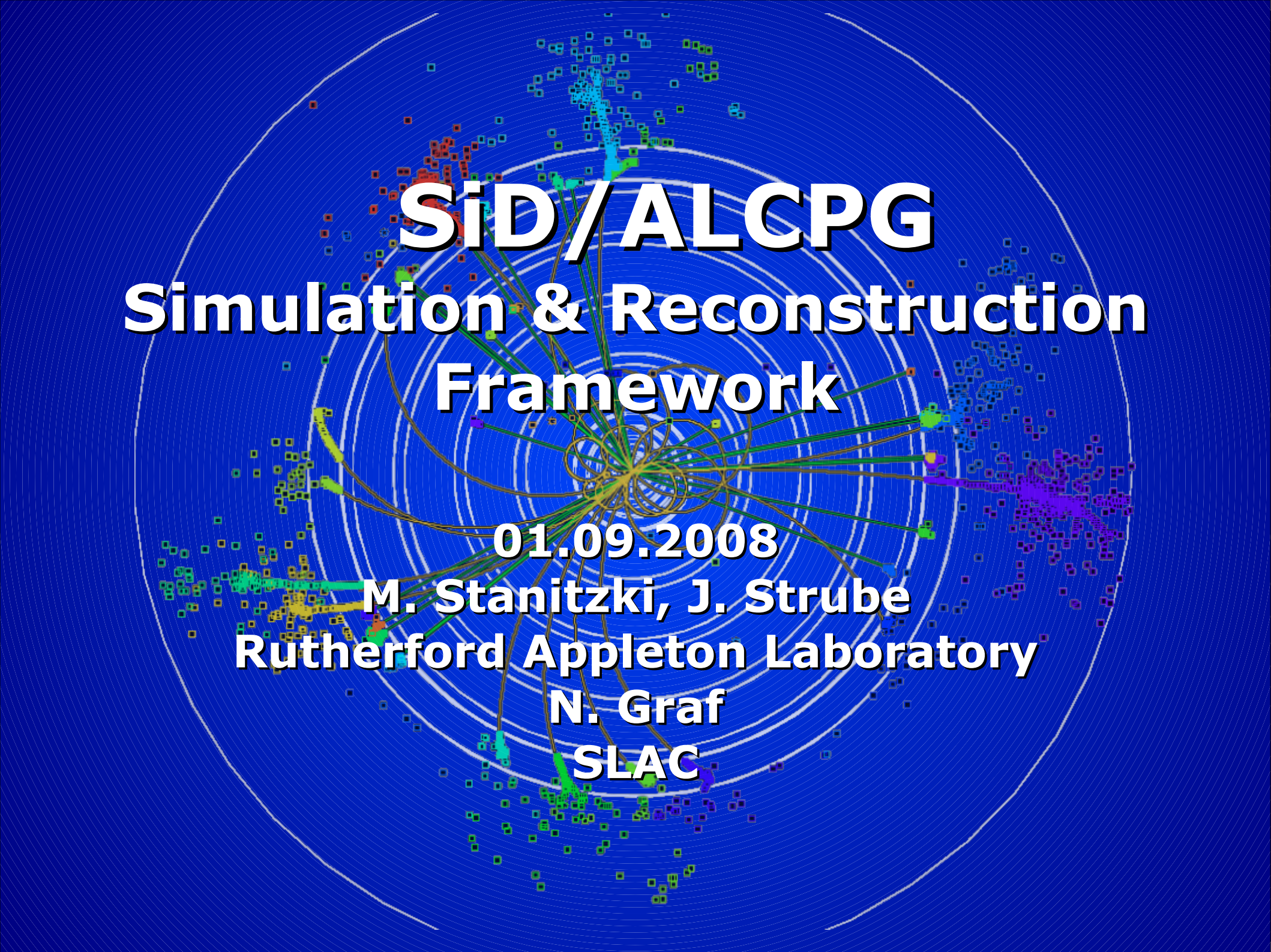


SiD/ALCPG
Simulation & Reconstruction Framework

01.09.2008

M. Stanitzki, J. Strube
Rutherford Appleton Laboratory
N. Graf
SLAC

Outline

- Overview
- SLIC The Detector Simulation
- org.lcsim The Reconstruction
  - Fast MC
  - Realistic Reconstruction
- Detector Optimization
- Summary

Mission statement

- Physics analyses
  - Refine and strengthen the arguments for the ILC.
- Detector design
  - Integrated system design for an optimal detector.
- Event reconstruction
  - Demonstrate that proposed detector systems can conduct the physics program, and allows cost-benefit optimization to be done.
- Testbeam and prototype infrastructure
  - Support subsystem design, readout and analysis.

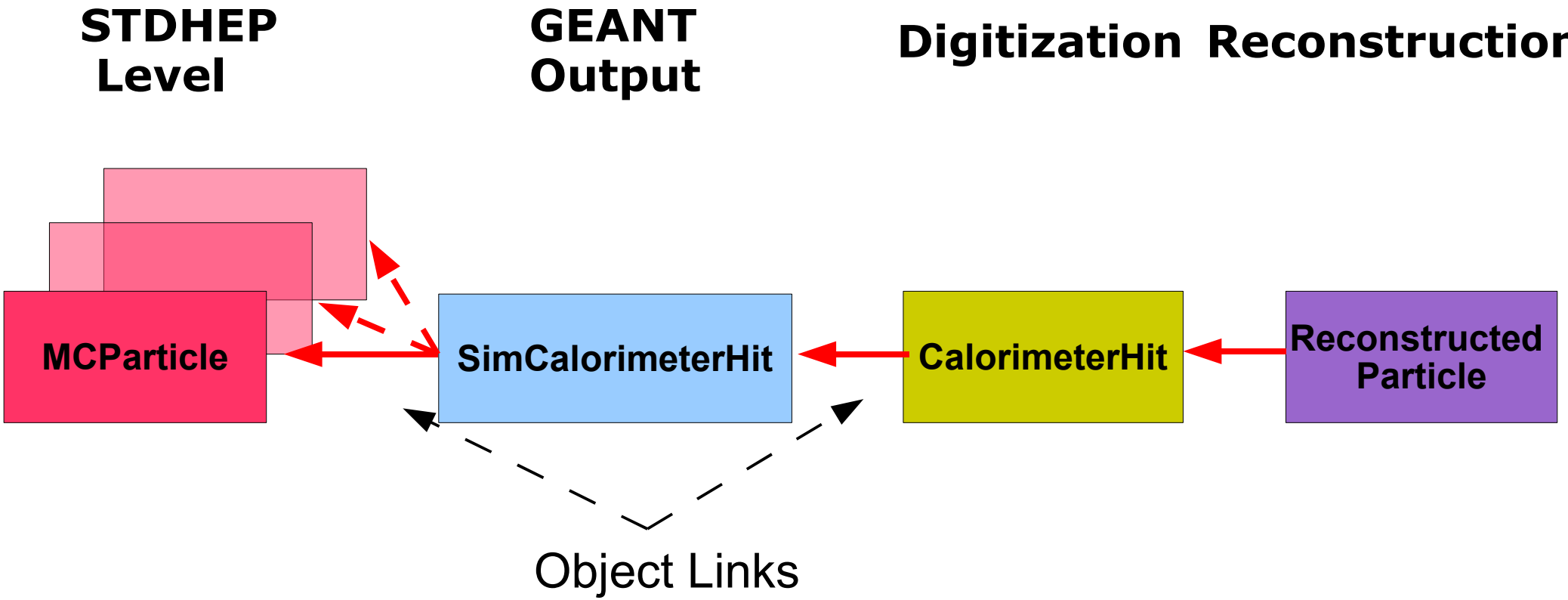
The foundations

- **GEANT4** is the underlying simulation code for all ILC simulation software, version 4.9 is current
  - Well tested by LHC community
  - Well maintained
- **STDHEP** is the standardized input format for MonteCarlo Simulations
- **LCIO** has been chosen as ILC-wide IO format
  - used by SiD and ILD
  - allows data exchange easily

LCIO

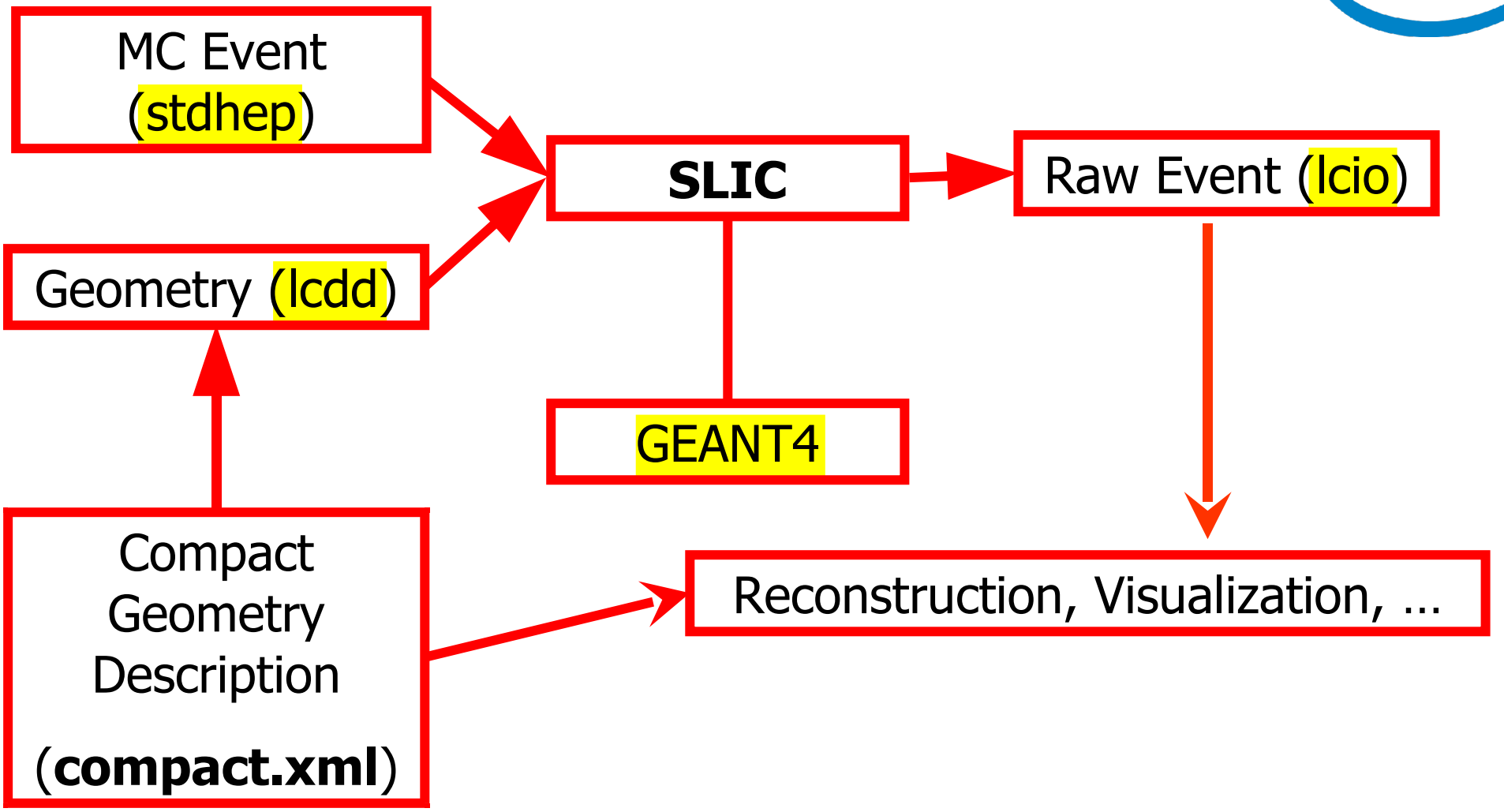
- Persistency Framework with C++/Java (and Fortran) bindings
- Standardized Classes
  - TrackerHit, CalorimeterHit ...
  - MCParticle
  - ReconstructedParticles
  - Generic Objects
  - Relations
  - Links
- More information here :http://lcio.desy.de/

An Example


STDHEP Level
GEANT Output
Digitization
Reconstruction
MCParticle
SimCalorimeterHit
CalorimeterHit
Reconstructed Particle
Object Links


Detector Modeling &
Simulation

The Tool Chain



SLIC

- Build on GEANT 4.9
- Binaries available for Linux/MacOs/Windows
  - or build it from scratch
- Detector description done using LCDD
  - very flexible
- Input is using STDHEP
- Output is done using LCIO
- http://www.lcsim.org/software/slic

Varying detectors

- Runtime XML format allows variations in detector geometries to be easily set up and studied:
  - Stainless Steel vs. Tungsten HCAL sampling material
  - RPC vs. GEM vs. Scintillator vs. dual readout
  - Layering (radii, number, composition)
  - Readout segmentation (size, projective vs. non-projective)
  - Tracking detector technologies & topologies
    - TPC, Pixels, Silicon microstrip, SIT, SET
    - "Wedding Cake" Nested Tracker vs. Barrel + Cap
  - Field strength
  - Far forward MDI variants

The compact.xml

- The detector geometry is specified in this file
- Plain XML, human-readable
- This file is common for simulation and reconstruction
  - One common Geometry description
- Can specify a lot of different geometries
- **GeomConverter** can convert descriptions between different formats

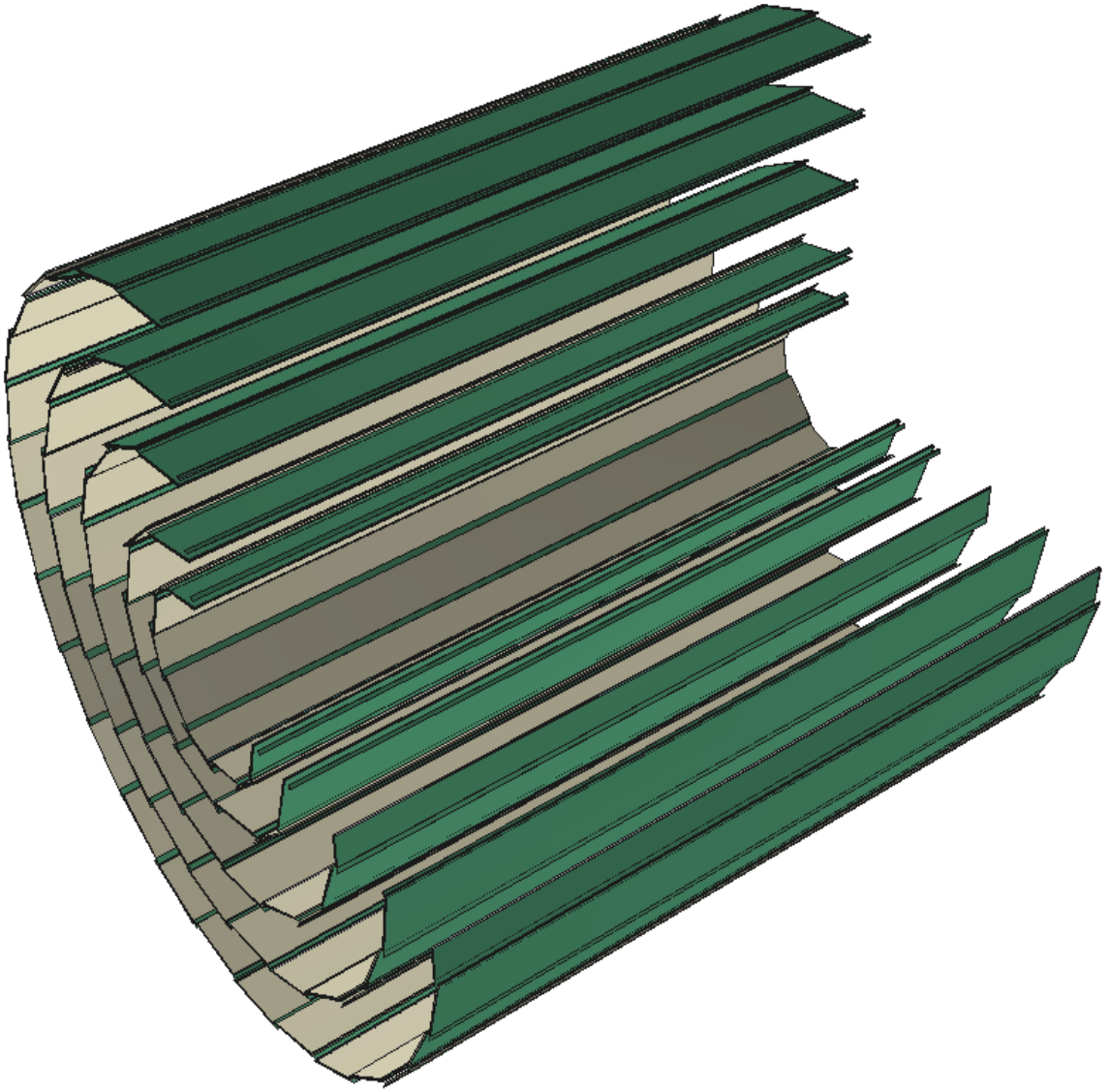
xml: Defining a Module

```
<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

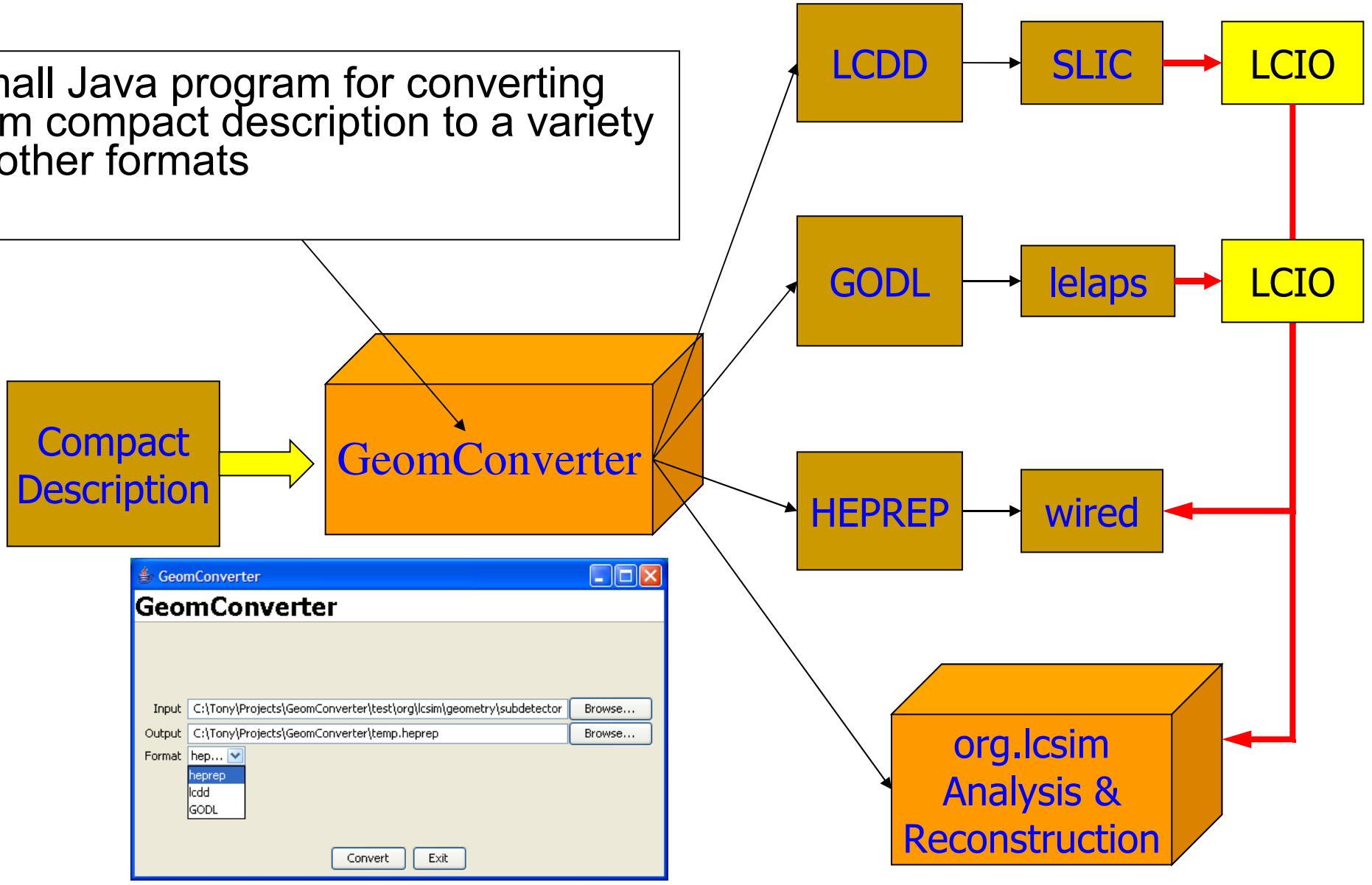
```
<layer module="VtxBarrelModuleInner" id="1">
        <barrel_envelope inner_r="13.0" outer_r="17.0" z_length="63 * 2"/>
        <rphi_layout phi_tilt="0.0" nphi="12" phi0="0.2618" rc="15.05" dr="-1.15"/>
        <z_layout dr="0.0" z0="0.0" nz="1"/>
</layer>
<layer module="VtxBarrelModuleOuter" id="2">
        <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>
```

The Barrel Vertex

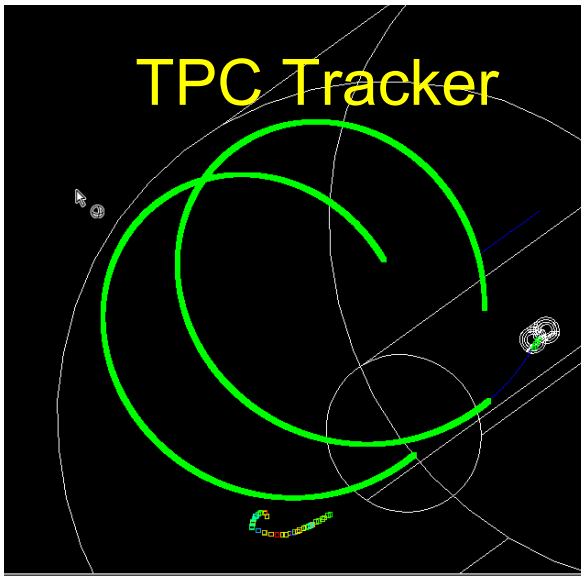


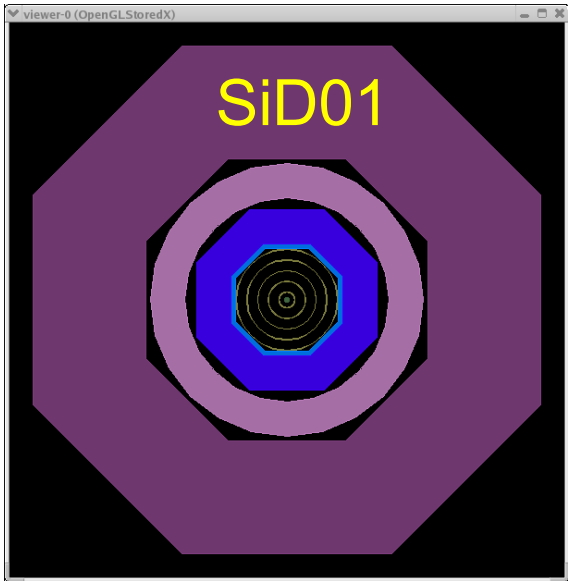
GeomConverter

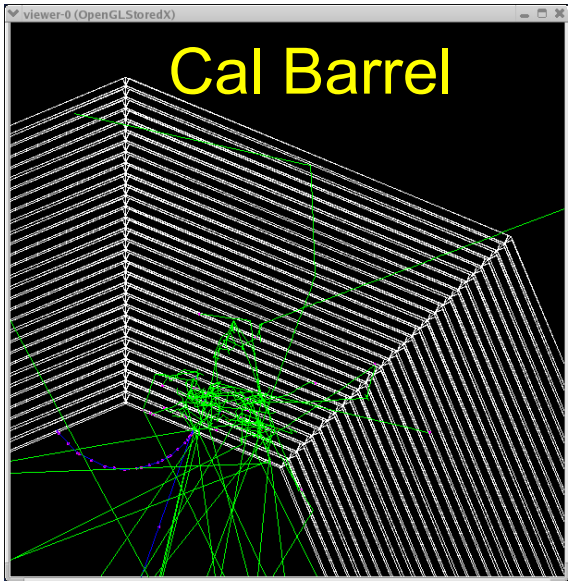
Small Java program for converting from compact description to a variety of other formats

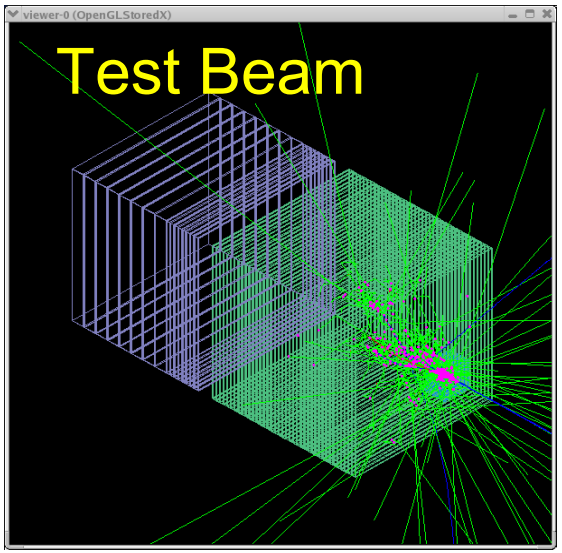


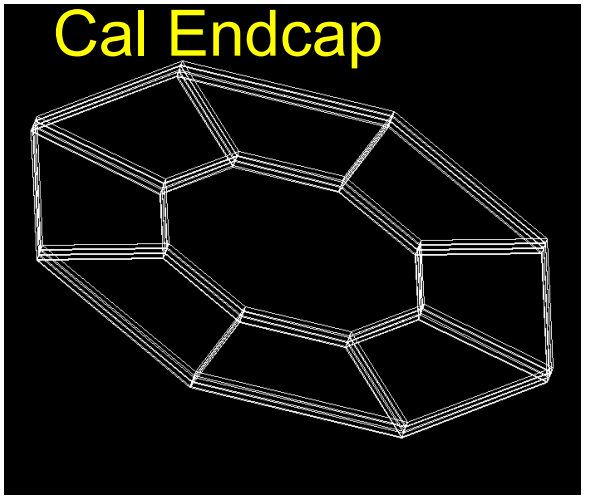
Sample Geometries



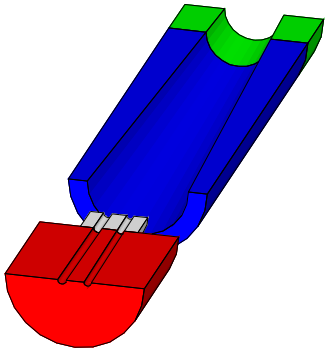








MDI-BDS



Implementing a CLIC detector

```
<lccdd xmlns:compact="http://www.lcsim.org/schemas/compact/1.0"
       xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
       xs:noNamespaceSchemaLocation="http://www.lcsim.org/schemas/compact/1.0/compact.xsd">

  <!-- info tag containing author, version, time, unique id (url) -->
  <info name="clic040708"
title="CLIC detector based on sid01"
author="Marcel Stanitzki Jan Strube Norman Graf"
url="http://confluence.slac.stanford.edu/display/ilc/"
status="development"
version="$Id: compact.xml,v 1. 2006/11/29 02:10:43 partridge Exp $">
    <comment>The compact format for the a CLIC detector based on SID01 040708</comment>
  </info>

<!--
Moved out Vertex to 4 cm ...
increased Beam pipe to 3.9 cm
moved HCAL to 54 layers
-->

  <!-- Constants -->
  <define>
    <constant name="cm" value="10"/>
  </define>

  <materials>
    <material name="TungstenDens23">
      <D value="17.7" unit="g/cm3"/>
      <fraction n="0.925" ref="W"/>
      <fraction n="0.066" ref="Ni"/>
      <fraction n="0.009" ref="Fe"/>
    </material>
   <
  </materials>
```

HEADER Block

Define your own constants and material here

Cont'd

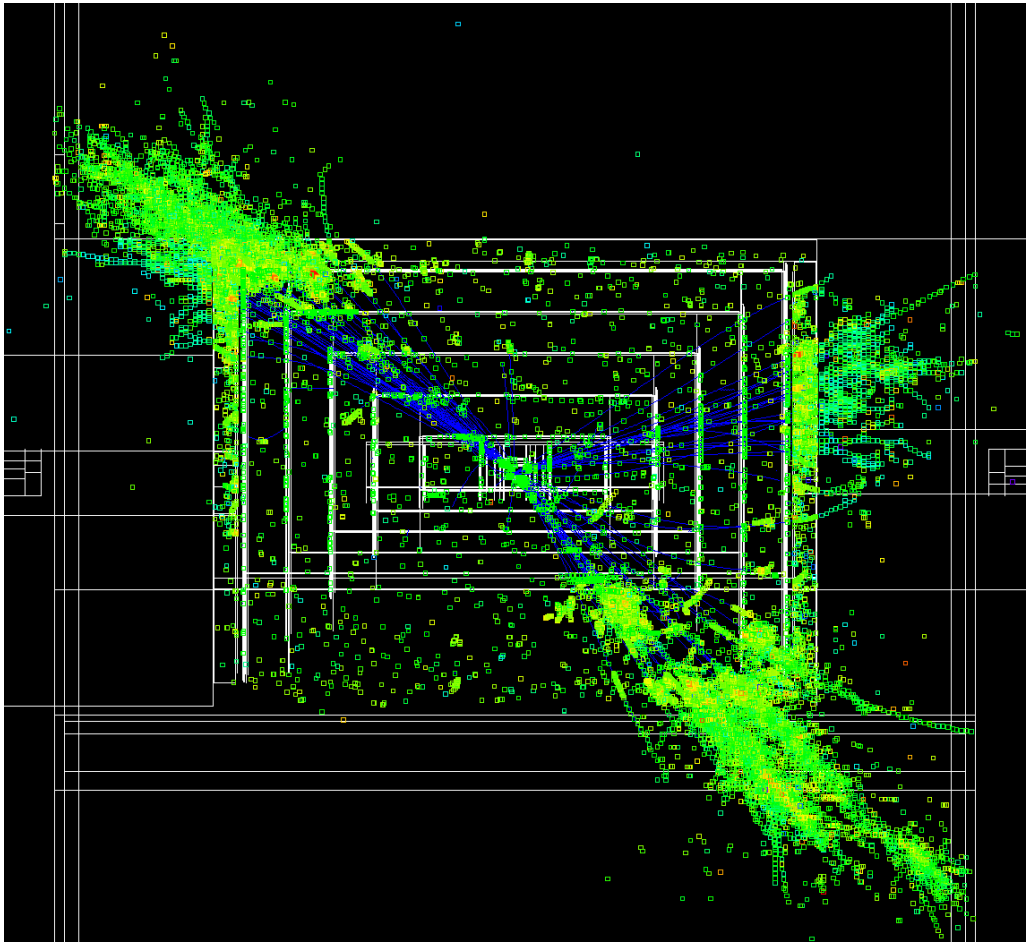
```
<detectors>
    <!-- Electromagnetic calorimeter -->
    <detector id="2" name="EMBarrel" type="CylindricalBarrelCalorimeter" readout="EcalBarrHits">
      <dimensions inner_r = "127.0*cm" outer_z = "182.0*cm" />
      <layer repeat="30">
          <slice material = "TungstenDens24" thickness = "0.25*cm" />
          <slice material = "Silicon" thickness = "0.032*cm" sensitive = "yes" />
          <slice material = "Copper"  thickness = "0.005*cm" />
          <slice material = "Kapton"  thickness = "0.030*cm" />
          <slice material = "Air"     thickness = "0.033*cm" />
      </layer>
    </detector>
  </detectors>

  <!-- Sensitive Detector readout segmentation -->
  <readouts>
    <readout name="EcalBarrHits">
      <segmentation type="NonprojectiveCylinder" gridSizePhi="0.35*cm" gridSizeZ="0.35*cm" />
      <id>system:8,layer:8,barrel:3,phi:32:16,z:-16</id>
    </readout>
  </readouts>

  <fields>
    <field type="Solenoid" name="GlobalSolenoid"
   inner_field="5.0"
   outer_field="-0.6"
   zmax="1000"
   outer_radius="(250.0+ 5.0 + 17.5 + 40./2.)*cm"/>
   <!-- SolenoidCoilBarrel inner_radius + Al support + Air gap + half coil-->
  </fields>
</lccdd>
```

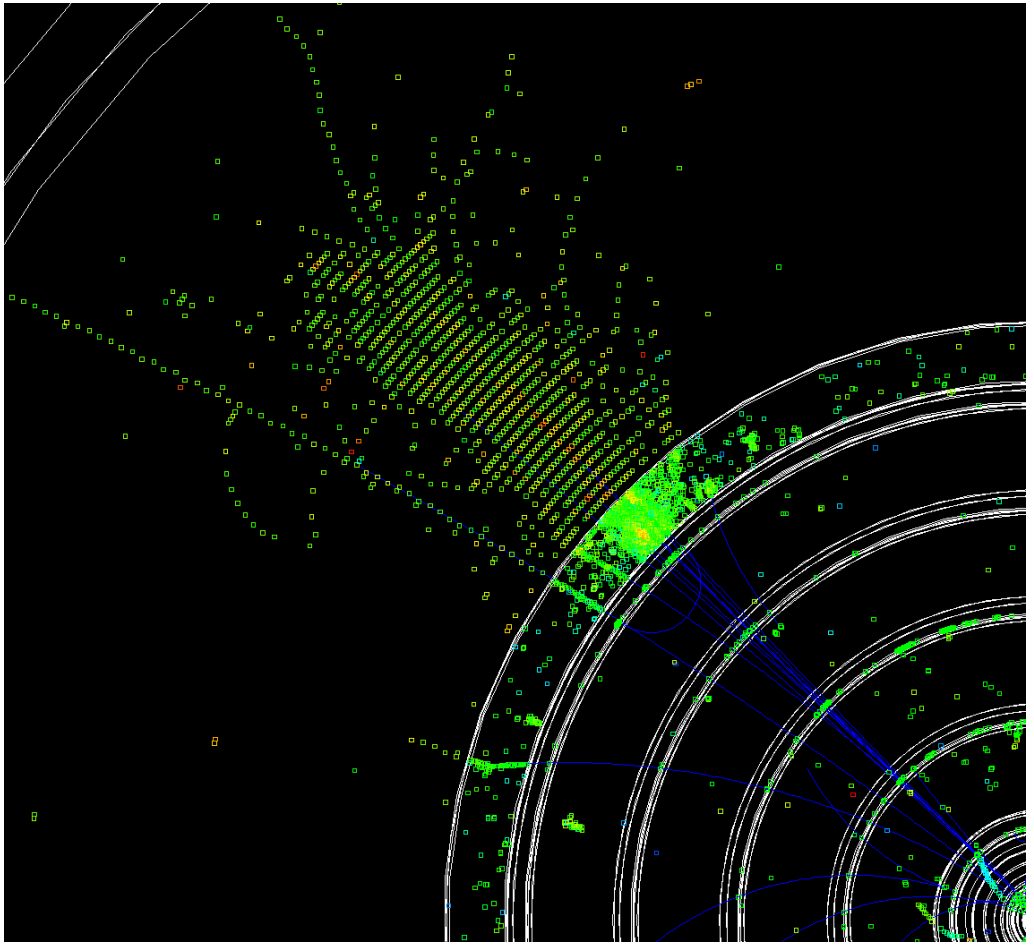
Make your detectors, e.g. a ECAL

Define the Readout

Define your Field

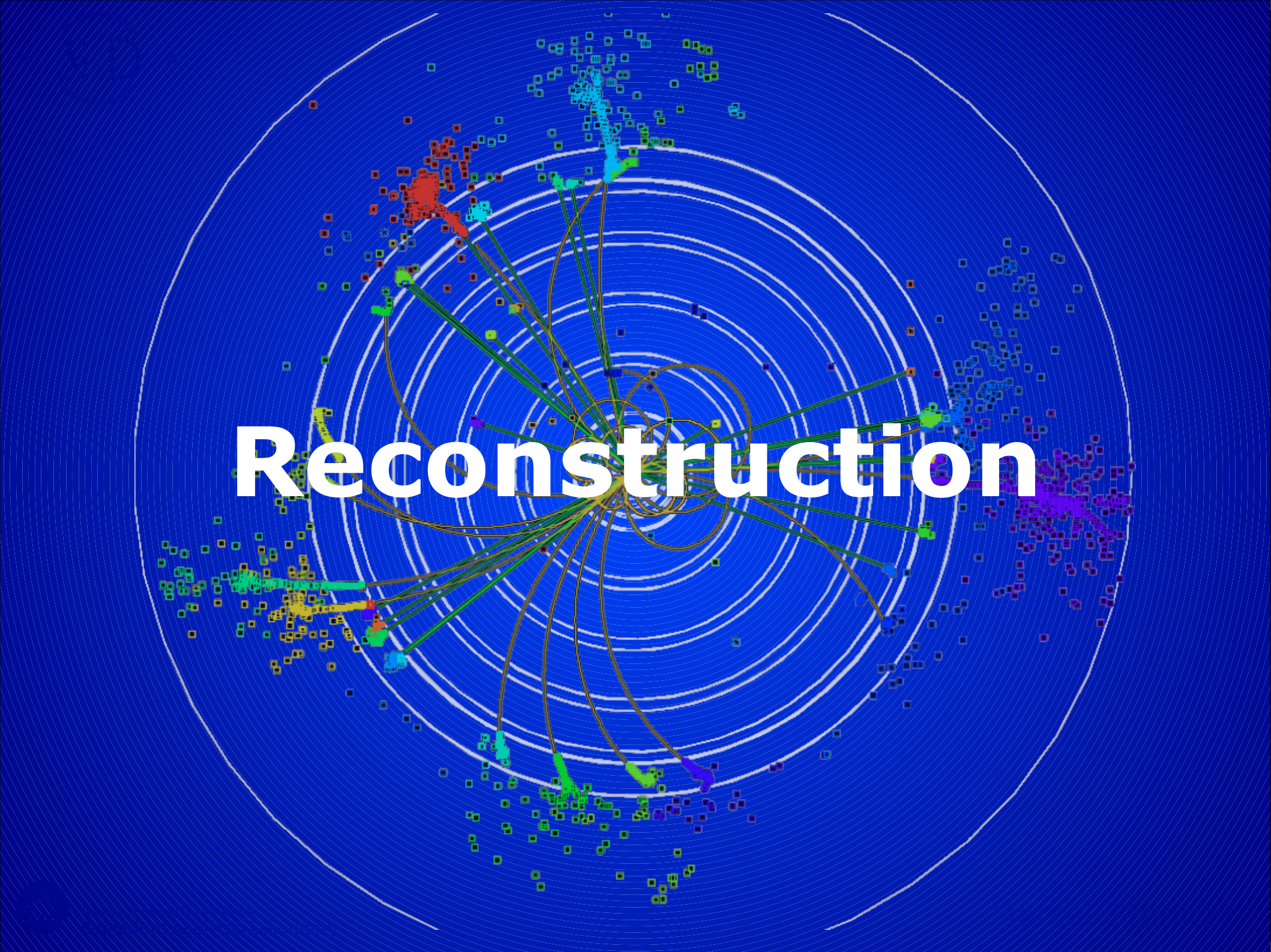
Sample events





$$e^+ e^- \rightarrow t\,\bar{t} \rightarrow 6\ jets$$

CLIC at 3 TeV


Reconstruction

The Reconstruction

- Can be divided in two parts
- Digitization
  - Make realistic hits
  - thresholds, charge sharing, etc.
- Event Reconstruction
  - Tracking, Clustering
  - Particle Flow
- Both is handled with org.lcsim

org.lcsim

- Java framework for analysis
- Using “Drivers” to analyze data
- AIDA used for histogramming
- Linked with JAS http://jas.freehep.org/jas3
- Linked with WIRED event display
- Also can be run standalone
- Uses LCIO extensively
- Geometry description from compact.xml
- Added bonus
  - Python bindings using Jython

A short example

```
import org.lcsim.util.aida.AIDA;
import hep.physics.vec.VecOp;
import java.util.List;
import org.lcsim.event.EventHeader;
import org.lcsim.event.MCParticle;
import org.lcsim.util.Driver;

public class Analysis101 extends Driver
{
   private AIDA aida = AIDA.defaultInstance();

   protected void process(EventHeader event)
   {
      List<MCParticle> particles =
      event.get(MCParticle.class,event.MC_PARTICLES);

      aida.cloud1D("nTracks").fill(particles.size());

      for (MCParticle particle : particles)
      {
         aida.cloud1D("energy").fill(particle.getEnergy());
         aida.cloud1D("cosTheta").fill(VecOp.cosTheta(particle.getMomentum()));
         aida.cloud1D("phi").fill(VecOp.phi(particle.getMomentum()));
      }
   }
}
```

Java imports

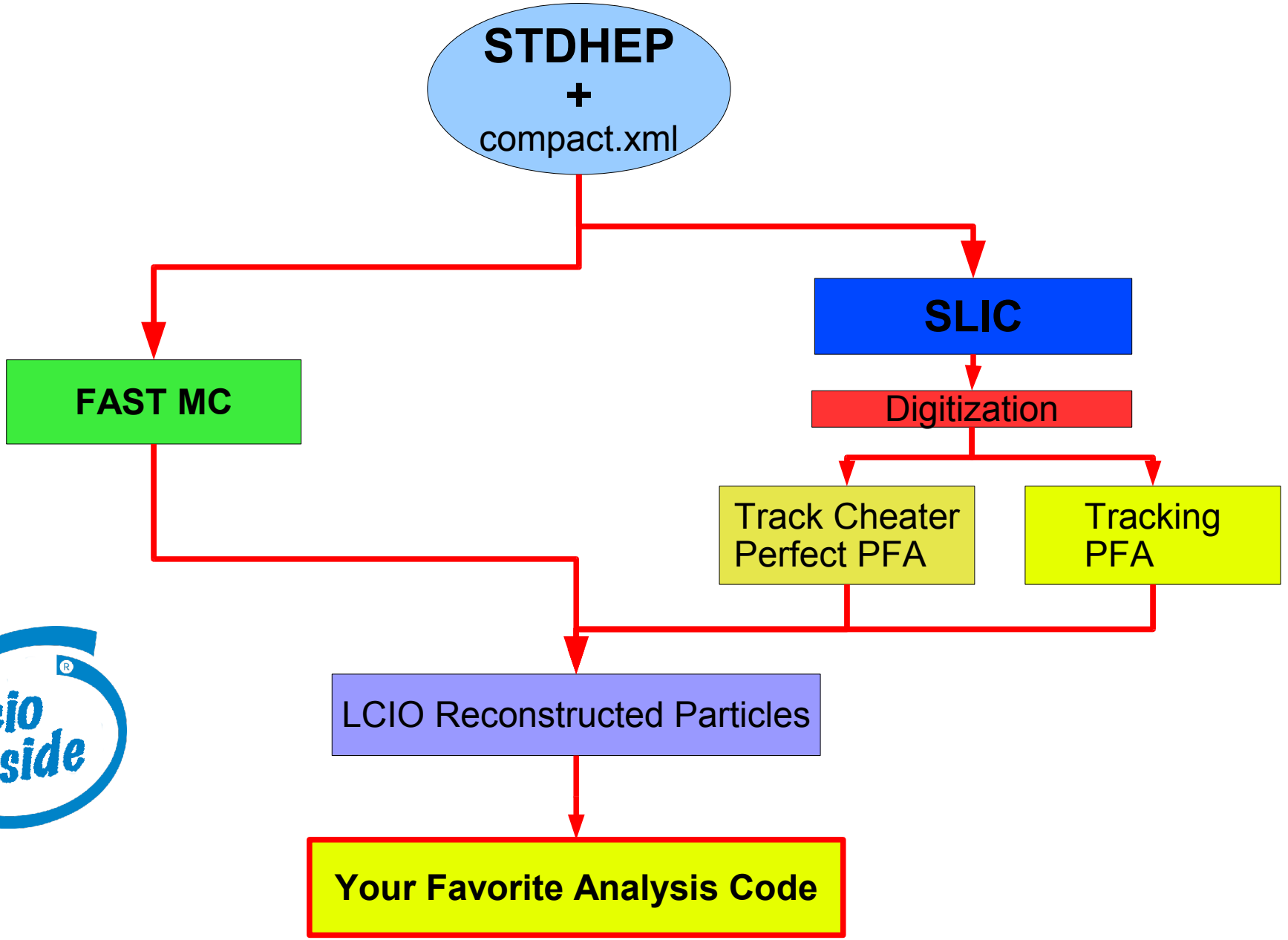
Make your own Driver

Loop over Data

What is available

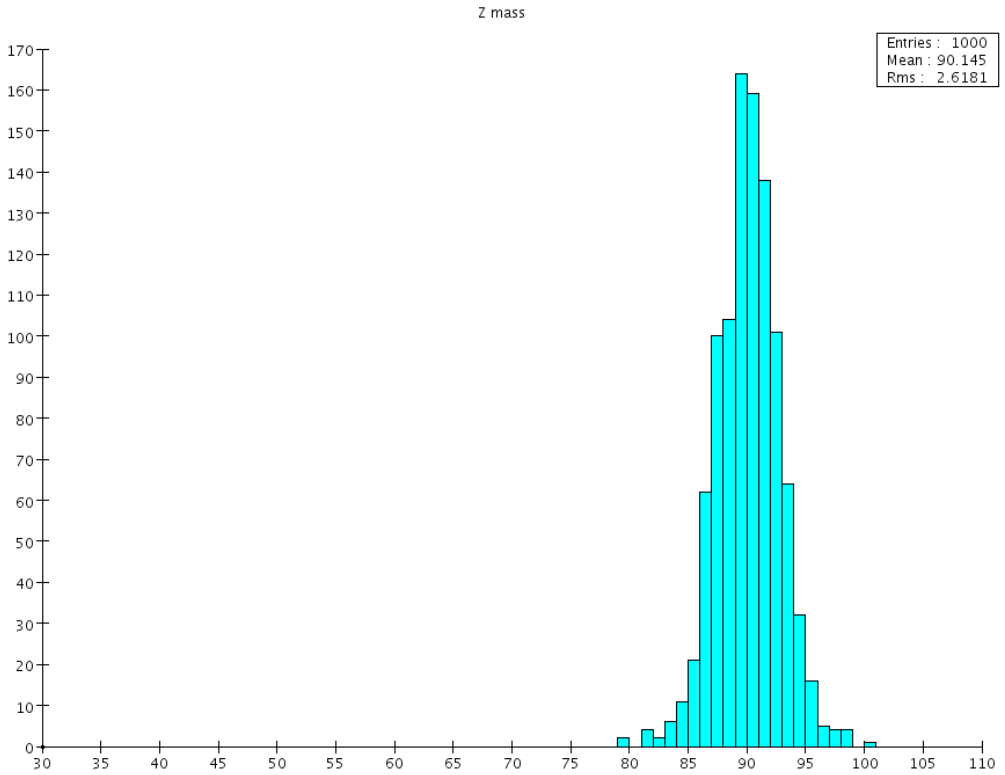
- Digitization packages
  - Calorimetry, Tracking ...
- Physics tools
  - Vectors, Jet Clustering, Event Shapes
  - Minuit Wrapper +JMinuit
- Event Reconstruction
  - Track Cheaters
  - Tracking
  - FastMC parametric simulation
  - Particle ID
  - PFA & Perfect PFA

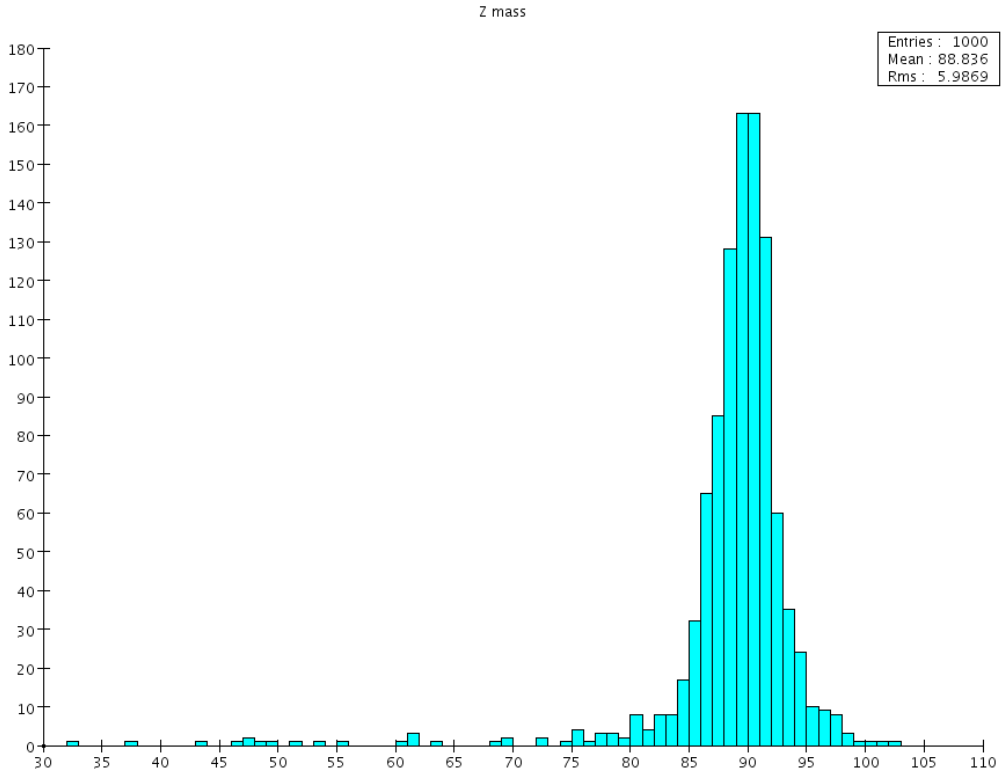
A quick way to analysis



One Example

1000 Events $Z \to uds$ @ 91 GeV





FASTMC

m_z= 90.15 ± 2.62

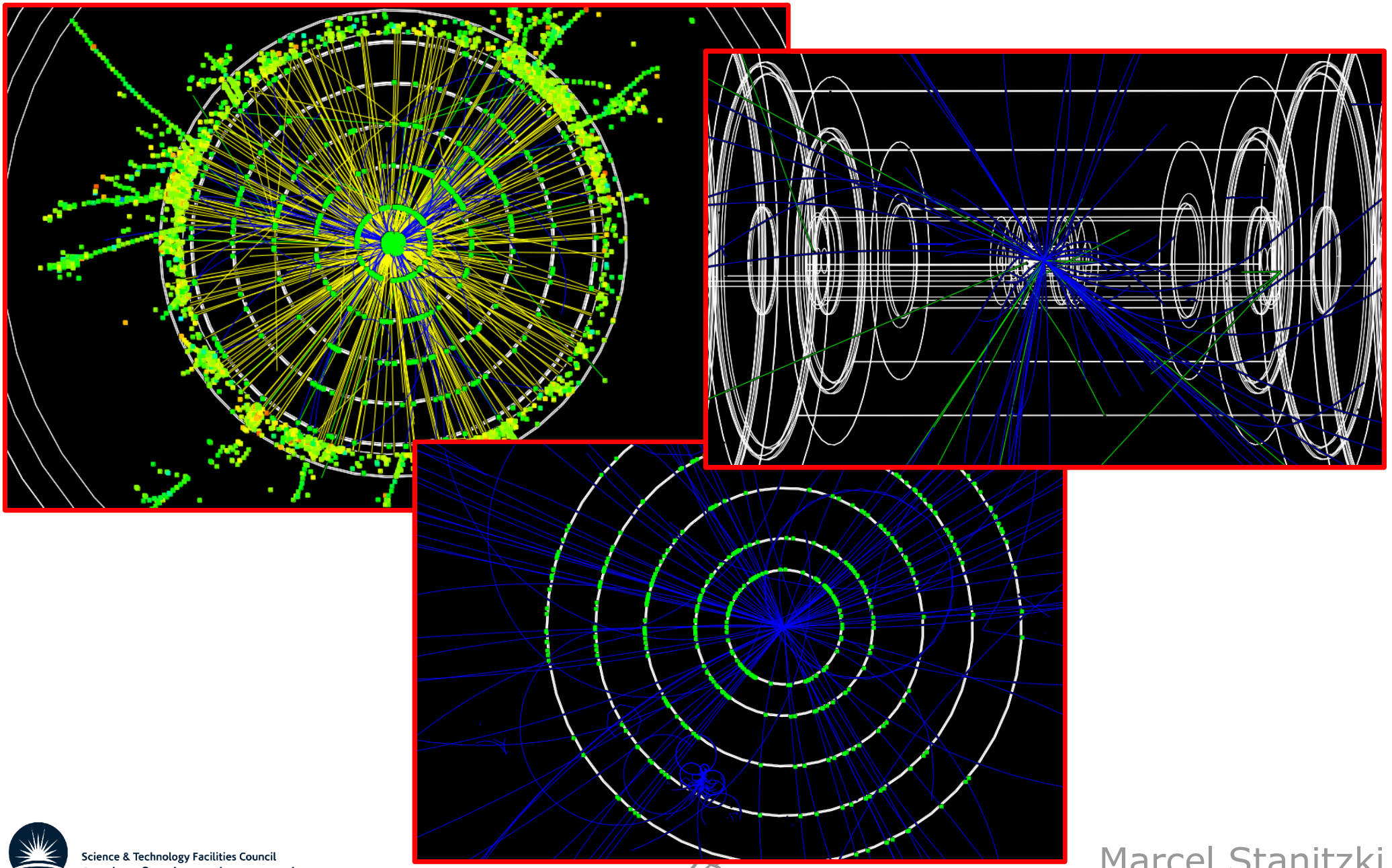
Perfect Pattern Recognition

m_z=88.84 ± 5.99

Jython ...

```
from org.lcsim.util.aida import AIDA
from hep.physics.vec import VecOp
from org.lcsim.event import MCParticle
from org.lcsim.util import Driver

class Analysis102(Driver):
   def __init__(self):
       self.aida = AIDA.defaultInstance()

   def process(self, event):
       # only necessary when adding Drivers to __this__ class.
       # in this case process() shouldn't do anything else
       #Driver.processChildren(self, event)
       # Get the list of MCParticles from the event
       particles = event.get(MCParticle.class, event.MC_PARTICLES)
       # Histogram the number of particles per event
       self.aida.cloud1D("nTracks").fill(particles.size())
       # Loop over the particles
       for iParticle in range(particles.size()):
          particle = particles.get(iParticle)
          self.aida.cloud1D("mass").fill(particle.getMass())
          self.aida.cloud1D("pSquared").fill(particle.getMomentum().magnitudeSquared())
```

WIRED



Current Status

- Tracking and PFA are actively being developed
- Frequent updates
- Will have a frozen "releases" for the LoI
  - That is a good starting point
- Can always start studies using
  - FastMC (very simplistic)
  - Cheaters using full GEANT simulation
  - LCIO output is the same

Beam Backgrounds

- GuineaPig pairs and photons
  - Added crossing angle, converted to stdhep, available here.
- Muons and other backgrounds from upstream collimators & converted to stdhep.
  - Need to validate and understand normalizations.
- $\gamma\gamma$ hadrons generated as part of the "2ab^{-1} SM sample."
- All events then capable of being processed through full detector simulation.
- Additive at the detector hit level, with time offsets, using LCIO utilities.
  - i.e. simulate response separately for signals and backgrounds, then merge them

Why Java ?

- JAVA is platform independent
  - easy to move between plenty of platforms
- Well defined Standard
- Good Eco-System
  - Compilers, IDE's, Libraries
- Garbage Collection
  - less memory leaks
- JAVA less error-prone than C++
  - faster turn-around for development

And ROOT ?

- There is no JAVA ROOT bindings (afaik)
- ROOT I/O poorly documented
- JAS can read ROOT files
- AIDA files can be converted to ROOT files
  - supports only tuples so far
  - Check out Jan's Code
- It is possible to move between ROOT and the rest of the world

Some Recommendations Simulation

- Choice of Physics List for GEANT
  - LCPhys, LHEP, ...
  - Choose **one** as baseline for comparison
- Choice of a baseline detector
  - Where everyone can refer to for comparison
- Define some standard benchmark numbers
  - e.g. Jet energy resolution with qq using rms_{90}
- The phase space is large so it helps to have some guidance

Some Recommendations MonteCarlo

- A central stdhep repository is very useful
  - People use same inputs
  - Common MC generators
  - establish a baseline
- LCIO files simulated with SLIC should also go there
- SLAC ILC repository is very helpful

Some Recommendations Reconstruction

- Stick to one Software Framework
  - helps focus effort
  - comparable results
- No need for yet another Linear Collider Software Framework
  - Effort is limited as it is
- Already plenty of code pieces available

Computing time

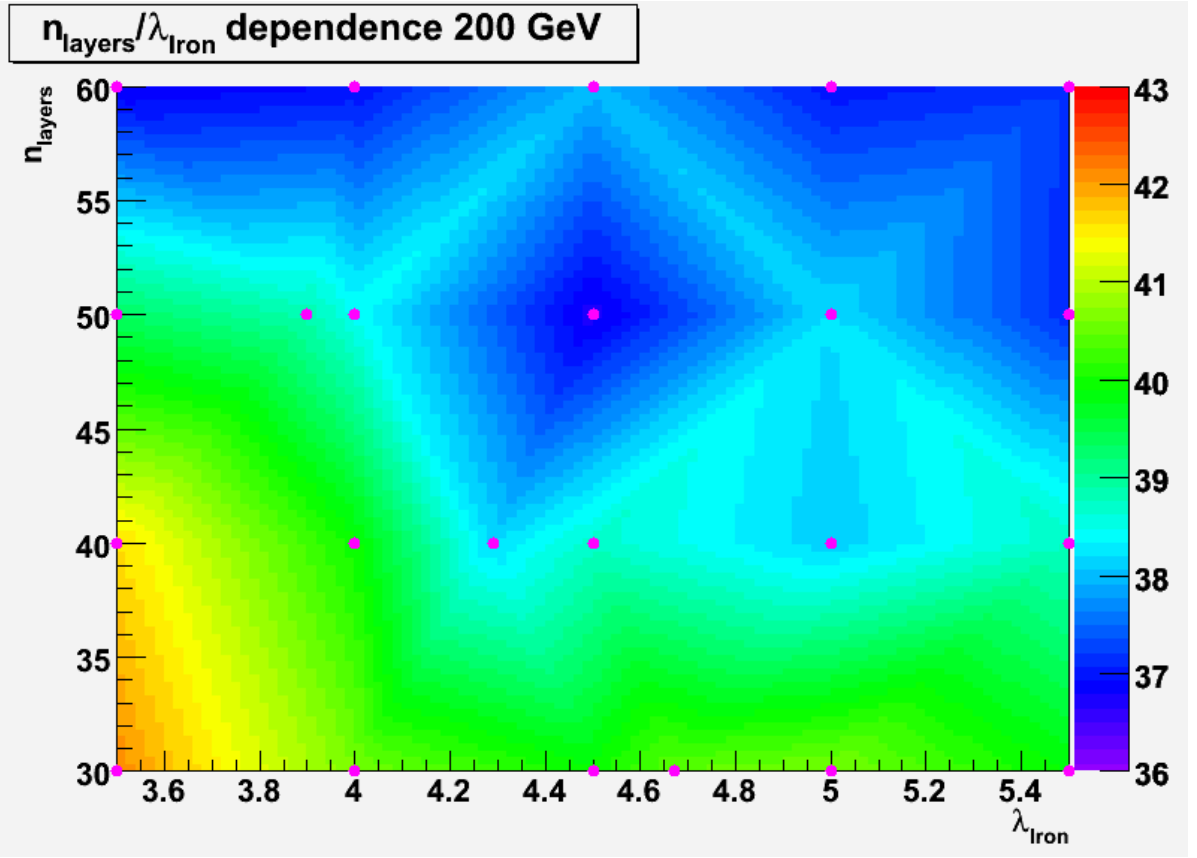
- A significant factor
- $e^+ e^- \rightarrow t\,\bar{t} \rightarrow 6\ jets$
  - ~ 5 minutes per Event (2.8 GHz Xeon) in SLIC
  - ~ 2 minutes for reconstruction in org.lcsim
- That gives the right ball park
- for a sensible analysis about signal 20000 events

Detector Optimization

- e.g. for a PFA detector
- The **Big Five**
  - Magnet Field
  - Tracker radius
  - Tracker length
  - ECAL/HCAL depth
  - ECAL/HCAL layers
- Plus
  - pixel sizes, material choices ..

Detector Optimization examples

- Study of HCAL
- Layers vs λ_{Iron}
- 26 points
- Each point needs 40000 events
  - Calibration (γ,neutrons)
  - qq
- total of ~1 million events for one study
- ~ about 4 weeks to produce on 75 nodes



Summary

- SiD framework allows doing studies with different detail
  - One detector description
  - One framework
  - One analysis code
- Computing needs are significant
  - Detectors are complex Number of channels is huge
  - Can be a limiting factor
- Detector optimization
  - Fix global parameters first (The Big Five)
  - then optimize sub-detectors afterwards
- How to get started ?
  - That is why we are here !

Additional resources

- SiD – http://silicondetector.org/
- lcsim.org - http://www.lcsim.org/
- ILC Forum - http://forum.linearcollider.org/
- Wiki - http://confluence.slac.stanford.edu/display/ilc/Home
- org.lcsim - http://www.lcsim.org/software/lcsim
- Detectors - http://www.lcsim.org/detectors
- LCIO - http://lcio.desy.de/
- SLIC - http://www.lcsim.org/software/slic
- JAS3 - http://jas.freehep.org/jas3
- AIDA - http://jas.freehep.org/jas3
- WIRED - http://wired.freehep.org/
-