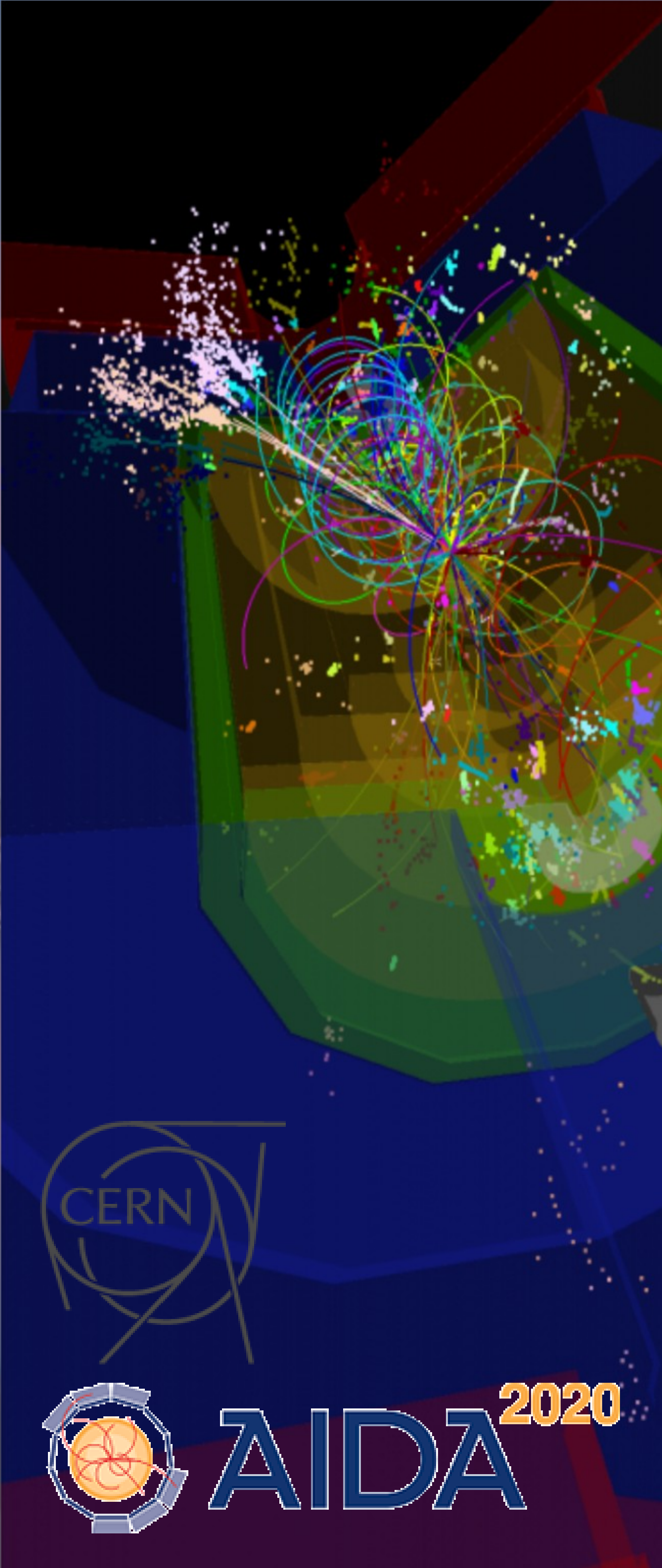


DD4hep Status

HEP detector description supporting the full experiment life cycle

M.Frank, F.Gaede, M.Petric, A.Sailer

- **Motivation and Goals**

  => Introduction / Reminders

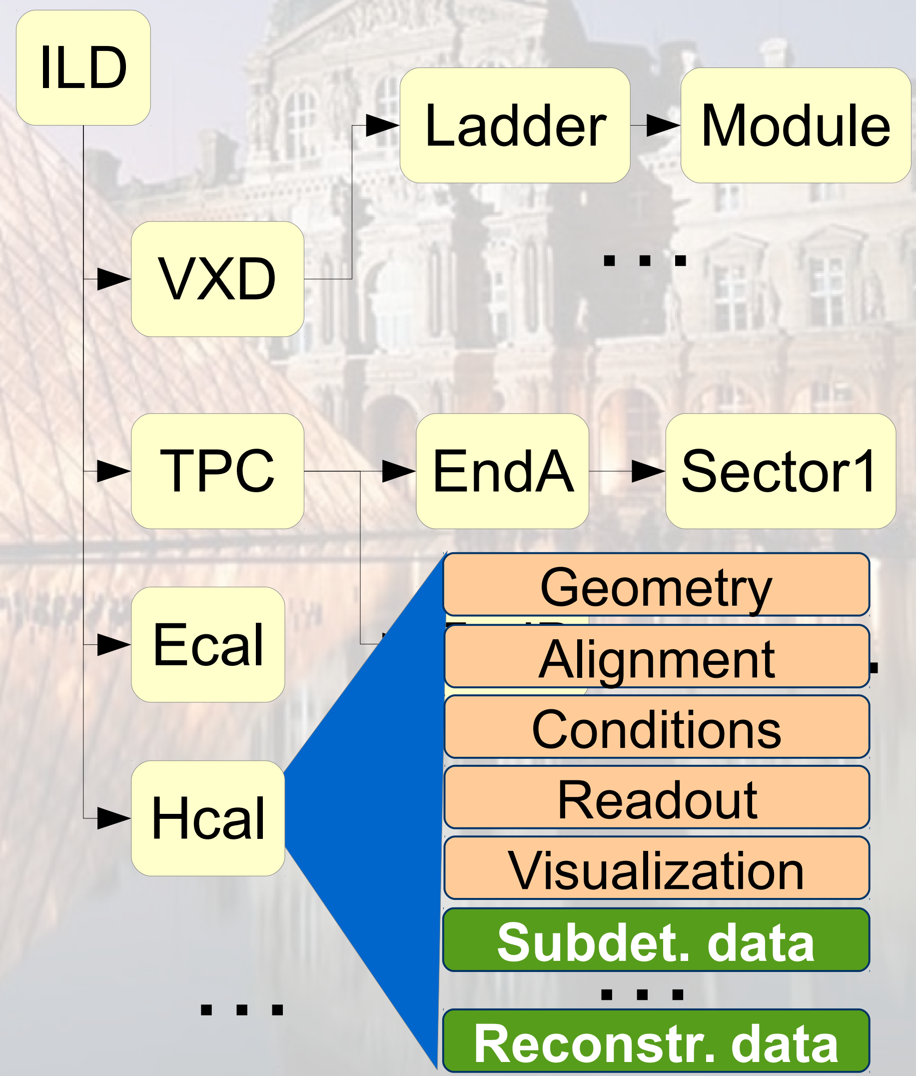
- **Simulation**
- **Conditions support**
- **Alignments support**
- **Miscellaneous**
- **Summary**

Motivation and Goal

- Develop a detector description
  - For the full experiment life cycle
    - detector concept development, optimization
    - detector construction and operation
    - “Anticipate the unforeseen”
  - Consistent description, with single source, which supports
    - simulation, reconstruction, analysis
  - Full description, including
    - Geometry, readout, alignment, calibration etc.

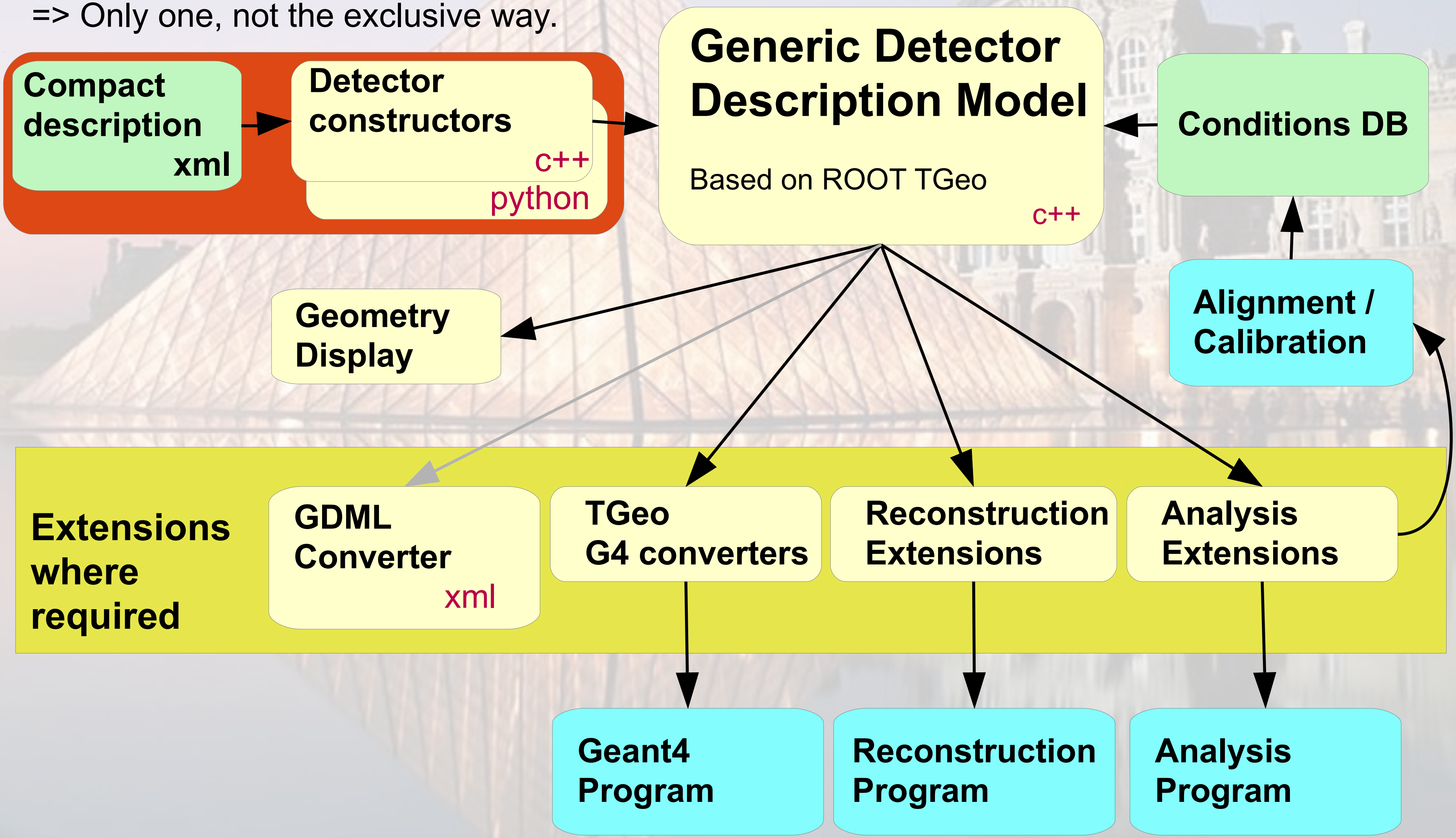
What is Detector Description ?

- **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**



DD4Hep - The Big Picture

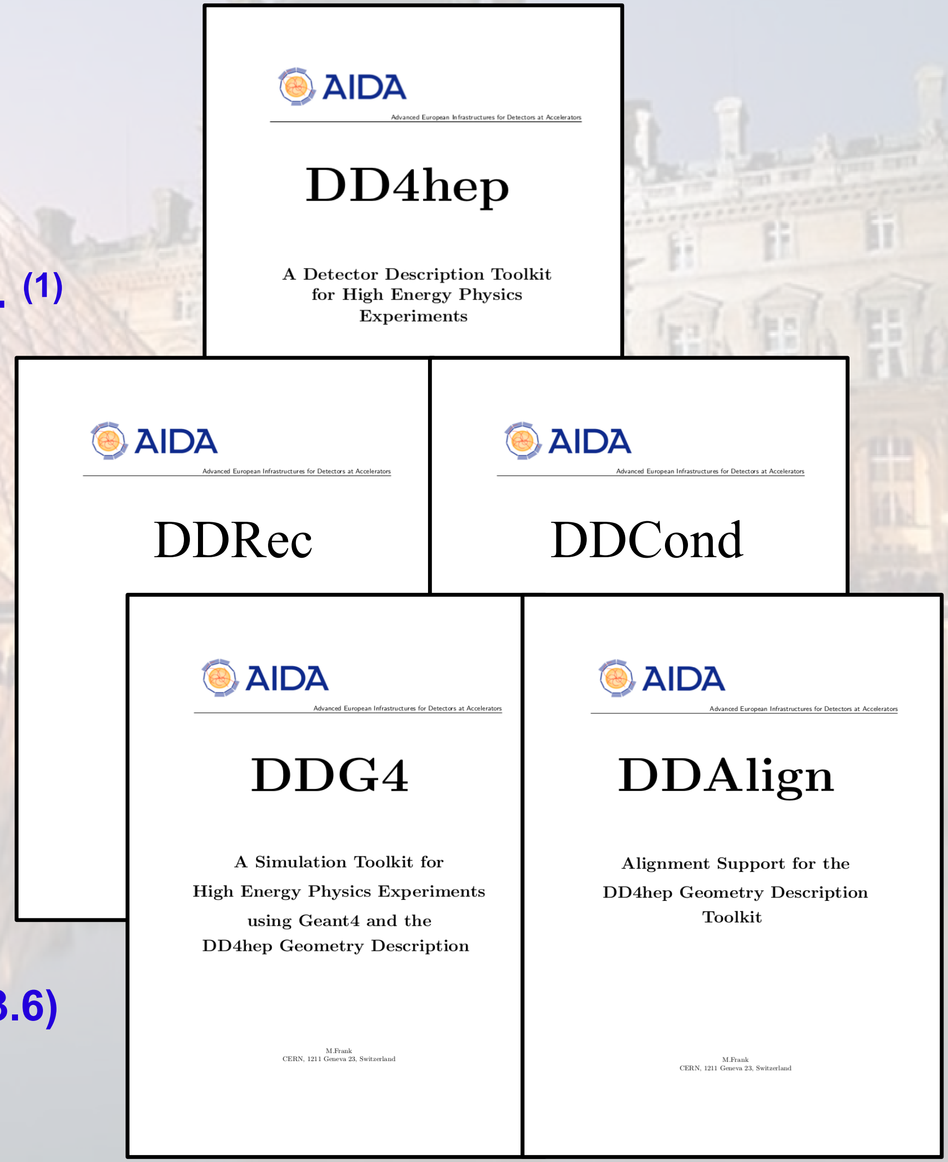
Note:
DD4hep population is plugin based
=> Only one, not the exclusive way.



Saga in 5 Episodes: Sub-packages

- **DD4hep – basics/core (1)**
- **DDG4 – Simulation using Geant4 (1)**
- **DDRec – Reconstruction supp.(2)**
- **DDAlign – Alignment support (3)**
- **DDCond – Detector conditions (3)**

(1) Bug-fixes and maintenance
(2) See presentation of F. Gaede (WP3, Task 3.6)
(3) Work since start of AIDA2020



- **Motivation and Goals**

  => Introduction / Reminders

- **Simulation**
- **Conditions support**
- **Alignments support**
- **Miscellaneous**
- **Summary**

DD4hep Core: Multiple Segmentations
Multiple Hit Collections

- **Extension component using existing interfaces**
- **From the wish-list of FCC**
- **Collection selection according to 'key' and 'key value' or 'key range'**

```
<readouts>
  <readout name="TestCalHits">
    <segmentation   type="MultiSegmentation"  key="layer">
      <segmentation name="Layer1grid" type="CartesianGridXY"    key_min="0x1" key_max="4"    grid_size_x="0.1
      <segmentation name="Layer2grid" type="CartesianGridXY"    key_value="5"                grid_size_x="0.2
      <segmentation name="Layer3grid" type="CartesianGridXY"    key_min="0x6" key_max="0xFF" grid_size_x="0.3
    </segmentation>
    <hits_collections>
      <hits_collection name="TestCallInnerLayerHits"  key="layer" key_value="0x1"/>
      <hits_collection name="TestCallMiddleLayerHits" key="layer" key_min="2"   key_max="5"/>
      <hits_collection name="TestCallOuterLayerHits"  key="layer" key_min="0x6" key_max="0xFF"/>
    </hits_collections>
    <id>system:8,barrel:3,layer:8,slice:8,x:32:-16,y:-16</id>
  </readout>
</readouts>
```

Simulation: DDG4

- Simulation = Geometry + Detector response + Physics
- Mature status
  - Eventual bug fixes, smaller improvements
- Improvements
  - Support for multiple primary vertices from a single input source
  - Multiple input sources were already supported
- Full framework used by the Linear Collider community
- Individual components used by the FCC community

DDG4: Optimization

vtunes output from 20 events $e^+e^- \rightarrow t\bar{t}$

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

Summary | Bottom-up | Caller/Callee | Top-down Tree

BEFORE | AFTER

Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Difference – Effective Time by Utilization	Spin Time	Ov.. Time	CPU Time: r005hs – Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Ov.. Time	CPU Time: r009hs – Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)
local_Rb_tree_increment	195.720s	0s	0s	195.720s	0s	0s	
std::_Rb_tree<int, std::pair<int const, int>, std::_Select	93.198s	0s	0s	93.198s	0s	0s	
std::_Rb_tree_const_iterator<std::pair<G4PrimaryPartic	53.518s	0s	0s	53.518s	0s	0s	
_dl_update_slotinfo	51.131s	0s	0s	51.131s	0s	0s	
__ieee754_atan2	36.939s	0s	0s	36.939s	0s	0s	
std::_Rb_tree<int, std::pair<int const, int>, std::_Select	32.062s	0s	0s	32.062s	0s	0s	
std::_Rb_tree<int, std::pair<int const, DD4hep::Simulati	24.100s	0s	0s	24.100s	0s	0s	
__ieee754_log	23.019s	0s	0s	23.019s		0s	
malloc	20.552s	0s	0s	20.591s		0s	0.039s
primary	17.040s	0s	0s	17.040s	0s	0s	
_int_malloc	16.347s	0s	0s	16.347s	0s	0s	
__strcmp_sse42	14.733s	0s	0s	14.753s	0s	0s	0.020s
std::_Rb_tree<int, std::pair<int const, int>, std::_Select	13.140s	0s	0s	13.140s	0s	0s	
DD4hep::Geometry::DetElement::placement	12.760s	0s	0s	12.760s	0s	0s	
DD4hep::Callback::make<DD4hep::Simulation::Geant4	9.360s	0s	0s	9.360s	0s	0s	

- **Nice example how a couple of stupid container look-ups can screw your day**
- **Now DDG4 framework overhead < 10 % including: Input, hit handling in sensitive detectors, MC truth handling, output**

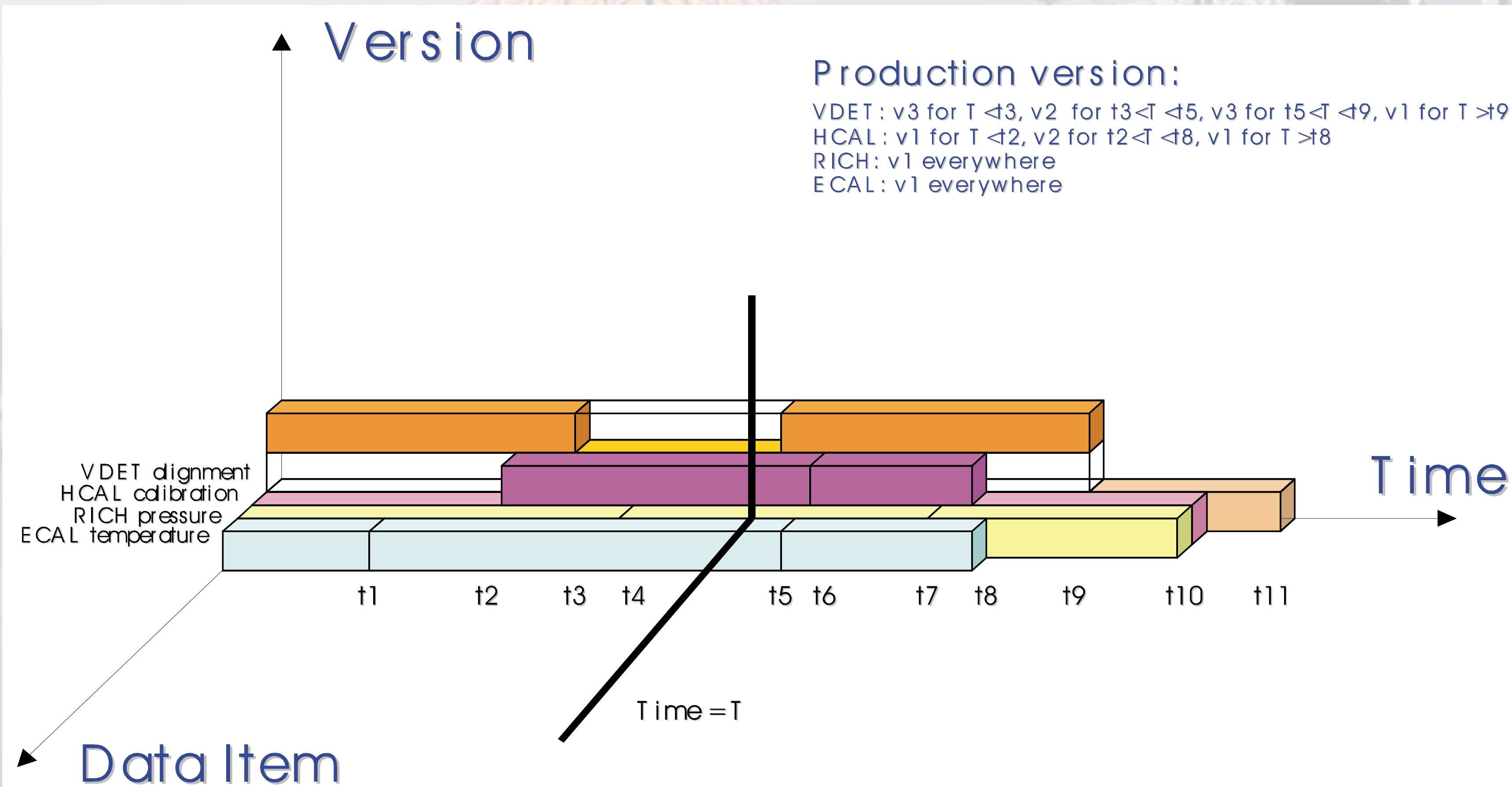
- Motivation and Goals
- Simulation
- **Conditions support**

  => DDCond
- Alignment Support
- Miscellaneous
- Summary

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 ?

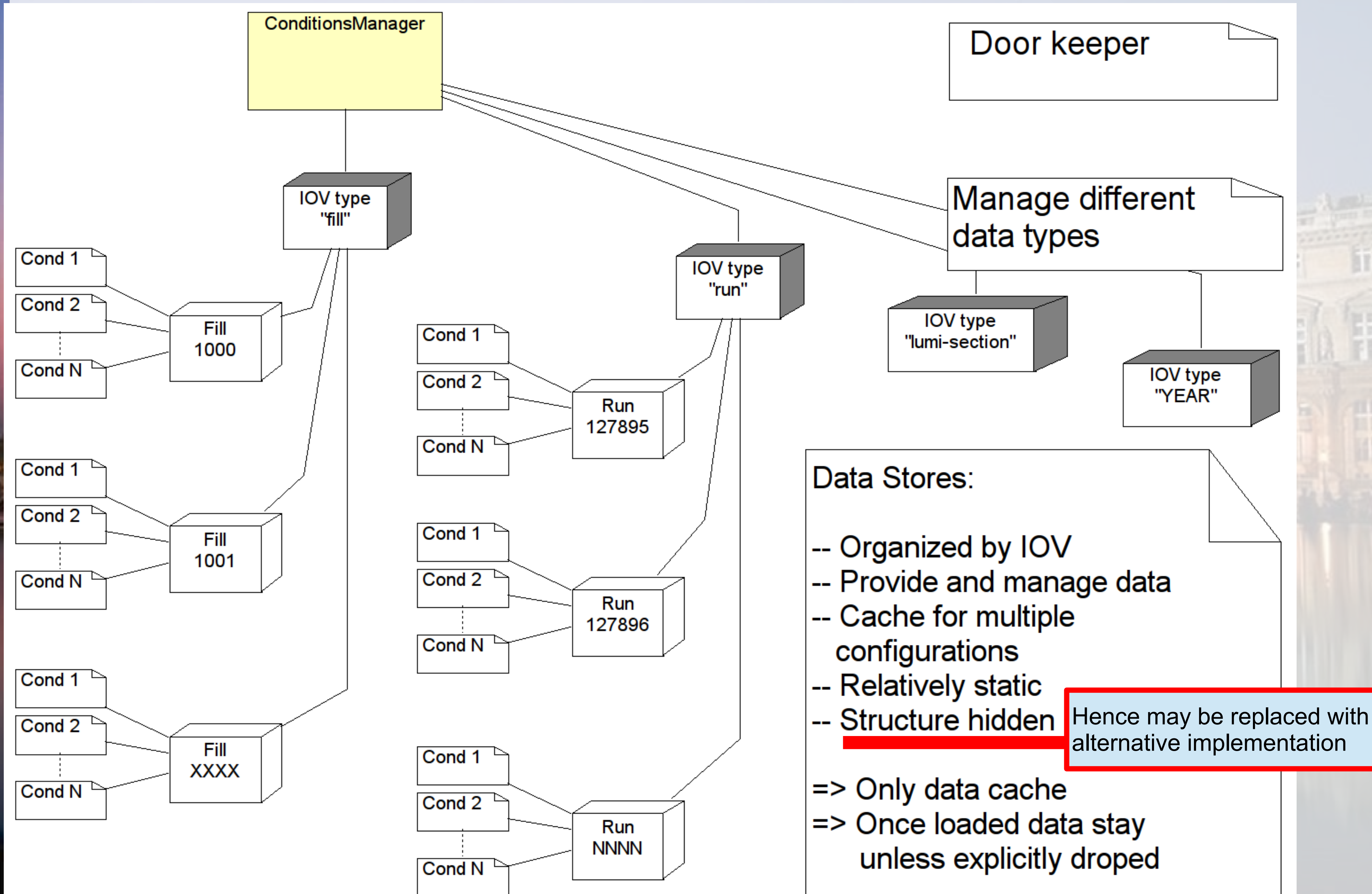
(when used by reasonable users)

- **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

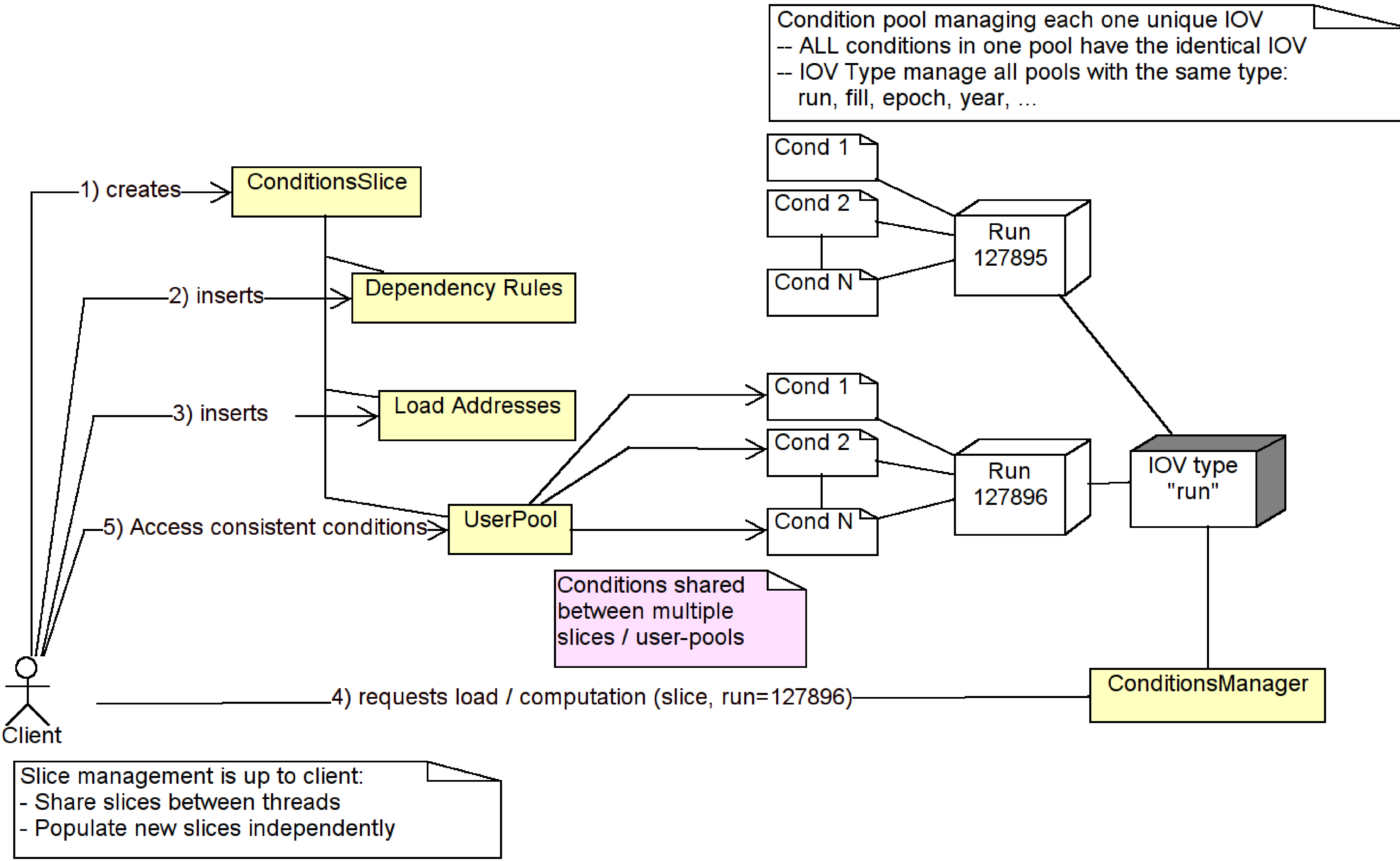
Yesterday and Today
Change of Paradigm

- **Historically**
  - C++ data processing frameworks were a novelty
  - Emphasis on flexibility, “discovery” of the data space
  - Only load what users ask for (Load-on-demand)
  - 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

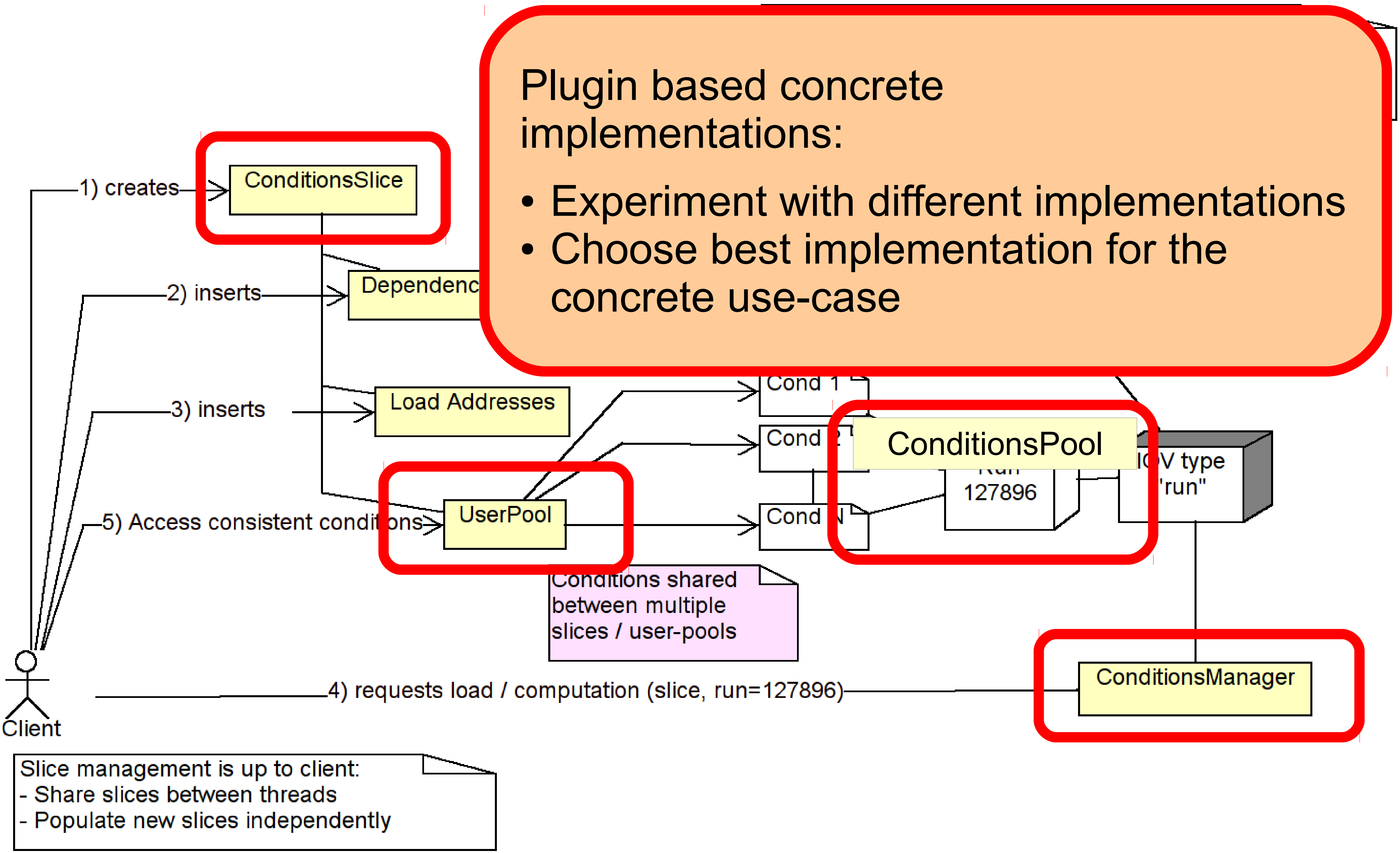
DDCond: The Data Cache



DDCond: User Data Access



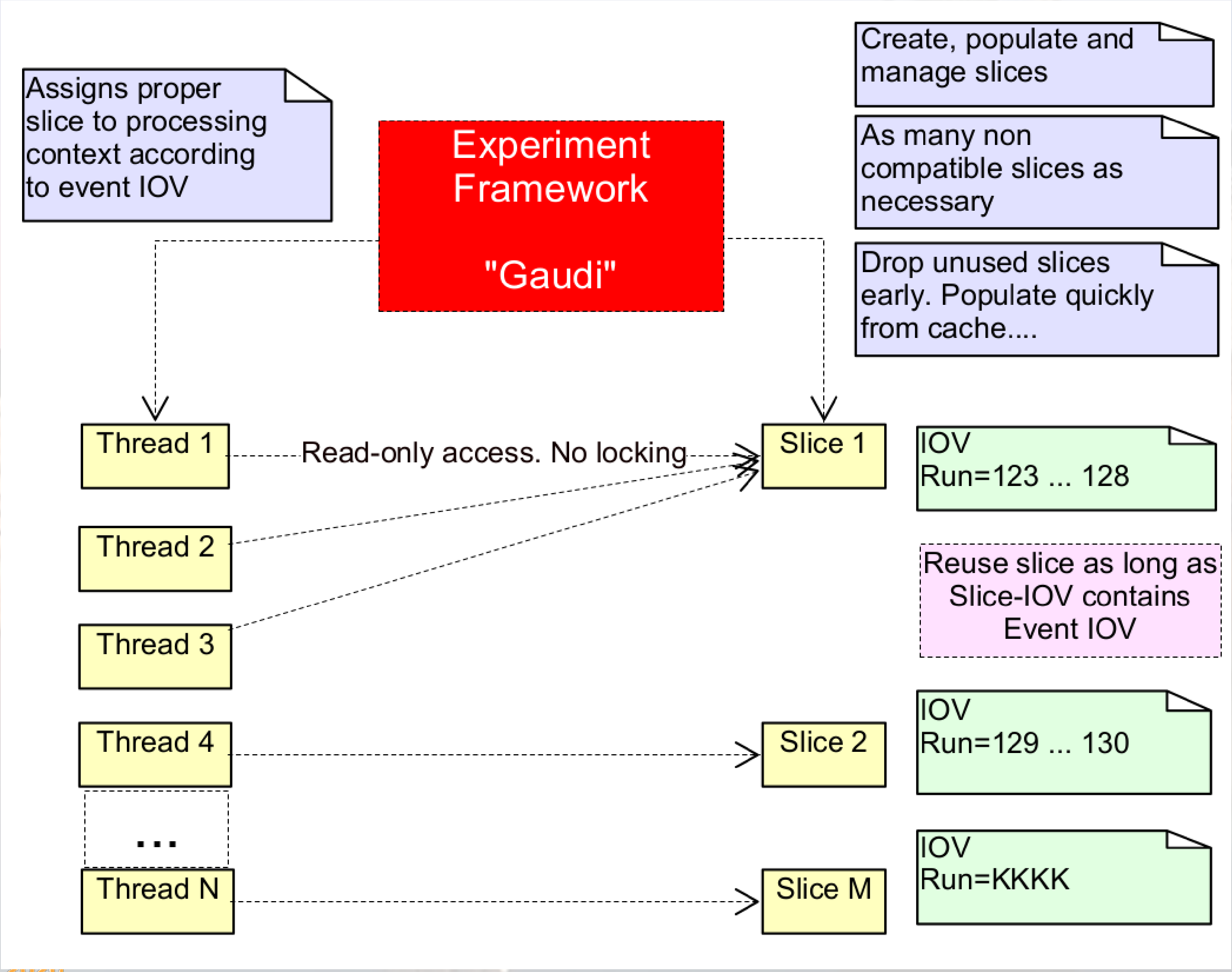
DDCond: Flexibility where necessary



Plugin based concrete implementations:

- Experiment with different implementations
- Choose best implementation for the concrete use-case

DDCond: Framework Mode



DDCond: Derived Conditions

- Data derived from conditions data are also conditions
  - Example: refractive index derived from atm. Pressure
  - Example: alignment transformations derived from Δs
  - Source may be one or multiple conditions
  - IOV is intersection of source IOVs
- Derived conditions depend on
  - Source condition(s)
  - Callback functor to perform the data transformation
- Derived condition dependencies must be registered to the projection slice
  - Computation is part of the “slice preparation”

Project Conditions Slice: Code Example

```
/// Initialize the conditions manager and set plugins (here: defaults)
ConditionsManager condMgr = ConditionsManager::from(lcdd);
condMgr["PoolType"]       = "DD4hep_ConditionsLinearPool";
condMgr["UserPoolType"]   = "DD4hep_ConditionsMapUserPool";
condMgr["UpdatePoolType"] = "DD4hep_ConditionsLinearUpdatePool";
condMgr.initialize();

/// Register IOV type used to define IOV structures
const IOVType* iov_type_run  = condMgr.registerIOVType(0,"run").second;

/// Create the conditions slice
ConditionsSlice* slice = new ConditionsSlice(condMgr);

/// Define slice content ....... (see next slide)

/// Now compute the conditions according to one IOV
IOV req_iov(iov_type_run,<specific value>);
/// Attach the proper set of conditions to the user pool
ConditionsManager::Result r = condMgr.prepare(req_iov,*slice);
```

Define Conditions Slice Content

```
/// Use the created conditions slice
ConditionsSlice* slice = ...


/// Register required DATA condition using key:
ConditionKey key("Some-global-identifier");
slice->insert(key,LoadInfo("Persistent-location-where-to-find-data"));


/// Register derived condition recipe:
/// – Depends on data from condition identified by "key": May be many!
/// – Uses "MYConditionUpdateCall" for the data transformation
ConditionsUpdateCall* call = new MYConditionUpdateCall();
ConditionKey          target_key("Some-other-global-identifier");
DependencyBuilder     builder(target_key, call);
builder.add(key);     /// Derived condition depends on "key"
slice->insert(builder.release());
```

Conditions Access from the DetElement

- So far we defined the mechanism to manage conditions
- But we also need a friendly user interface for clients
  - This is all DD4hep is about
  - Hide details in the framework, expose simplicity to users
  - Framework may also mean “experiment framework”
    Expect a bit of support as long as real users are not affected
- Conditions are accessed by key from the detector elements in the hierarchy
  - Keys are encrypted from a user defined path (e.g. address)
  - Or an alias name such as “Alignment”, “Pressure” etc.
- Let’s move on to the code examples

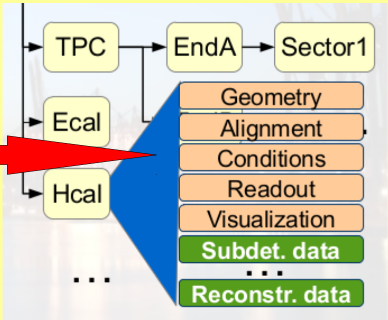
Conditions Data: Dynamic Binding

- Any data may be bound to a condition object
  - If size < 64 bytes data aggregated in condition object
  - Otherwise from heap
  - May use boost::spirit grammar definitions
- Data access for both cases:

```
/// Creator case: Create conditions object and bind the conditions data
Condition cond(name,type);
double& pressure = cond.bind<double>();
pressure = 981 * hPa;

/// Client case: access the conditions data using a projected slice
ConditionsSlice* slice = ...
ConditionsKey    key(name);
Condition        cond = slice->pool->get(key);
double& pressure = cond.get<double>();
```

Conditions Access: Code Example

```
// We need access to a projected slice
ConditionsSlice* slice      = ...
DetElement       detElement = ...

if ( detElement.hasConditions() )  {
  // Use specialized DetElement view (facade) to access conditions
  DetConditions dc(detElement);
  ConditionKey  pressure("/world/TPC/EndA/Sector1#pressure");
  // Could also map "Pressure" to "/world/TPC/EndA/Sector1#pressure"
  // as alias to local DetElement namespace!

  // Access the condition by key from the container
  Conditions::Container container = dc.conditions();
  Condition cond = container.get(pressure.hash,*slice->pool);

  // Access conditions data using dynamic type binding
  const vector<int>& table = cond.get<vector<int> >();
  ...
}
```



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 175 conditions + 105 k computations	~ 0.3 s
• Subsequent select 280 k equivalent to run-change with already loaded conditions	~ 0.13 s
• Slices for (175+105) for 20 runs (total of 5.8 Mcond) - Create conditions (175 k) - Computations (105 k) [approaching machine memory limit]	 ~ 0.22 s/run ~ 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 ?**

DDCond: Status

- Described functionality is implemented
  - Tested with xml-input
  - Interfaced to LHCb conditions database for performance tests
- Prerequisite for the development of the handling of (mis-)alignments
- Documentation to be written
- No persistency implementation envisaged besides simple xml
  - Flux in the LHC community: COOL to be retired
  - If required adapt to coming database plugins[1]

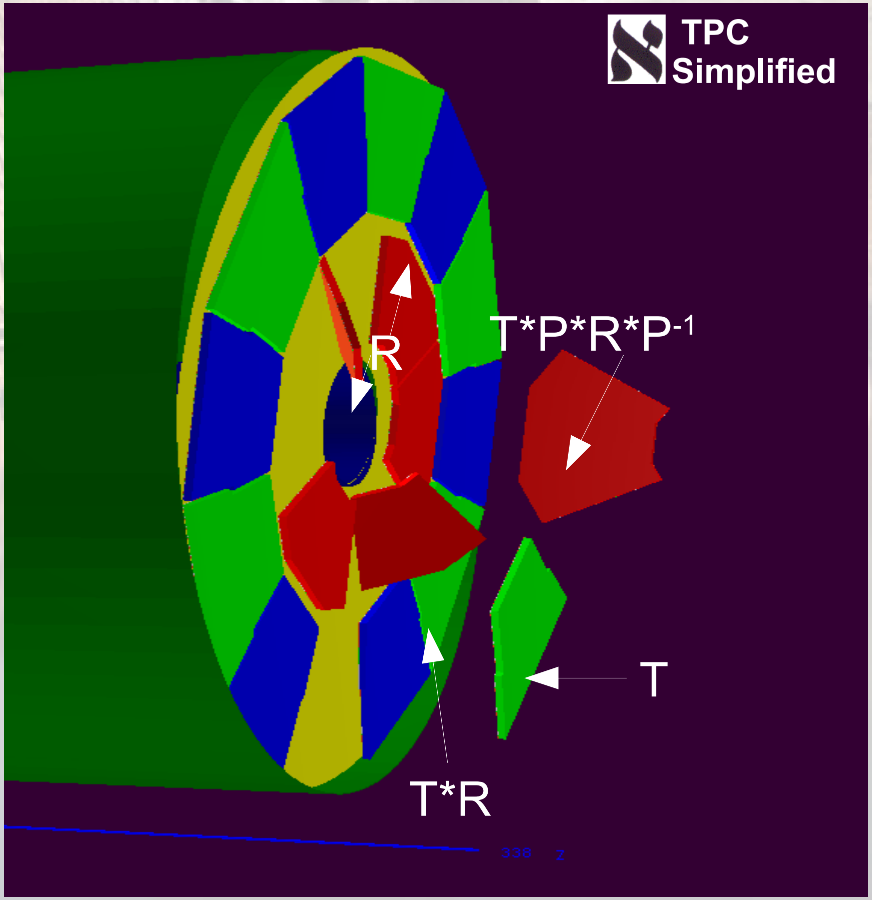
[1] See also presentation from H.Grasland (AIDA^{2020}, WP3, Task 3.4)

- Motivation and Goals
- Simulation
- Conditions support
- **Alignment Support**

  => DDAlign
- Miscellaneous
- Summary

DDAlign: Detector Alignment

- **Fundamental functionality to interpret event data**
  - **Model mis-placement by construction**
    - **Non-ideal mounts of detector components**
  - **Must handle imperfections**
    - **Geometry => (Mis)Alignment**
  - **Anomalous conditions**
    - **Pressures, temperatures**
    - **Contractions, expansions**

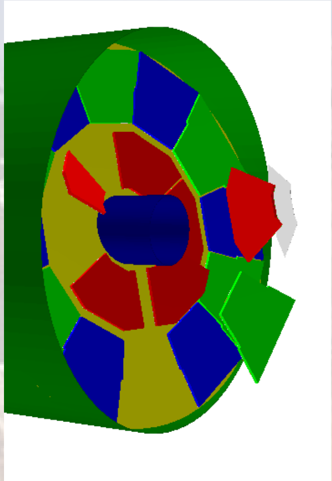


DDAlign: Standard Disclaimer

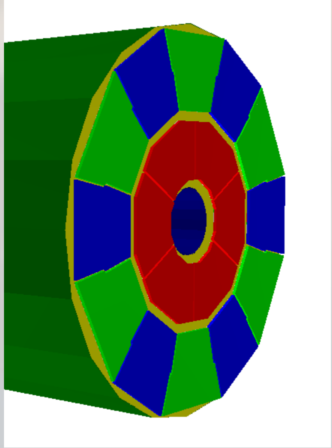
DDAlign does not provide *algorithms*[(1)] to determine alignment constants and never will.
DDAlign supports hosting the results of such algorithms and applies the resulting imperfections

(1) Algorithms are provided by WP3, Task 3.3 (C. Parkes et al.)
DD4hep (WP3, Task 3.2) collaborates with Task 3.3, but does not intend to interfere.
Milestone: MS 40 (31/01/2017)
Report: http://cds.cern.ch/record/2243542/files/AIDA-2020-MS40.pdf

DDAlign: Global and Local Alignments



- Global alignment corrections
  - Physically alters geometry
  - Intrinsic support by ROOT
  - By construction not multi-threaded
  - Possibility to simulate misaligned geometries
- Local alignment corrections
  - Geometry stays intact (either ideal or globally aligned)
  - Multi-threading supported, multiple versions
  - Local alignment corrections are conditions
  - Provide matrices from ideal geometry to world e.g. to adjust hit positions
- Support both, emphasis on local alignment



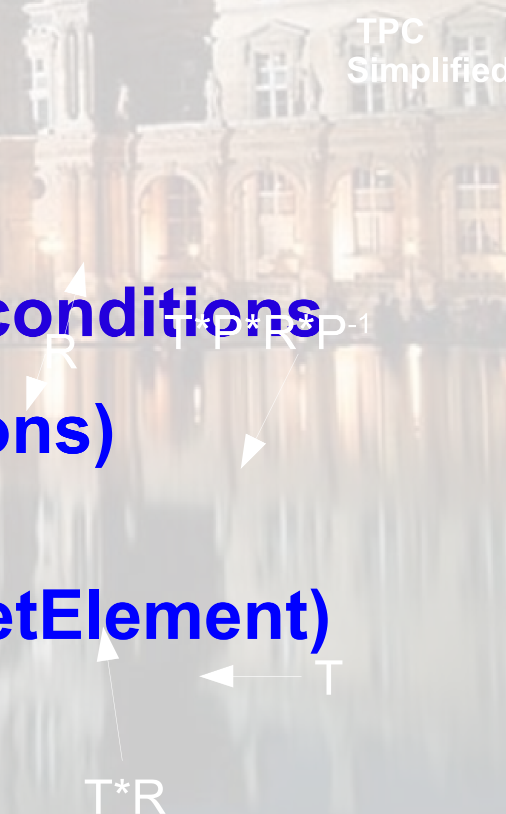
DDAlign: Global Alignments

- Interface implemented using TGeo:
  class TGeoPhysicalNode
- DD4hep interface needs revisiting
  - Implementation looks OK
  - Interface to load Δ – parameters from xml needs some adjustments
- Usage for iterative alignment purposes questionable
  - It was never foreseen in TGeo to reset an existing alignment and load new Δ – parameters(1)
- Was put on hold to support multi-threading
  - Requires “Local Alignments”

(1) private communication, A. Gheata, co-author of the ROOT Geometry Toolkit

Local Alignments and Conditions

- Local alignments data are conditions
  - Valid only for a certain time interval (IOV)
- Management is identical
  - Managed in pools
  - Access by slices
- Alignment transformations are derived conditions
  - Condition: Δ – parameters (corrections)
  - Derived: transformation matrices (to world or to hosting DetElement)



DDAlign: Alignment Corrections (Δ - Parameters)

```
class Delta   {
public:
  typedef Translation3D Pivot;
  Position       translation;
  Pivot          pivot;
  RotationZYX    rotation;
  unsigned int   flags = 0;
  ...
};
```

```
/// Initializing constructor
Delta(const Position& tr)
  : translation(tr), flags(HAVE_TRANSLATION) {}
/// Initializing constructor
Delta(const RotationZYX& rot)
  : translation(), rotation(rot), flags(HAVE_ROTATION) {}
```

- **Transformation matrix between two volumes is**
  - **Rotation**
  - **Or a rotation around pivot point**
  - **Followed by a translation**
  - **Combination**
- **Use hints for faster computation (flags)**

DDAlign: Apply Δ - Parameters

$$Tr_{Sec\,1}^{World} = Tr_{EndA}^{World} \times \left(Tr_{Sec\,1}^{Parent(EndA)} + \Delta_{Sec\,1} \right)$$

$$Tr_{EndA}^{World} = Tr_{TPC}^{World} \times \left(Tr_{EndA}^{Parent(TPC)} + \Delta_{EndA} \right)$$

$$Tr_{TPC}^{World} = Tr_{ILD}^{World} \times \left(Tr_{TPC}^{Parent(ILD)} + \Delta_{TPC} \right)$$

- Trickle-up the hierarchy and compute the matrices the most effective way
- Re-use intermediate results

Alignment handling: Code example

(see examples/AlignDet/src/*.cpp for the detailed usage of this code-fragments)

```
lcdd.fromXML(input);          // First we load the geometry

ConditionsManager condMgr  = ConditionsManager::from(lcdd);
AlignmentsManager alignMgr = AlignmentsManager::from(lcdd);

// Load delta parameters: Use here simple plugin
char* deltas[] = {"Delta-Params.xml"};
lcdd.apply("DD4hep_ConditionsXMLRepositoryParser",1,deltas);

// Project required conditions into conditions slice
IOV iov(iov_type_run,1500);   // Project conditions for run 1500
ConditionsSlice* slice = createSlice(condMgr, *iov_typ);
ConditionsManager::Result cres = condMgr.prepare(iov, *slice);

// Register callbacks to transform Delta to matrices
 ...

// Compute the tranformation matrices
AlignmentsManager::Result ares = alignMgr.compute(*slice);
```

Support for Alignment Calibrations

- Common activity with WP3 Task 3.3 (C.Burr et al.)
- Development of facade object to simplify
  - the access,
  - the modification and
  - the management of alignment corrections for calibration processes
- Functionality
  - Bulk buffering and application of Δ-parameters followed by re-computation of the transformation matrices

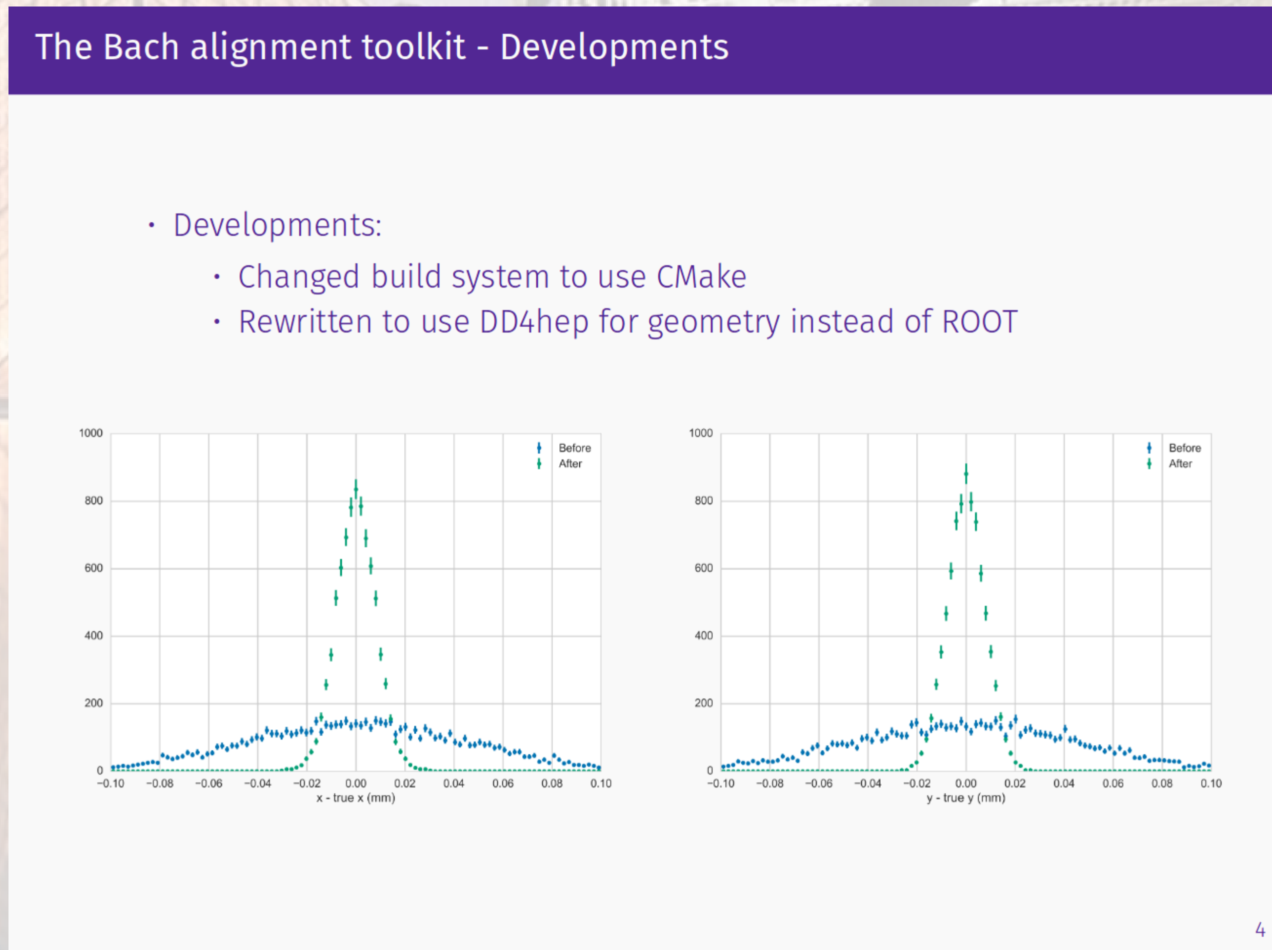
Alignment Calibrations: Code Example

(see examples/AlignDet/src/AlignmentExample_align_telescope.cpp for details)

```
/// Use the created (and projected) conditions slice
ConditionsSlice* slice = ...
/// Create calibration object.
AlignmentsCalib calib(lcdd,*slice);
/// Update call may be specialized. Hence, no default
calib.derivationCall = new DDAlignUpdateCall();
/// Attach to DetElement placements to be re-aligned
Alignment::key_type key_tel = calib.use("/world/Telescope");
Alignment::key_type key_m1  = calib.use("/world/Telescope/module_1");
calib.start();  // Necessary to enable dependency computations!

/// Let's “change” (re-align) some placements:
Delta delta(Position(333.0,0,0));
calib.setDelta(key_tel,Delta(Position(55.0,0,0)));
calib.setDelta(key_m1,Delta(Position(333.0,0,0),Rotation(pi/2,0,0));

/// Push delta-parameters to the conditions objects
calib.commit(AlignmentsCalib::DIRECT);

/// Now all alignment conditions have the updated delta parameters.
/// All marices of the derived conditions are updated!
```

Alignment: Results

- **DD4hep and alignment tools now used by Bach**
- **Please see presentation of C. Burr et al.**
- **MS40 (report)**



DDAlign: Status

- Implemented Global and Local (mis-)alignment
  - xml parser for Global (mis-)alignment constants needs re-visiting
- Started to integrate Local misalignments with the alignment procedures developed within WP3, Task 3.3
  - MS40: Running prototype for alignment Toolkit
  - To be tested in "real world" during test-beam at Desy (S. Borghi, C. Burr, C. Parkes)
- Documentation to be written

- Motivation and Goals
- Simulation
- Conditions support
- Alignment Support
- **Miscellaneous**
- Summary

Miscellaneous

- **Main weak point is documentation**
  - **Need to revisit DDAlign design document**
  - **DDCond and DDAlign user manuals**
- **Need to build a test suite**
  - **Mainly for global alignment procedures**
  - **For local alignment procedures this should come for ‘free’ from Task 3.3**

Toolkit Users

Good news: We start to see contributions from users outside base community (ILC, CLICdp)

- **FCC, SiD**

- **ILC** F. Gaede et al.
- **CLICdp** A. Sailer et al.
- **SiD** W. Armstrong
- **FCC-eh** P. Kostka et al.
- **FCC-hh** A. Salzburger et al.
- **CALICE** Calorimeter R&D, 280 persons: Started
- **FCC-ee** Some interest

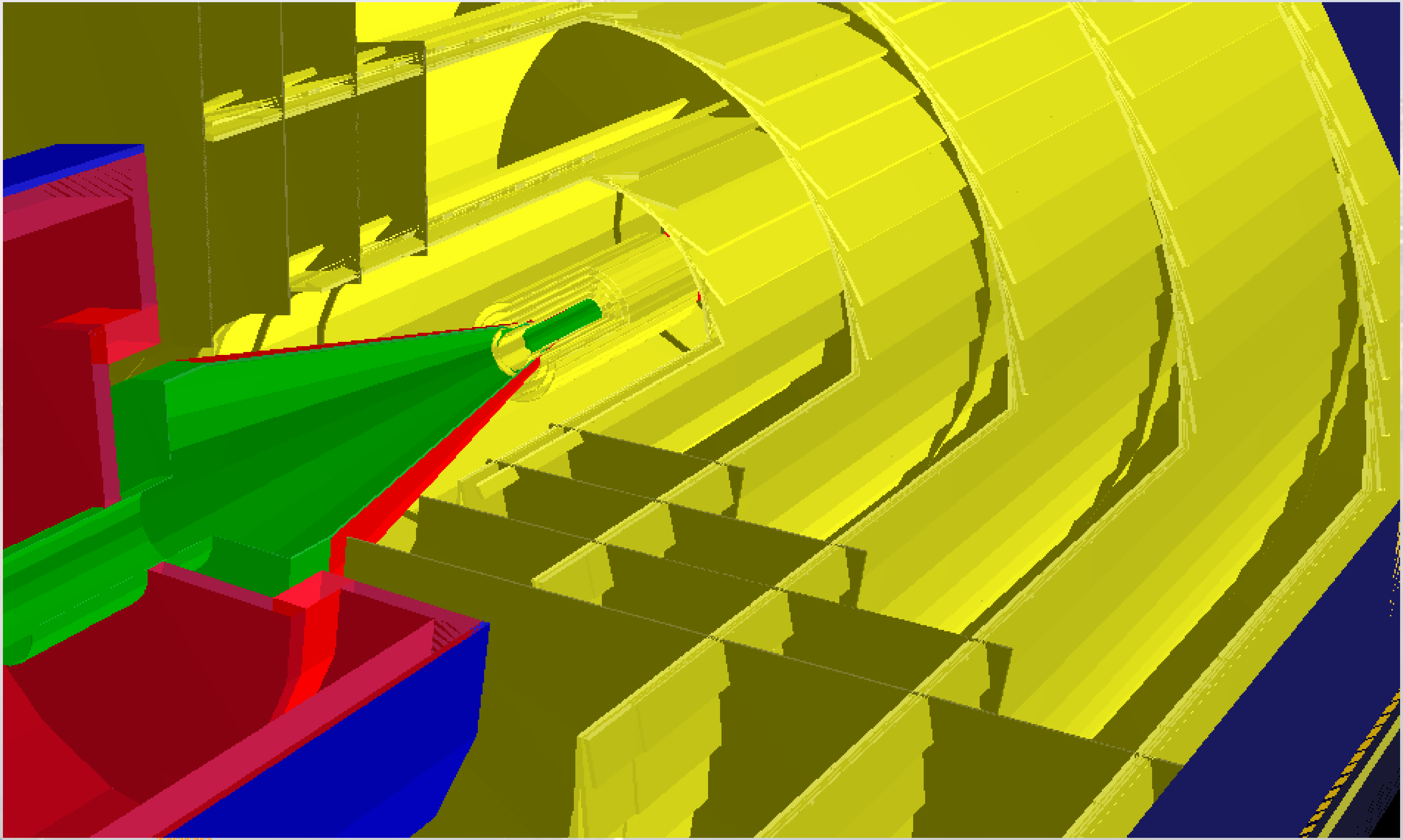
DD4hep	DDG4
X	X
X	X
X	?
X	X
X	

- Motivation and Goals
- Simulation
- Conditions support
- Alignment Support
- Miscellaneous
- **Summary**

Summary

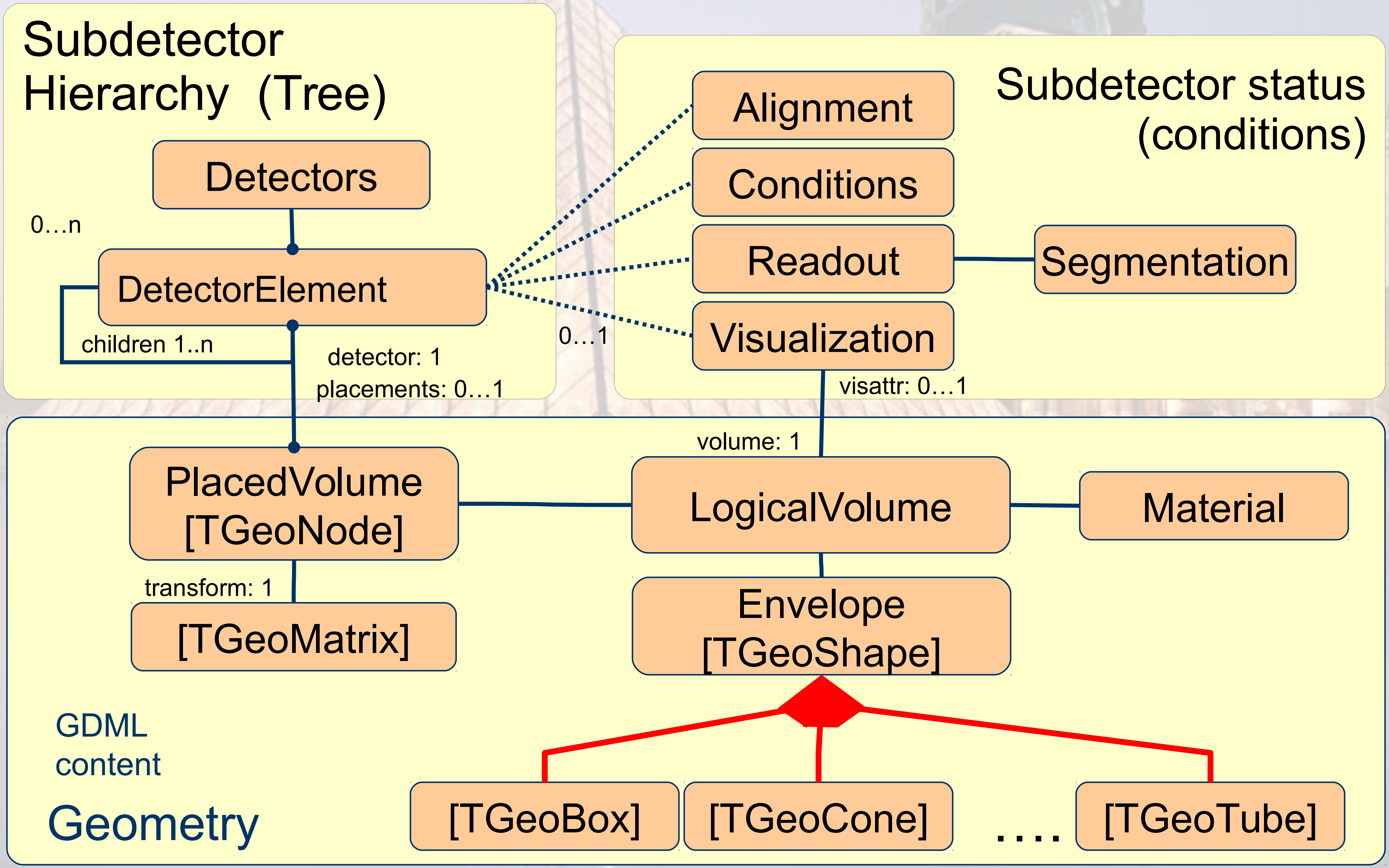
- **The DD4hep core and DDG4 simulation extension were consolidated and are to a large degree on maintenance level**
  - **Deployed by various customers**
- **Support for conditions handling is implemented**
- **Support for alignment handling is being used by collaborators from WP3**
- **Documentation for DDCond and DDAlign is weak and must be improved**
- **We are approaching the “polishing phase”**

Questions and Answers



Backup

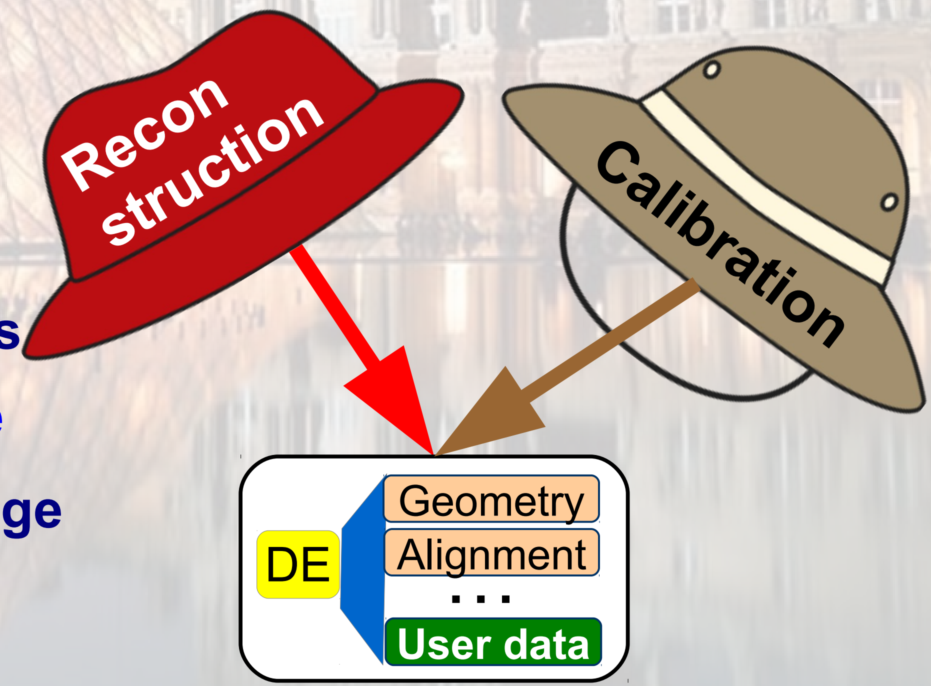
Implementation: Geometry



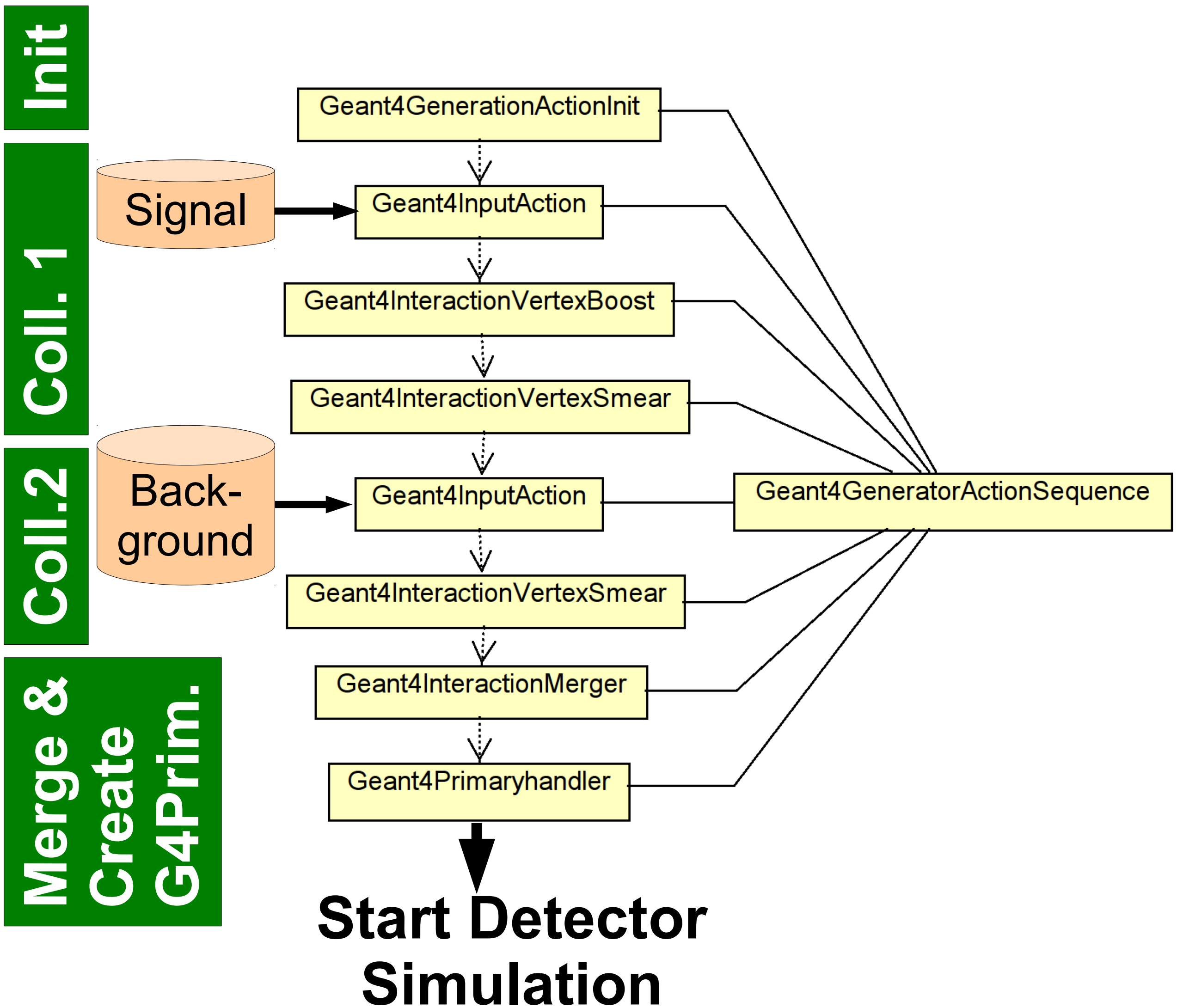
Views & Extensions: Users Customize Functionality

DD4hep is based on handles to data

- Clients only use the handles
- Possibility of many views based on the same DE data
  - Associate different behavior to the same data
  - Views consistent by construction
  - User data according to needs
- Be prudent: blessing or curse
  - User data: common knowledge
  - No one fits it all solution
  - Freedom is also to not do everything what somehow looks possible

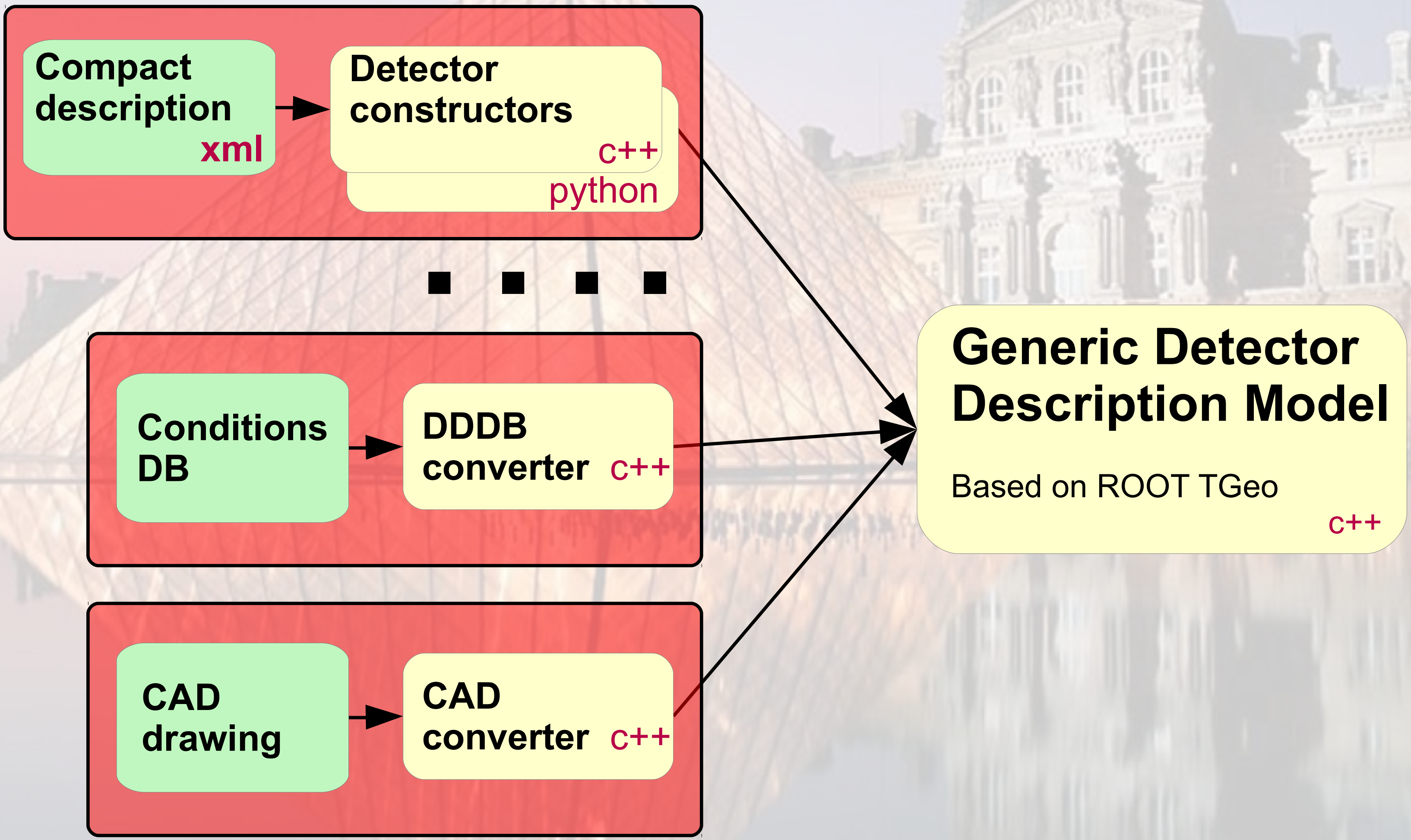


Example of a DDG4 Action Sequence: Event Overlay with Features



- Combine simple and reusable modules
  - Input module
  - Any data format
- Primary vertex smearing
- Primary vertex boost
- Common: initialization, final merge

Multiple Input Sources



LHCb Detector