

Performance Analysis on Blue Gene Q with HPCToolkit

John Mellor-Crummey
Department of Computer Science
Rice University

<http://hpctoolkit.org>



Acknowledgments

- **Funding**

- **DOE Office of Science SciDAC-2 (expired)**
 - **Center for Scalable Application Development Software**
Cooperative agreement number DE-FC02-07ER25800
 - **Performance Engineering Research Institute**
Cooperative agreement number DE-FC02-06ER25762
- **Sandia National Laboratory**

- **Project team**

- **Research Staff**
 - **Laksono Adhianto, Mike Fagan, Mark Krentel**
- **Students**
 - **Xu Liu, Milind Chabbi, Karthik Murthy**
- **Collaborator**
 - **Nathan Tallent (PNNL)**
- **Summer Interns**
 - **Michael Franco (Rice), Reed Landrum (Stanford), Sinchan Banerjee (MIT)**
- **Alumni**
 - **Gabriel Marin (ORNL), Robert Fowler (RENCI), Nathan Froyd (Mozilla)**

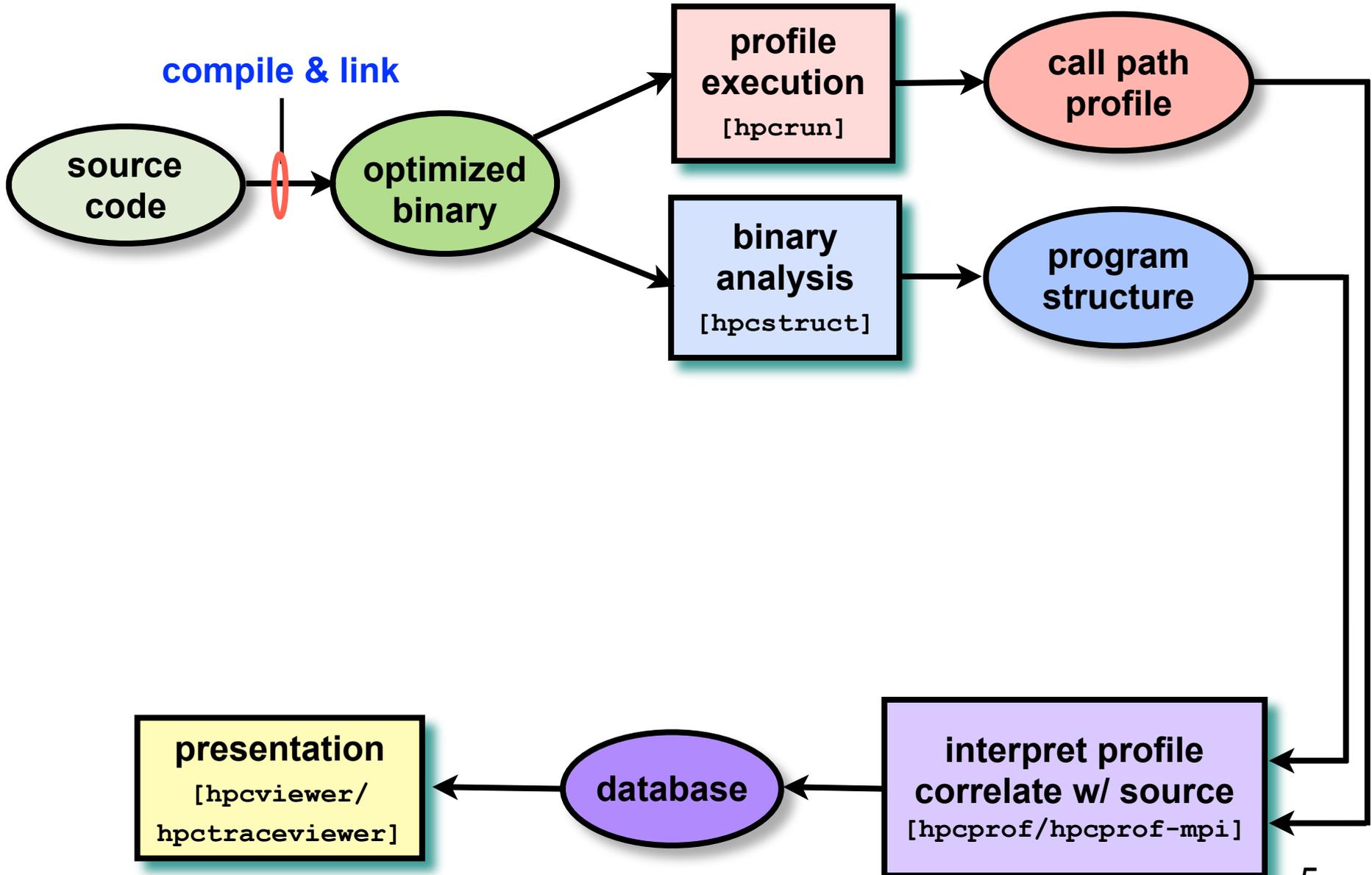
Outline

- **HPCToolkit overview**
- **New developments**
 - monitoring and attribution of L2Unit activity
 - a new emerging approach for performance analysis of OpenMP
- **Next steps**
- **Using HPCToolkit on Blue Gene/Q at ALCF**

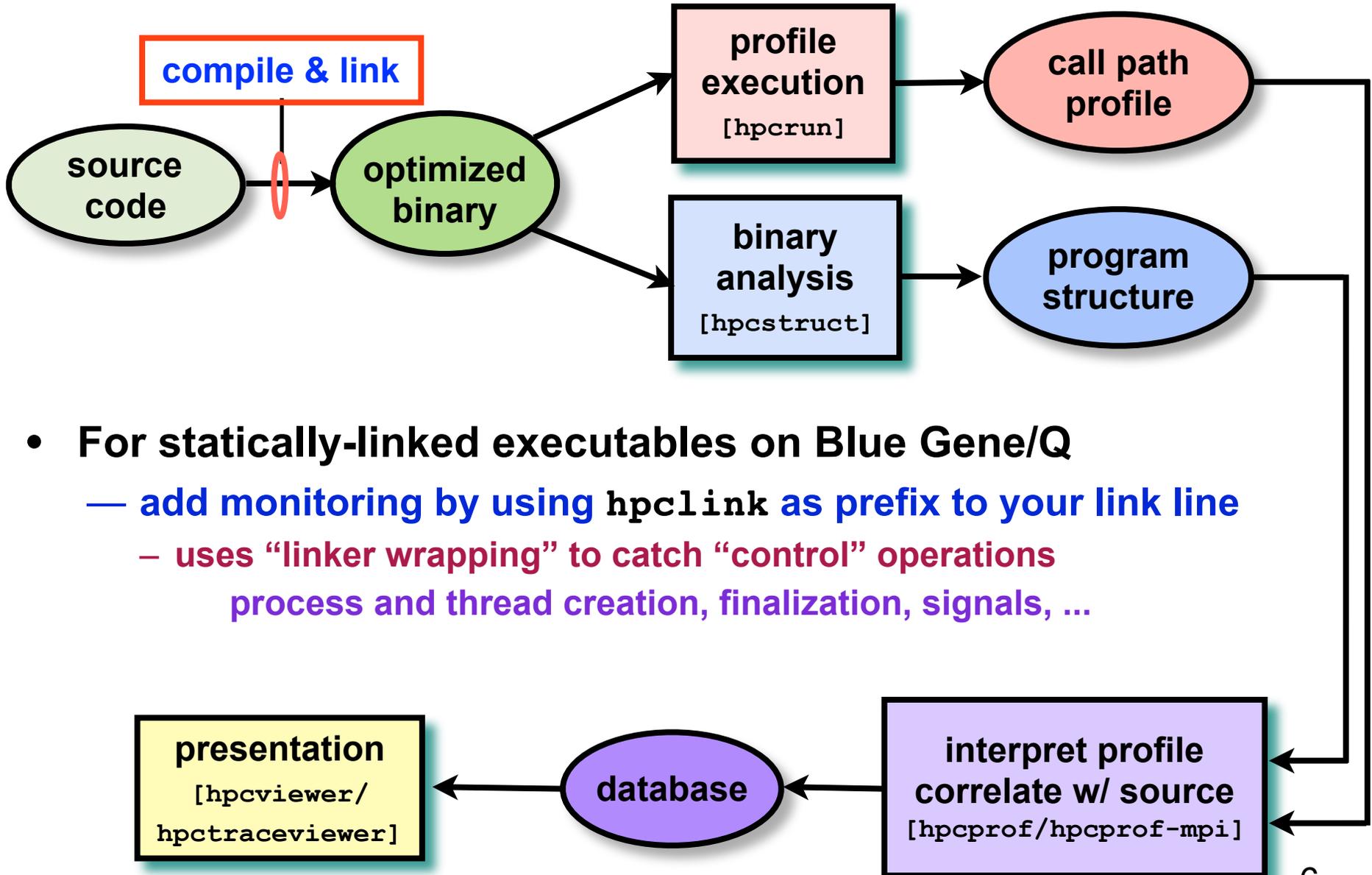
Rice University's HPCToolkit

- **Employs binary-level measurement and analysis**
 - observe **fully optimized**, **dynamically linked executions**
 - support **multi-lingual codes** with external binary-only libraries
- **Uses sampling-based measurement (avoid instrumentation)**
 - **controllable overhead**
 - **minimize** systematic error and avoid blind spots
 - enable data collection for **large-scale parallelism**
- **Collects and correlates multiple derived performance metrics**
 - **diagnosis typically requires more than one species of metric**
- **Associates metrics with both static and dynamic context**
 - **loop nests**, **procedures**, **inlined code**, **calling context**
- **Supports top-down performance analysis**
 - **identify costs of interest and drill down to causes**
 - **up and down call chains**
 - **over time**

HPCToolkit Workflow

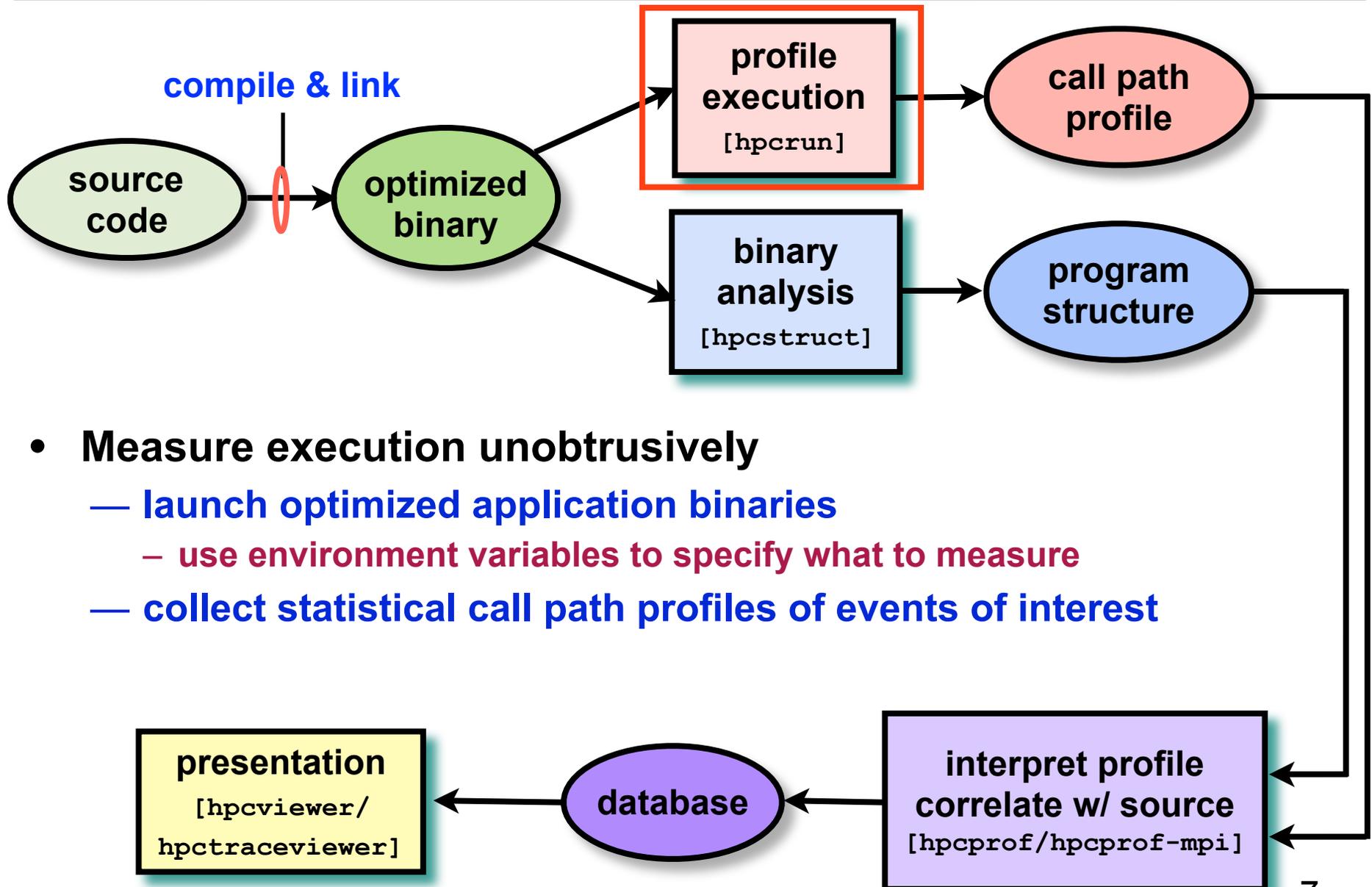


HPCToolkit Workflow



- For statically-linked executables on Blue Gene/Q
 - add monitoring by using `hpcLink` as prefix to your link line
 - uses “linker wrapping” to catch “control” operations
process and thread creation, finalization, signals, ...

HPCToolkit Workflow



- **Measure execution unobtrusively**
 - **launch optimized application binaries**
 - **use environment variables to specify what to measure**
 - **collect statistical call path profiles of events of interest**

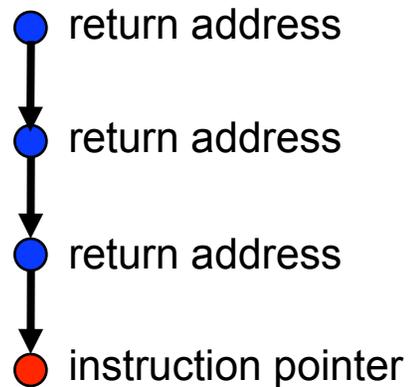
Call Path Profiling

Measure and attribute costs in context

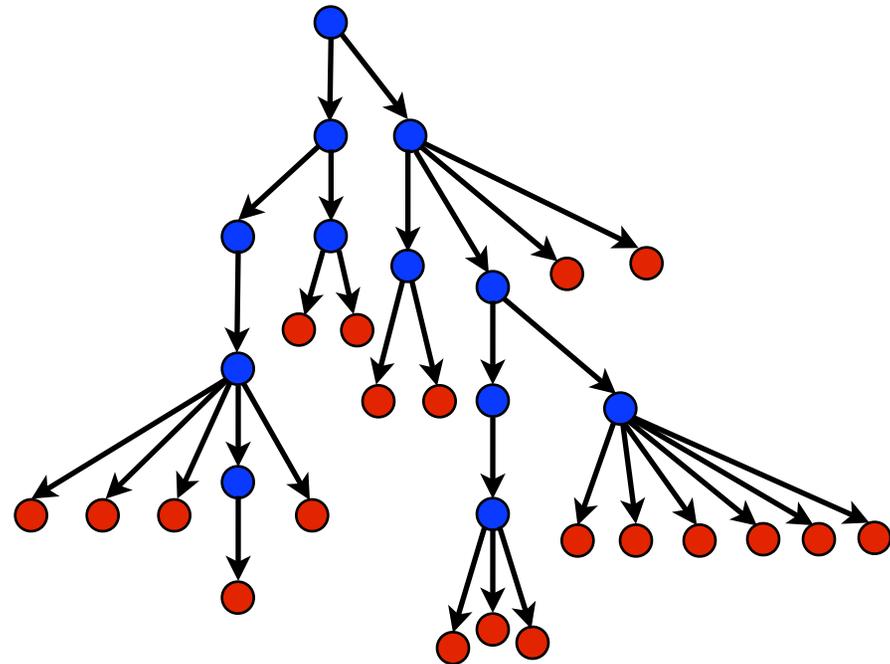
sample timer or hardware counter overflows

gather calling context using stack unwinding

Call path sample

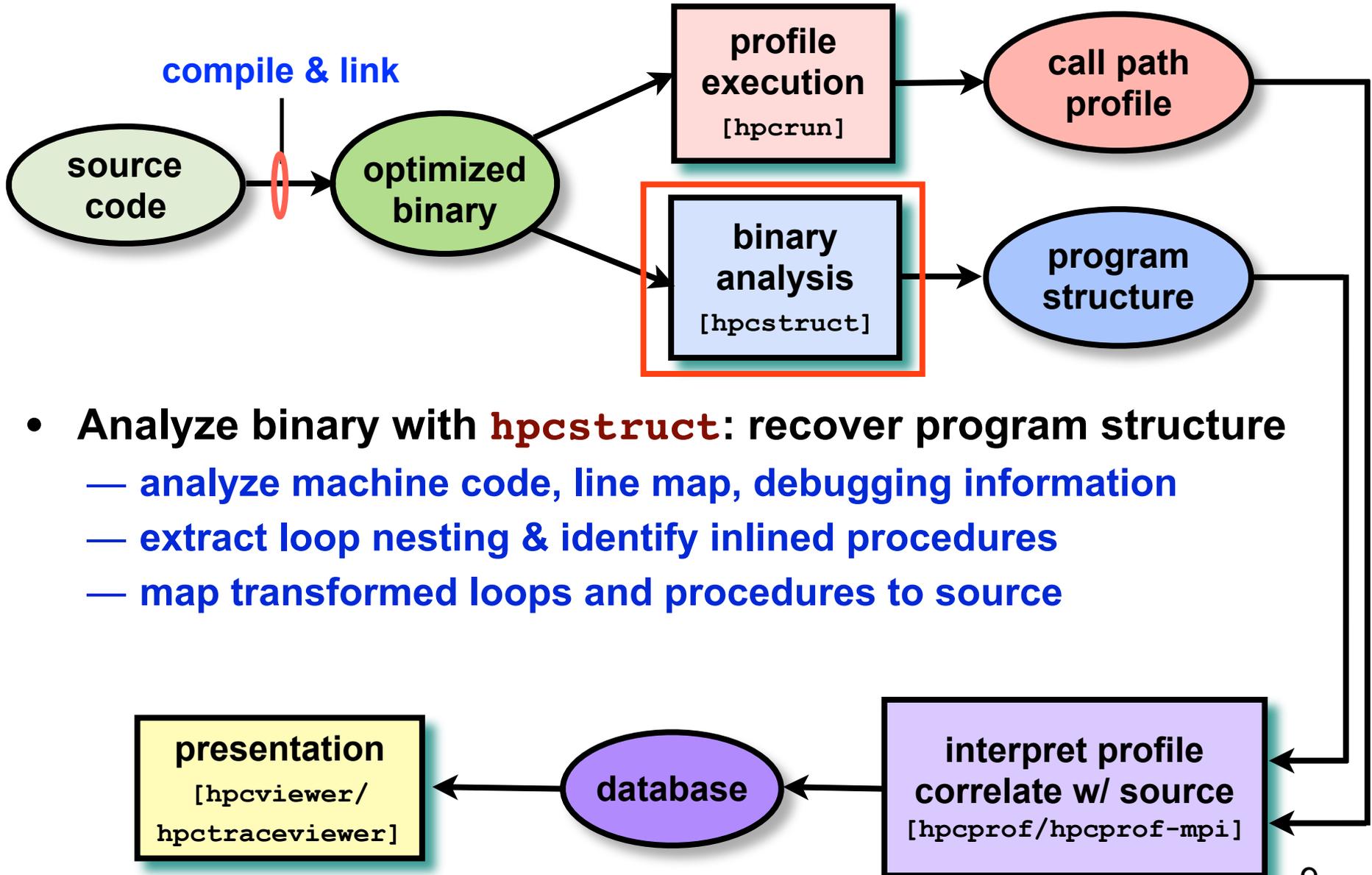


Calling context tree



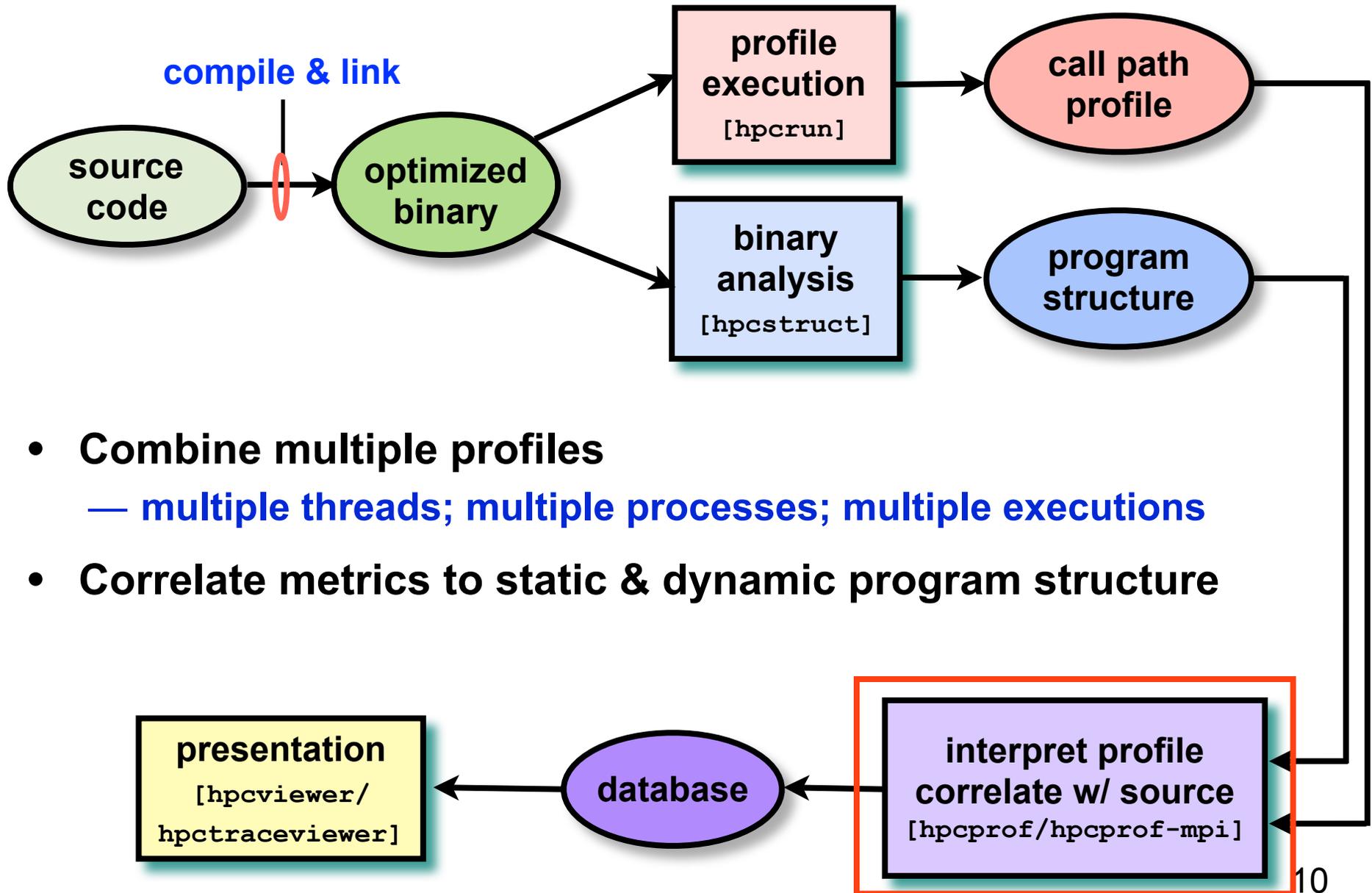
Overhead proportional to sampling frequency...
...not call frequency

HPCToolkit Workflow

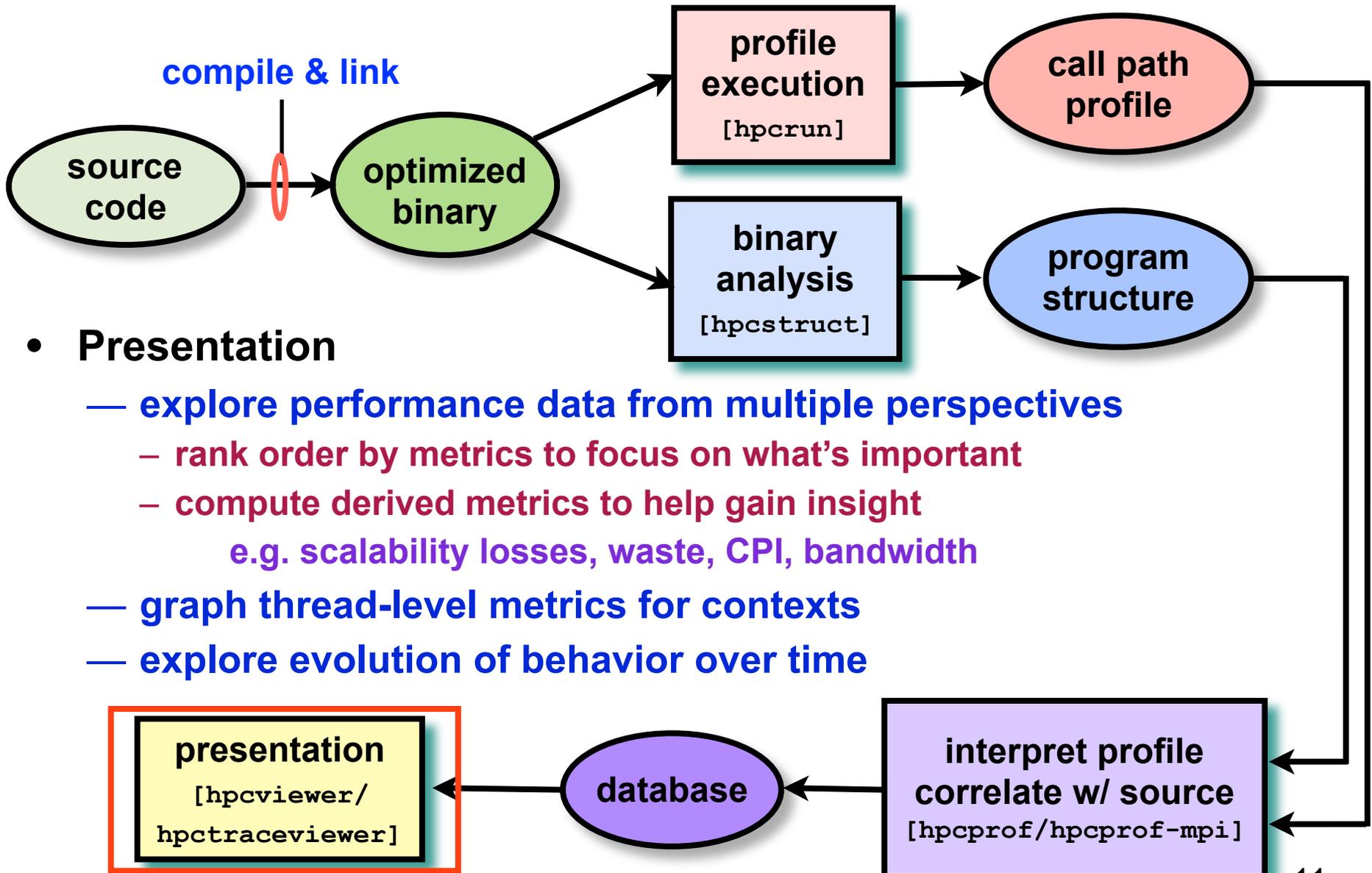


- Analyze binary with **hpcstruct**: recover program structure
 - analyze machine code, line map, debugging information
 - extract loop nesting & identify inlined procedures
 - map transformed loops and procedures to source

HPCToolkit Workflow



HPCToolkit Workflow



- **Presentation**

- explore performance data from multiple perspectives
 - rank order by metrics to focus on what's important
 - compute derived metrics to help gain insight
 - e.g. scalability losses, waste, CPI, bandwidth
- graph thread-level metrics for contexts
- explore evolution of behavior over time

Code-centric Analysis with hpcviewer

The screenshot displays the hpcviewer interface for the application 'amrGodunov3d.Linux.64.CC.ftn.OPTHIGH.MPI.ex'. The top pane shows the source code for 'PatchGodunov.cpp' with a red box highlighting the 'source pane'. A callout box on the right lists 'costs for' with three items: 'inlined procedures' (green), 'loops' (red), and 'function calls in full context' (blue). Below the code is a 'view control' bar with buttons for 'Calling Context View', 'Callers View', and 'Flat View'. A 'metric display' bar contains icons for flame, flame with 'f00', and a bar chart. The main area is a 'metric pane' table with columns for 'Scope', 'WALLCLOCK (us):Sum (l)', 'WALLCLOCK (us):Mean (l)', and 'WALLCLOCK'. A 'navigation pane' is overlaid on the table, showing a tree of scopes with red and green boxes highlighting specific nodes like 'loop at amrGodunov.cpp: 186' and 'inlined from AMR.cpp: 604'.

source pane

costs for

- inlined procedures
- loops
- function calls in full context

view control

metric display

navigation pane

metric pane

Scope	WALLCLOCK (us):Sum (l)	WALLCLOCK (us):Mean (l)	WALLCLOCK
Experiment Aggregate Metrics	1.92e+11 100 %	1.80e+08	
main	1.92e+11 100 %	1.80e+08	
282: amrGodunov()	1.87e+11 97.4%	1.75e+08	
loop at amrGodunov.cpp: 186	1.77e+11 92.1%	1.66e+08	
loop at amrGodunov.cpp: 214	1.77e+11 92.1%	1.66e+08	
216: AMR::run(double, int)	1.77e+11 92.1%	1.66e+08	
inlined from AMR.cpp: 604	1.77e+11 92.1%	1.66e+08	
loop at AMR.cpp: 615	1.77e+11 92.1%	1.66e+08	
loop at AMR.cpp: 622	1.77e+11 92.1%	1.66e+08	
654: AMR::timeStep(int, int, bool)	1.77e+11 92.1%	1.66e+08	
inlined from AMR.cpp: 794	1.77e+11 92.1%	1.66e+08	
loop at AMR.cpp: 943	1.77e+11 92.0%	1.66e+08	
953: AMR::timeStep(int, int, bool)	1.77e+11 92.0%	1.66e+08	
inlined from AMR.cpp: 794	1.77e+11 92.0%	1.66e+08	
loop at AMR.cpp: 943	1.73e+11 90.3%	1.62e+08	
953: AMR::timeStep(int, int, bool)	1.73e+11 90.3%	1.62e+08	
inlined from AMR.cpp: 794	1.73e+11 90.3%	1.62e+08	
903: AMRLevelPolytropicGas::advance()	1.73e+11 90.3%	1.62e+08	
919:BoxLayout::size() const	5.37e+06 0.0%	5.04e+03	
911: AMRLevelPolytropicGas::computeDt()	2.04e+05 0.0%	1.91e+02	
AMR.cpp: 795	2.40e+04 0.0%	2.25e+01	
967: AMRLevelPolytropicGas::postTimeStep()	1.20e+04 0.0%	1.12e+01	
801: std::ostream& std::ostream::_M_insert<long>(long)	1.20e+04 0.0%	1.12e+01	

Scalability Analysis

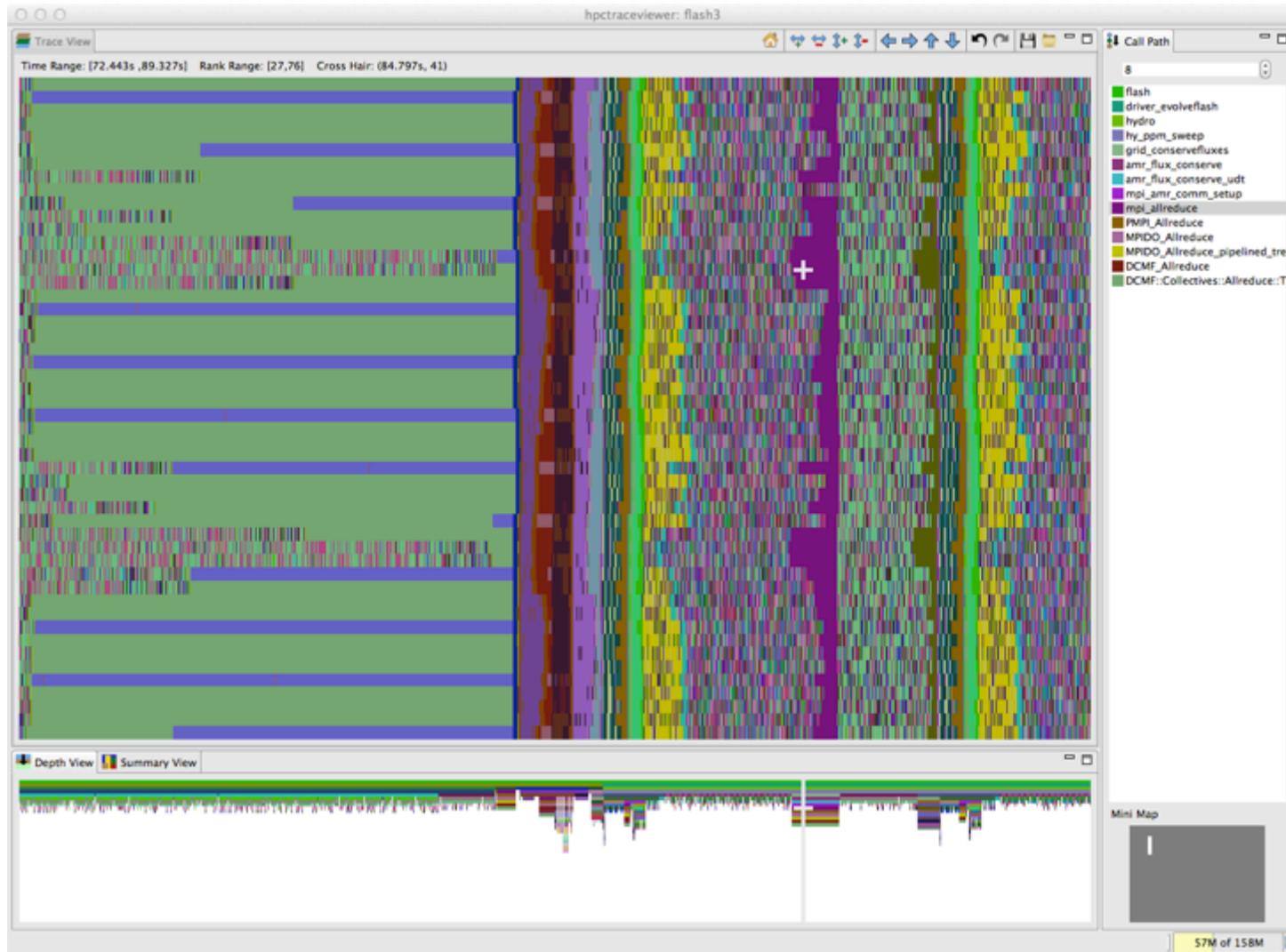
- Difference call path profile from two executions
 - different number of nodes
 - different number of threads
- Pinpoint and quantify scalability bottlenecks within and across nodes

The screenshot shows the hpcviewer interface. The top pane displays code from 'local_tree_build.F90' with lines 212-217 highlighted in blue. A red box with a red border contains the text: "21% of scaling losses caused by passing data around a ring of processors". A red arrow points from this box to the code and another red arrow points from the box to a table entry.

Scope	% scalability loss	256/WALLCLOCK (u)
Experiment Aggregate Metrics	2.46e+01 100 %	5.07e+08
flash	2.46e+01 100 %	5.07e+08
driver_evolveflash	1.41e+01 57.5%	4.46e+08
driver_initflash	1.04e+01 42.5%	6.02e+07
grid_initdomain	8.58e+00 34.9%	3.45e+07
gr_expanddomain	8.58e+00 34.9%	3.45e+07
loop at gr_expandDomain.F90: 119	6.85e+00 27.9%	3.42e+07
amr_refine_derefine	5.56e+00 22.6%	2.87e+06
amr_morton_process	5.45e+00 22.2%	9.75e+05
find_surrblks	5.18e+00 21.1%	8.40e+05
local_tree_build	5.18e+00 21.1%	8.25e+05
loop at local_tree_build.F90: 211	5.18e+00 21.1%	8.25e+05
loop at local_tree_build.F90: 216	5.18e+00 21.1%	8.25e+05
loop at local_tree_build.F90: 286	1.14e+00 4.6%	2.55e+05
pmpi_sendrecv_replace	5.47e-01 2.2%	5.00e+04

Time-centric Analysis with hpctraceviewer

Load imbalance among threads appears as different lengths of colored bands along the x axis



Measurement & Attribution of L2 Activity

- **L2Unit measurement capabilities**
 - e.g., counts load/store activity
 - node-wide counting; not thread-centric
 - global or per slice counting
 - supports threshold-based sampling
 - samples delivered late: about 800 cycles after threshold reached
 - each sample delivered to ALL threads/cores
- **HPCToolkit approach**
 - attribute a share of L2Unit activity to each thread context for each sample
 - e.g., when using a threshold of 1M loads and T threads, attribute 1M/T events to the active context in each thread when each sample event occurs
 - best effort attribution
 - strength: correlate L2Unit activity with regions of your code
 - weakness: some threads may get blamed for activity of others

Emerging Analysis for OpenMP

- **Challenges**

- conventional profiling tools can only provide implementation-level view of OpenMP threads
 - master thread
 - worker thread
- no context available for computation performed by worker threads
- hard to understand causes of idleness
 - insufficient parallelism
 - poor load balance
 - waiting for critical sections or locks

- **New approach**

- leading development of OpenMP tools API - OMPT
 - provides sufficient hooks to address all three challenges
- prototype implementation of OMPT in IBM's emerging LOMP OpenMP runtime
- prototype implementation using LOMP in HPCToolkit

Blame Shifting from Symptoms to Causes

- **Approach**
 - **shift blame for idleness to code executing while other threads idle**
 - **undirected blame**
 - **directed blame**
- **Implementation of undirected blame shifting**
 - **callback at thread transitions idle ↔ working**
 - **maintain two global counters**
 - **thread created (or dedicated HW resources that are reserved)**
 - **number of threads that are working**
 - **idleness is the difference between the two counters**
 - **at a sample event**
 - **if the thread is actively working**
 - attribute a sample of work to the present context
 - attribute partial blame for idleness to the *present* context
 - **else, ignore the sample event**

Next Steps

- **Finish OpenMP support**
 - finalize OpenMP tools interface with standards committee
 - merge OpenMP support into trunk
- **Scale I/O strategy**
 - one file per node rather than one file per thread
- **Scale traceviewer**
 - split traceviewer into client server
 - server runs as a parallel program on vis cluster
 - client runs on your laptop
- **Explore automated analysis of time-centric data**
- **Data-centric analysis**
- **Resource-centric performance analysis**
 - within and across nodes

HPCToolkit at ALCF

- **ALCF systems**
 - [/soft/perftools/hpctoolkit/pkgs/hpctoolkit](#)
- **Man pages**
 - [/soft/perftools/hpctoolkit/pkgs/hpctoolkit/share/man](#)
- **ALCF guide to HPCToolkit**
 - <http://www.alcf.anl.gov/resource-guides/vesta-hpctoolkit>

Detailed HPCToolkit Documentation

<http://hpctoolkit.org/documentation.html>

- **Comprehensive user manual:**
 - <http://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf>
 - **Quick start guide**
 - **essential overview that almost fits on one page**
 - **Using HPCToolkit with statically linked programs**
 - **a guide for using hpctoolkit on BG/Q, BG/P, and Cray XT**
 - **The hpcviewer and hpctraceviewer user interfaces**
 - **Effective strategies for analyzing program performance with HPCToolkit**
 - **analyzing scalability, waste, multicore performance ...**
 - **HPCToolkit and MPI**
 - **HPCToolkit Troubleshooting**
 - **why don't I have any source code in the viewer?**
 - **hpcviewer isn't working well over the network ... what can I do?**
- **Installation guide**

Using HPCToolkit

- Add hpctoolkit's bin directory to your path
 - see earlier slide for HPCToolkit's HOME directory on your system
- Adjust your compiler flags (if you want full attribution to src)
 - add -g flag after any optimization flags
- Add hpclink as a prefix to your Makefile's link line
 - e.g. `hpclink mpixlf -o myapp foo.o ... lib.a -lm ...`
- See what sampling triggers are available on BG/Q
 - use hpclink to link your executable
 - launch executable with environment variable `HPCRUN_EVENT_LIST=LIST`
 - you can launch this on 1 core of 1 node
 - no need to provide arguments or input files for your program
they will be ignored

Collecting Performance Data on BG/Q

- **Collecting traces on BG/Q**
 - set environment variable `HPCRUN_TRACE=1`
 - use `WALLCLOCK` or `PAPI_TOT_CYC` as one of your sample sources when collecting a trace
- **Launching your job on BG/Q using hpctoolkit**
 - `qsub -A ... -t 10 -n 1024 --mode c1 --proccount 16384 \
--cwd `pwd` \
--env OMP_NUM_THREADS=2:\
 HPCRUN_EVENT_LIST=WALLCLOCK@5000:\
 HPCRUN_TRACE=1\
your_executable`

Monitoring Large Executions

- **Collecting performance data on every node is typically not necessary**
- **Can improve scalability of data collection by recording data for only a fraction of processes**
 - **set environment variable HPCRUN_PROCESS_FRACTION**
 - **e.g. collect data for 10% of your processes**
 - **set environment variable HPCRUN_PROCESS_FRACTION=0.10**

Digesting your Performance Data

- Use hpcstruct to reconstruct program structure
 - e.g. `hpcstruct your_app`
 - creates `your_app.hpcstruct`
- Correlate measurements to source code with hpcprof and hpcprof-mpi
 - run hpcprof on the front-end to analyze data from small runs
 - run hpcprof-mpi on the compute nodes to analyze data from lots of nodes/threads in parallel
- Digesting performance data in parallel with hpcprof-mpi
 - `qsub -A ... -t 20 -n 32 --mode c1 --proccount 32 --cwd `pwd` \`
`/soft/perftools/hpctoolkit/pkg/hpctoolkit/bin/hpcprof-mpi \`
`-S your_app.hpcstruct \`
`-l /path/to/your_app/src/+ \`
`hpctoolkit-your_app-measurements.jobid`

Analysis and Visualization

- **Use hpcviewer to open resulting database**
 - **warning: first time you graph any data, it will pause to combine info from all threads into one file**
- **Use hpctraceviewer to explore traces**
 - **warning: first time you open a trace database, the viewer will pause to combine info from all threads into one file**
- **Try our our user interfaces before collecting your own data**
 - **example performance data:**
<http://hpctoolkit.org/examples.html>

A Special Note About `hpcstruct` and `xlf`

- IBM's `xlf` compiler emits machine code for Fortran that has an unusual mapping back to source
- To compensate, `hpcstruct` needs a special option
 - `--loop-fwd-subst=no`
 - without this option, many nested loops will be missing in `hpcstruct`'s output and (as a result) `hpcviewer`