Sensitive Detector and Hits

山下智弘
JST CREST/神戸大学
Borrowing especially from presentations of M. Asai(SLAC)
Geant4 Tutorial @ Japan 2007
17-19 Oct, 2007 @ RCNS, based on Geant4 9.0.p01

Outline

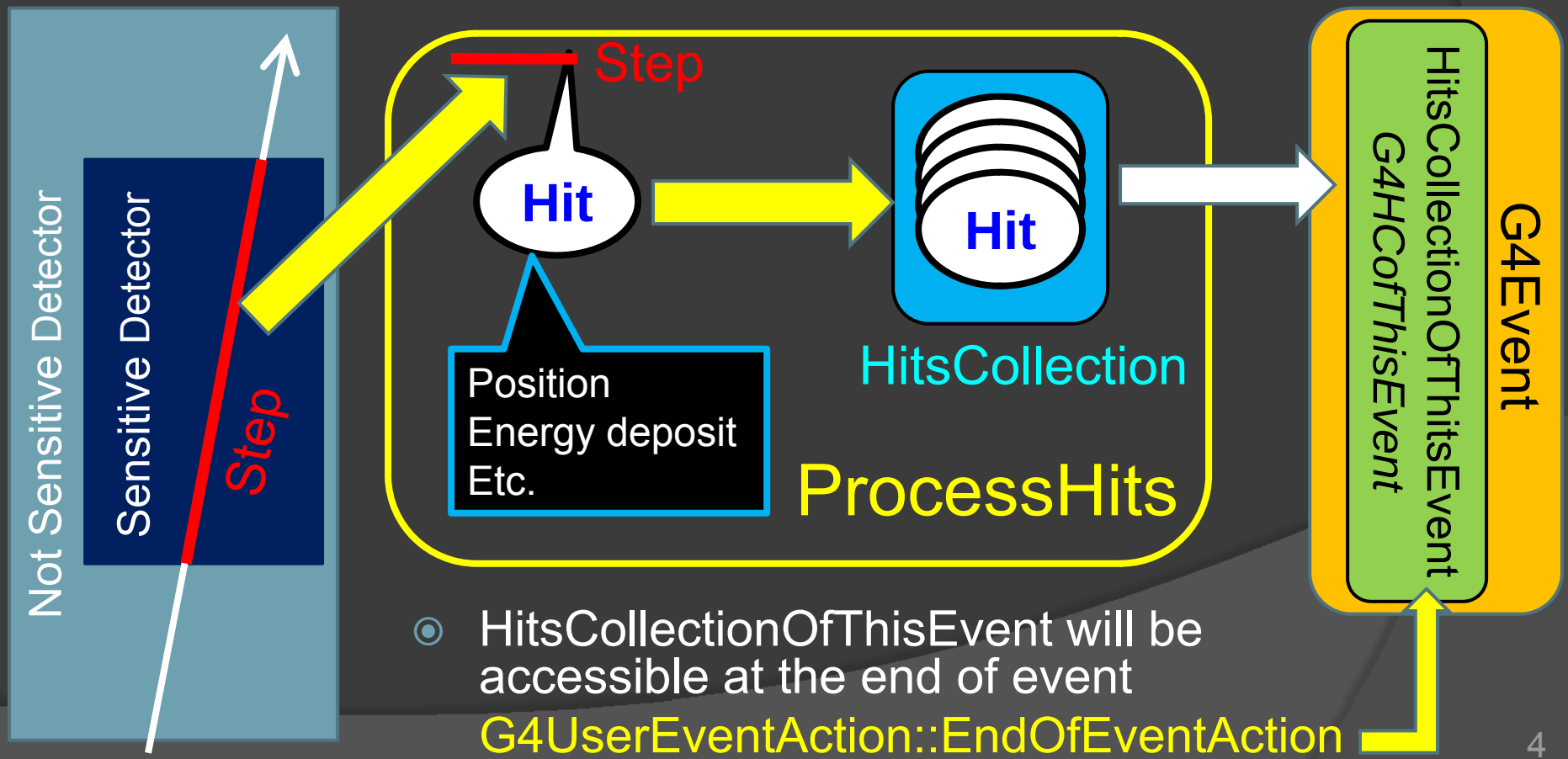
I. Introduction
II. Sensitive detector
III. Hits and hit collection
IV. How to describe detector sensitivity
V. Touchable

I. Introduction

- Given geometry, physics and primary track generation, Geantt4 does proper physics simulation "silently".
  - Need a bit of code to extract useful information
- There are two ways:
  - Use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
    - You have full access to almost all information
    - Straight-forward, but DIY(do-it-yourself)
  - Use Geant4 scoring functionality
    - Assign G4VSensitiveDetector to interested logical volume

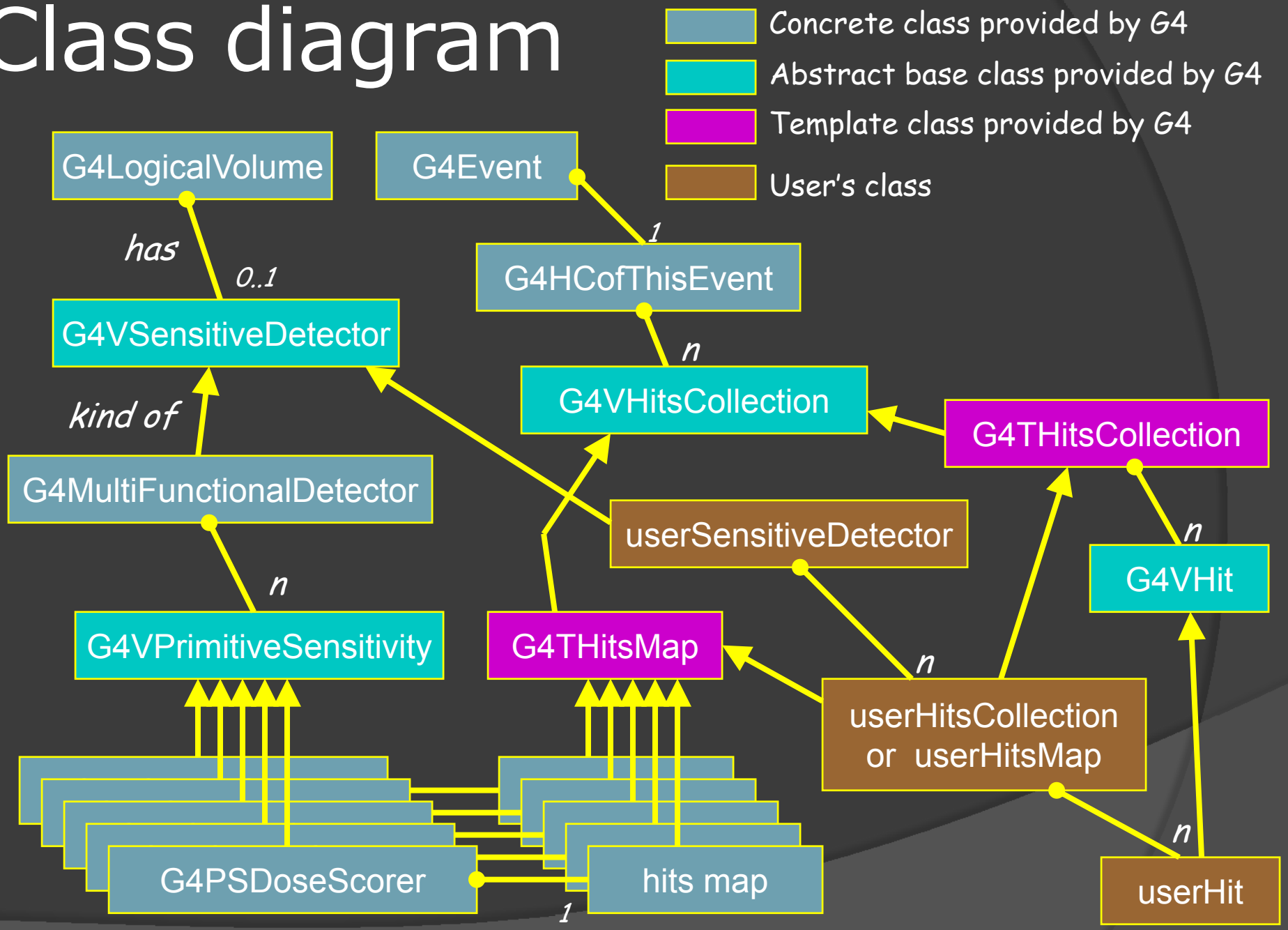
II Sensitive detector Introduction

- In ProcessHits method, sensitive detector creates hits from step and store them into HitsCollection
- HitsCollection will be stored in G4HCofThisEvent of G4Event



- HitsCollectionOfThisEvent will be accessible at the end of event G4UserEventAction::EndOfEventAction

Class diagram



II. Sensitive detector
1. Sensitive detector and Hit

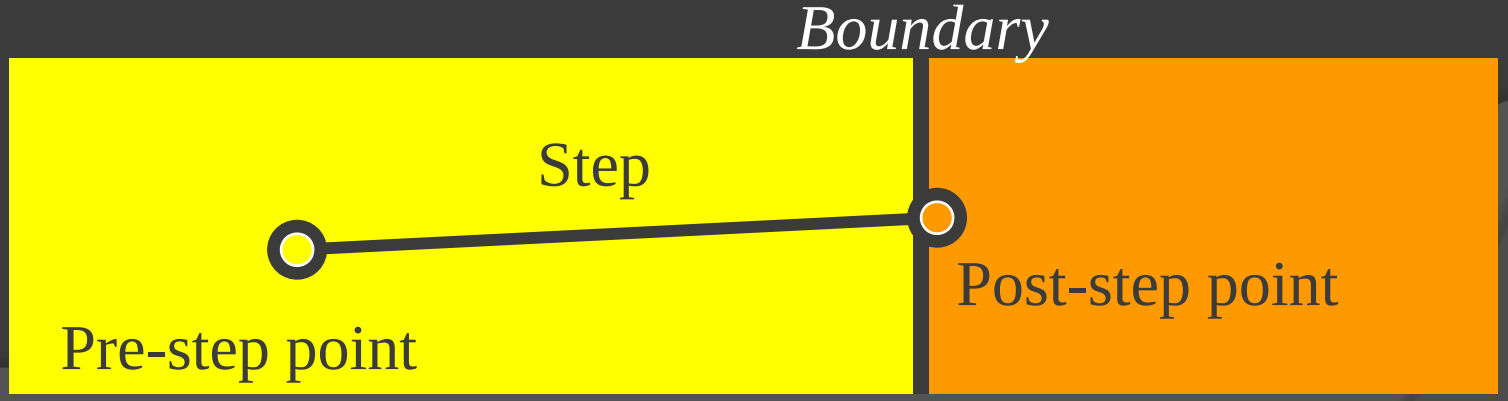
- Each LOGICAL VOLUME can have a pointer to a sensitive detector.
  - Then this volume becomes SENSITIVE.
- HIT is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive region of your detector.
  - Position, energy deposit, etc.
- A sensitive 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.

II. Sensitive detector
2. What can be 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 IID

II. Sensitive detector
3. 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”.



II. Sensitive detector
4. G4SDManager

- All Sensitive Detectors must be REGISTERED with the SDM in order to function properly (tip: do it in the SD's constructor):

```
G4SDManager *sdman=G4SDManager::GetSDMpointer()
sdman->AddSensitiveDetector (this);
```

- A Sensitive Detector Manager (of type G4SDManager) oversees to all operations by Sensitive Detectors.
- The SDM is a singleton (only one object at any moment) and can be accessed by using its static method
  - G4SDManager::GetSDMpointer()
- The SDM can return a hit collection ID (useful for fishing out your collection from the hits collections of the event

```
sdman->GetCollectionID("My Collection");
```

II. Sensitive detector
5. Digitizer module and digit

- Digit represents a detector output (e.g. ADC/TDC count, trigger signal, etc.).
- Digit is created with one or more hits and/or other digits by a user's concrete implementation derived from G4VDigitizerModule.
- In contradiction to the sensitive detector which is accessed at tracking time automatically, the digitize() method of each G4VDigitizerModule must be explicitly invoked by the user s code (e.g. at user's EventAction).

II. Sensitive detector
6. Defining a sensitive detector

- 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 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.

III. Hits and hit collection
1. Hit class

- 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
  - Geometrical information
  - Etc. 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.

III. Hits and hit collection
2. 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.

IV. How to describe detector sensitivity
1. Implementation of Hit(1) MyHit.hh

```
#include "G4VHit.hh"
#include “G4THitsCollection.hh”
#include "G4Allocator.hh"
class MyHit : public G4VHit {
  public:
      MyHit(some_arguments);
      virtual ~MyHit();
      virtual void Draw();
private:
      // some data members
  public:
      // some set/get methods
};

typedef G4THitsCollection<MyHit> MyHitsCollection;
```

IV. How to describe detector sensitivity
1. Implementation of Hit(2) MyHit.hh

```
extern G4Allocator<MyHit> MyHitAllocator;

inline void* MyHit::operator new(size_t)
{
    void *aHit;
    aHit = (void *) MyHitAllocator.MallocSingle();
    return aHit;
}

inline void MyHit::operator delete(void *aHit)
{
    MyHitAllocator.FreeSingle((ExN04CalorimeterHit*) aHit);
}
```

IV. How to describe detector sensitivity
1. Implementation of Hit(3) MyHit.hh

```
private:
      G4double edep;
      G4ThreeVector pos;

public:
      inline void SetEdep(G4double de)
      { edep = de; }
      inline G4double GetEdep()
      { return edep; }
      inline void SetPos(G4ThreeVector xyz)
      { pos = xyz; }
      inline G4ThreeVector GetPos()
      { return pos; }
```

IV. How to describe detector sensitivity
1. Implementation of Hit(4) MyHit.cc

```
G4Allocator<MyHit> MyHitAllocator;

MyHit::MyHit() {}
MyHit::MyHit() {}

ExN02TrackerHit::ExN02TrackerHit(const ExN02TrackerHit& right)
  : G4VHit() {
    edep      = right.edep;
    pos       = right.pos;
}

const MyHit& MyHit::operator=(const MyHit& right) {
    edep      = right.edep;
    pos       = right.pos;
    return *this;
}

G4int MyHit::operator==(const MyHit& right) const {
    return (this == &right) ? 1 : 0;
}
```

IV. How to describe detector sensitivity
2. Implementation of Sensitive Detector(1)

- 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;
};
```

IV. How to describe detector sensitivity
2. Implementation of Sensitive Detector(2)

```
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.

IV. How to describe detector sensitivity
2. Implementation of Sensitive Detector(3)

```
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.

IV. How to describe detector sensitivity
2. Implementation of Sensitive Detector(4)

```
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".

IV. How to describe detector sensitivity
2. Implementation of Sensitive Detector(5)

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

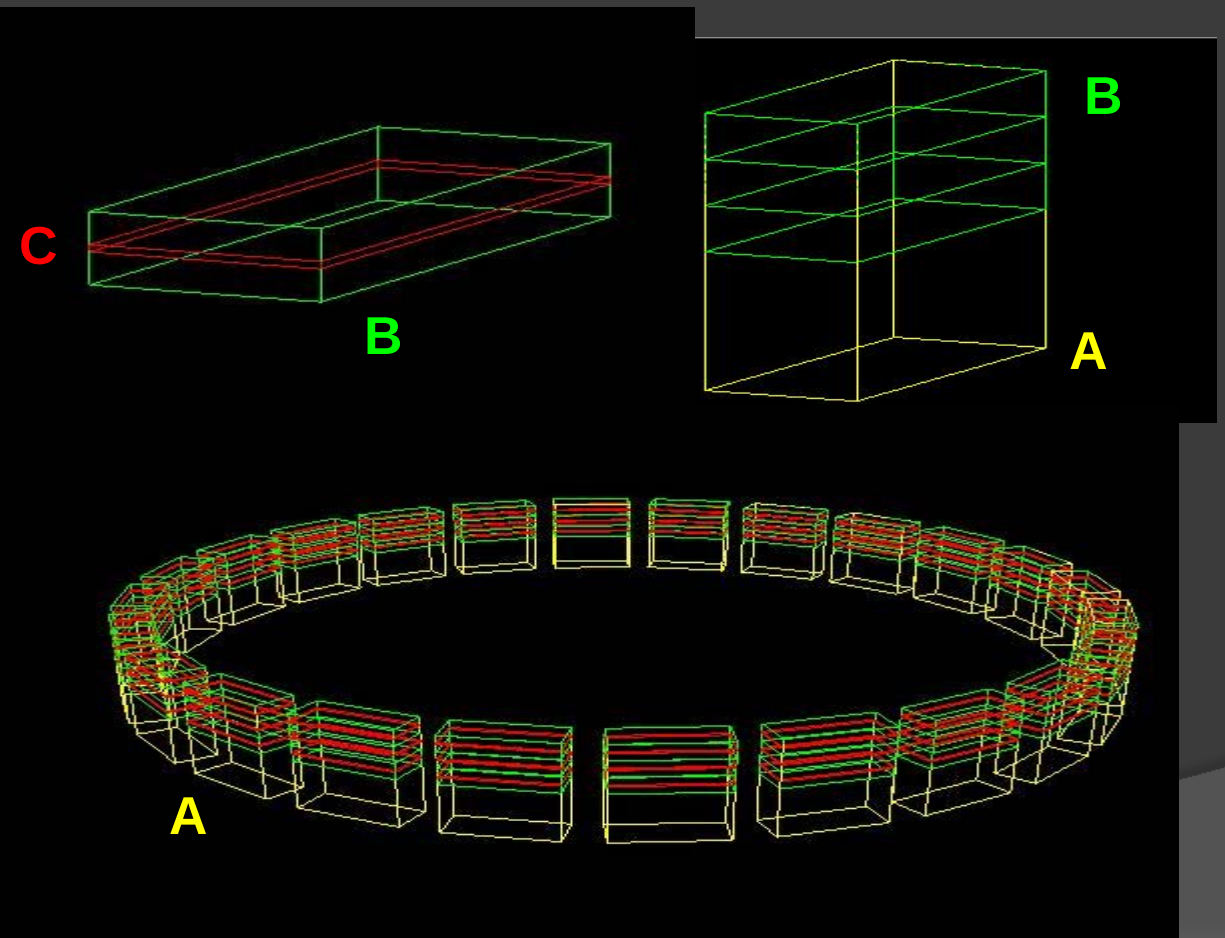
- This method is invoked at the end of processing an event.
- It is invoked even if the event is aborted.
- It is invoked before UserEndOfEventAction.

IV. How to describe detector sensitivity
3. Usage of G4HCofThisEvent

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

V. Touchable
1. Copy number of replicated volume

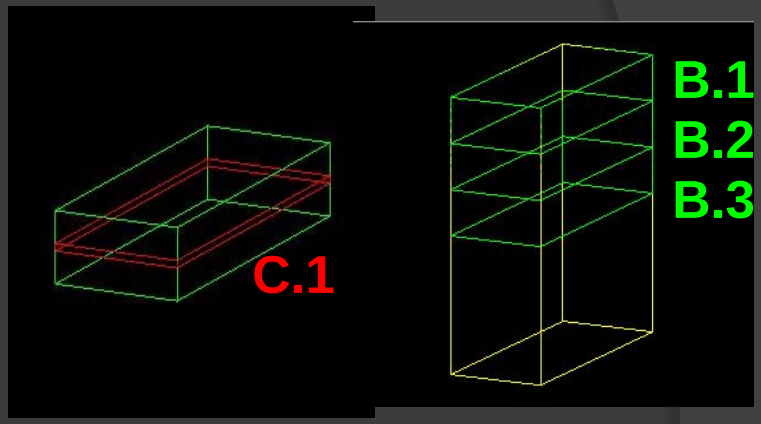
- Suppose a geometry is made of sensitive layers C which are placed in a volume B
- Volume B is a daughter volume of a divided volume A

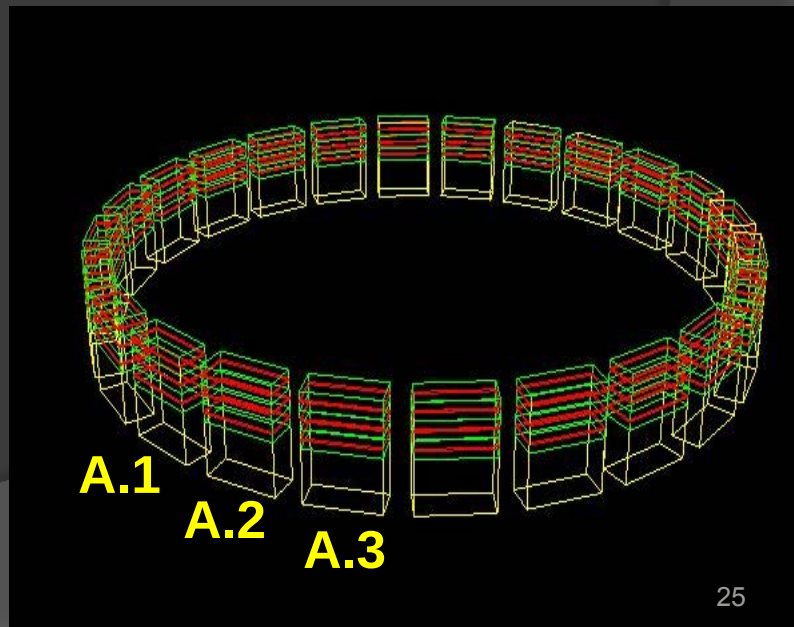


- The volume A has a 24 positions in the world
- While in the 'logical' geometry tree the volume C is represented by just one physical volume, in the real world there are many C 'volumes'
- How can we then identify these volumes C ?
- TOUCHABLE!

V. Touchable
2. Touchable

- A touchable for a volume serves the purpose of providing a unique identification for a detector element
- It is a geometrical entity (volume or solid) which has a unique placement in a detector description
  - It can be uniquely identified by providing the copy numbers for all daughters in the geometry hierarchy
  - In our case these are
    - CopyNo of C in B: 1
    - CopyNo of B in A: 1,2,3
    - CopyNo of A in the world: 1, .., 24
  - Example of touchable identification:
    - A.3/B.2/C.1





V. Touchable
2. TouchableHistory

G4bool ExN04CalorimeterSD::ProcessHits(G4Step*aStep,
G4TouchableHistory*ROhist){

- G4TouchableHistory
  - Representing a touchable detector element, and its history in the geomtrical hierarchy, including its net resultant local->global transform.
  - depth means always the number of levels up in the tree to be considered:
    depth = 0 : the bottom level (volume C in B)
    depth = 1 : the level of its mother volume (volume B in A)
    depth = 2 : the grandmother volume (volume A in world)
- G4int GetCopyNumber(G4int depth =0)
  - returns the copy number of the given level of current volume
- G4ThreeVector& GetTranslation(G4int depth = 0)
  - return the components of the volume's transformation
- G4RotationMatrix* GetRotation(G4int depth=0)
  - returns the rotation matrix
- GetVolume(G4int depth =0)
  - returns the physical volume

V. Touchable
3. Touchable usage

- Full geometrical information available via touchable
  - to processes, to user code, sensitive detectors, hits

```
G4bool CalorimeterSD::ProcessHits(G4Step*aStep, G4TouchableHistory*) {
  G4double edep = aStep->GetTotalEnergyDeposit();

  if(edep==0.) return false;

  const G4TouchableHandle touchable
        = aStep->GetPreStepPoint()->GetTouchableHandle();
  G4VPhysicalVolume* physVol = ROhist->GetVolume();
  G4int copyIDinZ = touchable->GetCopyNumber();
  G4int copyIDinPhi = touchable ->GetCopyNumber(1);
  if(CellID[copyIDinZ][copyIDinPhi]==-1) {
      CalorimeterHit* calHit = new CalorimeterHit (…)
…
```

Backup slides start here

IV. How to describe detector sensitivity
4. 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 construction of sensitive detectors (and thus 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.

II. Sensitive detector Class diagram