G4MultiFunctionalDetector, G4VPrimitiveSensitivity, G4VSDFilter and G4THitsMap

Makoto Asai (SLAC)

Geant4 Collaboration Meeting @ Bordeaux

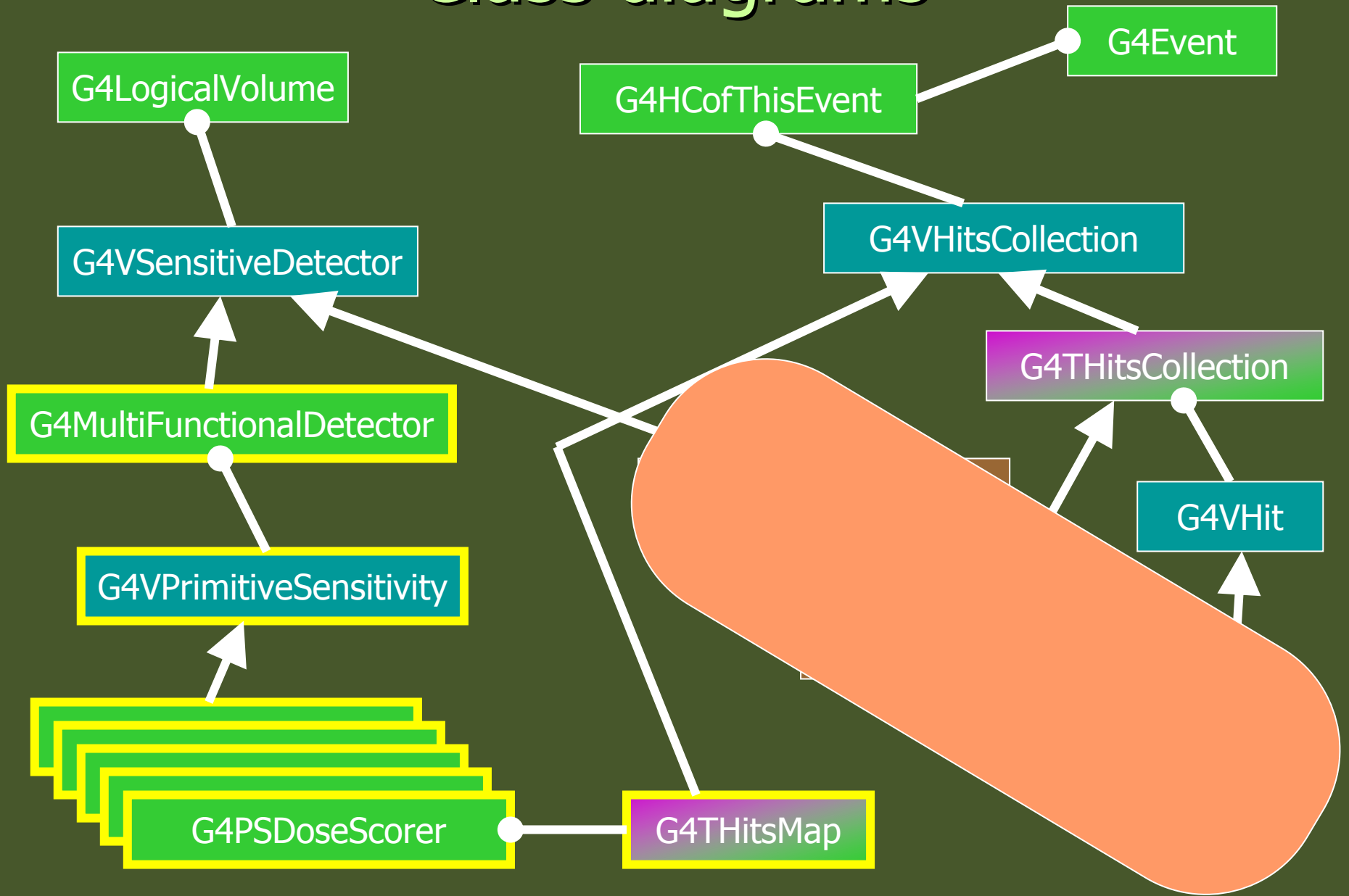
November 2005

Geant4

Concrete sensitivity classes

- Up to the current version (7.1), Geant4 provides only an abstract base class (G4VSensitiveDetector) for the user to define his/her detector sensitivity.
  - Various example detector classes are provided.
    - This is enough for HEP experiments, since their interest is mostly storing hits in their detectors.
  - But not convenient for space and medical applications.
    - Their interest is mainly scoring dose/flux.
- We will introduce G4MultiFunctionalDetector (concrete class derived from G4VSensitiveDetector) that allows the user to register concrete class objects of G4VPrimitiveSensitivity to build a scoring detector of his/her needs.
  - G4PSEnergyDeposit, G4PSFlatSurfaceFlux, G4PSDoseDeposit, G4PSTrackLength, etc. (class names are still preliminary) will be provided.
  - We will continue working for additional primitive sensitivity concrete classes.

Class diagrams

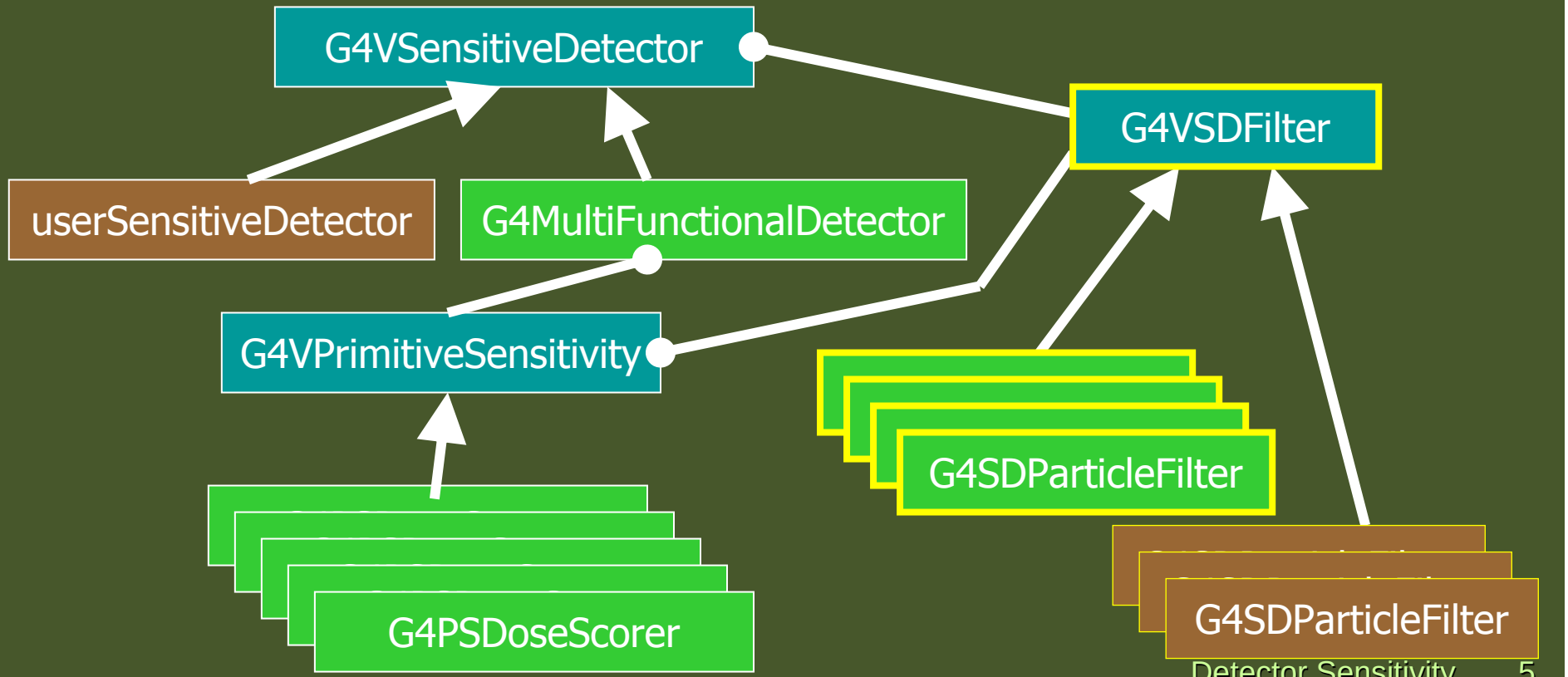


G4THitsMap

- Each G4VPrimitiveSensitivity class generates one hits collection per event. By registering more than one classes of G4VPrimitiveSensitivity, G4MultiFunctionalDetector generates more than one collections.
- G4THitsMap template class (an alternative to G4THitsCollection) will be introduced. It is also a derived class of G4VHitsCollection.
  - It is more convenient for scoring purposes. It does NOT mandate G4VHit concrete class to be stored, but for example a simple double value can be mapped with a copy number.
  - All new primitive sensitivity classes use G4THitsMap.

G4VSDFilter

- New class G4VSDFilter will be introduced. It 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.



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);
     G4VPrimitiveSensitivity* totalDose = new G4PSDoseDeposit("TotalDose");
     myScorer->Register(totalDose);
}
```

No need of implementing sensitive detector !

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 …
};
```

Note :

This sample code uses newly introducing concrete sensitivity classes.

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