# Central DCS / Run Control Concept

Sergey SERGEEV

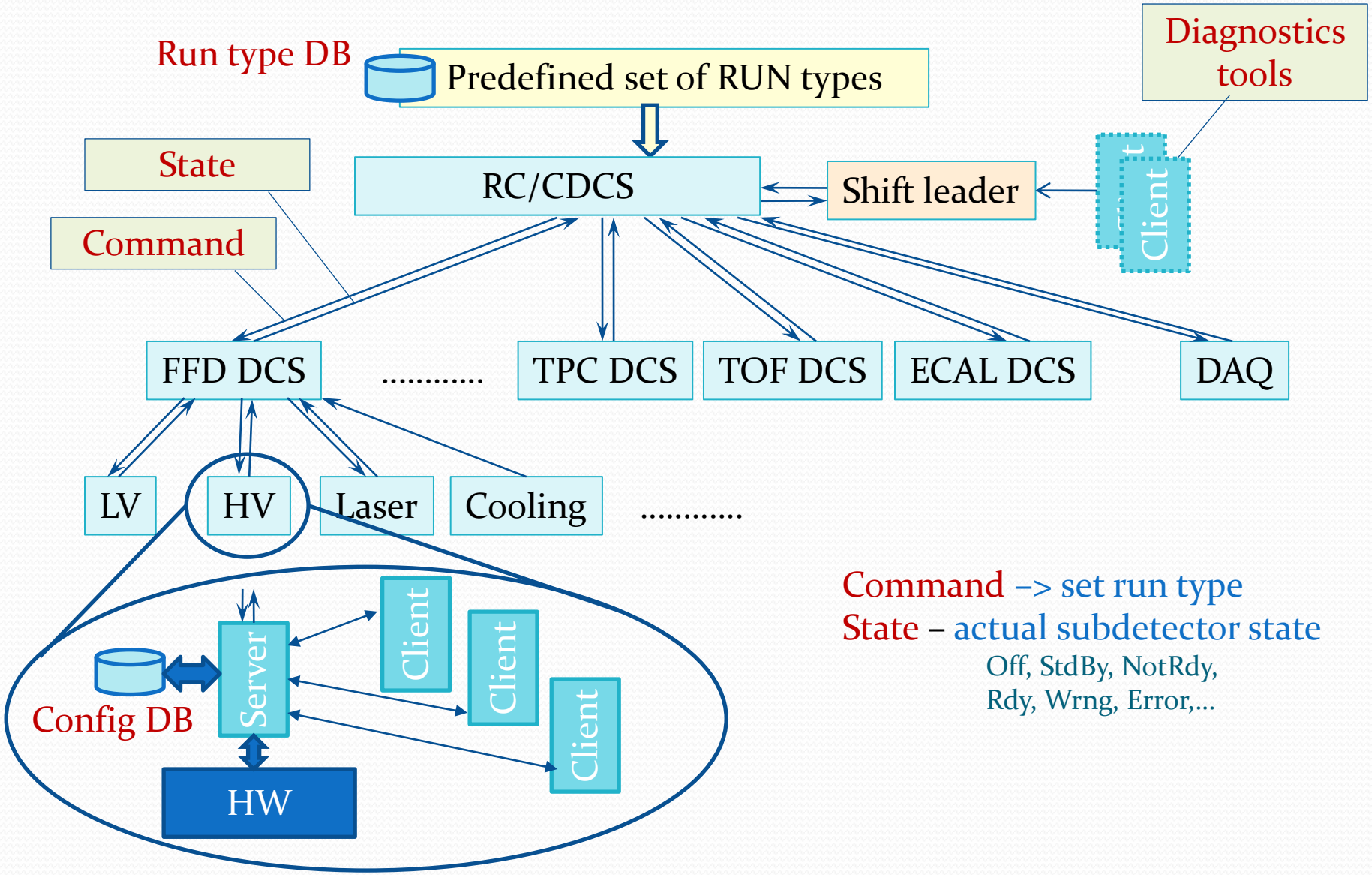
XIII MPD Collaboration meeting

Apr. 23-25, 2024

# Concepts

- Modular, used defined interface (DIM + defined message format).
- Extendable
- Modules could be replaced/modified
- As simple as possible (KISS – **K**eep **I**t **S**tupidly **S**imple) and transparent
- For MPD experiment only (not generic)
- Based on DB to be used by MPD (PostgreSQL ?)

# Experiment DCS structure




Command –> set run type

State – actual subdetector state

Off, StdBy, NotRdy,
Rdy, Wrng, Error,...

# CDCS interface I

- RC/CDCS subscribes to published by subdetectors **state** InfoItems with names

  MPD_DCS_State/<subdetector name>
- Run configuration contains subdetectors list used in a run
- RC/CDCS sends run type name (text) to all CommandItems of subdetectors being in a list. CommandItems should have a name like

  MPD_DCS_IniCmd /<Subdetector name>

# CDCS interface II

- Each subdetector DCS root node could (should?) have a CommandItem with name

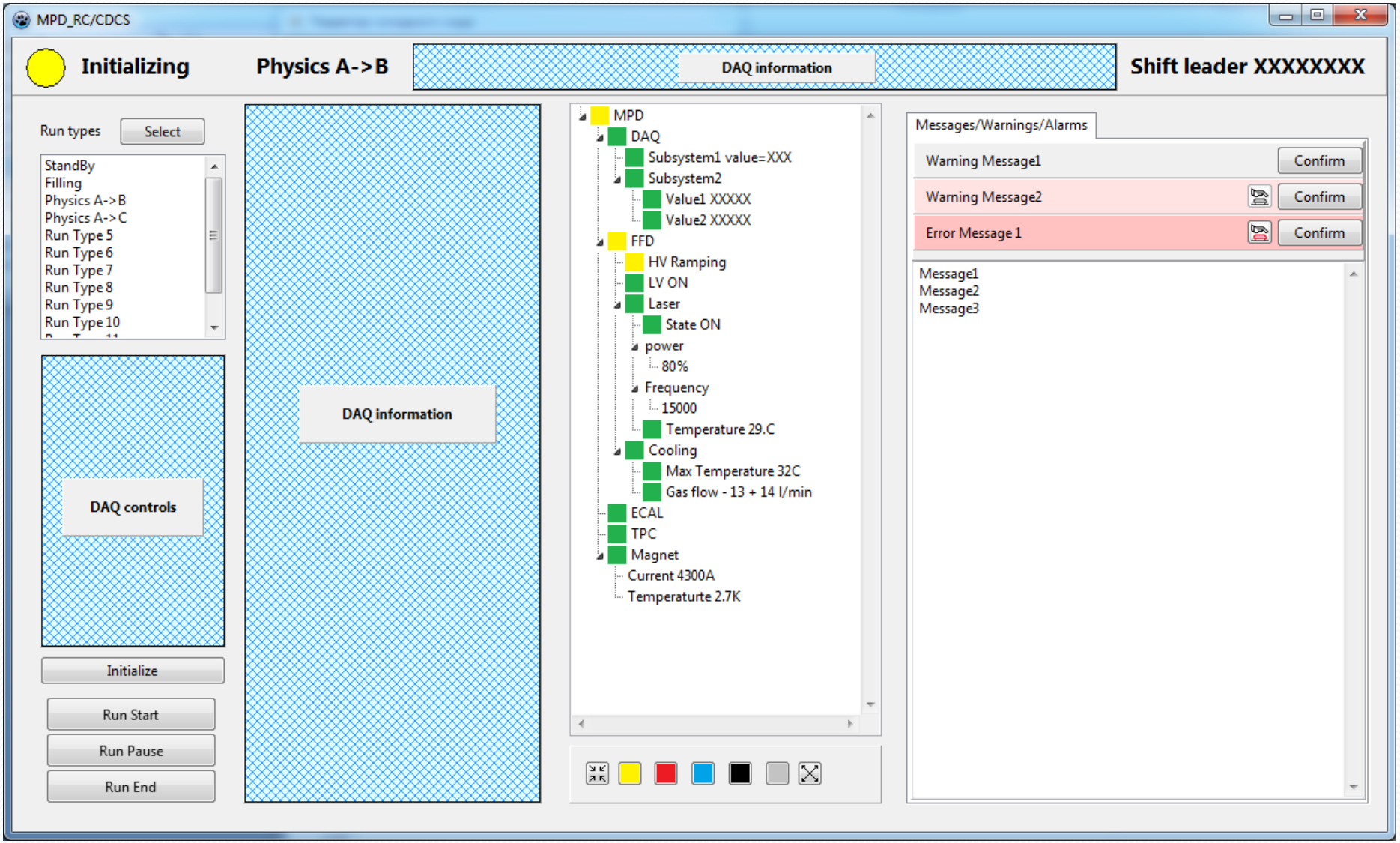
  MPD_DCS_DisplayCmd/<subdetector node name>

  A command received by this CommandItem should start diagnostic tool (see below)
- DAQ should have additional Info/Command items to provide vital information to/from the RC/CDCS (to be discussed with DAQ team)

# CDCS interface III

- RC/CDCS has a CommandItem MPD_DCS_Messages to receive messages from subsystems/subdetectors
- Format of message should be like (to be discussed)
  - <subdetector name>_<severity level>_<message text>
    - <severity level> defines a way to process the message
      - 0 -> just to show in a window. Could be scrolled by messages arriving later
      - 1 -> stays at the screen until confirmed
      - 2 -> stays at the screen until confirmed + sound alarm if not confirmed during defined time (1 min as an example)
      - 3 -> stays at the screen until confirmed + instant sound alarm
- All messages have a text content

# How it could look like



# States and colors

- State=-1, Item does not have a state, no color to be displayed
- State=0, OFF - any of sub-elements does not respond
- State=1, StdBy – any of sub-elements is in stand-by mode
- State=2, NotRdy – any of element is in transition state (Time-out should be implemented)
- State=3, Ready – all elements are OK
- State=4, Wrng – any of elements is in Warning state
- State=5, Error – any of elements is in Error state
- State=6, Ignrd – node in Partitioned state

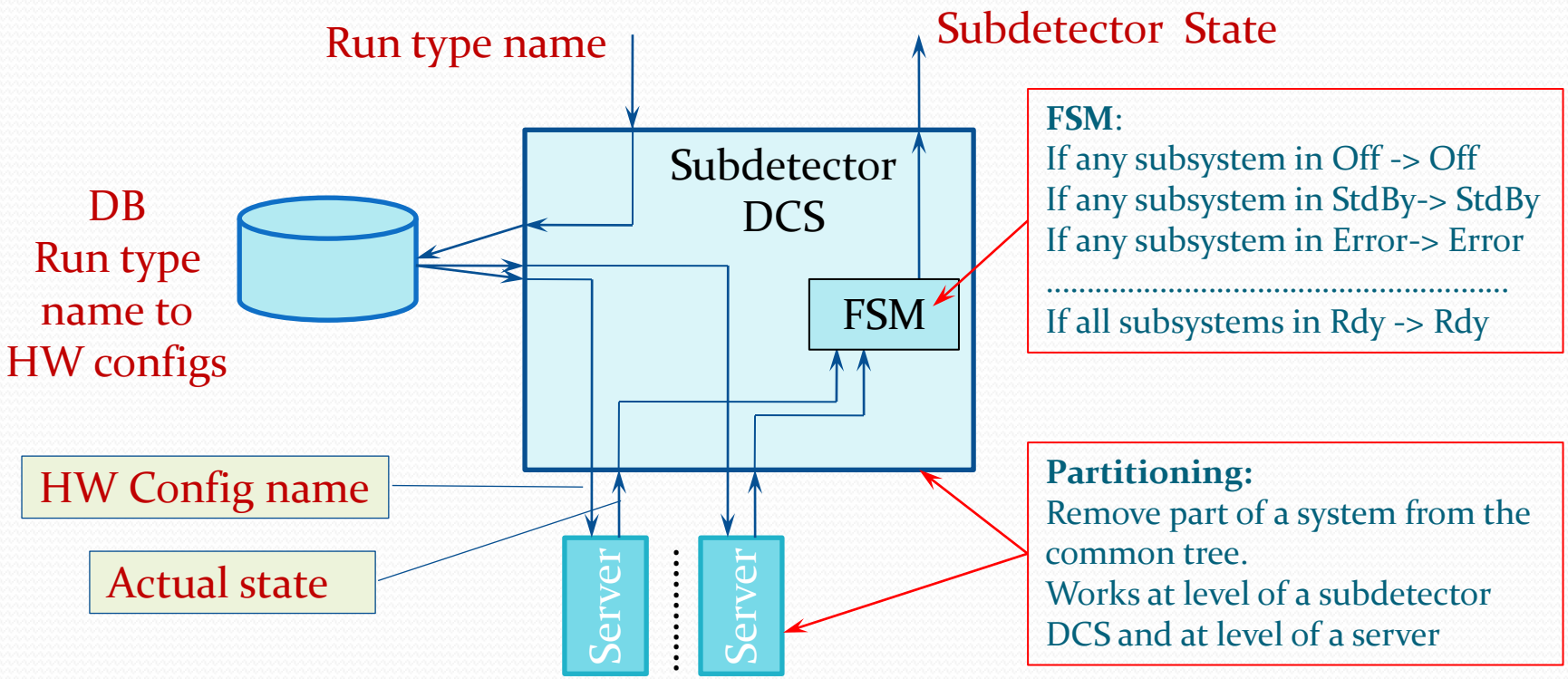
# Extended display (to be discussed)

- A subdetector should provide a set of diagnostics tools started by a CommandItem

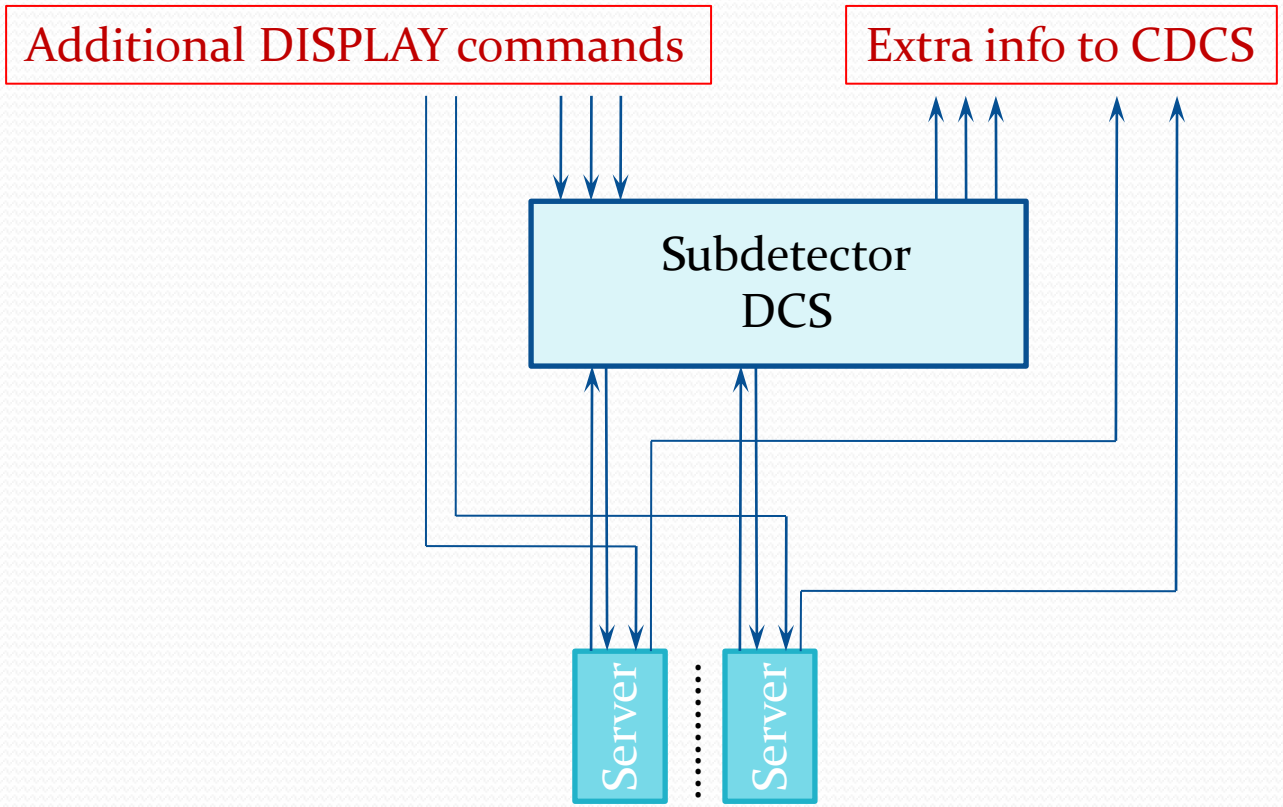
  MPD_DCS_DisplayCmd/<subdetector name>

- This should be an application running at the CDCS PC(?) or a web-page running AJAX script (?). The web server could be provided by a CDCS. Page content should be developed by the subdetector team and could be located at a common disk space
- Start parameters are defined in the CommandItem command content

# Subdetector DCS (obligatory)

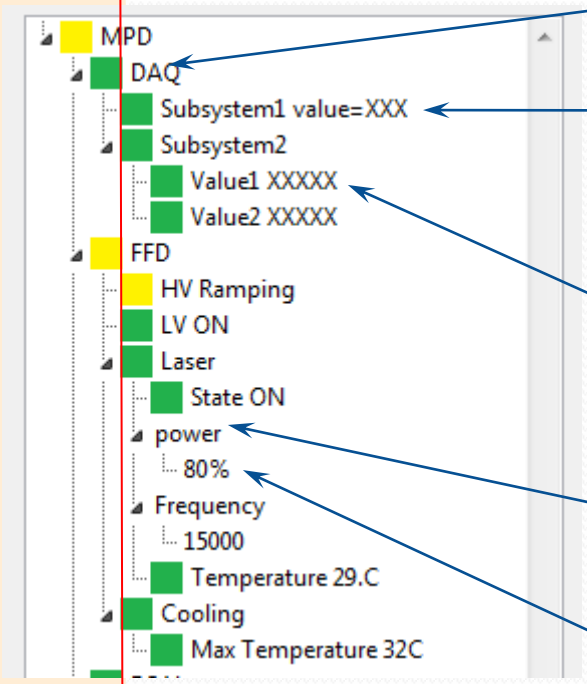


# Subdetector DCS (optional)



# Extra parameters interface

MPD state is built using 1-st level nodes




"MPD_DCS_State/DAQ", content="3"

"MPD_DCS_State/DAQ/Subsystem1", content="3_value=XXX"

"MPD_DCS_State/DAQ/Subsystem1/Value1", content="3_XXXXX"

"MPD_DCS_State/FFD/Laser_power", content="-1"

"MPD_DCS_State/FFD/Laser_power/", content="-1_80%"

Obligatory

Extra info to display in the tree, defined by subsystem/subdetector

# Thank you for attention

DIM

Distributed Information Management System

- Based on TCP/IP sockets
- Developed in 80-s at DELPHI experiment
- Main feature – converts hardware address space (IP + port) to logical name address space -> components could migrate on computers
- Event-driven (real-time)
- Could have multiple name domains
- Open source
- Works on Windows, VMS, several Unix flavors (Linux, Solaris, HP-UX, Darwin, etc.) and the real time OSs: OS9, LynxOs and VxWorks
- Libraries for C, C++, Jawa, Delphi (Lazarus), Python
- ***A lot of debugging tools***
- See https://dim.web.cern.ch/

# How it works

- At startup every Server registers its **services** at DNS (DIM name server)
- Any Client could request a connection to a **service**, after that the client receives actual IP and port number for requested service (performed inside the DIM library)
- DIM establishes a TCP/IP connection Server-Client
- Further communication is done directly via TCP/IP sockets (Server-Client only)
- **Pleasant bonus**: If a Service contains a "**description**" then debugging tools could interpret TCP/IP buffer content to display in a human-readable way

# Source code C++

```
DimService servint("TEST/INTVAL",ival);
DimService new_servint("new_TEST/INTVAL",boolval);
DimServer::start(newDns, "new_TEST");
```

Create 2 **Info** items

……

```
if(!boolval) boolval=1; else boolval=0;
ival++;
```

**New** data ready

……

```
servint.updateService();
new_servint.updateService();
```

Publish **New** data