The Linear Collider Software Framework

André Sailer[1], Akiya Miyamoto[2], Norman Graf[3], Frank Gaede[4]

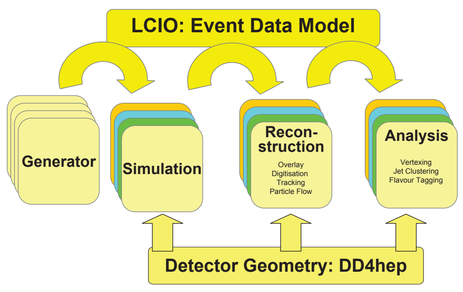
[1]CERN, [2]KEK, [3]SLAC, [4]Desy

Overview

For the future experiments at linear electron–positron colliders (ILC or CLIC), detailed physics and detector optimisation studies are taking place in the CLICdp, ILD, and SiD groups. The physics performance of different detector geometries and technologies has to be estimated realistically.

Software shared by the linear collider detector groups:

- Event data model and persistency format: LCIO
- Python and `cmake` based installation and configuration system
- Detector geometry description based on DD4hep (under development) and `Geant4` based simulation programs
- Java and C++ based reconstruction frameworks providing
  - Overlay of beam-induced backgrounds
  - Digitisation
  - Tracking
  - Particle Flow
  - Flavour Tagging
- Moving to more and more common tools



Event Data Model

LCIO (Linear Collider Input/Output) is the event data model and persistency format

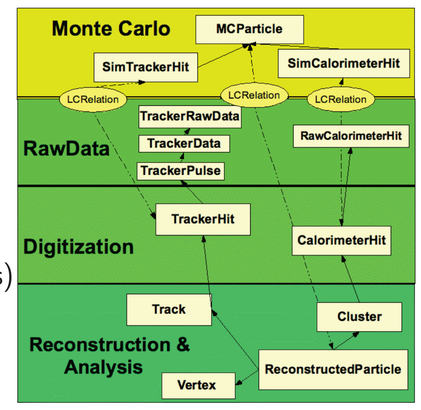
- API for C++ , Fortran (C), Java, and *Python* (via root dictionaries and `pyRoot`) are available

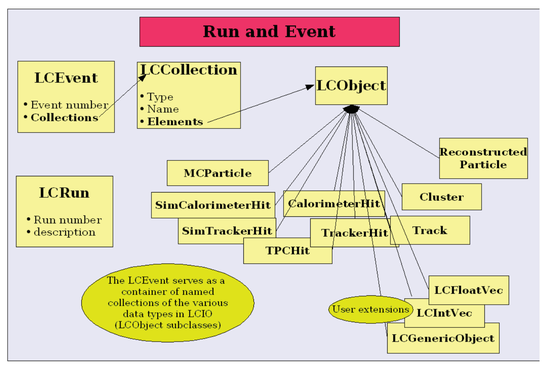
Provides the 'common language' for the LC community

- Sharing of files *and* software possible
- Used by detector groups and hardware RnD collaborations (e.g., Calice, LCTPC, EUTelescope)

Hierarchical model, where higher level objects can point back to their constituents

- Only indirect links between Monte Carlo Truth and reconstructed objects
- Allows for relations between MCParticles and Reconstructed objects (Clusters, Tracks, Particles)
- *Generic enough to go beyond linear collider detectors* (no external dependencies)





File Format and Event Store

- Event data model decoupled from persistency package
- Current persistency format: simple file format (SIO)
  - Compression via zlib
  - Pointers inside the record done via integer IDs and look-up table
- LCIO Event Store
  - Data are stored in named collections
  - Multiple collections of each type possible

```
from ROOT import TH1D
from pyLCIO.IOIMPL import LCFactory
import sys

hEn = TH1D('hEn', 'Energy', 100, 0, 10.0)
rdr = LCFactory.getInstance().createLCReader()
rdr.open('sys.argv[1]')

for evt in rdr:
  mcCol = evt.getCollection('MCPs')
  for mcP in mcCol:
    if mcP.getGeneratorStatus() == 1:
      hEn.Fill(mcP.getEnergy())

hEn.Draw()
```

Future plans

- Evolve LCIO without breaking existing code base (too much)
- Alternative for SIO files: fast I/O layer (based on ROOT)
- Possibly use HepMC for easier interface with MC generators

Geometry and Simulation

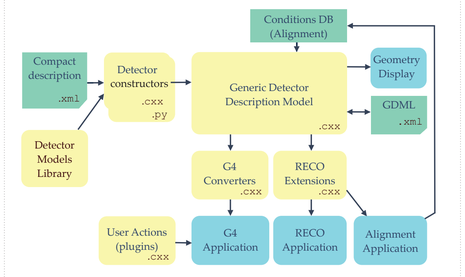
Simulation, reconstruction, and analysis steps require information about the detector geometry

- The required information is different, but has to be consistent
- Requires a tool to provide a unique source of geometry with different *views*

Detector optimisation means varying detector parameter

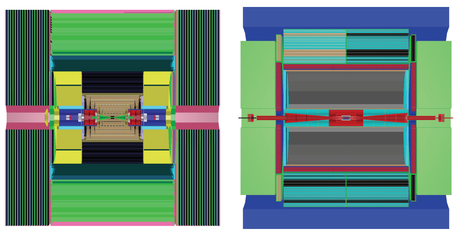
- Need flexible geometry to study detector performance with variable parameters (e.g., tracker radius, number of layers in calorimeters)

Decided to use `DD4hep` as single source of detector geometry



Courtesy of M. Frank (CERN)

Example Detector models in the `DD4hep` geometry



- Implementing **sub-detectors** in new geometry system
  - Implementing more-or-less generic, scalable sub-detectors (Trackers, Calorimeters) to be shared between detector groups and beyond
- Implementing **segmentation classes** (DDSegmentation) to be used in simulation and reconstruction
  - Cell ID encoding in simulation, and retrieve location only from cell ID in reconstruction
- Implementing **sensitive detectors** to be mixed and matched with segmentations

Simulation will be done with `Geant4`, investigating two options to pass `DD4hep` geometry to `Geant4`

- Create `LCDD`-files and use established simulation software `SLIC`
- In memory translation from `DD4hep` to `Geant4`: `DDG4` component of `DD4hep`

Grid Production Tool

The physics case for the linear colliders require large Monte Carlo campaigns to evaluate the physics potential. This is multiplied by the number of possible detector options studied. One requires a simple to use interface to the GRID computing resources

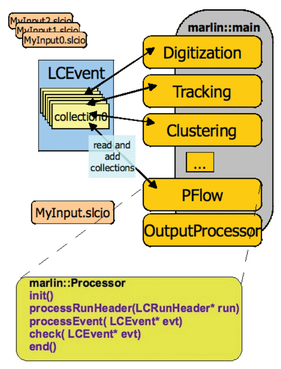
- ILCDirac extension of the DIRAC grid interware
- **High level interface** to run linear collider applications on the GRID: make it simple for the users
  - Submission of complicated workflow through chaining of applications
  - Makes use of the common LCIO event data model and file format

```
from DIRAC.Core.Base import Script
Script.parseCommandLine()
import UserJob
import Marlin
import DiracILC
d = DiracILC()
j = UserJob()
j.setOutputSandbox("recEvents.slcio")
m = Marlin()
m.setVersion("0116")
m.setSteeringFile("Steering.xml")
m.setInputFile("SimEvents.slcio")
j.append(m)
j.submit(d)
```

Reconstruction Software

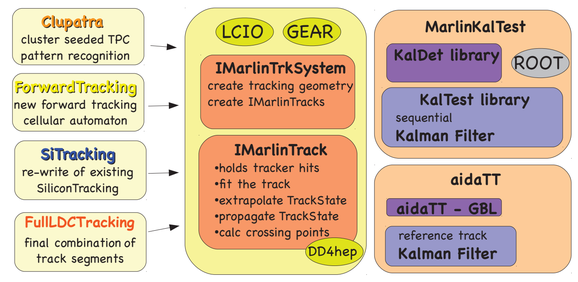
Existing linear collider reconstruction software has been successfully used for many physics studies and detector benchmarks. More and more realism in the detector models requires further sophistication and performance improvements of the reconstruction software

- Improve existing reconstruction software
  - Adapt reconstruction software to make use of the geometry information provided via `DD4hep`
  - Focus on improvement of track reconstruction
  - Develop realistic digitisation for the technologies considered for the linear collider detectors (e.g., silicon sensors, silicon photo multipliers, gaseous detectors)
- Particle flow reconstruction was improved
- LCFIPlus package has been developed for highly efficient b-jet tagging as well as c-jet detection



Data and Algorithms

Tracking and Interfaces



Tracking Software

Existing Tracking Software

- `Java` based silicon tracking in `org.lcsim`
- C++ based tracking software in `Marlin`: First separate steps for TPC and silicon tracking and then merging of track segments

AIDA Tracking Toolkit for track finding and fitting under development

- Design:
  - Completely modular
  - Well defined API for reconstruction frameworks
  - Separate data, algorithms, and functionality
  - Parallelizable on track level
- Features:
  - Track fitting (and finding), propagation and extrapolation
  - Transparent switch between different track fitting algorithms
  - Interface with `DD4hep` via `DDSurfaces`
- Track reconstruction in inhomogeneous magnetic fields

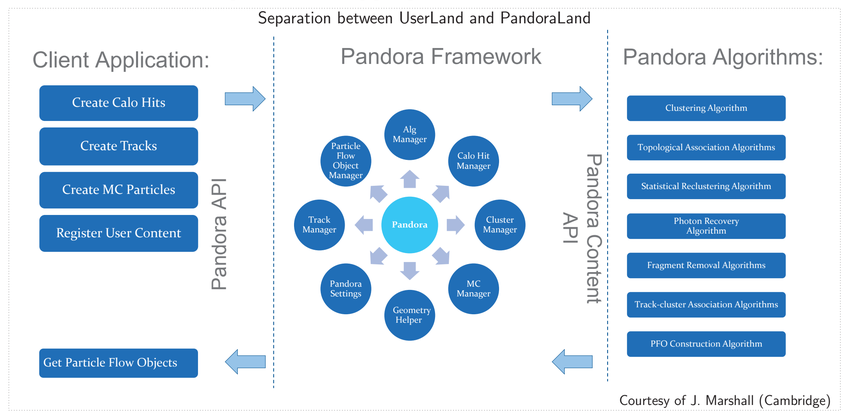
Particle Flow

At the heart of the Linear Collider Detectors lies the **Particle Flow Reconstruction**

- Reconstruct all particles with the most suitable detector: charged particles with trackers, photons and neutral hadrons in the Calorimeters
- Requires highly granular calorimeters and sophisticated clustering algorithms to separate showers from charged and neutral particles

Most LC detectors studies are using the PandoraPFA package

- PandoraPFA used in ILD, SiD, CLICdp for detector optimisation: Extensive testing and debugging for one detector model benefits all others
- Standalone package: clustering is independent of the reconstruction framework
- Geometry information and detector hits are passed to Pandora through lightweight interfaces
- Common use of particle flow package in different detectors allows for robust testing of the generic algorithms
- CMS detector optimisation studies using PandoraPFA for possible future high granularity calorimeter endcaps



Courtesy of J. Marshall (Cambridge)

Mail: andre.philippe.sailer@cern.ch

WWW: http://ilcsoft.desy.de, http://www.lcsim.org, http://aida.web.cern.ch