Version 11.2

User classes

Makoto Asai (Jefferson Lab)

Geant4 Tutorial Course

Contents

- User limits
- User classes
- Attaching user information to G4 classes
- Stacking mechanism

Contents

- User limits
- User classes
- Attaching user information to G4 classes
- Stacking mechanism

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

  Red : 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)

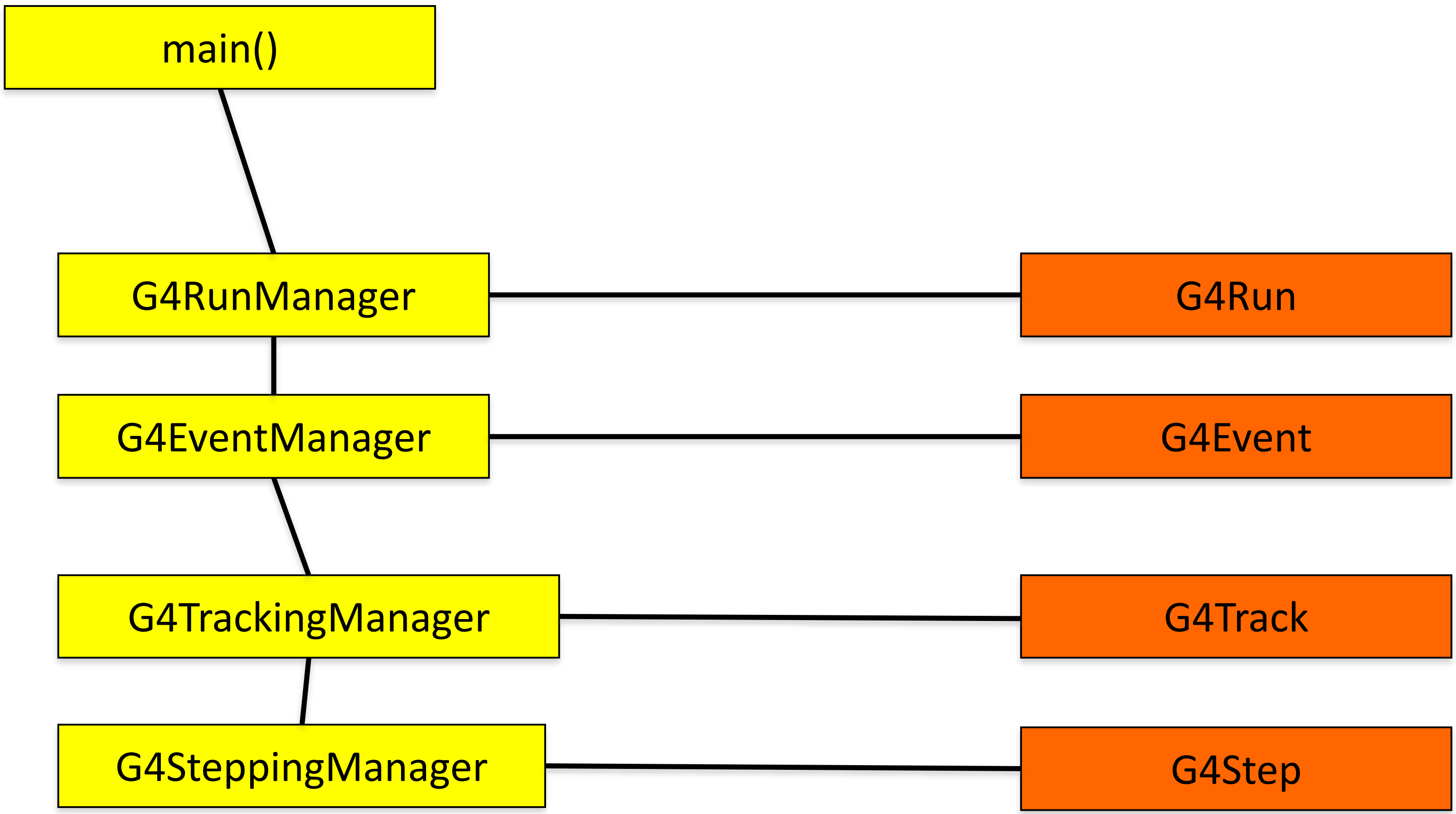
  fMinRange : min remaining range (only for charged particles)

  - 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.

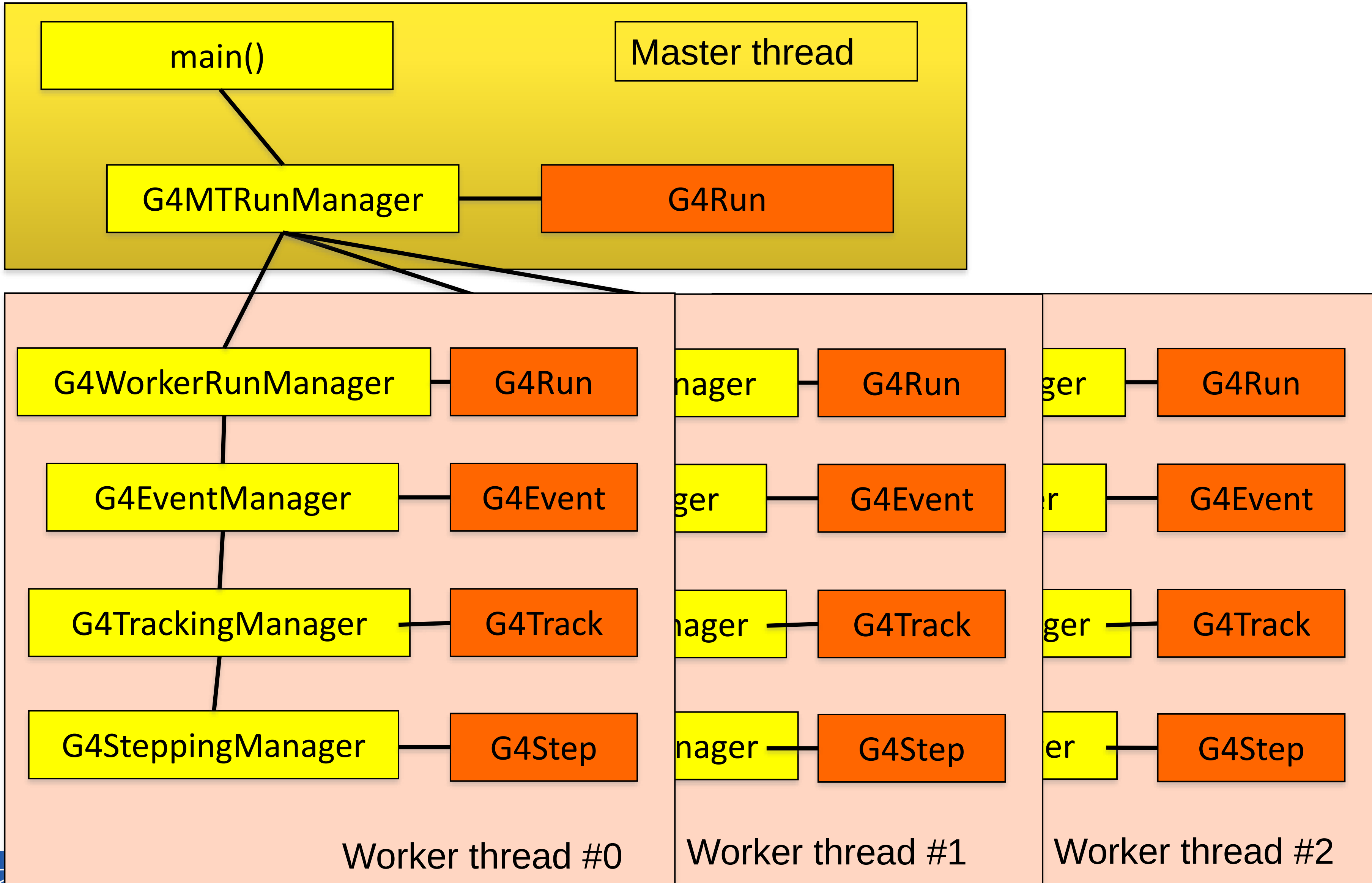
Contents

- User limits
- User classes
- Attaching user information to G4 classes
- Stacking mechanism

Geant4 key classes (sequential mode)



Geant4 key classes (multi-threaded mode)



To use Geant4, you have to…

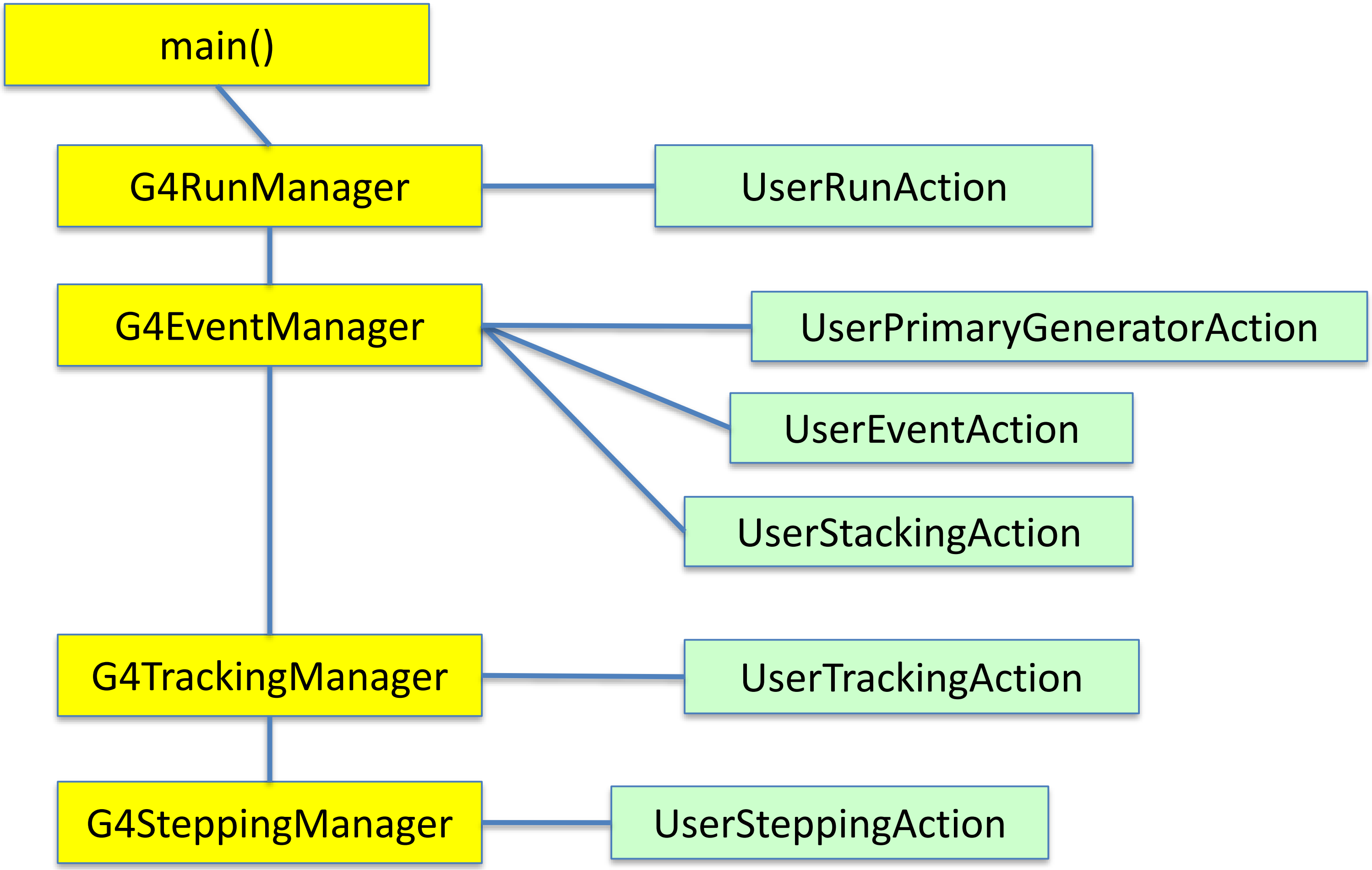
- Geant4 is a toolkit. You have to build an application.
- To make an application, you have to
  - Define your geometrical setup
    - Material, volume
  - Define physics to get involved
    - Particles, physics processes/models
    - Production thresholds
  - Define how an event starts
    - Primary 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

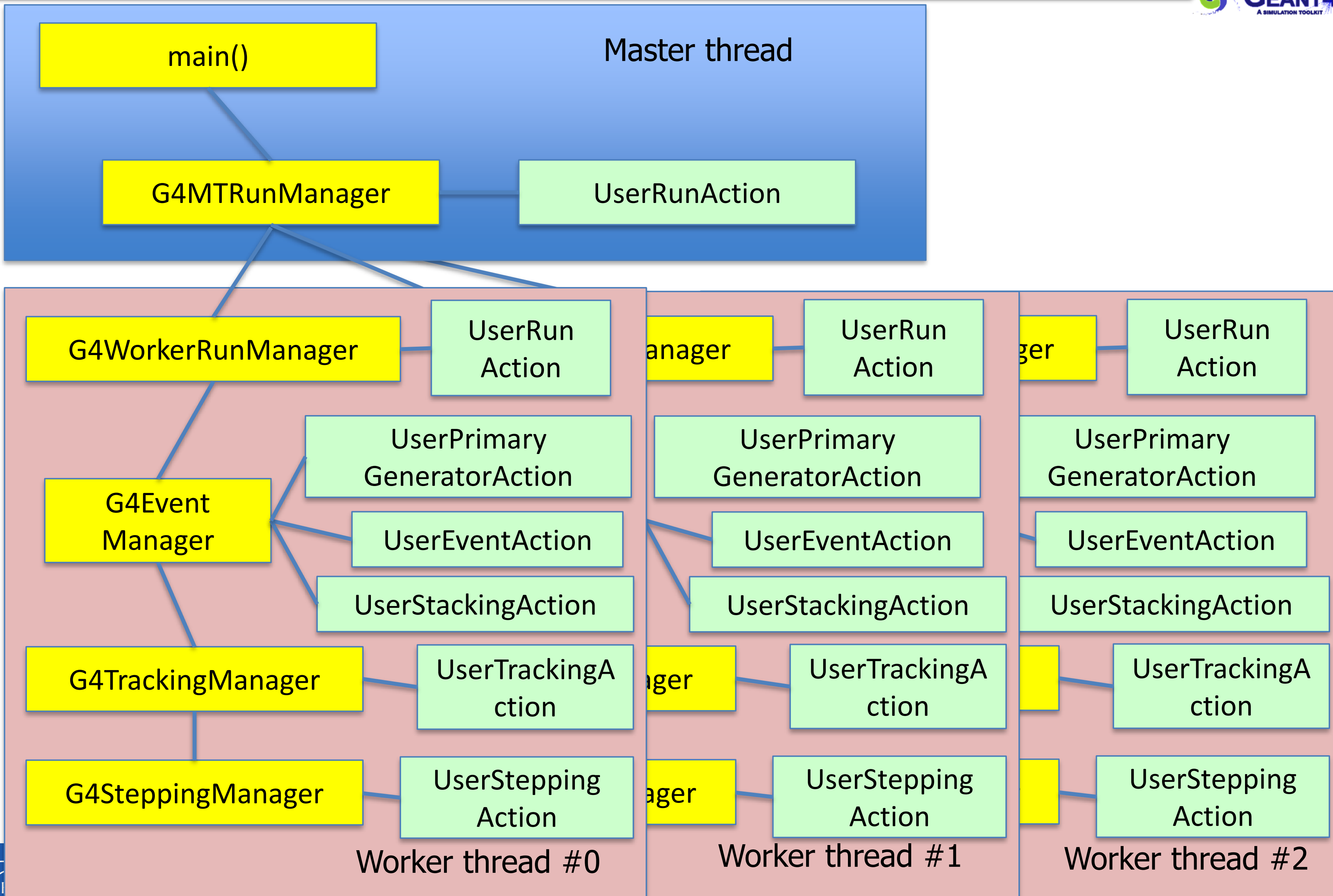
- main()
  - Geant4 does not provide *main().*
- Initialization classes
  - Use G4RunManager::SetUserInitialization() to define.
  - Invoked at the initialization
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
    - G4VUserActionInitialization
    - G4UserThreadInitialization, G4UserTaskThreadInitialization
- Action classes
  - Instantiate them in your G4VUserActionInitialization.
  - Invoked during an event loop
    - G4VUserPrimaryGeneratorAction
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction

Note : classes written in red are mandatory.

User action classes (sequential mode)



User action classes (multi-threaded mode)



UserActionInitialization

- G4VUserActionInitialization has two virtual methods
- BuildForMaster() is invoked once in the master thread

```
MyActionInitialization::BuildForMaster()
{ SetUserAction(new MyRunAction); }
```

- Build() is invoked for each worker thread

```
MyActionInitialization::Build()
{
  SetUserAction(new MyPrimaryGeneratorAction);
  SetUserAction(new MyLocalRunAction);
  SetUserAction(new MyEventAction);
  …
}
```

Contents

- User limits
- User classes
- Attaching user information to G4 classes
- Stacking mechanism

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.
    - Source of memory leak

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 may be used 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.

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->pre/postStepPoint->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 the track is entering to the calorimeter volume
  if( !(thePreRInfo->IsCalorimeter()) && (thePostRInfo->IsCalorimeter()) )
  { theTrack->SetTrackStatus(fSuspend); }
}
```

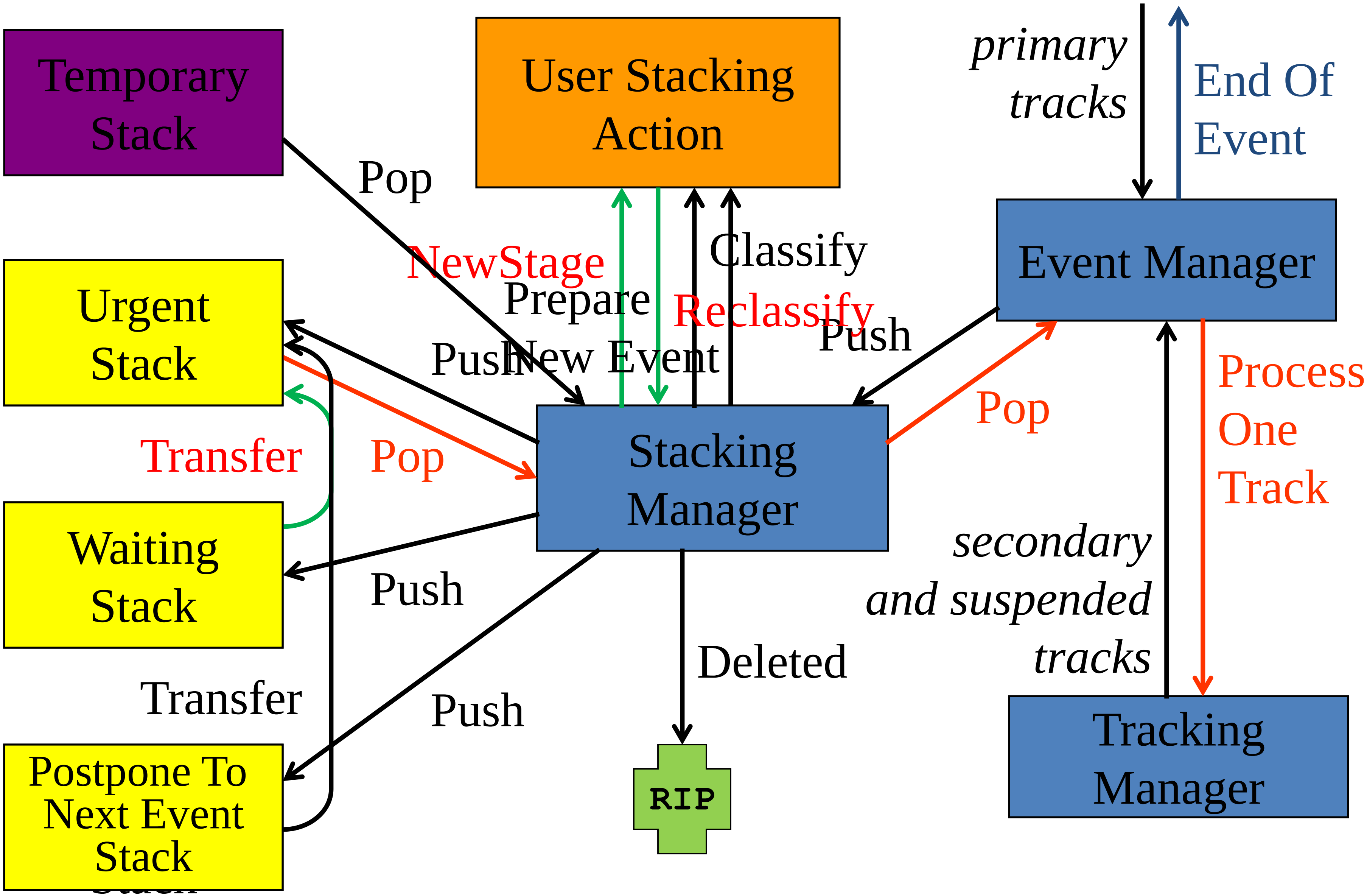
Contents

- User limits
- User classes
- Attaching user information to G4 classes
- Stacking mechanism

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 may arbitrarily 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 UserStackingAction 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

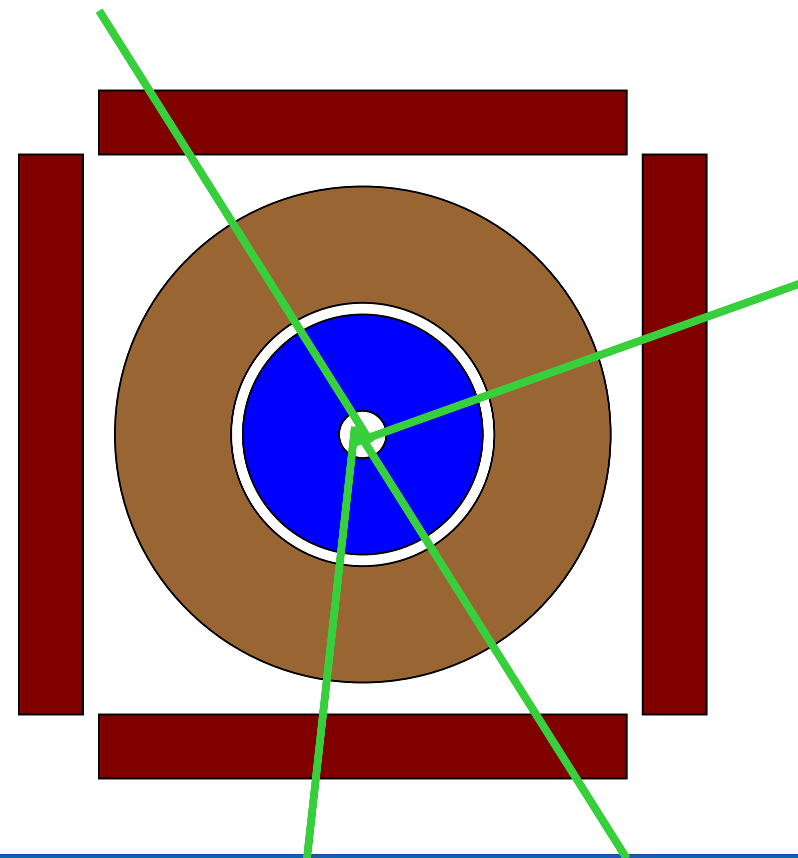


G4UserStackingAction

- User has to implement three methods.
- G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)
  - Invoked every time a new track is pushed to G4StackManager.
  - Classification
    - fUrgent – push into Urgent stack
    - fWaiting – push into Waiting stack
    - fPostpone – push into PostponeToNextEvent stack
    - fKill – delete the track : physics quantities of the track (energy, charge, etc.) are not conserved but completely lost
- 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.

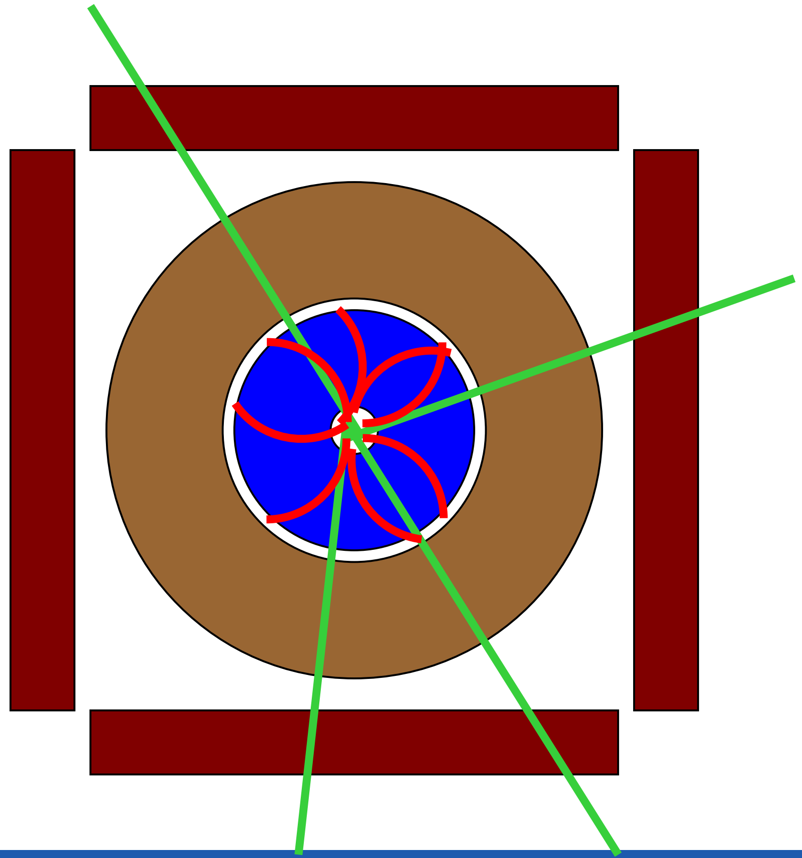
RE05StackingAction

- RE05 has simplified collider detector geometry and event samples of Higgs decaying 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.



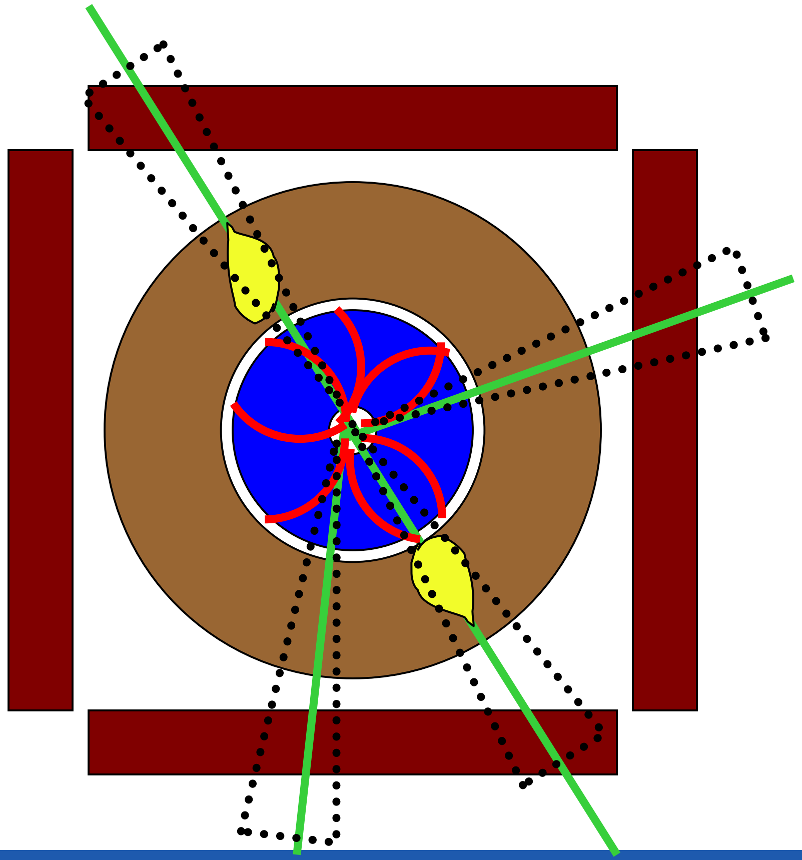
RE05StackingAction

- Stage 1
  - Only primary charged particles are pushed into Urgent stack and all other primaries and secondaries are pushed into Waiting stack.
  - Each 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.



RE05StackingAction

- 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".



Tips of stacking manipulations

- Classify all secondaries as fWaiting until Reclassify() method is invoked.
  - You can simulate all primaries before any secondaries.
- Classify 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 / scintillation 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 by the stacking mechanism, physics quantities of the track (energy, charge, etc.) are not conserved but completely lost.