Version 10.5

Introduction

John Apostolakis (CERN)
Geant4 Beginners Course

Most of the slides are obtained or adapted from slides of M.Asai (SLAC)

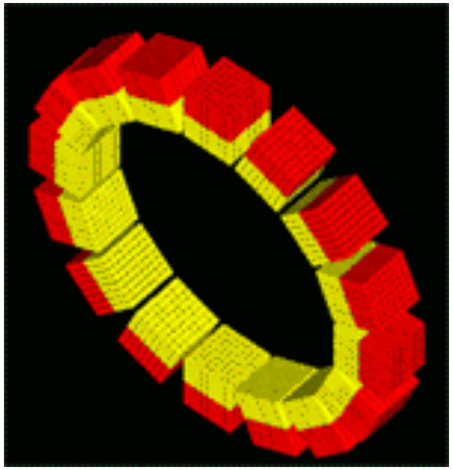
Outline

- What is Particle Transport Monte Carlo ?
- Geant4 and its components
- The kernel of Geant4 (the ‘skeleton’)
  - A tour of the Geant4 Kernel classes: from a step to a run
- Hands-on Part 1: A first run
- How to get Geant4 to do what you want (simulation your setup)
  - How to ‘keep’ the information you need

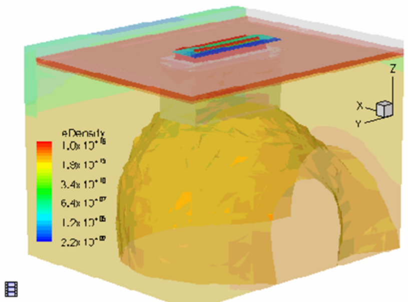
What is particle transport?

- It is a way to estimate the effects of radiation in a particular region. Given
  - a radiation source or **beam**,
  - a model of the geometry of a setup or detector
  - a volume or region in which to measure
- The simplest type of task is to estimate
  - **Energy deposition** in volume (e- displaced) and its variance
  - Dose / volume - weighted by its biological effect
  - Fluxes, e.g. of neutrons (=> nuclear reactions) in a particular region

  and similar ‘first order’ observable (with estimated errors.)
- It can also estimate complicated observables:
  - distributions of energy deposition, dose, .. Including width
  - **correlations** between observables or derived quantities - e.g. coincidence of gammas (PET)



Courtesy of GATE



Courtesy of R. Reid, Vanderbilt Univ.

Version 10.5

Monte Carlo Particle Transport

Slides adapted from “Introduction to detector simulation” by M. Asai & D. Wright (SLAC)

Monte Carlo method : definition

- The Monte Carlo method is a stochastic method for numerical integration.

- Generate N random "points" $\vec{x}_i$ in the problem space
- Calculate the "score" $f_i = f(\vec{x}_i)$ for the N "points"
- Calculate

$$\langle f \rangle = \frac{1}{N}\sum_{i=1}^{N} f_i, \quad \langle f^2 \rangle = \frac{1}{N}\sum_{i=1}^{N} f_i^2$$

- According to the Central Limit Theorem, for large N $\langle f \rangle$ will approach the true value $\bar{f}$. More precisely,

$$p(\langle f \rangle) = \frac{\exp\left[-(\langle f \rangle - \bar{f})^2/2\sigma^2\right]}{\sqrt{2\pi}\sigma}, \quad \sigma^2 = \frac{\langle f^2 \rangle - \langle f \rangle^2}{N-1}$$

Monte Carlo methods for radiation transport

- Fermi (1930): random method to calculate the properties of the newly discovered neutron
- Manhattan project (40's): simulations during the initial development of thermonuclear weapons. von Neumann and Ulam coined the term "Monte Carlo"

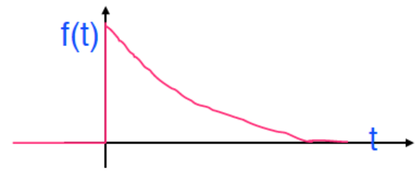
- Metropolis (1948) first actual Monte Carlo calculations using a computer (ENIAC)
- Berger (1963): first complete coupled electron-photon transport code that became known as ETRAN
- Exponential growth since the 1980's with the availability of digital computers

Simplest case – decay in flight (1)

- Suppose an unstable particle of life time *t* has initial momentum *p* (→ velocity *v*).
  - Distance to travel before decay : $d = t\,v$
- The decay time *t* is a random value with probability density function

$$f(t) = \frac{1}{\tau}\exp\left(-\frac{t}{\tau}\right) \quad t \ge 0$$

τ is the mean life of the particle



- the probability that the particle decays at time *t* is given by the cumulative distribution function *F* which is itself is a random variable with uniform probability on [0,1]

$$r = F(t) = \int_{-\infty}^{t} f(u)du$$

- Thus, having a uniformly distributed random number *r* on [0,1], one can sample the value *t* with the probability density function *f*(*t*).

$$t = F^{-1}(\,r\,) = -\tau \ln(\,1 - r\,) \qquad 0 \le r < 1$$

Simplest case – decay in flight (2)

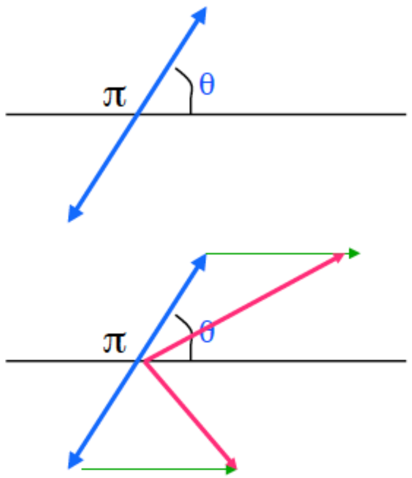
- When the particle has traveled the $d = t\,v$, it decays.
- Decay of an unstable particle itself is a random process → Branching ratio
  - For example:

$\pi^+ \rightarrow \mu^+ \nu_\mu$	(99.9877 %)
$\pi^+ \rightarrow \mu^+ \nu_\mu \gamma$	(2.00×10^{-4} %)
$\pi^+ \rightarrow e^+ \nu_e$	(1.23×10^{-4} %)
$\pi^+ \rightarrow e^+ \nu_e \gamma$	(7.39×10^{-7} %)
$\pi^+ \rightarrow e^+ \nu_e \pi^0$	(1.036×10^{-8} %)
$\pi^+ \rightarrow e^+ \nu_e e^+ e^-$	(3.2×10^{-9} %)

- Select a decay channel by shooting a random number
- In the rest frame of the parent particle, rotate decay products in $\theta\,[0,\pi)$ and $\phi\,[0,2\pi)$ by shooting a pair of random numbers

$$d\Omega = \sin\theta\, d\theta\, d\phi$$

$$\theta = \cos^{-1}(r_1), \quad \phi = 2\pi \times r_2 \qquad 0 \le r_1, r_2 < 1$$



- Finally, Lorentz-boost the decay products
- You need at least 4 random numbers to simulate one decay in flight

Simplest case – decay in flight (2)

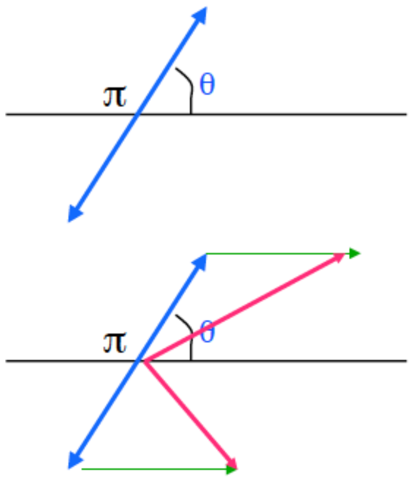
- When the particle has traveled the *d* = *t v*, it decays.
- Decay of an unstable particle itself is a random process → Branching ratio
  - For example:

$\pi^+ \rightarrow \mu^+ \nu_\mu$	(99.9877 %)
$\pi^+ \rightarrow \mu^+ \nu_\mu \gamma$	(2.00×10^{-4} %)
$\pi^+ \rightarrow e^+ \nu_e$	(1.23×10^{-4} %)
$\pi^+ \rightarrow e^+ \nu_e \gamma$	(7.39×10^{-7} %)
$\pi^+ \rightarrow e^+ \nu_e \pi^0$	(1.036×10^{-8} %)
$\pi^+ \rightarrow e^+ \nu_e e^+ e^-$	(3.2×10^{-9} %)

- Select a decay channel by shooting a random number
- In the rest frame of the parent particle, rotate decay products in $\theta\,[0,\pi)$ and $\phi\,[0,2\pi)$ by shooting a pair of random numbers

$$d\Omega = \sin\theta\, d\theta\, d\phi$$

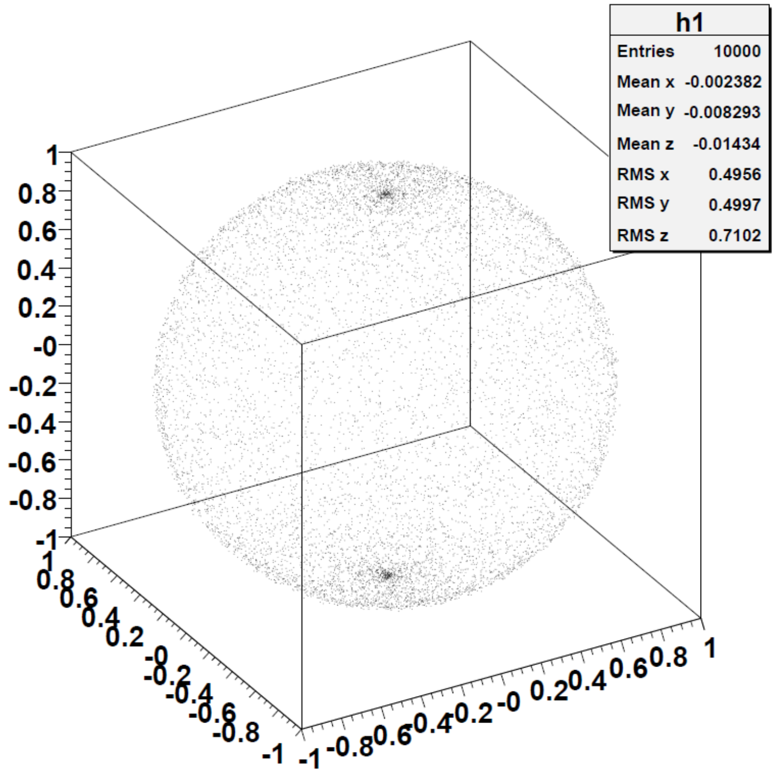
$$\theta = \cos^{-1}(r_1), \quad \phi = 2\pi \times r_2 \qquad 0 \leq r_1, r_2 < 1$$



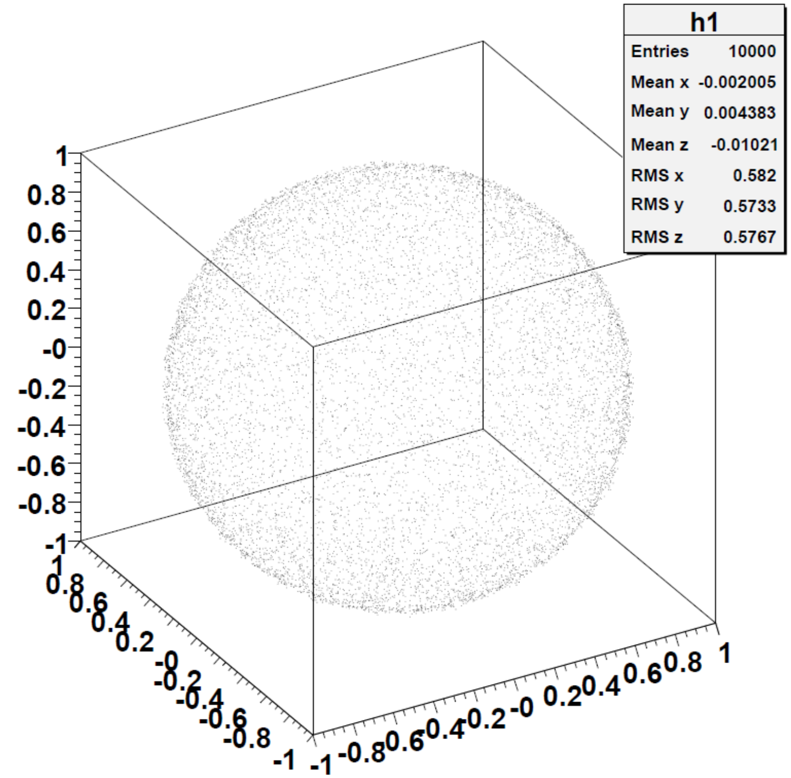
- Finally, Lorentz-boost the decay products
- You need at least 4 random numbers to simulate one decay in flight

Evenly distributed points on a sphere

$\theta = \pi \times r_1, \ \phi = 2\pi \times r_2$
$0 \le r_1, r_2 < 1$



$\theta = \cos^{-1}(r_1), \ \phi = 2\pi \times r_2$
$0 \le r_1, r_2 < 1$



$d\Omega = \sin\theta \, d\theta \, d\phi$

GEANT4 A SIMULATION TOOLKIT

Version 10.5

Geant4: the briefest history & tour

Slides adapted from M. Asai (SLAC)

Geant4 History

R&D phase (RD44)

- Early discussions, e.g. at CHEP 1994 @ San Francisco
  - CERN & Japan seeded R&D proposal
- Dec '94 – R&D project start
- Dec '98 - First Geant4 public release - version "0.0"

Production phase

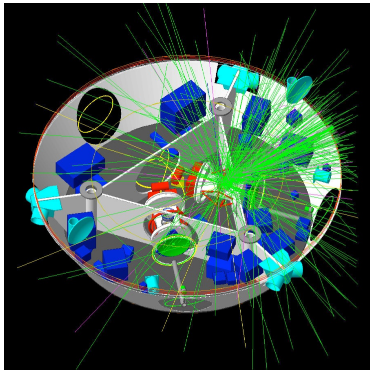
- 1999: Used to simulate X-ray mission in ESA's XMM mission
- 2001: Babar (SLAC) uses Geant4 in production
- 2004: ATLAS, CMS & LHCb start using Geant4 in production

- *Several major architectural revisions*
  - *E.g. STL migration, "cuts per region", parallel worlds,* ***multithreading***

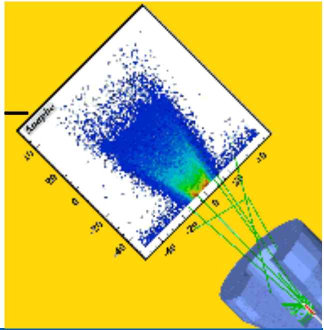
- Dec 2013 – Geant4 version 10.0 release – first with multi-threading
- Dec 2017 – Geant4 version 10.4 release
  - May 25th, '18 - Geant4 10.4-patch02 release
- Dec 2018 – Geant4 version 10.5 release ← Current version
- We currently provide one public release every year (Nov/Dec)
  - And one preview 'beta' release (June)

Essential elements of Geant4

- **Kernel** : Manages 'mandatory' parts, lets the physics & recording happen
  - *Geometry* & materials
  - *Tracks*
  - Events (collisions or primaries)
  - Runs



- **Physics** processes – cross sections and **final state generation**
  - models for electromagnetic, hadronic, …
  - assembled into coherent 'physics lists' for use in one or more application areas
- **Auxiliary** parts
  - User interface for **control** – communicating with the kernel (& the rest of G4)
  - **Visualization** – interfaces and concrete implementations
  - Interfaces for **input** of geometry, materials ('persistency')
  - Record keeping what the user requests (**hits** = energy deposit, flux, ..) – interfaces and examples



Geant4 – A Simulation Toolkit

http://www.geant4.org/

S. Agostinelli et al.
Geant4: a simulation toolkit
NIM A, vol. 506, no. 3, pp. 250-303, 2003

J. Allison et al.
Geant4 Developments and Applications
IEEE Trans. Nucl. Sci., vol. 53, no. 1, pp. 270-278, 2006

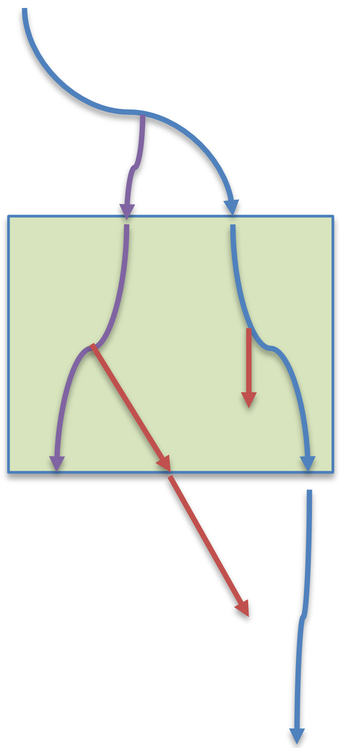
Version 10.5

Basic concepts and kernel structure

Slides adapted from M. Asai (SLAC)

Terminology (jargon)

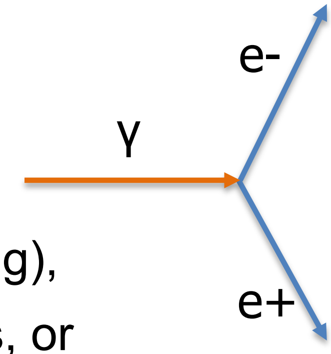
- Step point (←→ trajectory point)
- Step
- Track (←→ Trajectory)
- Event
- Run



- Process
  - At rest, along step, post step
- Cut = production threshold
- Sensitive detector, score, hit, hits collection

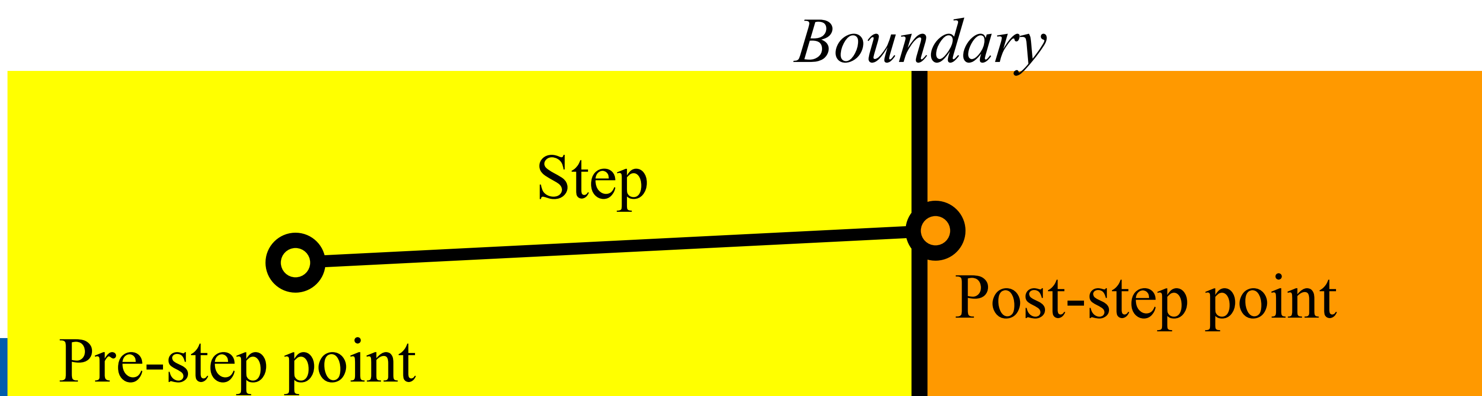
Track in Geant4

- *Track* is a snapshot of a particle. ($\vec{x}$, $\vec{p}$, PDG, $\vec{\sigma}$, q, ..)
  - It's physical quantities represent the current 'instant' in the simulation. It does not record previous quantities.
  - Step is the "delta" information of a track. Track is not a collection of steps. Instead, a track is updated in a series steps.
- Each *Track* object **disappears** (is deleted) when it either
  - leaves the outermost ('world') volume,
  - disappears in an interaction (e.g. by decay or inelastic scattering),
  - it's kinetic energy becomes zero and it has no "AtRest" process, or
  - the user decides to kill it ('artificially').
- All tracks disappear. None persist at the end of event.
  - To record tracks, you must use objects of a *trajectory* class.
- ***G4TrackingManager*** manages the processing a track, each one represented by an object of the ***G4Track*** class.
- ***G4UserTrackingAction*** is the optional user hook.



Step in Geant4

- A *Step* has two points and represents the "delta" information of a particle (energy loss over the step, time-of-flight during by the step, etc.).
- During simulation *Point* knows the volume(s) in which it belongs (& its material).
- If 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 such a *Step* knows materials of two volumes, *boundary processes* (such as reflection, refractions and transition radiation) can be simulated.
- A step is represented by the ***G4Step*** class
- The ***G4SteppingManager*** class manages processing of steps (and update tracks ..) and also calls the ***G4UserSteppingAction***, an optional user hook.



Event in Geant4

- An *event* is the basic unit of simulation in Geant4, represents a set of tracks
  - At its beginning primary tracks are generated (and pushed onto a stack).
  - One 'track' at a time is popped from the stack and it is "tracked"
    - Any resulting secondary tracks are pushed back onto the stack.
    - This "tracking" lasts as long as the stack has a track.
  - When the stack becomes empty, it's the end of processing that event.
- An object of the ***G4Event*** class represents an event. After its processing it contains few objects:
  - List of primary vertices and particles (its input)
  - Hits and Trajectory collections (its output.)
- The ***G4EventManager*** class coordinates the processing of an event.
- A user can create a ***G4UserEventAction*** to hook into an event's start & end.

Run in Geant4

- In analogy with real experiments, a ***G4Run*** starts with "Beam On".
- By definition within a run, the user cannot change
  - detector setup
  - settings of physics processes
- Typically a run consists of one event loop. (Events are treated one after another.)
- At a run's start geometry structures and physics configurations are prepared
  - the geometry is optimized for navigation,
  - cross-section tables are calculated for the setup's materials, ...
- ***G4RunManager*** class manages processing a run, a run is represented by a ***G4Run*** object (or a user-defined class derived from G4Run.)
  - A run class may have a summary results of the run.
- A user can create a subclass of ***G4UserRunAction*** to hook into its start & end.

Hands-on: Exercise 0

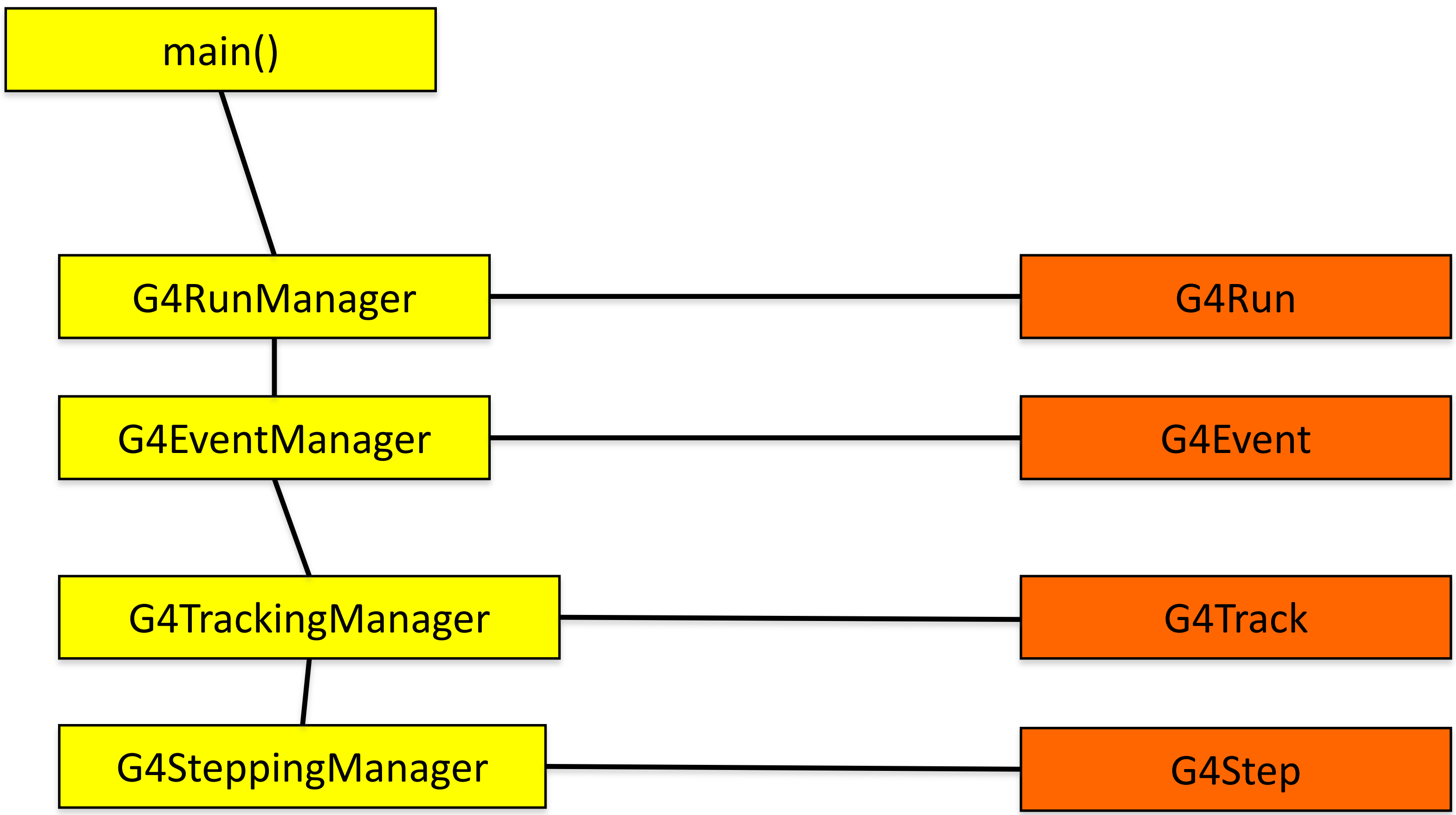
- We will make the first steps:
  - check that your installations work
  - look at one example
- In particular
  - compile example B1
  - run it (B1)
  - take a short look at 1-2 of its files

```
cd
mkdir g4work
cd g4work

echo $G4COMP
ls $G4COMP/

mkdir taskA
cd taskA

cp -r $G4EXAMPLES/basic/B1   ./
ls
less README
nedit exampleB1.cc  &

mkdir build
cd build
cmake -DGeant4_DIR=$G4COMP/  ../
make

./exampleB1
```

Some questions generated by this exercise

- How did we record the information to make the pictures ?
- How did we create the initial tracks ?
- What types of particles can be simulated by Geant4 ?
- How did we create the geometry we saw ?
- …

Key parts of a Geant4 Program – first view, from the kernel



Trajectory and trajectory point

- How to keep the transient information in a G4Track?
- G4Trajectory is the class which copies some of G4Track information. G4TrajectoryPoint is the class which copies some of G4Step information.
  - G4Trajectory has a vector of G4TrajectoryPoint.
  - At the end of event processing, G4Event has a collection of G4Trajectory objects.
    - /tracking/storeTrajectory must be set to 1.
- Keep in mind the distinction.
  - G4Track ←→ G4Trajectory, G4Step ←→ G4TrajectoryPoint
- Given G4Trajectory and G4TrajectoryPoint objects persist till the end of an event, you should be careful not to store too many trajectories.
  - E.g. avoid for high energy EM shower tracks.
- G4Trajectory and G4TrajectoryPoint store only the minimum information.
  - You can create your own trajectory / trajectory point classes to store information you need. G4VTrajectory and G4VTrajectoryPoint are base classes.

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.

Tracking and processes

- The Geant4 tracking ‘loop’ is general.
  - It is ***independent*** of the particle type,
  - It obtains the list the applicable physics processes from each particle (type)
  - It gives the chance to each process in turn:
    - To contribute to determining the step length
    - To contribute any possible changes in physical quantities of the track
    - To generate secondary particles
    - To suggest changes in the state of the track
      - e.g. to suspend, postpone or kill it.
- This generality has strengths (adaptability) and costs (performance.)

Extract useful information

- Given geometry, physics and primary track generation, Geant4 does all the physics simulation "silently".
  - You need to ask/get it to record the information useful to you.
- There are three ways:
  - Built-in scoring (via UI commands)
    - Most common physics quantities are available.
  - Assign G4VSensitiveDetector to a volume to generate "hit".
    - with user hooks (*G4UserEventAction*, *G4UserRunAction*) access or write out individual hits or sums of hits' energies per event or run.
  - Use scorers in tracking volume(s)
    - Create scores for each event
    - Create own Run class to accumulate scores
- Or score using the user hooks (*G4User Tracking* & *Stepping Action*, ..)
  - You have full access to almost all information
  - Straight-forward in sequential mode, but do-it-yourself (not recommended.)

Writing to cout / cerr - via G4cout, G4cerr !

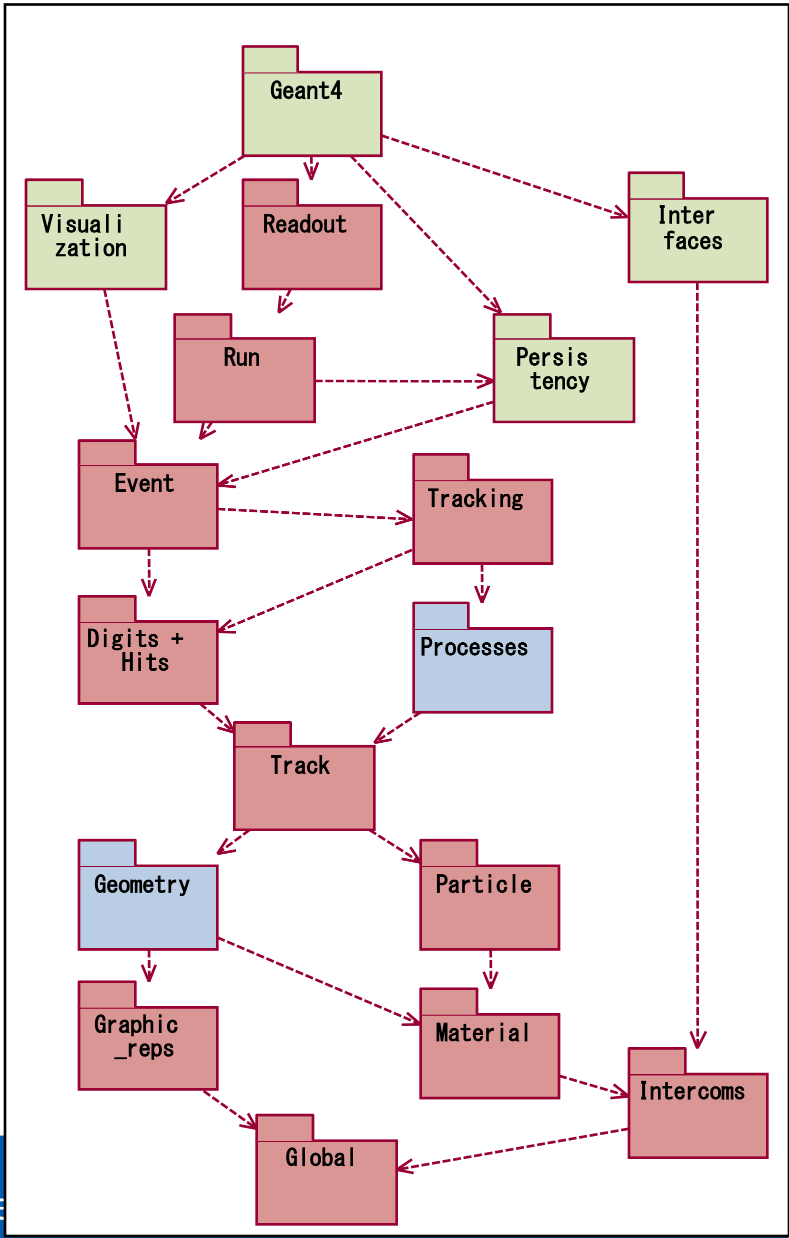
- G4cout and G4cerr are *ostream* objects defined by Geant4.
  - G4endl is also provided.

```
G4cout << "Hello Geant4!" << G4endl;
```

- Some GUIs buffer these output streams to display print-out in another window or provide storing / editing functionality.
  - The user is asked to avoid using *std::cout* and *std::cerr*.
- We recommend also that the user also avoids using the 'raw' *std::cin* for input.
  - Instead we suggest to use the G4 user-defined commands which tie into the Geant4 User Interface system (provided by the intercoms category).
- You can use 'ordinary' file I/O – GEant4 will not interfere with it.

Geant4 kernel

- Geant4 consists of 17 categories.
  - Independently developed and maintained by a Working Group each.
  - Interfaces between categories (e.g. top level design) are maintained by the global architecture WG.
- Geant4 Kernel
  - Handles run, event, track, step, hit, trajectory.
  - Provides frameworks of geometrical representation and physics processes.



Version 10.5

User classes

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

- main()
  - Geant4 does not provide *main().*
- Initialization classes
  - Use G4RunManager::SetUserInitialization() to define.
  - Invoked at the initialization
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
    - G4VUserActionInitialization
- Action classes
  - Instantiate in your G4VUserActionInitialization.
  - Invoked during an event loop
    - G4VUserPrimaryGeneratorAction
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction

Note : classes written in red are mandatory.

The main program

- Geant4 does not provide a *main*().
- In your *main(),* you have to
  - Construct G4RunManager (sequential mode) or G4MTRunManager (multithreaded mode)
  - Set user mandatory initialization classes to RunManager
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
    - G4VUserActionInitialization
- You can define VisManager, (G)UI session, optional user action classes, and/or your persistency manager in your *main()*.

Describe your detector

- Derive your own concrete class from G4VUserDetectorConstruction abstract base class.
- In the virtual method *Construct()*, that is invoked in the master thread (and in sequential mode)
  - Instantiate all necessary materials
  - Instantiate volumes of your detector geometry
- In the virtual method *ConstructSDandField()*, that is invoked in each worker thread (and in sequential mode)
  - Instantiate your sensitive detector classes and field classes and set them to the corresponding logical volumes and field managers, respectively.

Select physics processes

- Geant4 does not have any default particles or processes.
  - Even for the particle transportation, you have to define it explicitly.
- Derive your own concrete class from G4VUserPhysicsList abstract base class.
  - Define all necessary particles
  - Define all necessary processes and assign them to proper particles
  - Define cut-off ranges applied to the world (and each region)
- Primarily, the user's task is choosing a "pre-packaged" physics list, that combines physics processes and models that are relevant to a typical application use-cases.
  - If "pre-packaged" physics lists do not meet your needs, you may add or alternate some processes/models.
  - If you are brave enough, you may implement your physics list.

Generate primary event

- This is the only mandatory user action class.
- Derive your concrete class from G4VUserPrimaryGeneratorAction abstract base class.
- Pass a G4Event object to one or more primary generator concrete class objects which generate primary vertices and primary particles.
- Geant4 provides several generators in addition to the G4VPrimaryParticlegenerator base class.
  - G4ParticleGun
  - G4HEPEvtInterface, G4HepMCInterface
    - Interface to /hepevt/ common block or HepMC class
  - G4GeneralParticleSource
    - Define radioactivity

Optional user action classes

- All user action classes, methods of which are invoked during "Beam On", must be constructed in the user's *main*() and must be set to the RunManager.
- G4UserRunAction
  - G4Run* GenerateRun()
    - Instantiate user-customized run object
  - void BeginOfRunAction(const G4Run*)
    - Define histograms
  - void EndOfRunAction(const G4Run*)
    - Analyze the run
    - Store histograms
- G4UserEventAction
  - void BeginOfEventAction(const G4Event*)
    - Event selection
  - void EndOfEventAction(const G4Event*)
    - Output event information

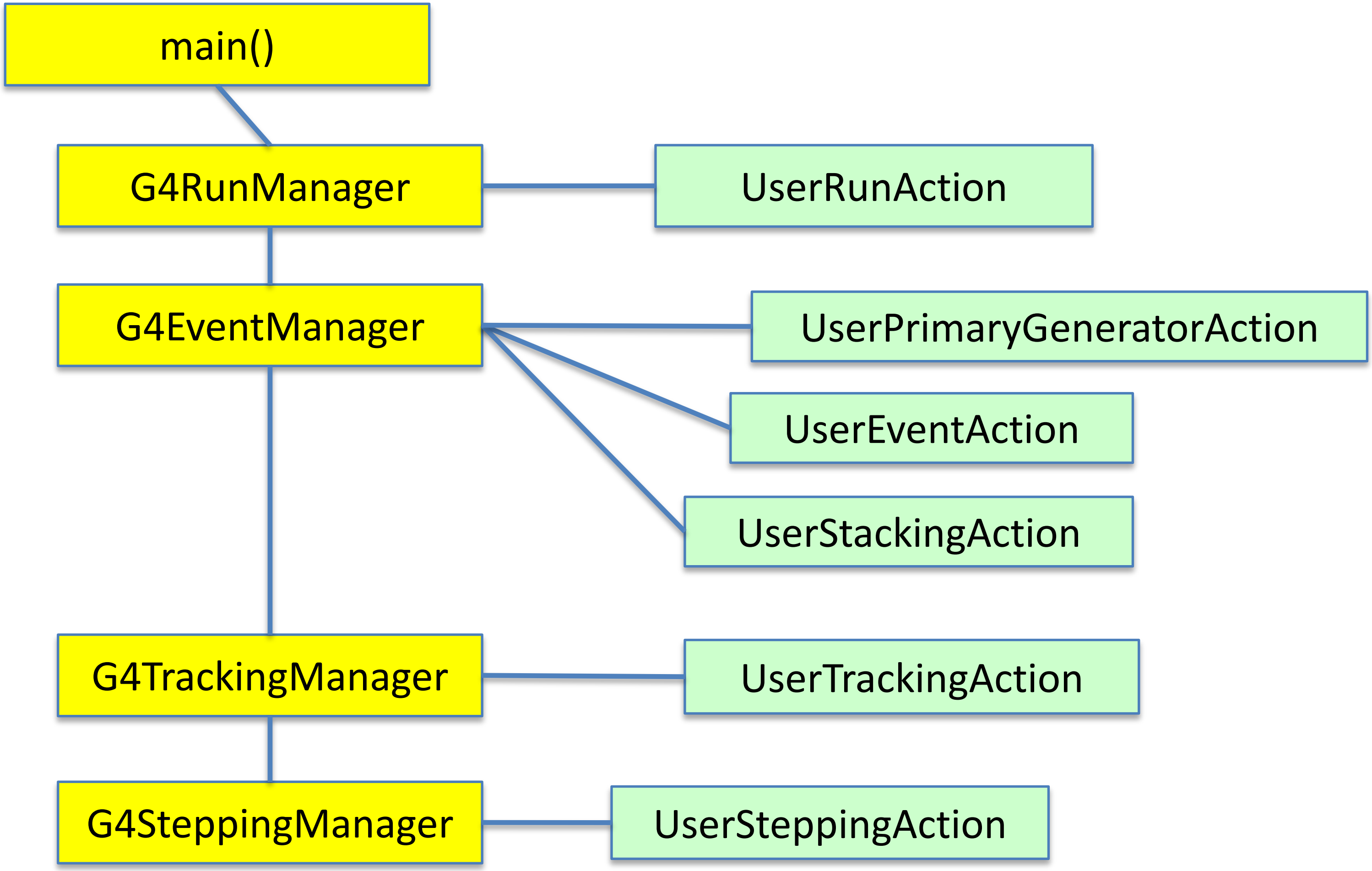
Optional user action classes

- G4UserStackingAction
  - void PrepareNewEvent()
    - Reset priority control
  - G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)
    - Invoked every time a new track is pushed
    - Classify a new track -- priority control
      - Urgent, Waiting, PostponeToNextEvent, Kill
  - void NewStage()
    - Invoked when the Urgent stack becomes empty
    - Change the classification criteria
    - Event filtering (Event abortion)

Optional user action classes

- G4UserTrackingAction
  - void PreUserTrackingAction(const G4Track*)
    - Decide trajectory should be stored or not
    - Create user-defined trajectory
  - void PostUserTrackingAction(const G4Track*)
    - Delete unnecessary trajectory
- G4UserSteppingAction
  - void UserSteppingAction(const G4Step*)
    - Kill / suspend / postpone the track
    - Draw the step (for a track not to be stored as a trajectory)

Sequential mode



What you need to know … and where you can learn it

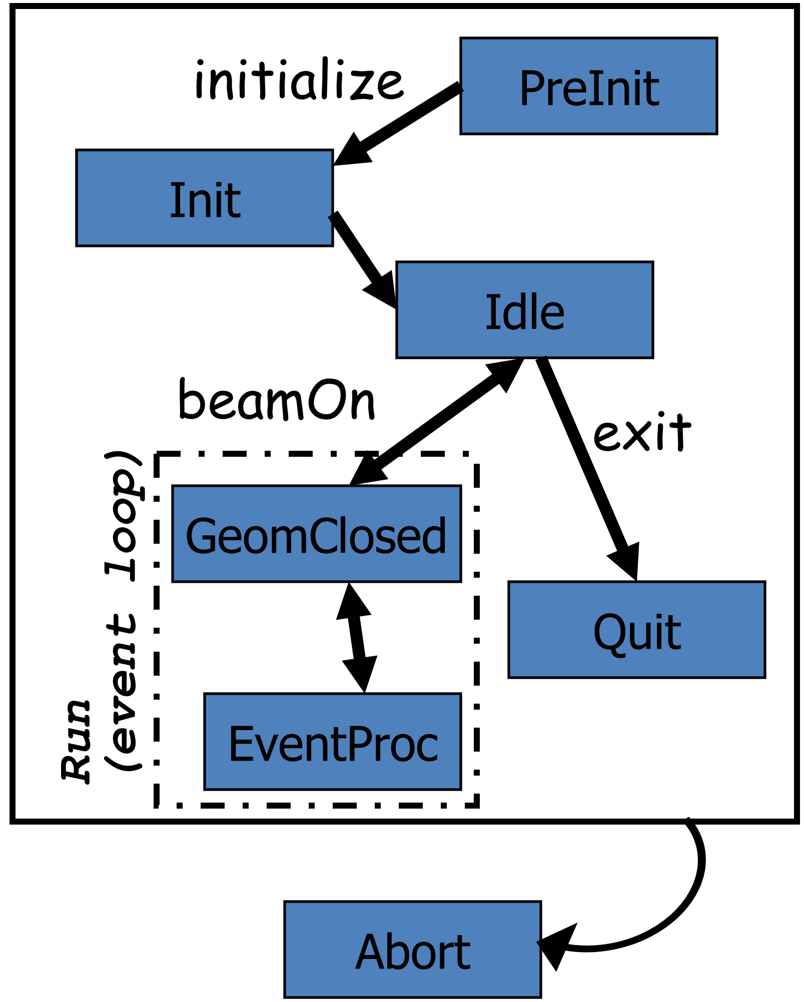
- Define material and geometry
  - ➔ G4VUserDetectorConstruction
    - Material and Geometry lectures
- Select appropriate particles and processes and define production threshold(s)
  - ➔ G4VUserPhysicsList
    - Physics lectures
- Instantiate user action classes
  - ➔ G4VUserActionInitialization
    - Hands-on
- Define the way of primary particle generation
  - ➔ G4VUserPrimaryGeneratorAction
    - Primary particle lecture
- Define the way to extract useful information from Geant4
  - ➔ G4VUserDetectorConstruction, G4UserEventAction, G4Run, G4UserRunAction
  - ➔ G4SensitiveDetector, G4VHit, G4VHitsCollection
    - Scoring lectures

Version 10.5

Additional concepts

Geant4 as a state machine

- Geant4 has seven application states.
  - G4State_PreInit
    - Initial condition
  - G4State_Init
    - During initialization
  - G4State_Idle
    - Ready to start a run
  - G4State_GeomClosed
    - Geometry is optimized and ready to process an event
  - G4State_EventProc
    - An event is processing
  - G4State_Quit
    - (Normal) termination
  - G4State_Abort
    - A fatal exception occurred and program is aborting



Note: Toggles between GeomClosed and EventProc occur for each thread asynchronously in multithreaded mode.