

## Multithreading in Geant4 version 10 and its integration to experiments' simulation frameworks

Makoto Asai (SLAC)  
On behalf of the Geant4 Collaboration  
LPCC detector simulation workshop  
March 19, 2014







# Contents

- General introduction and highlights of multithreading functionalities in Geant4 version 10.0
- Status of LHC experiments on shifting to multithreading
- Prospective














Note: For all other new or improved features in Geant4 version 10 except multithreading, please refer to the presentations at the previous [Geant4 Technical Forum](#).

*Geant4*

Multithreading in Geant4 v simulation

## Geant4 Software

### Introduction

Geant4 is being used in many different fields where simulation of radiation passing through and interacting with matter is critical. User domains include: high energy and nuclear physics, medical physics and space engineering, shielding protection and more. Its abstract layers based on robust OO design enables flexibility and extensibility of the code, and its open-source code and open collaboration have allowed substantial extensions of the code. New features are constantly added to the code, while increasing attention is paid to improving software performance and robustness by employing cutting-edge software engineering technologies.

### New era - Geant4 version 10 series

The next release of Geant4 – Version 10.0 (December 2013) will include event-level parallelism via multi-threading. To efficiently use new computing architectures the workload of a single job will be sub-divided to many worker threads each responsible for the simulation of one or more events. Current beta release has already shown good scalability on a number of different architectures: Intel Xeon servers, Intel Xeon Phi co-processors and low-power ARM processors

- Proof of principle
- Identify objects to be shared
- First testing
- Further refinements
- API re-design
- Example migration
- Further testing
- First optimizations
- Production ready
- Public release

Geant4  
prototype 9.4  
(2012)

Geant4  
prototype 9.5  
(2012)

Geant4 10.0  
(current)

Geant4 10.0  
(Dec. 2013)

Geant4 10 series  
(2014+)

### New physics

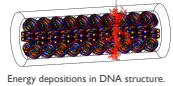
The flexibility and extensibility of Geant4 design allows it to be applied to new physics domains. These include the physics of condensed matter (phonon transportation in crystals, drift of electrons and holes in semiconductors) and processes for bio-chemical substances and DNA.

SuperCDMS Cryogenic Dark Matter Search seeks to directly detect dark matter. Geant4 models the caustic pattern in a Ge crystal (left) by tracking individual phonons (right)

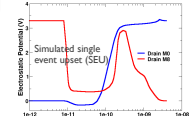
Geant4 performs mission critical studies of radiation and charging effects on spacecraft electronics. Impact of Neutron ion on MOS FET

Reaction	Reaction rate (10 <sup>17</sup> M <sup>-1</sup> s <sup>-1</sup> )
IP + e <sup>-</sup> → IP <sup>+</sup> + O <sup>-</sup> + H <sub>2</sub>	3.64
IP + OH <sup>-</sup> → IP <sup>+</sup> + H <sub>2</sub> O	1.44
IP + H <sub>2</sub> → IP <sup>+</sup> + H <sub>2</sub>	1.20
H <sub>2</sub> + O <sup>-</sup> → OH <sup>-</sup> + H <sub>2</sub>	4.17E10 <sup>1</sup>
H <sub>2</sub> O + e <sup>-</sup> → OH <sup>-</sup> + H <sub>2</sub>	1.61
H <sub>2</sub> O <sup>+</sup> + e <sup>-</sup> → OH <sup>-</sup> + H <sub>2</sub>	2.11
H <sub>2</sub> O <sup>+</sup> + OH <sup>-</sup> → H <sub>2</sub> O	14.3
OH <sup>-</sup> + H <sub>2</sub> → H <sub>2</sub> O	2.93
OH <sup>-</sup> + OH <sup>-</sup> → H <sub>2</sub> O	0.44
*e <sup>-</sup> + *e <sup>-</sup> → 2 O <sup>-</sup> + H <sub>2</sub>	0.60

Reactors of radicals available in Geant4.



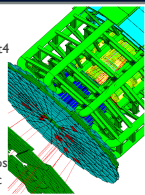
Energy depositions in DNA structure.



Simulated single event upset (SEU)


### Geometry

The flexibility and extensibility of Geant4 design also enables handling rich collection of shapes including CSG (Constructed Solid Geometry), BREP (Boundary REPresented), Boolean operation, Tessellated solid, etc. and the user can easily add new shapes. Geant4 geometry navigation can deal with setups up to billions of volumes with automatic optimization. In addition, geometry models can be 'dynamic', i.e. changing the setup at run-time, e.g. "moving objects".



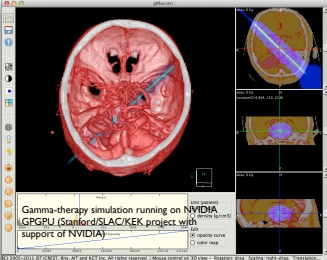
### Software quality assurance

Geant4 uses modern tools to manage the code and improve code quality: from handling issues with JIRA to continuous testing integration with CTest/CDash, profiler based optimizations, Quality Assurance (Coverity, Valgrind, etc.), and IDE integration (Xcode, Eclipse, VisualStudio).









### Investments for the future

Geant4 collaboration members are participating in various explorations of emerging technologies. These technologies include GPU/CUDA, OpenCL, OpenACC, vectorization, DSL, etc.



Gamma-therapy simulation running on NVIDIA GPGPU (Stanford/SLAC/KEK project with support of NVIDIA)

# Geant 4

Version 10.0-p01

General introduction  
and highlights of multithreading  
in Geant4 version 10

# Geant4 version 10 series

- The release in 2013 was a major release.
  - Geant4 version 10.0 – release date : Dec. 6, 2013
- The highlight is its **multi-threading capability**.
  - A few interfaces need to be changed due to multi-threading
- It offers two build options.
  - Multi-threaded mode (including single thread)
  - Sequential mode
    - In case a user depends on thread-unsafe external libraries, (s)he may install Geant4 in sequential mode.



- Proof of principle
- Identify objects to be shared
- First testing

- MT code integrated into G4

- **API re-design**
- Example migration
- Further testing
- First optimizations

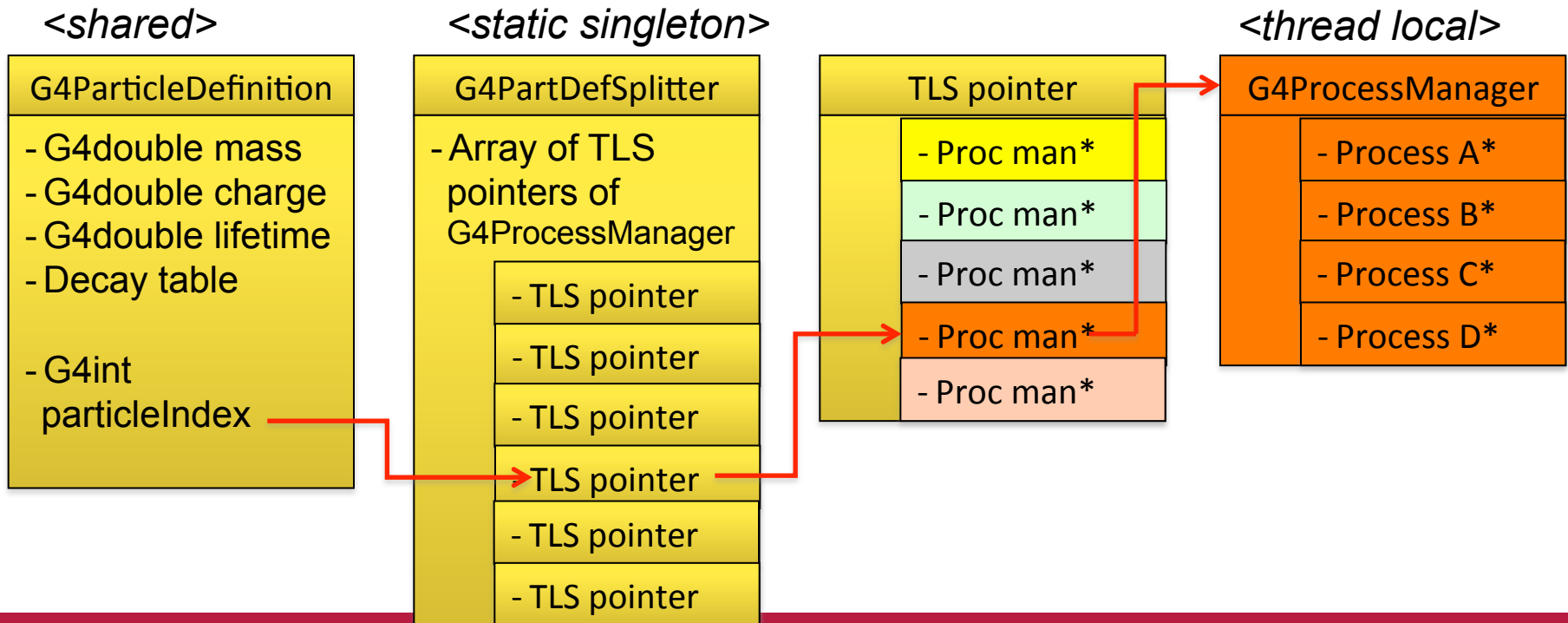
- **API refinements**
- Production ready
- Public release

- Further refinements and optimizations

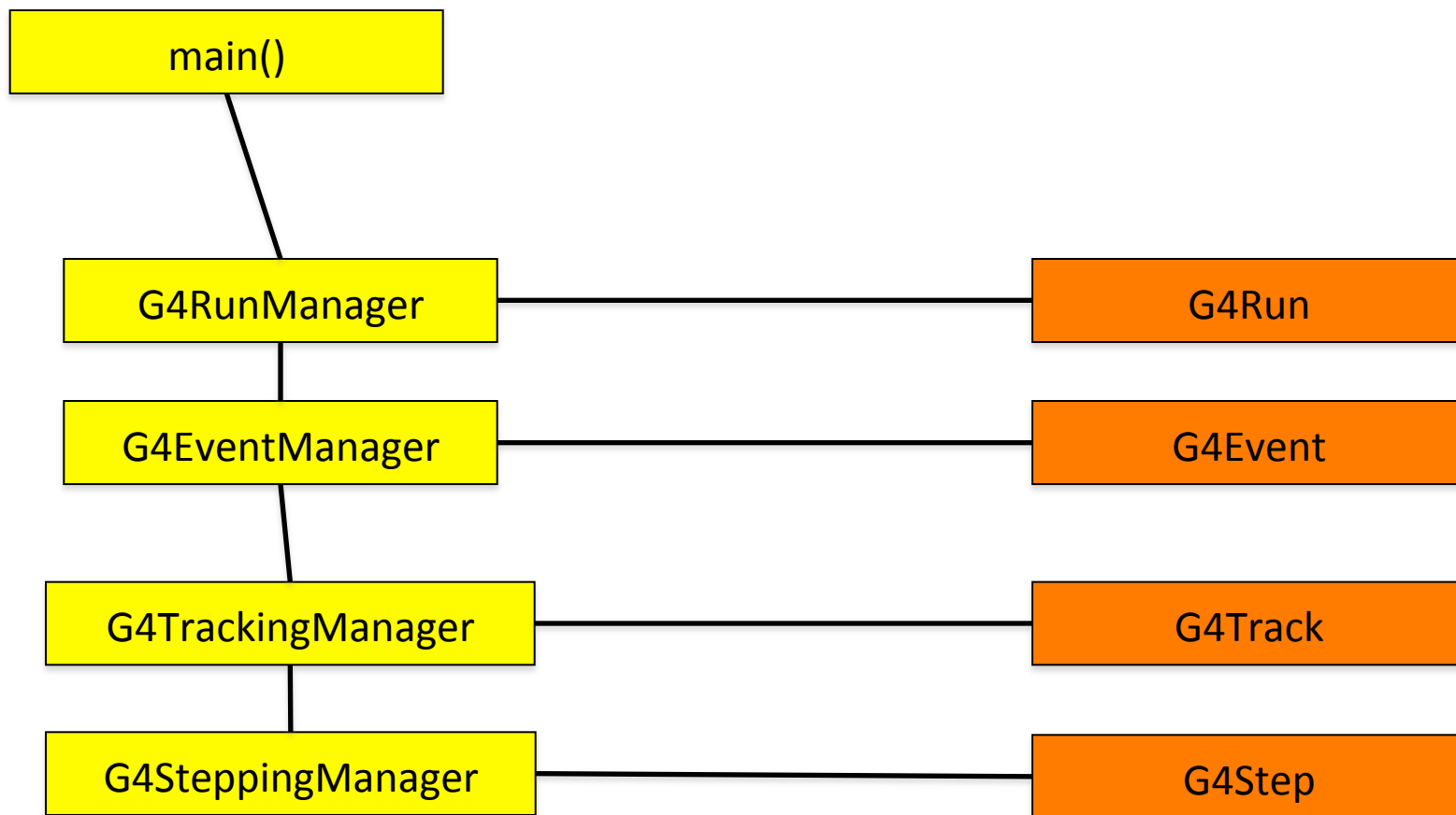
- This choice minimizes the changes in user-code
  - Maintain API changes at minimum
- All Geant4 code has been made thread-safe.
  - Thread-safety implemented via Thread Local Storage
- Most memory-consuming parts of the code (geometry, physics tables) are shared over threads.
  - “Split-class” mechanism: reduce memory consumption
    - Read-only part of most memory consuming classes are shared
    - Enabling threads to write to thread-local part
- Particular attention to create “lock-free” code: linearity (w.r.t. #threads) is the metrics we concentrated on for the v10.0 release.

# Split class – case of particle definition

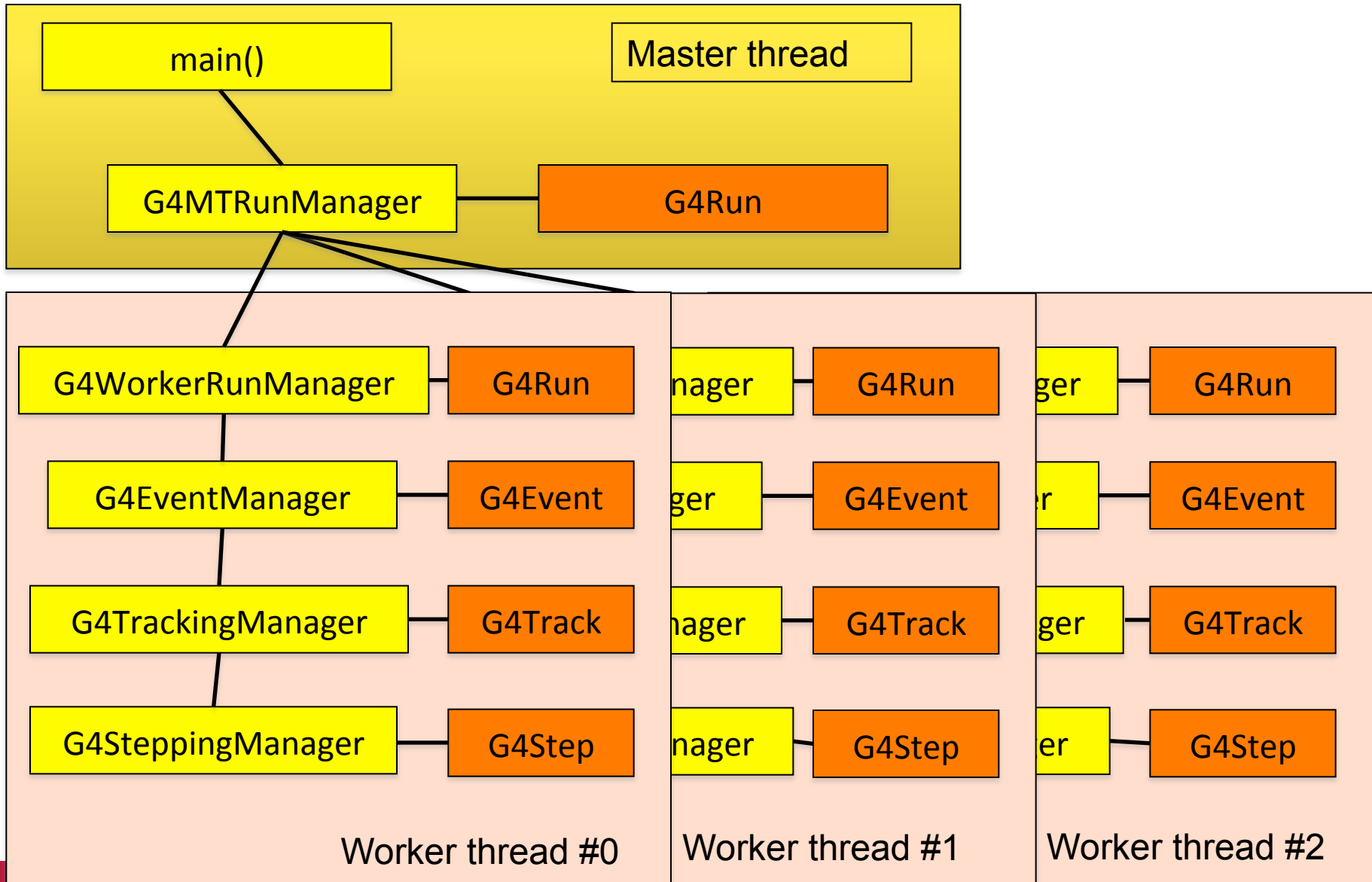
- In Geant4, each particle type has its own dedicated object of G4ParticleDefinition class.
  - Static quantities : mass, charge, life time, decay channels, etc.,
    - To be shared by all threads.
  - Dedicated object of G4ProcessManager : list of physics processes this particular kind of particle undertakes.
    - Physics process object must be thread-local.



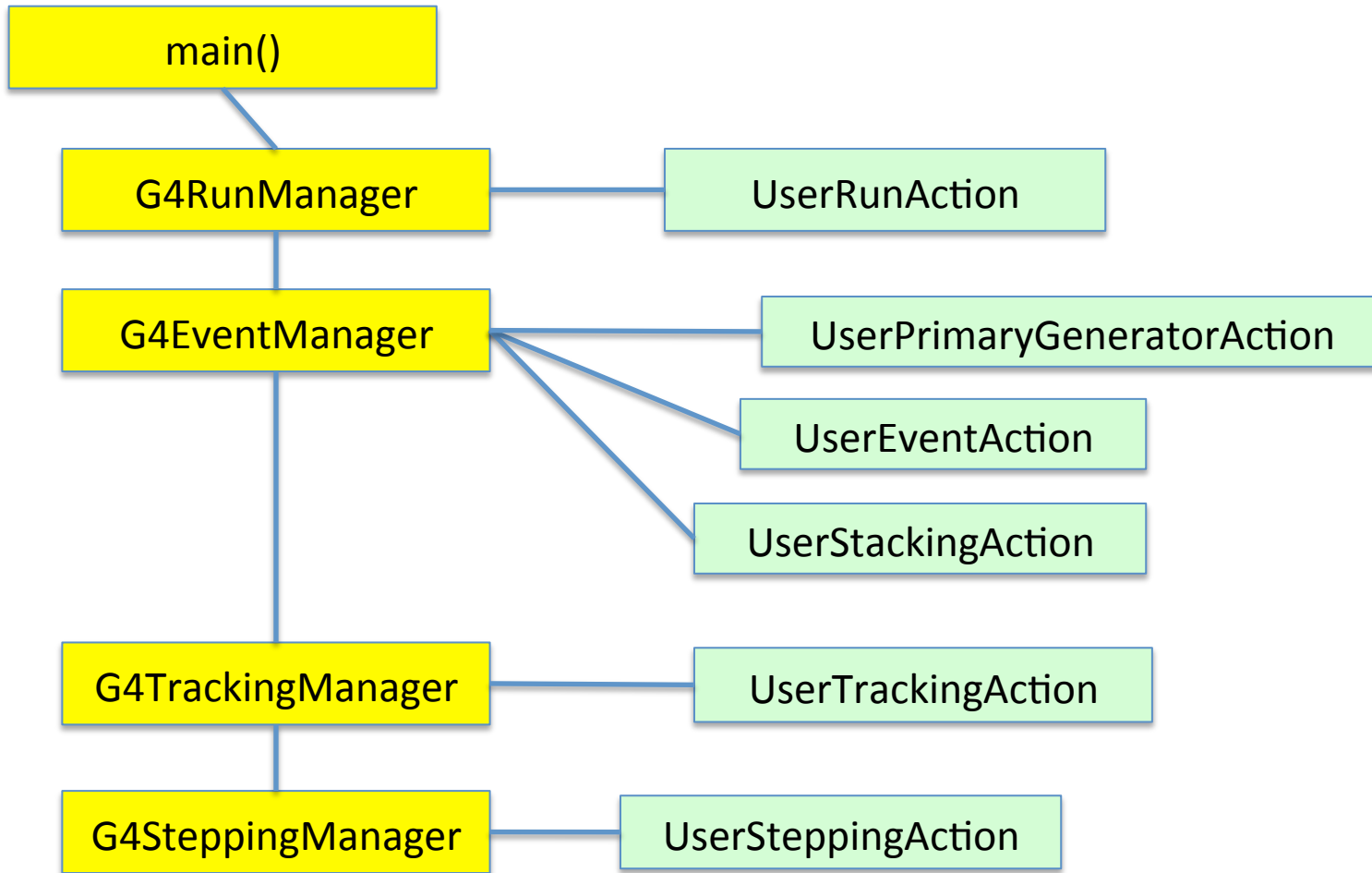
- This choice minimizes the changes in user-code
  - Maintain API changes at minimum
- All Geant4 code has been made thread-safe.
  - Thread-safety implemented via Thread Local Storage
- Most memory-consuming parts of the code (geometry, physics tables) are shared over threads.
  - “Split-class” mechanism: reduce memory consumption
    - Read-only part of most memory consuming classes are shared
    - Enabling threads to write to thread-local part
- Particular attention to create “lock-free” code: linearity (w.r.t. #threads) is the metrics we concentrated on for the v10.0 release.
- Initial performance penalties observed in early prototypes have already been addressed.
- Testing on both x86\_64 and MIC architectures
- Use of POSIX standards
  - Allowing for integration with user-preferred parallelization frameworks (e.g. MPI, TBB, etc.)



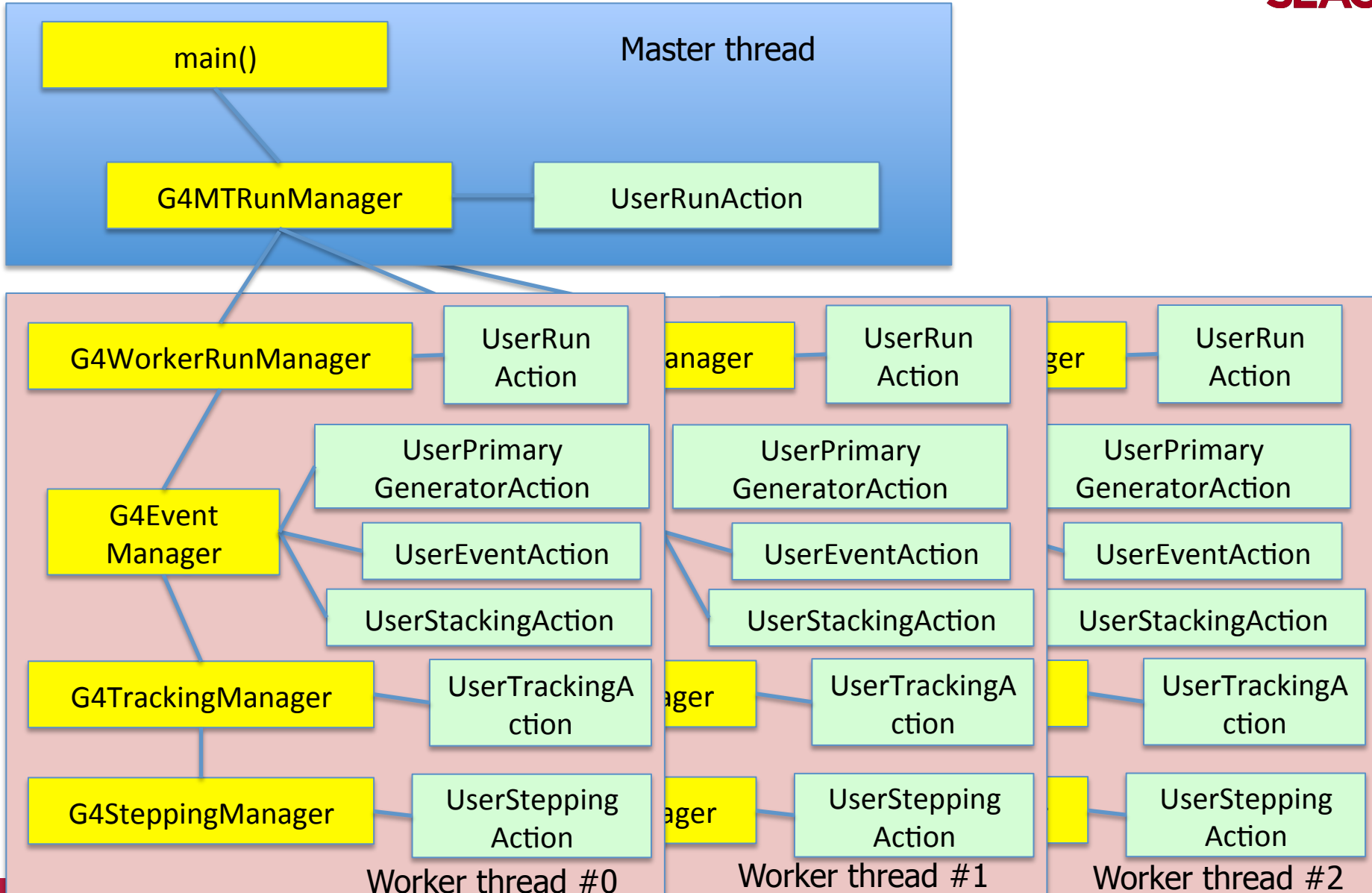




- In the multi-threaded mode, generally saying, data that are stable during the event loop are shared among threads while data that are transient during the event loop are thread-local.
- In general, geometry and physics tables are shared, while event, track, step, trajectory, hits, etc., as well as 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*.




# Multi-threaded mode



Multithreading in Geant4 version 10 and its integration to experiments'

# User classes

- Initialization classes
  - Use `G4RunManager::SetUserInitialization()` to define.
  - Invoked at the initialization
    - `G4VUserDetectorConstruction`
    - `G4VUserPhysicsList`
    - `G4VUserActionInitialization` 
- Action classes
  - Instantiate in `G4VUserActionInitialization`.
  - Invoked during an event loop
    - `G4VUserPrimaryGeneratorAction`
    - `G4UserRunAction`
    - `G4UserEventAction`
    - `G4UserStackingAction`
    - `G4UserTrackingAction`
    - `G4UserSteppingAction`

Note : classes written in **red** are mandatory.

- **G4UserActionInitialization** 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.

- G4Allocator objects are now thread-local.
- If the user uses G4Allocator for his/her own class, e.g. hit, trajectory or trajectory point, G4Allocator object must be thread local and thus must be instantiated within the thread. The object new-ed and allocated by the thread-local G4Allocator must be deleted within the same thread.
- Thread-local G4Allocator objects are automatically deleted when the worker thread terminates.

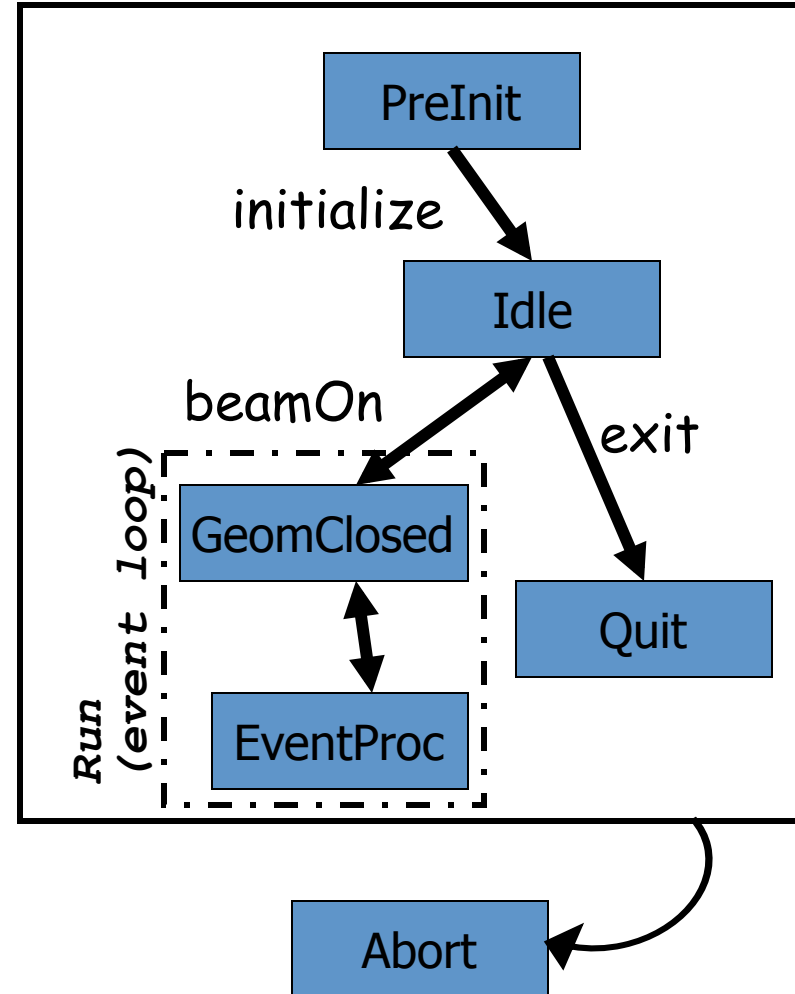
- In MyHit.hh

```
typedef G4THitsCollection<MyHit> MyHitsCollection;  
extern G4ThreadLocal G4Allocator<MyHit>* MyHitAllocator;  
inline void* MyHit::operator new(size_t)  
{ if(!MyHitAllocator) MyHitAllocator = new G4Allocator<MyHit>;  
  return (void *) MyHitAllocator->MallocSingle(); }  
inline void B2TrackerHit::operator delete(void *hit)  
{ MyHitAllocator->FreeSingle((MyHit*) hit); }
```

- In MyHit.cc

```
G4ThreadLocal G4Allocator<MyHit>* MyHitAllocator=0;
```

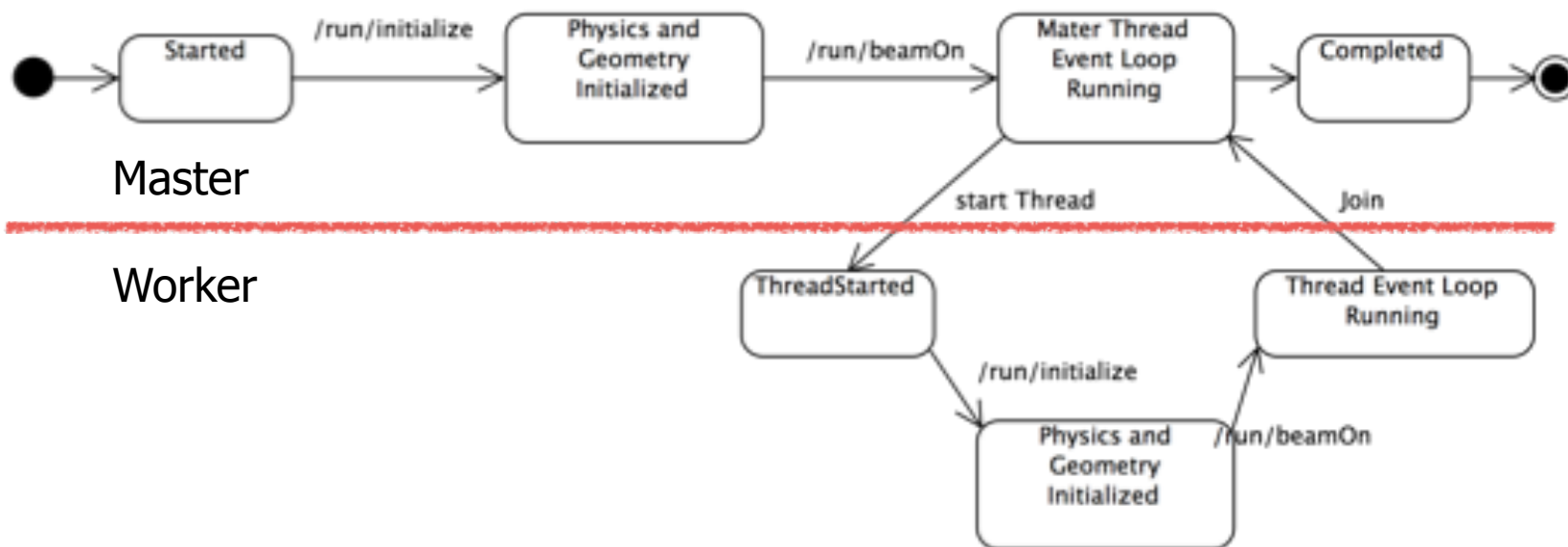
- Geant4 has six application states.
  - G4State\_PreInit
    - Material, Geometry, Particle and/or Physics Process need to be initialized/defined
  - G4State\_Idle
    - Ready to start a run
  - G4State\_GeomClosed
    - Geometry is optimized and ready to process an event
  - G4State\_EventProc
    - 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.

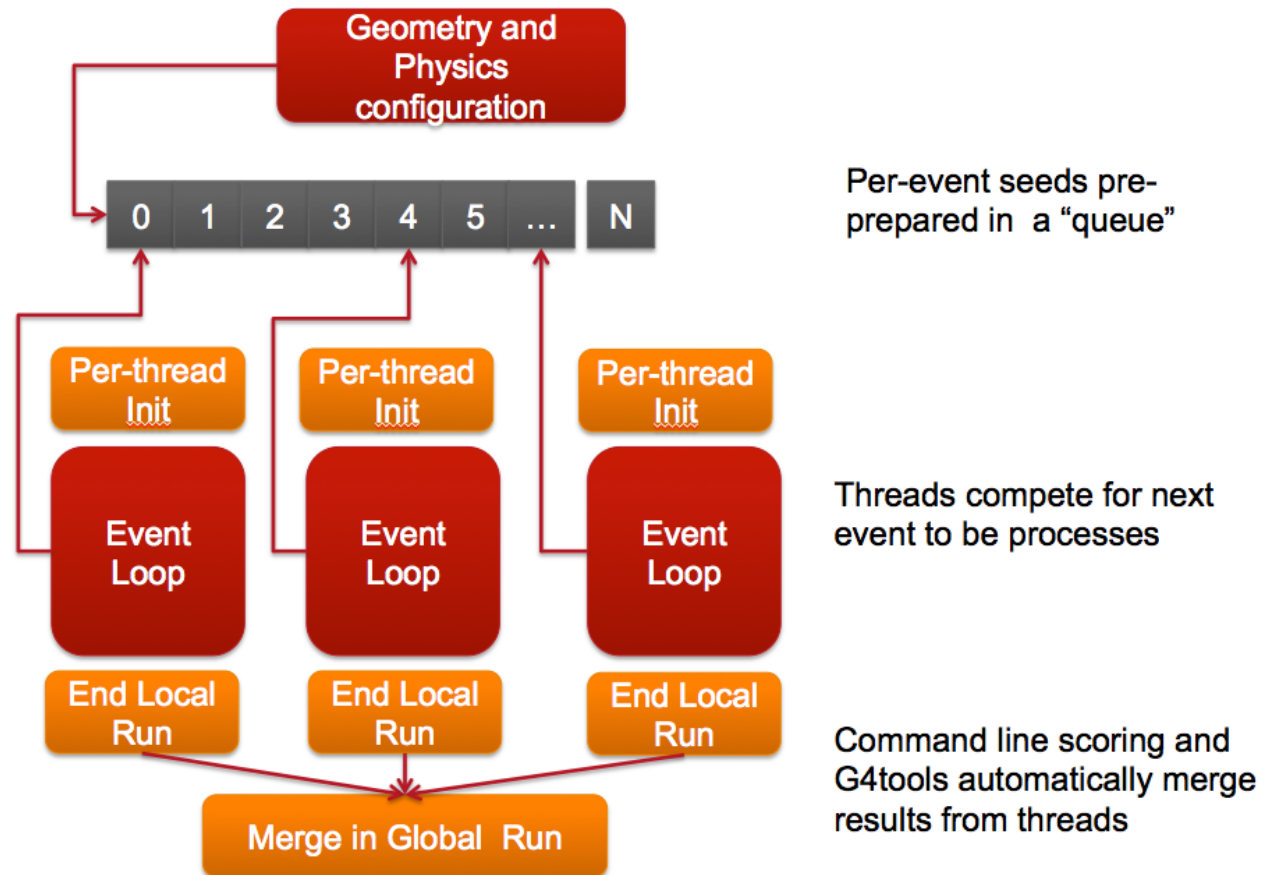


- A G4 (with MT) application can be seen as simple finite state machine
- Threads do not exist before first /run/beamOn
- When master starts the first run spawns threads and distribute work



# Event tasking is not round robin

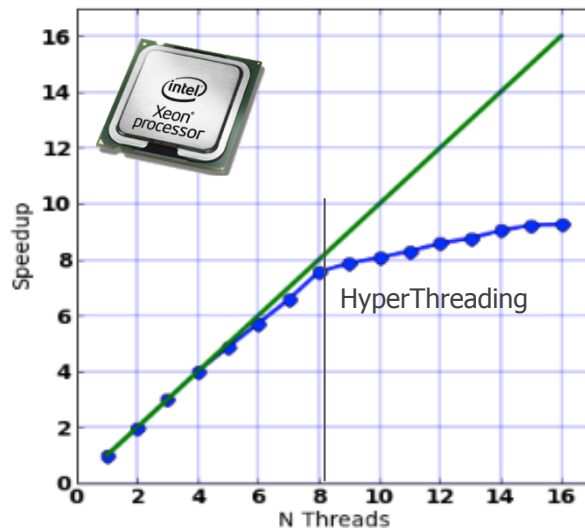
- During the master event loop, an event (or a bunch of events) is tasked to a worker thread in first-come-first-served basis.
  - To minimize the latency at the end of master event loop
  - Required toward our next goal of complete decoupling between the master event loop and worker thread initialization/termination
    - Desirable for TBB-based simulation (see later slides)
- Master thread generates all the necessary initial seeds for all events and dispatch.
  - For the sake of full reproducibility regardless of number of threads.



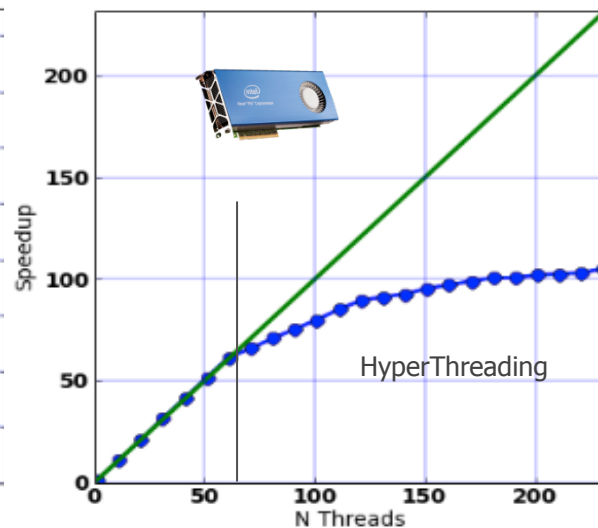
# Performance on different architectures

- Current release has already shown good scalability on a number of different architectures: Intel Xeon servers, Intel Xeon Phi co-processors and low-power ARM processors.
  - On Intel architectures, it has shown performance improvements not only up to the number of physical cores but in hyper-thread mode as well.

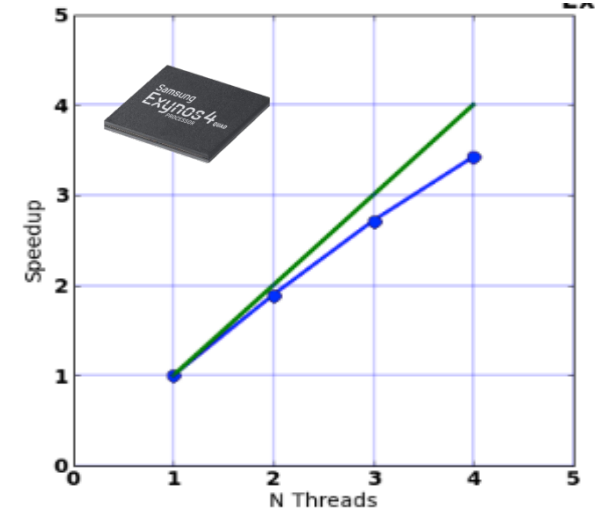
Intel Xeon L5520 @ 2.27GHz

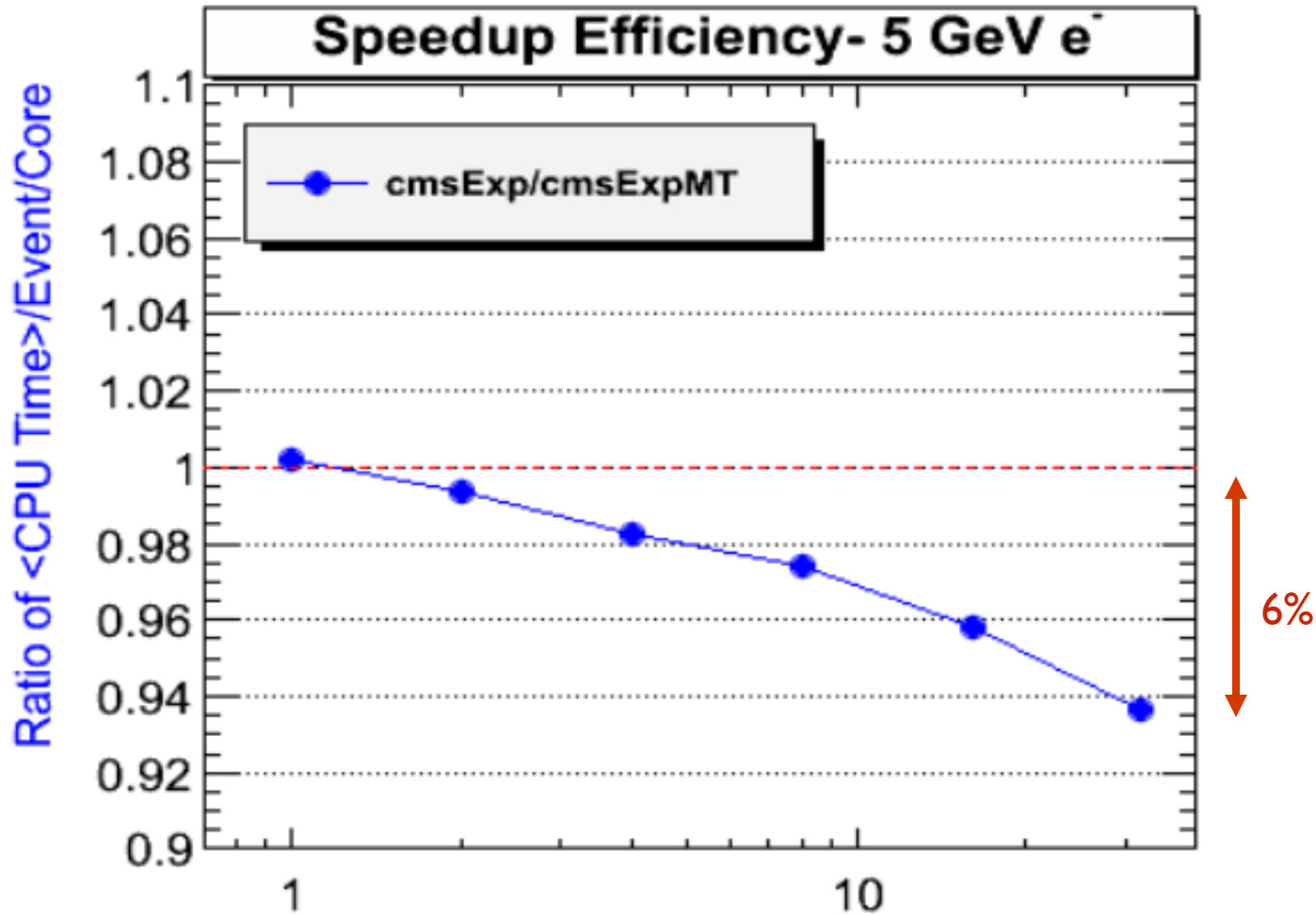


Intel Xeon Phi 7120P @ 1.238GHz



Exynos 4412 Quad-Core @ 1.7 GHz



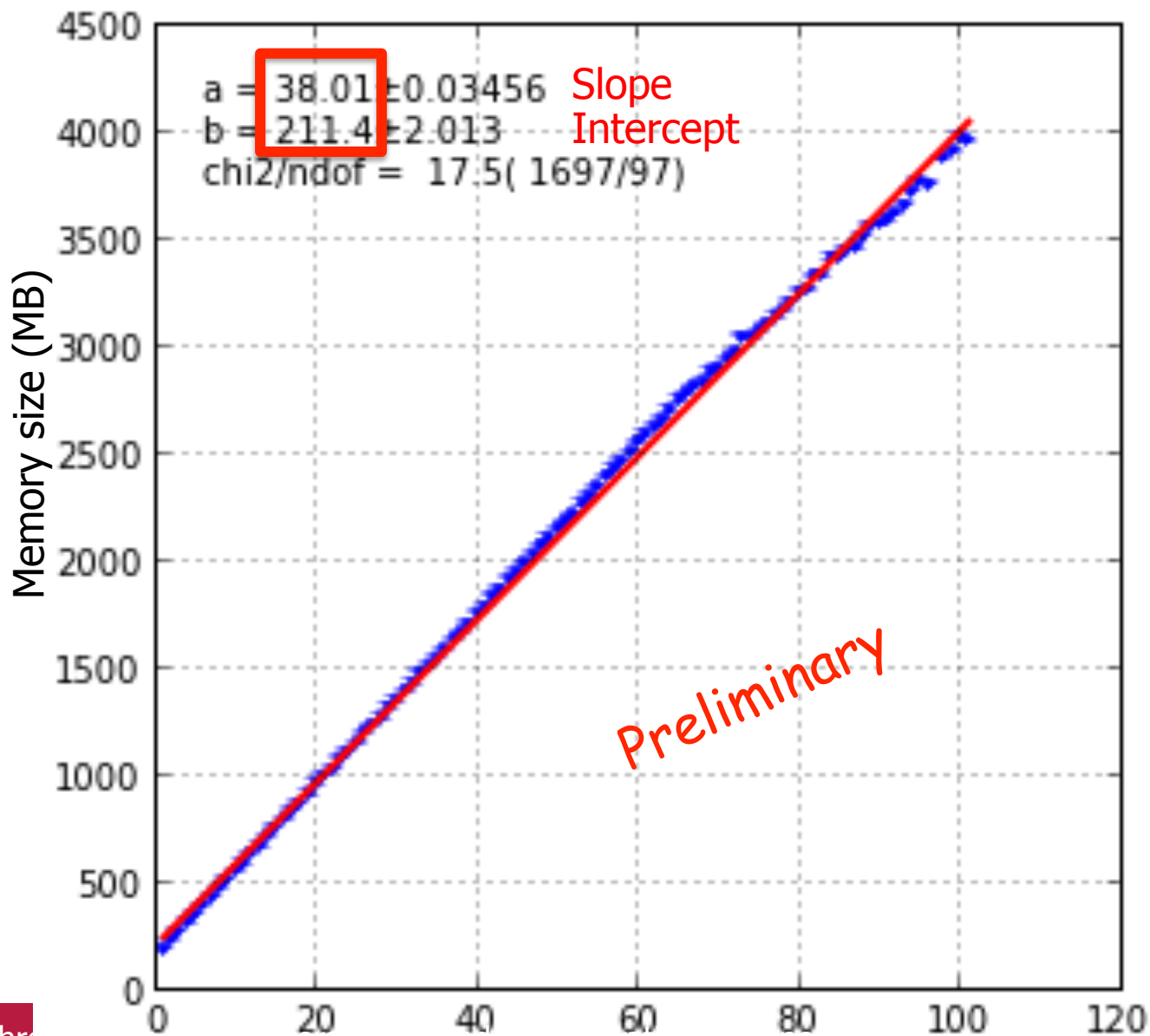


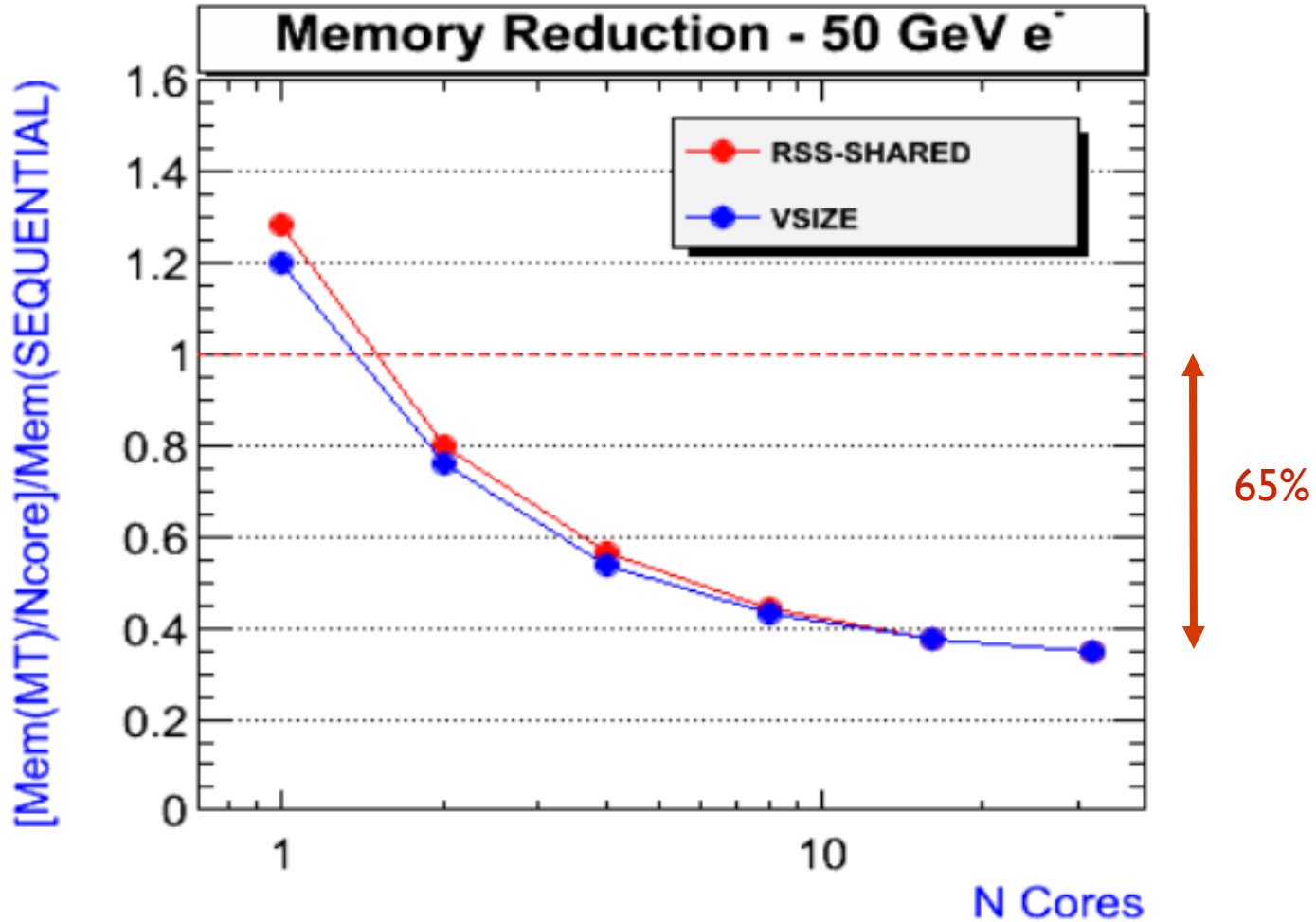
"CMS-ish" geometry

Processor: AMD Opteron(tm) Processor 6128 : 32 cores (4 CPU sockets x 8 cores)  
CPU: 2000 MHz, Cache: 512 KB, Total Memory: 66007532 kB  
OS: Linux kernel 2.6.32-358.11.1.el6.x86\_64 GCC 4.4.6 20120305 (Red Hat 4.4.6-4)

# Memory consumption

- Geant4 compiled for MIC architecture
- Full CMS detector without sensitive detectors, hits or trajectories
- No optimization yet
- ~40MB /thread
- Works in progress to reduce the memory consumption per thread.
  - For example eliminating big thread-local arrays in physics processes

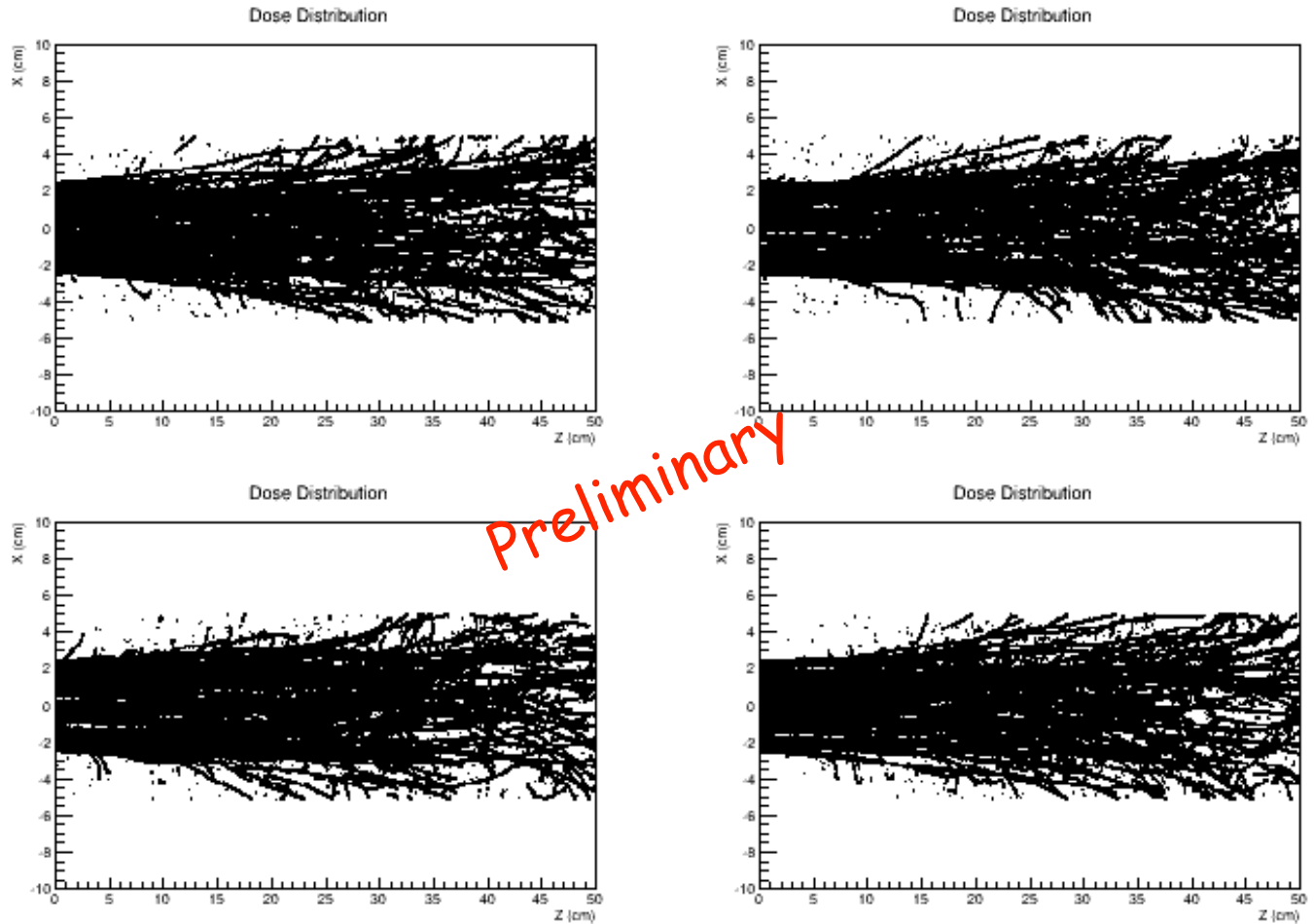




"CMS-ish" geometry

Processor: AMD Opteron(tm) Processor 6128 : 32 cores (4 CPU sockets x 8 cores)  
CPU: 2000 MHz, Cache: 512 KB, Total Memory: 66007532 kB  
OS: Linux kernel 2.6.32-358.11.1.el6.x86\_64 GCC 4.4.6 20120305 (Red Hat 4.4.6-4)

- Geant4 version 10 works with MPI.
  - Many nodes of many cores



- 4 MPI processes with 2 cores each
- Each MPI process owns histogram
- Threads merge dose calculation in shared histogram

# Preliminary studies on TBB



- Intel Threading Building Block is a library for task-based multi-threading code. Some LHC experiments show their interest in the use of TBB in their frameworks.
- We have verified that the G4 v10 can be used in a TBB-based application where TBB-tasks are responsible for simulating events.
  - We didn't need to modify any concrete G4 class/method to adapt to TBB.
- We provide an example in version 10.0 release to demonstrate the way of integrating Geant4 with TBB.
- We keep investigating where/how to reduce memory use.
- We will keep communicating with our users to polish our top-level interfaces.
  - Next step includes decoupling of master event loop and worker thread initialization/termination.



# Geant 4

Version 10.0-p01

Status of LHC experiments  
on shifting to multithreading

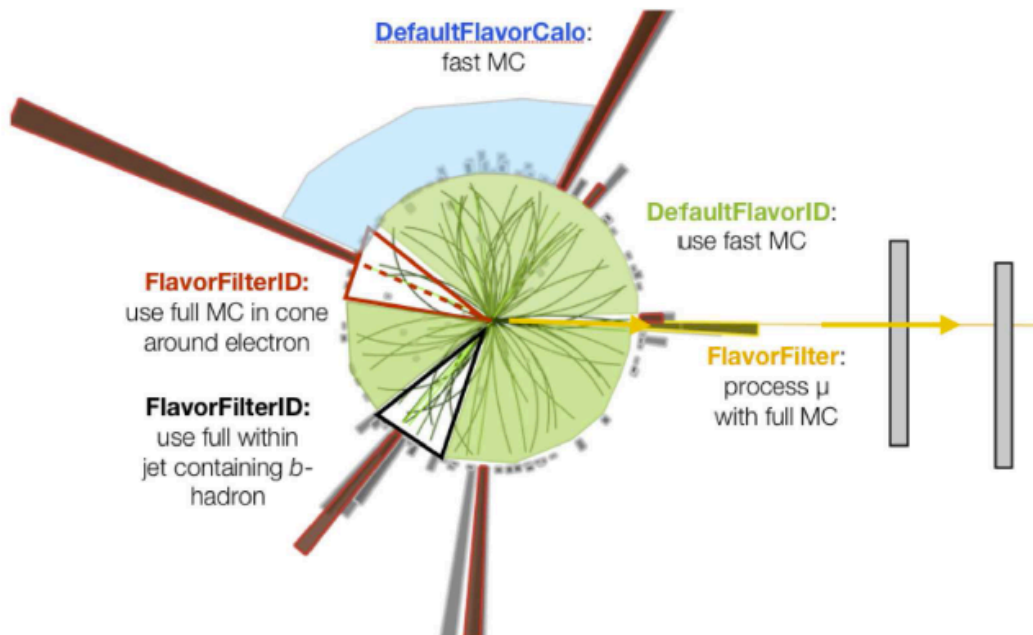
- We asked LHC experiments about their status and plan of shifting their simulation code/framework to multithreading. Inquiries are:
  - what have been done up to now,
  - problems/issues/difficulties identified so far,
  - possible solutions to these problems/issues/difficulties,
  - plans, including a time-table, toward the production, and
  - requests to the Geant4 Collaboration.
- Following slides are their feedbacks.

- what have been done up to now,
  - The migration of Geant4 VMC which is the framework used in ALICE simulation is close to be finished. The beta release is planned for this week. This will allow to start tests with ALICE geometry, start of migration of the ALICE VMC application and tests on limited detector setup. We will present more details at the ALICE talk at the workshop.
- problems/issues/difficulties identified so far,
  - Most of problems encountered during the work on Geant4 VMC were already reported and fixed; the remaining one with optical processes is now added in bugzilla as #1590. There are a few more minor issues which will be reported in our talk.
- plans, including a time-table, toward the production, and
  - Tests with Geant4 VMC examples run with ALICE geometry (in 1-2 months)
  - AliRoot (with a limited detector setup first) migration to MT: this means to adapt VMC application class for MT and fix thread safety problems in the code which is used in simulation: AliRoot core classes + detector "steering" classes. (in 1-2 years)
- requests to the Geant4 Collaboration.
  - Have a possibility to reduce the output from initialization of physics processes as this makes the output, especially when run in MT mode, very long.

# G4-MT for ATLAS

- Initial implementation of Geant4-MT for ATLAS done using G4-MT 9.4.p01 with some fixes borrowed from 9.5.p01.
- Used *semaphore* class to keep track of threads and mutexes. (A little awkward, but ok for initial tests.)

## The nutshell ISF vision



- Geant4 is implemented inside the Integrated Simulation Framework (ISF)
- ISF allows Geant4 to be used simultaneously with various types of fast simulation within individual events.

## Lessons learned with G4-MT for ATLAS

- ATLAS simulation is implemented inside Athena framework, and has a small dependence on some of the framework services (such as messaging) which are not thread-safe.
  - We need to limit the use of these services, and protect where necessary (if possible)
  - This highlights the need to move toward a thread-safe version of Athena (with simulation as an ideal test candidate)
- To use Geant4 simultaneously with fast simulation, control of final hit collections has to be maintained outside of Geant4.
  - Changes in release 10 of Geant4 make this easier → not expected to be a problem now.
- The ATLAS simulation framework creates all stepping and other types of actions as singletons, and modifying the framework to allow multiple instances of these actions would have required an extensive redevelopment of the framework
  - Initial studies will be done with a reduced set of stepping actions (i.e., no truth storing) to simplify testing (expected to be easier in rel. 10)

## Parallelisation studies

- Parallelisation of tracks:
  - Perhaps the easiest to implement (at least in standard G4-MT).
  - Not clear whether this gives any improvement over event-level parallelisation.
- Sub-detectors:
  - Limits number of threads/processes writing to the same truth/hit collections.
  - Would allow separate lookup tables for B-field and less memory per thread for geometries.
  - Cache optimisation
- Particle type: Cache optimisation has been studied in the past, will perhaps look again with Geant4 release 10.
- Separate threads for various simulation flavours (full and fast) in ISF.
- Best solution: probably a combination of all of the above. Studies needed to determine which to choose.



# Geant4 10.0 in CMSSW (1/2)

- **CMS started to test Geant4 10.0 from the beginning of November**
  - All compilation problems were resolved
  - Development build is multi-threaded used in sequential mode
- **Multithread Geant4 build problems for CMSSW were resolved**
  - It was realized that because of CMSSW plugin system the compilation option '-ftls-model=global-dynamic' is mandatory
    - Geant4 is using '-ftls-model=initial-exec'
  - The problem was in large TLS memory allocated for Geant4 objects
- **First estimate of CPU performance – similar to Geant4 9.6p02**
  - CPU profiling shows:
    - Mathematical functions, EM and hadronic physics take less time than that of 9.6p02
    - Leading CPU usage (~30%) for transportation in CMS geometry and magnetic field
- **Currently Geant4 10.0p01 has been integrated into CMSSW**
  - Sequential build is planned to be used for start of production for 2015 run
  - Physics checks have been already started

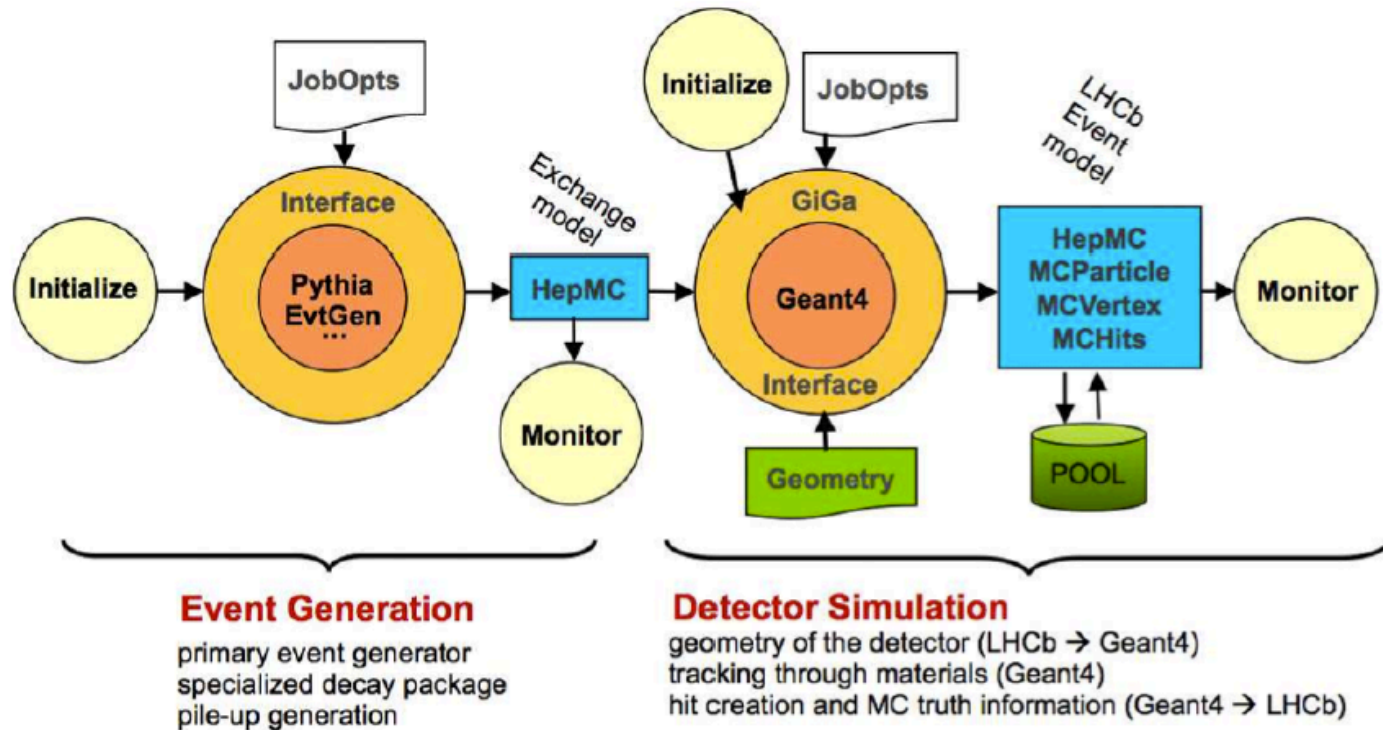


# Geant4 10.0 in CMSSW (2/2)

- **Multi-threaded build of Geant4 is available in the dedicated branch where CMS MT simulation is under development**
  - It is planned to have a production-ready version for Q3 of 2014
  - The goal is to have both sequential and MT working within the same production version of CMSSW
- **CMSSW is using CLHEP which should be thread safe**
  - CLHEP is used not only by Geant4 but also by other CMSSW packages
- **Main request to Geant4 team: develop extended TBB example within Geant4**
  - Demonstrate how Geant4 MT may be used efficiently within TBB
  - How manage memory allocated for Geant4 task if number of streams for SIM is dynamic?
    - A stream may perform different tasks
    - How to avoid re-initialisation of Geant4 when a stream switched from other task to Geant4 simulation?

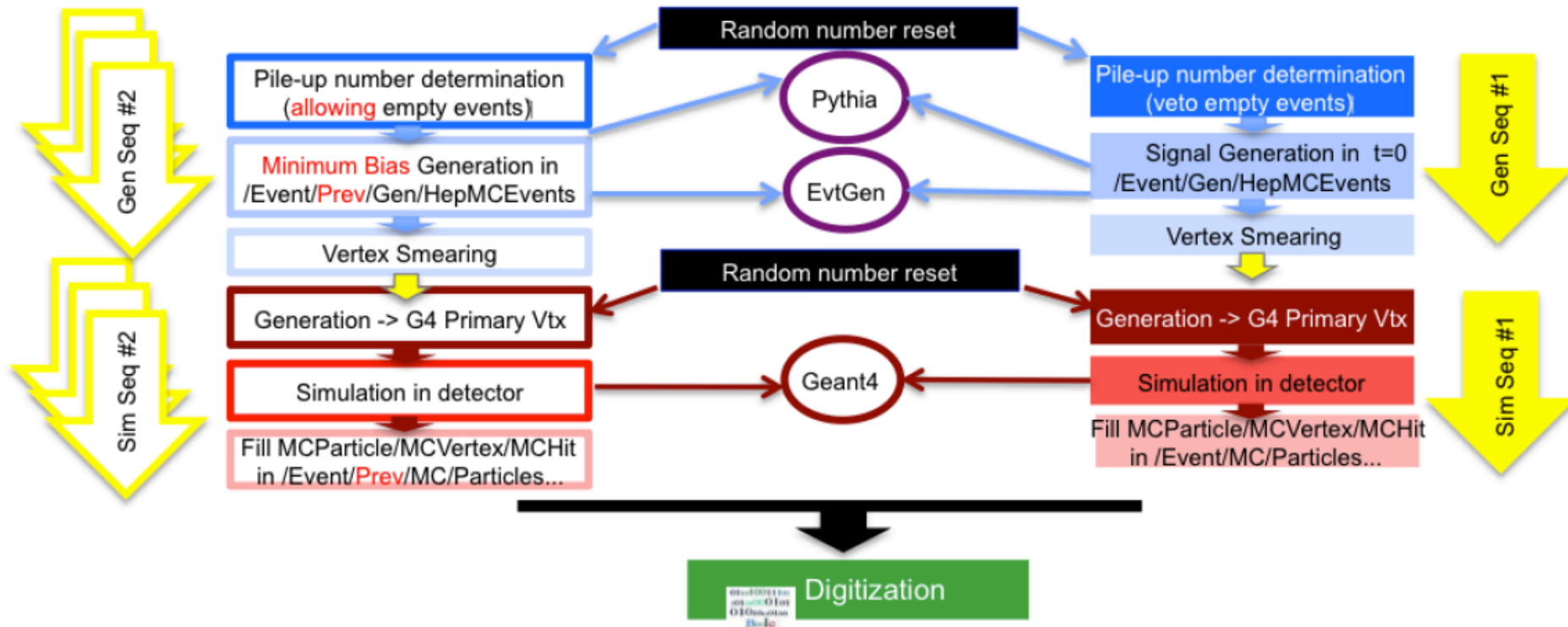


Gauss operates in two phases - generation and simulation.



- GEANT4 “events” scheduled by Gaudi.
- Detector geometry converted during initial event.
- Out of time pileup handled in same Gauss job, requiring only one instantiation of Generator, EVTGEN and GEANT4.

# Out of Time Pileup in Gauss (Spillover)



- Multiple generation and simulation sequences are defined in one Gauss job
- Specific seeds assigned for each step of the sequence.
- Data written to different paths corresponding to the various sequences.

Major changes to Gauss would be required to incorporate GEANT4 MT:

- Events scheduled by Gaudi rather than GEANT4.
  - Whole event loop (seeding, execution, data writing) would need to be re-written.
  - Unclear how to process OoT pile up events and signal separately with GEANT4 MT manager.
- Geometries in Gauss are converted into GEANT4 format during initialisation of first event.
  - Requires re-ordering of GEANT4 geometry initialisation.

Current activities therefore focus on **GaudiHive** for multi-threading.

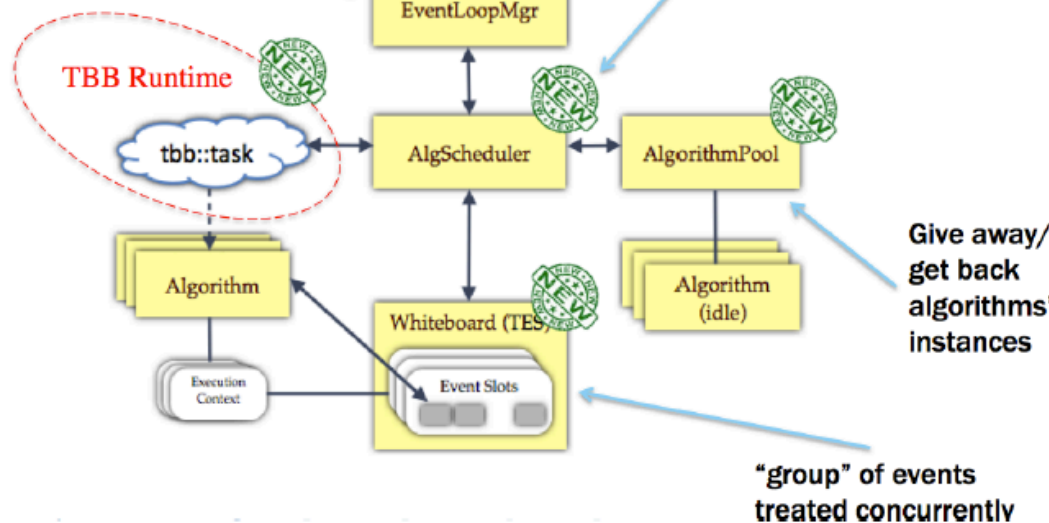
# GaudiHive

- GaudiHive is an implementation of the Gaudi framework which supports concurrency.
- Tests with “MiniBrunel” promising.

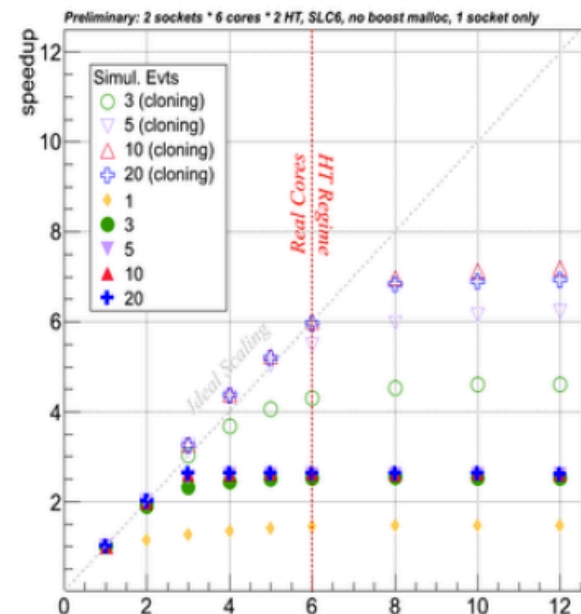
Loop on events and hand them over to scheduler

Schedules algorithms on the events

TBB Runtime



MiniBrunel 10k evts Wed Jun 5 12:46:32 2013



Using Gauss with GaudiHive is in the exploratory stages, but the ThreadSafe tools provided by GEANT4 MT are most welcome.

# Geant 4

Version 10.0-p01

Prospective



# Beyond Geant4 version 10.0

- Geant4 version 10 series will be evolving.
  - Proof of principle
  - Identify objects to be shared
  - First testing
  - API re-design
  - Example migration
  - Further testing
  - First optimizations
  - Further refinements and optimizations



- MT code integrated into G4
- API refinements
- Production ready
- Public release
- Performance improvements
  - Algorithm optimization / local vectorization
    - without losing code readability / maintainability / flexibility
  - Optimization of file access
- Memory space reduction in particular for per-thread memory
  - Sharing more physics vectors and other objects among threads
- Multithreading leftover
  - Some visualization, neutron\_hp, general particle source, Geant4e, etc.
- Completion of decoupling between master event loop and worker thread initialization / termination
  - See later slide

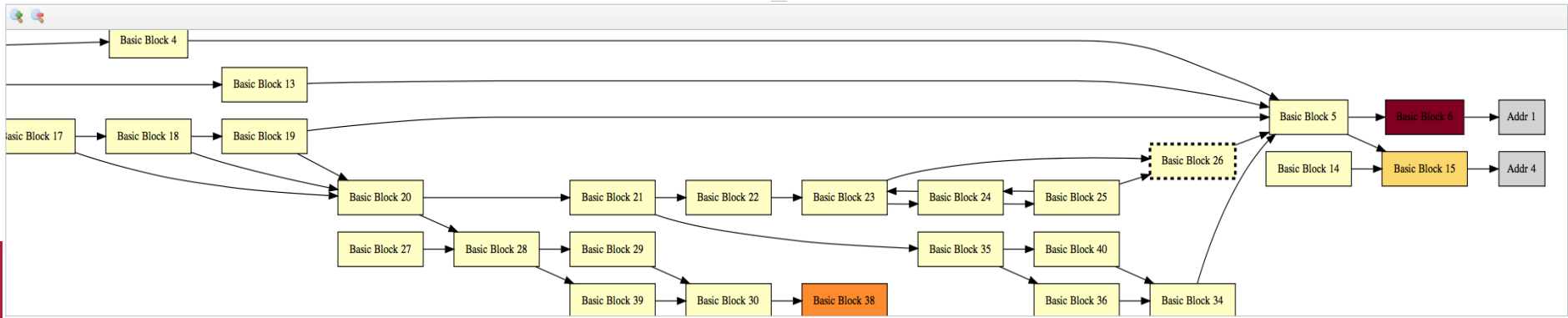
# Software quality improvements (<http://code.google.com/p/gooda/>)



- We are working with Google on performance measurements of Geant4-based application using Gooda tool, a PMU-based event data analysis package.

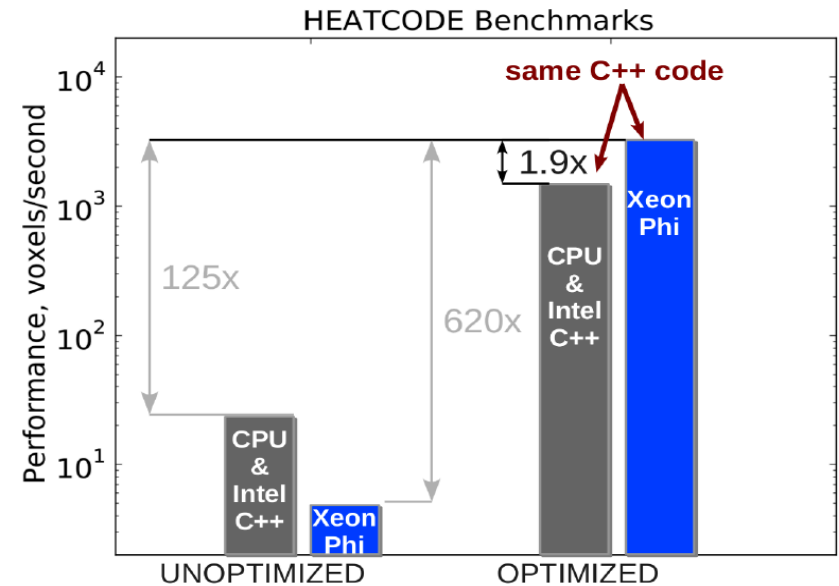
address	princ_l#	disassembly	unhalted_core_cycles	uops_retired:stall_cycles	uops_retired	instruction_retired	uops_retired:any	load_latency	instruction_starvation	bandwidth
0x30ce8	521	Basic Block 26 <0x30b2b>	249801 (100%)	183440 (73%)	81361	96333	157004 (62%)	102099 (40%)	3478 (1%)	
0x30ce8	521	lea 0x0(,%rax,8),%r8	268 (0%)	146 (54%)	95	105		20 (7%)		
0x30cef	521	null	139 (0%)	102 (73%)	32	38		20 (14%)		
0x30cf0	521	lea 0x8(,%rax,8),%r9	80 (0%)	37 (46%)	48	60				
0x30cf7	521	null								
0x30cf8	521	jmpq 30b2b	50 (0%)	7 (14%)	16	8				
0x30cfd	521	Basic Block 27 <0x30d00>								
0x30cfd	521	nopl (%rax)								
0x30d00	521	Basic Block 28 <0x30ed9>	7463 (2%)	5649 (75%)	3111	3133	3478 (46%)	4929 (66%)	129	
0x30d00	521	movq %xmm0,-0x8(%rsp)	805 (0%)	548 (68%)	317	286	99 (12%)	378 (46%)	20	
0x30d06	521	mov -0x8(%rsp),%rax	805 (0%)	716 (88%)	206	264	10 (1%)	566 (70%)	20	
0x30d0b	521	mov %rax,%rcx	566 (0%)	438 (77%)	190	173	70 (12%)	298 (52%)		
0x30d0e	521	shr \$0x34,%rcx	517 (0%)	373 (72%)	222	271	30 (5%)	149 (28%)		
0x30d12	521	sub \$0x3ff,%ecx	119 (0%)	88 (73%)	32	60				
0x30d18	521	cvtts2sd %ecx,%xmm4	199 (0%)	95 (47%)	40	60		20 (10%)		
0x30d1c	521	mov \$0x800fffffffffff...	1232 (0%)	957 (77%)	484	505	60 (4%)	765 (62%)	30	
0x30d23	521	null								
0x30d26	521	and %rcx,%rax	10 (0%)			8				
0x30d29	521	mov \$0x3fe00000000000...								
0x30d30	521	null								
0x30d33	521	or %rcx,%rax								
0x30d36	521	mov %rax,%rcx	537 (0%)	460 (85%)	222	222	40 (7%)	248 (46%)		

line number	source	unhalted_core_cycles	uops_retired:stall_cycles	uops_retired	instruction_retired	uops_retired:any	load_latency	instruct
513	G4double y;							
514	if(theEnergy <= edgeMin) {	43596 (17%)	35500 (81%)	5730	7862	24814 (56%)	19478 (44%)	
515	lastIdx = 0;	636 (0%)	519 (81%)	63	120	308 (48%)	318 (50%)	
516	y = dataVector[0];	1202 (0%)	1082 (90%)	111	158	4035 (335%)	924 (76%)	
517	} else if(theEnergy >= edgeMax) {	3170 (1%)	2353 (74%)	651	738	1113 (35%)	1520 (47%)	
518	lastIdx = numberOfNodes-1;							
519	y = dataVector[lastIdx];							
520	} else {							
521	lastIdx = FindBin(theEnergy, lastIdx);	109860 (43%)	76853 (69%)	45196	54010	65906 (59%)	41559 (37%)	
522	y = Interpolation(lastIdx, theEnergy);	84798 (33%)	61193 (72%)	29102	32887	59616 (70%)	35646 (42%)	
523	}							
524	return y;							
525	}	6539 (2%)	5941 (90%)	508	557	1034 (15%)	2554 (39%)	
526								
527	//-----							
528								
529	G4double G4PhysicsVector::FindLinearEner...							
530	{							
531	if(l >= numberOfNodes) { return 0.0; }							
532	size_t n1 = 0;							
533	size_t n2 = numberOfNodes/2;							
534	size_t n3 = numberOfNodes - 1;							



## Performance Expectations: “Two Birds with One Stone”

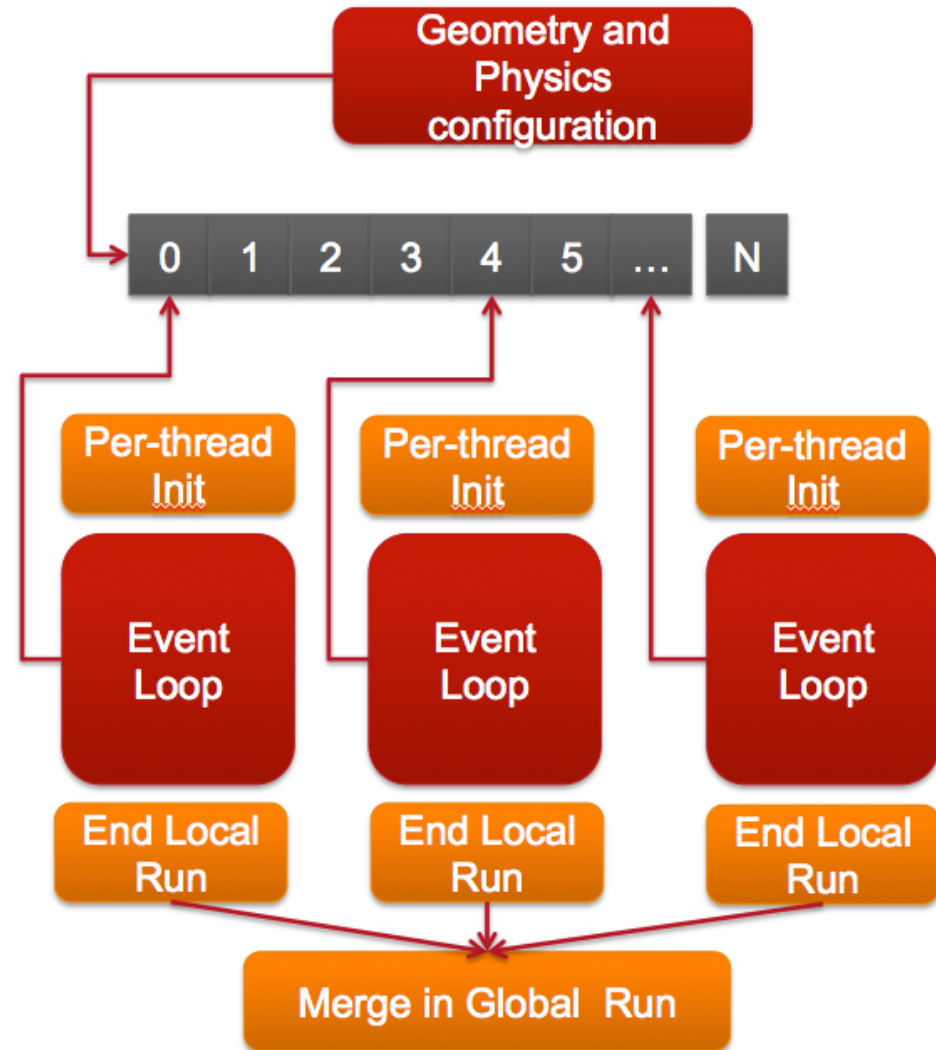
- Performance will be disappointing if code is not optimized for multi-core CPUs
- Optimized code runs better on the MIC platform *and* on the multi-core CPU
- Single code for two platforms + Ease of porting = Incremental optimization



More information in [case study](http://research.colfaxinternational.com) on [research.colfaxinternational.com](http://research.colfaxinternational.com)



- Ensuring a worker thread to join at any time during the event loop of the master
  - After the master thread finishes initialization for geometry and cross-section tables to be shared
- Ensuring a worker thread to leave at any time during the event loop of the master
  - After finishing assigned task (an event or a bunch of events)
- We plan to complete this decoupling with some additional APIs
  - First set of new APIs will come with v10.1-beta in June.
  - Your feedbacks are essential.



# To sum up

- Feedbacks are appreciated. Without users' feedbacks to Geant4-MT prototypes, we couldn't make Geant4 version 10.0.
- Version 10.0 was a big milestone for us as it was the first production version of Geant4 in multithreading mode. But it is not our ultimate goal in terms of making Geant4 multithreaded.
  - Our next goal includes complete decoupling of master event loop and worker initialization/termination so that each worker thread may join/leave at any time during the event loop with minimal initialization/termination overhead. We plan to deliver first additional APIs at 10.1-beta.
  - Your further feedbacks on version 10.0 are most valuable.
- We admit our user's guide is not perfect, in particular for advanced users who need to extend/alternate kernel functionalities to control worker thread.
  - We will improve our document.
  - Also, denser communication between experiments and Geant4 developers is mandatory.
    - Proposing a mini-workshop on May 12-13
- Next Geant4 Technical Forum is tomorrow at 16:00 CET.