Trigger-DAQ and slow control systems in the Mu2e experiment

Antonio Gioiosa

Università di Pisa, INFN Pisa

TIPP 2021

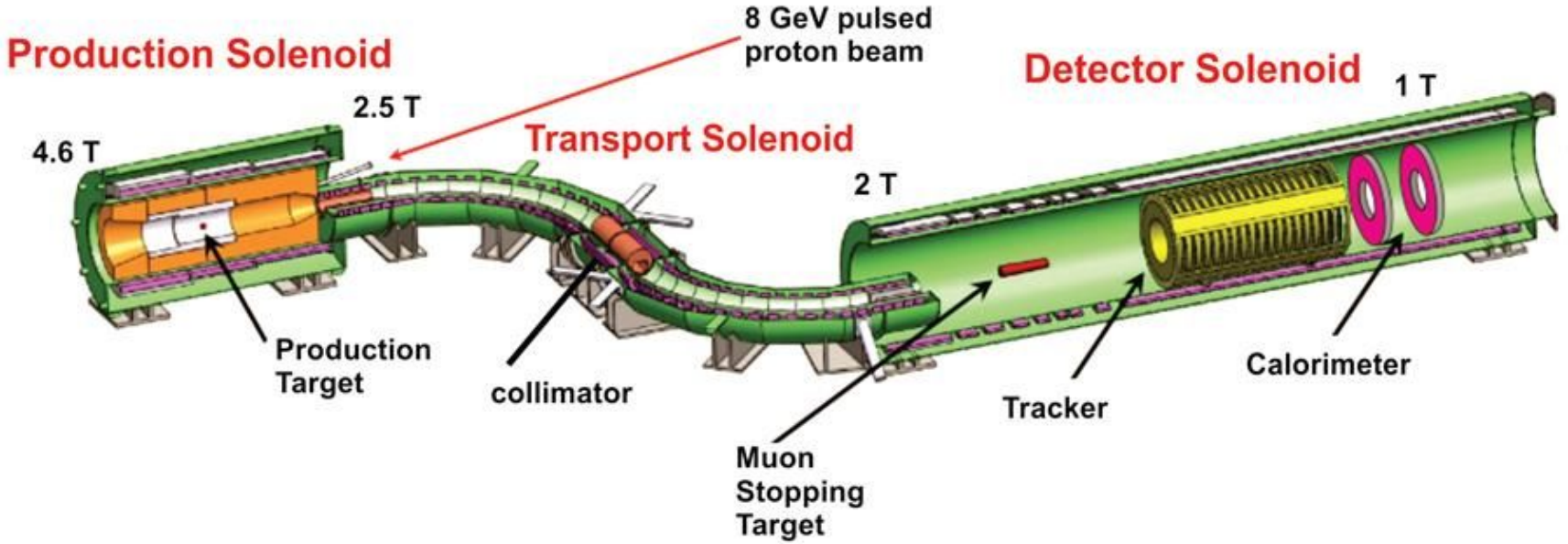
May 25, 2021

The Mu2e Experiment at Fermilab

$$(\mu^- + \mathrm{Al} \rightarrow e^- + \mathrm{Al})$$

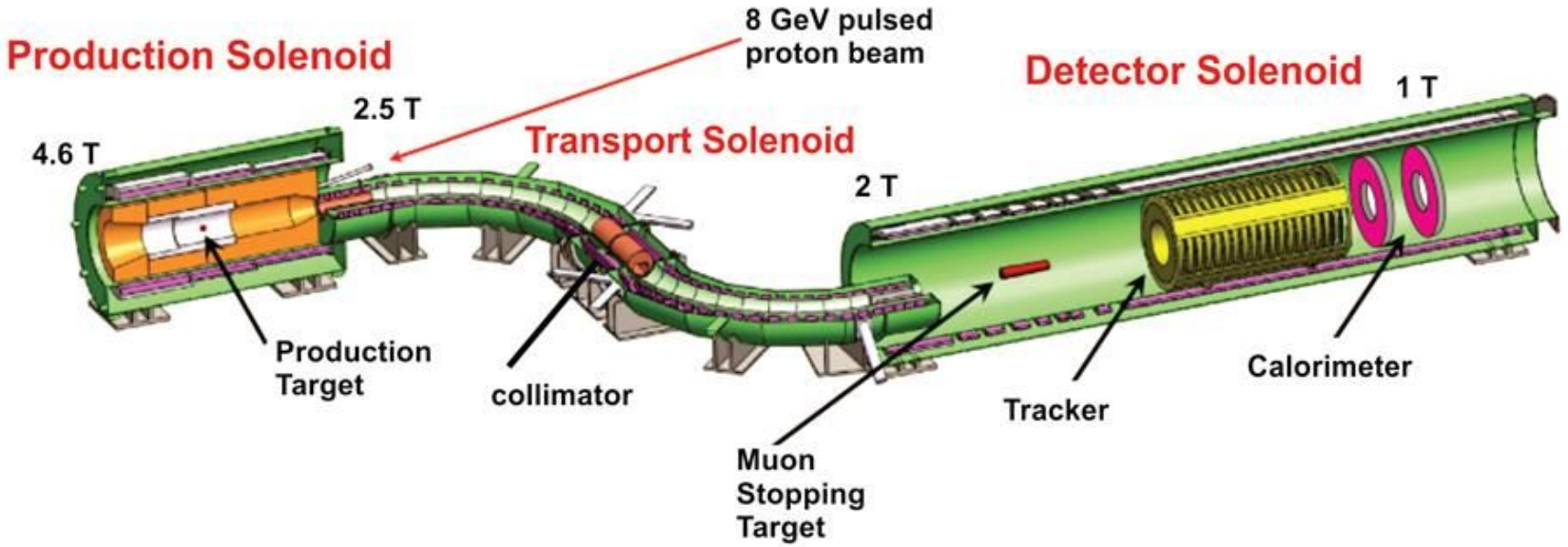
With the expected experimental sensitivity, Mu2e will improve the SINDRUM II limit ($7.0 \cdot 10^{-13}$) of four orders of magnitude

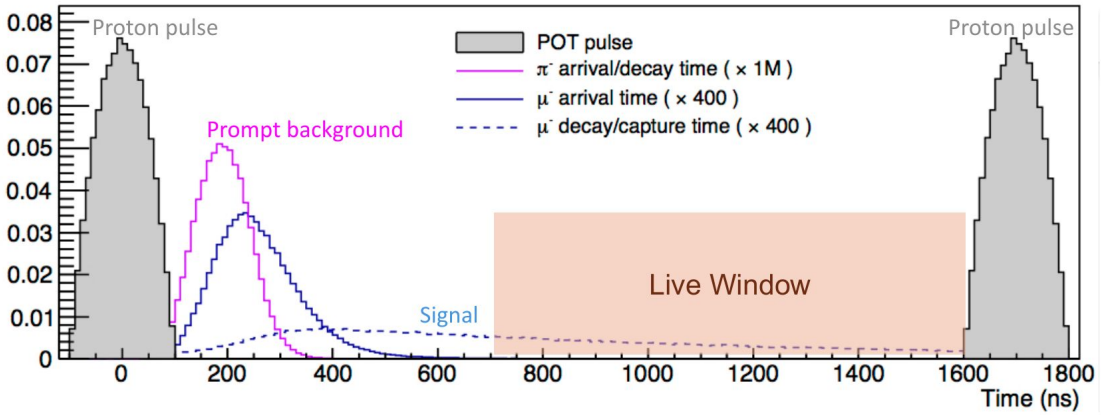
(Mu2e intends to reach a single event sensitivity of $3.0 \cdot 10^{-17}$, assuming we will run for three years, with $3.6 \cdot 10^{20}$ protons, with a run time of $6.0 \cdot 10^{7}$ s, requiring a background level below 1 event)



The Mu2e Experiment at Fermilab

The signal we are looking for is a delayed monoenergetic electron with an energy of just under 105 MeV (muon mass)



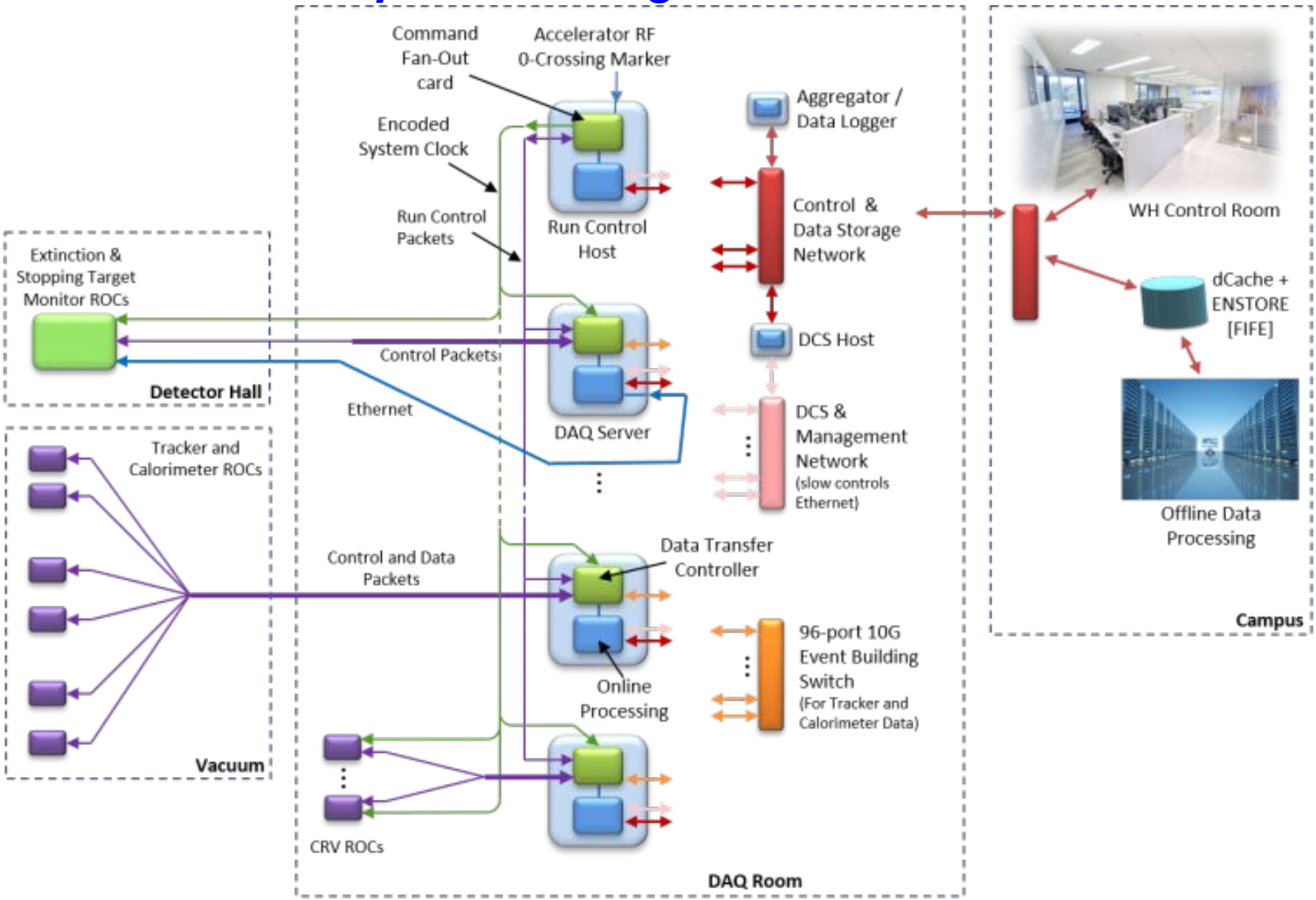


Mu2e TDAQ and Slow Control integration

Summary:

- Mu2e TDAQ components Diagram
- Mu2e Timing Distribution
- Mu2e TDAQ Readout scheme
- Online DAQ (***otsdaq***) overview
- Slow control and its integration in ***otsdaq***
  - Monitoring and Slow Controls GUI
  - Slow Controls Integration with ***otsdaq*** State Machine and Alarm handling
- Conclusions

Mu2e TDAQ components Diagram

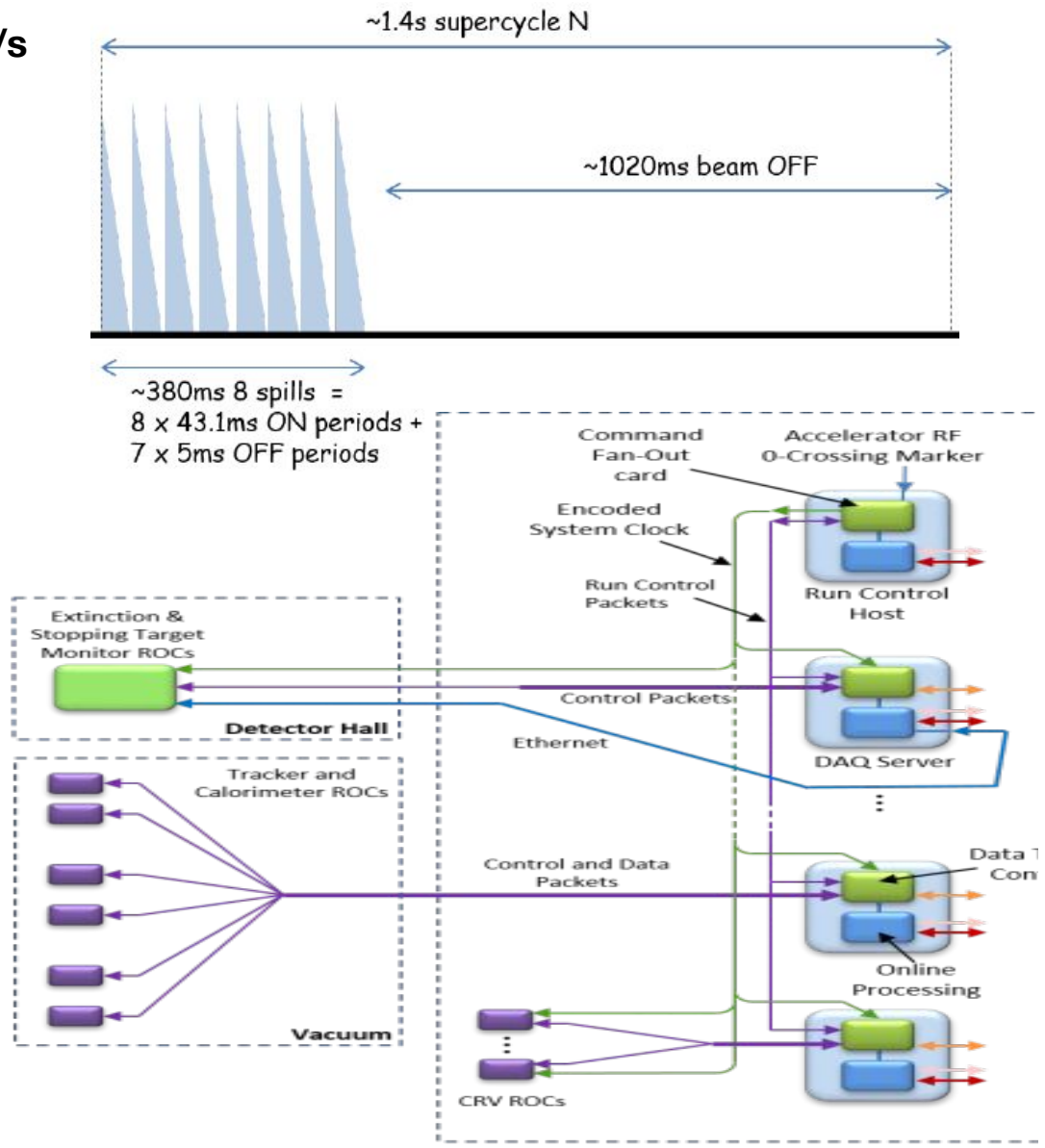


Mu2e Timing Distribution

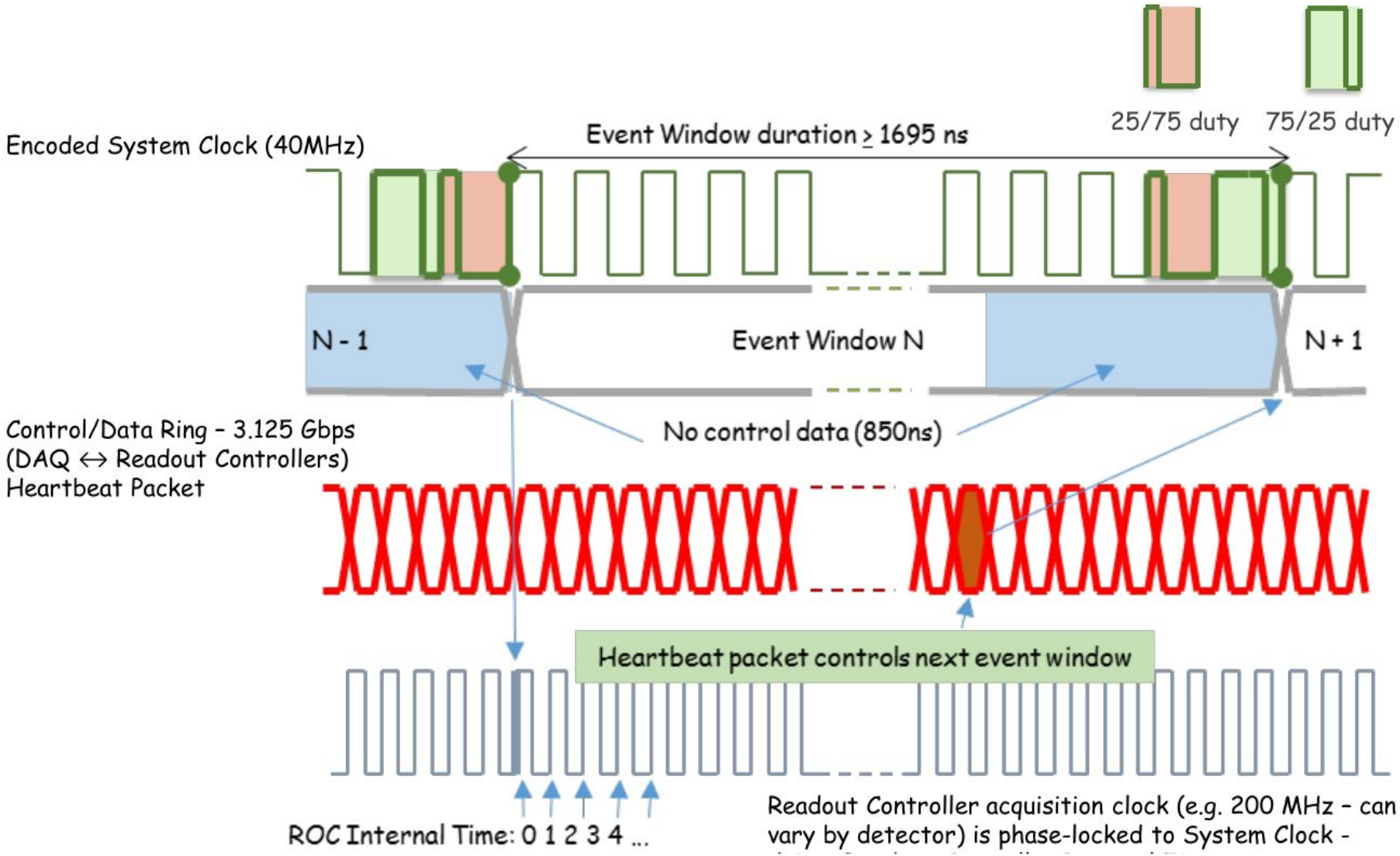
Requirement is to process 200K events/s

- Mu2e Runs are broken up into contiguous Event Windows
- Experiment defined Run Plan is coordinated by the Command Fan-Out Card (CFO)
- The System Clock (40MHz) and Event Window markers originate at the CFO ...and are distributed to ROCs:

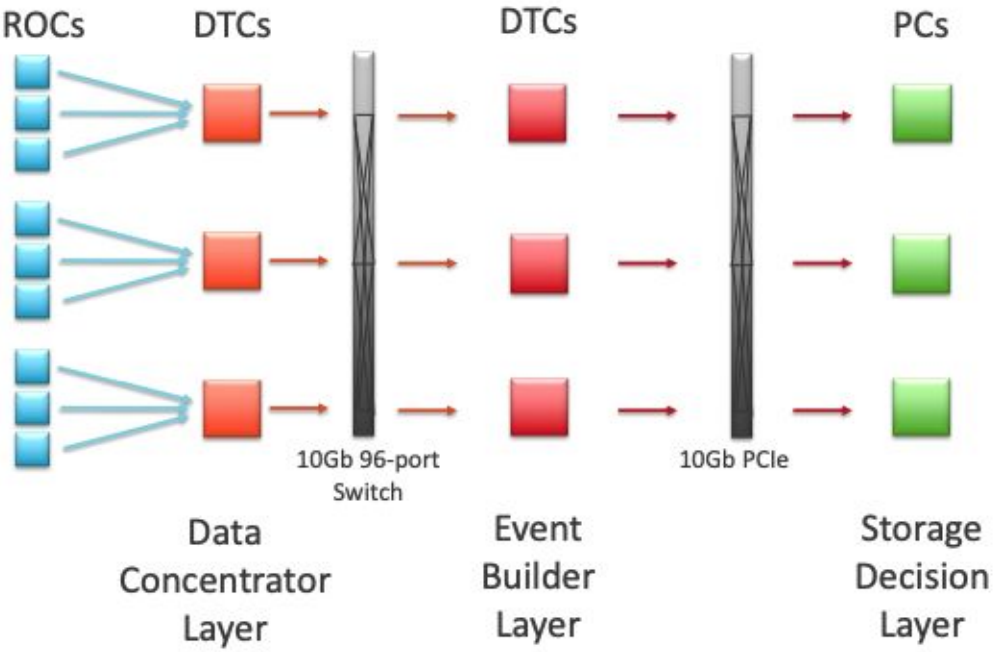
1. CFO distributes System Clock and Event Windows to DTCs with fixed latency
2. DTCs distribute System Clock and Event Windows to ROCs with fixed latency
3. ROCs respond to Data Requests



Mu2e Timing Distribution



TDAQ Readout scheme



- 396 ROCs 69 DTCs (Kintex-7) for data readout and event building
- Large front end buffers to average over long off-spill time
- 800 threads on 40 nodes for HLT → ~5 ms per event
- ~40 GB/s data read out to storage decision layer, ~280 MB/s written to disk

High Level Trigger Software

Mu2e Online DAQ solution: *otsdaq*

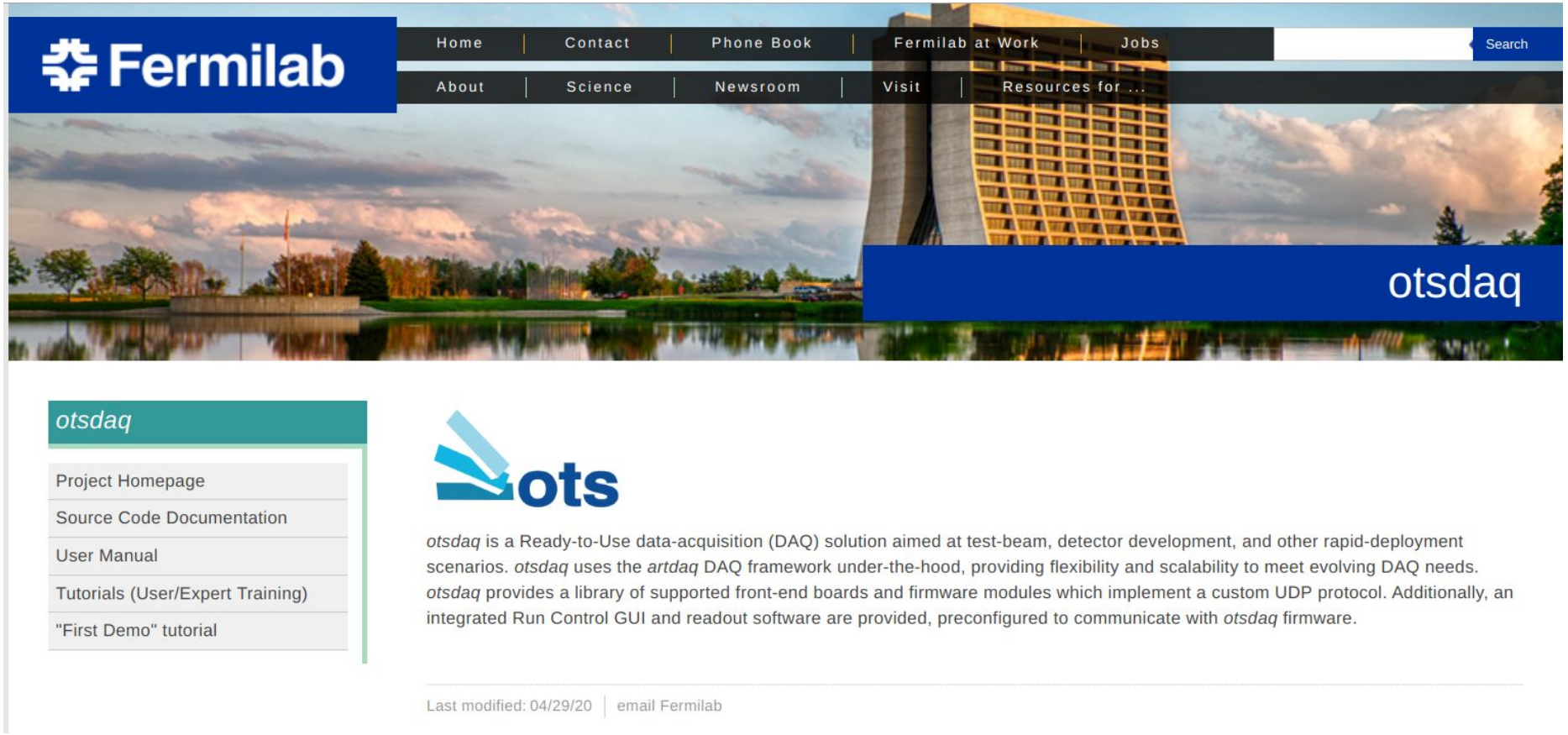
otsdaq overview

Acronym for "off-the-shelf data acquisition."

- ***otsdaq*** is a Ready-to-Use data-acquisition (DAQ) solution aimed at test-beam, detector development, and other rapid-deployment scenarios
- it uses the ***artdaq*** DAQ framework under-the-hood, providing flexibility and scalability to meet evolving DAQ needs
- ***otsdaq*** provides a library of supported front-end boards and firmware modules which implement a custom UDP protocol
- Developments are in two directions: **server** side and **web** side.
- An integrated Run Control GUI and readout software are provided, preconfigured to communicate with ***otsdaq*** firmware

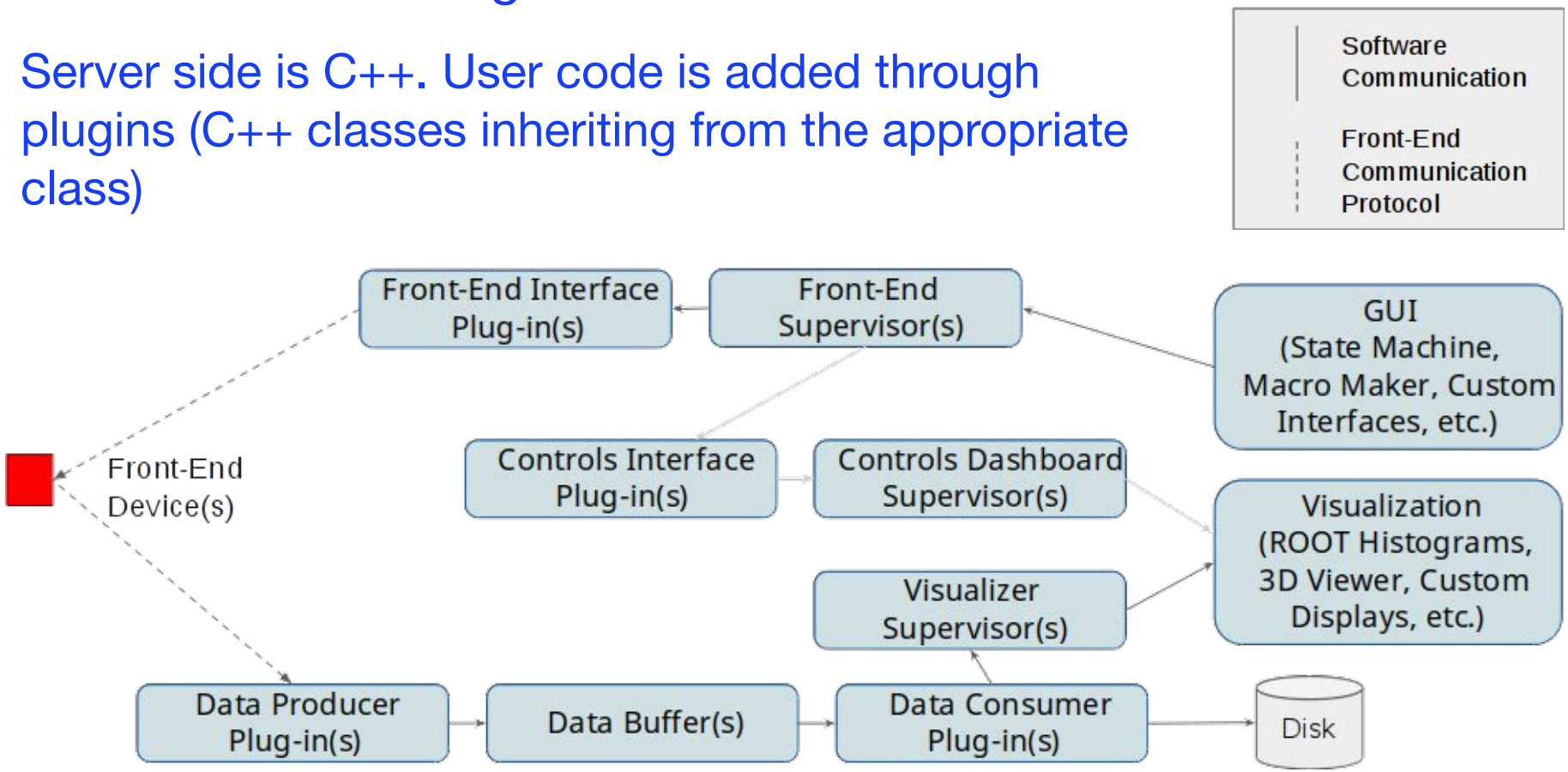
otsdaq overview

More info at ***otsdaq*** web page https://otsdaq.fnal.gov/



Data Flow Block Diagram

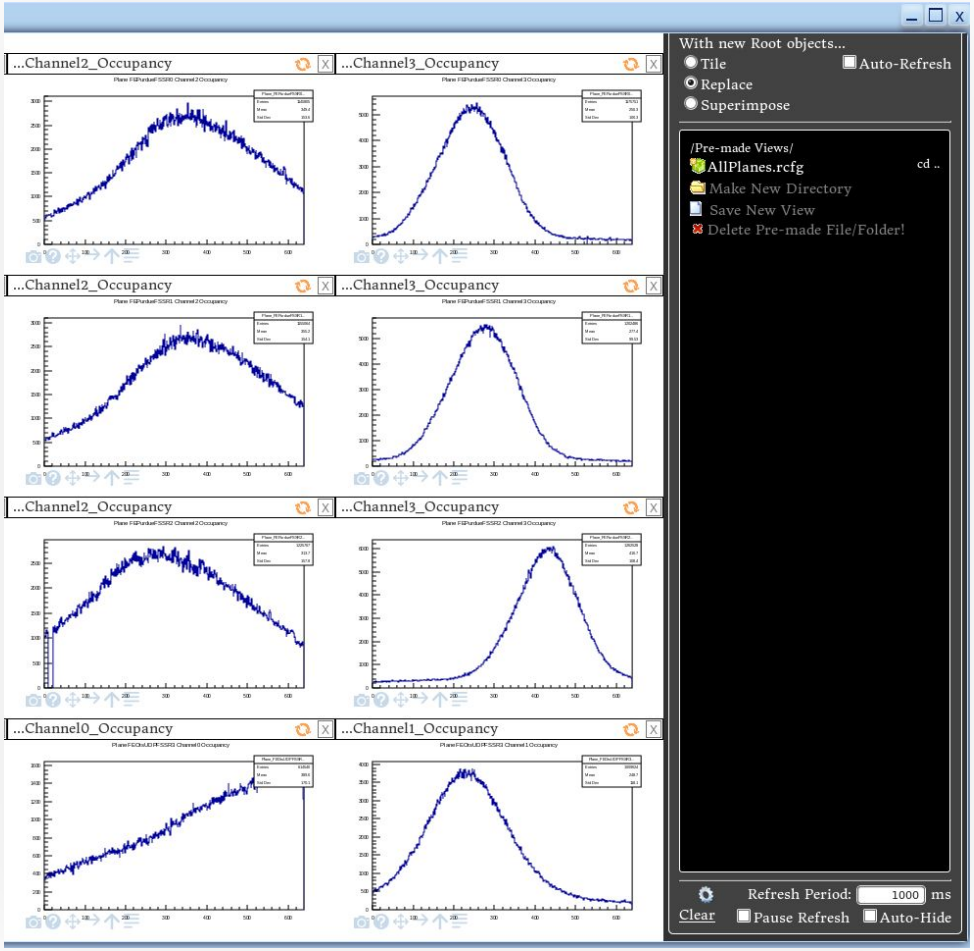
Server side is C++. User code is added through plugins (C++ classes inheriting from the appropriate class)



Web side is HTML and JavaScript. User code is added in the form of web-apps through .html files (including the appropriate .js and .css files)

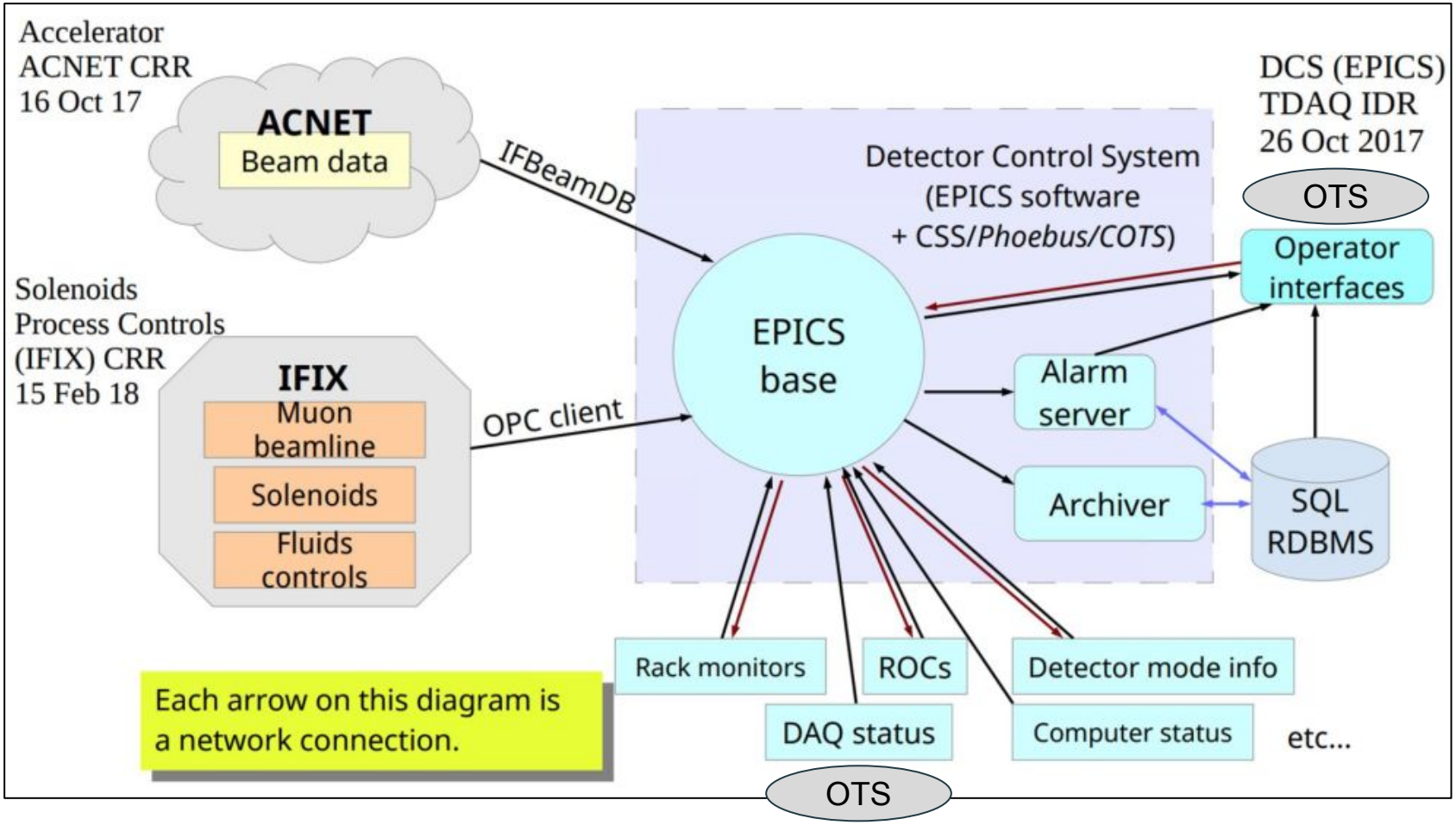
Data processing: Data Quality monitor GUI example

- Mu2e's event window data will be processed through artdaq modules
- Data processor and Data Quality Monitor DQM plugins are provided by otsdaq core
- DQM generates data products that are sent to an artdaq Dispatcher, which aggregates DQM metrics and presents them to a visualizer application

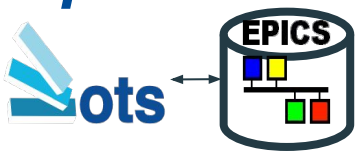


Slow Controls connection and EPICS plugin development in otsdaq

Experimental Physics and Industrial Control System



Slow Controls connection and EPICS plugin development in otsdaq

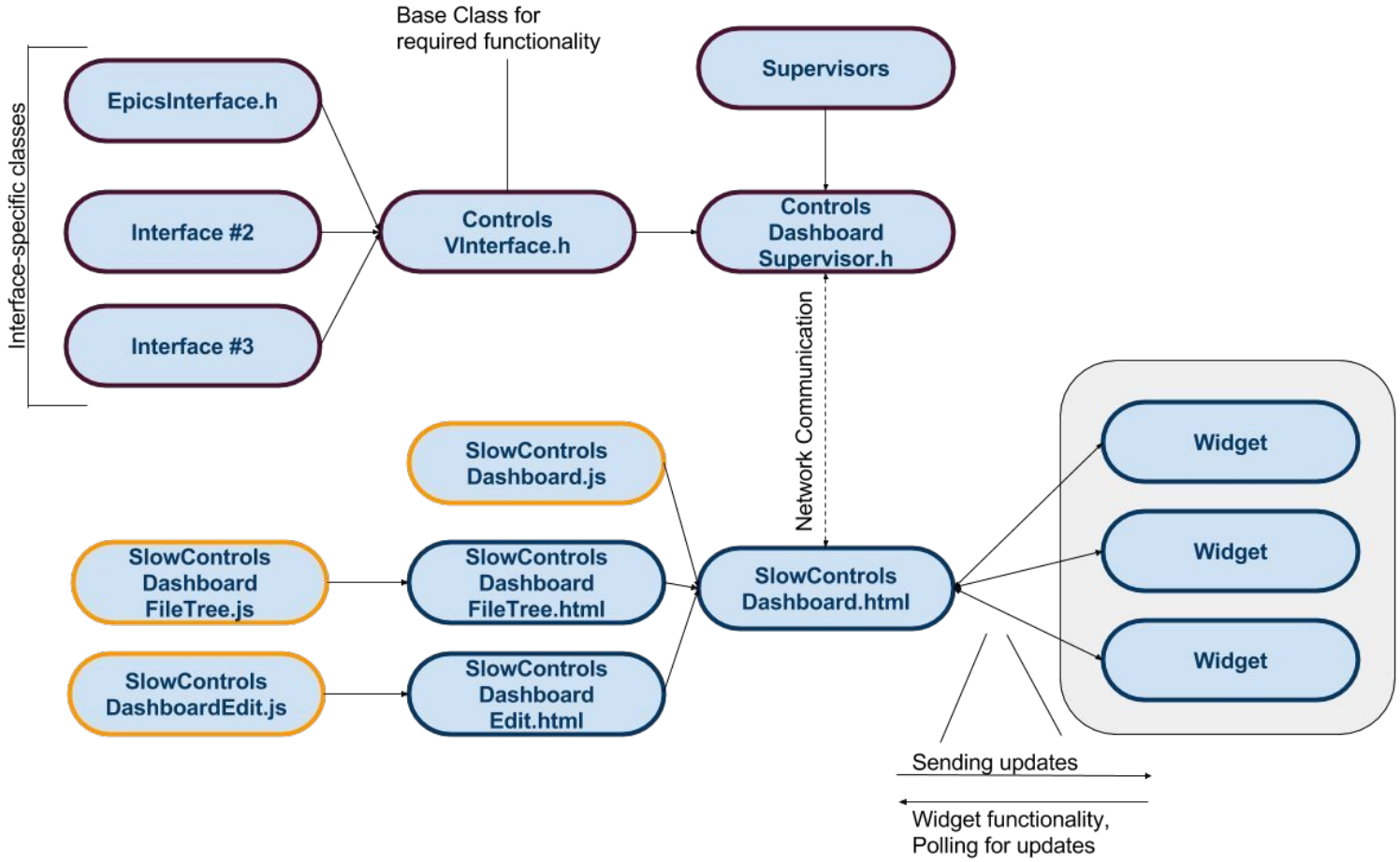


Channel subscription to EPICS uses Input Output Controller (IOC)

- integration of slow control in the online daq uses same Interface plugin for:
  a. Monitoring of all mu2e slow control channels
  b. Sending Process Variables (PVs) of DAQ hardware info as EPICS channels and PVs settings into EPICS databases
- The Interface plugin:
  a. Performs channel subscription to EPICS using Channel Access EPICS C++ libraries to send and retrieve slow control data information like: Value, Alarm (Status, Severity), Settings
  b. Uses Postgres database C++ libraries to set channels and retrieve channels and alarms histories from EPICS databases

Slow Controls Monitoring in otsdaq

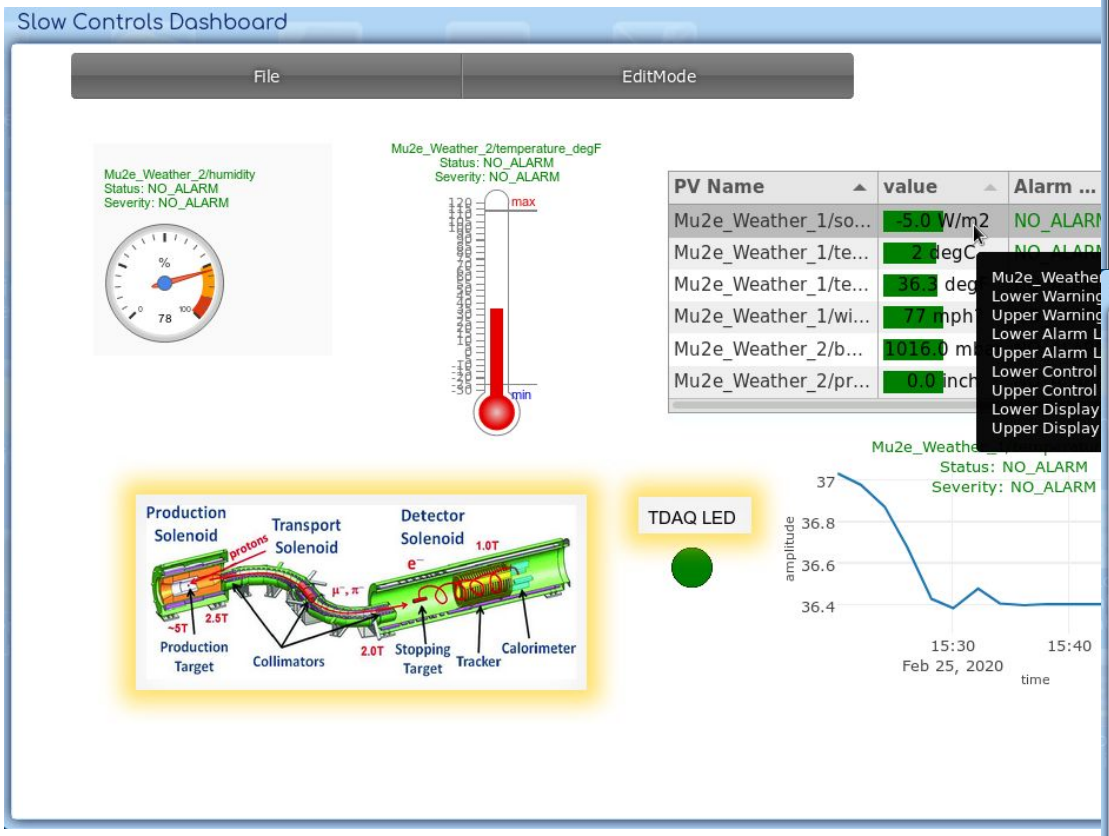
Slow Controls GUI Hierarchy



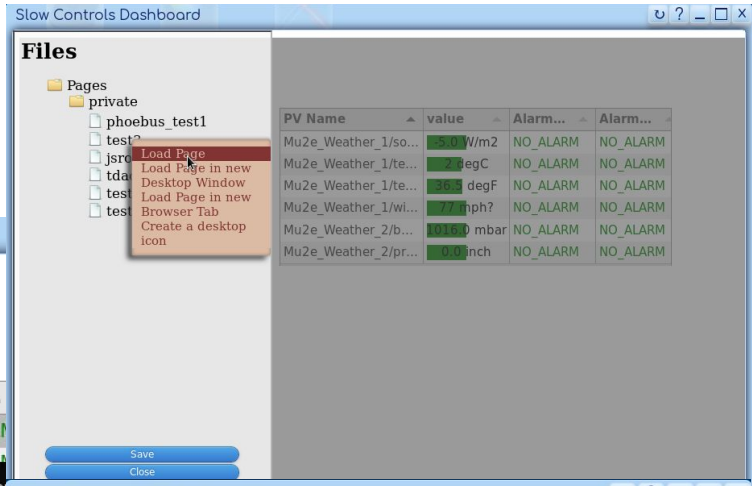
Slow Controls Monitoring GUI in otsdaq

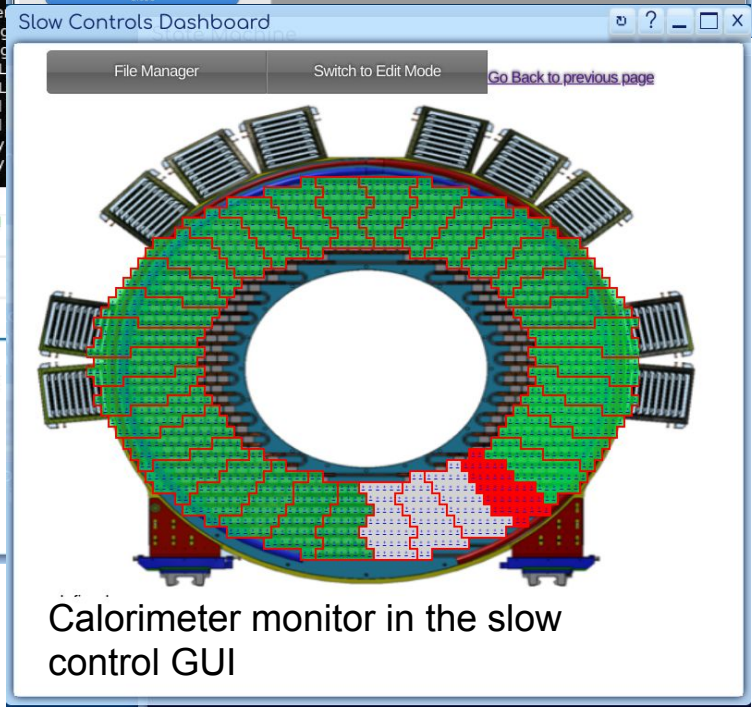
Examples

Example of loaded page



Example of page loading

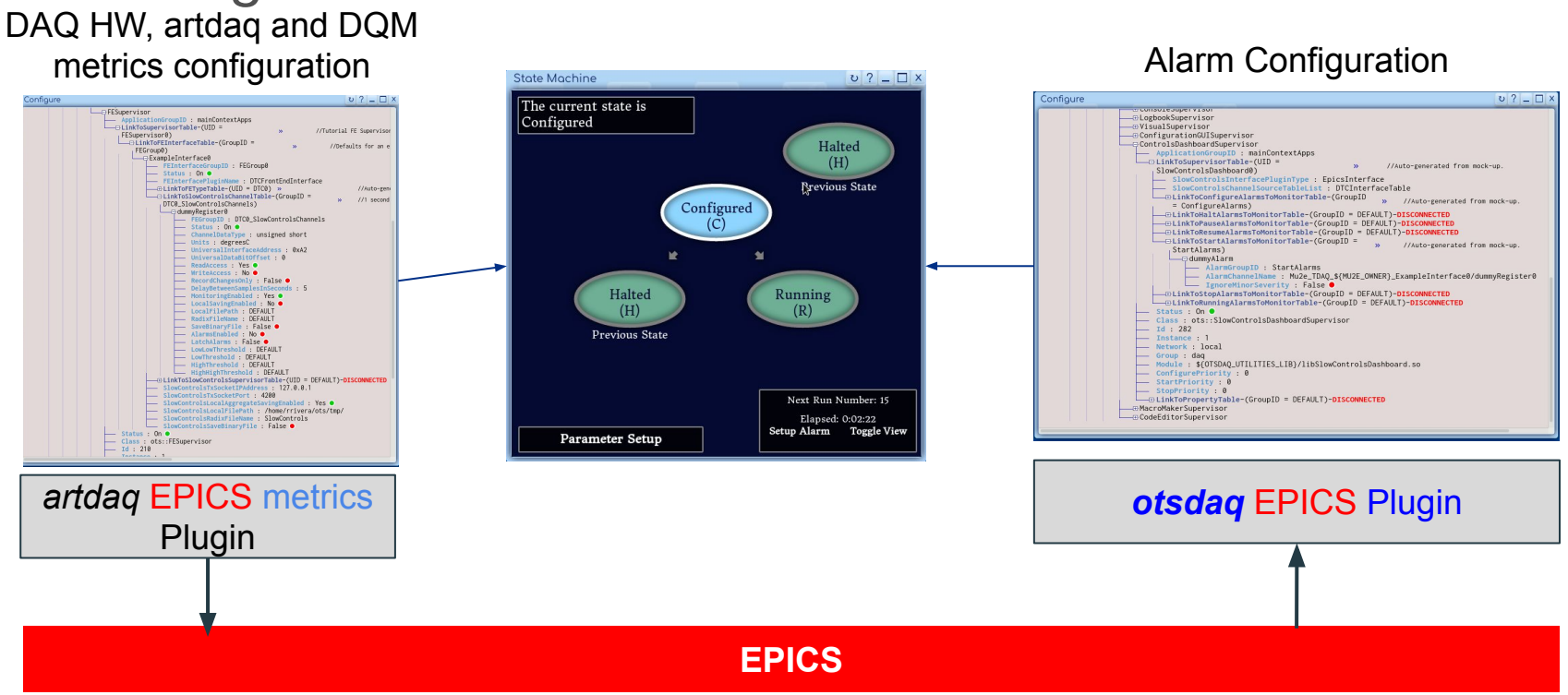




Calorimeter monitor in the slow control GUI

Integration with State Machine

- State Machine Configuration and data subscription to EPICS
- Alarm propagation (from EPICS) and *otsdaq* State Machine handling



Conclusions

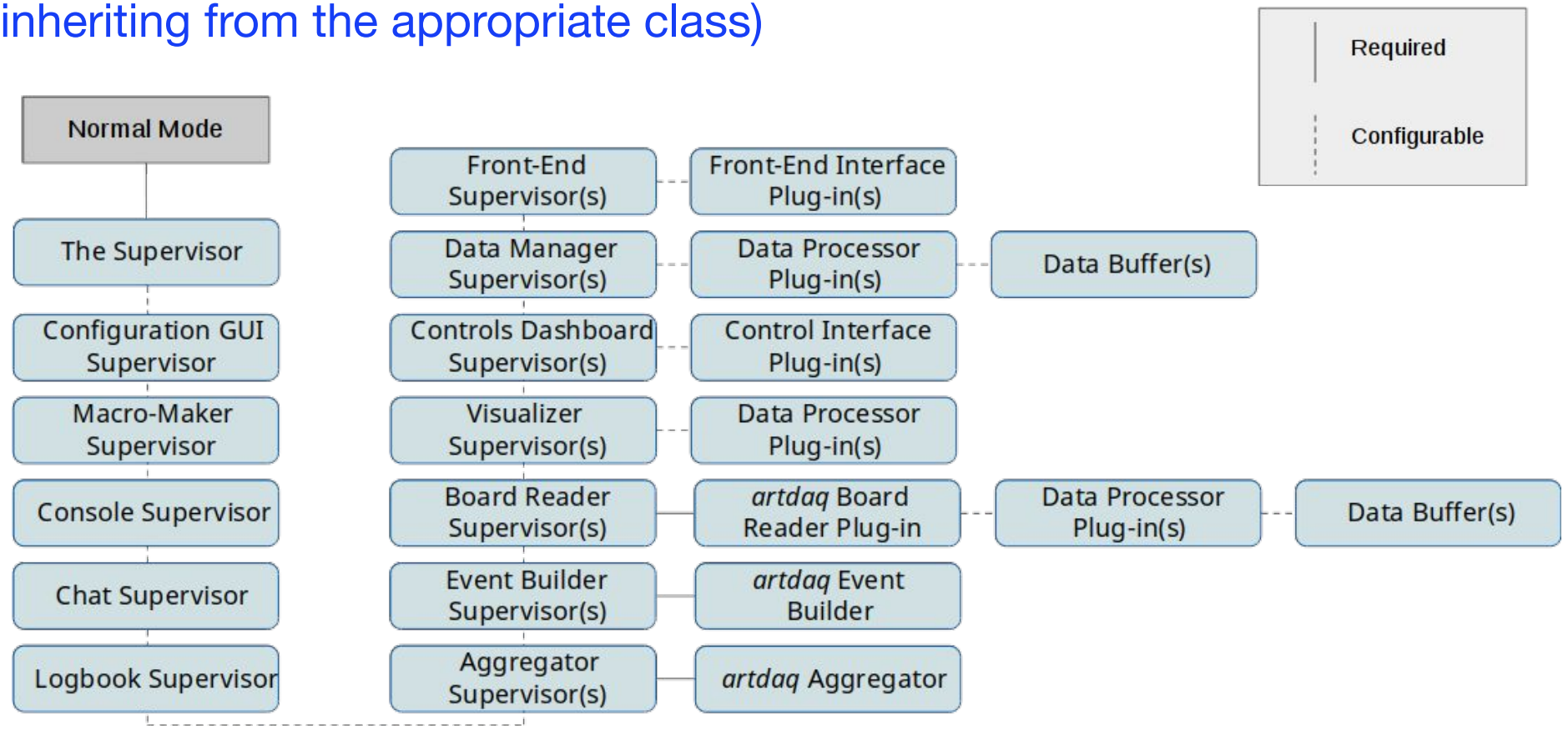
- Mu2e Experiment is under construction at Fermilab and will be ready for data taking in two/three years
- Mu2e TDAQ and slow control are in large part developed according to the requirements (200K events/s for data taking) and hardware tests are going on
- Slow control integration in the online DAQ system, *otsdaq*, provides an advanced slow controls monitoring, an interface to send *otsdaq* front-end DAQ hardware, data processing, and DQM slow controls information to EPICS, and a real configuration and Integration with the *otsdaq* State Machine

This work was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie Grant Agreement no 734303, 822185, 858199, 101003460

Backup Slides

otsdaq overview

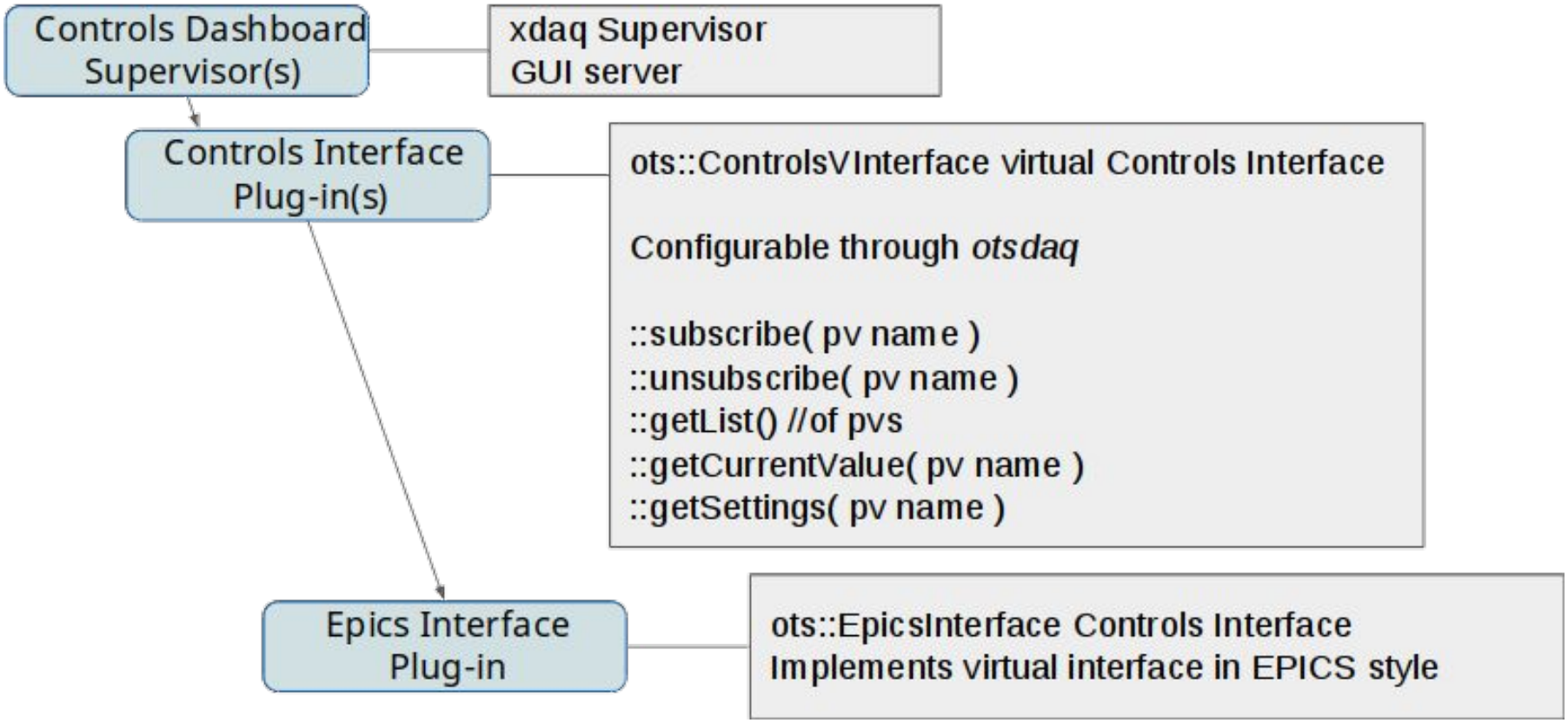
Server side is C++. User code is added through plugins (C++ classes inheriting from the appropriate class)



Web side is HTML and JavaScript. User code is added in the form of web-apps through .html files (including the appropriate .js and .css files)

Slow Controls Monitoring in otsdaq

Slow Controls C++ Hierarchy

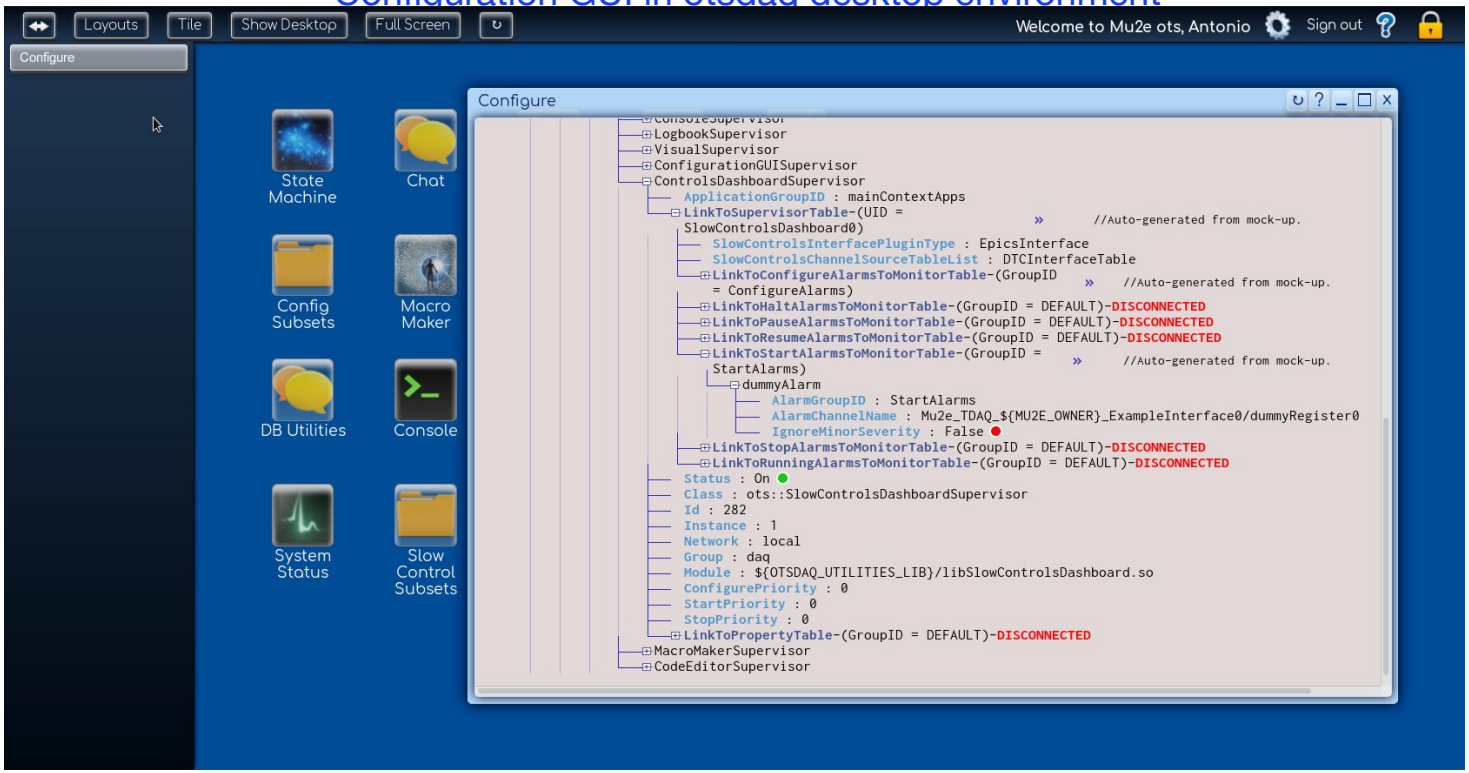


Slow Controls Monitoring in otsdaq

Configuring by specific tables in otsdaq

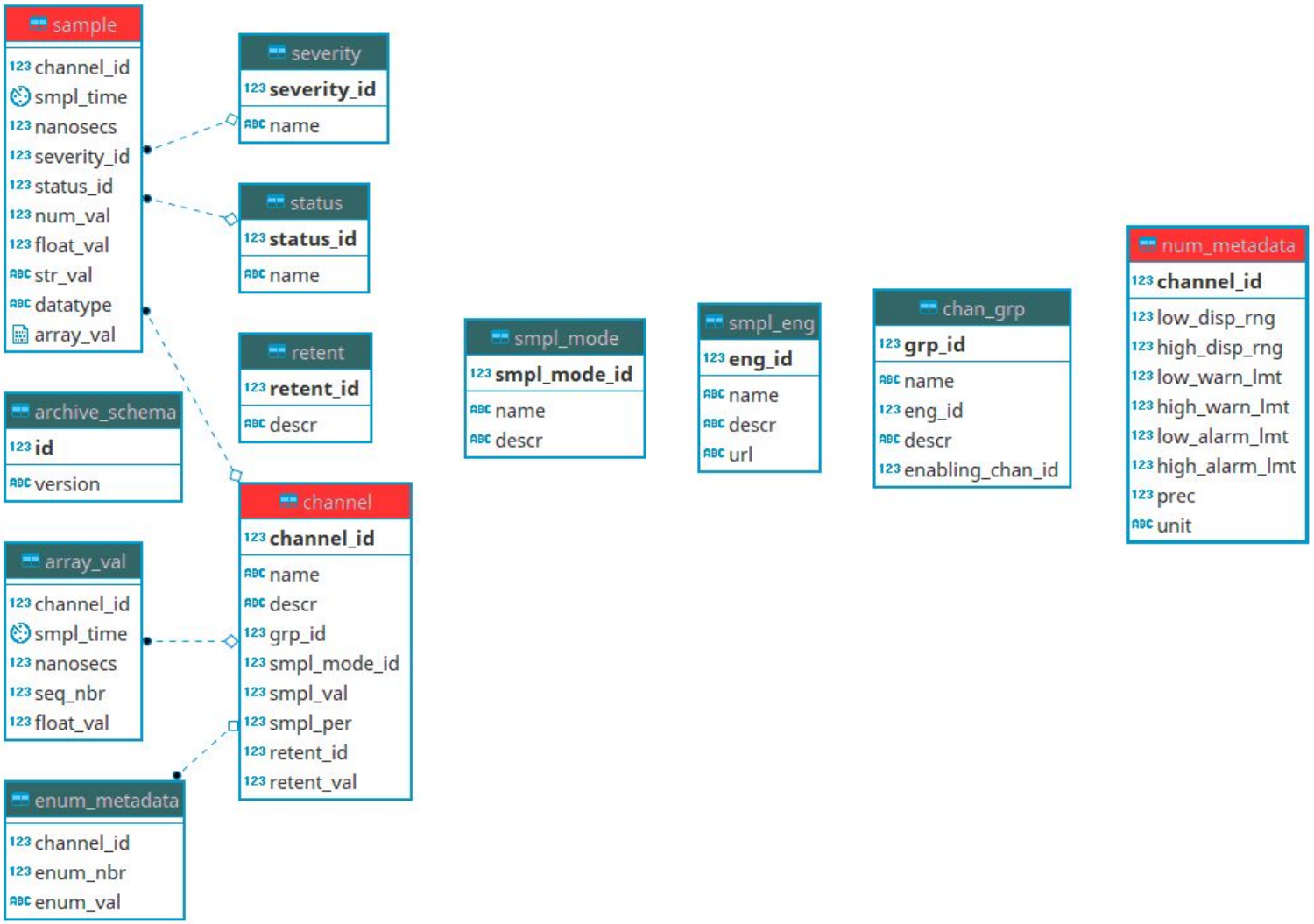
DesktopIconTable, XDAQApplicationPropertyTable, XDAQApplicationTable, XDAQContextTable

Configuration GUI in otsdaq desktop environment



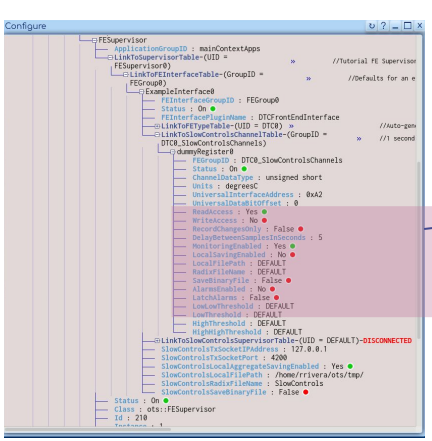
EPICS Database

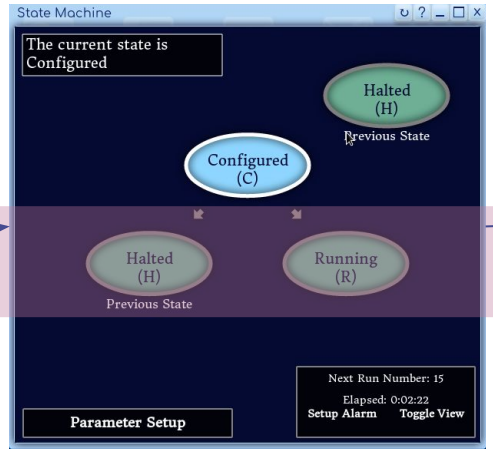
- Postgres DBMS



Integration with State Machine

- *otsdaq FE (DTC/ROC/CFO) / artdaq metric new channel or new slow control setting → configuring State Machine → EPICS DBs and IOC configuration*
- *otsdaq Interface→ otsdaq* CA subscription and DBs select → Monitoring





otsdaq EPICS interface

EPICS ARCHIVER DATABASE

OTS IOC

EPICS SERVER

EPICS
- ARCHIVER
- ALARM
- LOG

DATABASES

otsdaq EPICS interface:
Channel Access subscription for PVs Values, Settings, Status, Severity

retrieving:
PVs List and history
Last Alarms and Alams Log

Monitoring

Integration of otsdaq front-end DAQ hardware and artdaq metrics with EPICS

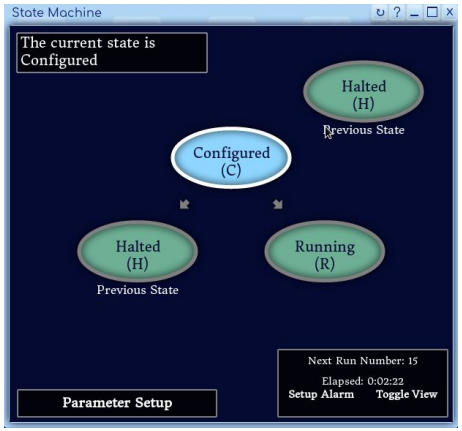
Actions designed and developed in *otsdaq*

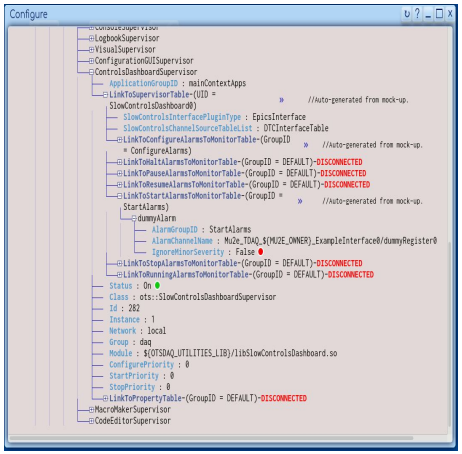
1. *otsdaq* DCS channels Front End and tables configuration
2. *otsdaq* State Machine configuration implementation
3. add/update channels info for IOC and Archiver DB
4. software IOC restarting
5. EPICS Archiver restarting
6. new *otsdaq* epics_plugin channels subscriptions to EPICS
7. Sending configured channels values to EPICS:

   otsdaq DCS channels new values → artdaq Metric Manager → software IOC → EPICS → otsdaq DCS GUI

Integration with State Machine

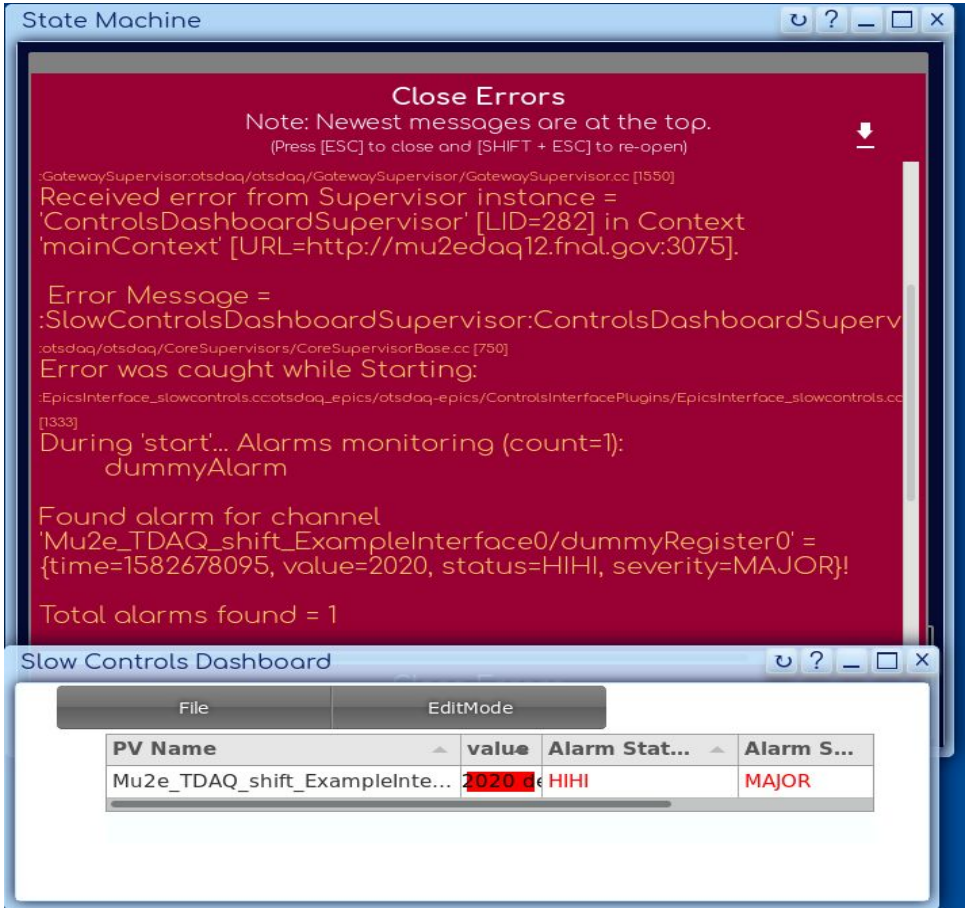
- Alarm propagation (from EPICS) and *otsdaq* state machine handling is available: needs just to identify which PV alarms, *status* and *severity* will be propagated
- *Tables and parameters designed for configuration*
  - SupervisorTable parameters:
    - *Slow Controls Interface Plugin Type*
    - *Slow Controls Channel Source Table List (HW list i.e. DTC Interface, CFO Interface)*
  - Alarms To Monitor Tables for transition to states:
    - *Configure*
    - *Halt*
    - *Pause*
    - *Resume*
    - *Start*
    - *Running*

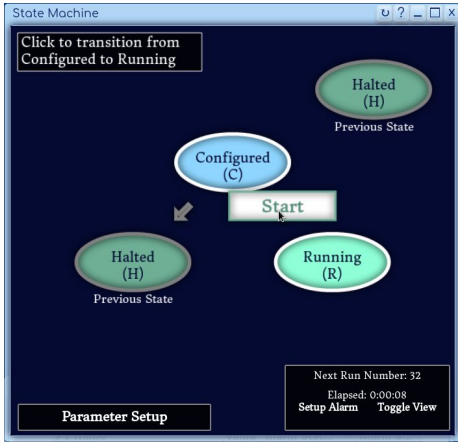


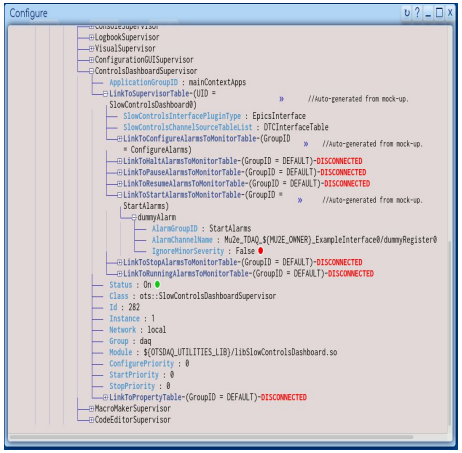


Integration with State Machine

- Alarm propagation (from EPICS) and *otsdaq* state machine handling: Example on “Start” transition







Slow Controls WEB Monitoring GUI in otsdaq

developed in JavaScript and HTML (client side) and C++ (server side)

Basic Widget Mechanism

- All widgets have six required methods:

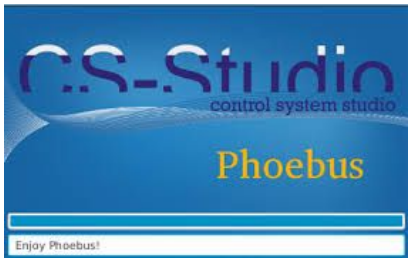
  init(), getParameters(), setParameters(), setupPVs(), newWidget(), and newValue()

Widget properties

- Dynamic sizing
- Proper handling of setups
- Value error, warning and alarm handling
- Disconnection handling

Load and save dashboard page in XML

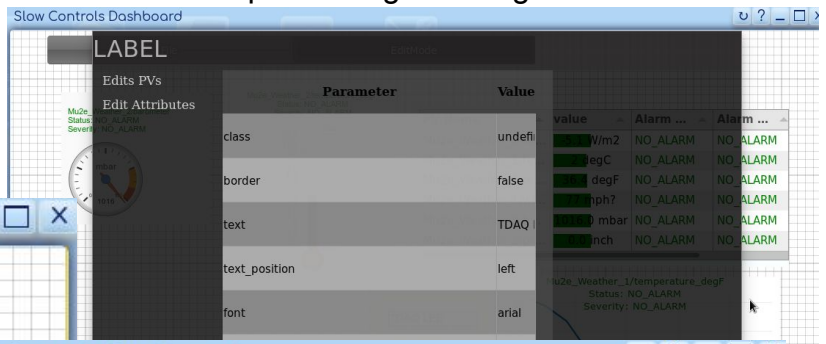
Cs-Studio Phoebus (EPICS GUI) compatible format



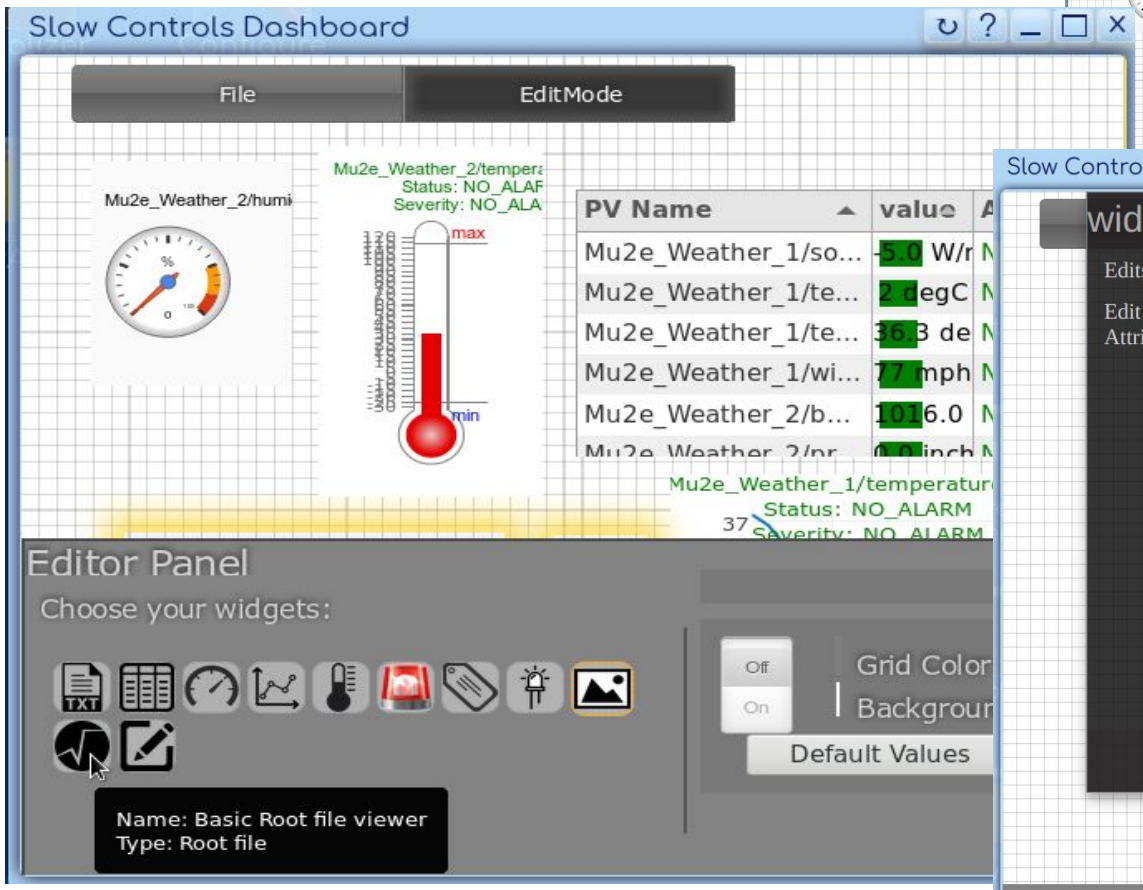
Slow Controls Monitoring in otsdaq

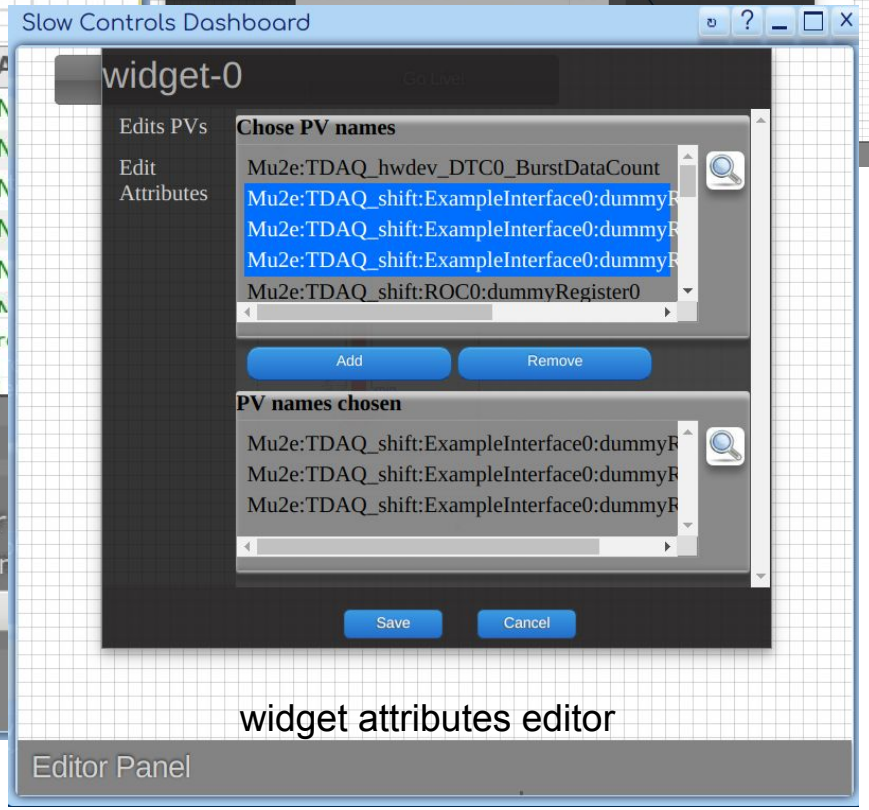
Examples

Example of widget settings window



Editor

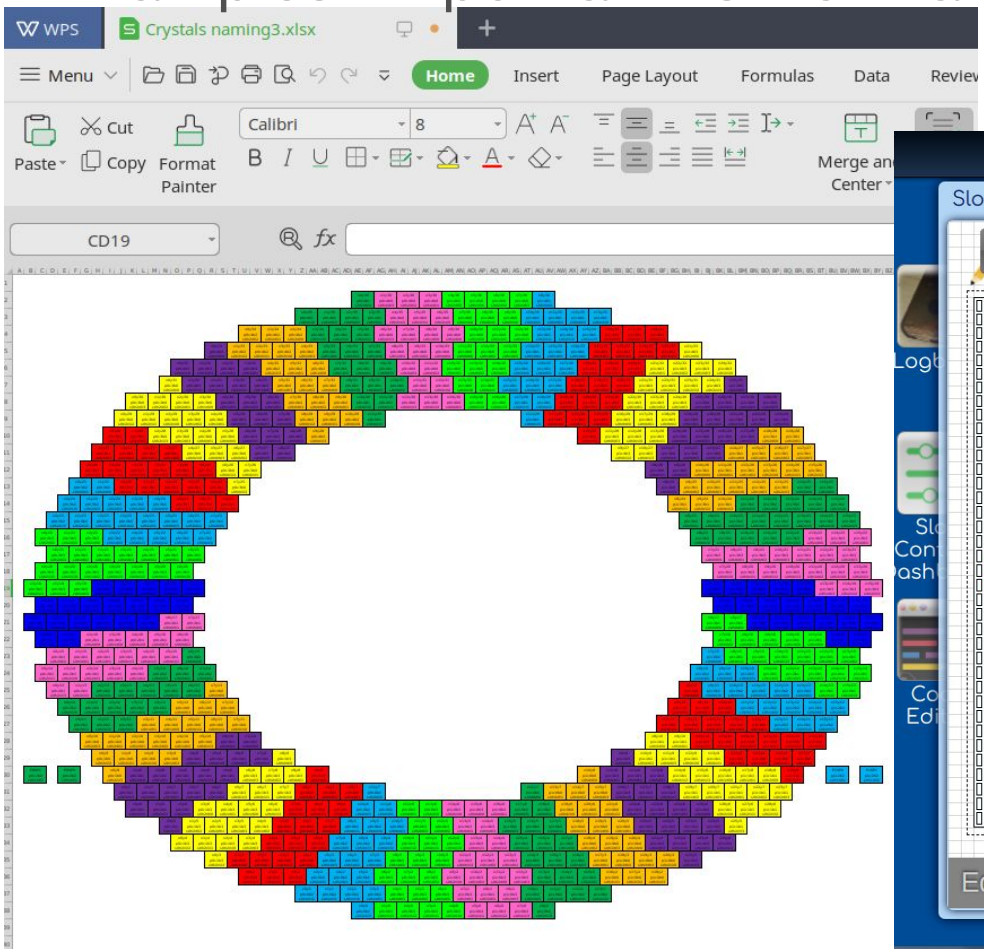


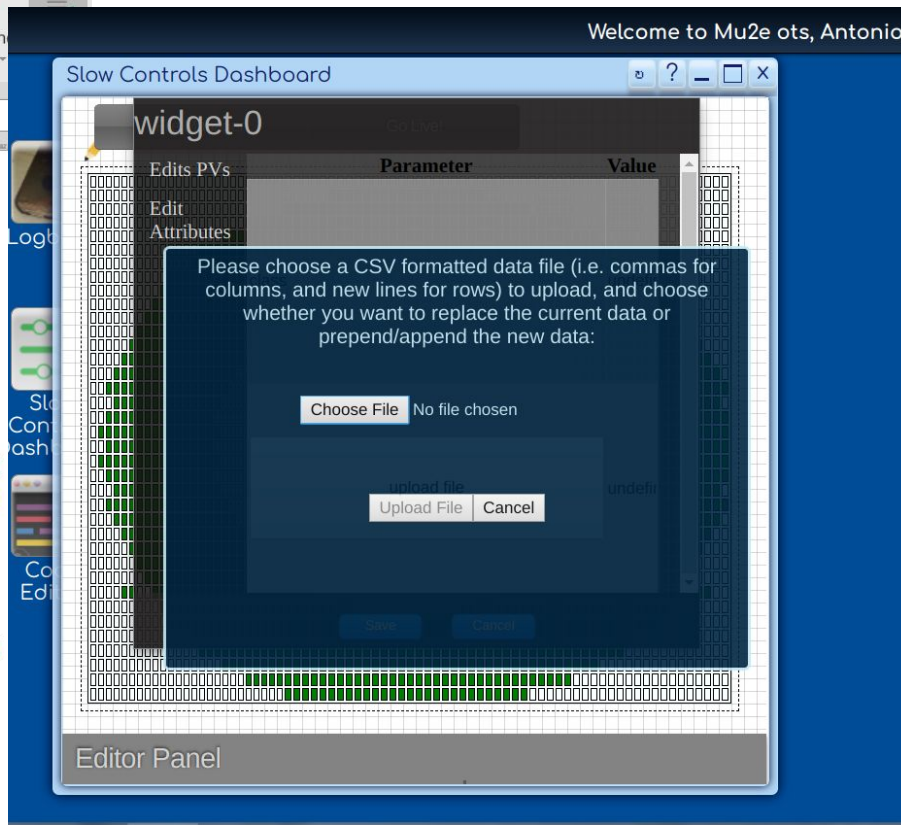


widget attributes editor

Calorimeter monitoring and the Slow Controls GUI

Examples: Import an xls file in a 2D-stop light widget





Slow Controls alarm notification by System Message

System message alarm notification example