Parallelization features
MT, MPI, TBB

11th Geant4 Space Users Workshop – 26-28 August 2015, Hiroshima

A. Dotti (adotti@slac.stanford.edu)
SLAC / EPP-Computing

U.S. DEPARTMENT OF ENERGY
Office of Science

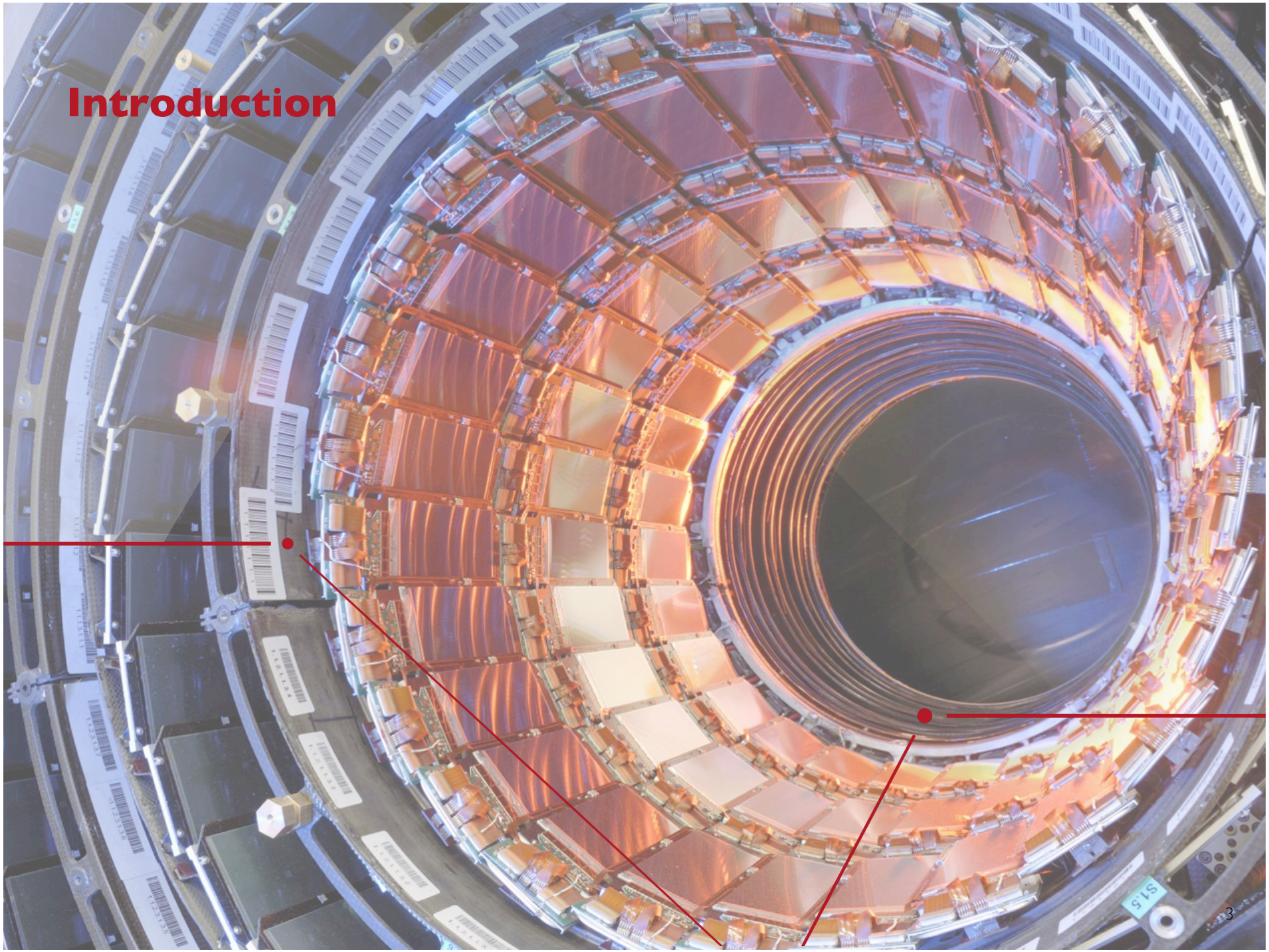
SLAC NATIONAL ACCELERATOR LABORATORY

Outline

SLAC

- Introduction and review of MT
- Recent results for MT
- TBB integration
- MPI integration

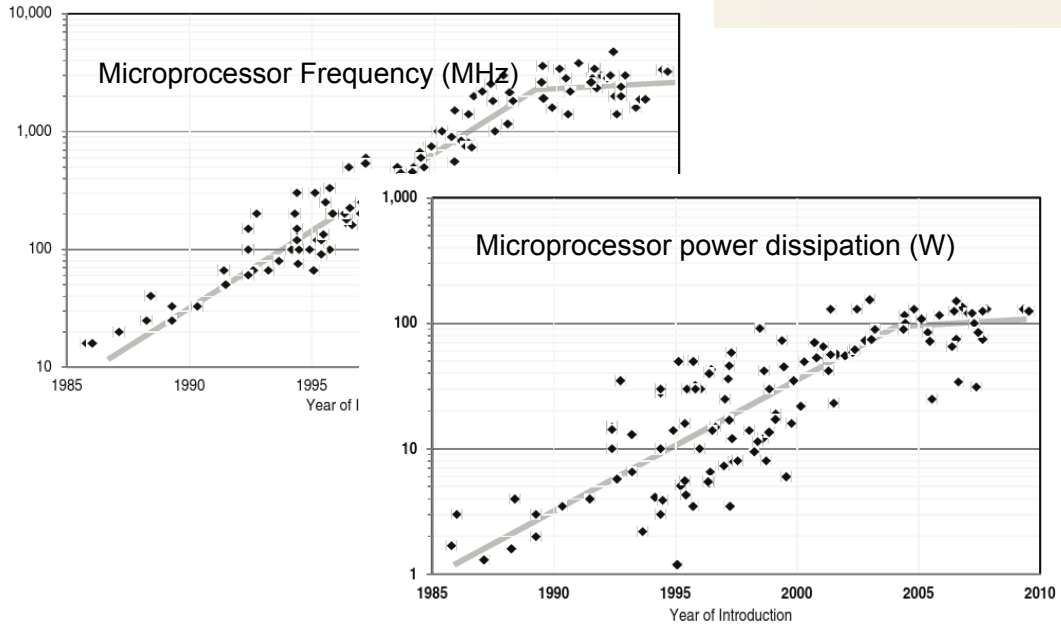
Introduction

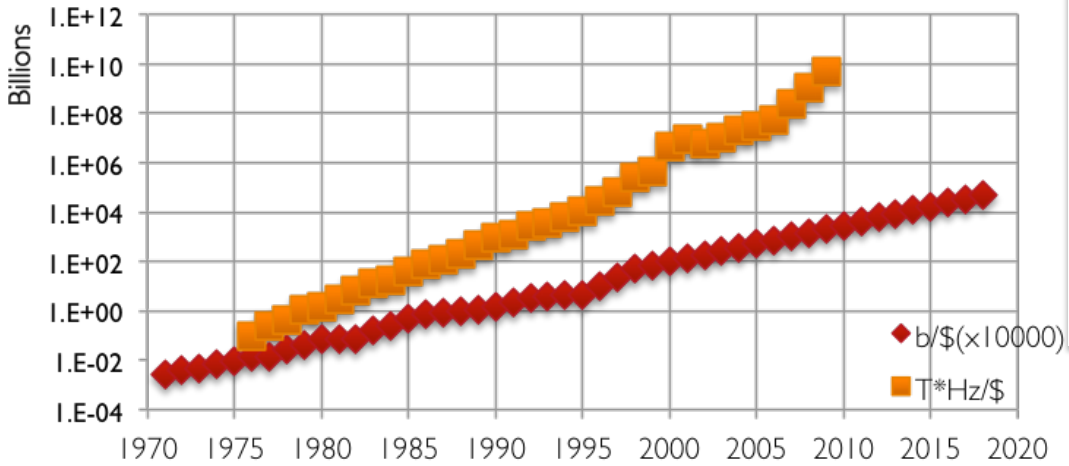


More details in: https://indico.esa.int/indico/event/50/session/11/contribution/13/material/slides/0.pdf

The challenges of many-core era

SLAC





Increase CPU frequency and feature size reduction: above thermal capacity of chips

Since ~2005:

- no more increate in CPU frequency
- still increase in number of transistors/$**: many-core era** (note: memory/$ scales slower)

Expect:

- CPU with many cores
- Single core performances not increasing (maybe even decreasing)
- Less memory/core

CPU Clock Frequecy l 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. 167. ; Average transistor price: Intel and Dataquest reports (December 2002), see Gordon E. Moore, "Our Revolution,"

More details in: https://indico.esa.int/indico/event/50/session/11/contribution/13/material/slides/0.pdf

In Brief

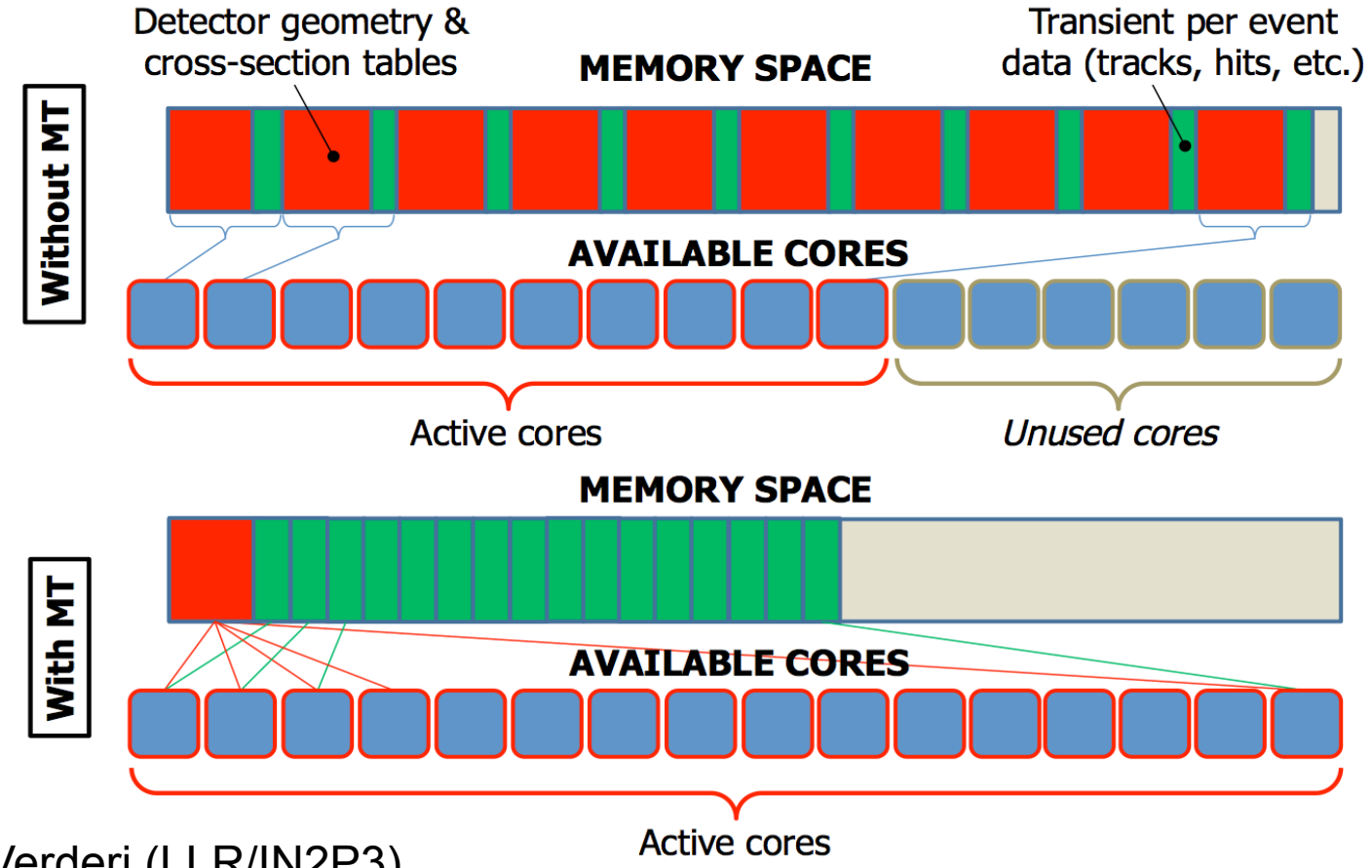
- Modern CPU architectures: need to introduce **parallelism**
- Memory and its access will limit number of concurrent processes running on single chip
- Solution: add parallelism **in the application code**

- 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

Multithreading in Geant4

SLAC

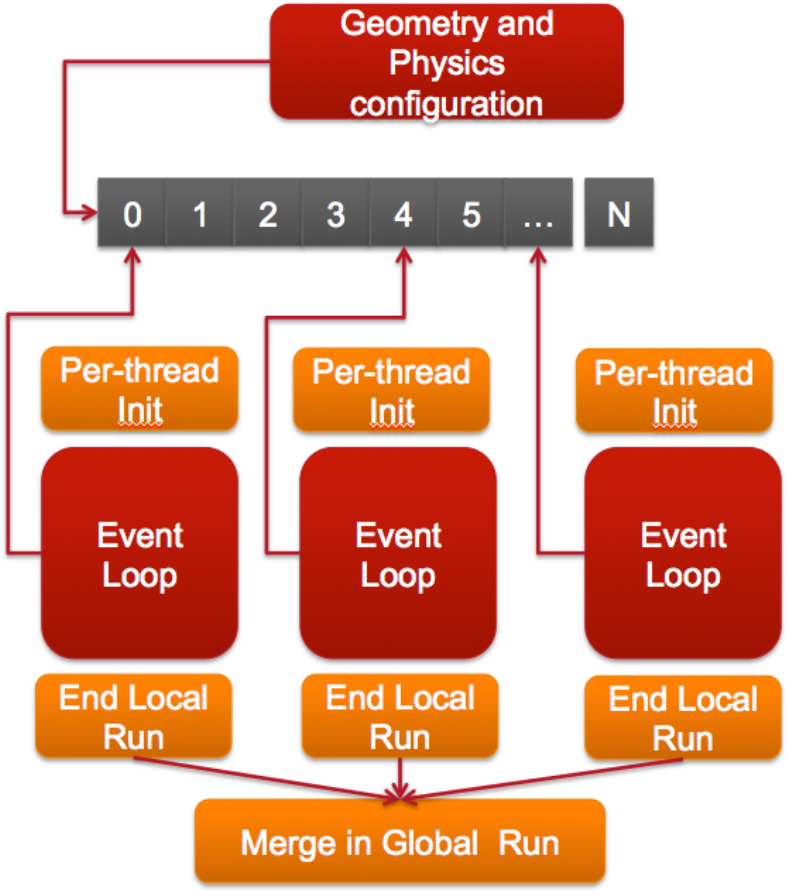
- Introduced in Version 10.0 (December 2013)
- Goal: effectively reduce memory footprint



M. Verderi (LLR/IN2P3)

Multithreading: Master/Worker Model

SLAC



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

Threads compete for next event to be processes (new in ref-08)

Command line scoring and G4tools automatically merge results from threads

Roadmap

SLAC

Version 10.0 (Dec. 2013)

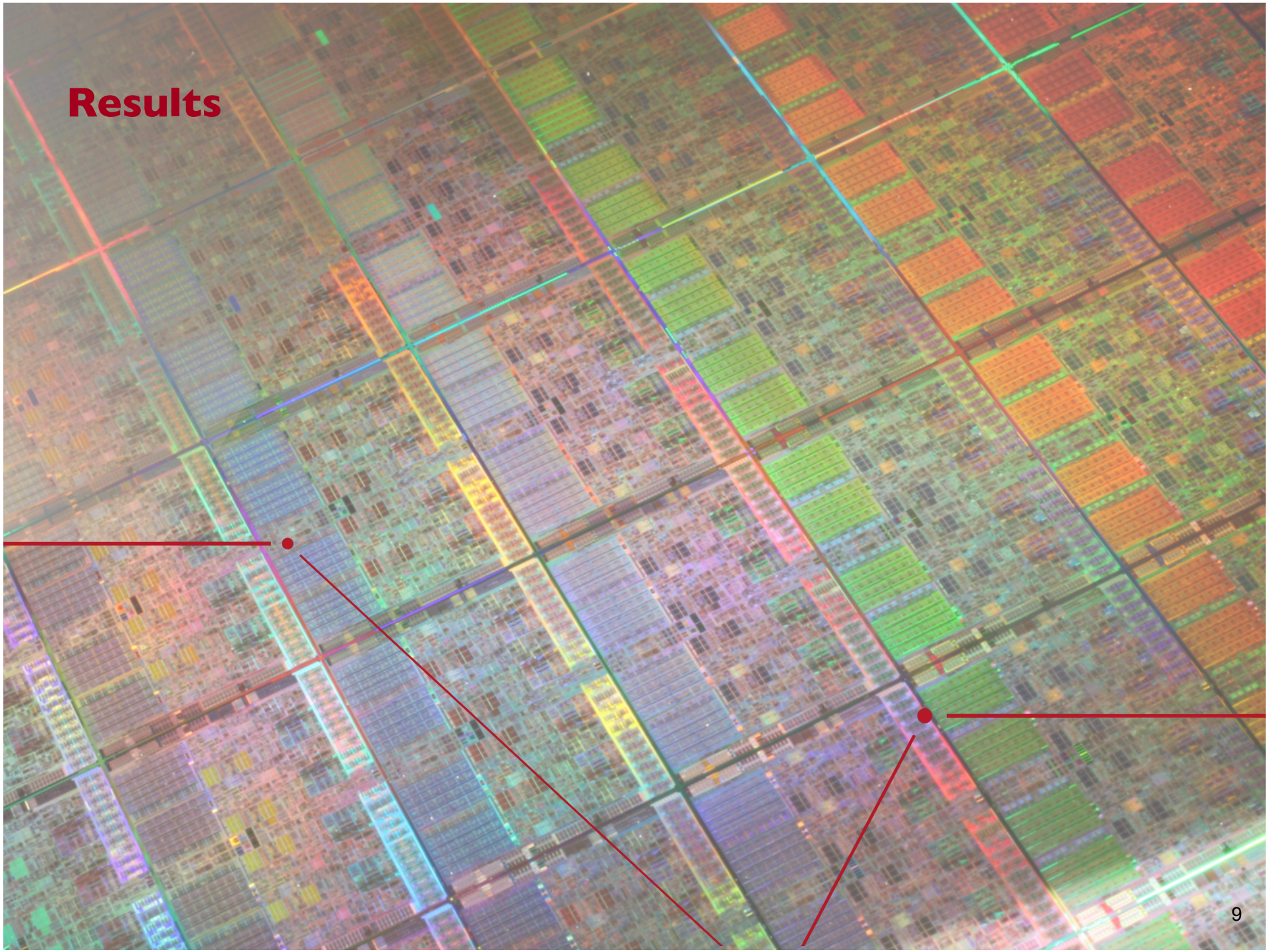
- Implement correct MT behavior (remove race conditions)
- Memory reduction from geometry and physics

Version 10.1 (Dec. 2014)

- Improve migration some components (GPS, RDM, Vis)
- Obtain further x2 memory reduction

Version 10.2 (Dec. 2015)

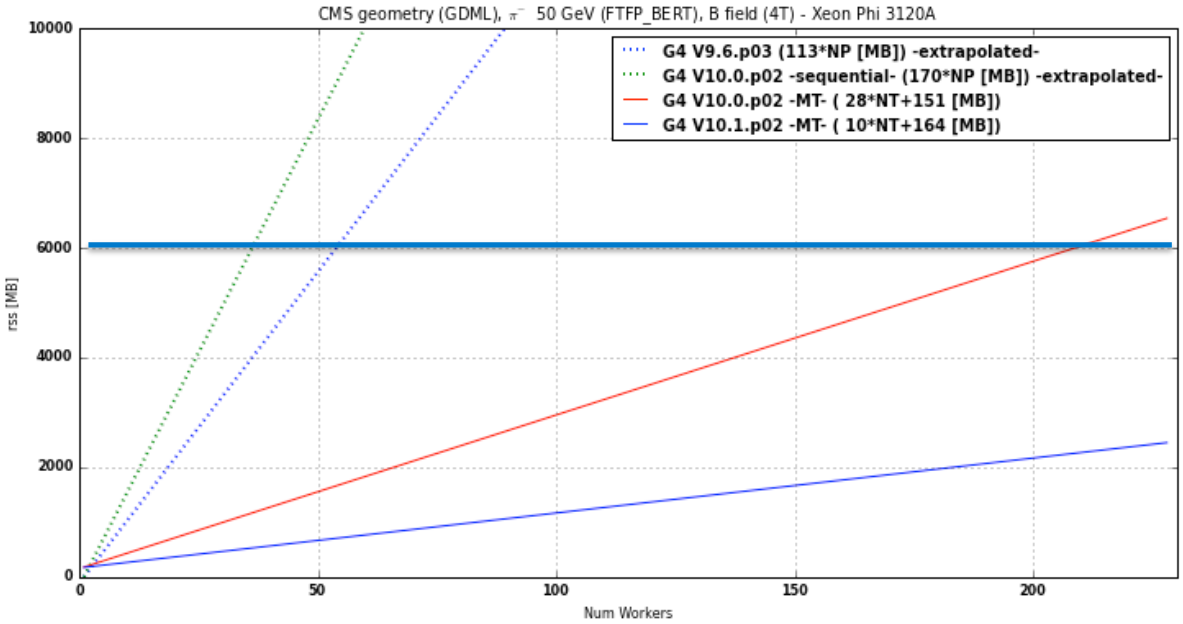
- Finalize VIS module
- Simplify integration of G4 with MPI and TBB


Results

Memory reduction

SLAC

Version	Intercept	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
10.1.p02	164 MB	10 MB



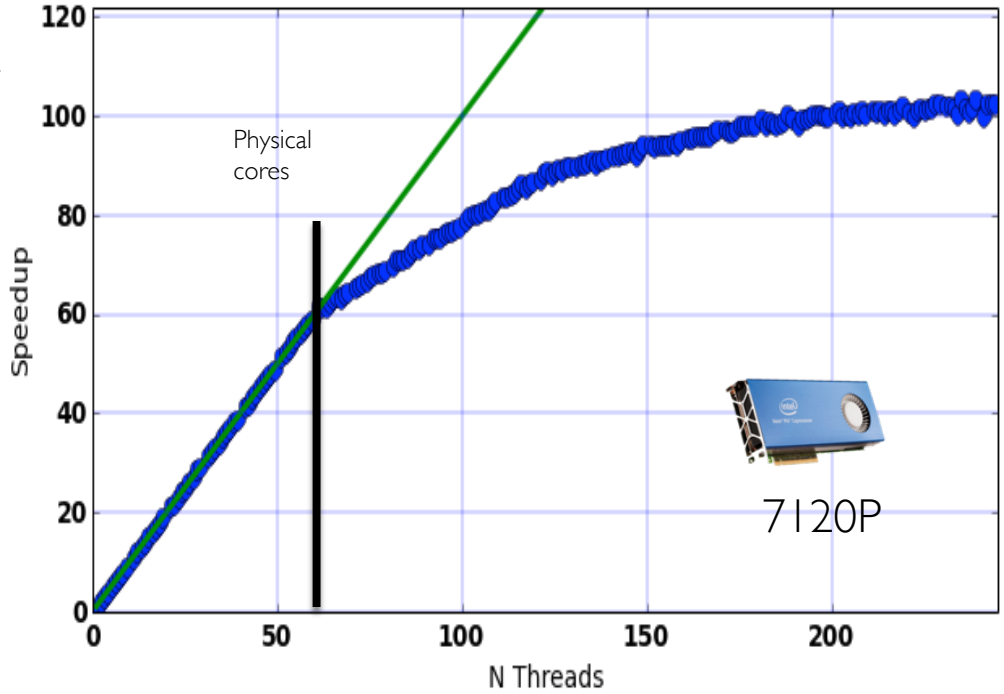
Memory limit for
Intel Xeon Phi 3120A

Speed-up

SLAC

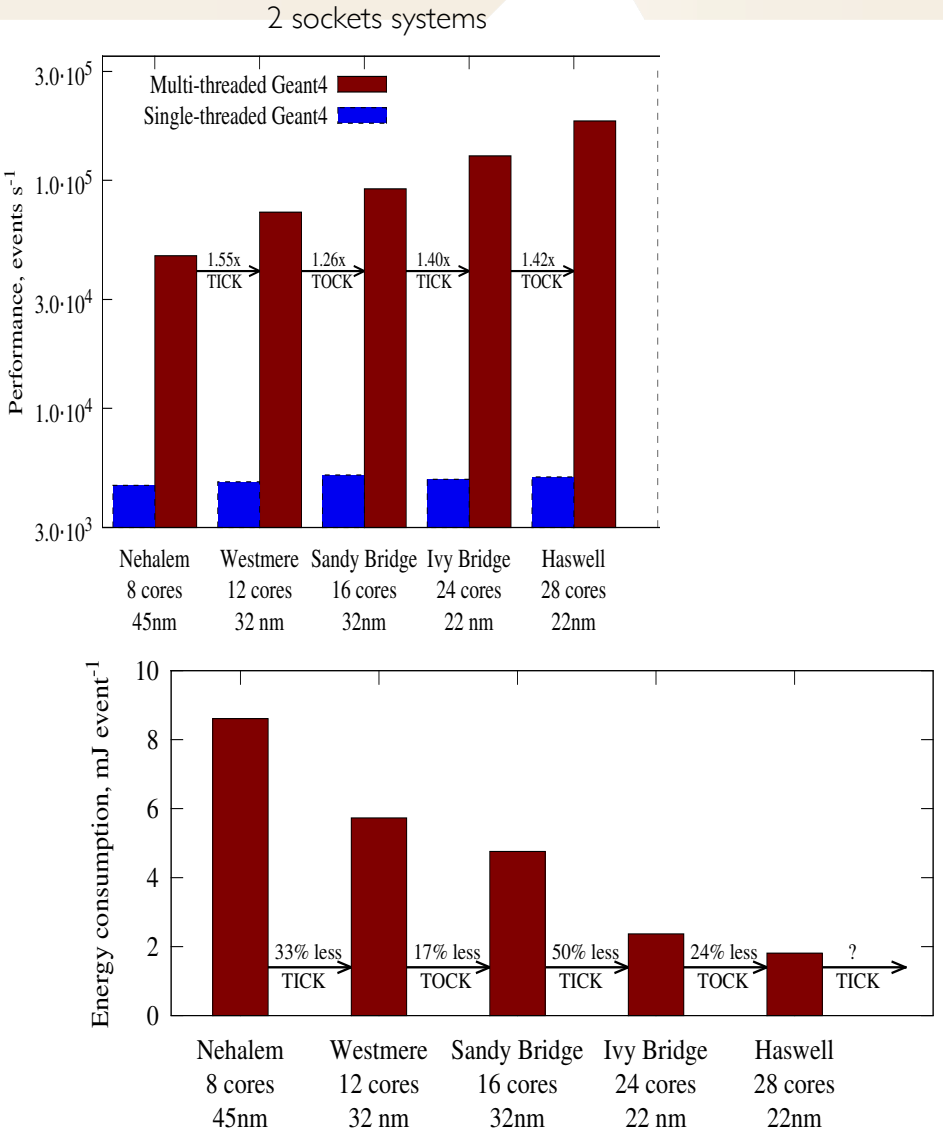
- Number of events/second is the most important metric for users
- **Very good linearity** (>93%) with the number of physical cores available
- Benefits from hyper-threading: ~30%
- Verified for different types of applications:
  - Medical physics applications
  - CERN Experiments setups

CERN Experiment geometry and physics



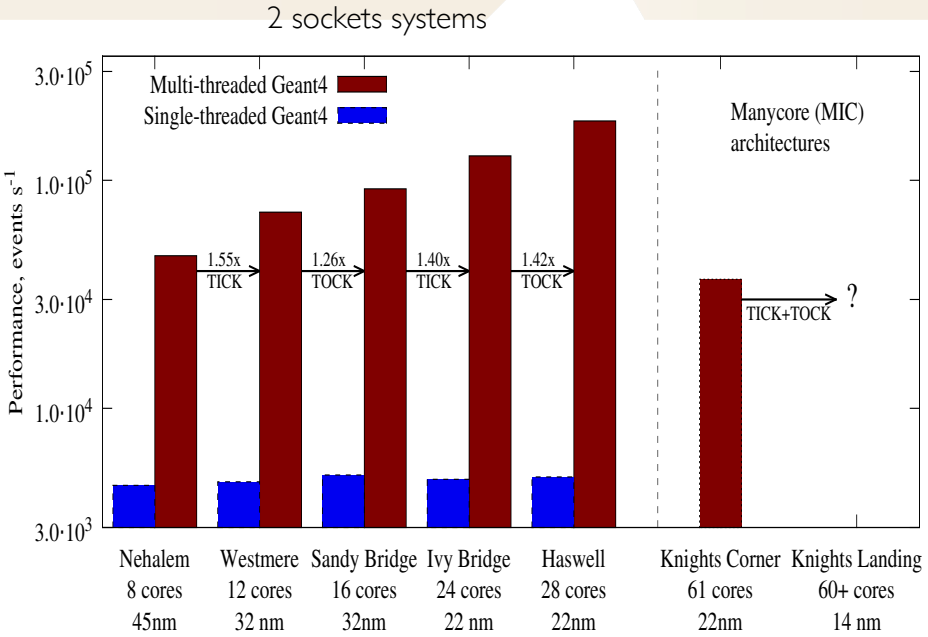
Architectures comparison

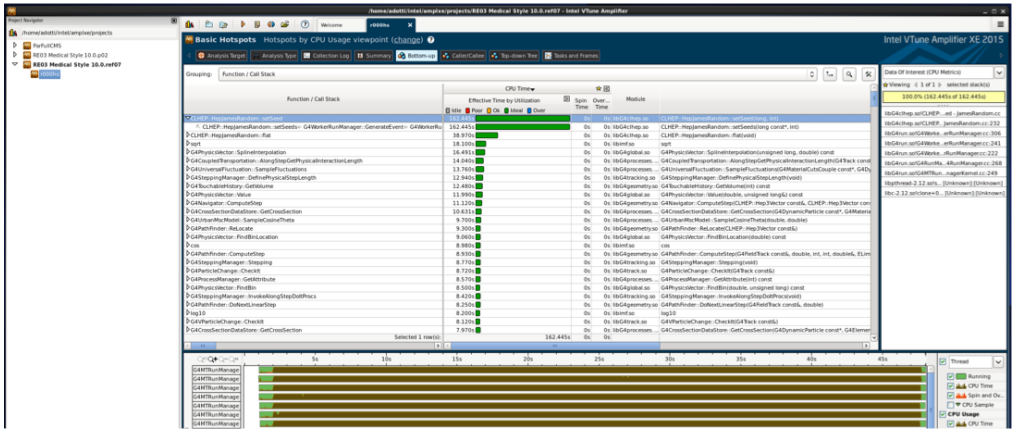
- Geant4 scales as expected on host CPU architectures
  - Multi-threading using all cores
- Energy efficiencies obtained:
  - Each new hw generation improves performance and decreases energy budget



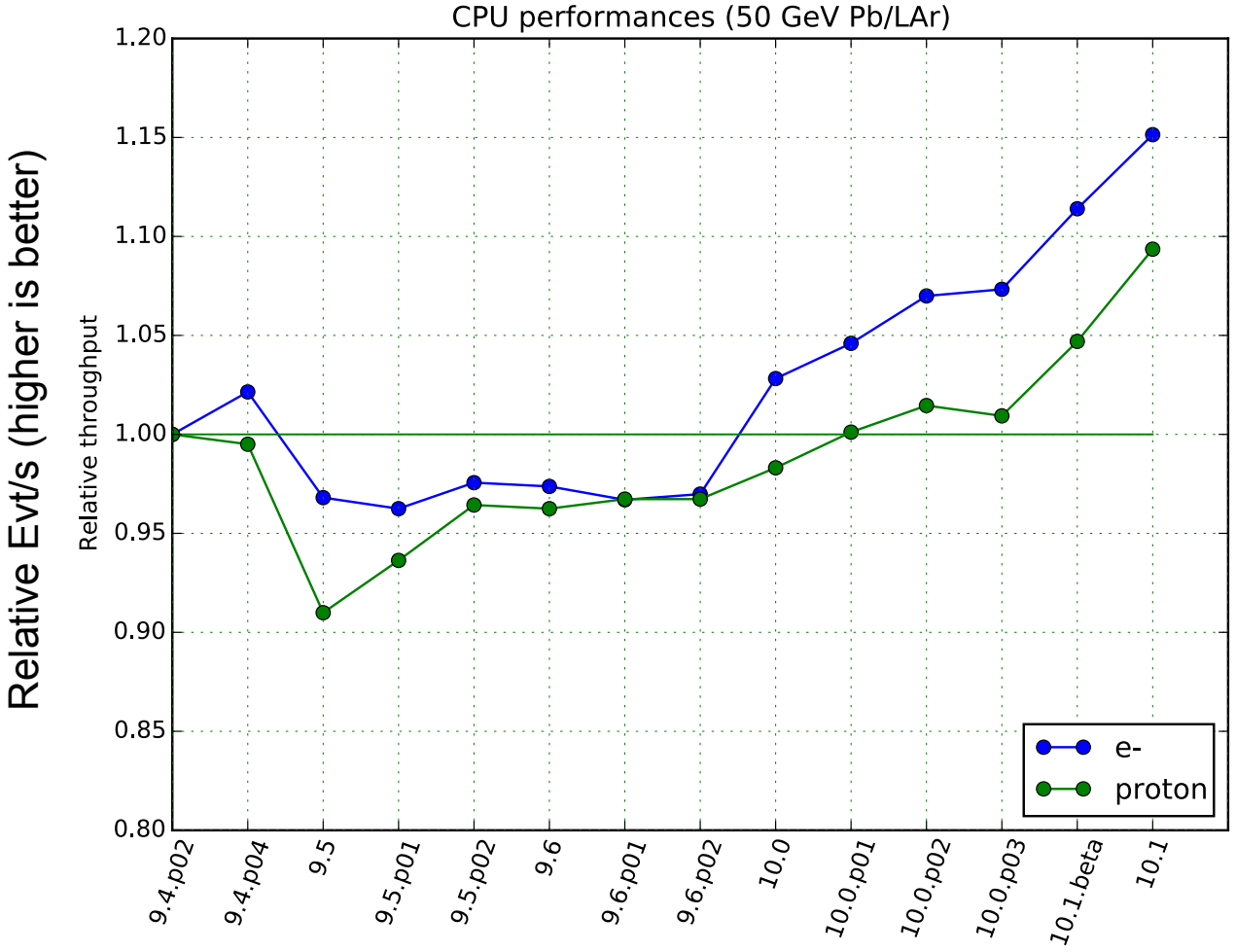
Architectures comparison

- MIC architecture: more work to be done
- From profiling analysis:
  - Sub-optimal use of vector registers
  - Need to optimize data container (e.g. cross-sections data-tables)



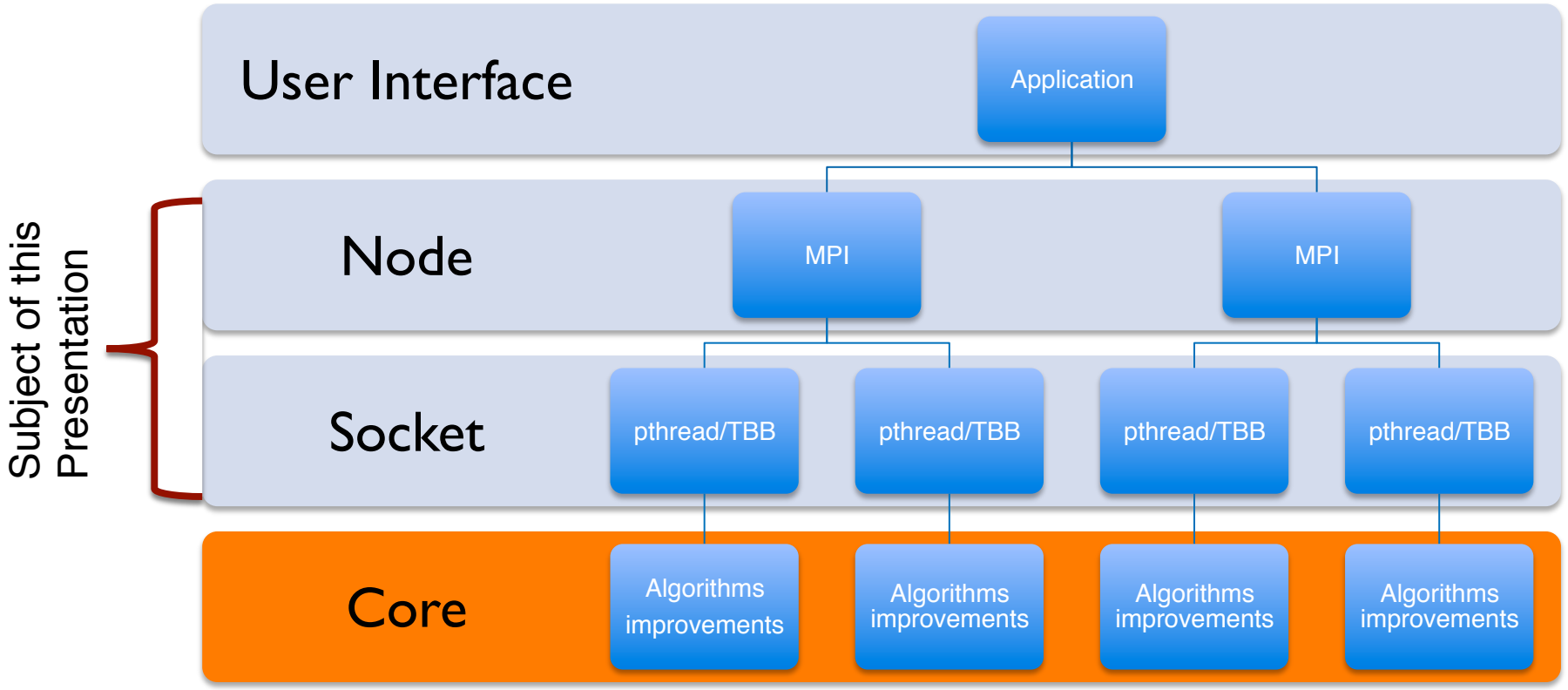


Benefits to sequential builds

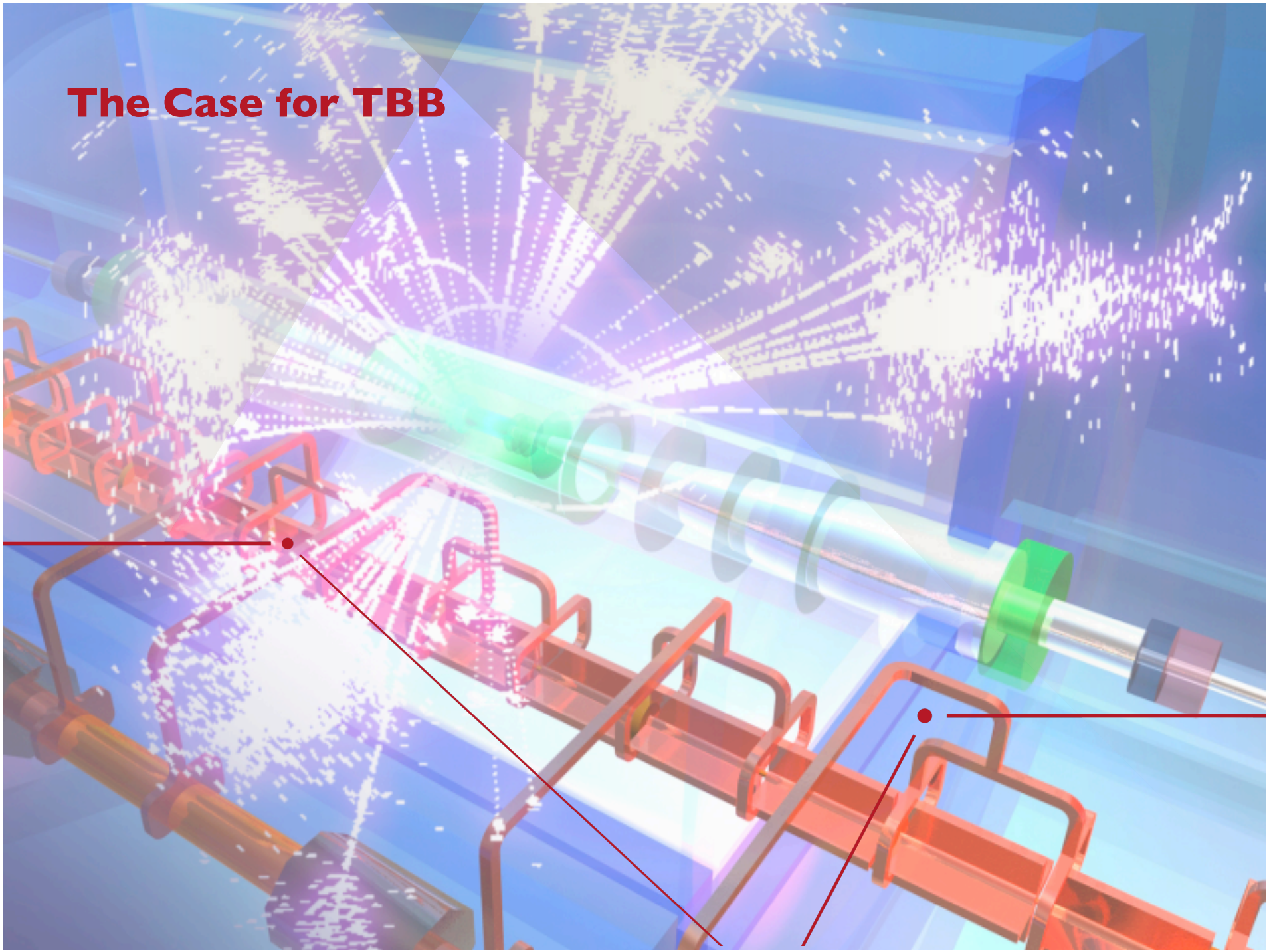


What is next

SLAC



The Case for TBB



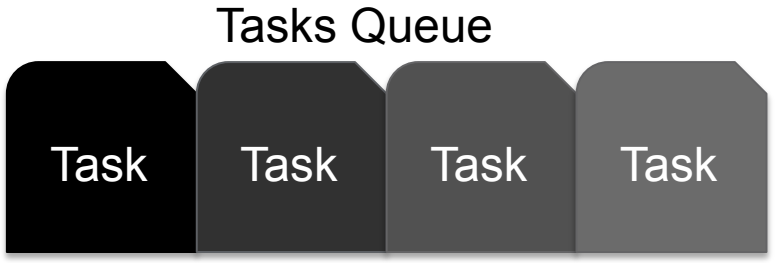
Intel Threading Building Blocks

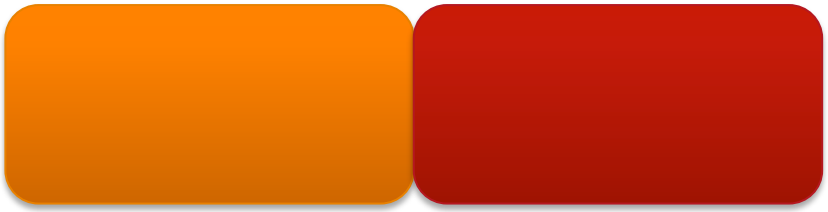
SLAC

- Parallelization **library** to express parallelism in form of tasks that can be executed concurrently
  - Intel product now available open-source (and free) for all systems and compilers
- **Hides the complexity of threading** to the user (responsible only to define the work unit and -if needed- their dependencies)
- For Geant4 job: each event (or group of) is one task
- For large software projects integration of many components libraries becomes simpler (strong interest from LHC experiments)

TBB in action

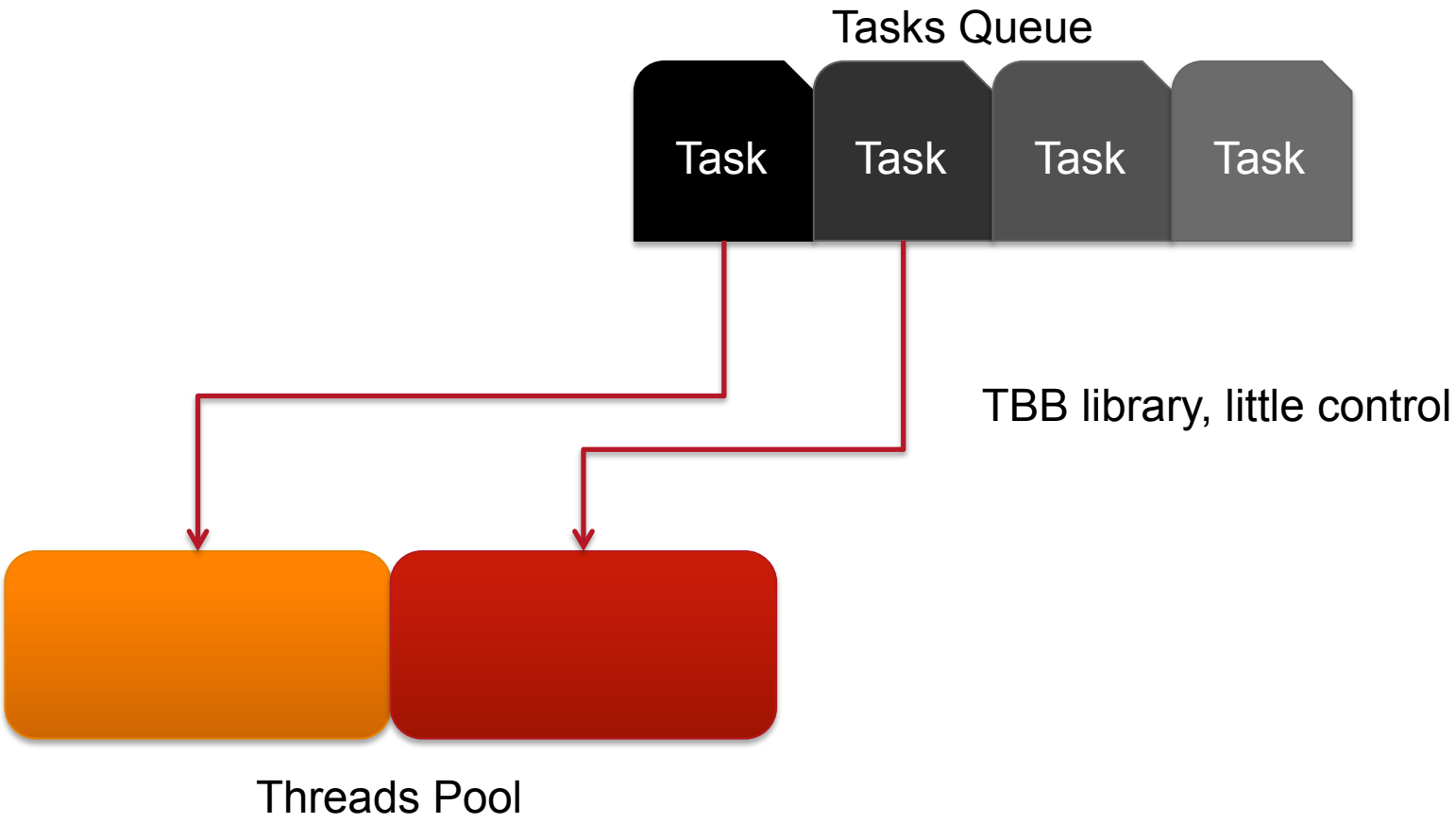
SLAC





Threads Pool

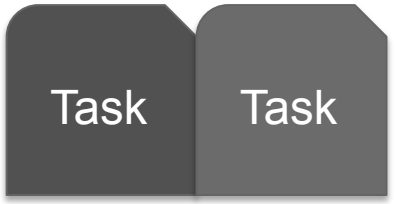
TBB in action



TBB in action

SLAC

Tasks Queue

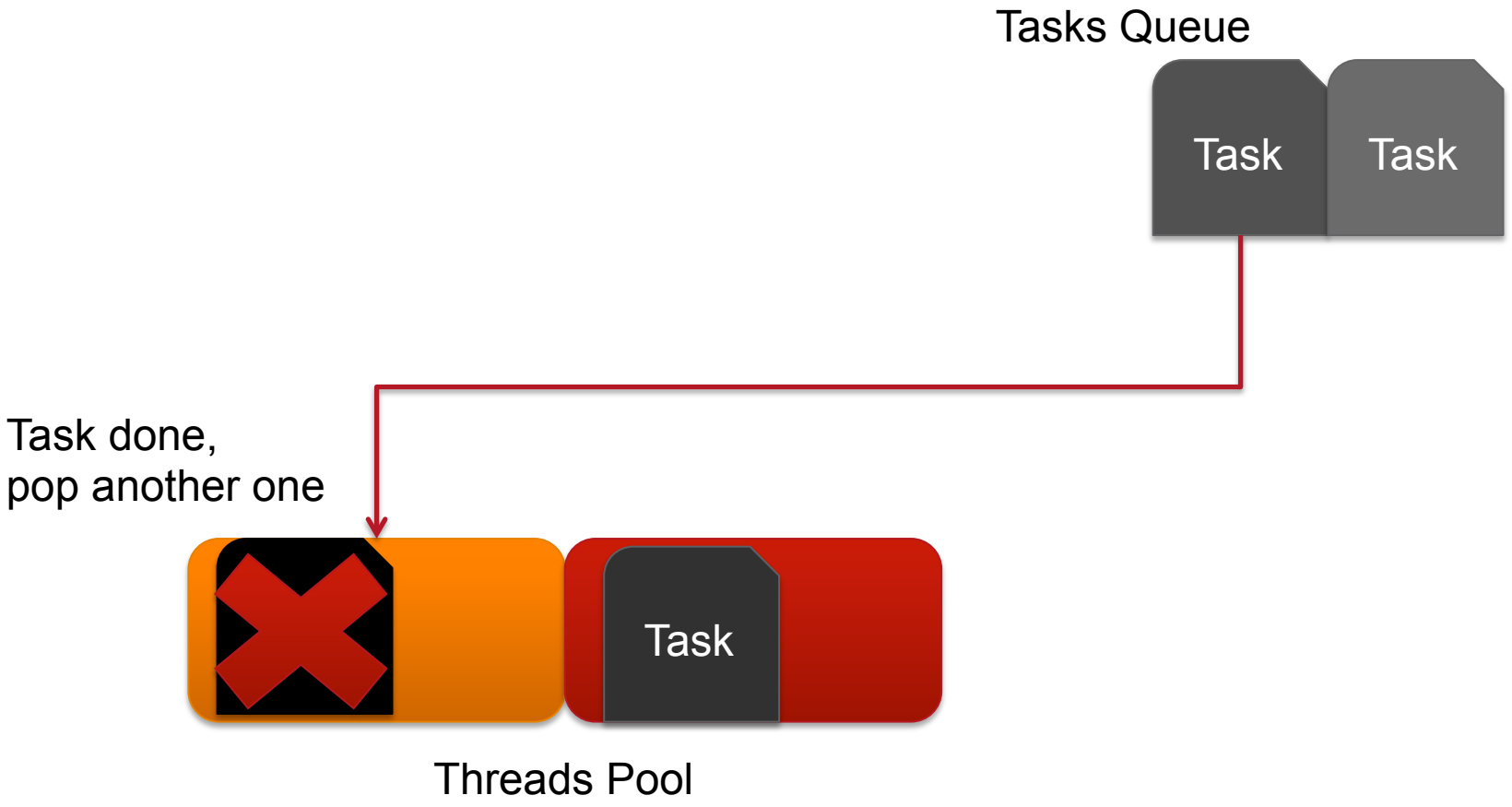


TBB library, little control

Threads Pool

TBB in action

SLAC



TBB Integration

SLAC

- Challenge:
  - TBB task model prefers "thread unaware" algorithms: i.e. workers should never get access to low-level threading details
  - Geant4 MT model requires direct control of Thread-Local-Storage
    - essential to perform lock-free code during simulation

- Latest versions of TBB introduced concept of *observer*: user-code that is executed once, by each low-level thread, before workers start
  - Create a Geant4 observer responsible of initializing TLS
  - **demonstrated in extended example (under development)**

- TBB integration becomes much easier:
  - CMS migrated to Geant4 MT, even without this, in a reasonable time
  - ATLAS already using this concept in athenaMT (under development) – with SLAC support

Decoupling of worker threads from master

SLAC

Current parallelization model:

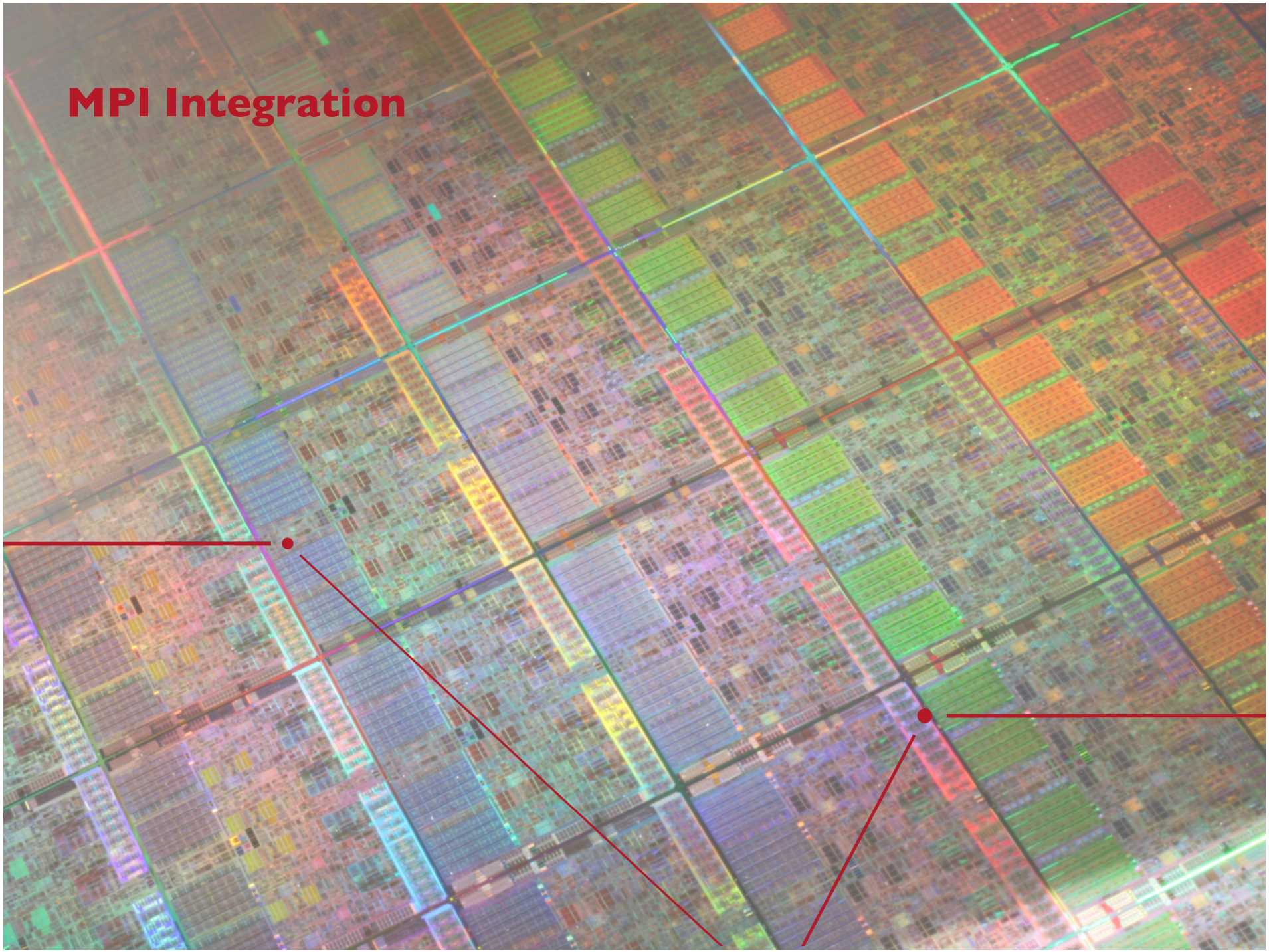
- threads organized as a **"static pool"**
- parallelization model is SPMD (single-program-multiple-data): threads are clones

Possible limitation for task-based frameworks (e.g. TBB)

- typically require worker decoupled from underlying threading model
- strong interest from LHC experiments (CMS, ATLAS and LHCb frameworks based to some extended on TBB: CMSSW, GaudiHive)

What we would like (work item for 2016):

- number of thread can vary in any moment during a job (constant during event loop to avoid synchronization primitives)
- any thread should be able to join/leave the workers pool


MPI Integration

Acknowledgemnts

For MPI integration in Geant4 we need to thank:

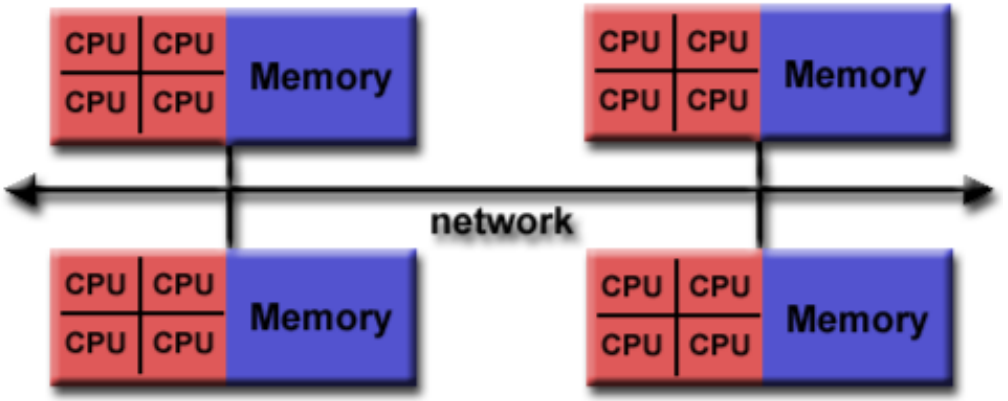
K. Murakami (KEK) as the original author of G4mpi library

G. Barrand, I.Hrivnacova (IN2P3) for the integration of analysis tools and G4mpi

What is MPI (Message Passing Interface)

SLAC

- **Distributed** memory parallelization framework
  - Clones of the job are started in parallel on a cluster or a multi-core machine
  - They cooperate on solving a problem exchanging messages
- An MPI application:
  - MPI is de-facto standard on large computing centers
  - Cannot achieve memory reduction
  - Is simpler to program w.r.t. a multi-threaded application
- **It is possible to combine MPI and MT**
  - With MPI scale across nodes
  - With MT scale across cores



MPI and Geant4

- MPI optimized for large (and/or frequent) messages

- Geant4 ranks have very little communication among them

- Still MPI is an attractive possibility for several reasons:

1. excellent support from a very large community
2. can use clusters or HPC systems where MPI is very common
3. preferred for asynchronous applications on Xeon Phi systems
4. very simple to use with Geant4

MPI Library and Example

SLAC

- Examples and runtime library in examples/extended/parallel/MPI
  - not built by default with Geant4 because requires external MPI package

- G4mpi library contains specialized managers and utilities (e.g. RNG handler, reducers)
- Compatible with many flavor of MPI

- Three examples show how to create parallel applications
  - two already available since version 9.1
  - substantially improved

- Substantially improved for 10.1 and more coming in 10.2
  - Better integration with CMake building system
  - Reducer for physics data

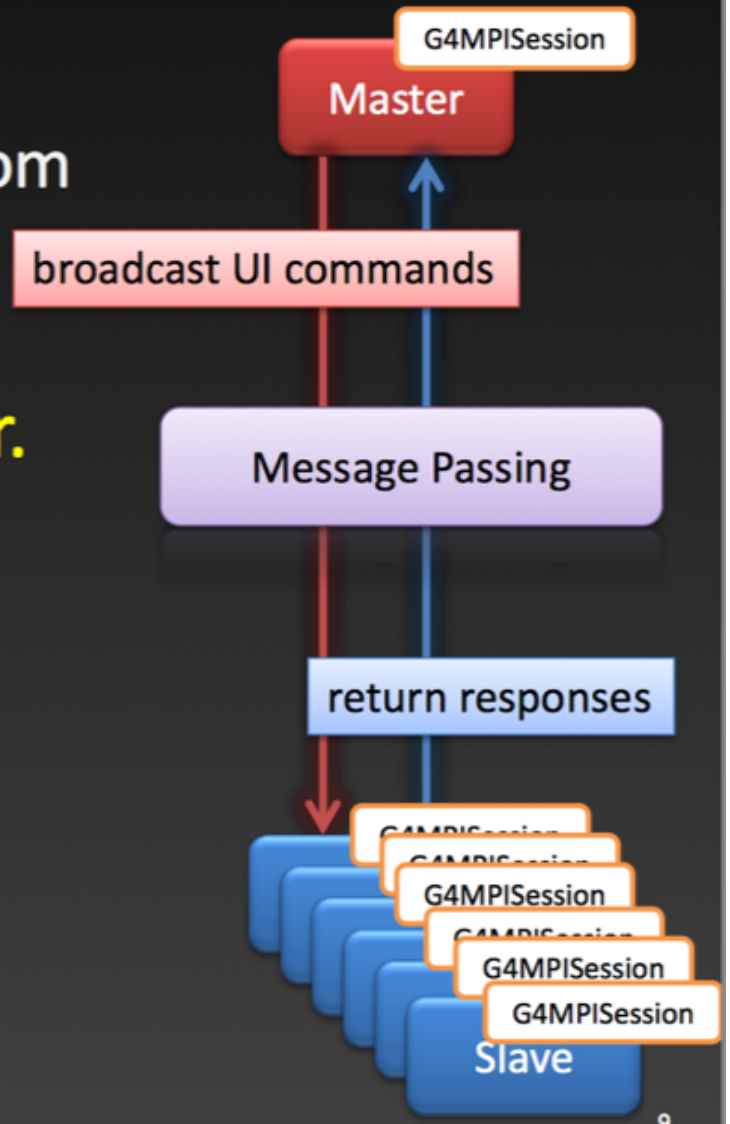
Implementation of G4MPI

Simple master-slave model

- *UI commands* are broadcasted from the master to slaves.

Implemented in "Interface" layer.

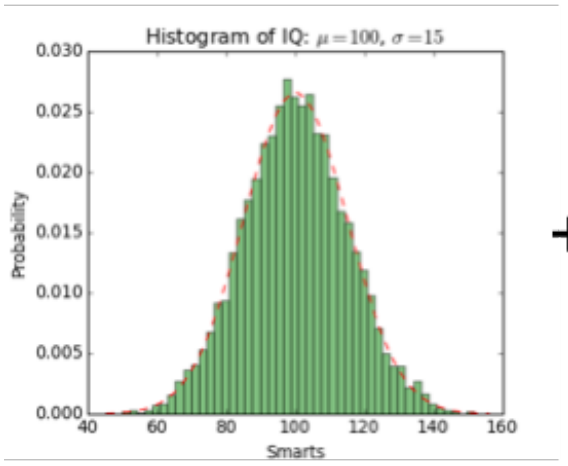
- There is no needs of special RunManager etc. for parallel.



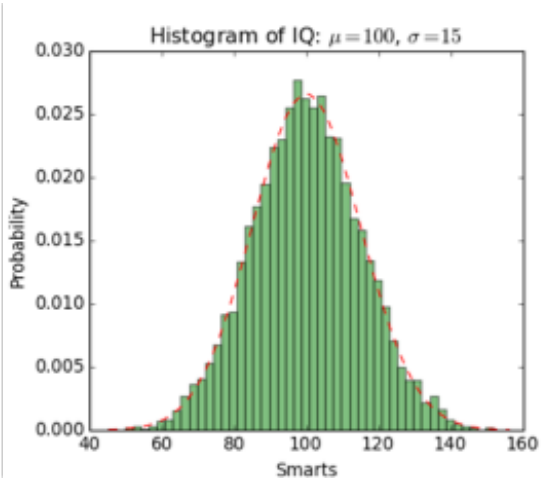
Results Reduction (new in 10.1)

SLAC

- **Similarly to what is done in MT** analysis quantities are merged at the end of the run
  - Ranks send back to master their partial results
  - Masters sums up everything in a single output
- Support for: command line scoring, G4Run user-data, g4analysis histograms



+



= MPI message: bins content and decorators
Same name objects are merged

Status

SLAC

- Reductions for MPI are available in Version 10.1, but…
  - optimization of communication pattern is underway: systems with many ranks can suffer important overheads
  - improvement of G4mpi library (some small user-code changes expected in 10.2)

- Promising development, we are also learning in the process, feedback is welcome

Slide from P. Calafiura (LBL)

A complex example: MPI + MT + …

SLAC

Cori – NERSC's next supercomputer system

NERSC will be installing a 30-petaflop/s KNL-based Cray system in mid-2016

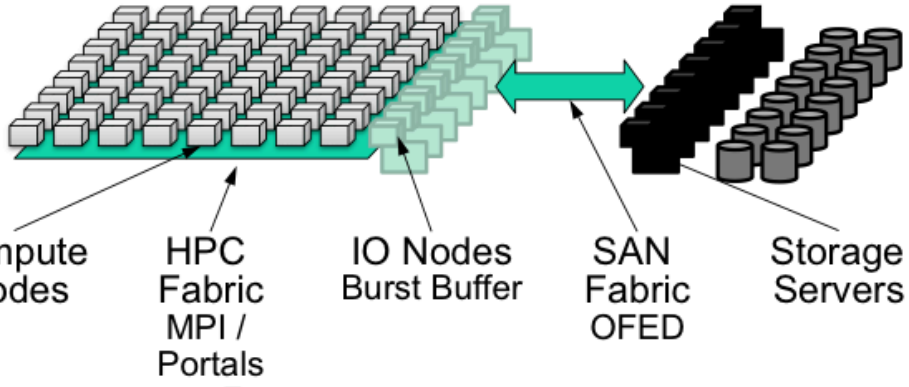
Over 9,300 single-socket KNL nodes

72 cores per node. 4 hardware threads each

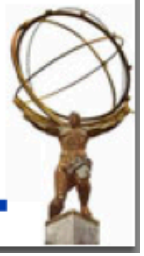
Better performance per watt than KNC and 3X single-thread performance

64-128 GB of DRAM memory per node

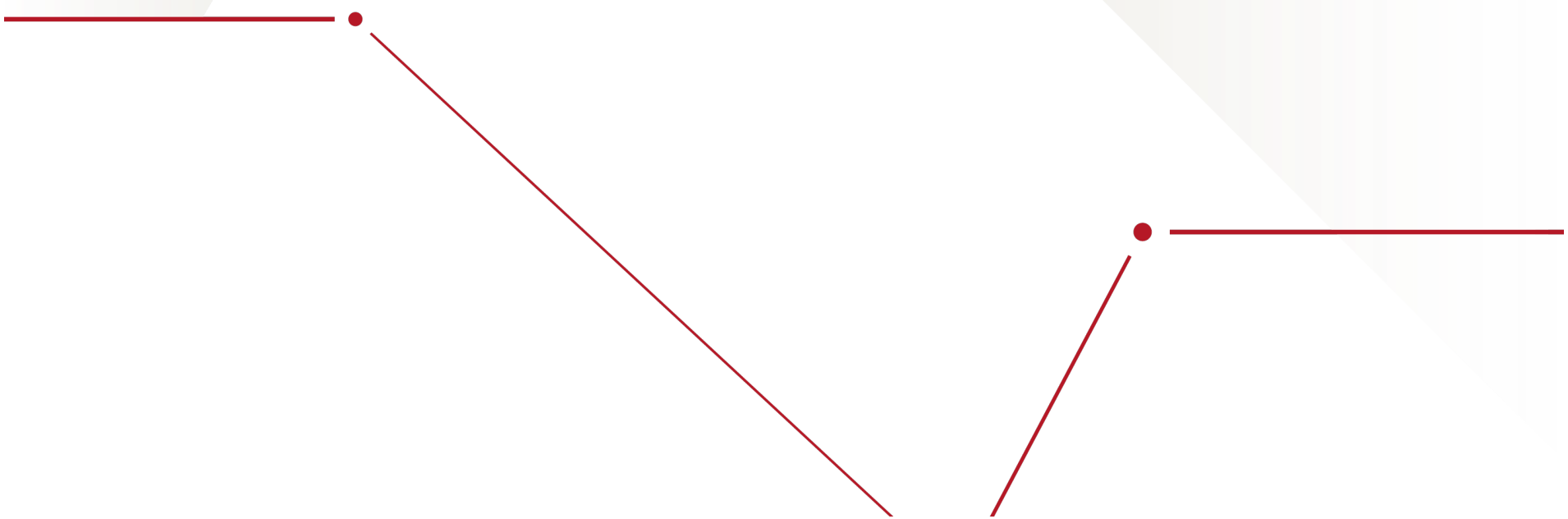
Gerty Cori



28PB HDD on storage servers with ~1PB SDD Burst Buffer 1-2 TB/s



Conclusions



Parallelization in Geant4

SLAC

Key driving forces:

- use **well established standards** and avoid "reinventing the wheel" (e.g. pthreads, MPI, …)

- **minimal API changes** to simplify user-code migration
  - we believe multi-threading is an excellent example of this success

- **iteratively improve** CPU and memory performances

- introduce new functionalities in close dialogue with users: MPI used by users since some years, now time to provide common tools

Parallelization in Geant4

SLAC

Geant4 MT has been adopted by many communities, including large experiments:

- we have successfully met our goals (memory reduction, scalability)

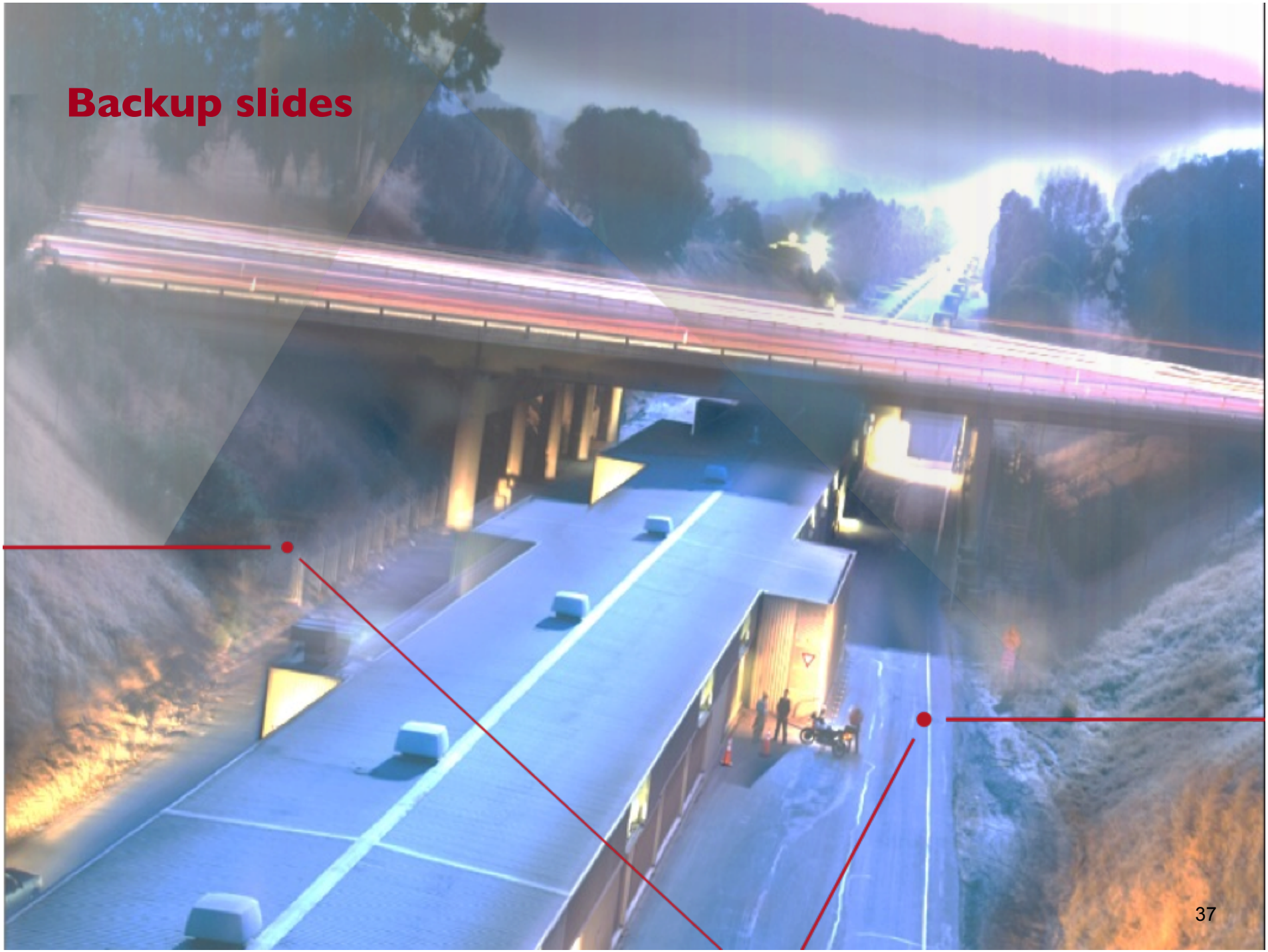
- focus is now shifting towards integration with external parallelization frameworks and improving algorithm performances

A word on future activities

SLAC

- Geant4 Version 10.2 will be based on C++11 standard
  - users need a recent compiler and a system supporting this standard
  - we'll migrate from pthreads to std::thread
    - it will simplify maintenance of our code and possibly allow for the porting of MT to Windows (Ver. 10) systems
  - some indications that G4 code could be faster

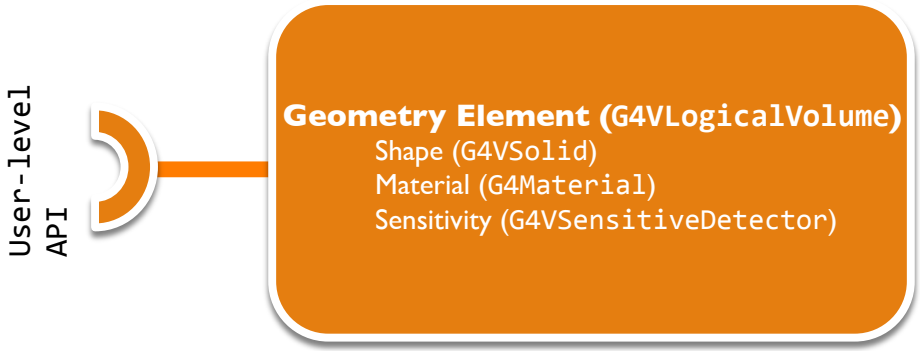
- We plan to evaluate (>2016) other technologies: Transactional Memory, OpenMP 4.0 and CilkPlus are very interesting options
  - as always we want to keep simple user-code migration


Backup slides

Thread Safety and memory usage reduction

SLAC

- Design: lock-free code during event-loop
  - 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
  - allows for **minimal API change**

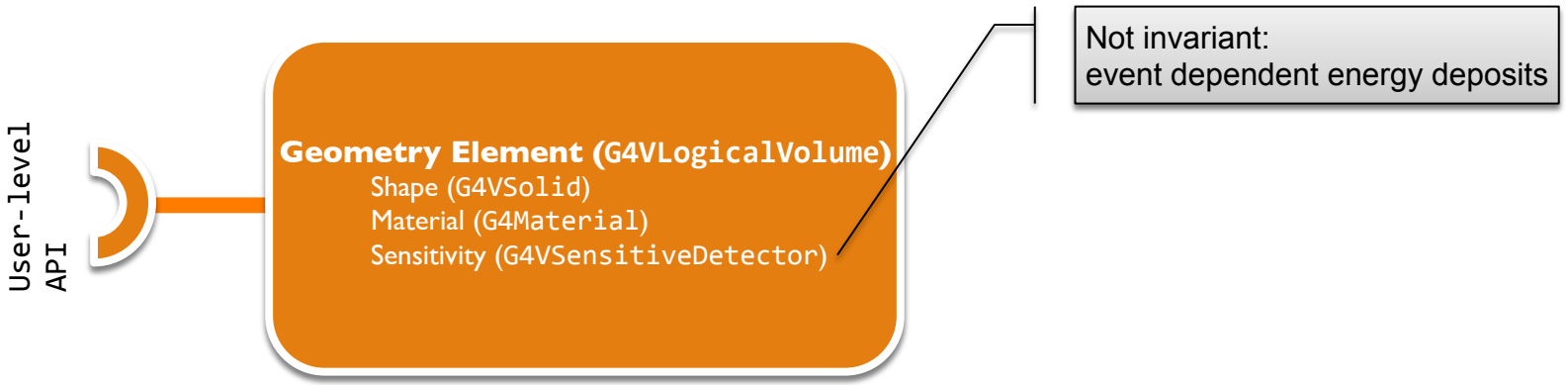


http://dx.doi.org/10.1051/snamc/201404213

Thread Safety and memory usage reduction

SLAC

- Design: lock-free code during event-loop
  - 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
  - allows for **minimal API change**

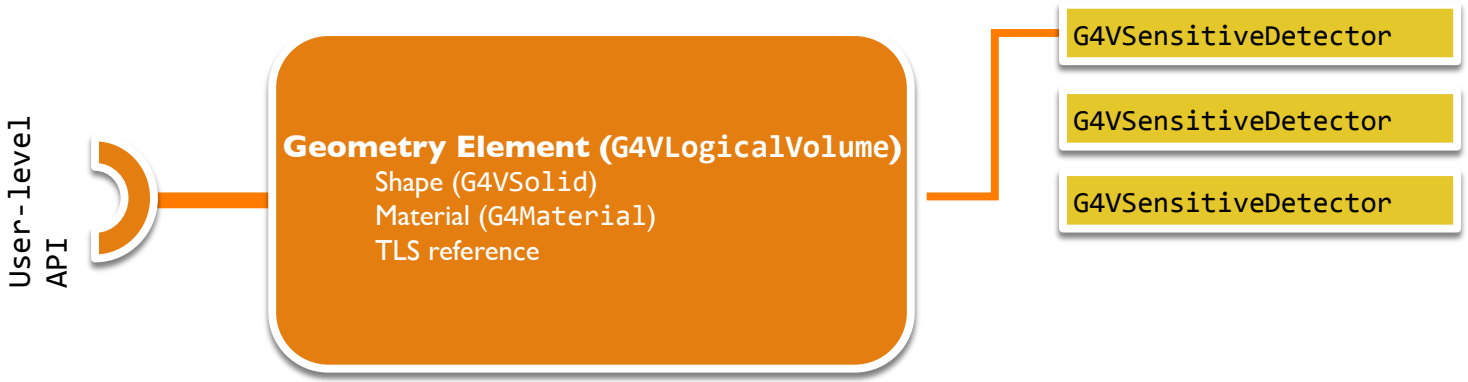


http://dx.doi.org/10.1051/snamc/201404213

Thread Safety and memory usage reduction

SLAC

- Design: lock-free code during event-loop
  - 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
  - allows for **minimal API change**

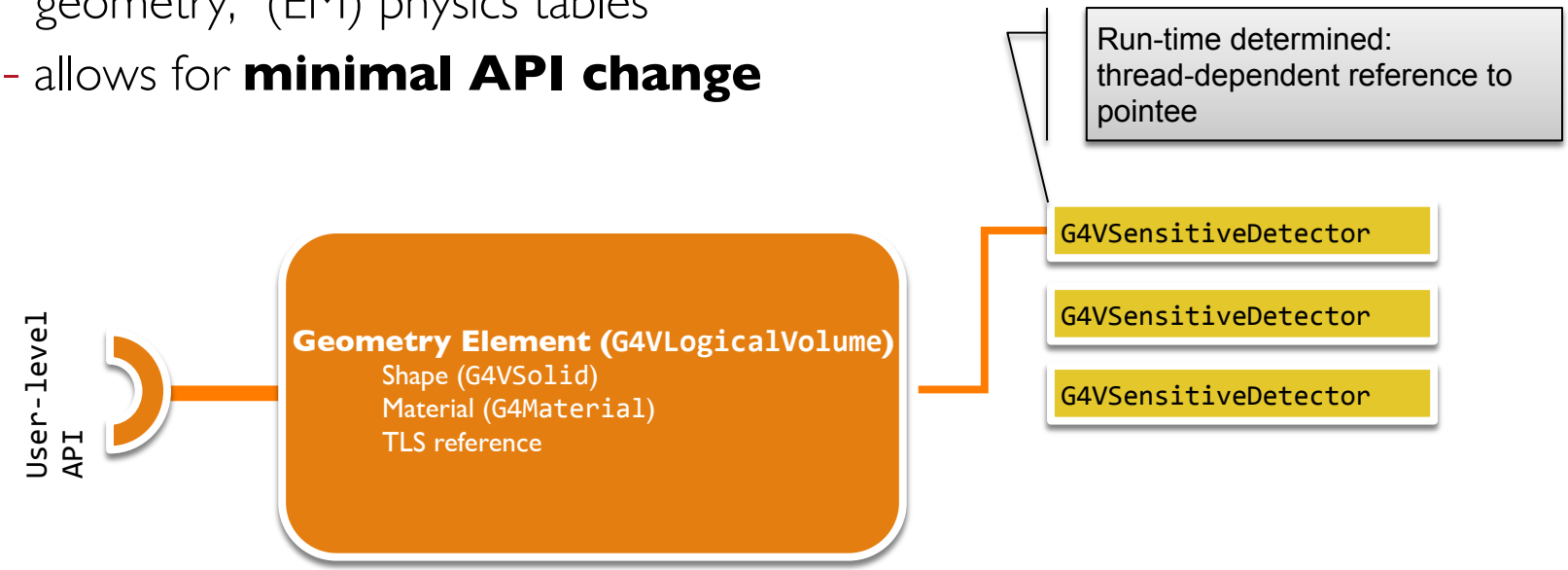


http://dx.doi.org/10.1051/snamc/201404213

Thread Safety and memory usage reduction

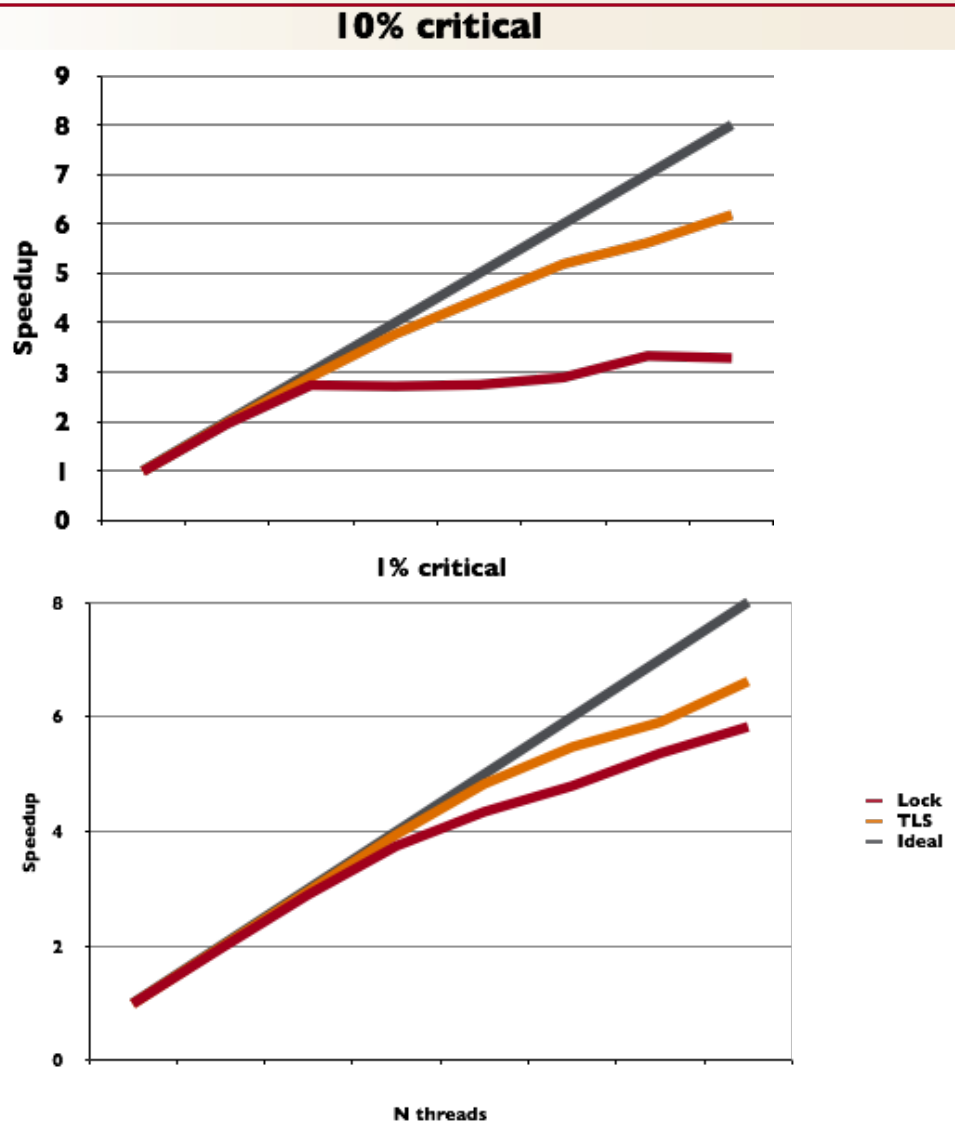
SLAC

- Design: lock-free code during event-loop
  - 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
  - allows for **minimal API change**



http://dx.doi.org/10.1051/snamc/201404213

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)
  - Improved support in C++11 standard
- Drawback: only simple data types for static/global variables can be made TLS

Visualization with MT

SLAC

- Real-time visualization poses some challenges in a MT application
  - visualization / workers synchronization
- Geant4 solution: adopt **producer/consumer paradigm**
  - workers produce events: pushed in a shared queue
  - independent visualization thread consumes (pulls) from queue
- Queue has a maximum allowed size
  - Current policy: back-pressure pauses worker threads
  - Under design: "sink" that drops events when too many are pushed
  - **Both will be available to the user**