Updates on Sub-Event Parallelism

Makoto Asai (JLab/CST)
asai@jlab.org

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

Geant4 evolutions in parallelization

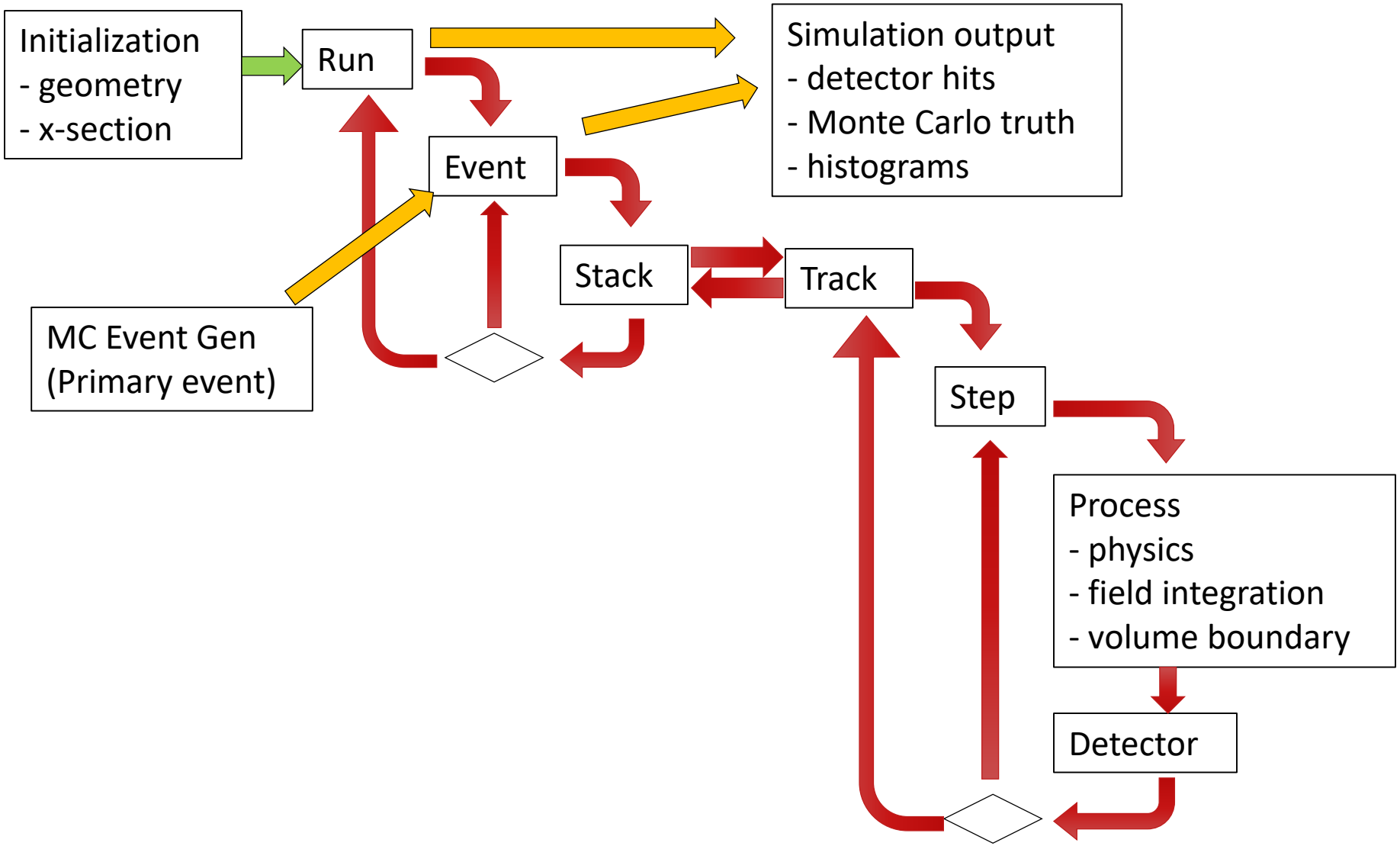
1. Sequential mode : original since Geant4 v1.0
   - Single core (thread) does everything
2. Multithreaded event-level parallel mode : since Geant4 v10.0 (Dec.2013)
   - Taking the advantage of independence of events, many cores (threads) process events in parallel (event-level parallelism)
   - Geometry / x-section tables are shared over threads
3. Task-based event-level parallel 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.2024~)
   - 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

- N.B. We made these evolutions without forcing the user to migrate
  - Except for using the new functionalities

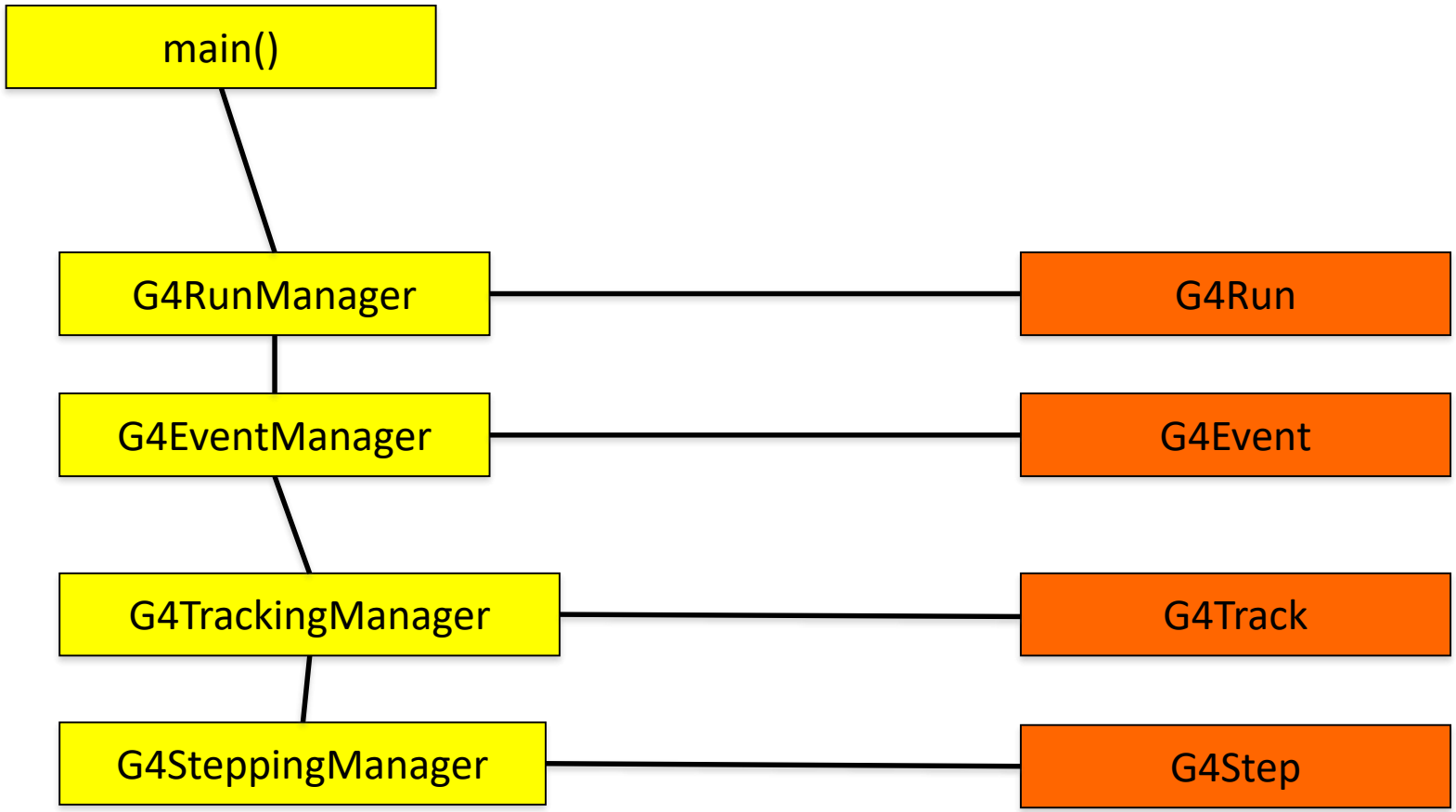
Sub-event parallel mode

- Split an event into sub-events and task them separately to worker threads.
- Two major use-cases:
  - CPU only
    - Ultra-high energy event that takes long time, e.g. air shower
    - Event-level parallelism does not help here.
  - Heterogeneous hardware
    - If several different types of accelerations are available, e.g. optical photon, EM physics, etc.
    - Different kind of sub-events could be tasked to different worker threads dedicated to their applicabilities.

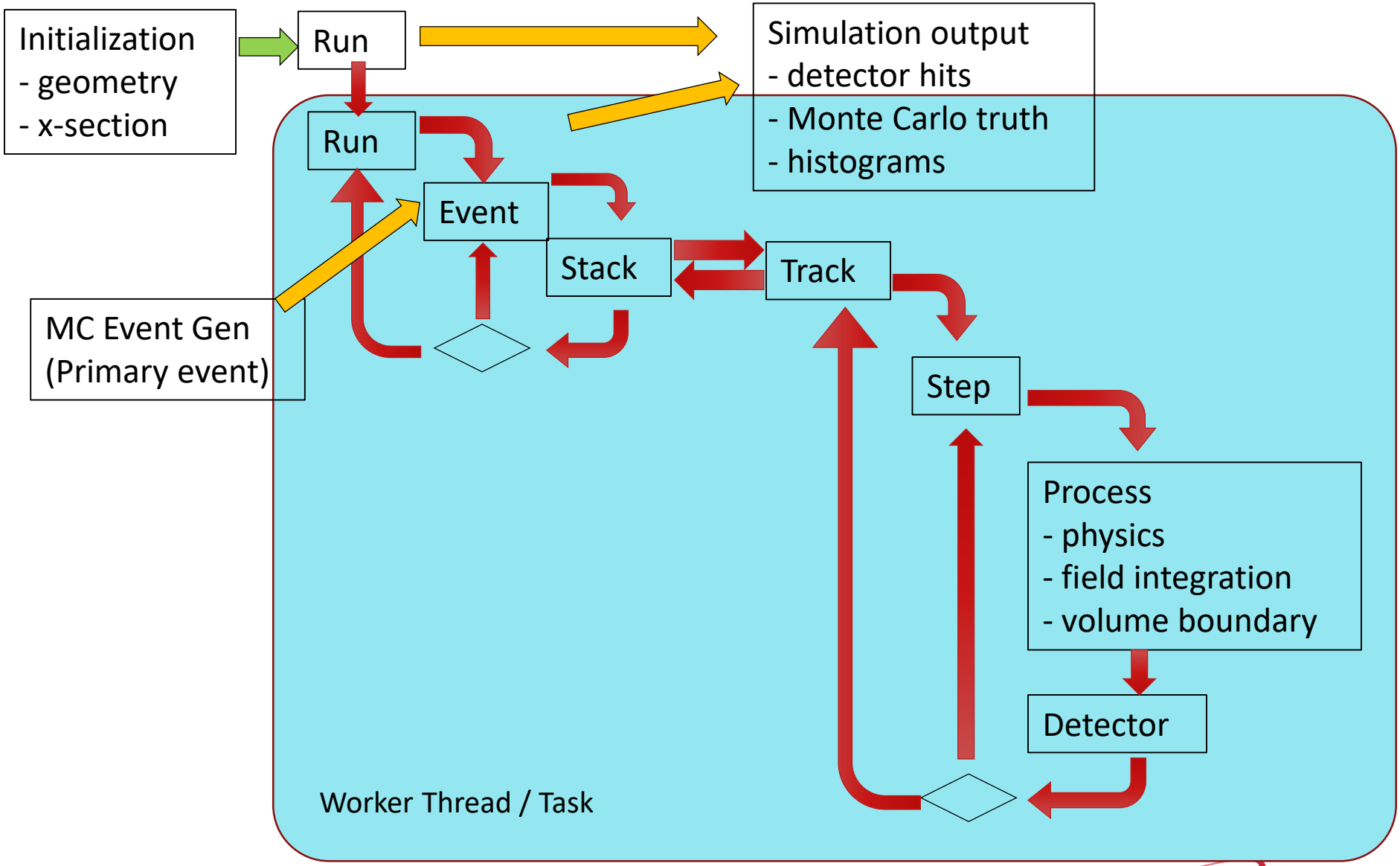
Geant4 as a detector simulation engine



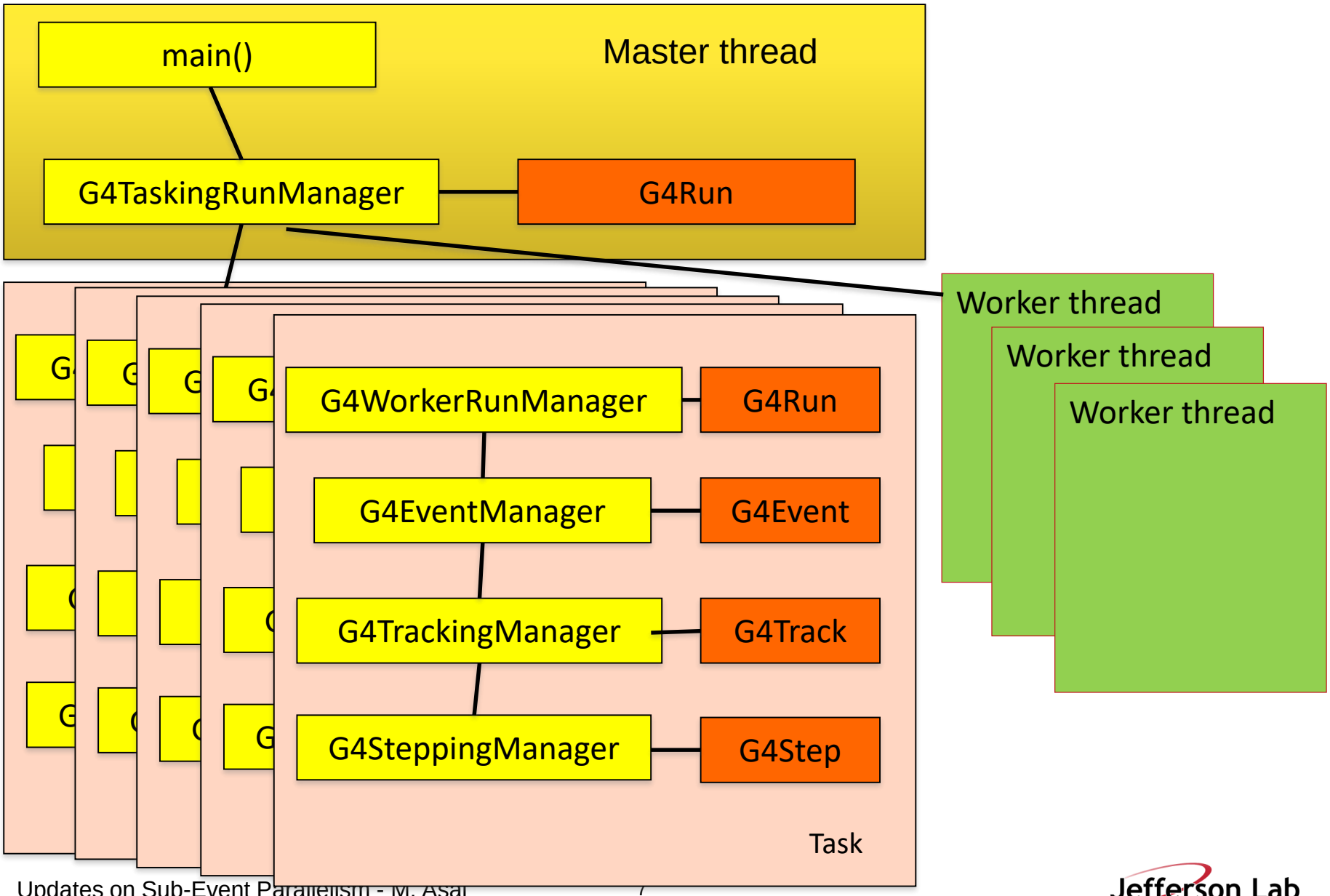
Sequential mode



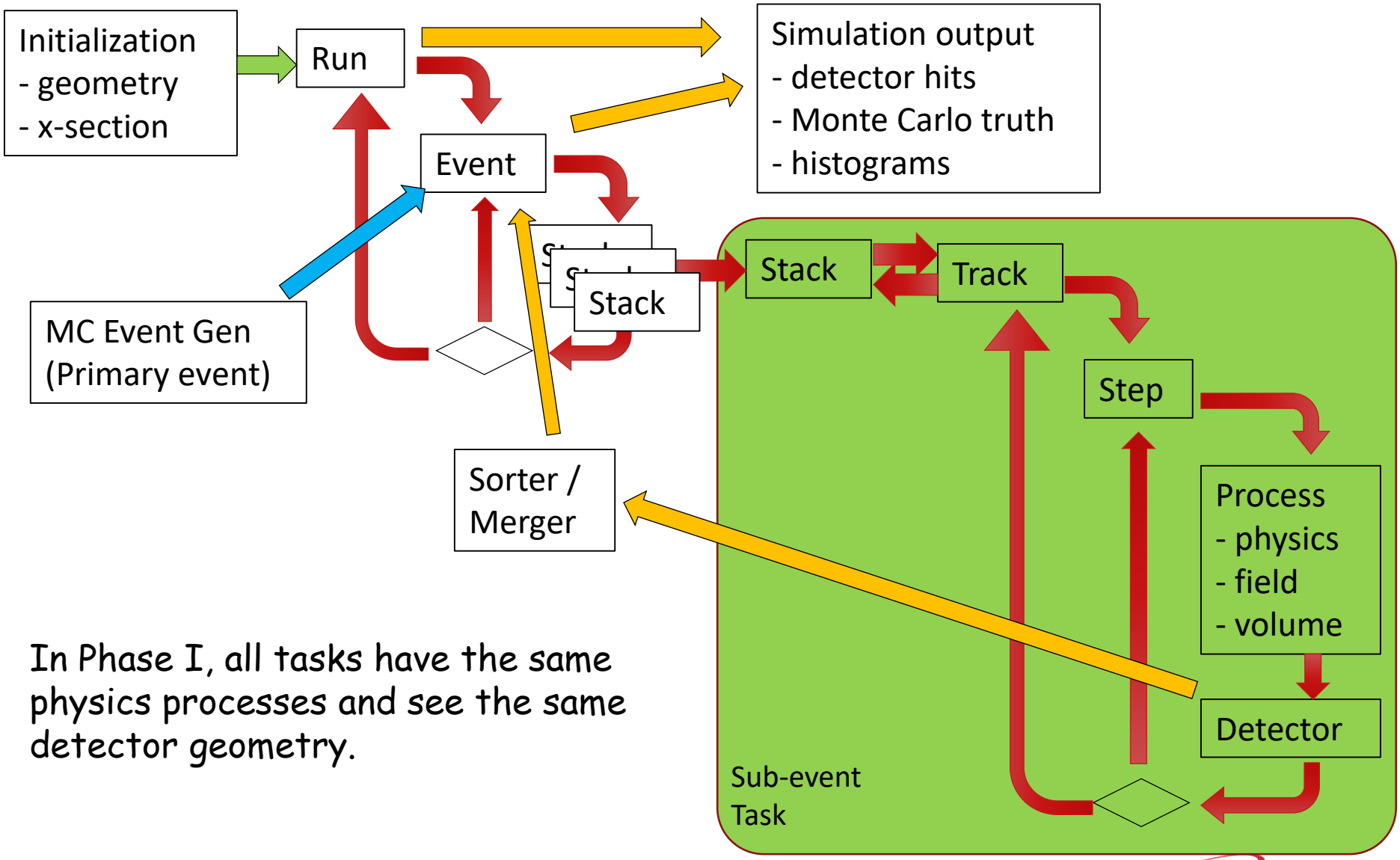
Event-level parallel mode (thread / task)



Task-based parallel mode

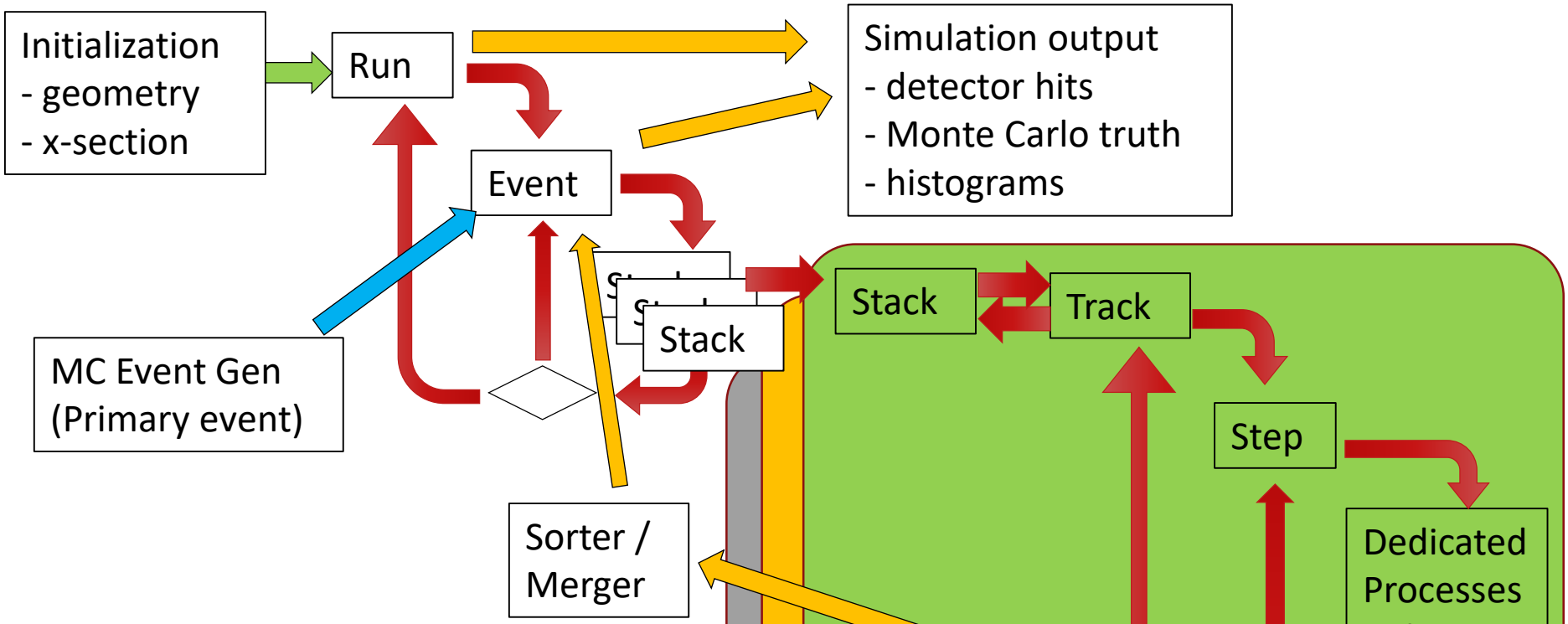


Task-based sub-event parallel mode (Phase I)



In Phase I, all tasks have the same physics processes and see the same detector geometry.

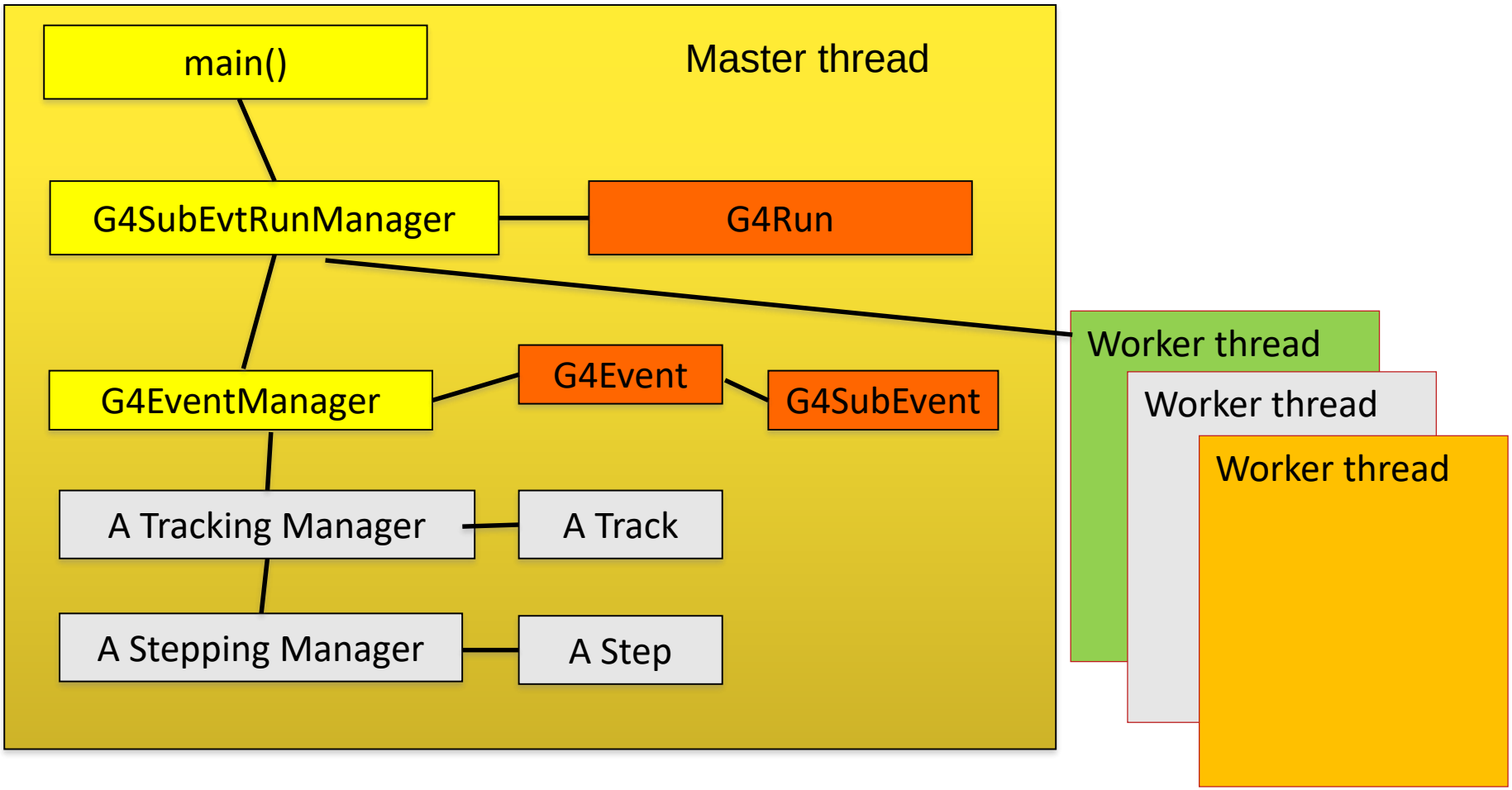
Task-based sub-event parallel mode (Phase II)



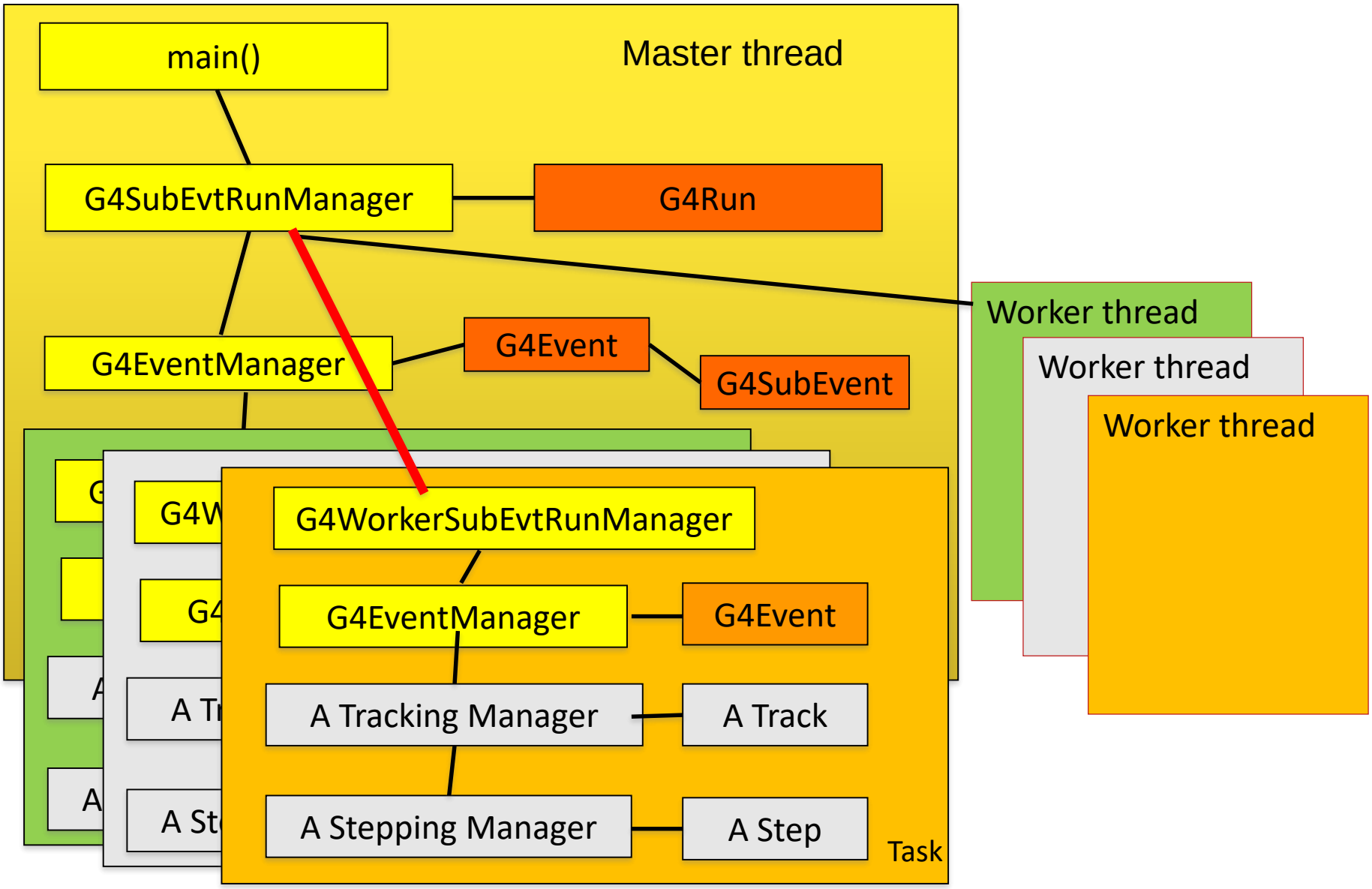
In Phase II, each task has only the necessary physics processes and sees the limited detector geometry that are necessary for that particular task.

- Essential for heterogeneous simulation

Sub-event parallel mode



Sub-event parallel mode



What is / will be unchanged / seamless

- Applications that work in the event-level parallel mode both with original MT mode or with the task-based MT mode work as-is. No modification needed.
- In the sub-event parallel mode:
  - Methods in *G4EventManager* that are used in the worker thread are not changed.
    - *ProcessOneEvent(G4TrackVector*)* method is used instead of *ProcessOneEvent(G4Event*).*
  - G4TrackingManager is unchanged.
  - “Specialized” tracking manager can be used as-is in the worker thread. It can be assigned to all worker threads or to some selected number of worker threads.
    - May have more than one kinds of specialized tracking managers.
  - Scores defined by the command-based scorers are automatically merged.
  - Trajectories created in worker threads are merged to the corresponding G4Event.
    - Optional – as it requires rather heavy copy operations.

Top-level scenario

- In the master thread, in addition to the ordinary stacks (Urgent, Waiting, (Waiting_1, …)), user may add stacks for each kind of sub-event tasks (*G4SubEventTrackStack*).
  - e.g. optical photons in scintillator, EM particles in calorimeter, …
- A *G4SubEventTrackStack* stacks tracks of selected type, and once the number of tracks becomes the user-specified threshold, it packs them into *G4SubEvent*.
- Worker thread thread pulls a sub-event of its type from the master thread.
  - *G4Event* in the master thread keeps record of spawned *G4SubEvent* objects.
  - Master thread may proceed to the next event(s) while workers still work for the sub-event(s) of previous event(s).
- Worker pushes the resulting thread-local *G4Event* object to the master.
  - It contains all the necessary outcomes (hits, scores, trajectories, etc.) made in the worker thread.
  - Thread-local event is merged into the corresponding *G4Event* in master.
  - After merging, thread-local event object and its contents are safely deleted by the worker thread.
- Worker thread does not have *PrimaryGeneratorAction* but use *G4EventManager::ProcessOneEvent(G4TrackVector*)* to process tracks provided in *G4SubEvent*.
  - *G4EventManager* of the worker thread does the job just the same as what it does in the event-level parallel mode.

G4UserSubEvtThreadInitialization

- By default, all worker threads have *G4WorkerSubEvtRunManager* with the default sub-event type 0.
- By overriding *CreateWorkerRunManager()* method of *G4UserSubEvtThreadInitialization* class, user may assign dedicated sub-event type to each worker.
  - Assignment can be changed on the fly if load-balancing is needed.

```
G4WorkerRunManager*
MyUserSubEvtThreadInitialization::CreateWorkerRunManager() const
{
  G4int threadID = G4Threading::G4GetThreadId();
  G4int subEvtType = 0;
  switch(threadID)
  {
   case 0: case 1: case 2:
     subEvtType = 1; break;
   default: break;
  }
  return new G4WorkerSubEvtRunManager(subEvtType);
}
```

G4UserActionInitialization::BuildForMaster()

- In sub-event parallel mode, *PrimaryGeneratorAction* must be assigned to the master thread.
- If *UserRunAction* or *UserEventAction* is needed, it must also be assigned to the master thread.
  - There is no “run” or “event” in the worker thread. *G4Event* object created in the worker thread is used as a container to bring the results made by the tasked sub-event.
- As the master thread does the ordinary tracking, it may take *UserTrackinigAction* and *UserSteppingAction* as well.

```
void MyActionInitialization::BuildForMaster() const
{
  if(G4RunManager::GetRunManager()->GetRunManagerType()
                                  ==G4RunManager::subEventMasterRM)
  { SetUserAction(new MyPrimaryGeneratorAction);
    SetUserAction(new MyStackingAction);
  }
  SetUserAction(new MyRunAction);
}
```

MyRunAction is used for both event-level and sub-event-level parallel modes

G4UserActionInitialization::BuildForMaster() (continued)

- For each sub-event type, capacity of *G4SubEvent* may be specified. Default is 1000.
- Once tracks in the stack for a sub-event type reach to the specified maximum capacity, a *G4SubEvent* object is instantiated and tasked to the corresponding worker.
  - At the end of an event (of master thread), remaining tracks are stored in a new *G4SubEvent* object and tasked. I.e. a sub-event does not contain tracks of more than one events.

```
void MyActionInitialization::BuildForMaster() const
{
  if(G4RunManager::GetRunManager()->GetRunManagerType()
                              ==G4RunManager::subEventMasterRM)
  {
     G4RunManager::GetRunManager()->RegisterSubEventType(0,2000);
     G4RunManager::GetRunManager()->RegisterSubEventType(1,500);
  }
}
```

G4UserActionInitialization::Build()

- *UserTrackingAction*, *UserSteppingAction* or *UserStackingAction* may be set to the worker thread. User may alter these classes depending on the sub-event type.

```
void MyActionInitialization::Build() const
{
  auto* runM = G4RunManager::GetRunManager();
  if(runM->GetRunManagerType()==G4RunManager::subEventWorkerRM)
  {
    if(runM->GetSubEventTyp() == 0)
    { SetUserAction(new MyEventActionForPhoton); }
    else if(runM->GetSubEventTyp() == 1)
    { SetUserAction(new MyEventActionForEM); }
  }
  else
  {
    SetUserAction(new MyPrimaryGeneratorAction);
    SetUserAction(new MyEventAction);
  }
}
```

In event-level parallelism PrimaryGeneratorAction is set to worker threads

UserStackingAction in the master thread

- In the master thread, in addition to the ordinary stacks (Urgent, Waiting, (Waiting_1, …)), user may add stacks for each kind of sub-event tasks (*G4SubEventTrackStack*).
  - e.g. optical photons in scintillator, EM particles in calorimeter, …
- In the *UserStackingAction::ClassifyNewTrack()* method, depending on the particle type, position, energy, volume/region etc., a track can be directed to the dedicated stack for sub-event.

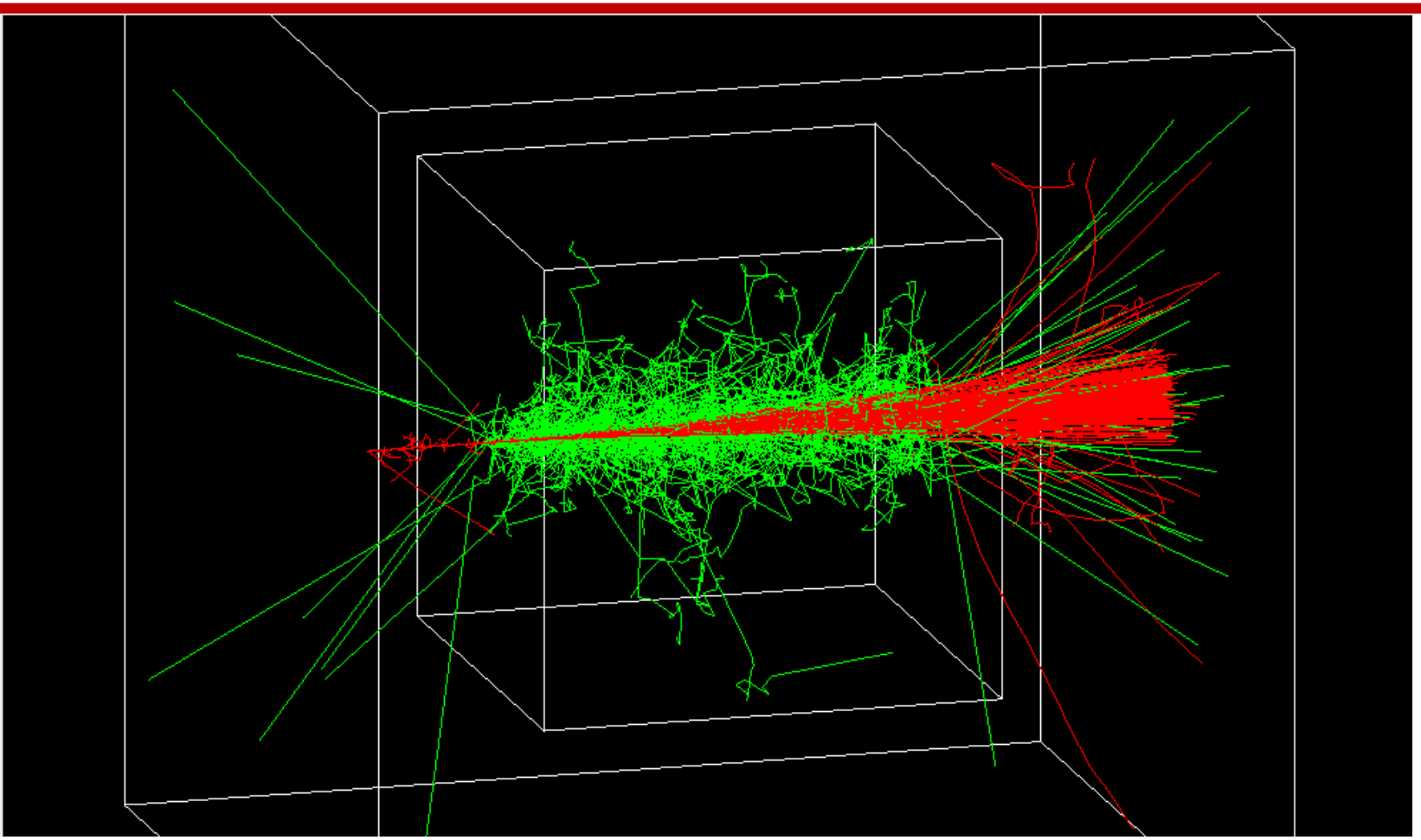
```
G4ClassificationOfNewTrack
MyStackingAction::ClassifyNewTrack(const G4Track* aTrack)
{
   if(aTrack->GetParticleDefinition()==G4OpticalPhoton::Definition())
   { return fSubEvent_0; }
   if(aTrack->GetParticleDefinition()==G4Gamma::Definition()
      && aTrack->GetKineticEnergy()>1.0*GeV)
   { return fSubEvent_1; }
   else
     …
}
```

Shortcut alternative to UserStackingAction

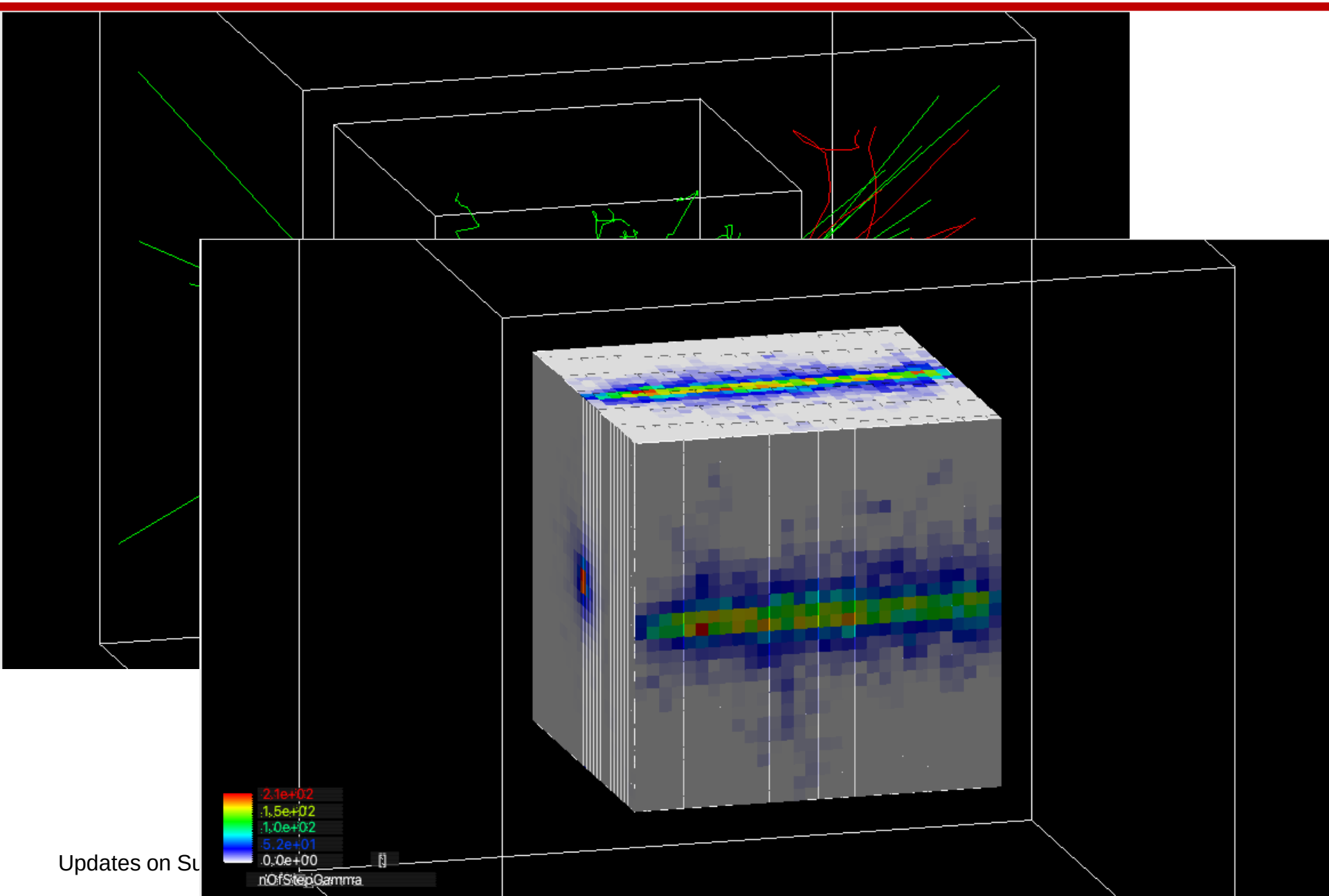
- If classification is based only on the particle type, it can be defined in *UserActionInitialization*.

```
void MyActionInitialization::BuildForMaster() const
{
  if(G4RunManager::GetRunManager()->GetRunManagerType()
                              ==G4RunManager::subEventMasterRM)
  {
     G4RunManager::GetRunManager()->SetDefaultClassification
      (G4OpticalPhoton::Definition(),fSubEvent_0);
  }
}
```

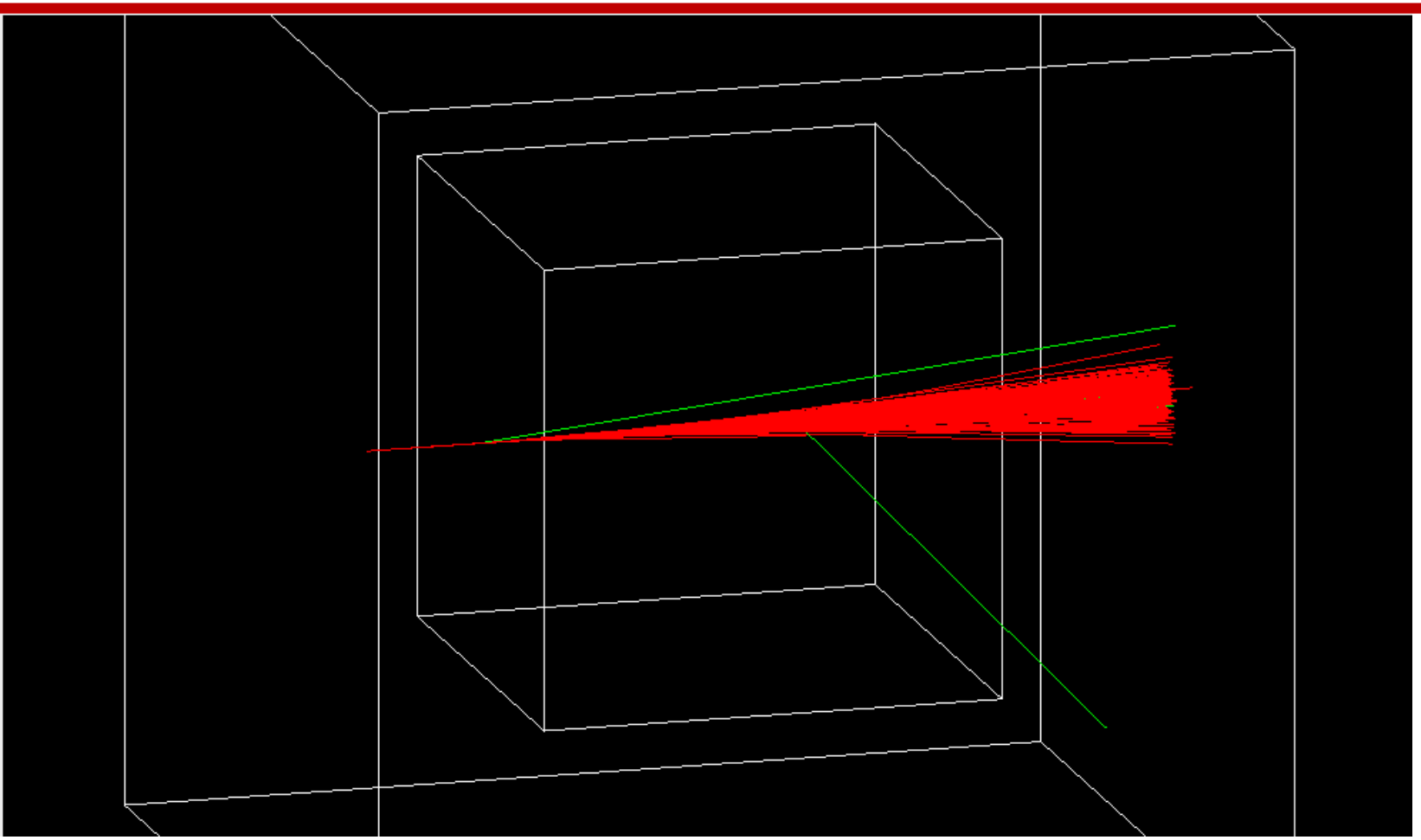
RE03 in event-level parallel mode (1 GeV muons)



RE03 in event-level parallel mode (1 GeV muons)

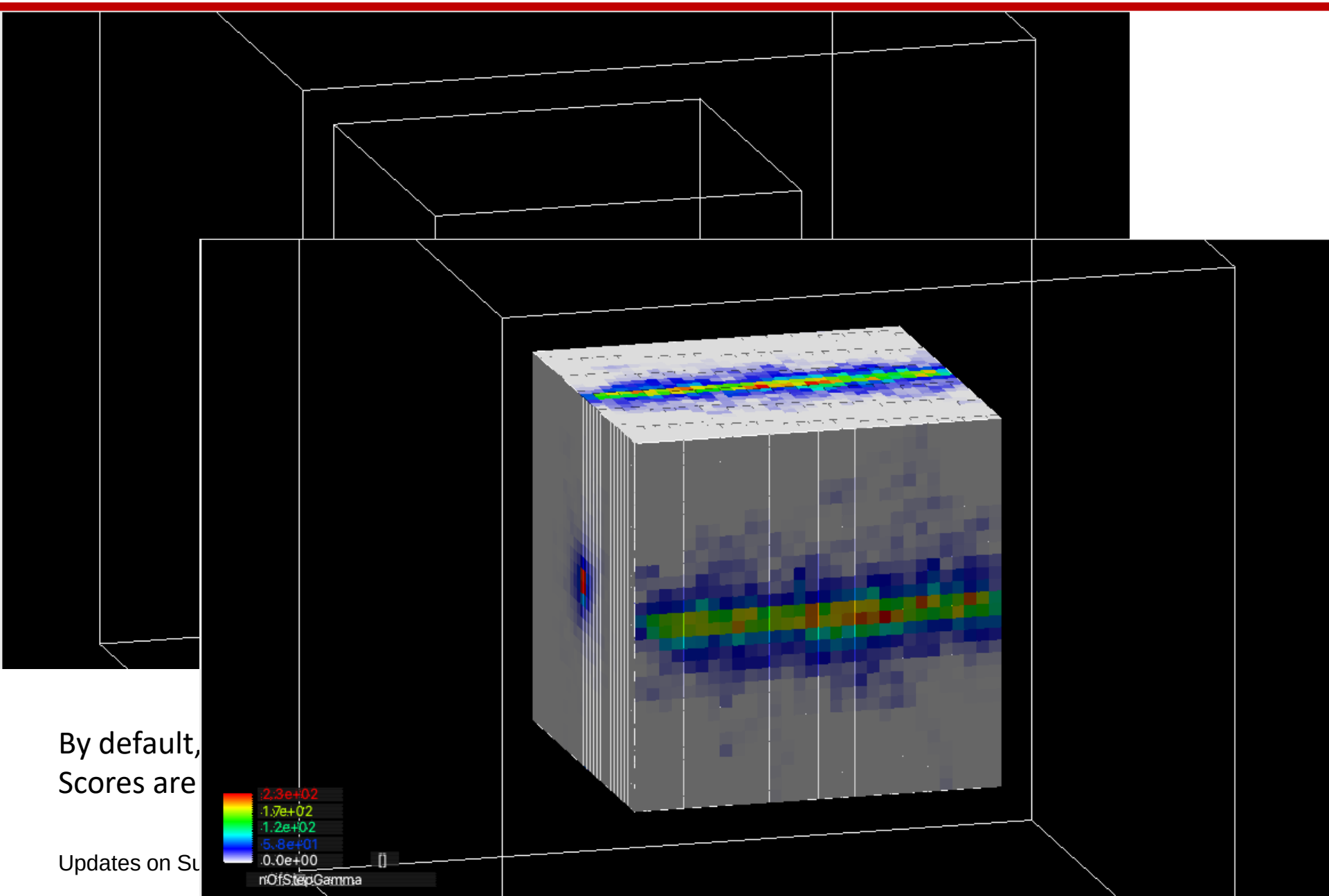


in sub-event parallel mode (e-, e+, γ sent to worker)



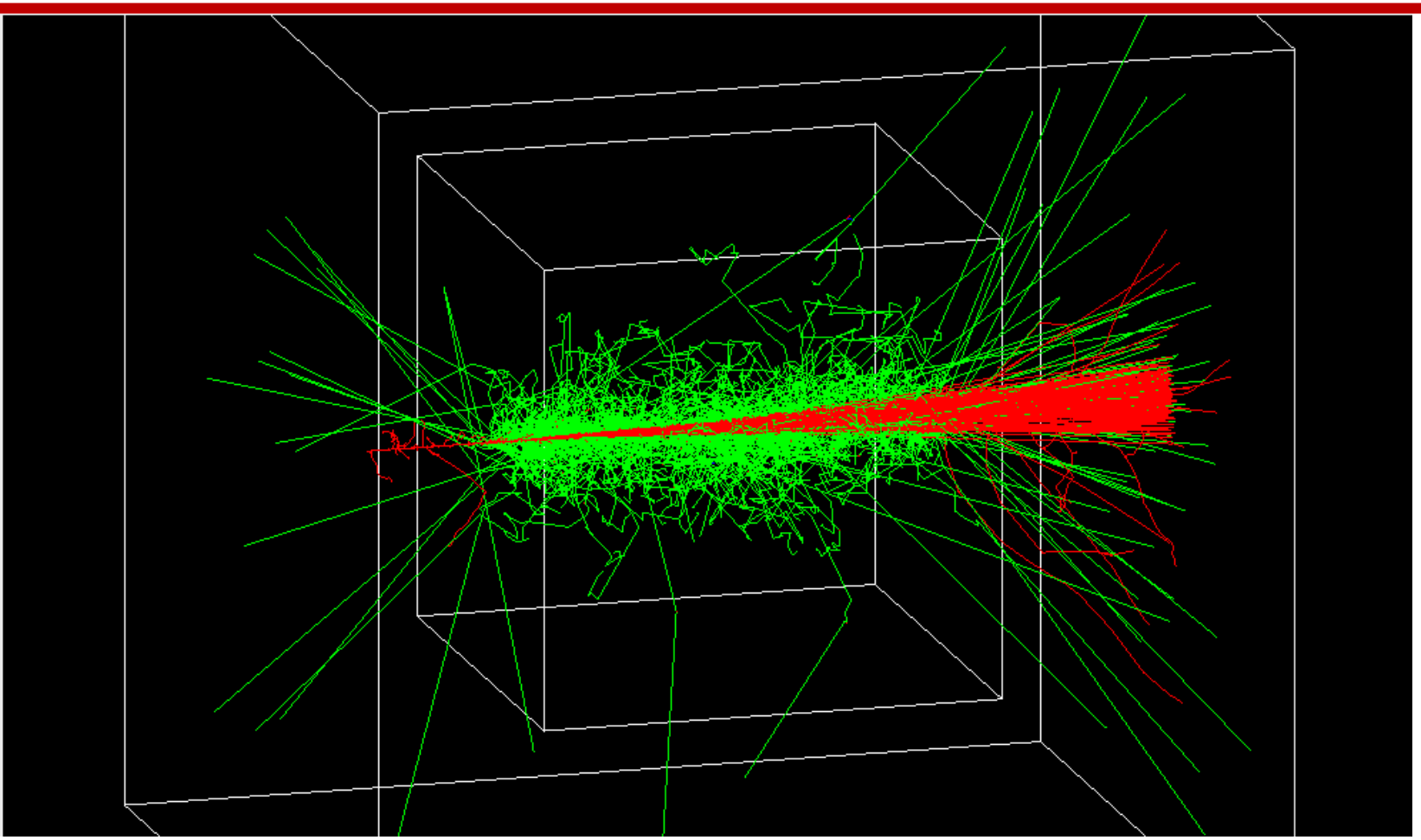
By default, trajectories created in worker threads are not merged to the master.
Scores are automatically merged.

in sub-event parallel mode (e-, e+, γ sent to worker)

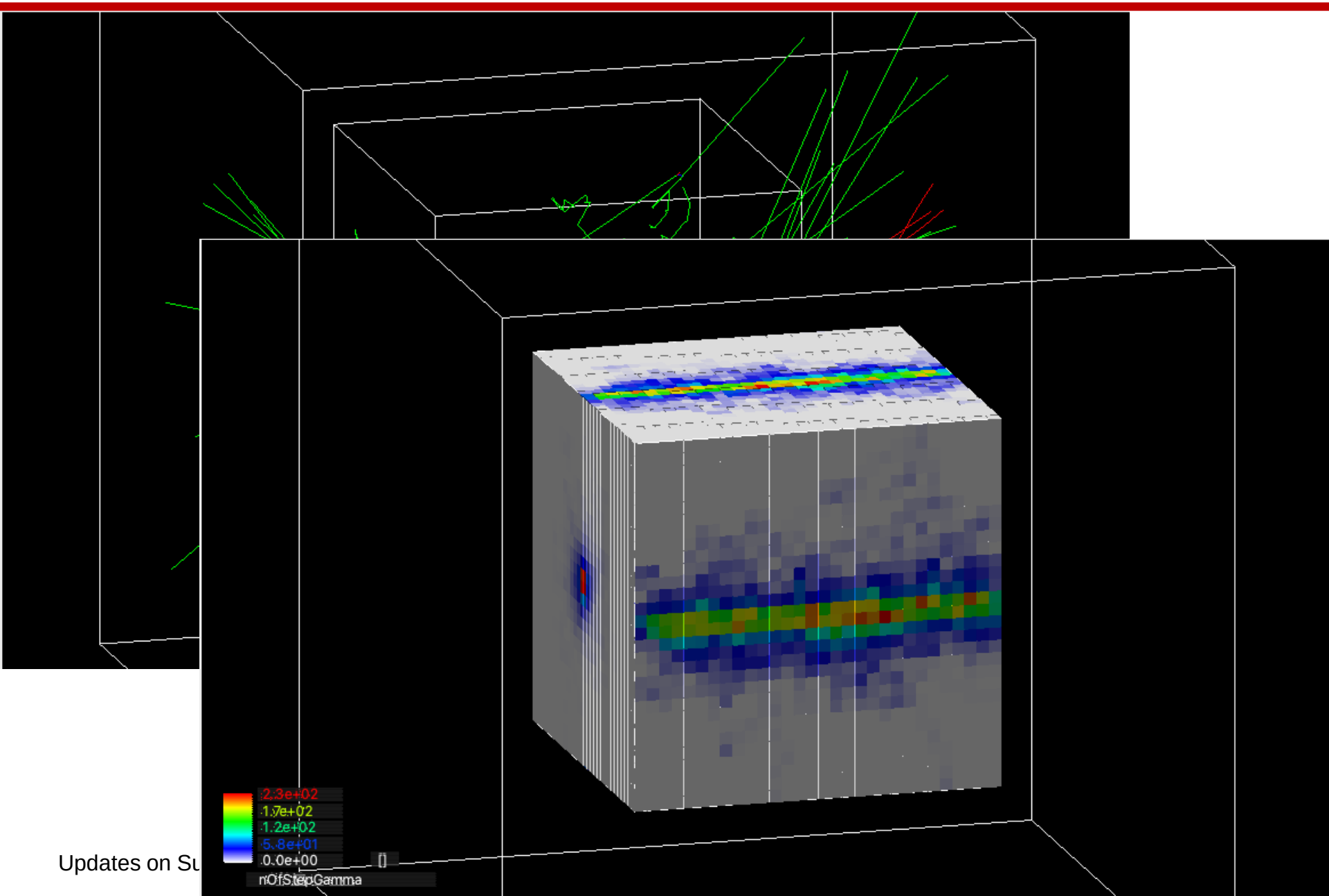


By default,
Scores are

in sub-event parallel mode (trajectories merged)



in sub-event parallel mode (trajectories merged)



Object ownership

- Given we use G4Allocator, we need clean management of Who-Owns-What, Who-Deletes-What and -When.
  - Objects instantiated by the master thread must be deleted by the master. Objects instantiated by the worker thread must be deleted by the responsible worker.
- G4SubEvent object and tracks in it are deleted by the master after the worker refers/uses them.
  - Tracks must be copied when sent from master to worker.
- Thread-local G4Event created by a worker thread and objects contained by it (hits, scores, trajectories) are deleted by the worker after letting the master thread copy/merge the contents.
  - Scores – automatically merged by Geant4 kernel classes
  - Trajectories – copied by CloneForMaster() method and stored into the master G4Event
    - Optional as it requires massive copying
  - Hits – user needs to implemet += operator of HitsCollection

Good news

- As a side effect of this sub-event parallelism development, G4Event are now cleanly deleted as soon as possible if these events are not requested to be kept for redrawing during the following Idle state.
  - Both in the original MT and Tasking event-level parallel modes
    - And in sub-event parallel mode
  - Kept events are deleted at the beginning of next run or at the termination of the program.
    - Up to now, all events were kept until next run due to some race issues between worker threads and the visualization thread.
  - Deletion of unnecessary events is essential when user simulates thousands of events and selects just a few interesting events to be revisited after the run.
    - In the *EndOfEventAction*, invoke *G4Event::KeepTheEvent()* for the events that are of your interest.

Current status and plan

- Most of the classes have been committed to the repository.
- Toward 11.3 release
  - Currently only the basic *G4Trajectory* can be merged to the master. *G4RichTrajectory* and *G4SmoothTreajectory* will be updated.
  - G4RunManagerFactory needs update
  - Code cleanup
  - A new example derived from extended/runAndEvent/RE03 will be added.
  - Documentation
- Phase-II development
  - Each worker may have only the necessary physics processes and sees the limited detector geometry that are necessary for that task.
  - Physics list / geometry per task