LCIO & Marlin

Overview and Status

LC Simulation Mini Workshop

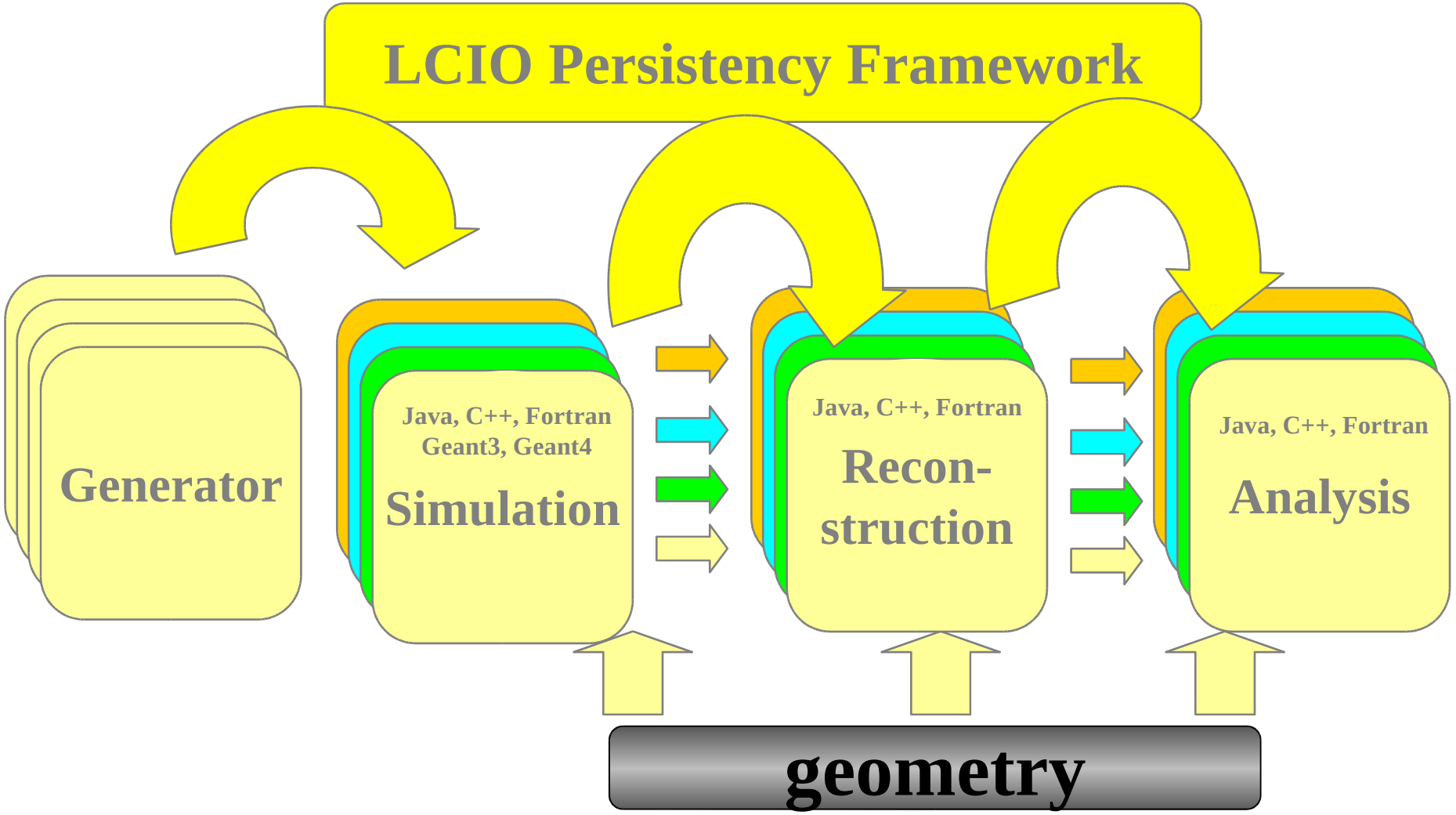
DESY, Hamburg 9.-10 Dec. 2004

Frank Gaede, DESY -IT-

Outline

- LCIO
  - Introduction
  - Implementation/Design
  - Data Model
  - Status
- Marlin
  - Introduction
  - Implementation
  - Usage
  - Status
- Summary

Motivation for LCIO

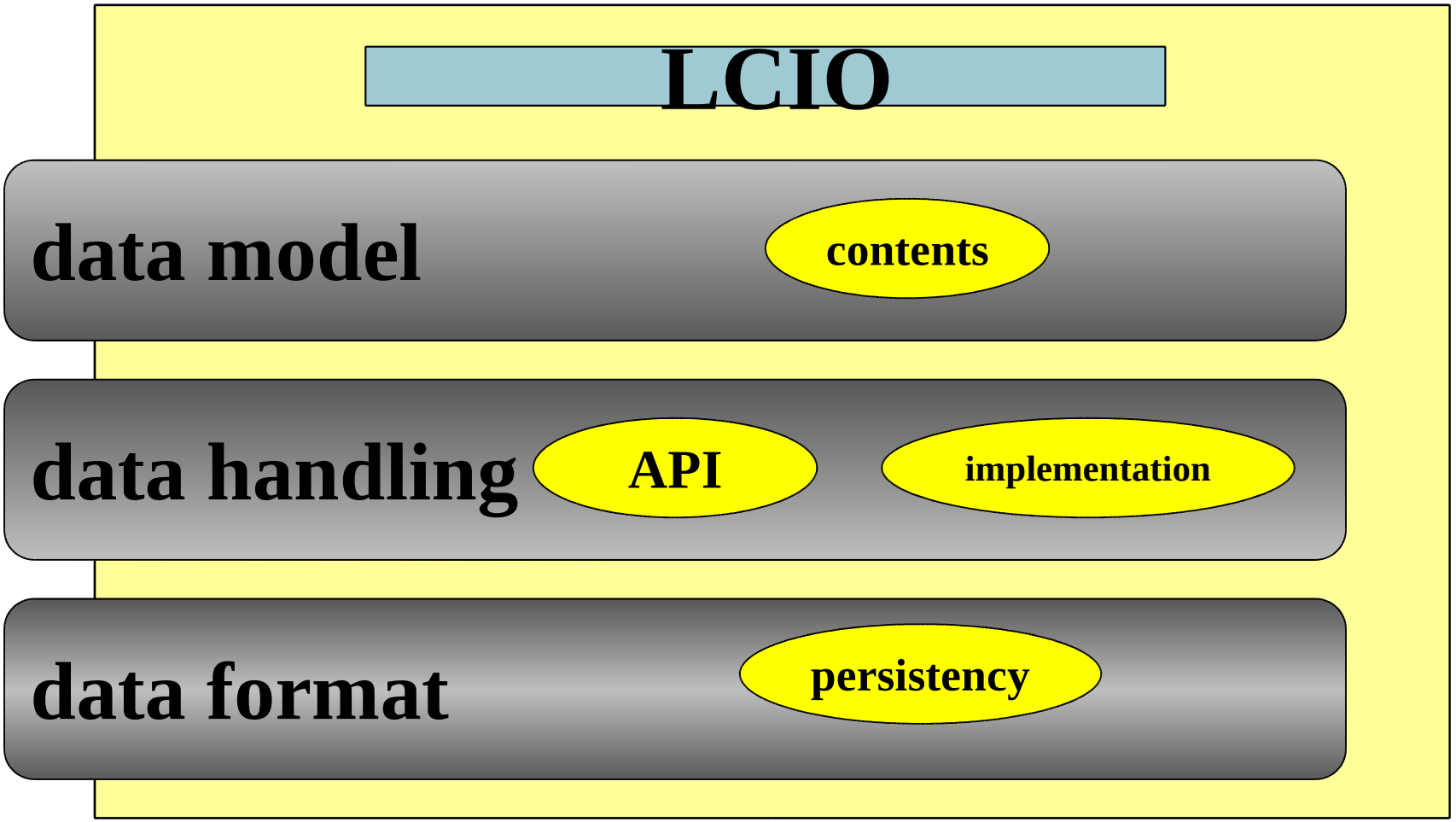


LCIO Project Overview

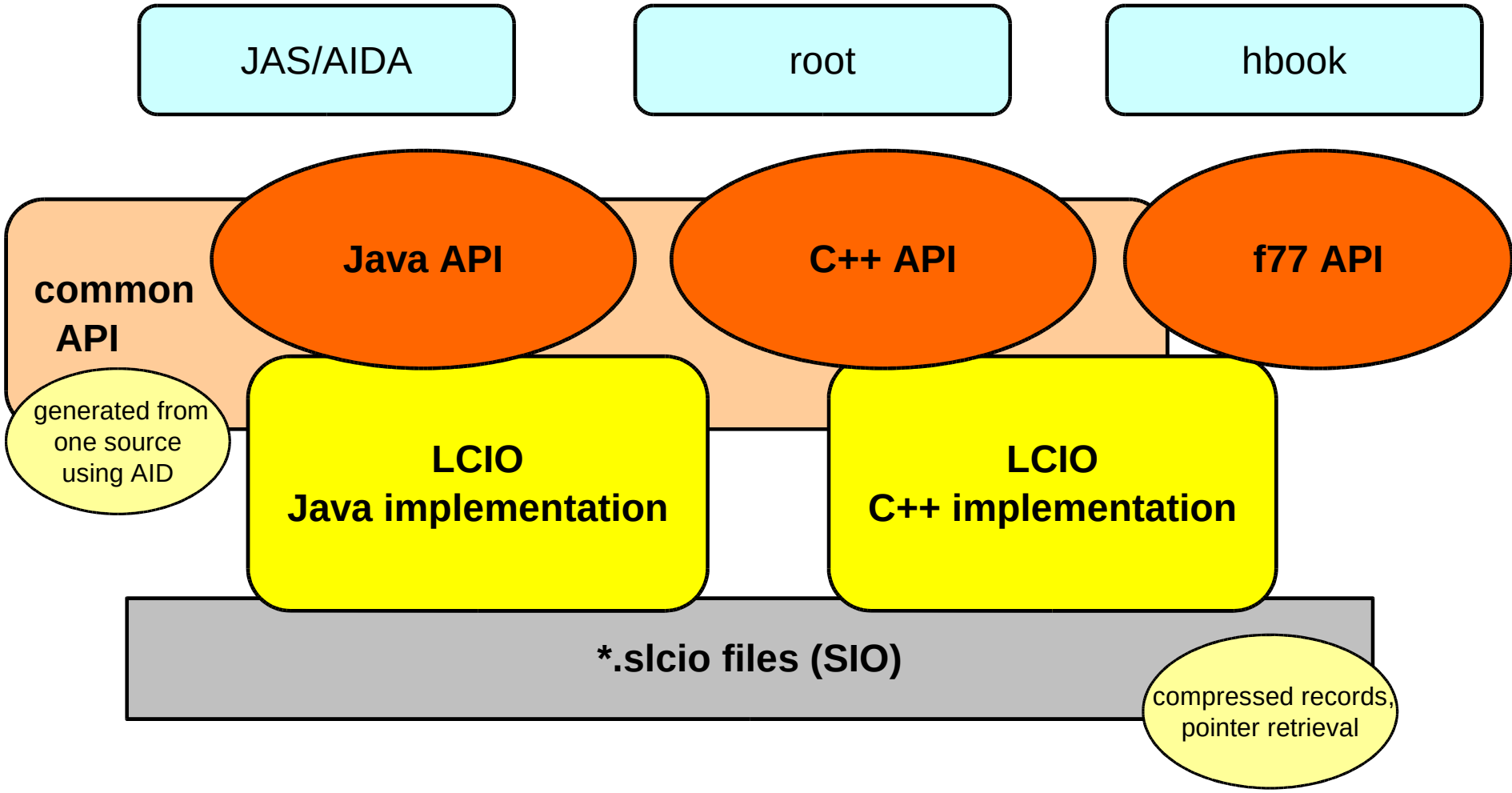
- DESY/SLAC (LLR) project team:
  - provide common basis for ILC software
- Requirements:
  - need Java, C++ and f77 (!) API
  - extendable data model for current and future simulation and testbeam studies
  - user code separated from concrete data format
  - easy to adapt LCIO in existing applications
  - no dependency on other frameworks

  -> keep it simple & lightweight

LCIO persistency framework



LCIO SW-Architecture



Data Format (persistency): SIO

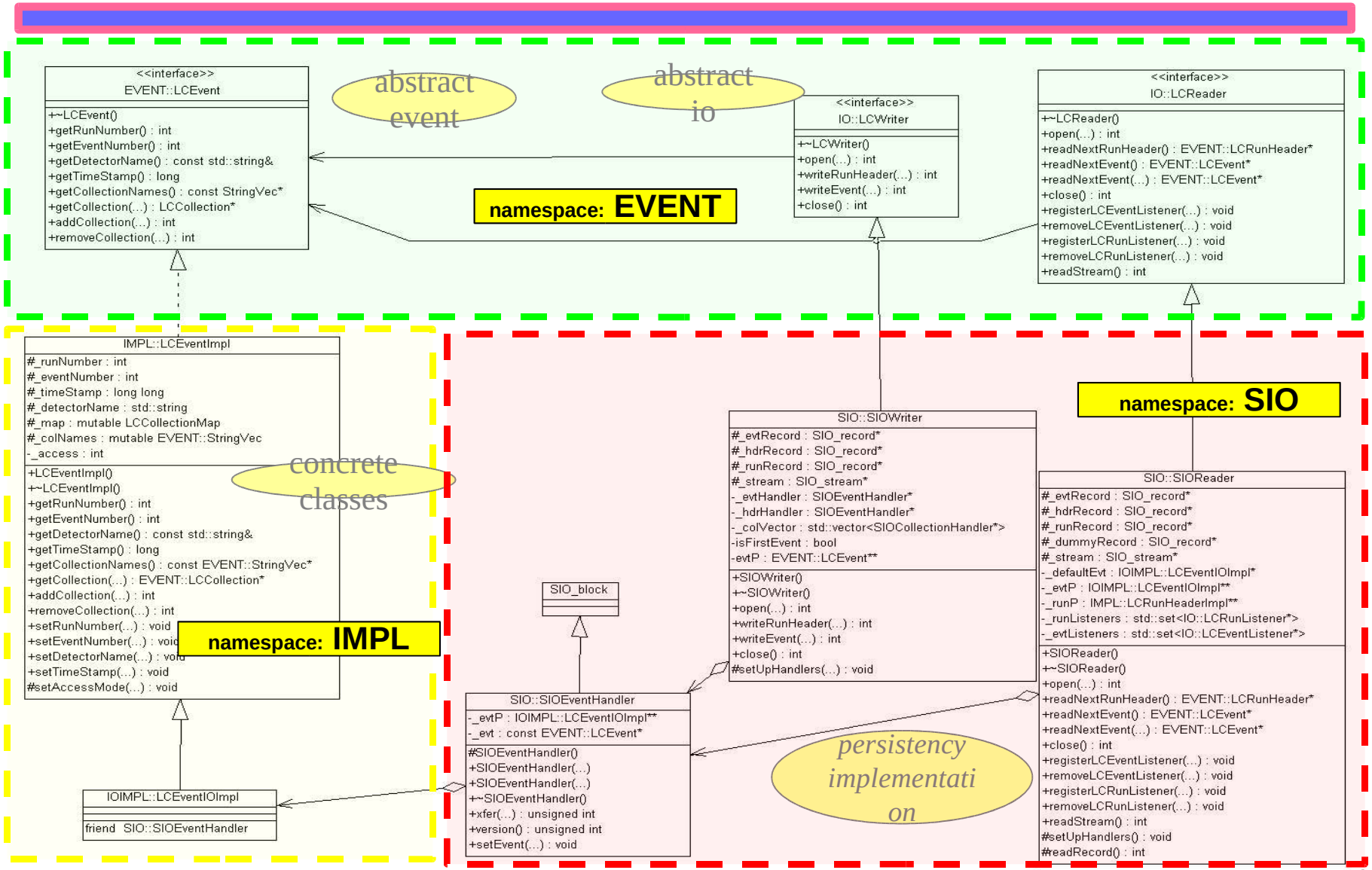
- SIO: Simple Input Output
  - developed at SLAC for NLC simulation
  - already used in hep.lcd framework
- features:
  - on the fly data compression ✪
  - some OO capabilities, e.g. pointers ✪
  - C++ and Java implementation available ✪
  - no direct access ✯

    -> use fast skip ✯

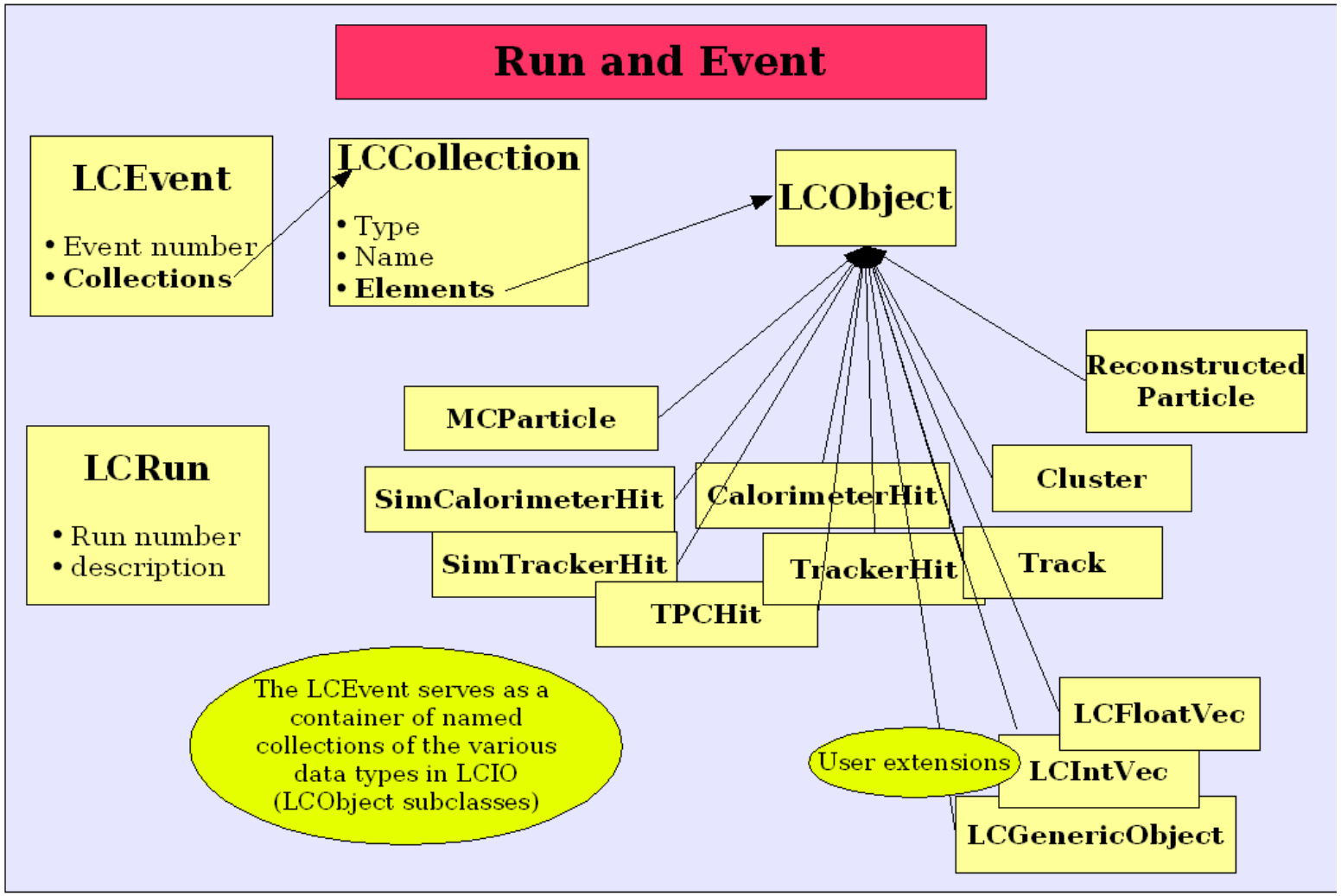
Implementation details

- common Java, C++ interface:
  - AID-tool from freehep.org
  - define interfaces in Java-like language with C++ extensions
  - -> generates files with Java interfaces
  - -> generates C++ header files with pure abstract base classes
- Fortran interface:
  - use C++-wrapper functions and **cfortran.h**
  - one function for every class member function
  - use integers to store pointers !
  - -> OO-like code in fortran

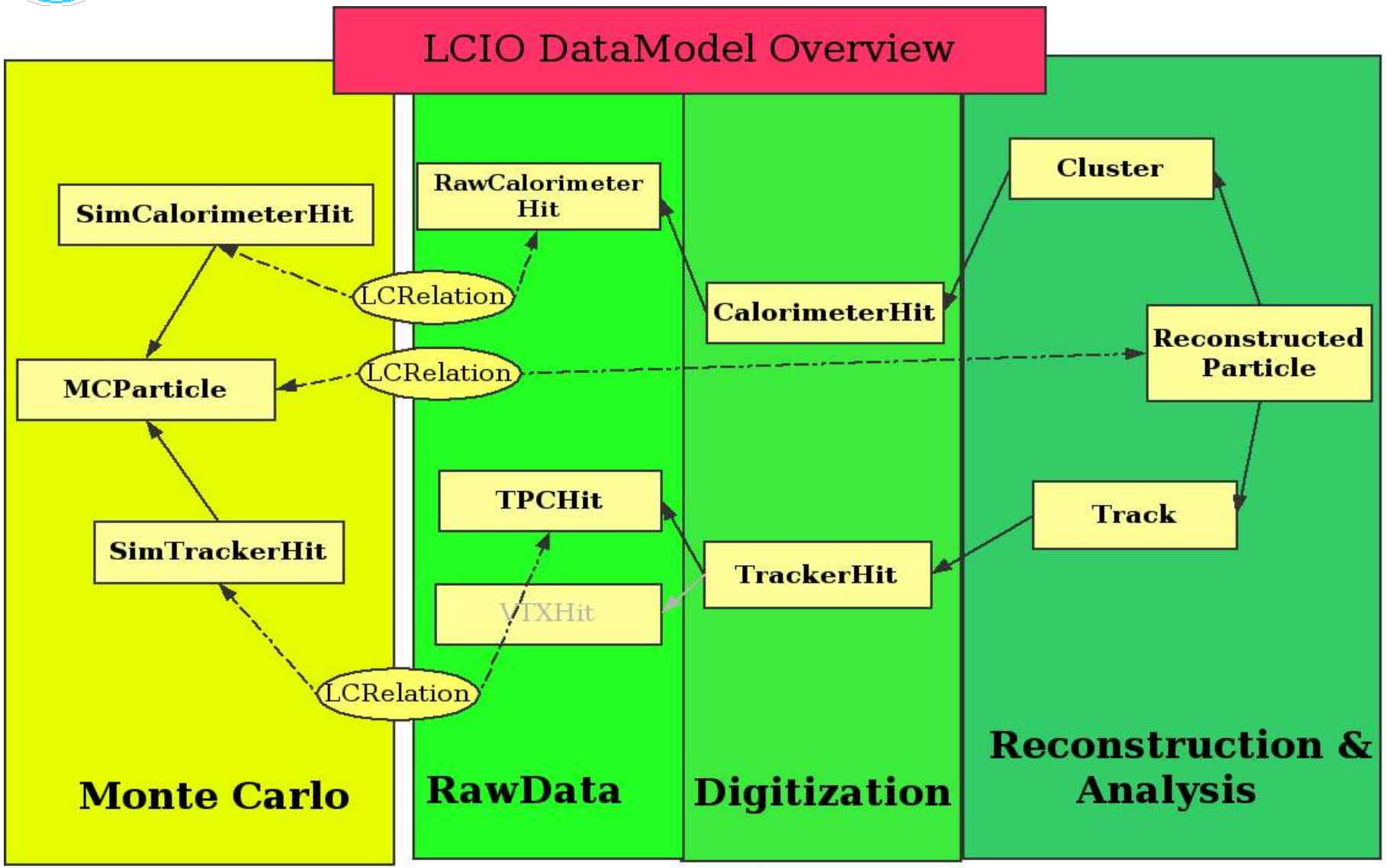
LCIO Design


<<interface>>
EVENT::LCEvent
+~LCEvent()
+getRunNumber() : int
+getEventNumber() : int
+getDetectorName() : const std::string&
+getTimeStamp() : long
+getCollectionNames() : const StringVec*
+getCollection(...) : LCCollection*
+addCollection(...) : int
+removeCollection(...) : int
abstract event
abstract io
<<interface>>
IO::LCWriter
+~LCWriter()
+open(...) : int
+writeRunHeader(...) : int
+writeEvent(...) : int
+close() : int
<<interface>>
IO::LCReader
+~LCReader()
+open(...) : int
+readNextRunHeader() : EVENT::LCRunHeader*
+readNextEvent() : EVENT::LCEvent*
+readNextEvent(...) : EVENT::LCEvent*
+close() : int
+registerLCEventListener(...) : void
+removeLCEventListener(...) : void
+registerLCRunListener(...) : void
+removeLCRunListener(...) : void
+readStream() : int
namespace: EVENT
IMPL::LCEventImpl
#_runNumber : int
#_eventNumber : int
#_timeStamp : long long
#_detectorName : std::string
#_map : mutable LCCollectionMap
#_colNames : mutable EVENT::StringVec
-_access : int
+LCEventImpl()
+~LCEventImpl()
+getRunNumber() : int
+getEventNumber() : int
+getDetectorName() : const std::string&
+getTimeStamp() : long
+getCollectionNames() : const EVENT::StringVec*
+getCollection(...) : EVENT::LCCollection*
+addCollection(...) : int
+removeCollection(...) : int
+setRunNumber(...) : void
+setEventNumber(...) : void
+setDetectorName(...) : void
+setTimeStamp(...) : void
#setAccessMode(...) : void
concrete classes
namespace: IMPL
IOIMPL::LCEventIOImpl
friend SIO::SIOEventHandler
SIO_block
SIO::SIOEventHandler
-_evtP : IOIMPL::LCEventIOImpl**
-_evt : const EVENT::LCEvent*
#SIOEventHandler()
+SIOEventHandler(...)
+SIOEventHandler(...)
+~SIOEventHandler()
+xfer(...) : unsigned int
+version() : unsigned int
+setEvent(...) : void
SIO::SIOWriter
#_evtRecord : SIO_record*
#_hdrRecord : SIO_record*
#_runRecord : SIO_record*
#_stream : SIO_stream*
-_evtHandler : SIOEventHandler*
-_hdrHandler : SIOEventHandler*
-_colVector : std::vector<SIOCollectionHandler*>
-isFirstEvent : bool
-evtP : EVENT::LCEvent**
+SIOWriter()
+~SIOWriter()
+open(...) : int
+writeRunHeader(...) : int
+writeEvent(...) : int
+close() : int
#setUpHandlers(...) : void
namespace: SIO
SIO::SIOReader
#_evtRecord : SIO_record*
#_hdrRecord : SIO_record*
#_runRecord : SIO_record*
#_dummyRecord : SIO_record*
#_stream : SIO_stream*
-_defaultEvt : IOIMPL::LCEventIOImpl*
-_evtP : IOIMPL::LCEventIOImpl**
-_runP : IMPL::LCRunHeaderImpl**
-_runListeners : std::set<IO::LCRunListener*>
-_evtListeners : std::set<IO::LCEventListener*>
+SIOReader()
+~SIOReader()
+open(...) : int
+readNextRunHeader() : EVENT::LCRunHeader*
+readNextEvent() : EVENT::LCEvent*
+readNextEvent(...) : EVENT::LCEvent*
+close() : int
+registerLCEventListener(...) : void
+removeLCEventListener(...) : void
+registerLCRunListener(...) : void
+removeLCRunListener(...) : void
+readStream() : int
#setUpHandlers() : void
#readRecord() : int
persistency implementation

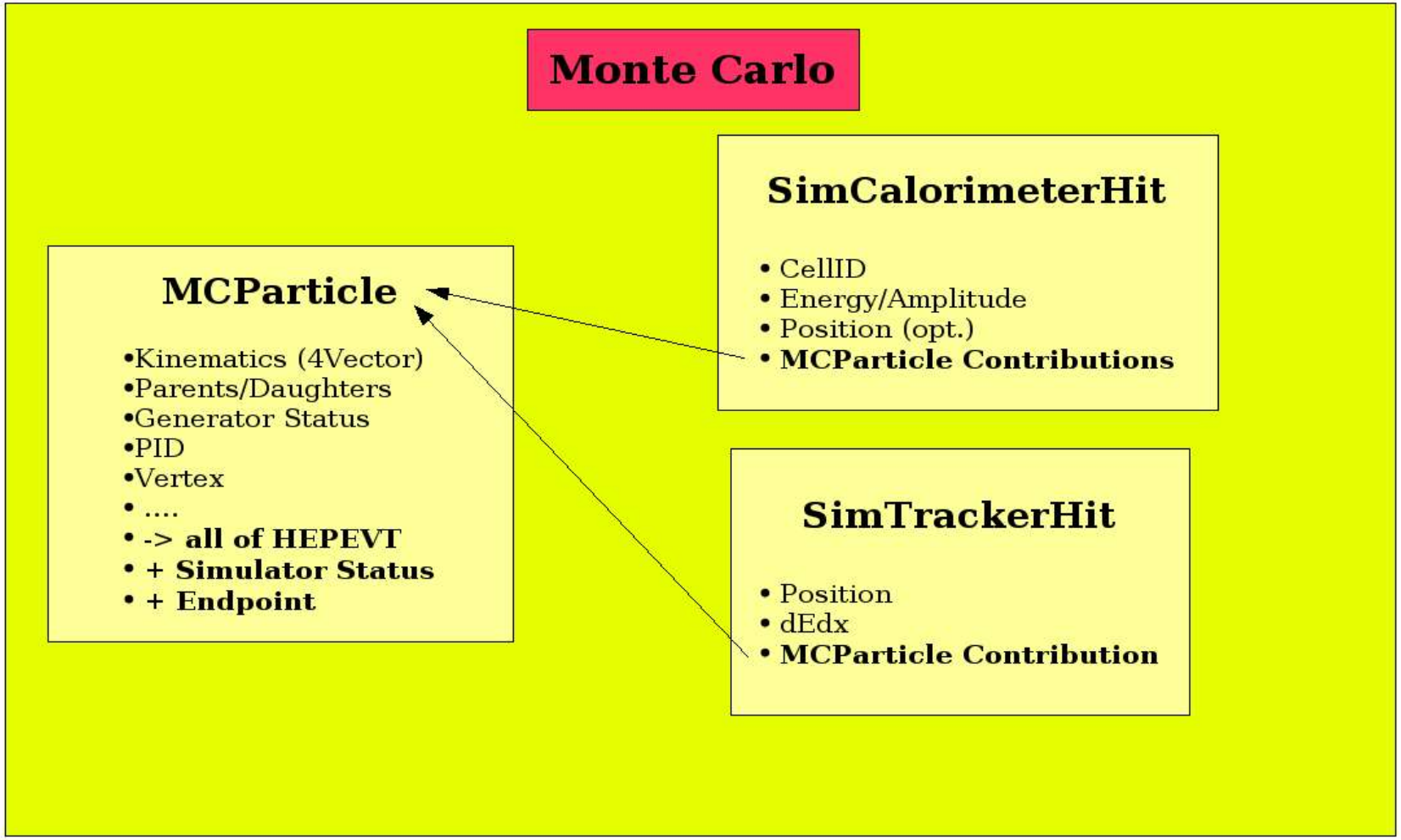
Data Model I


Run and Event
LCEvent
• Event number
• Collections
LCCollection
• Type
• Name
• Elements
LCObject
LCRun
• Run number
• description
MCParticle
ReconstructedParticle
SimCalorimeterHit
CalorimeterHit
Cluster
SimTrackerHit
TrackerHit
Track
TPCHit
The LCEvent serves as a container of named collections of the various data types in LCIO (LCObject subclasses)
User extensions
LCFloatVec
LCIntVec
LCGenericObject

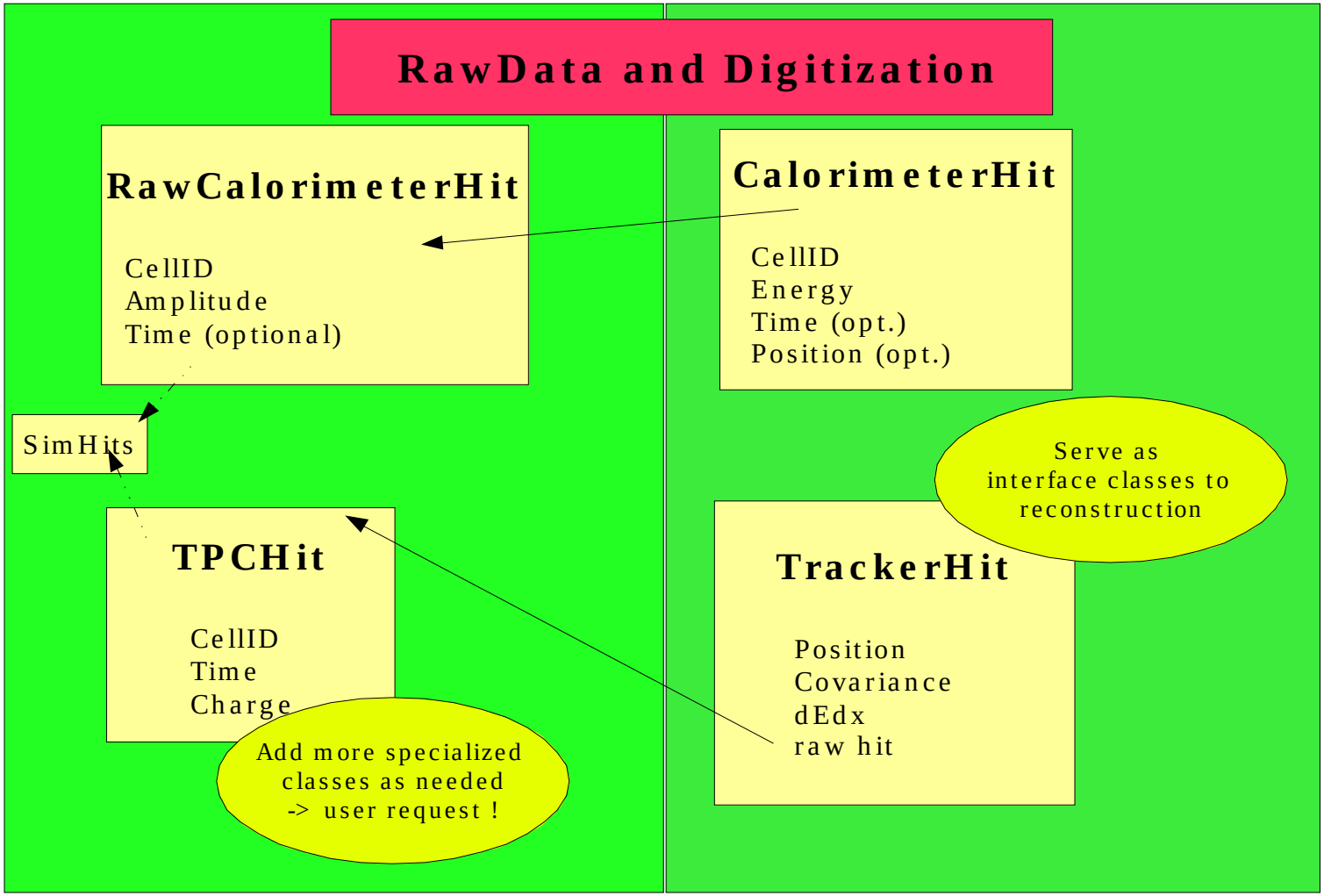
Data Model II



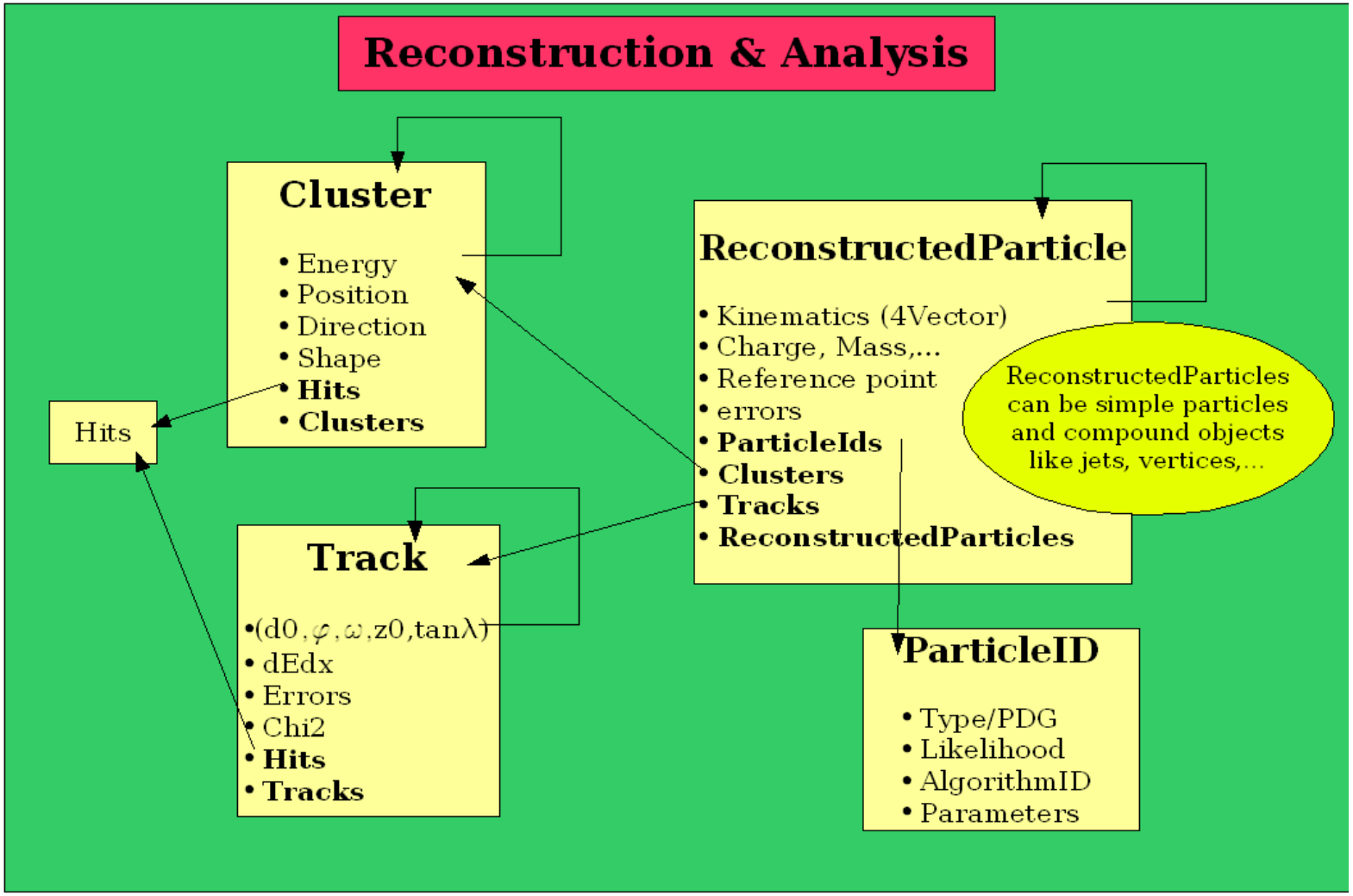
Data Model III



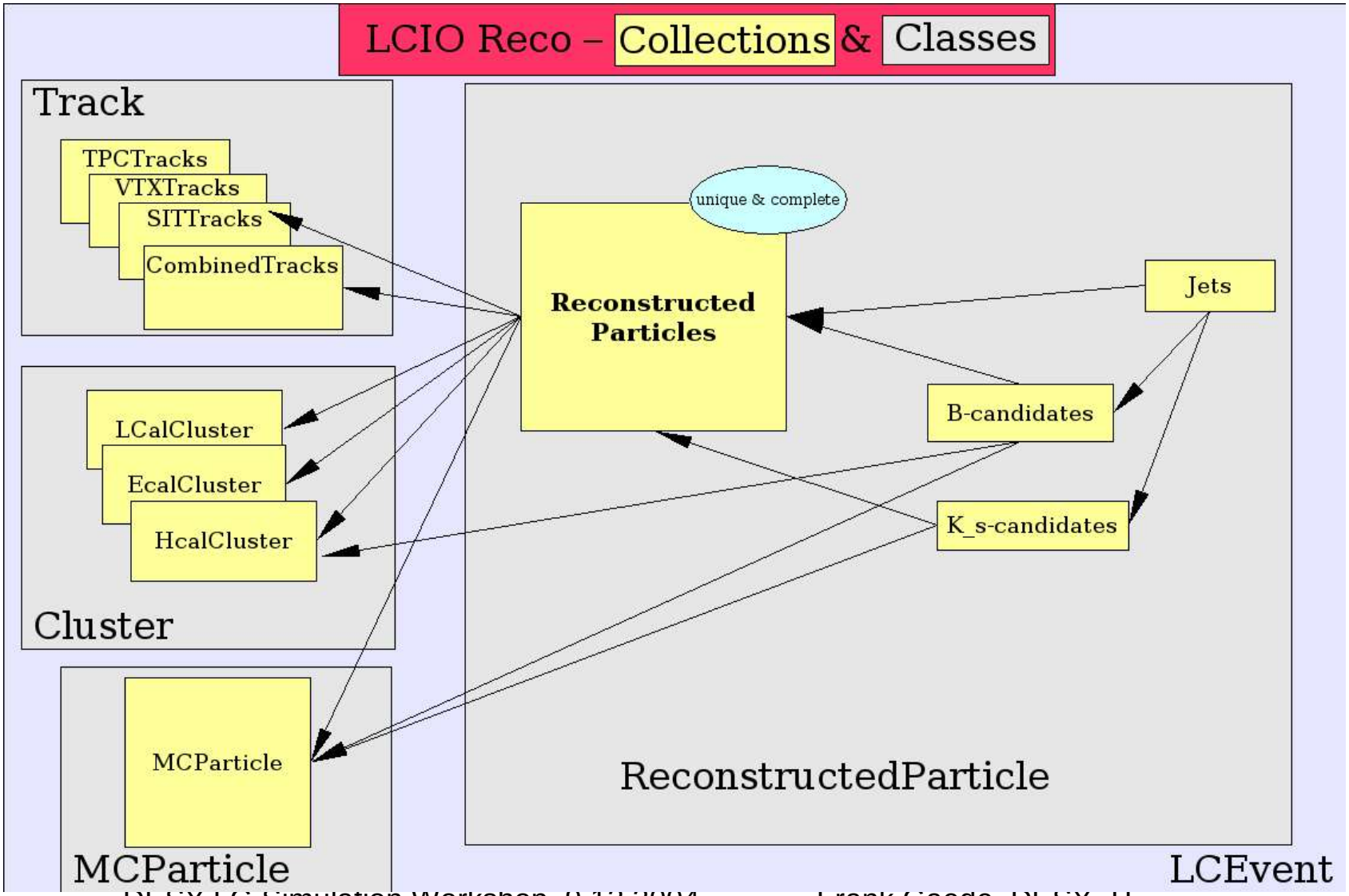
Data Model IV



Data Model V


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
•(d0,φ,ω,z0,tanλ)
• dEdx
• Errors
• Chi2
• Hits
• Tracks
ParticleID
• Type/PDG
• Likelihood
• AlgorithmID
• Parameters

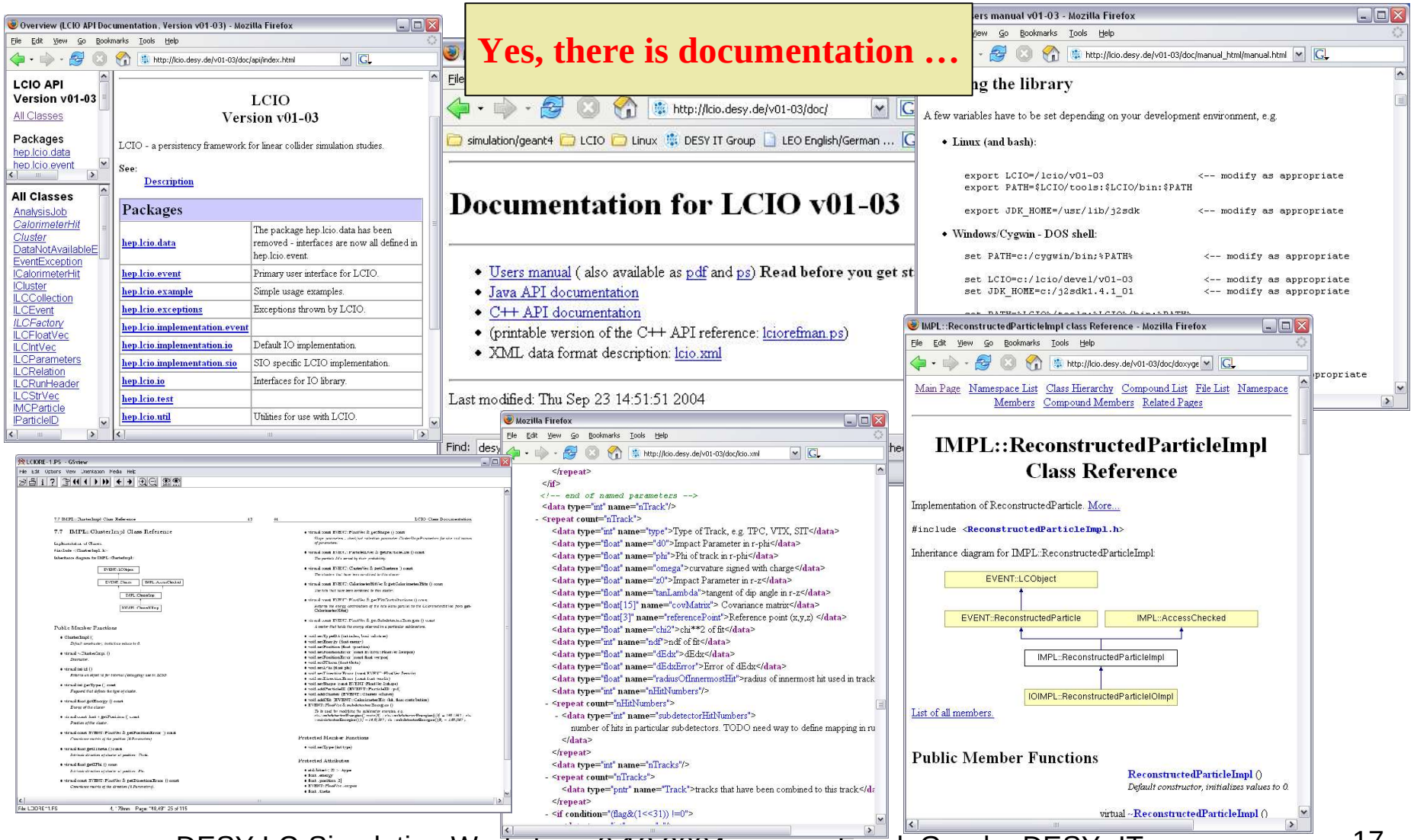
Example of LCIO Event



LCIO status – features

- first official release v01-00 (Nov 2003)
  - simulation data model (MCParticles+ SimHits)
- release v01-03 (Sep 2004)
  - reconstruction model (Rec.Particle, Tracks, Clusters)
  - user extensions (LCGenericObject)
  - raw data classes
  - support for CLHEP four vectors
  - transient data collections
  - StdHep interface
  - run, event and collection parameters
    -> store meta data describing what is in the collections
- release v01-03-01 (last week)
  - minor bug fixes – 100% compatible with 1.3

LCIO online documentation



LCIO on the web

- LCIO homepage: http://lcio.desy.de
  - downloads and documentation
- LCIO forum at: http://forum.linearcollider.org
  - user/developer questions and comments
  - discussions on new developments
- LCIO bug reports at: http://bugs.freehep.org
  - bug report and new feature requests

LCIO Users

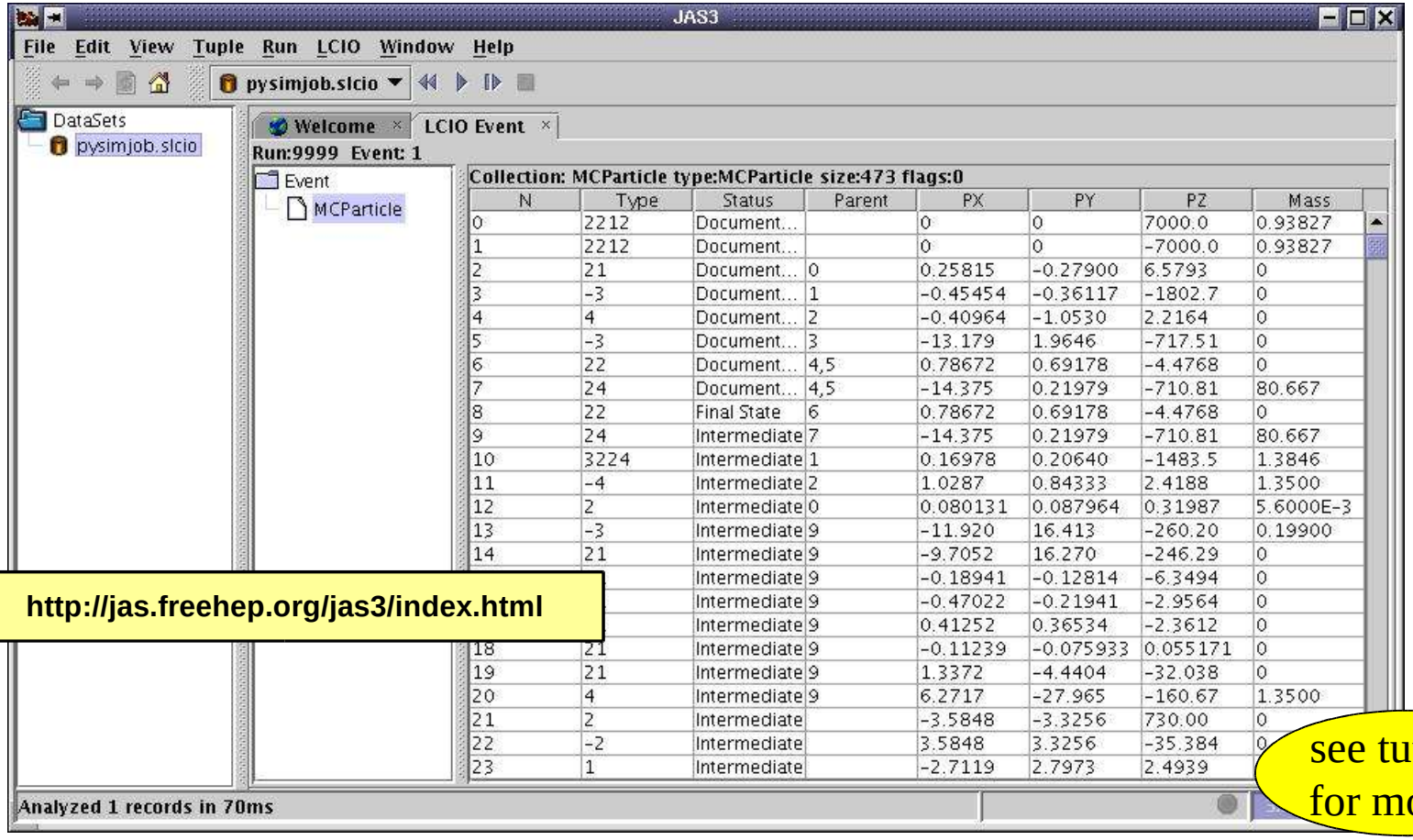
- Geant4 Full simulations:
  - LCDG4, LCS, Mokka*, Jupiter (planned)
- Reconstruction:
  - org.lcsim, Brahms*, Marlin*
- Fast simulation
  - Lelaps, Simdet*
- Testbeam
  - Calice: Ecal, Hcal PPTs
  - TPC prototypes
- Analysis Tools
  - JAS3*: file browser, code wizard
  - WIRED*: generic event display (picking of LCIO objects)

-> about to become de facto standard for ILC-software

(*programs available on Workshop DVD)

JAS3 – LCIO plugin

JAS3 provides native interfaces to LCIO:
browser, code wizard, event display WIRED



http://jas.freehep.org/jas3/index.html

see tutorial for more…

LCIO for transient data

- The LCEvent can be used as container for transient data in an application, e.g. reconstruction
- Application will call list of modules that read existing collections from the LCEvent and add resulting new Collections
- LCIO has (Event/Run)-Listener classes that can serve as base classes for modules
- easy to define an application framework based on LCIO for reconstruction and analysis:
  - org.lcsim (Java) , **Marlin (C++)**

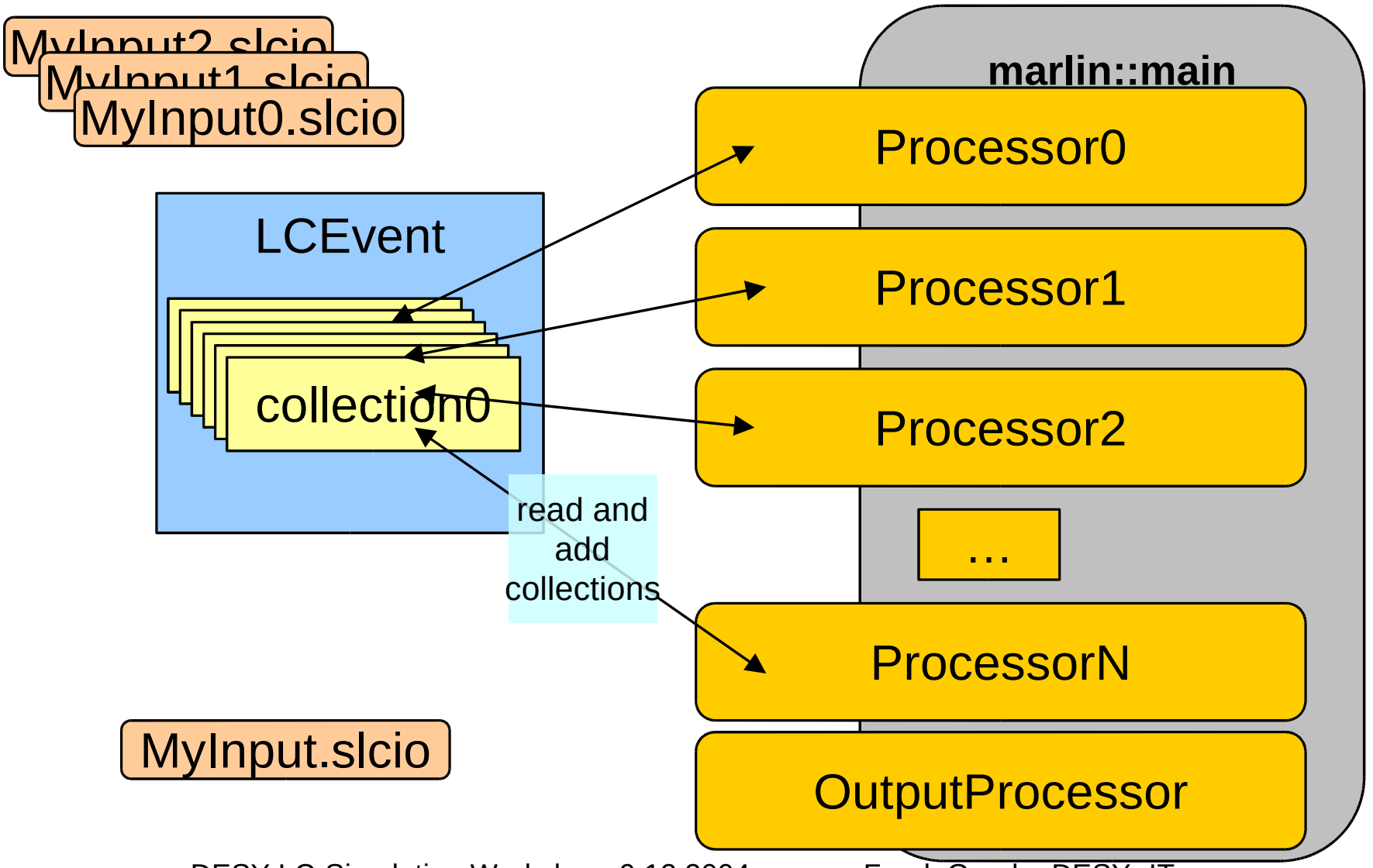
Marlin - Introduction

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

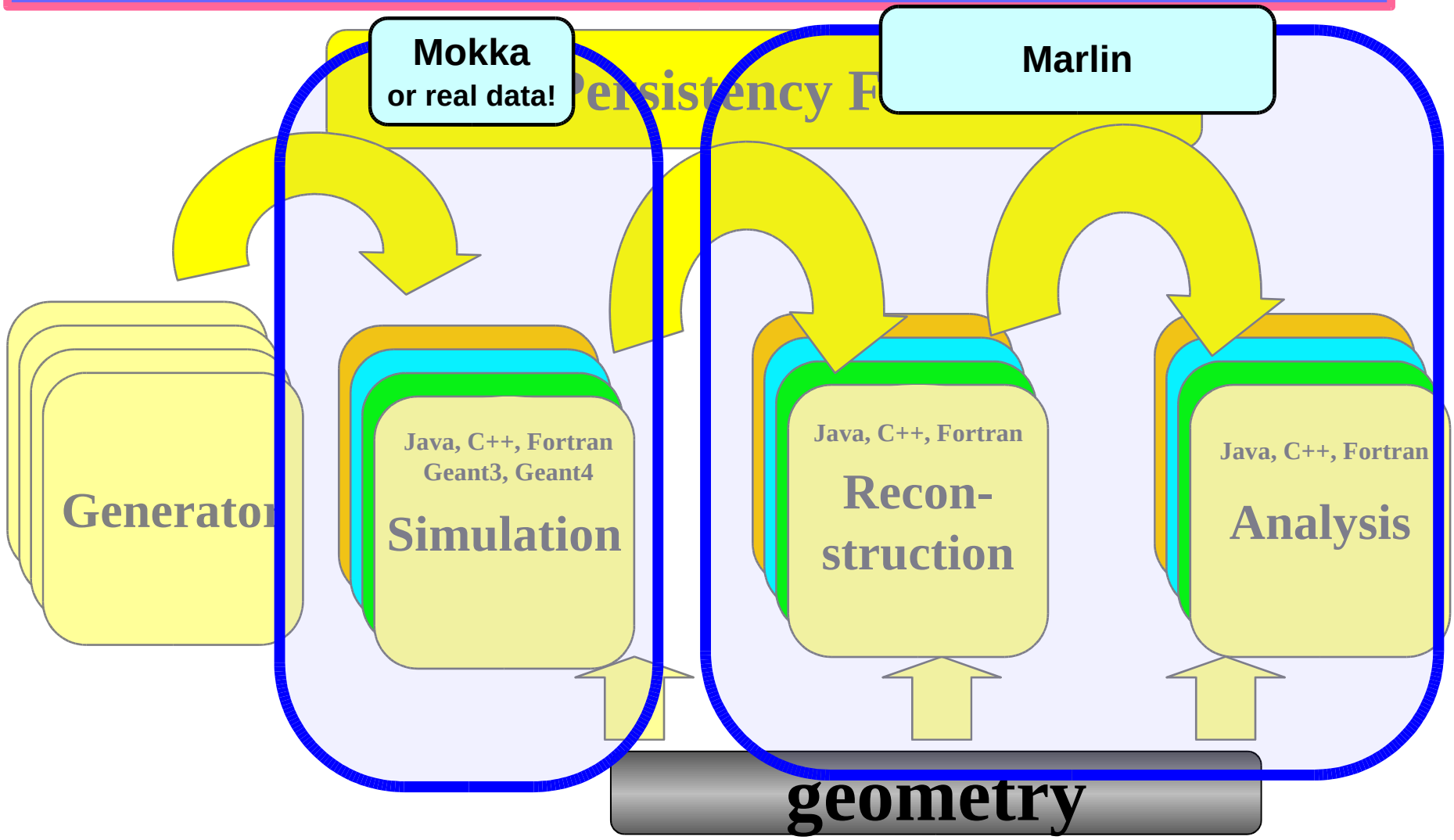
- modular software framework for the analysis and reconstruction of LCIO data
- uses LCIO as transient data model
- provides simple user steering:
  - user defined variables for each Processor
  - input/output files
- provide main program !
- software modules called **Processors**

(for similarity with org.lcsim – U.S Java based reconstruction)

Processors and the LCIOEvent



Scope for Marlin



LCIOProcessor

- LCIOProcessor: base class for all Processors
- provides hooks (callbacks) for user actions:
  - **init()**
    - called once at program start
    - use to initialize histograms, counters, etc.
  - **processRunHeader(LCRunHeader* run)**
    - called for bookkeeping – new run conditions ?
  - **processEvent(LCEvent* evt)**
    - the working horse – this where the analysis takes place
  - **check(LCEvent* evt)**
    - optional method, e.g. for checkplots, consistency checks…
  - **end()**
    - called once at end of job
    - write out histos, …

Provided Processors

- AIDAProcessor: Histogram module
  - based on AIDA (Abstract Interface for Data Analysis)
    - AIDA implementations, e.g. JAIDA/ AIDAJNI, OpenScientist,…
  - easy to create histograms, clouds and n-tuples
  - one folder per processor
  - compressed xml-data files (any AIDA tool)
    - can use JAS3 to view histograms
  - root files (need OpenScientist)
    - does not depend on root but can use root to view histograms
  - NB: users not required to use AIDAProcessor
- OutputProcessor
  - simply writes out the current event (no user code needed)

How to write your own Processor(s)

- inherit from LCIOProcessor
  - implement callbacks, e.g. init(), processEvent()
- register processor parameters
  - with name, description and default value
- edit steering file (see next slide)
  - use ‘MyMarlin –l ‘ for help
- rest is done by the framework !
- example template exists in Marlin cvs
  - $MARLIN/examples/mymarlin

  -> see tutorial this afternoon

Marlin steering files

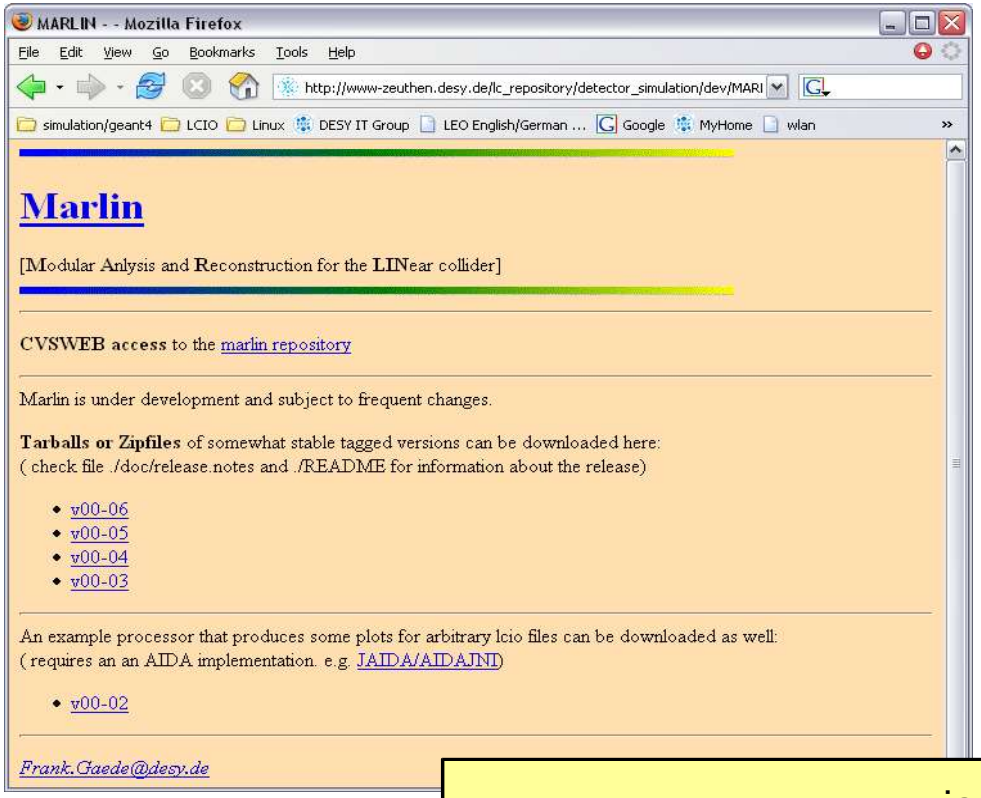
- global parameters, e.g.
  - LCIOInputFiles – the files to read in
  - ActiveProcessors – define which processors to run
  - SupressCheck – don't call check()
  - MaxRecordNumber
- processor specific parameters:
  - ProcessorType
  - parameters registered with the processor
    - named int, float and string variables or vectors
    - e.g.: InputCollectionName, CutValue, AlgorithmType

-> A Marlin application is fully configured through the steering file (no code change needed) !

Marlin TO DO List

- error handling
  - log files
  - error/warning messages
- naming convention for common parameters, e.g. InputCollectionName, OutputCollectionName
- convention for passing user data between Processors, e.g.:
  - as LCCollections of LCObjects
  - as global objects (singletons)
- some logic to control execution and I/O of events
  - e.g. a Processor might want to decide that the event is not worth processing then the rest of the Processors should not be called …
- more detailed documentation (so far APIdoc & Readme)
- additional functionality
  - user feedback needed !

Where to find Marlin



- download via cvs-web
  - tar-balls of releases
- cvs checkout
  - anonymous checkout
  - from new DESY ILC-CVS

(see talk of H.Vogt)

via
http://www.desy.de/~gaede/marlin
or
http://www-zeuthen.desy.de/linear_collider

Existing Processors

- HCalPPT Ganging (R.Poeschl)
- HCalPPT Digitization (G.Lima)
- LCLeptonFinder (J.Samson, on WS-DVD)
  - demoanalysis
- HCalAna (P. Melchior/F.Gaede)
- Checkplots (via Marlin cvs, on WS-DVD)

-> can provide example code

Summary

- LCIO is a persistency framework and data model for the ILC
  - Provides Java, C++ and f77 API
  - simulation and reconstruction data model
  - user extensions
- Future plans:
  - react to user requests
  - provide convenient methods
    - handling of relationships
    - handling of meta data
  - bug fix patches
- Marlin is a simple framework for analyzing LC data
  - based on LCIO (transient and persistent)
  - provides simple base class for user Processors
  - convenient steering for run control and user parameters
  - no need to deal with I/O or write main program
- publicly available via cvs-web
- AIDAProcessor for easy creation of Histograms

Marlin is meant as a starting point for a common SW framework for the LC. If people use this for their analysis/reconstruction it should be easy to adapt their code to new versions of the framework as they are developed.