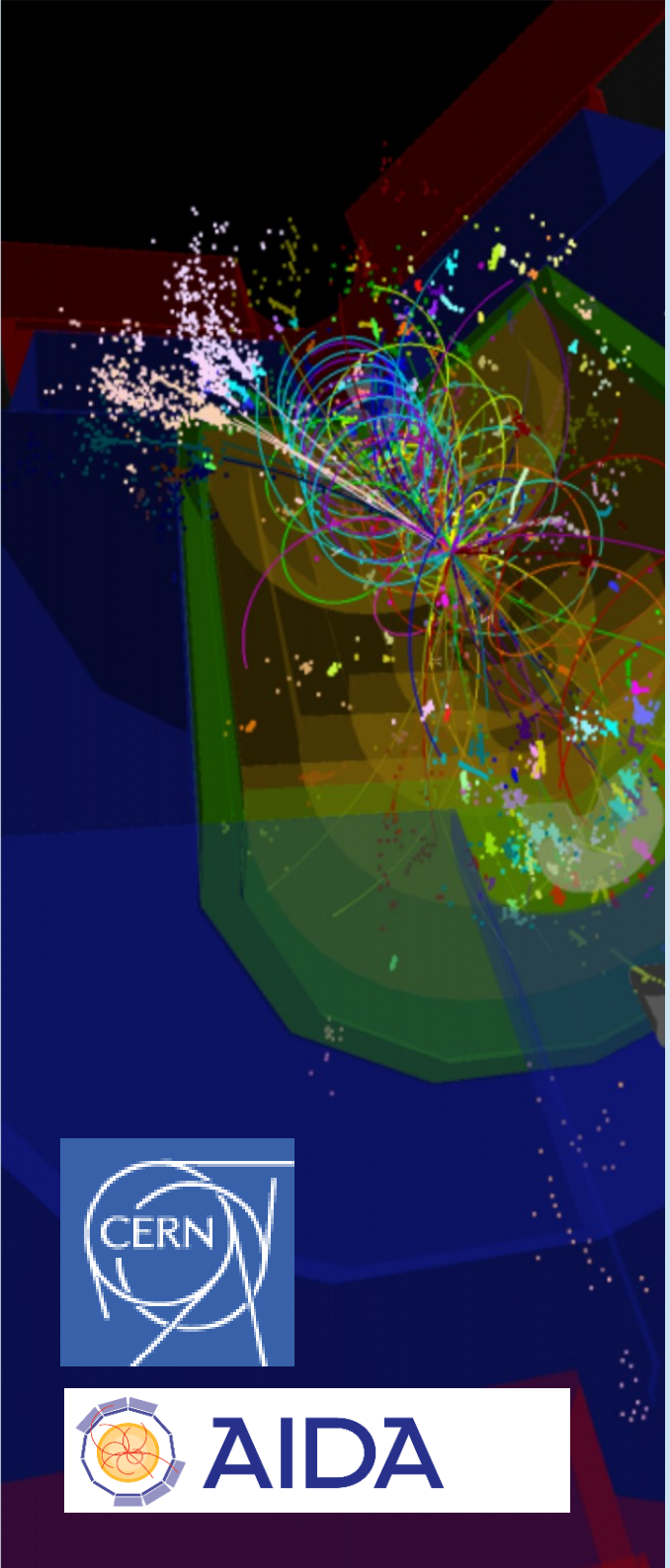
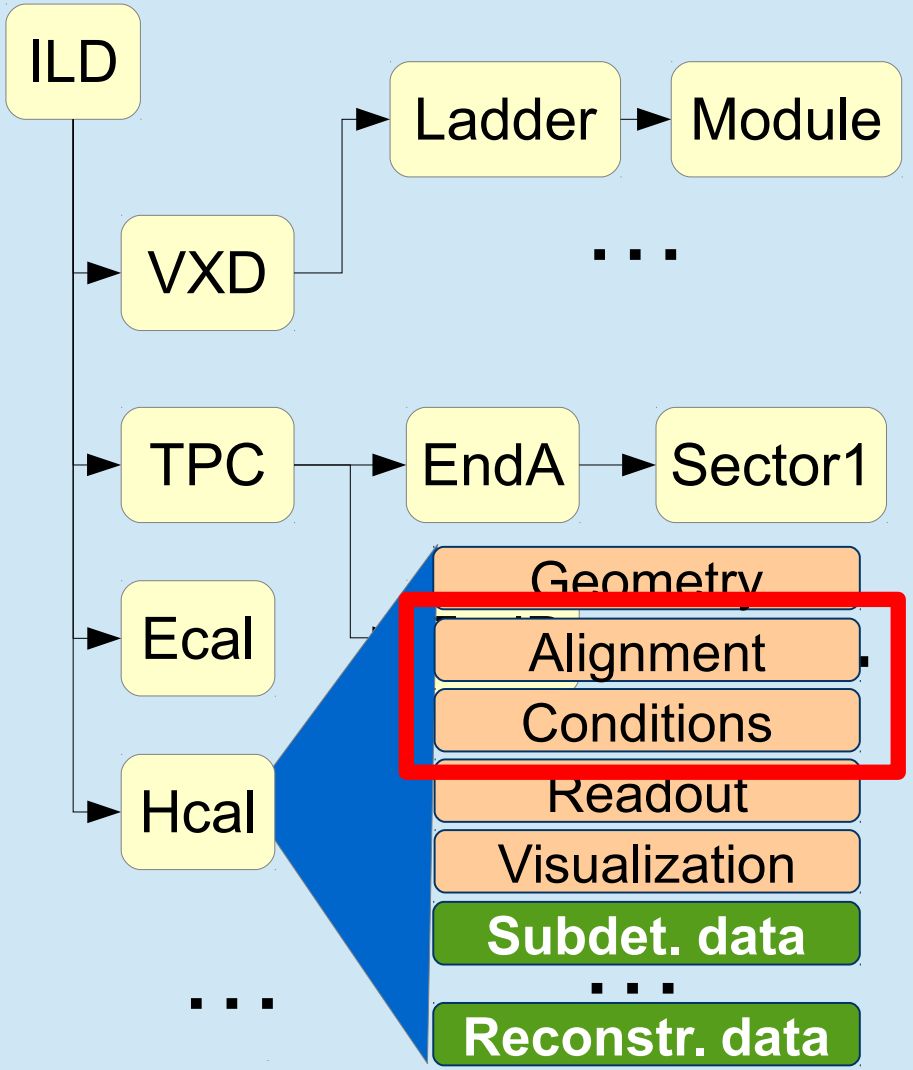


# DDCond Status

## Conditions in the Detector Description

# What is Detector Description ?

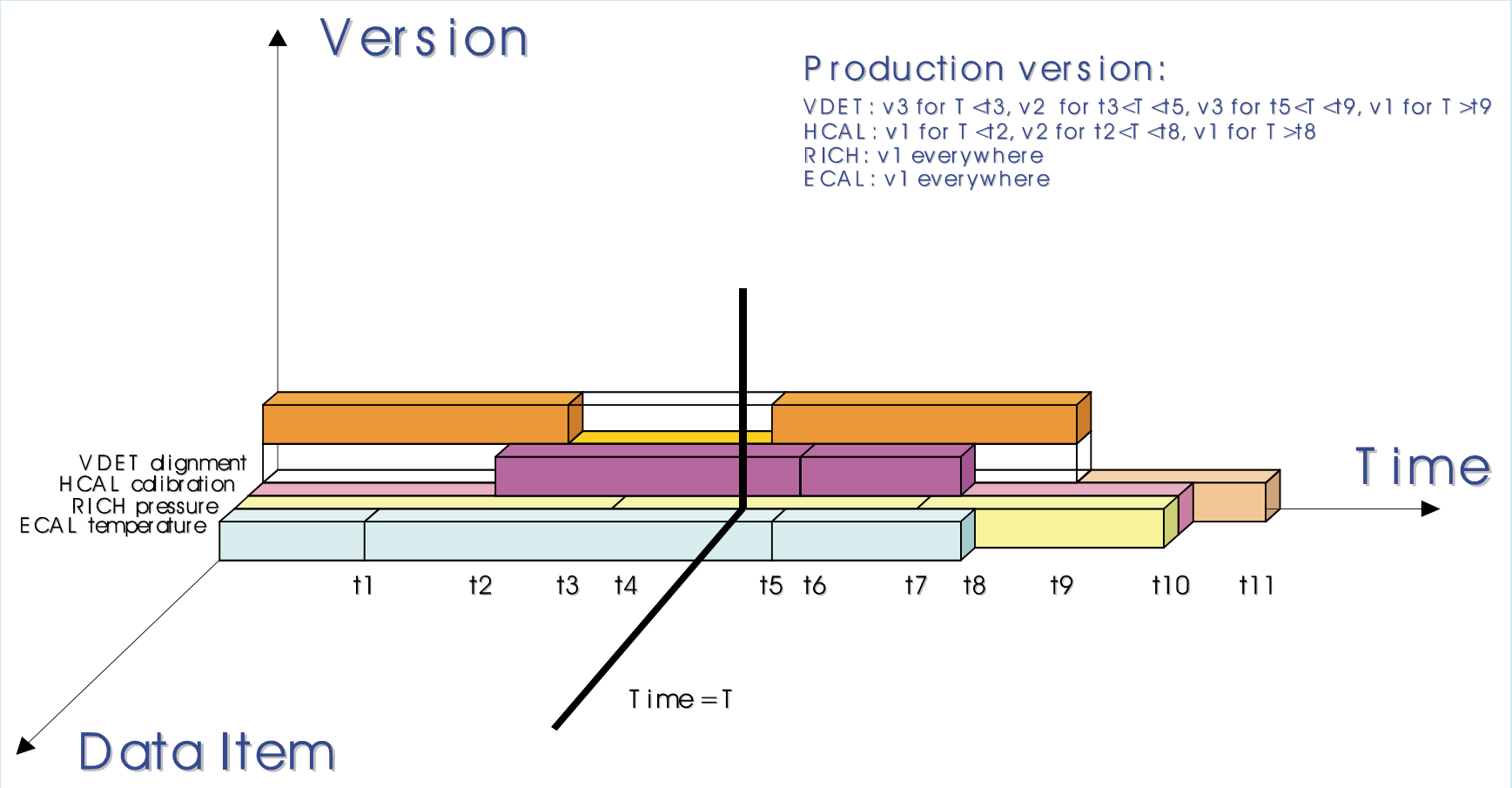
- **Description of a tree-like hierarchy of "detector elements"**
  - **Subdetectors or parts of subdetectors**
- **Detector Element describes**
  - **Geometry**
  - **Environmental conditons**
  - **Properties required to process event data**
  - **Optionally: experiment, sub-detector or activity specific data**



# DDCond: Conditions Data

- Time dependent data necessary to process the detector response [of particle collisions]
- Conditions data support means to
  Provide access to a consistent set of values according to a given time
  - Fuzzy definition of a “consistent set”
    typically referred to as “interval of validity”:
    time interval, run number, named period, ...
  - Configurable and extensible
- Data typically stored in a database

# Conditions Data: Consistent Dataset




[Pere Mato / 2000]

# DDCond: What do we want ?

- We want to provide access to consistent set of accompanying data for processing event data
  - See previous slide
- We want to be “fastest”
  - Need reasonable users
- We want to support multi-threading at it’s best
  - Not wait for flushed event pipelines before updates
    Fully transparent processing, minimal barriers
  - If we can do this, we can also expect some support from the experiment framework
- Reasonable use of resources
  - Cache where necessary but no more

# DDCond: What can we assume ?

**(when used by reasonable users)**

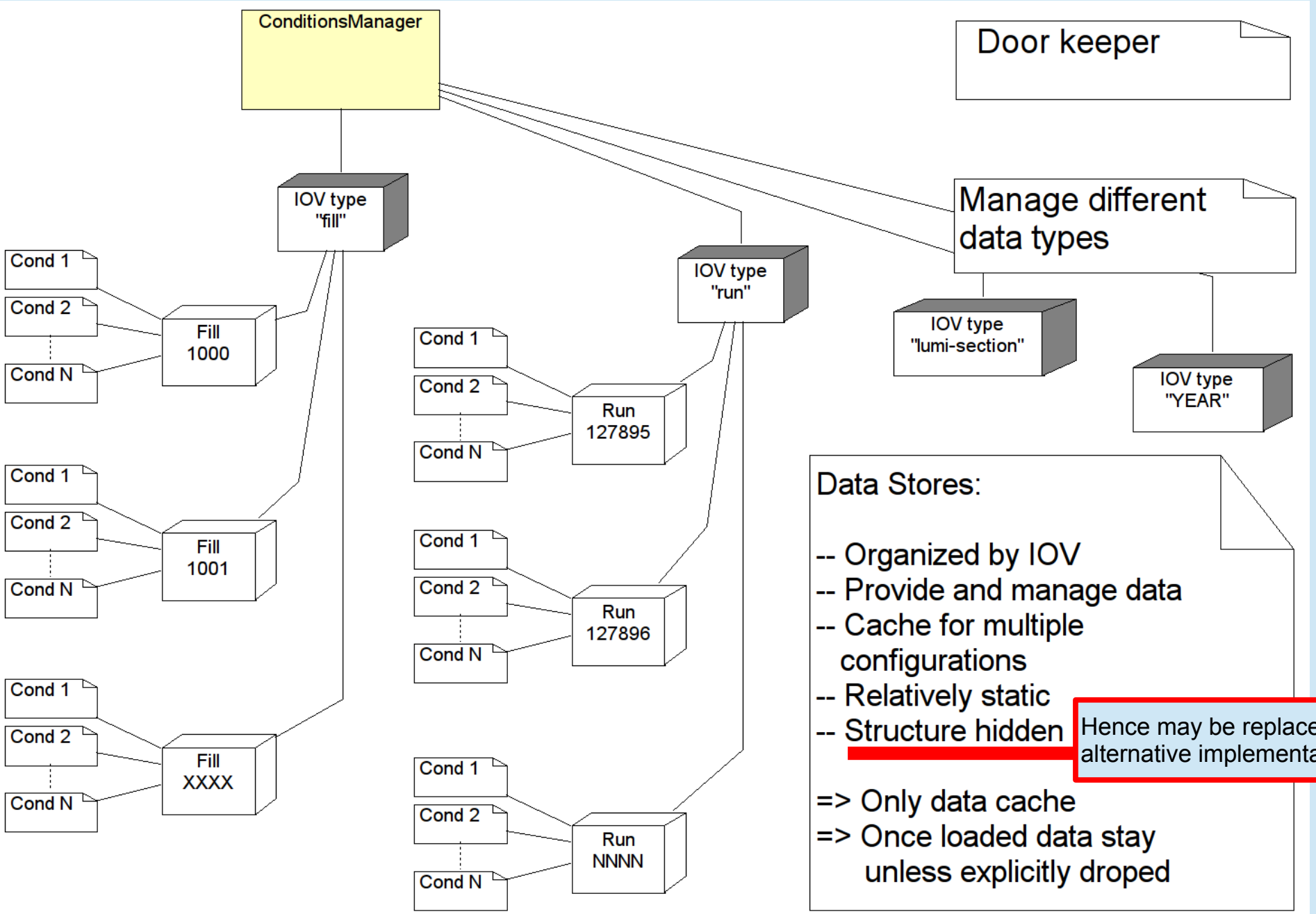
- **Conditions data are slowly changing**
  - **e.g. every run O(1h), lumi section O(10min), etc.**
- **Conditions data change in batches**
  - **Interval of validity is same for a group (subdetectors)**
  - **Not every SD defines it himself (I know, needs discipline)**
- **Conditions also are the result of computation(s)**
  - **Conditions data may also be the combination of other conditions data applied to a functional object Example: Alignment transformations from Delta-values**
  - **So-called "derived conditions" are mandatory**

# Yesterday and Today
## Change of Paradigm

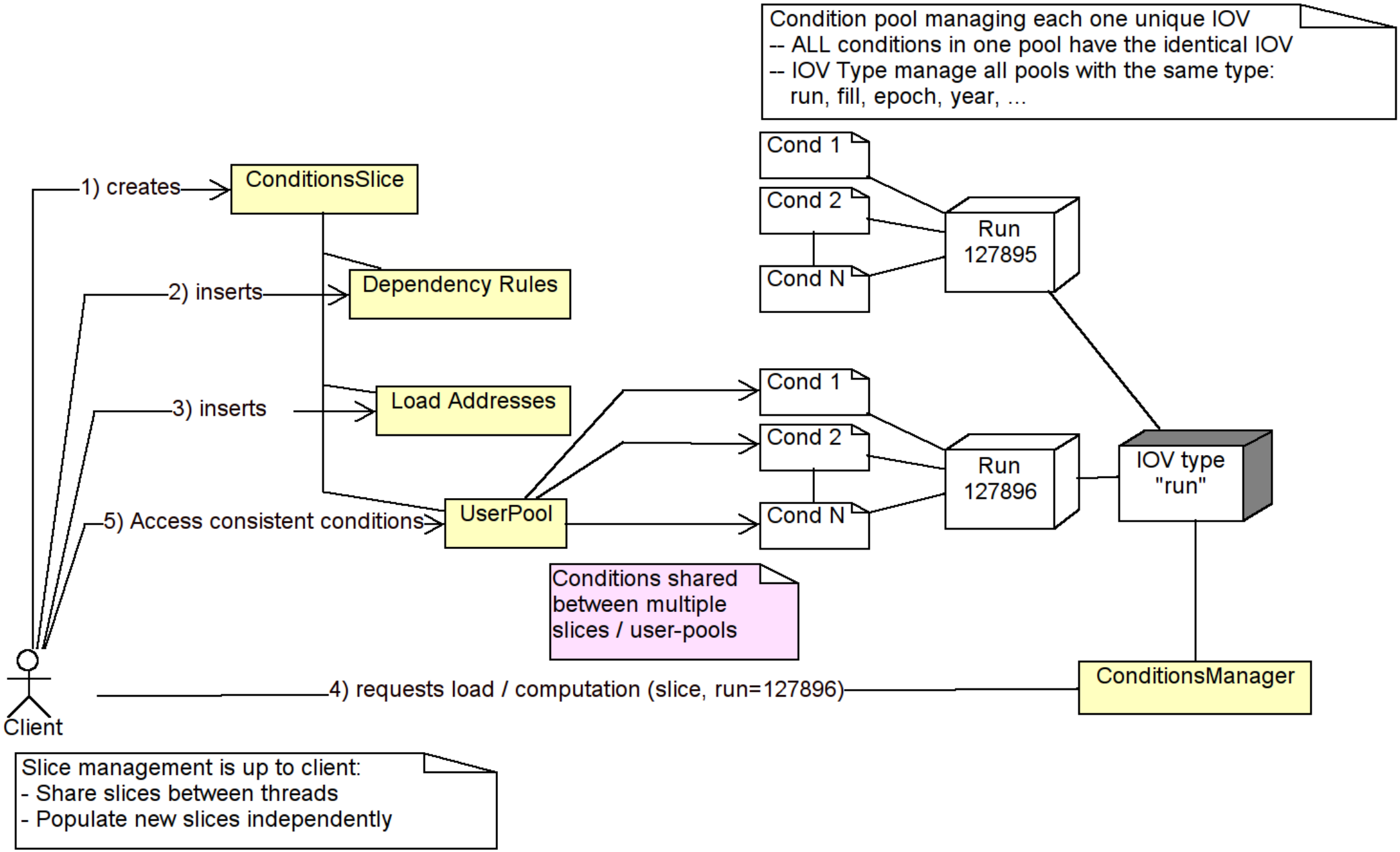
- **Historically**
  - **C++ data processing frameworks were a novelty**
  - **Emphasis on flexibility, "discovery" of the data space**
  - **Only load what users ask for**
  - **Multi-threading was no issue**

- **Today** **[no free lunch in life]**
  - **Load barriers and accessed conditions set is well specified [See for example ongoing discussions around Gaudi]**
  - **No late loading, no load on demand: minimize mutex-hell**
  - **Maybe a bit of overhead, but you gain by multi-threading**

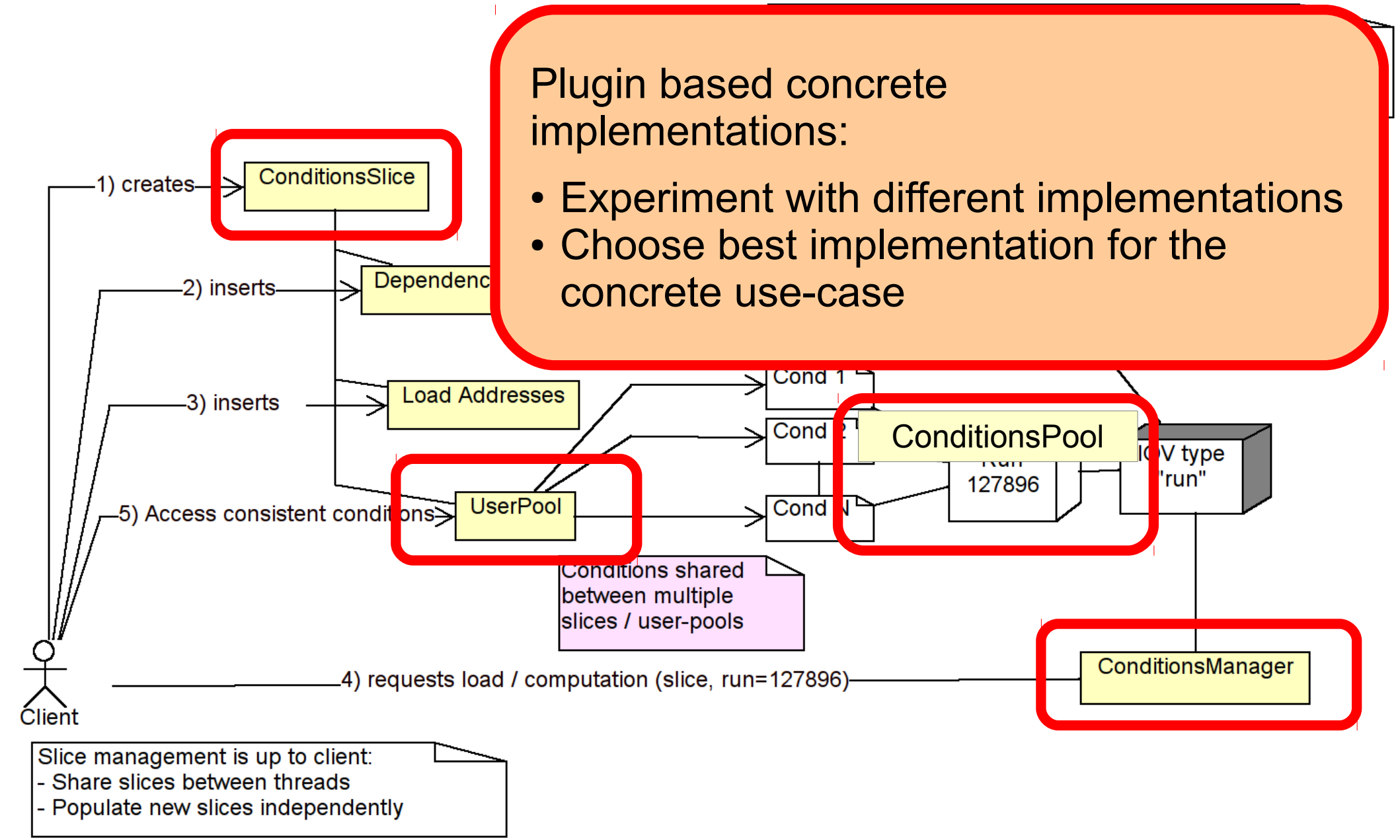
# DDCond: The Data Cache



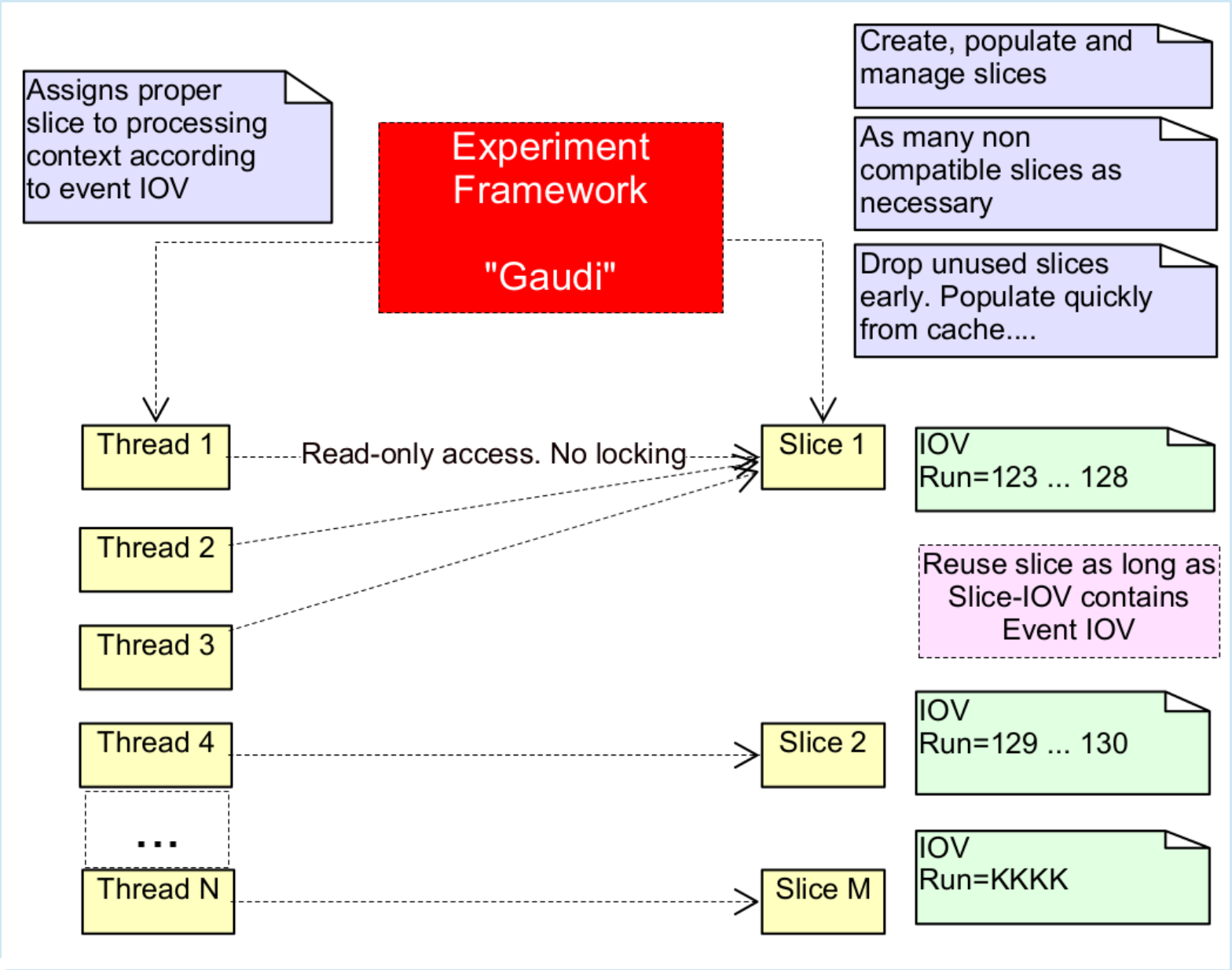
# DDCond: User Data Access



# DDCond: Flexibility where necessary



# DDCond: Framework Mode



# Pros and Cons

- **Multiple slices: No global barriers on "change-run"**

  **++ multi-threading,  ++ advanced slice preparation**

- **IOV-pools read-only after load + compute**

  **++ no locking hell for event processors, only for the loader**

- **No dependencies between IOV types (derived conditions)**

  **++speed, ++simplicity  --flexibility  (use cases ?)**

- **Many parallel IOV types are difficult to handle**

  **User problem: should limit yourself to 1,2 or 3**

- **IOV pools must be reasonably populated**

  **-- 1 condition per pool would be bad. Many is efficient...**
  **(→ need reasonable users)**

# Benchmarks and Timing (1)

- **CLICSiD example: ~ factor 5 beyond LHCb**
  - **Standard CERN desktop 2 years old, Ubuntu 16.04 32 bit**

| | |
|---|---|
| • Create 175 k conditions + registration to IOV type | ~ 0.22 s |
| • Create and select slice for<br>175 conditions + 105 k computations | ~ 0.3 s |
| • Subsequent select 280 k<br>equivalent to run-change with already loaded conditions | ~ 0.13 s |
| • Slices for (175+105) for 20 runs (total of 5.8 Mcond)<br>- Create conditions (175 k)<br>- Computations (105 k)<br>[approaching machine memory limit] | <br>~ 0.22 s/run<br>~ 0.35 s/run |

- **Looks quite scalable and quite fast**
  - **No database access nor XML parsing, but this was not part of this exercise**

# Benchmarks and Timing (2)

- **LHCb example**
  - **Standard CERN desktop 2 years old, Ubuntu 16.04 32 bit Statistics over 20 runs**

| | |
|---|---|
| • Load slice with 9353 multi-conditions from XML snapshot + registration to IOV type [Mostly XML parsing] | ~ 1.09 s |
| • Compute 2493 alignments from conditions | ~ 0.015 s |
| • Fill slice from cache | ~ 0.08 s |

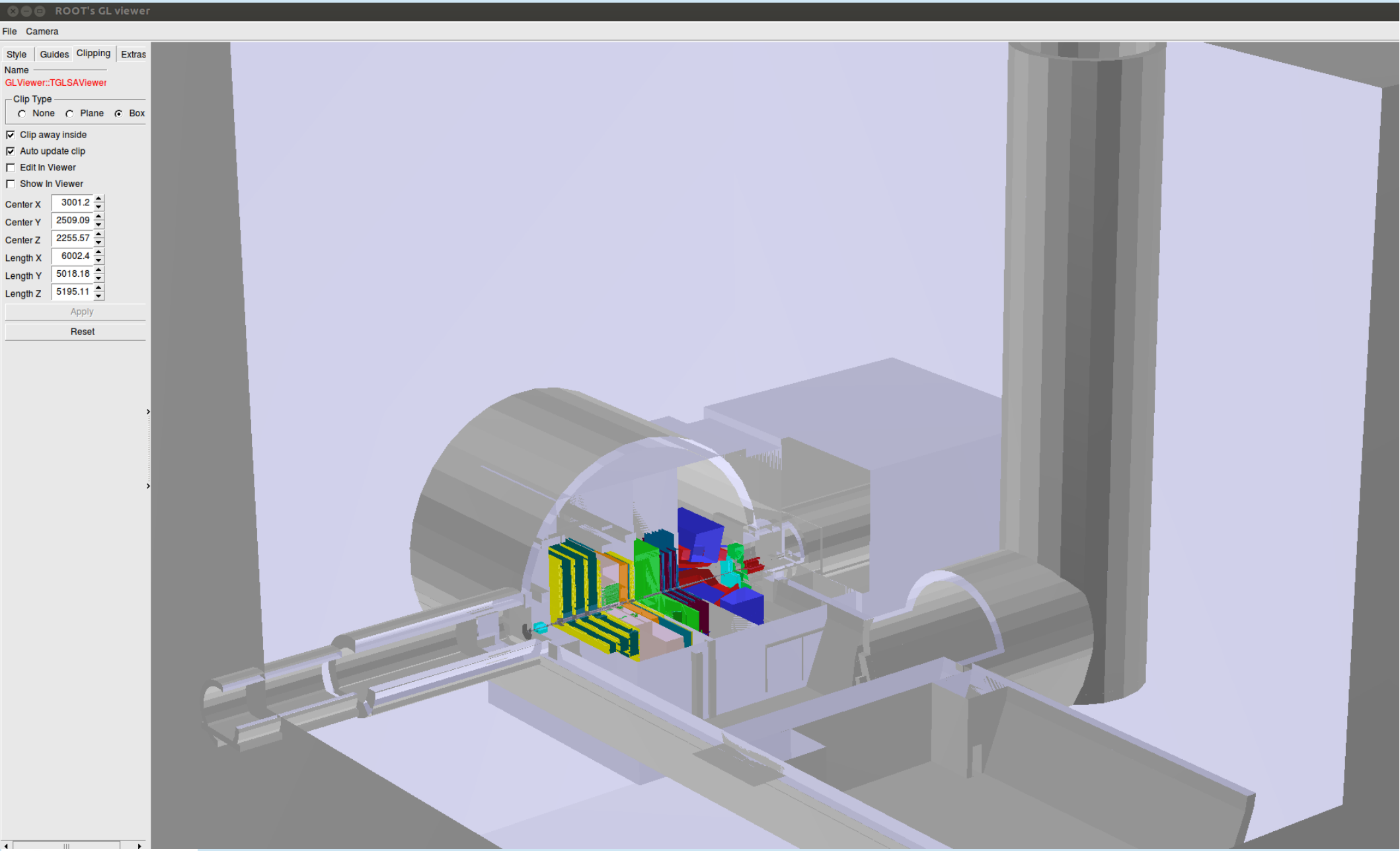
- **Subsequent accesses nearly for free, since caches are active**
- **Influence of disk cache of XML files on timing ?**

# DDAlign: Alignment Support for DD4hep

- In principle it's ready
- Will not cover here
- I also need a topic for the next meeting
  … it's simply embarrassing having nothing to show...

**Thank you for your attention!**

# LHCb Cavern


ROOT's GL viewer
File
Camera
Style
Guides
Clipping
Extras
Name
GLViewer::TGLSAViewer
Clip Type
None
Plane
Box
Clip away inside
Auto update clip
Edit In Viewer
Show In Viewer
Center X
3001.2
Center Y
2509.09
Center Z
2255.57
Length X
6002.4
Length Y
5018.18
Length Z
5195.11
Apply
Reset