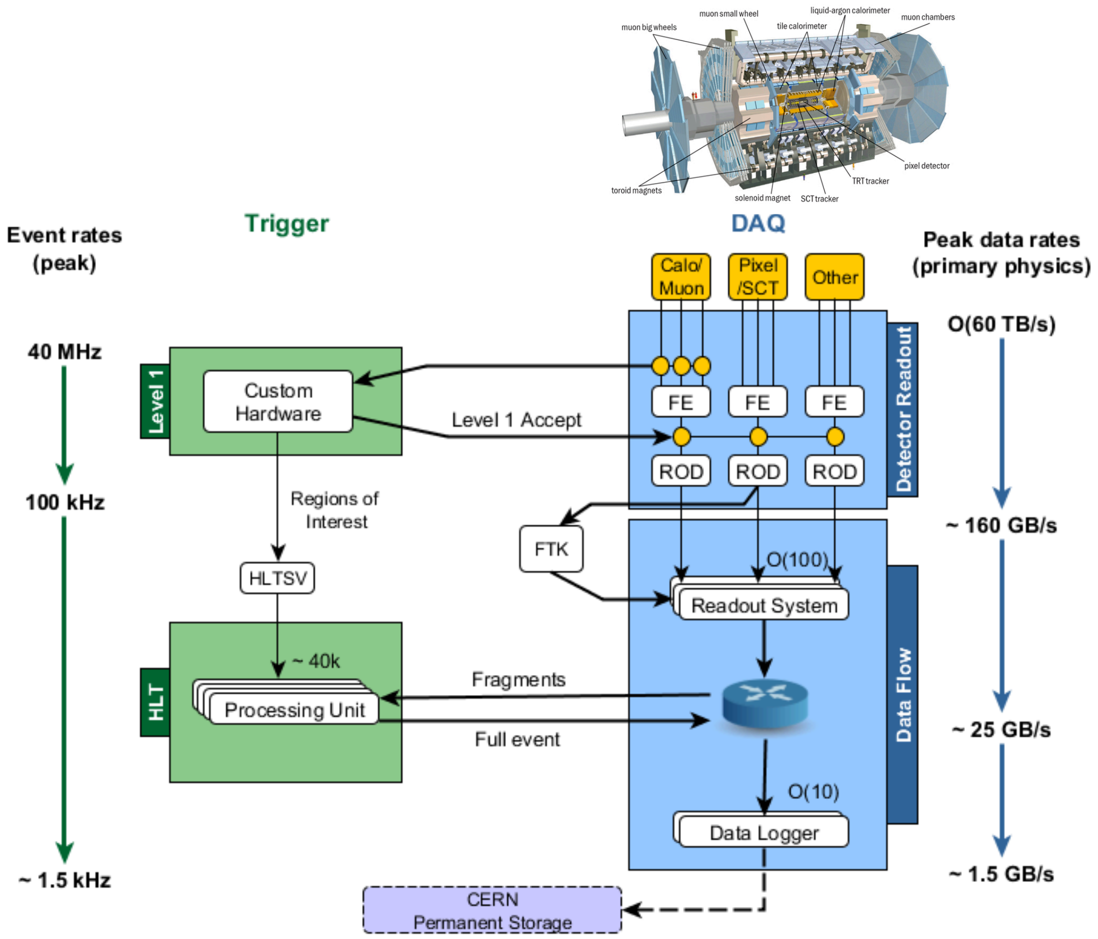
Highly extensible modular system for online monitoring of the ATLAS experiment.

Serguei Kolos (University of California Irvine)
on behalf of the ATLAS TDAQ Collaboration

Outline

- The ATLAS experiment at a glance
- The Online Monitoring challenges
- The Online Monitoring system architecture and evolution
- Conclusion

The ATLAS Experiment at a glance



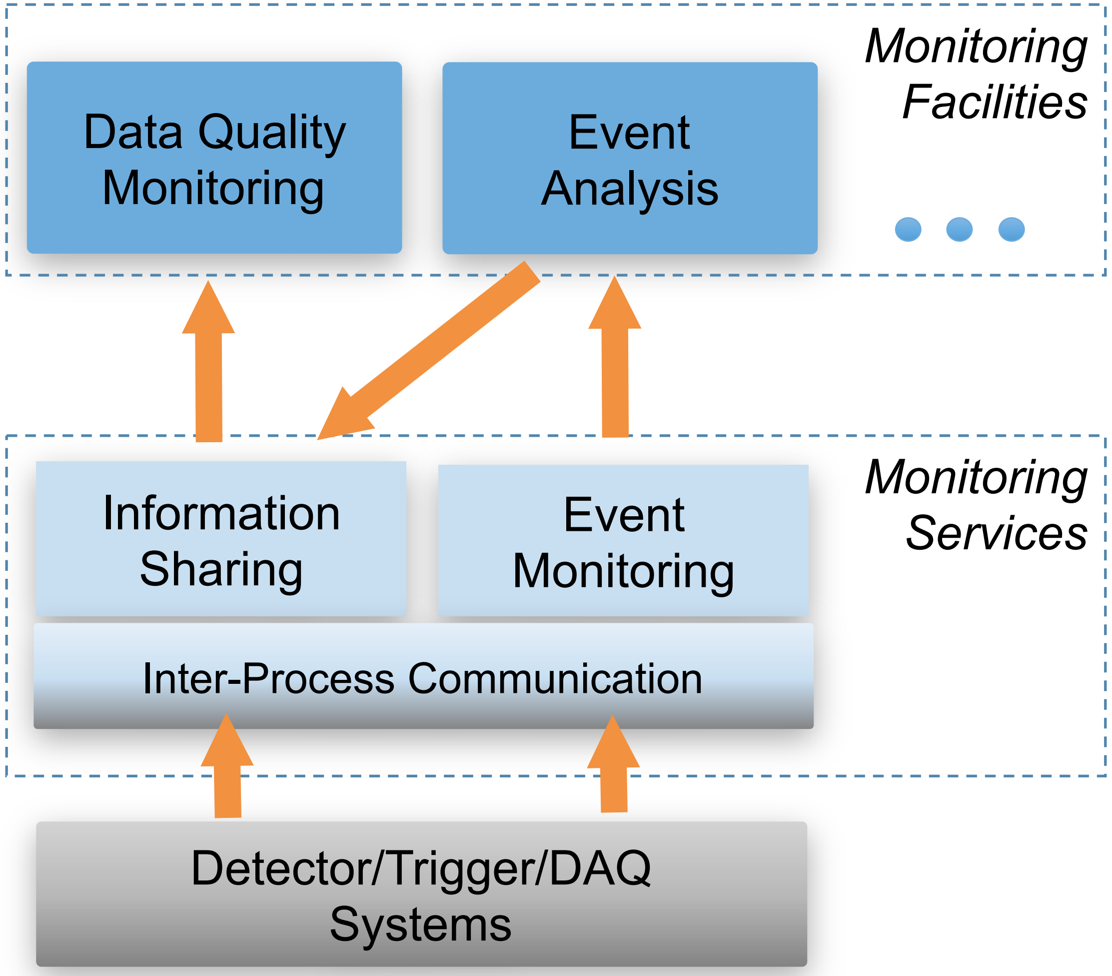
- ATLAS is one of the 4 big detectors at LHC, devoted to testing the predictions of the SM and searching for new physics:
  - 140M channels
  - 40 MHz output rate
- The ATLAS Trigger/DAQ system is composed of about 60'000 software processes distributed over more than 2'500 computers
- Online Monitoring is applied to:
  - Detector:
    - Dead-time, busy channels, etc.
  - Trigger system:
    - Consistency of trigger information, operational statistics
  - DAQ System:
    - Operational statistics, states of HW & SW components

The Challenges for the Online Monitoring System

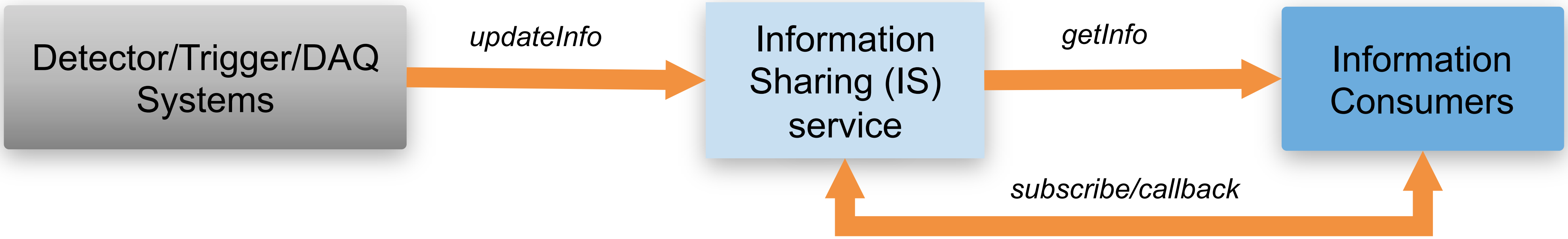
- Size & Scalability:
  - Very large number of parameters to be monitored
  - It gets even bigger after upgrades
- The ATLAS experiment construction has begun more than 20 years ago:
  - The system to be monitored had unprecedented complexity
  - A spectrum of computing technologies was very limited
  - Defining requirements to the Online Monitoring system has been an iterative process
- The lifetime of the experiment spans over several decades:
  - The Monitoring system shall be easily adaptable to technologies landscape evolution
  - The Monitoring system shall be capable to benefit from the HW evolution in this period

The Monitoring System Architecture: Services and Facilities

- Monitoring services provide "monitoring front-end" for Detector & TDAQ
  - **Information Sharing** – general purpose information sharing in distributed SW environment
  - **Event Monitoring** – remote data access optimized for raw data sampling
  - IPC is based on CORBA standard:
    - omniORB for C++ and JacORB for Java
- Monitoring Facilities implement high level tasks:
  - Entirely based on the Services
  - Independent from one another
  - Developed gradually as requirements evolved

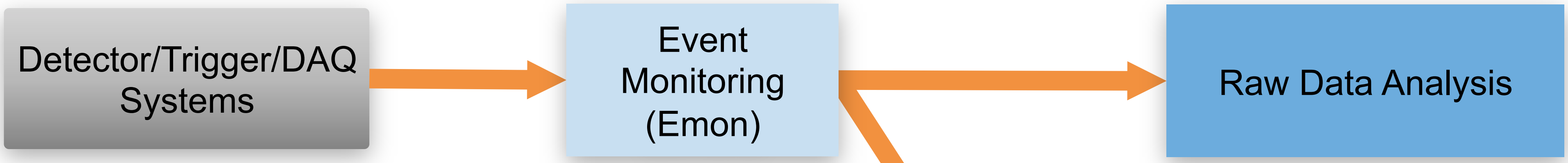


Operational Monitoring: Information Sharing Service

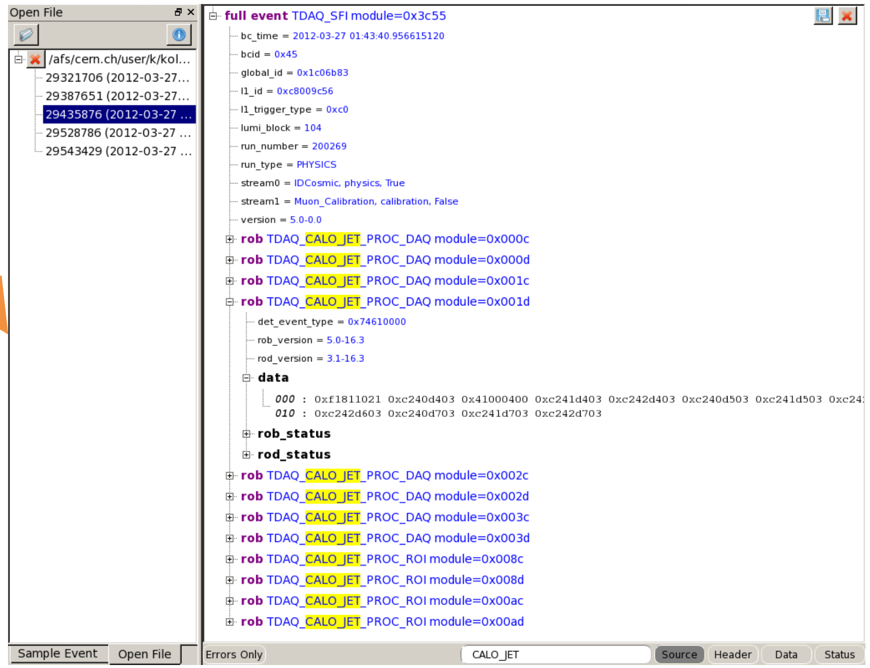


- Information Sharing (IS) is a general-purpose highly scalable distributed service based on CORBA standard
- Supports object-oriented model for information definition:
  - An information item is an instance of a class with arbitrary number of attributes
  - Any number of custom classes can be freely defined
- Is based on the client-server architecture decoupling producers and consumers of information
- Supports direct read as well as subscribe/callback information access pattern

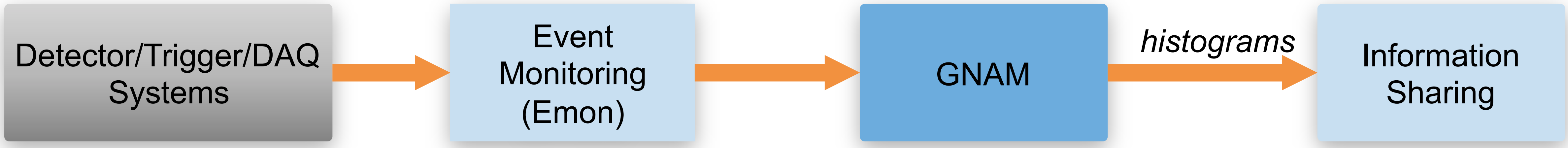
Online Monitoring: Raw Event Sampling



- Event Monitoring (Emon) service provides unified interface to get samples of raw events/fragments:
  - Off-loads event analysis to dedicated computers
  - Minimizes impact to the main data taking activity
  - Decouples event providers and consumers
- Event Viewer is a simple Emon client displaying a structure of a raw event/fragment:
  - Can display events from any sub-system that implements the Event Sampling Emon interface



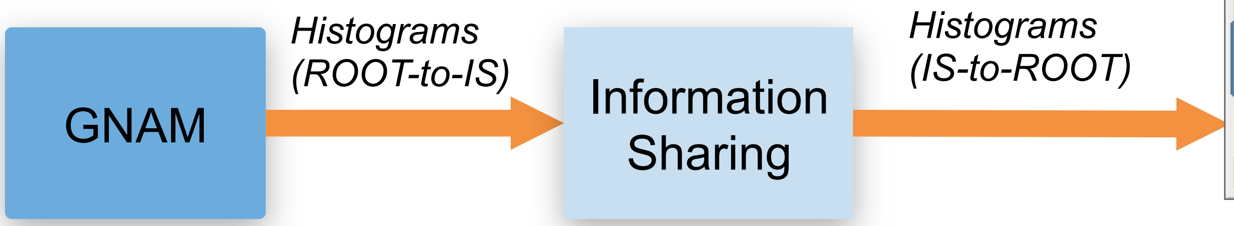
Online Row Data Analysis: GNAM Framework

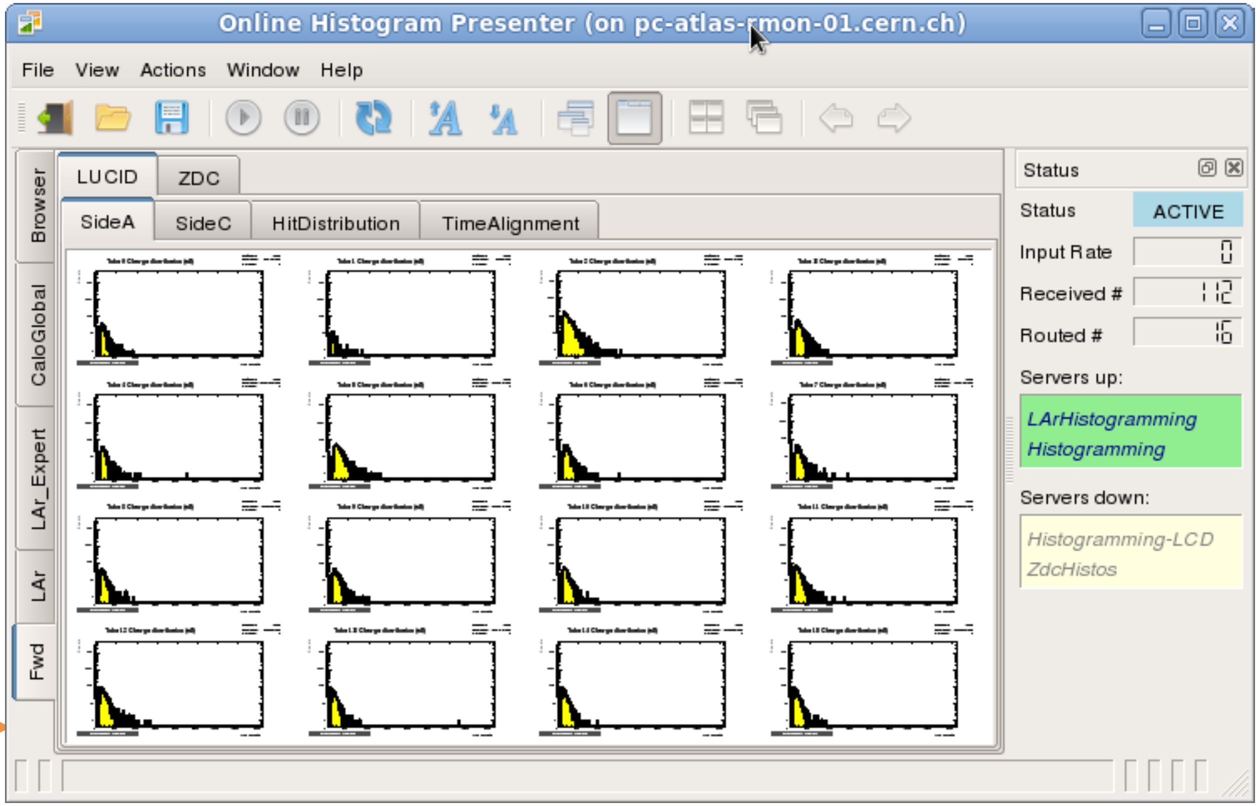


- GNAM is a light-weight framework for low-level raw data analysis:
  - Receives raw events via Emon interface
  - Passes the events to the custom plugin
    - The plugin analyses events and fills a custom set of histograms
  - Publishes these histograms to the IS service
- IS service has been re used for handling the second order monitoring information produced by the Monitoring Facilities

Adding Histograms Support to IS Service

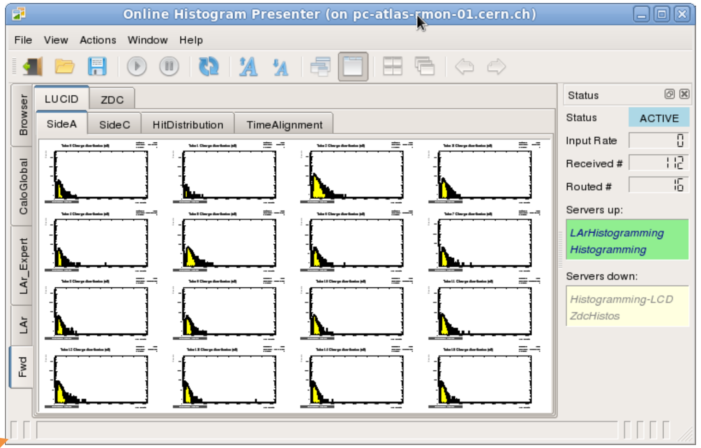
- Adding histograms support to IS was as simple as:
  - Defining a set of custom classes:
    - Histogram1D, Histogram2D, Histogram3D, Profile and Graph
  - Implementing conversion functions from ROOT to IS and vice versa
- The Online Histogram Presenter (OHP) is just another client of the IS:
  - It subscribes to a configurable set of histograms and displays them as they are updated

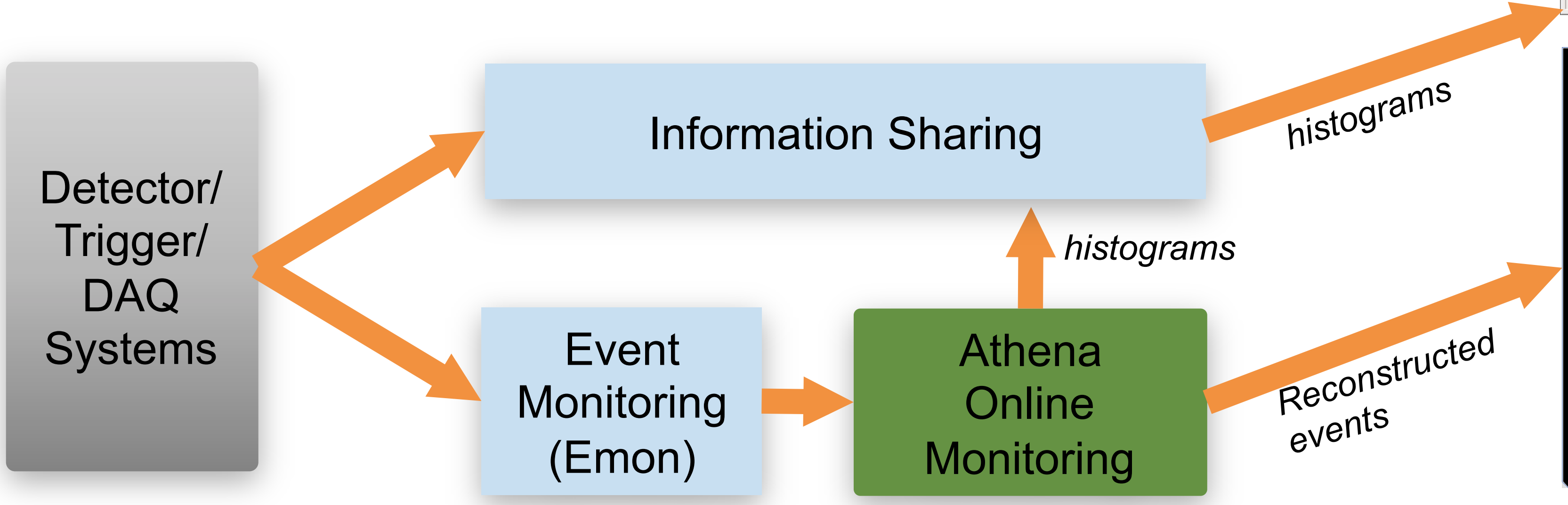


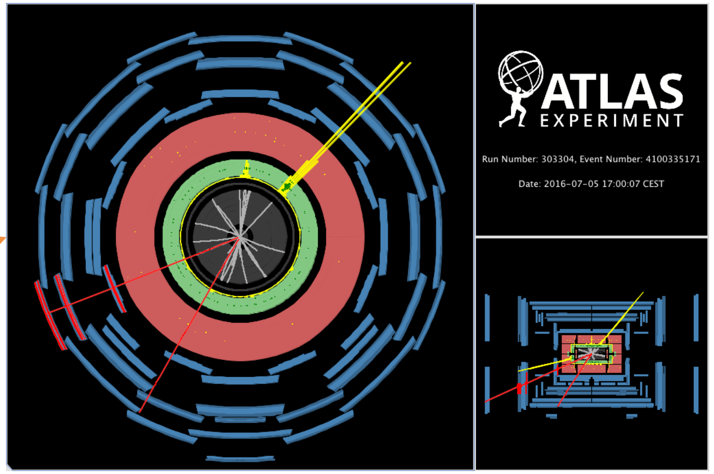


Using Offline Algorithms for Online Monitoring

- **Athena** is an implementation of Gaudi framework used by ATLAS for both HLT and Offline data processing
- Using Monitoring Services **Athena** has been integrated with online monitoring:
  - A dedicated ***ByteStreamInput*** interface implementation reads events via Emon and publishes histograms to IS
- That makes the **Atlantis** event display available online for free

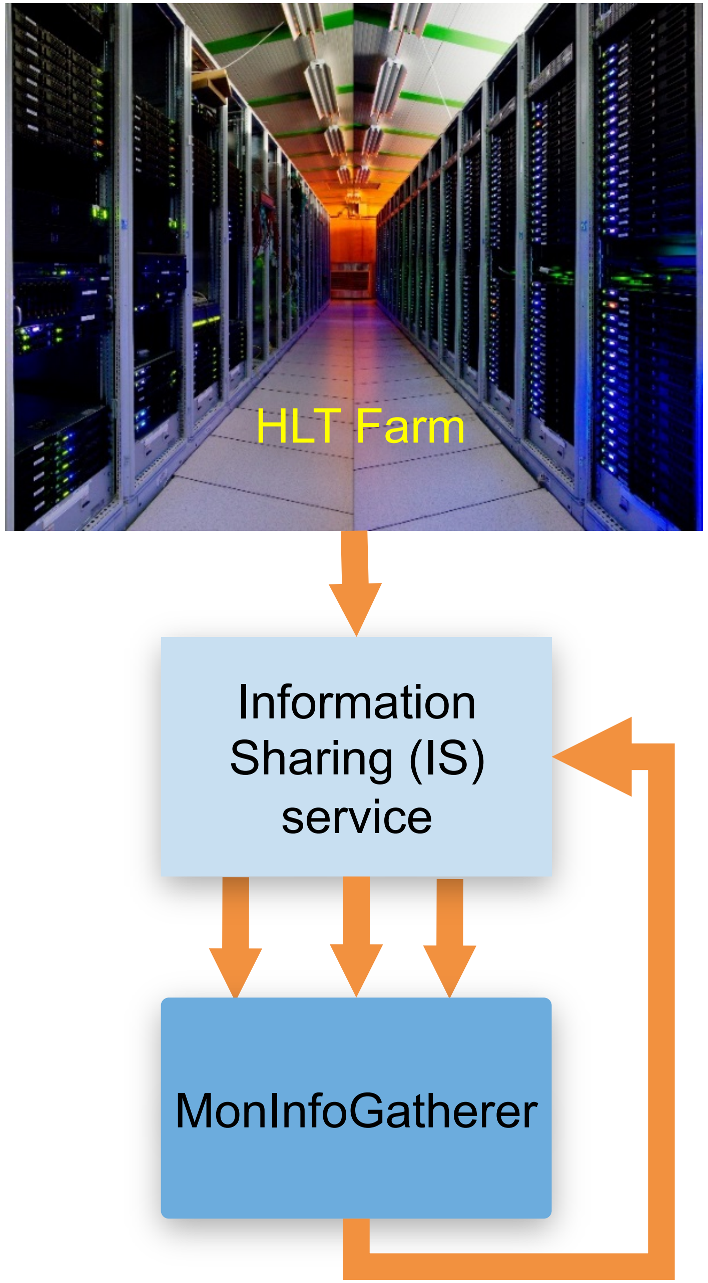






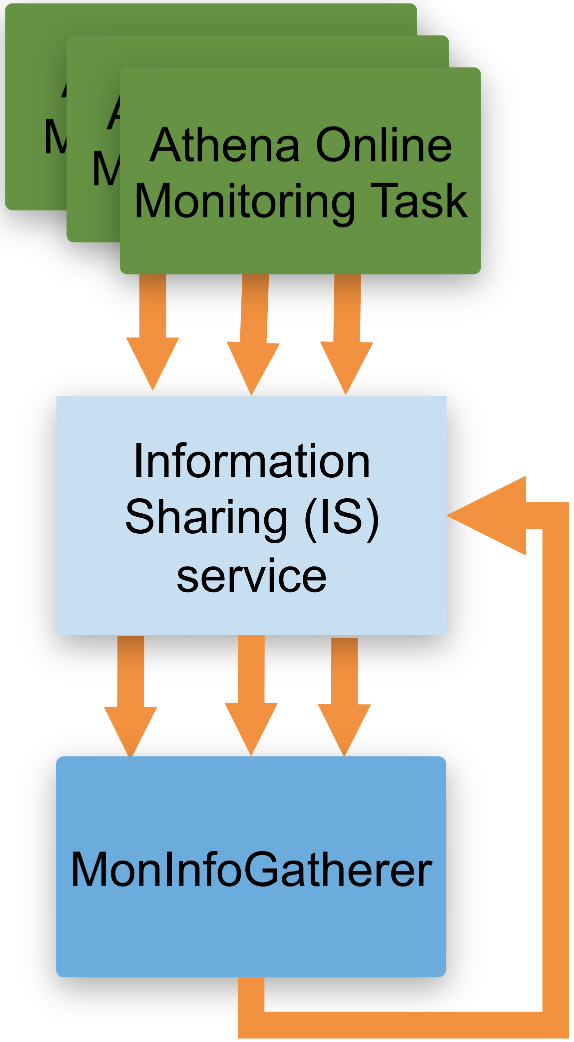
Merging Monitoring Information

- High Level Trigger (HLT) is a significant source of monitoring information:
  - Consists of about 50K processes
  - Each process produces few thousands histograms
- **MonInfoGatherer** – a Monitoring Facility for gathering arbitrary IS information including histograms
- For scalability gathering is done in multiple levels:
  1. Histograms from all processes running on the same computer are merged first (*typically 24 processes per host)*
  2. Histograms from all computers belonging to the same rack are merged (*typically 30 hosts per rack)*
  3. Histograms from all racks in the Farm are merged (*typically about 40)*
- For Run II **MonInfoGatherer** merges ~50M histograms every minute



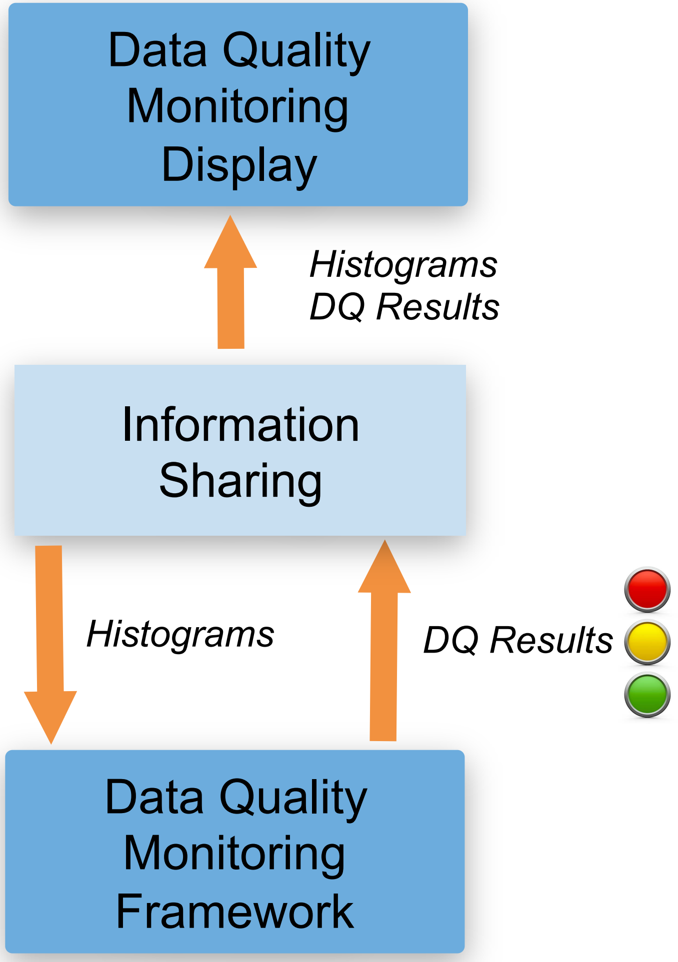
Speed up Statistics Acquisition

- A single event processing may take up to several seconds
- Some histograms must have O(1)K entries for reliable assessment
- An obvious solution is to run multiple event processing tasks and merge histograms they produce
  - For Run II there are ~100 monitoring jobs producing ~50K histograms each
- MonInfoGatherer has been re used for merging these histograms:
  - Reduced statistics acquiring time by a factor of 100



DQMF: Second Order Monitoring Information Provider

- Online Monitoring produces thousands of histograms:
  - Automated histogram analysis is the only feasible approach
- Data Quality Monitoring Framework (DQMF) is a Monitoring Facility that:
  - Subscribes for a pre-defined set of histograms
  - Receives histograms from IS when they are updated
  - Executes custom algorithms to analyze the histograms
  - Publishes results back to IS
- DQM Algorithms are shared by both the Offline and the Online DQ Monitoring due to the common SW infrastructure
- For Run II DQMF checks ~70K histograms every minute



Data Quality Monitoring Display

Information Sharing

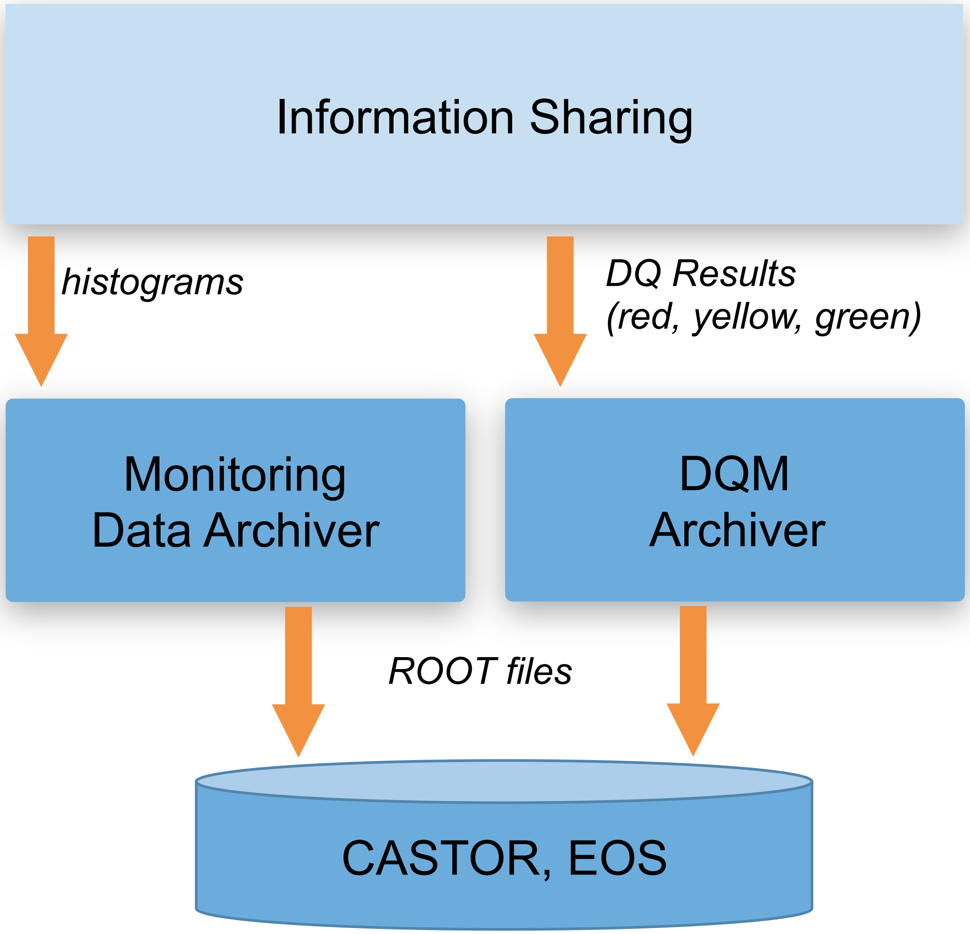
Histograms, DQ Results

- DQM Display is a standalone GUI
- Subscribes to histograms and DQ results and displays them when they are updated

Monitoring Data Archiving

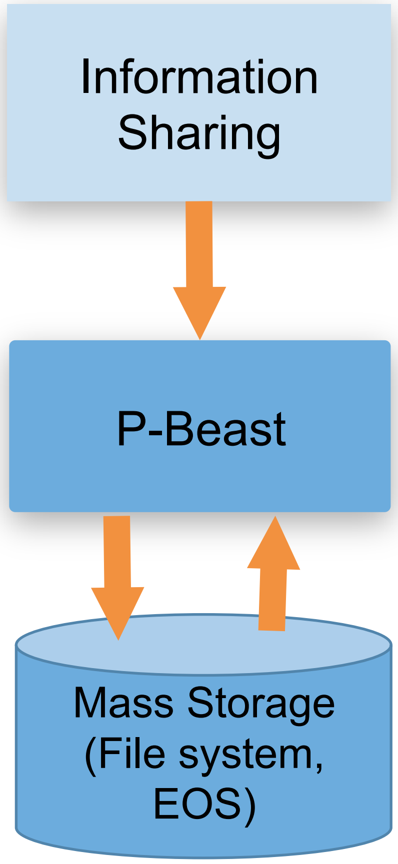
Archiving Monitoring Data

- Motivation for the monitoring data archiving:
  - Monitoring information from the past runs can be used for debugging unforeseen problems
  - Histograms can be used as references for the future runs
- Two independent facilities have been implemented for that:
  - Monitoring Data Archiver (MDA)
    - Saves pre-configured set of histograms periodically during a run as well as at the end of the run
  - Data Quality Monitoring Archiver:
    - Saves all DQM results produced for a given run



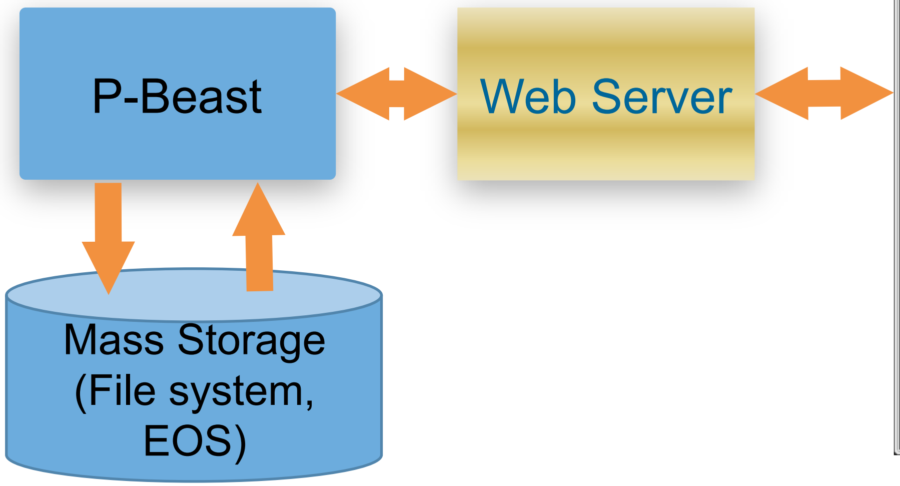
P-Beast: Save ‘Em All

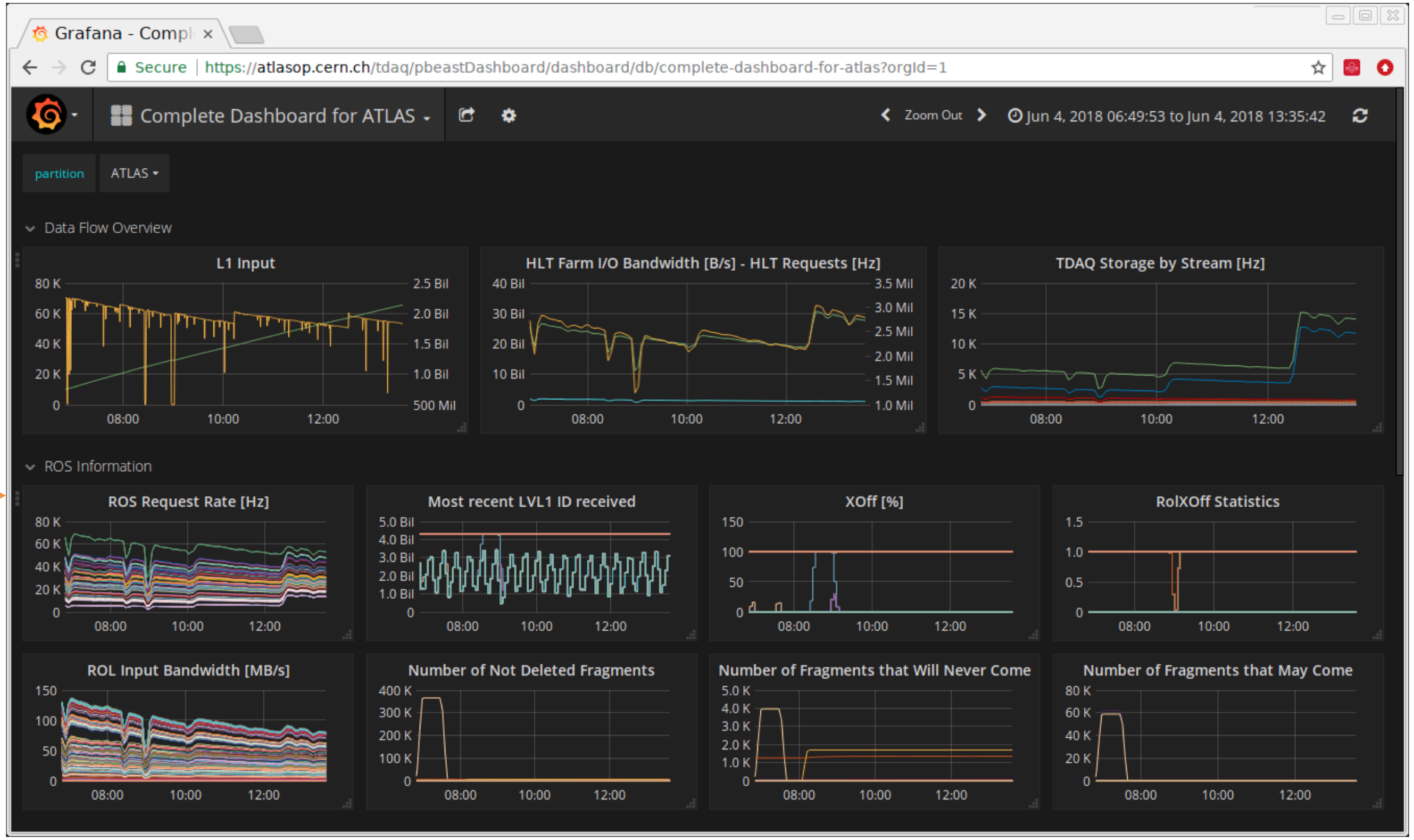
- Saving all monitoring data produced by the Online Monitoring system was a desire for many years:
  - It has finally became possible as computing storage and CPUs got cheaper and more powerful
- P-Beast is a recently developed Monitoring Facility:
  - Saves all online monitoring information except histograms
  - Uses in-house implementation of column-oriented data store:
    - Does data compression and eliminates duplicates on-the-fly
    - Saves data in file system cache, then transfers them to EOS
  - Receives updates at **~150 kHz** rate => saves **10 billions** numbers for an average data taking session



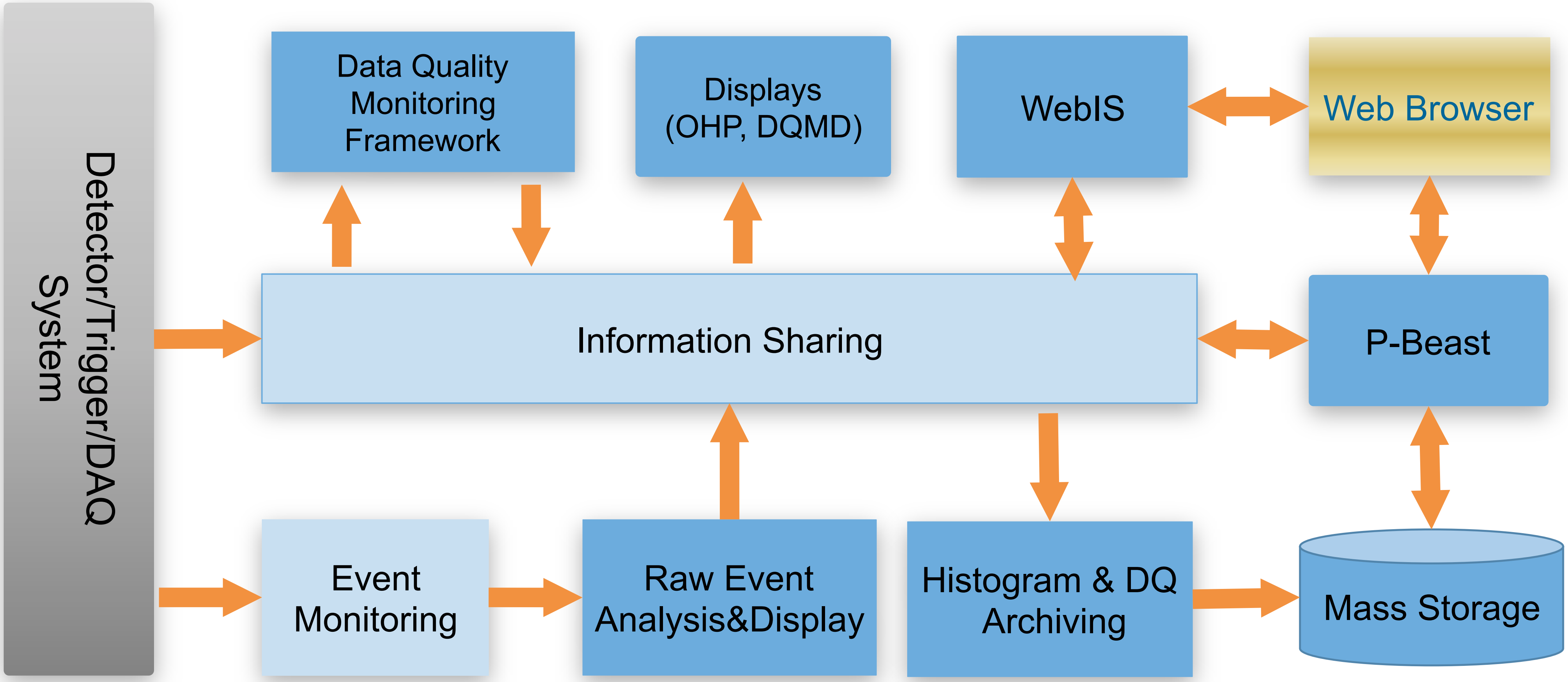
P-Beast: Displaying information in “real-time”

- Information appears in the P-Beast local storage with a few seconds delay
  - It can be used for the online displays
- Access to the information is provided via Web:
  - Customizable dashboards using **Grafana** open platform





The Online Monitoring System Outlook



Conclusion

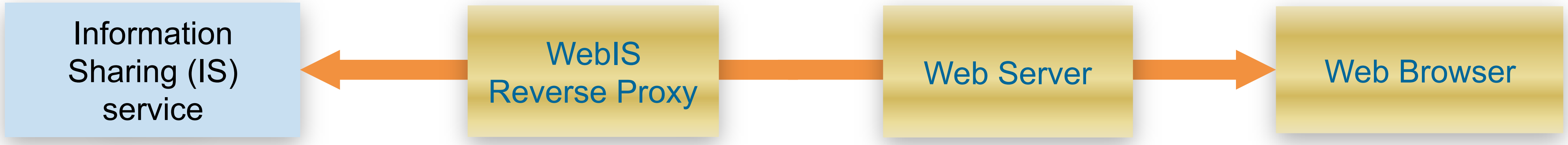
- The ATLAS Online Monitoring system is very flexible and easily extendable due to its hierarchical modular architecture:
  - Open for integration with external software
- The Monitoring Services provide solid base for the Online Monitoring system implementation:
  - Backward compatibility has been always maintained
  - Have been flexible enough to satisfy any new requirement to date
- High-level facilities can be added gradually when:
  - New requirements appear
  - New technologies got available on the market
  - HW evolution turns the impossible into reality
- This approach has proven to be extremely efficient:
  - It will be maintained for the future evolution of the online monitoring software

Backup

Inter-Process Communication for the ATLAS Trigger/DAQ System

- Several technologies have been evaluated in the middle of 90-th to choose the best one for the ATLAS TDAQ system
- Based on the results of this evaluation CORBA (Common Object Request Broker) standard has been adopted:
  - The first implementations of CORBA used for the ATLAS TDAQ were ILU from Xerox company for C++ and JDK built-in CORBA broker for Java
- In 2001 the TDAQ SW has been migrated to another CORBA implementations, which have still been used:
  - omniORB for C++
  - JacORB for Java
- Migration was easy and smooth despite the fact that ILU wasn't completely CORBA-compliant:
  - That was facilitated by the 2-layers architecture as only Monitoring Services were affected

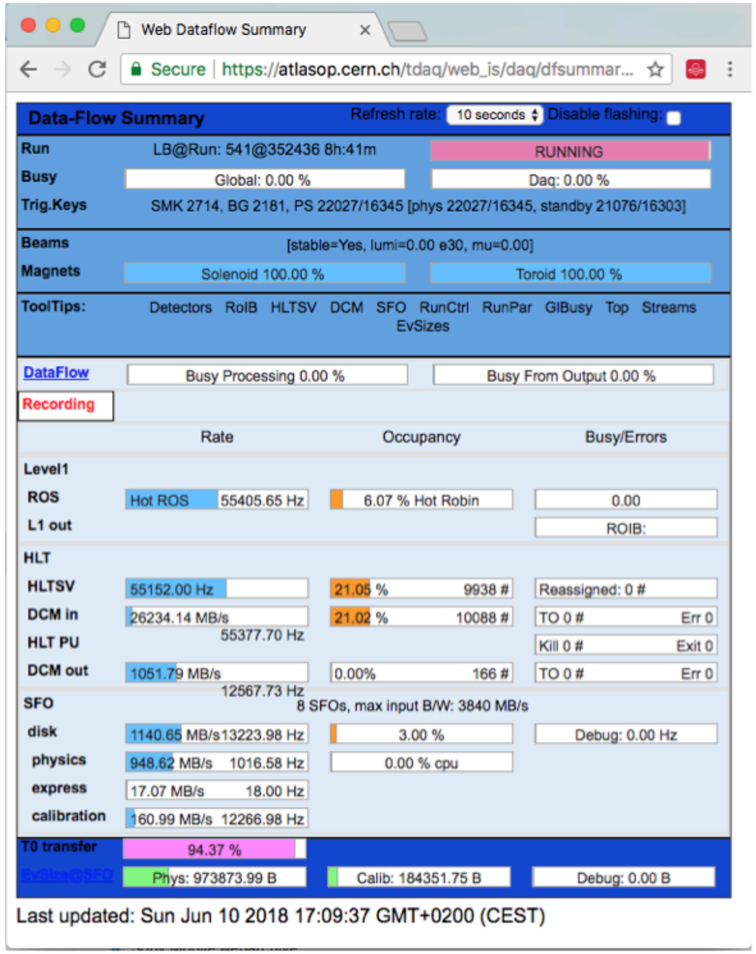
Visualizing Information: Get Access via Web



- Originally started with standalone displaying applications but the power of omni-present Web access has been quickly realized
- REST access style:
  - Each object in the Information Service is assigned a unique URL, e.g.
    - https://atlasop.cern.ch/info/current/ATLAS/is/RunParams/RunParams.RunParams
  - One can type this URL in the Web browser to display the given information
- WebIS – Custom Python HTTP server, which acts as reverse proxy:
  - Reads requested information from IS
  - Converts information to Json and sends it back to the requestor
- Generic facility which can be used to construct complex WEB based GUIs using:
  - CSS, JavaScript, Java applets, standalone Java applications

Web Access gives a lot of flexibility for information visualization

DAQ System Status Monitor



Generic Information Browser