

# Montecarlo Methods for Medical Physics

Francesco Longo  
([francesco.longo@ts.infn.it](mailto:francesco.longo@ts.infn.it))

# Summary of the Course

- Part1 (Apr 28)
  - General (and brief) introduction to Monte Carlo methods
  - Montecarlo methods in Medical Physics
- Part2 (May 3 – Today)
  - Introduction to the Geant4 toolkit
  - Fundamentals of a Geant4 application
    - Physics, Geometry, Particle Flux, Scoring
- Part3 (May 5 – 12)
  - Realisation of an example relevant to Medical Physics

# Evaluation for the “Laboratory”

- Discussion of Geant4 example
  - G4 example relevant to Medical Physics
- Discussion of requirements and methods
  - Medical Physics “environment”
  - Geant4 modeling
  - “Basic” analysis of results
- Realization of “new” example – Laboratory
  - Might just be an improvement of an existing G4 example

# Homework

- Review G4 web pages
- Find Appropriate documentation
- Find relevant Medical Physics examples
- Define your preferred project
  - Simple geometry
  - Particle distributions
  - Scoring needs



# Part 2

## Introduction to Geant4

# The Geant4 toolkit

[About](#)[Download](#)[Documentation](#)[User Forum](#) <sup>↗</sup>[Bug Reports](#) <sup>↗</sup>[Events](#)[Contact Us](#)

## Geant4

Toolkit for the simulation of the passage of particles through matter. Its areas of application include high energy, nuclear and accelerator physics, as well as studies in medical and space science.

[Getting started](#)

### Get started

Everything you need to get started with Geant4.

[I'm ready to start!](#)

### Download

Geant4 source code and installers are available for download, with source code under an [open source license](#).

Latest: [11.1.1](#)

### Docs

Documentation for Geant4, along with tutorials and guides, are available online.

[Read documentation](#)

### News

[» More](#)

23 Mar 2023

[2023 Planned Features](#)

03 Mar 2023

[Release 11.0.4](#)

10 Feb 2023

<https://geant4.web.cern.ch/>

# Outline of Part2

- General Introduction to G4
  - What is G4 ?
  - Review of user documentation
  - Geant4 as a toolkit
- Basics of OO programming
- Geant4 Kernel and basics of the toolkit
  - Run, Event, Step
  - Particle and Physics processes
  - User classes
- Introduction to G4 examples
- Brief explanation of major user classes
  - **Physics List**, Particle generator action, DetectorConstruction, SensitiveDetectors

# G4 Application Developer

🏠 Book For Application Developers



10.5

Search docs

Introduction

Getting Started with Geant4 - Running a Simple Example

Toolkit Fundamentals

Detector Definition and Response

Tracking and Physics

User Actions

Control

[Docs](#) » Geant4 Book For Application Developers

## Geant4 Book For Application Developers

### Scope of this manual

The User's Guide for Application Developers is the first manual the reader should consult when learning about GEANT4 or developing a GEANT4 -based detector simulation program. This manual is designed to:

- introduce the first-time user to the GEANT4 object-oriented detector simulation toolkit,
- provide a description of the available tools and how to use them, and
- supply the practical information required to develop and run simulation applications which may be used in real experiments.

This manual is intended to be an overview of the toolkit, rather than an exhaustive treatment of it. Related physics discussions are not included unless required for the description of a particular tool. Detailed discussions of the physics included in GEANT4 can be found in the [Physics Reference Manual](#). Details of the design and functionality of the GEANT4 classes can be found in the [User's Guide for Toolkit Developers](#).












# Geant4 Cross Reference

## Geant4 Cross Reference

### [Cross-Referencing](#) [Geant4](#) [Geant4/](#)

Version: [ [ReleaseNotes](#) ] [ [1.0](#) ] [ [1.1](#) ] [ [2.0](#) ] [ [3.0](#) ] [ [3.1](#) ] [ [3.2](#) ] [ [4.0](#) ] [ [4.0.p1](#) ] [ [4.0.p2](#) ] [ [4.1](#) ] [ [4.1.p1](#) ] [ [5.0](#) ] [ [5.0.p1](#) ] [ [5.1](#) ] [ [5.1.p1](#) ] [ [5.2](#) ] [ [5.2.p1](#) ] [ [5.2.p2](#) ] [ [6.0](#) ] [ [6.0.p1](#) ] [ [6.1](#) ] [ [6.2](#) ] [ [6.2.p1](#) ] [ [6.2.p2](#) ] [ [7.0](#) ] [ [7.0.p1](#) ] [ [7.1](#) ] [ [7.1.p1](#) ] [ [8.0](#) ] [ [8.0.p1](#) ] [ [8.1](#) ] [ [8.1.p1](#) ] [ [8.1.p2](#) ] [ [8.2](#) ] [ [8.2.p1](#) ] [ [8.3](#) ] [ [8.3.p1](#) ] [ [8.3.p2](#) ] [ [9.0](#) ] [ [9.0.p1](#) ] [ [9.0.p2](#) ] [ [9.1](#) ] [ [9.1.p1](#) ] [ [9.1.p2](#) ] [ [9.1.p3](#) ] [ [9.2](#) ] [ [9.2.p1](#) ] [ [9.2.p2](#) ] [ [9.2.p3](#) ] [ [9.2.p4](#) ] [ [9.3](#) ] [ [9.3.p1](#) ] [ [9.3.p2](#) ] [ [9.4](#) ] [ [9.4.p1](#) ] [ [9.4.p2](#) ] [ [9.4.p3](#) ] [ [9.4.p4](#) ] [ [9.5](#) ] [ [9.5.p1](#) ] [ [9.5.p2](#) ] [ [9.6](#) ] [ [9.6.p1](#) ] [ [9.6.p2](#) ] [ [9.6.p3](#) ] [ [9.6.p4](#) ] [ [10.0](#) ] [ [10.0.p1](#) ] [ [10.0.p2](#) ] [ [10.0.p3](#) ] [ [10.0.p4](#) ] [ [10.1](#) ] [ [10.1.p1](#) ] [ [10.1.p2](#) ] [ [10.1.p3](#) ] [ [10.2](#) ] [ [10.2.p1](#) ] [ [10.2.p2](#) ] [ [10.2.p3](#) ] [ [10.3](#) ] [ [10.3.p1](#) ] [ [10.3.p2](#) ] [ [10.3.p3](#) ] [ [10.4](#) ] [ [10.4.p1](#) ] [ [10.4.p2](#) ] [ [10.4.p3](#) ] [ [10.5](#) ] [ [10.5.p1](#) ]

- [ [source navigation](#) ] - [ [identifier search](#) ] - [ [freetext search](#) ] - [ [file search](#) ] -

Name	Size	Last modified (GMT)	Description
 <a href="#">analysis/</a>		2019-04-17 07:34:32	
 <a href="#">digits_hits/</a>		2019-04-17 07:34:31	
 <a href="#">environments/</a>		2019-04-17 07:34:33	
 <a href="#">error_propagation/</a>		2019-04-17 07:34:32	
 <a href="#">event/</a>		2019-04-17 07:34:32	
 <a href="#">examples/</a>		2019-04-17 07:34:31	
 <a href="#">externals/</a>		2019-04-17 07:34:32	
 <a href="#">g3tog4/</a>		2019-04-17 07:34:32	
 <a href="#">geometry/</a>		2019-04-17 07:34:32	
 <a href="#">global/</a>		2019-04-17 07:34:33	
 <a href="#">graphics_reps/</a>		2019-04-17 07:34:32	

# OOP programming

## OOP basic concepts

### ● Object, Class

- A class defines the abstract characteristics of a thing (object), including the thing's attributes and the thing's behaviour

### ● Inheritance

- “Subclasses” are more specialized versions of a class, which *inherit* attributes and behaviours from their parent classes (and can introduce their own)

### ● Encapsulation

- Each object exposes to any class a certain *interface* (i.e. those members accessible to that class)
- Members can be **public**, **protected** or **private**

### ● Abstraction

- Simplifying complex reality by modelling classes appropriate to the problem
- One works at the most appropriate level of inheritance for a given aspect of the problem

### ● Polymorphism

- It allows one to treat derived class members just like their parent class' members

Maria Grazia Pia

# OO technology

- Openness to **extension** and **evolution**  
new implementations can be added w/o changing the existing code
- Robustness and ease of **maintenance**  
**protocols** and well defined dependencies minimize coupling

## Strategic vision

## Toolkit

- A set of compatible components
- each component is **specialised** for a specific functionality
  - each component can be **refined** independently to a great detail
  - components can be **integrated** at any degree of complexity
  - it is easy to provide (and use) **alternative** components
  - the user application can be **customised** as needed



# Geant4 simulation toolkit

- Modeling the experimental set-up
- Tracking particles through matter
- Interaction of particles with matter
- Modeling the detector response
- Run and event control
- **Accessory utilities** (*random number generators, PDG particle information, physical constants, system of units etc.*)
  - User interface
    - Interface to event generators
  - Visualisation (*of the set-up, tracks, hits etc.*)
    - Persistency
    - Analysis



# Reference Physics Lists

- Reference physics lists attempt to cover a wide range of use cases
  - Extensive validation by LHC experiments for simulation hadronic showers
  - Comparison experiments for neutron production and transport demonstrates good agreement
    - QGSP\_BIC\_HP, QGSP\_BERT\_HP
  - user feedback, e.g. vi hypernews, is welcome
- Users responsible for validating results
- Documentation available from G4 Physics List manual
- Physics Lists User forum for questions and feedback

# To use Geant4, you have to...

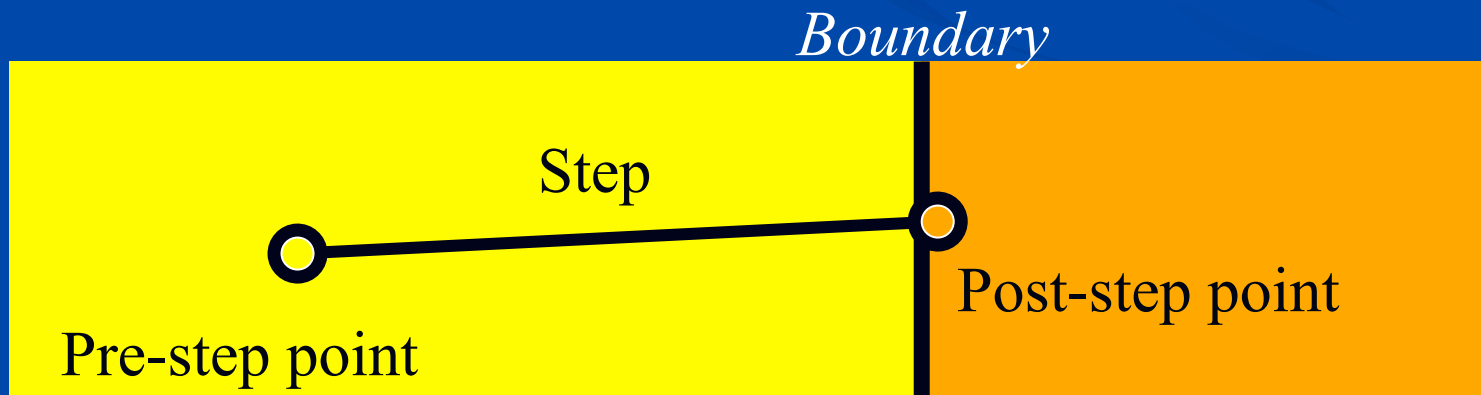
- Geant4 is a toolkit. You have to build an application.
- To make an application, you have to
  - Define your geometrical setup
    - Material, volume
  - Define physics to get involved
    - Particles, physics processes/models
    - Production thresholds
  - Define how an event starts
    - Primary track generation
  - Extract information useful to you
- You may also want to
  - Visualize geometry, trajectories and physics output
  - Utilize (Graphical) User Interface
  - Define your own UI commands
  - etc.

# User Classes needs

- Define material and geometry
  - ➔ G4VUserDetectorConstruction
- Select appropriate particles and processes and define production threshold(s)
  - ➔ G4VUserPhysicsList
- Define the way of primary particle generation
  - ➔ G4VUserPrimaryGeneratorAction
- Define the way to extract useful information from Geant4
  - ➔ G4UserSteppingAction, G4UserTrackingAction, etc.
  - ➔ G4VUserDetectorConstruction, G4UserEventAction, G4Run, G4UserRunAction
  - ➔ G4SensitiveDetector, G4VHit, G4VHitsCollection
  - ➔ G4PrimitiveScorers

# Step in Geant4

- Step has two points and also “delta” information of a particle (energy loss on the step, time-of-flight spent by the step, etc.).
- Each point knows the volume (and material). In case a step is limited by a volume boundary, the end point physically stands on the boundary, and it **logically belongs to the next volume**.
  - Because one step knows materials of two volumes, boundary processes such as transition radiation or refraction could be simulated.
- **G4SteppingManager** class manages processing a step, a step is represented by **G4Step** class.
- **G4UserSteppingAction** is the optional user hook.



# Particle in Geant4

- A particle in Geant4 is represented by three layers of classes.
- **G4Track**
  - Position, geometrical information, etc.
  - This is a class representing a particle to be tracked.
- **G4DynamicParticle**
  - "Dynamic" physical properties of a particle, such as momentum, energy, spin, etc.
  - Each G4Track object has its own and unique G4DynamicParticle object.
  - This is a class representing an individual particle.
- **G4ParticleDefinition**
  - "Static" properties of a particle, such as charge, mass, life time, decay channels, etc.
  - G4ProcessManager which describes processes involving to the particle
  - All G4DynamicParticle objects of same kind of particle share the same G4ParticleDefinition.

# Part 2

## G4 installation

# Installation of Geant4

- Follow instruction on G4 web site:  
<http://www.cern.ch/geant4>
- Download the code:  
<http://geant4.web.cern.ch/geant4/support/download.shtml>
- Download data tables
- System requirements
- Visualization, Geometry and Analysis tools

# Visualization, Analysis, Geometry ...

- Geant4 is a toolkit
- Many interfaces have been developed
  - Analysis (ROOT, jAIDA, OpenScientist)
  - Visualization (OpenGL, VRML, Dawn/eps ...)
  - Geometry (XML description, CAD interfaces...)
- The basics you need is:
  - g++
  - a visualization driver
- cmake to build the executables



# G4 Virtual Machine



---

## GEANT4 VIRTUAL MACHINE

G4VM with RockyLinux 8.5 Geant4.11.1.1

[Presentation](#) [News](#) [What is included](#) [Tutorial](#) [Hints](#) [Publications](#) [Download](#) [Contacts](#)

Since 2004, [LP2i Bordeaux](#), CNRS / IN2P3 / Bordeaux University laboratory provides free of charge and licensing to Geant4 users a **Geant4 Virtual Machine** with several visualisation, analysis and development tools.

Mode sombre : Off

<https://extra.lp2ib.in2p3.fr/G4/>

Geant4 Tutorial Introduction F.Longo

# Laboratory - 5

- Check the installation of G4 through the Virtual Machine

# Part 2

## G4 examples

# Evaluation for the “Laboratory”

- Discussion of Geant4 example
  - G4 example relevant to Medical Physics
- Discussion of requirements and methods
  - Medical Physics “environment”
  - Geant4 modeling
  - “Basic” analysis of results
- Realization of “new” example – Laboratory
  - Might just be an improvement of an existing G4 example

# Geant4 Basic Examples

# Basic Examples

- The set of basic examples is oriented to "novice" users and covering many basic general use-cases typical of an "application"-oriented kind of development.
- **Example B1**
  - Simple geometry with a few solids
  - Geometry with simple placements (G4PVPlacement)
  - Scoring total dose in a selected volume user action classes
  - Geant4 physics list (QBBC)
- **Example B2**
  - Simplified tracker geometry with global constant magnetic field
  - Geometry with simple placements (G4PVPlacement) and parametrisation (G4PVParametrisation)
  - Scoring within tracker via G4 sensitive detector and hits
  - Geant4 physics list (FTFP\_BERT) with step limiter
  - Started from novice/N02 example
- **Example B3**
  - Schematic Positron Emitted Tomography system
  - Geometry with simple placements with rotation (G4PVPlacement)
  - Radioactive source
  - Scoring within Crystals via G4 scorers
  - Modular physics list built via builders provided in Geant4

# Basic Examples

- The set of basic examples is oriented to "novice" users and covering many basic general use-cases typical of an "application"-oriented kind of development.
- **Example B4**
  - Simplified calorimeter with layers of two materials
  - Geometry with replica (G4PVReplica)
  - Scoring within layers in four ways: via user actions (a), via user own object (b), via G4 sensitive detector and hits (c) and via scorers (d)
  - Geant4 physics list (FTFP\_BERT)
  - Histograms (1D) and ntuple saved in the output file
  - Started from novice/N03 example
- **Example B5**
  - A double-arm spectrometer with wire chambers, hodoscopes and calorimeters with a local constant magnetic field
  - Geometry with placements with rotation, replicas and parameterisation
  - Scoring within wire chambers, hodoscopes and calorimeters via G4 sensitive detector and hits
  - Geant4 physics list (FTFP\_BERT) with step limiter
  - UI commands defined using G4GenericMessenger
  - Histograms (1D) and ntuple saved in the output file
  - Started from extended/analysis/A01

# Geant4 Extended Examples



# Extended Examples

- The set of "extended" examples is covering various use-cases and may require some additional libraries besides of Geant4.
- analysis
  - Histogramming through the AIDA interface
- biasing
  - Examples of event biasing, scoring and reverse-MC-
- common
  - A set of common classes which can be reused in other examples demonstrating just a particular feature
- electromagnetic
  - Specific EM physics simulation with histogramming
- errorpropagation
  - Use of the error propagation utility (Geant4e)
- eventgenerator
  - Applications demonstrating various ways of primary event generation: using Geant4 particle gun, Geant4 general particle source, using interface to HepMC, Pythia
- exoticphysics
  - Exotic simulation applications (classical magnetic monopole, etc...)
- field
  - Specific simulation setups in magnetic field

# Extended Examples

- g3tog3
  - Examples of usage of the g3tog4 converter tool
- geometry
  - Specific geometry examples and tools, OLAP tool for detection of overlapping geometries,
- hadronic
  - Specific hadronic physics simulation with histogramming
- medical
  - Specific examples for medical physics applications
- optical
  - Examples of generic optical processes simulation setups
- parallel
  - Examples of event-level parallelism in Geant4 using the TOP-C distribution, and MPI technique
- parameterisations
  - Examples for fast shower parameterisations according to specific models (gflash)

# Extended Examples

- persistency
  - Persistency of geometry (GDML or ASCII) and simulation output
- polarisation
  - Use of physics processes including polarization
- radioactivedecay
  - Examples to simulate the decays of radioactive isotopes and induced radioactivity resulted from nuclear interactions
- runAndEvent
  - Examples to demonstrate how to connect the information between primary particles and hits and utilize user-information classes
- visualization
  - Specific visualization features and graphical customisations

# Geant4 Advanced Examples

# Advanced Examples

- air\_shower
- ams\_Ecal
- Brachytherapy
- ChargeExchangeMC
- composite\_calorimeter
- eRosita
- Gammaknife
- gammaray\_telescope
- hadrontherapy
- human\_phantom
- iort\_therapy
- IAr\_calorimeter
- medical\_linac
- microbeam
- microelectronics
- nanobeam
- purging\_magnet
- radioprotection
- underground\_physics
- xray\_fluorescence
- xray\_telescope

# Laboratory - 6

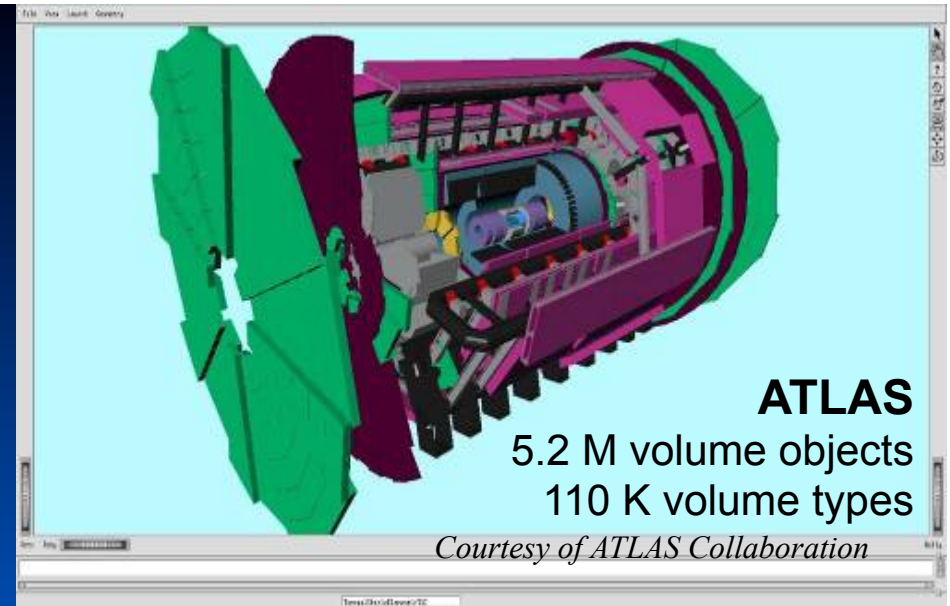
- Find the examples relevant to Medical Physics

# Part 2

## G4 geometry

# Geometry

- Role
  - detailed detector description
  - efficient navigation
- Three conceptual layers
  - **Solid**: shape, size
  - **LogicalVolume**: material, sensitivity, daughter volumes, etc.
  - **PhysicalVolume**: position, rotation
- One can do fancy things with geometry...



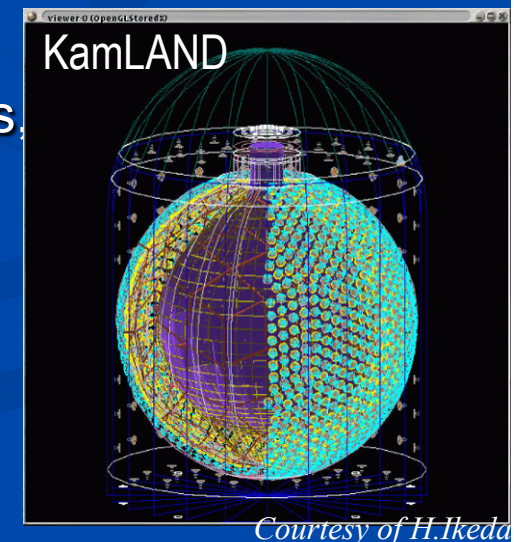
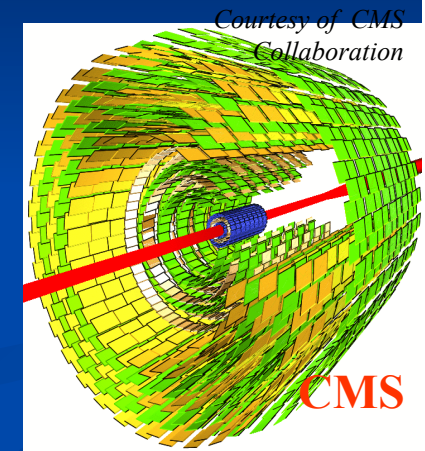
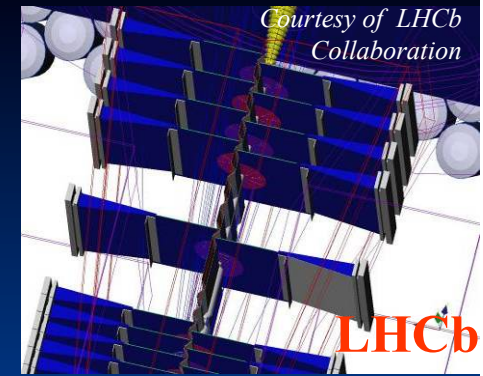
Boolean operations



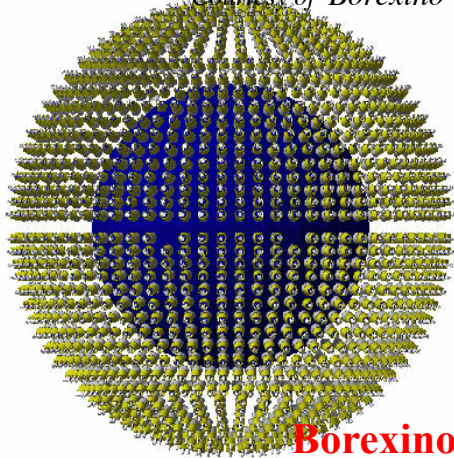


# Solids

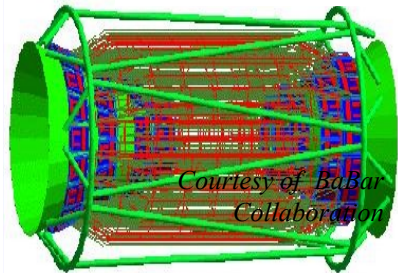
Multiple representations  
Same abstract interface



Courtesy of Borexino



BaBar

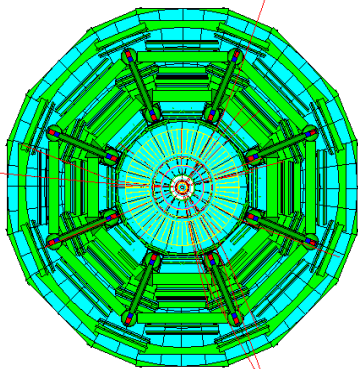


- CSG (Constructed Solid Geometries)

- simple solids

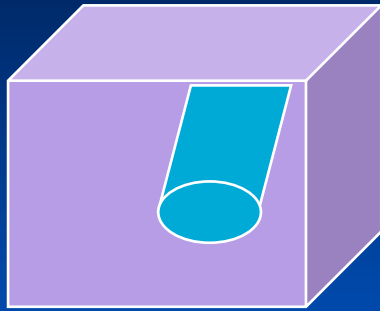
- STEP extensions *CAD exchange*

- polyhedra, spheres, cylinders, cones, toroids, etc.

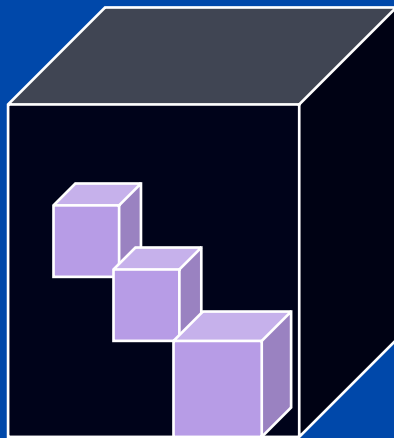
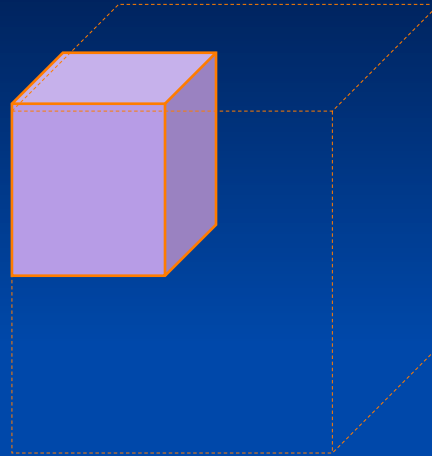


ATLAS

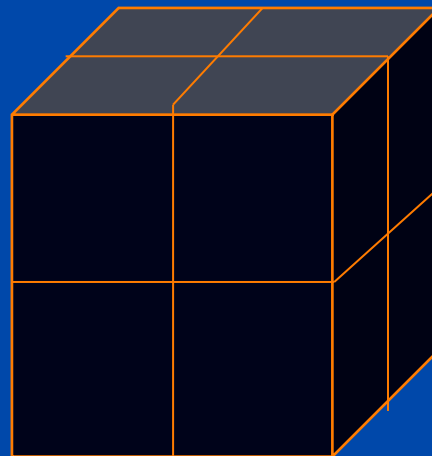
# Physical Volumes



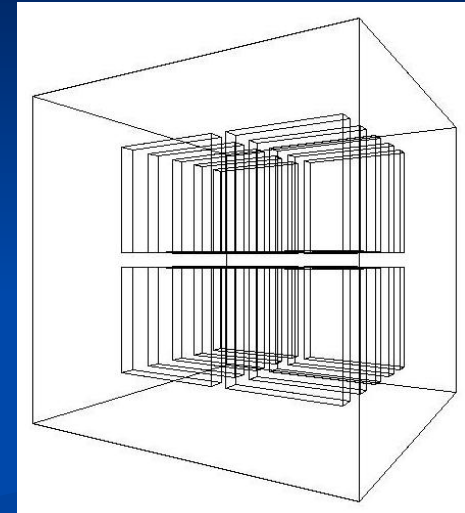
placement



parameterised



replica



assembled

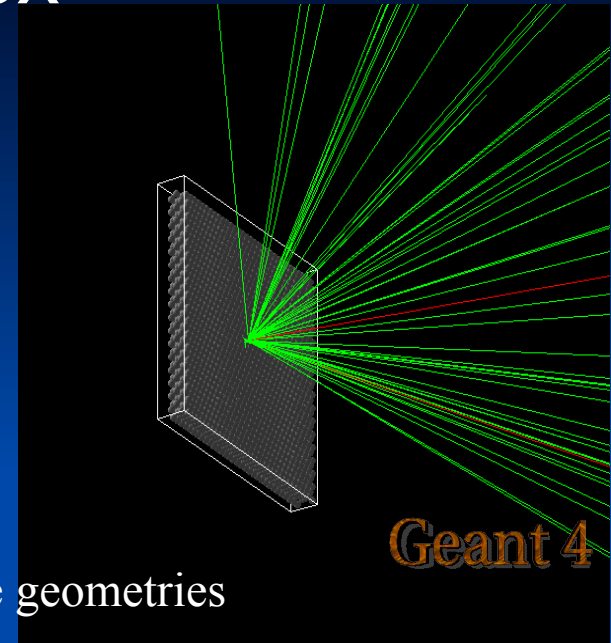
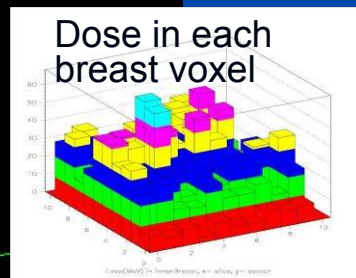
Versatility to describe complex geometries

# Not only large scale, complex detectors....

Analytical breast

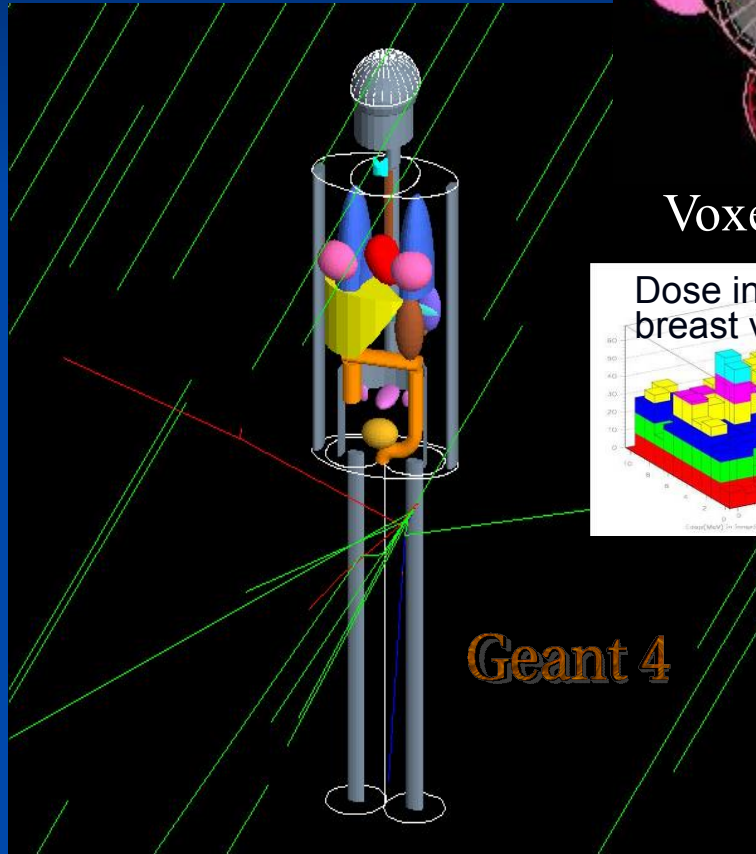


Voxel breast

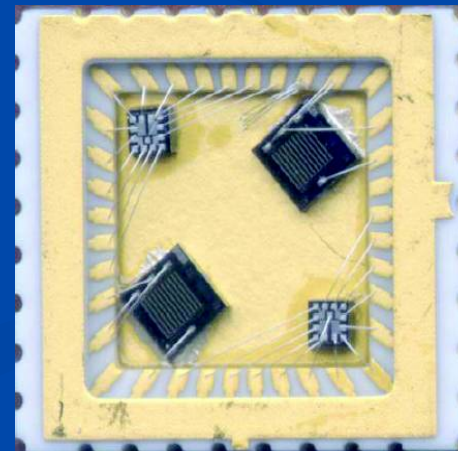


simple geometries

small scale components



Geant4 anthropomorphic phantoms



# G4VUserDetectorConstruction

# User classes

- **main()**
  - Geant4 does not provide *main()*.
  - Note : classes written in **yellow** are mandatory.
- Initialization classes
  - Use G4RunManager::**SetUserInitialization()** to define.
  - Invoked at the initialization
    - **G4VUserDetectorConstruction** ←
    - **G4VUserPhysicsList**
- Action classes
  - Use G4RunManager::**SetUserAction()** to define.
  - Invoked during an event loop
    - **G4VUserPrimaryGeneratorAction**
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction



# Your detector construction

```
#ifndef MyDetectorConstruction_h
#define MyDetectorConstruction_h 1
#include "G4VUserDetectorConstruction.hh"
class MyDetectorConstruction
    : public G4VUserDetectorConstruction
{
public:
    G4VUserDetectorConstruction();
    virtual ~G4VUserDetectorConstruction();
    virtual G4VPhysicalVolume* Construct();
    virtual void ConstructSDandField();

public:
    // set/get methods if needed
private:
    // granular private methods if needed
    // data members if needed
};
#endif
```

# Describe your detector

- Derive your own concrete class from **G4VUserDetectorConstruction** abstract base class.
- Implement the method `Construct()`
  - 1) Construct all necessary materials
  - 2) Define shapes/solids
  - 3) Define logical volumes
  - 4) Place volumes of your detector geometry
  - 5) Define visualization attributes for the detector elements (*optional*)
  - 6) Define regions (*optional*)
- Implement the method `ConstructSDandField()`
  - 1) Associate (magnetic) field to geometry (*optional*)
  - 2) Instantiate sensitive detectors / scorers and set them to corresponding volumes (*optional*)
- Set your construction class to `G4RunManager`
- It is suggested to **modularize** `Construct()` method w.r.t. each component or sub-detector for easier maintenance of your code.

# Geometry - Materials



# Definition of Materials

- Different kinds of materials can be described:
  - isotopes <-> G4Isotope
  - elements <-> G4Element
  - molecules, compounds and mixtures <-> G4Material
- Attributes associated to G4Material:
  - temperature, pressure, state, density
- Prefer low-density material to vacuum

- Single element material

```
double density = 1.390*g/cm3;  
double a = 39.95*g/mole;  
G4Material* lAr =  
    new G4Material("liquidArgon",z=18.,a,density);
```

# Material: molecule

- A Molecule is made of several elements (composition by **number of atoms**)

```
a = 1.01*g/mole;
G4Element* elH =
    new G4Element("Hydrogen", symbol="H", z=1., a);
a = 16.00*g/mole;
G4Element* elO =
    new G4Element("Oxygen", symbol="O", z=8., a);
density = 1.000*g/cm3;
G4Material* H2O =
    new G4Material("Water", density, ncomp=2);
G4int natoms;
H2O->AddElement(elH, natoms=2);
H2O->AddElement(elO, natoms=1);
```

# Material: compound

- Compound: composition by **fraction of mass**

```
a = 14.01*g/mole;
G4Element* elN =
    new G4Element(name="Nitrogen", symbol="N", z= 7., a);
a = 16.00*g/mole;
G4Element* elO =
    new G4Element(name="Oxygen", symbol="O", z= 8., a);
density = 1.290*mg/cm3;
G4Material* Air =
    new G4Material(name="Air", density, ncomponents=2);
G4double fracMass;
Air->AddElement(elN, fracMass=70.0*perCent);
Air->AddElement(elO, fracMass=30.0*perCent);
```

# Material: mixture

- Composition of compound materials

```
G4Element* e1C = ...; // define "carbon" element
G4Material* SiO2 = ...; // define "quartz" material
G4Material* H2O = ...; // define "water" material

density = 0.200*g/cm3;
G4Material* Aerog =
    new G4Material("Aerogel", density, ncomponents=3);
Aerog->AddMaterial(SiO2, fractionmass=62.5*perCent);
Aerog->AddMaterial(H2O, fractionmass=37.4*perCent);
Aerog->AddElement(e1C, fractionmass=0.1*perCent);
```

# Element with user defined abundance

- An element can be created according to user defined abundances
- Ex. Create an enriched Uranium for nuclear power generation

```
G4Isotope* isoU235 =
```

```
    new G4Isotope("U235", iz=92, ia=235, a=235.0439242*g/mole);
```

```
G4Isotope* isoU238 =
```

```
    new G4Isotope("U238", iz=92, ia=238, a=238.0507847 *g/mole);
```

```
G4Element* elenrichedU =
```

```
    new G4Element("enriched U", symbol="U" , ncomponents=2);
```

```
elenrichedU->AddIsotope(isoU235, abundance=80.*perCent);
```

```
elenrichedU->AddIsotope(isoU238, abundance=20.*perCent);
```

```
G4Material* materichedU=
```

```
    new G4Material("U for nuclear power generation" , density= 19.050*g/  
    cm3 , ncomponents = 1 , kStateSolid );
```

```
materichedU>AddElement( elenrichedU , fractionmass = 1 );
```

# NIST Material Database in Geant4

# NIST materials in Geant4

=====  
 ### Elementary Materials from the NIST Data Base  
 =====

Z	Name	ChFormula	density(g/cm <sup>3</sup> )	I(eV)
1	G4_H	H_2	8.3748e-05	19.2
2	G4_He		0.000166322	41.8
3	G4_Li		0.534	40
4	G4_Be		1.848	63.7
5	G4_B		2.37	76
6	G4_C		2	81
7	G4_N	N_2	0.0011652	82
8	G4_O	O_2	0.00133151	95
9	G4_F		0.00158029	115
10	G4_Ne		0.000838505	137
11	G4_Na		0.971	149
12	G4_Mg		1.74	156
13	G4_Al		2.6989	166
14	G4_Si		2.33	173

- NIST Elementary Materials
  - H to Cf
- NIST Compounds
- HEP and Nuclear Materials
  - Ex. liquid Ar PbWO<sub>4</sub>

=====  
 ### Compound Materials from the NIST Data Base  
 =====

N	Name	ChFormula	density(g/cm <sup>3</sup> )	I(eV)
13	G4_Adipose_Tissue		0.92	63.2
1		0.119477		
6		0.63724		
7		0.00797		
8		0.232333		
11		0.0005		
12		2e-05		
15		0.00016		
16		0.00073		
17		0.00119		
19		0.00032		
20		2e-05		
26		2e-05		
30		2e-05		
4	G4_Air		0.00120479	85.7
6		0.000124		
7		0.755268		
8		0.231781		
18		0.012827		
2	G4_CsI		4.51	553.1
53		0.47692		
55		0.52308		

# How to use

- Do not need anymore to predefine elements and materials
- Main new user interfaces:

```
G4NistManager* nist = G4NistManager::Instance();
```

```
G4Element* elm = manager->FindOrBuildElement("symb", G4bool iso);
```

```
G4Element* elm = manager->FindOrBuildElement(G4int Z, G4bool iso);
```

```
G4Material* mat = manager->FindOrBuildMaterial("name", G4bool iso);
```

```
G4Material* mat = manager->ConstructNewMaterial("name",  
    const std::vector<G4int>& Z,  
    const std::vector<G4double>& weight,  
    G4double density, G4bool iso);
```

```
G4double isotopeMass = manager->GetMass(G4int Z, G4int N);
```

## UI commands

```
/material/nist/printElement --- print defined elements
```

```
/material/nist/listMaterials --- print defined materials
```



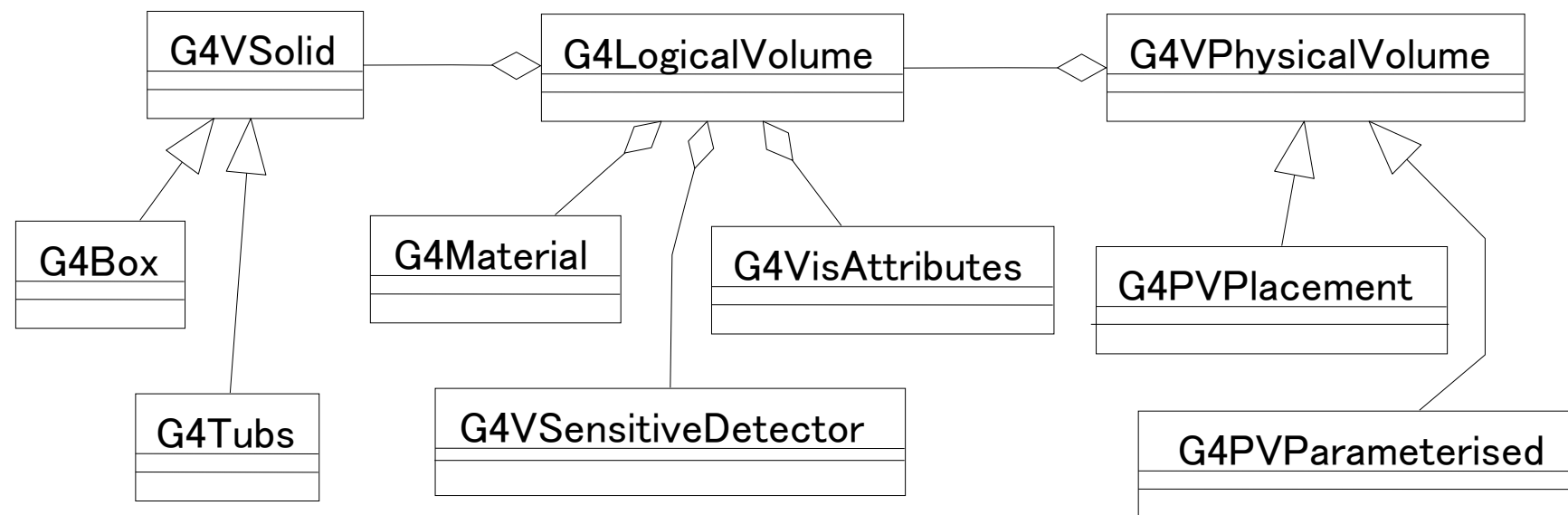
# Laboratory - 7

- Find the materials in G4 example basic B3

# Geometry - Volumes

# Detector geometry

- Three conceptual layers
  - **G4VSolid** -- *shape, size*
  - **G4LogicalVolume** -- *daughter physical volumes, material, sensitivity, user limits, etc.*
  - **G4VPhysicalVolume** -- *position, rotation*

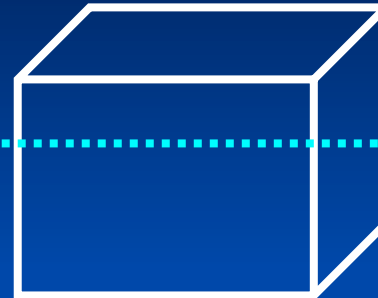


# Define detector geometry

- Basic strategy

```
G4VSolid* pBoxSolid =  
    new G4Box("aBoxSolid",  
             1.*m, 2.*m, 3.*m);  
G4LogicalVolume* pBoxLog =  
    new G4LogicalVolume( pBoxSolid,  
                        pBoxMaterial, "aBoxLog", 0, 0, 0);  
G4VPhysicalVolume* aBoxPhys =  
    new G4PVPlacement( pRotation,  
                      G4ThreeVector(posX, posY, posZ),  
                      pBoxLog, "aBoxPhys", pMotherLog,  
                      0, copyNo);
```

Logical volume :  
+ material, sensitivity, etc.  
solid, shape, and size

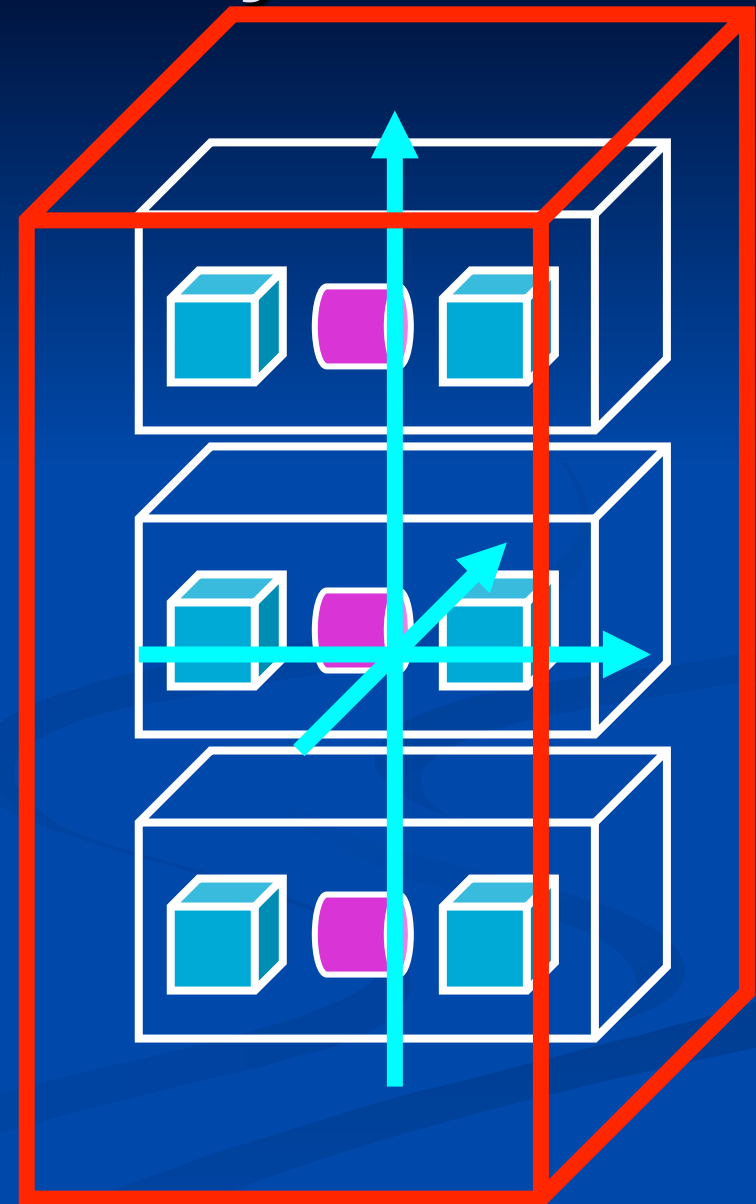


Physical volume :  
+ rotation and position

- A volume is placed in its mother volume. Position and rotation of the daughter volume is described with respect to the local coordinate system of the mother volume. The origin of mother volume's local coordinate system is at the center of the mother volume.
  - Daughter volume cannot protrude from mother volume.

# Geometrical hierarchy

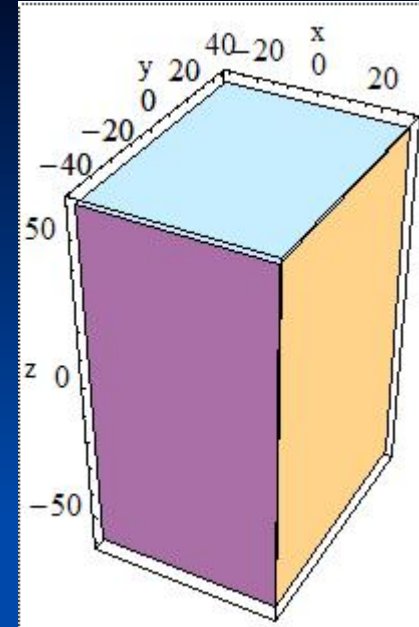
- One logical volume can be placed more than once. One or more volumes can be placed to a mother volume.
- Note that the mother-daughter relationship is an information of G4LogicalVolume.
  - If the mother volume is placed more than once, all daughters are by definition appear in all of mother physical volumes.
- The **world volume** must be a unique physical volume which **fully contains** all the other volumes.
  - The world volume defines the **global coordinate system**. The origin of the global coordinate system is at the center of the world volume.
  - Position of a track is given **with respect to the global coordinate system**.



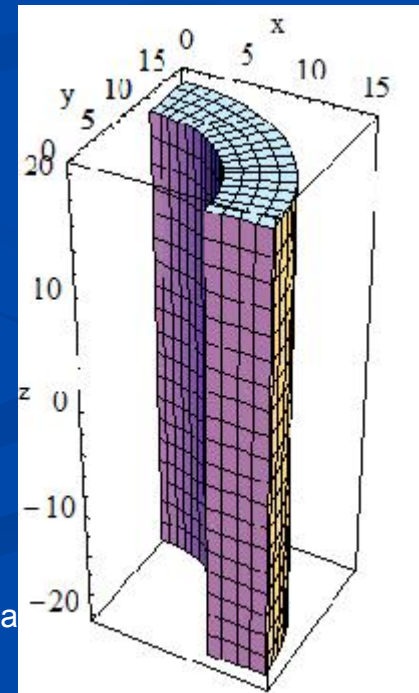
# Solid volume

# CSG: G4Box, G4Tubs

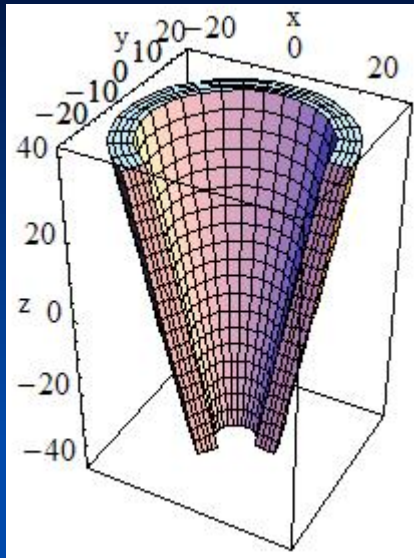
```
G4Box(const G4String &name, // name
      G4double half_x, // X half size
      G4double half_y, // Y half size
      G4double half_z); // Z half size
```



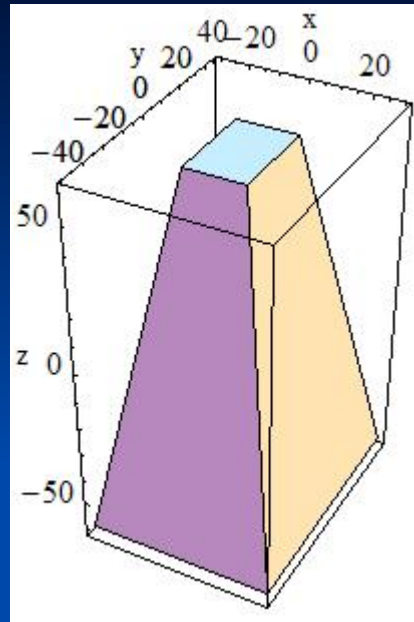
```
G4Tubs(const G4String &name, // name
      G4double pRmin, // inner radius
      G4double pRmax, // outer radius
      G4double pDz, // Z half length
      G4double pSphi, // starting Phi
      G4double pDphi); // segment angle
```



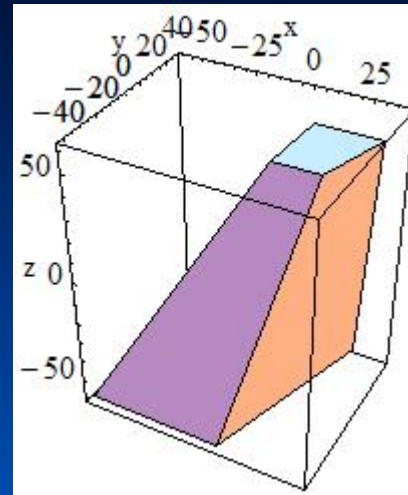
# Other CSG solids



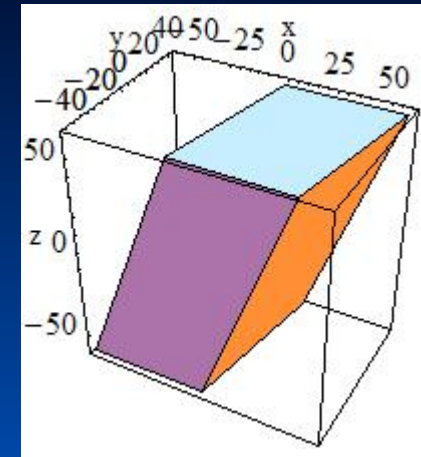
G4Cons



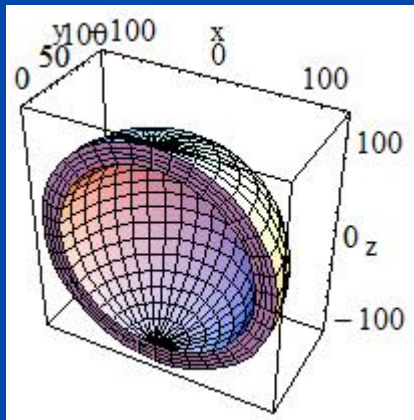
G4Trd



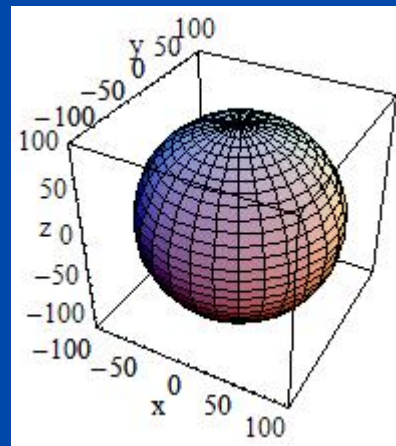
G4Trap



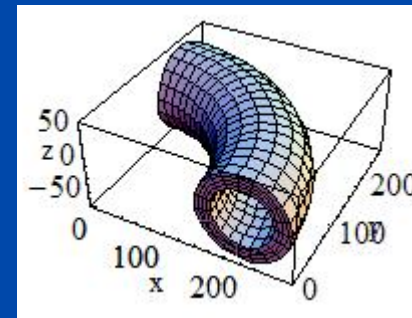
G4Para  
(parallelepiped)



G4Sphere



G4Orb  
(full solid sphere)



G4Torus

Consult to  
[Section 4.1.2 of Geant4 Application Developers Guide](#) for all available  
shapes.



# G4LogicalVolume

# G4LogicalVolume

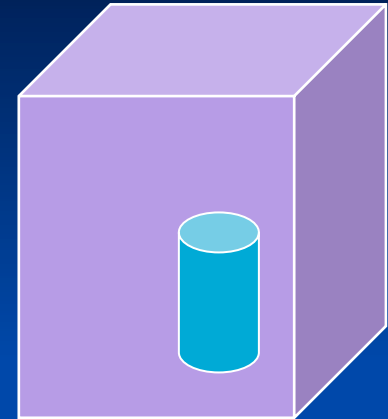
```
G4LogicalVolume(G4VSolid *pSolid,  
                G4Material *pMaterial,  
                const G4String &name,  
                G4FieldManager *pFieldMgr=0,  
                G4VSensitiveDetector *pSDetector=0,  
                G4UserLimits *pULimits=0);
```

- Contains all information of volume except position and rotation
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Position of daughter volumes
  - Magnetic field, User limits, Region
- Physical volumes of same type can share the common logical volume object.
- The pointers to solid must **NOT** be null.
- The pointers to material must **NOT** be null for tracking geometry.

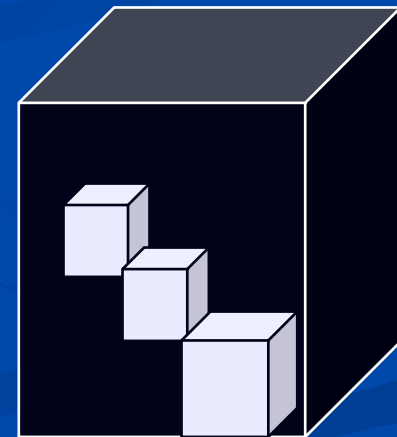
# Physical volume

# Physical Volumes

- Placement volume : it is one positioned volume
  - One physical volume object represents one “real” volume.
- Repeated volume : a volume placed many times
  - One physical volume object represents any number of “real” volumes.
  - reduces use of memory.
  - Parameterised
    - repetition w.r.t. copy number
  - Replica and Division
    - simple repetition along one axis



*placement*

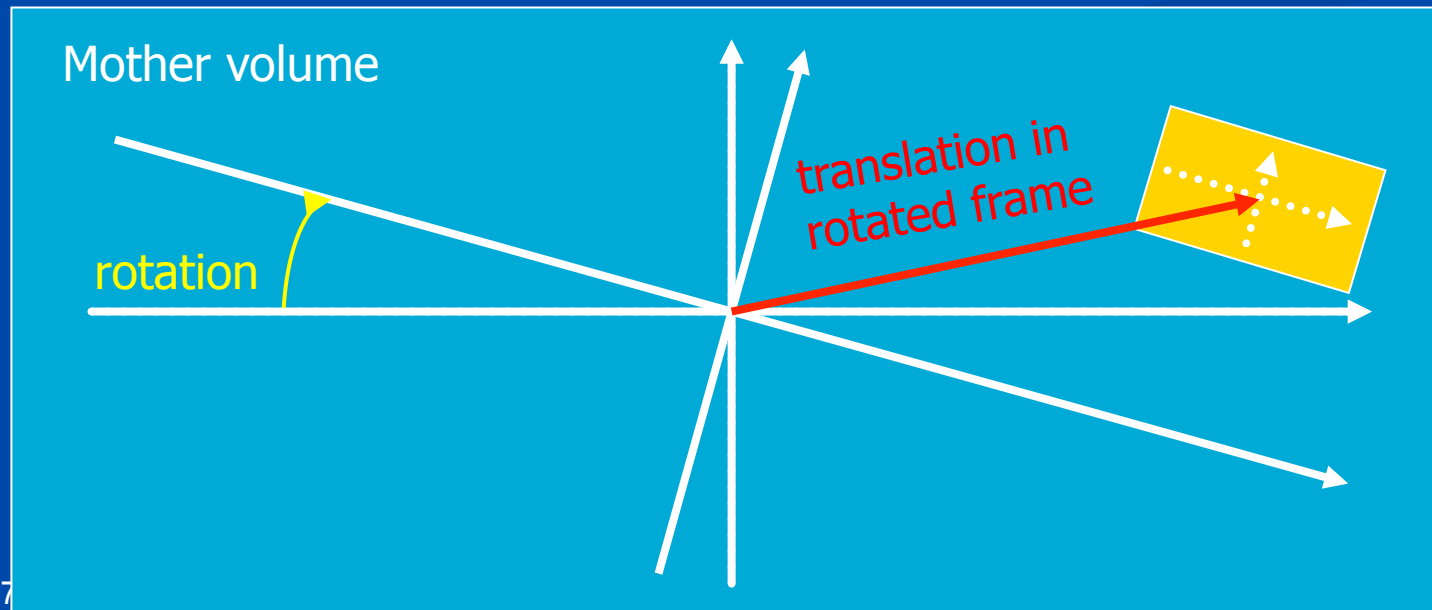


*repeated*

# G4PVPlacement

```
G4PVPlacement(G4RotationMatrix* pRot, // rotation of mother frame
              const G4ThreeVector &tlate, // position in rotated frame
              G4LogicalVolume *pDaughterLogical,
              const G4String &pName,
              G4LogicalVolume *pMotherLogical,
              G4bool pMany, // 'true' is not supported yet...
              G4int pCopyNo, // unique arbitrary integer
              G4bool pSurfChk=false); // optional boundary check
```

- Single volume positioned relatively to the mother volume.



# Alternative G4PVPlacement

G4PVPlacement (

```
G4Transform3D (G4RotationMatrix &pRot, // rotation of daughter frame  
               const G4ThreeVector &tlate), // position in mother frame
```

```
G4LogicalVolume *pDaughterLogical,
```

```
const G4String &pName,
```

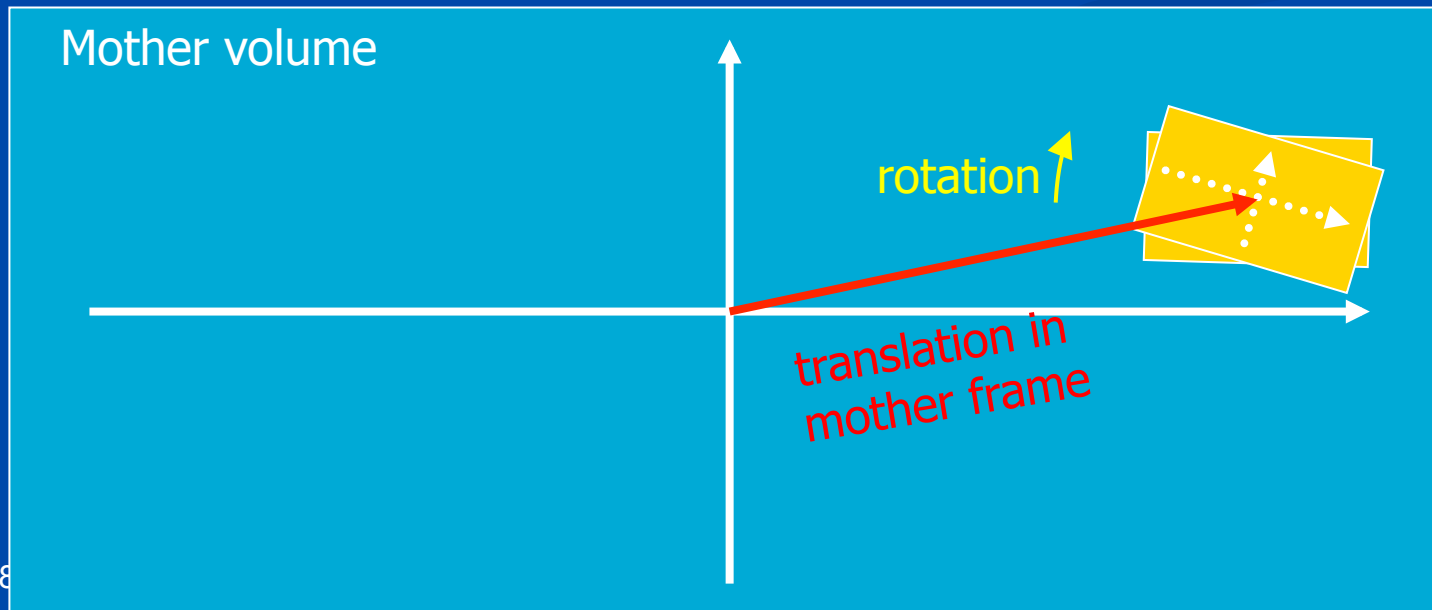
```
G4LogicalVolume *pMotherLogical,
```

```
G4bool pMany, // 'true' is not supported yet...
```

```
G4int pCopyNo, // unique arbitrary integer
```

```
G4bool pSurfChk=false); // optional boundary check
```

- Single volume positioned relatively to the mother volume.



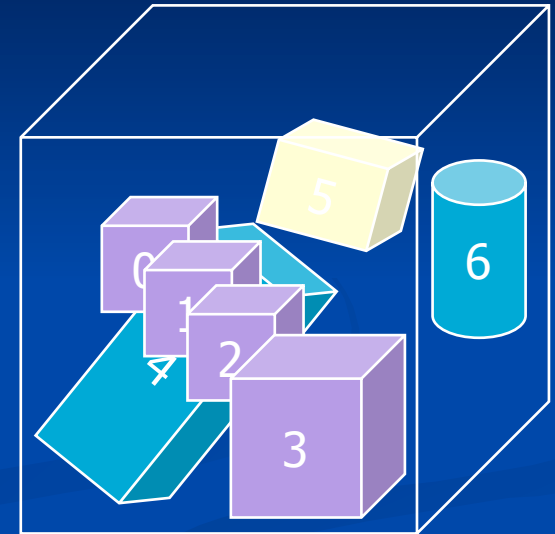
# G4PVParameterised

```
G4PVParameterised(const G4String& pName,  
                  G4LogicalVolume* pLogical,  
                  G4LogicalVolume* pMother,  
                  const EAxis pAxis,  
                  const G4int nReplicas,  
                  G4VPVParameterisation *pParam  
                  G4bool pSurfChk=false);
```

- Replicates the volume **nReplicas** times using the parameterization **pParam**, within the mother volume **pMother**
- **pAxis** is a **suggestion** to the navigator along which Cartesian axis replication of parameterized volumes dominates.
  - **kXAxis**, **kYAxis**, **kZAxis** : one-dimensional optimization
  - **kUndefined** : three-dimensional optimization

# Parameterized Physical Volumes

- User should implement a class derived from **G4VPVParameterisation** abstract base class and define following **as a function of copy number**
  - where it is positioned (transformation, rotation)
- Optional:
  - the size of the solid (dimensions)
  - the type of the solid, material, sensitivity, vis attributes
- All daughters must be fully contained in the mother.
- Daughters should not overlap to each other.
- Typical use-cases
  - Complex detectors
    - with large repetition of volumes, regular or irregular
  - Medical applications
    - the material in animal tissue is measured as cubes with varying material





# G4PVPParameterized : example

```
G4VSolid* solidChamber =  
    new G4Box("chamber", 100*cm, 100*cm, 10*cm);  
  
G4LogicalVolume* logicChamber =  
    new G4LogicalVolume  
    (solidChamber, ChamberMater, "Chamber", 0, 0, 0);  
  
G4VPVParameterisation* chamberParam =  
    new ChamberParameterisation();  
  
G4VPhysicalVolume* physChamber =  
    new G4PVPParameterised("Chamber", logicChamber,  
        logicMother, kZAxis, NbOfChambers, chamberParam);
```

# G4VPVParameterisation : example

```
class ChamberParameterisation : public G4VPVParameterisation
{
public:
    ChamberParameterisation();
    virtual ~ChamberParameterisation();
    virtual void ComputeTransformation // position, rotation
        (const G4int copyNo, G4VPhysicalVolume* physVol) const;
    virtual void ComputeDimensions // size
        (G4Box& trackerLayer, const G4int copyNo,
         const G4VPhysicalVolume* physVol) const;
    virtual G4VSolid* ComputeSolid // shape
        (const G4int copyNo, G4VPhysicalVolume* physVol);
    virtual G4Material* ComputeMaterial // material, sensitivity, visAtt
        (const G4int copyNo, G4VPhysicalVolume* physVol,
         const G4VTouchable *parentTouch=0);
        // G4VTouchable should not be used for ordinary parameterization
};
```

# Replicated Volumes

- The mother volume is **completely filled** with replicas, all of which are the **same size (width)** and **shape**.
- Replication may occur along:
  - Cartesian axes (X, Y, Z) – slices are considered perpendicular to the axis of replication
    - Coordinate system at the center of each replica
  - Radial axis (Rho) – cons/tubs sections centered on the origin and un-rotated
    - Coordinate system same as the mother
  - Phi axis (Phi) – phi sections or wedges, of cons/tubs form
    - Coordinate system rotated such as that the X axis bisects the angle made by each wedge



a daughter  
logical volume to  
be replicated



mother volume

# G4PVReplica

```
G4PVReplica(const G4String &pName,  
            G4LogicalVolume *pLogical,  
            G4LogicalVolume *pMother,  
            const EAxis pAxis,  
            const G4int nReplicas,  
            const G4double width,  
            const G4double offset=0.);
```

- `offset` may be used only for tube/cone segment
- Features and restrictions:
  - Replicas can be placed inside other replicas
  - Normal placement volumes can be placed inside replicas, assuming no intersection/overlaps with the mother volume or with other replicas

# Laboratory - 8

- Identify the geometrical shapes and positions used in G4 example B3

# Part 2

## G4 particle generation

# Primary particle generation

# G4VUserPrimaryGeneratorAction

- This class is one of mandatory user classes to **control the generation** of primaries.
  - This class itself **should NOT** generate primaries but **invoke** `GeneratePrimaryVertex()` method of primary generator(s) to make primaries.
- Constructor
  - Instantiate primary generator(s)
  - Set default values to it(them)
- **GeneratePrimaries()** method
  - Randomize particle-by-particle value(s)
  - Set these values to primary generator(s)
    - Never use hard-coded UI commands
  - Invoke **GeneratePrimaryVertex()** method of primary generator(s)



# **Built-in primary particle generators**

# G4ParticleGun

- Concrete implementations of G4VPrimaryGenerator
  - A good example for experiment-specific primary generator implementation
- It shoots one primary particle of a certain energy from a certain point at a certain time to a certain direction.
  - Various set methods are available
  - Intercoms commands are also available for setting initial values
- One of most frequently asked questions is :

I want “particle shotgun”, “particle machinegun”, etc.
- Instead of implementing such a fancy weapon, in your implementation of UserPrimaryGeneratorAction, you can
  - Shoot random numbers in arbitrary distribution
  - Use set methods of G4ParticleGun
  - Use G4ParticleGun as many times as you want
  - Use any other primary generators as many times as you want to make overlapping events

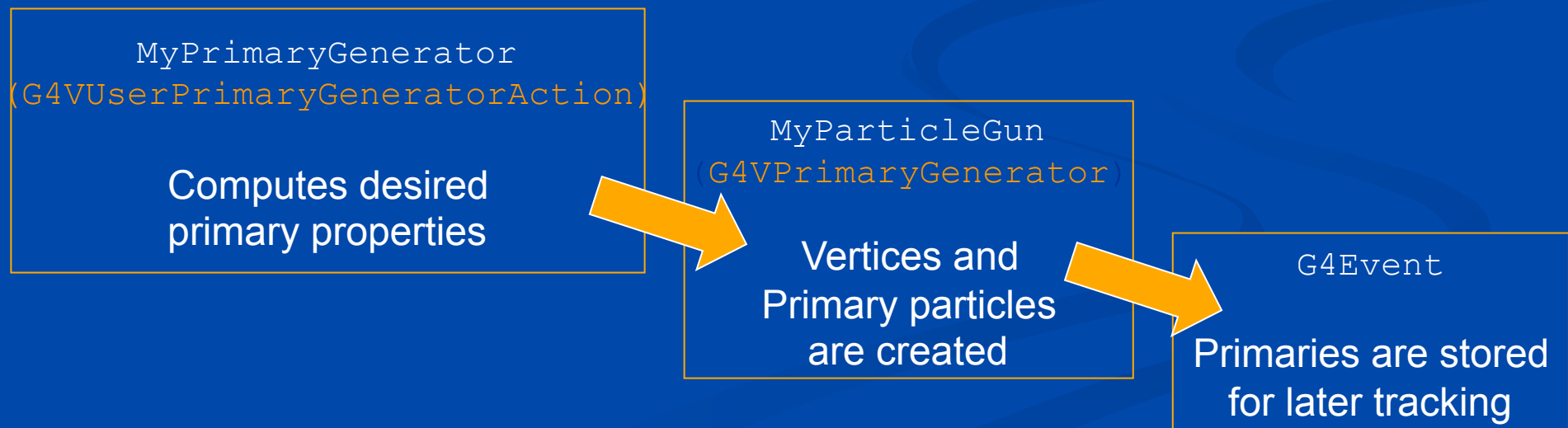
# G4VUserPrimaryGeneratorAction

```
void T01PrimaryGeneratorAction::
    GeneratePrimaries(G4Event* anEvent)
{ G4ParticleDefinition* particle;
  G4int i = (int)(5.*G4UniformRand());
  switch(i)
  { case 0: particle = positron; break; ... }
  particleGun->SetParticleDefinition(particle);
  G4double pp =
    momentum+(G4UniformRand()-0.5)*sigmaMomentum;
  G4double mass = particle->GetPDGMass();
  G4double Ekin = sqrt(pp*pp+mass*mass)-mass;
  particleGun->SetParticleEnergy(Ekin);
  G4double angle = (G4UniformRand()-0.5)*sigmaAngle;
  particleGun->SetParticleMomentumDirection
    (G4ThreeVector(sin(angle), 0., cos(angle)));
  particleGun->GeneratePrimaryVertex(anEvent);
}
```

- You can repeat this for generating more than one primary particles.

# Primary vertices and primary particles

- Primary vertices and primary particles are stored in G4Event in advance to processing an event.
  - `G4PrimaryVertex` and `G4PrimaryParticle` classes
  - These classes don't have any dependency to `G4ParticleDefinition` nor `G4Track`.
  - They will become "primary tracks" only at Begin-of-Event phase and put into a "stack"



# G4ParticleGun

- Concrete implementations of G4VPrimaryGenerator
  - A good example for experiment-specific primary generator implementation
- It shoots **one primary particle** of a **certain energy** from a **certain point** at a **certain time** to a **certain direction**.
  - Various C++ set methods are available
- Intercoms commands are also available for setting initial values
  - /gun/List List available particles
  - /gun/particle Set particle to be generated
  - /gun/direction Set momentum direction
  - /gun/energy Set kinetic energy
  - /gun/momentum Set momentum
  - /gun/momentumAmp Set absolute value of momentum
  - /gun/position Set starting position of the particle
  - /gun/time Set initial time of the particle
  - /gun/polarization Set polarization
  - /gun/number Set number of particles to be generated  
(per event)
  - /gun/ion Set properties of ion to be generated  
[usage] /gun/ion Z A Q

# Motivation for GPS

- After first simple tutorial trials, modelling sources in realistic set-up soon requires relatively more complex sources
- G4ParticleGun can be used in most cases (as in the series of examples during this tutorial), but
  - users still needs to code (C++) almost every change and
  - add related UI commands for interactive control
- Requirements for advanced primary particle modelling are often common to many users in different communities
  - E.g. uniform vertex distribution on a surface, isotropic generation, energy spectrum,...

# G4GeneralParticleSource

- A concrete implementation of G4VPrimaryGenerator
  - Suitable especially to space applications

```
MyPrimaryGeneratorAction::
```

```
    MyPrimaryGeneratorAction()
```

```
{ generator = new G4GeneralParticleSource; }
```

```
void MyPrimaryGeneratorAction::
```

```
    GeneratePrimaries(G4Event* anEvent)
```

```
{ generator->GeneratePrimaryVertex(anEvent); }
```

- Detailed description

[Geant4 GPS manual](#)

[LXR code browser \(expgps\)](#)



# Summary of GPS features

- Primary vertex can be **randomly positioned** with several options
  - Emission from point, plane,...
- **Angular emission**
  - Several distributions; isotropic, cosine-law, focused, ...
  - With some additional parameters (min/max-theta, min/max-phi,...)
- **Kinetic energy** of the primary particle can also be randomized.
  - Common options (e.g. mono-energetic, power-law), some extra shapes (e.g. black-body) or user defined
- **Multiple sources**
  - With user defined relative intensity
- Capability of event biasing (**variance reduction**).
  - By enhancing particle type, distribution of vertex point, energy and/or direction



# GPS

## Example 7

- Vertex on cylinder surface
- Cosine-law emission  
(to mimic isotropic source in space)
- Pre-defined spectrum  
(Cosmic Diffuse Gamma)

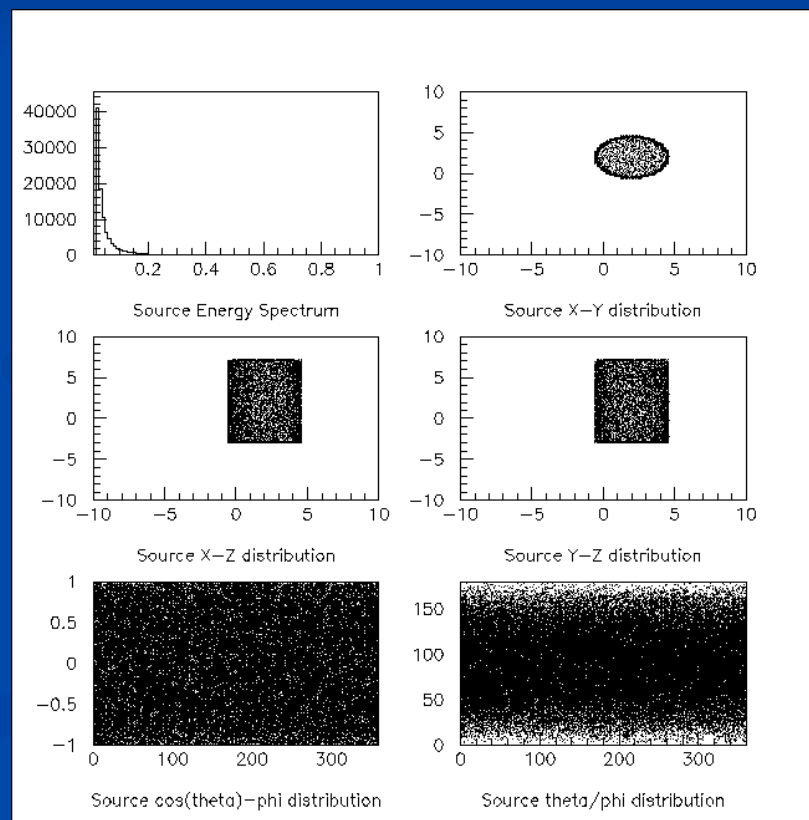
### ■ Macro

```
/gps/particle gamma
```

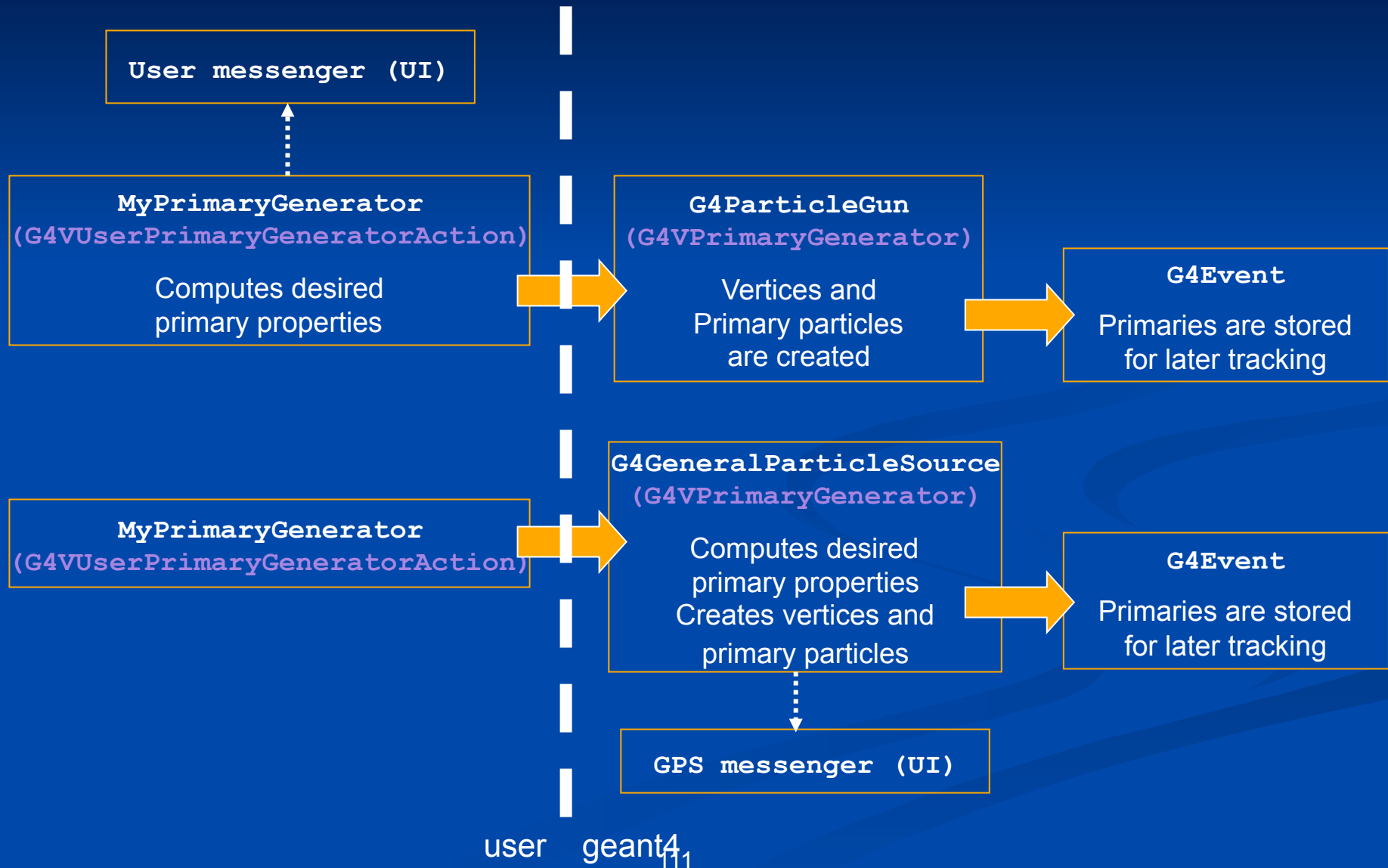
```
/gps/pos/type Surface  
/gps/pos/shape Cylinder  
/gps/pos/centre 2. 2. 2. cm  
/gps/pos/radius 2.5 cm  
/gps/pos/halfz 5. cm
```

```
/gps/ang/type cos
```

```
/gps/ene/type Cdg  
/gps/ene/min 20. keV  
/gps/ene/max 1. MeV  
/gps/ene/calculate
```



# GPS vs G4ParticleGun



# Laboratory - 9

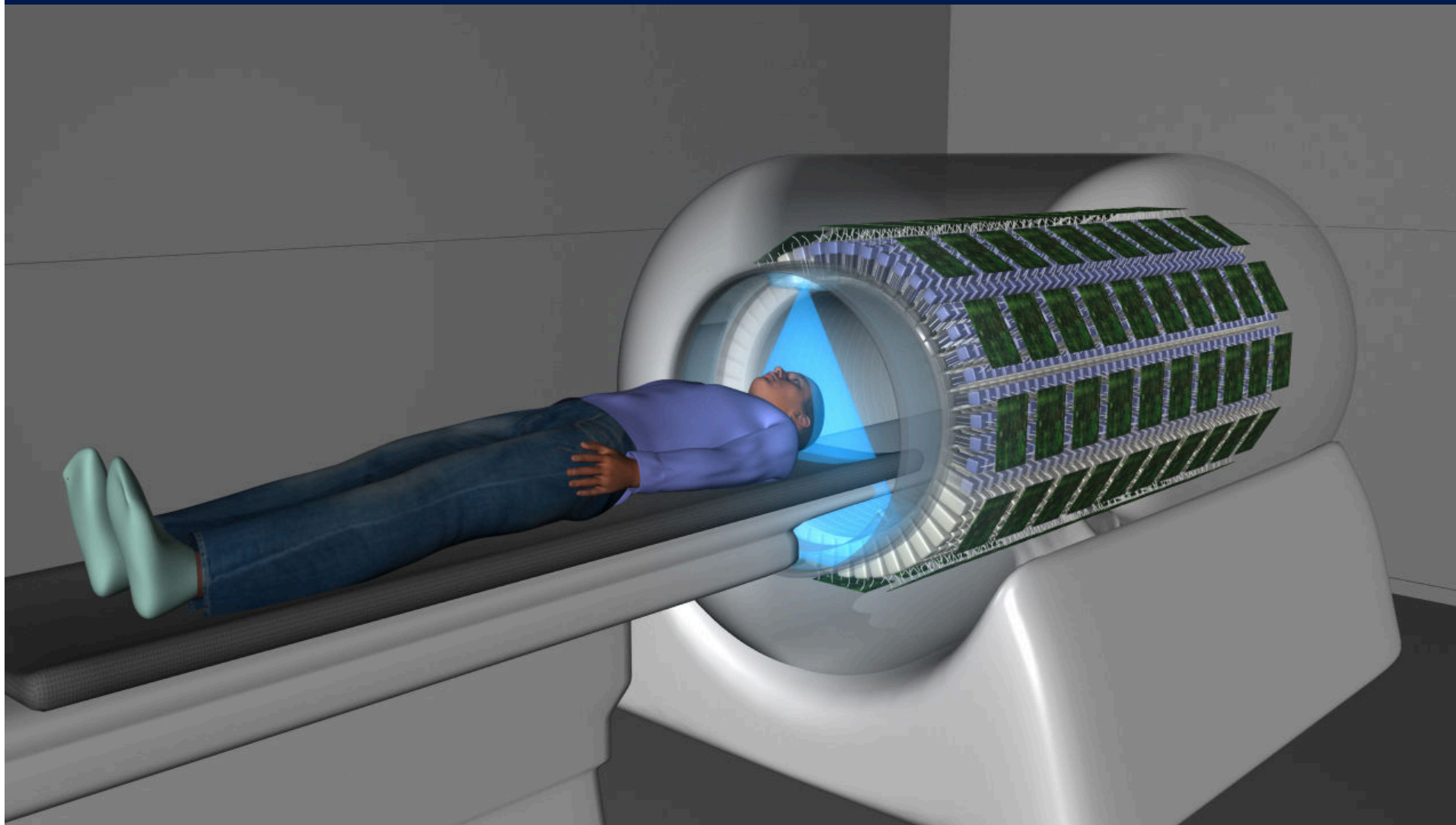
- Check the usage of particle generation method in G4 example B3

# Part 2

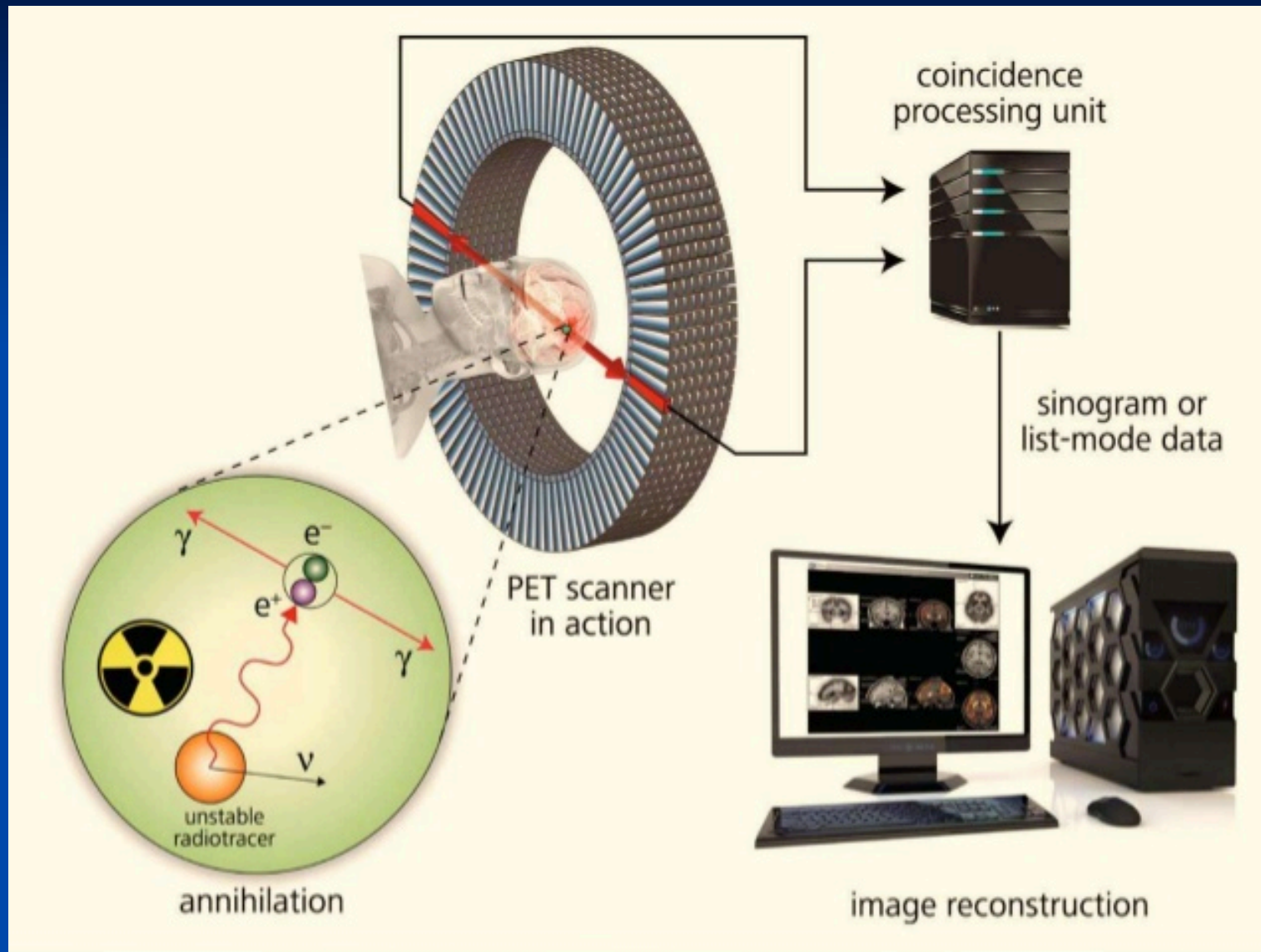
## Hands on G4 Example B3

# Analysis of example B3

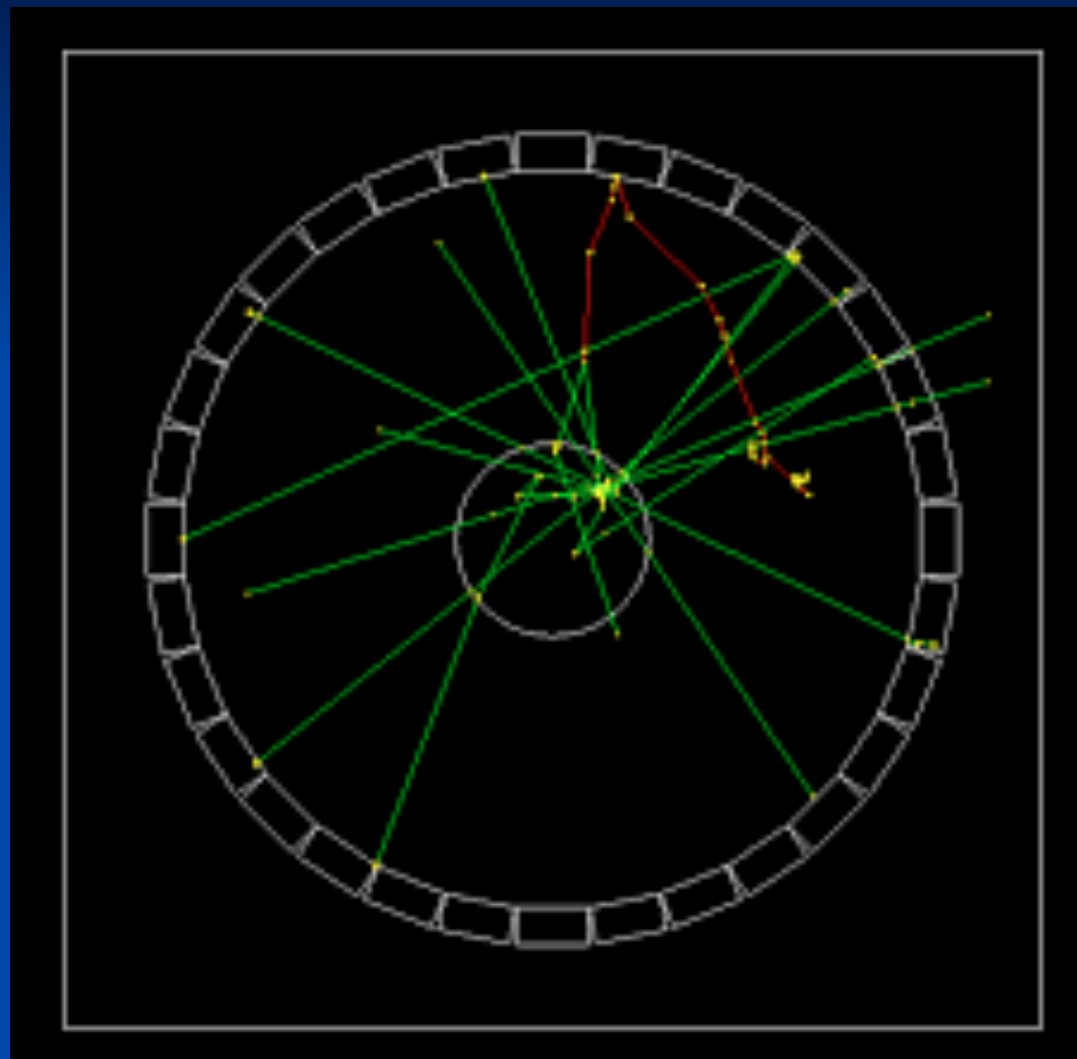
# The PET scanner



# The PET scanners



# The PET scanner





# Laboratory - 10

- What do we need to know more?

# Part 2

## Scoring in G4

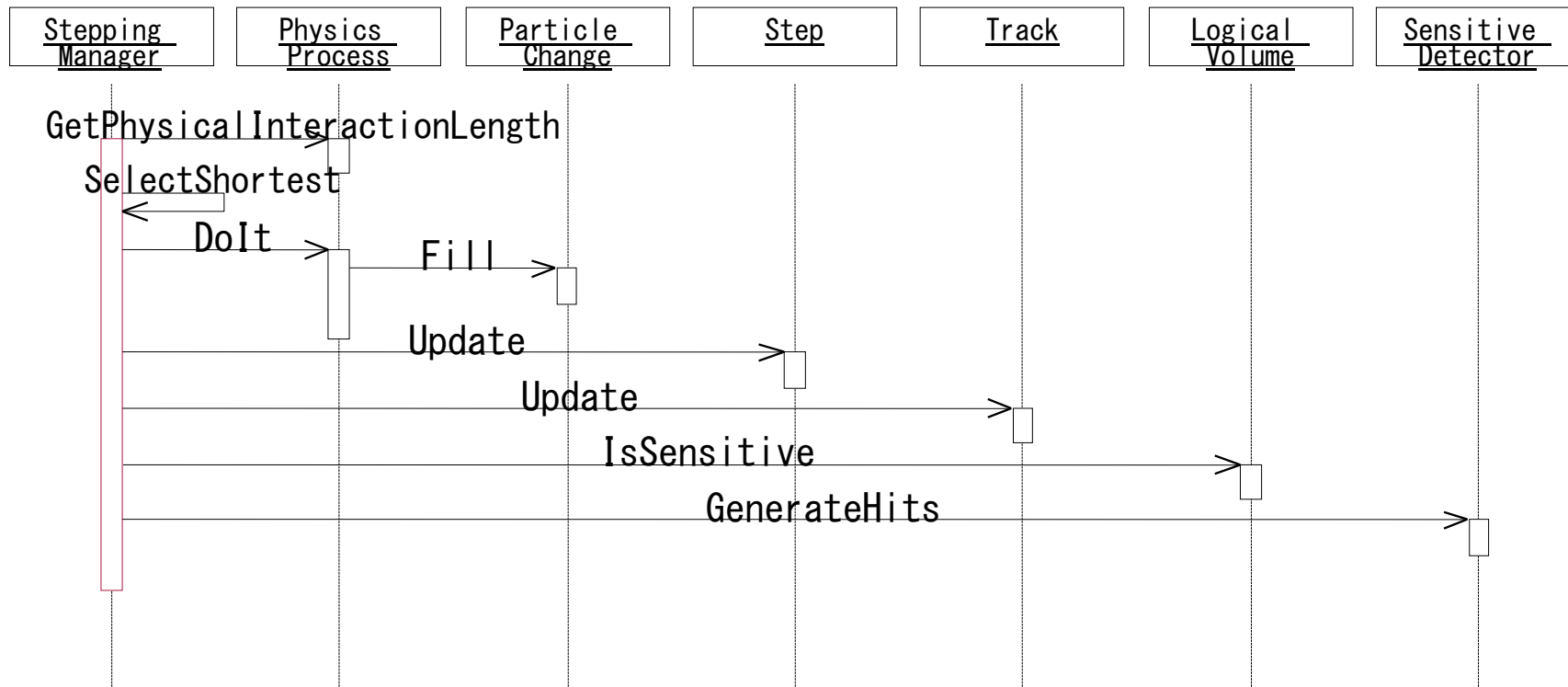
# Scoring in Geant4

# Extract useful information

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”.
  - You have to add a bit of code to **extract information useful to you**.
- There are three ways:
  - Use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
    - You have full access to almost all information
    - Straight-forward, but do-it-yourself
  - Use Geant4 scoring functionality
    - Assign **G4VSensitiveDetector** to a volume
    - **Hit** is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive (or interested) part of your detector.
    - **Hits collection** is automatically stored in G4Event object, and automatically accumulated if **user-defined Run** object is used.
    - Use user hooks (G4UserEventAction, G4UserRunAction) to get event / run summary
  - Using the **G4 primitive scorers**

# Sensitive detector

- A **G4VSensitiveDetector** object can be assigned to **G4LogicalVolume**.
- In case a step takes place in a logical volume that has a **G4VSensitiveDetector** object, this **G4VSensitiveDetector** is invoked with the **current G4Step** object.
  - You can implement your own sensitive detector classes, or use scorer classes provided by Geant4.



# Defining a sensitive detector

- Basic strategy

```
G4LogicalVolume* myLogCalor = .....;  
  
G4VSensitiveDetector* pSensitivePart =  
    new MyDetector("/mydet");  
  
G4SDManager* SDMan = G4SDManager::GetSDMpointer();  
SDMan->AddNewDetector(pSensitivePart);  
myLogCalor->SetSensitiveDetector(pSensitivePart);
```

- Each detector **object** must have a unique name.
  - Some logical volumes can share one detector object.
  - More than one detector objects can be made from one detector class **with different detector name**.
  - One logical volume cannot have more than one detector objects. But, one detector object can generate more than one kinds of hits.
    - e.g. a double-sided silicon micro-strip detector can generate hits for each side separately.

**Sensitive detector and hit**

# Sensitive detector and Hit

- Each Logical Volume can have a pointer to a sensitive detector.
  - Then this volume becomes **sensitive**.
- Hit is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive region of your detector.
- A sensitive detector creates hit(s) using the information given in G4Step object. The user has to provide his/her own implementation of the detector response.
- Hit objects, which are still the user's class objects, are collected in a G4Event object at the end of an event.



# Hit class

- Hit is a user-defined class derived from **G4VHit**.
- You can store various types information by implementing your own concrete Hit class. For example:
  - Position and time of the step
  - Momentum and energy of the track
  - Energy deposition of the step
  - Geometrical information
  - or any combination of above
- Hit objects of a concrete hit class must be stored in a dedicated collection which is instantiated from **G4THitsCollection template class**.
- The collection will be associated to a G4Event object via **G4HCofThisEvent**.
- Hits collections are accessible
  - through G4Event at the end of event.
    - to be used for analyzing an event
  - through G4SDManager during processing an event.
    - to be used for event filtering.

# Implementation of Hit class

```
#include "G4VHit.hh"
class MyHit : public G4VHit
{
    public:
        MyHit(some_arguments);
        virtual ~MyHit();
        virtual void Draw();
        virtual void Print();
    private:
        // some data members
    public:
        // some set/get methods
};

#include "G4THitsCollection.hh"
typedef G4THitsCollection<MyHit> MyHitsCollection;
```

# Sensitive Detector class

- Sensitive detector is a user-defined class derived from G4VSensitiveDetector.

```
#include "G4VSensitiveDetector.hh"
#include "MyHit.hh"
class G4Step;
class G4HCofThisEvent;
class MyDetector : public G4VSensitiveDetector
{
public:
    MyDetector(G4String name);
    virtual ~MyDetector();
    virtual void Initialize(G4HCofThisEvent*HCE);
    virtual G4bool ProcessHits(G4Step*aStep,
                               G4TouchableHistory*ROhist);
    virtual void EndOfEvent(G4HCofThisEvent*HCE);
private:
    MyHitsCollection * hitsCollection;
    G4int collectionID;
};
```

# Sensitive detector

- A **tracker** detector typically generates a **hit for every single step of every single (charged) track**.
  - A tracker hit typically contains
    - Position and time
    - Energy deposition of the step
    - Track ID
- A **calorimeter** detector typically generates a hit for every cell, and **accumulates energy deposition in each cell for all steps of all tracks**.
  - A calorimeter hit typically contains
    - Sum of deposited energy
    - Cell ID
- You can instantiate more than one objects for one sensitive detector class. Each object should have its unique detector name.
  - For example, each of two sets of detectors can have their dedicated sensitive detector objects. But, the functionalities of them are exactly the same to each other so that they can share the same class. See [examples/extended/analysis/A01](#) as an example.

# Implementation of Sensitive Detector

```
G4bool MyDetector::ProcessHits
(G4Step*aStep,G4TouchableHistory*ROhist)
{
  MyHit* aHit = new MyHit();
  ...
  // some set methods
  ...
  hitsCollection->insert(aHit);
  return true;
}
```

- This ProcessHits() method is invoked **for every steps** in the volume(s) where this sensitive detector is assigned.
- In this method, generate a hit corresponding to the current step (for tracking detector), or accumulate the energy deposition of the current step to the existing hit object where the current step belongs to (for calorimeter detector).
- Don't forget to collect geometry information (e.g. copy number) from **"PreStepPoint"**.
- Currently, returning boolean value is not used.

# G4MultiFunctionalDetector

- **G4MultiFunctionalDetector** is a concrete class derived from G4VSensitiveDetector. It should be set to a logical volume as a kind of sensitive detector.
- It takes arbitrary number of **G4VPrimitiveSensitivity** classes. By registering G4VPrimitiveSensitivity classes, you can define the scoring detector of your need.
  - Each G4VPrimitiveSensitivity class accumulates **one physics quantity** for each physical volume.
  - For example, G4PSDoseScorer (a concrete class of G4VPrimitiveSensitivity provided by Geant4) accumulates dose for each cell.
- By using G4MultiFunctionalDetector and provided concrete G4VPrimitiveSensitivity classes, you are freed from implementing sensitive detector and hit classes.



# Sensitive detector vs. primitive scorer

## Sensitive detector

- You have to implement your own detector and hit classes.
- One hit class can contain many quantities. A hit can be made for each individual step, or accumulate quantities.
- Basically one hits collection is made per one detector.
- Hits collection is relatively compact.

## Primitive scorer

- Many scorers are provided by Geant4. You can add your own.
- Each scorer accumulates one quantity for an event.
- G4MultiFunctionalDetector creates many collections (maps), i.e. one collection per one scorer.
- Keys of maps are redundant for scorers of same volume.

I would suggest to :

- ▶ Use primitive scorers
  - ▶ if you are **not** interested in recording each individual step **but** accumulating some physics quantities for an event or a run, and
  - ▶ if you do **not** have to have too many scorers.
- ▶ Otherwise, consider implementing your own sensitive detector.

# Primitive scorers

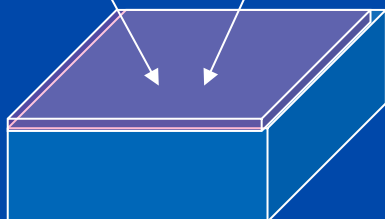


# List of provided primitive scorers

- Concrete Primitive Scorers ( See Application Developers Guide 4.4.6 )
  - Track length
    - G4PSTrackLength, G4PSPassageTrackLength
  - Deposited energy
    - G4PSEnergyDeposit, G4PSDoseDeposit, G4PSChargeDeposit
  - Current/Flux
    - G4PSFlatSurfaceCurrent, G4PSSphereSurfaceCurrent, G4PSPassageCurrent, G4PSFlatSurfaceFlux, G4PSCellFlux, G4PSPassageCellFlux
  - Others
    - G4PSMinKinEAtGeneration, G4PSNofSecondary, G4PSNofStep

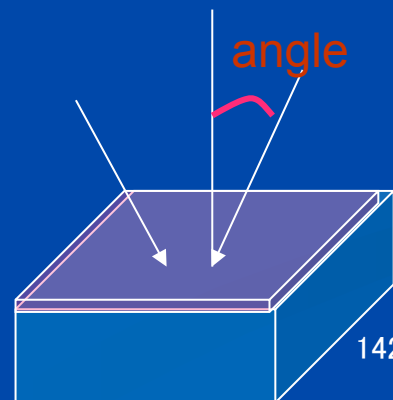
## SurfaceCurrent :

Count number of injecting particles at defined surface.



## SurfaceFlux :

Sum up  $1/\cos(\text{angle})$  of injecting particles at defined surface

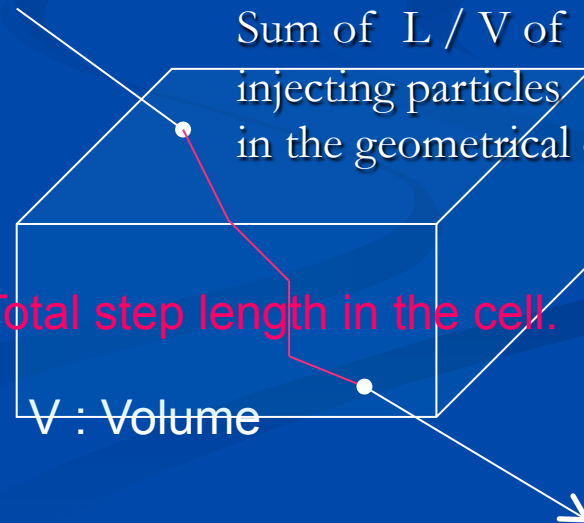


## CellFlux :

Sum of  $L / V$  of injecting particles in the geometrical cell.

L : Total step length in the cell.

V : Volume



# For example...

```
MyDetectorConstruction::Construct()  
{ ... G4LogicalVolume* myCellLog = new G4LogicalVolume(...);  
  G4VPhysicalVolume* myCellPhys = new G4PVParametrised(...);  
  G4MultiFunctionalDetector* myScorer = new  
    G4MultiFunctionalDetector("myCellScorer");  
  G4SDManager::GetSDMpointer()->AddNewDetector(myScorer);  
  myCellLog->SetSensitiveDetector(myScorer);  
  G4VPrimitiveSensitivity* totalSurfFlux = new  
    G4PSFlatSurfaceFlux("TotalSurfFlux");  
  myScorer->Register(totalSurfFlux);  
  G4VPrimitiveSensitivity* totalDose = new G4PSDoseDeposit("TotalDose");  
  myScorer->Register(totalDose);  
}
```

No need of implementing sensitive detector !

# Laboratory - 11

- Check the Scoring mechanism in example B3