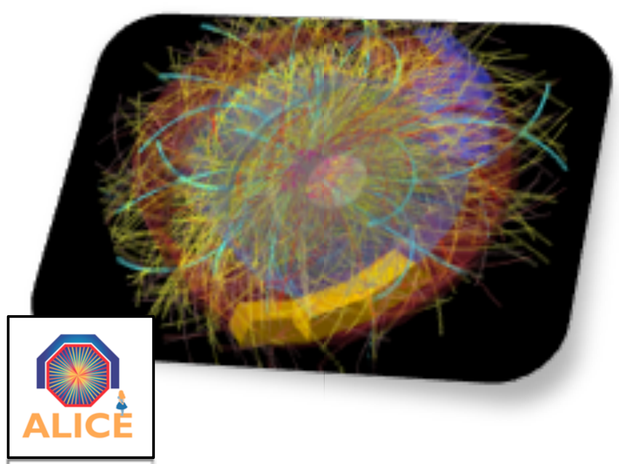
G4 Profiling and performance news from ALICE



Sandro Wenzel / CERN PH-SFT

Geant4 collaboration meeting, Sevilla, 24.09.2013

Situation:

* ALICE simulation is currently based on Geant3 (Fortran, no active development)
* ALICE potentially likes to move to Geant4 9.6
* however, currently rather large performance gap (factor 3 between Geant3 and Geant4) which should be made as small as possible

→ **Started systematic Geant4 benchmarking effort in ALICE**

Situation:

- ALICE simulation is currently based on Geant3 (Fortran, no active development)
- ALICE potentially likes to move to Geant4 9.6
- however, currently rather large performance gap (factor 3 between Geant3 and Geant4) which should be made as small as possible

→ **Started systematic Geant4 benchmarking effort in ALICE**

Outline:

* **Part I:** first profiling results: identification of (unexpected) hot-spot related to memory management
* **Part II**: opportunities from fast-math libraries
* **Part III**: first results from tuning simulation parameters (step size)

Understand what’s going on:

Valgrind

- full callgraph, profile and hotspot identification
- “time” spent in functions/libraries
- **expensive** (but “nightly” affordable): Ca. 90min for one (medium) event in ALICE on iCore7.

igprof, ...

- often used at CERN (statistical sampling)

Intel PIN Tools

- freely available instrumentation API used by all the Intel tools
- **fully programmable** instrumentation. Can give you exactly the information you want to know.
- suited to **log information on physics level:** properties of particles, where they go in detector, etc. (see also tool by Andrei Gheatta)

Tools / Approaches

Understand what's going on:

Valgrind

- full callgraph, profile and hotspot identification
- "time" spent in functions/libraries
- **expensive** (but "nightly" affordable): Ca. 90min for one (medium) event in ALICE on iCore7.

igprof, ...

- often used at CERN (statistical sampling)

Intel PIN Tools

- freely available instrumentation API used by all the Intel tools
- **fully programmable** instrumentation. Can give you exactly the information you want to know.
- suited to **log information on physics level:** properties of particles, where they go in detector, etc. (see also tool by Andrei Gheatta)

Benchmark environments

- Geant4 version **9.6.p01** (tarball)
- build with cmake (Release or RelwithDebInfo)
- in each case built whole **ppbench** software stack with the corresponding compiler version

os	proc	mem	machine	compiler
SLC6	iCore7 (3.4GHz, 4cores, no HT)	8GB	phpcsft96	**gcc/4.7**
SLC5	Intel Xeon (2.5 GHz, 4cores)	16GB	lxplus302	**gcc/4.3.6**
SLC5	Intel Xeon (2.27GHz, 8cores)	48GB	lxbuild175	gcc/4.3.6

Valgrind snapshot

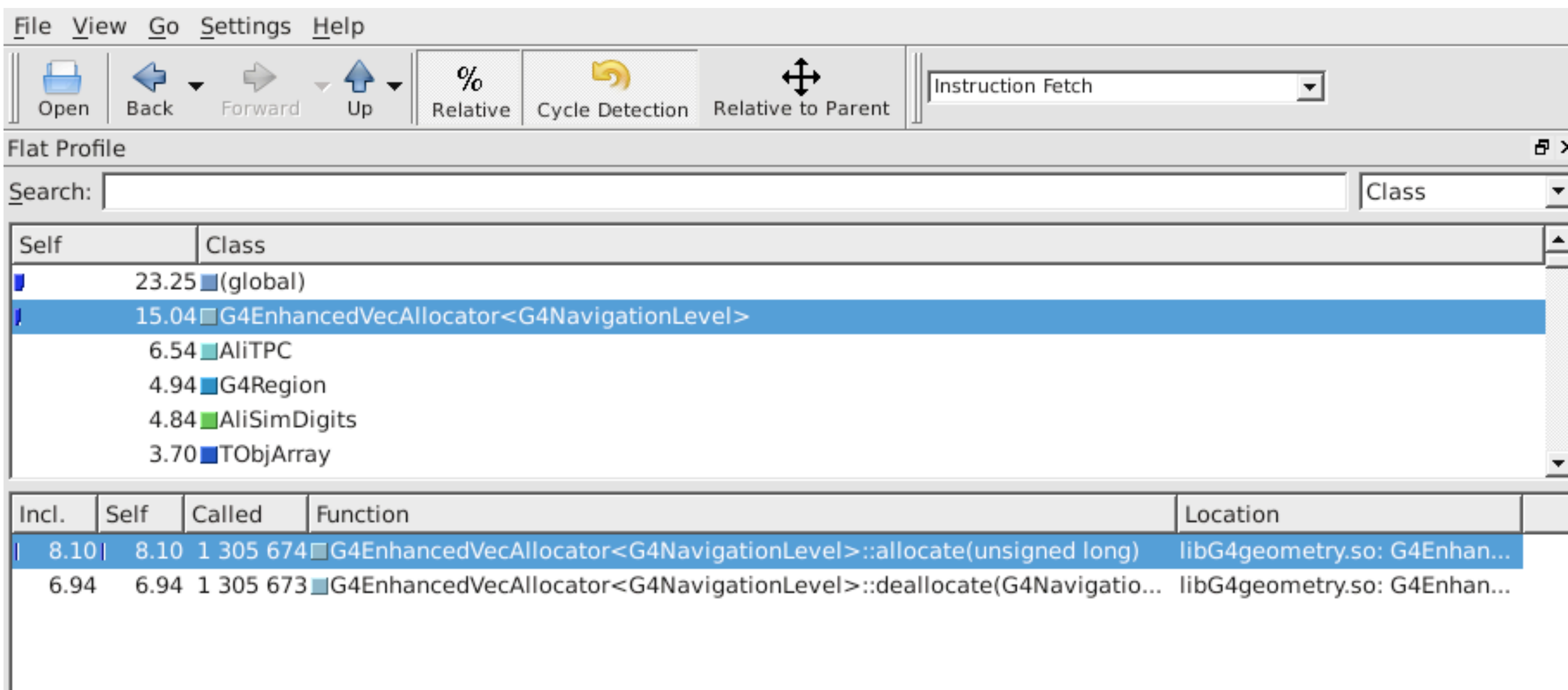
- run simulation for 1 event (including initialization and digitization) with valgrind

Valgrind snapshot

- run simulation for 1 event (including initialization and digitization) with valgrind
- resulting profile shows a whole list of important regions with reasonable contributions (digitization, G4processes , math functions ...)
  – **libm ~9%** – **G4processes ~8%** – **AliTPC~7%**

Valgrind snapshot

- run simulation for <u>1 event</u> (including initialization and digitization) with valgrind
- resulting profile shows a whole list of important regions with reasonable contributions (digitization, G4processes , math functions ...)
  - **libm ~9%**
  - **G4processes ~8%**
  - **AliTPC~7%**
- surprisingly **G4geometry** as largest contributor caused mostly by **one** unusual class **`G4EnhancedVecAllocator`**

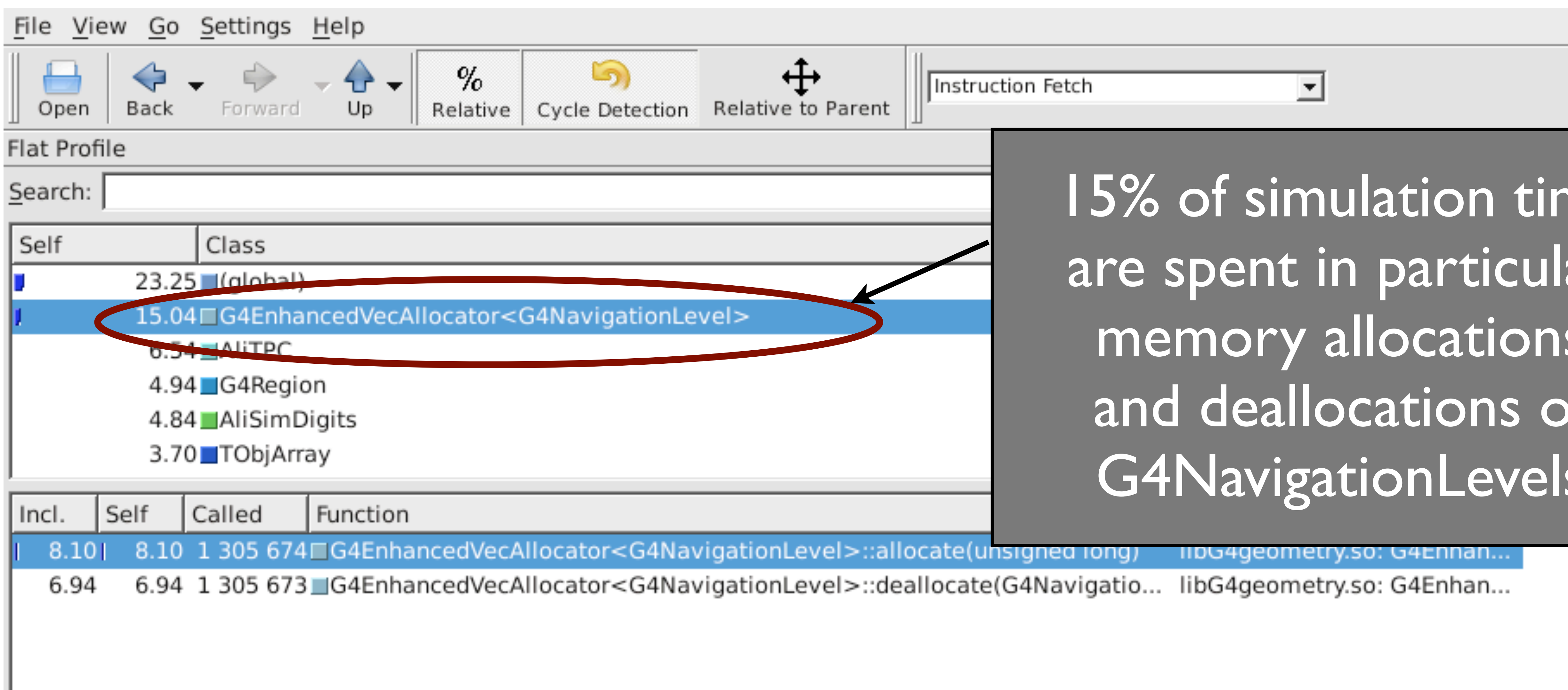


Valgrind snapshot

- run simulation for <u>1 event</u> (including initialization and digitization) with valgrind

- resulting profile shows a whole list of important regions with reasonable contributions (digitization, G4processes , math functions ...)
  – **libm ~9%** – **G4processes ~8%** – **AliTPC~7%**

- surprisingly **G4geometry** as largest contributor caused mostly by **one** unusual class **`G4EnhancedVecAllocator`**

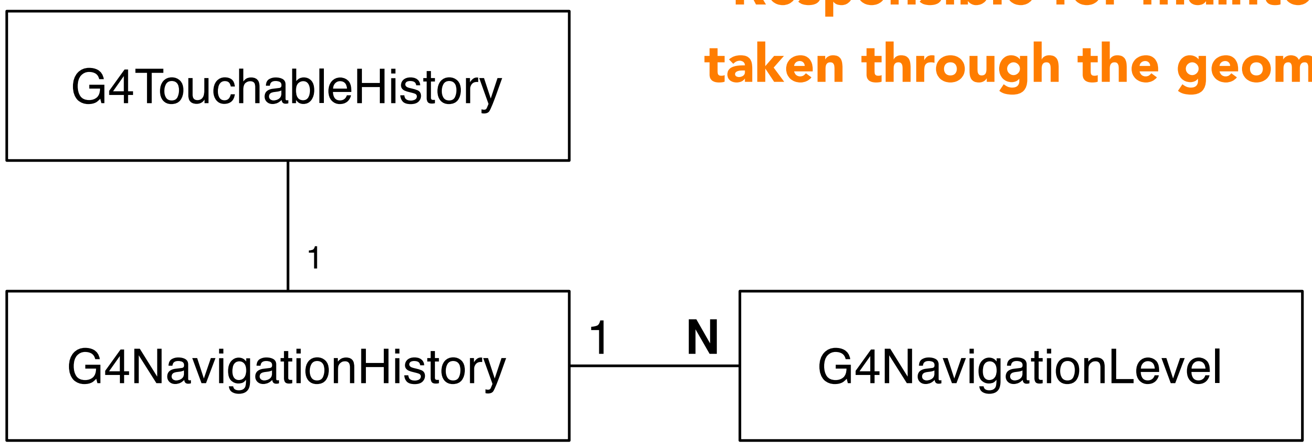


15% of simulation time are spent in particular memory allocations and deallocations of G4NavigationLevels

G4EnhancedVecAllocator

* **`G4EnhancedVecAllocator`** used only in class **`G4NavigationHistory`**

"Responsible for maintenance of the history of paths taken through the geometrical hierarchy"

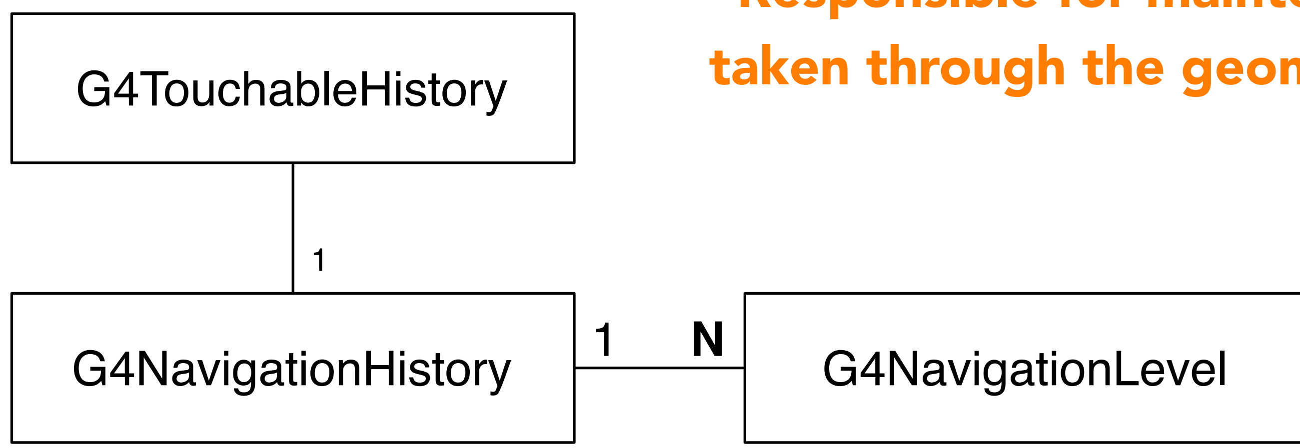


* **`G4NavigationHistory`** has vector of **`G4NavigationLevels`**.

std::vector< G4NavigationLevel, G4EnhancedVecAllocator < G4NavigationLevel > > fNavHistory

G4EnhancedVecAllocator

- **`G4EnhancedVecAllocator`** used only in class **`G4NavigationHistory`**



"Responsible for maintenance of the history of paths taken through the geometrical hierarchy"

- **`G4NavigationHistory`** has vector of **`G4NavigationLevels`**.

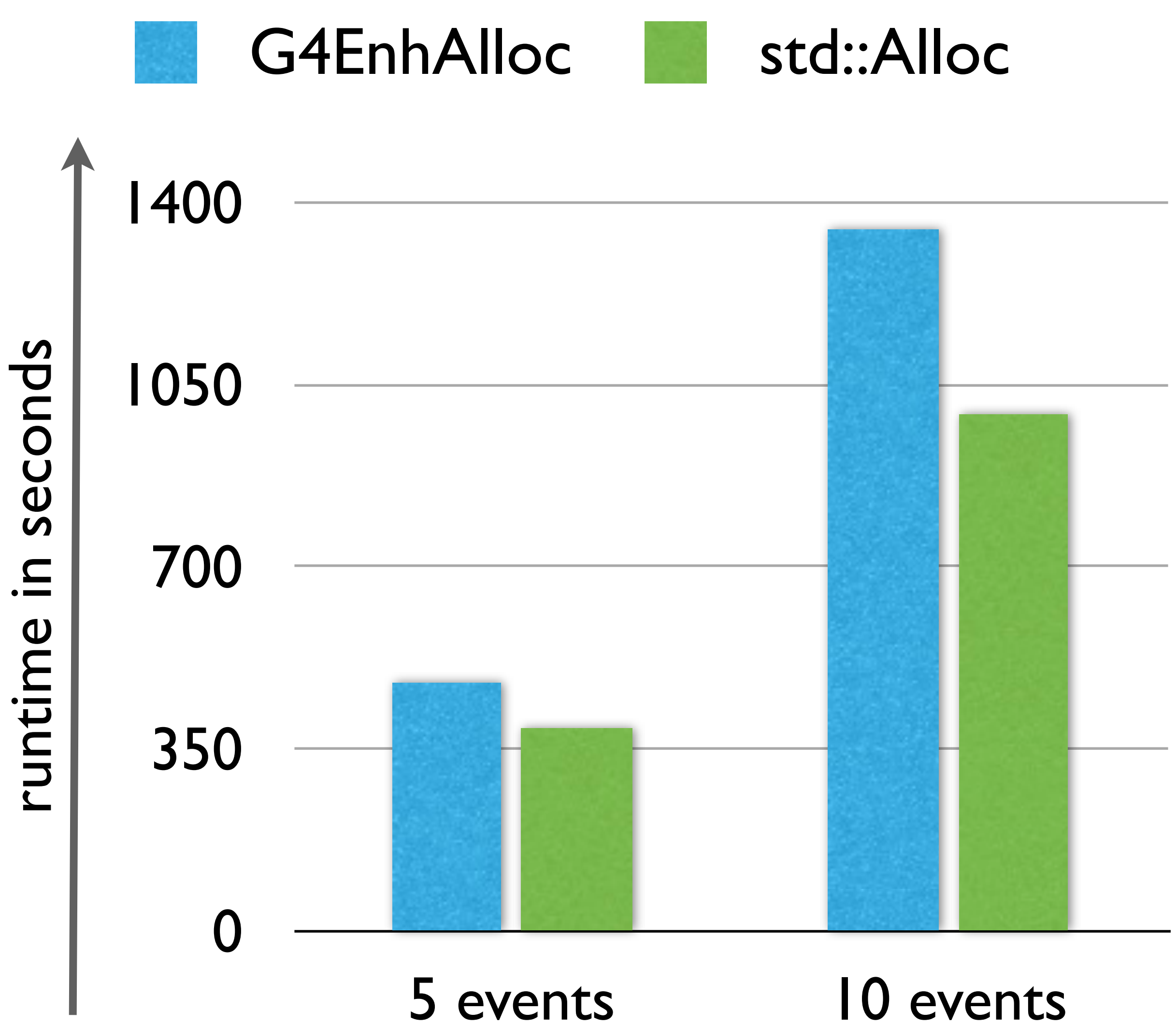
std::vector< G4NavigationLevel, G4EnhancedVecAllocator < G4NavigationLevel > > fNavHistory

- purpose of enhanced allocator is to **optimize memory management** for vectors of G4NavigationLevel (avoid memory fragmentation)

> what happens if we use standard C++ allocator instead?
>
> std::vector< G4NavigationLevel> fNavHistory

comparison results: total simulation time

- timing results on my iCore7 (i7-3770, 3.4GHz) , gcc4.7
- both allocator version give identical simulation results
- version built with std::allocator **systematically faster**



Mean performance difference

# Events	runtime ratio
5	1.22
10	1.34
20	1.32

Influence of compiler, OS?

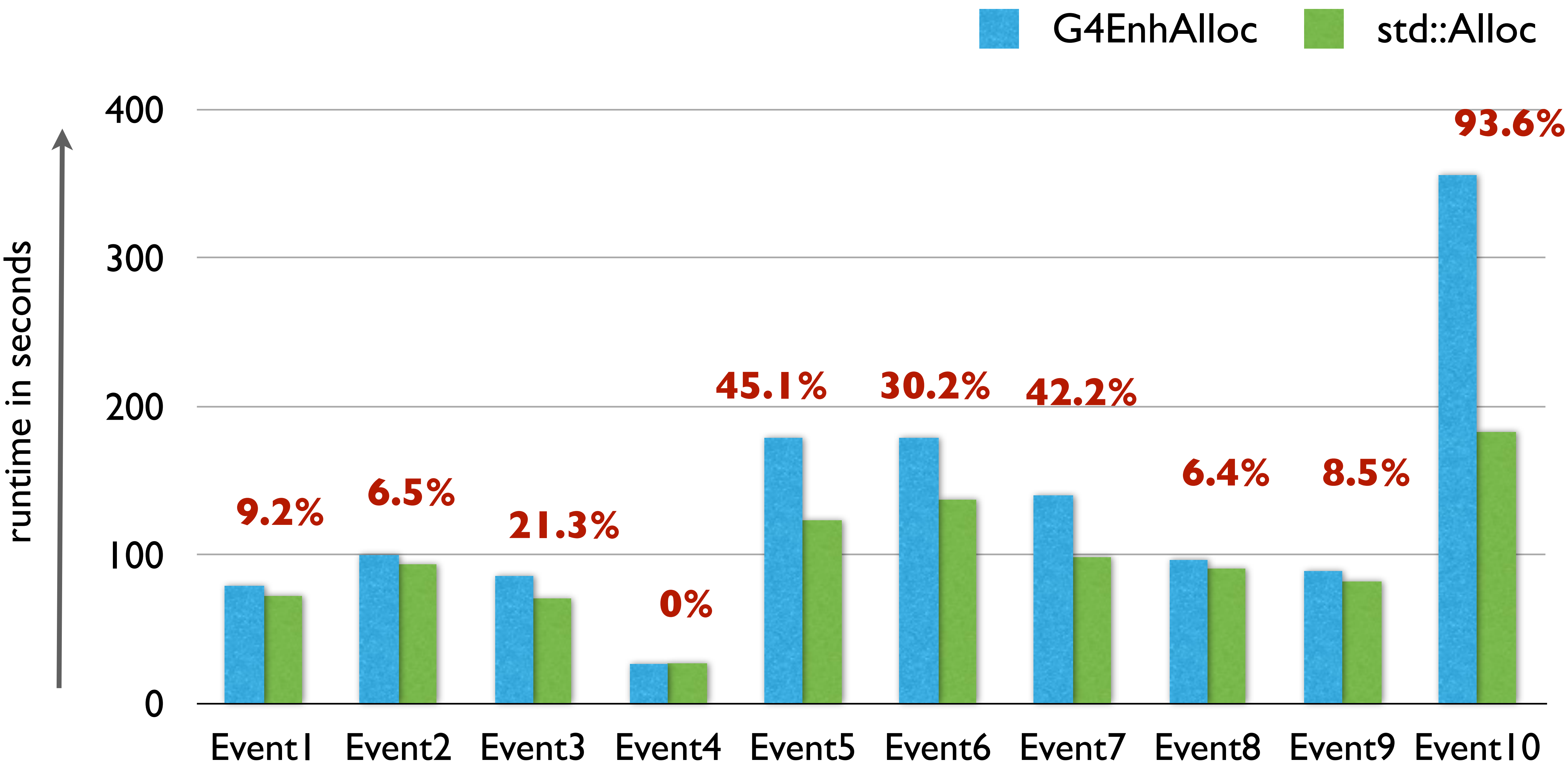
- runtime difference observed consistently on different machines/compiler versions (here for N=10 events)

os	proc	mem	machine	compiler	runtime ratio
SLC6	iCore7 (3.4GHz, 4cores)	8GB	phpcsft96	gcc/4.7	**1.34**
SLC5	Intel Xeon (2.5 GHz, 4cores)	16GB	lxplus302	gcc/4.3.6	**1.32**
SLC5	Intel Xeon (2.27GHz, 8cores)	48GB	lxbuild175	gcc/4.3.6	**1.35**

- get performance difference also with Intel compiler (v13)
- **but should extend tests to different platforms (Mac)**

Comparison per event (10event run)

- study performance difference per event in 10 event run
- correlation between total runtime and performance penalty?



Part II: Investigating Alternatives to Libm

- By default, Geant4 uses GNU math library (on linux, Mac)
  - **rocksolid**, but **not fastest implementation** around

Started investigations to quantify opportunities from using faster (less precise) libraries:

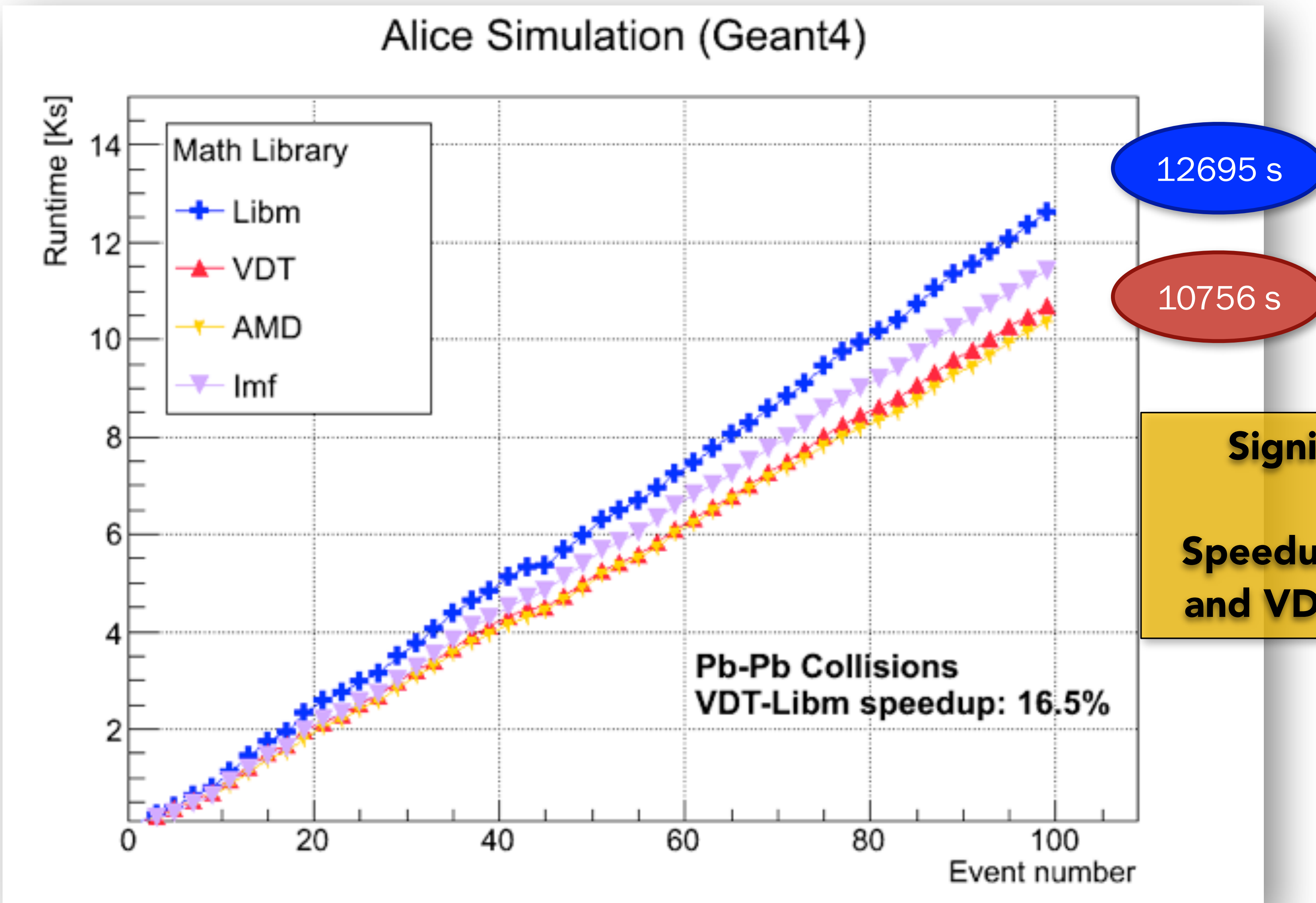
- Commercial or closed source libraries
  - Intel math library
  - AMD libm
- open source alternatives
  - **VDT** (CMS/CERN development)

    PIPARO, D., INNOCENTE, V. and HAUTH, Th.

    svnweb.cern.ch/trac/vdt or: github.com/drbenmorgan/vdt

Quantification of speedup-opportunity

- simulate 100 events in AliRoot / Geant4.9.6.p1
- concentrate on Geant4 speedup: **time the runloop (no digitization!)**
- selection and usage of fast-math library by **LD_PRELOAD**:
  - **`“export LD_PRELOAD = ..libvdt.so”`**
  - **no recompilation** necessary but performance gain might be smaller than a compile time inclusion



Significant gains!

Speedup of AMD libm and VDT almost equal

Sandro Wenzel, 24.09.2013

Part III: Tuning Simulation Parameters

- ALICE (Ivana!) started efforts to tune simulation parameters to optimize runtime
- Up until now, a **small step limit was imposed** in **low density materials** (too many steps done in comparison to real geometry steps, physical steps) although geometry/physical step could be much larger
- A way to control/play with this step implemented. First results (when limiting this step to 10m in low density materials) are available

New Status 09/2013

- rerun measurements with new software versions (Geant4.9.6p2, new SLC6 libm, ...) with all possible combinations of tunings
- time in seconds for N=50 events (Geant4 runloop)

	EnhAllocator		Std::Allocator	
cmath	**7400**		5680	
vdt	6500		5148	
	default step	large step limit	default step	large step limit

preliminary; validation outstanding

New Status 09/2013

* rerun measurements with new software versions (Geant4.9.6p2, new SLC6 libm, ...) with all possible combinations of tunings
* time in seconds for N=50 events (Geant4 runloop)

	EnhAllocator		Std::Allocator	
cmath	**7400**	6400	5680	**4430**
vdt	6500	6400	5148	4470
	default step	large step limit	default step	large step limit

* change in allocator biggest improvement
* removal of step limitation important but no orthotogal with fast-math; this indicates that overall "exp" and "log" become much less important
* with step limitation, use of fast-math relevant

total gain 1.65

preliminary; validation outstanding

Backup slides

towards a physics validation

* Complete validation out of scope ... but first idea is to look at Geant4 step lengths:

All tests use Geant4 9.6.p01 (std::alloc), compiled with gcc/4.7 on SLC6/SLC5

based on slides by D. Piparo / CERN

towards a physics validation

- Complete validation out of scope ... but first idea is to look at Geant4 step lengths:
  - given same sequence of events + random numbers, would ideally expect **same number of Geant4 step lengths for all math libraries**

All tests use Geant4 9.6.p01 (std::alloc), compiled with gcc/4.7 on SLC6/SLC5

based on slides by D. Piparo / CERN

towards a physics validation

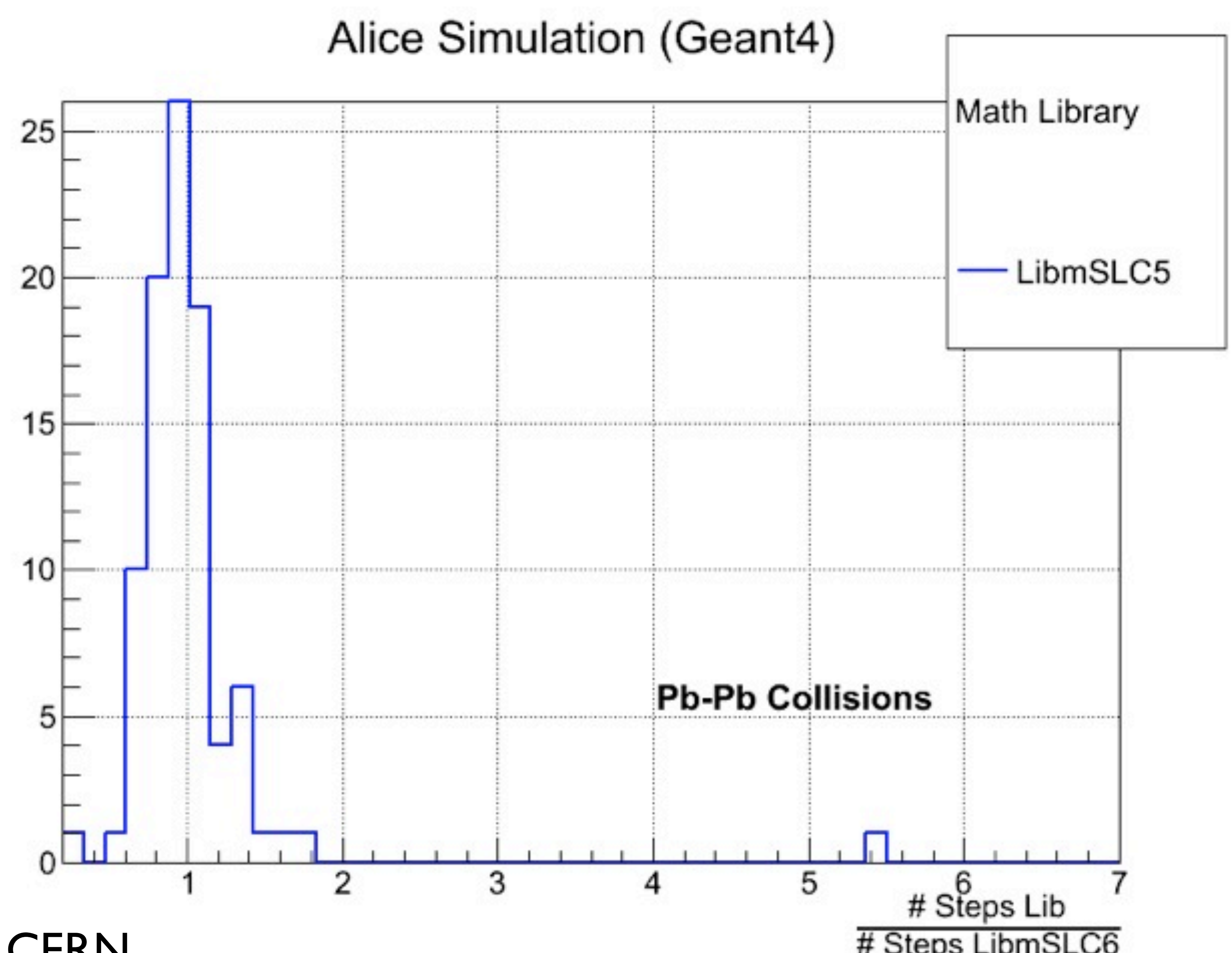
- Complete validation out of scope ... but first idea is to look at Geant4 step lengths:
  - given same sequence of events + random numbers, would ideally expect **same number of Geant4 step lengths for all math libraries**
  - instructive to check this taking the **libm** but **different kernels** (slc5 - slc6)

All tests use Geant4 9.6.p01 (std::alloc), compiled with gcc/4.7 on SLC6/SLC5

based on slides by D. Piparo / CERN

towards a physics validation

- Complete validation out of scope ... but first idea is to look at Geant4 step lengths:
  - given same sequence of events + random numbers, would ideally expect **same number of Geant4 step lengths for all math libraries**
  - instructive to check this taking the **libm** but **different kernels** (slc5 - slc6)

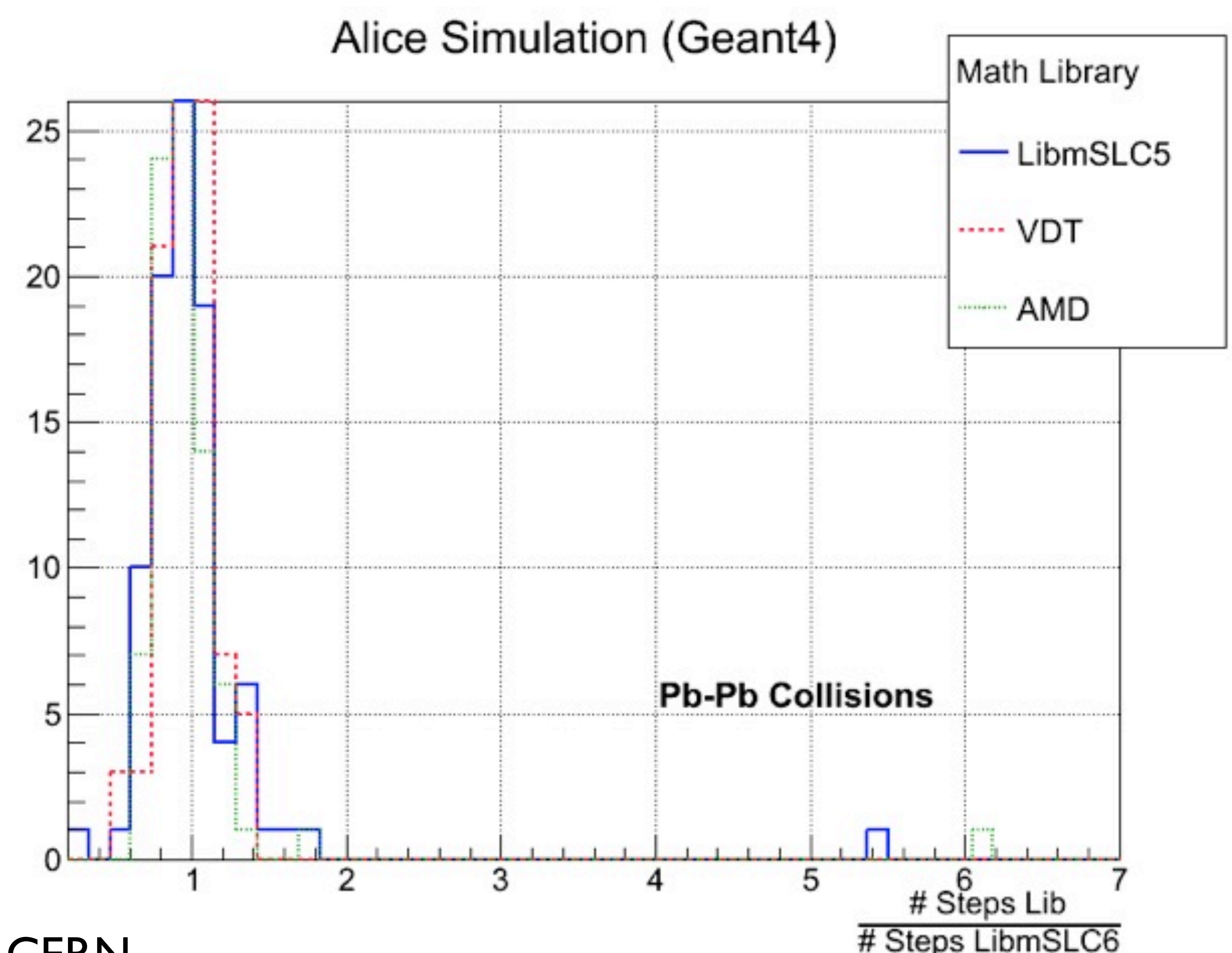


All tests use Geant4 9.6.p01 (std::alloc), compiled with gcc/4.7 on SLC6/SLC5

based on slides by D. Piparo / CERN

towards a physics validation

- Complete validation out of scope ... but first idea is to look at Geant4 step lengths:
  - given same sequence of events + random numbers, would ideally expect **same number of Geant4 step lengths for all math libraries**
  - instructive to check this taking the **libm** but **different kernels** (slc5 - slc6)
- **vdt, amd induce just same kind of variability !! (on slc6)**



All tests use Geant4 9.6.p01 (std::alloc), compiled with gcc/4.7 on SLC6/SLC5

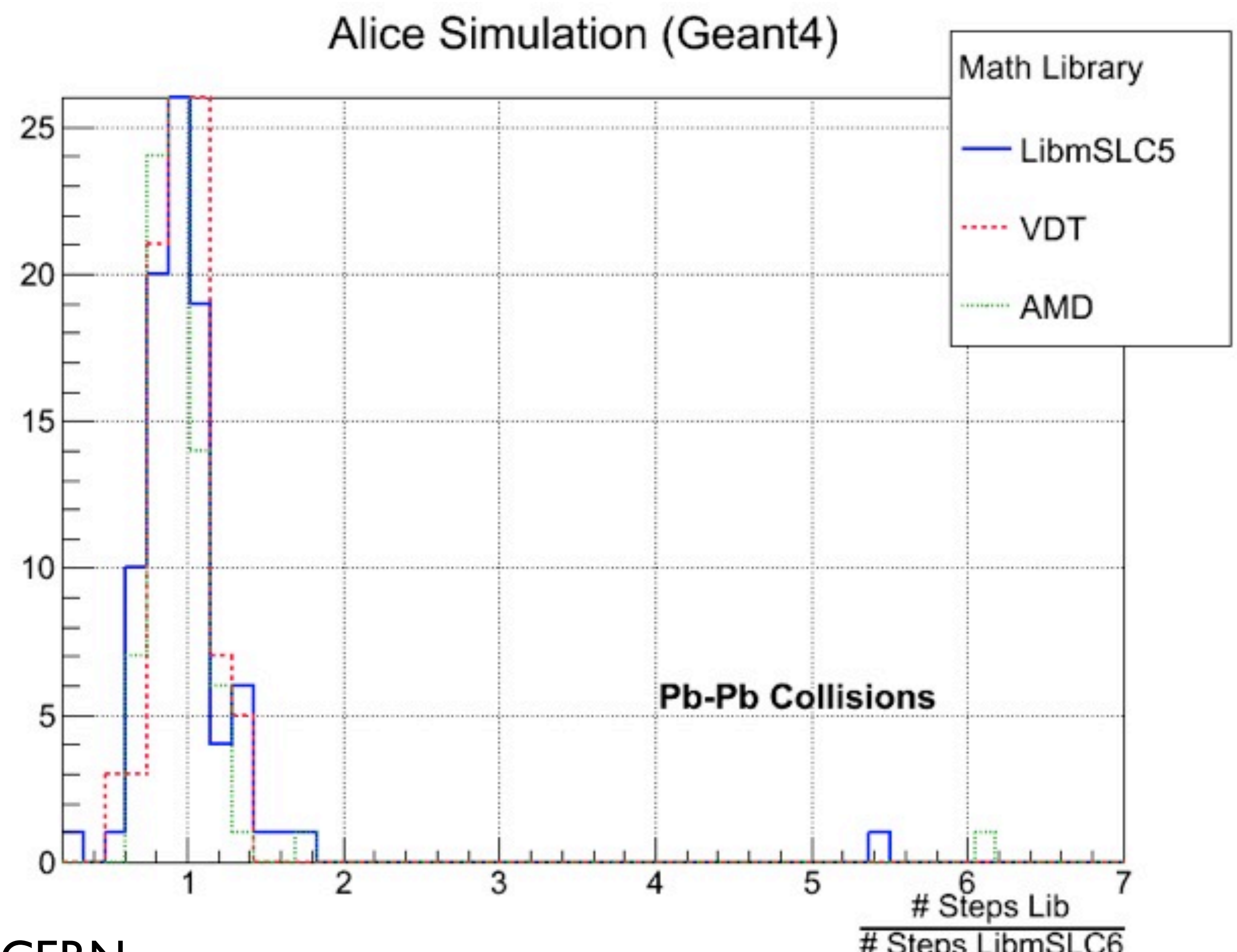
based on slides by D. Piparo / CERN

towards a physics validation

- Complete validation out of scope ... but first idea is to look at Geant4 step lengths:
  - given same sequence of events + random numbers, would ideally expect **same number of Geant4 step lengths for all math libraries**
  - instructive to check this taking the **libm** but **different kernels** (slc5 - slc6)
- **vdt, amd induce just same kind of variability !! (on slc6)**

This is encouraging!

Complete validation needed for final sign-off!



All tests use Geant4 9.6.p01 (std::alloc), compiled with gcc/4.7 on SLC6/SLC5

based on slides by D. Piparo / CERN

Next steps

- study memory impact of using std::allocator
- physics validation
- do next cycle of benchmarks (with improvements included)
    - get modified costs and new important code sections
    - digitization is a large part to tackle (preliminary performance increase) but more or less independent of G4