Multithreading capabilities in Version 10

Andrea Dotti (adotti@slac.stanford.edu)

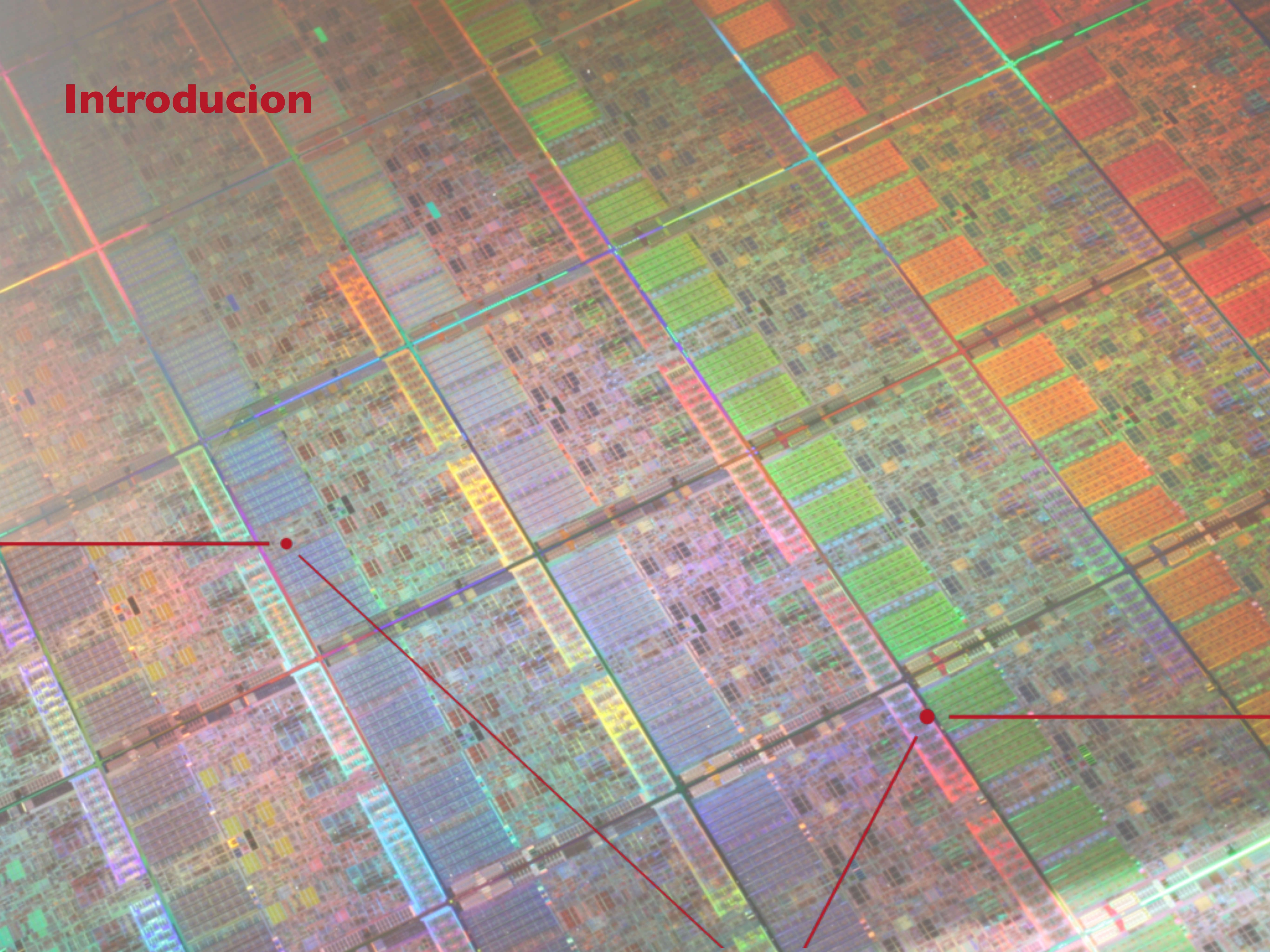
10th Geant4 Space Users Workshop

27-29 May 2014 Jackson Center, Huntsville, Alabama, USA

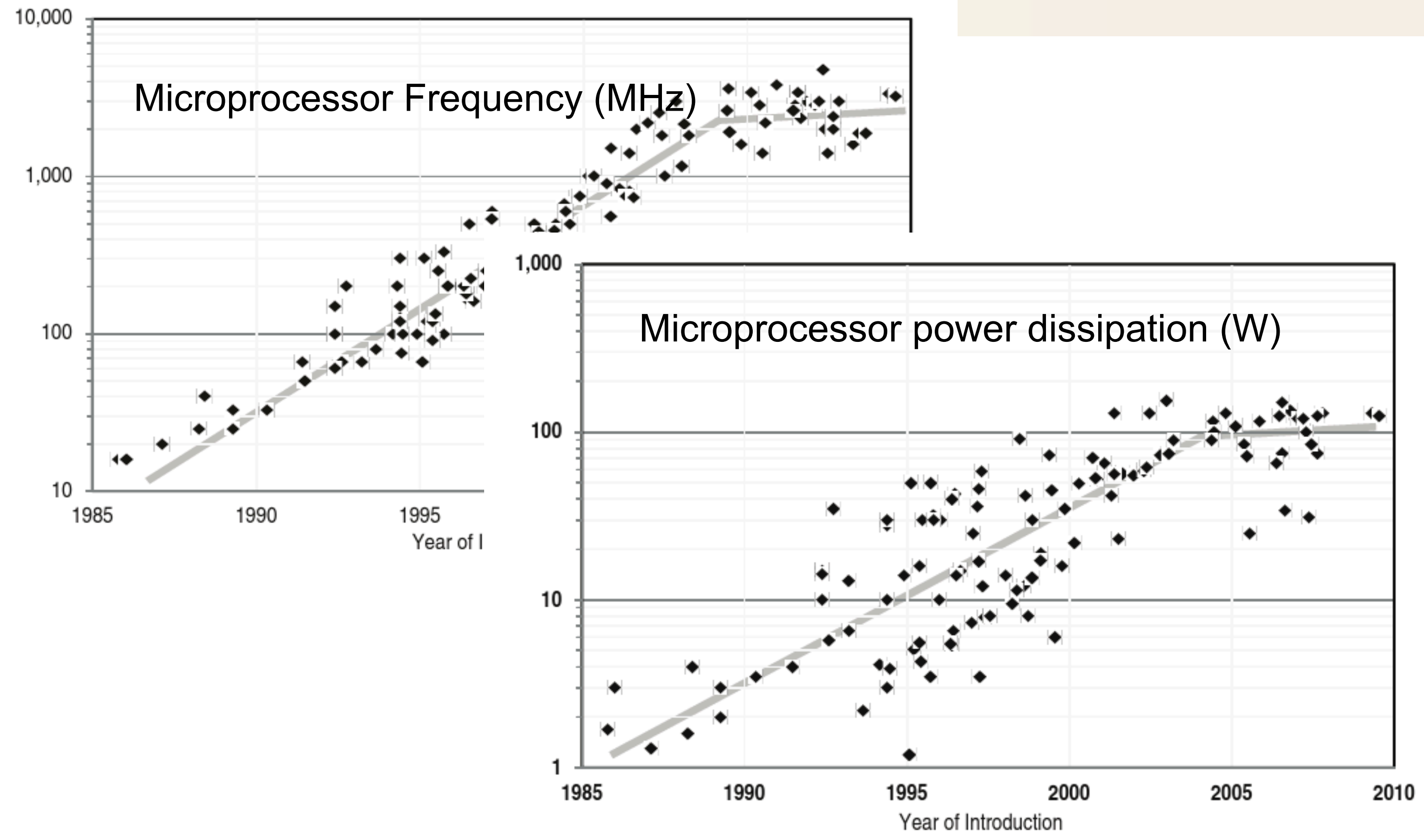
Outlook

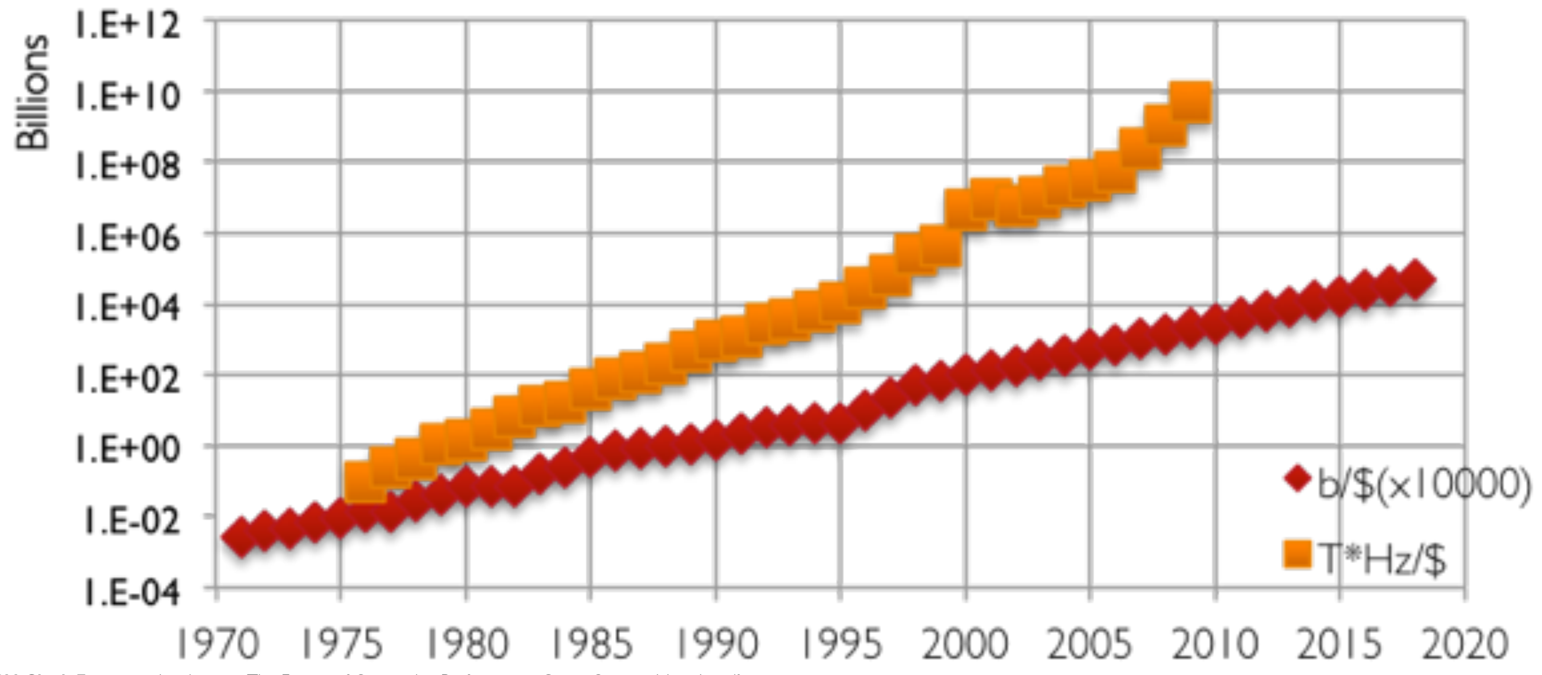
- Introduction: why we need multi-threading
- Design
- Results
- Extensions

Introducion



The challenges of many-core era

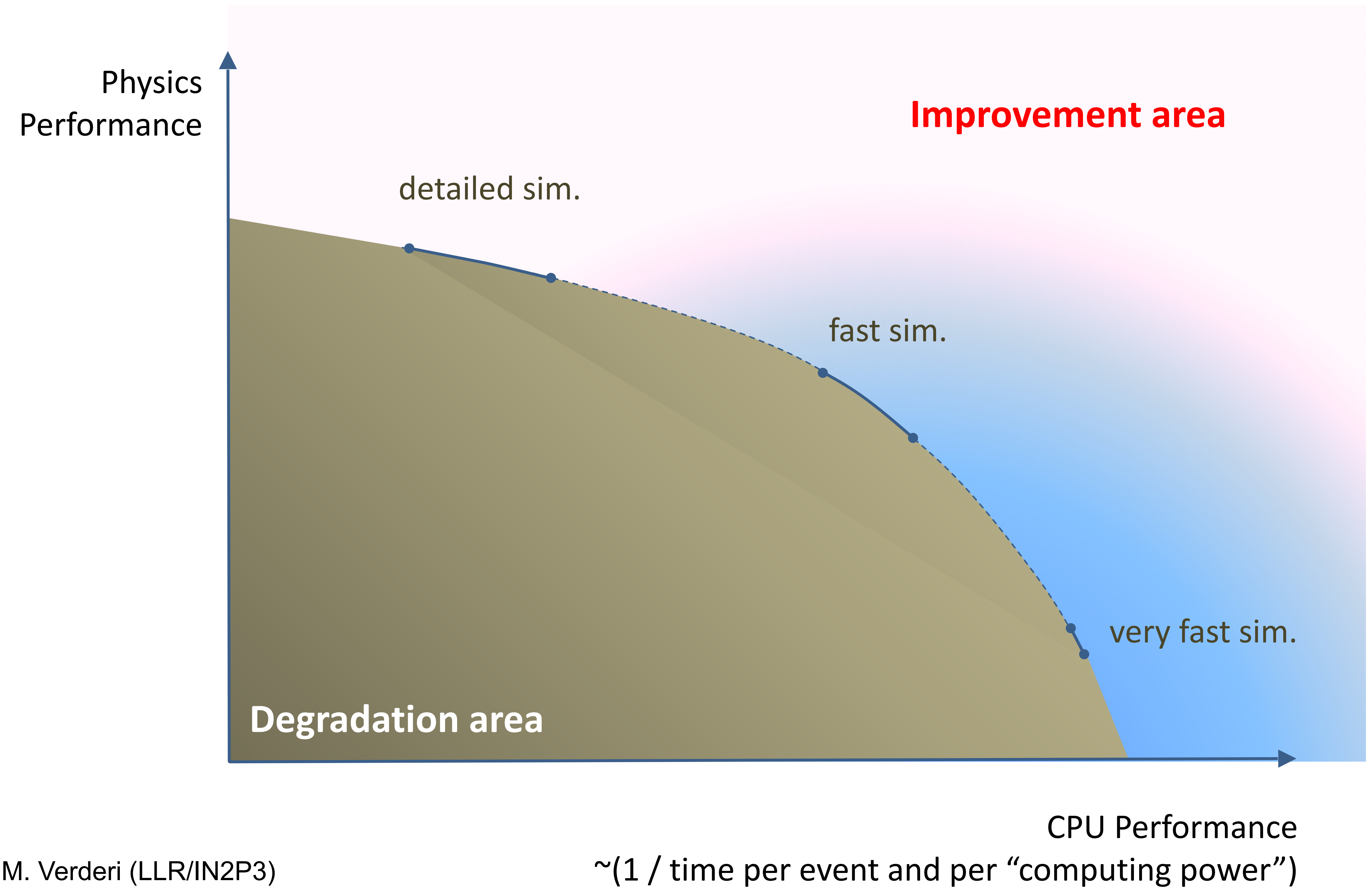




- Increase frequency of CPU causes **increase of power needs**
- Reached plateau around 2005
  - No more increase in CPU frequency
- However number of transistors /$ you can buy continues to grow
  - Multi/May-core era
- Note: quantity memory you can buy with same $ scales slower
- **Expect:**
  - Many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

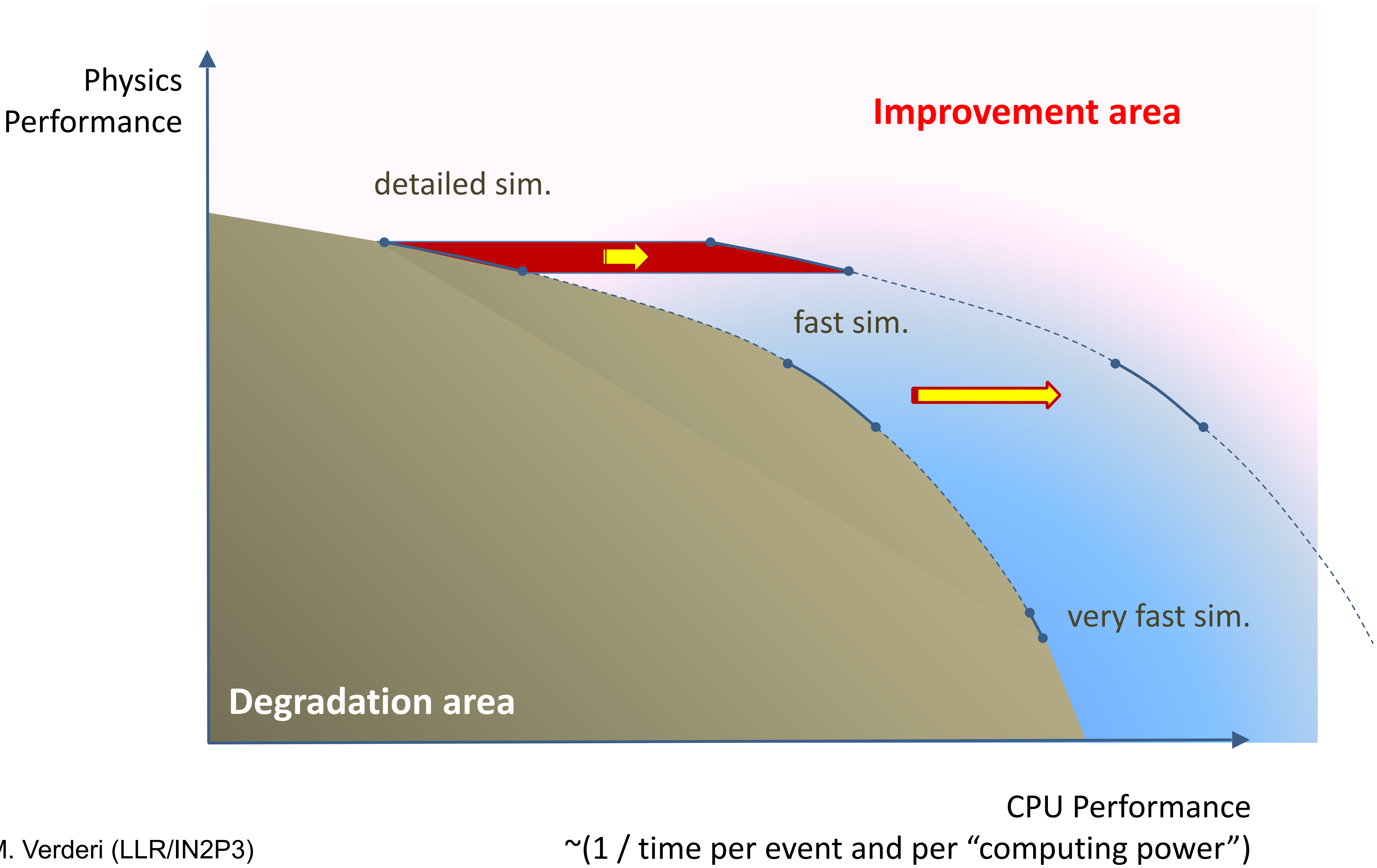
CPU Clock Frequecy 1and usage: The Future of Computing Performance: Game Over or Next Level?
DRAM cost: Data from 1971-2000:VLSI Research Inc. Data from 2001-2002: ITRS, 2002 Update, Table 7a, Cost-Near-Term Years, p. 172. Data from 2003-2018: ITRS, 2004 Update, Tables 7a and 7b, Cost-Near-Term Years, 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,"

CPU versus Physics performances



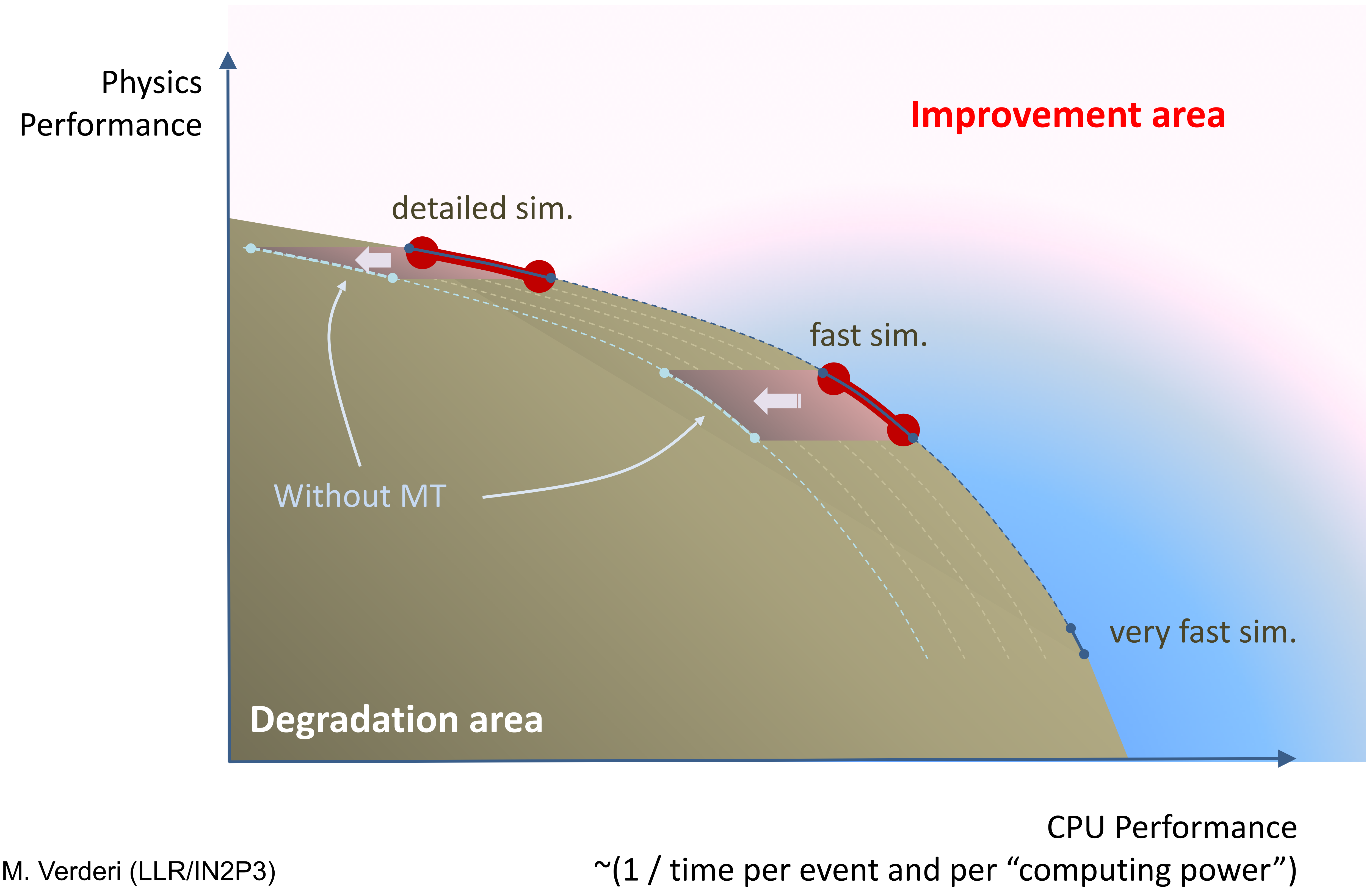
M. Verderi (LLR/IN2P3)

CPU versus Physics performances & new technologies



M. Verderi (LLR/IN2P3)

CPU versus Physics performances & Multi-threading



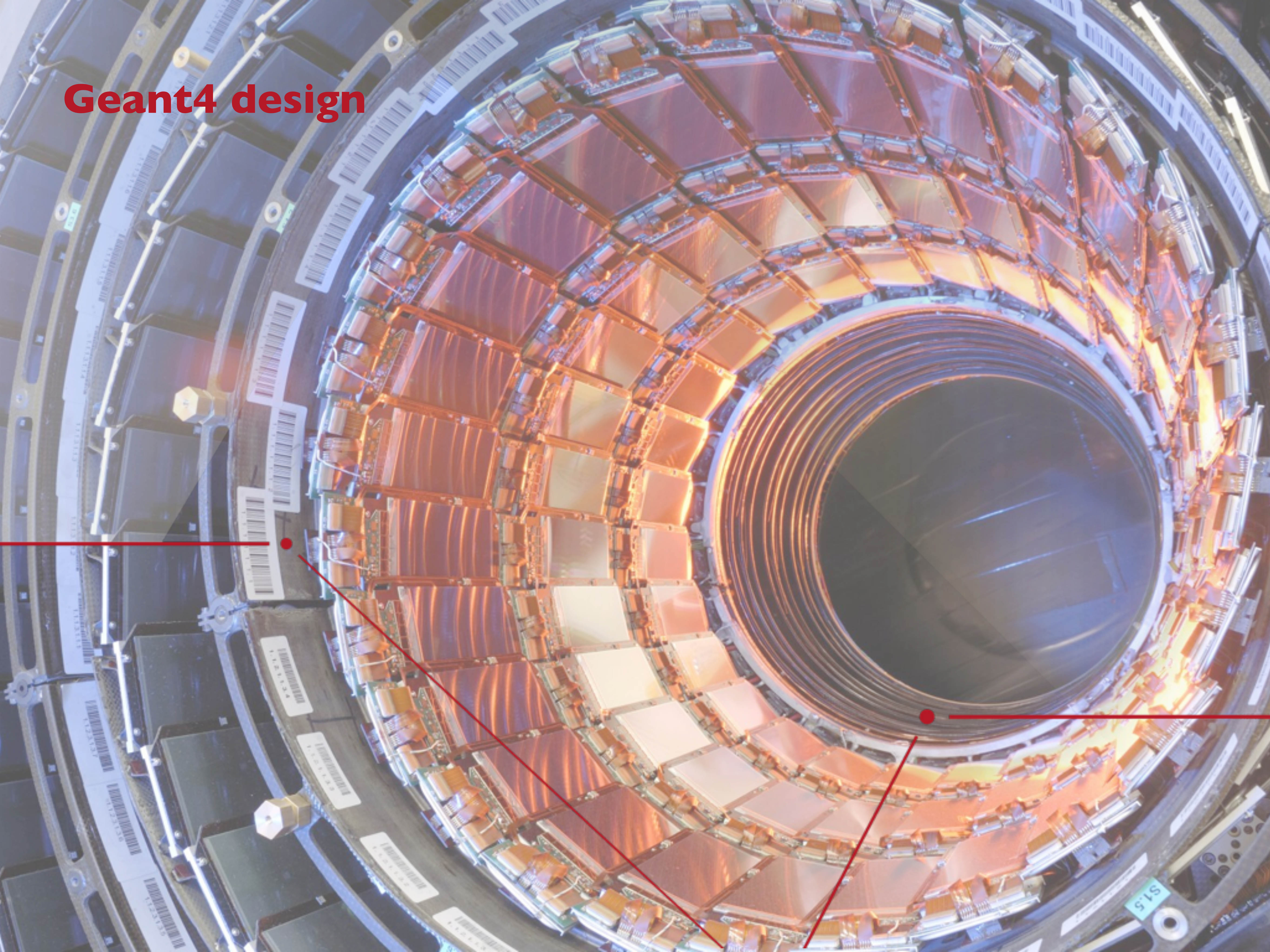
M. Verderi (LLR/IN2P3)

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 is the natural choice

Geant4 design

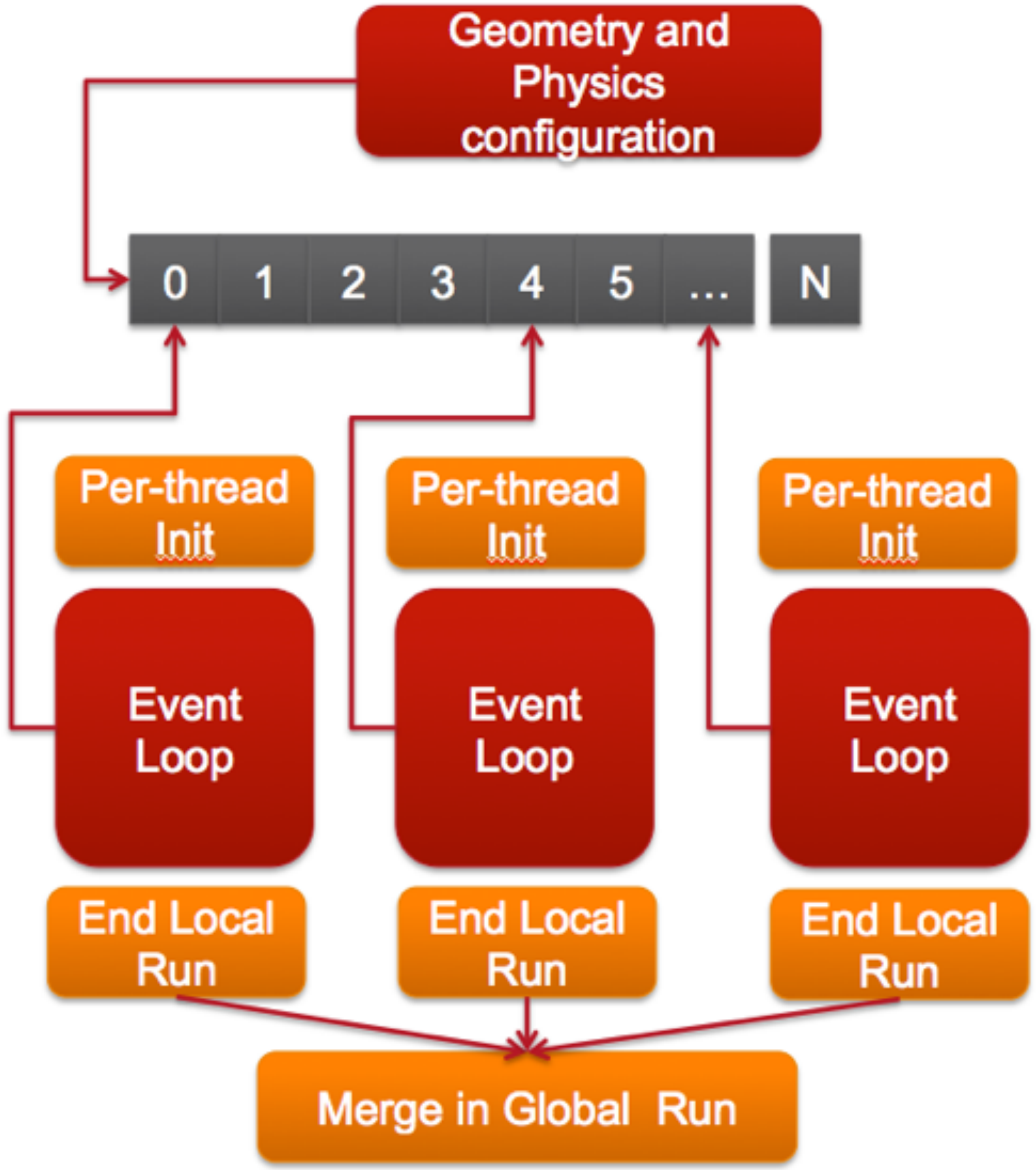


Event Level Parallelism

- **Version 10 supports (optional) event-level parallelism**
  - Can now take advantage of the full CPU power of your machine which likely has more than 1 core
  - You may still opt for a sequential (non-multi-threaded) build (e.g. if you rely on non thread-safe external code)

- Installation
  - No new dependencies, see the Geant4 Installation Guide accessible from the Geant4 web page (User Support -> Documentation -> Installation Guide)
  - Turn on MT via cmake switch
  - See also latest developments and performance at http://twiki.cern.ch/twiki/bin/view/Geant4/MultiThreadingTaskForce

General Design



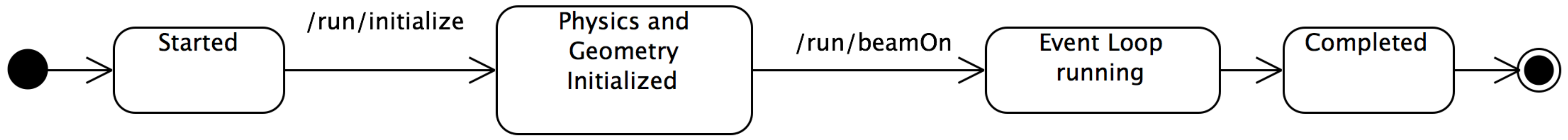
Per-event seeds pre-prepared in a "queue"

Threads compete for next event to be processes

Command line scoring and G4tools automatically merge results from threads

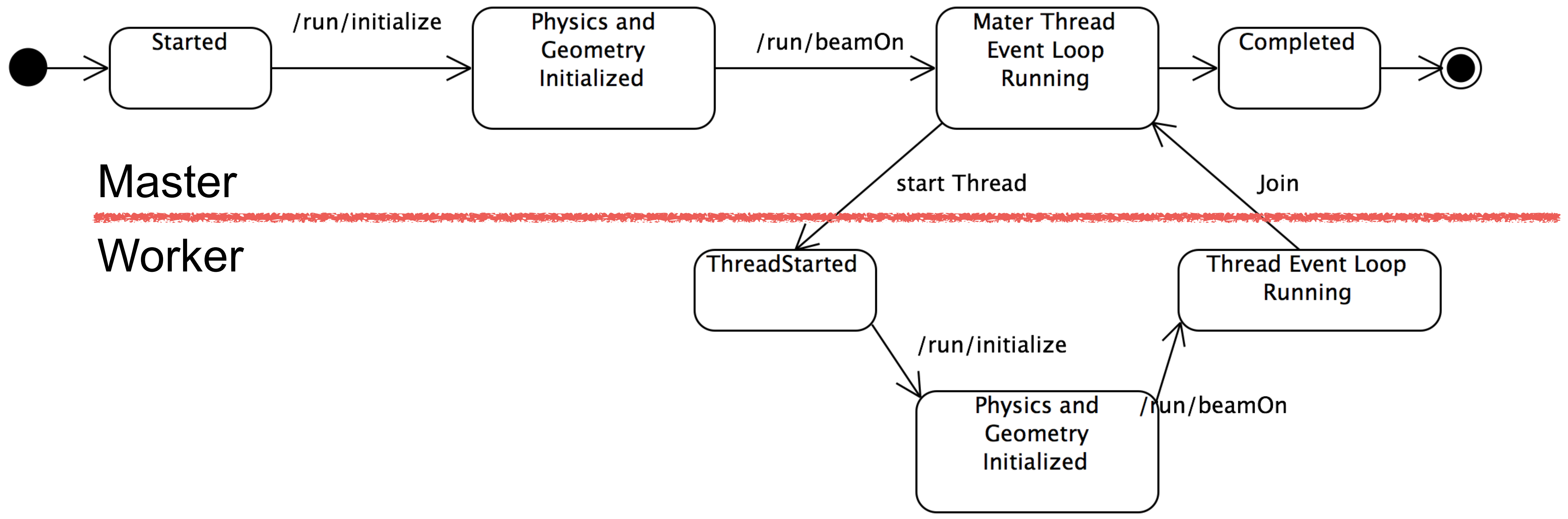
Simplified Master / Worker Model

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



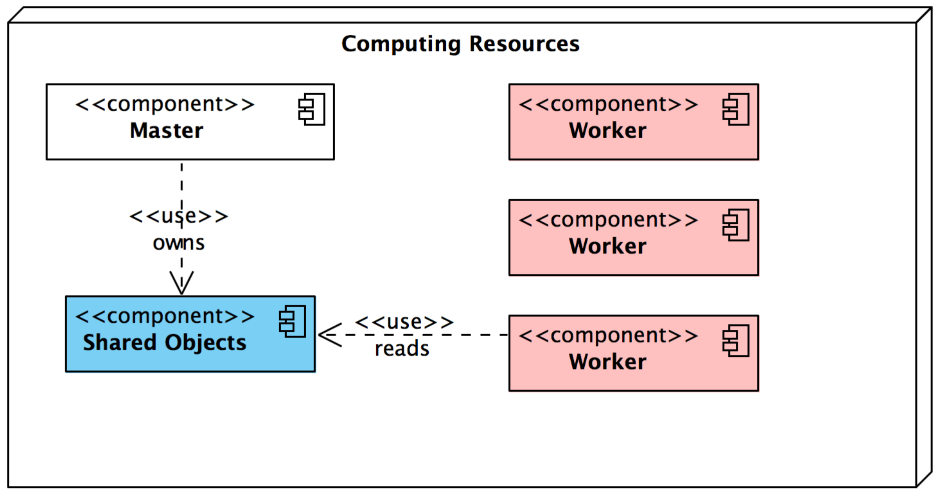
Simplified Master / Worker Model

- 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


Started
/run/initialize
Physics and Geometry Initialized
/run/beamOn
Mater Thread Event Loop Running
Completed
Master
Worker
start Thread
Join
ThreadStarted
Thread Event Loop Running
/run/initialize
Physics and Geometry Initialized
/run/beamOn

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
  - The reason for this is discussed in second part



How to configure Geant4 for MT

- `cmake -DGEANT4_BUILD_MULTITHREADED=ON […]`
- Requires "recent" compiler that supports ThreadLocalStorage technology (to be discussed Thursday) and pthread library installed (usually pre-installed on POSIX systems)
- Check cmake output for:

```
-- Performing Test HAVE_TLS
-- Performing Test HAVE_TLS - Success
```

- If it complains then your compiler is too old, sorry…
- Mac OS X, you need to use clang>=3.0 (not gcc!). On Mac OS X 10.7:

```
cmake -DCMAKE_CXX_COMPILER=clang++ -DCMAKE_C_COMPILER=clang \
      -DGEANT4_BUILD_MULTITHREADED=ON […]
```

- Sorry no WIN support!
- Compile as usual

Code Compatibility

- Some API have changed to enable MT (this is why this is a major release)
  - The exercises of this tutorial will show how to implement these correctly for MT

- **You can use an application developed for G4 Ver 9.6 without changing your code in sequential mode** (except for other mandatory modifications not MT-related)

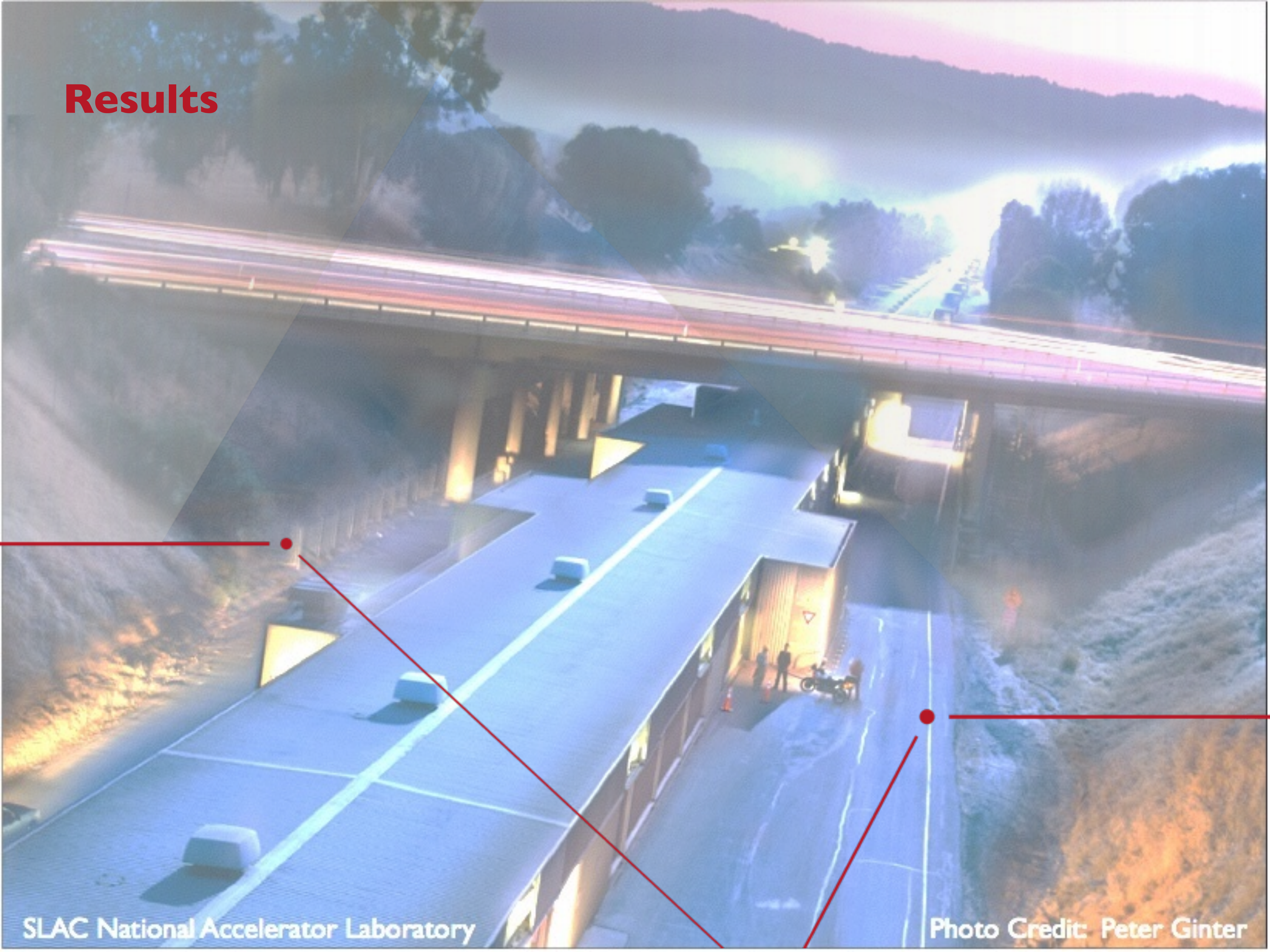
- **An MT-ready application, can also run in sequential mode without changing your code** (but not vice-versa)

Three steps to migrate

You can get benefits of MT with three steps

1. First **migrate** your sequential application to version 10.0 compiled in sequential mode
   - It's a major release so some migration is needed also non MT related (e.g. retired physics models)
2. Then **re-compile** Geant4 libraries activating MT but still keeping your application in sequential mode
   - It should work as expected
3. Then **migrate** to MT the application and start debugging it
   - For simple application should be trivial if no static/global objects are present
   - For larger user-code thread-safety has to be implemented

Results

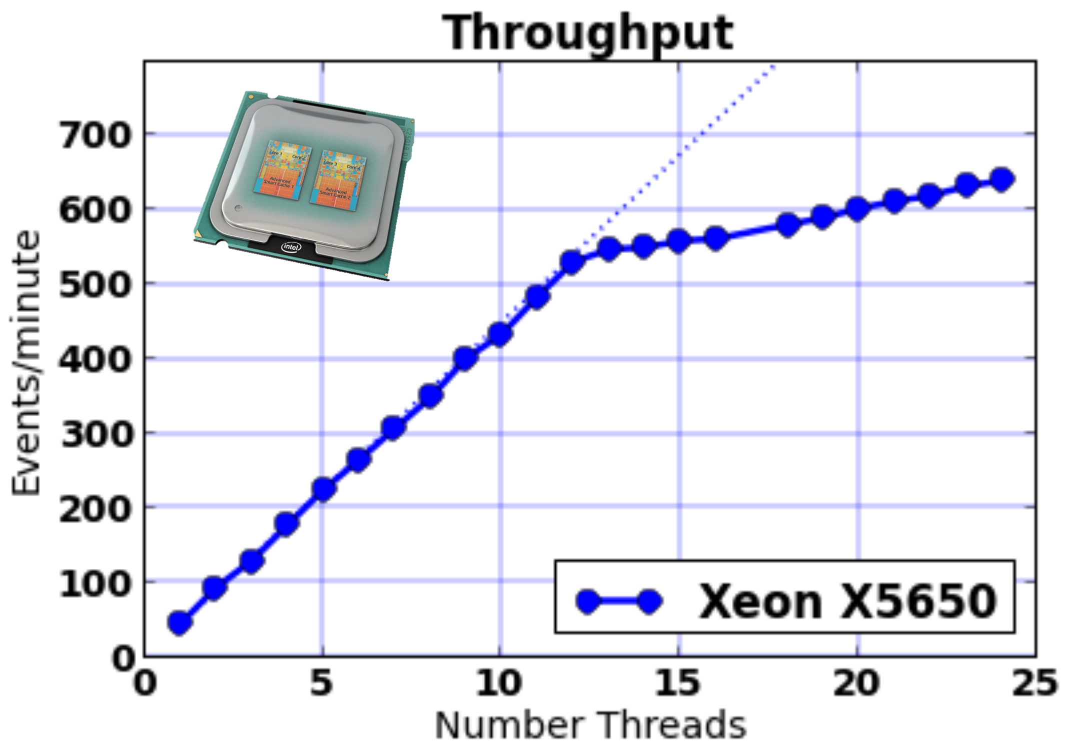

SLAC National Accelerator Laboratory
Photo Credit: Peter Ginter

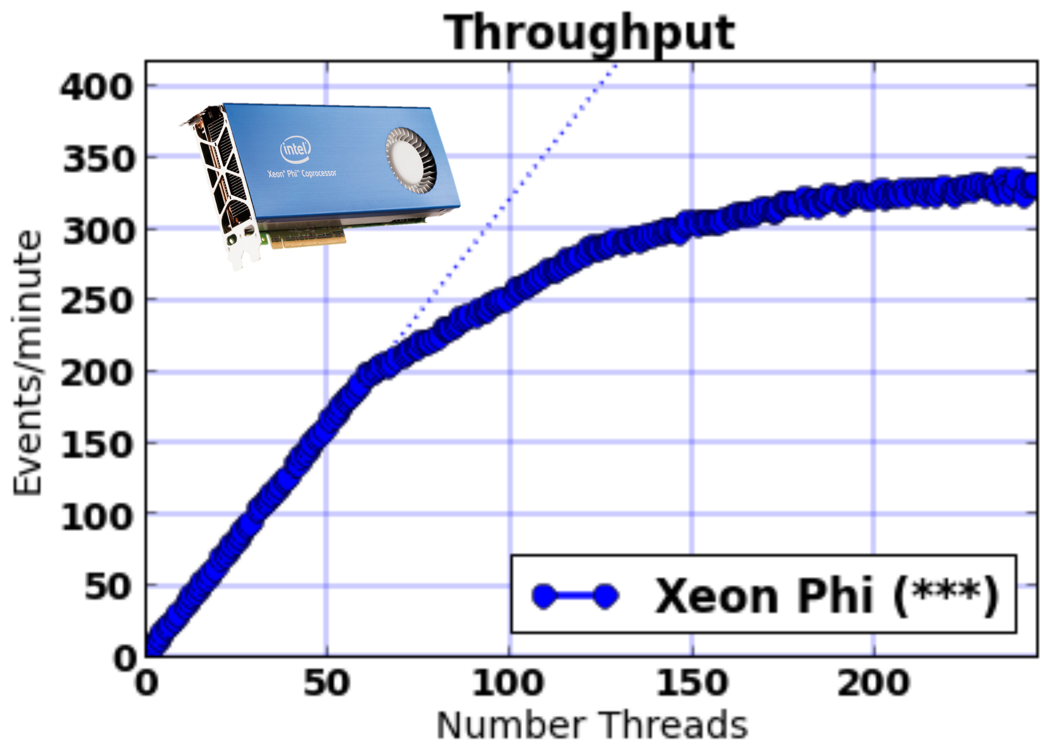
Reproducibility

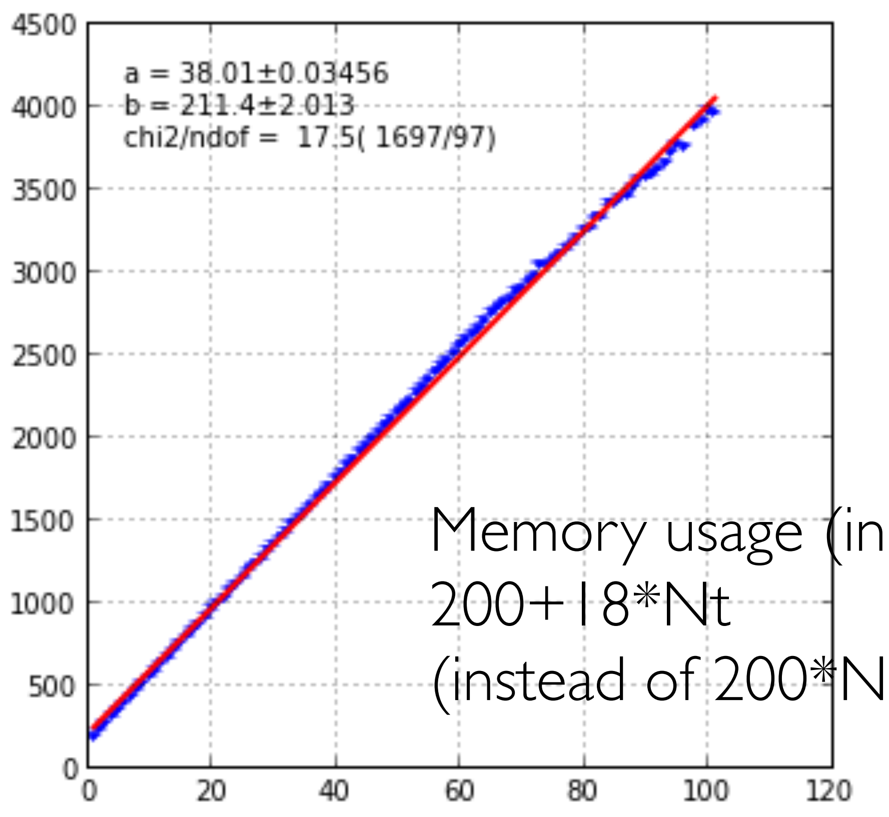
- Geant4 Version 10.0 guarantees strong reproducibility
  - Given a setup and the random number engine status it is possible to reproduce any given event independently of the number of threads or the order in which events are processed
- Note: (optional) radioactive decay module breaks this in MT, we are currently working on a fix
  - This does not mean the results are wrong!

- **Simulation results are equivalent between Sequential and MT**

CPU / Memory performances



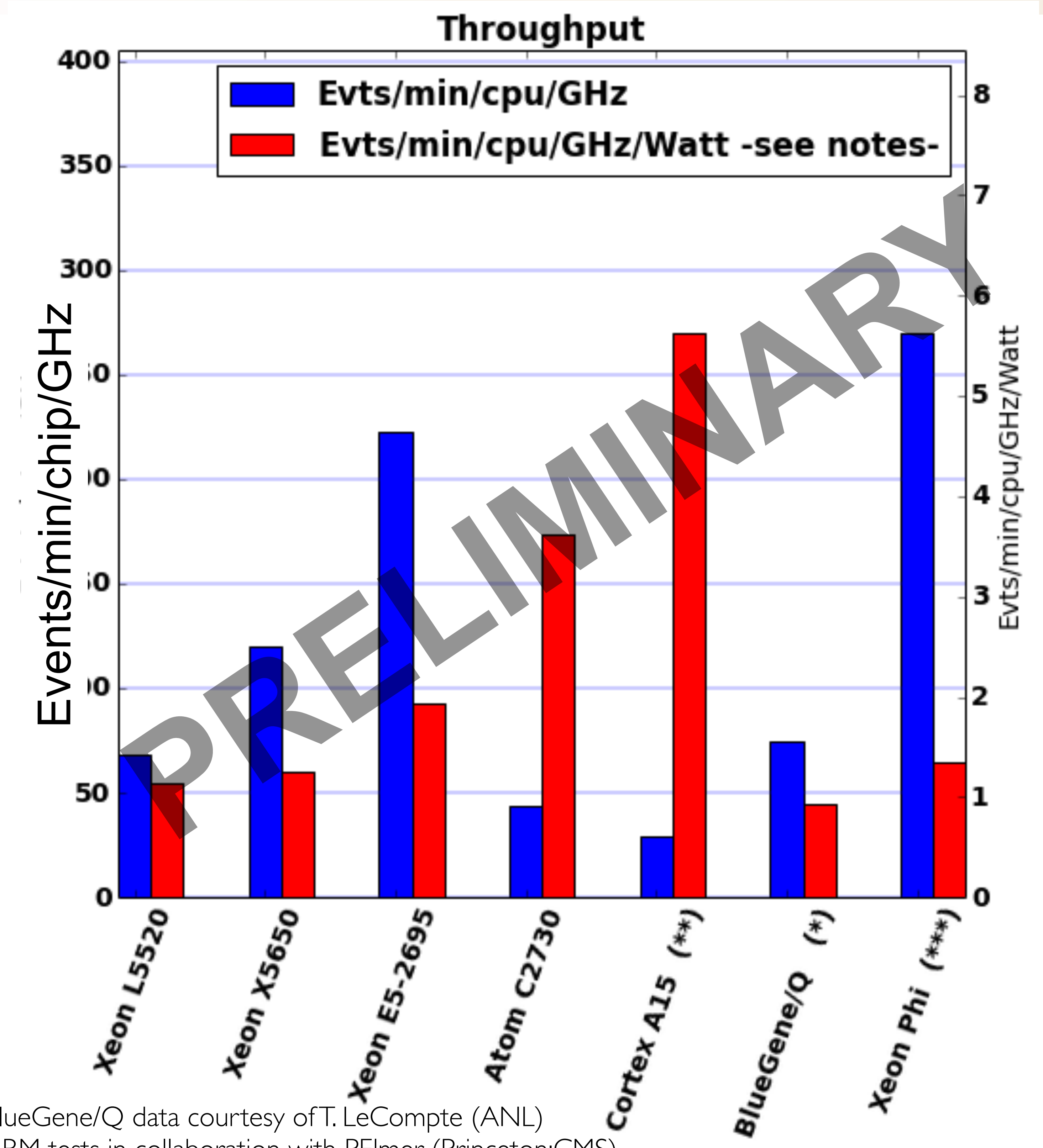




Memory usage (in MB)
200+18*Nt
(instead of 200*Np)

Obtained with HEP style geometry

Different Architectures



Geant4 has been run with success on a variety of hardware architectures:

- Intel / AMD
- MIC
- PowerPC (BG/Q)
- ARM / Intel Atom

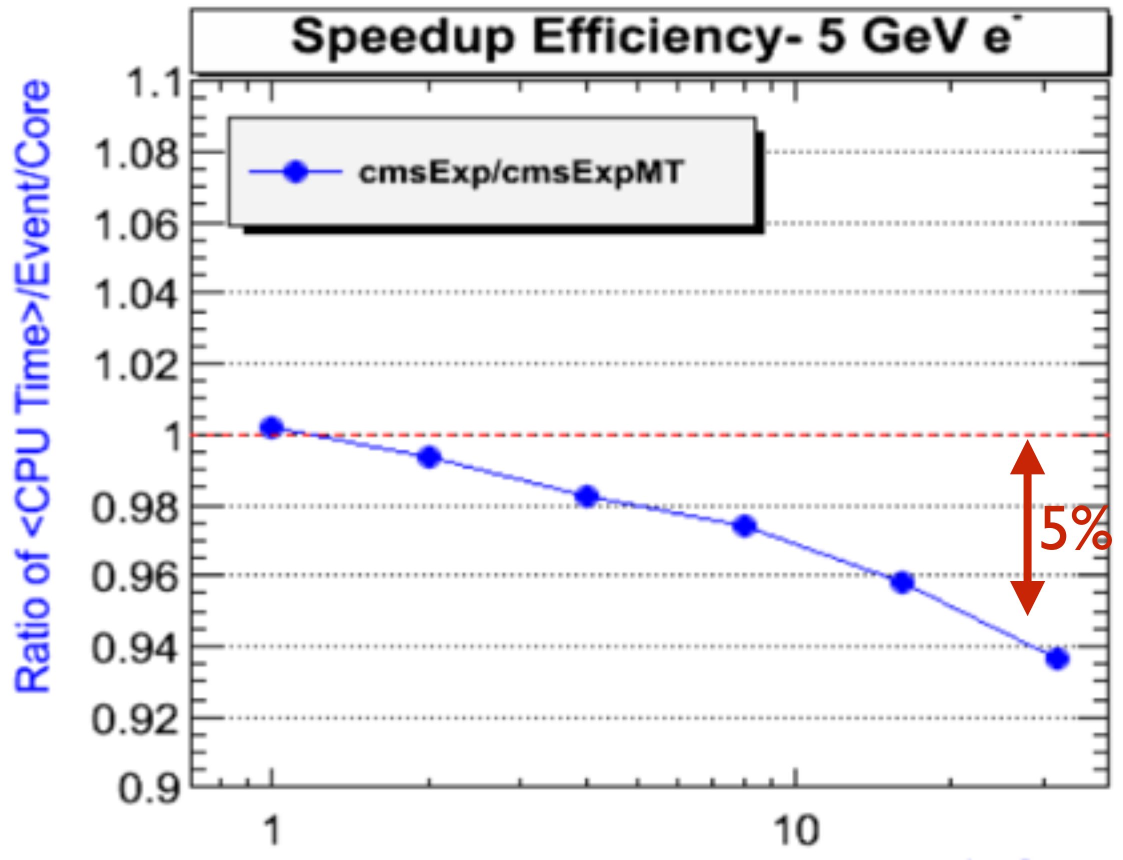
Absolute performances:

```
====== Max Events/min/cpu =======
154.4619 Intel Xeon L5520@2.27GHz
319.7392 Intel Xeon X5650@2.67GHz
534.6305 Intel Xeon E5-2695 v2@2.40GHz
73.8040 Intel Atom C2730@1.7GHz
46.8705 Exynos 5410 Octa Cortex-A15@1.6GHz
119.2088 BlueGene/Q@1.6GHz
334.4548 Intel Xeon Phi 7120P@1.238GHz
```

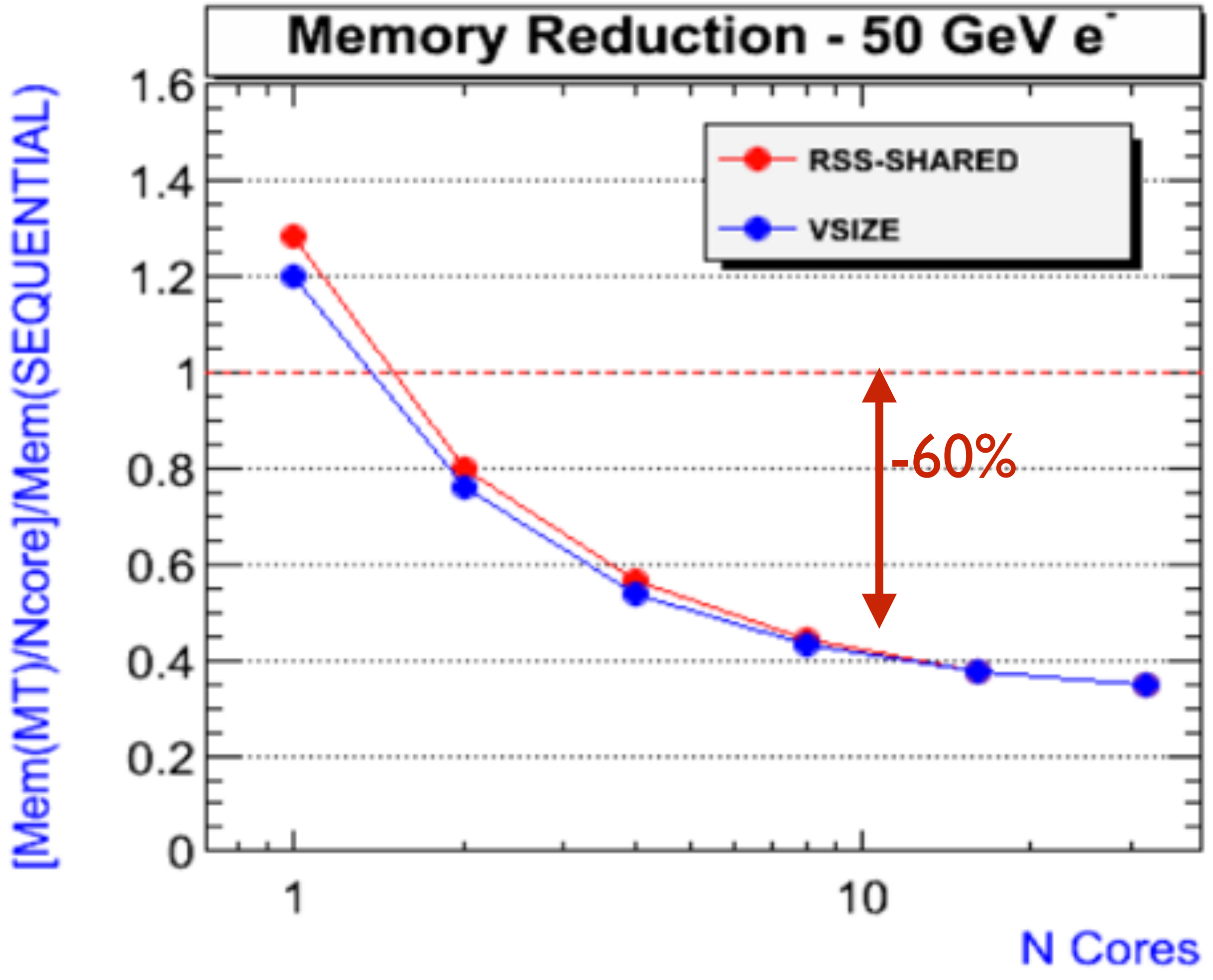
Obtained with HEP style geometry

BlueGene/Q data courtesy of T. LeCompte (ANL)
ARM tests in collaboration with P.Elmer (Princeton;CMS)
Hardware courtesy of OpenLab (CERN)

Comparison with Sequential

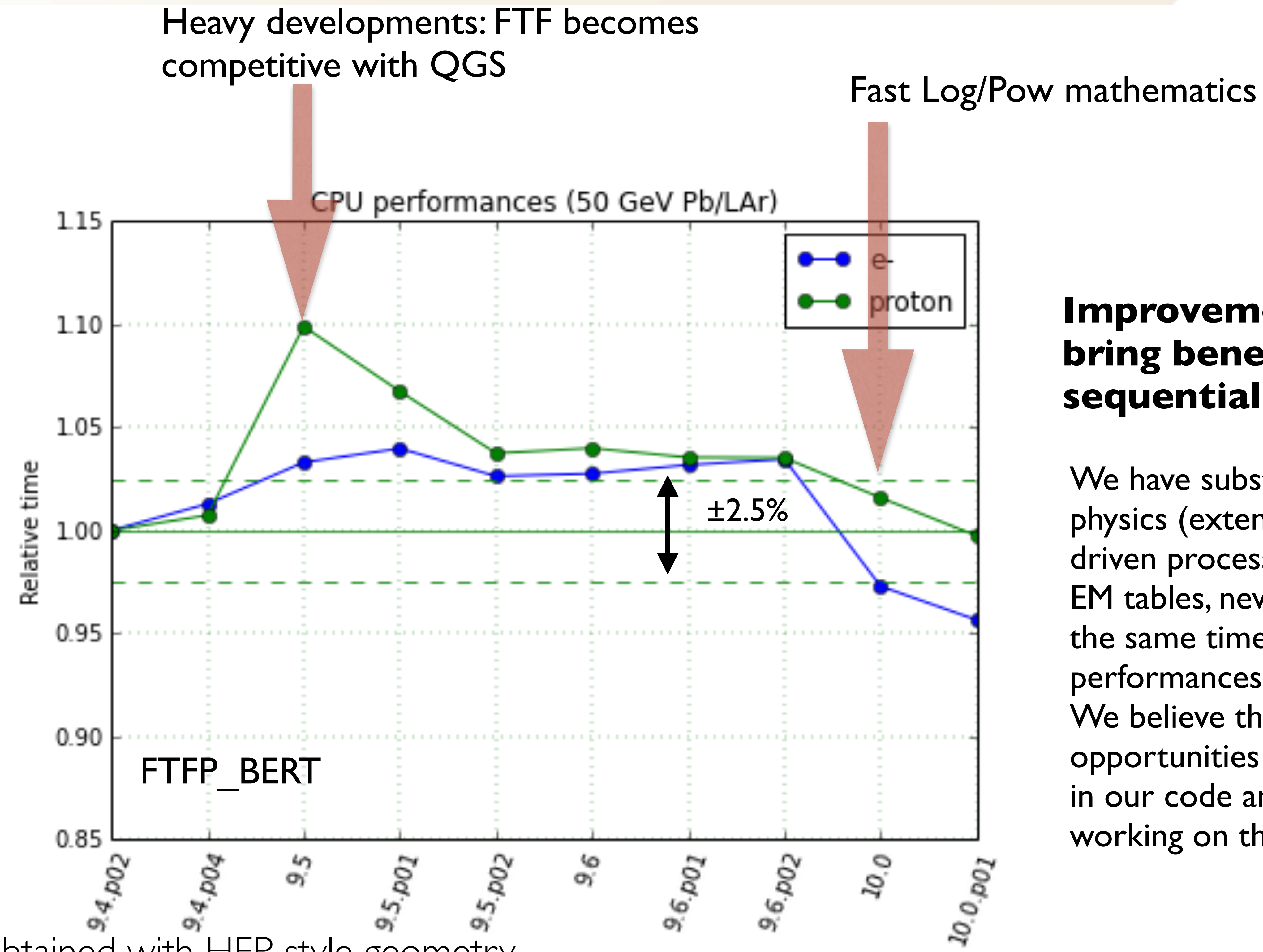


This is the reason why MT will be needed



Obtained with HEP style geometry

Absolute throughput (sequential)

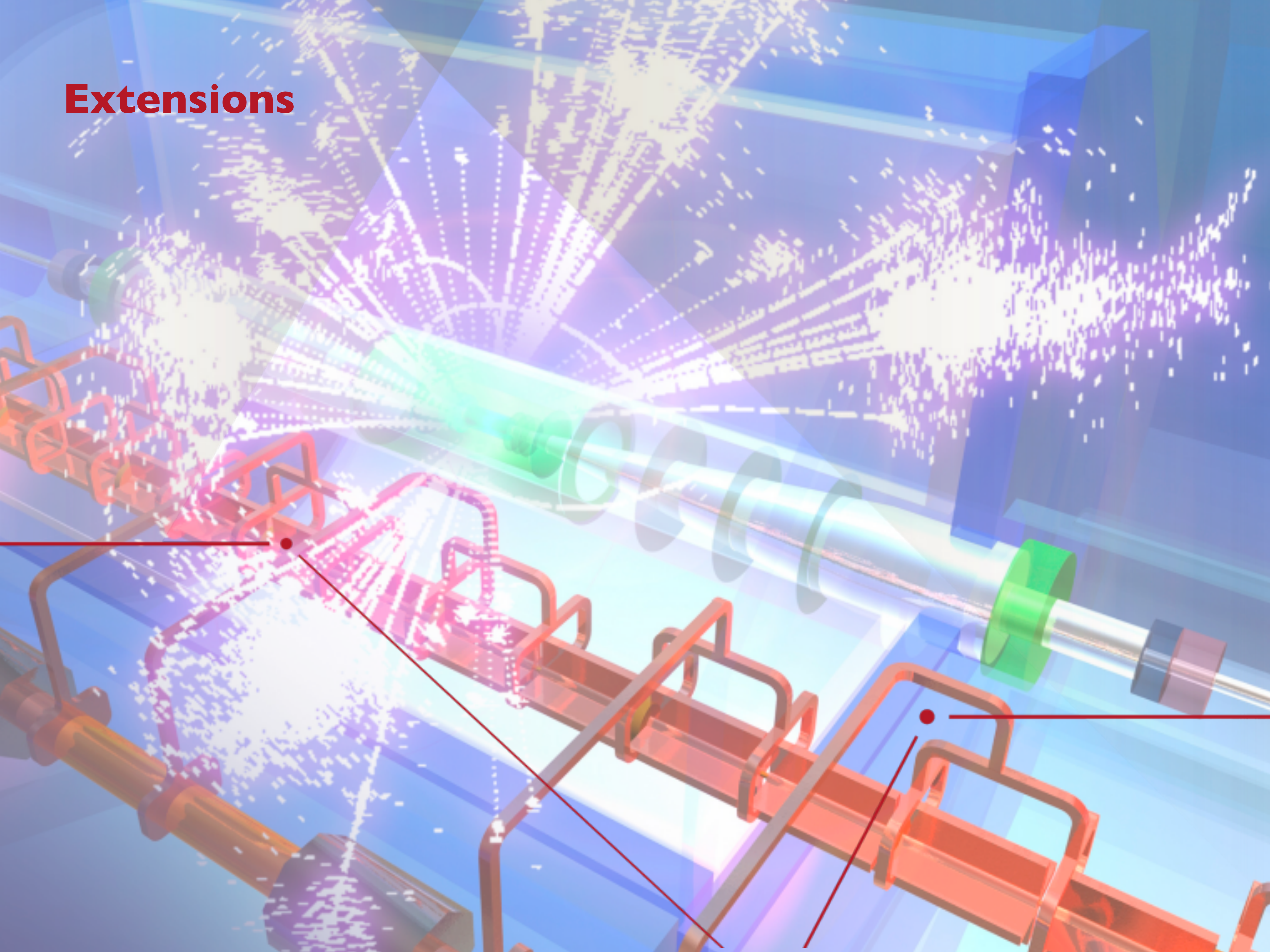


Obtained with HEP style geometry

Improvements for MT bring benefits also 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

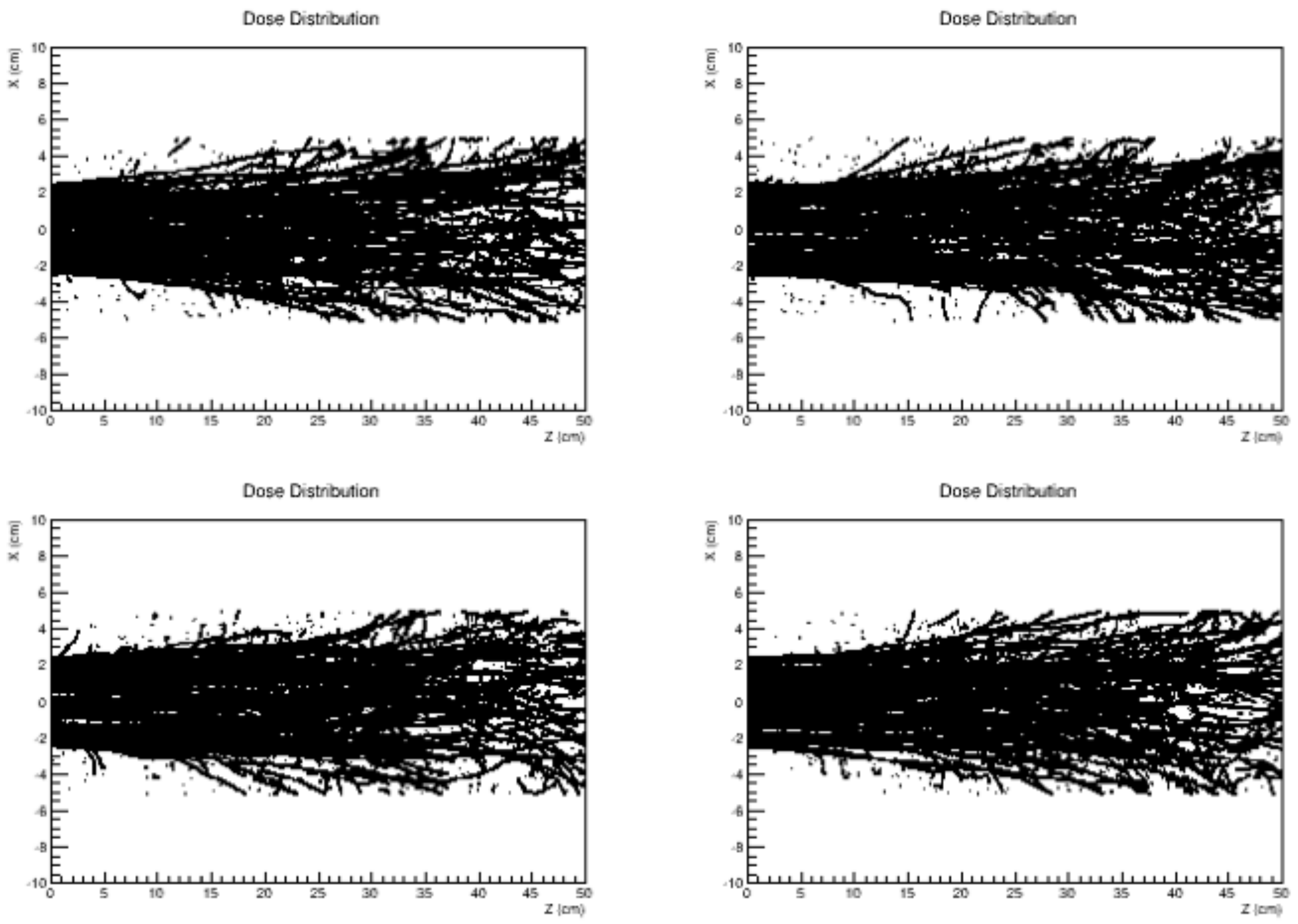
Extensions



Integration with external parallelization framewors

POSIX standards facilitate integration with external libraries/frameworks:

- **MPI based parallelism** already available in Geant4
- TBB based examples being developed

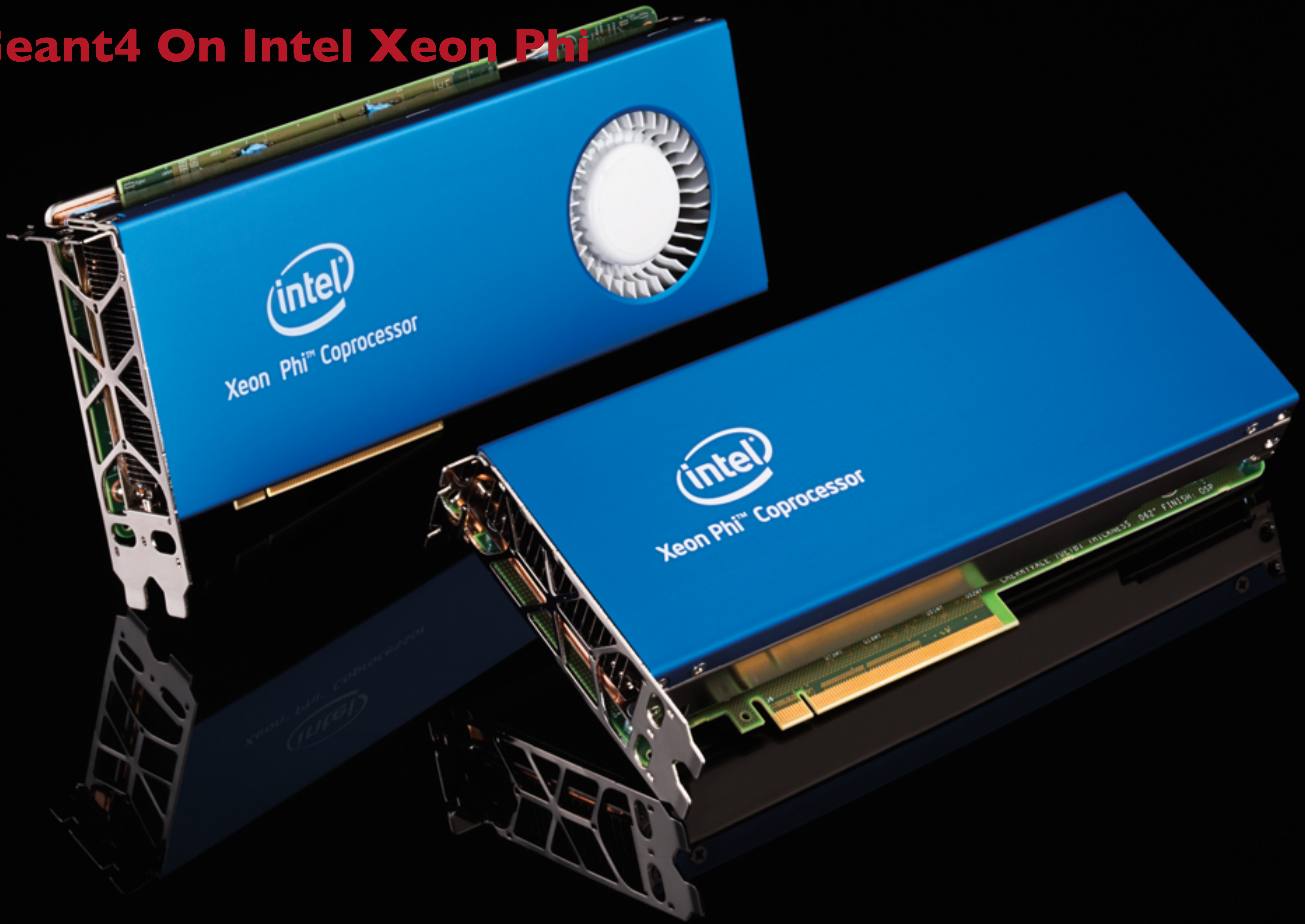


Example:
4 MPI jobs
2 threads/job
MPI job owns histogra

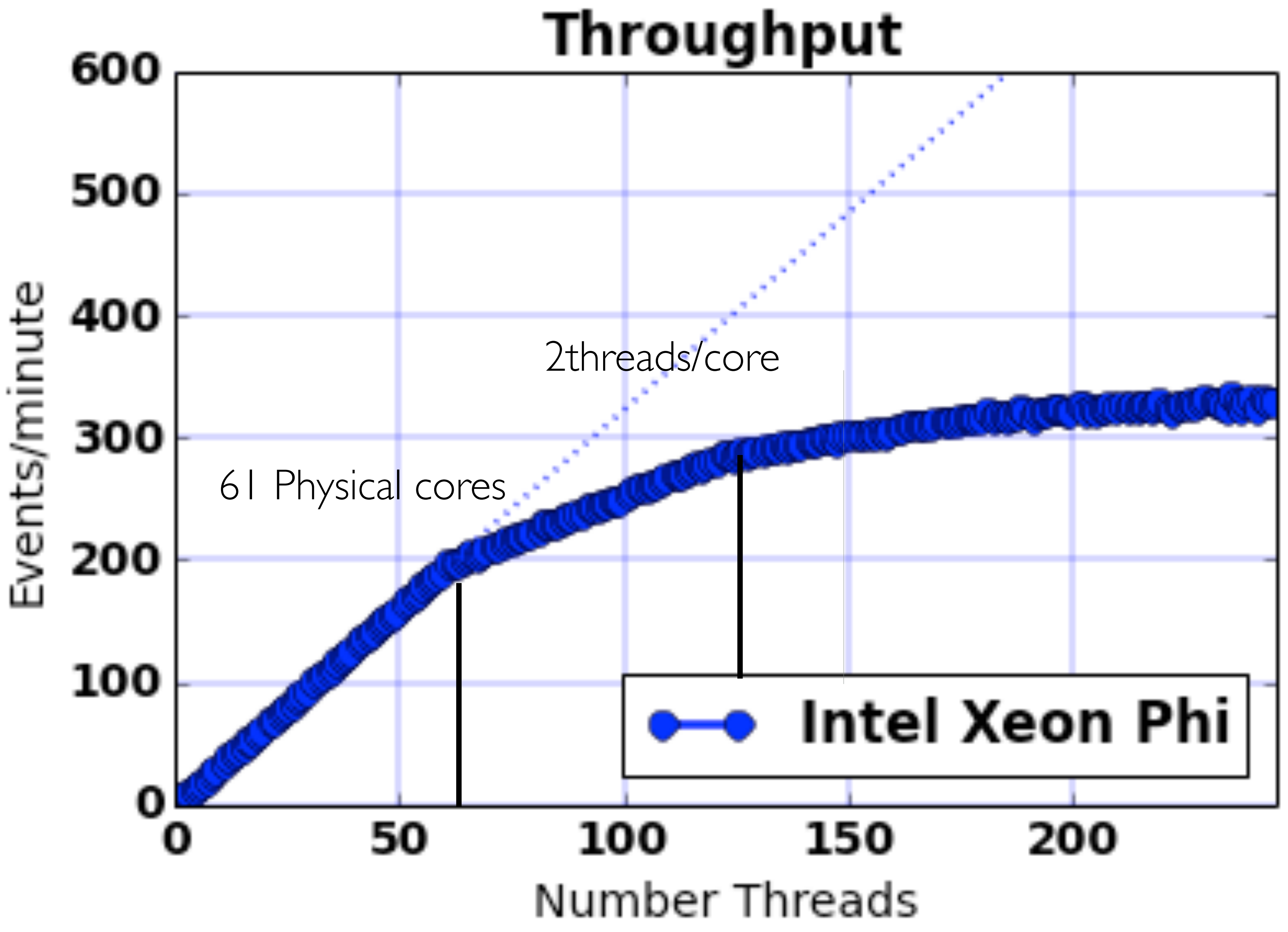
MPI vs MT

- MPI is a **multi-process** application
  - Copies of the same application are started on a (distribution) system
  - Each one is completely independent of the others
  - A communication layer is established between ranks
- MT is a **shared-model** application
  - Threads are independent but they share the memory of the machine
  - Special attention is needed to avoid race-conditions (thread-safety)
- In a **distributed memory system** (a cluster, a host with coprocessors) a mixed approach may be the best solution
  - Spawn, via MPI, multiple applications on nodes
  - On each node use MT to efficiently use memory
- If application is not memory bound pure MPI may be easier to use

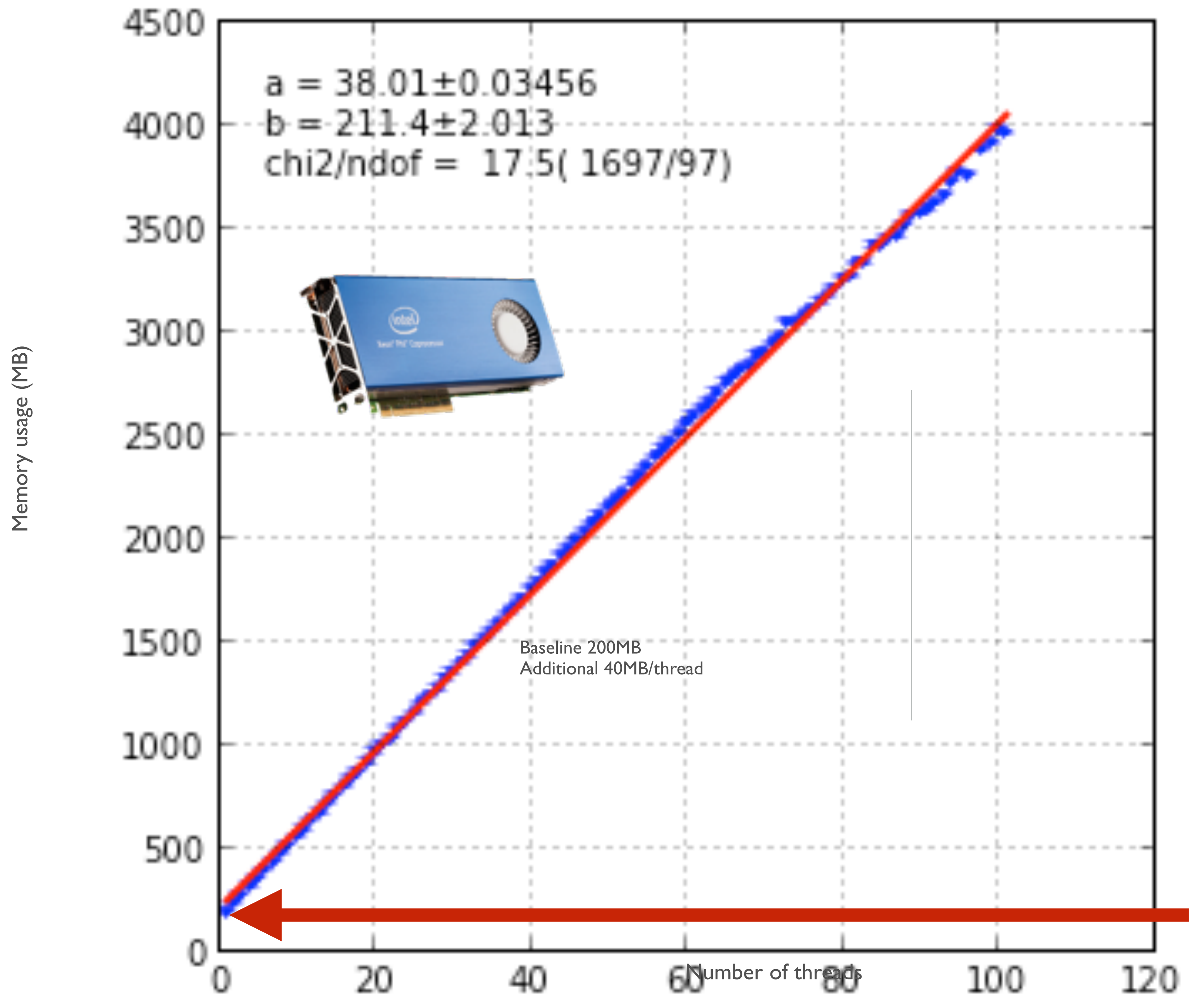
Geant4 On Intel Xeon Phi



Results: linearity



Results: memory usage

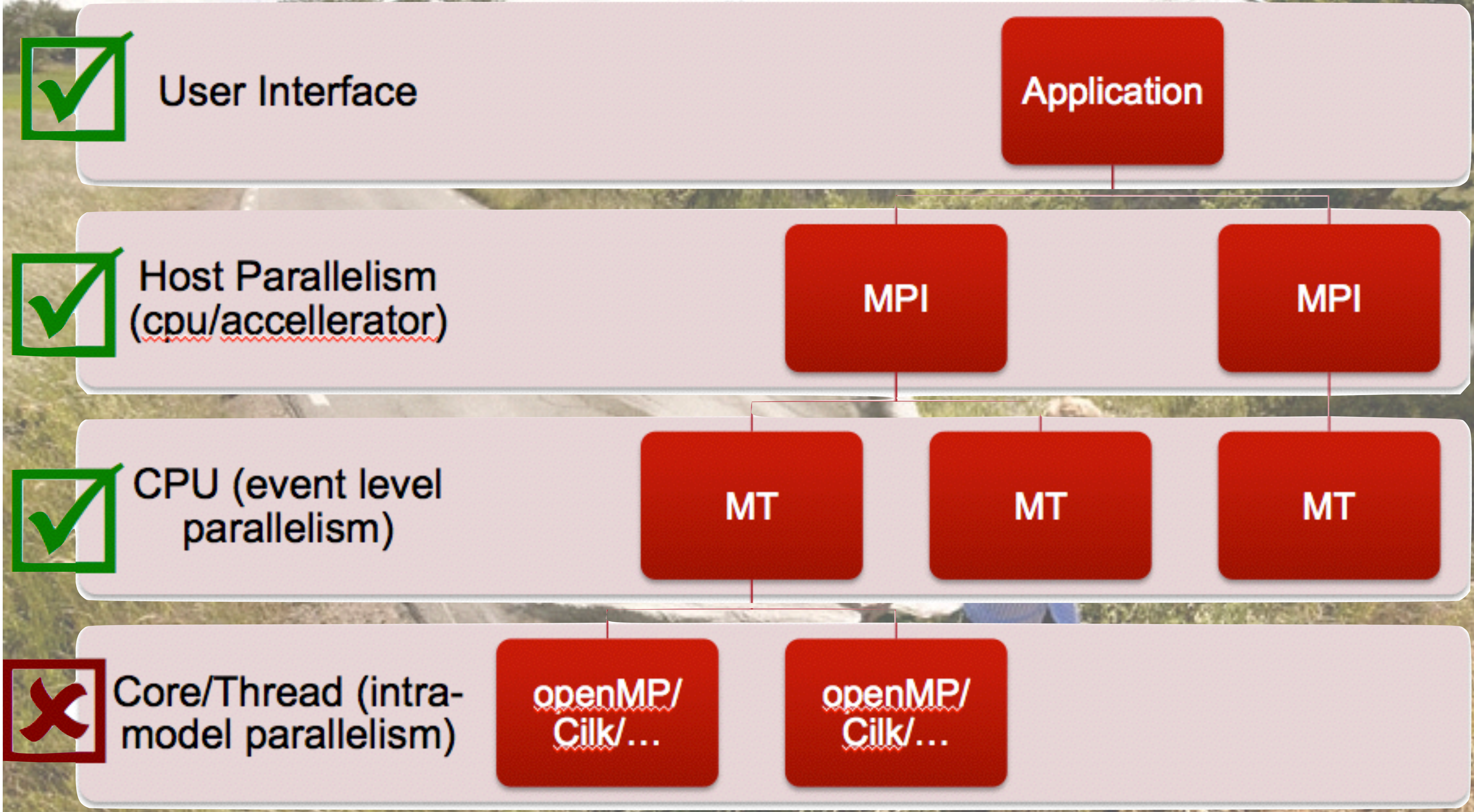


Slope: 38MB/thread

Baseline: 0 thread memory consumption

The road forward

The road forward


✓ User Interface
Application
✓ Host Parallelism (cpu/accellerator)
MPI
MPI
✓ CPU (event level parallelism)
MT
MT
MT
✗ Core/Thread (intra-model parallelism)
openMP/ Cilk/…
openMP/ Cilk/…

Conclusions

- Geant4 Version 10.0 supports event level parallelism via multi-threading
  - Implements a master/worker model
  - Most memory consuming objects: geometry and EM physics tables are shared between threads
- Very good results achieved
  - Linearity of throughput achieved for better than 90%
  - Memory footprint kept under control
  - Different architectures tested: Intel, ARM, Xeon Phi, BlueGene/Q
  - Support MPI and TBB via examples
- In the future we will concentrate in further improving absolute performances
  - Improving intra-physics model performances
  - Evaluating C++11 , openMP (Cilk++,…)