slic
A Geant4-based detector simulation package

Norman Graf
(SLAC)

EIC Software Consortium Meeting

October 16, 2017

Simulation Mission Statement

- Provide full simulation capabilities for physics program:
  - Physics signal & beam background simulations
  - Detector designs
  - Trigger simulations
  - Reconstruction and analysis
- Need flexibility for:
  - Optimizing detector geometries
  - Different reconstruction algorithms
  - Different machine environments
- Limited resources demand efficient solutions, focused effort.

Overview: Goals

- Facilitate contribution from physicists in different locations with various amounts of time available.
- Use standard data formats, when possible.
- Provide a general-purpose framework for physics software development.
- Develop a suite of reconstruction and analysis algorithms and sample codes.
- Simulate physics processes with full detector designs and full backgrounds.

Full Simulation History

- Provide static binary to run full detector simulations using runtime xml detector descriptions.
  - in-house lcdparm xml format (1998)
  - collaboration with R. Chytracek on GDML (2000)
- GISMO (C++ GEISHA + EGS, lcdparm) 1998
- LCDRoot (Geant4 + Root, lcdparm) 1999
- LCDG4 (Geant4 + sio, lcdparm) 2002
- LCS (Geant4 + lcio, lcdparm) 2004
- slic (Geant4 + lcio, GDML) 2005

Detector Design (GEANT 4)

- Need to be able to flexibly, but believably simulate the detector response for various designs.
- GEANT is the de facto standard for HEP physics simulations.
- Use runtime configurable detector geometries
- Write out “generic” hits to digitize later.
- Beam backgrounds and time structure at accelerators will require detailed full detector simulations involving correct handling of event overlays.

Full Detector Response Simulation

- Use Geant4 toolkit to describe interaction of particles with matter and fields.
- Interface layer of non-G4 C++ provides access to:
  - Event Generator particle input
  - Detector Geometry description input
  - Detector Hits output
- Geometries fully described at run-time!
  - In principle, as fully detailed as desired.
  - Uses lcdd, an extension of GDML.
- Solution is applicable beyond LC problem domain.

Geometry Definition

- Goal was to free the end user from having to write any C++ code or be expert in Geant4 to define the detector.
- All of the detector properties should be definable at runtime with an easy-to-use format.
- Selected xml, and extended the existing GDML format for pure geometry description.

LCDD and GDML

- Adopted GDML as base geometry definition, then extended it to incorporate missing detector elements.

LCDD

- detector info
- identifiers
- sensitive detectors
- regions
- physics limits & cuts
- visualization
- magnetic fields

GDML

- expressions (CLHEP)
- materials
- solids
- volume definitions
- geometry hierarchy

LCDD Structure

<lcdd>	LCDD Root Element
<header>	Information about the Detector
<iddict>	Identifier Specifications
<sensitive_detectors>	Detector Readouts
<limits>	Physics Limits
<regions>	Regions (sets of volumes)
<display>	Visualization Attributes
<gdml>	GDML Root Element
<define>	Constants, Positions, Rotations
<materials>	Material Definitions
<solids>	Solid Definitions
<structure>	Volume Hierarchy
</gdml>	
<fields>	Magnetic Field
</lcdd>	

lcdd Features

- **Regions**: production cuts
- **Physics limi**ts: track length, step length, etc.
- **Visualization**: color, level of detail, wireframe/solid
- **Sensitive detectors**
  - calorimeter, optical calorimeter, tracker
  - segmentation
- **IDs**
  - volume identifiers (physical volume id)
- **Magnetic field**s
  - dipole, solenoid, field map
- utilities
  - information on Geant4 stores
  - GDML load/dump

“Compact” Description

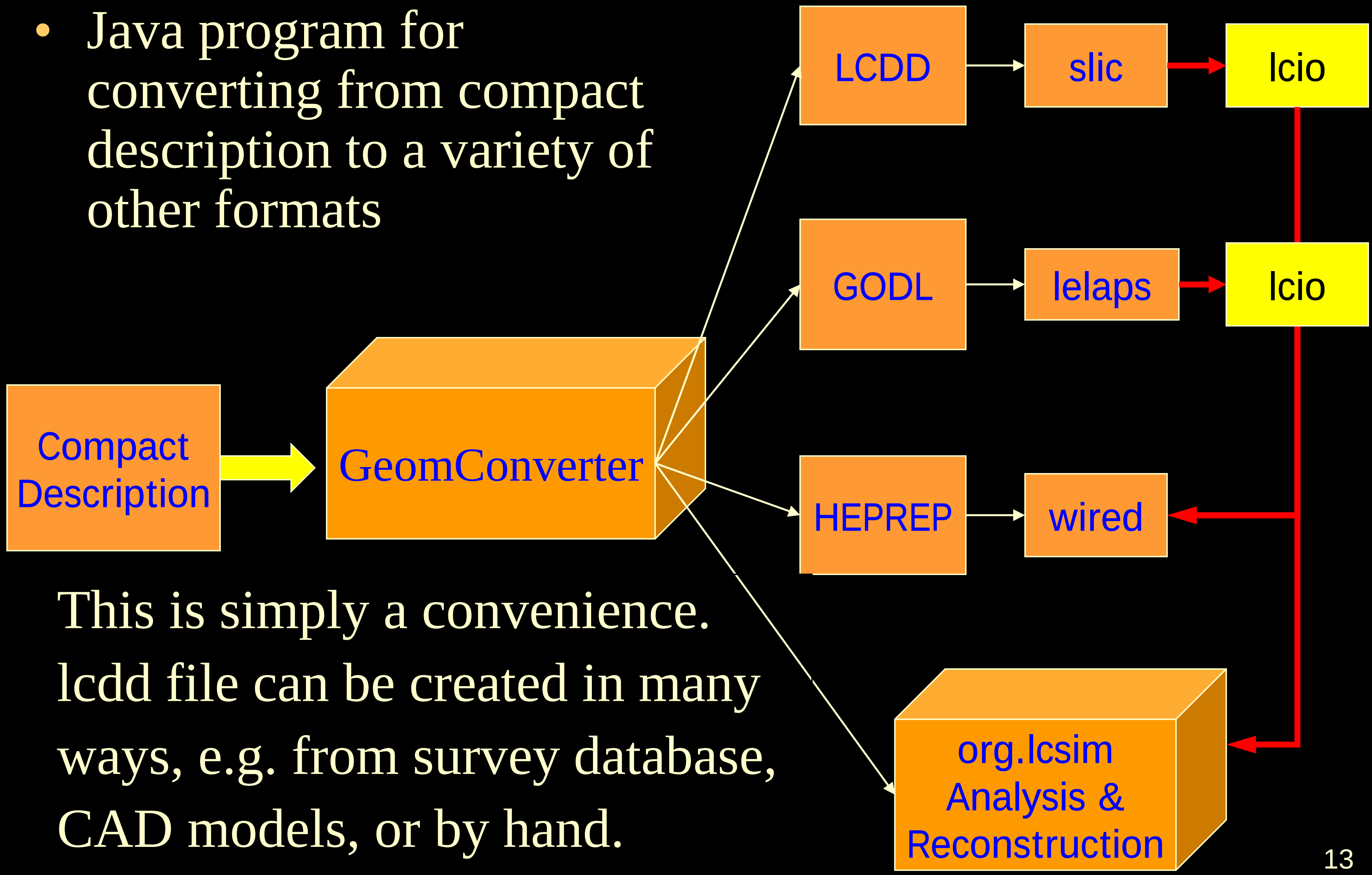
- The lcdd file is very descriptive, but therefore also very verbose.
- Can be written by hand, but prone to human error.
  – Also, just specific to the simulation and not easily accessible to reconstruction and visualization.
- Developed a “compact” detector description which encapsulates the basic properties of a detector and which is further processed by code to produce the input specific to different clients.

Compact Detector Description

- A number of generally useful detector types (at least for HEP collider detectors) have been developed, such as:
  - Sampling calorimeters
  - TPCs
  - Silicon trackers (microstrip as well as pixel)
  - Generic geometrical support structures
- Can also incorporate GDML snippets
  - Allows inclusion of more complicated volumes derived for instance from engineering (CAD) drawings.

GeomConverter

- Java program for converting from compact description to a variety of other formats



This is simply a convenience. lcdd file can be created in many ways, e.g. from survey database, CAD models, or by hand.

Compact Description - Example

```
<detector
    id="3"
    name="HADBarrel"
    type="CylindricalBarrelCalorimeter"
    readout="HcalBarrHits"
    vis="HADVis">
  <dimensions inner_r = "141.0*cm" outer_z = "294*cm" />
  <layer repeat="40">
    <slice material="Steel235" thickness="2.0*cm"/>
    <slice material="RPCGasDefault" thickness="0.12*cm"
      sensitive="yes" region="RPCGasRegion"/>
  </layer>
</detector>
```

global unique identifier

global unique name

detector type

readout collection

visualization settings

layering

absorber

sensitive layer

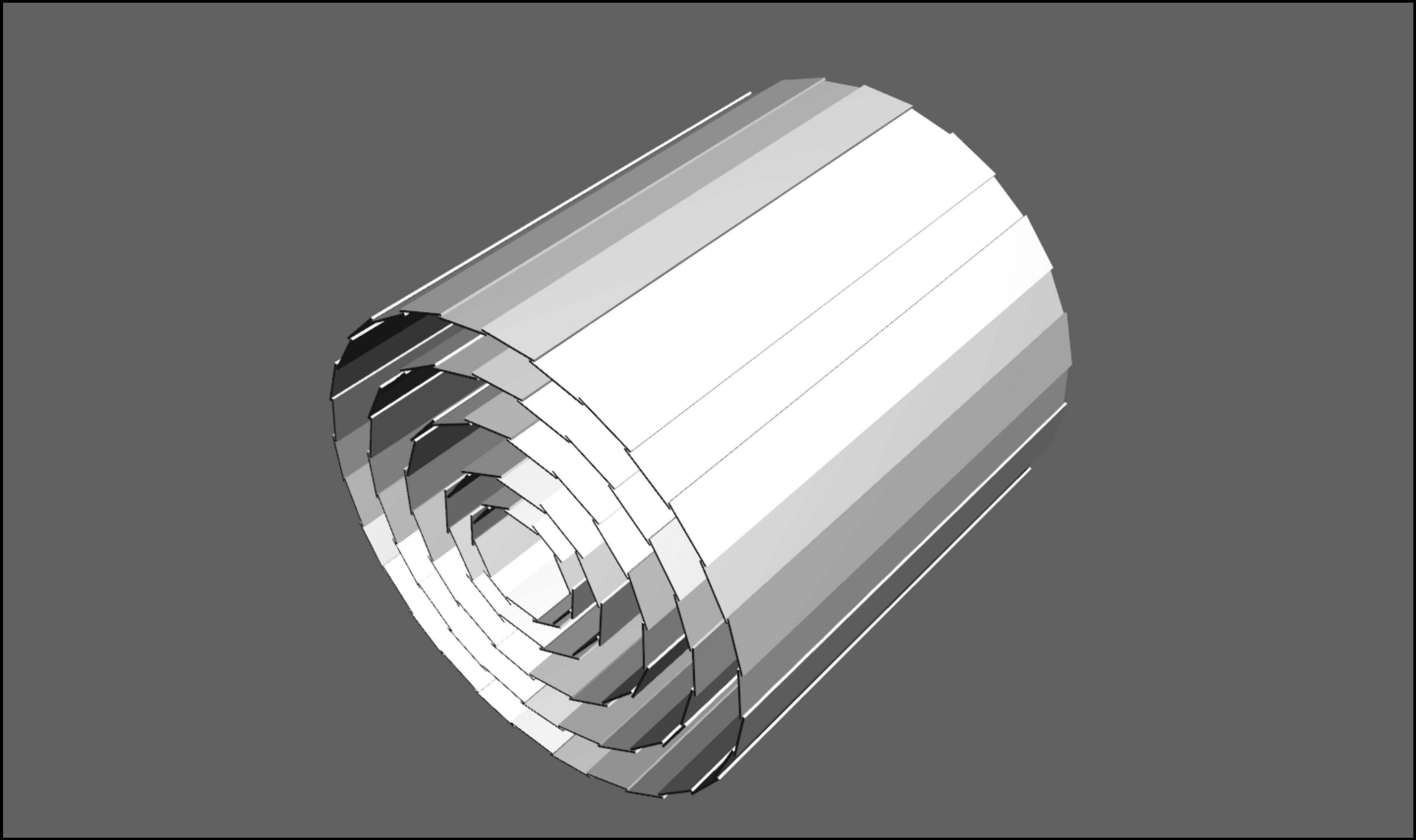
xml: Defining a Tracker Module

```
<module name="VtxBarrelModuleInner">
  <module_envelope width="9.8" length="63.0 * 2" thickness="0.6"/>
  <module_component width="7.6" length="125.0" thickness="0.26"
                    material="CarbonFiber" sensitive="false">
                    <position z="-0.08"/>
  </module_component>
  <module_component width="7.6" length="125.0" thickness="0.05"
                   material="Epoxy" sensitive="false">
                   <position z="0.075"/>
  </module_component>
  <module_component width="9.6" length="125.0" thickness="0.1"
                    material="Silicon" sensitive="true">
                    <position z="0.150"/>
  </module_component>
</module>
```

xml: Placing the modules

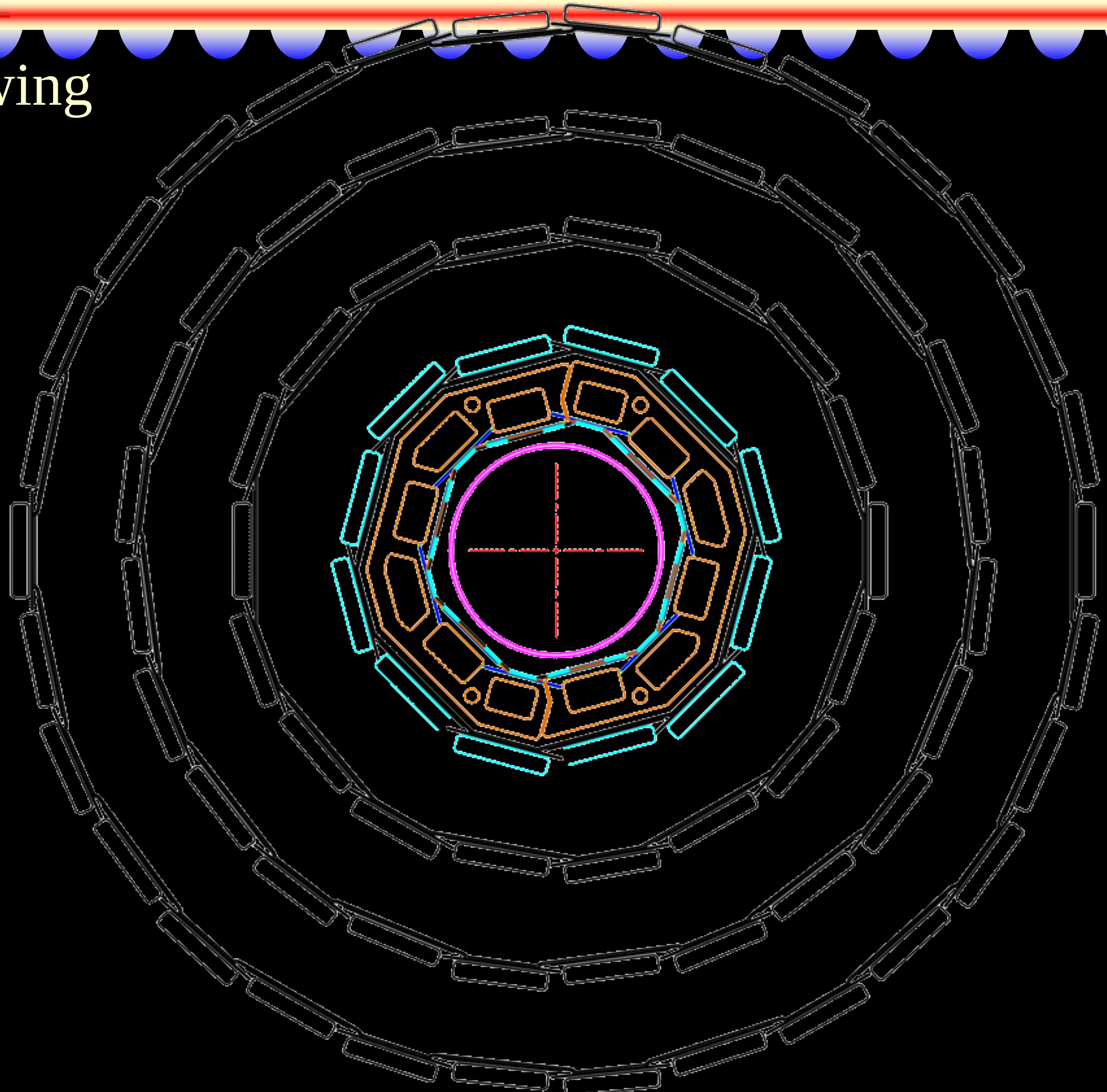
```xml
<layer module="VtxBarrelModuleInner" id="1">
        <barrel_envelope inner_r="13.0" outer_r="17.0" z_length="63 * 2"/>
        <rphi_layout phi_tilt="0.0" nphi="12" phi0="0.2618" rc="15.05" dr="-1.15"/>
        <z_layout dr="0.0" z0="0.0" nz="1"/>
</layer>
<layer module="VtxBarrelModuleOuter" id="2">
        <barrel_envelope inner_r="21.0" outer_r="25.0" z_length="63 * 2"/>
        <rphi_layout phi_tilt="0.0" nphi="12" phi0="0.2618" rc="23.03" dr="-1.13"/>
        <z_layout dr="0.0" z0="0.0" nz="1"/>
</layer>
<layer module="VtxBarrelModuleOuter" id="3">
        <barrel_envelope inner_r="34.0" outer_r="38.0" z_length="63 * 2"/>
        <rphi_layout phi_tilt="0.0" nphi="18" phi0="0.0" rc="35.79" dr="-0.89"/>
        <z_layout dr="0.0" z0="0.0" nz="1"/>
</layer>
<layer module="VtxBarrelModuleOuter" id="4">
        <barrel_envelope inner_r="46.6" outer_r="50.6" z_length="63 * 2"/>
        <rphi_layout phi_tilt="0.0" nphi="24" phi0="0.1309" rc="47.5" dr="0.81"/>
        <z_layout dr="0.0" z0="0.0" nz="1"/>
</layer>
<layer module="VtxBarrelModuleOuter" id="5">
        <barrel_envelope inner_r="59.0" outer_r="63.0" z_length="63 * 2"/>
        <rphi_layout phi_tilt="0.0" nphi="30" phi0="0.0" rc="59.9" dr="0.77"/>
        <z_layout dr="0.0" z0="0.0" nz="1"/>
</layer>
```

A Barrel Vertex Detector



Example Vertex Detector

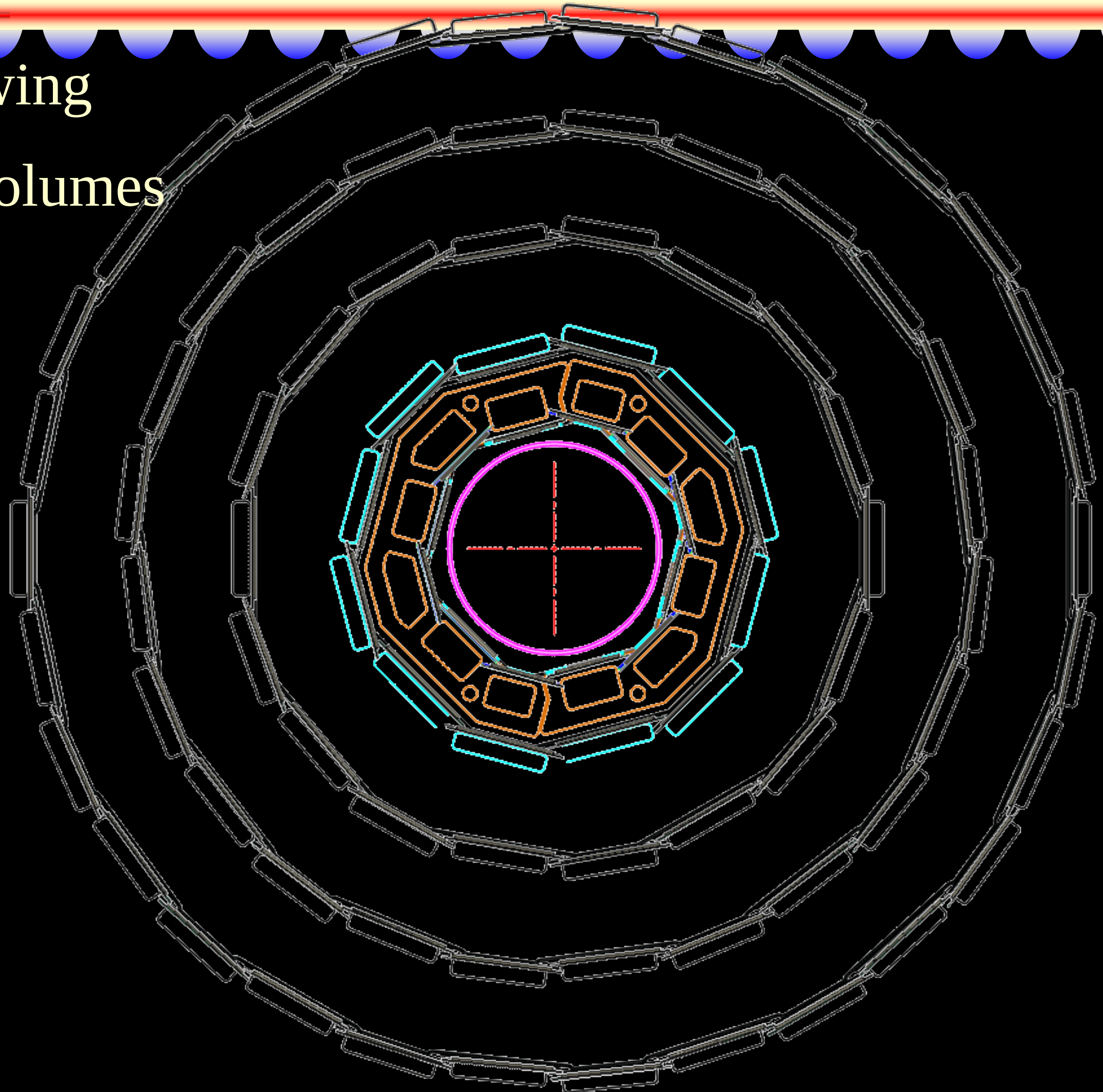
CAD Drawing



Example Vertex Detector

CAD Drawing

GEANT Volumes

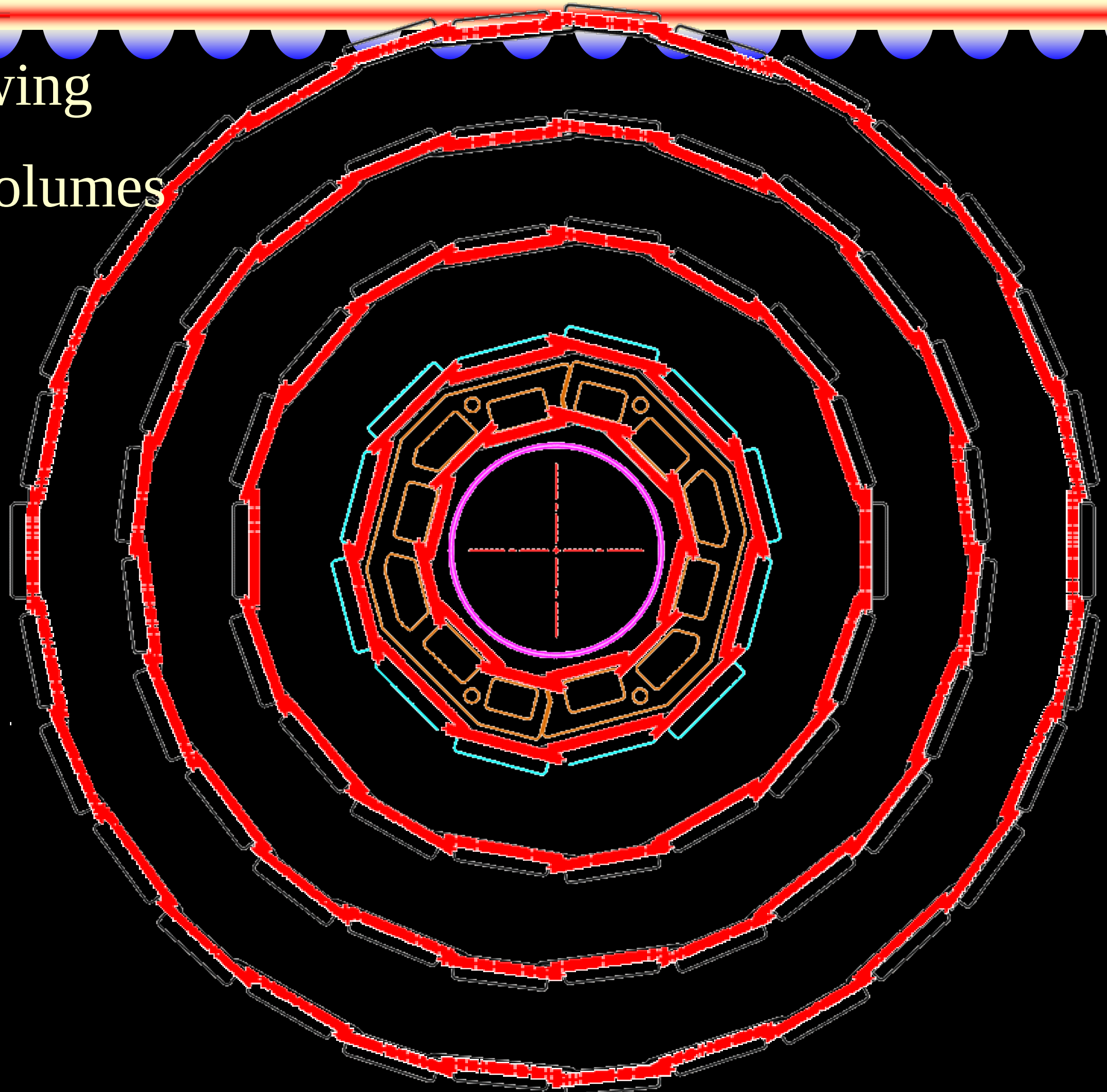


Example Vertex Detector

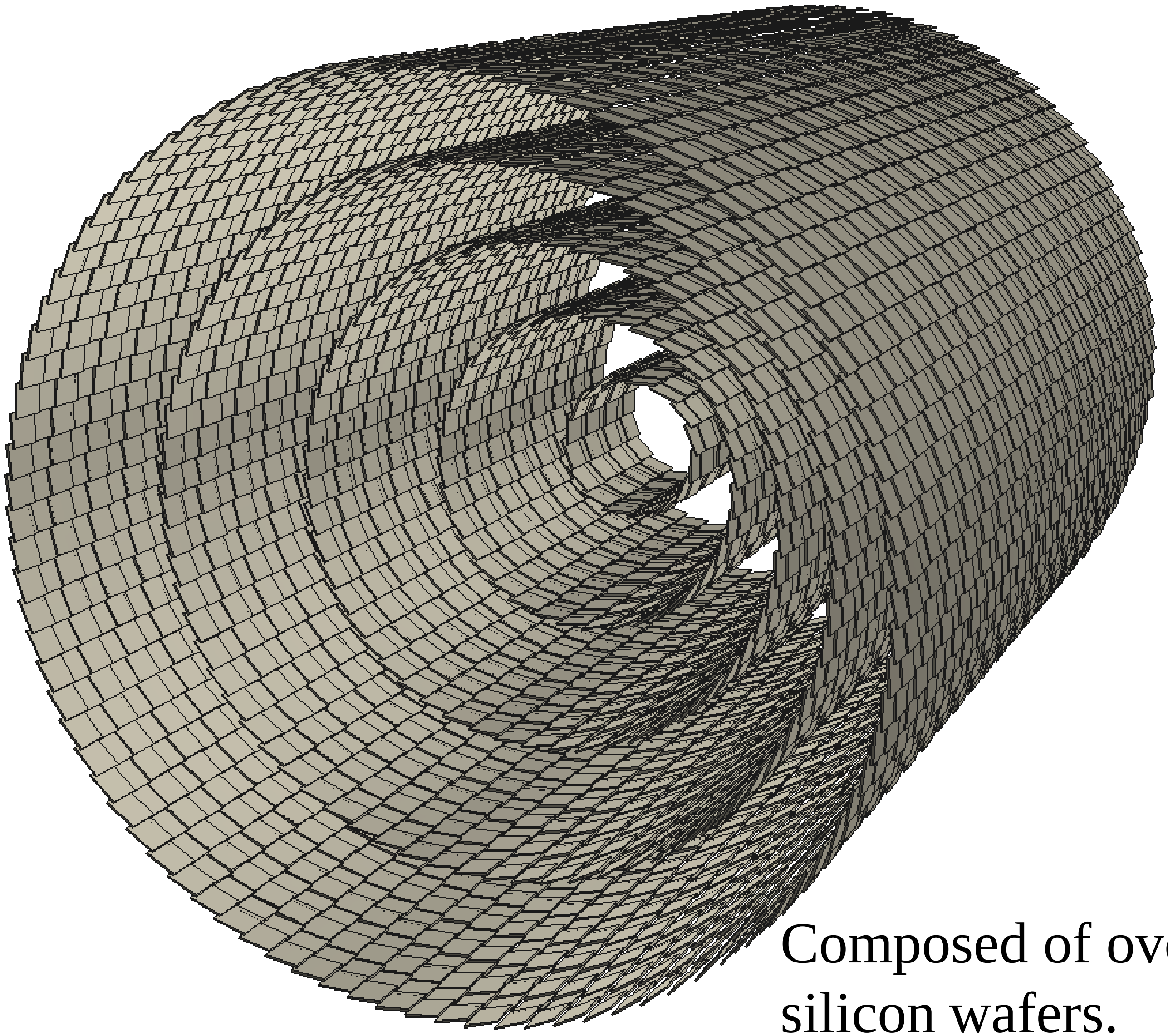
CAD Drawing

GEANT Volumes

LCIO Hits

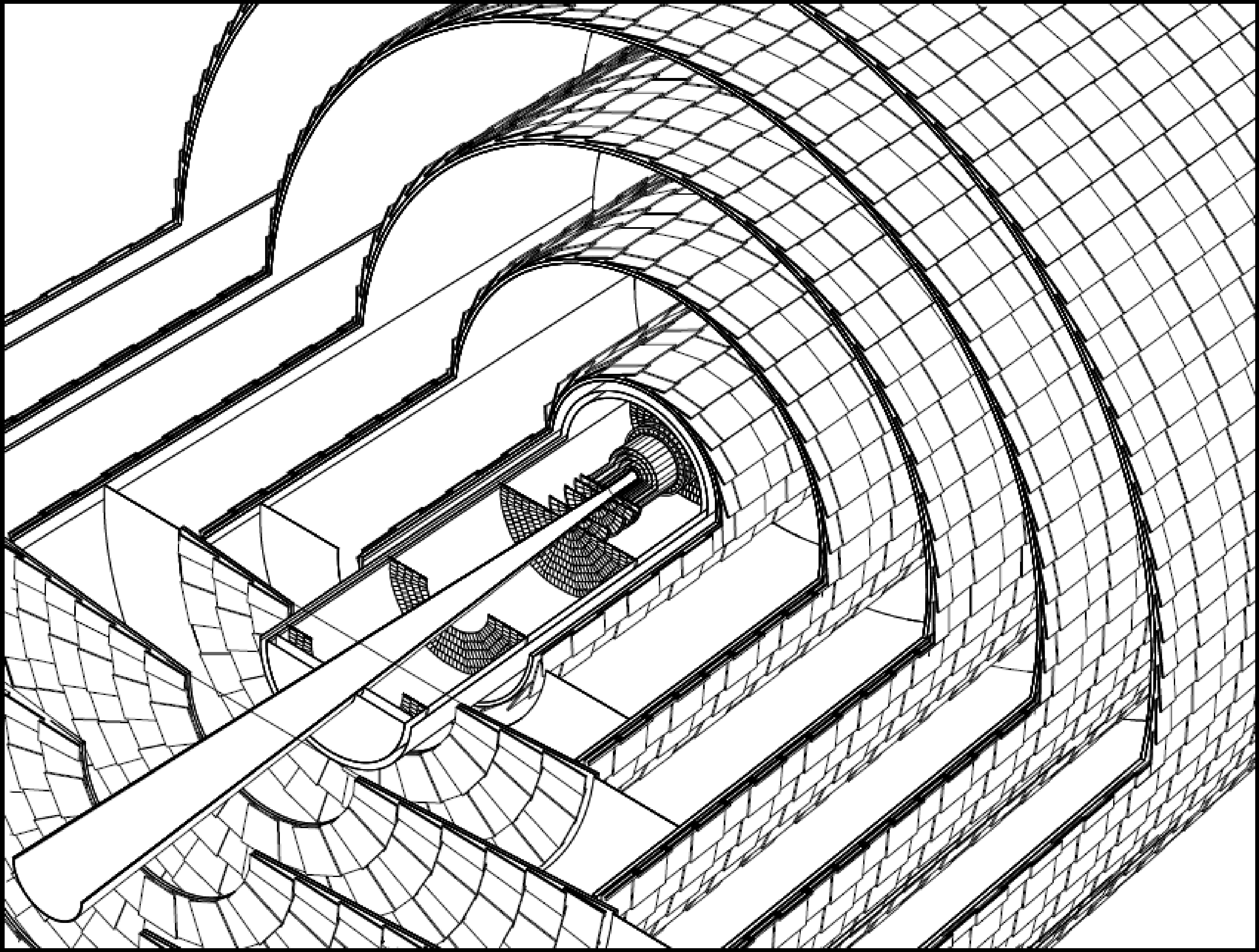


Barrel Outer Tracker



Composed of overlapping silicon wafers.

Complete Silicon Tracker



Generic Hits Problem Statement

- We wish to define a generic output hit format for full simulations of the response of detector elements to physics events.
- Want to preserve the “true” Monte Carlo track information for later comparisons.
- Want to defer digitization as much as possible to allow various resolutions, readout technologies, etc. to be efficiently studied.

Types of Hits

- "Tracker" Hits
  - Position sensitive.
  - Particle unperturbed by measurement.
  - Save "ideal" hit information.
- "Calorimeter" Hits
  - Energy sensitive.
  - Enormous number of particles in shower precludes saving of each "ideal" hit.
  - Quantization necessary at simulation level.

Tracker Hit

- **MC Track handle**
- Encoded **detector ID** (detector dependent)
- **Hit position** in sensitive volume
- **Track momentum** at hit position.
- **Energy deposited** in sensitive volume.
- **Time** of track's crossing.
- **Path length** in sensitive volume.

Sufficient information to do hit digitization.

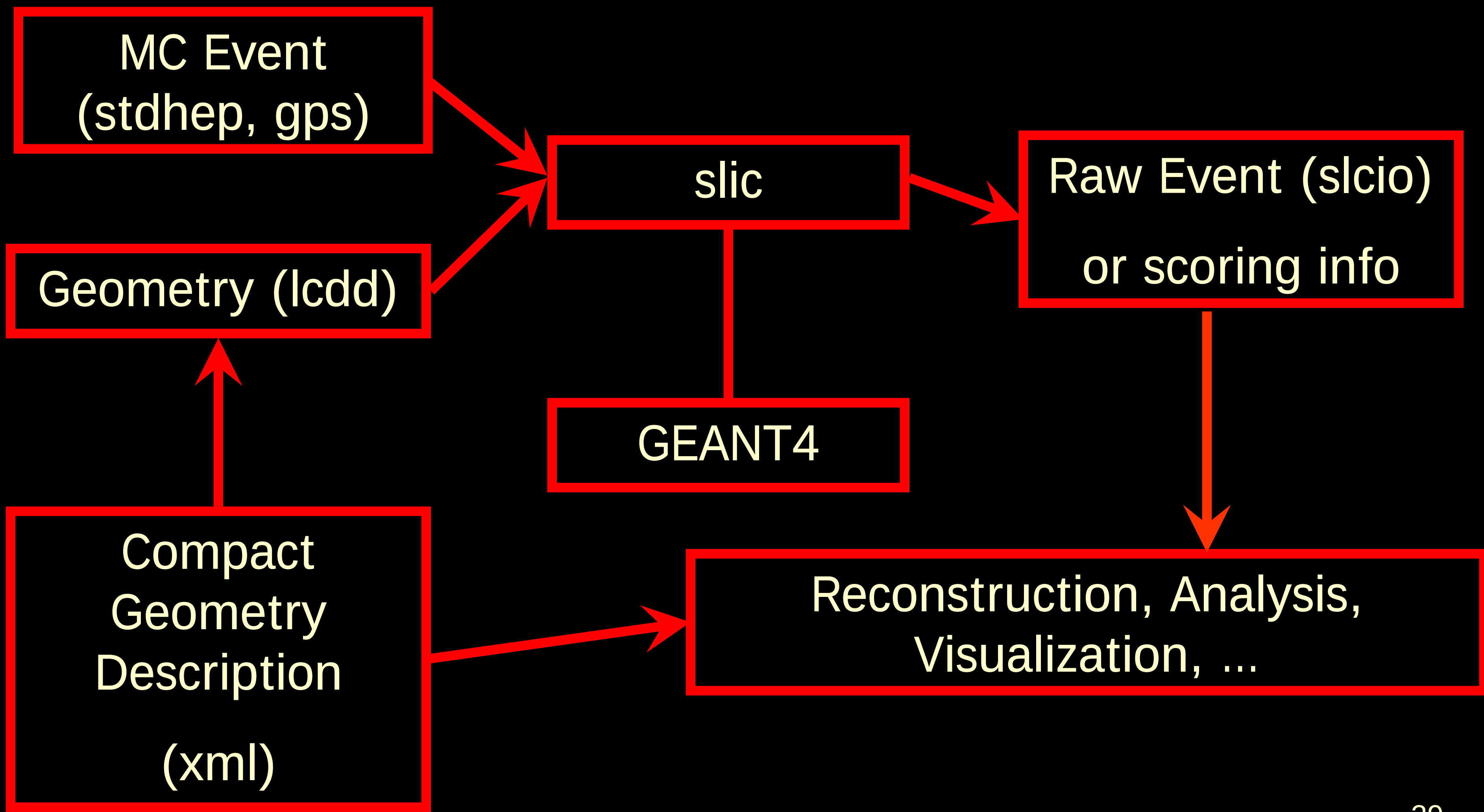
Calorimeter Hit

- Encoded detector ID (detector dependent)
- MC IDs for tracks contributing to this cell.
- Energy deposited.
- Time of energy deposition.
- Repeated for each energy contribution.
- Support recently added for optical calorimeters
  - Can store Cerenkov and scintillation light.

slic: The Executable

- Provide static executable on Linux, Windows, Mac.
- Commandline or G4 macro control.
- Only dependence is local detector description file.
  - Trivial grid/cloud usage (no database call-backs, etc.)
- Event input via stdhep, particle gun, …
- Detector input via GDML, lcdd
- Response output via LCIO using generic hits or Geant4 scoring via macros.

Detector Full Simulation



Detector Variants

- Runtime XML format allows variations in detector geometries to be easily set up and studied:
  - Sampling calorimeters:
    - absorber materials, dimensions
    - Readout technologies, e.g. RPC, scintillator
    - Layering (radii, number, composition)
    - Readout segmentation (size, projective vs. nonprojective)
  - Total absorption crystal calorimeters
    - Optical properties
  - Tracking detector technologies & topologies
    - TPC, silicon microstrip, silicon pixels

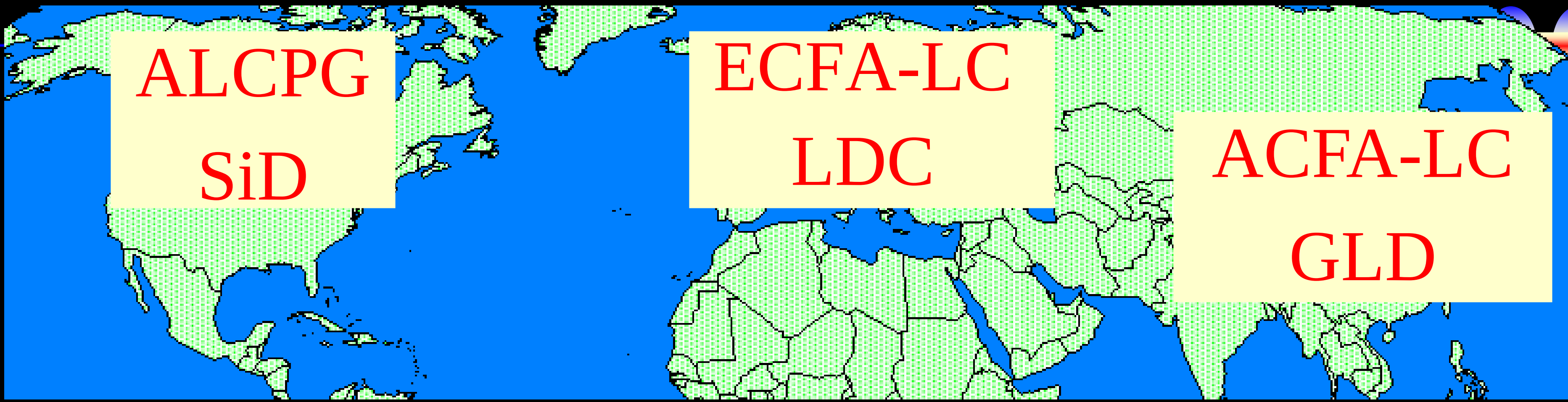
slic & lcdd: Summary

- Provides a complete and flexible detector simulation package capable of simulating arbitrarily complex detectors with runtime detector description.
- Used by ILC, CLIC, Muon Collider &FCC detector community for simultaneous and iterative evolution of different detector concepts and their variations.
- Being used by HPS @ JLab
- Has been applied to CPT simulations.
- Could be used by other communities (astro, medical) for rapid prototyping or full simulation.

slic & lcdd

- Despite their potential for widespread application, slic and lcdd were not universally adopted for ILC simulations.
  - Primarily used by the ILC SiD consortium
- Proved very useful in CLIC detector comparisons
- Real success story from the ILC physics and detector community is interoperability
  - Simple and open Event Data Model
  - Simple and open Data Persistency
  - Simple and open Detector Description
    - not just geometry!

LCIO

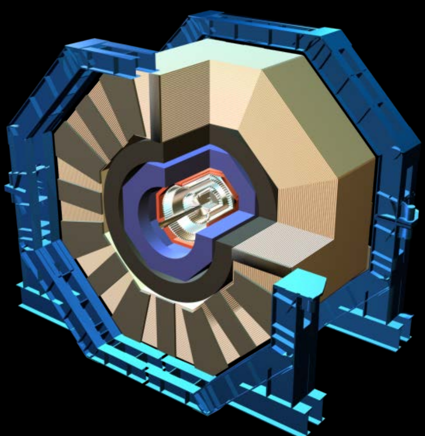


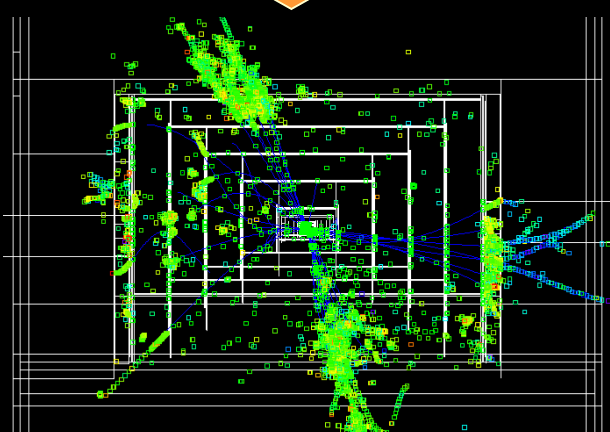
slic	MOKKA	JUPITER
org.lcsim	MarlinReco	Satellites
Java	C++	root

ILC Full Detector Concepts

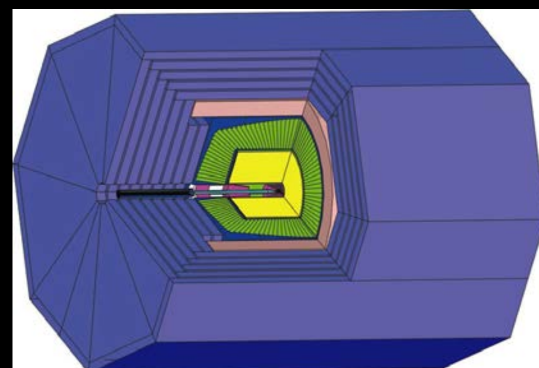
Same input event, different detectors

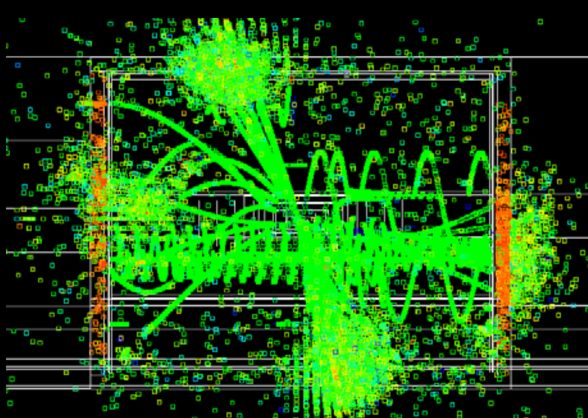
SiD



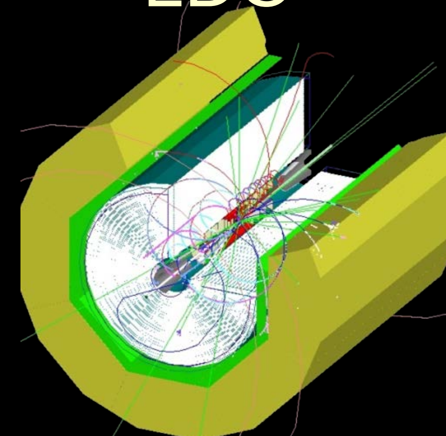


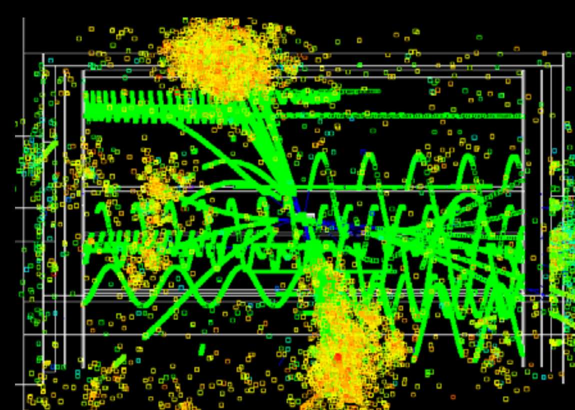
GLD



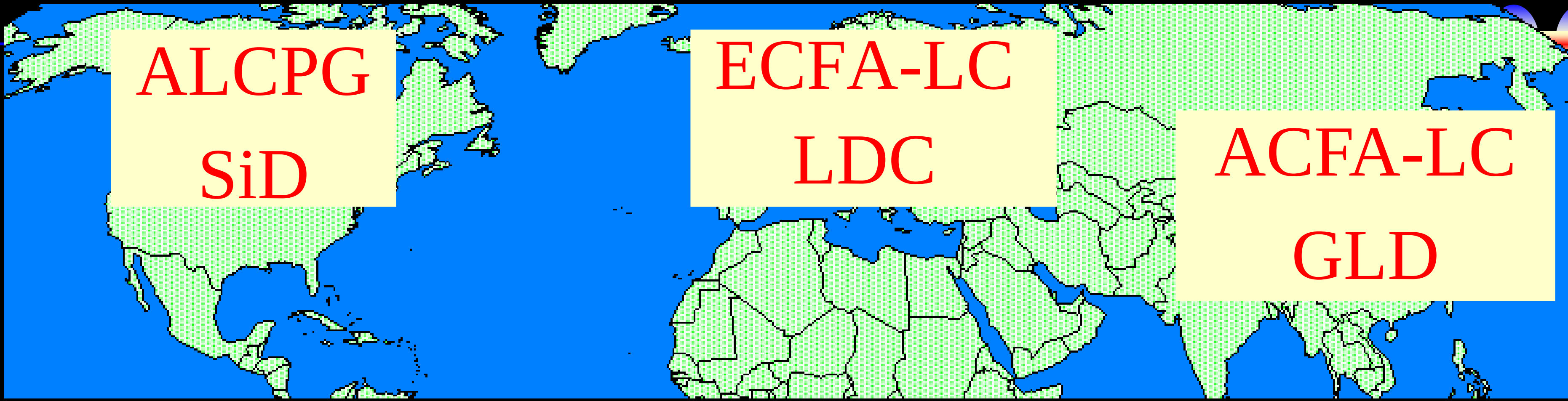


LDC





LCIO



slic	MOKKA	JUPITER
org.lcsim	MarlinReco	Satellites
Java		root

LCIO

Common Data Model

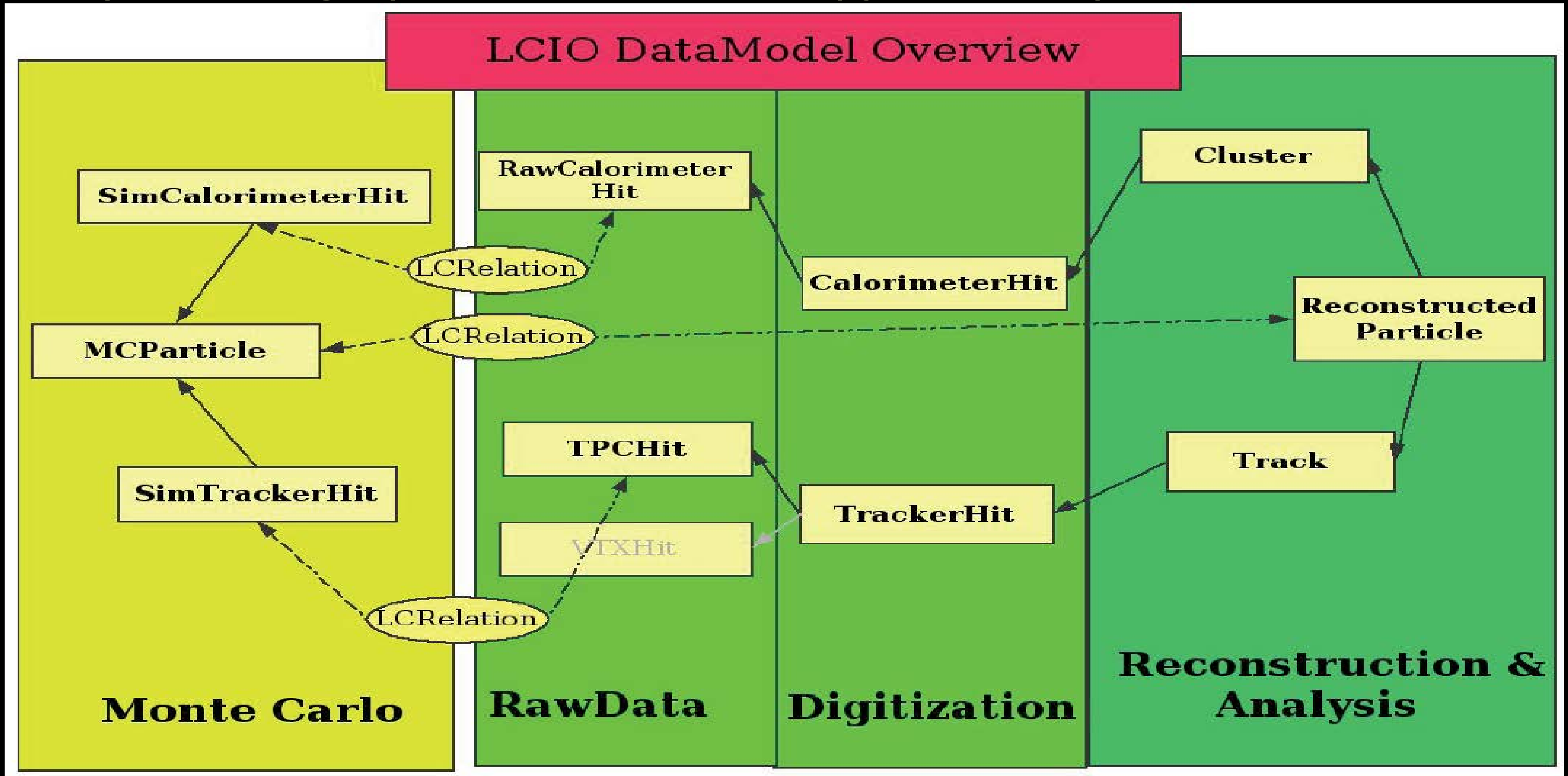
Common IO Format

LCIO

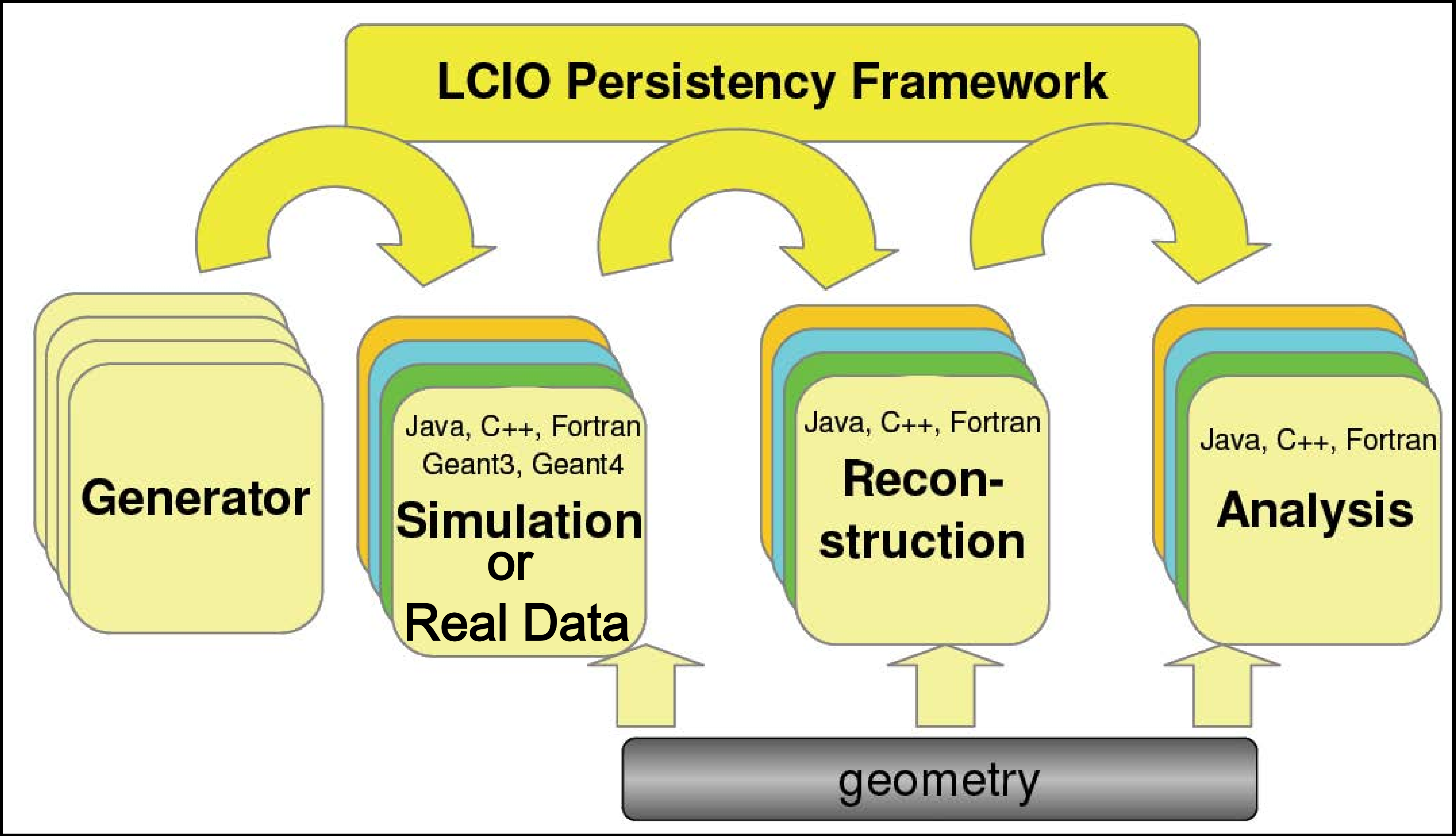
- Persistency framework for LC simulations.
- Currently uses SIO: Simple Input Output
  - on-the-fly data compression
  - random access
  - C++, Java, python (and FORTRAN!) implementations available
- Changes in IO engine designed for (e.g. root, hdf5)
- Extensible event data model
  - Generic Tracker and Calorimeter Hits.
  - Monte Carlo particle hierarchy.

LCIO Overview

- Event Data Model and persistency format
  - MC simulation
  - Data (experimental or testbeam)
  - Reconstructed Objects
- Multiple bindings (C++, Java, Fortran, python, root)

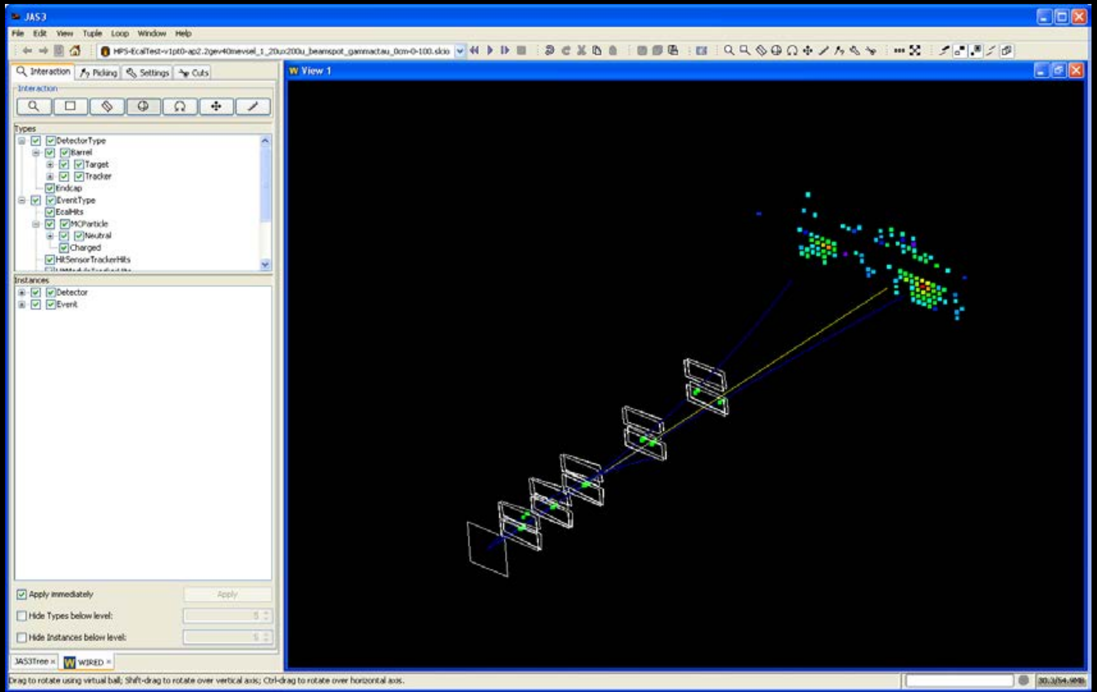


LCIO Overview



LCIO Event Display

- Fully integrated within JAS using Wired.
- Fully interactive event display
- Detector & Event objects selectable, pickable, queryable, can have cuts applied, etc.
  - Not just a static image.



Raw Data

- LCIO has been used for many years by various testbeam campaigns and experiments, both tracking and calorimetry.
  - EDM supports raw data taking and analysis.
  - Simple, robust & fast
- Many tools exist for data monitoring, QA, analysis, etc.
- Allows ~seamless integration of MC, testbeam and experimental data.

Summary

- It's very difficult to ~~herd cats~~ keep physicists from reinventing the wheel and writing new software packages.
  - Has both advantages and disadvantages
- It is more important to be able to exchange detector designs and data.

KEEP IT SIMPLE!

FACILITATE INTEROPERABILITY!

- Get the event data model right and keep it open.
- Pick a detector definition which is exchangeable.
  - More than just geometry!