Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle *Since 1965*

Belle II Conditions Database

LYNN WOOD, TODD ELSETHAGEN, MALACHI SCHRAM, ROMULUS GODANG

Pacific Northwest National Laboratory

US Belle II Summer School, Aug 15-18, 2016

Purpose of a Conditions Database

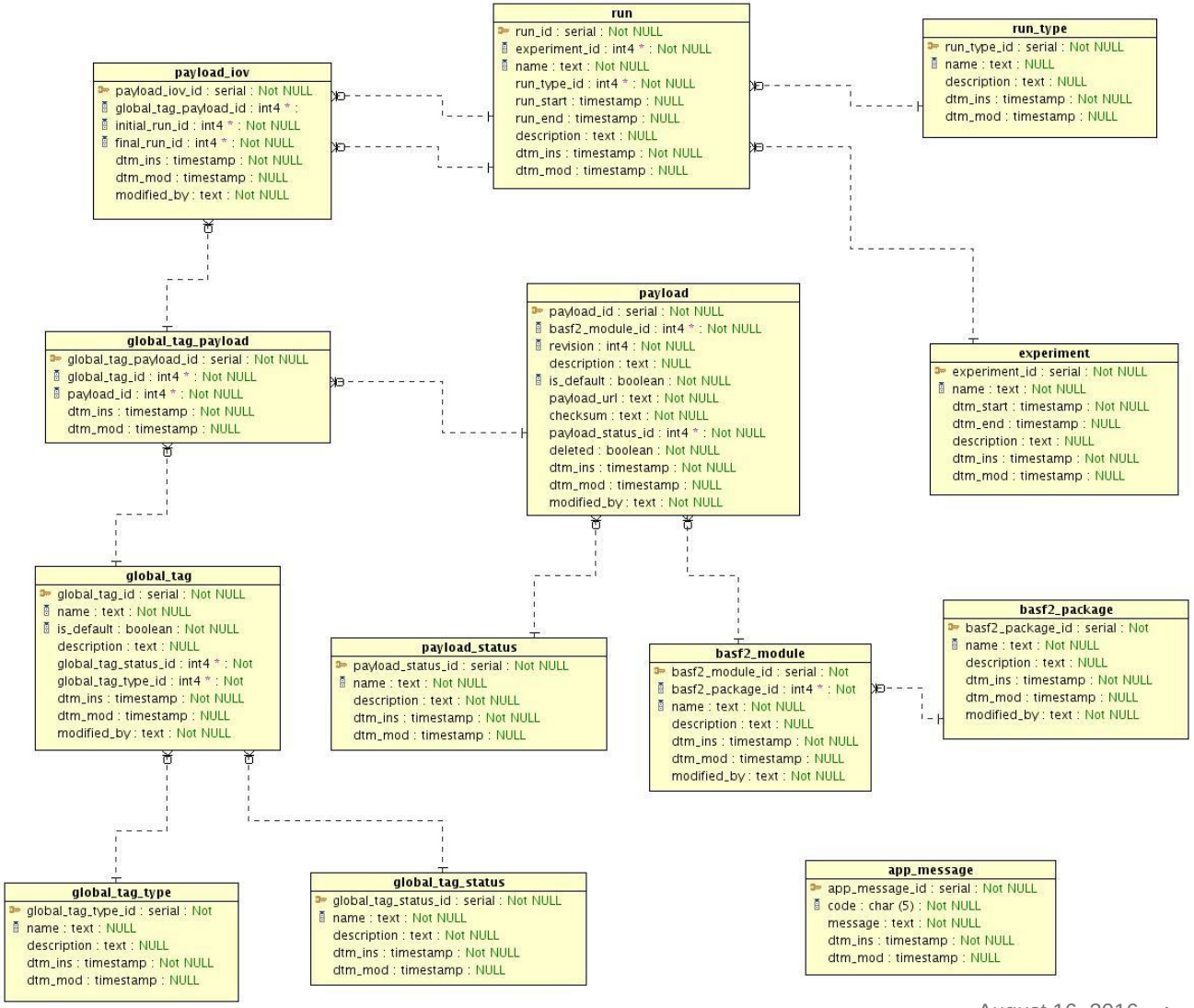
- Holds conditions data – status of detectors
  - Calibration, alignment, geometry, environmental parameters
  - Types/values can be simple or complex
- Uses include reprocessing of raw data, MC production, possibly user analysis?
- Different than a…
  - *Configuration* DB that holds settings for the detector
  - *Logging* DB that records time-dependent information from the detector
  - *File system/metadata* DB that keeps tracks of experiments, runs, files, etc.
- *But* almost always will interact with those in some way!

Database Schema

Pacific Northwest NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

- A DB schema defines the types of entries in the DB and their relationships to each other

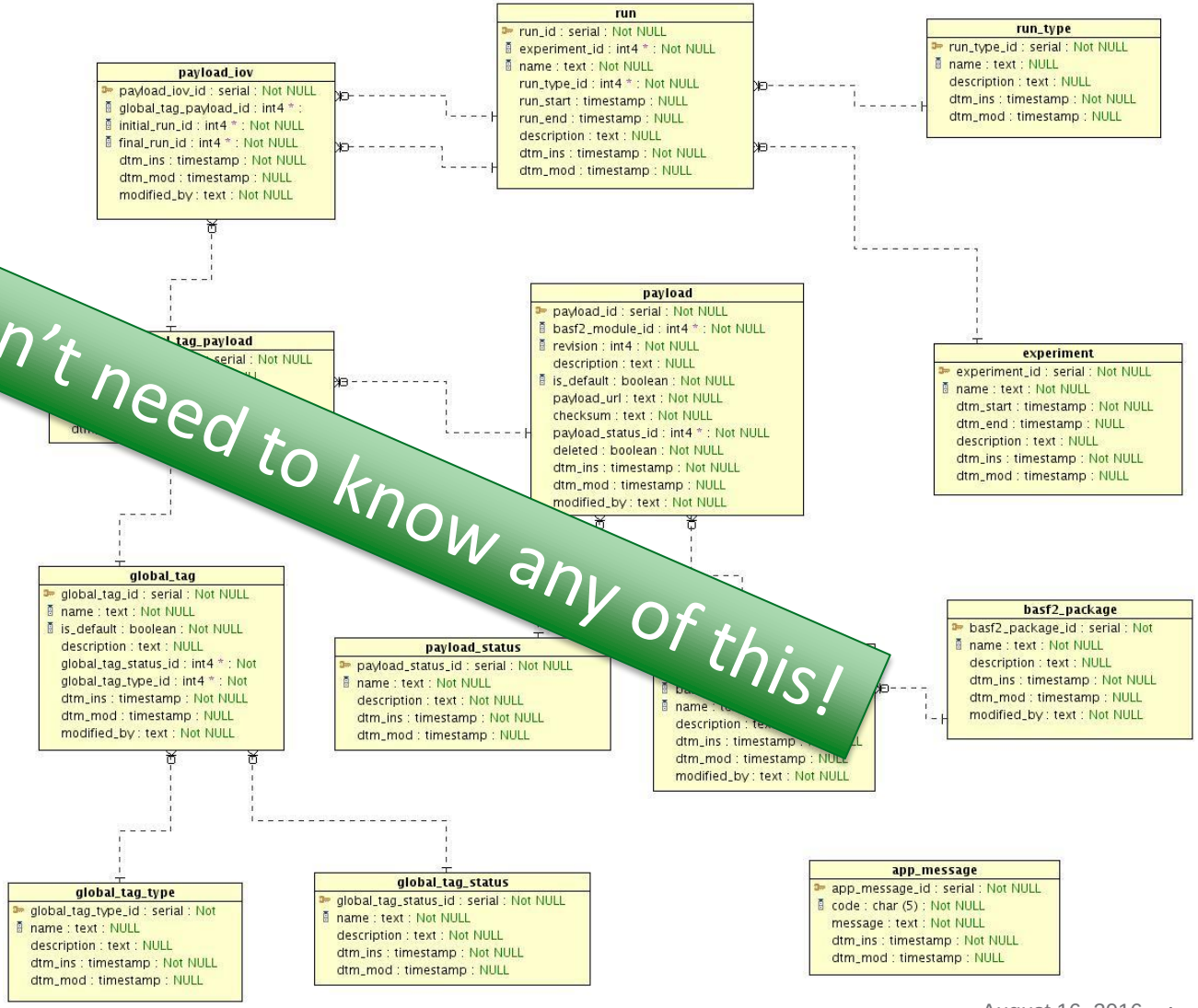


Database Schema

Pacific Northwest NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

- A DB schema defines the types of entities in the DB and their relationships to each other



Structured Query Language (SQL)

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- SQL is a standard computer language for relational DB management and data manipulation. It is used to query, insert, update, and modify data.

- Examples:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

```
SELECT STREAM *
FROM (
  SELECT STREAM '"Orders"' as type, orderId FROM Orders
  UNION ALL
  SELECT STREAM 'SHIPMENT' as type, orderId FROM Shipments
)
```

Structured Query Language (SQL)

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- SQL is a standard computer language for relational DB management and data manipulation. It is used to query, insert, update, and modify data.

- Examp

```
ers.OrderID, Customers.CustomerName, Orders.OrderDate
INNER JO
ON Orders.Custo  ers.CustomerID;
```

You don't need to know any of this!

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

```
SELECT STREAM *
FROM (
  SELECT STREAM '"Orders"' as typ  FROM Orders
  UNION ALL
  SELECT STREAM 'SHIPMENT' as type, orderId FROM Shipments
)
```

REST API

- The Belle II conditions DB is access via a “REST” interface, basically just HTML accesses to a server
  - This is very common in industry, and makes the database easier to scale and maintain

Example: **get /v1/globalTag/{gtName}/globalTagPayloads**

- Data is returned in either JSON or XML format:


```
[
 {
  "globalTagPayloadId": 0,
  "globalTag": {
   "globalTagId": 0,
   "description": "string",
   "isDefault": false,
...
```

REST API

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- The Belle II conditions DB is access via a “REST” interface, basically just HTML accesses to a server
  - This ery common in industry, and makes the database easier to scale and ma

Example: get /… /{gtName}/globalTagPayloads

- Data is returned in either JSON … at:

```
[
 {
  "globalTagPayloadId": 0,
  "globalTag": {
   "globalTagId": 0,
   "description": "string",
   "isDefault": false
…
```

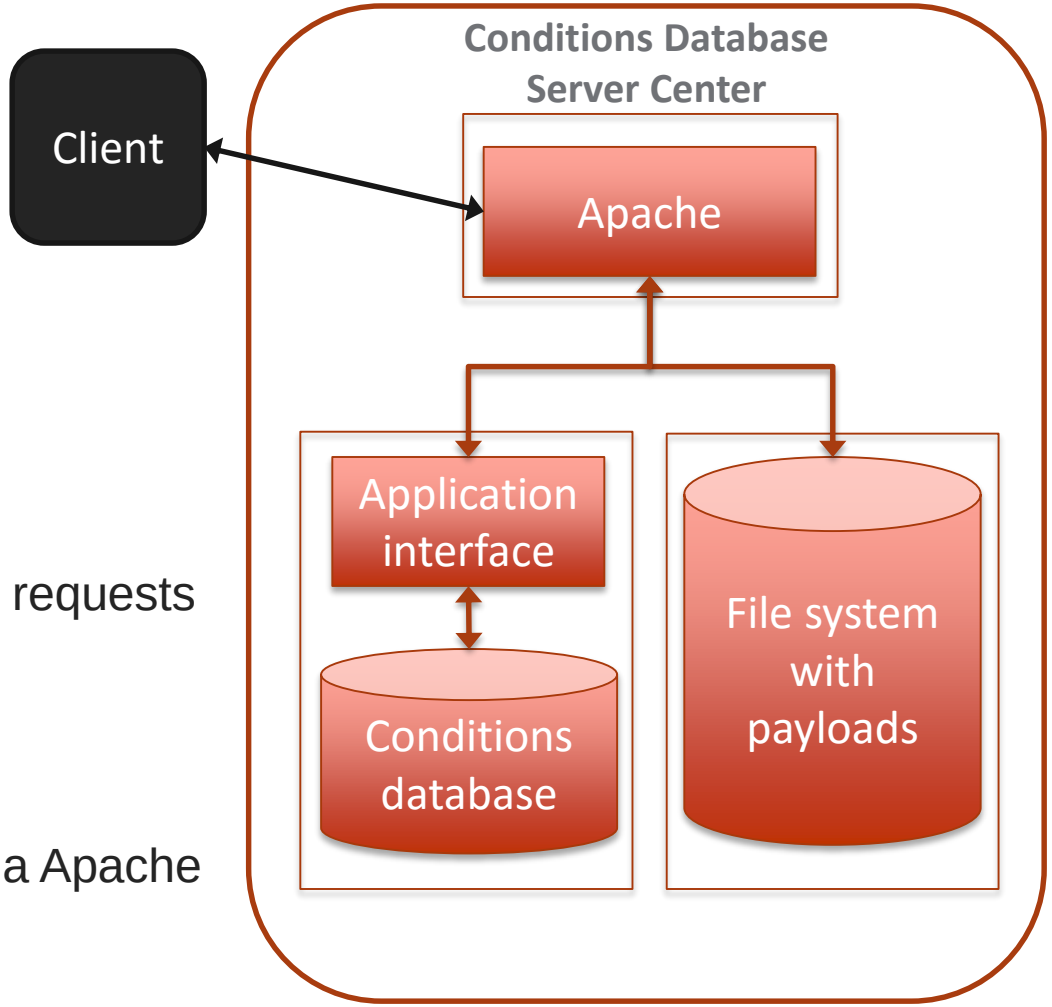
You don’t need to know any of this!*

*(unless you’re trying something tricky, and even then this is MUCH easier than schemas and SQL…)

What the Conditions DB Server Looks Like Now…

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- **HTTP server**
  - Apache, etc.
- **Database**
  - PostgreSQL
- **Application interface**
  - Translates the REST (HTTP) requests into SQL
- **File system**
  - Simplest option: on server via Apache

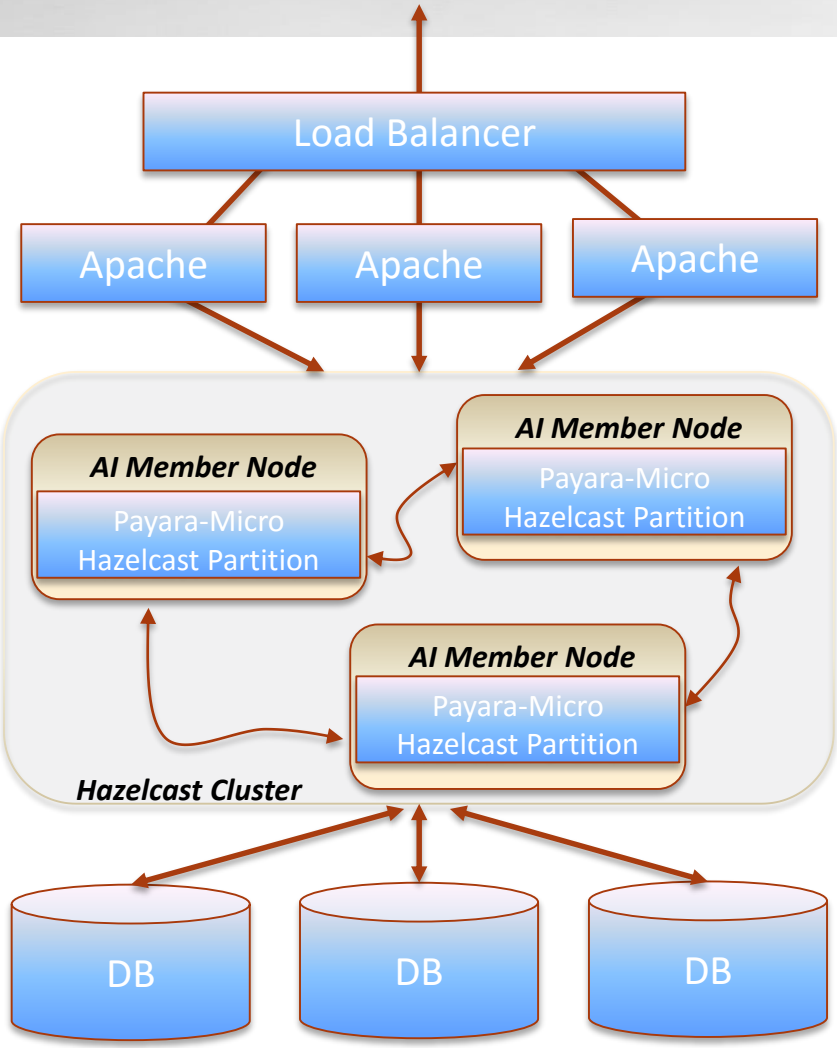


You don't need to know any of this!

What We Are Working Towards

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- Database loading will be significant when data reprocessing is underway
- Goal: make all components reliable and scalable



You don't need to know any of this!

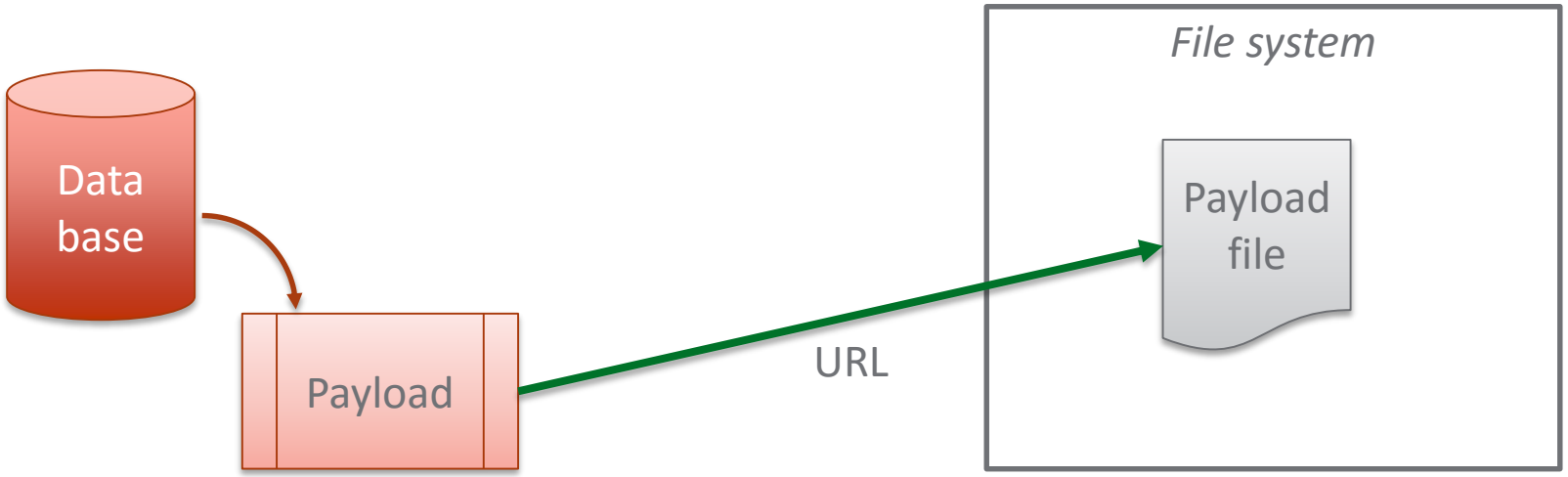
What You Need to Know

- The components that the Belle II Conditions DB is designed with:
  - Payloads (and payload files)
  - Intervals of Validity (IOV)
  - Global tags
- How the conditions DB is accessed from Basf2
- Upcoming changes

Payloads and Payload Files

- A **payload** is a piece of conditions data
  - (Belle II assumes ROOT objects, but the DB technically doesn't care)

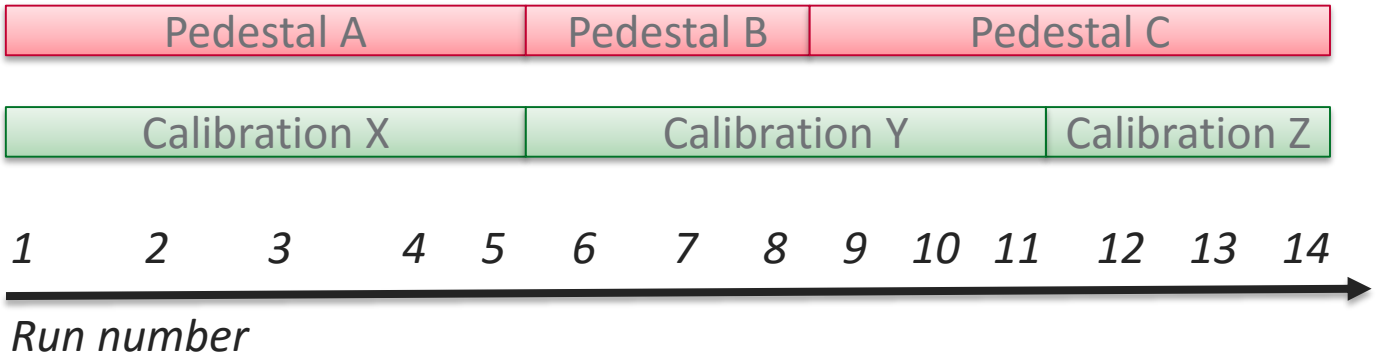
- In the Belle II database, the payloads are stored <u>outside</u> the DB itself
  - The DB only contains references (URLs) to the **payload files**
  - Designed this way for multiple reasons: keeps DB itself small, allows separation of DB server and file server, etc.



Intervals of Validity (IOV)

- Each payload is valid for a certain period of time that needs to be identified in the database
  - In Belle II, current granularity is a “run” (~hours)
  - An IOV contains a start run and a stop run (or -1 for “forever”)

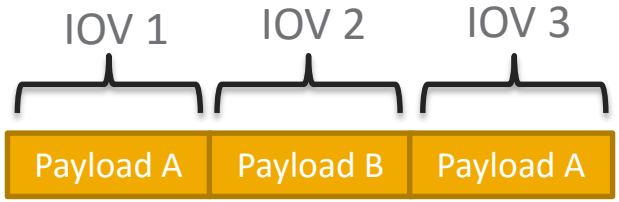
- Each type of payload may (will) have different IOVs, depending on when conditions change

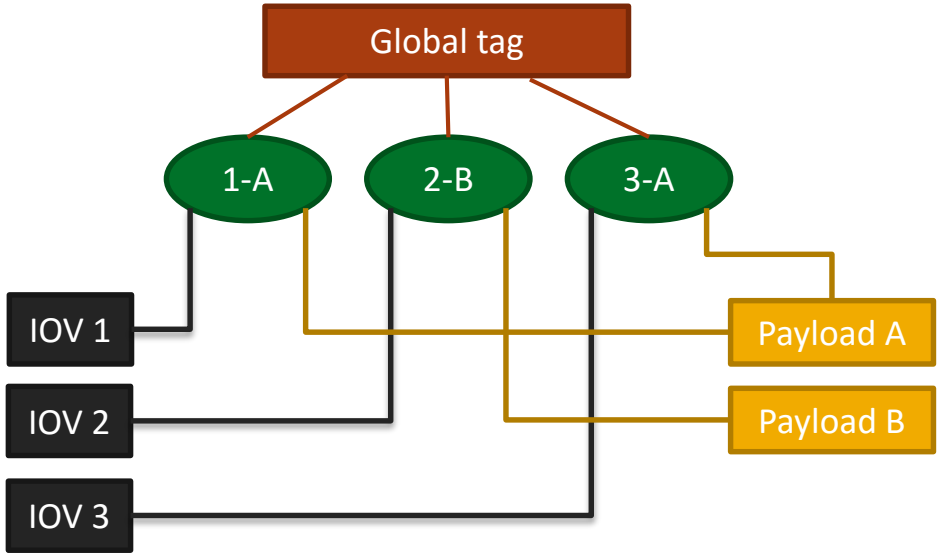


Global Tags

- There will be many payloads valid over many different IOVs, so we need a way to keep track of all of all the “good” ones

- This is a **global tag**: contains a list of IOV-payload relationships
  - Note that separation of IOVs and payloads allows payload reuse in different IOVs





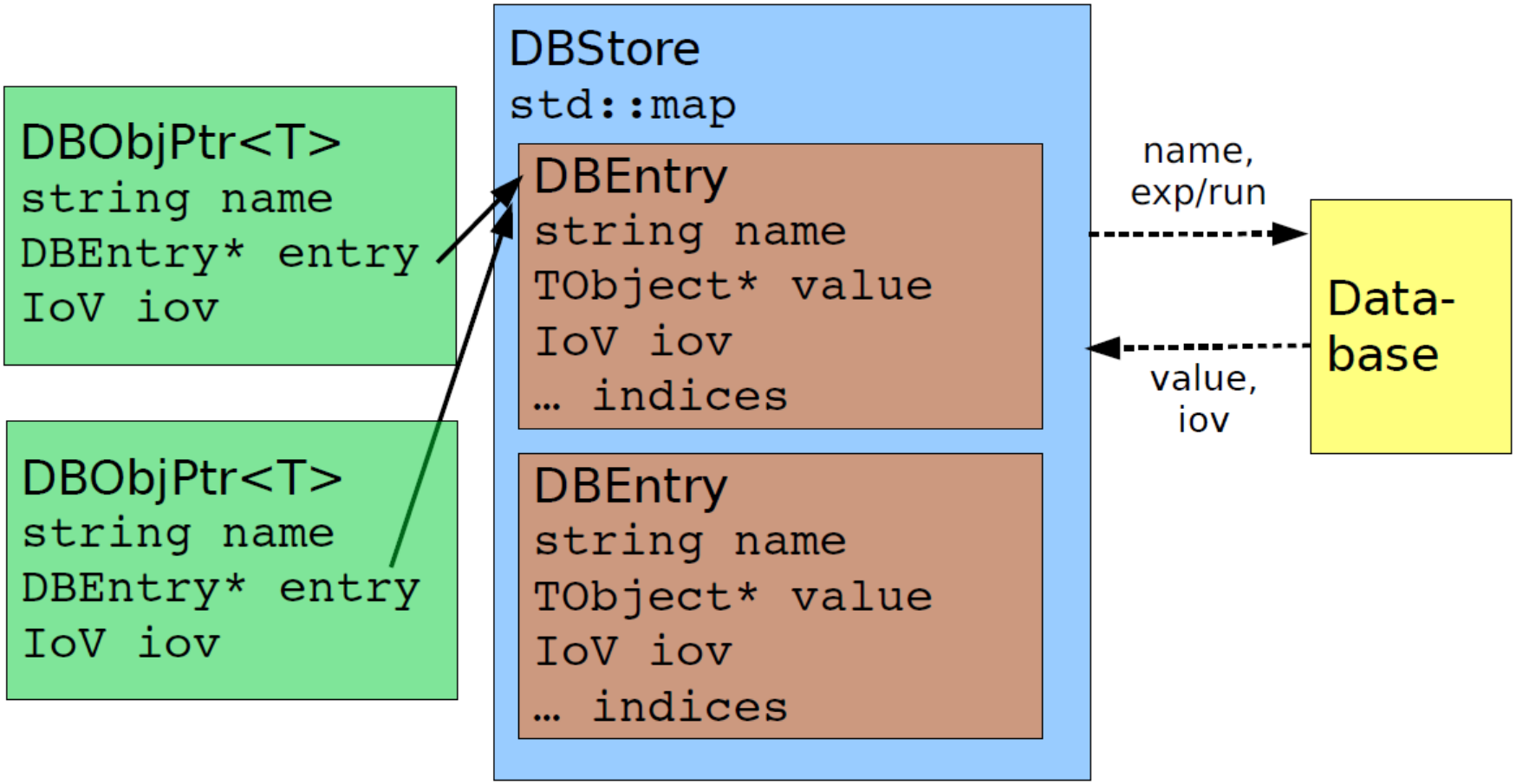
Global Tag Management

- Three levels of control are planned (but not yet present):
  - ***User*** Read access to database
  - ***Developer*** Can add payloads, IOVs, global tags
  - ***Coordinator*** Can change global tag status
- Global tags have three states: "**new**", "**invalid**", or "**published**"
- Concept of a "Global tag coordinator" (GTC) was discussed at the Software/Computing Workshop in May 2016
  - Would coordinate creation and validation of global tags for reprocessing
- For development work, everyone creates their own global tags
  - **Naming convention will be published soon!**

Database Interface in Basf2

- Similar to the successful Basf2 “DataStore” concept
- Encapsulates low-level interface(s) – no SQL, REST, etc. required!
- Handles many things automatically:
  - Experiment and run number
  - Memory management
  - Caching (only accesses DB if new data is needed)
  - Run dependency
    → **automatically get the correct payload** (like DataStore)
- Hooks available to handle data changes, accessing array elements by index, etc.

Basf2 DBStore



DB Read Access

- Reading from database is straightforward:

```
DBObjPtr<BeamParameters> beams("superkekb/beams");
beams->getHEREnergy();
```

- Sequence:
  - *(global tag is identified in steering script before analysis starts)*
  - Asks DBStore about object with name "superkekb/beams"
  - If no DBEntry with this name, creates on and accesses the DB
  - Fills the IOV and value
  - Returns a pointer to the DBEntry

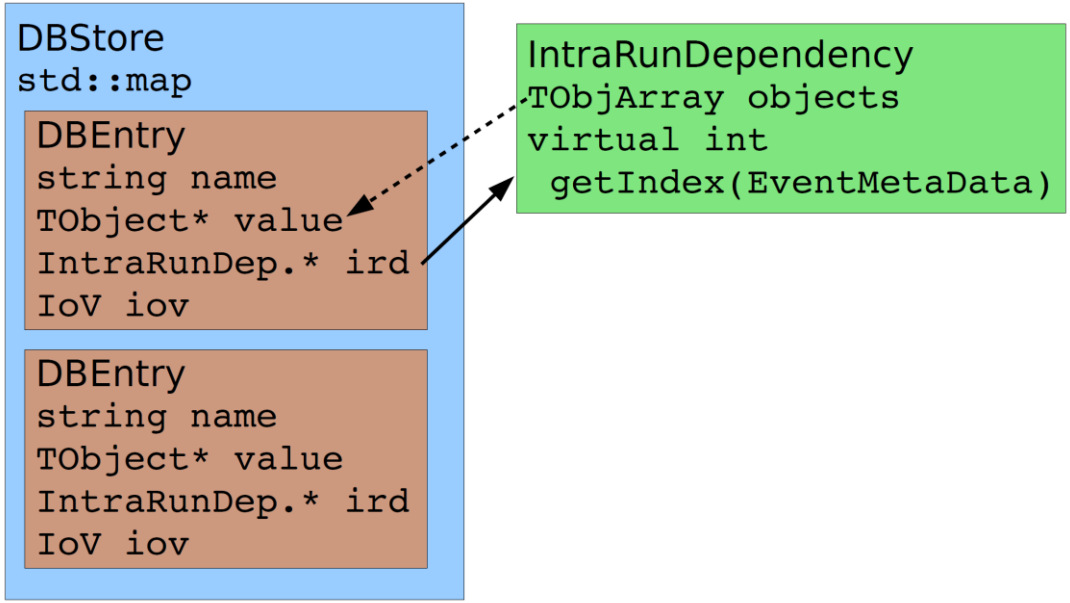
- There is a similar class "DBArray" for arrays
  - Similar naming convention as DataStore (add "s" to arrays)

Automatic Updates

- When a new run is accessed:
  - Checks all DBEntries
  - If the new experiment/run is outside of a current IOV, the DBStore gets the next IOV and value for that payload
- DB access is done as a bulk request (not individual ones)
- Possible to add callback
  - Example: parameter calculated from conditions data needs to be recalculated if the input values change

Intra-Run Dependency

- What if conditions data changes **more frequently** than run-level granularity?
- "Intra-run dependency" support for storing multiple values in a single payload



- Current granularity: event number within a run
- DBStore automatically identifies these when loaded from the database
- Also checks for changes at event level as well as run level

Local Databases

- Possible to use local DB instead of official DB
  - Useful for testing, storage of payloads during development, etc.

- Database represented as text file with IOV assignments
- Payload files are identical (but located on local machine)

- Better still, the Basf2 DBStore already creates a local copy of all the payload files when they are downloaded
  - Also creates a “dbcache.txt” file that can be used as a local database

  → **Local DB automatically generated from global DB**

Database Chaining

- Possible to create a list of databases
  - Each one is searched in turn until a payload is found
  - Can combine local and global (remote) DBs
- Configured by the steering file; default sequence is:
  - Local database in “localdb” directory
  - Global database global tag “production”
  - Local database in “data/framework” (fallback with default payloads)
- Can be overridden/added to by user’s steering script

DB Write Access

- Somewhat more complicated; need to deal with global tags and IOVs
  - See examples in ARICH source code

```
TClonesArray agelConstants("Belle2::ARICHAerogelInfo");
// fill agelConstants here…

// global tag identified previously (in steering script)
IntervalOfValidity iov(0, 0, -1 ,-1);   // from expt 0, run 0 to forever

// store under user-defined name
// Database::Instance().storeData("testAerogel", &agelConstants, iov);
// store under standard name
Database::Instance().storeData(&agelConstants, iov);
```

Code Examples

- ARICH was the first detector to use the Conditions DB en-masse by moving all their XML-based data into the DB (hooray!)

- Code examples in **software/arich**:
  - dobjects/include/* ← object (payload) definitions
  - dobjects/src/*
  - examples/ArichDatabaseInstructions.txt ← how to use the scripts
  - examples/ArichDatabaseImporter_export.py
  - examples/ArichDatabaseImporter_import.py

- From the framework:
  - framework/tests/database.cc ← test suite for DB
  - framework/tests/upload_localdatabase ← transfers local DB to global

Continued Conditions DB Development

- **Schema change in progress**
  - Experiment/run are currently "objects" in DB; adds complexity but not values
    → changing them to simple numbers
  - Will move all (valuable) data from original format to new schema
- Second, more robust server up and running at PNNL; will be used for stress-testing
- **<u>All subdetector groups</u>** are supposed to stop using XML files for conditions data by Nov 2016…
  - All groups have made progress, but lots of work still to do!

“Documentation”

- In confluence:
  - https://confluence.desy.de/display/BI/Software+DatabaseInterface
- REST API:
  - http://belle2db.hep.pnnl.gov/b2s/api/
- Mailing list:
  - database-conditions@belle2.org
- Meetings:
  - Typically Tuesdays every ~3 weeks or so
  - https://confluence.desy.de/display/BI/Software+DBMeetingPage