Geant4 internal Classes and Objects

User Action & Information Classes

Gunter Folger / CERN

MC-PAD, DESY/Hamburg

28-30 January 2010

Contents

- Internal Objects
  - Run and Event
  - Track and Step
  - StepPoint
  - Dynamic Particle
- UserAction classes
  - Run and Event
  - Track and Step
- Track Stack management
- UserInformation classes
  - G4VUserEventInformation
  - G4VUserTrackInformation
  - G4VUserPrimaryVertexInformation
  - G4VUserPrimaryParticleInformation
  - G4VUserRegionInformation

Acknowledgement

- Most of the slides shown were originally created by Makoto Asai / SLAC
- I wish to thank Makoto for allowing me to re-use his material.
- Without his expertise in the topics covered I would not have managed.

Introduction (1)
Extract information from G4 internal objects

- Simulation is successively split into
- Run consists of
- Event(s), consists of
- Particle(s) transported in
- Steps through detector setup,
- depositing energy (ionization),
- and creating secondaries

- Corresponding / related Objects
- G4RunManager, G4Run
- G4Event
- G4Track, G4DynamicParticle
- G4Step, G4StepPoint
- G4Trajectory
- G4Stack

Introduction (2)

- User at each moment has possibility to take control or access information via UserAction classes
  - G4UserRunAction Actions for each Run
  - G4UserEventAction Actions for each Event
  - G4UserTrackingAction Actions for each Track
  - G4UserSteppingAction Actions for each Step
  - G4UserStackingAction Tracks Stack management

Introduction (3)

- User can replace Geant4 classes by providing his own classes derived from the base classes:
  - G4Run
  - G4Trajectory
  - G4VTrajectoryPoint
- User can attach optional User Information classes to
  - G4Event
  - G4Track
  - G4PrimaryVertex
  - G4Region

Terminology (jargons)

- Run, event, track, step, step point
- Track ←→ trajectory,
  Step ←→ trajectory point
- Process
  - At rest, along step, post step

Geant4 internal classes

and corresponding

User Action classes

RunManager in Geant4

- G4RunManager class manages processing a run
  - Must be created by user
  - May be user derived class
  - Must be singleton
- User must register in RunManager using
  - SetUserInitialization() method
    - Geometry
    - Physics
  - SetUserAction() method
    - Event generator
  - Optional UserAction objects

Run in Geant4

- Run is a collection of events
  - A run consists of one event loop
  - Starts with a /run/beamOn command.
- Within a run, conditions do not change, i.e. the user cannot change
  - detector setup
  - settings of physics processes
- At the beginning of a run, geometry is optimized for navigation and cross-section tables are calculated according to materials appear in the geometry and the cut-off values defined.
- Run is represented by G4Run class or a user-defined class derived from G4Run.
  - A run class may have a summary results of the run.
- G4RunManager is the manager class
- G4UserRunAction is the optional user hook.

Optional User **Run** Action Class

- G4UserRunAction
  - G4Run* GenerateRun()
    - Instantiate user-customized run object
  - void BeginOfRunAction(const G4Run*)
    - Define histograms
  - void EndOfRunAction(const G4Run*)
    - Analyze the run
    - Store histograms

Event in Geant4

- An event is the basic unit of simulation in Geant4.
- At beginning of processing, primary tracks are generated. These primary tracks are pushed into a stack.
- A track is popped up from the stack one by one and "tracked". Resulting secondary tracks are pushed into the stack.
  - This "tracking" lasts as long as the stack has a track.
- When the stack becomes empty, processing of one event is over.
- G4Event class represents an event. It has following objects at the end of its (successful) processing.
  - List of primary vertices and particles (as input)
  - Hits and Trajectory collections (as output)
- G4EventManager class manages processing an event.
- G4UserEventAction is the optional user hook.

Optional User Event Action Class

- G4UserEventAction
  - void BeginOfEventAction(const G4Event*)
    - Event selection
      - Using information from event generator, vertices, primary particles
    - Optionally attach G4VUserEventInformation object
  - void EndOfEventAction(const G4Event*)
    - Output event information
    - Analyse event
      - Access to hits collection via G4Event::GetHCofThisEvent()
      - Acces digitisation collection via G4Event:: GetDCofThisEvent()
    - Fill histograms

Track in Geant4

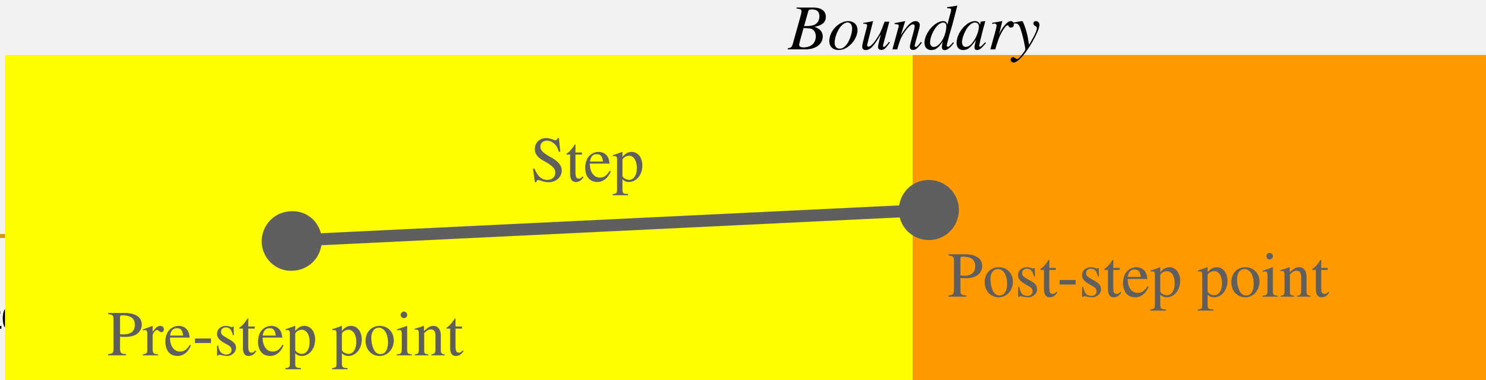
- Track is a snapshot of a particle.
  - It has physical quantities of current instance only. It does not record previous quantities.
  - Step is a "delta" information to a track. Track is not a collection of steps. Instead, a track is being updated by steps.
- Track object is deleted when
  - it goes out of the world volume,
  - it disappears (by e.g. decay, inelastic scattering),
  - it goes down to zero kinetic energy and no "AtRest" additional process is required, or
  - the user decides to kill it artificially.
- No track object persists at the end of event.
  - For the record of tracks, use trajectory class objects.
- G4TrackingManager manages processing a track, a track is represented by G4Track class.
- G4UserTrackingAction is the optional user hook.

Tracking User Action Classes

- G4UserTrackingAction
  - void PreUserTrackingAction(const G4Track*)
    - Decide if trajectory should be stored or not
    - Create user-defined trajectory
  - void PostUserTrackingAction(const G4Track*)
    - Delete unnecessary trajectory

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.).
  - Point is represented by G4StepPoint class
- 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.



Stepping User Action Class

- G4UserSteppingAction
  - void UserSteppingAction(const G4Step*)
    - Change status of track
      - Kill / suspend / postpone the track
    - Draw the step (for a track not to be stored as a trajectory)

Track status

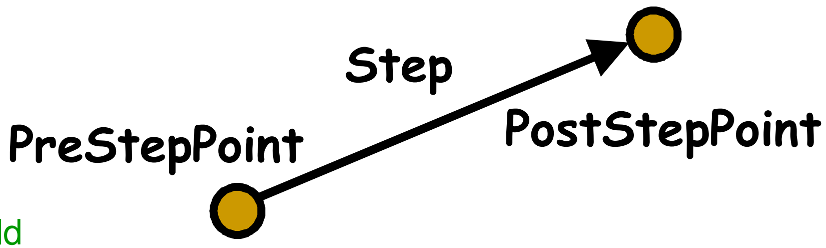
- At the end of each step, according to the processes involved, the state of a track may be changed.
  - The user can also change the status in UserSteppingAction.
  - Statuses shown in blue are for users only, i.e. Geant4 kernel won't set them.
- **fAlive**
  - Continue the tracking.
- **fStopButAlive**
  - The track has come to zero kinetic energy, but still AtRest process to occur.
- **fStopAndKill**
  - The track no longer exists --it has decayed, interacted or gone out of the world boundary.
  - Secondaries will be pushed to the stack.
- **fKillTrackAndSecondaries**
  - Kill the current track and also associated secondaries.
- **fSuspend**
  - Suspend processing of the current track and push it and its secondaries to the stack.
- **fPostponeToNextEvent**
  - Postpone processing of the current track to the next event.
  - Secondaries are still being processed within the current event.

StepPoint in Geant4

- Two step point objects attached to step
  - Pre-step point and post-step point
- G4StepPoint has information of track representing a particle at this point
  - Time (global event time, local, proper time since creation of particle
  - Position, kinetic energy, momentum
  - Material
  - …

Step status

- Step status is attached to G4StepPoint to indicate why that particular step was determined.
  - Use "PostStepPoint" to get the status of this step.
  - "PreStepPoint" has the status of the previous step.
- **fWorldBoundary**
  - Step reached the world boundary
- **fGeomBoundary**
  - Step is limited by a volume boundary except the world
- **fAtRestDoItProc**, **fAlongStepDoItProc**, **fPostStepDoItProc**
  - Step is limited by a AtRest, AlongStep or PostStep process
- **fUserDefinedLimit**
  - Step is limited by the user Step limit
- **fExclusivelyForcedProc**
  - Step is limited by an exclusively forced (e.g. shower parameterization) process
- **fUndefined**
  - Step not defined yet



- If you want to identify the first step in a volume, pick fGeomBoudary status in PreStepPoint.
- If you want to identify a step getting out of a volume, pick fGeomBoundary status in PostStepPoint

Recap – User action classes

- All needed UserAction classes
  - must be constructed in main()
  - must be provided to the RunManager using SetUserAction() method
- One mandatory User Action class
  - Event generator must be provided
  - Event generator class must be derived from G4VUserPrimaryGeneratorAction
- List of optional User Action classes
  - G4UserRunAction
  - G4UserEventAction
  - G4UserTrackingAction
  - G4UserSteppingAction
  - G4UserStackingAction

Time for exercise

- Exercise 1.2.1
- main() has UserAction added.
  - What G4UserEventAction is used for?
  - What G4UserRunAction is used for?
- Understand EventAction::EndOfEventAction(...)

Backup......

Summary

- Overview of the ‘kernel’ classes involved in simulation
- User action classes allow user to control simulation or get information and results
  - Action classes for event generation, run, event, track, and step
- Stack management allows to order priority of simulation of particles
- User information classes allow to keep arbitrary information
  - For events, tracks, primary vertex and particles, and for region.

Geant4 Track Stack

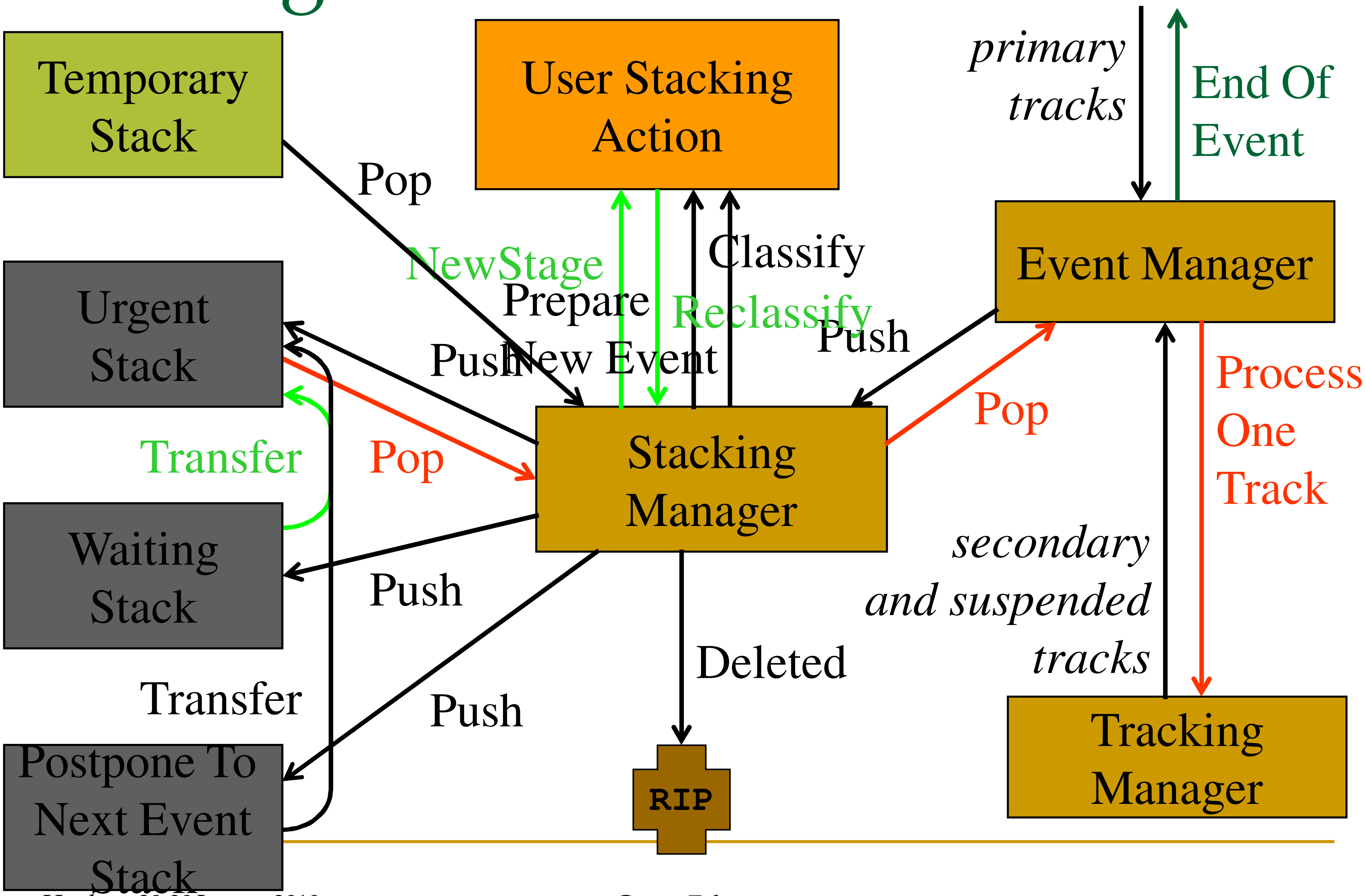
Track stacks in Geant4

- By default, Geant4 has three track stacks.
  - "Urgent", "Waiting" and "PostponeToNextEvent"
  - Each stack is a simple "last-in-first-out" stack.
  - User can arbitrary increase the number of stacks.
- A Track is popped up only from Urgent stack.
- Once Urgent stack becomes empty, all tracks in Waiting stack are transferred to Urgent stack.
- Utilizing more than one stacks, user can control the priorities of processing tracks without paying the overhead of "scanning the highest priority track".
  - Proper selection/abortion of tracks/events with well designed stack management provides significant efficiency increase of the entire simulation.

Optional User Action Classes (3) G4UserStackingAction

- User has to implement three methods.
  - G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)
    - Invoked every time a new track is pushed to G4StackManager.
    - Classification
      - fUrgent - pushed into Urgent stack
      - fWaiting - pushed into Waiting stack
      - fPostpone - pushed into PostponeToNextEvent stack
      - fKill - killed
  - void NewStage()
    - Invoked when Urgent stack becomes empty and all tracks in Waiting stack are transferred to Urgent stack.
    - All tracks which have been transferred from Waiting stack to Urgent stack can be reclassified by invoking stackManager->ReClassify()
  - void PrepareNewEvent()
    - Invoked at the beginning of each event for resetting the classification scheme.

Stacking mechanism



Examples of stacking manipulations

a) Simulate all primaries before any secondaries.
   - Classify all secondaries as fWaiting until Reclassify() method is invoked.

b) Roughly simulate the event before being bothered by low energy EM showers.
   - Classify secondary tracks below a certain energy as fWaiting until Reclassify() method is invoked.

c) Simulate secondaries before continuing to simulate primary
   - Suspend a track on its fly. Then this track and all of already generated secondaries are pushed to the stack.
   - Given a stack is "last-in-first-out", secondaries are popped out prior to the original suspended track.
   - Quite effective for Cherenkov lights

d) Simulate all tracks in a given region prior to other regions
   - Suspend all tracks that are leaving from this region, and classify these suspended tracks as fWaiting until Reclassify() method is invoked.
   - Note that some back splash tracks may come back into this region later.

- See novice example N04 for implementation of a combination of a) and a variation of d) in ExN04StackingAction class.

Attaching User Information to selected Geant4 classes

Attaching user information to some Geant4 kernel classes

- Abstract classes
  - You can use your own class derived from provided base class
  - G4Run, G4VTrajectory, G4VTrajectoryPoint
    - Other examples: G4VHit, G4VDigit
- Concrete classes
  - You can attach a user information object
    - G4Event - G4VUserEventInformation
    - G4Track - G4VUserTrackInformation
    - G4PrimaryVertex - G4VUserPrimaryVertexInformation
    - G4PrimaryParticle - G4VUserPrimaryParticleInformation
    - G4Region - G4VUserRegionInformation
  - User information object is deleted when associated Geant4 object is deleted.
  - Objects are managed, but not used by Geant4

UserInformation classes (1)

- G4VUserEventInformation
  - Additional data user wants to store for the event
    - Only Print() method is required
  - User needs to register an instance in his G4UserEventAction class indirectly with G4Event
  - Using G4EventManager::SetUserInformation(G4VUserEventInformation * ..)
      - Cannot register directly in G4Event, as this is a const pointer
    - Get previously registered object using GetUserInformation() from G4Event or G4EventManager
  - Object is deleted when G4Event object is deleted

UserInformation classes (2)

- G4VUserTrackInformation
  - Data user want to keep for track, and not in trajectory
    - Only Print() method is required
  - Pointer to UserInformation object is kept in G4Track
    - should be set from G4UserTrackingAction indirectly via
    - G4TrackingManager::SetUserInformation(G4VUserTrackInformation * ..)
      - Cannot register directly in G4Track, as this is a const pointer
    - Get previously registered object using GetUserInformation() from G4Track or G4TrackManager
  - Object is deleted when G4Track object is deleted

UserInformation classes (3)

- G4VUserPrimaryVertexInformation
  - Attach information to G4PrimaryVertex
- G4VUserPrimaryParticleInformation
  - Attach information to G4PrimaryParticle
- G4VUserRegionInformation
  - Attach information to G4Region
- Us Set/Get-UserInformation methods in G4PrimaryVertex, …, to attach object.

Transporting a Particle

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

- Geant4 tracking is general. It is independent of
  - the particle type
  - the physics processes attached to a particle
- It gives the chance to all processes
  - 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.

Processes in Geant4

- Each particle has its own list of applicable processes.
  - At each step, all processes for the particle are asked to propose a physical interaction lengths.
  - The process which requires the shortest interaction length (in space-time) limits the step.
  - A combination of processes including the step limiting process is invoked
- Each process has one or combination of the following natures.
  - AtRest
    - e.g. muon decay at rest
  - AlongStep (a.k.a. continuous process)
    - e.g. Cerenkov process
  - PostStep (a.k.a. discrete process)
    - e.g. decay on the fly
- In Geant4, particle transportation is a process as well,
  - a particle "interacts" with geometrical volume boundaries and field of any kind.
  - Because of this, shower parameterization process can take over from the ordinary transportation without modifying the transportation process.

Stacking User Action Class

- G4UserStackingAction
  - Manipulate track stack,
  - 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)

Trajectory and trajectory point (1)

- Track does not keep its trace. No track object persists at the end of event.
- 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 objects.
  - At the end of event processing, G4Event has a collection of G4Trajectory objects.
    - /tracking/storeTrajectory must be set to 1.
- G4Trajectory and G4TrajectoryPoint objects persist till the end of an event
  - Be careful not to store too many trajectories, memory growth.
    - E.g. avoid for high energy EM shower tracks.

Trajectory and trajectory point (2)

- Keep in mind the distinct classes conceptually corresponding
  - G4Track ← → G4Trajectory
  - G4Step ← → G4TrajectoryPoint
- G4Trajectory and G4TrajectoryPoint as provided by Geant4 store only the minimum information.
  - You can create your own trajectory / trajectory point classes to store information you need.
  - User classes must be derived from G4VTrajectory and G4VTrajectoryPoint base classes.
  - Do not use G4Trajectory nor G4TrajectoryPoint concrete class as base classes unless you are sure not to add any additional data member.

Caveat: Use of G4Allocator in G4Trajectory, G4TrajectoryPoint

- Instantiation / deletion of an object is a heavy operation.
  - It may cause a performance concern, in particular for objects that are frequently instantiated / deleted.
    - E.g. hit, trajectory and trajectory point classes
- G4Allocator is provided to ease such a problem.
  - It allocates a chunk of memory space for objects of a certain class.
- Please note that G4Allocator works only for a concrete class.
  - It works only for “final” class.
  - It does NOT work for a base class, in case you add a data member to your concrete class.
- Do NOT use Geant4G4Trajectory, G4TrajectoryPoint as your base class. Nor use any example concrete hit classes as base class.
  - These classes actually use G4Allocator.
  - It causes a memory leak
    - if you derive your class from such classes AND add a data member.
  - We are discussing about a protection against such incorrect use.