http://www.geant4.org

Multi-threaded Geant4 on Intel Many Integrated Core architectures

CHEP2016 – San Francisco, 8-14 October 2016

Andrea Dotti (adotti@slac.stanford.edu) ; SLAC/SD/EPP/Computing

Makoto Asai (SLAC), Steven A. Farrell (LBNL)

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY
BERKELEY LAB

Outlook

SLAC

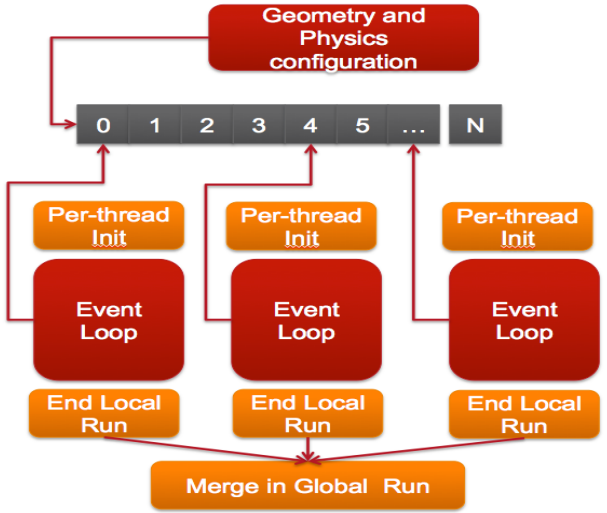
Introduction: parallelism in Geant4

Results: memory usage, scalability, KNC vs KNL

Scaling at very large systems: integration w/ MPI, Geant4 at extreme scales

Recent Geant4 Improvements

Parallelism in Geant4: Master/Worker model



Since Geant4 Version 10.0 (December 2013): introduced event level parallelism.

Driving forces:

- maintain and improve physics quality
- **minimize user code changes**
- keep approach simple
  (physicists ≠ computing scientists)

Design/prototyping phases started in 2010-2012

Not an endpoint but an important milestone for the future Geant4

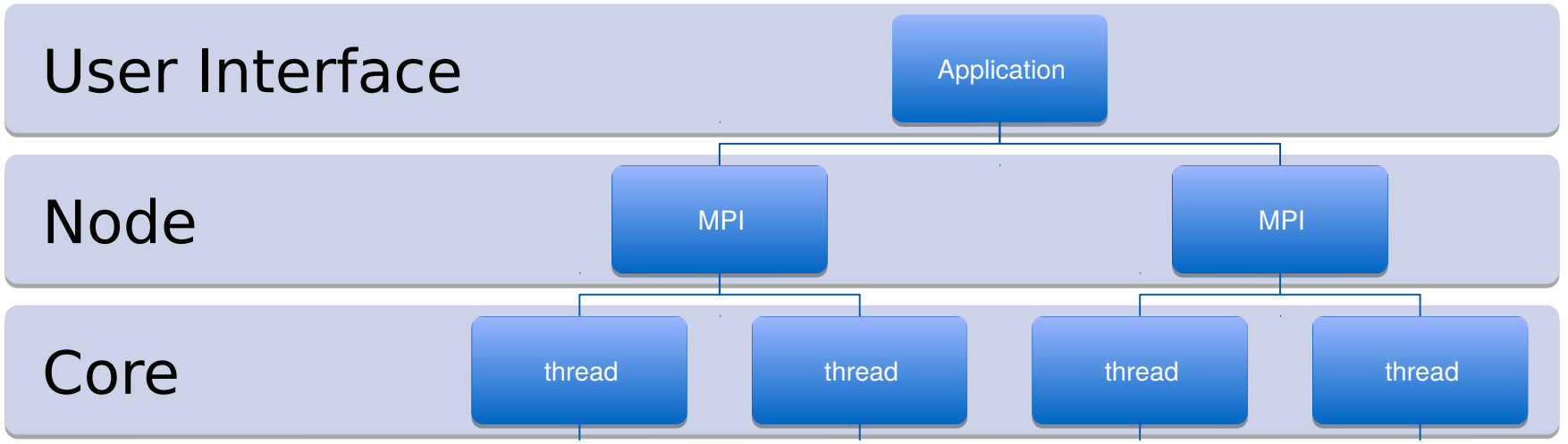
Expecting more features released in the next years

Version 10.0 (Dec. 2013)
- Implement correct MT behavior (remove race conditions)
- Memory reduction from geometry and physics

Version 10.1 (Dec. 2014)
- Improve migration some components (GPS, RDM, Vis)
- Obtain further x2 memory reduction

Version 10.2 (Dec. 2015)
- Finalize VIS module
- Simplify integration of G4 with MPI and TBB

Geant4 Strategy for parallelism



We provide defaults for all level of parallelism, users can overwrite with experiment framework specific technologies

E.g. LHC experiments: GRID instead of MPI, TBB instead of pthread

Geant4 Strategy for parallelism

SLAC

Our new general paper

J. Allison et al. “Recent Developments in Geant4”

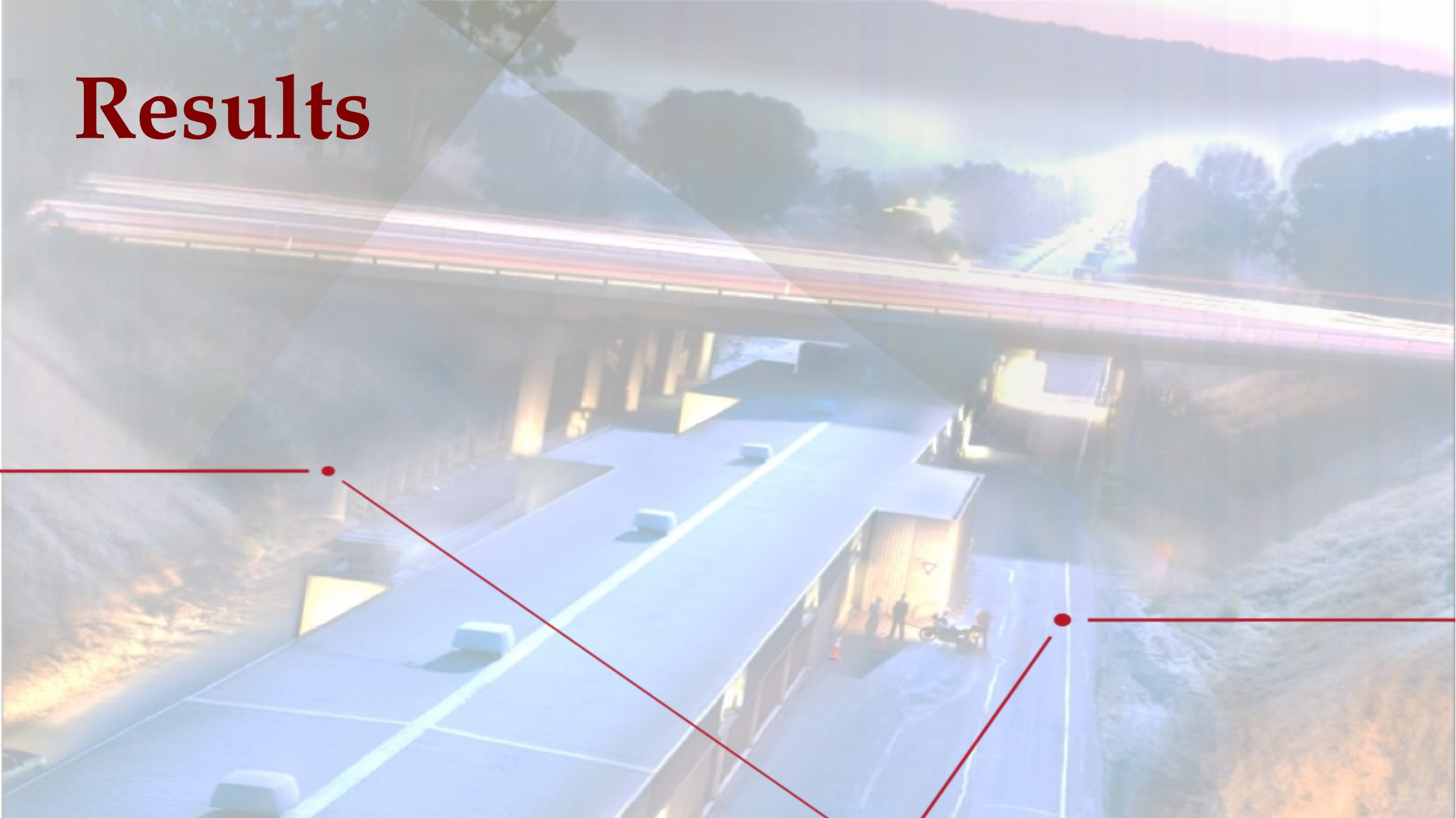
NIM A 835, Vol 835, 1-Nov.2016, pp. 186-225

Also includes details on Geant4 multi-threading capabilities

We provide defaults for all level of parallelism, users can overwrite with experiment framework specific technologies

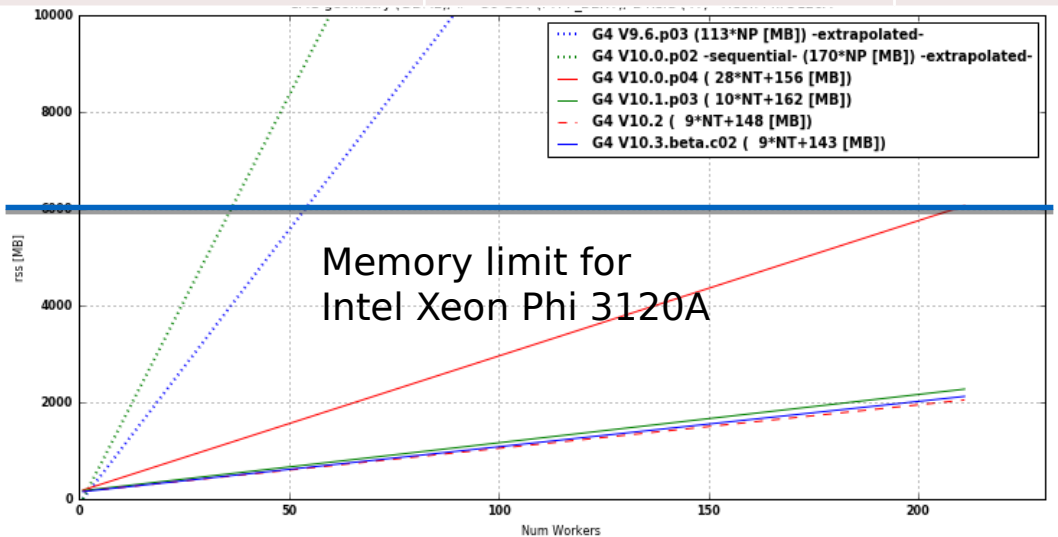
E.g. LHC experiments: GRID instead of MPI, TBB instead of pthread

Results



Memory reduction

Version	Initial memory	Memory/thread
9.6 (no MT)	113 MB	(113 MB)
10.0.p02 (no MT)	170 MB	(170 MB)
10.0.p02	151 MB	28 MB
10.3.beta	148 MB	9 MB



Geant4 MT design principle: share between threads read-only data (geometry, physics tables): **lock-free event loop**

Goal: **substantially reduce memory usage w.r.t. pure multi-process application (e.g. MPI)**

Recent campaign to reduce more than a factor 2 memory use in MT mode

Recent feedback from CMS: full CMSSW sw stack of ttbar events: ~200MB/thread

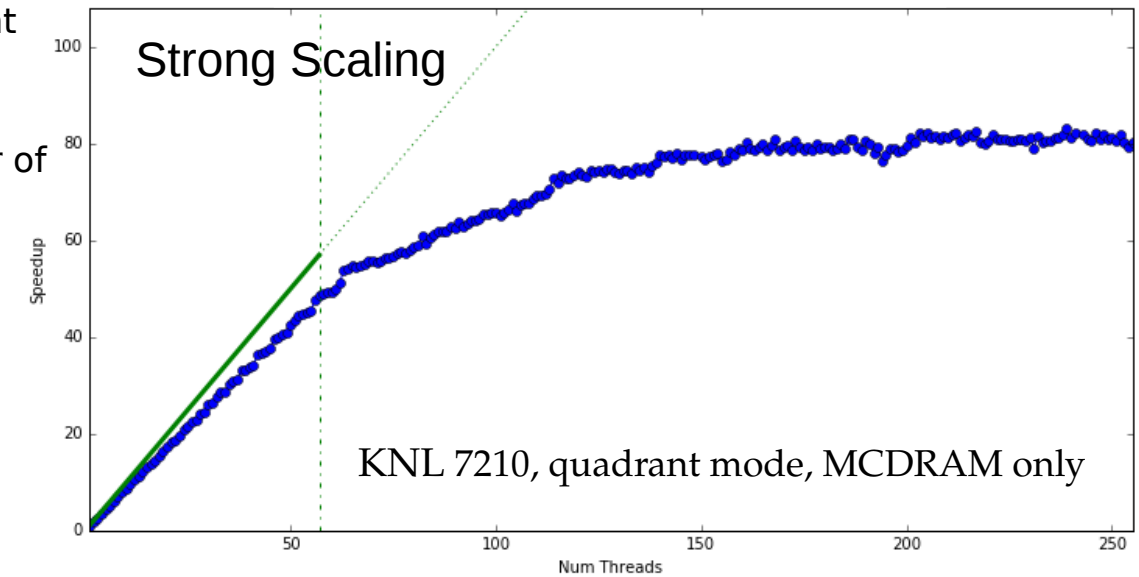
Includes all user-code

Needs KNL for moderate/large number of threads

HepExpMT benchamrk: Simplified CMS geometry (via GDML), uniform B-Field, 50 GeV π- w/ FTFP_BERT

Linearity speedup

- Number of events/second is the most important metric for users
- **Very good linearity** (>93%) with the number of physical cores available
- Benefits from hyper-threading: ~30%
- Verified for different types of applications:
  - Medical physics applications
  - HEP experiments



HepExpMT benchmark: Simplified CMS geometry (via GDML), uniform B-Field, 50 GeV π- w/ FTFP_BERT

Access to KNL processor provided by Colfax International

KNL vs KNC

- We provide support for running G4 on KNC, https://goo.gl/qEFo6u , will update for KNL
- Due to x86 binary compatibility, work-flow is tremendously simplified
  - remember no explicit vectorization in Geant4!

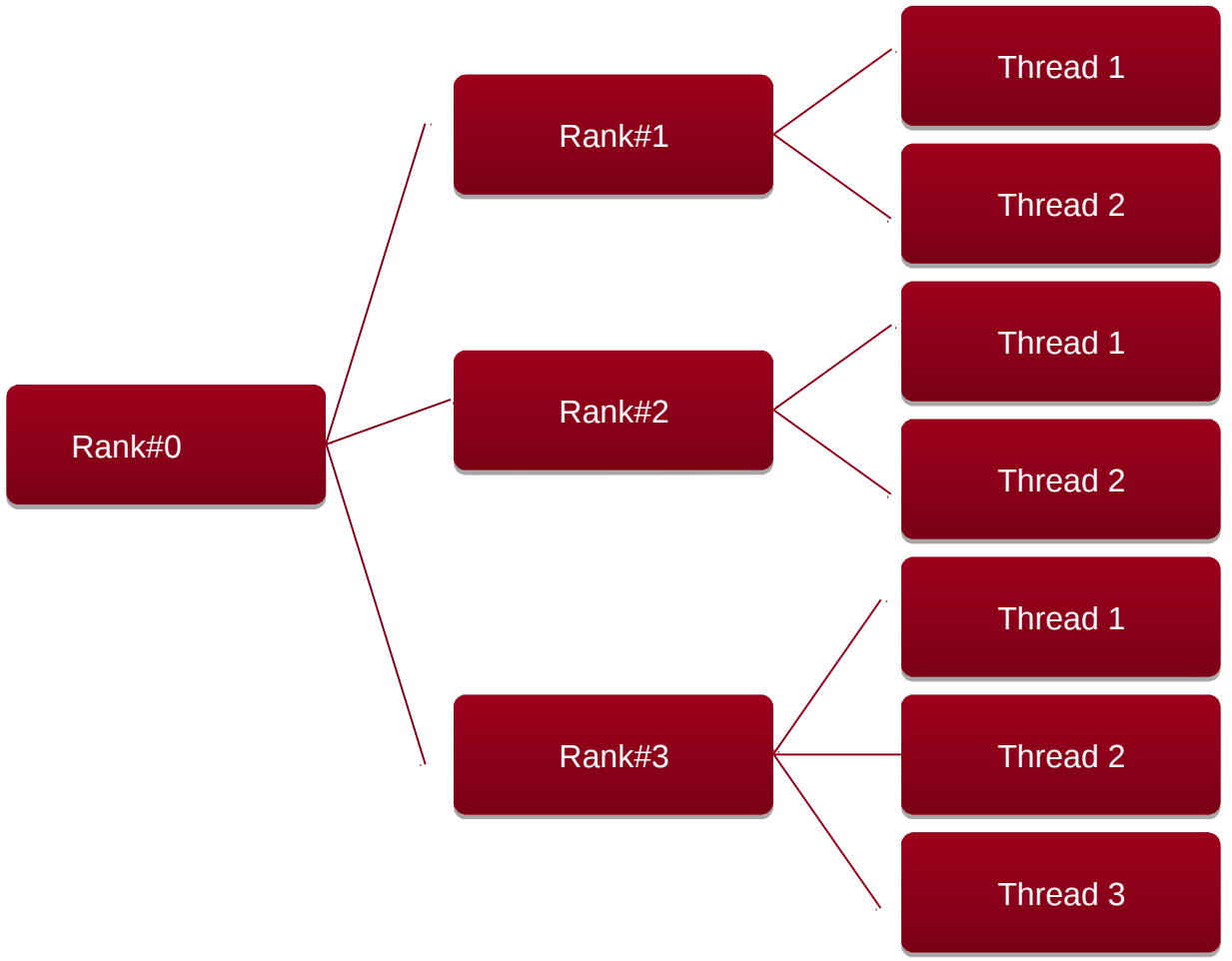
System	Time to completion (5k events)
Xeon E5-2620 @ 2.1 GHz (12x2 cores)	570 s
KNC (31s1P) @ 1.0 GHz (228 threads)	1000 s
KNL (7210, quadrant mode, MCDRAM only) @ 1.3 GHz (255 threads)	378 s (x3 improvement w.r.t. KNC)
KNL (shared library)	480 s (25% slower)

Single core KNL slowdown w.r.t. host: x4

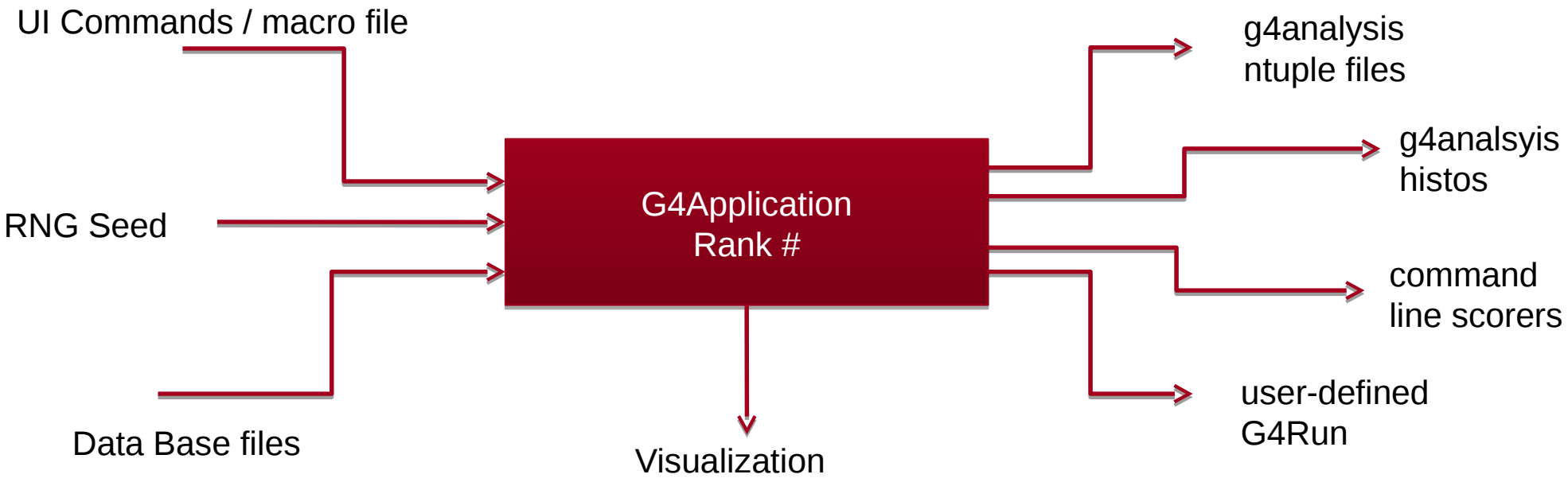
Scaling w/ MPI

MPI and Geant4

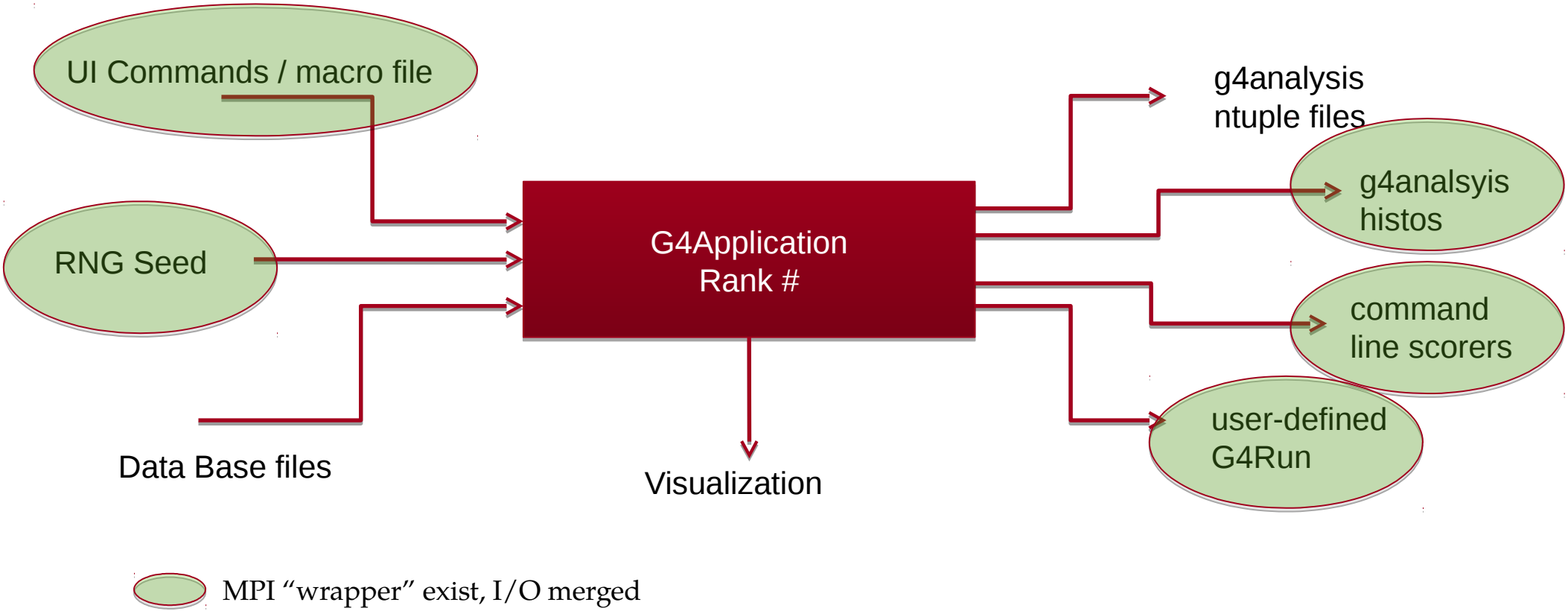
Rank#0 broadcasts UI commands and RNG seeds
Workers send back results for merging: histograms, ntuples, scorers



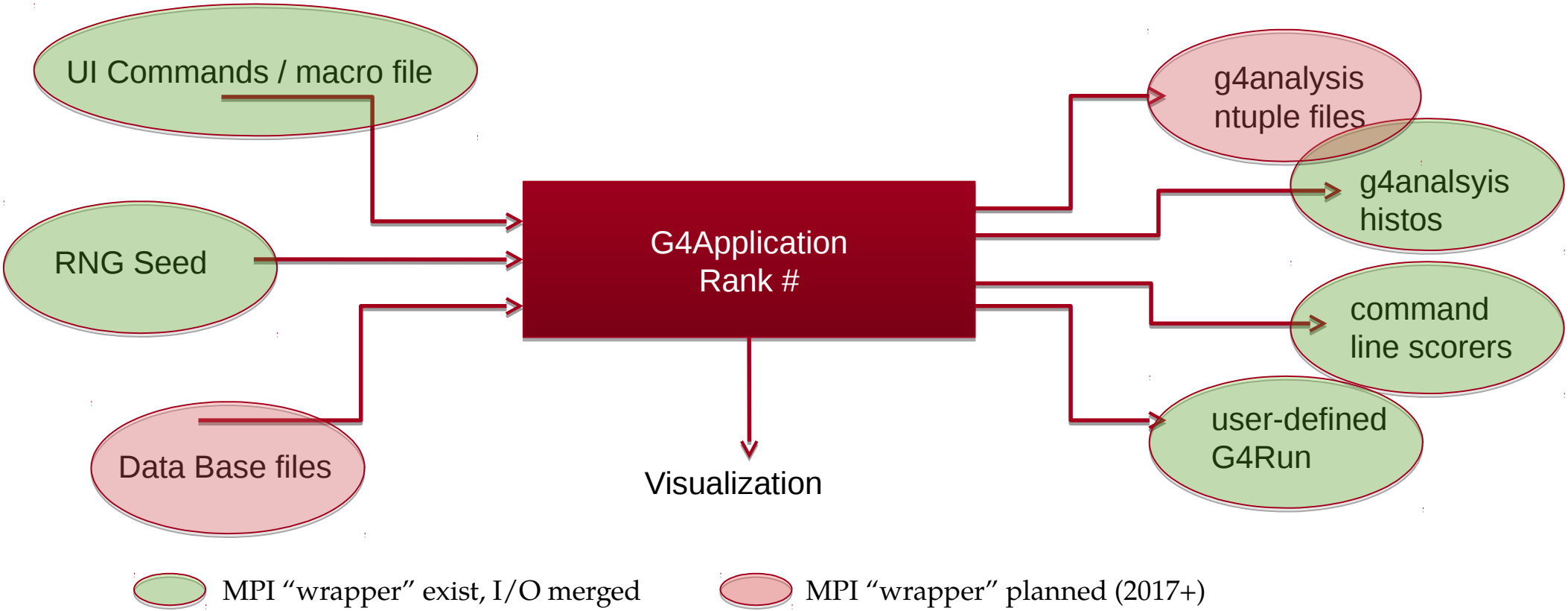
Geant4 applications from MPI point of view



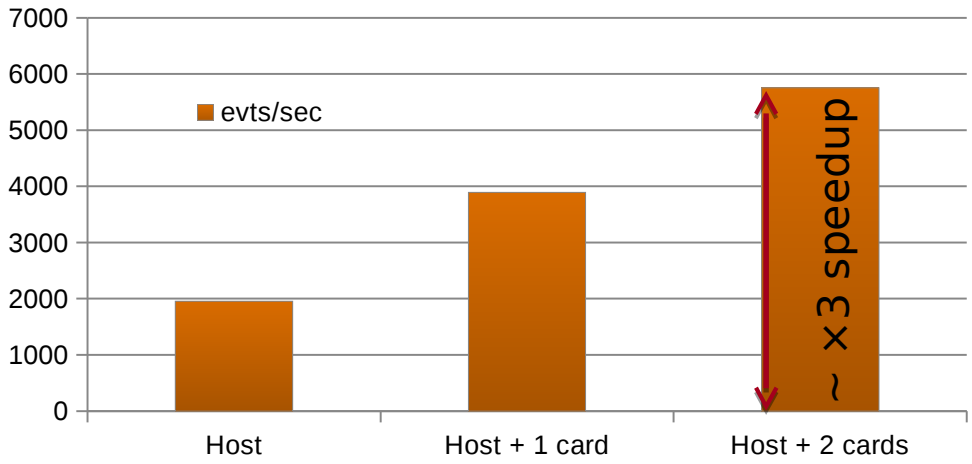
Geant4 applications from MPI point of view



Geant4 applications from MPI point of view



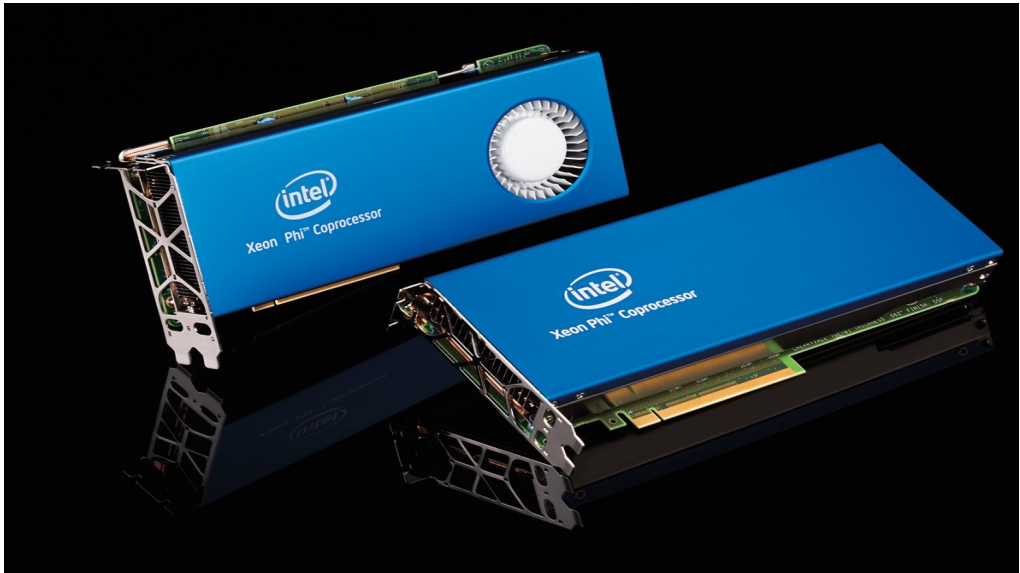
Scaling to multiple systems



System:

Intel E5-2600 @ 2.2GHz (8C/16T)

2 Xeon Phi cards model 3120A (57C/228T)



MPI application started on host and on two MICs: a small cluster in your desktop

"Medical" benchmark: proton 200 MeV on water phantom

Preparing for Next Generation SC

SLAC

KNL target systems: Theta@ANL, Cory@NERSC

Currently:

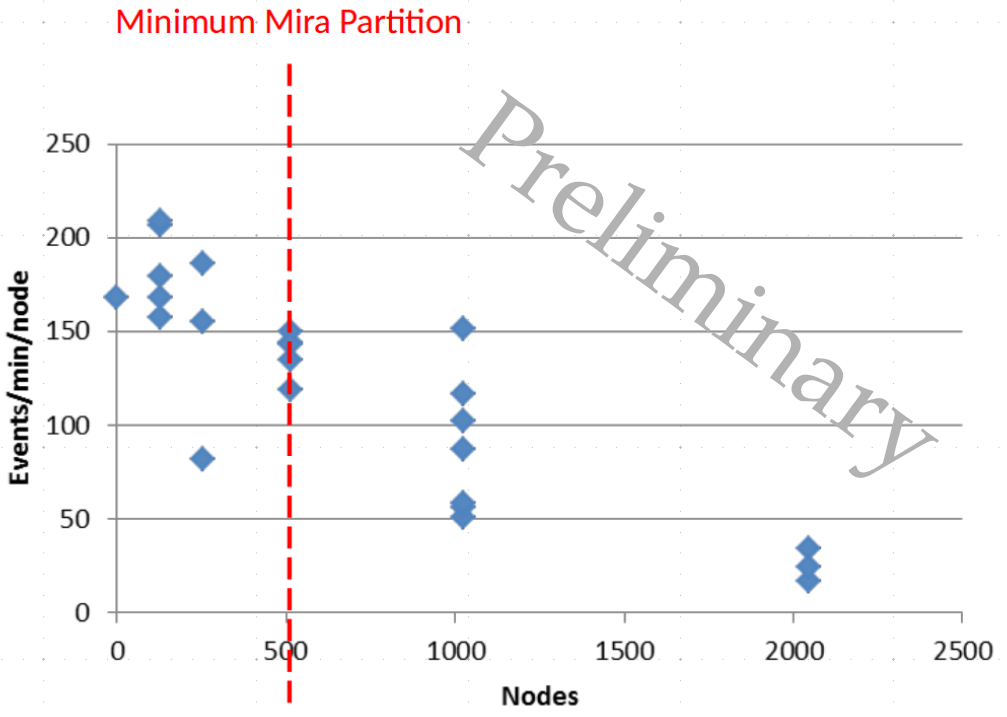
- Testing Geant4 on single KNL systems
- Test Geant4 on very large partitions on existing systems

Testing Geant4 at Mira@ANL (BlueGene/Q) with up to ~¼ million threads

Scaling up to 64k threads, above that hit scaling limit

- We believe this is due to limitations in I/O, need to aggregate access to disk, cout/cerr
- In the work plan for next years

We are on the good path to scale to O(10^6) threads with simple applications if we manage to address I/O



Courtesy of T. LeCompte ALCF (at ANL)

1 node = 16 BlueGene/Q cores @ 1.6 GHz

Preparing for Next Generation SC

SLAC

KNL target systems: Theta@ANL, Cory@NERSC

Currently:

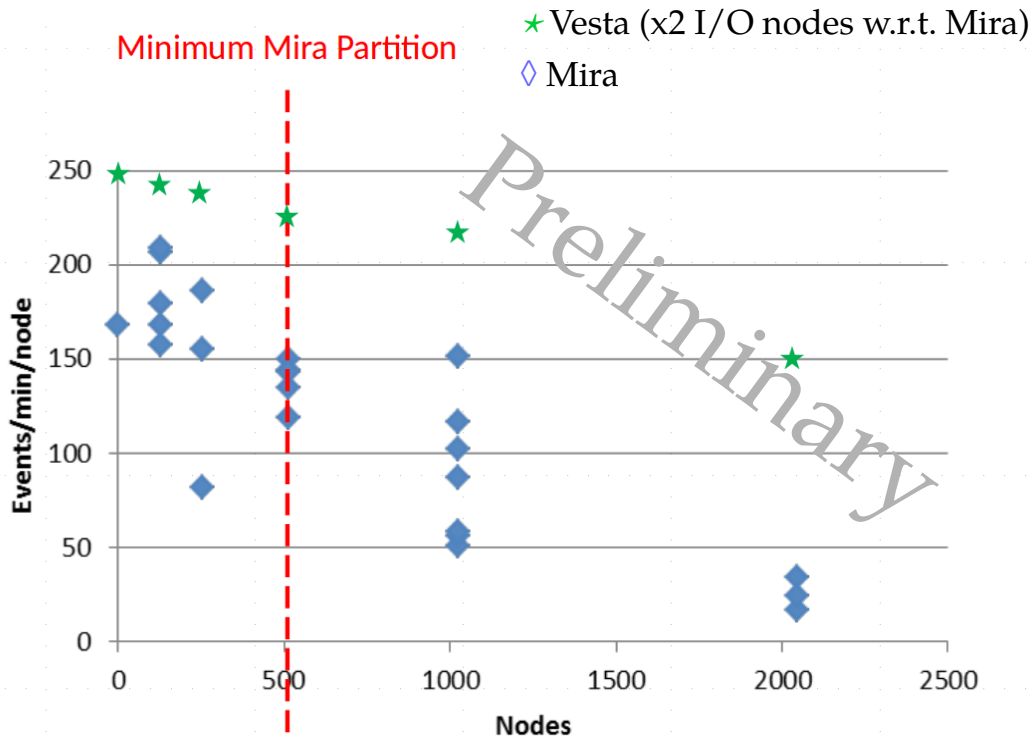
- Testing Geant4 on single KNL systems
- Test Geant4 on very large partitions on existing systems

Testing Geant4 at Mira@ANL (BlueGene/Q) with up to ~¼ million threads

Scaling up to 64k threads, above that hit scaling limit

- We believe this is due to limitations in I/O, need to aggregate access to disk, cout/cerr
- In the work plan for next years

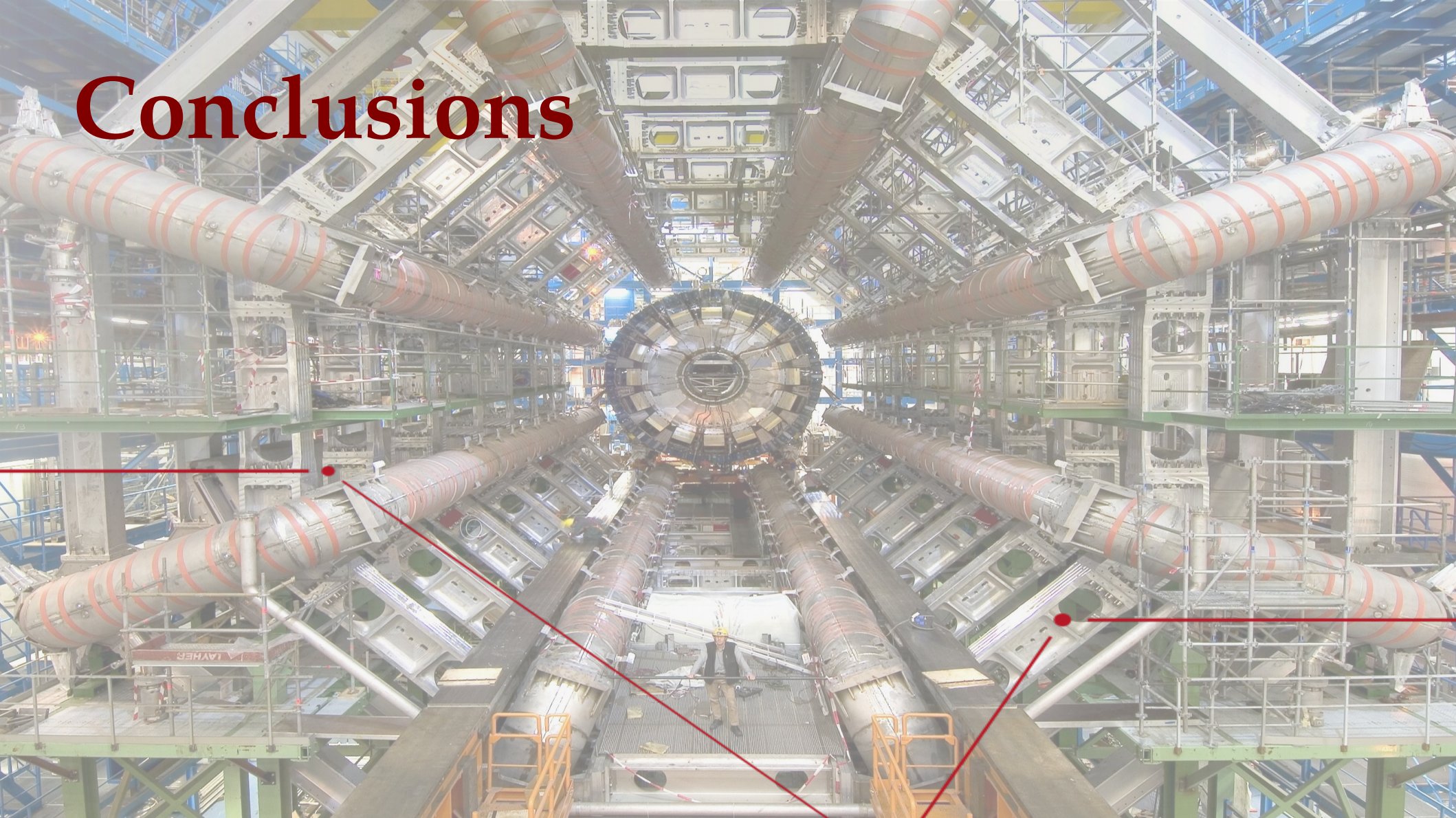
We are on the good path to scale to O(10^6) threads with simple applications if we manage to address I/O



Courtesy of T. LeCompte ALCF (at ANL)

1 node = 16 BlueGene/Q cores @ 1.6 GHz

Conclusions



Summary

SLAC

Geant4 is being run regularly on MIC systems

- Recent testing on KNL shows:
  - Extremely simplified work-flow
  - Factor 3 performance increase w.r.t. KNC

Recent developments on adding MPI support (mostly to the benefit of smaller experiments)

- Mixed MPI+MT jobs show best performances
- Good performance up to large number of total threads O(10k)
- First preliminary tests at SuperComputer scale shows that I/O becomes the dominant limiting factor at O(100k) total threads. Note that this can be only partially in Geant4 (strong interactions with framework and persistency systems)

Getting ready for next generation SuperComputers is a challenge, but Geant4 design and planned improvements should allows scaling to very large number of threads

For the longer term plans we need to improve the algorithms performances (vectorization and more important memory access):

- Use of modern sub-components is planned (e.g. VecGeom)
- Review of critical algorithms (e.g. new EM models)

HepExpMT

SLAC

Testing done with a standalone application:

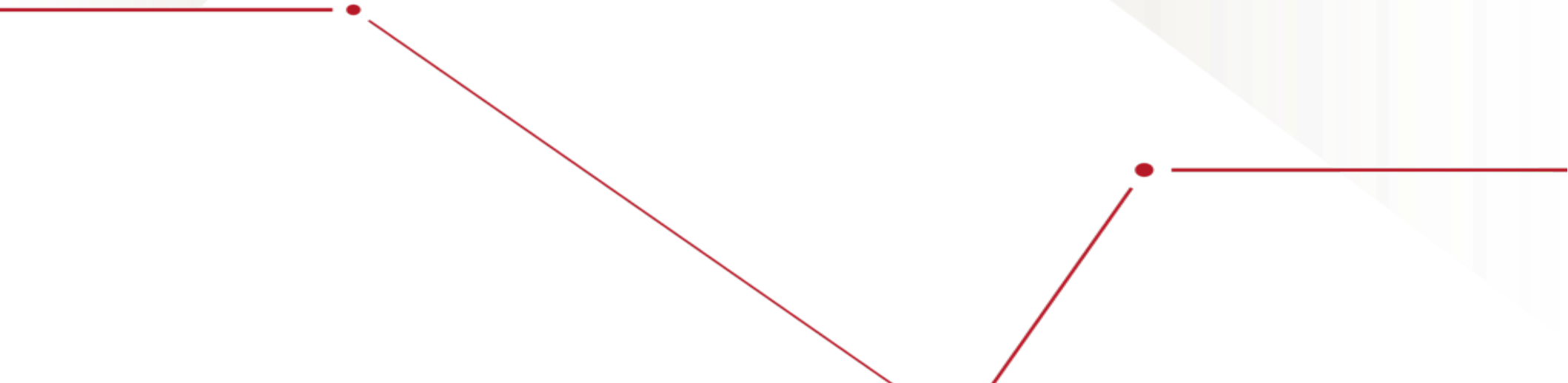
- To be used as a “public candle” for Geant4 performance measurement
- Some “advanced” features (e.g. MPI) and I/O testing

To simplify application compilation a script is provided that:

1) Downloads G4
2) Configure G4 and Application
3) Compiles G4 and Application in a coherent environment

Check it out at: https://twiki.cern.ch/twiki/bin/view/Geant4/Geant4HepExpMTBenchmark

Backup



Stating the physics problem

The study of the interaction of radiation (e.g. particles, x-rays) with matter has applications in several scientific areas:

Basic research (e.g. at accelerators to discover new phenomena)

Medical imaging (e.g. x-rays)

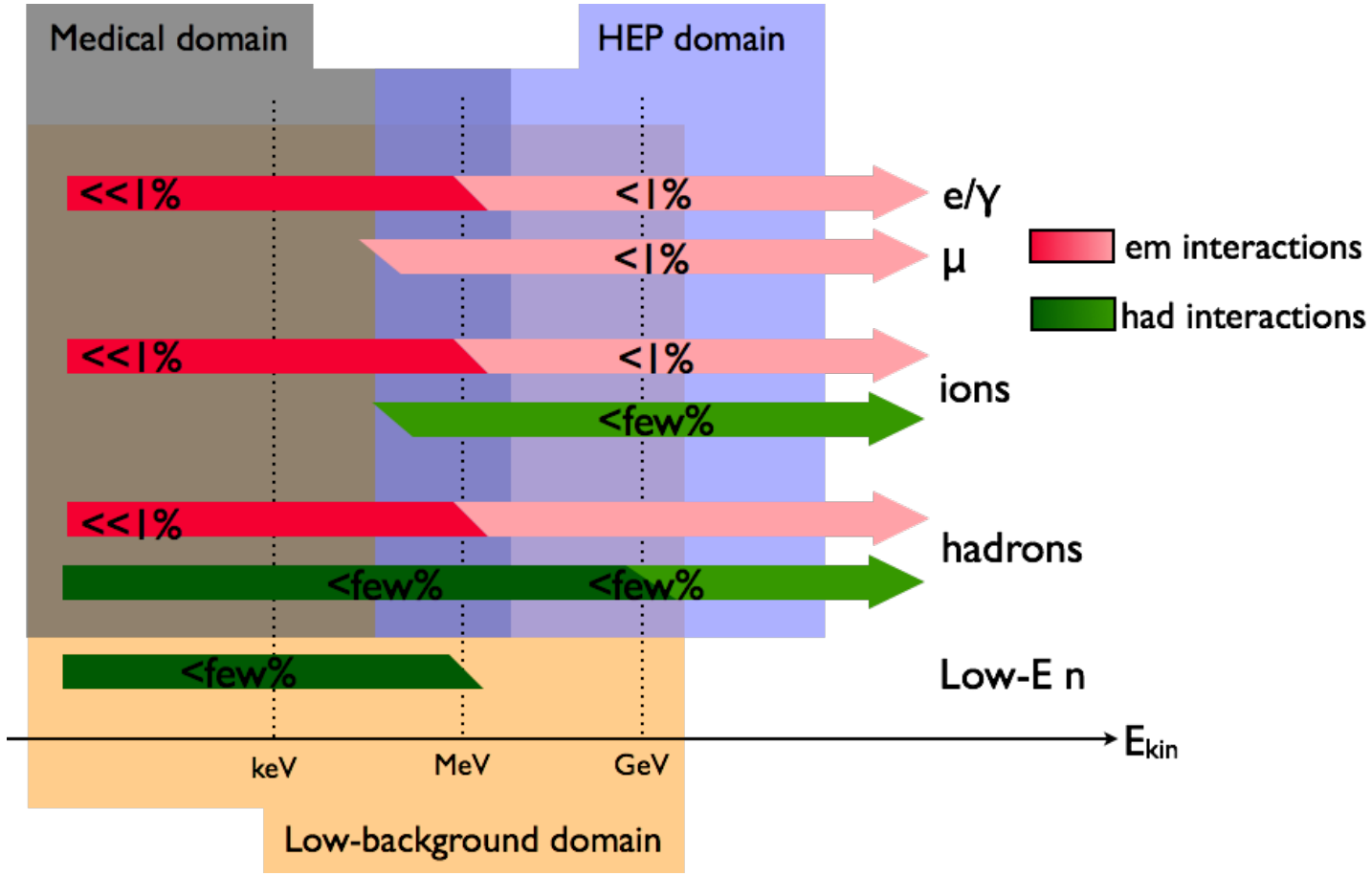
Medical treatment (e.g. radio-therapy)

Industrial (e.g. energy production, shielding)

Essential tools in these fields are simulation programs. The most precise are based on Monte Carlo techniques

Several codes exists: Geant4 is one of them, the most widely adopted

Physics Requirements



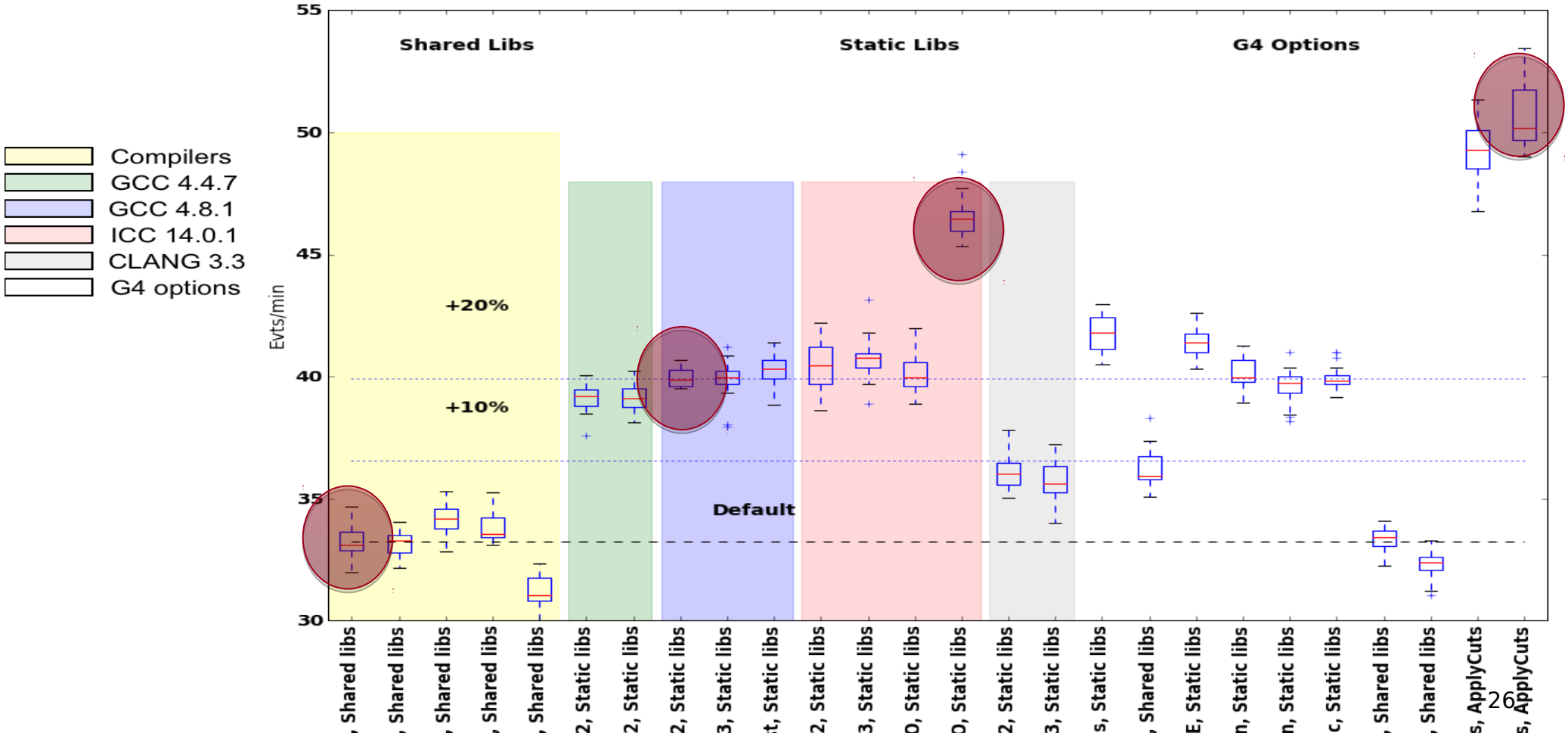
Geant 4

http://www.geant4.org/

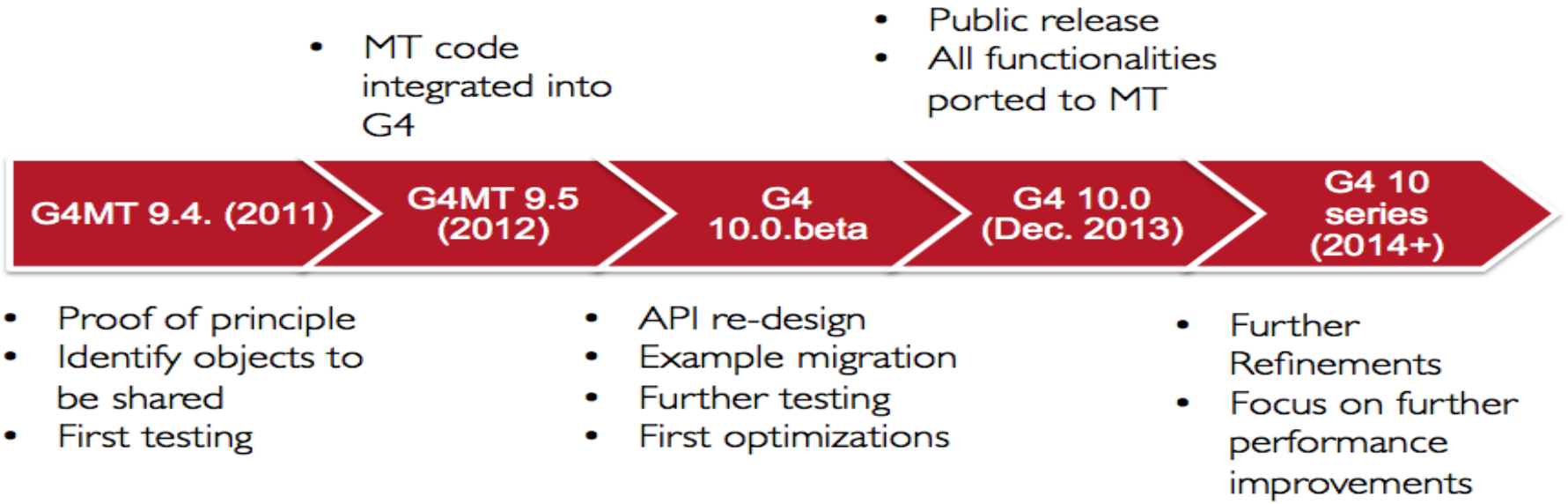
S. Agostinelli et al.
Geant4: a simulation toolkit
NIM A, vol. 506, no. 3, pp. 250-303, 2003

J. Allison et al.
Geant4 Developments and Applications
IEEE Trans. Nucl. Sci., vol. 53, no. 1, pp. 270-278, 2006

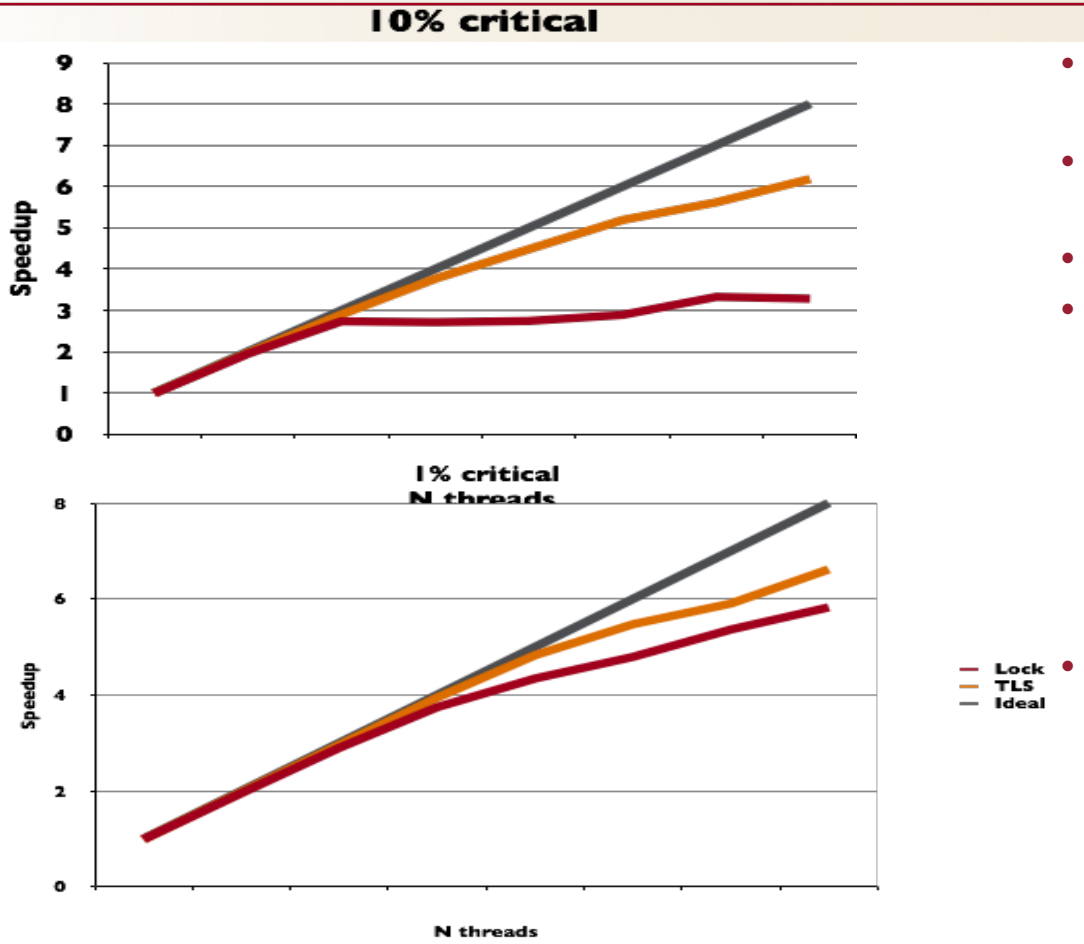
Options comparison: old results



Geant4 Multi Threading capabilities



Thread Local Storage



- 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 (currently 1%), only simple data types for static/global variables can be made TLS

NB: results obtained on toy application, not real G4

The splic-class mechanism concept

- 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