


GEANT4
A SIMULATION TOOLKIT

Version 11.2

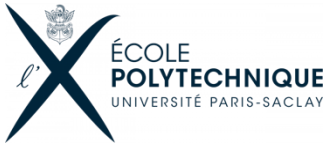
# Multithreading .. and tasking


**John Apostolakis** (CERN)


Slides from **Makoto Asai** (Jefferson Lab) – with small changes

Outline:

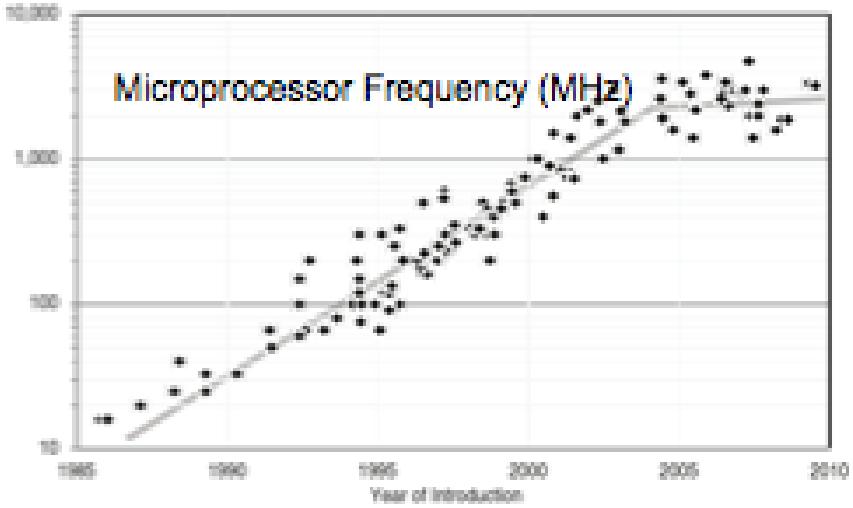
- Introduction
- Multithreading in Geant4 : the basics
- UI commands for multithreading

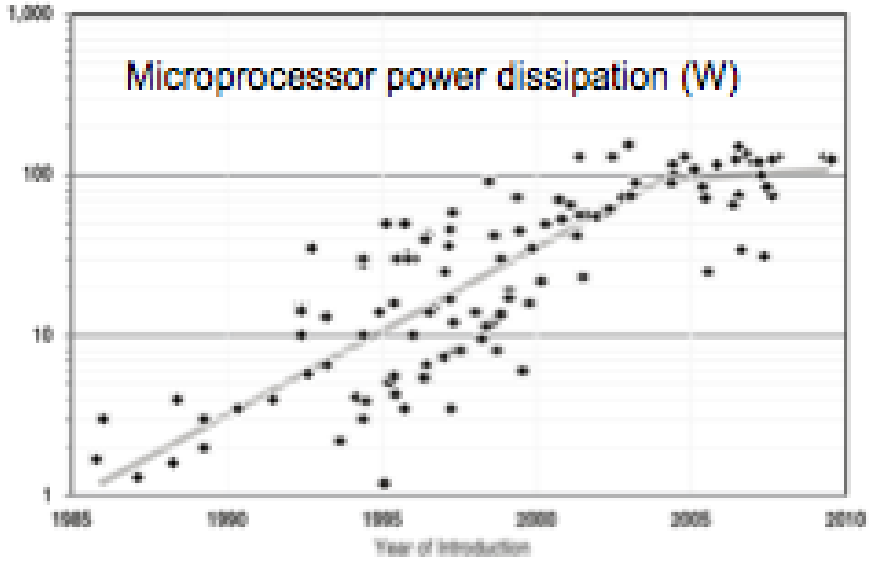





# INTRODUCTION

# The challenges of many-core era

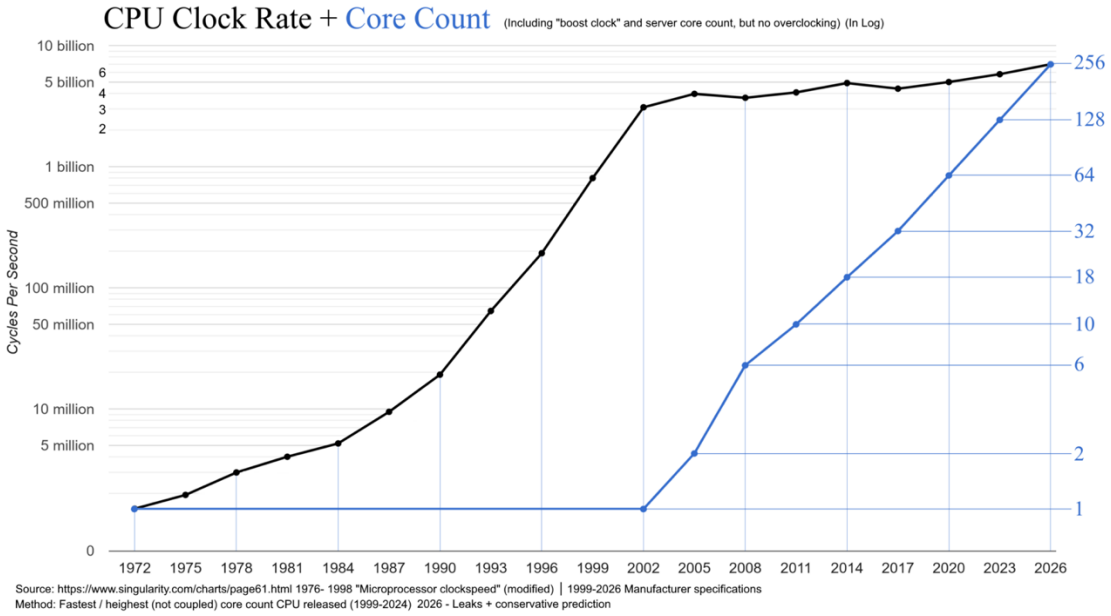



- Higher frequency of CPU **increase power consumption**
- Reached plateau around 2005
- No more increase in CPU frequency
- However number of transistors per chip continues to grow
- Multi/Many-core era
- Note: quantity memory you can buy with same $ scales slower
- **Expect:**
- Many core (double/2yrs?)
- Single core performance increases slowly
- Less memory/core
- New software models need to take these into account: increase parallelism

**CPU Clock Frequecy** 1and 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,"

# The challenges of many-core era




Graph by RenderingBlenders , reused under Creative Commons License version 1. Original in Wikimedia

- Higher frequency of CPU **increase power consumption**
- Reached plateau around 2005
- No more increase in CPU frequency
- However number of transistors per chip continues to grow
- Multi/Many-core era
- Note: quantity memory you can buy with same $ scales slower
- **Expect:**
- Many core (double/2yrs?)
- Single core performance increases slowly
- Less memory/core
- New software models need to take these into account: increase parallelism

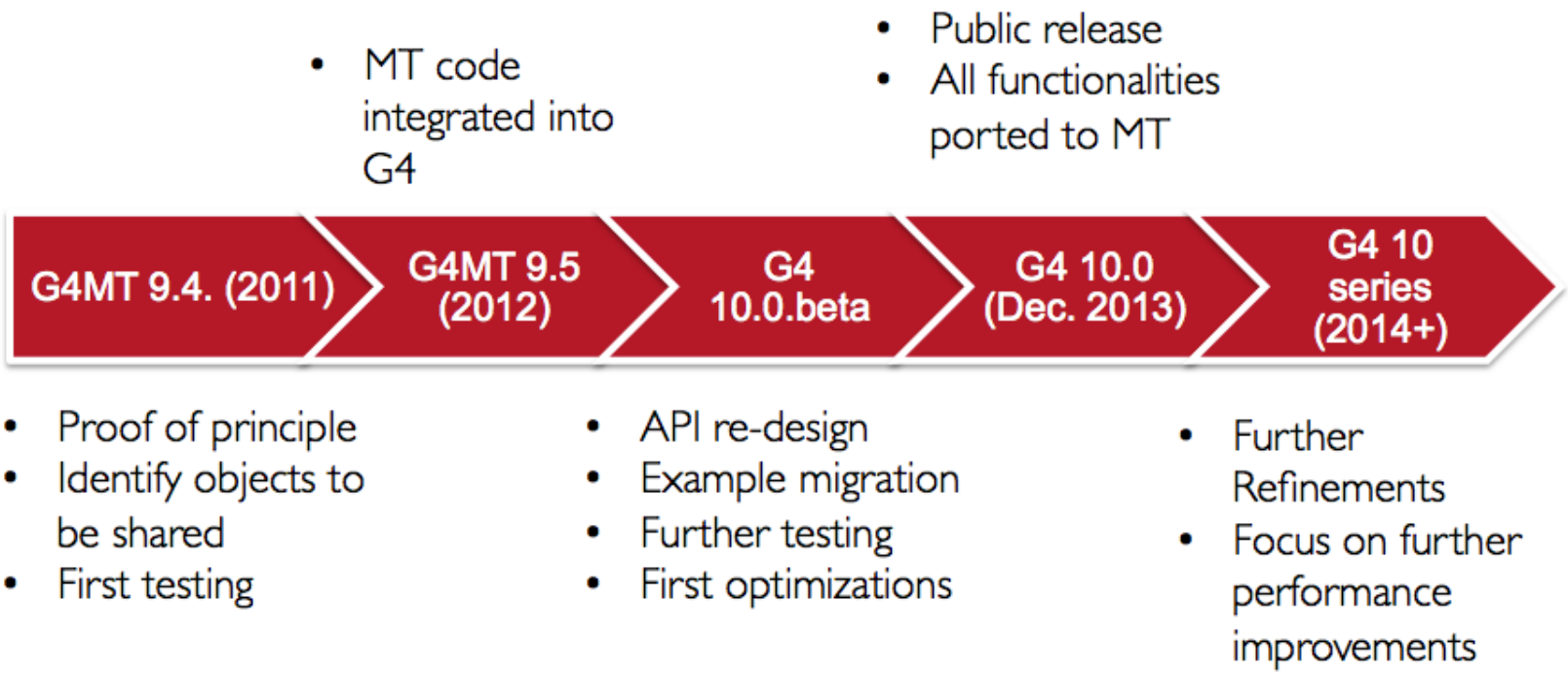
# In Brief

- Modern CPU architectures: need to introduce **parallelism**
- Memory and its access will limit number of concurrent processes running on single chip
- Solution: add parallelism **in the application code**

- Geant4 needs back-compatibility with user code and **simple approach** (physicists != computer scientists)
- **Events are independent**: each event can be simulated separately

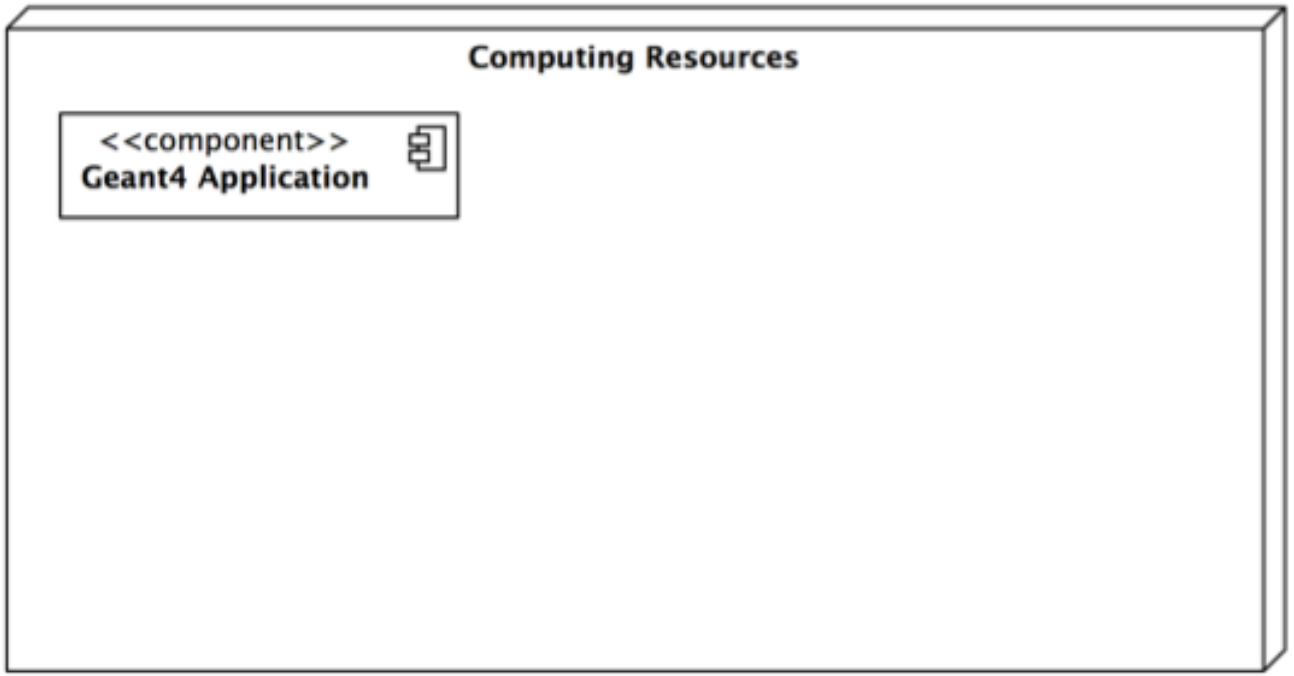
- Multi-threading for event level parallelism was the natural choice

# Geant4 Multi Threading – origin & evolution of capabilities



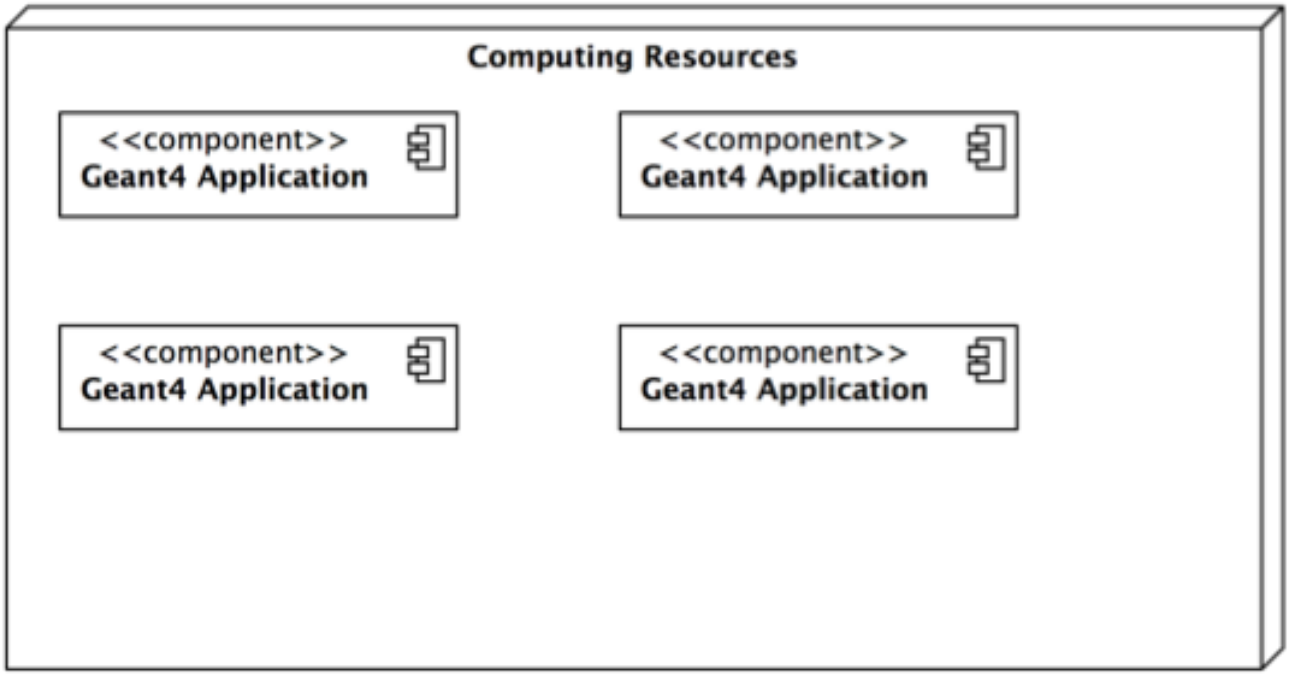
# What is a thread?

Sequential application



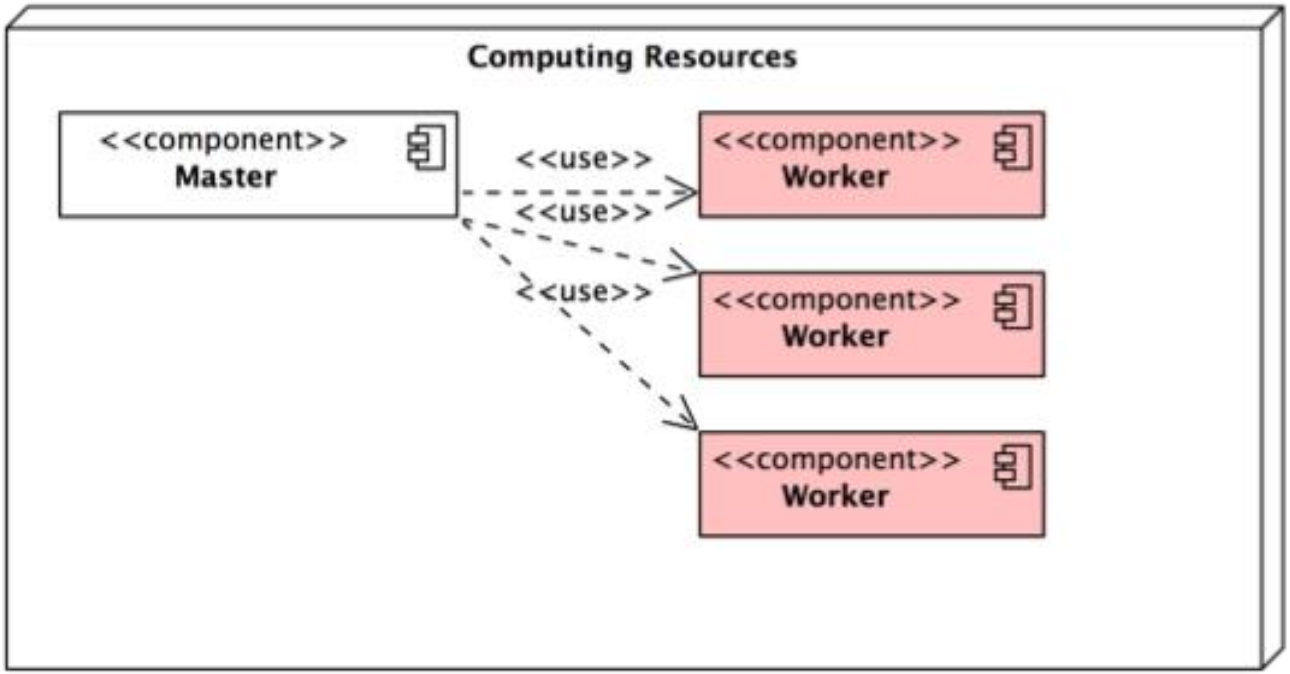
# An alternative to using threads

Sequential application: start N (cores/CPUs) copies of application if fits in memory



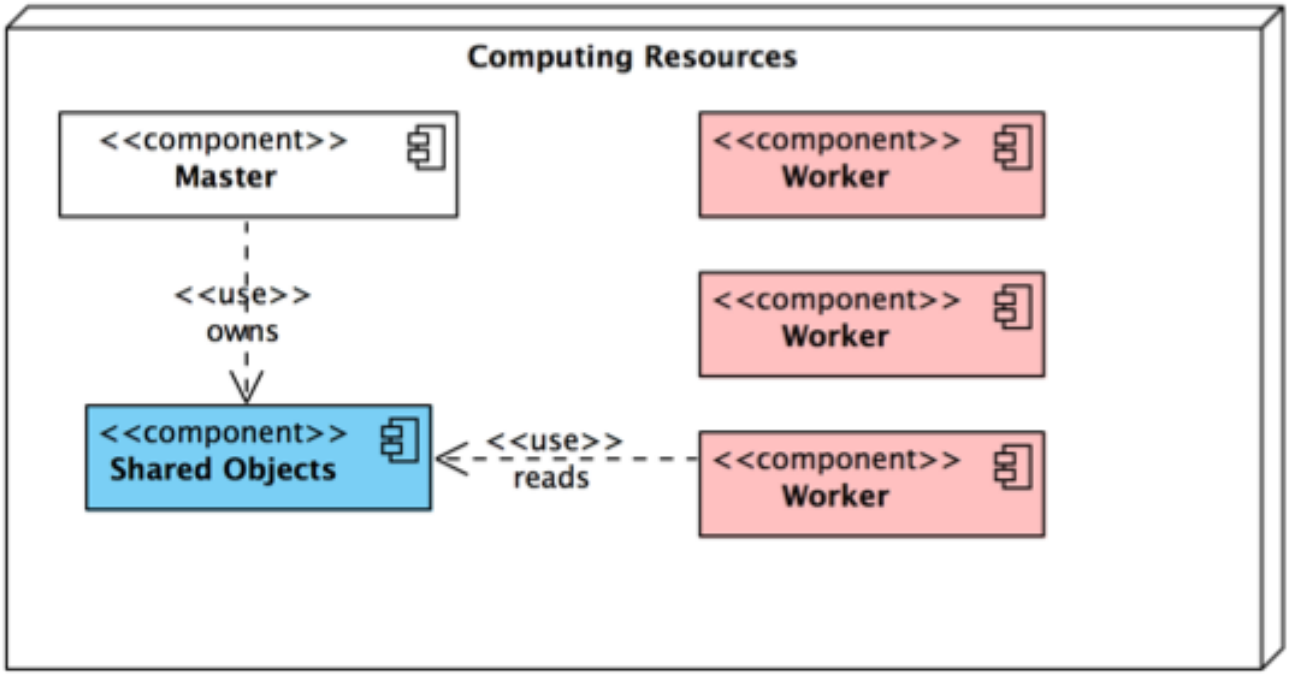
# What is a thread?

MT Application: single application starts threads. For G4: application (master) controls workers that do simulation, no memory sharing now, each worker is a copy of the application



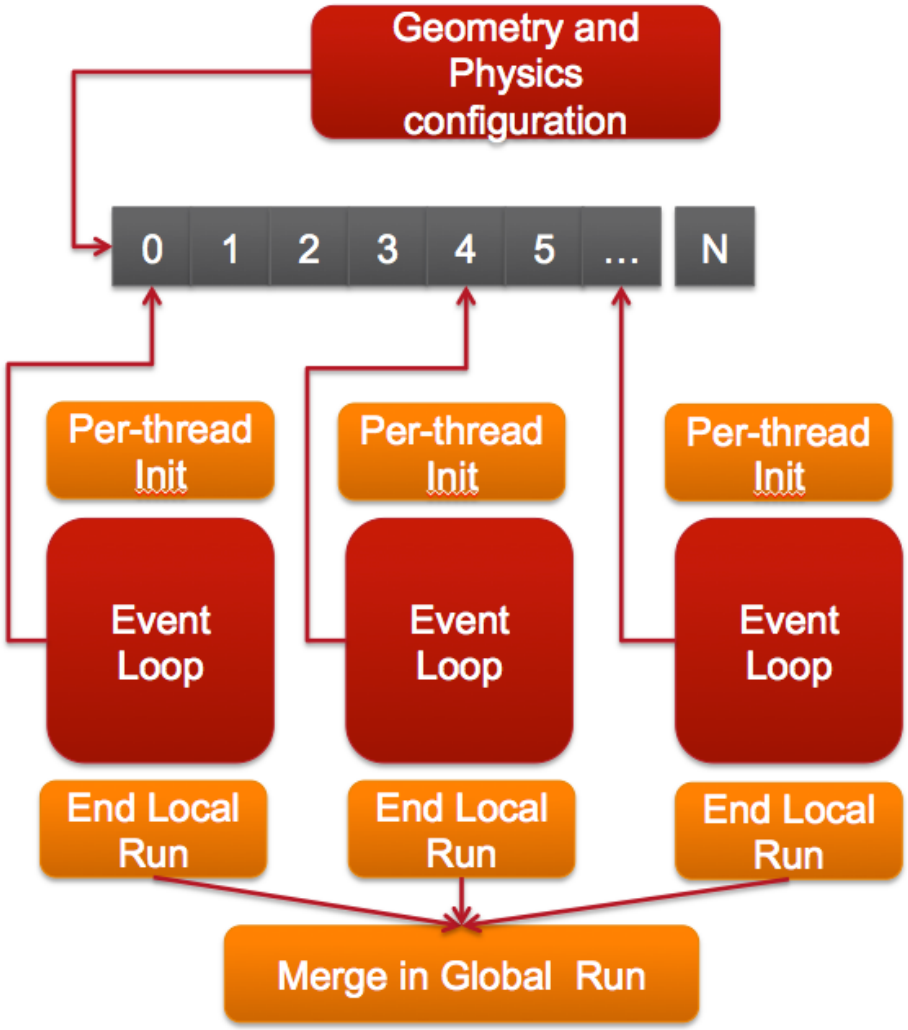
# What is a thread?

Memory reduction: introduce shared objects, memory of N threads is less than memory used by N copies of application



# THE BASICS OF MULTI-THREADING IN GEANT4

# General Design




Per-event seeds pre-prepared in a "queue"

Threads compete for next event to be processes

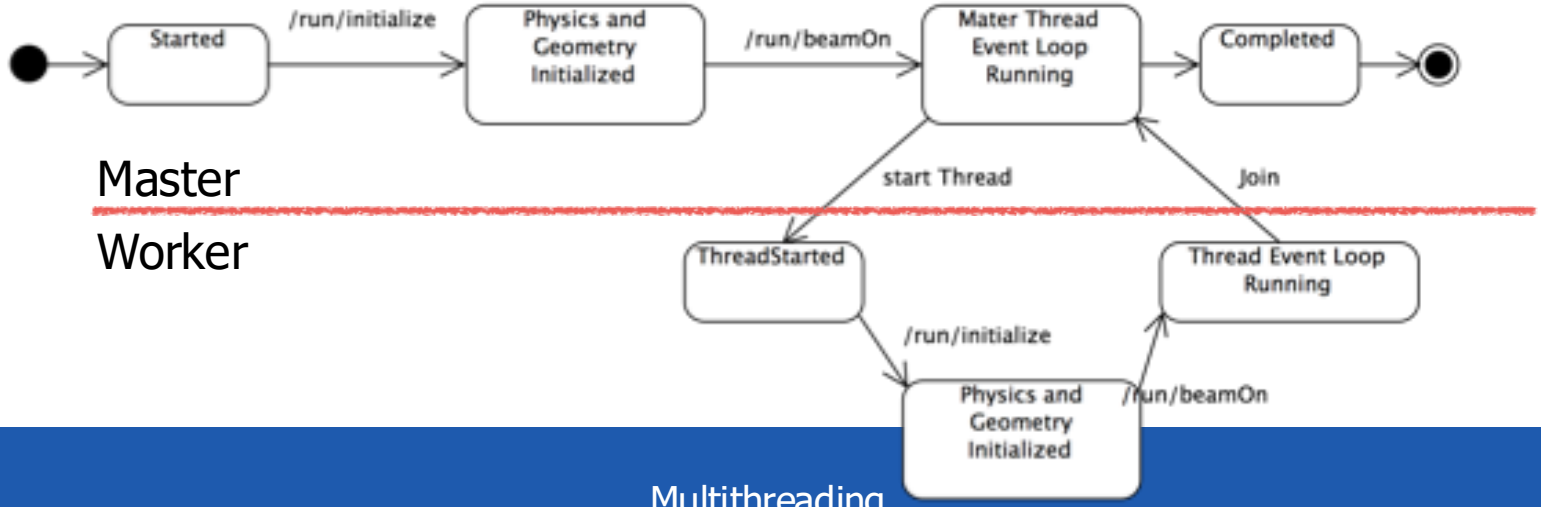
Command line scoring and G4tools automatically merge results from threads

# Simplified Master / Worker Model

- A G4 (with MT) application can be seen as simple finite state machine

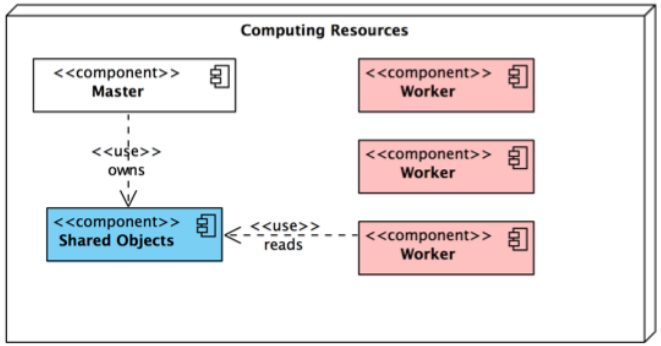
# Simplified Master / Worker Model

- A G4 (with MT) application can be seen as simple finite state machine
- Threads do not exists before first /run/beamOn
- When master starts the first run spawns threads and distribute work



# Shared Vs Thread-local

- To reduce memory footprint threads must share at least part of the objects
- **General rule in G4: threads can share whatever is invariant during the event loop** (e.g. threads do not change these objects while processing events, these are used "read-only")
  - Geometry definition
  - Electromagnetic physics tables

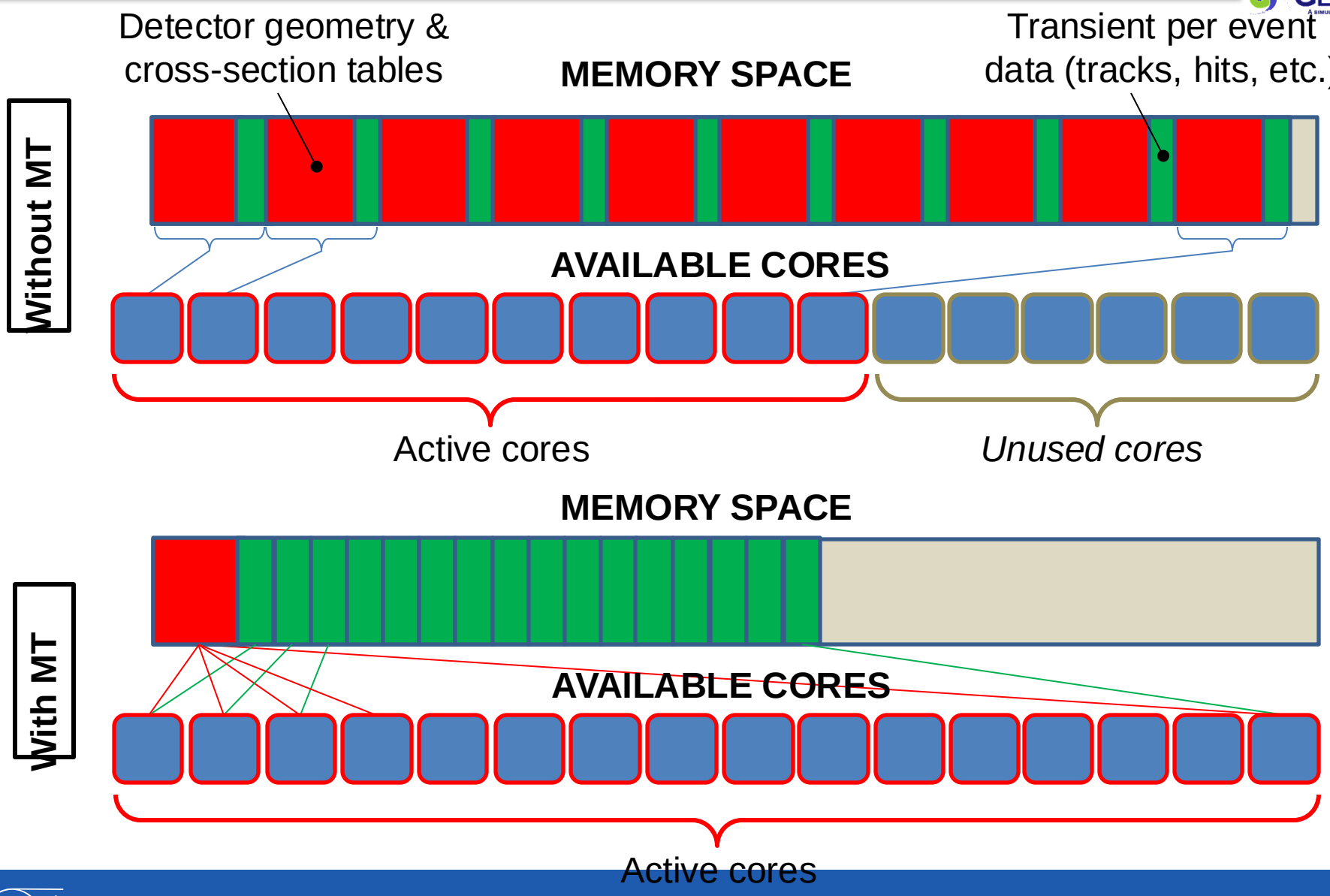


# Shared ? Private?

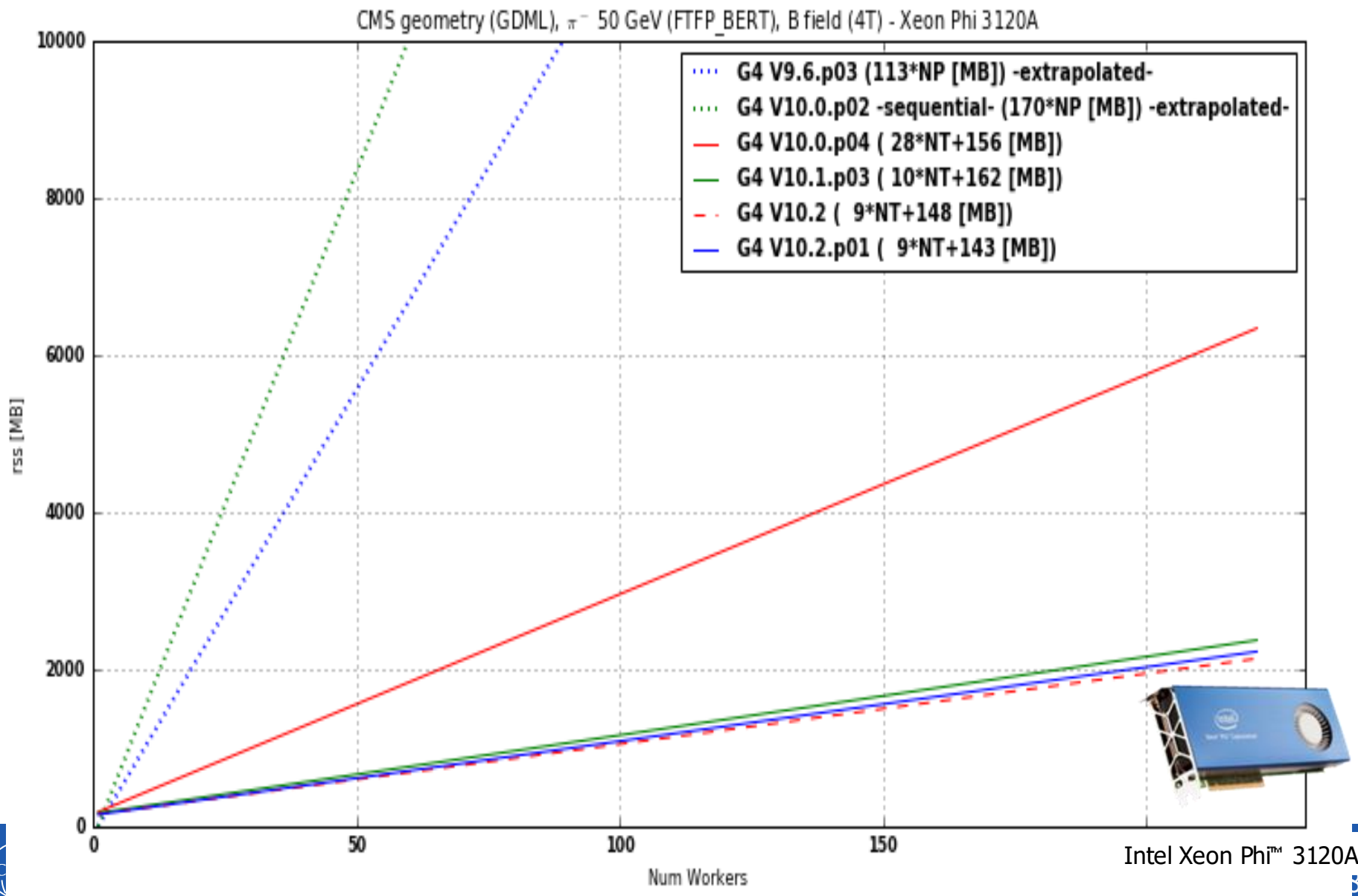
- In the multi-threaded mode, generally saying, data that are stable during the event loop are shared among threads while data that are transient during the event loop are thread-local.

- Shared by all threads
  : stable during the event loop
  - Geometry
  - Particle definition
  - Cross-section tables
  - User-initialization classes

- Thread-local
  : dynamically changing for every event/track/step
  - All transient objects such as run, event, track, step, trajectory, hit, etc.
  - Physics processes
  - Sensitive detectors
  - User-action classes

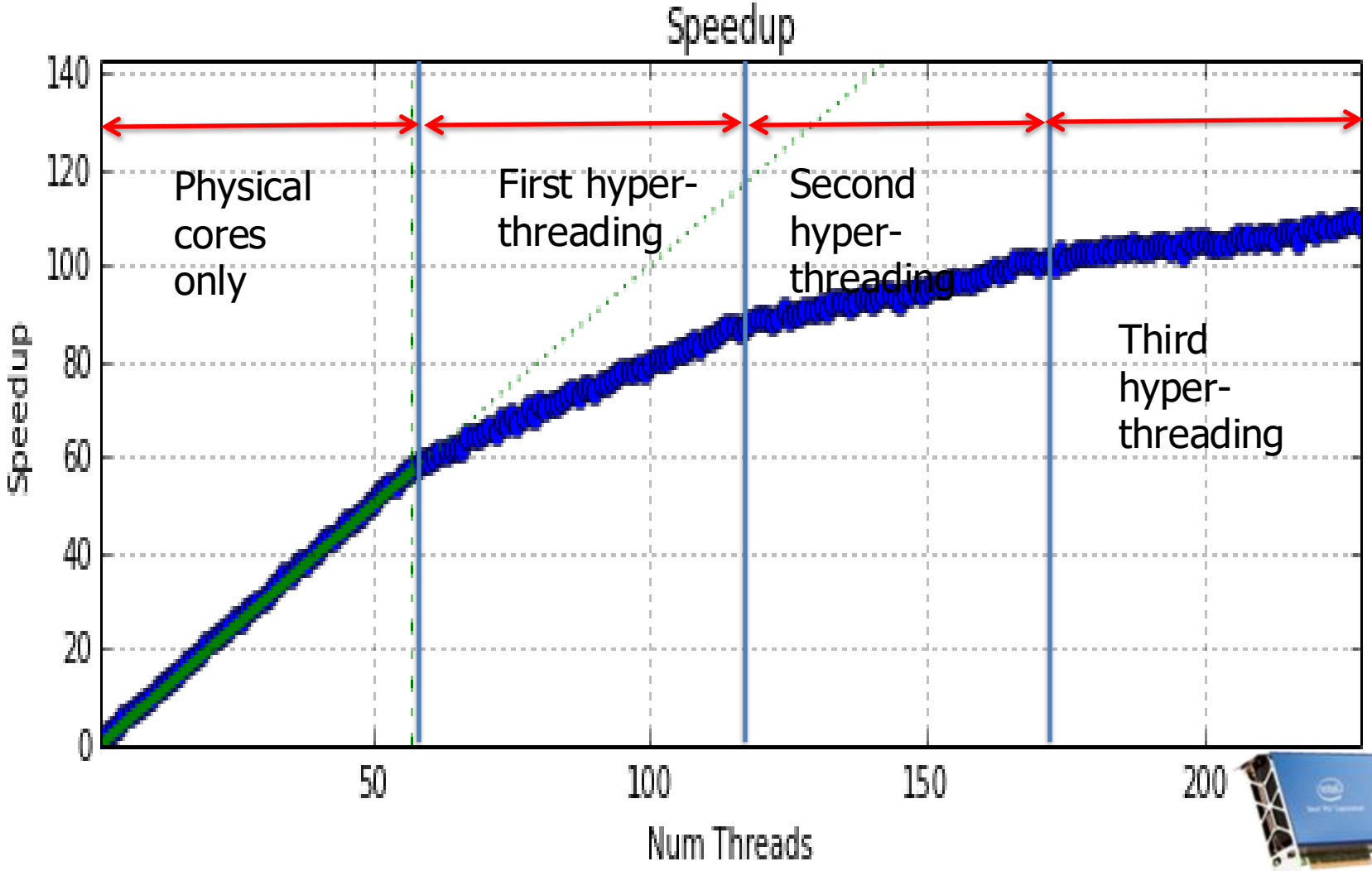

Without MT
Detector geometry & cross-section tables
MEMORY SPACE
Transient per event data (tracks, hits, etc.)
AVAILABLE CORES
Active cores
Unused cores
With MT
MEMORY SPACE
AVAILABLE CORES
Active cores

# Memory consumption on Intel Xeon Phi




Intel Xeon Phi™ 3120A

# Scalability on Intel Xeon Phi



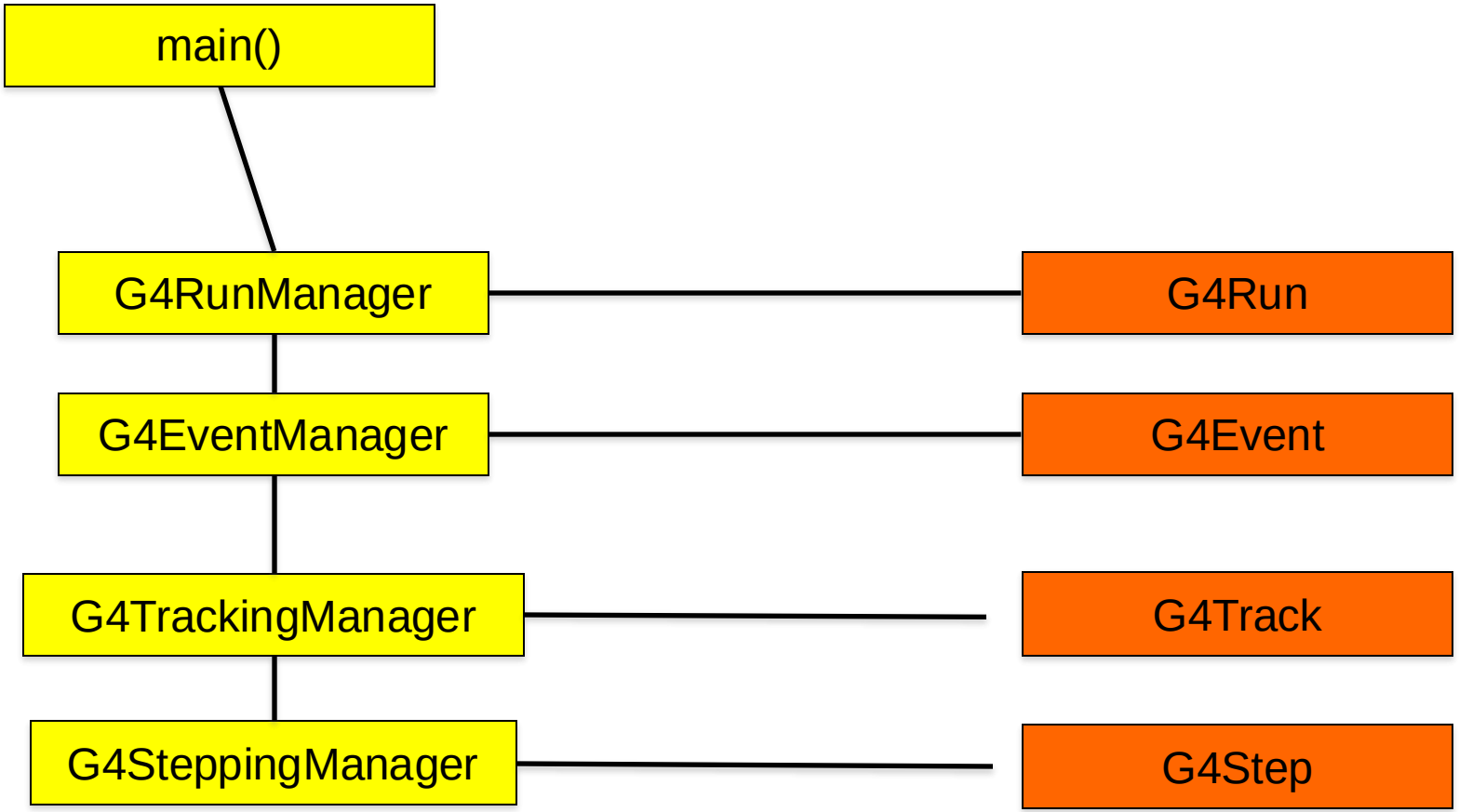



Intel Xeon Phi™ 3120A

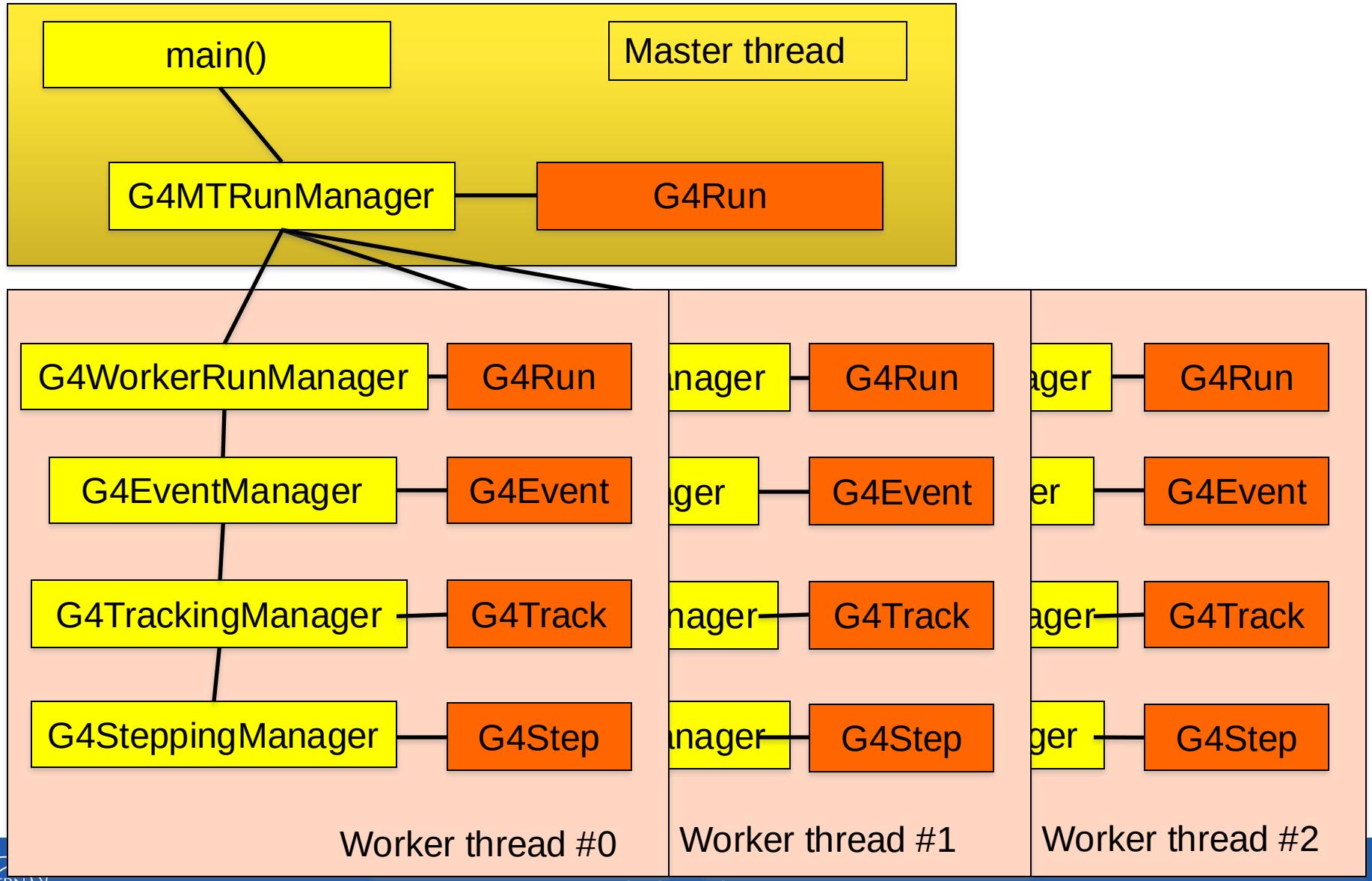
# Shared ? Thread-local?

- In general, geometry and physics tables are shared, while event, track, step, trajectory, hits, etc., as well as several Geant4 manager classes such as EevntManager, 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.

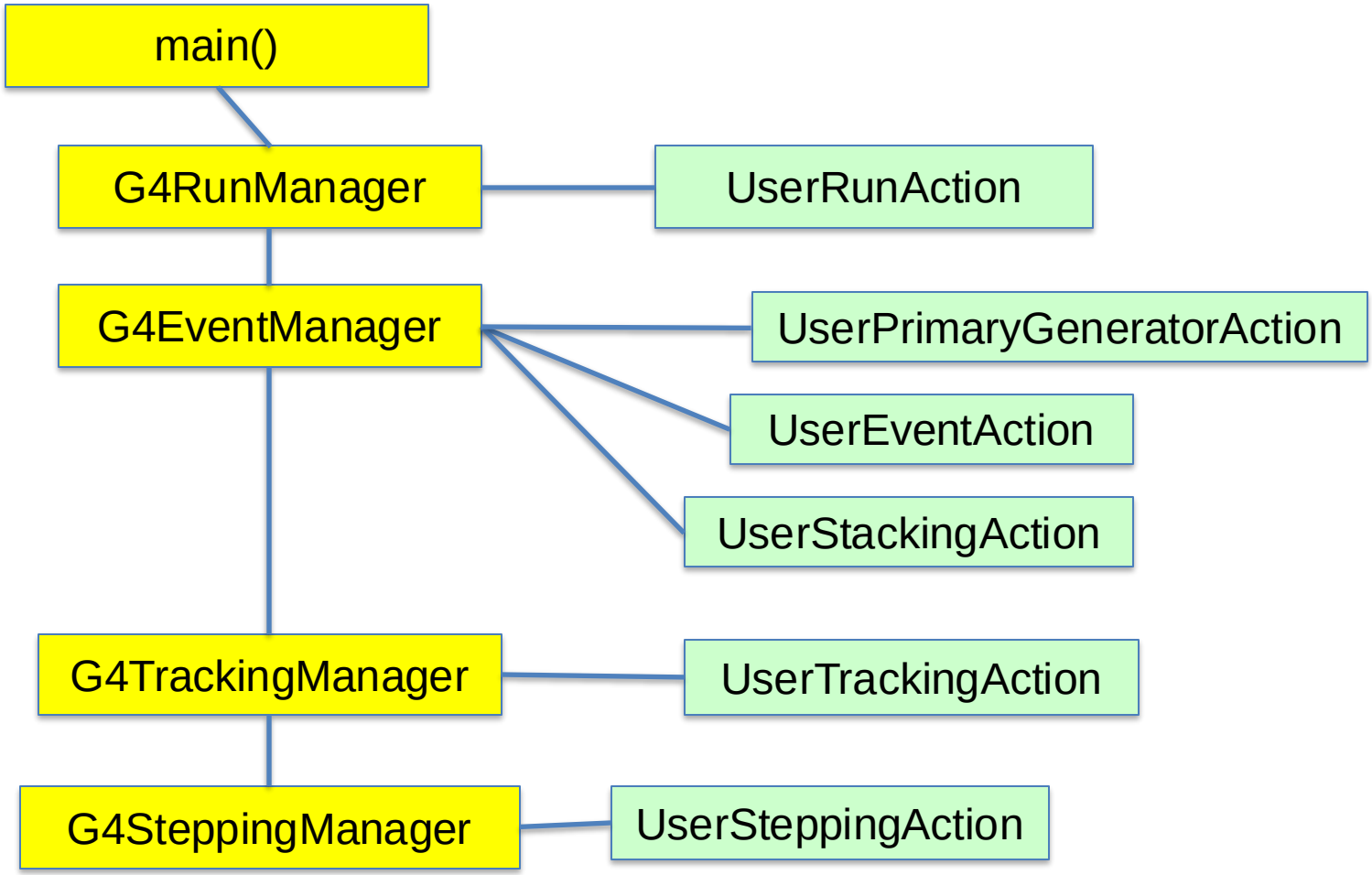
# Sequential mode



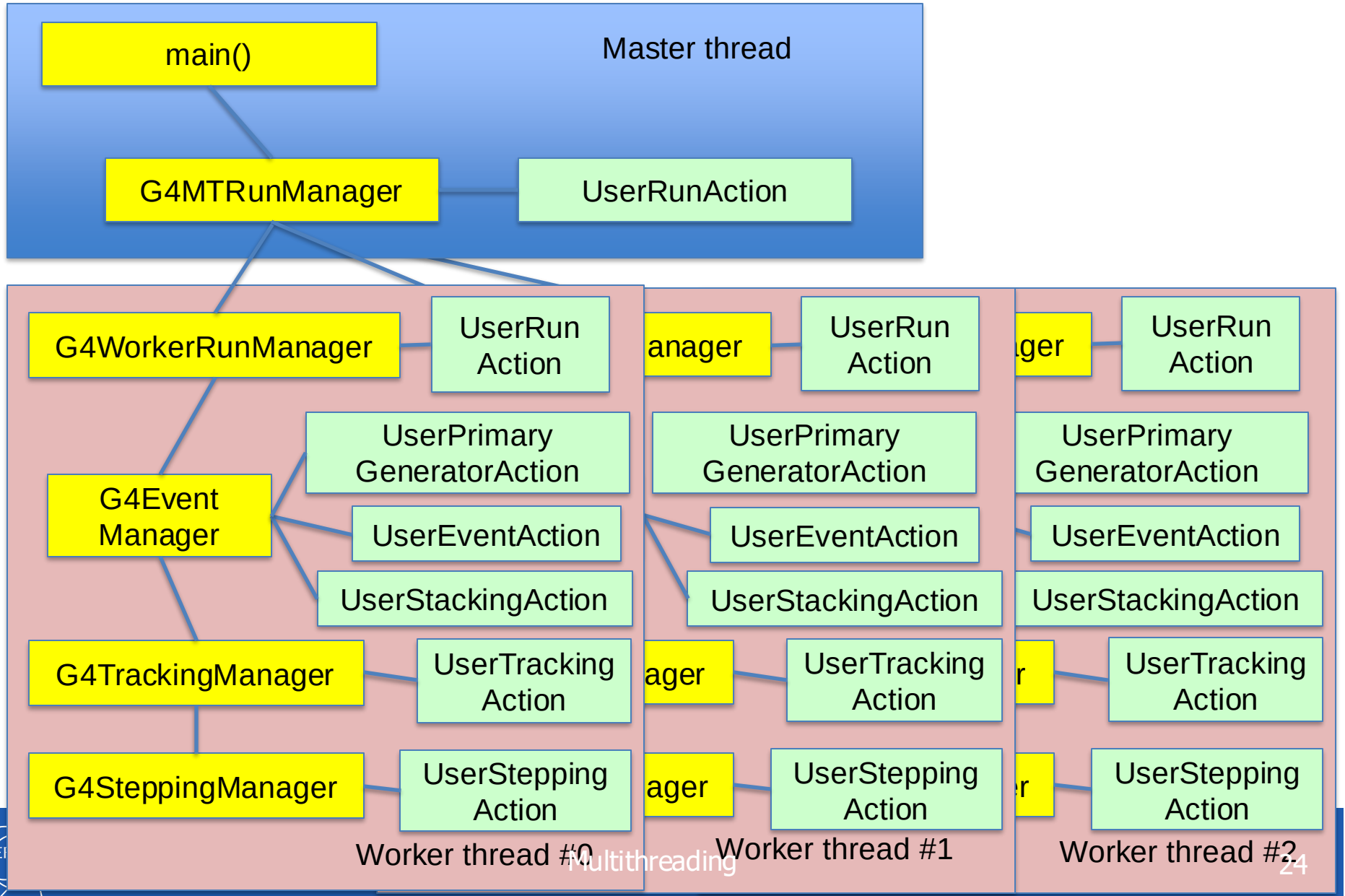
# Multi-threaded mode



# Sequential mode



# Multi-threaded mode



# UI COMMANDS FOR MULTITHREADING

# 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*. This 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.

# /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 in particular, 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 in particular for applications with relatively small primary particle energy and large number of events.

# UI commands for cout/cerr

- /control/cout/useBuffer <*flag*>
  - Store G4cout and/or G4cerr stream to a buffer so that output of each thread is grouped.
  - The buffered text will be printed out on a screen for each thread at a time at the end of the job or at the time the user changes the destination to a file.
- /control/cout/ignoreThreadsExcept <*threadID*>
  - Omit output from threads except the one from the specified thread.
  - If *threadID* is greater than the actual number of threads, no output is shown from worker threads.
  - To reset, use -1 as *threadID*.
- /control/cout/prefixString <*prefix*>
  - In case G4cout and/or G4cerr are not buffered, output of all threads are displayed on the screen simultaneously.
  - With this command, the user may specify a prefix for each output line which is supplemented by the thread ID. By default it is “G4MT”
- /control/cout/setCoutFile <*fileName*> <*ifAppend*>
  /control/cout/setCerrFile <*fileName*> <*ifAppend*>
  - Send *G4cout/G4cerr* stream to a file dedicated to each thread. The file name has "G4W_n_" prefix where *n* represents the thread ID.
  - File name may be changes for each run. If *ifAppend* parameter is false, the file is overwritten when exactly the same file has already existed.
  - To change the *G4cout/G4cerr* destination back to the screen, specify the special keyword "**Screen**" as the file name.

# LATEST DEVELOPMENTS

# Tasking and developments in releases 10.7 and 11.0

- Tasking was introduced in Geant4 10.7 to adapt to frameworks of LHC experiments which are task oriented. It comes in two variants:
  - a native C++ implementation of the 'task model', and
  - (Intel) Thread Building Block based 'TBB task mode'.
- The tasking system, based on PTL (Parallel Tasking Library) v2.0.0, is the default parallelism scheme for multi-threading starting in Geant4 11.0. To obtained either
  - use the dedicated run manager (G4TaskRunManager), or
  - Get it via the factory G4RunManagerFactory

  Both enabled use of tasks for the event loop.
- The default behaviour for tasking is to submit the tasks to an internal thread-pool and task-queue.

- Note: The tasking system with Intel TBB can be selected by specifying the option GEANT4_USE_TBB=ON is specified when configuring CMake.

# Moving from threads to tasks – since release 10.7

- There are now several types of RunManagers provided in the Geant4 release:
  - Sequential ( G4RunManager )
  - 'Old-style' Multi-threading ( G4MTRunManager )
  - G4TaskRunManager in 'native' mode
  - G4TaskRunManager in TBB mode
- A new class **G4RunManagerFactory** can be used to create any of these:

```
#include "G4RunManagerFactory.hh"
// [Option #1]
// Select from enum class G4RunManagerType: Default, Serial, MT, Tasking, TBB
auto* runMgr = G4RunManagerFactory::CreateRunManager(G4RunManagerType::Default);
// [Option #2] choose by string:  "default", "serial", "mt", "task", "tbb"
auto* runMgr = G4RunManagerFactory::CreateRunManager( "default");
```

- This was provided for the first time in release 10.7 (December 2020).