Geant 4

Version 10.0-p01

Scoring II

Makoto Asai (SLAC)

Geant4 Tutorial Course

Contents

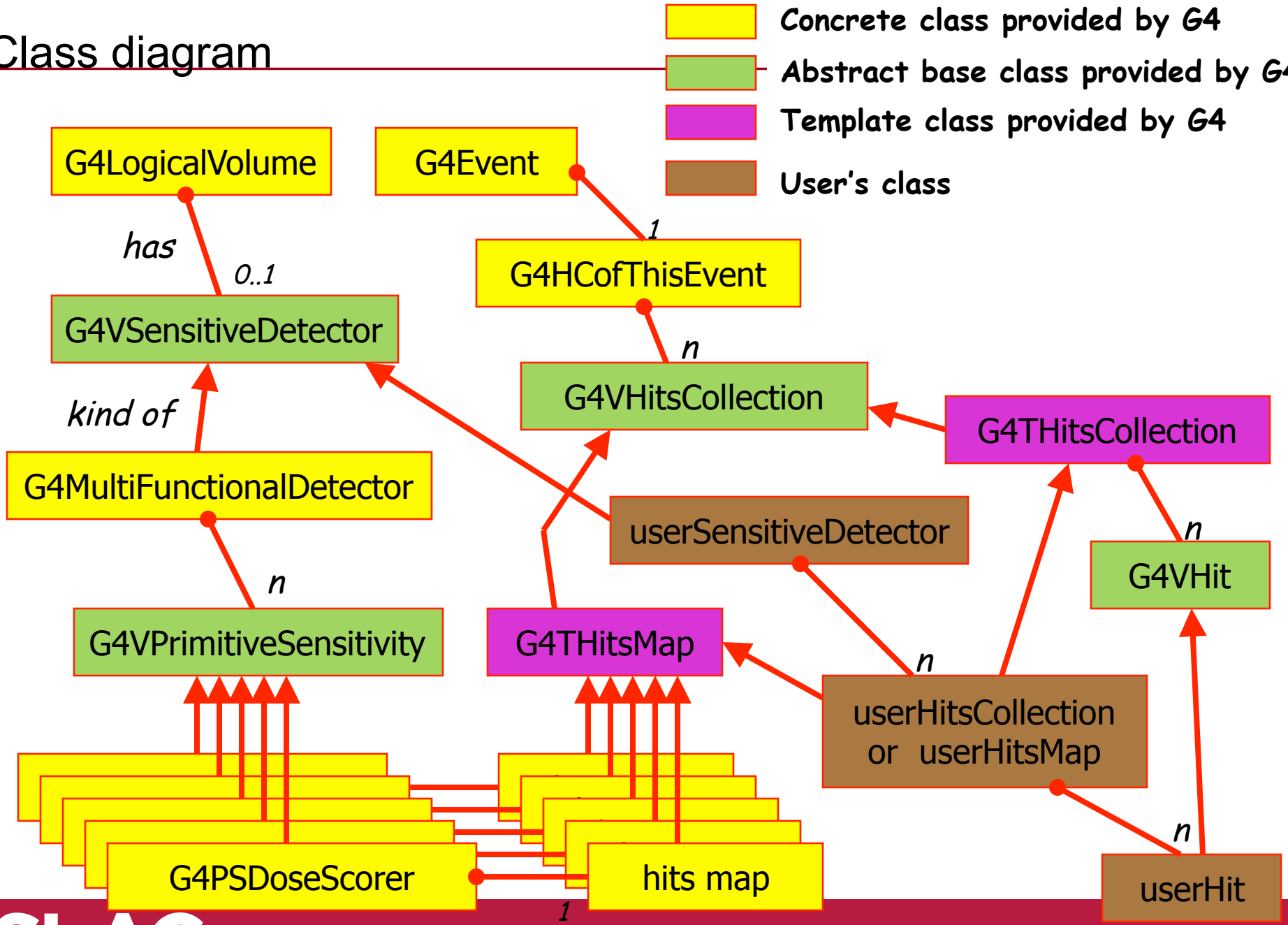
- Define scorers in the tracking volume
- Accumulate scores for a run
- Sensitive detector vs. primitive scorer
- Basic structure of detector sensitivity
- Sensitive detector and hit
- Touchable
- Use of G4HCofThisEvent class

Geant 4

Version 10.0-p01

Define scorers to the tracking volume

Class diagram

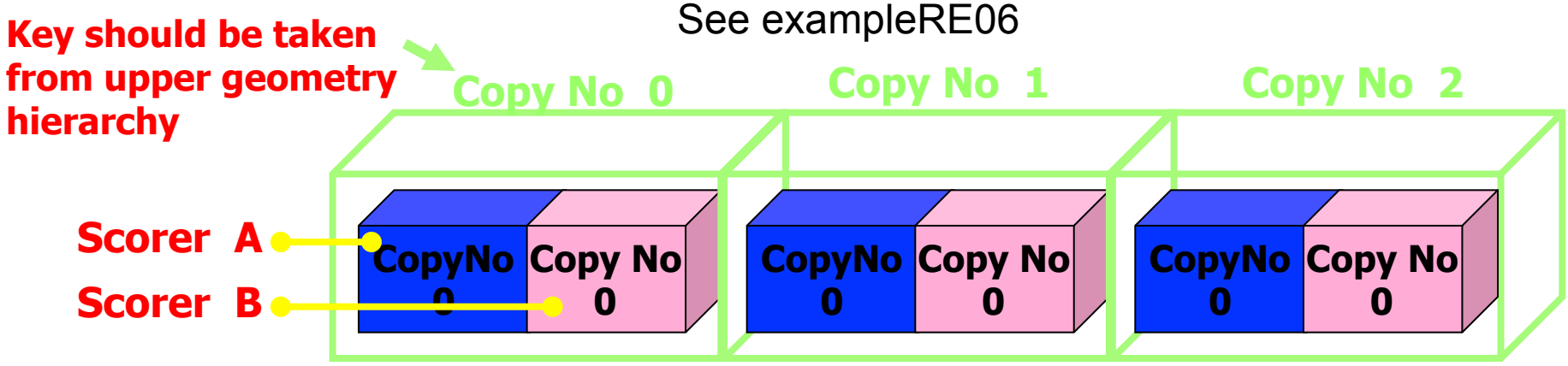


Keys of G4THitsMap

- All provided primitive scorer classes use G4THitsMap<G4double>.
- By default, the copy number is taken from the physical volume to which G4MultiFunctionalDetector is assigned.
  - If the physical volume is placed only once, but its (grand-)mother volume is replicated, use the "depth" argument of the constructor of the primitive scorer to indicate the level where the copy number should be taken.

    e.g. G4PSCellFlux(G4String name, G4String& unit, G4int depth=0)

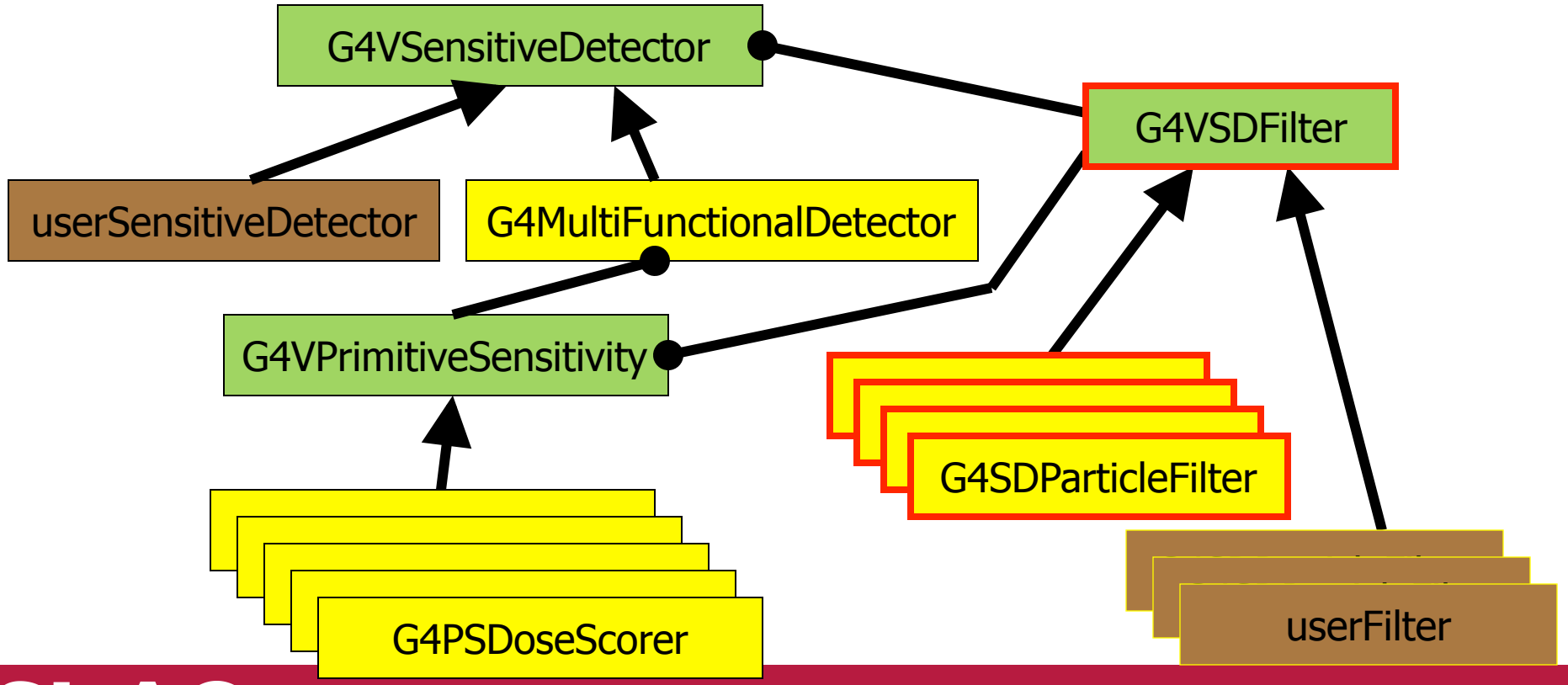
    See exampleRE06



  - If your indexing scheme is more complicated (e.g. utilizing copy numbers of more than one hierarchies), you can override the virtual method GetIndex() provided for all the primitive scorers.

G4VSDFilter

- G4VSDFilter can be attached to G4VSensitiveDetector and/or G4VPrimitiveSensitivity to define which kinds of tracks are to be scored.
  - E.g., surface flux of protons can be scored by G4PSFlatSurfaceFlux with a filter that accepts protons only.



example…

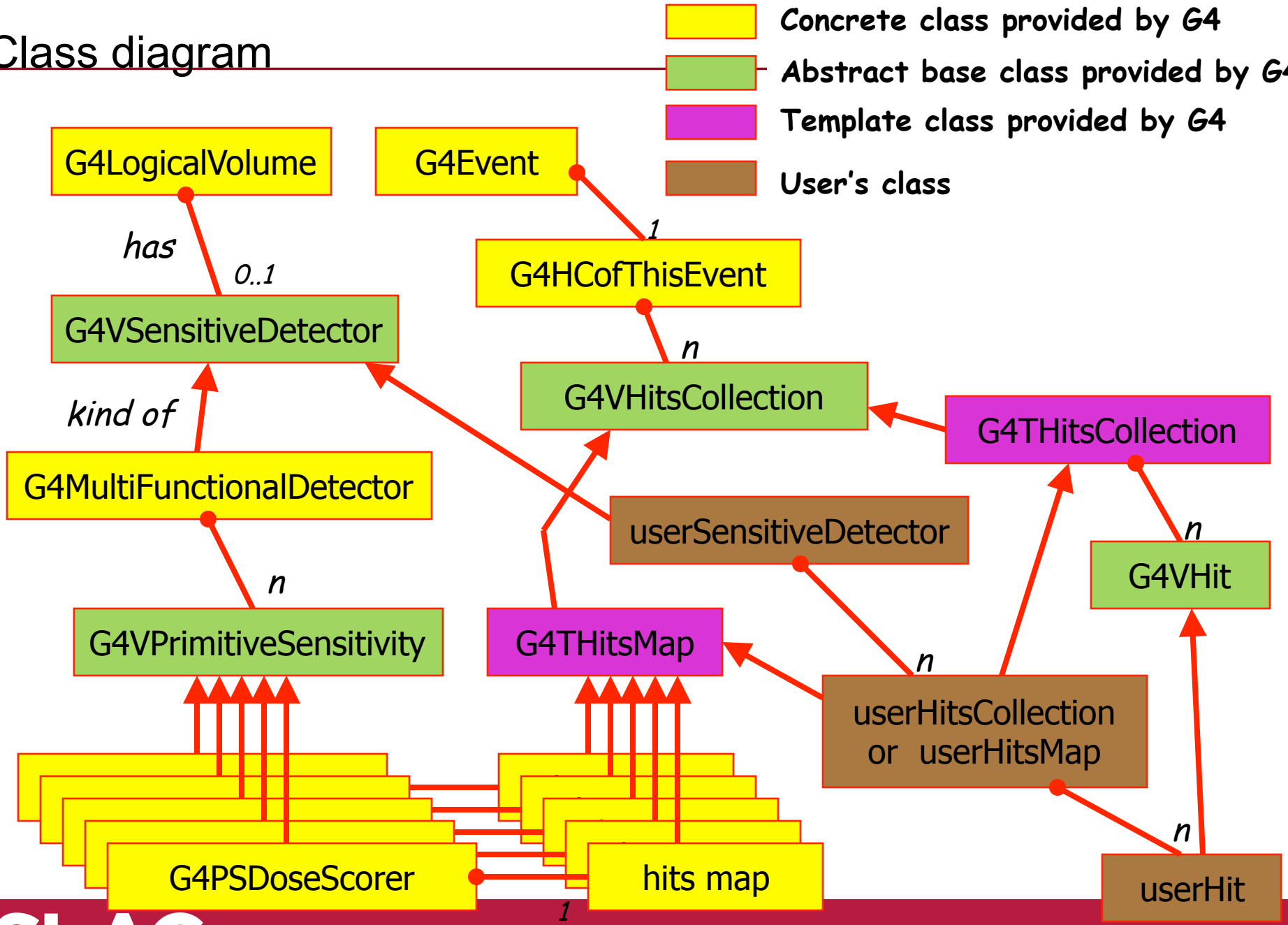
```
MyDetectorConstruction::ConstructSDandField()
{
    G4MultiFunctionalDetector* myScorer = new G4MultiFunctionalDetector("myCellScorer");
    G4VPrimitiveSensitivity* totalSurfFlux = new G4PSFlatSurfaceFlux("TotalSurfFlux");
    myScorer->Register(totalSurfFlux);
    G4VPrimitiveSensitivity* protonSufFlux = new G4PSFlatSurfaceFlux("ProtonSurfFlux");
    G4VSDFilter* protonFilter = new G4SDParticleFilter("protonFilter");
    protonFilter->Add("proton");
    protonSurfFlux->SetFilter(protonFilter);
    myScorer->Register(protonSurfFlux);

    SetSensitiveDetector("myLogVol",myScorer);
}
```

Geant 4

Version 10.0-p01

Accumulate scores for a run

Class diagram



Score == G4THitsMap<G4double>

- At the end of successful event, G4Event has a vector of G4THitsMap as the scores.
- Create your own Run class derived from G4Run, and implement two methods.
- RecordEvent(const G4Event*) method is invoked in the worker thread at the end of each event. You can get all output of the event so that you can accumulate the sum of an event to a variable for entire run.
- Merge(const G4Run*) method of the run object in the master thread is invoked with the pointer to the thread-local run object when an event loop of that thread is over. You should merge thread-local scores to global scores.
- Your run class object should be instantiated in GenerateRun() method of your *UserRunAction*.
  - This UserRunAction must be instantiated both for master and worker threads.

Customized run class

```
#include “G4Run.hh”
#include “G4Event.hh”
#include “G4THitsMap.hh”
Class MyRun : public G4Run
{
  public:
    MyRun();
    virtual ~MyRun();
    virtual void RecordEvent(const G4Event*);
    virtual void Merge(const G4Run*);
  private:
    G4int nEvent;
    G4int  totalSurfFluxID,  protonSurfFluxID,  totalDoseID;
    G4THitsMap<G4double> totalSurfFlux;
    G4THitsMap<G4double> protonSurfFlux;
    G4THitsMap<G4double> totalDose;
  public:
    … access methods …
};
```

Implement how you accumulate event data

Implement how you merge thread-local scores

Customized run class

```
MyRun::MyRun()
{
  G4SDManager* SDM = G4SDManager::GetSDMpointer();
  totalSurfFluxID = SDM->GetCollectionID("myCellScorer/TotalSurfFlux");
  protonSurfFluxID = SDM->GetCollectionID("myCellScorer/ProtonSurfFlux");
  totalDoseID = SDM->GetCollectionID("myCellScorer/TotalDose");
}
```

name of G4MultiFunctionalDetector object

name of G4VPrimitiveSensitivity object

Customized run class

```
void MyRun::RecordEvent(const G4Event* evt)
{
  G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
  G4THitsMap<G4double>* eventTotalSurfFlux
          = (G4THitsMap<G4double>*)(HCE->GetHC(totalSurfFluxID));
  G4THitsMap<G4double>* eventProtonSurfFlux
          = (G4THitsMap<G4double>*)(HCE->GetHC(protonSurfFluxID));
  G4THitsMap<G4double>* eventTotalDose
          = (G4THitsMap<G4double>*)(HCE->GetHC(totalDoseID));
  totalSurfFlux += *eventTotalSurfFlux;
  protonSurfFlux += *eventProtonSurfFlux;
  totalDose += *eventTotalDose;

  G4Run::RecordEvent(evt);
}
```

No need of loops.
+= operator is provided !

Don't forget to invoke base class method!

Customized run class

```
void MyRun::Merge(const G4Run* run)
{
  const MyRun* localRun = static_cast<const MyRun*>(run);

  totalSurfFlux += *(localRun . totalSurfFlux);
  protonSurfFlux += *(localRun . protonSurfFlux);
  totalDose += *(localRun . totalDose);

  G4Run::Merge(run);
}
```

Cast !

No need of loops.
+= operator is provided !

Don't forget to invoke base class method!

RunAction with customized run

```
G4Run* MyRunAction::GenerateRun()
{ return (new MyRun()); }
void MyRunAction::EndOfRunAction(const G4Run* aRun)
{
  const MyRun* theRun = static_cast<const MyRun*>(aRun);

  if( IsMaster() )
  {
    // … analyze / record / print-out your run summary
    // MyRun object has everything you need …
   }
}
```

IsMaster() returns true for the RunAction object assigned to the master thread. (also returns true for sequential mode)

- As you have seen, to accumulate event data, you do NOT need
  - Event / tracking / stepping action classes
- All you need are your Run and RunAction classes.

Geant 4

Version 10.0-p01

Sensitive detector
vs.
primitive scorer

Extract useful information

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”.
  - You have to add a bit of code to extract information useful to you.
- There are three ways:
  - 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
- You may also use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
  - You have full access to almost all information
  - Straight-forward, but do-it-yourself

Sensitive detector vs. primitive scorer

Sensitive detector

- You have to implement your own detector and hit classes.
- One hit class can contain many quantities. A hit can be made for each individual step, or accumulate quantities.
- Basically one hits collection is made per one detector.
- Hits collection is relatively compact.

Primitive scorer

- Many scorers are provided by Geant4. You can add your own.
- Each scorer accumulates one quantity for an event.
- G4MultiFunctionalDetector creates many collections (maps), i.e. one collection per one scorer.
- Keys of maps are redundant for scorers of same volume.

I would suggest to :

- Use primitive scorers
  - if you are not interested in recording each individual step but accumulating some physics quantities for an event or a run, and
  - if you do not have to have too many scorers.
- Otherwise, consider implementing your own sensitive detector.

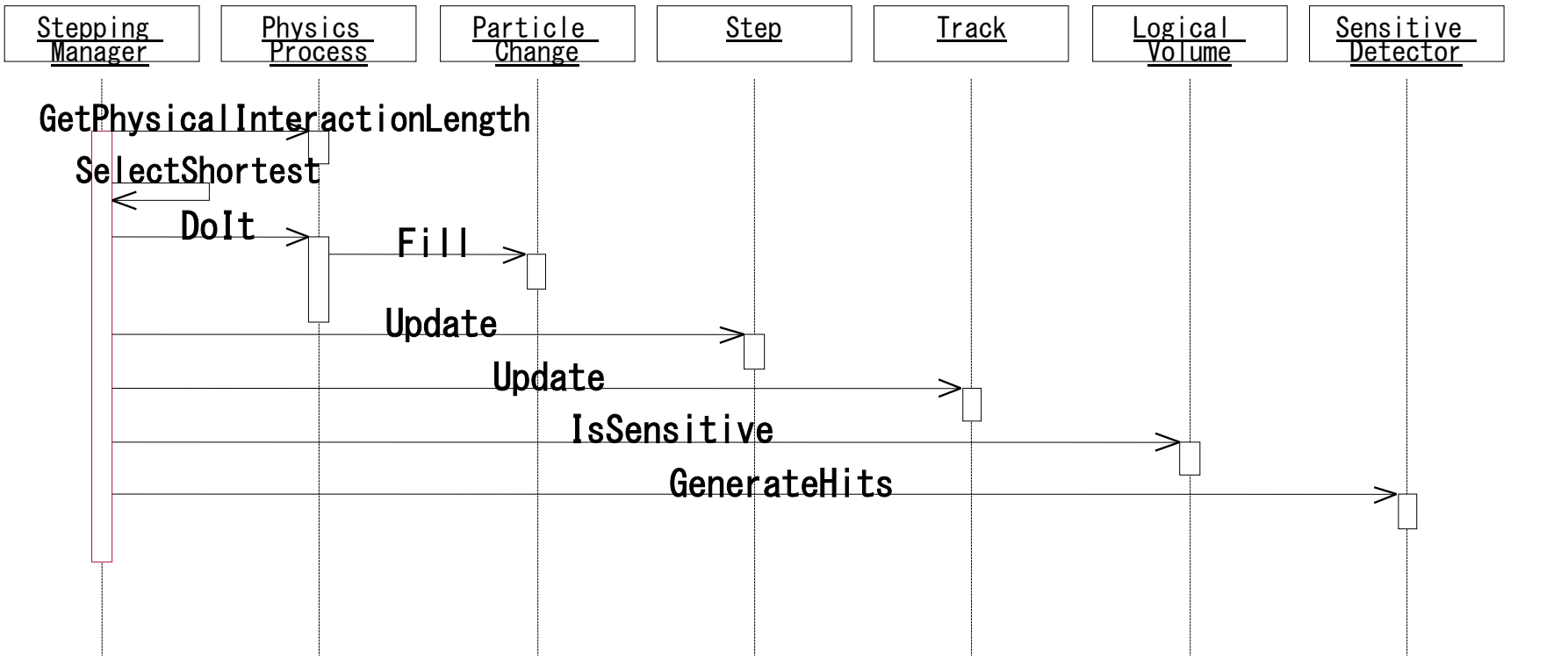
Geant 4

Version 10.0-p01

Basic structure of detector sensitivity

Sensitive detector

- A G4VSensitiveDetector object can be assigned to 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.
  - You can implement your own sensitive detector classes, or use scorer classes provided by Geant4.



Defining a sensitive detector

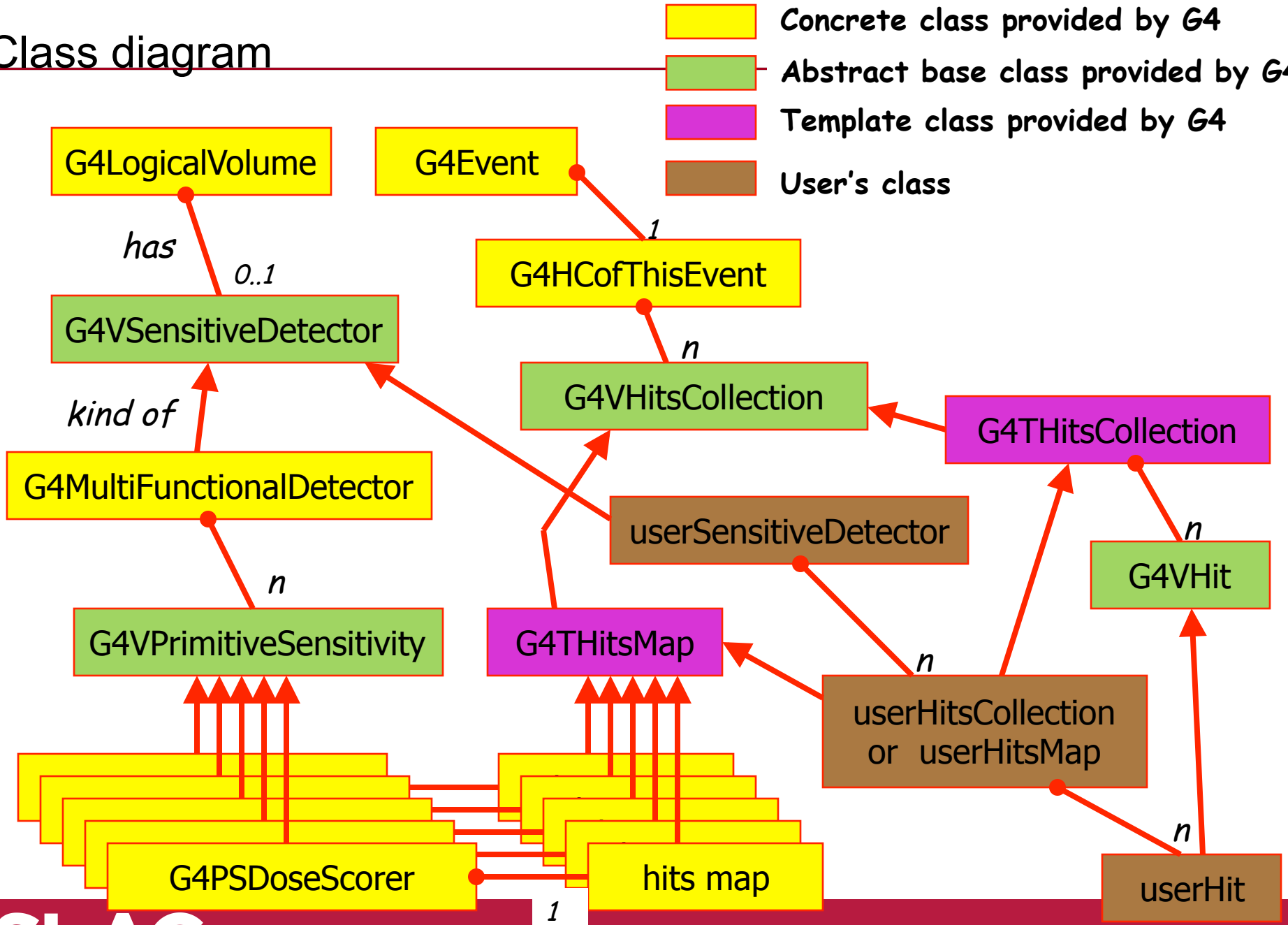
- Basic strategy

  In your ConstructSDandField() method

```
G4VSensetiveDetector* pSensetivePart
  = new MyDetector("/mydet");
SetSensitiveDetector("myLogicalVolume",pSensetivePart);
```

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

Class diagram



Hits collection, hits 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).
  - In other words, G4THitsCollection requires you to implement your hit class.
- G4THitsMap is a template map class so that it 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 hit class.
    - All of currently provided scorer classes use G4THitsMap with simple double.
  - Since G4THitsMap is a template, it can be used by your sensitive detector class to store hits.

Geant 4

Version 10.0-p01

Sensitive detector and hit

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 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

SLAC

- Hit is a user-defined class derived from G4VHit.
- You can store various types information by implementing your 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 will be associated to a G4Event object via G4HCofThisEvent.
- Hits collections are accessible
  - through G4Event at the end of 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"
#include "G4Allocator.hh"
class MyHit : public G4VHit
{
  public:
      MyHit(some_arguments);
      inline void*operator new(size_t);
      inline void operator delete(void *aHit);
      virtual ~MyHit();
      virtual void Draw();
      virtual void Print();
  private:
      // some data members
  public:
      // some set/get methods
};

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

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.
  - Do NOT use G4Allocator for abstract base class.
- G4Allocator must be thread-local. Also, objects instantiated by G4Allocator must be deleted within the same thread.
  - Such objects may be referred by other threads.

Use of G4Allocator

MyHit.hh

```
#include "G4VHit.hh"
#include "G4Allocator.hh"
class MyHit : public G4VHit
{
  public:
      MyHit(some_arguments);
      inline void*operator new(size_t);
      inline void operator delete(void *aHit);
   . . .
};
extern G4ThreadLocal G4Allocator<MyHit>* MyHitAllocator;
inline void* MyHit::operator new(size_t)
{
    if (!MyHitAllocator)
        MyHitAllocator = new G4Allocator<MyHit>;
    return (void*)MyHitAllocator->MallocSingle();
}
inline void MyHit::operator delete(void* aHit)
{ MyHitAllocator->FreeSingle((MyHit*)aHit); }
```

MyHit.cc

```
#include "MyHit.hh"
G4ThreadLocal G4Allocator<MyHit>* MyHitAllocator;
```

Sensitive Detector class

- Sensitive detector is a user-defined class derived from G4VSensitiveDetector.

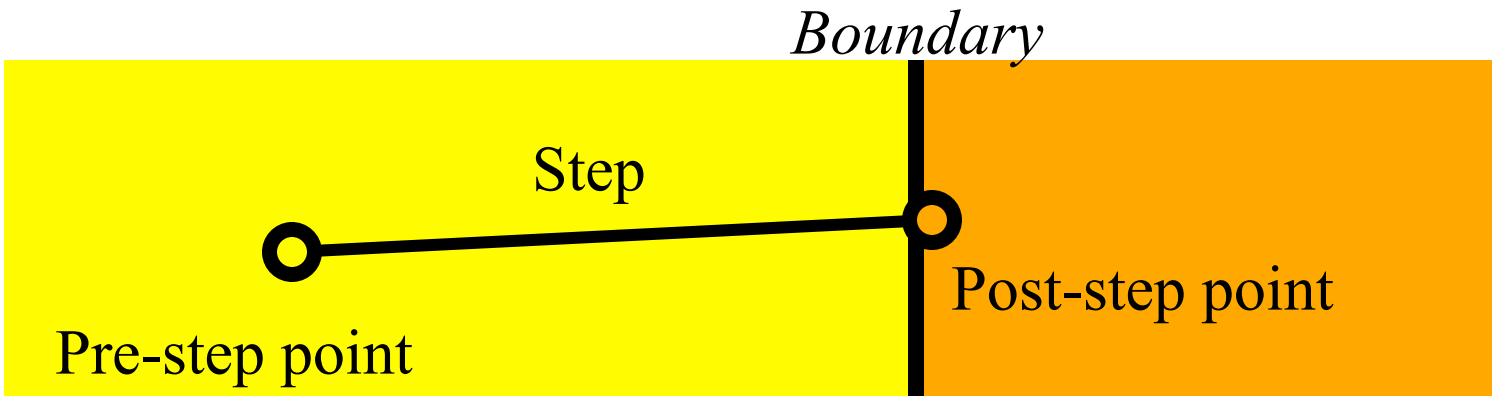
```
#include "G4VSensitiveDetector.hh"
#include "MyHit.hh"
class G4Step;
class G4HCofThisEvent;
class MyDetector : public G4VSensitiveDetector
{
  public:
      MyDetector(G4String name);
      virtual ~MyDetector();
      virtual void Initialize(G4HCofThisEvent*HCE);
      virtual G4bool ProcessHits(G4Step*aStep,
                                 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 ID
- 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
    - Cell ID
- You can instantiate more than one objects for one sensitive detector class. Each object should have its unique detector name.
  - For example, each of two sets of detectors can have their dedicated sensitive detector objects. But, the functionalities of them are exactly the same to each other so that they can share the same class. See examples/basic/B5 as an example.

Step

- 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 you 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, define the name of the hits collection which is handled by this sensitive detector
- In case your sensitive detector generates more than one kinds of hits (e.g. anode and cathode hits separately), define all collection names.

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 events.
  - 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.
- Instantiate hits collection(s) and attach it/them to G4HCofThisEvent object given in the argument.
- In case of calorimeter-type detector, you may also want to instantiate hits for all calorimeter cells with zero energy depositions, and insert them to 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 steps 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 to (for calorimeter detector).
- Don't forget to collect geometry information (e.g. copy number) from "PreStepPoint".
- Currently, returning 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.

Geant 4

Version 10.0-p01

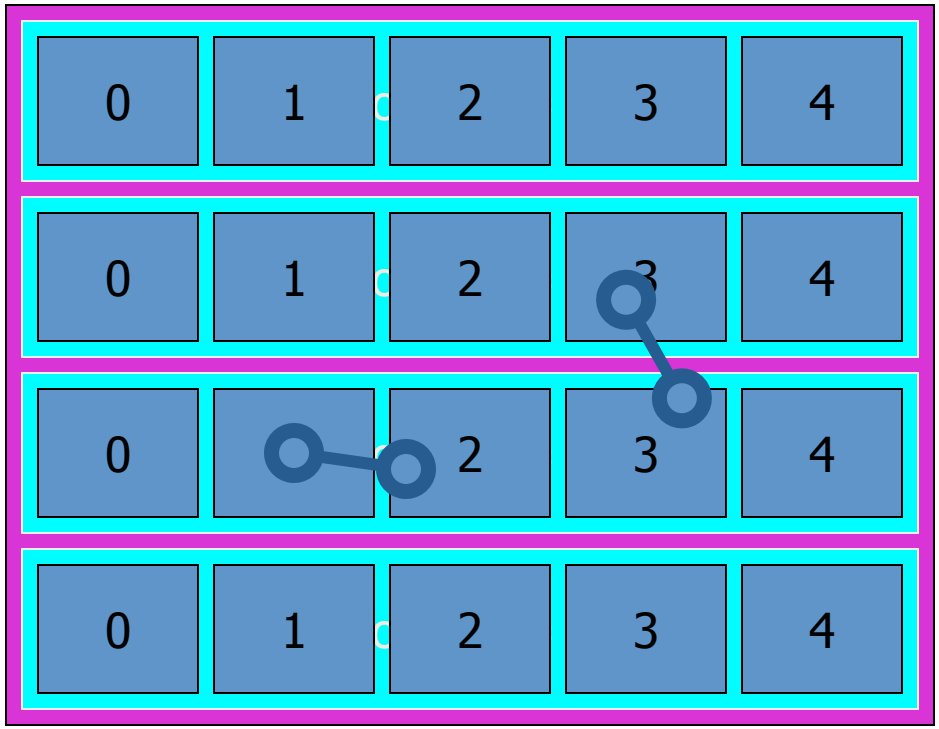
Touchable

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 world coordinate system
  - Global and local time
  - Material
  - G4TouchableHistory for geometrical information
- G4TouchableHistory object is a vector of information for each geometrical hierarchy.
  - copy number
  - transformation / rotation to its mother
- Since release 4.0, *handles* (or *smart-pointers*) to touchables 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, suppose what happens if a step belongs to two cells.



  - Remember geometrical information in G4Track is identical to "PostStepPoint".
  - You cannot get the correct copy number for "PreStepPoint" if you directly access to the physical volume.
- Use touchable to get the proper copy number, transform matrix, etc.

Touchable

- G4TouchableHistory has information of 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);
```

Geant 4

Version 10.0-p01

Use of G4HCofThisEvent class

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.
  - Hits collections are stored by pointers of G4VHitsCollection base class. Thus, you have to cast them to 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.