New Simulation Tools

N. Nikiforou, CERN/PH-LCD
On behalf of the CLICdp Collaboration and the Linear Collider DD4hep WG

Introduction

- **"(Full) Simulation Tool":** allows creation of **geometry** in **Geant4** and provides access to its **kernel** to **control the simulation** of the interaction of particles with matter
  - **Input** (generator particles/events) or control particle gun
  - **Output** of "hits", i.e. response of sensitive detectors in a convenient format (e.g. LCIO file)
  - Handle properly the **MC Particle History** and store a meaningful **Truth Record**
  - It would be good to have some sort of **visualization** independently of Geant4 (heavy)
    - Even better, if you could check the geometry without having to build against Geant4

Existing simulation tools for LC

- All included in recent iLCSoft releases
- `Mokka`: Main workhorse for ILD, used also by CLICdp
  - C++ drivers (part of package)
  - Parameters stored in **a MySQL database**
  - No longer maintained
- `SLIC`: Simulation tool for SiD, used also by CLICdp
  - Loads geometry from single **LCDD** (⊃ GDML) **xml file**
    - Itself created from `GeomConverter` or `DD4hep`
  - Developed and maintained at SLAC
- *`DDSim`*: New kid on the block
  - **DD4hep** application using its **DDG4** library
  - M. Frank, F. Gaede, A. Sailer and others from ILD/CLICdp
  - Focus of this talk

DD4hep motivation and goals

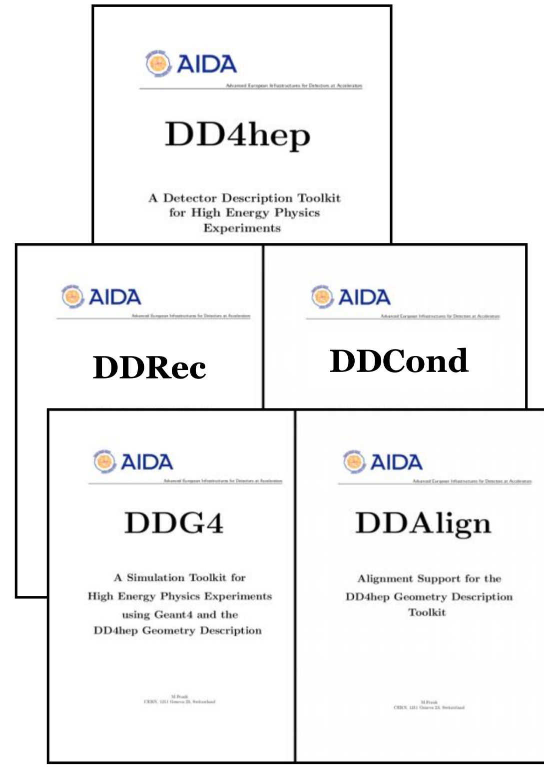
- Complete detector description
  - Includes geometry, materials, visualization, readout, alignment, calibration, etc.
- Support full experiment life cycle
  - Detector concept development, detector optimization, construction, operation
  - Easy transition from one phase to the next
- Consistent description, single source of information
  - Use in simulation, reconstruction, analysis, etc.
- Ease of use
- Few places to enter information
- Minimal dependencies

DD4hep was built on experience

- Started as a collaborative effort between colleagues from CLICdp, ILD and SiD
- Extended the philosophy in **`Mokka/SLIC`** where driver construction is controlled by a set of parameters
  - Wanted to have proper driver scalability and flexibility
- Adopted the compact xml format and philosophy developed for the **`GeomConverter/SLIC`** chain
  - Can have a "compact", natural description of the detector layout and overall sizes
  - Decentralized "database" in the form of the xml
- Can build **`DD4hep`** without **`Geant4`** if **`DDG4`** is not needed

DD4hep Components

- **DD4hep**: basics/core
  - Basically stable
- **DDG4**: Simulation using Geant4
  - Validation ongoing
- **DDRec**: Reconstruction support
  - Driven by LC Community
  - See more on next talks but also talks in Simulation/Detector Performance/Reconstruction sessions
- **DDAlign, DDCond** : Alignment and Conditions support
  - Being developed



- **http://aidasoft.web.cern.ch/DD4hep**

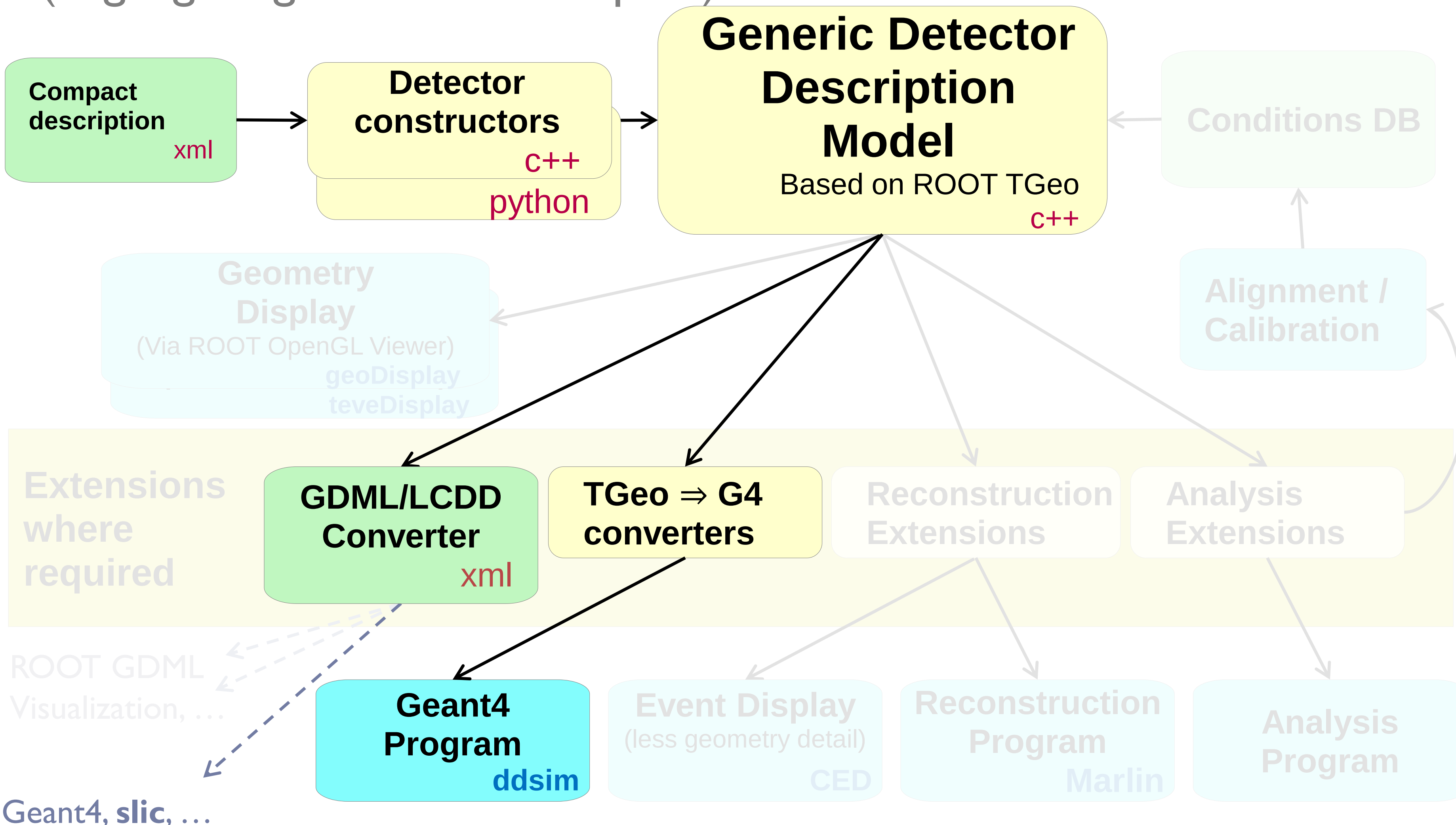
Current DD4hep toolkit users

		DD4hep	DDG4
ILD	F. Gaede et al., ported complete model ILD_o1_v05 from previous simulation framework (Mokka)	✓	✓
CLICdp	New detector model being implemented after CDR, geometry under optimization	✓	✓
FCAL	Testbeam simulation	✓	✓
FCC-eh	P. Kostka et al.	✓	✓
FCC-hh	A. Salzburger et al.	✓	

Feedback from users is invaluable and helps shaping DD4hep!

DD4hep – The big picture

(Highlighting the simulation path)



M. Frank

The TGeo advantage

- Visualize and check the geometry in detail outside Geant4 first with ROOT's OpenGL viewers
  - Easier manipulation of the scene (rotate, pan, clip, …)
  - Tools (overlap check, independent GDML dump, …)

- Can implement Event Displays using TEve
- Implement toggling of display of subdetectors on the fly, chose to show just envelopes, just surfaces, …

- Nice treatment of assemblies (especially assemblies-in-assemblies)
  - Avoid having to describe complex shapes to hold modules like the spiral vertex detector endcaps

DDG4: Gateway to Geant4

- DD4hep facilitates **in-memory translation of geometry** from TGeo to Geant4
- Plugin Mechanism:
  - Sensitive detectors, segmentations and configurable actions, …
- **All shared with Reconstruction!**

- Configuration mechanism (via python, XML, CINT)
  - Physics lists, regions, limits, fields, …

- For example, configure and launch the simulation using python (next slide)

DDG4 configuration

- DDG4 is highly modular
- Easy to configure, especially if one uses the python dictionaries
- Configure actions, filters, sequences, cuts, …

```
...
part = DDG4.GeneratorAction(kernel,
                            "Geant4ParticleHandler/ParticleHandler")
kernel.generatorAction().adopt(part)
part.SaveProcesses = ['Decay']
part.MinimalKineticEnergy = 1*MeV
part.KeepAllParticles = False
...
user = DDG4.GeneratorAction(kernel,
               "Geant4TCUserParticleHandler/UserParticleHandler")
user.TrackingVolume_Zmax = DDG4.tracker_region_zmax
user.TrackingVolume_Rmax = DDG4.tracker_region_rmax
...
```

Where can I find all this?

- DD4hep comes complete with example drivers and compact files in iLCSoft releases
  - Under `DD4hep/<version>/DDDetectors`
  - More examples and use cases under `DD4hepExamples`
- For the Linear Collider Community we have another package: `LCGeo`
  - We collect here the concrete implementations of Detector Models (currently for CLICdp and ILD)
    - All their versions, additional specialized subdetector drivers if needed
  - We also have use case examples, configuration files and tools **including `ddsim`, a tool to run DDG4 simulation**

ddsim executable

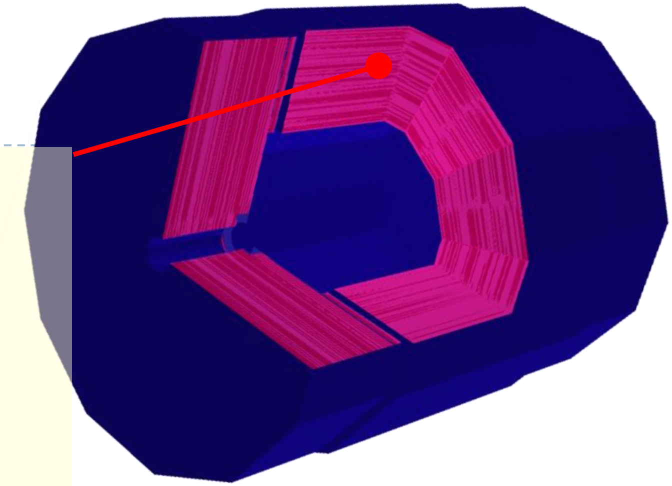
- Python executable with many command-line argument configuration options
  - Configure most useful and common user options in the command line
  - Even supports tab-completion of arguments and their options! (A. Sailer)

```
ddsim –h
usage: Running DD4hep Simulations: [-h] [--steeringFile STEERINGFILE]
[--compactFile COMPACTFILE] [--runType {batch,vis,run,shell}]
[--inputFiles INPUTFILES [INPUTFILES ...]] [--outputFile OUTPUTFILE] [-v PRINTLEVEL]
[--numberOfEvents NUMBEROFEVENTS] [--skipNEvents SKIPNEVENTS]
[--physicsList PHYSICSLIST] [--crossingAngleBoost CROSSINGANGLEBOOST]
[--vertexSigma VERTEXSIGMA VERTEXSIGMA VERTEXSIGMA VERTEXSIGMA]
[--vertexOffset VERTEXOFFSET VERTEXOFFSET VERTEXOFFSET VERTEXOFFSET]
[--macroFile MACROFILE] [--enableGun]
[--enableDetailedShowerMode]
```

Continuously implementing more options!

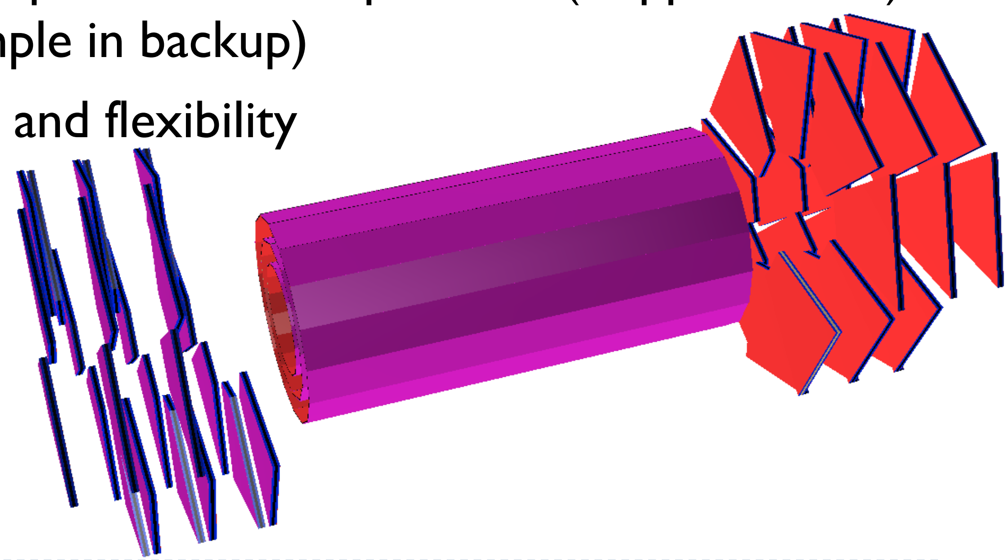
- Calls Python library which is also modular and even more configurable (more advanced)
  - Users can write applications using DDG4

Implementing detectors



```xml
<detector id="DetID_HCAL_Barrel" name="HCalBarrel" type="HCalBarrel_o1_v01"
readout="HCalBarrelHits" vis="HCALVis" >
 <dimensions nsides="HCal_symm" rmin="HCal_Rin" z="HCal_Z" />
 <layer repeat="(int) HCal_layers" vis="HCalLayerVis" >
  <slice material="Steel235" thickness="0.5*mm" vis="AbsVis"/>
  <slice material="Steel235" thickness="19*mm" vis="AbsVis"/>
  <slice material="Polysterene" thickness="3*mm" sensitive="yes"/>
  <slice material="PCB" thickness="0.7*mm"/>
  <slice material="Steel235" thickness="0.5*mm" vis="AbsVis"/>
  <slice material="Air" thickness="2.7*mm"/>
  </layer>
</detector>
```

- Fairly scalable and flexible drivers (Generic driver palette available)
- Visualization, Radii, Layer/module composition in compact xml (snipped above), volume building in C++ driver (example in backup)
- User decides balance between detail and flexibility
- Usually could do a lot just by modifying the xml. For example:
  - Scale detector
  - Create double layers
  - Create "spiral" endcap geometry
  - …



Envelopes

- Good practice: each subdetector should be contained in an **envelope** defining its boundaries

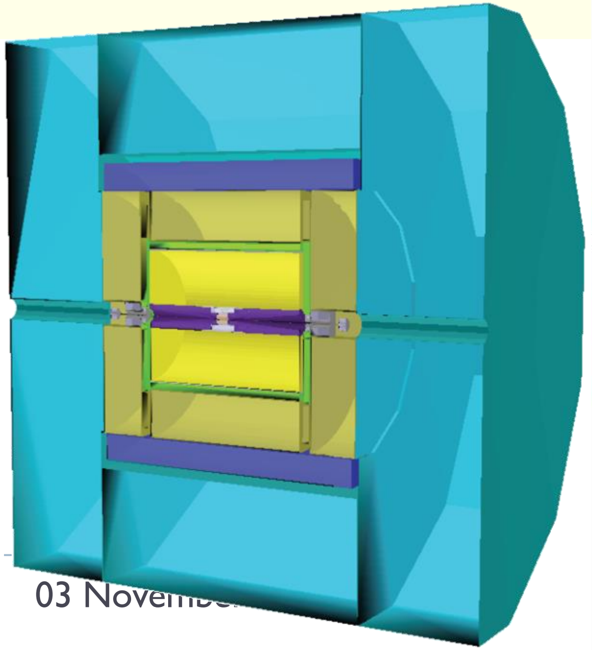
- Fairly complex envelopes can be fully described in the XML
- Using high-level parameters
  - e.g Inner/outer radius

```xml
<envelope vis="ILD_ECALVis">
 <shape type="BooleanShape" operation="Subtraction" material="Air">
  <shape type="BooleanShape" operation="Subtraction" material="Air">
   <shape type="BooleanShape" operation="Intersection" material="Air">
    <shape type="Box" dx="R_out" dy="R_out" dz="Z_max"/>
    <shape type="PolyhedraRegular" numsides="symmetry" rmin="0"
           rmax="R_out" dz="2.0*Z_max"/>
    <rotation x="0*deg" y="0*deg" z="90*deg-180*deg/symmetry"/>
   </shape>
   <shape type="Box" dx="R_in" dy="R_in" dz="Z_max"/>
  </shape>
  <shape type="Box" dx="R_out" dy="R_out" dz="Z_min"/>
 </shape>
</envelope>
```

- Envelope placed with a single line in the C++ driver

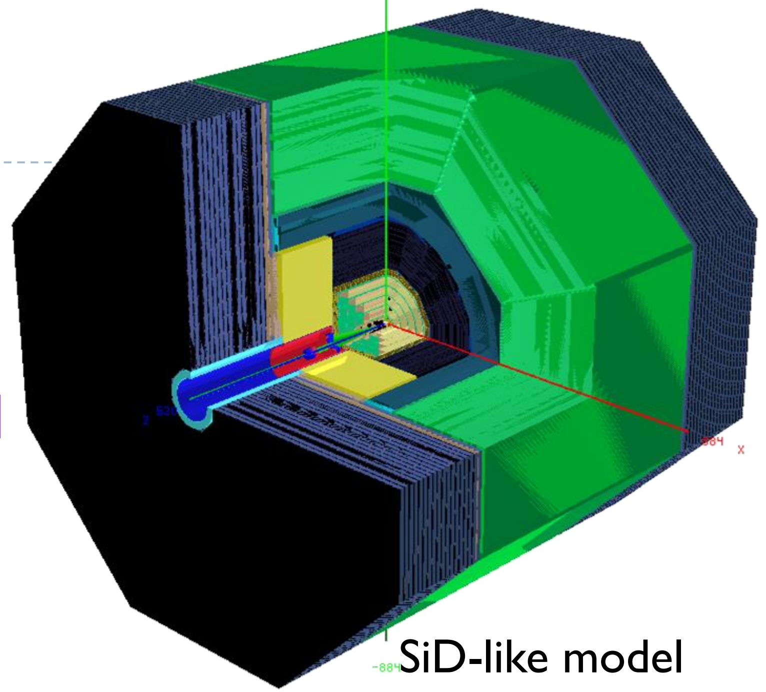
```cpp
Volume envelope = XML::createPlacedEnvelope(lcdd, element, sdet);
if (lcdd.buildType()==BUILD_ENVELOPE) return sdet;
```

- Use flag in `geoDisplay` to build a simplified geometry using only the envelopes
  - e.g. ILD Detector envelopes
- Could use envelopes in "Fast" simulation

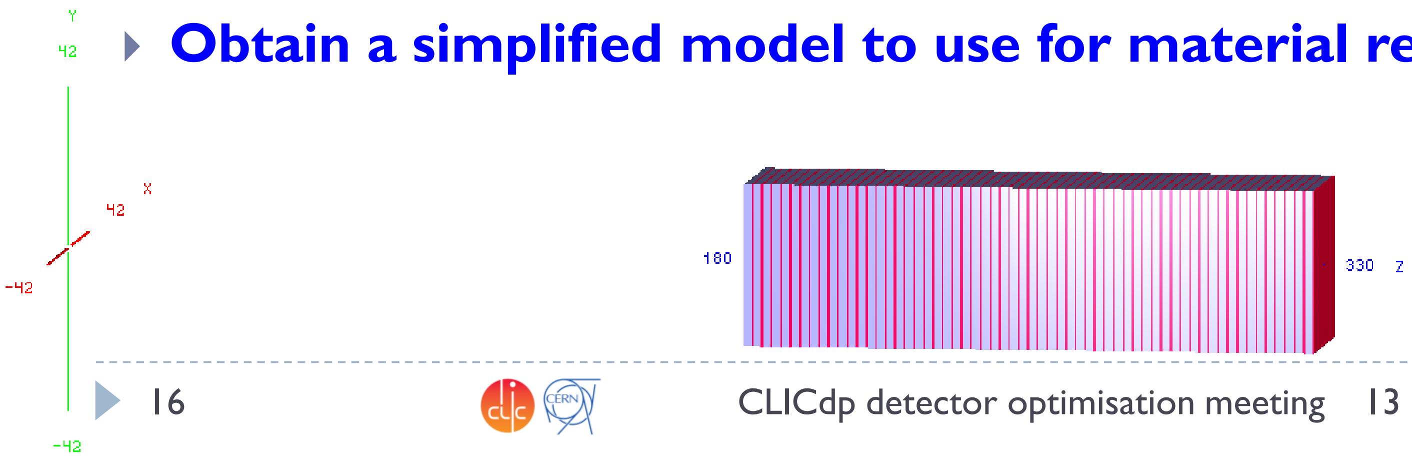


Driver flexibility

- **SiD model** example part of **DD4hep** package (right)
- Quick-n-dirty HCal stack below created from driver above in **1 min!**
  - **No code recompilation**
  - **Just modified compact xml file**
  - **Comment out includes of all other subdetectors**
  - **Leave just HCal Endcap for which I change symmetry from 8 to 4, set "outer radius" to 30 cm, "inner radius" to 0 and turn off reflection about the IP**
  - **Obtain a simplified model to use for material response studies**



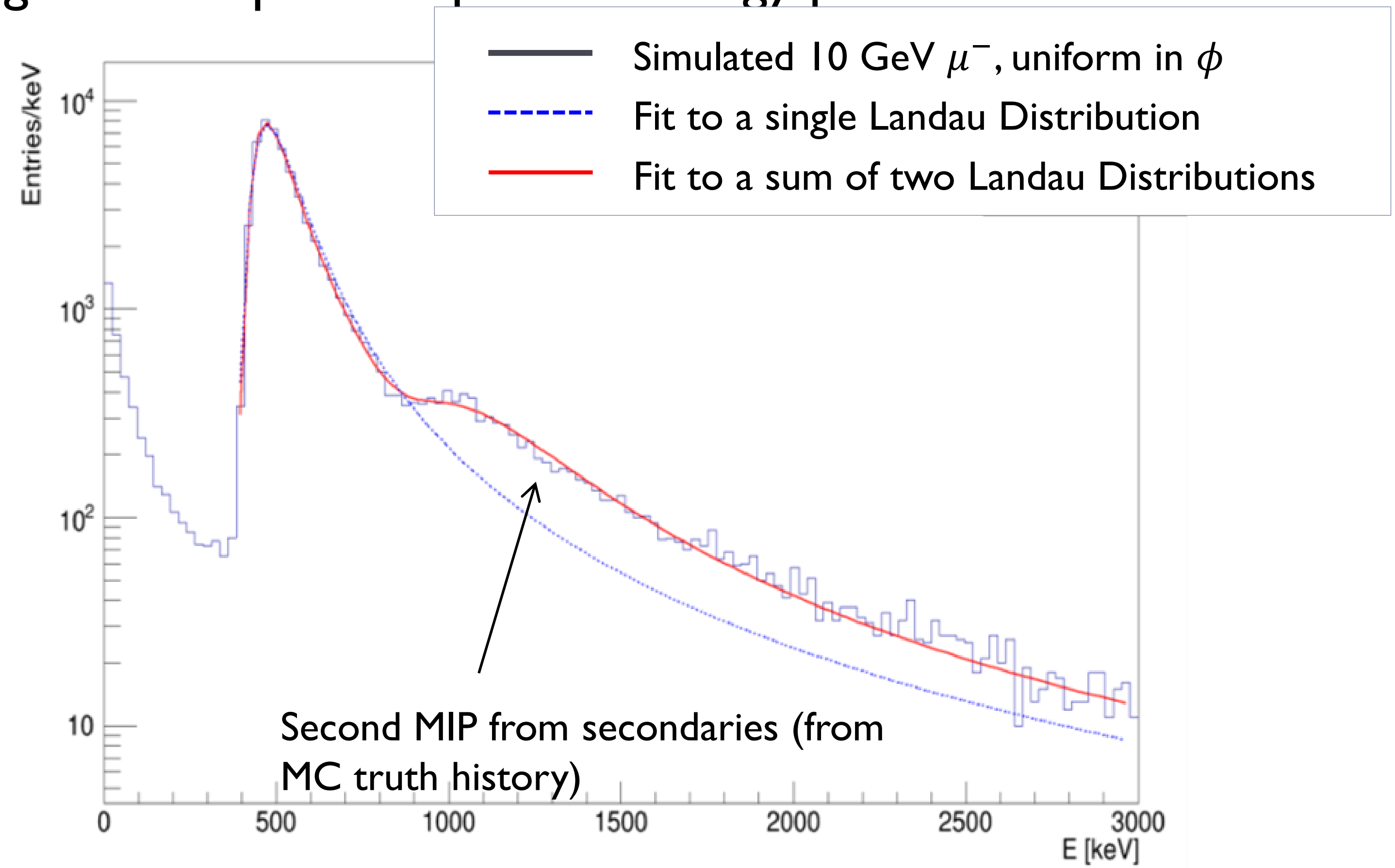
SiD-like model



HCal stack along z-axis (60 layers of steel interleaved with scinitllator)

Simulating single muons with DDSim

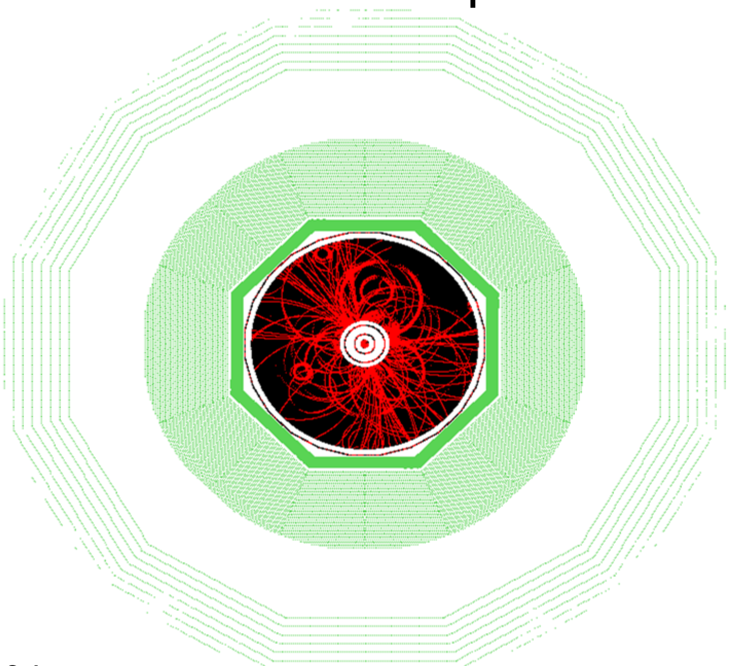
- We can validate the tool by using single particles
- Looking for example at deposited energy per hit in the HCal



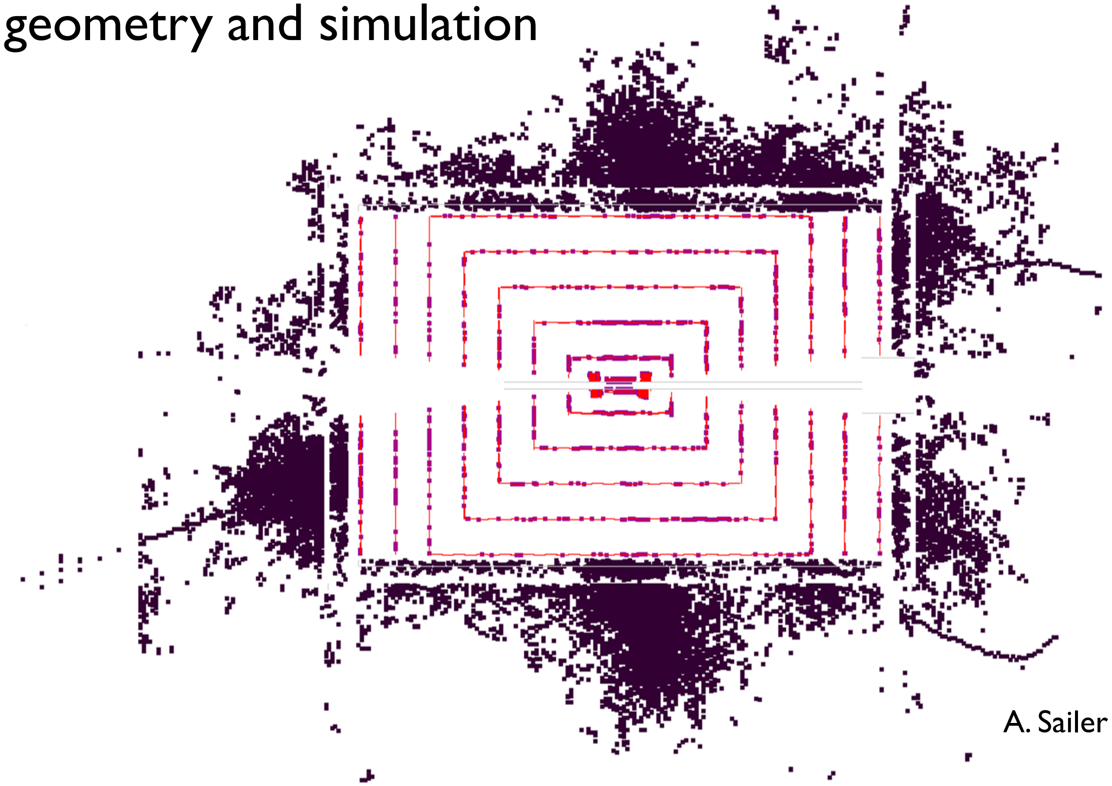
MC Particle history treatment built upon experience from **Mokka**/**SLIC**

Simulating physics events with DDSim

- Can use hit maps to validate geometry and simulation



S. Lu



A. Sailer

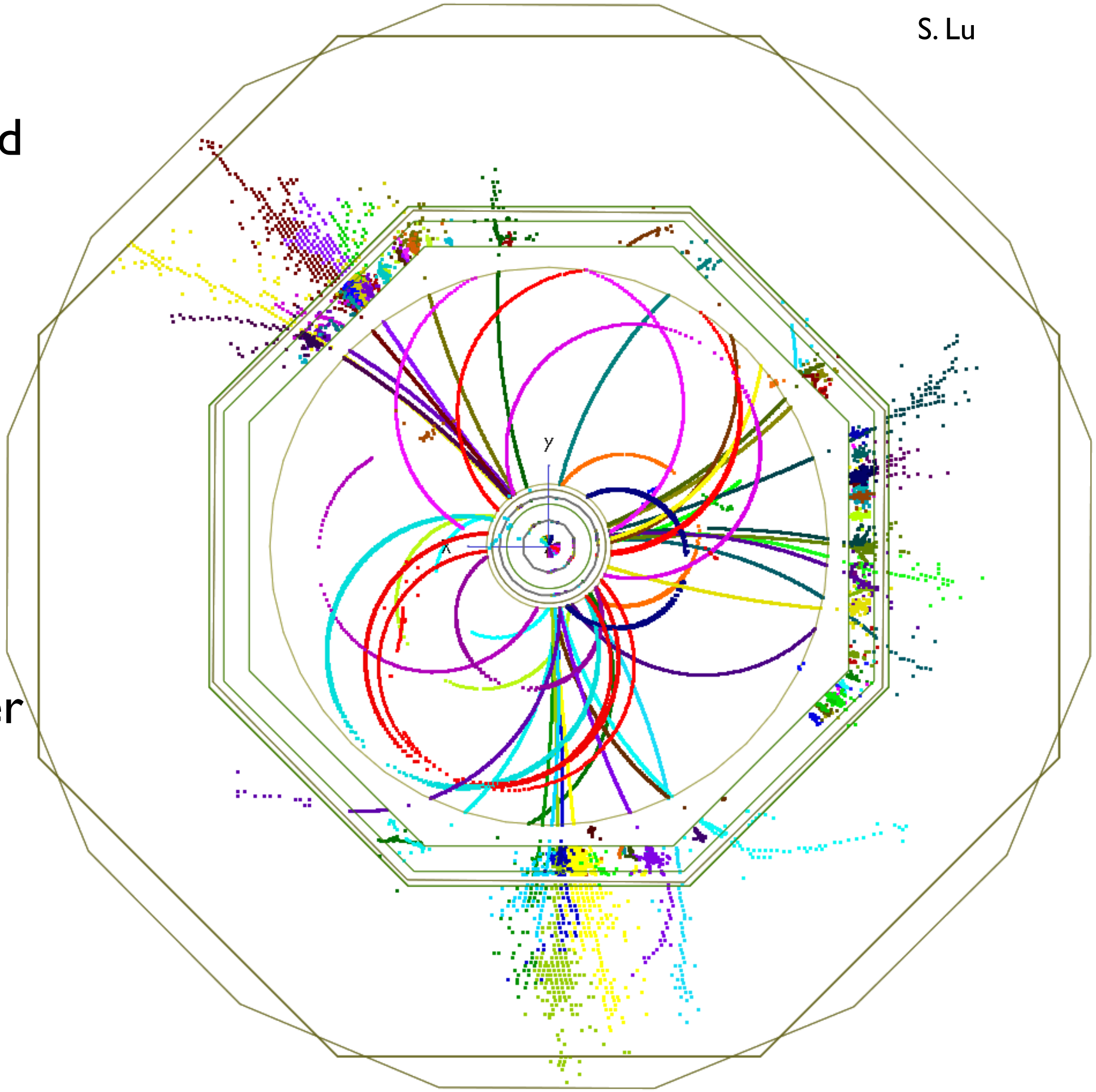
Hit map from $Z \rightarrow uds$ events at 500 GeV simulated in **ILD_o1_v05** using **DDsim**. **Green** shows calorimeter hits, **Red**/**Black** show tracker hits

A $t\bar{t}$ event at 500 GeV simulated in **CLIC_o2_v03** using **DDsim**. **Black** points are hits, **Red** lines are measurement surfaces, **Gray lines are auxiliary surfaces** used in reconstruction

Event Simulated, Reconstructed and Visualized Fully with DD4hep

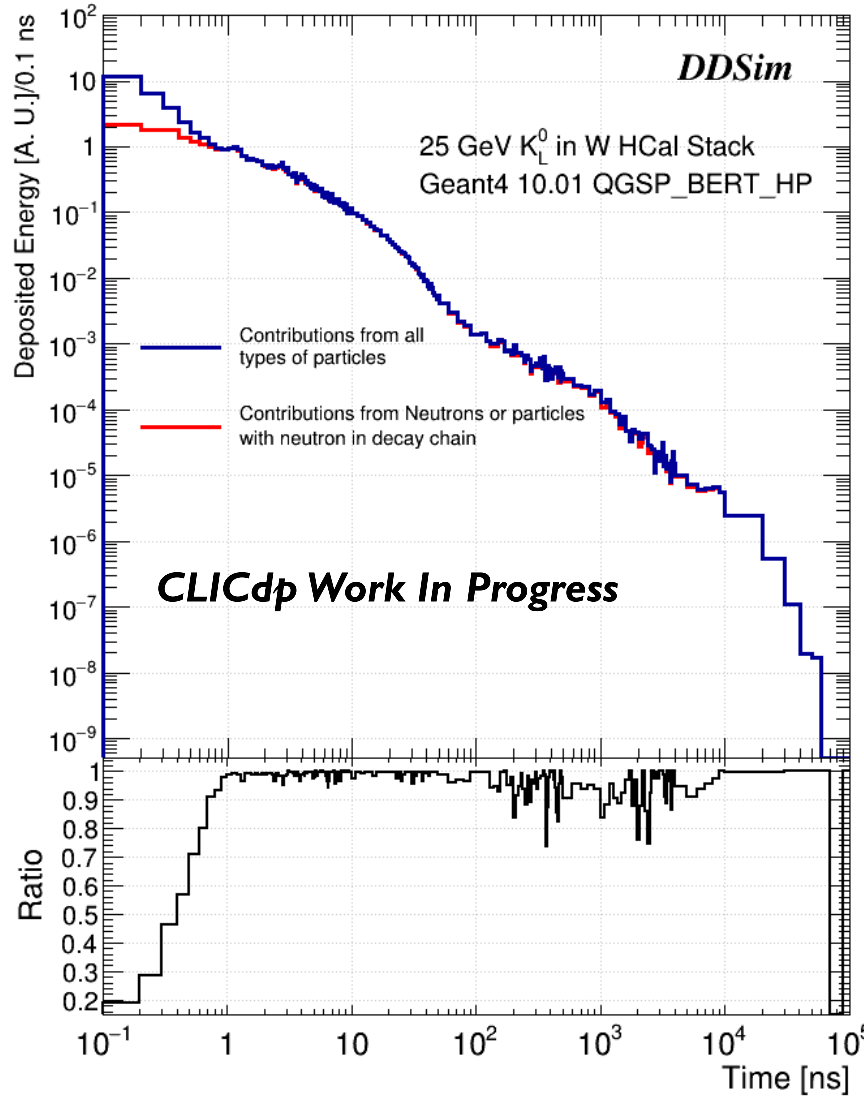
- **ILD_o1_v05** model implemented in **DD4hep**
- $Z \to uds$ event at $\sqrt{s} = 500$ GeV simulated in **DDSim**
- Tracks reconstructed using **DDSurfaces**
- PFOs from **DDMarlinPandora** using the **DDRec** data structures
- Event display from the **CED** viewer interfaced with **DD4hep**
  - Also uses **DDRec** and **DDSurfaces**

S. Lu



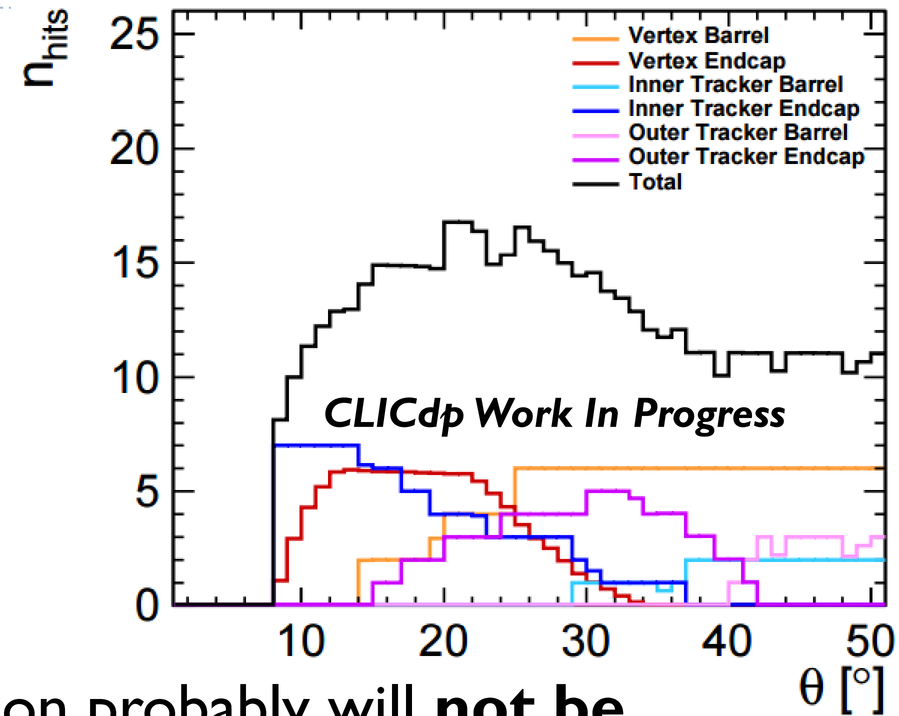
Detector optimization with DDSim

- Can have a larger more detailed MC Particle Truth Record by increasing "Tracking Region", lowering energy cuts
- E.g. expanded region to include calorimeters
  - Track provenance of every hit contribution in the hadronic shower
  - Try to understand timing in Fe/W



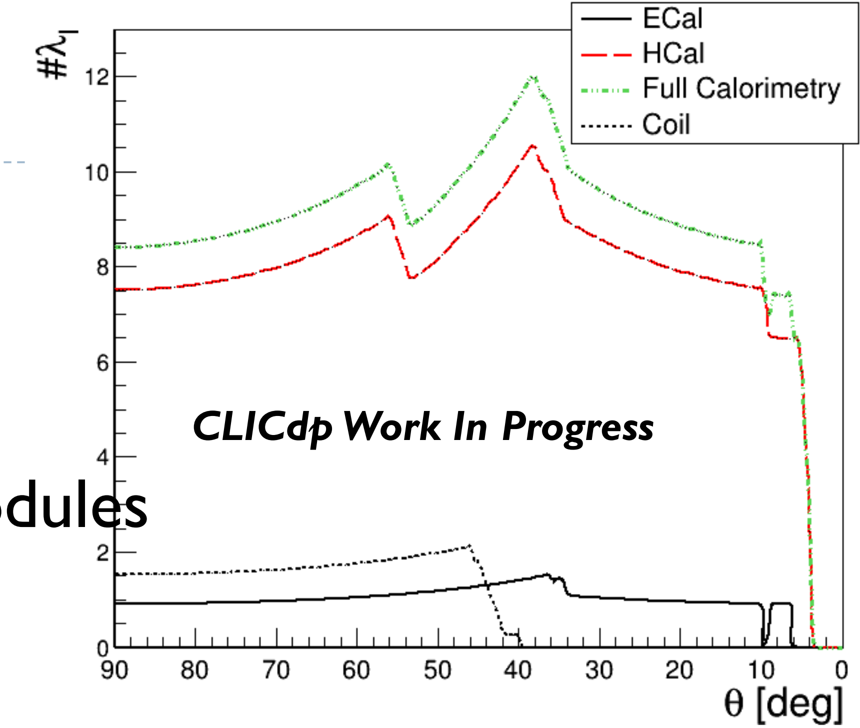
Detector optimization with DDSim

R. Simoniello



- **Control over sensitive detector actions**
- E.g. Use a tracker action that combines all interactions in the Silicon as one `SimTrackerHit`
  - Use **muon** tracks to count hit coverage w.r.t. angle
- **NB:** For physics events reconstruction probably will **not be** combining the hits in simulation [this will probably stay as the default tracker action]
  - Combine hits in the Digitization stage
  - Already simulating $Z \rightarrow uds$ and $t\bar{t}$ events up to 3 TeV to aid with Det. Optimization and Reconstruction software development

Geant4 material scan



- Can request a Geant4 UI to **interact with G4 Kernel**
  - csh-like, or Qt-based GUI
- Access to whatever Geant4 modules are loaded
  - E.g. material scan, visualization, …
- G4 Material scan can be restricted to regions
  - /control/matScan/region CalorimeterRegion
- It’s nice that in DD4hep regions can be defined and assigned to detectors trivially in the xml regardless of their shape

```
<detector id="DetID_HCAL_Barrel" name="HCalBarrel" type="HCalBarrel_o1_v01“ readout="HCalBarrelHits" vis="HCALVis"
region="CalorimeterRegion" >
. . .
</detector>
```

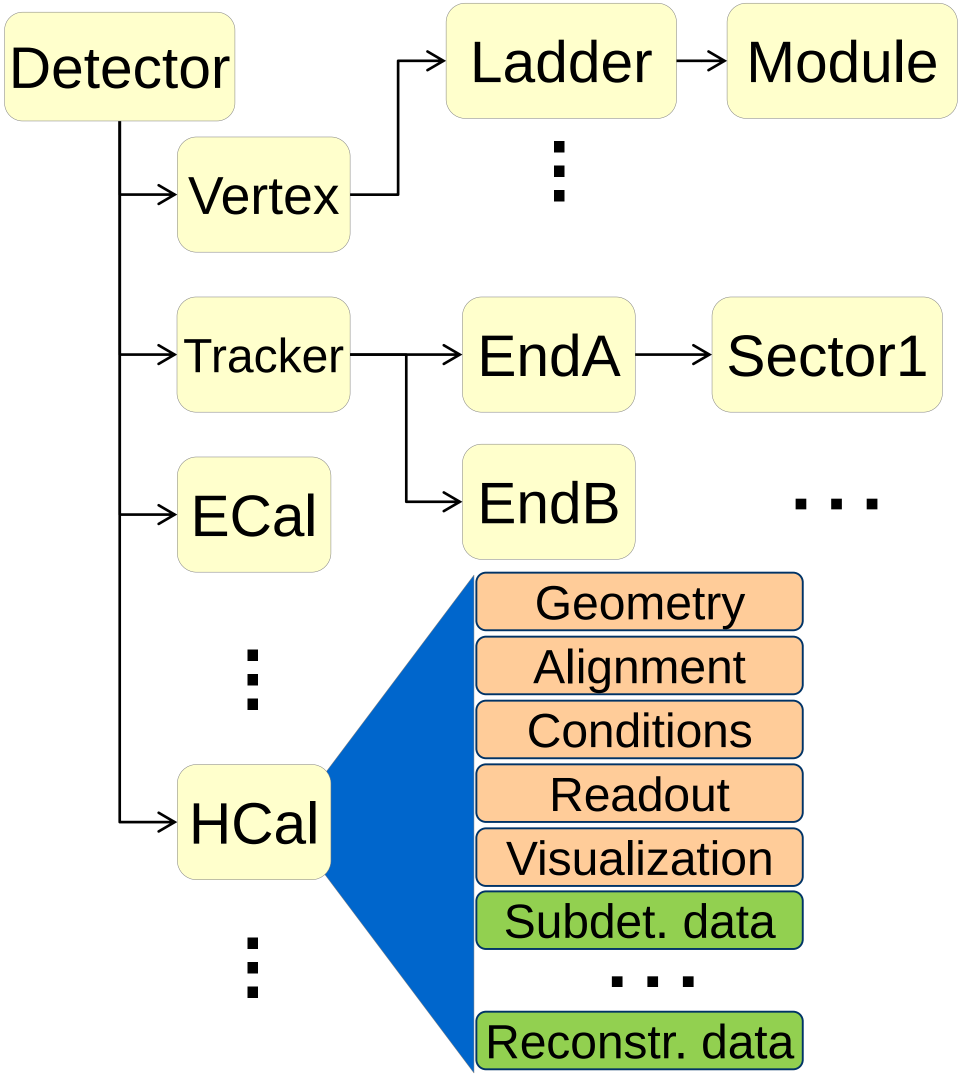
Summary and outlook

- **DD4hep** provides consistent single source of detector geometry for simulation, reconstruction, analysis
- Additional package **LCgeo** holds developing implementations of Detector Model geometries for CLICdp and ILD
- **DDSim is a new, flexible simulation tool using DD4hep's DDG4 interface to Geant4**
- Already in use by LC and FCC Communities
  - Full integration with **iLCsoft** software framework almost complete
- Development continues in parallel with validation

BACKUP SLIDES

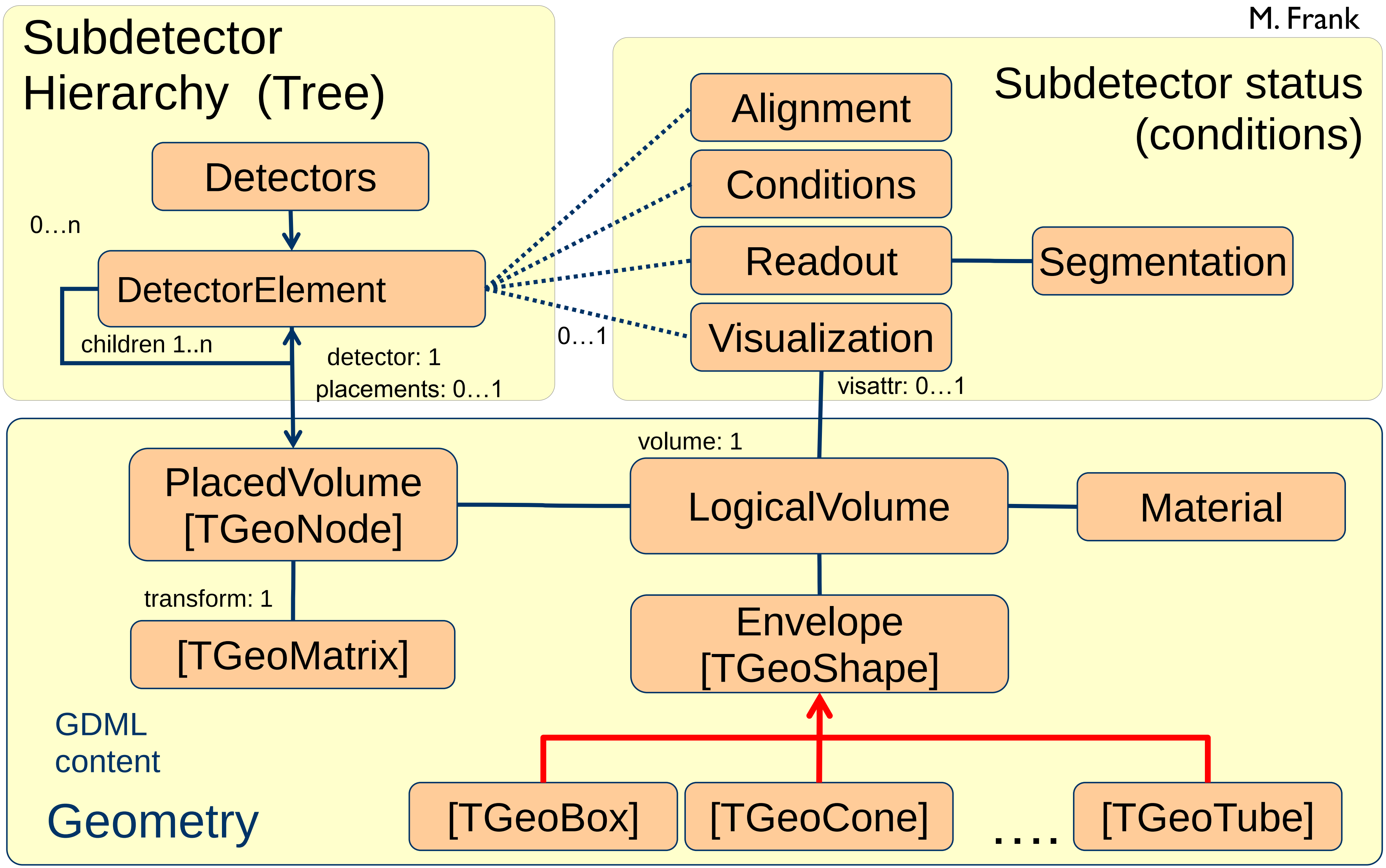
What is Detector Description

- Description of a tree-like hierarchy of **"detector elements"**
  - Subdetectors or parts of subdetectors
- Detector Element describes
  - Geometry
  - Environmental conditions
  - Properties required to process event data
  - Extensions (optionally): experiment, sub-detector or activity specific data, measurement surfaces, …



M. Frank

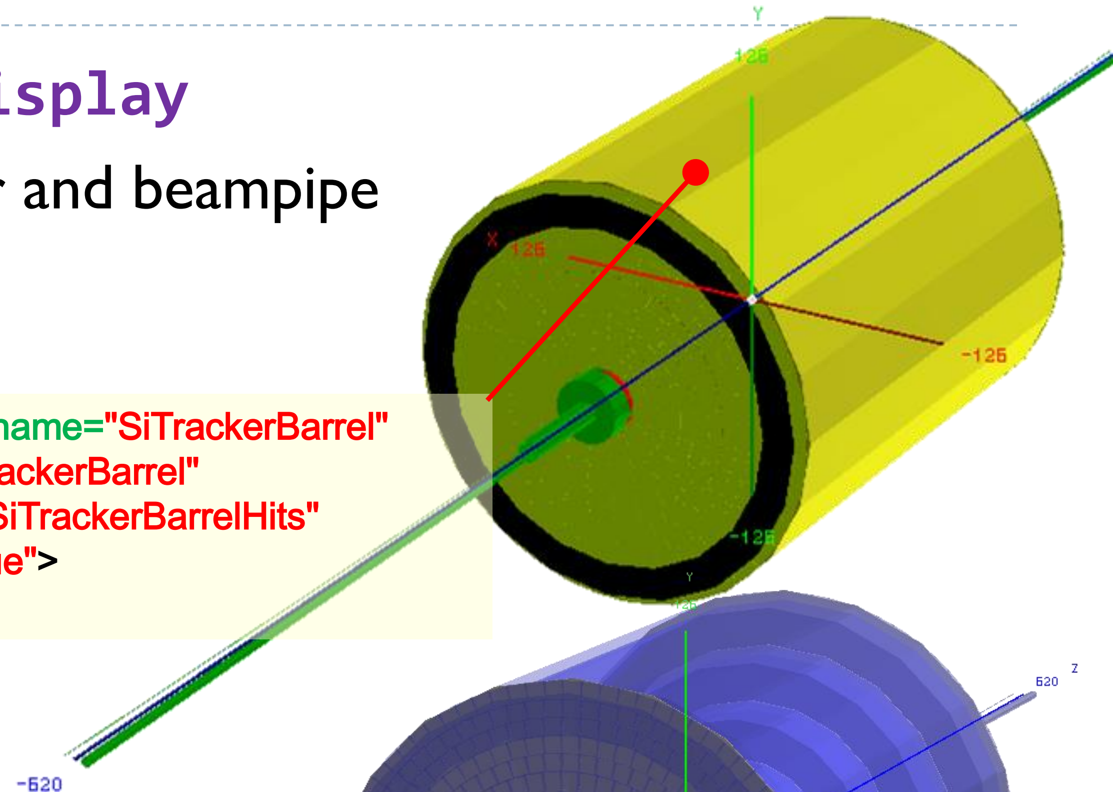
Geometry Implementation

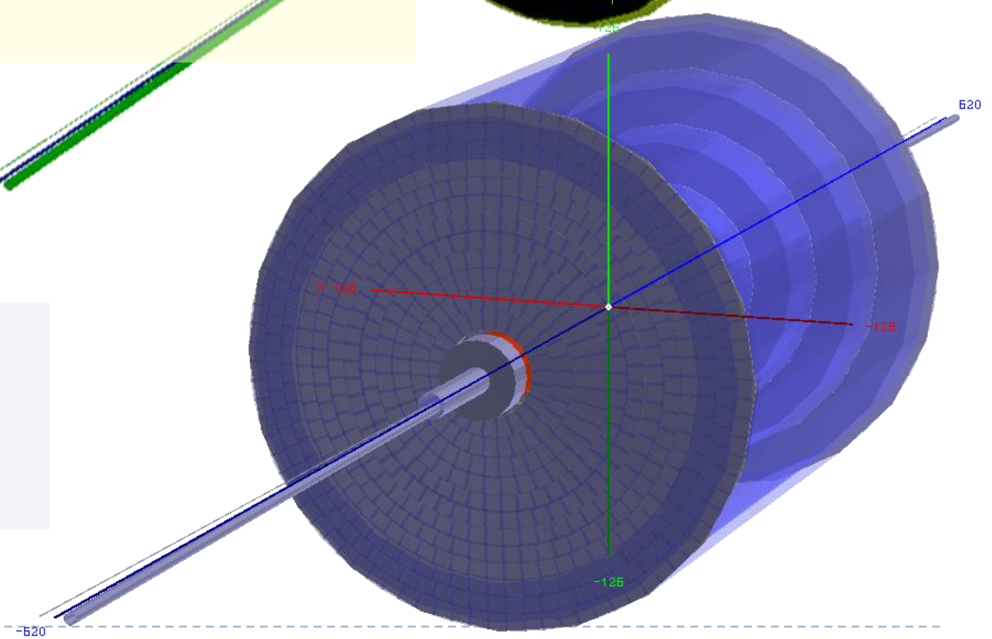


CLIC_SID_CDR Tracker

- Visualized here in **geoDisplay**
- Around Vertex Detector and beampipe

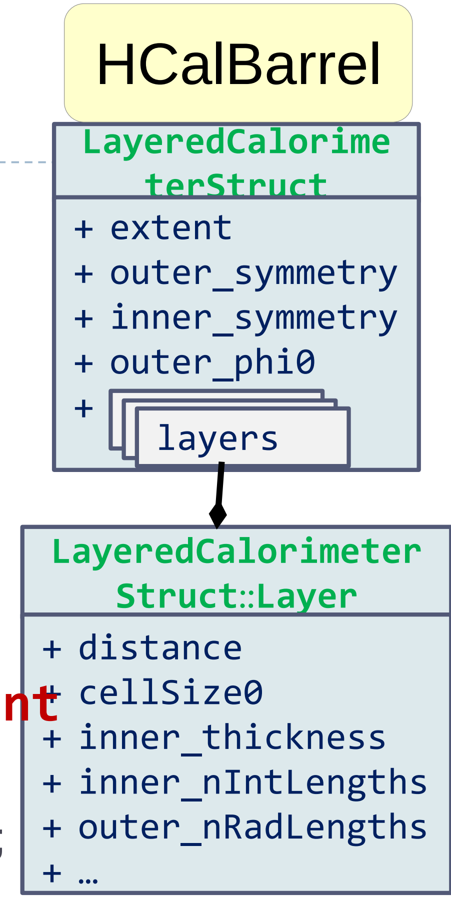
```
<detector name="SiTrackerBarrel"
type="SiTrackerBarrel"
readout="SiTrackerBarrelHits"
reflect="true">
```





The same tracker visualized with ROOT's TGeoManager using and intermediate GDML file dumped from Geant4 after loading geometry from DD4hep

DDRec: Reconstruction extensions



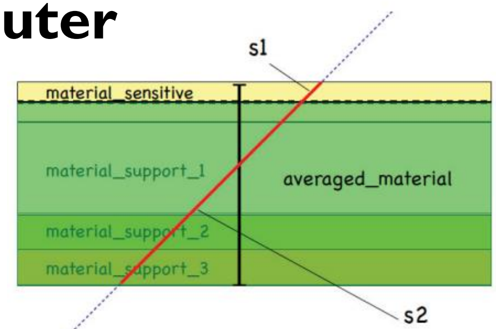
Extend subdetector driver with arbitrary user data

- Summary of more *abstract* information useful for **reconstruction**
- Populate during driver construction
  - Driver has the all the information
  - Take advantage of material map
- e.g: attach a `LayeredCalorimeterStruct` to the `DetElement` for HCalBarrel
  - `sdet.addExtension<DDRec::LayeredCalorimeterData>(caloData);`
- Additional *simple* data structures available
- Users can even attach their own more complicated objects
- Other use cases: auxiliary information for tracking, slimmed-down geometry for a faster event display (e.g. CED[†])

† http://ilcsoft.desy.de/portal/software_packages/ced/

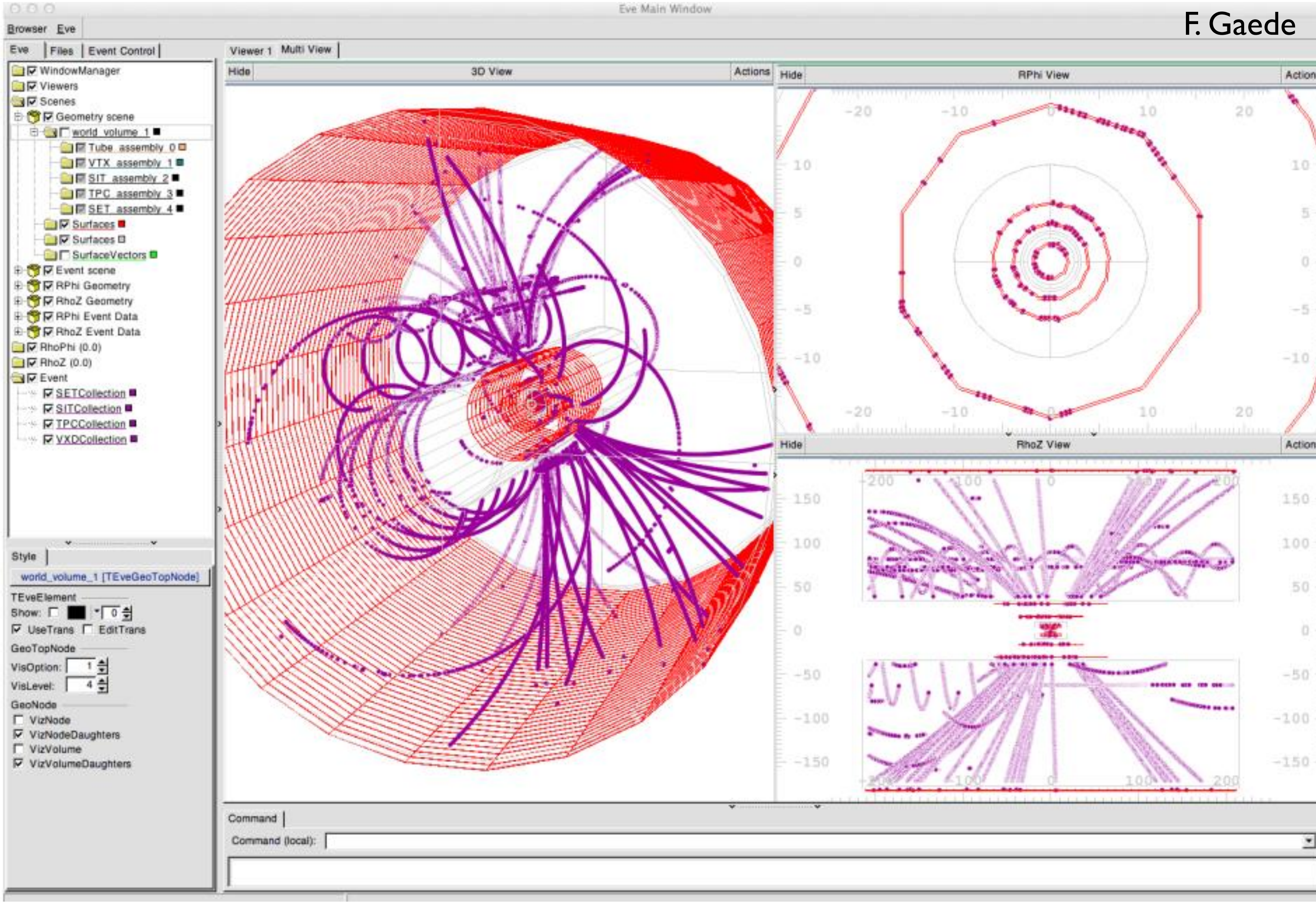
Measurement surfaces

- Special type of extension, used primarily in **tracking**
  - Did not find an implementation in TGeo
  - Implemented in DDRec/DDSurfaces
- Attached to `DetElements` and `Volumes` (defining their boundaries)
  - Can be added to drivers via **plugins** without modifying detector constructor
- They hold **u**,**v**,**n**ormal and **o**rigin vectors and **inner/outer thicknesses**
- Material properties **averaged automatically**
- Could also be used for **fast simulation**

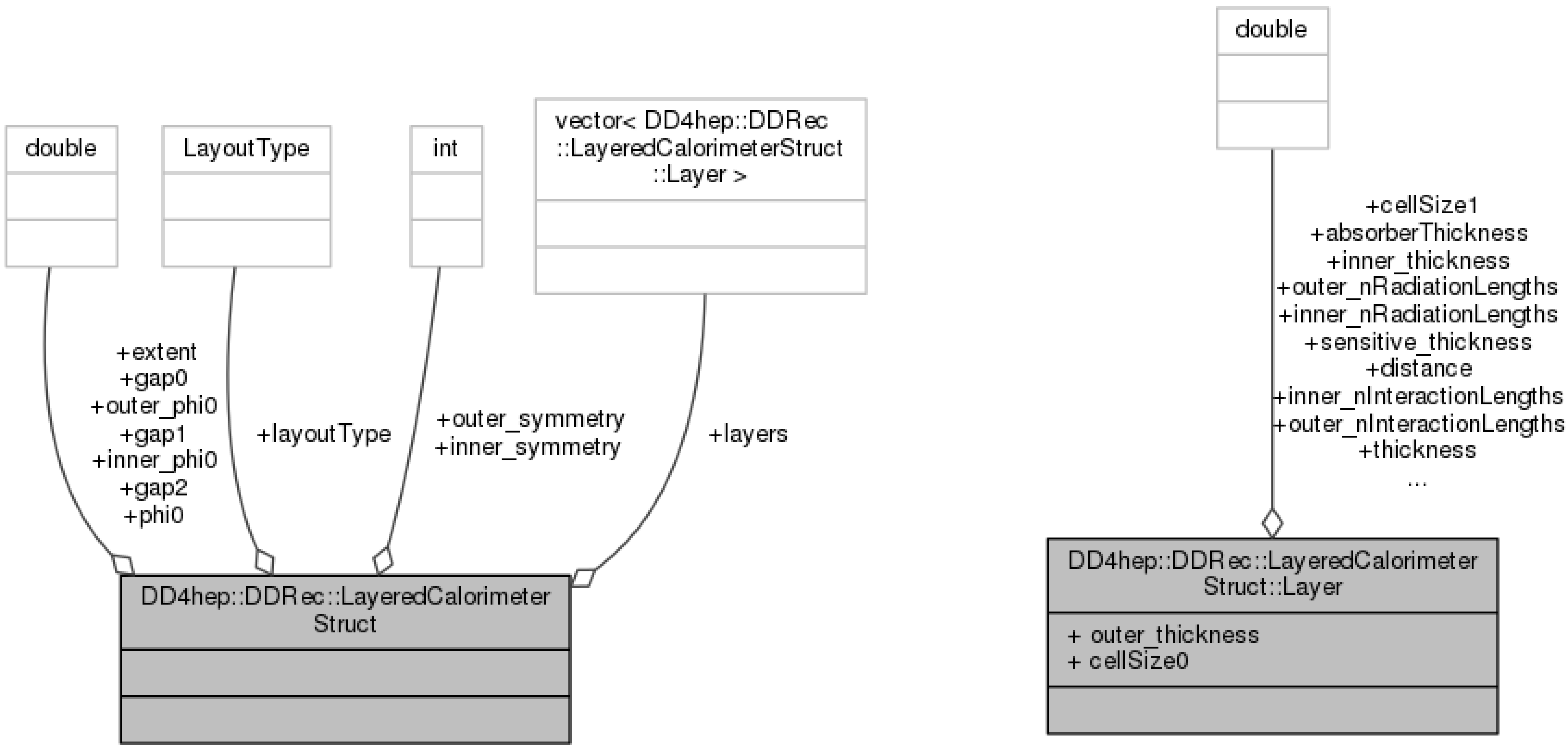


- Outlines of surfaces drawn in `teveDisplay` for CLICdp Vertex Barrel and Spiral Endcaps

Surfaces and Hits in teveDisplay



LayeredCalorimeterStruct



```
for (xml_coll_t c(x_det, _U(layer)); c; ++c) {
  xml_comp_t x_layer = c;
  int repeat = x_layer.repeat();            // Get number of times to repeat this layer.
  const Layer* lay = layering.layer(layer_num - 1); // Get the layer from the layering engine.
  // Loop over repeats for this layer.
  for (int j = 0; j < repeat; j++) {
    string layer_name = _toString(layer_num, "layer%d");
    double layer_thickness = lay->thickness();
    DetElement layer(stave, layer_name, layer_num);
    DDRec::LayeredCalorimeterData::Layer caloLayer ;
     // Layer position in Z within the stave.
    layer_pos_z += layer_thickness / 2;
    // Layer box & volume
    Volume layer_vol(layer_name, Box(layer_dim_x, detZ / 2, layer_thickness / 2), air);

    // Create the slices (sublayers) within the layer.
    double slice_pos_z = -(layer_thickness / 2);
    int slice_number = 1;
    double totalAbsorberThickness=0.;

    for (xml_coll_t k(x_layer, _U(slice)); k; ++k) {
      xml_comp_t x_slice = k;
      string slice_name = _toString(slice_number, "slice%d");
      double slice_thickness = x_slice.thickness();
      Material slice_material = lcdd.material(x_slice.materialStr());
      DetElement slice(layer, slice_name, slice_number);

      slice_pos_z += slice_thickness / 2;
      // Slice volume & box
      Volume slice_vol(slice_name, Box(layer_dim_x, detZ / 2, slice_thickness / 2), slice_material);
       if (x_slice.isSensitive()) {
        sens.setType("calorimeter");
        slice_vol.setSensitiveDetector(sens);
      }
      // Set region, limitset, and vis.
      slice_vol.setAttributes(lcdd, x_slice.regionStr(), x_slice.limitsStr(), x_slice.visStr());
      // slice PlacedVolume
      PlacedVolume slice_phv = layer_vol.placeVolume(slice_vol, Position(0, 0, slice_pos_z));

      slice.setPlacement(slice_phv);
      // Increment Z position for next slice.
      slice_pos_z += slice_thickness / 2;
      // Increment slice number.
      ++slice_number;
    }
```

Example HCal Barrel Driver

- Always within a function called

```
static Ref_t
create_detector(LCDD&
lcdd, xml_h e,
SensitiveDetector sens)
{
…
return sdet;
}
```

- Macro to declare detector constructor at the end:

```
DECLARE_DETELEMENT(HCalB
arrel_o1_v01,
create_detector)
```