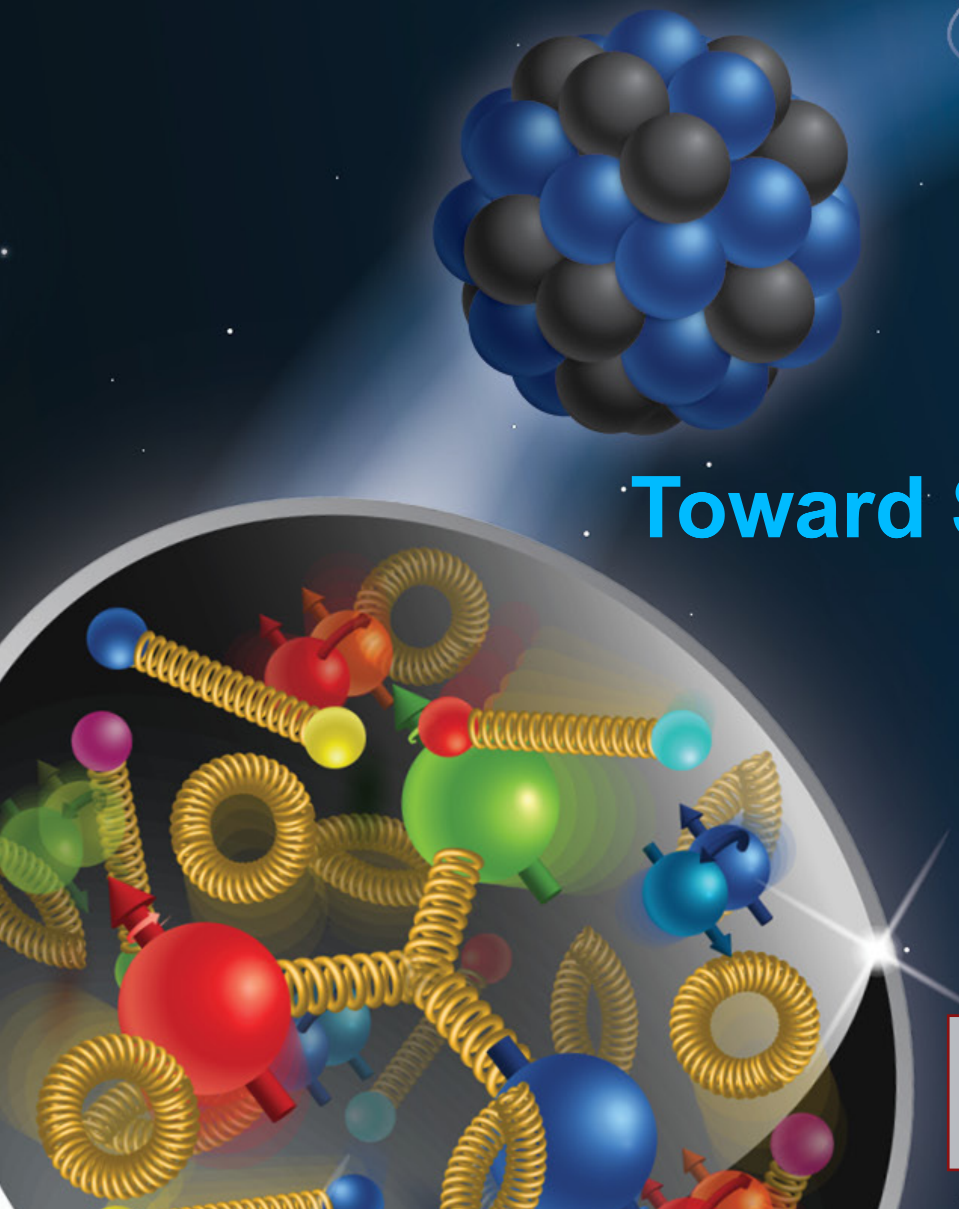


# Toward Sub-Event Parallelism

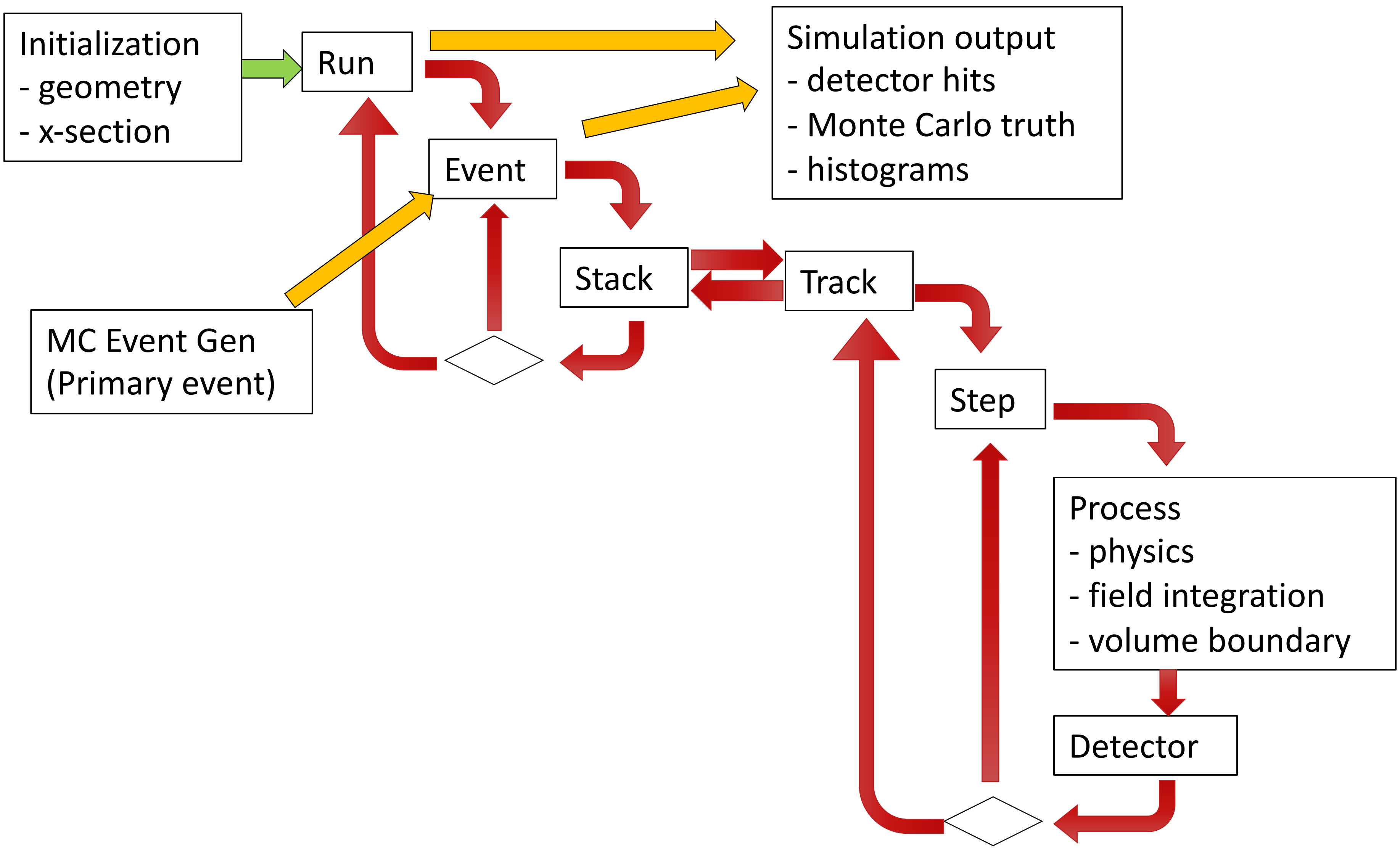
Makoto Asai (JLab/SCT)
asai@jlab.org

## Geant4 evolutions in parallelization

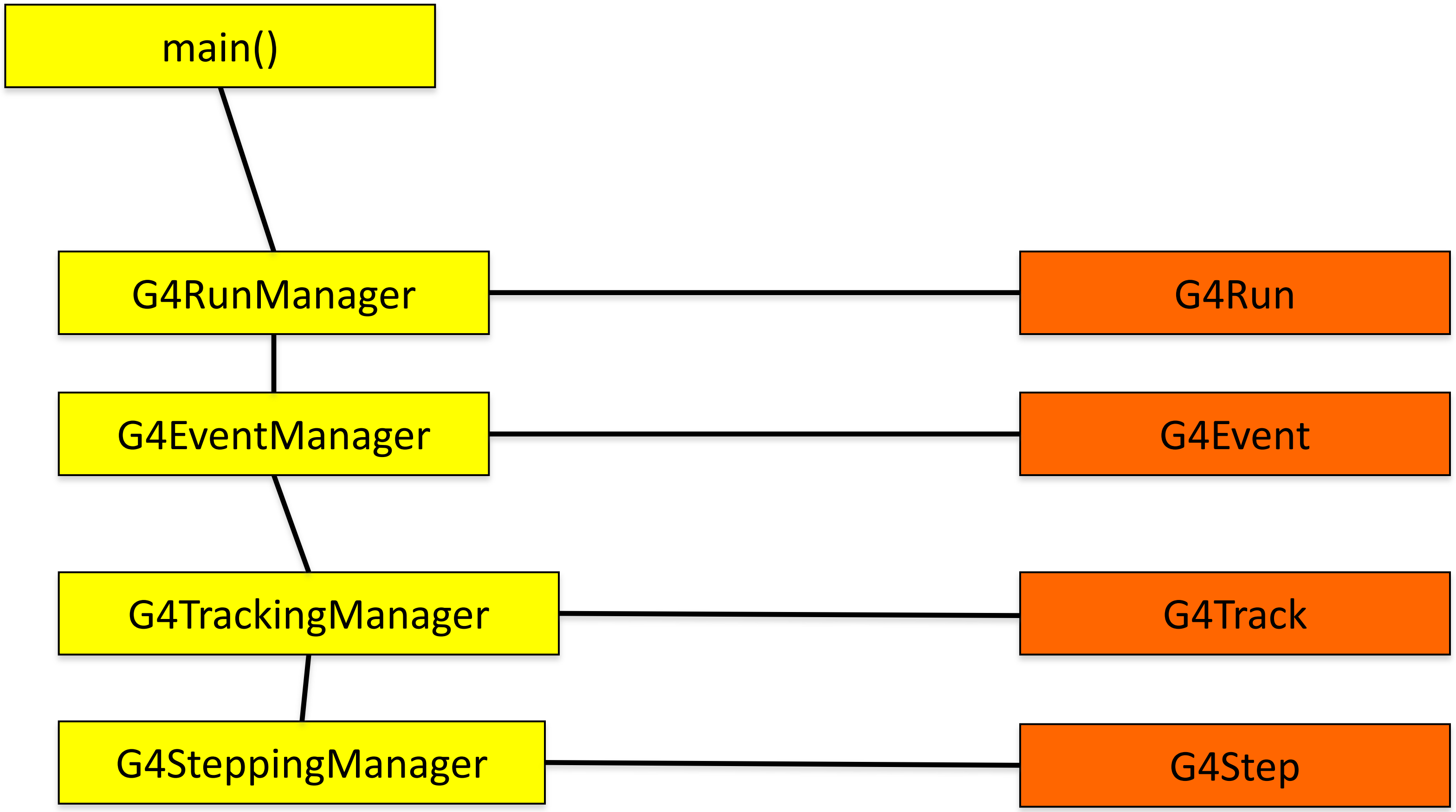
1. Sequential mode : original since Geant4 v1.0
   - Single core (thread) does everything
2. Multithreaded event-level parallel mode : since Geant4 v10.0 (Dec.2013)
   - Taking the advantage of independence of events, many cores (threads) process events in parallel (event-level parallelism)
   - Geometry / 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.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

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

# Geant4 as a detector simulation engine

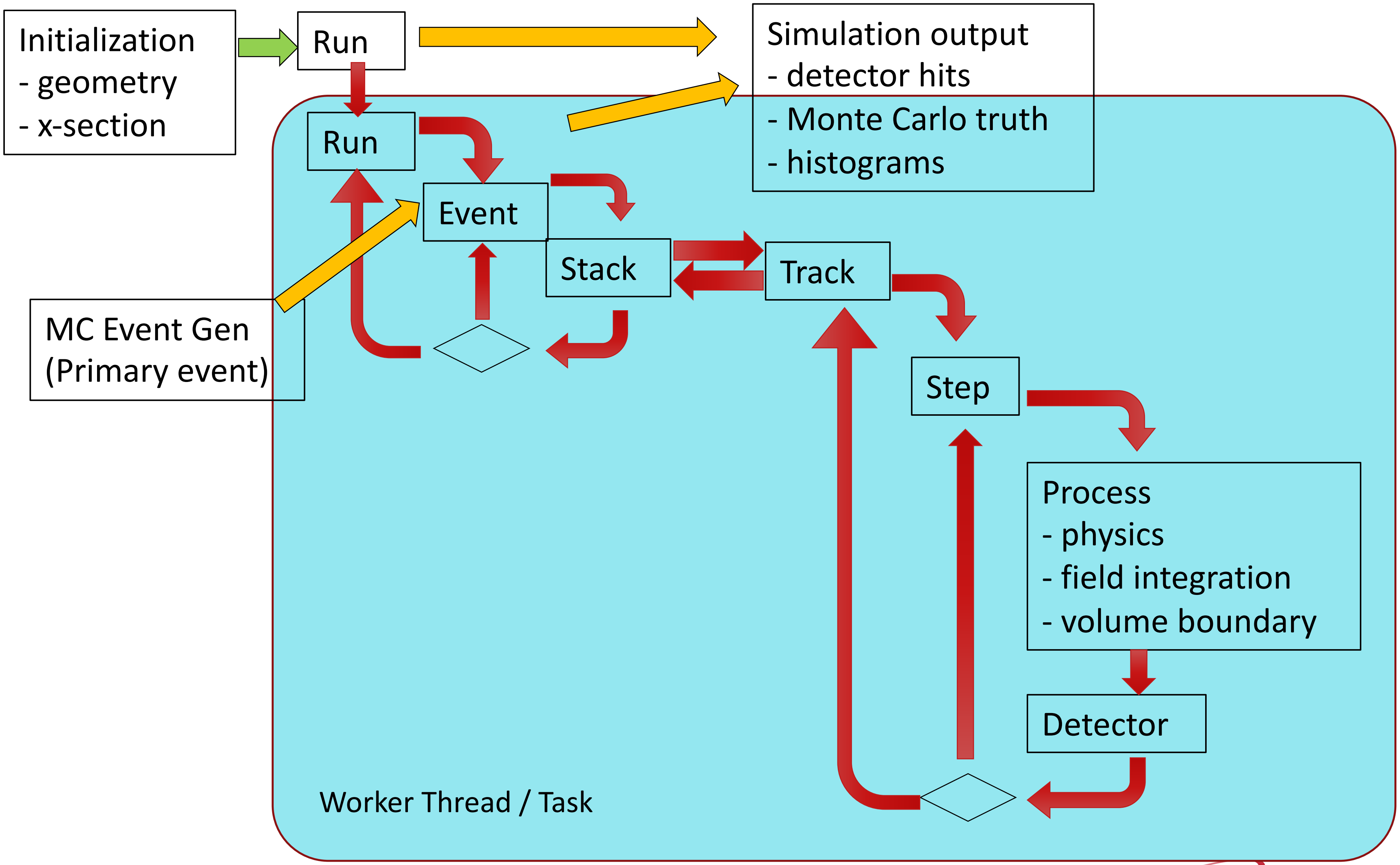


# Sequential mode

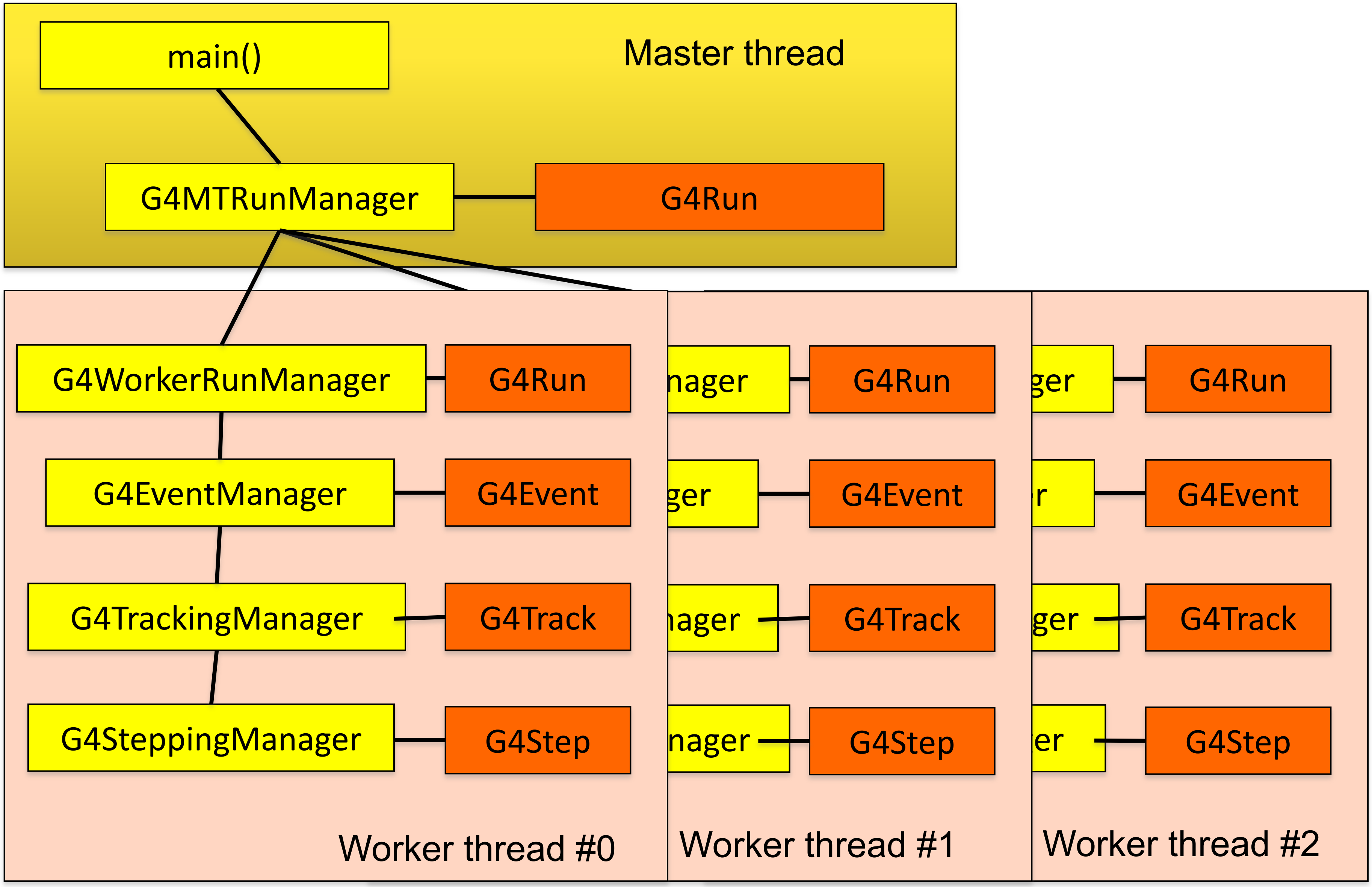


Jefferson Lab

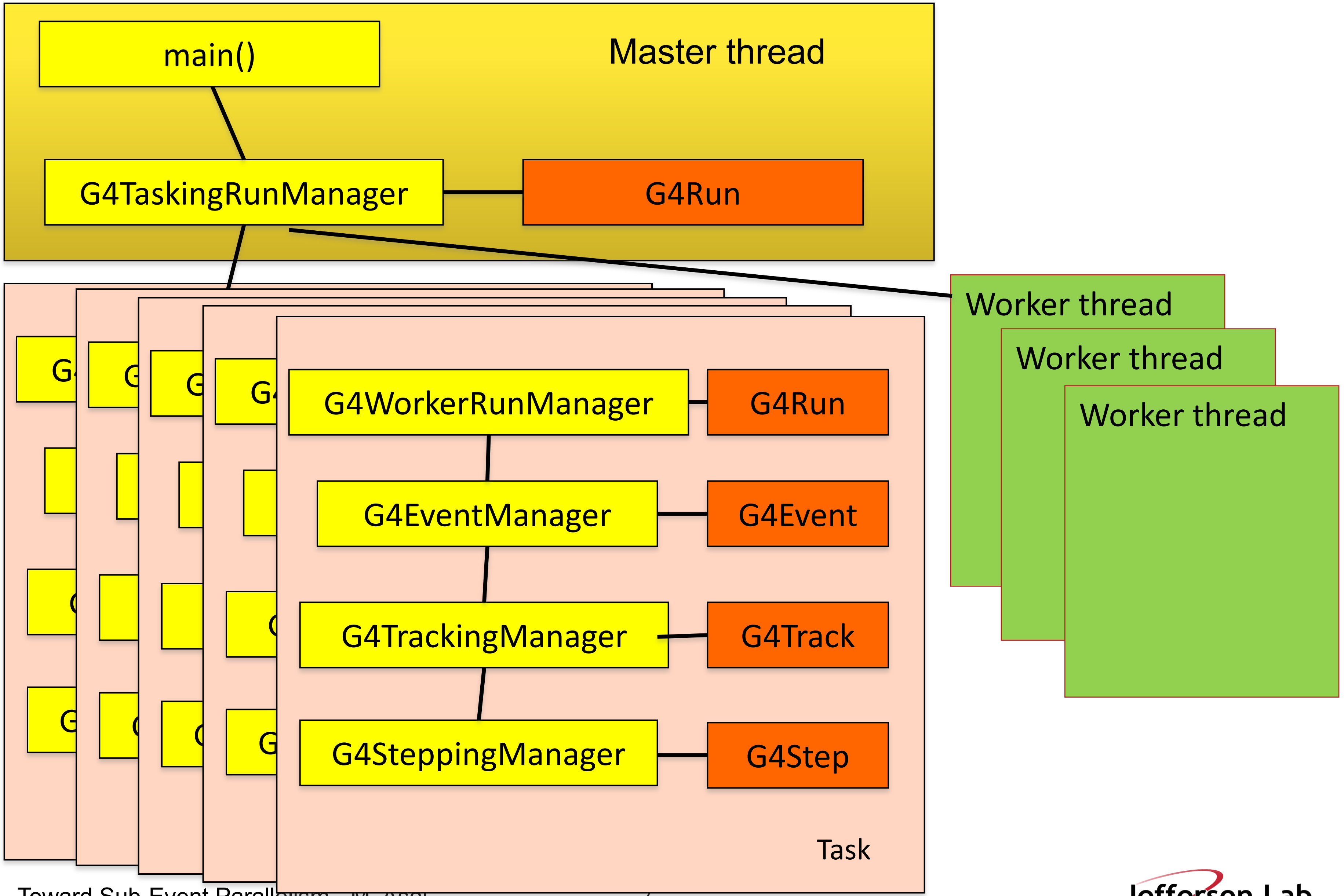
# Event-level parallel mode (thread / task)



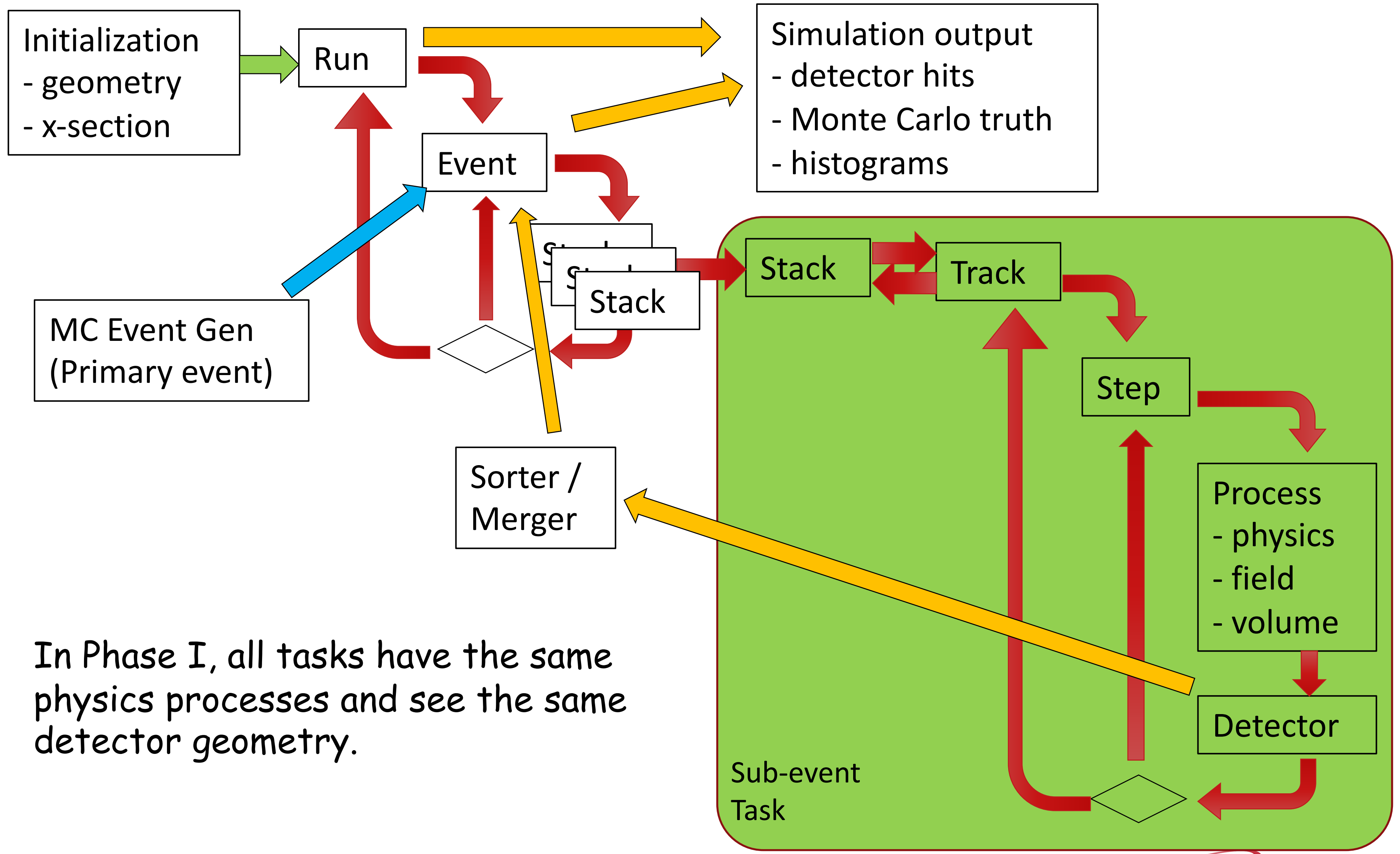
# Multithreaded mode



## Task-based parallel mode

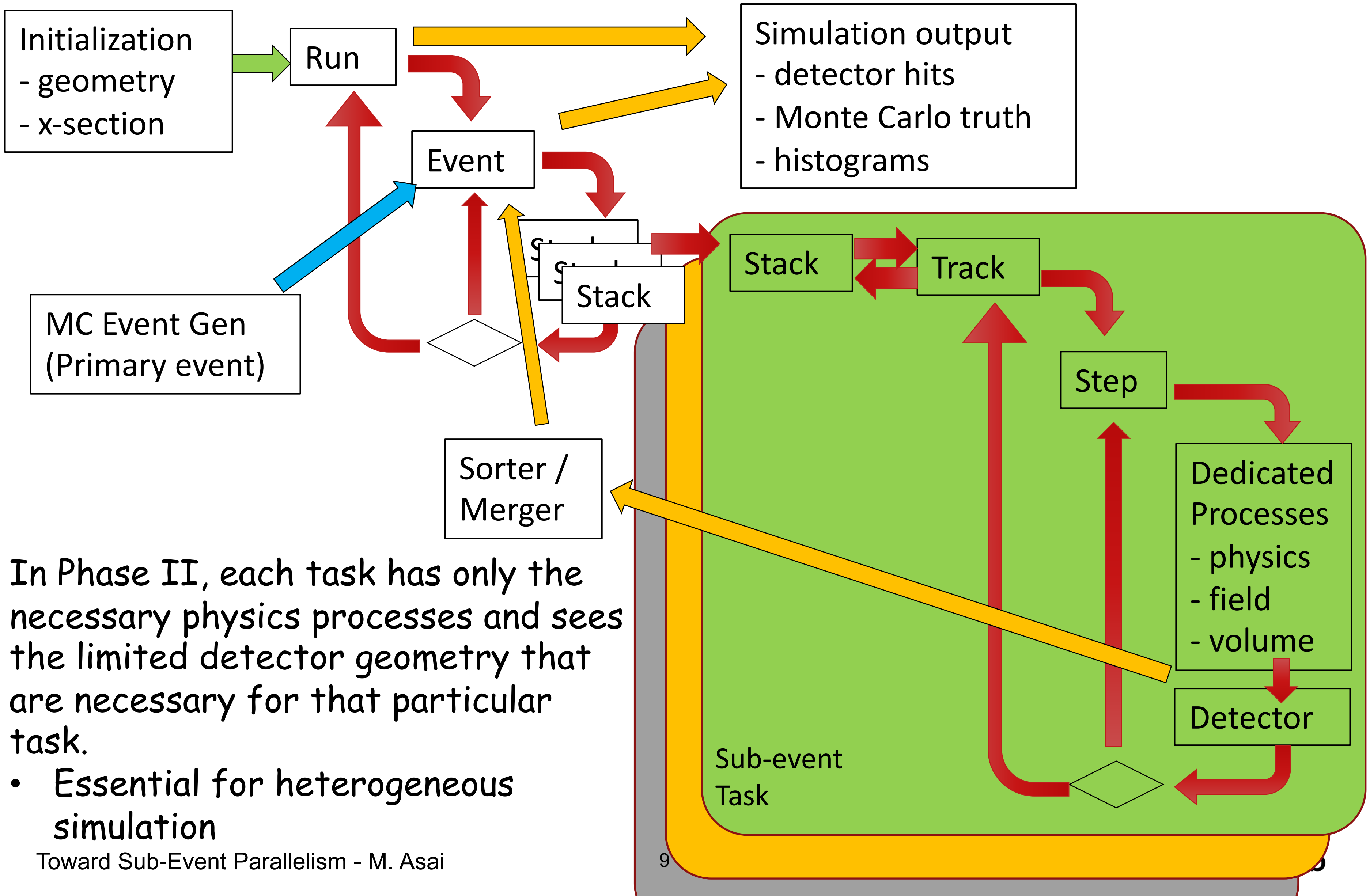


# Task-based sub-event parallel mode (Phase I)



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

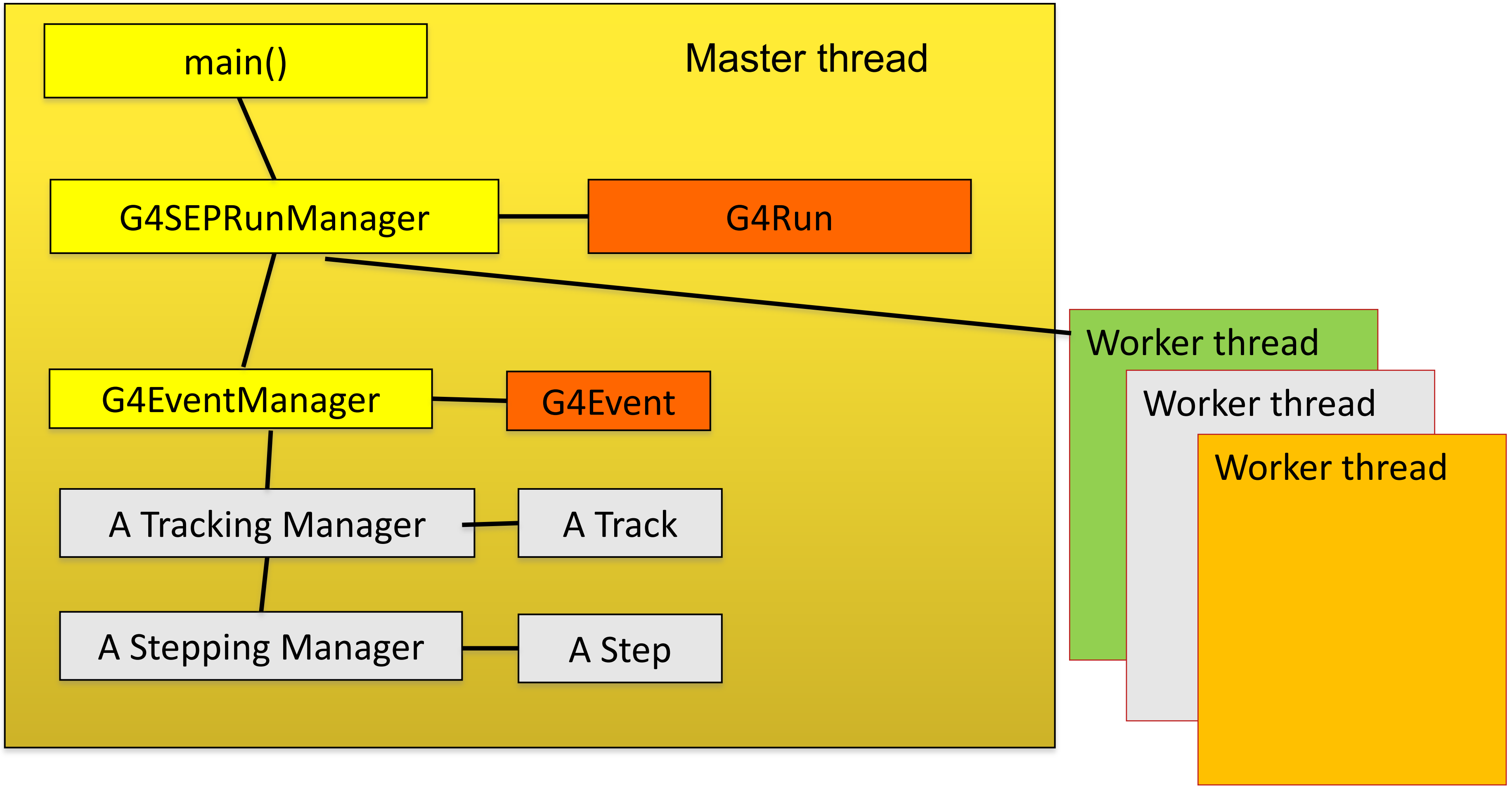
# Task-based sub-event parallel mode (Phase II)



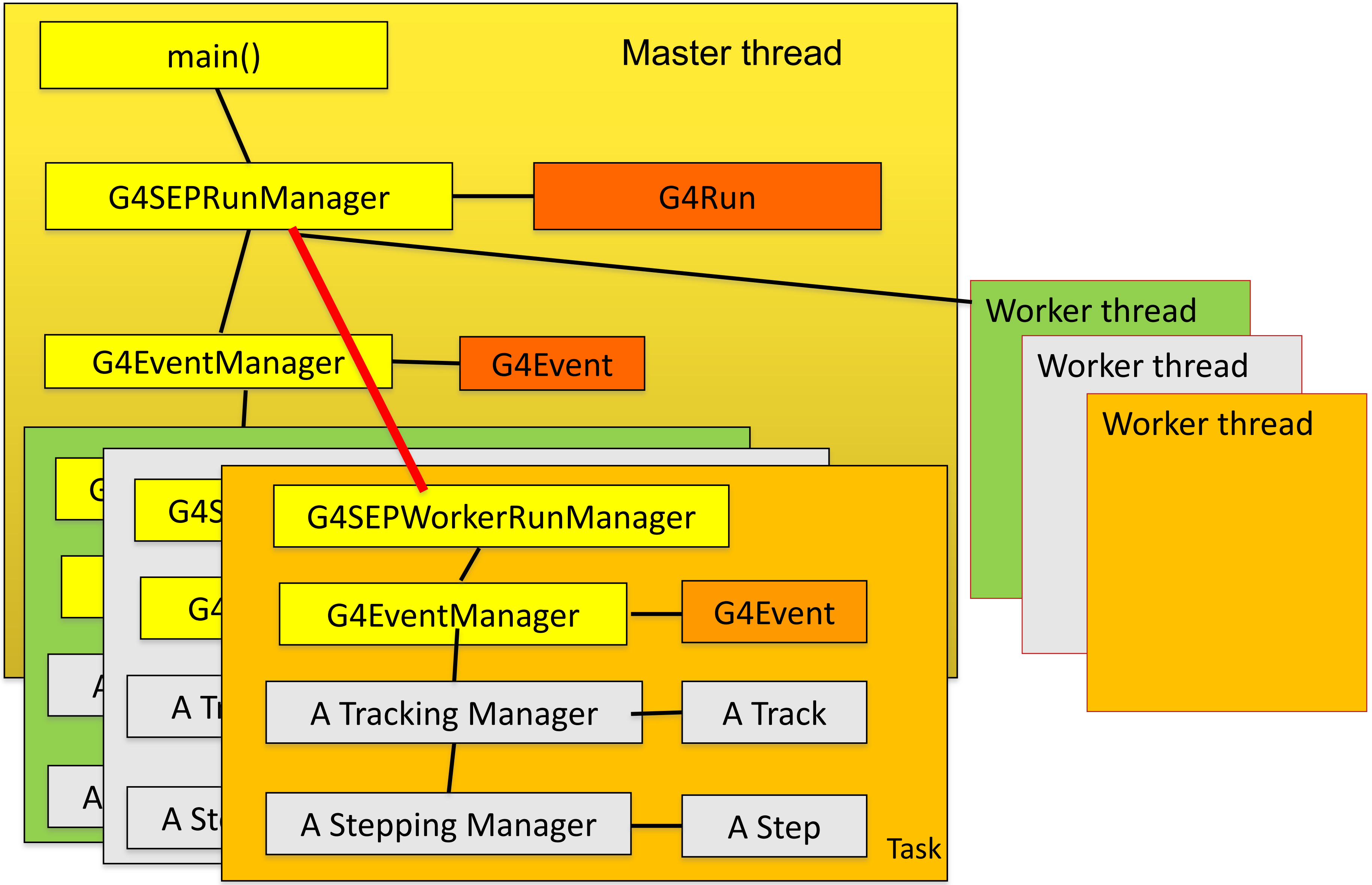
In Phase II, each task has only the necessary physics processes and sees the limited detector geometry that are necessary for that particular task.

- Essential for heterogeneous simulation

## Sub-event parallel mode



## Sub-event parallel mode



## 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, EM particles in calorimeter, …
- A G4SubEventTrackStack stacks tracks of specified type, and once the number of track becomes the user-specified threshold, it packs them into G4SubEvent.
- G4Event keeps pointers of G4SubEvent objects and keep record of them.
- WorkerRunManager thread pulls a sub-event of its type from the master RunManager.
- 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 Tasking mode
  - It stores all the necessary outcomes (hits, scores, *trajectories*, etc.) into the local G4Event object.
- WorkerRunManager pushes the resulting G4Event object back to the master RunManager.

## 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 must be deleted by the master after the worker refers/uses them.
  - Tracks must be copied.
- G4Event created by a worker thread and objects contained by it (hits, scores, *trajectories*) must be deleted by the worker after letting the master thread copy/merge the contents.
  - Scores – merged by Geant4 kernel classes
  - *Trajectories – optional* : copied by Clone() method and stored into the master G4Event
  - Hits – need += operator of HitsCollection implemented by the user
    - tracker hits : copy and insert
    - calorimeter hits : add

## Sub-event parallelism in Geant4

- Sub-event parallelism will be introduced without forcing user's code to migrate.
  - Sequential, threading and tasking modes will work fine as they do now.
- To use sub-event parallelism (for Phase-I)
  - Use a newly introducing G4SEPRunManager and G4SEPWorkerRunManager for sub-event parallelism
  - Define what sort of tracks (sub-event) should be sent to which worker thread

```
auto rm = G4RunManager::GetRunManager()
rm->RegisterSubEventType(1,1000);
rm->SetDefaultClassification
          (G4Electron::Definition(),fSubEvent_1);
rm->SetDefaultClassification
          (G4Positron::Definition(),fSubEvent_1);
rm->SetDefaultClassification
          (G4Gamma::Definition(),fSubEvent_1);
rm->RegisterSubEventType(2,10000);
rm->SetDefaultClassification
          (G4OpticalPhoton::Definition(),fSubEvent_2);
```

## Current status

- Most of the changes except G4SEPRunManager and G4SEPWorkerRunManager are in the repository.
    - Remaining two classes will come in a couple of weeks.
- Note:
    - When sub-event parallelism is used, UserEventAction::EndOfEventAction() is invoked by the master RunManager instead of EventManager.
        - As, some sub-events may still be in process in worker threads after the processing of an event in the master thread is completed.
    - Currently, visualization is considered only with master thread.
        - We don't know yet if we can visualize something from worker threads.
        - Need discussion with Vis WG next year.

## By the way, a side topic

- The development of photon entanglement process requires first-in-first-out stack.
  - To flip two tracks after letting each of them make several steps and suspended.
  - Ordinary stack (last-in-first-out) doesn't do this.
    - Once you suspend a track, it is popped up from the stack prior to the other track that had already been suspended.
- You can add one more waiting stack and send the suspended track to the second waiting stack.
  - Once the urgent stack becomes empty, tracks in the first (default) waiting stack are sent to the urgent stack, and tracks in the second waiting stack are sent to the first waiting stack. So, you basically have first-in-first-out stack.

```
auto rm = G4RunManager::GetRunManager();
rm->SetNumberOfAdditionalWaitingStacks(1);
rm->SetDefaultClassification(fSuspend, fWaiting_1);
```

- This same trick may be used for R-hadron, etc.
- Note: this setting is thread-local.

## Toward the Phase-II sub-event parallelism

- In sub-event parallelism Phase-II
  - G4SEPWorkerRunManager is extensible to accommodate more than one kinds of sub-events, and they should be instantiated in G4UserThreadInitialization.
  - Physics list and/or detector construction dedicated to each task if appropriate.
    - For example, with G4HepEM, AdePT, Celeritas, Opticks, etc.
- Each type of WorkerThread may have its unique PhysicsList
  - For example, if a worker is for G4HepEM, it does not need any other particles rather than EM particles, and physics processes assigned to it may not necessarily be the same as those in the master thread.
- Each type of WorkerThread may have its unique geometry
  - For example, if a worker is for Opticks, it does not need any detailed geometry of the detector except the scintillator / crystal and light guides.

## Contents

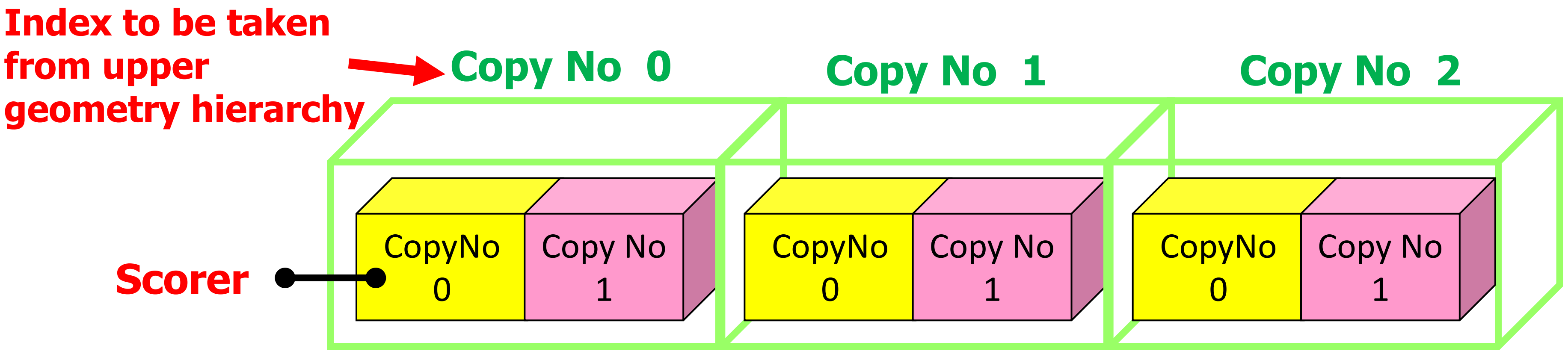
- Parallel world
- Layered mass geometry
- Real-world scoring and scoring probe
- Histogram filling through a scorer

## Define scorer to a tracking volume

- Define a scorer to a logical volume.

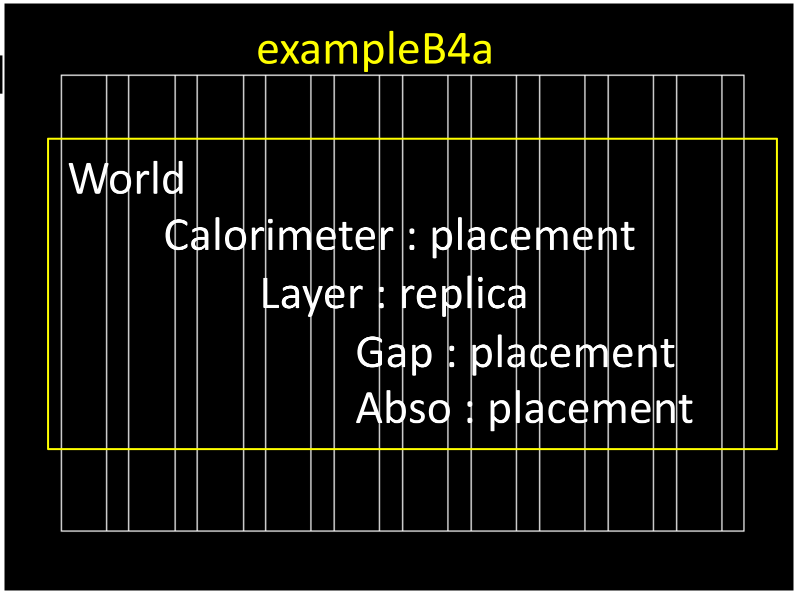
```
      /score/create/realWorldLogVol <LV_name> <anc_lvl>
```

- One can define arbitrary scoring quantities and filters.
  - Same recipe as scoring mesh.
  - Scores are automatically merged over worker threads.
  - Drawing is not yet supported.
- All physical volumes that share the same *`<LV_name>`* have the same primitive scorers but score separately.
  - Copy number of the physical volume is the index.
  - If the physical volume is placed only once, but its (grand-)mother volume is replicated, use the *`<anc_lvl>`* parameter to indicate the ancestor level where the copy number should be taken.



## Command-based real-world scorer

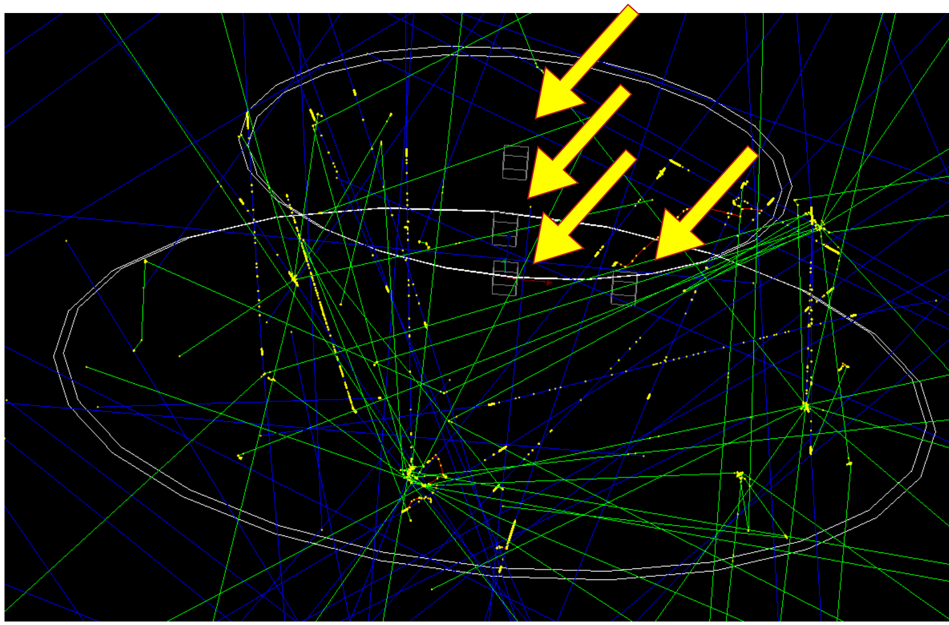
- Do not use this /score/create/realWorldLogVol command to a mother logical volume.
  - For example of this exampleB4, "Layer" is fully filled with "Gap" and "Abso" daughter volumes. You won't see any energy deposition in "Layer" volume.



```
/score/create/realWorldLogVol Gap 1
/score/quantity/energyDeposit eDep MeV
/score/quantity/trackLength sLen mm
/score/filter/charged cFilter
/score/create/realWorldLogVol Abso 1
/score/quantity/energyDeposit eDep MeV
/score/quantity/trackLength sLen mm
/score/filter/charged cFilter
/score/close
```

If this is not set, given "Gap" and "Abso" are placed with copy number 0, energy deposition and track length are accumulated for all layers.

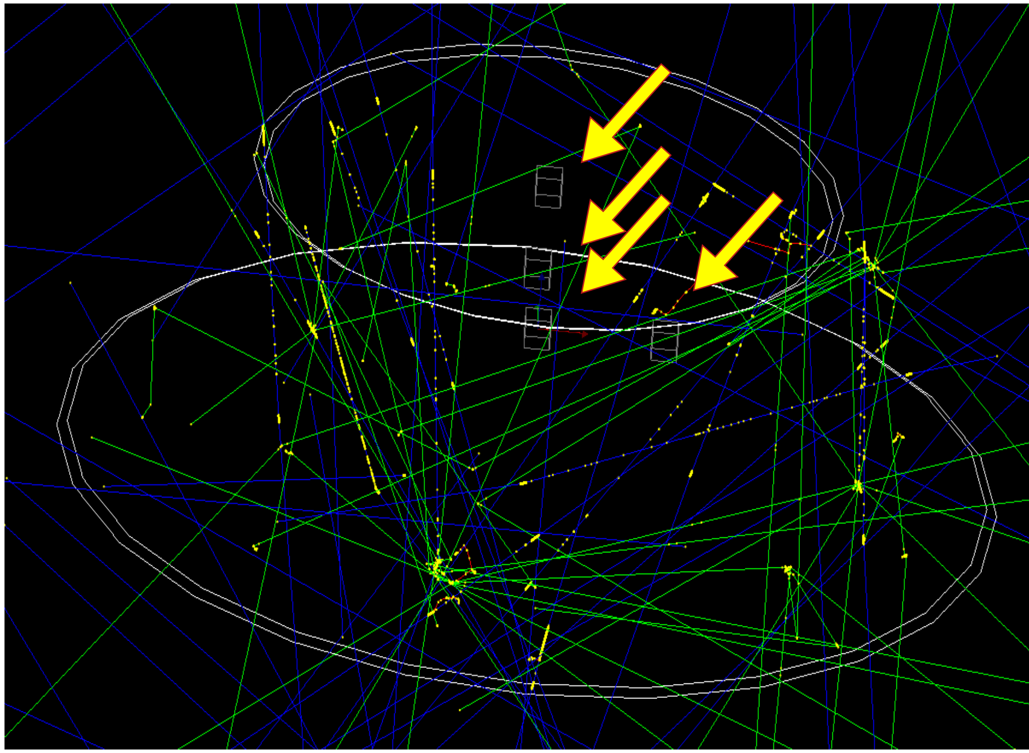
## Command-based probe scorer



- User may locate scoring “probes” at arbitrary locations. A “probe” is a virtual cube, to which any Geant4 primitive scorers could be assigned.
- Given these probes are located in an artificial “parallel world”, probes may overlap to the volumes defined in the mass geometry.
- If probes are located more than once, all probes have the same scorers but score individually.
- In addition, the user may optionally set a material to the probe. Once a material is set to the probe, it overwrites the material(s) defined in the mass geometry when a track enters the probe cube.
  - Because of this overwriting, physics quantities that depend on material or density, e.g. energy deposition, would be measured accord to the specified material
  - You are alternating material, i.e. simulation results are affected. Make probes small and as few as needed.

## Scoring probe

```
/score/create/probe Probes 5. cm
/score/probe/material G4_WATER
/score/probe/locate 0. 0. 0. cm
/score/probe/locate 25. 0. 0. cm
/score/probe/locate 0. 25. 0. cm
/score/probe/locate 0. 0. 25. cm
/score/quantity/energyDeposit eDep MeV
/score/quantity/doseDeposit dose mGy
/score/quantity/volumeFlux volFlx
/score/quantity/volumeFlux protonFlux
/score/filter/particle protonFilter proton
/score/close
```



Note: To visualize the probes defined in a parallel world, the following command is required.

```
/vis/drawVolume worlds
```

# Contents

- Parallel world
- Layered mass geometry
- Real-world scoring and scoring probe
- Histogram filling through a scorer

## 1-D histogram directly filled by a primitive scorer

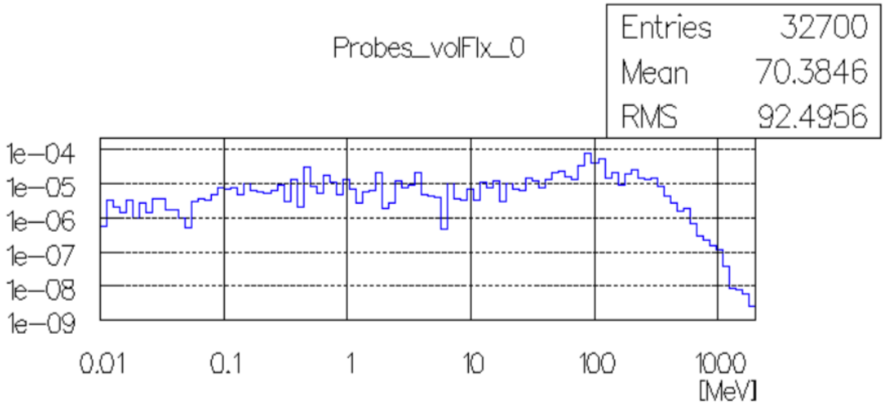
- Through a newly introduced interface class (G4TScoreHistFiller) a primitive scorer can directly fill a 1-D histogram defined by G4Analysis.
  - Track-by-track or step-by-step filling allows command-based histogram such as energy spectrum.
- G4TScoreHistFiller template class must be instantiated in the user's code with his/her choice of analysis data format.

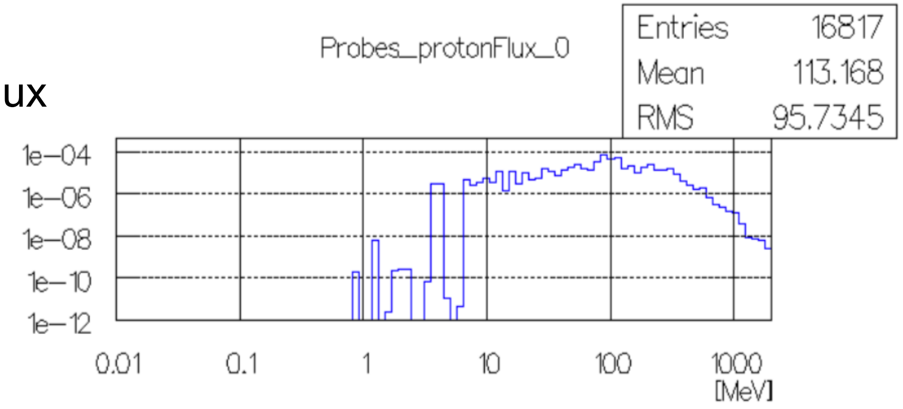
```
#include “G4AnalysisManager.hh”
#include “G4TScoreHistFiller.hh”
auto analysisManager = G4AnalysisManager::Instance();
analysisManager->SetDefaultFileType(“root”);
auto histFiller = new G4TScoreHistFiller<G4AnalysisManager>;
```

- Primitive scorer must be defined in advance to setting a histogram.
- Histogram must be defined through /analysis/h1/create command in advance to setting it to a primitive scorer.
- This functionality is available only for primitive scorers defined in real-world scorer or probe scorer.
  - Not available for box or cylindrical mesh scorer due to memory consumption concern.

# 1-D histogram directly filled by a primitive scorer

```
/score/create/probe Probes 5. cm
/score/probe/locate 0. 0. 0. cm
/score/quantity/volumeFlux volFlux
/score/quantity/volumeFlux protonFlux
/score/filter/particle protonFilter proton
/score/close
/analysis/h1/create volFlux Probes_volFlux
                    100 0.01 2000. MeV ! log
/score/fill1D 1 Probes volFlux
/analysis/h1/create protonFlux Probes_protonFlux
                    100 0.01 2000. MeV ! log
/score/fill1D 2 Probes protonFlux
```





N.B. If probe is placed more than once, *fill1D* command should be called to each *copyNo.*

```
    /score/fill1D 1 Probes volFlux 0
```

Jefferson Lab
Thomas Jefferson National Accelerator Facility

## Questions?

U.S. DEPARTMENT OF ENERGY | Office of Science

## Monte Carlo simulation on GPU

- It is hopeless to port the entire Geant4 to a single process on GPU.
  - Each GPU process should have strictly limited scope.
    - Physics coverage, particle type, geometry/material
    - E.g. optical photon transport in Cerenkov detector, EM shower in calorimeter (w/o back splash or punch through)
- A task on GPU should behave like a blackhole.
  - The darker a task is, the better performance it has.
    - “Darker” means “less output”.
  - Individual step/track/trajectory should not be taken out from a task.
    - Reshuffling tracks over tasks is no a good thing to do.
  - Minimize output information.
    - E.g. transferring output is a serious bottleneck, even for shared memory.
- Sub-event parallelism is the only solution that allows various tasks of different processes running on GPUs in parallel while conducting the full event simulation.