

# Design of the Data Quality Monitoring system for the BM@N experiment

Igor Alexandrov, Evgeny Alexandrov, Alexander Chebotov, Konstantin Gertsenberger  
JINR, MLIT, LHEP



7-11 July 2025, Dubna, JINR  
11th International Conference GRID 2025



Joint Institute for Nuclear Research

# Outlines

- **Goals of the system creation**
- **Short review of DQM/DQA systems in LHC experiments**
- **General architecture**
- **Object model of the histogram producer and analyzer interfaces**
- **DQM processes and data flow**
- **DQM configuration description**
- **Example of the possible DQM configuration**
- **Database schema**
- **Alarms behavior**
- **Possible view for DQM shifter WEB GUI**

# Data Quality Monitoring (DQM) and Quality Assurance (QA) : Goals of the systems

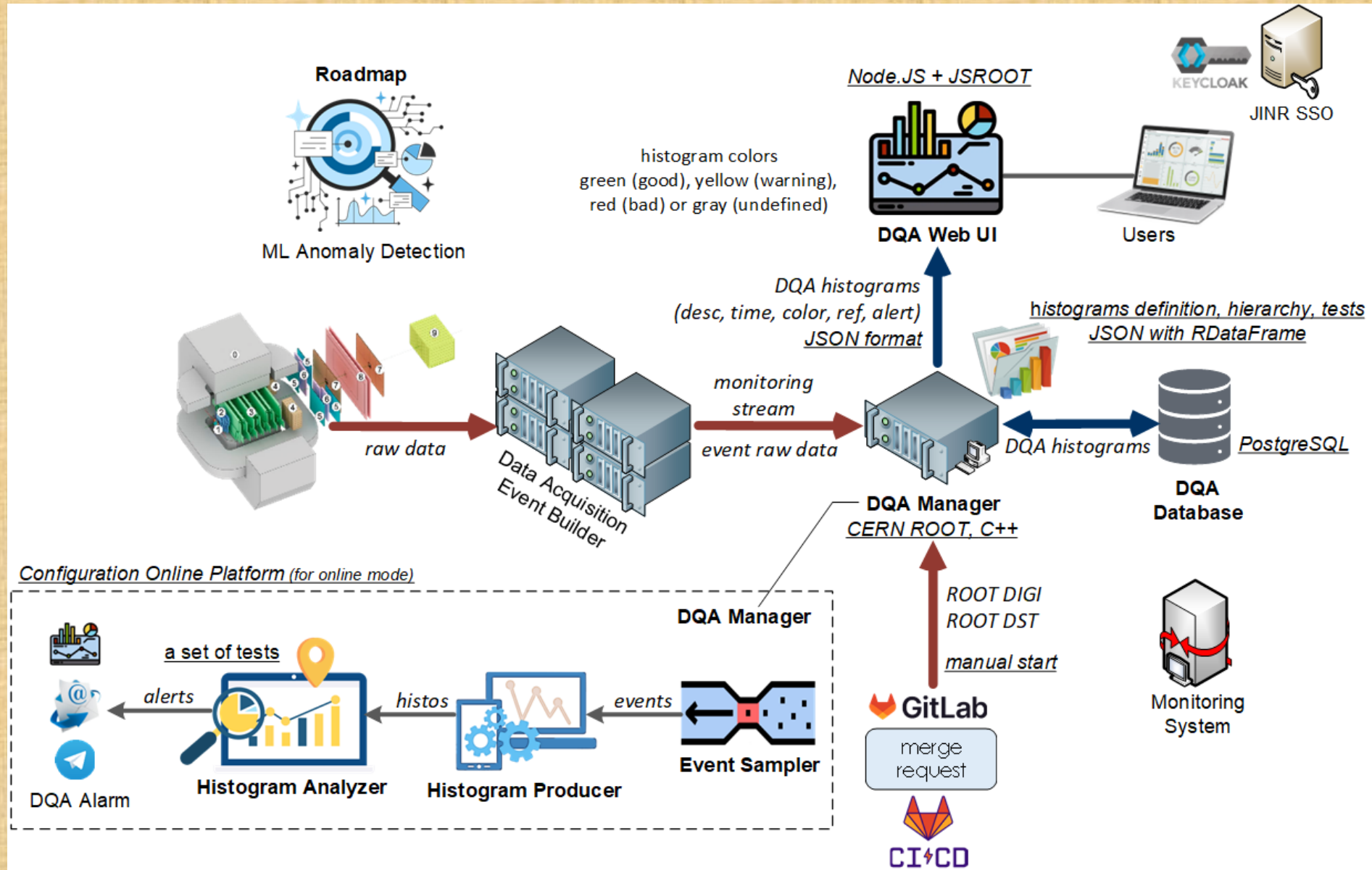
- Data Quality Monitoring and Quality Assurance system for the BM@N experiment should be developed to provide control histograms in the same way in 3 modes
  - for online decoding and reconstruction (online data monitoring)
  - for qualitative assessment of new BmnRoot versions (after MRs)
  - for manual run to check user versions of the software
- The BM@N DQA architecture must ensure predefined checks and graphical representation of control histograms on a central Web service, which receives the displayed data distributions from the BM@N histogram producer (DQA central manager)
- The DQA system must provide the ability to easily add new control histograms with predefined checks and alerts in a user-friendly format (suitable for physicists and detector team, who are not developers), for instance, using JSON-description



# DQM & QA systems in LHC experiments

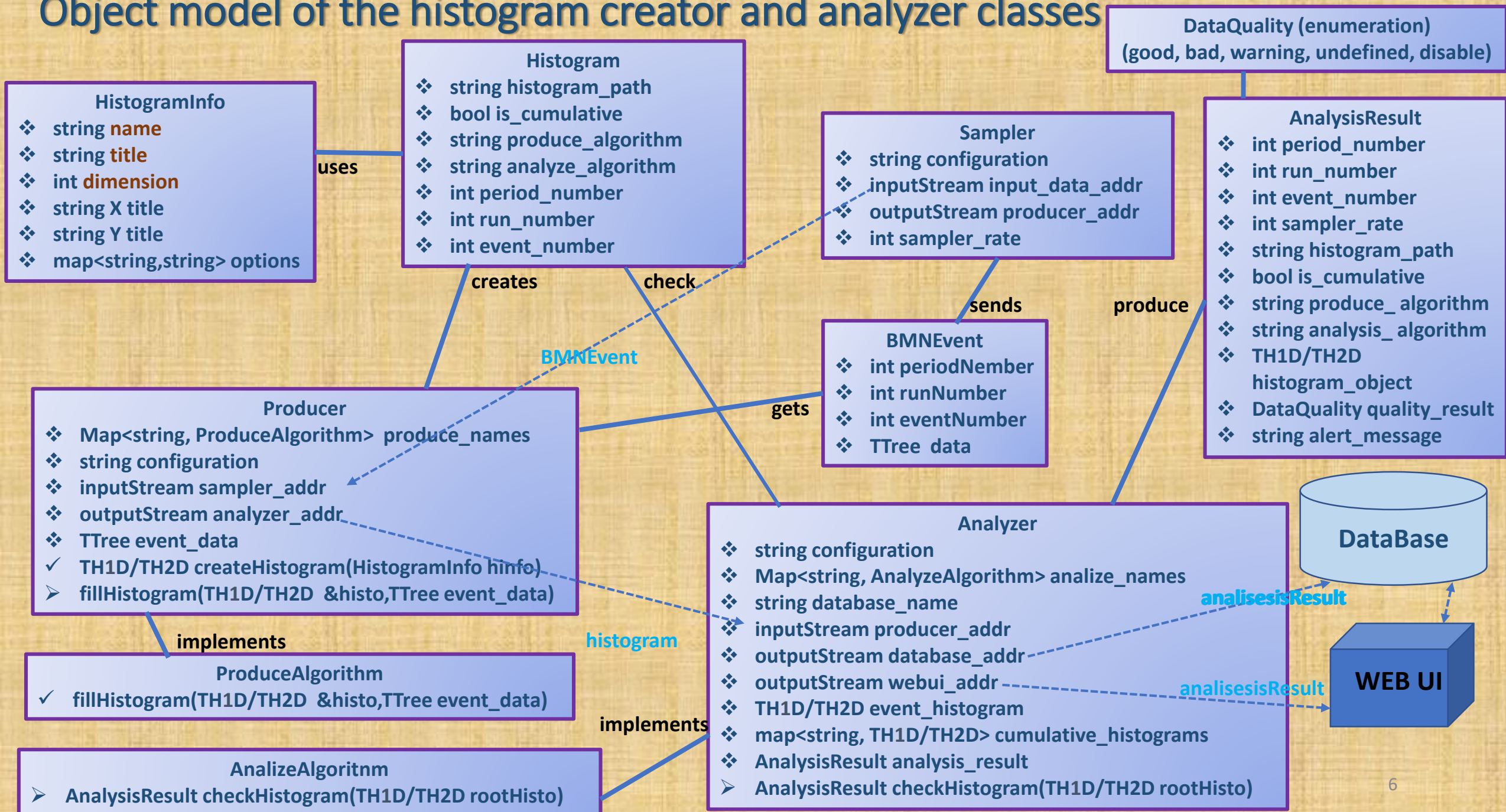
- Some interesting ideas in the LHC experiments:
  - produce some sampling (events go to the DQM depending on their frequency after trigger system)
    - **message passing** technique, scalability (ALICE)
    - set the **rate** for DQM input
    - produce histograms as main input for the DQM
      - but **not only histograms** can be used for quality check
  - moving as much as possible to **automation** of data quality assurance
  - **flexibility** in using check algorithms (storing sources of algorithms, their names and parameters, what to be on output, destination and other in DQM database)
    - loading from its library the check algorithm as implementation of interface (ATLAS)
    - some interfaces have thresholds as parameters in order to create results: bad or good event
  - some of check algorithms
    - empty histogram (a threshold)
    - counters of subdetector responses by sectors
    - Kolmogorov-Smirnov test
  - moving to **rich shifter GUI**, mostly **web GUI**
    - **different base tools/languages** for implementation (python as an example in ALICE, C++ in ATLAS)
    - **alarms** in case of high probability that events have bad quality
  - **ALL** these experiments suppose to use ML-based (**machine learning**) quality assessment in Run 4

# The DQM General architecture

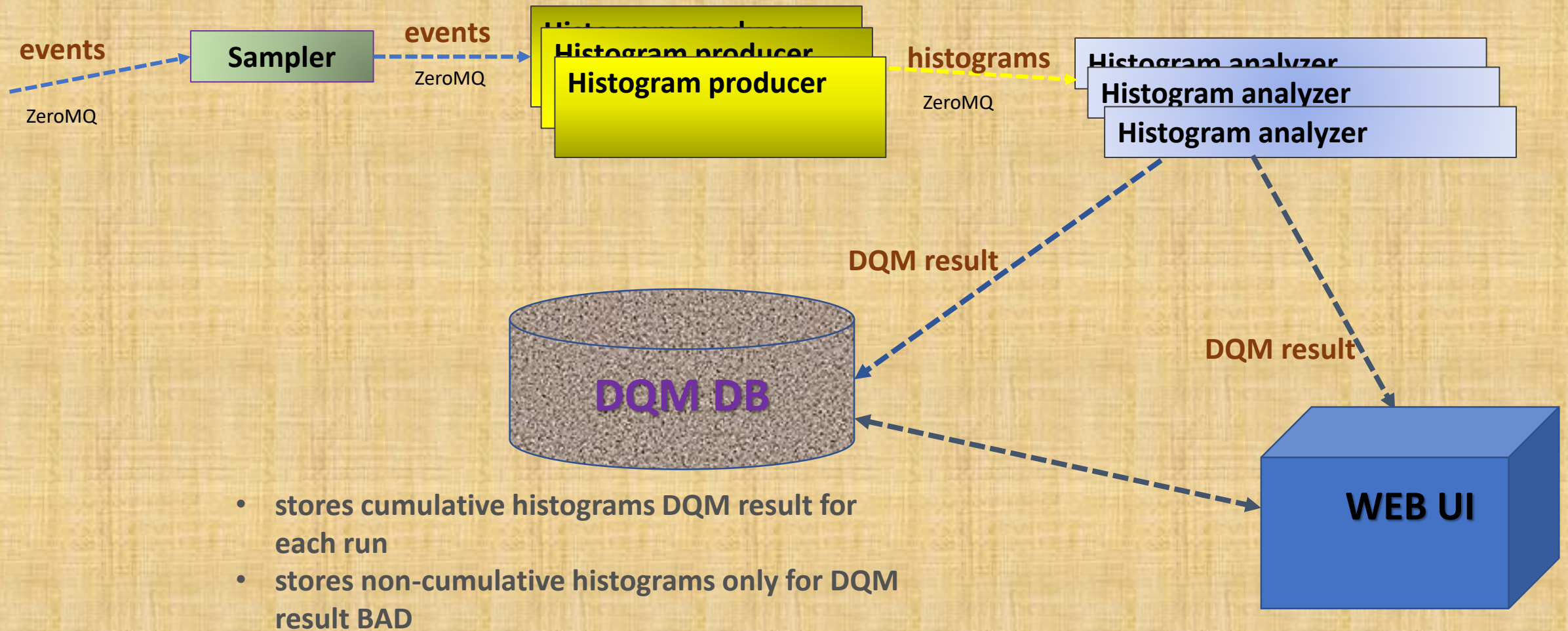




## Object model of the histogram creator and analyzer classes



# The DQM processes and data flow



# DQM configuration description

- The configuration for running DQM is stored in JSON format in a file that contains all the information concerning
  - sampler (event selector for data quality assurance) and histogram producer
  - addresses of the database and Web interface
  - analyzers to be applied to the histograms
- The description of the histograms is stored in JSON format and is splitted into separate files so that:
  - each detector group can describe histograms only related to their tasks
  - histograms not related to the group will not be contained in their configuration file
  - all JSON-files with the description of the histograms are loaded when the DQM system starts
- The content of the JSON files corresponds to the given object model

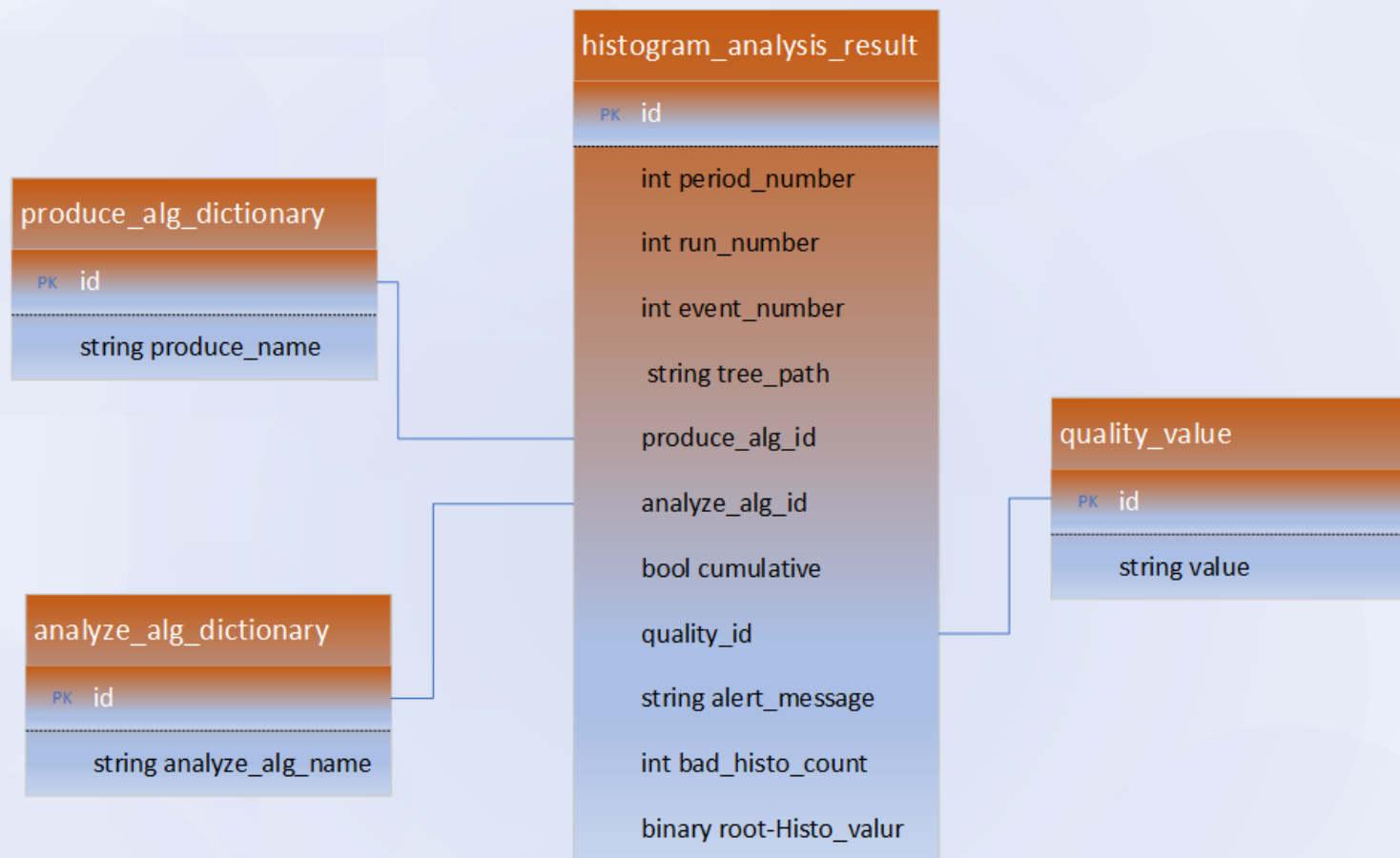


# Example of the possible DQM configuration

```
{ "DQM_SETUP": "setup1",
  "histogram_library": "lib/histograms_produce_alg",
  "analyzer_library": "lib/dqm_analiz_alg",
  "tasks": [
    {
      "task_gem" : {
        "producer_input" : "host1:20001",
        "analyzer_input" : "host2:20002",
        "database " : "host3:20000",
        "web" : "host4:8000",
        "histogram": {
          "tree_path" : "GEM/Silicon",
          "alg_name" : "statistics",
          "cumulative": "true",
          "info": {
            "name": "statistics_gems",
            "title": "Statistics GEM Silicon",
            "x title": "Number of digits in event silicon",
            "y title": "Number of digits in event GEM",
            "dimension" : "2" , "comment1": "TH2D in use",
            "Options" : { "comment2" : "draw options,
              color options, text options, SURFace options etc",
                "c_opt" : "c",
                "arr_opt" : "arr",
                "line_color" : "red",
                "fill_color" : "grey"
            }
          }
        }
      },
      "analyze" {
        [ {
          "analyze1" : {
            "alg_name": "kolmogorov_alg",
            "cumulative" : "true",
            "parameters" : {
              "threshold1" : "10",
              "threshold2" : "15"
            }
            "draw_out": "hostm:8000"
          }
        }, { "analyze2" : {
          "alg_name": "empty_check2",
          "cumulative" : "false",
          "parameters" : {
            "threshold1" : "1",
            "threshold2" : "4"
          }
        }
      ]
    }, //comment3 : "END task_gem",
  ], {
    "task_gem_station" : { "histogram": {
      "producer_input": "host1:20004",
      "etc" : "....."
    }
  }
}

, "sampler" : {
  "input": "host1:20000",
  "out" : ["host1:20001", "host2:20004"],
  "rate" : [ "10", "5" ]
}
```

# DQM database schema



# Alarms description

- Alarm is displayed in Web UI
- Alarm is raised when DQM result value is BAD
- **GREEN** color is used to display the data quality value when the DQM value is GOOD, **RED** color – if the value is BAD
- DQM tree colors in Web UI
  - If all children of some DQM root are GOOD, then the name of this root is displayed in **GREEN**
  - If any child of DQM tree directory is BAD
    - the name of this child element is displayed in **RED**
    - all parents' names are displayed in **ORANGE**



# Possible view for DQM shifter WEB GUI

Time: 15:04:07

Run State: RUNNING

State: **GOOD**

Run Number: 2900

Event Number: 150036

## ☐ Root TREE

### ■ GEM Station

#### ■ Module 0

■ Layer 0

■ Layer 1

■ Module 1

■ Module 2

■ .....

### ■ GEM Silicon

HISTOGRAMS

ALAM LOG

# Conclusion

- The goals of the system are described
- Review of LHC experiments DQM and DQA systems is presented with estimation of possibility to use in BM@N system some ideas and solutions
- DQM system general architecture and system framework is shown
- Object model of the histogram creator and analyzer interfaces are presented.
- DQM configuration description and database schema are described
- The alarms behavior are shown
- The preliminary view of the Web GUI for shift operators is presented

**Thank you for your attention!**

# BACKUP



# DQM & QA systems in LHC experiments: **ATLAS**

- **ATLAS: light, flexible** (input-output-configuration interfaces, algorithms as plugins)
  - **DQM Core: executes DQ Algorithms** (any common operation like histogram comparison, histogram fitting, thresholds application, etc.); has three abstract interfaces for the communication with the external systems
    - DQM Input (receives histograms, messages, counters)
    - DQM Output (way of publishing DQ Results produced by the DQ algorithms)
    - DQM Configuration interface (way of reading configuration info which defines behavior of the DQM Core in a specific environment)
  - **DQ Configuration is described as a hierarchical tree of objects of two different types: DQ Regions and DQ Parameters.**
  - **DQ Region**
    - Children (DQ Region or DQ Parameter)
    - DQ Summary Maker
  - **DQ Parameter**
    - location of the monitoring information (represents the state of a particular detector element)
    - weight
    - DQ Algorithm that has to be used
    - specific parameters and thresholds
    - reference values or histograms
    - the actions which have to be taken depending on the results

## DQM & QA systems in LHC experiments : ATLAS (2)

## DQ Algorithm

## DQM Framework (DQMF) provides a number of predefined DQ Algorithms

**DQ Algorithms are integrated into the DQMF in a dynamic plug-in manner**

allows adding new algorithms on the fly without modifying the core software

**each DQ Parameter has at least one DQ Algorithm associated with it**

executed whenever a piece of info which is associated with that DQ Parameter becomes available

# DQ Summary Maker

**special implementation of the DQ Algorithm interface that evaluates the DQ Result for a given DQ Region**

## DQMF Agent

**instantiates appropriate implementations of the DQMF generic interfaces (i.e. DQM Input, DQM Output and DQM Configuration)**

takes care of starting and stopping the DQM Core engine in appropriate moments

**In the online environment the DQ assessment has to be started at start of run event and stopped when the run is finished**

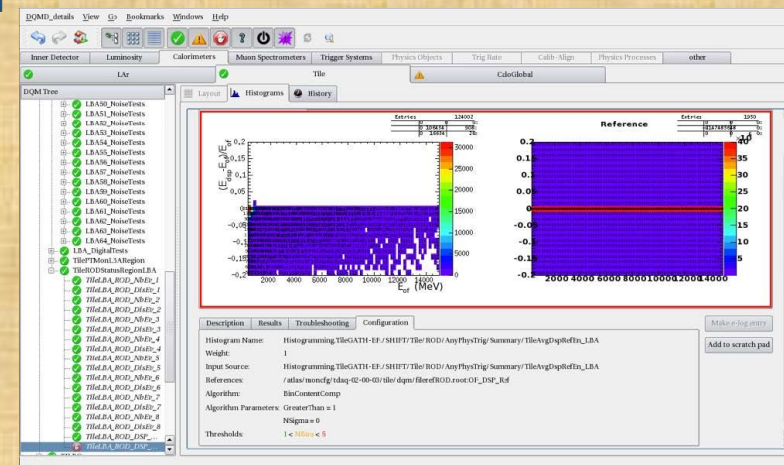
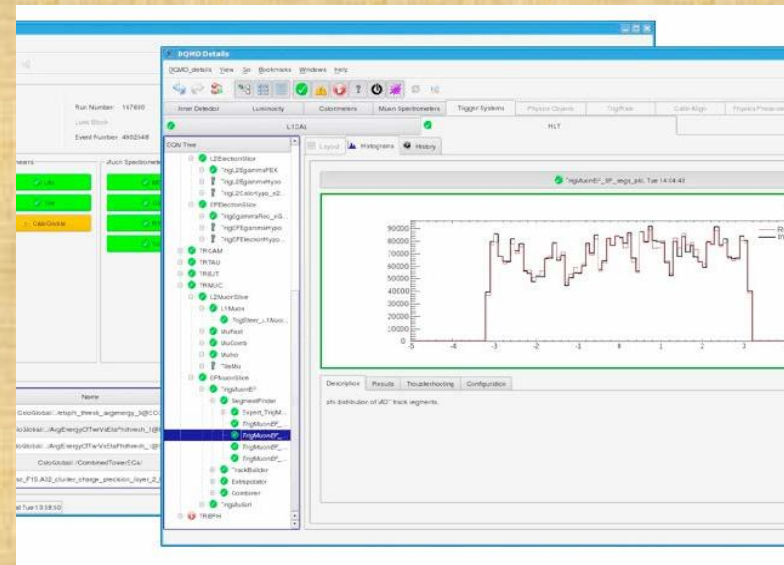
**DQMF may contain one or more DQMF Agents with each of them responsible for a well defined subset of the whole ATLAS system.**

## DQ Result

consist of a colored tag and any output that the algorithms might want to attach

If some areas of the detector are disabled, then the corresponding dq results will be **black** otherwise, results might be

green (good), yellow (warning), red (bad) or gray (undefined)





# Data Quality Monitoring (DQM) and Quality Assurance (QA) :

## Goals of the systems

- **histogram creation tool**
  - interface for histograms creation
  - library with simple implementations of this interface
- **create flexible tool for online creation, filling, transport and archival of histogram and other monitor elements**
- **create flexible online tool to perform algorithms for automated quality and validity tests**
- **keep the results of the DQM process**
- **create GUI for DQM user**
  - visualization of the histograms and quality test results
  - alarms in case of bad quality data

## Main goals

- ❑ **online and offline data quality check and results visualization**
- ❑ **fast automation reaction on data quality problems**



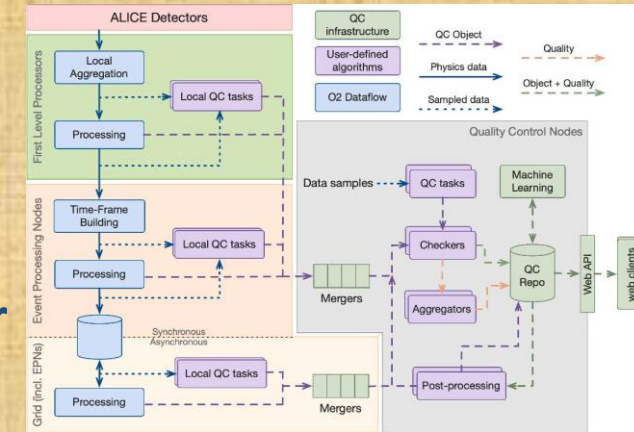
# DQM & QA systems in LHC experiments : **ALICE**

- **ALICE: DQM + QA = Data Quality Control (DQC)**

- largest DQC systems worldwide
- first in the high energy physics community to leverage the **message passing technique** and the actor model to such an extent
- high-level quality assessment of the 3.5 TB/s data produced by the detector

- **QC system**

- multi-step process
  - sampling the data, usually at a rate of 1%
  - QC Tasks will then execute user-defined algorithms to process it and generate a QC Object, often a histogram
  - Given the parallel nature of this processing, with a copy of the task running on each of the hundreds of nodes, these **histograms** are then **merged**
  - merged results are evaluated by a series of Checks to determine one or several **Qualities**, which can themselves be aggregated to give a general assessment of the health of the data
- based on a **message passing paradigm** where data flows **asynchronously** through a set of devices connected via buffered channels
- channels use ZeroMQ by passing either the whole message payloads or just pointers to the shared memory region
- QC Objects and Qualities are stored in Conditions Database



# DQM & QA systems in LHC experiments : **LHCb**

- **LHCb: collected data are grouped together in runs**
  - **LHCb DQ workflow**
    - small subset of the data selected by the trigger is fully-reconstructed on the LHCb Online computing farm
    - reconstruction produces sets of histograms which allow the (sub-)detector performance to be assessed
    - these histograms are presented by the Data Quality Monitoring(DQM) software to the DQ shifter
    - shifter compares whether the run is suitable for physics analysis or not, by comparing it to a reference run previously set by experts.
    - software package previously used in DQM shifts, was based on dedicated custom C++ code and X Window System. Now implemented "Monet" which is a python based web application that supersedes the Presenter.
  - **Update**
    - using Python as the primary language allows the usage of rich set of libraries provided in the large ecosystem of third-party Python packages. This simplifies both development, as common functionality has already been implemented elsewhere, and maintainability, as the size of the required LHCb specific code is reduced.
    - for plotting Bokeh libraries provide interactive plots in web browsers, has pythonic interface
  - **RoboShifter: automatic problem detection**
    - predicts probability of given run being bad
    - decisions made by each tree are summed with weights, representing the importance of each tree
    - each tree corresponds to a single histogram
    - possible to compute, for each histogram, its contribution to the probability of the run being bad
    - histograms with the highest contributions can be presented to DQ shifter as potentially problematic ones.
  - **Machine learning at LHCb: vector Kolmogorov-Smirnov distances btw histograms and their references; AdaBoost algorithm,**
    - track pattern recognition
    - long track reconstruction
    - downstream Track Reconstruction (reconstruction of the daughters of long-lived particles)
    - fake track rejection
    - topological trigger (HLT2)
    - jet tagging
    - charged particle identification



# DQM & QA systems in LHC experiments : CMS

- **CMS:** The DQM software is a central tool in the CMS experiment. High-level goal of the system is to discover and pin-point errors - problems occurring in detector hardware or reconstruction software
  - tools for
    - creation, filling, transport and archival of histogram and scalar monitor elements
    - standardized algorithms for performing automated quality and validity tests on value distributions
  - monitoring systems live online for
    - the detector, the trigger, and the DAQ hardware status and data throughput,
    - the online reconstruction
    - validating calibration results, software releases and simulated data
  - visualization of the monitoring results
  - certification of datasets and subsets thereof for physics analyses
  - retrieval of DQM quantities from the conditions database
  - standardization and integration of DQM components in CMS software releases
  - organization and operation of the activities, including shifts and tutorials



# The DQM tree

DQM Configuration keeps set of DQM Tree in JSON format.

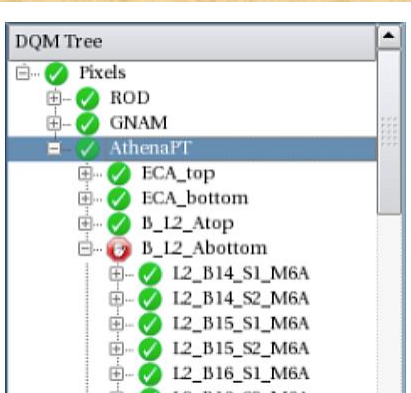
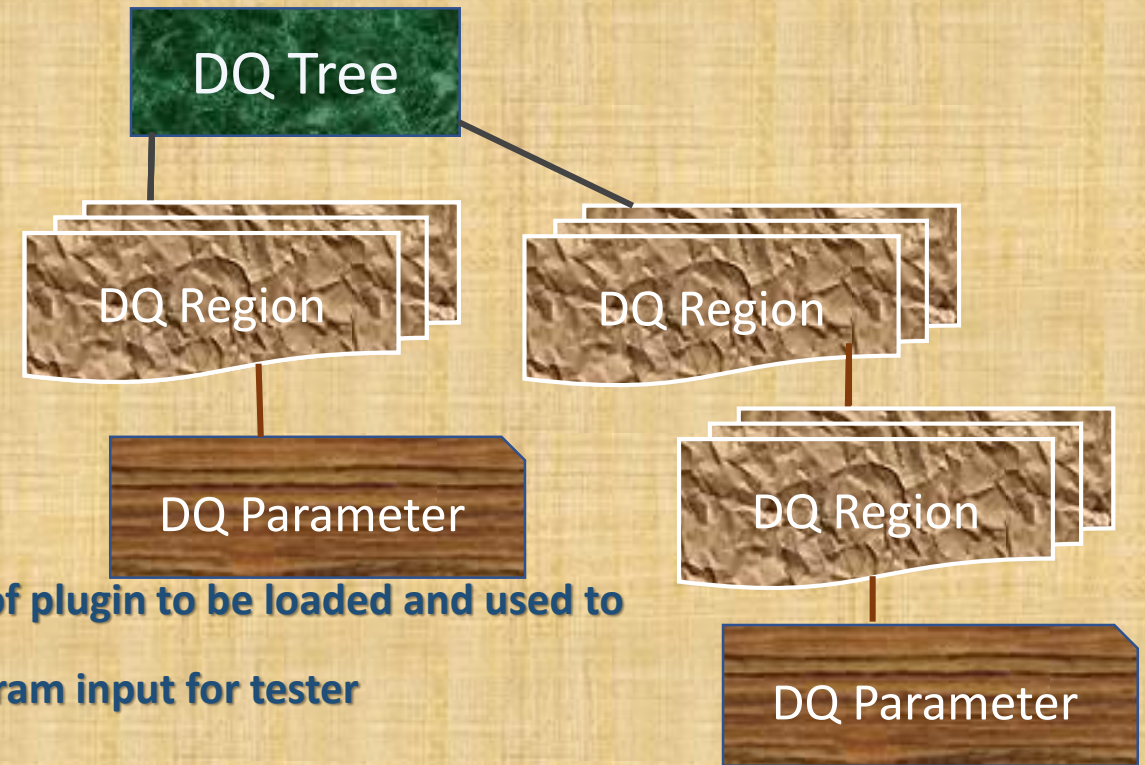
- **DQ Tree**

- Histogram producer library (plugins)
- Analyzer library (plugins)

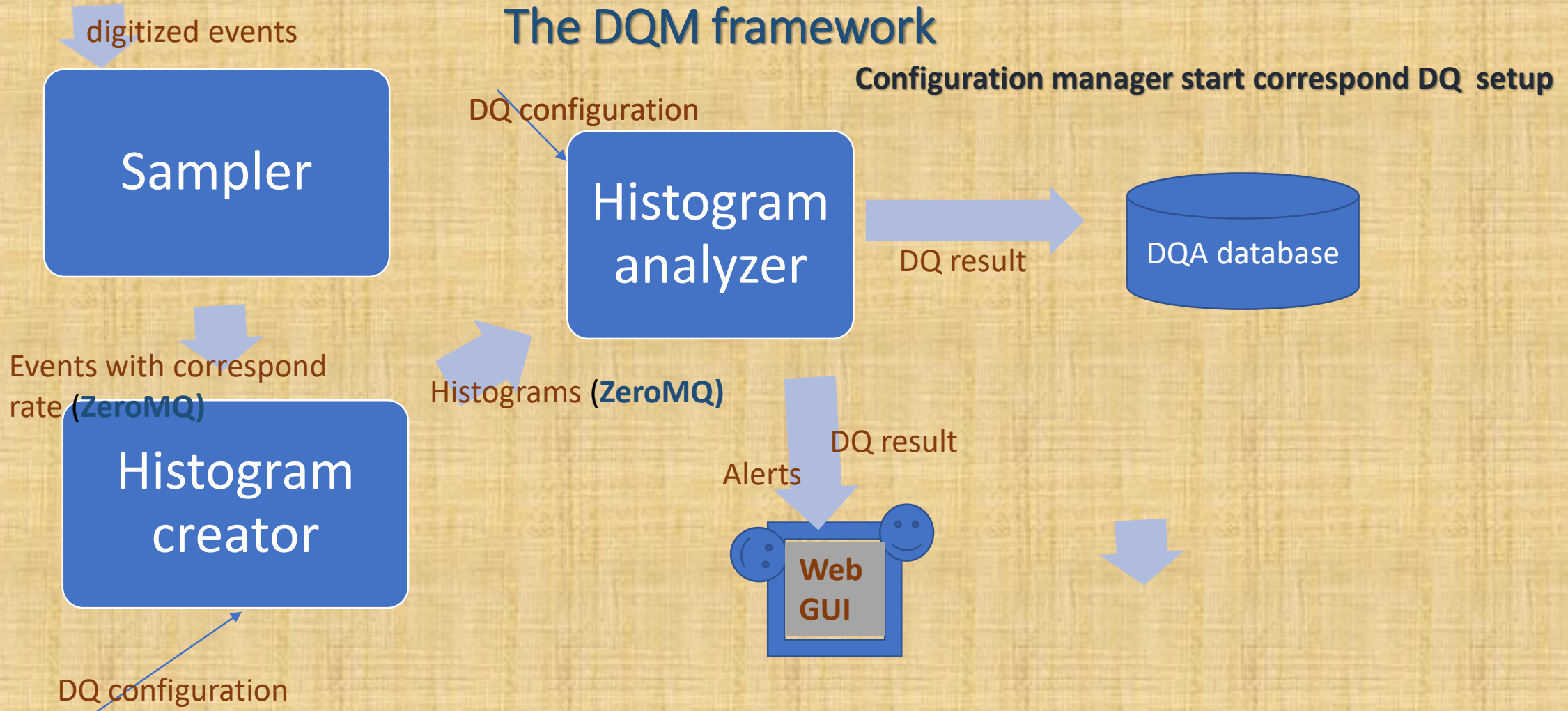
- **DQ Region ... DQ Region**

- DQ summary maker
- DQ Parameter
  - Input from sampler
  - Histogram producer

- Algorithm for histogram creation (name of plugin to be loaded and used to produce histograms)
- Output with histogram produced - histogram input for tester
- Analyzer (quality tester)
  - histogram input (from previous output)
  - Algorithm (quality checker algorithm)
  - Threshold (numbers to understand if histogram good or bad)
  - Output
    - result (good, bad, undefined, disable)
    - DQ Tree path
    - histogram



# The DQM framework



# Example of the possible DQM configuration

```
{
  "DQM_SETUP": "setup1",
  "histogram_library": "lib/histograms",
  "analyzer_library": "lib/dqm_analiz",
  "tasks": [
    {
      "task_gem" : {
        "histogram": {
          "tree_path" : "GEM/Silicon",
          "plugin_name" : "statistics",
          "cumulative": : "true",
          "input": "host1:20001",
          "output1": "host2:20002",
          "output2": "host2:20003",
          "info": {
            "name": "statistics_gems",
            "title": "Statistics GEM Silicon",
            "x title": "Number of digits in event silicon",
            "y title": "Number of digits in event GEM",
            "category" : "D", "//comment1":"C,S,I,L,F,D",
            "Dimensions" : {
              "XDim" : { "xbins": "40", "xlow": "-3", "xup" : "100" },
              "YDim" : { "ybins" : "50", "ylo": "0", "yup" : "60" }.
              "ZDim" : { "zbins" : "0", "zlow" : "0", "zup" : 0 }
            }
          }
        },
        "Options" : { "//comment2" : "draw options,
          color options, text options, CONT options, LEGO
          options, SURface options etc",
          "opt_lego" : "lego",
          "c_opt" : "c",
          "arr_opt" : "arr",
          "line_color" : "red",
          "fill_color" : "grey"
        }
      },
      "analyze" {
        dbase": "db:sql2000",
        [ {
          "analyze1" : {
            "plugin_name": "empty_check",
            "cumulative" : "true",
            "parameters" : {
              "threshold1" : "10",
              "threshold2" : "15"
            }
          },
          "draw_out": "DQMmonitoring1.html"
        }
      ],
      { "analyze2" : {
        "plugin_name": "empty_check",
        "cumulative" : "false",
        "parameters" : {
          "threshold1" : "1",
          "threshold2" : "4"
        }
      },
      "draw_out": "DQMmonitor2.html"
    }
  ], //comment3 : "END task_gem"
}, { "task_gem_station" : {
  "input": "host1:20004",
  "etc" : "....."
}
}, {
  "sampler" : {
    "input": "host1:20000",
    "out" : ["host1:20001","host2:20004"]
  },
  "rate" : [ "10", "5" ]
}
}
}
```