

Geant4 Status

Presented by A. Dotti for the Geant4 collaboration
Second Annual Concurrency Forum

- Geant4 Version 10: general design
- Results
- Outlook for the future

Introduction

- 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

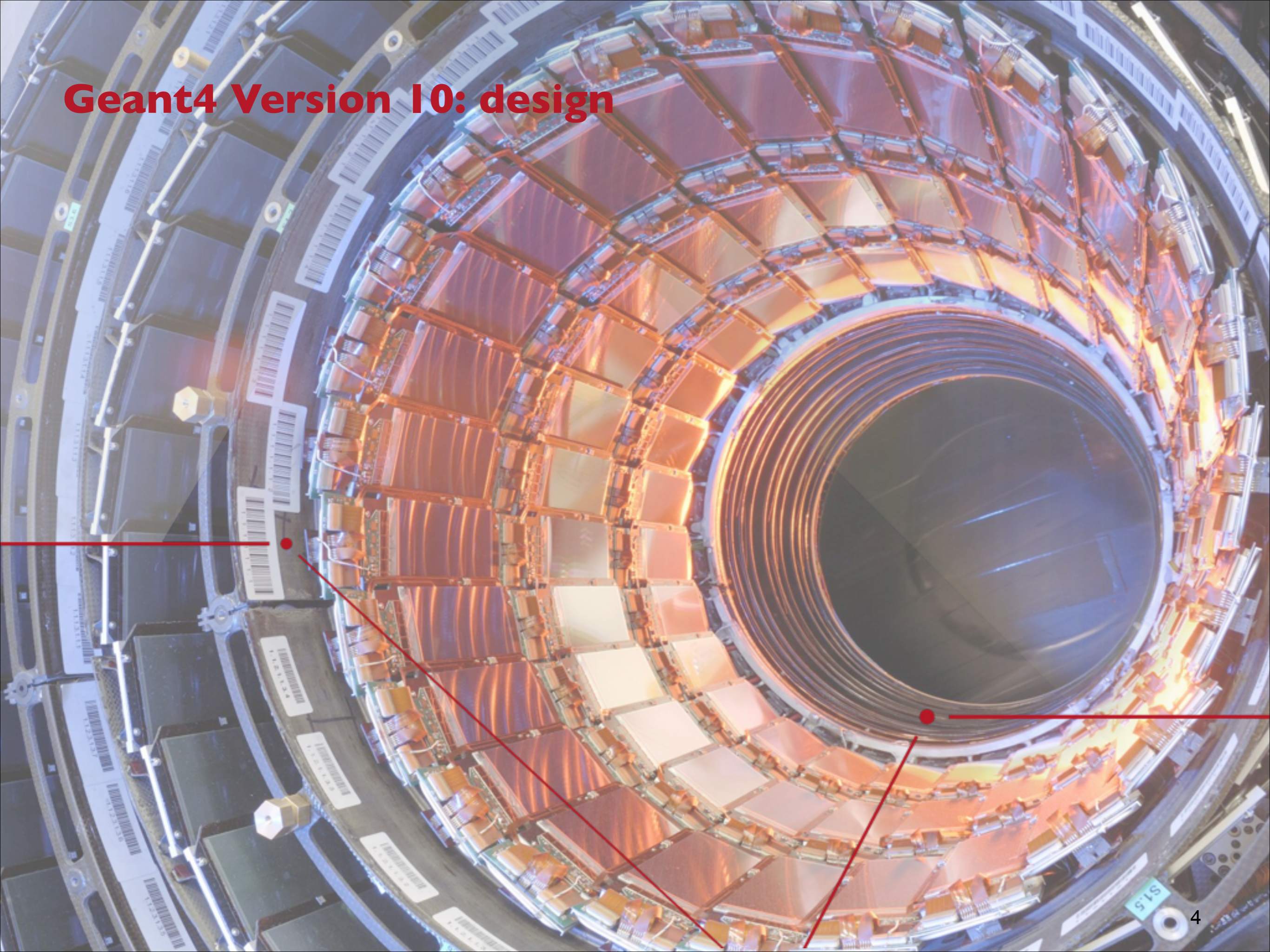
- MT code integrated into G4

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

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

- Further refinements and optimizations

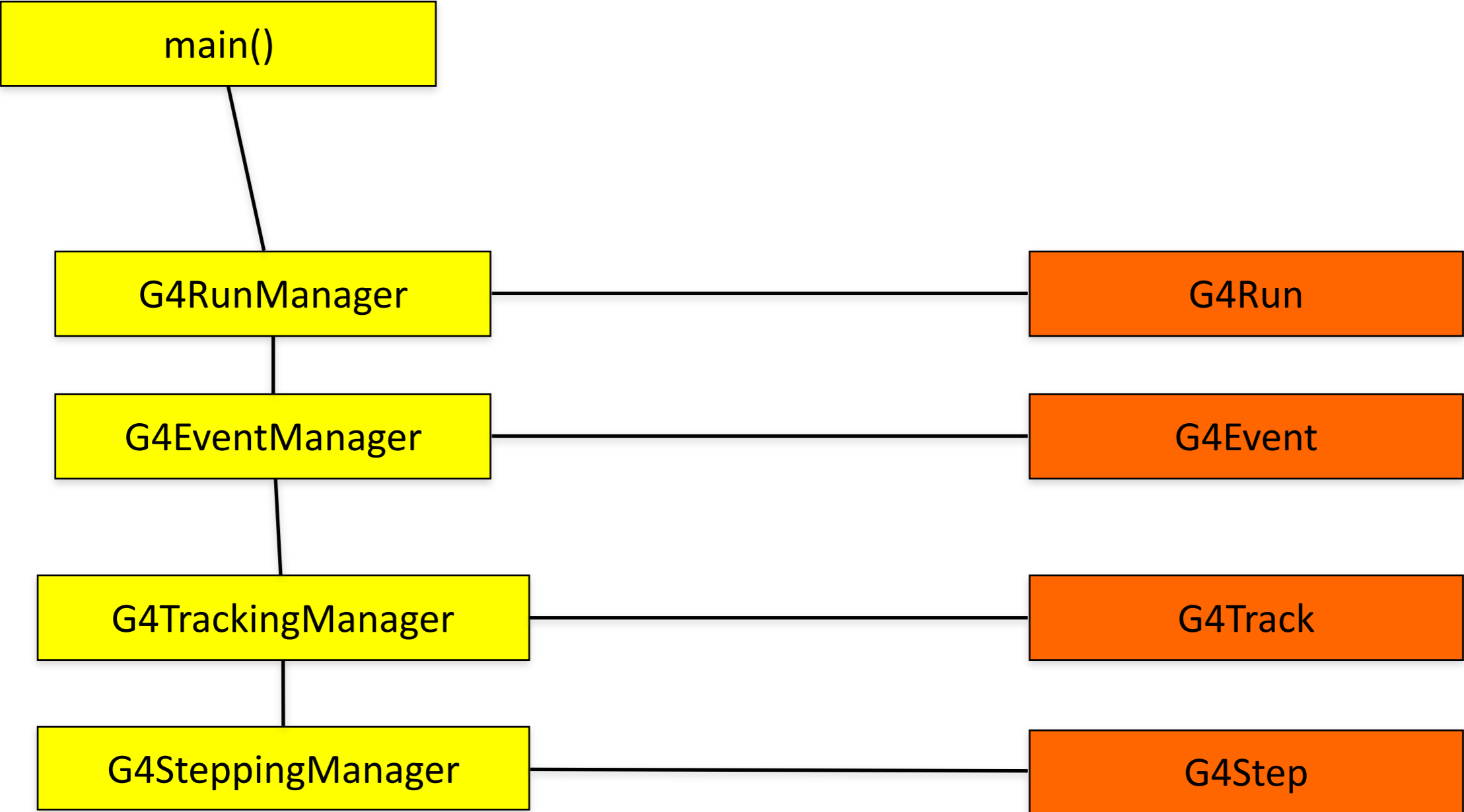
Geant4 Version 10: design



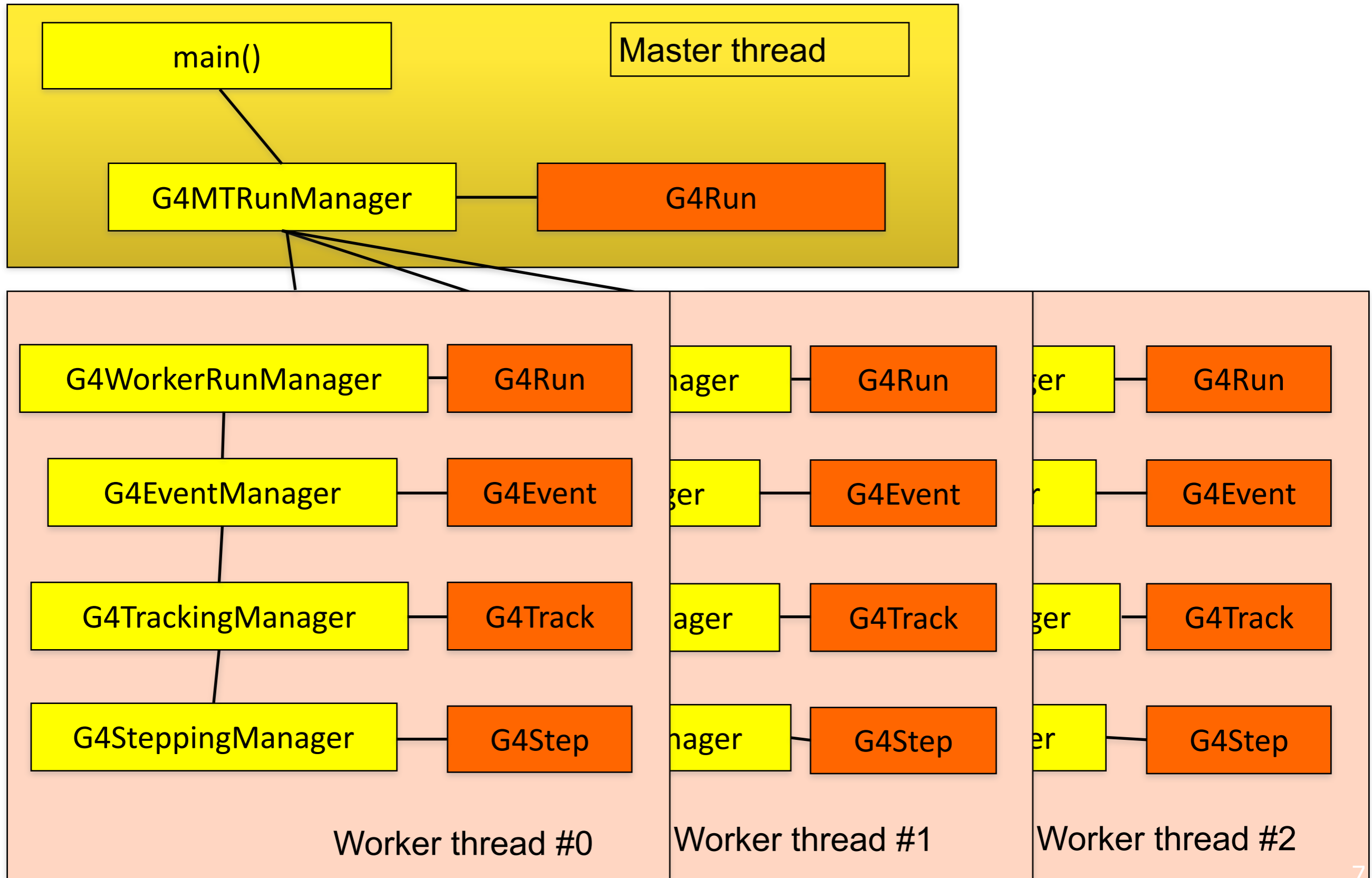
Geant4 Version 10: event-level parallelism

- 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.

Sequential mode



Multi-threaded mode

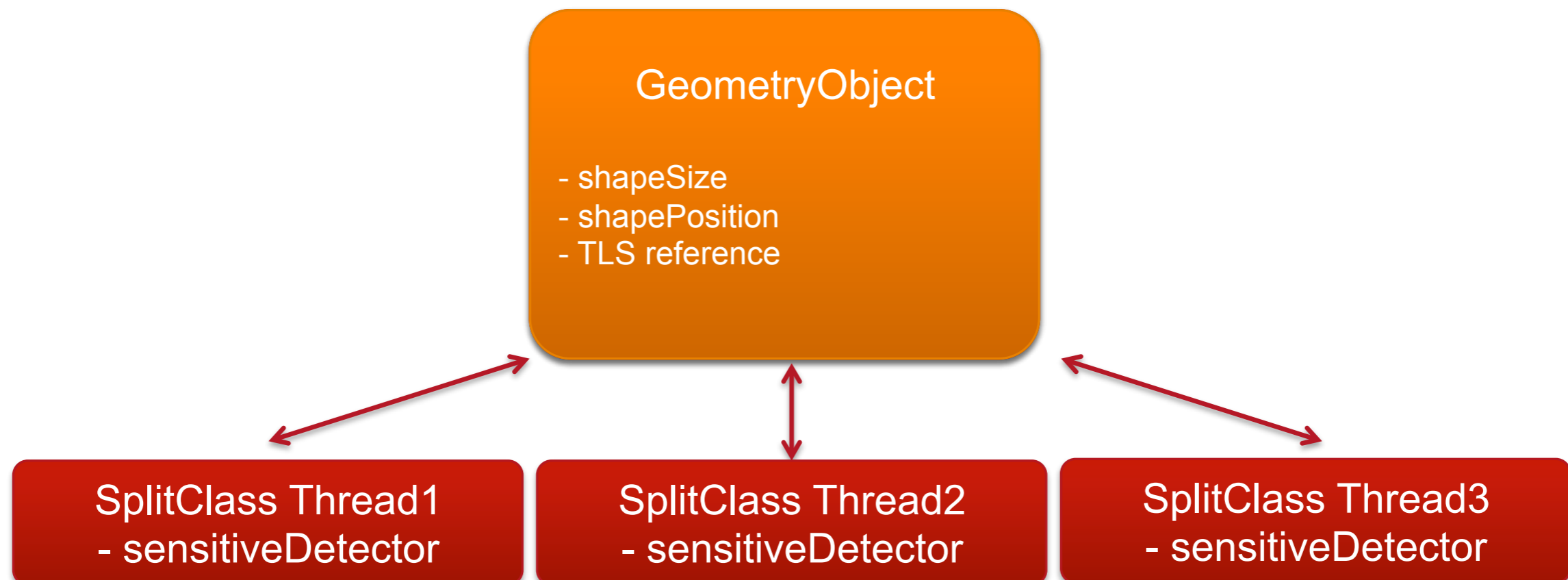


Thread-private vs shared

- 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.

Thread-safety in Version 10.0

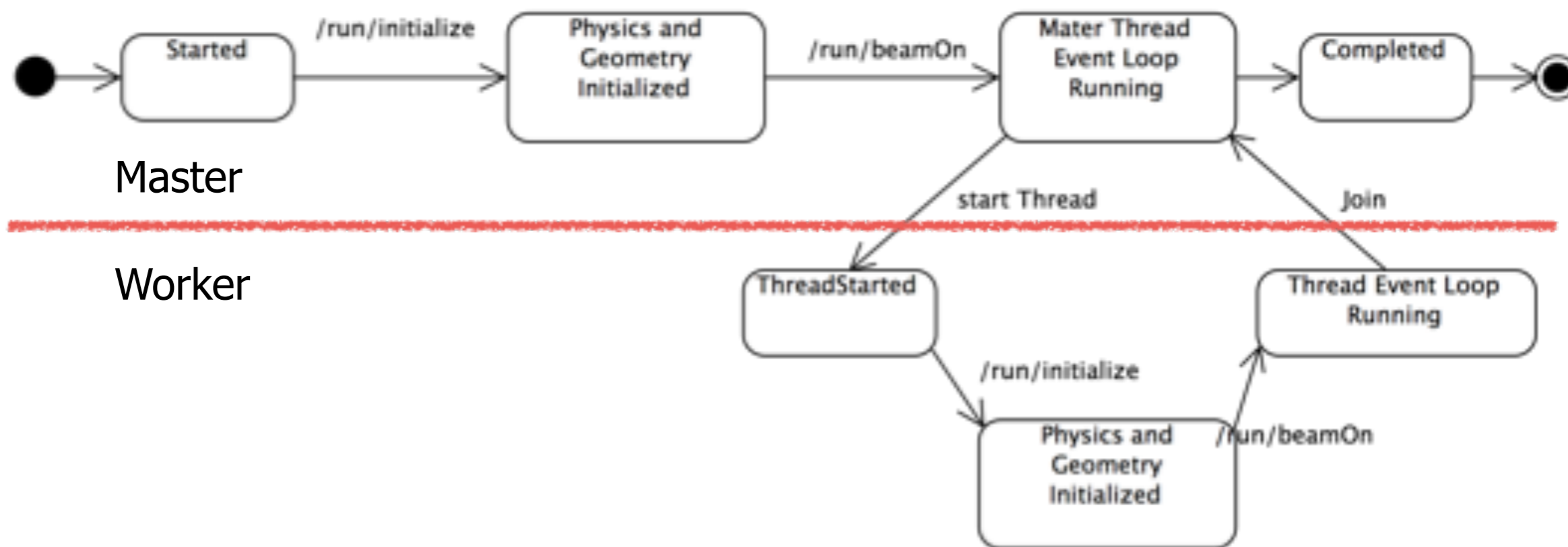
- Design: lock-free code during event-loop
- Thread-safety implemented via **Thread Local Storage**
- “Split-class” mechanism: reduce memory consumption
 - Read-only part of most memory consuming objects shared between thread: geometry, (EM) physics tables
 - Rest is thread-private



To avoid confusion...

- Multi-threading capabilities in G4 is more than “spawning tasks in parallel”
- Geant4 code is “parallel-aware”
 - Effort in: thread-safe, substantial memory sharing, API definition, utilities and wrappers
- This allows for toolkit to be **integrated in any parallelization framework**
 - e.g. we already have MPI and TBB examples
 - To be improved: better memory handling for TBB
- We **also** provide a POSIX-based thread management system: G4MTRunManager
 - Enough for simple applications
 - As for the sequential G4RunManager, we expect large experimental frameworks to extend this as starting point

- 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

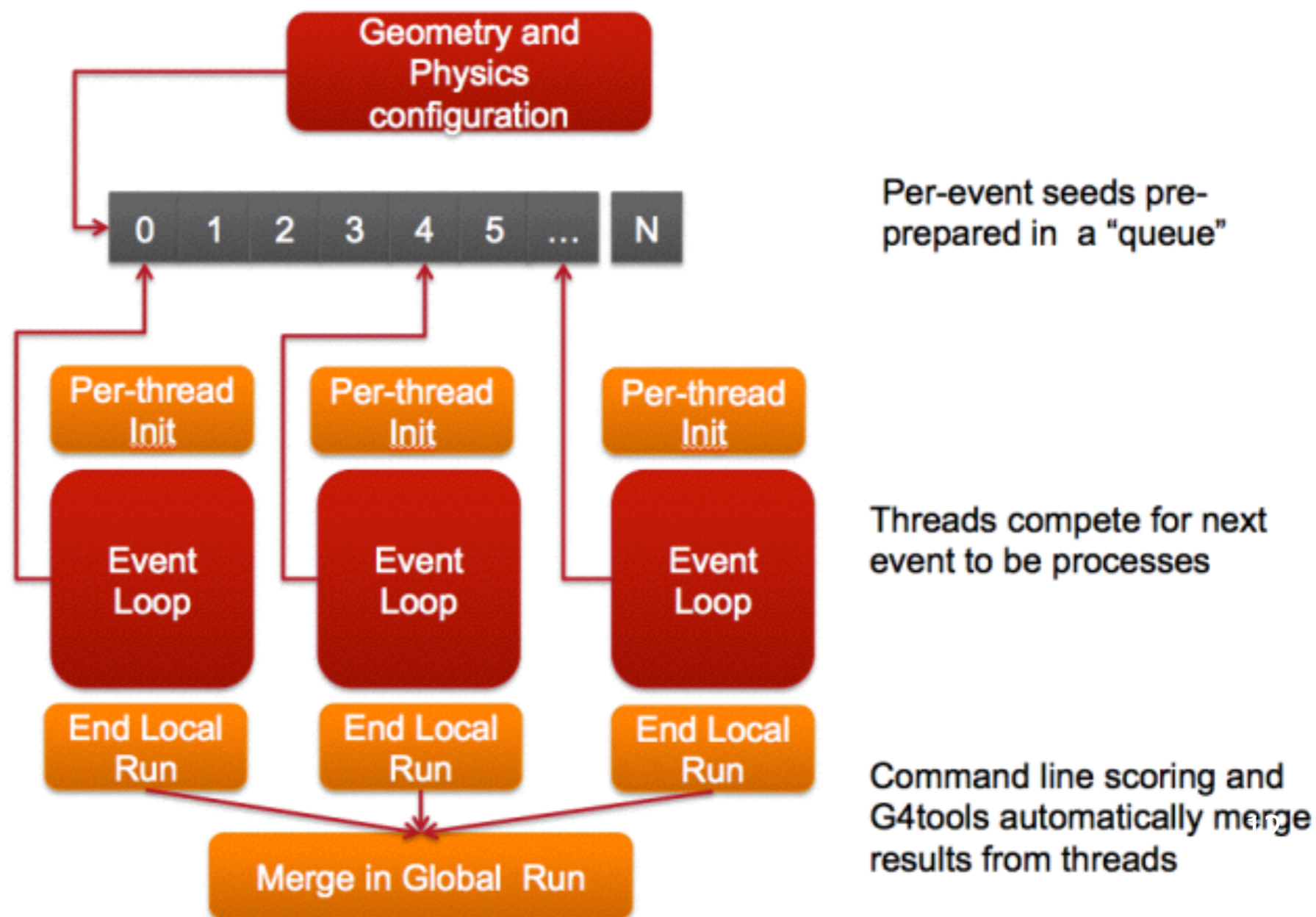


Nota Bene

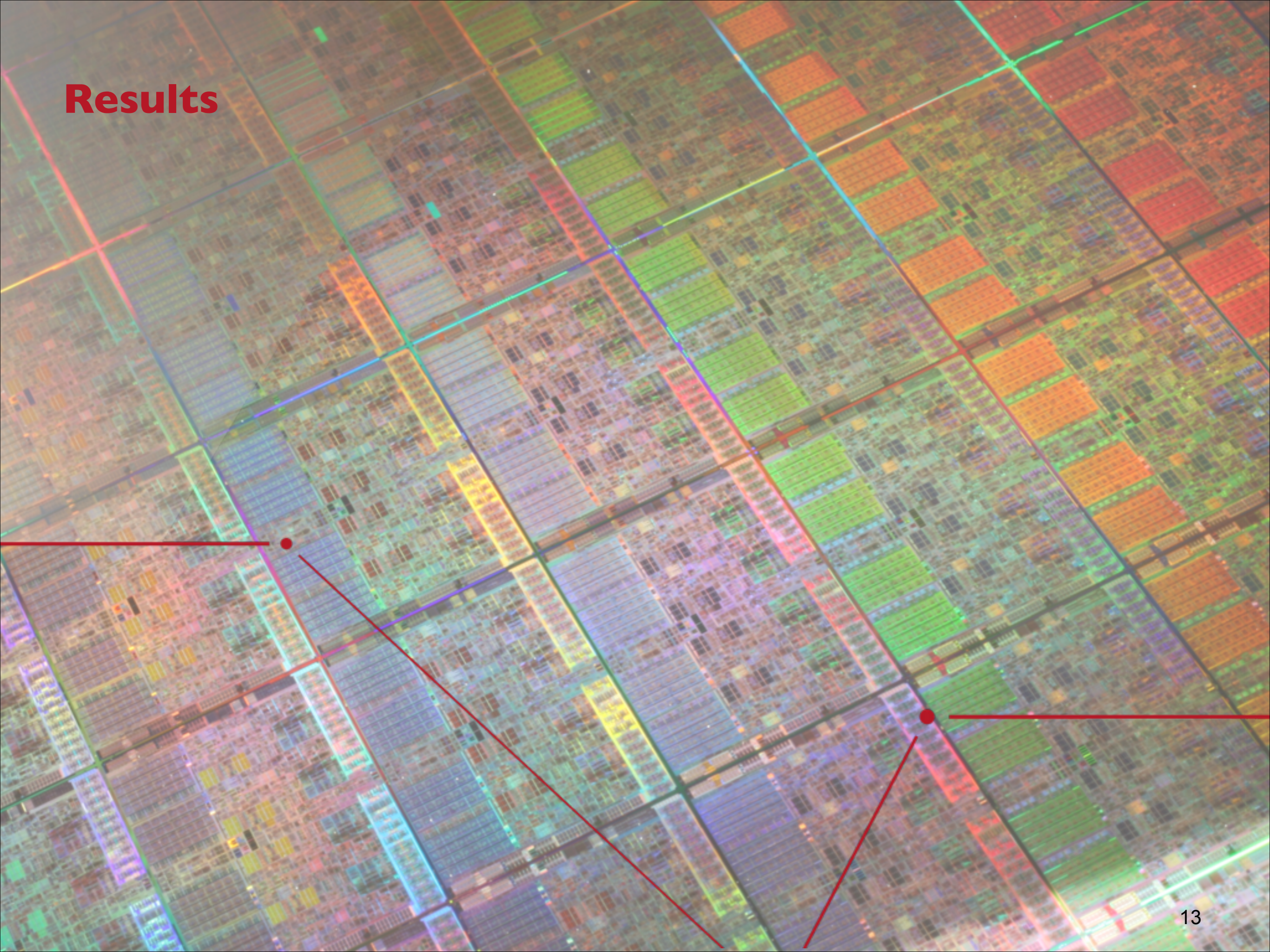
This is a simplified schema: threads are created on run initialization so that they are accessible to interactive commands if needed

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.

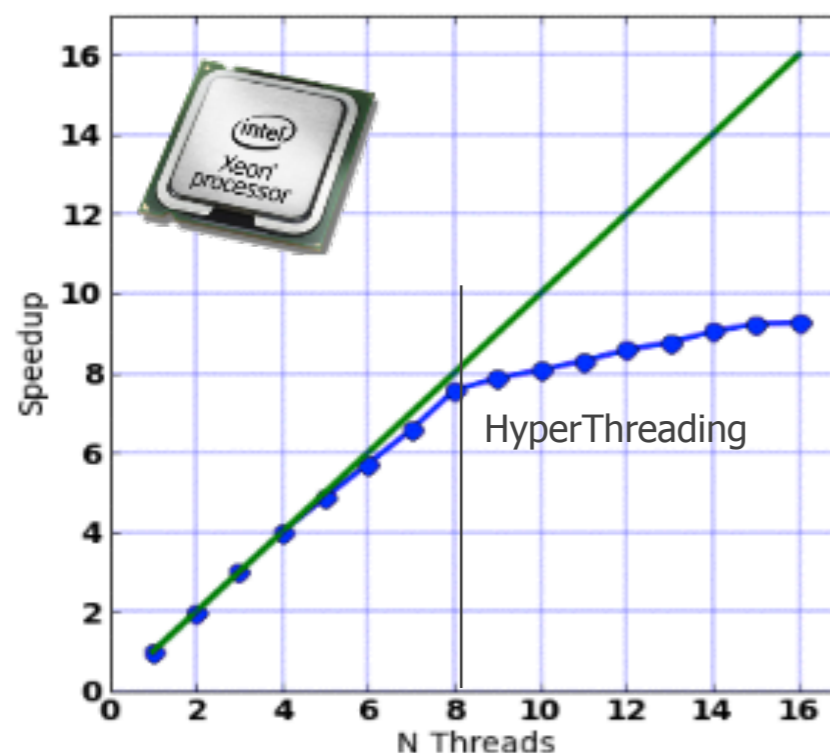


Results

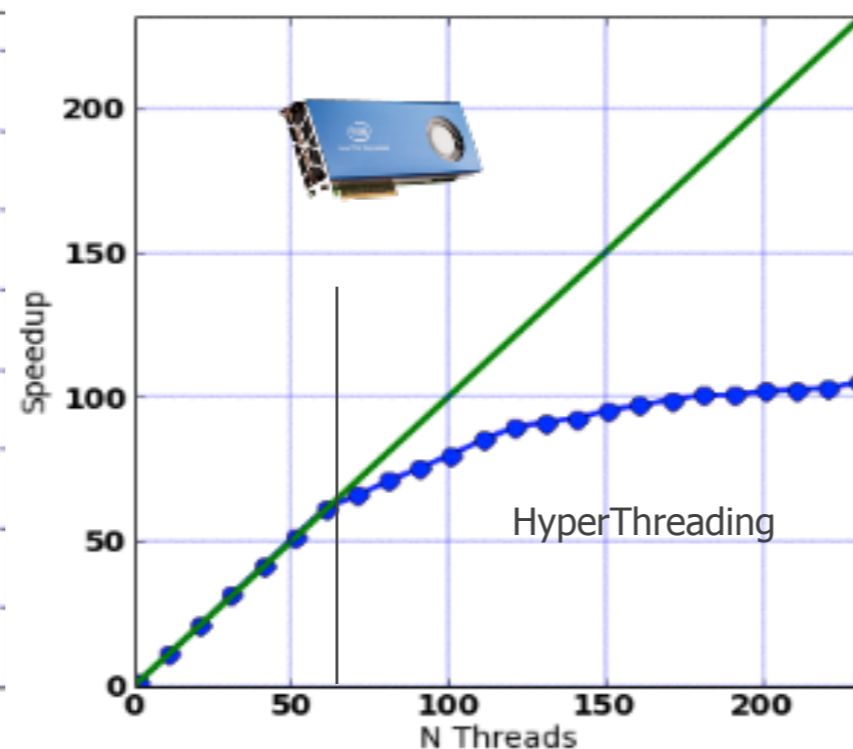


- Current release has already shown good scalability on a number of different architectures: Intel Xeon servers, Intel Xeon Phi co-processors and low-power ARM processors.
 - On Intel architectures, it has shown performance improvements not only up to the number of physical cores but in hyper-thread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

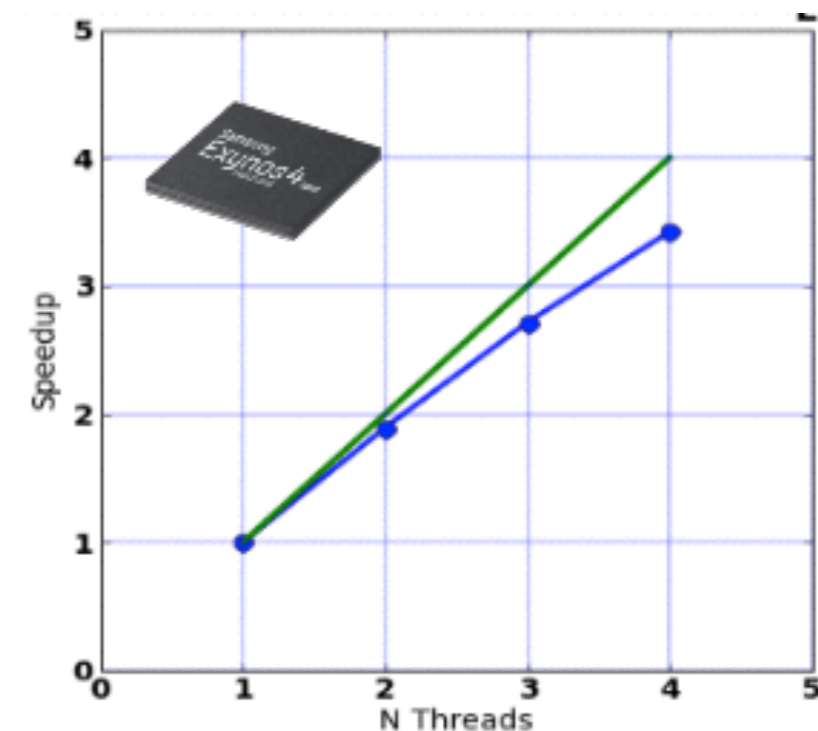
Intel Xeon L5520 @ 2.27GHz



Intel Xeon Phi 7120P @ 1.238GHz





Exynos 4412 Quad-Core @ 1.7 GHz

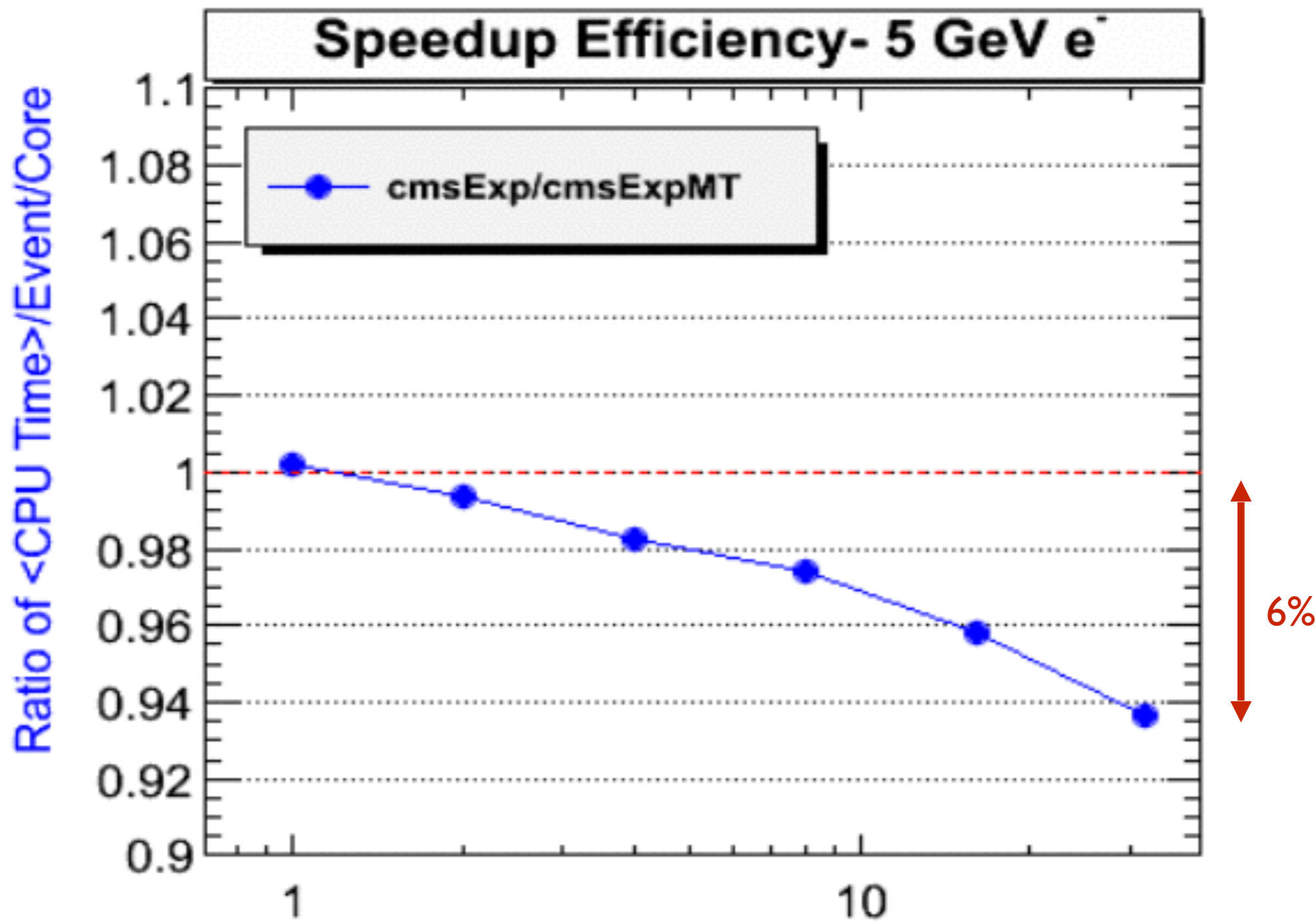


Thanks to Openlab for providing Intel Xeon Phi hardware
Thanks to CMS for providing ARM ODROID boards

General considerations

- **Fully reproducible:** given an event and its initial seed the RNG history is independent of the number of threads and order in which these are simulated
 - Corollary 1: given the seeds, sequential and MT builds are equivalent
 - Corollary 2: being able to reproduce a single event in a dedicated job (i.e. crashes)
- MT functionality introduces **minimal overhead** ($\sim 1\%$) w.r.t. sequential
- Very good **linear speedup** up to very large number of threads $O(100)$
- Good **memory reduction:** only 30-50MB/thread (depends on application)
- Hyper-threading adds additional +20% throughput
- Working out-of-the-box with success on **different architectures:** x86, ARM, MIC,  Atom,  IBM Bluegene/Q

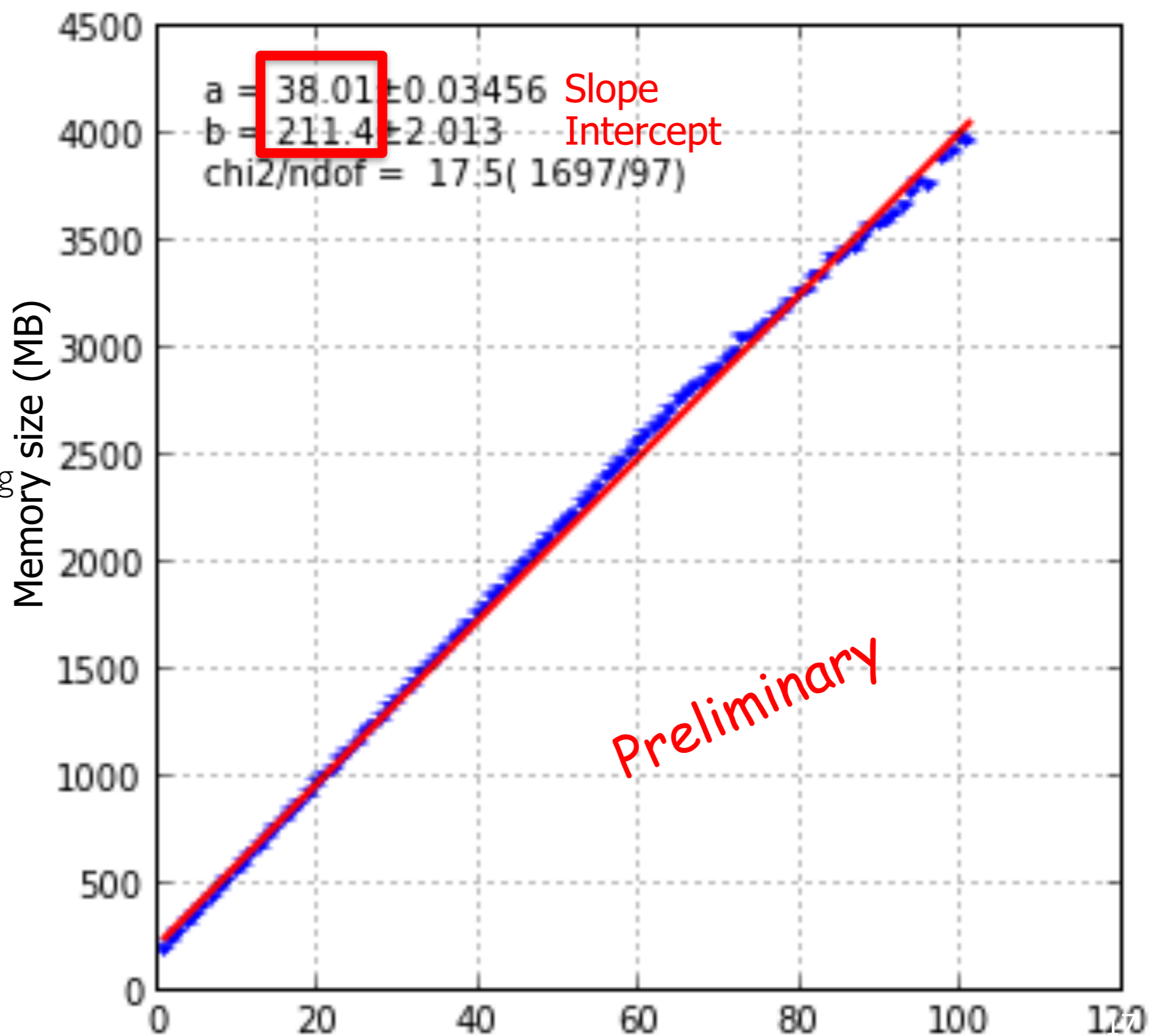
More details and comparisons tomorrow presentation on G4 on MIC

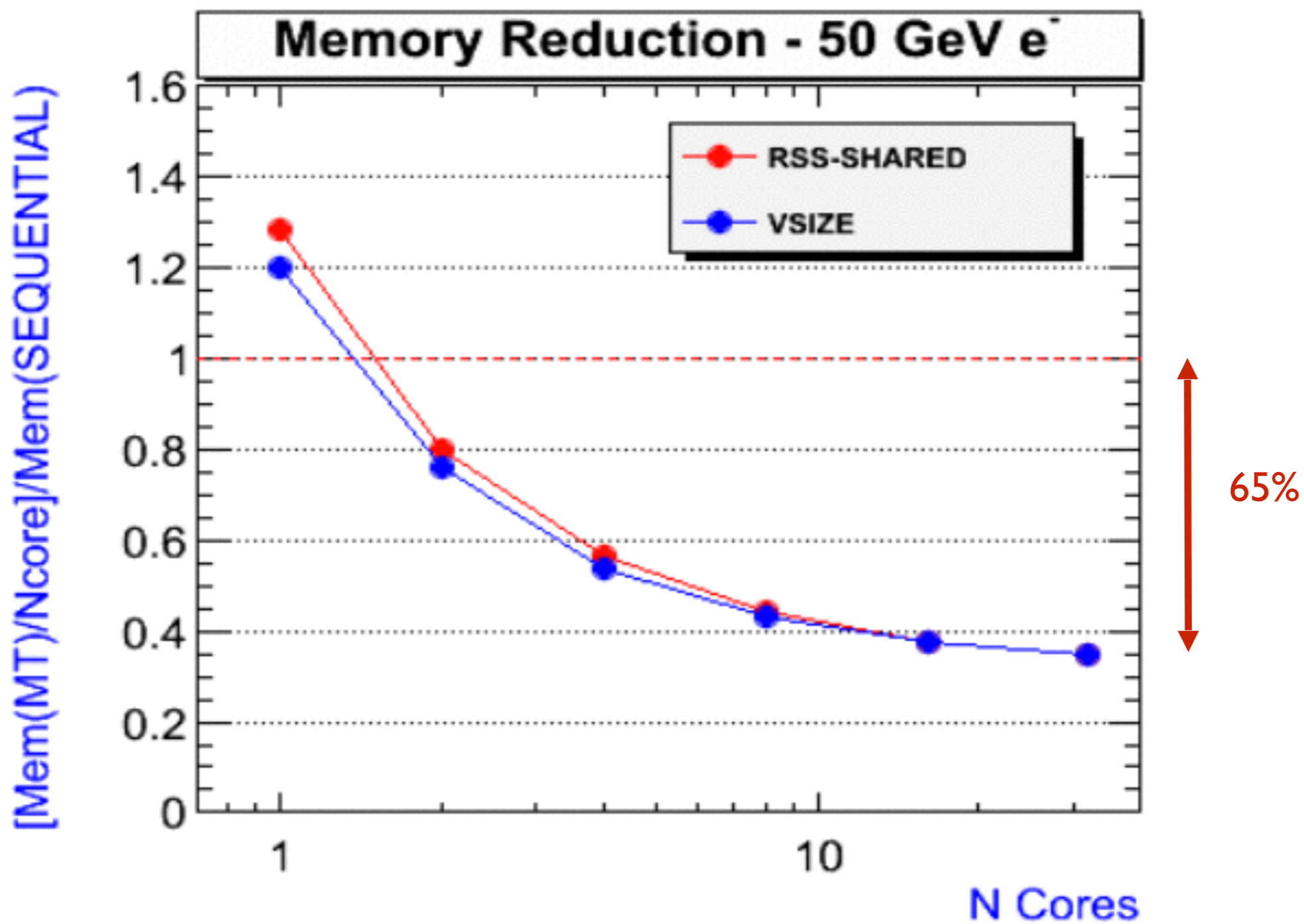


“CMS-style” geometry

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 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

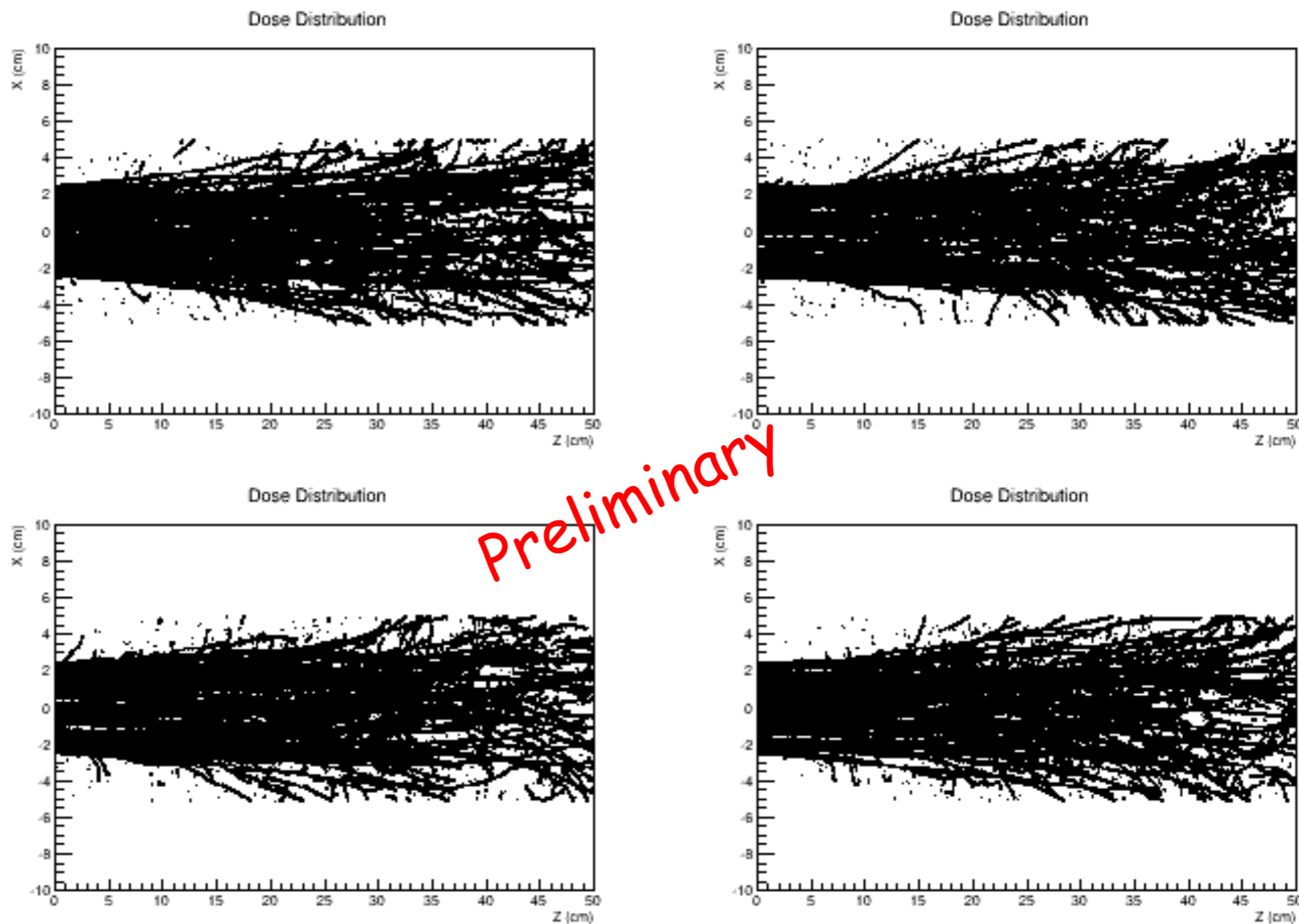




“CMS-ish” geometry

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



- 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.

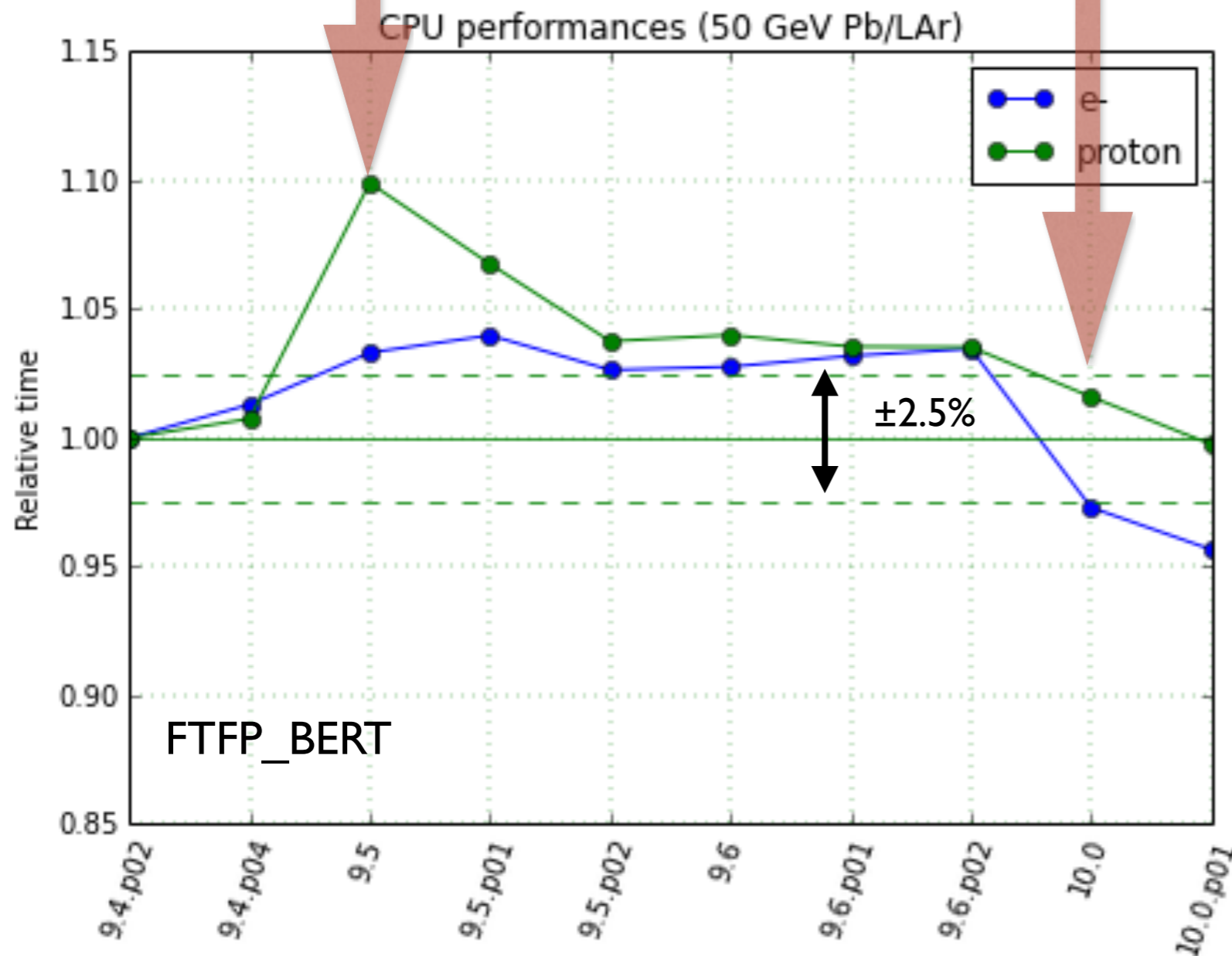
Feedback from LHC

- We appreciate all large LHC experiments (ALICE, ATLAS, CMS, LHCb) have plans to investigate use of Geant4 Version 10.0 and potentially multi-threading
- Positive feedback received at Second LPCC Detector Simulation Workshop
- Full presentations at LPCC Detector simulation workshop available at: <http://goo.gl/G9Gvle>
- Close communication with experiments is fundamental!

Absolute throughput (sequential)

Heavy developments: FTF becomes competitive with QGS

Fast Log/Pow mathematics

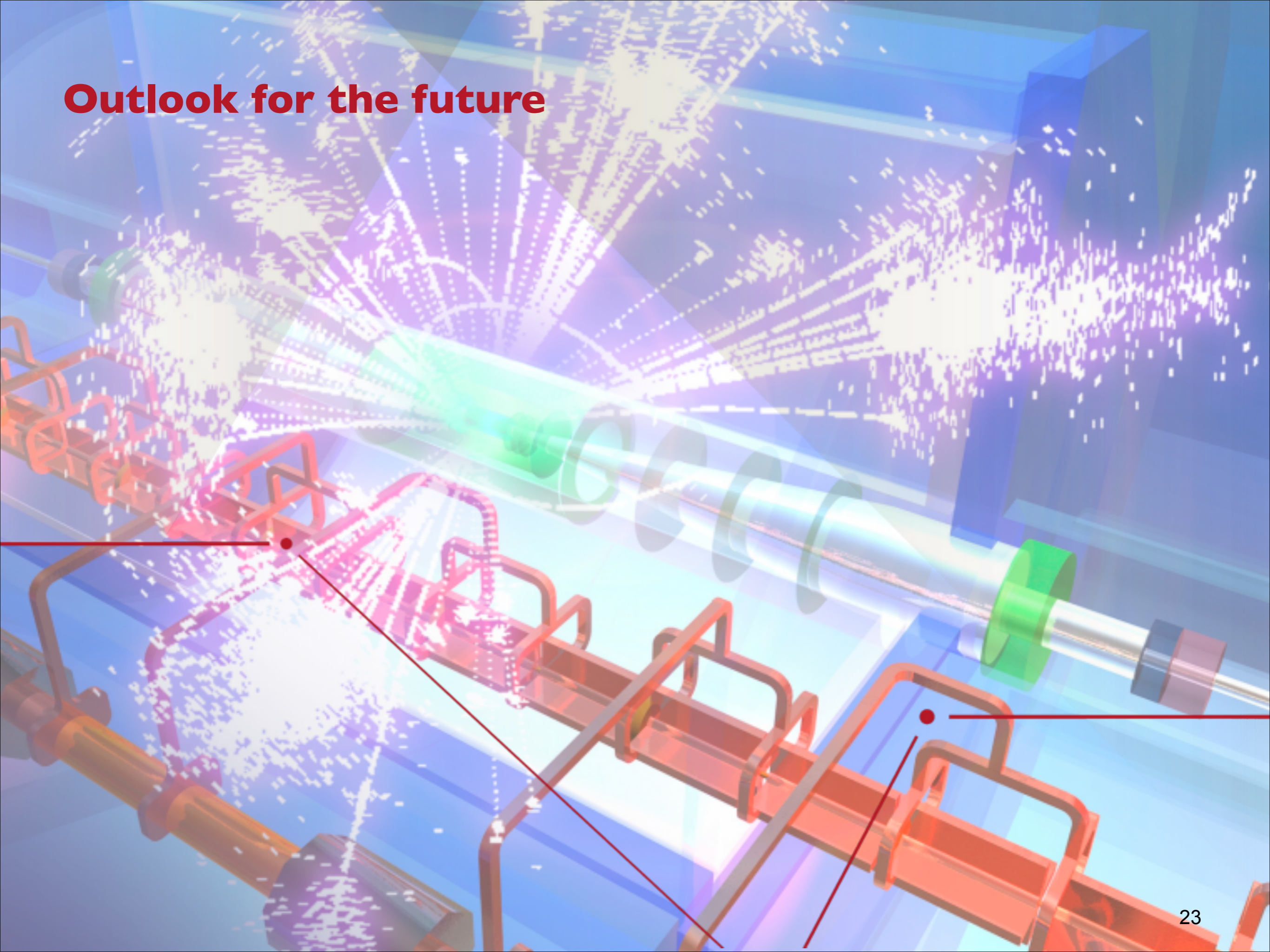


Improvements for MT also bring benefits to sequential

We have substantially improved physics (extended HAD theory driven processes, more precise EM tables, new processes) and at the same time improved CPU performances.

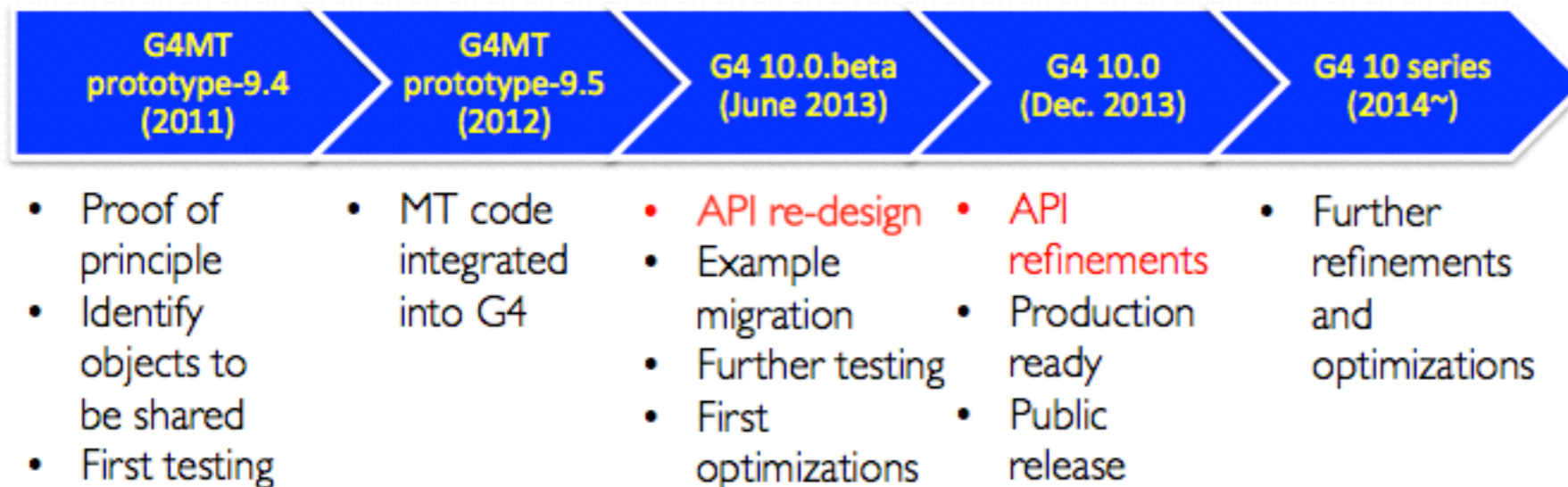
We believe there are more opportunities for optimizations in our code and we are actively working on them

Outlook for the future



Beyond Geant4 version 10.0

- Geant4 version 10 series will be evolving.

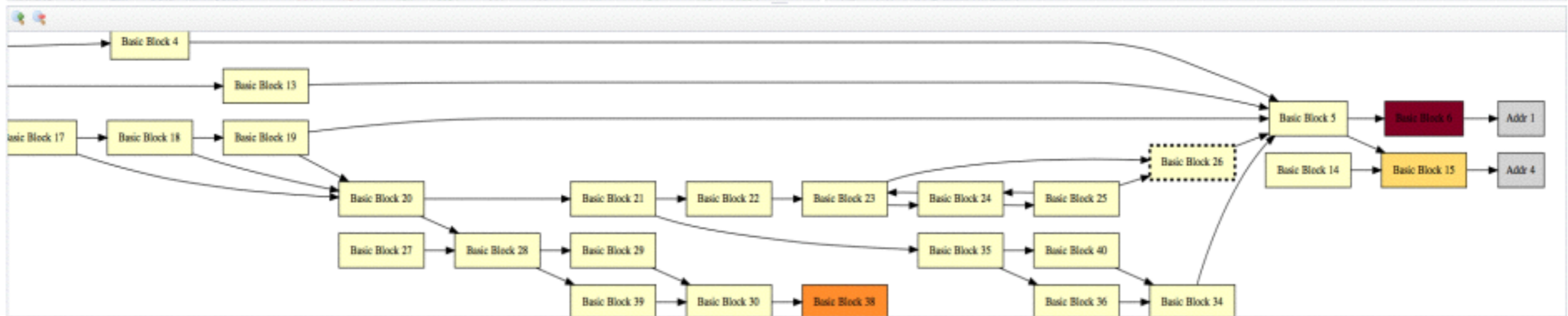


- Performance improvements
 - Algorithm optimization / local vectorization
 - without losing code readability / maintainability / flexibility
 - Optimization of file access
- Memory space reduction in particular for per-thread memory
 - Sharing more physics vectors and other objects among threads
- Multithreading 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

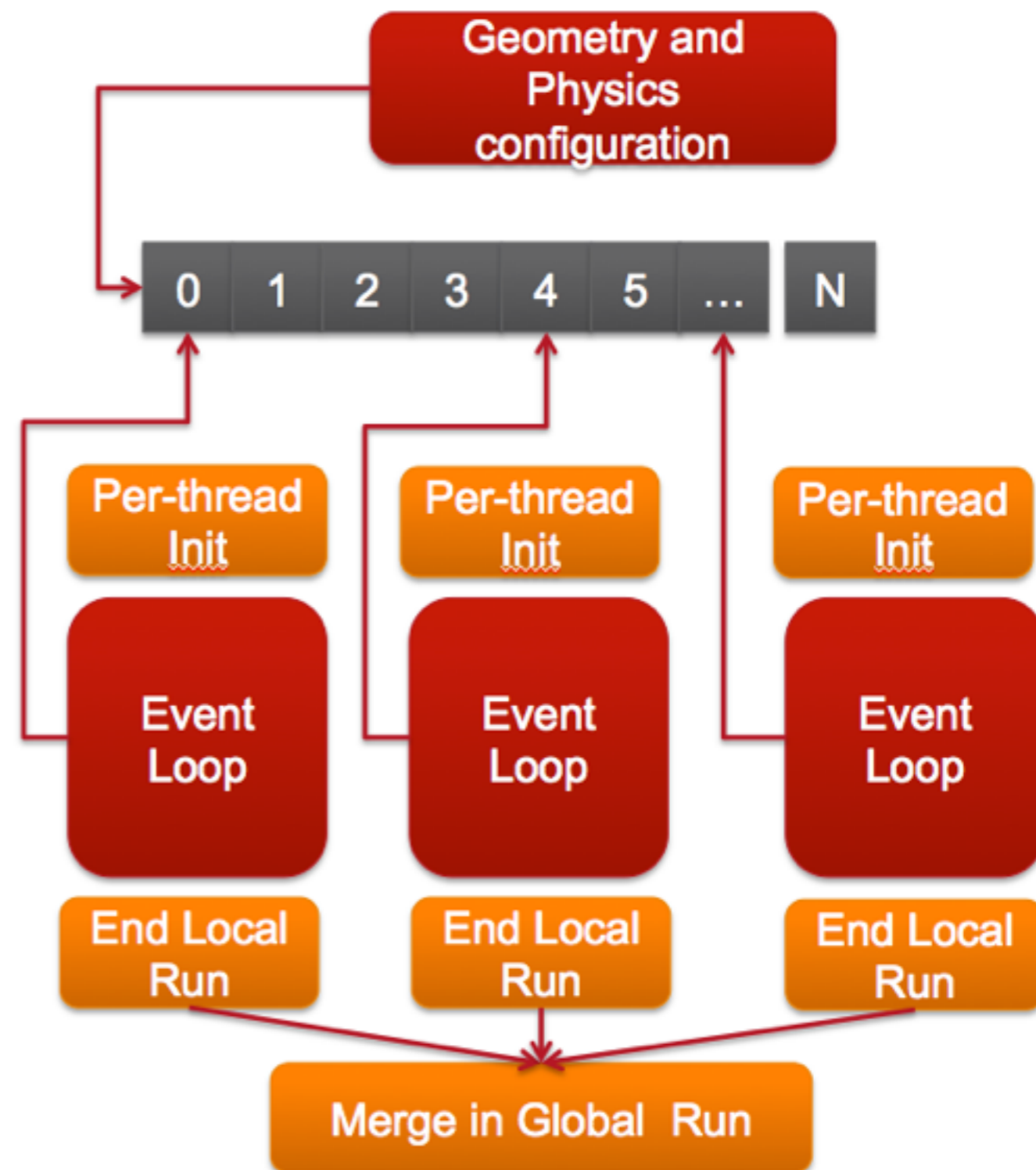
- We are working with Google on performance measurements of Geant4-based application using Gooda tool, a PMU-based event data analysis package.

address	princ_id	disassembly	unhalted_core_cycles	uops_retired:stall_cycles	instruction_retired	uops_retired:any	load_latency	instruction_starvation	bandwidth
0x30ce8	521	Basic Block 26 <0x30b2b>	268 (0%)	146 (54%)	95	105		20 (7%)	
0x30ce8	521	Tea 0x0(,%rax,8),%r8	139 (0%)	102 (73%)	32	38		20 (14%)	
0x30cef	521	null							
0x30cf0	521	Tea 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 50x34,%rcx	517 (0%)	373 (72%)	222	271	30 (5%)	149 (28%)	
0x30d12	521	sub 50x3ff,%ecx	119 (0%)	88 (73%)	32	60			
0x30d18	521	cvtst2sd %ecx,%xmm4	199 (0%)	95 (47%)	40	60		20 (10%)	
0x30d1c	521	mov 50x800fffffffffffff..	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 50x3fe00000000000..							
0x30d30	521	null							
0x30d33	521	or %rcx,%rax							

line number	source	unhalted_core_cycles	uops_retired:stall_cycles	instruction_retired	uops_retired:any	load_latency	instruction_starvation
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::FindLinearEnerg..						
530	{						
531	if(1 >= numberOfNodes) { return 0.0; }						
532	size_t n1 = 0;						
533	size_t n2 = numberOfNodes/2;						
534	size_t n3 = numberOfNodes - 1;						



- 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.



- **Further reduce memory consumption.** Rule of the thumb: fit complex simulations w/ $O(100)$ threads in $O(\text{GB})$ memory
 - e.g. typical computing power of accelerators
- In our experience: minimize memory usage can sometime **conflict** with other performance considerations (e.g. reduce memory “churn” via caching need special attention for thread-safety)
- Most memory consuming objects: geometry and EM physics
 - Efficient memory reduction already achieved in 10.0.beta
 - Next: need to **concentrate on Hadronics physics** (especially: cross-sections, specific models with large not-shared tables -BIC-)

Conclusions

- 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.
- Good results obtained: scalability, memory reduction, CPU improvements
- We do expect improvements for Geant4 Version 10.1 (December 2014)
 - complete decoupling of master event loop and worker initialization/termination so that each worker thread may join/leave at any time