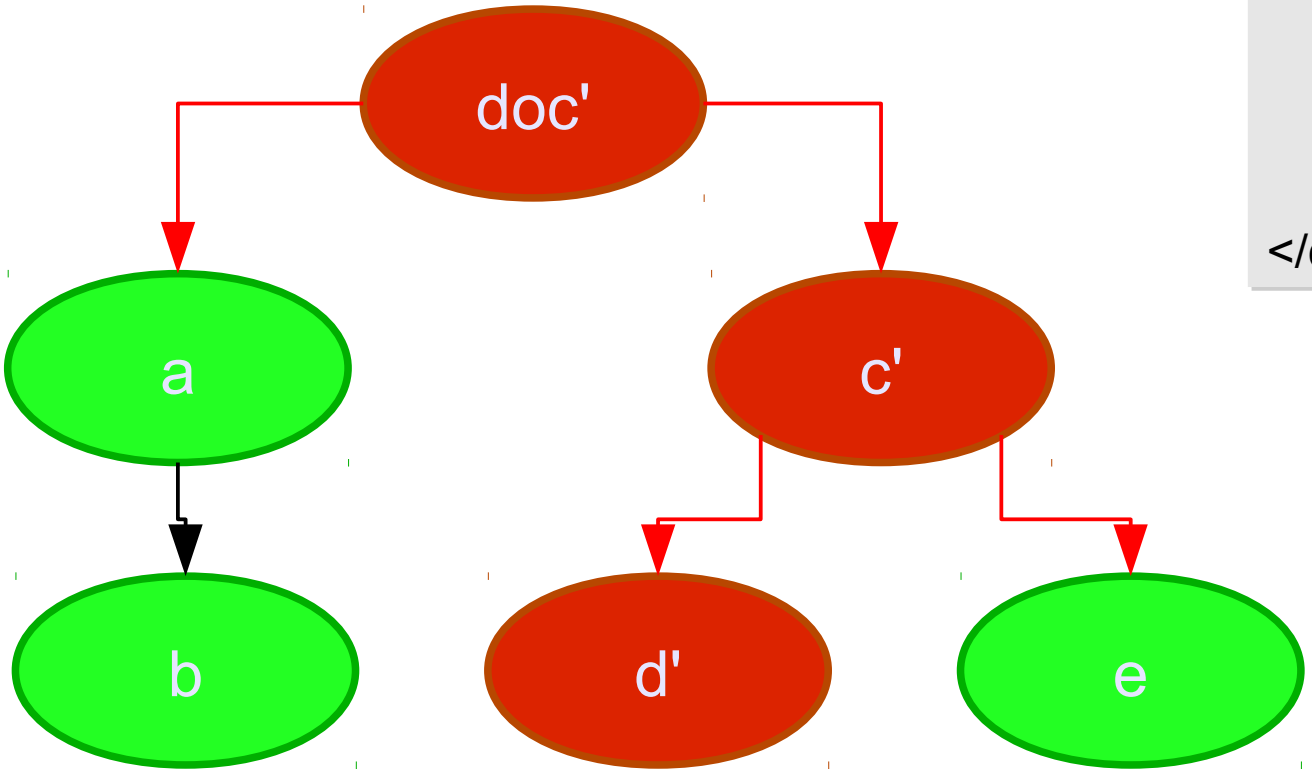
Thank you!

# Data Structure in the Database




```
<doc>
    <a>
        <b>12</b>
    </a>
    <c>
        <d>42</d>
        <e>hello</e>
    </c>
</doc>
```

# Updates




```
<doc>
    <a>
        <b>12</b>
    </a>
    <c>
        <d>44</d>
        <e>hello</e>
    </c>
</doc>
```

updated data