# LCIO

## for FCC

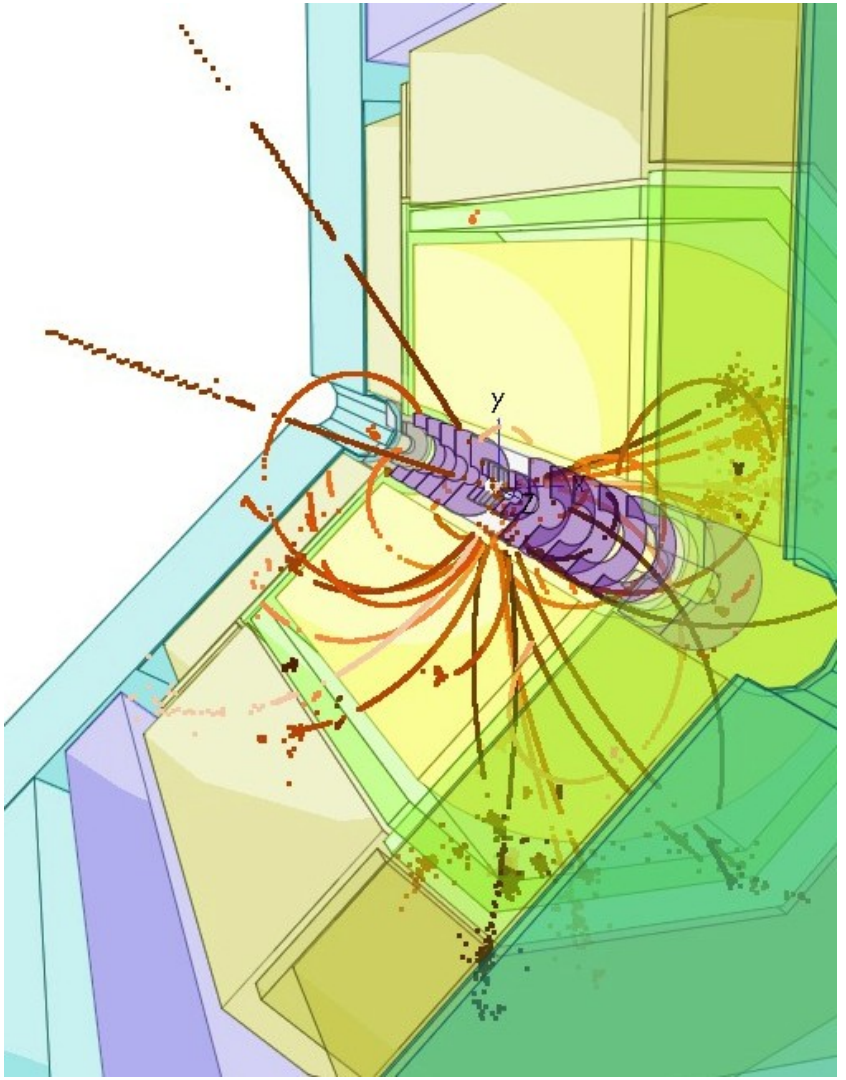
Frank Gaede, DESY

FCC Software Meeting

July 10, 2014

# Outline

- LCIO Overview
- Software Design
- Event Data Model
- Generic User Data
- LCIO and ROOT
- LCIO Applications
- Summary & Outlook



# LCIO overview

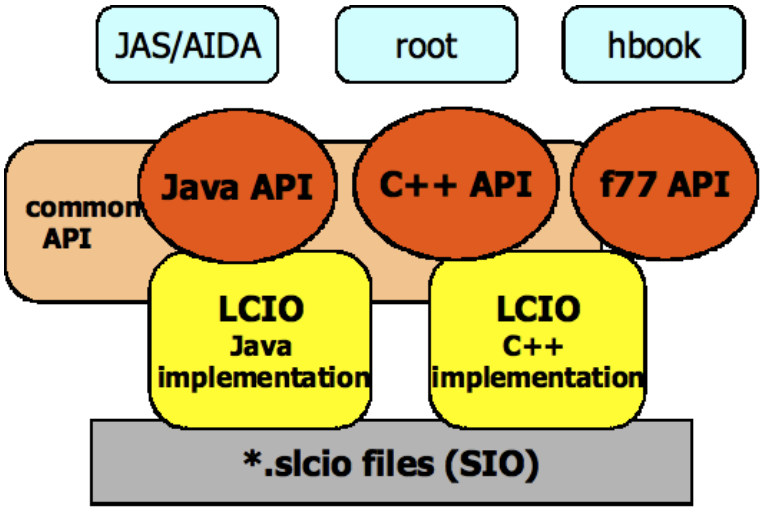
- LCIO is a software package that provides an event data model and a persistency format for Linear Collider physics studies
- development started in 2002 as a DESY/SLAC project
- main goals then - and still today:

  - provide a common language (EDM) for the LC community
  - enable sharing and common development of software tools and frameworks
  - foster collaboration and avoid duplication of effort

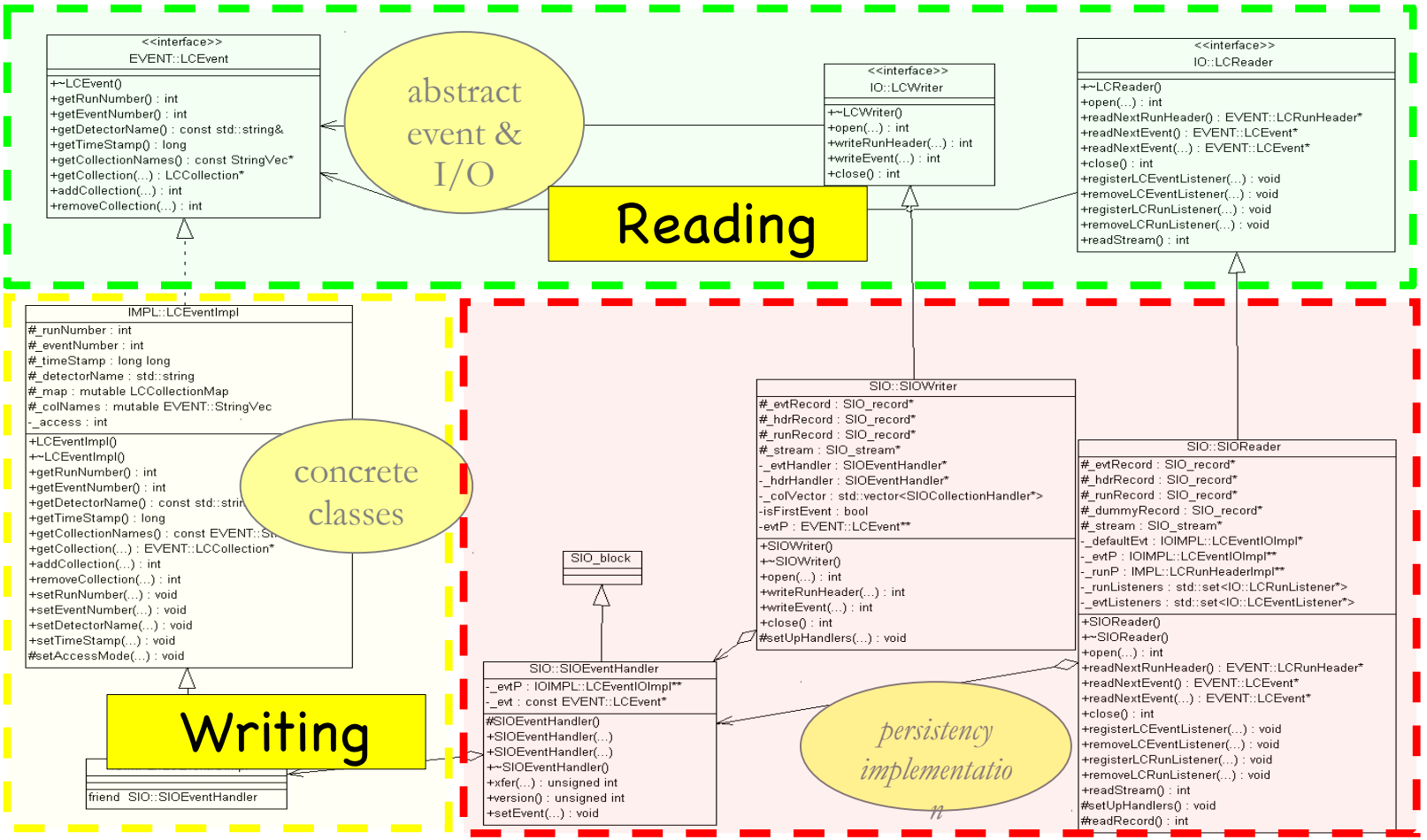
- has become the standard for all LC physics studies since
- used for Monte Carlo simulation and test beams by CLIC, ILD, SiD, Calice, LCTPC, EUTelescope (also Atlas)...

# LCIO Software design I

- LCIO provides C++, Java and Fortran (C, cfortran.h) API
  - Python bindings (ROOT dictionary)
- two independent implementations: C++ and Java
  - C++ (or Java) only builds possible
  - Java currently not used
- common file format SIO:
  - simple binary I/O based on records
  - using zlib for compression
  - pointers inside one record implemented through integerIDs and lookup tables
  - ships with LCIO



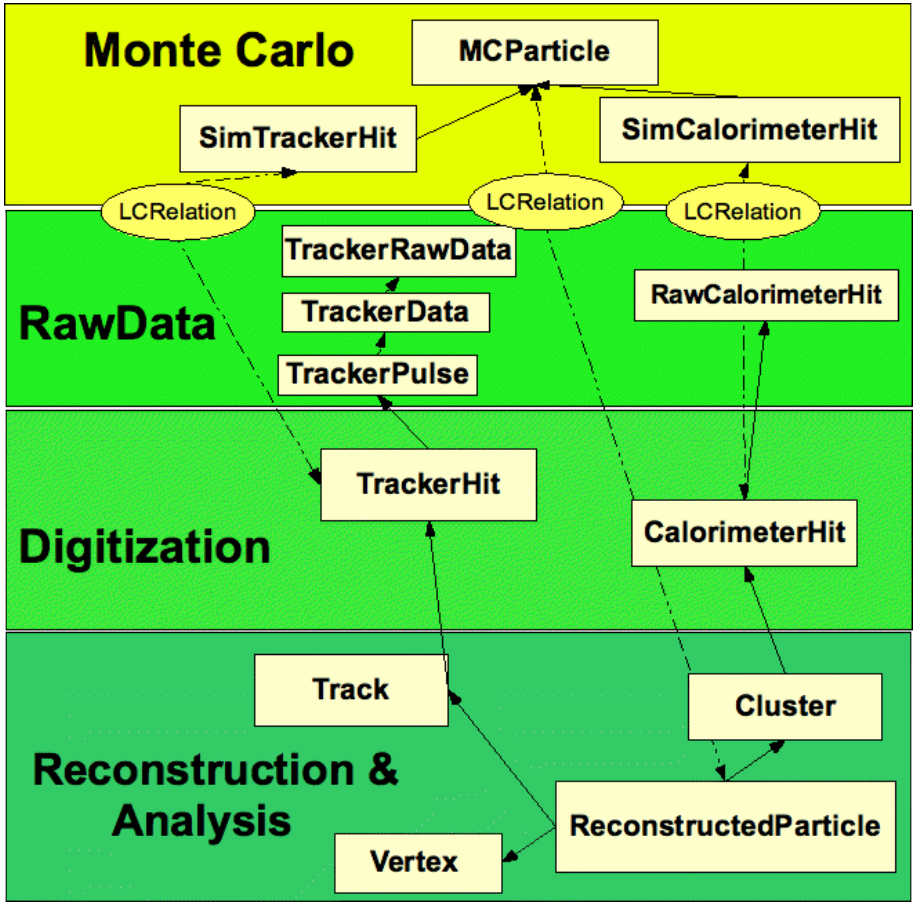
# LCIO software design II




- 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)

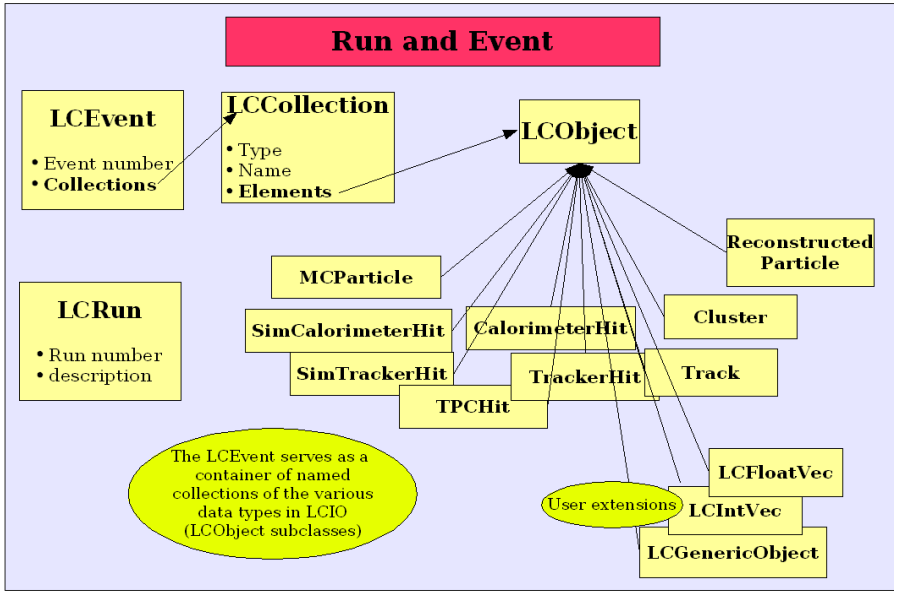
# The Event Data Model

- LCIO defines a hierarchical event data model, with higher level objects pointing back to their constituents
  - only indirect (LCRelations) link to MC-Truth information
  - additional relations possible:
    - direct RecoParticle-MCParticle
    - direct Track-MCParticle
- event data classes originally targeted at the Linear Collider but should be generic enough for any collider experiment



# LCIO event store

- LCIO data is organized in events
  - one record per event
- the data are stored in named collections in the LCEvent
- multiple collections of same type possible
- subset collections: pointers only
- underlying events (pile-up) possible by merging several events into one
  - also used for LC studies (gg->hadron background)



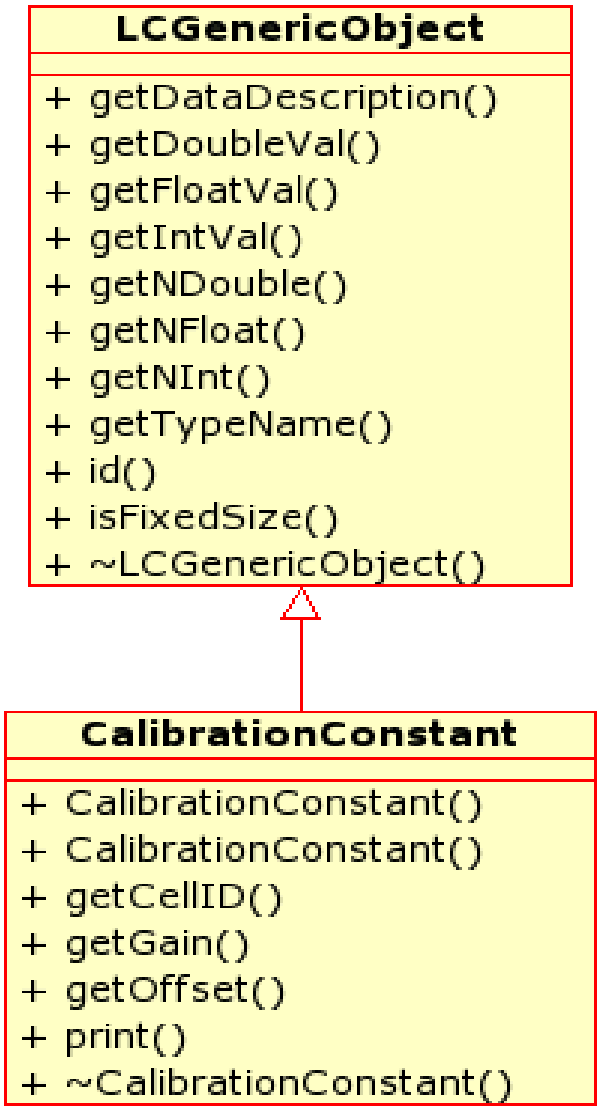
# LCIO EDM example: MCParticle

| | |
|---|---|
| virtual double | **getEnergy** () const =0<br>*Returns the energy of the particle (at the vertex) in [GeV] computed from the particle's momentum and mass - only float used in files.* |
| virtual const float * | **getSpin** () const =0<br>*Returns the spin (helicity) vector of the particle.* |
| virtual const int * | **getColorFlow** () const =0<br>*Returns the color flow as defined by the generator.* |
| virtual const **MCParticleVec** & | **getParents** () const =0<br>*Returns the parents of this particle.* |
| virtual const **MCParticleVec** & | **getDaughters** () const =0<br>*Returns the daughters of this particle.* |
| virtual int | **getNumberOfParents** () const =0<br>*Returns the number of parents of this particle - 0 if mother.* |
| virtual **MCParticle** * | **getParent** (int i) const =0<br>*Returns the i-th parent of this particle.* |
| virtual int | **getPDG** () const =0<br>*Returns the PDG code of the particle.* |
| virtual int | **getGeneratorStatus** () const =0<br>*Returns the status for particles as defined by the generator, typically*<br>*0 empty line*<br>*1 undecayed particle, stable in the generator*<br>*2 particle decayed in the generator*<br>*3 documentation line.* |
| virtual int | **getSimulatorStatus** () const =0<br>*Returns the status for particles from the simulation, e.g.* |
| virtual bool | **isCreatedInSimulation** () const =0<br>*True if the particle has been created by the simulation program (rather than the generator).* |
| virtual bool | **isBackscatter** () const =0<br>*True if the particle was created by the simulator as a result of an interaction or decay in non-tracking region, e.g.* |
| virtual bool | **vertexIsNotEndpointOfParent** () const =0<br>*True if the particle was created as a result of a continuous process where the parent particle continues, i.e.* |
| virtual bool | **isDecayedInTracker** () const =0<br>*True if the particle decayed or interacted in a tracking region.* |
| virtual bool | **isDecayedInCalorimeter** () const =0<br>*True if the particle decayed or interacted (non-continuous interaction, particle terminated) in non-tracking region.* |
| virtual bool | **hasLeftDetector** () const =0<br>*True if the particle left the world volume undecayed.* |
| virtual bool | **isStopped** () const =0<br>*True if the particle lost all kinetic energy inside the world volume and did not decay.* |
| virtual const double * | **getVertex** () const =0<br>*Returns the production vertex of the particle in [mm].* |
| virtual float | **getTime** () const =0<br>*The creation time of the particle in [ns] wrt.* |
| virtual const double * | **getEndpoint** () const =0<br>*Returns the endpoint of the particle in [mm] if the endpoint has been set explicetly.* |
| virtual const double * | **getMomentum** () const =0<br>*Returns the particle's 3-momentum at the production vertex in [GeV]*<br>• *only float used in files.* |
| virtual double | **getMass** () const =0<br>*Returns the mass of the particle in [GeV] - only float used in files.* |
| virtual float | **getCharge** () const =0<br>*Returns the particle's charge.* |
| virtual int | **getNumberOfDaughters** () const =0<br>*Returns the number of daughters of this particle.* |
| virtual **MCParticle** * | **getDaughter** (int i) const =0<br>*Returns the i-th daughter of this particle.* |

More at: **http://lcio.desy.de/v02-04-03/doc/doxygen_api/html/index.html**

# LCIO – Generic User Information

- LCIO data model defines everything needed for LC physics 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)
- used extensively for conditions data in LCCD (Calice, LCTPC,...)



# LCIO runtime extensions (C++)

- LCIO provides runtime extensions to objects allowing to attach arbitrry 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

# Building LCIO

- LCIO has no external dependencies
  - optionally depend on ROOT to create a dictionary
  - uses CMake as build tool

- Download and build LCIO (C++)

```
svn co svn://svn.freehep.org/lcio/trunk lcio
cd lcio ; mkdir build ; cd build ;
cmake -D BUILD_ROOTDICT=On ..
make install
```

- run example programs, e.g.:

```
export PATH=$LCIO/bin:$PATH
simjob ; anajob simjob.slcio ; dumpevent simjob.slcio 1
```

# LCIO and ROOT

- the ROOT dictionary for LCIO provides:
  - direct usage of LCIO classes in ROOT macros, e.g.
    - open LCIO files in ROOT and fill histograms
    - possibility to write LCIO events or parts thereof to ROOT
    - see: $LCIO/examples/cpp/rootDict
  - Python bindings for LCIO:
    - create ROOT hists from LCIO
    - more eloborate examples:
      - $LCIO/example/python
  - possibility to provide a ROOT I/O layer for LCIO

```
'''
 export PYTHONPATH=$ROOTSYS/lib:$PYTHONPATH
 export PYTHONPATH=${LCIO}/src/python:${LCIO}/examples/python:${PYTHONPATH}
'''
from ROOT import TH1D
from pyLCIO import IOIMPL
import sys

hen   = TH1D( 'hen',  'MC particle energy', 100, 0., 10. )

reader = IOIMPL.LCFactory.getInstance().createLCReader()

reader.open( sys.argv[1] )

for evt in reader:

    mcol = evt.getCollection("MCParticle")
    for mcp in mcol:

        if( mcp.getGeneratorStatus() == 1):
            hen.Fill( mcp.getEnergy() )


hen.Draw()

userInput = raw_input( 'Press any key to continue' )
```

# LCIO Applications

- Marlin application framework used by ILD, CLIC, (SiD) uses LCIO as transient event data model
- => using LCIO provides access to full suite of reconstruction code used in the LC community
  - full C++ track reconstruction with a TPC as central tracker - all silicon tracking under development (CLICdp)
  - PandoraPFA particle flow algorithm
  - flavor tagging, vertex finding, MCTruth matching,....
- DD4hep/DDG4 simulation will use LCIO as EDM
  - either exclusively or through direct binding to an internal EDM
  - new simulation application currently developed for ILD/CLIC
- LC test beams also use LCIO
  - conditions data base LCCD w/ LCIO
  - raw data classes

# LCIO evolution

- LCIO has been used quite successfully by the LC community for more than a decade and the EDM has evolved during this time to meet all requirements of the physics groups
- to meet future demands (e.g. with a real ILC) we plan to improve the I/O layer:
  - task in AIDA-2 proposal: create a fast I/O layer (based on ROOT) using array of structs, keeping EDM interface essentially unmodified
  - work planned as collaboration between DESY and CERN-SFT
  - goal is to evolve LCIO w/o heavily breaking existing code base
- possibility to move towards using HepMC under discussion
  - would offer the possibility for generators to create LCIO files directly
- keep improving LCIO based on community requests

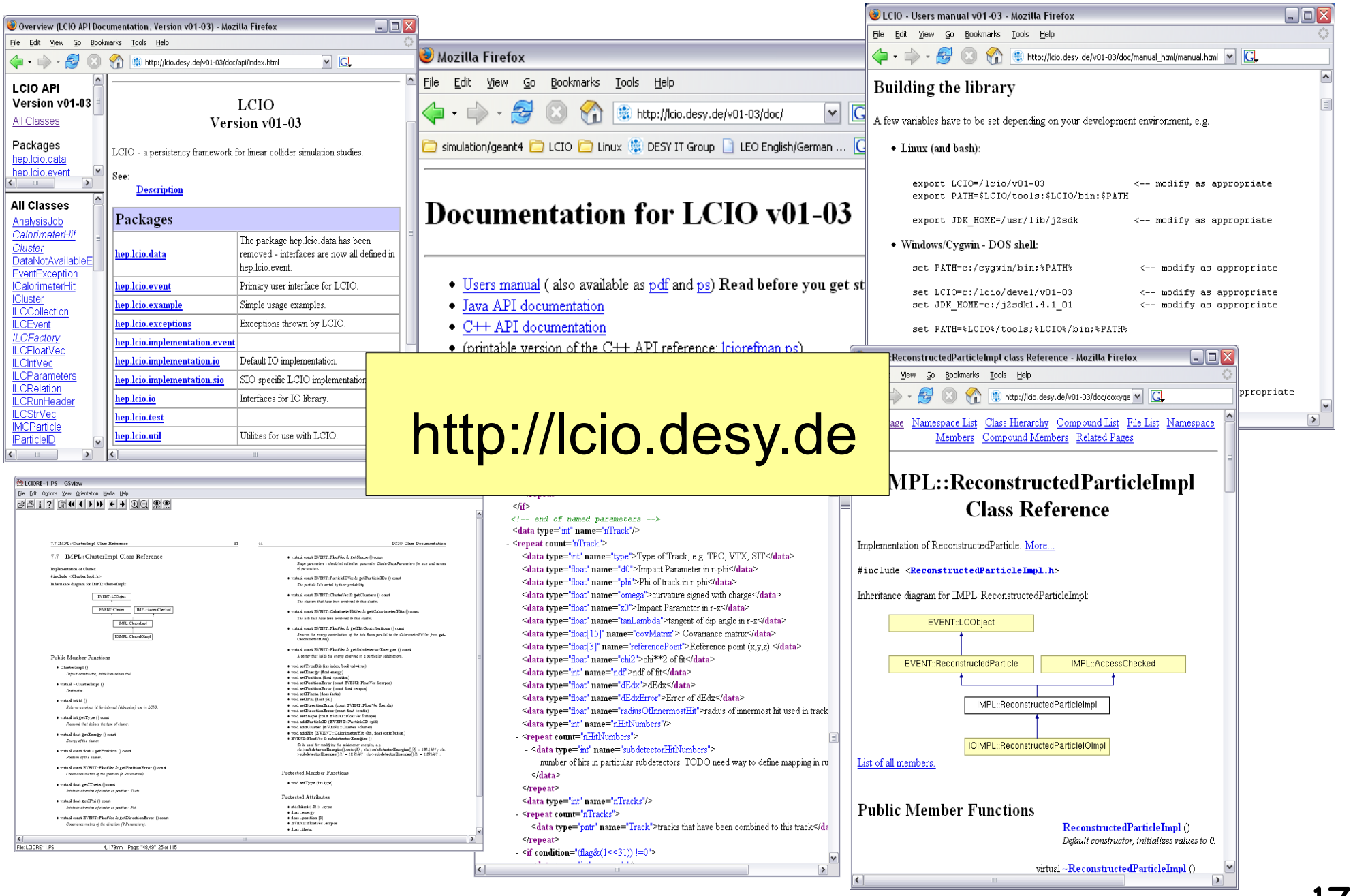
# Summary & Outlook

- LCIO is the EDM and persistency solution for all LC activities from detector design studies, analyses to test beams
- a large code base for reconstruction and analysis exists based on LCIO, a lot of which could be ported/adapted to FCC studies
- next big evolution planned for LCIO will be a high performant I/O

- using LCIO for FCC detector concept studies could work to the mutual benefit of both communities
  - FCC could effectively start right away with simulation studies (DD4hep)
  - LC and FCC could join manpower to improve LCIO as we go ahead

# additional material

# LCIO Online documentation




Frank Gaede, FCC Software Meeting, July 10, 2014

# 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>()**

# 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