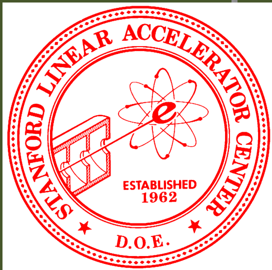
Stanford
Linear
Accelerator
Center



Primary Particle Generation, Tracking & Stacking

Makoto Asai (SLAC)

Geant4 Tutorial Course @ Bordeaux

November 2005

Geant 4

Contents

- Primary particle generation
  - G4VUserPrimaryGeneratorAction
  - Primary vertex and primary particle
  - Built-in primary particle generators
- Tracking mechanism in Geant4
  - Tracking mechanism
  - Track and step statuses
- Attaching user information
- Stacking mechanism

Primary particle generation

User classes

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

Note : classes written in yellow are mandatory.

G4VUserPrimaryGeneratorAction

- This class is one of mandatory user action classes to control the generation of primaries.
  - This class itself should NOT generate primaries but invoke `GeneratePrimaryVertex()` method of primary generator(s) to make primaries.
- Constructor
  - Instantiate primary generator(s)
  - Set default values to it(them)
- GeneratePrimaries() method
  - Randomize particle-by-particle value(s)
  - Set them to primary generator(s)
    - Never use hard-coded UI commands
  - Invoke `GeneratePrimaryVertex()` method of primary generator(s)

Primary vertices and particles

- Primary vertices and primary particles should be stored in G4Event before processing an event.
  - G4PrimaryVertex and G4PrimaryParticle classes
    - These classes don't have any dependency to G4ParticleDefinition nor G4Track.
  - Capability of bookkeeping decay chains
    - Primary particles may not necessarily be particles which can be tracked by Geant4.
- Geant4 provides some concrete implementations of G4VPrimaryGenerator.
  - G4ParticleGun
  - G4HEPEvtInterface
  - G4HEPMCInterface
  - G4GeneralParticleSource

G4ParticleGun

- Concrete implementations of G4VPrimaryGenerator
  - A good example for experiment-specific primary generator implementation
- It shoots one primary particle of a certain energy from a certain point at a certain time to a certain direction.
  - Various set methods are available
  - Intercoms commands are also available
- One of most frequently asked questions is :

  I want "particle shotgun", "particle machinegun", etc.

- Instead of implementing such a fancy weapon, in your implementation of UserPrimaryGeneratorAction, you can
  - Shoot random numbers in arbitrary distribution
  - Use set methods of G4ParticleGun
  - Use G4ParticleGun as many times as you want
  - Use any other primary generators as many times as you want to make overlapping events

G4VUserPrimaryGeneratorAction

```
void T01PrimaryGeneratorAction::
         GeneratePrimaries(G4Event* anEvent)
{ G4ParticleDefinition* particle;
  G4int i = (int)(5.*G4UniformRand());
  switch(i)
  { case 0: particle = positron; break; ... }
  particleGun->SetParticleDefinition(particle);
  G4double pp =
    momentum+(G4UniformRand()-0.5)*sigmaMomentum;
  G4double mass = particle->GetPDGMass();
  G4double Ekin = sqrt(pp*pp+mass*mass)-mass;
  particleGun->SetParticleEnergy(Ekin);
  G4double angle = (G4UniformRand()-0.5)*sigmaAngle;
  particleGun->SetParticleMomentumDirection
          (G4ThreeVector(sin(angle),0.,cos(angle)));
  particleGun->GeneratePrimaryVertex(anEvent);
}
```

- You can repeat this for generating more than one primary particles.

Interfaces to HEPEvt and HepMC

- Concrete implementations of G4VPrimaryGenerator
  - A good example for experiment-specific primary generator implementation
- G4HEPEvtInterface
  - Suitable to /HEPEVT/ common block, which many of (FORTRAN) HEP physics generators are compliant to.
  - ASCII file input
- G4HepMCInterface
  - An interface to HepMC class, which a few new (C++) HEP physics generators are compliant to.
  - ASCII file input or direct linking to a generator through HepMC.

G4GeneralParticleSource

- A concrete implementation of G4VPrimaryGenerator
  - Suitable especially to space applications

```
MyPrimaryGeneratorAction::
         MyPrimaryGeneratorAction()
{ generator = new G4GeneralParticleSource; }
void MyPrimaryGeneratorAction::
         GeneratePrimaries(G4Event* anEvent)
{ generator->GeneratePrimaryVertex(anEvent); }
```

- Detailed description

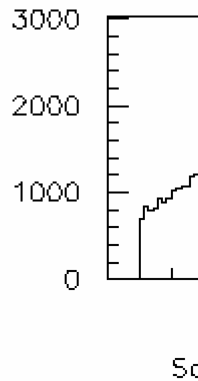
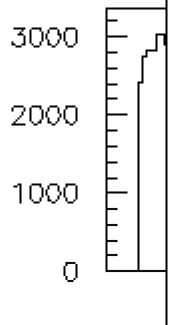
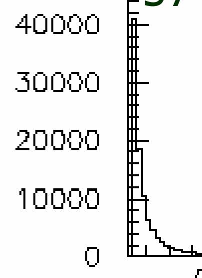
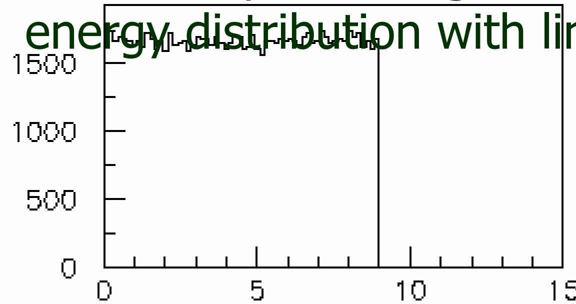
  http://reat.space.qinetiq.com/gps/

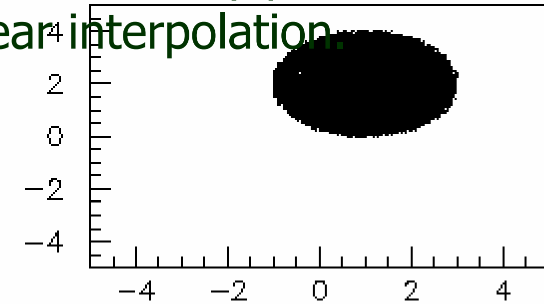
Square

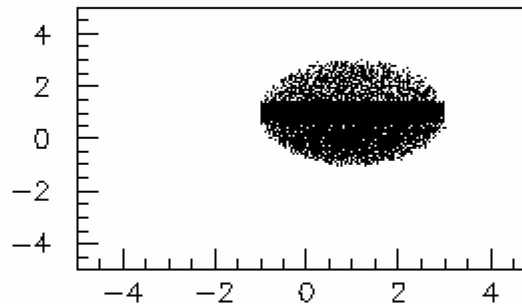
Spheri

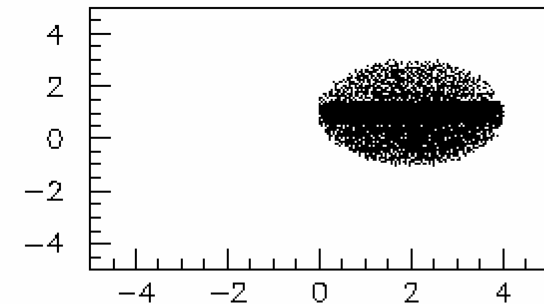
Cylindri
energy

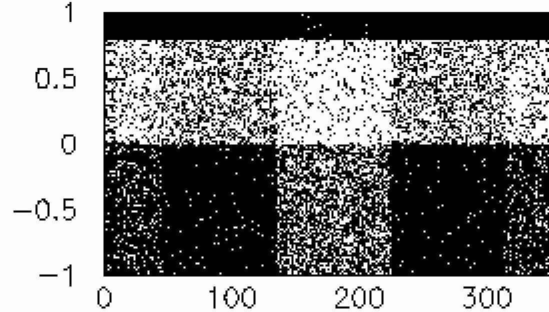
Spherical volume with z biasing, isotropic radiation with theta and phi biasing, integral arbitrary point-wise energy distribution with linear interpolation.

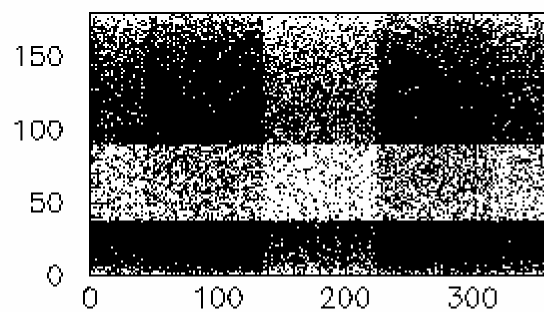





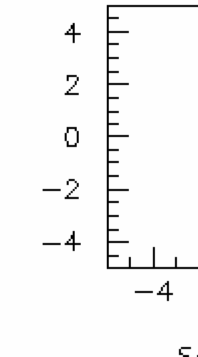
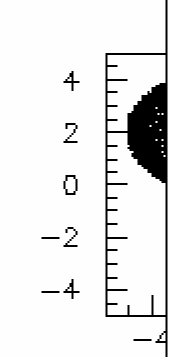
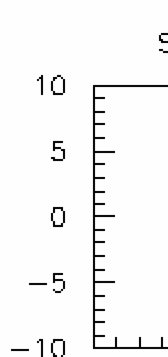
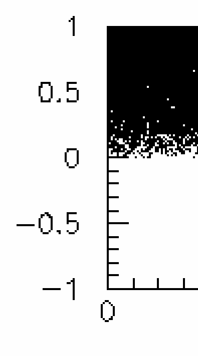
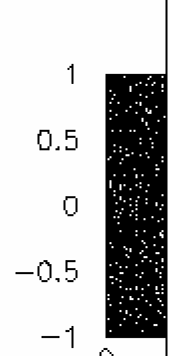
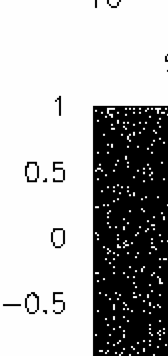










Tracking mechanism in Geant4

Tracking and processes

- Geant4 tracking is general.
  - It is independent to
    - the particle type
    - the physics processes involving 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.

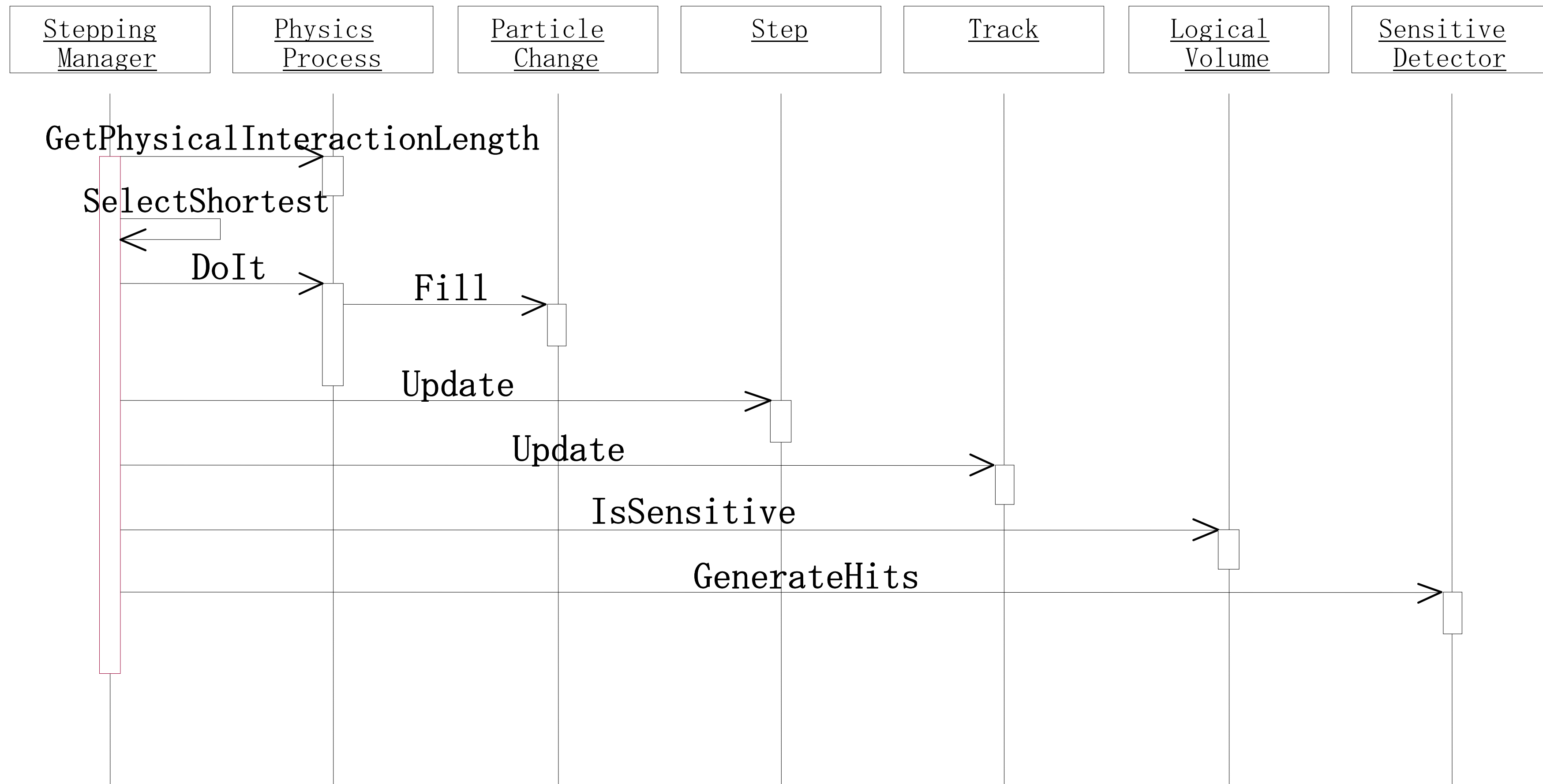
Processes in Geant4

- In Geant4, particle transportation is a process as well, by which 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.
- Each particle type has its own list of applicable processes. At each step, all processes listed are invoked to get proposed physical interaction lengths.
- The process which requires the shortest interaction length (in space-time) limits the step.
- All processes are derived from G4VProcess abstract base class. Each particle type has its individual G4ProcessManager class object which holds a vector of assigned processes.

Process and step

- 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. Celenkov process
  - PostStep (a.k.a. discrete process)
    - e.g. decay on the fly
- Continuous processes contribute cumulatively along the step, while discrete processes contribute selectively (if not exclusively) at the end point of the step.
  - A special process, e.g. shower parameterization process, may take over all continuous and discrete processes.

How Geant4 runs (one step)



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 yellow are artificial, i.e. Geant4 kernel won't set them, but the user can set.

- fAlive
  - Continue the tracking.
- fStopButAlive
  - The track has come to zero kinetic energy, but still AtRest process to occur.
- fStopAndKill
  - The track has lost its identity because it has decayed, interacted or gone beyond 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.

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.

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

Attaching user information to some kernel classes

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
    - 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 some cases 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. These classes keep only the most common quantities.
  - 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 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.

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(…));
```

RE01RegionInformation

- This 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.

Stack management

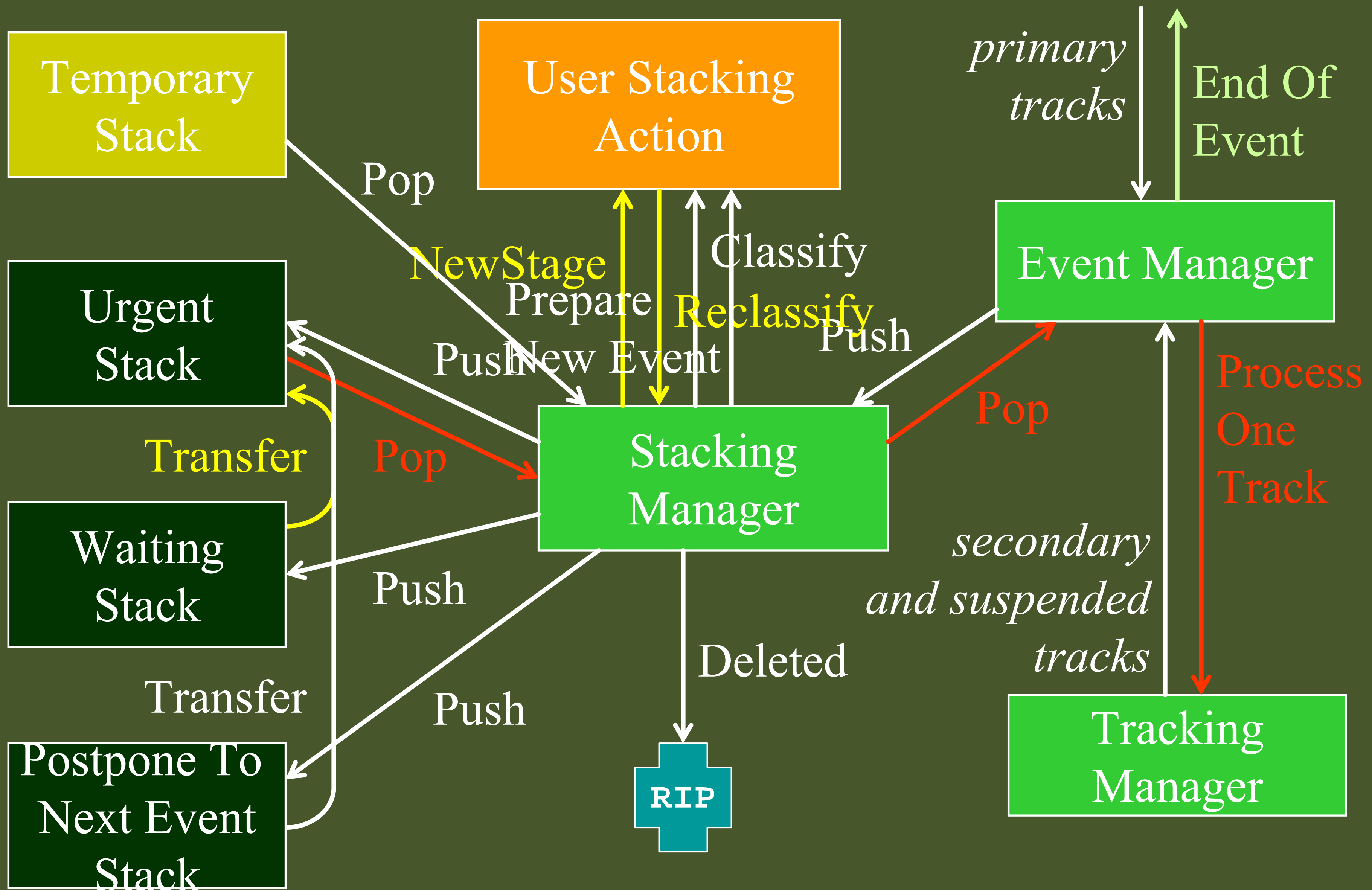
Track stacks in Geant4

- 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" 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.

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 had 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



Tips of stacking manipulations

1. Classify all secondaries as fWaiting until Reclassify() method is invoked.
   - You can simulate all primaries before any secondaries.
2. 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 tracked prior to the original suspended track.
   - Quite effective for Cherenkov lights
3. 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.
4. Classify tracks below a certain energy as fWaiting until Reclassify() method is invoked.
   - You can simulate the event roughly before being bothered by low energy EM showers.