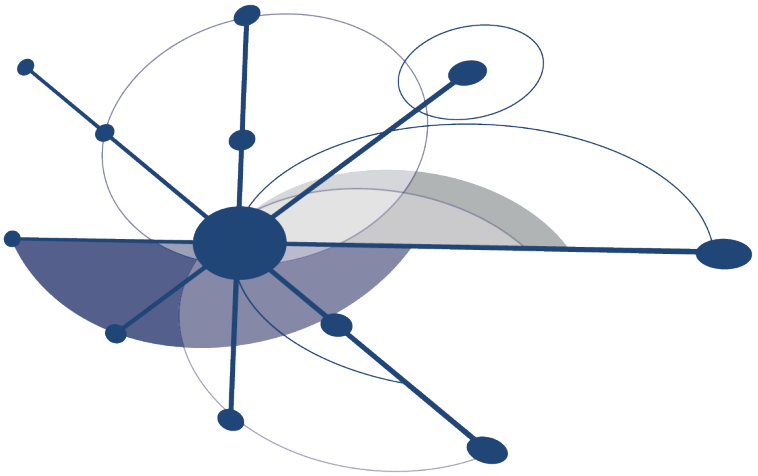
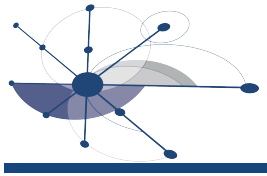


Center for Scalable Application Development Software: Performance Tools

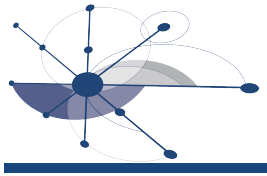
Barton Miller (Wisconsin)
John Mellor-Crummey (Rice)
Ewing Lusk (ANL)

SciDAC
Scientific Discovery through Advanced Computing



Toward Performance Tools for the Petascale

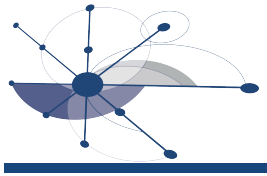
- A multi-prong effort towards
  - new tool ideas
  - realization in software
  - feedback from engagement with peer tool groups and application teams
- Targeting high and low
  - IBM Blue Gene and Cray XT leadership class platforms
  - multicore processors
- Key technology areas
  - binary analysis
  - instrumentation and measurement
  - performance and differential analysis
  - user interfaces and visualization



Engaging the Tools Community

CScADS Workshops on Performance Tools

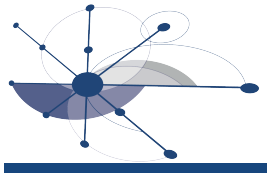
- Problem
  - measurement, analysis, and modeling of application performance on petascale systems is too large for any one group to develop alone
- Approach
  - promote development of sharable software components
  - accelerate the development of performance tools for leadership computing platforms.
- Participants from
  - AMD, Cray, HP, IBM, Intel, Sun
  - ORNL, LBNL, PNL, LANL, LLNL, Sandia, BSC, Krell Institute, FZJ
  - Berkeley, Maryland, New Mexico, UNC, Oregon, PSU, Rice, Tennessee, UAB, Wisconsin
- Impact
  - effort to catalyze collaboration among the tools community is beginning to bear fruit



Engaging the Tools Community

CScADS Workshops on Performance Tools

- A meeting of experts; a heavy focus on problems and solutions
- Detailed presentations, with live demos encouraged
- An emphasis on “what can you share with others?” and “what are you using from others?”
- Multi-day working groups on collaborations and sharing
  - performance data XML formats
  - binary analysis
  - performance visualization components
  - stack walking: issues, approaches, and interfaces
  - control flow parsing
  - access to hardware event counters
  - scalable I/O for performance data
  - trace generation and processing

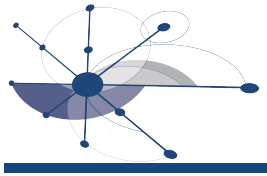


Engaging the Tools Community

CScADS Workshops on Performance Tools

Some outcomes

- libmonitor (Rice) for sampling or tracing operations at the library API level, now in use by Open|Speedshop – a DOE tri-labs performance tools project.
- SymtabAPI (Wisconsin) for parsing symbol information from object files, now in use by the HPCToolkit project.
- MRNet (Wisconsin) for multicast and reduction communication for tool control, now in use by the LLNL as part of the STAT debugger project.
- sionlib (Jülich) for parallel access to task-local files, being targeted for us in HPCToolkit.
- XED2 (Intel) instruction decoder for analysis of x86 machine instructions, now in use by HPCToolkit.
- Perfmon2 for hardware-based performance monitoring for Linux, now used by Cray as part of their Compute Node Linux operating system for the Cray XT.



Component-based Approach

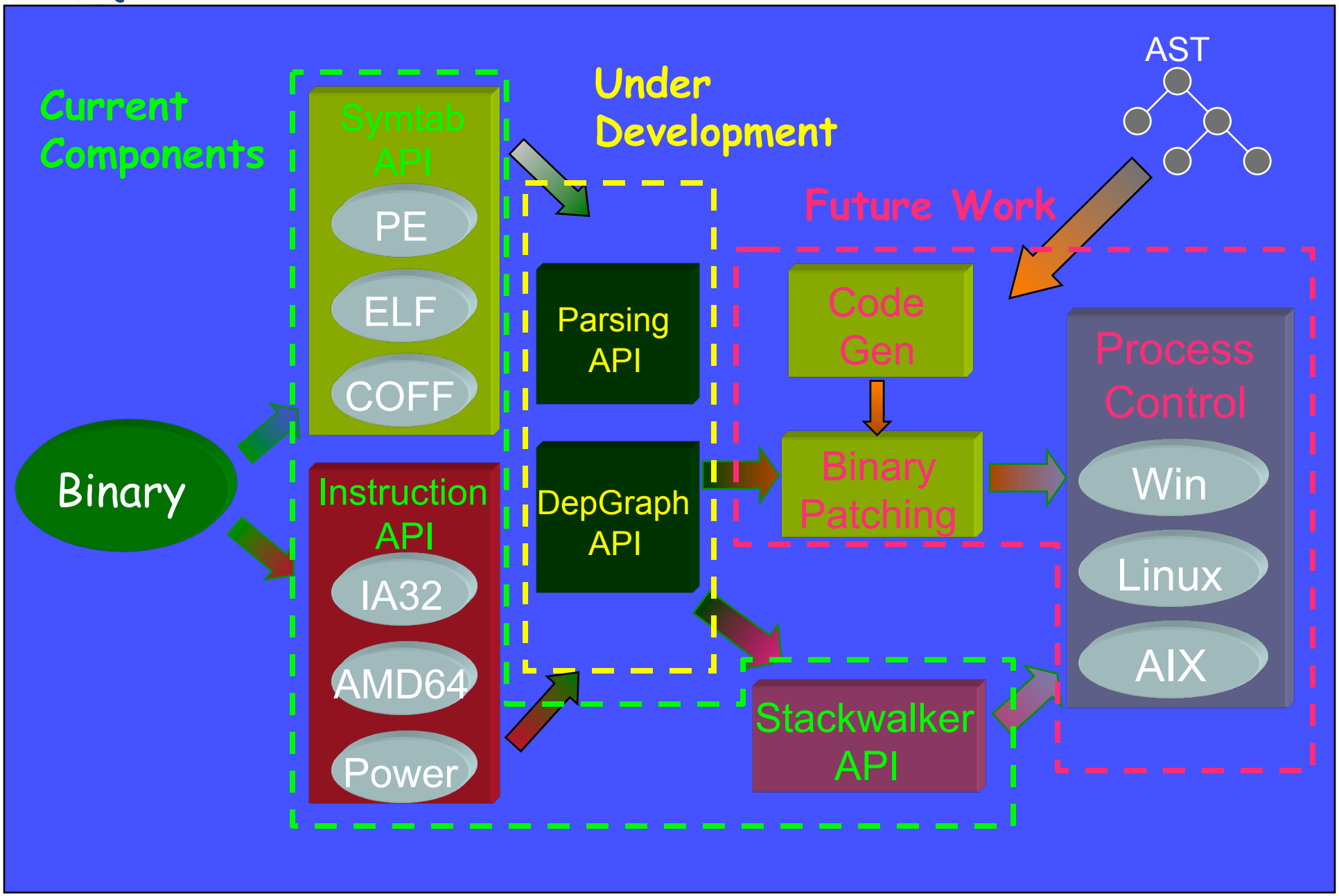
Infrastructure for Performance Tools

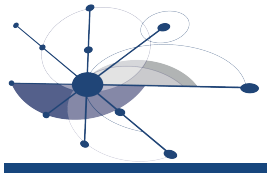
- Increases sharing and reuse
  - reduces redundant development
- Large research tool groups can focus on their priority missions without having to develop all parts of an end-to-end solution
- Small research groups (young investigators!) can explore focused research topics with a software code base comparable to that of larger groups

Collaborations with internal (Rice, Wisconsin) and external (LLNL, Cray, Intel, Berkeley, Oregon, BSC, Jülich) groups on various APIs

Workshop discussions are a critical part of the design process

Dyninst Component Roadmap



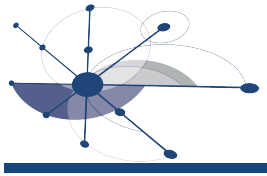


The Deconstruction of DynInst

Realizing our Push Toward Tool Components

- InstructionAPI
  - abstract representation of instruction decode and address modes.
- SymtabAPI
  - abstraction of symbols, debug and dynamic linkage information
  - updating to support binary rewriting
- StackwalkerAPI
  - walk stacks: first or third party, standard vs. optimized frames, custom frames (from instrumentation or exceptions)
  - uses a variety of techniques from full symbols and libunwind to stripped binaries requiring control-flow analysis
  - true synthesis of Wisconsin and Rice algorithms and code
- CFGAPI
  - platform independent representation of Control Flow Graph, associated query routines, and extensible data structures

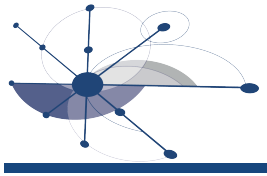
CScADS (all or partially) funded components are underlined



The Deconstruction of DynInst

Just over the Horizon

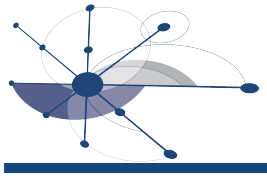
- PDGAPI
  - platform independent representation of Program Dependence Graph, including DDG and CDG API's.
- ProcessControlAPI
  - operating system independent interface for controlling processes and threads, monitoring events and state changes and reading and writing the application address space (tricky in for multi-threaded aps).
- CodeGenAPI
  - instruction set independent code generator for incremental and dynamically generated code.
  - lightweight and fast
  - context based, considering, e.g., register and stack frame use.
- CodePatchAPI
  - patch area allocation, code relocation, and code hooking.



libmonitor

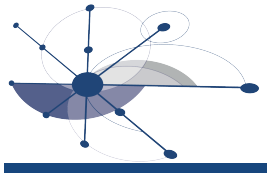
An Interface between OS and First-party Tools

- Processes
  - parent: pre_fork, post_fork
  - child: init_process, fini_process
- Threads
  - parent: init_thread_support, thread_pre_create, thread_post_create
  - child: init_thread, fini_thread
- MPI
  - mpi_pre_init, mpi_post_init, mpi_pre_fini, mpi_post_fini
- Signals
  - selectively catch signals before or instead of delivering to application
- Intercept functions to maintain control
  - e.g. dlopen, sigmask, pthread_sigmask, exit, signal, sigaction
- Stack unwinding support
  - stack_bottom; identification of PC for bottommost frame

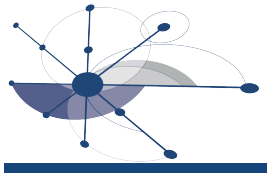


A Few Component Consumers

- Rice: using SymtabAPI and libmonitor in HPCToolkit
- Krell Institute (Open|SpeedShop) using SymtabAPI to get symbols for their offline collectors; using libmonitor in first-party tools
- UNC and LLNL: using SymtabAPI and StackwalkerAPI for PnMPI project
- LLNL (STAT project): using SymtabAPI and StackwalkerAPI
- SiCortex: porting SymtabAPI to Linux/MIPs; using libmonitor underneath HPCToolkit
- Cray: started work using StackwalkerAPI and SymtabAPI for new APT (Abnormal Process Termination) system
- Univ. of Oregon: using binary rewriter as part of TAU instrumentation
- Forschungszentrum Jülich: using SymtabAPI for Scalasca
- Berkeley (BitBlaze): APIs for binary processing (security tools)
- BSC: using Dyninst for function-level instrumentation

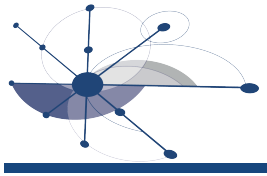


HPCToolkit



HPCToolkit Goals

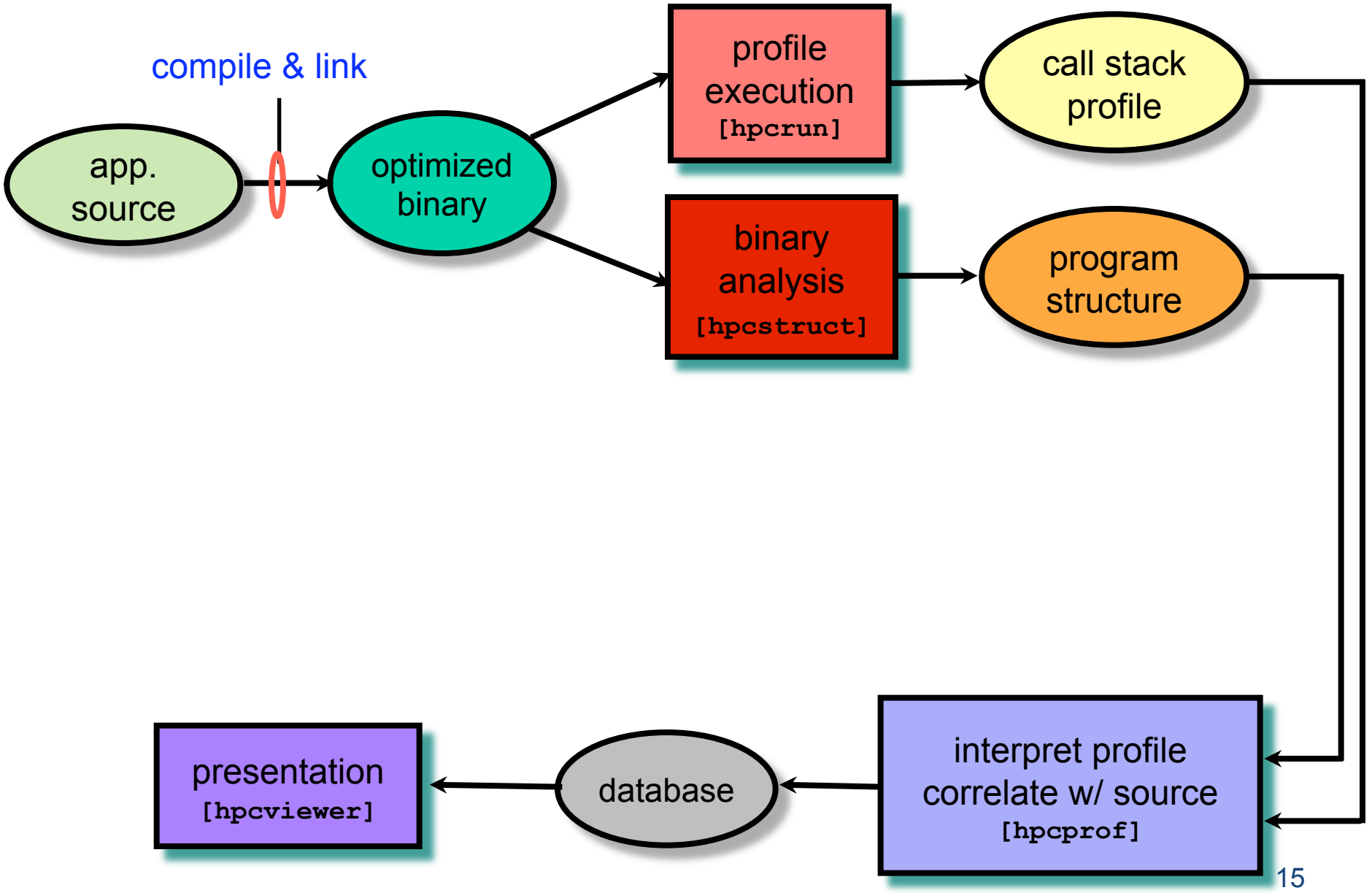
- Accurate measurement of complex parallel codes
  - large, multi-lingual programs
  - fully optimized code: loop optimization, templates, inlining
  - binary-only libraries, sometimes partially stripped
  - complex execution environments
    - dynamic loading or static binaries
    - SPMD parallel codes with threaded node programs
    - batch jobs
  - production executions
- Effective performance analysis
  - pinpoint and explain problems
    - intuitive enough for scientists and engineers
    - detailed enough for compiler writers
  - yield actionable results
- Scalable to petascale systems



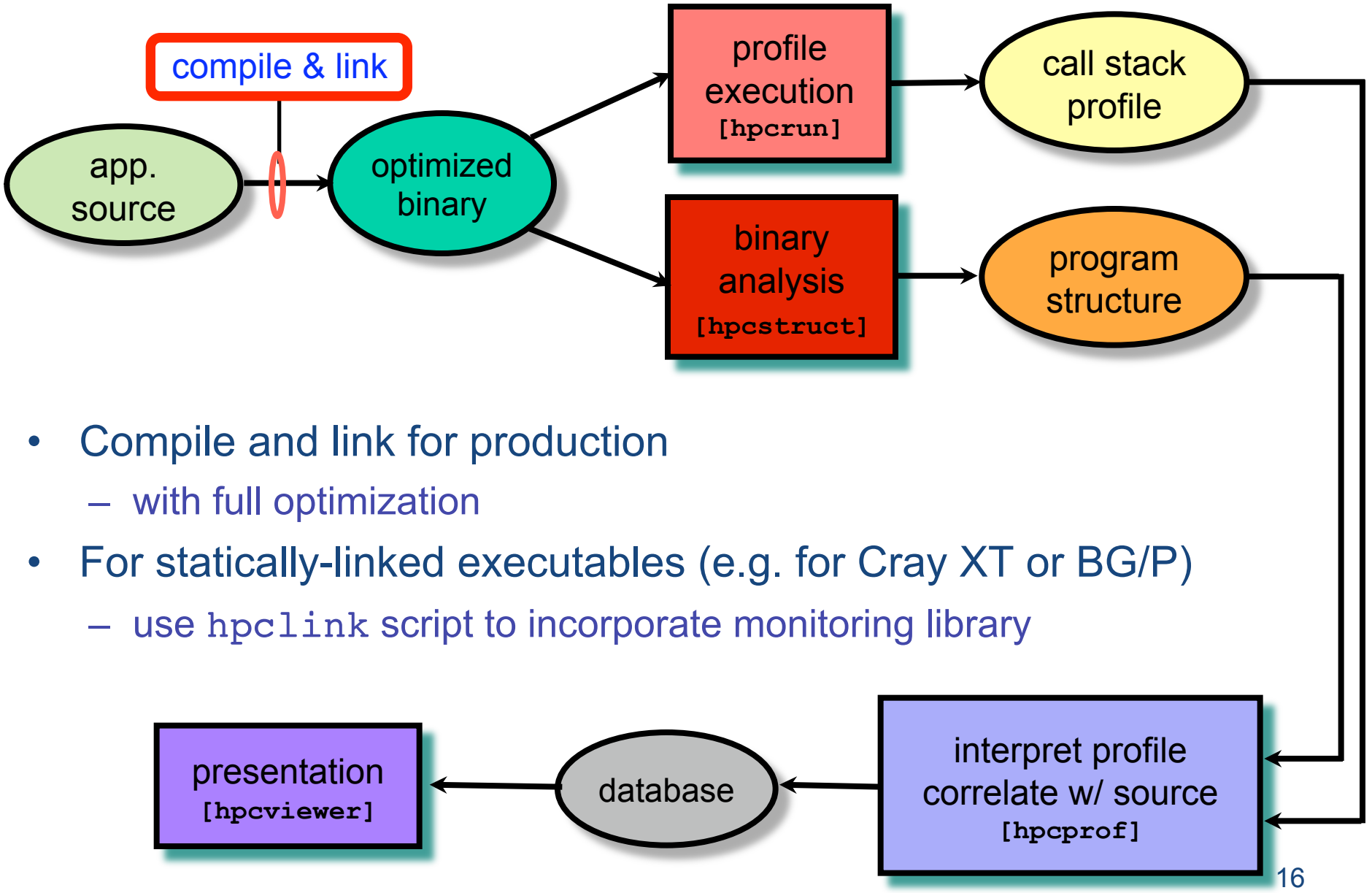
HPCToolkit Approach

- Binary-level measurement and analysis
  - observe executions of fully optimized, static or dynamic binaries
- Sampling-based measurement
  - minimize systematic error and avoid blind spots
  - support data collection for large-scale parallelism
- Associate metrics with both static and dynamic context
  - loop nests, procedures, inlined code, calling context
- Synthesize derived performance metrics
  - diagnosis requires more than one metric
  - derived metrics: “miss rate,” “scalability loss”
- Support top-down performance analysis
  - start with what’s most important
  - keep unnecessary detail out of the way
- Support multiple views of performance data
  - different views help pinpoint different problems

HPCToolkit Performance Tools

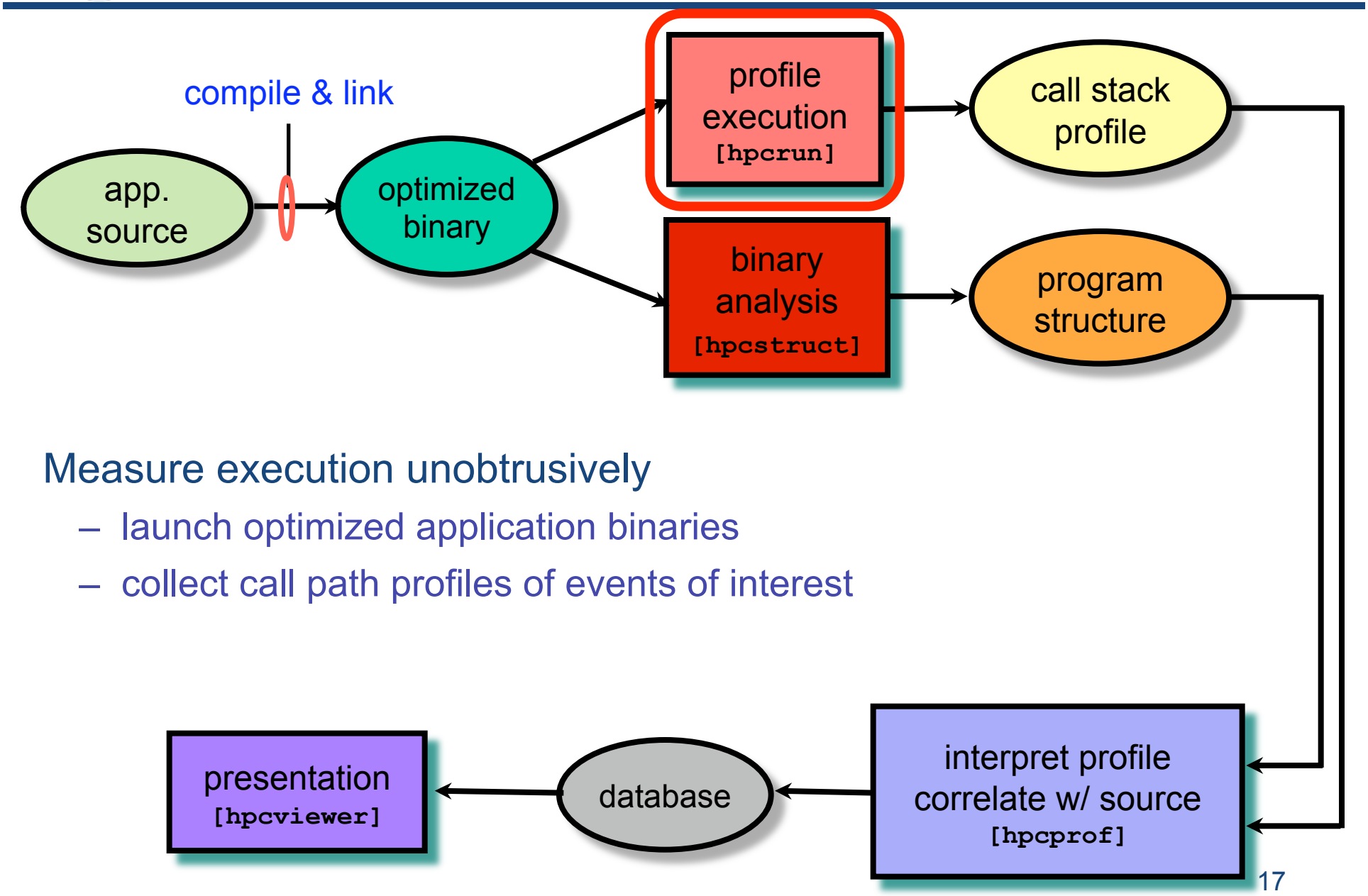


HPCToolkit Performance Tools



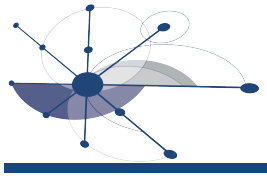
- Compile and link for production
  - with full optimization
- For statically-linked executables (e.g. for Cray XT or BG/P)
  - use `hpclink` script to incorporate monitoring library

HPCToolkit Performance Tools



Measure execution unobtrusively

- launch optimized application binaries
- collect call path profiles of events of interest

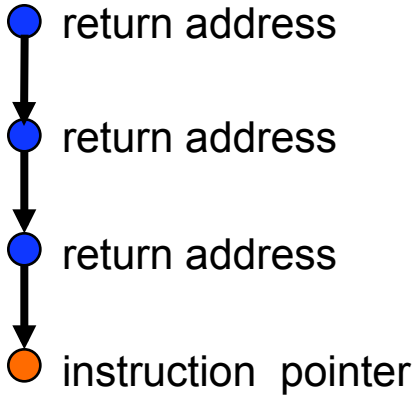


Call Path Profiling

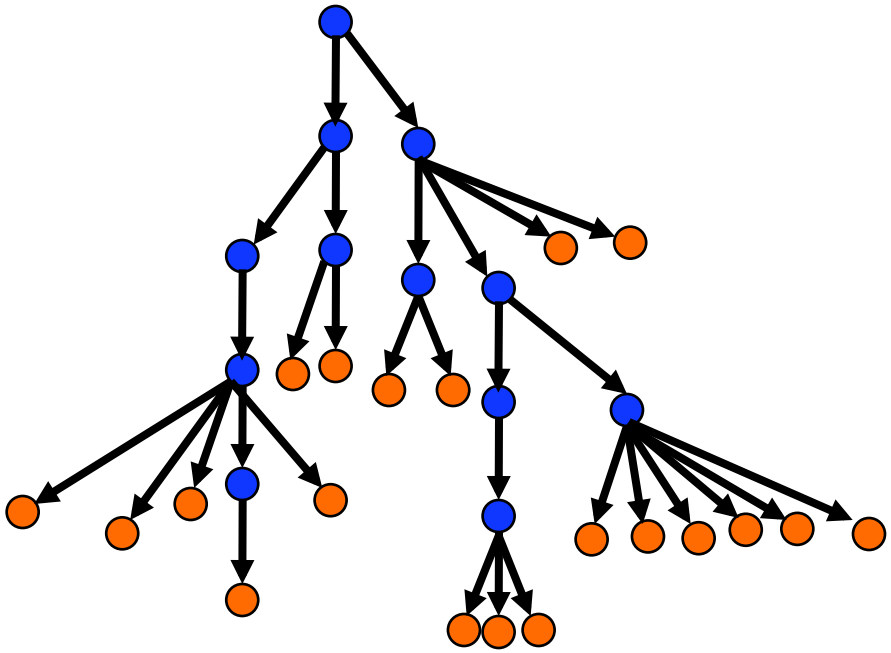
Measure and Attribute Costs in Context

- Sample timer or hardware counter overflows
- Gather calling context using stack unwinding

Call path sample

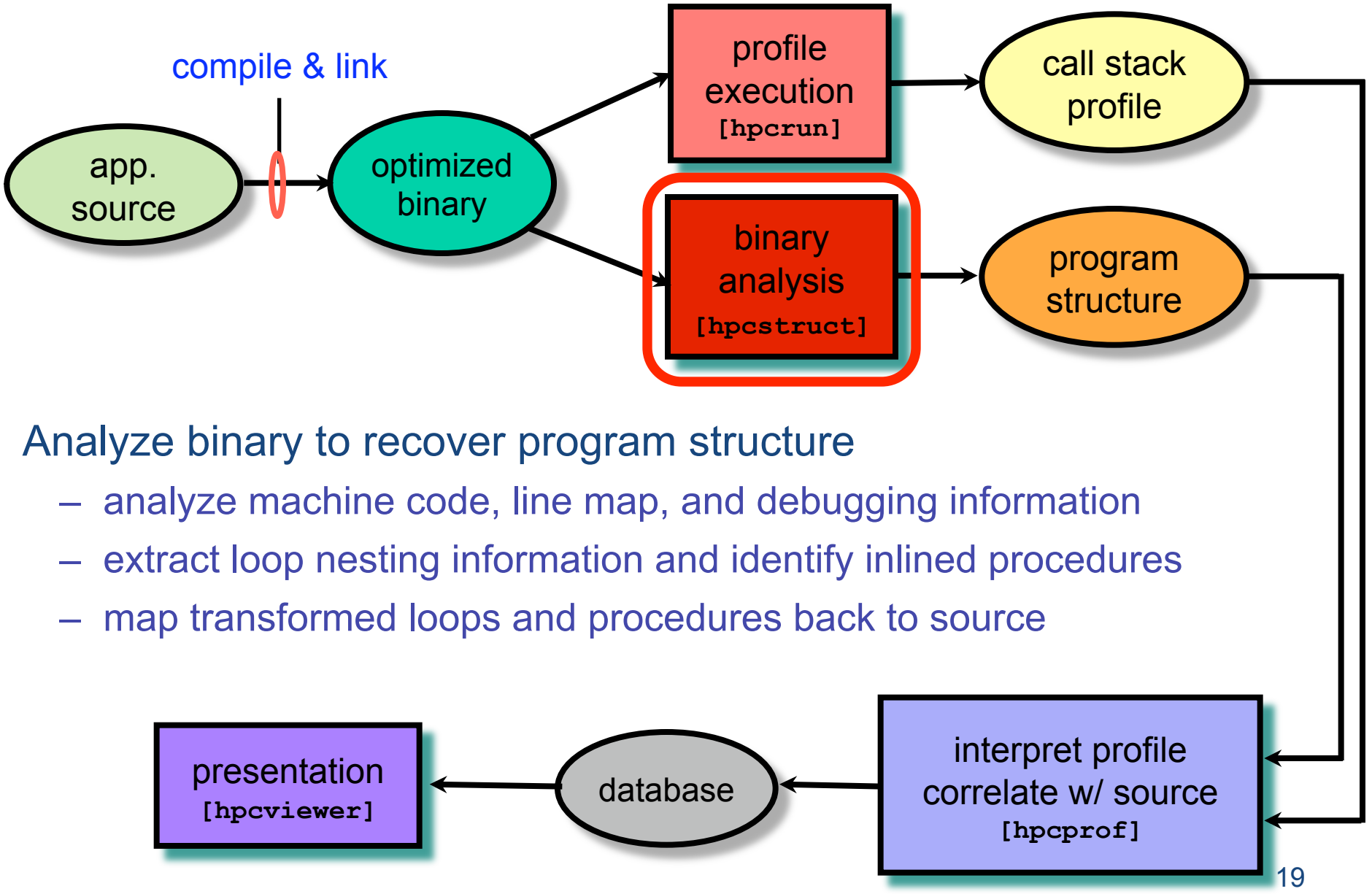


Calling Context Tree (CCT)



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

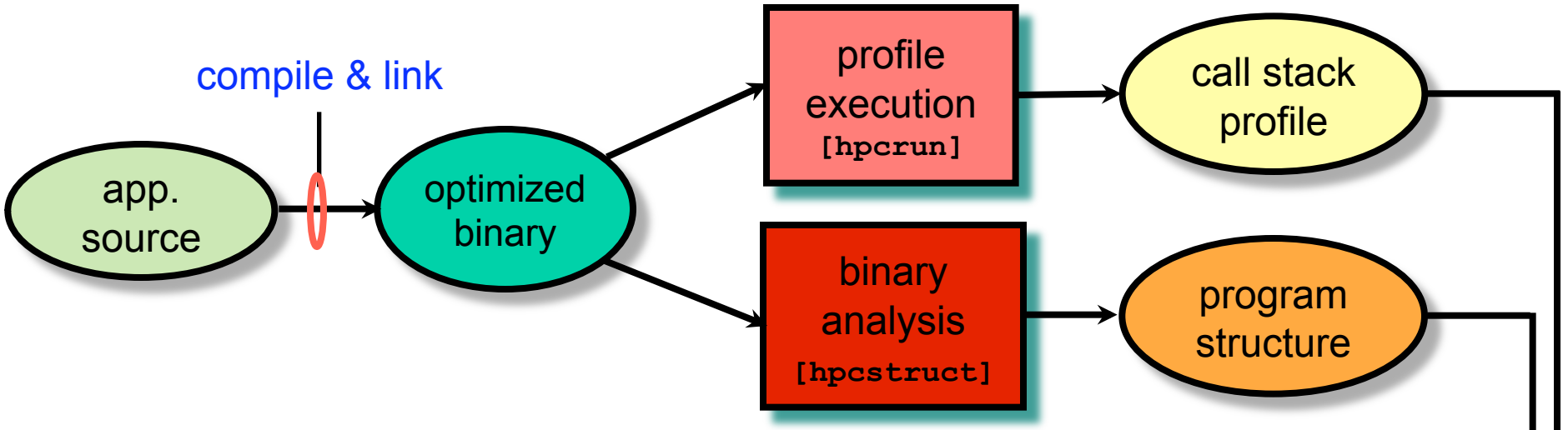
HPCToolkit Performance Tools



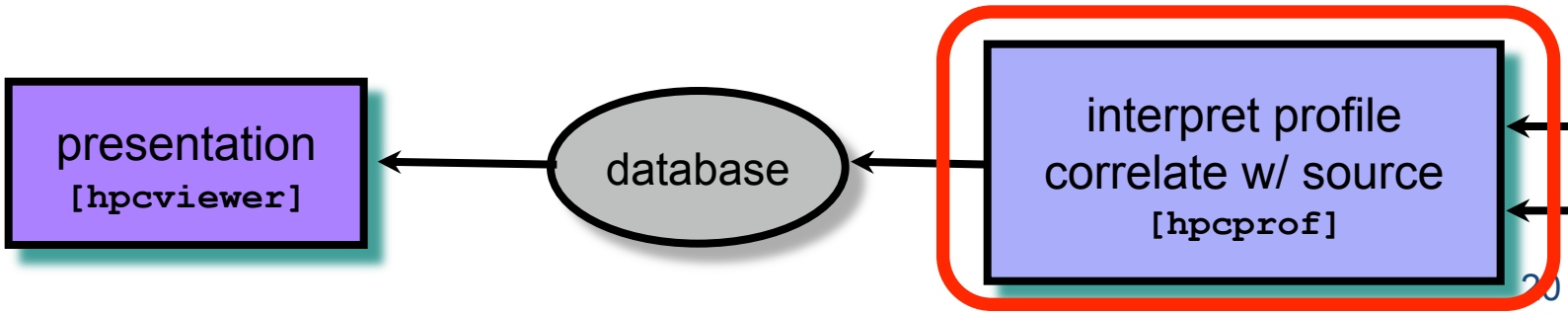
Analyze binary to recover program structure

- analyze machine code, line map, and debugging information
- extract loop nesting information and identify inlined procedures
- map transformed loops and procedures back to source

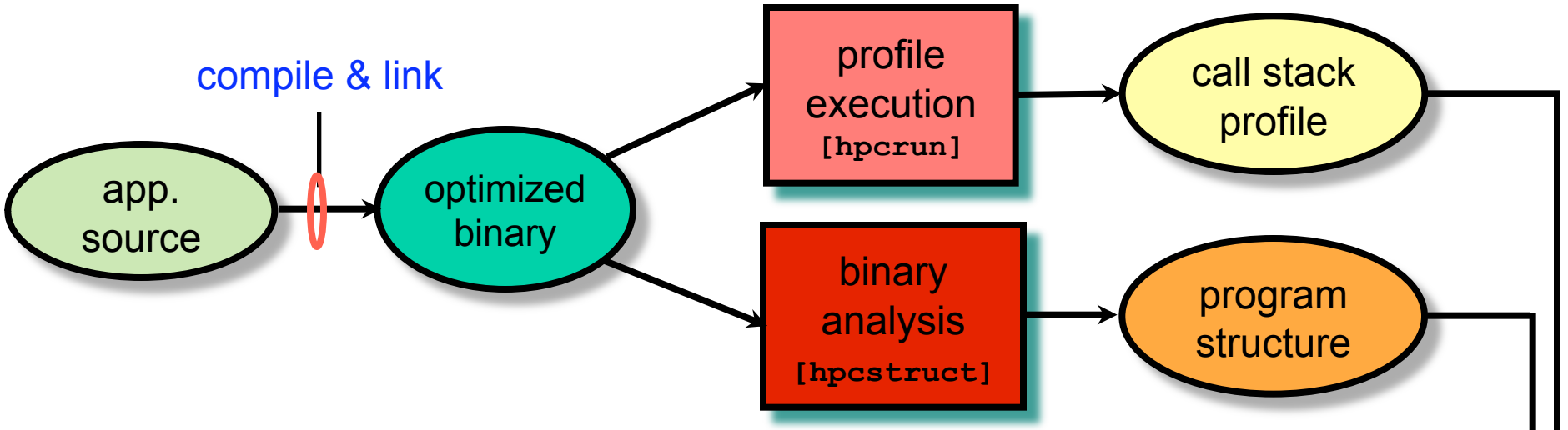
HPCToolkit Performance Tools



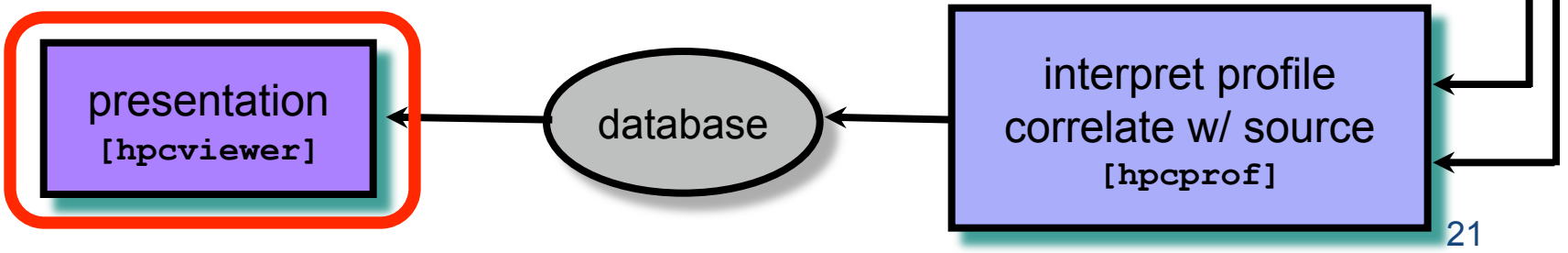
- Combine multiple profiles
  - multiple threads; multiple processes; multiple executions
- Correlate measurements to static and dynamic program structure



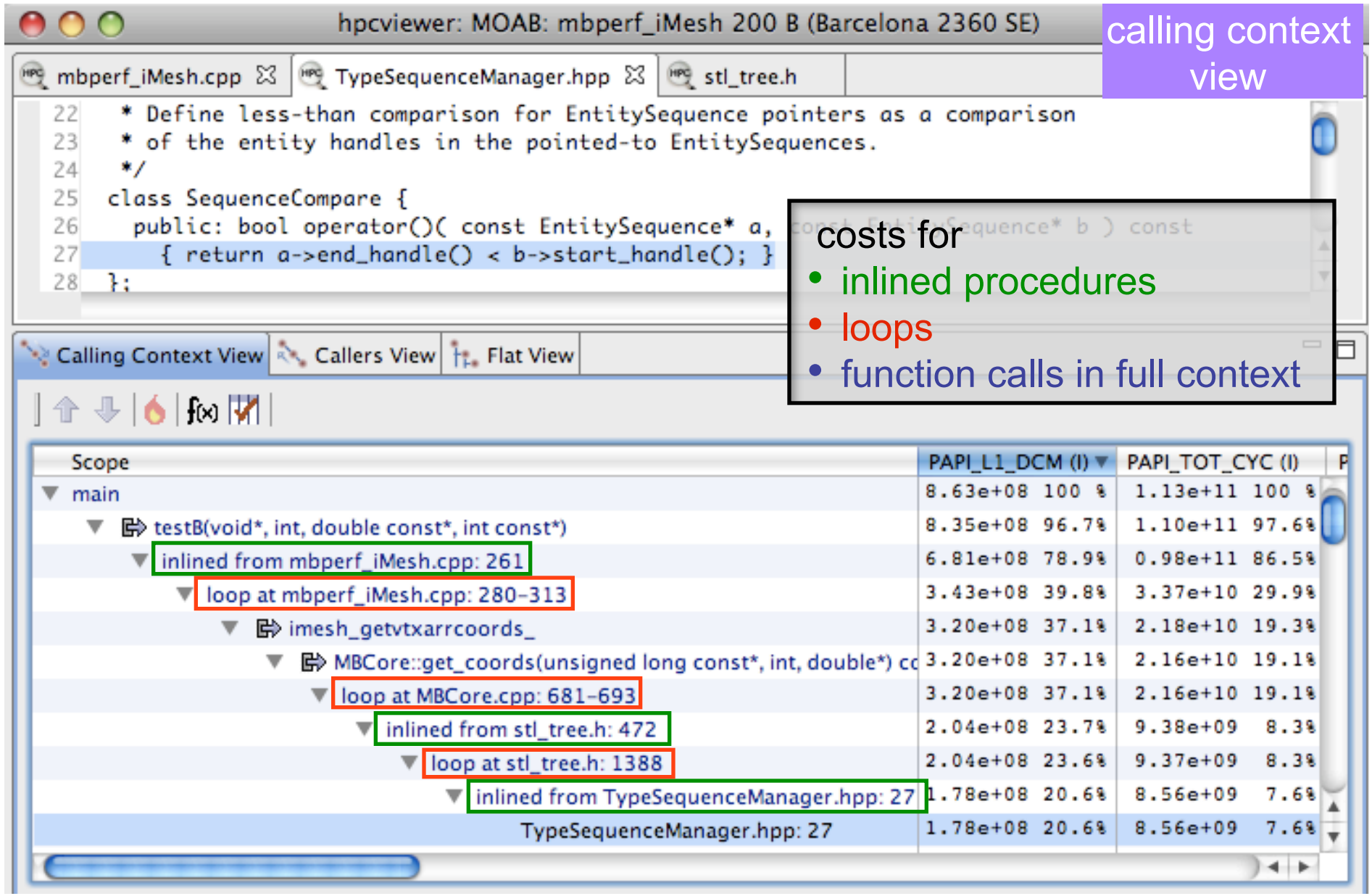
HPCToolkit Performance Tools



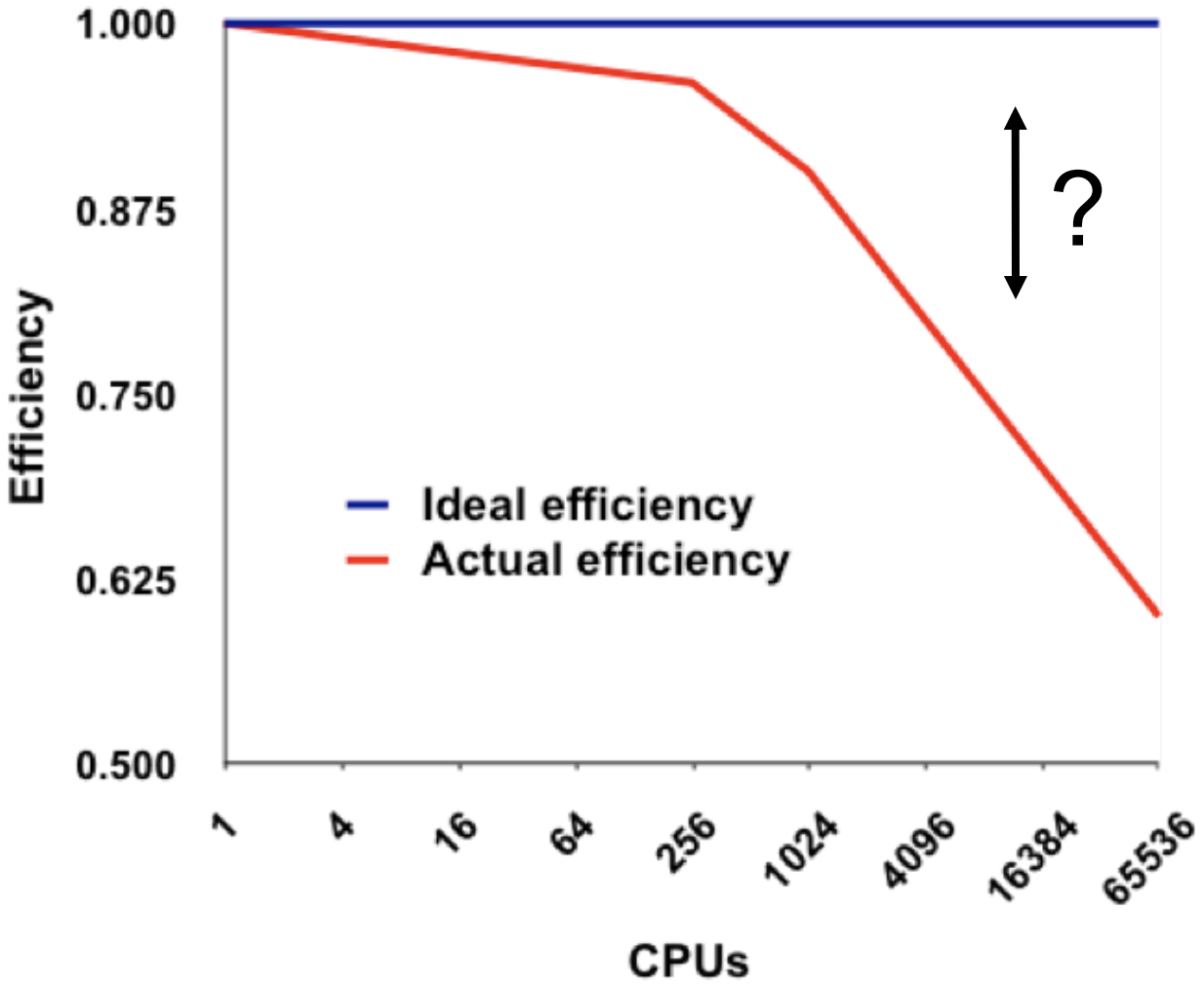
- Explore performance data from multiple perspectives
- Rank order by metrics to focus on what's important
- Compute derived metrics to gain insight



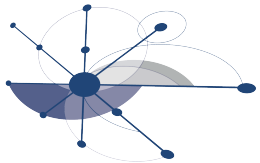
MOAB Mesh Library from ITAPS



Pinpointing Scalability Bottlenecks

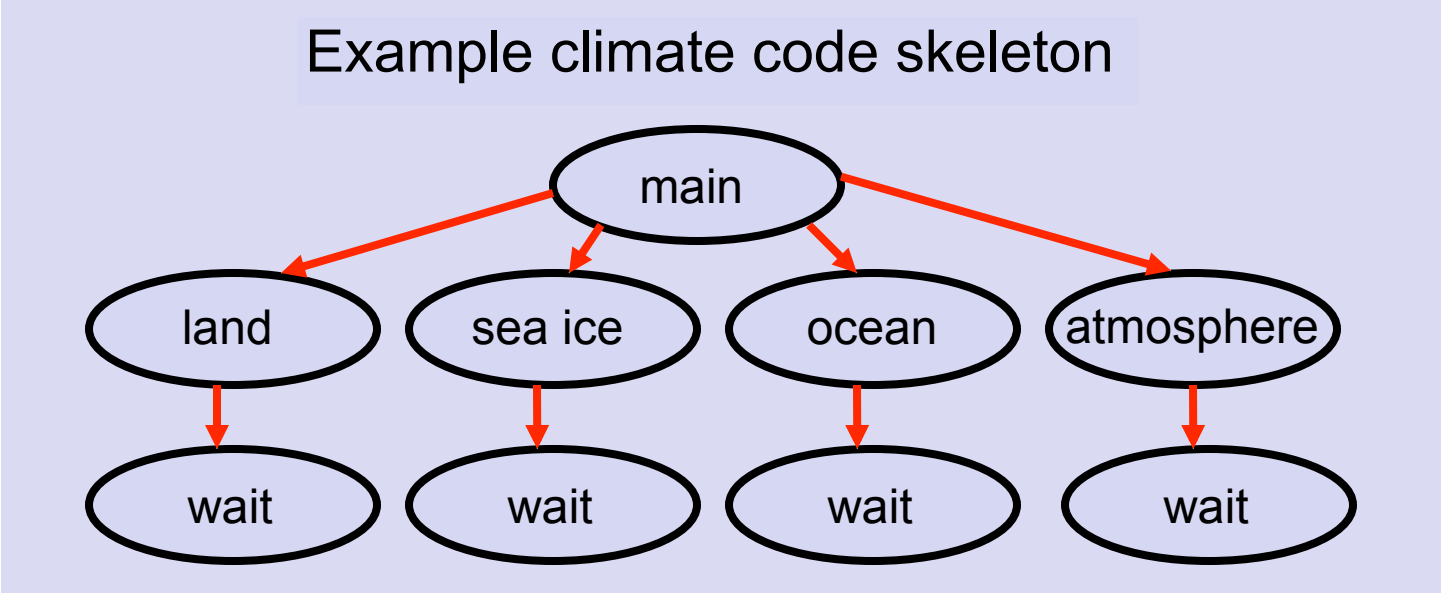


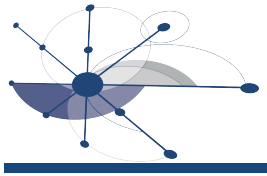
Note: higher is better



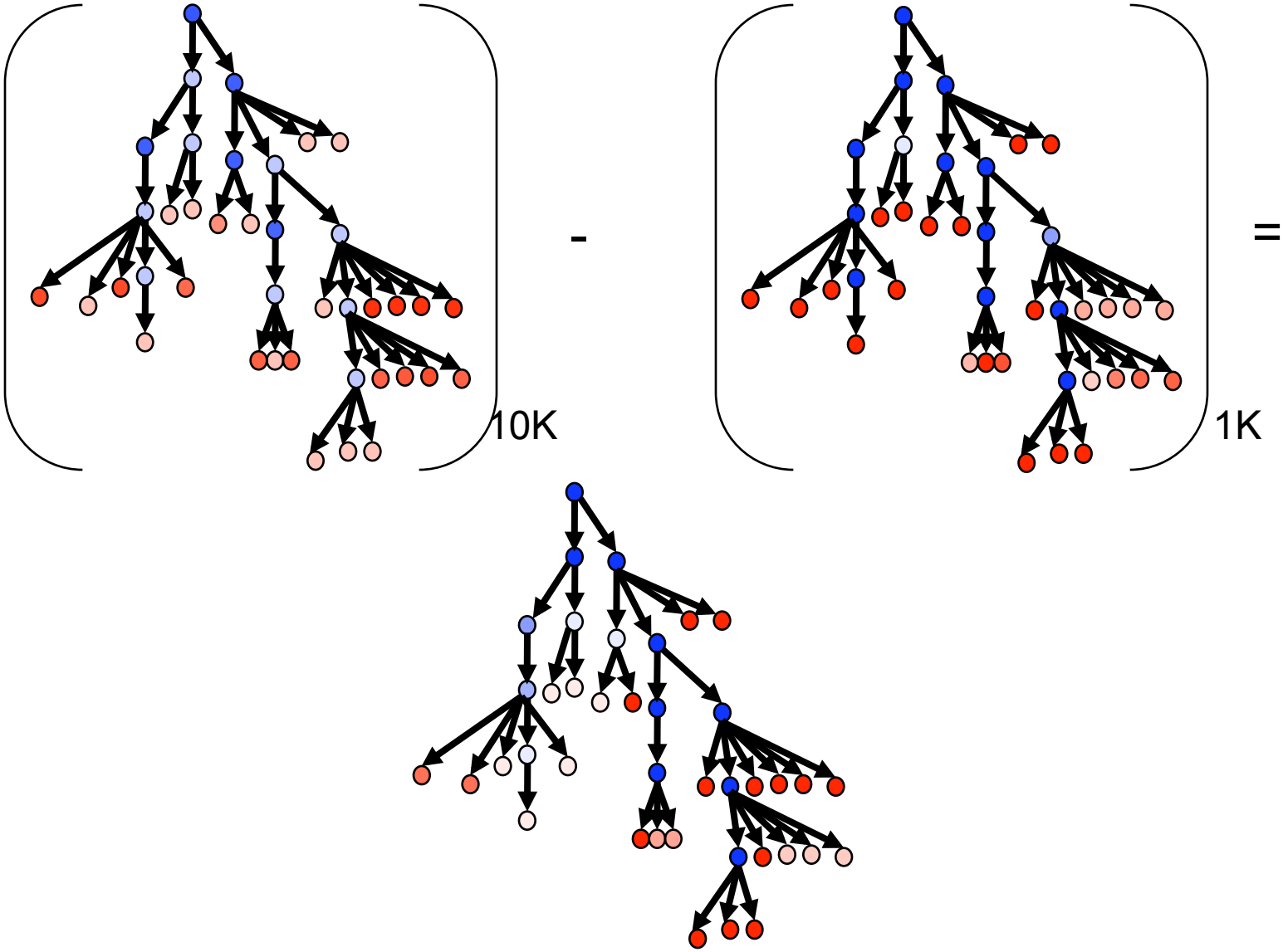
Bottleneck Analysis Challenges

- Parallel applications
  - modern software uses layers of libraries
  - performance is often context dependent
- Monitoring
  - bottleneck nature: computation, data movement, synchronization?
  - size of petascale platforms demands acceptable data volume
  - low perturbation for use in production runs

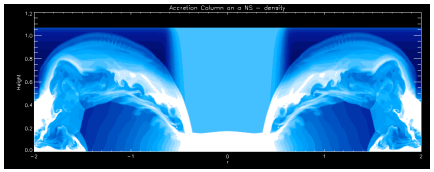




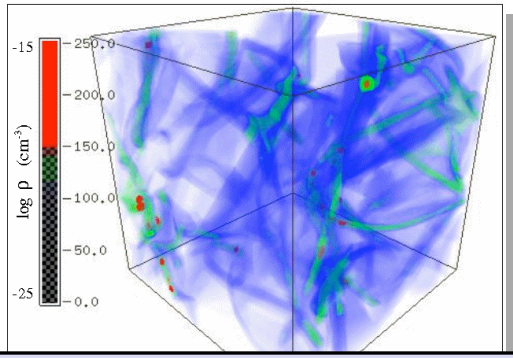
Analyzing Weak Scaling: 1K to 10K processors

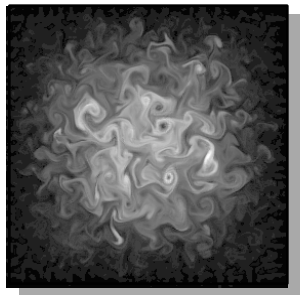


FLASH from University of Chicago

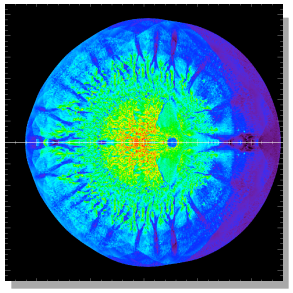


Shortly: Relativistic accretion onto NS

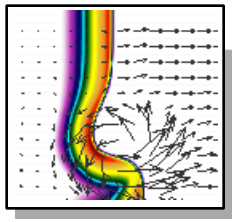




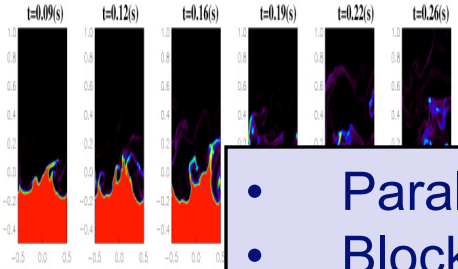
Compressed turbulence



Type Ia Supernova



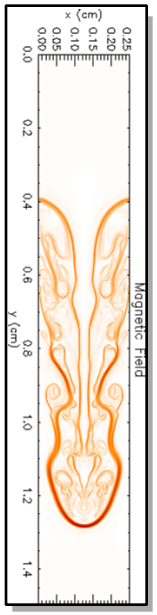
Flame-vortex interactions



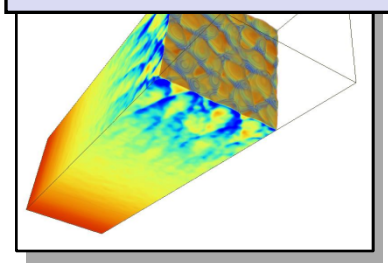
Wave breaking or

nteractions

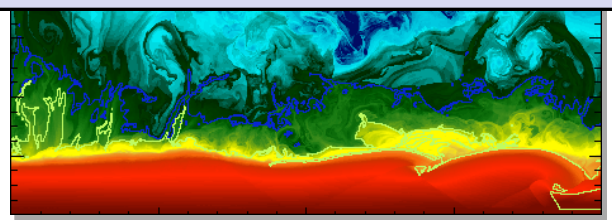
- Parallel, adaptive-mesh refinement (AMR) code
- Block structured AMR; a block is the unit of computation
- Designed for compressible reactive flows
- Can solve a broad range of (astro)physical problems
- Portable: runs on many massively-parallel systems
- Scales and performs well
- Fully modular and extensible: components can be combined to create many different applications



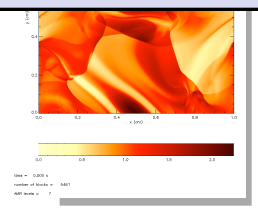
Magnetic Rayleigh-Taylor



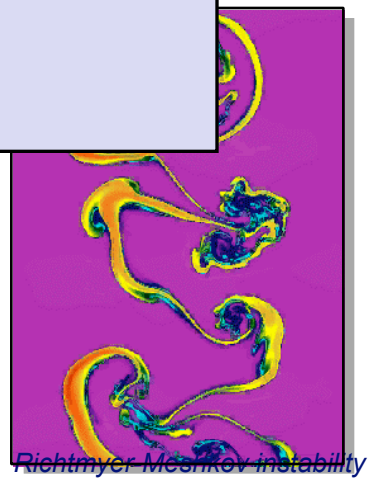
Cellular detonation



Helium burning on neutron stars



Orzag/Tang MHD vortex



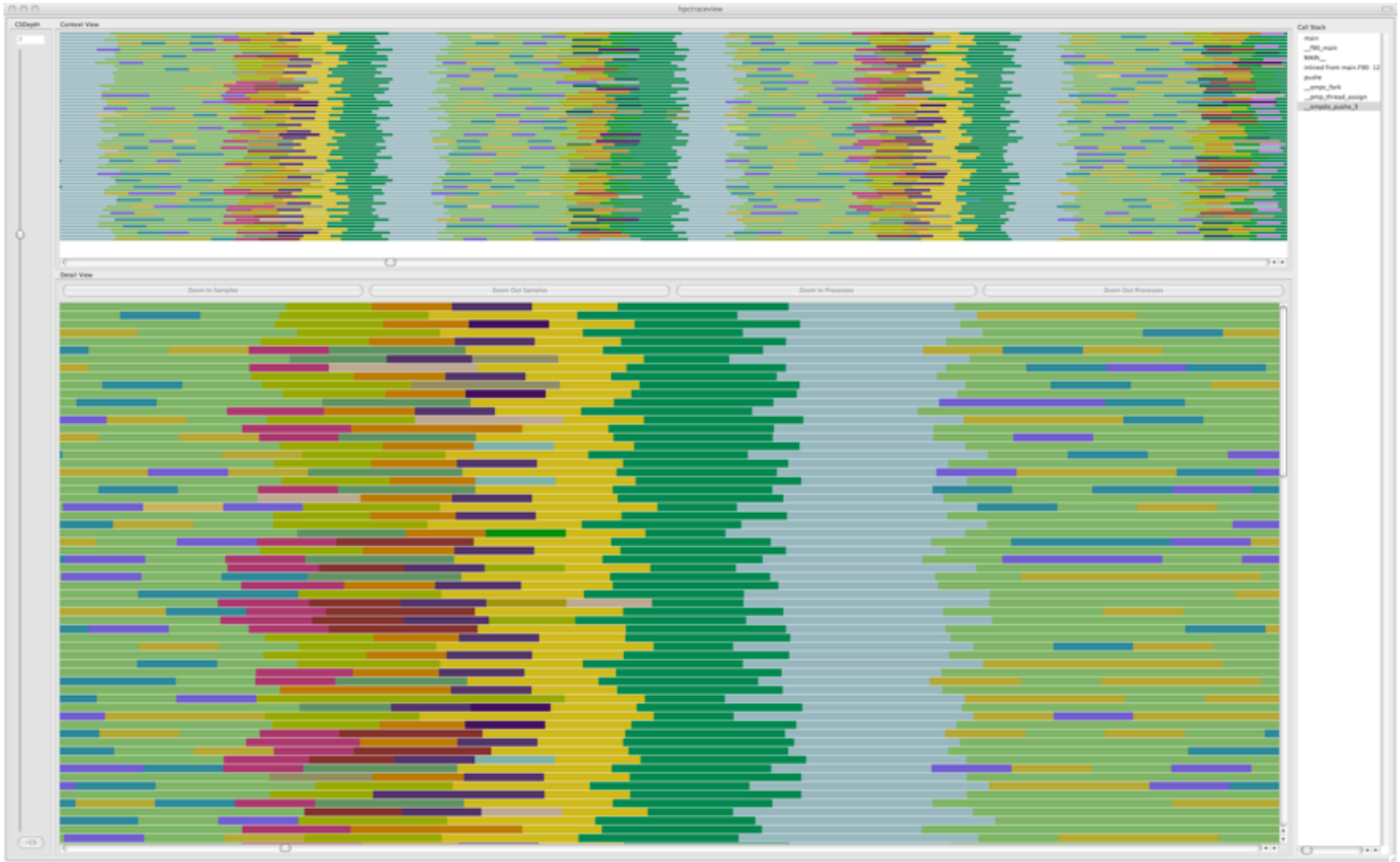
Richtmyer-Meshkov instability

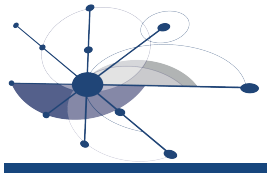
Text and figures courtesy of FLASH Team, University of Chicago

Viewing Traces of Asynchronous Samples

Viewing a GTC call stack sample trace with `hpctraceviewer`

- 32 process MPI program
- Each process has a pair of threads managed with OpenMP

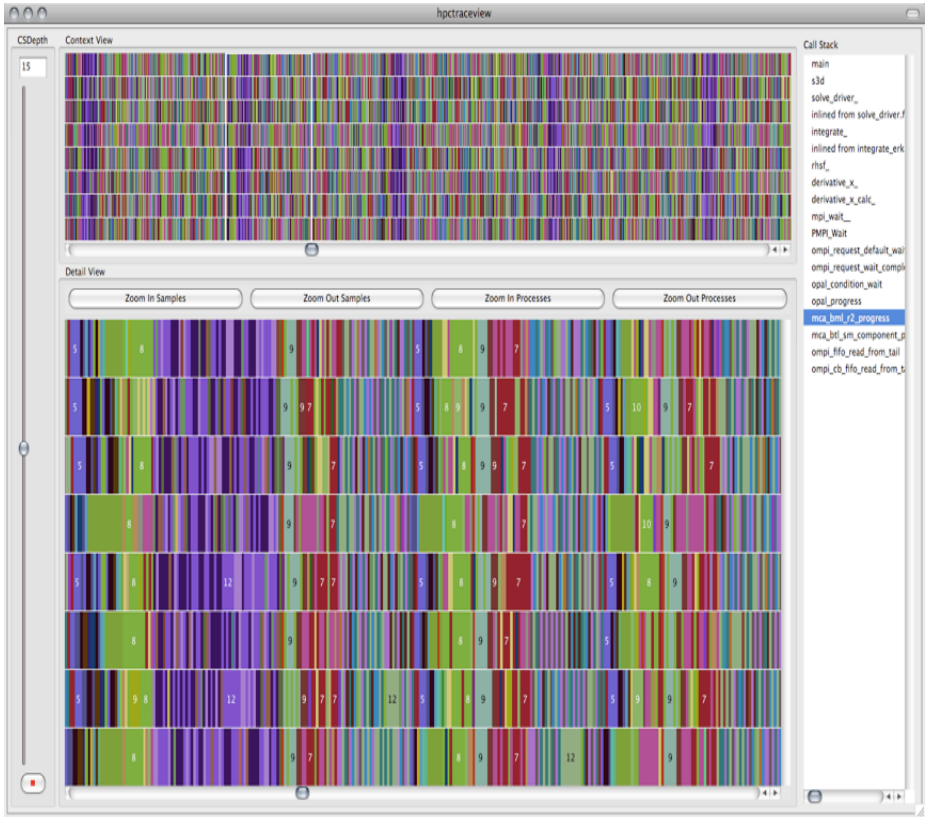




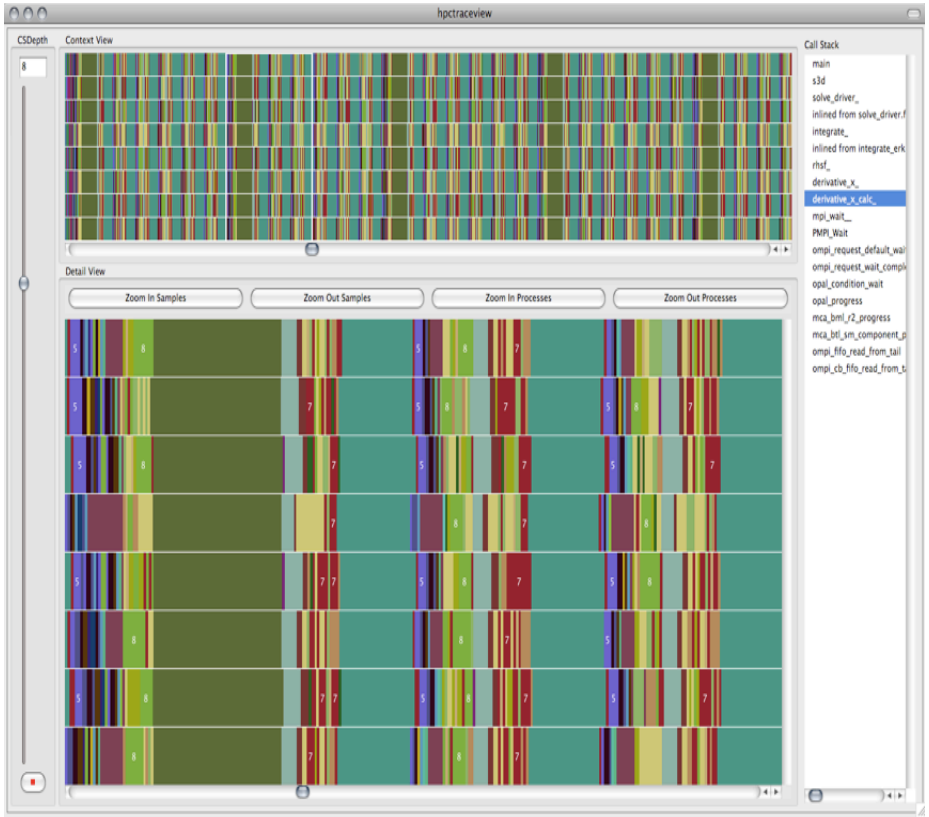
Multiple Levels of Abstraction

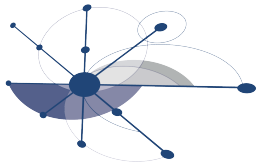
- Call stack sample trace for S3D
- 8 cores on a dual quad-core node; OpenMPI

call stack depth 15



call stack depth 8





Novel Capabilities of HPCToolkit

Measurement

- Binary analysis for (1) recovering functions in partially stripped code, (2) unwinding fully-optimized code, (3) recovering program structure
- Nearly perfect call stack sampling of fully optimized code with low overhead

Binary Analysis for Measurement and Attribution of Program Performance, PLDI, June 2009. To appear.

Pinpoint Scalability Bottlenecks using Differential Profiling

Scalability Analysis of SPMD Codes using Expectations, ICS, June 2007

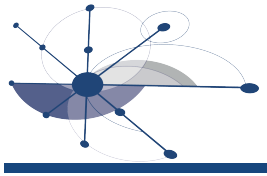
Pinpoint Performance Losses in Multithreaded Executions

Attribute insufficient parallelism and parallelization overhead for multithreaded programs on a work-stealing runtime using sampling

Effective Performance Measurement and Analysis of Multithreaded Applications, PPoPP, February 2009.

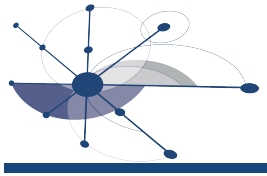
Performance Analysis using Sampling on Leadership Platforms

Diagnosing Performance Bottlenecks in Emerging Petascale Applications, *Submitted to SC09*

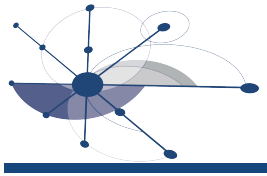


HPCToolkit Summary

- Precise measurement with low overhead
  - e.g. PFLOTRAN scaling study on Cray XT
    - measured cycles, L2 miss, FLOPs, & TLB @ 1.5% overhead, 512 cores
    - unwind errors 148 out of 289M on 8192 cores
  - suitable for use on production runs
- Insightful analyses
- Actionable feedback
- Scalable to the petascale
- Newly operational on leadership computing platforms
  - Cray XT (CNL) : March 27, 2009
  - Blue Gene/P: April 8, 2009
  - Opteron+IB (Linux): February 12, 2009 (TACC's Ranger)

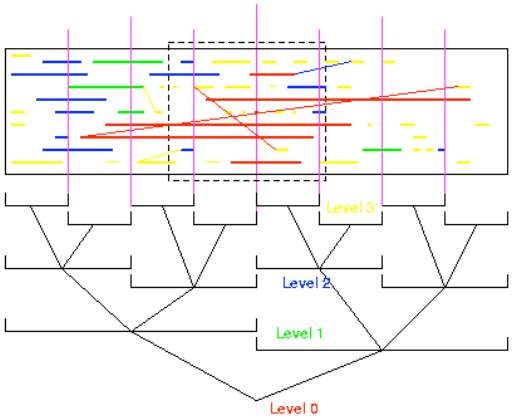


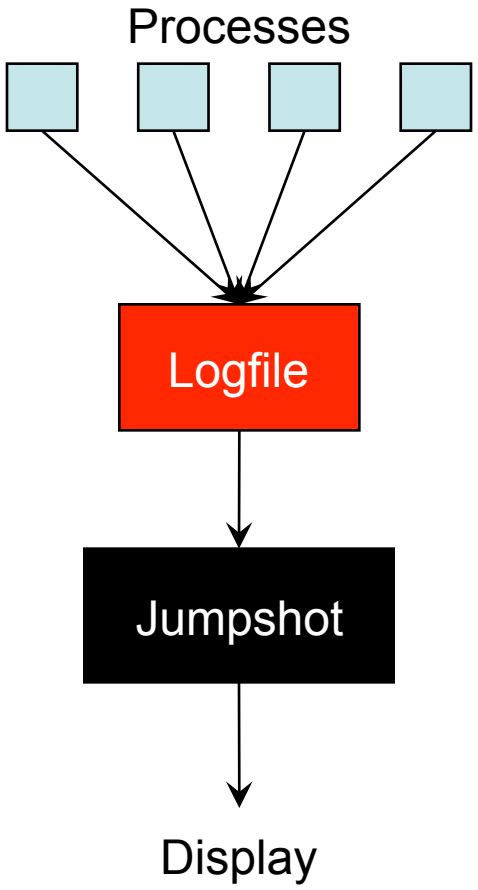
Jumpshot



Performance Visualization with Jumpshot

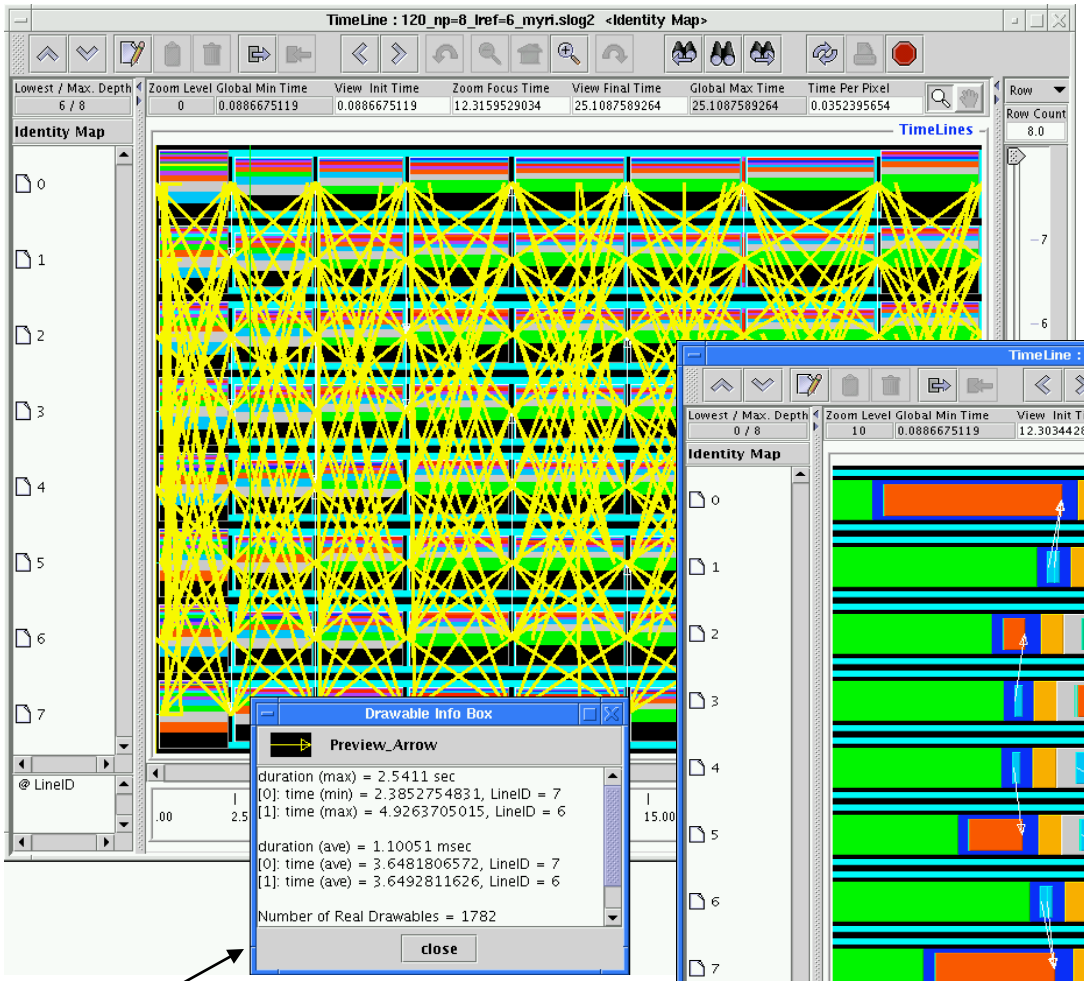
- For detailed analysis of parallel program behavior, timestamped events are collected into a log file during the run.
- A separate display program (Jumpshot) aids the user in conducting a post mortem analysis of program behavior.
- We use an indexed file format (SLOG-2) that uses a preview to select a time of interest and quickly display an interval, without ever needing to read much of the whole file.





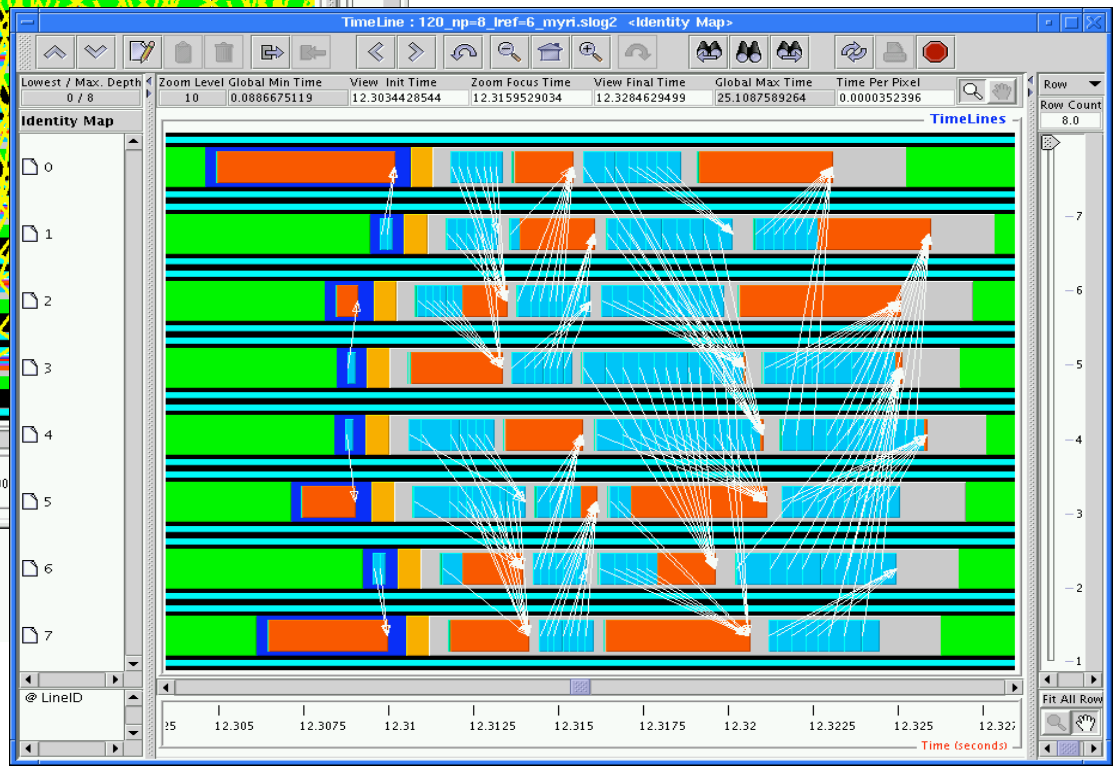
(clog → slog)

Viewing Multiple Scales with Jumpshot

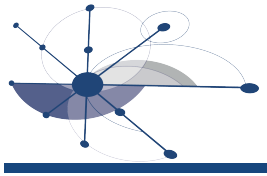


Each line represents 1000's of messages

Detailed view shows opportunities for optimization

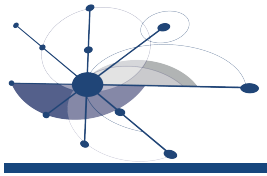


1000x zoom



More on Jumpshot

- Connection to other projects
  - distributed with MPICH2 (thousands of downloads per month)
  - Jumpshot viewer included as part of TAU, with converters
  - Berkeley UPC/GASP emits SLOG2 files for Jumpshot
- Scalability minuses
  - use with a thousand time lines still a research issue
  - need adaptive summary for amalgamating messages
- Scalability pluses
  - SLOG2 format allows interactive access to large trace files
  - summary states and messages
  - statistics view
  - can view subset of processes
- Basic Jumpshot premise
  - sometimes you have to look at the details



Jumpshot

- Using Jumpshot to study ADLB (Asynchronous Dynamic Load Balancing) library used in UNEDF SciDAC
  - understanding very irregular behavior over time