




















Geant4 Kernel

Makoto Asai
SLAC National Accelerator Laboratory

Contents

- Geant4 terminology
- UI commands
- User classes

Geant4 Physics & Applications

A Monte Carlo toolkit for passage of particles through matter

Geant4 Hadronic Physics

Hadronic interactions involve three main regimes : high energy, with string models (Quark Gluon String (QGS), Fritiof (PYTHIA)), intermediate energy, with intra-nuclear cascade models (Bertini (BERT), binary (BIC)), and low energy, with precompound, Fermi break-up, fission/evaporation, capture at rest models and radioactive decays. From 20 MeV down to thermal energy neutrons are handled by means of cross-section databases, with the High Precision (HP) package.

High Energy
Quark/gluon dominating behavior

Intermediate Energy
Nucleon dominating behavior

Low Energy
Nucleus dominating behavior

Neutron simulation down to thermal energies:

Geant4 can use the same neutron data library that MCNPX. Verification of neutron and gamma output of outgoing neutrons produced in neutron collision.

HEP Applications

High Energy Physics has been the first domain to use Geant4 in production, with the BaBar experiment. LHC experiments have been using Geant4 in detector design and are using it in physics analysis. Geant4 is also the simulation engine choice of the next generation of electron machines.

The CMS detector

The ATLAS detector

The recent High Luminosity

Responding to the simulation needs of the LHC era, with the Higgs boson hunting, had been the initial motivation of the creation of the proto-Geant4 project, RD44, in 1994.

Geant4 Electromagnetic Physics

The electromagnetic physics covers interactions of pions, muons and electrons, and ionization of all charged particles. A "standard" package offers an implementation suited for applications disregarding effects below a few ~10 keV, and a "low energy" one provides approaches (Livermore, Penelope) for more accurate modeling of atomic shell effects allowing simulation down to ~250 eV. A very low extension, Geant4-DNA, includes particle-molecule effects for an energy limit of ~10 eV. The same approach is developed for silicon.

Proton neutron

Carbon ion

Water

Molecule

Gamma

Space Applications

Applications of Geant4 in space cover planetary scale simulation for soil level media activation studies, soil composition through X-ray re-emission, space ship simulation for radio-protection and electronic single event upset predictions, electronic chip scale simulation for accurate understanding of single event upset generation. It includes also underground, ground level or satellite cosmic ray experiments simulation.

Planetocosmos : a simulation tool for planetary scale particle transport. The red curve is a proton trajectory in the Earth magnetic field, interaction level around a planet, at ground level, and with related activated isotopes can then be predicted.

Very Low Energy
Atomic and molecular structures dominating

DNA Scale Level Simulation

Project initiated by the ESA, in view of manned mission to Mars: it is a bottom-up approach of dosimetry. Physics processes are extended down to a few eV, based on particle-molecule cross-sections. The approach is applied also to silicon, for accurate simulation of Single Upset Events.

DNA geometry model simulated : 46 Simulation of water chemical species migration accounting for electrical mutual interaction after a 50 MeV proton irradiation. Post irradiation chemical attacks amount for ~60% of total damages on DNA.




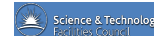

Medical Applications










Medical Applications interest in Monte Carlo is the accuracy capability in complex structures. Geant4 is used for radio-therapy medical research fields. It is used also in optimization of brachytherapy devices, radio-protection and nuclear imaging. Large users communities exist in US, Europe and Japan. CPU performance boost allowed by Geant4 MT or by GPU prototype versions open the possibility for routine usage in treatment planning.


Proton beam line, range shifter and dose deposit simulations in HiBMC (Japan). The proton energy is 150 MeV (TASO IIRL NGS 2007 N60-1).

DICOM geometry and dose calculation with g4mcrcm a tool.

Projectile de Broglie λ (fm)



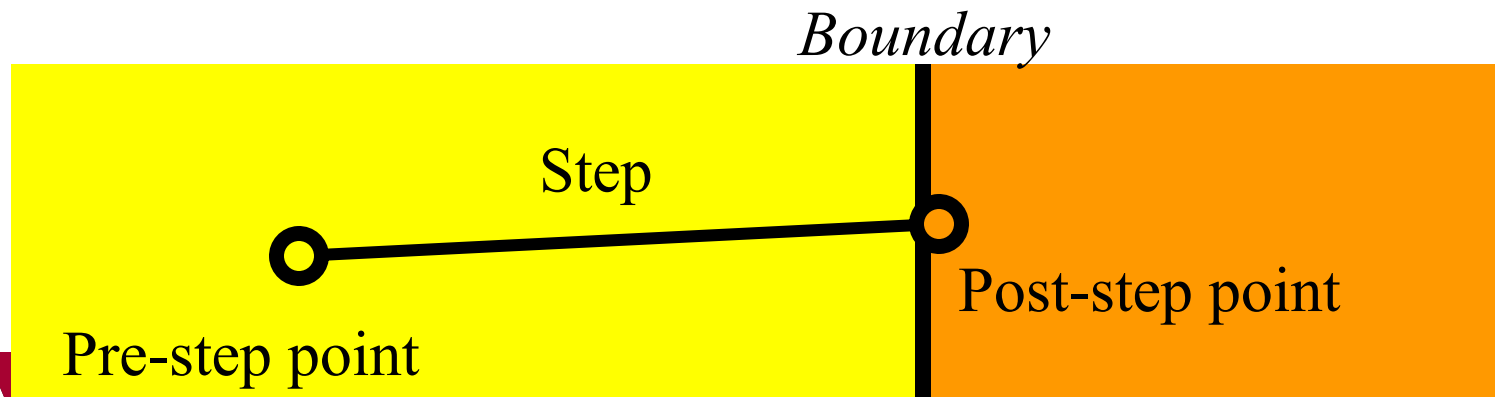
- Run, event, track, step, step point
- Track \leftrightarrow trajectory, step \leftrightarrow trajectory point
- Process
 - At rest, along step, post step
- Cut = production threshold

- As an analogy of the real experiment, a run of Geant4 starts with “Beam On”.
- Within a run, the user cannot change
 - detector setup
 - settings of physics processes
- Conceptually, a run is a collection of events which share the same detector and physics conditions.
 - A run consists of one event loop.
- At the beginning of a run, geometry is optimized for navigation and cross-section tables are calculated according to materials appear in the geometry and the cut-off values defined.
- **G4RunManager** class manages processing a run, a run is represented by **G4Run** class or a user-defined class derived from G4Run.
 - A run class may have a summary results of the run.
- **G4UserRunAction** is the optional user hook.

- An event is the basic unit of simulation in Geant4.
- At beginning of processing, primary tracks are generated. These primary tracks are pushed into a stack.
- A track is popped up from the stack one by one and “tracked”. Resulting secondary tracks are pushed into the stack.
 - This “tracking” lasts as long as the stack has a track.
- When the stack becomes empty, processing of one event is over.
- **G4Event** class represents an event. It has following objects at the end of its (successful) processing.
 - List of primary vertices and particles (as input)
 - Hits and Trajectory collections (as output)
- **G4EventManager** class manages processing an event. **G4UserEventAction** is the optional user hook.

- Track is a **snapshot** of a particle.
 - It has physical quantities of **current instance** only. It does not record previous quantities.
 - **Step is a “delta” information to a track. Track is not a collection of steps. Instead, a track is being updated by steps.**
- Track object is deleted when
 - it goes out of the world volume,
 - it disappears (by e.g. decay, inelastic scattering),
 - it goes down to zero kinetic energy and no “AtRest” additional process is required, or
 - the user decides to kill it artificially.
- **No track object persists at the end of event.**
 - For the record of tracks, use trajectory class objects.
- **G4TrackingManager** manages processing a track, a track is represented by **G4Track** class.
- **G4UserTrackingAction** is the optional user hook.

- Step has two points and also “delta” information of a particle (energy loss on the step, time-of-flight spent by the step, etc.).
- Each point knows the volume (and 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**.
 - Because one step knows materials of two volumes, boundary processes such as transition radiation or refraction could be simulated.
- **G4SteppingManager** class manages processing a step, a step is represented by **G4Step** class.
- **G4UserSteppingAction** is the optional user hook.



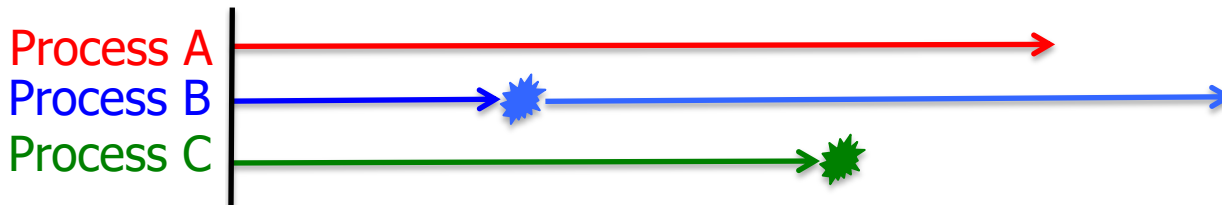
- Track does not keep its trace. No track object persists at the end of event.
- **G4Trajectory** is the class which copies some of G4Track information.
G4TrajectoryPoint is the class which copies some of G4Step information.
 - G4Trajectory has a vector of G4TrajectoryPoint.
 - At the end of event processing, G4Event has a collection of G4Trajectory objects.
 - /tracking/storeTrajectory must be set to 1.
- Keep in mind the distinction.
 - $G4Track \leftrightarrow G4Trajectory$, $G4Step \leftrightarrow G4TrajectoryPoint$
- Given G4Trajectory and G4TrajectoryPoint objects persist till the end of an event, you should be careful not to store too many trajectories.
 - E.g. avoid for high energy EM shower tracks.
- G4Trajectory and G4TrajectoryPoint store only the minimum information.
 - You can create your own trajectory / trajectory point classes to store information you need. G4VTrajectory and G4VTrajectoryPoint are base classes.

- A particle in Geant4 is represented by three layers of classes.
- **G4Track**
 - Position, geometrical information, etc.
 - This is a class representing a particle to be tracked.
- **G4DynamicParticle**
 - "Dynamic" physical properties of a particle, such as momentum, energy, spin, etc.
 - Each G4Track object has its own and unique G4DynamicParticle object.
 - This is a class representing an individual particle.
- **G4ParticleDefinition**
 - "Static" properties of a particle, such as charge, mass, life time, decay channels, etc.
 - G4ProcessManager which describes processes involving to the particle
 - All G4DynamicParticle objects of same kind of particle share the same G4ParticleDefinition.

- In Geant4, particle transportation is a process as well, by which a particle interacts with geometrical volume boundaries and field of any kind.
 - Because of this, shower parameterization process can take over from the ordinary transportation without modifying the transportation process.
- Each particle type (i.e. G4ParticleDefinition object) has its own list of applicable processes. At each step, all processes listed are invoked to get proposed physical interaction lengths.
- The process which requires the shortest interaction length (in space-time) limits the step.
- Each process has one or combination of the following natures.
 - AtRest
 - e.g. muon decay at rest
 - AlongStep (a.k.a. continuous process)
 - e.g. Cerenkov process
 - PostStep (a.k.a. discrete process)
 - e.g. decay on the fly

Process competition

- “Ordinary” physics makes point-like interaction. Given many physics processes have chances to occur, one needs to make a fair competition among these eligible processes.
- Given PDF of each process, one can sample the path length **normalized by mean free path** (radiation length, hadronic interaction length, decay time, etc.) for each physics process.
- Compare the path lengths proposed by all physics processes. The process that proposes the shortest length occurs.
 - Given the length is normalized, competition should be made by the actual length (normalized length x mean free path of the material).
- Once the particle experiences an interaction by a physics process, the path length for that process is re-sampled, while proposed path lengths of other processes are reduced by the length traveled.
- Continuous processes (continuous energy loss, multiple scattering, Cherenkov radiation, etc.) are applied cumulatively.



- A Cut in Geant4 is a **production threshold**.
 - Not tracking cut, which does not exist in Geant4 as default.
 - **All tracks are traced down to zero kinetic energy.**
 - It is applied **only** for physics processes that have infrared divergence
- Much detail will be given at later talks on physics.

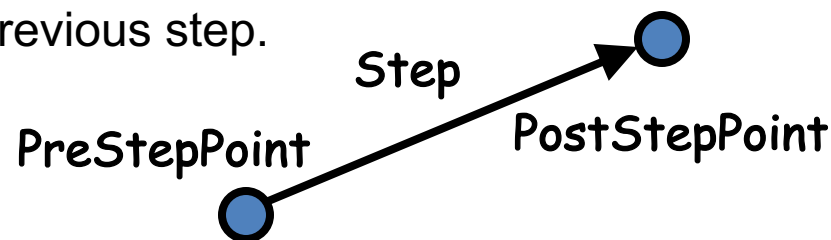
Track status

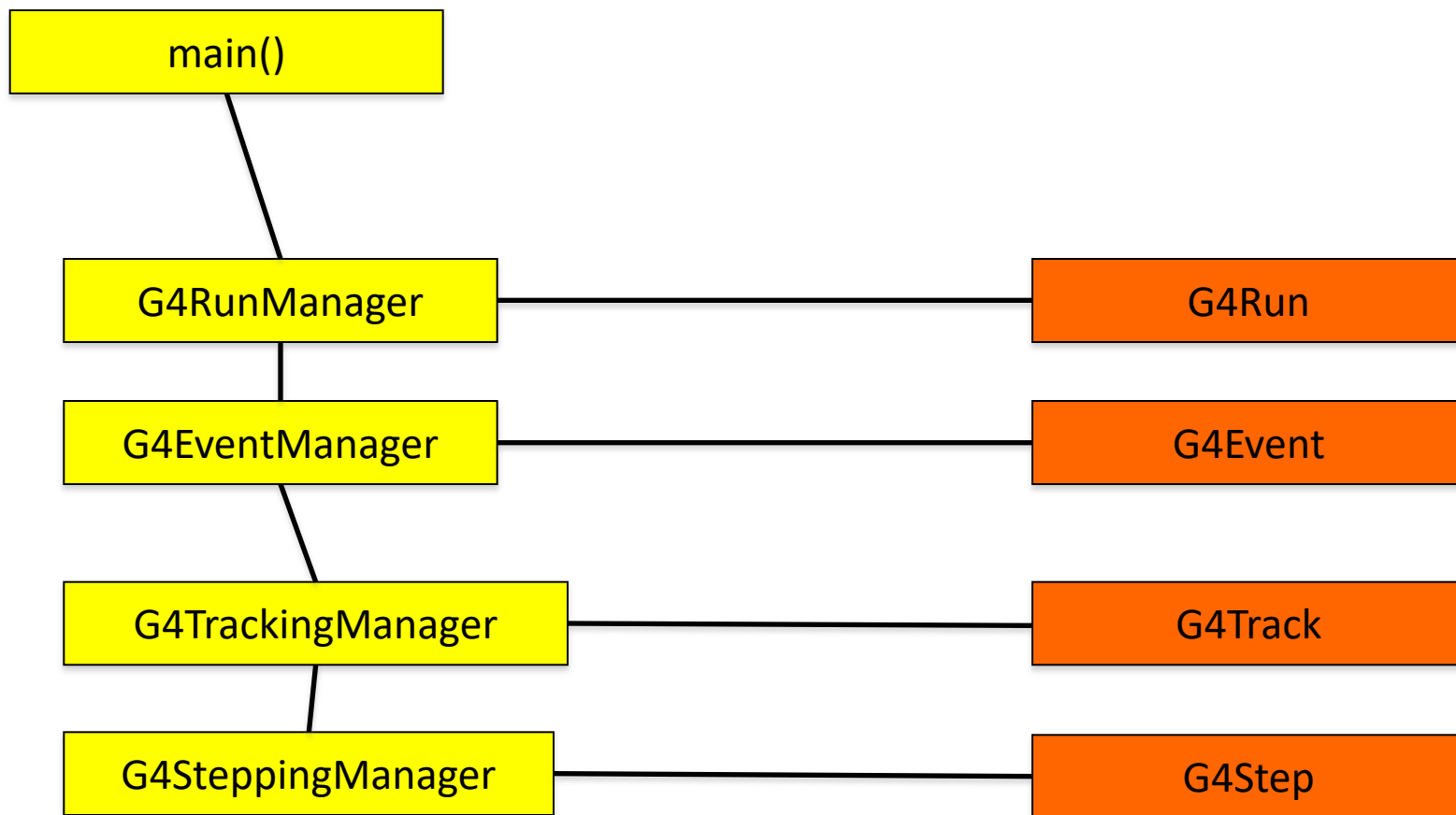
- At the end of each step, according to the processes involved, the state of a track may be changed.
 - The user can also change the status in **UserSteppingAction**.
 - Statuses shown in **green** are artificial, i.e. Geant4 kernel won't set them, but the user can set.
- fAlive
 - Continue the tracking.
- fStopButAlive
 - The track has come to zero kinetic energy, but still AtRest process to occur.
- fStopAndKill
 - The track has lost its identity because it has decayed, interacted or gone beyond the world boundary.
 - Secondaries will be pushed to the stack.
- **fKillTrackAndSecondaries**
 - Kill the current track and also associated secondaries.
- **fSuspend**
 - Suspend processing of the current track and push it and its secondaries to the stack.
- **fPostponeToNextEvent**
 - Postpone processing of the current track to the next event.
 - Secondaries are still being processed within the current event.

Step status

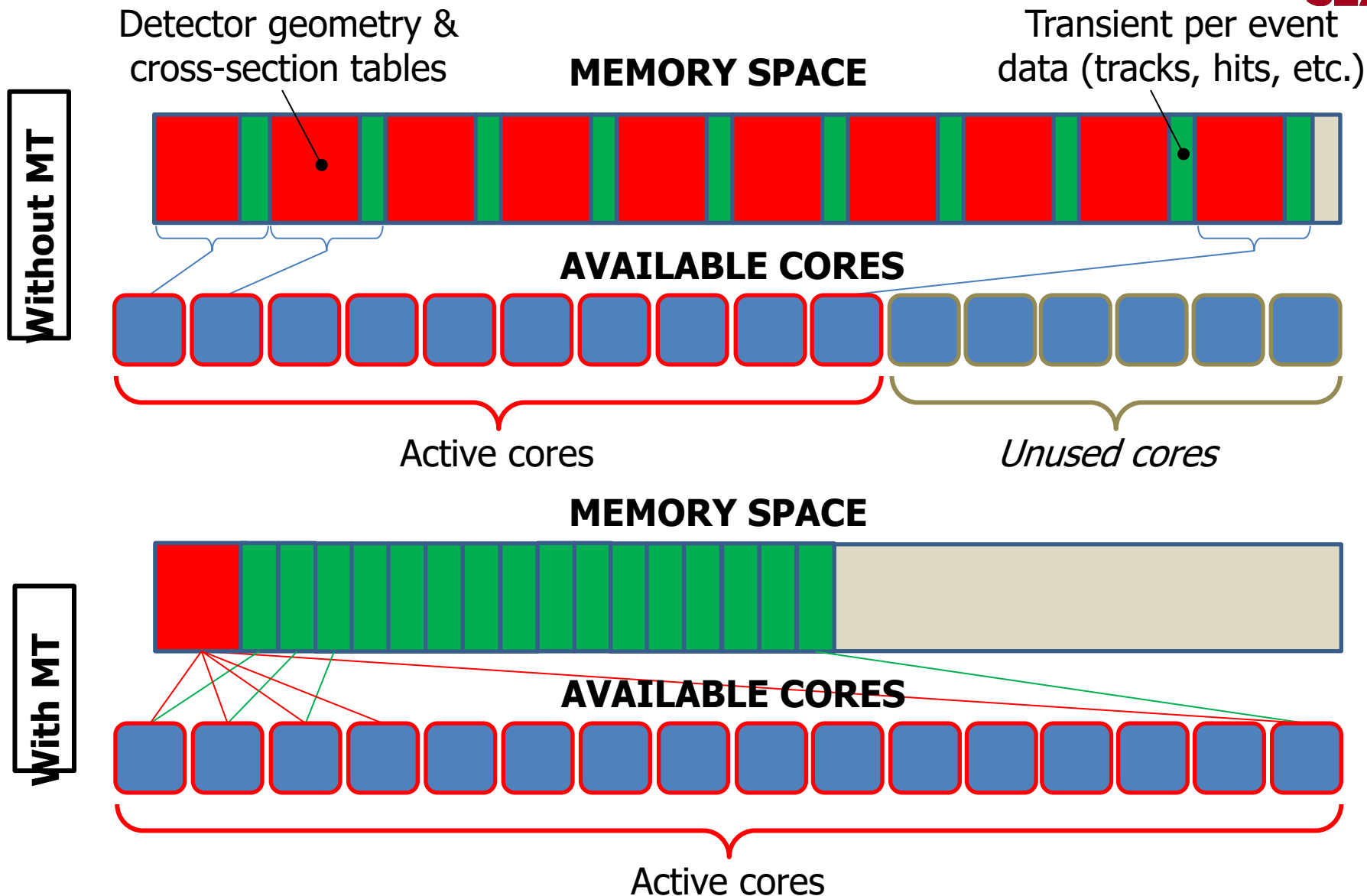


- Step status is attached to G4StepPoint to indicate why that particular step was determined.
 - Use “**PostStepPoint**” to get the status of this step.
 - “**PreStepPoint**” has the status of the previous step.
- fWorldBoundary
 - Step reached the world boundary
- fGeomBoundary
 - Step is limited by a volume boundary except the world
- fAtRestDoltProc, fAlongStepDoltProc, fPostStepDoltProc
 - Step is limited by a AtRest, AlongStep or PostStep process
- fUserDefinedLimit
 - Step is limited by the user Step limit
- fExclusivelyForcedProc
 - Step is limited by an exclusively forced (e.g. shower parameterization) process
- fUndefined
 - Step not defined yet
- If you want to identify **the first step in a volume**, pick **fGeomBoudary** status in **PreStepPoint**.
- If you want to identify **a step getting out of a volume**, pick **fGeomBoundary** status in **PostStepPoint**

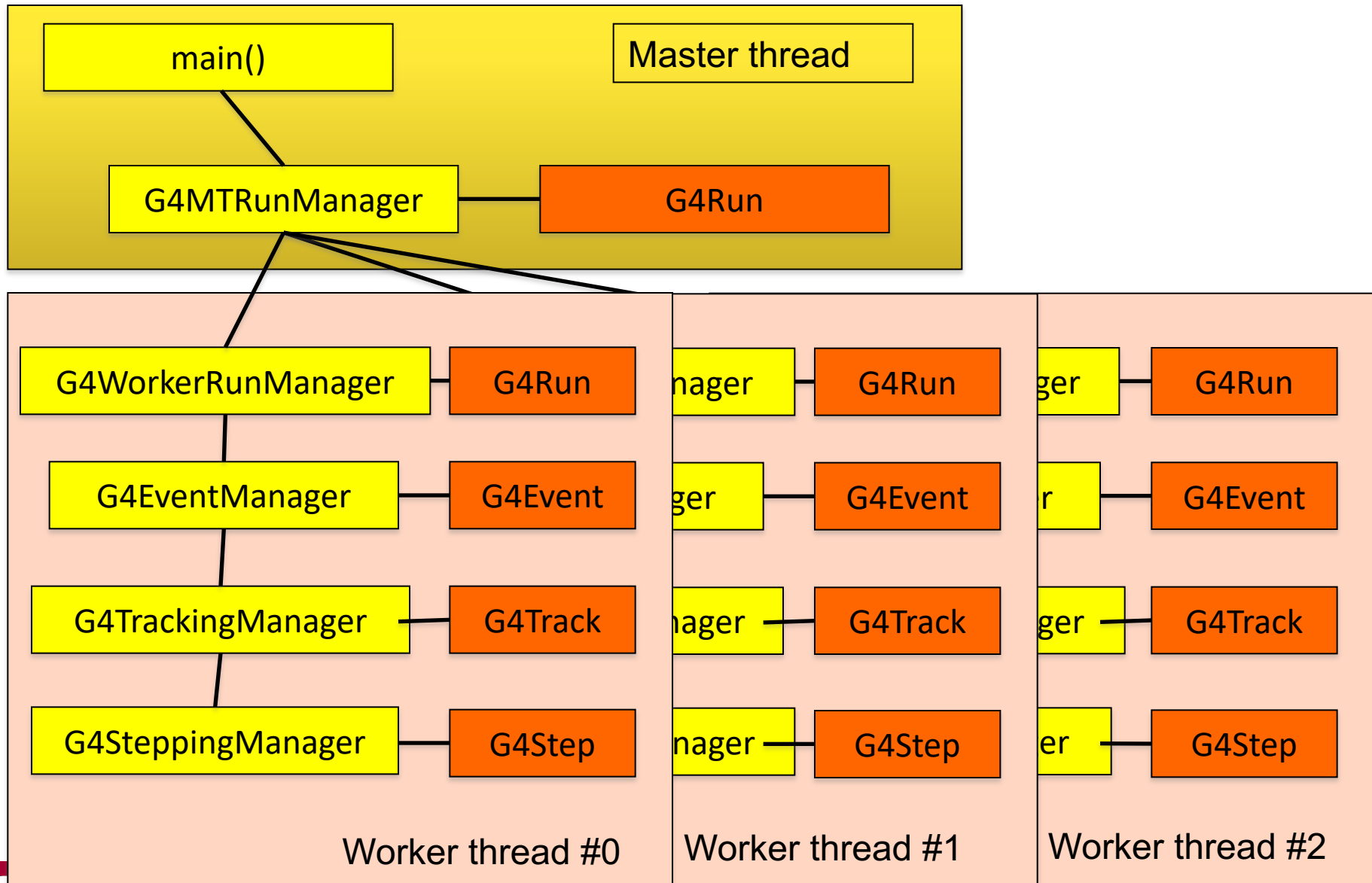




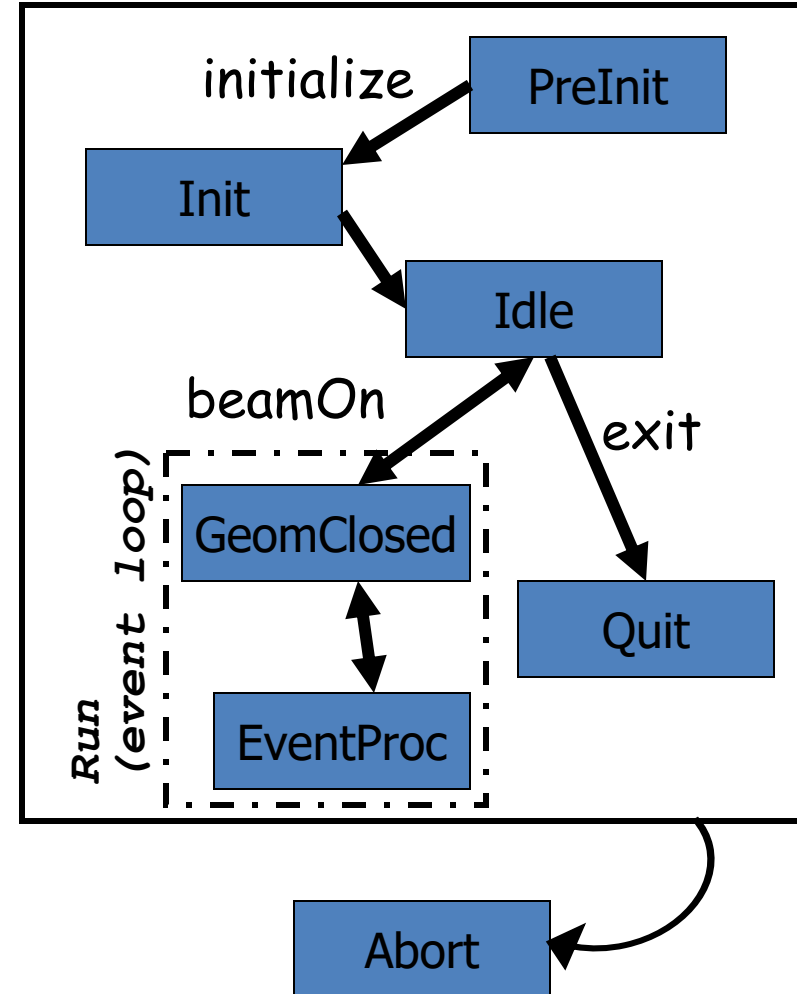
Geant4 runs in multithreaded mode



- In the multi-threaded mode, generally saying, data that are stable during the event loop are shared among threads while data that are transient during the event loop are thread-local.
- In general, geometry and physics tables are shared, while event, track, step, trajectory, hits, etc., as well as several Geant4 manager classes such as EventManager, TrackingManager, SteppingManager, TransportationManager, FieldManager, Navigator, SensitiveDetectorManager, etc. are thread-local.
 - All physics processes are also thread-local.
- Among the user classes, user initialization classes (G4VUserDetectorConstruction, G4VUserPhysicsList and newly introduced G4VUserActionInitialization) are shared, while all user action classes and sensitive detector classes are thread-local.
 - It is not straightforward (and thus not recommended) to access from a shared class object to a thread-local object, e.g. from detector construction to stepping action.
 - Please note that thread-local objects are instantiated and initialized at the first *BeamOn*.
- To avoid potential errors, it is advised to always keep in mind which class is shared and which class is thread-local.



- Geant4 has seven application states.
 - G4State_PreInit
 - Initial condition
 - G4State_Init
 - During initialization
 - G4State_Idle
 - Ready to start a run
 - G4State_GeomClosed
 - Geometry is optimized and ready to process an event
 - G4State_EventProc
 - An event is processing
 - G4State_Quit
 - (Normal) termination
 - G4State_Abort
 - A fatal exception occurred and program is aborting



Note: Toggles between *GeomClosed* and *EventProc* occur for each thread asynchronously in multithreaded mode.

- Internal unit system used in Geant4 is completely hidden not only from user's code but also from Geant4 source code implementation.
- Each hard-coded number must be multiplied by its proper unit.

```
radius = 10.0 * cm;
```

```
kineticE = 1.0 * GeV;
```

- To get a number, it must be divided by a proper unit.

```
G4cout << eDep / MeV << " [MeV]" << G4endl;
```

- Most of commonly used units are provided and user can add his/her own units.
- By this unit system, source code becomes more readable and importing / exporting physical quantities becomes straightforward.
 - For particular application, user can change the internal unit to suitable alternative unit without affecting to the result.

- **G4cout** and **G4cerr** are *ostream* objects defined by Geant4.
 - **G4endl** is also provided.

```
G4cout << "Hello Geant4!" << G4endl;
```

- Some GUIs are buffering output streams so that they display print-outs on another window or provide storing / editing functionality.
 - The user should not use `std::cout`, etc.
- The user should not use `std::cin` for input. Use user-defined commands provided by intercoms category in Geant4.
 - Ordinary file I/O is OK.

UI commands

- A UI command consists of
 - Command directory `/run/verbose 1`
 - Command `/vis/viewer/flush`
 - Parameter(s)
- A parameter can be a type of string, boolean, integer or double.
 - Space is a delimiter.
 - Use double-quotes (“”) for string with space(s).
- A parameter may be “omittable”. If it is the case, a default value will be taken if you omit the parameter.
 - Default value is either predefined default value or current value according to its definition.
 - If you want to use the default value for your first parameter while you want to set your second parameter, use “!” as a place holder.

`/dir/command ! second`

- Geant4 UI command can be issued by
 - (G)UI interactive command submission
 - Macro file
 - Hard-coded implementation
 - Slow but no need for the targeting class pointer
 - Should **not** be used inside an event loop

```
G4UImanager* UI = G4UImanager::GetUIpointer();  
UI->ApplyCommand("/run/verbose 1");
```

- The availability of individual command, the ranges of parameters, the available candidates on individual command parameter **may vary** according to the implementation of your application and may even **vary dynamically** during the execution of your job.
- some commands are available only for limited Geant4 **application state(s)**.
 - E.g. `/run/beamOn` is available only for *Idle* states.

- Command will be refused in case of
 - Wrong application state
 - Wrong type of parameter
 - Insufficient number of parameters
 - Parameter out of its range
 - For integer or double type parameter
 - Parameter out of its candidate list
 - For string type parameter
 - Command not found

- Macro file is an ASCII file contains UI commands.
- All commands must be given with their **full-path directories**.
- Use “#” for comment line.
 - First “#” to the end of the line will be ignored.
 - Comment lines will be echoed if `/control/verbose` is set to 2.
- Macro file can be executed
 - interactively or in (other) macro file

```
/control/execute file_name
```

- hard-coded

```
G4UImanager* UI = G4UImanager::GetUIpointer();  
UI->ApplyCommand("/control/execute file_name");
```

- You can get a list of available commands **including your custom ones** by
 - `/control/manual [directory]`
 - Plain text format to standard output
 - `/control/createHTML [directory]`
 - HTML file(s) - one file per one (sub-)directory
- List of built-in commands is also available in section 7.1 of *User's Guide For Application Developers*.

- Alias can be defined by

```
/control/alias [name] [value]
```

- It is also set with `/control/loop` and `/control/foreach` commands
- Aliased value is always treated as a string even if it contains numbers only.

- Alias is to be used with other UI command.

- Use curly brackets, `{` and `}`.
- For example, frequently used lengthy command can be shortened by aliasing.

```
/control/alias tv /tracking/verbose
```

```
{tv} 1
```



- Aliases can be used recursively.

```
/control/alias file1 /diskA/dirX/fileXX.dat
```

```
/control/alias file2 /diskB/dirY/fileYY.dat
```

```
/control/alias run 1
```

```
/myCmd/getFile {file{run}}
```


- `/control/loop` and `/control/foreach` commands execute a macro file more than once. Aliased variable name can be used inside the macro file.
- `/control/loop` *[macroFile]* *[counterName]*
[initialValue] *[finalValue]* *[stepSize]*
 - *counterName* is aliased to the number as a loop counter
- `/control/foreach` *[macroFile]* *[counterName]* *[valueList]*
 - *counterName* is aliased to a value in *valueList*
 - *valueList* must be enclosed by double quotes (" ")
- on UI terminal or another macro file
`/control/loop myRun.mac Ekin 10. 20. 2.`
- in myRun.mac 
`/control/foreach mySingleRun.mac pname "p pi- mu-"`
- in mySingleRun.mac 
`/gun/particle {pname}`
`/gun/energy {Ekin} GeV`
`/run/beamOn 100`

- `/control/getEnv [shell_variable_name]`
 - Get a shell environment variable and define it as an alias
- `/control/getVal [aliasName] [UI_command]`
 - Get the current value of a UI command and set it to the alias
- `/control/doif [val1] [comp] [val2] [UI_command]`
`/control/strdoif [str1] [comp] [str2] [UI_command]`
 - Execute the UI command if comparison is true
- `/control/doifInteractive [UI_command]`
`/control/doifBatch [UI_command]`
 - Execute the UI command if running in interactive/batch mode
- `/control/add [new_val] [val1] [val2]`
`/control/subtract`, `/control/multiply`, `/control/divide`, `/control/remainder`

Some convenient com

- /control/getEnv [shell_var]
 - Get a shell environment variable
- /control/getVal [aliasName]
 - Get the current value of an alias
- /control/doif [val1] [command]
/control/strdoif [str1] [command]
 - Execute the UI command if the condition is true
- /control/doifInteractive [val1] [command]
/control/doifBatch [UI_command]
 - Execute the UI command if the condition is true
- /control/add [new_val] [val]
/control/subtract, /control/multiply

```
#####  
# Selection of primary generator  
#####  
/eAST/generator/useParticleGun 0  
/eAST/generator/useParticleSource 1  
/eAST/generator/useHepMC3 0  
#####  
# eAST and Geant4 initialization  
#####  
/eAST/initialize  
#####  
# Set up general particle source if it is used  
#####  
/control/getVal ifGPS /eAST/generator/useParticleSource  
/control/doif {ifGPS} > 0 /control/execute gps_point.mac  
#####  
# Run for batch mode  
#####  
/control/doifBatch /run/eventModulo 0 1  
/control/doifBatch /run/beamOn 100  
#####  
# Run for interactive mode  
#####  
/control/doifInteractive /run/beamOn 0  
/control/doifInteractive /control/execute vis.mac
```

- In your *main()*

```
int main(int argc, char** argv)
{
    ...
    if (argc != 1)
    { // batch mode
        G4String command = "/control/execute ";
        G4String fileName = argv[1];
        Ulmanager->ApplyCommand(command+fileName);
    }
    else
    { // interactive mode : define UI session
        G4UIExecutive* ui = new G4UIExecutive(argc, argv);
        ui->SessionStart();
        delete ui;
    }
}
```

User classes

To use Geant4, you have to...

- Geant4 is a toolkit. You have to build an application.
- To make an application, you have to
 - Define your geometrical setup
 - Material, volume
 - Define physics to get involved
 - Particles, physics processes/models
 - Production thresholds
 - Define how an event starts
 - Primary track generation
 - Extract information useful to you
- You may also want to
 - Visualize geometry, trajectories and physics output
 - Utilize (Graphical) User Interface
 - Define your own UI commands
 - etc.

- `main()`
 - Geant4 does not provide *main()*.
- Initialization classes
 - Use `G4RunManager::SetUserInitialization()` to define.
 - Invoked at the initialization
 - `G4VUserDetectorConstruction`
 - `G4VUserPhysicsList`
 - `G4VUserActionInitialization`
- Action classes
 - Instantiate in your `G4VUserActionInitialization`.
 - Invoked during an event loop
 - `G4VUserPrimaryGeneratorAction`
 - `G4UserRunAction`
 - `G4UserEventAction`
 - `G4UserStackingAction`
 - `G4UserTrackingAction`
 - `G4UserSteppingAction`

Note : classes written in **red** are mandatory.

- Geant4 does not provide a *main()*.
- In your *main()*, you have to
 - Construct G4RunManager (sequential mode) or G4MTRunManager (multithreaded mode)
 - Set user mandatory initialization classes to RunManager
 - G4VUserDetectorConstruction
 - G4VUserPhysicsList
 - G4VUserActionInitialization
- You can define VisManager, (G)UI session, optional user action classes, and/or your persistency manager in your *main()*.

- Derive your own concrete class from **G4VUserDetectorConstruction** abstract base class.
- In the virtual method *Construct()*, that is invoked in the master thread (and in sequential mode)
 - Instantiate all necessary materials
 - Instantiate volumes of your detector geometry
- In the virtual method *ConstructSDandField()*, that is invoked in each worker thread (and in sequential mode)
 - Instantiate your sensitive detector classes and field classes and set them to the corresponding logical volumes and field managers, respectively.

- Geant4 does not have any default particles or processes.
 - Even for the particle transportation, you have to define it explicitly.
- Derive your own concrete class from **G4VUserPhysicsList** abstract base class.
 - Define all necessary particles
 - Define all necessary processes and assign them to proper particles
 - Define cut-off ranges applied to the world (and each region)
- Primarily, the user's task is choosing a “pre-packaged” physics list, that combines physics processes and models that are relevant to a typical application use-cases.
 - If “pre-packaged” physics lists do not meet your needs, you may add or alternate some processes/models.
 - If you are brave enough, you may implement your physics list.

- This is the only mandatory user action class.
- Derive your concrete class from **G4VUserPrimaryGeneratorAction** abstract base class.
- Pass a G4Event object to one or more primary generator concrete class objects which generate primary vertices and primary particles.
- Geant4 provides several generators that are all derived from G4VPrimaryParticleGenerator base class.
 - G4ParticleGun
 - G4HEPEvtInterface, G4HepMCInterface
 - Interface to /hepevt/ common block or HepMC class
 - G4GeneralParticleSource
 - Define radioactivity

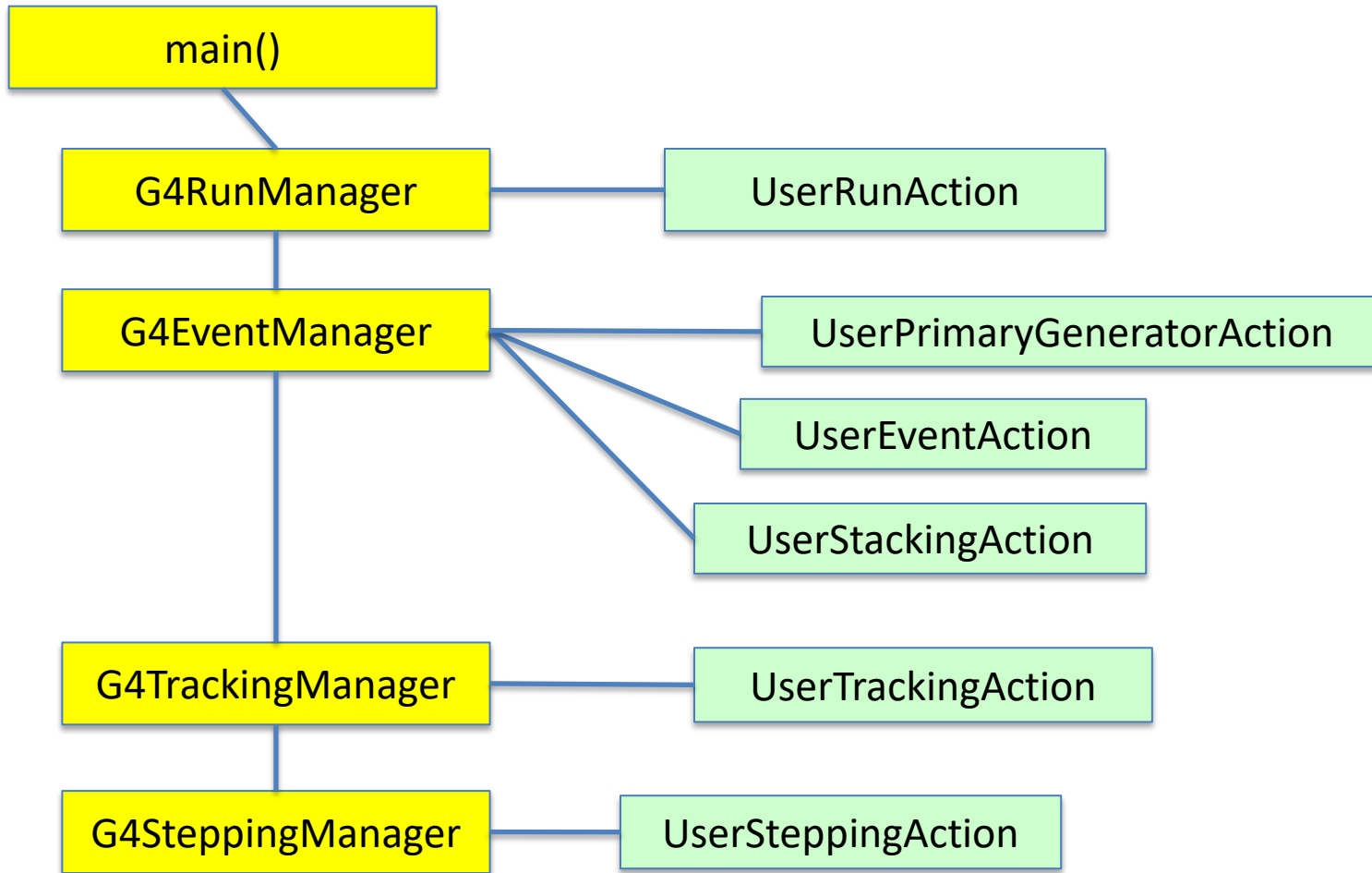
- Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”.
 - You have to do something to **extract information useful to you**.
- There are three ways:
 - Built-in scoring commands
 - Most commonly-used physics quantities are available.
 - Use scorers in the tracking volume
 - Create scores for each event
 - Create own Run class to accumulate scores
 - Assign **G4VSensitiveDetector** to a volume to generate “hit”.
 - Use user hooks (G4UserEventAction, G4UserRunAction) to get event / run summary
- You may also use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
 - You have full access to almost all information
 - Straight-forward in sequential mode, but do-it-yourself

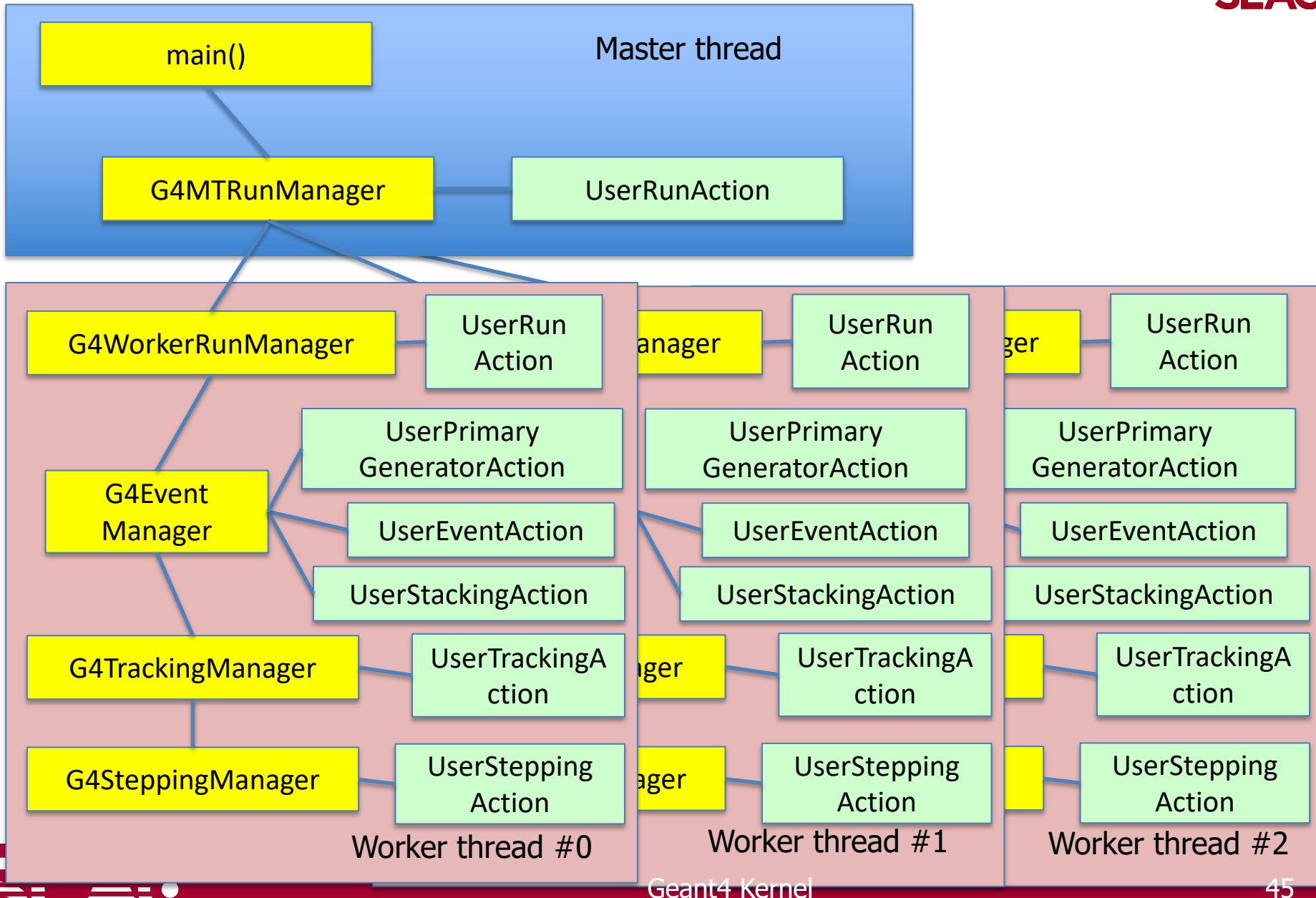
Optional user action classes

- All user action classes, methods of which are invoked during “Beam On”, must be constructed in the user’s *main()* and must be set to the RunManager.
- **G4UserRunAction**
 - G4Run* GenerateRun()
 - Instantiate user-customized run object
 - void BeginOfRunAction(const G4Run*)
 - Define histograms
 - void EndOfRunAction(const G4Run*)
 - Analyze the run
 - Store histograms
- **G4UserEventAction**
 - void BeginOfEventAction(const G4Event*)
 - Event selection
 - void EndOfEventAction(const G4Event*)
 - Output event information

- **G4UserStackingAction**
 - void PrepareNewEvent()
 - Reset priority control
 - G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)
 - Invoked every time a new track is pushed
 - Classify a new track -- priority control
 - Urgent, Waiting, PostponeToNextEvent, Kill
 - void NewStage()
 - Invoked when the Urgent stack becomes empty
 - Change the classification criteria
 - Event filtering (Event abortion)

- **G4UserTrackingAction**
 - void PreUserTrackingAction(const G4Track*)
 - Decide trajectory should be stored or not
 - Create user-defined trajectory
 - void PostUserTrackingAction(const G4Track*)
 - Delete unnecessary trajectory
- **G4UserSteppingAction**
 - void UserSteppingAction(const G4Step*)
 - Kill / suspend / postpone the track
 - Draw the step (for a track not to be stored as a trajectory)





- **G4VUserActionInitialization** has two virtual methods.
- *Build()*
 - Invoked at the beginning of each worker thread as well as in sequential mode
 - Use *SetUserAction()* method to register pointers of all user actions.
 - In multithreaded mode, all user action class objects instantiated in this method are thread-local.
 - User run action instantiated in this method is for thread-local run
- *BuildForMaster()*
 - Invoked only at the beginning of the master thread in multithreaded mode
 - Use *SetUserAction()* method to register pointer of user run action for the global run.

Let me remind you...

- Define material and geometry
 - ➔ G4VUserDetectorConstruction
- Select appropriate particles and processes and define production threshold(s)
 - ➔ G4VUserPhysicsList
- Instantiate user action classes
 - ➔ G4VUserActionInitialization
- Define the way of primary particle generation
 - ➔ G4VUserPrimaryGeneratorAction
- Define the way to extract useful information from Geant4
 - ➔ Built-in scorer and analysis UI commands
 - ➔ G4VUserDetectorConstruction, G4UserEventAction, G4Run, G4UserRunAction
 - ➔ G4SensitiveDetector, G4VHit, G4VHitsCollection