Interaction with the Kernel

XII Seminar on Software for Nuclear, Subnuclear and Applied Physics
May 24-29 2015, Alghero

Optional User classes

Main Classes

Initialization classes

G4VUserDetectorConstruction

G4VUserPhysicsList

Action classes

G4UserPrimaryGeneratorAction

G4UserRunAction
G4UserEventAction
G4UserTrackingAction
G4UserStackingAction
G4UserSteppingAction

Optional classes

✓ Five (or more) concrete base classes can be implemented by the user, adopering the virtual member functions to obtain the control of the simulation at various stages

✓ Each member function of the base classes has a dummy implementation

Optional User classes

Main Classes

Initialization classes

G4VUserDetectorConstruction

G4UserActionInitialization

G4VUserPhysicsList

Action classes

G4UserPrimaryGeneratorAction

G4UserRunAction

G4UserEventAction

G4UserTrackingAction

G4UserStackingAction

G4UserSteppingAction

Optional classes

✓ Five (or more) concrete base classes can be implemented by the user, adopering the virtual member functions to obtain the control of the simulation at various stages

✓ Each member function of the base classes has a dummy implementation

Optional User classes

❖ The user may implement the member functions he desires in his/her derived classes

⟶ E.g. one may want to perform some action at each tracking step

❖ In the Multi-threaded mode the user action classes must be registered to the G4RunManager class, which manages the simulation, via the ActionInitialization class.

1. The ActionInitialization class is initialized, using the `SetUserInitialization` method:

```
runManager->SetUserInitialization(new myUserInitialization)
```

2. The action classes are registered in the ActionInitialization class using the SetUserAction method:

```
runManager->SetUserAction(new myRunAction)
```

myActionInitialization (MT mode)

In MT mode is mandatory for action class instances

- Thread local user actions:

```
void MyActionInitialization::Build() const
{
  //Set mandatory classes
  SetUserAction(new MyPrimaryGeneratorAction());
  // Set optional user action classes
  SetUserAction(new MyEventAction());
  SetUserAction(new MyRunAction());
}
```

- Register the RunAction for the master

```
void MyActionInitialization::BuildForMaster() const
{
  // Set optional user action classes
SetUserAction(new MyRunAction());
}
```

Geant4 Terminology

Keywords:

↓

Event, Run, Step, Track

The Event (G4Event)

➡The Event is the basic unit of the Simulation

➡ At the beginning of the simulation, the primary track are generated and are pushed in a stack

➡ A track is popped up from the stack one-by-one and tracked

- ✤ Secondary tracks are also pushed into the stack
- ✤ When the stack gets empty, the processing of the event is completed

➡**`G4Event`** class represents an event. At the end of the event it has:

- ✤ List of primary vertices and particles (the input)
- ✤ Hits and Trajectory collections (the outputs)

➡**`G4EventManager`** class manages the event.

➡**`G4UserEventAction`** class manages the event.

The Run (G4Run)

➡The Run is a collection of events and in analogy with a real experiment it starts with the command: “Beam On”

During the run the user cannot change:

✓ The Geometrical Setup
✓The adopted Physics Models and Processes
✓ The Source Features

➡The G4RunManager class manages each the processing of each run, by means of the:

✤**G4Run** class
✤**G4RunAction** class

The Step(G4Step)

➡The particle trajectory can be considered as a sequence of segments called “step”

➡**G4Step** represents a step in particle propagation

➡A **G4Step** object stores transient information of the step and is updated every time a process is invoked

The user can retrieve the desired information from each step composing the total track after the step is completed

➡The G4UserSteppingAction is the class devoted to the retrival of information from the step

➡The UserSteppingAction method of this class get the pointer of the G4Step

The Track (G4Track)

The Track is a snapshot of a particle and it is represented by the G4Track class

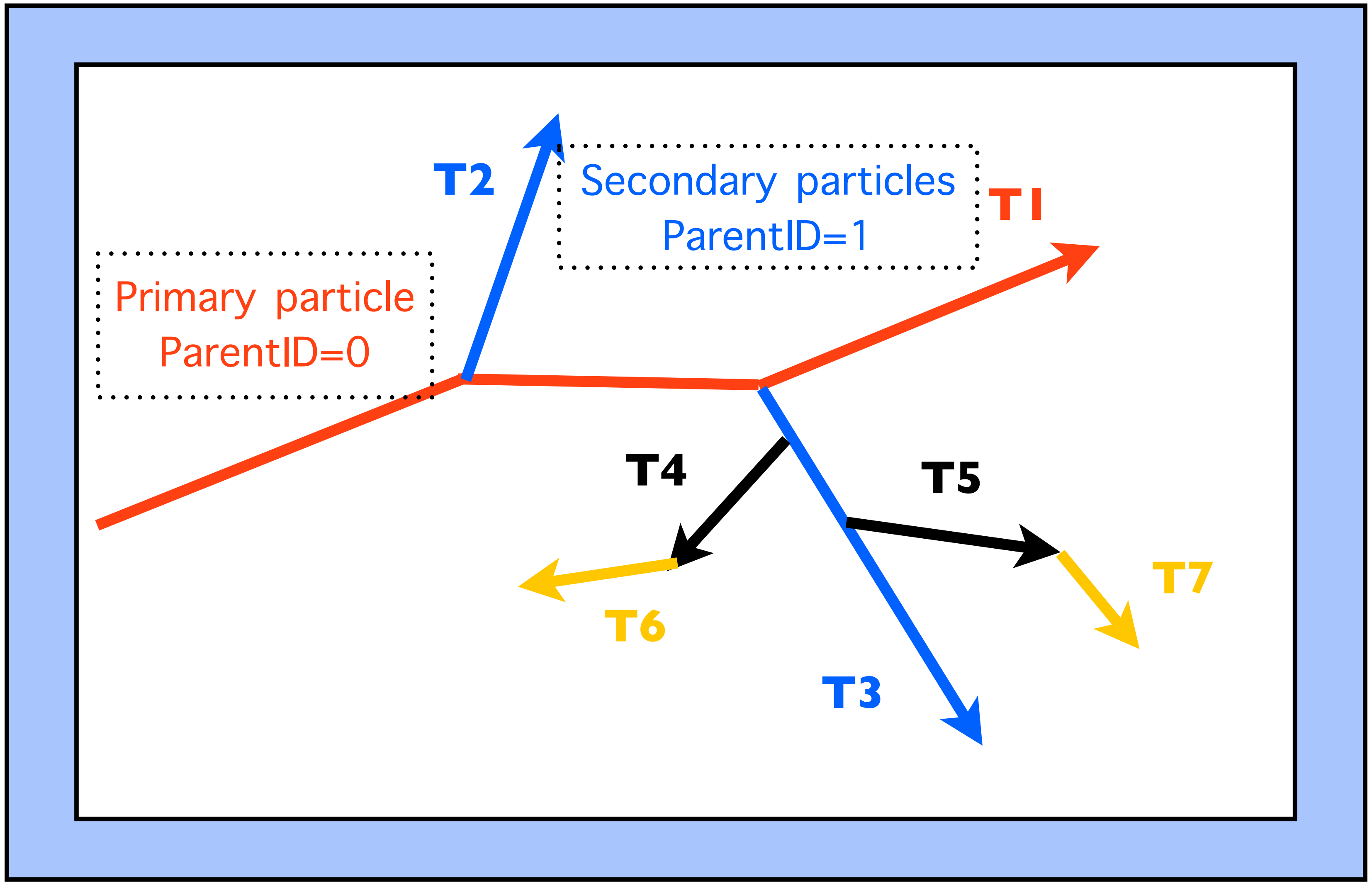
- It keeps current information of the particle (i.e. energy, momentum, position, polarization, ..)
- It is updated after every step

The track object is deleted when:

- It goes outside the world volume
- It disappears in an interaction (decay, inelastic scattering)
- It is slowed down to zero kinetic energy and there are no 'AtRest' processes
- It is manually killed by the user

No track object persists at the end of the event
G4TrackingManager class manages the tracking
G4UserTrackingAction is the optional User hook

Classification of the tracks



Tracking order follows last in first out rule:

T1 ->T3 -> T5 -> T7 -> T4 -> T6 -> T2

To retrieve the information about the track, it is possible to invoke the method GetParentID() (and others) of the G4Track class:

```
G4int parent_ID = aTrack->GetParentID();
```

Tracks

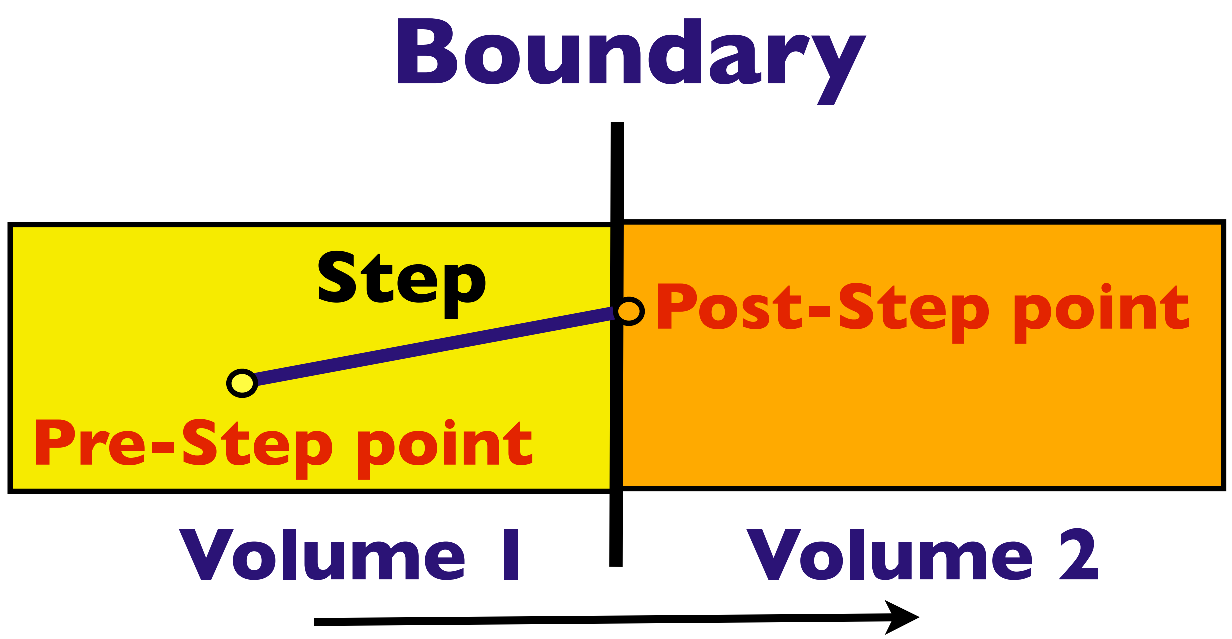
```
*******************************************************************************************
* G4Track Information:   Particle = e-,   Track ID = 87,   Parent ID = 1
*******************************************************************************************

Step#    X(mm)     Y(mm)     Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0 -1.87e+03      5.63     -5.52    0.0326        0        0         0 physicalTreatmentRoom initStep
    1 -1.87e+03      5.85     -4.72     0.032 0.000545    0.924     0.924 physicalTreatmentRoom msc
    2 -1.87e+03      5.92      -3.9    0.0317  0.00036    0.928      1.85 physicalTreatmentRoom msc
    3 -1.87e+03      5.89     -3.65    0.0289  0.00013      0.3      2.15 physicalTreatmentRoom eIoni
    :----- List of 2ndaries - #SpawnInStep=  1(Rest= 0,Along= 0,Post= 1), #SpawnTotal=  1 ---------------
    : -1.87e+03       5.89      -3.65   0.00258                 e-
    :------------------------------------------------------------------ EndOf2ndaries Info ---------------
    4 -1.87e+03      5.81     -2.87    0.0279  0.00104    0.928      3.08 physicalTreatmentRoom msc
    5 -1.87e+03      5.35     -2.11    0.0273 0.000654    0.928      4.01 physicalTreatmentRoom msc
    6 -1.87e+03      5.01     -1.28    0.0248  0.00249    0.928      4.94 physicalTreatmentRoom msc
    7 -1.87e+03      5.03     -0.37    0.0231  0.00163    0.928      5.87 physicalTreatmentRoom msc
    8 -1.87e+03      4.78     0.503     0.022  0.00109    0.928      6.79 physicalTreatmentRoom msc
    9 -1.87e+03      4.64      1.35    0.0202  0.00184    0.928      7.72 physicalTreatmentRoom msc
   10 -1.87e+03      4.68      2.26    0.0181  0.00204    0.928      8.65 physicalTreatmentRoom msc
   11 -1.87e+03      4.63      2.46    0.0165 0.000345    0.231      8.88 physicalTreatmentRoom eIoni
    :----- List of 2ndaries - #SpawnInStep=  1(Rest= 0,Along= 0,Post= 1), #SpawnTotal=  2 ---------------
    : -1.87e+03       4.63       2.46   0.00133                 e-
    :------------------------------------------------------------------ EndOf2ndaries Info ---------------
   12 -1.87e+03       4.6      2.49    0.0125        0   0.0383      8.92 physicalTreatmentRoom eIoni
    :----- List of 2ndaries - #SpawnInStep=  1(Rest= 0,Along= 0,Post= 1), #SpawnTotal=  3 ---------------
    : -1.87e+03        4.6       2.49   0.00402                 e-
    :------------------------------------------------------------------ EndOf2ndaries Info ---------------
```

```
*******************************************************************************************
* G4Track Information:   Particle = e-,   Track ID = 242,   Parent ID = 87
*******************************************************************************************

Step#    X(mm)     Y(mm)     Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0 -1.87e+03       6.1      5.41   0.00138        0        0         0 physicalTreatmentRoom initStep
    1 -1.87e+03      6.11      5.39  0.000253  0.00112   0.0481    0.0481 physicalTreatmentRoom msc
    2 -1.87e+03      6.12      5.39         0 0.000253   0.0088    0.0569 physicalTreatmentRoom eIoni
```

The Step in Geant4



✦The G4Step contains the information about the Pre-Step point and the Post-Step point and the “variation “ of a physical quantity in the step (i.e the energy loss on the step). To access these information or objects instance it is possible to use manu Get method of the G4Step class:

```
G4StepPoint *PreStep=track->GetPreStepPoint()
```

```
PreStepX=PreStep->GetPosition().x()
```

✦For each point (Pre-step and Post-step) the user knows the crossed volume:

✦ 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

✦G4SteppingManager class manages processing a step;
✦G4UserSteppingAction is the optional User hook

The geometry boundary

- To check, if a step ends on a boundary, one may compare if the physical volume of pre and post-step points are equal

One can also use the step status:

- Step Status provides information about the process that restricted the step length

It is attached to the step points: the pre has the status of the previous step, the post of the current step

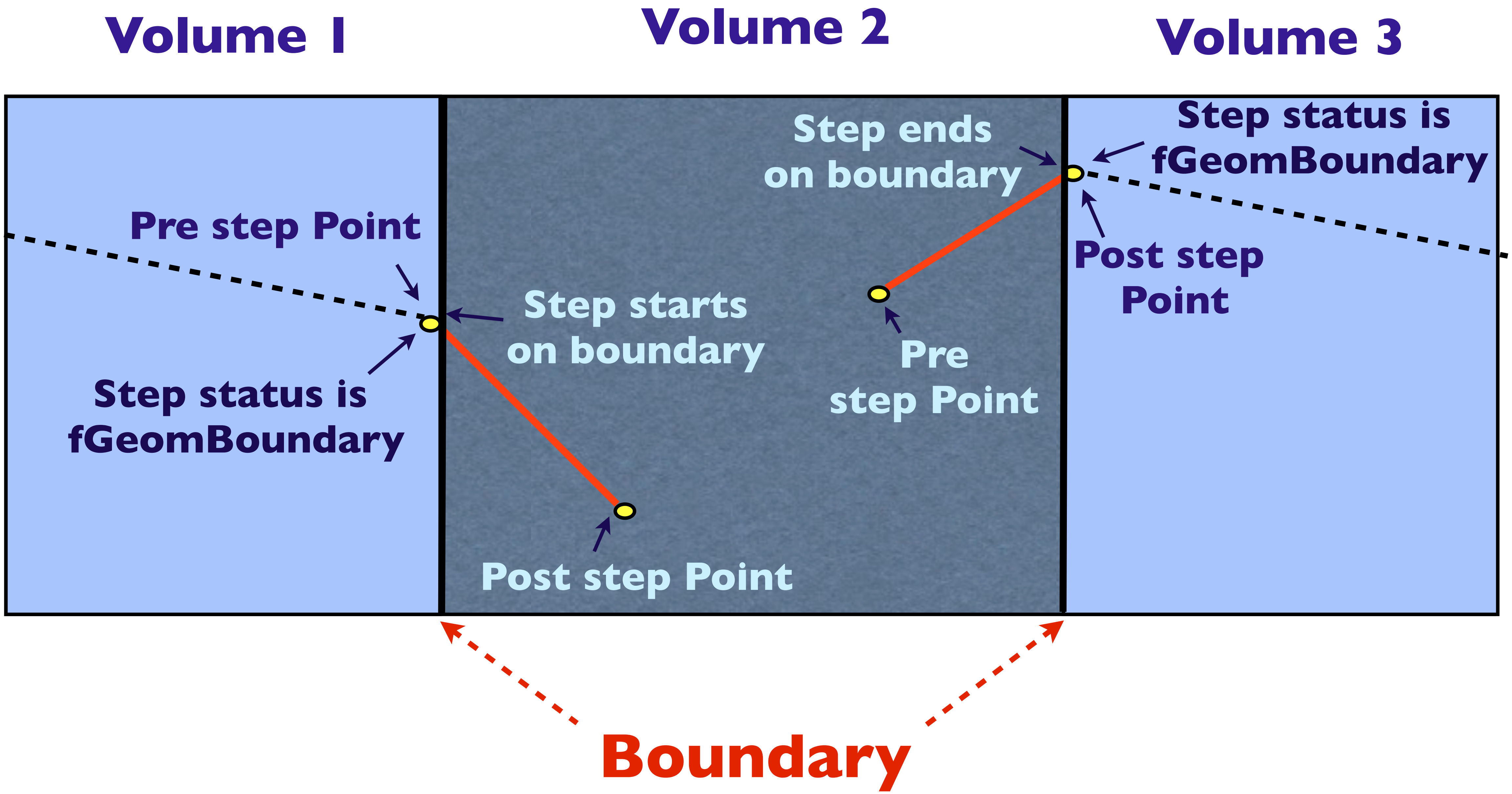
- If the status of POST is “fGeometryBoundary” the step ends on a volume boundary (does not apply to word volume)

- To check if a step starts on a volume boundary you can also use the step status of the PRE-step point

```
if(preStepPoint->GetStepStatus()==fGeomBoundary){
G4cout<<“Step starts on geometry boundary”<<G4endl;}

if(postStepPoint->GetStepStatus()==fGeomBoundary(){
G4cout<<“Step ends on geometry boundary”<<G4endl;}
```

Boundary and step



User Action classes

UserRunAction

➡Used to initialise, analyse, store histogram at run level

➡Has two methods: BeginOfRunAction() and EndOfRunAction() used to retrieve information respectively at the beginning and the end of the run

UserEventAction

➡Retrieve the information at event level, one can apply an event selection

➡ Has two methods BeginOfEventAction() and EndOfEventAction()

UserStackingAction

➡Used to classify the tracks and to decide the priority of tracks.

UserSteppingAction

➡Retrieve the wanted information of the particle at the end of the step, invoking a specific method

Status of the tracks

Track Status	Description
fAlive	The particle is continued to be tracked
fStopButAlive	Kin. Energy = 0, but AtRest process will occur
fStopAndKill	Track has lost identity (has reached world boundary, decayed, ...), Secondaries will be tracked
fKillTrackAndSecondaries	Track and its secondary tracks are killed
fSuspend	Track and its secondary tracks are suspended (pushed to stack)
fPostponeToNextEvent	Track but NOT secondary tracks are postponed to the next event (secondaries are tracked in current event)

Output stream (G4cout)

G4cout is a iostream object defined by Geant4.

- ✤The usage of these objects is exactly the same as the ordinary std::cout except that the output streams will be handled by G4UImanager
- ✤ G4endl is the equivalent of std::endl to end a line

✓Output strings may be displayed on another window or stored in a file

✓One can also use the file streams (std::ofstream) provided by the C++ libraries

```
G4double eKin = aStep -> GetPreStepPoint() -> GetKineticEnergy();
G4double PosX = aStep->GetTrack()->GetPosition().x();
G4double PosY = aStep->GetTrack()->GetPosition().y();
G4double PosZ = aStep->GetTrack()->GetPosition().z();
G4String material= aStep -> GetTrack() -> GetMaterial() -> GetName();
G4String volume=  aStep->GetTrack()->GetVolume()->GetName();

G4cout <<"Kinetic energy"<<" "<< eKin << " "
       <<" X Position "<< " " << PosX << " "
       <<" Y Position "<< " " << PosY << " "
       <<" Z Position "<< " " << PosZ << " "
       <<" Material "<< " " << material << " "
       <<" Volume "<< " "   << volume << " "
       << G4endl;
```

Write an ASCII file

1. Add to the include list of your class the <fstream> header file
   - This will allow to use the C++ libraries for stream on file
2. Put into the class declaration (file .hh) an ofstream (=output file stream) object (or pointer):

   `std::ofstream myFile;`

   In this way, the file object will be visible in all methods of the class
3. Open the file, in the class constructor, or into a specific method:

   myFile.open(“filename.out”, std::ios::trunc);

   - To append data to an existing file, you must specify std::ios::app

```
    std::ofstream myFile("Data.out", std::ios::app);
   myFile  <<      eKin                << '\t' << "   "
           <<    EventID               <<  '\t'<< "   "
           <<   PreStepX               << '\t' << "   "
           <<   PreStepY               << '\t' << "   "
           <<   PreStepZ               << '\t' << "   "
           << G4endl;
```

- This could be for instance the EndOfEventAction() of the G4UserEventAction user class or in the UserSteppingAction class

Output

Data.out

9.86726	0	317.026	-2.42728	-10.5573
9.87157	1	317.026	-3.56108	-4.84368
9.8466	2	317.026	-7.58074	5.06641
9.848	3	317.026	1.00224	5.56123
9.83987	4	317.026	2.6007	2.84975
9.83912	5	317.026	-9.34777	-1.12885
9.85523	6	317.026	0.903539	-5.44811
9.85022	7	317.026	-7.91006	4.18064
9.83825	8	317.026	-6.61794	-3.01946
9.86582	9	317.026	-5.28934	4.04027
9.85413	10	317.026	-5.54807	9.7194
9.83946	11	317.026	14.144	-1.08552
9.86147	12	317.026	0.045469	1.76874
9.8513	13	317.026	0.562574	-3.44896
9.82307	14	317.026	4.52927	-7.51171
9.87058	15	317.026	-4.71805	-6.84709
9.84838	16	317.026	9.18044	2.37358
9.77604	17	317.026	7.93081	-2.60215
9.86598	18	317.026	-5.56149	-0.43634
9.85405	19	317.026	5.32018	0.742721
9.85778	20	317.026	-10.236	4.99975
9.88042	21	317.026	7.52852	0.867449
9.84586	22	317.026	1.80864	4.68796

...

G4analysis tool

Data analysis in Geant4

Basic classes for data analysis have recently been implemented in Geant4 (g4analysis)

- ✦Support for histograms and ntuples
- ✦Output in ROOT, XML, HBOOK and CSV (ASCII)

The resulting files can be opened and analyzed by tools such as: Gnuplot, Excel, OpenOffice, Matlab, Origin, ROOT, PAW,...

Appropriate only for easy/quick analysis: for advanced tasks, the user must write his/her own code and to use an external analysis tool

Native Geant4 analysis classes

- A basic analysis interface is available in Geant4 for histograms (1D and 2D) and ntuples
  - ➡Make life easier because they are MT-compliant (no need to worry about the interference of threads)
- Unique interface to support different output formats ROOT, AIDA XML, CSV and HBOOK
  - ➡Code is the same, just change one line to switch from one to an other
- Everything done via the public analysis interface G4AnalysisManager
  - ➡Singleton class: Instance()
  - ➡UI commands available for creating histograms at run-time and setting their properties

- Selection of output format is hidden in a user-defined .hh file
- All the rest of the code unchanged
  - ➡Unique interface

```
#ifndef MyAnalysis_h
#define MyAnalysis_h 1

#include "g4root.hh"
//#include "g4xml.hh"
//#include "g4csv.hh"  // can be used only
with ntuples

#endif
```

Open file and book histograms

```
#include "MyAnalysis.hh"

void MyRunAction::BeginOfRunAction(const G4Run* run)
{
  // Get analysis manager
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->SetVerboseLevel(1);
  man->SetFirstHistoId(1);

  // Creating histograms
  man->CreateH1("h","Title", 100, 0., 800*MeV);
  man->CreateH1("hh","Title",100,0.,10*MeV);

 // Open an output file
  man->OpenFile("myoutput");
}
```

Fill histograms and close

```
#include "MyAnalysis.hh"
void MyEventAction::EndOfEventAction(const G4Run* aRun)
{
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->FillH1(1, fEnergyAbs);
  man->FillH1(2, fEnergyGap);
}
void MyRunAction::EndOfRunAction(const G4Run* aRun)
{
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->Write();
  man->CloseFile();
}
MyRunAction::~MyRunAction()
{
  delete G4AnalysisManager::Instance();
}
```

Analysis and UI commands

UI support available, to change parameters (e.g. file name) at run-time:

```
/analysis/setFileName name               # Set name for the
                                         histograms and ntuple file
/analysis/setHistoDirName name           # Set name for the
                                         histograms directory
/analysis/setNtupleDirName name          # Set name for the
                                         histograms directory
/analysis/setActivation true|false       # Set activation option
/analysis/verbose level                  # Set verbose level

/analysis/h1/create
   name title [nbin min max] [unit] [fcn] [binScheme]     #
Create 1D histogram
```

Ntuples

G4tool supports ntuples

➡Any number of ntuples, each with any number of columns
➡The content can be int/float/double

For more complex tasks (e.g. full functionality of ROOT TTrees) have to link ROOT directly

Similar strategy as for histograms. Access happens through the common interface G4AnalysisManager

- Saved on the same output file with histograms

Book ntuples

```
#include "MyAnalysis.hh"
void MyRunAction::BeginOfRunAction(const G4Run* run)
{
  // Get analysis manager
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man-> SetFirstNtupleId(1);

  // Creating ntuple
  man->CreateNtuple("name", "Title");
  man->CreateNtupleDColumn("Eabs");
  man->CreateNtupleDColumn("Egap");
  man->FinishNtuple();

  man->CreateNtuple("name2","title2");
  man->CreateNtupleIColumn("ID");
  man->FinishNtuple();
}
```

Fill ntuples

```
#include "MyAnalysis.hh"
void MyEventAction::EndOfEventAction(const G4Run* aRun)
{
  G4AnalysisManager* man =
G4AnalysisManager::Instance();
  man->FillNtupleDColumn(1, 0, fEnergyAbs);
  man->FillNtupleDColumn(1, 1, fEnergyGap);
  man->AddNtupleRow(1);

  man->FillNtupleIColumn(2, 0, fID);
  man->AddNtupleRow(2);

}
```