Kernel II

Makoto Asai (SLAC)

Geant4 Tutorial Course

Contents

▸ User limits

▸ Attaching user information to G4 classes

▸ Stacking mechanism

User limits

G4UserLimits

▸ User limits are artificial limits affecting to the tracking.

```
G4UserLimits(G4double ustepMax = DBL_MAX,
             G4double utrakMax = DBL_MAX,
             G4double utimeMax = DBL_MAX,
             G4double uekinMin = 0.,
             G4double urangMin = 0. );
```

- ▸ fMaxStep;   // max allowed Step size in this volume
- ▸ fMaxTrack;   // max total track length
- ▸ fMaxTime;   // max global time
- ▸ fMinEkine;   // min kinetic energy remaining (only for charged particles)
- ▸ fMinRange;   // min remaining range (only for charged particles)

```
Blue : affecting to step
Yellow : affecting to track
```

▸ You can set user limits to logical volume and/or to a region.

- ▸ User limits assigned to logical volume do not propagate to daughter volumes.
- ▸ User limits assigned to region propagate to daughter volumes unless daughters belong to another region.
- ▸ If both logical volume and associated region have user limits, those of logical volume win.

Processes co-working with G4UserLimits

- In addition to instantiating G4UserLimits and setting it to logical volume or region, you have to assign the following process(es) to particle types you want to affect.

- Limit to step

  fMaxStep : max allowed Step size in this volume

  - G4StepLimiter process must be defined to affected particle types.
  - This process limits a step, but it does not kill a track.

- Limits to track

  fMaxTrack : max total track length

  fMaxTime : max global time

  fMinEkine : min kinetic energy (only for charged particles)

  fMinRange : min remaining range (only for charged particles)

  - G4UserSpecialCuts process must be defined to affected particle types.
  - This process limits a step and kills the track when the track comes to one of these limits. Step limitation occurs only for the final step.

Attaching user information to some kernel classes

Attaching user information

▸ Abstract classes

  ▸ You can use your own class derived from provided base class

  ▸ G4Run, G4VHit, G4VDigit, G4VTrajectory, G4VTrajectoryPoint

▸ Concrete classes

  ▸ You can attach a user information class object

    ▸ G4Event - G4VUserEventInformation

    ▸ G4Track - G4VUserTrackInformation

    ▸ G4PrimaryVertex - G4VUserPrimaryVertexInformation

    ▸ G4PrimaryParticle - G4VUserPrimaryParticleInformation

    ▸ G4Region - G4VUserRegionInformation

  ▸ 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.

▸ 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. These classes keep only the most common quantities.

  ▸ If the you want to keep some additional information, you are encouraged to implement your own concrete classes deriving 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.

Use of G4Allocator

- 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 G4Trajectory, G4TrajectoryPoint nor any example concrete hit classes as your 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.

Creation of trajectories

▸ Naïve creation of trajectories occasionally causes a memory consumption concern, especially for high energy EM showers.

▸ In UserTrackingAction, you can switch on/off the creation of a trajectory for the particular track.

```
void MyTrackingAction
          ::PreUserTrackingAction(const G4Track* aTrack)
{
  if(...)
  { fpTrackingManager->SetStoreTrajectory(true); }
  else
  { fpTrackingManager->SetStoreTrajectory(false); }
}
```

▸ If you want to use user-defined trajectory, object should be instantiated in this method and set to G4TrackingManager by SetTrajectory() method.

```
 fpTrackingManager->SetTrajectory(new MyTrajectory(…));
```

Bookkeeping issues

- Connection from G4PrimaryParticle to G4Track

  G4int G4PrimaryParticle::GetTrackID()

  - Returns the track ID if this primary particle had been converted into G4Track, otherwise -1.

    - Both for primaries and pre-assigned decay products

- Connection from G4Track to G4PrimaryParticle

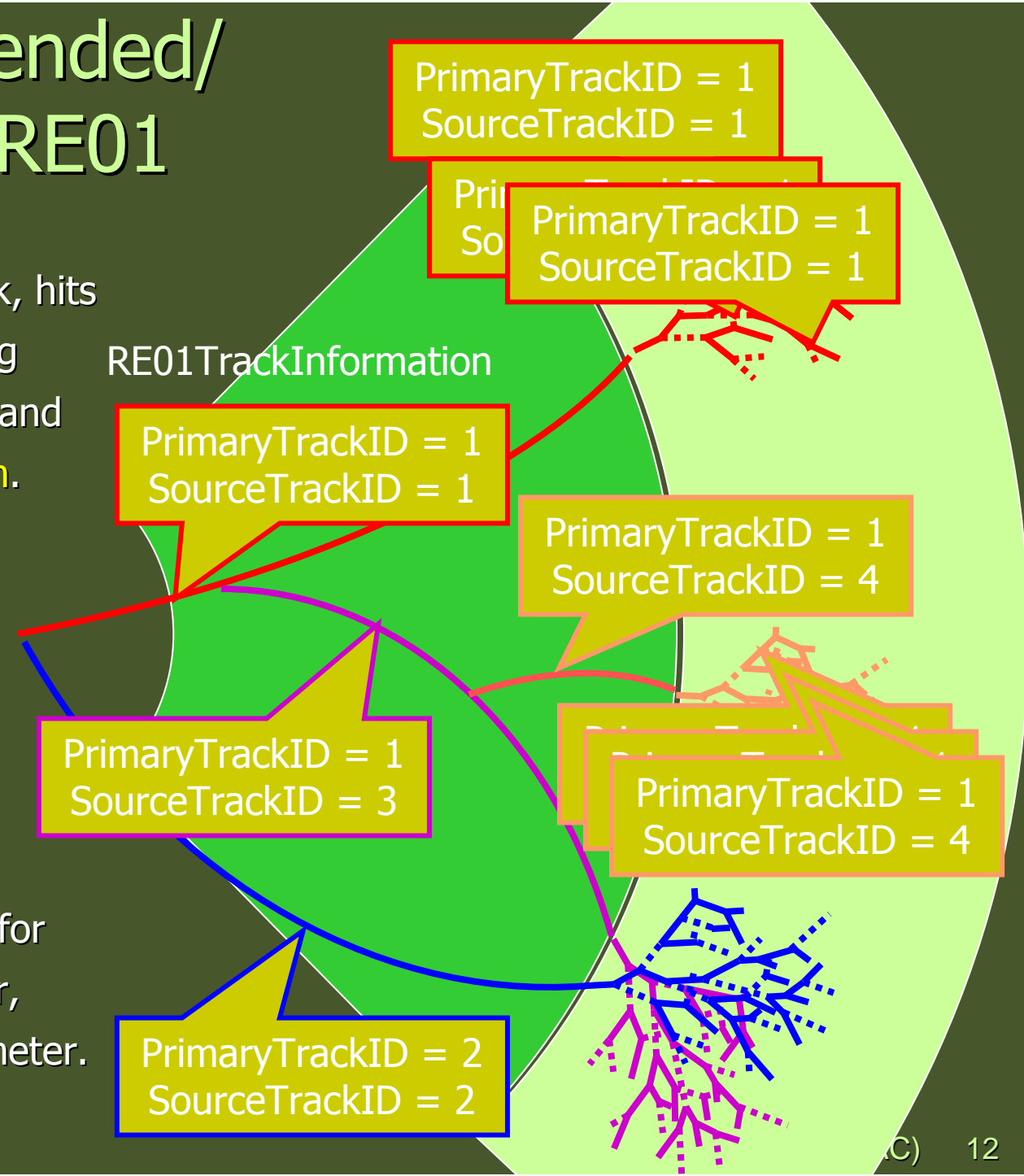 G4PrimaryParticle* G4DynamicParticle::GetPrimaryParticle()

  - Returns the pointer of G4PrimaryParticle object if this track was defined as a primary or a pre-assigned decay product, otherwise null.

- G4VUserPrimaryVertexInformation, G4VUserPrimaryParticleInformation and G4VUserTrackInformation can be utilized for storing additional information.

  - Information in UserTrackInformation should be then copied to user-defined trajectory class, so that such information is kept until the end of the event.

Examples/extended/runAndEvent/RE01

- An example for connecting G4PrimaryParticle, G4Track, hits and trajectories, by utilizing G4VUserTrackInformation and G4VUserRegionInformation.

- SourceTrackID means the ID of a track which gets into calorimeter.

- PrimaryTrackID is copied to UserTrackInformation of daughter tracks.

- SourceTrackID is updated for secondaries born in tracker, while just copied in calorimeter.

PrimaryTrackID = 1
SourceTrackID = 1

PrimaryTrackID = 1
SourceTrackID = 1

RE01TrackInformation

PrimaryTrackID = 1
SourceTrackID = 1

PrimaryTrackID = 1
SourceTrackID = 4

PrimaryTrackID = 1
SourceTrackID = 3

PrimaryTrackID = 1
SourceTrackID = 4

PrimaryTrackID = 2
SourceTrackID = 2

12

Examples/extended/runAndEvent/RE01

```
Primary particles ----------------------------------------------------------

Primary vertex (0,0,0)    at t = 0 [ns]
  ==PDGcode 25 is not defined in G4 (19.53824,24.846369,-6.0465937) [GeV] >>> G4Track ID 1
    ==PDGcode 23 is not defined in G4 (1.1302123,-23.156443,114.16953) [GeV] >>> G4Track ID 6780
      ==PDGcode 13 (mu-) (-22.464989,-38.451706,20.864853) [GeV] >>> G4Track ID 6782
        ==PDGcode -13 (mu+) (23.595201,15.29526,93.304688) [GeV] >>> G4Track ID 6781
```

```
TrackID =6782 : ParentID=6780 : TrackStatus=1
  Particle name : mu-   PDG code : 13   Charge : -1
  Original momentum : -22.464989 -38.451706 20.864853 GeV
  Vertex : 4.11461
    Current traject
  Point[0] Positio
  Point[1] Positio
  Point[2] Positio
  Point[3] Positio
  Point[4] Positio
  Point[5] Positio
```

Trajectory of track6782

```
TrackID 6782    Position (-126.11431,-215.85917.117.12988)       : 1878.8831 [keV]
TrackID 6782    Position (-127.89383,                     
TrackID 6782    Position (-151.33864,
TrackID 6782    Position (-176.56317,-302.20101,163.96689)       : 1776.6378 [keV]
TrackID 6782    Position (-201.7911,-345.36988,187.38871)        : 2413.8986 [keV]
TrackID 6782    Position (-227.01961,-388.53841,210.81469)       : 550.67792 [keV]
TrackID 6782    Position (-227.70865,-389.71739,211.45445)       : 638.57593 [keV]
TrackID 6782    Position (-228.6702,-391.36253,212.34721)        : 778.03992 [keV]
```

Tracker hits of track6782

```
Source track ID 6782 (mu-,49.162515[GeV]) at (-252.24762,-431.70723,234.23766)
Original primary track ID 1 (unknown,335.96305[GeV])
Cell[11,31]    0.028283647 [GeV]
Cell[12,31]    0.039822296 [GeV]
Cell[13,31]    0.050185748 [GeV]
Cell[14,31]    0.049883344 [GeV]
Cell[15,31]    0.041446764 [GeV]
Cell[16,31]    0.06386168 [GeV]
Cell[16,32]    0.0036926015 [GeV]
Cell[17,32]    7.2955385e-05 [GeV]
Cell[17,31]    0.0043463898 [GeV]
Cell[15,32]    0.010138473 [GeV]
Cell[14,32]    0.0018386352 [GeV]
Cell[13,32]    0.0018836759 [GeV]
Cell[12,32]    0.00036846059 [GeV]
###   Total energy deposition in calorimeter by a source track in 13 cells : 0.29582467 (GeV)
```

Calorimeter hits of track6782

Energy deposition includes not only muon itself but also all secondary EM showers started inside the calorimeter.

RE01RegionInformation

- RE01 example has three regions, i.e. default world region, tracker region and calorimeter region.
  - Each region has its unique object of RE01RegionInformation class.

```
class RE01RegionInformation : public G4VUserRegionInformation
{
  …
  public:
   G4bool IsWorld() const;
   G4bool IsTracker() const;
   G4bool IsCalorimeter() const;
  …
};
```

- Through step->preStepPoint->physicalVolume->logicalVolume->region->regionInformation, you can easily identify in which region the current step belongs.
  - Don't use volume name to identify.

Use of RE01RegionInformation

```
void RE01SteppingAction::UserSteppingAction(const G4Step * theStep)
{ // Suspend a track if it is entering into the calorimeter

  // get region information
  G4StepPoint* thePrePoint = theStep->GetPreStepPoint();
  G4LogicalVolume* thePreLV = thePrePoint->GetPhysicalVolume()->GetLogicalVolume();
  RE01RegionInformation* thePreRInfo
   = (RE01RegionInformation*)(thePreLV->GetRegion()->GetUserInformation());
  G4StepPoint* thePostPoint = theStep->GetPostStepPoint();
  G4LogicalVolume* thePostLV = thePostPoint->GetPhysicalVolume()->GetLogicalVolume();
  RE01RegionInformation* thePostRInfo
   = (RE01RegionInformation*)(thePostLV->GetRegion()->GetUserInformation());

  // check if it is entering to the calorimeter volume
  if( !(thePreRInfo->IsCalorimeter()) && (thePostRInfo->IsCalorimeter()) )
  { theTrack->SetTrackStatus(fSuspend); }
}
```
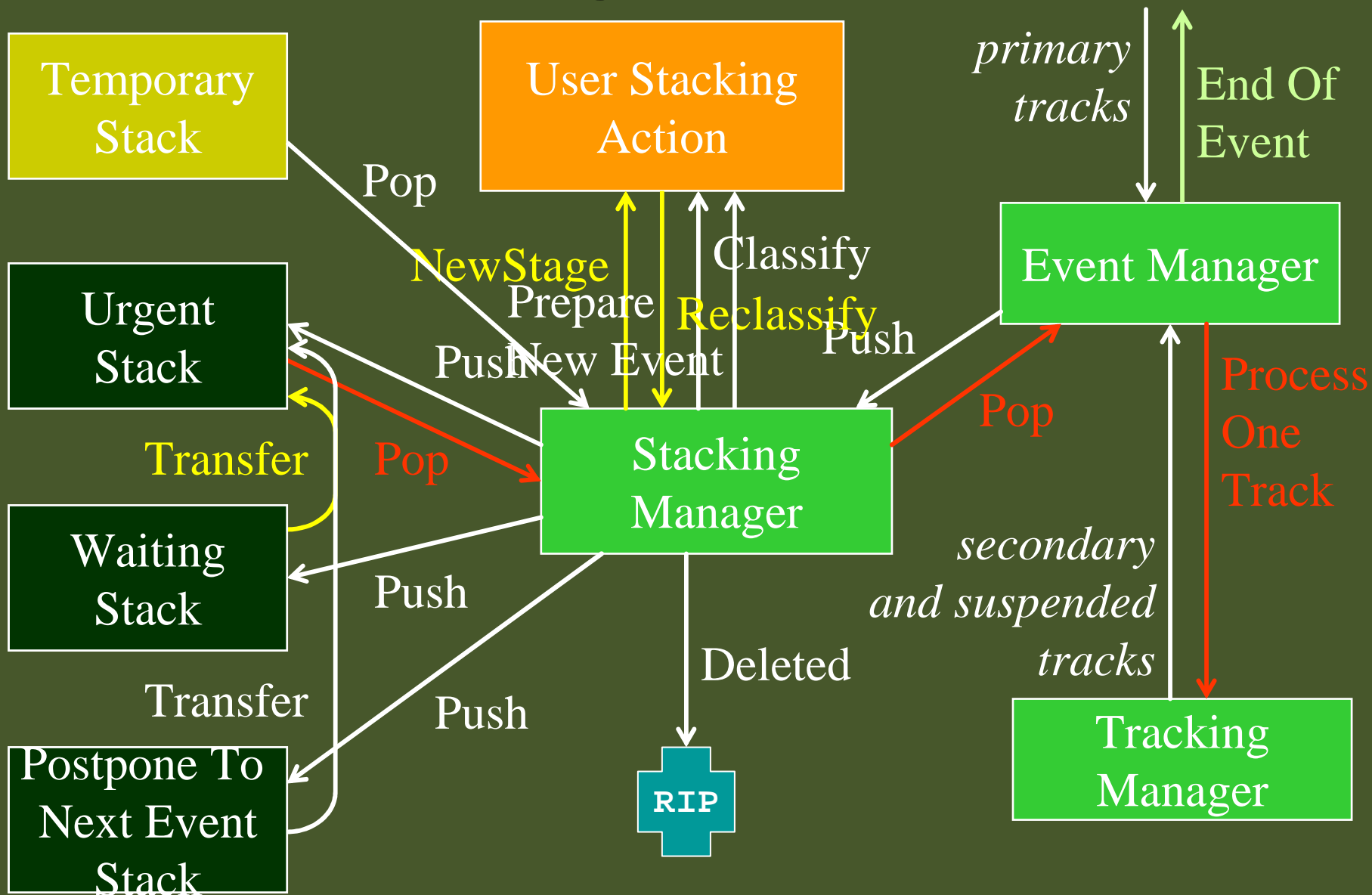
Stack management

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 newly storing 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".
  - 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 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.

Tips of stacking manipulations

- Classify all secondaries as fWaiting until Reclassify() method is invoked.
  - You can simulate all primaries before any secondaries.
- Classify secondary tracks below a certain energy as fWaiting until Reclassify() method is invoked.
  - You can roughly simulate the event before being bothered by low energy EM showers.
- 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
- Suspend all tracks that are leaving from a region, and classify these suspended tracks as fWaiting until Reclassify() method is invoked.
  - You can simulate all tracks in this region prior to other regions.
  - Note that some back splash tracks may come back into this region later.

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 in UserSteppingAction, physics quantities of the track (energy, charge, etc.) are not conserved but completely lost.
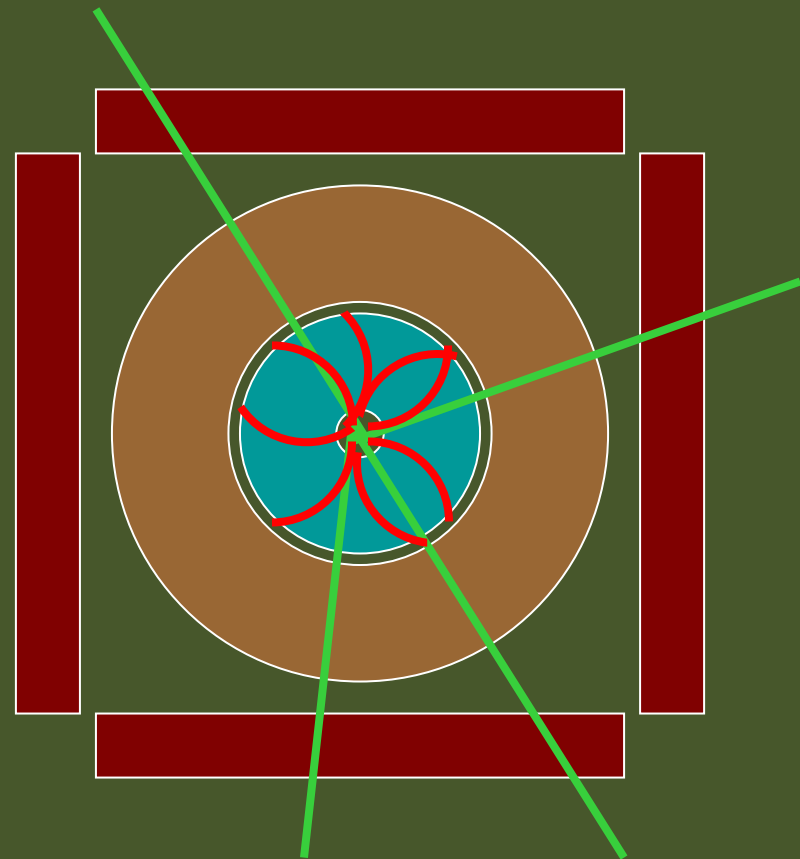
ExN04StackingAction

- 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.
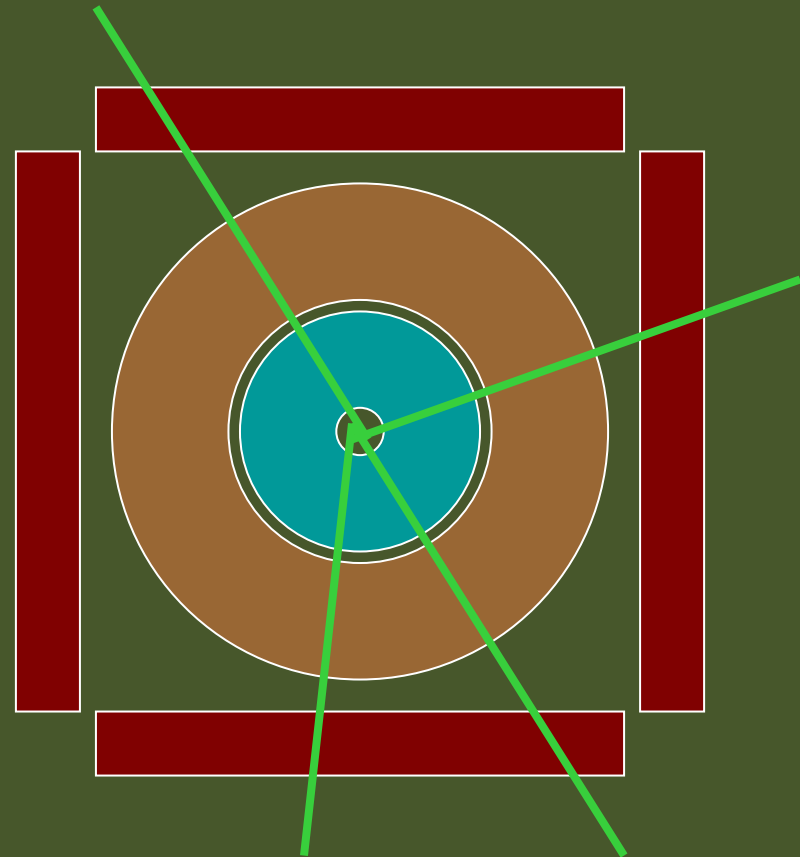
ExN04StackingAction

- 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".