# Metadata in Mu3e

Urs Langenegger

2024/03/~~19~~21

- Introduction
- CDB/RunDB
- "Configs"

"There is no silver bullet." (F. Brooks)

## Introduction

- "Databases and configs" is not the (entire) issue - it's metadata
- Examples for metadata in HEP experiments
  - ▷ construction data $\rightarrow$ partsdb
  - ▷ detector configurations (a.k.a. DACs, tunes, masks, . . . )
  - ▷ runs database $\rightarrow$ RunDB
  - ▷ job configuration (files)
  - ▷ conditions data (calibration/alignment/quality/. . . ) $\rightarrow$ CDB
  - ▷ MC samples requests, production campaigns
  - ▷ file catalogs (versioned, historical, . . . )
  - ▷ slow control (a.k.a detector control) status (also outside of running time)
- "Normally"
  - ▷ all metadata end up in various databases,
  - ▷ each dedicated to one specific type of metadata
- Except for job configurations
  - ▷ normally not in a database
  - $\rightarrow$ tagged (checked out) "releases", e.g. in the online (computing) environment

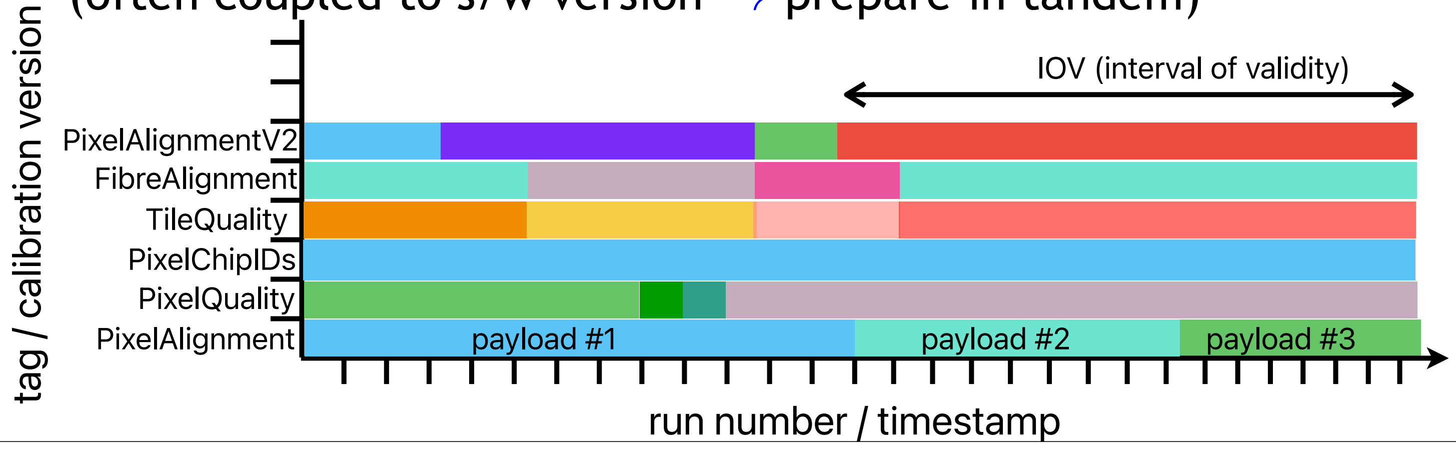
# Introduction - CDB

- CDB = conditions database

| Category | Example | Application |
|---|---|---|
| geometry | construction data | reconstruction |
| geometry | alignment | better reconstruction |
| detector status | dead chips | better tracking efficiency |
| detector status | $B$-field strength | correct reconstruction |
| beam status | beam current | background conditions |
| . . . | . . . | . . . |

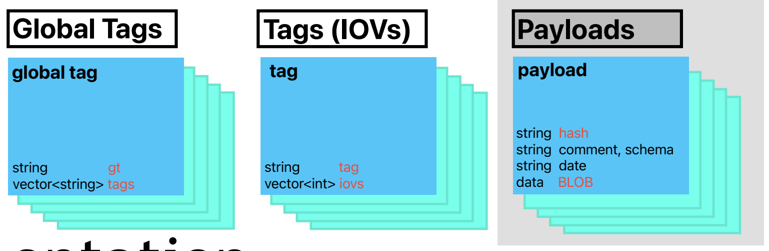
$\rightarrow$ But not everything (in the same structure)!

- Payloads, interval-of-validity (IOV), and tags
  - ▷ tag = tagname plus iov list
  - ▷ global tag = complete set of tags required for any run (often coupled to s/w version $\rightarrow$ prepare in tandem)

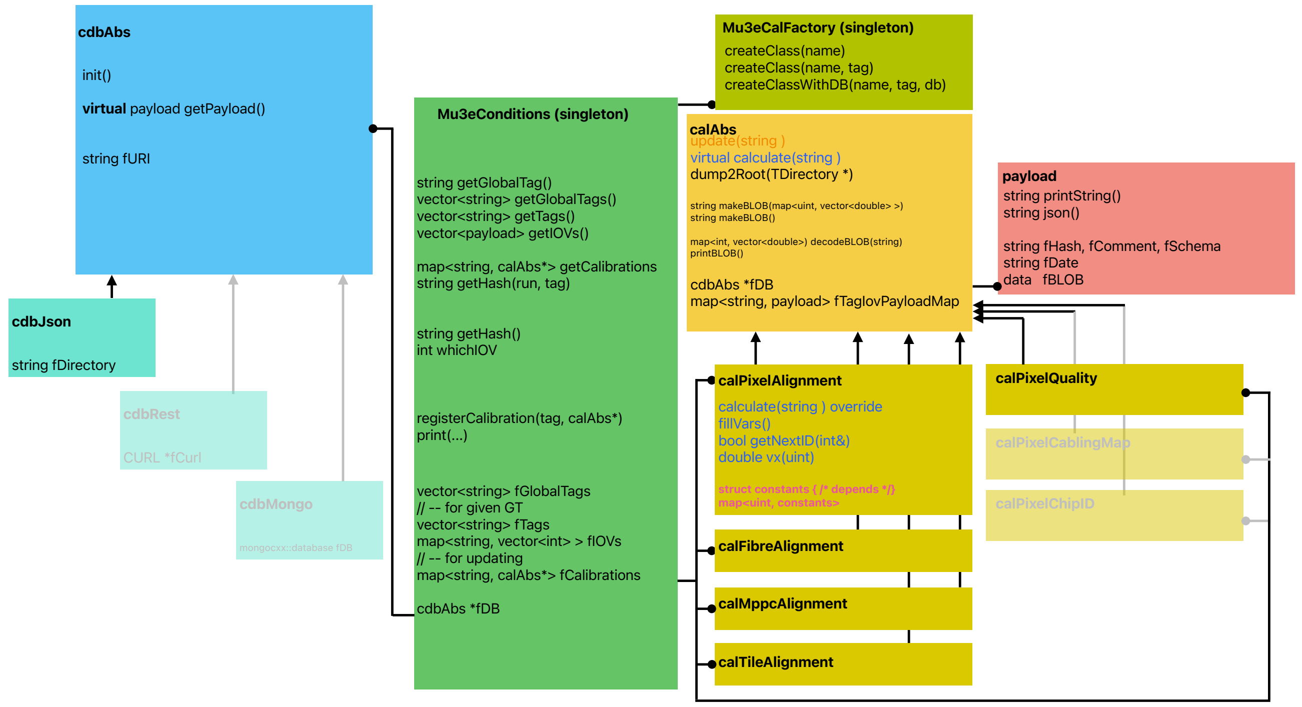


▷ conditions:
- everything detector-related that is not event data and required for event processing and producing physics results

# CDB data model and code organization



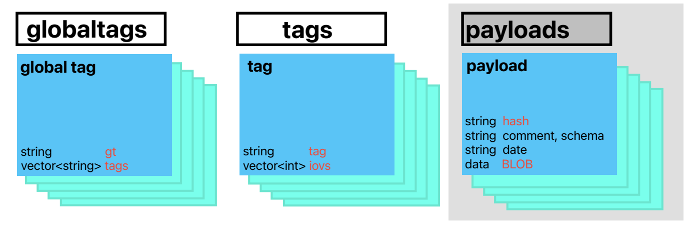
- Database contents
  - ▷ ≈ three collections ("tables")
  - ▷ mapped somehow to code
- CDB access/backend server implementation
  - ▷ filesystem-based cdbJSON with no external dependency
  - ▷ REST api cdbRest, curl as external dependency (based on http)
    provides access to mongodb backend server (currently: pc11740.psi.ch)
  - ▷ Mu3eConditions central entry point for (trirec/sim/DQM) code

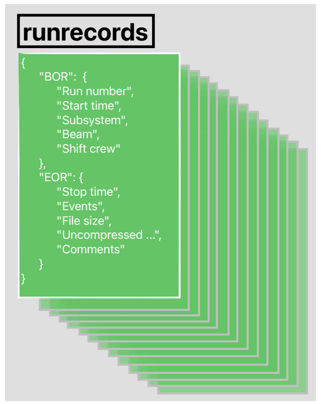


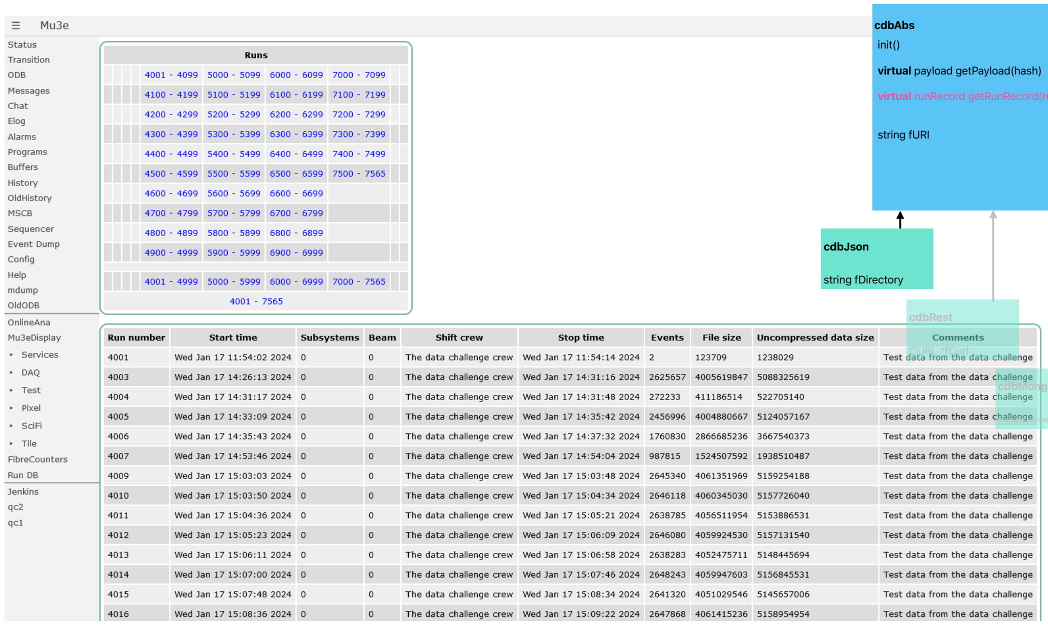
Nik

# RunDB

- RunDB integrated into CDB
  - ▷ collection of runrecords
  - ▷ MIDAS can write to mongodb via rest API
- Interface to RunDB?
  - ▷ first steps done in DC2023
  - ▷ other options will follow
- Integration into CDB not a necessity
  - ▷ stand-alone DB also possible (as in CMS!)

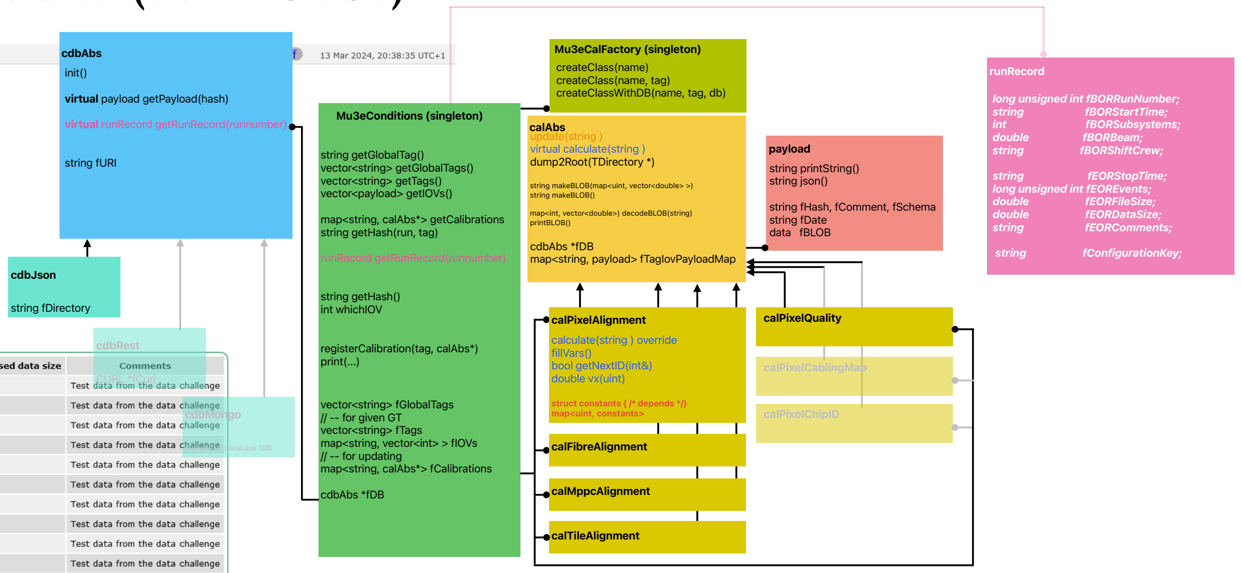






| Runs | | | |
|---|---|---|---|
| 4001 - 4099 | 5000 - 5099 | 6000 - 6099 | 7000 - 7099 |
| 4100 - 4199 | 5100 - 5199 | 6100 - 6199 | 7100 - 7199 |
| 4200 - 4299 | 5200 - 5299 | 6200 - 6299 | 7200 - 7299 |
| 4300 - 4399 | 5300 - 5399 | 6300 - 6399 | 7300 - 7399 |
| 4400 - 4499 | 5400 - 5499 | 6400 - 6499 | 7400 - 7499 |
| 4500 - 4599 | 5500 - 5599 | 6500 - 6599 | 7500 - 7565 |
| 4600 - 4699 | 5600 - 5699 | 6600 - 6699 | |
| 4700 - 4799 | 5700 - 5799 | 6700 - 6799 | |
| 4800 - 4899 | 5800 - 5899 | 6800 - 6899 | |
| 4900 - 4999 | 5900 - 5999 | 6900 - 6999 | |
| 4001 - 4999 | 5000 - 5999 | 6000 - 6999 | 7000 - 7565 |
| 4001 - 7565 | | | |

| Run number | Start time | Subsystems | Beam | Shift crew | Stop time | Events | File size | Uncompressed data size | Comments |
|---|---|---|---|---|---|---|---|---|---|
| 4001 | Wed Jan 17 11:54:02 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 11:54:14 2024 | 2 | 123709 | 1238029 | Test data from the data challenge |
| 4003 | Wed Jan 17 14:26:13 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 14:31:16 2024 | 2625657 | 4005619847 | 5088325619 | Test data from the data challenge |
| 4004 | Wed Jan 17 14:31:17 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 14:31:48 2024 | 272233 | 411186514 | 522705140 | Test data from the data challenge |
| 4005 | Wed Jan 17 14:33:09 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 14:35:42 2024 | 2456996 | 4004880667 | 5124057167 | Test data from the data challenge |
| 4006 | Wed Jan 17 14:35:43 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 14:37:32 2024 | 1760830 | 2866685236 | 3667540373 | Test data from the data challenge |
| 4007 | Wed Jan 17 14:53:46 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 14:54:04 2024 | 987815 | 1524507592 | 1938510487 | Test data from the data challenge |
| 4009 | Wed Jan 17 15:03:03 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 15:03:48 2024 | 2645340 | 4061351969 | 5159254188 | Test data from the data challenge |
| 4010 | Wed Jan 17 15:03:50 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 15:04:34 2024 | 2646118 | 4060345030 | 5157726040 | Test data from the data challenge |
| 4011 | Wed Jan 17 15:04:36 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 15:05:21 2024 | 2638785 | 4056511954 | 5153886531 | Test data from the data challenge |
| 4012 | Wed Jan 17 15:05:23 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 15:06:09 2024 | 2646080 | 4059924530 | 5157131540 | Test data from the data challenge |
| 4013 | Wed Jan 17 15:06:11 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 15:06:58 2024 | 2638283 | 4052475711 | 5148445694 | Test data from the data challenge |
| 4014 | Wed Jan 17 15:07:00 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 15:07:46 2024 | 2648243 | 4059947603 | 5156845531 | Test data from the data challenge |
| 4015 | Wed Jan 17 15:07:48 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 15:08:34 2024 | 2641320 | 4051029546 | 5145657006 | Test data from the data challenge |
| 4016 | Wed Jan 17 15:08:36 2024 | 0 | 0 | The data challenge crew | Wed Jan 17 15:09:22 2024 | 2647868 | 4061415236 | 5158954954 | Test data from the data challenge |



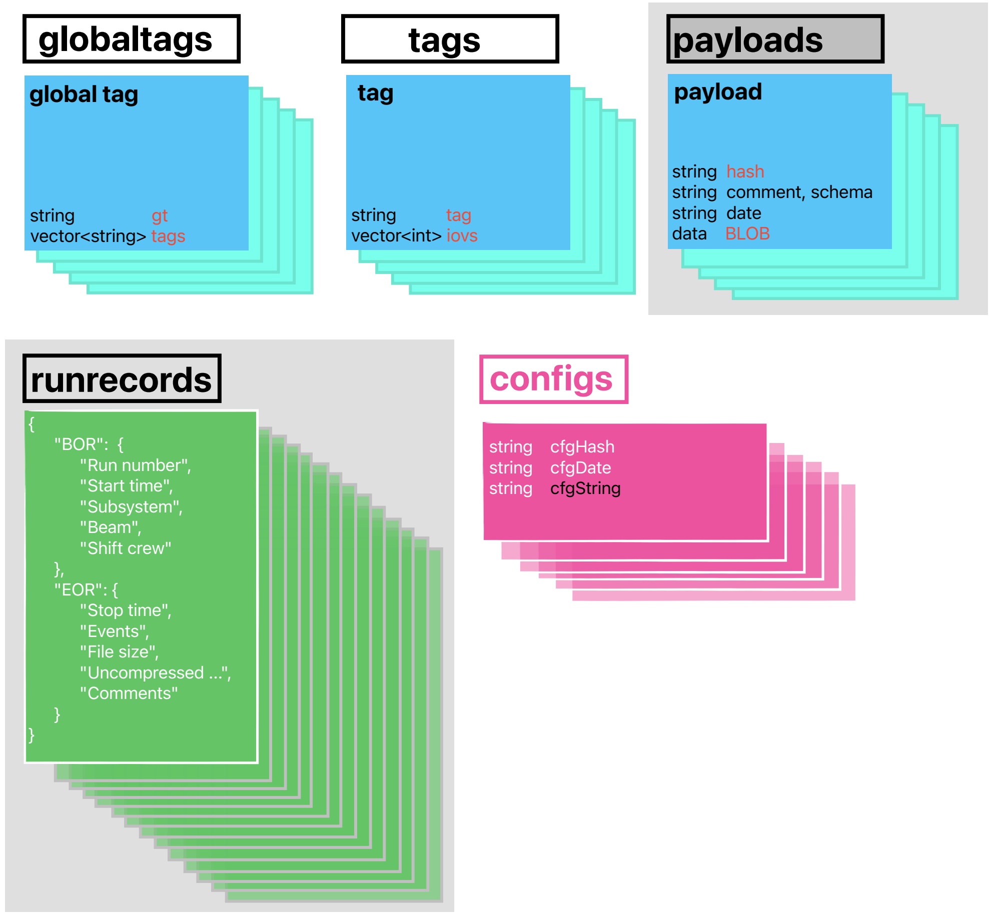
## "Databases and (job!) configs"

[CHEP06 (ATLAS)](#)
[2203.00463](#)

- Terminology (somewhat handwaving)
  - ▷ conditions:
    - everything detector-related that is not event data and required for event processing and producing physics results
  - ▷ configurations:
    - The makeup of a system. To "configure software" means selecting programmable options that make the program function to the user’s liking. (pcmag)
    - In communications or computer systems, a configuration of a system refers to the arrangement of each of its functional units, according to their nature, number and chief characteristics. (wikipedia)
- Why "Databases and configs"?
  - ▷ CDB "everywhere" around, clear access
  - ▷ possibility for versioning
  - ▷ access to configs from different contexts
    - online/DAQ - reco - analysis
  - ▷ MEG(2) does it (?)
- Wrong approach (imnsho!)
  - ▷ config versioning and access should use tagged/checked-out releases

## Job configurations in the CDB?

- Technical base for discussion
  - ▷ despite mnsho
- Updated data model
  - ▷ configs in 2 flavors
    - "JSON" files in sim
      but with ".include" keys
    - "conf" files in trirec
    - not optimal (imnsho)
  - → unfortunate terminology!
  - ▷ both configs in CDB
    - stored as string
    - with minor metadata
    - payload differs in backends



```
{
  "BOR": {
      "Run number",
      "Start time",
      "Subsystem",
      "Beam",
      "Shift crew"
  },
  "EOR": {
      "Stop time",
      "Events",
      "File size",
      "Uncompressed ...",
      "Comments"
  }
}
```

- Important: configs in/from CDB are "payloads", NOT files!
  - ▷ JSON file-based backend server (cdbJSON) does not change this!
  - → CDB configs are NOT simply JSON files!
  - ▷ Rest-API backend server (cdbRest) stores BSON files!
    (stored in mongodb)

# Job configs in the CDB? Discussion

- Possible to separate some parts of current configs into conditions
  - ▷ e.g., "detector" for
    - phase1a (2024; no recoil stations)
    - phase1b (2025; with recoil stations)
  - ▷ Would need input how to define "detector" conditions, used for
    - simulation and reconstruction
  - → This could(?) make sense, but see caveat on next page

- No provenance tracking! That still has to continue as before!

- The correct approach (imnsho)
  - ▷ possibly add another "operational" level to tags, e.g.
    v5.2 → v5.2.0, . . . , v5.2.34
  - ▷ online operations based on "release" directory, e.g.
    mu3ebe>ls mu3e
    mu3e-v5.2.0 mu3e-v5.2.1 mu3e-v5.2.2 mu3e-v5.2.3 . . .
  - ▷ possibly define ENV var ("MU3EBASE") to point to correct release
  - ▷ all configs (for filter, DQM, . . . ) pulled from one place (e.g $MU3EBASE)
  - ▷ same approach works for offline

# Dinner discussion AK/GH/UL

- Keep config files in tagged (git hashes) software release versions
  - ▷ unclear about MC simulations "detector" configurations
    - Caveat: CDB access complicates grid production
  - ▷ more configurations should be possible
    - which tracking code
    - which vertexing code
- Put MC sample configs ~~into CDB~~
  - ▷ e.g. "signal", "IC", mixed versions, . . .
  - ▷ maybe "diff" to tagged version
  - ▷ minimal setup for entries
    - key
    - version tag (preferably)
    - git hash
    - "diff"
  - ▷ file catalogs should be a separate entity

⇒ MC sample catalog (configs/samples)

```
{"cfgHash" : "cfg_trirec_mcidealv5.0",
"cfgDate" : "2024-02-02 15:27:45",
"cfgString" : "
# conf file format (based on boost info file forma
# http://www.boost.org/doc/libs/1_57_0/doc/html/bo
#
# Format:
#   '#' - comments
#   '.' - directives (example: .include \"file.con
#   'p = v' - create parameter 'p' and assign valu

trirec {

    rec_fb = 1
    rec_tl = 1

    rec_version = 5

} # trirec

"}
```

# Another dinner discussion, with NB

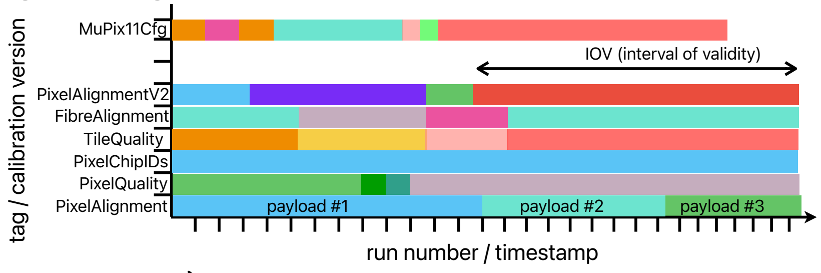
- Detector configurations
  - ▷ DACs, tunes, masks, . . .
  - → need a versioned (backed up) repository somewhere

- If this repository is implemented in the CDB:
  - ▷ caveats as for job configs plus one more (currently)
    - CDB stores "payloads", not a simple/binary/JSON flat files
    - dependency on "mu3e" repository (maybe we should move?)
  - ▷ quasi-arbitrary key naming scheme could be possible(?)
  - ▷ interface for accessing detector configs, e.g.

```
#include "Mu3eConditions.hh"
#include "cdbJSON.hh"
#include "cdbRest.hh"

// -- in your executable
cdbAbs *pDB(0);
string gt("data2024_cosmic");
pDB = new cdbJSON(gt, "/data/mu3e/cdb/json", verbose);
// pDB = new cdbRest(gt, "http://pc11740.psi.ch/cdb", verbose);

Mu3eConditions *pDC = Mu3eConditions::instance(gt, pDB);
// ...
string sconfDet = pDC->getConfString("mutrig", "v9");
// do whatever you did with files beforehand
```



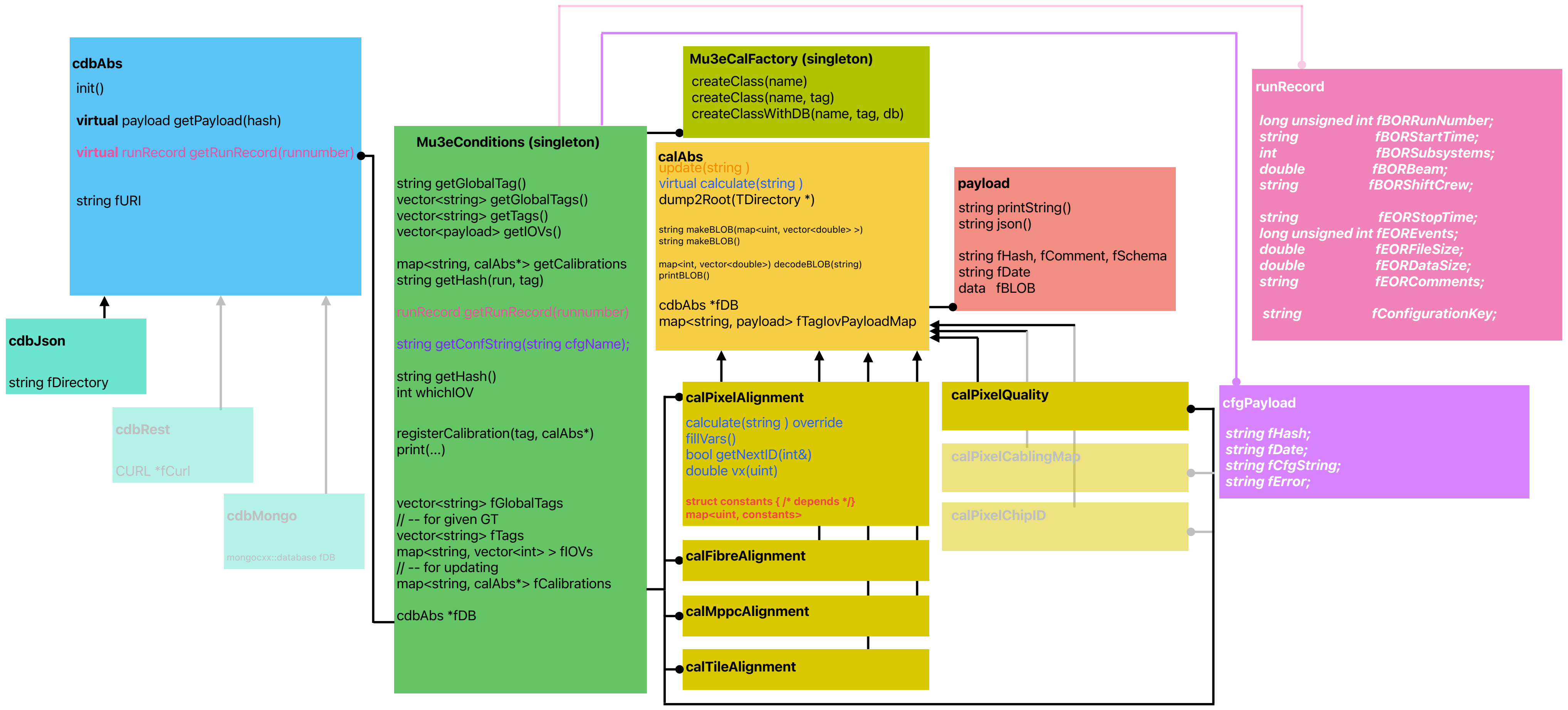
  - ▷ Raw REST api also possible ("payload" interpretation in your scripts)

# Conclusions

- Metadata is the real issue
  - (job) configs are just one aspect
  - goes beyond (strictly interpreted) CDB
- Job configs are better kept in software releases (imnsho)
  - use (git) tagging for version control
- Detector configs can be integrated into CDB
  - must consider scaling issues (bottleneck at detector initialization)
- Other metadata for long-term storage possible ~~in CDB~~
  - MC samples requests, production campaigns
  - file catalogs (versioned, historical, . . . )
- Remark about file format
  - mongodb (cdbRest) can store JSON files (binary, BSON)
  - mongodb (cdbRest) cannot store plain "conf" files
  - cbdJSON could store plain configuration files (not an option!)
- No provenance tracking! That still has to continue as before!

## HIC SVNT LEONES

# CDB with configs: classes



# Configs in CDB

- cdbJSON stores both .conf and .json in (plain) ASCII:

json/configs/cfg_detector_mcidealv5.0

```
{"cfgHash" : "cfg_detector_mcidealv5.0",
"cfgDate" : "2024-02-02 15:27:45",
"cfgString" : "
{ \"detector\" : {
  \"#\" : \"units are mm, ns, MeV, Hz, Tesla\",
  \"world\" : {
    \"length\" : 3200.0,
    \"width\" : 1500.0,
    \"height\" : 1500.0
  },
  \"phase_options\" : {
    \"0\" : \"phase 1a\",
    \"1\" : \"phase 1b\",
    \"2\" : \"phase 2\"
  },
  \"phase\" : 1,
  \"target\" : {
    \"shape_options\" : {
      \"0\" : \"double cone\",
      \"1\" : \"plane\",
      \"2\" : \"garland\",
      \"3\" : \"reverse garland\",
      \"4\" : \"two-turn garland\",
      \"5\" : \"reverse two-turn garland\",
      \"6\" : \"no target\"
    },
    \"shape\" : 0,
    \"thickness1\" : 0.075,
    \"thickness2\" : 0.085,
    \"length\" : 50.0,
    \"radius\" : 19.0,
    \"offset\" : {
      \"x\" : 0.0,
      \"y\" : 0.0,
```

json/configs/cfg_trirec_mcidealv5.0

```
{"cfgHash" : "cfg_trirec_mcidealv5.0",
"cfgDate" : "2024-02-02 15:27:45",
"cfgString" : "
# conf file format (based on boost info file format)
# http://www.boost.org/doc/libs/1_57_0/doc/html/boost_prop
#
# Format:
#    '#' - comments
#    '.' - directives (example: .include \"file.conf\")
#    'p = v' - create parameter 'p' and assign value 'v'

trirec {

    rec_fb = 1
    rec_tl = 1

    rec_version = 5

} # trirec

"}
```

$\rightarrow$ editable files

- cdbRest (accessing mongodb) obviously differs (BSON files)!

## Usage

- By default, you can get the configs from the CDB
  - ▷ according to global tag (GT)
- Change to "local" (modified) configs

```
merlin>../_build/mu3eTrirec/mu3eTrirec --cdb.dbconn=rest \
       --cdb.cfg=./cfg_vertex_new:./cfg_trirec_new  \
       directory/mu3e_sorted_000779.root directory/mu3e_trirec_000779.root
```

  - ▷ Usage:

    --cdb.cfg=/path/config

    --cdb.cfg=/path/config1:/path/config2

- Notes
  - ▷ Such (local) configs will replace original (CDB) configs
  - ▷ Same for local calibration payloads

    --cdb.cal=/path/calPayload1:/path/calPayload2

⇒ This is an abuse of the concept of GT
  - ▷ GT handles calibrations (versions, software releases, . . . )
  - ▷ GT should not change as rapidly as configs change initially
  - ▷ could be changed to add another key for configs