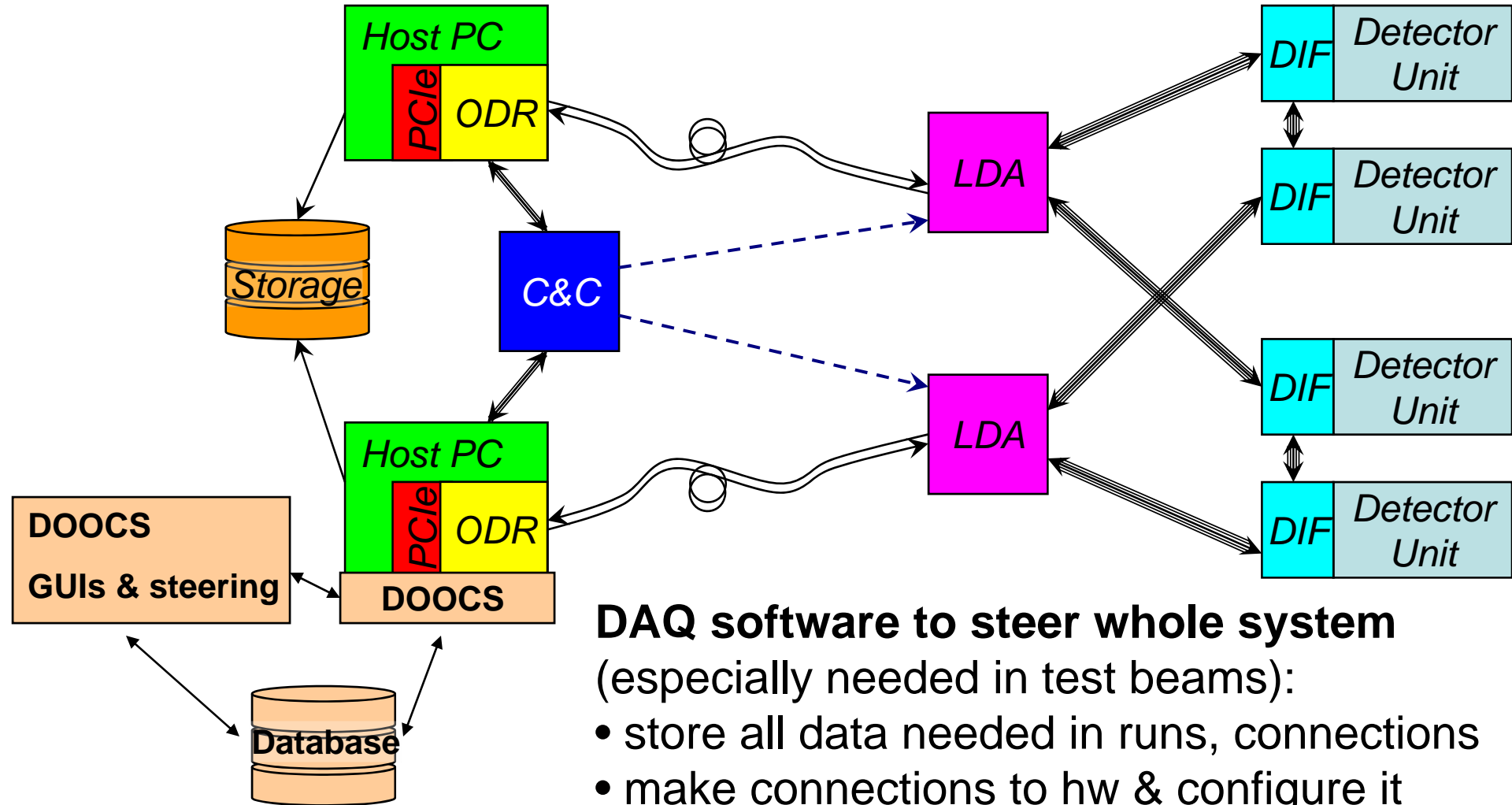


Development of the DAQ software for the technical prototype:

Status & Outlook

Valeria Bartsch UCL

Overview over the task

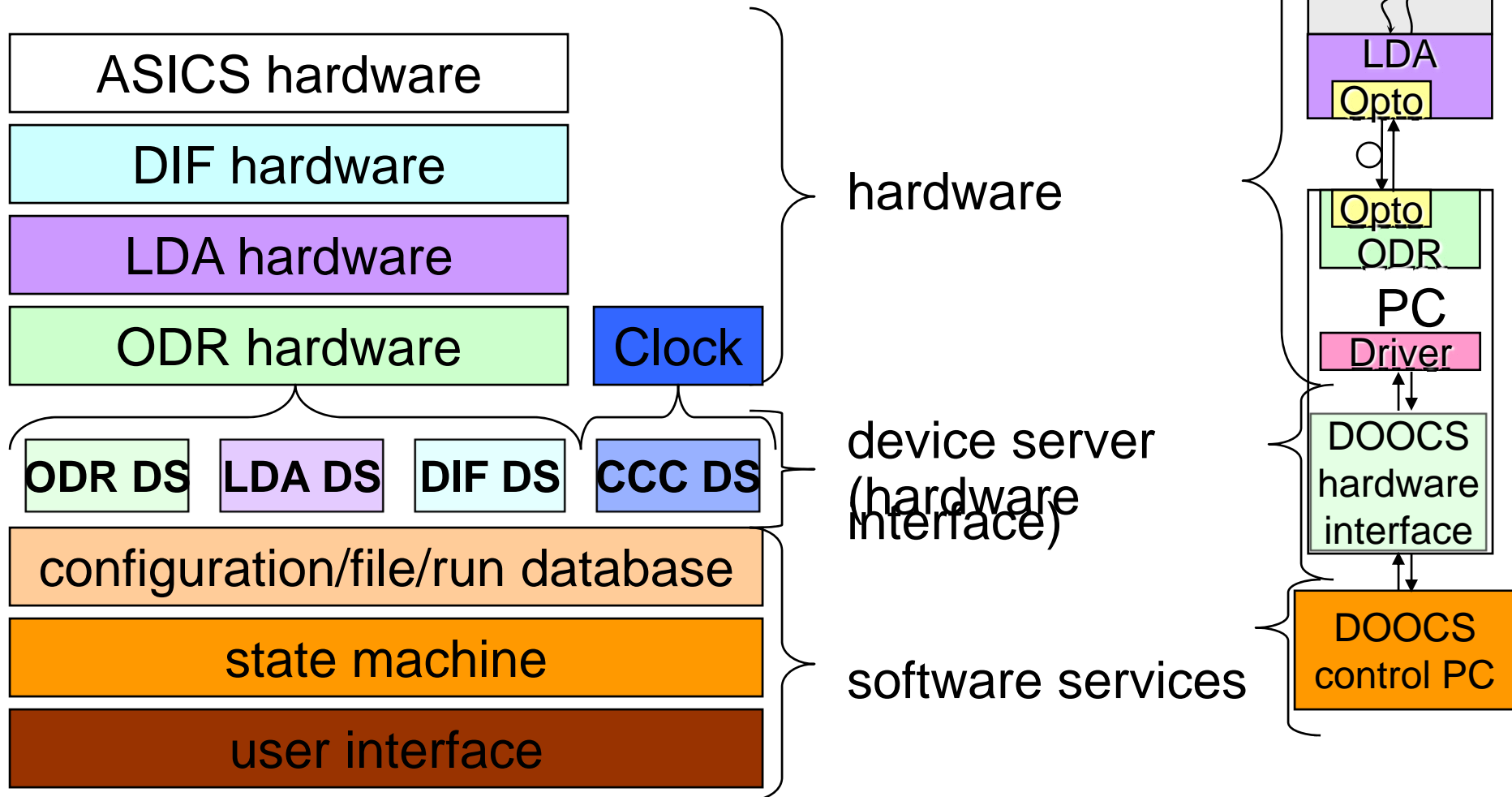


DAQ software to steer whole system

(especially needed in test beams):

- store all data needed in runs, connections
- make connections to hw & configure it
- control & change the state of the hw
- deliver interfaces for experts & shifters

DAQ software overview



DAQ software overview

- development started on almost all of the main software components
- emulator for the LDA, DIF development not yet started
- user interface, ODR interface ready
- database and state machine started

ODR DS

LDA DS

DIF DS

CCC DS

configuration/file/run database

state machine

user interface

DAQ software new developments

New developments since meeting at DESY in March:

State machine

DAQ software state machine

- state machine in “source/daq/fsm” of the DOOCS CVS repository adapted to our needs (changes not yet committed to the CVS directory)
- able to drive the state machine defined
- functionality for the ODR added (LDA and DIF are not testable in a system yet)
- at the moment state machine drives exactly one device, collector (also to be found in “source/daq/fsm” has to be tested yet)
- functionality for LDA and DIF can be easily added

Finite State Machine

- transition can be steered by GUI
- another property DAQ_FSM_TR_STS shows if the transition was successful

The screenshot shows a terminal window with the DOOCS Communication and Plot Utility GUI overlaid. The GUI has several red boxes highlighting key elements:

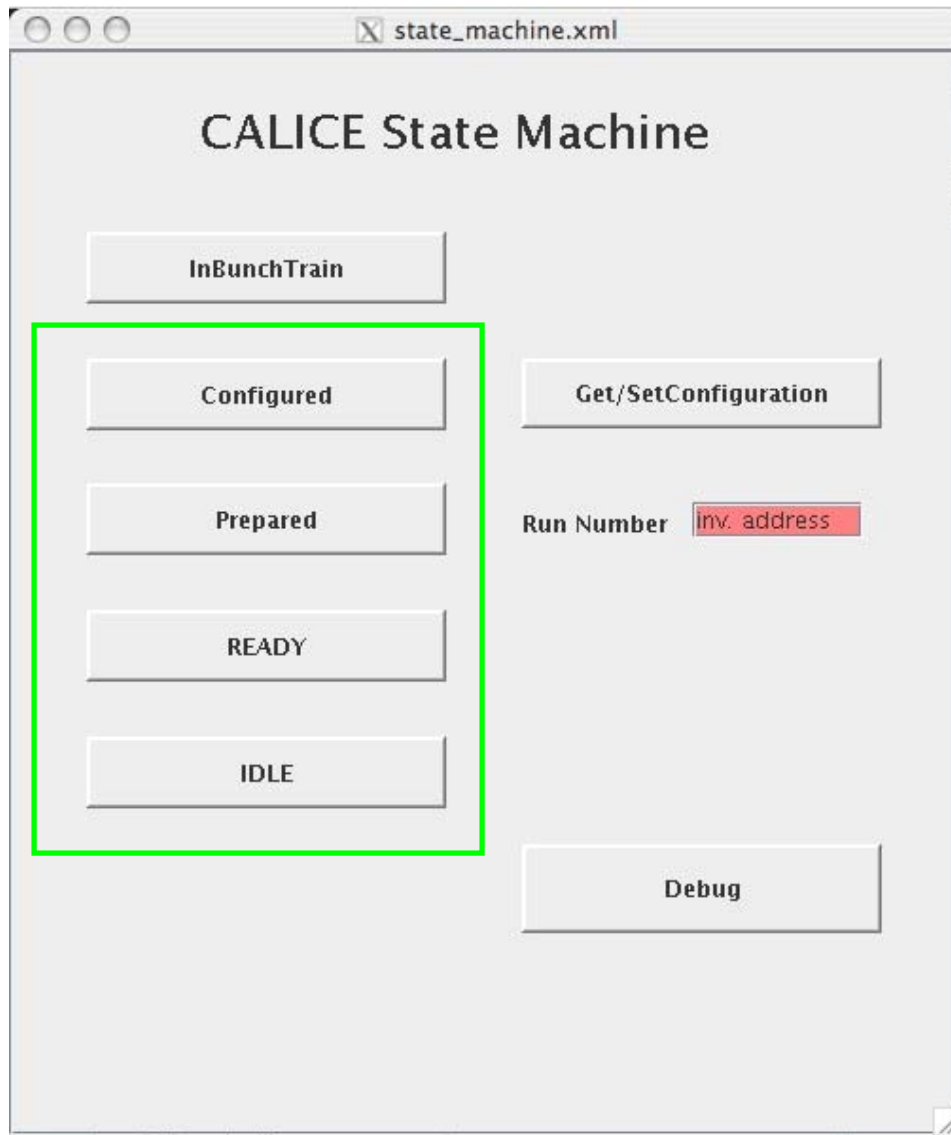
- A box around the 'Property' list in the 'DAQ_FSM_CMD_C' section, showing properties like 'LAST_UPDATE', 'UPDATE_THREAD', 'DEVICE.INFO', 'DAQ_FSM_CMD_C', 'DAQ_FSM_CMD_A', 'DAQ_FSM_STATE_A', 'DAQ_FSM_TR_STS', and 'PORT_NUM'.
- A box around the 'Send' button in the 'Result: 1' section.
- A box around the 'Send Data: 1' field in the 'Send Data: 1' section.

The terminal output below the GUI shows the following sequence of events:

```
Segmentation fault
bartsch>
bartsch>
bartsch>
bartsch>
bartsch> /unix/lc/bartsch/doocs_nw/Linux/obj/server/calice/odr/odr_server
ODR_Server Server-4.9.15 17:14.09 30.04.2009 -> working dir. </unix/lc/bartsch/doocs_nw/source/ser
/server/calice/o
ver/calice/odr>
ODR_Server odr_server.conf 17:14.09 30.04.2009 -> no hard error
ODR_Server HashIt 17:14.09 30.04.2009 -> DATARA.DESC ion of channel property already exist
s
ODR_Server HashIt 17:14.09 30.04.2009 -> DATASZ.DESC ion of channel property already exist
s
State 1: INITIALIZED
ODR_Server Server-4.9.15 17:14.09 30.04.2009 -> System restart.
ODR_Server Server-4.9.15 17:14.09 30.04.2009 -> Expert uid/gid are missing
-D_REENTRANT -I ODR_Server Server-4.9.15 17:14.09 30.04.2009 -> started with 2 devices on caliceut2.hep.ucl.ac.uk
/lib/include -I/ ODR_Server Server-4.9.15 17:14.09 30.04.2009 -> Env variable ALARMSVR not set: using TEST.UTIL
inux/lib/includ ODR_Server Server-4.9.15 17:14.09 30.04.2009 -> Env variable ALARMSVR not set: using TEST.UTIL
llR6/lib -L/unix State 1: INITIALIZED
ix/lc/bartsch/d ODR1 Transition OK state
inux/obj/server/
FSM -lEqServer -ldoocsapi -lm -linsr -lpthread -lrc -L/unix/lc/products/lc_installation/root/root/7/110 -lcore -lclnt -lrcv
ad -lTree -lRint -lPostscript -lMatrix -lPhysics -pthread -lm -ldl -rdynamic

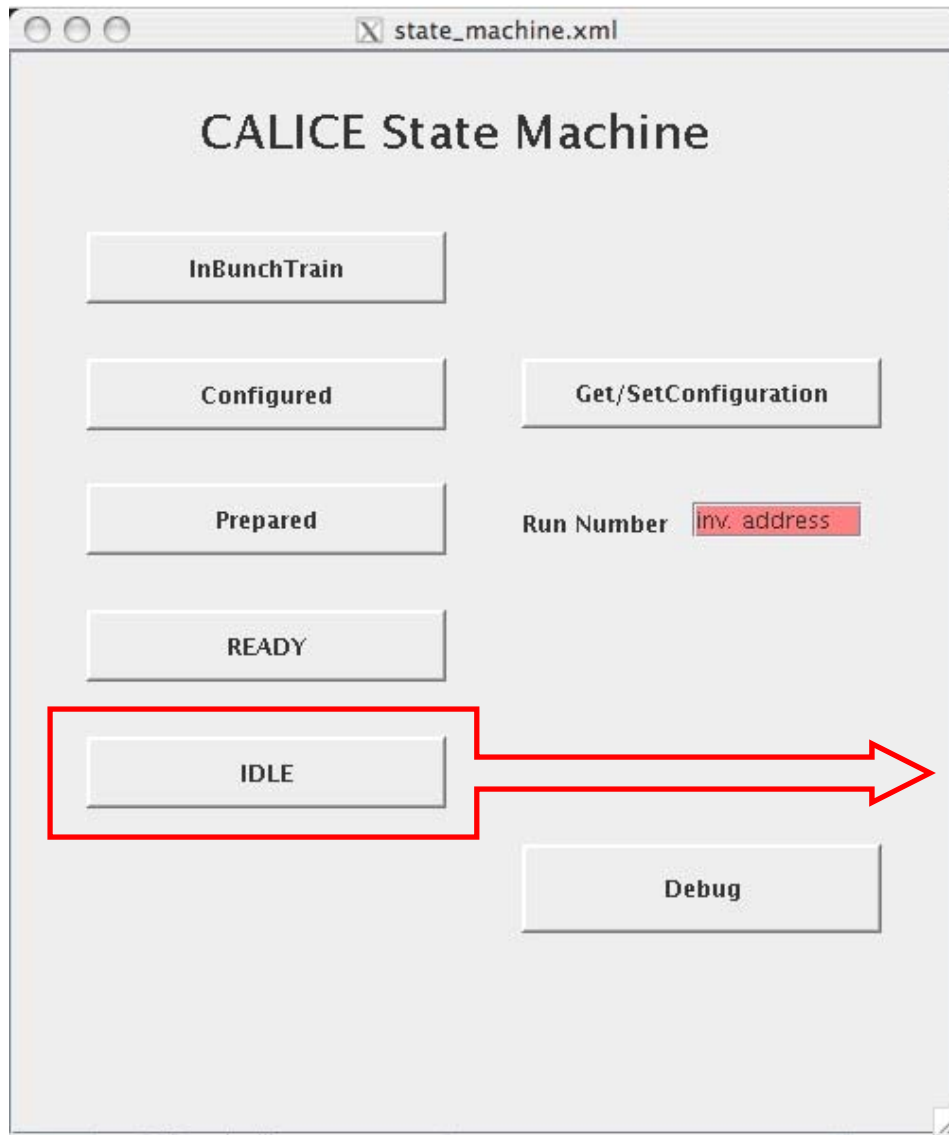
macs odr_rpc_server.cc
macs ClientSocketN.h
macs ClientSocketN.h
```

Reminder of the state machine



The states surrounded by the green box currently work.

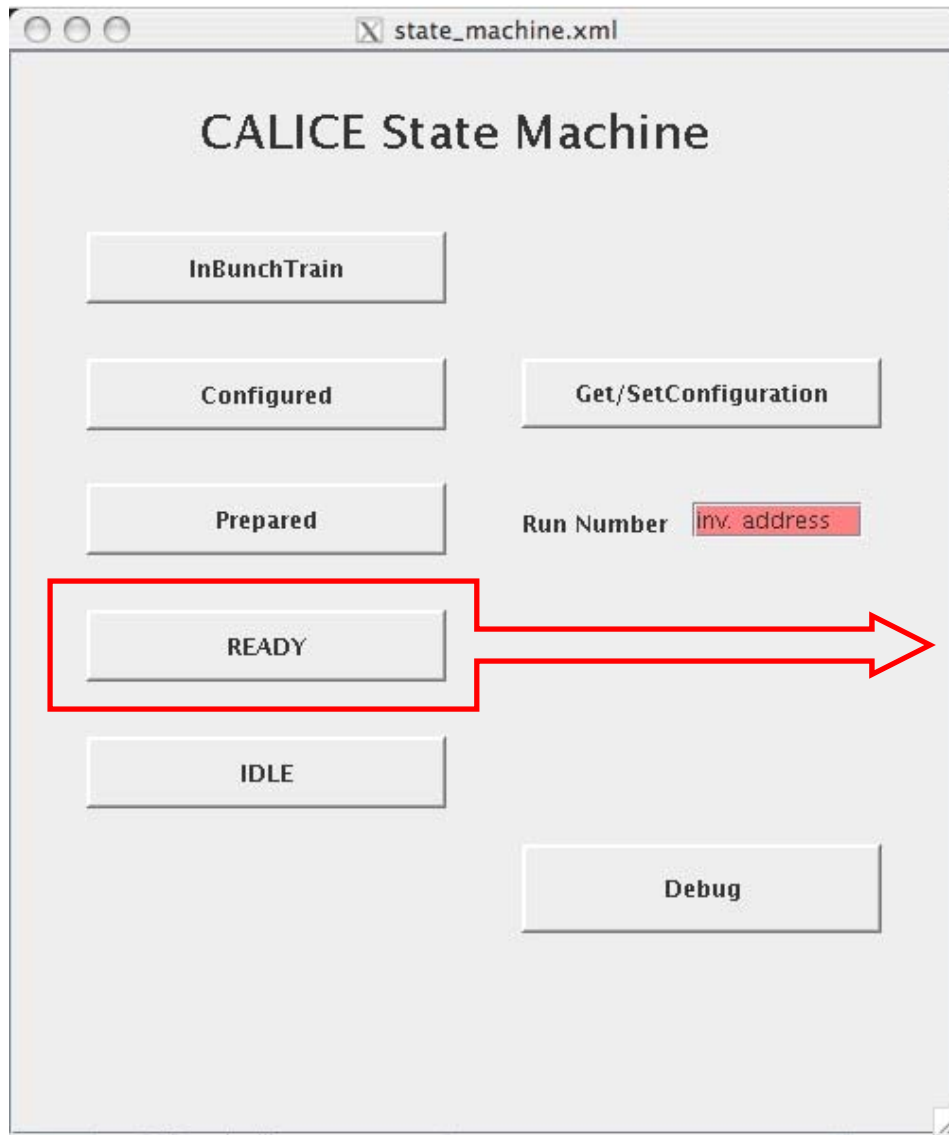
Reminder of the state machine



State IDLE:

- device server running
- device server will not crash, but return to state IDLE if anything happens to connection to device
- device not connected

Reminder of the state machine

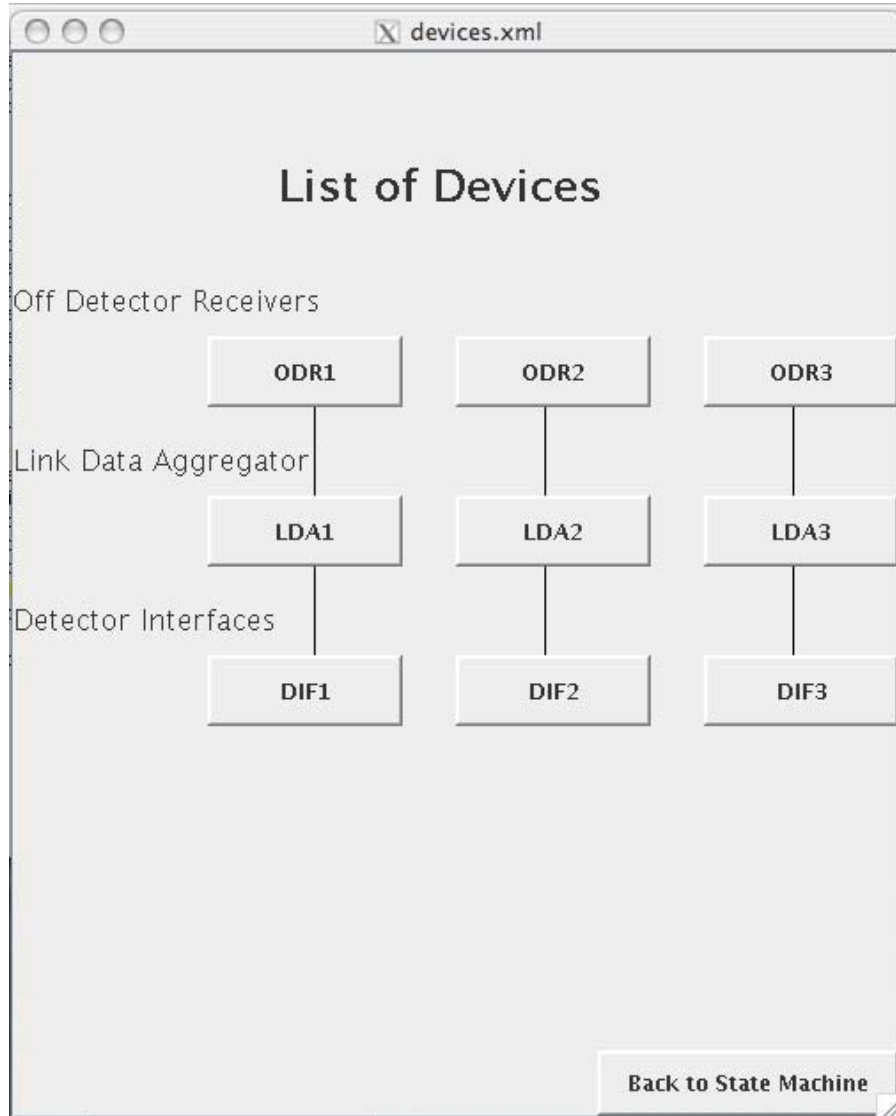


State READY:

- device connected
- thread started

- if anything goes wrong, press Debug button...

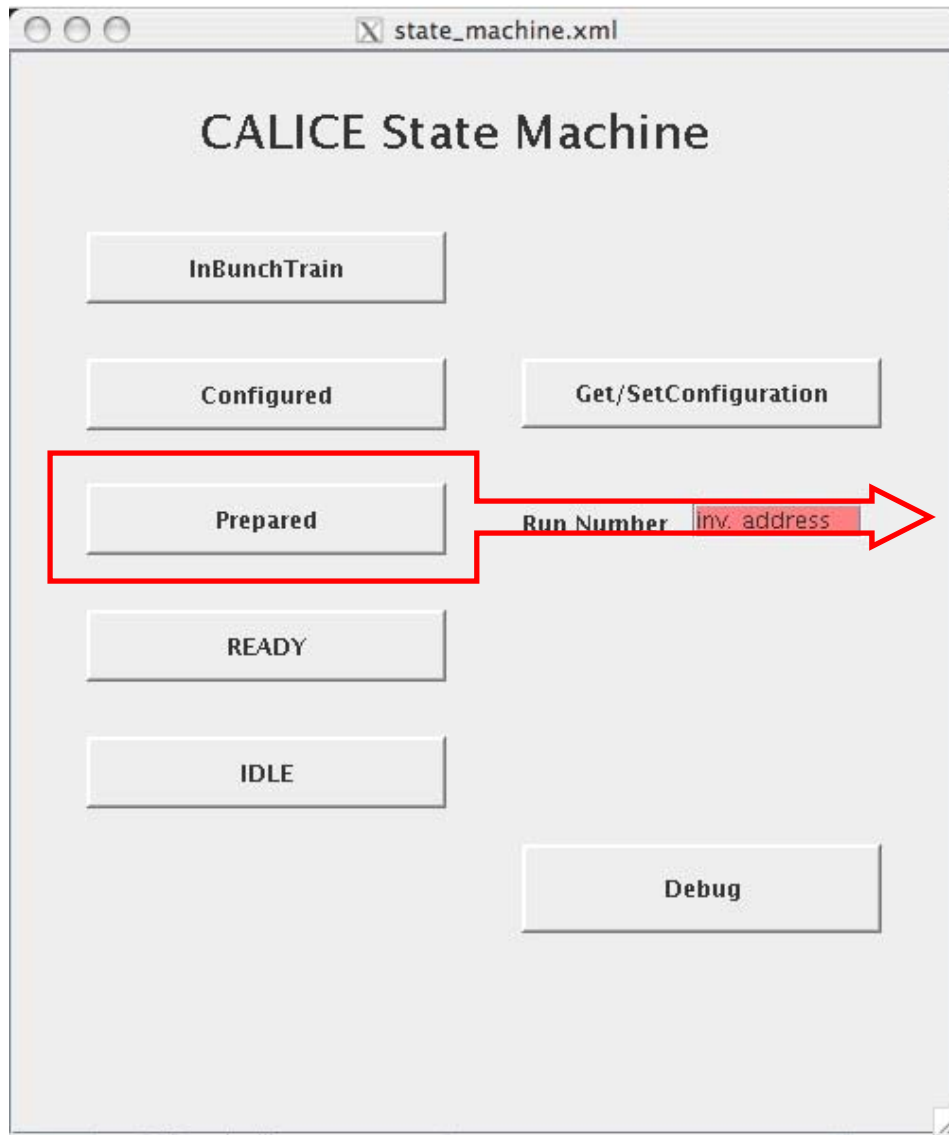
Reminder of the state machine



Debug / devices screen:

- this screen shows the devices and should turn green / red depending of the status of the device
- click on the device will give further information about the device
- which further information is needed is still to be discussed

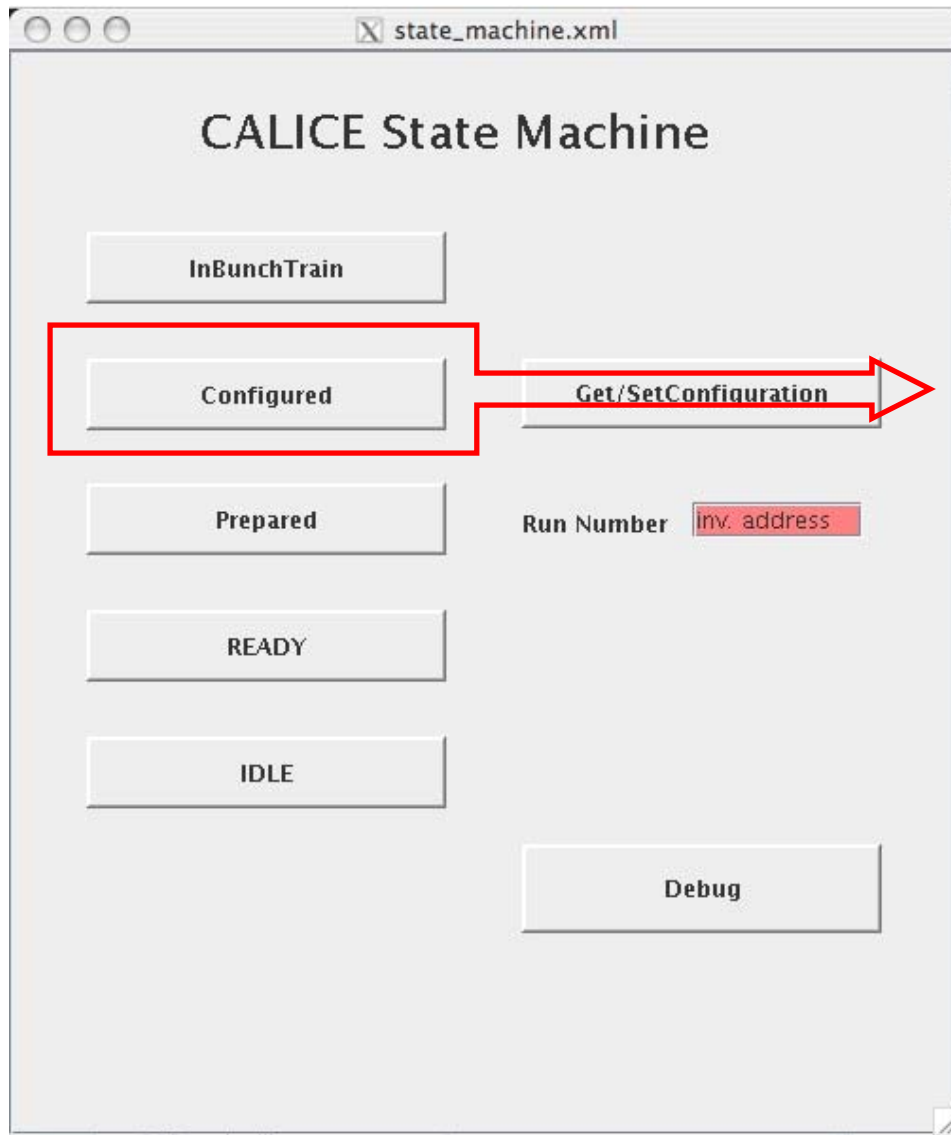
Reminder of the state machine



State PREPARED:

- run number sent from database
- this should be done by a separate db device server, at the moment implemented in ODR device server

Reminder of the state machine



State **CONFIGURED**

- configuration established
- configuration is read from data base
- at the moment configuration fixed by property of device server
- future development: make it more flexible from GUI
- if you press Get/Set Configuration...

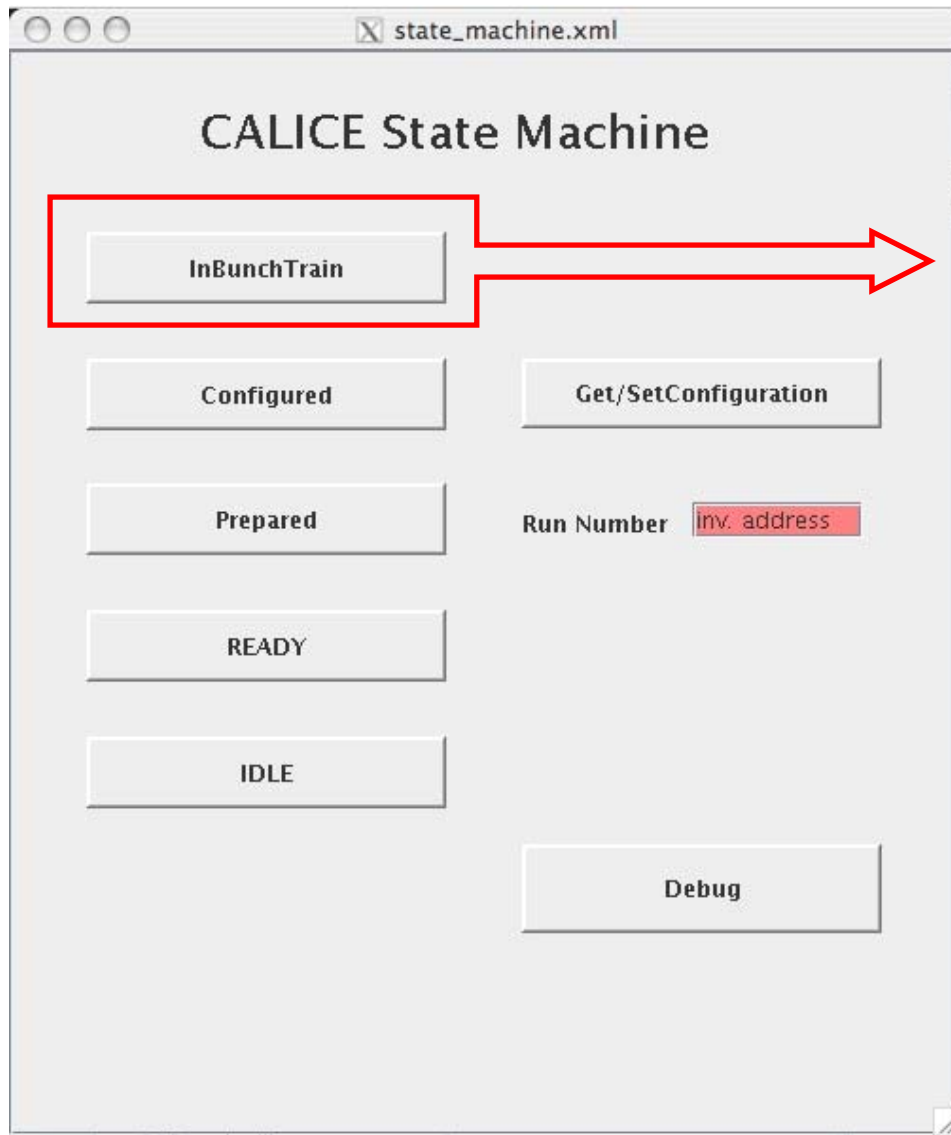
Reminder of the state machine

The screenshot shows a window titled "detector_config.xml" with tabs for "ODR", "LDA", and "DIF". The main content is titled "List of ODR configurations". It features a "configuration id" field with a "no value" dropdown and a "make new configuration" button. Below this is a table of configuration parameters, each with a "no value" dropdown. The parameters are: data size, no of messages, run time (s), dump to screen, grouping, no of IO Threads, dump data to disk 0/1, dump data size, word to extract, DG:0 Network:1, active channels 0/1, No of DMA, start data write, event grouping, DMA debug, text next message, statistics update freq, run, quit, and empty. At the bottom, there are four "Connect ODR to Configuration ID" labels, each with a dropdown menu and a red "address" button. A "back to device menu" button is located at the bottom right.

Parameter	Value
configuration id	no value
data size	no value
no of messages	no value
run time (s)	no value
dump to screen	no value
grouping	no value
no of IO Threads	no value
dump data to disk 0/1	no value
dump data size	no value
word to extract	no value
DG:0 Network:1	no value
active channels 0/1	no value
No of DMA	no value
start data write	no value
event grouping	no value
DMA debug	no value
text next message	no value
statistics update freq	no value
run	no value
quit	no value
empty	no value

- Detector Config screen:**
- not implemented yet
 - shows configurations in database connected to configuration id
 - makes new configuration if there is no good configuration in database
 - can connect chosen configuration to run number

Reminder of the state machine



State InBunchTrain

- file names from ODR are read and send to database to be connected to the run

Outlook

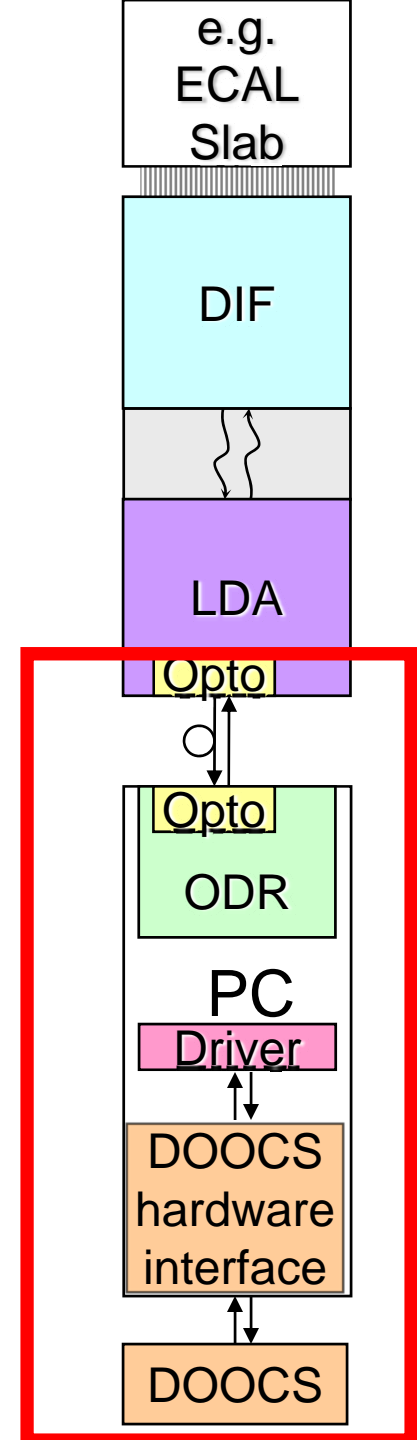
- Finish state machine for ODR
- Implement GUIs
- Implement device server for DIF
- Adapt device server for LDA emulator to real LDA
- Build state machine for LDA/DIF/CCC (only need to implement the states READY/CONFIGURED)
- Error handling

Backup slides

Hardware interface

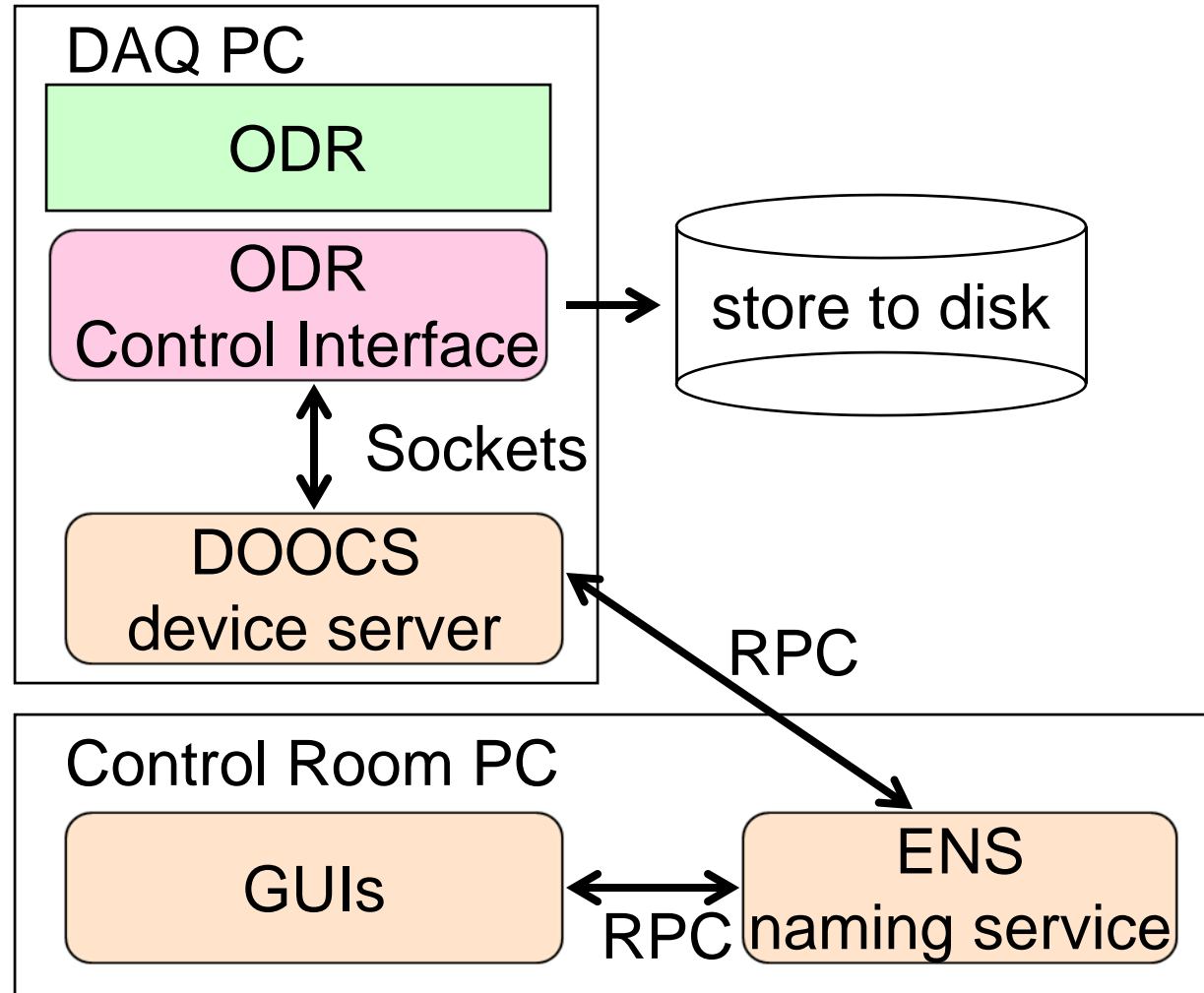
- ODR layer first accessible layer
- ODR ready since last summer

⇒ Demonstrator for the ODR with a LDA emulator shown at the Manchester meeting

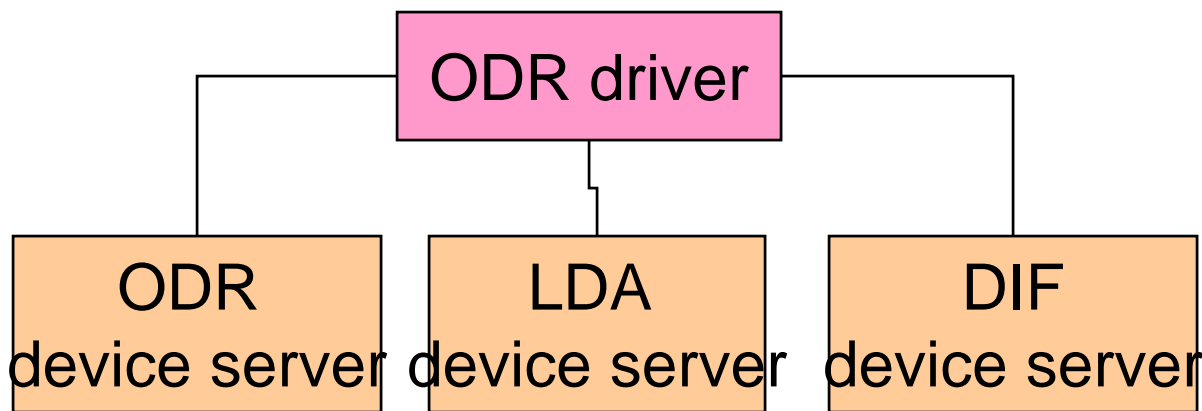


Overview over the ODR interface

- communication between different parts of DOOCS by RPCs
- configuration files used to find different parts of the system



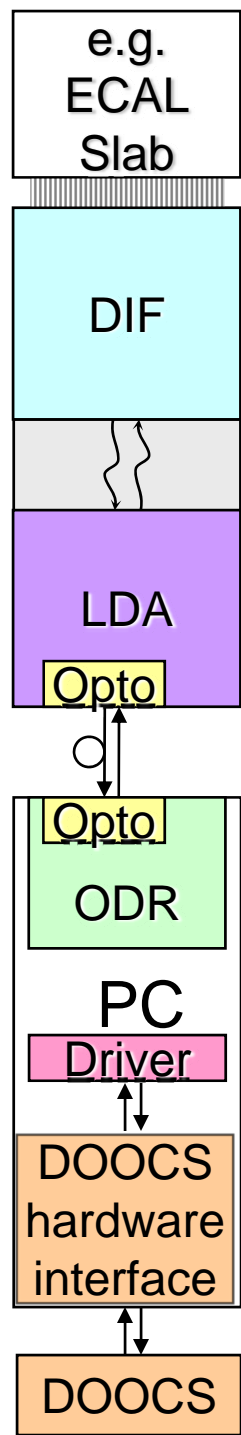
ODR, LDA and DIF device server - envisaged connection with DOOCS -



a different socket for each device server instance:

- 1 ODR socket,
- 4 LDA sockets,
- 32 DIF sockets

⇒ ODR driver needs to detect from where signal is coming and where signal is going



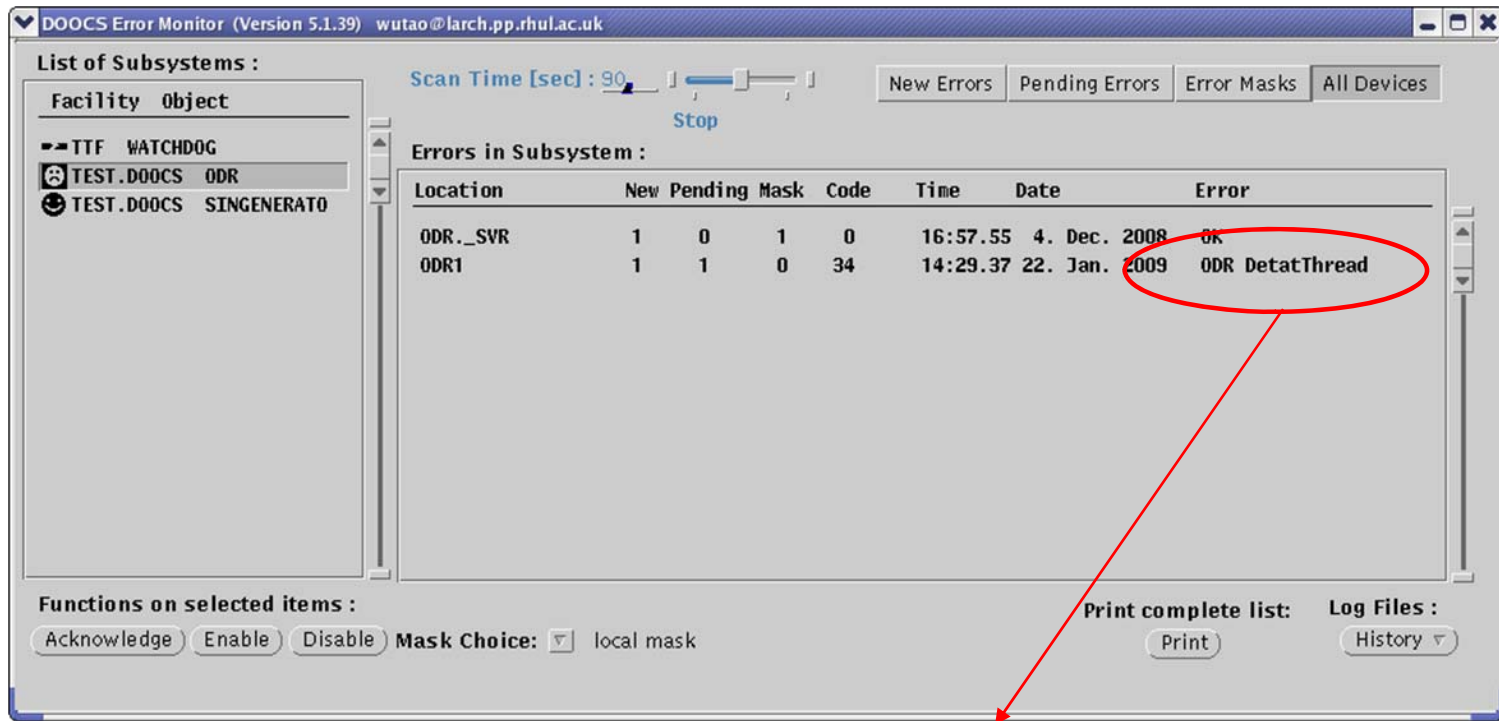
ODR, LDA and DIF device server - hardware, firmware, driver solutions -

How to implement scenario the ODR driver, ODR firmware, hardware:

- **Firmware:** can easily distinguish between upstream (LDA/DIF data) and ODR data
⇒ Firmware needs to be tweaked a little for this
- **ODR driver:** can look at upstream data
⇒ can distinguish between LDA and DIF data

Error handling

- XError GUI interface -



```
ODR Device Error: Can not Detach Thread !  
ODR_Server      ODR1      14:29.37  22.01.2009  -> ODR DetatThread
```

- it is understood how to use Xview alarm handling in DOOCS
- some examples have been implemented for the ODR device server

Database for DAQ

Database handles:

- Connection between devices
 - File storage
 - Runs
 - Device configurations
- ⇒ Resulting in a complicated entity diagram

Database implementation:

- MYSQL chosen as database type
- InnoDB chosen for safe multithread use, backups
- Connection Pool chose to access with several threads

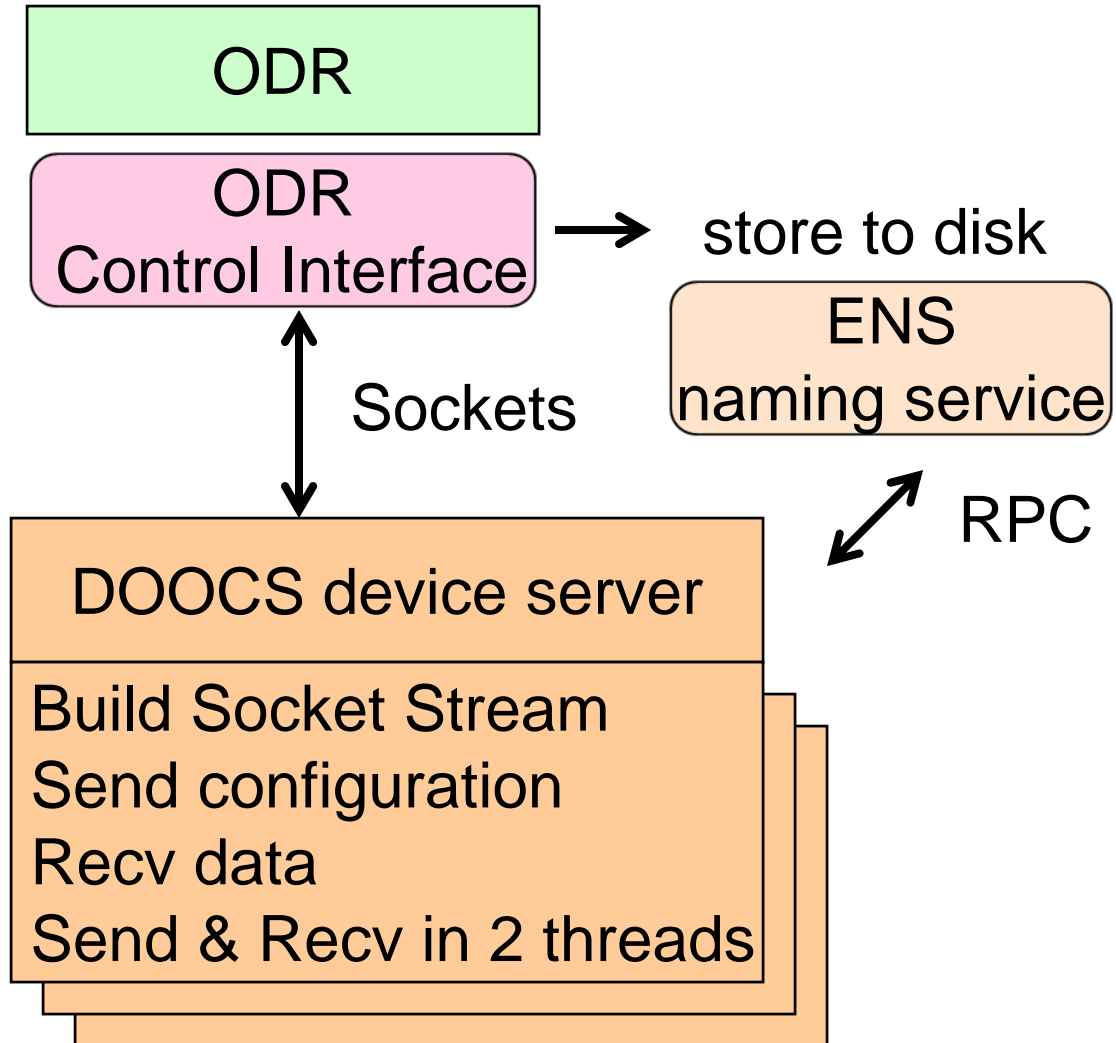
Clock and Control Card Device Server

David Decotigny

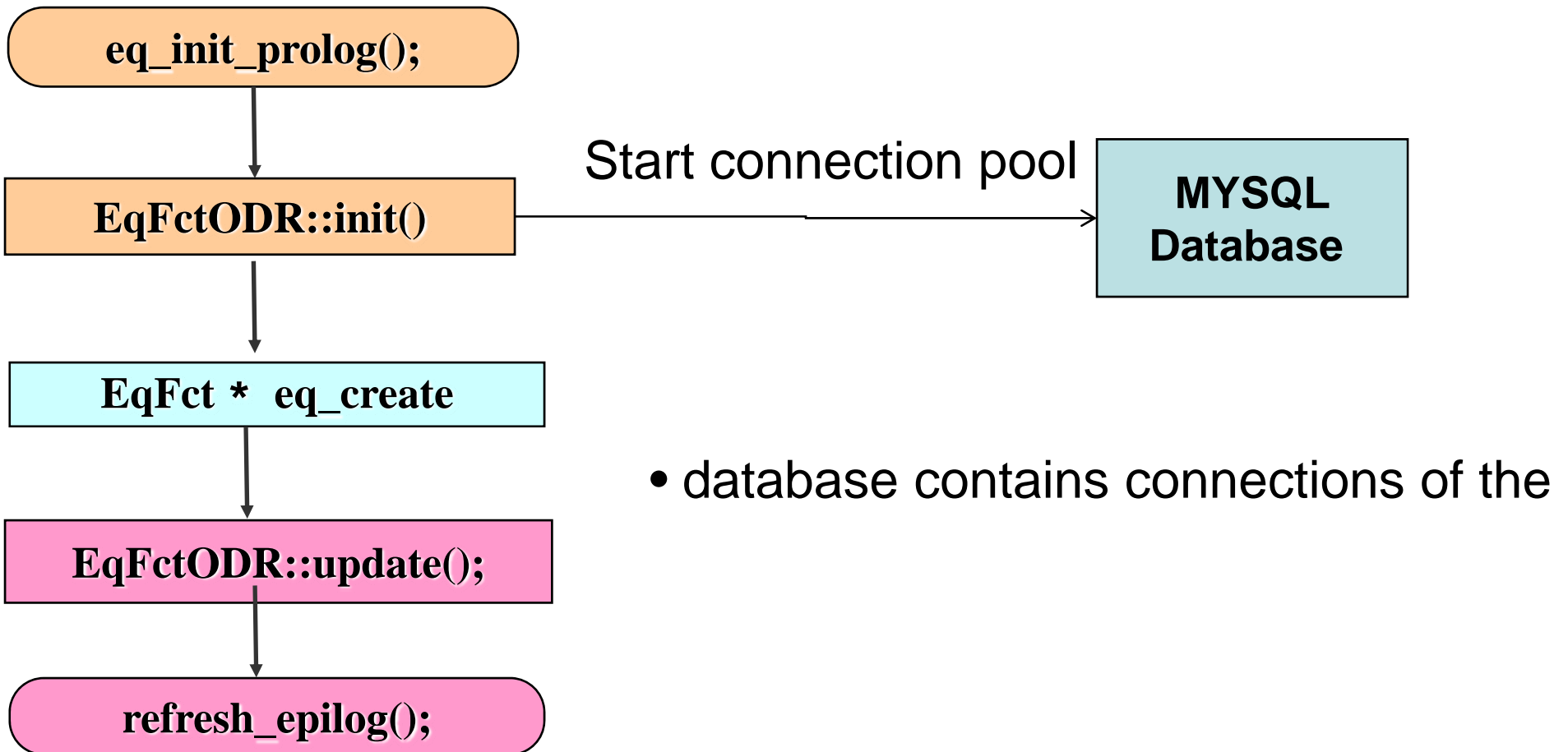
- device server exists
 - registers can be read/written
 - names are assigned as written in design document
-
- no tests on real card up to now
 - error handling still missing
 - at the moment Properties = Registers
- ⇒ need to have more friendly interface for shifters

Overview over the ODR interface

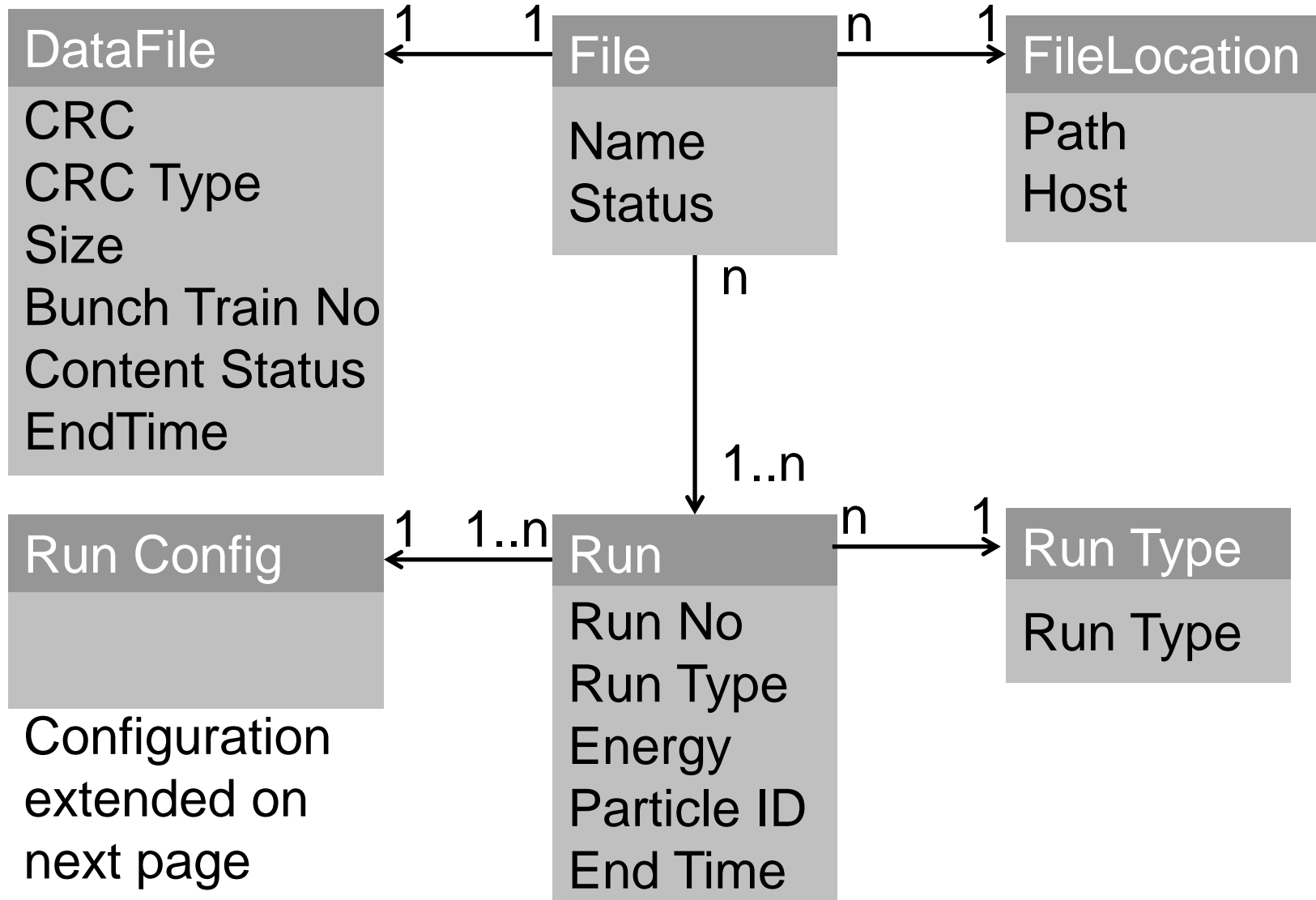
- one device server can have many instance all connecting to different ports and hostnames
- using 2 threads: one for receiving, one for sending on the socket
- sockets format chosen to build an interface to the ODR and the LDA



Database Access

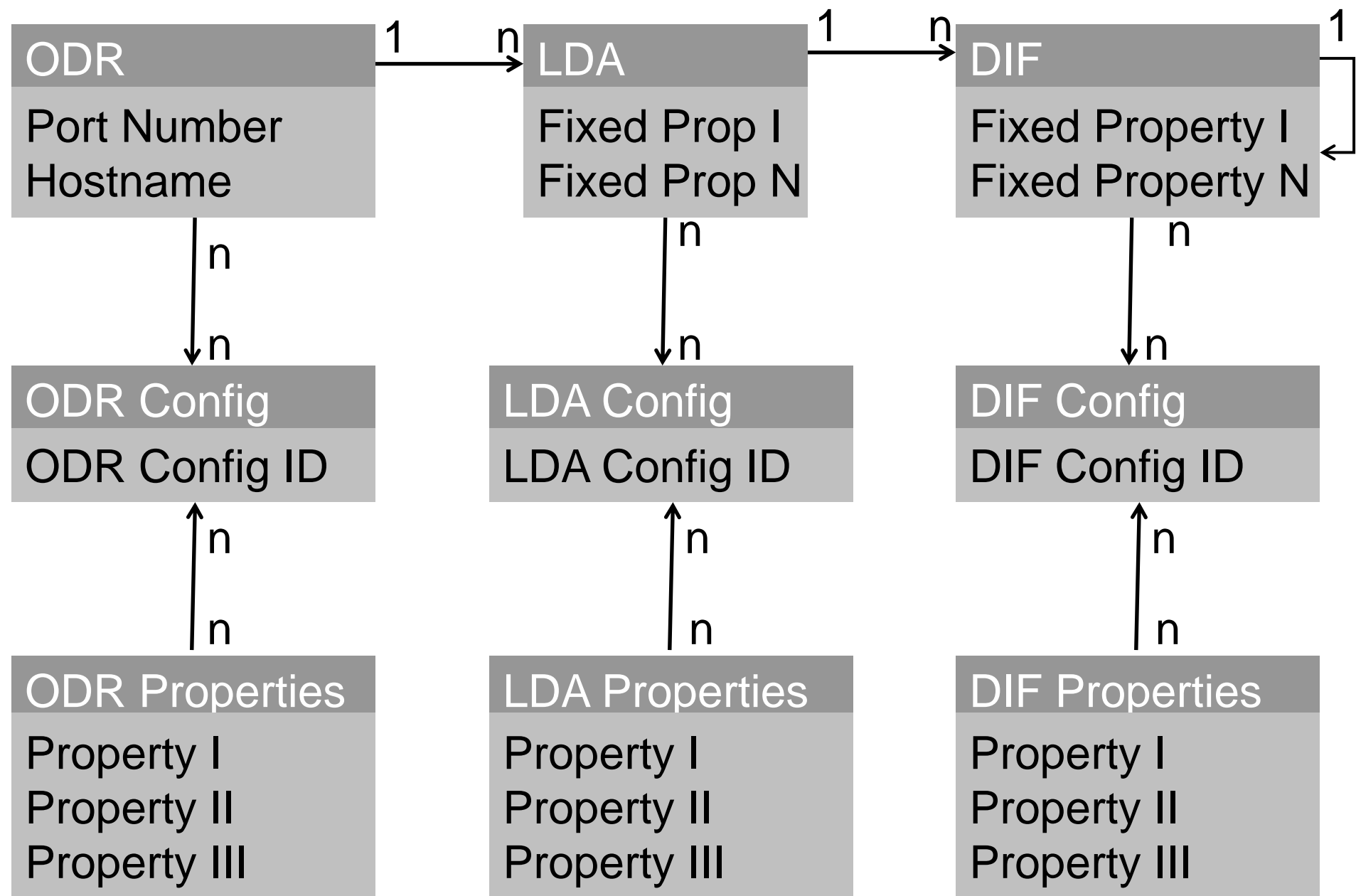


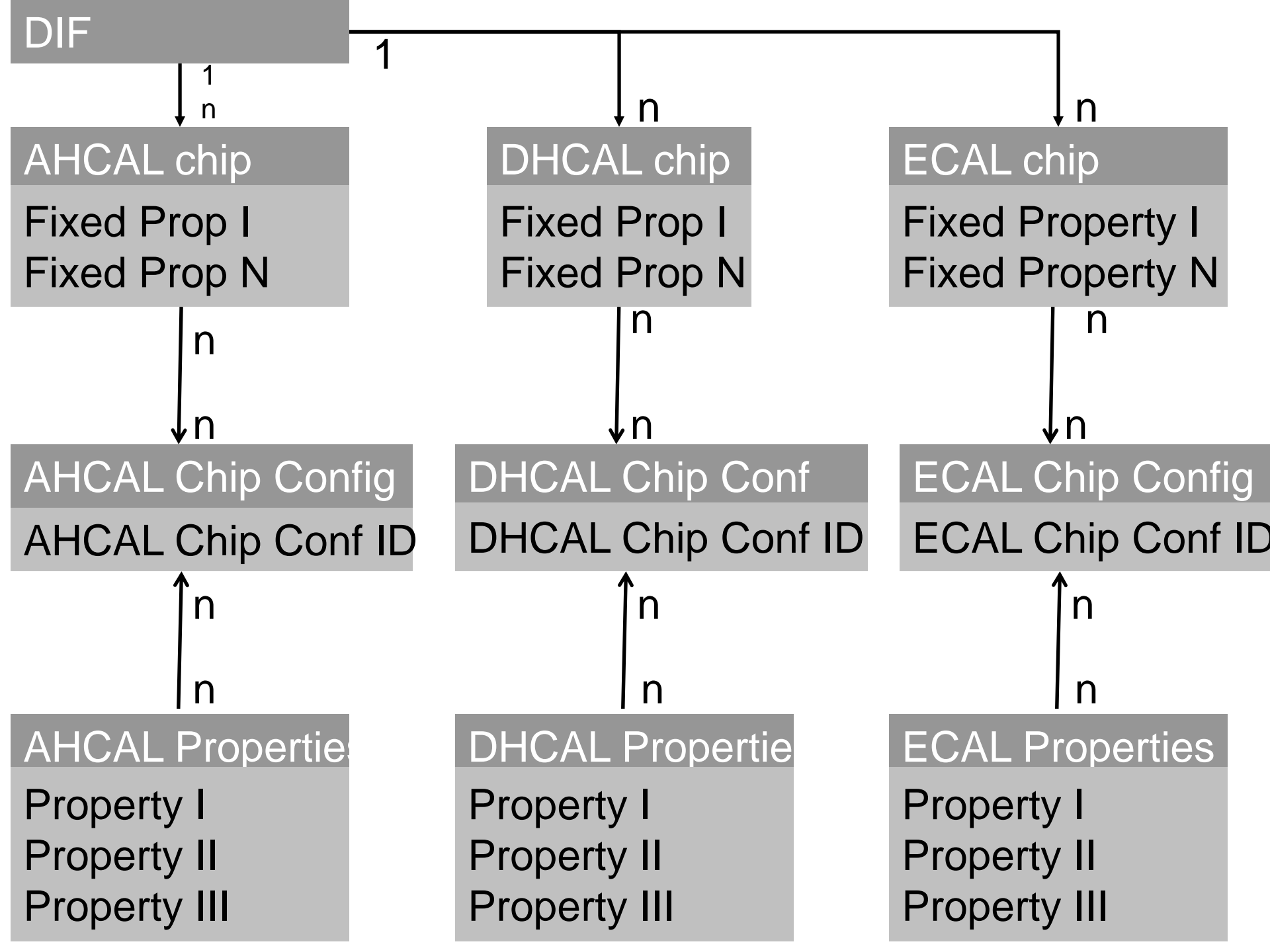
Runs and files



Configuration & Devices

& DCC for DHCAL

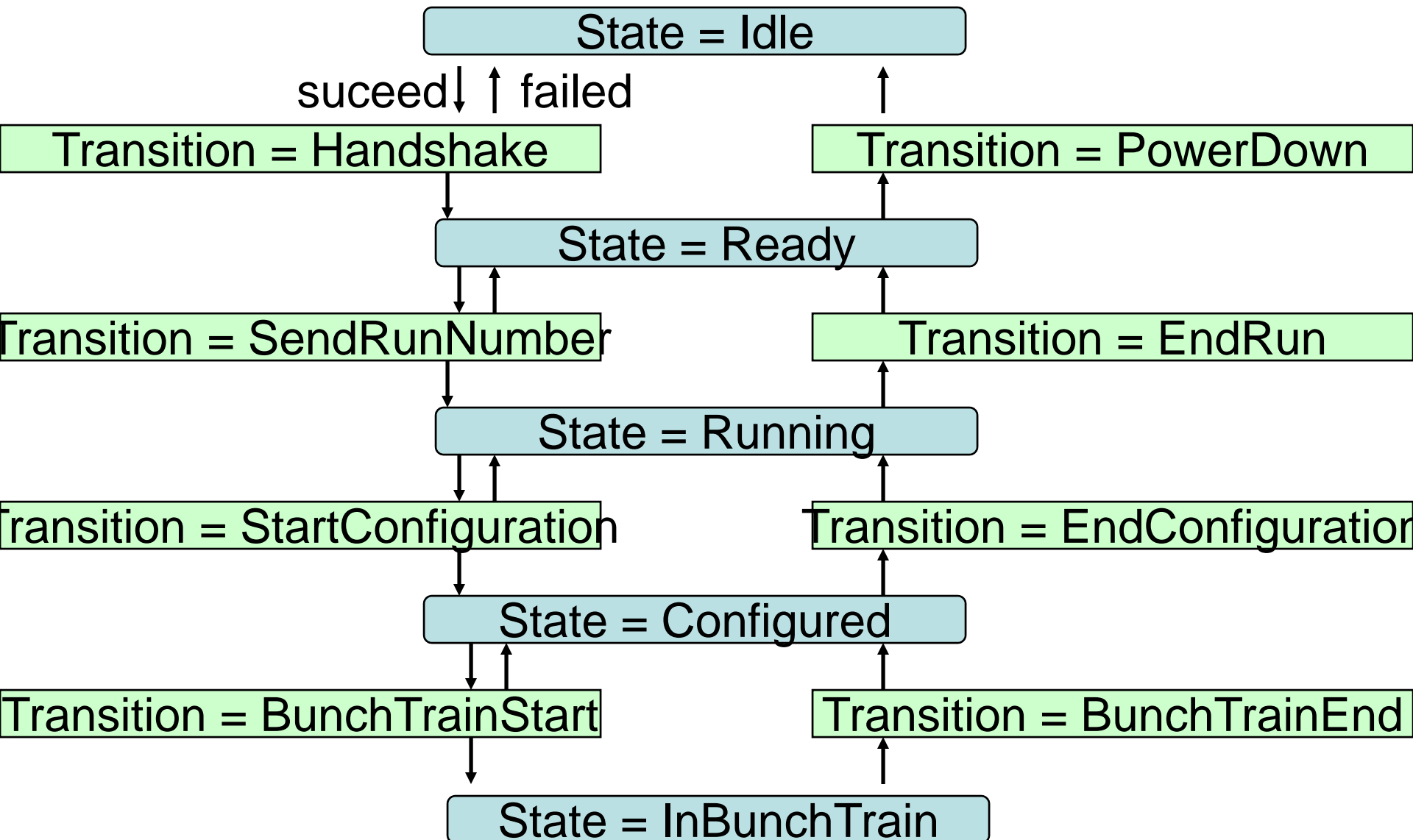




State machine

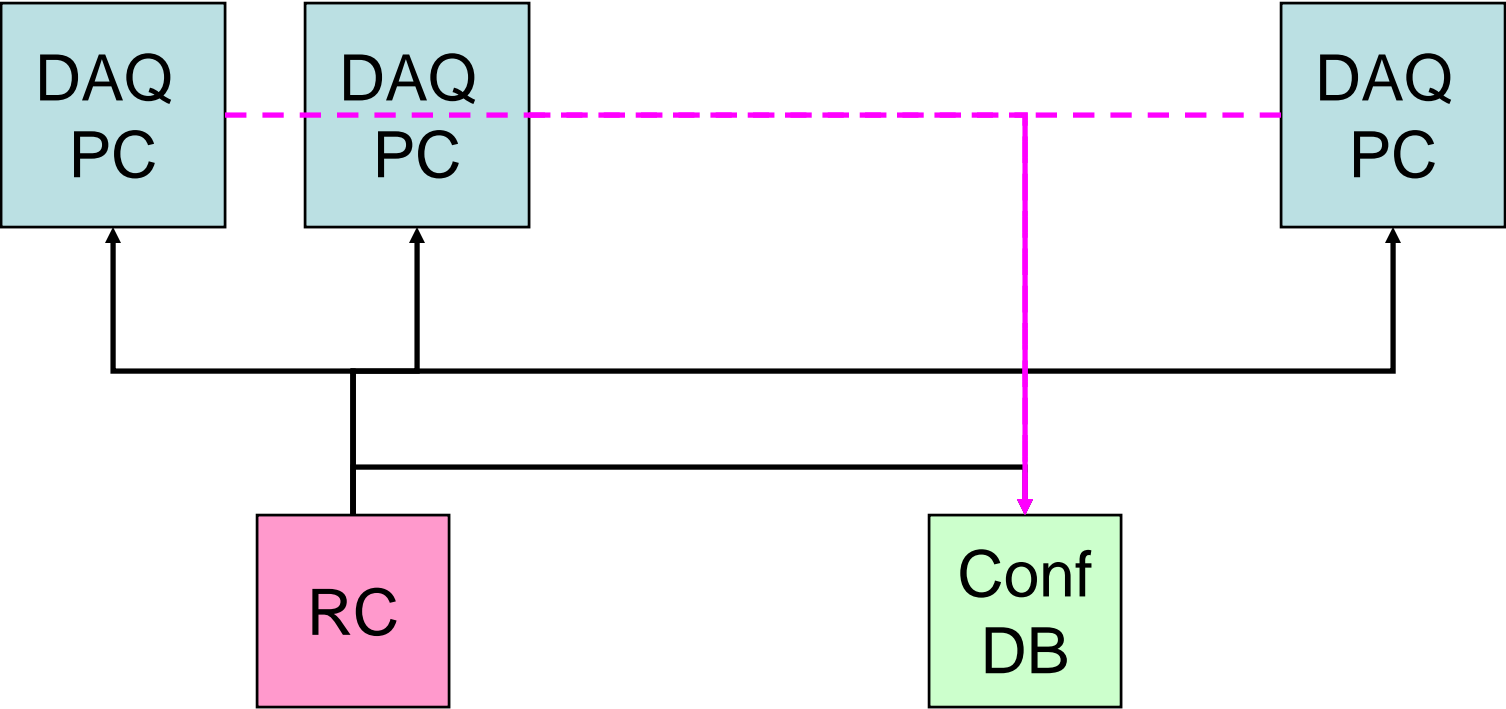
- What we need to do to ramp up for data taking:
- Send hardware handshake to check connections (could also be done by getting conf.)
- Let file database know about run number
- Tell ODR which run number we have right now to put it into the file name
- Send conf.
- Receive automatic acknowledgement or send getConfiguration command

State Analysis

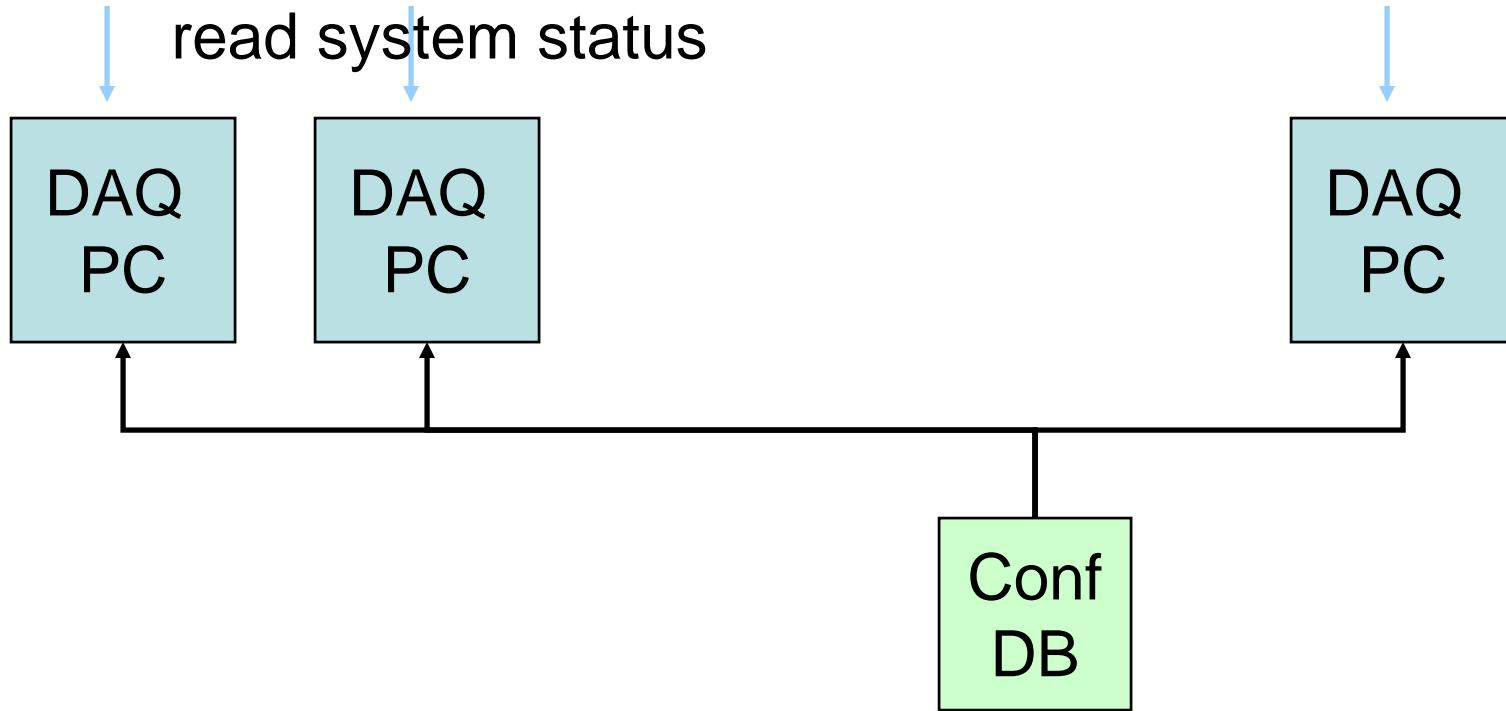


Transition: Handshake

→ establish connections

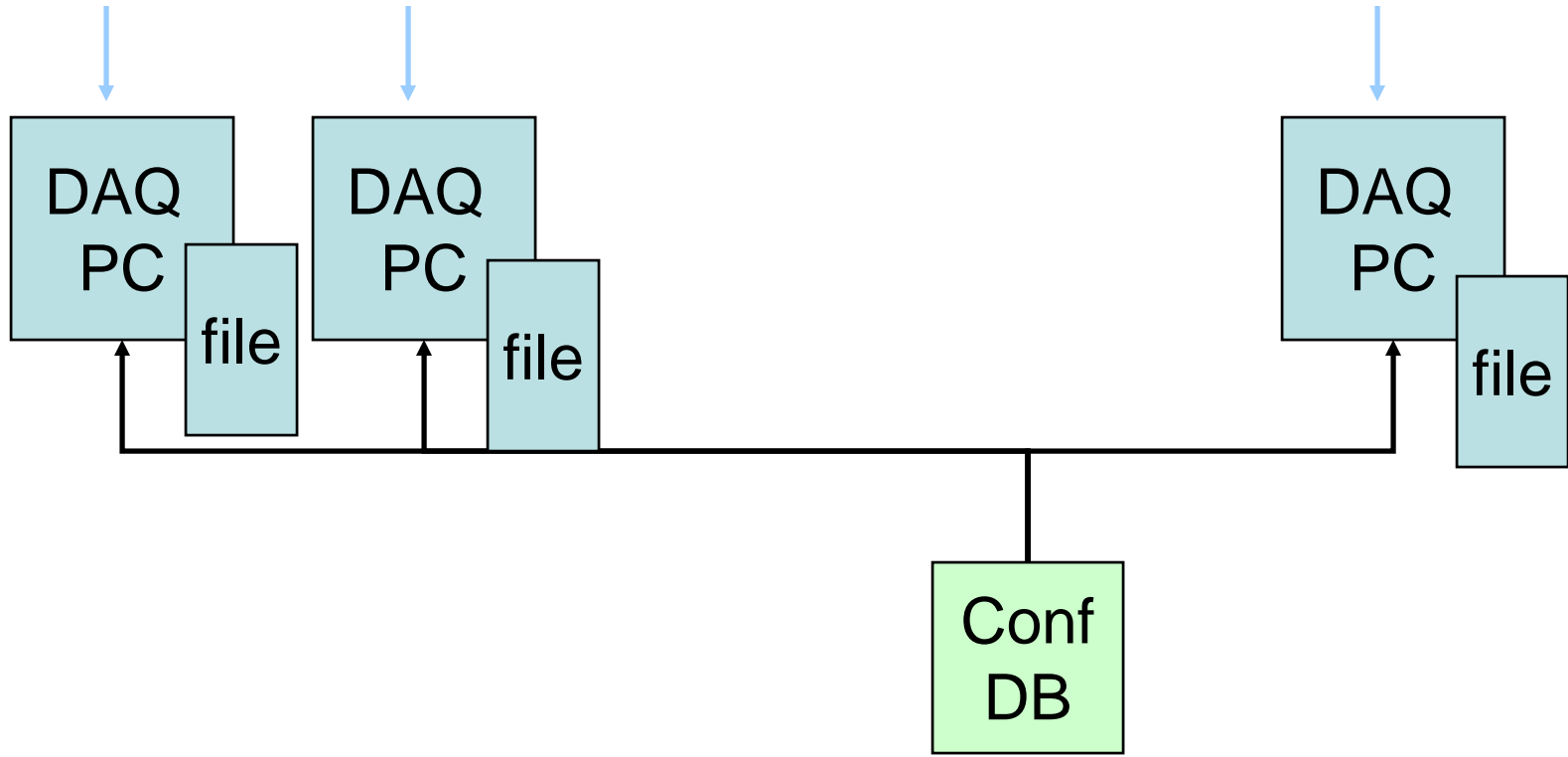


Transition: StartRun



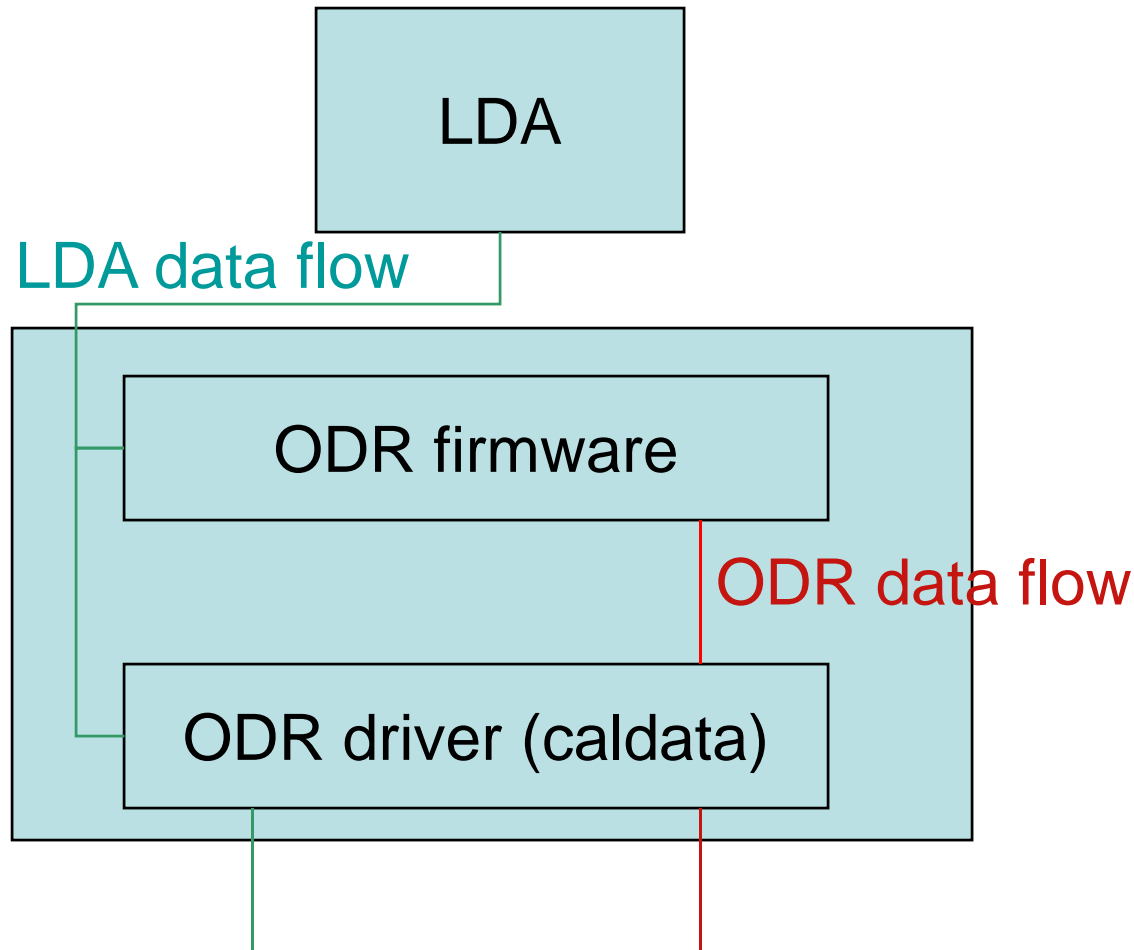
Send run number to ODR software,
Make new run number plus unique in file
database
(filename = [run_number + unique identifier])
and fill in configurations

Transition: StartConfiguration



Extract conf files for all device servers from db,
Recheck that configuration has been received

ODR, LDA and DIF device server - hardware, firmware, driver solutions -



- different control/configuration data paths because ODR firmware can distinguish between data flow to/from ODR and upstream

Methods of the device server

eq_init_prolog();

EqFctODR::init()

EqFct * eq_create

EqFctODR::update();

refresh_epilog();

The init() method is call for every location during startup of the server. Initialization of the hardware may be done here

during startup of the server to create the locations, properties loaded.

This update method usually does the real work in a DOOCS server. It runs in a loop over all locations