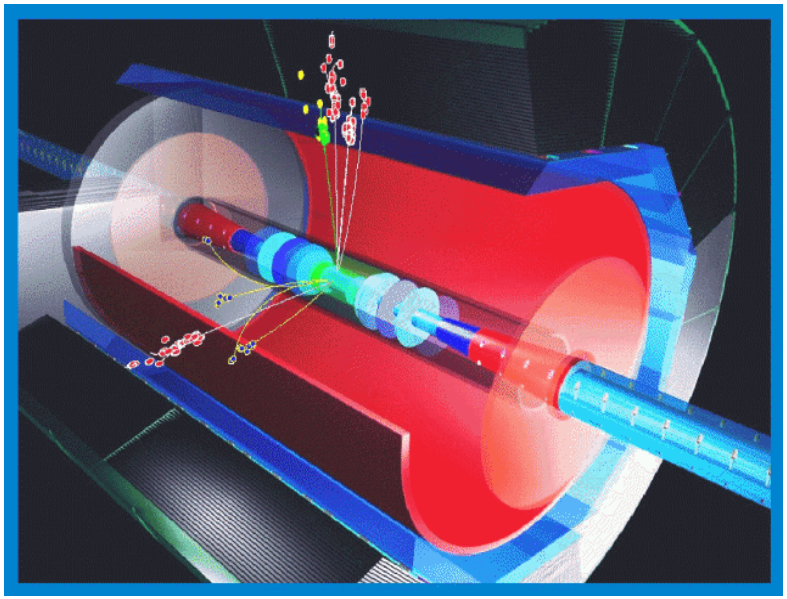
Simulation and Reconstruction Software for the ILC

Frank Gaede
DESY

FNAL ILC Computing Seminar
September 25, 2006

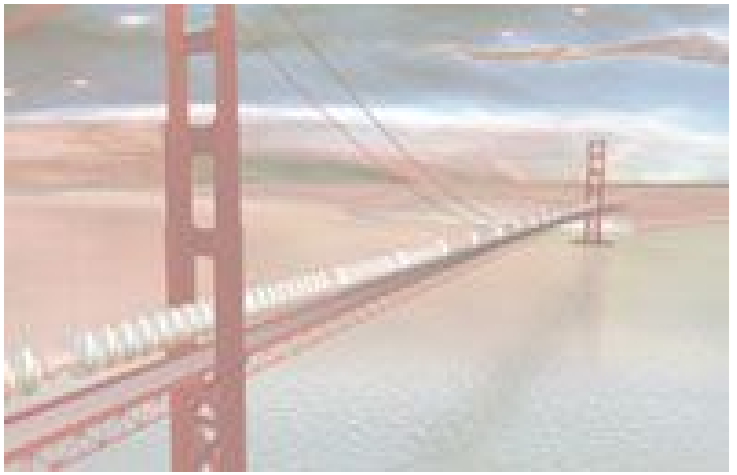
Outline

- Introduction to ILC
- Overview international software for the ILC
- LDC Software Tools
  - **LCIO** – data model & persistency
  - **Mokka** – geant4 full simulation
  - **Marlin** – C++ framework
  - **MarlinReco** – reconstruction toolkit
  - **LCCD** - conditions data toolkit
  - **GEAR** – geometry description
- Interoperability
- Summary/Outlook



ILC Overview I

- future linear collider (e+/e-) project
  - >30km length
  - 1stage
    - energy 200→500 GeV, scannable
    - 500 fb-1in first 4 years
  - 2nd stage
    - energy upgrade to ~1TeV
    - ~1000 fb-1in 3-4 years
  - options
    - gama/gamma, gamma/e-, e-/e-, Giga-Z
    - 2 IRs for 2 experiments ?

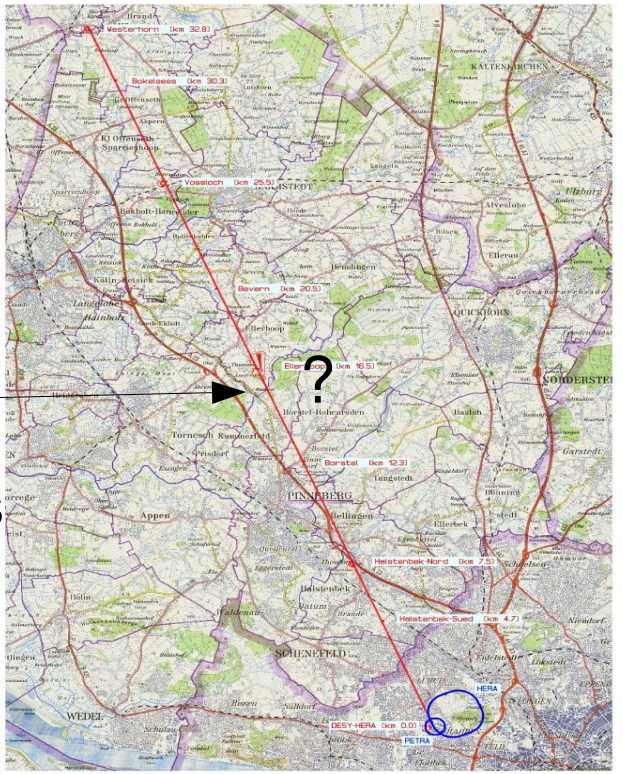


ILC Overview II

- originally three regional projects (~1991(!?)-2004)
  - America: NLC Asia: JLC Europe: Tesla
- now: International Linear Collider - **ILC**
  - ITRP technology decision (Aug 2004):

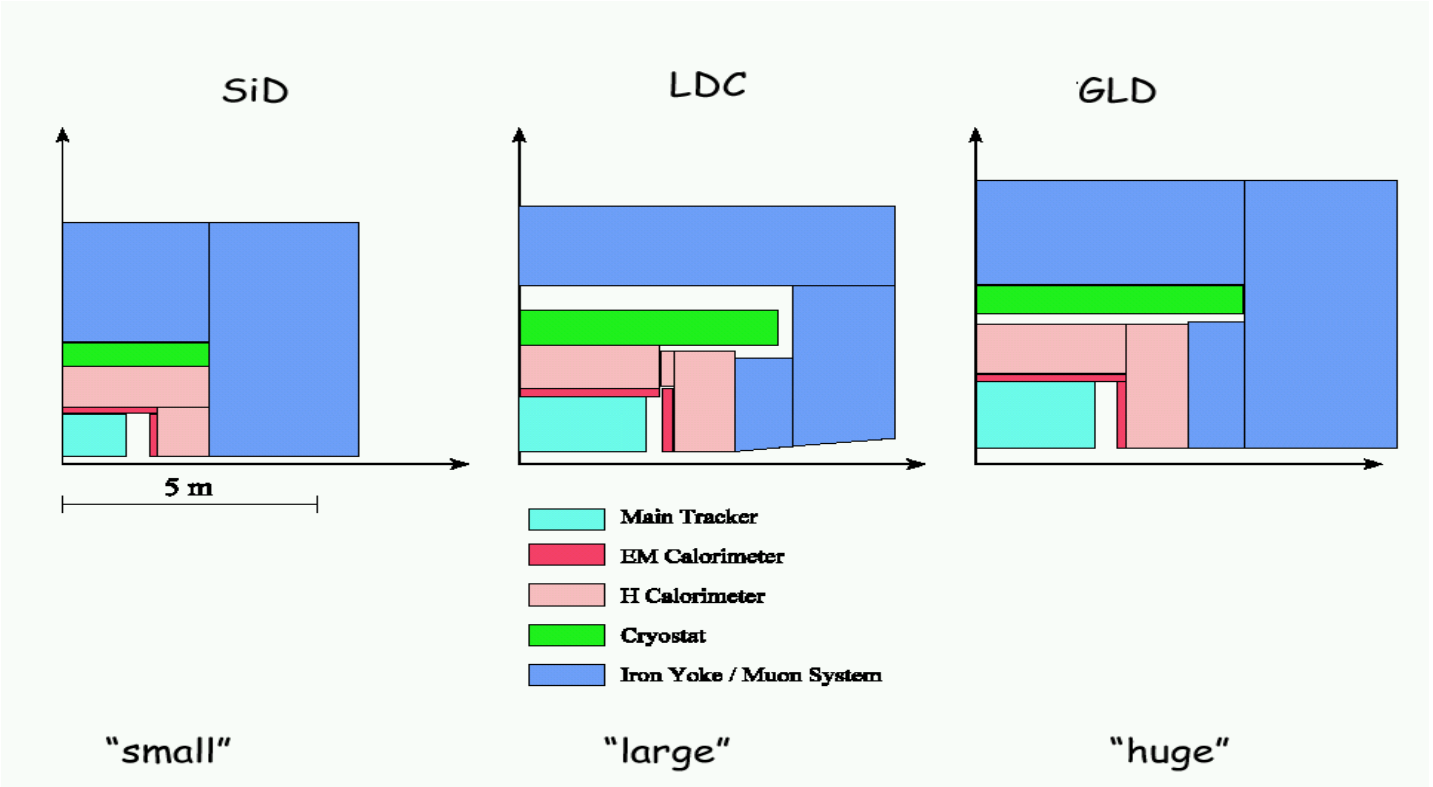
    "We recommend that LC be based on super-conducting RF technology."
- rough timeline (planned):
  - (2005) Accelerator CDR
  - (2007) Accelerator TDR – Detector CDRs

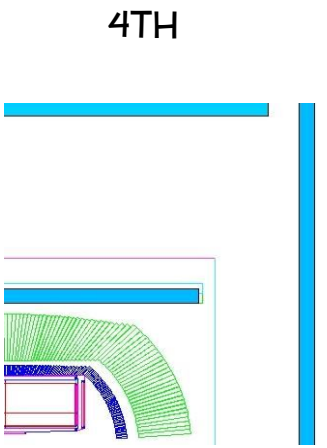
    from concept study (next slide)
  - (2008/09?) LC site selection
  - (2009/10?) Global lab selects experiments
  - (?) Start construction
  - (??) Data



ILC Detector Concept Studies

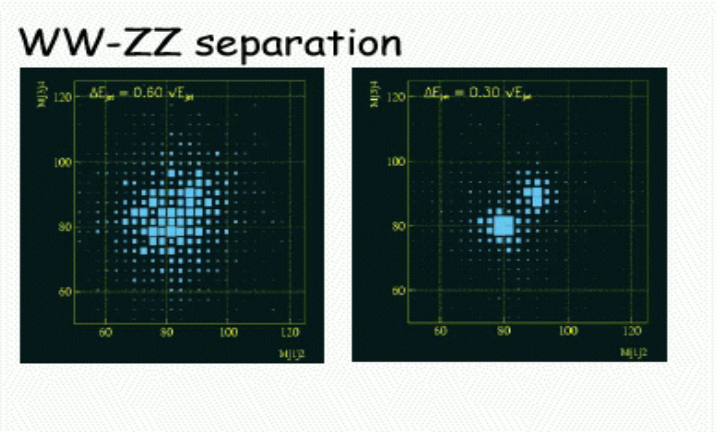
- WWS (WorldWide Study on Physics and Detector for Future Linear Colliders) :
  - coordinate studies on four interregional detector concepts, and work toward detector TDRs (2007/2008):
  - currently ongoing: work on DODs





Reconstruction @ the ILC

- general ILC detector features:
  - precision tracking
  - precision vertexing
  - high granularity in calorimeters
    - (Ecal ~1cm, Hcal ~1-5cm)
- important: very high jet-mass resolution ~30%/sqrt(E/GeV)



Particle Flow

- reconstruct all single particles
- use tracker for charged particles
- use Ecal for photons
- use Hcal for neutral hadrons

why develop reconstruction sw now ?

- reconstruction algorithms (pflow) are part of the overall detector performance
- need full reconstruction to optimize detector concepts

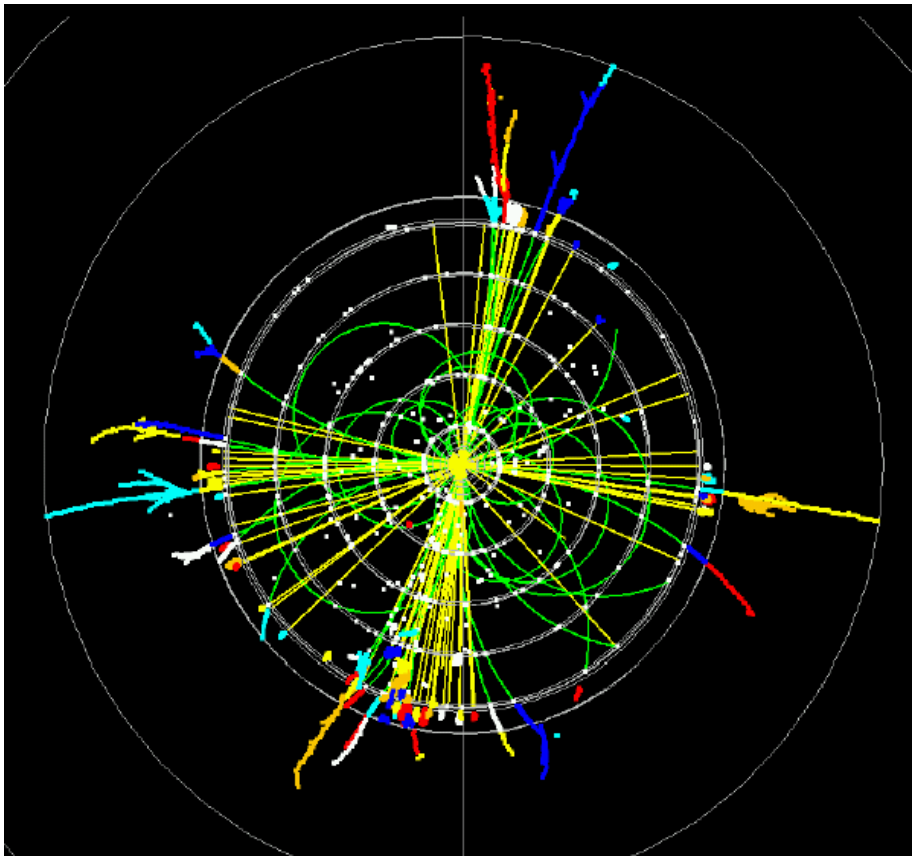
$$\sigma_{E_{jet}}^2 = \epsilon_{trk}^2 \sum_i E_{trk,i}^4 + \epsilon_{ECal}^2 E_{ECal} + \epsilon_{HCal}^2 E_{HCal} + \sigma_{confusion}^2$$

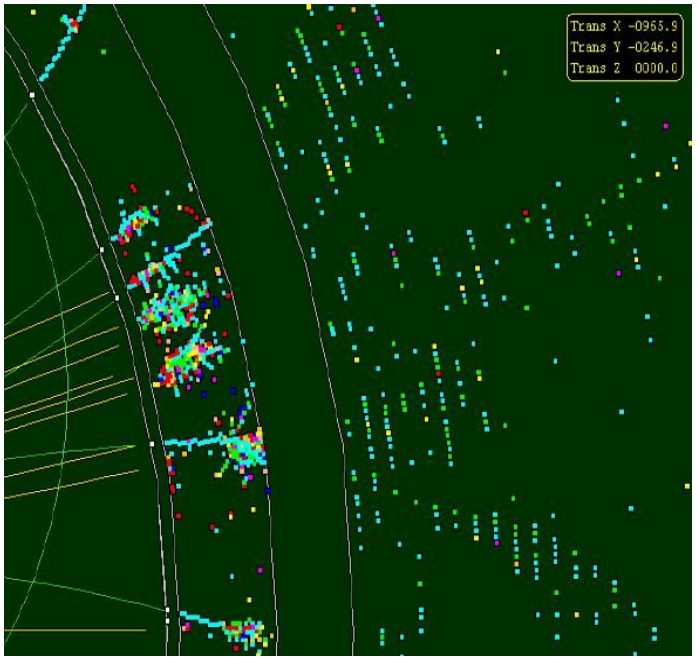
$$\epsilon_{trk} = \delta(1/p) \approx 5 \cdot 10^{-5}, \quad \epsilon_{ECal} = \frac{\delta E}{\sqrt{E}} \approx 0.1, \quad \epsilon_{HCal} \approx 0.5$$

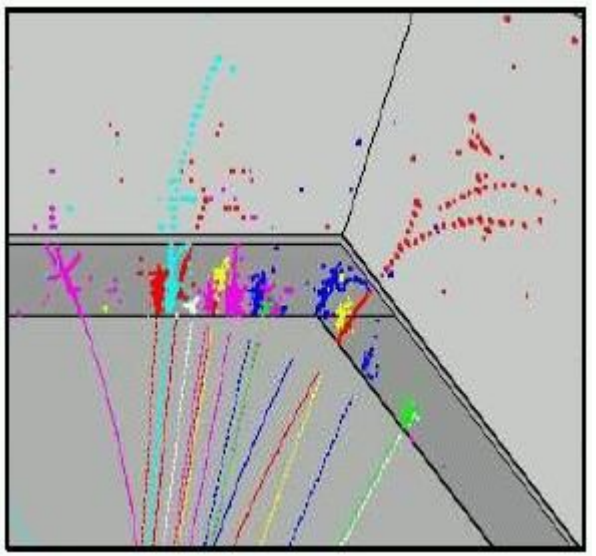
dominant contribution (E<50 GeV):

- Hcal resolution
- confusion term

Imaging calorimeter & ParticleFlow







Need software tools to improve clustering & pflow:
- detailed hadronic shower simulation (**Geant4**)
- framework for developing and comparing pflow algorithms

Note: precision tracking and vertexing also very challenging and important !

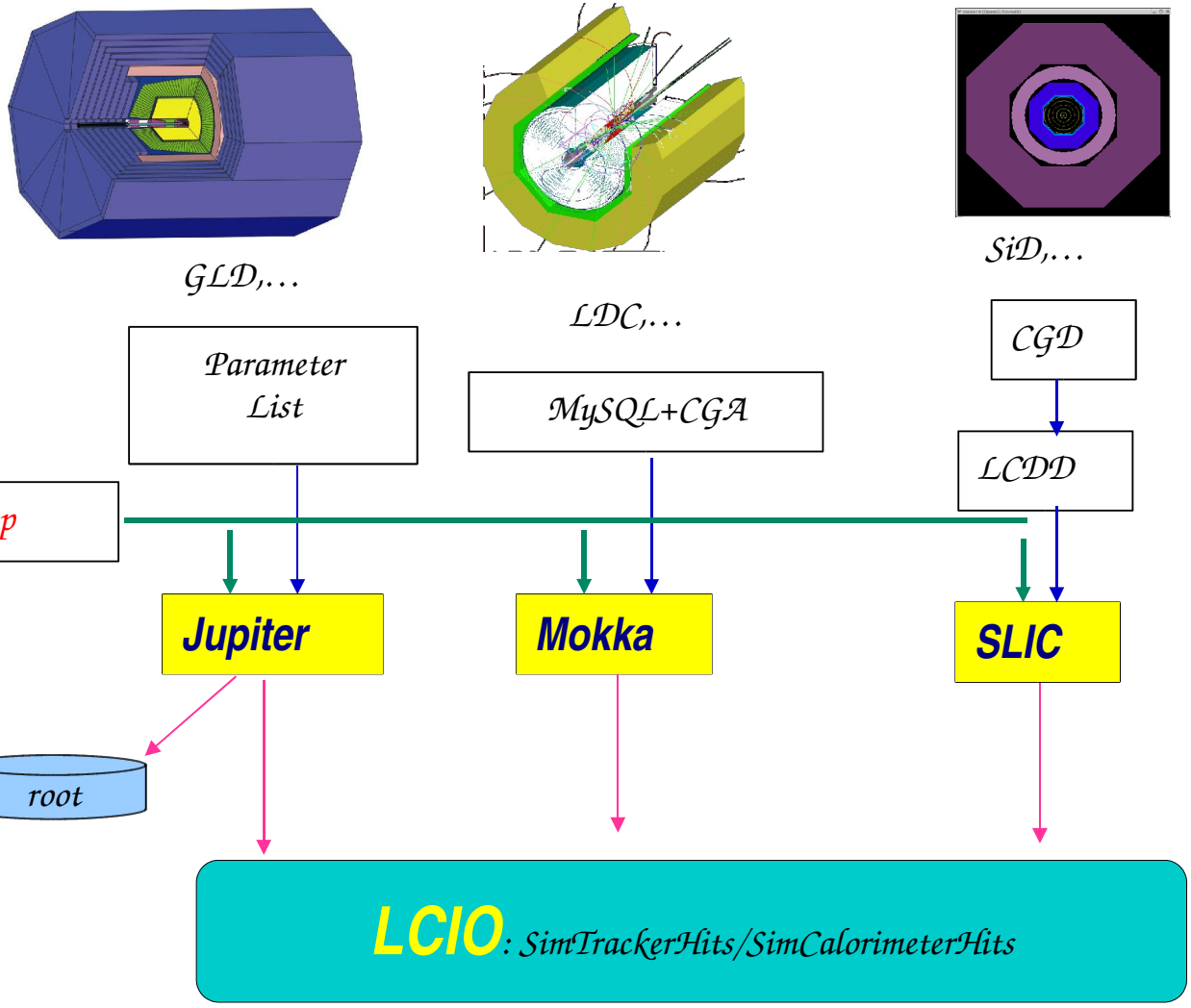
ILC software packages

	Description	Detector	Language	IO-Format	Region
Simdet	fast Monte Carlo	TeslaTDR	Fortran	StdHep/LCIO	EU
SGV	fast Monte Carlo	simple Geometry, flexible	Fortran	None (LCIO)	EU
Lelaps	fast Monte Carlo	SiD, flexible	C++	SIO, LCIO	US
Mokka	full simulation – Geant4	TeslaTDR, LDC, flexible	C++	ASCI, LCIO	EU
SLIC	full simulation – Geant4	SiD, flexible	C++	LCIO	US
Jupiter	full simulation – Geant4	JLD (GDL)	C++	Root (LCIO)	AS
Marlin	reconstruction and analysis application framework	Flexible	C++	LCIO	EU
org.lcsim	reconstruction framework (under development)	SiD (flexible)	Java	LCIO	US
Jupiter-Satelite	reconstruction and analysis	JLD (GDL)	C++	Root	AS
LCCD	Conditions Data Toolkit	All	C++	MySQL, LCIO	EU
LCDD	Geometry description	SiD (flexible)	Java/C++	XM/GDML	US
GEAR	Geometry description	Flexible	C++ (Java?)	XML	EU
LCIO	Persistency and datamodel	All	Java, C++, Fortran	-	AS,EU,US
JAS3/WIRED	Analysis Tool / Event Display	All	Java	xml,stdhep, heprep,LCIO,	US,EU

not included: proposed ILCroot/Aliroot

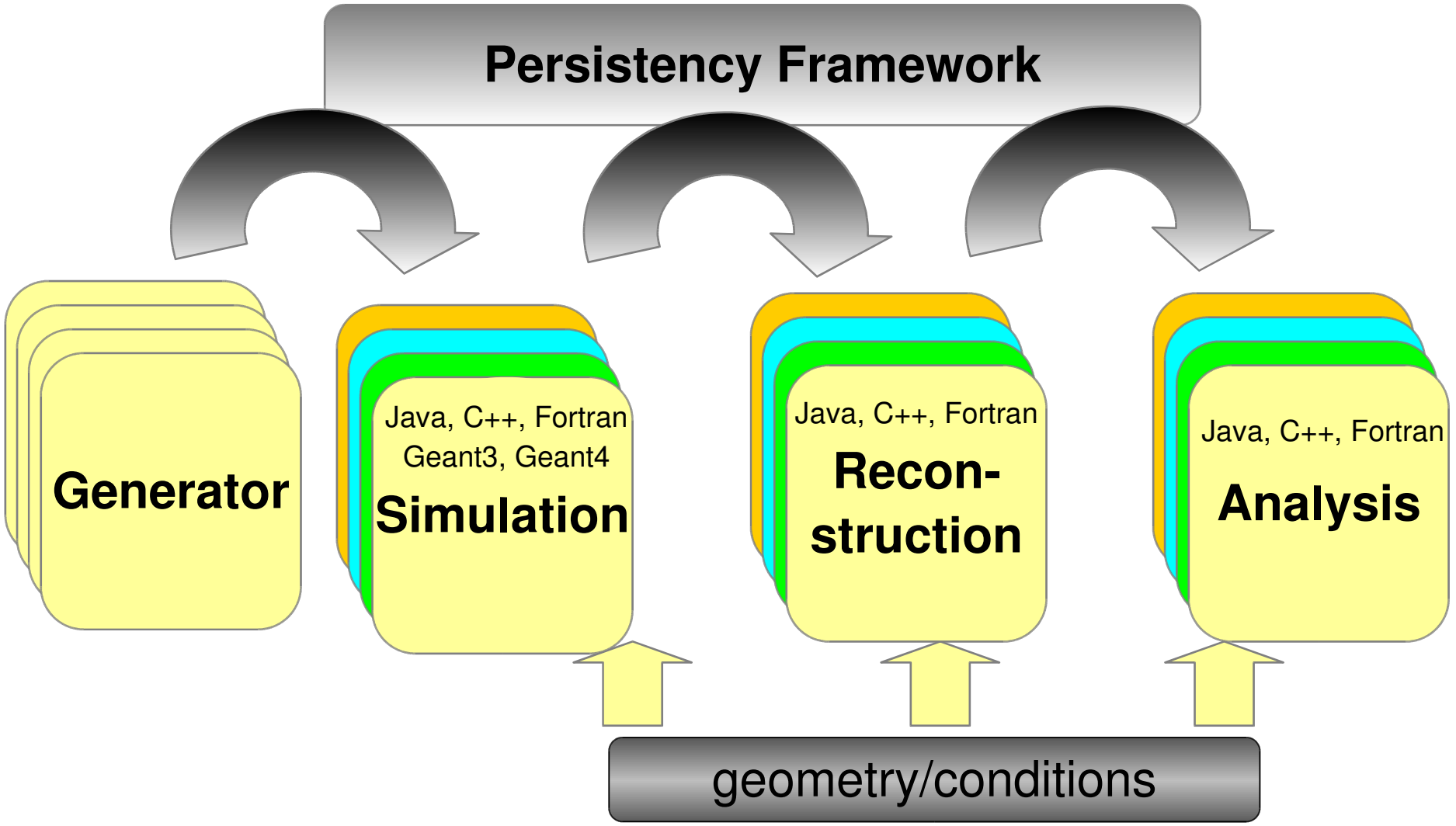
ILC Simulation Frameworks (Geant4)

- *Geant4, StdHep and LCIO are common feature*
- *Each trying to be generic with different approach*
  different ways to define geometries

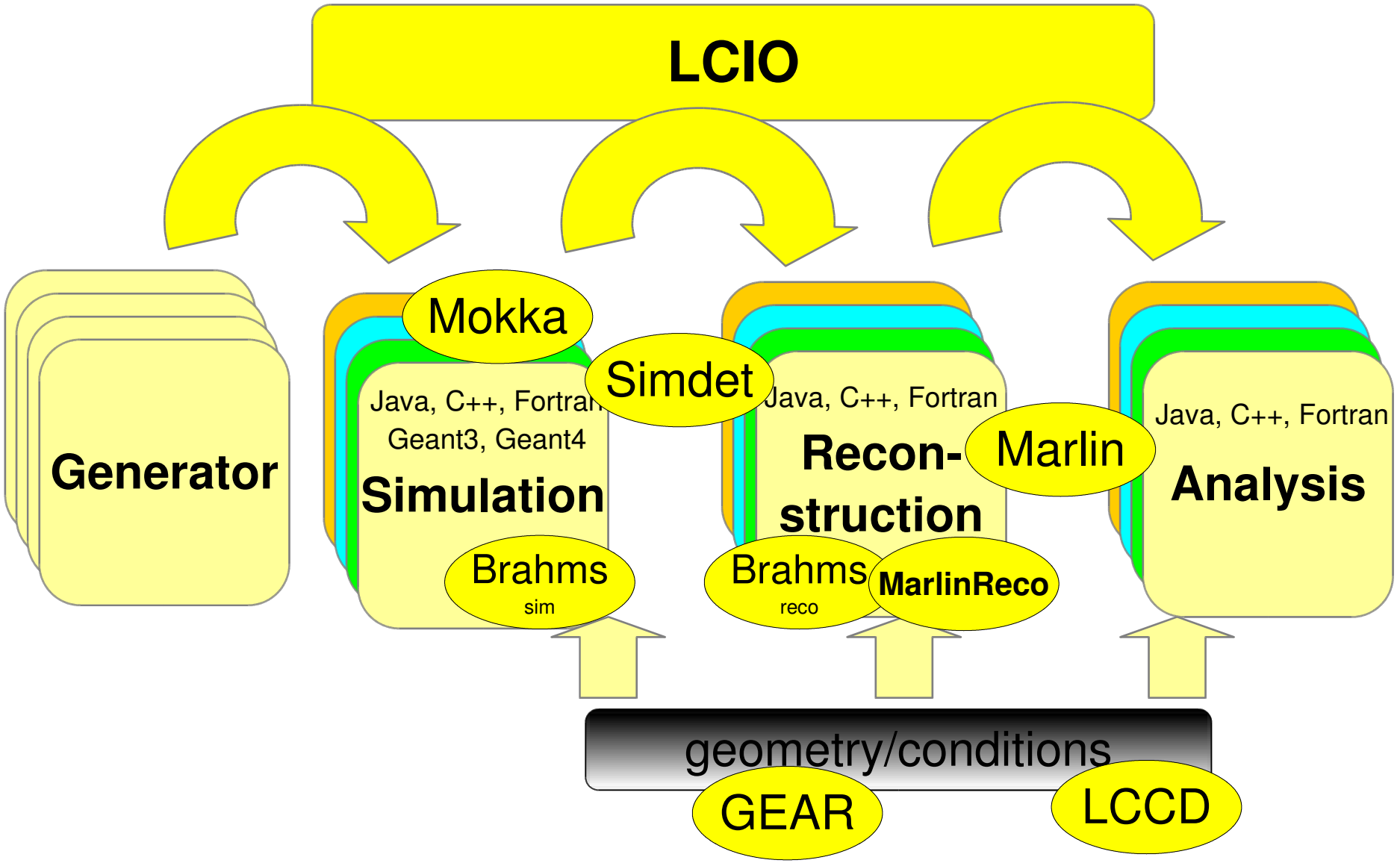


(A.Miyamoto)

ILC software chain



ILC software tools used for LDC



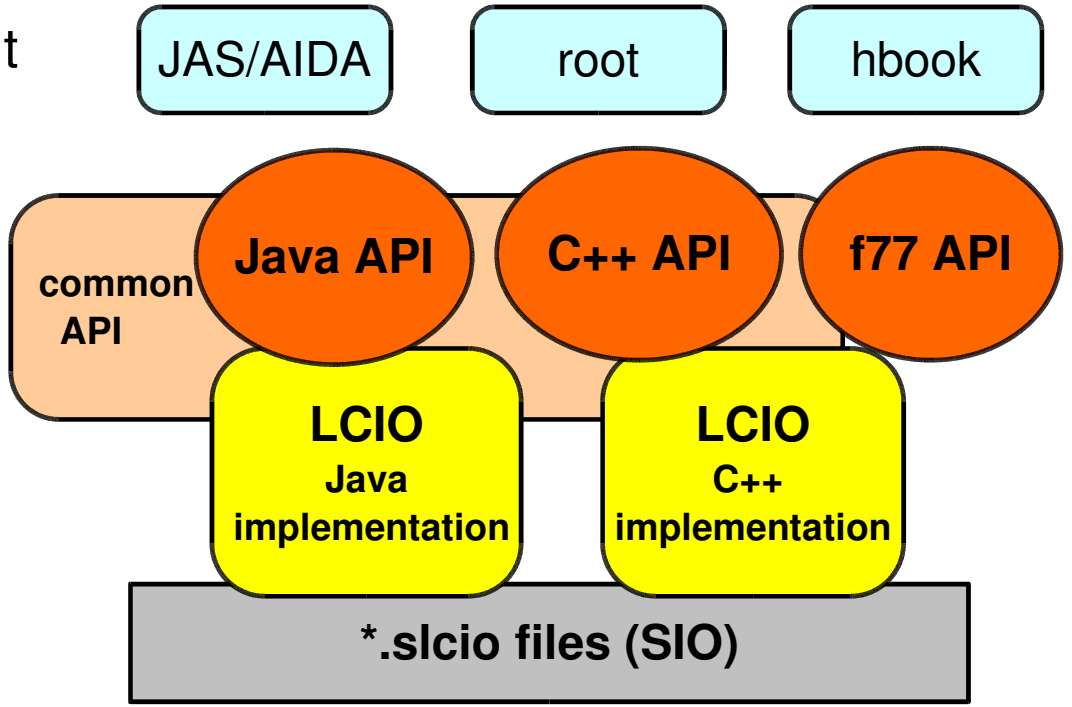
LCIO overview

- DESY and SLAC joined project:
  - provide common basis for ILC software
- Features:
  - Java, C++ and f77 (!) API
  - extensible data model for current and future simulation and testbeam studies
  - user code separated from concrete data format
  - no dependency on other frameworks

simple & lightweight

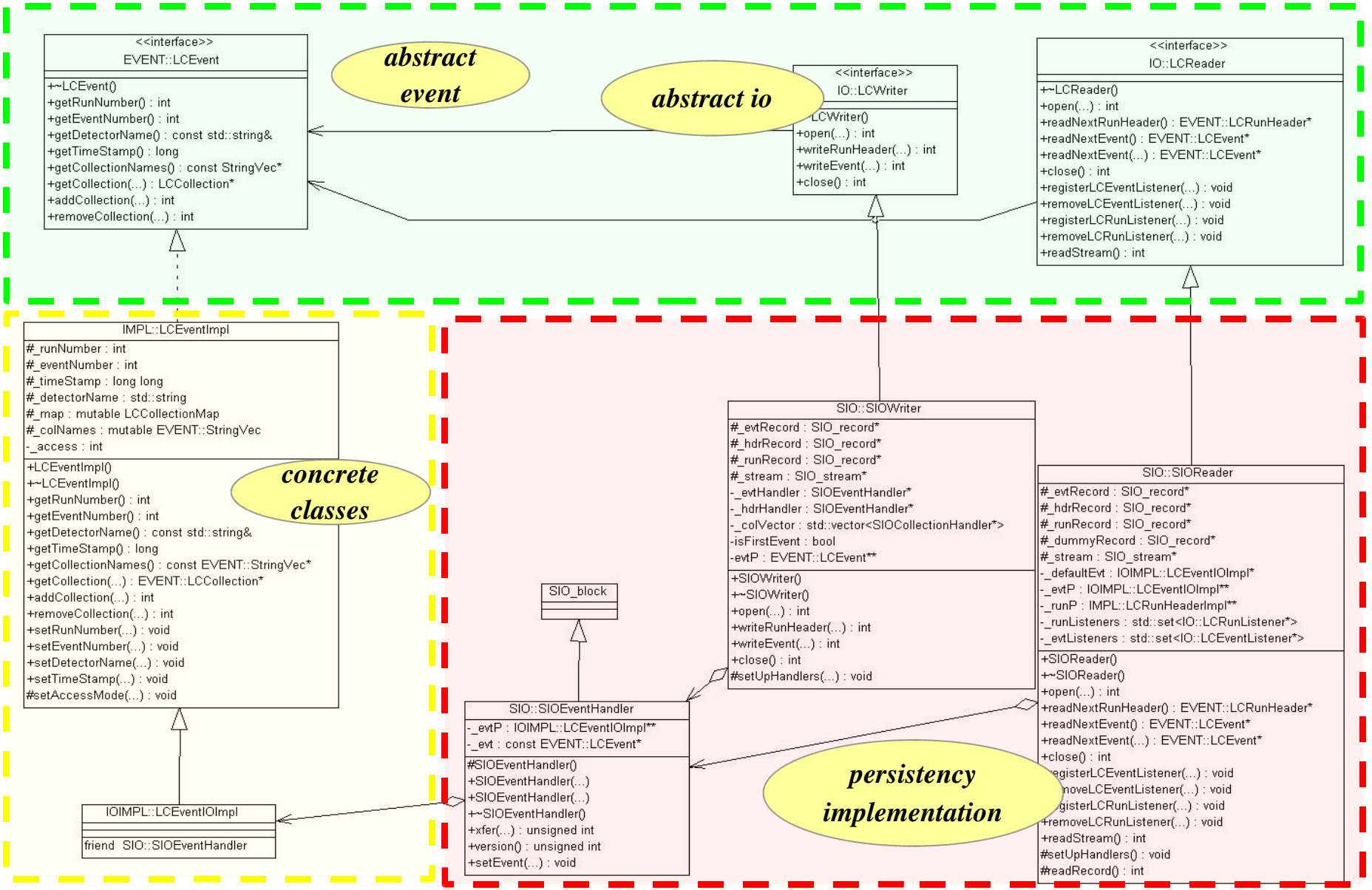
now de facto standard persistency & datamodel for ILC software

SW-Architecture

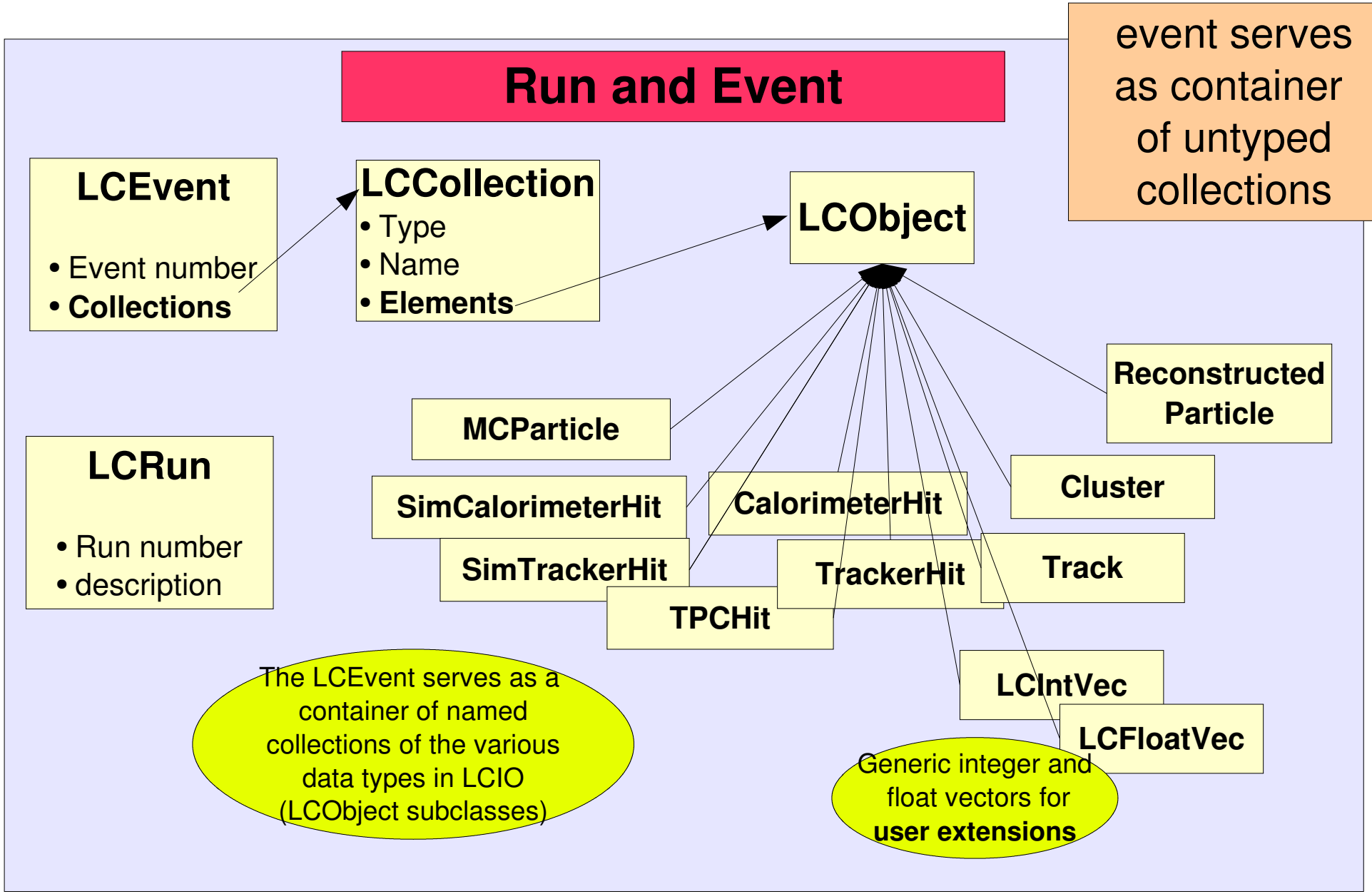


current release: **v01-07**

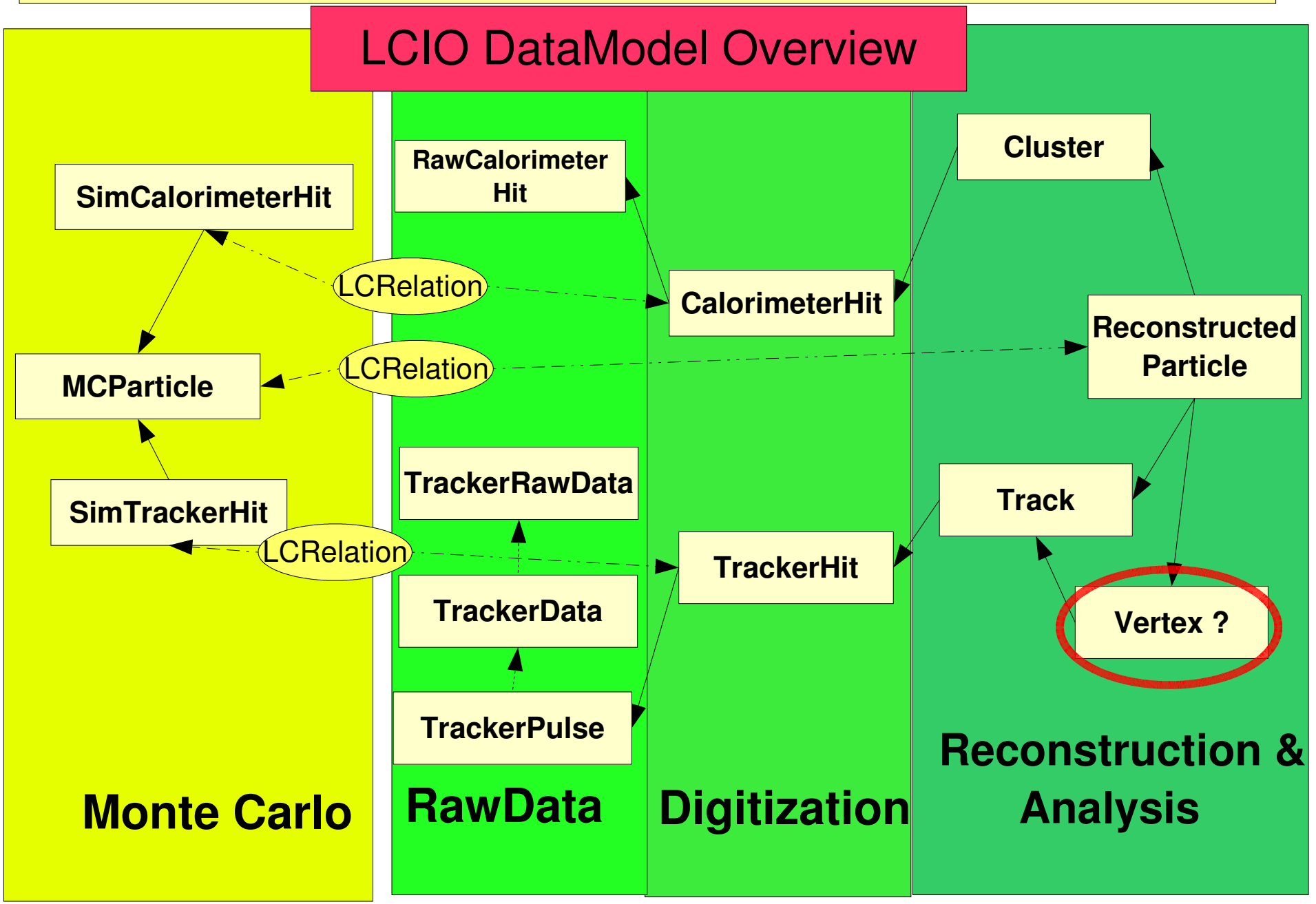
LCIO class design



LCIO Event Data Model I

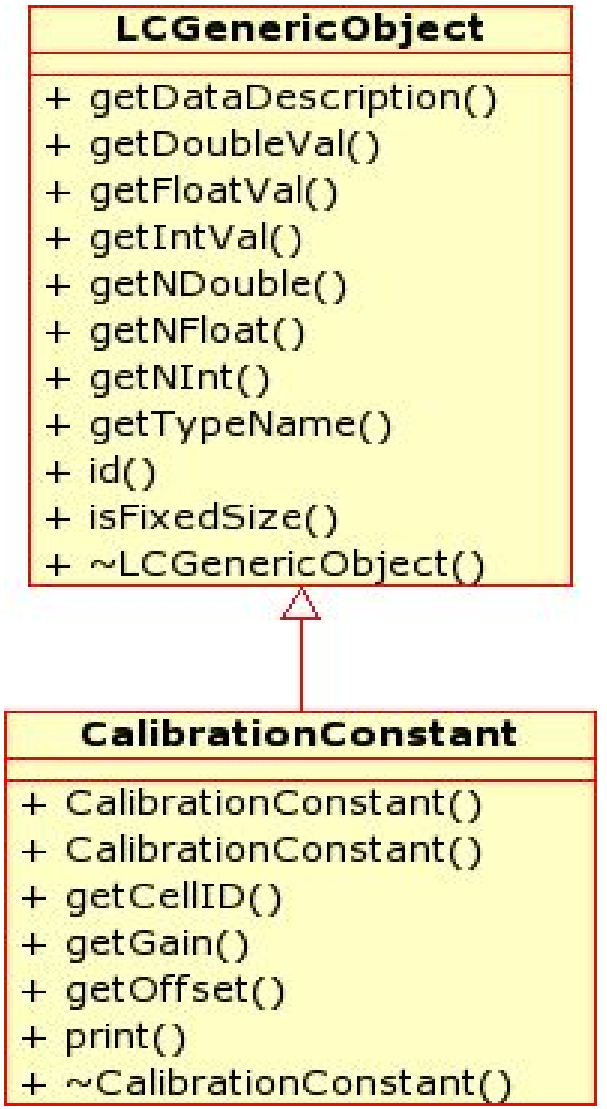


LCIO Event Data Model II



LCIO – Generic User Information

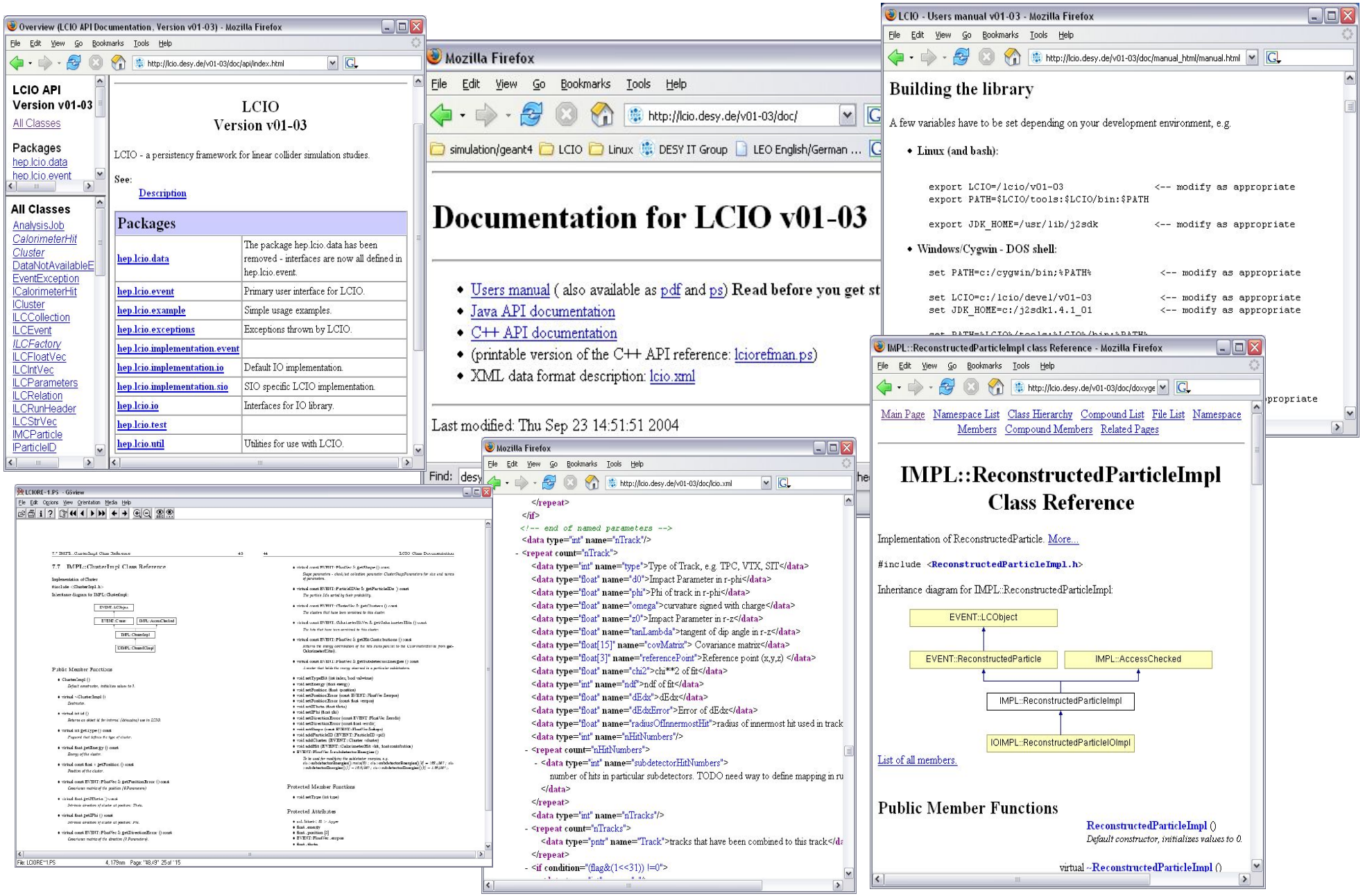
- LCIO data model defines the object needed for ILC simulation studies, but
  - users want additional information in files for specific studies
  - can't create new classes within LCIO for all requests and purposes
  - need generic user class: **LCGenericObject**
    - almost arbitrary data objects
    - typically access provided through user subclass - but not needed:
    - has description string for reading the data without need to have access to data dictionary (library)



LCIO Event Data Model III

- the LCIO event data model is fairly complete and flexible
- however it is adapted and extended as needed by the community
  - maintaining downward compatibility
  - with international discussion and agreement
  - example: introduction of a new **Vertex** class in LCIO
    - originally proposed by LCFI group
    - see discussion @ http://forum.linearcollider.org/
    - new raw data classes for prototypes

LCIO Online documentation



Simulation tools: Simdet, Brahms

- **SIMDET**
  - parameterized fast Monte Carlo (f77)
  - tracks + cov. matrix and clusters
  - hard coded geometry: TESLA TDR Detector

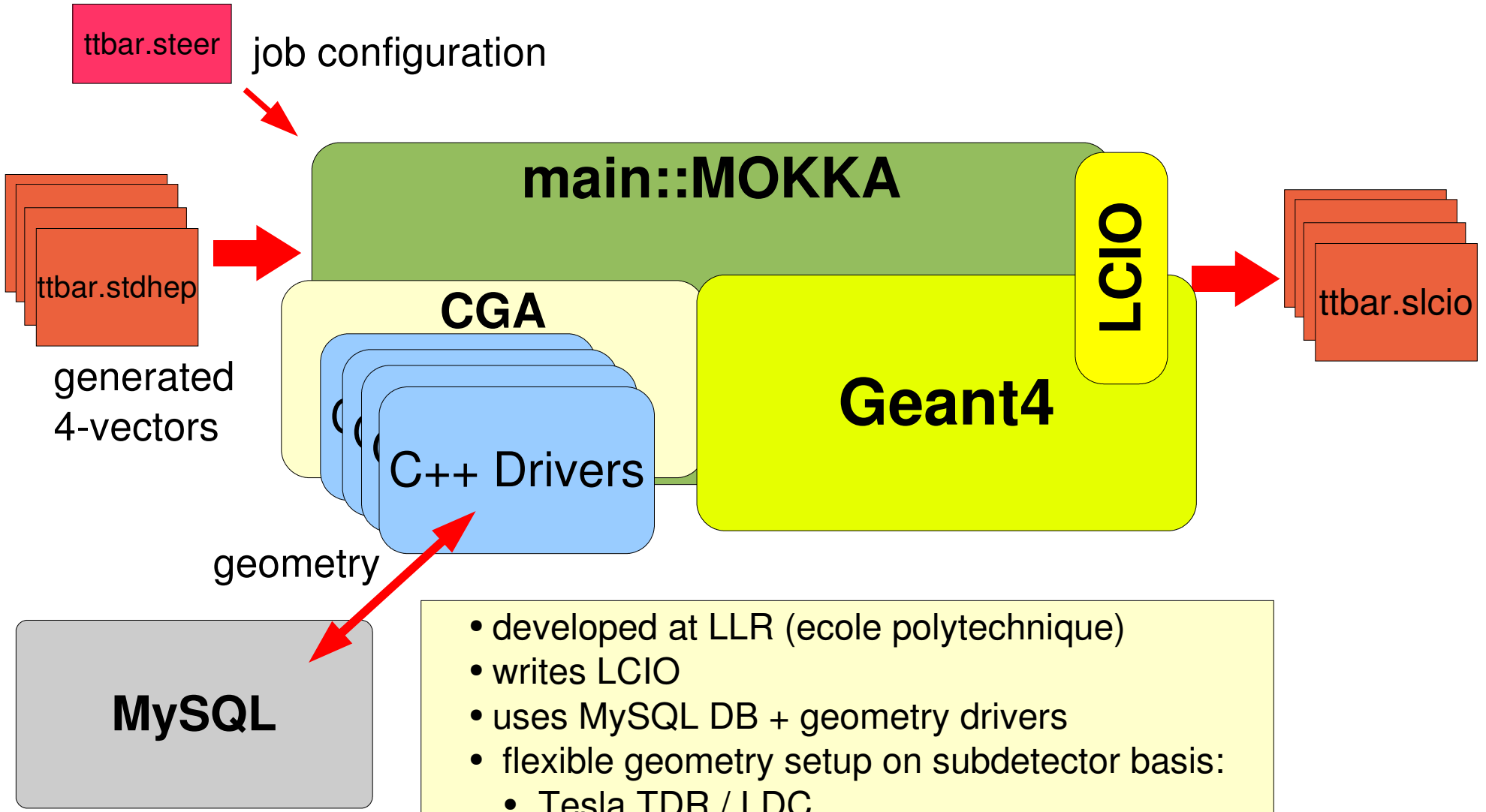
writes LCIO

- **Brahms**
  - geant3 full simulation (f77)
  - hard coded geometry: TESLA TDR Detector
  - full standalone reconstruction part (pflow)
    - tracking based on LEP reconstruction code

reads + writes LCIO

for download (cvs web interface)
and more information:
http://www-zeuthen.desy.de/linear_collider

Mokka overview

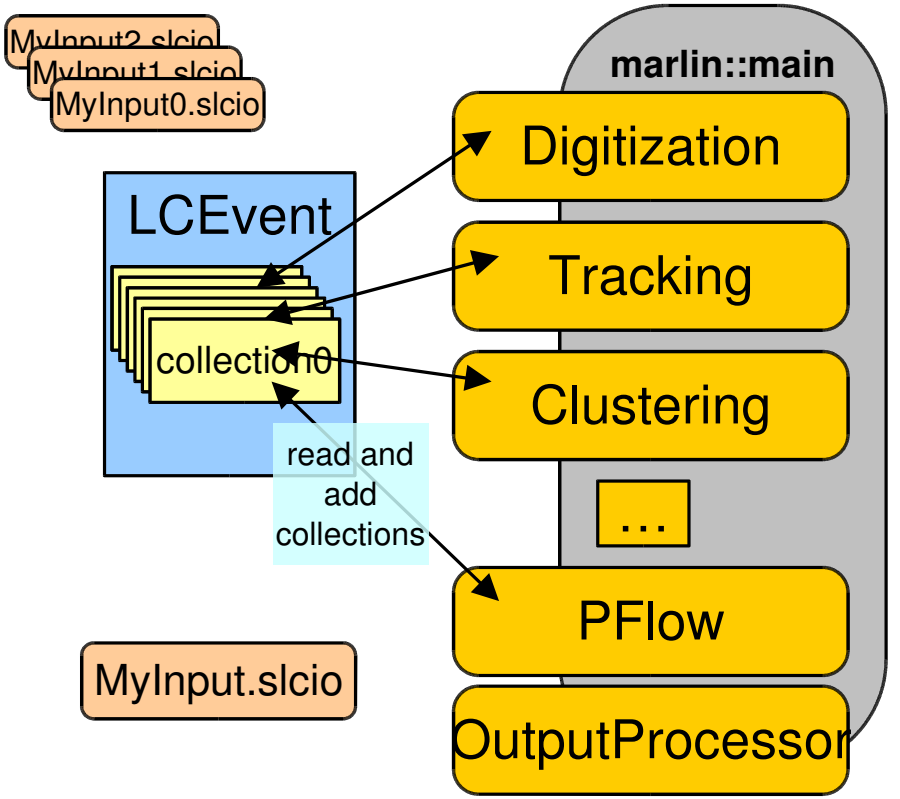


- developed at LLR (ecole polytechnique)
- writes LCIO
- uses MySQL DB + geometry drivers
- flexible geometry setup on subdetector basis:
  - Tesla TDR / LDC
  - SiD
  - Calice testbeam prototypes
- used in European/ LDC study

Marlin

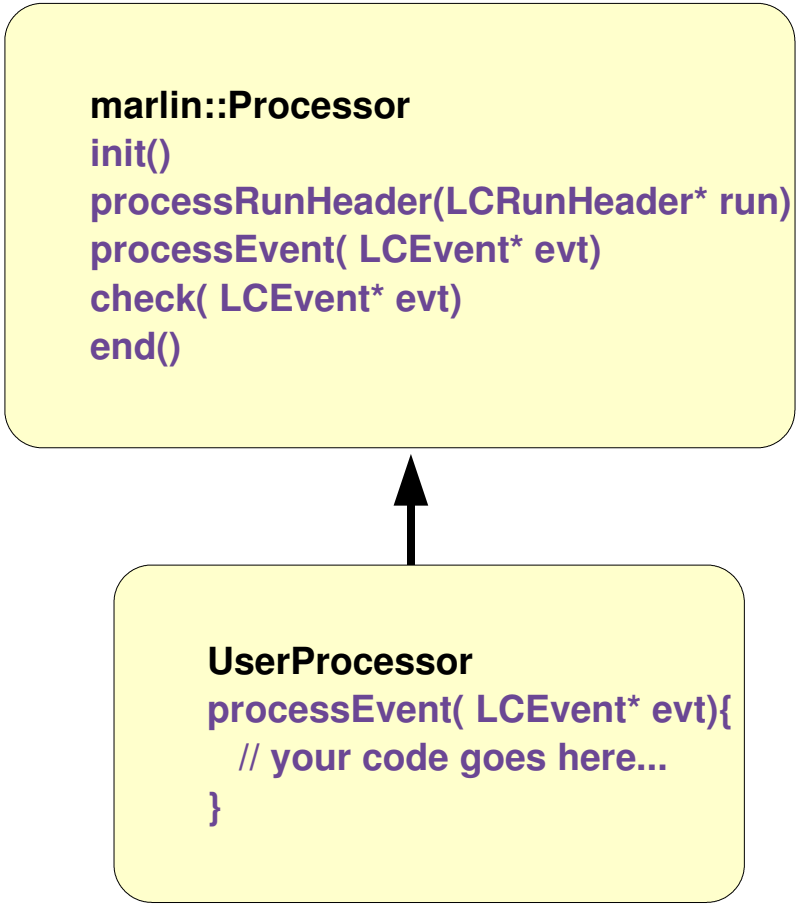
Modular**A**nalysis & **R**econstruction for the **L I N**ear Collider

- modular C++ **application framework** for the analysis and reconstruction of LCIO data
- uses LCIO as transient data model
- software modules called Processors
- provides main program !
- provides simple user steering:
  - program flow (active processors)
  - user defined variables
    - per processor and global
  - input/output files
  - **Plug&Play** of processors



Marlin Processor

- provides main user callbacks
- has own set of input parameters
  - int, float, string (single and arrays)
  - parameter description
- naturally modularizes the application
- order of processors is defined via steering file:
  - easy to exchange one or several modules w/o recompiling
  - can run the same processor with different parameter set in one job
- processor task can be as simple as creating one histogram or as complex as track finding and fitting in the central tracker



Marlin core features

- fully configurable through steering files:
  - program flow
  - input parameters (processor based and global)
- self-documenting:
  - ./bin/Marlin -x
    prints example steering file with
    all available processors with their parameters and example/default values
- AIDA interface for histogramming
  - easy creation of histograms through absract interface
  - AIDAJNI/JAIDA, RAIDA (root based), ...
- configurable output
  - drop collections by name/type
- simple examples
  - user processor template, GNUmakefile,...
- easily extensible
  - makefiles 'automatically' include user packages with processors

Marlin – XML steering files

```
- <marlin>
  - <execute>
      <processor name="MyAIDAProcessor"/>
      <processor name="MyEventSelection"/>
    - <if condition="MyEventSelection">
        <group name="Tracking"/>
        <processor name="MyClustering"/>
        <processor name="MyPFlow"/>
        <processor name="MyLCIOOutputProcessor"/>
      </if>
    </execute>
  - <global>
      <parameter name="LCIOInputFiles"> simjob.slcio </parameter>
      <parameter name="MaxRecordNumber" value="5001"/>
      <parameter name="SupressCheck" value="false"/>
    </global>
  - <processor name="MyLCIOOutputProcessor" type="LCIOOutputProcessor">
      <parameter name="LCIOOutputFile" type="string">outputfile.slcio </parameter>
      <parameter name="LCIOWriteMode" type="string">WRITE_NEW</parameter>
    </processor>
  - <group name="Tracking">
      <parameter name="NTPCLayers" value="200"/>
      <processor name="MyTrackfinder" type="Trackfinder"/>
    - <processor name="MyTrackfitter" type="Trackfitter">
        <parameter name="Algorithm" value="DAF"/>
      </processor>
    </group>
    <!-- ... -->
  </marlin>
```

- Program flow defined in <execute>...</execute> section
- logical conditions from parameters evaluated at runtime

- global Parameters defined in <global/> section

- local Parameters defined in mandatory <parameter/> section

- Processors can be enclosed by <group/> tag
- Parameters in <group/> joined by all processors

a Marlin application is fully configured through the steering files (no user main program) !!

Marlin Status and Plans

- current version: v00-09-05:
- some new features:
  - made compatible with CLHEP 2.x
  - made compatible with RAIDA
  - split outputfiles wrt. size: ttbar_000.slcio, ttbar_001.slcio,...
  - bug fixes
- plans:
  - improve dealing with and creation of steering files
    - show parameters in order specfied
    - provide minimal steering files
    - provide help for one single processor
  - user suggestions/ needs ?

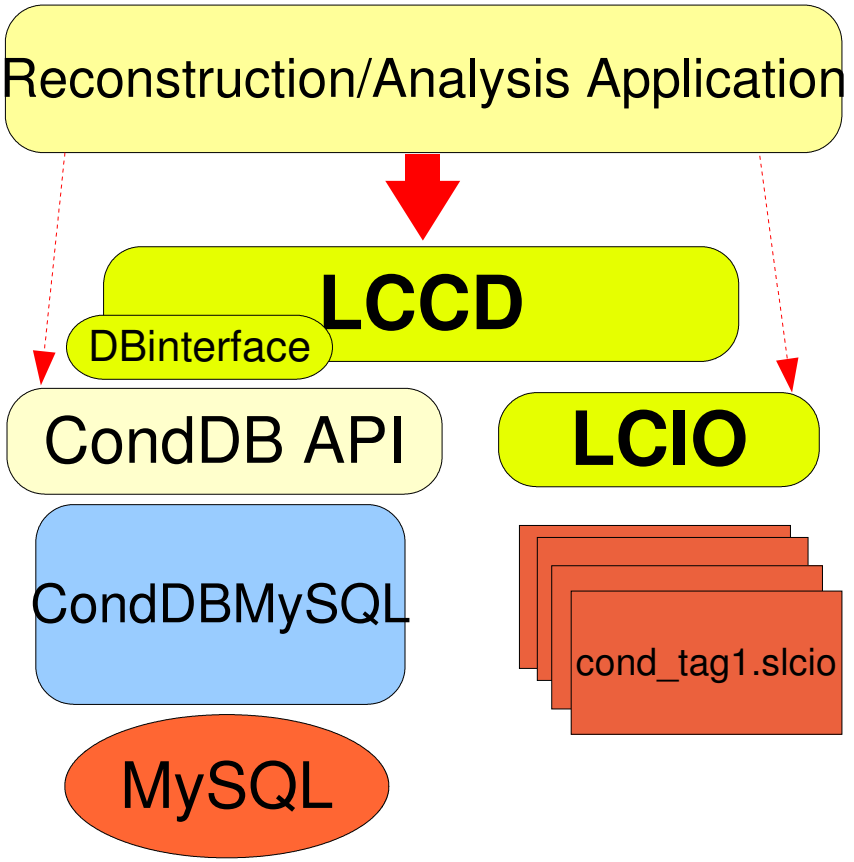
Marlin for reconstruction

- Marlin is a generic framework for processing of ILC data (LCIO)
  - simple and lightweight, depends on LCIO only
- some ingredients for typical reconstruction are missing:
  - description of detector geometry
  - conditions data (testbeam)
  - utility software, math libraries, ...
- optionally Marlin supports additional tools:
  - **GEAR** – geometry description
  - **LCCD** – conditions data
  - MarlinUtil – utility library

LCCD

Linear **C**ollider **C**onditions **D**ata Toolkit

- Reading conditions data
  - from conditions database
  - from simple LCIO file
  - from LCIO data stream
  - from dedicated LCIO-DB file
- Writing conditions data
- tag conditions data
- Browse the conditions database
  - through creation of LCIO files
    - vertically (all versions for timestamp)
    - horizontally (all versions for tag)



LCCD is used by Calice for the conditions data of the ongoing testbeam studies

Gear

GEometry **A**PI for **R**econstruction

```
- <gear>
  - <!--
      Example XML file for GEAR describing the LDC detector
    -->
  - <detectors>
    - <detector id="0" name="TPCTest" geartype="TPCParameters" type
        <maxDriftLength value="2500."/>
        <driftVelocity value=""/>
        <readoutFrequency value="10"/>
        <PadRowLayout2D type="FixedPadSizeDiskLayout" rMin="386.0"
        maxRow="200" padGap="0.0"/>
        <parameter name="tpcRPhiResMax" type="double"> 0.16 </para
        <parameter name="tpcZRes" type="double"> 1.0 </parameter>
        <parameter name="tpcPixRP" type="double"> 1.0 </parameter>
        <parameter name="tpcPixZ" type="double"> 1.4 </parameter>
        <parameter name="tpcIonPotential" type="double"> 0.00000003
      </detector>
    - <detector name="EcalBarrel" geartype="CalorimeterParameters">
        <layout type="Barrel" symmetry="8" phi0="0.0"/>
        <dimensions inner_r="1698.85" outer_z="2750.0"/>
        <layer repeat="30" thickness="3.9" absorberThickness="2.5"/>
        <layer repeat="10" thickness="6.7" absorberThickness="5.3"/>
      </detector>
    - <detector name="EcalEndcap" geartype="CalorimeterParameters">
        <layout type="Endcap" symmetry="2" phi0="0.0"/>
        <dimensions inner_r="320.0" outer_r="1882.85" inner_z="2820.
        <layer repeat="30" thickness="3.9" absorberThickness="2.5"/>
        <layer repeat="10" thickness="6.7" absorberThickness="5.3"/>
      </detector>
    </detectors>
  </gear>
```

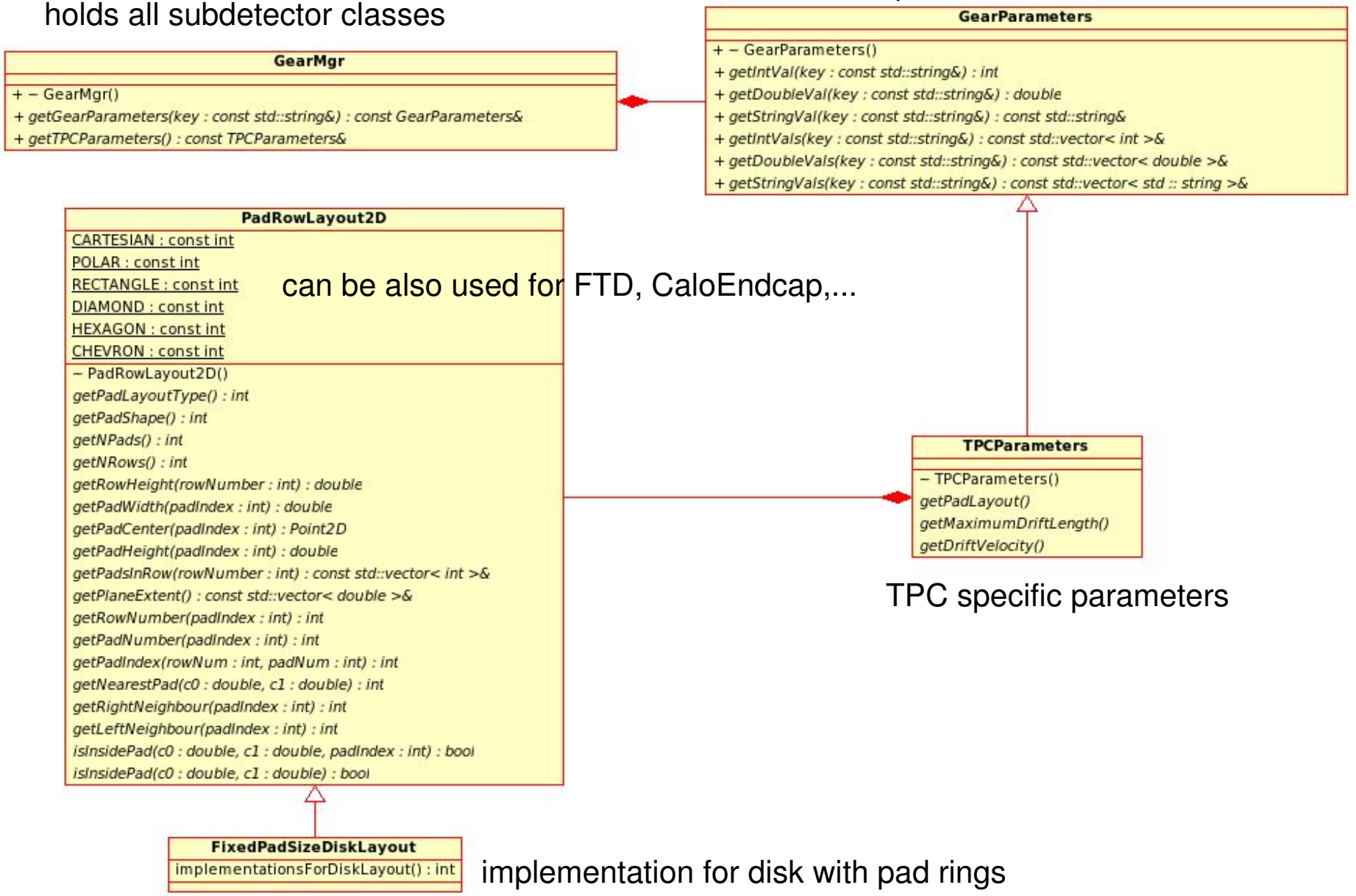
compatible with US – compact format

- well defined geometry definition for reconstruction that
  - is flexible w.r.t different detector concepts
  - has high level information needed for reconstruction
  - provides access to material properties - planned
- abstract interface (a la LCIO)
- concrete implementation based on XML files
  - and Mokka-CGA

GEAR example:TPC description

holds all subdetector classes

named parameters for additional attributes



can be also used for FTD, CaloEndcap,...

TPC specific parameters

implementation for disk with pad rings

GEAR – material properties

GearDistanceProperties

- GearDistanceProperties()
getMaterialNames(p0 : const Point3D&, p1 : const Point3D&) : const std::vector< std :: string >&
getMaterialThicknesses(p0 : const Point3D&, p1 : const Point3D&) : const std::vector< double >&
getNRadlen(p0 : const Point3D&, p1 : const Point3D&) : double
getNIntlen(p0 : const Point3D&, p1 : const Point3D&) : double
getBdL(pos : const Point3D&) : double
getEdL(pos : const Point3D&) : double

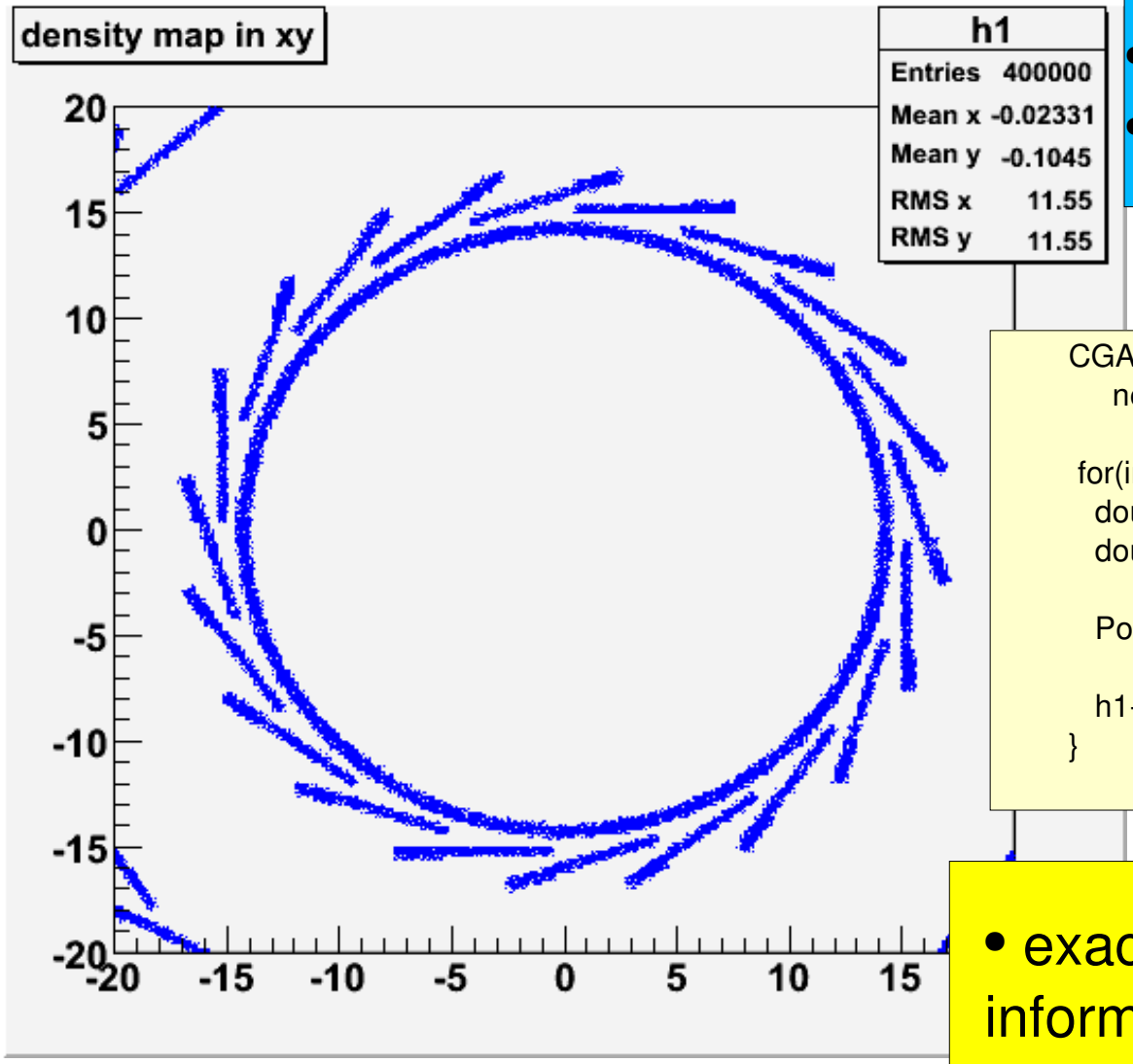
- proposal from Argonne Simulation Meeting 2004
- now implemented with Mokka CGA (geant4)

GearPointProperties

- GearPointProperties()
getCellID(pos : const Point3D&) : int
getMaterialName(pos : const Point3D&) : const std::string&
getDensity(pos : const Point3D&) : double
getTemperature(pos : const Point3D&) : double
getPressure(pos : const Point3D&) : double
getRadlen(pos : const Point3D&) : double
getIntlen(pos : const Point3D&) : double
getLocalPosition(pos : const Point3D&) : Point3D
getB(pos : const Point3D&) : double
getE(pos : const Point3D&) : double
getListOfLogicalVolumes(pos : const Point3D&) : std::vector< std :: string >
getListOfPhysicalVolumes(pos : const Point3D&) : std::vector< std :: string >
getRegion(pos : const Point3D&) : std::string
isTracker(pos : const Point3D&) : bool
isCalorimeter(pos : const Point3D&) : bool

question: is this the correct interface?
e.g. **more** needed for tracking to compute dEdx and in more compact form
i.e. the **actual volumes** possibly with **averaged** material (performance)

CGAGear



- implemented by G.Musat, LLR
- to be released soon

```
CGAGearPointProperties * pointProp =
    new CGAGearPointProperties(steer.str(),...);

 for(int i=0 ; i<nPoint ; ++i){
   double xr =  xmin +  ( xmax - xmin)  * random() ;
   double yr =  ymin +  ( ymax - ymin)  * random() ;

   Point3D p( xr, yr, z0 ) ;

   h1->fill(  xr, yr, pointProp->getDensity( p )  ) ;
}
```

- exact geant4 material & field information at runtime !
- performance ?
- practical issues (linking g4) ?

Gear status

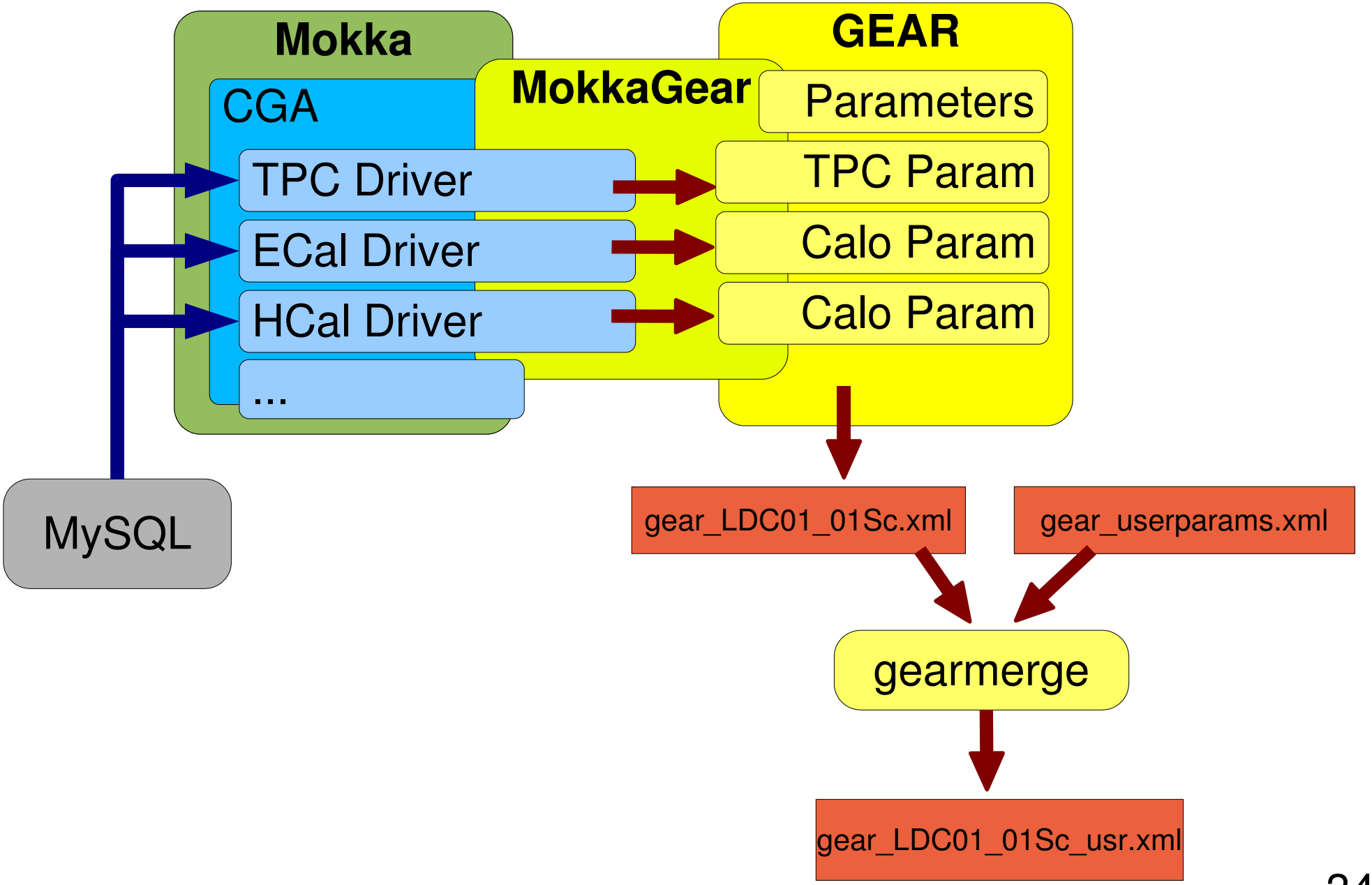
- version v00-03
  - TPC, Hcal, Ecal and VTX interfaces defined and implemented
  - user parameters
  - write xml files from parameters in memory
  - tool to merge files: gearmerge
  - description of TPC prototypes (rectangular pad plane)
- first version of VXD description
- GearCGA - material properties

MokkaGear

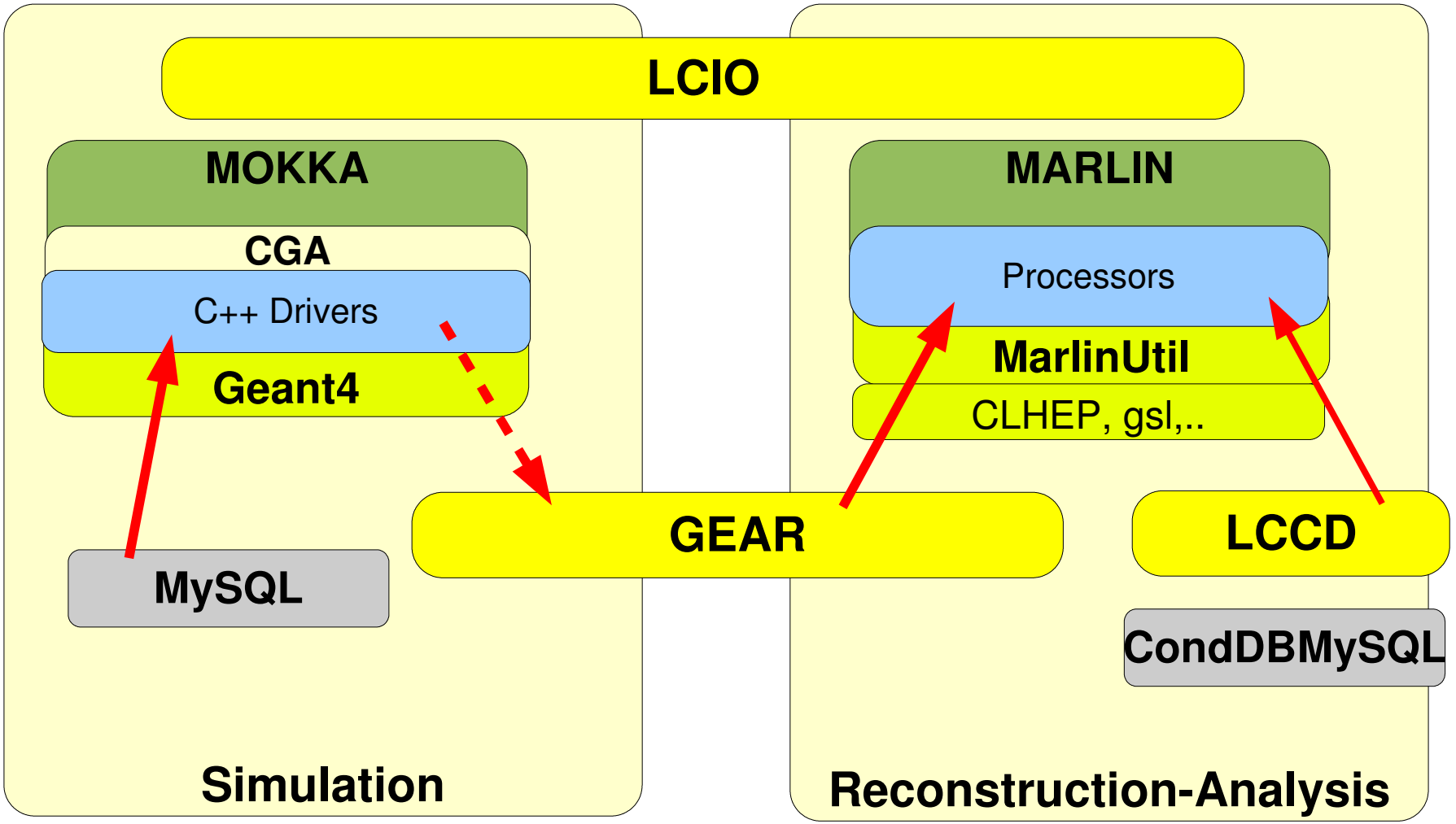
- extension to Mokka (R.Lippe – Diploma Thesis)
- extract geometry information in drivers when detector is built
- use Gear to create XML files for reconstruction
- currently implemented:
  - TPC (tpc04), Ecal (ecal02) and Hcal (hcal04)
- **released with Mokka 6.1**
- optional feature
  - only if Gear is installed and included

aim: have only one source of of information
for describing the detector geometry !

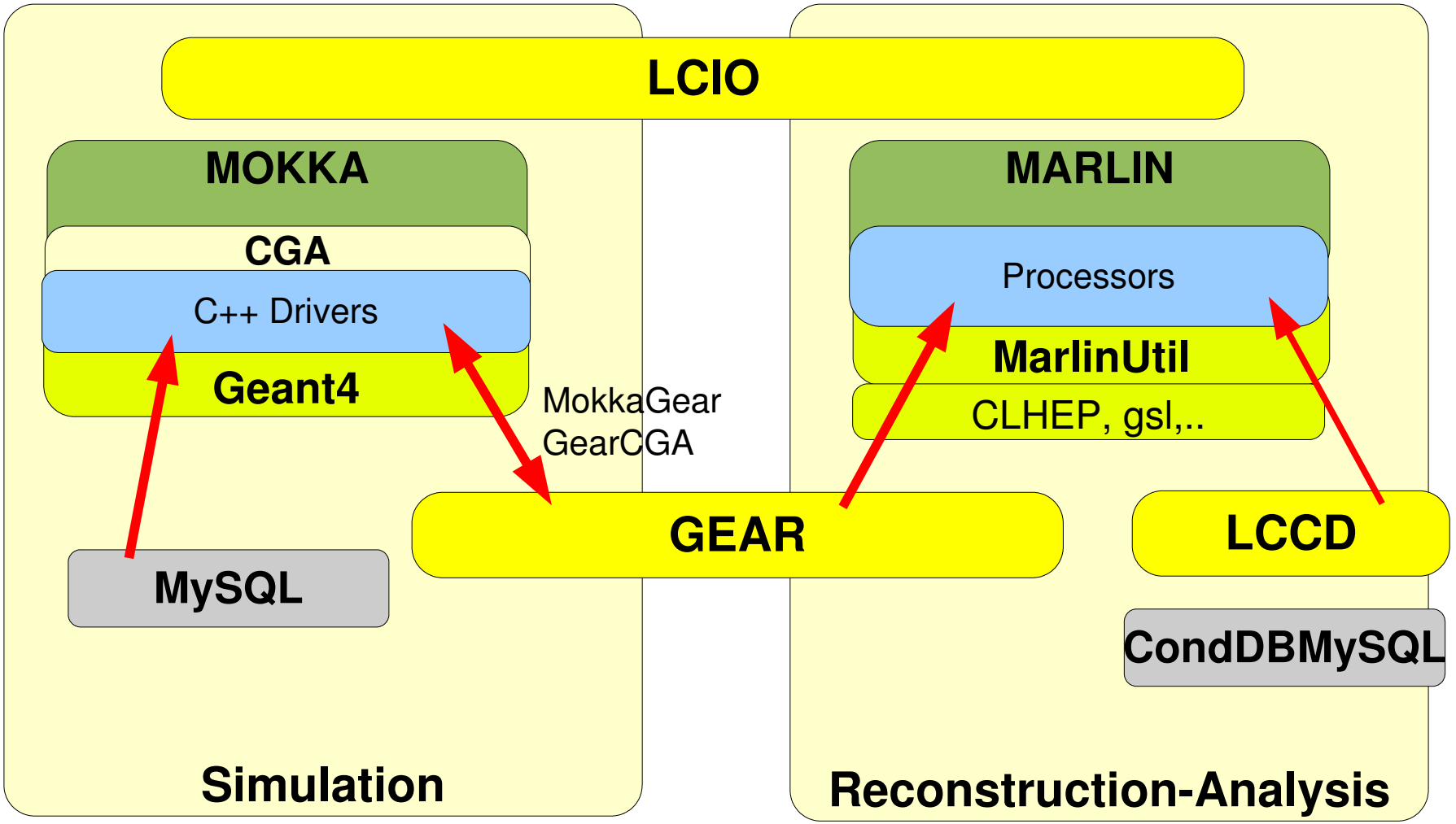
MokkaGear


Mokka
CGA
TPC Driver
ECal Driver
HCal Driver
...
MokkaGear
GEAR
Parameters
TPC Param
Calo Param
Calo Param
MySQL
gear_LDC01_01Sc.xml
gear_userparams.xml
gearmerge
gear_LDC01_01Sc_usr.xml

LDC simulation framework



LDC simulation framework



MarlinReco

- Marlin serves as a **framework** for the distributed development of reconstruction algorithms
  - provides a well defined modularity
- MarlinReco is a **toolkit** which aims at providing reconstruction algorithms for detector concept studies
  - (almost) complete set of standard reconstruction (pflow)
  - cheaters for cross checks (and replacements)
  - all processors can seamlessly be combined together with other reconstruction code or plugged into your analysis
    - e.g. together with PandoraPFA (M.Thomson), ZVtop (LCFI)...

MarlinReco packages

- **TrackDigi**
  - TPCDigi
  - new VTXDigi
- **CaloDigi**
  - LDCCaloDigi
- **Tracking**
  - LEPTracking (wrapped f77)
  - new VTXTracking
  - TrackCheater
- **Clustering**
  - TrackwiseClustering
  - ClusterCheater
- **Pflow**
  - Wolf
- **Analysis**
  - EventShapes
  - SatoruJetFinder

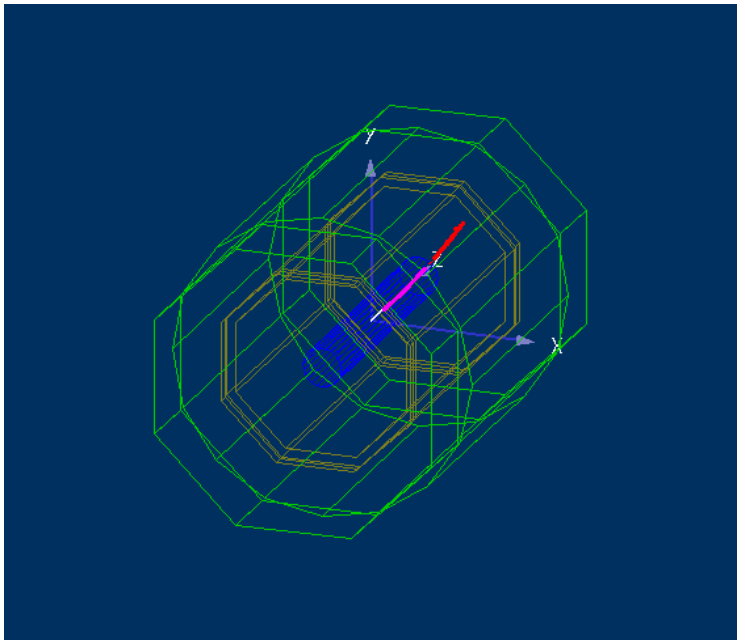
most MarlinReco processors (algorithms) are **geometry independent**
–> they can be applied to **other detector concepts** (via Gear file)

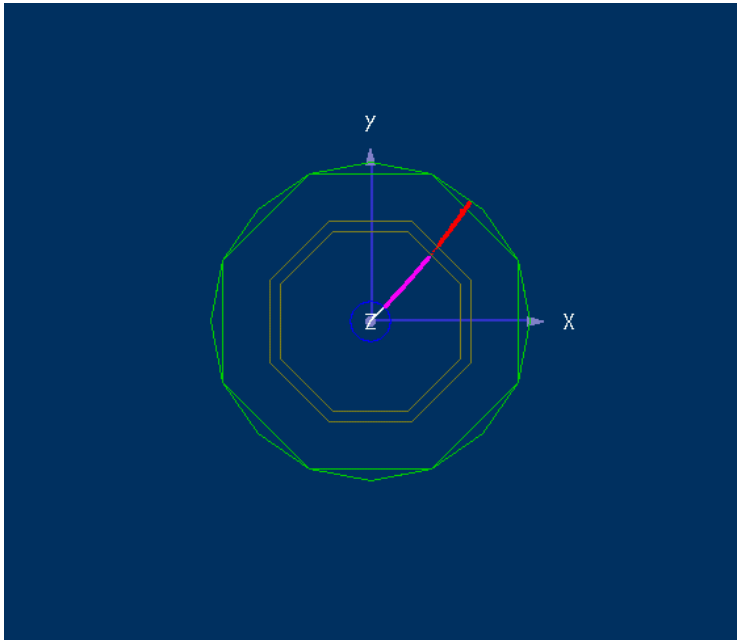
MarlinReco support packages

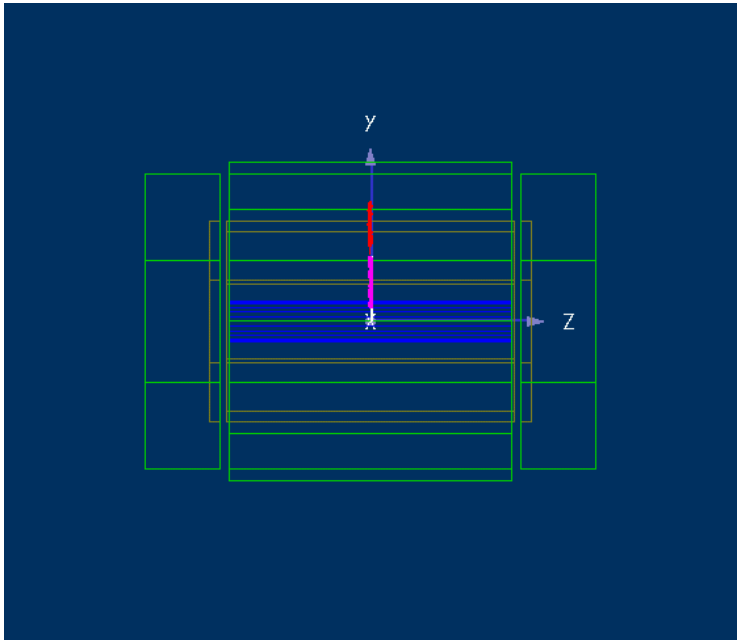
- MarlinUtil (O. Wendt)
  - Utility and Helper classes
    - helix fitter, cluster shapes,...
    - common code for CED
- RAIDA
  - AIDA root implementation
- CED (A. Zhelezov)
  - event display based on GLUT/ OpenGL
  - client server architecture
- CEDViewer
  - event display client processors
    - CEDViewer. GenericViewer

CEDViewer

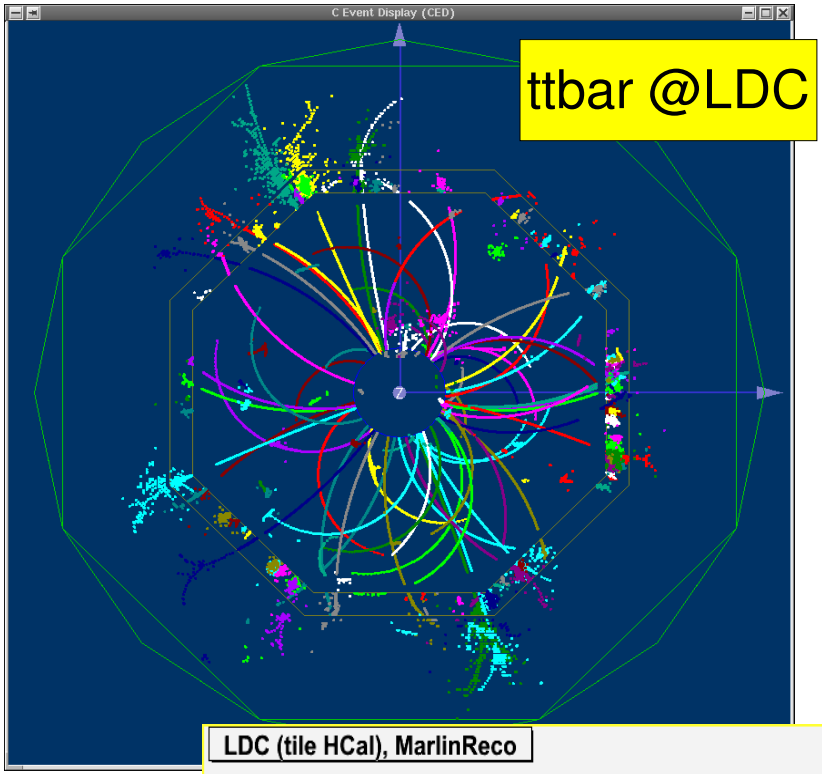
simple example taken from
$MarlinReco/examples/LDC
steer_ldc.xml
gear_ldc.xml



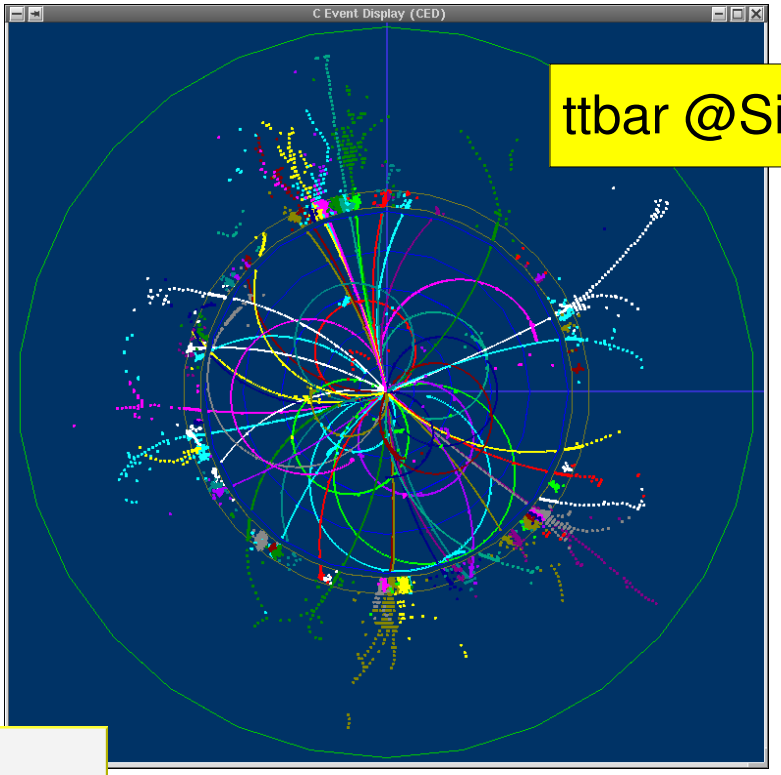




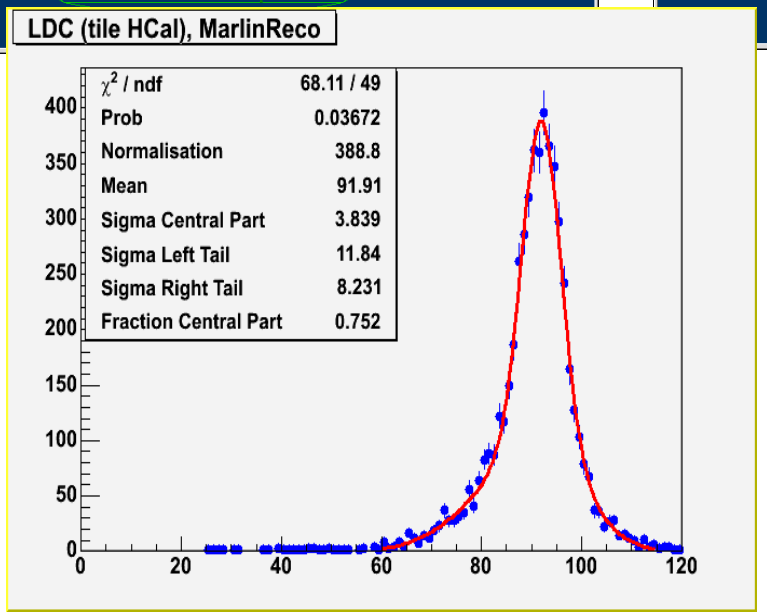
MarlinReco @work



ttbar @LDC

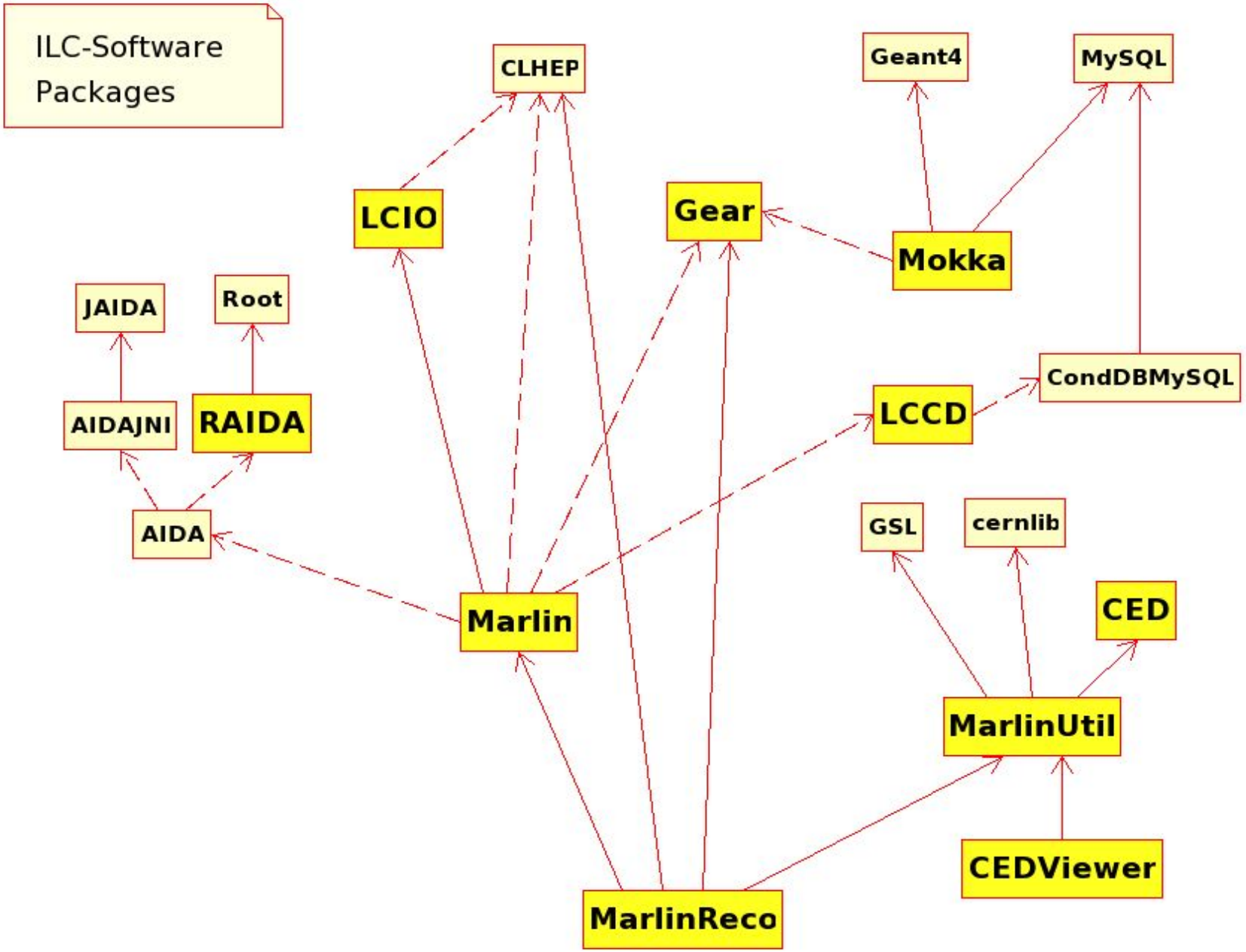


ttbar @SiD

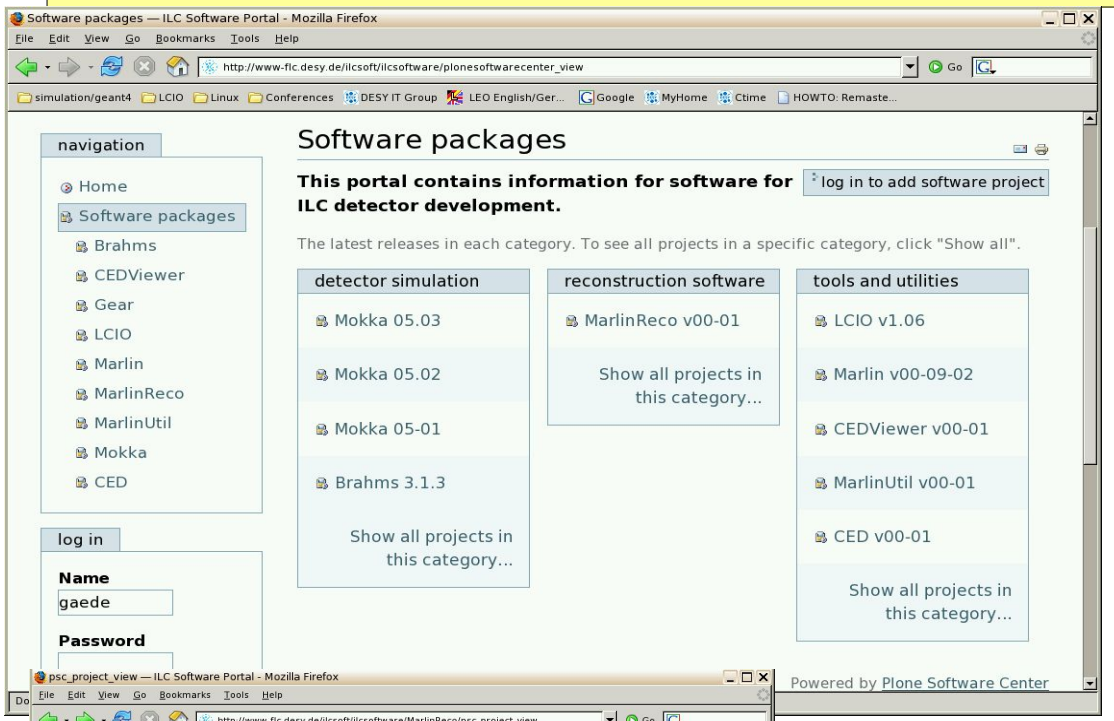


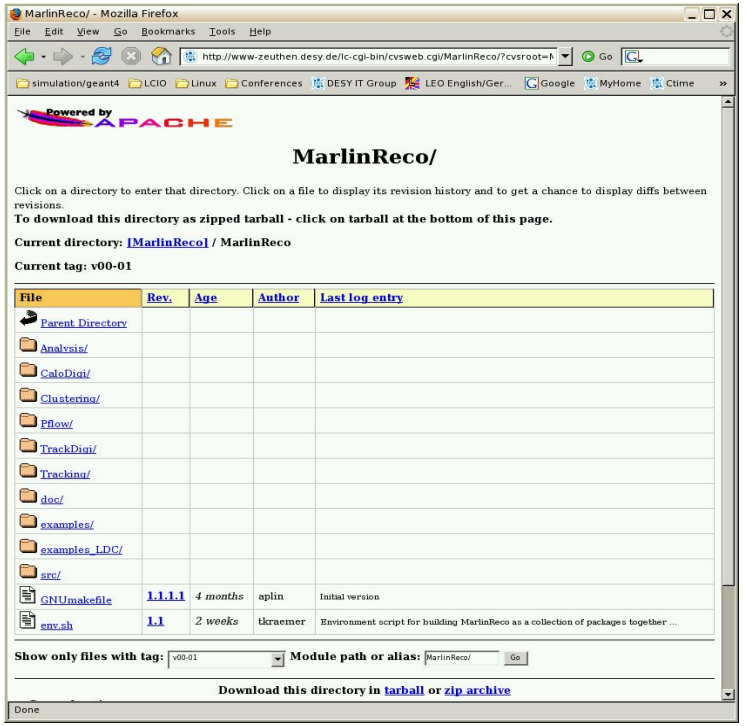
LCIO provides interoperability

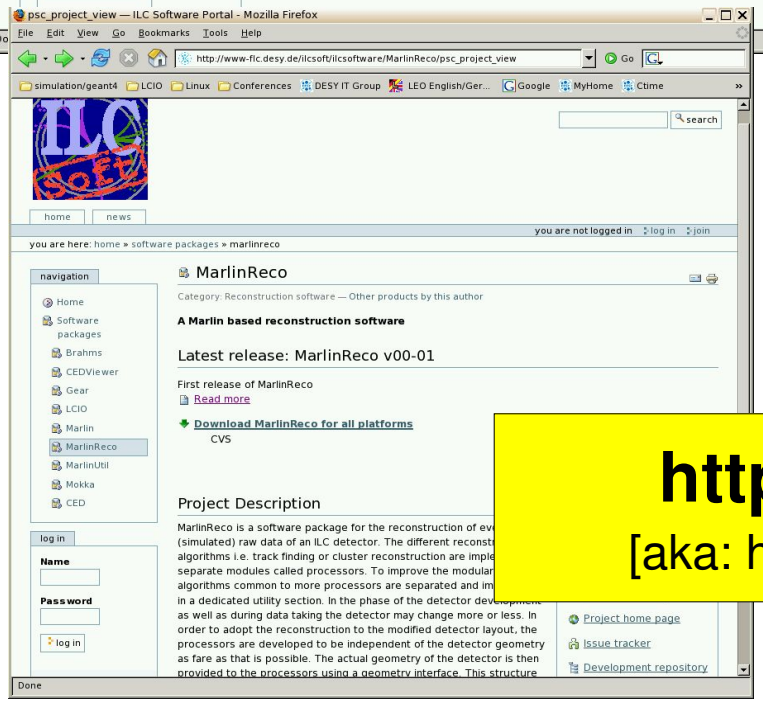
software package dependencies

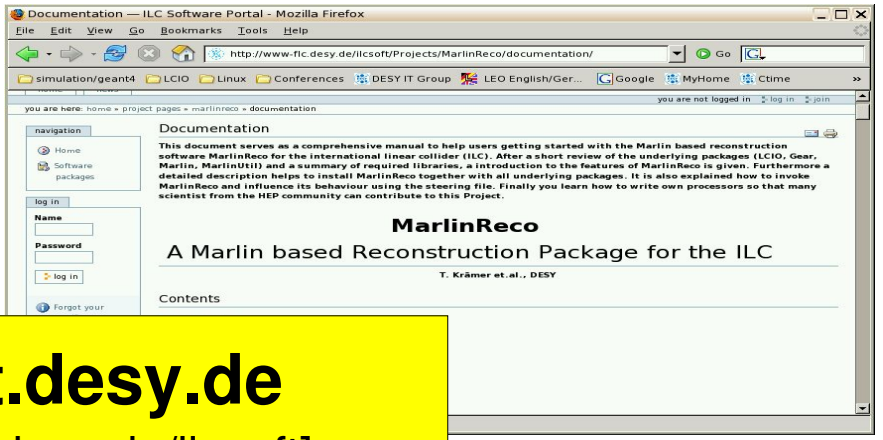


ILC software portal







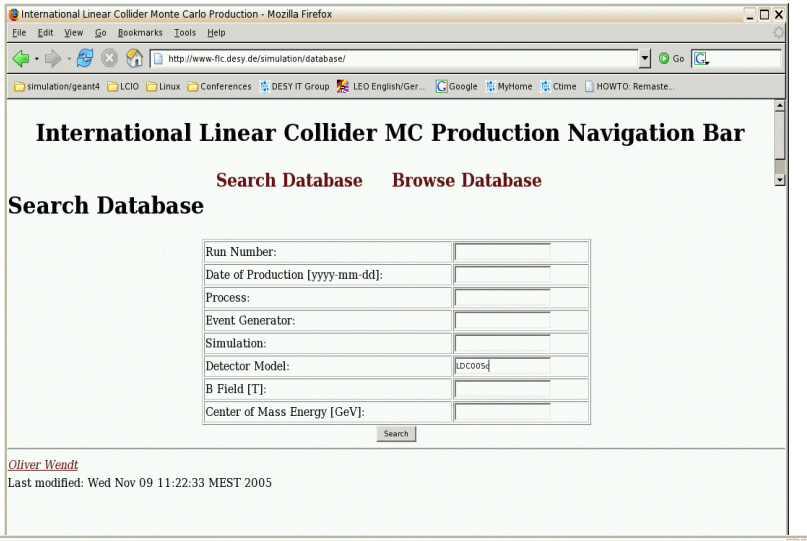


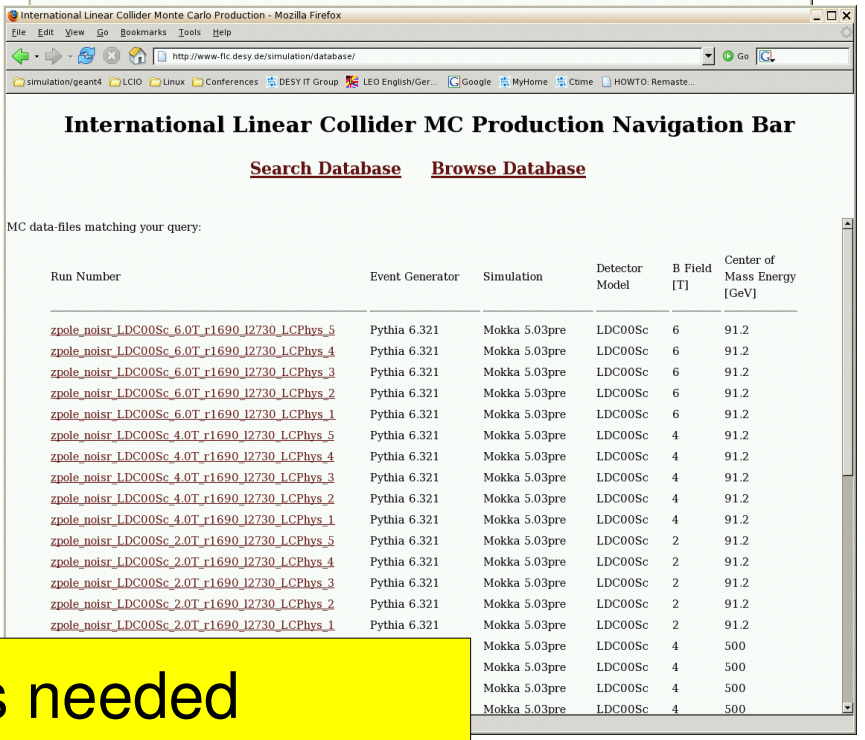
http://ilcsoft.desy.de

[aka: http://www-flc.desy.de/ilcsoft]

ILC grid - “mass production”

- detector optimization – vary
  - B,R_TPC,L_TPC,...
- need considerable number of events with detector parameter variations for benchmark reactions
- produced these files on the **grid** for VO ILC – 450 kevts:
  - Z0 and uds, ccbb , ttbar, WW, ZH @ 500 GeV
  - 4 detector variants, 3 T and 4 T field
- database with available data files
- use **grid tools** to distribute/download the data !

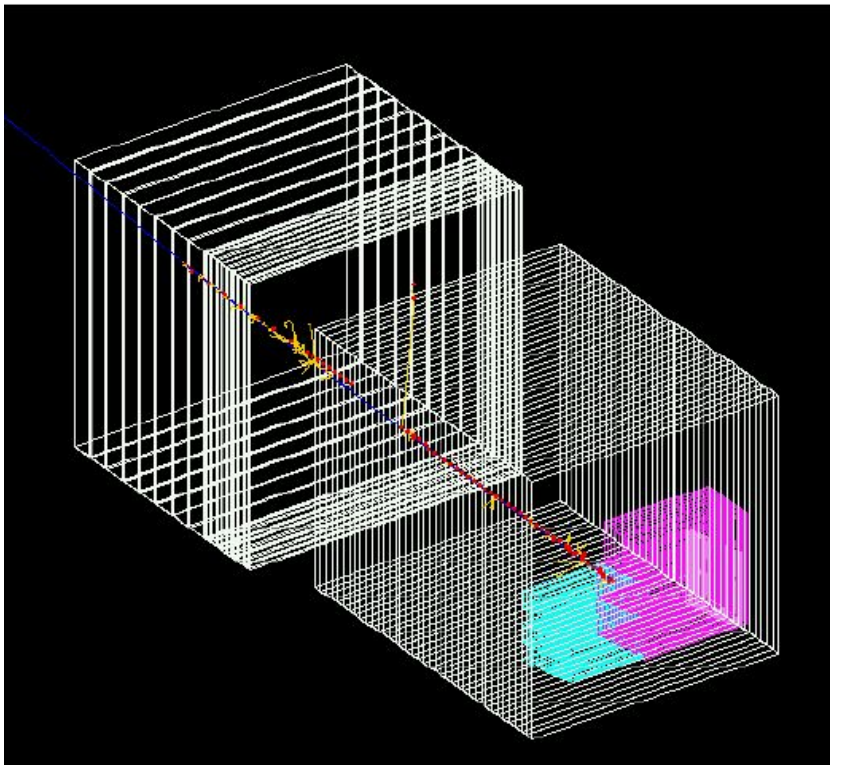




- provide the simulated data that's needed
- exercise the software & computing infrastructure

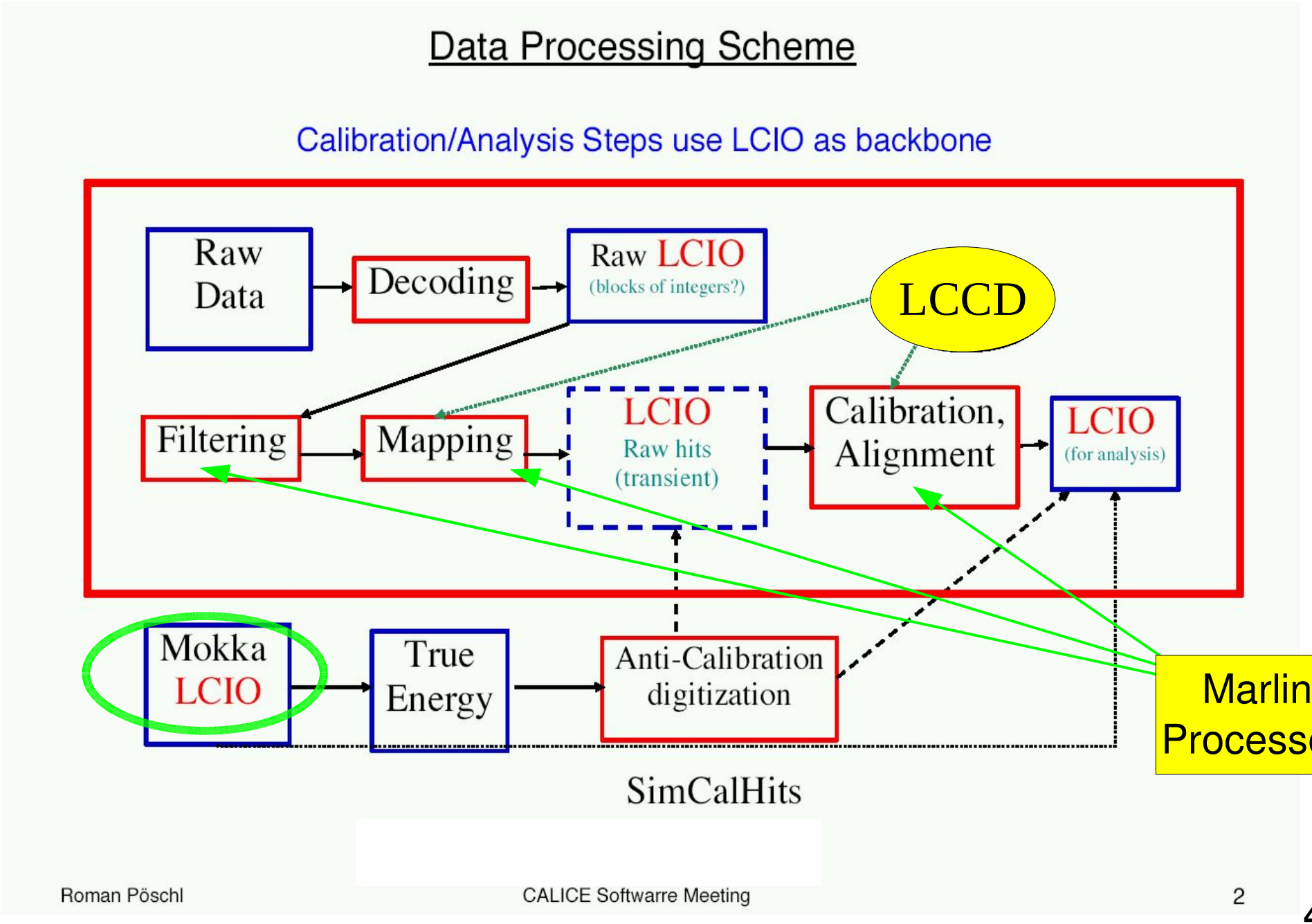
CALICE testbeam

- testbeams for CALICE prototypes (Ecal, Hcal, Tailcatcher)
  - DESY 2005:
    - e- 1-6 GeV
  - CERN/FNAL 2006/2007
    - mu,pi,p 1-100GeV
- O(10^8) events
- n*O(10^8) Monte Carlo events
- ~ 20 000 channels



- detector R&D
  - test DAQ
  - test detector concepts
- software
  - hadronic physics in Geant4
  - **test complete software chain !**

Calice testbeam software



Interoperability of ILC software I

- the interoperability of the ILC software frameworks is a key ingredient for collaboration on optimizing the detector
- LCIO as a persistency format and event data model provides a minimal basis for the interoperability:
  - exchange of software that uses LCIO is input and output
  - exchange of data files (e.g. from g4-based full simulation)
  - use LCIO files as intermediate glue between modules
- however still a lot of double work exists:
  - geometry definition
  - tracking (pattern recognition, Kalman filter)
  - vertexing
  - similar algorithms in different frameworks (languages Java/C++)

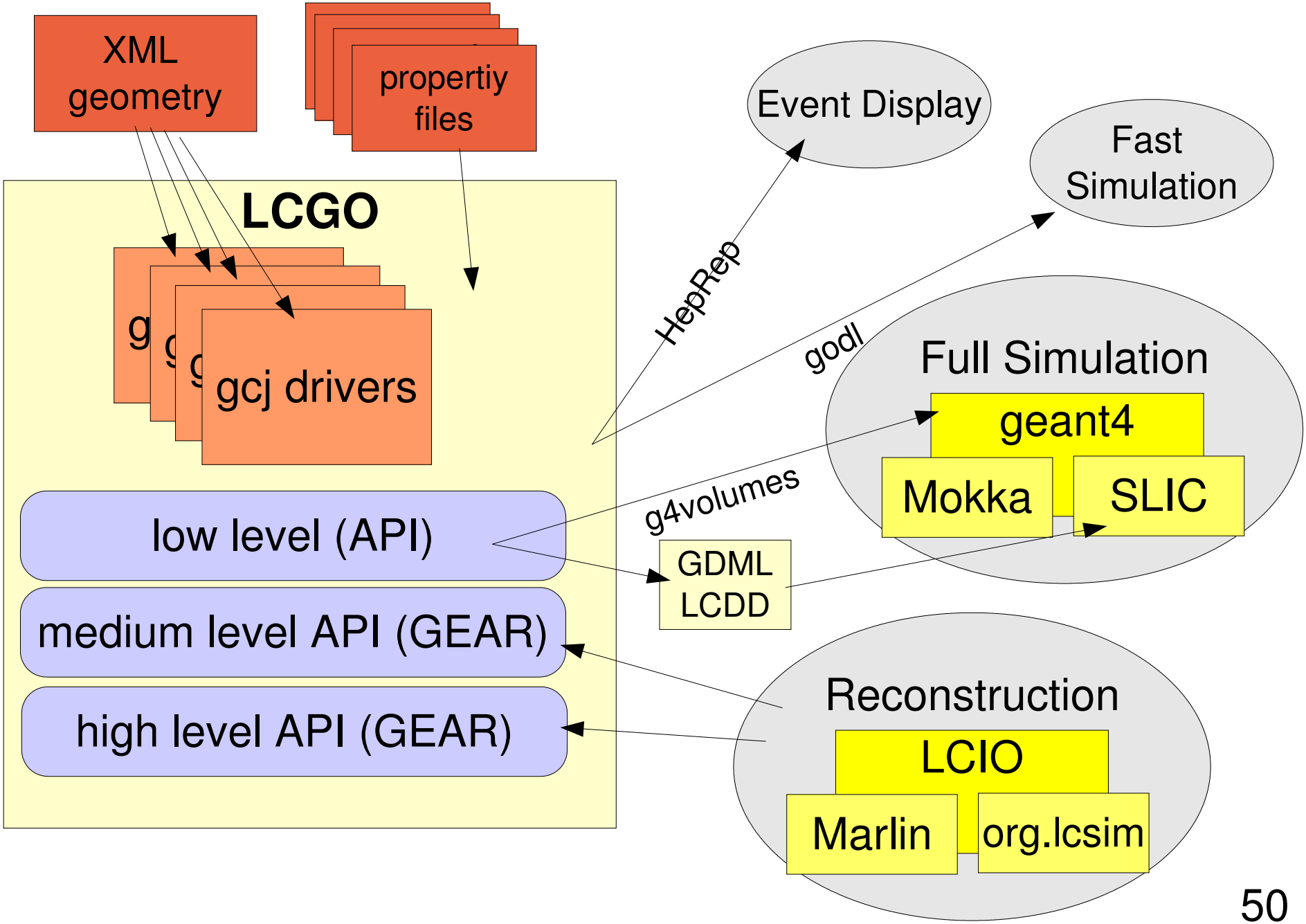
Interoperability of ILC software II

- next step of inter operation is to use software modules developed in a different framework
  - example prototypes: org.lcsim(Java) vs. Marlin(C++)
    - calling Marlin processor from Java main program:
      - use SWIG to create Java binding of Marlin/LCIO (SLAC)
    - calling org.lcsim drivers from Marlin main:
      - write/read LCIO data to/from stream in memory in between calls to different language modules (DESY)
  - promising first results, however a number of technical issues need to be resolved
  - problem: two independent (somewhat similar) ways of defining the geometry in the two frameworks
  - -> **need common geometry package**

A Common Geometry Toolkit

- **LCGO**: A common geometry toolkit to be used in all (?) ILC frameworks
  - SLAC-DESY project - initially
  - -> of course open for all collaborators, e.g. FNAL !
- requirements/goals for LCGO:
  - be at least as functional as existing systems (org.lcsim, GEAR, Mokka, SLIC,...)
  - enable smooth transition path from existing systems
  - encourage/increase interoperability between systems
  - have no known principle short comings: “everything should be possible”

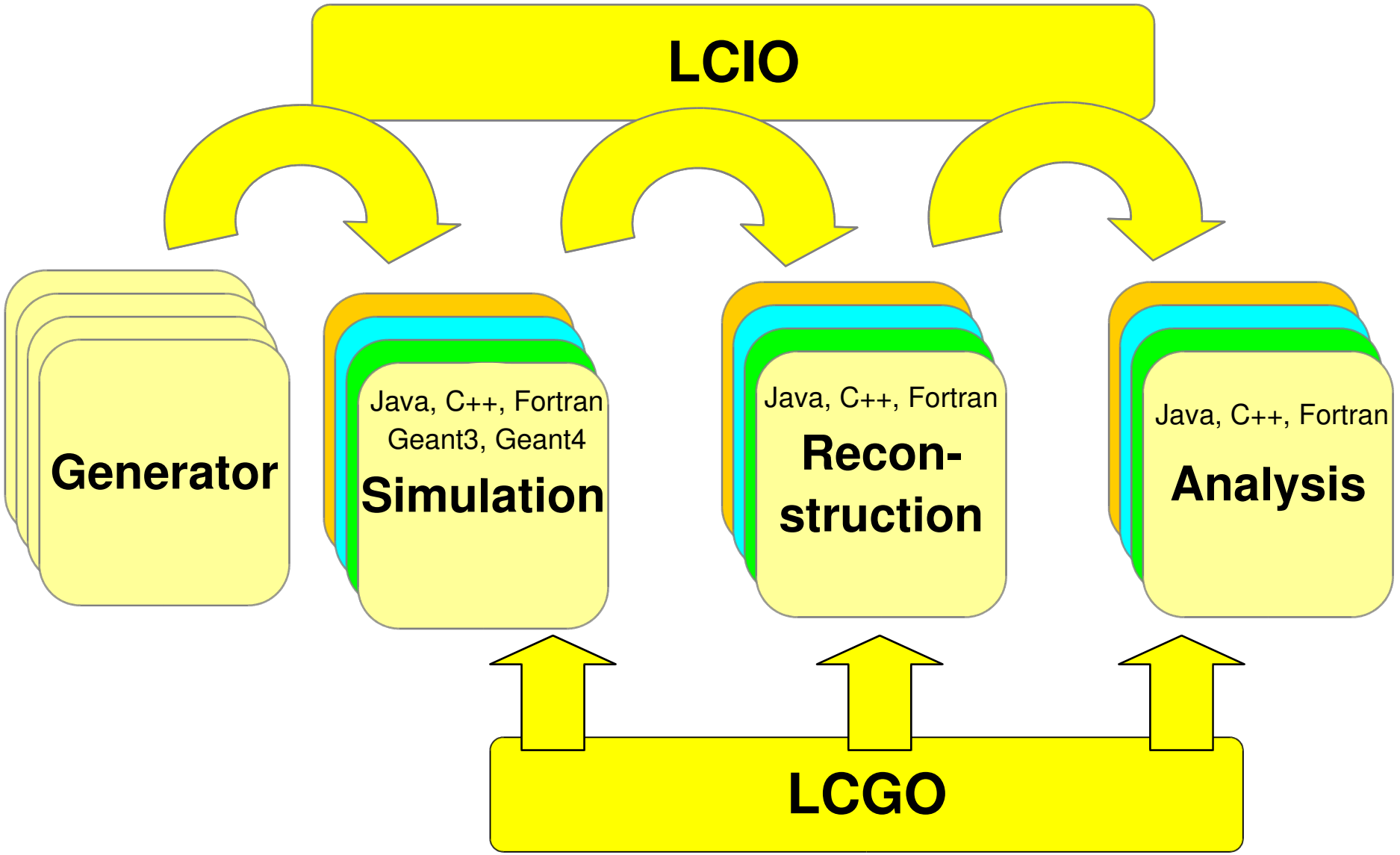
LCGO implementation prelim.



some LCGO planned features

- extended GEAR interface (medium and high level)
  - tracking (and clustering PFA)
    - average material volumes
    - intersection with 'next' volume
    - dE/dx
    - field maps
    - access to volumes
    - extensions of detectors (a la gear)
      - e.g. #layers, thickness, width,..
- material database
- field maps
- properties (sampling fractions)
- readout properties
  - cellId <-> position
  - cellid range (noise simulation)
  - cell sizes
  - neighbors
- Vector and Matrix classes ?
  - ThreeVector, Point3D
  - Planes, cylinders, ... ?
  - FourVector
  - SymMatrix (covariances)

ILC interoperable software chain



Summary

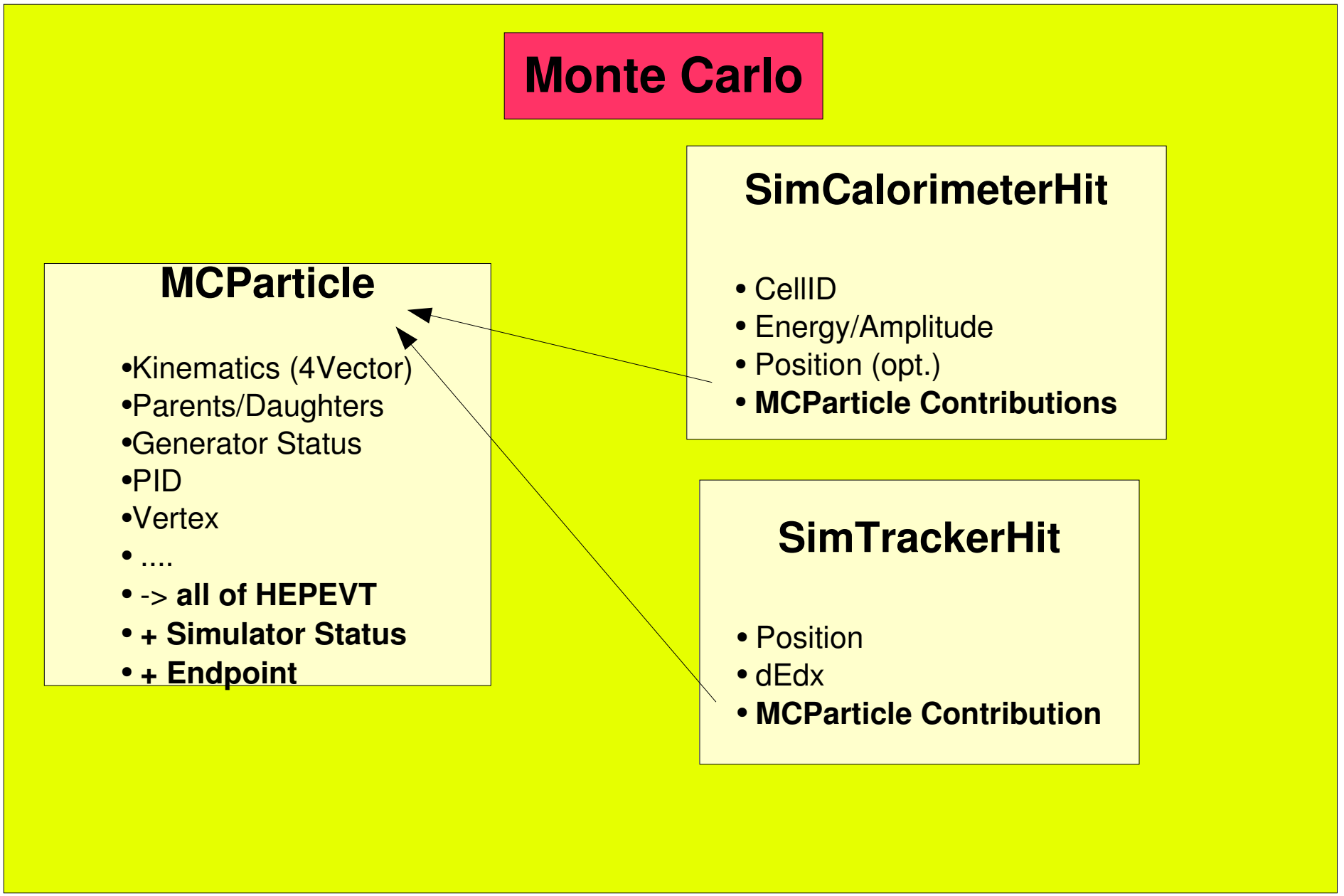
- a fairly complete OO-software framework exists for the LDC study based on Mokka, Marlin, LCIO, LCCD and GEAR
  - used for detector concept study
- to be extended with LCGO

details @ software portal:
http://ilcsoft.desy.de/

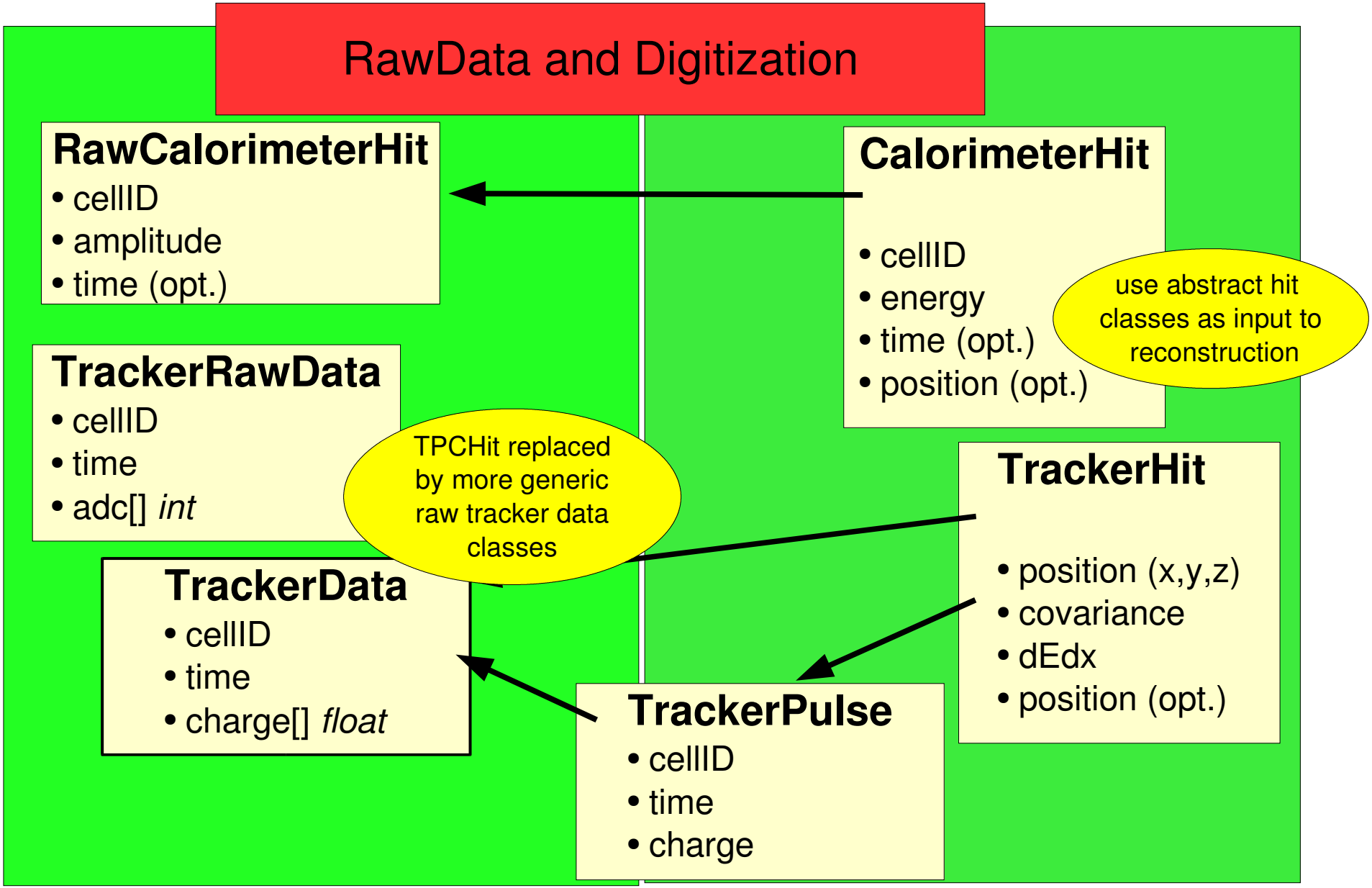
GOAL:
- **increase interoperability with other frameworks**
- **unify rather than duplicate**
- **safe man power to work on real issues like**
- **reconstruction, i.e. Tracking, PFA,...**

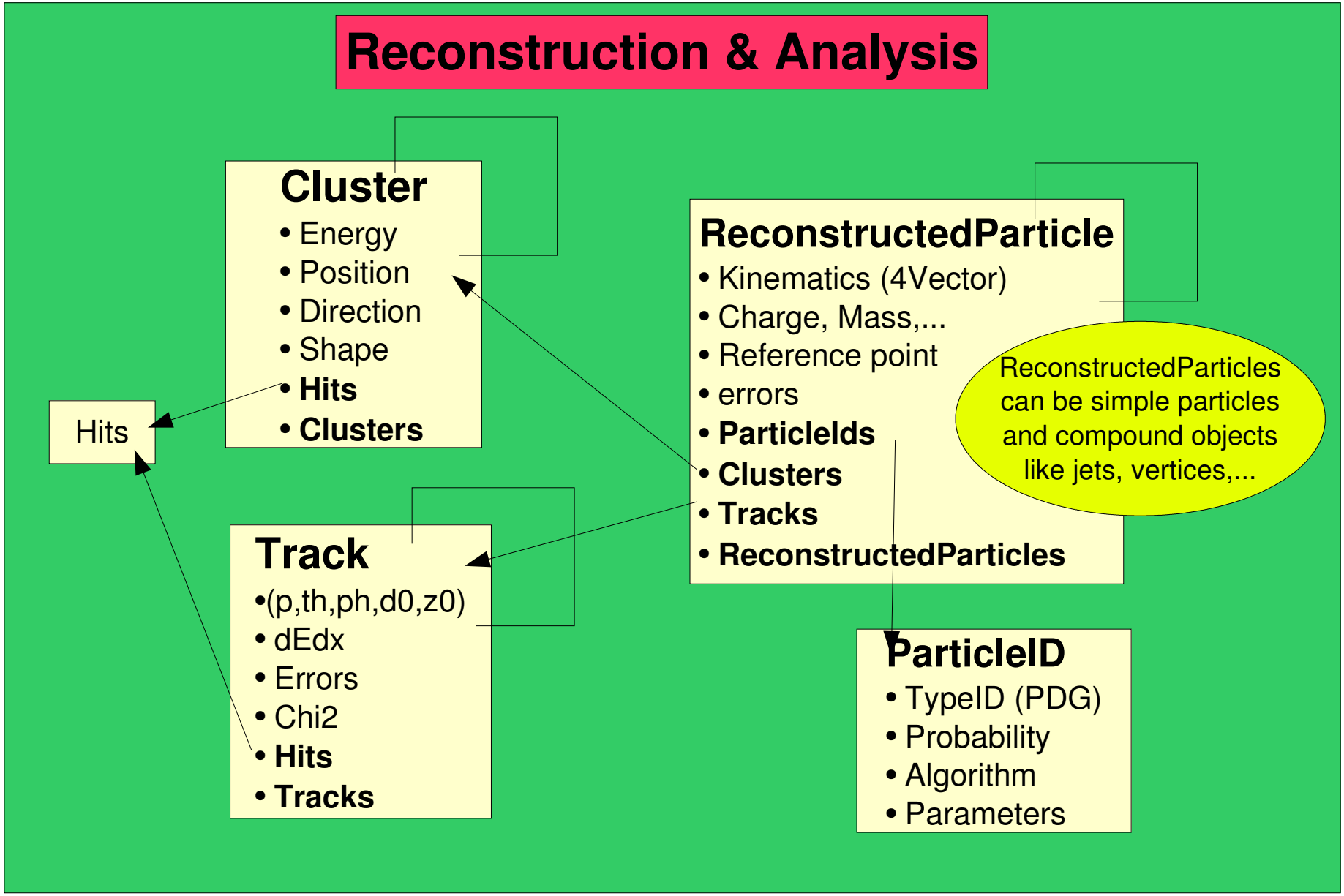
Please consider using common software tools
in particular **LCIO**
for your ILC study and provide feedback
and **contribute** to the effort !

Backup slides ...


Monte Carlo
SimCalorimeterHit
• CellID
• Energy/Amplitude
• Position (opt.)
• MCParticle Contributions
MCParticle
•Kinematics (4Vector)
•Parents/Daughters
•Generator Status
•PID
•Vertex
• ....
• -> all of HEPEVT
• + Simulator Status
• + Endpoint
SimTrackerHit
• Position
• dEdx
• MCParticle Contribution

Datamodel IV




Reconstruction & Analysis
Cluster
• Energy
• Position
• Direction
• Shape
• Hits
• Clusters
Hits
ReconstructedParticle
• Kinematics (4Vector)
• Charge, Mass,...
• Reference point
• errors
• ParticleIds
• Clusters
• Tracks
• ReconstructedParticles
ReconstructedParticles can be simple particles and compound objects like jets, vertices,...
Track
•(p,th,ph,d0,z0)
• dEdx
• Errors
• Chi2
• Hits
• Tracks
ParticleID
• TypeID (PDG)
• Probability
• Algorithm
• Parameters

LCIO Future Plans

- current data model fairly complete
  - check usability of data model, e.g. track parameters
    - ongoing through development of reconstruction code !
  - need to define conventions on how to use LCIO
    - collection names, object types, collections to be present in event
    - see talk by
- need to make LCIO more convenient (and efficient)
  - decoding of MCParticle information (parent/daughter, ...)
  - inverse relations (get all tracks for one MCParticle)
  - attach user information to LCObjects
- would like to make C++ version more C++ like, e.g. allow to use STL algorithms (templates in general)
  - planed for LCIO v02-00
- **need user input !!**

LCIO status [v01-07]

- added optional momentum[3] and pathLength to SimTrackerHit (silicon digitization)
- LCReader::skipNEvents()
  - dump the n-th event in $LCIO/bin/dumpevent
- C++ utility code for encoding/decoding of cellids in hit classes (32bit and 64bit)
- support for large files (>2GB) and UTIL::LCSplitWriter
- optional python binding (J.McCormick)
  - files are downward compatible with LCIO 1.6
  - see $LCIO/doc/versions.readme for more

Future plans - activities

- short/mid term:
  - interoperability with other frameworks
    - interface Java/C++ based on LCIO event data model
    - unify the geometry description based on xml (Gear/LCCD)
  - improve/extend software
    - PFA algorithms, track reconstruction,...
    - event display, conditions db,....
- mid/long term:
  - investigate a new data storage format for LCIO (possibly based on rootIO)
  - ...