Development of the DAQ software for the technical prototype:

Status & Outlook

Valeria Bartsch UCL

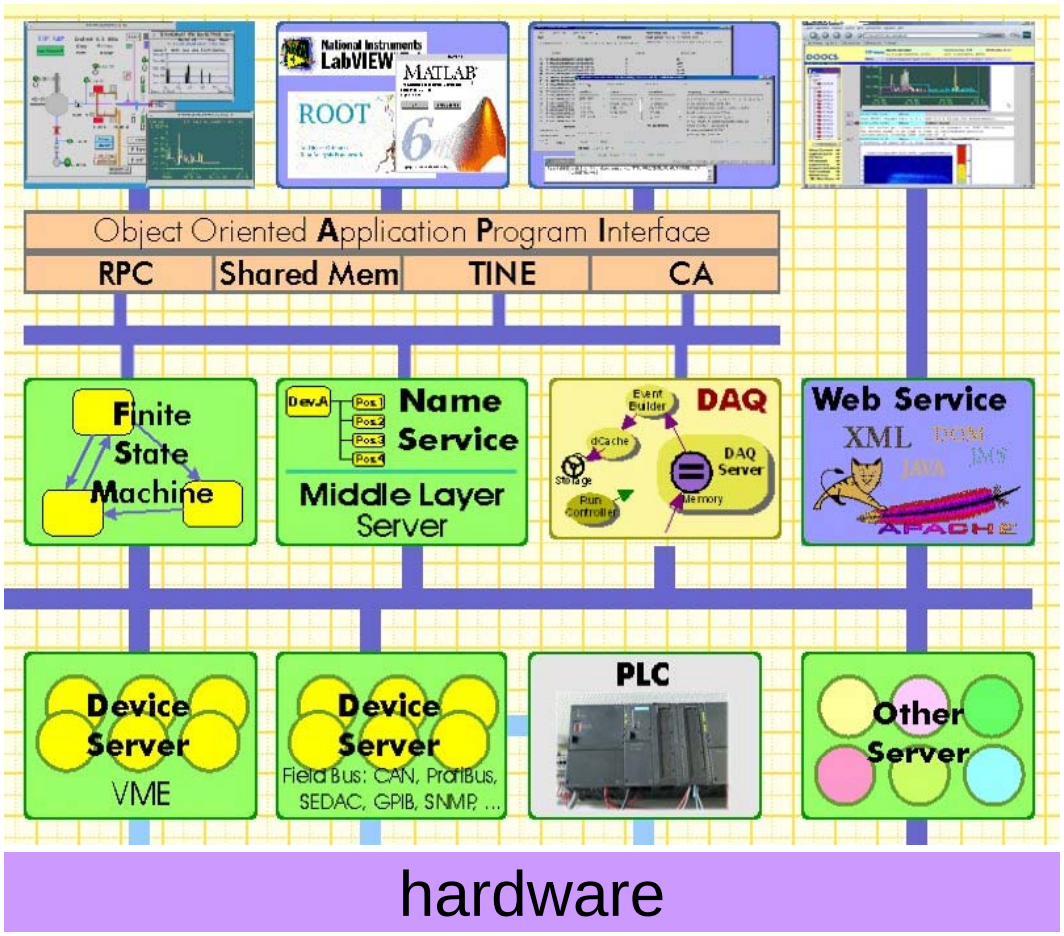
David Decotigny LLR

Tao Wu RHUL

Andrzej Misiejuk RHUL

Overview over the task
- DOOCS software used as framework-

http://tesla.desy.de/doocs/doocs.html



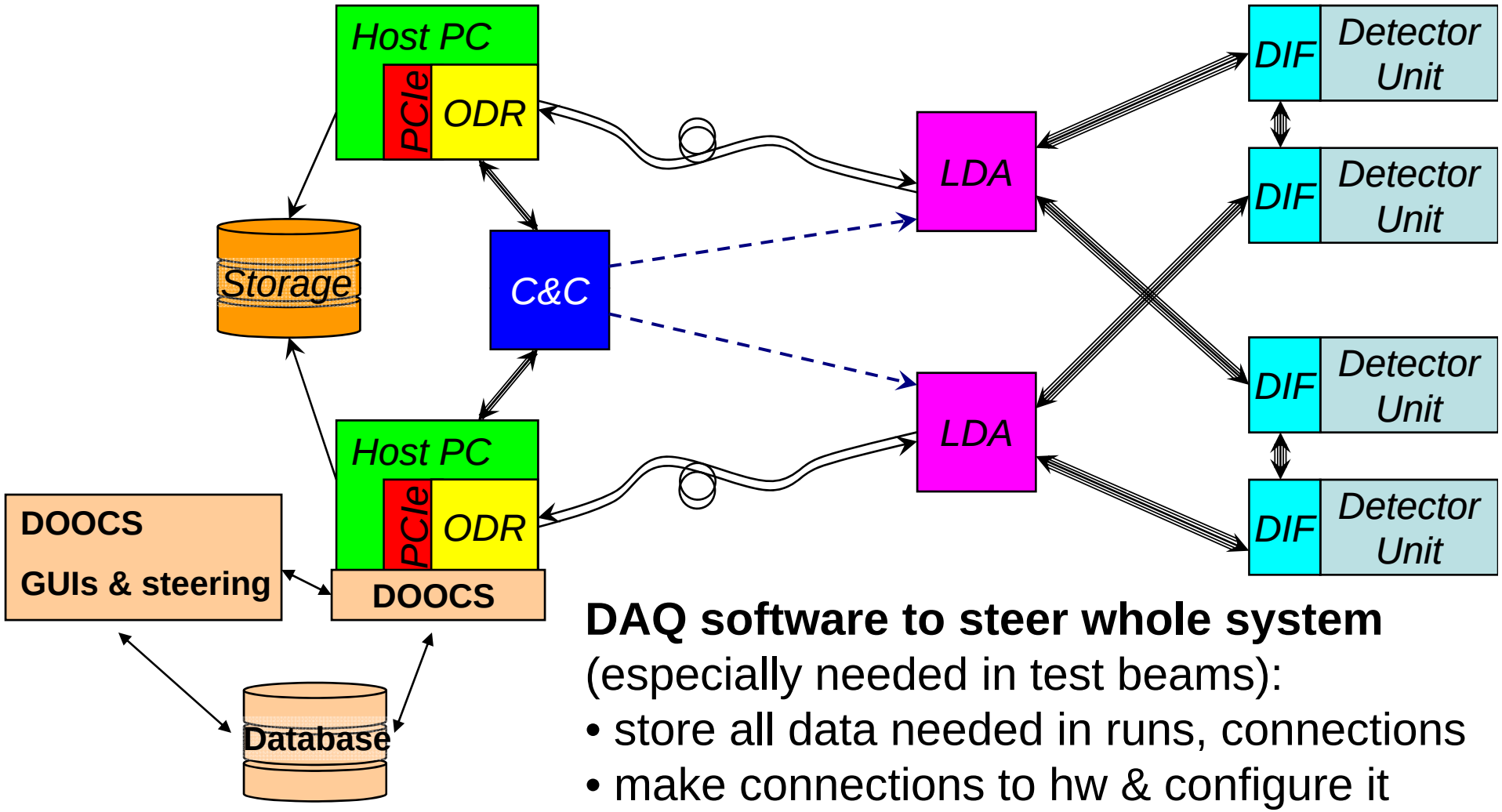
User Interface

Program Interface

Middle Layer

Hardware interface

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

Task Overview

- **Done:**
  - ENS naming service understood and working on a distributed system
  - ODR device server & well tested: reading and writing
  - LDA emulator
  - Implementation of error handling
  - GUIs

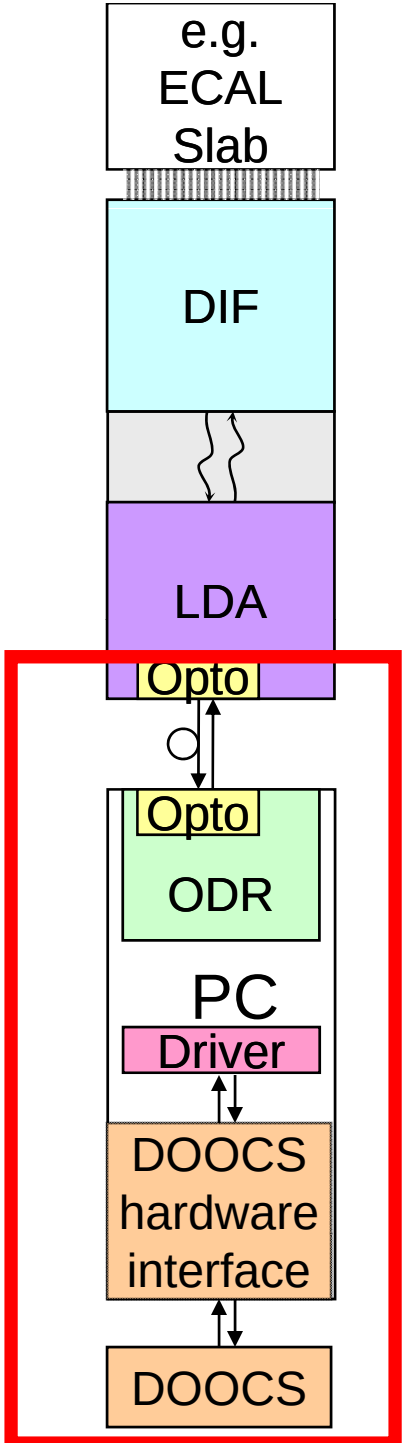
Task Overview

- **Being done now**
  - Data handling - Configuration/Device/Data DB
  - C&C device server
  - ODR state machine
- **Not yet, needs h/w development**
  - LDA device server
  - DIF device server
  - Full state machine

Hardware interface

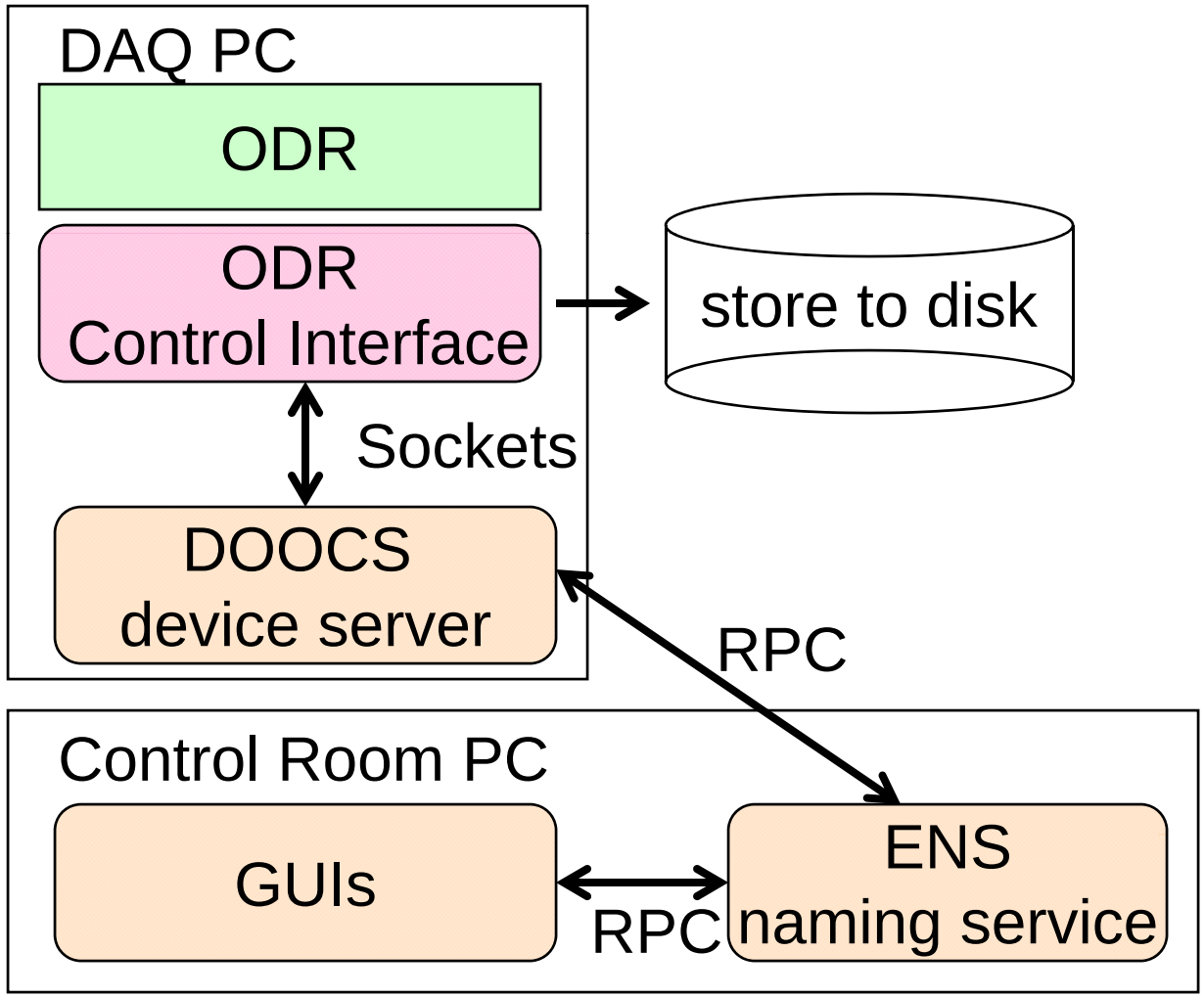
- ODR layer first accessible layer
- ODR ready since last summer

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

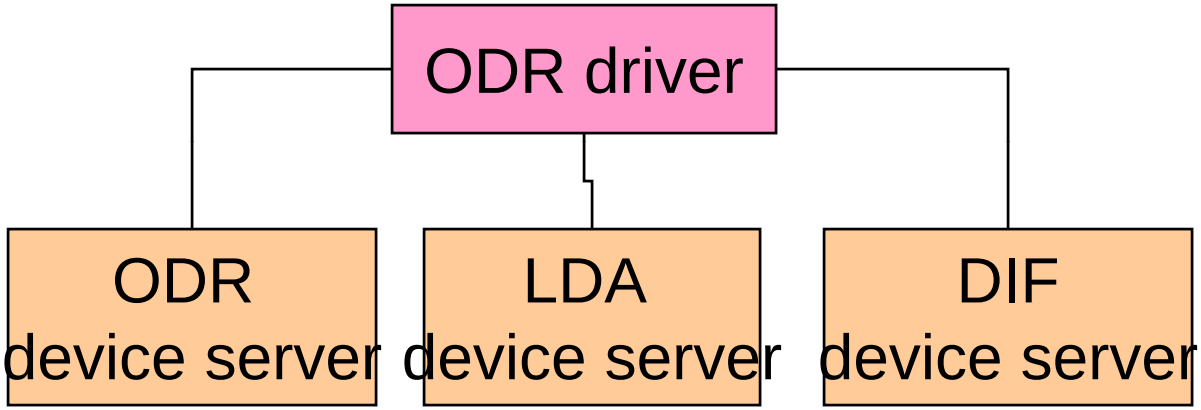


Overview over the ODR interface

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



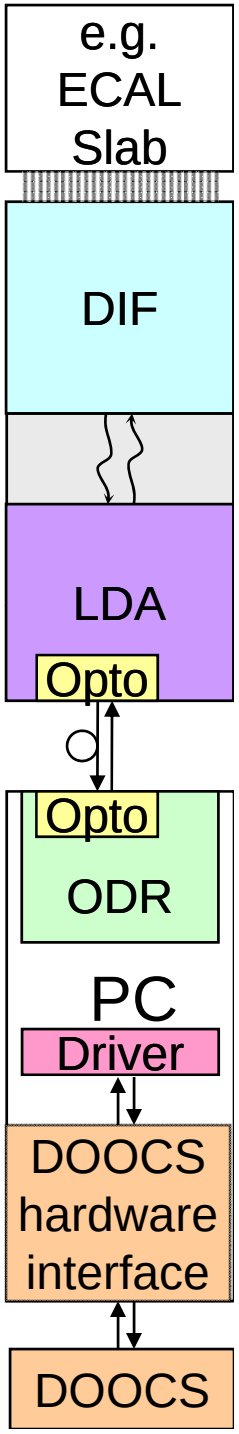
ODR, LDA and DIF device server
- envisaged connection with DOOCS -



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:

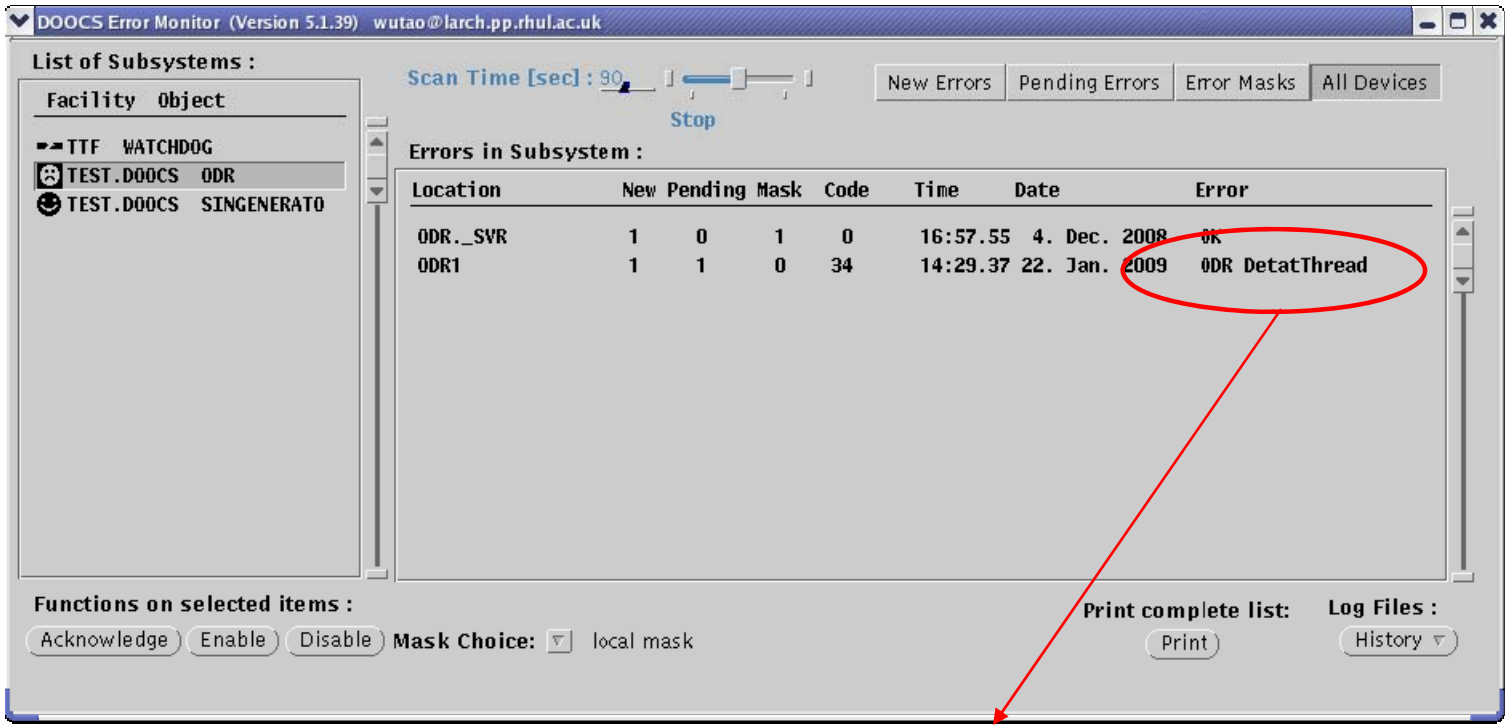
- Firmware: can easily distinguish between upstream (LDA/DIF data) and ODR data

$\Rightarrow$ Firmware needs to be tweaked a little for this

- ODR driver: can look at upstream data

$\Rightarrow$ 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

$\Rightarrow$ 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

$\Rightarrow$need to have more friendly interface for shifters

Conclusion & Outlook

- ODR device server used as guineapig in order to test and implement all features needed
  - Error handling: done
  - GUIs: developed
  - Database access: started
  - State machine: to be developed

- New device servers to follow soon when hardware is ready:
  - Design concepts: understood
  - CCC device server: waiting for real device to test
  - LDA device server: only emulator exists (how realistic is it?)
  - DIF device server: waiting for real device

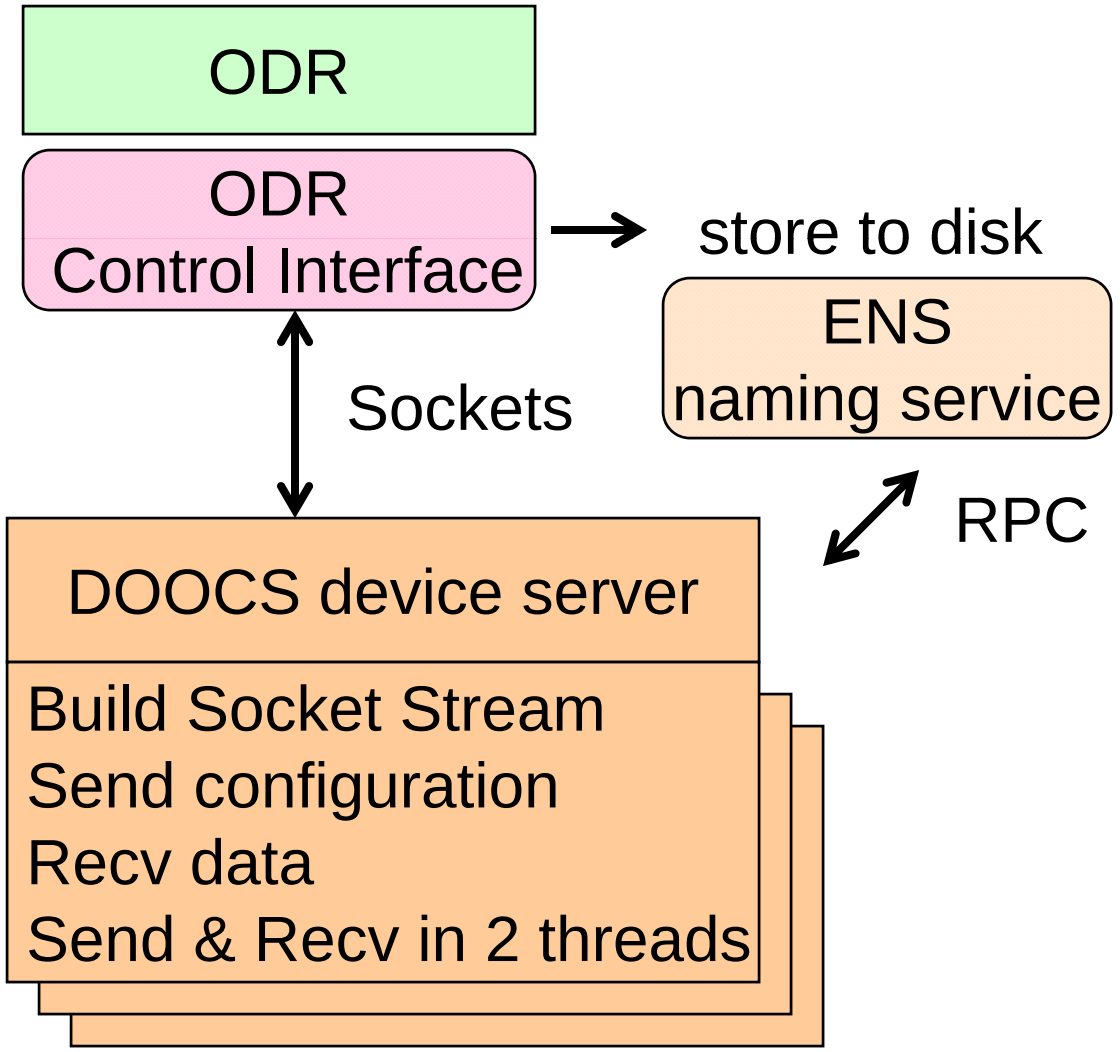
Backup slides

Overview over the ODR interface

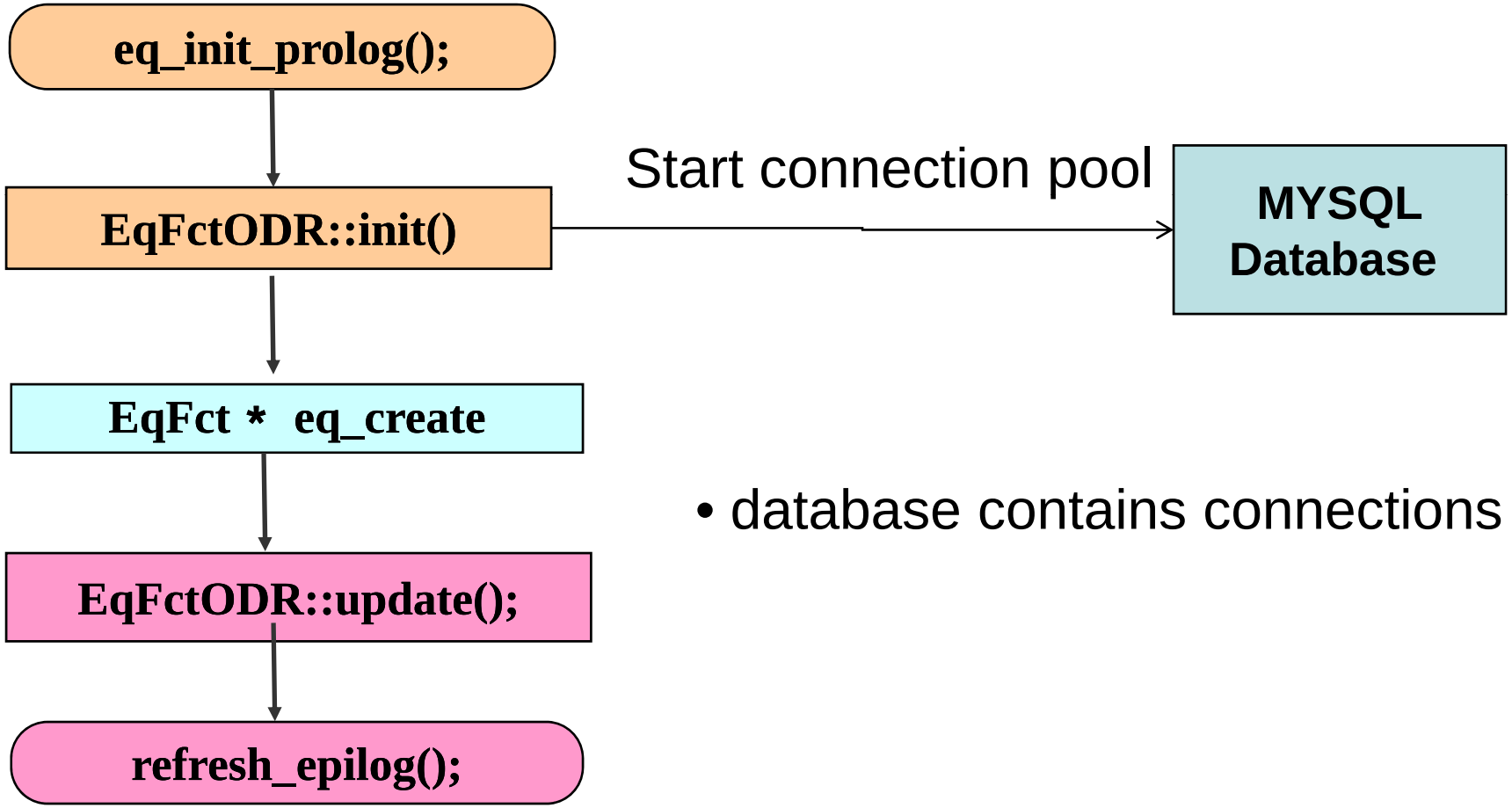
- one device server can have many instance all connecting to different ports and hostnames

- 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

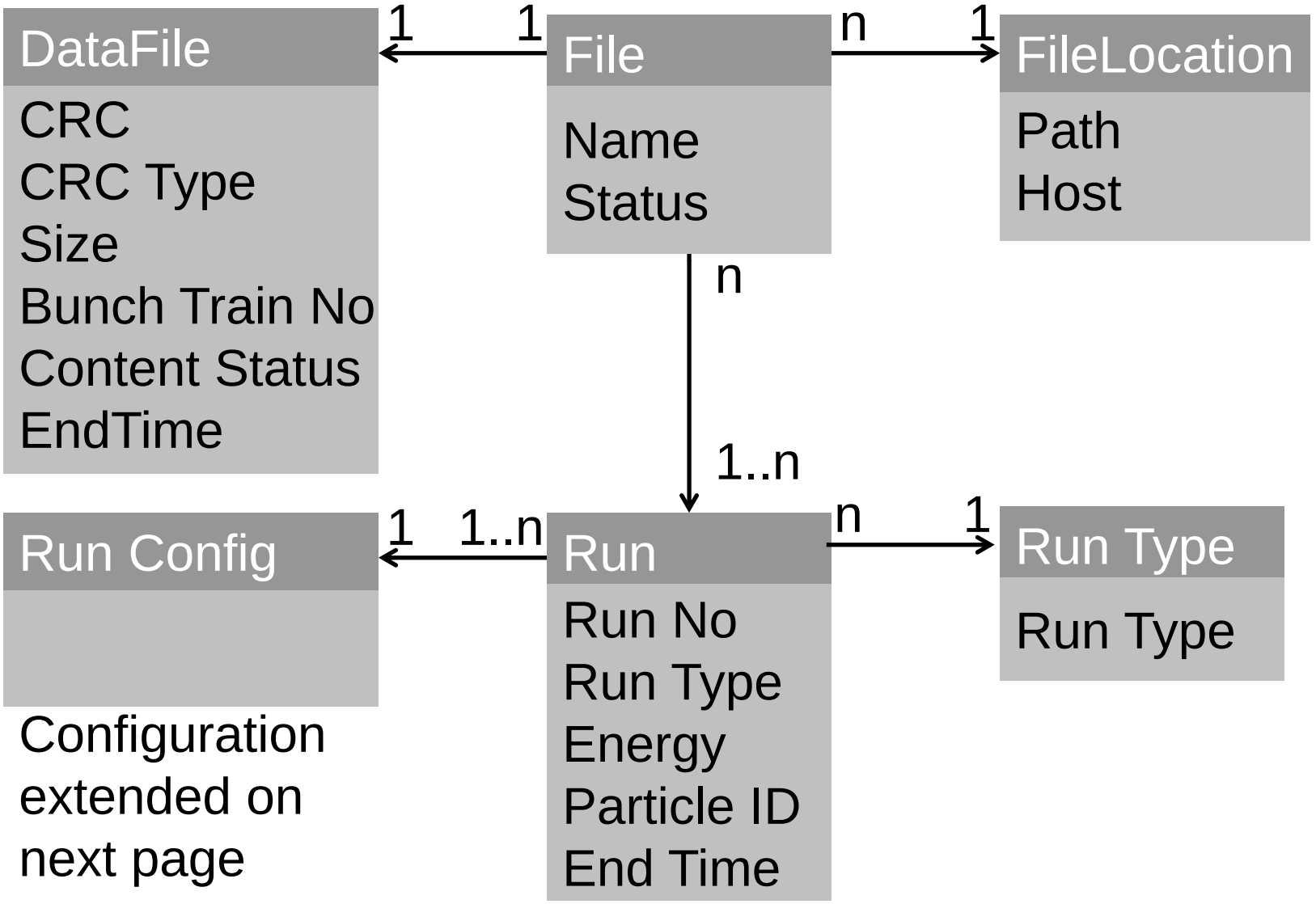


Database Access



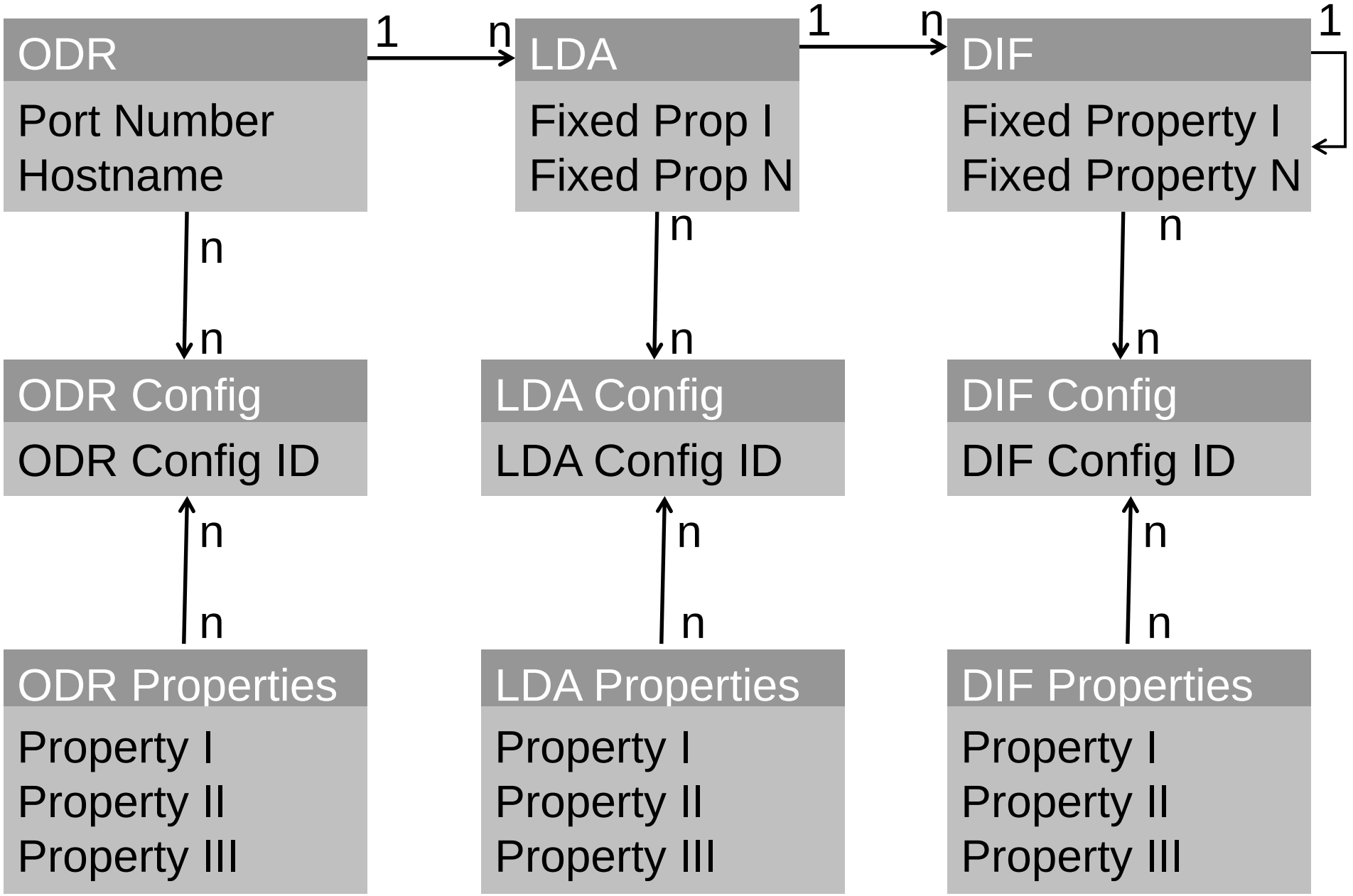
- database contains connections of the

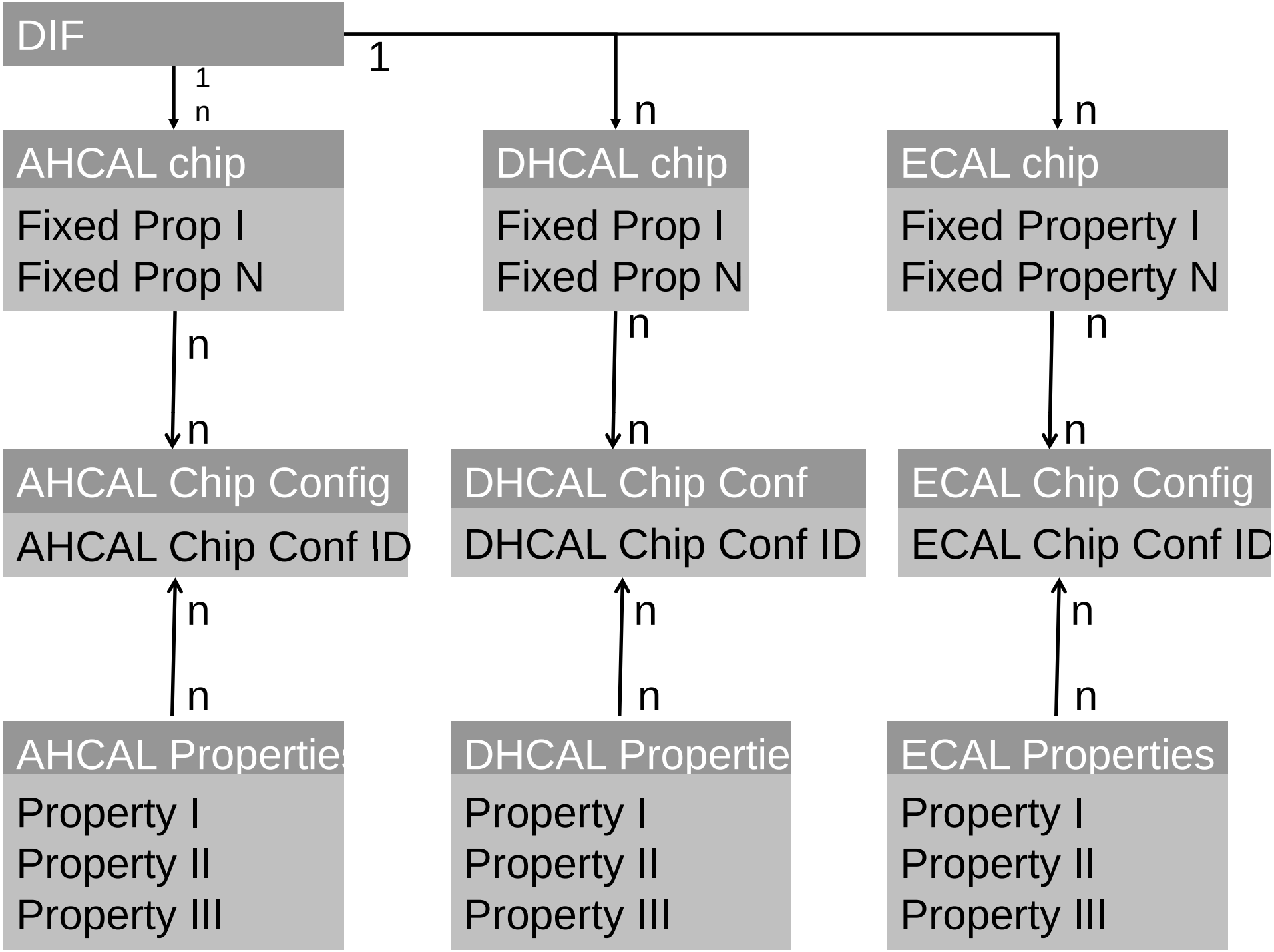
Runs and files



Configuration & Devices

& DCC for DHCAL

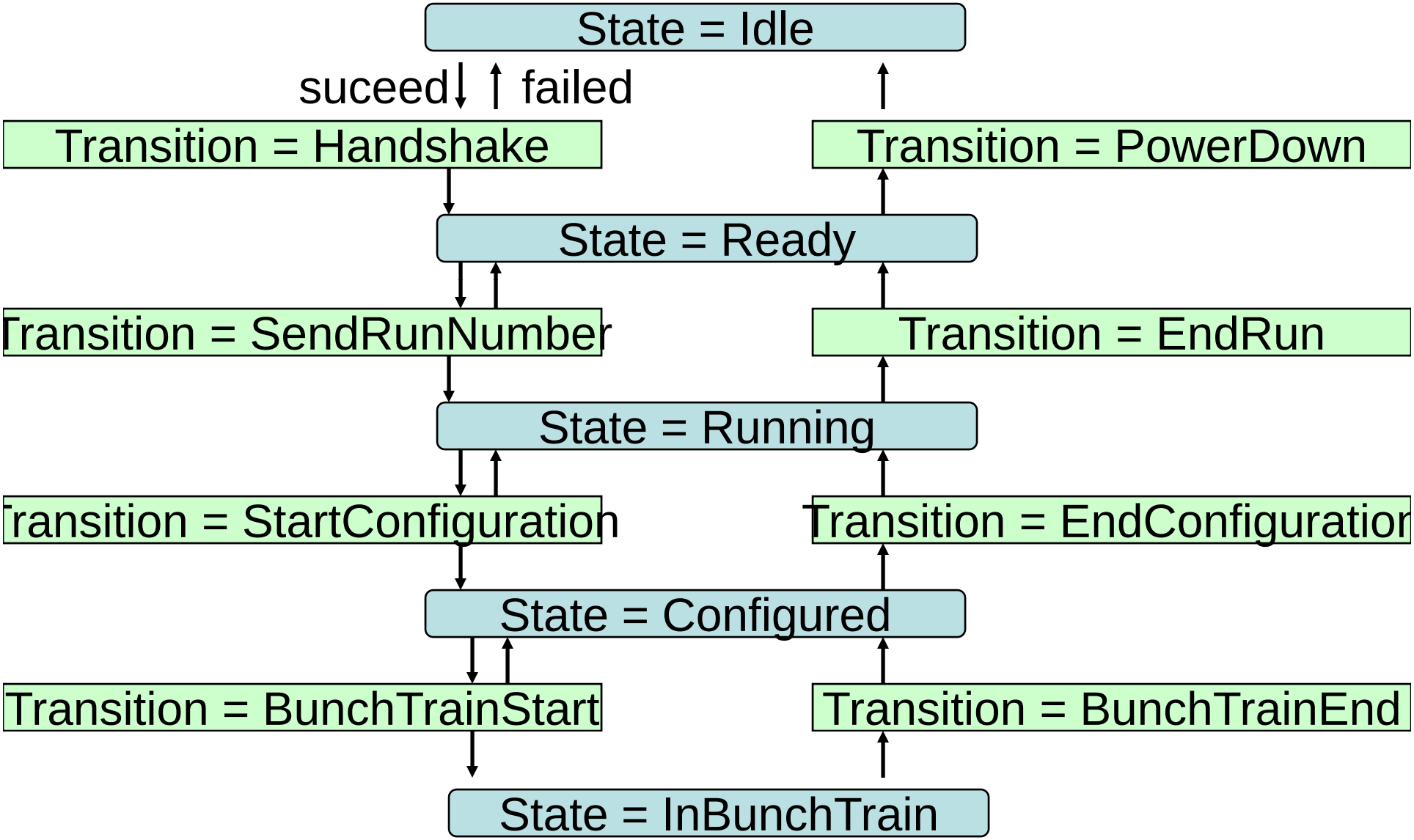



DIF
1
1
n
n
n
AHCAL chip
Fixed Prop I
Fixed Prop N
DHCAL chip
Fixed Prop I
Fixed Prop N
ECAL chip
Fixed Property I
Fixed Property N
n
n
n
n
n
n
AHCAL Chip Config
AHCAL Chip Conf ID
DHCAL Chip Conf
DHCAL Chip Conf ID
ECAL Chip Config
ECAL Chip Conf ID
n
n
n
n
n
n
AHCAL Properties
Property I
Property II
Property III
DHCAL Properties
Property I
Property II
Property III
ECAL Properties
Property I
Property II
Property III

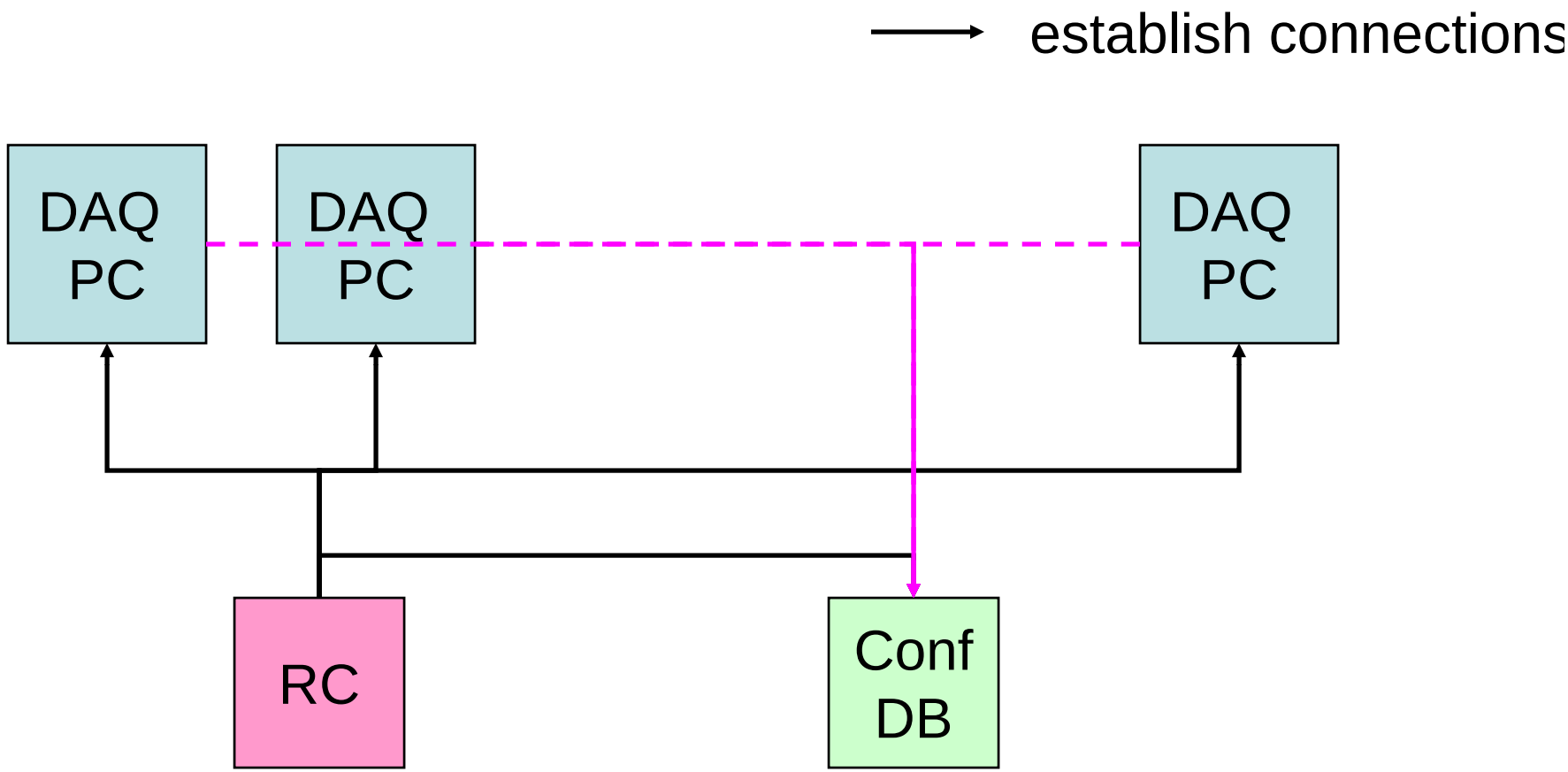
State machine

- 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

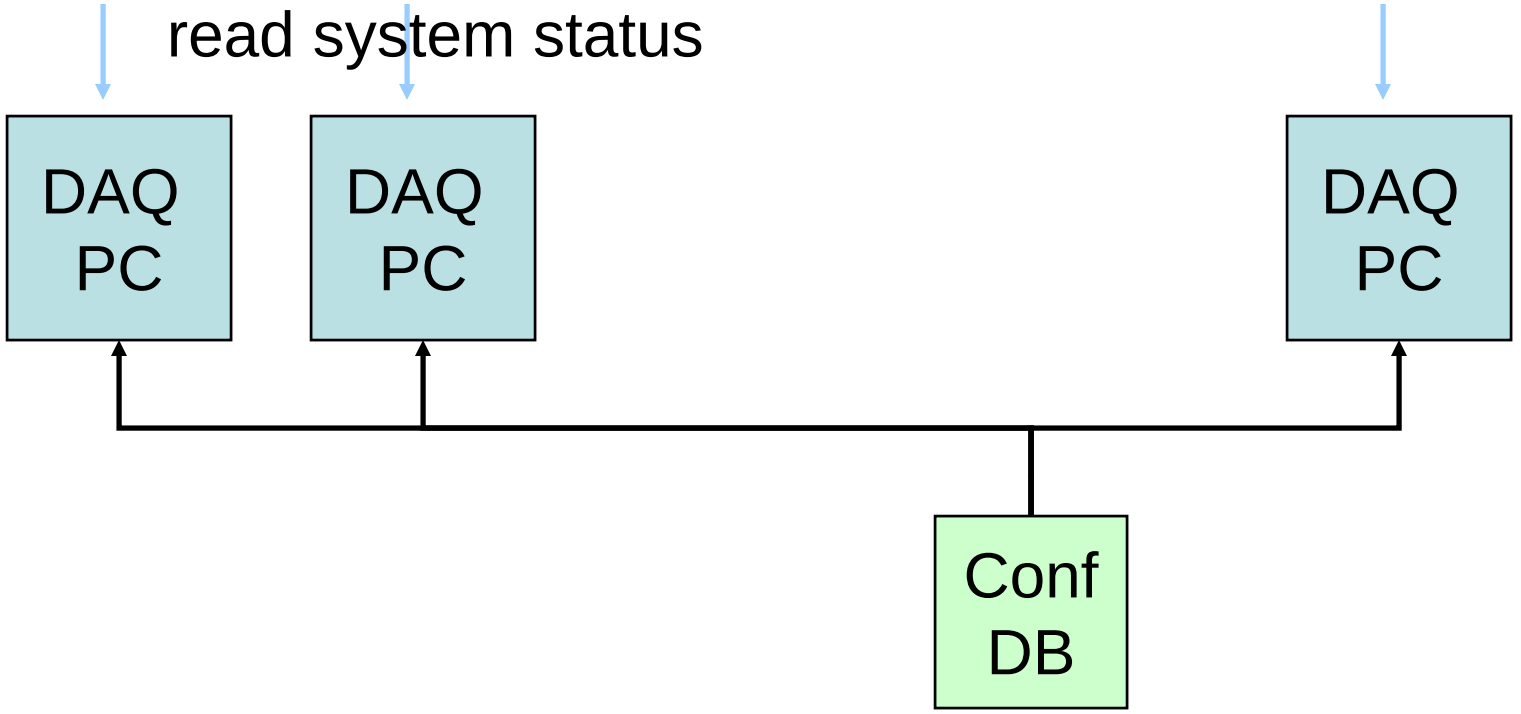
State Analysis



Transition: Handshake

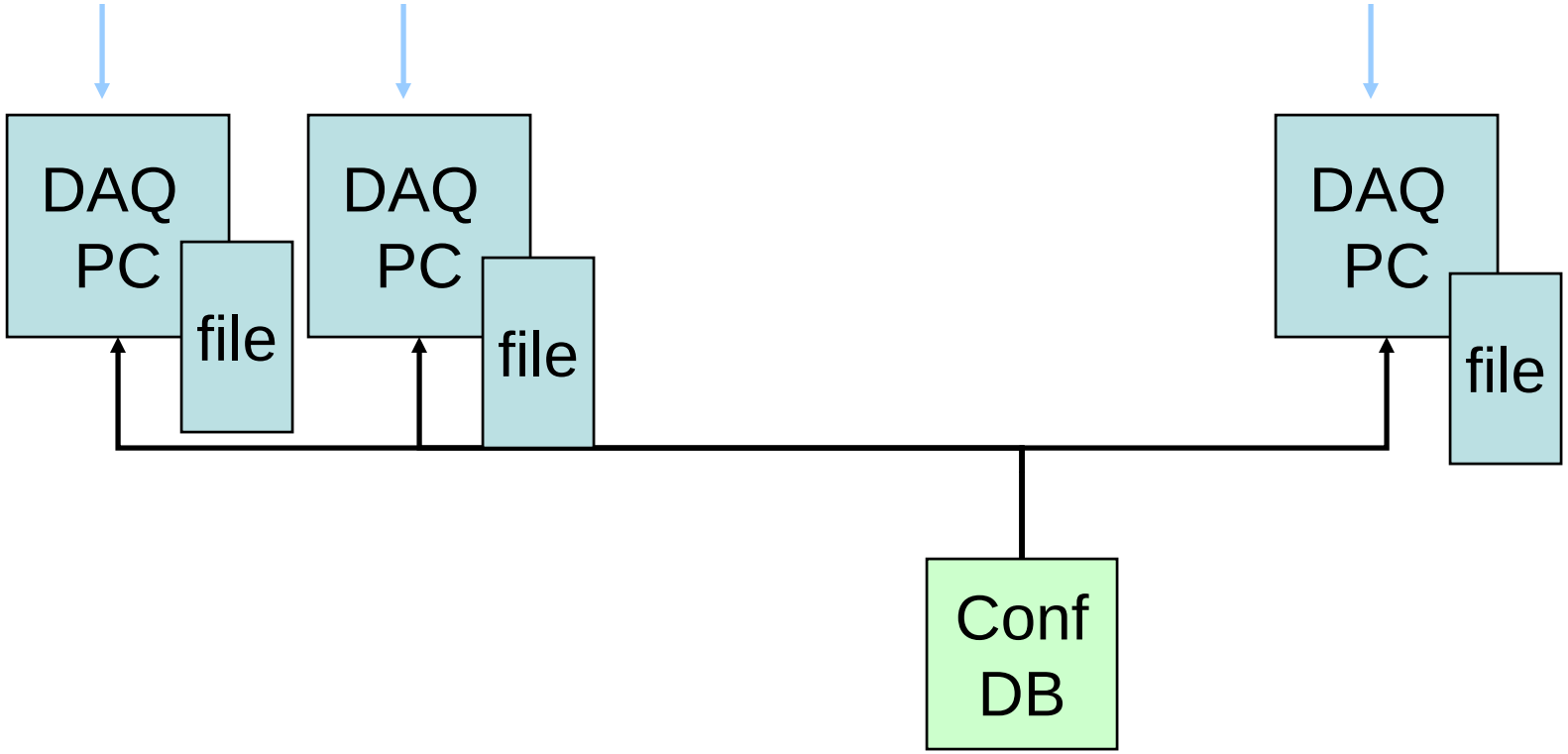


Transition: StartRun



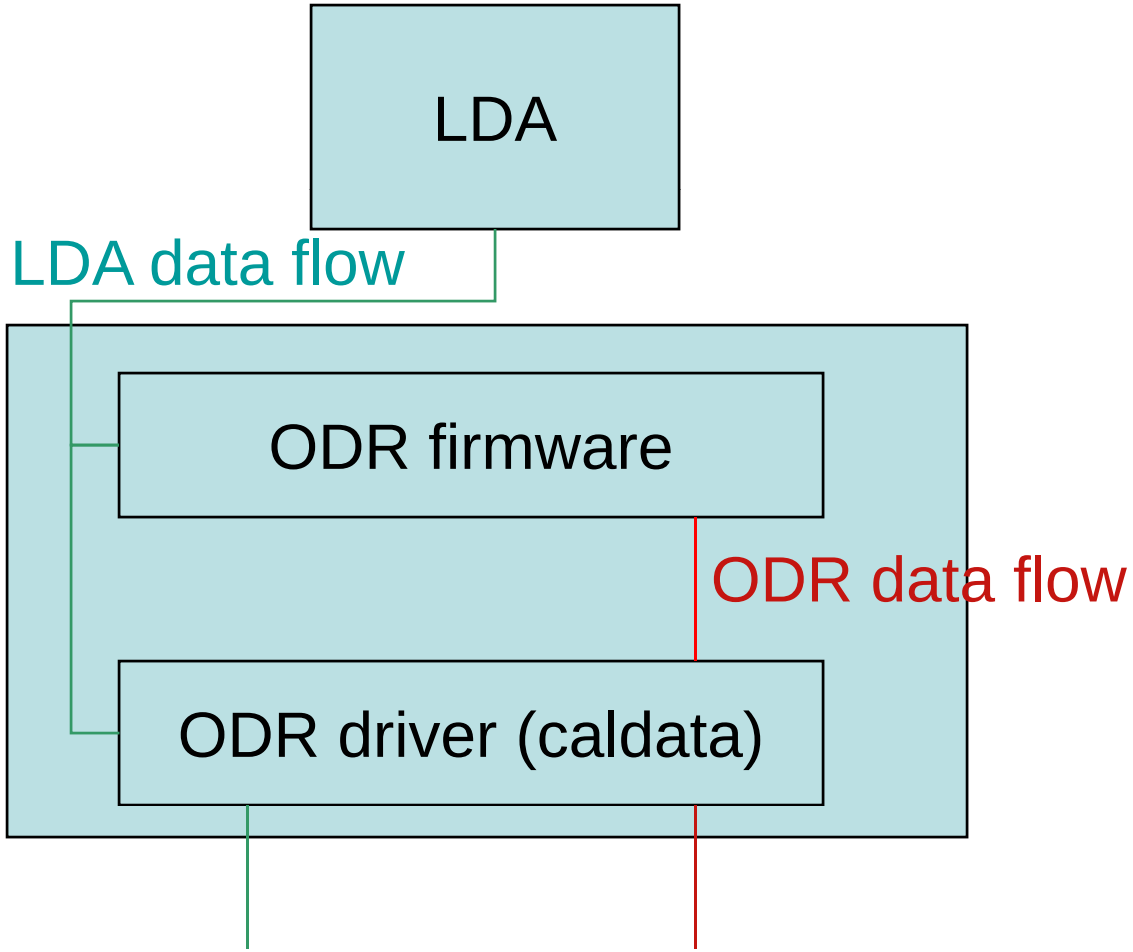
Send run number to ODR software,
Make new run number plus unique in file database
(filename = [run_number + unique identifier])
and fill in configurations

Transition: StartConfiguration



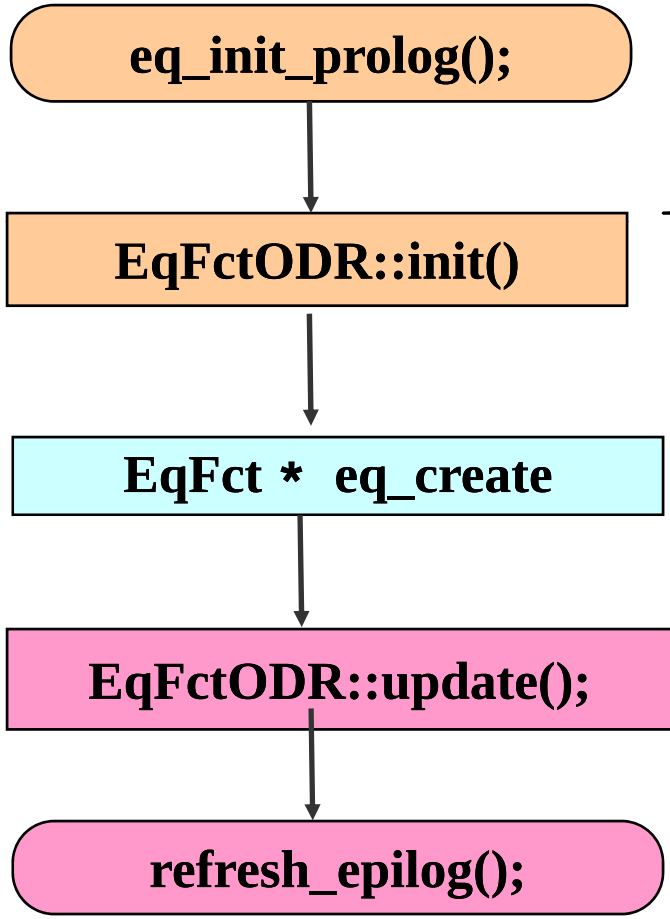
Extract conf files for all device servers from db,
Recheck that configuration has been received

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



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