Conditions Database for Run3

Proposal for a REST architecture

A.Formica on behalf of Atlas DB team

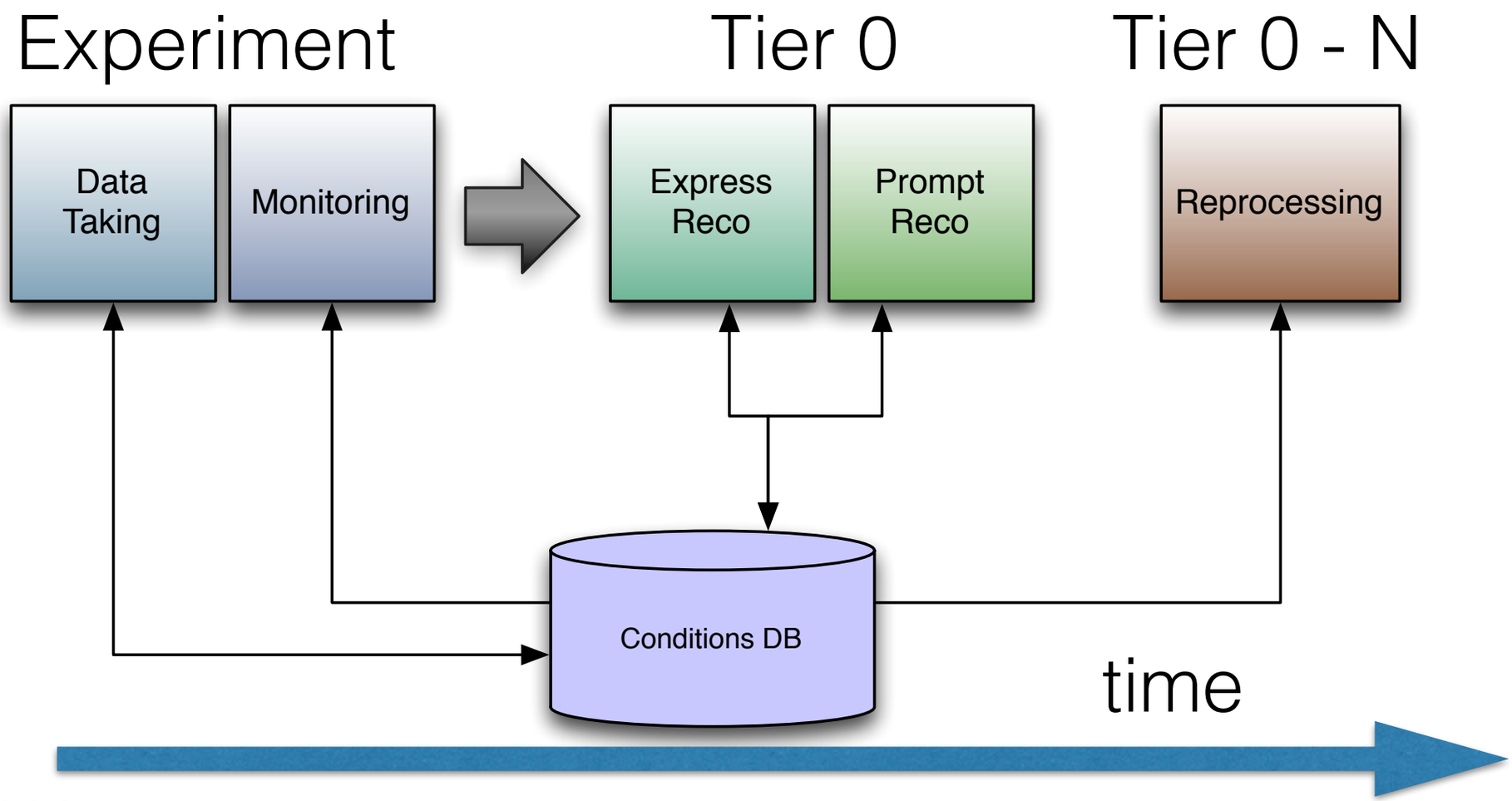
Topics

- Conditions data
  - usage in Atlas data flows
- Database schema
  - based on CMS Run2 experience
- Architecture to manage Conditions data
  - REST services for conditions
- Link to previous presentations….
  - Tim workshop in Genova, https://indico.cern.ch/event/342881/session/9/contribution/21/attachments/1159952/1669544/Report_from_DB_TIM.pdf
  - CHEP 2015, http://indico.cern.ch/event/304944/session/3/contribution/5/material/slides/0.pdf

Conditions data

- A definition attempt
  - in general they are non-event data varying with time
  - a specific subset is critical for the data flow
    - Status and Configuration for detectors and Trigger
    - Run information
    - Detector control system: HV, LV, ….
    - Detector calibrations (calibrations, alignments, …)
    - Beam and Luminosity information

Dataflows

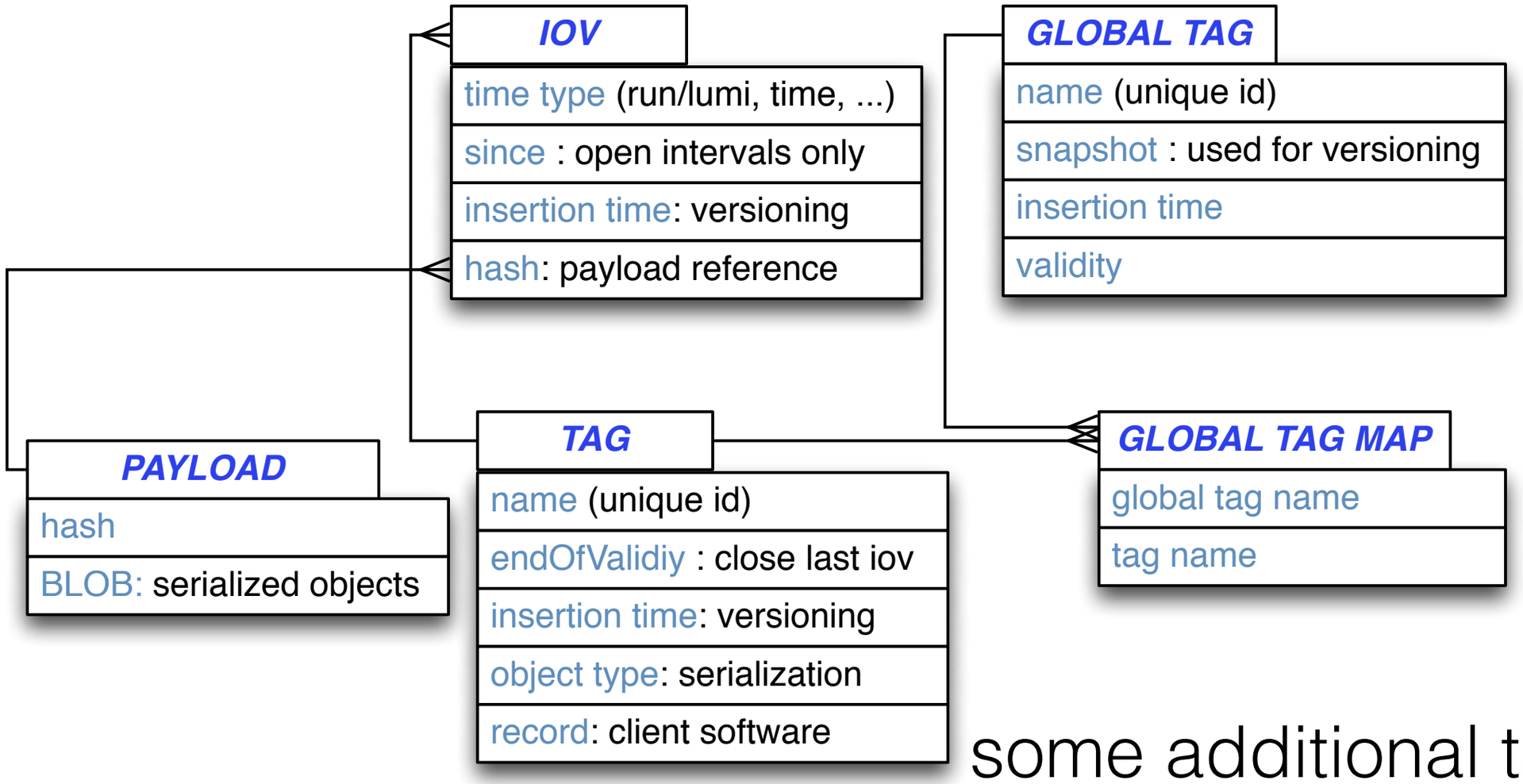


- usage
  - conditions data are used and produced at different steps of our dataflows
  - calibrations are refined in the express and prompt reconstruction
  - in general data reprocessing should benefit of the best knowledge we have on our calibrations

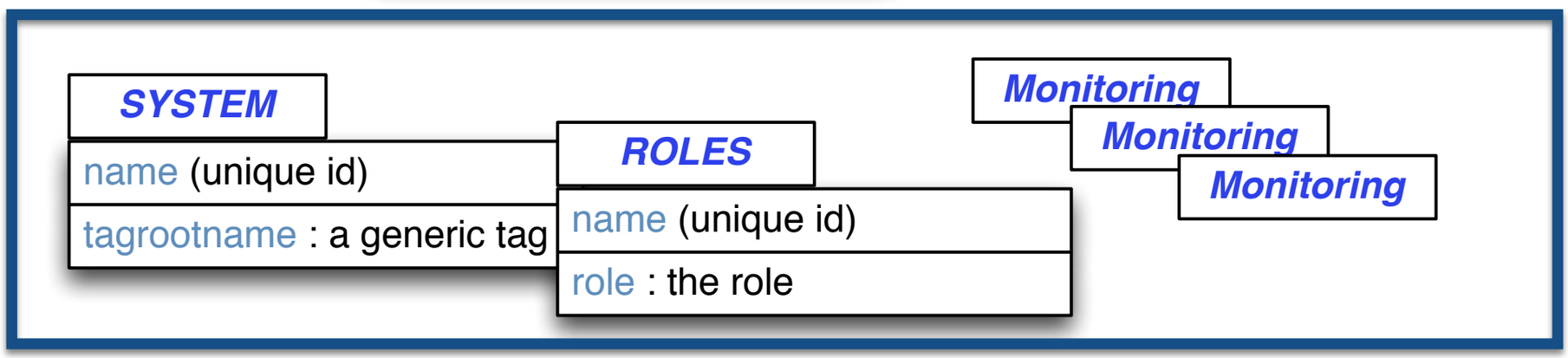
the Conditions DB : intro

- A common project (ATLAS / CMS)
  - CMS has developed during LS1 a new database schema trying to simplify a lot the structure respect to Run1: this schema is today in production and used for Run2
  - ATLAS has shown interest in this project, since the main concepts that CMS has been using for this new DB architecture are very similar to those who have been identified as useful from our Run1 experience, but the changes in core software needed to implement the new schema and adopt it are pretty big: so we aim for Run3 !!
  - Other developments have been triggered by this collaboration, which are more related to the data management and access
- Data model
  - we will enter in some details at the level of the DB structure, but here I think that further discussions should be more profitable if our CMS collegues are present: they have a production system and they can justify or provide feedback on the choices that were made
- Towards a REST architecture
  - we will enter in some details about the architecture for the conditions data management: this is an going development, and the choices have to be evaluated by DB coordination in both experiments

Data model



some additional tables (TBD)



Data model in words

- Conditions data
  - **Payload** : values are consumed as an aggregated set, typically an header and some parameter container (s)
- Conditions metadata
  - **IOV**: *time* information, based on 1 time column (time, run number, …), valid until the next entry in *time*
    an IOV point to 1 payload (via an sha256 hash)
  - **Tag**: label to identify a specific set of IOVs
  - **Global Tag**: consistent set of tags, involved in a given data-flow. A tag can be associated to many global tags.

Payload

- A binary large object
  - the payload is stored as a BLOB inside the DB, leaving to the users the choice of the implementation
  - metadata present in the Tag table allow to deserialize the BLOB in the correct way
- no generic implementation
  - CMS has chosen for several reason a Boost library to serialize and deserialize the DB content
  - other choices are possible (ROOT, HDF5, …), some of them available for multiple languages
- remarks (suggestions?)
  - the choice is a compromise between optimisation and flexibility
  - multi-language support could be useful
  - avoid too many serialization formats…

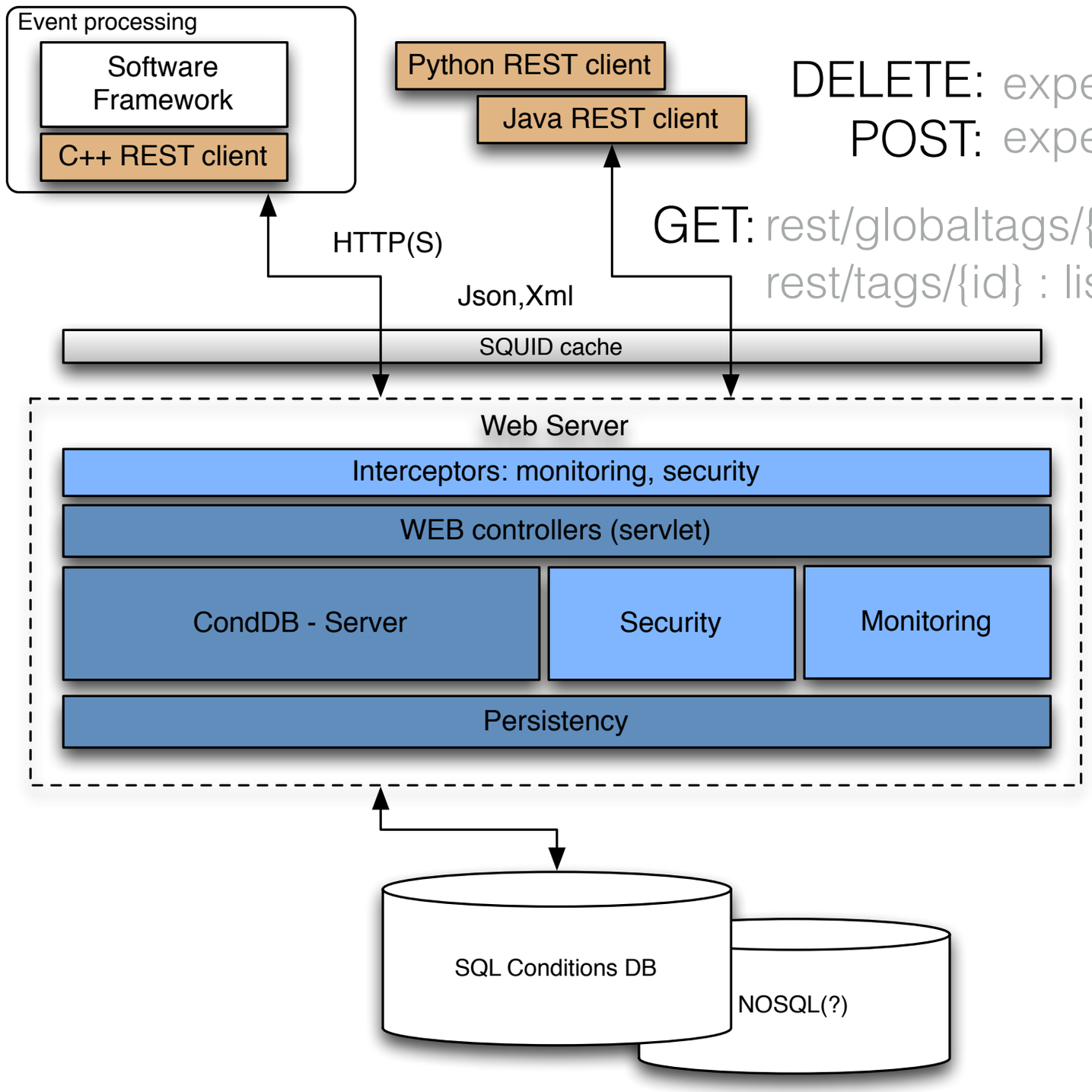
conditions data management

- Lessons from the past
  - in our data-flows, we write occasionally but we read a lot, and from a large number of clients
  - client-server model could fit for writing but not for reading; in the distributed computing context we are using an intermediate server to access the data (Frontier)
  - bounding the API to a couple of languages we limit the usage from different clients
- DB technology
  - today we are happy about Oracle (mainly thanks to the support we have at CERN)
  - we should in any case support multiple DB platforms

a REST architecture

- Multi-tier model for conditions data management
  - enforcing the role of the middle tier for conditions management would allow us to isolate the client from any complicated software strictly DB - related
  - simplification client software allow as well an easier management of the client libraries and queries optimisation are less intrusive (no need to release a new client version)
- something pretty common nowadays
  - a lot of the daily interaction with external services is done today via REST architectures
  - we already use a sort of REST architecture (or at least we use HTTP) to access conditions in both experiments

architecture for Run3



DELETE: expert/tags/{id} : delete tag <id>
POST: expert/tags {tag-obj} : create a tag

GET: rest/globaltags/{id}?trace=on : trace global tag <id>
rest/tags/{id} : list info on tag <id>

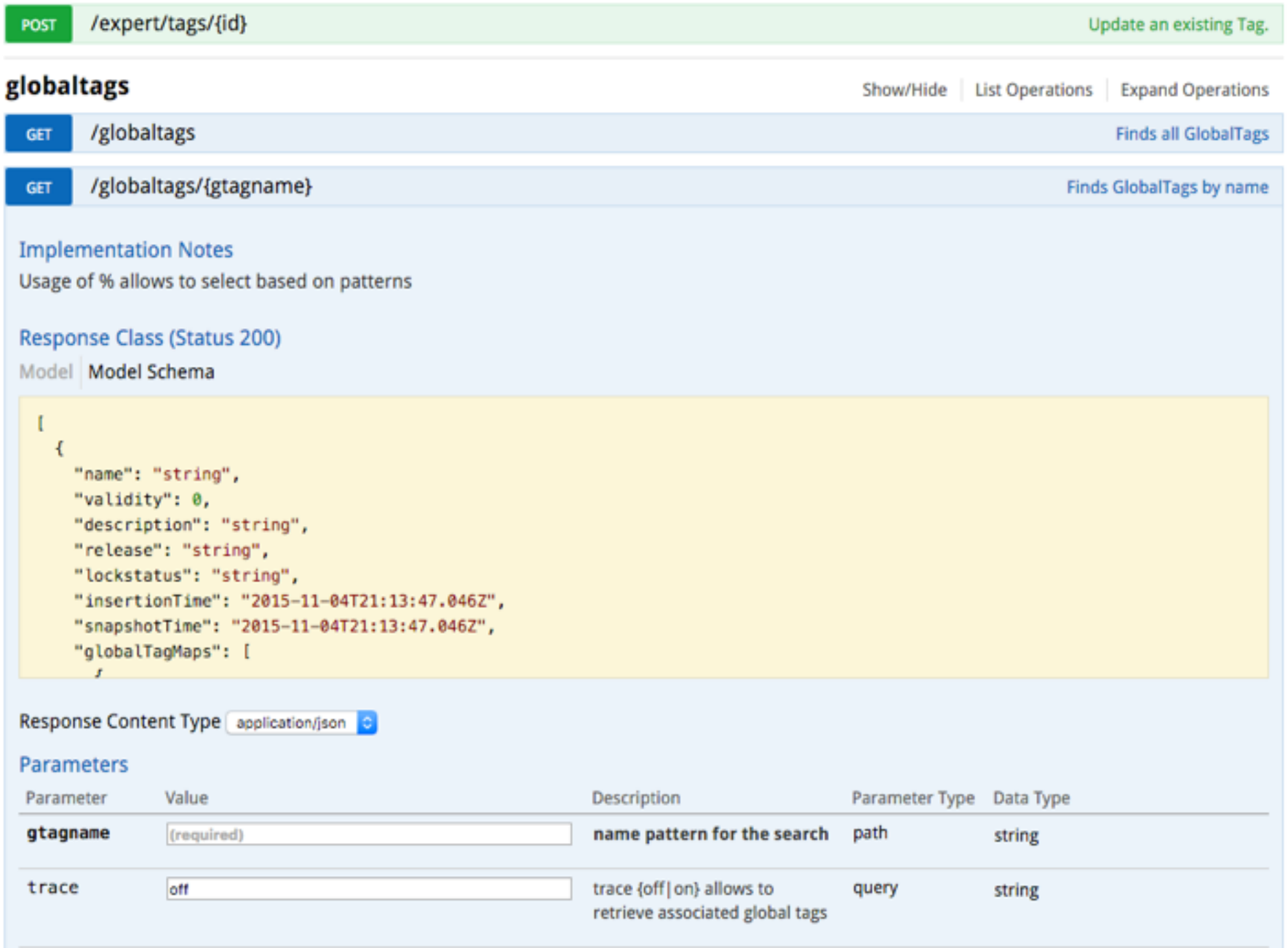
Remarks:

use standard libraries for persistency

multiple DB support portable format in requests (Json)

follow REST design in URLs

prototyping



- doc http://aiatlas062.cern.ch:8080/swagger/
- code https://gitlab.cern.ch/formica/PhysCondDB (branch: swagger-tests)

- inside GPN
- Initial documentation for testing
  - swagger for (almost) automatic generation
- python and C++ clients
  - clients development is on going (Grigory Rybkin)

on going

- ATLAS
  - follow up the developments needed in code software
  - **calibration for analysis:** deploy a prototype in the context of calibration files related to end users analysis
  - C++ client development for the REST services
  - investigate Payload serialization solutions
- CMS (TBC)
  - **starting next year:** give feedback on the prototype
  - use the present data to study the performances and participate to the developments

spare slides

- Some numbers from Run1
- Producing conditions
  - lot of similarities in ATLAS and CMS, even if a completely different implementation is used
- Consuming conditions
  - here I will mention how the new architecture works…but since this is for CMS, forgive any incompleteness: Giacomo Govi and Andreas Pfeiffer are the real experts !

Data volume in Run 1

- ATLAS & CMS
  - about 1.5 TB of conditions data gathered in Run1
- ATLAS
  - Most data (about half of the whole volume) are coming from DCS (HV, LV, temperatures,..) and they are time based by nature (in reality these are timeseries) and not versioned
  - Usually calibrations-like conditions are instead versioned and run/lumi-block based
  - IOVs can vary a lot, from ~10 to 100K (e.g. luminosity)
  - Payload size is also very variable: ~few bytes to ~O(10Mb)

Producing conditions

- Reprocessing data-flow
  - In general here the conditions are frozen, no update is performed at this level
  - Systems try to deliver the best conditions “before”
- Express and prompt reconstruction
  - these steps happens in a range from hours to ~1day from data taking, and its the place where we update more often the conditions
  - IOVs are in general appended, cannot overwrite past
  - Protection mechanisms are in place to guarantee that we are always capable to reproduce results (by using the same conditions)

consuming conditions

- Workflows run with a specific Global Tag
  - resolve the full list of associated Tags
- A Tag is associated to a Payload type and an IOV sequence
  - load IOV “pages” (e.g.: run2000, run2100, run2200,…)
    - this is needed for caching at the level of the Frontier server !!
    - the query needs to be reproducible
  - the IOVs in 1 page are loaded once the first event is known:
    - load the full IOV sequence for that page in every Tag
    - cache the page until a new one is required
- Payloads are loaded “on demand”
  - only when the job needs a payload this one is loaded
  - payloads are loaded via a hash and cached at the level of Frontier server