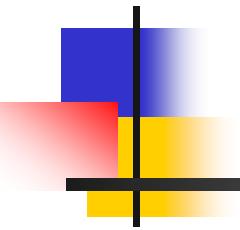
Interaction with the Geant4 kernel – part 1

Luciano Pandola

INFN – Laboratori Nazionali del Sud

Partially based on a presentation by G.A.P. Cirrone (INFN-LNS)



Part I: The main ingredients

Optional user classes - 1

- Five concrete base classes whose virtual member functions the user may override to gain control of the simulation at various stages
  - G4User**Run**Action
  - G4User**Event**Action
  - G4User**Tracking**Action
  - G4User**Stacking**Action
  - G4User**Stepping**Action

e.g. actions to be done at the beginning and end of each event

- Each member function of the base classes has a dummy implementation (**not** purely virtual)
  - Empty implementation: does nothing

Optional user classes - 2

- The user may implement the member functions he desires in his/her derived classes
  - E.g. one may want to perform some action at each tracking step
- Objects of user action classes must be **registered** to the **G4(MT)RunManager** via the ActionInitialization

```
runManager->SetUserAction(new
  MyActionInitialization);
```

MyActionInitialization (MT mode)

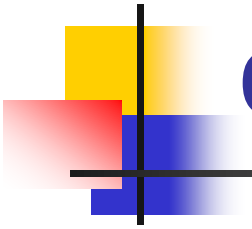
- Register thread-local user actions

```
void MyActionInitialization::Build() const
{
  //Set mandatory classes
  SetUserAction(new MyPrimaryGeneratorAction());
  // Set optional user action classes
  SetUserAction(new MyEventAction());
  SetUserAction(new MyRunAction());
}
```

- Register RunAction for the master

```
void MyActionInitialization::BuildForMaster() const
{
  // Set optional user action classes
SetUserAction(new MyMasterRunAction());
}
```

Geant4 terminology: an overview

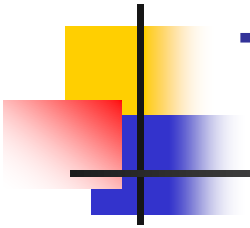


- The following keywords are often used in Geant4
  - **Run, Event, Track, Step**
  - **Processes**: At Rest, Along Step, Post Step
  - **Cut** (or production threshold)

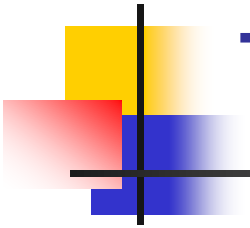
The Run (G4Run)

- As an analogy with a real experiment, a run of Geant4 starts with '**Beam On**'
- Within a run, the User **cannot change**
  - The detector setup
  - The physics setting (processes, models)
- A Run is a collection of events with the same detector and physics conditions
- At the beginning of a Run, geometry is optimised for navigation and cross section tables are (re)calculated
- The G4RunManager class manages the processing of each Run, represented by:
  - **G4Run** class
  - **G4UserRunAction** for an optional User hook

The Event (G4Event)



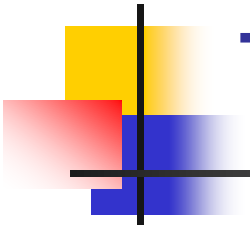
- An Event is the basic unit of simulation in Geant4
- At the beginning of processing, primary tracks are generated and they are pushed into a stack
- A track is popped up from the stack one-by-one **and** '**tracked**'
  - Secondary tracks are also pushed into the stack
  - When the stack gets empty, the processing of the event is completed
- **`G4Event`** class represents an event. At the end of a successful event it has:
  - List of primary vertices and particles (as input)
  - Hits and Trajectory collections (as outputs)
- **`G4EventManager`** class manages the event
- **`G4UserEventAction`** is the optional User hook



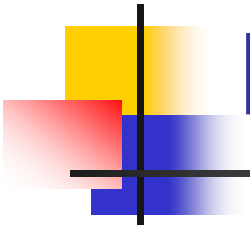
The Step (G4Step)

- **`G4Step`** represents a step in the particle propagation
- A G4Step object stores transient information of the step
  - In the tracking algorithm, G4Step is updated each time a process is invoked
- You can extract information from a step after the step is completed
  - Both, the **`ProcessHits()`** method of your sensitive detector and **`UserSteppingAction()`** of your step action class file get the pointer of G4Step
  - Typically , you may retrieve information in these functions (for example fill histograms in Stepping action)

The Track (G4Track)



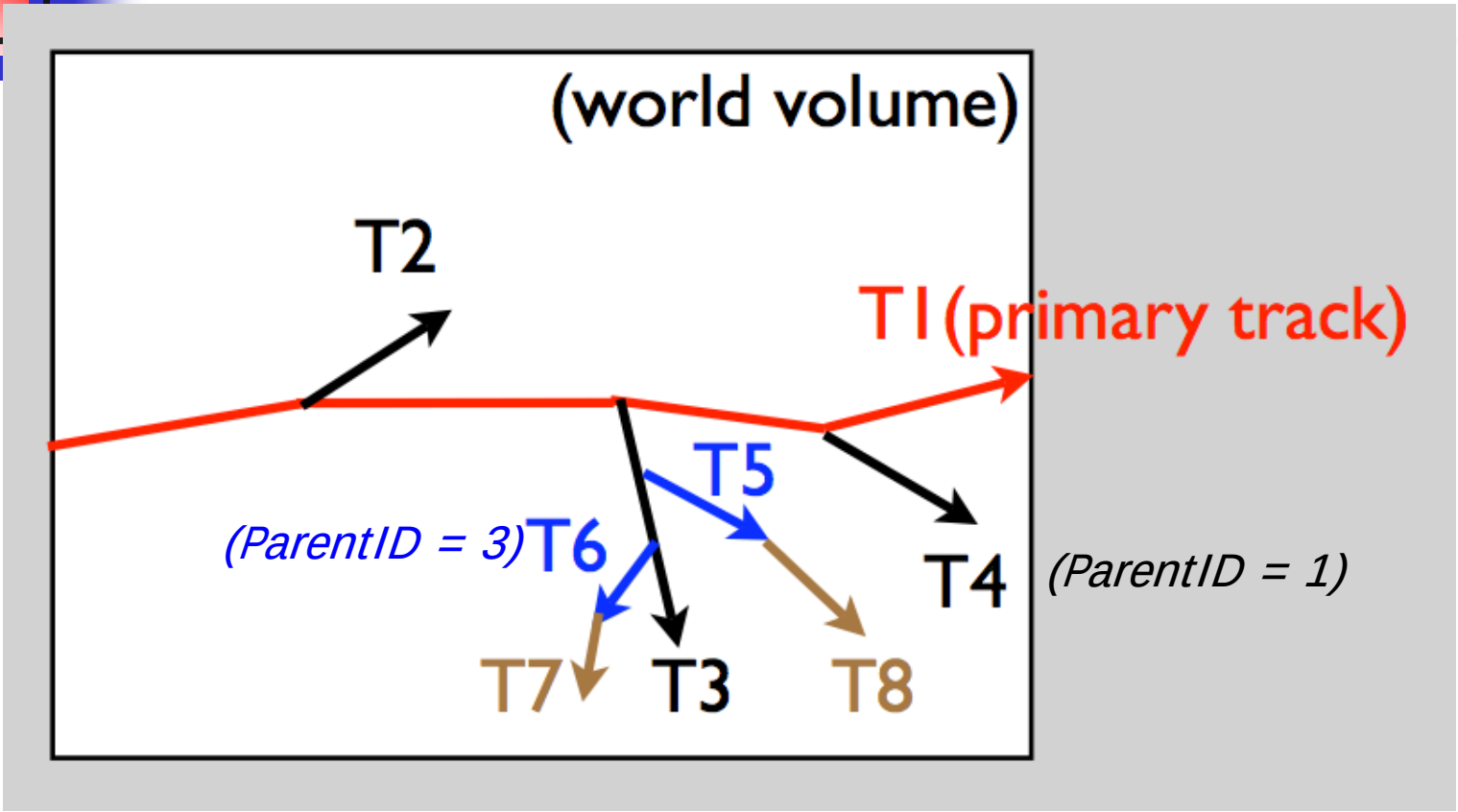
- The Track is a **snapshot of a particle** and it is represented by the **`G4Track`** class
  - It keeps 'current' information of the particle (i.e. energy, momentum, position, polarization, ..)
  - It is updated after every step
- The track object is deleted when
  - It goes outside the world volume
  - It disappears in an interaction (decay, inelastic scattering)
  - It is slowed down to zero kinetic energy and there are no 'AtRest' processes
  - It is manually killed by the user
- No track object **persists** at the end of the event
- **`G4TrackingManager`** class manages the tracking
- **`G4UserTrackingAction`** is the optional User hook



Run, Event and Tracks

- **One Run consists of**
  - Event #1 (track #1, track #2, ……)
  - Event #2 (track #1, track #2, ……)
  - ……..
  - Event #N (track #1, track #2, ……)

Example of an Event and Tracks



- Tracking order follows **'last in first out'** rule:
  T1 -> T4 -> T3 -> T6 -> T7 -> T5 -> T8 -> T2

```
**********************************************************************************************************
* G4Track Information:   Particle = e-,   Track ID = 87,   Parent ID = 1
**********************************************************************************************************

Step#    X(mm)     Y(mm)     Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0 -1.87e+03      5.63     -5.52    0.0326        0        0         0 physicalTreatmentRoom initStep
    1 -1.87e+03      5.85     -4.72     0.032 0.000545    0.924     0.924 physicalTreatmentRoom msc
    2 -1.87e+03      5.92      -3.9    0.0317  0.00036    0.928      1.85 physicalTreatmentRoom msc
    3 -1.87e+03      5.89     -3.65    0.0289  0.00013      0.3      2.15 physicalTreatmentRoom eIoni
    :----- List of 2ndaries - #SpawnInStep=  1(Rest= 0,Along= 0,Post= 1), #SpawnTotal=  1 ---------------
    : -1.87e+03      5.89     -3.65   0.00258                  e-
    :---------------------------------------------------------- EndOf2ndaries Info ---------------
    4 -1.87e+03      5.81     -2.87    0.0279  0.00104    0.928      3.08 physicalTreatmentRoom msc
    5 -1.87e+03      5.35     -2.11    0.0273 0.000654    0.928      4.01 physicalTreatmentRoom msc
    6 -1.87e+03      5.01     -1.28    0.0248  0.00249    0.928      4.94 physicalTreatmentRoom msc
    7 -1.87e+03      5.03     -0.37    0.0231  0.00163    0.928      5.87 physicalTreatmentRoom msc
    8 -1.87e+03      4.78     0.503     0.022  0.00109    0.928      6.79 physicalTreatmentRoom msc
    9 -1.87e+03      4.64      1.35    0.0202  0.00184    0.928      7.72 physicalTreatmentRoom msc
   10 -1.87e+03      4.68      2.26    0.0181  0.00204    0.928      8.65 physicalTreatmentRoom msc
   11 -1.87e+03      4.63      2.46    0.0165 0.000345    0.231      8.88 physicalTreatmentRoom eIoni
    :----- List of 2ndaries - #SpawnInStep=  1(Rest= 0,Along= 0,Post= 1), #SpawnTotal=  2 ---------------
    : -1.87e+03      4.63      2.46   0.00133                  e-
    :---------------------------------------------------------- EndOf2ndaries Info ---------------
   12 -1.87e+03       4.6      2.49    0.0125        0   0.0383      8.92 physicalTreatmentRoom eIoni
    :----- List of 2ndaries - #SpawnInStep=  1(Rest= 0,Along= 0,Post= 1), #SpawnTotal=  3 ---------------
    : -1.87e+03       4.6      2.49   0.00402                  e-
    :---------------------------------------------------------- EndOf2ndaries Info ---------------
```

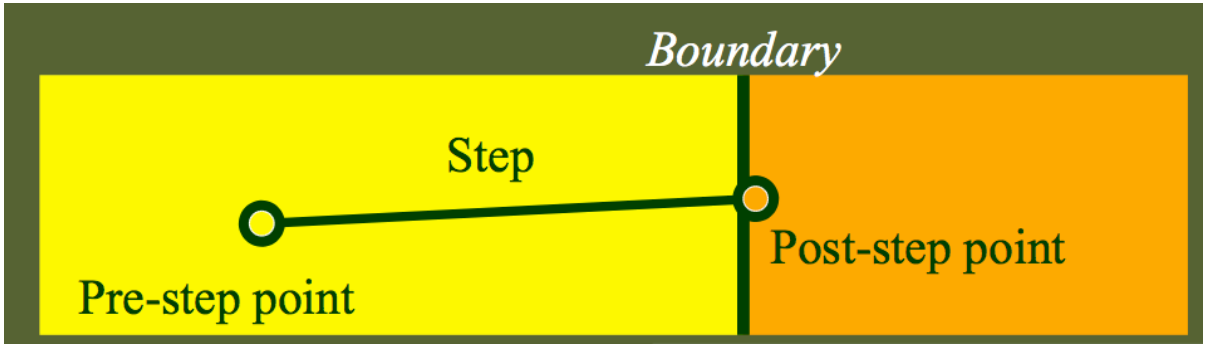
```
**********************************************************************************************************
* G4Track Information:   Particle = e-,   Track ID = 242,   Parent ID = 87
**********************************************************************************************************

Step#    X(mm)     Y(mm)     Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0 -1.87e+03       6.1      5.41   0.00138        0        0         0 physicalTreatmentRoom initStep
    1 -1.87e+03      6.11      5.39  0.000253  0.00112   0.0481    0.0481 physicalTreatmentRoom msc
    2 -1.87e+03      6.12      5.39         0 0.000253   0.0088    0.0569 physicalTreatmentRoom eIoni
```

Example: retrieving information from tracks

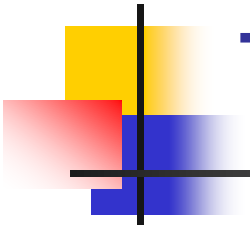
```
// retrieving information from tracks (given the G4Track object "track"):

if(track -> GetTrackID() != 1)  {
    G4cout << "Particle is a secondary" << G4endl;

    // Note in this context, that primary hadrons might loose their identity
    if(track -> GetParentID() == 1)
                    G4cout << "But parent was a primary" << G4endl;

    G4VProcess* creatorProcess = track -> GetCreatorProcess();

    if(creatorProcess -> GetProcessName()  == "LowEnergyIoni") {
          G4cout << "Particle was created by the Low-Energy " <<
                  << "Ionization process" << G4endl;
    }
}
```

The Step in Geant4

- The G4Step has the information about the **two points** (pre-step and post-step) and the **'delta'** information of a particle (energy loss on the step, ...)
- Each point knows the volume (and the 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



- **`G4SteppingManager`** class manages processing a step; a 'step' in represented by the **`G4Step`** class
- **`G4UserSteppingAction`** is the optional User hook



The G4Step object

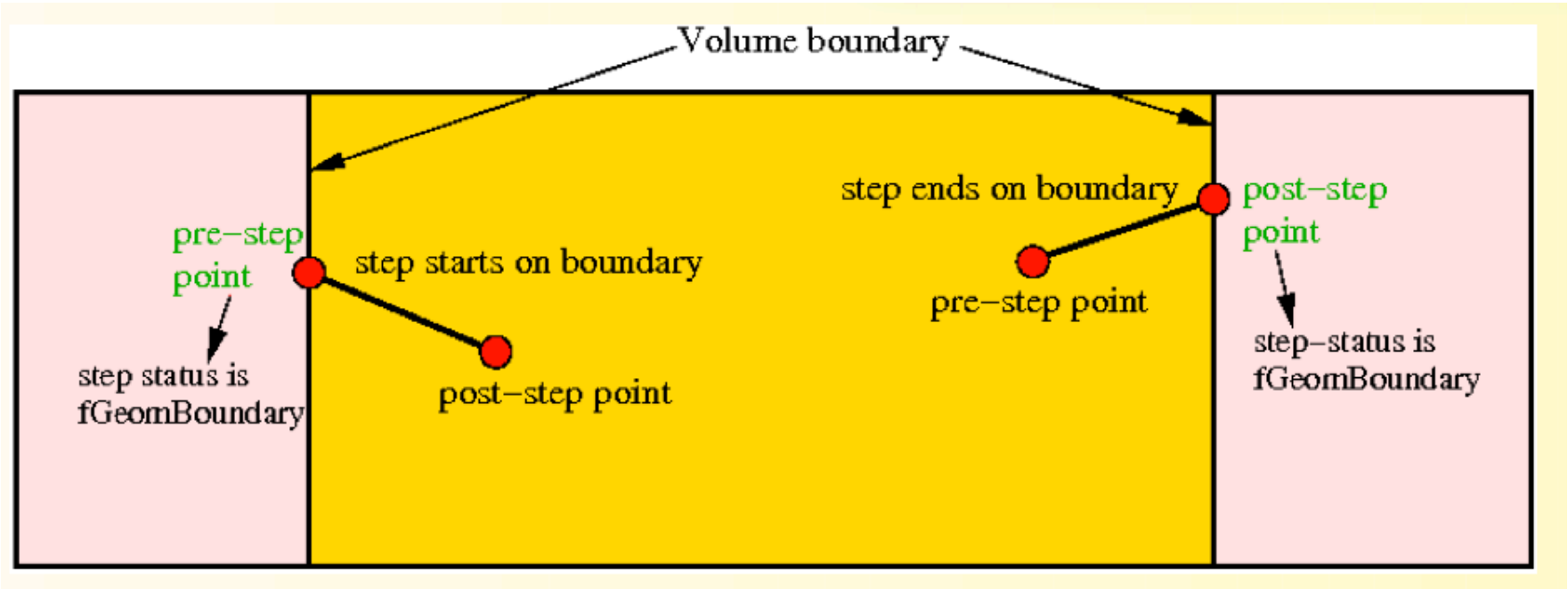
- A **`G4Step`** object contains
  - The two endpoints (pre and post step) so one has access to the volumes containing these endpoints
  - Changes in particle properties between the points
    - Difference of particle energy, momentum, ...
    - Energy deposition on step, step length, time-of-flight, ...
  - A pointer to the associated **`G4Track`** object
- **`G4Step`** provides many Get methods to access these information or object istances
  - **`G4StepPoint* GetPreStepPoint(),`** .....

The geometry boundary

- To check, if a step ends on a boundary, one may compare if the physical volume of pre and post-step points are equal
- One can also use the **step status**
  - Step Status provides information about the process that restricted the step length
  - It is attached to the step points: the pre has the status of the previous step, the post of the current step
  - If the status of POST is "**`fGeometryBoundary`**" the step ends on a volume boundary (does not apply to word volume)
  - To check if a step starts on a volume boundary you can also use the step status of the PRE-step point

Step concept and boundaries

Illustration of step starting and ending on boundaries



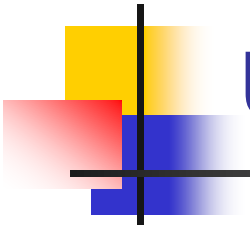
Geant4 terminology: an overview

	Object	Description
Run	G4Run	Largest unit of simulation, that consist of a sequence of events: If a defined number of events was processed a run is finished.
Event	G4Event	Basic simulation unit in Geant4: If a defined number of primary tracks and all resulting secondary tracks were processed an event is over.
Track	G4Track	A track is NOT a collection of steps: It is a snapshot of the status of a particle after a step was completed (but it does NOT record previous steps). A track is deleted, if the particle leaves world, has zero kinetic energy, ...
Step	G4Step	Represents a particle step in the simulation and includes two points (pre-step point and post-step point).

Example of usage of the hook user classes - 1

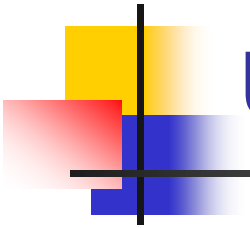
- **`G4UserRunAction`**
  - Has two methods (**`BeginOfRunAction()`** and **`EndOfRunAction()`**) and can be used e.g. to initialise, analyse and store histogram
  - Everything User want to know at this stage
- **`G4UserEventAction`**
  - Has two methods (**`BeginOfEventAction()`** and **`EndOfEventAction()`**)
  - One can apply an event selection, for example
  - Access the hit-collection and perform the event analysis

Example of usage of the hook user classes - 2



- **`G4UserStakingAction`**
  - Classify priority of tracks
- **`G4UserTrackingAction`**
  - Has two methods (**`PreUserTrakingAction()` `and PostUserTrackinAction()`**)
  - For example used to decide if trajectories should be stored
- **`G4UserSteppingAction`**
  - Has a method which is invoked at the end of a step

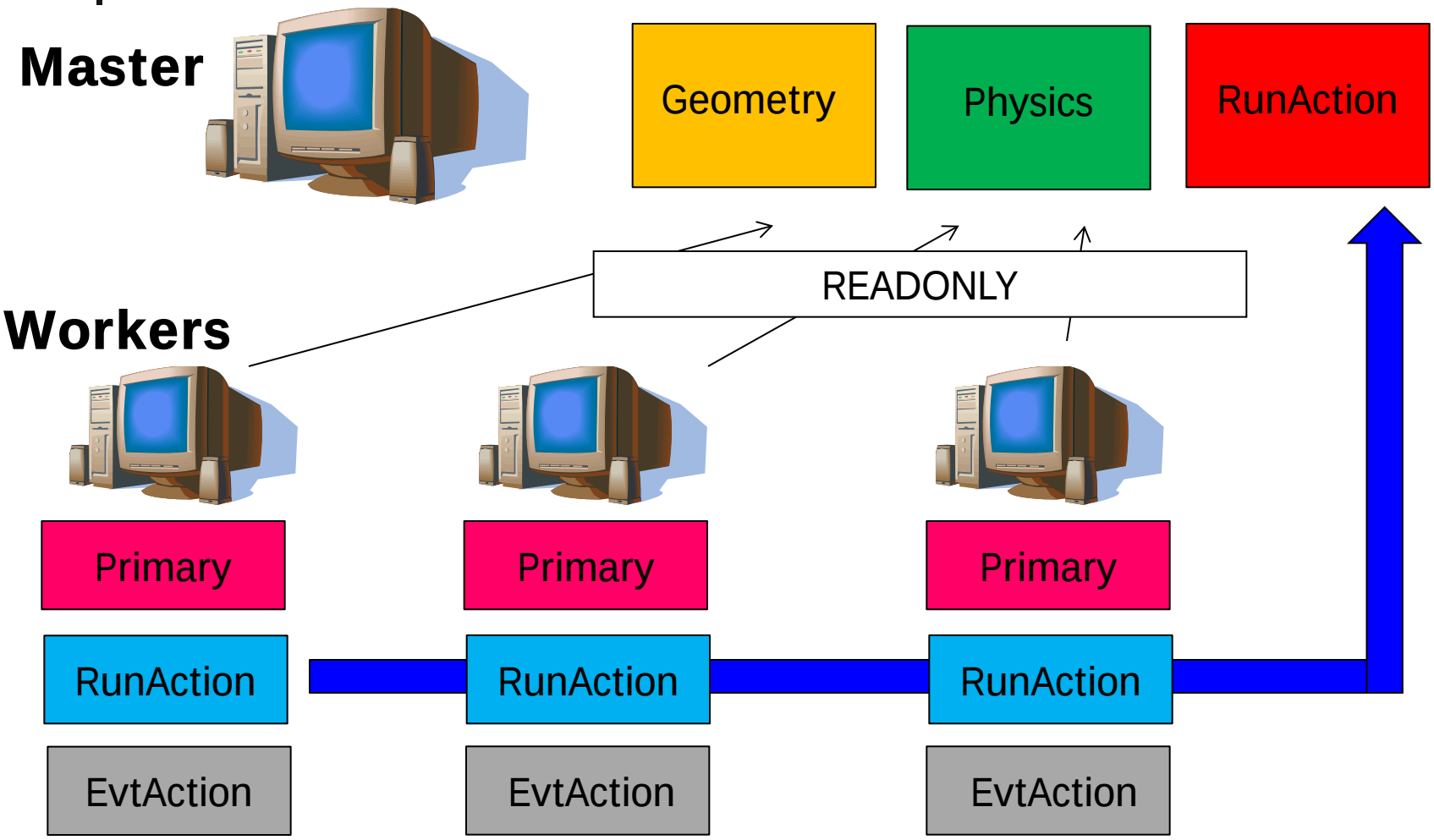
Example of usage of the hook user classes - 3

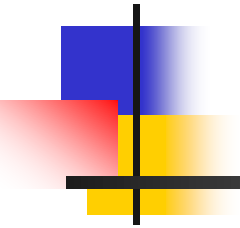


- Derived **`G4Run`** class
  - User-custom class which derives from **`G4Run`** of Geant4

    `MyRun : public G4Run()`
- Can overwrite two methods:
  - **`RecordEvent()`**
    - Called at the end of each event: **alternative to EndOfEventAction()** of the EventAction class
  - **`Merge()`**
    - Called at the end of each run by the **master**
- **When/why** to use it?
  - Convenient in **MT-mode**, because allows the **merging** of information (global quantities) from **thread-local runs** into the master
    - UserEventAction is thread-local

Concept for multi-thread ...





Part II: Retrieving information from steps and tracks

Example: check if step is on boundaries

```
// in the source file of your user step action class:

#include "G4Step.hh"

UserStepAction::UserSteppingAction(const G4Step* step) {

  G4StepPoint* preStepPoint = step -> GetPreStepPoint();
  G4StepPoint* postStepPoint = step -> GetPostStepPoint();

  // Use the GetStepStatus() method of G4StepPoint to get the status of the
  // current step (contained in post-step point) or the previous step
  // (contained in pre-step point):
  if(preStepPoint -> GetStepStatus() == fGeomBoundary) {

     G4cout << "Step starts on geometry boundary" << G4endl;
  }
  if(postStepPoint -> GetStepStatus() == fGeomBoundary) {

     G4cout << "Step ends on geometry boundary" << G4endl;

     // You can retrieve the material of the next volume through the
     // post-step point :
     G4Material* nextMaterial = step -> GetPostStepPoint()->GetMaterial();
  }
}
```

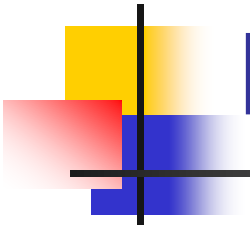
Example: step information in SD

```
// in source file of your sensitive detector:

MySensitiveDetector::ProcessHits(G4Step* step,
                                 G4TouchableHistory*) {

   // Total energy deposition on the step (= energy deposited by energy loss
   //  process and energy of secondaries that were not created since their
   // energy was < Cut):
   G4double energyDeposit = step -> GetTotalEnergyDeposit();

   // Difference of energy , position and momentum of particle between pre-
   // and post-step point
   G4double deltaEnergy = step -> GetDeltaEnergy();
   G4ThreeVector deltaPosition = step -> GetDeltaPosition();
   G4double deltaMomentum = step -> GetDeltaMomentum();

   // Step length
   G4double stepLength = step -> GetStepLength();

}
```

Something more about tracks

- After each step the track can change its state
- The status can be (in red can only be set by the User)

Track Status	Description
fAlive	The particle is continued to be tracked
fStopButAlive	Kin. Energy = 0, but AtRest process will occur
fStopAndKill	Track has lost identity (has reached world boundary, decayed, ...), Secondaries will be tracked
fKillTrackAndSecondaries	Track and its secondary tracks are killed
fSuspend	Track and its secondary tracks are suspended (pushed to stack)
fPostponeToNextEvent	Track but NOT secondary tracks are postponed to the next event (secondaries are tracked in current event)



Particles in Geant4

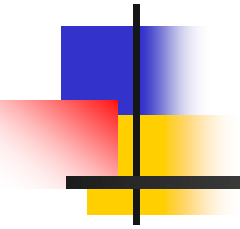
- A particle in general has the following **three sets** of **properties**:
  - Position/geometrical info
    - **`G4Track`** class (representing a particle to be tracked)
  - Dynamic properties: momentum, energy, spin,..
    - **`G4DynamicParticle`** class
  - Static properties: rest mass, charge, life time
    - **`G4ParticleDefinition`** class
- All the **`G4DynamicParticle`** objects of the same kind of particle share the same **`G4ParticleDefinition`**

Particles in Geant4

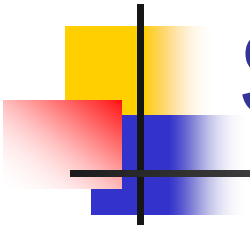
Class	What does it represent?	What does it contain?
G4Track	Represents a particle that travels in space and time	Information relevant to tracking the particle, e.g. position, time, step,..., and *dynamic information*
G4DynamicParticle	Represents a particle that is subject to interactions with matter	Dynamic information, e.g. particle momentum, kinetic energy, ..., and *static information*
G4ParticleDefinition	Defines a physical particle	Static information, e.g. particle mass, charge, ... Also physics processes relevant to the particle

Examples: particle information from step/track

```
#include “G4ParticleDefinition.hh“
#include “G4DynamicParticle.hh“
#include “G4Step.hh“
#include “G4Track.hh“

// Retrieve from the current step the track  (after PostStepDoIt of step is
// completed):
G4Track* track = step -> GetTrack();

 // From the track you can obtain the pointer to the dynamic particle:
const G4DynamicParticle* dynParticle = track -> GetDynamicParticle();

 // From the dynamic particle, retrieve the particle definition:
G4ParticleDefinition* particle = dynParticle -> GetDefinition();

 // The dynamic particle class contains e.g. the kinetic energy after the step:
G4double kinEnergy = dynParticle -> GetKineticEnergy();

 // From the particle definition class you can retrieve static information like
 // the particle name:
G4String particleName = particle -> GetParticleName();

 G4cout <<  particleName <<  “: kinetic energy of “
          <<  kinEnergy/MeV <<  “ MeV“
          <<  G4endl;
```



Part III: Sensitive Detectors



Sensitive Detector (SD)

- A logical volume becomes sensitive if it has a pointer to a **sensitive detector** (**`G4VSensitiveDetector`**)
  - A sensitive detector can be instantiated several times, where the instances are assigned to different logical volumes
    - Note that 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 (using class inheritance)
    - Highly **customizable**
  - Use Geant4 built-in tools: Primitive scorers

Adding sensitivity to a logical volume

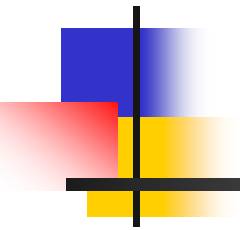
- Create an instance of a sensitive detector
- Assign the pointer of your SD to the logical volume of your detector geometry
- Must be done in **`ConstructSDandField`**() of the user geometry class

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

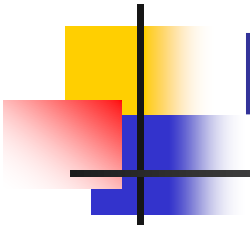
```
boxLogical->SetSensitiveDetector(mySensitive);
```
assign to logical volume

(or)

```
SetSensitiveDetector("LVname",mySensitive);
```
assign to logical volume (alternative)



Part IV: Native Geant4 scoring



Extract useful information

- 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 **customized** sensitive detectors (see later in this lecture), which can be used with full flexibility to gain 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 should be **assigned to a logical volume** as a **kind of** (ready-for-the-use) **sensitive detector**
- It takes an arbitrary number of **`G4VPrimitiveSensitivity`** classes, to define the scoring quantities that you need
  - Each **`G4VPrimitiveSensitivity`** accumulates one physics quantity for each physical volume
  - E.g. **`G4PSDoseScorer`** (a concrete class of **`G4VPrimitiveSensitivity`** provided by Geant4) accumulates dose for each cell
- By using this approach, **no need to implement sensitive detector** and **hit classes**!

G4VPrimitiveSensitivity

- Primitive scorers (classes derived from **`G4VPrimitiveSensitivity`**) 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!)

myCellScorer/TotalSurfFlux
myCellScorer/TotalDose

For example ...

```
MyDetectorConstruction::ConstructSDandField()
{
   G4MultiFunctionalDetector* myScorer = new
   G4MultiFunctionalDetector("myCellScorer");

    myCellLog->SetSensitiveDetector(myScorer);

    G4VPrimitiveSensitivity* totalSurfFlux = new
      G4PSFlatSurfaceFlux("TotalSurfFlux");
    myScorer->RegisterPrimitive(totalSurfFlux);
    G4VPrimitiveSensitivity* totalDose = new
      G4PSDoseDeposit("TotalDose");
    myScorer->RegisterPrimitive(totalDose);
}
```

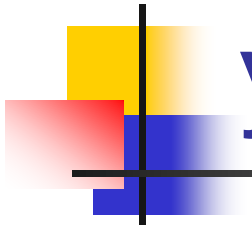
instantiate multi-functional detector

attach to volume

create a primitive scorer (surface flux) and register it

create a primitive scorer (total dose) and register it

Some primitive scorers that you may find useful

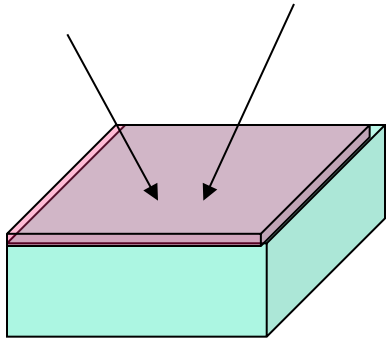


- Concrete Primitive Scorers (→ Application Developers Guide 4.4.5)
  - Track length
    - G4PSTrackLength, G4PSPassageTrackLength
  - Deposited energy
    - G4PSEnergyDepsit, G4PSDoseDeposit
  - Current/Flux
    - G4PSFlatSurfaceCurrent, G4PSSphereSurfaceCurrent,G4PSPassageCurrent, G4PSFlatSurfaceFlux, G4PSCellFlux, G4PSPassageCellFlux
  - Others
    - G4PSMinKinEAtGeneration, G4PSNofSecondary, G4PSNofStep, G4PSCellCharge

A closer look at some scorers

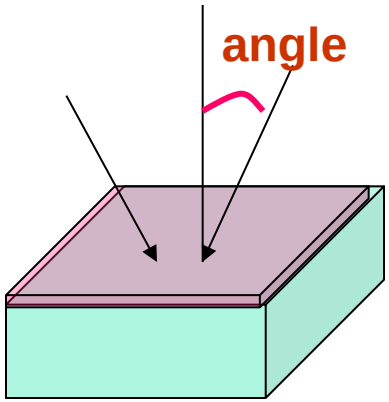
SurfaceCurrent :

Count number of injecting particles at defined surface.



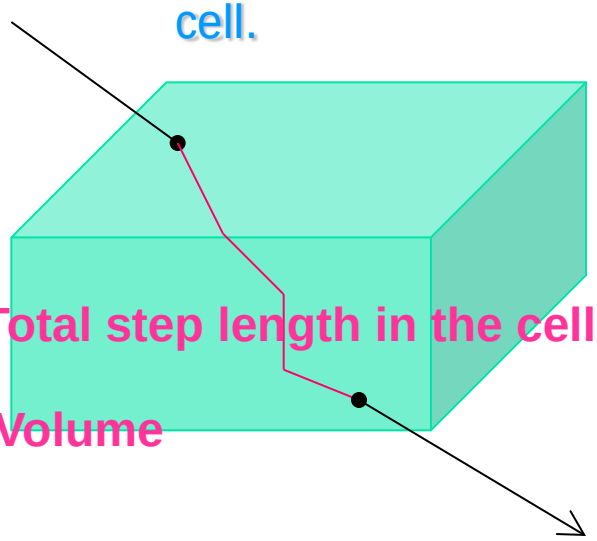
CellFlux :

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



SurfaceFlux :

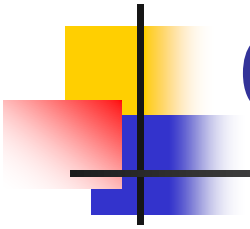
Sum up 1/cos(angle) of injecting particles at defined surface



L : Total step length in the cell

V : Volume

V : Volume



G4VSDFilter

- A **`G4VSDFilter`** can be attached to `G4VPrimitiveSensitivity` to define **which kind of tracks** have to be scored (e.g. one wants to know surface flux of protons only)
  - `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)

For example ...

```
MyDetectorConstruction::ConstructSDandField()
{
   G4VPrimitiveSensitivity* protonSurfFlux
   = new G4PSFlatSurfaceFlux("pSurfFlux");
    G4VSDFilter* protonFilter = new
       G4SDParticleFilter("protonFilter");
    protonFilter->Add("proton");

    protonSurfFlux->SetFilter(protonFilter);

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

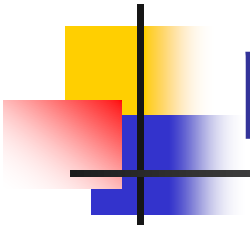
create a primitive scorer (surface flux), as before

create a particle filter and add protons to it

register the filter to the primitive scorer

register the scorer to the multifunc detector (as shown before)

How to retrieve information - part 1



- At the end of the day, one wants to retrieve the information from the scorers
  - True also for the customized hits collection
- Each scorer creates a hit collection, which is attached to the **`G4Event`** object
  - Can be retrieved and read at the end of the event, using an integer ID
  - Hits collections mapped as **`G4THitsMap<G4double>*`** so can loop on the individual entries
  - Operator **`+=`** provided which automatically sums up hits (no need to loop)

How to retrieve information – part 2

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

Get **ID** for the collection (given the name)

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

Get **all HC** available in this event

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

Get the HC with the **given ID** (need a cast)

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

Loop over the **individual entries** of the HC: the key of the map is the copyNb, the other field is the real content

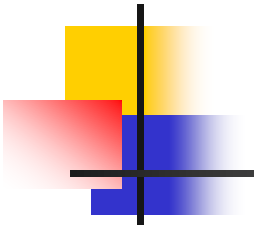
Command-based scoring

Thanks to the newly developed parallel navigation, an **arbitrary scoring mesh geometry** can be defined which is independent to the volumes in the mass geometry.
Also, G4MultiFunctionalDetector and primitive scorer classes now offer the built-in scoring of most-common quantities

UI commands for scoring → no C++ required, apart from instantiating `G4ScoringManager` in `main()`

- Define a scoring mesh
  - /score/create/boxMesh <mesh_name>
  - /score/open, /score/close
- Define mesh parameters
  - /score/mesh/boxsize <dx> <dy> <dz>
  - /score/mesh/nbin <nx> <ny> <nz>
  - /score/mesh/translate,
- Define primitive scorers
  - /score/quantity/eDep <scorer_name>
  - /score/quantity/cellFlux <scorer_name>
  - currently **20 scorers** are available
- Define filters
  - /score/filter/particle <filter_name> <particle_list>
  - /score/filter/kinE <filter_name> <Emin> <Emax> <unit>
  - currently **5 filters** are available
- Output
  - /score/draw <mesh_name> <scorer_name>
  - /score/dump, /score/list

How to learn more about built-in scoring



Have a look at the **dedicated extended examples** released with Geant4:

examples/extended/runAndEvent/RE02
(use of primitive scorers)

examples/extended/runAndEvent/RE03
(use of UI-based scoring)