How to use LCIO

Frank Gaede
DESY

ILC Software Workshop
Cambridge, April 4-6, 2006

LCIO overview

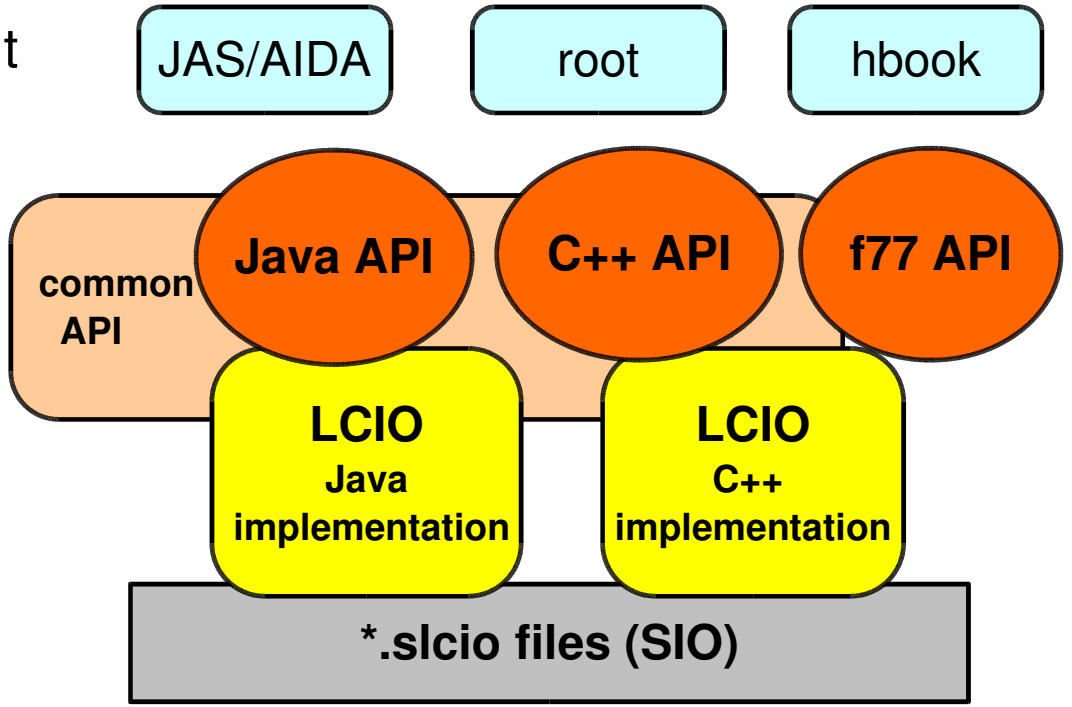
- DESY and SLAC joined project:
  - provide common basis for ILC software
- Features:
  - Java, C++ and f77 (!) API
  - extensible data model for current and future simulation and testbeam studies
  - user code separated from concrete data format
  - no dependency on other frameworks

simple & lightweight

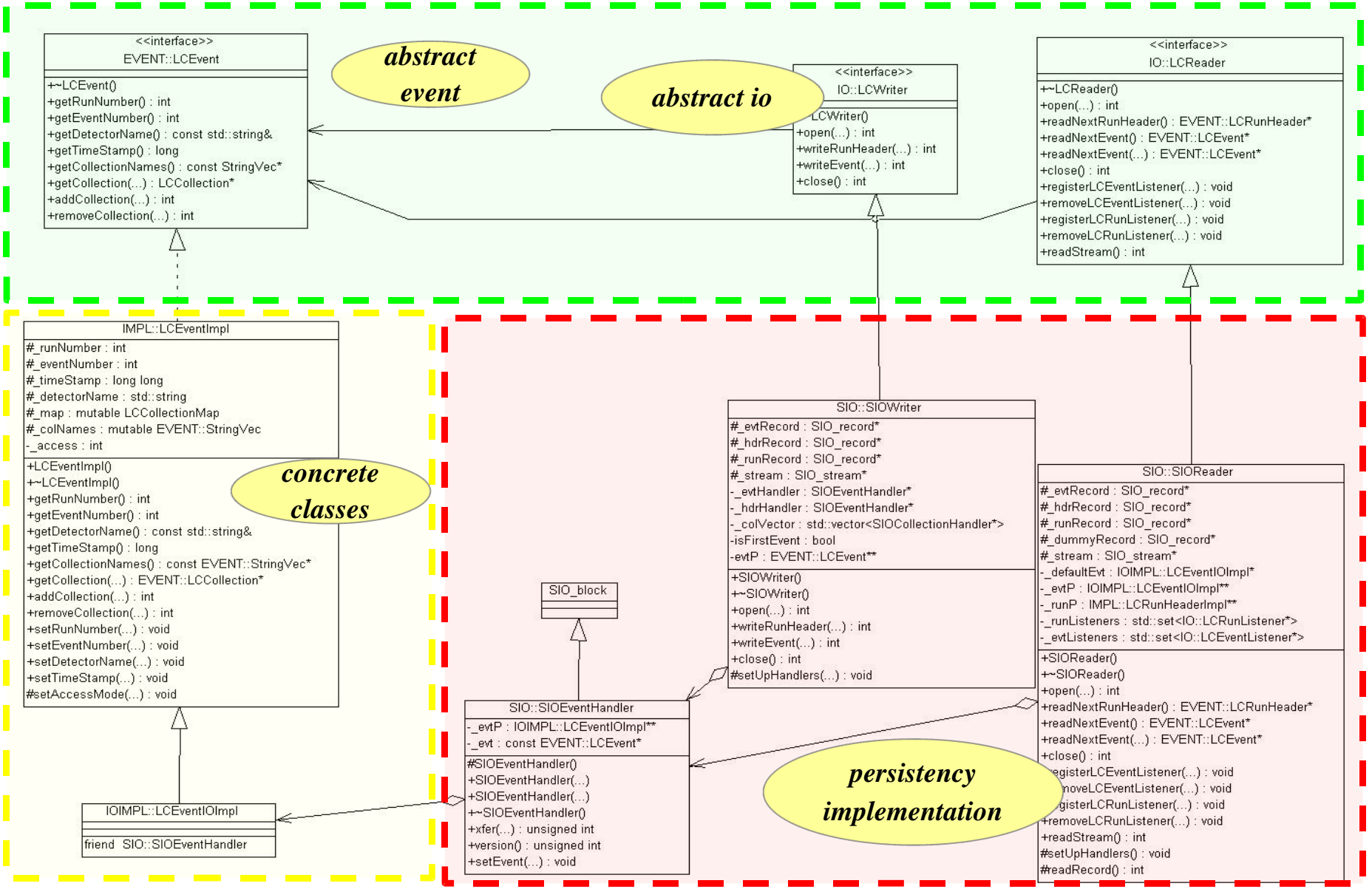
new release: **v01-06**

now de facto standard persistency & datamodel for ILC software

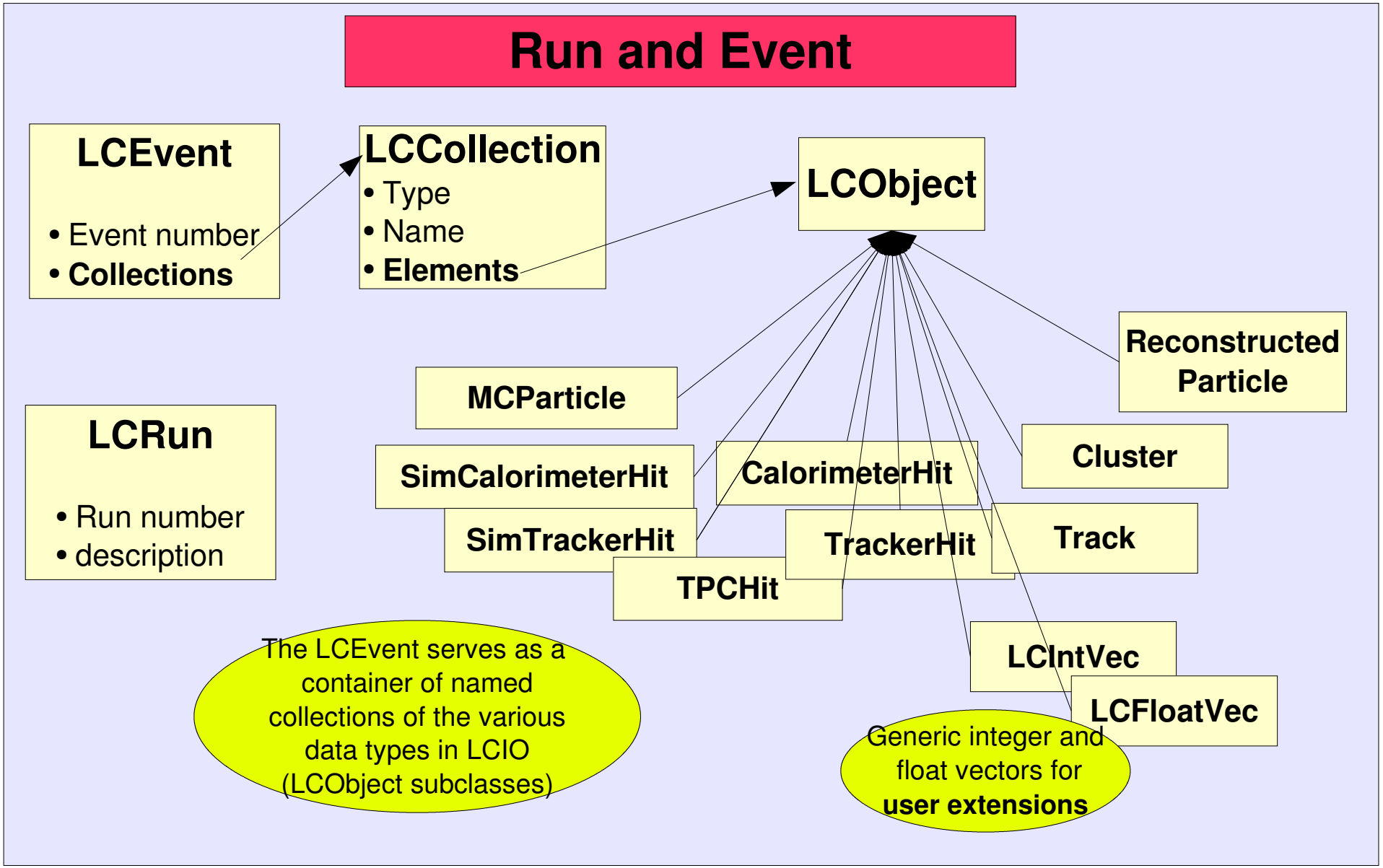
SW-Architecture



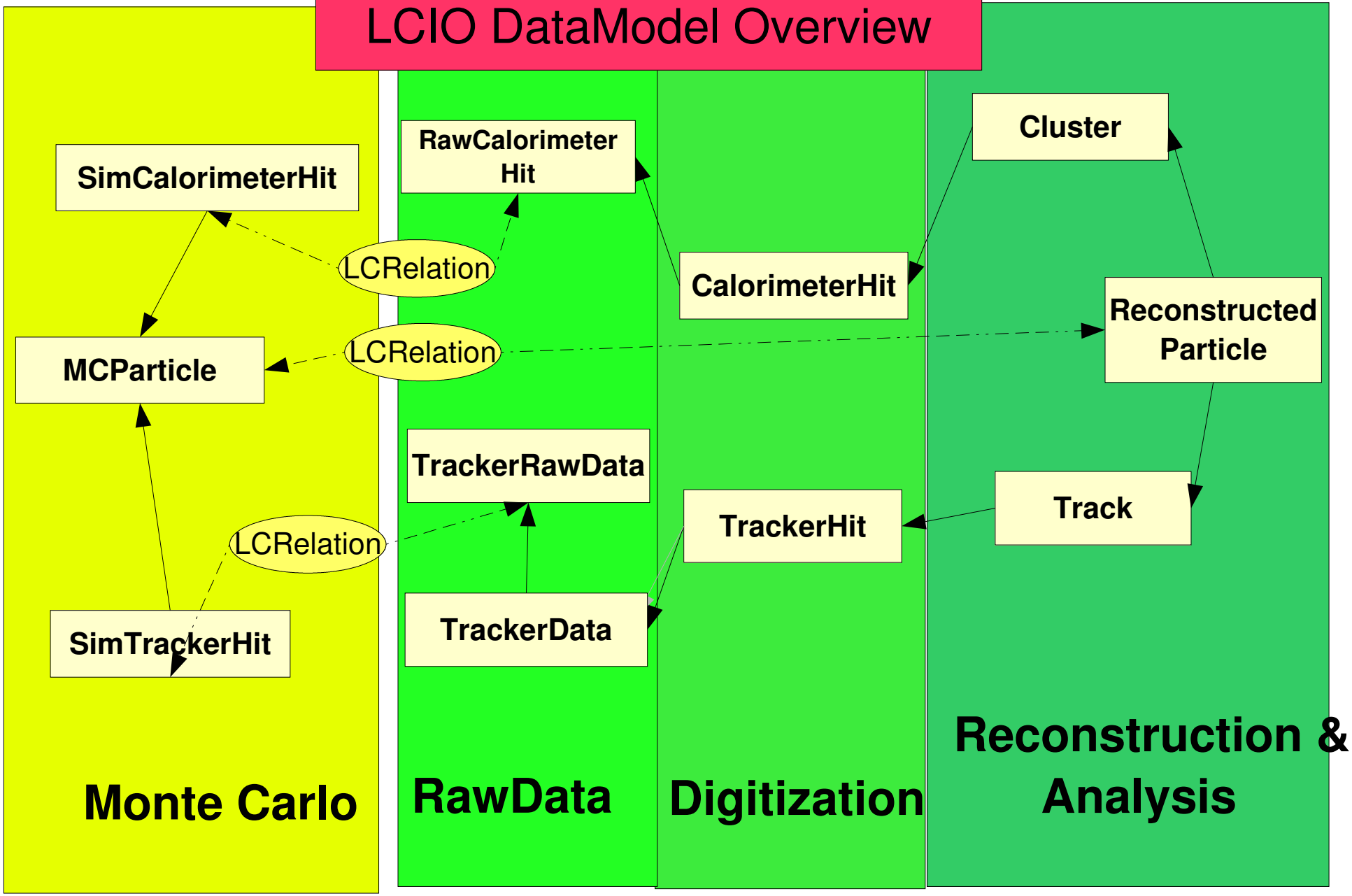
LCIO class design

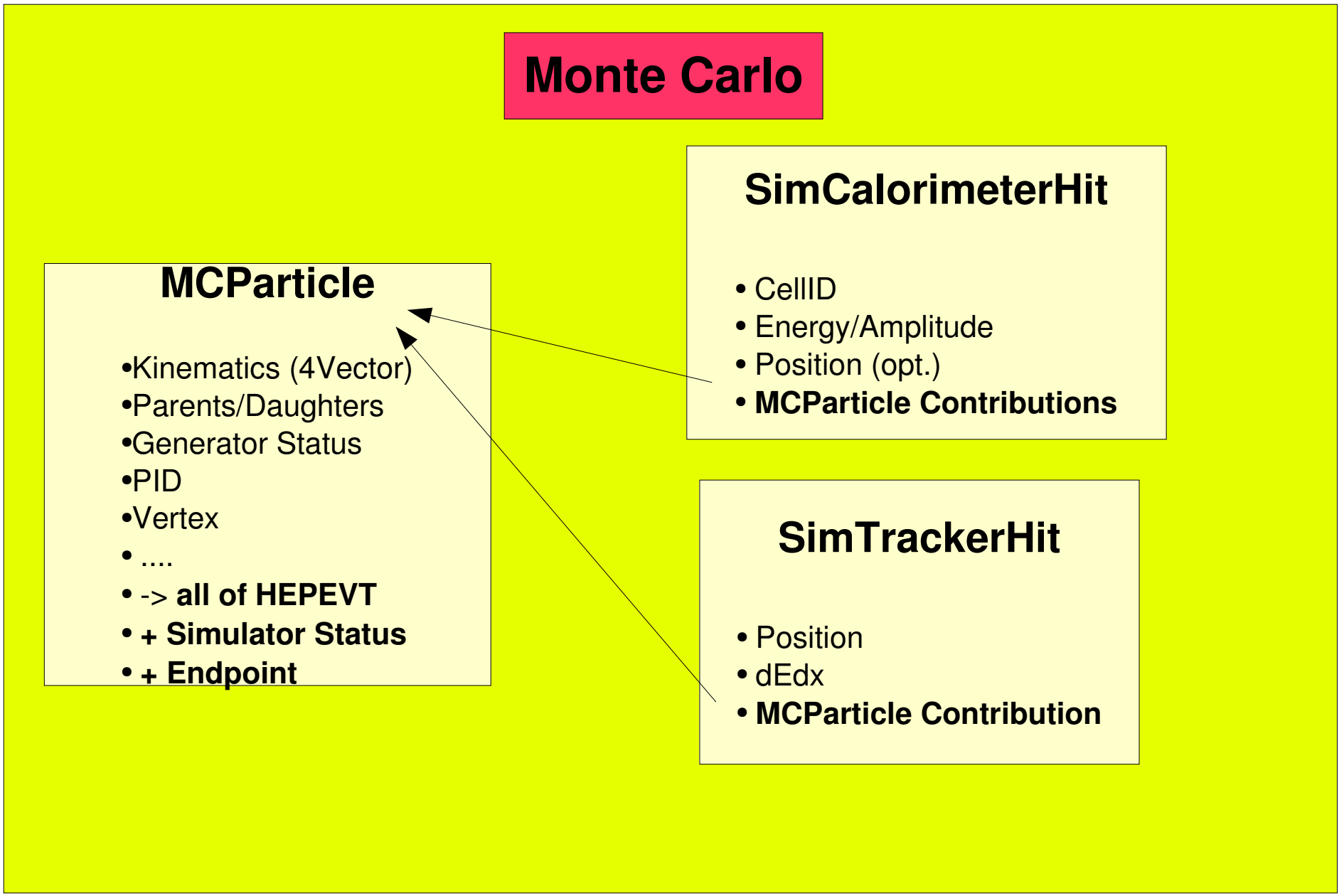


Run and Event

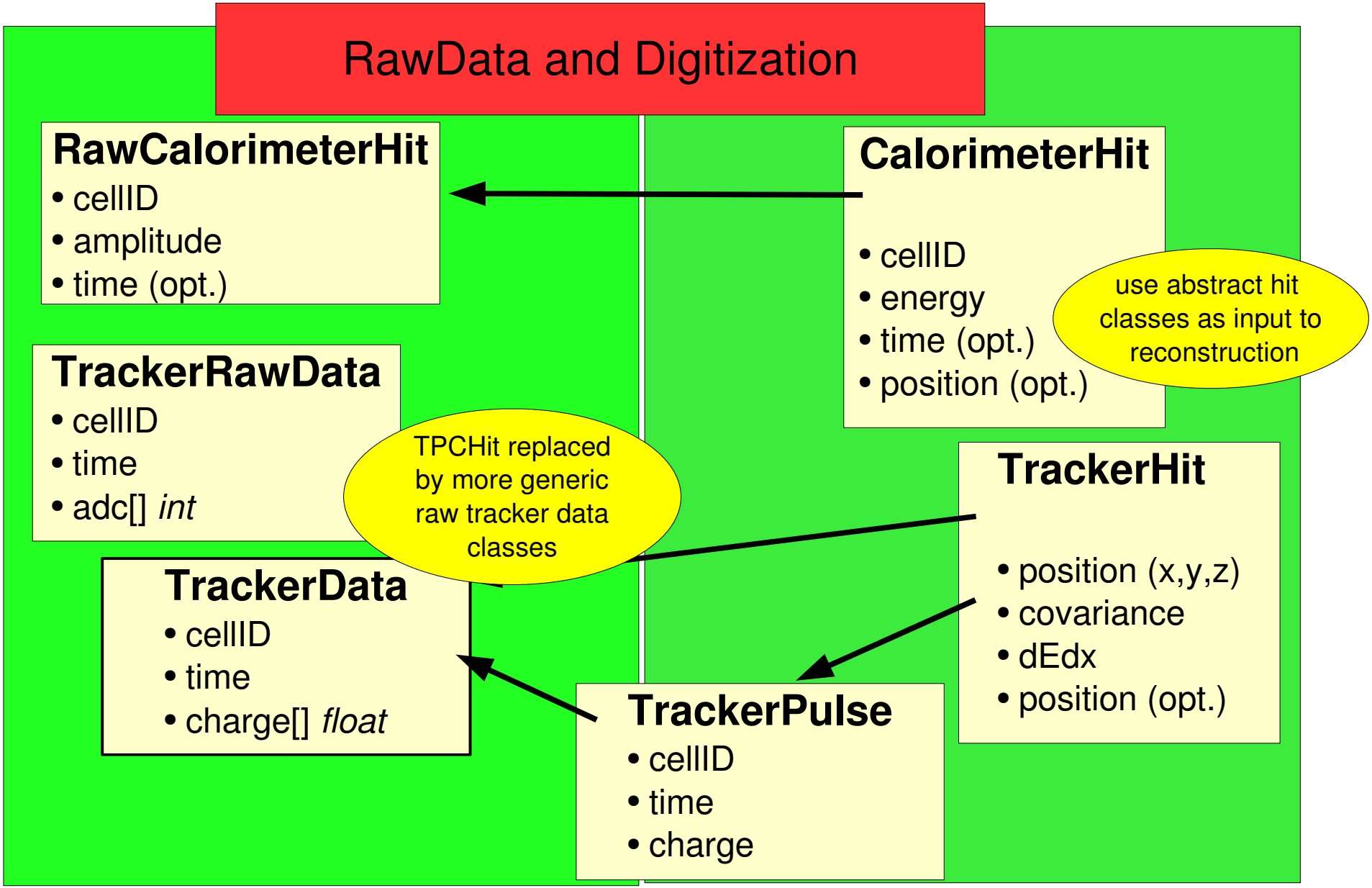


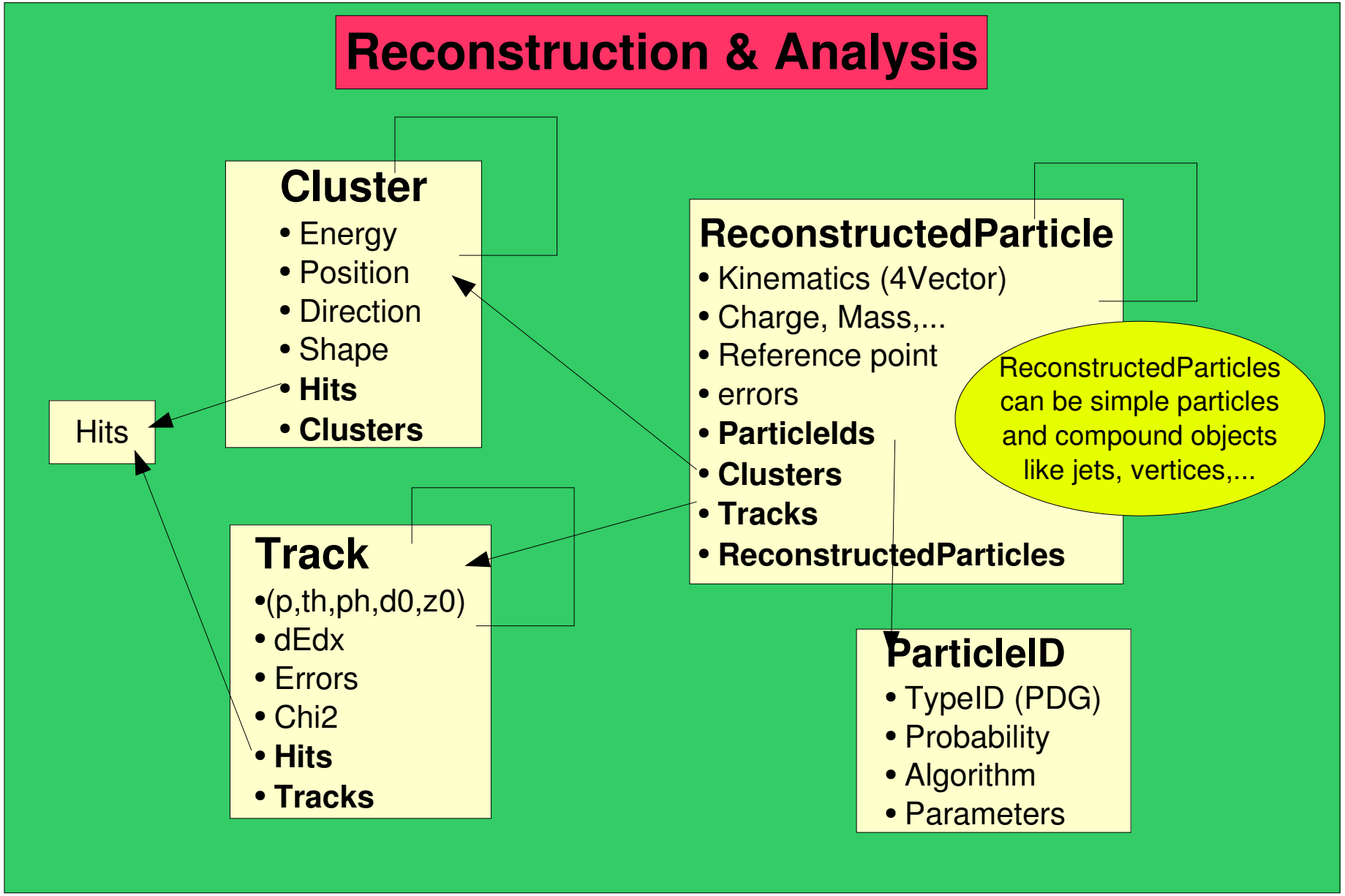
LCIO DataModel Overview




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

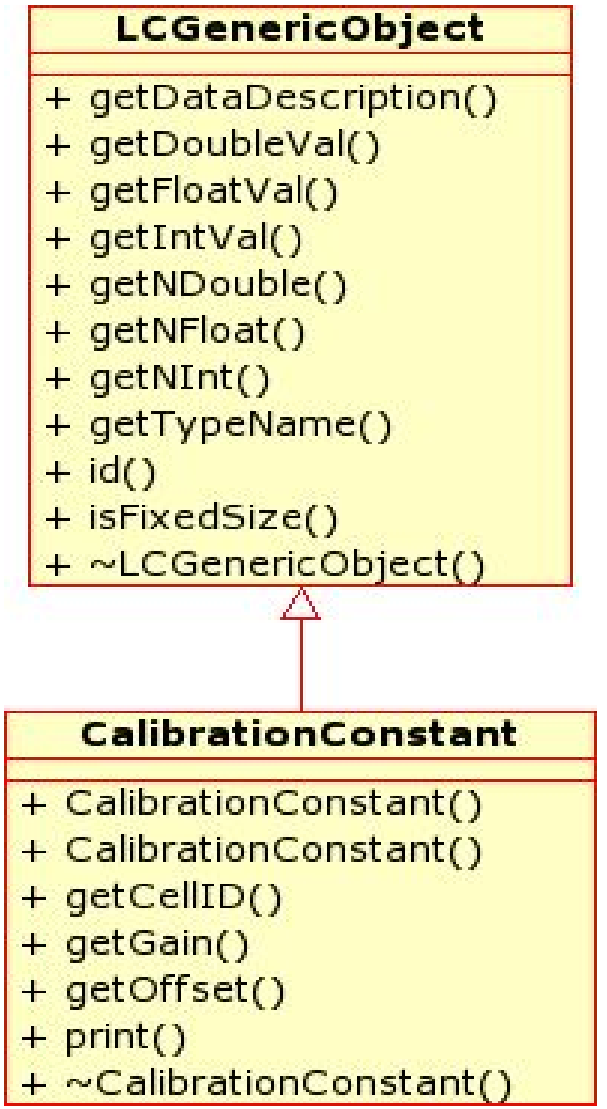
Datamodel IV




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 – 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)



emacs@linux.site

File Edit Options Buffers Tools C Help

```
#include "lcio.h"
#include "UTIL/LCFixedObject.h"

#define NINT 1
#define NFLOAT 2
#define NDOUBLE 0

using namespace lcio ;

class CalibrationConstant ;

/** Example for a simple calibration class based on the LCFixedObject template.
 *  <p>
 *  LCFixedObject uses an instance of LCGenericObjectImpl that holds the data, thus
 *  there is no overhead when the data is read from a database or file
 *  for copying it  to some local structure (Decorator pattern).<br>
 *
 */
class CalibrationConstant : public LCFixedObject<NINT,NFLOAT,NDOUBLE> {

public:

  /** Convenient c'tor.
   */
  CalibrationConstant(int cellID, float offset, float gain) {

    obj()->setIntVal( 0 , cellID  ) ;
    obj()->setFloatVal( 0 ,  offset ) ;
    obj()->setFloatVal( 1 ,  gain ) ;
  }

  /** 'Copy constructor' needed to interpret LCCollection read from file/database.
   */
  CalibrationConstant(LCObject* obj) : LCFixedObject<NINT,NFLOAT,NDOUBLE>(obj) { }

  /** Important for memory handling*/
  virtual ~CalibrationConstant() { /* no op*/  }


  // the class interface:
  int getCellID()   { return obj()->getIntVal(0) ;  }
  float getOffset() { return obj()->getFloatVal( 0 )  ; }
  float getGain()   { return obj()->getFloatVal( 1 )  ; }

  void print(  std::ostream& os ) ;


  // -------- need to implement abstract methods from LCGenericObject

  const std::string getTypeName() const {
    return"CalibrationConstant" ;
  }

  const std::string getDataDescription() const {
    return "i:cellID,f:offset,f:gain" ;
```

--:-- CalibrationConstant.h (C CVS-1.4 Abbrev)--L9--C17-- 4%

example for user class:
$LCIO/src/cpp/src/
EXAMPLE/CalibrationConstant.h

emacs@linux.site

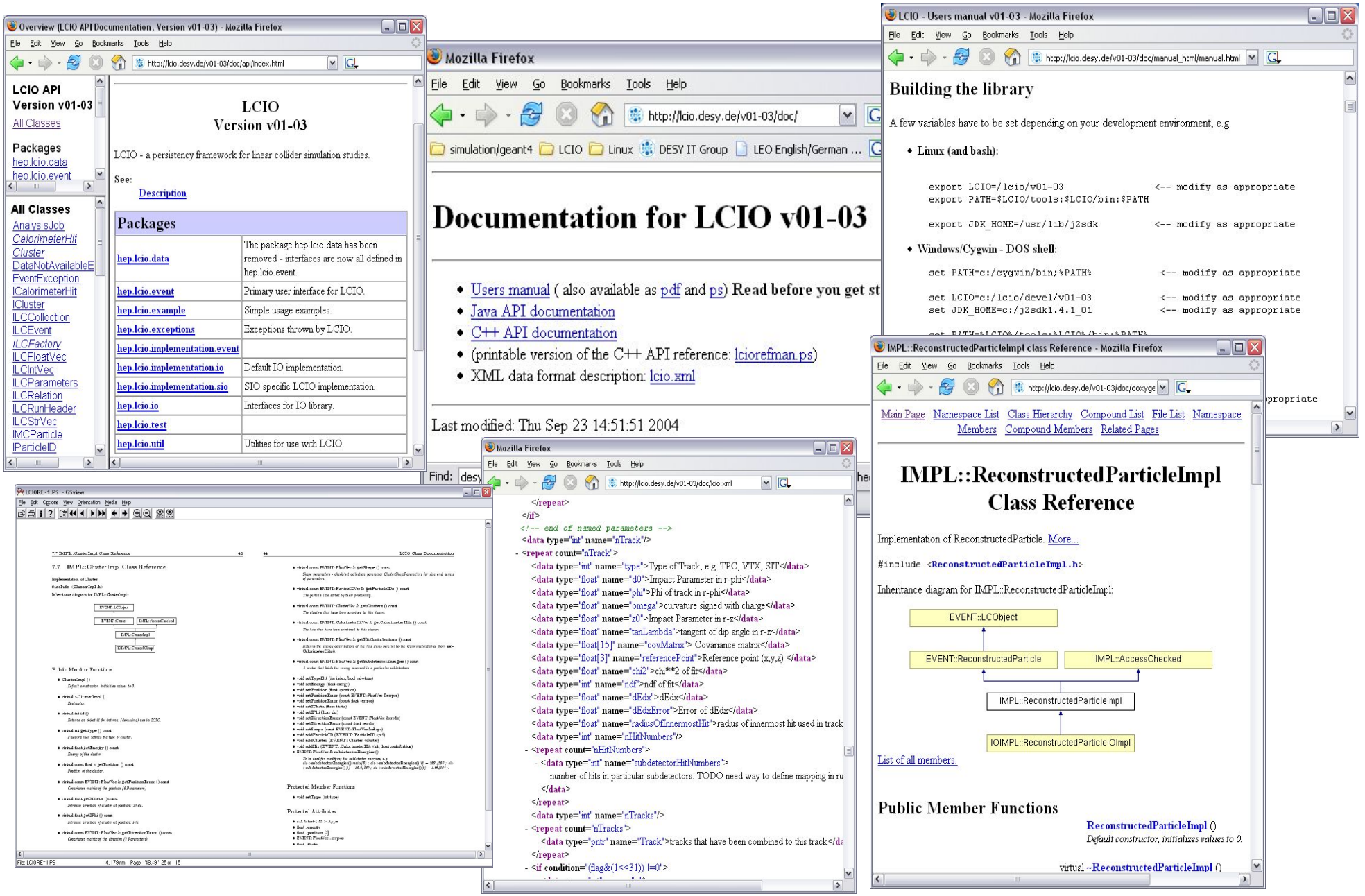
File Edit Options Buffers Tools C++ Help

```
LCReader* lcReader = LCFactory::getInstance()->createLCReader() ;

lcReader->open( FILEN ) ;

LCEvent* evt ;
int nEvents = 0 ;

//----------- the event loop -----------
while( (evt = lcReader->readNextEvent()) != 0 ) {

  // loop over collections  and check if we have objects of user defined
  // type "CalibrationConstant"
  // NB: this is only needed if you don't know the name of the collection
  // that holds the calibration constants

  const StringVec* colNames = evt->getCollectionNames() ;

  for(unsigned int i=0 ; i < colNames->size() ; i++ ){

    LCCollection* col = evt->getCollection( (*colNames)[i] ) ;

    if( col->getParameters().getStringVal("TypeName")  ==  "CalibrationConstant"  ) {


      // now print the calibration constants
      for(int j=0;j<col->getNumberOfElements();j++){

        CalibrationConstant cal( col->getElementAt( j )  ) ;

        std::cout << " calibration for cellid: " << cal.getCellID()
                  << " offset: "                 << cal.getOffset()
                  << " gain: "                   << cal.getGain()
                  << std::endl ;


      }

    }
  }

  nEvents ++ ;
```

--:-- readcalibration.cc   (C++ CVS-1.1 Abbrev)--L33--

example for user class:
$LCIO/src/cpp/src/
EXAMPLE/readcalibration.cc

LCIO Online documentation



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 discussion I

- LCIO meeting yesterday:
  - TrackerHit: now 3d point xyz
  - proposal
    - have various types: cylindrical, xy-plane, zplane, 1d, 2d ,...
    - depending on type of tracker used
    - still use existing class TrackerHit
  - option:
    - define new classes for these specific tracker types

Note: this breaks our previous
LCIO philosophy with one (simple) class TrackerHit !

keep it simple - as simple as possible but no simpler...

LCIO discussion II

- how to store multiple Track fits in LCIO
  - option1 (current version):
    - store one collection for every type of fit, e.g.:
      - “Tracks” - default, fitted at IP (holds pointers to hits)
      - “TracksAtInnermostHit” (points to “Tracks” tracks)
      - “TracksAtOutermostHit” (points to “Tracks” tracks)
      - “TracksAtCaloFace” (points to “Tracks” tracks)
    - pro: already implemented / easy to use collection with particular fit (calo)
    - con: maybe not so intuitive ?
  - option 2
    - modify Track class so that it holds mutiple fits
    - pro: more intuitive (?)
    - con: needs coding / not so easy to access a particular fit

LCIO discussion III

- what is your experience with LCIO ?
  - files sizes ?
  - speed reading files ?
  - direct access ?
- what feature is missing in LCIO that your need to do your work ?
- anything else you would like to discuss ?

Backup slides ...

LCIO status [v01-04]

- new features:
  - support and definition of 64bit time stamps
    - **ns since 1/1/1970 (UTC)**
  - multiple I/O streams in C++
  - LCGenericObjects in Java (user defined data objects)
  - subset collections
    - hold pointers/references to objects already existent in the event, e.g. LeptonCandidates from ReconstructedParticles
    - transient and persistent
      - if persistent, only pointers/references are stored in the file
      - files are downward compatible, LCIO 1.3 can read new files
- bug fixes
- improved documentation

LCIO status [v01-05]

- changed return values of E, P, m to double for MCParticle and ReconstructedParticle
  - stored are still floats
  - requires some trivial changes (float->double) in code where indicated by the compiler !
- new template UTIL::LCTypedVector<T> for creating std::vector<LCObjectType> from LCCollections
  - allows to use iterators and STL algorithms, e.g. std::for_each()
- added first implementations of generic tracker raw data classes (TPC, VTX, SiliconStrip,...)
  - TrackerRawData, TrackerData, TrackerPulse
- files are downward compatible with LCIO 1.4
- bug fixes
- see $LCIO/doc/versions.readme for more

LCIO status [v01-07(06)]

- 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