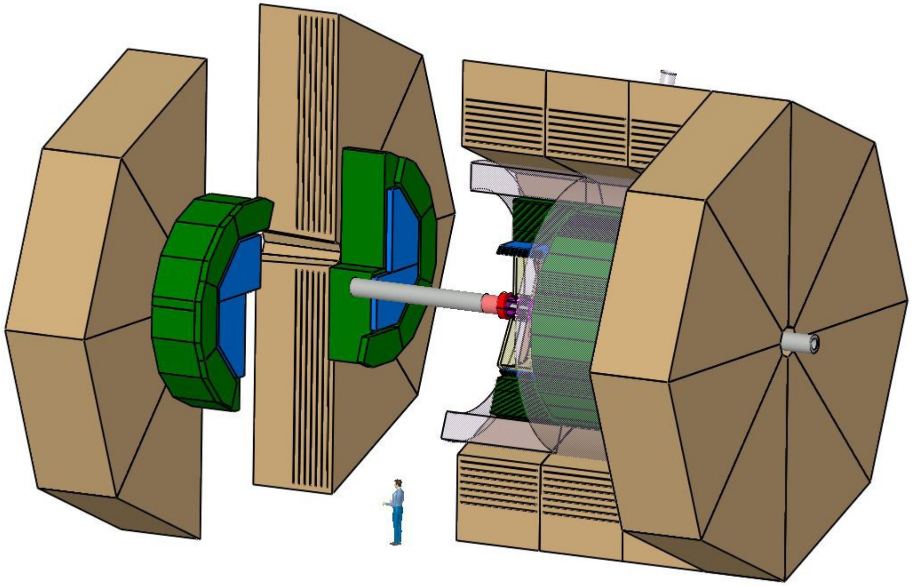
LCGO - geometry description for ILC detectors

Tony Johnson, Norman Graf, SLAC
Frank Gaede, DESY

CHEP 2007, Victoria Canada
September 2-9, 2007

Outline

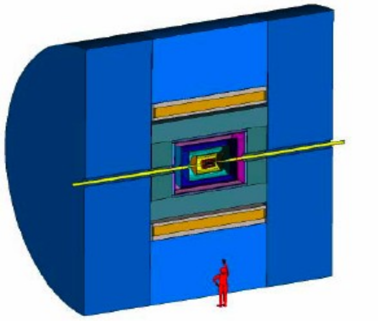
- introduction – motivation
- existing geometry packages for ILC
  - GeomConverter/LCDD – SID geometry
  - Gear – LDC geometry
- LCGO design/software architecture
- gcj – Java compiler and C++
- first results - issues
- summary - outlook



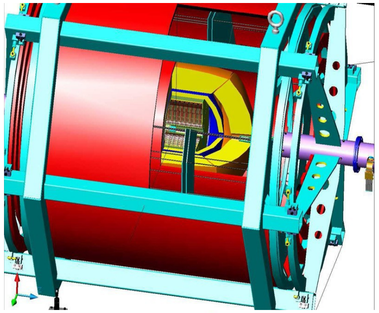
introduction - motivation

- 4 international detector concept studies for the ILC ongoing
- 4 independent sw frameworks exist
- cross comparison and benchmarking require interoperability
- some of which is provided through use of common event data model/ file format LCIO
- desirable to increase this interoperability by developing a common geometry toolkit: **LCGO**
  - initially an SID/LDC (SLAC/DESY) joined project
  - (work in progress)

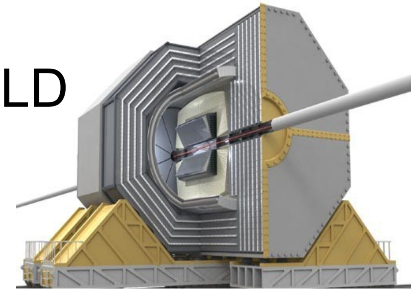
SID



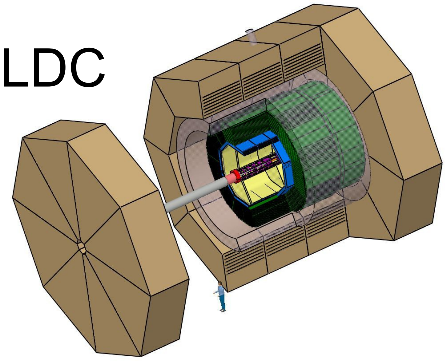
4th



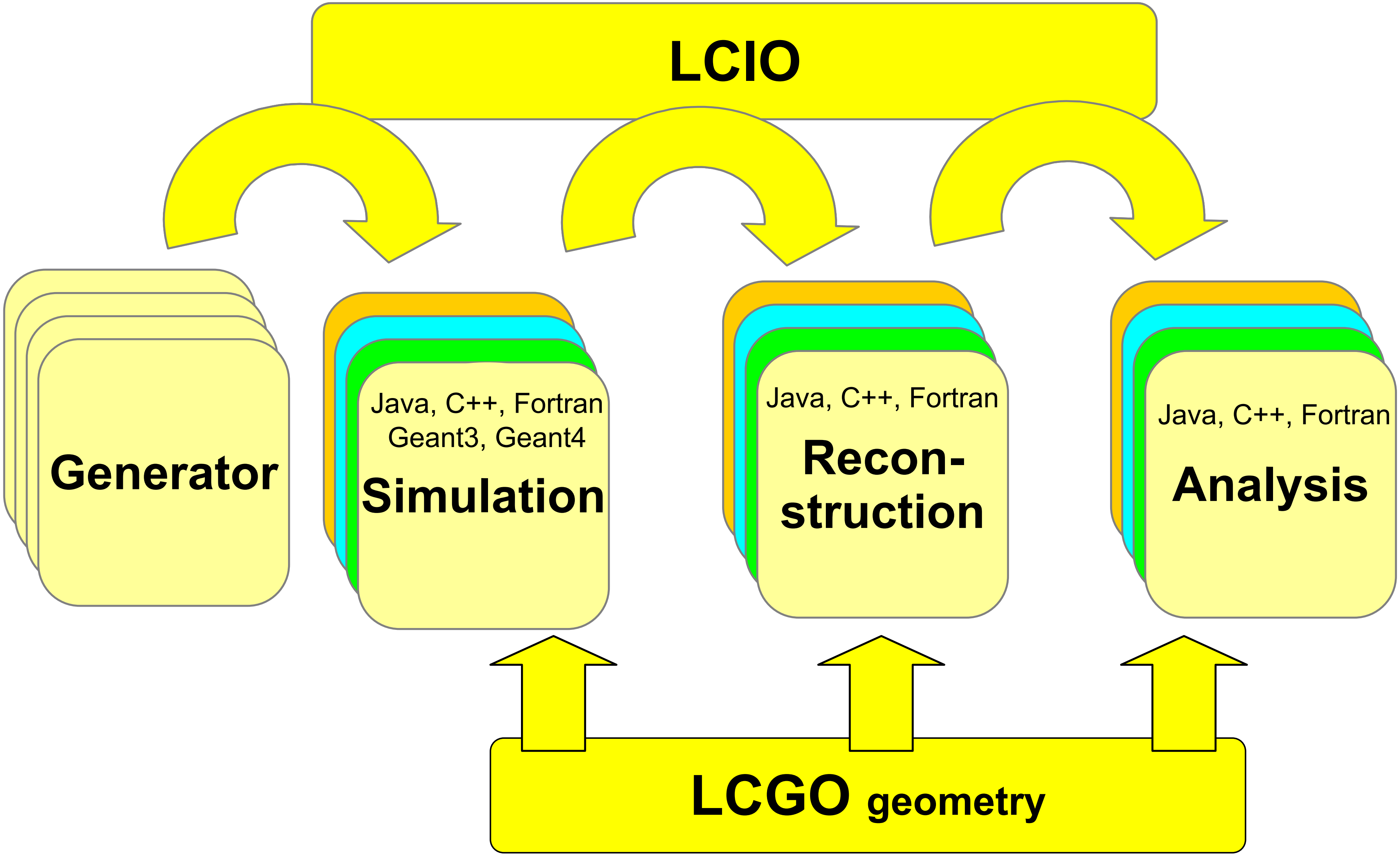
GLD



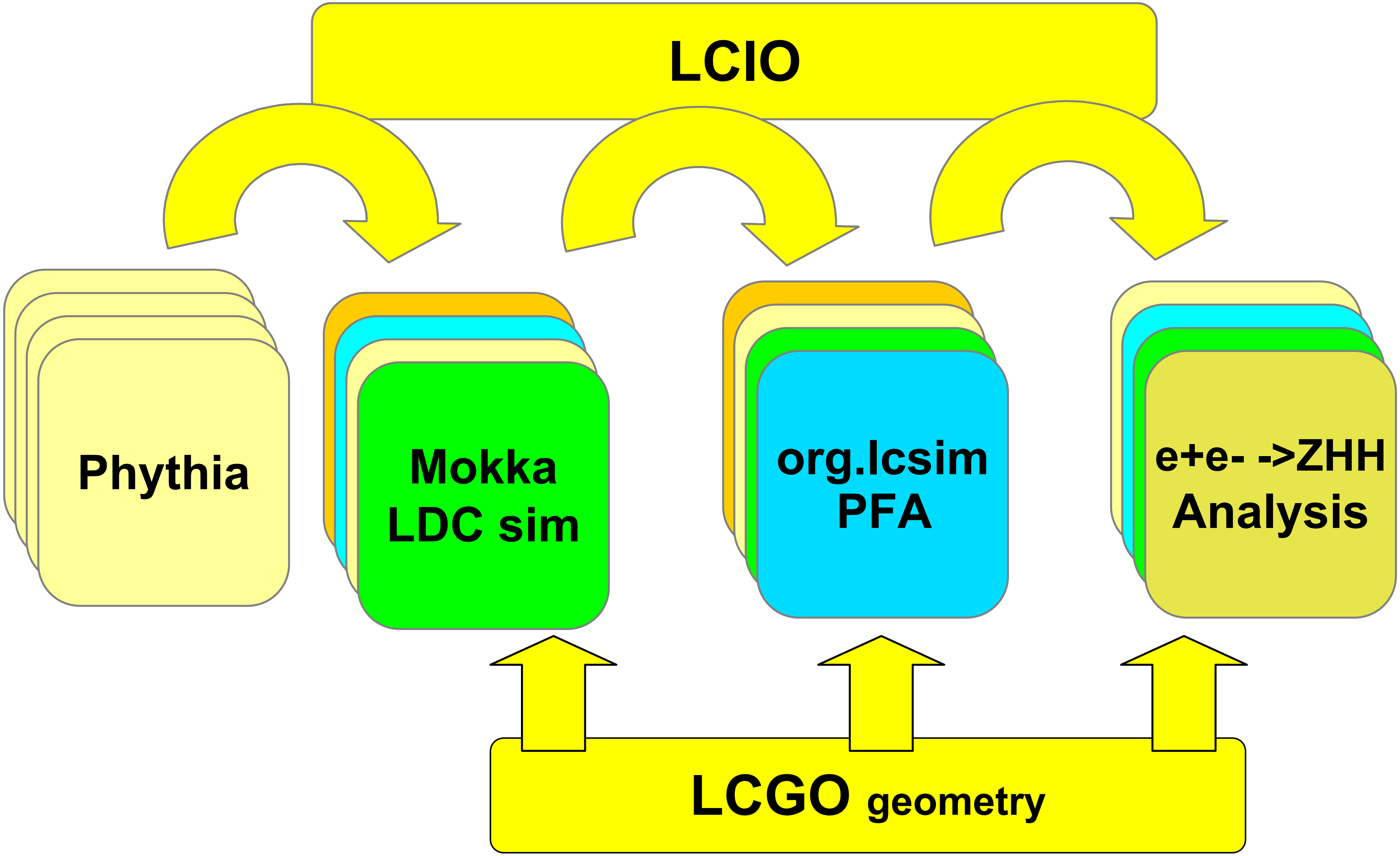
LDC



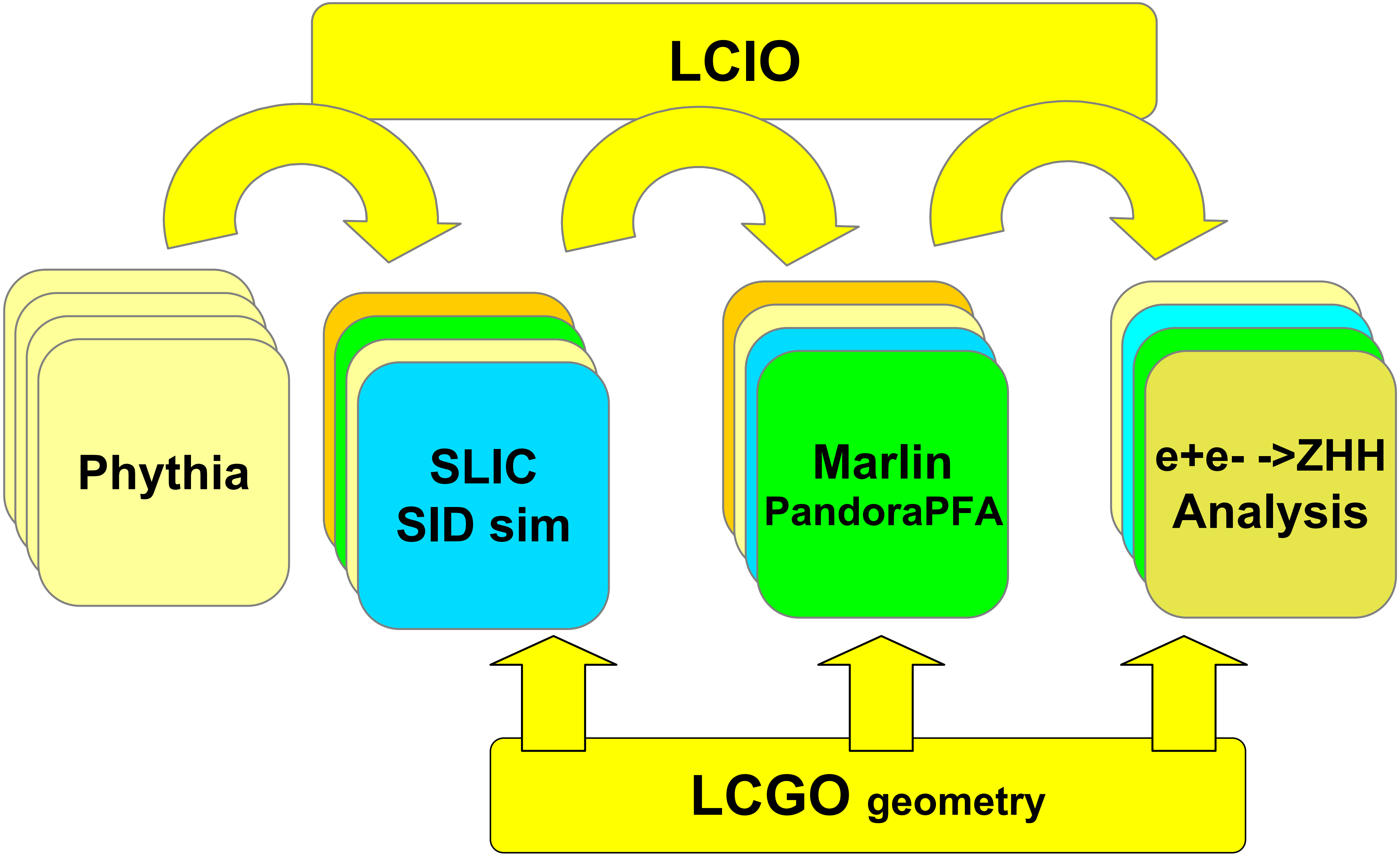
ILC interoperable software chain



ILC interoperable software chain



ILC interoperable software chain



LCIO

Phythia

SLIC
SID sim

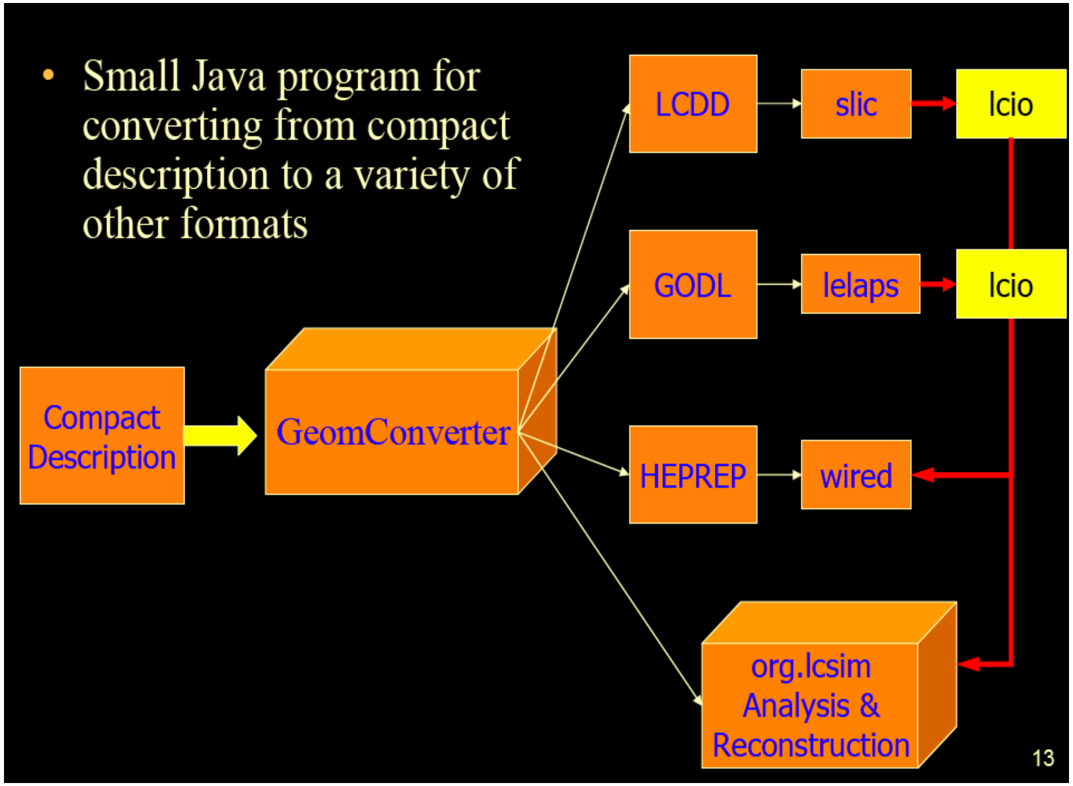
Marlin
PandoraPFA

e+e- ->ZHH
Analysis

LCGO geometry

SID geometry description

- geometry definition in xml files
- GeomCoverter tool (java) provides various representations of geometry:
  - fast & full simulation
  - event display
  - reconstruction
- detailed geometry for geant4:
  - LCDD – extension of GDML
- reconstruction:
  - cellid <-> position
  - local <-> global coordinate
  - find neighbors
  - materials, shapes
  - conditions (time- dependent) data

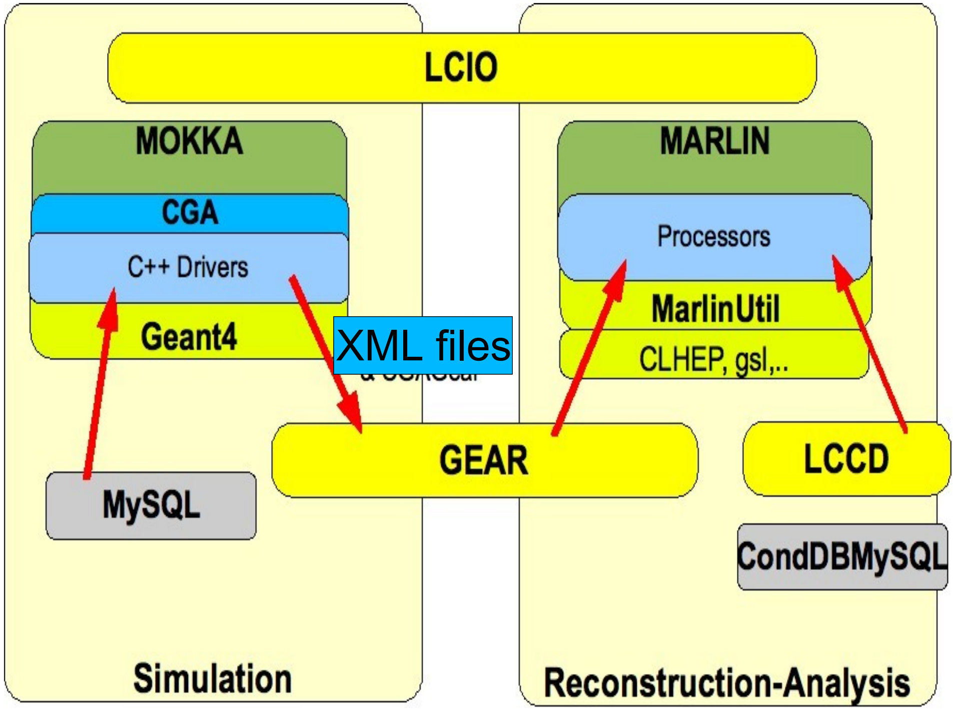


- Small Java program for converting from compact description to a variety of other formats

• rather complete geometry framework for simulation and reconstruction
• bound to Java based system

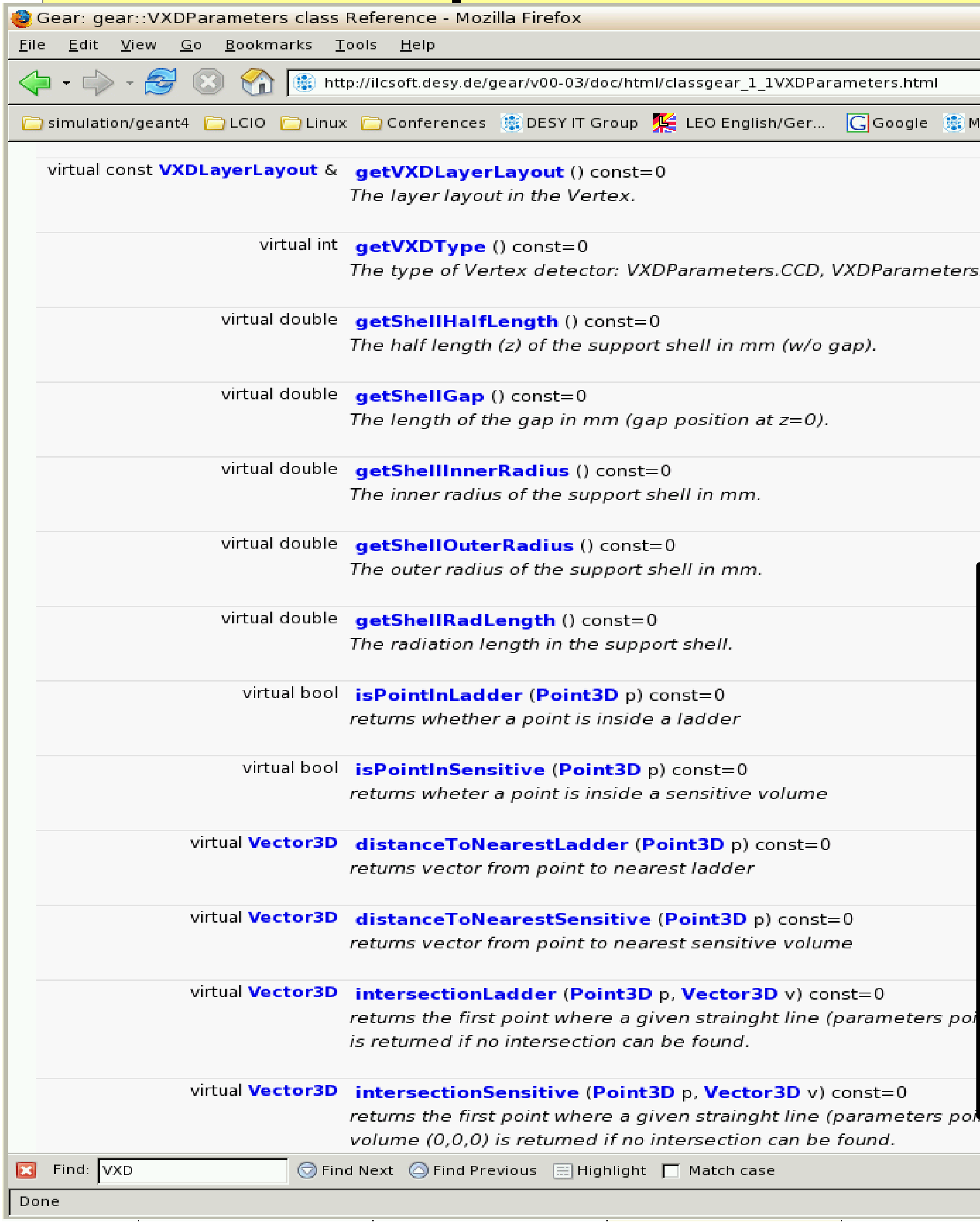
LDC geometry description

- detailed geometry for simulation with Mokka/geant4:
  - MySQL data base with parameters
  - C++ drivers per subdetector
- reconstruction:
  - high level abstract interface:
    - per subdetector type (Hcal,TPC,...) parameters/quantities for reco
      - geometry + some navigation
      - implementation uses xml files
    - abstract interface for detailed geometry &materials:
      - point properties
      - path properties
      - implementation based on geant4 -> rather slow in reco loops



- enforce only one source of geometry:
  - write xml files from Mokka C++ drivers
  - read xml files in Marlin reco job
- 'odd' procedure for tbeams: have to start with simulation geometry

example – GEAR API VXD


Gear: gear::VXDParameters class Reference - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://ilcsoft.desy.de/gear/v00-03/doc/html/classgear_1_1VXDParameters.html
Go
simulation/geant4 LCIO Linux Conferences DESY IT Group LEO English/Ger... Google MyHome Ctime
Find: VXD
Find Next Find Previous Highlight □ Match case
Done

virtual const **VXDLayerLayout** & **getVXDLayerLayout** () const=0
The layer layout in the Vertex.

virtual int **getVXDType** () const=0
The type of Vertex detector: VXDParameters.CCD, VXDParameters.CMOS or VXDPara

virtual double **getShellHalfLength** () const=0
The half length (z) of the support shell in mm (w/o gap).

virtual double **getShellGap** () const=0
The length of the gap in mm (gap position at z=0).

virtual double **getShellInnerRadius** () const=0
The inner radius of the support shell in mm.

virtual double **getShellOuterRadius** () const=0
The outer radius of the support shell in mm.

virtual double **getShellRadLength** () const=0
The radiation length in the support shell.

virtual bool **isPointInLadder** (**Point3D** p) const=0
returns whether a point is inside a ladder

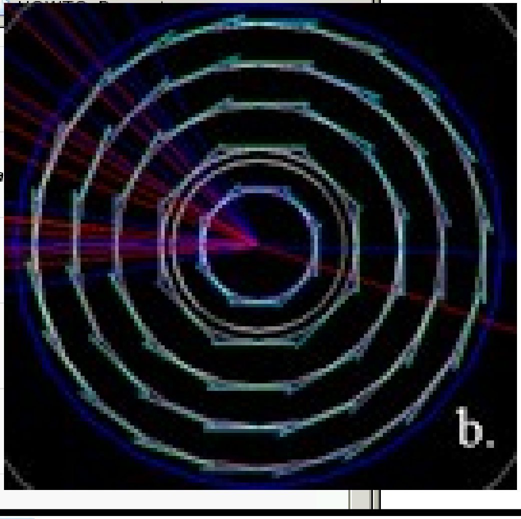
virtual bool **isPointInSensitive** (**Point3D** p) const=0
returns wheter a point is inside a sensitive volume

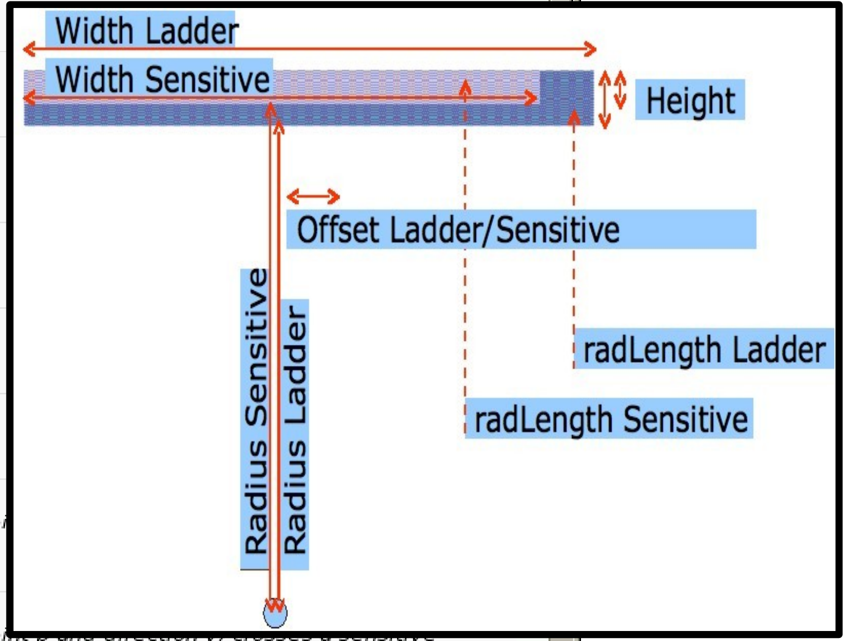
virtual **Vector3D** **distanceToNearestLadder** (**Point3D** p) const=0
returns vector from point to nearest ladder

virtual **Vector3D** **distanceToNearestSensitive** (**Point3D** p) const=0
returns vector from point to nearest sensitive volume

virtual **Vector3D** **intersectionLadder** (**Point3D** p, **Vector3D** v) const=0
returns the first point where a given strainght line (parameters poi
is returned if no intersection can be found.

virtual **Vector3D** **intersectionSensitive** (**Point3D** p, **Vector3D** v) const=0
returns the first point where a given strainght line (parameters point p and direction v) crosses a sensitive volume (0,0,0) is returned if no intersection can be found.



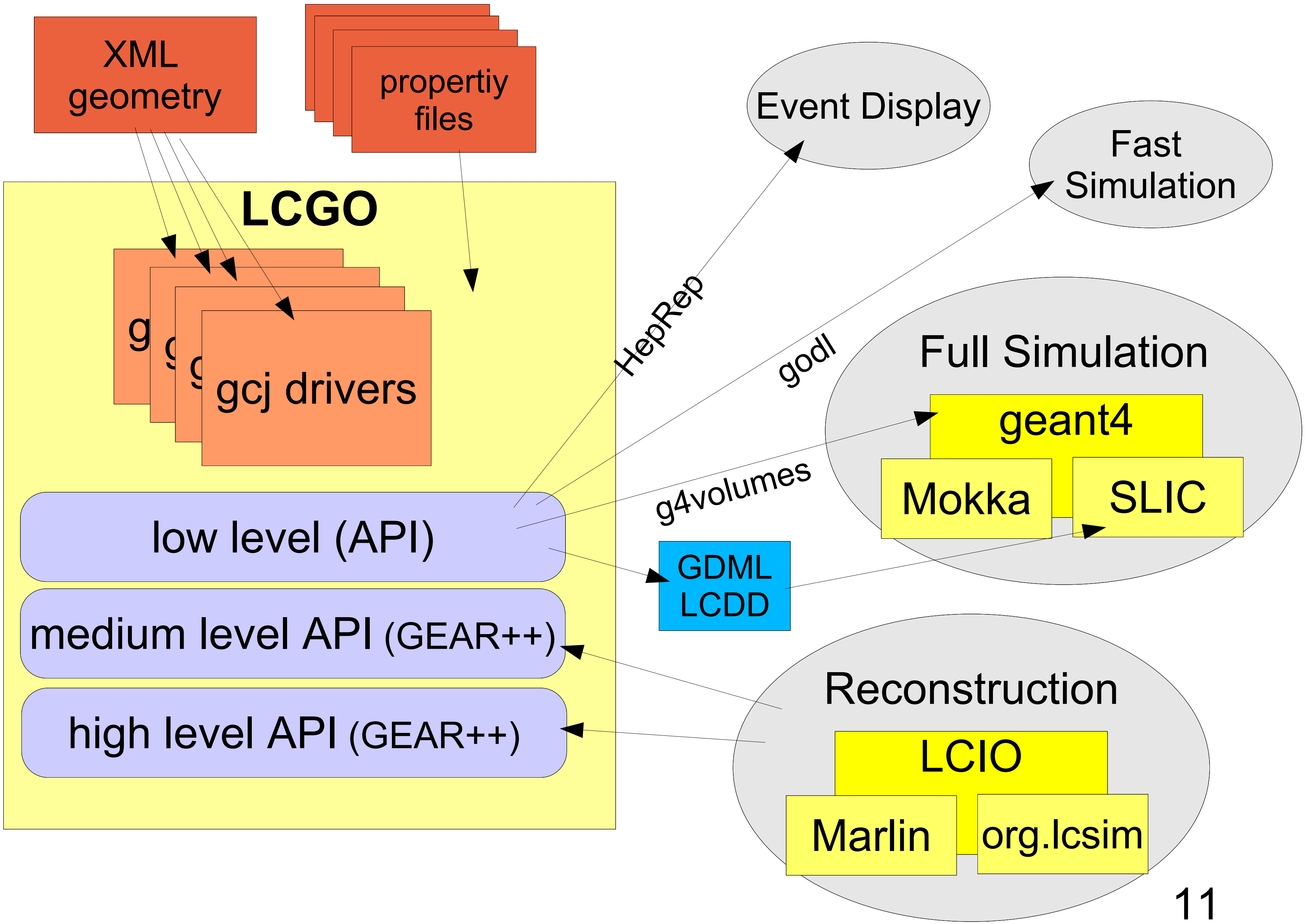


Frank Gaede, CHEP 2007, Victoria, Canada Sep 2-9, 2007

LCGO idea

- combine the best of both systems into a new system that can be used by all ILC detector concepts
- want to have the same functionality in both the Java and the C++ world (a la LCIO)
  - LCIO: two independent implementations (Java/C++) with a common abstract interface:
    - successful, however significant “double” work
- idea to use the same code basis (Java) also for C++ through gcj – CNI
  - have multilevel interface (simulation, reconstruction,...)
  - use xml and properties files for parameters
  - use java-drivers per supdetector

LCGO sw-architecture



planned feature list

- full detailed geomtery description a la geant4/TGeo
- reconstruction:
  - average material volumes
  - intersection with 'next' volume
  - dE/dx
  - field maps
  - access to volumes
    - #layers, thickness, width,..

most of these features already exist in either or both of the current systems

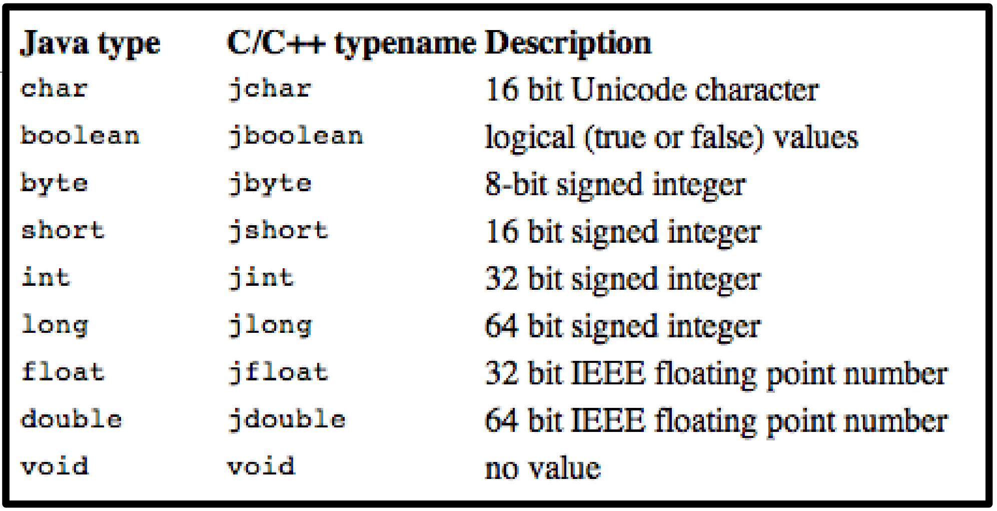
- material database
- field maps
- properties (sampling fractions)
- readout properties
  - cellId <-> position
  - cellid range (noise simulation)
  - cell sizes
  - neighbors
- Vector and Matrix classes
  - ThreeVector, Point3D
  - Planes, cylinders
  - FourVector
  - SymMatrix (covariances)
- navigation
  - where am I (going next)

gcj – GNU Java compiler

- GCJ is a portable, optimizing, ahead-of-time compiler for the Java Programming Language
  - http://gcc.gnu.org/java
- compiles
  - .java to .class files (source to byte code)
  - .java to .o files (source to native code)
  - .class to .o files (byte to native)
- libgcj.so provides:
  - core class libraries (1.4 +some 1.5)
  - garbage collector
  - byte code interpreter

gcj and C++

- Java is essentially a subset of C++
  - classes, virtual functions, static members, method overloading, vtable, constructor methods, object allocation on the heap,...
- plus powerful class library
- gcj and g++ use same ABI
  - object representation and calling conventions
- -> CNI: compiled native interface
  - similar to JNI: a lot simpler – however less portable
  - can be used to call Java code from C++
- issues
  - basic types
  - Java references – C++ pointers
  - Exceptions
    - can't mix Java & C++
  - Arrays - Strings
  - `typedef JArray<jfloat> *jfloatArray;`
  - `JvNewStringLatin1("cstring") ;`

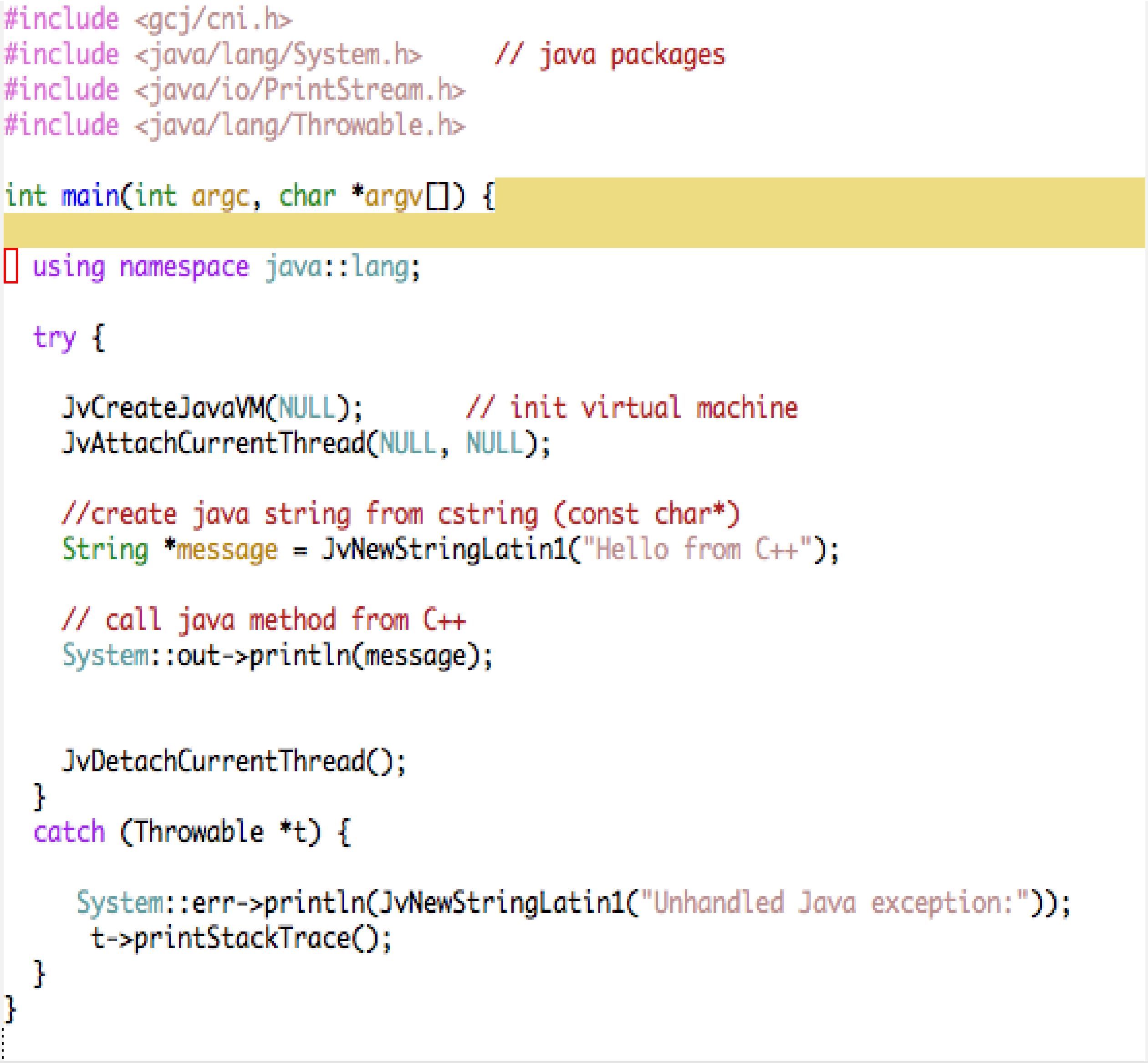


Java type	C/C++ typename	Description
`char`	`jchar`	16 bit Unicode character
`boolean`	`jboolean`	logical (true or false) values
`byte`	`jbyte`	8-bit signed integer
`short`	`jshort`	16 bit signed integer
`int`	`jint`	32 bit signed integer
`long`	`jlong`	64 bit signed integer
`float`	`jfloat`	32 bit IEEE floating point number
`double`	`jdouble`	64 bit IEEE floating point number
`void`	`void`	no value

gcj/C++ example

- can use all of java class libraries (compiled into libgcj.so)
- can compile every user provided java package into shared library:
- gcj --classpath="..." -fPIC -findirect-dispatch -shared -o mypkg.so mypkg.jar
- create java objects in C++ (on the heap) and call methods
  - automatically garbage collected !

'full' java class library available in C++
- no method/class stubs
or interfacing code needed !

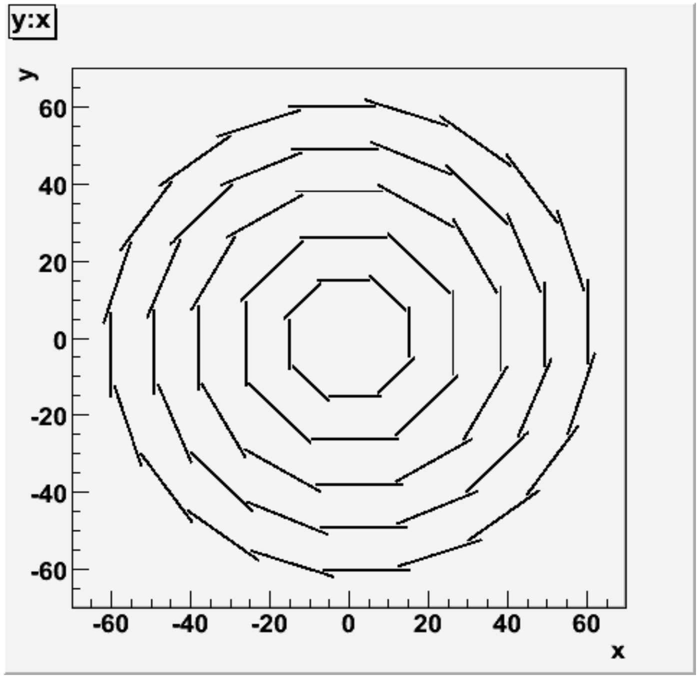


```
#include <gcj/cni.h>
#include <java/lang/System.h>     // java packages
#include <java/io/PrintStream.h>
#include <java/lang/Throwable.h>

int main(int argc, char *argv[]) {

 using namespace java::lang;

 try {

   JvCreateJavaVM(NULL);        // init virtual machine
   JvAttachCurrentThread(NULL, NULL);

   //create java string from cstring (const char*)
   String *message = JvNewStringLatin1("Hello from C++");

   // call java method from C++
   System::out->println(message);


   JvDetachCurrentThread();
 }
 catch (Throwable *t) {

    System::err->println(JvNewStringLatin1("Unhandled Java exception:"));
     t->printStackTrace();
 }
}
```

LCGO prototype

- part of core libraries of org.ldcsim/GeomConverter compiled to shared library
  - missing part uses Java 1.5 features
    - -> need to wait for gcc4.3/gcj or reimplement with 1.4
- part of GEAR ported to Java
  - complete Vertex detector geometry description plus navigation code
  - user parameters
  - compiled to shared library
- successful test programs in Java and C++
    - reading of xml-description
    - accessing the navigation code
  - -> proof of concept

results - issues



- testing the speed of navigation code:
  - create 6M points in 3D and call
  - gear::VXD::isPointInLadder(p) ;
    - simple algorithm using planes (boxes)
    - vector3D and scalar products
- result:
  - Java code compiled to native is ~4 times slower than plain C++
  - same code with JDK1.6 only ~25% slower !
  - not understood and somewhat unexpected
  - if this persists, concept needs to be revised - at least navigation part is performance critical for reconstruction
- possibly gcc4.3 will have improved performance (and Java1.5 support)

Summary & Outlook

- LCGO is a project for a geometry package for linear collider detectors
- key idea is to use code and concepts from existing packages GeomConverter/LCDD and GEAR in Java - compiled into native with gcj for the C++ world
- currently performance issues under study
- gcj provides a convenient way together with CNI for using existing Java code from C++
  - almost complete Java1.4 class library exists
  - Java1.5 under development (gcc4.3)
  - provides interesting alternative to JNI