Geant4 Version 10:

The challenges of the many-core computing era

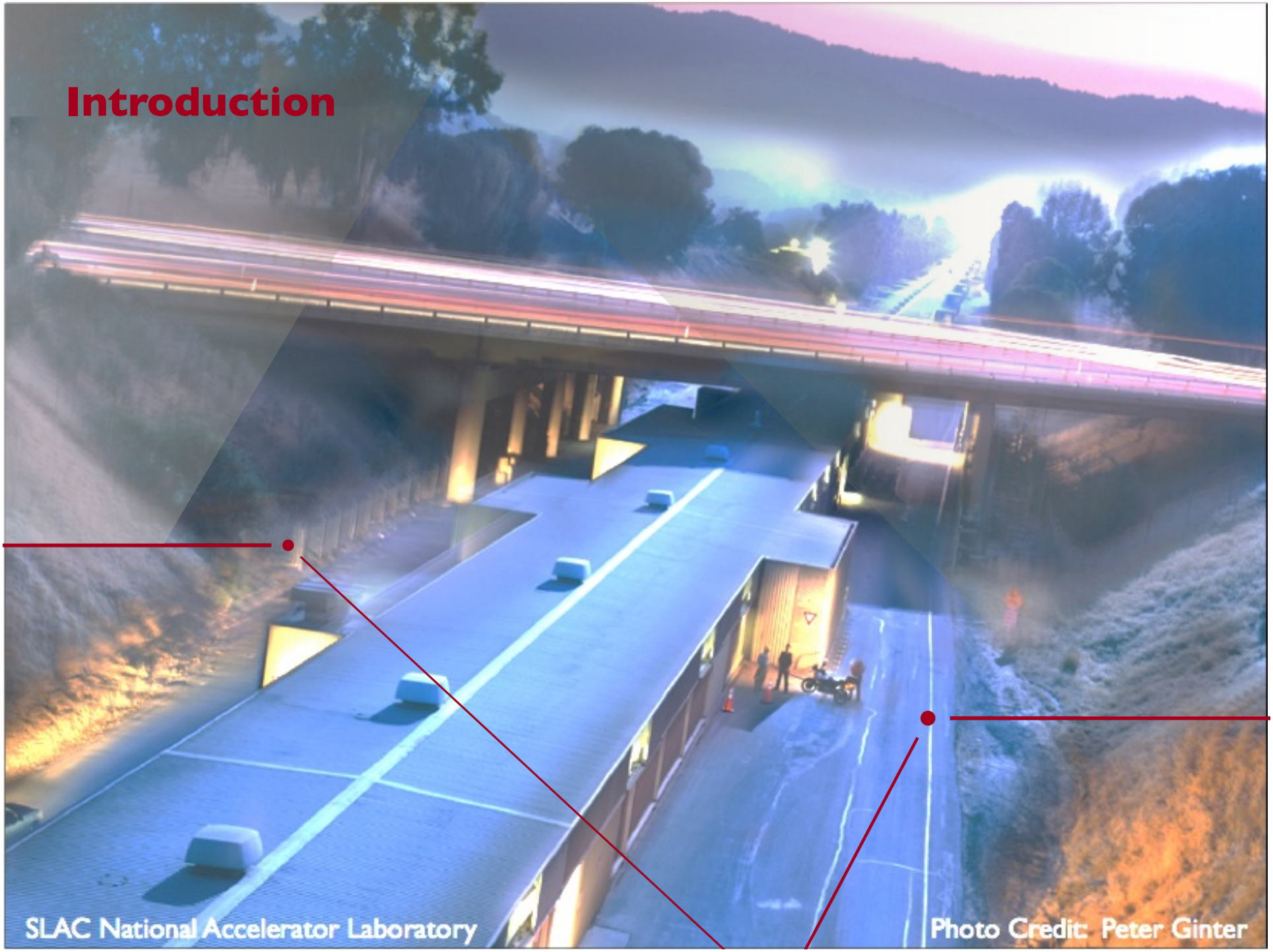
A. Dotti for the Geant4 multithreading task-force

PPA/SCA Division Meeting ; June 26, 2013

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Introduction



What's new in Geant4 Version 10?

SLAC

- Next version of Geant4 (December 2013) will be a **major release**
  - Version 10.0.beta in its way this week
- Incudes several improvements
  - All categories will include important improvements
  - SLAC: improved graphic; improvement of hadronic physics (intermediate/ low-energy cascade, low-energy neutron); kernel re-design
- **Main highlight is Multi-threading capabilities**
  - Introduce event-level parallelism
  - Effort coordinated and maintained at SLAC

Geant4 MT: from prototypes to production

SLAC

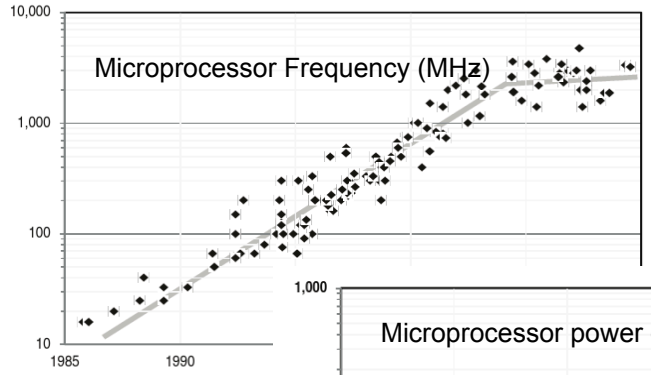
- MT code integrated into G4
- Public release
- All functionalities ported to MT

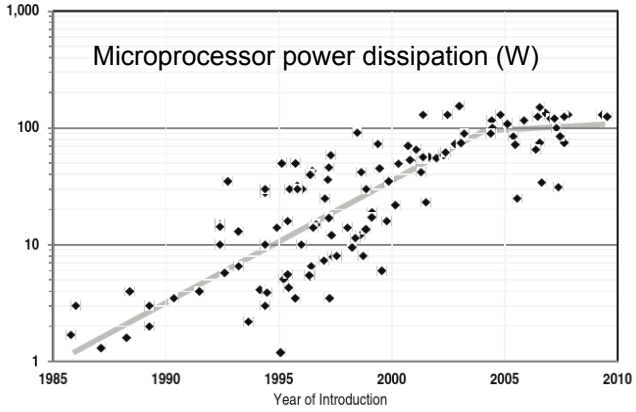
- Proof of principle
- Identify objects to be shared
- First testing

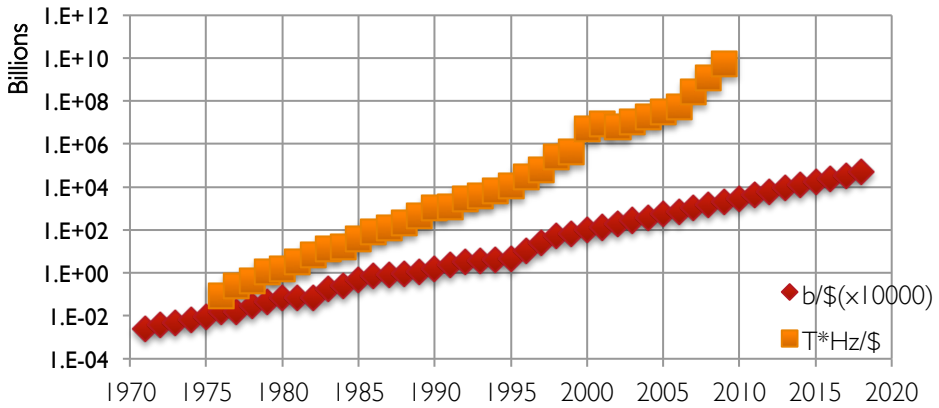
- API re-design
- Example migration
- Further testing
- First optimizations

- Further Refinements
- Focus on further performance improvements

Why parallelism? (a reminder)



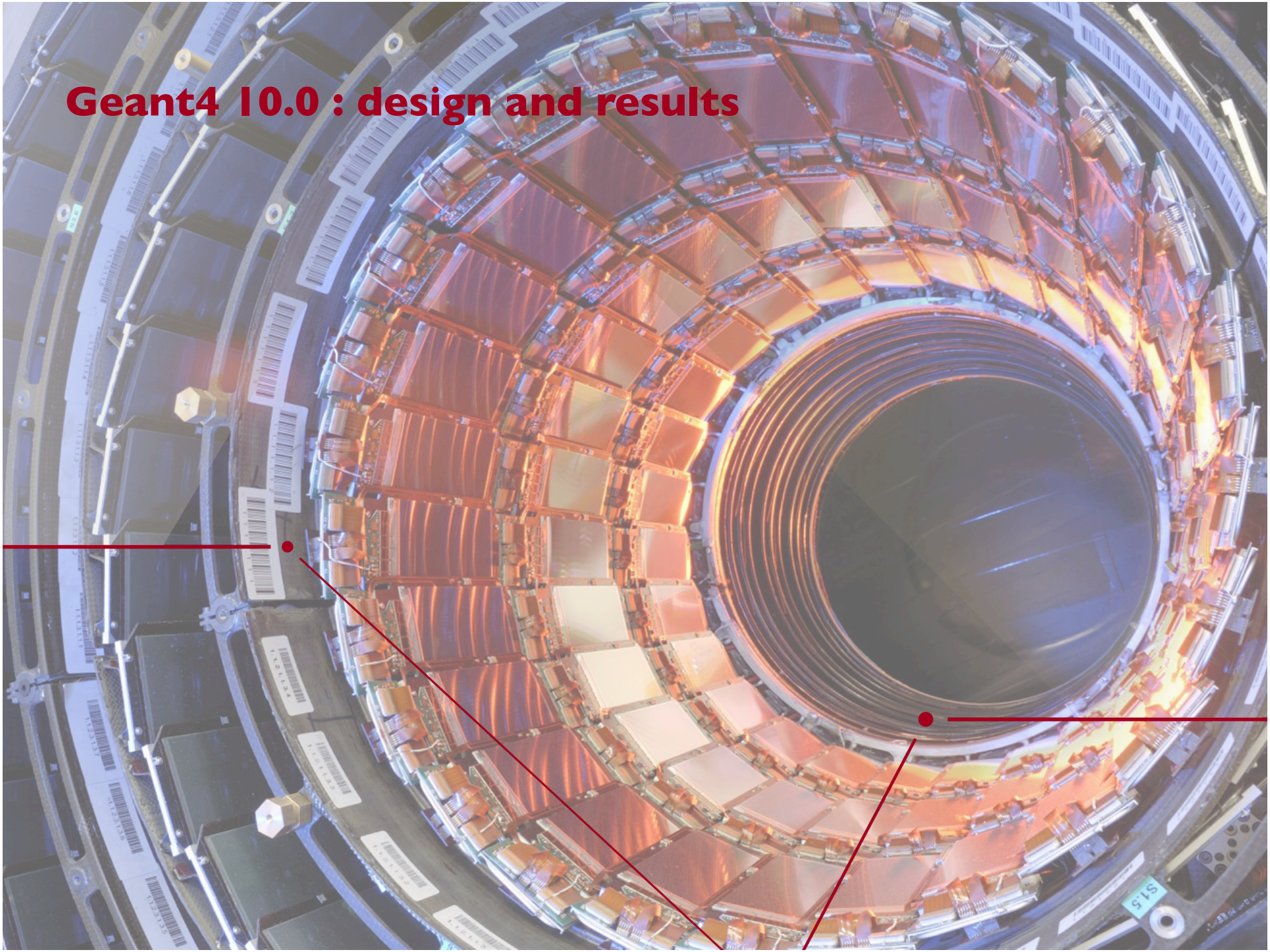




- Increase frequency of CPU causes **increase of power needs**
- Reached plateau around 2005
  - No more increase in CPU frequency
- However number of transistors /$ you can buy continues to grow
  - Multi/May-core era
- Note: quantity memory you can buy with same $ scales slower
- **Expect:**
  1. Many core (double/2yrs?)
  2. Single core performance will not increase as we were used to
  3. Less memory/core
- New software models need to take these into account: increase parallelism

CPU Clock Frequecy and 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. 16
Average transistor price: Intel and Dataquest reports (December 2002), see Gordon E. Moore, "Our Revolution,"

Geant4 10.0 : design and results

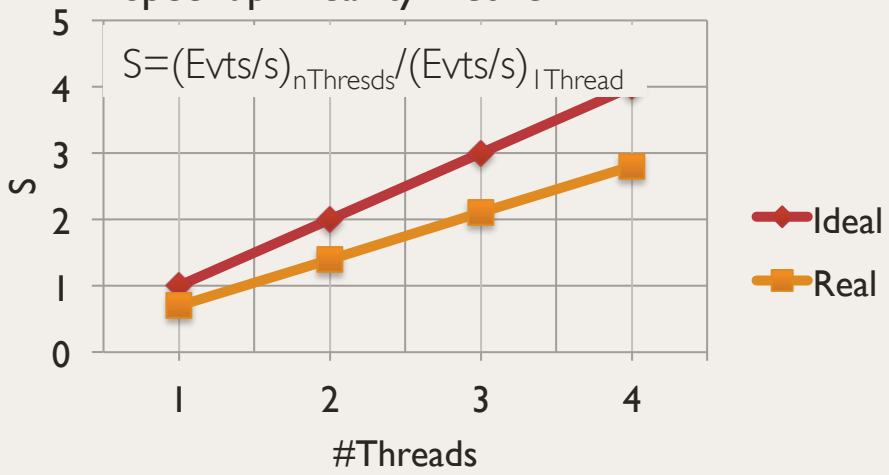


Geant4 Multi-threading: event level parallelism

SLAC

- **Design to minimize changes in user-code**
  - Maintain API changes at minimum
- Focus on **"lock-free" code**: linearity of speed-up (w.r.t. #threads) is the metrics we are currently concentrating on (then we'll optimize absolute throughput)
- Enforce use of **POSIX standards** to allow for integration with user preferred parallelization frameworks (e.g. MPI, TBB, …)

Speedup linearity metric



Absolute throughput metric

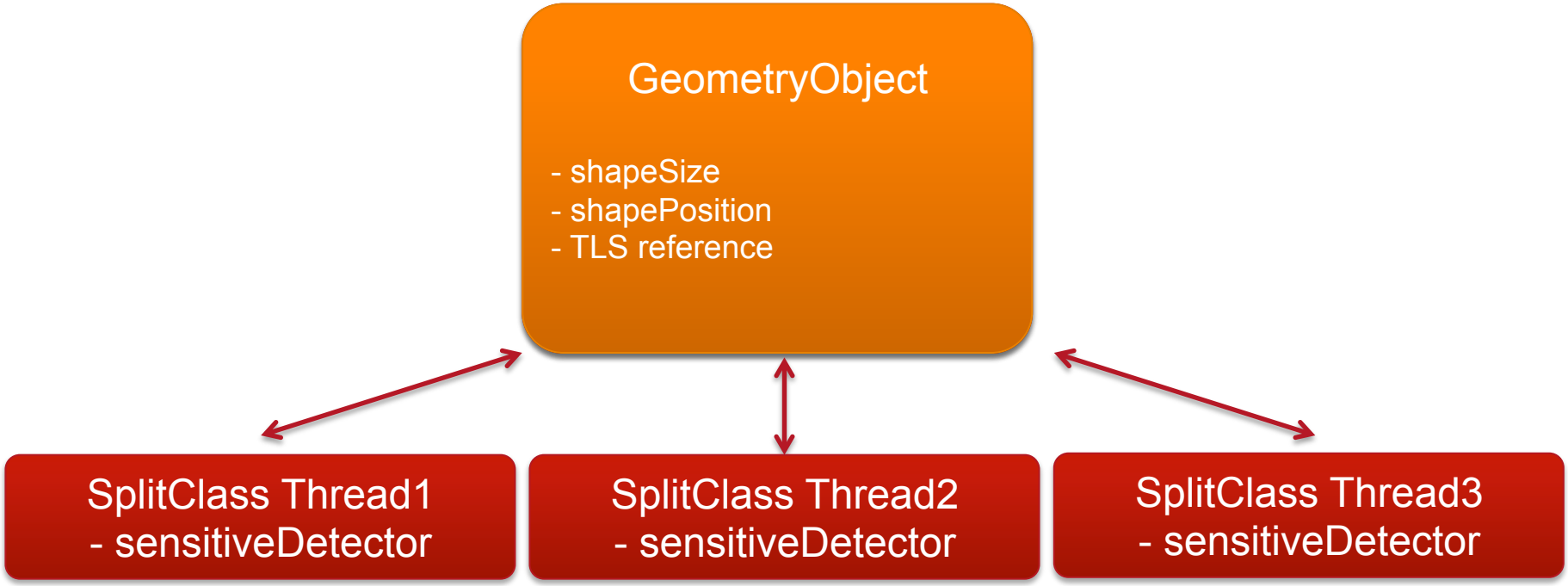
Sequential	2 Evts/s
MT w/ 1 thread	1.9 Evts/s
MT w/ 2 threads	3.8 Evts/s

No real numbers, just illustrative

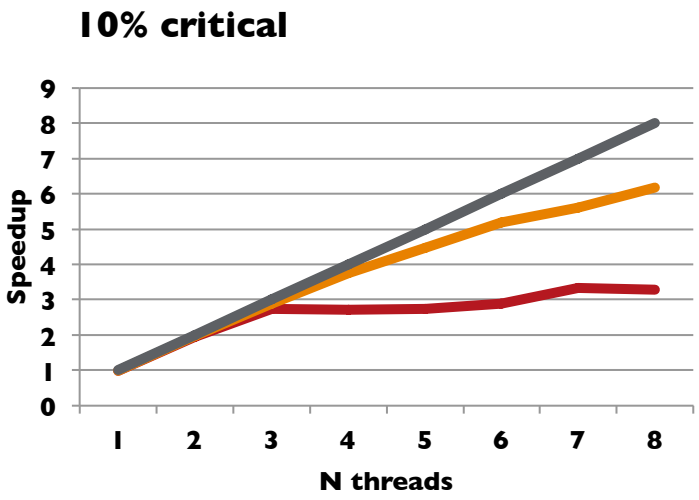
Basic design choice

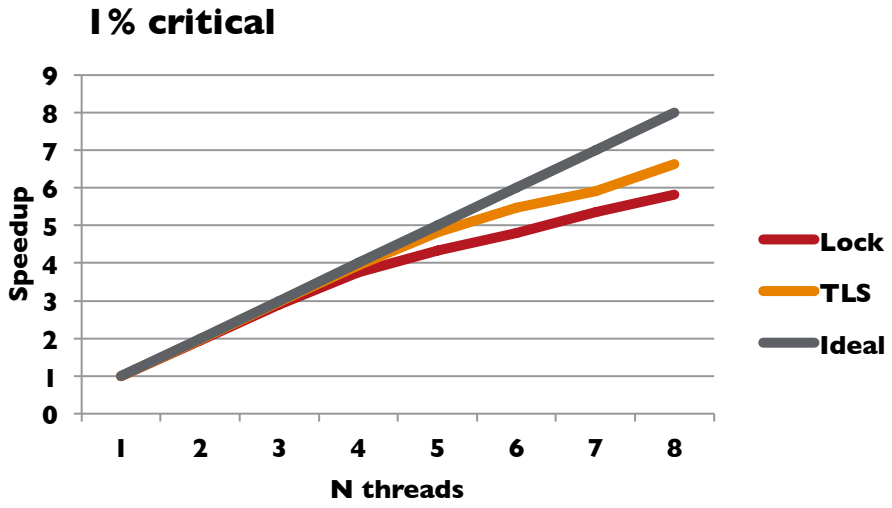
SLAC

- 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, Physics Tables
  - Rest is thread-private



Thread Local Storage





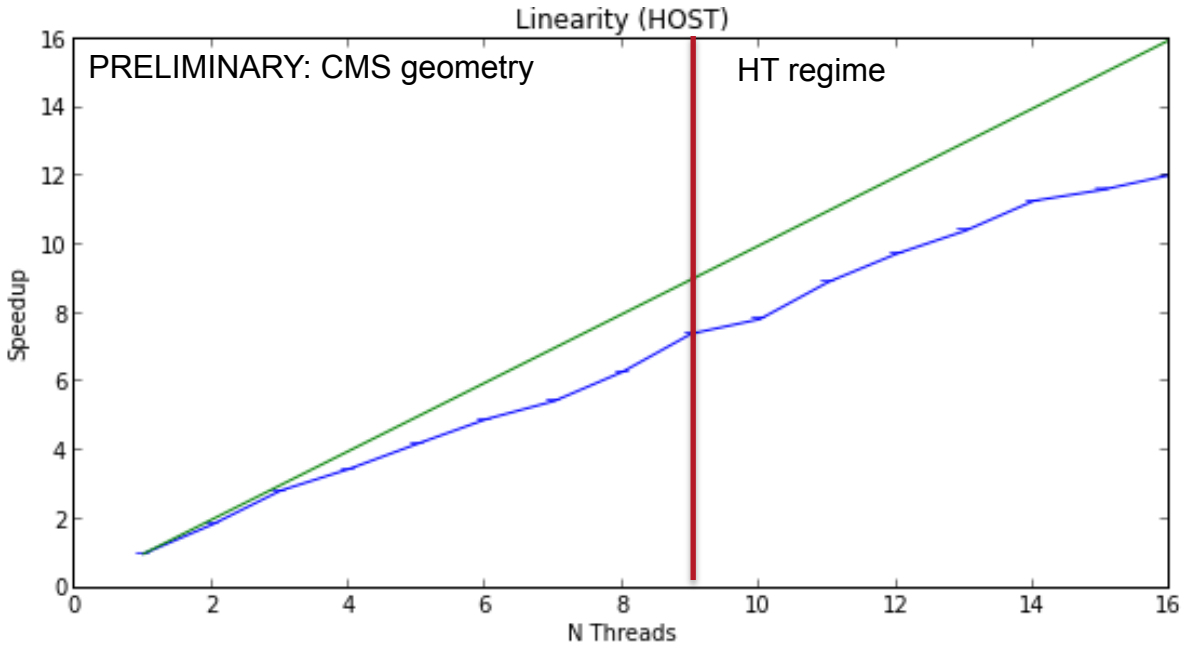
NB: results obtained on toy application, not real G4

- Each (parallel) program has sequential components
- **Protect access to concurrent resources**
- Simplest solution: use mutex/lock
- TLS: each thread has its own object (no need to lock)
  - **Supported by all modern compilers**
  - "just" add `__thread` to variables

```
__thread int value = 1;
```

  - Improved support in C++11 standard
- Drawback: increased memory usage and small cpu penalty, only simple data types of data of static/ global variables can be made TLS

Results: x86_64



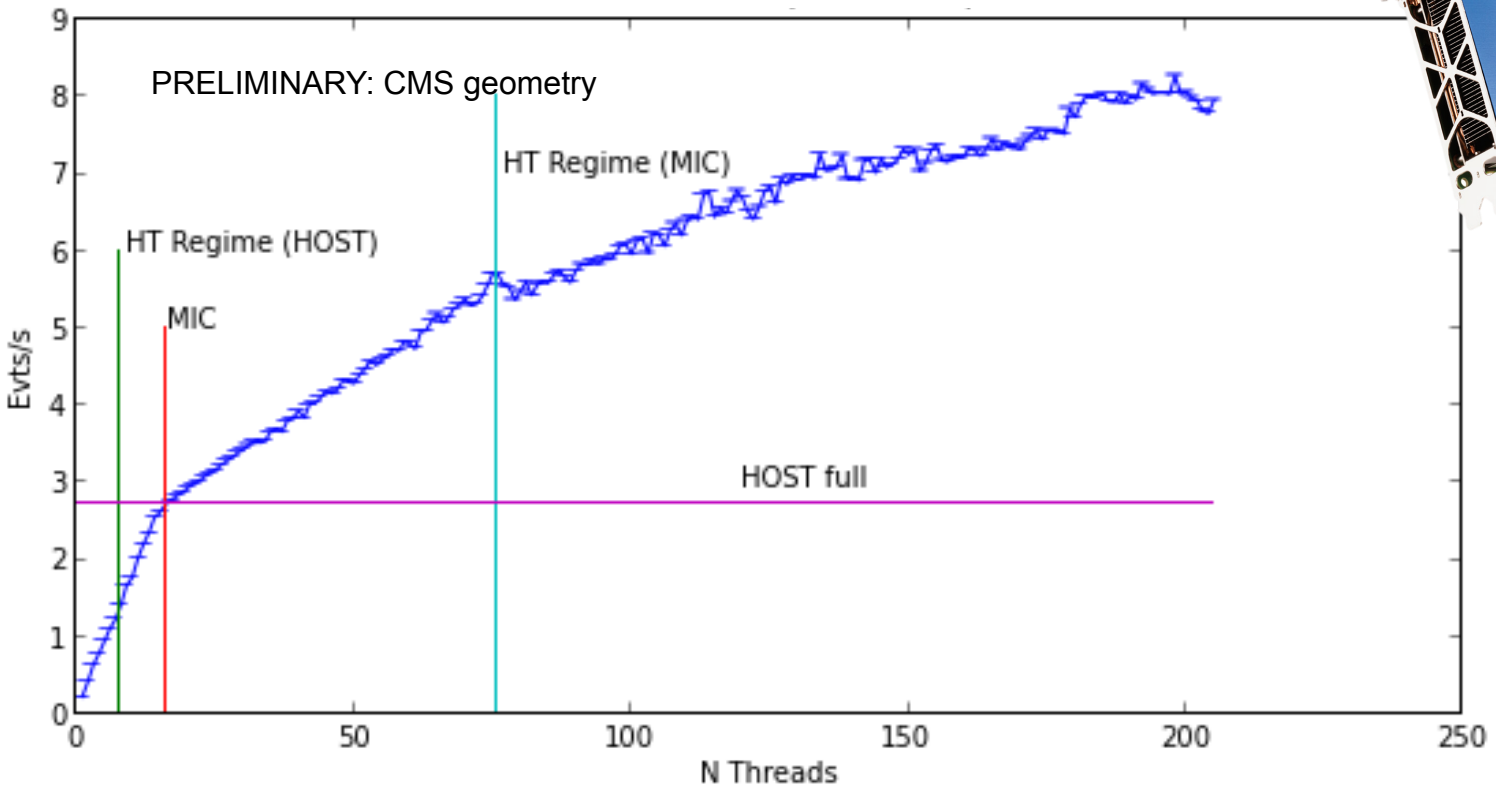
Intel(R) Xeon(R) CPU L5520 @ 2.27GHz

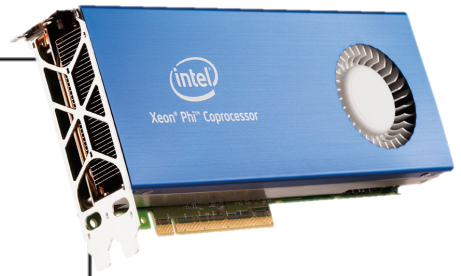
- **Good linearity demonstrated**
- Efficiency w.r.t. perfect linearity 90% (80% in HT)
- **Out-of-the box** Geant4 with MT=ON
- Further improvements expected
  - Use of thread-private malloc library
  - Reduce use of TLS when not necessary
  - See *Euro-Par2010, Part II LNCS6272, pp.287-303*: full efficiency recovered

Results: large number of threads (MIC architecture)

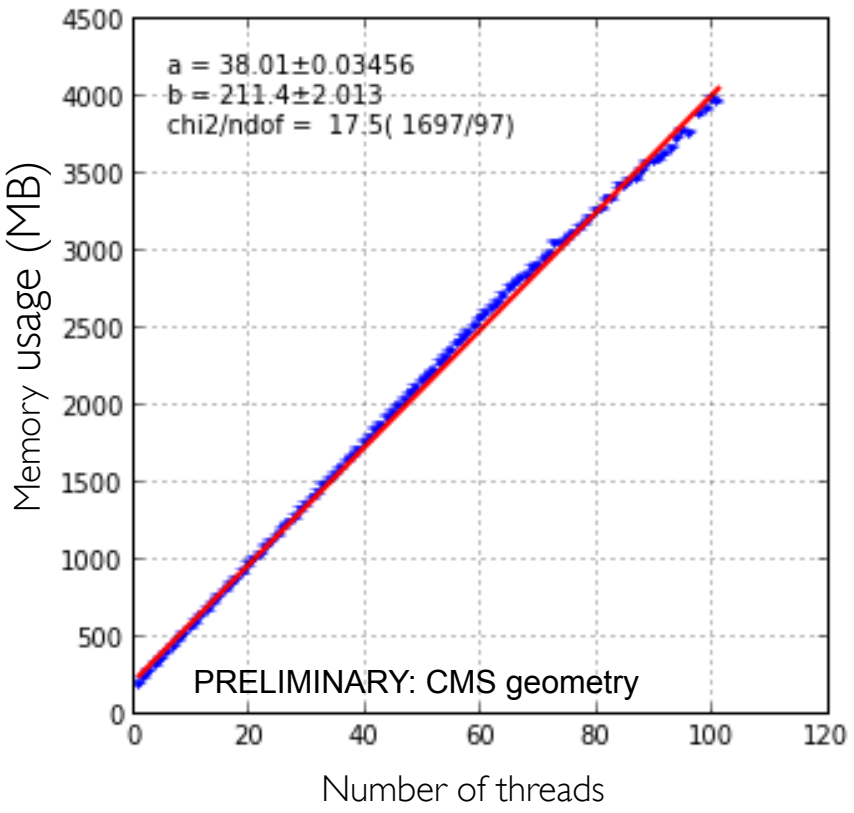
SLAC

- **Hybrid mode: Host + Intel Xeon Phi coprocessor**
- First look at total throughput: Evts/s
  - **Very good results: factor ~x3 in events produced w.r.t. host only**
- Up to 8 MIC cards can be hosted by single host
- Need to coordinate processes (e.g. MPI, intel-MIC-offload)
- Very different initialization time between host and MIC (see later checkpointing)





Results: memory consumption



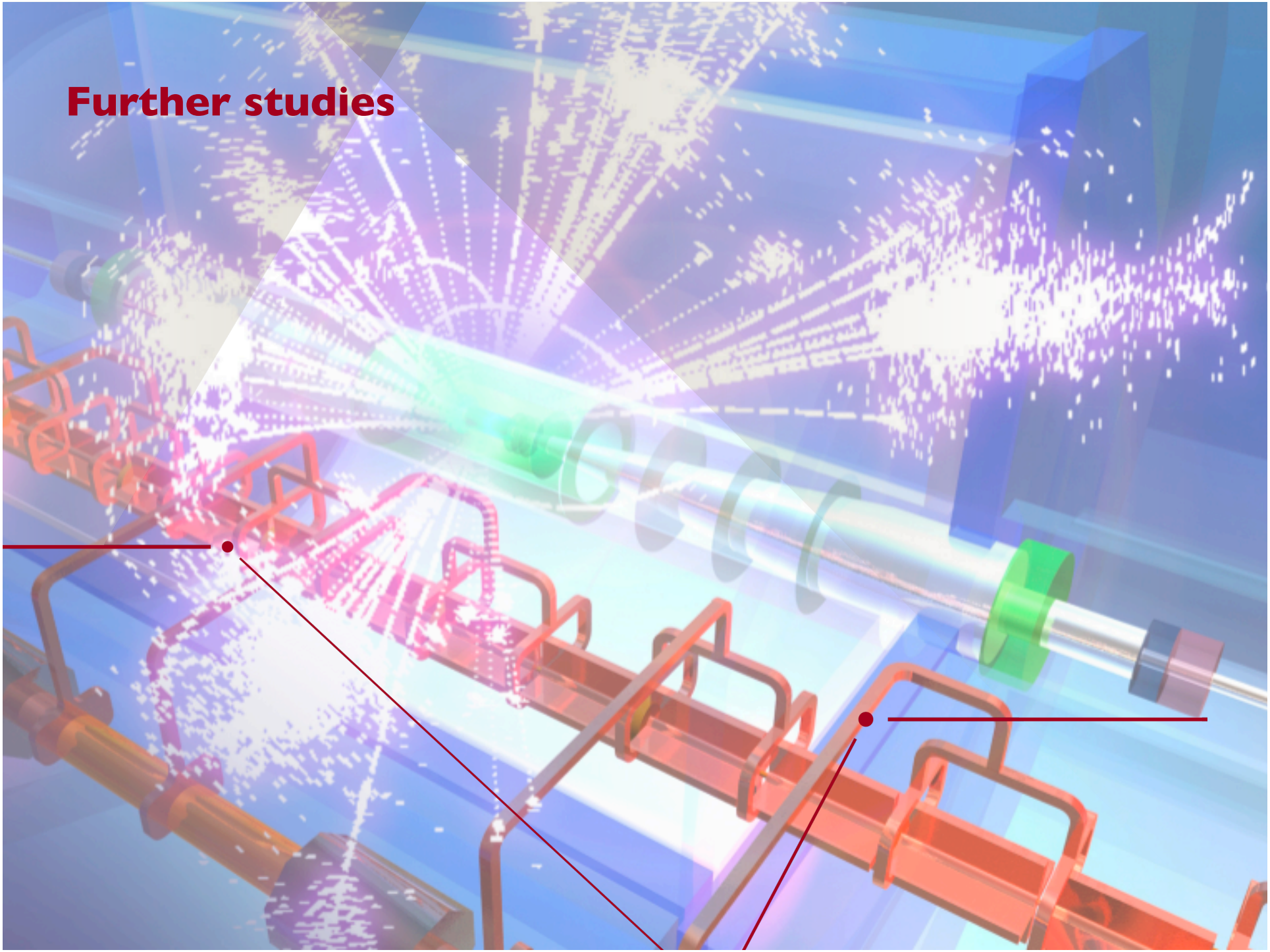
- No optimizations done yet
- 40MB /thread
- Shows benefit of G4MT design: **speedup (almost) linear with small increase of memory usage**
- Baseline: 200MB (geometry, physics)

Status of Geant4 Version 10.0

SLAC

- **Beta version will be announced at the end of this week**
- All main functionalities have been ported to MT
  - **Only two limitations:** radioactive-decay or ion beams not yet ported (need further work on ion / isomers / decaying nuclei treatment). Visualization is not yet fully functional (no event-by-event visualization during the event loop)
- **Migration of user code is relatively simple:** existing examples and tests can be migrated in few hours, complex user applications will require more work
  - Particular attention should be put in the analysis (hits collection and processing): in particular some external analysis tools are not (fully) thread safe

Further studies



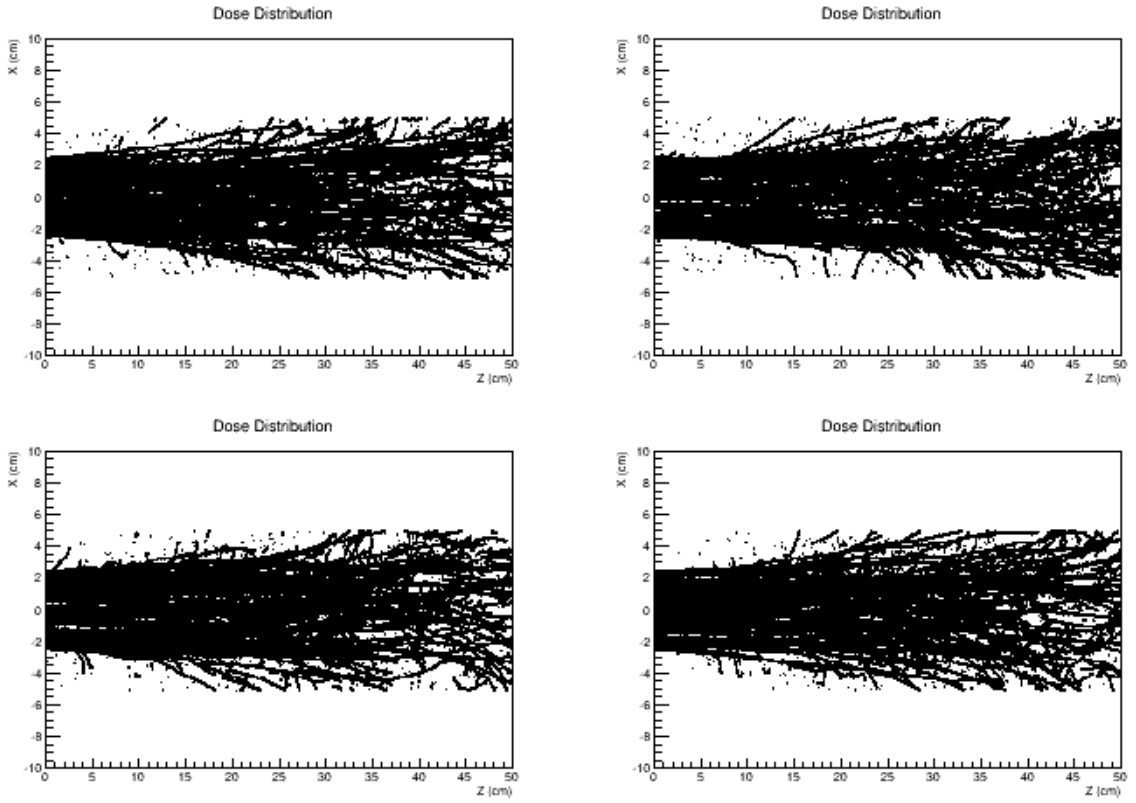
Heterogeneous parallelism: MPI based G4MT

SLAC

- **MPI based parallelism** already available in Geant4
  - MPI works together with MT

Example:
4 MPI jobs
2 threads/job
MPI job owns histogram

Next Step:
Host + MIC simulation
Based on MPI



Improve start-up time: Checkpointing

SLAC

- Each parallel application has sequential part (G4MT: geometry definition, physics initialization, threads creation and initialization)
- With large number of threads **sequential CPU-time** can become important fraction of total run-time (especially true for accelerators)
- **Substantially reduced via checkpoints** (dump of program image to disk at specific points. A controlled "core-dump"), **restart from checkpoint image**
- **DMTCP**: checkpointing for multi-threaded programs (G. Cooperman et al)
- Tested with success for Geant4 MT (in collaboration with CMS experiment). On Xeon Phi:
  - Start CMS simulation, checkpoint at first event (5 mins initialization)
  - Replay/restart application from checkpoint file (10 seconds restart)
  - Interesting possibility for production system: copy checkpoint image on many machines (or accelerators), start clones of process, re-seed processes

Work in progress

SLAC

- **Intel Thread Building Block (TBB)**: task based parallelism framework (expression of interest by some LHC experiment)
  - TBB works with G4MT: provide one or two examples for final release
- **ThreadPrivate malloc library** (TPMalloc – G. Cooperman et al): each thread has its own heap, remove hidden locks in "new/delete". Target to be provided as "external optional component".
- Review APIs with feedback from early users: **further simplify user-code migration**
- Identify and solve hotspots, **improve performances**
- Fully functional Visualization drivers

Future directions

"The only thing we know about the future is that it will be different."
Peter Drucker

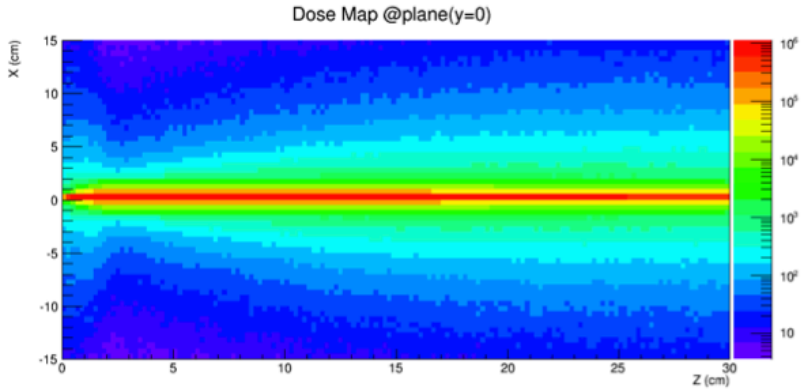
Accelerators

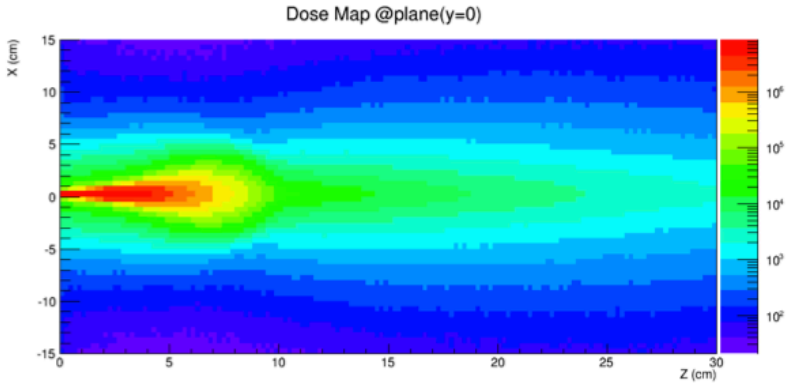
SLAC

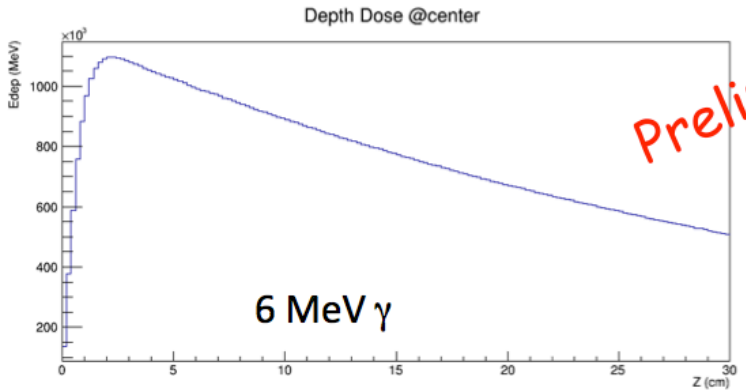
- **Increasing interest in accelerators**: two main technologies GPGPU / MIC architectures
  - First two Top500 supercomputers based on accelerators (#1: Thianhe-2, Intel Xeon Phi ; #2: Titan, Nvidia K20)
- In some cases (GPU) **rewrite completely code** in technology specific language
- GPGPUs are particularly tailored to specific problems: **very high performances** can be reached in specific domain
- Intel Xeon Phi advantage: **no-need to rewrite code**, optimizations done for MIC architecture are valid for host CPU

GPU

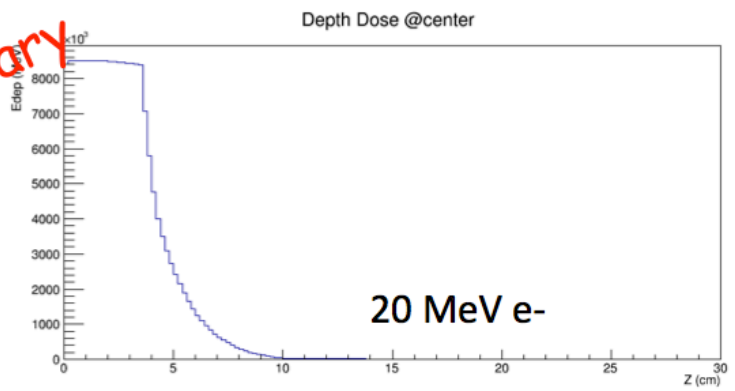
- SLAC/Stanford-ICME/KEK/NVIDIA project
- Full EM physics <100 MeV electrons/gammas
  - Only one kind of material (water) with varying density (medical DICOM)
  - No geometrical navigation (only voxelization)
- Benchamark on TESLA K20: O(100) faster than CPU G4 job (full normal navigation, full physics)
- Very promising for domain-specific applications







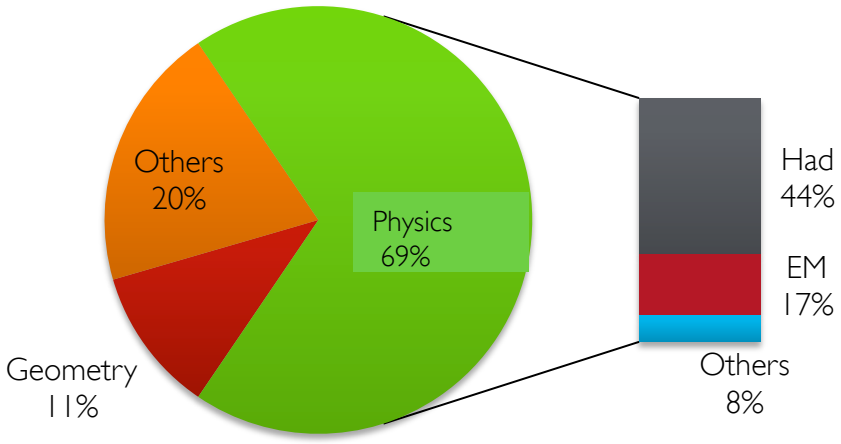
Preliminary



Beyond event-level parallelism

SLAC

- To fully use new architecture potentials we need to **investigate further level of parallelism**
- SIMD (a.k.a. vectorization): very challenging (very limited success for HEP sw). Two options available:
  1. Rewrite code with intrinsics or low-level constructors (bad idea: not portable, very complex for large sw as G4)
  2. Use compilers **auto-vectorization** features together with high-level construct
- Compiler auto-vectorization for G4 (out of the box): ~10k candidate loops, only about 5% actually auto-vectorized



Possible strategy:
Map candidate loops with most time
Consuming routines, iteratively try out auto-vectorization

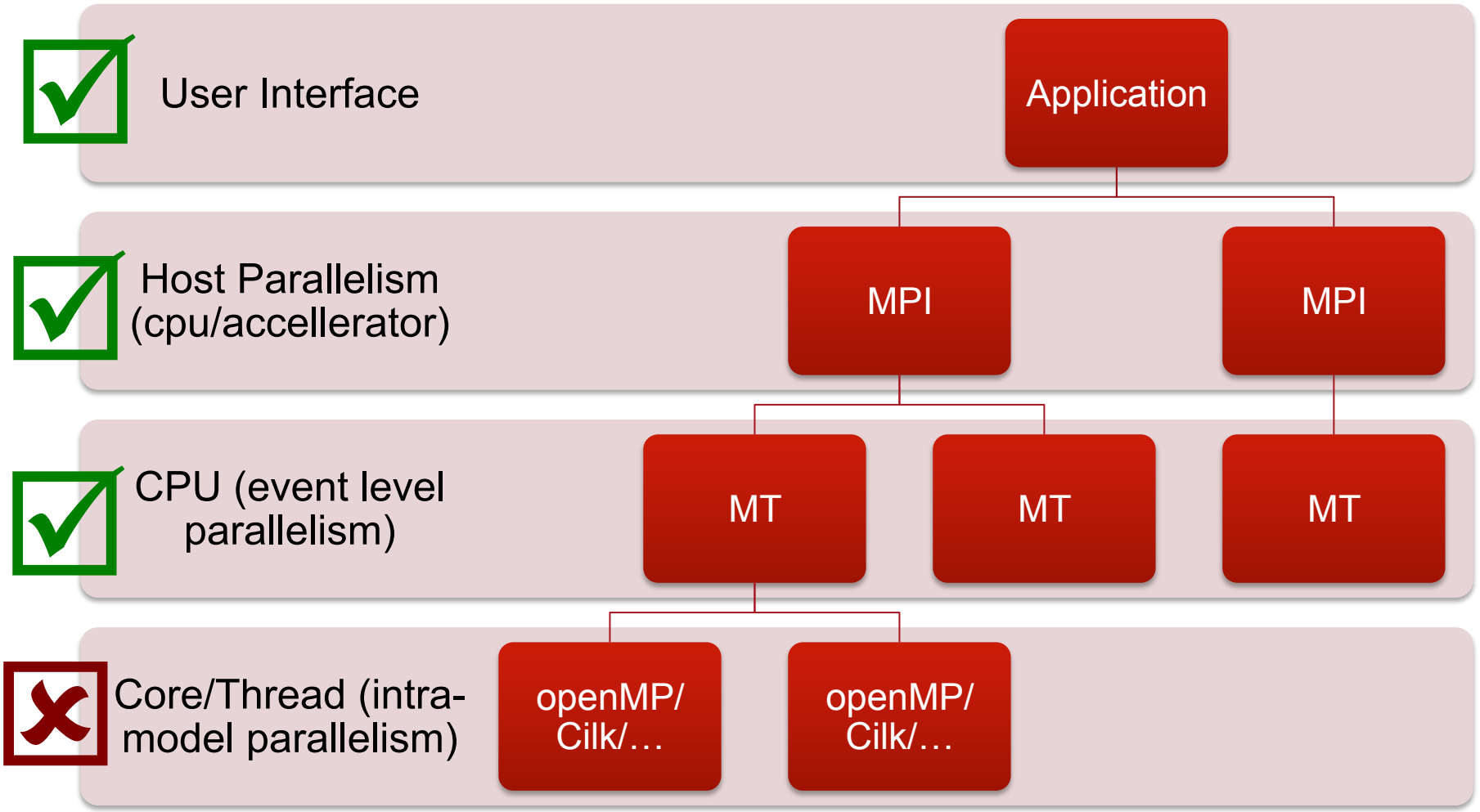
My personal view

SLAC

- Trivial sub-event level parallelism (process tracks in parallel) **does not scale well:** each thread needs thread-private memory (or lock mechanism) in addition to the overhead for book-keeping of "split" event.
- **Use of high-level parallelization** constructs to put parallelization in modules/algorithms for example:
  1. **openMP** : de-facto standard, supports for Intel accelerators
  2. **Intel CilkPlus** : very simple and powerful (function vectorization via #pragmas), GCC support only in branch (integration with TBB)
  3. **openACC** : relatively new open standard aimed at developing directives for accelerators (technology independent)
- Need to experiment with all technologies and understand benefits / challenges
- Possible path: identify one or two of the most time consuming elements in simulations (e.g. cross-section calculations , Bertini intra-nuclear cascade) and try to apply sub-event parallelism

A roadmap: my view

SLAC



Conclusions

SLAC

- Geant4 Version 10.0 well on track for end of year release
- Major developments for event-level parallelism
  - **Very promising results obtained on both "traditional" CPUs and MIC architectures**
  - Expect further improvements
- Possible to integrate G4MT with additional high-level parallelization frameworks (TBB, MPI)
- **Scalability demonstrated up to O(100) threads**
  - Ready for future challenges of current and next generation large simulation campaigns (i.e. LHC –scale)
  - New possibilities for "smaller" simulation needs (efficient use of many core machines, accelerators on desktops)
- **Multi-threading and thread safety is the first indispensable step towards further review of Geant4 code**

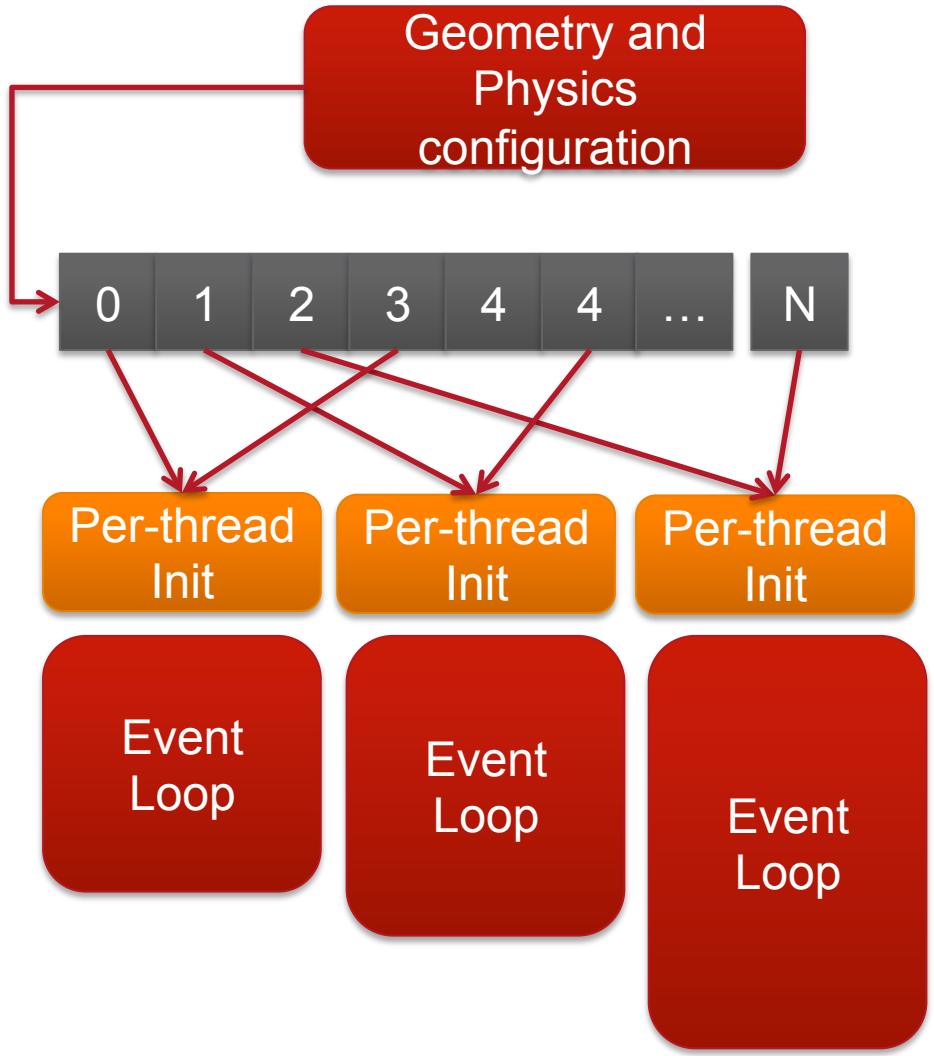
Acknowledgments

I would like to thank few people that helped to produce the material here presented and gave important contributions during discussions:

- K. Murakami (KEK), N. Henderson (SU), A. Vladimirov (SU), G. Cooperman (NortheasternU), X. Dong (NortheasternU), G. Cosmo (CERN), P. Elmer (PrincetonU)

Backup

Random Seeds



- To gurantee reproducibility
- Each thread has its own RNG
- Master thread pre-generates per-event seed
- Each event is re-seeded
- Further refinement on RNG to be studied (pRNG)