DD4hep - Geometry Description for HEP Experiments

Christian Grefe

CERN

28. May 2013

Outline

Disclaimer

- This is work in progress - DD4hep is still in development
- Expect changes of the API, functionality or class hierarchy
- You are welcome to try it out - some things might not work, yet
- Let us know about your experience and ideas for improvement
- People involved: Markus Frank (CERN), Frank Gaede (DESY), André Sailer (CERN), Jan Strube(CERN), C.G.
- Tutorial slides by Markus Frank (LC Software Meeting 2013): `http://indico.cern.ch/conferenceDisplay.py?confId=228477`

Ideas Behind DD4hep

- Follow the proven design of the SiD GeomConverter
  - Detector constructors define the available sub detector prototypes (similar also to Mokka drivers)
  - Define the full detector in a human-readable XML format: compact.xml
  - Decoupled from full simulation software through a well defined interface: LCDD
- Use TGeo for the internal representation of the geometry
- Provide plug-ins that allow export of the geometry to other format, e.g. LCDD
- Link against DD4hep to navigate the geometry at run-time, e.g. reconstruction

Getting DD4hep

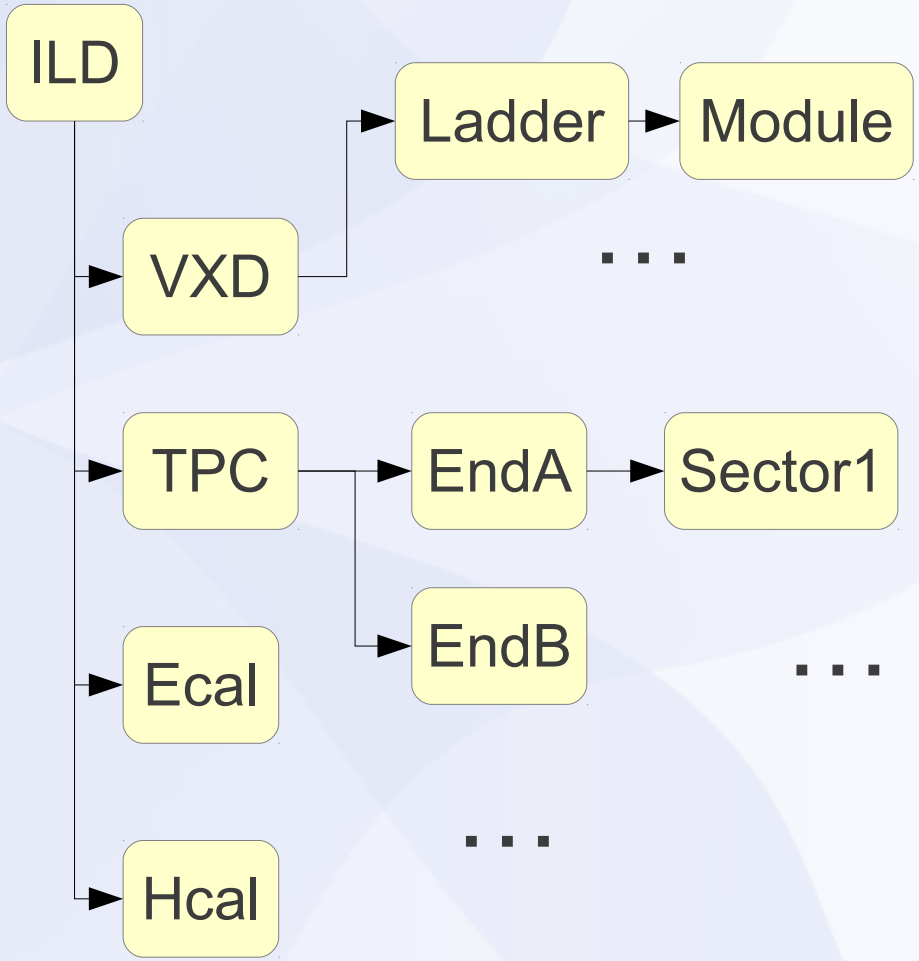
- DD4hep website: `http://aidasoft.web.cern.ch/DD4hep`
- SVN repository: `https://svnsrv.desy.de/public/aidasoft/DD4hep/trunk/`
- Builds with cmake (see ReadMe.txt for instructions)
- Before calling "make" need to source `thisdd4hep.sh` which is generated by cmake
- Without the library paths set the compilation will fail!
- Latest tag included in ILCSoft v01-17-01 (released last week)

Core C++ Interface

- All objects available that are required to fully describe the detector geometry: detector element, sensitive detector, material, alignment entry, readout, visualization, limit set, region, field
- Objects are handles
  - Copy is very cheap - just copy underlying pointer
  - Objects are created only once - consistent geometry for all clients
  - Lookup by name
- These objects are usually created by reading a compact.xml

Detector Elements

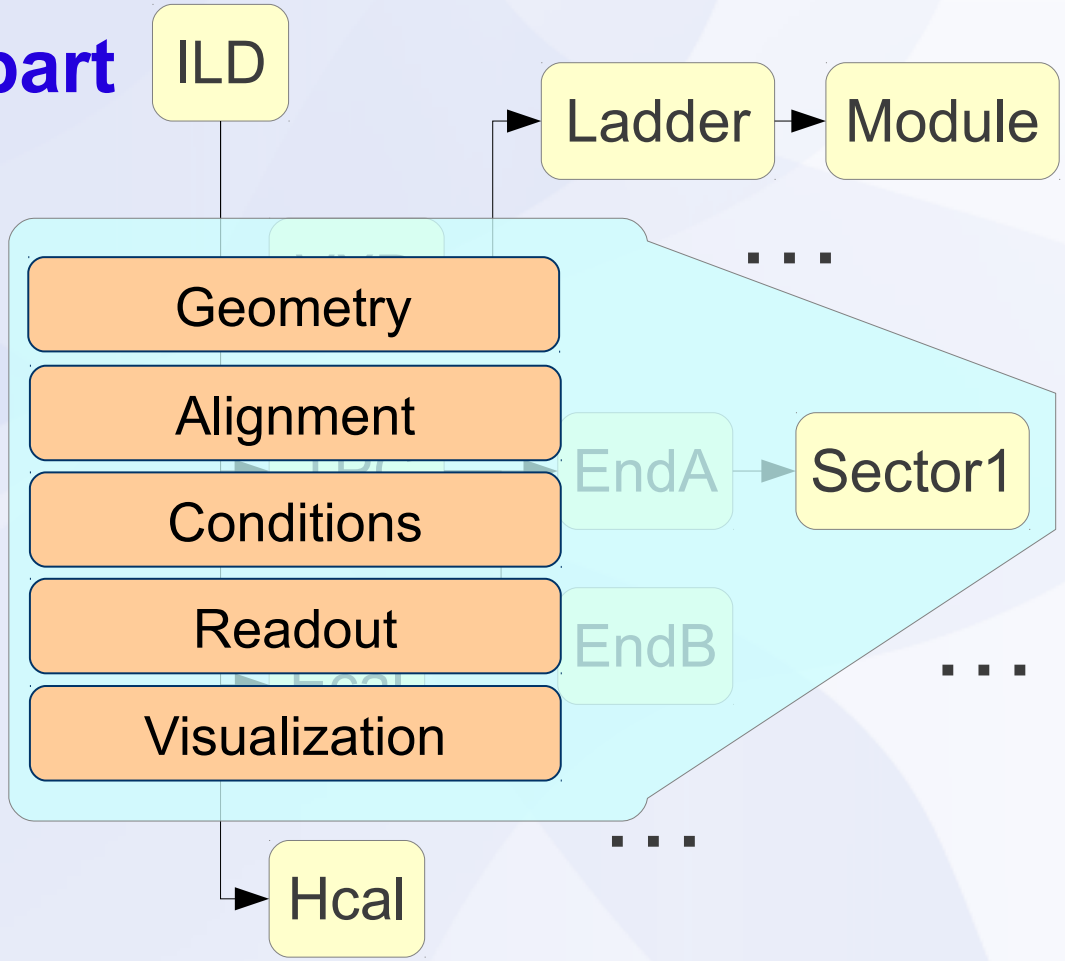
- **Description of a tree-like hierarchy of "detector elements"**
  - **Subdetectors or parts of subdetectors**
  - **Example:**
    - Experiment
    - TPC
    - Endcap A/B
    - Sector
    ...



M. Frank

Detector Elements

- **Subdetector or the part of a subdetector including the description of its state**
  - **Geometry**
  - **Environmental conditons**
  - **Properties required to process event data**



M. Frank

Compact XML Description

- **lccdd** *Linear collider compact detector description*
  - **includes** *XML include files for material DB*
  - **info** *Info about the detector model, author etc.*
  - **define** *Constant definitions*
  - **materials** *Extensions to material DB*
  - **display** *Visualization settings*
  - **detectors** *Subdetector definitions*
  - **readouts** *Readout information for simulation*
  - **limits** *Limitsets for simulation*
  - **fields** *Electric/magnetic field definitions*

Also in DD C++ API

M. Frank

Defining a Subdetector

- The detectors section is the core part of the compact XML

```
<detectors>
    <detector id="1" name="VertexBarrel" type="SiliconBarrelTracker">
        ...
    </detector>
</detectors>
```

- Mandatory elements:
  - name - defines the unique identifier for this detector element
  - type - defines which detector constructor to use
- Optional elements:
  - limits - apply a GEANT4 limit set to this subdetector
  - vis - apply a set of visualization attributes to this subdetector
  - readout - link this subdetector to a readout object, also defines name of output collection

Defining a Subdetector

- The structure below the detector element is completely up to the detector type
- Necessary to allow subdetectors of arbitrary complexity

```
<detectors>
    <detector id="1" name="VertexBarrel" type="SiliconBarrelTracker">
        <module name="module1">
            ... additional elements ...
        </module>
        <layer id="1" module="module1">
            ... additional elements ...
        </layer>
    </detector>
</detectors>
```

- With great power comes great responsibility:
  ⇒ Use sensible attribute and element names
  ⇒ Stick to conventions used in similar detector types

Detector Constructors

- Each detector constructor defines a new subdetector type usable from XML

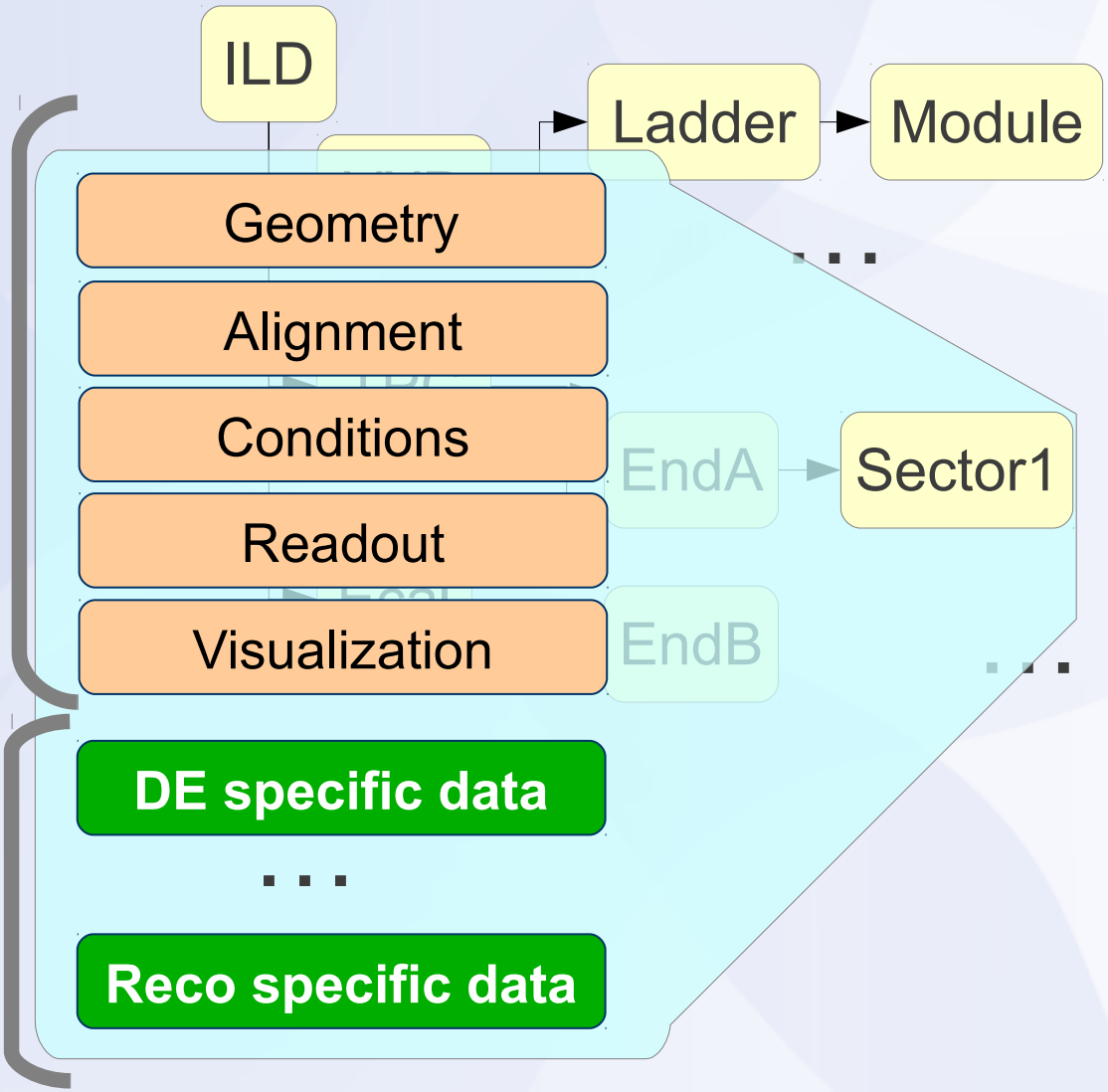
```
static Ref_t create_detector(LCDD& lcdd, xml_h e, SensitiveDetector sens) {
    xml_det_t  x_det     = e;
    string     det_name  = x_det.nameStr();
    string     det_type  = x_det.typeStr();
    Material   air       = lcdd.air();
    DetElement sdet(det_name,x_det.id());
    Volume     motherVol = lcdd.pickMotherVolume(sdet);
    int n = 0;
    for(xml_coll_t i(x_det,_U(layer)); i; ++i, ++n) {
        xml_comp_t x_layer = i;
        string  l_name = det_name+_toString(n,"_layer%d");
        DetElement layer(sdet,_toString(n,"layer%d"),x_layer.id());
        Tube    l_tub;
        Volume  l_vol(l_name,l_tub,air);
        double  z    = x_layer.outer_z();
        double  rmin = x_layer.inner_r();
        double  r    = rmin;
        int m = 0;
        for(xml_coll_t j(x_layer,_U(slice)); j; ++j, ++m) {
            // layer logic ...
        }
        l_tub.setDimensions(rmin,r,z,0,2*M_PI);
        l_vol.setVisAttributes(lcdd,x_layer.visStr());
        PlacedVolume lpv = motherVol.placeVolume(l_vol,IdentityPos());
        lpv.addPhysVolID("system",sdet.id()).addPhysVolID("barrel",0);
        PlacedVolume lpv = motherVol.placeVolume(l_vol,IdentityPos());
        lpv.addPhysVolID("system",sdet.id()).addPhysVolID("barrel",0);
        layer.setPlacement(lpv);
    }
    return sdet;
}
DECLARE_DETELEMENT(MultiLayerTracker,create_detector);
```

Detector Views

- Using the detector element directly can be cumbersome:
  $\Rightarrow$ define specific views that define short-cuts and define a high level interface
- Separates data from behavior
- A view does not need to be a DetElement
- Typically you want to provide a constructor taking a reference of a DetElement
- For simple views it can be sufficient to just wrap the logic to access a certain information in the DetElement
- For more complex use cases an Extension can be attached to a DetElement (one per extension type)
- This extension type can be abstract $\Rightarrow$ allows to have a generic calorimeter extension with specific behavior for the various calorimeter detector types
- Extensions can be attached at construction time or any time later, e.g. in the reconstruction code

DetElement Extensions

- **Default DetElement data**
- **Added subdetector specific data**

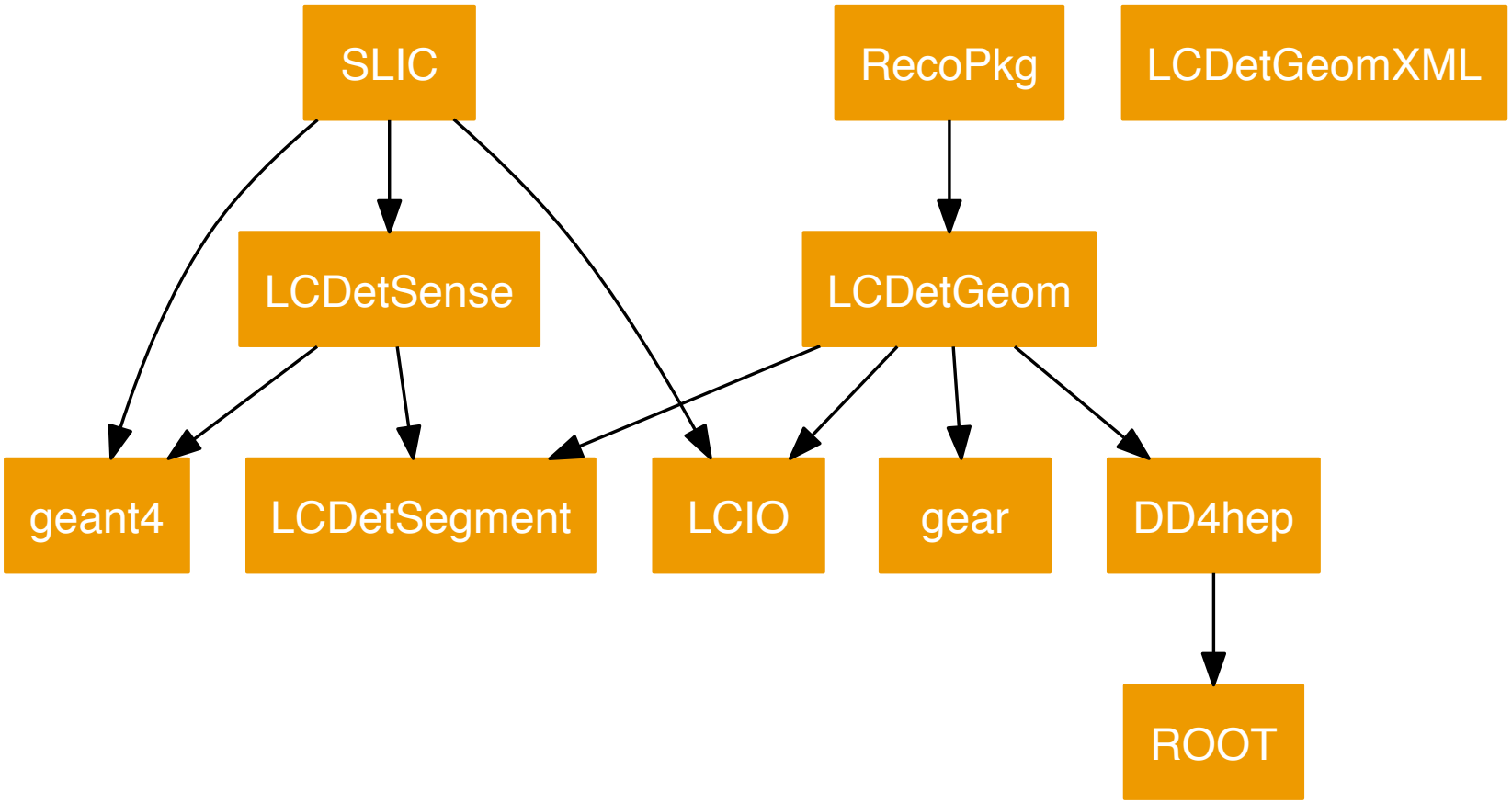


M. Frank

Reconstruction Interface

- Currently in the process of defining several high level reconstruction views that can fulfill all our reconstruction requirements (GEAR replacement)
- Keep these interfaces as generic as possible, for example:
  layered detector, calorimeter, tracker, cylinder, polyhedra, barrel, endcap
- Allows to mix interfaces, e.g. CylindricalBarrelCalorimeter
- Where necessary each of those views will have a corresponding abstract extension
- See my branch in SVN for some examples

Package Dependencies (still under discussion)

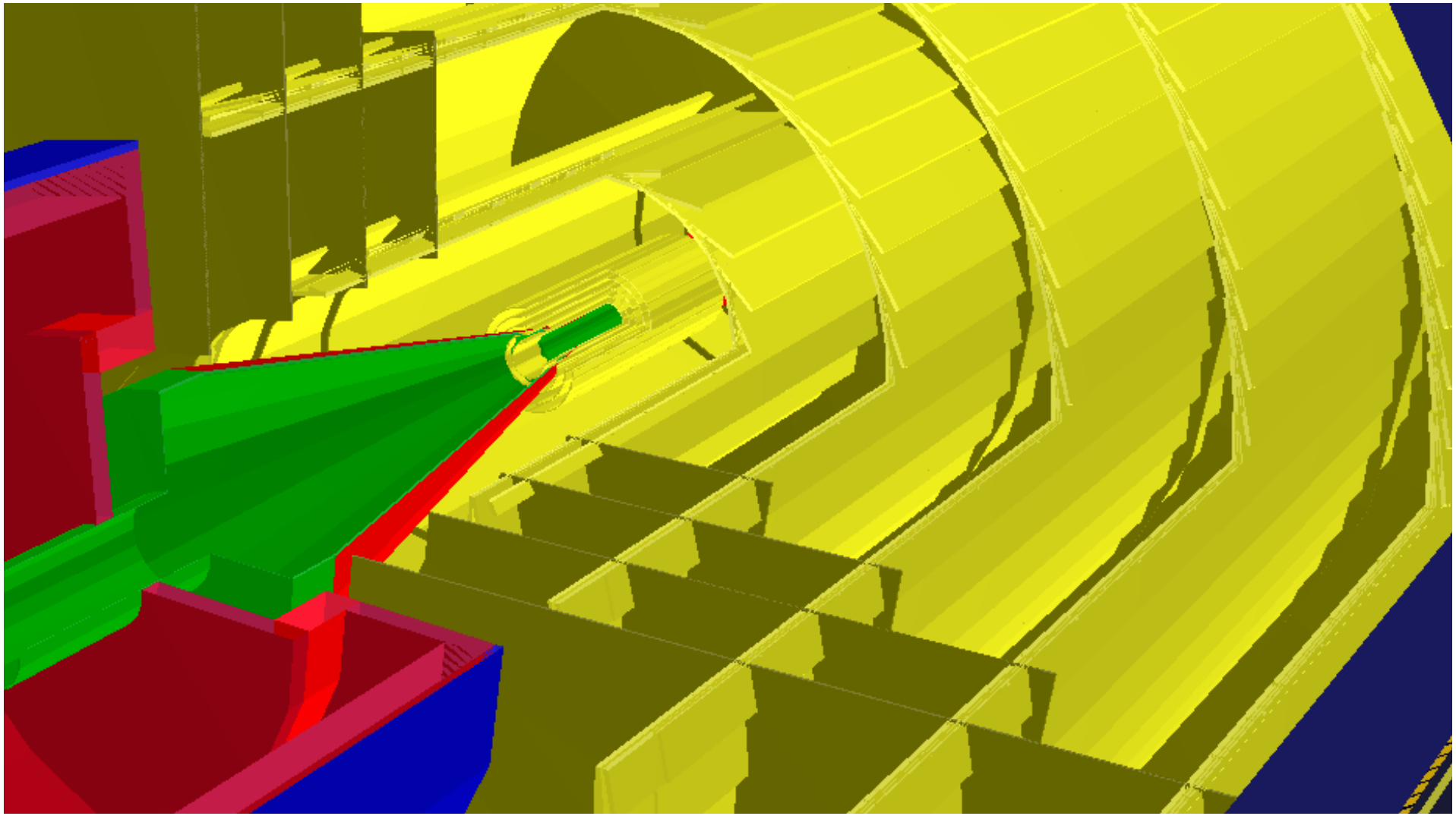


F. Gaede

Status and Plans

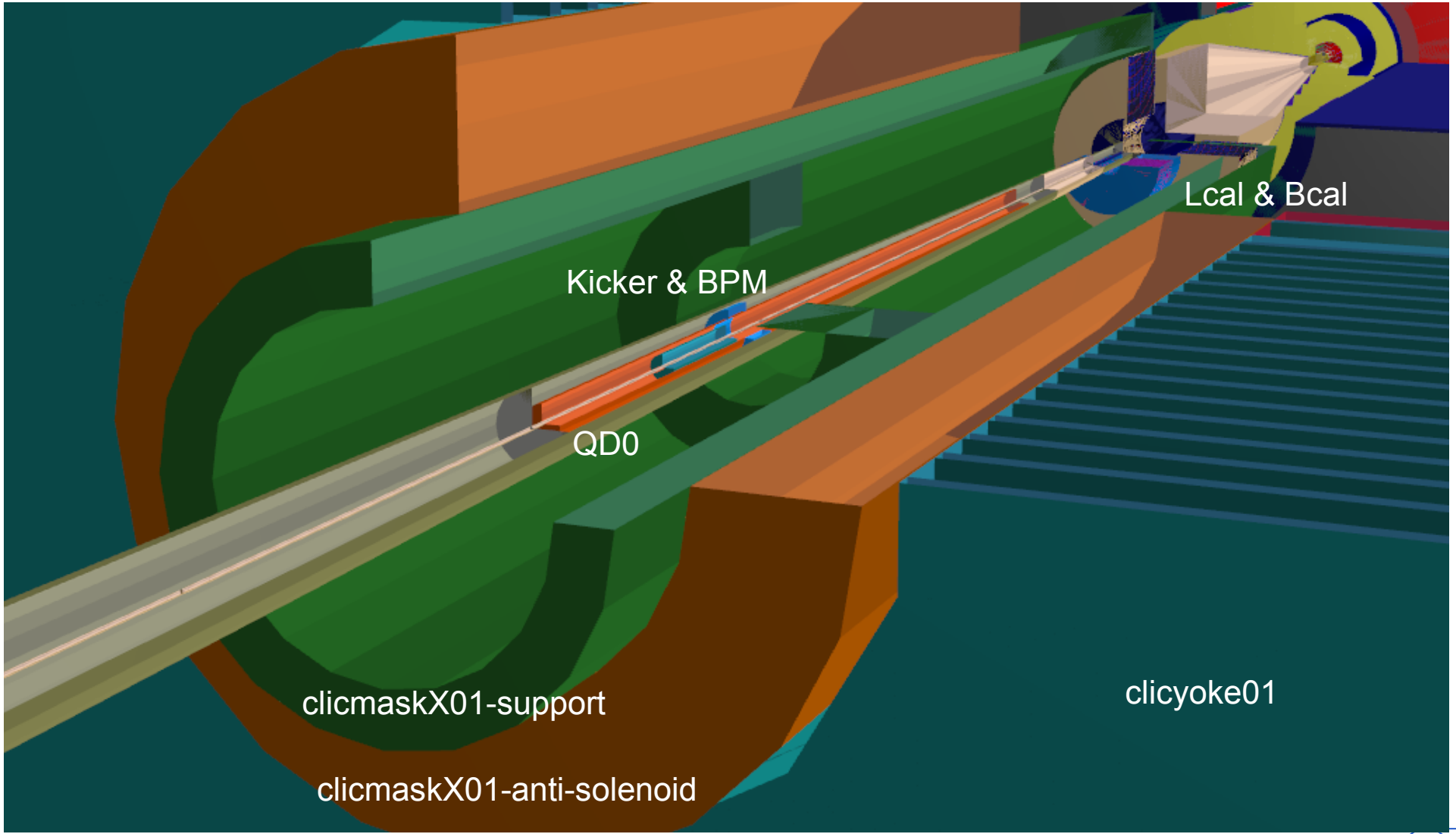
- Core features fully implemented
- Many detector constructors implemented that resemble the subdetectors used in CLIC_SiD and CLIC_ILD
- Work currently in progress:
  - Definition of reconstruction interface for some example cases: forward calorimeters (A. Sailer), barrel calorimeters (C.G.)
  - Cell ID and detector ID encoding / decoding to allow look-up of closest detector element based on hit cell ID $\Rightarrow$ find the sensitive layer to get the local coordinate system
- Plan to have fully functional simulation and reconstruction examples in coming months (full chain)
- Review design choices before moving forward
- Implement all subdetectors and their reconstruction interfaces required by the latest ILD and SiD models
- Will require involvement of subdetector experts
- Stable version with full functionality in 2014

CLIC_SiD Tracker



M. Frank

CLIC_ILD Forward Region



M. Frank