Multithreading and Heterogeneous Computing

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

- Multithreading in Geant4
- Shell variables and UI commands for multithreading
- Sub-event parallelism
- Heterogeneous computing

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

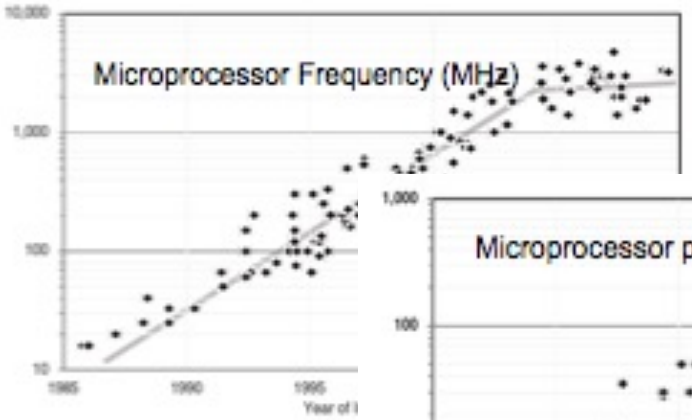
- Multithreading in Geant4
- Shell variables and UI commands for multithreading
- Sub-event parallelism
- Heterogeneous computing

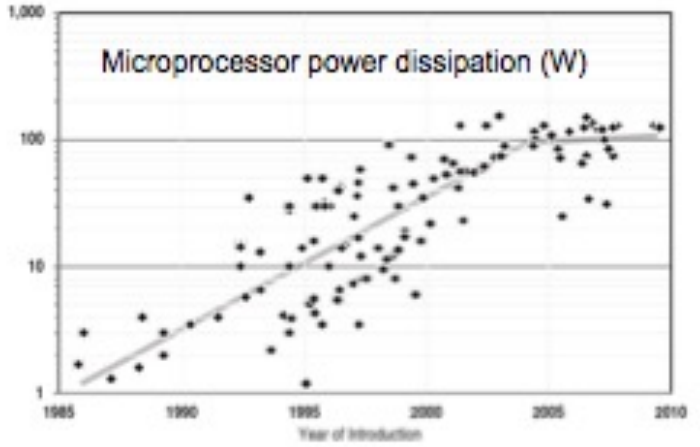
GEANT4
A SIMULATION TOOLKIT

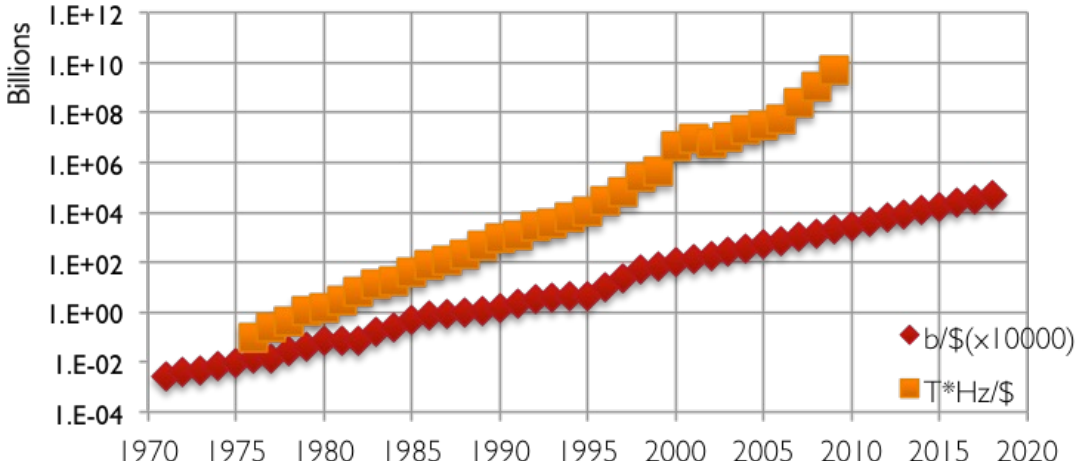
U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

Many-core saga

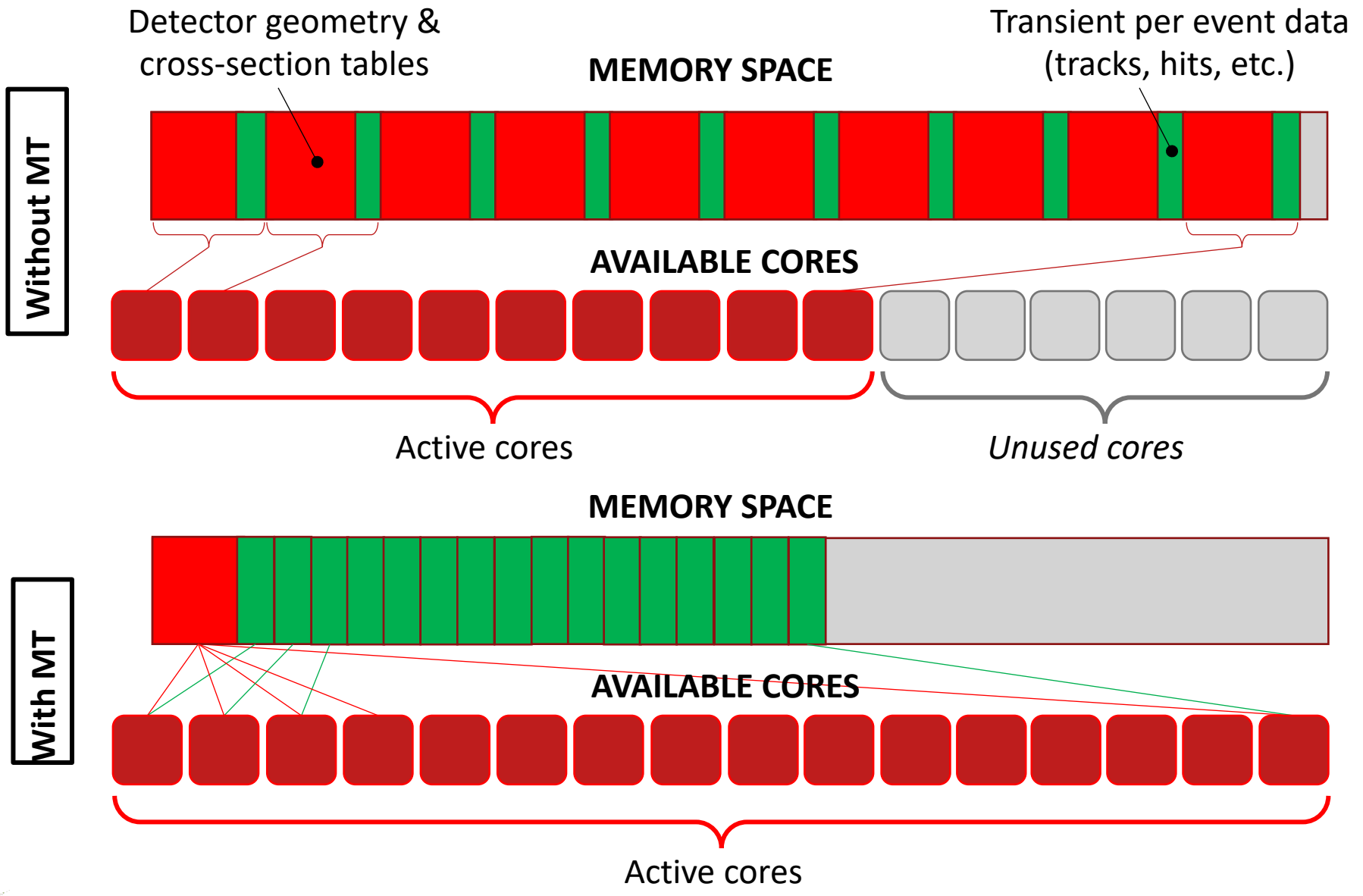




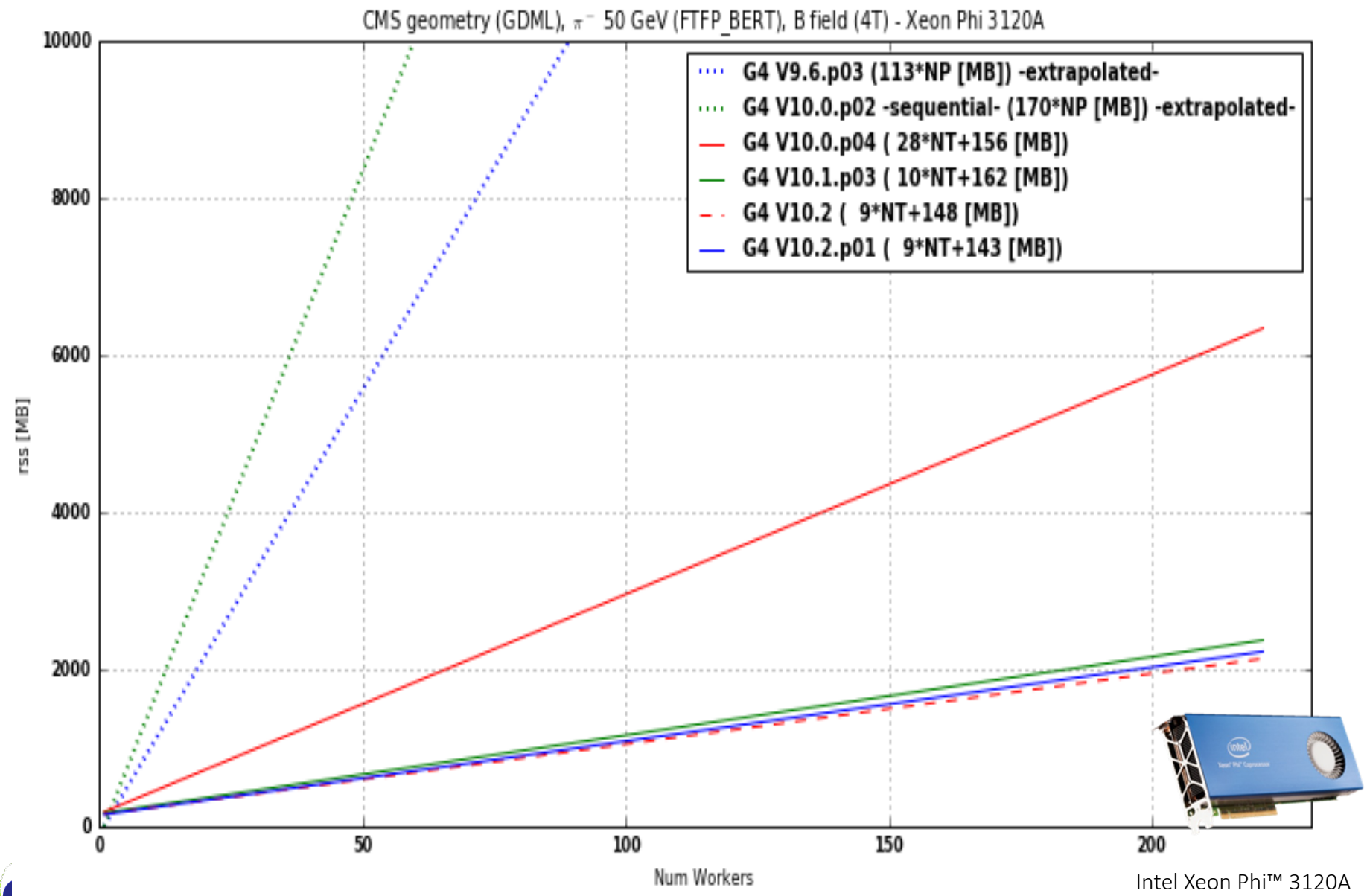


- CPU frequency had come to plateau in 2005. Number of transistors you can buy per $ is still growing.
  - Many core
- Size of memory you can buy per $ is not increasing as fast as number of transistors.
  - Memory size per core is shrinking.
- Naïve parallel processing of many jobs won't work.
  - Requires multithreading application with shared memory.

CPU Clock Frequecy land usage: The Future of Computing Performance: Game Over or Next Level?
DRAM cost: Data from 1971-2000: VLSI Research Inc. Data from 2001-2002: ITRS, 2002 Update, Table 7a, Cost-Near-Term Years, p. 172. Data from 2003-2018: ITRS, 2004 Update, Tables 7a and 7b, Cost-Near-Term Years, pp. 20-21.
CPU cost: Data from 1976-1999: E. R. Berndt, E. R. Dulberger, and N. J. Rappaport, "Price and Quality of Desktop and Mobile Personal Computers: A Quarter Century of History," July 17, 2000, ;Data from 2001-2016: ITRS, 2002 Update, On-Chip Local Clock in Table 4c: Performance and Package Chips: Frequency On-Chip Wiring Levels -- Near-Term Years, p. 167. ;
Average transistor price: Intel and Dataquest reports (December 2002), see Gordon E. Moore, "Our Revolution,"

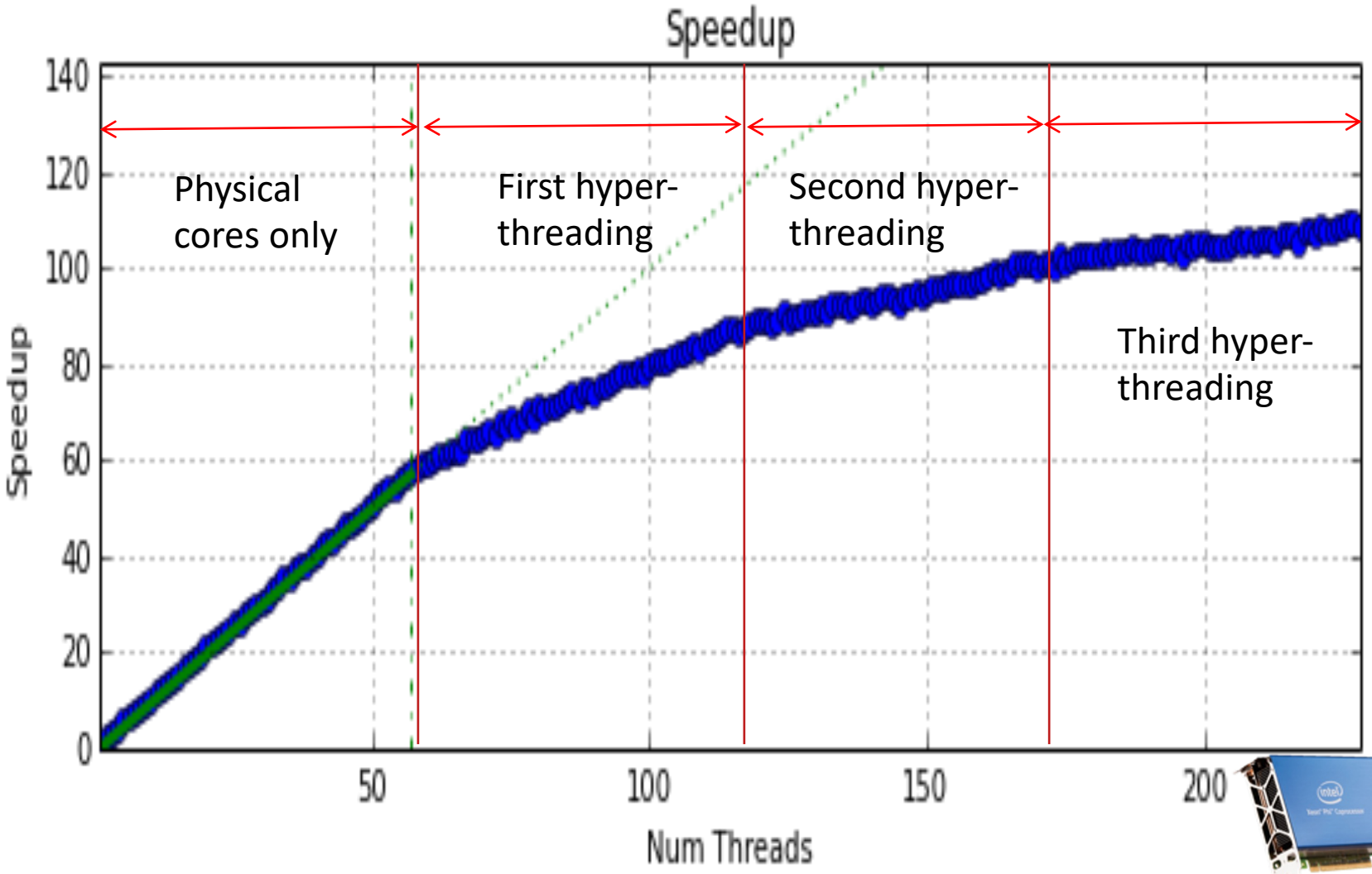

Without MT
Detector geometry &
cross-section tables
MEMORY SPACE
Transient per event data
(tracks, hits, etc.)
AVAILABLE CORES
Active cores
Unused cores
With MT
MEMORY SPACE
AVAILABLE CORES
Active cores

Memory consumption on Intel Xeon Phi



Intel Xeon Phi™ 3120A

Scalability on Intel Xeon Phi

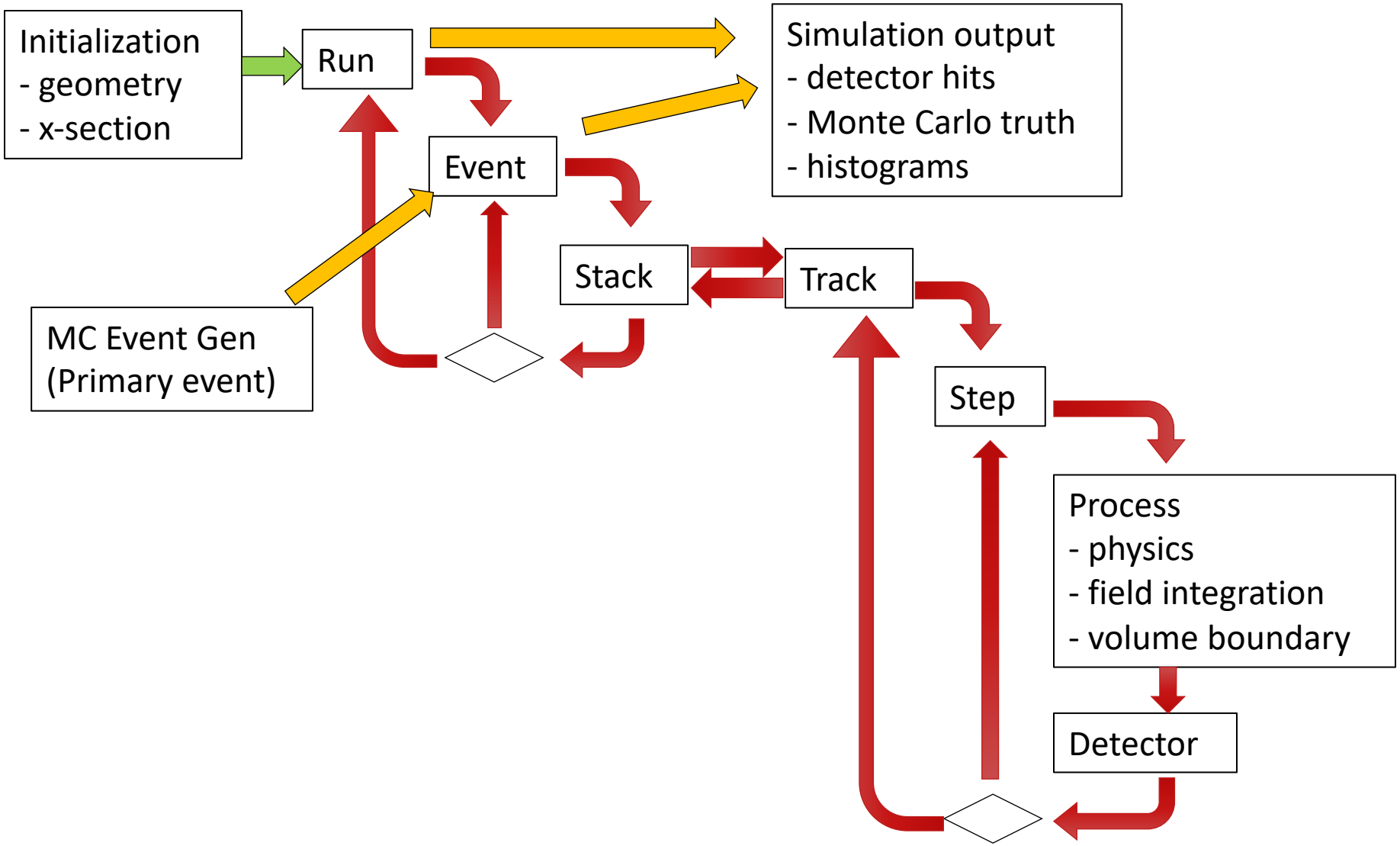


Intel Xeon Phi™ 3120A

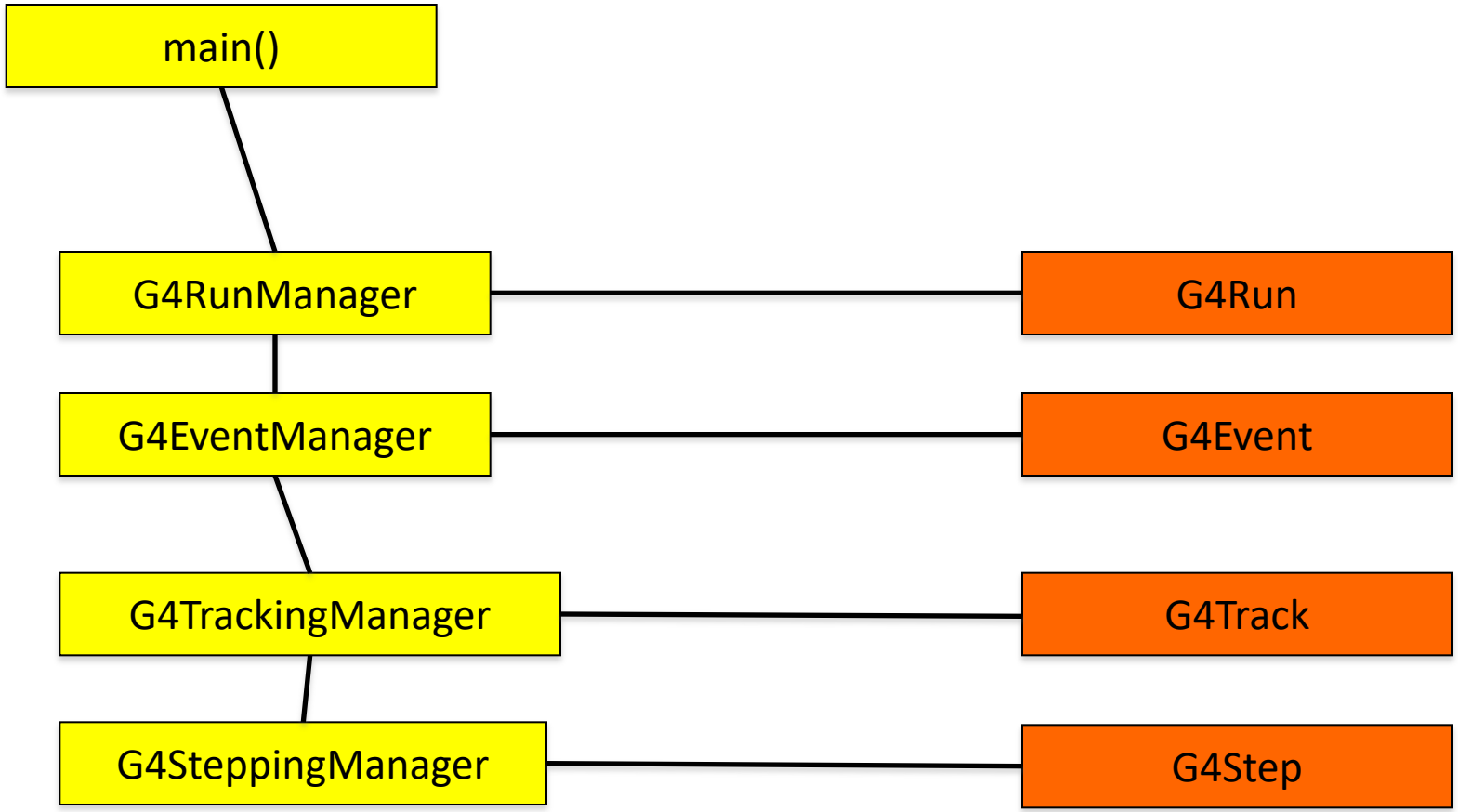
Geant4 evolutions in parallelization

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.2024~)
   - Split an event into sub-events and task them separately
   - Sub-event :
     - Group of tracks of selected kinds or getting into a particular detector component
   - Suitable for heterogeneous hybrid hardware

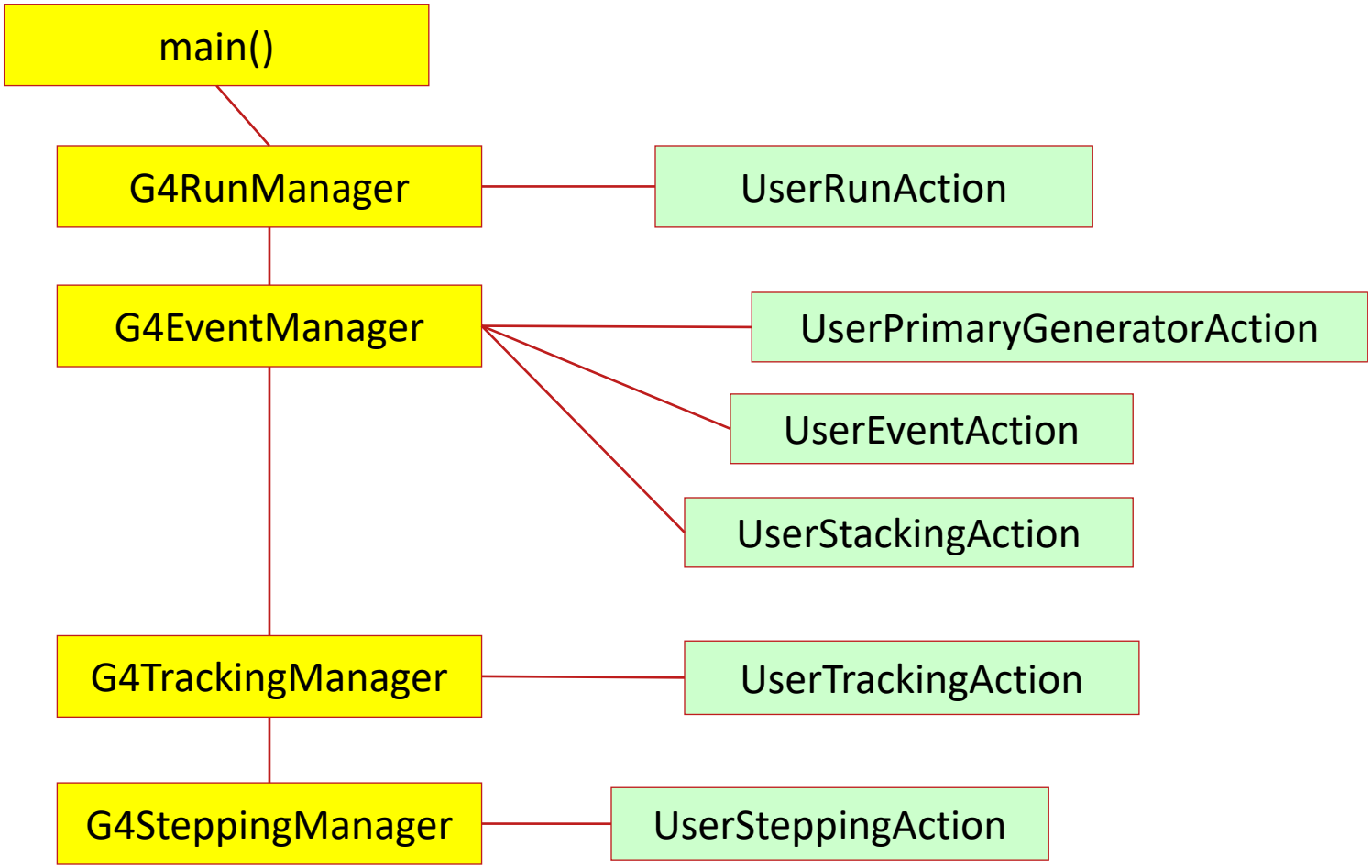
Geant4 as a detector simulation engine



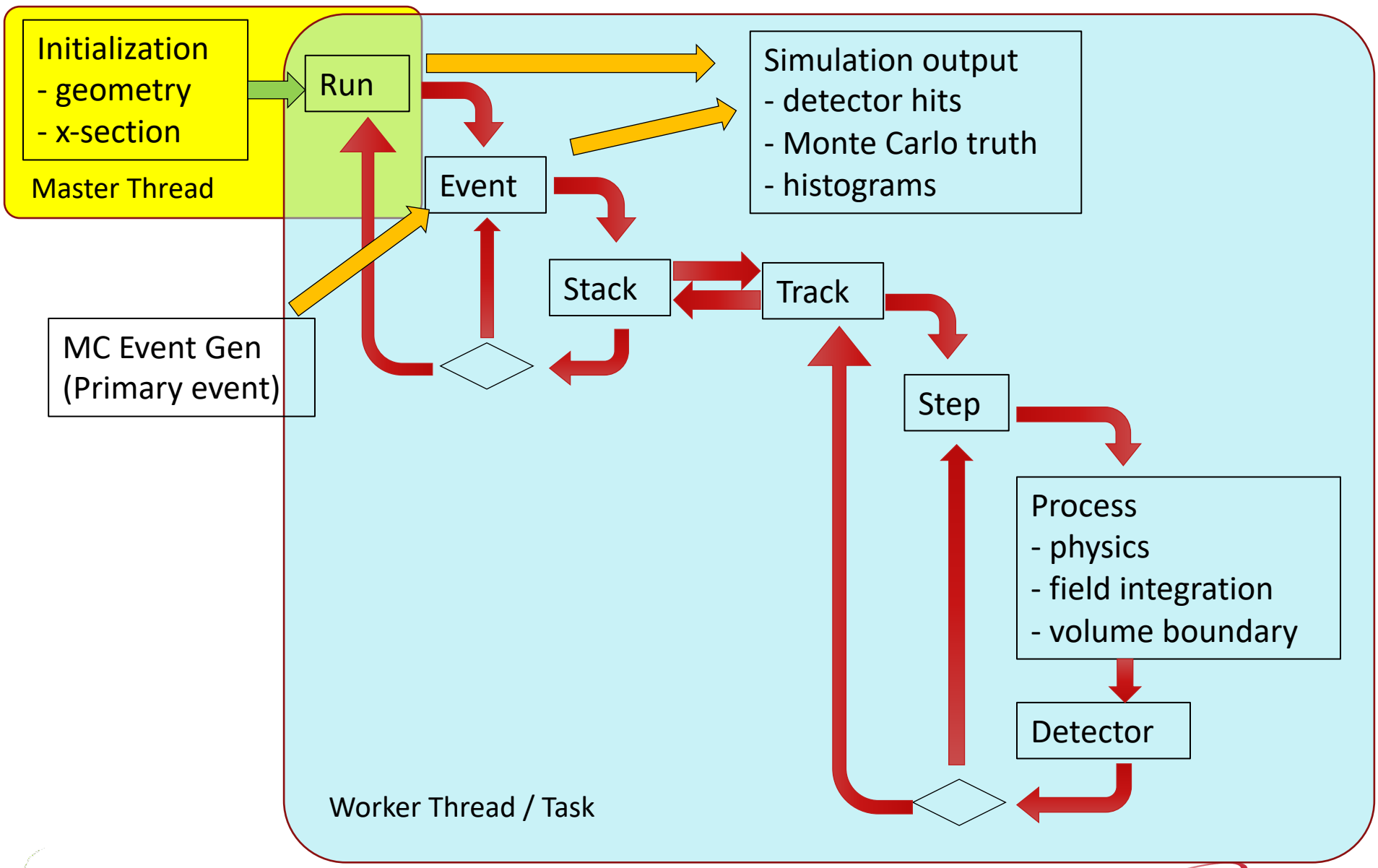
Sequential mode



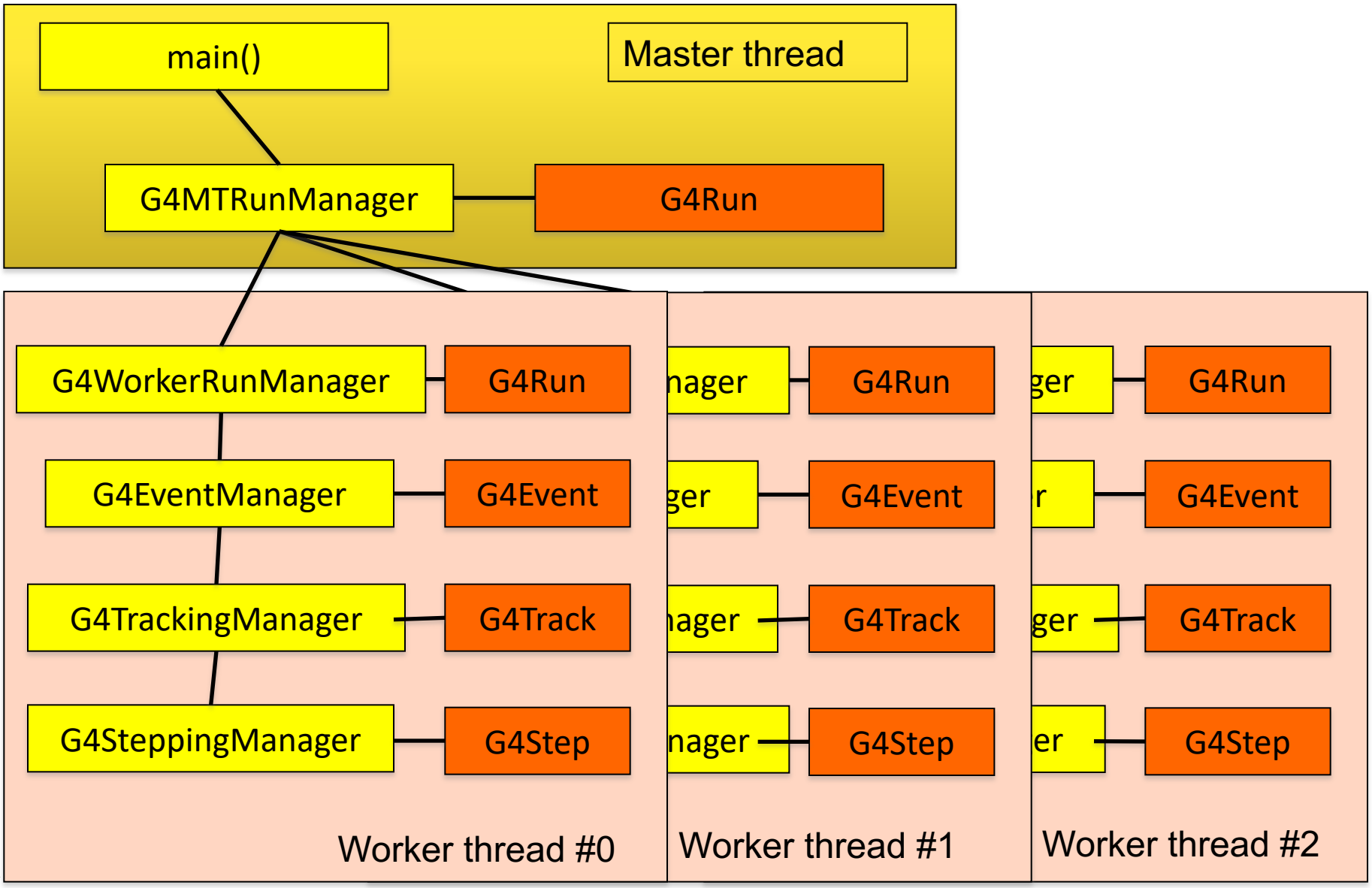
Sequential mode



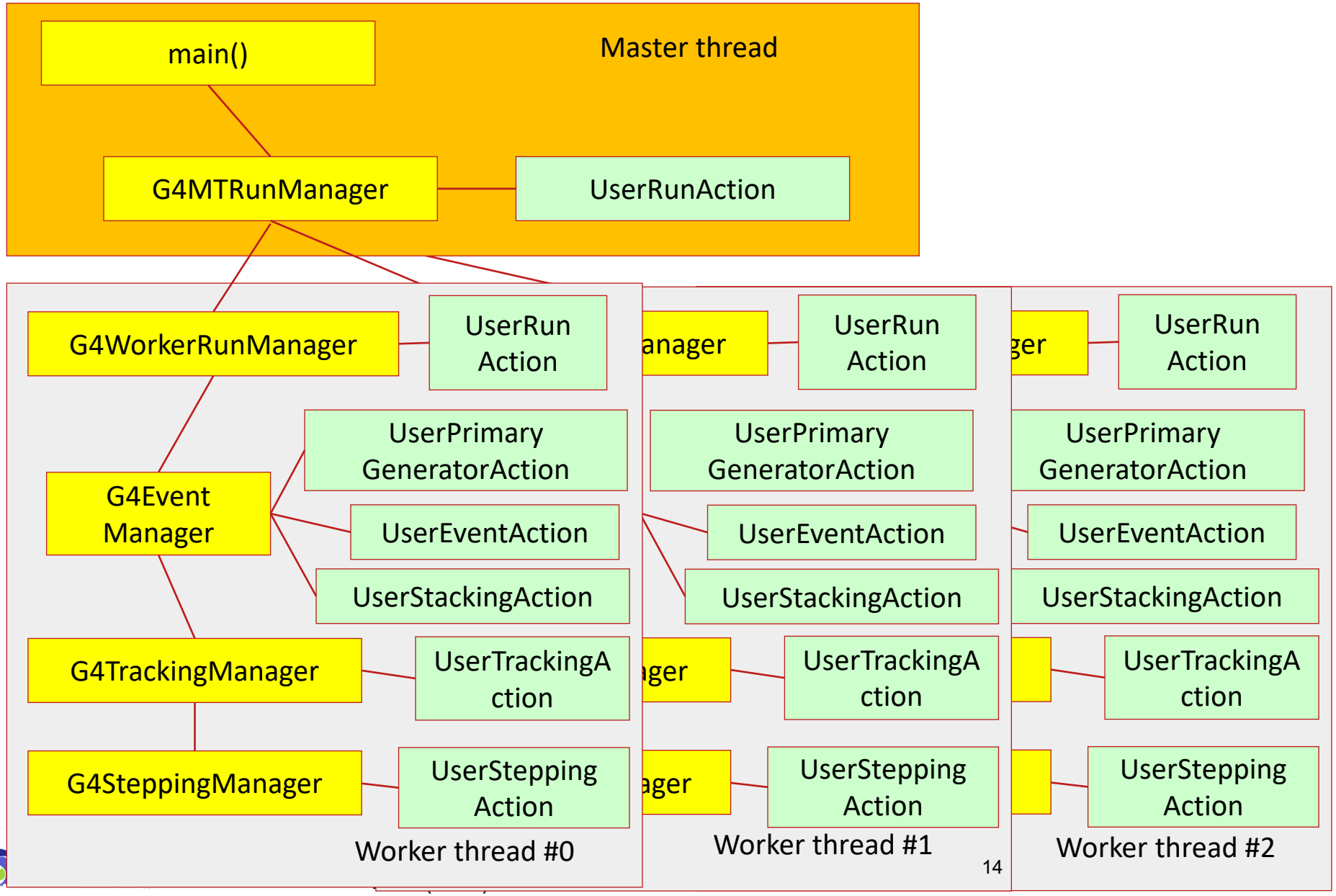
Geant4 as a detector simulation engine



Multi-threaded mode



Multi-threaded mode



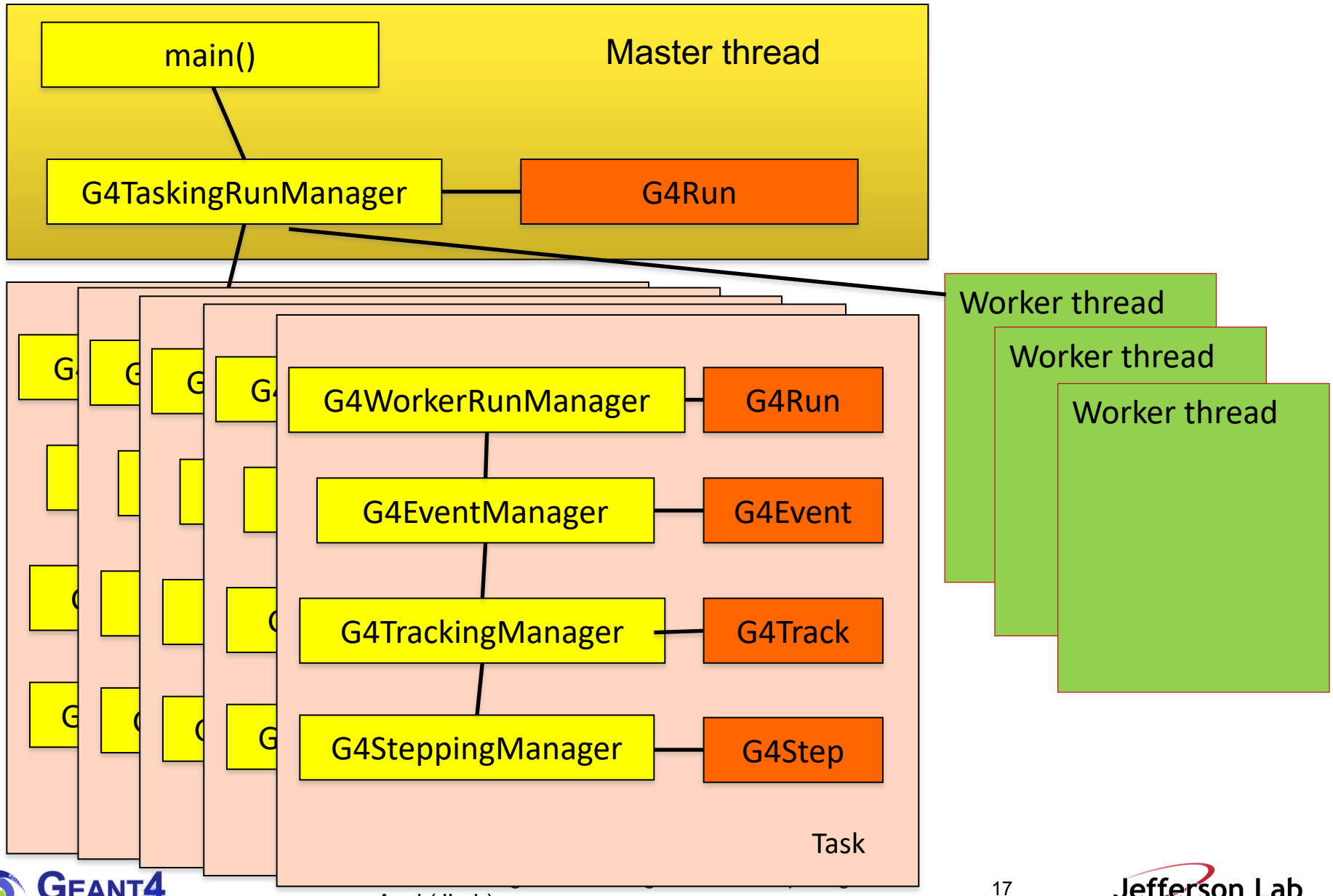
Shared? Thread-local?

- 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.
  - Classes that have cache should be 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 most of 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.

Task-based parallel mode

- Since Geant4 version 11.0.
- Decoupling task (event loop) from thread.
- Geant4 supports tasking model with two different backend threading libraries.
  - PTL (Parallel Tasking Library) : light-weight C++11/14 tasking system
  - TBB (Intel Thread Building Blocks)
- Typically, number of tasks is much larger than the number of available threads.
  - For flexible load-balancing
- User code that works with the original multithreading mode does not need any migration.

Task-based parallel mode



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.

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- Multithreading in Geant4
- Shell variables and UI commands for multithreading
- Sub-event parallelism
- Heterogeneous computing

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

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**".

/run/eventModulo <N> <seedOnce>

- Set the event modulo for dispatching events to worker threads
  - Each worker thread is tasked to simulate <N> events and then comes back to G4MTRunManager for next set.
- If it is set to zero (default value), N is roughly given by this.
  - N = int(sqrt(number_of_events / number_of_threads))
- The value N may affect on the computing performance if N is too small compared to the total number of events.
- The second parameter <seedOnce> specifies how frequent each worker thread is seeded by the random number sequence centrally managed by the master G4MTRunManager.
  - If <seedOnce> is set to 0 (default), seeds that are centrally managed by G4MTRunManager are set for every event of every worker thread. This option guarantees event reproducibility regardless of number of threads.
  - If <seedOnce> is set to 1, seeds are set only once for the first event of each run of each worker thread. Event reproducibility is guaranteed only if the same number of worker threads are used. On the other hand, this option offers better computing performance for applications with relatively small primary particle energy and large number of events.

Number of worker threads

- You can specify the number of worker threads.
  - They do not include master thread or visualization thread.
- Shell environment variable *G4FORCENUMBEROFTHREADS* will overwrite the following alternative settings. *G4FORCENUMBEROFTHREADS* can be an integer or a keyword "max". If "max" is specified, Geant4 uses all threads of the machine including all hyper threads.
- UI command */run/numberOfThreads, /run/useMaximumLogicalCores*
  - This UI command has to be issued at *PreInit*> state.
- *G4RunManager::SetNumberOfThreads(G4int)*
  - This method must be invoked prior to *G4RunManager::Initialize()*.

- UI command */run/pinAffinity*
  - Locks worker threads to specific logical cores.
  - You may gain some performance, but at your own risk!

Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- Multithreading in Geant4
- Shell variables and UI commands for multithreading
- Sub-event parallelism
- Heterogeneous computing

GEANT4
A SIMULATION TOOLKIT

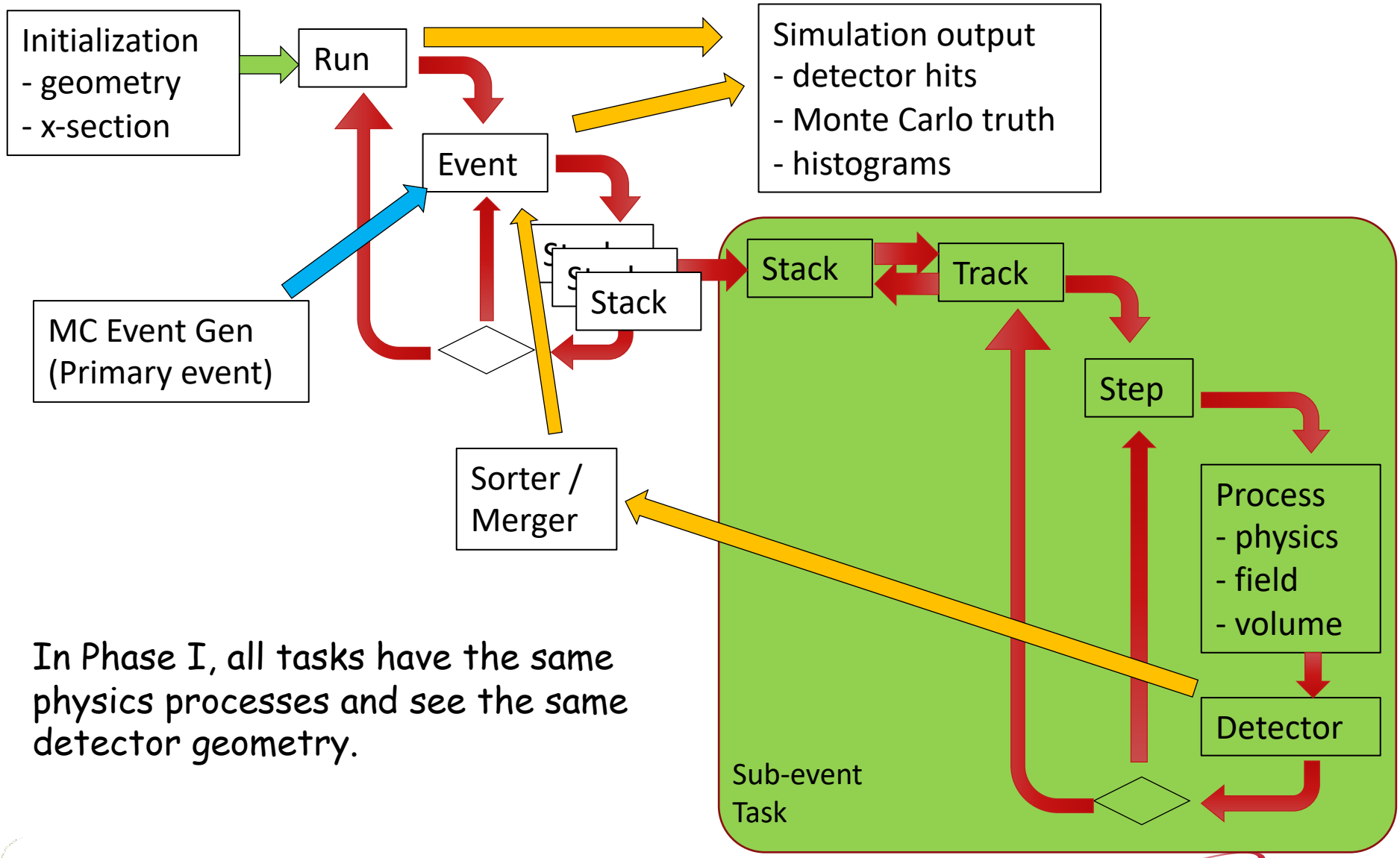
U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

Sub-event parallelism in Geant4

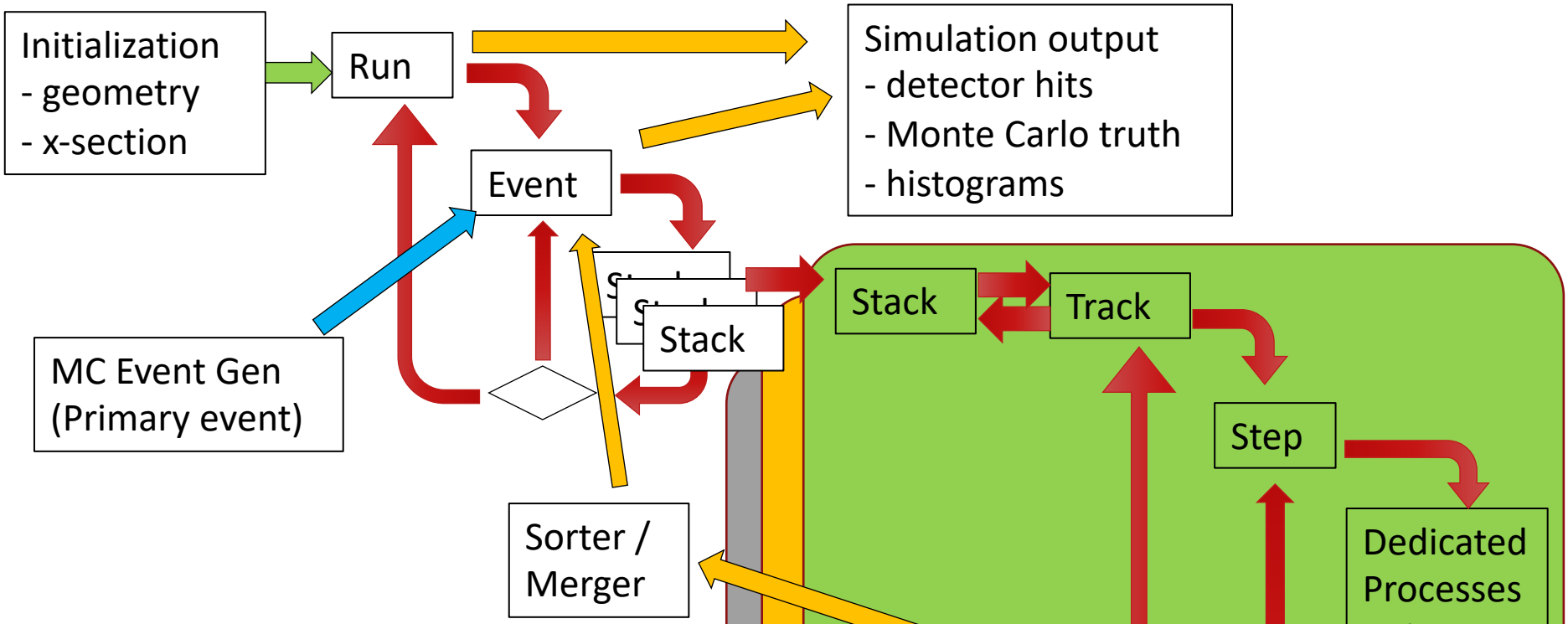
- Split an event into sub-events and task them separately
  - Sub-event :
    - Group of tracks of selected kinds or getting into a particular detector component
  - Suitable for heterogeneous hybrid hardware
- Sub-event parallelism will be introduced without forcing user's code to migrate.
  - Sequential, threading and tasking modes will work fine as they do now.
- Goal is to utilize heterogeneous hardware architectures.
  - Recent studies showed that each GPU process should have strictly limited scope to achieve the desired performance.
    - Limited physics coverage, particular particle types, specific geometry / material
    - E.g. optical photon transport, EM shower in calorimeter
- While the master thread manages the entire event, it tasks sub-events to worker threads of different architectures that are suitable for their unique capabilities.

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



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

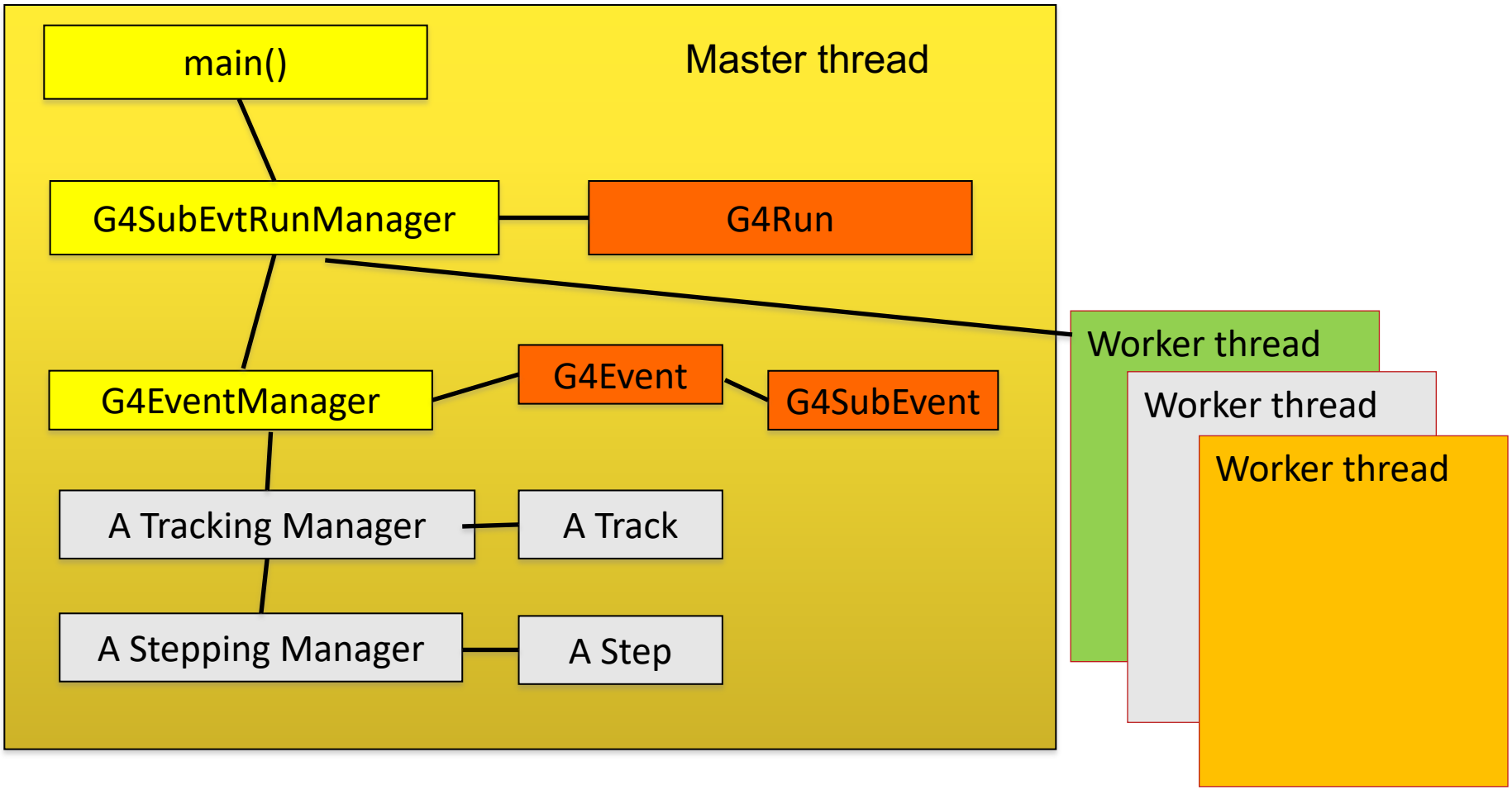
Task-based sub-event parallel mode (Phase II)



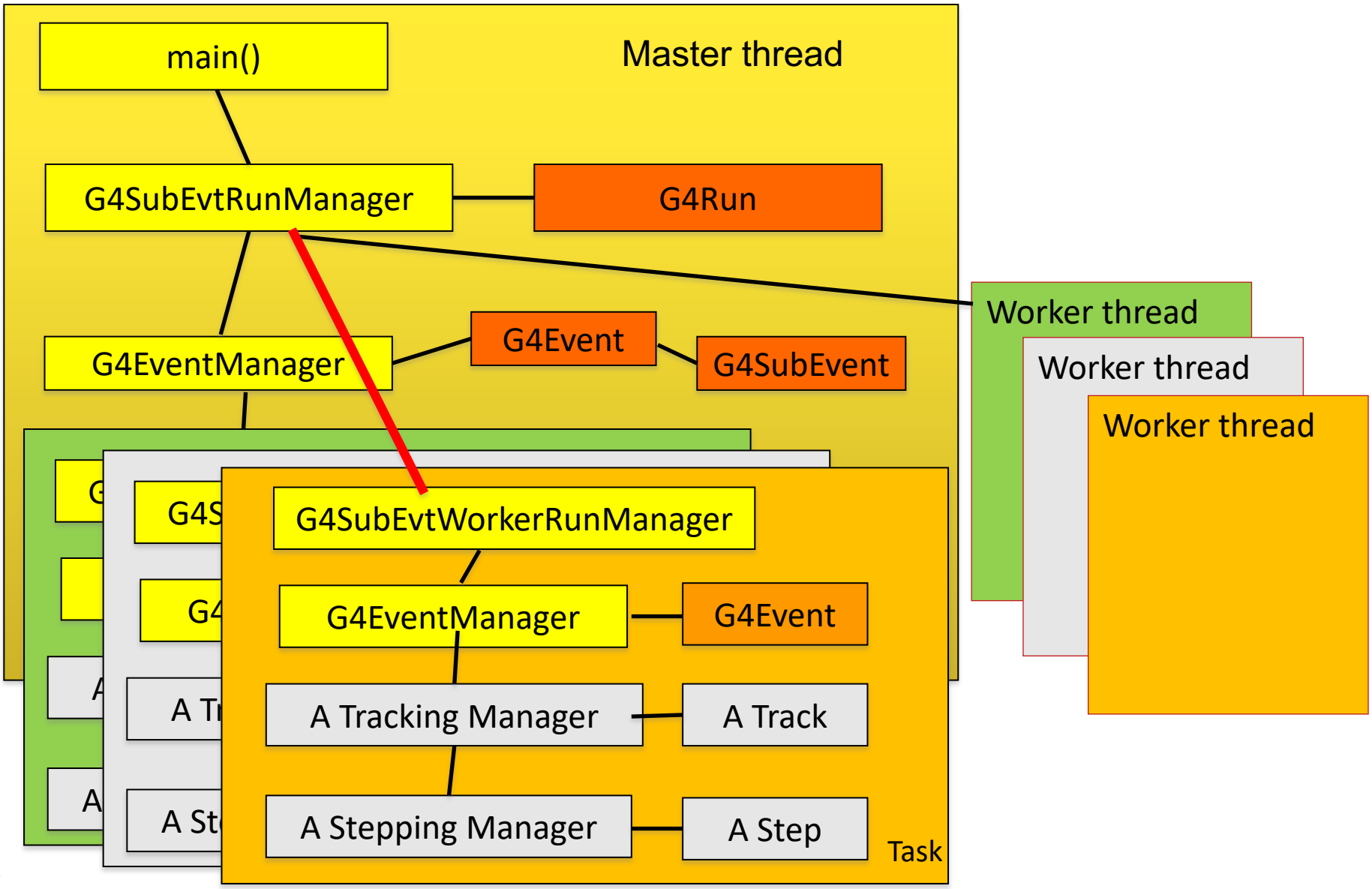
In Phase II, each task has only the necessary physics processes and sees the limited detector geometry that are necessary for that particular task.

- Essential for heterogeneous simulation

Sub-event parallel mode



Sub-event parallel mode



Contents

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- Multithreading in Geant4
- Shell variables and UI commands for multithreading
- Sub-event parallelism
- Heterogeneous computing

GEANT4
A SIMULATION TOOLKIT

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

Heterogeneous computing

- Heterogeneous computing is computing which uses not only CPUs but with other hardware architectures such as Coprocessor, GPU (Graphics Processing Unit), TPU (Tensor Processing Unit), NPU (Neural Processing Unit), FPGA (Field Programable Gate Array), ASIC (Application-Specific IC), etc.
- Goal is to optimize power, performance and cost simultaneously.
- Taking advantage of increasing heterogeneity in computing resources.
  - For example, in exa-scale computers at DOE facilities with GPUs

AMD

NVIDIA

Intel

AMP

GPU codes

- Recent studies showed that each GPU process should have strictly limited scope to achieve the desired performance.
  - Limited physics coverage, particular particle types, specific geometry / material
  - E.g. optical photon transport, EM shower in calorimeter
- Opticks : open-source project for optical photon transport by integrating Nvidia OptiX ray-tracing package
  - Developed by Simon Blyth for Daya Bay and JUNO experiments
    - https://bitbucket.org/simoncblyth/opticks
  - examples/advanced/CaTS demonstrates the use of Opticks in Geant4
    - CPU transports all particles except optical photon
- AdePT, Celeritas : R&D projects for $e^-/e^+/\gamma$ EM shower in calorimeter
  - AdePT (CERN) : G4HepEM
  - Celeritas (ORNL, FNAL, etc.) : Standard EM
  - Both projects showed the limitation is not in physics but in geometry navigation.
    - Severe thread divergence due to individual algorithms for each solid shapes.
    - Requires boundary-based geometry description.

Is GPU the silver bullet?

- For the next HL-LHC (and likely for EIC) simulation throughput needs to increase by O(10^2).
- CMS recently measured x2.2 – x2.5 speedup depending on event types if they killed all EM particles instantly at their creations.
  - This means, even if the EM processes on GPU are infinitely fast, you won't gain more than x2.2 – x2.5.
- To make the simulation orders of magnitude faster, fast simulation, a.k.a. shower parameterization, is mandatory.
  - We still need full simulation to validate / optimize fast simulation.
  - AI/ML on GPU would help a lot.
    - Optimizing fast simulation parameters
    - Hit pattern generation