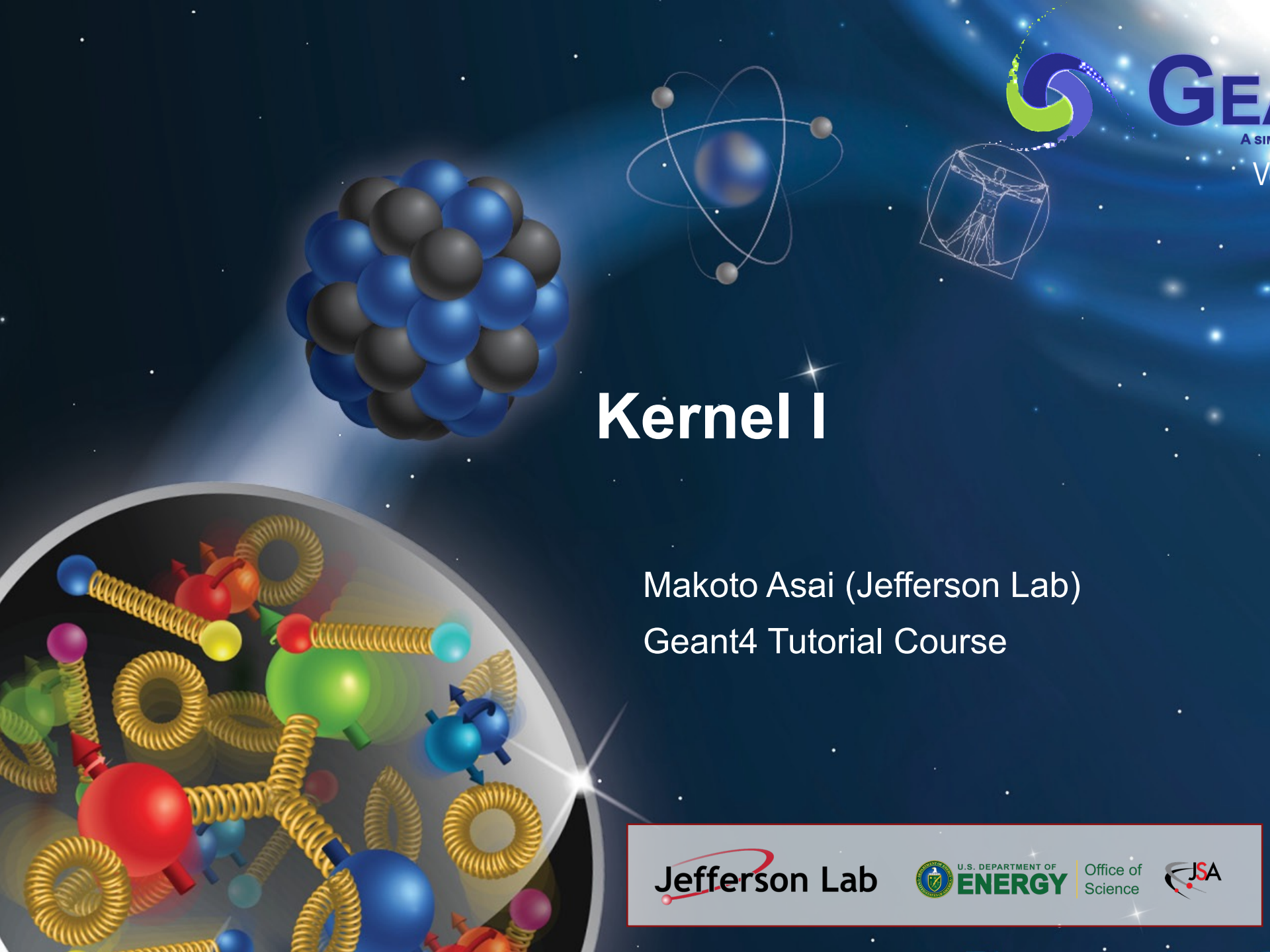


Kernel I

Makoto Asai (Jefferson Lab)

Geant4 Tutorial Course

Jefferson Lab
U.S. DEPARTMENT OF ENERGY | Office of Science
JSA

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- General introduction and brief history
- Geant4 kernel
  - Basic concepts and kernel structure
  - User classes
- Introduction to multithreading

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- General introduction and brief history
- Geant4 kernel
  - Basic concepts and kernel structure
  - User classes
- Introduction to multithreading

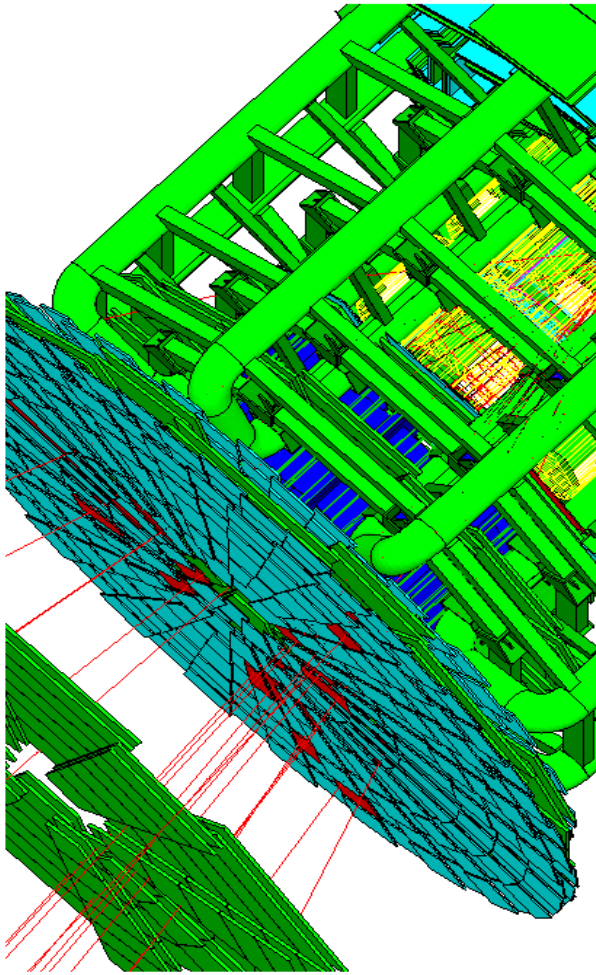
GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science

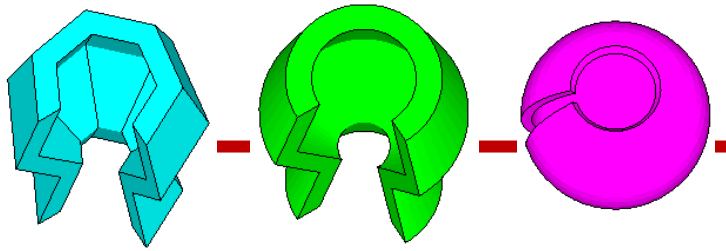
JSA

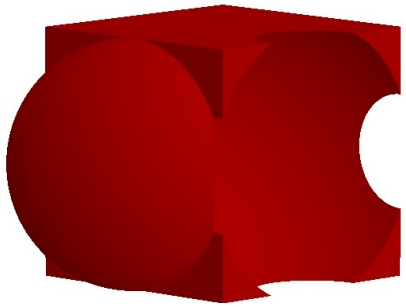
Key Geant4 functionalities

- Geant4 is a general purpose Monte Carlo simulation tool for elementary particles passing through and interacting with matter. It finds quite a wide variety of user domains including high energy and nuclear physics, space engineering, medical applications, material science, radiation protection and security.
- Geant4 offers most, if not all, of the functionalities required for the simulation of elementary particle and nucleus passing through and interacting with matter.
  - Kernel
  - Geometry and navigation
  - Physics processes
  - Scoring
  - GUI and Visualization drivers
- Thanks to the polymorphism mechanism of C++, the users can easily plug-in their extensions without interfering with the other part of Geant4.
- Extensive user guide documents and examples are provided.

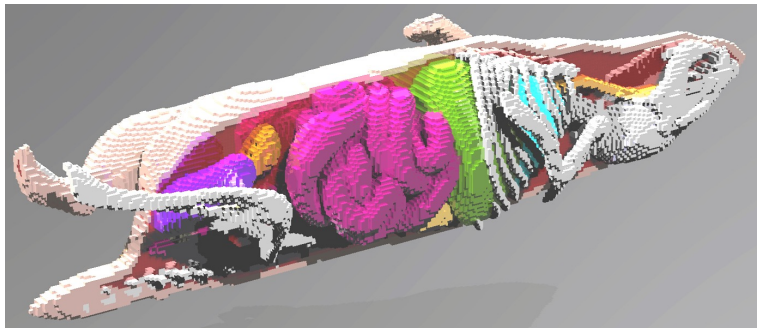


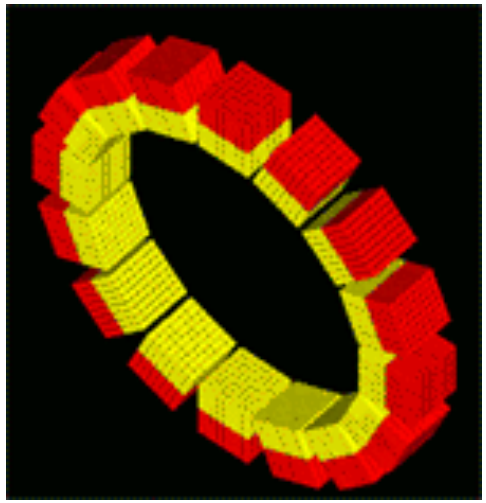
Key geometry capabilities





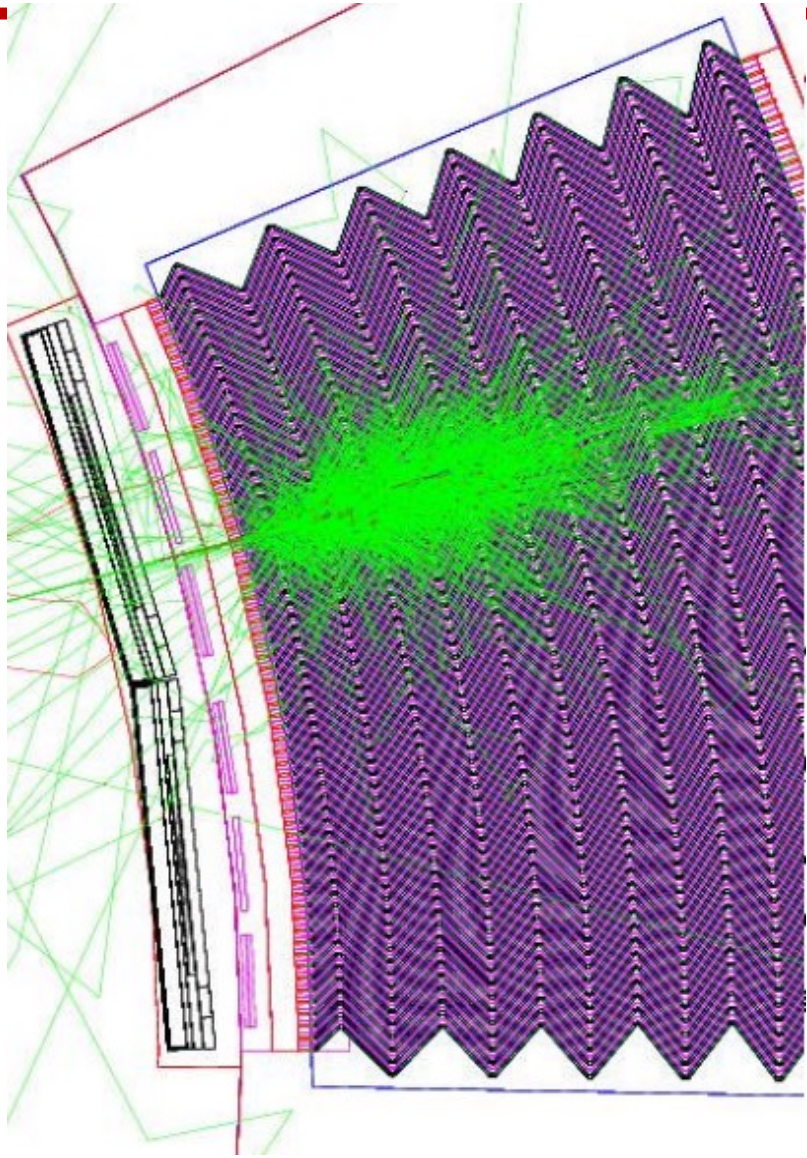
- Richest collection of shapes
  - CSG (Constructed Solid Geometry), Boolean operation, Tessellated solid, etc.
  - The user can easily extend
- Describing a setup as hierarchy or ‘flat’ structure
  - Describing setups up to billions of volumes
  - Tools for creating & checking complex structures
  - Interface to CAD
- Navigating fast in complex geometry model
  - Automatic optimization
- Geometry models can be ‘dynamic’
  - Changing the setup at run-time, e.g. “moving objects”



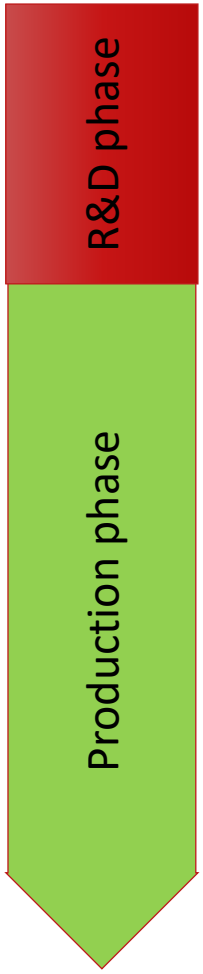


Physics models in Geant4

- Geant4 offers
  - Electromagnetic processes
  - Hadronic and nuclear processes
  - Photon/lepton-hadron processes
  - Optical photon processes
  - Decay processes
  - Shower parameterization
  - Event biasing techniques
  - And you can plug-in more
- Geant4 provides sets of alternative physics models so that the user can freely choose appropriate models according to the type of his/her application.
  - For example, some models are more accurate than others at a sacrifice of speed.



Geant4 History

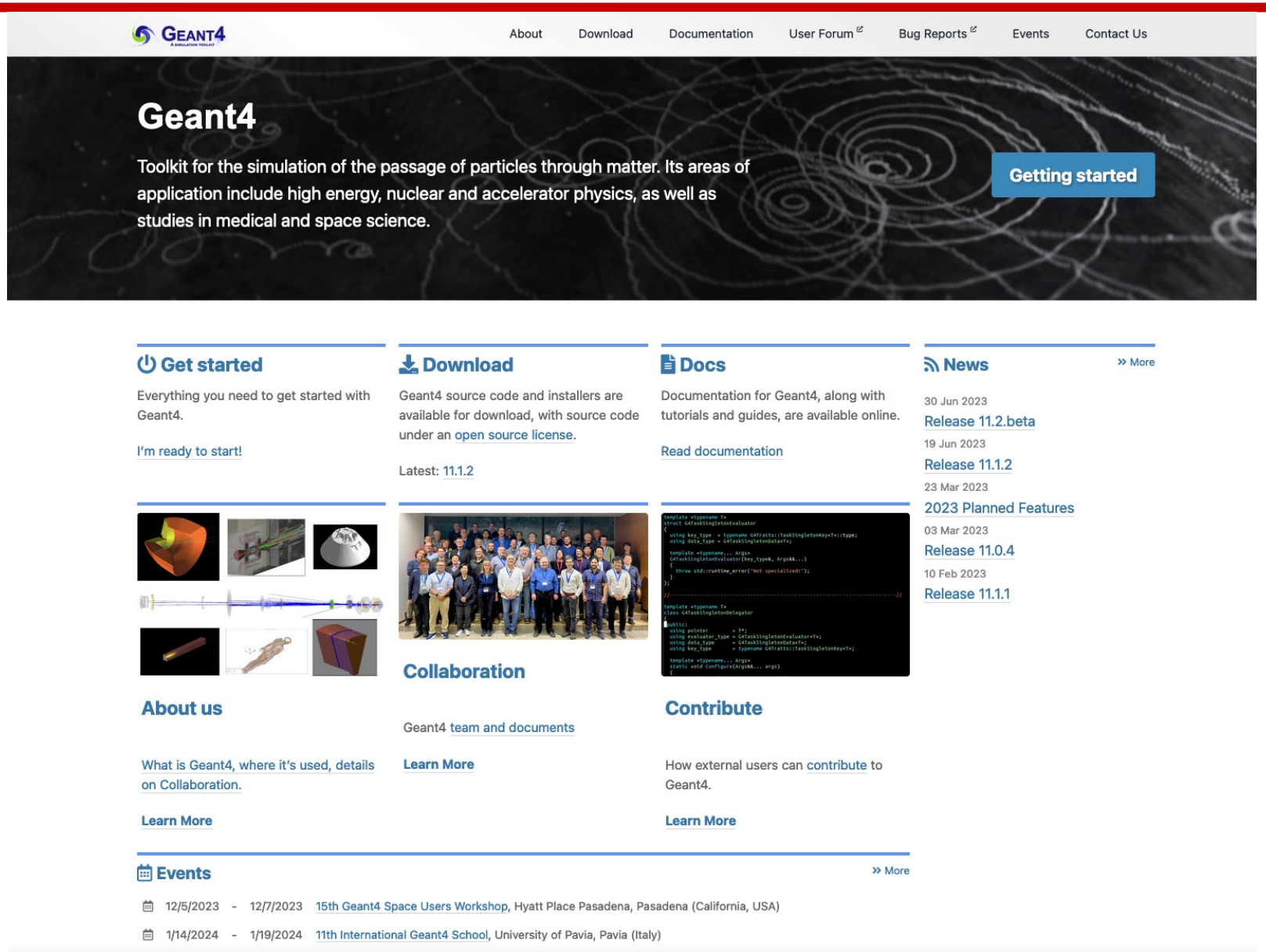


- Early discussions even before having reliable C++ compilers.
- Dec ’94 – R&D project start
- Apr ’97 - First alpha release
- Jul ’98 - First beta release
- Dec ’98 - First Geant4 public release - version 1.0

- Several major architectural revisions
  - E.g. STL migration, “cuts per region”, parallel worlds, command-based scorer, **multithreading, task-based parallelization**

- Dec 9th, ’23 – Geant4 version 11.1 release
  - Jun 12th, ’22 - Geant4 11.1-patch02 release ← Current version
- We currently provide one public release every year.
  - Next version - Geant4 11.2 (planned on Dec 8th, 2023)

https://geant4.org/



Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- General introduction and brief history
- Geant4 kernel
  - Basic concepts and kernel structure
  - User classes
- Introduction to multithreading

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

Terminology (jargons)

- Run, event, track, step, step point
- Track ←→ trajectory, step ←→ trajectory point
- Process
  - At rest, along step, post step
- Cut = production threshold
- Sensitive detector, score, hit, hits collection,

Run in Geant4

- 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.

Event in Geant4

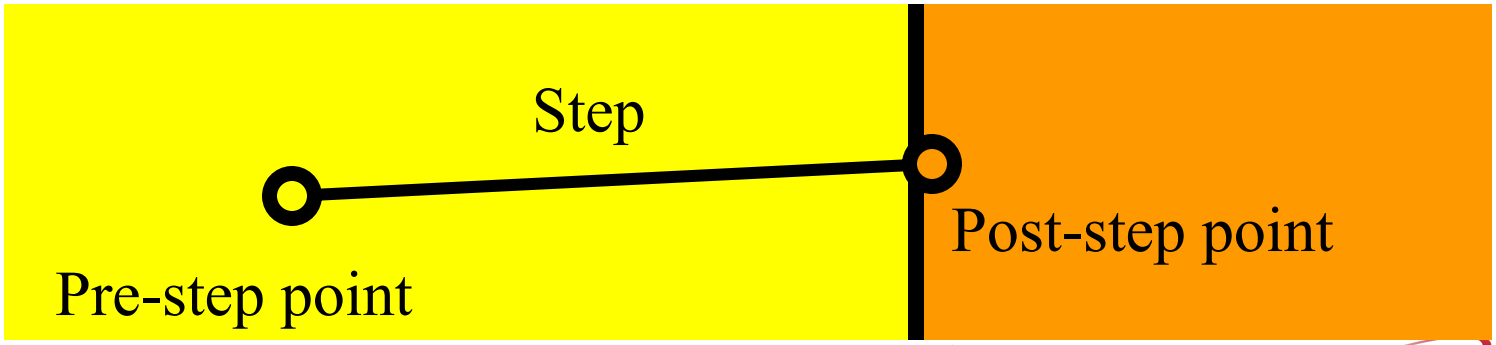
- An event is the basic unit of simulation in Geant4.
- At beginning of processing, primary tracks are generated. These primary tracks are pushed into a stack.
- A track is popped up from the stack one by one and "tracked". Resulting secondary tracks are pushed into the stack.
  - This "tracking" lasts 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 in Geant4

- 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 in Geant4

- 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.



Trajectory and trajectory point

- 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 ←→ G4Trajectory, G4Step ←→ 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.

Particle in Geant4

- 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.

Tracking and processes

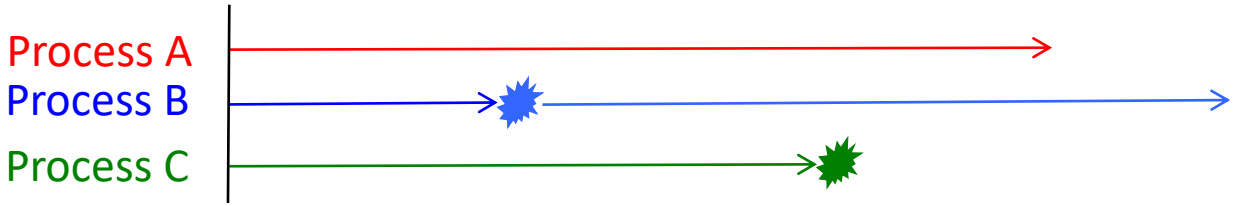
- Geant4 tracking is general.
  - It is independent to
    - the particle type
    - the physics processes involving to a particle
  - It gives the chance to all processes
    - To contribute to determining the step length
    - To contribute any possible changes in physical quantities of the track
    - To generate secondary particles
    - To suggest changes in the state of the track
      - e.g. to suspend, postpone or kill it.

Processes in Geant4

- In Geant4, “process” is an abstract concept that may affect any of the track data.
  - Momentum, energy, position, secondary generation, fate of the track, etc.
- 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 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. Celenkov 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.



Cuts in Geant4

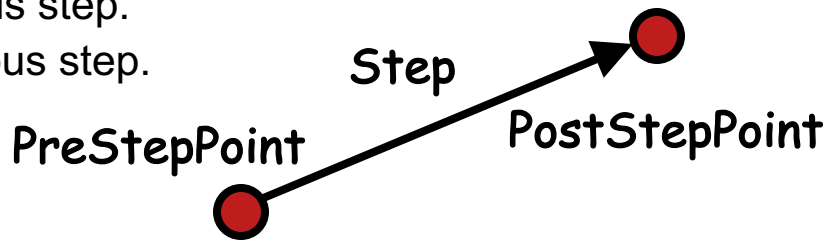
- A Cut in Geant4 is a production threshold.
  - Not tracking cut, which does not exist in Geant4 as default.
    - All tracks are traced down to zero kinetic energy.
  - It is applied only for physics processes that have infrared divergence
- Much detail will be discussed at later talks on physics.

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
  - fAtRestDoItProc, fAlongStepDoItProc, fPostStepDoItProc
    - 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

Extract useful information

- 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

Unit system

- 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;
```

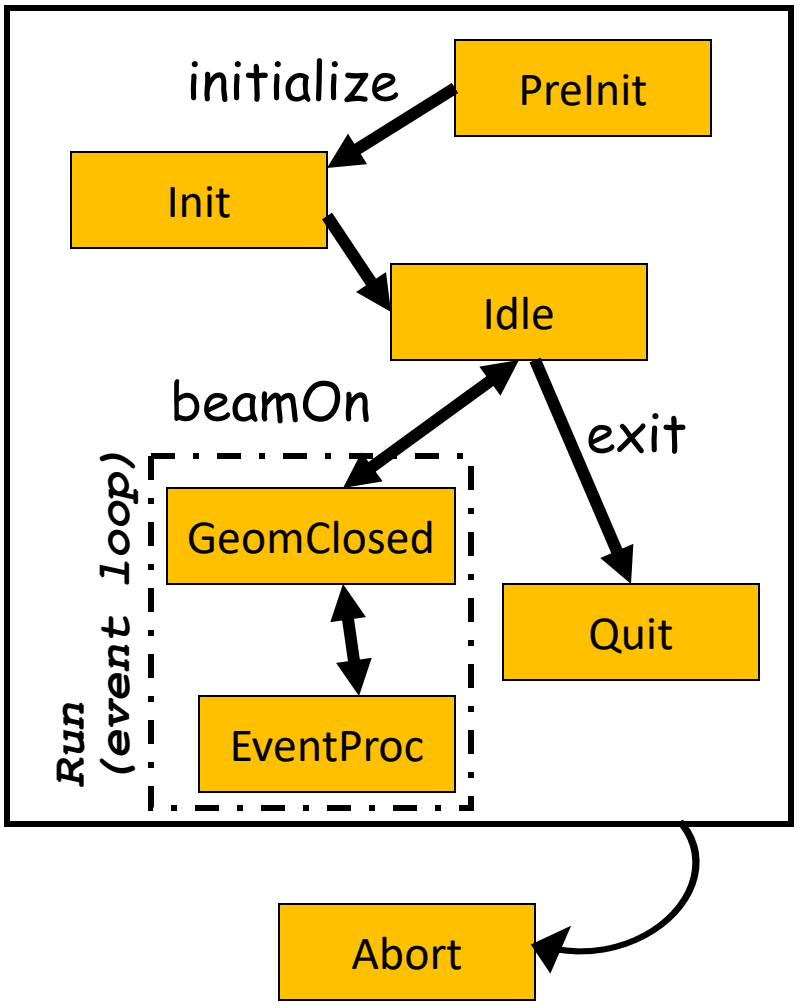
- To get a number, it must be divided by a proper unit.

```
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.

Geant4 as a state machine

- 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.

G4cout, G4cerr

- 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.

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- General introduction and brief history
- Geant4 kernel
  - Basic concepts and kernel structure
  - User classes
- Introduction to multithreading

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

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.

User classes

- 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.

The main program

- Geant4 does not provide a *main*().
- In your *main(),* you have to
  - Construct G4RunManager
    - G4MTRunManager (multithreaded mode)
    - G4TaskingRunManager (tasking mode – default)
  - 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()*.

Describe your detector

- 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.

Select physics processes

- 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.

Generate primary event

- 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 in addition to the G4VPrimaryParticlegenerator base class.
  - G4ParticleGun
  - G4HEPEvtInterface, G4HepMCInterface
    - Interface to /hepevt/ common block or HepMC class
  - G4GeneralParticleSource
    - Define radioactivity

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

Optional user action classes

- 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)

Optional user action classes

- 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

Instantiate user action classes

- 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
    Material and Geometry lectures
- Select appropriate particles and processes and define production threshold(s)
  - ➔ G4VUserPhysicsList
    Physics lectures
- Instantiate user action classes
  - ➔ G4VUserActionInitialization
    Hands-on
- Define the way of primary particle generation
  - ➔ G4VUserPrimaryGeneratorAction
    Primary particle lecture
- Define the way to extract useful information from Geant4
  - ➔ G4VUserDetectorConstruction, G4UserEventAction, G4Run, G4UserRunAction
  - ➔ G4SensitiveDetector, G4VHit, G4VHitsCollection
    Scoring lectures

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- General introduction and brief history
- Geant4 kernel
  - Basic concepts and kernel structure
  - User classes
- Introduction to multithreading

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

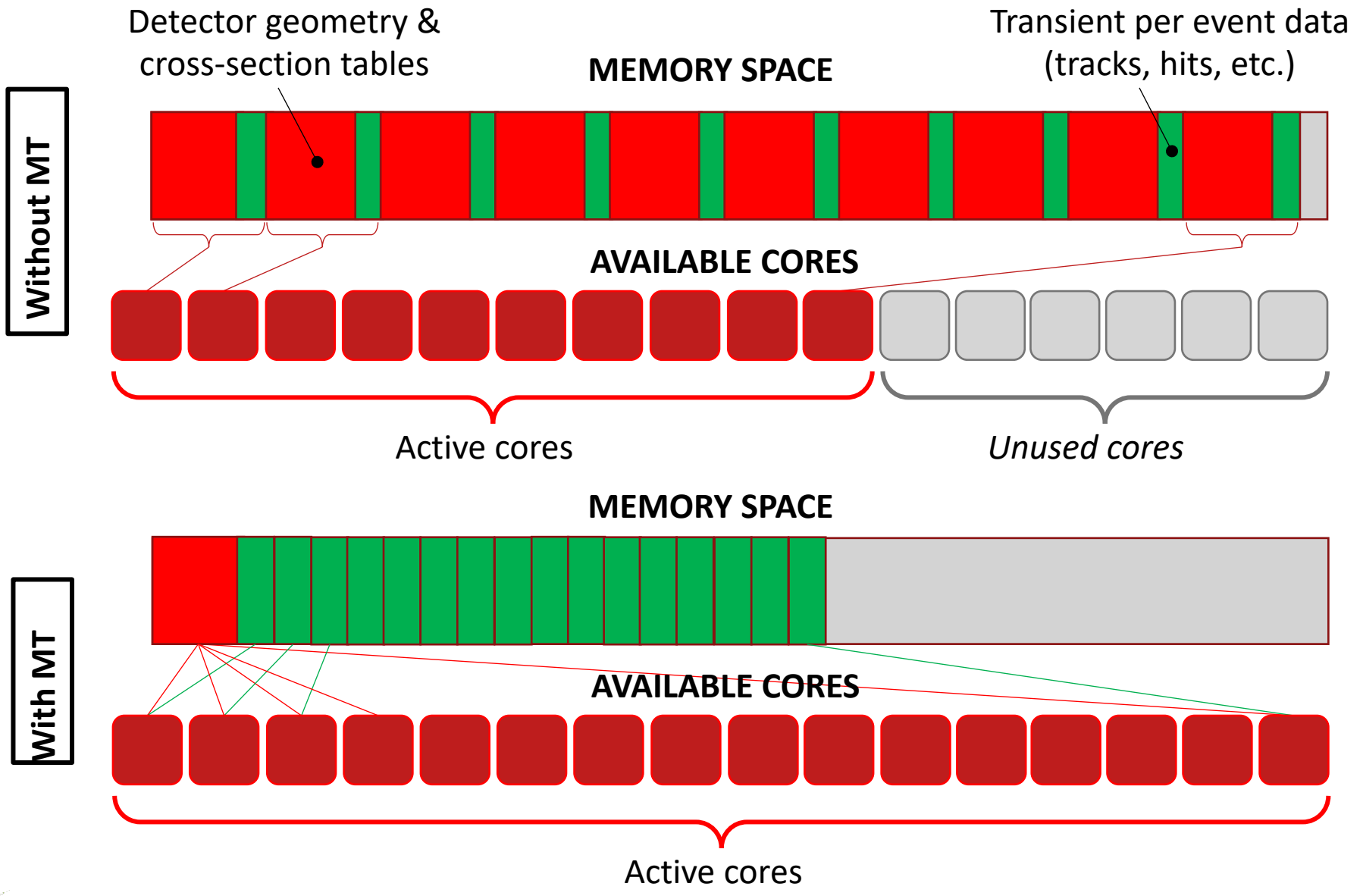
Geant4 evolutions in parallelization

1. Sequential mode : original since Geant4 v1.0
   - Single core (thread) does everything

2. Multithreaded event-level parallelism mode : since Geant4 v10.0 (Dec.2013)
   - Taking the advantage of independence of events, many cores (threads) process events in parallel (event-level parallelism)
   - Geometry / x-section tables are shared over threads

3. Task-based event-level parallelism mode : since Geant4 v11.0 (Dec.2021)
   - Decoupling task (event loop) from thread
   - More flexible load-balancing

4. Task-based sub-event parallel mode : planned (Dec.2023~)
   - Split an event into sub-events and task them separately
   - Sub-event :
     - Sub-group of primary tracks, or
     - Group of tracks getting into a particular detector component
       - Suitable for heterogeneous hybrid hardware


Without MT
Detector geometry &
cross-section tables
MEMORY SPACE
Transient per event data
(tracks, hits, etc.)
AVAILABLE CORES
Active cores
Unused cores
With MT
MEMORY SPACE
AVAILABLE CORES
Active cores

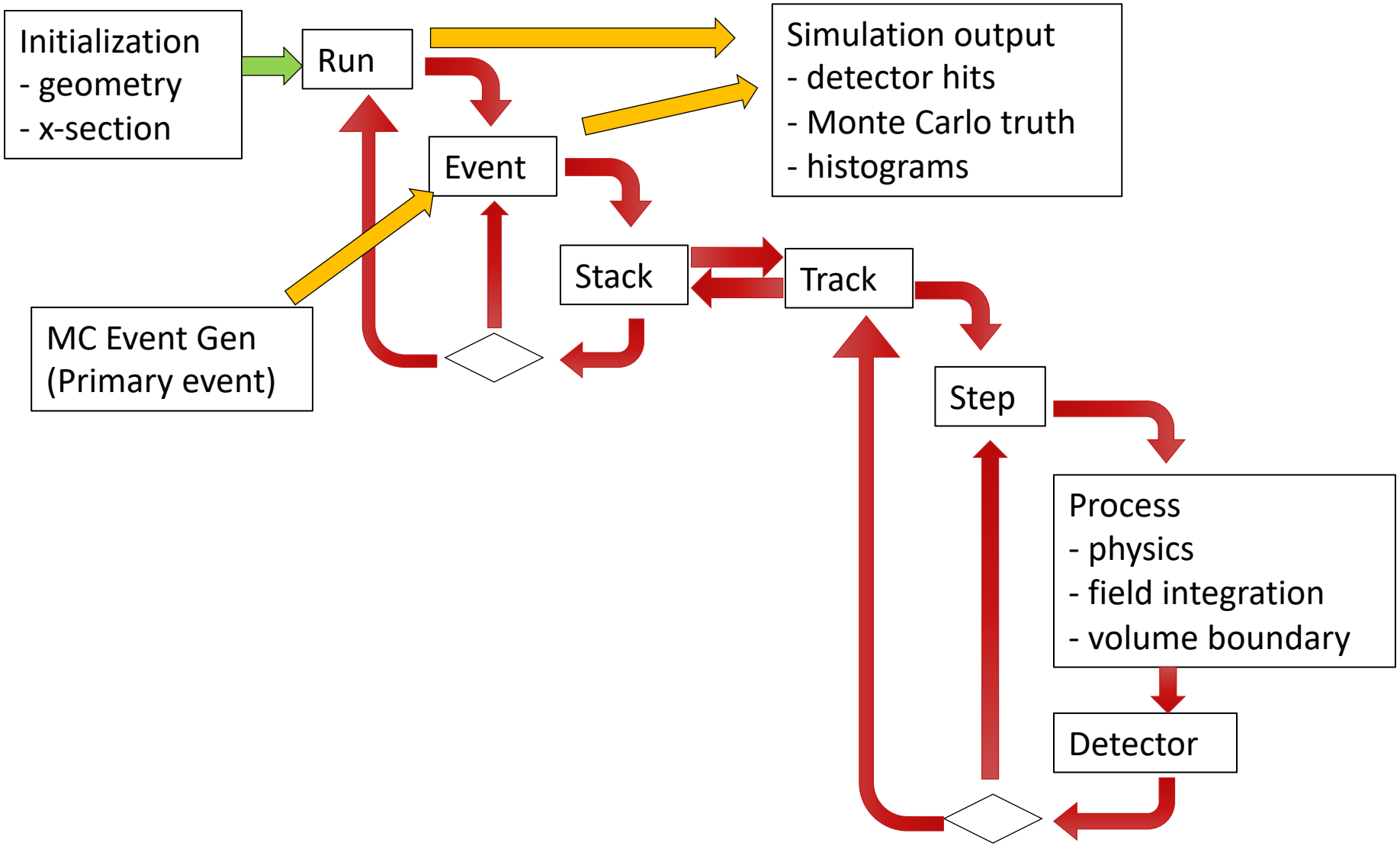
G4RunManagerFactory

- Since version 11.0, Geant4 introduced task-based event parallelism.
- There are four different running modes available
  - Sequential mode
  - Multithreaded mode by Posix thread
    - was default in version 10 series
  - Task-based multithread mode by PTL (Parallel Tasking Library)
    - default
  - Task-based multithread mode by Intel TBB (Threading Building Blocks)
- RunManager can be instantiated by G4RunManagerFactory with preferred threading option.

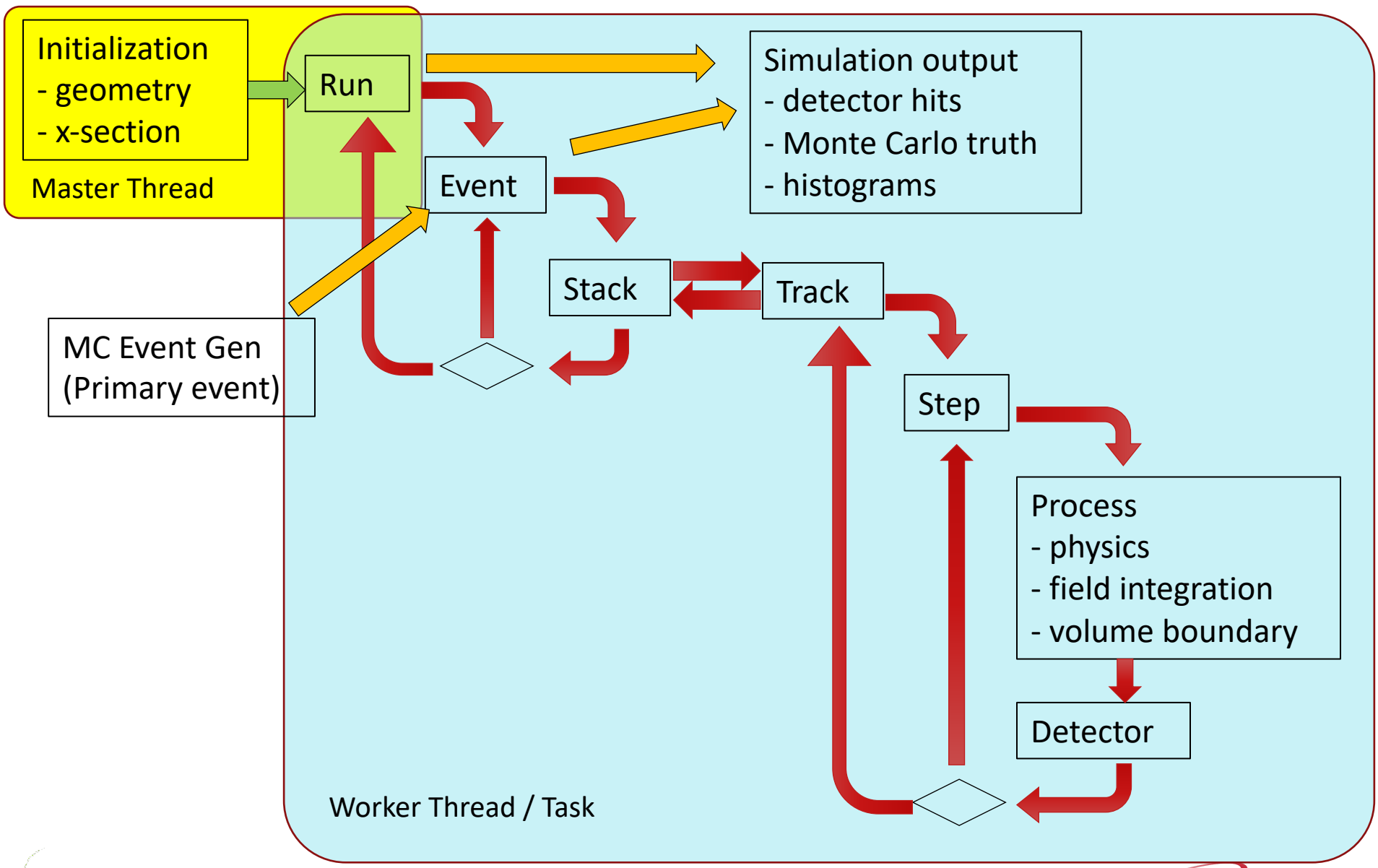
```
auto* runManager =
      G4RunManagerFactory::CreateRunManager();
```

- Type of RunManager can be specified by the shell variable *$G4RUN_MANAGER_TYPE*
  - *Serial, MT, Threading, TBB*
- You may still use the explicit constructors such as G4MTRunManager.

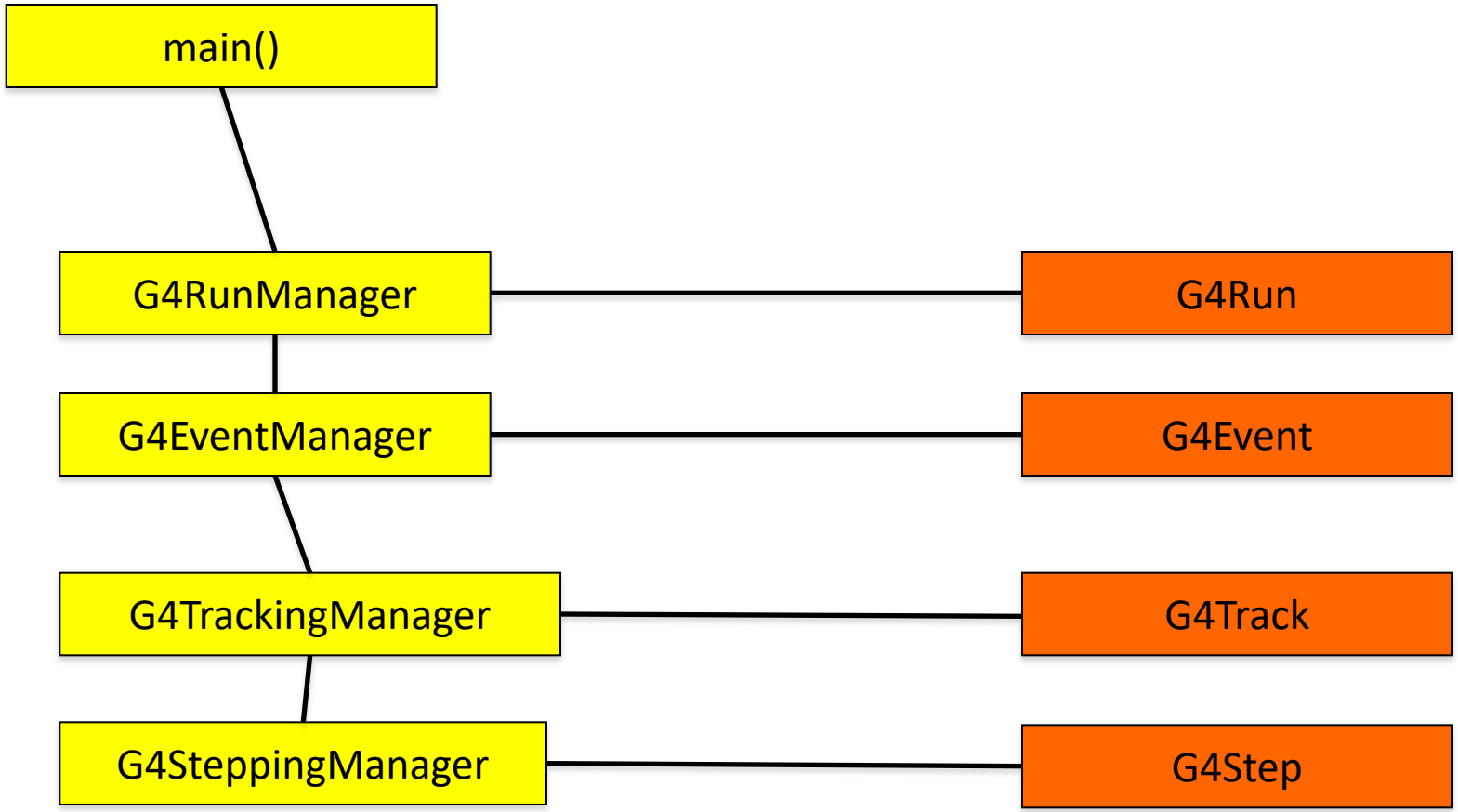
Geant4 as a detector simulation engine



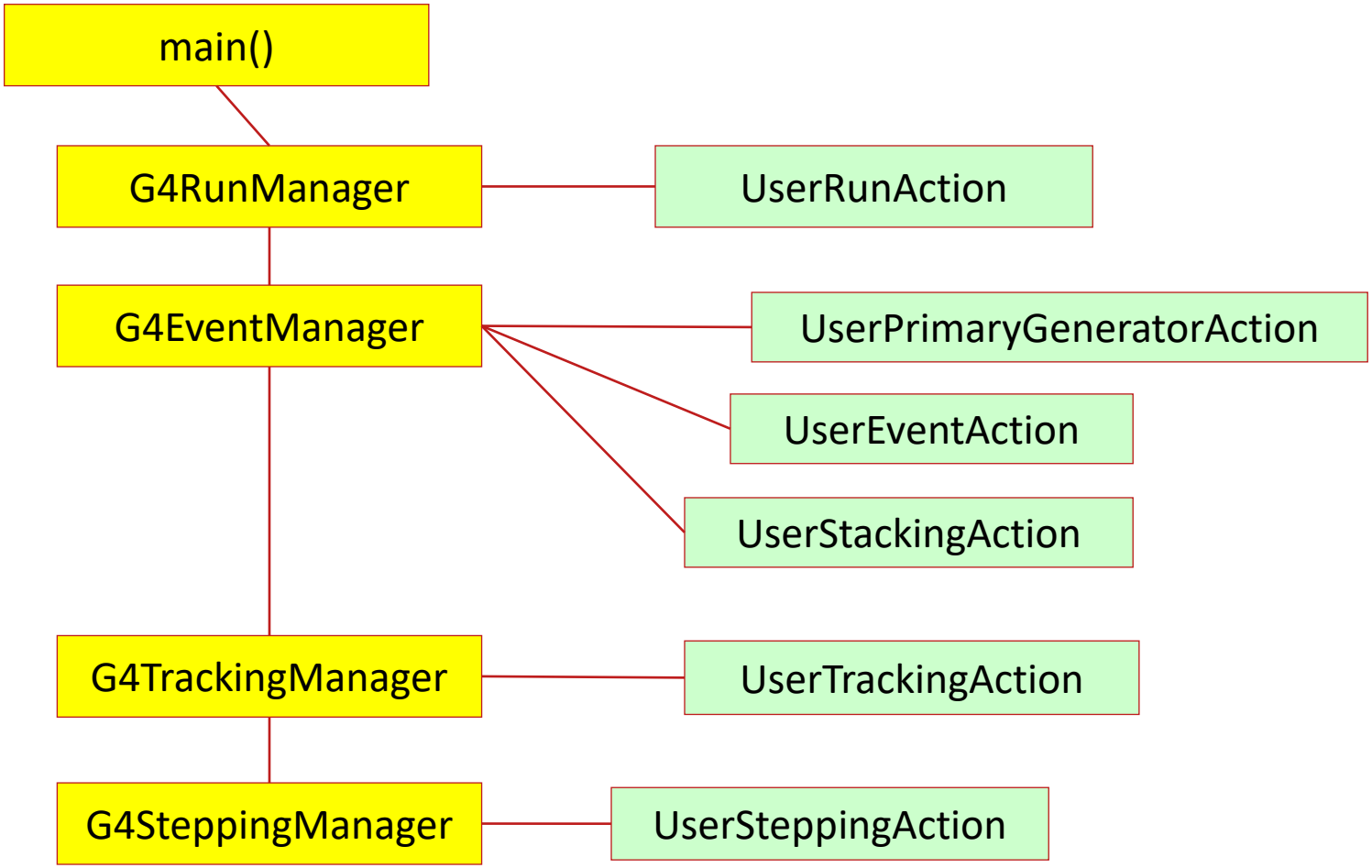
Geant4 as a detector simulation engine



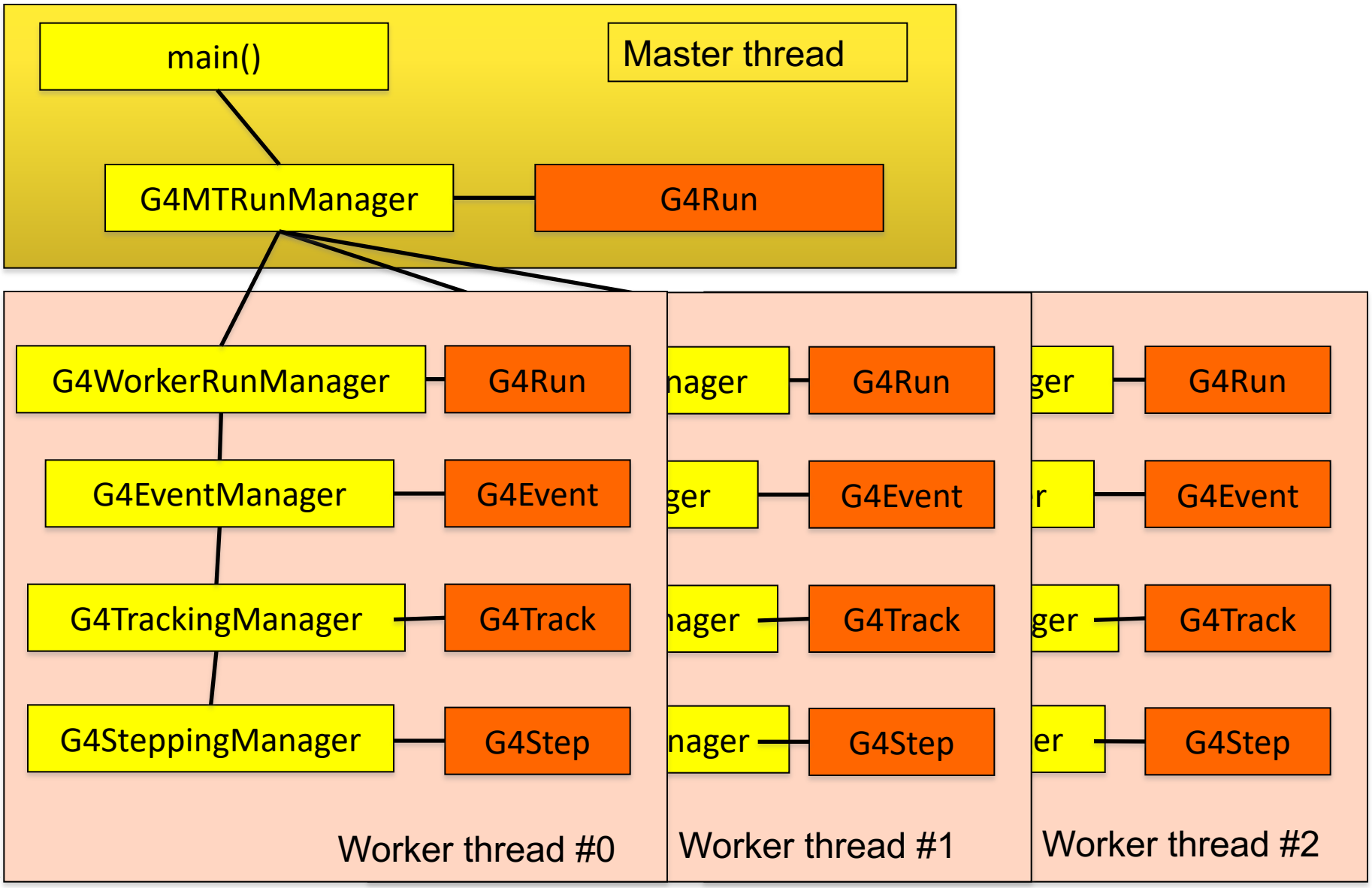
Sequential mode



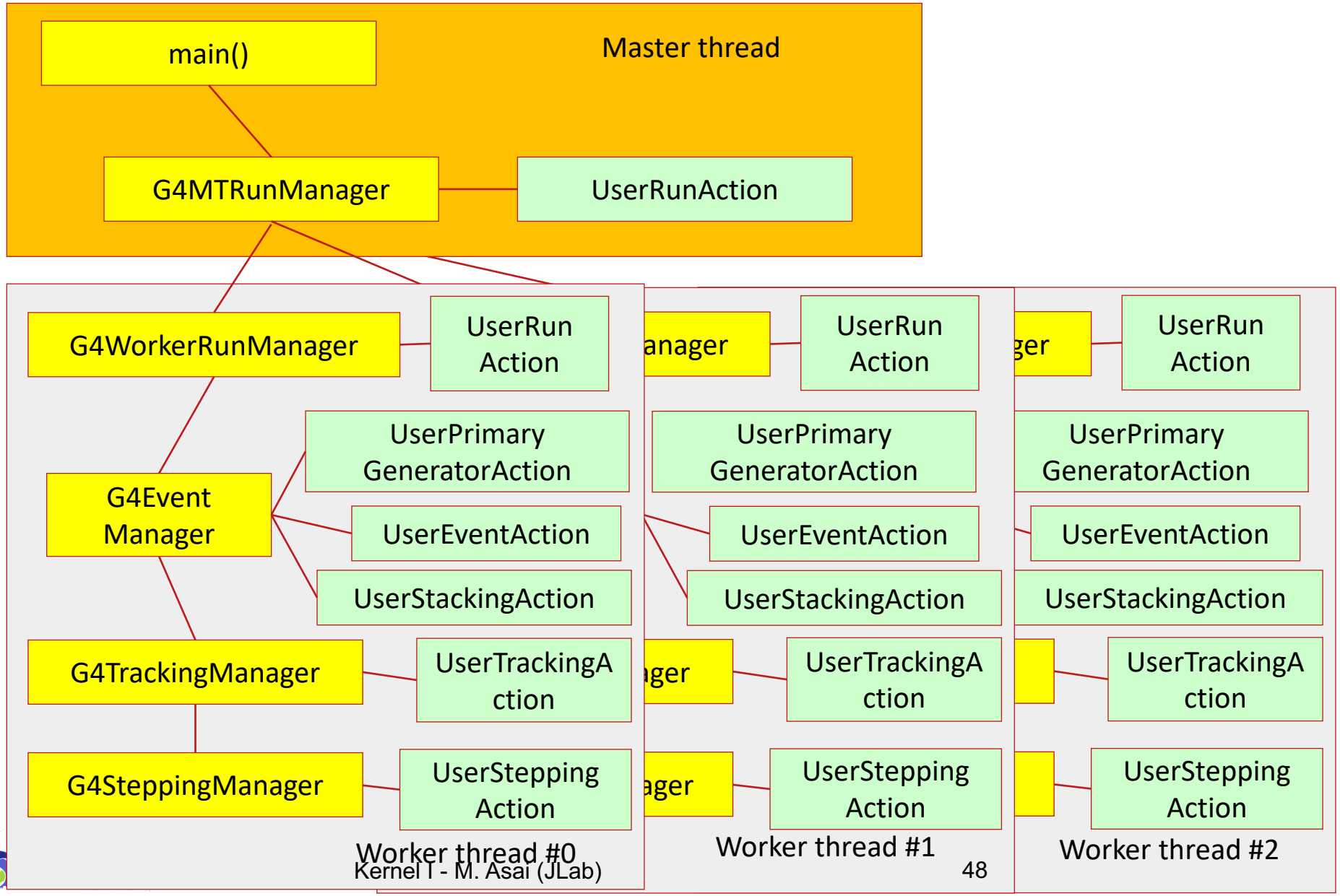
Sequential mode



Multi-threaded mode



Multi-threaded mode



Shared? Thread-local?

- 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.
- 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.

G4cout in multithreaded mode

- By default, every G4cout string is displayed on the screen in the order as it is generated.
  - A line made by a worker thread is preceded by the worker identifier.
- It is not very readable if lines of several worker threads interleave.

  /control/cout/ignoreThreadsExcept <threadID>
  - Omit cout from worker threads except the specified one.
  - If specified thread ID is greater than the number of threads, no cout is displayed from worker threads. -1 to reset.

  /control/cout/useBuffer <true/false>
  - Send cout stream to a buffer dedicated to each worker thread.
  - The buffered text will be printed at the end of the job for each thread at a time, so that output of each thread is grouped.

  /control/cout/setCoutFile <fileName> <appendFlag>
  - Send G4cout stream to a file dedicated to a thread.
  - If append flag is true output is appended to the existing file, otherwise file output is overwritten.
  - To return to a display output, use special file name "**Screen**".

UI command in multithreaded mode

- In multithreaded mode, each thread (both master and worker) has its own G4UImanager object.
  - Commands instantiated in a thread are registered to the G4UImanager of that thread.
- Every UI command is executed in the master thread if the command is issued in the master thread (including interactive session or macro file), and then broadcasted to each worker thread prior to the beginning of thread-local event loop (i.e. at the time of */run/beamOn*).
  - If the user wants to broadcast command(s) immediately, use */run/workersProcessCmds* command.
  - If a command should not be distributed to worker threads, *SetToBeBroadcasted(false)* should be set for that command.
  - If a command is hard-coded in an object in a worker thread, it is executed only in that worker thread. It won't be transmitted to other worker threads.