Interaction with kernel - part 2

Vienna workshop on simulations 2024
25th April

Carlo Mancini Terracciano
carlo.mancini-terracciano@uniroma1.it

Outline

- Sensitive detectors
- Native scorers
- How to retrive information from native scorers

Sensitive detectors

- A logical volume becomes sensitive if it has a pointer of **G4VSensitiveDetector**
- Actually to a concrete class that inherits from it
- A sensitive detector (SD) can be instantiated several times, assigning each instance to a different logical volume
- SD objects must have unique detector names
- A logical volume can only have one SD object attached
  (But you can implement your detector to have many functionalities)
- Two possibilities to make use of the SD functionality:
  - Create your own sensitive detector (defining a class inheriting from **G4VSensitiveDetector**)
    - Highly customizable (not shown in this short course)
  - Use Geant4 built-in tools: Primitive scorers

Adding sensitivity to a logical volume

- Create an instance of a sensitive detector and register it to the Sensitive Detector Manager
- Assign the pointer of your SD to the logical volume of your detector geometry
- Must be done in **ConstructSDandField()** of the user geometry class

```
G4VSensitiveDetector* mySensitive
  = new MySensitiveDetector(SDname="MyDetector");

G4SDManager* sdMan =G4SDManager::GetSDMpointer();
sdMan->AddNewDetector(mySensitive);

logicVol->SetSensitiveDetector(mySensitive);
//Or:
//SetSensitiveDetector("LVname",mySensitive);
```

Native scorers

- Geant4 provides a number of primitive scorers, each one accumulating one physics quantity (e.g. total dose) for an event
- This is alternative to the custom sensitive detectors (not shown in this course), which can be used with full flexibility to have complete control
- It is convenient to use primitive scorers instead of user-defined sensitive detectors when:
  - you are not interested in recording each individual step, but accumulating physical quantities for an event or a run
  - you have not too many scorers

G4MultiFunctionalDetector

- **G4MultiFunctionalDetector** is a concrete class derived from **G4VSensitiveDetector**
- It has to be assigned to a logical volume as a sensitive detector
- It takes an arbitrary number of **G4VPrimitiveScorer** classes, to define the scoring quantities that you need
- Each **G4VPrimitiveScorer** accumulates one physics quantity for each physical volume
- E.g. **G4PSDoseScorer** (a concrete class of **G4VPrimitiveScorer** provided by Geant4) accumulates dose for each cell
- By using this approach, there's no need to implement sensitive detector and hit classes!

G4VPrimitiveScorer

- Primitive scorers (classes inheriting from **G4VPrimitiveScorer**) have to be registered to the **G4MultiFunctionalDetector**
  - RegisterPrimitive()
  - RemovePrimitive()
- They are designed to score one kind of quantity (surface flux, total dose) and to generate one hit collection per event
  - automatically named as: <MultiFunctionalDetectorName>/<PrimitiveScorerName>
- hit collections can be retrieved in the EventAction or RunAction (as those generated by sensitive detectors)
- do not share the same primitive scorer object among multiple **G4MultiFunctionalDetector** objects (results may mix up!)
- Create as many instances of the scorer as needed

Example

```
MyDetectorConstruction::ConstructSDandField()
{
  G4MultiFunctionalDetector* myScorer = new
    G4MultiFunctionalDetector("myCellScorer");
  myCellLog->SetSensitiveDetector(myScorer);

  G4VPrimitiveScorer* totalSurfFlux =
    new G4PSFlatSurfaceFlux("TotalSurfFlux");
  myScorer->RegisterPrimitive(totalSurfFlux);
  G4VPrimitiveScorer* totalDose =
    new G4PSDoseDeposit("TotalDose");
  myScorer->RegisterPrimitive(totalDose);
}
```

instantiate a multi-functional detector

attach it to a volume

create two primitive scorers (surface flux and total dose) and register them

Some primitive scorers

- Concrete Primitive Scorers
  (Application Developers Guide 4.4.5)
  https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/html/Detector/hit.html#concrete-classes-of-g4vprimitivescorer

  - Track length: G4PSTrackLength, G4PSPassageTrackLength
  - Deposited energy: G4PSEnergyDepsit, G4PSDoseDeposit
  - Current/Flux: G4PSFlatSurfaceCurrent, G4PSSphereSurfaceCurrent, G4PSPassageCurrent, G4PSFlatSurfaceFlux, G4PSCellFlux, G4PSPassageCellFlux
  - Others: G4PSMinKinEAtGeneration, G4PSNofSecondary, G4PSNofStep, G4PSCellCharge

G4VSDFilter

- You can also filter which kind of tracks you want to consider (e.g. protons only)

- Attaching a **G4VSDFilter** to **G4VPrimitiveScorer**:
  - **G4SDChargeFilter** (accepts only charged particles)
  - **G4SDNeutralFilter** (accepts only neutral particles)
  - **G4SDKineticEnergyFilter** (accepts tracks in a defined range of kinetic energy)
  - **G4SDParticleFilter** (accepts tracks of a given particle type)
  - **G4VSDFilter** (base class to create user-customized filters)

Example

```
MyDetectorConstruction::ConstructSDandField()
{
  G4VPrimitiveScorer* protonSurfFlux
      = new G4PSFlatSurfaceFlux("pSurfFlux");

  G4VSDFilter* protonFilter
     = new G4SDParticleFilter("protonFilter");
  protonFilter->Add("proton");

  protonSurfFlux->SetFilter(protonFilter);

  myScorer->RegisterPrimitive(protonSurfFlux);
}
```

create a primitive scorer (surface flux), as before

create a particle filter for protons

register the filter to the primitive scorer

register the scorer to the multifunctional detector (as before)

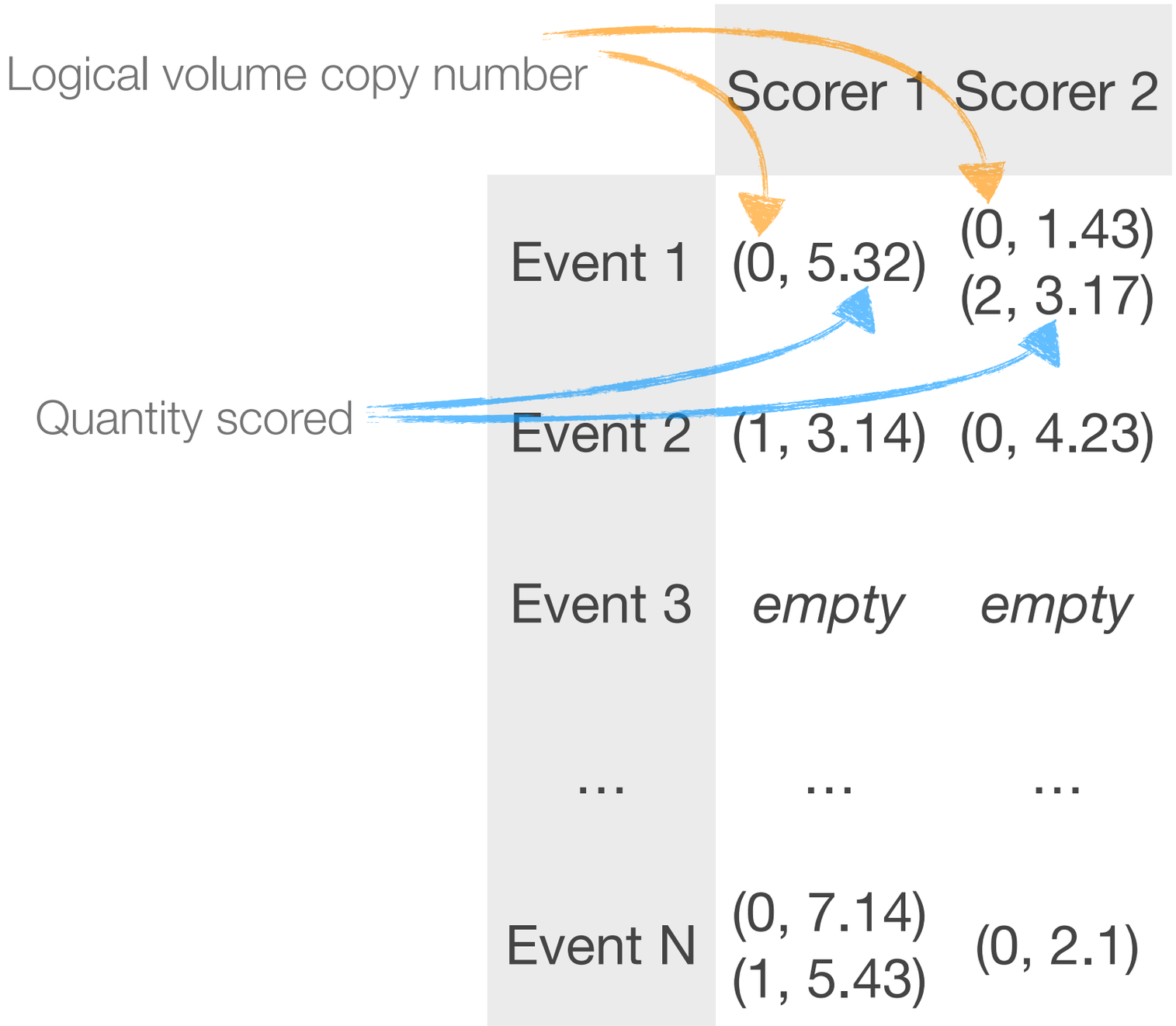
Hot to retrive information

- At the conclusion of a simulation, extracting data from scorers is essential
- Each scorer generates a hit collection
- This collection is associated with the specific **G4Event** instance
- Hit collections can be accessed at the event's end using an integer ID
- Hit collections are organized as **G4THitsMap<G4double>***
- This allows for iteration over individual entries
- The **operator+=** is provided for hit collections, it automatically aggregates all hits, i.e. no need for manual looping and summation!

How to retrive information

	Scorer 1	Scorer 2
Event 1	(0, 5.32)	(0, 1.43) (2, 3.17)
Event 2	(1, 3.14)	(0, 4.23)
Event 3	*empty*	*empty*
...	...	...
Event N	(0, 7.14) (1, 5.43)	(0, 2.1)

How to retrive information



How to retrive information

	Scorer 1	Scorer 2
Event 1	(0, 5.32)	(0, 1.43) (2, 3.17)
Event 2	(1, 3.14)	(0, 4.23)
Event 3	*empty*	*empty*
…	…	…
Event N	(0, 7.14) (1, 5.43)	(0, 2.1)

HCofThisEvent

(hit collection of this event)

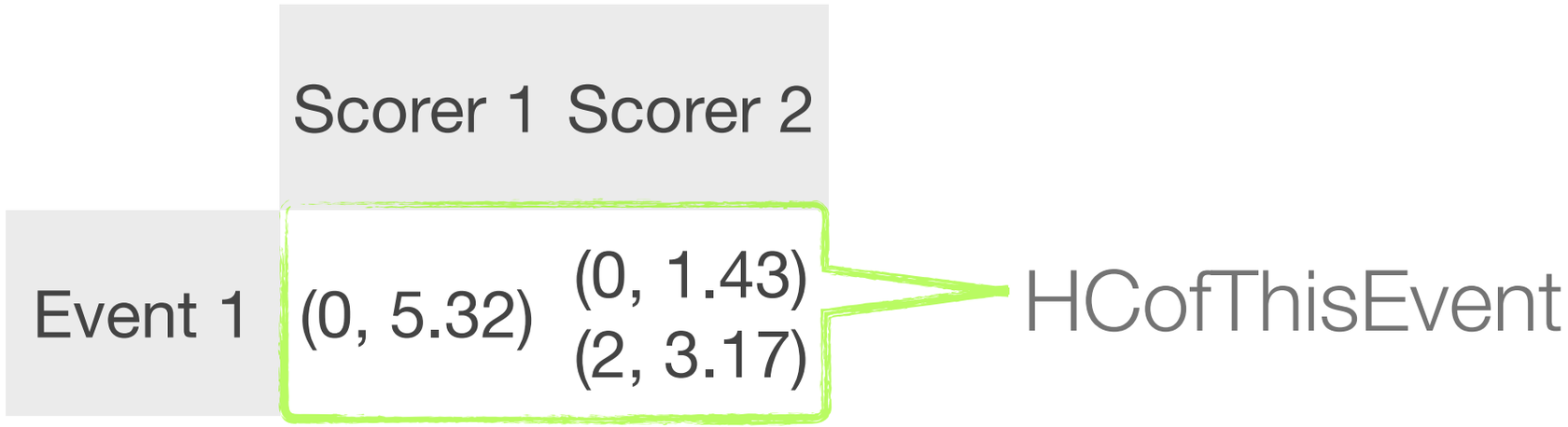
Hot to retrive information

- Retrieve the ID for a collection using its name

```
G4int collID = G4SDManager::GetSDMpointer()
  ->GetCollectionID("myCellScorer/TotalSurfFlux");
```

- Get all Hits Collections available in this event

```
G4HCofThisEvent* HCE = event->GetHCofThisEvent();
```



Hot to retrive information

- Get all Hits Collections available in this event

```
G4HCofThisEvent* HCE = event->GetHCofThisEvent();
```

- Get the Hit Collection with the given ID (a cast is needed)

```
G4THitsMap<G4double>* evtMap =
   static_cast<G4THitsMap<G4double>*> (HCE->GetHC(collID));
```

	Scorer 1	Scorer 2
Event 1	(0, 5.32)	(0, 1.43) (2, 3.17)

evtMap

Hot to retrive information

- Get the Hit Collection with the given ID (a cast is needed

```
G4THitsMap<G4double>* evtMap =
  static_cast<G4THitsMap<G4double>*> (HCE->GetHC(collID));
```

- Iterate through each entry in the Hit Collection (HC)

```
for (auto pair : *(evtMap->GetMap())) {
   G4double flux = *(pair.second);
   G4int copyNb  = pair.first;
}
```

- '**copyNb**' serves as the map key, while the associated field represents the actual data

Hot to retrive information

- Iterate through each entry in the Hit Collection (HC)

```
for (auto pair : *(evtMap->GetMap())) {
    G4double flux = *(pair.second);
    G4int copyNb  = *(pair.first);
}
```

- 'copyNb' serves as the map key, while the associated field represents the actual data

	Scorer 1	Scorer 2
Event 1	(0, 5.32)	(0, 1.43) (2, 3.17)

pair

Hands on

- https://geant4.lns.infn.it/vienna2024/task4/task4c.html