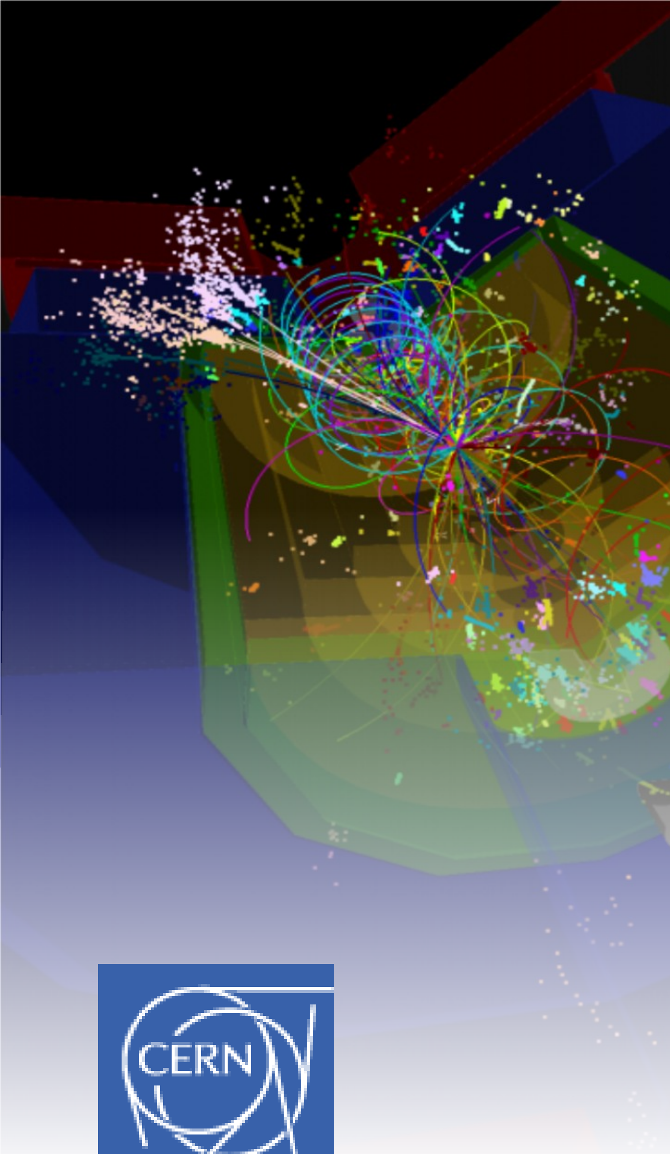


DD4hep

Detector Description Toolkit

DD4hep work status, components and usage

June 25th., 2015

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, single source of information, which supports**
    - simulation, reconstruction, analysis
  - **Full description, including**
    - Geometry, readout, alignment, calibration etc.
- **Driven by lazyness of users**
  - **Get most out of it with minimal efforts**

(*) DD4hep is a sub-package of AIDA2020 WP3

Foreword: About DD4hep & Co

- It is an effort of very few people
  with a simple and comprehensive vision:

Detector description for the lazy ones ... get it all with minimal effort and no technical restrictions

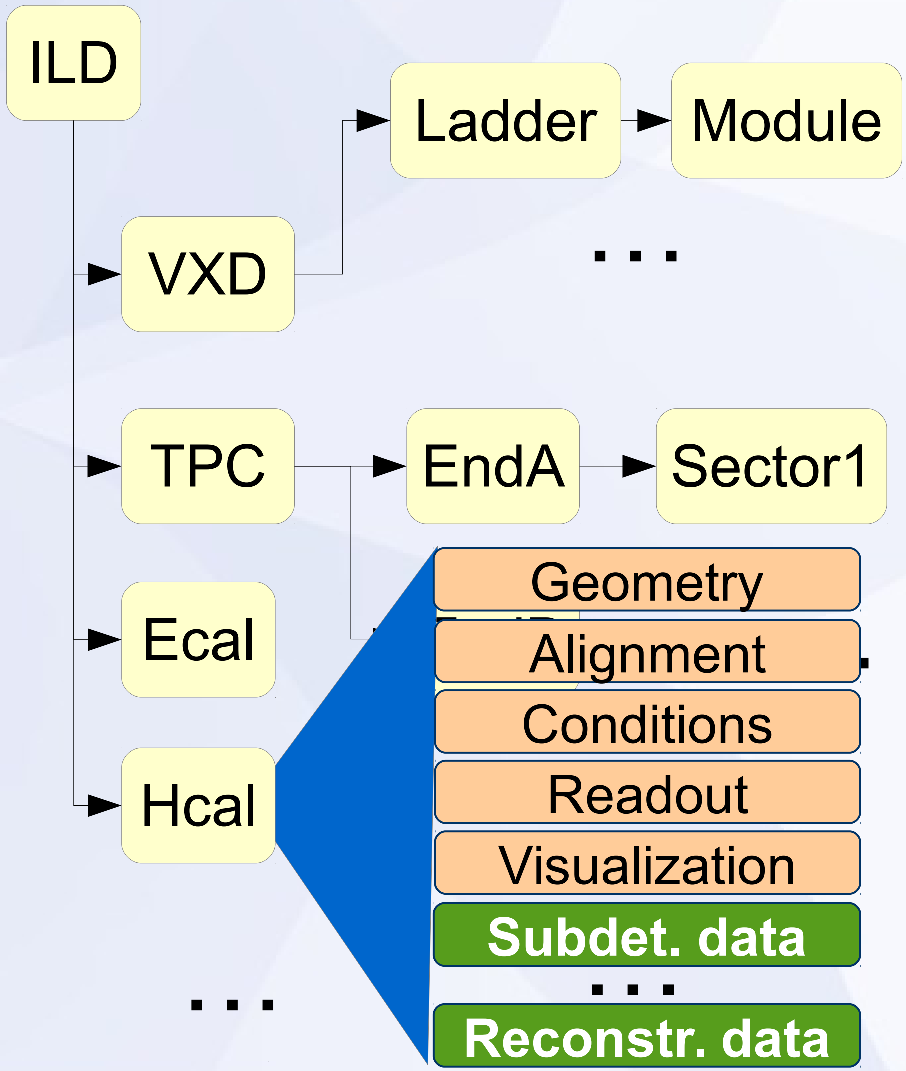
- We welcome new collaborators / users
  and provide support
  - Suggestions are welcome but not under pressure
  - Contributions are even more welcome
  - Users must act responsible … in design
    and when in trouble:
    => Feed back proper analysis to fix problem

    => “It doesn't like me and answers SEGV”

blessing and curse

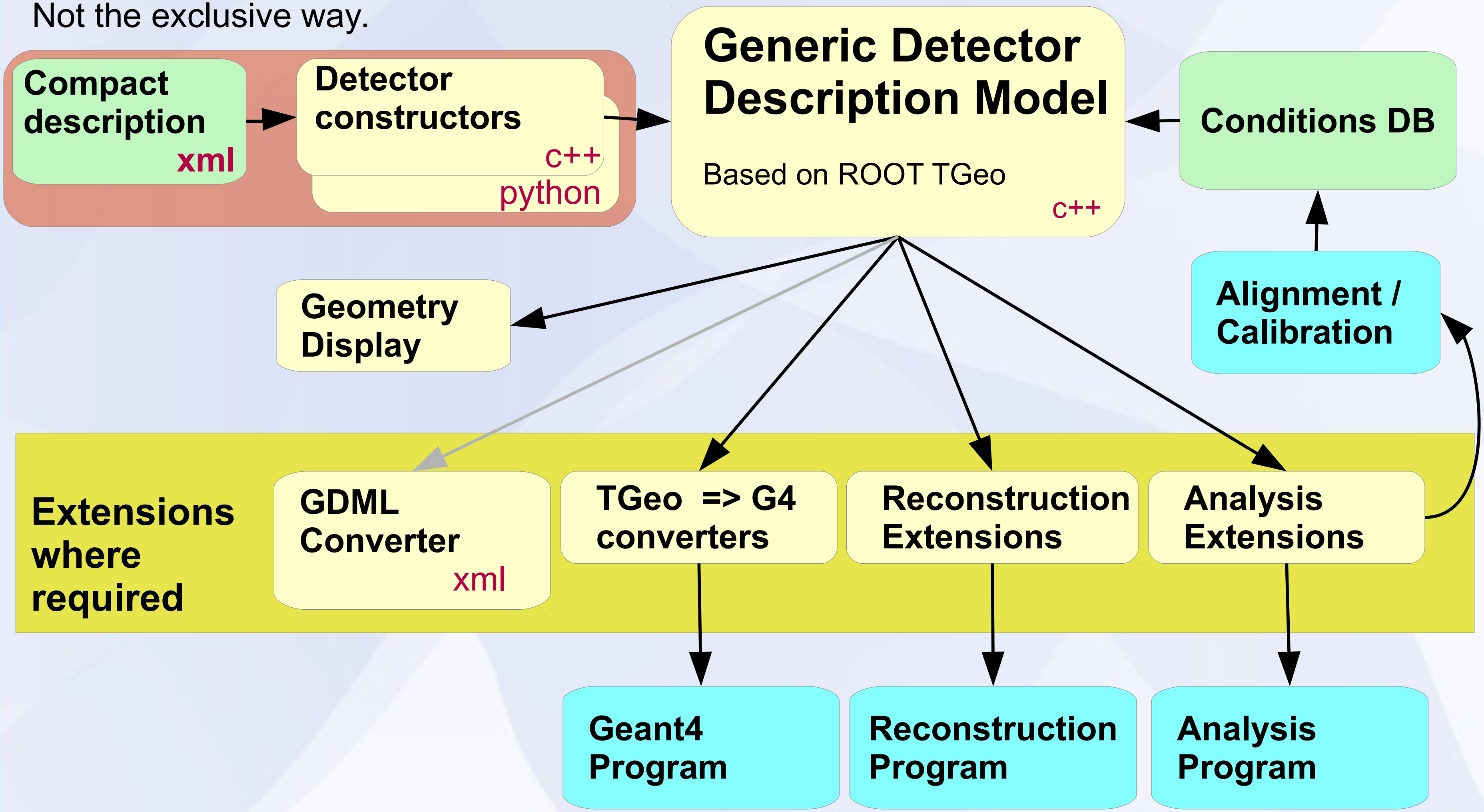
What is Detector Description ?

- **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:
One way to populate DD4hep (plugin based)
Not the exclusive way.

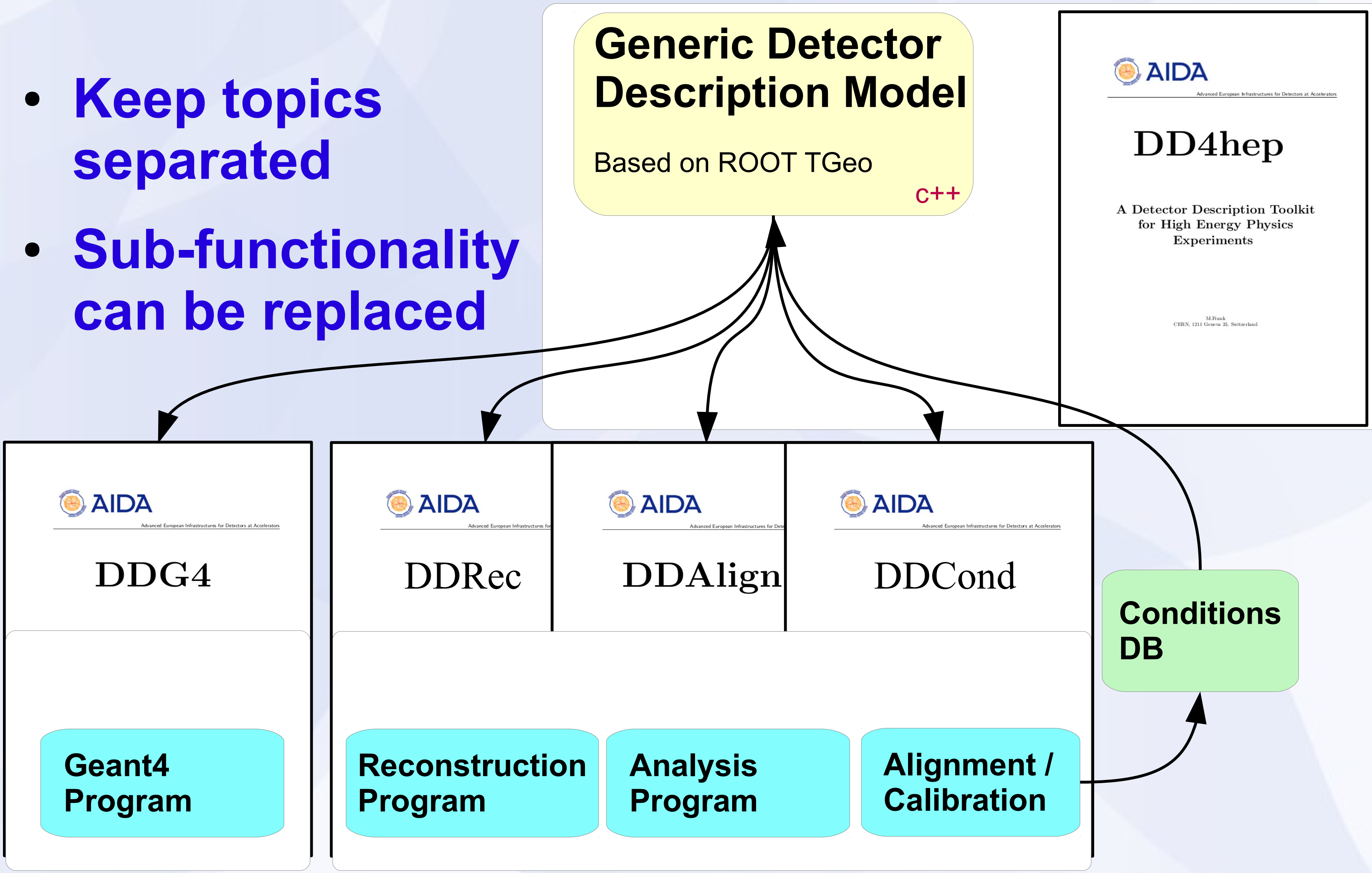


Saga in 5 Episodes: Sub-packages

- **DD4hep – basics/core**
- **DDG4 – Simulation using Geant4**
- **DDRec – Reconstruction supp.**
  - Driven by LC community
- **DDAlign – Alignment support**
- **DDCond – Detector conditions**

Functional Separation: Ensure Flexibility

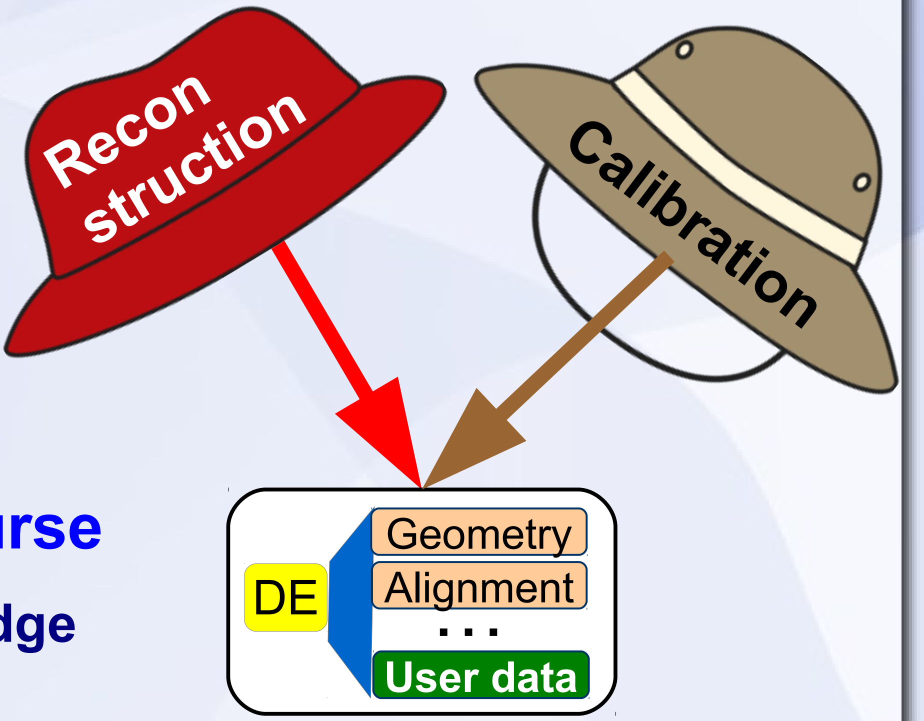
- **Keep topics separated**
- **Sub-functionality can be replaced**



Views & Extensions: Users Customize Functionality

DD4hep is based on handles (smart pointers)

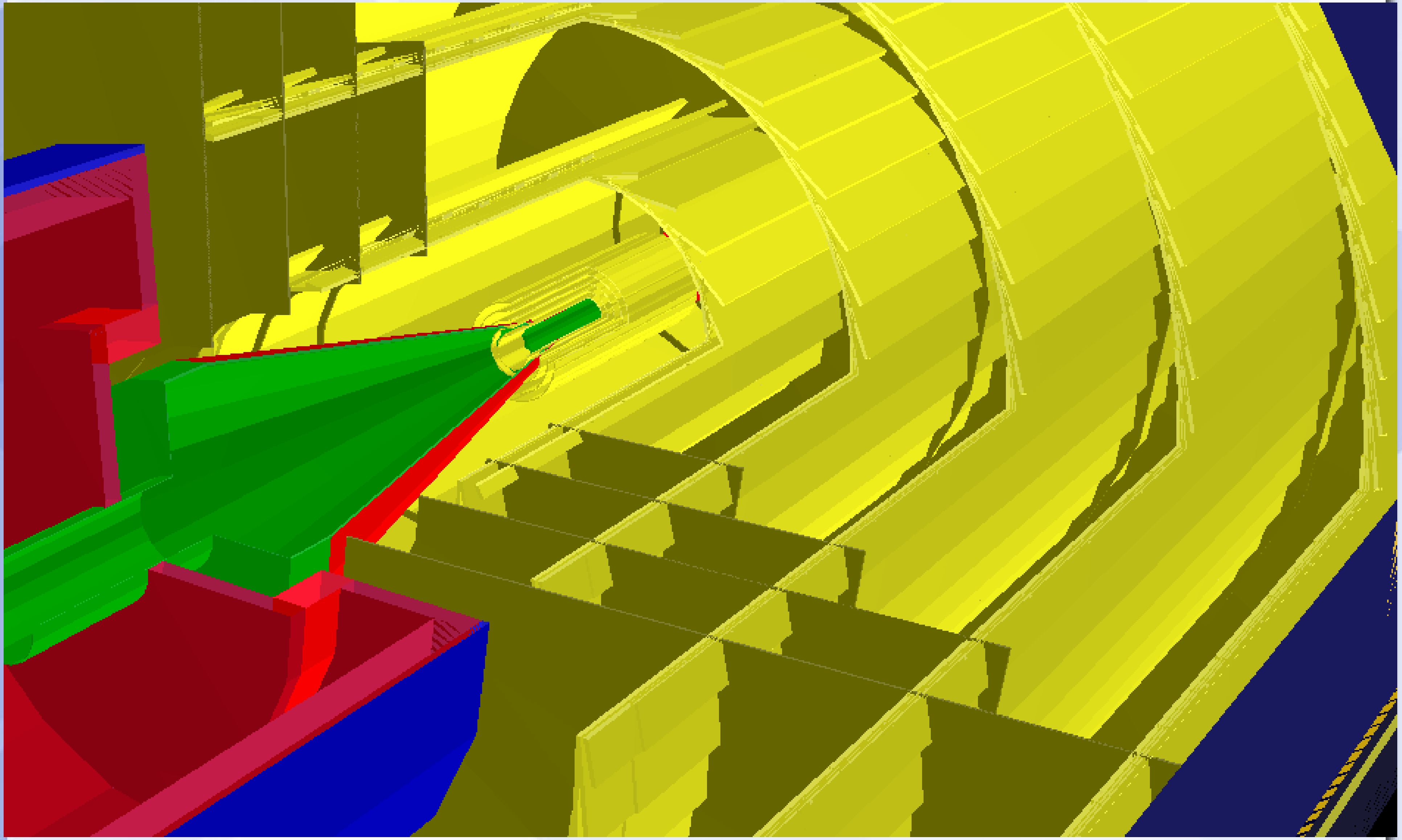
- Rarely deal with data directly
- Possibility of many views based on the same DE data
  - Same ‘data’ associated to different ‘behaviors’
  - All views are consistent and creation is efficient: pointer-copy
  - Add data according to needs
- Be prudent: blessing or curse
  - User data: common knowledge
  - No one fits it all solution



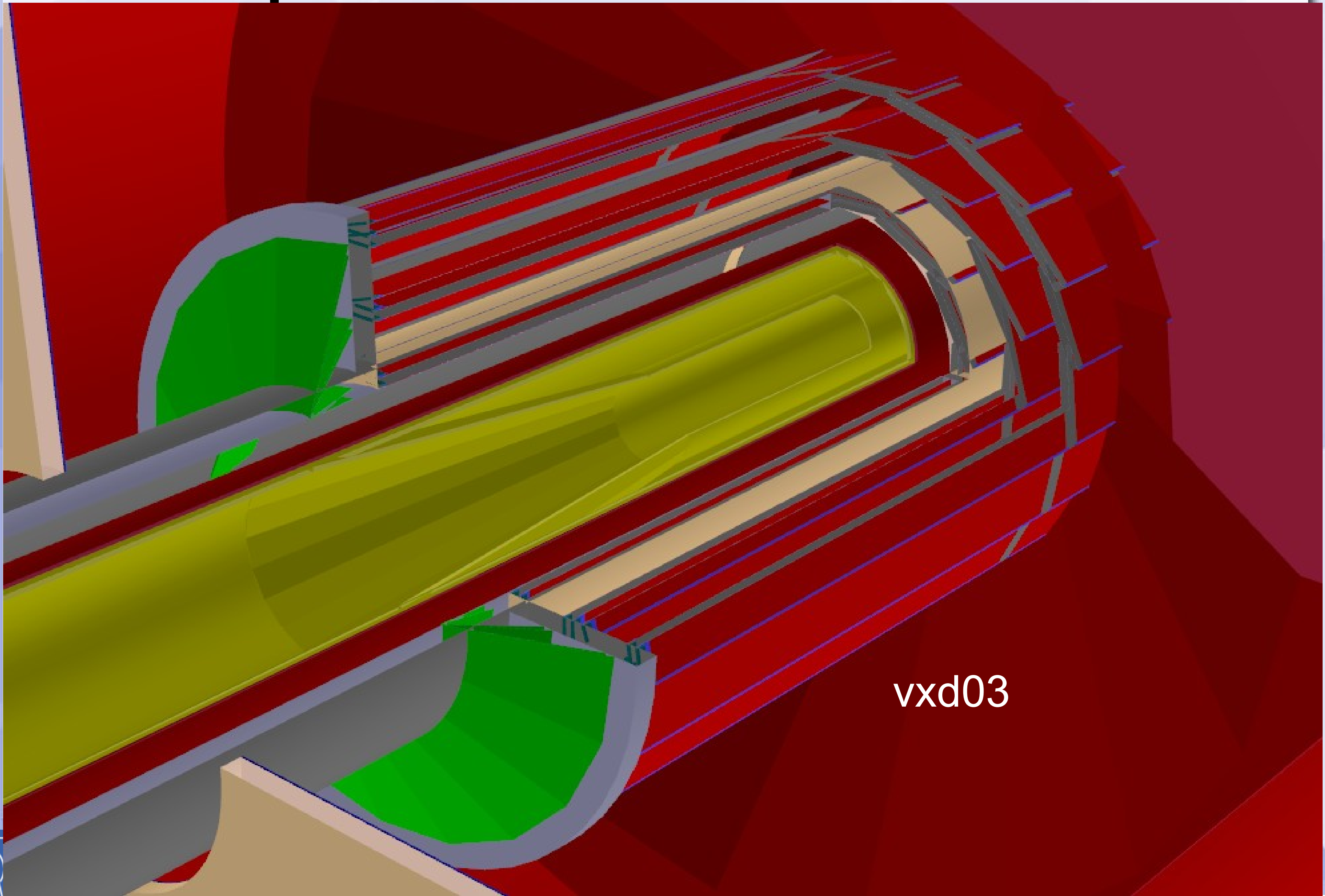
DD4hep Core

- Handles all functionality of detector elements
- Basically stable
  - Bug fixes, enhancements
- Objects are fully reflective
  - C++ dictionary defined
  - Intrinsic support for cross-language development
- Reflection supports interactivity
  - CINT command prompt
  - Python using 'cppyy'

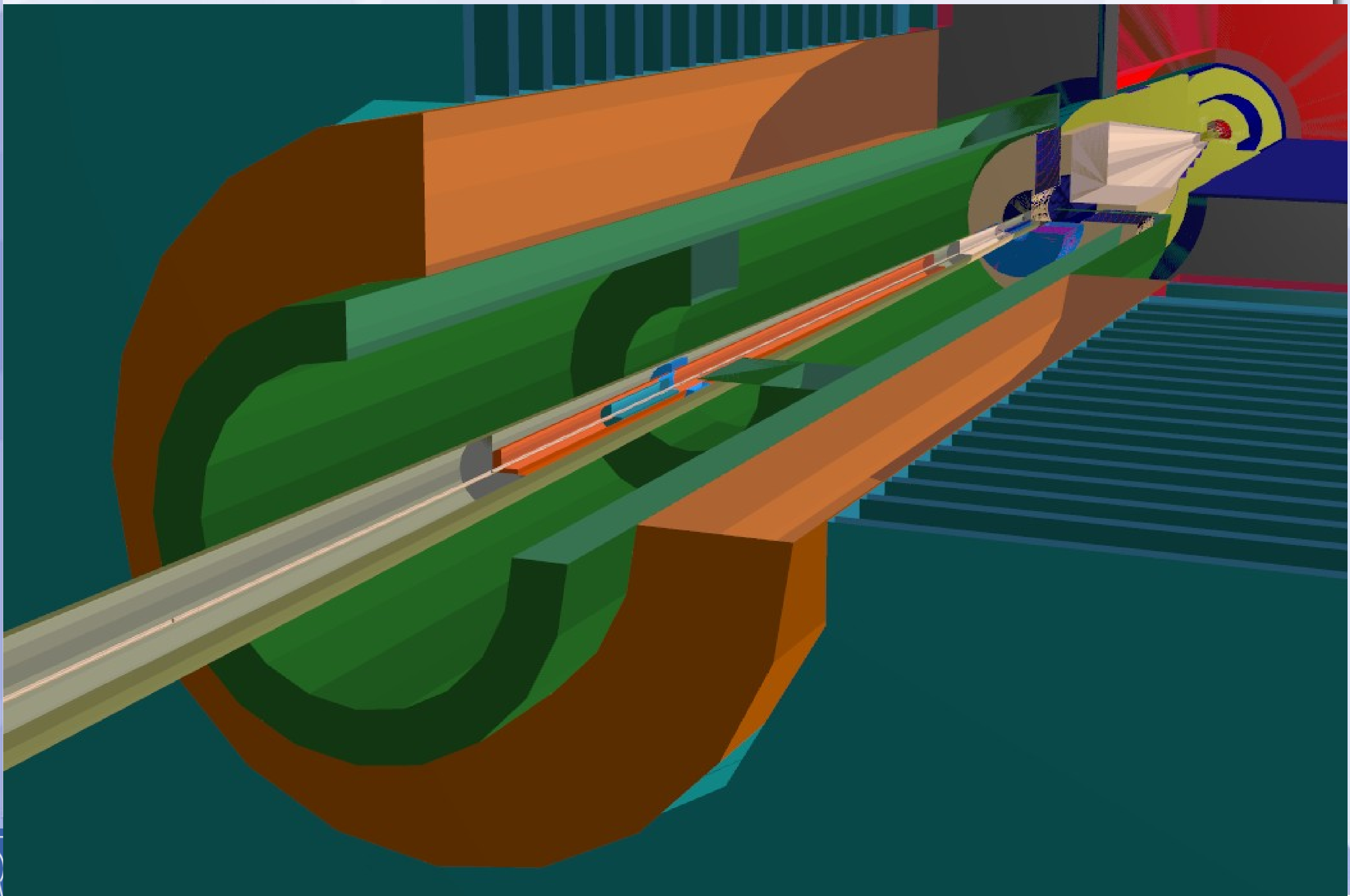
DD4hep Core: Screenshot ILC/SiD



DD4hep Core: Screenshot ILC/Tesla



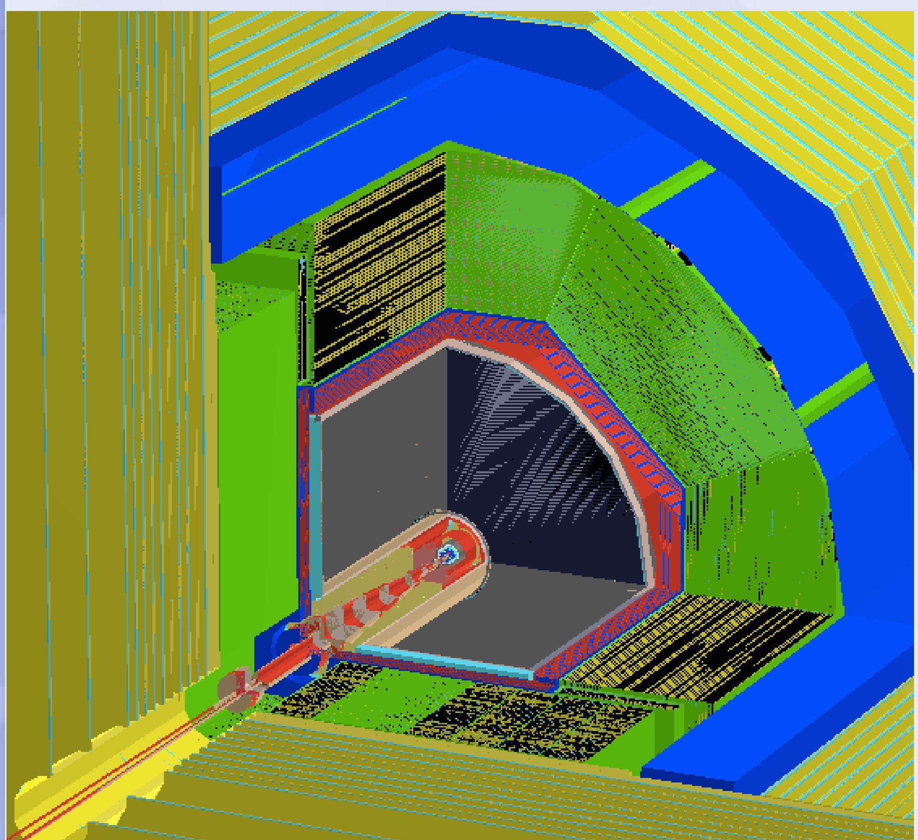
DD4hep Core: Screenshot ILC/Tesla

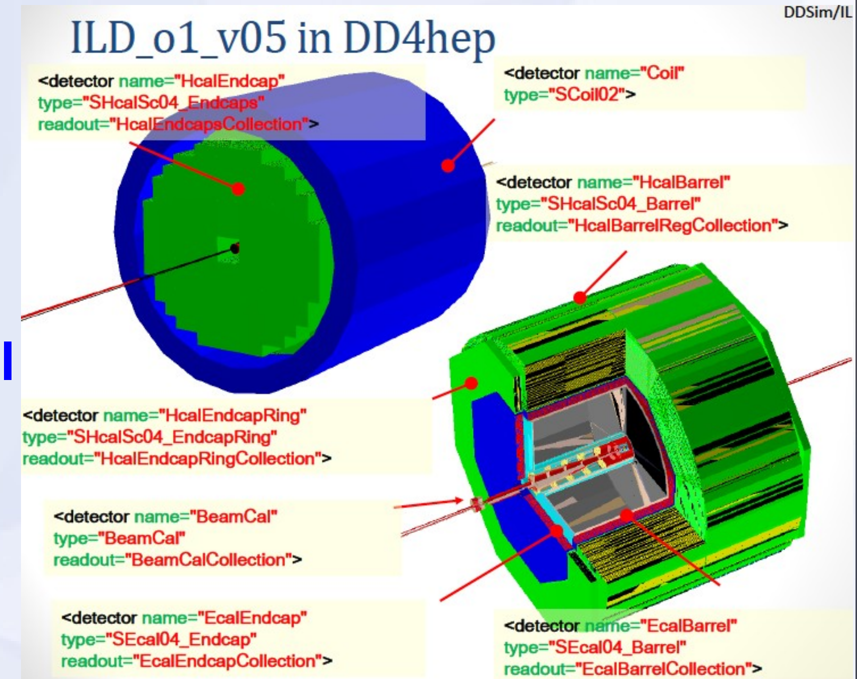


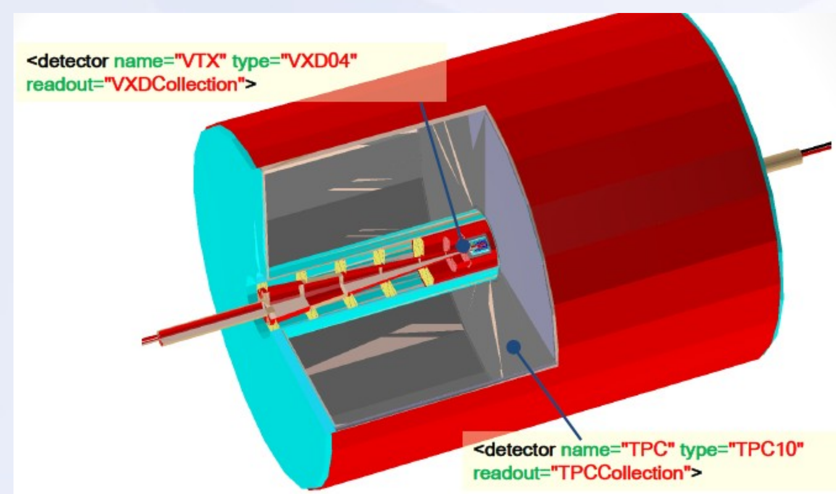
ILD: Model ILD_o1_v05

(F.Gaede, L.Shaojun)

- **VXD, FTD, SIT, TPC, SET, beam pipe**
- **Ecal, Hcal, Yoke, Beamcal, Lcal, LHcal**







Simulation: DDG4

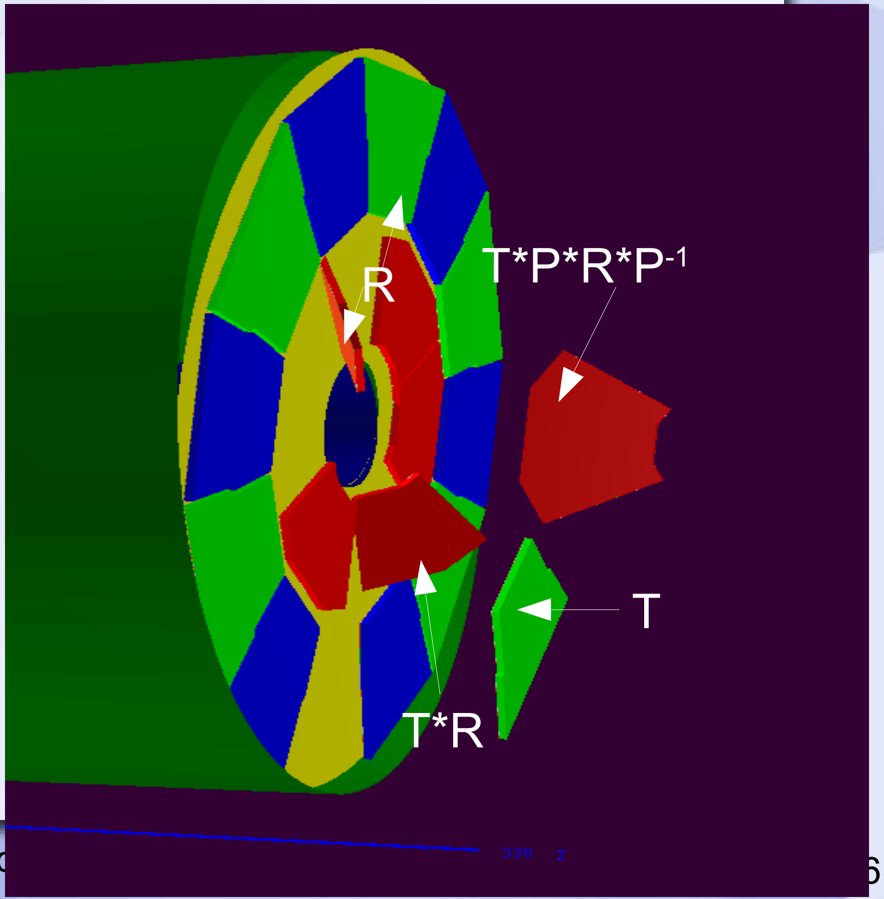
- Simulation = Geometry + Detector response + Physics
- Concept: Formalization of Geant4
  - Automatic conversion from ROOT to Geant4
  - Instantiate objects palette:
    Physics list, -constructors, sens. detectors
  - Start simulating
- Basic sensitive detectors implemented and in use
- Status: implemented and under validation
- No extra (C++) user code necessary
  - But not inhibited e.g. sophisticated sensitive detectors
- Flexible configuration with XML, python or Cint

DDG4: Upcoming Developments

- Support for fast and parametrized simulation
  - Speed-up by avoiding full Geant4 machinery
  - Workshop @ CERN this autumn
- Multi-threading support
  - According to Geant4 rules
  - Multiple instances of elements handling actions during energy deposits while tracking
- Revisit integration into experiment frameworks
  - See also talk from B.Hegner
- Move to ROOT 6

DDAlign: Detector Alignment

- Fundamental functionality to interpret event data in the real world
  - Selling argument for existing experiments
  - Must handle imperfections
    - Geometry => (Mis)Alignment
  - Anomalous conditions
    - Pressures, temperatures
      => Gains, refractive indices
      => Contractions, expansions
  - Basic functionality present
    - No connection to persistency



DDAlign: Detector Alignment

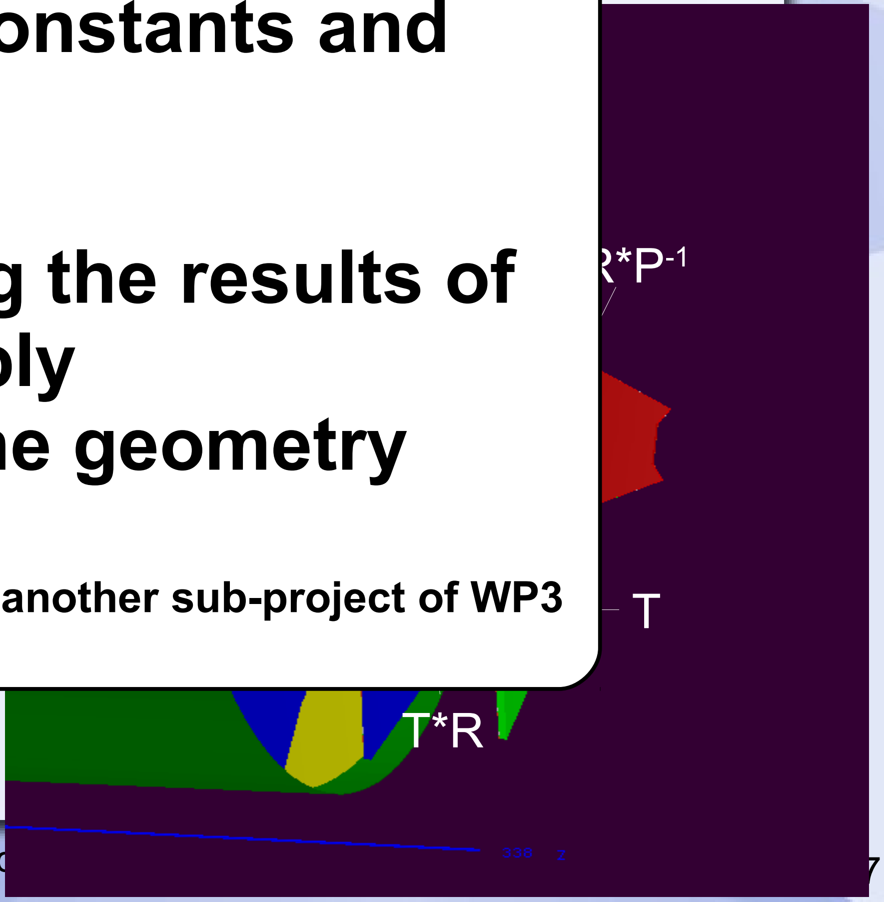
- Fu … in
  th…

<u>Please Note:</u>

DDAlign does not provide *algorithms* to determine alignment constants and never will (*)

DDAlign supports hosting the results of the algorithms and to apply alignment constants to the geometry

(*) Alignment procedures investigated by another sub-project of WP3

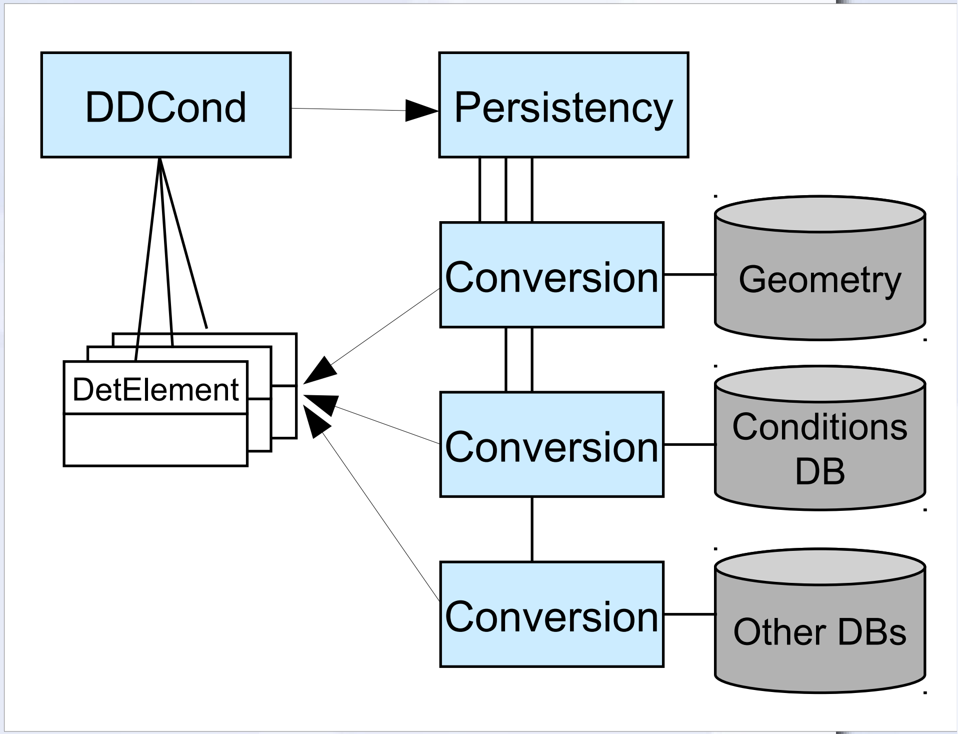


DDCond: Conditions Data
Tales of thin air …

- 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”
  - May be time interval, run number, named period, ...
  - Configurable and extensible
- Data typically stored in a database

DDCond: Workplan
The only thing that exists ...

- **The transient implementation**
  - **Flexible definition and handling of intervals of validity ==> Key point**
- **Persistent implementation**
  - **Define interface/ABC**
  - **Proof of concept using one XML, SQLite, Oracle, ...**



Toolkit Users

Users are mandatory for feedback to avoid developments in thin air (i.e. purely academic)

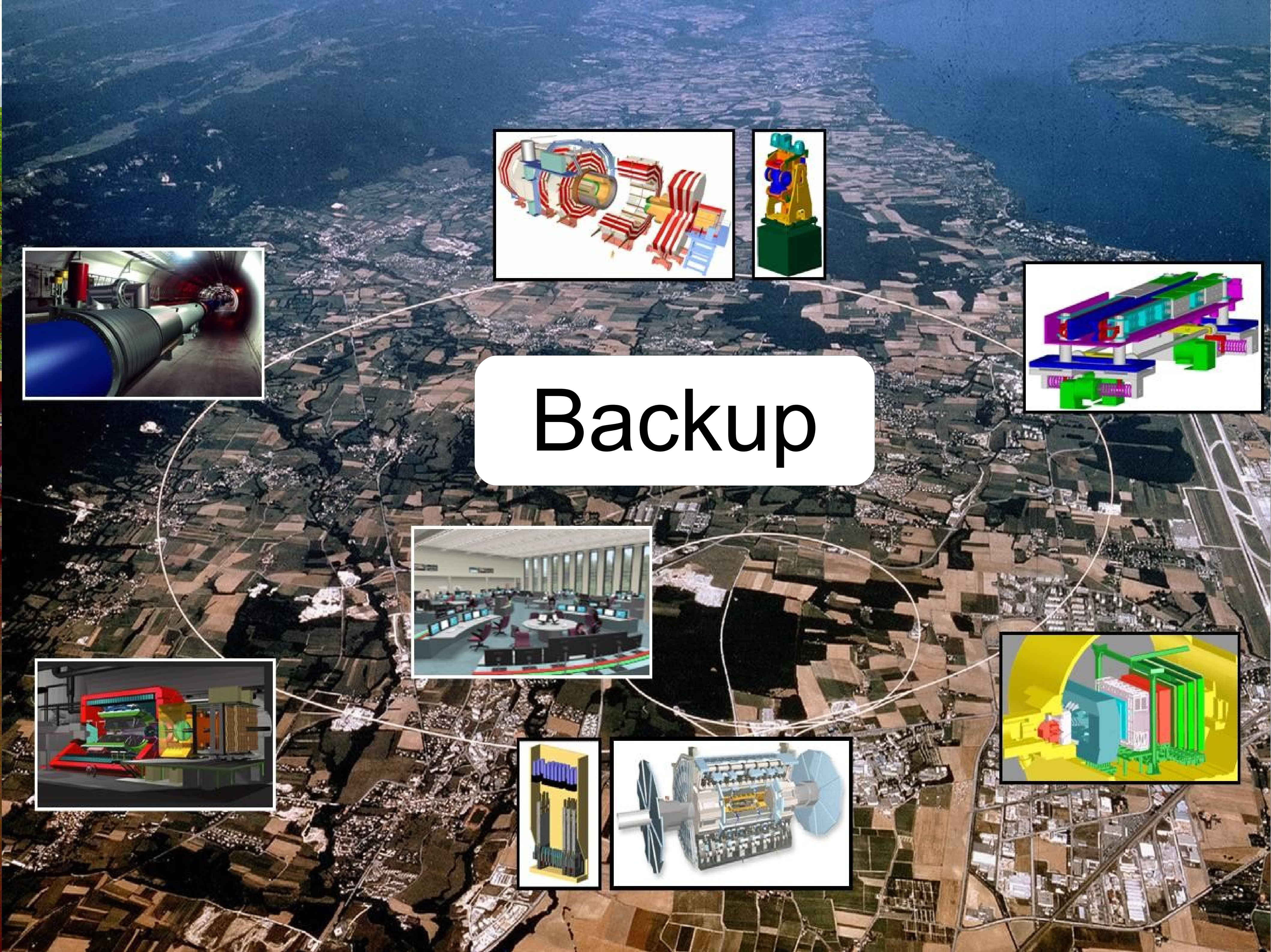
- **ILD:** F. Gaede et al., ported complete Mokka model ILD_o1_v05
- **CLICdp:** starting new design after CDR
- **FCC-eh:** P. Kostka et al.
- **FCC-hh:** A.Salzburger et al.

DD4hep	DDG4
X	X
X	x
X	X
X	

Summary and Outlook

- The DD4hep toolkit (+extensions) start to become accepted: Client validation has started
- Simulation kit DDG4 being validated
- Alignment support to be completed
  - Requires conditions support for full functionality
    => DDCond: extension to be developed
- Validate, verify, enhance and document
- Happy to welcome new users and their contributions

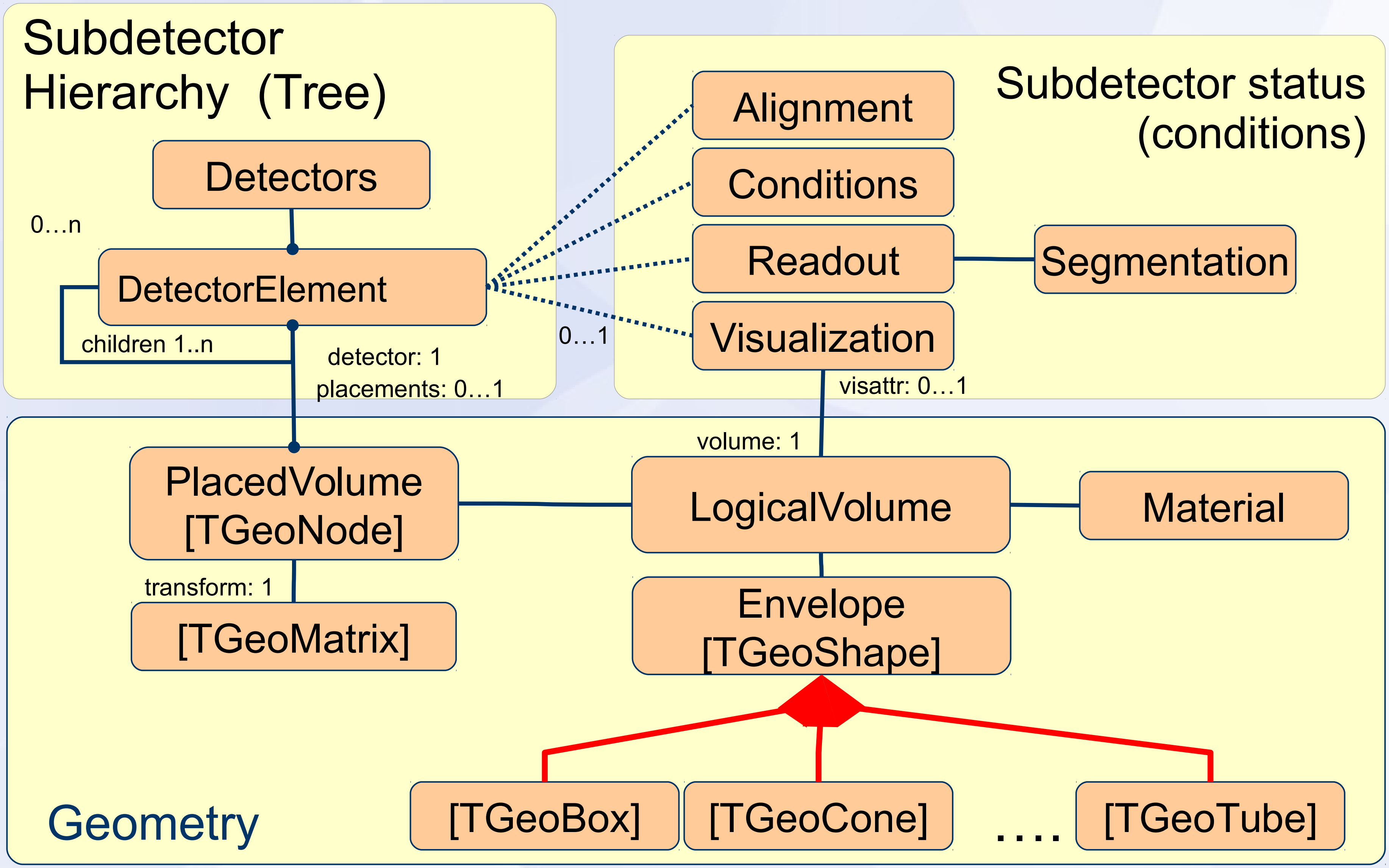
Backup



Design Principles

- **Separation of data and behavior**
  - **Data are fully accessible (no encapsulation!)**
  - **Behavioral classes are wrappers around objects containing data only**
  - **There may be many behavioral wrapper implementations using the same data objects**
    - **User chooses “most suitable” behavior**
  - **One “data-object” may be shared among many behavioral wrapper instances**

Class Diagram: Detector Element



Standard Detector Palette: DDDetectors

- Mostly arose from the SiD model
  - Layer based detectors
  - Tracker barrel & endcap
  - Several calorimeter constructs
- Partially with measurement surfaces

  (see also talk by F. Gaede)
- Plugin mechanism to enhance detector elements
  - Neat mechanism to attach user defined optional data

    => Proof that 'anticipate the unforeseen' works
  - NOT intrusive to detector constructors
  - Flexible definition of the measurement surface

Geant4 Interactivity

```
Idle> ls /ddg4
Command directory path : /ddg4/


Guidance :
Control for all named Geant4 actions

 Sub-directories :
   /ddg4/RunInit/   Control hierarchy for Geant4 action:RunInit
   /ddg4/RunAction/   Control hierarchy for Geant4 action:RunAction
   /ddg4/EventAction/   Control hierarchy for Geant4 action:EventAction
   /ddg4/LcioOutput/   Control hierarchy for Geant4 action:LcioOutput
```

Geant4 interactivity interfaced to every action object

- **Enabled on request**

```
 Sub-directories :
 Commands :
   show * Show all properties of Geant4 component:UserParticleHandler
   Control * Property item of type bool
   MinimalKineticEnergy * Property item of type double
   Name * Property item of type std::string
   OutputLevel * Property item of type int
   TrackingVolume_Rmax * Property item of type double
   TrackingVolume_Zmax * Property item of type double
   name * Property item of type std::string
Idle> /ddg4/UserParticleHandler/TrackingVolume_Rmax
Geant4UIMessenger: +++ UserParticleHandler> Unchanged property value TrackingVolume_Rmax = 1265.
Idle> /ddg4/UserParticleHandler/TrackingVolume_Rmax 1.3*m
Geant4UIMessenger: +++ UserParticleHandler> Setting property value TrackingVolume_Rmax = 1.3*m  native:1300.
Idle> /ddg4/UserParticleHandler/TrackingVolume_Rmax
Geant4UIMessenger: +++ UserParticleHandler> Unchanged property value TrackingVolume_Rmax = 1300.
Idle>
```

Actions have properties
(similar to Gaudi)

- **Interrogate properties**
- **Modify properies**

Configure DDG4 Application with python

```
kernel = DDG4.Kernel()
lcdd = kernel.lcdd()
kernel.loadGeometry("file:"+install_dir+"/DDDet
kernel.loadXML("file:"+example_dir+"/DDG4_field
DDG4.importConstants(lcdd)
```

```
Generation of isotrope tracks of a given multip
"""
# First particle generator: pi+
gen = DDG4.GeneratorAction(kernel,
          "Geant4IsotropeGenerator/IsotropPi+")
gen.Particle = 'pi+'
gen.Energy = 100 * GeV
gen.Multiplicity = 2
gen.Mask = 1
kernel.generatorAction().adopt(gen)
# Install vertex smearing for this interaction
gen = DDG4.GeneratorAction(kernel,
          "Geant4InteractionVertexSmear/SmearPi
gen.Mask = 1
gen.Offset = (20*mm, 10*mm, 10*mm, 0*ns)
gen.Sigma = (4*mm, 1*mm, 1*mm, 0*ns)
kernel.generatorAction().adopt(gen)
```

- **Python configuration snippets**
  - **Loading geometry**
  - **Configuring actions**
  - **Steer Geant4 until it's prompt/batch**
- **C++ config ~ same**
- **Alternative: xml**
  Load xml with lcdd

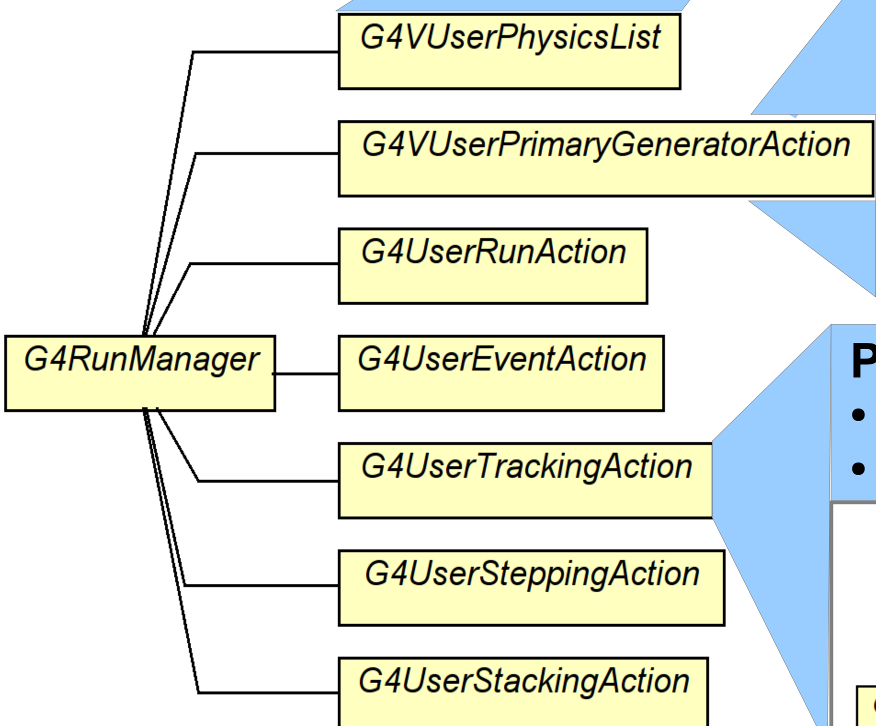
Geant4 Provided Hooks

[and what we want to do inside]

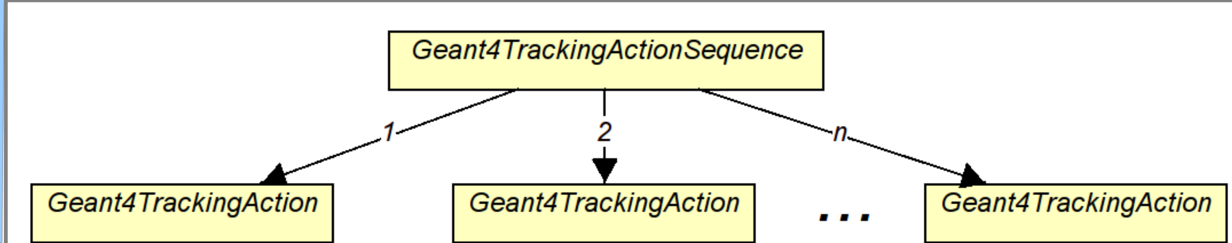
Main issue: flexible configuration

Flexible definition of the physics list
- Define particles, processes, physics constructors or use/extend predefined physics lists

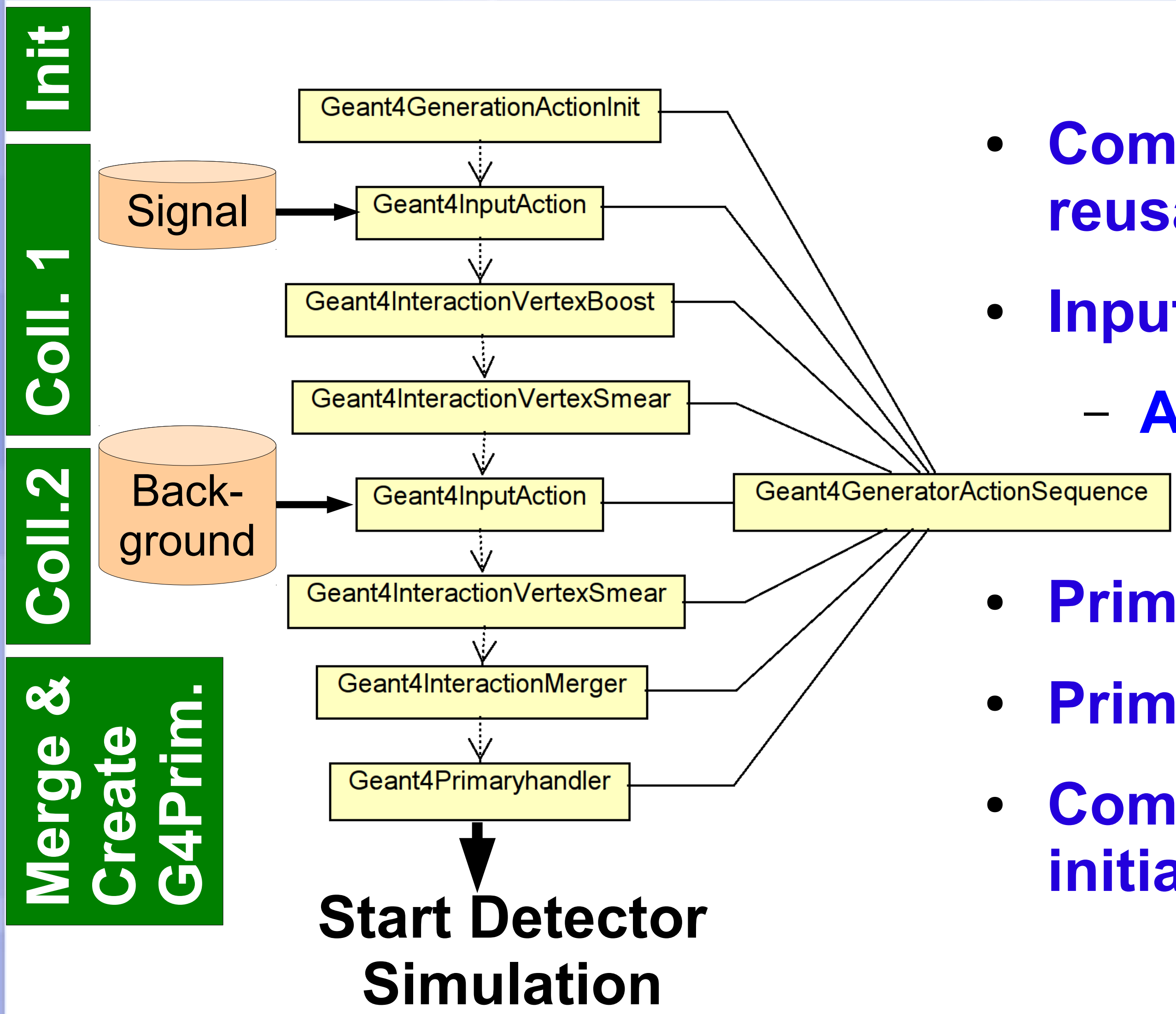
Flexible data input
- Programmable sequence. Input from particle gun, lcio, stdhep or HepMC (text) – easily extensible
- Modules to smear and boost primary vertices
- Modules to construct interaction overlays
- Further extensions may independently added



Provide user programmable sequences
- Either as explicit object type using ABC
- Or registering a member function as callback



Example of an 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