ILC Framework for Belle II

Kolja Prothmann,
Christian Kiesling,
Andreas Moll,
Frank Simon
Max-Planck-Institut für Physik
Zbynek Drasal
Charles University Prague

Max-Planck-Institut für Physik
(Werner-Heisenberg-Institut)

Excellence Cluster
Universe

Outline

- Experiments using ILC software
- Overview of ILC framework
  - Data model: LCIO
  - Geometry and particle transport: Mokka / Gear
  - Digitization and Reconstruction: Marlin / Marlin GUI
  - Event Display: GeV
- Discussion
- Conclusion

available on: */bwf/g67home/kolja/tutorial*

Experiments using the ILC Framework

Calorimeter test experiment
data taking ~100 Mio events
~1 year of H1 data

LOI Letter of Intent
For their ILD detector at ILC
MonteCarlo production
~50 Mio events

Eudet project

Test experiment
For tracking
Detectors
(TPC, Sidet)

CLIC: Simulation Studies

Manpower on the ILC Framework

- ILC Community Europe
  - Frank Gaede
  - Steve Aplin
  - 2 Post Docs

  } DESY/ LCIO

  France → Mokka

  Czech Republic (Prague)

  Austria (Vienna)
- Japan (ILC community)
- CERN CLIC

- LOI done
  now 1 year of framework development until TDR

  ROOT/IO will be implemented

  If framework is chosen for Belle II:

  → Framework team within Belle II needed

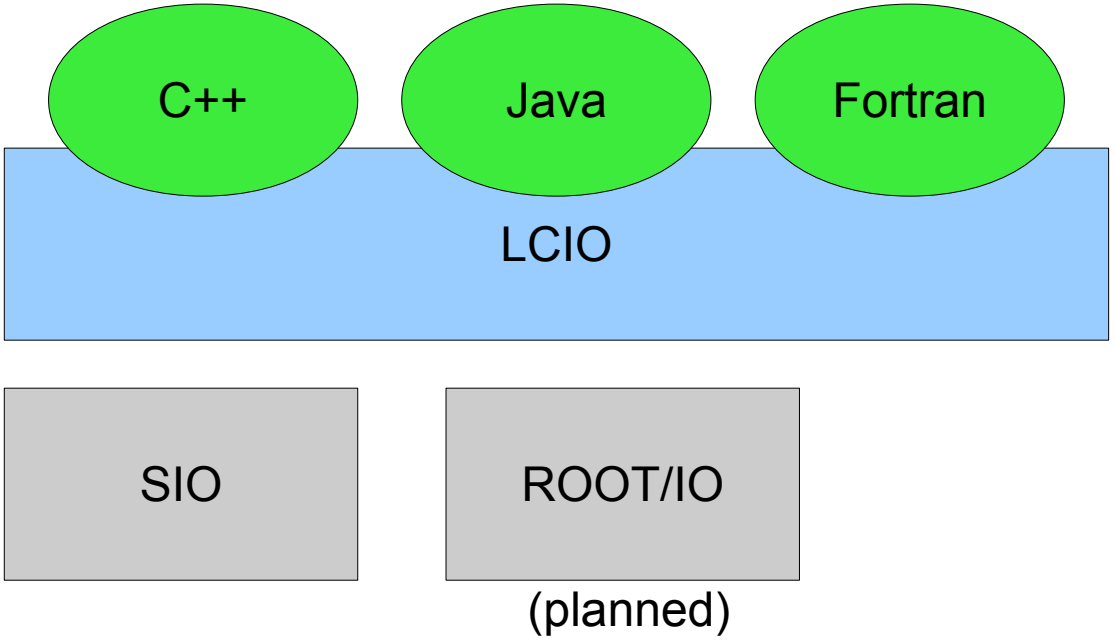
Data Model

transparent to the user

All languages see the same **API** (same Objects)

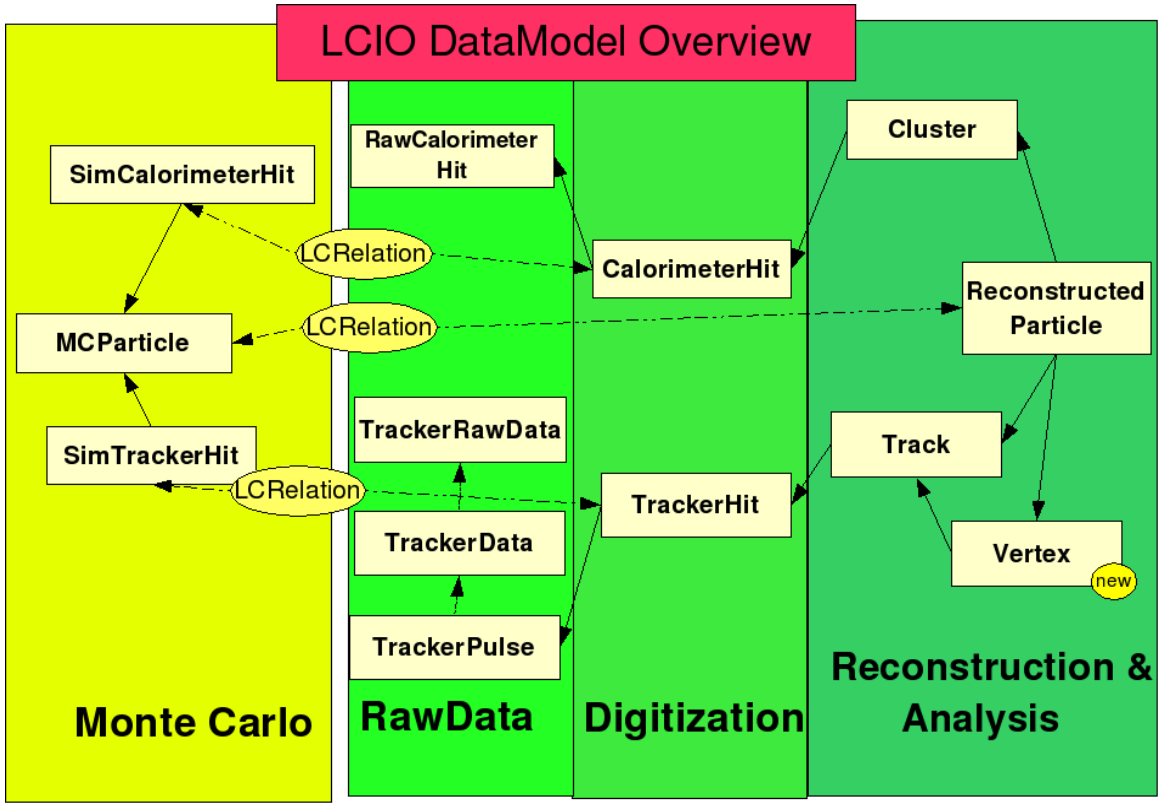
Data Model:
Objects and Relations

Persistence layer:
- stores objects in files
- designed to be exchanged easily

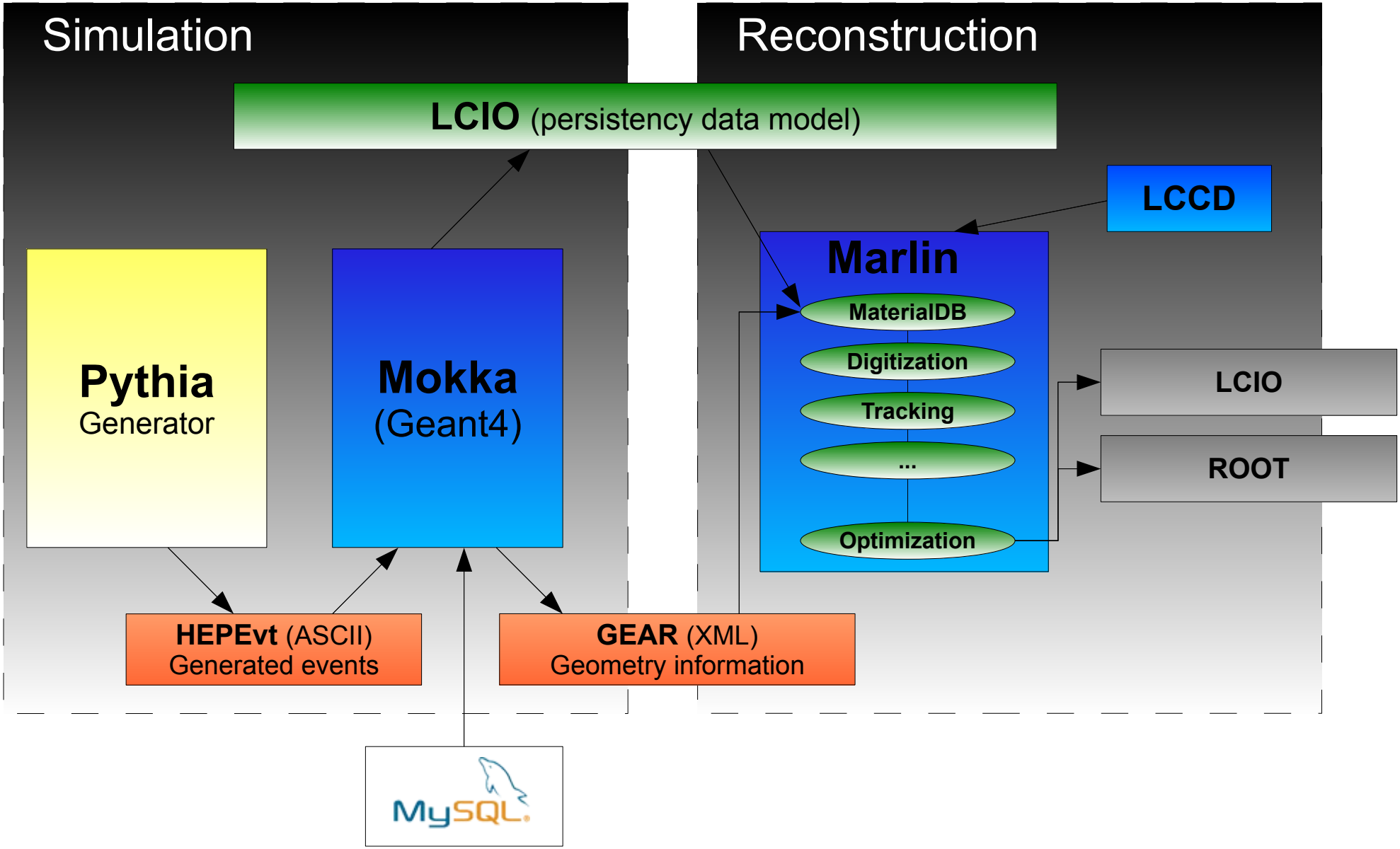


Data Model: details

- bidirectional Relations between LCObjects
  - one to one
  - one to many
  - many to many
- suitable for DAQ and Raw Data
- Model can be modified for Belle II (for example PID)



Schematics of the ILC Framework

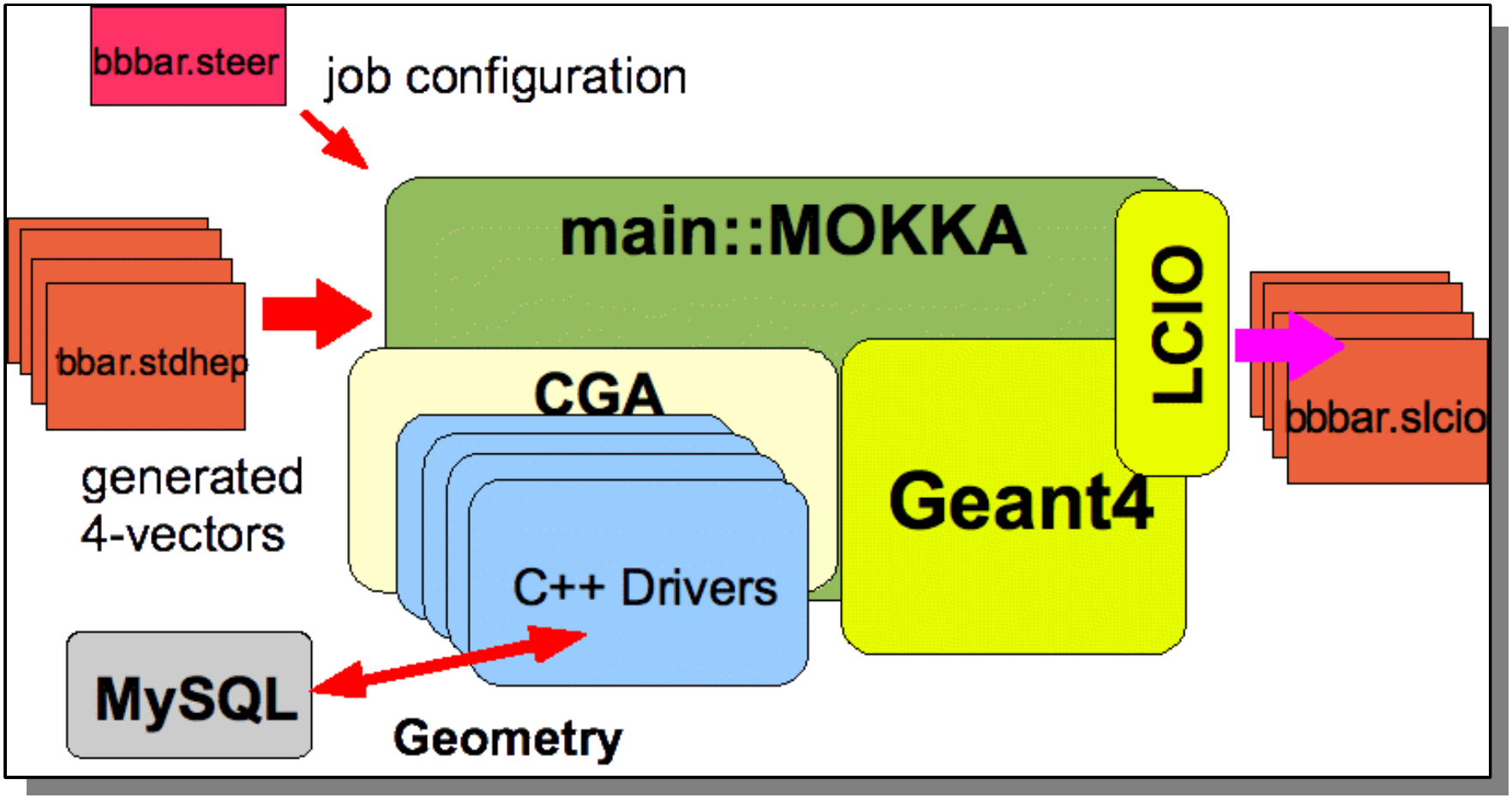


Panther banks to HEPEvt converter

- new BASF module (hepevt)
- Extracts the Monte Carlo information generated by **evtgen**
- reads the **Gen_HEPEVT** table from the panther banks
- writes out a std. **HEPEvt** file which can be used in **Mokka**

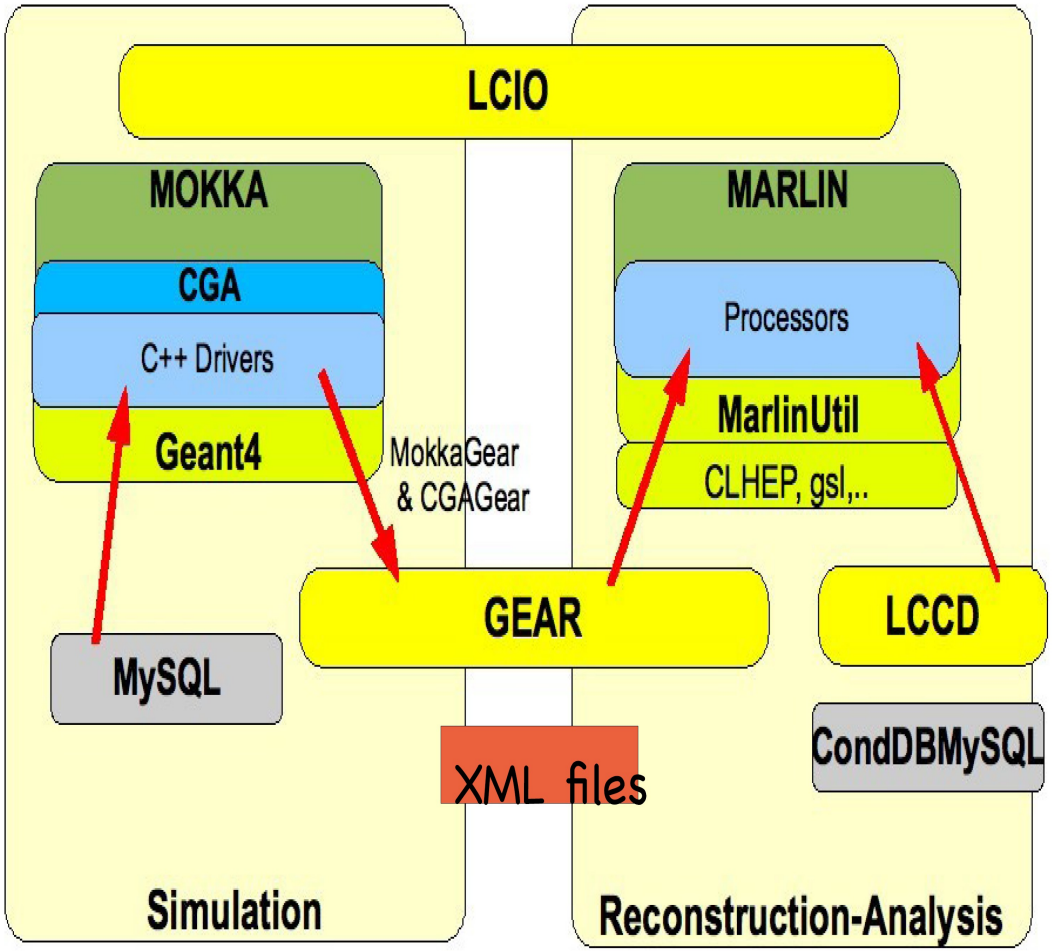
➔ Bridge from BASF to ILC framework!

Geometry and Particle Transport: Mokka



GEAR – GEometry Api for Reconstruction

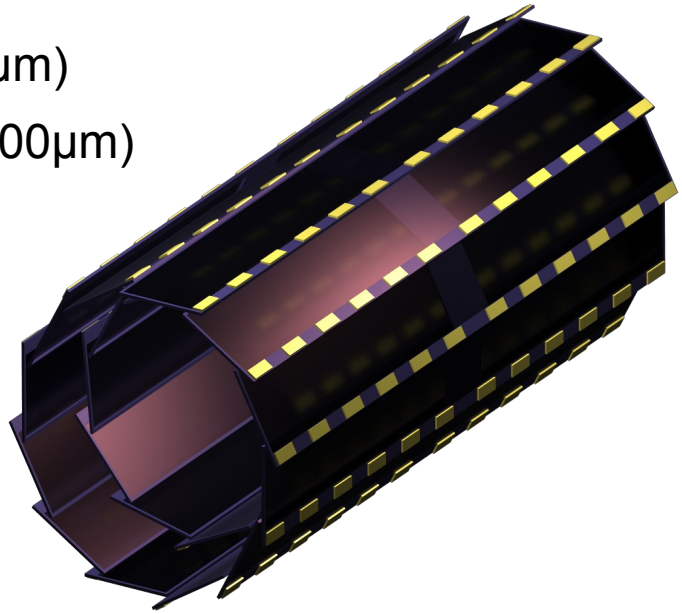
- Provides a simplified geometry description for reconstruction
- Provides separation between Data(LCIO) & Geometry
- at the moment data is stored in a separate XML file
  - The datasource can be changed to Belle II needs e.g. use a database



Mokka – Belle II Geometry - 1

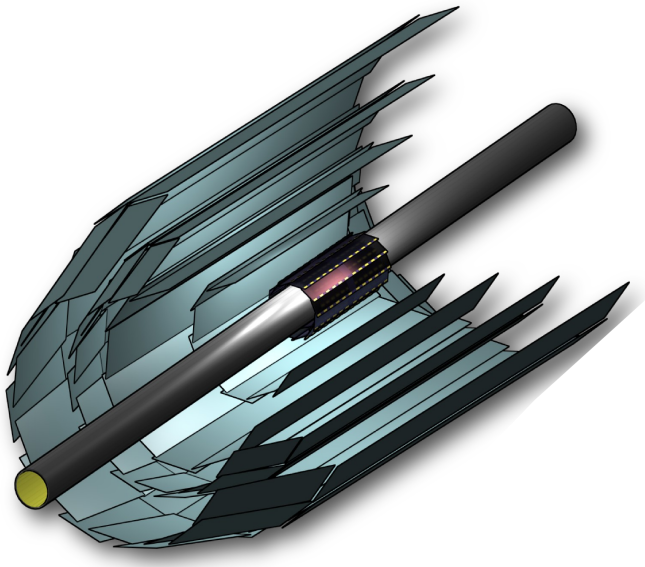
- Mokka model: ***complete Belle II Tracker*** (beam pipe + PXD + SVD + CDC)
  - ***Beam pipe:*** cylindrical onion-like structure
    - inner golden layer + inner Be wall + cooling gap (paraffin) + outer Be wall
  - ***PXD:*** 2 layers of Si pixel detectors – DEPFETs
    - active part: layers → ladders → Si sensors (50μm)
    - passive part: Si rims (450μm) + 12 switchers (300μm)

	R [mm]	# ladders
PXD layer 1	18	10
PXD layer 2	22	12

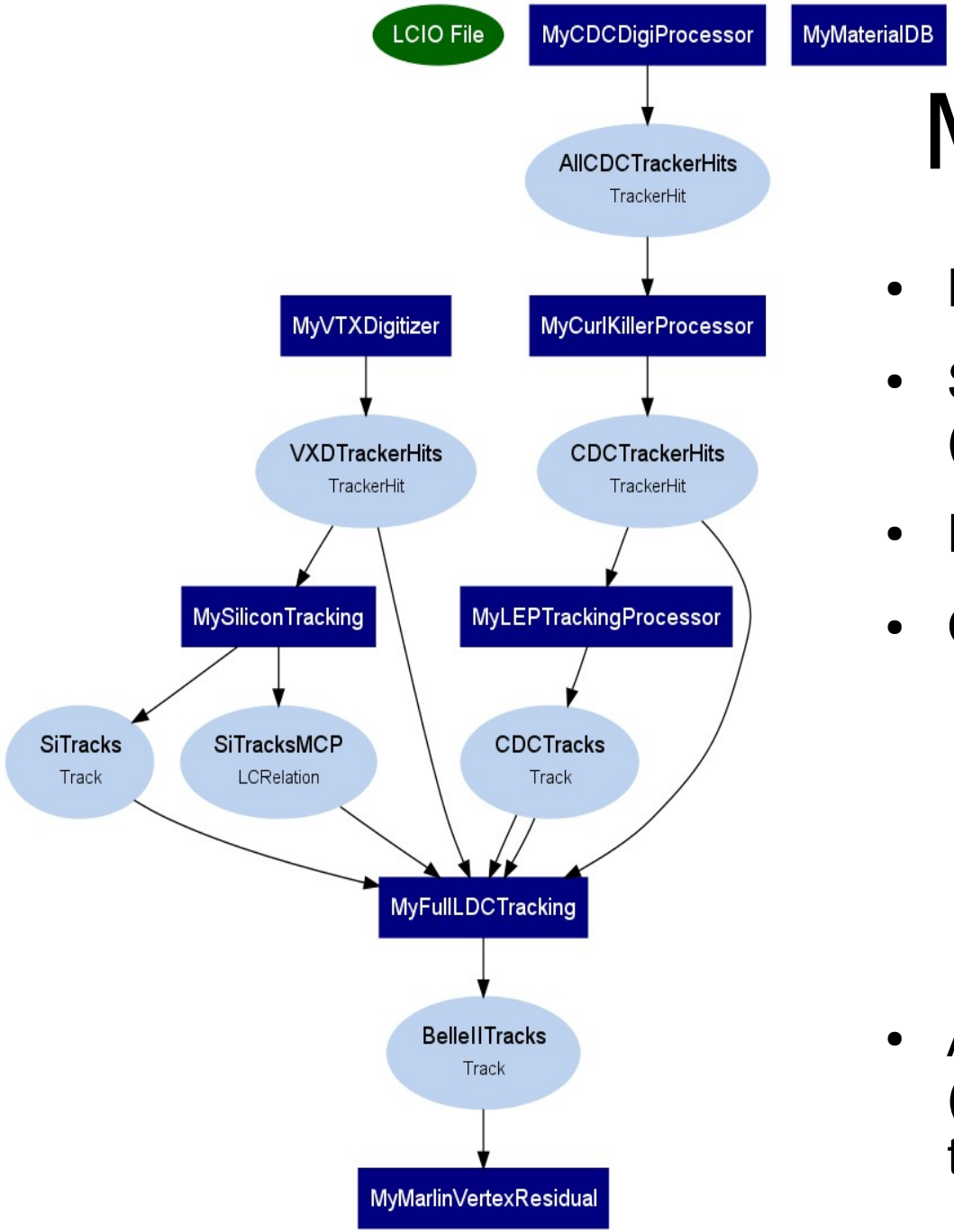


Mokka – Belle II Geometry - 2

- ***SVD:*** 4 layers of Si strip detectors (DSSDs) in barrel part
  - organized in stagger-like structure
  - active part: layers → ladders → Si sensors (300μm)



- ***CDC:*** Al cylinder with cone-shaped inner parts (as Belle)
  - active medium: gas He/C_2H_6 (50:50)
  - uses Gaussian smearing as digitization
  - geometry as of December 2008



Marlin Processors

- Digitizers (PXD, SVD and CDC)
- Stand alone tracking (PXD+SVD or CDC)
- Full tracking (PXD+SVD+CDC)
- Other Processors available:
  - LCFIVertex (Vertexing)
  - Pandora (Particleflow: ILC specific)
  - Curlkiller, ...
- Analysis Processor (Knowledge of Belle would go in there)

How to create a Marlin steering file?

- Use the MarlinGUI

- Let Marlin create a sample XML steering file containing all available processors:
  Marlin -x > steering.xml
  remove unwanted processors with editor

- Graphical representation of your processor chain:
  Marlin -d steer.xml flow.dot
  dot -Tpng flow.dot flow.dot.png

MarlinGUI

File View Help

List of all Collections Found in LCIO Files

Name	Type

Global Section

Global Section LCIO Files

CWD: /home/pclh1-6/kolja

Add
Remove
Move Up
Move Down

Global Section Parameters

Parameter Name	
GearXMLFile	gear_
MaxRecordNumber	5001

Browse for GEAR File

View Options

Hide Inactive Processors | Hide Active Processor Errors

Active Processors

Name	Type
MySimpleFastMCProcessor	SimpleFastMCProcessor
MyConditionsProcessor	ConditionsProcessor
MyTestProcessor	TestProcessor
MyLCIOOutputProcessor	LCIOOutputProcessor

Operations

Add
Edit
Delete
Deactivate
Move Up
Move Down
Show Cond.

Error Description for selected Active Processor

Inactive Processors

Name

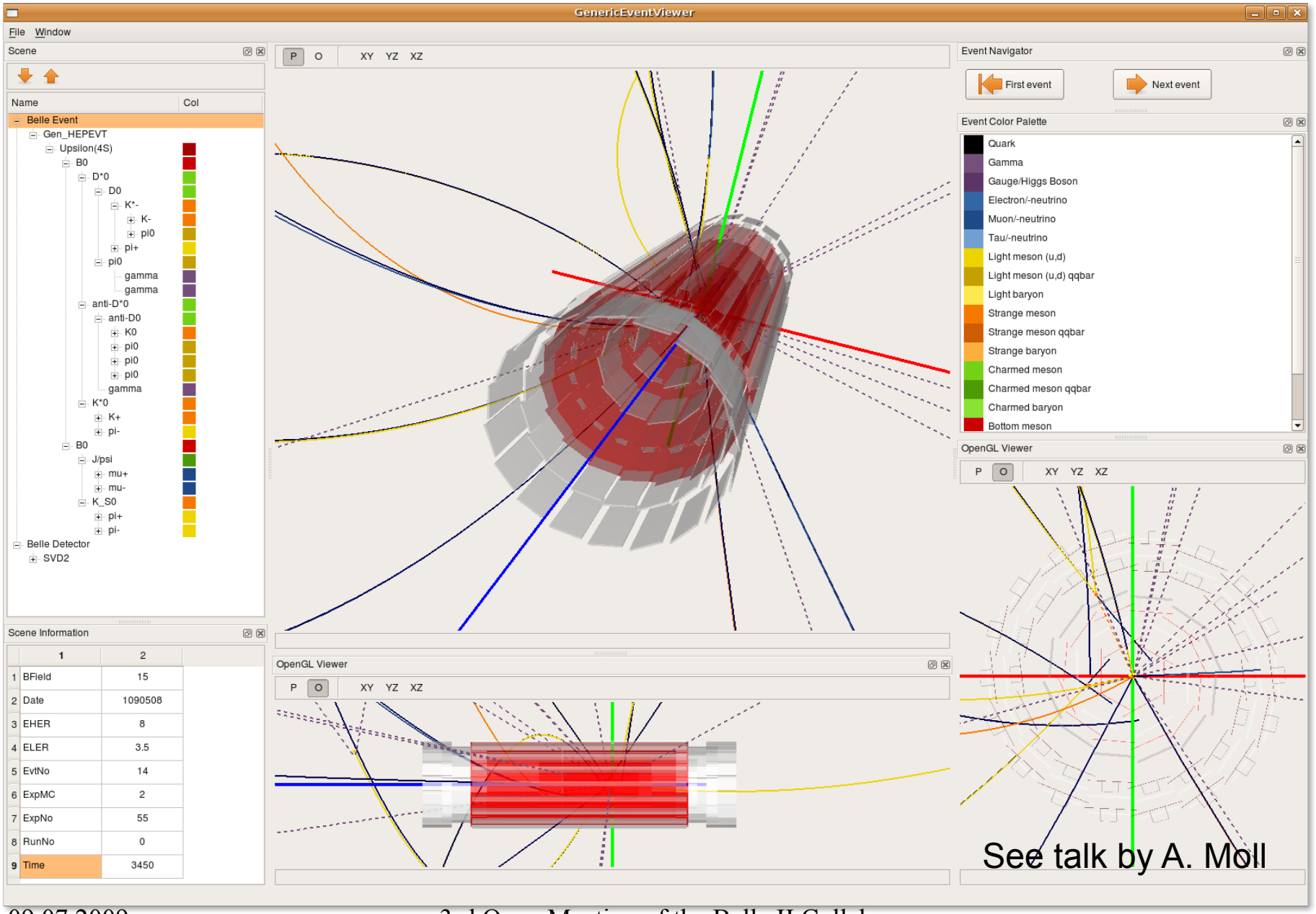
Operations

Activate
Edit
Delete

Writing a Marlin processor

- inherit from a generic Processor class
- implement
  - constructor()
  - init()
  - processRunHeader(LCRunHeader* run)
  - processEvent(LCEvent* evt)
  - end()
- parameters
  - input collections / output collections + collection relations
  - int, double, float, strings …
- advanced logging mechanism (streamlog)
  - level of detail (MESSAGE0-3, WARNING, ERROR)

GeV - Generic Event Viewer



What the ILC software cannot do at present

- No multithreading:
  - one instance of Marlin has to be started per CPU core
  - works on our CERN Tier2 center with 880 CPU cores with a simple python script
  - ROOT/IO and SIO are not multithreaded!
    - If we implement multithreadding: histogram manager + seed manager needed
    - a collection of python scripts for managing these issues
- Cannot write arbitrary objects to LCIO, only LCObjects
  - very clean data structure!
    - ATLAS and CMS use ROOT but separate data & histograms

  (ROOT/IO will be implemented as persistency layer within 1 year)

Features of the ILC software

- excellent documentation
  - both the code & usage of the code
- simple and clear structure

Very easy for newcomers!

- robust, fast and tested in running experiments!
- very modular flexible structure with processors
  - e.g. Tracking is done with several separate processors
- Already adapted to many Belle II needs (PXD/SVD Digitizers & Simulation)
- Belle II can save a lot of manpower by using the existing ILC framework

Conclusion

- ILC framework developed and used by ILC/CLIC community
- Modular very flexible structure
  - successfully adapted for Belle II Tracking
  - Geometry of PXD, SVD and CDC implemented
  - Realistic Digitization including clustering is done for PXD and SVD
- Further modifications needed to fully adapt to Belle II
  - PID, ECL and KLM can be added with limited amount of effort
- The ILC framework is running in KEK !!!

Feel free to test it!

/bwf/g67home/kolja/tutorial

Backup

How to start with ILC Frame

- ilc software installed in /bwf/g67home/kolja/ilc
- use bash to source environment script env.software.sh
  - *bash*
  - *.<space>/bwf/g67home/kolja/ilc/env.software.sh*
- All steering files macros and output files can be found in: /bwf/g67home/kolja/tutorial

The Mokka steering file

(mokkaTrkSBelle_CPS1600_SUP10.steer)

```
/Mokka/init/detectorModel TrkSBelle_CPS1600_SUP10
/Mokka/init/dbHost pcbelle01.mpp.mpg.de

/Mokka/init/modelsDBName Models
/Mokka/init/materialsDBName Materials

/Mokka/init/user belleII
/Mokka/init/dbPasswd belleII

#Geant4 macro file name
/Mokka/init/initialMacroFile pgun2.0GeV80.g4
/Mokka/init/lcioFilename TrkSBelle_CPS1600_SUP10.slcio

# gear output file name
/Mokka/init/MokkaGearFileName gearTrkSBelle_CPS1600_SUP10.xml

/Mokka/init/BatchMode true
```

Detector model

Geant4 macro

lcio output file

GEAR output file

no graphics

The Geant4 macro (pgun2.0GeV80.g4)

```
/run/verbose 0
/event/verbose 0
/tracking/verbose 0

# Name of StdHEP file to be used for simulation
#/generator/generator
/afs/ipp/home/k/kolja/belle/hepevt/mc03.HEPEvt
/generator/generator particleGun
/gun/position 0 0 0
/gun/direction 0.984808 0 0.173648
/gun/phiSmearing 180 deg
#/gun/thetaSmearing 20
/gun/directionSmearingMode uniform
/gun/energy 2.00 GeV
/gun/particle mu+


# Number of events to be simulated, greater than
# the actual number in HEPEvt file
/run/beamOn 1000
```

optional MonteCarlos

direction θ=80º

phi-smearing = +-180º

direction is uniformly generated

muon particle gun

number of events to sim.

Mokka – the geant4 simulation

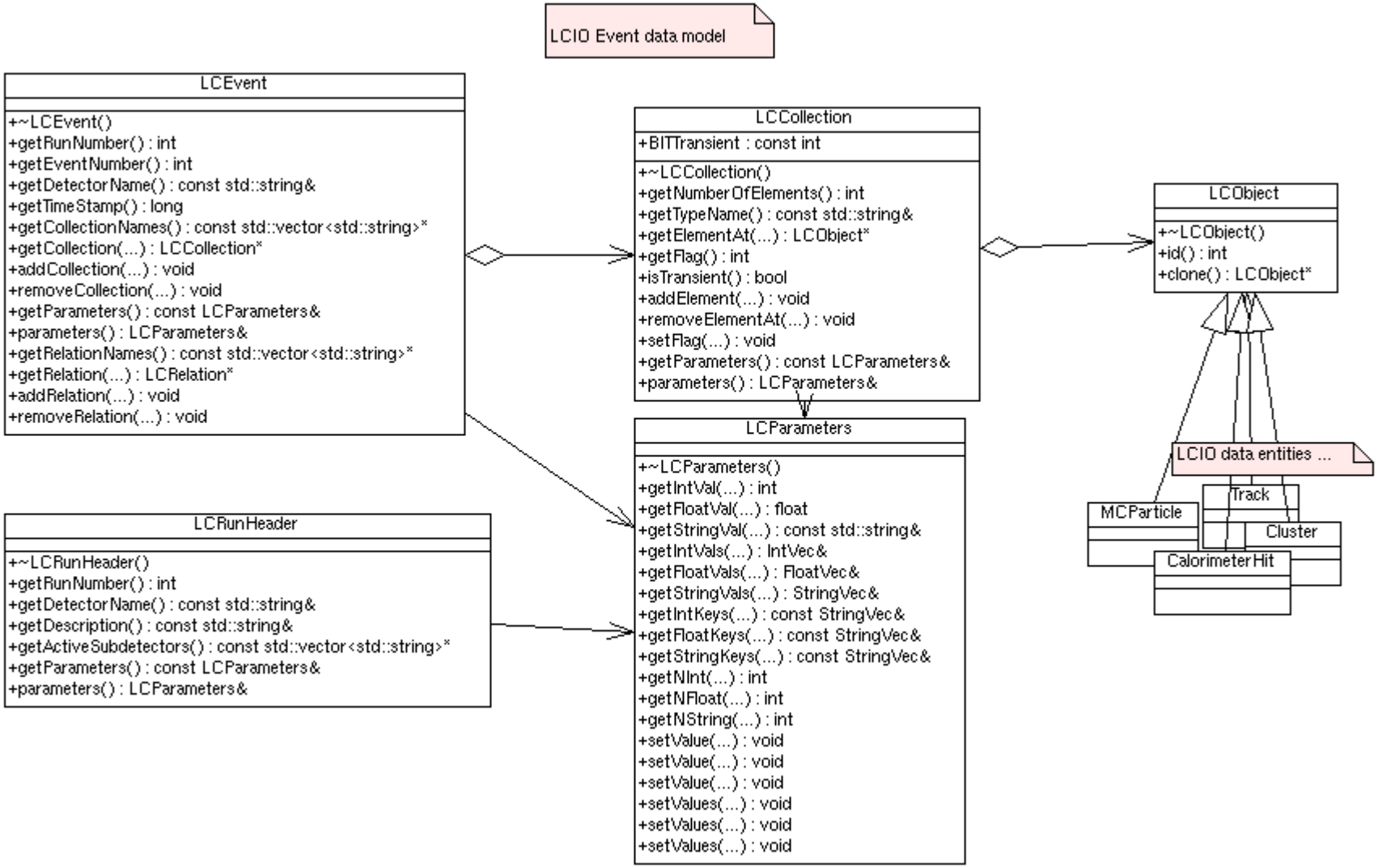
- The Mokka steering file (mokkaTrkSBelle_CPS1600_SUP10.steer)
  - Detector Model (TrkSBelle_CPS1600_SUP10)
  - lcio output filename
  - gear output filename
  - geant4 macro filename
- The geant4 macro (pgun2.0GeV80.g4)
  - particle gun
  - (or HEPEvt file containing particles & 4vectors)
  - directions
  - number of events to simulate

LCIO output file and *dumpevent*

- Contains LCCollections:
  - MCParticle (Information about the particles in the simulation)
  - VXDCollection (All simulated hits from PXD and SVD)
  - CDCCollection (All simulated hits in the CDC)
- Contains LCRelations in between individual LCCollections

- LCIO can be examined
  - with the program *dumpevent "lciofile" n*
    - where n is the event number (starting from 1)
  - or Marlin, JAS3, GeV or any program see LCIO examples

LCIO slide 3


LCIO Event data model
LCEvent
+~LCEvent()
+getRunNumber() : int
+getEventNumber() : int
+getDetectorName() : const std::string&
+getTimeStamp() : long
+getCollectionNames() : const std::vector<std::string>*
+getCollection(...) : LCCollection*
+addCollection(...) : void
+removeCollection(...) : void
+getParameters() : const LCParameters&
+parameters() : LCParameters&
+getRelationNames() : const std::vector<std::string>*
+getRelation(...) : LCRelation*
+addRelation(...) : void
+removeRelation(...) : void
LCCollection
+BITTransient : const int
+~LCCollection()
+getNumberOfElements() : int
+getTypeName() : const std::string&
+getElementAt(...) : LCObject*
+getFlag() : int
+isTransient() : bool
+addElement(...) : void
+removeElementAt(...) : void
+setFlag(...) : void
+getParameters() : const LCParameters&
+parameters() : LCParameters&
LCObject
+~LCObject()
+id() : int
+clone() : LCObject*
LCIO data entities ...
Track
MCParticle
Cluster
CalorimeterHit
LCRunHeader
+~LCRunHeader()
+getRunNumber() : int
+getDetectorName() : const std::string&
+getDescription() : const std::string&
+getActiveSubdetectors() : const std::vector<std::string>*
+getParameters() : const LCParameters&
+parameters() : LCParameters&
LCParameters
+~LCParameters()
+getIntVal(...) : int
+getFloatVal(...) : float
+getStringVal(...) : const std::string&
+getIntVals(...) : IntVec&
+getFloatVals(...) : FloatVec&
+getStringVals(...) : StringVec&
+getIntKeys(...) : const StringVec&
+getFloatKeys(...) : const StringVec&
+getStringKeys(...) : const StringVec&
+getNInt(...) : int
+getNFloat(...) : int
+getNString(...) : int
+setValue(...) : void
+setValue(...) : void
+setValue(...) : void
+setValues(...) : void
+setValues(...) : void
+setValues(...) : void

Marlin processor code 1
(MarlinVertexResidual)

create one instance of the processor

```
MarlinVertexResidual aMarlinVertexResidual;

MarlinVertexResidual::MarlinVertexResidual() :
	Processor("MarlinVertexResidual") {
	//Processor description
	_description = "writes out d0 and z0 residuals of the tracks";

	//Register steering parameters
	registerInputCollection(LCIO::TRACK, "TrackCollectionName",
		"Name of track collection of reconstructed particles",
		_trackColName, std::string("LDCTracks"));

	registerProcessorParameter("RootOutputFileName",
		"root file to output the Z0 and D0 residuals", _rootFileName,
		std::string("residual.root"));
}
```

processor name

info text for documentation

default values

member variable

Marlin processor code 2
(Easy ROOT integration)

```
void MarlinVertexResidual::init() {
        _nRun = 0,  _nEvt = 0;

        _RootFile = new TFile(_rootFileName.c_str(), "RECREATE");
        _RootFile->cd("");
        _RootTree = new TTree("MarlinTrackTree", "Residuals of the track");
        _RootTree->Branch("D0", &D0, "D0/D");
        _RootTree->Branch("Z0", &Z0, "Z0/D");
}
void MarlinVertexResidual::processEvent(LCEvent* evt) {
        LCCollection * col = 0;
        try {
                col = evt->getCollection(_trackColName.c_str());
        } catch (DataNotAvailableException &e) {}

        Track * trk = dynamic_cast<Track*> (col->getElementAt(0));
        D0 = trk->getD0();
        Z0 = trk->getZ0();
        _RootTree->Fill();
}
void MarlinVertexResidual::end() {
        _RootFile->cd("");
        _RootFile->Write();
        _RootFile->Close();
}
```

open ROOT file

get collection from LCIO

get track(s) from collection

fill data in ROOT file

close ROOT file

My Marlin steering file 1 (marlinTrkSBelle_CPS1600_SUP10.xml)

```
 1  <marlin>
 2   <!-- Execute following processors -->
 3   <execute>
 4     <processor name="MyMaterialDB"/>
 5     <processor name="MyVTXDigitizer"/>
 6     <processor name="MyCDCDigiProcessor"/>
 9     <processor name="MyLEPTrackingProcessor"/>
10     <processor name="MySiliconTracking"/>
11     <processor name="MyFullLDCTracking"/>
14     <processor name="MyMarlinVertexResidual" />
15   </execute>

16   <global>
17     <!-- LCIO input files  -->
18     <parameter name="LCIOInputFiles"> TrkSBelle_CPS1600_SUP10.slcio </parameter>
19     <!--  GEAR input files -->
20     <parameter name="GearXMLFile"> gearTrkSBelle_CPS1600_SUP10.xml </parameter>
21
22     <!-- OTHER parameters  -->
23     <parameter name="Verbosity"> SILENT </parameter>
24     <parameter name="MaxRecordNumber" value="1001" />
25     <parameter name="SupressCheck" value="false" />
26     <parameter name="SkipNEvents" value="0" />
27   </global>
```

My Marlin steering file 2
(marlinTrkSBelle_CPS1600_SUP10.xml)

```
<processor name="MyCDCDigiProcessor" type="TPCDigiProcessor">
 <!--Produces TPC TrackerHit collection from SimTrackerHit collection, smeared in RPhi and Z-->
  <!--Name of the SimTrackerHit collection-->
  <parameter name="CollectionName" type="string"> CDCCollection </parameter>
  <!--Name of the digitized TrackerHit collection-->
  <parameter name="TPCTrackerHitsCol" type="string"> AllCDCTrackerHits </parameter>
 </processor>


<processor name="MyMarlinVertexResidual" type="MarlinVertexResidual">
  <parameter name="TrackCollection" type="string" lcioInType="Track"> LDCTracks </parameter>
  <parameter name="RootOutputFileName" type="string"
                                                    value="TrkSBelle_CPS1600_SUP10_2.0GeV_80.root"/>
</processor>
```