Design and Implementation of a Monitoring System

Serguei Kolos
(University of California, Irvine)

The Outline

- Introduction: The Key Principals of Online Monitoring
- Monitoring in a DAQ system
- Scaling up to the size of the HEP experiments
- Summary & Final Remarks

What Monitoring is used for?

- A good Monitoring System should be capable to answer all possible questions about the system being monitored:
  - What happened?
  - Where that happened?
  - When that happened?
- There are some questions it is not competent to answer, e.g.:
  - What to do next?
- But it tries to provide all possible information to those who can answer such questions

"I swear to tell the truth, the whole truth, and nothing but the truth, from my perspective."

How a Monitoring System does its job?

- All of us are equipped with a perfect monitoring system:
  - We have 5 types of sensors for detecting events in the outside space
  - The information is transferred via nerve fibers to our brains which initiates an appropriate reaction
  - Some information is memorized for the future reference
- Monitoring for a complex HW+SW system acts in a similar way:
  - The HW and SW elements play a role of sensors by publishing their status to the Monitoring System
  - The Monitoring System transports this information to the system "brains":
    - A Human operator
    - An Expert system
  - It also records the monitoring information for a later in depth analysis

The Simplest Monitoring Code Sample

This is a monitoring message

```
print(“Hello, World”)
```

This is the monitoring API function

Is the Hello.py application well designed?

print(“Hello, World”)

- The issue is that the “`print(…)`” function provides both the API and the implementation, which are tightly bound together
- The destination for the messages passed to “`print(…)`” is fixed
- Can we do better?

Logging Module to the rescue

```
import logging

logging.basicConfig(format='%(asctime)s
%(levelname)s [%(filename)s:%(lineno)s -
%(funcName)s()] %(message)s', level=logging.INFO)

logging.info(“Hello, World”)
```

Use the standard well-designed API

The output format can be easily changed

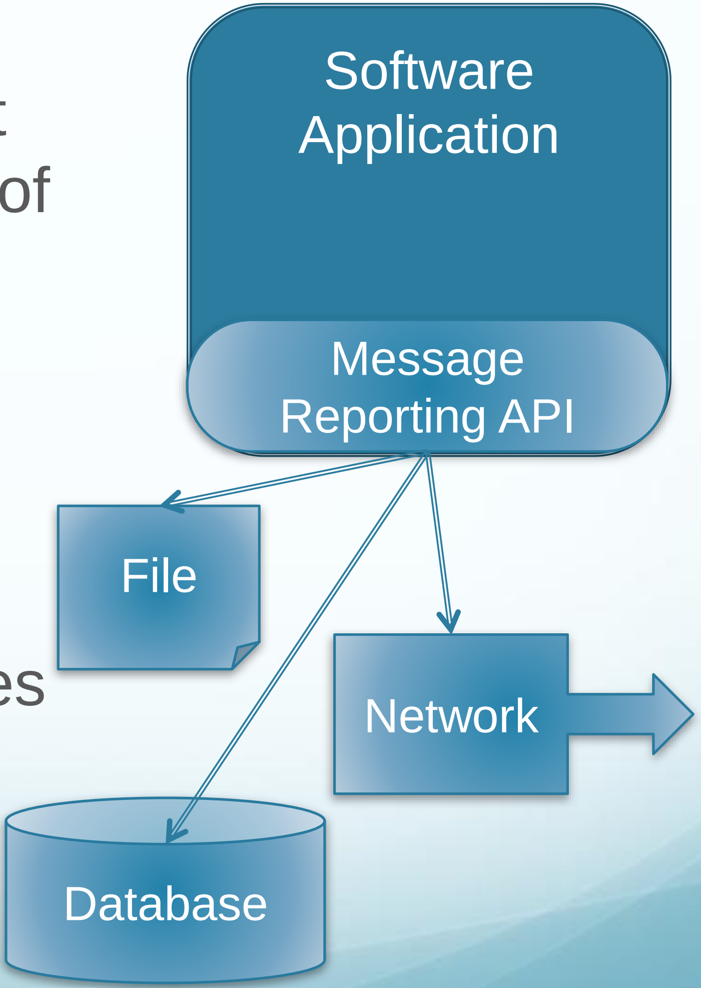
```
2017-02-02 16:21:13,583 INFO [hello.py:4 - <module>()] Hello, World
```

A lot of extra useful information is available for free

The message destination can be changed without touching the application code

The Key Principal: Separate API from Implementation

- A Message Reporting API must not depend on a particular destination of messages
- Multiple destinations can be supported by different implementations of the same API
- Common properties will be automatically added to all messages by the API:
    - Time, Process ID, User ID, etc.



Monitoring in a DAQ system

What shall be monitored in a DAQ System

- System incidents:
  - Expected and especially Unexpected incidents shall be immediately reported in a form of a message
- System status:
  - Health of individual HW & SW components:
    - Resources consumption, IO rates, etc.
  - Surrounding environment:
    - Temperature, humidity, etc.
  - Operational statistics:
    - Number of triggers, number of recorded events, etc.
- Data Quality Monitoring:
  - Recorded physics data shall be constantly monitored for sensibility

DAQ Messaging Service

- Logging API is a good candidate for the DAQ Messaging Service API
- There are several important considerations to be taken into account about messages:
  - Messages shall contain human readable (and understandable!) explanation of the corresponding incidents:
    - A shifter calling you in the middle of a night can just read it
  - Messages shall be ready for machine processing:
    - Every message shall contain a unique ID which corresponds to the given incident
    - Messages shall contain some common automatically generated parameters:
      - Time stamp, application ID, computer name, etc.

Some Remarks on Time Stamps

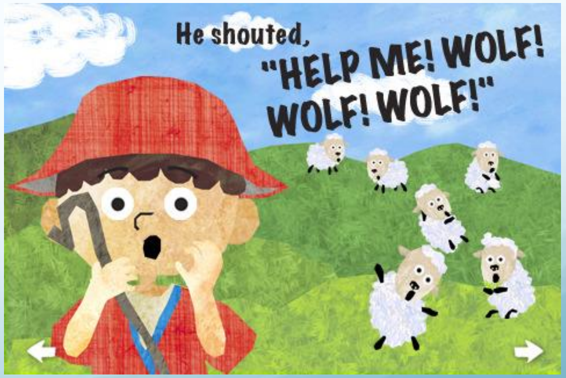
- Time Stamp is a vital property of a message and of any monitoring information in general:
  - Used for correlating messages from different sources
  - Used for correlating monitoring information with real-life events
- The time stamp guidelines:
  - Use the best possible precision (nanoseconds)
  - Use UTC time when reporting
  - Conversion to the human readable local time shall be done by a message displaying applications with respect to its location

Message Types

- CRITICAL – severe error has been detected. It can lead to system damage or affect Data Quality. Immediate attention is required
- ERROR – recoverable error happened. Check the consequences
- WARNING – nothing is bad so far, but system may be operating dangerously close to a certain limit
- INFO – an expected incident occurred
- DEBUG – shall be disabled in production system

CRITICAL Messages & Alarms

- A DAQ system may produce a CRITICAL message when immediate shifter attention is required:
  - Such a message can be used to trigger an Alarm:
    - A dedicated message display application changes the color (and the shape – for color blind people) of an appropriate item
    - It can also make a sound
- Do not overuse alarms:
  - ‘Cry-Wolf’ effect may result in a severe system damage or data corruption



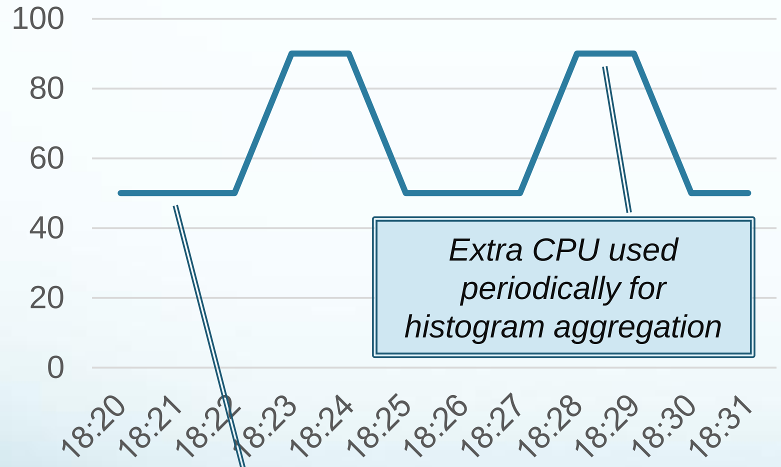
Do not neglect WARNINGs

- Reporting a CRITICAL event is straightforward
- Deciding when to report a WARNING is much more cumbersome:
  - Nothing wrong happened so far, but the system is dangerously close to a certain limit
  - Requires some extra health-checking code to verify operational conditions
- Warnings shall not be neglected as sooner or later they become errors:
  - They can be effectively used for the future error prevention

When to issue a WARNING: A real life example

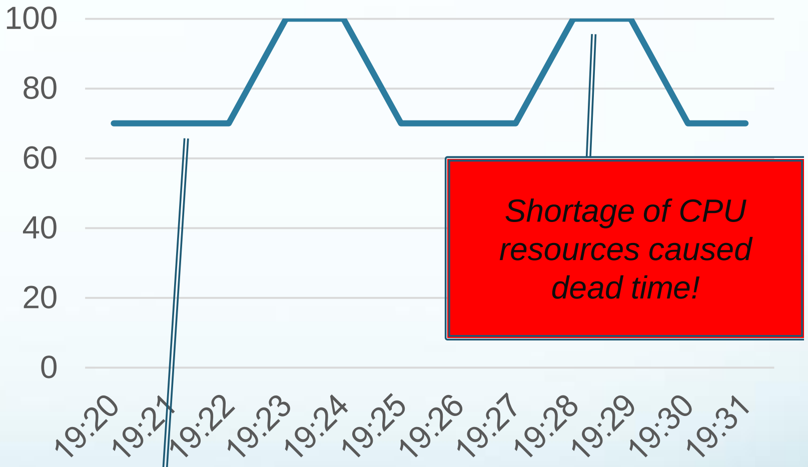
RUN X:
Everything goes well so far

RUN X+1:
CPU shortage causes dead-time


Trigger Farm CPU usage

Extra CPU used periodically for histogram aggregation

CPU used for physics data processing


Trigger Farm CPU usage

Shortage of CPU resources caused dead time!

Adding new Trigger algorithm increased the average CPU usage

DAQ System Status Information

- A DAQ system has many numeric parameters to be monitored:
  - Logging API is not convenient for that
- A simple status information item can be represented in a form of a <key:value> pair:
  - “key” - a unique identity of a particular information
  - “value” - information value for a given moment:
    - This makes it obvious that every pair has to be time-stamped
- A “value” can be a simple number as well as an object :
  - The object represents a state of a DAQ SW or HW element with multiple properties
  - The object has a number of attributes that contain the values of the corresponding properties for a given moment in time

Information Sharing: A Simple Provider API

```
interface InfoRepository {
        void registerInfo(String id, InfoObject
obj);
        void unregisterInfo(String id);
}
```

- A SW Application can register any information object with a unique ID:
  - Afterwards it simply keeps updating the attributes of the objects in a usual way to reflect the status of the corresponding SW or HW component
- The Interface intentionally does no expose any details of how information will be handled:
  - This will be defined by an implementation

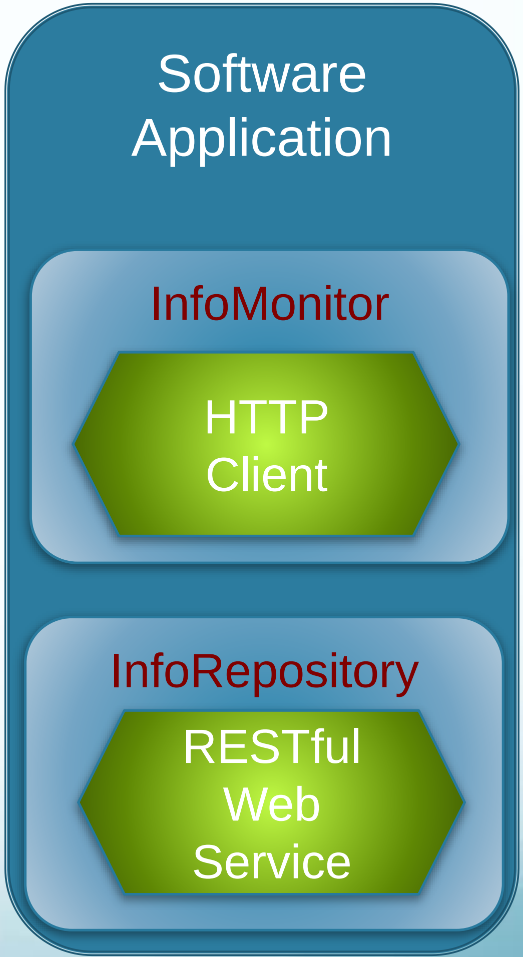
Information Sharing: A Simple Consumer API

```
interface InfoMonitor {
        Enumeration<String> getInfoList();
        InfoObject getInfo(String info);
}
```

- A SW Application can use the ***InfoMonitor*** Interface:
  - To get a list of available monitoring information
  - To get a value of an information object
- Implementation of this interface is bound to the one of the ***InfoRepository***

Information Sharing: A Possible Implementation

- REST – **Re**presentational **S**tate **T**ransfer
  - Client-server HTTP-based stateless communication
  - Fits very well to the Status Monitoring communication model
- The Web Service implements ***InfoRepository*** Interface
- The HTTP Client implements ***InfoMonitor*** Interface
- BONUS:
  - A standard Web Browser can be used to display monitoring information


Software Application

InfoMonitor

HTTP Client

InfoRepository

RESTful Web Service

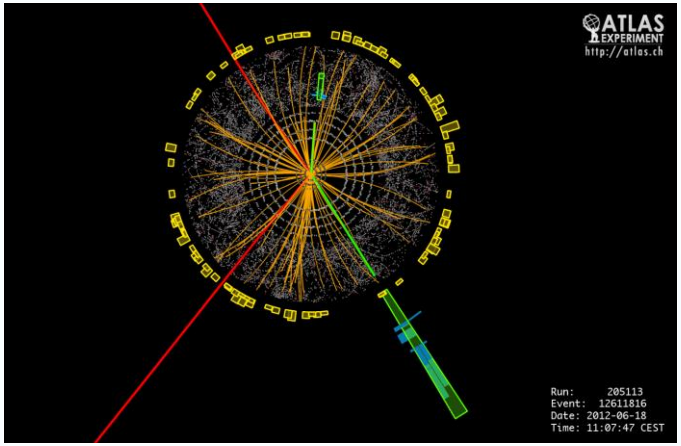
A DAQ specialty: Data Quality Monitoring

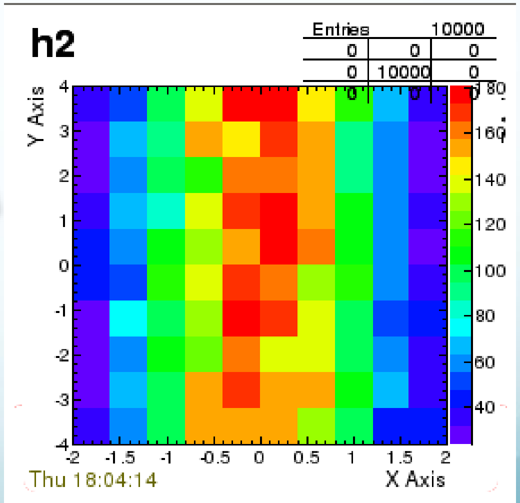
Data Quality Monitoring

- Watching out the behavior of the DAQ system itself is not sufficient:
  - The DAQ may be functioning perfectly well but at the same time is taking meaningless data, for example, due to wrong calibration constants
- A dedicated service is required for checking that:
  - Detector readout provides meaningful data
  - Trigger does reasonable selection

Data Quality Monitoring: Input Data

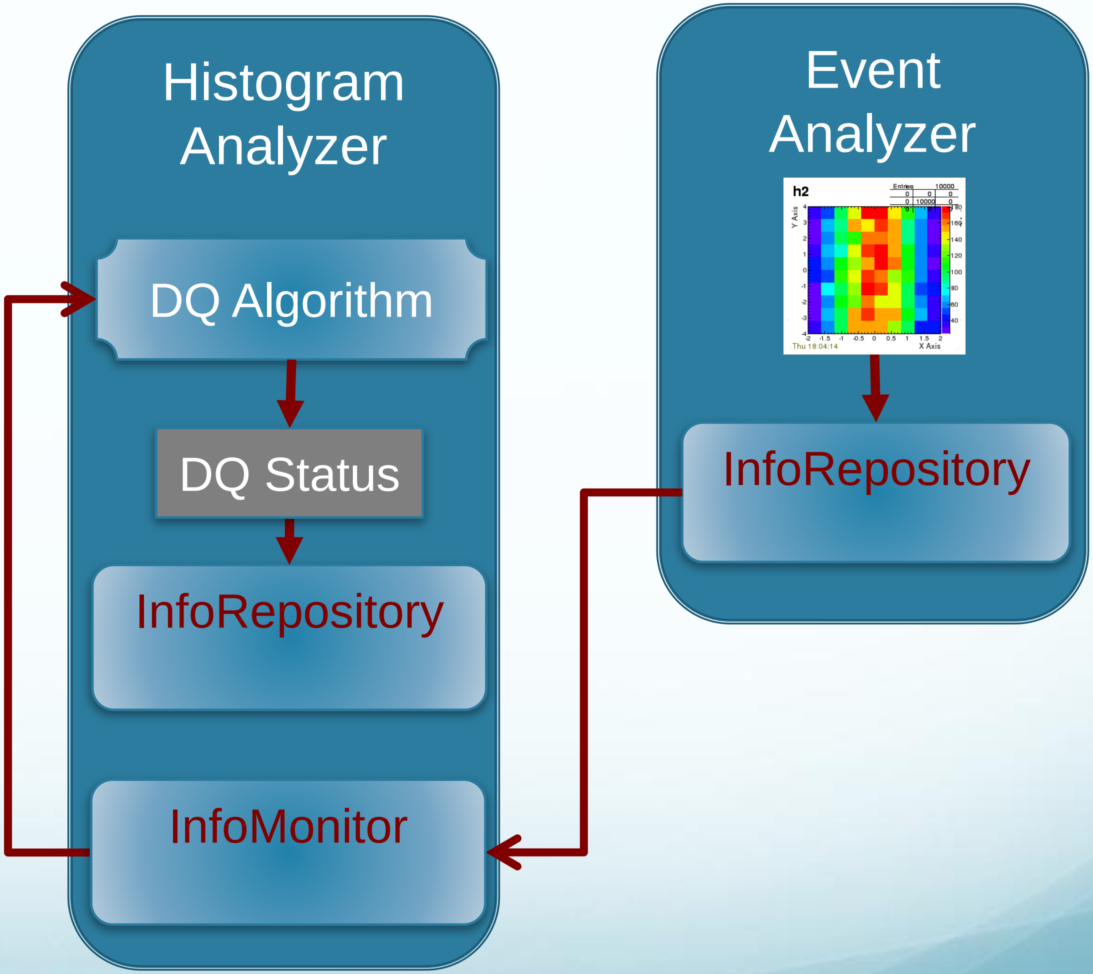
- There are two main types of information that can be used for DQ Monitoring:
  - A picture of a reconstructed physics event
  - Histograms that accumulate data from a statistical sample of raw physics events
- Only eyes of a true Experts can make sense of such data





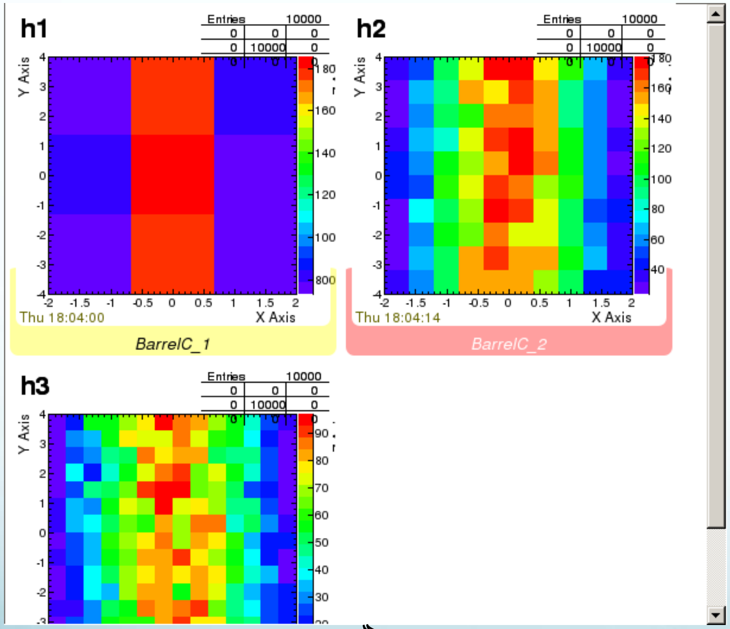
Data Quality Monitoring Automated: Step 1

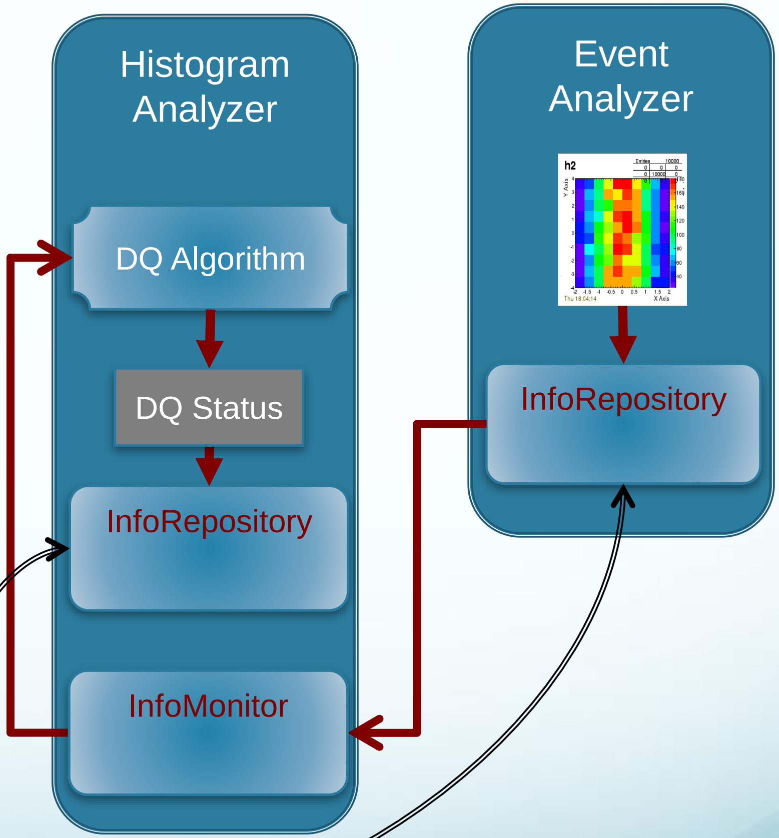
- ***InfoRepository*** interface can be used to register histograms:
  - Histogram is just an object with a number of specific attributes
- Histogram Analyzer registers a DQ Status object per histogram:
  - Periodically reads the corresponding histogram and executes DQ Algorithm that updates this DQ Status



Data Quality Monitoring Automated: Step 2

- A DQ Display application can read and display both the histograms and the DQ Statuses using the ***InfoMonitor*** Interface





Histogram Analyzer

DQ Algorithm

DQ Status

InfoRepository

InfoMonitor

Event Analyzer

InfoRepository

Scaling up the Monitoring System

The HEP Experimental Realm

- A DAQ system of a modern HEP experiment includes:
    - O(1K) computers and network devices
    - O(10K) SW applications
    - O(100K) HW sensors

- If Monitoring APIs were properly designed they will not be affected by the scalability requirements
- However the implementations of the Monitoring APIs may need to be adjusted

Managing Monitoring Information in a distributed system

- There are two main parameters which can be tuned for addressing scalability requirements:
  - The information access model:
    - Pull data vs Subscribe/Callback
  - The communication model:
    - Peer-to-Peer vs Client-Server
- The choice of a particular model will limit a set of technologies which can be used for the implementations

Information Access Models

- Pull Model (synchronous communication):
  - An Information Consumer requests information whenever it is required
  - RESTful Web Service is a good example of a technology which can be used to implement this access model
- Subscribe/Callback model (asynchronous communication):
  - An Information Consumer subscribes for the relevant subset of information to get notified when the information is updated
  - RESTful architectur can not be used because a subscription changes the state of the Information Provider
  - One can consider using one of the existing IPC systems:
    - Low-level Libraries: ZeroMQ, Boost ASIO, etc.
    - High-level communication brokers: Ice from ZeroC, CORBA: TAO, omniORB

Communication Models

- Peer-to-Peer:
  - Information Providers announce available information
  - Information Consumers connect to the Providers to read information or subscribe for updates
- Client-Server:
  - Information Providers send their information to the server periodically or on demand
  - Information Consumers connect to the server to read information or subscribe for updates

Which one is the best?

Peer-to-Peer

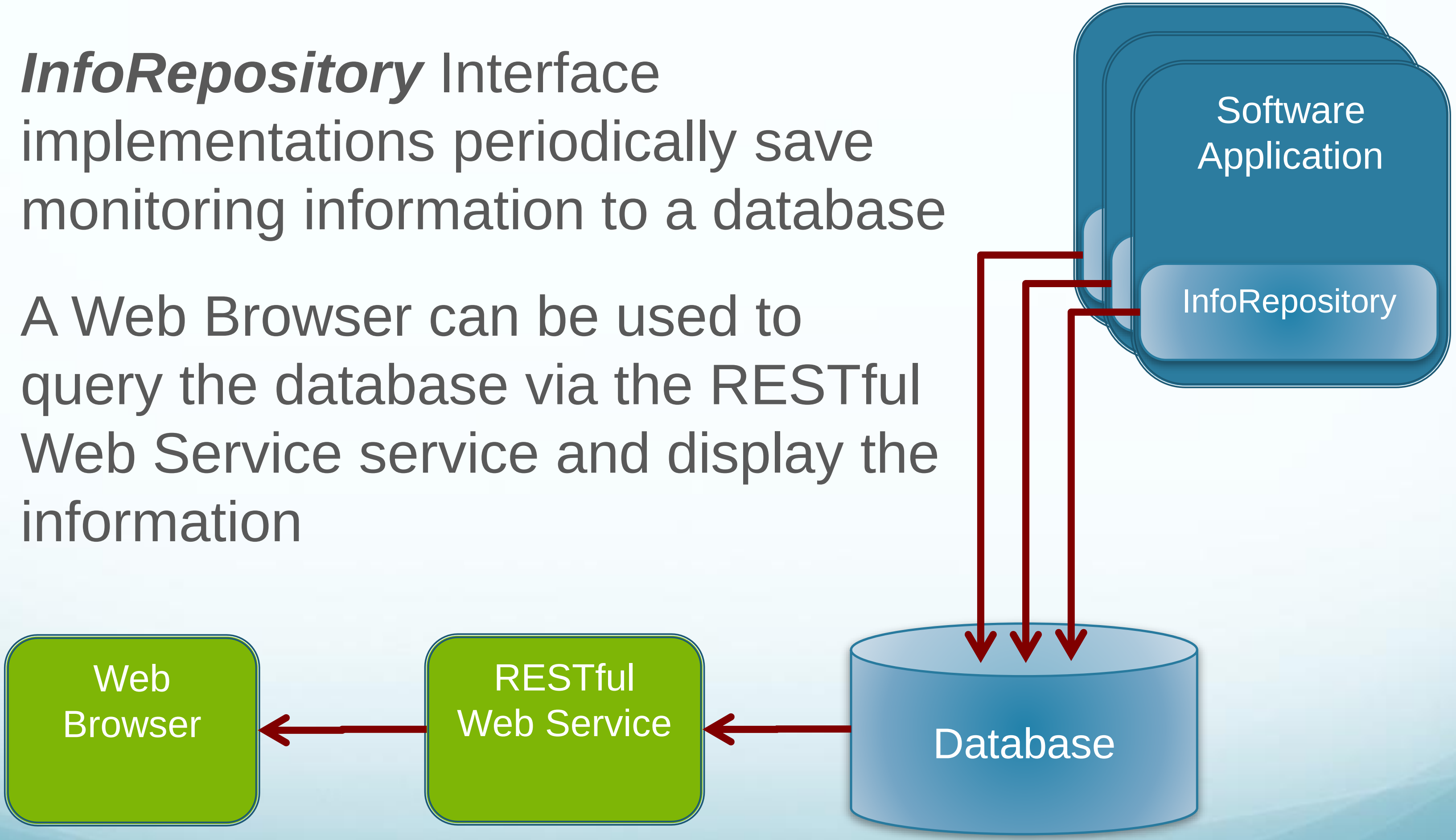
- Has no single point of failure
- Scales better
- Uses more connections
- Providers are exposed to Consumers due to the direct connections
- More difficult to implement and maintain

Client-Server

- Separates Providers from Consumers
- Simplifies information access
- Implementation and maintenance is simpler
- Has a single point of failure
- Scalability requires multiple servers and additional HW resources

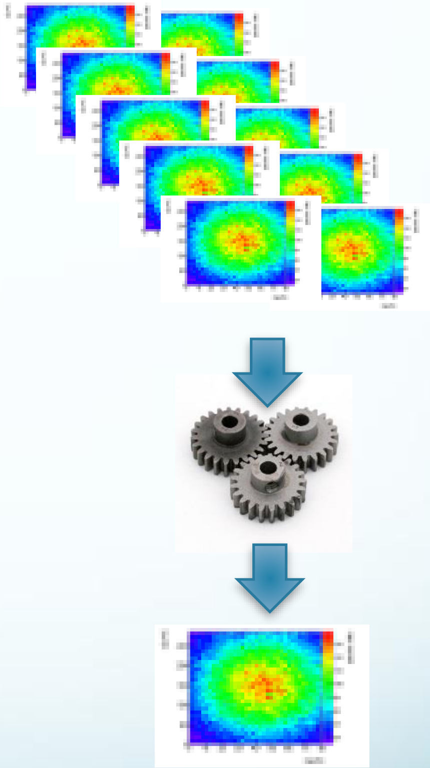
A Client-Server Implementation Example

- ***InfoRepository*** Interface implementations periodically save monitoring information to a database
- A Web Browser can be used to query the database via the RESTful Web Service service and display the information



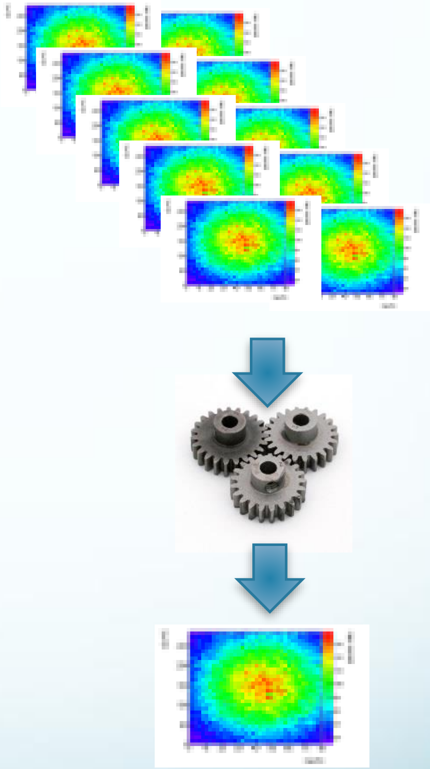
Monitoring Information Aggregation

- Monitoring information produced by individual DAQ Applications may need to be collected to provide high-level system status, e.g.:
  - Aggregating counters produced by individual computers of the Trigger Computer Farm one can access a complete state of the Trigger system
  - Aggregating histograms from all Trigger applications will give accumulated statistics for all processed events



Information Aggregation Implementation Options

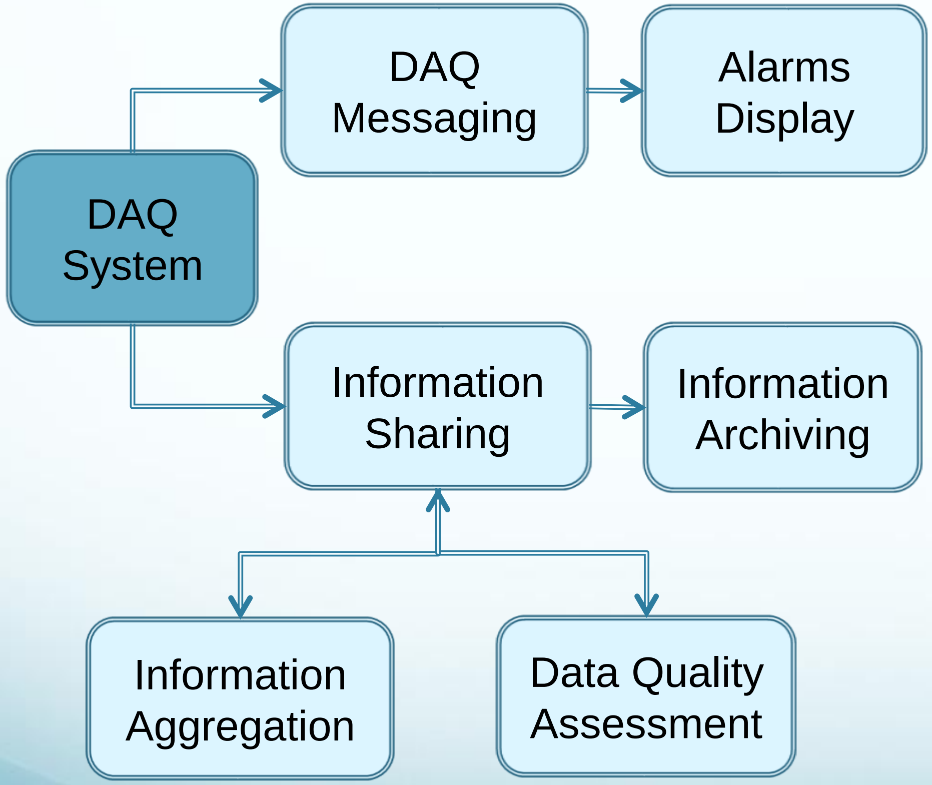
- The way of aggregating information may depend on the data access and communication models used for the Monitoring system implementation
- Per-to-Peer communication model is used:
  - Information Aggregator will retrieve data from all Providers
- Client-Server communication model is used:
  - Information Aggregator will retrieve data from the central server
- Subscribe/Callback mechanism is available:
  - Information Aggregator will subscribe for the information changes and do the aggregation only when information is updated



Archiving Monitoring Information

- Ideally all monitoring information shall be archived to a permanent storage:
  - Do post mortem analysis
    - Special attention should be paid to WARNINGs
  - Investigating problems
- Conceptually Archiver is just a special type of the Information Consumer
- In practice the task is non-trivial due to the huge amount of information
- It might be feasible to have multiple Archivers for different information types:
  - Histograms, errors, operational status

Bringing These All Together

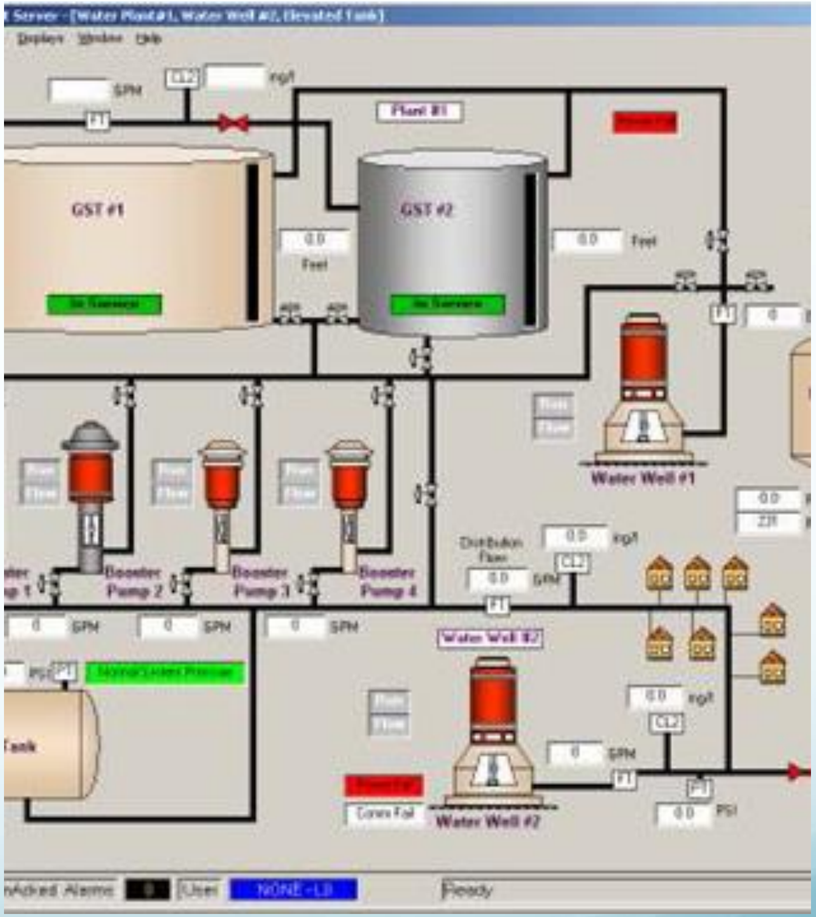


- A Monitoring system should consist of a number of independent services:
    - More convenient for handling different types of monitoring information:
        - E.g. messaging system vs status monitoring
    - More simple to address scalability requirements
    - Easier for development and maintenance

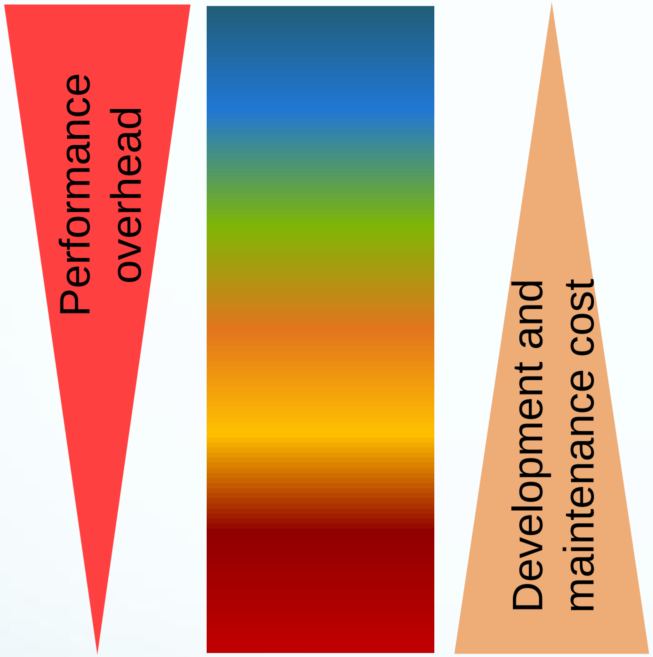
Choosing Technologies for A Monitoring System Implementation

Commercial Solutions: SCADA systems

- Supervisory Control and Data Acquisition
  - It is primarily dedicated for control but does the monitoring as well
  - Modern implementations scale well for small and mid-size systems
- A SCADA system can be implemented using LabView:
  - Graphical programming language for the system design
  - Powerful and configurable graphical interface
- Is used mostly for HW control and monitoring
  - May not fit well to the DAQ specific monitoring, i.e. Data Quality



The spectrum of the IPC technologies



- Ice from ZeroC
- CORBA: TAO, omniORB, …
- Messaging systems: Qpid, ActiveMQ, RabbitMQ, ...
- RESTful Web Services
- Libraries: Boost ASIO, ZeroMQ, ACE, …
- Socket API, TCP, RDMA, …

- In realty situation is slightly more complicated:
  - Maintenance cost and performance overhead very often depend on the implementation quality
- A choice would depend on a particular system requirements:
  - System size, Programming languages, available resources, implementation time scale, etc.

Data format for network transfer

- For HTTP communication Json is the natural format:
  - `{ “CPU”: 90, “Memory”: 4.3, … }`
- Advantages of Json:
  - Simple, Human readable, self-contained
- However performance is the weak point:
  - Parsing Json is CPU-hungry
  - Transferring attributes names adds noticeable overhead to the amount of data passed over network
- Compact protocol buffer format can be considered as an alternative:
  - E.g. google/protobuf, binary Json, etc.

Data Archiving Technologies

- A choice strongly depends on the requirements of a particular experiment
- Large HEP experiments generate O(1)TB of monitoring information per year
- Traditional (SQL) databases are not good for storing and indexing such amount of data
- Big Data approach is the new trend in this area
    - Hadoop, Teradata, Cassandra and many others

Visualization Technologies

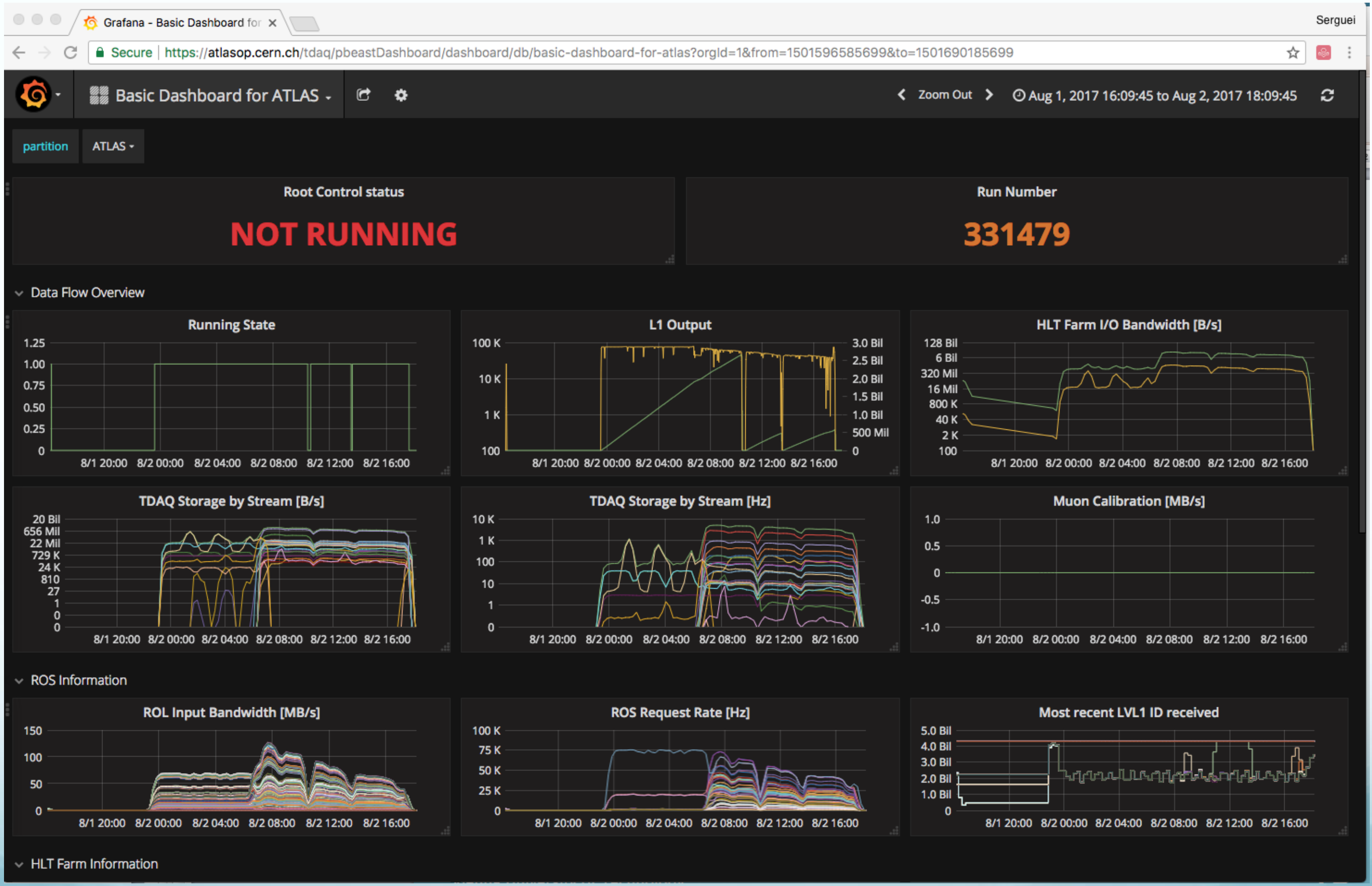
GUI Frameworks/Libraries

- Normally is bound to some specific programming languages:
  - Qt – C++, Python
  - Swing – Java
- Has to be installed in a binary form:
  - Platforms compatibility
  - 3rd party libraries
- Very good performance

WEB Browsers

- Visualization is easily customizable (javascript, CSS, etc.)
- A lot of out-of-the-box graphical visualization libraries are available
- No additional software required on a client computer
- Available all over the globe
- Performance may not be adequate for real-time systems

Web-based ATLAS Online Monitoring Customizable Dashboard implemented using Grafana



Grafana - Basic Dashboard for
Serguei
Secure
https://atlasop.cern.ch/tdaq/pbeastDashboard/dashboard/db/basic-dashboard-for-atlas?orgId=1&from=1501596585699&to=1501690185699
Basic Dashboard for ATLAS
Zoom Out
Aug 1, 2017 16:09:45 to Aug 2, 2017 18:09:45
partition
ATLAS

Root Control status

NOT RUNNING

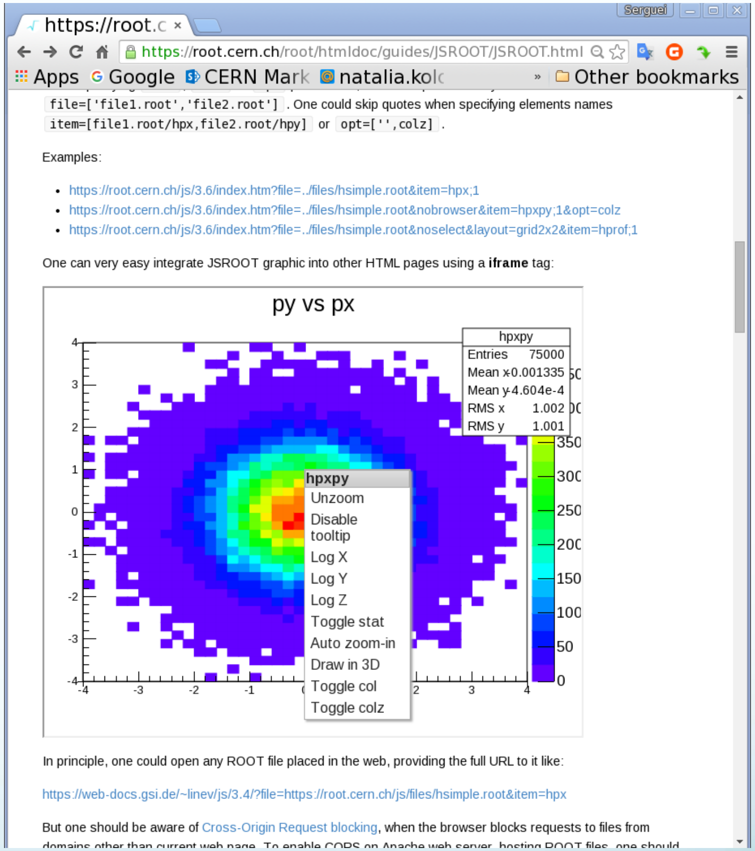
Run Number

331479

Web Browser Visualization Technologies: Physics Special

- Physicist needs histograms, which limits a range of suitable visualization technologies

- Starting from Version 6 ROOT contains JavaScript library for histograms visualization in Web browsers:
  - **JSROOT**
  - https://root.cern.ch/root/htmldoc/guides/JSROOT/JSROOT.html



`file=['file1.root','file2.root']` . One could skip quotes when specifying elements names `item=[file1.root/hpx,file2.root/hpy]` or `opt=['',colz]` .

Examples:

- https://root.cern.ch/js/3.6/index.htm?file=../files/hsimple.root&item=hpx;1
- https://root.cern.ch/js/3.6/index.htm?file=../files/hsimple.root&nobrowser&item=hpxpy;1&opt=colz
- https://root.cern.ch/js/3.6/index.htm?file=../files/hsimple.root&noselect&layout=grid2x2&item=hprof;1

One can very easy integrate JSROOT graphic into other HTML pages using a **iframe** tag:

In principle, one could open any ROOT file placed in the web, providing the full URL to it like:

https://web-docs.gsi.de/~linev/js/3.4/?file=https://root.cern.ch/js/files/hsimple.root&item=hpx

But one should be aware of Cross-Origin Request blocking, when the browser blocks requests to files from

Summary & Final Remarks

The Key Points to Keep in Mind

- A well-designed and complete Online Monitoring API is a pre-requisite for implementing a DAQ system:
  - Use standard Monitoring APIs as much as possible:
    - e.g. Logging API
  - Think carefully when designing a custom API:
    - It shall not depend on a communication technology
    - It shall not change when a data access or communication model would need to be changed
- It is acceptable that final implementations of Monitoring APIs are not available at the very beginning:
  - Any way they may need to be changed in the course of the DAQ system development

Monitoring vs DAQ

- Monitoring is often considered as an ad hoc system, which can be developed at leisure when a system to be monitored (e.g. DAQ) is ready:
  - Big mistake!
- A properly designed monitoring system will help to keep things under control even during the DAQ system development:
  - Reduces time for testing and debugging of the DAQ system
  - Feedback will help fixing issues in the Monitoring system itself

"It looks like you have everything under control."