Geant4 Detector Response

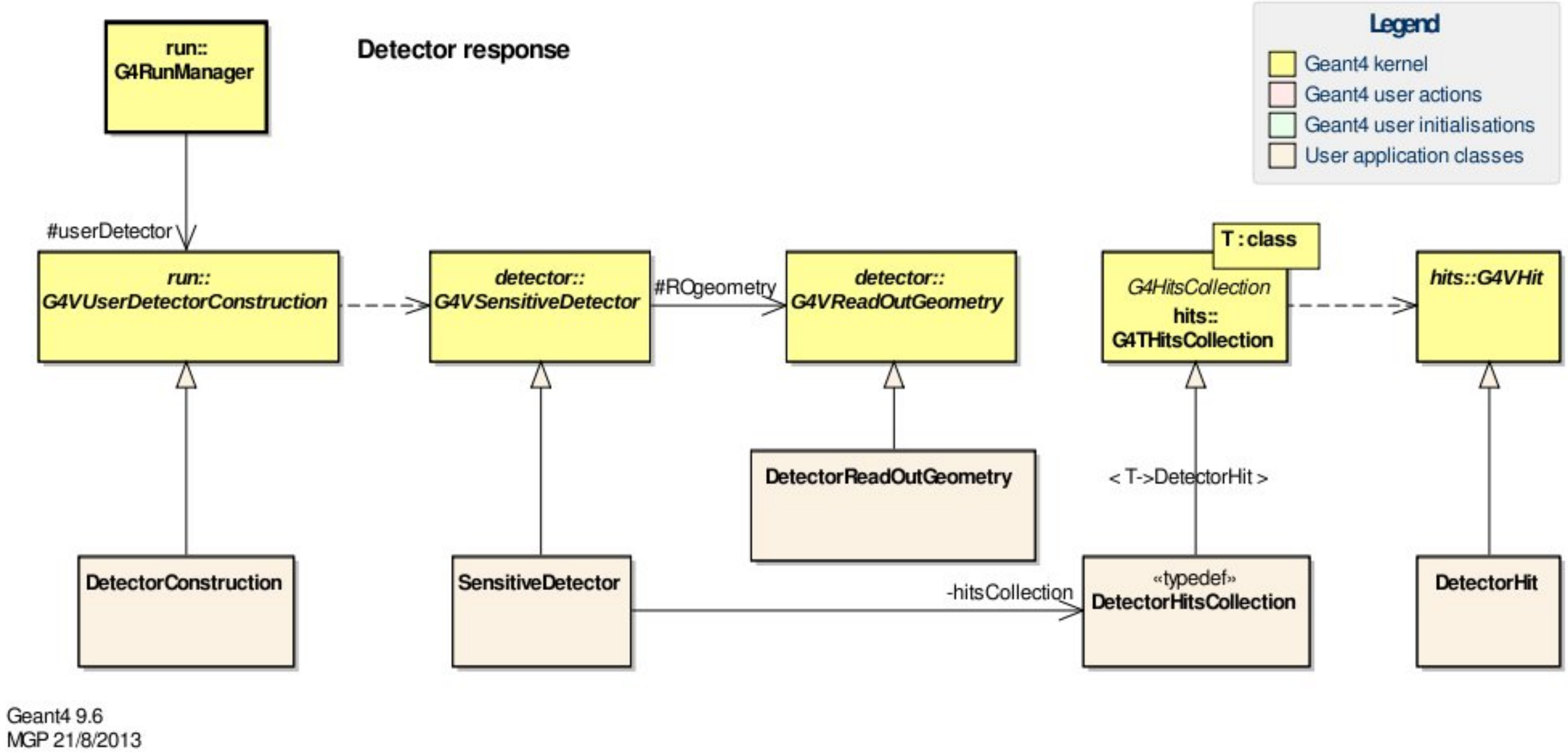
Geant4 and its Application to HEP and Astrophysics
December 5, 2022

Sunanda Banerjee

Extraction of Useful information

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”
  - The user needs to add a bit of code to extract useful information
- There are three ways:
  - Use built-in scoring commands
    - Most commonly-used physics quantities are available
  - Use scorers in the tracking volume
    - Create scores for each event
    - Create own Run class to accumulate scores
  - Assign G4VSensitiveDetector to a volume to generate “hit”
    - Use user hooks (G4UserEventAction, G4UserRunAction) to get event / run summary
- The user may also use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
  - The user has full access to almost all information

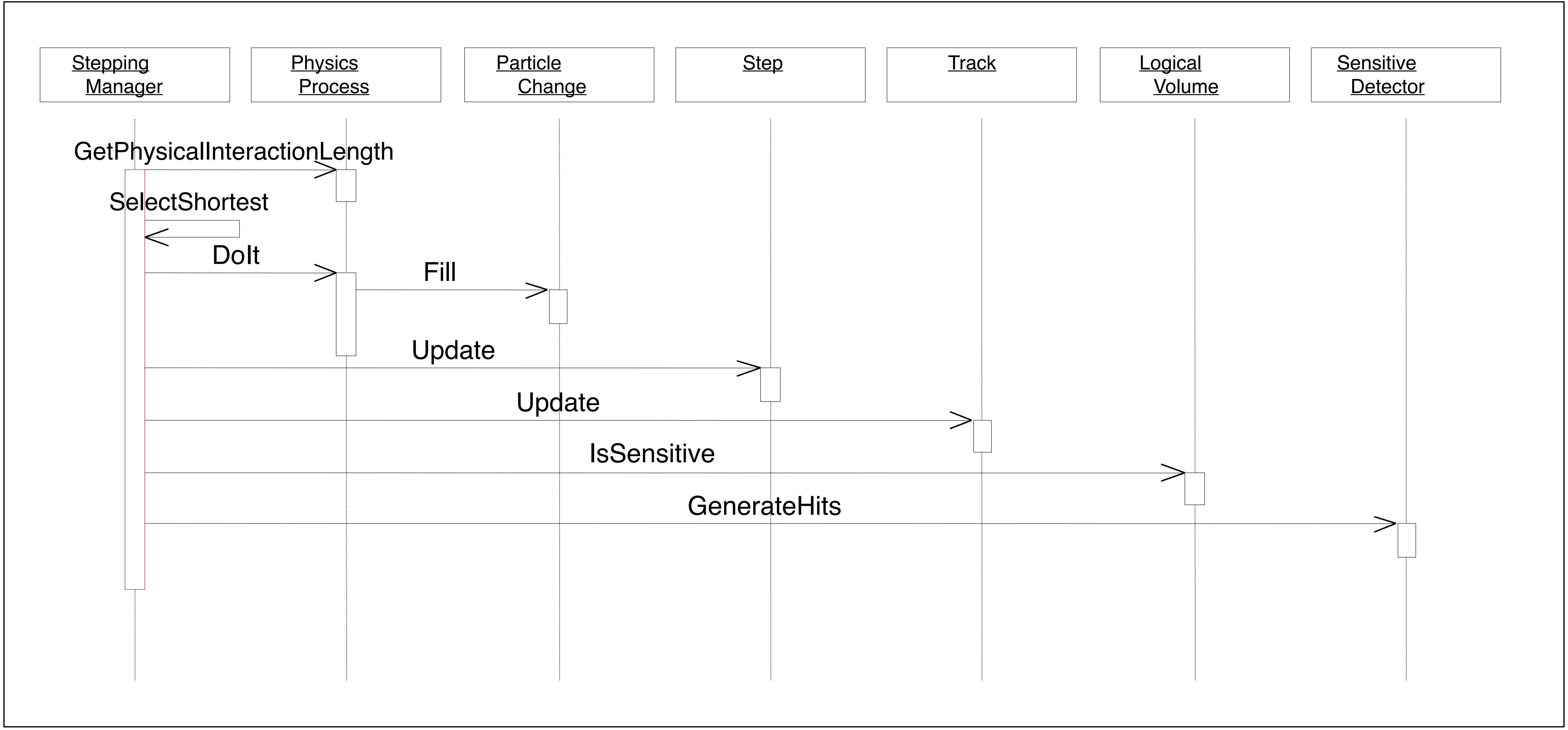
Detector Response



- Please refer to the earlier lecture for DetectorConstruction and ReadOutGeometry

Sensitive Detector

- A G4VSensitiveDetector object can be assigned to a G4LogicalVolume
- In case a step takes place in a logical volume that has a G4VSensitiveDetector object, this G4VSensitiveDetector is invoked with the current G4Step object



Defining a Sensitive Detector

- The basic strategy

```
G4LogicalVolume* myLogCalor = ……;
G4VSensetiveDetector* pSensetivePart = new MyDetector(“/
  mydet”);
G4SDManager* SDMan = G4SDManager::GetSDMpointer();
SDMan->AddNewDetector(pSensitivePart);
myLogCalor->SetSensitiveDetector(pSensetivePart);
```

- Each detector object must have a unique name
  - Some logical volumes can share one detector object
  - More than one detector object can be made from one detector class with different detector name
  - One logical volume cannot have more than one detector object. But, one detector object can generate more than one kind of hits
    - e.g. a double-sided silicon micro-strip detector can generate hits for each side separately

Hit Collection, Hit Map

- G4VHitsCollection is the common abstract base class of both G4THitsCollection and G4THitsMap

- G4THitsCollection is a template vector class to store pointers of objects of one concrete hit class type
  - A hit class (deliverable of G4VHit abstract base class) should have its own identifier (e.g. cell ID)

  - G4THitsCollection requires the user to implement their own hit class

- G4THitsMap is a template map class that stores keys (typically cell ID, i.e. copy number of the volume) with pointers of objects of one type

  - Objects may not be those of the hit class
    - All of the currently provided scorer classes use G4THitsMap with simple double

  - Since G4THitsMap is a template, it can be used by the sensitive detector class to store hits

Sensitive Detector and Hit

- Each Logical Volume can have a pointer to a sensitive detector
  - Then this volume becomes sensitive

- Hit is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive region of your detector

- A sensitive detector creates hit(s) using the information given in the G4Step object. The user has to provide his/her own implementation of the detector response

- Hit objects, which are still the user’s class objects, are collected in a G4Event object at the end of an event

Hit Class

- Hit is a user-defined class derived from G4VHit

- The user can store various types of information by implementing one's own concrete Hit class. For example:
  - Position and time of the step
  - Momentum and energy of the track
  - Energy deposition of the step
  - Geometrical information
  - or any combination of above

- Hit objects of a concrete hit class must be stored in a dedicated collection which is instantiated from G4THitsCollection template class

- The collection is associated with a G4Event object via G4HCofThisEvent

- Hits are accessible as collections:
  - through G4Event at the end of the event
    - to be used for analyzing an event
  - through G4SDManager during processing an event
    - to be used for event filtering

Implementation of Hit class

```
#include "G4VHit.hh"
class MyHit : public G4VHit
{
  public:
      MyHit(some_arguments);
      virtual ~MyHit();
      virtual void Draw();
      virtual void Print();
  private:
      // some data members
  public:
      // some set/get methods
};

#include “G4THitsCollection.hh”
typedef G4THitsCollection<MyHit> MyHitsCollection;
```

Sensitive Detector class

- The sensitive detector is a user-defined class derived from the class G4VSensitiveDetector

```
#include "G4VSensitiveDetector.hh"
#include "MyHit.hh"
class G4Step;
class G4HCofThisEvent;
class MyDetector : public G4VSensitiveDetector
{
  public:
     MyDetector(G4String name);
     virtual ~MyDetector();
     virtual void Initialize(G4HCofThisEvent*HCE);
     virtual G4bool ProcessHits(G4Step*aStep,
                           G4TouchableHistory*ROhist);
     virtual void EndOfEvent(G4HCofThisEvent*HCE);
  private:
     MyHitsCollection * hitsCollection;
     G4int collectionID;
};
```

Sensitive Detector

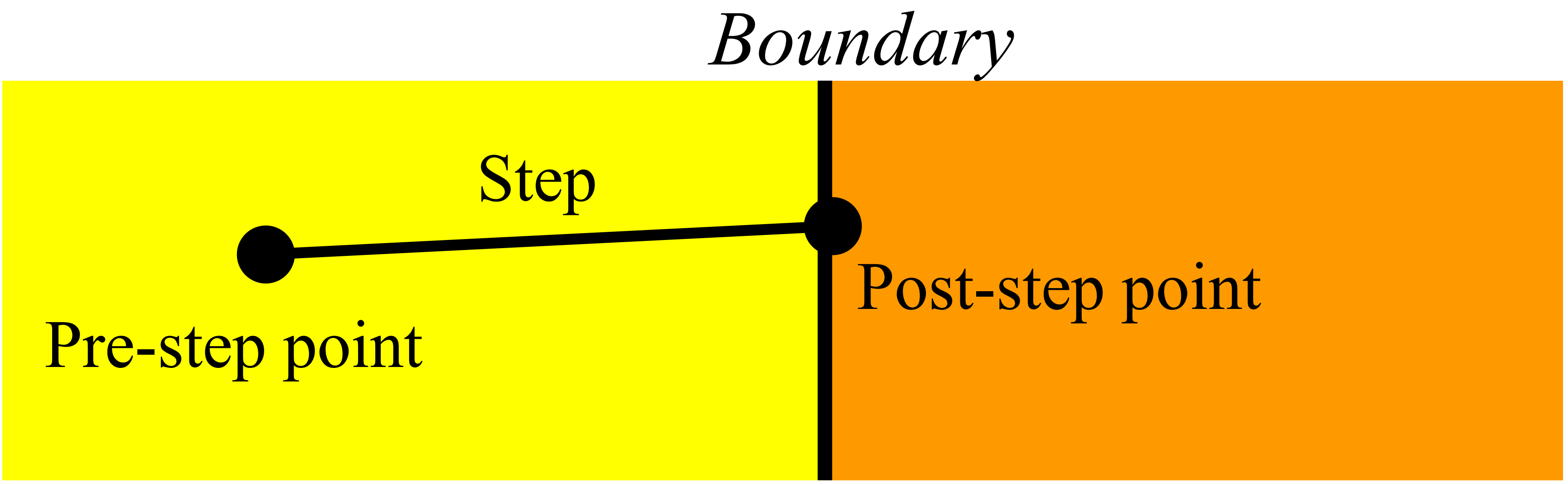
- A tracker detector typically generates a hit for every single step of every single (charged) track
  - A tracker hit typically contains
    - Position and time
    - Energy deposition of the step
    - Track identifier
    - Some cell identifier

- A calorimeter detector typically generates a hit for every cell and accumulates energy deposition in each cell for all steps of all tracks
  - A calorimeter hit typically contains
    - Sum of deposited energy
    - Some timing information
    - Cell Identifier

- The user can instantiate more than one object for one sensitive detector class. Each object should have its unique detector name
  - For example, each of the two sets of detectors can have its dedicated sensitive detector objects. But, their functionalities of them are exactly the same as each other so they can share the same class. See examples/extended/analysis/A01 as an example

Step

- A Step has two points and also “delta” information of a particle (energy loss on the step, time-of-flight spent by the step, etc.)
- Each point knows the volume (and material). 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
- Note that the user must get the volume information from the “PreStepPoint”



Implementation of Sensitive Detector - 1

```
MyDetector::MyDetector(G4String detector_name)
              :G4VSensitiveDetector(detector_name),
               collectionID(-1)
{
  collectionName.insert(“collection_name");
}
```

- In the constructor, the name of the hits collection which is handled by this sensitive detector is to be defined

- In case the sensitive detector generates more than one kind of hits (e.g. anode and cathode hits separately), all collection names need to be defined

Implementation of Sensitive Detector - 2

```
void MyDetector::Initialize(G4HCofThisEvent*HCE)
{
  if(collectionID<0) collectionID = GetCollectionID(0);
  hitsCollection = new MyHitsCollection
        (SensitiveDetectorName,collectionName[0]);
  HCE->AddHitsCollection(collectionID,hitsCollection);
}
```

- Initialize() method is invoked at the beginning of each event.

- Get the unique ID number for this collection
  - GetCollectionID() is a heavy operation. It should not be used for every event
  - GetCollectionID() is available after this sensitive detector object is constructed and registered to G4SDManager. Thus, this method cannot be invoked in the constructor of this detector class

- The hits collection(s) are to be instantiated and then attached to the G4HCofThisEvent object given in the argument

- In the case of a calorimeter-type detector, hits for all calorimeter cells may be instantiated with zero energy depositions, and then inserted into the collection

Implementation of Sensitive Detector - 3

```
G4bool MyDetector::ProcessHits
  (G4Step*aStep,G4TouchableHistory*ROhist)
{
  MyHit* aHit = new MyHit();
  ...
  // some set methods
  ...
  hitsCollection->insert(aHit);
  return true;
}
```

- This ProcessHits() method is invoked for every step in the volume(s) where this sensitive detector is assigned

- In this method, generate a hit corresponding to the current step (for tracking detector), or accumulate the energy deposition of the current step to the existing hit object where the current step belongs (for calorimeter detector)

- The geometry information is to be collected (e.g. copy number) from “PreStepPoint”

- Currently, the returned boolean value is not used

Implementation of Sensitive Detector - 4

```
void MyDetector::EndOfEvent(G4HCofThisEvent*HCE)  {;}
```

- This method is invoked at the end of processing an event
  - It is invoked even if the event is aborted
  - It is invoked before UserEndOfEventAction

Step Point and Touchable

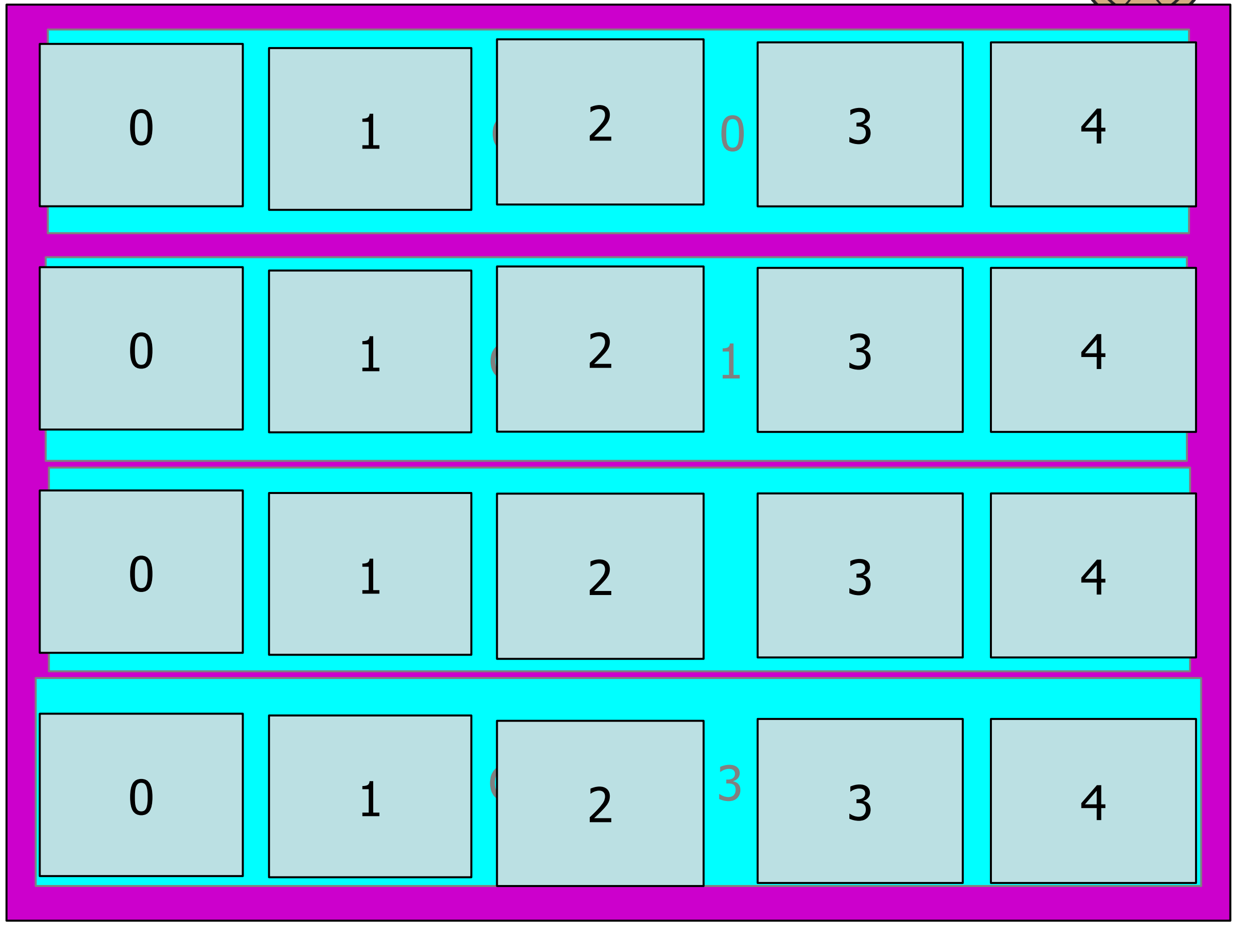
- As mentioned already, G4Step has two G4StepPoint objects as its starting and ending points. All the geometrical information of the particular step should be taken from "PreStepPoint"
  - Geometrical information associated with G4Track is identical to "PostStepPoint"

- Each G4StepPoint object has
  - Position in the world coordinate system
  - Global and local time
  - Material
  - G4TouchableHistory for geometrical information

- The G4TouchableHistory object is a vector of information for each geometrical hierarchy
  - copy number
  - translation/rotation to its mother

- Since release 4.0, *handles* (or *smart-pointers*) to touchable are intrinsically used. Touchables are reference counted

Copy Number



- Suppose a calorimeter is made of 4x5 cells
  - and it is implemented by two levels of replica
- In reality, there is only one physical volume object for each level. Its position is parameterized by its copy number
- To get the copy number of each level for the cell when the step belongs to two cells,
  - geometrical information in G4Track is identical to that in “PostStepPoint”
  - the user cannot get the correct copy number for "PreStepPoint" if one directly accesses the physical volume
- The touchable is to be used to get the proper copy number, transform matrix, etc.

Touchable

- G4TouchableHistory has information on the geometrical hierarchy of the point.

```
G4Step* aStep;
G4StepPoint* preStepPoint = aStep->GetPreStepPoint();
G4TouchableHistory* theTouchable =
   (G4TouchableHistory*)(preStepPoint->GetTouchable());
G4int copyNo = theTouchable->GetVolume()->GetCopyNo();
G4int motherCopyNo
          = theTouchable->GetVolume(1)->GetCopyNo();
G4int grandMotherCopyNo
          = theTouchable->GetVolume(2)->GetCopyNo();
G4ThreeVector worldPos = preStepPoint->GetPosition();
G4ThreeVector localPos = theTouchable->GetHistory()
   ->GetTopTransform().TransformPoint(worldPos);
```

G4HCofThisEvent

- A G4Event object has a G4HCofThisEvent object at the end of (successful) event processing. G4HCofThisEvent object stores all hits collections made within the event.

  - Pointer(s) to the collections may be NULL if collections are not created in the particular event

  - Hit collections are stored by pointers of the G4VHitsCollection base class. Thus, one has to cast them into types of individual concrete classes

  - The index number of a Hits collection is unique and unchanged for a run. The index number can be obtained by

    ```
    G4SDManager::GetCollectionID("detName/colName");
    ```

    - The index table is also stored in G4Run

Usage of G4HCofThisEvent

```
void MyEventAction::EndOfEventAction(const G4Event* evt)  {
  static int CHCID = -1;
  If(CHCID<0) CHCID = G4SDManager::GetSDMpointer()
     ->GetCollectionID("myDet/collection1");
  G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
  MyHitsCollection* CHC = 0;
  if (HCE) {
     CHC = (MyHitsCollection*)(HCE->GetHC(CHCID)); }
  if (CHC) {
    int n_hit = CHC->entries();
    G4cout<<“My detector has ”<<n_hit<<" hits."<<G4endl;
    for (int i1=0;i1<n_hit;i1++) {
      MyHit* aHit = (*CHC)[i1];
      aHit->Print();
    }
  }
}
```

When to invoke GetCollectionID()?

- Which is the better place to invoke G4SDManager::GetCollectionID() in a user event action class, in its constructor or in the BeginOfEventAction()?

- It actually depends on the user's application
  - Note that the construction of sensitive detectors (and thus the registration of their hits collections to SDManager) takes place when the user issues RunManager::Initialize(), and thus the user’s geometry is constructed.

- In case user's EventAction class should be instantiated before Runmanager::Initialize() (or /run/initialize command), GetCollectionID() should not be in the constructor of EventAction.

- While, if the user has nothing to do to Geant4 before RunManager::Initialize(), this initialize method can be hard-coded in the main() before the instantiation of EventAction (e.g. exampleA01), so that GetCollectionID() could be in the constructor

Additional Slides