Towards LCIOv2

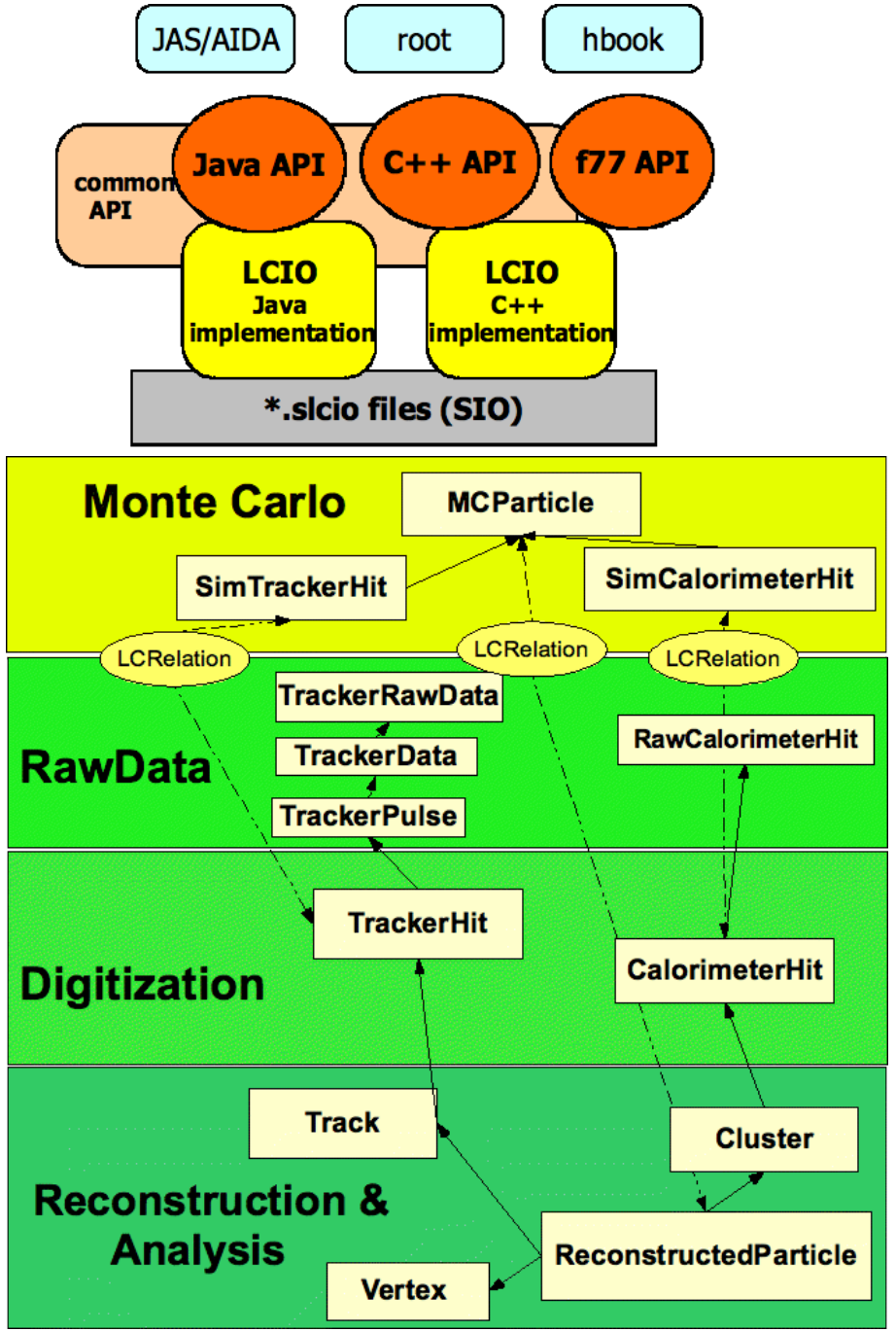
Improving the EDM and the I/O

Frank Gaede, DESY

Linear Collider Software Workshop

DESY, July 4th, 2010

LCIO: persistency & event data model

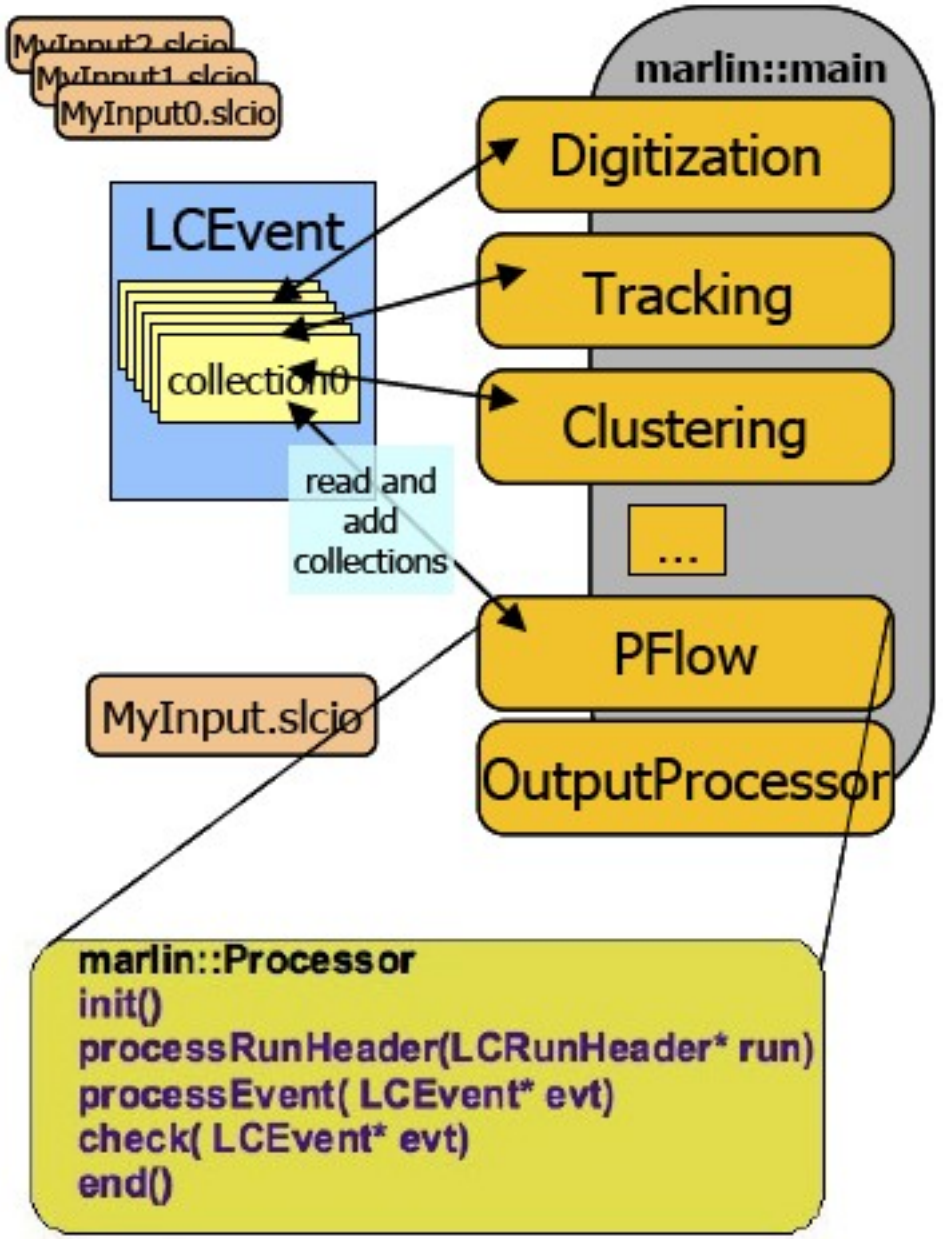


- joined DESY and SLAC project – first presented @ CHEP 2003
- provides persistency (I/O) and an event data model (EDM) to ILC detector R&D community
- features:
  - Object I/O (w/ pointer chasing)
  - schema evolution
  - compressed records
  - hierarchical data model
  - **decoupled from I/O by interfaces**
    - C++, Java (and Fortran)
  - some generic user object I/O

LCIO is used by ILD, SID, Calice, EUPixelTelescope, LCTPC,...

LCIO in Marlin

- LCIO provides the event data model in Marlin:
- transient==persistent event data
- software bus model
  - Marlin processors, eg PandoraPFAProcessor, LCFIVertexProcessor programmed against LCIO



- effectively all existing LC software tools are programmed against LCIO
- need to evolve LCIO in a backward compatible way

LCIOv2 - envisaged features

- LCIO provides a rather complete event data model and has been used successfully in SID and ILD LOI mass production and in various R&D testbeam programs
- LCIOv2 needs to be backward compatible and should provide some new features
  - **direct access to events** (done -> see this talk)
  - **partial reading of events**
  - **splitting of events over files**
  - **storing of (arbitrary) user classes**
  - **simplify using LCIO with ROOT**
    - (ROOT macros, TTreeViewer, I/O (?) ,...)
  - **improving the event data model**
    - (1d,2d hits, tracks/trajectories)

partial reading & splitting of events

- needed for performance and cost (disk space) issues:
  - read only objects of interest in analysis (PandoraPFOs)
  - store simulation and reconstruction output in separate files
- main obstacle: need pointer/reference mechanism across I/O records and files
  - not available in SIO now and can't use TRefs in ROOT
- need index based pointers independent of I/O, e.g.:
  - long64 index = HASH(collName) << 32 | collIndex
- experimental C++ version exists in ROOT I/O branch for partial reading of events (not yet file splitting)
  - need further testing & implementation in SIO (also Java)
  - need extension of LCIO::Reader interface

storing of arbitrary user classes

- LCIO event data model rather complete - but also clear need for storing user defined information
  - LCGenericObjects can store almost arbitrary data structures based on ints, floats and doubles
    - files can be read w/o any additional code (dictionary)
    - small performance penalty
    - extensively used in LCCD (conditions data) by testbeams
- occasional user request for 'natively' storing arbitrary user classes in LCIO
  - possible in principle with LCIO/SIO (not documented and somewhat 'discouraged') - would come 'for free' w/ ROOT I/O
- IMHO: success of LCIO is to a large extend due to the slightly restrictive definition of the event data model i.e. the interfaces between modules/processors

ROOT I/O for LCIO

- user request to have closer link of LCIO to ROOT
  - use LCIO classes in ROOT macros (former GLD groups)
  - have fast interactive analysis with ROOT tree
- investigate the optional use of ROOT I/O for LCIO
  - would provide 'missing features': direct access, partial reading and splitting of events (and streaming of user classes)
- created experimental branch in cvs (rio_v00-00)
  - create ROOT dictionary w/ help from ROOT team (A.Naumann)
  - implemented index based pointers for C++
  - needed some changes to LCIO classes: LCTCollection<T>, std::vector as members, ,...
  - can create almost complete copies of LCIO DST in ROOT
    - no subcollections (pointers only) yet
  - streaming mode for Marlin under development
- see: talks at ILD software working group meetings for details
- still some issues to resolve (interface to Java !!)

a ROOT dictionary for LCIO

- LCIO now comes with a ROOT dictionary for all LCIO classes (optional) - with this one can:
  - use LCIO classes in ROOT macros
  - write simple ROOT trees, e.g. std::vector<MCParticleImpl*>
  - use TTreeDraw for quick interactive analysis of LCObjects:

```
//---gamma conversions:
TCut isPhoton("MCParticlesSkimmed.getPDG()==22" ) ;
LCIO->Draw("MCParticlesSkimmed._endpoint[][0]:
            MCParticlesSkimmed._endpoint[][1]",isPhoton ) ;
```

  - write complete LCIO events in one ROOT branch
  - see: $LCIO/examples/cpp/rootDict/README for details & help

- -> we are interested in feedback from the users if this provides already the requested features

Improving the LCIO event data model

- suggested improvements to the event data model:
- 1D, 2D tracker hits
  - LCIO (Sim)TrackerHit is a 3D space point - whereas actual measurements are either 1D (strip) or 2D (TPC) where the detector surface (line) provides the additional geometry information
- Track
  - currently Track has pointers to all TrackerHits and one set of (Helix) parameters
  - generally one wants to have multiple fits for one set of hits, e.g. at the IP or at the face of the calorimeter
  - Trajectory could be introduced as high level convenient view to these fits
  - currently not straight forward (though possible) to store kinks in LCIO

- details are coupled to development in tracking code
- hope to make progress at this meeting (LCIO meeting friday)
- also user feedback welcome

LCIO release v01-51

- improved EDM
  - renamed dEdx in (Sim)TrackerHit to EDep
  - added EDepError to Trackerhit
  - added error Matrix for charge and time meassurement to TrackerPulse
- new features
  - implemented real direct access to events (no fast skip)
  - old files simply made 'direct accessible' on open()/close()
  - new ostream operators<<(...) in C++
    - cout << ((MCParticle*) c->getElementAt(i)) << endl ;
  - Java builds with Maven
  - improved CMake builds for developers
- bug fixes...

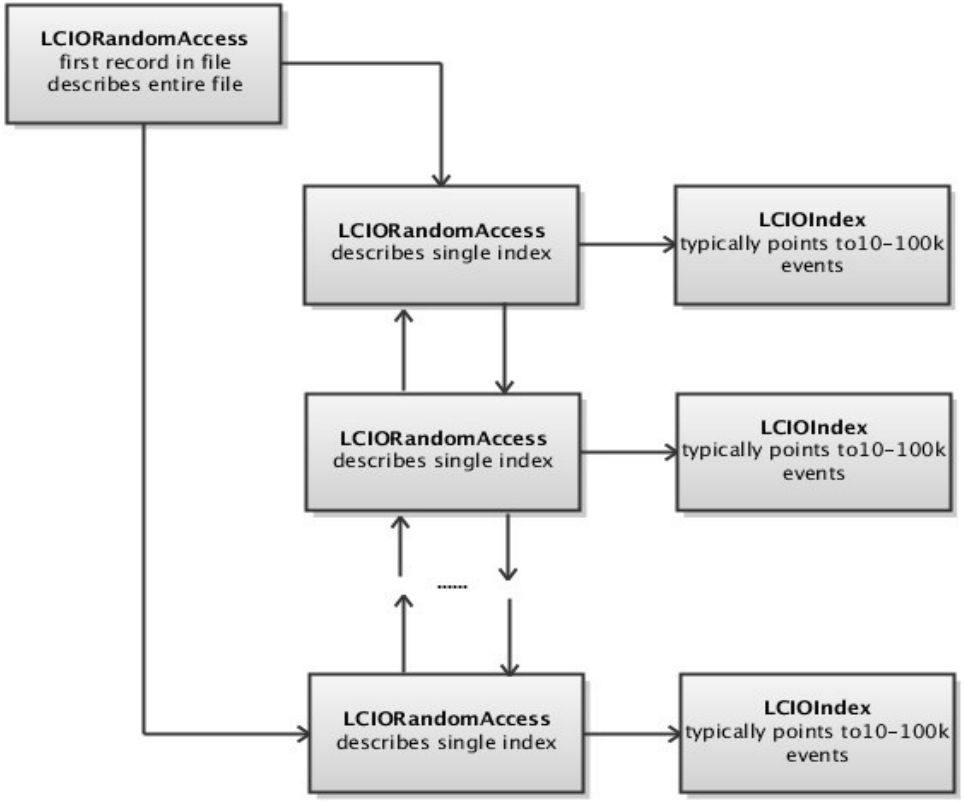
Summary & Outlook

- currently the LCIO team is working on 'LCIOv2' to further improve LCIO and address the following feature requests:
  - direct access (done in v01-51)
  - partial reading and splitting over files (partly done, experimental)
  - linking LCIO closer to ROOT for analysis (partly done w/ ROOT dict)
  - improving the event data model (ongoing - hope to make progress after ILD WS)
- other improvements not shown today are also discussed:
  - make interfaces more consistent and more convenient to use
  - better define the suggested use of meta-data
  - provide LCIO file browser (JAS)
  - ...
- make use of the 'spare time' in 2010 until preparation for the DBD reports in 2012 will have to start
- continuos user feedback is important and welcome !

additional material

direct access to LCIO events

- direct access to LCIO events needed:
  - overlay of random background events
  - physics analysis – reading of pre-selection
  - so far available through fast skip or creation of TOC on opening (slow)
  - → introduced two additional records **LCIORandomAccess/LCIOIndex**
  - records written at end of file on close()
  - can append to file
  - **can add direct access to existing file**
    - if opened in append on writable file system (not tape)
- released in v01-51



improved Tracker Hit classes

released in v01-51

- TrackerPulse
  - added covariance (error) matrix for charge and time measurements

```
/** Covariance matrix of the charge (c) and time (t) measurements.
 * Stored as lower triangle matrix, i.e.
 *  cov(c,c) , cov(t,c) , cov(t,t).
 */
virtual const FloatVec & getCovMatrix() const = 0;
```

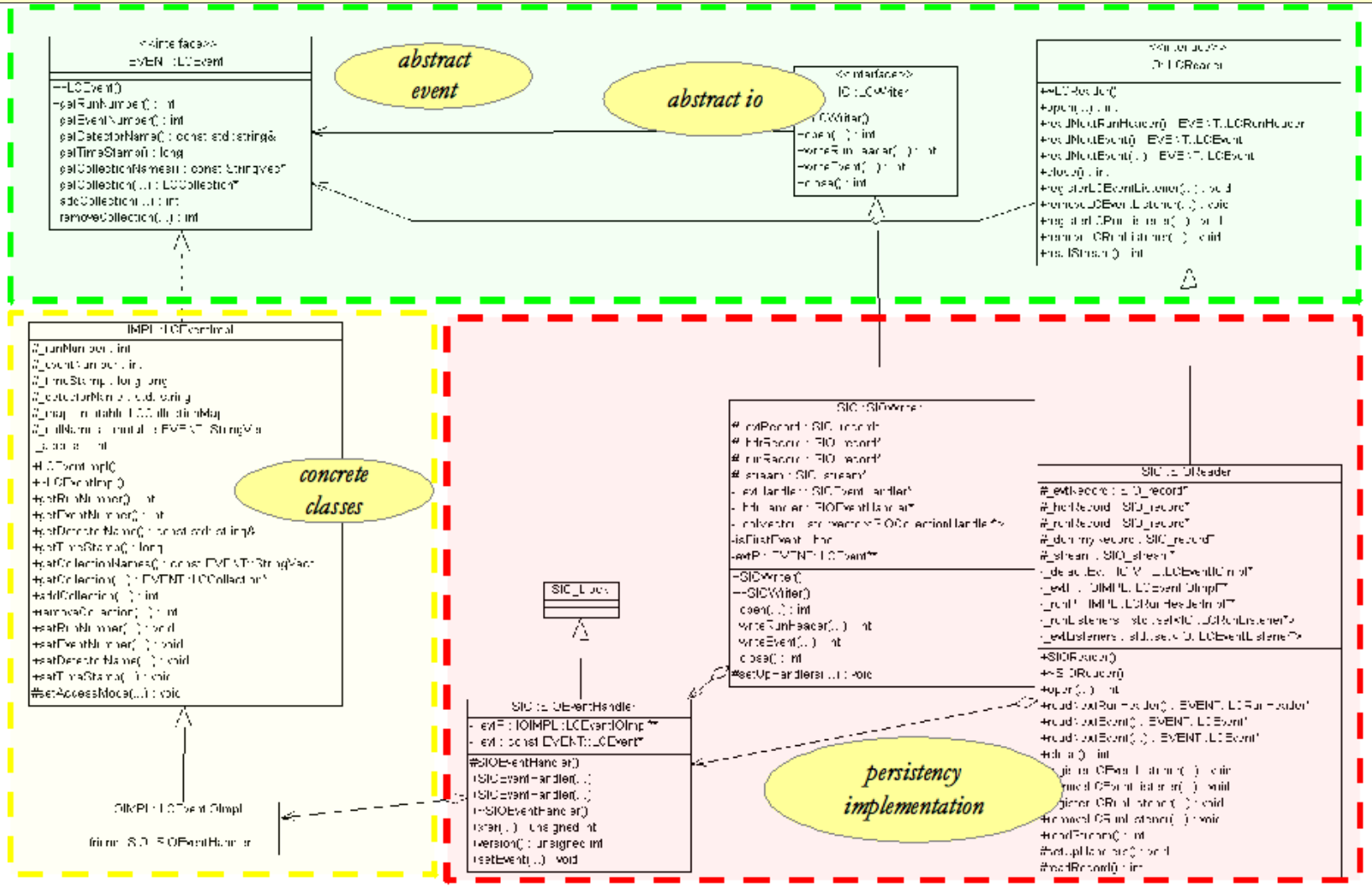
- (Sim)TrackerHit
  - renamed dEdx to EDep - deposited energy
    - dEdx methods are deprecated: they still can be used but result in a printed warning ...
  - added EDep to TrackerHit
    - measurement error

```
/** The dE/dx of the hit in [GeV].
 *  DEPRECATED. renamed to getEDep()
 */
virtual float getdEdx() const = 0;
```

```
/** The energy deposited on the hit [GeV] */
virtual float getEDep() const = 0;

/** The error measured on EDep [GeV] */
virtual float getEDepError() const = 0;
```

LCIO software design

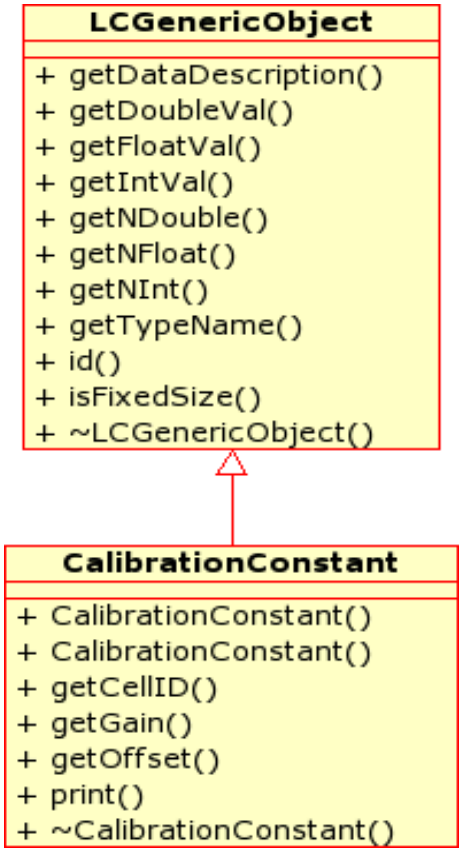


- event data model is strictly decoupled from persistency package – currently SIO, but can be changed
- user code only sees pure abstract interface (Reading) or LCIO implementation classes (Writing)

LCIO – user extensions

- LCIO defines the event data model and provides the persistency for it
- however users want to extend existing classes and persist their own classes
- LCGenericObject provided by LCIO:
  - users can store 'arbitrary' data structures in LCGenericObject w/o writing streamer code
  - performance not great
- LCIO runtime extensions (C++)
- extension of the object with arbitrary (even non-LCObject) classes
- bidirectional relations between LCObjects
  - one to one
  - one to many
  - many to many

no persistency yet



SIO persistency

- simple C++ persistency tool developed at SLAC
- provides some OO-features like pointer chasing
- user needs to write streamer code (done in LCIO)

- missing so far:
  - splitting of events over files
  - direct access
  - user streamer code
- could be implemented rather easily, if needed