G4 Ver 10.00.p01

# Multithreading 2

## SLAC Geant4 Tutorial 2014

Monday March 3 2014 - Jen-Hsun Huang Engineering Center Stanford University

A. Dotti (adotti@slac.stanford,edu)

# Note

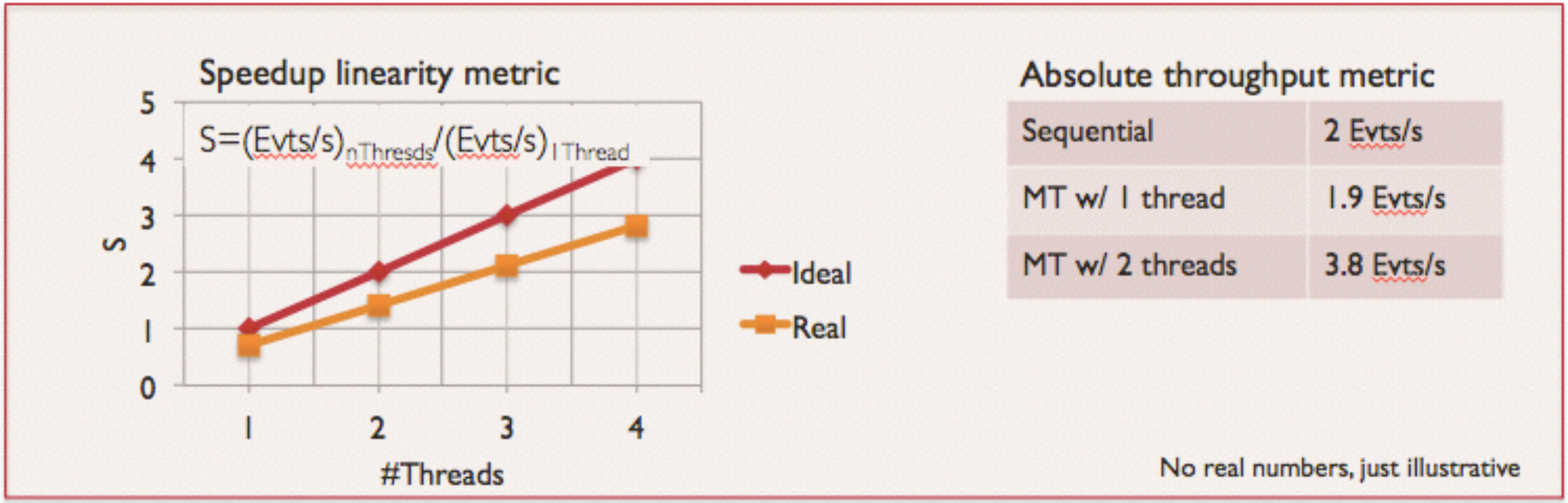
This is the second part of Monday MT talk

- What is thread-safety: a simple example
- ThreadLocalStorage and split-class mechanism
- Some results
- Extending Threading model
- External parallelism frameworks: MPI, TBB
- Geant4 on Intel Xeon Phi

# The challenges of MT: thread-safety

# Definitions

- **Design to minimize changes in user-code**
  - Maintain API changes at minimum
- Focus on **linearity of speed-up** (w.r.t. #threads) is the most important metric
- Enforce use of **POSIX standards** to allow for integration with user preferred parallelization frameworks (e.g. MPI, TBB, …)



$S=(Evts/s)_{nThresds}/(Evts/s)_{1\,Thread}$

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

# Thread safety a simple example

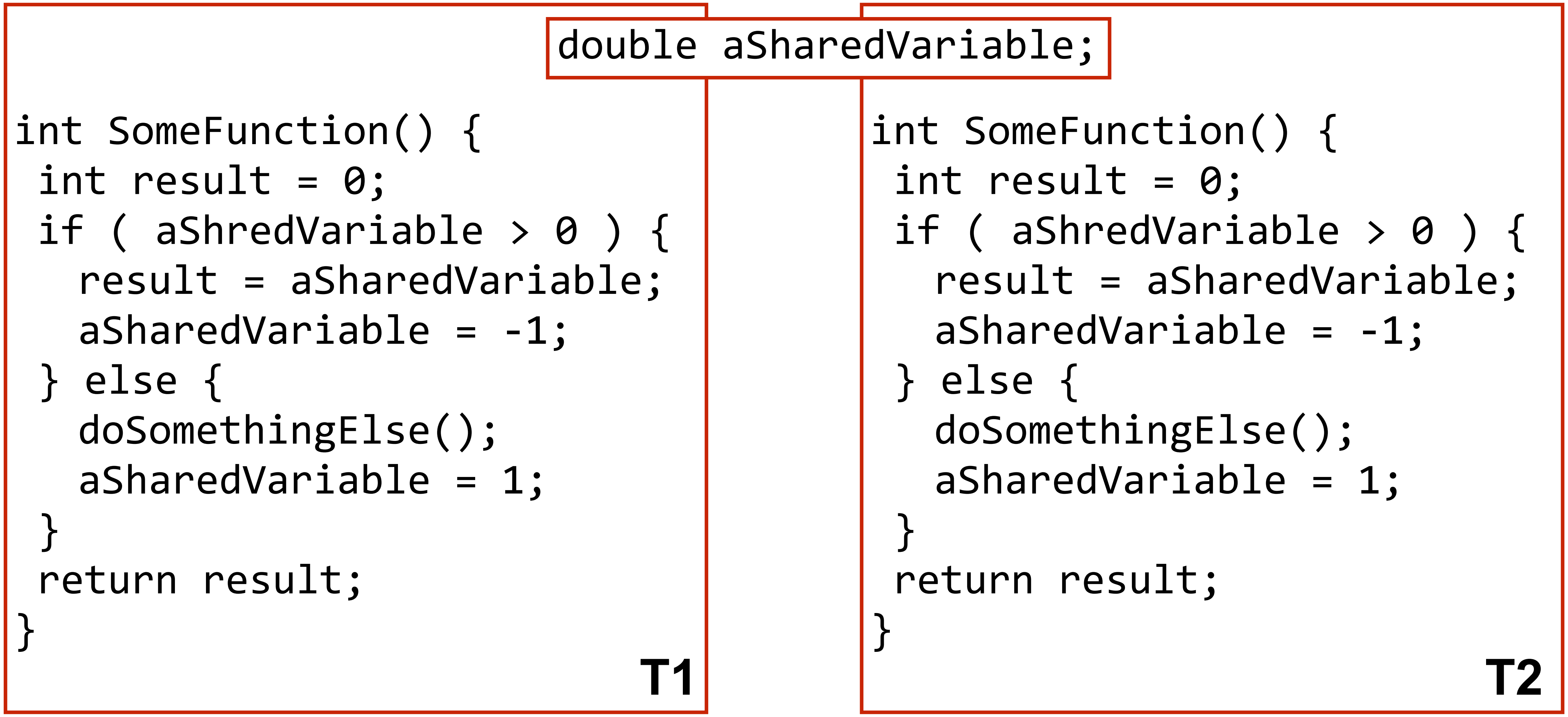
Consider a function that reads and writes a shared resource (a global variable in this example).

```
double aSharedVariable;

int SomeFunction() {
 int result = 0;
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 return result;
}
```

# Thread safety a simple example

Now consider two threads that execute at the same time the function. Concurrent access to the shared resource



```
double aSharedVariable;
```

```
int SomeFunction() {
 int result = 0;
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 return result;
}
```

**T1**

```
int SomeFunction() {
 int result = 0;
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 return result;
}
```

**T2**

# Thread safety a simple example

result is a local variable, exists in each thread separately not a problem, T1 starts arrives **here** and then is halted

```
double aSharedVariable;
```

```
int SomeFunction() {
 int result = 0;
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 return result;
}
```

**T1**

```
int SomeFunction() {
 int result = 0;
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 return result;
}
```

**T2**

# Thread safety a simple example

Now T2 starts and arrives **here**, the shared resource value is not yet updated, what is the expected behavior? what is happening?

```
double aSharedVariable;
```

```
int SomeFunction() {
 int result = 0;
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 return result;
}
```

**T1**

```
int SomeFunction() {
 int result = 0;
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 return result;
}
```

**T2**

# Thread safety a simple example

Use mutex / locks to create a barrier. T2 will not start until T1 reaches UnLock

Significantly reduces performances (general rule in G4, not allowed in methods called during the event loop)

http://en.wikipedia.org/wiki/Lock_(computer_science)

```
double aSharedVariable;
```

```
int SomeFunction() {
 int result = 0;
 Lock(&mutex);
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 Unlock(&mutex);
 return result;
}
```

**T1**

```
int SomeFunction() {
 int result = 0;
 Lock(&mutex);
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 Unlock(&mutex);
 return result;
}
```

**T2**

# Thread safety a simple example

- Do we really need to share aSahredVariable?
  - if not, minimal change required, each thread has its own copy
  - Simple way to "transform" your code (but very small cpu penalty, no memory usage reduction)
- General rule in G4: do not use unless really necessary

```
double __thread aSharedVariable;

int SomeFunction() {
 int result = 0;
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 return result;
}
```

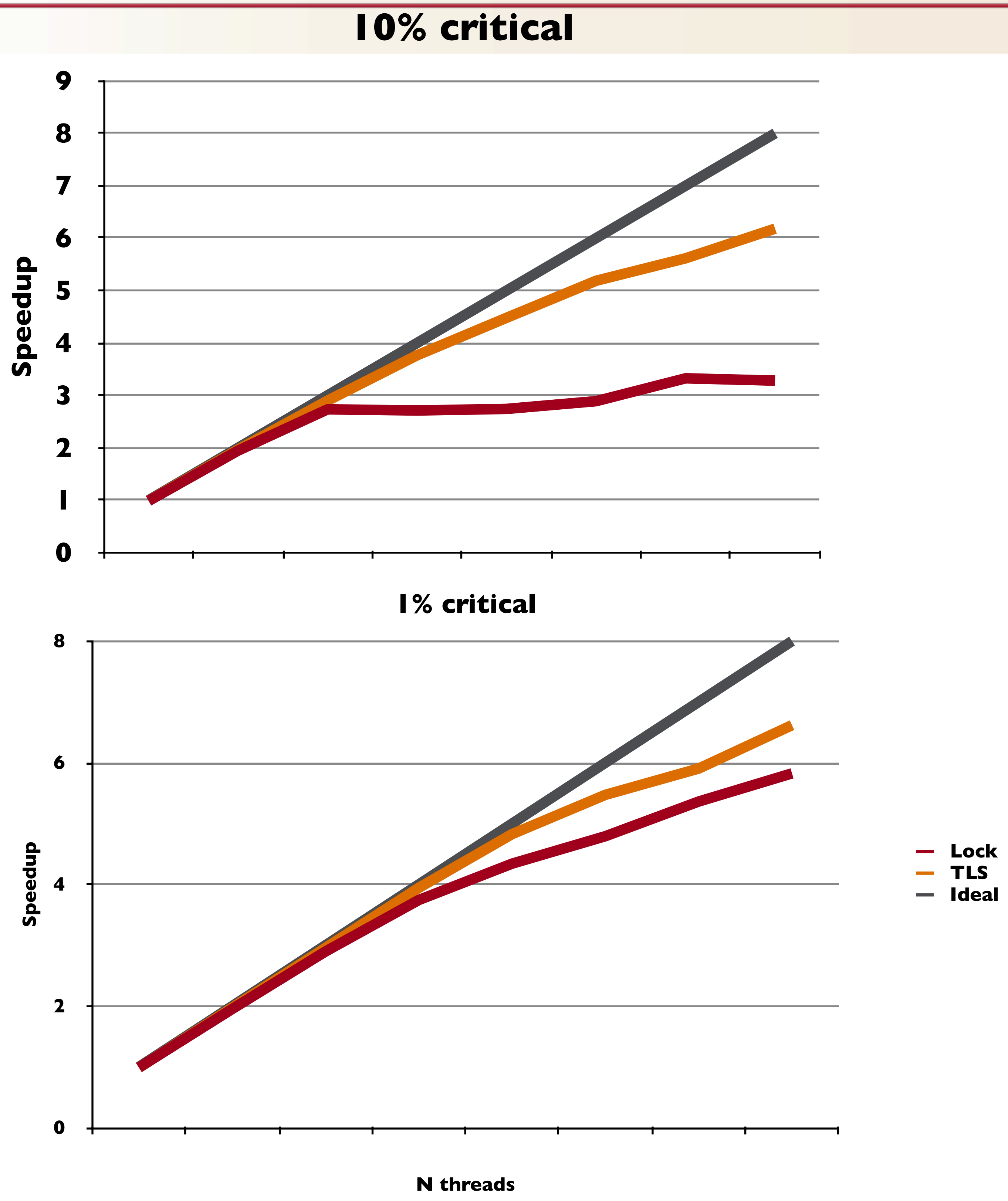
**T1**

```
double __thread aSharedVariable;

int SomeFunction() {
 int result = 0;
 if ( aShredVariable > 0 ) {
   result = aSharedVariable;
   aSharedVariable = -1;
 } else {
   doSomethingElse();
   aSharedVariable = 1;
 }
 return result;
}
```

**T2**

# Thread Local Storage



**NB**: results obtained on toy application, not real G4

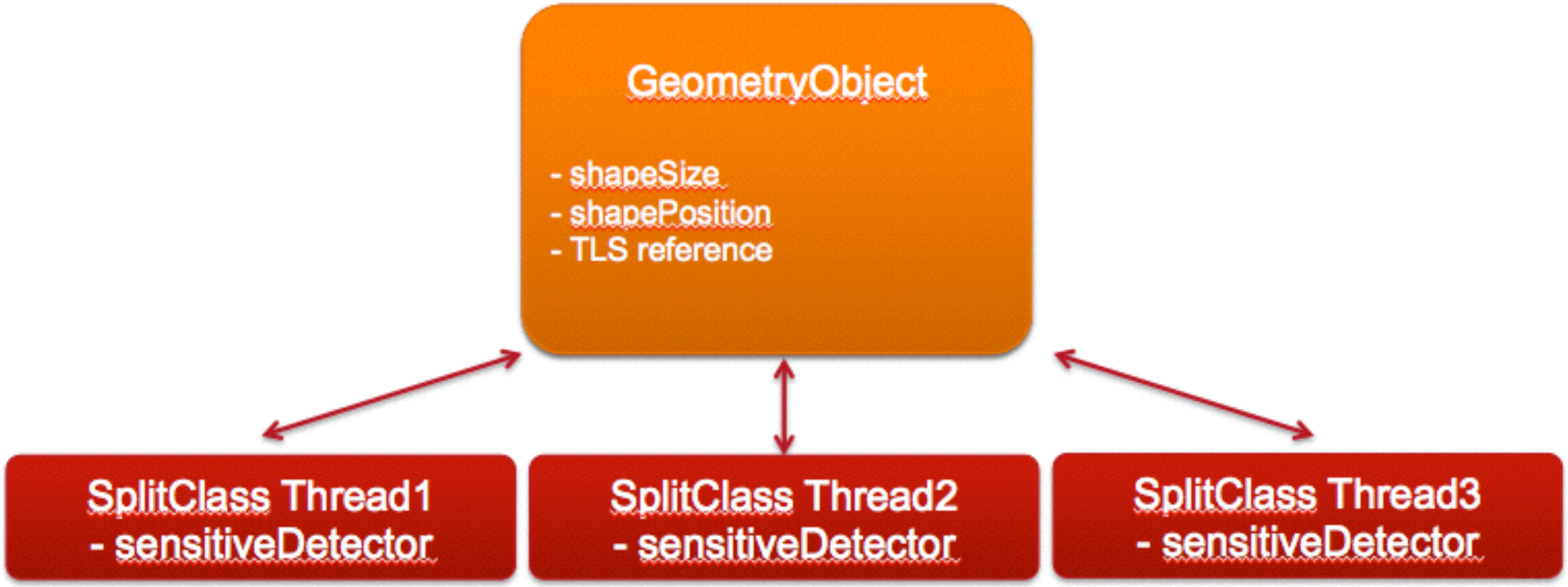
- Each (parallel) program has sequential components
- **Protect access to concurrent resources**
- Simplest solution: use mutex/lock
- TLS: each thread has its own object (no need to lock)
  - **Supported by all modern compilers**
  - "just" add `__thread` to variables

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

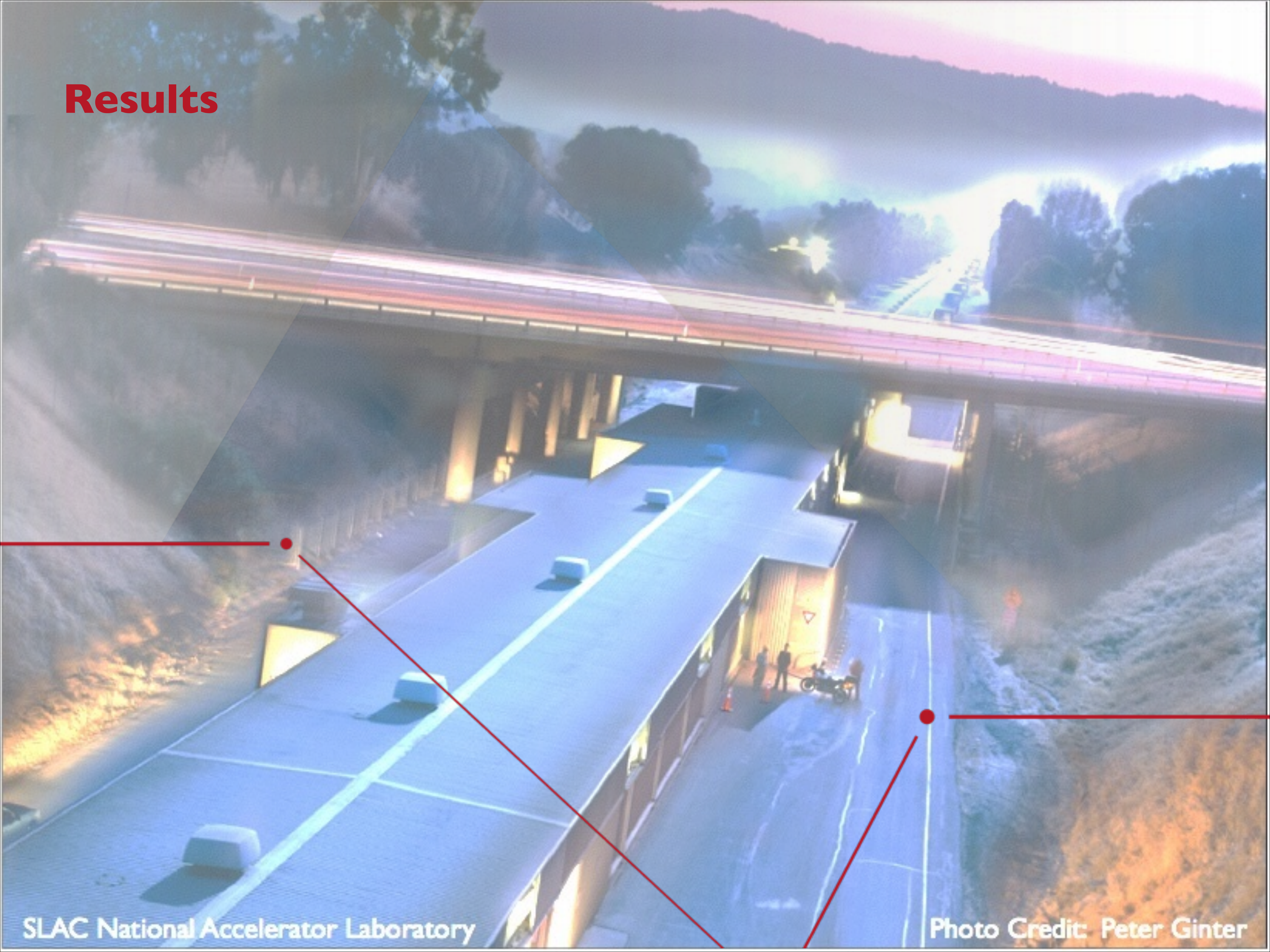
  - Improved support in C++11 standard
- Drawback: increased memory usage and small cpu penalty (currently 1%), only simple data types for static/global variables can be made TLS

# 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



## Results



SLAC National Accelerator Laboratory

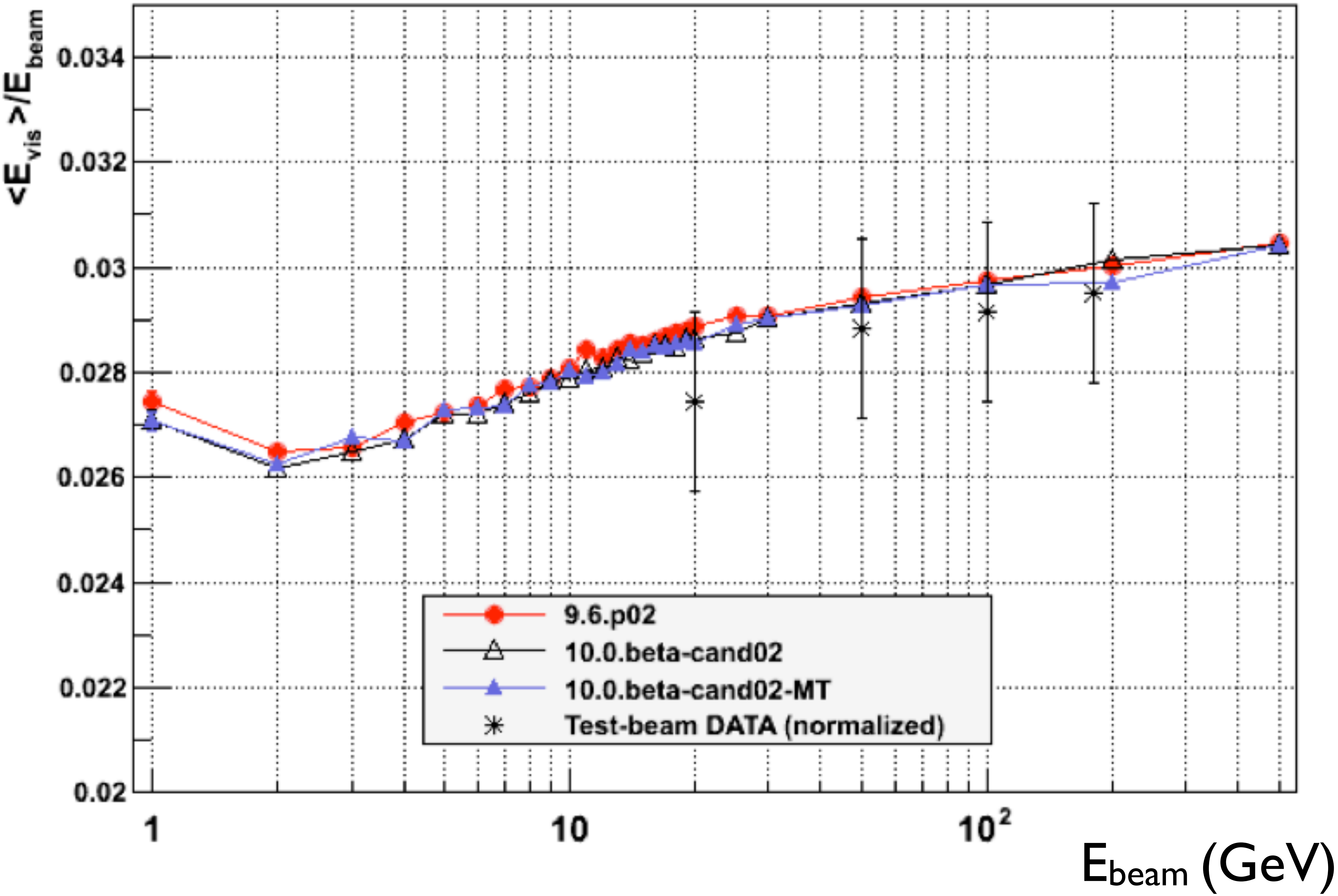
Photo Credit: Peter Ginter

# 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!

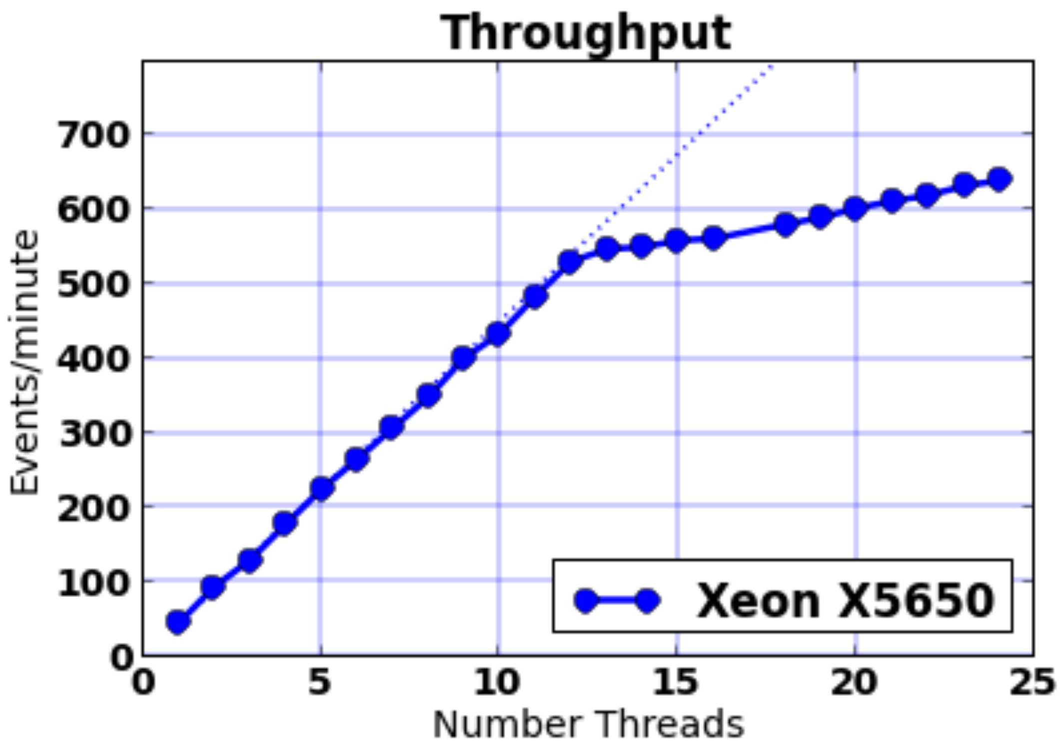
- Simulation results is equivalent between Sequential and MT

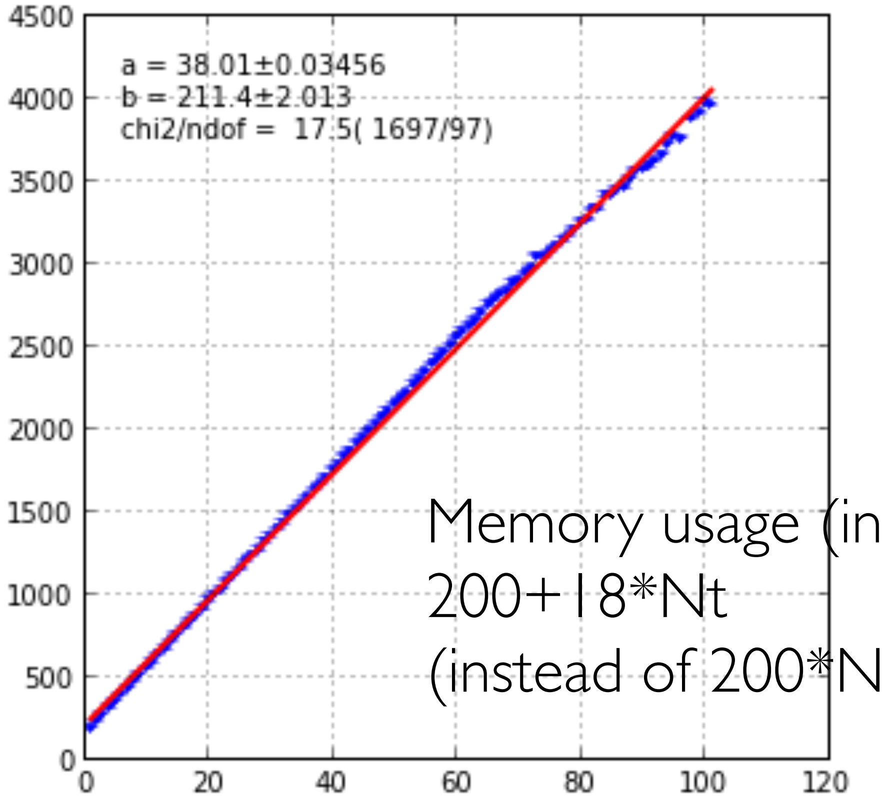
# $\pi^-$ on Fe/Scintillator sampling calorimeter



G4 Ver 10.00.p01

# CPU / Memory performances

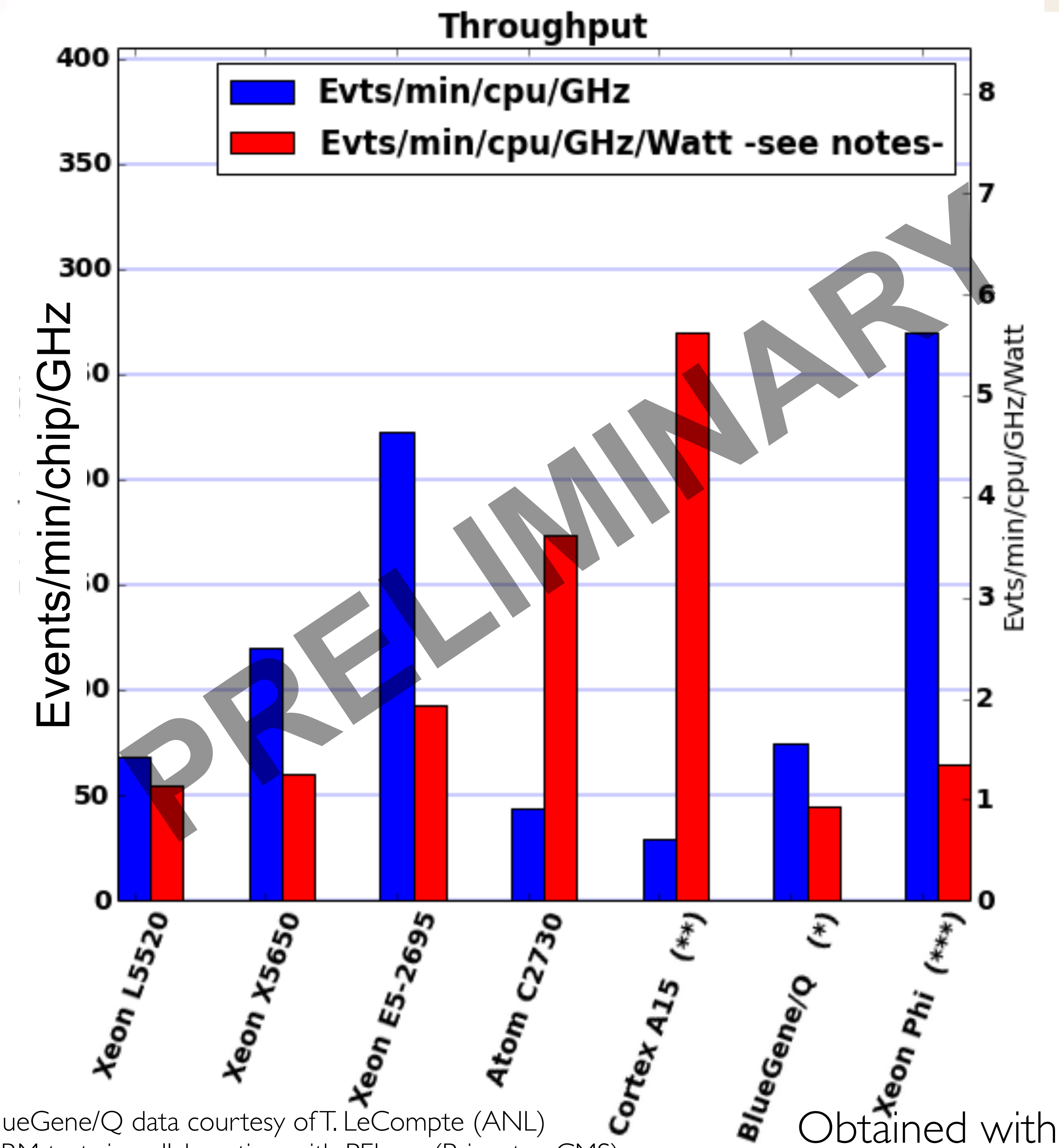




Memory usage (in MB)
200+18*Nt
(instead of 200*Np)

Obtained with "CMS-style" geometry
"Your milage may vary"

# Different Architectures



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

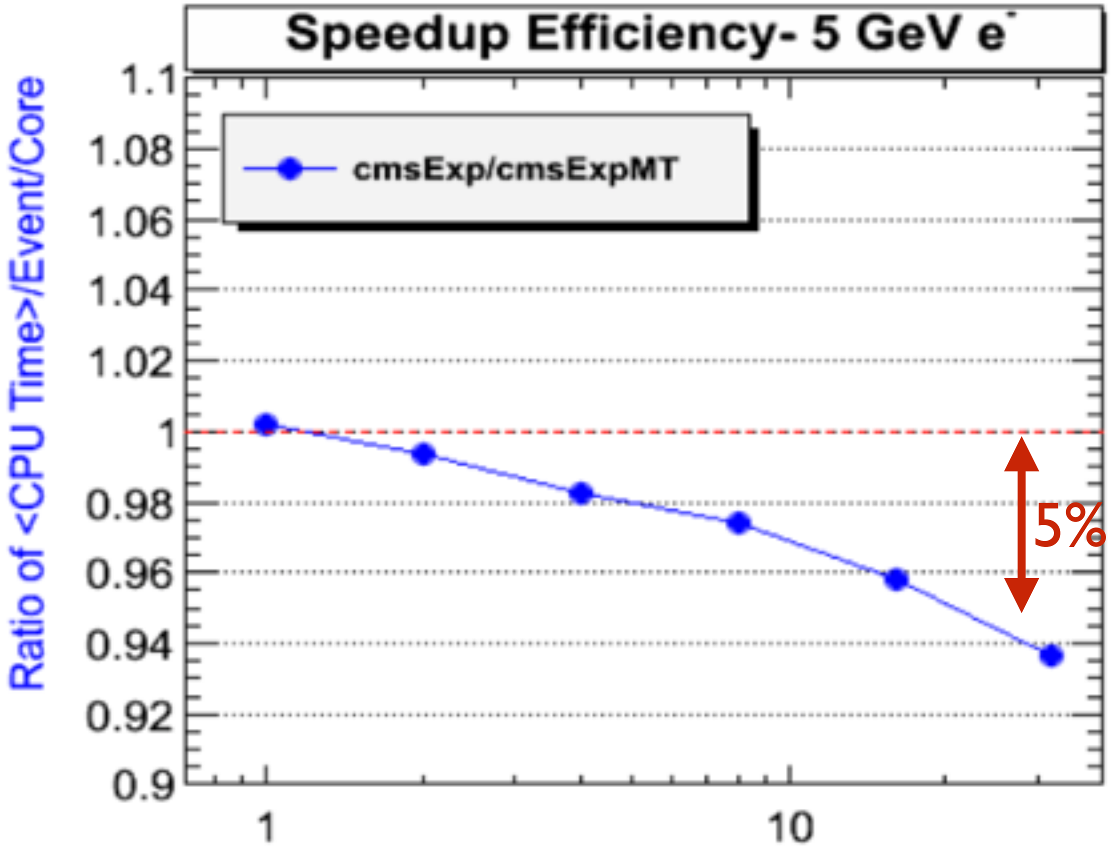
- Intel / AMD
- MIC
- PowerPC (BG/Q)
- ARM / Intel Atom

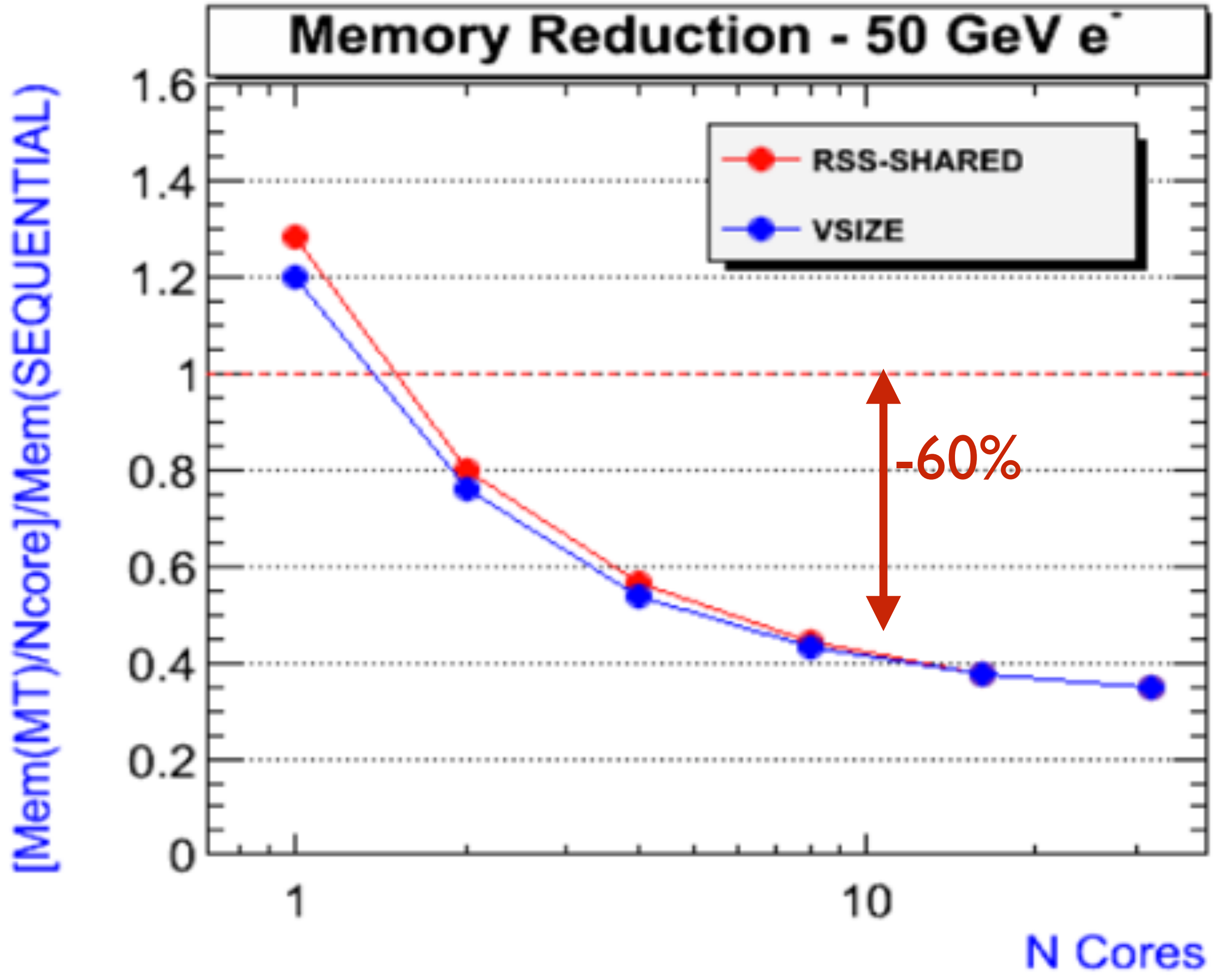
BlueGene/Q data courtesy of T. LeCompte (ANL)
ARM tests in collaboration with P.Elmer (Princeton;CMS)
Hardware courtesy of OpenLab (CERN)

Obtained with "CMS-style" geometry
"Your milage may vary"

G4 Ver 10.00.p01

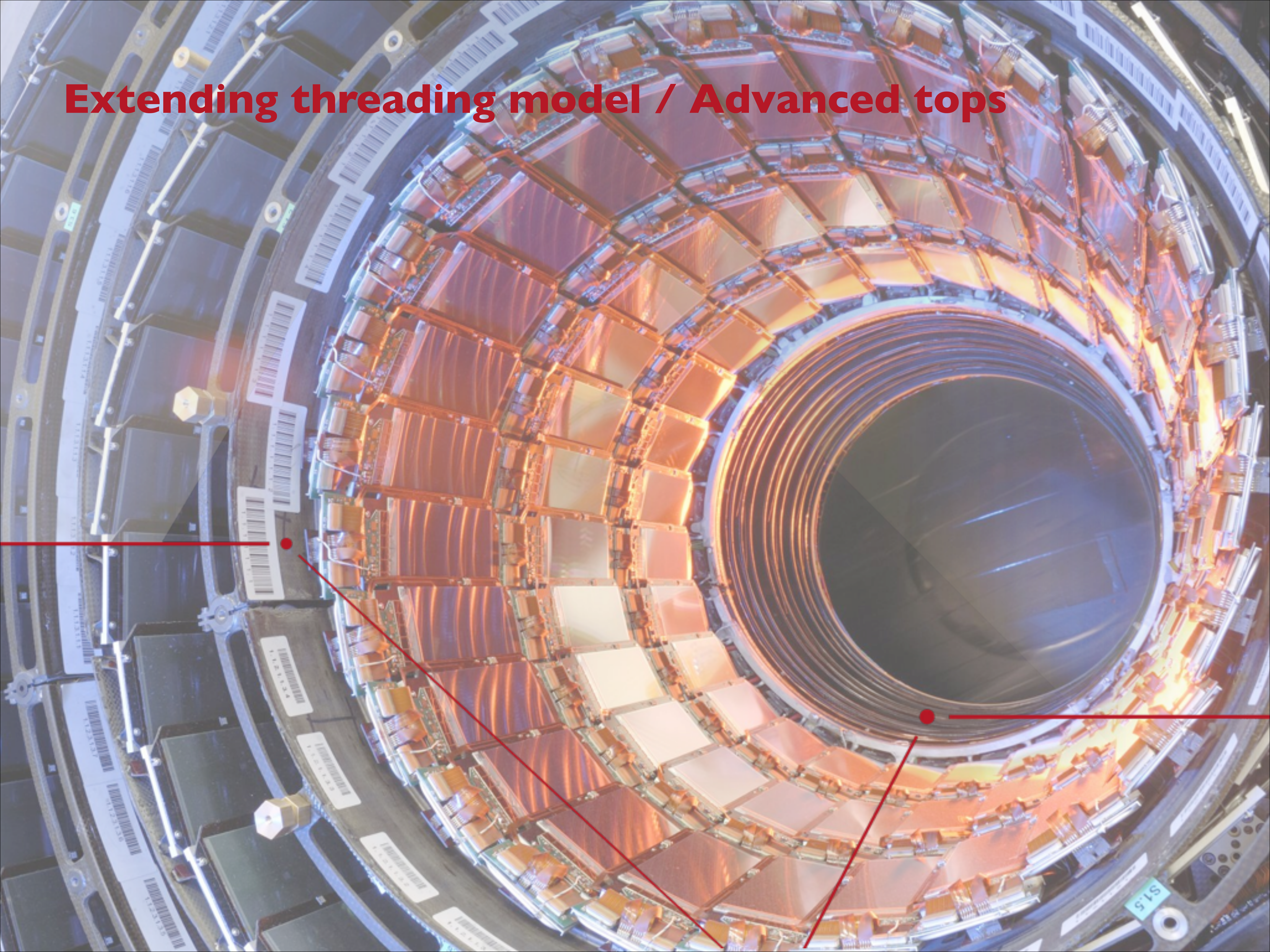
# Comparison with Sequential





Obtained with "CMS-style" geometry
"Your milage may vary"

# Extending threading model / Advanced tops



# User hooks

- In special cases you may need to customize some aspects of the Thread behavior (only for experts)
- You can:
  - Build your class inheriting from `G4UserWorkerIntialization` allows to add user code during thread initialization stages (see .hh for details).
  - The threading model is handled in `G4UserWorkerThreadInitialization`, sub-class to customize (how threads start, how they join, etc). See .hh for details

- Instantiate in main (as all other initializations) and add them to kernel via `G4MTRunManager::SetUserInitialiation( … )`

# Locks and Mutex

To add a lock mechanism (remember: **will spoil performances** but may be needed with non thread-safe code):

```
#include “G4AutoLock.hh”
namespace {
   G4Mutex aMutex = G4MUTEX_INITIALIZER;
}

void myfunction() {
  //enter critical section
  G4AutoLock l(&aMutex); //will automatically unlock when out
of scope
  return;
}
```

# Memory handling

Instead of using `__thread` keyword, use `G4ThreadLocal`. E.g.

```
static G4ThreadLocal G4double aValue = 0;
```

**Few classes/utilities have been created to help handling of objects.**

Described in Chapter 2.14 of Users's Guide For Toolkit Developers. In brief:

- `G4Cache` : Allows to create a thread-local variable in shared class
- `G4ThreadLocalSingleton` : for thread-private "singleton" pattern
- `G4AutoDelete` : automatically delete thread objects at the end of the job

http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForToolkitDeveloper/html/ch02s14.html

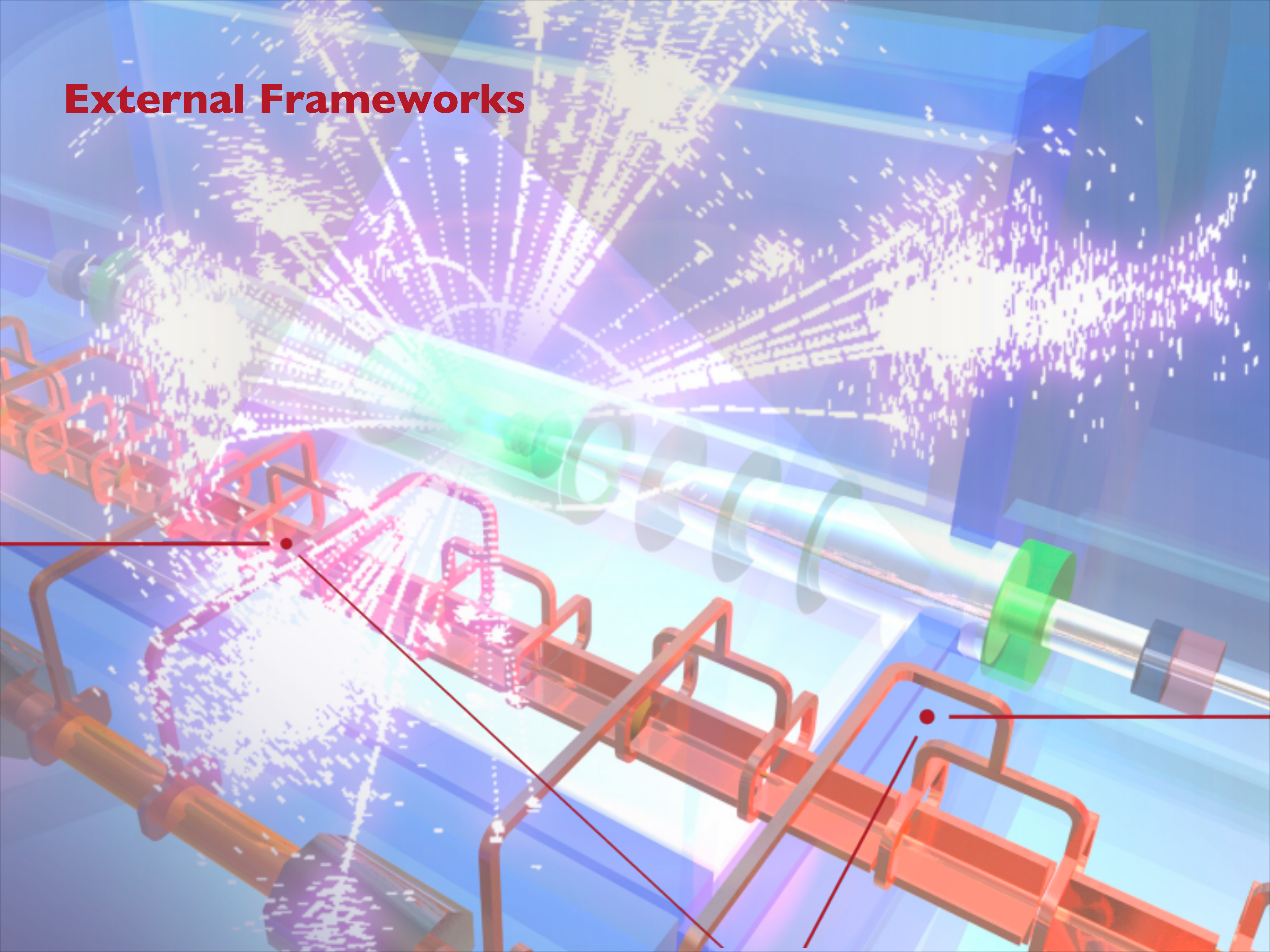
# Migrating from 9.6

Migration of a 9.6 application to MT is a 5-steps process

1. **Move** user-action instantiation to new G4UserActionInitialization class
2. **Use** G4MTRunManager in your main() function
3. **Split** Detector Construction in two: SD and Field go in new method ConstructSDandField
4. Use G4Run to **accumulate** run data, implement G4RunAction::Merge method
5. If you use anywhere G4Allocator (typically for hits), **transform** them to be G4ThreadLocal

All aspects have been covered in Hands On, see https://indico.cern.ch/event/250021/session/7/contribution/1/material/slides/1.pdf for more details

# External Frameworks



# Integration with MPI

**MPI based parallelism** already available in Geant4

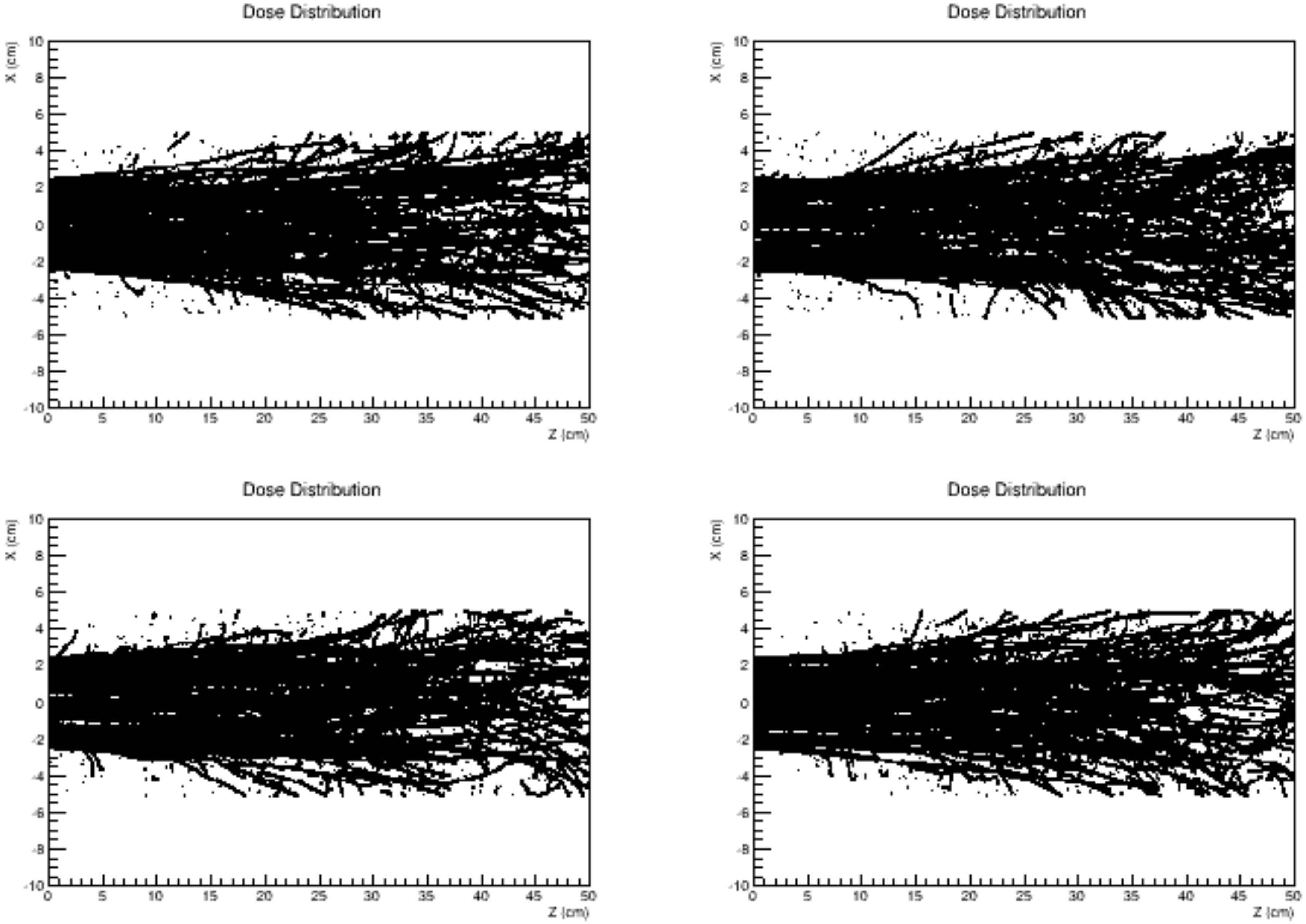
- MPI works together with MT
- See: examples/extended/parallel/MPI
- Expect new features in this category in the future: we are currently evaluating extensions

**Example:**

4 MPI jobs

2 threads/job

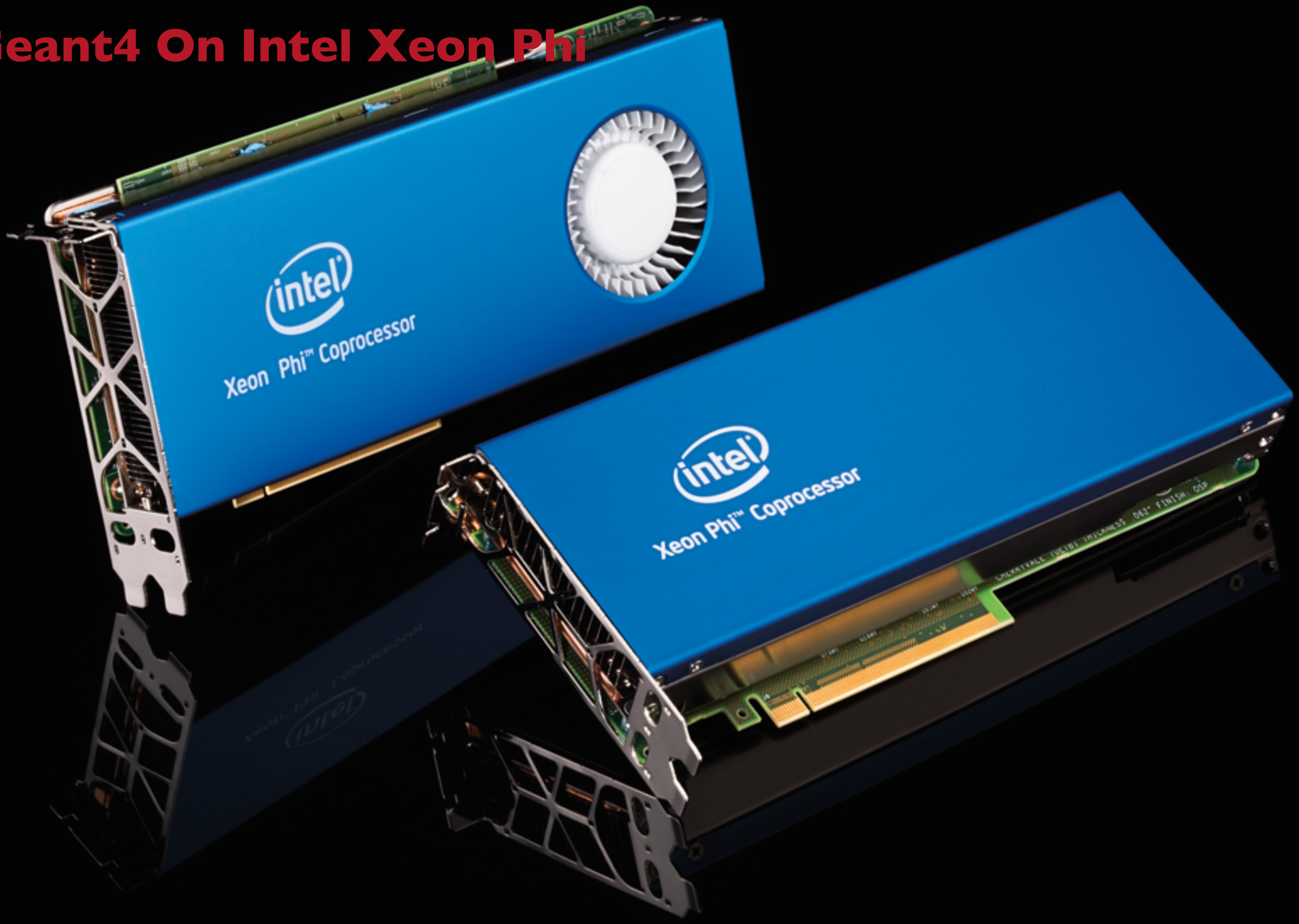
MPI job owns histogram



# Integration with TBB

- Intel Thread Building Block library
  - Task-based parallelism
  - Freely available for Linux/Mac/WIN
- We provide an example:
  - example/extended/parallel/TBB
  - Basic integration of TBB with Geant4 Version 10.0
  - Basically it replaces the POSIX multi-threading system we provide
- Note: we plan to intensively work on TBB examples in 2014, we will review and extend this example

# Geant4 On Intel Xeon Phi



# Disclaimer

- **We are not** encouraging you to run and buy a Xeon Phi
- **We are not** in any way involved with Intel and/or hardware vendors

- This is our experience: take it as an **example** of new hardware architecture
- Other architectures are also present: GPGPUs, Low-power consumption servers. For some of them we have some experience (ask us)
- We have chosen Intel Xeon Phi as a test-bed for its **simplicity in programming model**

# What is Intel Xeon Phi (aka MIC)?

- A PCIe card that acts as a **"co-processor"**
  - In a certain sense similar to using a GPU for general computing (I know, I'm not precise here…)
  - Up to 8 cards per host
- Based on **x86 instruction sets**
  - You do not need to rewrite your code, "just" recompile
- It requires Intel compiler (**not free**) and RTE
- **61 cores (x4 ways hyper-threading),** w/ max 16GB of RAM
  - Each core is much less powerful than a core of your host
  - In our experience: if your G4 code scales well 1 full card ~ 1 host
- Two ways of running code on the card:
  - Offload (a-la GPGPU)
  - Native: start a cross-compiled application on the card
- Geant4 has been ported to compile and run on MIC cards in **Native mode**

# How to compile

- Binaries for MIC are not compatible with host: **you need to cross-compile**
- We are also learning this process, in (near) future **we will provide detailed instructions/tools**. Feedback is more than welcome!
- What you need: **Intel C++ Compiler** (icpc)

```
export LDFLAGS="$LDFLAGS -mmic "
export CXXFLAGS="$CXXFLAGS -mmic”
export CFLAGS=“$CFLAGS -mmic”
export AR=/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-ar
export LD=/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-ld
cmake -DCMAKE_TOOLCHAIN_FILE=… \
      -DCMAKE_AR=${AR} -DCMAKE_LINKER=${LD} \
      -DCMAKE_CXX_COMPILER=icpc -DCMAKE_C_COMPILER=icc \
      […all the rest that you need, switch of graphics, but turn on MT!
…]
```

Change paths according to your installation

# toolchain file content

```
# this one is important
SET(CMAKE_SYSTEM_NAME Linux)
#this one not so much
SET(CMAKE_SYSTEM_VERSION 1)

SET(CMAKE_C_COMPILER icc)
SET(CMAKE_CXX_COMPILER icpc)
SET(CMAKE_LINKER /usr/linux-k1om-4.7/bin/x86_64-k1om-linux-ld)
SET(CMAKE_AR /usr/linux-k1om-4.7/bin/x86_64-k1om-linux-ar)
# where is the target environment
SET(CMAKE_FIND_ROOT_PATH  /opt/sw/linux/x86_64/intel/xe2013/composerxe)

# search for programs in the build host directories
SET(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
# for libraries and headers in the target directories
SET(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
SET(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

Change paths according to your installation