Interaction with the Geant4 Kernel-I

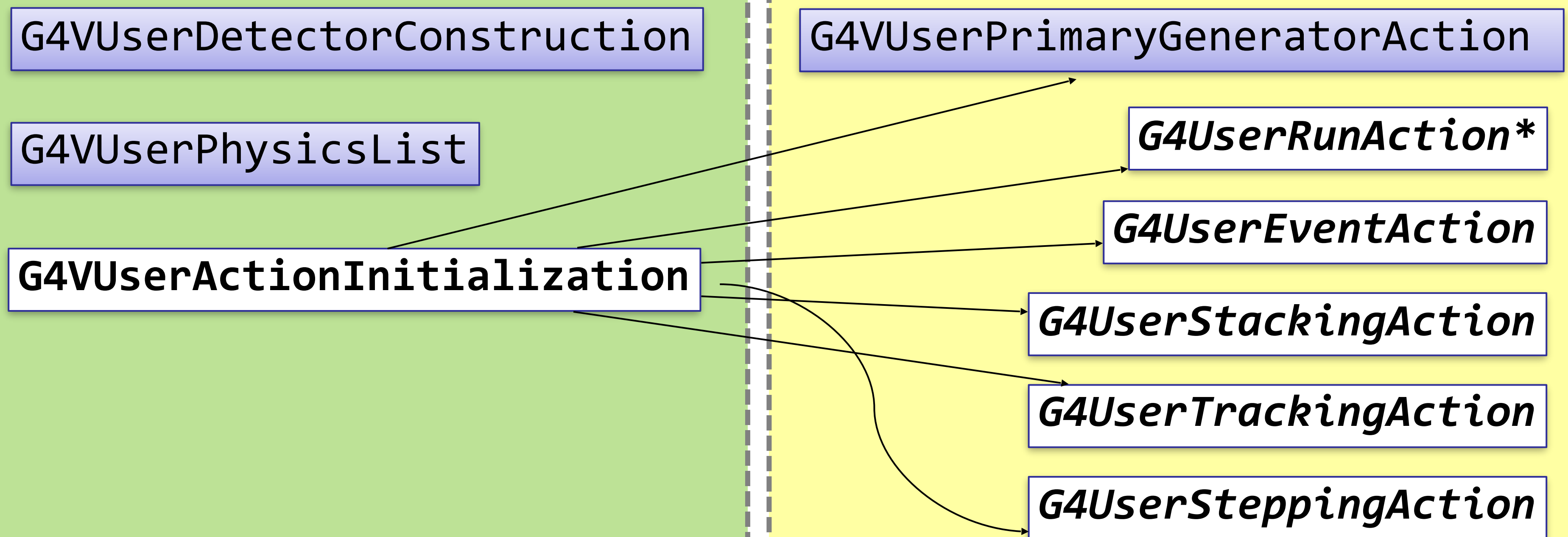
Giada Petringa (LNS-INFN)

The 2nd Geant4 School in China
Shandong University
Qingdao

...User classes (continued)

At initialization

At execution



Global: only one instance exists in memory, shared by all threads.

Local: an instance of each action class exists for each thread.

(*) Two RunAction's allowed: one for master and one for threads

Contents

- Run, Event, Track, ...
  - a word about multi-threading
- Optional user action classes
- Command-based scoring
- Accumulables
- Analysis tools

Part I: Run, Track, Event, ...

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)
  - **Worker** / **Master threads**

Run, Event and Tracks

The Event (G4Event)

- An Event is the basic unit of simulation
- At the beginning of event, primary tracks are generated and they are pushed into a stack
- Tracks are 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)

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
- The `G4(MT)RunManager` class manages the processing of each run, represented by:
  - **`G4Run`** class
  - **`G4UserRunAction`** for an optional user hook

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

G4Track status

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

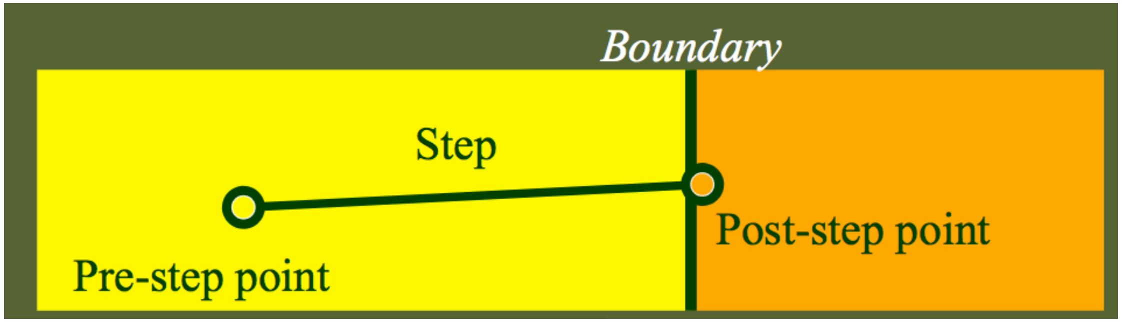
Track Status	Description
fAlive	The particle is continued to be tracked
fStopButAlive	Kin. Energy = 0, but AtRest process will occur
fStopAndKill	Track has lost identity (has reached world boundary, decayed, ...), Secondaries will be tracked
fKillTrackAndSecondaries	Track and its secondary tracks are 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)

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 (e.g. multiple scattering)
- You can extract information from a step after the step is completed, e.g.
  - in `ProcessHits()` method of your sensitive detector *(later)*
  - in `UserSteppingAction()` of your step action class *(later)*

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



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
  - Volume hiearchy information
- **`G4Step`** provides many Get... methods to access these information or objects
  - **`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

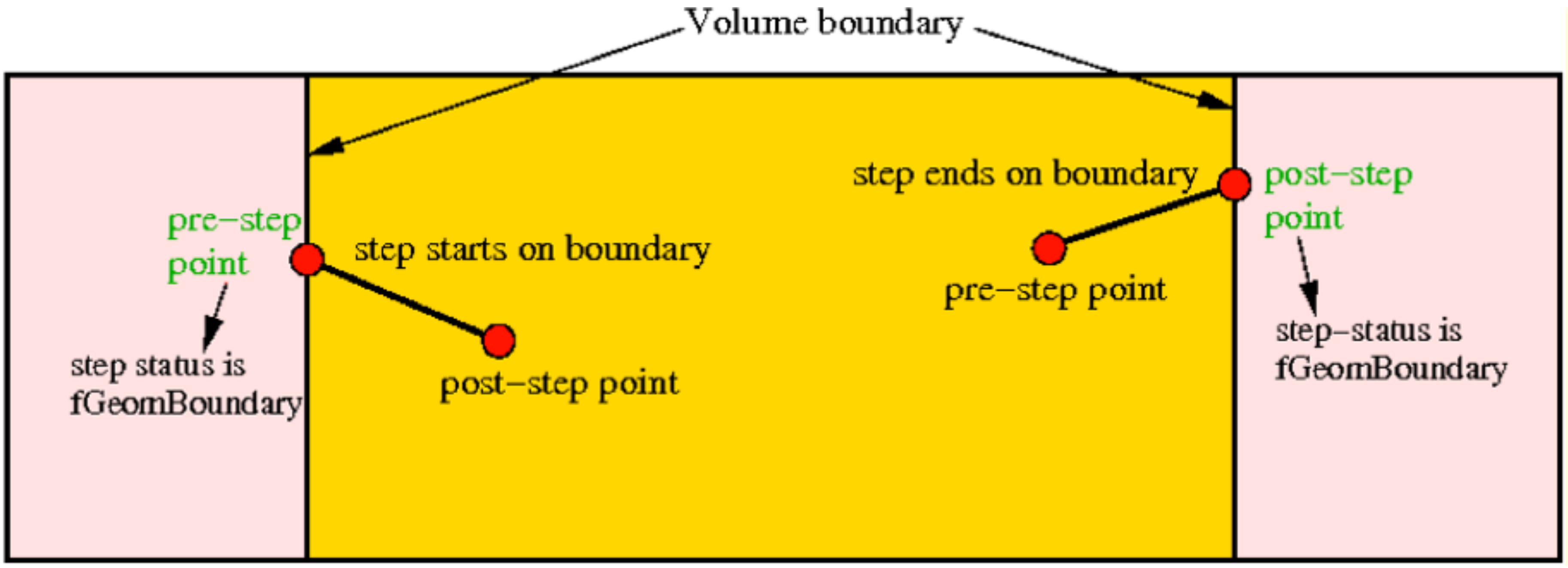
- One can also use the **step status**
  - Step Status provides information about the process that restricted the step length

  - 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



Example: boundaries

```
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();
```

Geant4 terminology: an overview

- **Run:** is a collection of events with the same detector and physics conditions;
- **Event:** is a collection of primary and secondary particles in a stack
- **Track:** is a **snapshot** of a particle
- **Step:** represents a step in the particle propagation

- **Processes: …**
- **Cut: …**
- **Worker** / **Master threads: …**

Part II: Optional user action classes

Optional user action classes

- Five **base classes** with **virtual methods** the user may override to step during the execution of the application
  - `G4User`**`Run`**`Action`
  - `G4User`**`Event`**`Action`
  - `G4User`**`Tracking`**`Action`
  - `G4User`**`Stacking`**`Action`
  - `G4User`**`Stepping`**`Action`
- Default implementation (**not** purely virtual): **Do nothing** ☺️
- Therefore, **override** only the methods you need.

G4UserRunAction

This class has three virtual methods which are invoked by G4RunManager for each run:

GenerateRun() ==> G4Run* GenerateRun()
This method is invoked at the beginning of BeamOn. Because the user can inherit the class G4Run and create his/her own concrete class to store some information about the run, the GenerateRun() method is the place to instantiate such an object

BeginOfRunAction() ==> void BeginOfRunAction(const G4Run*)
This method is invoked before entering the event loop. This method is invoked after the calculation of the physics tables.

EndOfRunAction() ==> void EndOfRunAction(const G4Run*)
This method is invoked at the very end of the run processing. It is typically used for a simple analysis of the processed run.

G4UserEventAction

This class has two virtual methods which are invoked by G4EventManager for each event:

beginOfEventAction() ==> void BeginOfEventAction(const G4Event*)
This method is invoked before converting the primary particles to G4Track objects. A typical use of this method would be to initialize and/or book histograms for a particular event.

endOfEventAction() ==> void EndOfEventAction(const G4Event*)
This method is invoked at the very end of event processing. It is typically used for a simple analysis of the processed event.

G4UserStackingAction

This class has three virtual methods, **ClassifyNewTrack**, **NewStage** and **PrepareNewEvent** which the user may override in order to control the various track stacking mechanisms.

ClassifyNewTrack() ==>
G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)

is invoked by G4StackManager whenever a new G4Track object is "pushed" onto a stack by G4EventManager.

G4ClassificationOfNewTrack has four possible values:
fUrgent - track is placed in the *urgent* stack
fWaiting - track is placed in the *waiting* stack, and will not be simulated until the *urgent* stack is empty
fPostpone - track is postponed to the next event
fKill - the track is deleted immediately and not stored in any stack.

These assignments may be made based on the origin of the track which is obtained as follows:
G4int parent_ID = aTrack->get_parentID();
where
parent_ID = 0 indicates a primary particle
parent_ID > 0 indicates a secondary particle
parent_ID < 0 indicates postponed particle from previous event.

G4UserStackingAction

NewStage() ==> void NewStage()
is invoked when the *urgent* stack is empty and the *waiting* stack contains at least one G4Track object.

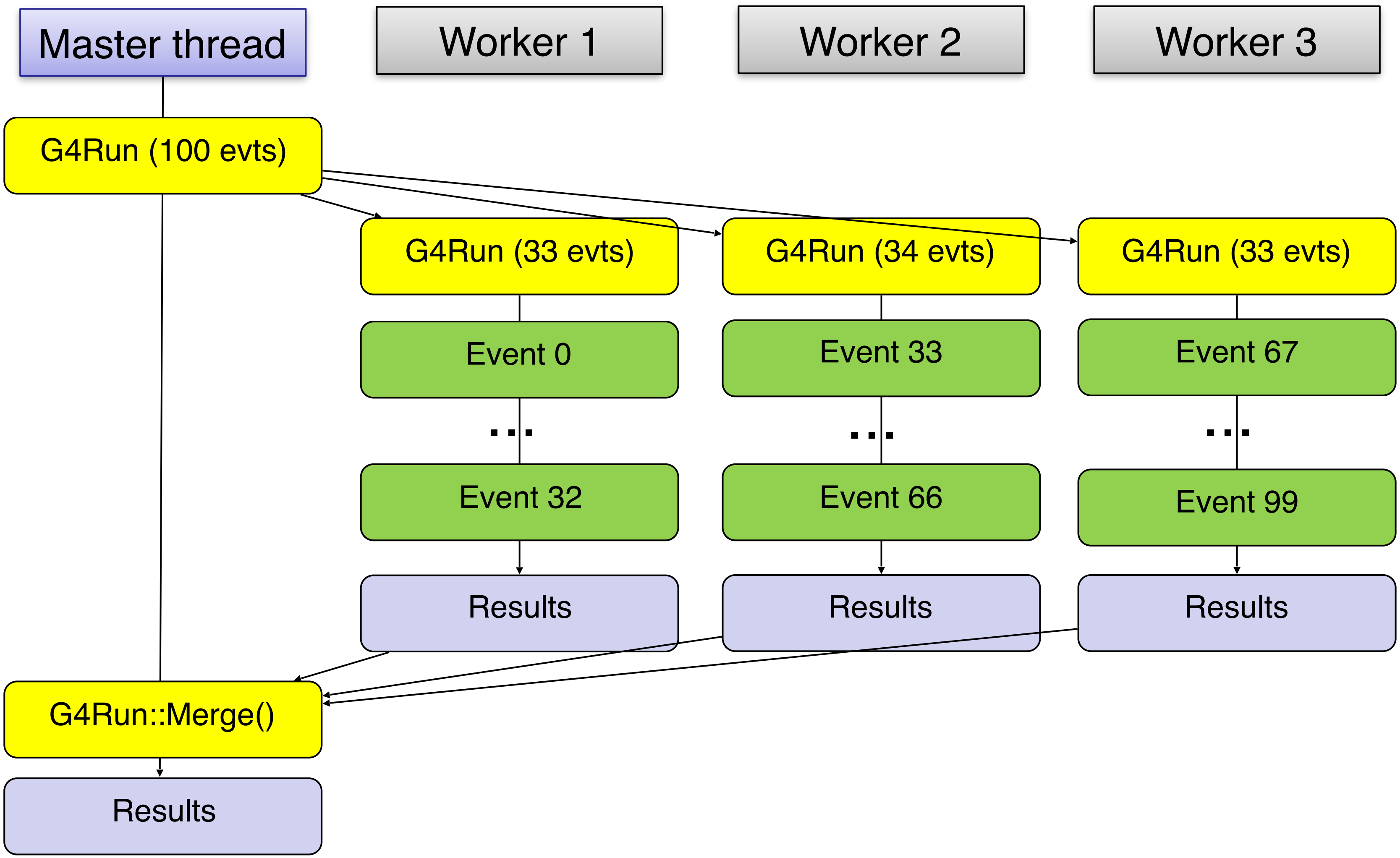
PrepareNewEvent() ==> void PrepareNewEvent()
is invoked at the beginning of each event. At this point no primary particles have been converted to tracks, so the *urgent* and *waiting* stacks are empty.

G4UserSteppingAction

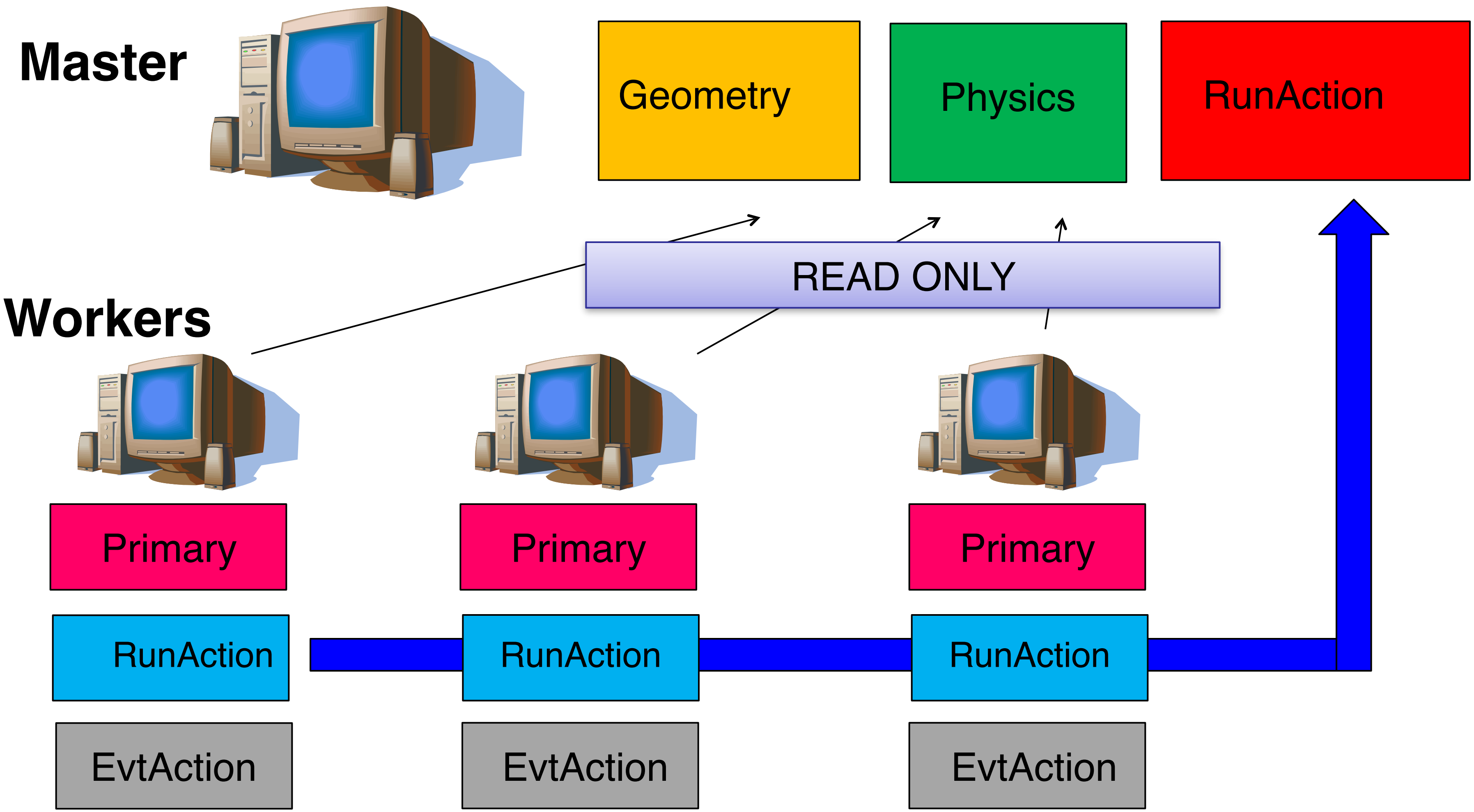
UserSteppingAction() ==> void UserSteppingAction(const g4Step*)

Get information about particles;
kill tracks under specific circumstances

Multi-threaded processing of events



User actions in multi-threaded run



Part III: Command-based scoring

Command-based scoring

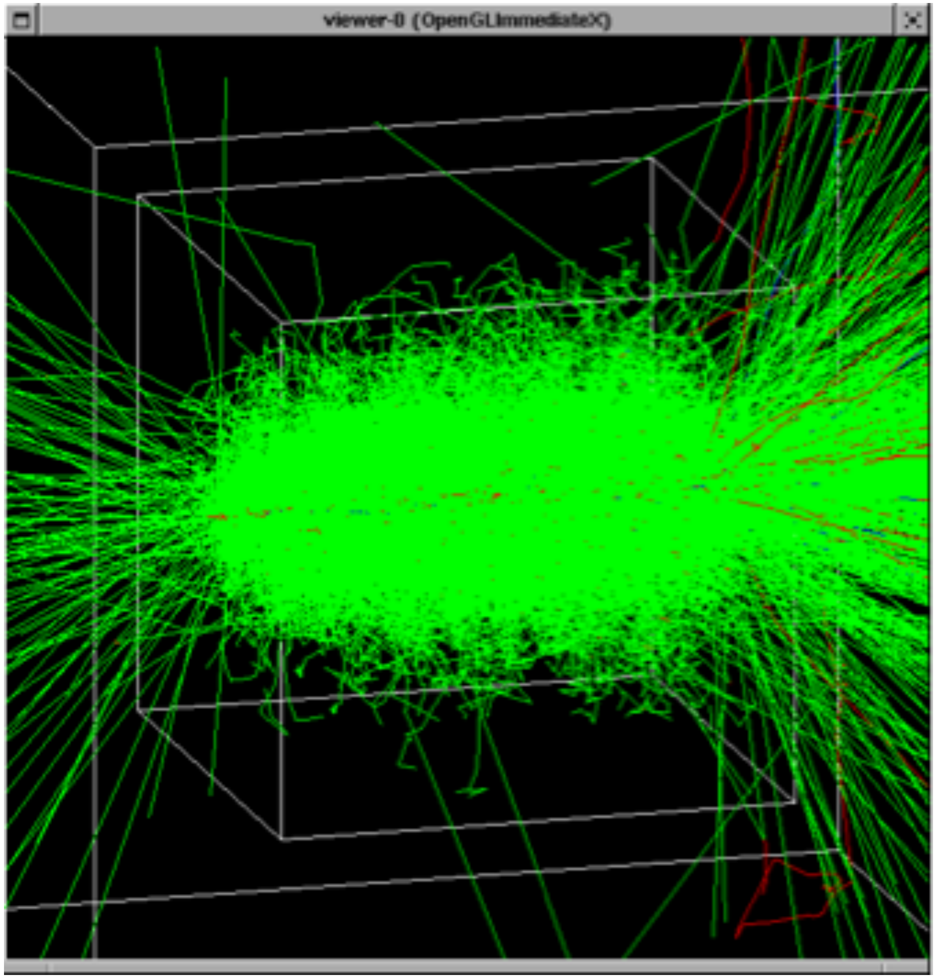
UI commands for scoring → no C++ required, apart from accessing `G4ScoringManager`

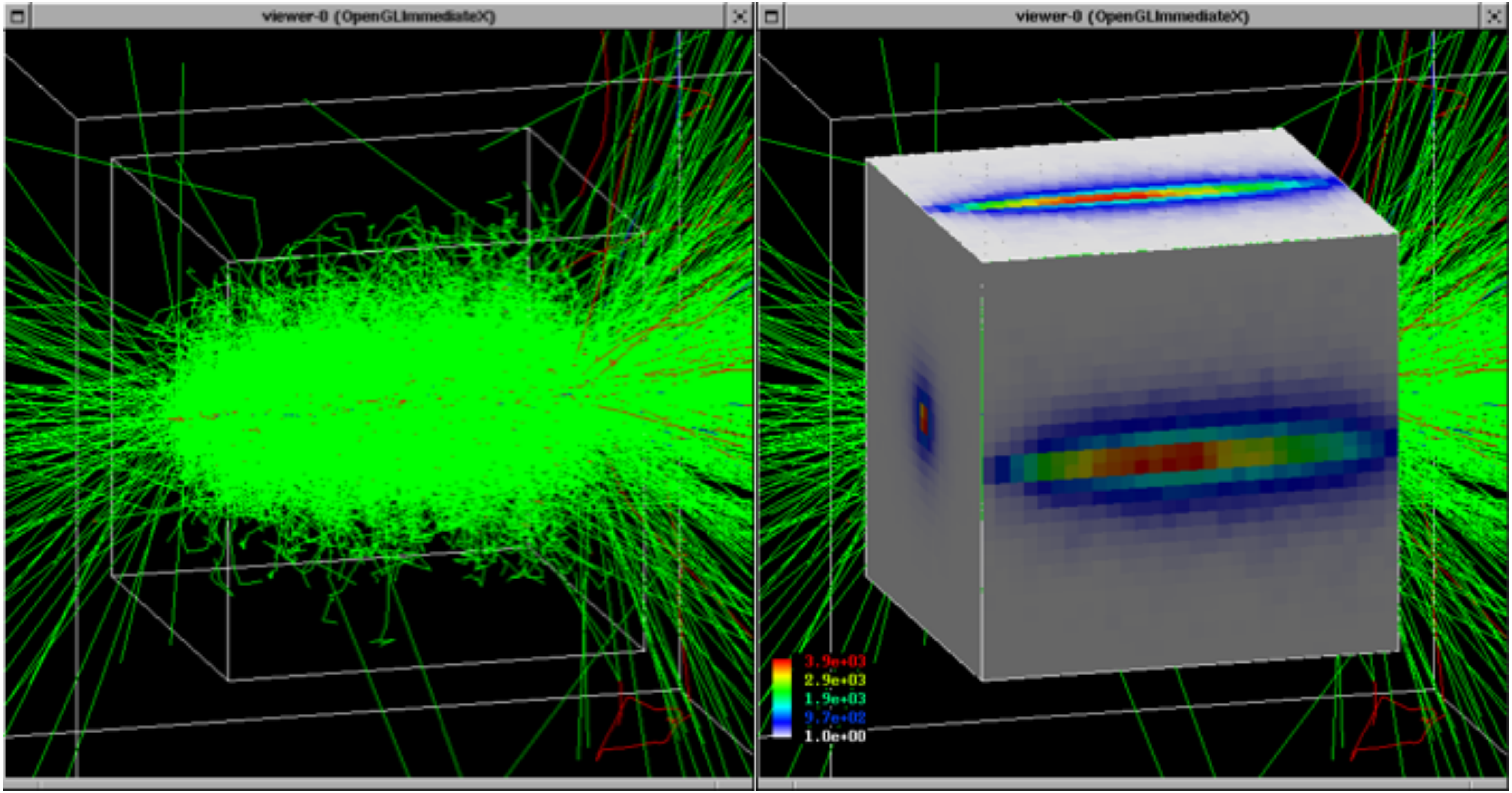
```
int main() {
    ...
    G4ScoringManager::GetScoringManager();
    ...
}
```

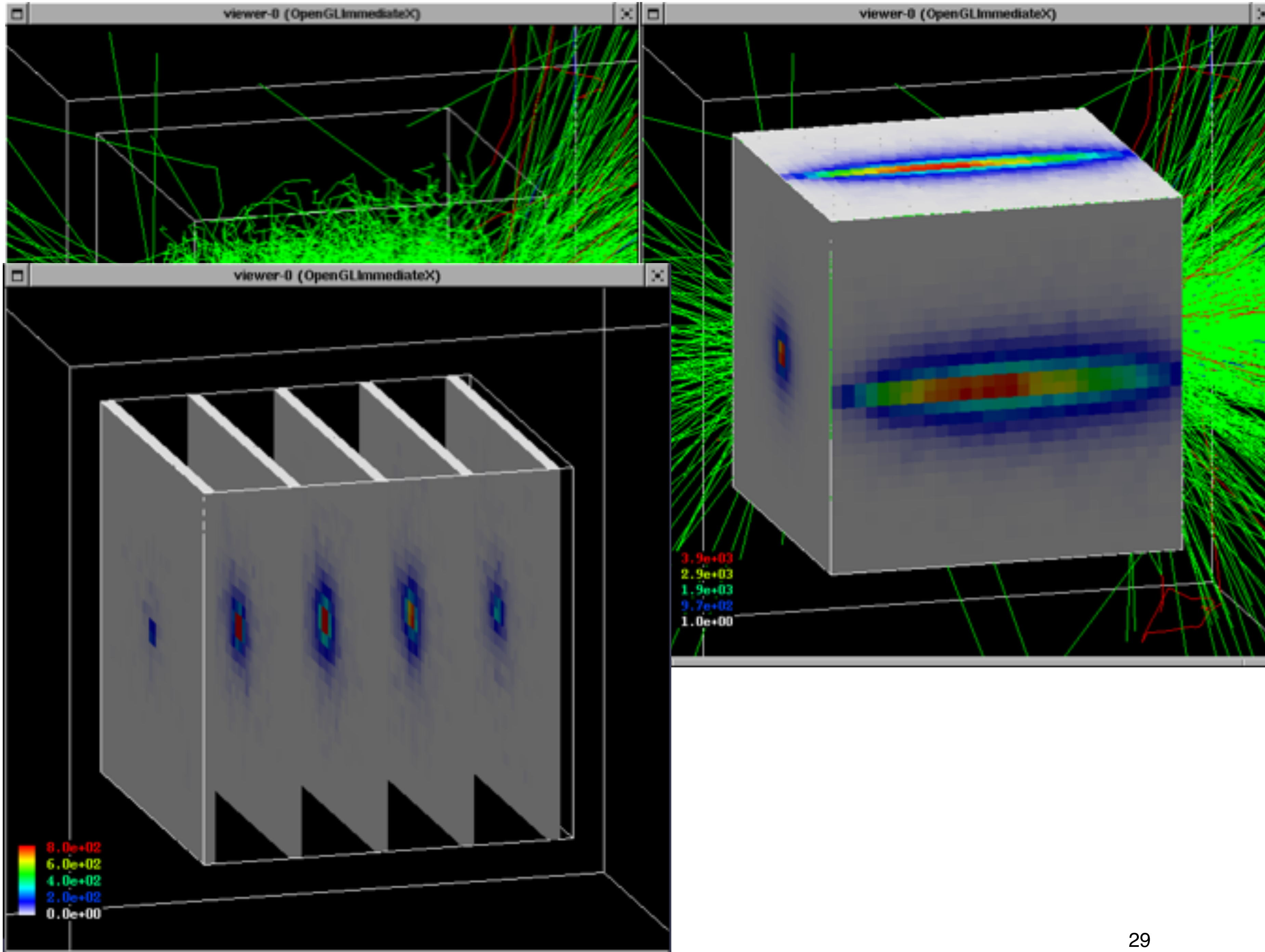
- Define a scoring mesh
  /score/create/boxMesh <mesh_name>
  /score/open, /score/close
- Define mesh parameters
  /score/mesh/boxsize <dx> <dy> <dz>
  /score/mesh/nbin <nx> <ny> <nz>
  /score/mesh/translate,
- Define primitive scorers
  /score/quantity/eDep <scorer_name>
  /score/quantity/cellFlux <scorer_name>
  currently **20 scorers** are available

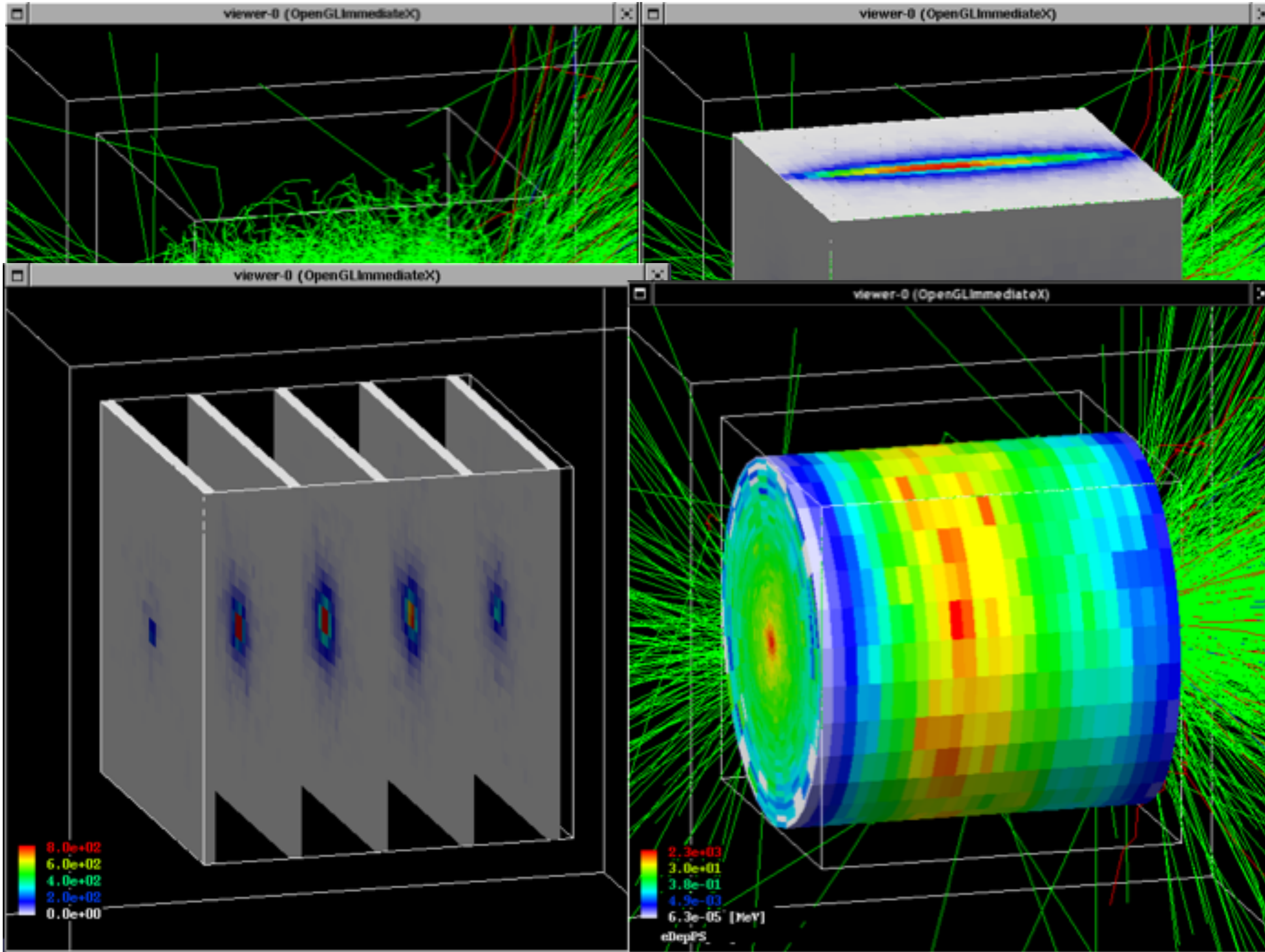
- Define filters
  /score/filter/particle <filter_name> <particle_list>
  /score/filter/kinE <filter_name> <Emin> <Emax> <unit>
  currently **5 filters** are available
- Output
  /score/draw <mesh_name> <scorer_name>
  /score/dump, /score/list

https://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/AllResources/Control/UIcommands/_score_.html


viewer-0 (OpenGLImmediateX)


viewer-0 (OpenGLImmediateX)
viewer-0 (OpenGLImmediateX)
3.9e+03
2.9e+03
1.9e+03
9.7e+02
1.0e+00


viewer-0 (OpenGLImmediateX)
viewer-0 (OpenGLImmediateX)
viewer-0 (OpenGLImmediateX)
3.9e+03
2.9e+03
1.9e+03
9.7e+02
1.0e+00
8.0e+02
6.0e+02
4.0e+02
2.0e+02
0.0e+00


viewer-0 (OpenGLImmediateX)
viewer-0 (OpenGLImmediateX)
viewer-0 (OpenGLImmediateX)
viewer-0 (OpenGLImmediateX)
8.0e+02
6.0e+02
4.0e+02
2.0e+02
0.0e+00
2.3e+03
3.0e+01
3.8e-01
4.9e-03
6.3e-05 [MeV]
eDepPS

G4Accumulable<T>

- Templated class can be used to facilitate merging of the values accumulated on workers to the master thread
  - Accumulable during Run
  - Value merge at the end (explicit)
  - Scalar variables only (otherwise, expert)
- Alternative to ntuples/histograms *(later)*
- Managed by `G4AccumulableManager`

<=10.2: Previously named **G4Parameter**!

Detached session: g4analysis tools

Geant4 analysis classes

- A basic analysis interface is available in Geant4 for histograms (1D and 2D) and ntuples

- Unified interface to support different output formats
  - ROOT, CSV, AIDA XML, and HBOOK
  - Code is the same, just change one line to switch from one to an other

- Everything is done using `G4AnalysisManager`
  - **UI commands** available

g4analysis

- Selection of output format is performed by including a proper header file:

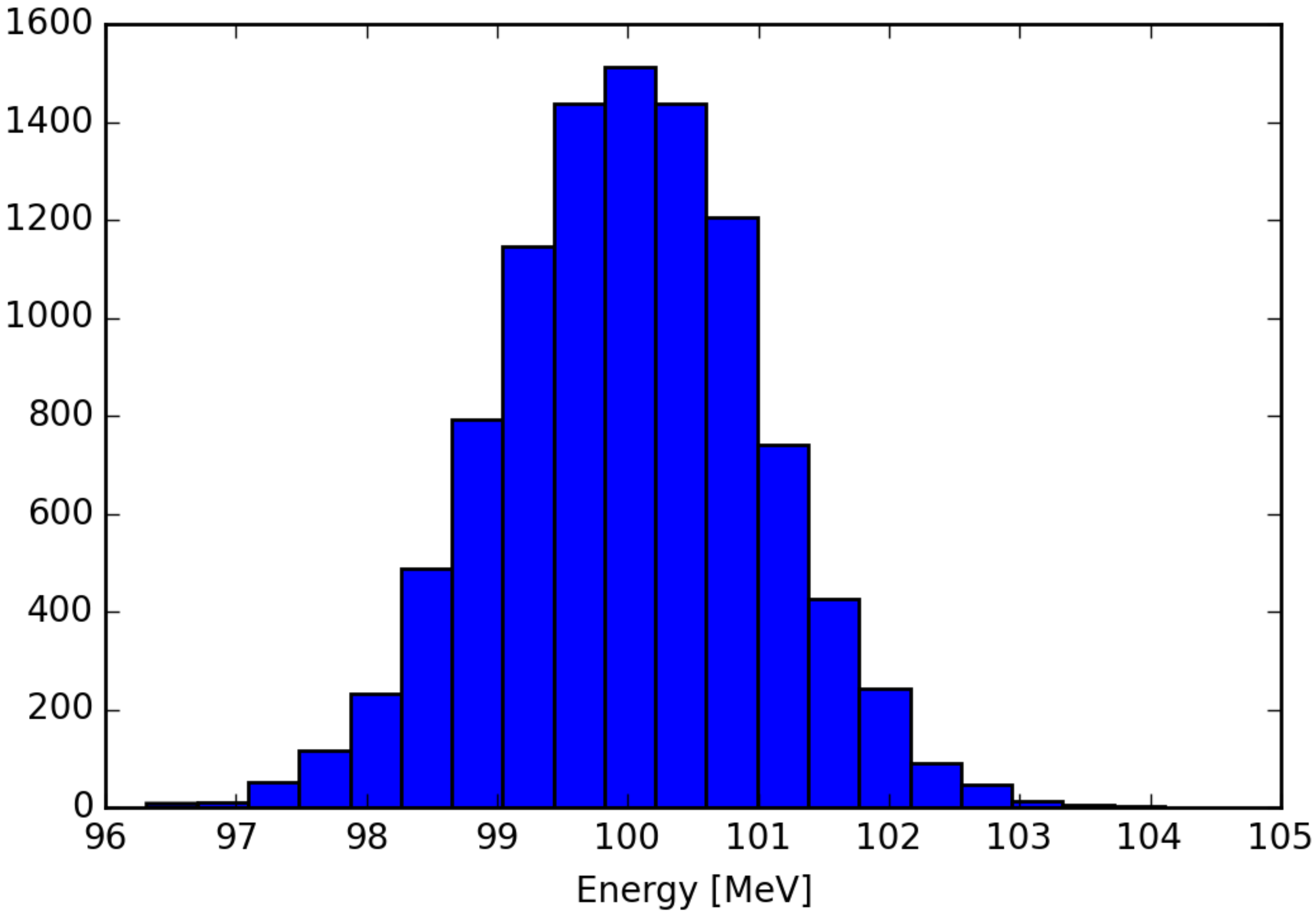
```
#ifndef MyAnalysis_h
#define MyAnalysis_h 1

#include "g4root.hh"
//#include "g4xml.hh"
//#include "g4csv.hh"  // can be used only with ntuples

#endif
```

Advanced topic: It is possible to use more formats at the same time. See documentation.

Histograms



1600
1400
1200
1000
800
600
400
200
0
96
97
98
99
100
101
102
103
104
105
Energy [MeV]

Open file and book histograms

```
#include "MyAnalysis.hh"

void MyRunAction::BeginOfRunAction(const G4Run* run)
{
  // Get analysis manager
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->SetVerboseLevel(1);
  man->SetFirstHistoId(1);

  // Creating histograms
  man->CreateH1("h", "Title", 100, 0., 800*MeV);
  man->CreateH1("hh", "Title", 100, 0., 10*MeV);

 // Open an output file
  man->OpenFile("myoutput");
}
```

Start numbering of histograms from ID=1

ID=1

ID=2

Open output file

Fill histograms and write the file

```
#include "MyAnalysis.hh"

void MyEventAction::EndOfEventAction(const G4Run* aRun)
{
  auto man = G4AnalysisManager::Instance();
  man->FillH1(1, fEnergyAbs);
  man->FillH1(2, fEnergyGap);
}

MyRunAction::~MyRunAction()
{
  auto man = G4AnalysisManager::Instance();
  man->Write();
}

int main()
{
  ...
  auto man = G4AnalysisManager::Instance();
  man->CloseFile();
}
```

ID=1

ID=2

Ntuples

ParticleID	Energy	x	y
0	99.5161753	-0.739157031	-0.014213165
1	98.0020355	1.852812521	1.128640204
2	100.0734469	0.863203688	-0.277949199
3	99.3508677	-2.063452685	-0.898594988
4	101.2505954	1.030581054	0.736468229
5	98.9849841	-1.464509417	-1.065372115
6	101.1547644	1.121931704	-0.203319254
7	100.8876748	0.012068917	-1.283410959
8	100.3013861	1.852532119	-0.520615895
9	100.6295882	1.084122362	0.556967258
10	100.4887681	-1.021971662	1.317380892
11	101.6716567	0.614222096	-0.483530242
12	99.1083093	-0.776034456	0.203524549
13	97.3595776	0.814378204	-0.690615126
14	100.7264612	-0.408732803	-1.278746667

Ntuples support

- **g4tools** support ntuples
  - **any** number of ntuples
  - **any** number of columns
  - supported types: **int/float/double**

- For more complex tasks (other functionality of ROOT TTrees) have to link **ROOT** directly

Book ntuples

```
#include "MyAnalysis.hh"

void MyRunAction::BeginOfRunAction(const G4Run* run)
{
  // Get analysis manager
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man-> SetFirstNtupleId(1);

  // Creating ntuples
  man->CreateNtuple("name", "Title");
  man->CreateNtupleDColumn("Eabs");
  man->CreateNtupleDColumn("Egap");
  man->FinishNtuple();

  man->CreateNtuple("name2","title2")
  man->CreateNtupleIColumn("ID");
  man->FinishNtuple();
}
```

Start numbering of ntuples from ID=1

ID=1

ID=2

Fill ntuples

- File handling and general clean-up as shown for histograms

```
#include "MyAnalysis.hh"

void MyEventAction::EndOfEventAction(const G4Run* aRun)
{
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->FillNtupleDColumn(1, 0, fEnergyAbs);
  man->FillNtupleDColumn(1, 1, fEnergyGap);
  man->AddNtupleRow(1);

  man->FillNtupleIColumn(2, 0, fID);
  man->AddNtupleRow(2);

}
```

ID=1,
columns 0, 1

ID=2,
column 0

Conclusion

- Concepts of run, event, step, track, particle
- User action classes
- Data output – g4tools

Task 4

- Task 4a User actions
- Task 4b Command-based scoring
- Task 4c Geant4 native scoring (multi-functional detectors)

Exercise 4a.1: Kill a particle

Exercise 4a.2: Calculate total track length

Exercise 4b.2: Create a scoring mesh

Exercise 4b.1: Enable the scoring manager

Exercise 4b.3: Visualize the mesh

Exercise 4b.4: Dump results