Session 1B: Computing Performance

(S.Y. Jun & D. Elvira)

G4CP Results and Issues from LHC experiments	*Dr. John APOSTOLAKIS*	
WH8XO - Hornet Nest		11:00 - 11:25
G4CP Results and Issues from Intensity Frontier	*Dr. Robert HATCHER*	
TAU	*Boyana NORRIS*	
WH8XO - Hornet Nest		11:50 - 12:15
Memory Leakage Monitoring	*Soon Yung JUN et al.*	
WH8XO - Hornet Nest		12:15 - 12:35
Discussion		
WH8XO - Hornet Nest		12:35 - 12:45

CPU Performance: ATLAS&CMS (John Apostolakis)

CMS Full Simulation Computing performance

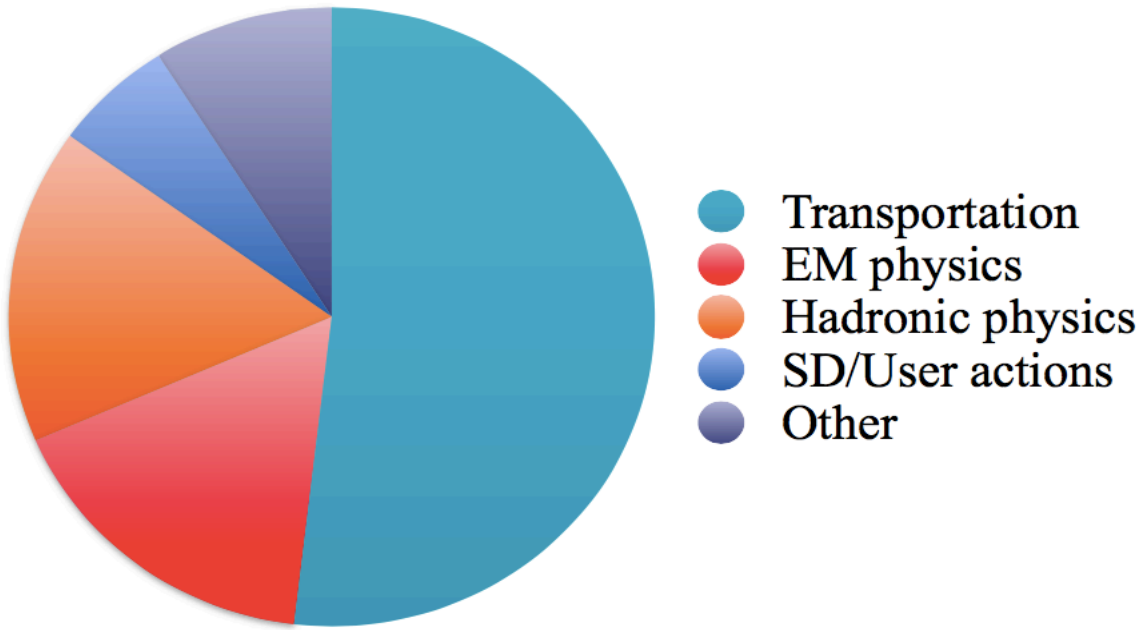
Vladimir Ivantchenko for the CMS Simulation team

ATLAS Full Simulation Computing performance

Zach Marshall, Elmar Ritsch, Philip Clark, Steve Farrell

Geant4 Status in CMS

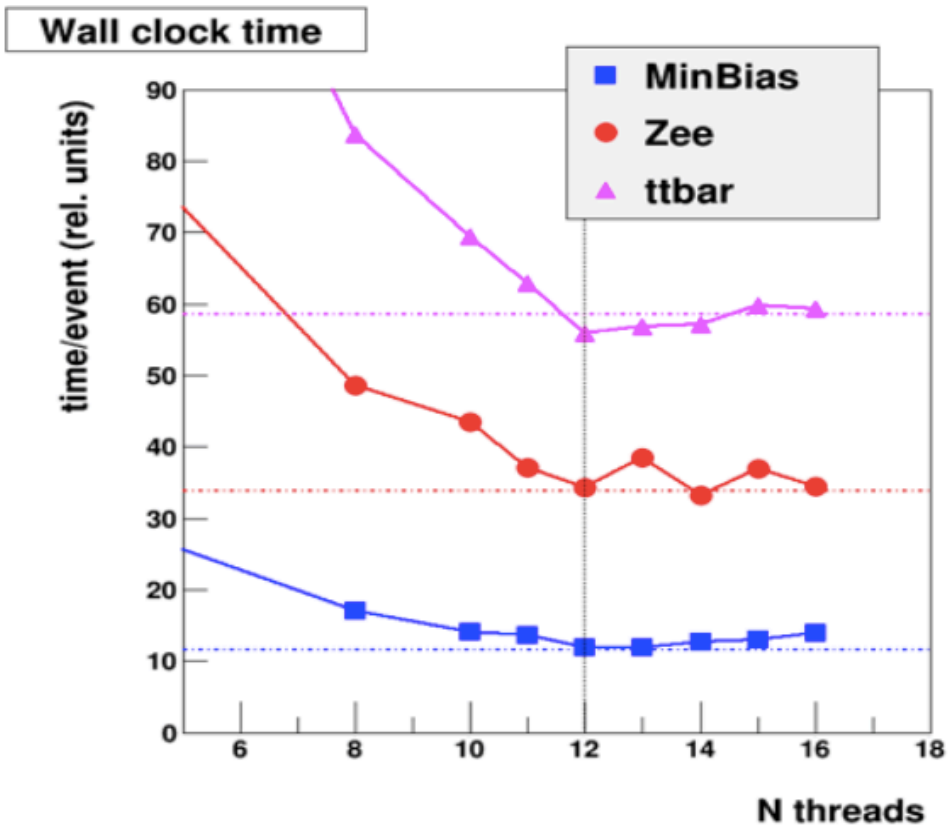
- 2015 production: 10.0.p02 (sequential), QGSP_FTFP_BERT_EML, ~ 5 billion events (2015) (+⬆)
  - CPU

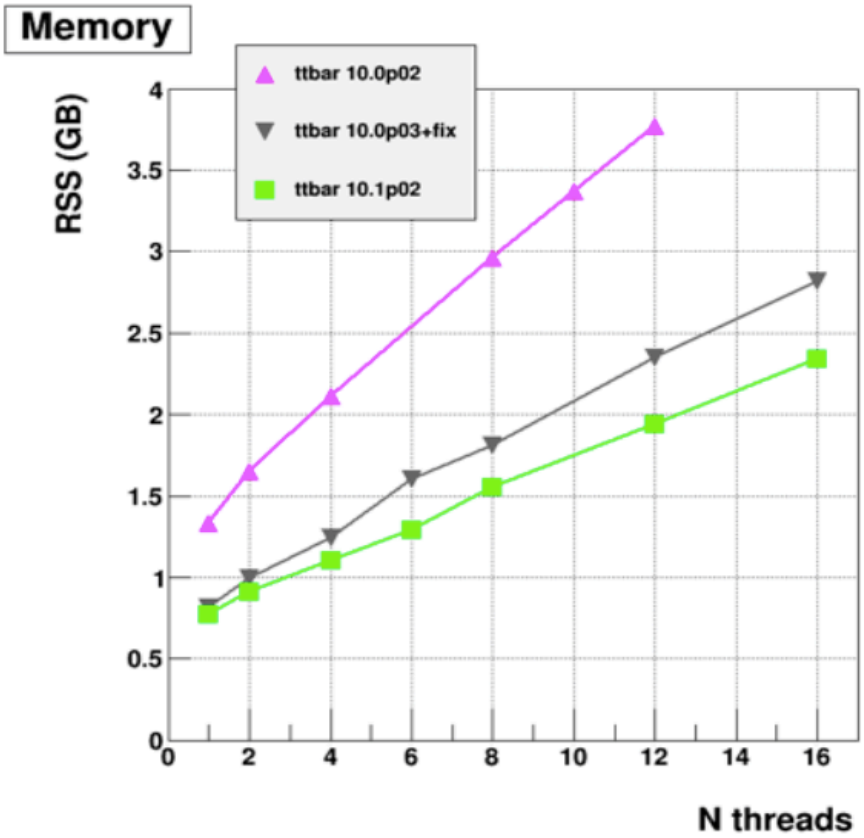


- Technical performance Improvements for Run 2
  - Upgrade to 10.0 (~5%)
  - Russian Roulette (~30%)
  - CMSSW optimization (~15%)
  - Library repackaging (~10%)

Geant4 Status in CMS

- Multi-threaded Geant4 (10.0p03) is fully integrated with CMS multi-threaded framework – plan to use it in production 2016
- Performance of CMS MT GEN- SIM (CHEP2015)
  - Excellent scaling performance in Time/Event
  - ~2 GB RSS for 12 single threaded jobs (+200 MB per thread)





Geant4 Status in ATLAS

- Integrated Sim. Framework (ISF) use almost all production.
  - 9.6 full simulation used in 80% of production (ratio will drop)
  - Expect move to 10.1 for next campaign (end 2015)
- Stability and production
  - Crash rate: ~1.5% failure for jobs of 1,000 events (unacceptable)
  - The Multi Level Locator has proven to be a weakness
- Hot spots and remedies
  - Neutrons take a lot of CPU time.
  - Might seek to use available biasing features
- Memory – use and churn
  - Memory consumption is significant, but not enormous concerns
  - Memory churn was issue, but Geant4 no longer dominates churn
- Seen potential of static builds vs DLLs (difficult for ATLAS)
- Reasonably advanced prototype of MT app for Cori

Geant4 Performance: Fermilab Intensity Frontier Experiments

Robert Hatcher
Fermilab

Geant4 Collaboration Mtg 2015-09-28
Parallel Session 1B - Computing Performance

A Variety of Applications

- See Plenary: Geant4 user requirements from Intensity Frontier Experiments for some background material

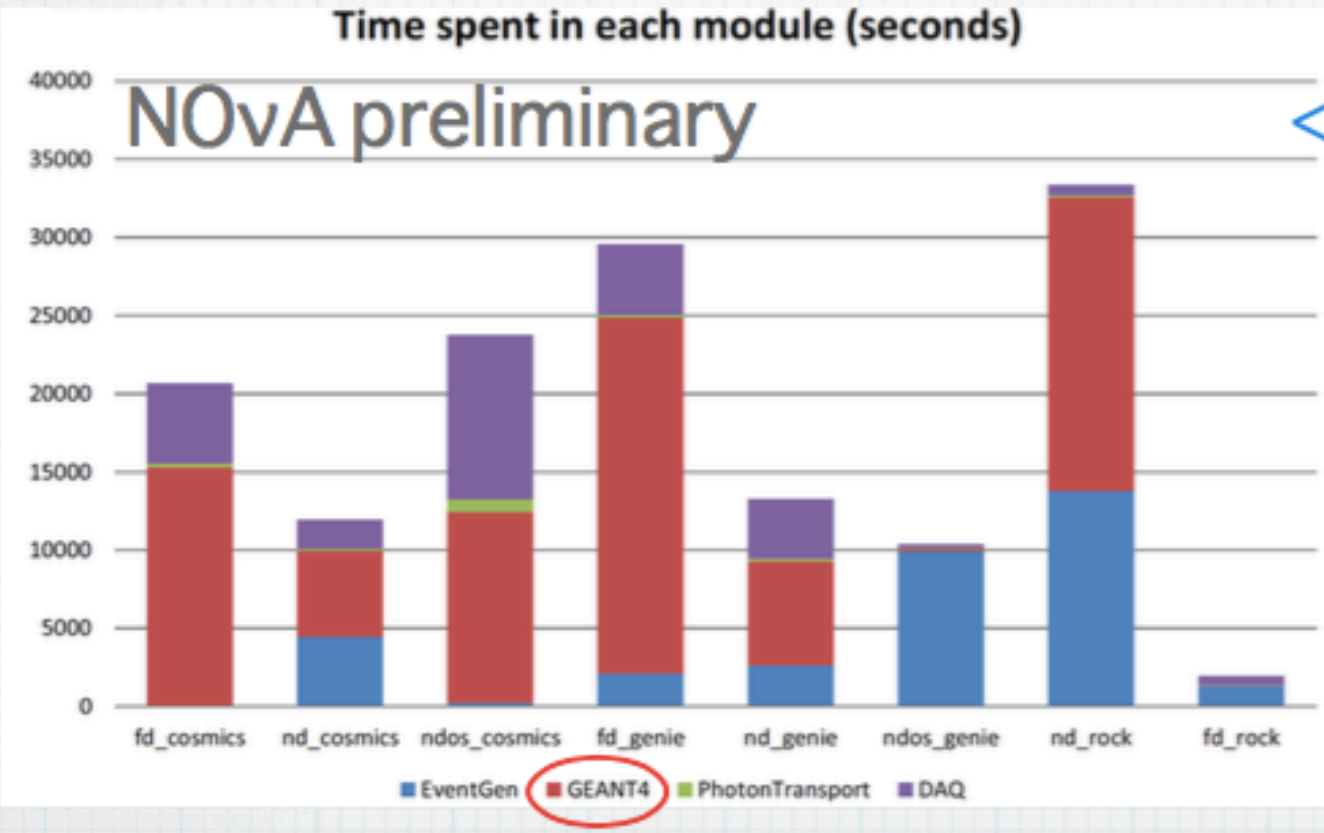
- Beamline simulations
  - Experiments on the same beamline can often share the same simulation of the beam, but there are several beamlines
  - 2 μ + 3.5 ν (NuMI LE vs. ME = 1.5)

- Detector simulations
  - Experiments on the same beamline have different needs when it comes to detector simulations, varied technologies
  - 2 μ + ~14 ν (including Near/Far, DUNE prototypes) ✓

- Actual statistics are difficult to obtain/summarize — many different activities, no centralized accounting, simulation intermixed w/ reconstruction, shared efforts (ν beamlines)

ν Detector Simulation

- Three simulation sub-components generally done together
  - Flux + GENIE [ν+(A,Z)➛final state particles leaving the nucleus]
  - Geant4 propagation for energy depositions ✓
  - light collection / transport + FE electronics & DAQ

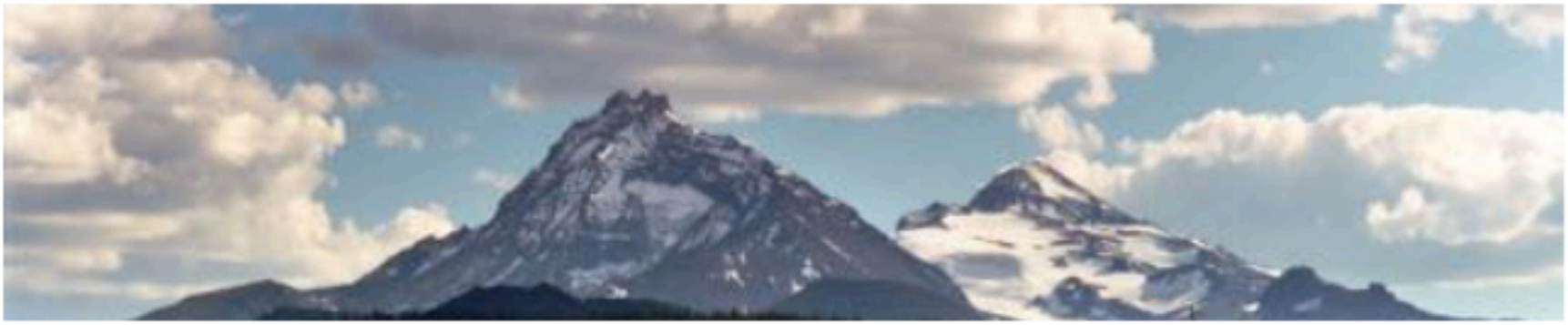


ratios and total times will vary w/ the experiment, backgrounds sources, etc.

(don't take absolute time too serious — some were constrained by what resources were available; also I'm not sure of the normalization)

IF Summary

- A variety of programs – hard to generalize CPU/memory uses
- Still use relatively old versions of Geant4 (9.2, 9.4, 9.6)
- Generally, open to new technology (MT, track parallelism, multi- cores, etc) if no extra efforts are necessary
- Effort underway to centralize MC production needs and estimate required resources
- G4CPT will seek for representative applications to evaluate computing performance of IF-experiments (profiling and benchmarking)



TAU Performance System®

Examples of Performance Analysis of Geant4 Electromagnetic Processes

Boyana Norris

Associate Professor

Computer and Information Science

University of Oregon

TAU Performance System®

- Tuning and Analysis Utilities (19+ year project)
- Comprehensive performance profiling and tracing
  - Integrated, scalable, flexible, portable
  - Targets all parallel programming/execution paradigms
- Integrated performance toolkit
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- Easy to integrate in application frameworks

http://tau.uoregon.edu

How to compile Geant4

- For probe-based measurement:
  - For full instrumentation, simply configure with:

```
cmake -DCMAKE_CXX_COMPILER=tau_cxx.sh \
      -DCMAKE_CC_COMPILER=tau_cc.sh …
```

  - For selective instrumentation, create a **select.tau** file, then define the **TAU_OPTIONS** environment variable accordingly, for example:

```
export TAU_OPTIONS='-optVerbose –optRevert \
-optTauSelectFile="$HOME/Geant4/select.tau"'
```

- For sampling-based measurement
  - Configure and compile as usual (ensure –g is used)

TAU

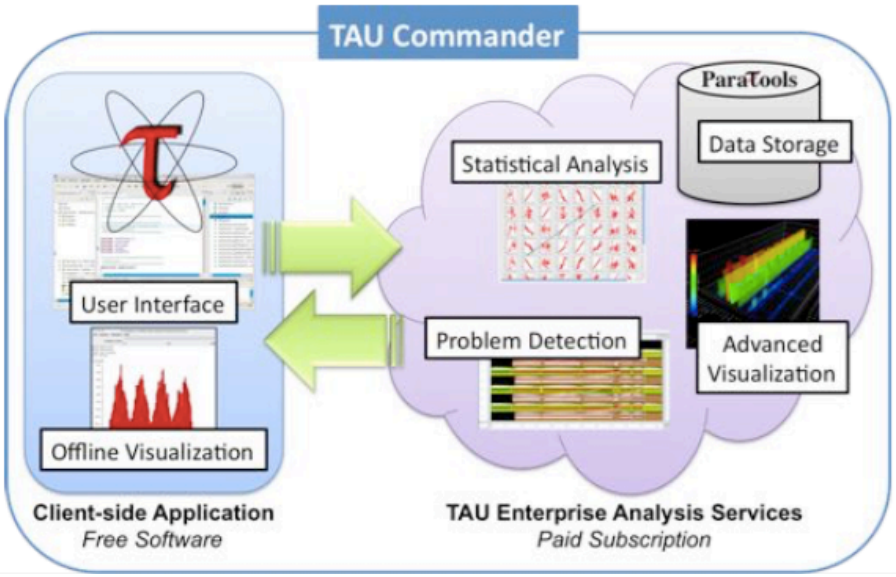
- Low overhead
- Comprehensive
- Hardware counters and derivative metrics
- Inclusive vs. exclusive
- Variety of meta-tools for sophisticated analysis
- …

Tool overheads (sampling only)

- SimplifiedCalo
  - TAU and HPCToolkit:
    - callpaths
    - sampling period=10,000µs

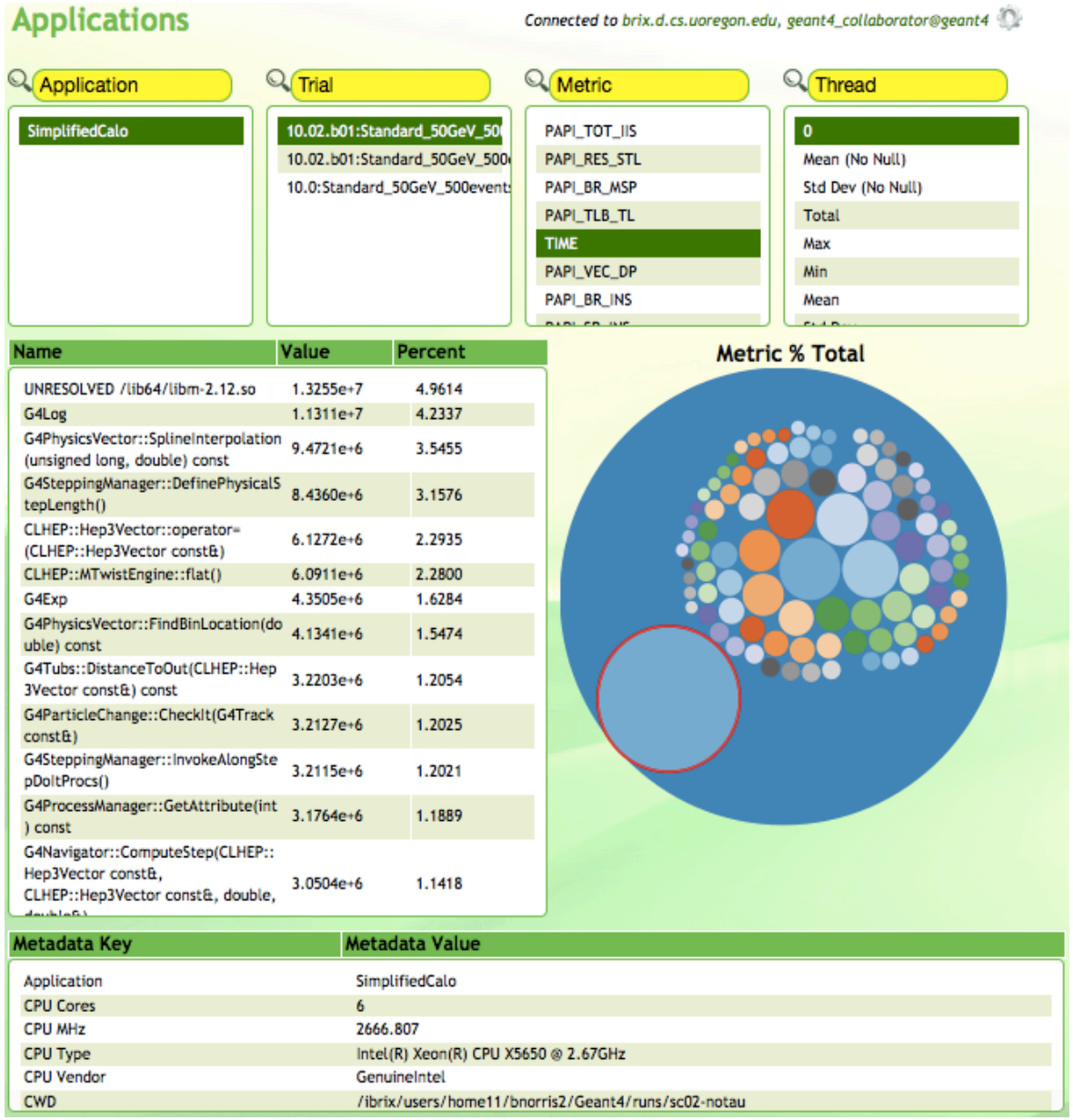
Tool	HEP Event, 50 events	HEP Event, pythia_pp_ttbar (1000 events)
No profiling	241	458
FAST	237	452
HPCToolkit	245	470
TAU sampling	241	459

TAU Commander



Ex: TAU for Geant4

- Interactive canvas
  - Application
  - Trial
  - Metric (PAPI HWC)
  - Thread (multi/many)
- DB-based analysis
- Working in progress
  - Add more analysis
  - Add display options



Memory Leakage Monitoring

S. Y. Jun (Fermilab), G. Cosmo (CERN), A. Dotti (SLAC)

20th Geant4 Collaboration Meeting at Fermilab

Sept. 28 - Oct. 2, 2015

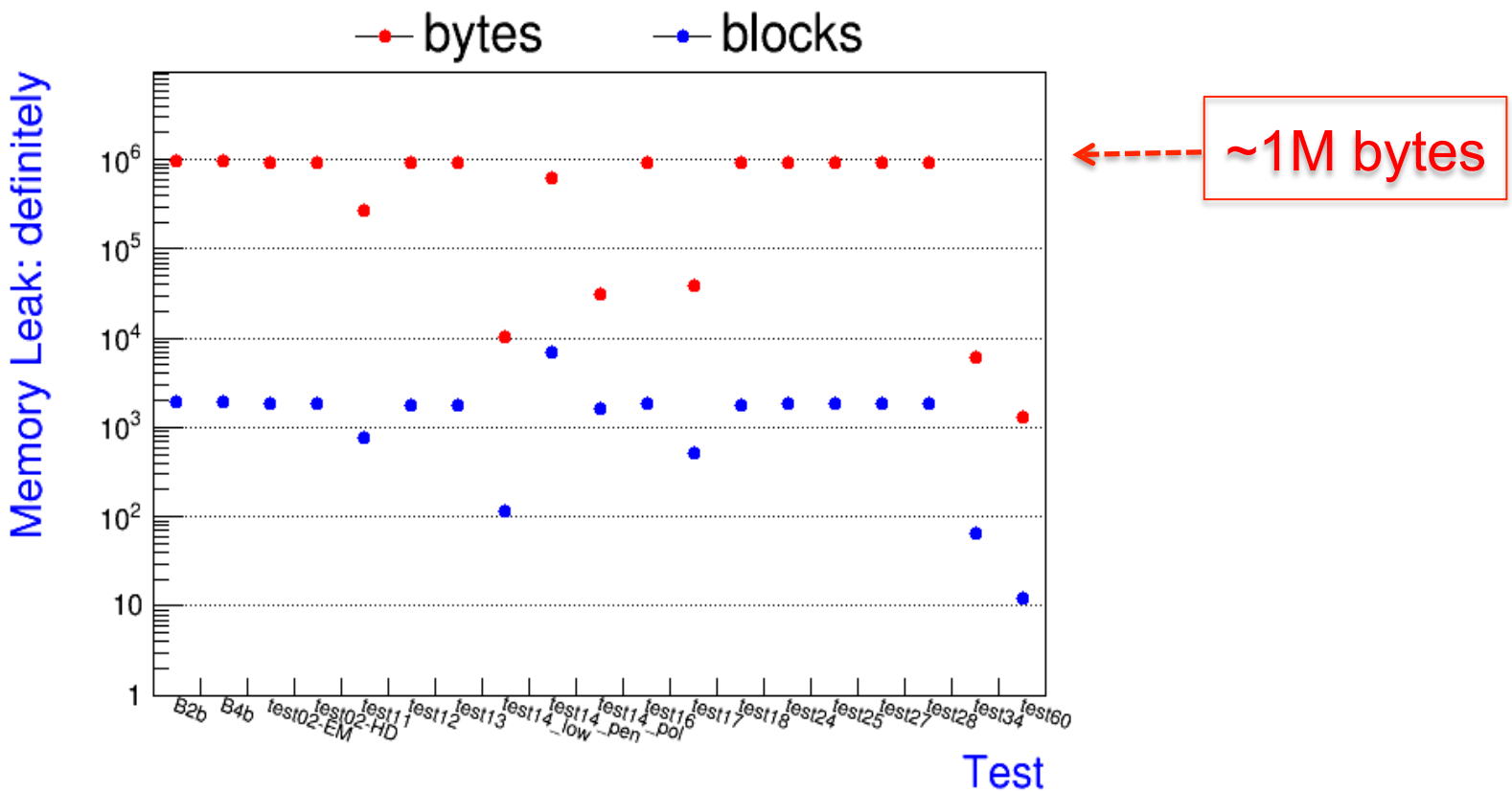
Memory leak

- Leaks from Geant4? - relatively clean. Two distinct types:
  - Memory allocated at initialization, but not explicitly released at the end of program (the majority of the cases, less critical)
  - Memory allocated within the event loop, but not freed (the most critical and relevant for production runs in the experiments)
- Problem Statement:
  - Indication of a poor design for ownership or lifetime of objects
  - Reduce existing memory leaks
  - Monitor newly introduced leaks

- Tools
  - Igprof (a low-overhead memory profiler - memory footprints)
  - Valgrind (a great tool for memcheck, but too slow - complete)
  - Coverity (a static code analysis)
  - a custom monitoring tool (under developing - efficient)

Valgrind Tests: (ex: Geant4 10.2.beta)

- Output: /afs/cern.ch/sw/geant4/dev/QA_tools/Valgrind/logs/
- Definitely Lost: no pointer to the block can be found (i.e, lost the pointer at the earlier point) – 19 test for major releases



- Geant4 code being released is relatively clean

Summary of Coverity Analysis

- Static analysis: http://coverity.cern.ch/ (289 issues under G4)
- Two types of resource leaks (39) under the Geant4 project
  - *new* on a data-member and does not free it
  - a *new* of an object in a method and no clear ownership

```
class G4Something;
class G4Class {
  G4Something* pointer;
    ~G4Class() { /*?? should I delete pointer??*/ }
    void set( G4Something* p) { pointer = p;}
    G4Something* get() const { return pointer; }
};
//Usage
  G4Something* smt = new G4Something;
  G4Class* cls = new G4Class();
  cls->set( smt );
//Who owns smt? Who should delete it?
```

- Use std::unique_ptr and move – make ownership explicitly

A Custom Memory Leak Monitor

- Check unreleased memory at the exit of an application
  - A very light leak monitoring tool (efficient) and complimentary to Valgrind (correctness)
- Push/pop memory alloc/dealloc during an application is running and dump undeleted pointers at the end program
  - Override new and delete (new[] and delete[]) with custom operators by adding/removing the address of the caller

```
void* operator new(size_t size, const std::nothrow_t&) _NOEXCEPT
{
    return new(size, (char*)__builtin_return_address(0),0);
}
```

  - builtin_return_address(0) : return address of the current function
  - addr2line(pointer) : convert the address of pointer to the file name and line number

A Custom Monitoring Tool (exampleB2b)

- Summary

```
LEAK SUMMARY from geant4.10.1.r06/examples/basic/B2/B2b/B2b.log
Number of leak found   = 4399
Number of leak from G4 = 2854
Unique number of lines = 125
Number of Geant4 files = 61
Total size of leakage  = 1281681 bytes
Total leak from Geant4 = 1037255 bytes
```

- List of file names and line numbers for undeleted objects

```
processes/hadronic/cross_sections/src/G4ElectroNuclearCrossSection.cc:2282
processes/hadronic/cross_sections/src/G4HadronCrossSections.cc:1231
processes/hadronic/models/cascade/cascade/src/G4Dineutron.cc:70
processes/hadronic/models/cascade/cascade/src/G4Diproton.cc:70
processes/hadronic/models/cascade/cascade/src/G4InuclSpecialFunctions.cc:149
```

- Run the memory leak monitor for each reference release
  - Select representative examples/tests
  - Post the list of potential leak (file names and line numbers)
  - Report a summary (and changes by the release version)

Thank You to All Contributors