Using the FAST *profiler to analyze Geant4*

Marc Paterno
6 October 2010

Project goals

- Nov. 2009 goal: to make it possible to routinely evaluate the performance of new releases of Geant4, and make the results publicly available.
- Technique is to use the CMS simulation program as the environment for evalution.
- On demand, we want to:
  - build a specific release of Geant4,
  - build a specific release of the CMS simulation,
  - run a standard sample repeatedly,
  - produce data for analysis, and
  - automate production of some standard plots.

Tools for data collection and analysis

- We use the FAST profiler for function call data
- The CMS timing service (collects event-by-event execution time)
- The FNAL CMS cluster (a mixture of AMD and Intel machines) for running jobs
- Data are organized to allow import into analysis tools, or into a RDBMS
- We use R (`http://www.r-project.org`) for analysis

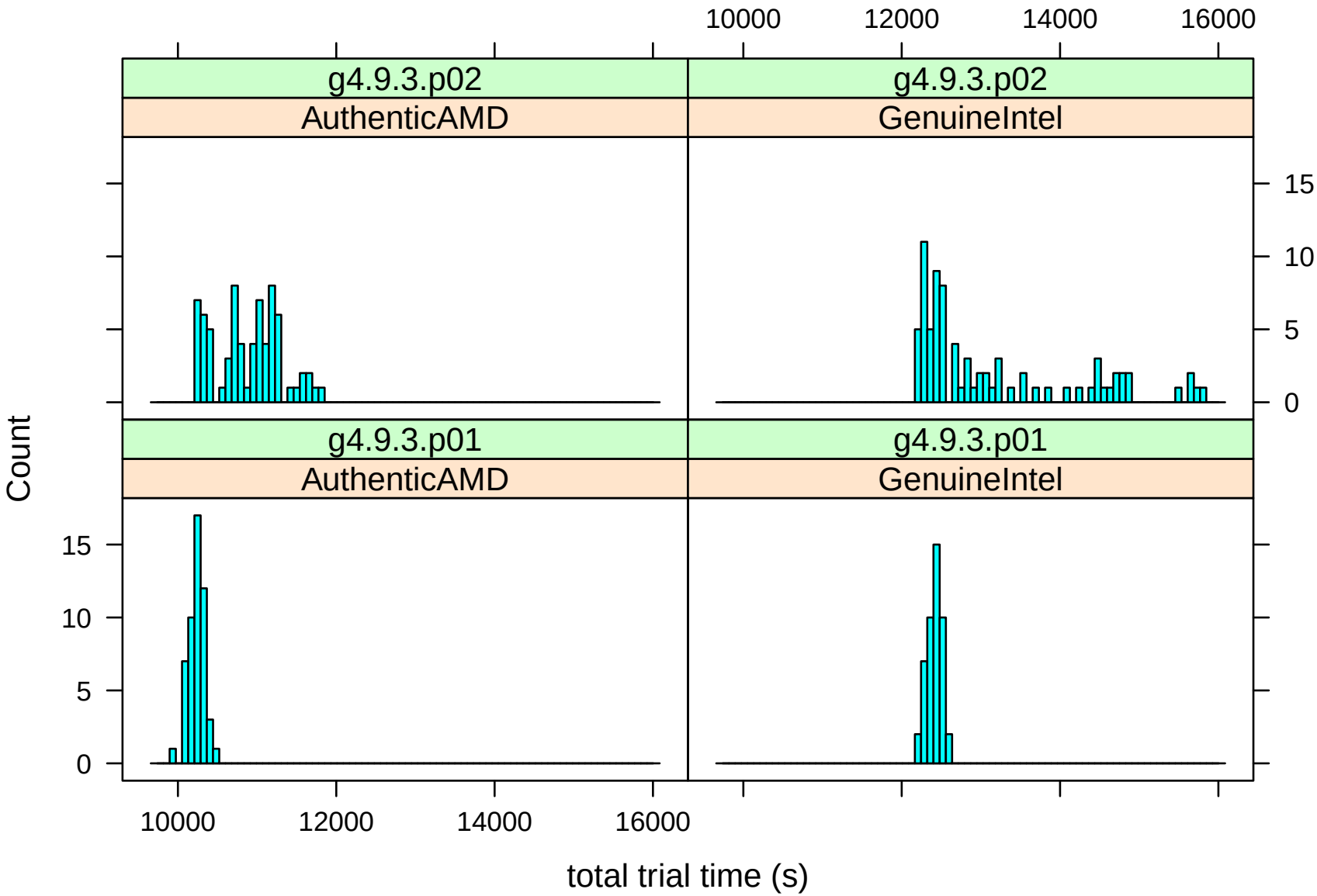
Current status

- FAST profiler is released (public beta 4 is current)
- Build system automation is done
- Campaign[1] automation is done
- Standard analysis scripts are working
- Import of data into RDBMS is not yet implemented; we analyze data directly from campaign output files

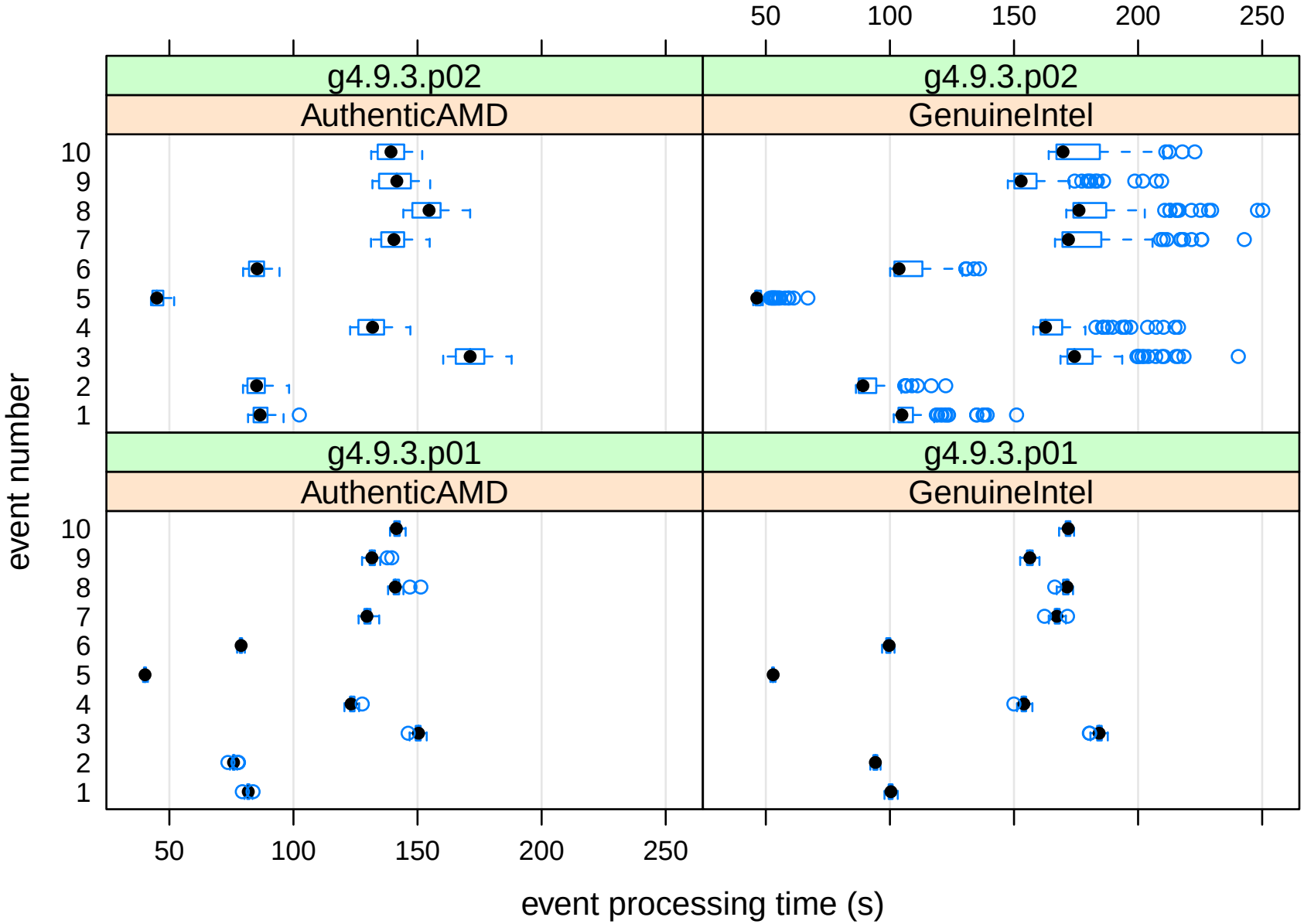
[1] A set of jobs run using one version of code and identical configurations

Trial timing distributions

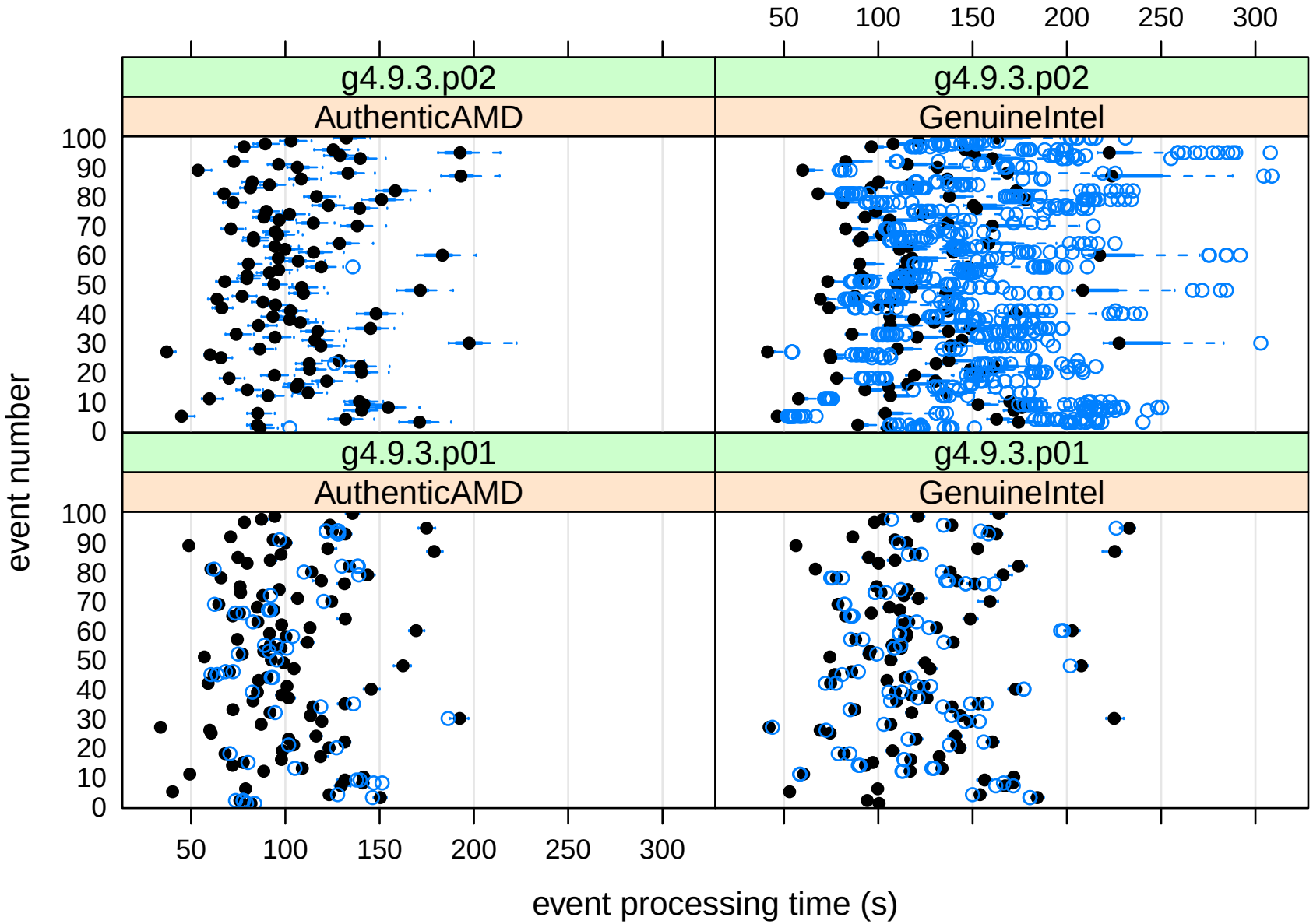
- Geant4 version 9.3.p01 and p02; AMD and Intel hardware



Event timing distributions (first 10 events)

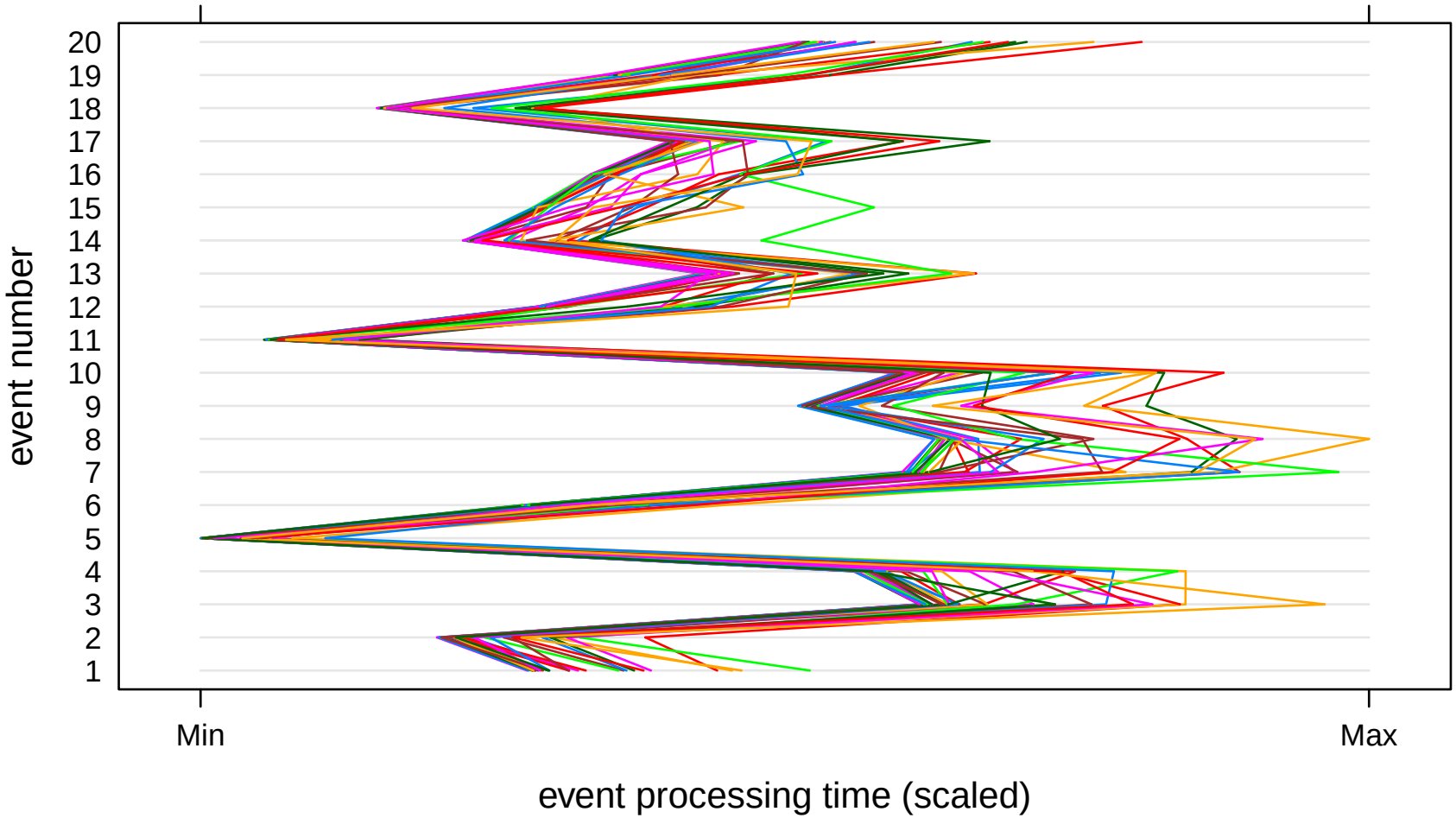


Event timing distributions (all events)



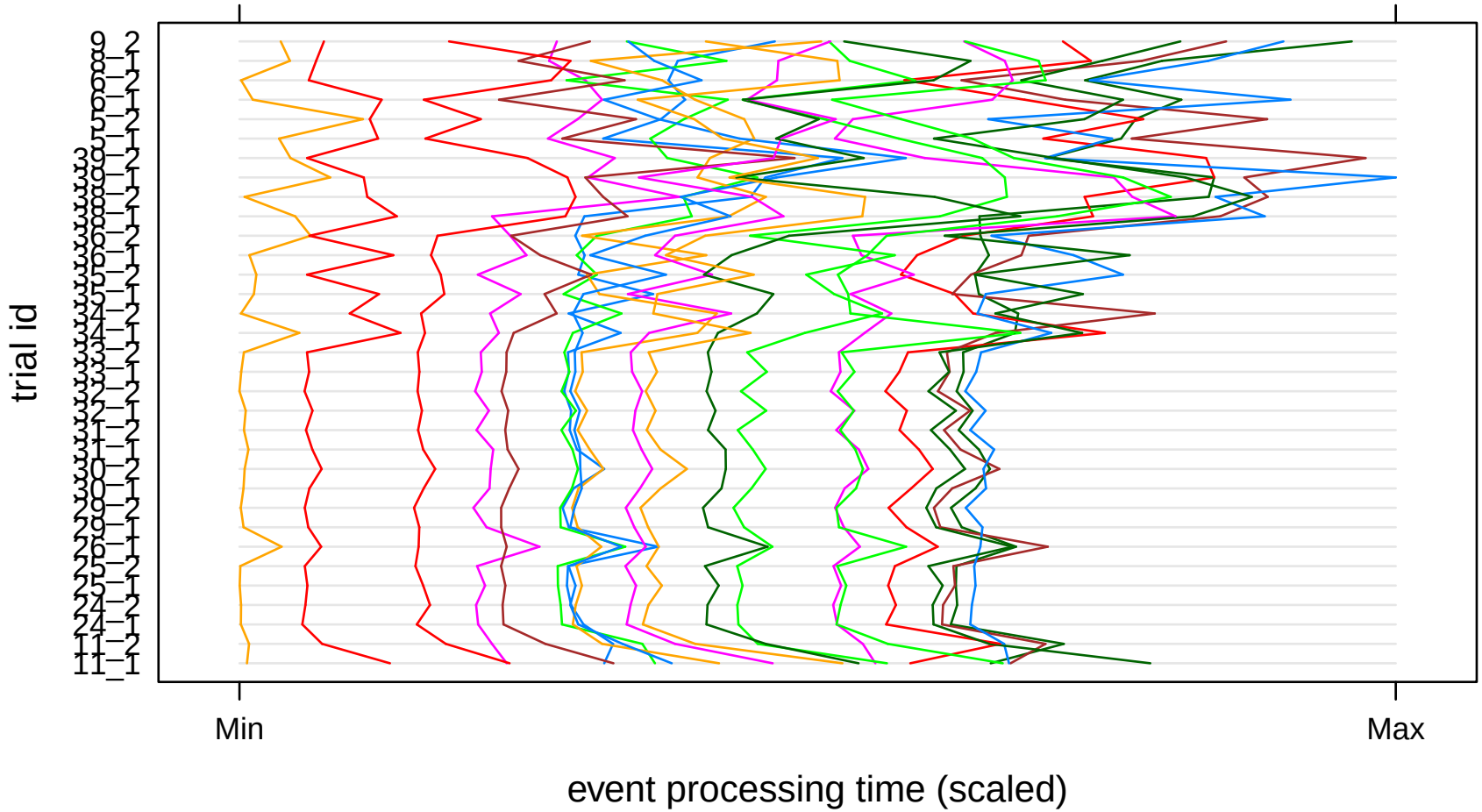
Variation of event time, threaded by trial

Select only Intel-based Geant4 9.3.p02 samples (upper right panel), and trace individual trials through the first 20 events



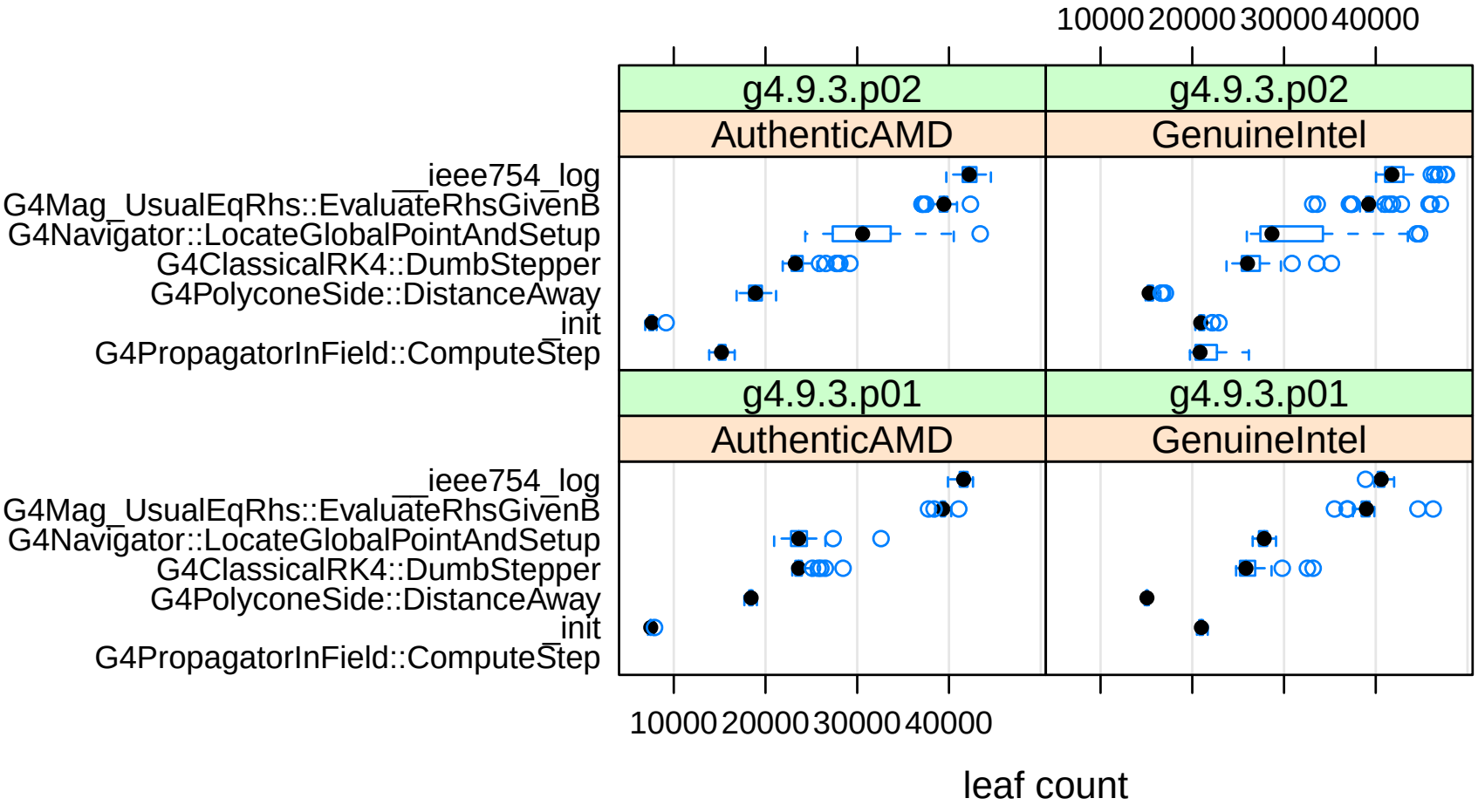
Variation of event time, threaded by event

Now view the same data the opposite way, concentrating on how each event is handled by the different trials



Further investigation is needed to understand these data.

Most time-consuming functions



- `G4PropagatorInField::ComputeStep` appears in p02, but not p01. We have not yet investigated. It uses $\sim 1.5\%$ of the program time.

Purpose

The Flexible Analysis and Storage Toolkit (FAST) is a set of tools designed to help understand and improve the performance—primarily the speed—of singly-threaded computer programs written in C++, C or Fortran. It has components for the collection, analysis, and display of performance data.

The tools in the suite are designed to allow the user access to as much of the measured data as possible, and to customize his view of the data. They are designed for exploratory data analysis, because understanding the performance of large and complex programs is a task that requires as much creativity as many scientific analyses.

Scope

FAST includes several tools and features:

- a sampling-based profiler for collecting measurements,
- tools for extracting call path, function call and library call information from the raw data,
- tools for the collection of information about the environment in which the profiled program is run,
- tools for graphical display of call paths, and
- a documented file format in which profiling data are written, to facilitate statistical analysis of profiling data.

Analysis tools are still under development.

FAST can profile multi-process programs, but not multi-threaded programs.

Getting started

- Obtain and install the prerequisites
  - Ruby (`http://www.ruby-lang.org`)
  - Graphviz (`http://www.graphviz.org`), for call graph visualization
  - Optionally, `ps2pdf` (from `http://www.ghostscript.com`), for PDF output of call graphs
  - Optionally, `R` (`http://www.r-project.org`) for analysis of data
  - All prerequisites are available for Linux, Mac OS X and Windows
- Download and build FAST (`https://cdcvs.fnal.gov/redmine/projects/fast`)

Collecting data

```
bash$ profrun
   Usage: profrun [ profrun options ... ] program [ program options ... ]
   where program is the program to be profiled and options are indicated
   below.

Examples: profrun examples/ex01/Linux.x86_64/ex01
          profrun -s examples/ex01/Linux.x86_64/ex01

Profrun options
    -h or -help       Print this help message.
    -v or -version    Print SimpleProfiler version.
    -s or -sar        Run program while collecting SAR data (Linux only).
    -n or -numactl    Turn on the use of numactl (Linux only).
    -m or -memory[=<timeout>]
                       Turn on the use of smaps monitoring. The optional
                       argument specifies the sampling interval (default 5).
    -setsid           Make the invoked program its own session leader

Program options
    Any options used by the program you wish to profile.
```

Output files

- All filenames have the format `profdata_<n>_<m>*`
- *n* and *m* are identifying process ids, usually the same
- The most important are the `names`, `paths` and `libraries` files
- Most output files are human-readable tab-separated text

A full description of the output files is provided in the **FAST Users's Manual**

Names file contents

For each function seen, we record:

- A unique function id
- The address of the function
- The leaf, total, and path count for the function, and the leaf and path fractions
- The library in which the function is found
- The mangled and unmangled names of the function

Definition

The leaf count for a function is the number of samples in which that function was observed at the top of the call stack.
The path count for a function is the number of samples in which that function was observed anywhere in the call stack.
The total count for a function is the number of times that function was observed in the call stack.

Libraries file contents

For each library, we record:

- The full path to the library (a unique identifier)
- The "short name" for the library
- The sum of the leaf counts of all functions belonging to this library

Paths file contents

Observed call stack addresses are translated into function names; the call stack thus yields a sequence of function names.

Definition

Each distinct sequence of function names observed is a path.

For each observed path we record:

- A unique identifier for that path
- The number of times the path was observed
- The ids of the functions comprising the path

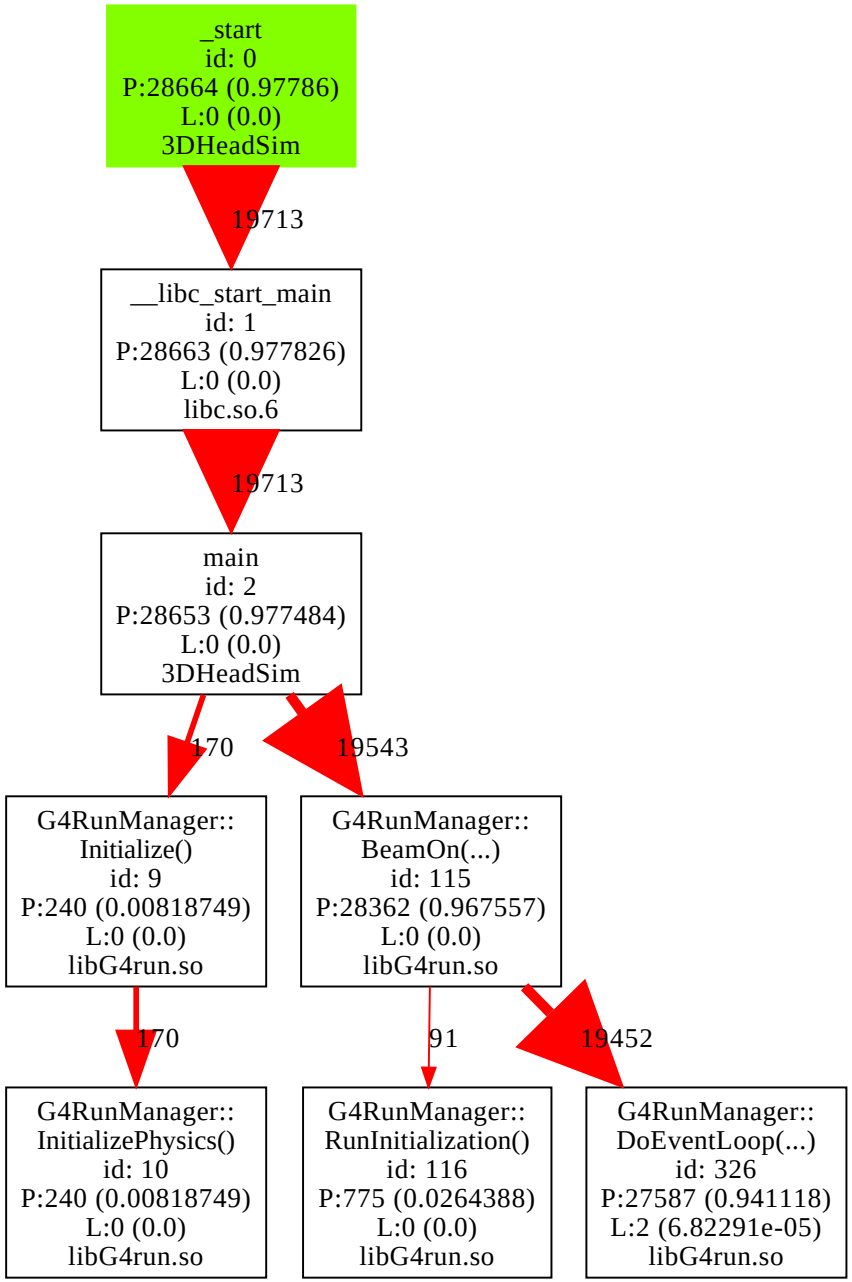
Example of use

To show an example of use, I collected data from 114 runs of a Geant4-based medical simulation, of a proton beam incident on a simulation of a human head.

- Geant4 version 9.3p01
- OS: Scientific Linux SLF release 5.3 (Lederman)
- Compiler: gcc (GCC) 4.1.2 20080704 (Red Hat 4.1.2-46)
- Processor: Quad-Core AMD Opteron(tm) Processor 2352

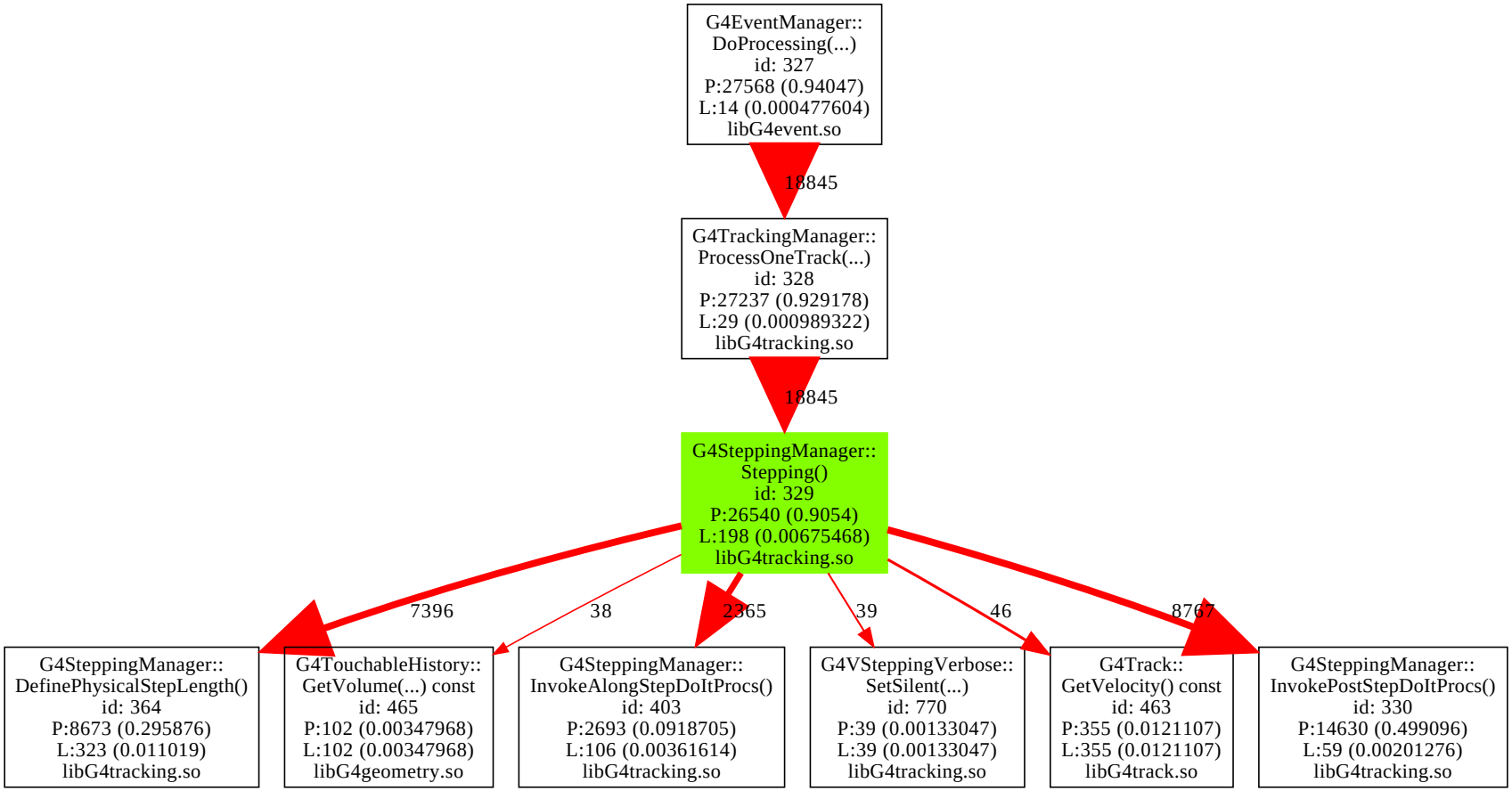
`3DHeadSim` program courtesy of:
Scott Penfold
Centre for Medical Radiation Physics
University of Wollongong
NSW, Australia.

Call path analysis



- Call graphs are produced by `profgraph`
- Use `profgraph -h` for complete help
- Call paths can be trimmed (removing infrequently sampled paths) and truncated (decreasing the width of the window into the call stack plotted)
- Call paths can be centered on any function of interest; that function is shown in green.

Another call path



This was created with:

```
profgraph -n profdata_22658_22658 329 2 1 25
```

Conclusion

- The FAST profiler and associated tools are now available.
- Statistically significant data samples and appropriate analysis tools are necessary for serious comparisons.
- Freely available tools aid in exploring the data.
- Tools for "standard" analysis of new releases are ready.
- $\sim 85\%$ of long-running jobs complete:
  - some failures in `libunwind`,
  - some failures in our postprocessing of data,
  - some failures in profiled application,
  - some failures due to cluster environment issues.
- Almost "turnkey" operation now possible.
- Web-publishing of results still must be done manually.
- Validation of results is not automated; we do not want to publish any mischaracterization of Geant4 or CMS code.

`https://cdcvs.fnal.gov/redmine/projects/fast`

Thank you.
The remainder of this file contains backup slides.

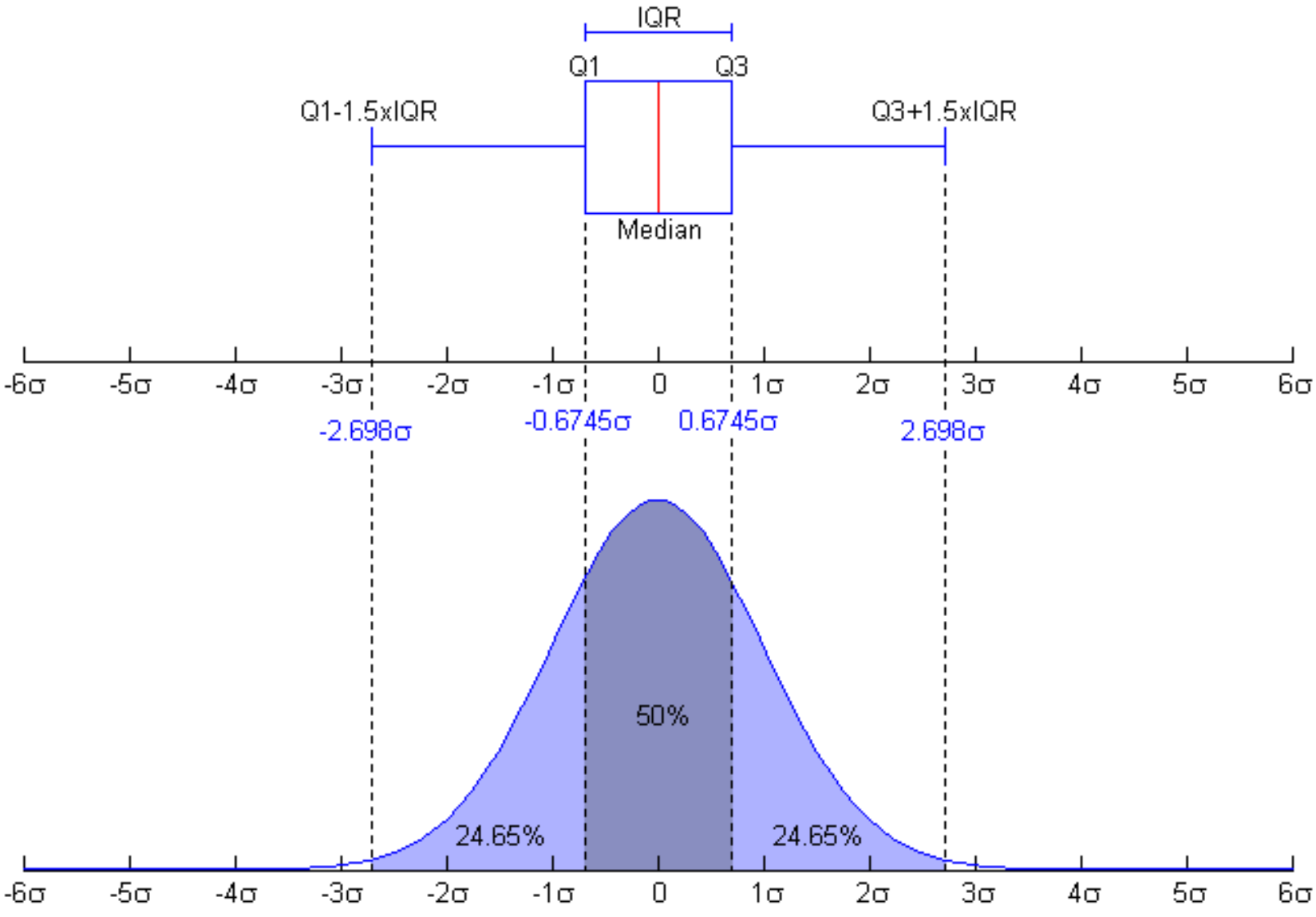
Supported platforms

- Currently, data collection is only supported on Linux; we test using Scientific Linux 5
- Analysis is supported anywhere the necessary tools work (Linux, Windows, Mac OS X)
- All profiler output files (including our binary data files) are portable between platforms

Hardware used

Name	Description	Memory
AMD	Quad-Core AMD Opteron(tm) Processor 2389	24 GB
Intel	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz	16 GB

The box-and-whisker plot



Another look: leading path counts