Design and Implementation of a Monitoring System

Serguei Kolos,

University of California, Irvine

ISOTDAQ
International School of Trigger and Data Acquisition

What you are expected to learn in the next hour

- Why systems need to be monitored
- The Basic one-size-fits-all Architecture
  - Technology independent
- Implementation Strategy:
  - With a few technology examples
- Data Quality Monitoring

Why systems need to be Monitored?

- Cos the Universe is not perfect
- The rate of failures is proportional to the system complexity
- Monitoring is indispensable for successful operation

How Higgs boson discovery would look like in an ideal world

What happens in reality

- A complex project has a chance to succeed only if it is ready to deal with problems
- Monitoring System provides the first line of defense:
  - Detects issues
  - Reports them
  - Helps to Investigate

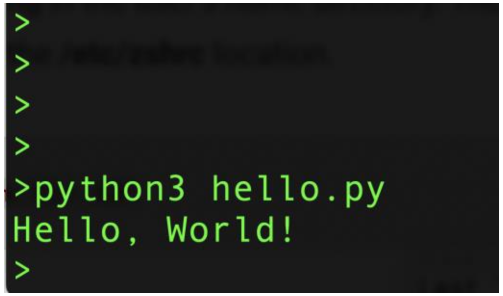
The Simplest Monitoring Example

The monitoring API function

The message

```
print("Hello, World")
```

```
>
>
>
>
>
>
>python3 hello.py
>
```

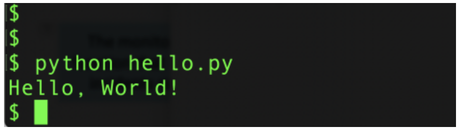


```
>
>
>
>
>python3 hello.py
Hello, World!
>
```

The Basic Monitoring Architecture

print()

The Python
builtins module



```
$
$
$ python hello.py
Hello, World!
$ █
```

- **The critical component**
- All system components should use the same **API**

- Transparent for the system components

- Should be customizable by the end users

The **print()** function drawbacks

print(“Hello, World”)

- Not bad for the ***HelloWorld*** application but doesn’t scale to any real system
- With multiple applications running for a long time on many computers, we want to know:
  - **When** did something happen?
  - **Where** did it come from?
  - **How important** is it?
- Do better solutions exist?

Logging API to the rescue

```python
import logging

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

logging.info("Hello, World!")
```

Use the standard well-designed API

The output format can be easily customized

```
$
$
$ python hello.py
2022-06-14 14:57:37,493 INFO [hello.py:5<module>()] Hello, World!
$
```

Timestamp

Severity

Origin

The Updated Monitoring Architecture


Logging API

Communication

Visualization

- **The Logging API**
  - Well-designed and mature
- The **Logging API** and **Communication** layers are completely independent
- **Communication:**
  - Many of the shelf implementations exist on the market
  - Can be exchanged transparently for the end-user applications

Programming Languages Support

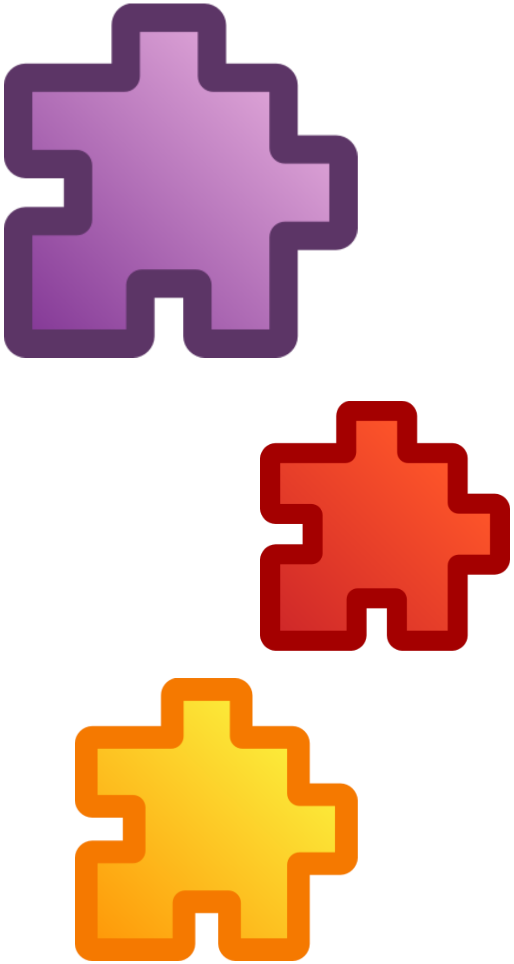
Python

```
import logging

class Logger:
  def critical(msg, *args, **kwargs):
  def debug(msg, *args, **kwargs):
  def error(msg, *args, **kwargs):
  def info(msg, *args, **kwargs):
  def warning(msg, *args, **kwargs):
```

Java

```
import java.util.logging.Logger

class Logger {
  void severe(String msg);
  void fine(String msg);
  void error(String msg);
  void info(String msg);
  void warning(String msg);
}
```

Existing Appenders for the Java Logging API



- **CassandraAppender** - writes its output to an Apache Cassandra database
- **FileAppender** – writes events to an arbitrary file.
- **FlumeAppender** - Apache Flume is a distributed, reliable and highly available system for efficiently collecting, aggregating, and moving large amounts of log data
- **JDBCAppender** - writes log events to a relational database table using standard JDBC
- **NoSQLAppender** - writes log events to a NoSQL database
- **SMTPAppender -** sends an e-mail when a specific logging event occurs, typically on errors or fatal errors
- **ZeroMQAppender -** uses the JeroMQ library to send log events to one or more ZeroMQ endpoints

What about C++?

- Rare case where using MACRO for the public API is a viable option

```
DAQ_LOG_CRITICAL(“File ‘” << file_name << “’ not found”)
DAQ_LOG_ERROR(…)
DAQ_LOG_WARNING(…)
DAQ_LOG_INFO(…)
DAQ_LOG_DEBUG(…)
```

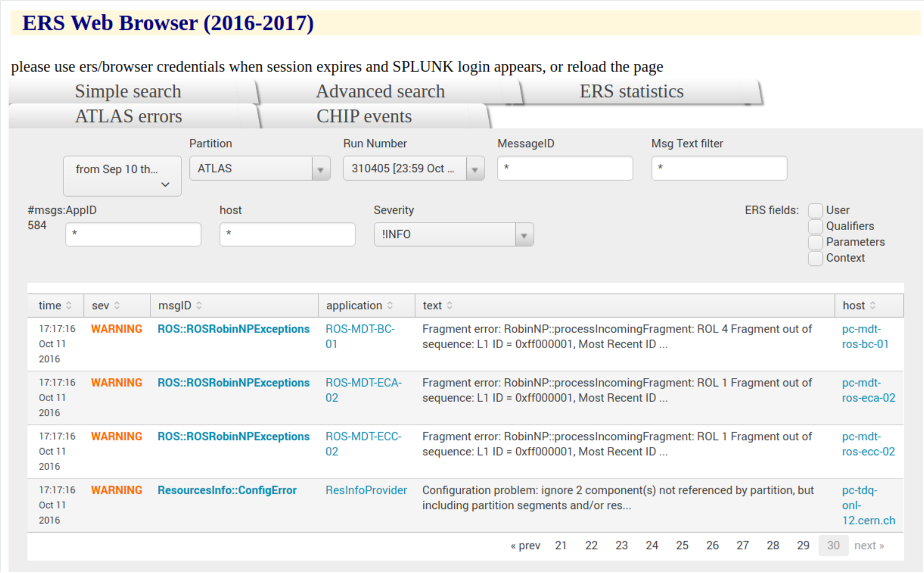
- Initial implementation may be trivial:

```
#define DAQ_LOG_CRITICAL(m) std::cerr << m << std::endl;
```

- A scalable implementation can be provided later:
  - Will not affect users’ code
- Getting better with the C++ language evolution:
  - A few implementations recently appeared based on the new C++20 std::format and C++23 std::print specifications

Example: The ATLAS Error Reporting Service

C++ MACRO → CORBA[1] → Splunk[2]



ERS Web Browser (2016-2017)

please use ers/browser credentials when session expires and SPLUNK login appears, or reload the page

from Sep 10 th...

Partition: ATLAS

Run Number: 310405 [23:59 Oct ...

MessageID: *

Msg Text filter: *

#msgs:AppID 584: *

host: *

Severity: !INFO

ERS fields: ☐ User ☐ Qualifiers ☐ Parameters ☐ Context

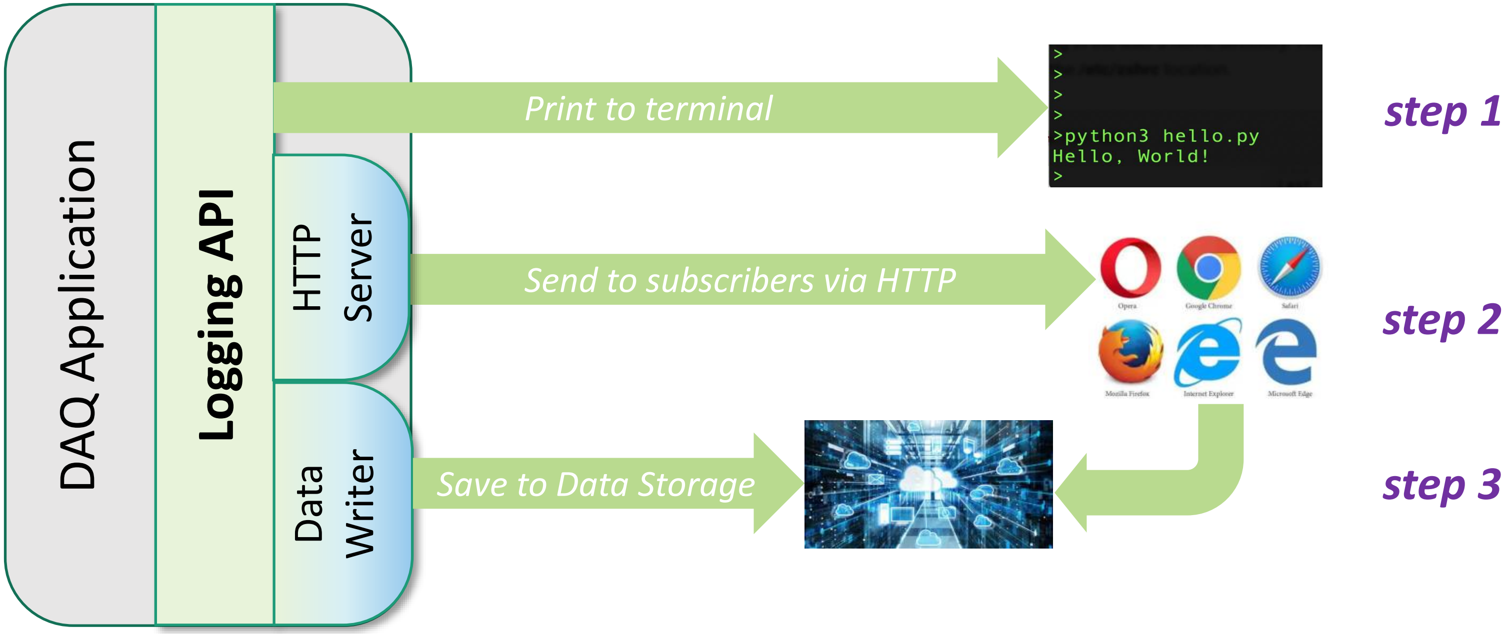
time	sev	msgID	application	text	host
17:17:16 Oct 11 2016	WARNING	ROS::ROSRobinNPExceptions	ROS-MDT-BC-01	Fragment error: RobinNP::processIncomingFragment: ROL 4 Fragment out of sequence: L1 ID = 0xff000001, Most Recent ID ...	pc-mdt-ros-bc-01
17:17:16 Oct 11 2016	WARNING	ROS::ROSRobinNPExceptions	ROS-MDT-ECA-02	Fragment error: RobinNP::processIncomingFragment: ROL 1 Fragment out of sequence: L1 ID = 0xff000001, Most Recent ID ...	pc-mdt-ros-eca-02
17:17:16 Oct 11 2016	WARNING	ROS::ROSRobinNPExceptions	ROS-MDT-ECC-02	Fragment error: RobinNP::processIncomingFragment: ROL 1 Fragment out of sequence: L1 ID = 0xff000001, Most Recent ID ...	pc-mdt-ros-ecc-02
17:17:16 Oct 11 2016	WARNING	ResourcesInfo::ConfigError	ResInfoProvider	Configuration problem: ignore 2 component(s) not referenced by partition, but including partition segments and/or res...	pc-tdq-onl-12.cern.ch

[1] Common Object Request Broker Architecture – inter-process communication technology

[2] Splunk – A software platform to stream and collect data

Monitoring System evolution during the project lifetime

- The destination of the messages can be changed at any moment:
  - No changes in the Software Applications required!
- Data Storage is optional but very handy:
  - Adds **persistence** – can be used for postmortem analysis



```
>
>
>
>
>python3 hello.py
Hello, World!
>
```

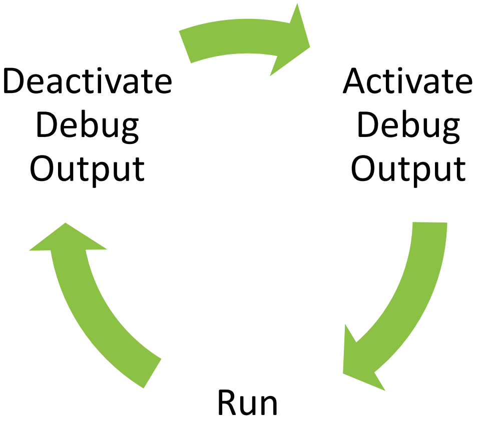
Set Priorities Properly

- Choose (or implement) the Monitoring **API** before starting to implement the DAQ system:
  - The Monitoring must be used by all components of the DAQ system
  - Changing them later will be a pain
- Can take care about **Communication** and **Visualization** implementations later:
  - Using simple output to terminal would be sufficient for the beginning

- Advantages:
  - Using the monitoring system will exercise its functionality and performance
  - Learn the best ways of presenting information
  - Speed up the DAQ system development

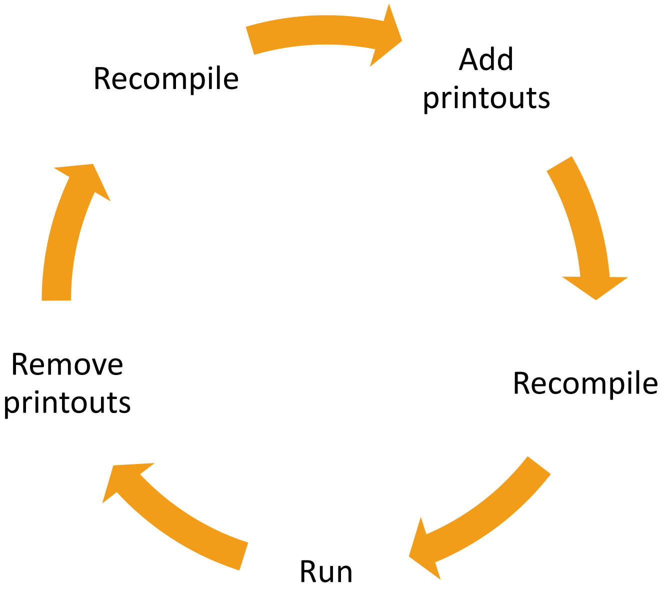
How Monitoring System can speed up DAQ System Development

Efficient development cycle using Monitoring API



- *Reduces time for debugging*
- *Optimizes the placement of DEBUG output in the code*

"Traditional" development cycle



Can we Extend the Same Ideas to the Other Types of Monitoring Data?

Monitoring Data Types

- **Messages** – used to inform about anything of importance that happens in the system
- **Metrics** – show how the system performs:
  - Values of properties of the software and hardware system components
  - ***Counters***, *Gauges* and ***Histograms***

Main Metrics Types

Counter

- Monotonically increasing integer number
- Simple to monitor:
    - Last value for the last time period
- Examples:
    - Cumulative totals: number of triggers, number of bytes sent/received, etc.

Gauge

- Arbitrary changing value:
    - Integer or floating point
- Monitoring can be tricky:
    - Last value
    - Mean value
    - Min/Max values
- Examples:
    - Resources usage: CPU, memory, buffe
    - Rates: triggers/s, bytes/s, etc.
    - HW Properties: voltage, current, temperature, etc.

Metrics Monitoring Requirements

✓Must be displayed as time series

✓Must be accessible in real-time

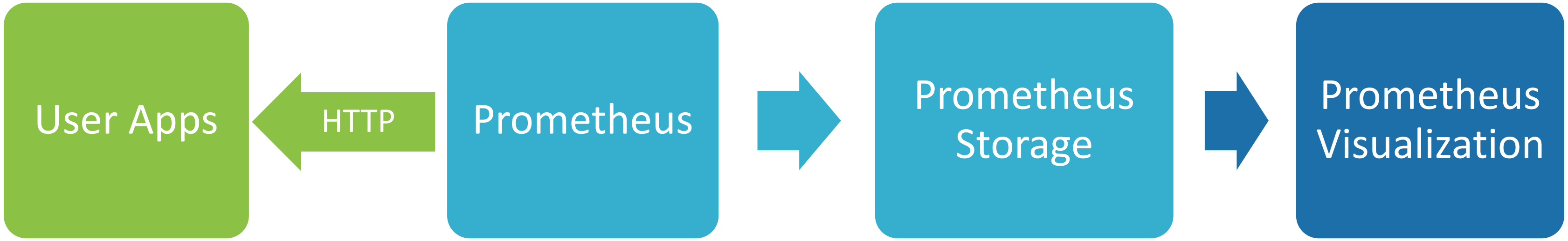
✓Must be recorded to be checked later

Reusing the Same Architecture

- For both **Communication** and **Visualization** components many implementations exist on the market:
    - All they need is a stream of time series data
    - They can store and visualize them
- They may be freely exchanged during the project’s life-time
- For this the **API** **must be independent** of the **Communication** and **Visualization**

Does a Common API for Metrics exist?

- There is no commonly accepted API for Metrics
- SW tools for metrics collection and analysis usually define just a format of metrics stream they accept
- A well-known example is Prometheus:
  - Retrieves data via HTTP
  - No programming language API



User Apps

HTTP

Prometheus

Prometheus Storage

Prometheus Visualization

Custom API for Metrics Monitoring

```
package Atlas.Monitoring;

interface Gauge {
    void setValue(double v);
}

interface Counter {
    void increment();
    void reset();
}

interface Metrics {
    Counter createCounter(String name)
                 throw (AlreadyExistsException);
    Gauge createGauge(String name)
                 throw (AlreadyExistsException);
}
```

*Makes it independent of the **Communication** implementation*

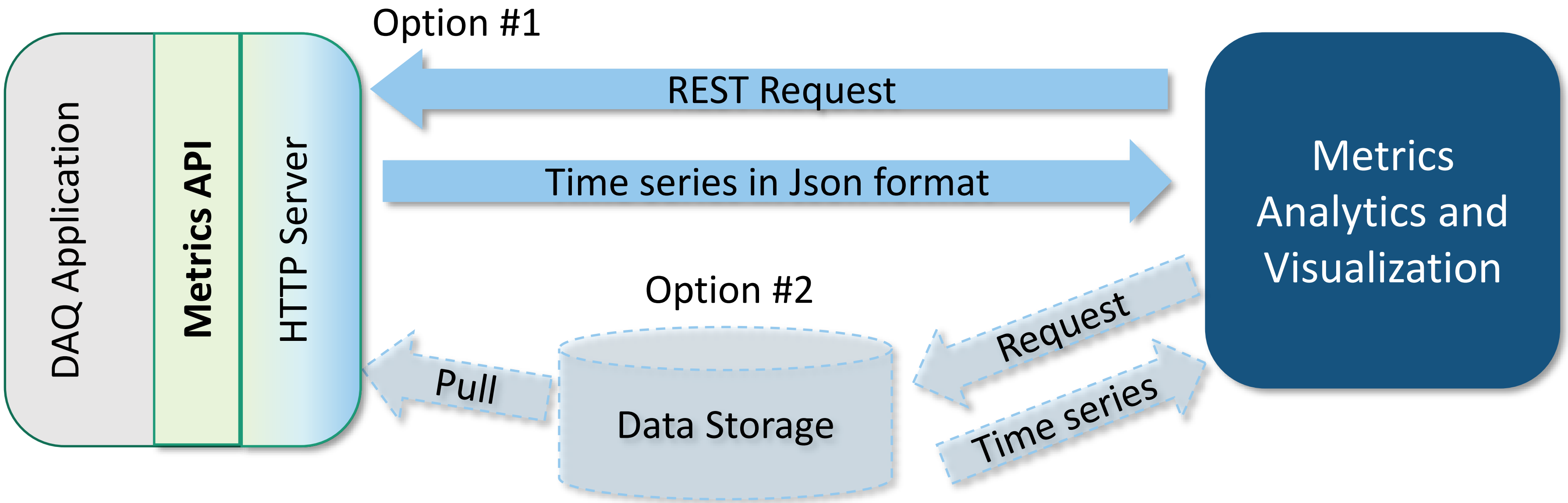
Supports different treatment for Counters and Gauges

Enforces uniqueness of Metrics IDs

Metrics IDs

- All Metrics must have unique IDs
- Uniform human-readable naming schema greatly simplifies Metrics handling:
  - Finding required Metrics is straightforward
  - Easy selection and filtering using regular expressions
- A possible approach:
  - ***`System/Sub-system/Component/Metrics`***
- Examples:
  - */ATLAS/DAQ/EventRecoder/**EventsNumber***
  - */ATLAS/DAQ/EventRecoder/**RecordingRate***

Some Implementation Options



- The underlying implementation can be updated as the project evolves:
  - Does not affect the applications
  - The same Analytics and Visualization tools can still be used

RESTful Protocol

- REST – **Re**presentational **S**tate **T**ransfer
- Client-server HTTP-based stateless communication protocol
- Supported by most of the modern information storage as well as Web-based Visualization systems:
  - Supports seamless interoperations
- Makes it easy to switch from one Storage or Visualization platform to another

REST Protocol Example

- Request:

```
https://atlasop.cern.ch/monitoring/
    ? id=ATLAS.Dataflow.RecordedEvents.Rate
    & from=now-30d
    & to=now
```

- Response:

  Json Time Series, e.g.:

```
[
     {t:1579104640,v:12345},
     {t:1579104645,v:12346},
     {t:1579104650,v:12347},
     {t:1579104655,v:12348}
]
```

Web-Based Visualization Tools

- Javascript tools which work in Web Browsers:
  - **Grafana** - the open observability platform
  - **Prometheus** – monitoring platform
  - **D3** – a low-level JavaScript toolbox for data visualization
  - **Rickshaw** – a JavaScript toolkit for creating interactive time series graphs
  - There are many others as well…
- Very convenient for the end users:
  - Don’t require extra software installation
  - Provide real-time monitoring data access from any place of the World

Are there some other Advantages of the common API?

- The **API** can hide implementation of common data handling patterns

➢ Produce Derivative Metrics

➢ Perform Metrics Rate Downsampling

➢ Keeps “Observer Effect” under control

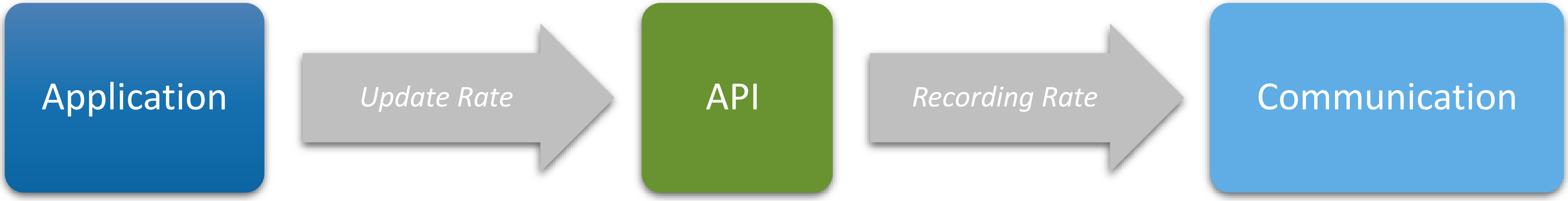
Derivative Metrics

```
import Atlas.Monitoring;

Counter events =
  Metrics.createCounter( “/DAQ/EventRecoder/Events”);
…

void eventReceived() {
  evenets.increment();
}
```

1. “/DAQ/EventRecoder/Events”
2. “/DAQ/EventRecoder/EventsRate”

- Derivative Metrics can be automatically produced:
  - Counters => Rates
  - Gauges => Min, Mean, Max, Frequency distributions (histograms)

Metrics Rate Down-sampling



Application

Update Rate

API

Recording Rate

Communication

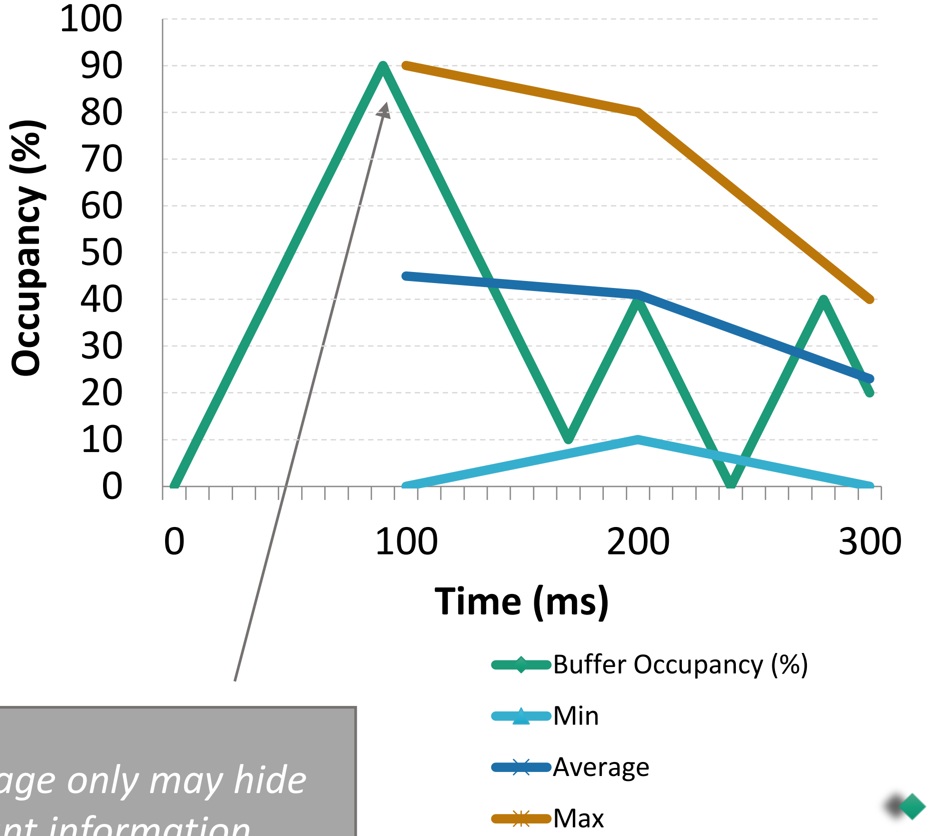
- Metrics update rate is defined by the data handling rate:
  - E.g. rate of triggers for the ATLAS experiment is 100 kHz
- High update rates must be scaled down:
  - Takes too much space in the data storage
    - 100 kHz of event rate => (8 + 8)*3600*10^5 = ~6 GB data per hour per single metrics
  - Cannot be visualized:
    - 4K displays have 3840 pixels along X axis
    - Can display data for 40ms only

Metrics Rate Down-sampling

- Metrics values can be down-sampled by the **API** implementation:
  - Reduces recording rate
  - Simplifies storage requirements
- Output update interval can be made configurable:
  - A default value for all metrics
  - Individual values per specific metrics
- Transparent for the **Applications** and **Communication** components

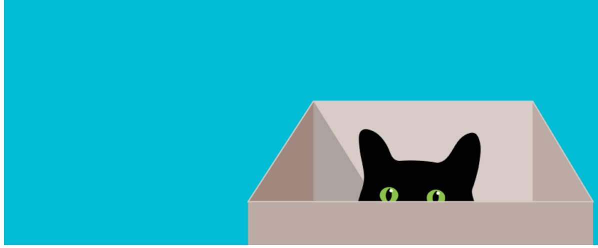
Down-sampling: Counters vs Gauges

- Counter:
  - Publish the last value for each output update interval
- Gauge:
  - Publish three values for each update interval:
    - Min, Average, Max



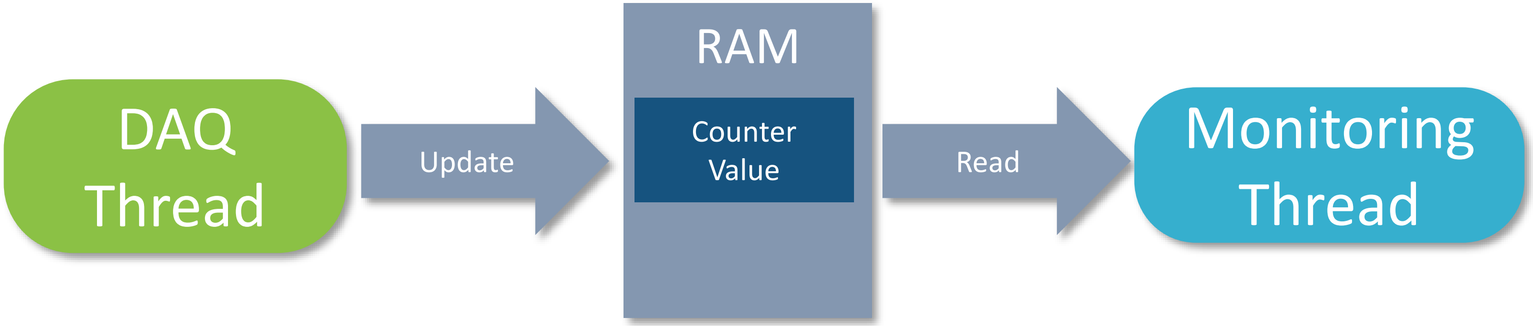
Using Average only may hide important information

The Observer Effect



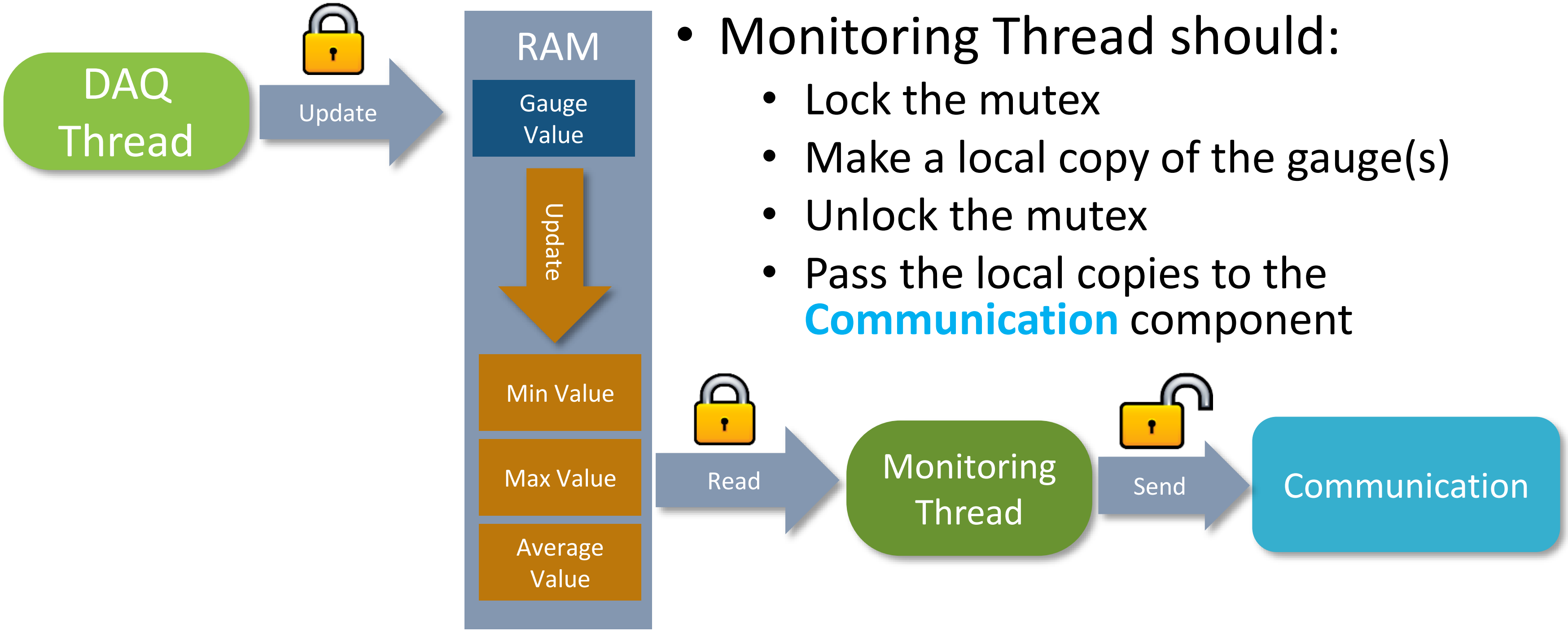
- An observation affects the system:
  - It consumes resources (CPU, memory, network bandwidth)
  - It may affect performance of the monitored application
- Information must be passed to the **Communication** component **asynchronously**:
  - Monitoring information is updated by the DAQ thread
  - Down-sampling and publishing must be done by another thread
- Thread-safety must be considered:
  - But excessive thread-safety measures may hit the DAQ application performance

Thread-safety Overhead: Counters



- Counters don’t require critical section
- Atomic variables are available in all modern languages

Thread-safety Overhead: Gauges



- Monitoring Thread must not hold the lock when passing data to **Communication** component
- Monitoring Thread should:
  - Lock the mutex
  - Make a local copy of the gauge(s)
  - Unlock the mutex
  - Pass the local copies to the **Communication** component

Thread-Safety Overhead

- Locking an unlocked mutex takes ~50 CPU cycles => 20ns
- Not a problem when:
  - DAQ thread locks the mutex often
  - Monitoring threads locks it once every few seconds
- Mutex contention happens when a mutex is locked by multiple threads equally often
- Even non-contended mutex may produce overhead:
  - 10 kHz input rate:
    - Mutex locking takes 0.2ms every second => 0.02% overhead
  - 1 MHz input rate:
    - Mutex locking takes 20ms every second => 2% overhead

Scaling up the Monitoring System

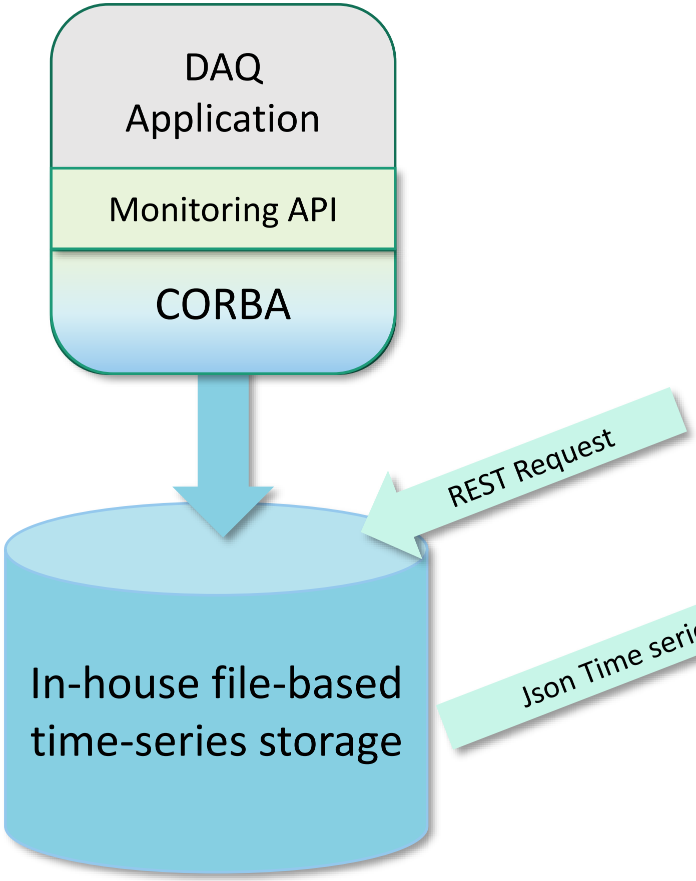
The HEP Experimental Realm

- DAQ system of a modern HEP experiment consists of:
    - O(1K) computers and network devices
    - O(10K) SW applications
    - O(100K) Metrics
- A single counter metrics for 24h run requires:
    - (8 + 8)*360*24 = **138K**B of storage
- 100K Metrics => 14GB per day => 100GB per week => **5TB per year**
- **The main difficulty is given by the O(10)KHz of data generation rate**

Time-Series Storage Systems

- Dedicated time-series storage systems would usually work better than general purpose RDBMS
    - **Whisper** – a lightweight, flat-file database format for storing time-series data
    - **InfluxDB** – a time-series database written in Go
    - **Cassandra** – scalable, high availability storage platform for time-series data
    - **MongoDB** - a general purpose, document-based, distributed database
    - **Prometheus** – monitoring platform that has its own time-series optimized storage

The ATLAS Experiment: Web-based Metrics Monitoring



DAQ Application

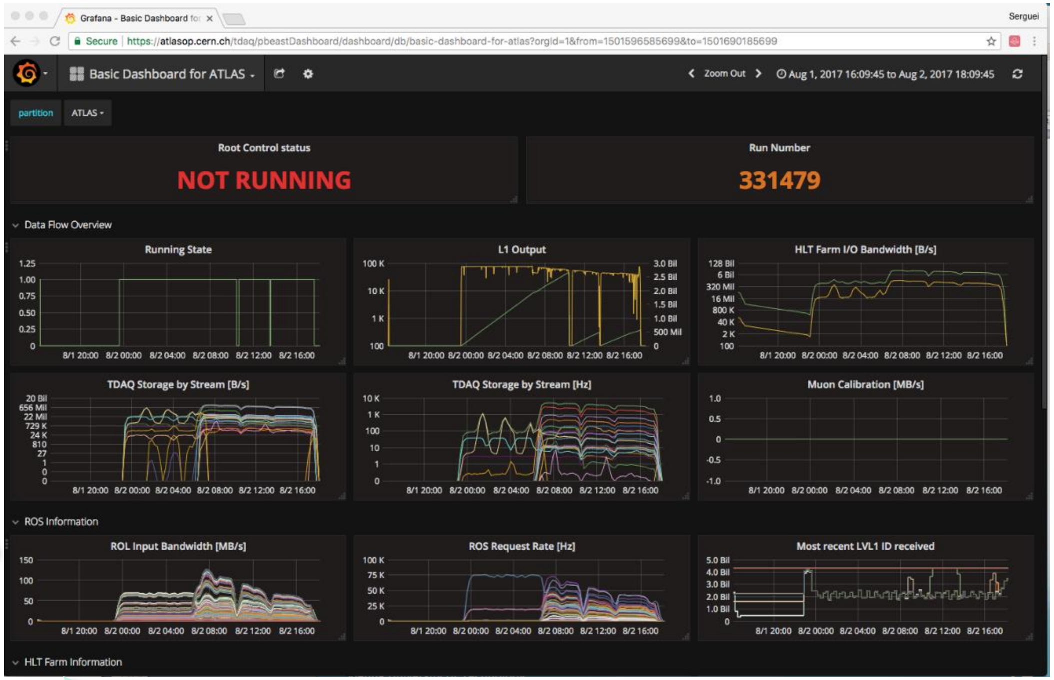
Monitoring API

CORBA

In-house file-based time-series storage

REST Request

Json Time series



Root Control status

NOT RUNNING

Run Number

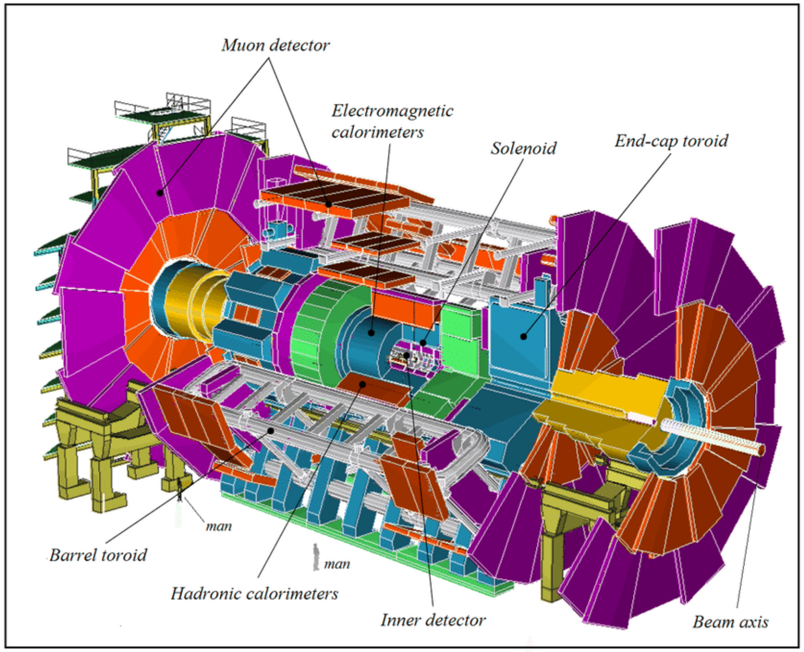
331479

Grafana customizable dashboard

DAQ Specialty:
Data Quality Monitoring

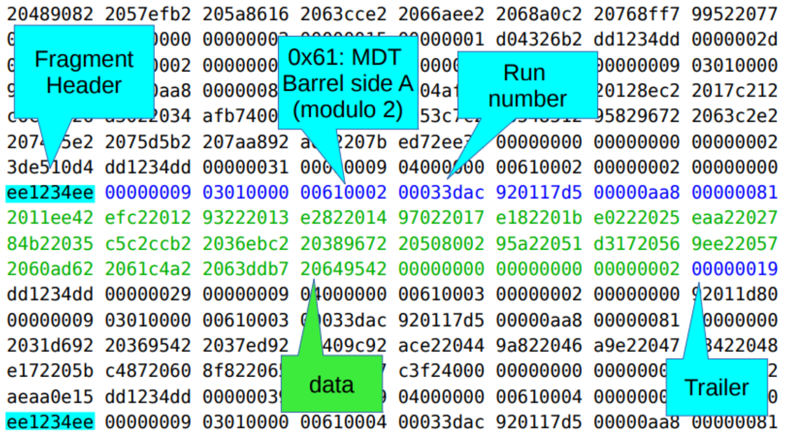
How to Monitor the Detector?

- Detectors of LHC experiments are incredibly complex devices:
    - Up to 10^8 output data channels
    - Mostly custom electronics
    - 40 MHz operational frequency
- Traditional monitoring would yield in O(1) PHz (petahertz) of metrics update rate:
    - These metrics are not even attempted to be produced
- However, DAQ system has a handle on these metrics…

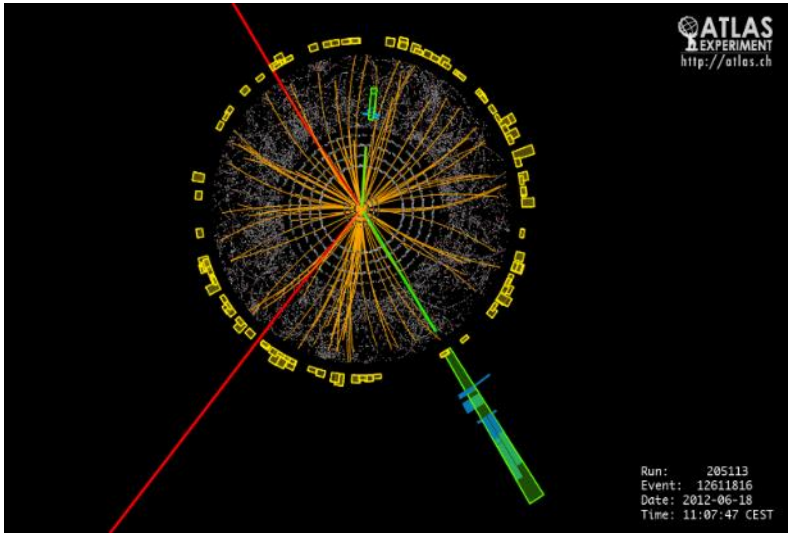


Detector Metrics

- Every **Physics Event** contains states of a sub-set of detector channels:
    - An expert can spot problems by looking into a graphical event representation
    - Such experts are not many and can't be in the Control Room 24/7

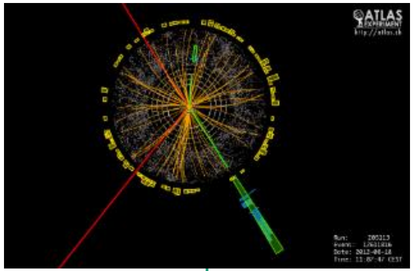


```
20489082 2057efb2 205a8616 2063cce2 2066aee2 2068a0c2 20768ff7 99522077
0            0000 0000000           00000001 d04326b2 dd1234dd 0000002d
0            0002 0000000           0000              00000009 03010000
9            0aa8 0000008           04a               20128ec2 2017c212
c            2034 afb7400           53c7              95829672 2063c2e2
207 5e2  2075d5b2 207aa892 a  2207b ed72ee   00000000 00000000 00000002
3de510d4 dd1234dd 00000031 000 0009 04000 00 00610002 00000002 00000000
ee1234ee 00000009 03010000 00610002 00033dac 920117d5 00000aa8 00000081
2011ee42 efc22012 93222013 e2822014 97022017 e182201b e0222025 eaa22027
84b22035 c5c2ccb2 2036ebc2 20389672 20508002 95a22051 d3172056 9ee22057
2060ad62 2061c4a2 2063ddb7 20649542 00000000 00000000 00000002 00000019
dd1234dd 00000029 00000009 0 000000 00610003 00000002 00000000  2011d80
00000009 03010000 00610003  033dac 920117d5 00000aa8 00000081  0000000
2031d692 20369542 2037ed92  409c92 ace22044 9a822046 a9e22047  3422048
e172205b c4872060 8f82206           c3f24000 00000000 0000000         2
aeaa0e15 dd1234dd 0000003           04000000 00610004 0000000         0
ee1234ee 00000009 03010000 00610004 00033dac 920117d5 00000aa8 00000081
```



Automated Data Quality Analysis

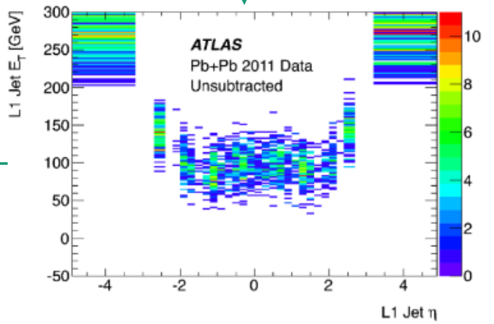
- Dedicated DAQ applications apply standard physics analysis algorithms to a statistical sub-set of ***Physics Events***:
  - Extract Detector Metrics and build their statistical distributions(histograms)
  - Analyze histograms and produce a new set of Metrics – Data Quality statuses



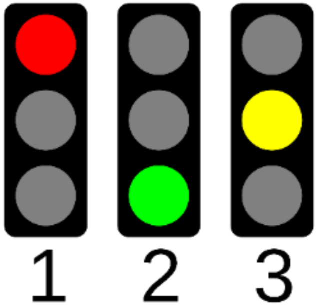
Samples of Physics Events

Physics Event Analysis Algorithms

Statistical Distributions



Statistical Analysis Algorithms



Summary: The Key Points

- ☑ Have your Monitoring System API ready from the beginning of the main project
- ☑ Use standard Monitoring APIs whenever it is possible:
  - e.g. Logging API
- ☑ Think carefully when designing a custom API:
  - It must not depend on a particular technology
- ☑ The Monitoring System implementation may evolve during DAQ system development
- ☑ Use existing solutions for Communication and Visualization components:
  - In-house development must be well justified