Conditions Database:
Experience from LHC and Belle II
(and how I can finally sleep at night)

Paul Laycock

BROOKHAVEN NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

My experience with Conditions Database Management

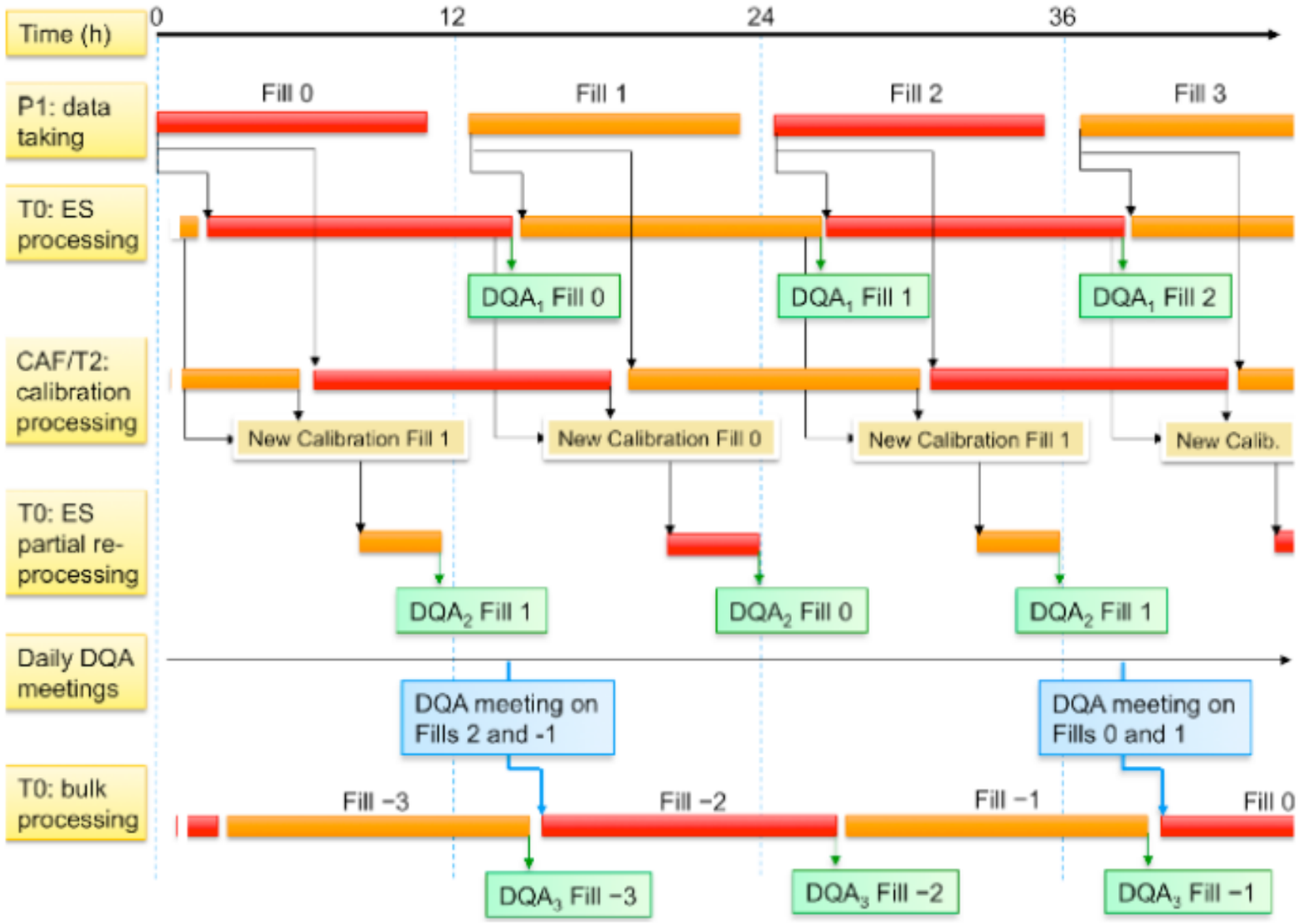
- I worked on ATLAS conditions management for nearly a decade
  - I started as ATLAS Conditions Coordinator in 2008-2012
  - ATLAS Data Preparation Coordinator 2014-2016
  - Worked on upgrading Conditions DB design to “best practice”

- I gathered some experts for the HSF Community White Paper (contributions mainly from LHC+Belle II)
  - We converged on a definition of best practice:
  - https://arxiv.org/abs/1901.05429

- I’m now responsible for US Software and Computing lead for Belle II
  - The Belle II Conditions DB design is close to the HSF best practice
  - Responsible for migrating to Rucio data management

- and I’m BNL’s S&C technical contact for DUNE

Conditions management



- Data will be processed several times, (many times early on) - organising the conditions to be used in offline processing can be a real bottleneck to delivering quality physics

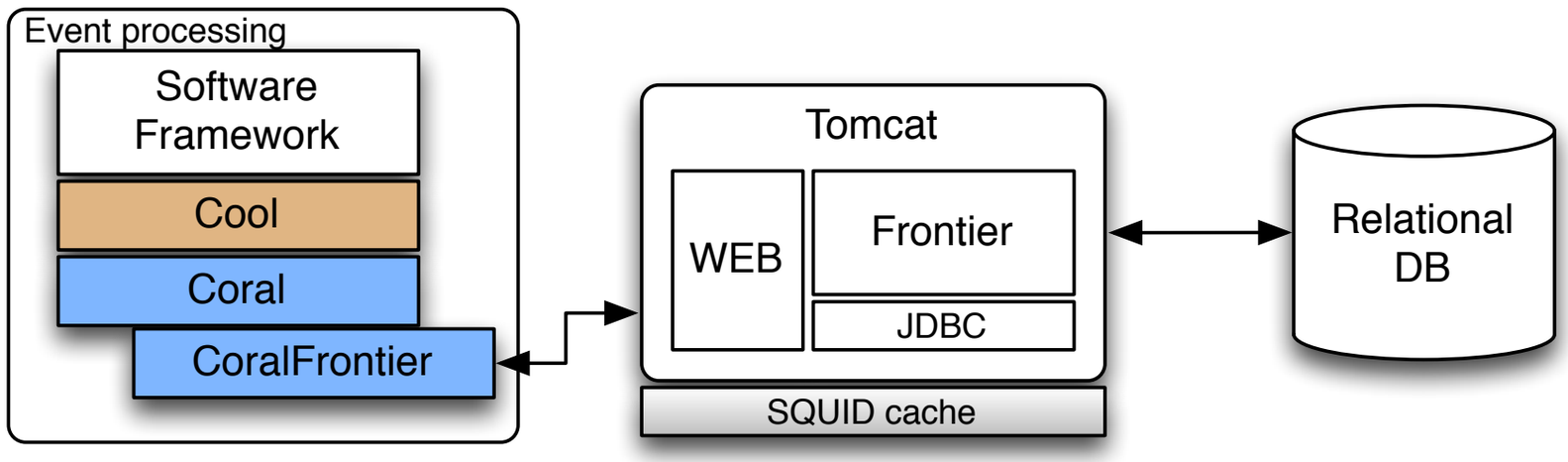
Conditions interfaces for ATLAS

- Slow control measurements and similar that need to be refined (bigger IOVs) before being injected into Conditions DB - some were missed
  - Components still got missed and that broke overlay (and still does)
- Automated calibrations wrote to the Conditions DB with appropriate granularity
- Trigger and DAQ wrote configurations to the Conditions DB
- Expert calibrations wrote directly to the same Conditions DB
- Online (HLT) used an independent instance of Conditions DB (different content)
- … not exhaustive, we need to define these for DUNE of course…

- **<u>Software framework view:</u>**
- Global tag for configuration: **GlobalTag = “FinalFinalBestCalibration2040”**
  - resolves to payload-type tags for every payload used in reco
- Reco module asks a ConditionsService for payload-type and gives a timestamp
  - **cdbSvc->Get(“MyCalibrationType”, Run_number)**

ATLAS Conditions DB Design

- Schematic of current ATLAS infrastructure, based on COOL



- COOL was an LCG project: http://lcgapp.cern.ch/project/CondDB/COOL_2_1_0/
- ***ATLAS used it for ALL use cases***, and consequently it became ***very complicated***
- COOL designed to work with multiple backends, but the ATLAS solution ended up tightly coupled to the s/w framework
  - ***Complicated schema defined, and then expertise disappeared quickly***
- Much DBA effort needed to make oracle queries performant and performance really relies on a powerful oracle backend, much s/w framework service expertise needed to make service performant
- Frontier caching layer absolutely required to avoid making (complicated) queries to the DB, which in turn made Frontier complicated

Current problems

- COOL has been successfully used in Runs 1 and 2
  - But it only performs well for offline because we introduced Frontier

- We are tied to COOL, and its support is diminishing
  - By Run 3 we would be the only experiment using it

- We made very diverse requirements on COOL and we use them
  - Huge diversity of (undocumented) payload formats
  - Compare to CMS - only serialised C++, all in one package

- We have thousands of tables
  - One schema per system, online and offline, data and MC, all of the tables required to support COOL design, problematic to manage so many tables

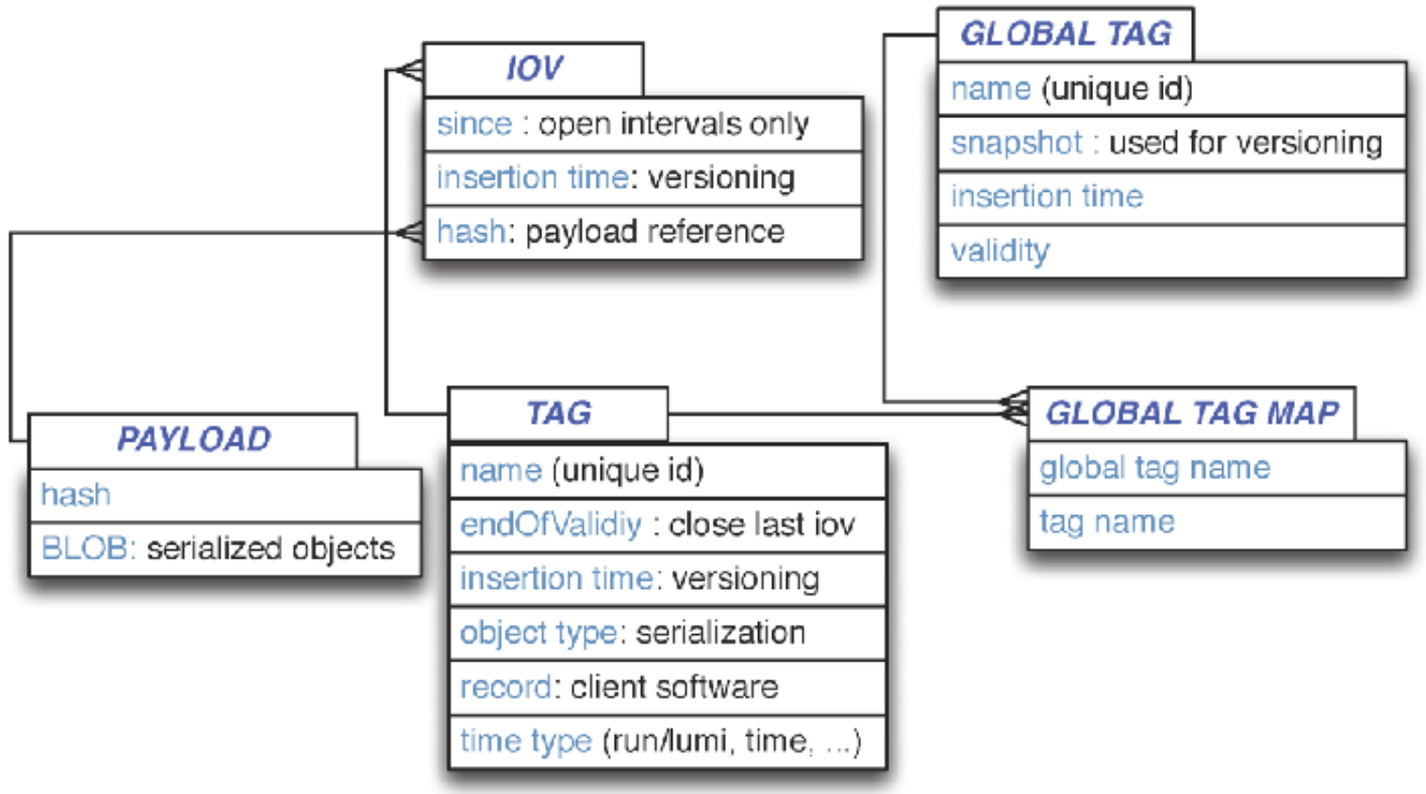
Current problems (2)

- COOL doesn't do everything
  - Global tags and UPD protection means lots of custom (AtlCool) tools
- We constrain detector monitoring granularity with offline requirements
  - DCS data are written to oracle and then copied to COOL for offline use
- By design, COOL isn't built for cacheable queries
  - There are infinite ways of getting to the same payload
  - Retrieve IOVs and payloads at the same time
  - Lots of DBA expert time to tune oracle queries, plus IOVDbSvc optimisations to achieve good performance, but serious problems remain, especially Overlay
- We do not have many conditions experts…
  - …and yet we have the most complex conditions database, ATLAS will run for 20 more years

HSF/Belle II Conditions Design

- REST Interfaces
- Metadata Model: relational DB
- Payloads looks like noSQL
  - Addressed by (unique) hash
- Belle II separates these

https://belle2db.sdcc.bnl.gov/b2s/rest/v2/iovs/?gtName=B2BII&runNumber=6

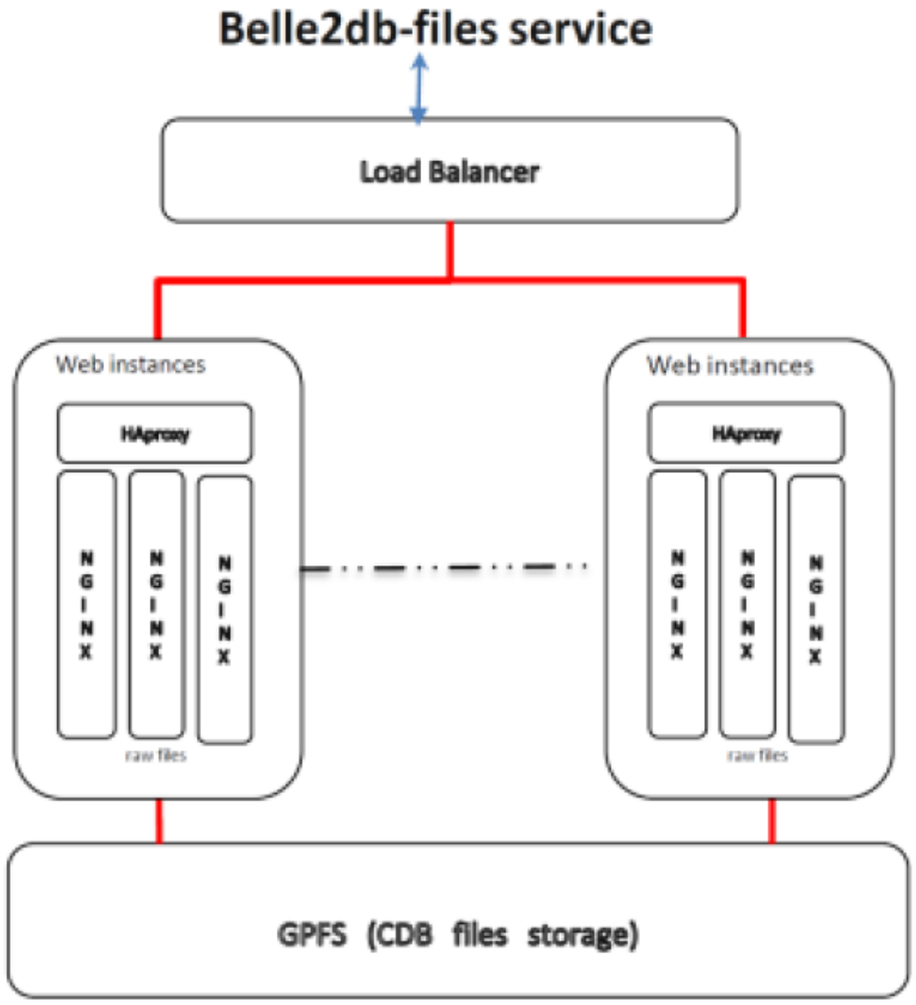


- Single tables for payload, tags, IOVs
  - ***Payloads factorised from metadata***
  - ***No need to define a schema per subsystem, it's just a BLOB***

- IOVs and payloads resolved independently:
  - Only use start time - cachable IOV queries (1st query)
  - Uniquely identified payloads - minimal use of cache
  - ***Cache-friendly design***

Conditions for HPC and Analysis

- For Belle II, the ***same conditions database service*** is used for user ***analysis conditions***
  - Encourages good practice for conditions book-keeping
  - Discourages copy-and-paste from twiki/afs/bloke-down-pub



- Failover strategy ***(local, cvmfs, REST service)*** means analysers can work offline after the first run

- HPCs with ***cvmfs*** are covered, otherwise may think of a ***DB snapshot*** (when conditions are well known and stable) or an edge service for more flexibility

16

- A design where you factor out the path-to-file can be a natural fit to HPC