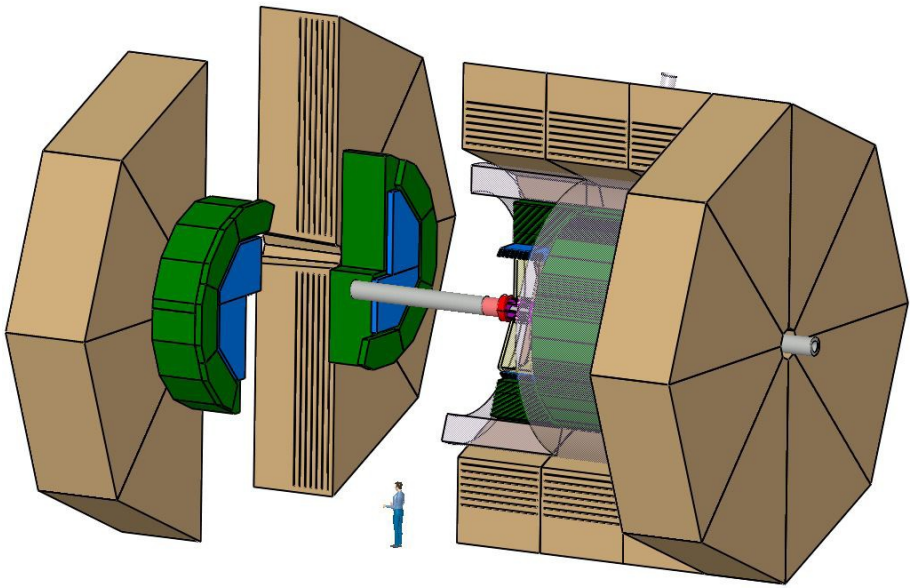
The LDC Software Framework for the ILC detector

Frank Gaede
DESY
CHEP 2007, Victoria Canada
September 2-9, 2007

Outline

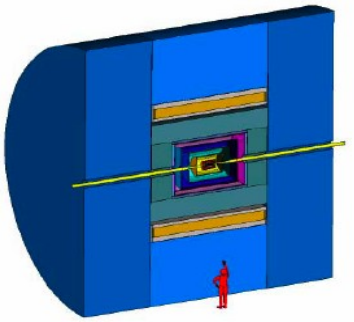
- introduction - overview
- **LCIO** persistency & datamodel
  - recent developments
- **Marlin** reconstruction framework
  - design/software architecture
  - new features
  - supporting tools
  - applications/results
- summary



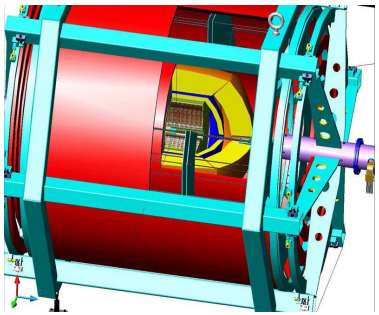
Introduction

- 4 international detector concept studies for the ILC ongoing
  - DCRs written this year
  - 3 LOIs planned for 2008 (joined LDC/GLD)
  - 2 EDRs planned for 2010
- 4 independent sw frameworks exist
  - some interoperability provided through use of common event data model/ file format LCIO
- "Marlin et al" is the LDC concept's framework
  - design presented at CHEP2006
  - this talk: Evolution of the framework

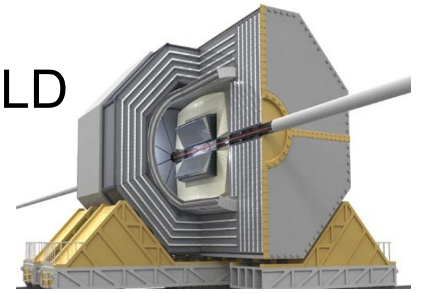
SID



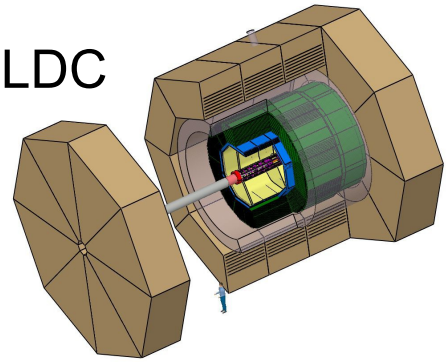
4th



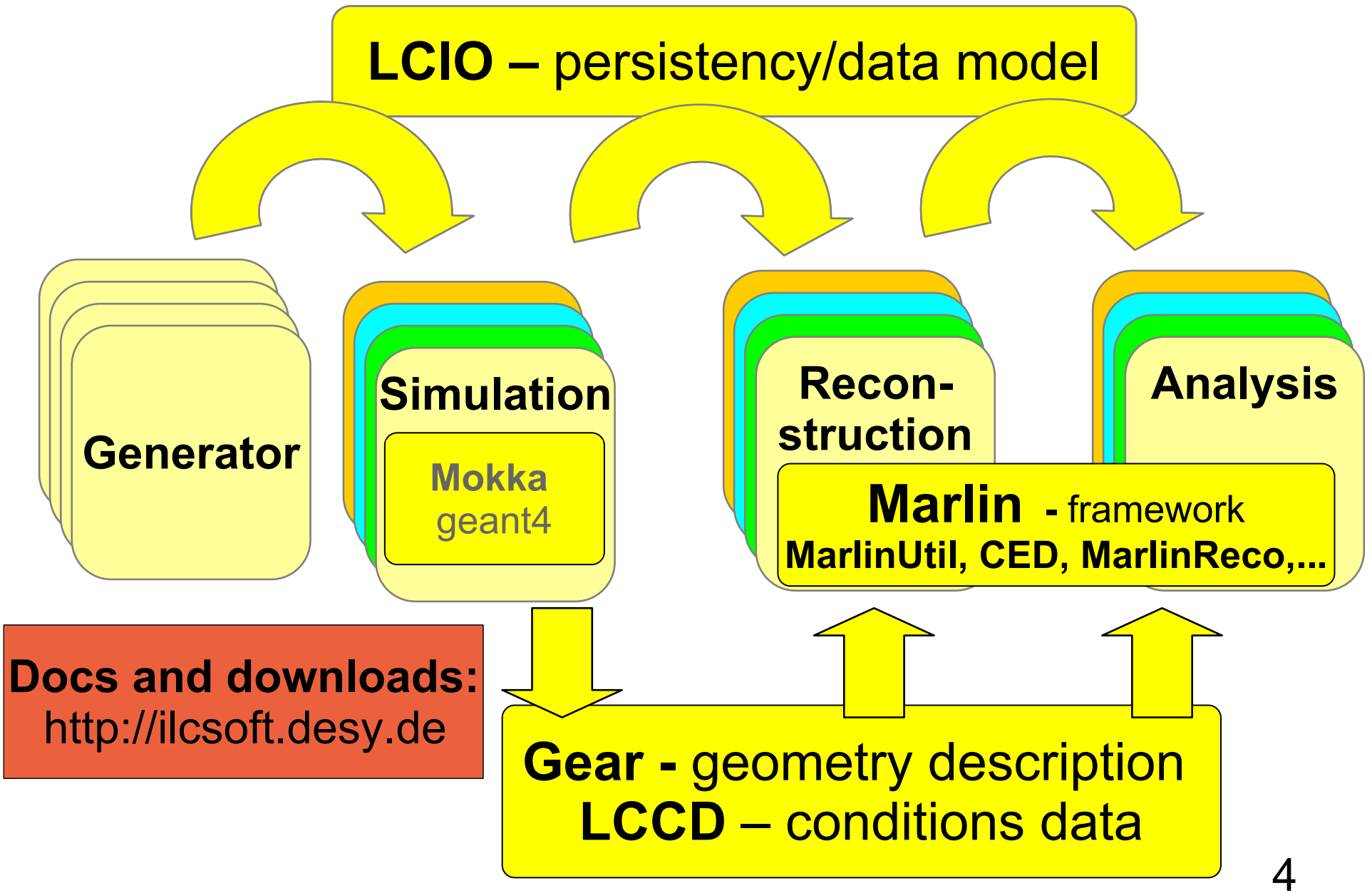
GLD



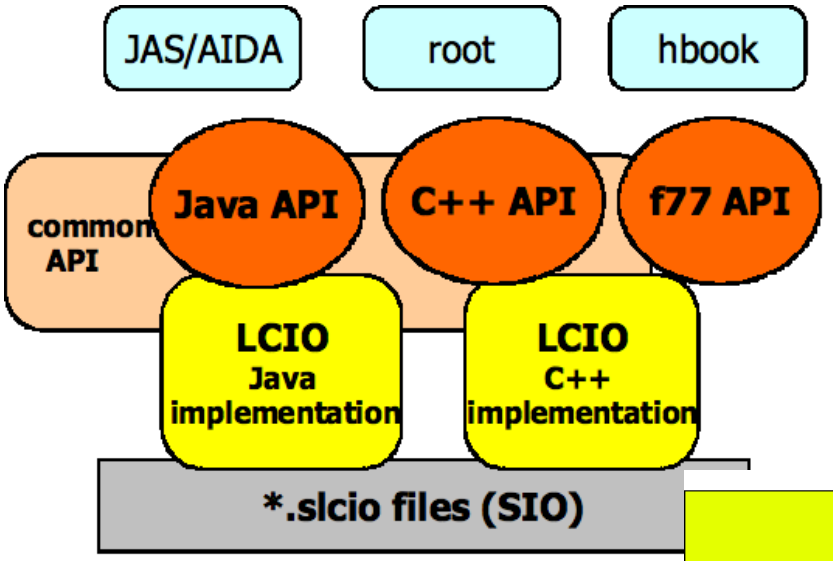
LDC



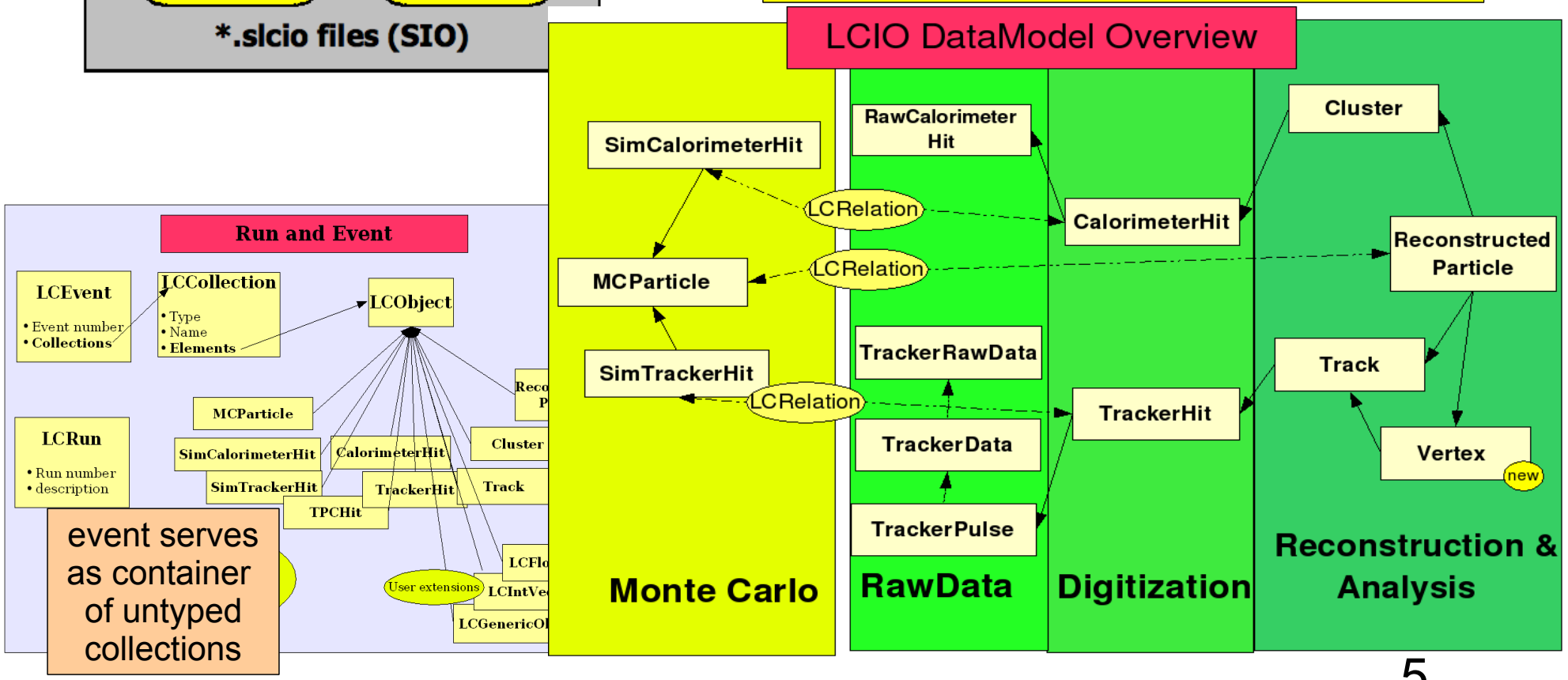
LDC sw-framework overview



LCIO: persistency & event data model



- DESY SLAC joined project
  (first presented at CHEP03)
- Java, C++ and f77 (!) API
- extensible data model
- now standard for ILC persistency & datamodel
  -> used in all detector concept studies



LCIO new developments

- extended the event data model
  - introduced dedicated Vertex class
    - (originally thought vertices as part of RecontructedParticles)
  - introduced raw data classes for tracking detector testbeams: TrackerRawData, TrackerData, TrackerPulse
- comand line tool (Java) for checking and manipulating events: dump, merge, split,...
- runtime extensions (next slides)
- under development
  - improve I/O performance
  - direct access: reading part of the event, split files,...
  - user defined data classes

LCIO runtime extensions (C++)

- long pending user request:
  - attach user objects to LCObjects
  - fast and easy creation of links (relations) between various LCObject subtypes, eg. TrackerHits and Track
- features
  - extension of the object with arbitrary (even non-LCObject) classes
  - extension of single objects or vectors, lists of objects
  - optionally ownership is taken for extension objects (memory management)
  - bidirectional relations between LCObjects
    - one to one
    - one to many
    - many to many

to be used in reconstruction and analysis algorithms - no persistency

LCIO runtime extensions

extensions and relations identified through a tagging **class T**

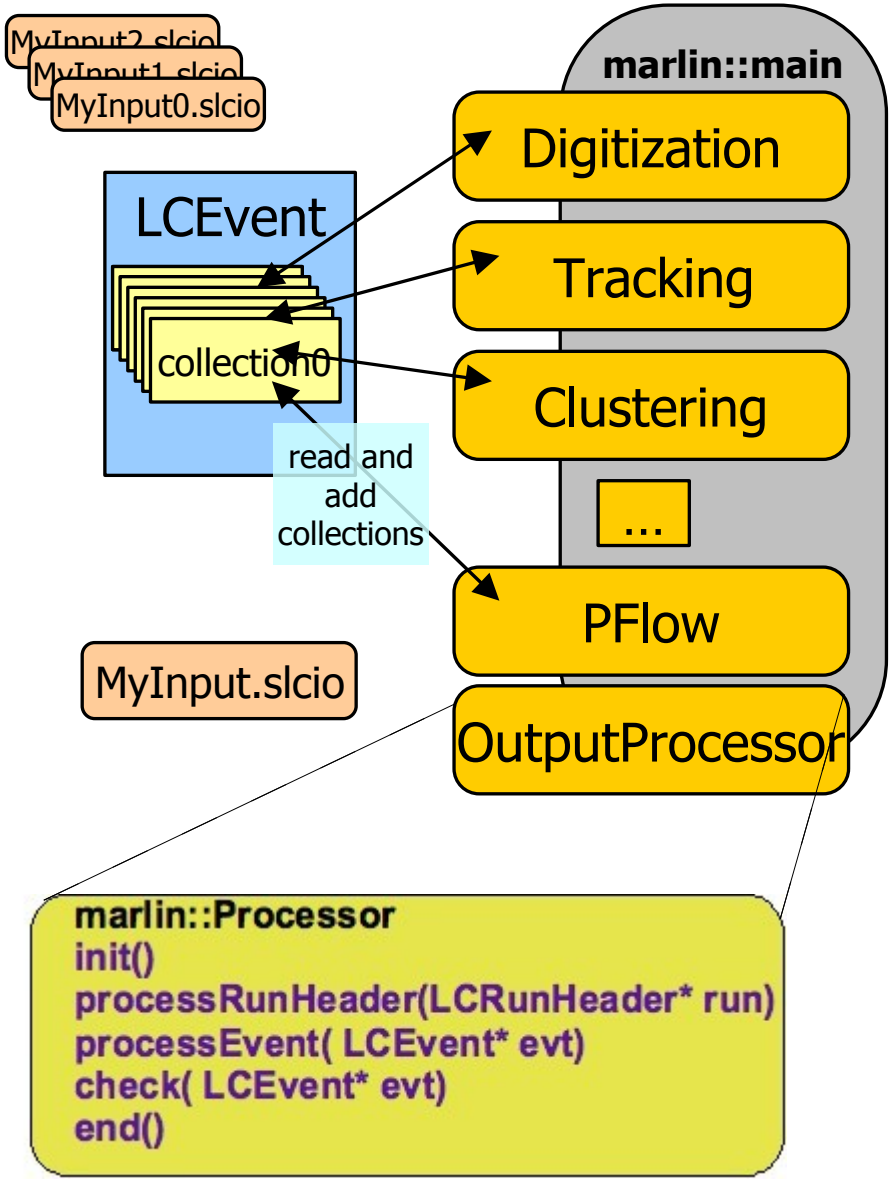
```
// a simple int extension
struct Index : LCIntExtension<Index> {} ;

// a many to many relationship between MCParticles
struct ParentDaughter : LCNToNRelation<ParentDaughter,MCParticle,MCParticle> {} ;
//..
 MCParticle*  mcp = dynamic_cast<MCParticle*>( mcpcol->getElementAt(i) ) ;
//..

 mcp->ext<Index>() = i;    // set an int


 const MCParticleVec& daughters = mcp->getDaughters() ;

 for(unsigned j=0 ; j< daughters.size()   ; j++ ){

   // ---- set biderctional relation
   add_relation<ParentDaughter>( mcp, daughters[j] ) ;
 }

//-------------------------------------------------

 cout << " myindex = " << mcp->ext<Index>  << endl ;

 ParentDaughter::to::rel_type daulist =  mcp->rel<ParentDaughter::to>() ;

 for( ParentDaughter::to::const_iterator idau = daulist->begin();
   idau != daulist->end(); ++idau){

    cout << (*idau)->ext<Index>() << ", " ;
 }
 cout << endl ;
```

for extensions use **ext<T>()**
for relations use **rel<T::to>()** and **rel<T::from>()**

Marlin – core application framework

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

- modular C++ application framework for the analysis and reconstruction of ILC data
- **LCIO** as transient data model
- xml steering files:
  - fully configure application
  - order of modules/processors
  - parameters global + processor
- self documenting
  - parameters registered in user code
- consistency check of input/output collection types
- **Plug & Play** of modules

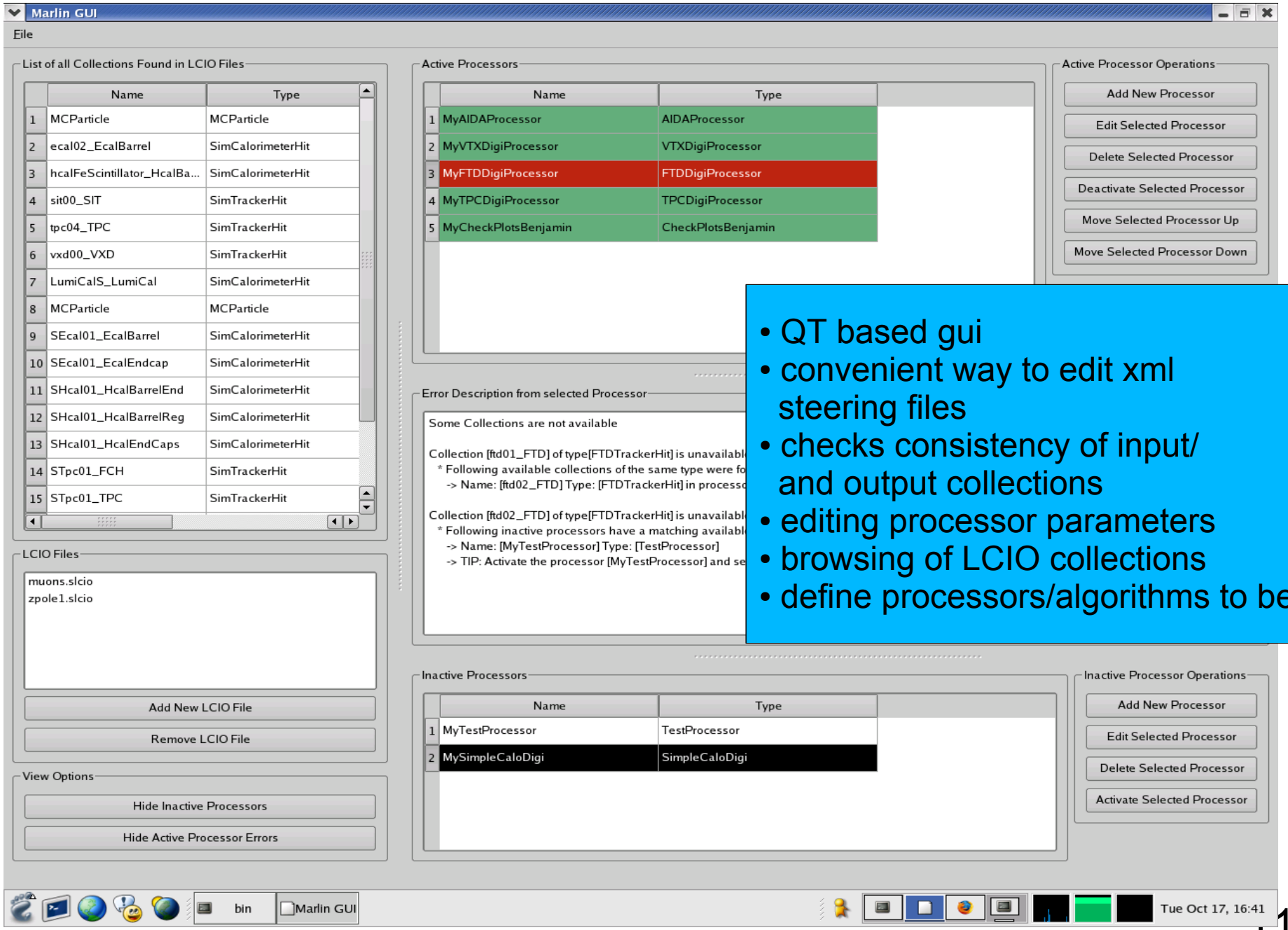


Marlin new developments

- Marlin fully functional since 2005
  - -> focus on increasing user, i.e. developer convenience
- introduced new build system: CMake
  - high level scripts for creating makefiles for common platforms Linux, MacOS, Windows
  - 'successor of GNU autotools', e.g. used by KDE
  - allows easy configuration of build process and options
- switched to shared libraries
- provide support for plugins
  - packages with processors built into shared libraries
  - loaded at program start up
  - no relinking of full application necessary
- MarlinGUI, flow charts, logging mechanism (next slides)

MarlinGUI

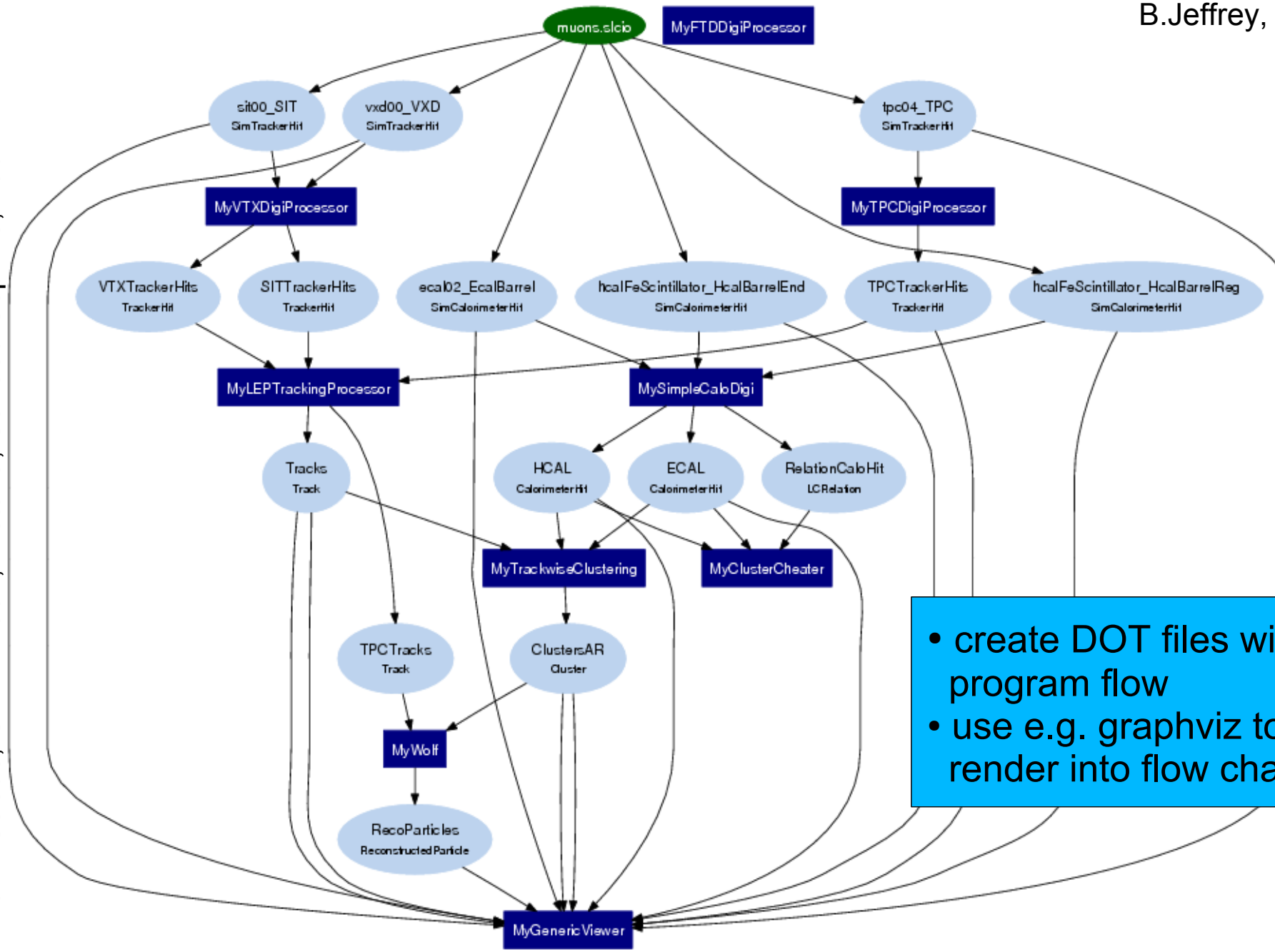
J.Engels, DESY



- QT based gui
- convenient way to edit xml steering files
- checks consistency of input/ and output collections
- editing processor parameters
- browsing of LCIO collections
- define processors/algorithms to be run

Marlin program flow charts

B.Jeffrey, Oxford



- create DOT files with program flow
- use e.g. graphviz to render into flow chart

streamlog – logging library

- standalone logging library
  - shipped with Marlin but can also be used in non-Marlin cde
  - verbosity levels: DEBUG0-4, MESSAGE0-4, WARNING0-4, ERROR0-4
  - verbosity level and current namespace can be changed on the scope level (processor name in Marlin)
  - no runtime and space overhead for DEBUG messages when compiled with NDEBUG (production)
    - per compilation unit (plugin)
  - very little overhead (if-statement) for other messages
  - simple macro for replacing std::cout (std::ostream)

```
streamlog_out( DEBUG ) << " digitizing hit : "
                       << hit->getCellID() << std::endl ;
[ DEBUG "TrackDigitizer" ]  digitizing hit : 12345678
```

Marlin supporting packages

- the Marlin framework is completed by additional packages for
- description of detector geometry: **GEAR**
  - API for high level geometry description (xml files)
  - API for detailed material description (based on geant4)
- conditions data: **LCCD**
  - provide transparent access to conditions data in LCIO collections from:
  - DB (CondDBMySQL), simple files, DB snaphots, data stream
- utility software , math libraries, ...
  - MarlinUtil – utility library
  - CED - event display
  - RAIDA – root AIDA implementation
  - CLHEP, gsl, cernlib,....

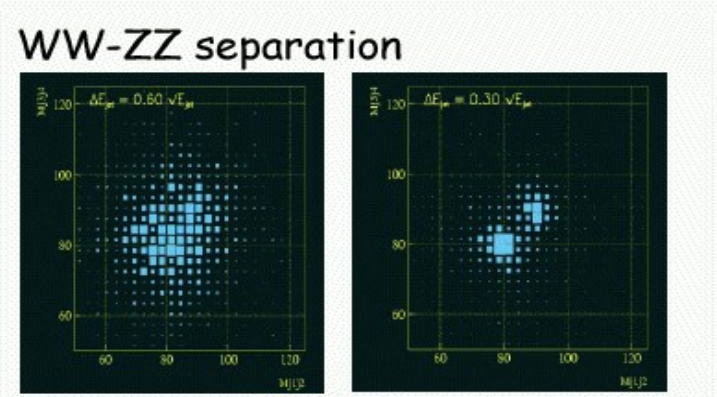
Applications of Marlin et al

- LDC detector optimization (MonteCarlo)
  - MarlinReco – full reconstruction suite
    - Digitization Calo,TPC,Silicon, PatternRecognition/Tracking, clustering, ParticleFlow algorithms
  - PandoraPFA
    - ParticleFlow algorithm
  - LCFIVertex
    - ZVTop/ZVKin vertex finding and fitting algorithms
  - various physics analyses ...
- testbeams (Data & MonteCarlo)
  - the LDC software framework has been adopted by and improved within the EUDET project for ILC testbeam infrastructure
    - Calice - calorimeter
    - MarlinTPC – TPC tracking
    - EUTelescope – pixel telescope for silicon tracking

using the same core framework for MC/offline and testbeam/online provides synergies for both worlds

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

dominant contribution (E<50 GeV):

- Hcal resolution
- confusion term

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

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

three of the four detector concepts follow the PFA paradigm to reach the necessary energy resolution

example: PandoraPFA (M.Thomson,Cambridge)

rms90

E_{JET}	$\sigma_E/E = \alpha/\sqrt{(E/GeV)}$ $\lvert\cos\theta\rvert<0.7$	σ_E/E
45 GeV	0.295	4.4 %
100 GeV	0.305	3.0 %
180 GeV	0.418	3.1 %
250 GeV	0.534	3.3 %

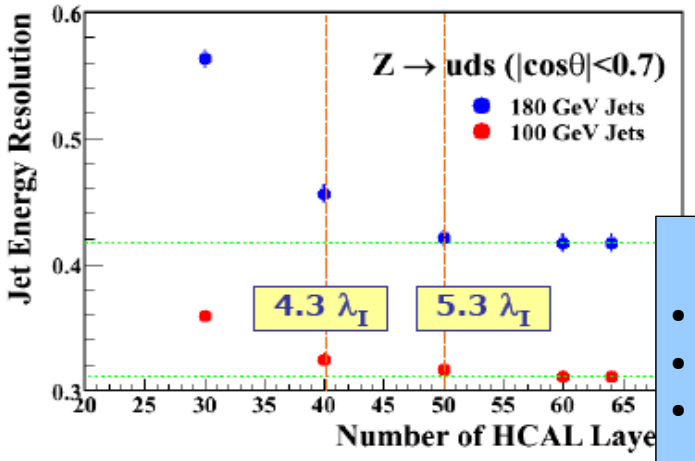
0.35 at LCWS06

For jet energies < 100 GeV
ILC "goal" reached !!!

★For a Gauge boson mass resolution of order $\Gamma_{W/Z}$

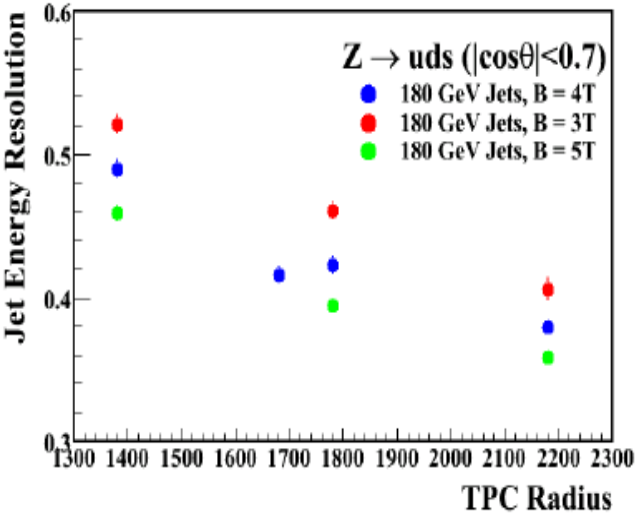
E_{jj}/GeV	$\alpha(E_j)$	σ_{Ej}/E_j
91	< 26 %	3.8 %
200	< 38 %	3.8 %
360	< 51 %	3.8 %
500	< 60 %	3.8 %

'proof of concept' for PFA @ILC
-> use for detector optimization



PFA improves with:
- thicker Hcal
- larger Tracking radius
- higher Bfield

- can use PFA for cost conscious optimization



managing ilc sw-installations

- ilc software requirements and complexity has grown
- ~30 packages with sometimes optional dependencies
- need for a tool to make installation and build process easier:
- ilcinstall (python)
  - script to install all of the LDC software in one go
  - fully configurable:
    - versions, dependencies/build options, links to existing packages/tools, e.g. root, CLHEP,...
    - define software releases
  - used for reference installations in afs (SL3/SL4)
  - together with plugin mechanism users can
    - link their package against these
    - update some packages w/o full rebuild

/afs/desy.de/group/it/ilcsoft/v01-01

Summary & Outlook

- Marlin is a modular C++ application framework for the analysis of ILC data based on LCIO, used in
  - LDC detector optimization studies (particle flow)
  - ILC testbeam programs
- recently the focus has been on improving the usability
  - MarlinGUI, installation scripts, runtime extensions
- actively used by the community, eg.
- PandoraPFA package demonstrates the PFA concept
- ongoing and upcoming testbeams require further developments and adaption (LCIO-persistency)

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

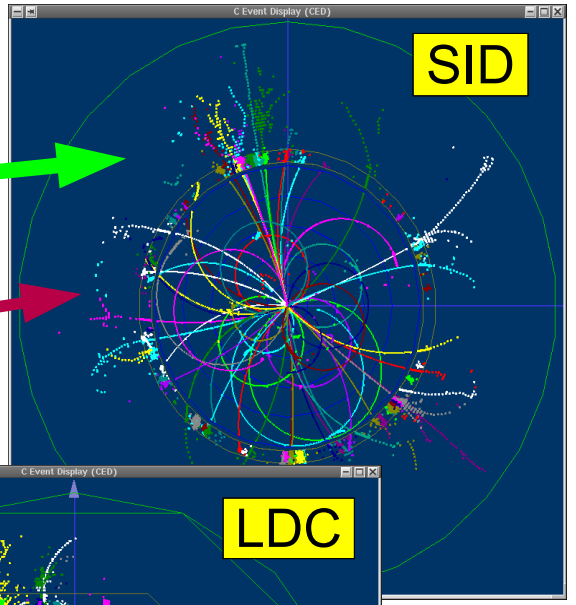
additional material

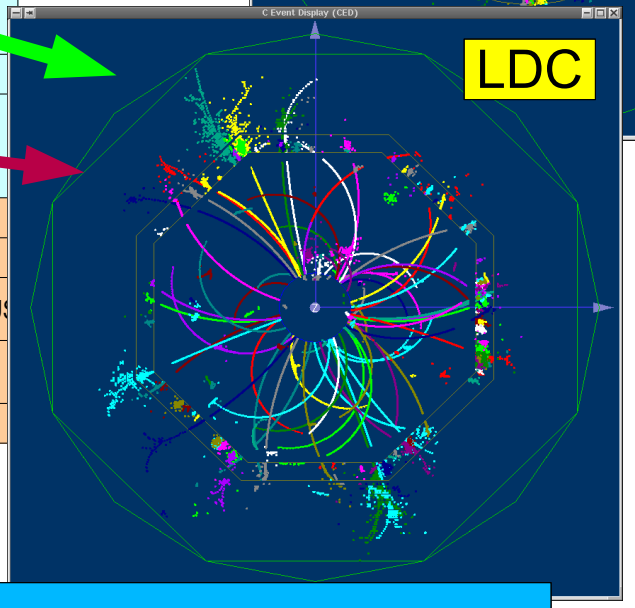
interoperability with other ILC SW

	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++	ASCII, LCIO	EU
Brahms-Sim	Geant3 – full simulation	TeslaTDR	Fortran	LCIO	EU
SLIC	full simulation – Geant4	SiD, flexible	C++	LCIO	US
LCDG4	full simulation – Geant4	SiD, flexible	C++	SIO, LCIO	US
Jupiter	full simulation – Geant4	JLD (GDL)	C++	Root (LCIO)	AS
Brahms-Reco	reconstruction framework (most complete)	TeslaTDR	Fortran	LCIO	EU
Marlin	reconstruction and analysis application framework	Flexible	C++	LCIO	EU
hep.lcd	reconstruction framework	SiD (flexible)	Java	SIO	US
org.lcsim	reconstruction framework (under development)	SiD (flexible)	Java	LCIO	US
Jupiter-Satelites	reconstruction and analysis	JLD (GDL)	C++	Root	AS
LCCD	Conditions Data Tookit	All	C++	MySQL, LCIO	EU
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, eprep,LCIO,..	US,EU

simulation

reconstruction





- almost all international ILC software uses/based on LCIO
- provides some basis for interoperability
- possibly develop tighter common, multiple language (Java/C++) framework in the future !?

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>
```

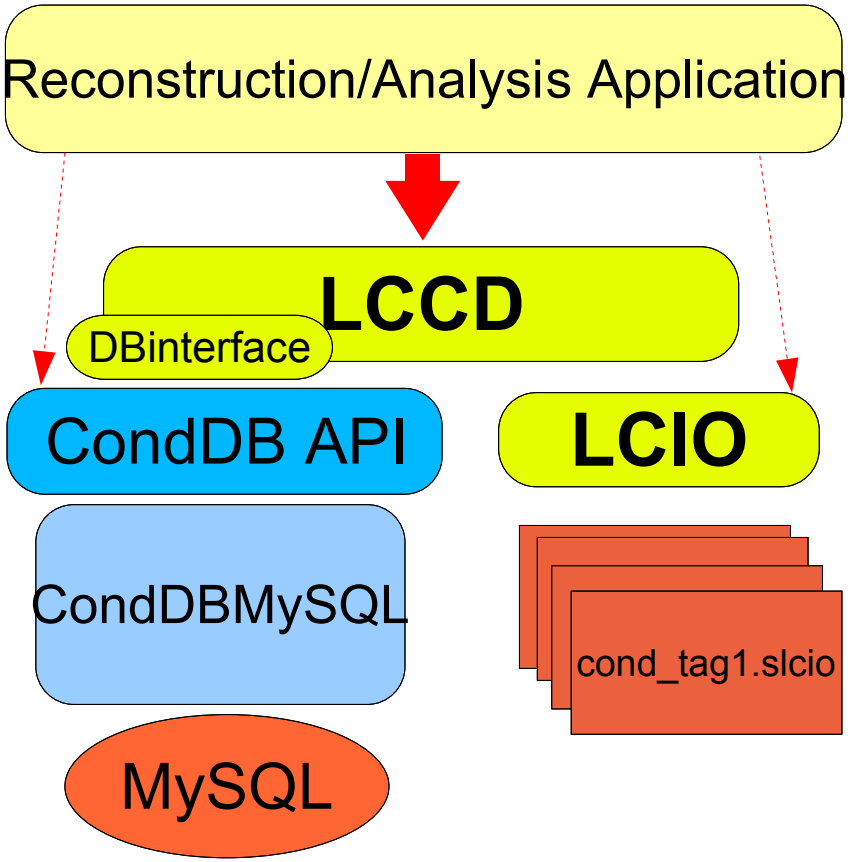
compatible with US – compact format

- 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 – planned

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 for the conditions data of the ongoing ILC testbeam studies

software package dependencies