DQM @HLT

Top trigger tutorial

2nd October 2013

Federico for the DQM team

[many thanks to Darren for the help on the trigger side]

Goal of the tutorial

- Have an overall picture of the DQM framework
- Get familiar with the core components needed to develop a DQM module
- Try to run an example:
  - what do I want to monitor?
  - which are the steps needed in order to book, fill and publish the histograms?
- Be aware that several services are provided centrally
  - generic client application
  - environment to set and run quality tests
- Make sure to respect the DQM policies while developing
- Get familiar with the main DQM sequences which are regularly run
- Be able to test a new developed DQM module

DQM in CMS

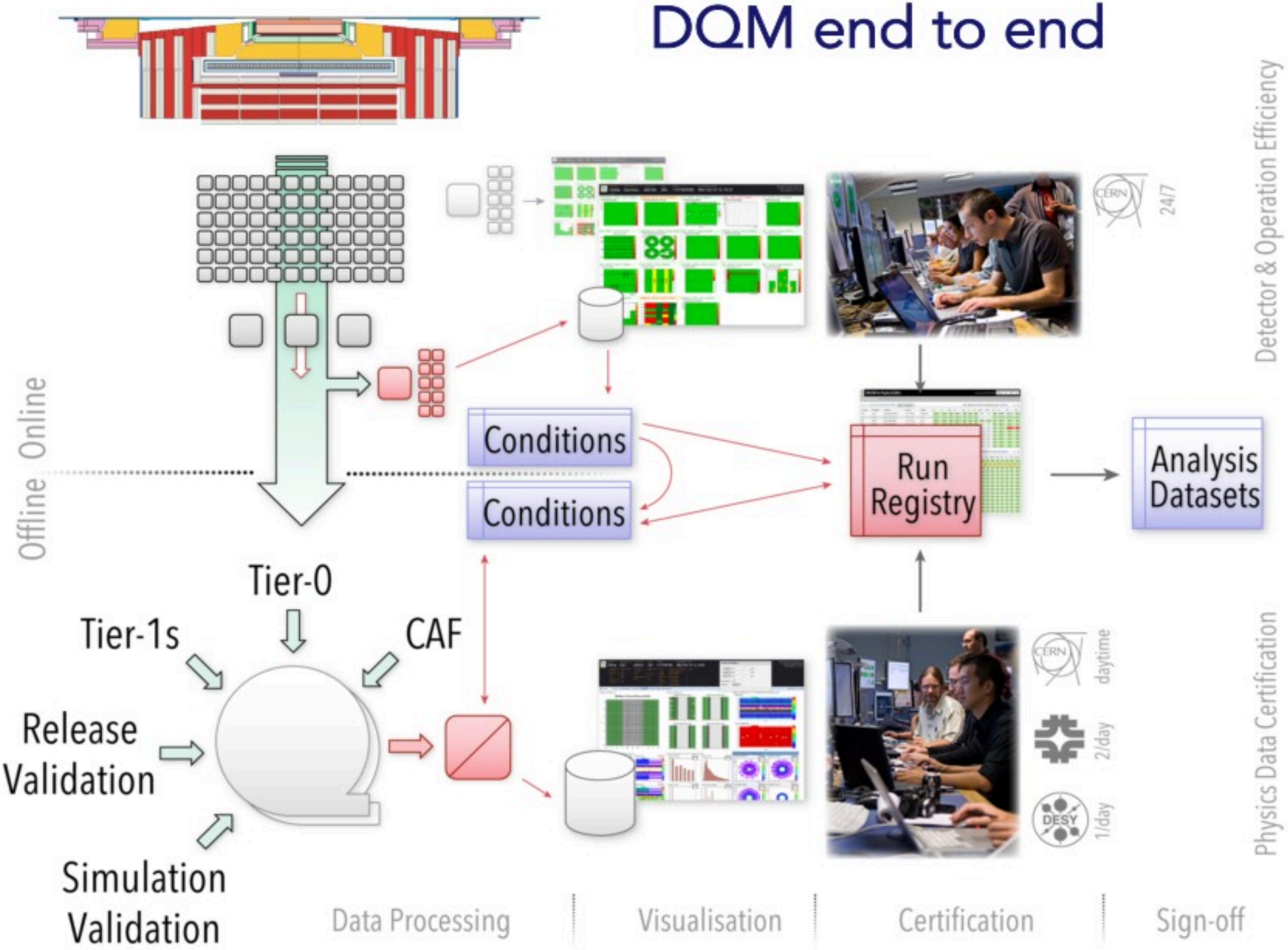
- The DQM system is designed to provide a homogeneous monitoring environment across various applications related to data taking at CMS:
  - Online, for real-time detector monitoring
  - Offline, for the final, fine-grained Data Certification
  - Release-Validation, to constantly validate the functionalities and the performance of the reconstruction software
  - in Monte Carlo productions

- The “DQM Framework” is currently Run-based and is logically divided in 2 main components:
  - Core Part, developed and maintained centrally
  - subsystem-specific modules and histogram production software

- All software is fully integrated in the standard CMS software framework (CMSSW)
  - C++, python code.

- The Core components are required to compile also outside of CMSSW for usage/ inclusion into the CMS DQM GUI, which is a standalone project.

DQM end to end



Offline Online

Tier-0

Tier-1s

CAF

Release Validation

Simulation Validation

Conditions

Conditions

Run Registry

Analysis Datasets

Detector & Operation Efficiency

CERN 24/7

Physics Data Certification

CERN daytime

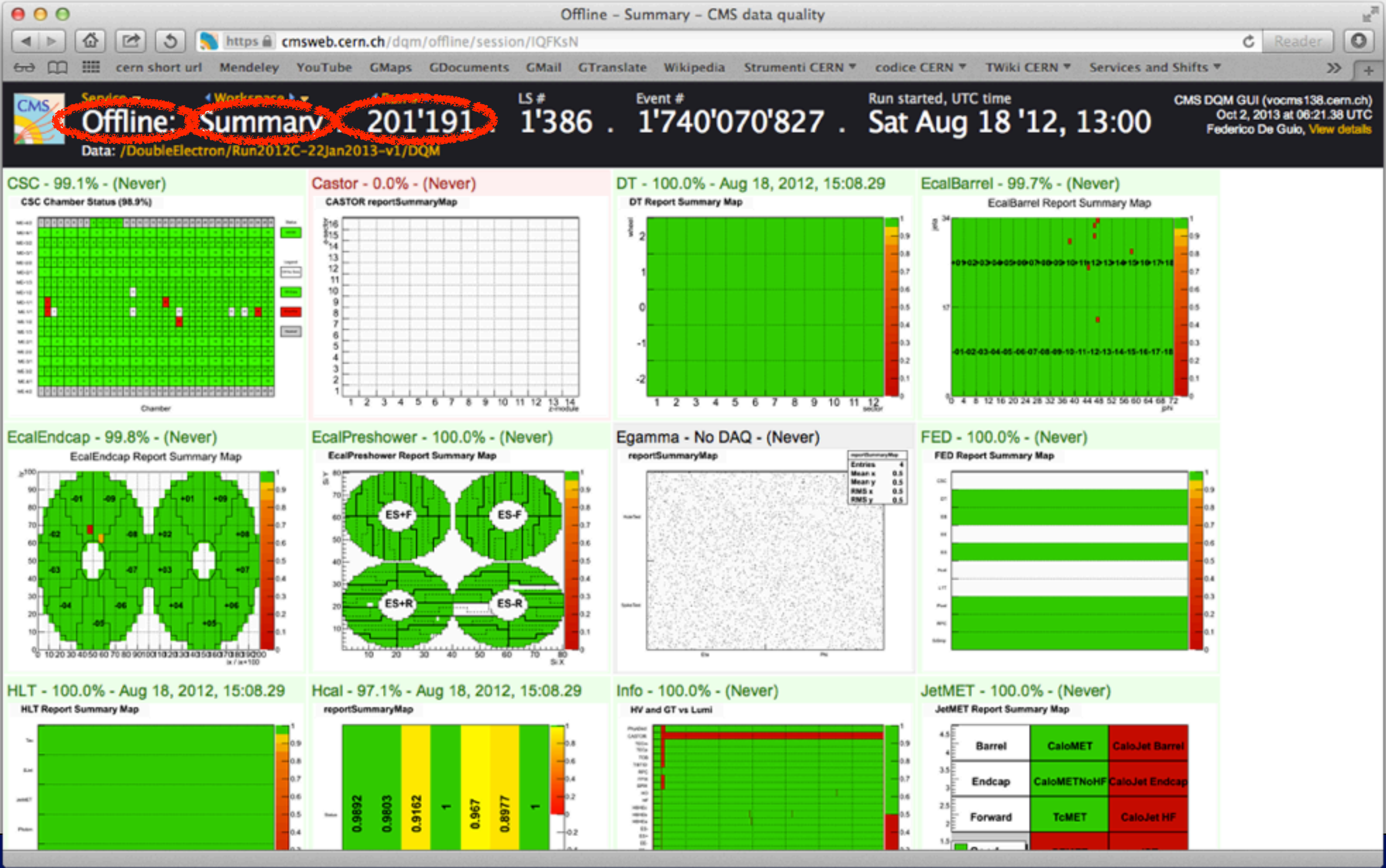
2/day

DESY 1/day

Data Processing | Visualisation | Certification | Sign-off

The DQM GUI

- `https://cmsweb.cern.ch/dqm/$flavor  the $flavor could be: online, offline, relval`
- Once a new module is included in the official sequences, the histograms appear in the GUI



Offline – Summary – CMS data quality

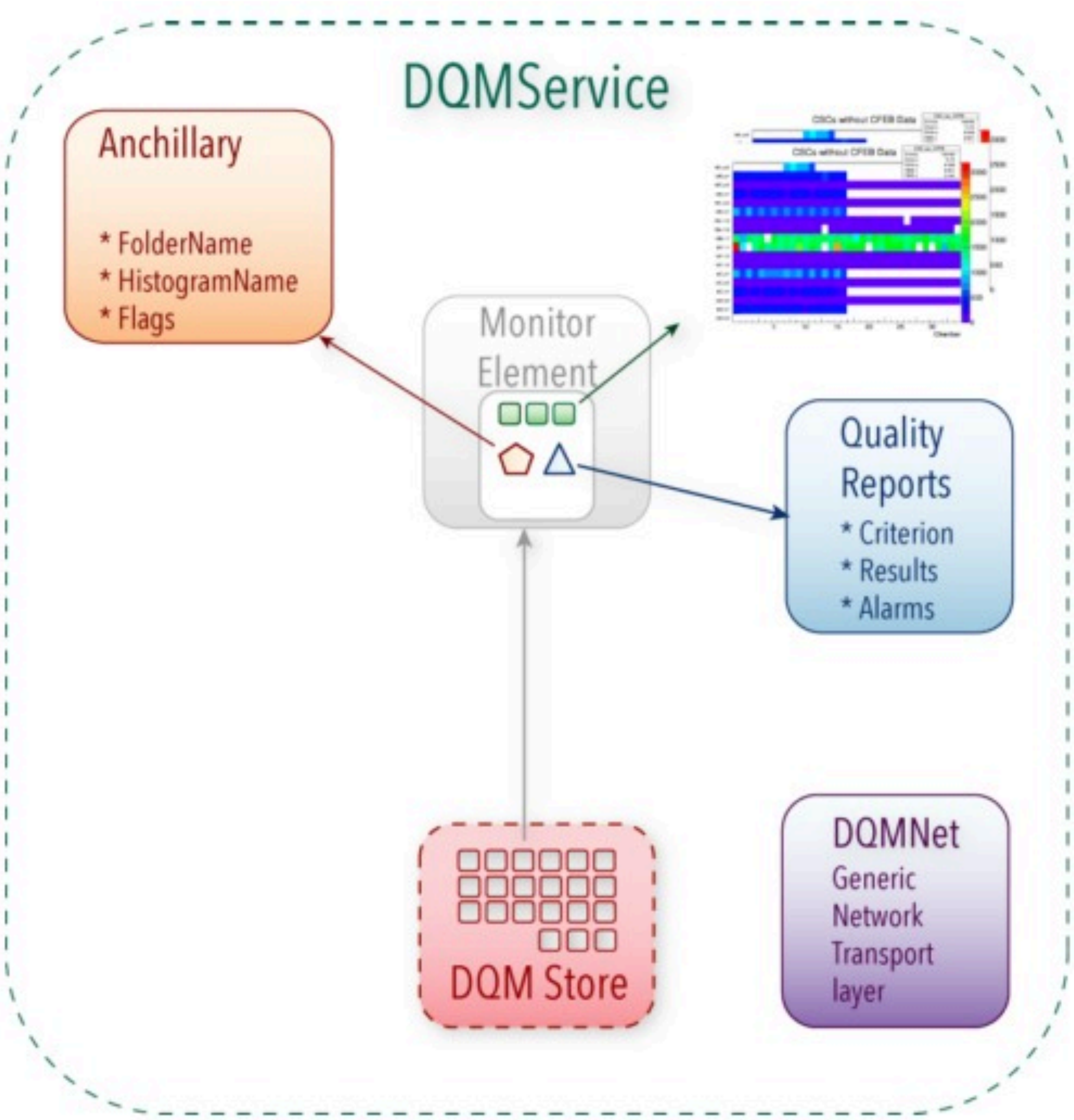
https cmsweb.cern.ch/dqm/offline/session/IQFKsN

Service: Offline: Workspace: Summary Run: 201'191 . LS # 1'386 . Event # 1'740'070'827 . Run started, UTC time Sat Aug 18 '12, 13:00

Data: /DoubleElectron/Run2012C-22Jan2013-v1/DQM

CMS DQM GUI (vocms138.cern.ch)
Oct 2, 2013 at 06:21.38 UTC
Federico De Guio, View details

DQM Core Components



DQMStore is the shared containers that holds all Monitoring Information.

The MonitorElement(ME) is the central monitoring tool

- ✓ ROOT objects
- ✓ Quality Information
- ✓ Folder hierarchy
- ✓ Flags

DQMNet is the layer to ship monitoring information over network.

DQMService ties DQMStore and DQMNet together.

https://cmssdt.cern.ch/SDT/lxr/source/DQMServices/Core/interface/DQMStore.h
https://cmssdt.cern.ch/SDT/lxr/source/DQMServices/Core/interface/MonitorElement.h
https://cmssdt.cern.ch/SDT/lxr/source/DQMServices/Components/interface/QualityTester.h
https://cmssdt.cern.ch/SDT/lxr/source/DQMServices/Core/interface/DQMNet.h
https://cmssdt.cern.ch/SDT/lxr/source/DQMServices/Core/src/DQMService.h

Saving and accessing the MEs

DQMStore

- The DQMStore is a CMSSW Service and it is available during the whole duration of the job
  - unique and gigantic piece of memory which contains all the MEs
- Creating an instance of DQMStore:

```
DQMStore* dbe_ =
  edm::Service<DQMStore>().operator->();
```

- Move between folders:

```
dbe_->setCurrentFolder("What_I_do_in_the_client/Ratio");
```

MonitorElement

- Several types of MEs reflecting different histograms types: `TH1, TH2, TProfile, ...`
- booking:

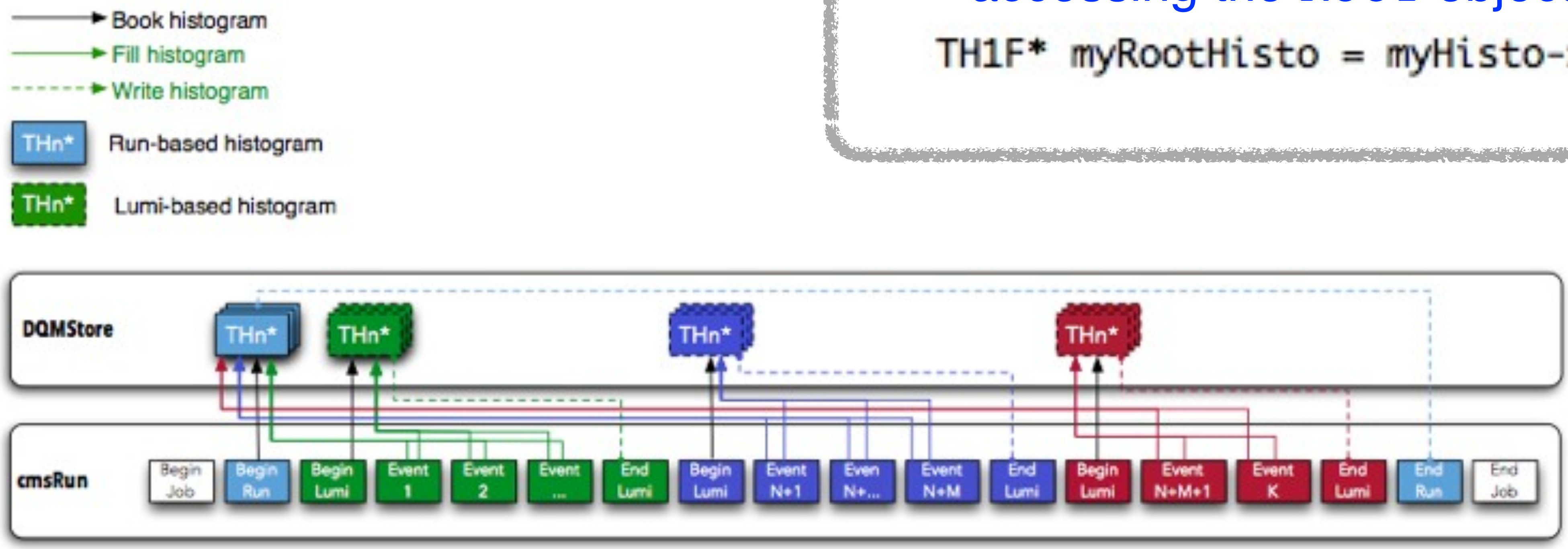
```
MonitorElement* h_myHisto =
  dbe->book1D("myHisto", "myHisto", 10,0.,10.);
```

- Get the ME from DQMStore:

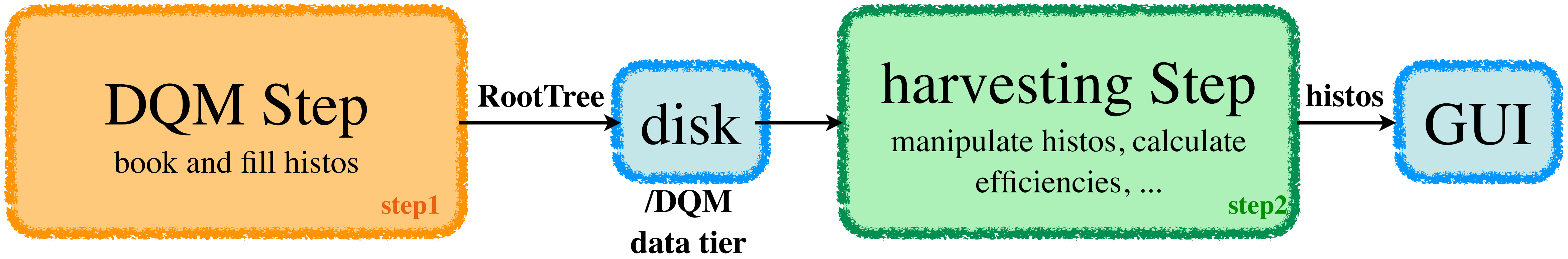
```
MonitorElement* numerator =
  dbe_->get(myHisto);
```

- accessing the `ROOT` object:

```
TH1F* myRootHisto = myHisto->GetTH1F();
```



DQM Offline WF structure



- DQM Step (Step1):
  - the booking and the filling of the histograms is performed here
  - many jobs run in parallel. the statistics available in a single job is not the full one
- Harvesting Step (Step2):
  - merge the statistics belonging to the same runs
  - perform operation where the full statistics is needed (i.e. Efficiencies)
- In both cases: **`EDMAnalyzers`** with the usual transitions:
  - `beginJob, beginRun, beginLuminosityBlock, analyze, ...`
- Specific **`DQMRootSource`** and **`DQMRootOutputModule`**
  - not `edm` format, but a simpler structure of root trees
  - allow to dump the content of the `DQMStore` in a `ROOT` file and to populate it back during the harvesting step
- **`DQMSaver`** to save the histos in the final format
  - the output contains the full stats and can be directly uploaded to the GUI

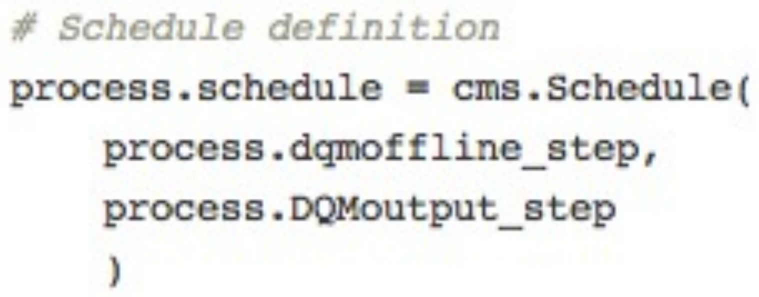
An end-to-end example [1]

Dummy analysis just to prove the principle

- Goal of the exercise:
  - choose a dataset (a Top one) and run over it. Could be either a data skim or a MC.
  - access and plot some basic quantities for the objects in the HLT event and compare them against the RECO variables

Step1

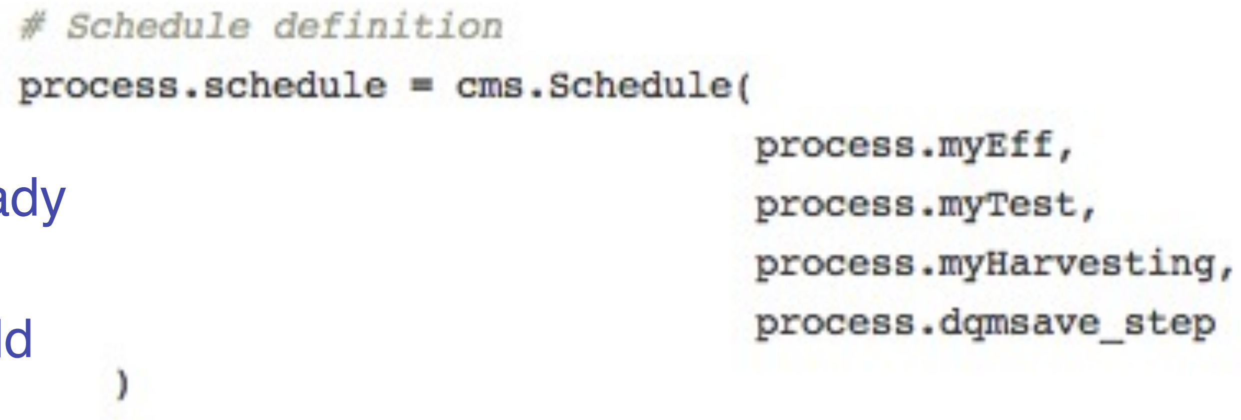
- Needed ingredients:
  - vertex information
  - electron, MET, jet collections
  - trigger event (need to specify a HLT filter and a path)
- Perform a simple analysis
  - apply some selections such as the eleID
  - perform a comparison HLT vs RECO variables
  - book and fill the histograms with the key variables



```
# Schedule definition
process.schedule = cms.Schedule(
    process.dqmoffline_step,
    process.DQMoutput_step
    )
```

Step2

- Once the histograms are filled: define the efficiencies I am interested in
  - numerator and denominator have to be filled already
- Define the automatic tests I want to perform
  - check if the efficiency is above a certain threashold
- Save the output file



```
# Schedule definition
process.schedule = cms.Schedule(
                                  process.myEff,
                                  process.myTest,
                                  process.myHarvesting,
                                  process.dqmsave_step
  )
```

An end-to-end example [2]

- The CMSSW modules reflect the two steps just described:
  - Use available relVals
    - /RelValTTbarLepton/CMSSW_7_0_0_pre4-PRE_ST62_V8-v1/FEVTHLTDEBUG
    - Both RECO and HLT collections are available in the event at the same time
  - Run the DQMStep and save the output using the `DQMRootOutputModule`
  - Run the HarvestingStep and save the output (ready to be sent to the GUI)

- The example will be queued for inclusion in release. For now checkout:
  - `https://github.com/deguio/cmssw/tree/myDQMTutorial`

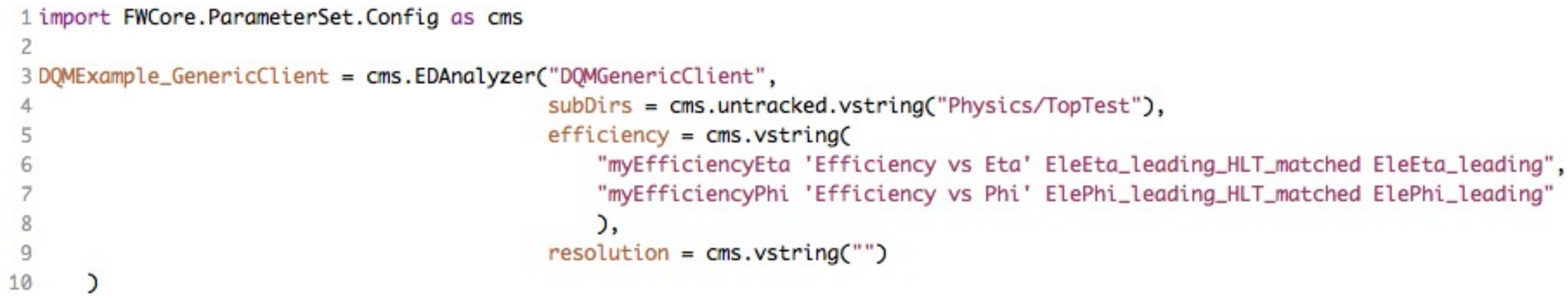
- The release used is CMSSW_7_0_0_pre4 which is currently the developer release
  - new developments are accepted only for 70X cycle

```
scramv1 project -n CMSSW700pre4_DQMtutorial CMSSW CMSSW_7_0_0_pre4;
cd CMSSW700pre4_DQMtutorial/src;
cmsenv;
git cms-merge-topic deguio:myDQMTutorial;
scram b -j8;
cd DQMServices/Examples/python/test;
cmsRun DQMExample_Step1_cfg.py;
cmsRun DQMExample_Step2_cfg.py;
```

- I have uploaded the output in a temporary GUI:
  - http://lxplus403.cern.ch:8060/dqm/dev

Operations with the `DQMGenericClient`

- A generic client has been made available to perform standard operations such as:
  - compute efficiencies, normalize to entries, make cumulative distributions, ...
- The generic client is an EDAnalyzer and can be configured using a python configFile
  - need to provide the operation you want to perform
  - need to set the input and the output



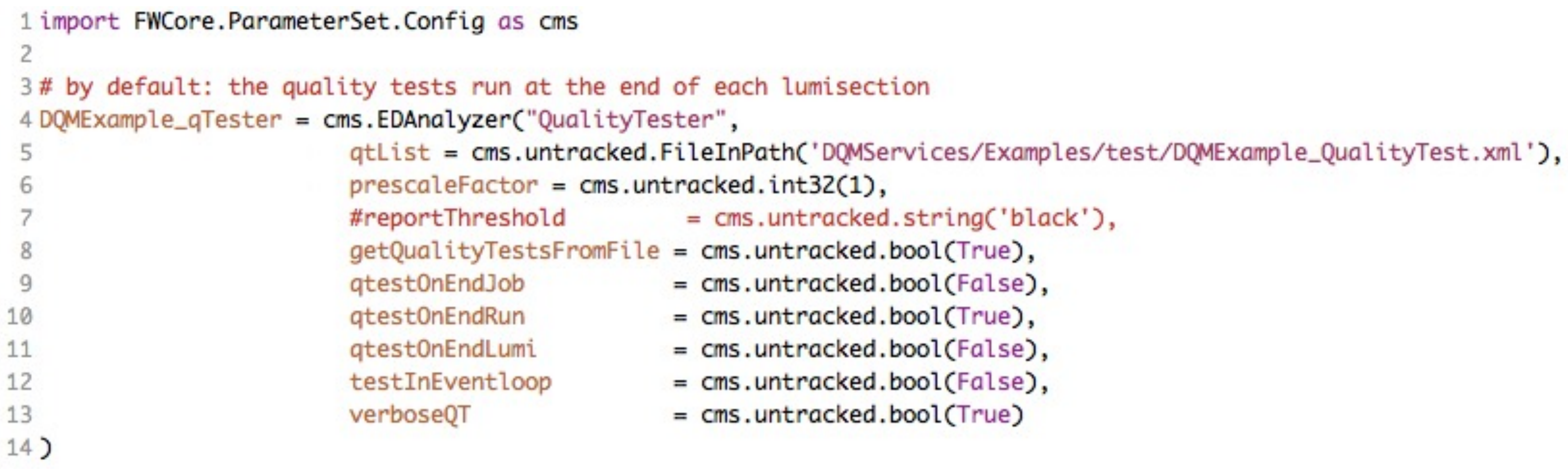
```
import FWCore.ParameterSet.Config as cms

DQMExample_GenericClient = cms.EDAnalyzer("DQMGenericClient",
                                         subDirs = cms.untracked.vstring("Physics/TopTest"),
                                         efficiency = cms.vstring(
                                             "myEfficiencyEta 'Efficiency vs Eta' EleEta_leading_HLT_matched EleEta_leading",
                                             "myEfficiencyPhi 'Efficiency vs Phi' ElePhi_leading_HLT_matched ElePhi_leading"
                                             ),
                                         resolution = cms.vstring("")
    )
```

- Among the advantages:
  - efficiency flags set properly. This is needed for the GUI
  - errors are calculated in the correct way
- See the class and all the options at:
  - https://cmssdt.cern.ch/SDT/lxr/source/DQMServices/ClientConfig/interface/DQMGenericClient.h

Quality tests

- In order to evaluate the validity of the monitoring element content in an automated way, a set of quality tests has been developed and integrated within the DQM Framework.
  - provides a fast feedback to shift crew about the data quality in terms of warnings, alarms or error reports, but useful offline also

- Tests definition:
  - https://cmssdt.cern.ch/SDT/lxr/source/DQMServices/Core/interface/QTest.h
  - more tests can be added
  - all the tests are configurable through the XML parser
  - each test needs tuning
  - https://twiki.cern.ch/twiki/bin/view/CMS/DQMQualityTests



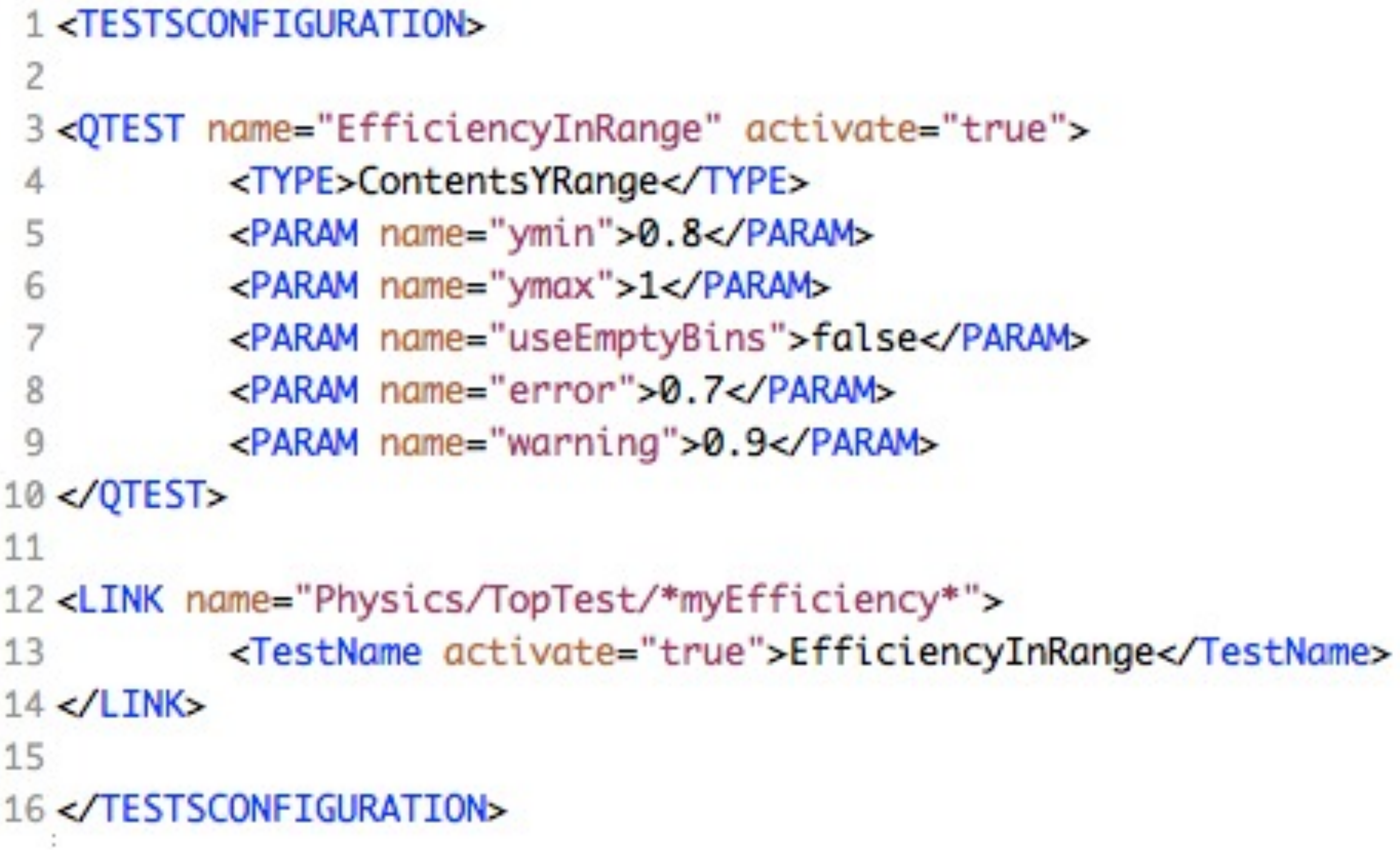
```
import FWCore.ParameterSet.Config as cms

# by default: the quality tests run at the end of each lumisection
DQMExample_qTester = cms.EDAnalyzer("QualityTester",
                       qtList = cms.untracked.FileInPath('DQMServices/Examples/test/DQMExample_QualityTest.xml'),
                       prescaleFactor = cms.untracked.int32(1),
                       #reportThreshold          = cms.untracked.string('black'),
                       getQualityTestsFromFile = cms.untracked.bool(True),
                       qtestOnEndJob           = cms.untracked.bool(False),
                       qtestOnEndRun           = cms.untracked.bool(True),
                       qtestOnEndLumi          = cms.untracked.bool(False),
                       testInEventloop         = cms.untracked.bool(False),
                       verboseQT               = cms.untracked.bool(True)
)
```

QTest configuration and alarmed histos

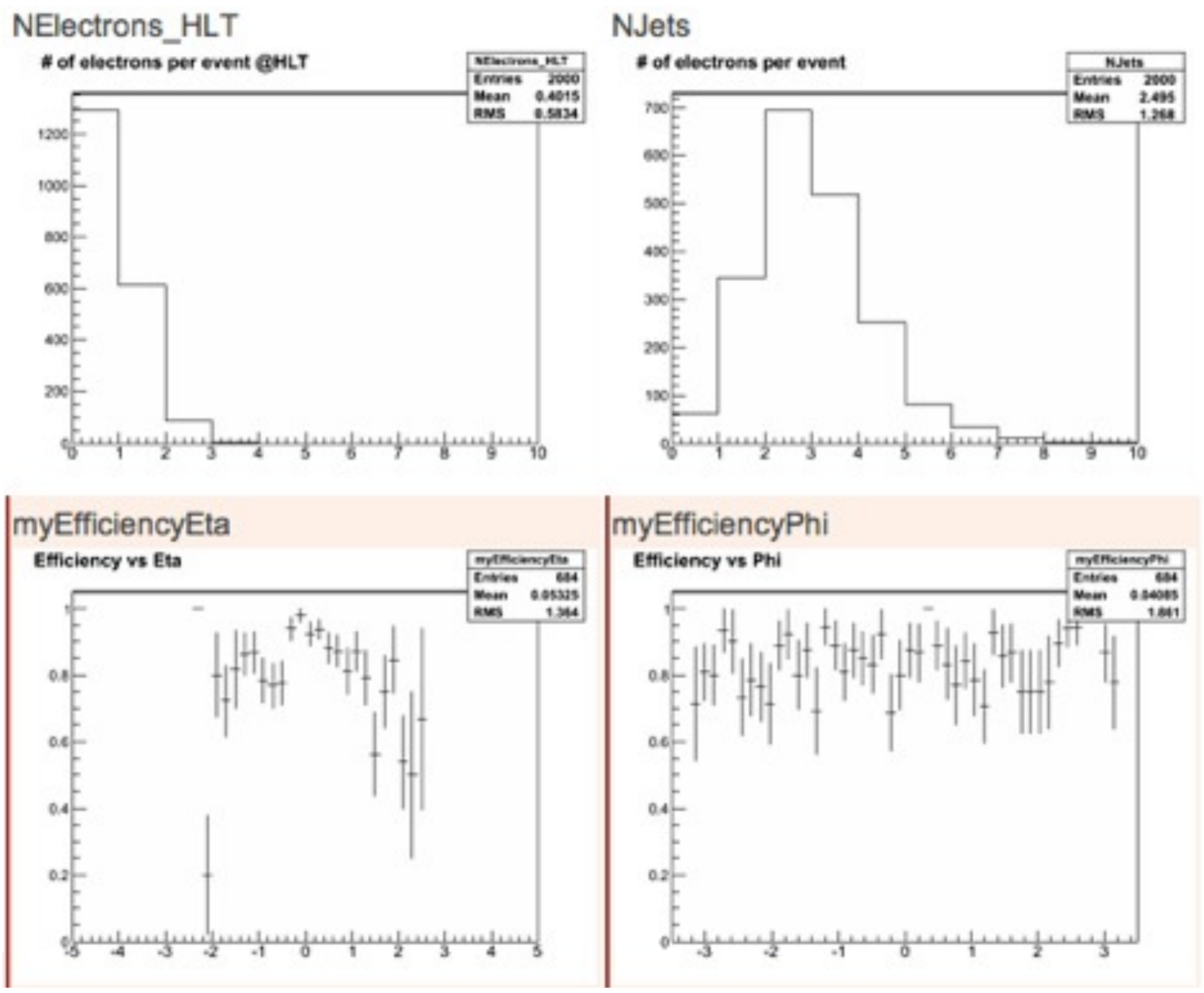
- The outcome of the test is attached to the MEs you want to monitor.
- If the test is not passed, the GUI will show the alarms in red. In the online sound alarms can be triggered

- In this example the test is: `ContentsYRange` [0.8 - 1]
  - a threshold has been set on the fraction of bins that pass the test

- The test is linked and run on the MEs: `Physics/TopTest/*myEfficiency*`



```
<TESTSCONFIGURATION>

<QTEST name="EfficiencyInRange" activate="true">
        <TYPE>ContentsYRange</TYPE>
        <PARAM name="ymin">0.8</PARAM>
        <PARAM name="ymax">1</PARAM>
        <PARAM name="useEmptyBins">false</PARAM>
        <PARAM name="error">0.7</PARAM>
        <PARAM name="warning">0.9</PARAM>
</QTEST>

<LINK name="Physics/TopTest/*myEfficiency*">
        <TestName activate="true">EfficiencyInRange</TestName>
</LINK>

</TESTSCONFIGURATION>
```



Policies and suggestions

- Make always sure to have clear in mind your use case
  - In which DQM step you have to book and fill your histograms?
  - in most of the cases the development is straightforward

Policies

- STEP1: from a study conduced months ago, we concluded that the best option is to book histograms in DQMAnalyzer::beginRun
  - the migration of the booking in the BR is ongoing for all the DQM modules
  - this is needed also in view of the transition to the threaded framework
  - all the details are available at:
    - https://indico.cern.ch/getFile.py/access?contribId=5&resId=0&materialId=slides&confId=226659
- STEP2: Currently the clients perform booking and operation in DQMClient::endRun
  - an exception is the skim case: if you want to integrate over many (more than one) runs, the operations must be performed in the DQMClient::endJob

Suggestions

- Always protect your code: make sure that the collections you need are present in the event and valid
- Always use the tools provided centrally
  - maintained centrally
  - can be extended if needed
  - same code available for many

Multirun harvesting [a specific use case]

- The goal of the **Multirun harvesting** is to merge several runs in the second step and have **high statistics distributions** available in the GUI for a given run-range
  - think about the Z peak for instance

- This is the typical use case of the Skims where different runs could be mixed together in a single file

- With the `DQMStore::CollateHistograms` option enabled in the harvesting step, the statistics of the processed runs is summed.
  - To be combined with the following DQMFileSaver options:

```
process.dqmSaver.saveByRun = cms.untracked.int32(-1)
process.dqmSaver.saveAtJobEnd = cms.untracked.bool(True)
process.dqmSaver.forceRunNumber = cms.untracked.int32(999999)
```

- In this case the LS based plots are **not supported by default. The same is true also for all the quantities derived from them in the clients.**

- Having the flexibility of the **`DQMFileSaver`** in the DQM clients could help in handling the multirun harvesting case.
  - the actions performed in the clients could be moved in the endRun/endJob depending on the needs

DQM Sequences and code

- The DQM code in CMSSW is organized as follows:
  - /DQMServices: Core classes
  - /DQM and /DQMOffline: DPG and POG oriented modules
  - /DQM/Physics: PAG oriented modules

MC dependencies are forbidden

  - /Validation: validation packages (relVal+MC)
  - /HLTriggerOffline: validation packages for HLT

can run over the MC

- Sequences:
  - the Validation sequences are defined in Validation/Configuration and called centrally as defined in Configuration/StandardSequences
  - for the offline DQM sequences everything is in DQMOffline/Configuration. A DQM matrix allows to run partial sequences.
  - The online sequences are instead defined (with the needed/customized reco steps) outside CMSSW. See https://github.com/cms-sw/DQM-Integration/tree/master/python/test

- In some cases the organization of the code follows historical reasons
- In general we can discuss together where a new package should be

How to test the main WFs [a.k.a. whiteRabbit.py]

- We maintain a system of scripts to test the main workflows:
  - The DQM sequences can be run over MC, FastSim, Data in different scenarios (PP, HI)
  - The monitor of the memory consumption is performed and allows to estimate the sustainability of the changes implemented

- Documentation and code:
  - general instructions: https://twiki.cern.ch/twiki/bin/viewauth/CMS/DQMOffline
  - list of the tests available: https://twiki.cern.ch/twiki/bin/view/CMS/DQMOfflineTests
  - code: `https://cmssdt.cern.ch/SDT/lxr/source/DQMServices/Components/test/`

- Recommendation:
  - always test your code before submitting it for integration
  - avoid using too many bins if it is not really needed
  - be careful in particular with the 2D histograms

- Simple syntax:
  - `python whiteRabbit.py -j4 -n1,2,11,12`

- As for the other central tools, it is possible to add tests if your use case is not covered
  - we are happy to help and implement them

Not for today

- No time to discuss the GUI today, but not crucial. The output from step2 can be inspected by hand. You can already start developing...

- For the next time:
  - how to setup a private/temporary GUI for testing
  - how to upload histograms in the GUI
  - how to develop a render plugin
  - how to develop a layout

- If you are curious the following instructions are straightforward:
  - https://twiki.cern.ch/twiki/bin/viewauth/CMS/DQMTest#Installing_the_GUI_Server

Summary

- Now you are able to develop your own DQM package. The physics content should come from you.. ;)

- The available documentation about DQM is linked (maybe not directly) from:
  - https://twiki.cern.ch/twiki/bin/viewauth/CMS/DQM
  - We will try to keep examples and instructions up to date as much as possible

- If your use case is not supported we are available to discuss a solution together
  - the instruments we provide centrally can be improved with your help

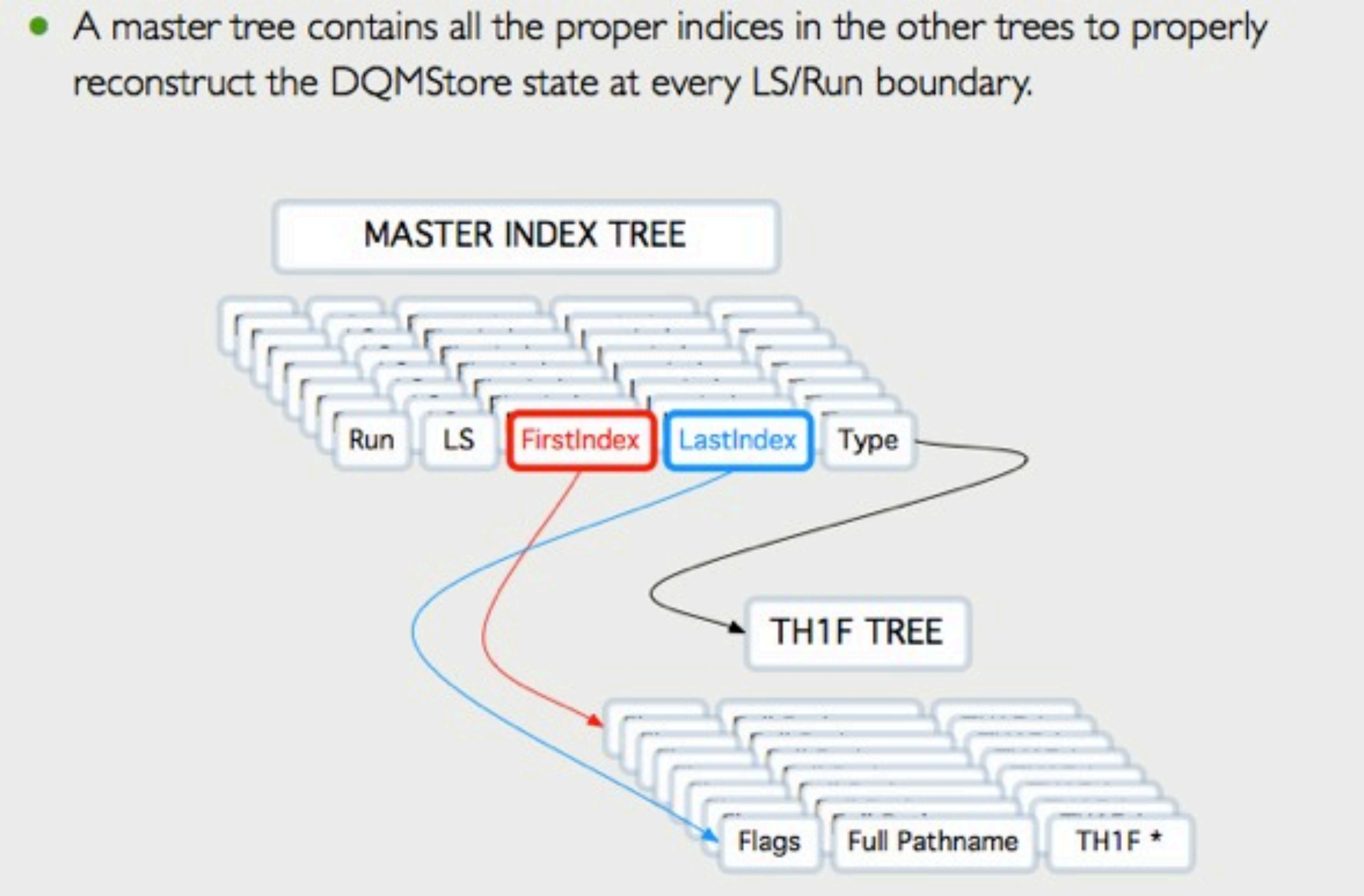
- CMSSW is now on gitHub. Need to pass through it if you want to submit changes/new developments in release
  - (non-)CMSSW oriented tutorials are available:
    - http://git-scm.com/book
    - http://cms-sw.github.io/cmssw/index.html
  - we are available to help if needed

- **We would be happy to have you in our team of developers.**

Backup

The DQMIO format organization

- With the DQMIO the MEtoEDM and EDMtoME steps are not needed anymore.
  - **DQMRootOutputModule** to save the histos in the new format
  - **DQMRootSource** to access the histos in the /DQM data tier
- The structure of the DQMIO is as follows:

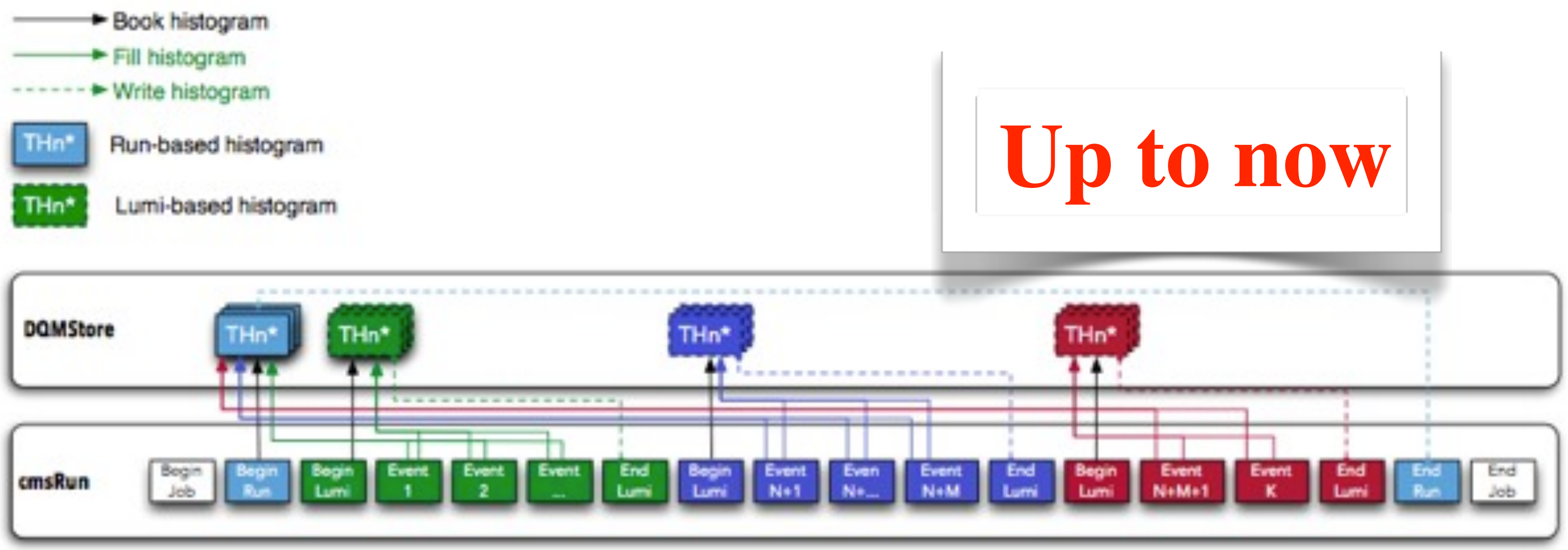


- A master tree contains all the proper indices in the other trees to properly reconstruct the DQMStore state at every LS/Run boundary.

- **EDM/ME (EDProducer+EDAnalyzer) --> DQMIO (InputSource+OutputModule)**
  - **The transitions will be different.** Special attention needed here to ensure the correctness of the results

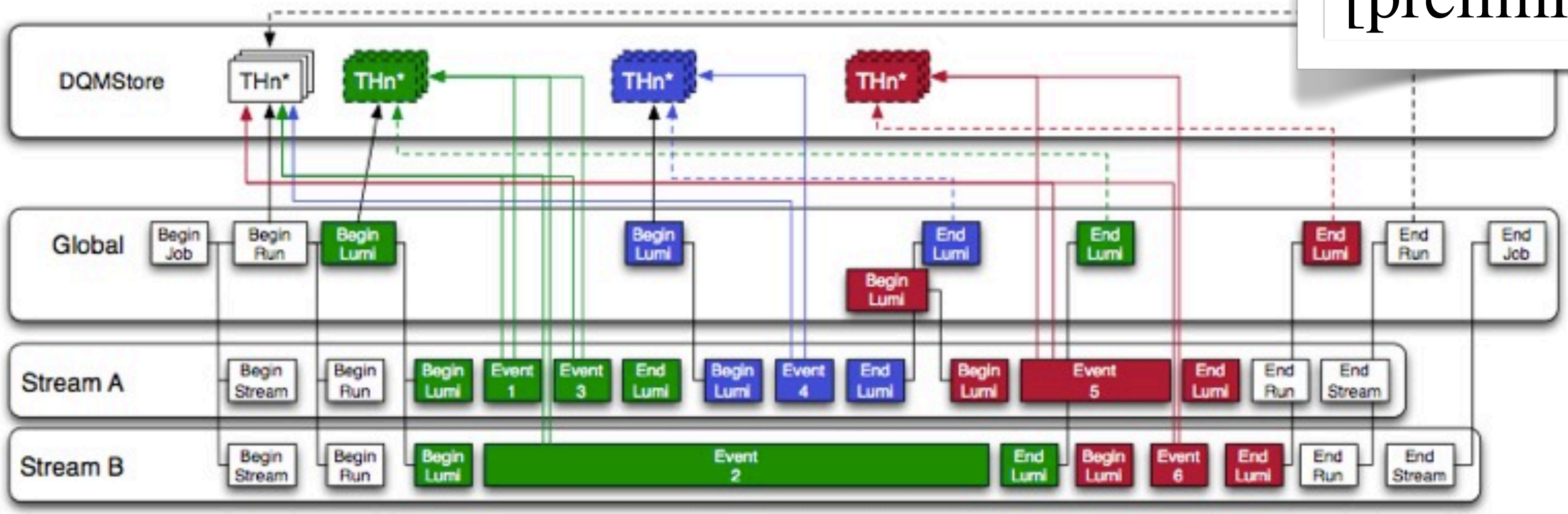
Booking in a multithread environment

- When a multithread version of the CMSSW framework will be deployed, the events will be processed in parallel
- The event will not be seen as a global entity anymore
- Different runs, events, LS will be processed in different streams



Up to now

- **Development ongoing. At this stage the booking in the BR appears to be the best choice.**



Multithread

[preliminary]