Conditions Database

Ana Paula Vizcaya Hernandez, Lino Gerlach, Norm Buchanan and Paul Laycock (blame me!)

US DUNE Computing Consortium meeting - August 31st, 2022

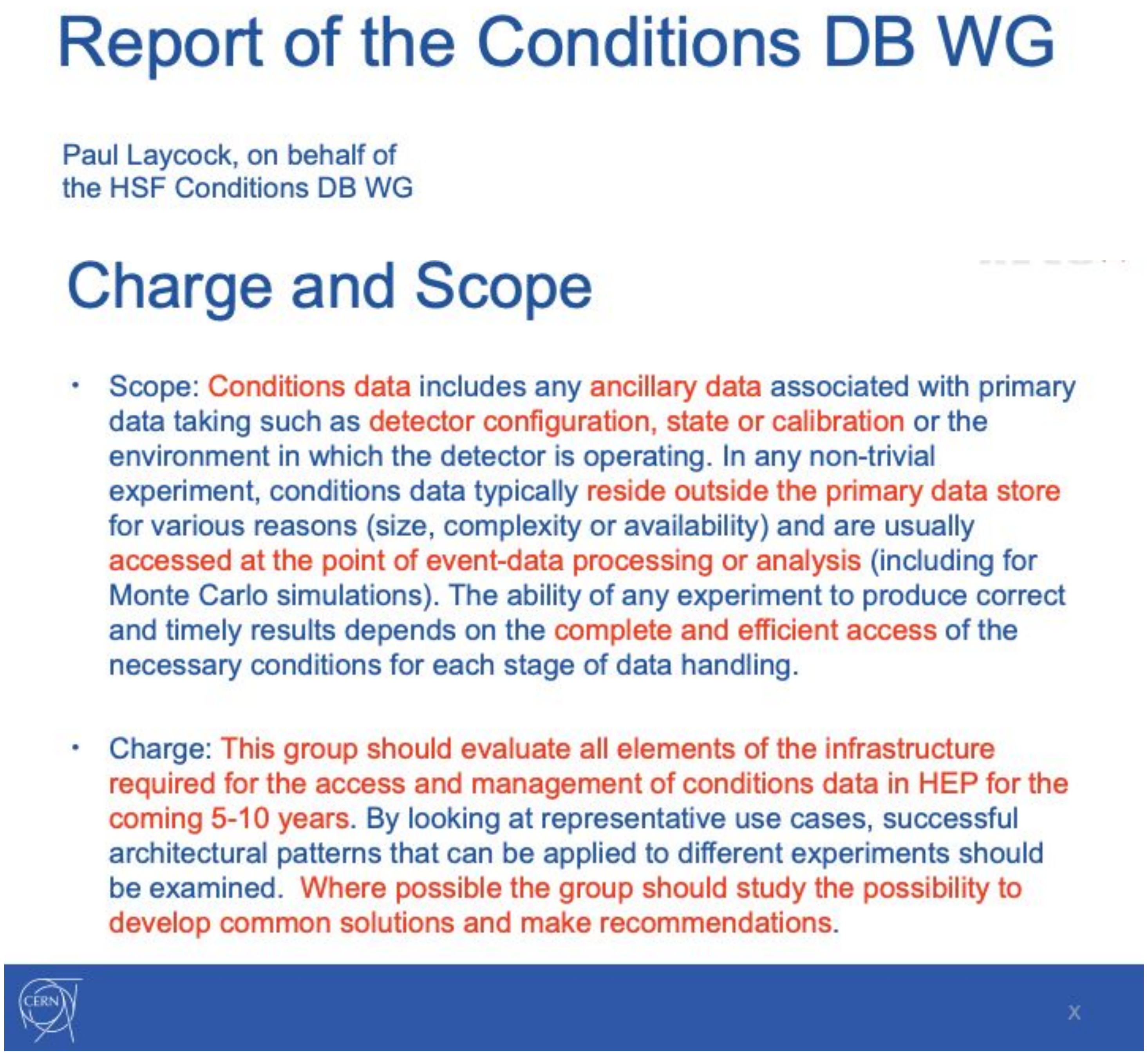
@BrookhavenLab

HSF History

In 2017 we wrote the HSF Community White paper

I led the working group and we converged on the core ideas and best practice:

- High degree of separation between client and server, client-side is simple but takes care of (de)serialisation
- Conditions accessed via a REST interface
- Caching must be built in, good experience using web proxy technologies. Clients should be able to deal with multiple proxies and servers
- Relational DB for data model is preferred



Report of the Conditions DB WG

Paul Laycock, on behalf of
the HSF Conditions DB WG

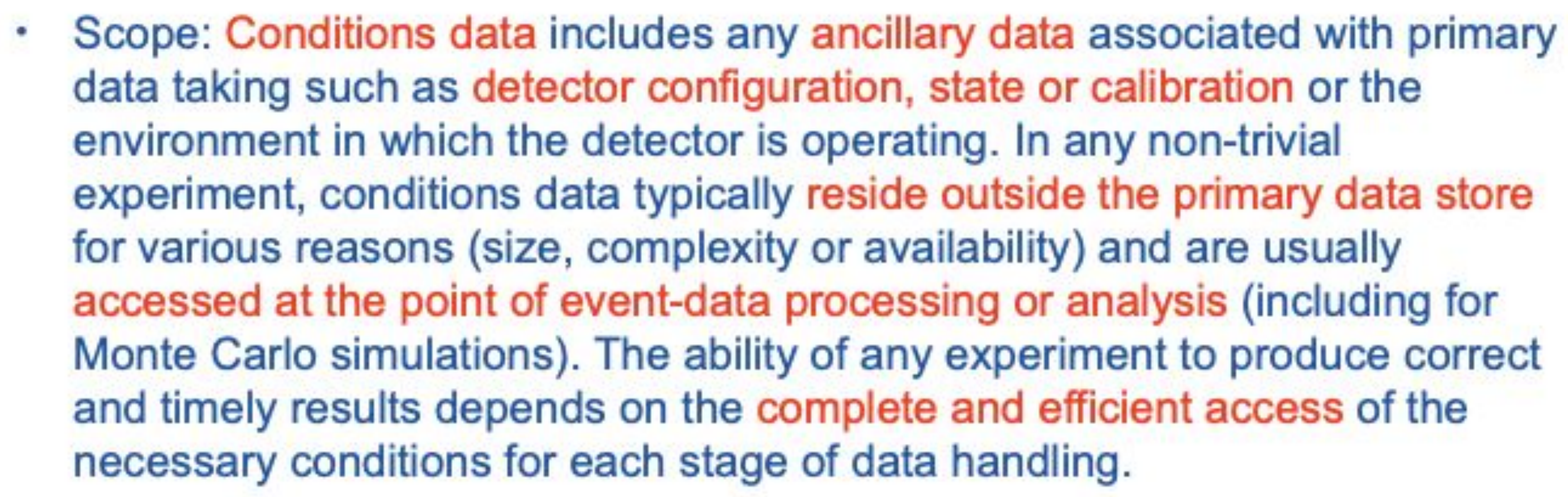
Charge and Scope

- Scope: Conditions data includes any ancillary data associated with primary data taking such as detector configuration, state or calibration or the environment in which the detector is operating. In any non-trivial experiment, conditions data typically reside outside the primary data store for various reasons (size, complexity or availability) and are usually accessed at the point of event-data processing or analysis (including for Monte Carlo simulations). The ability of any experiment to produce correct and timely results depends on the complete and efficient access of the necessary conditions for each stage of data handling.

- Charge: This group should evaluate all elements of the infrastructure required for the access and management of conditions data in HEP for the coming 5-10 years. By looking at representative use cases, successful architectural patterns that can be applied to different experiments should be examined. Where possible the group should study the possibility to develop common solutions and make recommendations.

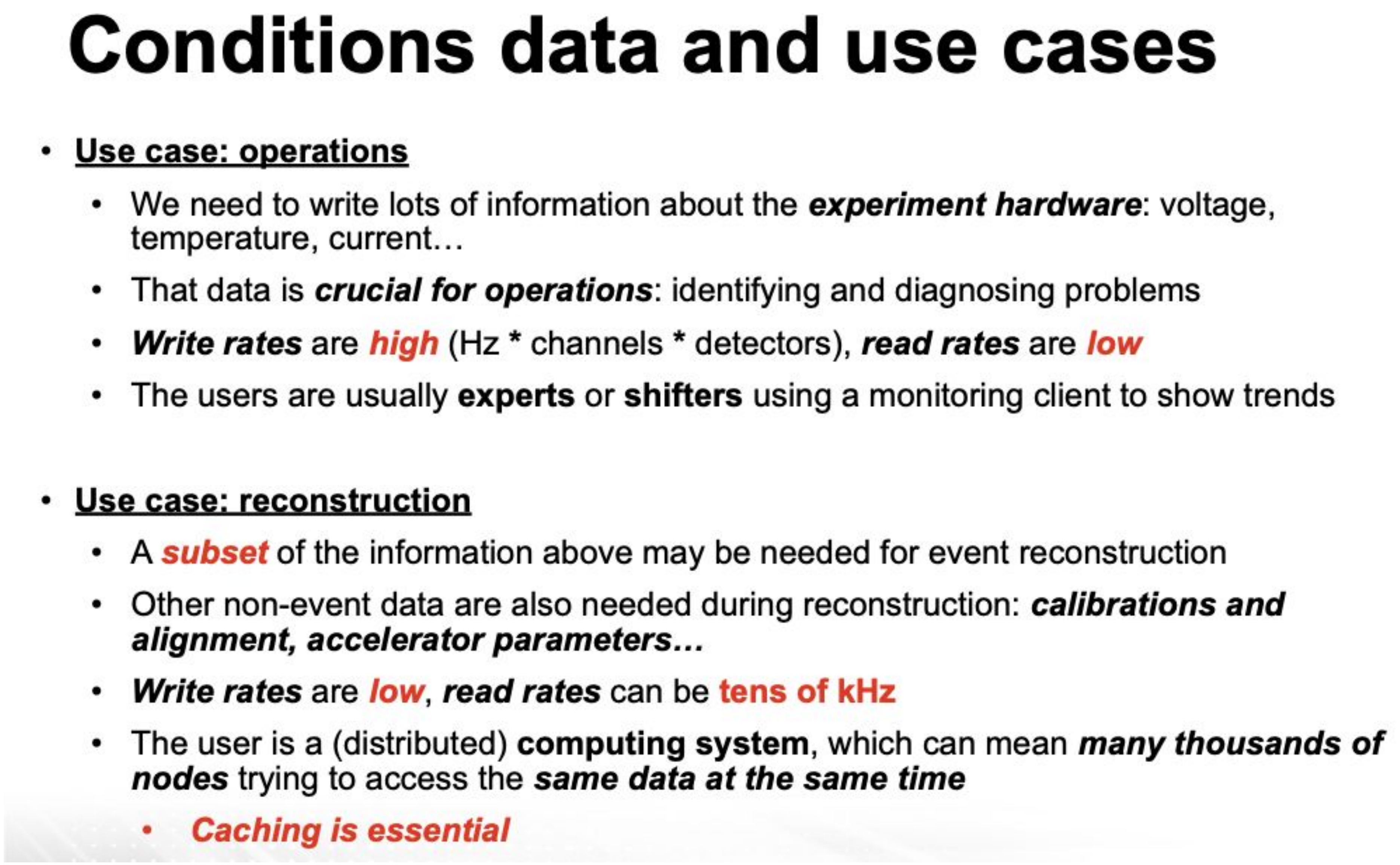
What are “conditions data”

This is not an unambiguous term! Review the HSF scope:



- Scope: Conditions data includes any ancillary data associated with primary data taking such as detector configuration, state or calibration or the environment in which the detector is operating. In any non-trivial experiment, conditions data typically reside outside the primary data store for various reasons (size, complexity or availability) and are usually accessed at the point of event-data processing or analysis (including for Monte Carlo simulations). The ability of any experiment to produce correct and timely results depends on the complete and efficient access of the necessary conditions for each stage of data handling.

In practice, in large experiments with distributed computing, it’s the “access at the point of event-data processing or analysis” that is VERY important. The solution to that has very particular requirements.



Conditions data and use cases

- **Use case: operations**
  - We need to write lots of information about the ***experiment hardware***: voltage, temperature, current…
  - That data is ***crucial for operations***: identifying and diagnosing problems
  - ***Write rates*** are ***high*** (Hz * channels * detectors), ***read rates*** are ***low***
  - The users are usually **experts** or **shifters** using a monitoring client to show trends

- **Use case: reconstruction**
  - A ***subset*** of the information above may be needed for event reconstruction
  - Other non-event data are also needed during reconstruction: ***calibrations and alignment, accelerator parameters…***
  - ***Write rates*** are ***low***, ***read rates*** can be **tens of kHz**
  - The user is a (distributed) **computing system**, which can mean ***many thousands of nodes*** trying to access the ***same data at the same time***
    - ***Caching is essential***

Offline generally has very little influence on the online DB design

Brookhaven National Laboratory

Conditions data access from software frameworks

Most people don't have experience with, and therefore are uncomfortable working with, databases. Those creative spirits will go to great lengths to avoid interacting with a database. See e.g. the HSF Data Analysis WG's paper on Metadata: https://arxiv.org/abs/2203.00463.

We want an API that hides the database access. Just configure it, and then request your data using the minimal information possible

We would like to move to a run-index design. That requires some reformatting work, not for the DAQ case which is inherently Run-indexed, but for some of the ad hoc DRA DBs that already exist. The basic design idea is:

- Get your conditions data from ONE DB interface (not an ad hoc DB interface for each source)
- Access your data by Run number

```
CondSvc.setGlobalTag(<GlobalTagName>);
```

Once per job

```
CondSvc.get(<MyConditionsType>, <RunNumber>);
```

In each module

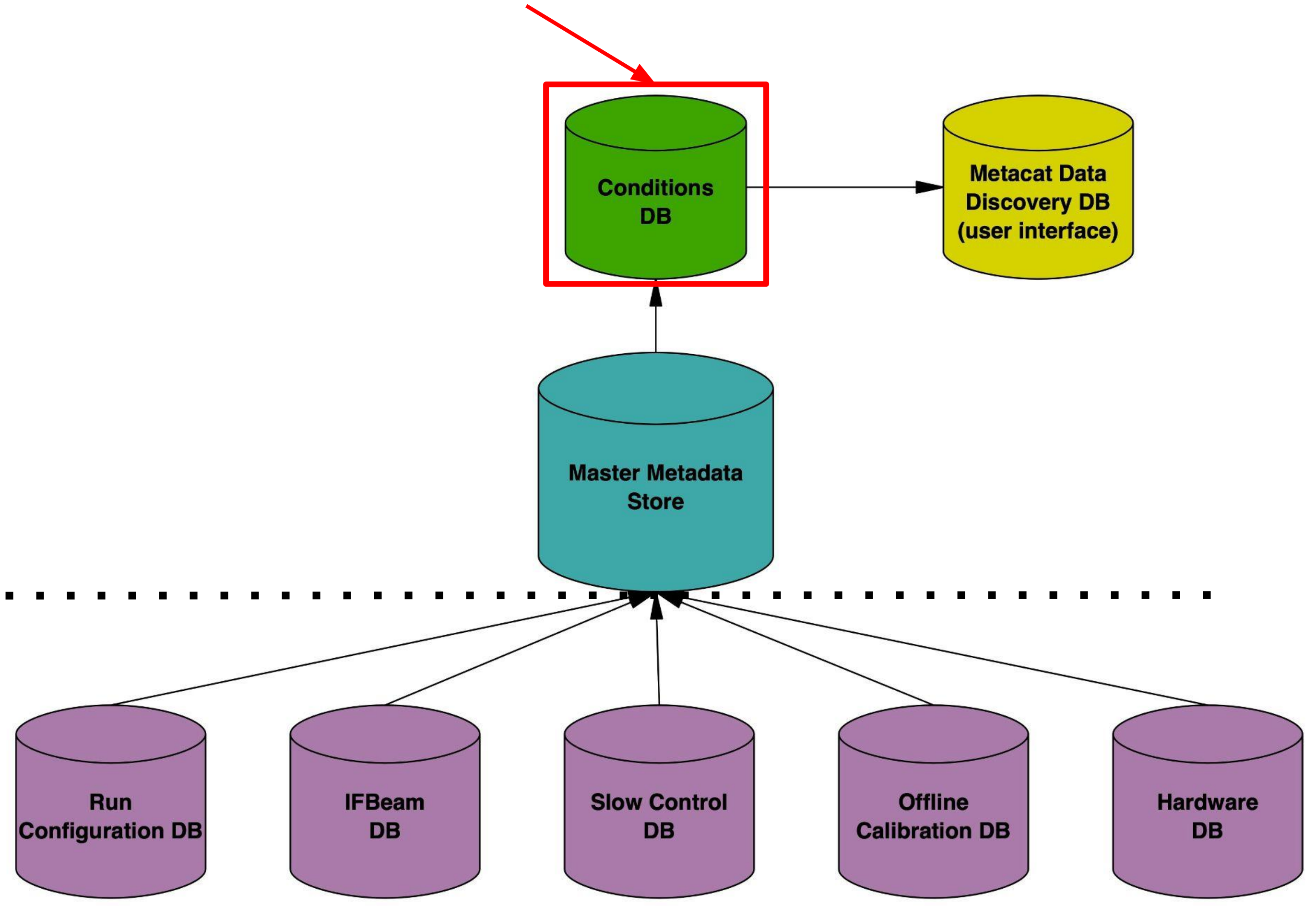
Data flow

That software framework API should be hitting

Central offline database for reconstruction and analysis

Aggregation database for offline

Online mission-critical databases



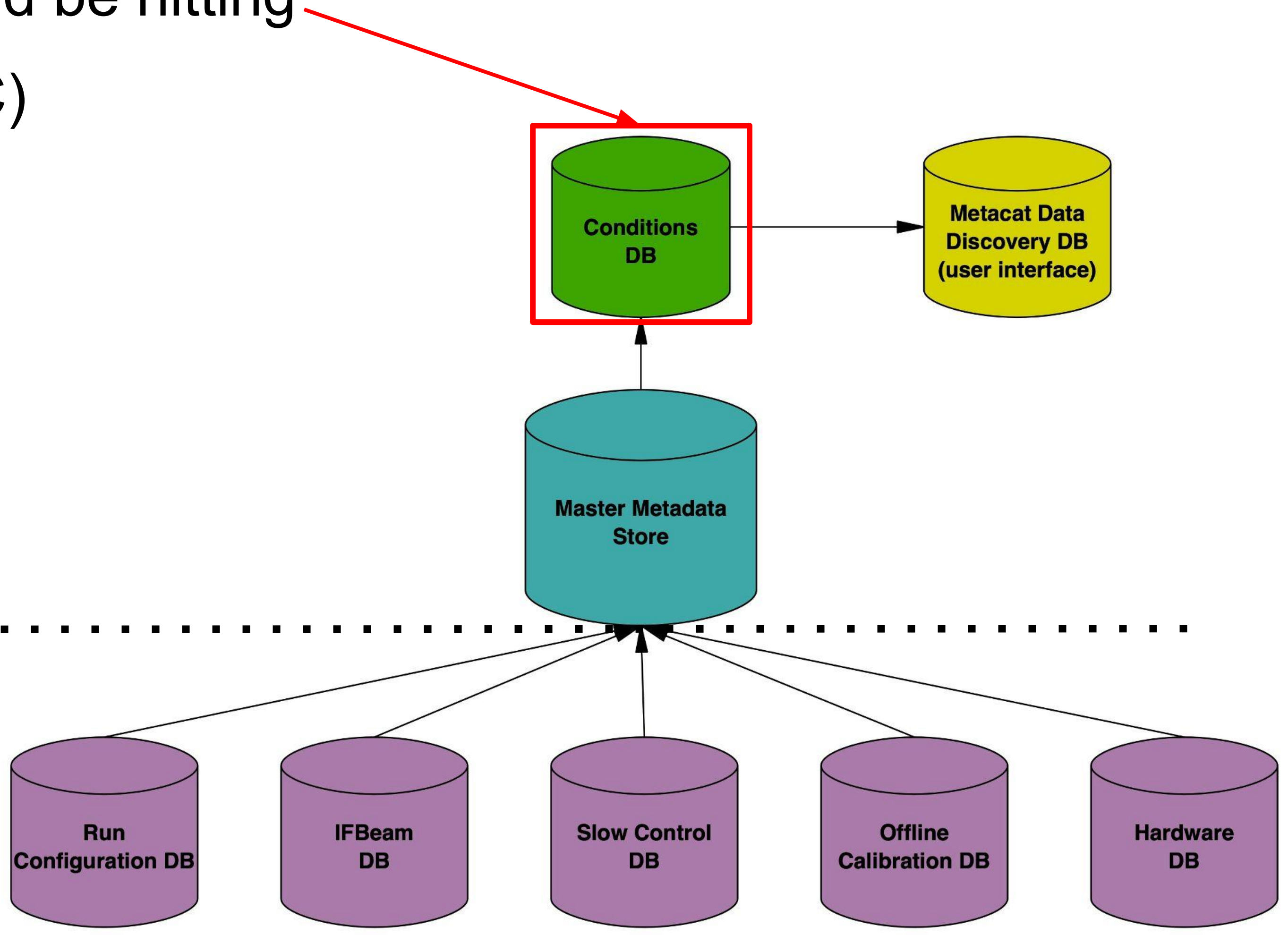
Data reduction

That software framework API should be hitting and ideally nothing else (think HPC)

Central offline database for reconstruction and analysis

Aggregation database for offline

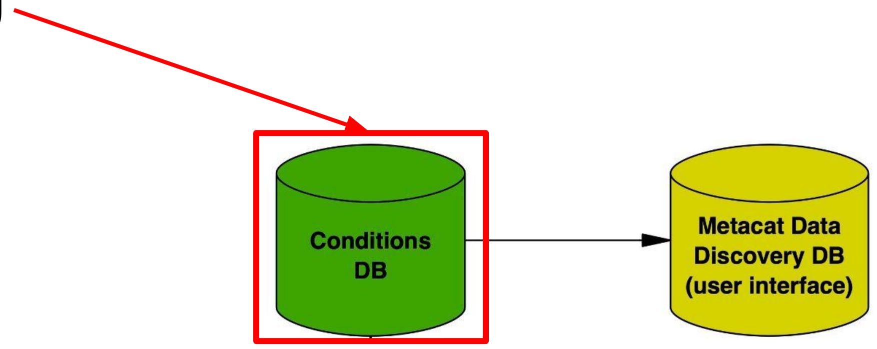
Online mission-critical databases



Data reduction

That software framework API should be hitting

and ideally nothing else (think HPC)



Central offline database for reconstruction and analysis

```
CondSvc.setGlobalTag(<GlobalTagName>);
```

Once per job

```
CondSvc.get(<MyConditionsType>, <RunNumber>);
```

In each module

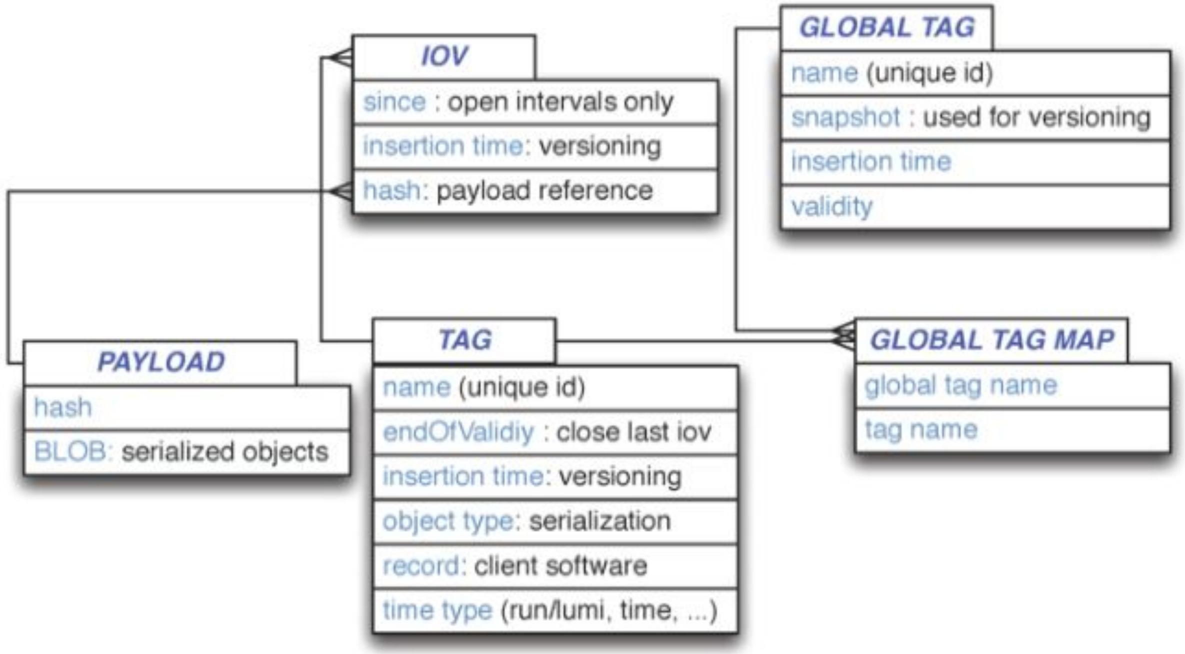
HSF Schema

- Now we have a dedicated HEP Software Foundation (HSF) conditions data activity:
  https://hepsoftwarefoundation.org/activities/conditionsdb.html
- Key recommendations for conditions data handling
  - Separation of payload queries from metadata queries

The main idea of the schema is that it doesn’t actually store payloads, it stores references Query this schema to effectively get a list of

calibrationType : payloadReference

pairs. Retrieve payloads separately (when needed)


IOV
since : open intervals only
insertion time: versioning
hash: payload reference
GLOBAL TAG
name (unique id)
snapshot : used for versioning
insertion time
validity
PAYLOAD
hash
BLOB: serialized objects
TAG
name (unique id)
endOfValidiy : close last iov
insertion time: versioning
object type: serialization
record: client software
time type (run/lumi, time, ...)
GLOBAL TAG MAP
global tag name
tag name

Under the hood - example

That software framework API should be hitting and ideally nothing else (think HPC)

```
CondSvc.setGlobalTag(“ProtoDUNE-awesome”)
```

Once per job (global configuration)

```
CondSvc.get(“SpaceChargeCorrection”, 72)
```

In each module

Inside the service…

```
CondDict = https://myserver.fnal.gov/myservice/whichPayloads/?gtName=ProtoDUNE-awesome&runNumber=72

LFN = CondDict[calibrationType]     // (calibrationType = “SpaceChargeCorrection” in this example)

payload failover defined as physical filename options (local, then /cvmfs/, then mylovelyserver)
```

Brookhaven National Laboratory

Under the hood - example (2)

```
CondSvc.setGlobalTag(“ProtoDUNE-awesome”)
```
Once per job (global configuration)

```
CondSvc.get(“SpaceChargeCorrection”, 72)
```
In each module

Imagine the SC correction for run number 72 is stored in a file called “SCE_72.root”, and we construct LFNs based on md5 checksums to make sure we’re efficient and only insert unique entries

so the LFN is “/adhod8393jkldm/SCE_72.root” *(data management experts, think of LFN vs PFN)*

the framework downloads payloads and caches them in a local /conditions directory

```
Trying… local “/conditions/adhod8393jkldm/SCE_72.root”... not found
Trying… cvmfs “/cvmfs/dune/conditions_payloads/adhod8393jkldm/SCE_72.root” not found
Trying… server “https://myserver.fnal.gov/payloads-service/adhod8393jkldm/SCE_72.root - found !
```

Under the hood - HPC

That software framework API should be hitting and ideally nothing else (think HPC)

`CondSvc.setGlobalTag` (“ProtoDUNE-awesome”) Once per job (global configuration)

`CondSvc.get` (“SpaceChargeCorrection”, 72) In each module

If I can’t access the server for the metadata DB, I can modify the service… effectively

```
CondDict = something which can give me the {calibrationType : payloadReference} pairs
```

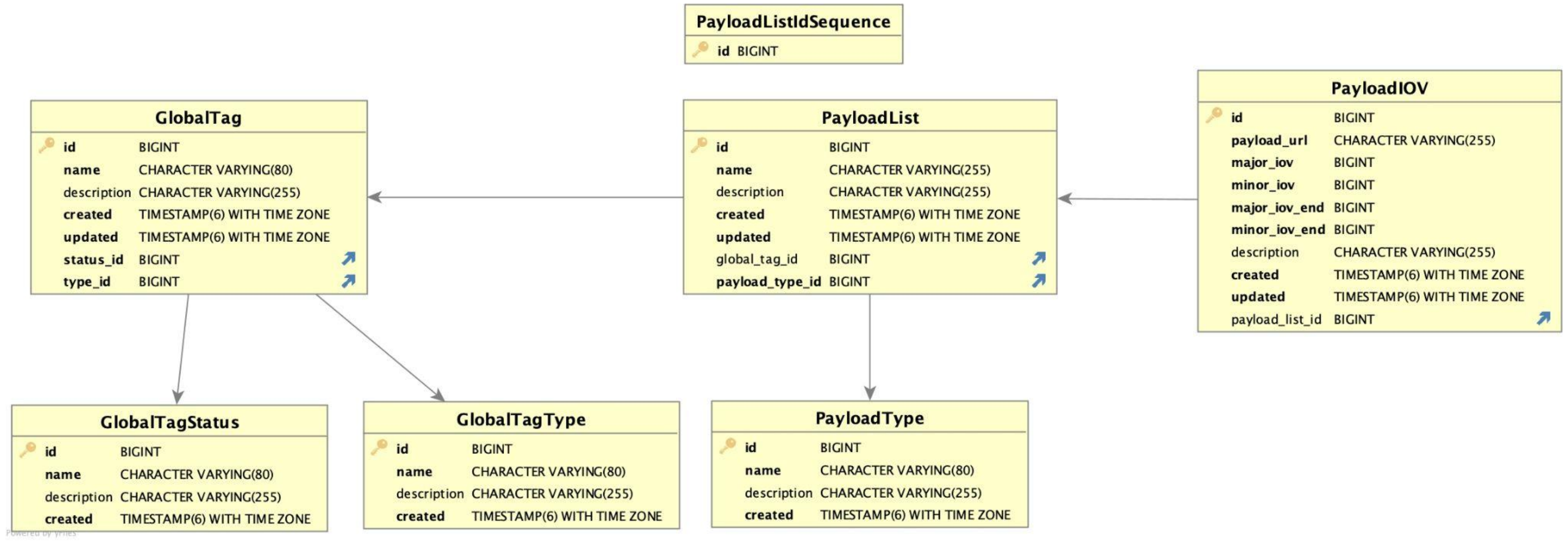
Belle II fails over to a sqlite.db file on /cvmfs, ATLAS has an sqlite backend option…

The payload locations already obey an LFN->PFN design

and the service itself can be lightweight

Brookhaven National Laboratory

HSF reality in pre-production for sPHENIX



PayloadListIdSequence

id	BIGINT

GlobalTag

id	BIGINT
name	CHARACTER VARYING(80)
description	CHARACTER VARYING(255)
created	TIMESTAMP(6) WITH TIME ZONE
updated	TIMESTAMP(6) WITH TIME ZONE
status_id	BIGINT
type_id	BIGINT

PayloadList

id	BIGINT
name	CHARACTER VARYING(255)
description	CHARACTER VARYING(255)
created	TIMESTAMP(6) WITH TIME ZONE
updated	TIMESTAMP(6) WITH TIME ZONE
global_tag_id	BIGINT
payload_type_id	BIGINT

PayloadIOV

id	BIGINT
payload_url	CHARACTER VARYING(255)
major_iov	BIGINT
minor_iov	BIGINT
major_iov_end	BIGINT
minor_iov_end	BIGINT
description	CHARACTER VARYING(255)
created	TIMESTAMP(6) WITH TIME ZONE
updated	TIMESTAMP(6) WITH TIME ZONE
payload_list_id	BIGINT

GlobalTagStatus

id	BIGINT
name	CHARACTER VARYING(80)
description	CHARACTER VARYING(255)
created	TIMESTAMP(6) WITH TIME ZONE

GlobalTagType

id	BIGINT
name	CHARACTER VARYING(80)
description	CHARACTER VARYING(255)
created	TIMESTAMP(6) WITH TIME ZONE

PayloadType

id	BIGINT
name	CHARACTER VARYING(80)
description	CHARACTER VARYING(255)
created	TIMESTAMP(6) WITH TIME ZONE

- Written by the Belle II CDB developer, based on HSF schema and leveraged a lot of useful discussions in the HSF CDB activity with ATLAS and CMS experts
  - Postgres backend, Django REST for the service
- sPHENIX use case is 200k cores running at BNL, so performance is a very high priority!
  - Expect to migrate Belle II to the same software
- Using cvmfs for file storage (and can write directly to the underlying file system at BNL)

Proposal

We try the thing produced for sPHENIX (takes data next year, in pre-production now). Main dev is funded by Belle II Ops program, part of my team based at CERN

We deploy it at FNAL, simple Postgres DB schema, lightweight REST service preferably deployed on OKD (what we're using at BNL)

Lino and Ana Paula are working out who does what:

https://docs.google.com/document/d/1hndm5W2OI9A7SQ4hAgP5GNiyfNvkcvVi4wVxf9fc6cg/edit?usp=sharing

Most of the work is independent of the service and related to moving to run-based access and a common API

The software API/interface allows us to switch out the backend in case of need and in many ways is the main aim of this, moving people to a simple and unique interface to conditions data

Brookhaven National Laboratory