Scoring I

Makoto Asai (SLAC)

Geant4 Tutorial Course

Geant4

Contents

- Retrieving information from Geant4
- Basic structure of detector sensitivity
- Sensitive detector vs. primitive scorer
- Primitive scorers
- Filter class
- Accumulating scores for a run

Retrieving information from Geant4

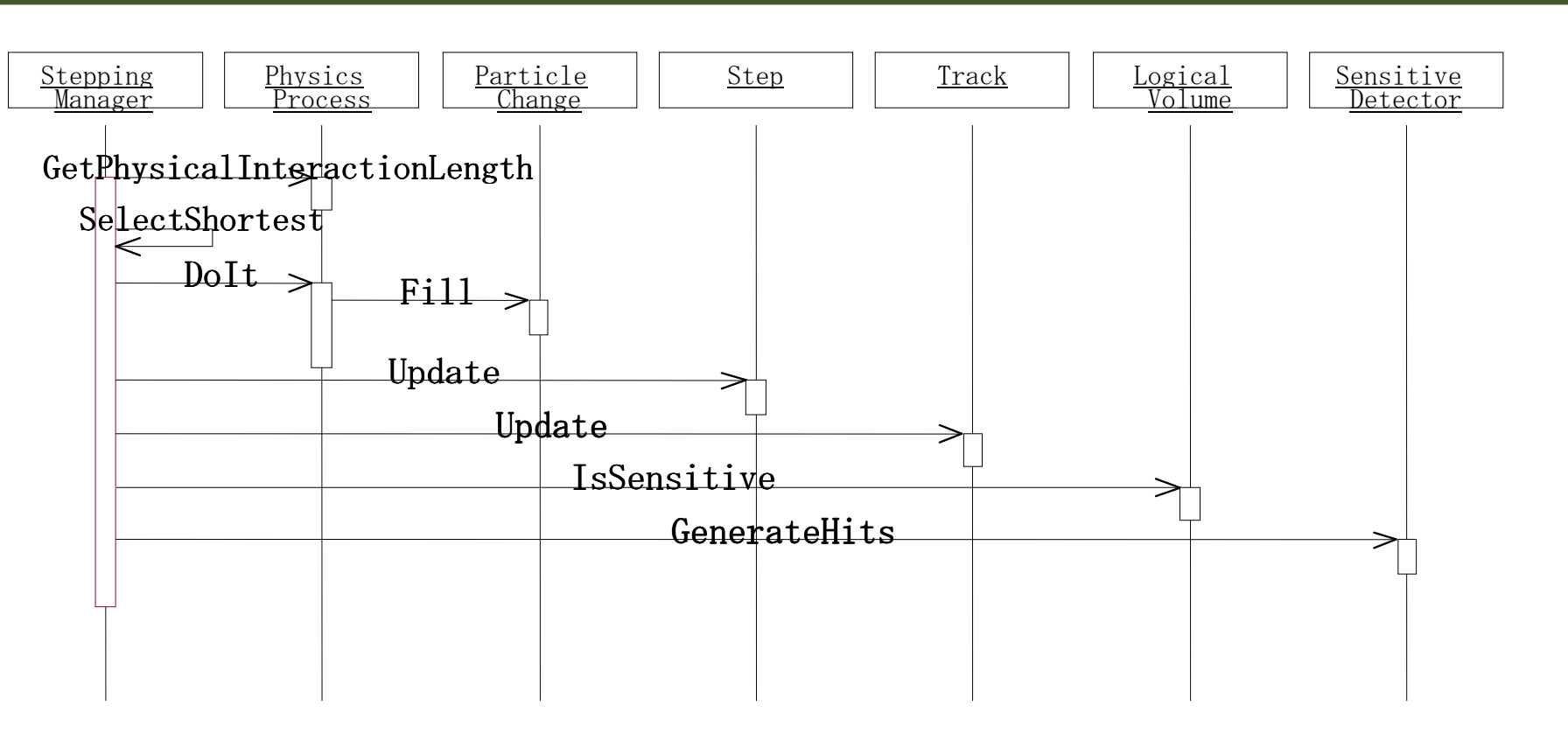
Extract useful information

- 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 two ways:
  - Use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
    - You have full access to almost all information
    - Straight-forward, but do-it-yourself
  - Use Geant4 scoring functionality
    - Assign G4VSensitiveDetector to a volume
    - Hit is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive (or interested) part of your detector.
    - Hits collection is automatically stored in G4Event object, and automatically accumulated if user-defined Run object is used.
    - Use user hooks (G4UserEventAction, G4UserRunAction) to get event / run summary

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.



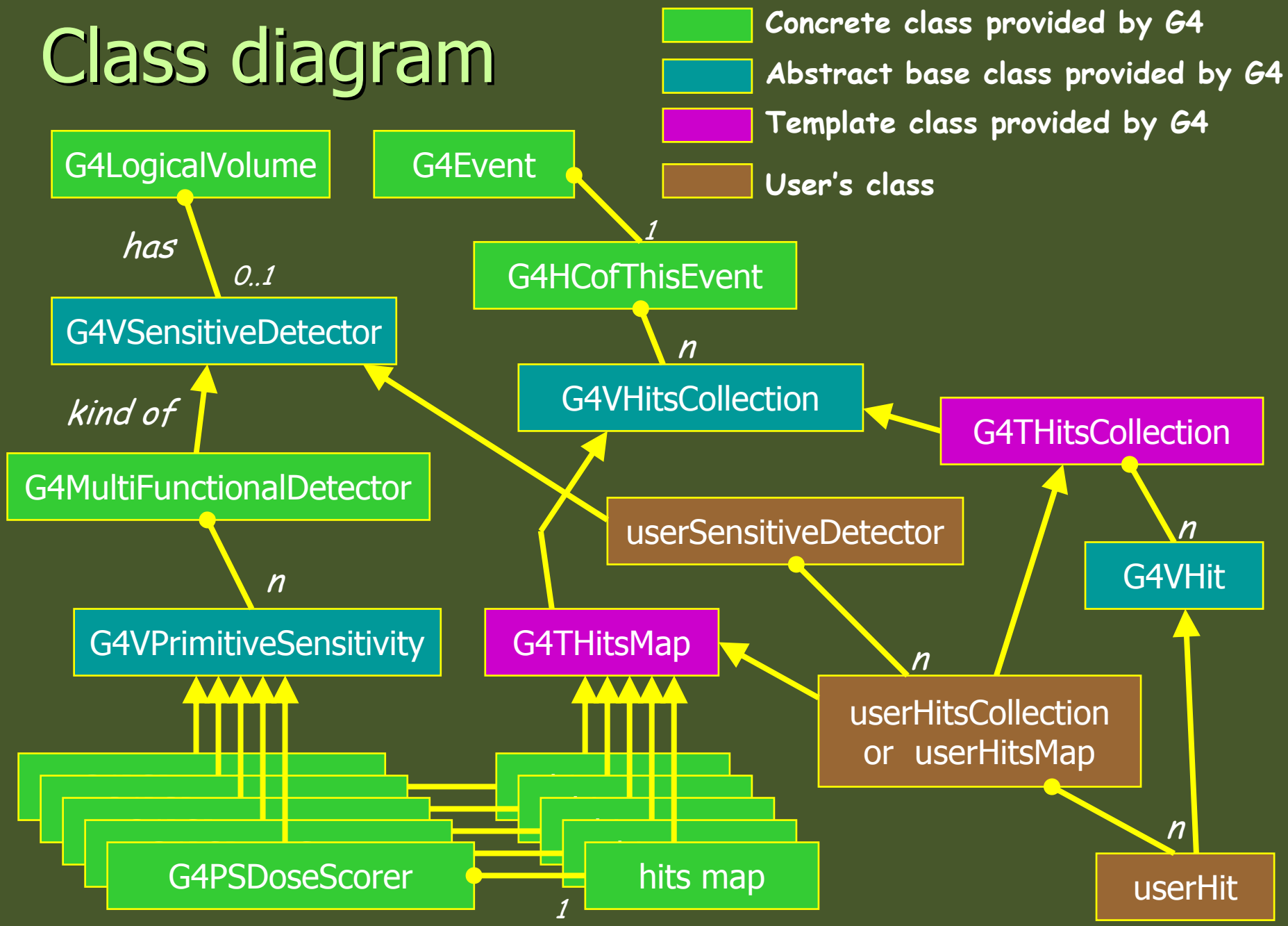
Defining a sensitive detector

- Basic strategy

```
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.

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.

Sensitive detector
vs.
primitive scorer

G4MultiFunctionalDetector

- G4MultiFunctionalDetector is a concrete class derived from G4VSensitiveDetector. It should be set to a logical volume as a kind of sensitive detector.
- It takes arbitrary number of G4VPrimitiveSensitivity classes. By registering G4VPrimitiveSensitivity classes, you can define the scoring detector of your need.
  - Each G4VPrimitiveSensitivity class accumulates one physics quantity for each physical volume.
  - For example, G4PSDoseScorer (a concrete class of G4VPrimitiveSensitivity provided by Geant4) accumulates dose for each cell.
- By using G4MultiFunctionalDetector and provided concrete G4VPrimitiveSensitivity classes, you are freed from implementing sensitive detector and hit classes.

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.

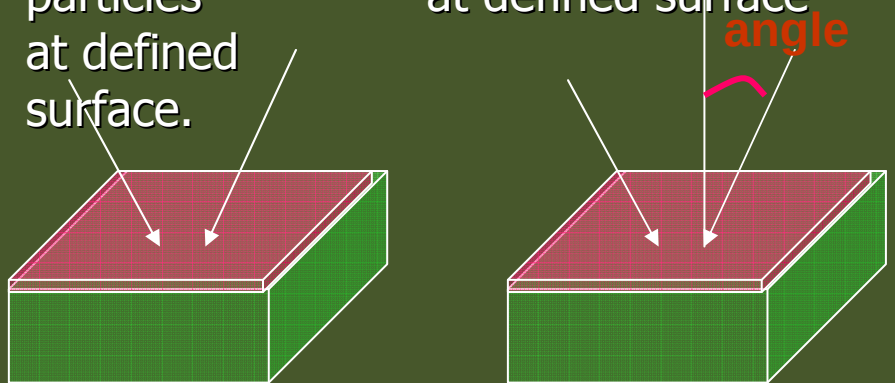
Primitive scorers

List of provided primitive scorers

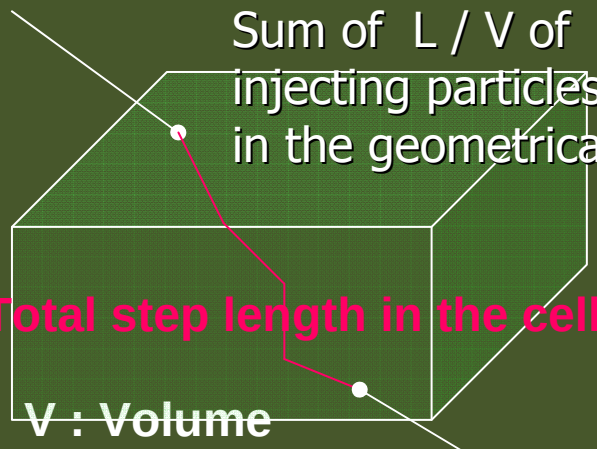
- Concrete Primitive Scorers (See Application Developers Guide 4.4.6)
  - Track length
    - G4PSTrackLength, G4PSPassageTrackLength
  - Deposited energy
    - G4PSEnergyDepsit, G4PSDoseDeposit, G4PSChargeDeposit
  - Current/Flux
    - G4PSFlatSurfaceCurrent, G4PSSphereSurfaceCurrent,G4PSPassageCurrent, G4PSFlatSurfaceFlux, G4PSCellFlux, G4PSPassageCellFlux
  - Others
    - G4PSMinKinEAtGeneration, G4PSNofSecondary, G4PSNofStep

SurfaceCurrent : Count number of injecting particles at defined surface.

SurfaceFlux : Sum up 1/cos(angle) of injecting particles at defined surface



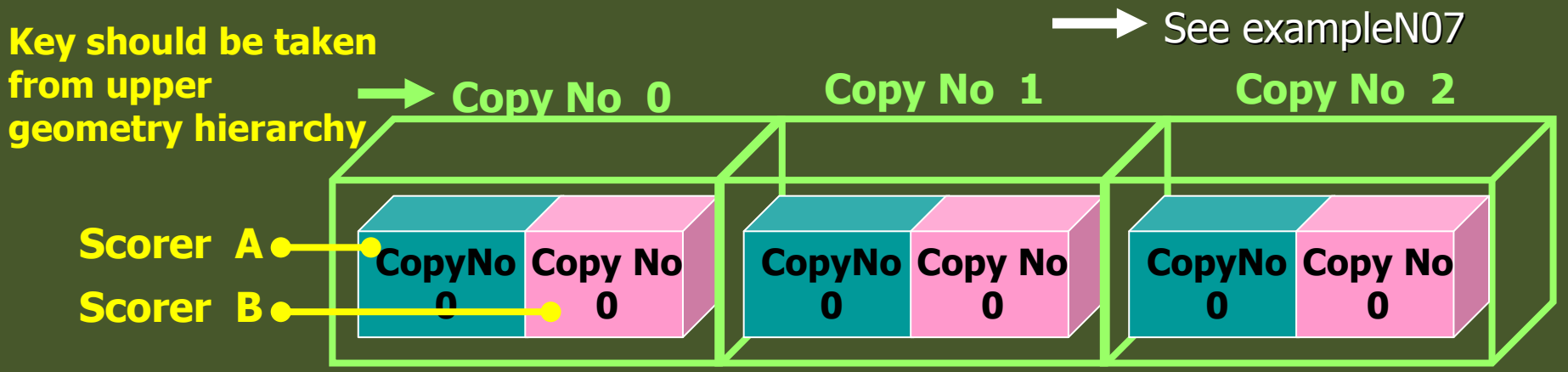
CellFlux : Sum of L / V of injecting particles in the geometrical cell.



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 second argument of the constructor of the primitive scorer to indicate the level where the copy number should be taken.

    e.g. G4PSCellFlux(G4Steing name, G4int depth=0)



  - 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.

For example...

```
MyDetectorConstruction::Construct()
{ …  G4LogicalVolume* myCellLog =  new G4LogicalVolume(…);
     G4VPhysicalVolume* myCellPhys = new G4PVParametrised(…);
     G4MultiFunctionalDetector* myScorer = new
        G4MultiFunctionalDetector("myCellScorer");
     G4SDManager::GetSDMpointer()->AddNewDetector(myScorer);
     myCellLog->SetSensitiveDetector(myScorer);
     G4VPrimitiveSensitivity* totalSurfFlux = new
        G4PSFlatSurfaceFlux("TotalSurfFlux");
     myScorer->Register(totalSurfFlux);
     G4VPrimitiveSensitivity* totalDose = new G4PSDoseDeposit("TotalDose");
     myScorer->Register(totalDose);
}
```

No need of implementing sensitive detector !

Creating your own scorer

- Though we provide most commonly-used scorers, you may want to create your own.
  - If you believe your requirement is quite common, just let us know, so that we will add a new scorer.
- G4VPrimitiveScorer is the abstract base class.

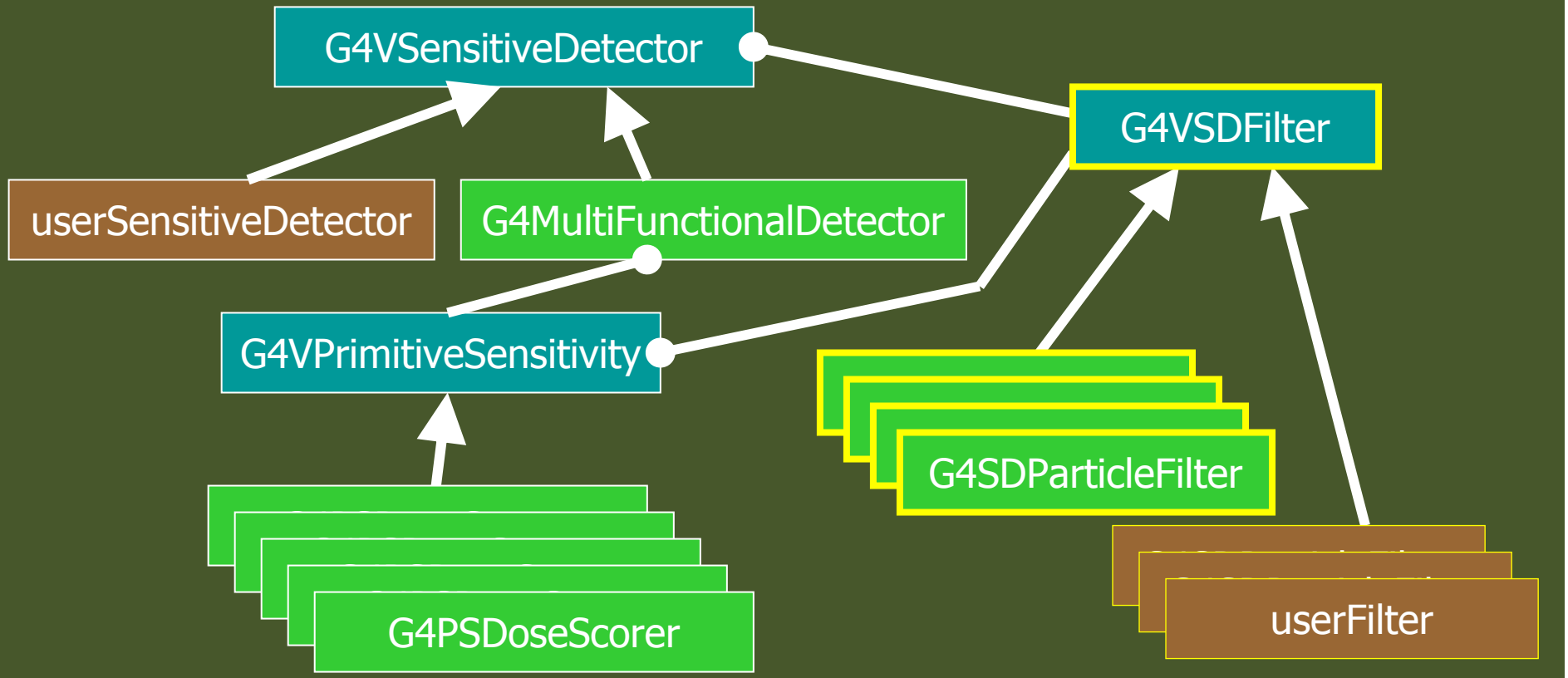
```
class G4VPrimitiveScorer
{
 public:
   G4VPrimitiveScorer(G4String name, G4int depth=0);
   virtual ~G4VPrimitiveScorer();
 protected:
   virtual G4bool ProcessHits(G4Step*,
                              G4TouchableHistory*) = 0;
   virtual G4int GetIndex(G4Step*);
 public:
   virtual void Initialize(G4HCofThisEvent*);
   virtual void EndOfEvent(G4HCofThisEvent*);
   virtual void clear();
      …
};
```

- GetIndex() has already been introduced. Other four methods written in yellow will be discussed at "Scoring 2" talk.

Filter class

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.



List of provided filter classes

- G4SDChargedFilter, G4SDNeutralFilter
  - Accept only charged/neutral tracks, respectively
- G4SDKineticEnergyFilter
  - Accepts tracks within the defined range of kinetic energy
- G4SDParticleFilter
  - Accepts tracks of registered particle types
- G4SDParticleWithEnergyFilter
  - Accepts tracks of registered particle types within the defined range of kinetic energy
- G4VSDFilter
  - Abstract base class which you can use to make your own filter

```
class G4VSDFilter
{
  public:
      G4VSDFilter(G4String name);
      virtual ~G4VSDFilter();
  public:
      virtual G4bool Accept(const G4Step*) const = 0;
…
```

For example…

```
MyDetectorConstruction::Construct()
{ …  G4LogicalVolume* myCellLog =  new G4LogicalVolume(…);
     G4VPhysicalVolume* myCellPhys = new G4PVParametrised(…);
     G4MultiFunctionalDetector* myScorer = new G4MultiFunctionalDetector("myCellScorer");
     G4SDManager::GetSDMpointer()->AddNewDetector(myScorer);
     myCellLog->SetSensitiveDetector(myScorer);
     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);
}
```

Accumulating scores for a run

A tip for scoring

- For scoring purposes, you need to accumulate a physical quantity (e.g. energy deposition of a step) for entire run of many events. In such a case, do NOT sum up individual energy deposition of each step directly to a variable for entire run.
  - Compared to the total sum for entire run, each energy deposition of single step is too tiny. Rounding error problem may easily happen.
    - Total energy deposition of 1 million events of 1 GeV incident particle ends up to 1 PeV (10^{15} eV), while energy deposition of each single step is O(1 keV) or even smaller.
- Create your own Run class derived from G4Run, and implement RecordEvent(const G4Event*) virtual method. Here you can get all output of the event so that you can accumulate the sum of an event to a variable for entire run.
  - RecordEvent(const G4Event*) is automatically invoked by *G4RunManager*.
  - Your run class object should be instantiated in GenerateRun() method of your *UserRunAction*.

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*);
  private:
    G4int nEvent;
    G4int totalSurfFluxID, protonSurfFluxID, totalDoseID;
    G4THitsMap<G4double> totalSurfFlux;
    G4THitsMap<G4double> protonSurfFlux;
    G4THitsMap<G4double> totalDose;
    G4THitsMap<G4double>* eventTotalSurfFlux;
    G4THitsMap<G4double>* eventProtonSurfFlux;
    G4THitsMap<G4double>* eventTotalDose
  public:
    … access methods …
};
```

Implement how you accumulate event data

Customized run class

```
MyRun::MyRun() : nEvent(0)
{
  G4SDManager* SDM = G4SDManager::GetSDMpointer();
  totalSurfFluxID = SDM->GetCollectionID("myCellScorer/TotalSurfFlux");
  protonSurfFluxID = SDM->GetCollectionID("myCellScorer/ProtonSurfFlux");
  totalDoseID = SDM->GetCollectionID("myCellScorer/TotalDose");
}
void MyRun::RecordEvent(const G4Event* evt)
{
  nEvent++;
  G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
  eventTotalSurfFlux = (G4THitsMap<G4double>*)(HCE->GetHC(totalSurfFluxID));
  eventProtonSurfFlux = (G4THitsMap<G4double>*)(HCE->GetHC(protonSurfFluxID));
  eventTotalDose = (G4THitsMap<G4double>*)(HCE->GetHC(totalDose));
  totalSurfFlux += *eventTotalSurfFlux;
  protonSurfFlux += *eventProtonSurfFlux;
  totalDose += *eventTotalDose;
}
```

name of G4MultiFunctionalDetector object

name of G4VPrimitiveSensitivity object

No need of loops.
+= operator is provided !

RunAction with customized run

```
G4Run* MyRunAction::GenerateRun()
{ return (new MyRun()); }
void MyRunAction::EndOfRunAction(const G4Run* aRun)
{
  MyRun* theRun = (MyRun*)aRun;
  // … analyze / record / print-out your run summary
  // MyRun object has everything you need …
}
```

- 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.
- With newly introducing concrete sensitivity classes, you do NOT even need
  - Sensitive detector implementation

→ Refer to exampleN07

Accessing to a hits map

- G4THitsMap<G4double> is an STD map, mapping a key (G4int) to a pointer to a double value, i.e. equivalent to **std::map<G4int,G4double*>**.

```
G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
G4THitsMap<G4double>* evtMap
   = (G4THitsMap<G4double>*)(HCE->GetHC(colID));
```

- To get number of entries

```
G4int n = evtMap->entries();
```

- To access to each entry sequentially

```
std::map<G4int,G4double*>::iterator itr
   = evtMap->GetMap()->begin();
for( ; itr!=evtMap->GetMap()->end(); itr++ )
{ G4int key = (itr->first);
  G4double val = *(itr->second); }
```

Pointer is returned.

- To access to a double value with a key

```
G4double* pVal = (*evtMap)[key];
If(pVal) val = *pVal;
```

Null pointer is returned if the key does not exist in the map.