Introduction to Geant4

Andrea Di Simone

Outline

- Introduction
  - simulation: why?
  - G4: history
  - OO software
- The Geant4 Toolkit
  - internals
  - materials
  - geometry
  - visualization
  - physics processes
  - event generation
  - hits and sensitive detectors

Introduction

Simulation: why?

- Simulation has a crucial role in today's HEP experiments.
- It is the easiest and more reliable way to answer to many important questions:
  - Design of the experiment
    - Do I really have to use Al for this particular part of the detector, or I can live with Cu?
  - Evaluation of the discovery potential of the experiment
    - Suppose that I can control the alignment of the different subdetectors only up to a certain precision, what would be the impact on physics results?
  - Development of reconstruction and analysis code
    - What's the impact of Bremmsstrahlung in the tracker on the energy reconstruction? How can I correct for it?

What is detector simulation?

- A detector simulation program must provide the possibility of describing accurately an experimental setup (both in terms of materials and geometry)
- The program must provide the possibility of generating physics events (kinematics) and efficiently track particles through the simulated detector
- The interactions between particles and matter must be simulated by taking into account all possible physics processes, for the whole energy range
- The possibility of recording at run time all quantities needed for reproducing the experiment's behavior must be provided
- Some graphic support must be in place
- …and much more...

Why a new version of Geant?

- Geant3 was a detector simulation program developed for the LEP era
  - Fortran, ZEBRA
  - Electromagnetic physics directly from EGS
  - Hadronic physics added as an afterthought (and always by interfacing with external packages)
  - Powerful but simplistic geometry model
  - Physics processes very often limited to LEP energy range (100 GeV)
  - (Painfully) debugged with the help and the collaboration of 100s of physicist from all over the world
- LHC detectors need powerful simulation tools for the next 20 years
  - reliability, extensibility, maintainability, openness
  - good physics, with the possibility of extending

What's new

- **Geant3**
  - The geometry model is limited to a pre-defined set of basic shapes. Adding a new shape requires changing the code of several (~20) routines. Interface to CAD systems in not possible
  - Tracking is the result of several iterations (by different people), resulting in difficult to maintain (and to understand) code.
  - EM physics is built in, but several processes are missing and their implementation would be very hard
  - Hadronic physics is implemented via external (and obsolete) packages. Modifications require the author's intervention
- **Geant4**
  - The geometry has been based since the beginning on a CAD-oriented model. The introduction of a new shape does not influence tracking
  - Tracking has been made independent from geometrical navigation, tracking in electromagnetic fields (or any field) has been improved
  - EM and hadronic physics implemented in terms of processes. A process can be easily added or modified by the user and assigned to the relevant particles with no change in the tracking. The cut philosophy has been changed so as to make result less dependent on the cuts used for the simulation. Framework for physics parameterisation in place

Geant4 - How?

- The design concepts behind the Geant4 development have been:
  - ease of maintainability
  - openness of the design
  - performance
  - development by a rather substantial group of physicists
- An Object Oriented approach facilitates the achievement of these goals
  - the problem domain is naturally split into categories which can be tackled by different groups (with different expertise)
  - the process (if done properly) is self-documenting. The components are relatively independent from each other and can easily be replaced without affecting the whole program too much

Geant4 - How?

- Geant4 is a toolkit
  - the framework part in Geant4 has been reduced to a minimum, so as to allow the user to implement his/her own program structure
  - bits and pieces of Geant4 can be used independently from the rest
  - no main program is provided
  - libraries have been made as granular as possible, in order to reduce (re-) compilation time and to allow the user to only link against those parts of G4 which are needed
- C++ as implementation language
  - de-facto standard for what concerns OO programming these days
  - high performance
  - big commercial support, well known in the scientific world
  - there is practically no Fortran in Geant4 anymore, hence the user must know some elements of C++ in order to get the best from G4

OO software

- The developer not only defines the data contained in a given structure, but also the operations one can do with the data
- We call such a structure an object
- In C++ objects are **instances** of a **class**. Data inside a class are called members, while the possible operations are methods.

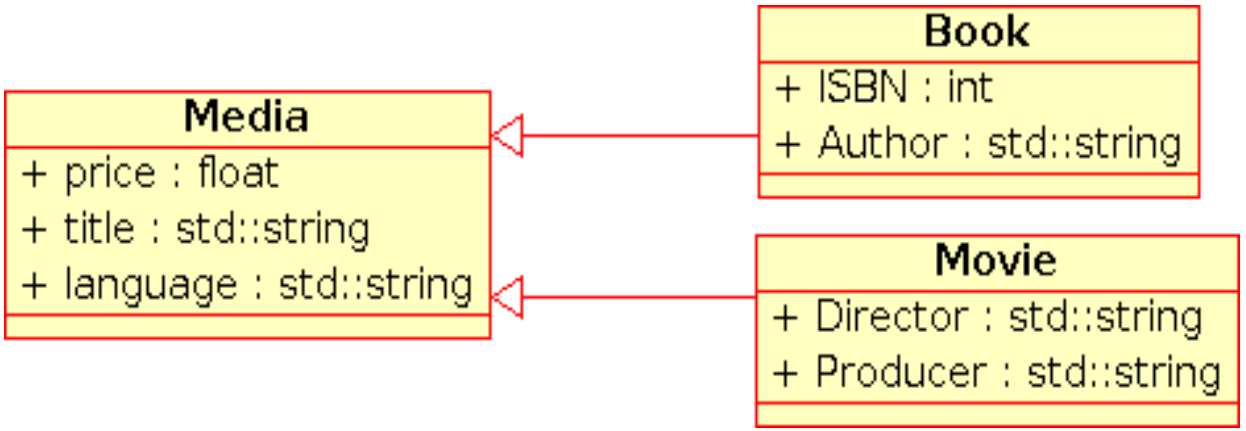
```
int i=123;
double d=1.23;

Vector v(1,2,3);          [Vector::Vector(double x, double y, double z)]
String s("this is a string");   [String::String(char* string)]

double l=v.Length();      [double Vector::Length()]
String s1=s.Invert();     [String String::Invert()]
```

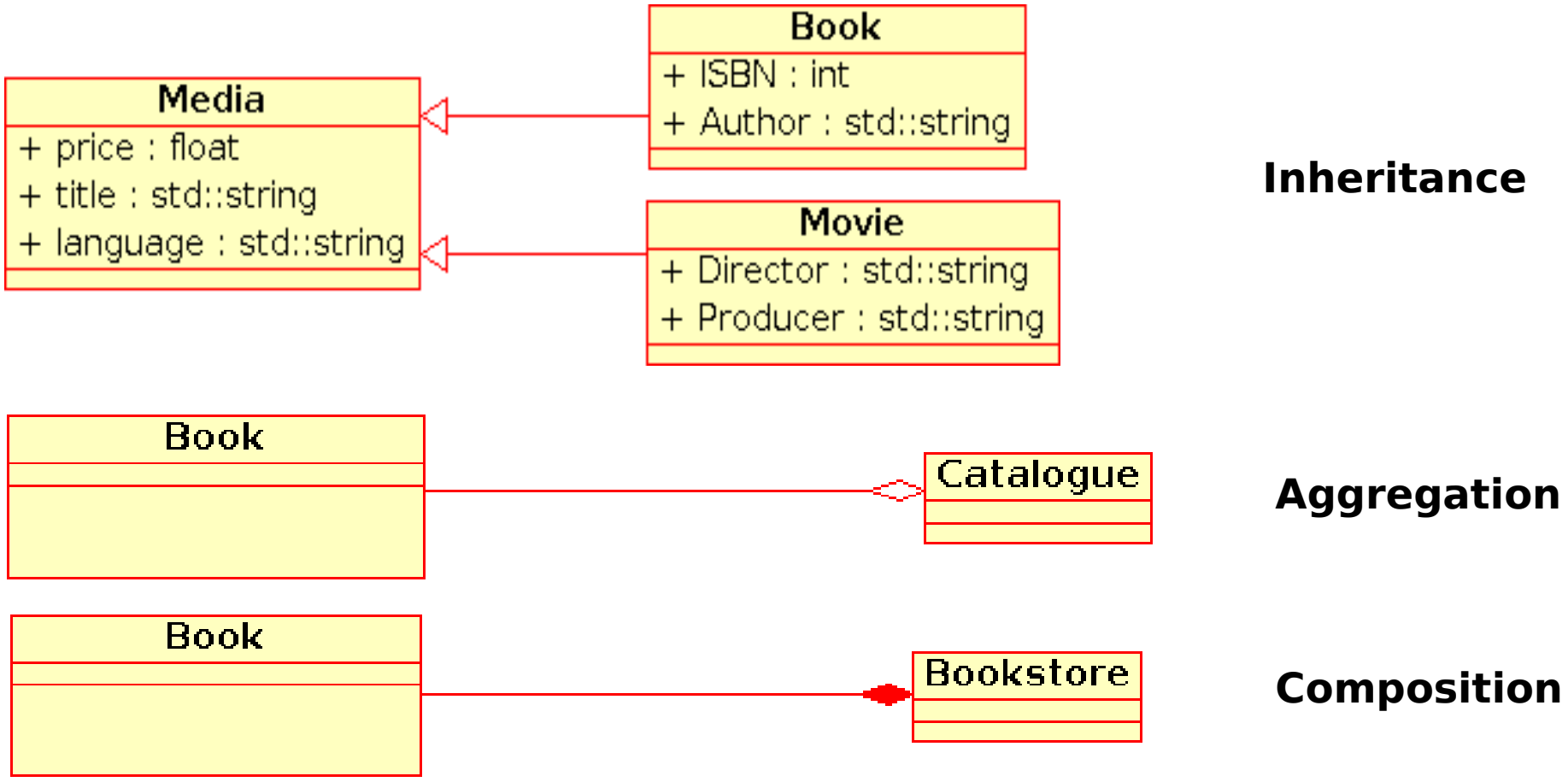
OO software (2)

- In C++, a class can inherit its characteristics from other classes
- The “daughter” class has all members and methods of the “mother”
  - can add more members of methods, or provide new implementation of the inherited methods
- Example: a class Media, and its derived classes



OO software (3)

- UML provides an easy way to represent the way an object is related to other objects

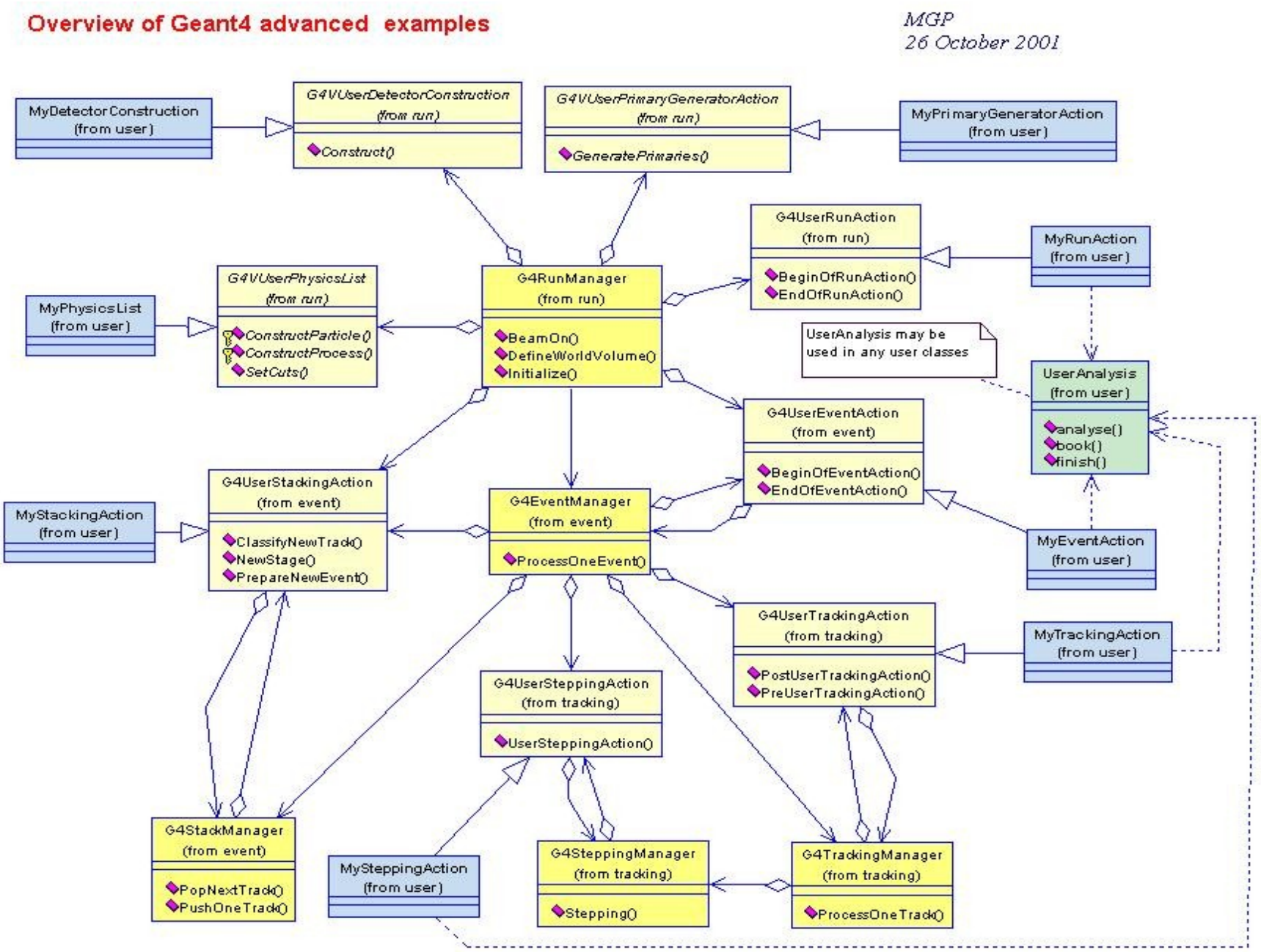


Naming conventions

- All Geant4 source files have a .cc extensions, all Geant4 header files carry a .hh extension
- All Geant4 classes have their name prefixed with a G4
  - G4RunManager, G4Step, G4LogicalVolume
- Abstract classes add a V to the prefix
  - G4VHit, G4VPhysicalVolume
- Each word in a composite name is capitalized
  - G4UserAction, G4VPVParameterisation
- Methods and functions obey the same naming conventions as the class names
  - G4RunManager::SetUserAction(), G4LogicalVolume::GetName()

G4 internals

G4: architectural overview



Basic Types

- For basic numeric types, different compilers on different platforms provide different value ranges
- To assure portability, Geant4 redefines the basic types for them to have always the same bit yield

int	→	**G4int**	
long	→	**G4long**	
float	→	**G4float**	
double	→	**G4double**	
bool	→	**G4bool**	**(native, or from RW, or from CLHEP)**
string	→	**G4String**	**(from RW, and now from STL)**

- the definitions of these types are all placed in a single header file (globals.hh), which also provides inclusion of all system headers, as well as global functions needed by the Geant4 kernel

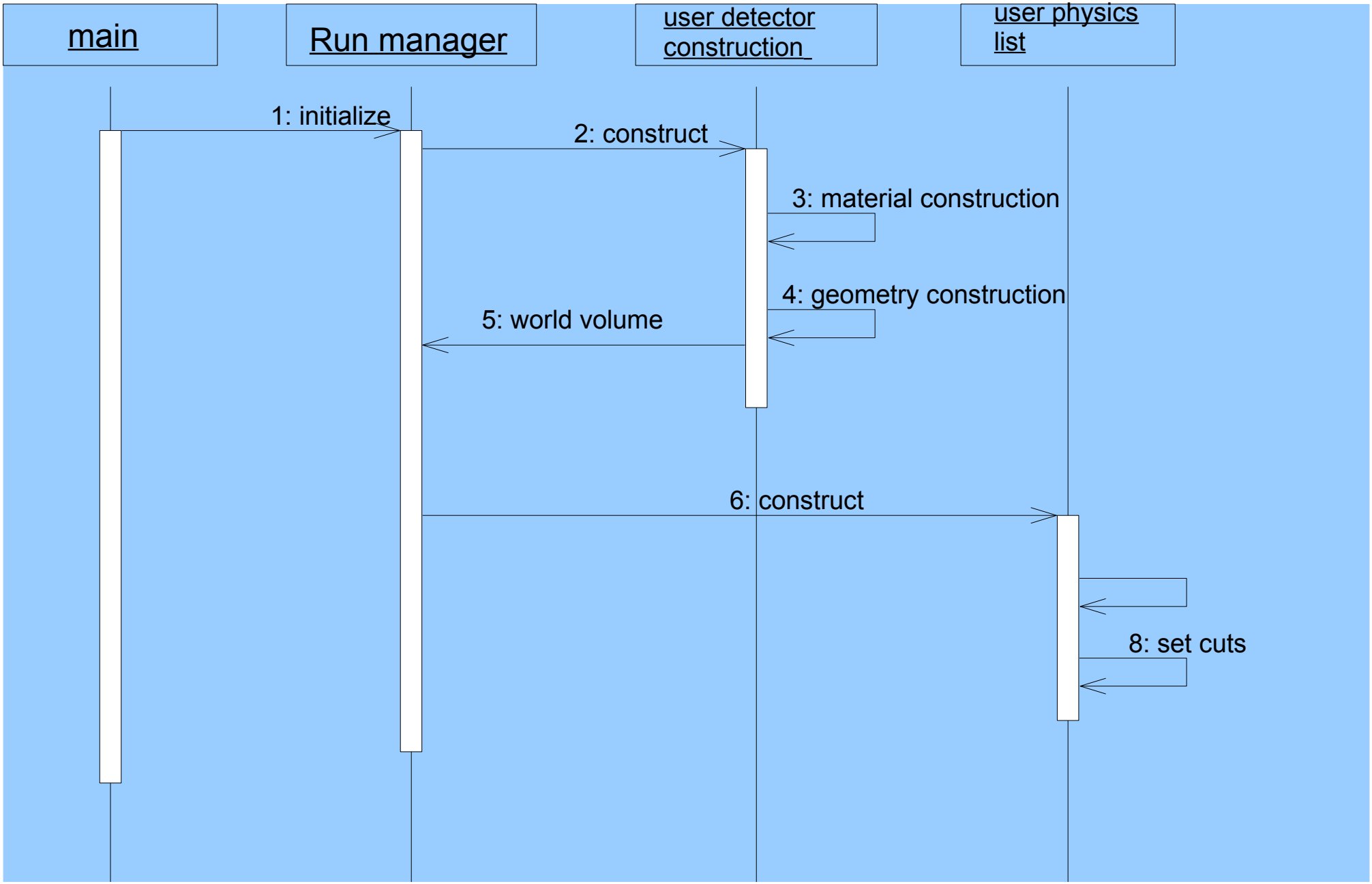
The main program

- Geant4 is a detector simulation toolkit, hence it does not provide a main() method
- Users must supply their own main program to build their simulation program
- The G4RunManager class is the only manager class in the Geant4 kernel which should be explicitly instantiated in the main program to specify:
  - How the detector geometry should be built
  - Which physics processes one is interested in
  - How the primary particles in an event should be produced
  - Additional requests during the simulation procedures

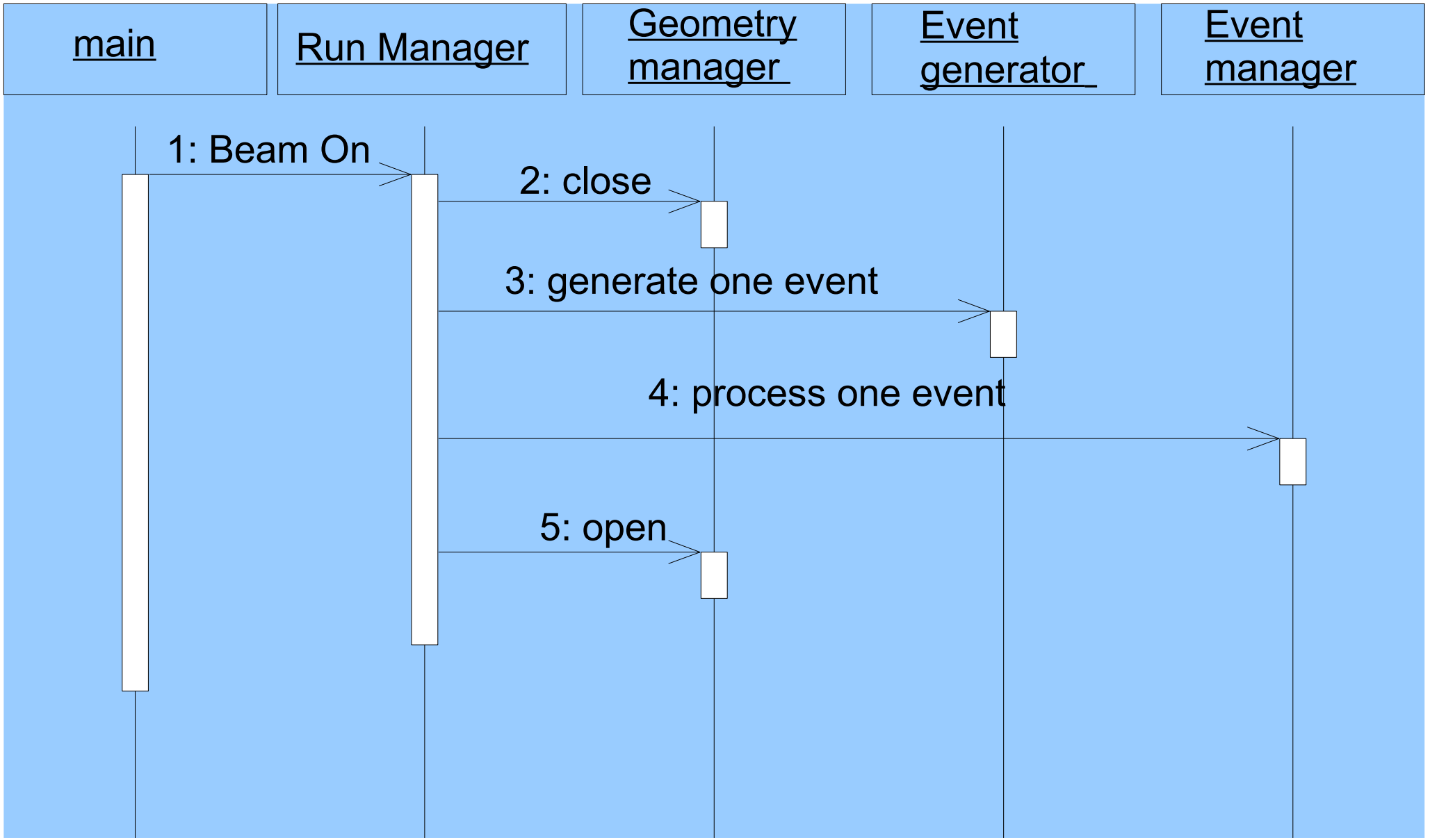
G4RunManager

- G4RunManager is the root class of the Geant4 hierarchy
- It controls the main flow of the program:
  - Construct the manager classes of Geant4 (in its constructor)
  - Manages initialization procedures including methods in the user initialization classes (in its method Initialize())
  - Manages event loops (in its method BeamOn())
  - Terminates manager classes in Geant4 (in its destructor)
- The method Initialize() takes care of building the detector geometry (as specified by the user), the physics processes and of setting all parameters needed for G4 to run
- The detector setup and the physics processes and cuts cannot be modified during a run. The G4RunManager must be notified if one of these were to change, before a new run

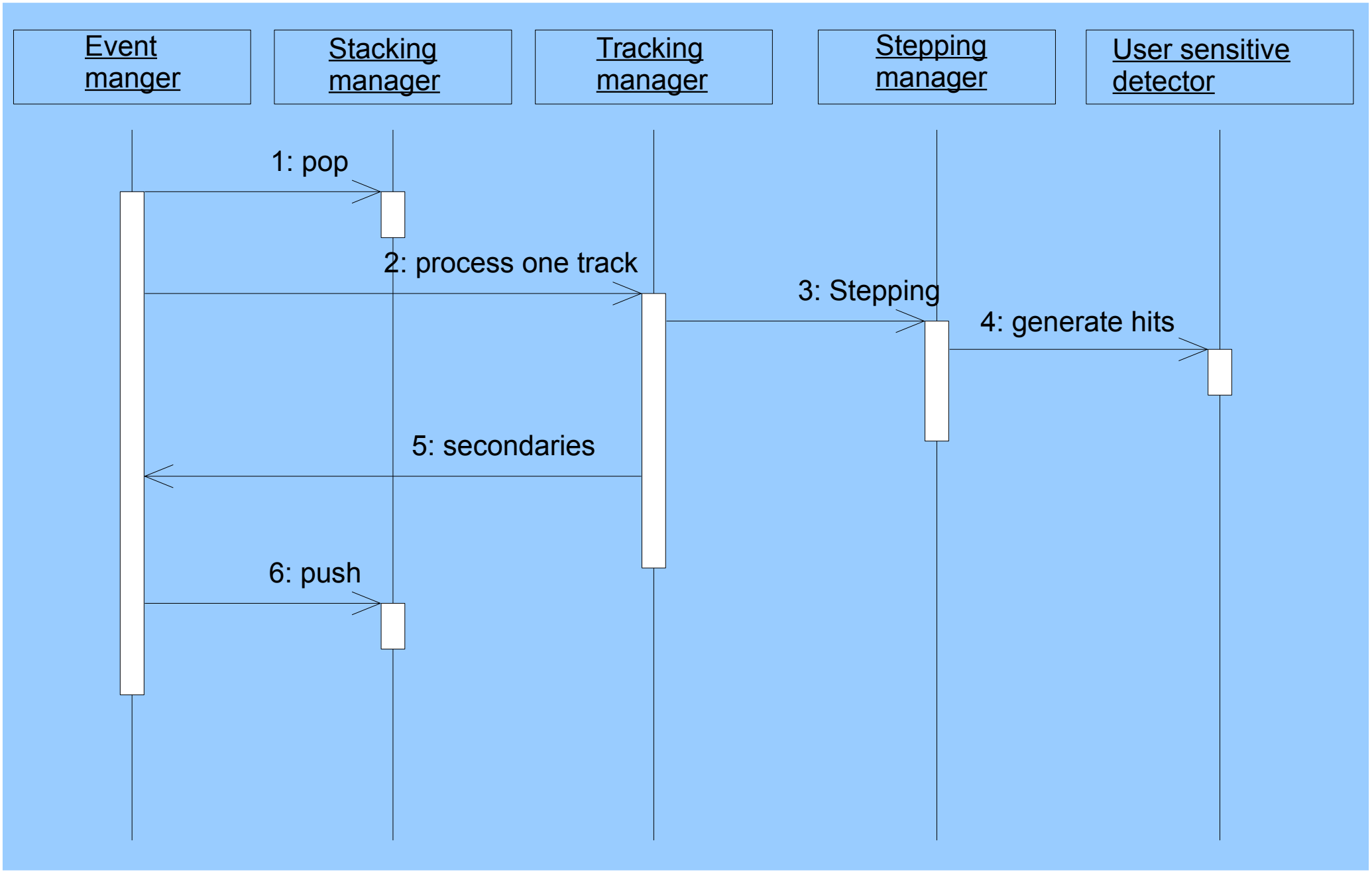
G4RunManager - init



G4RunManager - beamOn



Event processing



Tracks and steps

- Several classes involved in tracking
- Each one carries only the information which is strictly necessary

G4Step:
- PreStep and PostStepPoint
- step length
- pointer to G4Track
- total deposited energy

G4StepPoint:
- (x, y, z, t)
- (px, py, pz, Ek)
- pointer to physical volume
- distance to volume boundary
- pointer to physics processes
- track length.

G4Track:
- (x,y,z)
- momentum direction
- kinetic energy
- track length
- pointer to particle
- pointer to physical volume
- Track ID
- parent's Track ID
- current step number
- (x,y,z) at track origin
- momentum at track origin
- kinetic energy at track origin
- pointer to originating process

User initialization and action classes

- Geant4 has two kinds of user defined classes
  - User initialization classes
    - used for customizing the Geant4 initialization
    - assigned to G4RunManager by invoking the SetUserInitialization() method
  - User action classes
    - used during the run processing
    - assigned to G4RunManager by invoking the SetUserAction() method
- The implementation of three user defined classes is mandatory
  - setting up of the geometry
  - event kinematics
  - physics processes

Mandatory user classes

- Three user classes have to be implemented by the user (two initialization classes and one action class)
- The base classes of these mandatory classes are abstract and no default implementation is provided
- G4RunManager checks whether objects belonging to these classes have been instantiated when Initialize() and BeamOn() are invoked
- Users must inherit from the abstract base classes provided by Geant4 and derive their own classes

Mandatory user classes (2)

- G4VUserDetectorConstruction (initialization)
  - the detector set-up must be described in a class derived from this one
    - Materials
    - Geometry of the detector
    - Definition of sensitive detectors
    - Readout schemes
- G4VUserPhysicsList (initialization)
  - Particles and processes to be used in the simulation
  - cutoff parameters
- G4VUserPrimaryGeneratorAction (action)
  - Primary event kinematics

Optional user action classes

- G4UserRunAction
  - run by run
- G4UserEventAction
  - event by event
- G4UserStackingAction
  - to control the order with which particles are propagated through the detector
- G4UserTrackingAction
  - Actions to be undertaken at each end of the step
- G4UserSteppingAction
  - Actions to be undertaken at the end of every step

An example of main()

```
#include “G4RunManager.hh”          // from Geant4, declaration of the run manager
#include “G4UImanager.hh            // from Geant4, declaration of the User Interface manager
#include “MyDetectorConstruction.hh” // by the user; definition of the detector geometry
#include “MyPhysicsList.hh”         // by the user, list of physics processes to be added
#include “MyPrimaryGenerator.hh”    // by the user, kinematics of the event

int main () {
        G4RunManager* runManager=new G4RunManager;                    // the run manager
        runManager->SetUserInitialization(new MyDetectorConstruction);  // the geometry
        runManager->SetUserInitialization(new MyPhysicsList);         // the physics
        runManager->SetUserAction(new MyPrimaryGenerator);            // the kinematics
        runManager->Initialize();                                     // run initialization
        G4UImanager* UI=G4UImanager::GetUIpointer();                  // pointer to the UI
        UI->ApplyCommand(“run/verbose 1”);                            // set the “print”level
        int numberOfEvent=3;                                          // nr. of evts to be run
        runManager->BeamOn(numberOfEvent);                           // generate the events
        delete runManager;                                            // end of run
        return 0;
}
```

Geant4 and CLHEP

- Geant4 makes a rather substantial use of CLHEP components
  - System of units
  - Vector classes and matrices
    - G4ThreeVector (typedef to Hep3Vector)
    - G4RotationMatrix (typedef to HepRotation)
    - G4LorentzVector (typedef to HepLorentzVector)
    - G4LorentzRotation (typedef to HepLorentzRotation)
  - Geometrical classes
    - G4Plane3D (typedef to HepPlane3D)
    - G4Transform3D (typedef to HepTransform3D)
    - G4Normal3D (typedef to HepNormal3D)
    - G4Point3D (typedef to HepPoint3D)
    - G4Vector3D (typedef to HepVector3D)

System of units

- Geant4 offers the possibility to choose and use the units one prefers for any quantity
- Geant4 uses an internal and consistent set of units based on:
  - millimeter (mm)
  - nanosecond (ns)
  - Mega electron Volt (MeV)
  - Positron charge (eplus)
  - Degree Kelvin (kelvin)
  - Amount of substance (mole)
  - Luminosity intensity (candela)
  - Radian (radian)
  - steradian (steradian)

3-Vectors

- Geant4 makes use of the CLHEP HepVector3D and Hep3Vector for implementing several 3-dimensional object (G4ThreeVector, G4ParticleMomentum…)
- The definition of a 3-vector is pretty straightforward:
  - G4ThreeVector *p=new G4ThreeVector(10,20,100);
- Every component can be accessed very easily:
  - G4double px=p->x();
- and set very easily;
  - p->setZ(50);
- the components in polar coordinates are given by
  - phi(), theta(), mag()
- and set using:
  - setPhi(), setTheta(), setMag()

3-Vectors (2)

- They can be normalized:
  - p->unit();
- rotated around one of the cartesian axes
  - p->rotateY(2.73);
- or around any other 3-vector
  - p->rotate(1.57,G4ThreeVector(10,20,30));
- for reference:
  - google -> “Hep3Vector”

Rotation Matrices

- Geant4 uses the rotation matrix implementation which comes with CLHEP (HepRotation, typedef'd into G4RotationMatrix)

```
#include "G4RotationMatrix.hh"
....
G4RotationMatrix *rm = new G4RotationMatrix;
```

- You can then rotate about the coordinate axes:

```
rm->rotateX(45*deg);                    // rotation about X
```

- and combine several rotations into a 3D one:

```
rm->rotateX(30*deg);
rm->rotateY(20*deg);
```

Materials

Materials

- In nature, general materials (compounds, mixtures) are made by elements and elements are made by isotopes. These are the three main classes designed in Geant4
- The G4Element class describes the properties of the atoms: atomic number, number of nucleons, atomic mass, shell energy…
- The G4Isotope class allows to describe the isotopic composition of a material
- The G4Material class describes the macroscopic properties of the matter: density, state, temperature, pressure, radiation length, mean free path, dE/dx…
- G4Material is the class visible to the rest of the toolkit and it is used by the tracking, the geometry and physics

Define a simple material

- A simple material can be created by specifying its name, density, mass of a mole and atomic number:

```
#include “G4Material.hh”
…
G4double density=1.390*g/cm3;
G4double a=39.95*g/mole;
G4double z=18.;
G4String name;
…
G4Material* LAr = new G4Material(name=“Liquid Argon”, z, a, density);
```

- The pointer to the material will then be used to specify the material a given volume is made of

Define a molecule

- A molecule is built from its components, by specifying the number of atoms in the molecule.

```
#include “G4Element.hh”
#include “G4Material.hh”
...
G4double a=1.01*g/mole;
G4double z;
G4String name,symbol;
G4Element* H = new G4Element(name=“Hydrogen”, symbol=“H”,z=1.,a);

a=16.0*g/mole;
G4Element* O = new G4Element(name=“Oxygen”, symbol=“O”,z=8.,a);

G4double density=1.000*g/cm3;
G4int ncomponent,natoms;
G4Material* H2O = new G4Material(name=“Water”,density,ncomponents=2);
H2O->AddElement(H,natoms=2);
H2O->AddElement(O,natoms=1);
```

Define a mixture (by fractional mass)

- Air is built from Nitrogen and Oxygen by giving the fractional mass of each component

```
#include “G4Element.hh”
#include “G4Material.hh”
...
G4double a=14.01*g/mole;
G4double z;
G4String name,symbol;
G4Element* N = new G4Element(name=“Nitrogen”, symbol=“N”,z=7.,a);

a=16.0*g/mole;
G4Element* O = new G4Element(name=“Oxygen”, symbol=“O”,z=8.,a);

G4double fractionmass,density=1.290*mg/cm3;
G4int ncomponent,natoms;
G4Material* Air = new G4Material(name=“Air”,density,ncomponents=2);
Air->AddElement(N,fractionmass=70*percent);
Air->AddElement(O,fractionmass=30*percent);
```

Materials as mixtures of materials

- ArCO2 can be defined as mixture of an element and a material:

```
#include “G4Element.hh”
#include “G4Material.hh”
...
G4double a,z,fractionmass,density;
G4String name,symbol;
G4int ncomponents,natoms;

G4Element* Ar = new G4Element(name=“Argon”, symbol=“Ar”,z=18.,a=39.95*g/mole);
G4Element* C = new G4Element(name=“Carbon”, symbol=“C”, z=6., a=12.00*g/mole);
G4Element* O = new G4Element(name=“Oxygen”, symbol=“O”,z=8.,a=16.00*g/mole);

G4Material* CO2 = new G4Material(name=“CO2”,density=1.977*mg/cm3,ncomponents=2);
CO2->AddElement(C,natoms=1);
CO2->AddElement(O,natoms=2);

G4Material* ArCO2=new G4Material(name=“ArCO2”,density=1.8*mg/cm3,ncomponents=2);
ArCO2-> AddElement(Ar,fractionmass=93*percent);
ArCO2-> AddMaterial(CO2,fractionmass=7*percent);
```

Materials in non STP conditions

```
#include “G4Element.hh”
#include “G4Material.hh”
...
G4double a,z,fractionmass,density;
G4String name,symbol, ncomponent,natoms;
G4Element* Ar = new G4Element(name=“Argon”, symbol=“Ar”,z=18.,a=39.95*g/mole);
G4Element* C = new G4Element(name=“Carbon”, symbol=“C”, z=6., a=12.00*g/mole);
G4Element* O = new G4Element(name=“Oxygen”, symbol=“O”,z=8.,a=16.00*g/mole);

G4double temperature=300.*kelvin;
G4double pressure=2*atmosphere;
G4Material* CO2 = new G4Material(name=“CO2”,density=1.977*mg/cm3,ncomponents=2
                                                kStateGas,temperature,pressure);
CO2->AddElement(C,natoms=1);
CO2->AddElement(O,natoms=2);

G4Material* ArCO2=new G4Material(name=“ArCO2”,density=1.8*mg/cm3,ncomponents=2
                                                 kStateGas,temperature,pressure);
ArCO2-> AddElement(Ar,fractionmass=93*percent);
ArCO2-> AddMaterial(CO2,fractionmass=7*percent);
```

Geometry

Detector geometry

- A detector geometry in Geant4 is made of a number of volumes.
- The largest volume is called the World volume. It must contain all other volumes in the detector geometry
- The other volumes are created and placed inside previous volumes, including the World.
- Each volume is created by describing its shape and its physical characteristics and then placing it inside a containing volume.
- The coordinate system used to specify where the daughter volume is placed is the one of the mother.

G4VUserDetectorConstruction

- G4VUserDetectorConstruction is one of the three abstract classes in Geant4 the user MUST inherit from to create his/her implementation of the detector geometry
- An instance of a class inherited from G4VUserDetectorConstruction is passed to the G4RunManager by calling SetUserInitialization(). The Run Manager keeps a pointer to the detector geometry which the user wants to use and checks (before startup) that this pointer has indeed been set
- G4VUserDetectorConstruction's method Construct() is invoked by the Run Manager to set up the detector geometry. Construct() should hence set up everything needed for the geometry definition
- Construct() returns a pointer the the World's Physical Volume

Volumes

- To have a volume implemented in Geant4 one has to go through three steps.
  - A **Solid** is used to describe a volume's shape. A solid is a geometrical object that has a shape and specific values for each of that shape's dimensions
  - A **Logical Volume** is use for describing a volume's full properties. It starts from its geometrical properties (the solid) and adds physical characteristics, like the material, the sensitivity, the magnetic field, the color…
  - What remains to describe is the position of the volume. For doing that, one creates a **Physical Volume**, which places a copy of the logical volume inside a larger, containing volume.

Solids

- The STEP standard supports multiple solid representations
  - Constructive Solid Geometry (CSG)
  - SWEPT solids
  - Boundary Represented solids (BREPs)
- Different representations are suitable for different purposes, applications, required complexity and levels of detail.
  - CSGs give superior performance and they are easy to use, but they cannot reproduce complex solids as used in CAD systems
  - BREPs allow to reproduce the most complex solids, thus allowing the exchange of models with CAD systems, but they are normally quite inefficient, as far as tracking is concerned

G4Box

- The definition of a Box in Geant4 can be found in:

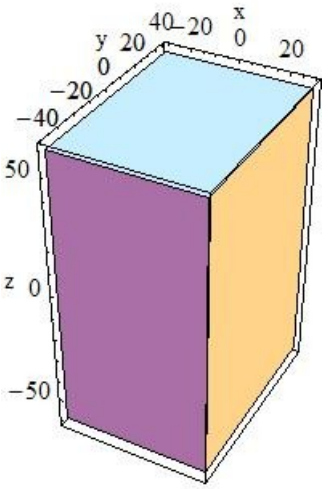
  $G4INSTALL/source/geometry/solids/CSG/include/G4Box.hh

- To create a box use the constructor:

  G4Box(const G4String& pName, G4double pX, G4double pY, G4double pZ)

  where:
  pX: half length in X
  pY: half length in Y
  pZ: half length in Z

  G4Box* a_box=new G4Box("My Box",10*cm,0.5*m,30*cm);



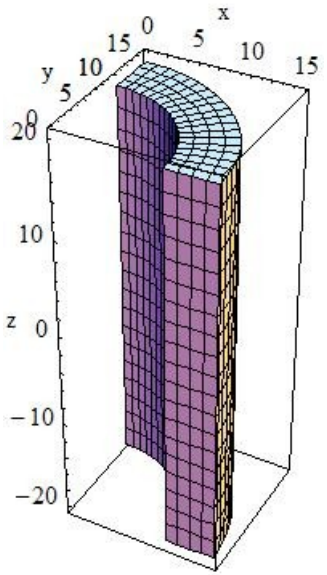
G4Tubs

- The definition of a Tube (or a section of it) is in:

  $G4INSTALL/source/geometry/solids/CSG/include/G4Tubs.hh

- use the constructor:

G4Tubs(const G4String& pName, G4double pRmin, G4double pRmax,
G4double pDz, G4double pSPhi, G4double pDPhi)



where:
pRmin: Inner Radius
pRmax: Outer Radius
pDZ: half length in Z
pSPhi: starting phi angle
pDPhi: angular span of the section

G4Tubs* a_tube=new G4Tubs(“a Tube”,10*cm,30*cm,20*cm,0.,270.*deg);

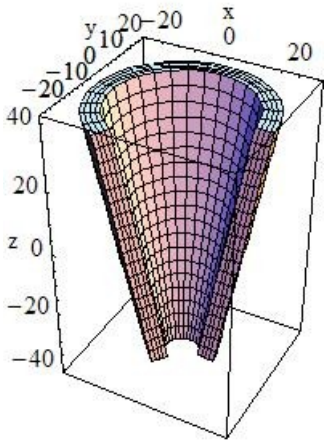
G4Cons

- The definition of a cone (or a section of it) is in:

  $G4INSTALL/source/geometry/solids/CSG/include/G4Cons.hh

- Use the constructor:

  G4Cons(const G4String& pName, G4double pRmin1, G4double pRmax1,
  G4double pRmin2, G4double pRmax2,
  G4double pDz,
  G4double pSPhi, G4double pDPhi)



where:

pRmin1,pRmax1	:	Inner/Outer Radius at z=-pDz
pRmin2,pRmax2	:	Inner/Outer Radius at z=pDz
pDZ:		half length in Z
pSPhi:		starting phi angle
pDPhi:		angular span of the section

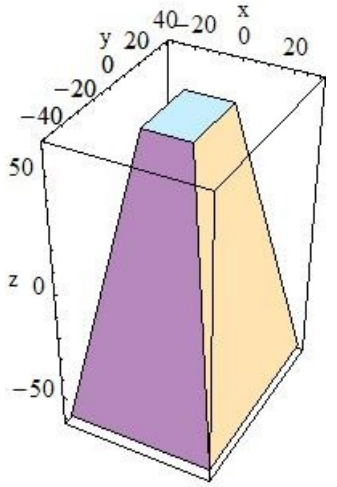
G4Cons* a_cone=new G4Cons(“a Cone”,1*cm,5*cm,10*cm,30*cm,20*cm,0.,180.*deg);

- The definition of a trapezoid is in

  $G4INSTALL/source/geometry/solids/CSG/include/G4Trd.hh

- The general constructor is:

```
G4Trd(const G4String& pName, G4double dx1, G4double dx2,
                             G4double dy1, G4double dy2,
                             G4double pDz)
```



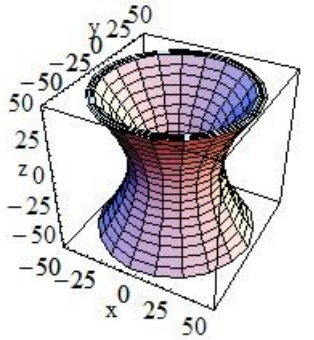
where:

dx1:	Half length along x at the surface positioned at -dz
dx2:	Half length along x at the surface positioned at +dz
dy1:	Half length along y at the surface positioned at -dz
dy2:	Half length along y at the surface positioned at +dz
dz:	Half length along z axis

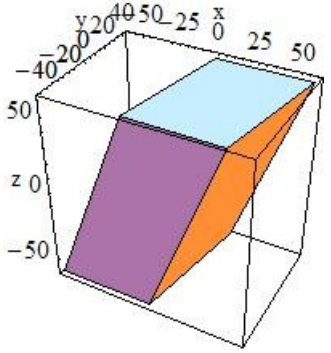
```
G4Trd* a_trd=new G4Trd("a Trd",10*cm,20*cm,1*m,1*m,2*m);
```

Other CSG shapes

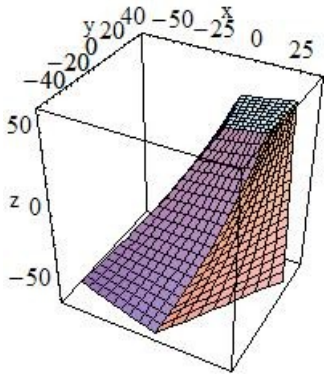
- G4Hype
  - an hyperbolic Tubs with curved sides parallel to the z axis



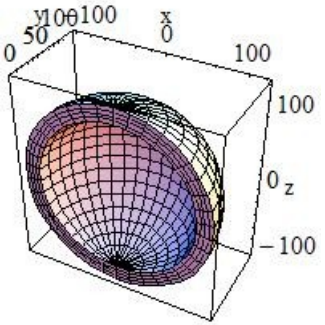
- G4Para
  - a general Parallelepiped



- G4Trap
  - a general trapezoid (possibly twisted)

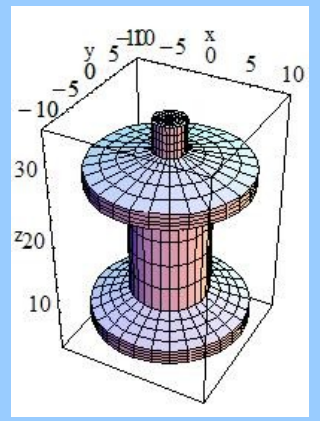


- G4Sphere
  - a (section of a) sphere



Polycone and polyhedra

- A polycone solid is a shape defined by a set of inner and outer conical or cylindrical surface sections and two planes perpendicular to the Z axis. Each conical surface is defined by its radius at two different planes perpendicular to the Z axis. Inner and outer conical surfaces are defined using common Z planes

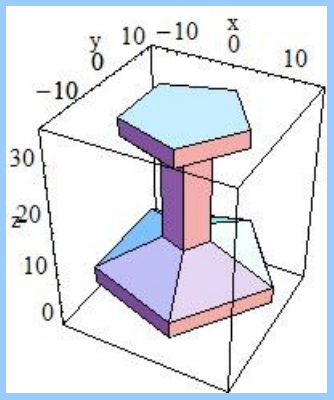



```
G4Polycone( G4String name,
            const G4double start_angle,          // starting angle
            const G4double opening_angle,        // opening angle
            const int num_z_planes,              // nr. of planes
            const G4double z_start,              // starting value of z
            const G4double z_values[],           // z coordinate of each plane
            const G4double RMIN[],               // inner radius of the cone at each plane
            const G4double RMAX[])               // outer radius of the cone at each plane
```

Polycone and polyhedra

- The polyhedra solid is a shape defined by an inner and outer polygonal surface and two planes perpendicular to the Z axis. Each polygonal surface is created by linking a series of polygons created at different planes perpendicular to the Z axis. All these polygons have the same nr of sides

```
G4Polyhedra( G4String name,
             const G4double start_angle,          // starting angle
             const G4double opening_angle,        // opening angle
             const G4int n_sides                  // nr. of sides
             const G4int num_z_planes,            // nr. of planes
             const G4double z_start,              // starting value of z
             const G4double z_values[],           // z coordinate of each plane
             const G4double RMIN[],               // inner radius of the cone at each plane
             const G4double RMAX[])               // outer radius of the cone at each plane
```



Logical volumes

- To create a Logical volume one must start from a solid and a material.

```
#include “G4LogicalVolume.hh”
#include “G4Box.hh”
#include “G4Material.hh”
….
G4Box* a_box =  new G4Box(“A box”,dx,dy,dz);
G4double a=39.95*g/mole;
G4double density=1.390*g/cm3;
G4Material* LAr = new G4Material(name=“Liquid Argon”,z=18.,a,density);
G4LogicalVolume* a_box_log = new G4LogicalVolume(a_box,LAr,”a simple box”);
```

Positioning a volume

- To position a volume, one must start with a logical volume and decide what volume (which must already exist) to place it inside, where to position it (wrt the mother's reference system) and how to rotate it
- A physical volume is simply a positioned instance of a logical volume

```
#include "G4VPhysicalVolume.hh"
#include "G4PVPlacement.hh"
…
G4RotationMatrix *rm=new G4RotationMatrix;
rm->rotateX(30*deg);
G4Double aboxPosX=-1.0*m;
G4VPhysicalVolume* a_box_phys=
                    new G4PVPlacement(rm,                          // pointer to G4RotMatrix!!
                                      G4ThreeVector(aboxPosX,0,0),  // position
                                      a_box_log,                    // its logical volume
                                      "a box",                      // its name
                                      experimentalHall_log,         // its mother
                                      false,                        // no boolean ops.
                                      1);                           // the copy nr.
```

Positioning a volume (2)

- An exception exist to the rule that a physical volume must be placed inside a mother volume. The World volume must be created as a G4PVPlacement with a null mother pointer and be positioned unrotated at the origin of the global coordinate system

```
#include “G4PVPlacement.hh”
…
G4VPhysicalVolume experimentalHall_phys
                        =new G4PVPlacement(0,                       // No rotation!
                                           G4ThreeVector(0.,0.,0.), // No Translation!
                                           experimentalHall_log,    // The logical volume
                                           “Experimental Hall”,     // its name
                                           0,                       // No mother volume!
                                           false,                   // no boolean operations
                                           0);                      // copy number
```

Physical Volume types

- Physical Volumes represent the spatial positioning of the volumes describing the detector elements
- Several different techniques can be used:
  - Placement
    - involves the definition of the transformation for the volume to be positioned
  - Repeated positioning
    - defined by using the nr. of times a volume should be replicated at a given distance along a given direction
  - Parameterised volumes
    - defined by describing a parameterised formula to specify the characteristics and positions of multiple copies of a volume

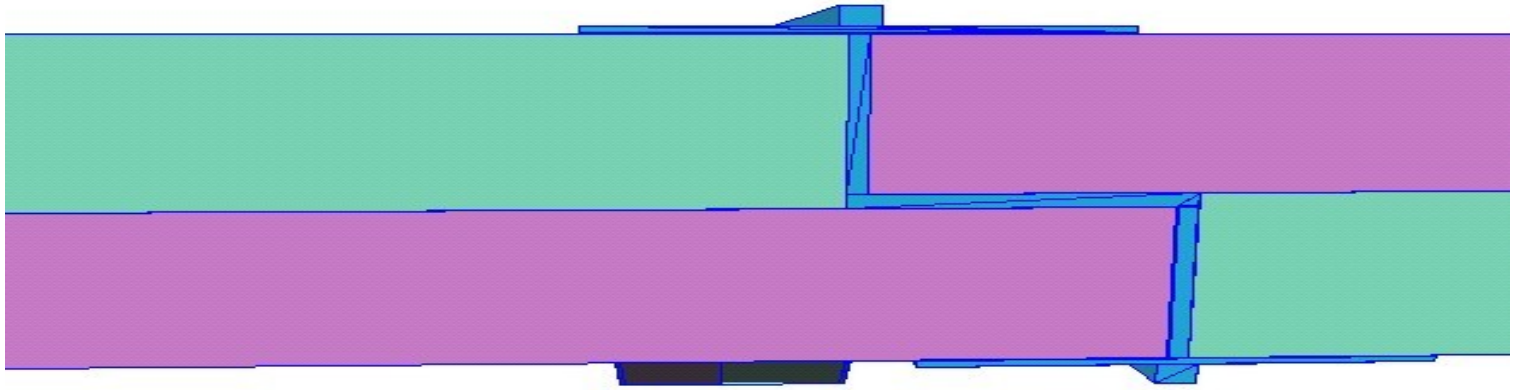
Parameterised Volumes

- Parameterised Volumes are Repeated Volumes in the case in which the multiple copies of a volume can be different in size, solid type or material
- The solid’s type, its dimensions, the material and the transformation matrix can all be parameterised in function of the copy number, either when a strong symmetry exists and when it does not
- The user implements the desired parameterisation function and the program computes and updates automatically at run time the information associated to the Physical Volume

Solids from boolean operations

- A very interesting feature of Geant4 is the possibility of combining simple solids with boolean operations





Boolean solids

- To create such a “new” boolean solid, one needs:
  - Two solids
  - a boolean operation (union, intersection, subtraction)
  - (optionally) a transformation for the second solid
- The solids used should be either CSG solids or another boolean solid (result of a previous boolean operation)

```
#include “G4UnionSolid.hh”
#include “G4SubtractionSolid.hh”
#include “G4IntersectionSolid.hh”
….
G4Box *box1= new G4Box(“Box #1”,20,30,40);
G4Tubs *cylinder1= new G4Tubs(“Cylinder #1”,0,50,50,0,2*M_PI);

G4VSolid *aUnion=new G4UnionSolid(“Box+Cylinder”,box1,cylinder1);
G4VSolid *anIntersection=
            new G4IntersectionSolid(“Box intersect Cylinder”,box1,cylinder1);
G4VSolid *aSubtraction=
            newG4SubtractionSolid(“Box-Cylinder”,box1,cylinder1);
```

Touchables

- Touchable Volumes provide a way of uniquely identifying a detector element in the geometrical hierarchy
  - for navigating in the geometry tree and storing hits
  - for providing a geometry description alternative to the one used by the Geant4 tracking system
  - for parameterised geometries
- A touchable is a geometrical volume (solid) which has a unique placement in a detector description
- In Geant4, touchables are implemented as solids associated to a transformation matrix in the global reference system or as a hierarchy of physical volumes

G4VTouchable

- All G4VTouchable implementations must respond to at least two requests:
  - GetTranslation() and GetRotation() that return the components and the volume's transformation
  - GetSolid() that gives the solid associated to the touchable
- Additional requests can be satisfied by:
  - Touchables keeping a physical volume
    - GetVolume() which returns a physical volume
    - GetReplicaNumber() which returns the copy nr. (if set)
  - Touchables that store a complete stack of parent volumes
    - GetHistoryDepth() to retrieve how many levels are there in the tree
    - MoveUpHistory(n) allows to navigate in the tree

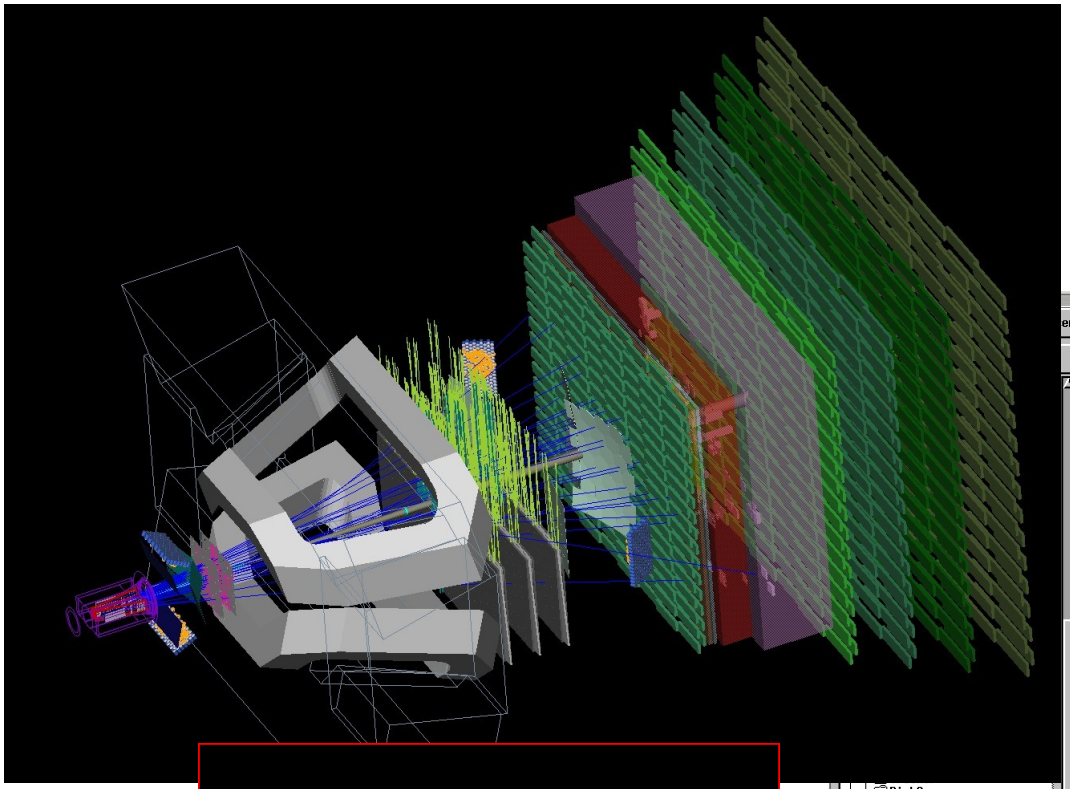
Touchable History

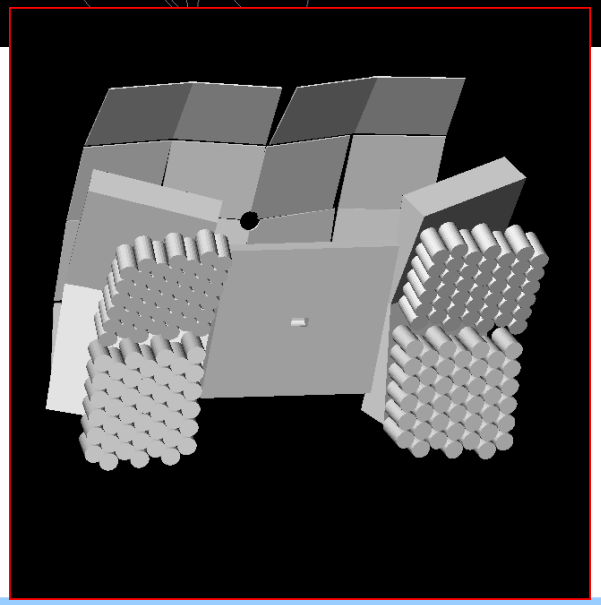
- A logical volume represents unpositioned detector elements, even a physical volume can represent multiple detector elements (replicas)
- Touchables provide a unique identification for a detector element
  - Transportation and tracking exploit the implementation of touchables implemented via the TouchableHistory
- TouchableHistory is the minimal set of information required to build the full genealogy on top of a given physical volume up to the root of the geometrical tree
- Touchables are made available to the user at every step during tracking by means of a G4Step object

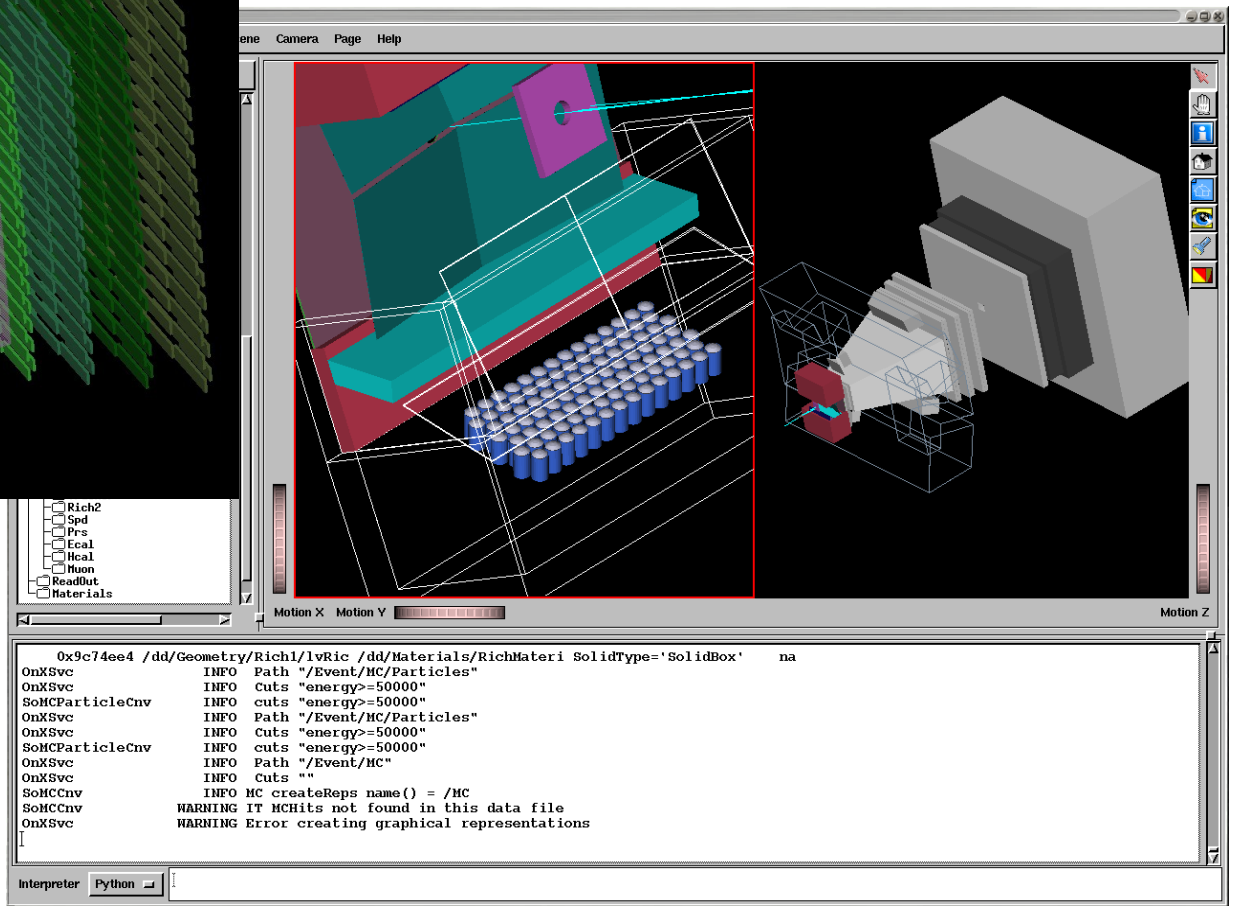
Touchable History (2)

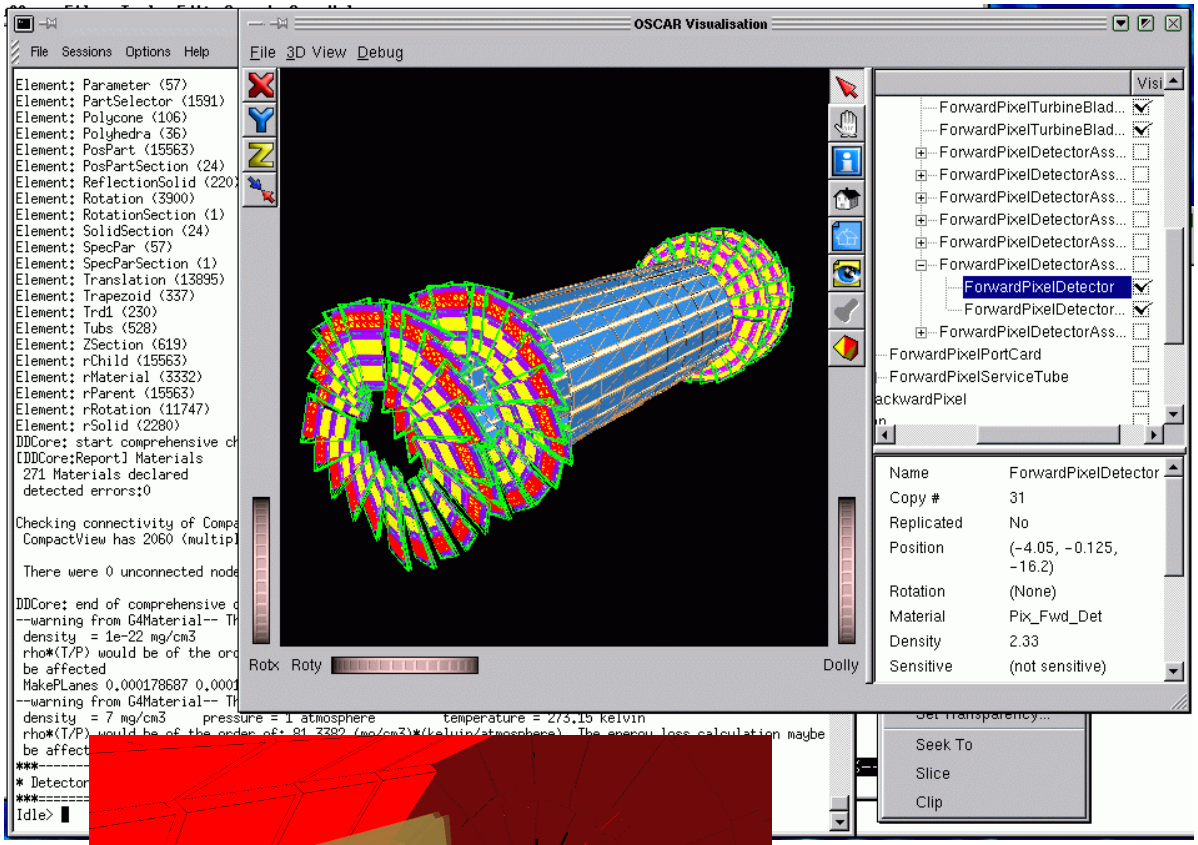
- To create a G4TouchableHistory you must ask the navigator
  - G4TouchableHistory* CreateTouchableHistory() const;
- to navigate in the geometrical tree you can use
  - G4int GetHistoryDepth() const;
  - G4int MoveUpHistory(G4int num_levels=1);
- The first method tells you how many level deep in the current tree the volume is, the second asks the touchable to eliminate its deepest level
- The MoveUpHistory significantly modifies the state of the touchable

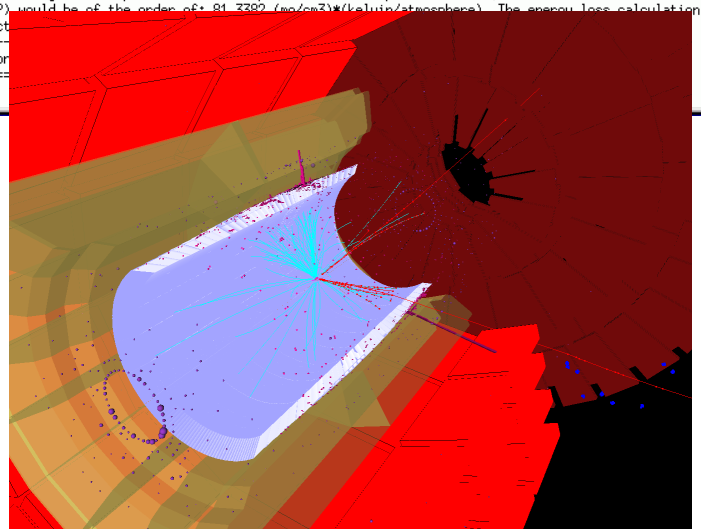
Gallery of applications

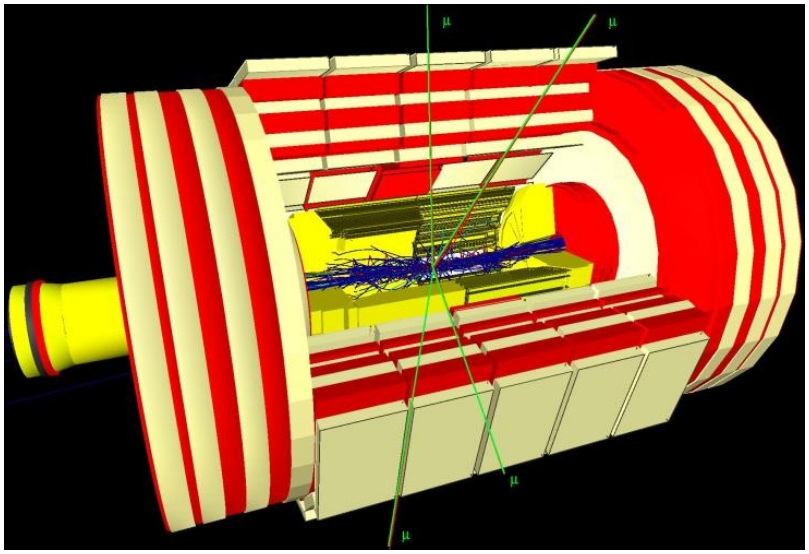


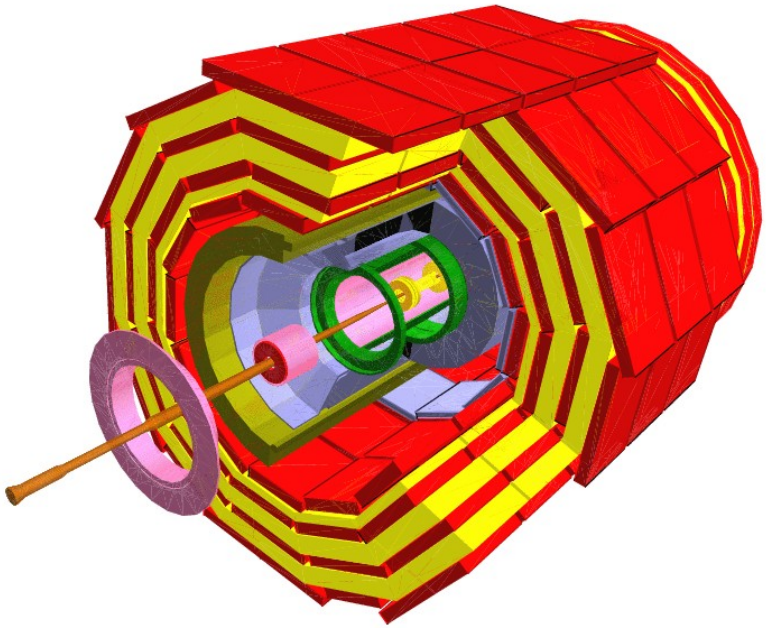



ene Camera Page Help
Rich2
Spd
Prs
Ecal
Hcal
Muon
ReadOut
Materials
Motion X Motion Y
Motion Z
0x9c74ee4 /dd/Geometry/Rich1/lvRic /dd/Materials/RichMateri SolidType='SolidBox' na
OnXSvc INFO Path "/Event/MC/Particles"
OnXSvc INFO Cuts "energy>=50000"
SoMCParticleCnv INFO cuts "energy>=50000"
OnXSvc INFO Path "/Event/MC/Particles"
OnXSvc INFO Cuts "energy>=50000"
SoMCParticleCnv INFO cuts "energy>=50000"
OnXSvc INFO Path "/Event/MC"
OnXSvc INFO Cuts ""
SoMCCnv INFO MC createReps name() = /MC
SoMCCnv WARNING IT MCHits not found in this data file
OnXSvc WARNING Error creating graphical representations
Interpreter
Python

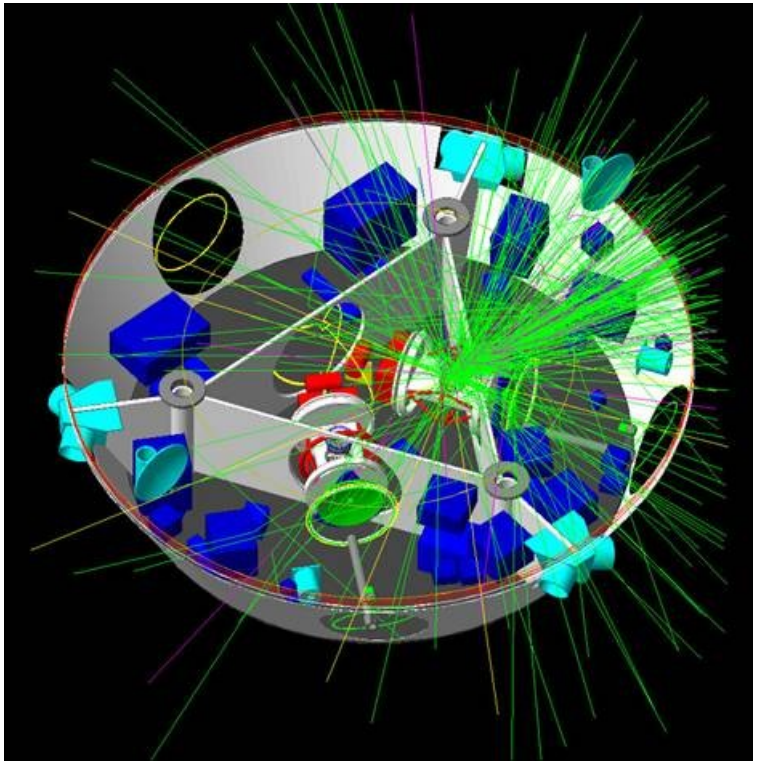

File Sessions Options Help
Element: Parameter (57)
Element: PartSelector (1591)
Element: Polycone (106)
Element: Polyhedra (36)
Element: PosPart (15563)
Element: PosPartSection (24)
Element: ReflectionSolid (220)
Element: Rotation (3900)
Element: RotationSection (1)
Element: SolidSection (24)
Element: SpecPar (57)
Element: SpecParSection (1)
Element: Translation (13895)
Element: Trapezoid (337)
Element: Trd1 (230)
Element: Tubs (528)
Element: ZSection (619)
Element: rChild (15563)
Element: rMaterial (3332)
Element: rParent (15563)
Element: rRotation (11747)
Element: rSolid (2280)
DDCore: start comprehensive ch
[DDCore:Report] Materials
271 Materials declared
detected errors:0
Checking connectivity of Compa
CompactView has 2060 (multipl
There were 0 unconnected node
DDCore: end of comprehensive c
--warning from G4Material-- Th
density = 1e-22 mg/cm3
rho*(T/P) would be of the ord
be affected
MakePLanes 0.000178687 0.0001
--warning from G4Material-- Th
density = 7 mg/cm3 pressure = 1 atmosphere temperature = 273.15 kelvin
be affect
***
* Detector
***
Idle>
OSCAR Visualisation
File 3D View Debug
Rotx Roty
Dolly
Visi
ForwardPixelTurbineBlad...
ForwardPixelTurbineBlad...
ForwardPixelDetectorAss...
ForwardPixelDetectorAss...
ForwardPixelDetectorAss...
ForwardPixelDetectorAss...
ForwardPixelDetectorAss...
ForwardPixelDetectorAss...
ForwardPixelDetector
ForwardPixelDetector...
ForwardPixelDetectorAss...
ForwardPixelPortCard
ForwardPixelServiceTube
ackwardPixel
Name ForwardPixelDetector
Copy # 31
Replicated No
Position (-4.05, -0.125, -16.2)
Rotation (None)
Material Pix_Fwd_Det
Density 2.33
Sensitive (not sensitive)
Seek To
Slice
Clip



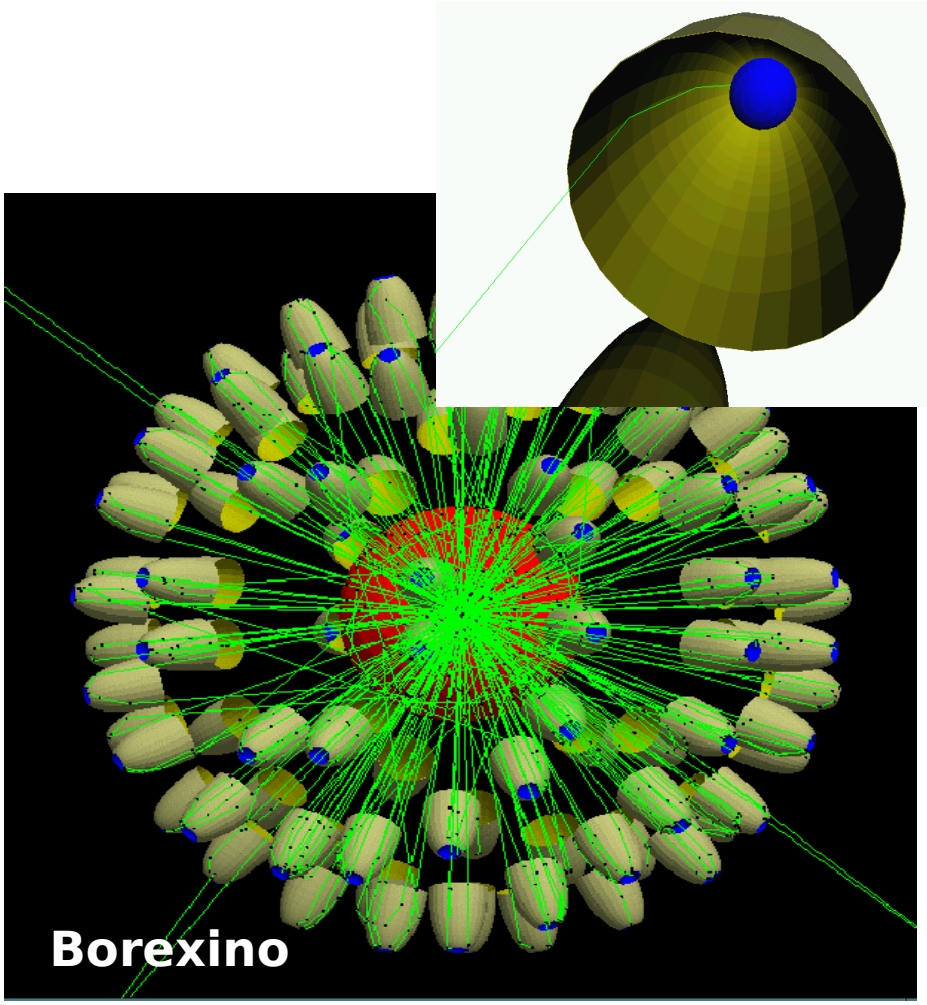




Non HEP applications



High-energy proton interaction in LISA spacecraft



Visualization

Geant4 and visualization

- The Geant4 visualization system was developed in response to a diverse set of requirements:
  - Quick response to study geometries, trajectories and hits
  - High-quality output for publications
  - Flexible camera control to debug complex geometries
  - Tools to show volume overlap errors in detector geometries
  - Interactive picking to get more information on visualized objects
- Geant4 visualization was designed around an abstract interface that supports many graphics systems.
- An interface to one of these systems is called a driver

Visualization drivers

- If you want very responsive photorealistic graphics (and have the OpenGL libraries installed)
    - OpenGL is a good solution (if you have the Motif extensions, this also gives GUI control)
- If you want very responsive photorealistic graphics plus more interactivity (and have the OpenInventor libraries installed)
    - OpenInventor is a good solution
- If you want GUI control, want to be able to pick on items to inquire about them (identity, momentum, etc.), perhaps want to render to vector formats, and a wireframe look will do
    - HepRep/WIRED will meet your needs
- If you want to render highest quality photorealistic images for use in a poster or a technical design report, and you can live without quick rotate and zoom
    - DAWN is the way to go

Visualization drivers (2)

- If you want to render to a 3D format that others can view in a variety of commodity browsers (including some web browser plug-ins)
    - VRML is the way to go
- If you want to visualize a geometry that the other visualization drivers can't handle, or you need transparency or mirrors, and you don't need to visualize trajectories
    - RayTracer will do it
- If you just want to quickly check the geometry hierarchy, or if you want to calculate the volume or mass of any geometry hierarchy
    - ASCIITree will meet your needs
- You can also add your own visualization driver.
    - Geant4's visualization system is modular. By creating just three new classes, you can direct Geant4 information to your own visualization system.

Visualization Attributes

- Visualization attributes are a set of information associated with objects which can be visualized
- This information is just used for visualization and it is not included in the geometrical information such as shapes, position and orientation
- A set of visualization attributes is held by an object of class G4VisAttributes

Activating visualization

- Visualization drivers are part of the Geant4 distribution
  - remember to compile the ones you need!
- Adding visualization to your simulation program is as easy as adding to your main() the lines

```
#include "G4VisExecutive.hh"
 .....
 // Instantiation and initialization of the Visualization Manager

 G4VisManager* visManager = new G4VisExecutive;
 visManager -> initialize ();
```

- This will load all the drivers which were activated during the Geant4 compilation and make them available at runtime

Visibility

- Visibility is a boolean flag used to control the visibility of the objects passed to the Visualization Manager for visualization
- Visibility is set by:

  G4VisAttributes::SetVisibility(G4bool visibility);
- If visibility=false, visualization is skipped for those objects to which this set of attributes is assigned
- The following public data member is by default defined

  static const G4VisAttributes Invisible
- This can be referred at as G4VisAttributes::Invisible

  experimentalHall_log->SetVisAttributes(G4VisAttributes::Invisible);

Color

- G4VisAttributes holds its color entry as an object of the class G4Colour
- G4Colour has four fields which represent the RGBA (red,green,blue,alpha) of a color. Each component takes a value between 0 and 1. Alpha is the opacity.
- A G4Colour object is created by giving the red, blue and green components of the color

```
G4Color( G4double red=1.0,G4double green=1.0,
         G4double blue=1.0,
         G4double a=1.0);
```

- The default value of each component is 1, which means white

Color (2)

```
G4Colour white()                ;       // white
G4Colour white(1.,1.,1.)        ;       // white
G4Colour gray(.5,.5,.5)         ;       // gray
G4Colour black(0.,0.,0.)        ;       // black
G4Colour red(1.,0.,0.)          ;       // red
G4Colour green(0.,1.,0.)        ;       // green
G4Colour blue(0.,0.,1.)         ;       // blue
G4Colour cyan(0.,1.,1.)         ;       // cyan
G4Colour magenta(1.,0.,1.)      ;       // magenta
G4Colour yellow(1.,1.,0.)       ;       // yellow
```

G4VisAttributes and volumes

- In constructing the detector components, one may assign a set of visualization attributes to each Logical Volume in order to visualize it later
- G4LogicalVolume holds a pointer to a G4VisAttributes. This field is set and referenced by using:

```
void G4LogicalVolume::SetVisAttributes(const G4VisAttributes* pVa);
void G4LogicalVolume::SetVisAttributes(const G4VisAttributes& Va);
```

- If you want to display your experimental hall in cyan and in forced wireframe style you do:

```
ExpHall_log=new G4LogicalVolume(ExpHall_box,Air,"exp_hall");
.....
G4VisAttributes *expHallVisAtt=new
   G4VisAttributes(G4Colour(0.,1.,1.));
expHallVisAtt->SetForceWireframe(true);
ExpHall_log->SetVisAttributes(expHallVisAtt);
```

Physics processes

G4VUserPhysicsList

- The G4VUserPhysicsList class is the base class which one has to derive from to specify all particles and physics processes to be used in the simulation
- The user must inherit from G4VUserPhysicsList and implement three methods:
  - ConstructParticle() Particle Construction
  - ConstructProcess() Physics processes
  - SetCuts() Cut values (in stopping range)

Particles

- Geant4 provides a set of ordinary particles such as electron, proton, gamma etc.
- The G4ParticleDefinition class provides a representation of a particle. Each particle has its own class derived from it
- G4ParticleDefinition contains properties which characterize individual particles such as name, mass, charge, spin… Except for lifetime, decay table and the “stable” flag, these attributes are read-only and cannot be changed without re-building the libraries
- There are 6 major particle classes:

leptons	bosons
mesons	shortlived
baryons	ions

Particles (2)

- Each particle is represented by its own class
- Each particle class has only one static object (singleton)
  - G4Electron represents the electron
  - G4Electron::theElectron is the only object of the G4Electron class
  - A pointer to the electron object can be obtained by using the static method G4Electron::ElectronDefinition()
- The class G4ParticleTable is a collection of defined particles. It provides a series of utility methods to retrieve particle definitions:
  - FindParticle(G4String name) find the particle by name
  - FindParticle(G4int PDGencoding) find the particle by its PDG code
- G4ParticleTable is a singleton and the static method G4ParticleTable::GetParticleTable() returns its pointer

Particle construction

- The ConstructParticle() method from G4VUserPhysicsList must be implemented by the user to create all particles which are needed in the simulation

```
#include “G4Geantino.hh”
#include “G4Electron.hh”
#include “MyPhysicsList.hh”
void MyPhysicsList::ConstructParticle()
{
        G4Geantino::GeantinoDefinition();
        G4Electron::ElectronDefinition();
}
```

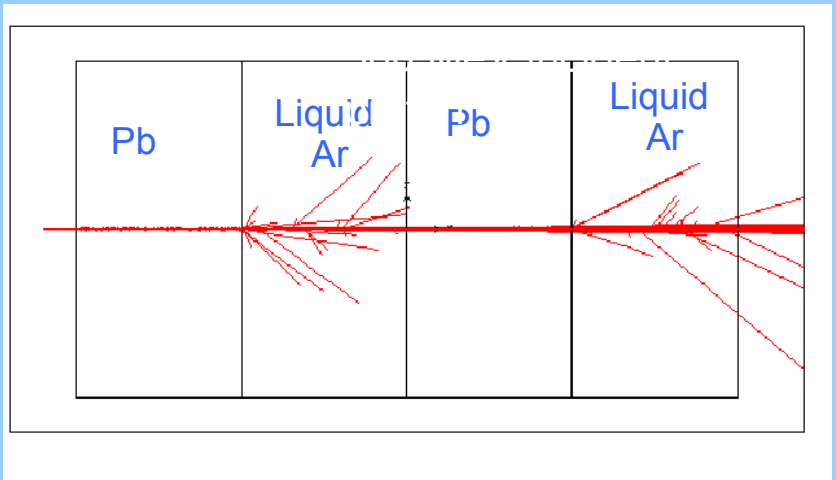
- Utility classes provided to define all particles of the same types in Geant4.

```
void MyPhysicsList::ConstructLeptons()
{
// Construct all leptons
G4LeptonConstructor pConstructor;
pConstructor.ConstructParticle();
}
```

Production threshold

- All production thresholds for secondaries are given in range, not in energy

Geant4



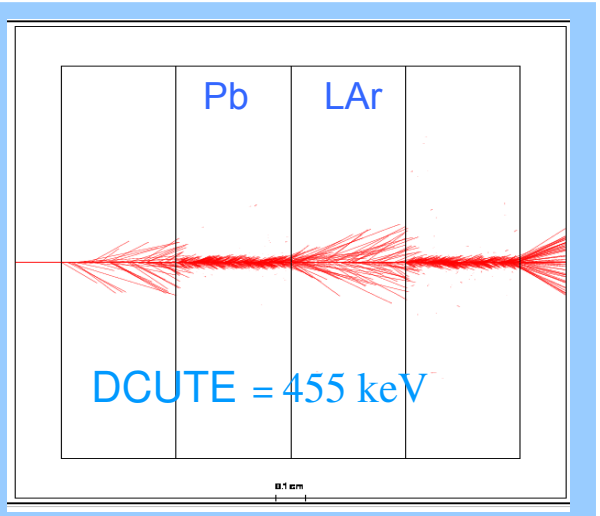
(e-) range cut: 1.5 mm

455 keV in liquid Ar

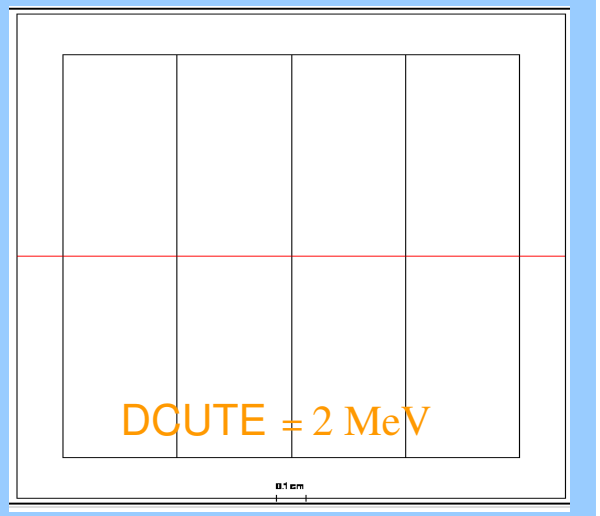
2 MeV in Pb

Geant3

Set energy cut at Ar value;



Set energy cut at Pb value, killing also secondaries in Ar...



Setting cuts

- SetCuts() is a virtual method of the class G4VUserPhisicsList to be implemented in order to set the cut in range for each particle
- Construction of geometry, particles and processes should precede the invocation of SetCuts()
- By means of SetCuts(), the cut in range is then converted to the cut-off energy for all materials defined in the geometry
- G4VUserPhysicsList keeps a defaultCutValue as the default cut off value for the range. A method SetCutsWithDefault() is invoked by the G4RunManager for all normal applications and this default value is passed to the argument of the SetCuts() method

Setting cuts (2)

- The default cut value is 0.7mm in all cases. Its value can be changed directly in the constructor of the class you inherit from G4VPhysicsList or by using the SetDefaultCutValue() method
- The cut value can be retrieved by using the method

  GetLengthCuts()
- whilst the cut-off energy for each material can be obtained by invoking

  GetEnergyThreshold(G4Material *)
- If you want to set different cut values for different particles, you must respect a specific definition order, as some particles depend on some other particle's cut value for calculating the cross-sections

  gamma →electron →positron →proton/antiproton

Setting cuts (3)

- To ease the implementation of the SetCuts() method, the G4VUserPhysicsList class provides an utility method
  - SetCutValue(G4double cut_value, G4String particle_name)
- An example of implementation

```
void MyPhysicsList::SetCuts()
{
const G4double cut=.1*mm;
SetCutValue(cut,”gamma”);
SetCutValue(cut,”e+”);
SetCutValue(cut,”e-”);
SetCutValue(cut,”proton”);
}
```

Production threshold vs tracking cut

- It is the responsibility of each individual process to produce secondary particles according to its own capabilities. On the other hand, only the Geant4 kernel can ensure an overall coherence of the whole simulation
- hence, as a general principle
  - Each process has its intrinsic limits to produce secondary particles
  - All particles produced will be tracked up to zero range
  - Each particle has a suggested cut in range (converted to an energy cut for all materials)
- The results of this is that the cut associated to each particle is a (recommended) production threshold on secondary particles

Production below threshold

- A process can still decide to produce secondary particles even below the recommended production threshold
  - if, by checking the range of the secondary produced against quantities like safety (~the distance to the next boundary), it turns out that the particle, even below threshold, might reach a sensitive part of the detector
  - when mass-to-energy conversion can occur, to conserve the energy. For instance, in gamma conversion, the positron is always produced, even at 0 energy, for further annihilation

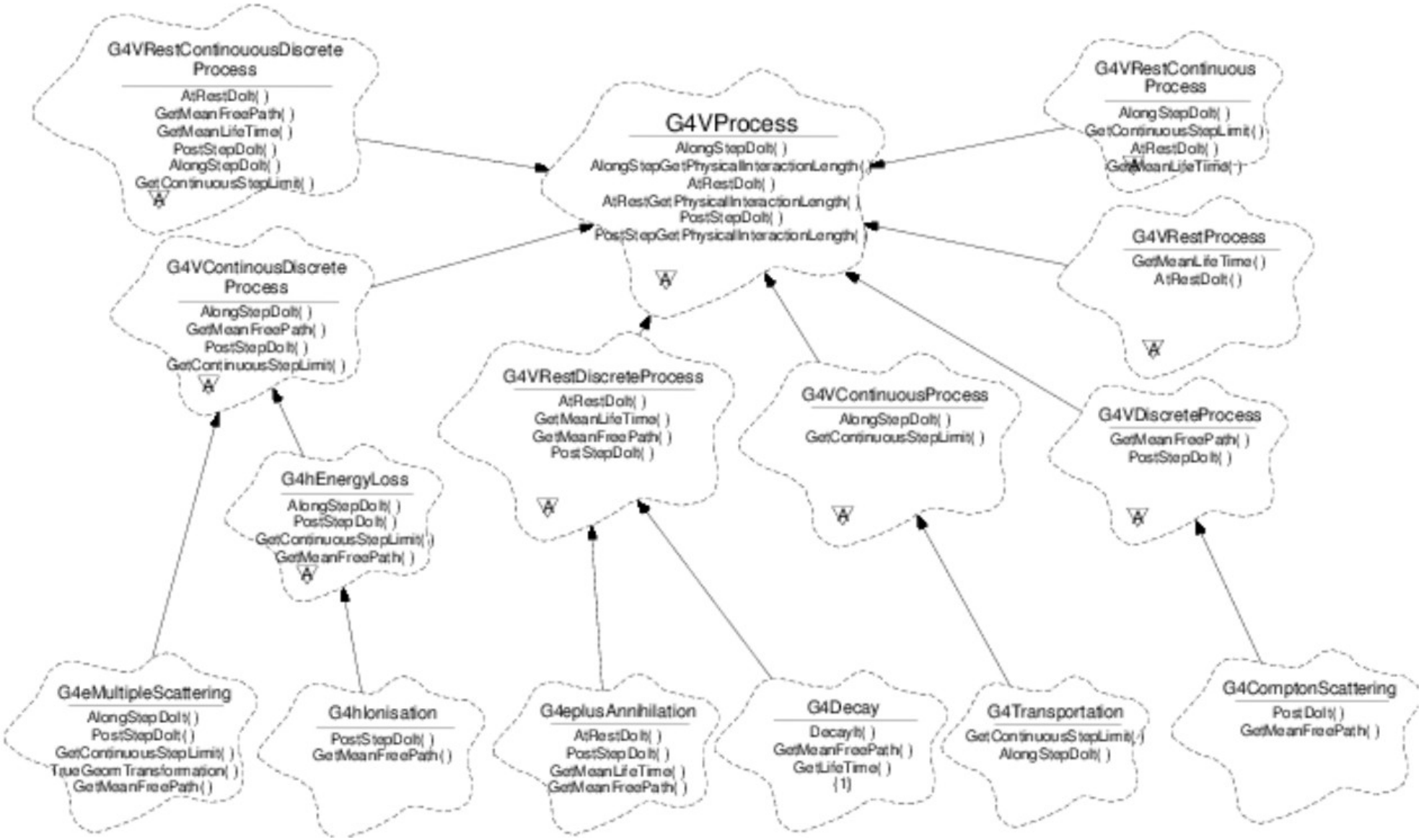
Physics Processes

- Physics processes describe the interaction of particles with matter
- In Geant4 there are 7 major categories of physics processes
  - electromagnetic
  - hadronic
  - transportation
  - decay
  - optical
  - photolepton_hadron
  - parameterisation

Physics Processes (2)

- A class G4VProcess is provided as the base class for all physics processes
- All physics processes are described by using three (virtual) methods:
  - AtRestDoIt()
  - AlongStepDoIt()
  - PostStepDoIt()
- The following classes are then used as base classes for simple processes
  - G4VAtRestProcess — Process with AtRestDoIt() only
  - G4VContinuousProcess — Process with AlongStepDoIt only
  - G4VDiscreteProcess — Process with PostStepDoIt
- 4 additional classes (such as G4VContinuousDiscreteProcess) are provided for complex processes

Physics processes



G4ProcessManager

- G4ProcessManager is a member of G4ParticleDefinition
- G4ProcessManager keeps a list of processes a particle can undergo. Moreover, it keeps the information on the order of invocation of the processes as well as which kind of DoIt method is valid for which process
- Physics processes should be registered with the G4ProcessManager of each particle (together with ordering information) by using AddProcess() and SetProcessOrdering(). For registering simple processes, one can use AddAtRestProcess(), AddContinuousProcess() and AddDiscreteProcess()
- The G4ProcessManager has the possibility of switching on/off some processes at run time by using ActivateProcess() and InActivateProcess()

AddTransportation()

- Transportation (propagating particles in space) is treated as a process, hence it must be defined for all particles
- The G4Transportation class must be registered with all particle classes. An AddTransportation() method is provided in G4VUserPhysicsList and it must be called in ConstructPhysics()

```
void G4VUserPhysicsList::AddTransportation() {
  G4Transportation* theTransportation=new G4Transportation;
  theParticleIterator->reset();
  while ( (*theParticleIterator) () ) {
    G4ParticleDefinition *particle=theParticleIterator->value();
    G4ProcessManager* pmanager=particle->GetProcessManager();
      if (!particle->IsShortLived()) {
        pmanager->AddProcess(theTransportationProcess);
        pmanager->SetProcessOrderingToFirst(theTransportationProcess,idxAlongStep);
      }
   }
}
```

ConstructProcess()

- ConstructProcess() is a pure virtual method which is used to create physics processes and register them to particles
- For each particle which has been declared in ConstructParticle(), the user must get hold of the G4ProcessManager attached to the particle and register all relevant physics processes by using the AddProcess(G4VProcess *) method

ConstructProcess for gammas

```
Void MyPhysicsList::ConstructProcess()

{
AddTransportation();
ConstructEM();
}
void MyPhysicsList::ConstructEM()
{
G4ParticleDefinition* particle = G4Gamma::GammaDefinition();
G4ProcessManager *pmanager= particle->GetProcessManager();
G4PhotoElectricEffect *thePhotoElectricEffect = new
   G4PhotoElectricEffect;
G4ComptonScattering *theComptonScattering = new
   G4ComptonScattering;
G4GammaConversion *theGammaConversion = new
   G4GammaConversion;
pmanager->AddDiscreteProcess(thePhotoElectricEffect);
pmanager->AddDiscreteProcess(theComptonScattering);
pmanager->AddDiscreteProcess(theGammaConversion);
}
```

G4UserLimits

- G4UserLimits is a placeholder for user-defined step limitations
- If associated with a LV, an object of type G4UserLimits can be used effectively to influence the tracking behavior in that particular volume
- A G4UserLimits object can contain:
  - the maximum allowed size for a step
  - the maximum allowed length of a track
  - the maximum allowed TOF for a track
  - the minimum kinetic energy of a track (only for charged particles)
  - the minimum range for a track
- Set/Get methods permit to have access to this information
- In addition to the user limits themselves, special processes must be added:
  - G4StepLimiter
  - G4UserSpecialCuts

Simulation input (aka Event Generation)

G4VUserPrimaryGeneratorAction

- G4VUserPrimaryGeneratorAction is one of the mandatory classes the user has to derive a concrete class from
- In the concrete class, one must specify how the primary event should be generated
- G4VUserPrimaryGeneratorAction has a pure virtual method named GeneratePrimaries(), invoked at the beginning of each event. This method receives (as an argument) a pointer to the current G4Event which must then be filled with particles that will then be tracked trough the detector

  GeneratePrimaries(G4Event* anEvent);

The primary event

- The primary event (kinematics) is made of G4PrimaryParticles

  G4PrimaryParticle(G4int PDGcode,G4double Px,G4double Py,G4double Pz)

  G4PrimaryParticle(G4ParticleDefinition *pDef, G4double Px,G4double Py, G4double Pz)

- Primary particles are associated to their production vertex (of type G4PrimaryVertex)

  G4PrimaryVertex(G4double Vx,G4double Vy,G4double vz,G4double T0)

  G4PrimaryVertex(G4ThreeVector Vpos,G4double T0)

- Primary particles are associated to the primary vertex by means of the primary vertex' method AddPrimary

  AddPrimary(G4PrimaryParticle* aParticle)

- Vertices are given to the event by means of AddPrimaryVertex

  AddPrimaryVertex(G4PrimaryVertex *aVertex)

G4VPrimaryGenerator

- The GeneratePrimaries() in G4VPrimaryGeneratorAction is invoked at the beginning of each event to fill up a G4Event with particles to be tracked through the simulated detector
- To provide the possibility of having different ways of generating the kinematics of an event available in the same simulation program, Geant4 provides an abstract generator class (G4VPrimaryGenerator) which users can inherit from to implement their preferred way of generating an event.
- The (purely abstract) method that users must implement and where the actual generation of an event will take place is
  - GeneratePrimaryEvent(G4Event* an Event)
- The PrimaryGeneratorAction can then be used for switching between various PrimaryGenerator's

Generation of an event

- The primary generator should be selected in the constructor of the concrete class derived from G4VUserPrimaryGeneratorAction
- This particle generator must then be destroyed in the destructor
- G4VUserPrimaryGeneratorAction has a pure virtual method named GeneratePrimaries(), invoked at the beginning of each event. In this method, one must invoke the GeneratePrimaryVertex() method of the G4VPrimaryGenerator concrete class that one instantiated

G4ParticleGun

- G4ParticleGun is a primary particle generator provided by Geant4. It generates primary particles with a given momentum and position
- Randomization of energy, momentum and position is not provided directly by G4ParticleGun but it can be realized by invoking several of its methods; this must be done in GeneratePrimaries() before invoking generatePrimaryVertex()
- The following methods are provided for G4ParticleGun

  void SetParticleDefinition(G4ParticleDefinition*)
  void SetParticleMomentum(G4ParticleMomentum)
  void SetParticleMomentumDirection(G4ThreeVector)
  void SetParticleEnergy(G4double)
  void SetParticleTime(G4double)
  void SetParticlePosition(G4ThreeVector)
  void SetParticlePolarization(G4ThreeVector)
  void SetNumberOfParticles(G4int)

Simulation output aka Hits

Hit

- A Hit is a snapshot of a physical interaction of a track in a sensitive region of the detector
- One can store various informations associated with a G4Step object, like:
  - Position and time of the step
  - momentum of the track
  - energy deposition of the step
  - geometrical information

G4VHit

- G4VHit is an abstract class which represents a hit
- The User has to inherit from this base class and derive his/her own concrete class
- The data members in particular (I.e. the information that the hit will carry along) must be chosen by the user
- G4VHit has two virtual methods, Draw() and Print() which must be implemented to have the hits drawn and printed
- Hits are associated to the current event by means of a concrete class derived from G4VHitsCollection, which represents a vector of user-defined hits
- G4THitsCollection is a template class derived from G4VHitsCollection which can contain a particular concrete class derived from G4VHit

G4VSensitiveDetector

- G4VSensitiveDetector is an abstract base class which represents a detector
- The principal mandate of a sensitive detector is the construction of one or more hit objects using the information given in the G4Step object and an optional G4TouchableHistory class object for the ReadOut geometry: these object are the arguments of the method ProcessHits()
- The concrete sensitive detector class should be instantiated with the unique name of a detector. The name can be associated with one or more global names for categorizing the detectors

  myEMcal = new myEMcal("/myDet/myCal/myEMcal");
- The pointer to the sensitive detector must be set to one or more G4LogicalVolume object and registered with G4SDManager

  scin_log->SetSensitiveDetector(myEMcal);

G4VSensitiveDetector (2)

- G4VSensitiveDetector has three major virtual methods:
  - Initialize(): This method is invoked at the beginning of each event. The argument of this method is an object of G4HCofThisEvent class. Hits collections, where hits produced in this particular event are stored, can be associated to G4HCofThisEvent
  - ProcessHits(): This method is invoked by G4SteppingManager when a step takes place in the G4LogicalVolume which point to this sensitive detector. The first argument of this method is a G4Step object for the current step. The second argument is a G4TouchableHistory object for the ReadOut geometry, if described. In this method, one or more G4VHit objects should be constructed if the current step has to be registered
  - EndOfEvent(): This method is invoked at the end of each event, again to associate hits collections to G4HCofThisEvent

G4SDManager

- A Sensitive Detector Manager (of type G4SDManager) oversees to all operations by Sensitive Detectors.
- The SDM is a singleton (only one object at any moment) and can be accessed by using its static method

  G4SDManager::GetSDMpointer()
- All Sensitive Detectors must be registered with the SDM in order to function properly (tip: do it in the SD's constructor):

  G4SDManager *sdman=G4SDManager::GetSDMpointer()

  sdman->AddSensitiveDetector (this);
- The SDM can return a hit collection ID (useful for fishing out your collection from the hits collections of the event
  - sdman->GetCollectionID("My Collection");

Hits Collections

- Hits collections can be created by booking a unique name in the SD constructor:

```
collectionName.insert("CalorimeterCollection");
```

- …and by creating the collection in the Initialize method of the SD

```
caloHitsCollection=new CalorimeterHitsCollection
                (SensitiveDetectorName,collectionName[0]);
```

- Newly created hits can be inserted in the collection then:

```
int icell=caloHitsCollection->insert(caloHit);
```

- The collection must be inserted in the hits collections of the event (either in SD's Initialize() or SD's EnfOfEvent()

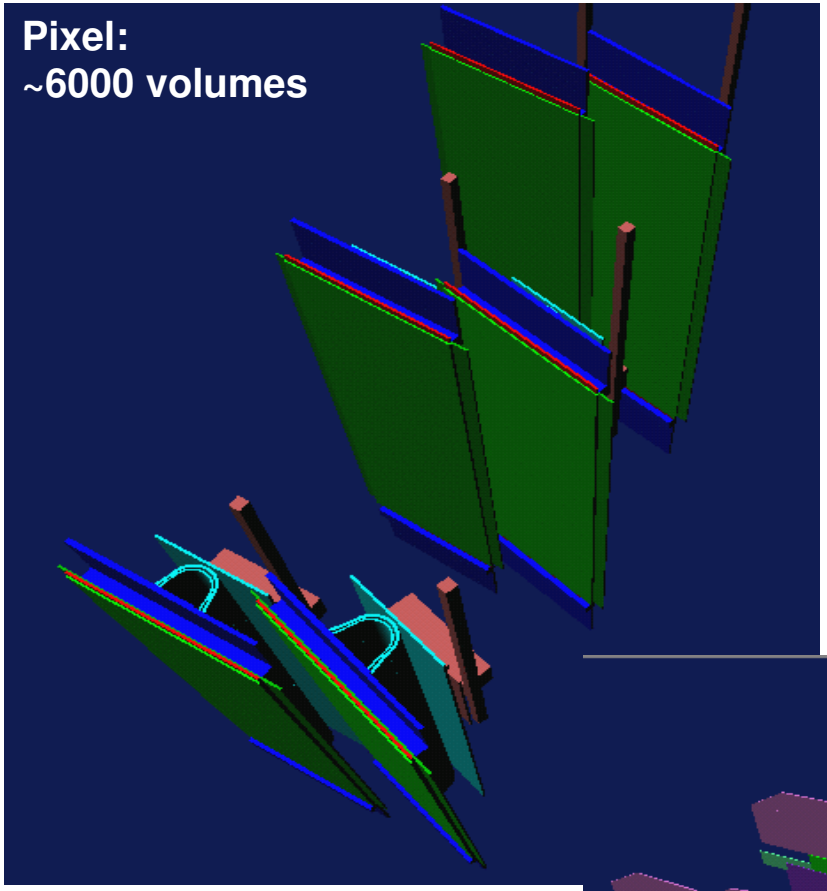
```
static G4int HCID=-1;
if (HCID<0) HCID=GetCollectionID(0);
HCE->AddHitsCollection(HCID,caloHitsCollection);
```

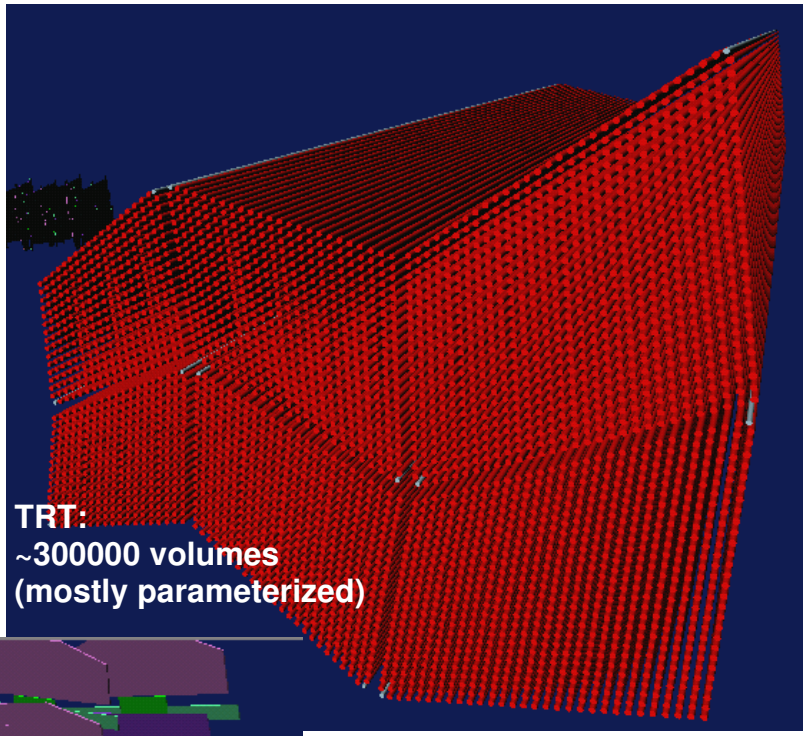
Access to the hits collections

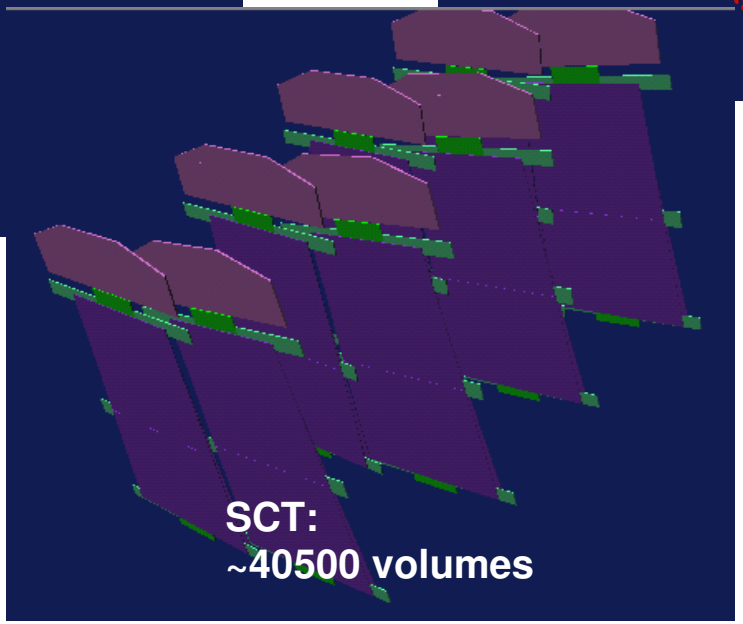
- Hits collections are accessed for various purposes
  - Digitization
  - Event Filtering in G4VUserStackingAction
  - “End of Event” simple analysis
  - Drawing/printing hits
- To access the hits collections one must go through the SD Manager

```
G4SDManager* fSDM=G4SDManager::GetSDMpointer();
G4RunManager* fRM=G4RunManager::GetRunManager();
G4int collectionID=fSDM->GetCollectionID(“collection name”);
const G4Event* currentEvent=fRM->GetCurrentEvent();
G4HCofThisEvent *HCofEvent=currentEvent->GetHCofThisEvent();
MyHitsCollection *myCollection=
(MyHitsCollection *)(HCofEvent->GetHC(collectionID));
```

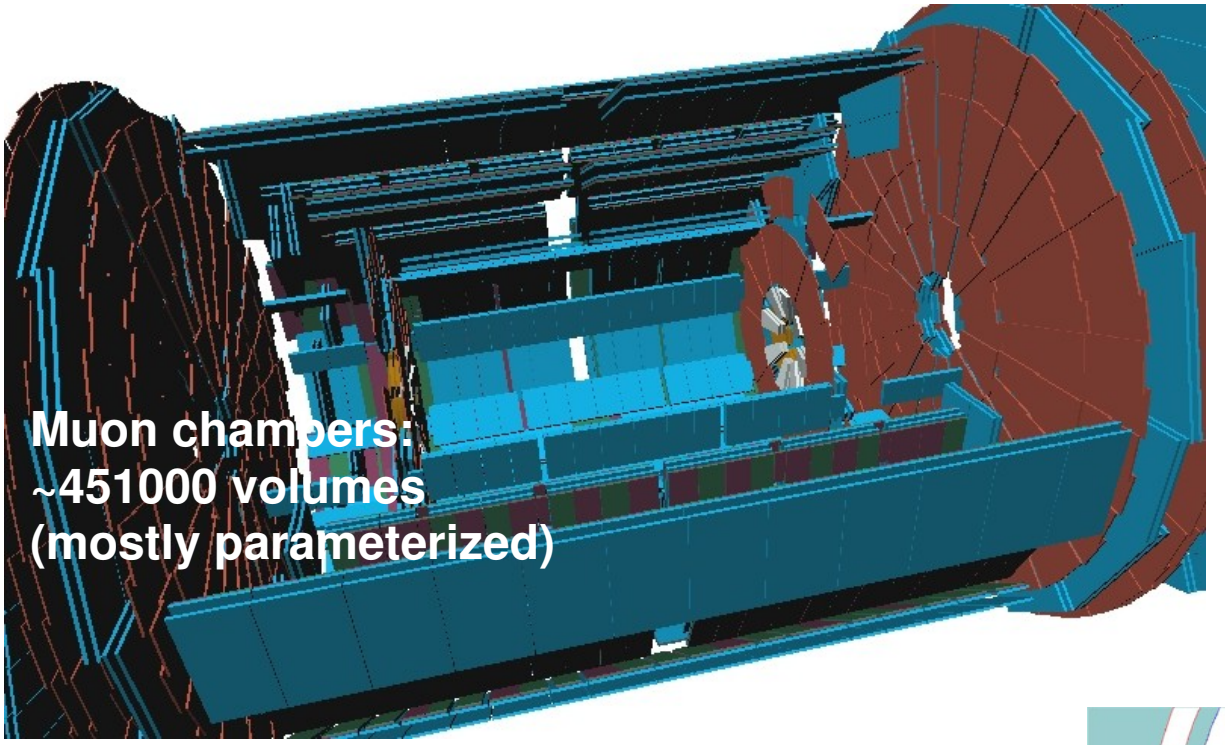
Gallery of applications

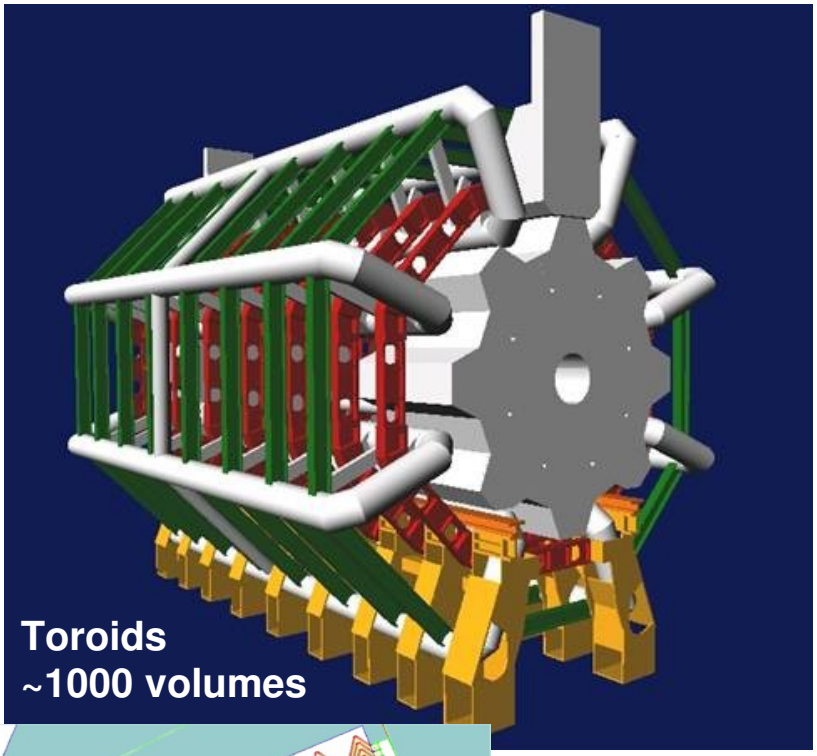

Pixel:
~6000 volumes

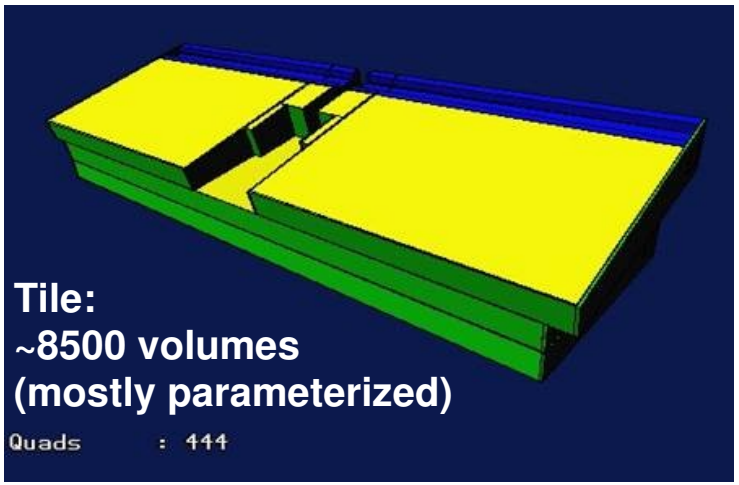

TRT:
~300000 volumes
(mostly parameterized)

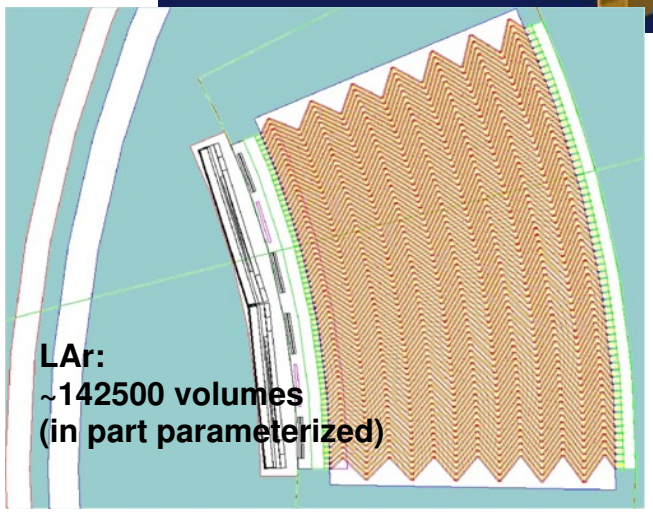

SCT:
~40500 volumes

ATLAS (2)


Muon chambers:
~451000 volumes
(mostly parameterized)

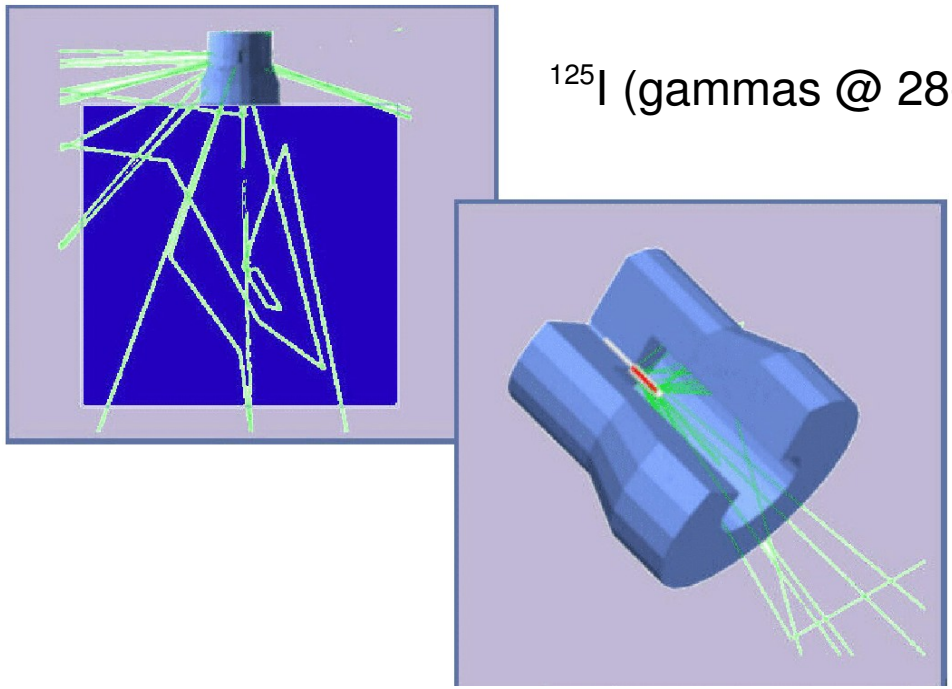

Toroids
~1000 volumes


Tile:
~8500 volumes
(mostly parameterized)
Quads : 444

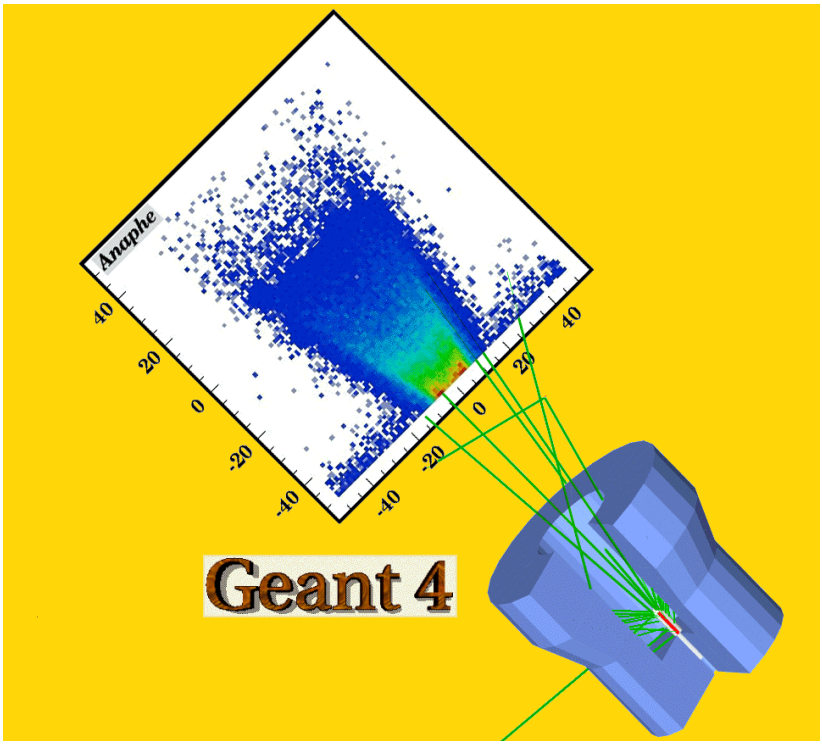

LAr:
~142500 volumes
(in part parameterized)

Non HEP applications

- Brachitherapy:
  - used in treatment of cancer
  - deposition of small radioactive sources near the tumoral mass
  - need to know precisely the dose released in parts of the body

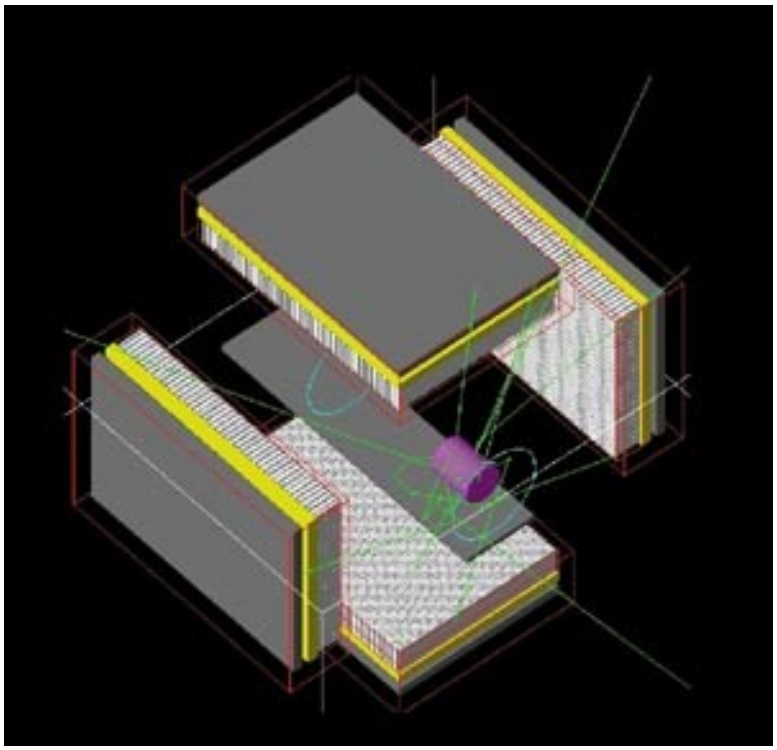


^{125}I (gammas @ 28-35 keV)

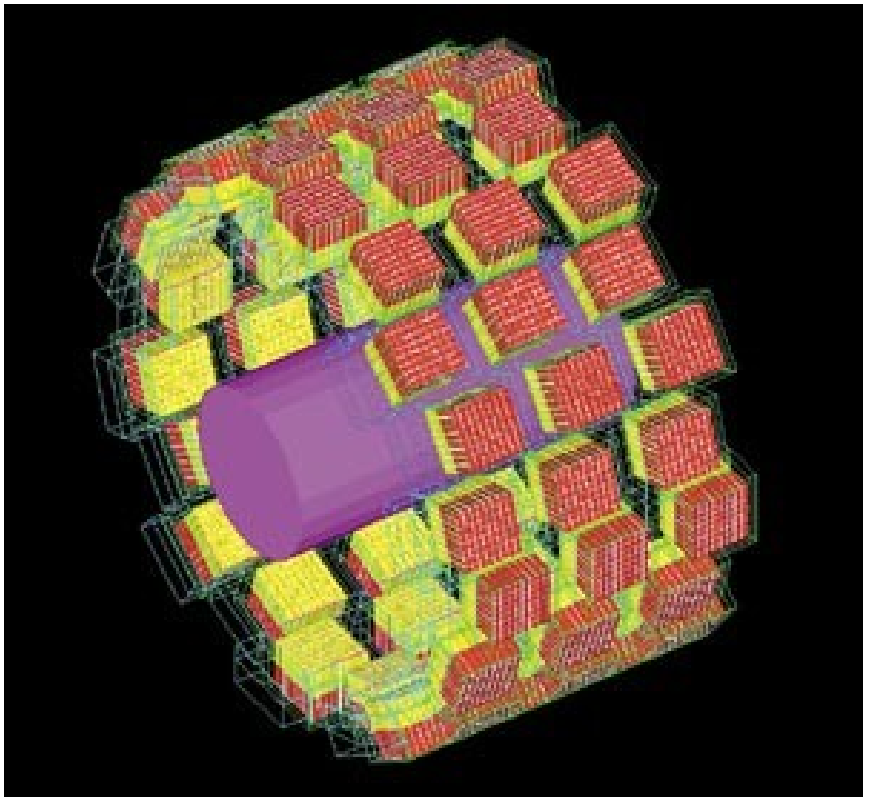


Non HEP applications

- GATE - Geant4 Application for Emission Tomography



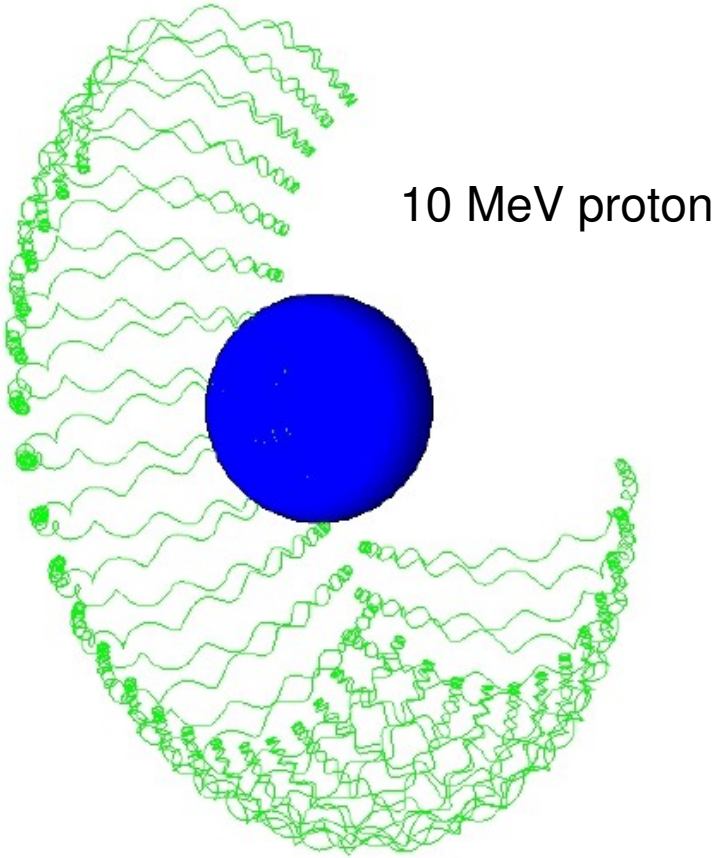
SPECT benchmark to test the installation of GATE. Image: Ghent University and U678 INSERM.

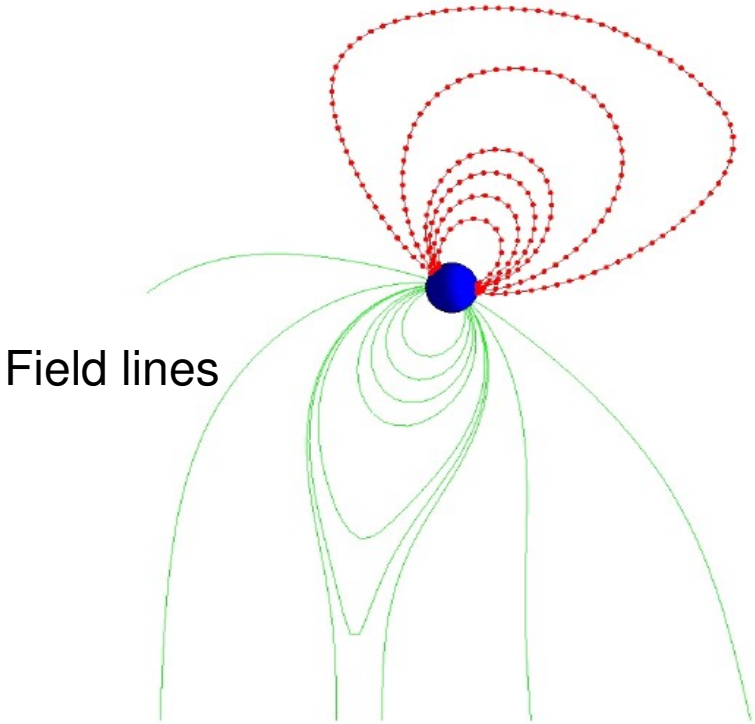


ClearPET scanner design for the Crystal Clear Collaboration, with four rings of 20 interleaved LSO/ LuYAP phoswich detector modules.

Non HEP applications

- MAGNETOCOSMICS:
  - simulates motion of cosmic particles in earth's magnetic field





Supported platforms

- List of course is release-dependent
- This is for G4.9.1
- Official platforms:
  - Linux, gcc-3.4.6. Tested on 32 bits architectures and 64 bits architectures (Intel or AMD) with the Scientific Linux CERN 4 (SLC4) distribution (based on RedHat Linux Enterprise 4). Versions of Geant4 have also been compiled successfully on other Linux distributions, like Debian, Suse or other RedHat systems.
  - MacOSX 10.4, gcc-4.0.1
  - SUN Solaris 5.8, C++ CC-5.5.
  - Windows/XP and CygWin Tools with: Visual C++ 8.0 (Visual Studio 2005)
- Additional verified configurations:
  - Linux, gcc-3.2.3, gcc-4.2.2
  - Linux, Intel-icc 9.1
  - SUN Solaris 5.8, C++ CC-5.4 Patch 111715-02
  - Windows/XP and CygWin Tools with: Visual C++ 7.1 .NET

Supported platforms

- Platforms configured but not tested and not supported:
  - AIX 4.3.2, xlC 6.0
  - DEC V4.0, cxx C++ V6.1-027
  - HP 10.20, aCC C++ B3910B A.01.23
  - SGI V6.5.5, CC 7.2.1

