XV Seminar on Software for Nuclear, Subnuclear and Applied Physiscs

Hotel Porto Conte, Alghero
27 May - 1 June 2018

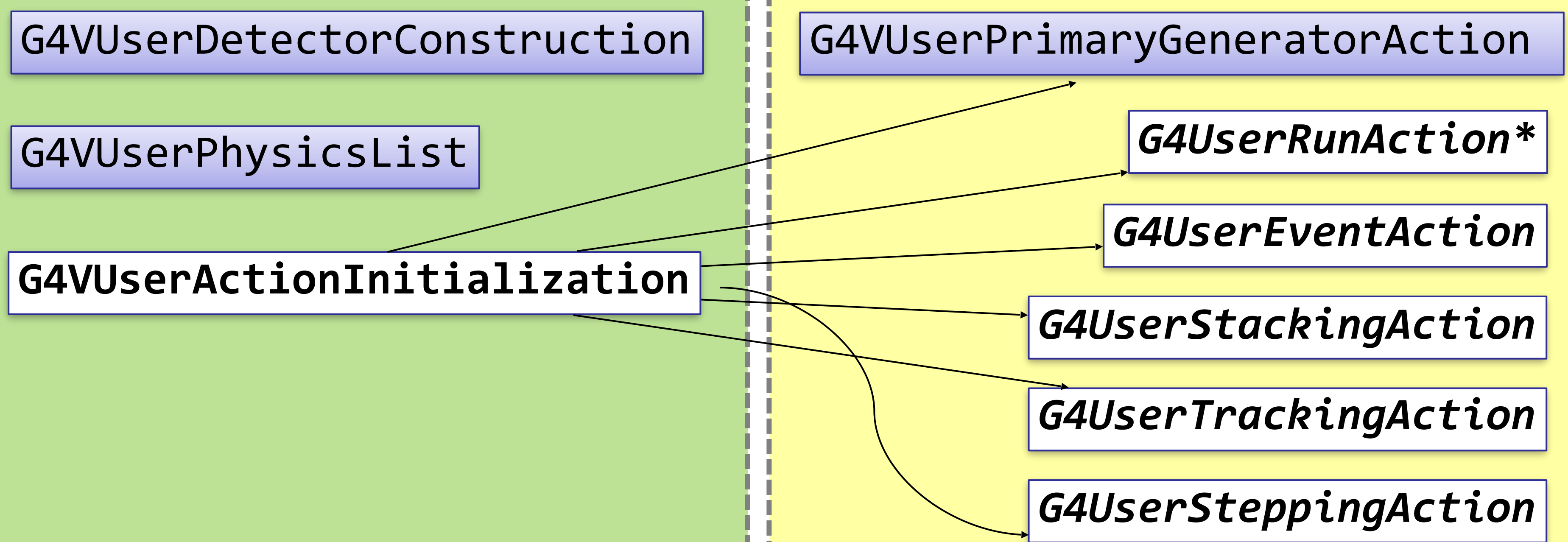
Interaction with the Geant4 kernel I

Giada Petringa
(LNS-INFN)

...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
- 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
- At the beginning of a run, geometry is optimised for navigation and cross section tables are (re)calculated
- 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)

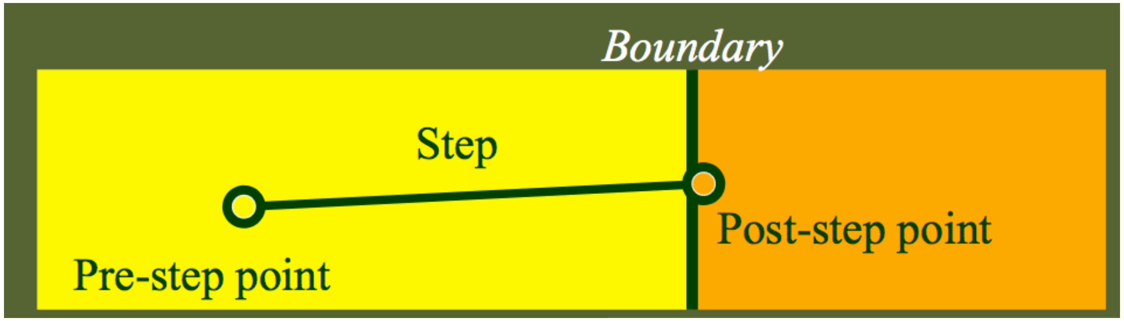
Track Status	Description
fAlive	The particle is continued to be tracked
fStopButAlive	Kin. Energy = 0, but AtRest process will occur
fStopAndKill	Track has lost identity (has reached world boundary, decayed, ...), Secondaries will be tracked
fKillTrackAndSecondaries	Track and its secondary tracks are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

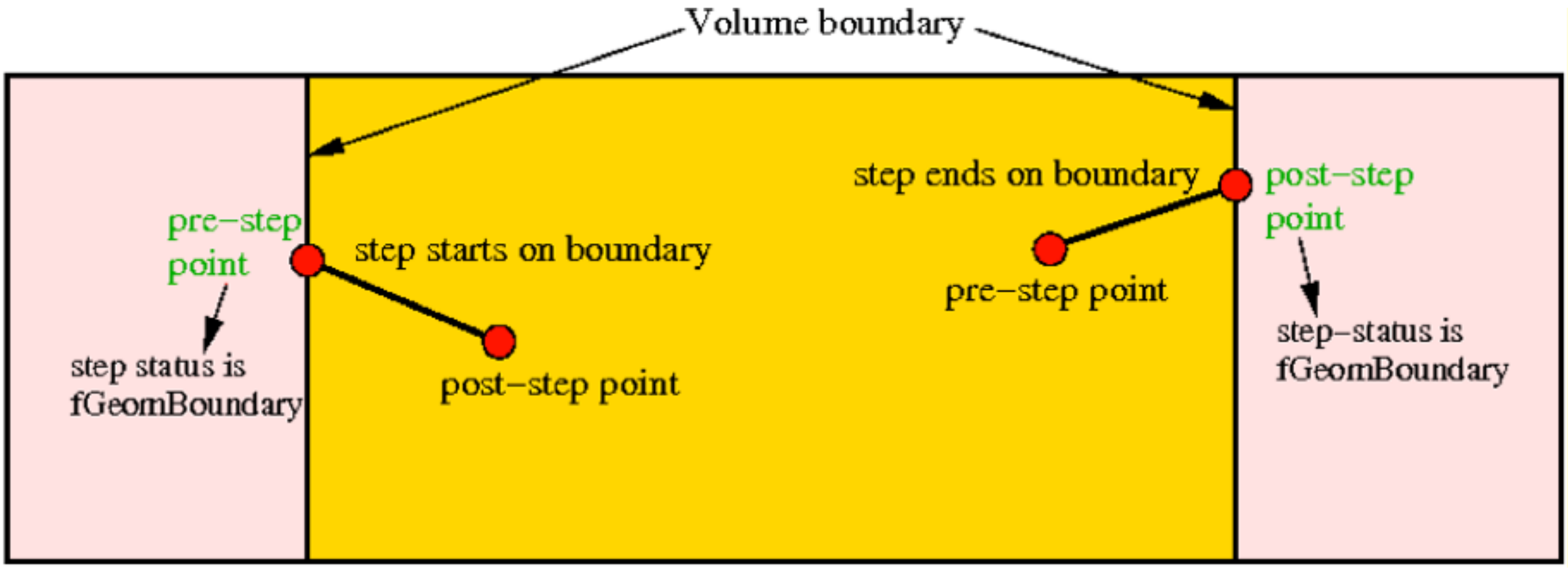
- One can also use the **step status**
  - Step Status provides information about the process that restricted the step length

  - 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();
```

Part II: Optional user action classes

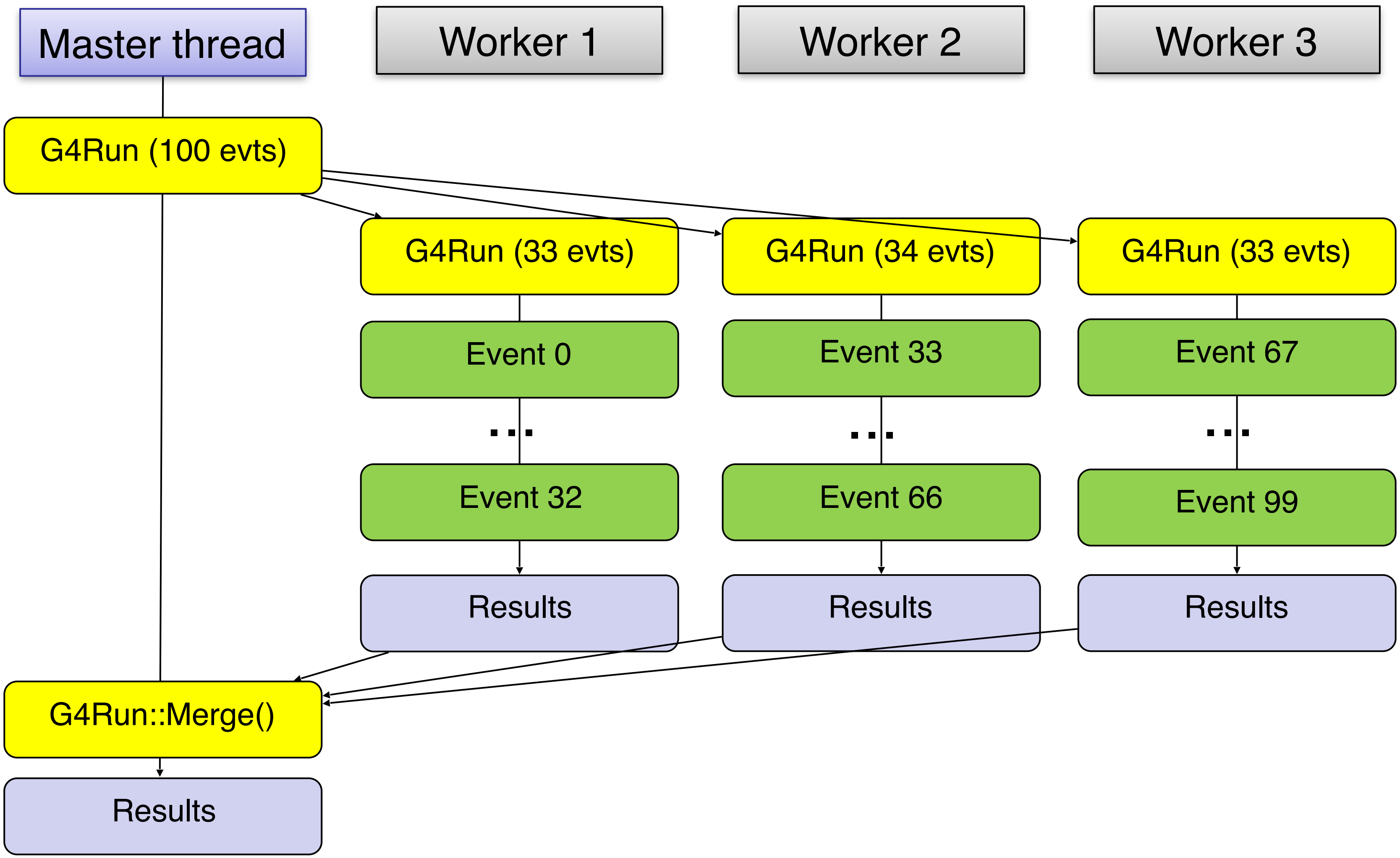
Optional user action classes

- Five **base classes** with **virtual methods** the user may override to step during the execution of the application

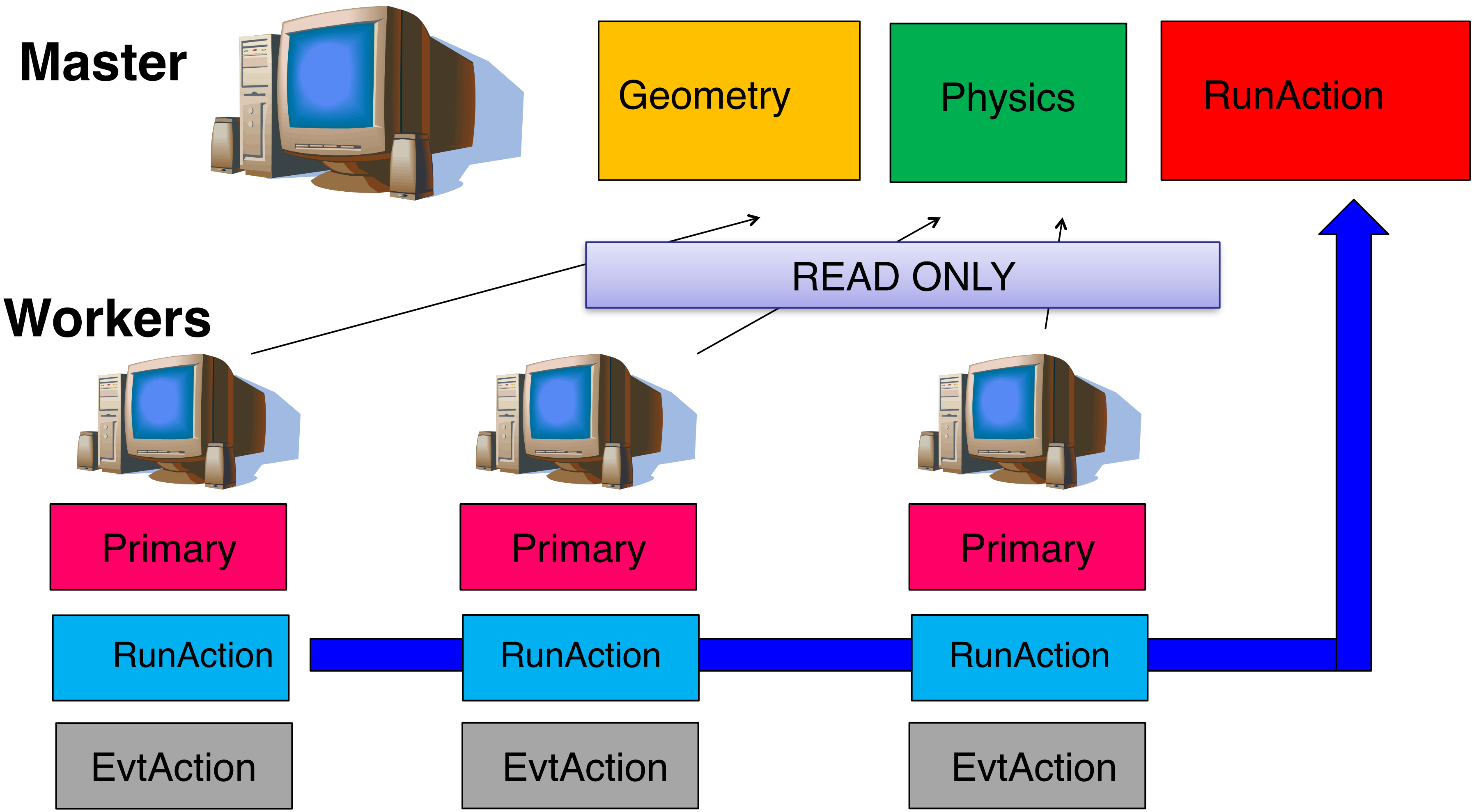
  - `G4UserRunAction`
  - `G4UserEventAction`
  - `G4UserTrackingAction`
  - `G4UserStackingAction`
  - `G4UserSteppingAction`

- Default implementation (**not** purely virtual): **Do nothing** ☺️
- Therefore, **override** only the methods you need.

Multi-threaded processing of events



User actions in multi-threaded run



G4UserRunAction

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

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()
This method is invoked before entering the event loop. This method is invoked after the calculation of the physics tables.

EndOfRunAction()
This method is invoked at the very end of the run processing. It is typically used for a simple analysis of the processed run.

G4UserRunAction

```
G4Run* GenerateRun()
```

This method should be used to instantiate a user-specific run class object

```
void BeginOfRunAction(const G4Run*)
```

Likely uses of this method include: setting a run identification number – histograms –run conditions

```
void EndOfRunAction(const G4Run*)
```

G4UserEventAction

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

beginOfEventAction()
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()
This method is invoked at the very end of event processing. It is typically used for a simple analysis of the processed event.

G4UserEventAction

```
void BeginOfEventAction(const G4Event*)
```

This method is invoked before converting the primary particles to G4Track objects.

```
void EndOfEventAction(const G4Event*)
```

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

This value should be determined by the user.

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.

G4UserEventAction

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

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.

G4UserStackingAction

```
G4ClassificationOfNewTrack ClassifyNewTrack(const
G4Track*)
```

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

```
void NewStage()
void PrepareNewEvent()
```

Uses:

- Pre-selection of tracks (~manual cuts)
- Optimization of the order of track execution

G4UserSteppingAction

```
void UserSteppingAction(const G4Step*)
```

Uses:

- Get information about particles
- Kill tracks under specific circumstances

Part III: Command-based scoring

Command-based scoring

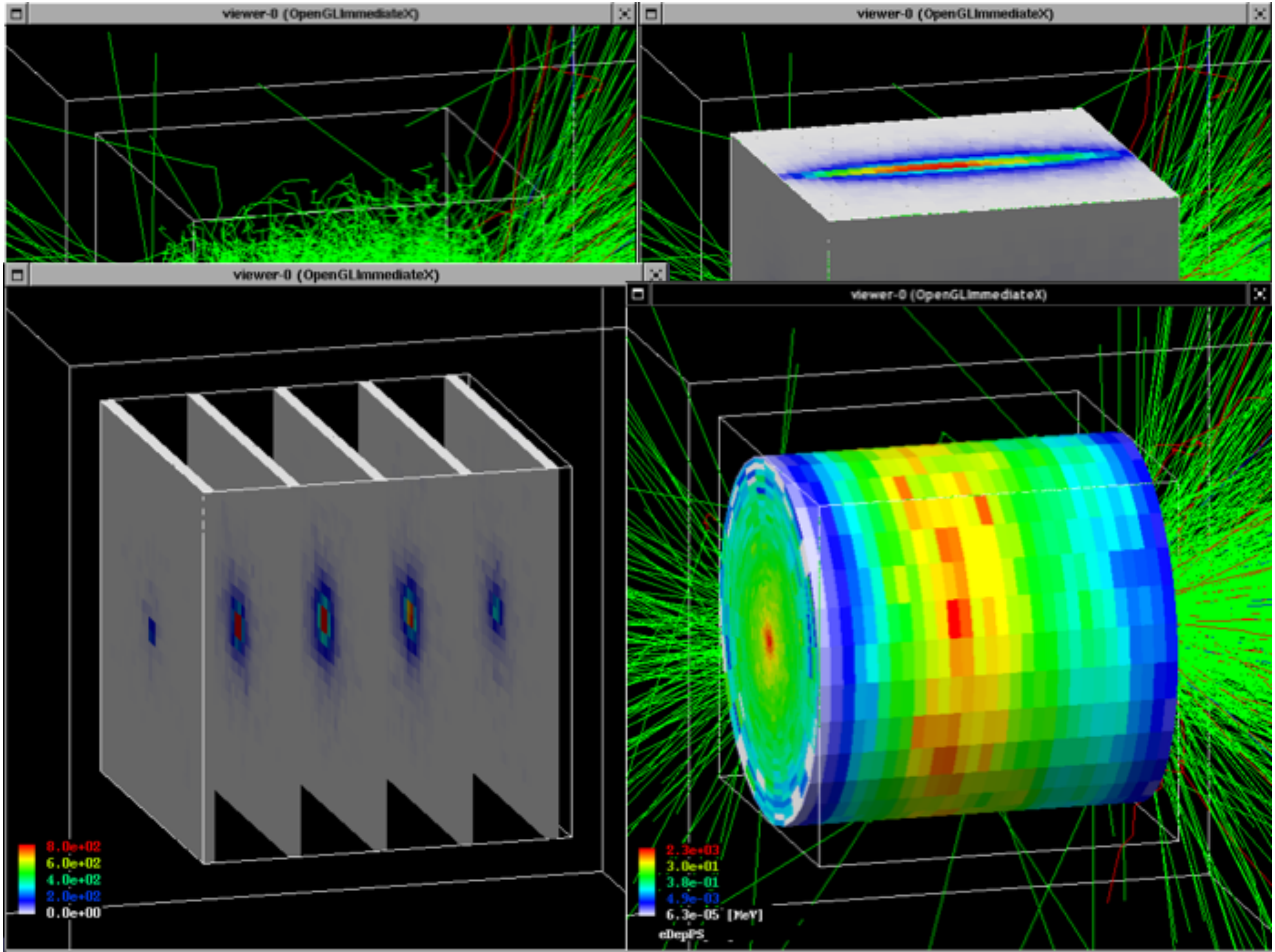
UI commands for scoring → no C++ required, apart from accessing `G4ScoringManager`

```
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)
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

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:

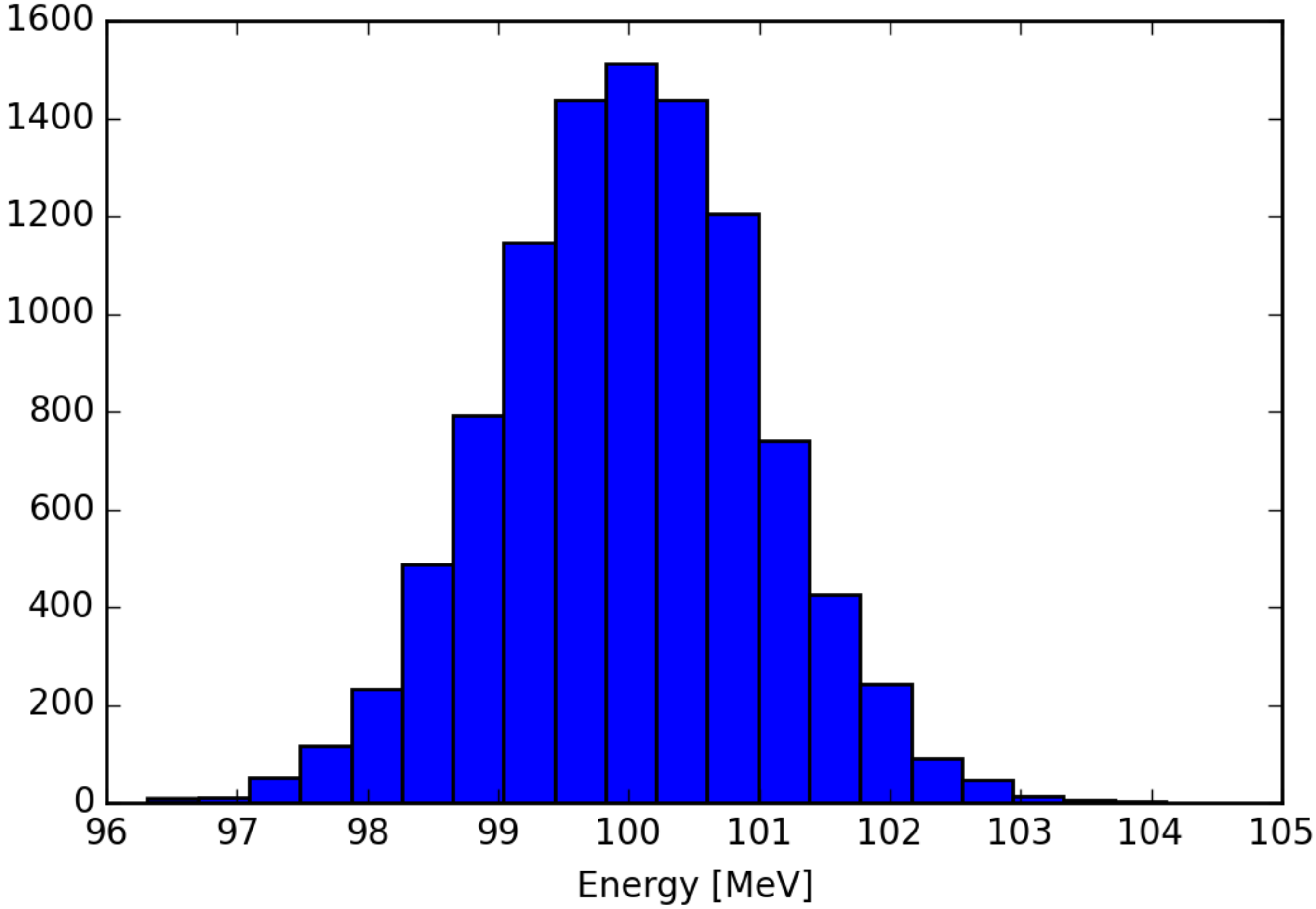
```
#ifndef MyAnalysis_h
#define MyAnalysis_h 1

#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

Okay that's all.

More slides (back-up)...

Example: custom messengers

```
#include <G4UImessenger.hh>
#include <G4UIcmdWithoutParameter.hh>
#include <G4UIdirectory.hh>

class HiMessenger : public G4UImessenger
{
public:
    HiMessenger() {
        _directory = new G4UIdirectory("/hi/");
        _command = new G4UIcmdWithoutParameter("/hi/sayIt", this);
    }

    void SetNewValue(G4UIcommand* command, G4String newValue) {
        if (command == _command) {
            G4cout << "Hi there :-)" << G4endl;
        }
    }
private:
    G4UIdirectory* _directory;
    G4UIcmdWithoutParameter* _command;
};
```

Example: output to a text file

MT

```
#include <fstream>

class SteppingAction
{
    // ...
    std::ofstream fout;
};

SteppingAction::SteppingAction() : fout("outfile.txt") { } // ...

void SteppingAction::UserSteppingAction(const G4Step* aStep)
{
    G4Track* theTrack = aStep->GetTrack();
    G4double edep = aStep->GetTotalEnergyDeposit();
    G4double kineticEnergy = theTrack->GetKineticEnergy();

    // The output
    fout
      << "Energy deposited--->" <<  " " <<  edep << " "
      << "Kinetic Energy --->" << "  " << kineticEnergy << " " << G4endl;
}
```

And even more slides...

Histograms API (1)

- Support linear and log scales and irregular bins
- **`CreateH2()`** for 2D histograms

```
G4int CreateH1(const G4String& name, const G4String& title,
               G4int nbins, G4double xmin, G4double xmax,
               const G4String& unitName = "none",
               const G4String& fcnName = "none",
               const G4String& binSchemeName = "linear");
```

```
G4int CreateH1(const G4String& name, const G4String& title,
               const std::vector<G4double>& edges,
               const G4String& unitName = "none",
               const G4String& fcnName = "none");
```

Histograms API (2)

- You can change parameters of an existing histogram
- You can fill with a weight
- Methods to scale, retrieve, get rms and mean

```
G4bool SetH1Title(G4int id, const G4String& title);
G4bool SetH1XAxisTitle(G4int id, const G4String& title);
G4bool SetH1YAxisTitle(G4int id, const G4String& title);

G4bool FillH1(G4int id, G4double value, G4double weight = 1.0);

G4bool ScaleH1(G4int id, G4double factor);

G4int GetH1Id(const G4String& name, G4bool warn = true) const;
```

Introduction: data analysis with Geant4

- For a long time, Geant4 did not attempt to provide/ support **any data analysis** tools
  - The focus was given (and is given) to the central mission as a Monte Carlo simulation toolkit
  - As a general rule, the user is expected to provide her/his own code to output results to an appropriate analysis format
- Basic classes for data analysis have recently been implemented in Geant4 (g4analysis)
  - Support for histograms and ntuples
  - Output in **ROOT**, **XML**, **HBOOK** and **CSV** (ASCII)
  - Appropriate only for easy/quick analysis: for advanced tasks, the users must write their own code and to use an external analysis tool

Introduction: how to write simulation results

- Formatted (= human-readable) **ASCII files**
  - Simplest possible approach is comma-separated values (.csv) files
  - The resulting files can be opened and analyzed by tools such as: Matlab, Python, Excel, ROOT, Gnuplot, OpenOffice, Origin, PAW, …
- **Binary files** with complex analysis objects (Ntuples)
  - Allows to control what plot you want with modular choice of conditions and variables
    - Ex: energy of electrons knowing that (= cuts): (1) position/location, (2) angular window, (3) primary/secondary …
  - Tools: Root , PAW, AIDA-compliant (PI, JAS3 and OpenScientist)

Output stream (G4cout)

- `G4cout` is a `iostream` object defined by Geant4.
  - Used in the same way as standard `std::cout`
  - Output streams handled by `G4UImanager`
  - `G4endl` is the equivalent of `std::endl` to end a line
- Output strings may be displayed in another window (Qt GUI) or redirected to a file

- You can also use the file streams (`std::ofstream`) provided by the C++ libraries

MT

Example: **Output on screen**

```
void SteppingAction::UserSteppingAction(const G4Step* aStep)
{
    // Collect data
    G4Track* theTrack = aStep->GetTrack();
    G4DynamicParticle* particle = theTrack->GetDynamicParticle();
    G4ParticleDefinition* parDef = particle->GetDefinition();

    G4double edep = aStep->GetTotalEnergyDeposit();
    G4double particleCharge = particle->GetCharge();
    G4double kineticEnergy = theTrack->GetKineticEnergy();

    // The output
    G4cout
      << "Energy deposited--->" <<  " " <<  edep << " "
      << "Charge--->" <<  " " << particleCharge << " "
      << "Kinetic Energy --->" << "  " << kineticEnergy << " " << G4endl;
}
```

Output on screen – an example

```
Begin of Event: 0

Energy deposited---> 9.85941e-22 Charge---> 6 Kinetic energy---> 160
Energy deposited---> 8.36876     Charge---> 6 Kinetic energy---> 151.631
Energy deposited---> 8.63368     Charge---> 6 Kinetic energy---> 142.998
Energy deposited---> 5.98509     Charge---> 6 Kinetic energy---> 137.012
Energy deposited---> 4.73055     Charge---> 6 Kinetic energy---> 132.282
Energy deposited---> 0.0225575   Charge---> 6 Kinetic energy---> 132.254
Energy deposited---> 1.47468     Charge---> 6 Kinetic energy---> 130.785
Energy deposited---> 0.0218983   Charge---> 6 Kinetic energy---> 130.76
Energy deposited---> 5.22223     Charge---> 6 Kinetic energy---> 125.541
Energy deposited---> 7.10685     Charge---> 6 Kinetic energy---> 118.434
Energy deposited---> 6.62999     Charge---> 6 Kinetic energy---> 111.804
Energy deposited---> 6.50997     Charge---> 6 Kinetic energy---> 105.294
Energy deposited---> 6.28403     Charge---> 6 Kinetic energy---> 99.0097
Energy deposited---> 5.77231     Charge---> 6 Kinetic energy---> 93.2374
Energy deposited---> 5.2333      Charge---> 6 Kinetic energy---> 88.0041
Energy deposited---> 3.9153      Charge---> 6 Kinetic energy---> 84.0888
Energy deposited---> 14.3767     Charge---> 6 Kinetic energy---> 69.7121
Energy deposited---> 14.3352     Charge---> 6 Kinetic energy---> 55.3769
```