XIV Seminar on Software for Nuclear, Subnuclear and Applied Physics

Alghero, 5-9 June 2017

Interaction with the Geant4 kernel I.

Geant4 tutorial

Contents

- Run, event, track, step
- Optional user action classes
- Command-based scoring
- Accumulables
- Analysis tools
- Multithreading

Part I:
Run, event, track, step

Run, Event and Tracks

The Run (G4Run)

- Collection of events with the same detector and physics conditions
- Within a run, the user **cannot change**
  - The detector setup
  - The physics setting (processes, models)
- At the beginning of a run, geometry is optimised for navigation and cross section tables are (re)calculated
- As an analogy with a real experiment, a run of Geant4 starts with '**Beam On**'
- Managed by G4(MT)RunManager
- Other related classes:
  - **`G4UserRunAction`** for an optional user hook
  - **`G4Run`** descendants to enrich with additional behaviour

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** from the stack one-by-one **and 'tracked'**
  - Secondary tracks are **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 Track (G4Track)

- The Track is a **snapshot of a particle**
- It keeps ‘current’ information of the particle (i.e. energy, momentum, position, polarization, ..) -> G4DynamicParticle
  - 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
- When there is no track left, **event** ends.

- `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)

G4Track properties

- GetTrackID()
- GetParentID()
- GetCreatorProcess()
- …

- GetDynamicParticle()

- GetTrack()

Note: Most of the Get…() methods are duplicated in G4Step, G4StepPoint, … and vice versa

The Step (G4Step)

- **`G4Step`** represents a step in the particle propagation
- 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 stepping action class *(later)*
- The `G4Step` contains **two points** (pre-step and post-step, of class G4StepPoint) and the **'delta'** information of a particle (energy loss on the step, ...)

G4Step properties

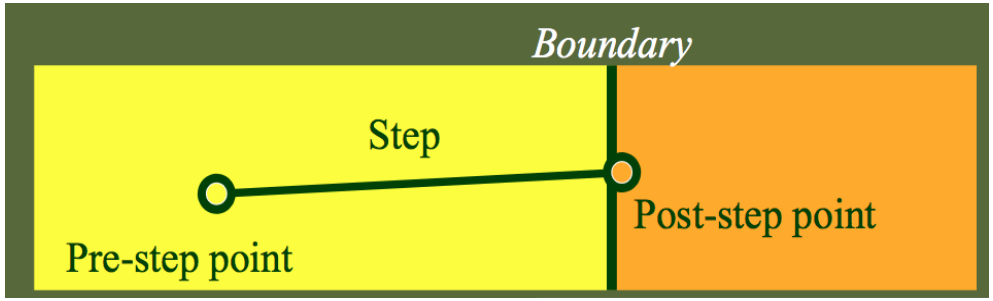
- GetStepLength()
- GetTotalEnergyDeposit()
- GetDeltaPosition()
- GetDeltaEnergy()
- GetDeltaMomentum()
- …

- GetTrack()

- GetPreStepPoint()
- GetPostStepPoint()

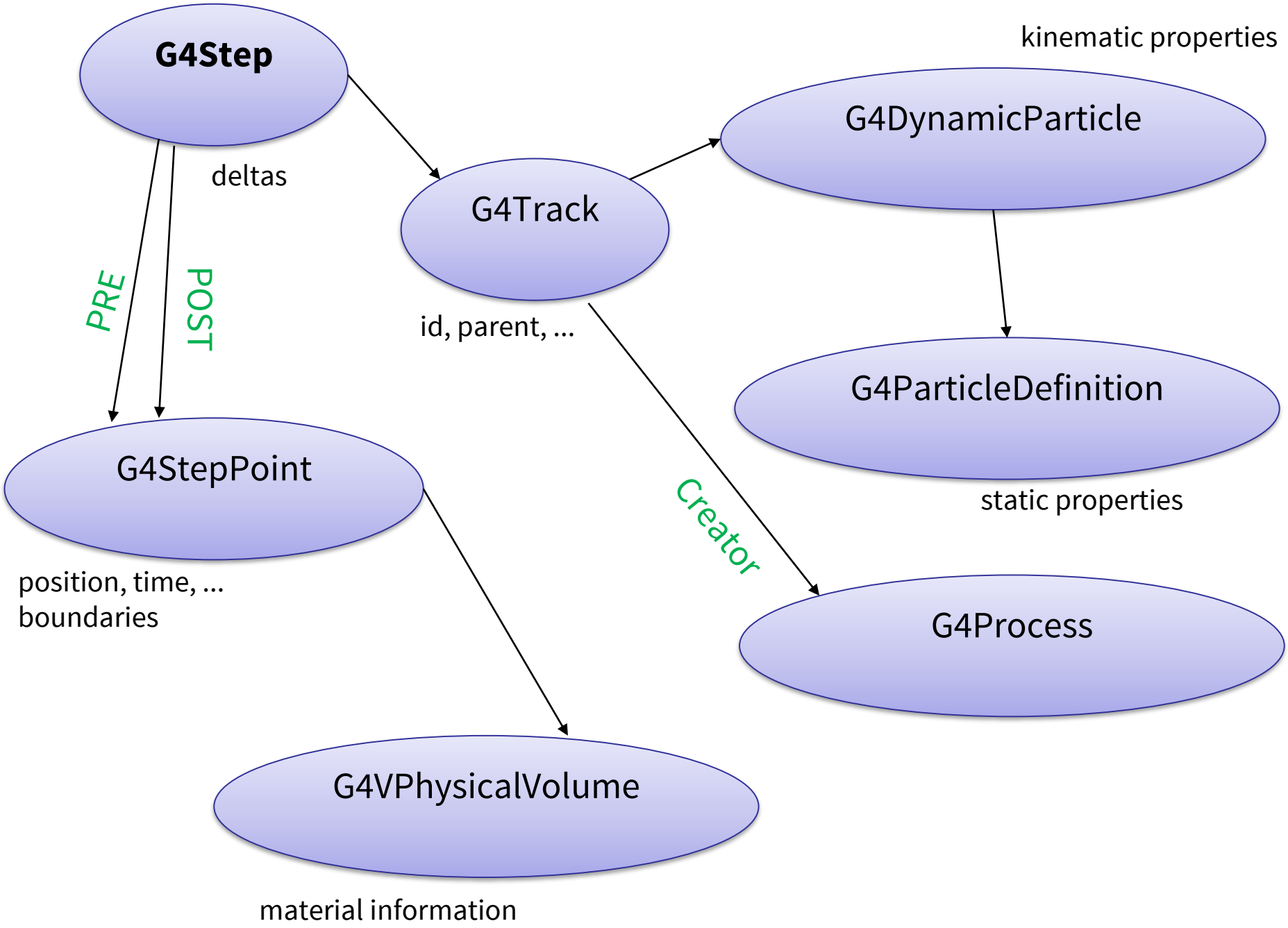
G4StepPoint properties

- GetPosition()
- GetLocalTime()
- GetGlobalTime()
- GetPhysicalVolume()
- GetStepStatus()



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**
  - 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


G4Step
deltas
PRE
POST
G4Track
id, parent, ...
kinematic properties
G4DynamicParticle
G4ParticleDefinition
static properties
G4StepPoint
position, time, ...
boundaries
Creator
G4Process
G4VPhysicalVolume
material information

Example: **parent tracks & process**

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

    if (track->GetParentID() == 1)
    {
        G4cout << "But parent was a primary" << G4endl;
    }

    // Get process information
    G4VProcess* creatorProcess = track->GetCreatorProcess();
    G4String processName = creatorProcess->GetProcessName();
    G4cout << "Particle was created by " << processName << G4endl;
    }
}
```

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();
```

Example: **step deltas**

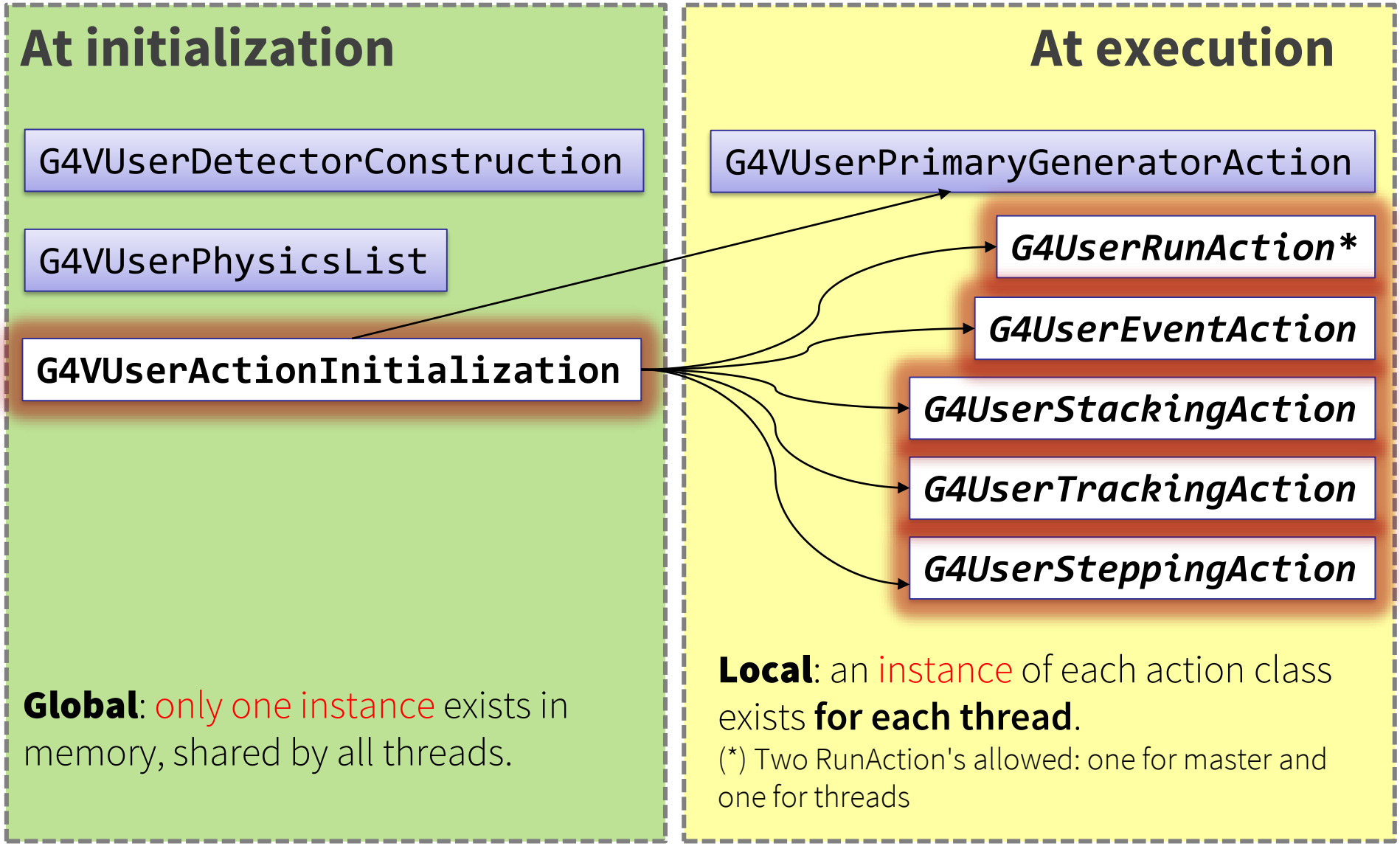
```
MySensitiveDetector::ProcessHits(G4Step* step, G4TouchableHistory* ignore) {
  // Total energy deposition on the step (= energy deposited by energy loss
  // process and energy of secondaries that were not created since their
  // process and energy of secondaries that were not created since their
  // 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();
}
```

Example: **particle information**

```
// 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 II:
Optional user action classes

...User classes (continued)



Optional user action classes

- Five **base classes** with **virtual methods** the user may override to step during the execution of the application ("user hooks“)
  - `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.
- Registered in the **action initialization** class

G4UserRunAction

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

Uses:

- Book/output histograms and other analysis tools
- Custom G4Run with additional information
- Define parameters

G4UserEventAction

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

Uses:

- Hit collection and event analysis
- Event selection
- Logging (e.g. output event number)

G4UserStackingAction 

```
G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)
void NewStage()
void PrepareNewEvent()
```

Uses:

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

G4UserTrackingAction

```
void PreUserTrackingAction(const G4Track*)
void PostUserTrackingAction(const G4Track*)
```

Uses:

- Track pre-selection
- Store trajectories

G4UserSteppingAction

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

Uses:

- Get information about particles
- Kill tracks under specific circumstances

User-defined run class

```
class MyRun : public G4Run
{ ... };
```

Virtual methods

- `RecordEvent()`
  - called at the end of each event
  - **alternative to EndOfEventAction()** of the EventAction class
- `Merge()`
  - Called at the end of each worker run by the **master**

When/why to use it?

- Convenient in **MT-mode**, because it allows the **merging** of information (global quantities) from **thread-local runs** into the master

User action classes registration

- In multi-threading mode, objects of user action classes must be **registered** to the **G4(MT)RunManager** via a user-defined action initialization class

MT

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

- In sequential mode, the actions can be registered to the run manager directly (not recommended).

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

MyActionInitialization

- 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());
}
```

Also the primary generator

- Register run action for the master (optional)

```
void MyActionInitialization::BuildForMaster() const
{

    SetUserAction(new MyMasterRunAction());
}
```

MT

Multiple user actions

- **G4MultiRunAction**
- **G4MultiEventAction**
- **G4MultiTrackingAction**
- **G4MultiSteppingAction**
- **no G4MultiStackingAction**

```
auto multiAction = new G4MultiEventAction{ new MyEventAction1, new MyEventAction2 };
//...
multiAction->push_back(new MyEventAction3);
SetUserAction(multiAction);
```

Containers enabling to have multiple user actions of the same "kind", implemented as customized `std::vector`'s.

Part III:
Command-based scoring

Command-based scoring

- Easy way to score physics information
- No C++, just macro commands
- Geometry independent of volumes
- Output in text format (CSV-like)
- Visualization capabilities

Necessary C++ lines

- Initialize **G4ScoringManager** in main:

```
// ...
#include <G4ScoringManager.hh>

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

- Put all the rest in the **macro file**...

Note: In principle, the scoring can be used directly from C++ code, but this is not recommended

Writing scoring macros

1) Create and place the scoring mesh
2) Add scorers (and filters)
3) Close the mesh
4) Run the simulation
5) Write to output file / visualize

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

Create scoring mesh

Define a mesh with a name:

```
/score/create/boxMesh myMesh
```

Set the dimensions:


```
/score/mesh/boxSize 1. 1. 1. m
```

Set the position:

```
/score/mesh/translate/xyz 1. 1. 0. cm
/score/mesh/rotate/rotateX 45 deg
```

Set the number of voxels:

```
/score/mesh/nBin 50 50 20
```

Add scorers

Add a scorer (using quantity and name)

```
/score/quantity/energyDeposit eDep
/score/quantity/nOfStep nOfStepGamma
```

Add filter applying to the current scorer

```
/score/filter/particle gammaFilter gamma
```

Available scorers

energyDeposit
cellCharge
cellFlux
passageCellFlux
doseDeposit
nOfSecondary
trackLength
passageCellCurrent
passageTrackLength
flatSurfaceCurrent
flatSurfaceFlux
nOfCollision
population
nOfTrack
nOfTerminatedTrack

See: /score/quantity

Available filters

- charged / neutral particles: **charged neutral**
- energy range: **kineticEnergy**
- particle name: **particle**
- combination: **particleWithKineticEnergy**

Scoring output

After run (i.e. after /run/beamOn), select scorers and export them to a file:

```
/score/dumpQuantityToFile myMesh scorerName output.csv
```

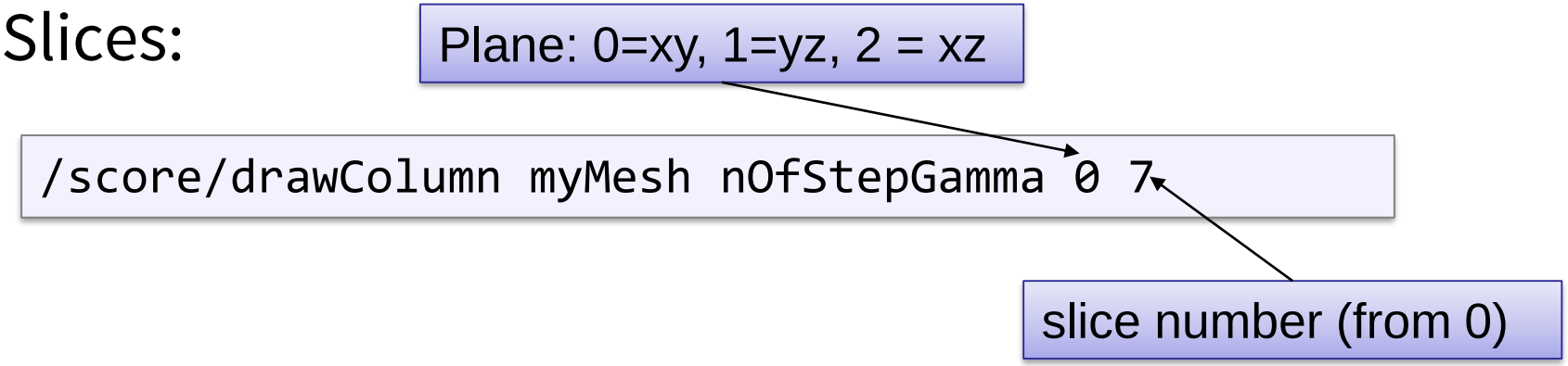
Visualization of scoring

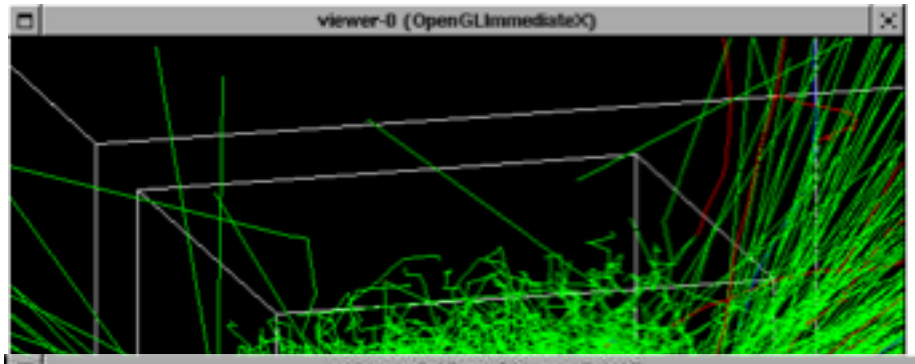
In the same window/view as geometry.

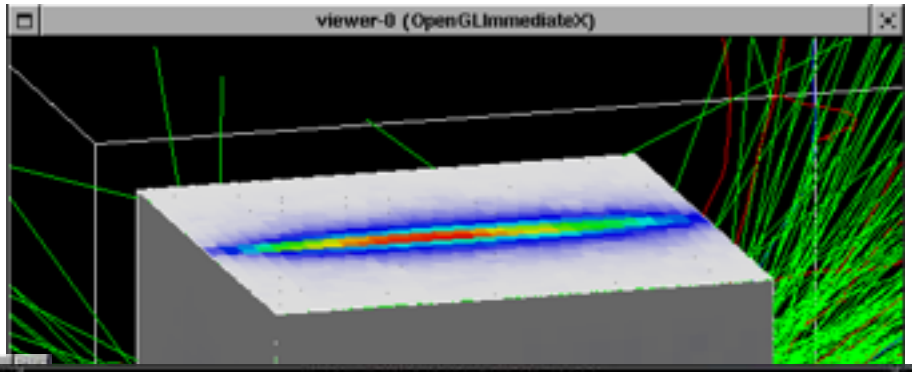
Projections:

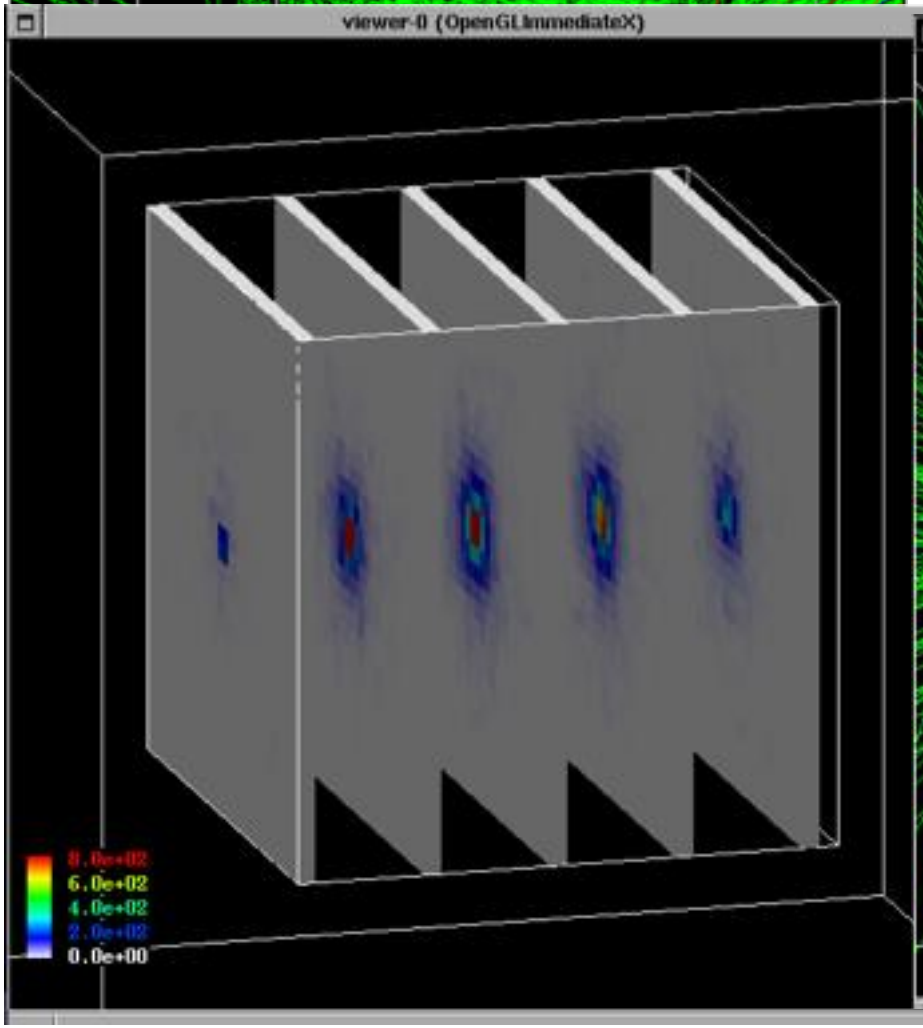
```
/score/drawProjection myMesh eDep
```

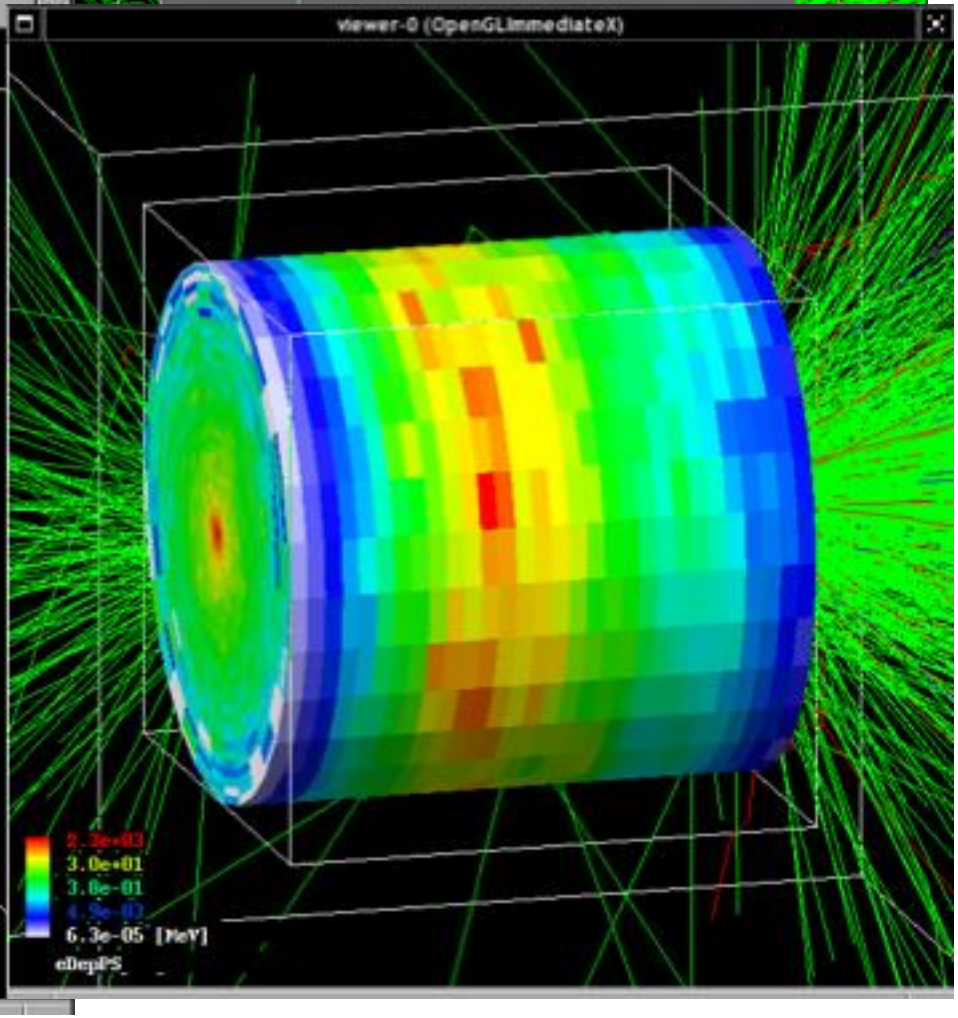
Slices:




viewer-0 (OpenGLImmediateX)


viewer-0 (OpenGLImmediateX)


viewer-0 (OpenGLImmediateX)
8.0e+02
6.0e+02
4.0e+02
2.0e+02
0.0e+00


viewer-0 (OpenGLImmediateX)
2.3e+03
3.0e+01
3.8e-01
4.9e-03
6.3e-05 [MeV]
eDeplPS

Example

```
/run/initialize

# Define the scoring
/score/create/boxMesh myMesh
/score/mesh/boxSize 100. 100. 100. cm
/score/mesh/nBin 30 30 30
/score/quantity/energyDeposit eDep
/score/quantity/nOfStep nOfStepGamma
/score/filter/particle gammaFilter gamma
/score/quantity/nOfStep nOfStepEMinus
/score/filter/particle eMinusFilter e-
/score/close

# Run the simulation
/run/beamOn 100

# Output
/score/dumpQuantityToFile myMesh eDep eDep.csv
/score/dumpQuantityToFile myMesh nOfStepGamma gammasteps.csv
/score/dumpQuantityToFile myMesh nOfStepEMinus electronsteps.csv
```

Part IV:
G4Accumulable

https://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch09s04.html

G4Accumulable<T>

- Templated class to collect simple information
  - Thread-safe
  - Accumulable during Run
  - Value merge at the end (explicit) MT
  - Scalar variables only (otherwise, expert)
- Alternative to ntuples/histograms *(later)*
- Managed by `G4AccumulableManager`

<=10.2: Previously named `G4Parameter`!

G4Accumulable – C++ (1)

1) Declare (instance) variables (of RunAction)

```
G4Accumulable<G4int>    fNElectrons;
G4Accumulable<G4double> fAverageElectronEnergy;
```

2) Register to accumulable manager (in RunAction constructor)

```
G4AccumulableManager* accManager = G4AccumulableManager::Instance();
accManager->RegisterAccumulable(fNElectrons);
accManager->RegisterAccumulable(fAverageElectronEnergy);
```

3) Reset to zero values (in RunAction::BeginOfRunAction)

```
G4AccumulableManager* accManager = G4AccumulableManager::Instance();
accManager->Reset();
```

4) Update during run (e.g. in Stacking action)

```
fNElectrons += 1;     // Normal arithmetics
```

► G4Accumulable.hh

G4Accumulable – C++ (2)

5) Merge after run (in RunAction::EndOfRunAction)

MT

```
G4AccumulableManager* accManager = G4AccumulableManager::Instance();
accManager->Merge();
```

6) Report after run (in RunAction::EndOfRunAction)

```
G4AccumulableManager* accManager = G4AccumulableManager::Instance();
if (IsMaster())
{
    if (fNElectrons.GetValue())
    {
        G4cout << " * Produced " << fNElectrons.GetValue();
        G4cout << " secondary electrons/event. Average energy: ";
        G4cout << fAverageElectronEnergy.GetValue() / keV / fNElectrons.GetValue();
        G4cout << " keV" << G4endl;
    }
    else
        G4cout << " * No secondary electrons produced" << G4endl;
}
```

Part V:
g4analysis tools

Geant4 analysis classes

- A basic analysis interface is available in Geant4 for histograms (1D and 2D) and ntuples
  - **Thread-safe** (ROOT is not! Manual text output usually not!)
- 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 another
- Everything is done using `G4AnalysisManager`
  - singleton class => use Instance()
  - **UI commands** available

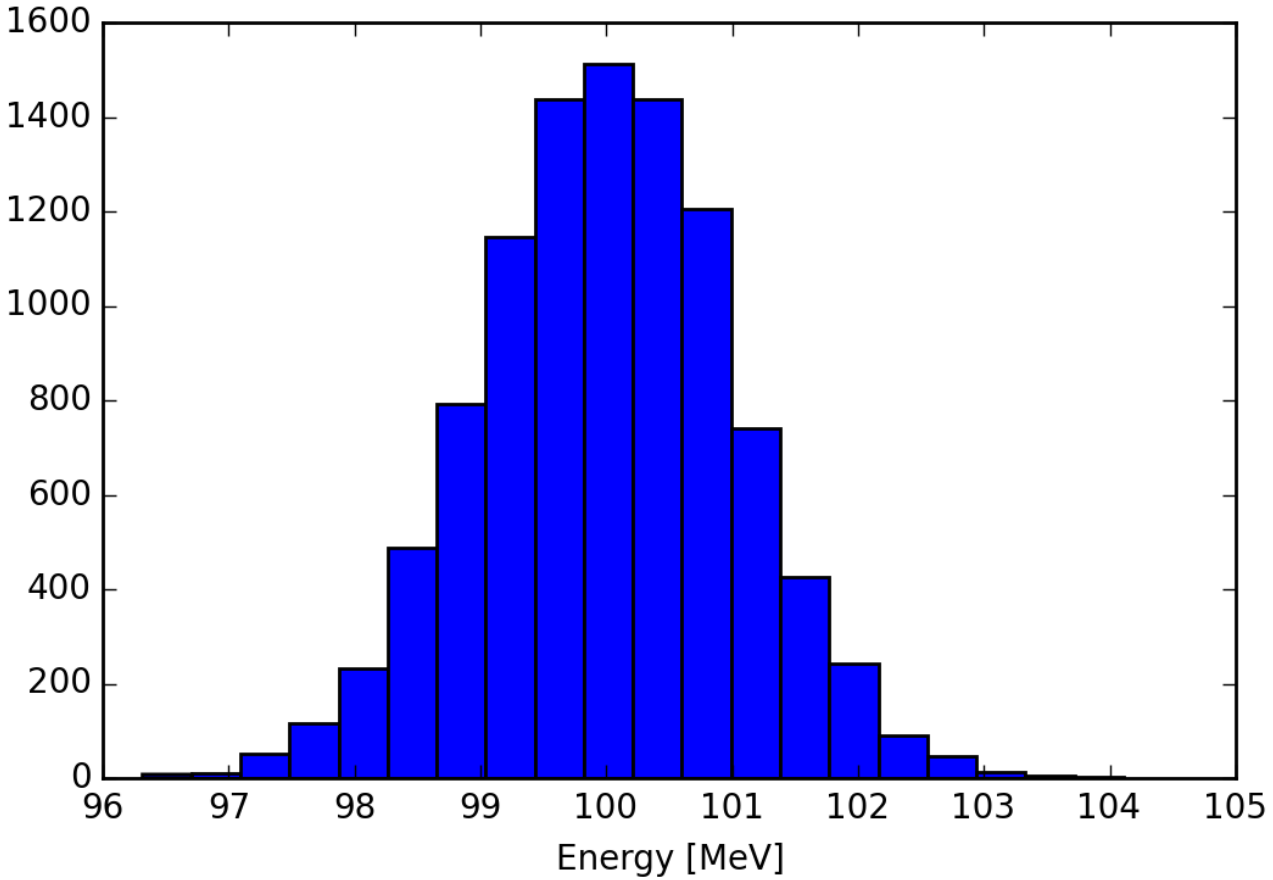
g4analysis

- 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), you 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

Part VI:
Multithreading

Execution modes in Geant4

- **Sequential mode**
  - everything run in one thread only
  - accepts both user actions and action initialization to support old code (Geant4 < 10.0)
- **Multithreaded mode**
  - “master” thread for the application
  - events simulated in multiple “worker” threads
  - accepts only action initialization
  - not supported in Windows OS ☹

Good news: The same code may support both modes!

Multithreading in Geant4

Main thread

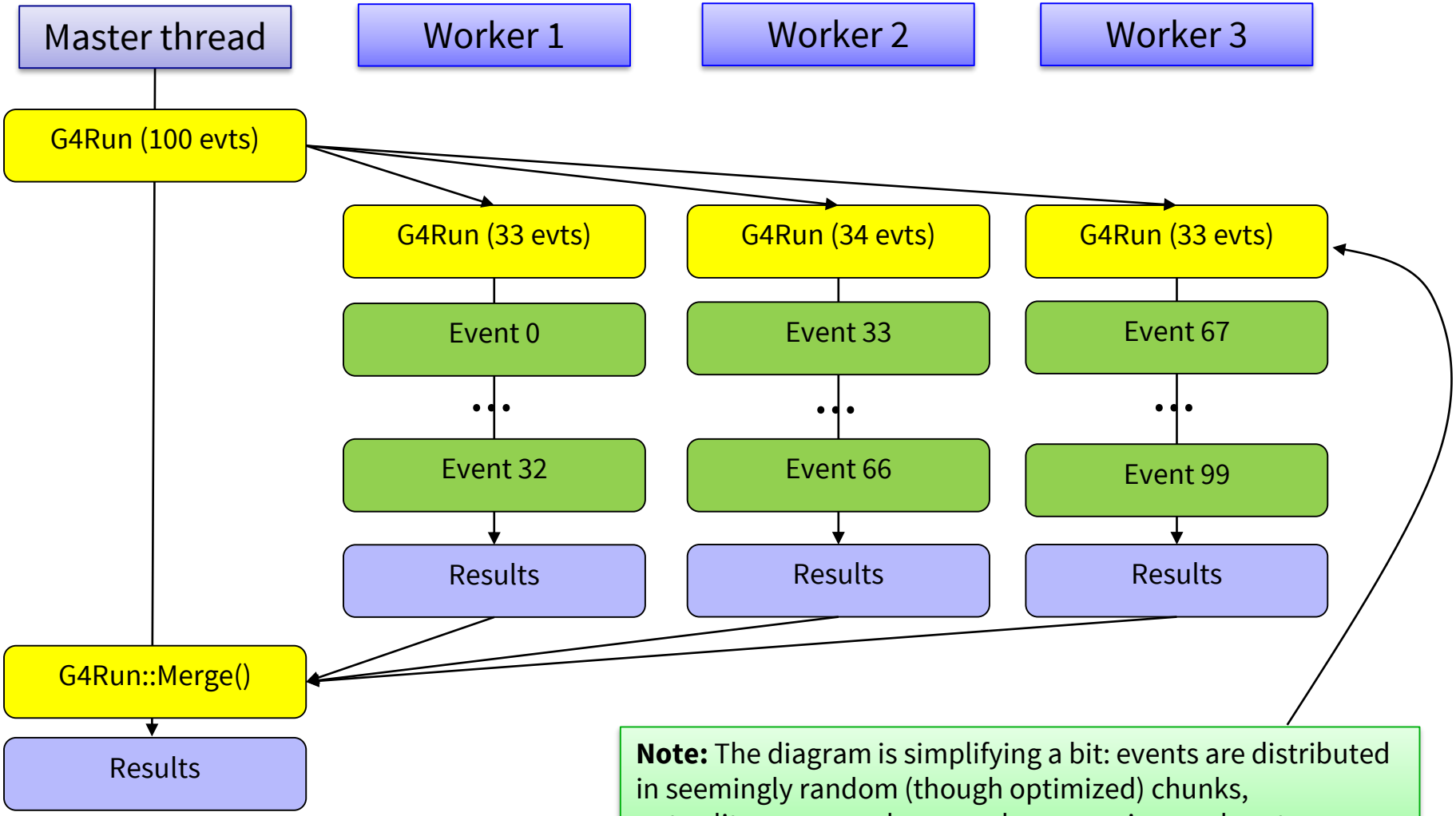
- initialize geometry and physics
- user interface
- start worker threads
- distribute events
- merge results

SPLIT RESPONSIBILITIES

Worker threads

- event simulation
- partial results
- user actions

Multithreaded processing of events



main() for both modes

- CMake setting
  -DGEANT4_BUILD_MULTITHREADED=ON/OFF
- Preprocessor macro G4MULTITHREADED

```
#include <G4MTRunManager.hh>
#include <G4RunManager.hh>

int main() {
    #ifdef G4MULTITHREADED
        G4MTRunManager* runManager = new G4MTRunManager;
    #else
        G4RunManager* runManager = new G4RunManager;
    #endif
    // ..
}
```

Good news!

You don’t have to care (too much) about threading issues, provided that you:

- Don’t manually open **external files** (more on that later)
- Use **g4analysis / command-based scoring** for output
- Avoid **static** variables and fields (especially in user actions)
- Correctly **merge runs** if using accumulables or hits
- Use the **G4(MT)RunManager** trick in main() (see above)
- Use **G4ActionInitialization**
- Don’t **experiment** with Geant4 kernel (especially not in user actions)

If you don’t meet these conditions, you must write thread-aware code.

…HANDS-ON SESSION

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


```
#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: 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

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
```