SLAC
NATIONAL ACCELERATOR LABORATORY

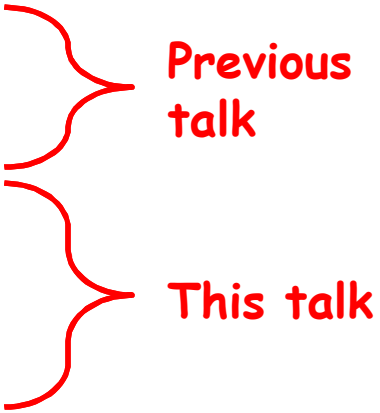
Scoring II

Makoto Asai (SLAC)

Geant4 Tutorial Course

Contents

- Retrieving information from Geant4
- Command-based scoring
- Add a new scorer/filter to command-based scoring
- Define scorers in the tracking volume
- Accumulate scores for a run



SLAC
NATIONAL ACCELERATOR LABORATORY

Add a new scorer/filter to command-based scorers

Geant 4

Scorer base class

- G4VPrimitiveScorer is the abstract base of all scorer classes.
- To make your own scorer you have to implement at least:
  - Constructor
  - Initialize()
    - Initialize G4THitsMap<G4double> map object
  - ProcessHits()
    - Get the physics quantity you want from G4Step, etc. and fill the map
  - Clear()
  - GetIndex()
    - Convert three copy numbers into an index of the map
- G4PSEnergyDeposit3D could be a good example.
- Create your own messenger class to define /score/quantity/<your_quantity> command.
  - Refer to G4ScorerQuantityMessenger class.

Filter class

- 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;
…
```

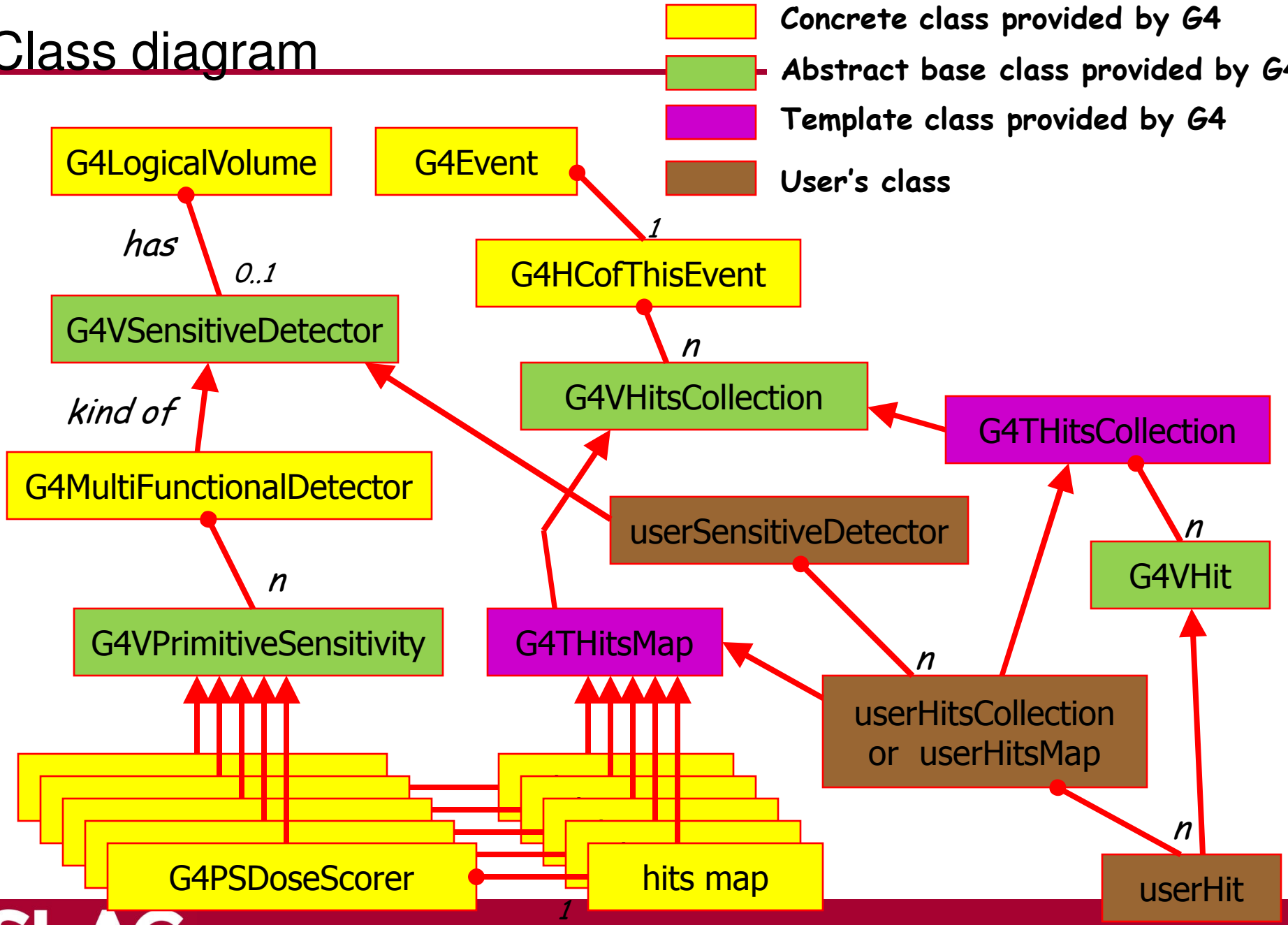
- Create your own messenger class to define /score/filter/<your_filter> command.
  - Refer to G4ScorerQuantityMessenger class.

SLAC
NATIONAL ACCELERATOR LABORATORY

Define scorers to the tracking volume

Geant 4

Class diagram



example

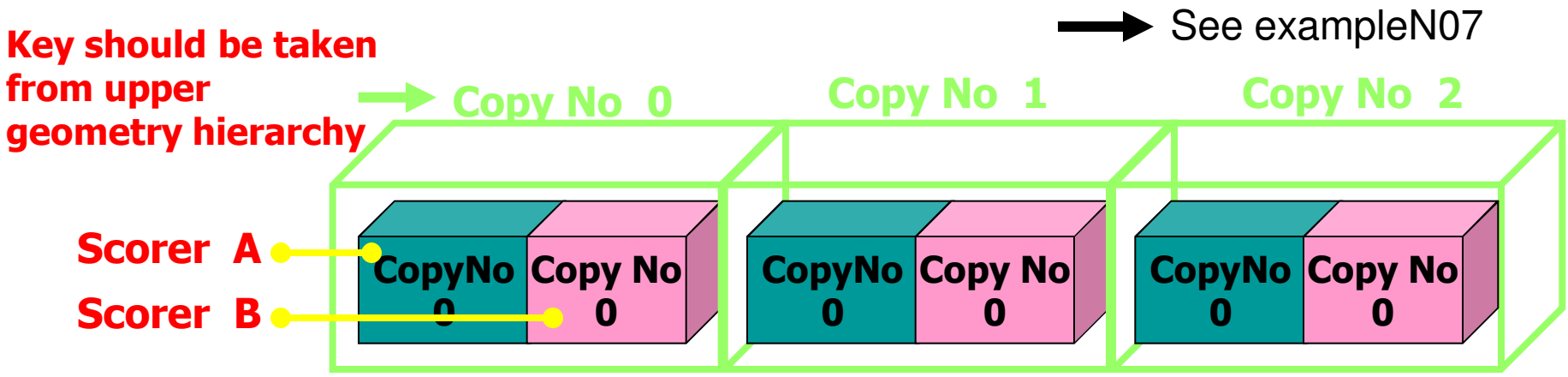
```
MyDetectorConstruction::Construct()
{ …  G4LogicalVolume* myCellLog =  new G4LogicalVolume(…);
     G4VPhysicalVolume* myCellPhys = new G4PVParametrised(…);
     G4MultiFunctionalDetector* myScorer =
        new G4MultiFunctionalDetector("myCellScorer");
     G4SDManager::GetSDMpointer()->AddNewDetector(myScorer);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

You may register arbitrary number of primitive scorers.

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.

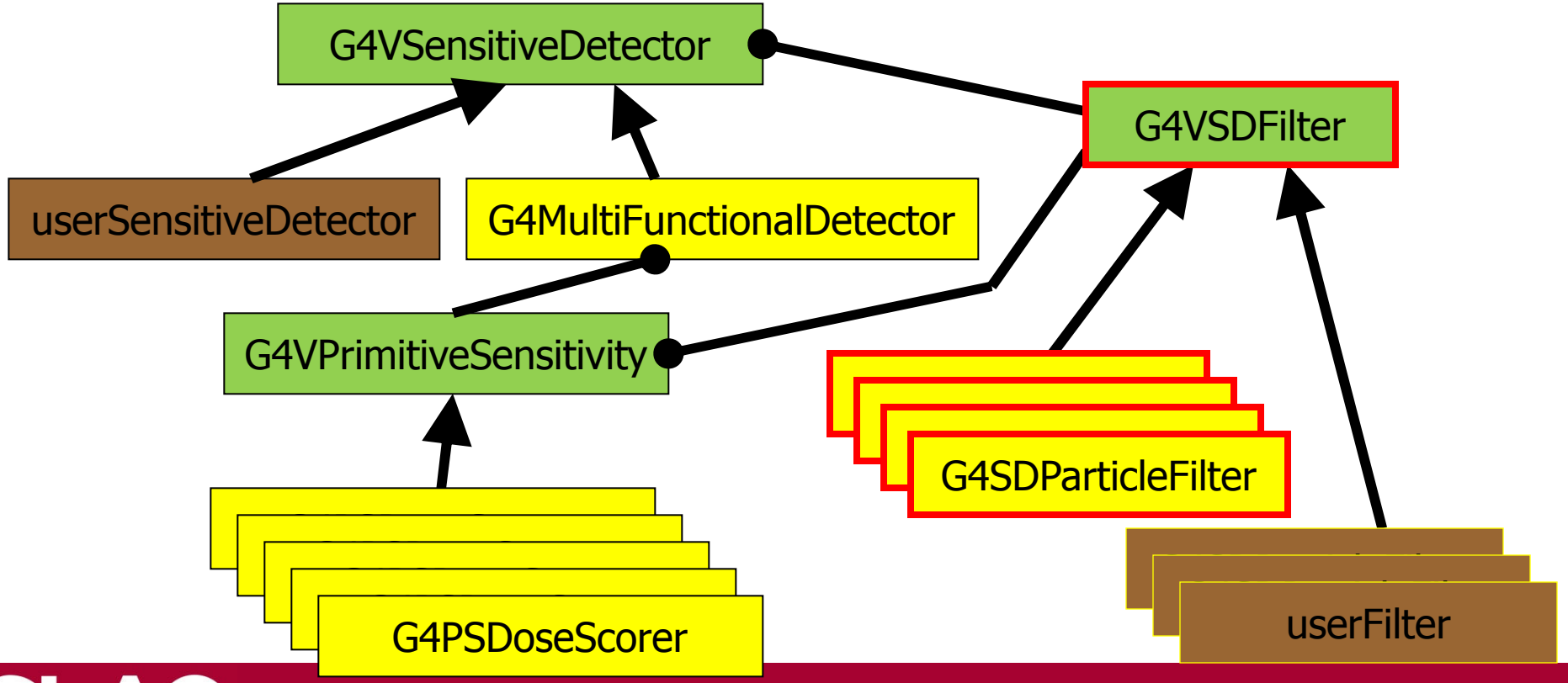
Creating your own scorer

- 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();
      …
};
```

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…

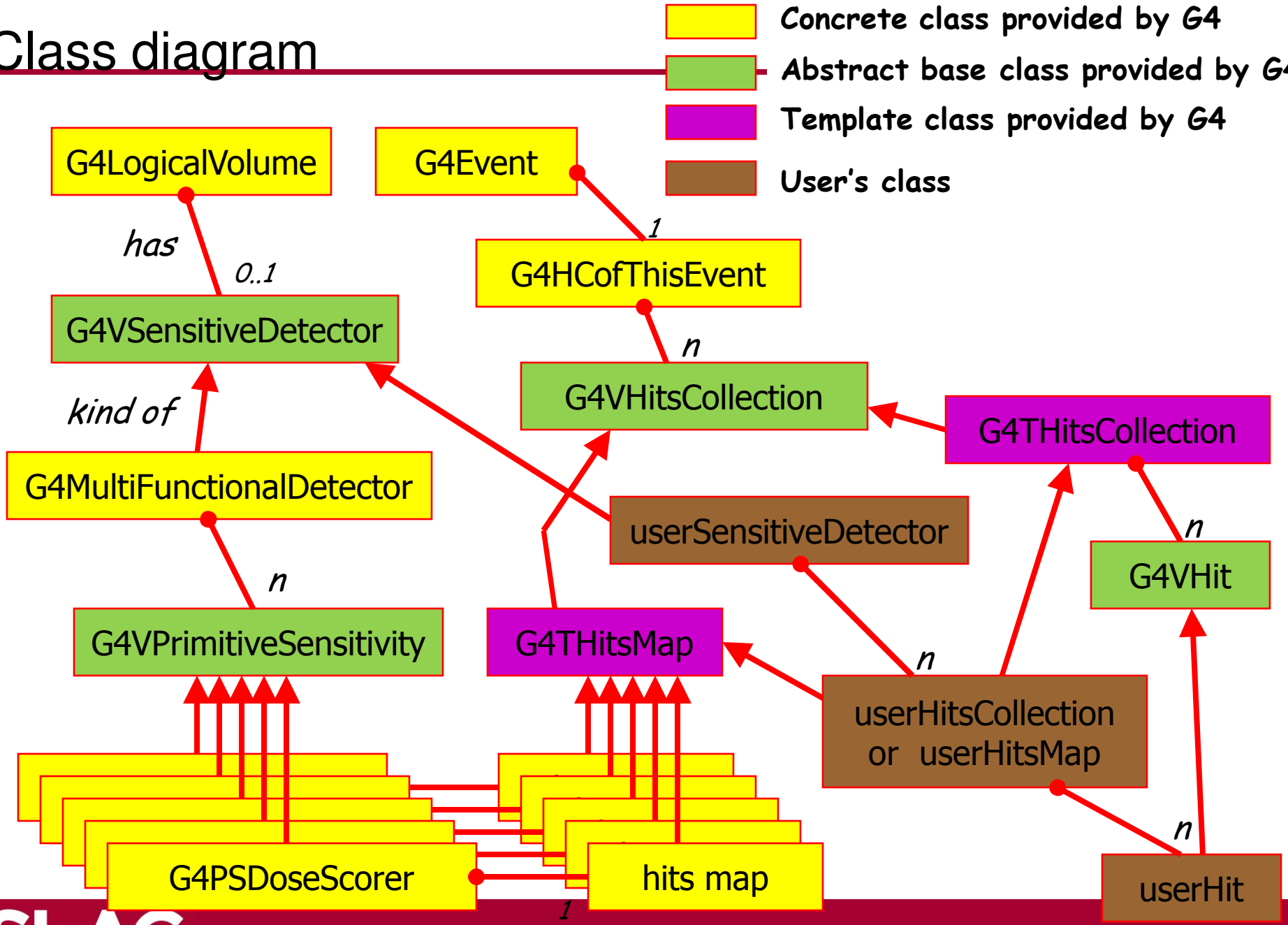
```
MyDetectorConstruction::Construct()
{ …  G4LogicalVolume* myCellLog =  new G4LogicalVolume(…);
      G4VPhysicalVolume* myCellPhys = new G4PVParametrised(…);
      G4MultiFunctionalDetector* myScorer = new G4MultiFunctionalDetector("myCellScorer");
      G4SDManager::GetSDMpointer()->AddNewDetector(myScorer);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

SLAC
NATIONAL ACCELERATOR LABORATORY

Accumulate scores for a run

Geant 4

Class diagram



Score == G4THitsMap<G4double>

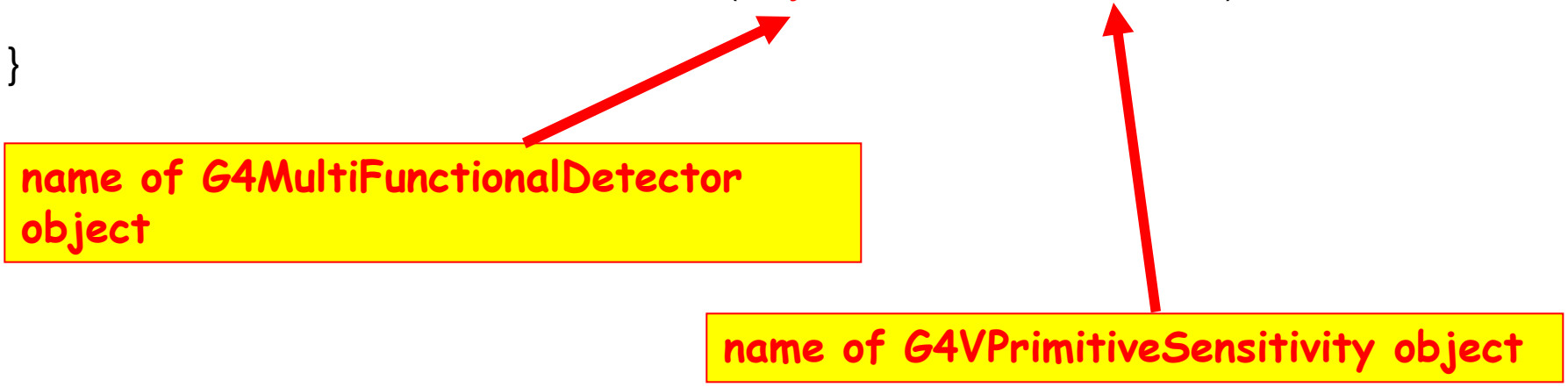
- At the end of successful event, G4Event has a vector of G4THitsMap as the scores.
- Create your own Run class derived from G4Run, and implement RecordEvent(const G4Event*) virtual method. Here you can get all output of the event so that you can accumulate the sum of an event to a variable for entire 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;
public:
    … access methods …
};
```

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");
}
```



Customized run class

```
void MyRun::RecordEvent(const G4Event* evt)
{
  nEvent++;
  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(totalDose));
  totalSurfFlux += *eventTotalSurfFlux;
  protonSurfFlux += *eventProtonSurfFlux;
  totalDose += *eventTotalDose;
}
```

No need of loops.
+= operator is provided !

RunAction with customized run

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

Refer to exampleN07