Advanced features

Makoto Asai (SLAC Computing Services)

Geant4 Tutorial Course @ Fermi Lab

October 29th, 2003

Geant4

Contents

- More on user action classes
  - ■ User-defined trajectory
  - ■ Suspend, postpone or kill a track
  - ■ Stack management
- ◆ Cuts per region
- ◆ Event biasing
- ◆ Defining a new shape
- ◆◆ Parameterization (fast simulation) and ghost volume
- ◆◆ Making a new physics process
- Some remarks on external functionalities
  - Persistency
  - Parallelization and integration in a distributed computing environment

■ more difficult
◆ most difficult
◆◆ expert only

More on user action classes - 1
User-defined trajectory
and
Attaching user information to some kernel classes

Geant 4

Attaching user information

- Abstract classes
  - User can use his/her own class derived from the provided base class
  - G4Run, G4VHit, G4VDigit, G4VTrajectory, G4VTrajectoryPoint
- Concrete classes
  - User can attach a user information class object
  - G4Event - G4VUserEventInformation
  - G4Track - G4VUserTrackInformation
  - G4PrimaryVertex - G4VUserPrimaryVertexInformation
  - G4PrimaryParticle - G4VUserPrimaryParticleInformation
  - G4Region - G4VUserRegionInformation
    - Classes in green are coming with Geant4 version 6.0.
    - User information class object is deleted when associated Geant4 class object is deleted.

Trajectory and trajectory point

- Trajectory and trajectory point class objects persist until the end of an event.
  - And most likely stored to disk as "simulation truth"
- G4VTrajectory is the abstract base class to represent a trajectory, and G4VTrajectoryPoint is the abstract base class to represent a point which makes up the trajectory.
  - In general, trajectory class is expected to have a vector of trajectory points.
  - Geant4 provides G4Trajectoy and G4TrajectoryPoint concrete classes as defaults.
- If the user wants to keep some additional information and/or wants to change the drawing style of a trajectory, he/she is encouraged to implement his/her own concrete classes.
  - Such user-defined trajectory object should be instantiated in **`PreUserTrackingAction()`** in user tracking action and set to G4TrackingManager

More on user action classes - 2
Suspend, postpone or kill a track

Geant 4

G4TrackStatus

- enum G4TrackStatus
  - fAlive
    - Continue tracking
  - fStopButAlive
    - Invoke active rest physics processes and kill the current track afterward
  - fStopAndKill
    - Kill the current track
    - Secondaries are still alive
  - fKillTrackAndSecondaries
    - Kill the current track and also associated secondaries.
  - fSuspend
    - Suspend the current track and send it back to the stack.
    - Associated secondaries are also sent to the stack. Given a stack is "last-in-first-out", secondaries are tracked preceded to the suspended track.
  - fPostponeToNextEvent
    - Postpone the tracking of the current track to the next event.
    - Associated secondaries are sent to the stack

Set the track status

- In UserSteppingAction, user can change the status of a track.

```
void MySteppingAction::UserSteppingAction
                              (const G4Step * theStep)
{
  G4Track* theTrack = theStep->GetTrack();
  if(…) theTrack->SetTrackStatus(fSuspend);
}
```

- If a track is killed, physics quantities of the track (energy, charge, etc.) are not conserved but completely lost.

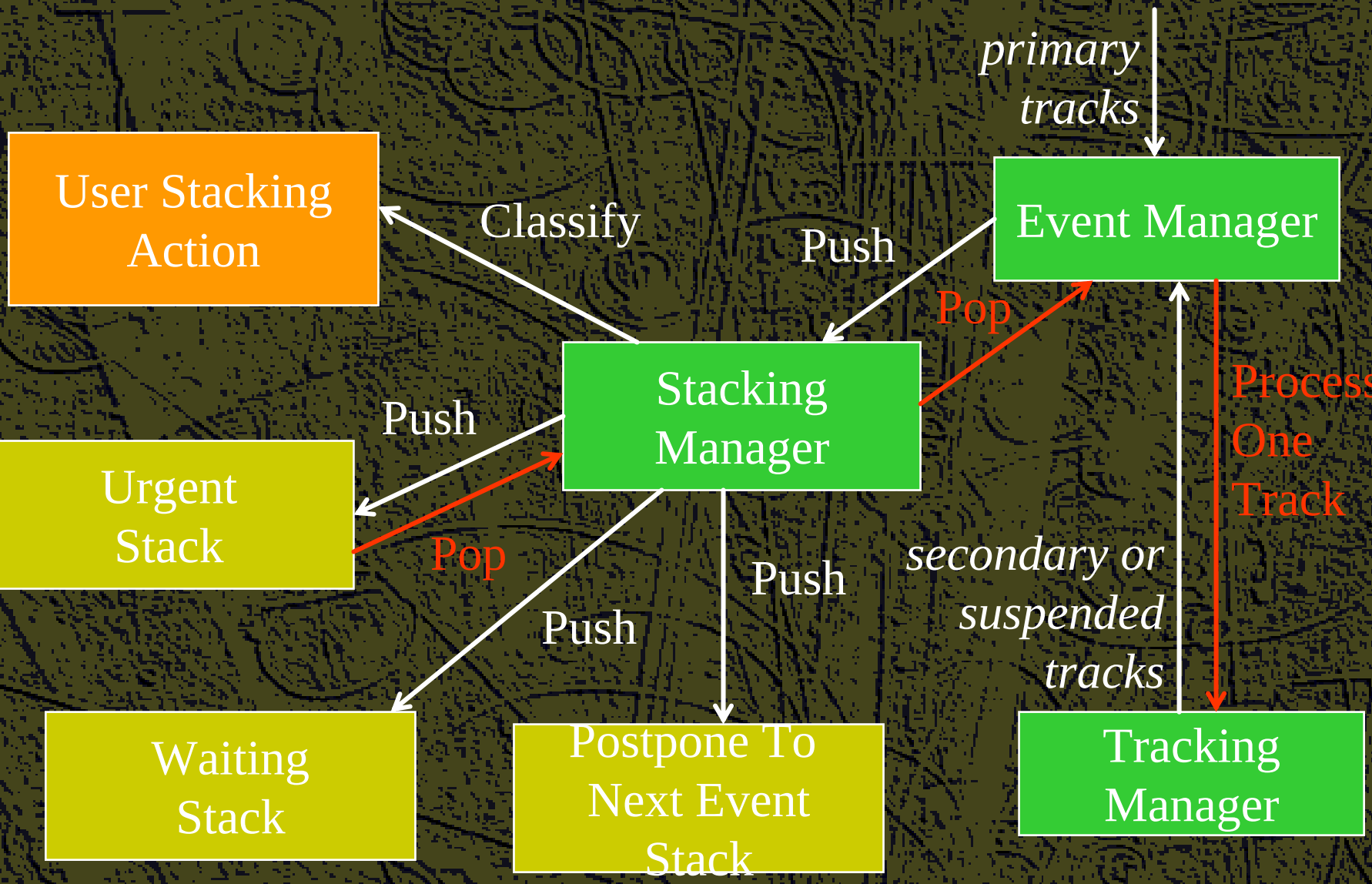
More on user action classes - 3
Stack management

Geant 4

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.
- ClassifyNewTrack() method of UserStackingAction decides which stack each new coming track to be stacked (or to be killed).
  - By default, all tracks go to Urgent stack.
- A Track is popped up only from Urgent stack.
- Once Urgent stack becomes empty, all tracks in Waiting stack are transferred to Urgent stack.
  - And NewStage() method of UsetStackingAction is invoked.
- Utilizing more than one stacks, user can control the priorities of processing tracks without paying the overhead of "scanning the highest priority track" which was the only available way in Geant3.
  - Proper selection/abortion of tracks/events with well designed stack management provides significant efficiency increase of the entire simulation.

Stacking mechanism

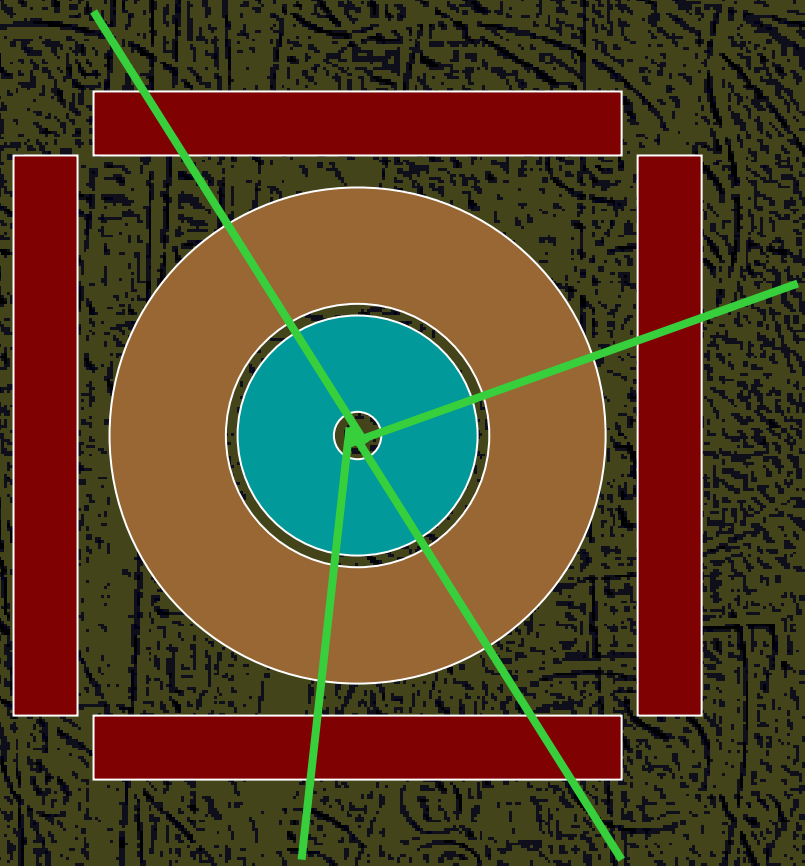


G4UserStackingAction

- 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 once Urgent stack becomes empty and all tracks in Waiting stack are transferred to Urgent stack
  - All tracks which are transferred from Waiting stack to Urgent stack can be re-classified by invoking stackManager->ReClassify()
- void PrepareNewEvent()
  - Invoked at the beginning of each event for resetting the classification scheme.

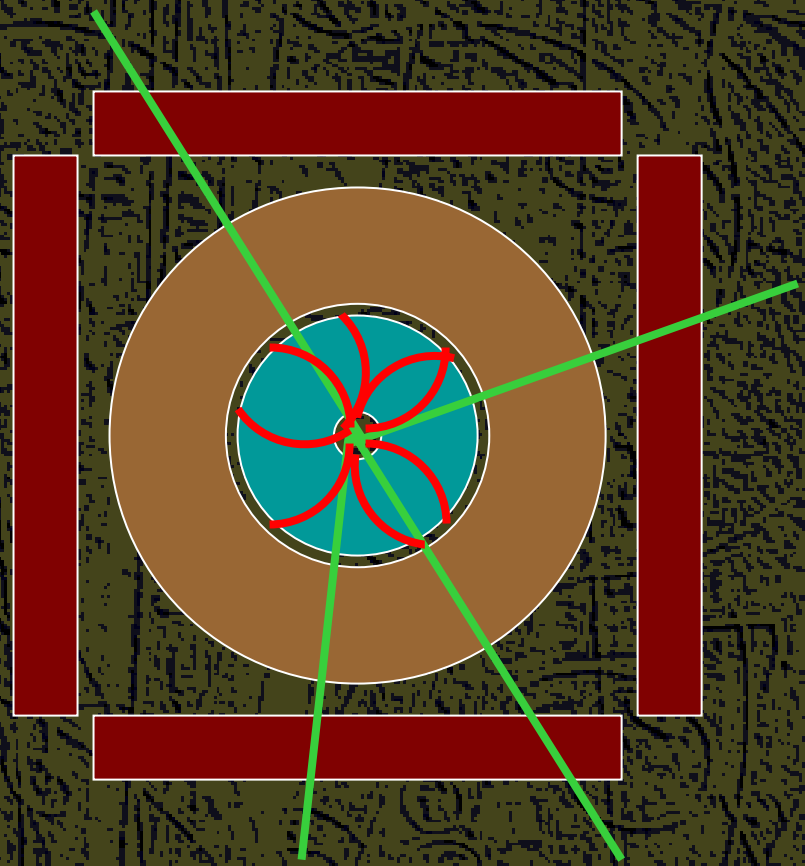
ExN04StackingAction

- ExampleN04 has simplified collider detector geometry and event samples of Higgs decays into four muons.
- Stage 0
  - Only primary muons are pushed into Urgent stack and all other primaries and secondaries are pushed into Waiting stack.
  - All of four muons are tracked without being bothered by EM showers caused by delta-rays.
  - Once Urgent stack becomes empty (i.e. end of stage 0), number of hits in muon counters are examined.
  - Proceed to next stage only if sufficient number of muons passed through muon counters. Otherwise the event is aborted.



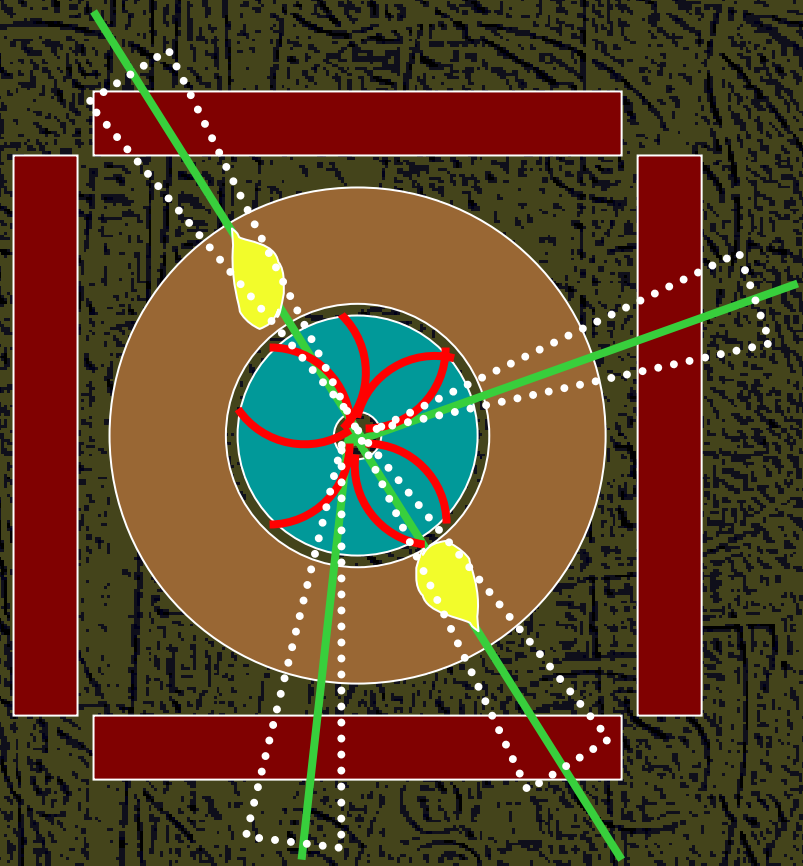
ExN04StackingAction

- Stage 1
  - Only primary charged particles are pushed into Urgent stack and all other primaries and secondaries are pushed into Waiting stack.
  - All of primary charged particles are tracked until they reach to the surface of calorimeter. Tracks reached to the calorimeter surface are suspended and pushed back to Waiting stack.
  - All charged primaries are tracked in the tracking region without being bothered by the showers in calorimeter.
  - At the end of stage 1, isolation of muon tracks is examined.



ExN04StackingAction

- Stage 2
  - Only tracks in "region of interest" are pushed into Urgent stack and all other tracks are killed.
  - Showers are calculated only inside of "region of interest".
  - Let's look at the code.



Cuts per region

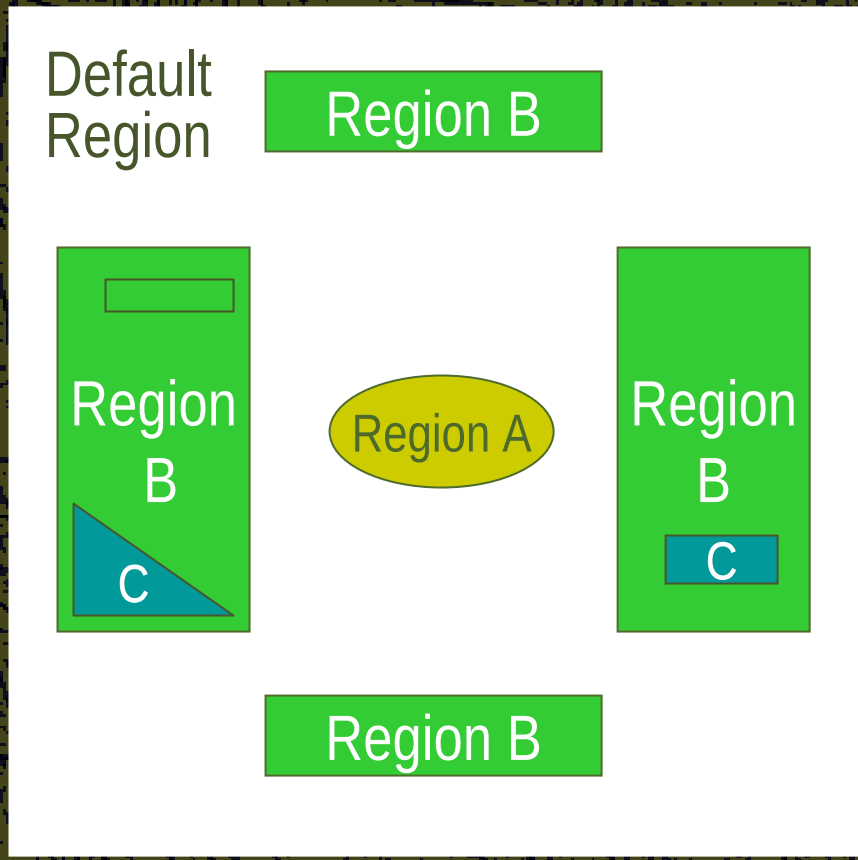
Geant 4

Cuts per Region

- Geant4 has had a unique production threshold ('cut') expressed in length (i.e. minimum range of secondary).
  - For all volumes
  - Possibly different for each particle.
- Yet appropriate length scales can vary greatly between different areas of a large detector
  - E.g. a vertex detector (5 μm) and a muon detector (2.5 cm).
  - Having a unique (low) cut can create a performance penalty.
- Requests from ATLAS, BABAR, CMS, LHCb, ..., to allow several cuts
  - Globally or per particle
- New functionality,
  - enabling the tuning of production thresholds at the level of a sub-detector, i.e. region.
  - Cuts are applied only for gamma, electron and positron and only for processes which have infrared divergence.
- 'Full release' in Geant4 5.1 (end April, 2003)
  - Comparable run-time performance

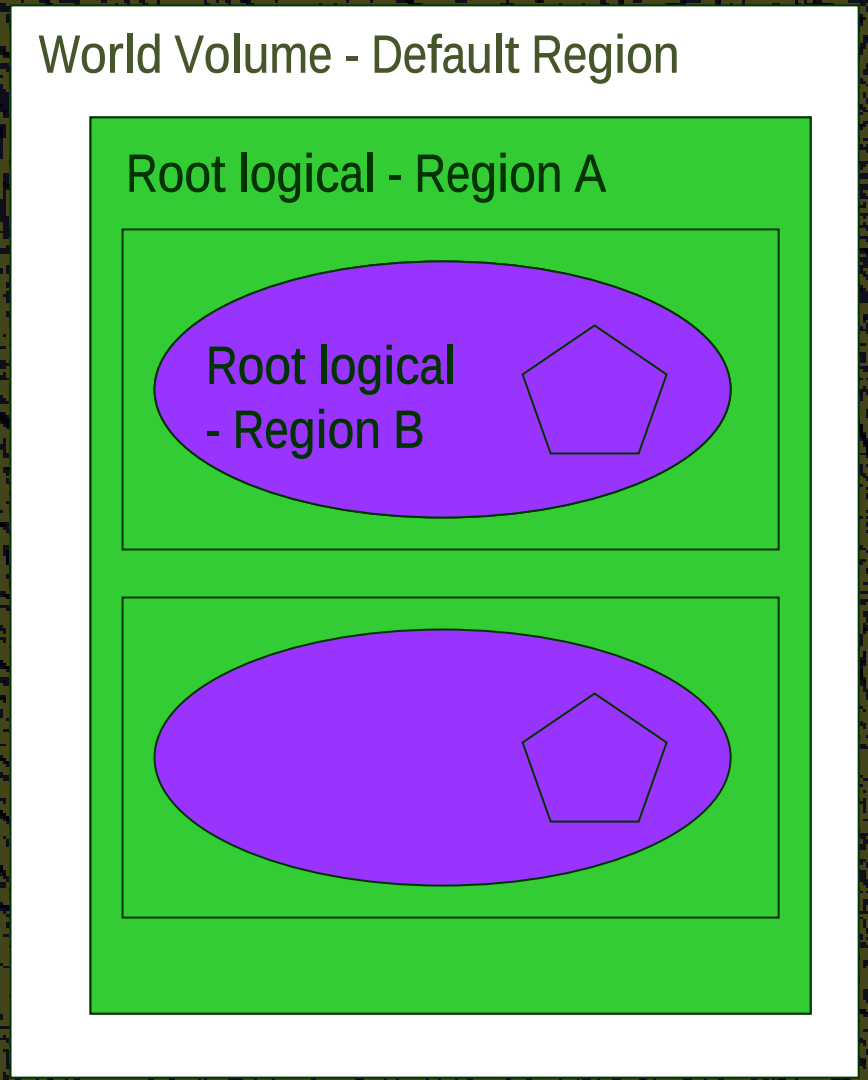
Region

- Introducing the concept of region.
  - Set of geometry volumes, typically of a sub-system;
    - barrel + end-caps of the calorimeter;
    - “Deep” areas of support structures can be a region.
  - Or any group of volumes;
- A set of cuts in range is associated to a region;
  - a different range cut for each particle among gamma, e-, e+ is allowed in a region.



Region and cut

- Each region has its unique set of cuts.
- World volume is recognized as the default region and the default cuts defined in Physics list are used for it.
  - User is not allowed to define a region to the world volume or a cut to the default region.
- A logical volume becomes a root logical volume once it is assigned to a region.
  - All daughter volumes belonging to the root logical volume share the same region (and cut), unless a daughter volume itself becomes to another root.
- Important restriction :
  - No logical volume can be shared by more than one regions, regardless of root volume or not.



Event biasing

Geant 4

Event biasing in Geant4

- Event biasing (variance reduction) technique is one of the most important requirements, which Geant4 collaboration is aware of.
- This feature could be utilized by many application fields such as
  - Radiation shielding
  - Dosimetry
- Since Geant4 is a toolkit and also all source code is open, the user can do whatever he/she wants.
  - CMS, ESA, Alice, and some other experiments have already had their own implementations of event biasing options.
- It's much better and convenient for the user if Geant4 itself provides most commonly used event biasing techniques.

Event biasing techniques

- Primary event biasing
  - Biasing primary events and/or primary particles in terms of type of event, momentum distribution, etc.
- Leading particle biasing
  - Taking only the most energetic (or most important) secondary
- Physics based biasing
  - Biasing secondary production in terms of particle type, momentum distribution, cross-section, etc.
- Geometry based biasing
  - Importance weighting for volume/region
  - Duplication or sudden death of tracks
- Forced interaction
  - Force a particular interaction, e.g. within a volume

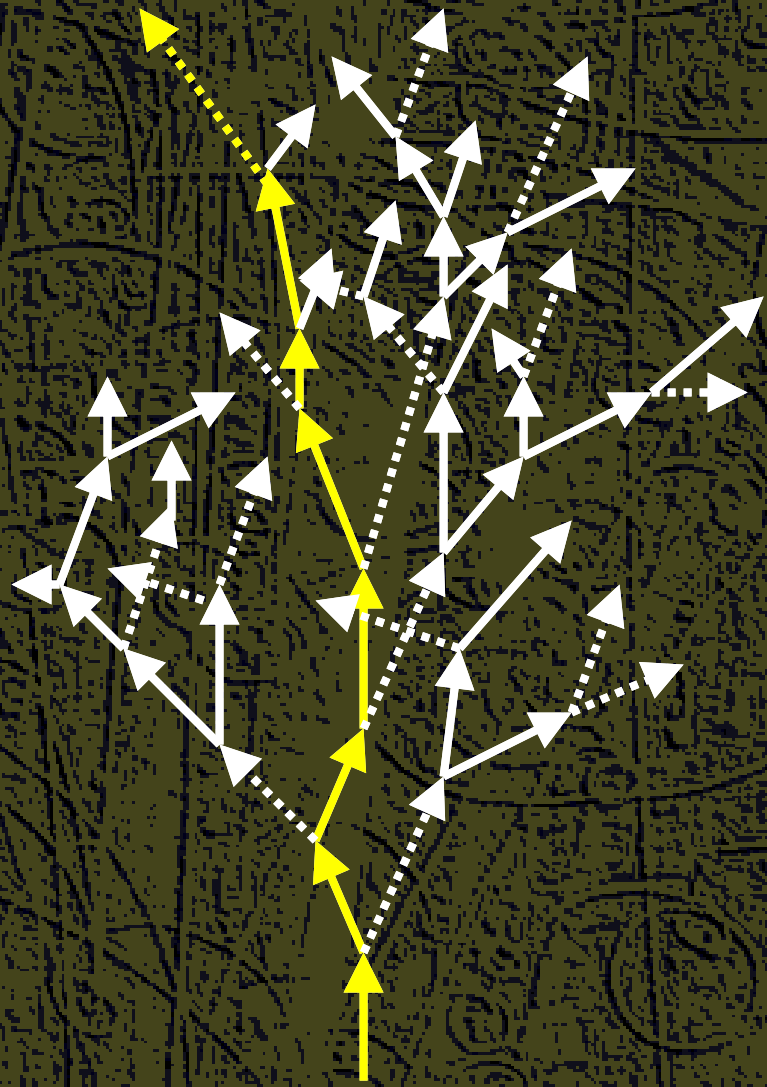
→ Weight on Track / Event

Current features in Geant4

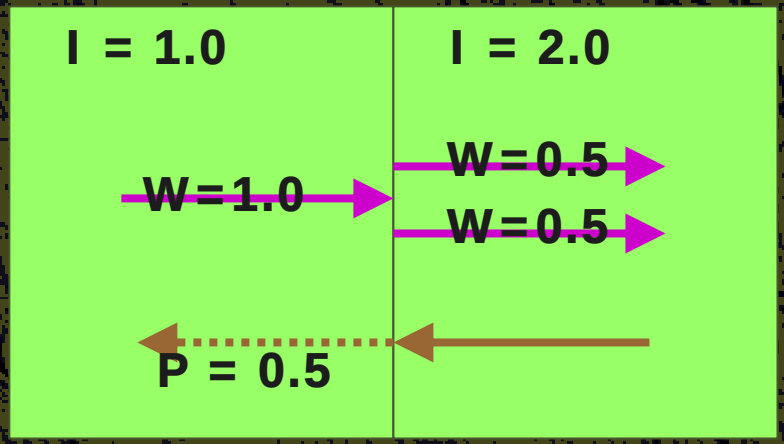
- Partial MARS migration
  - n, p, pi, K (< 5 GeV)
  - Since Geant4 0.0
- General particle source module
  - Primary particle biasing
  - Since Geant4 3.0
- Radioactive decay module
  - Physics process biasing in terms of decay products and momentum distribution
  - Since Geant4 3.0
- Cross-section biasing (partial) for hadronic physics
  - Since Geant4 3.0
- Leading particle biasing
  - Since Geant4 4.0
- Geometry based biasing
  - Weight associating with real volume or artificial volume
  - Since Geant4 5.0

Leading particle biasing

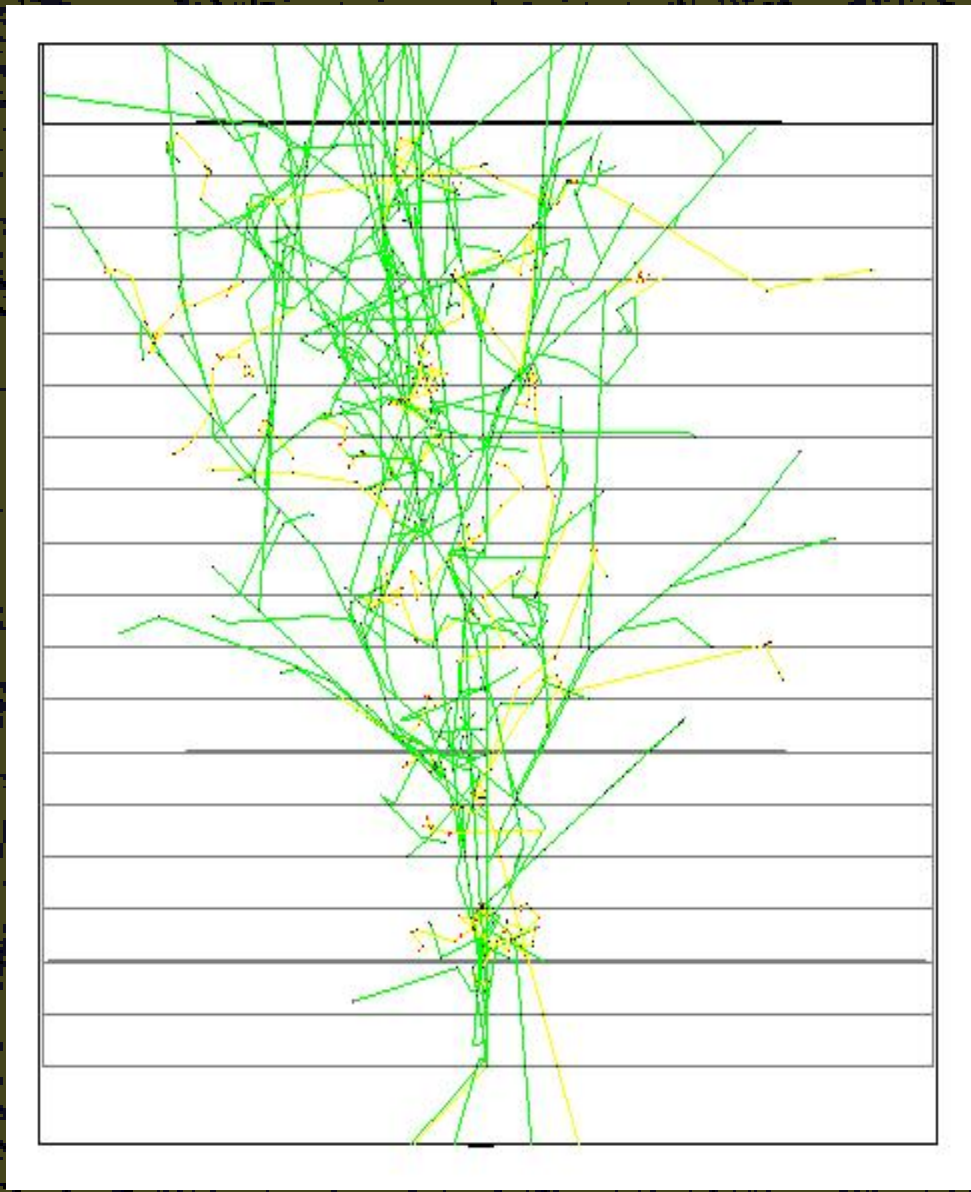
- Simulating a full shower is an expensive calculation.
- Instead of generating a full shower, trace only the most energetic secondary.
  - Other secondary particles are immediately killed before being stacked.
  - Convenient way to roughly estimate, e.g. the thickness of a shield.
  - Of course, physical quantities such as energy are not conserved for each event.



Geometrical importance biasing



- Define importance for each geometrical region
- Duplicate a track with half (or relative) weight if it goes toward more important region.
- Russian-roulette in another direction.
- Scoring particle flux with weights
  - At the surface of volumes



Plans of event biasing in Geant4

- Full interface to MARS
  - For fully biased mode
- Cross-section biasing for physics processes
- General geometrical weight field
  - In continuous process for geometrical, angular, energy biasing and weight window.
- Another biasing options are under study.
- Other scoring options rather than surface flux counting which is currently supported are under study.

→User's contribution is welcome.

Defining a new shape

Geant 4

User defined solids

- For most complicated shapes,
- All solids should derive from **`G4VSolid`** and implement its abstract interface
  - will guarantee the solid is treated as any other solid predefined in the kernel
- Basic functionalities required for a solid
  - Compute distances to/from the shape
  - Detect if a point is inside the shape
  - Compute the surface normal to the shape at a given point
  - Compute the extent of the shape
  - Provide few visualization/graphics utilities

What a solid should reply to... (1)

`EInside Inside(const G4ThreeVector& p) const;`

- Should return, considering a predefined tolerance:
  - **`kOutside`** - if the point at offset **`p`** is outside the shapes boundaries
  - **`kSurface`** - if the point is close less than **`Tolerance/2`** from the surface
  - **`kInside`** - if the point is inside the shape boundaries

`G4ThreeVector SurfaceNormal(const G4ThreeVector& p) const;`

- Should return the outwards pointing unit normal of the shape for the surface closest to the point at offset **`p`**.

`G4double DistanceToIn(const G4ThreeVector& p,`
`const G4ThreeVector& v) const;`

- Should return the distance along the normalized vector **`v`** to the shape from the point at offset **`p`**. If there is no intersection, returns **`kInfinity`**. The first intersection resulting from 'leaving' a surface/volume is discarded. Hence, it is tolerant of points on the surface of the shape

What a solid should reply to...(2)

`G4double` **`DistanceToIn`**`(const G4ThreeVector& p) const;`

- Calculates the distance to the nearest surface of a shape from an outside point `p`. The distance can be an underestimate

```
G4double DistanceToOut(const G4ThreeVector& p,
  const G4ThreeVector& v, const G4bool  calcNorm=false,
  G4bool* validNorm=0, G4ThreeVector* n=0) const;
```

- Returns the distance along the normalised vector `v` to the shape, from a point at an offset `p` inside or on the surface of the shape. If `calcNorm` is `true`, then it must also set `validNorm` to either:
  - `True` - if the solid lies entirely behind or on the exiting surface. Then it must set `n` to the outwards normal vector (the Magnitude of the vector is not defined)
  - `False` - if the solid does not lie entirely behind or on the exiting surface

`G4double` **`DistanceToOut`**`(const G4ThreeVector& p) const;`

- Calculates the distance to the nearest surface of a shape from an inside point `p`. The distance can be an underestimate

Solid: more functions...

```
G4bool CalculateExtent(const EAxis pAxis,
       const G4VoxelLimits& pVoxelLimit,
       const G4AffineTransform& pTransform,
       G4double& pMin, G4double& pMax) const;
```

- Calculates the minimum and maximum extent of the solid, when under the specified transform, and within the specified limits. If the solid is not intersected by the region, return **`false`**, else return **`true`**

Member functions for the purpose of visualization:

```
void DescribeYourselfTo(G4VGraphicsScene& scene) const;
```

- "double dispatch" function which identifies the solid to the graphics scene

```
G4VisExtent GetExtent() const;
```

- Provides extent (bounding box) as possible hint to the graphics view

Fast simulation
(shower parameterization)

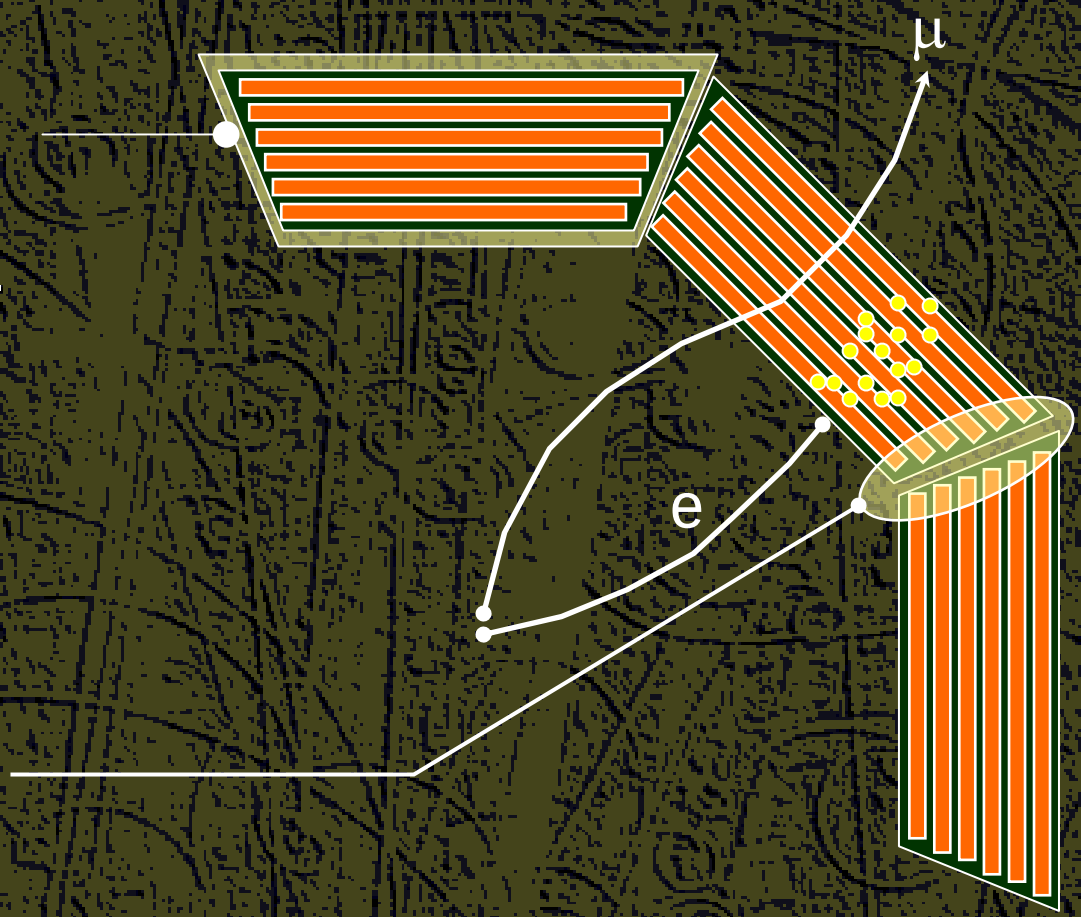
Geant 4

Fast simulation - Generalities

- Fast Simulation, also called as shower parameterization, is a shortcut to the "ordinary" tracking.
- Fast Simulation allows you to take over the tracking and implement your own "fast" physics and detector response.
- The classical use case of fast simulation is the shower parameterization where the typical several thousand steps per GeV computed by the tracking are replaced by a few ten of energy deposits per GeV.
- Parameterizations are generally experiment dependent. Geant4 provides a convenient framework.

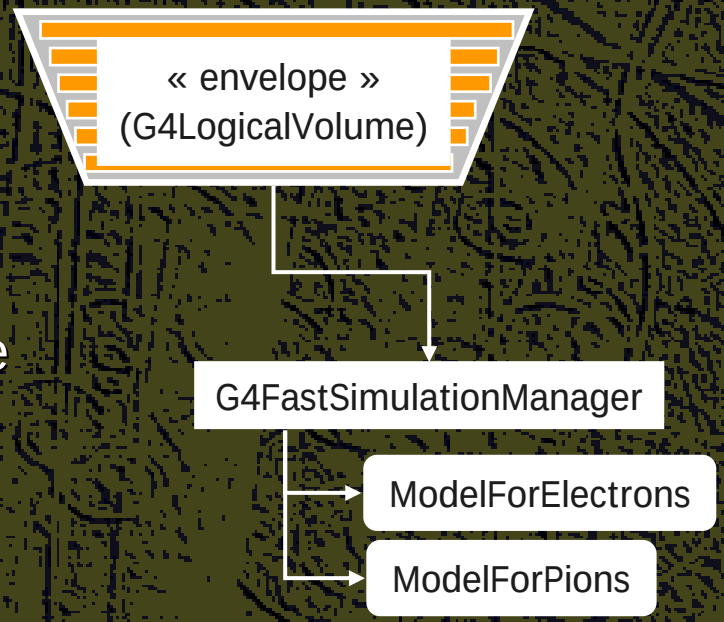
Parameterization features

- Parameterizations take place in an *envelope*. This is typically a mother volume of a sub-system or of a major module of such a sub-system.
- Parameterizations are often dependent to particle types and/or may be applied only to some kinds of particles.
- They are often not applied in complicated regions.



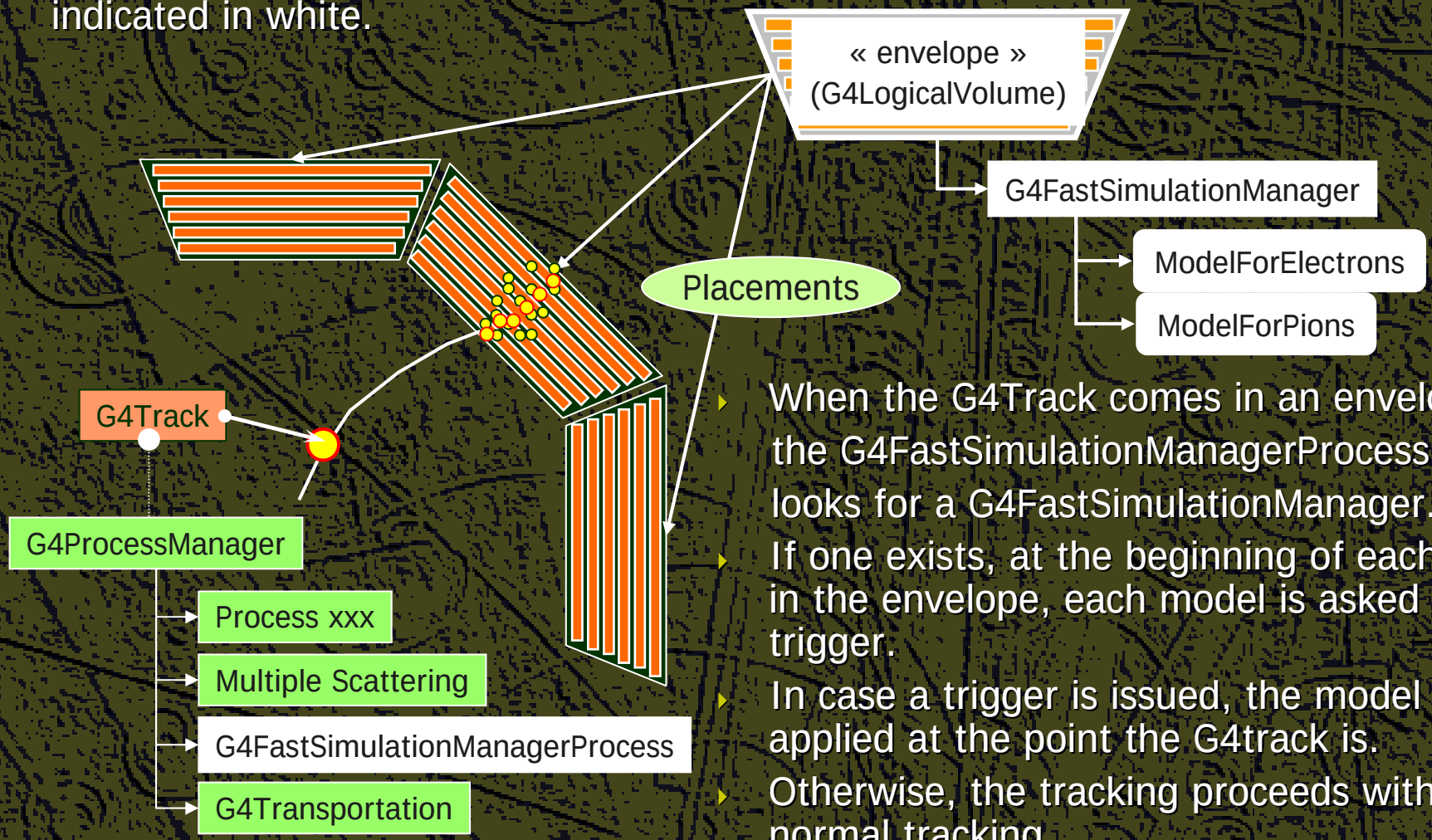
Models and envelope

- Concrete models are bound to the envelope through a G4FastSimulationManager object.
- This allows several models to be bound to one envelope.
- The envelope is simply a G4LogicalVolume which has G4FastSimulationManager.
- All its [grand[…]]daughters will be sensitive to the parameterizations.
- A model may returns back to the "ordinary" tracking the new state of G4Track after parameterization (alive/killed, new position, new momentum, etc.) and eventually adds secondaries (e.g. punch through) created by the parameterization.



Fast Simulation

- The Fast Simulation components are indicated in white.



- When the G4Track comes in an envelope, the G4FastSimulationManagerProcess looks for a G4FastSimulationManager.
- If one exists, at the beginning of each step in the envelope, each model is asked for a trigger.
- In case a trigger is issued, the model is applied at the point the G4track is.
- Otherwise, the tracking proceeds with a normal tracking.

G4FastSimulationManagerProcess

- The G4FastSimulationManagerProcess is a process providing the interface between the tracking and the fast simulation.
- It has to be set to the particles to be parameterized:
  - The process ordering must be the following:

    [n-3] ...

    [n-2] Multiple Scattering

    [n-1] G4FastSimulationManagerProcess

    [n] G4Transportation
  - It can be set as a discrete process or it must be set as a continuous & discrete process if using ghost volumes.

Ghost Volume

- Ghost volumes allow to define envelopes independent to the volumes of the tracking geometry.
  - For example, this allows to group together electromagnetic and hadronic calorimeters for hadron parameterization or to define envelopes for imported geometries which do not have a hierarchical structure.
- In addition, Ghost volumes can be sensitive to particle type, allowing to define envelops individually to particle types.
- Ghost Volume of a given particle type is placed as a clone of the world volume for tracking.
  - This is done automatically by G4GlobalFastSimulationManager.
- The G4FastSimulationManagerProcess provides the additional navigation inside a ghost geometry. This special navigation is done transparently to the user.

Making a new physics process

Geant 4

Introduction

- Geant4 has been designed to allow users to implement new processes and to plug-in with any other existing processes.
- Process is designed in a generic/general way.
  - It is good, but it requires the user to understand some key concepts/components.
- Most important concepts/components
  - G4VProcess
  - Interaction length
  - Force condition
  - G4VParticleChange
- G4Decay is a relatively simple class and it could be a good example to learn how a process is implemented in Geant4.

G4VProcess abstract base class

- G4VProcess has six pure virtual methods, which have already been introduced yesterday
  - AtRestGetPhysicalInteractionLength
  - AlongStepGetPhysicalInteractionLength
  - PostStepGetPhysicalInteractionLength
  - AtRestDoIt
  - AlongStepDoIt
  - PostStepDoIt
- The user has to implement some/all of these methods according to the nature of the process.
  - Deriving directly from G4VProcess base class or from an intermediate layer abstract class, e.g. G4VDiscreteProcess.
- G4VProcess also has several virtual (but not pure virtual) methods.
  - BuildPhysicsTable(const G4ParticleDefinition&)
    - Invoked by the run manager for potential initialization and/or (re-)calculation of cross-section table
  - StartTracking()
    - Invoked by the tracking manager typically for calculating a fate of this particular track to be proposed

Interaction length

- Each process has "number of interaction length left", N_{int}.
- For a new track, or after the occurrence of the process itself, N_{int} is reset.
  - For example by randomizing with the exponential law
- At the beginning of each step, the process subtracts the amount of interaction length spent by the previous step.
- Then the process proposes the step size (by its GPIL method) in terms of actual space-time by multiplying remaining N_{int} with λ_{free}, "mean free path" of the current material.
- Refer to the implementation G4VDiscreteProcess class as an example.
- According to the nature of the process you are implementing, the way of estimating the remaining interaction length (i.e. proposed step length) is different.
  - For example, optical refraction process itself does not require the remaining interaction length, but it takes place when a step is limited by a transportation process.

Selection mechanism of processes

- The naïve case
  - In case a track is at rest
    - A process which proposes shortest time interval is chosen.
  - In case a track is in motion
    - All registered AlongStep processes and a PostStep process which proposes shortest step length are chosen.
- Some AtRest and PostStep processes need to be invoked anyway, regardless of which process limits a step.
  - For example, PostStepDoIt of the transportation process (ContinuousDiscrete process with AlongStep and PostStep DoIt's) needs to be invoked for the sake of relocation even if the step is limited by other process.
  - For another example, optical refraction process should be invoked if not itself but the transportation process limits a step.
- Some processes (typically shower parameterization process) may want to take over other processes.
- For such complicated requirements, Each of AtRest and PostStep (not AlongStep) GPIL methods returns "force condition" in addition to the proposal of the step length.

Force condition

- G4ForceCondition enumeration
- NotForced
  - DoIt of this process is invoked only if this process limits a step
- Forced
  - As far as the particle survives after the step, DoIt of this process must be invoked even if this process does not limit a step
- StronglyForced
  - DoIt of this process is invoked anyway even if the track is killed at this step.
- Conditionally
  - PostProcessDoIt of this process is forced to invoke only when corresponding AlongStepGPIL limits the Step.
- ExclusivelyForced
  - DoIt of this process is exclusively invoked even if other process limit a step or propose "forced" condition. All other DoIt of AlongStep and PostStep are ignored.
- InActivated
  - This process is inactivated by a user

G4VParticleChange

- Each DoIt method takes const G4Track* and const G4Step*.
  - DoIt method itself is not allowed to modify a track or a step.
- Each DoIt method must return its own G4VParticleChange (or its derivative class) object to affect to the physical quantities and/or status of the track.
- G4VParticleChange has
  - a virtual method to update a step (and corresponding physical quantities)
    - Not all processes change all physical quantities of a step. Thus derived concrete particleChange class may access to only some portion of the quantities which this particular process actually affects to.
  - proposal of a new status of the track
  - secondaries produced by the process

Some remarks on external functionalities

Geant 4

Persistency

- Geant4 does not rely on any particular persistency solution.
  - User should provide his/her own solution
    - Exception : Cross-section tables
- Geant4 provides various examples
- Event input
  - G4HEPEvtInterface, G4HepMCInterface
- Geometry
  - XML, GDML, STEP, GGE (Geant4 Geometry Editor), etc.
- Histograms
  - AIDA, ROOT
- Primaries, hits, trajectories, digits
  - G4VPersistencyManager abstract base class
  - Convert Geant4 objects to user persistency objects
    - ASCII file, ROOT, Objectivity/DB, etc.

Parallelization

- By design, Geant4 can be executed in more than one processes/machines in parallel.
- Geant4 itself does not provide any mechanism of parallelization but with some external utilities.
  - "Event parallelism"
    - Master process distributes events to slave processes.
    - Geometry, physics processes, user classes, parameters are sent to slave processes before processing events.
    - Event output and histogram entries are sent back to the master process to be collected.
- Geant4 provides an example based on TOP-C.
  - examples/extended/parallel
  - TOP-C : developed by G.Cooperman (Northeastern U.)