Database applications and developments in ATLAS

Dario Barberis

Genoa University/INFN

Topics

DB

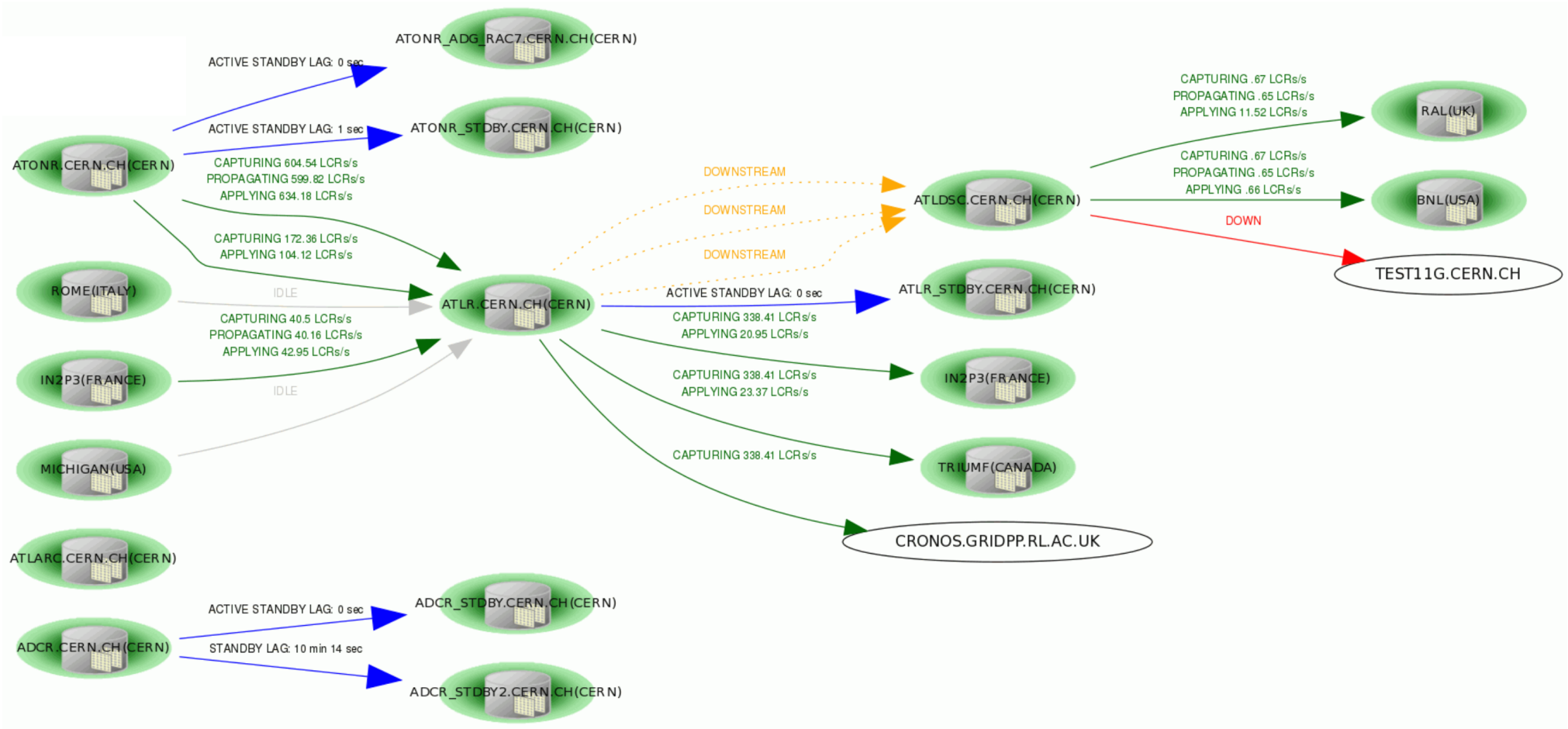
- Database usage by ATLAS in LHC Run2

PCD WS

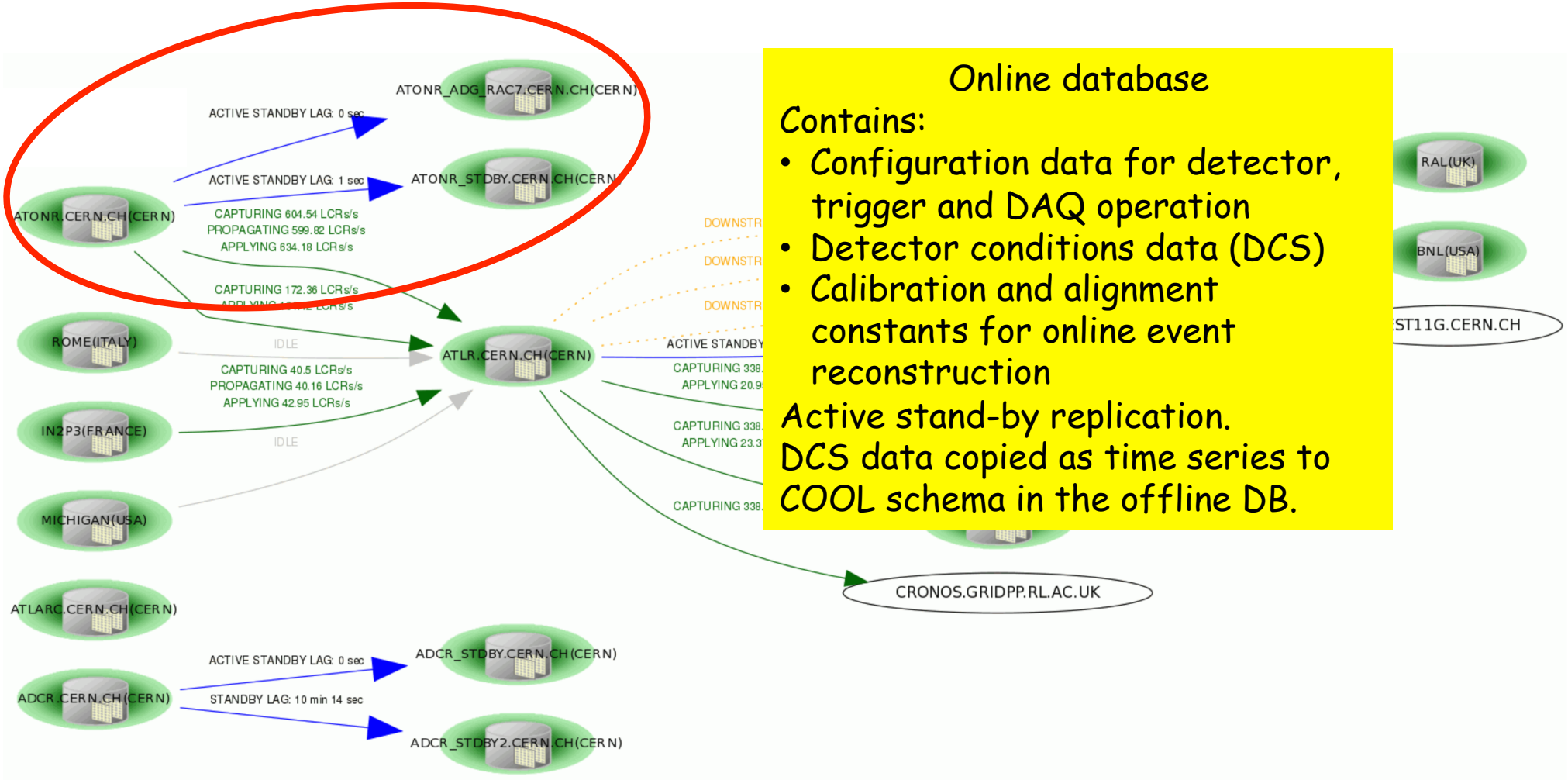
- Ideas for a Physics Conditions Data Web Service for Run3

- Event cataloguing: the new EventIndex

Databases in ATLAS for Run2



Databases in ATLAS: ATONR



Online database

Contains:

- Configuration data for detector, trigger and DAQ operation
- Detector conditions data (DCS)
- Calibration and alignment constants for online event reconstruction

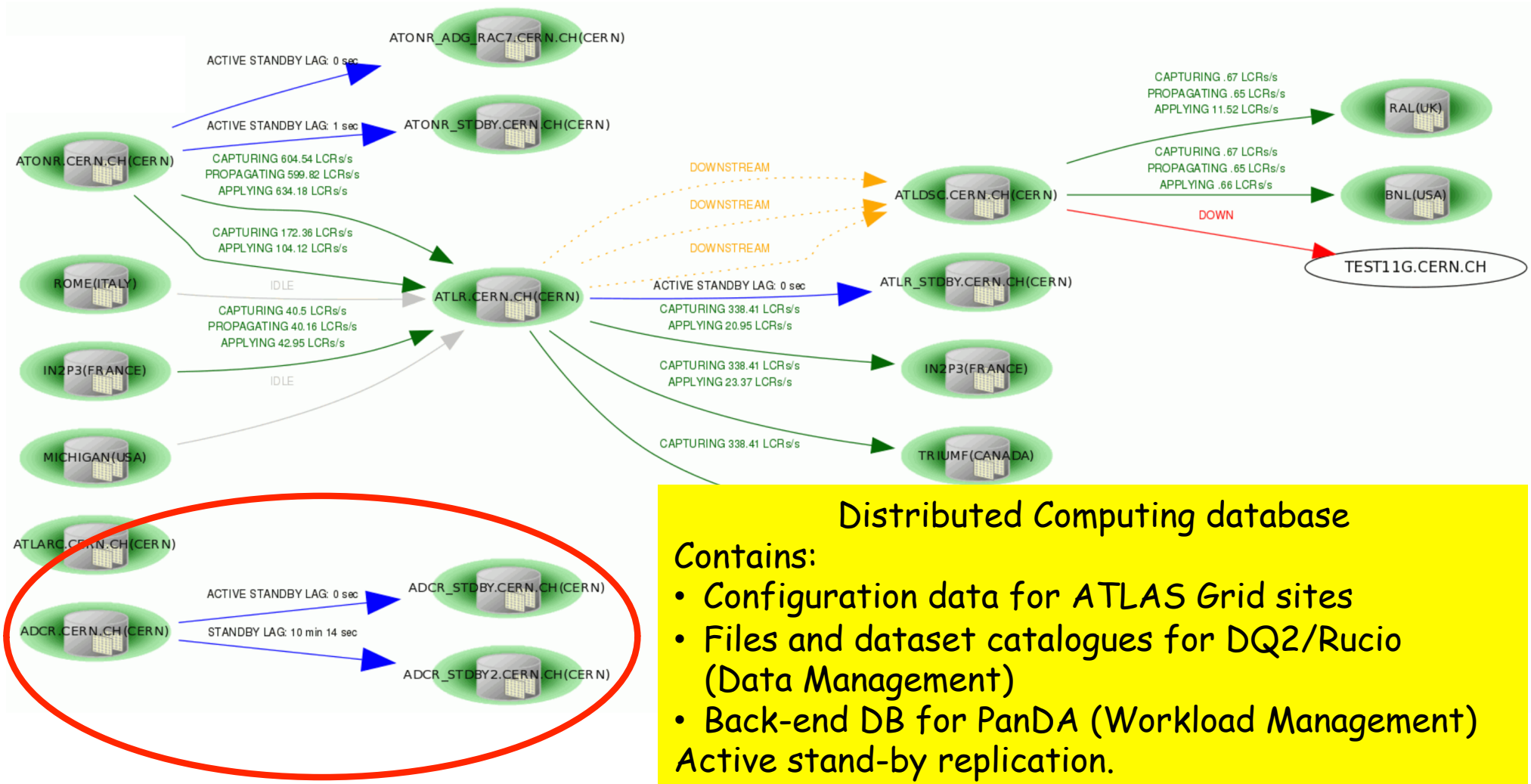
Active stand-by replication.

DCS data copied as time series to COOL schema in the offline DB.

Digression: COOL in 1 slide

- COOL is the database schema used by ATLAS and LHCb to store all time-dependent conditions data
- Data are organised in
  - (hierarchical) "folders"
    - Each folder contains a single data type
  - Intervals of Validity (IoV)
    - Each IoV refers to a time range during which the associated data are valid
    - IoVs can be expressed as timestamps or run-lumiblock
  - Versions
- Versions can be associated an alphanumeric "tag"
- "Global tags" can be associated to groups of folder versions that have to be used together
  - Selecting a given global tag in a job guarantees that the retrieved data will be consistent.
- COOL is written in C++ and uses (through the CORAL interface) an SQL database as data store
  - Large data structures can be stored in external files referenced by COOL

Databases in ATLAS: ADCR



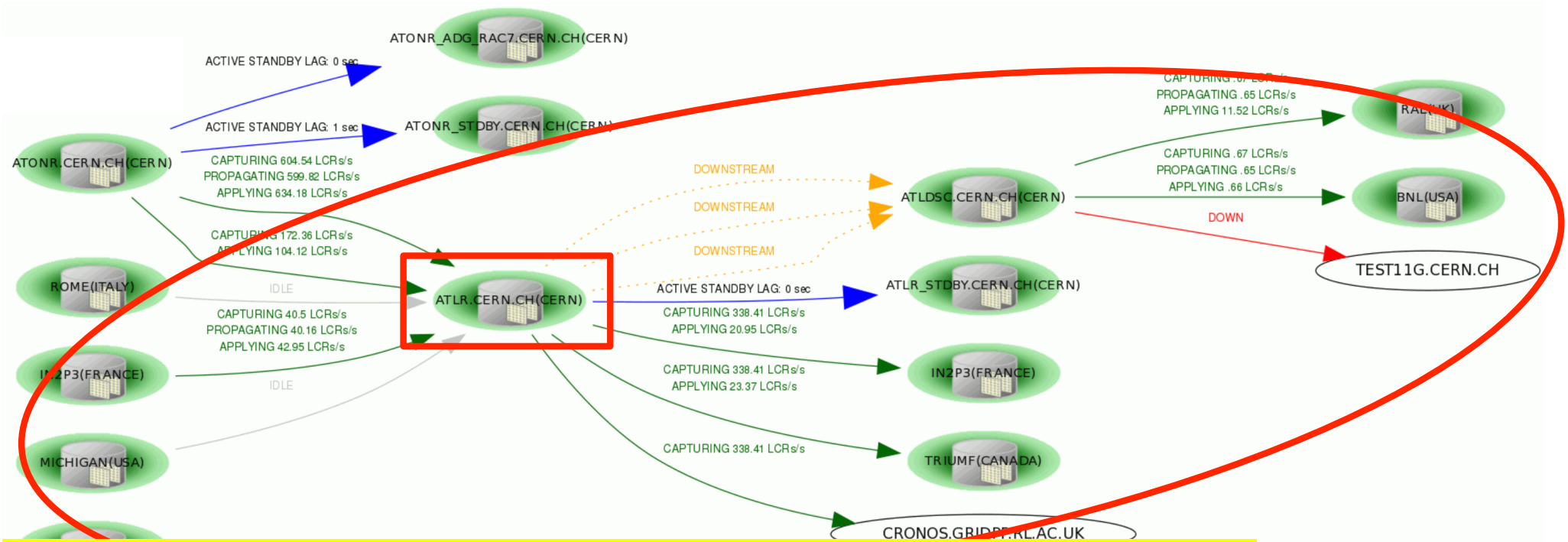
Distributed Computing database

Contains:

- Configuration data for ATLAS Grid sites
- Files and dataset catalogues for DQ2/Rucio (Data Management)
- Back-end DB for PanDA (Workload Management)

Active stand-by replication.

Databases in ATLAS: ATLR



Conditions database

Contains:

- Detector geometry and trigger DB
- Detector conditions data (DCS)
- Calibration and alignment constants for offline event reconstruction

Active stand-by replication.

COOL data replicated to IN2P3-CC, RAL and TRIUMF for Frontier access.

Gets data from Muon Calibration Centres and AMI DB in Lyon.

A Conditions Data Service for Run3 (1)

PCD WS

- We learned a lot from the present implementation of COOL+CORAL (schema+interface)
  - It has been a success in Run1 and (hopefully) Run2 for ATLAS Conditions Data
- Nevertheless we became also aware of a certain number of limitations of the present system, that we may try to overcome for Run3
  - We should also profit of CMS experience to establish the basis of a future CondDB project aiming at improving the present software chain and overcoming the limitations
- Main problems encountered:
  - API has grown following a lot of requirements
    - today we know that some of those were not really needed
  - Global Tag administration is complex, should be better integrated
  - API installation is difficult, most common usage is from lxplus
  - CERN-IT support is decreasing: it could be difficult to adapt CORAL to other relational technologies in case we decide to move away from Oracle
  - We mixed raw (DCS) conditions data with calibration data — this should be avoided in the future

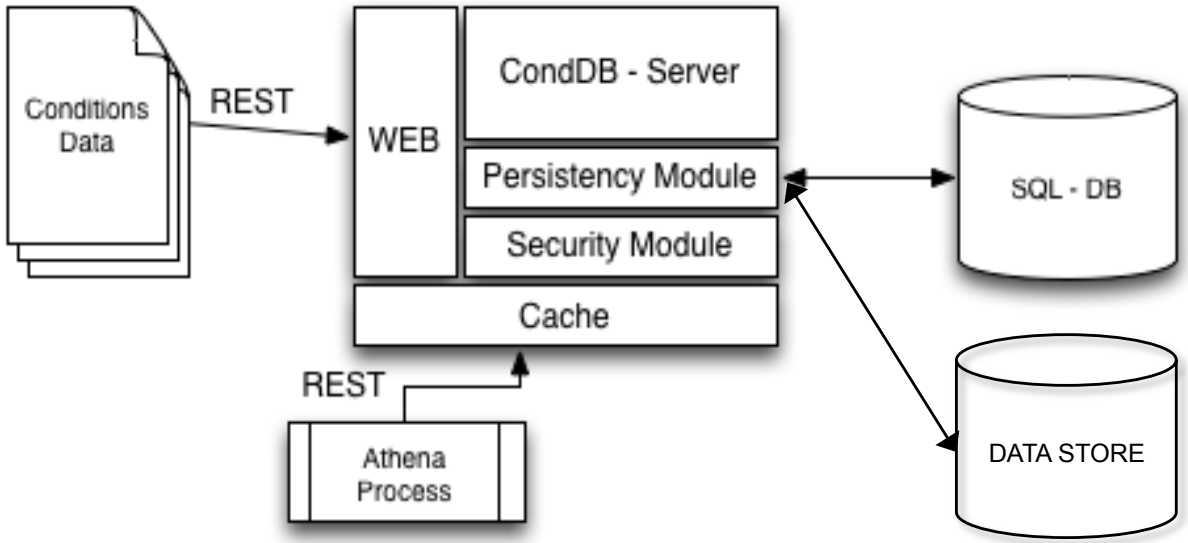
A Conditions Data Service for Run3 (2)

PCD WS

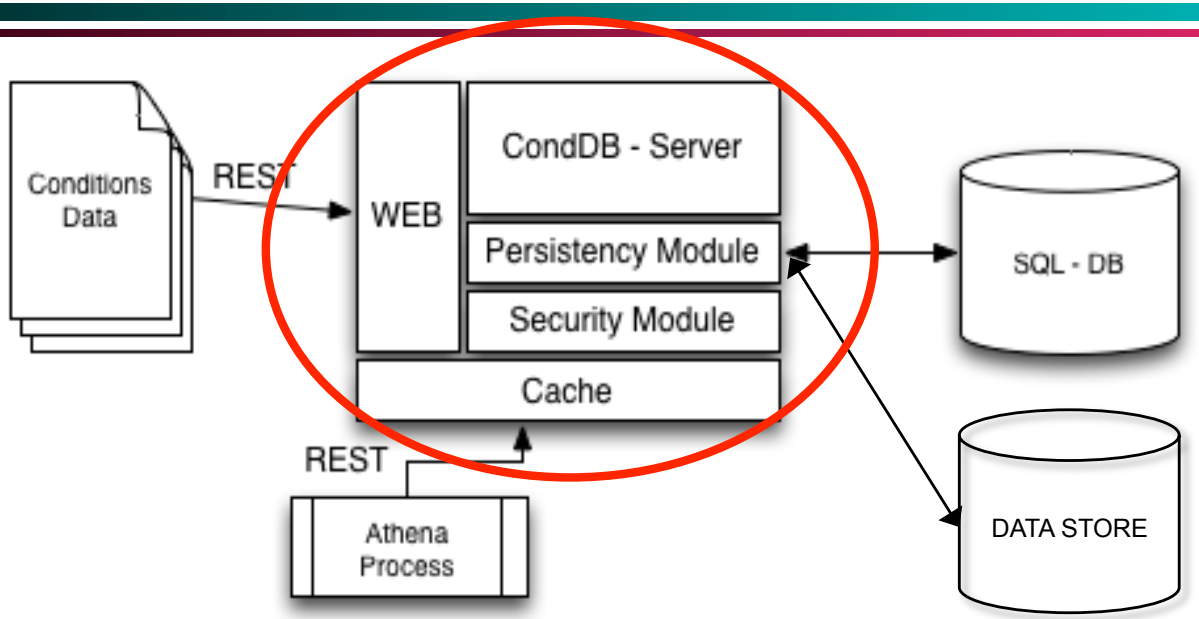
- A new architecture should:
  - Be database technology independent (as much as possible)
    - Once a technology is chosen it is always better to profit of it as much as possible
    - But the core part of the architecture should be independent of the implementation and support as many relational platforms as possible
  - Restructure COOL metadata to include Global Tags in a better way
    - The metadata experience we have accumulated via COMA indicates without any doubt that we can simplify the present relational structure.
    - We know today what we really need: all the information is in COOL, but we need to re-organise it to improve the administration and the access.
    - Additional feedback may come from CMS: their ConditionsDB has been completely re-designed for Run2, and we need to discuss pro and contra of their new solution.
  - Simplify client access
    - Push further the Frontier approach using RESTful services. The business code should sit in a server, the client has to be as light as possible.

A Conditions Data Service for Run3 (3)

- Simplified view of multi-tier architecture:
  - Use simple clients and HTTP REST services in the same way as Frontier
  - Decouple the CD service from the framework, in order to be able to access the service from anywhere and any system.
  - The client should be very lightweight.
- A relatively simple system with similar functionalities to those we have now with CondDB (Oracle) + POOL files + Frontier + IovDBService in Athena can be composed of:



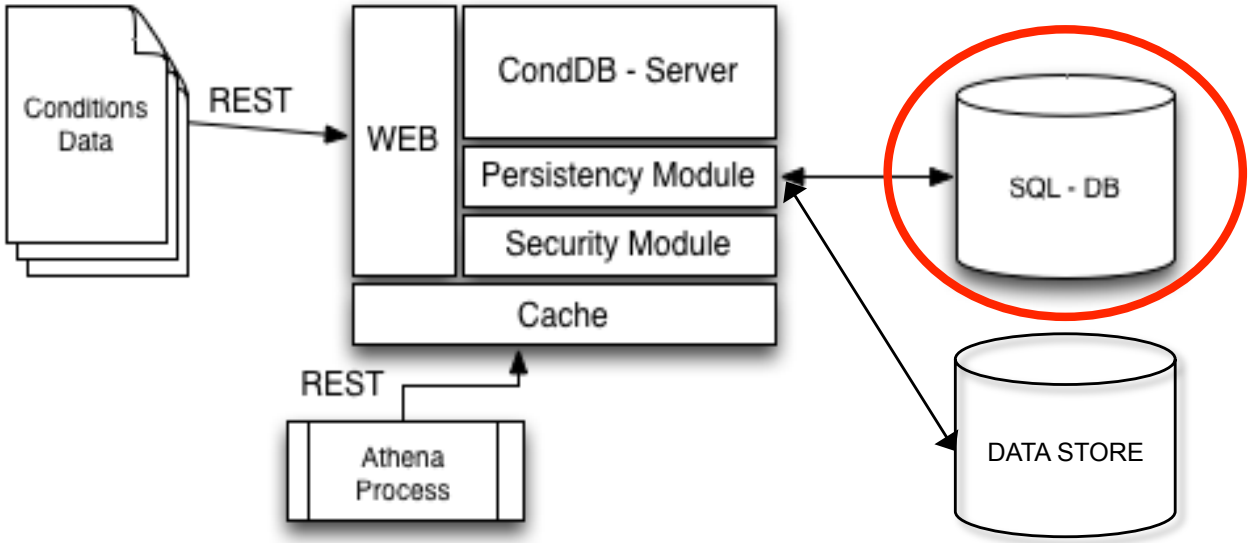
Conditions Data Web Service (1)



a) A Web Server that connects all the other components and acts as a single front-end to the data storage facility(ies).

- It would contain all the tools to add data, organise them "properly", index what has to be indexed, and serve the data to the clients.
- This is the core system.

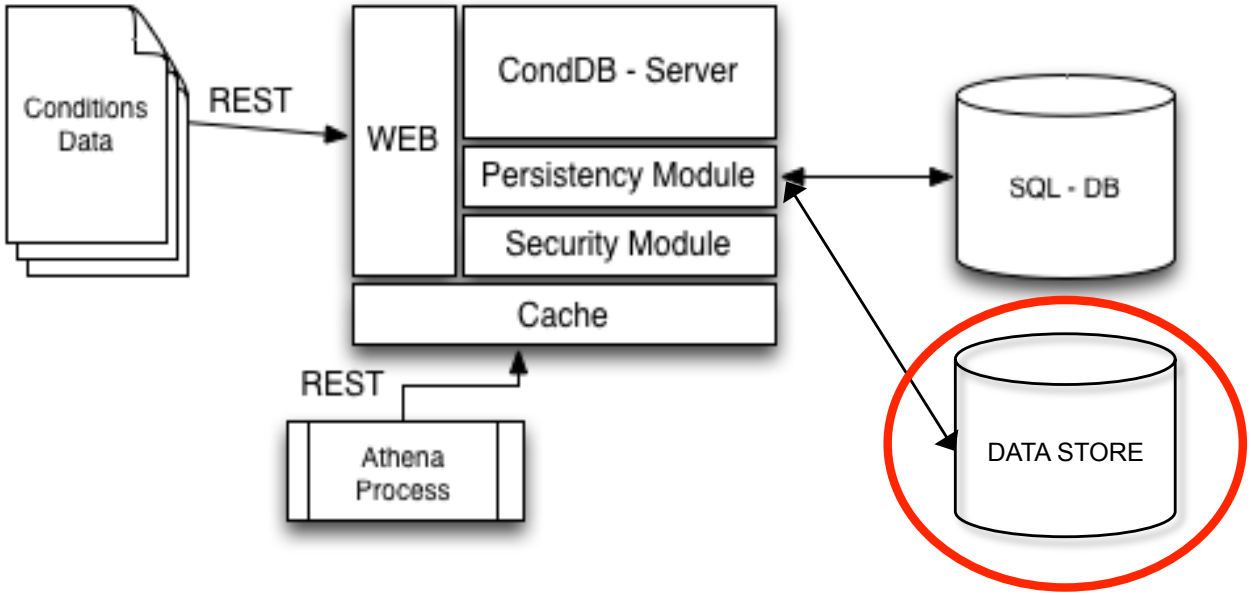
Conditions Data Web Service (2)



b) A relational DB to hold all metadata, IoVs, tags, global tags and whatever else is necessary, including references to the actual data stores.

- If this part is relatively simple it could be kept in "standard" SQL with an implementation that could be moved from Oracle to (say) PostgreSQL or anything else supported by CERN in the future.

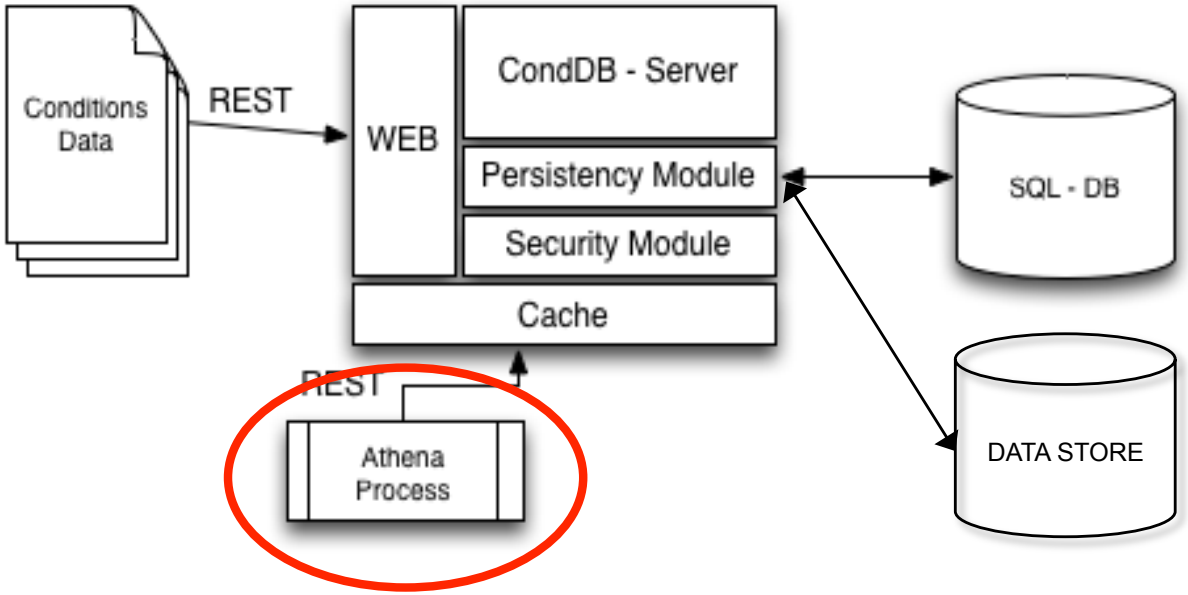
Conditions Data Web Service (3)



c) A data store for the actual data. The data should be organised in relatively large chunks, optimised for loading by Grid jobs.

- These chunks could be BLOBs (of whichever type) in the SQL DB or files in a NoSQL system (Hadoop or similar).
- The advantage of a NoSQL system is that there are already utilities for organising the data and do fast search and retrieval without having to define "too much" the schema apriori.

Conditions Data Web Service (4)



d) A trivial client that takes requests for a given data type, time range (Interval of Validity) and (global) tag, calls
curl http://cdserver.cern.ch/something
and returns the payload in JSON format.

Conditions Data Service Development

PCD WS

- So far this is little more than an idea
- Started talks with CMS as they developed part of this global idea (the CondDB schema in Oracle) already for Run2 and are deploying it now
- The timescale is for Run3 but if done earlier it's not bad
- Belle2 also expressed an interest – we'll see as they need something ready for 2016
- The project can be kept totally independent of the contents (and therefore of the experiment)
- This project is "small" – can be done by a few dedicated people if they have complementary competence and experience
- In the discussions so far: **A. Formica** (Saclay/ATLAS), E. Gallas (Oxford/ATLAS), D. Barberis (Genoa/ATLAS), G. Govi (FNAL/CMS)

It's his idea!

The ATLAS EventIndex

- A complete catalogue of ATLAS events
  - All events, real and simulated data
  - All processing stages
- Contents
  - Event identifiers
  - Online trigger pattern and hit counts
  - References to the events at each processing stage (RAW, ESD, (x)AOD, NTUP) in all permanent files on storage (file GUID plus internal pointer to the event)
- Motivations for this new development:
  - The Event Tag DB system was designed and developed long ago.
    - Even before I started as ATLAS Computing Coordinator in 2003!
  - Up to now it reached a level of implementation that supports some of the original use cases but not yet all of them.
  - The current system needed a lot of work and capital investment (Oracle storage) to be set up and still has not reached maturity
    - i.e. full usability by its primary customers: physics groups and users
  - Completely automatic system operation is also not fully achieved

EventIndex Design

- "Something like the TAGs" is needed by ATLAS and the need will increase with increased statistics
- Oracle storage seems not to be the best choice for this kind of information
  - Schemas with no flexibility, cost issues
- Modern structured storage technologies are much cheaper, easier to use and faster for sparse searches over large amounts of data
  - CERN set up a Hadoop service following the Database TEG (Technical Evolution Group) report in Spring 2012

➡ Design the data store using Hadoop and its tools

- Use cases:
  - Event picking
    - Give me the reference (pointer) to "this" event in "that" format for a given processing cycle
  - Production consistency checks
    - Technical checks that processing cycles are complete
  - Event service
    - Give me the references for this list of events (to be distributed to HPC or cloud clusters for processing)
    - Technically the same as event picking

EventIndex Project Breakdown

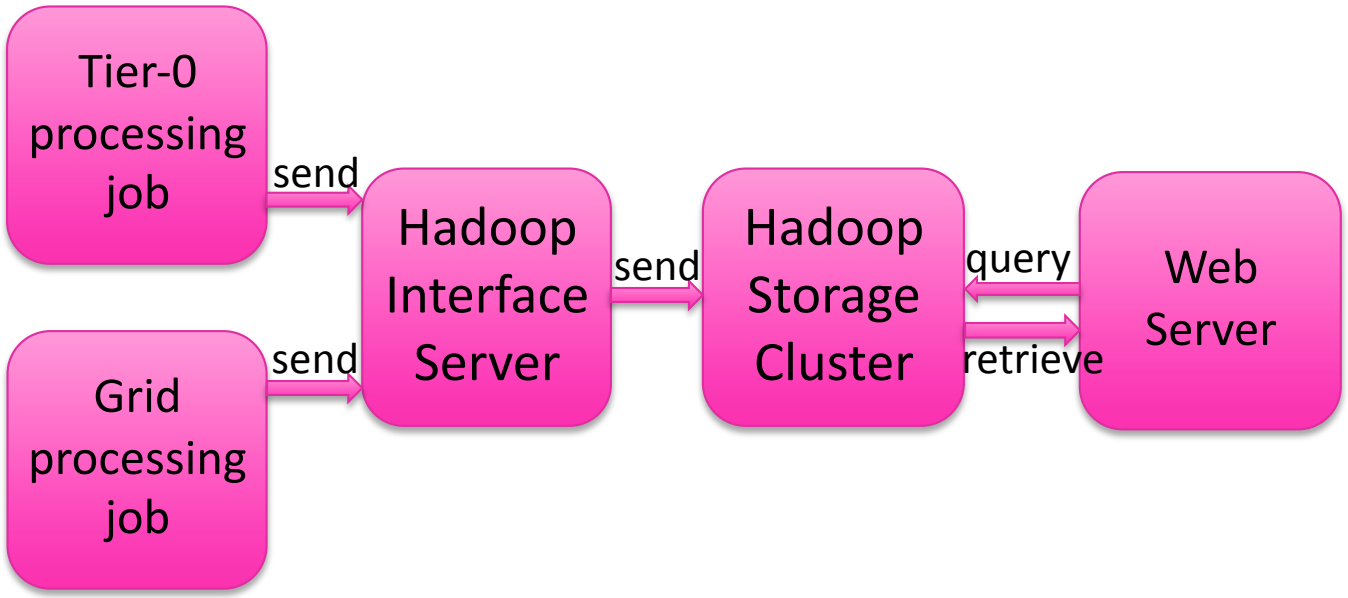
- We defined 4 major work areas (or tasks):

  1) Core system (LAL Orsay, UTFSM Valparaiso, CERN)

  2) Data collection and storage (IFIC Valencia)

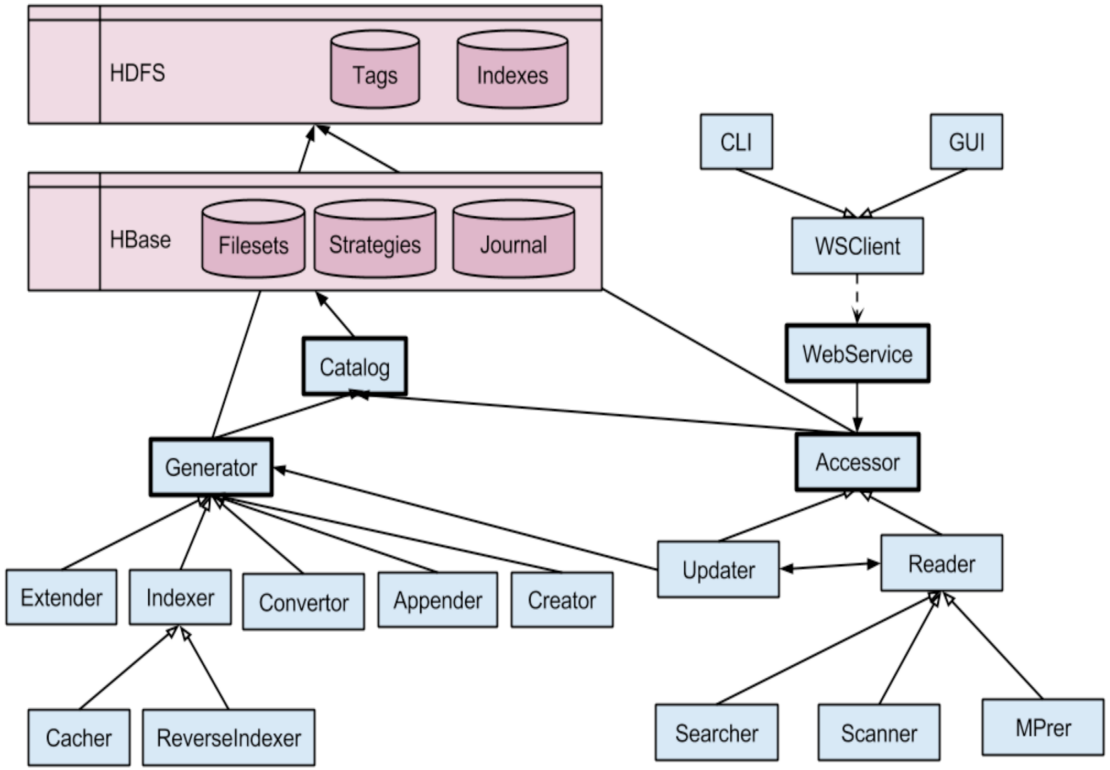
  3) Query services (LAL Orsay)

  4) Functional testing and monitoring (Genova, UAM Madrid, UTFSM Valparaiso)



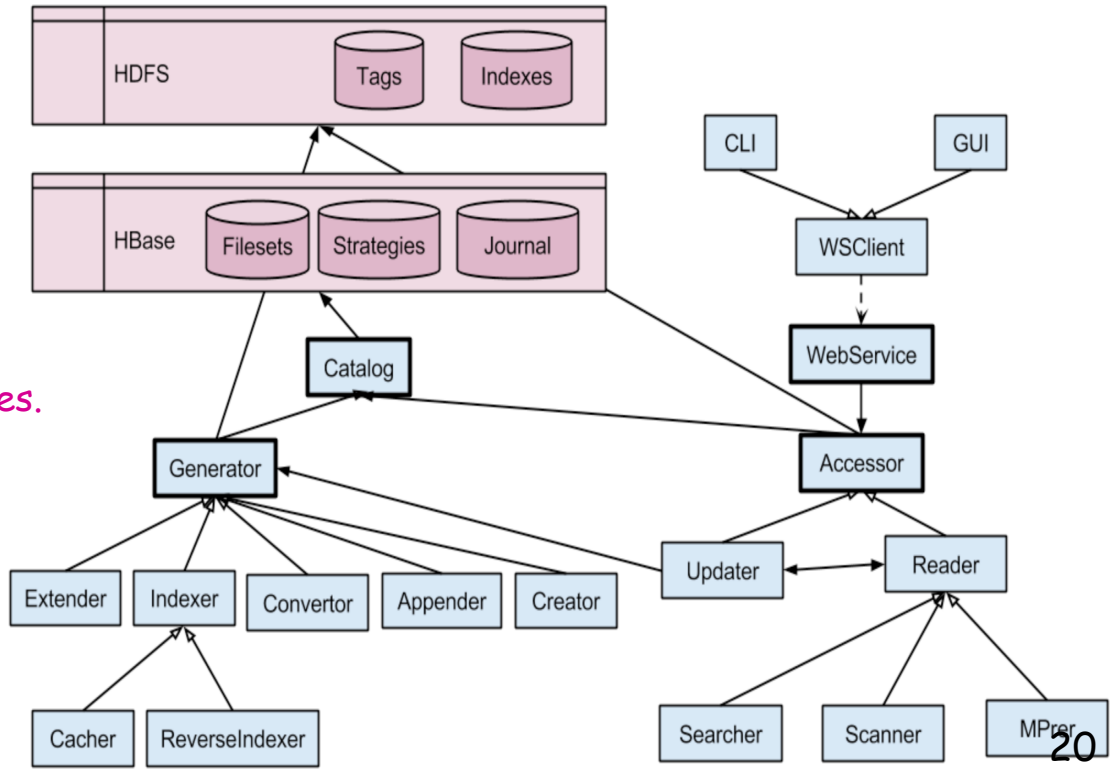
EI Task 1: Core System (1)

- Design and prototyping (till end 2013), then development and deployment of the core infrastructure to implement the EventIndex using Hadoop technology.
- Got access initially to 2 test Hadoop clusters in CERN-IT-DB and CERN-IT-DSS groups; now new (larger) cluster managed by CERN-IT-DSS.
- Uploaded ~1 TB of data from TAGs (full 2011 Tier-0 processing) for initial tests in different formats:
  - Plain CSV files
  - HBase (Hadoop's DB format)
  - Different binary formats
- After several tests, the baseline solution was chosen:
  - MapFiles with the data
  - Indexed in HBase
- Searches can be then performed in two steps:
  1. get reference from the index file
  2. using that reference, get the data
- Searches can be performed using keys, with immediate results (as keys are in memory), or with full searches on file-based data.



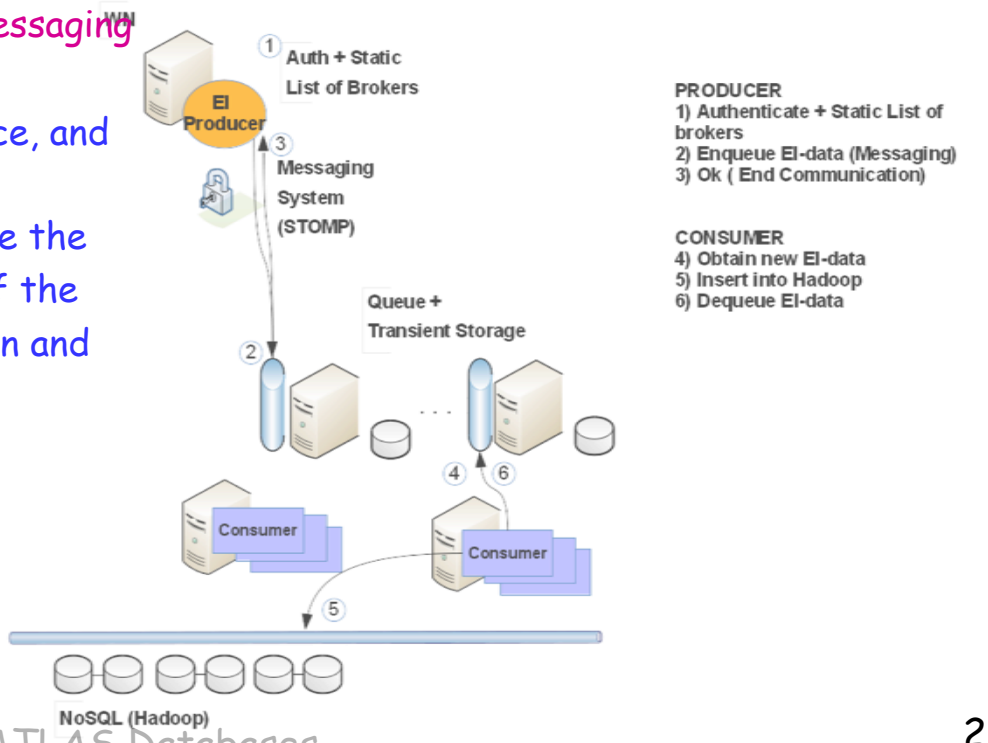
EI Task 1: Core System (2)

- Access to the data is achieved by a single and simple interface for:
  - upload: copy file into HDFS and create basic structure
  - get and search
  - adding new (vertical) data (in general by a MapReduce job)
  - create new indices
- These interfaces are implemented in Hadoop by Java classes with remote access and used through a Python client.
- Internal organization of the core system:
  - The Generator module creates new filesets and registers them into the Catalog.
  - The Creator creates new filesets.
  - The Extender adds information (new columns).
  - The Appender adds new data.
  - The Indexer creates new indices.
  - The Convertor converts filesets to different formats, including the MapFiles.
  - The Accessor allows remote access, through an HTTP server on a dedicated machine.
  - The Reader performs the search using one of the available tools (Searcher, Scanner or MapReducer), depending on the query.



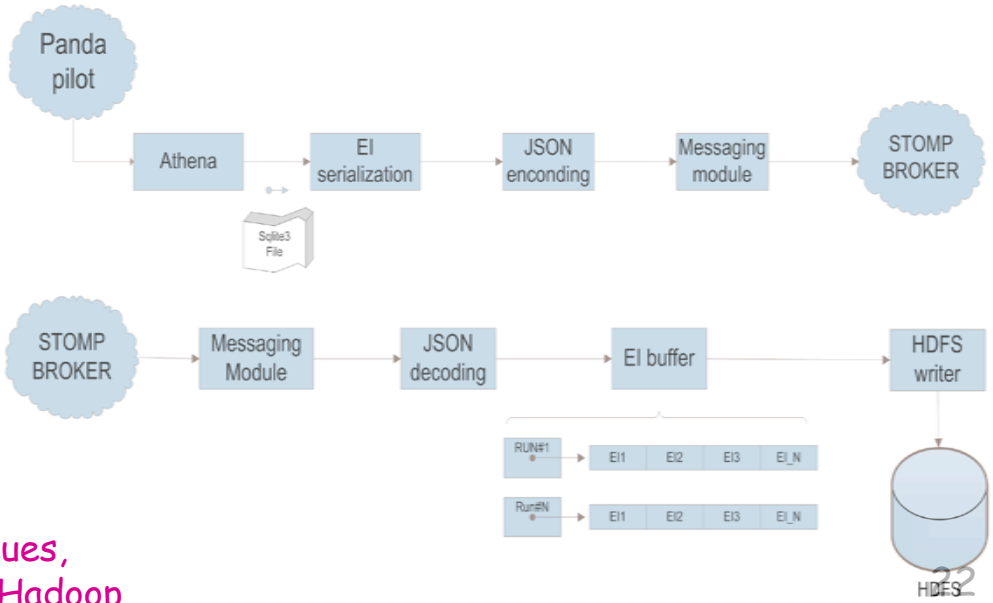
EI Task 2: Data Collection (1)

- Data to be stored in the EventIndex are produced by all production jobs that run at CERN or on the Grid.
- For every permanent output file, a snippet of information, containing the file unique identifier and for each event the relevant attributes, has to be sent to the central server at CERN.
  - The system should be ready to accept over 80 Hz of file records and insert into the back-end over 30 kHz of event records (plus contingency).
- The architecture is based on Producers on the worker nodes where event data are processed, a messaging system to transfer this information and asynchronous Consumers that effectively insert the data in the Hadoop database.
  - The producers run within the "pilot" process on each worker node and transmit the data using a messaging system to a queue in one of the endpoints.
- ActiveMQ has been chosen as the messaging service, and STOMP as the message protocol.
- An ssl endpoint at the broker can be used to secure the communications, and we can also have replication of the message queues in other sites, to permit duplication and high availability.
  - The consumers are asynchronous processes, continuously reading data.
- The task of the consumers is to get new data, check their validity with the production system, and pass them into Hadoop storage and related cataloguing tasks.



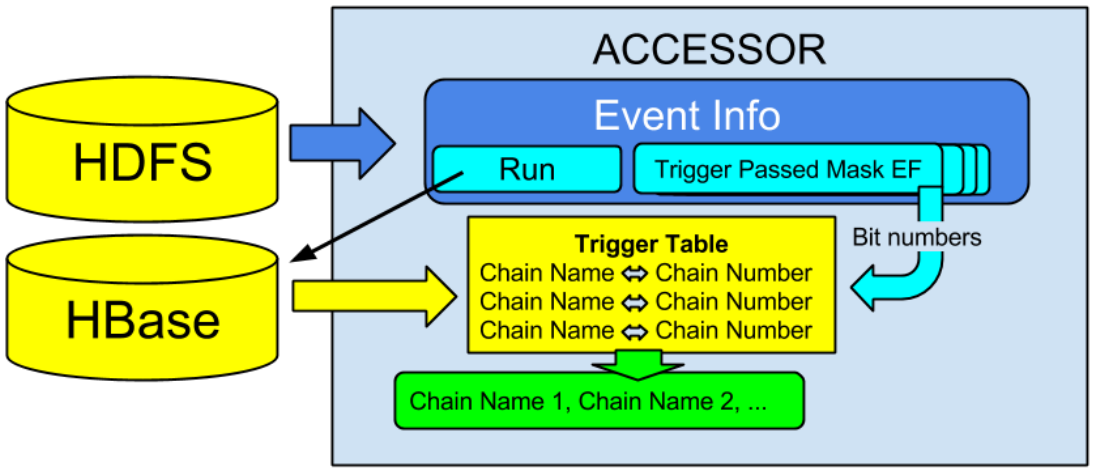
EI Task 2: Data Collection (2)

- The data producer process is split in two parts:
  - the first part reads the data file to create an intermediate EventIndex file;
  - the second one reads information from the EventIndex file, builds the messages and sends them to the broker.
- The EventIndex file is used as intermediate storage to decouple the processing of the input file from sending the messages.
  - In the current implementation, the first step consists of a Python script that reads the event file and extracts the relevant information to be transmitted (event identification, trigger masks, references to the files containing the event).
  - The EventIndex file is a SQLite3 file of Python serialized objects, containing key-value pairs in one table "shelf".
- The second step is a Python script that reads the EventIndex file, builds a sequence of messages (about 10 kB each) and sends them to the broker.
  - The messages are sent in a compact format encoded using JSON.
  - Tests show that it is possible with a single broker to achieve the required data transfer rates, with peaks in the load of 200 kHz of event records, and about 60 kB/s.
- If an adequate number of consumer processes is active, thepayload is retrieved immediately and no backlog is created
- The consumers:
  - receive the messages,
  - decode the information in JSON format,
  - order the events by run number in memory queues,
  - build CSV records and append them to files in Hadoop



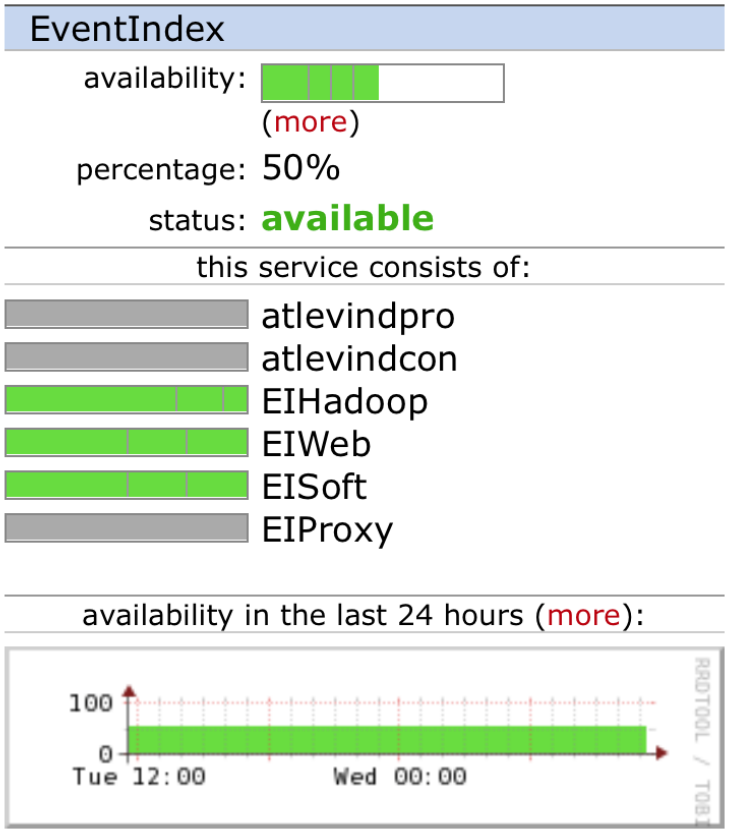
Task 3: Query Services

- They are adaptations and/or evolutions of the web services and APIs used to query the Tag DB to the EventIndex using Hadoop technology
- Simple data scan operations on the test dataset (1 TB of 2011 data) using the available Hadoop cluster of 20 nodes take about 15 minutes
  - We consider this as the worst case, as normally no query would need to read in all data from disk for a full year of data taking.
- Queries that give the run number and event number (the event picking use case) and use the so-far not-so-much optimised "accessor" method return their result in 3.7 seconds
  - For comparison, the same query using a full Map-Reduce job returns in 90 seconds.
- The results of queries are stored and made available for subsequent queries, that are therefore much faster
- An important use case is the query (or count) of the events that satisfied a given trigger condition, or a combination of trigger conditions.
  - Queries to the EventIndex involving trigger chains need to be first decoded by knowing the trigger menu via the run number and finding in the COMA database the corresponding trigger info for each bit in the trigger words. The COMA database is stored in Oracle and accessed using ODBC drivers; the retrieved information is then used by MapReduce jobs in the Hadoop cluster.



Task 4: Functional Testing & Monitoring

- Performance tests were done throughout the design and development phase, comparing different design options
  - Reported in the previous slides
- Next step is the large-scale upload of all Run1 date (starting from real data, but simulation too) to be done to commission the system end-to-end



- System monitoring is also in place, recording the health of all servers along the chain and data rates
  - Unfortunately SLS at CERN is being replaced with a system with lower functionality so we need to do extra work

EventIndex Outlook

- The EventIndex project is on track to provide ATLAS with a global event catalogue in advance of the resumption of LHC operations in 2015.
- The global architecture is defined and testing work is in progress; results obtained so far are encouraging.
- Next steps consist of the full implementation of the final "first operation version", loaded with Run1 data, during the next 2 months, and the completion of deployment and operation infrastructure by the start of 2015.

NB: the EventIndex is not designed only for ATLAS. Its infrastructure can be used for any other scientific application that has a lot of data with very small granularity. The details of the implementation of course are specific to ATLAS. In fact the funding we got in Italy is for a generic catalogue for BigData that can be structured as a large number of small and logically equivalent units.

Closing Remarks

- The Database group in ATLAS is small but runs a number of operation and development activities
- Diversification is the word of the day: we should use the technologies that best fit the applications we have
  - Some applications are well matched to relational SQL databases
  - Others are more data-intensive and need a flexible data store and search engine rather than fixed schemas and transaction management
- The EventIndex and the new proposed Physics Conditions Data Web Service are examples of "small" projects that can go from ideas to development, deployment and operations in 2-3 years with a reduced number of people
  - Using of course the technologies that match the problem and the knowledge of the developers!