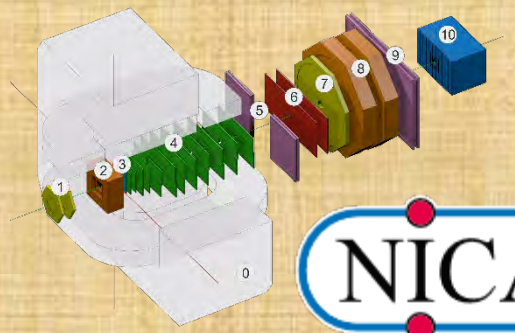
Data quality monitoring and assurance system prototype: status of implementation

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



NICA BM@N

Joint Institute for Nuclear Research

May 12 – 14, 2026 VBLHEP JINR
16th Collaboration Meeting of the BM@N Experiment at NICA

Outlines

- Goals of the system creation
- General architecture
- The DQM processes and data flow diagram
- DQA configuration
- Task manager functionality
- DQA base module
- Main classes diagram
- Database schema
- Status of the prototype implementation
- Conclusion and next steps

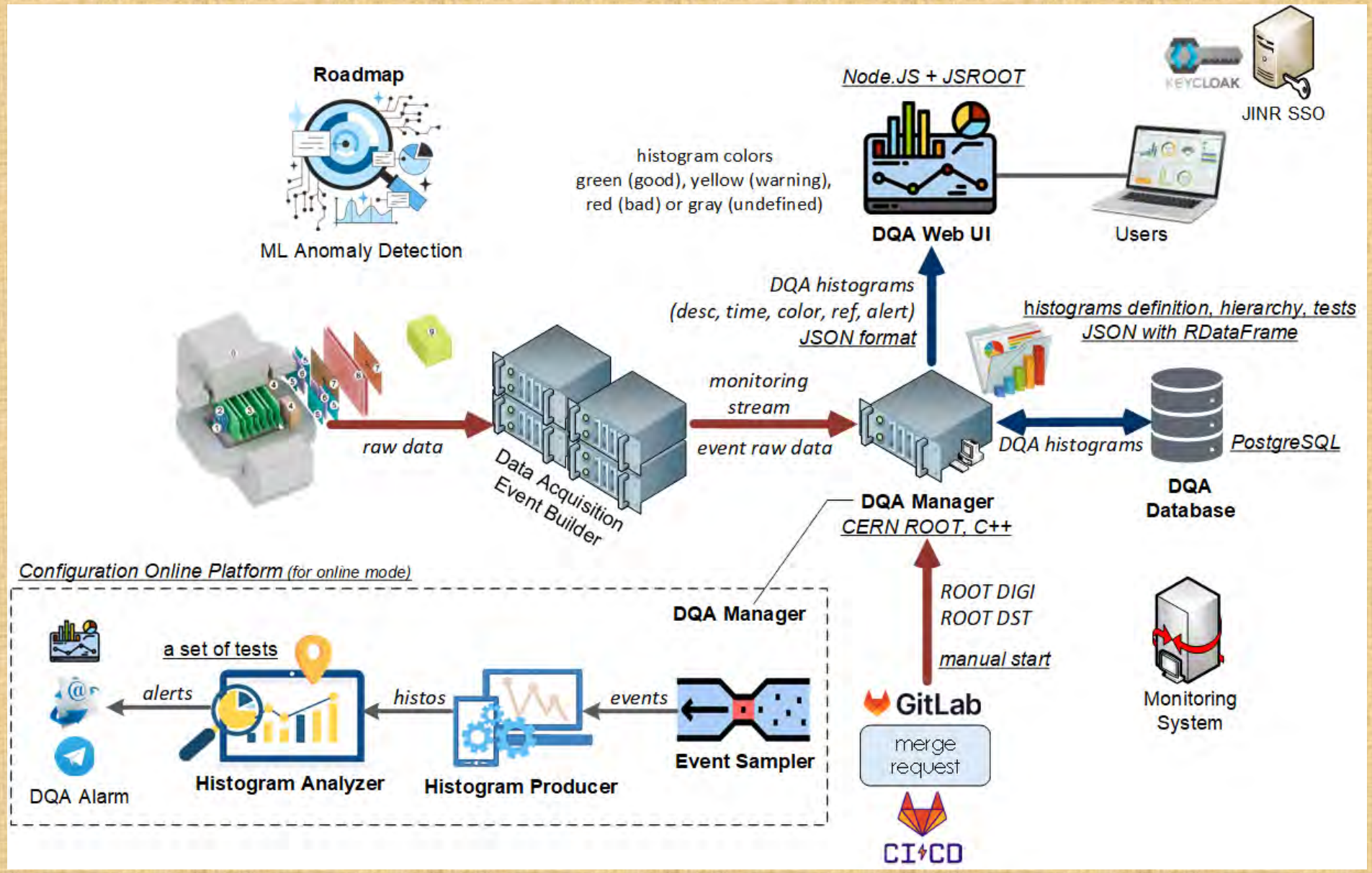
Data Quality Monitoring (DQM) and Quality Assurance (QA) : Goals of the systems

- **Data Quality Monitoring and Quality Assurance (DQA) 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 (DQM part)**
  - **for qualitative assessment of new BmnRoot versions (after git merges)**
  - **for manual run to check user versions of the software**

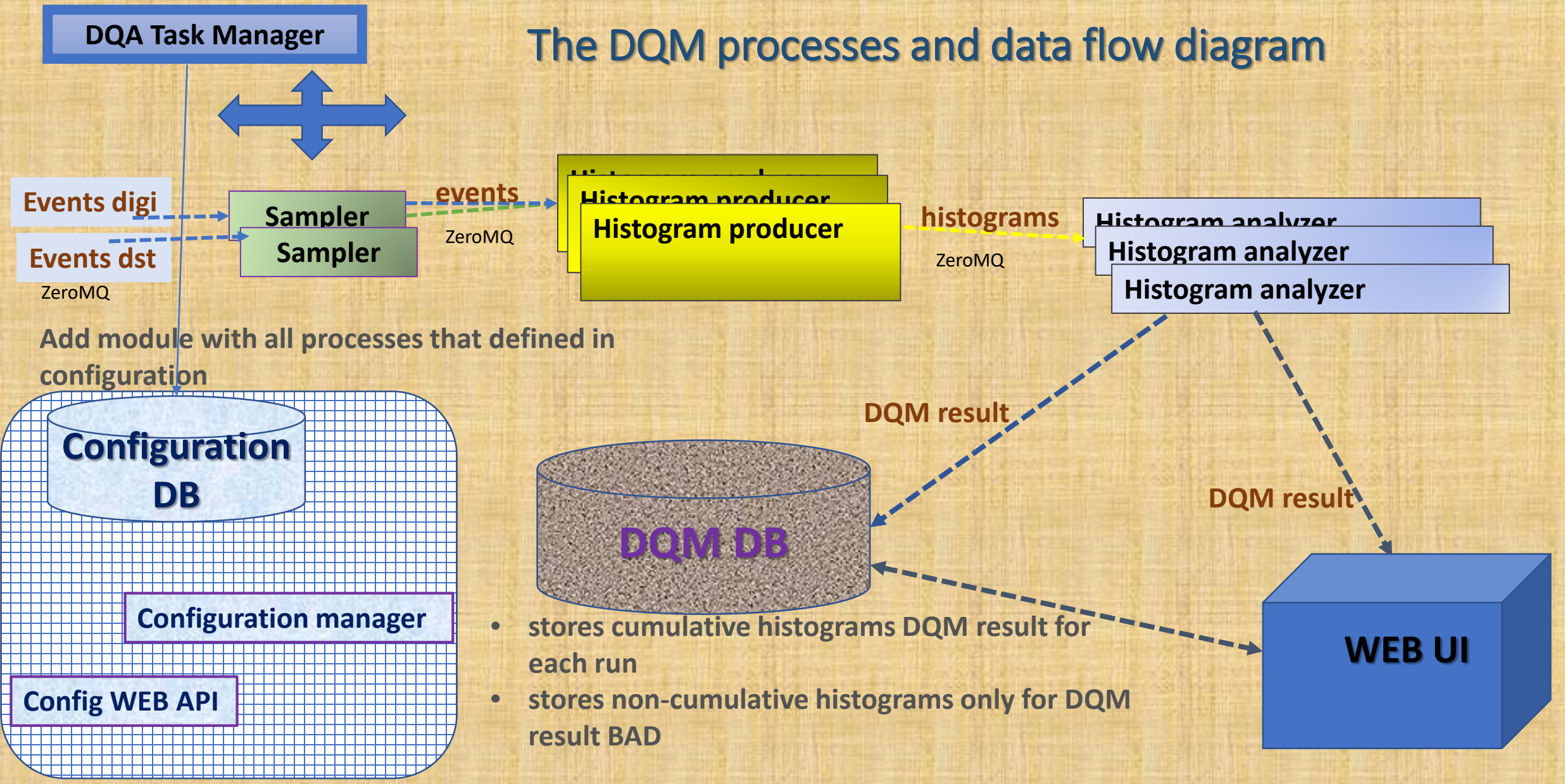
- **The BM@N DQA architecture must ensure *predefined checks* and graphical representation of control histograms on a central Web service, which receives the displayed data distributions from the BM@N histogram producer (DQA central manager)**

- **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**

The DQM General architecture



The DQM processes and data flow diagram



- stores cumulative histograms DQM result for each run
- stores non-cumulative histograms only for DQM result BAD

DQM configuration structure

- JSON is used as a format of configuration description
- Two types of configuration file:
  - general system configuration (in one file, managed by system administrator)
  - histogram and analyses algorithms definition (can be set of files, managed by detector groups)

DQM general system configuration structure

- General system configuration includes:
  - DQA module common data
    - setup name, library paths, default addresses etc
  - array of configuration tasks
    - task name
    - input and/or output streams for sampler, producer, analyzer, web and DB
    - types of stream interfaces
    - path to histogram configuration file
  - definition of streams
    - stream type, name, real address

Example of the general system configuration

```
{  "DQM_SETUP": "setup1",
    "histogram_lib": "lib/histograms_produce_alg",
    "analyzer_lib": "lib/dqm_analis_alg",
   "sampler_path" : "~/dqa_sampler",
   "producer_path" : "~/dqa_producer",
   "analyzer_path" : "~/dqa_analyzer",
"defaults for all tasks:" : "see below",
 "database" : "host3:20003",
 "web" : "host4:8000",
 "sampler" : "samplerDigiFile1",
 "rate" : 1,
"socketTypeInd" : [1,0,0],
  "tasks": [
 { "name" : "task_gem",
   "ddata"  : "digi1",
   "rate" : 10,
   " histo_ref " : "~/configHisto0.json",
   "pdata" : "producer_adddr1",
   "astream" : ["analyzer_addr1", "analyzer_addr3" ]
 },
 { "name" : "task_gem_dst",
   "dsdata" : "dst1",
   "sampler" : "samplerDst",
   "rate" : 10,
   " histo_ref " : "~/configHisto0.json",
   "pdata" : "producer_adddr1",
   "astream" : ["analyzer_addr1", "analyzer_addr3" ]
 },
{ "name" : "task_gem1",
  "ddata"  : "digi1",
  " histo_ref " : "~/configHisto1.json",
  "pdata" : "producer_addr2"
 },
 { "name" : "task_gem2",
   "dsdata" : "dst1",
   "sampler" : "samplerDst",
   " histo_ref " : "~/configHisto2.json",
   "pdata" : "producer_addr2“,
  "socketTypeInd" : [2,0,0],
 },
 { "name" : "task_gem0",
   "dsdata" : "dst1",
   "sampler" : "samplerDst",
  "pdata" : ["producer_addr2"],
  " histo_ref " : "~/configHisto3.json",
  "wdata" : "webname3"
 }
],
 "Streams" : [
{"ddata"  : "digi1",        "address" : "host1:20001"},
{"dsdata" : "dst1",         "address" : "host2:20001"},

{"sampler" : "samplerDigiFile1",     "address" : “test/filePath"},
{"pdata" : "producer_addr1", "address" : "host4:20003"},
{"pdata" : "producer_addr2", "address" : "host4:200033"},

{"adata" : "analyzer_addr1", "address" : "host5:20002"},
{"adata" : "analyzer_addr2", "address" : "host5:200022"},
{"adata" : "analyzer_addr3", "address" : "host5:200022"},
{"adata" : "analyzer_addr0", "address" : "host5:20003"},

{"wdata" : "webname1",      "address" : "host6:20000"},
{"wdata" : "webname2",      "address" : "host6:20001"},
{"wdata" : "webname3",      "address" : "host6:20006"},
{"dbdata" : "db1",          "address" : "host7:20000"}
]
}
```

Configuration: Histogram and analyzer description

- Histogram configuration structure
  - config name
  - tree-paths
  - produce algorithm name, filter expression (if algorithm is filter)
  - histogram info
    - name, title, xtitle, ytitle, dimension
    - histogram info options if differ from default values
      - draw options, color options, text options, SURFace options etc.
  - Array of analyses algorithm description
    - Name, is enable, is cumulative, min and max thresholds
- Rules for supporting histogram description configuration files
  - each detector group will describe histograms only related to their tasks
  - histograms not related to the group will not be contained in their configuration file
  - each histogram producer program loads its related configuration file with histogram description during initialization

Example of the histogram configuration

```
{
"hist_conf_name":"filter1",
 "histogram" : {
    "tree_path" : ["GEM/Silicon","GEM/CSC"],
    "algname" : "filter",
    "filter"  : "p > 5",
    "info" : {
      "name" : "statistics_gems p5",
      "title" : "Statistics GEM Silicon p5",
      "xtitle" : "Number of digits in event silicon",
      "ytitle" : "Number of digits in event GEM",
      "dimension" : 2 , "//comment1" : "TH2D IN USE",
      "options" : {
        "//comment2" : "ROOT HISTOGRAM OPTIONS, ADDITIONAL TO DEFAULTS,
              like draw options, color options, text options, SURFace options etc",
        "c_opt" : "c",
        "arr_opt" : "arr",
        "line_color" : "red",
        "fill_color" : "grey"
      }
    }
  }
```

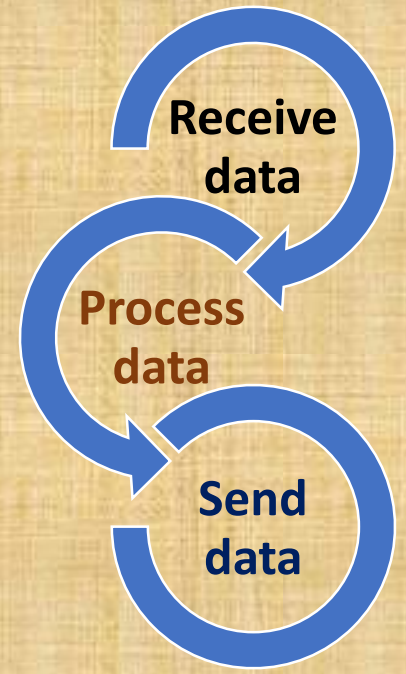
```
, "analyzers" : [
    { "alg_name" : "kolmogorov_alg",
      "enable"   : false,
      "cumulative" : true,
      "thresholdMin" : 10,
      "thresholdMax" : 15
    },
    { “alg_name”: “empty_check”,
      "enable"   : true,
      "cumulative" : false,
      "thresholdMin" : 1,
      "thresholdMax" : 4
    }
  ]
}
```

Task manager functionality

- Task manager (TM) possibilities
  - loads configuration and produce configuration task objects
  - produces samplers, producers and analyzers with following possible modes depending on configuration
    - creates configuration tasks as separate module in DB of Configuration system (for DQM)
      - Tasks of module can be started through WEB GUI as part of online setup
    - Starts by itself all producers, samplers and analyzers as separate processes
    - starts all above objects in the same process but in separate threads
    - Starts sampler, producer and analyzer in one thread (case with only one task in configuration)

DQA base module

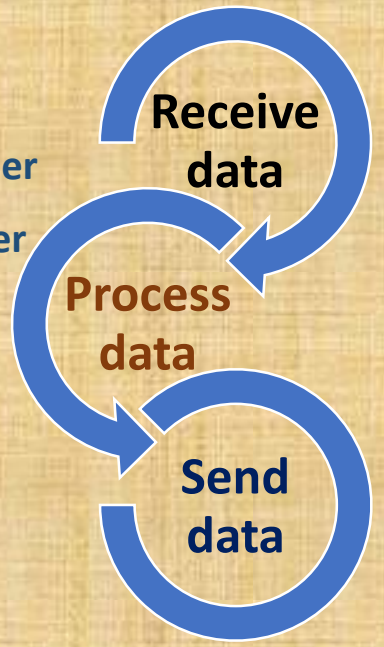
- Sampler, Producer and Analyzer have the same principals of the functionality running in loop
  - use zeroMQ library for multiprocessing data interaction
  - receive and send methods in processes should be consistent each other
    - take it into account in configuration
  - all these processes working in similar way:
    - start unlimited loop, inside loop perform 3 actions:
      - receive data, process data in limited time, send the result
  - DQABaseModule: parent class with virtual methods for data processing



Main classes diagram

TaskManagerDQA
json confObj
vector<TaskDQA*> tasks
…

- static int socketTypeS[] = { ZMQ_PUB, ZMQ_PUSH, FILE_OUT}; //zmq patterns for Sender
- static int socketTypeR[] = { ZMQ_SUB, ZMQ_PULL, FILE_IN }; //zmq patterns for Reciever



TaskDQA
string _name
string _evntAddr
string _sampAddr
string _histRef
…

ReceiverDQA
virtual int receiveData()
virtual TTree* receiveTTreeData()
byte* getBuffer()
…

ReceiverZMQ

ReceiverFile

BaseModuleDQA
virtual void createComponents()
string _name
SenderDQA* _sender
ReceiverDQA* _receiver
ProcessorDQA* _processor
…

SamplerDQA

ProducerDQA

AnalyzerDQA

SenderDQA
virtual int sendData(int sendSize)
virtual int sendTreeData(TTree* sendTree)
virtual void allocateBuffer(int len)
…

SenderZMQ

SenderFile

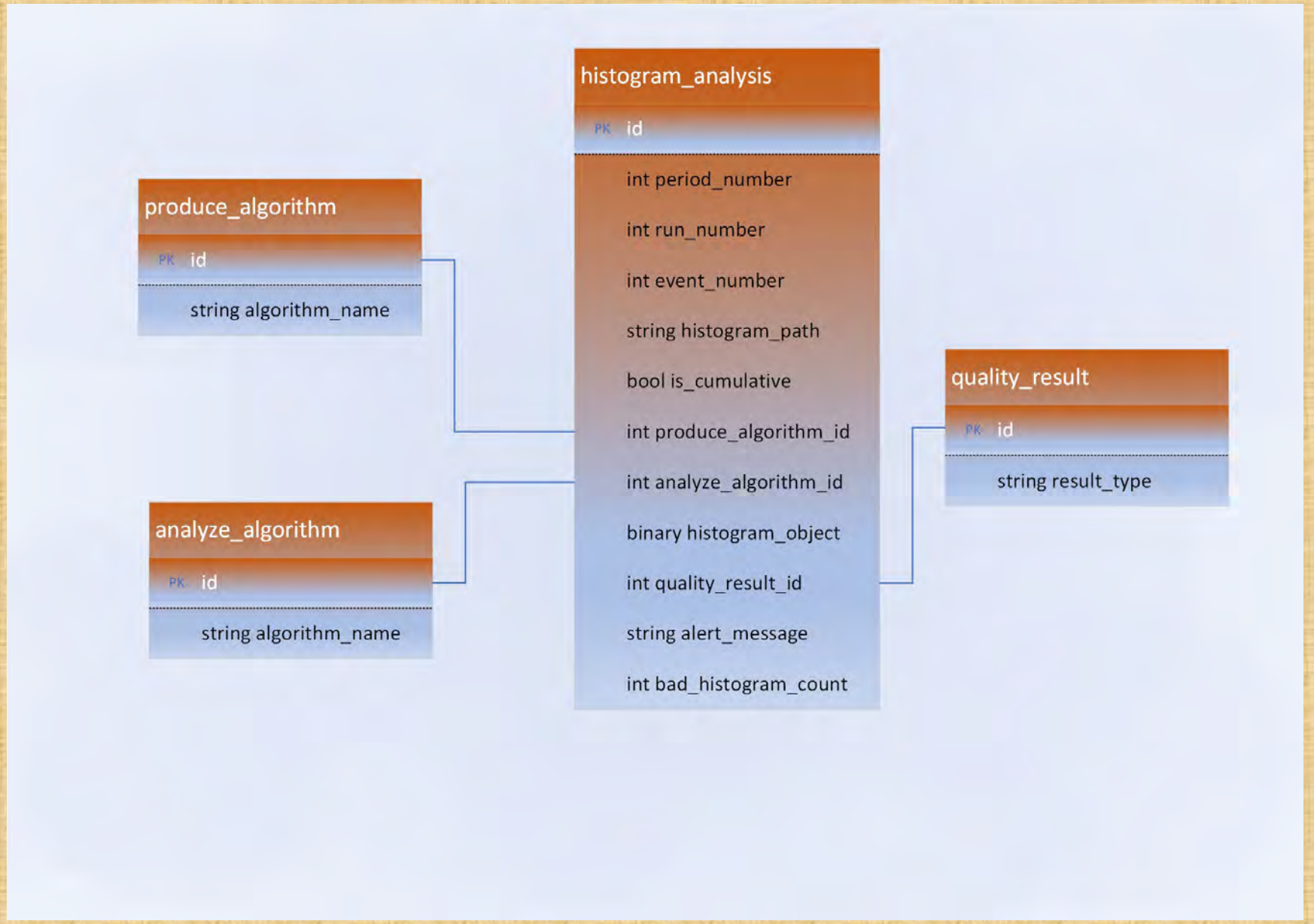
ProcessorDQA
virtual int processData(byte* inbuffer, int lenIn, byte* outbuffer)
virtual int processData(TTree* inTree, TTree* resTree)
…

ProcessorSampler

ProcessorProducer

ProcessorAnalyzer

DQM database schema



Status of the prototype implementation (1)

- The design and implementation of the configuration have been updated
- Status of base classes that should be implemented:
  - Configuration
    - ConfigUtils, ConfHistInfo, ConfHisto, ConfAnalyzer, ConfTask, ConfigDQA
      - 90%
  - Connections support
    - ReceiverDQA, ReceiverZMQ, ReceiverFile, SenderDQA, SenderZMQ, SenderFile
      - 90%
  - Task management
    - TaskManagerDQA: 10%
    - TaskDQA: 90%, ProcessorDQA: 99%, ProcessorSampler: 95%, ProcessorProducer, ProcessorAnalyzer
    - BaseModuleDQA: 95%, SamplerDQA: 95%, ProducerDQA: 90%, AnalyzerDQA

Done
Written and compiled

Status of the prototype implementation (2)

Done
Written and compiled

- Status of base classes that should be implemented:
  - Produce algorithms
    - ProduceAlgorithm: 90%, ProduceFilterAlgorithm: 10%
  - Analises algorithms
    - AnalysisAlgorithm, AnalysisCalcAlgorithm
      - 5%
  - Binaries
    - sampler.cxx : 95%
    - producer.cxx : 90%
    - analyzer.cxx : 5%

Overall prototype implementation status: approximately 40% complete (excluding the WEB API)

Conclusion and next steps

- Design was updated
  - Configuration now more user friendly
  - DQM task manager added
- The prototype implementation is approximately 40% complete (apart of WEB API)

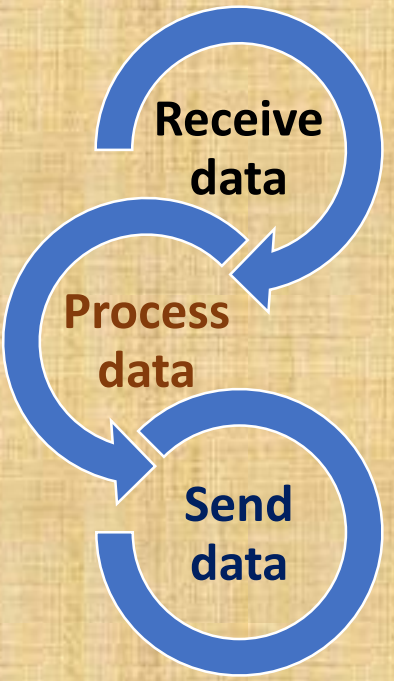
NEXT steps

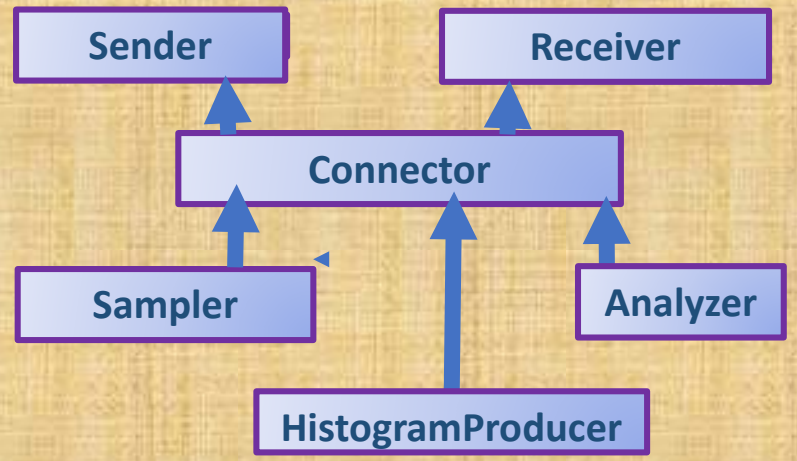
- Complete the task creation and management portion of the prototype
- Start the web GUI design and implementation

BACKUP

Communication implementation

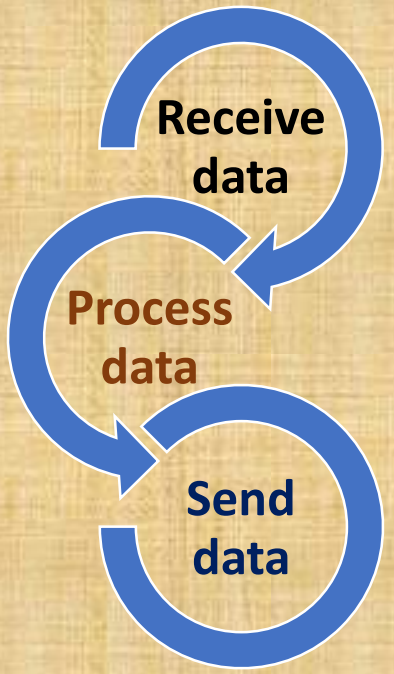
- Use
  - push-pull (base)
    - garantee rate as can be several pull receivers
  - publish-subscribe
- Send/receive 2 types of data
  - byte buffer
    - Sends all data
  - TTree
    - Sends only those tree paths data that mentioned in histogram configuration
- All is written, some parts are under debug

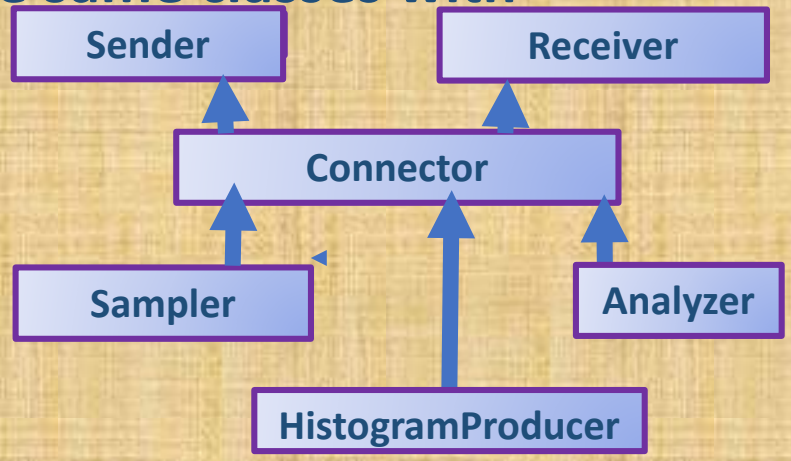




Communication implementation

- Sampler, Producer and Analyzer have the same schema of the functionality running in loop
  - use zeroMQ library for multiprocessing data interaction
  - processes receive and send methods should be consistent each other, taking it into account in configuration
  - all these processes working in similar loop: receive data, process data in limited time, send the result
  - implement common functionality in the same classes with virtual methods for data processing

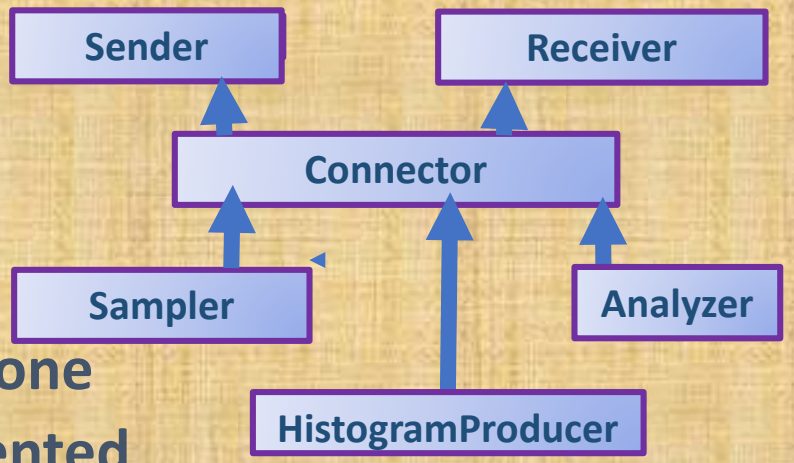




Alarms description

- 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

DQA communication



- Creation of the system prototype was just started
  - classes to keep all configuration is implemented
  - load configuration from json file and print its data is done
  - save loaded configuration in json format also implemented (to be used for configuration creation in future GUI tools)
  - sampler implementation is in progress

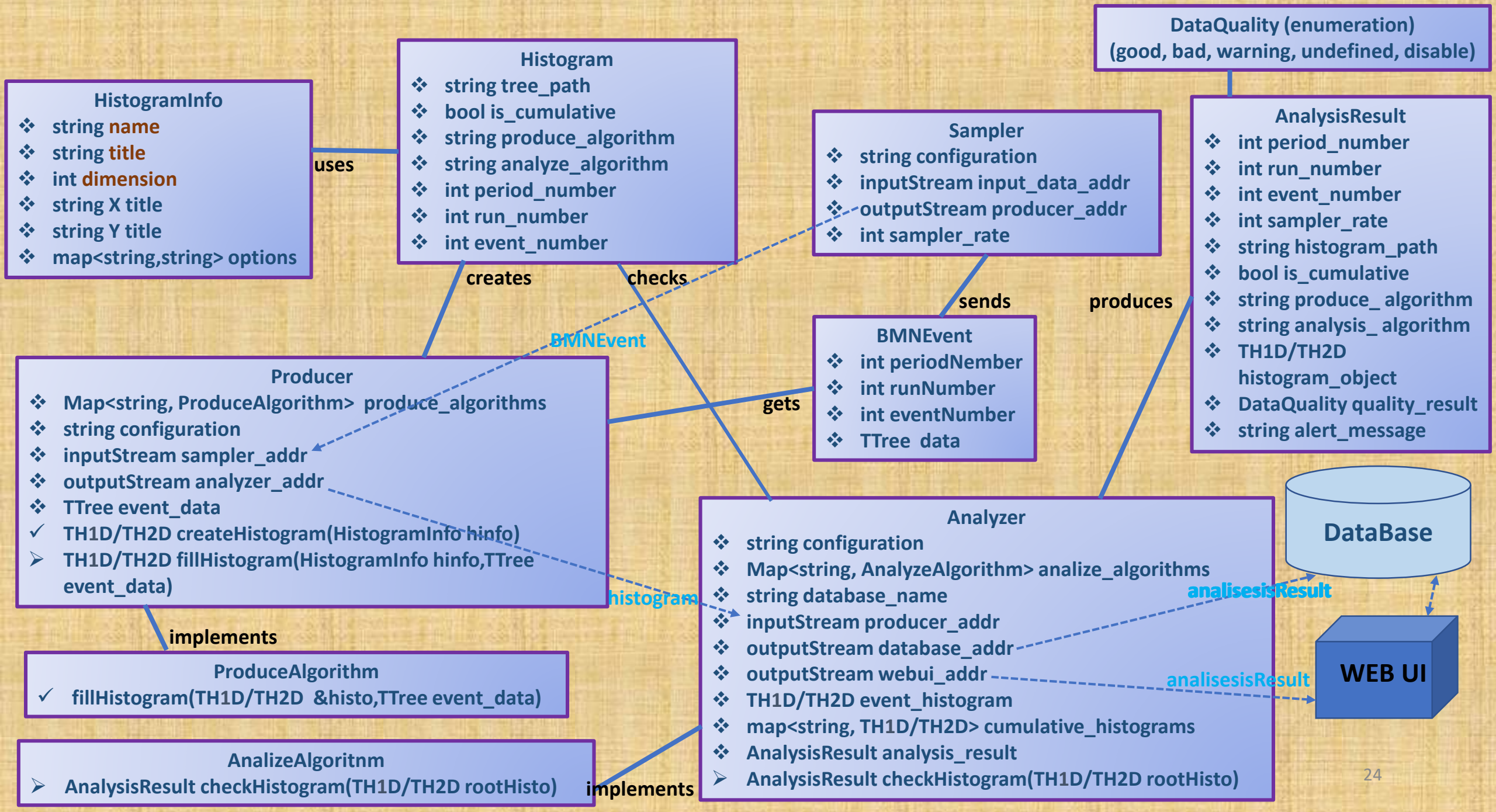
Example of the possible DQM configuration

```
{  "DQM_SETUP": "setup1",
    "histogram_lib": "lib/histograms_produce_alg",
    "analyzer_lib": "lib/dqm_analis_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",
         "xtitle": "Number of digits in event silicon",
         "ytitle": "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"
       }
    }:wq
```

```
"analyzers" {
 [ {
        "analize1" : {
           "alg_name": "kolmogorov_alg",
           "cumulative" : true,
           //"parameters" : {
             "threshouldMin" : 10,
             "threshouldMax" : 15
           // }

        }
   },{ "analize2" : {
          "alg_name": "empty_check2",
          "cumulative" : false,
          // "parameters" : {
            "threshouldMin" : 1,
            "threshouldMax" : 4
          //}
      }
   } ] }
    } , "//comment3" : "END task_gem"
},{
   "task_gem_station" : { "histogram": {
         "producer_input": "host1:20004",
          "etc" : "....."          }
  } } ]
```

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


DataQuality (enumeration)
(good, bad, warning, undefined, disable)
HistogramInfo
string name
string title
int dimension
string X title
string Y title
map<string,string> options
uses
Histogram
string tree_path
bool is_cumulative
string produce_algorithm
string analyze_algorithm
int period_number
int run_number
int event_number
Sampler
string configuration
inputStream input_data_addr
outputStream producer_addr
int sampler_rate
AnalysisResult
int period_number
int run_number
int event_number
int sampler_rate
string histogram_path
bool is_cumulative
string produce_ algorithm
string analysis_ algorithm
TH1D/TH2D histogram_object
DataQuality quality_result
string alert_message
creates
checks
sends
produces
BMNEvent
BMNEvent
int periodNember
int runNumber
int eventNumber
TTree data
gets
Producer
Map<string, ProduceAlgorithm> produce_algorithms
string configuration
inputStream sampler_addr
outputStream analyzer_addr
TTree event_data
TH1D/TH2D createHistogram(HistogramInfo hinfo)
TH1D/TH2D fillHistogram(HistogramInfo hinfo,TTree event_data)
histogram
Analyzer
string configuration
Map<string, AnalyzeAlgorithm> analize_algorithms
string database_name
inputStream producer_addr
outputStream database_addr
outputStream webui_addr
TH1D/TH2D event_histogram
map<string, TH1D/TH2D> cumulative_histograms
AnalysisResult analysis_result
AnalysisResult checkHistogram(TH1D/TH2D rootHisto)
analisesisResult
analisesisResult
DataBase
WEB UI
implements
ProduceAlgorithm
fillHistogram(TH1D/TH2D &histo,TTree event_data)
implements
AnalizeAlgoritnm
AnalysisResult checkHistogram(TH1D/TH2D rootHisto)

Possible view for DQM shifter WEB GUI

Time: 15:04:07 Run Number: 2900
Run State: RUNNING Event Number: 150036
State: **GOOD**

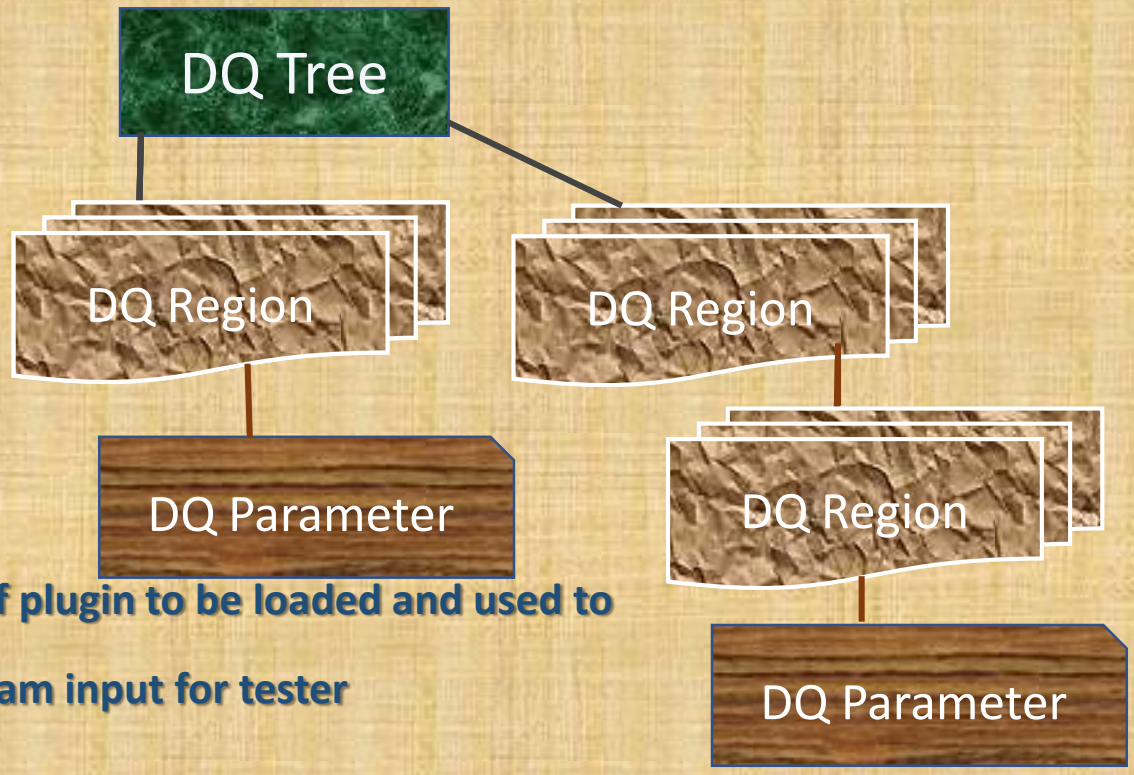
- Root TREE
- GEM Station
  - Module 0
    - Layer 0
    - Layer 1
  - Module 1
  - Module 2
  - ……..
- GEM Silicon

HISTOGRAMS

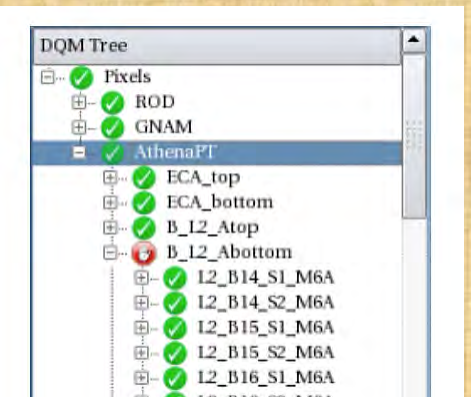
ALAM LOG

The DQM tree

DQM Configuration keeps set of DQM Tree in JSON format.



- **DQ Tree**
  - Histogram producer library (plugins)
  - Analizer 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



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

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

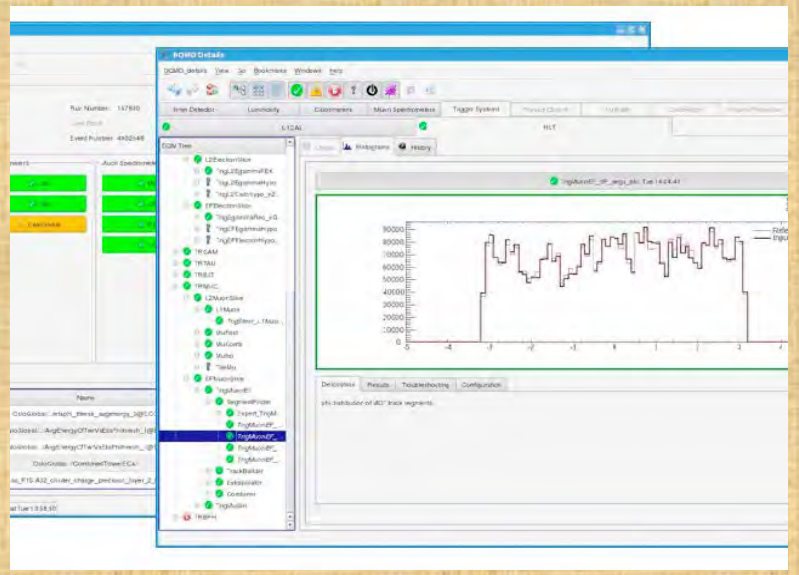
- special implementation of the DQ Algorithm interface that evaluates the DQ Result for a given DQ Region

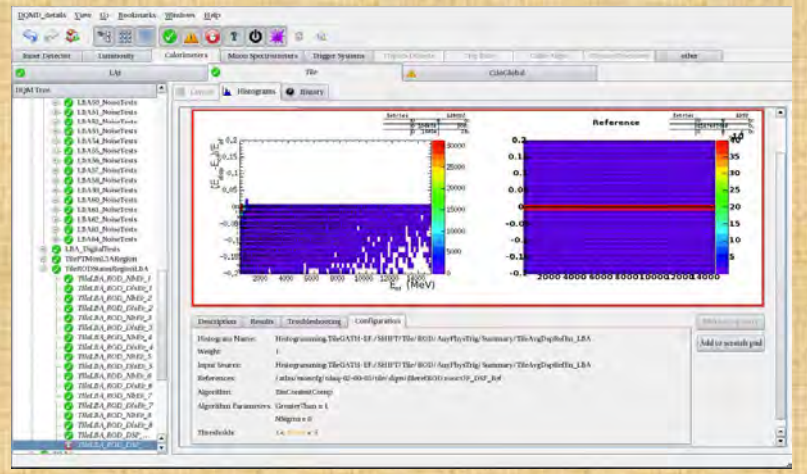
DQMF Agent

- 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)





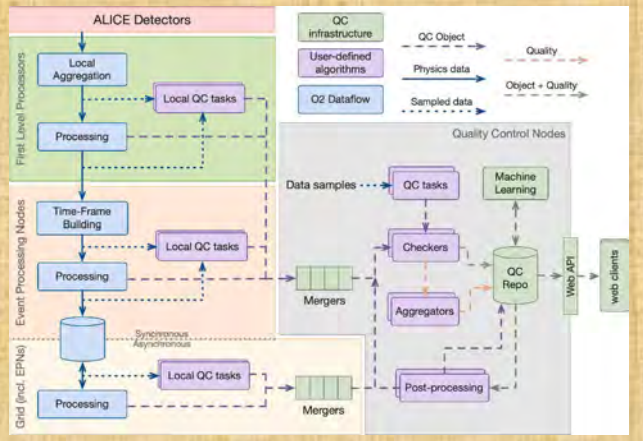
Data Quality Monitoring (DQM) and Quality Assurance (QA) : Goals of the systems

- 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

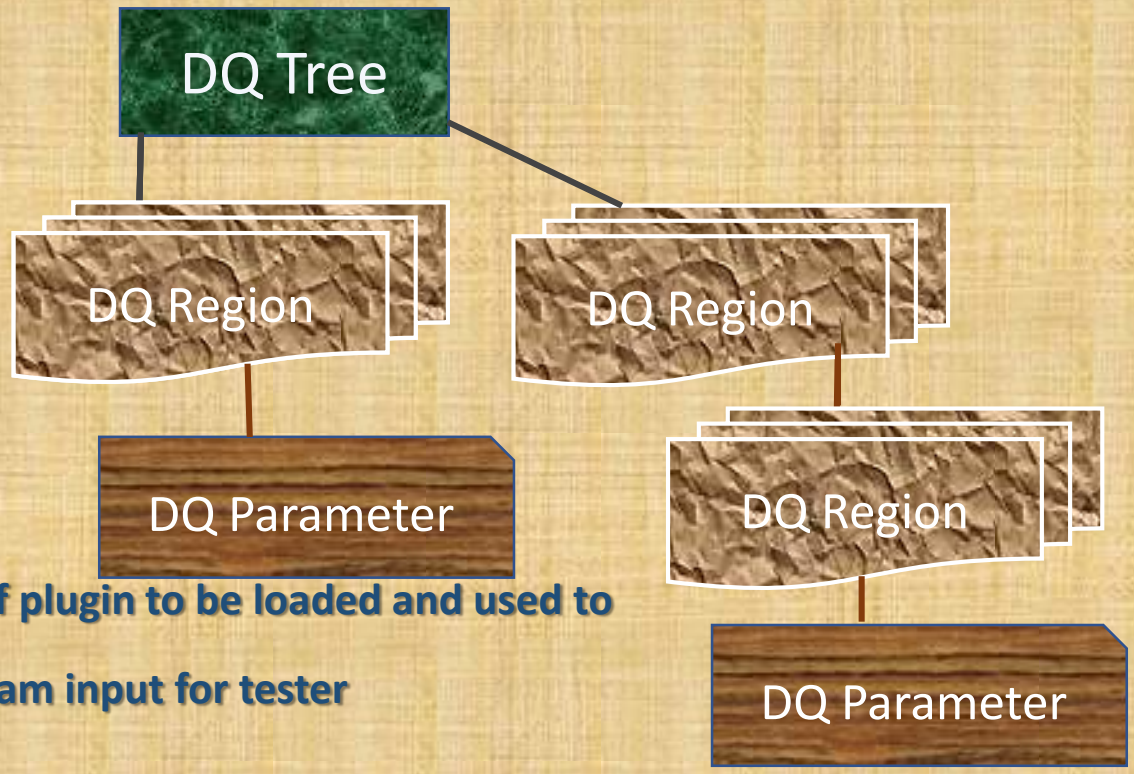
- **LHCb: collected data are grouped together in runs**
  - **LHCb DQ workflow**
    - small subset of the data selected by the trigger is fully-reconstructed on the LHCb Online computing farm
    - reconstruction produces sets of histograms which allow the (sub-)detector performance to be assessed
    - 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

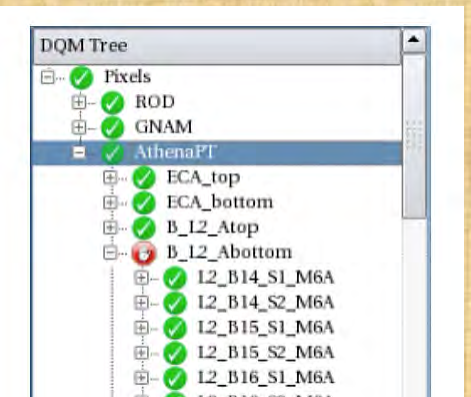
- **CMS: The DQM software is a central tool in the CMS experiment. High-level goal of the system is to discover and pin-point errors - problems occurring in detector hardware or reconstruction software**
  - tools for
    - creation, filling, transport and archival of histogram and scalar monitor elements
    - 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

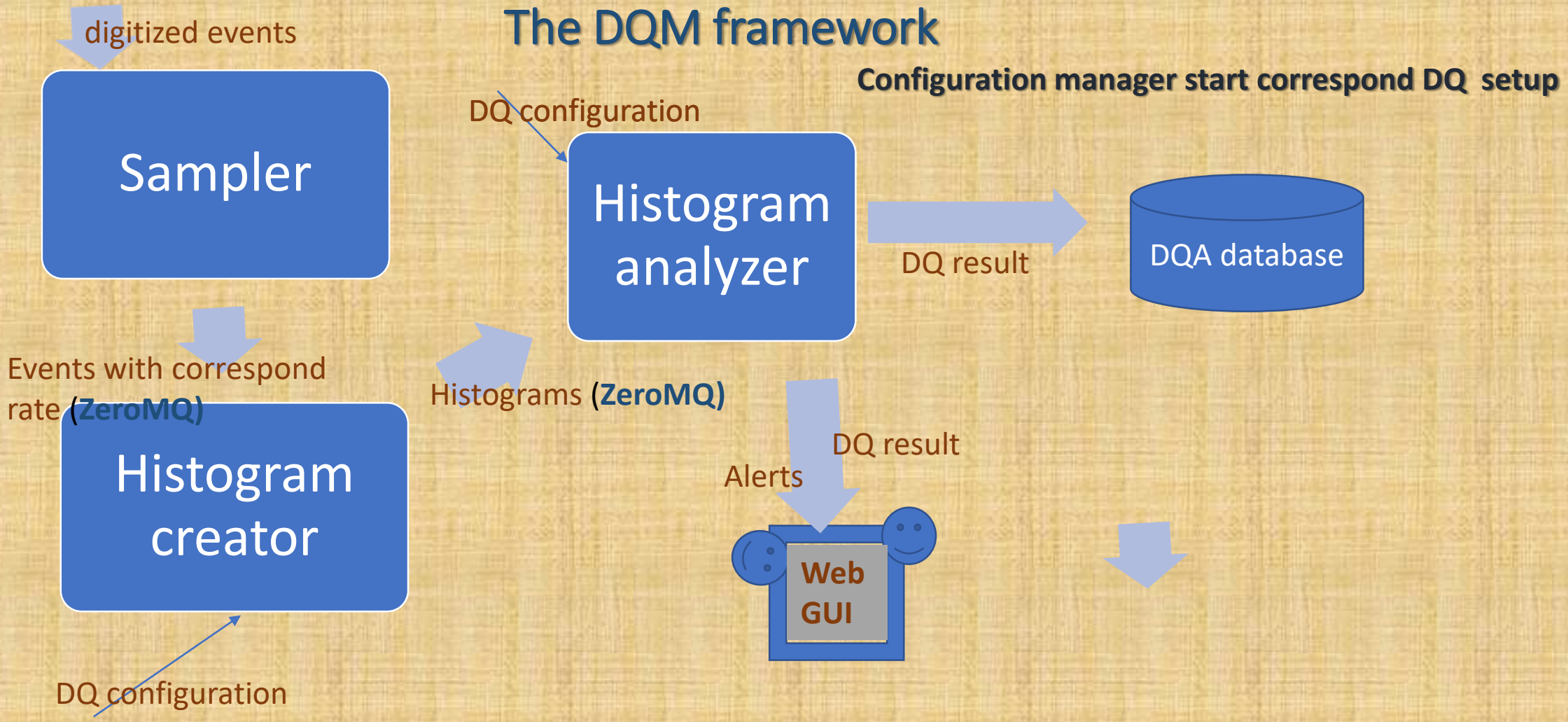
DQM Configuration keeps set of DQM Tree in JSON format.



- DQ Tree
  - Histogram producer library (plugins)
  - Analizer 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",
  "analizer_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" : { "xbin":"40", "xlow": "-3", "xup" : "100" },
          "YDim" : { "ybin" : "50", "ylow" : "0", "yup" : "60" }.
          "ZDim" : { "zbin" : "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"
          }
      }
    }
    "analize" {
        dbase": "db:sql2000",
[ {
        "analize1" : {
           "plugin_name": "empty_check",
           "cumulative" : "true",
           "parameters" : {
             "threshould1" : "10",
             "threshould2" : "15"
           }
          "draw_out": "DQMmonitoring1.html"
        }
    },{ "analize2" : {
          "plugin_name": "empty_check",
          "cumulative" : "false",
          "parameters" : {
            "threshould1" : "1",
            "threshould2" : "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" ]
 }
 } ]
}
```