Fine-tuning your HPC Investments with Performance Analysis

John Mellor-Crummey

Department of Computer Science
Rice University
johnmc@cs.rice.edu

`http://www.hipersoft.rice.edu/hpctoolkit`

Performance Analysis for Oil and Gas HPC

- Understand how efficiently your HPC resources are used today
- Identify bottlenecks and improve efficiency on current systems
- Focus attention on appropriate emerging technologies
  - —multi-core processors
  - —application accelerators
  - —interconnection network fabrics and topologies
  - —storage systems
- Guide procurement of future systems that meet your needs

Performance Analysis Challenges

- Microprocessor architectures are hard to program effectively
  - —processors that are pipelined, out of order, superscalar
  - —multi-level memory hierarchy
  - —multi-level parallelism: multi-core, SIMD instructions
- Gap between typical and peak performance is huge
- HPC applications pose challenges for tools
  - —large complex programs
    - multi-lingual
    - multiple instantiations of code: templates, inlining
    - threaded parallelism
  - —binary-only external libraries
    - sometimes partially stripped
  - —complex execution environments
    - dynamic loading of code
    - batch parallel execution on clusters

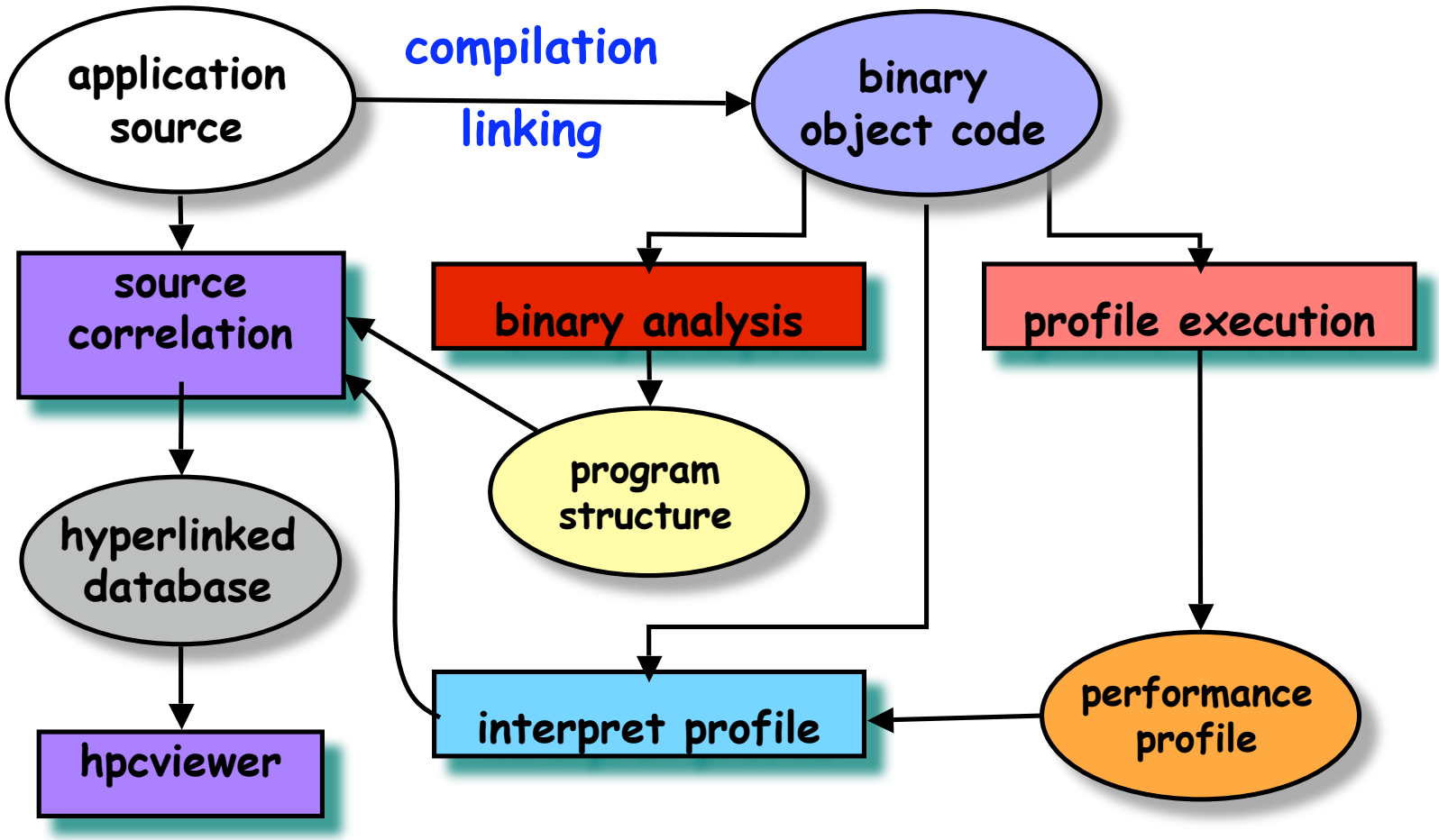
Rice's HPCToolkit Performance Tool Goals

- **Measurement of both serial and parallel codes**
  - —cope with all the complexities of real codes
  - —especially multi-threaded codes on multi-core processors
- **Scalable data collection for large-scale parallel executions**
- **Insightful analysis that pinpoints and explains problems**
- **Effective presentation of analysis results**
  - —correlate measurements with code (yield actionable results)
  - —intuitive enough for application scientists to use
  - —detailed enough to meet the needs of compiler writers

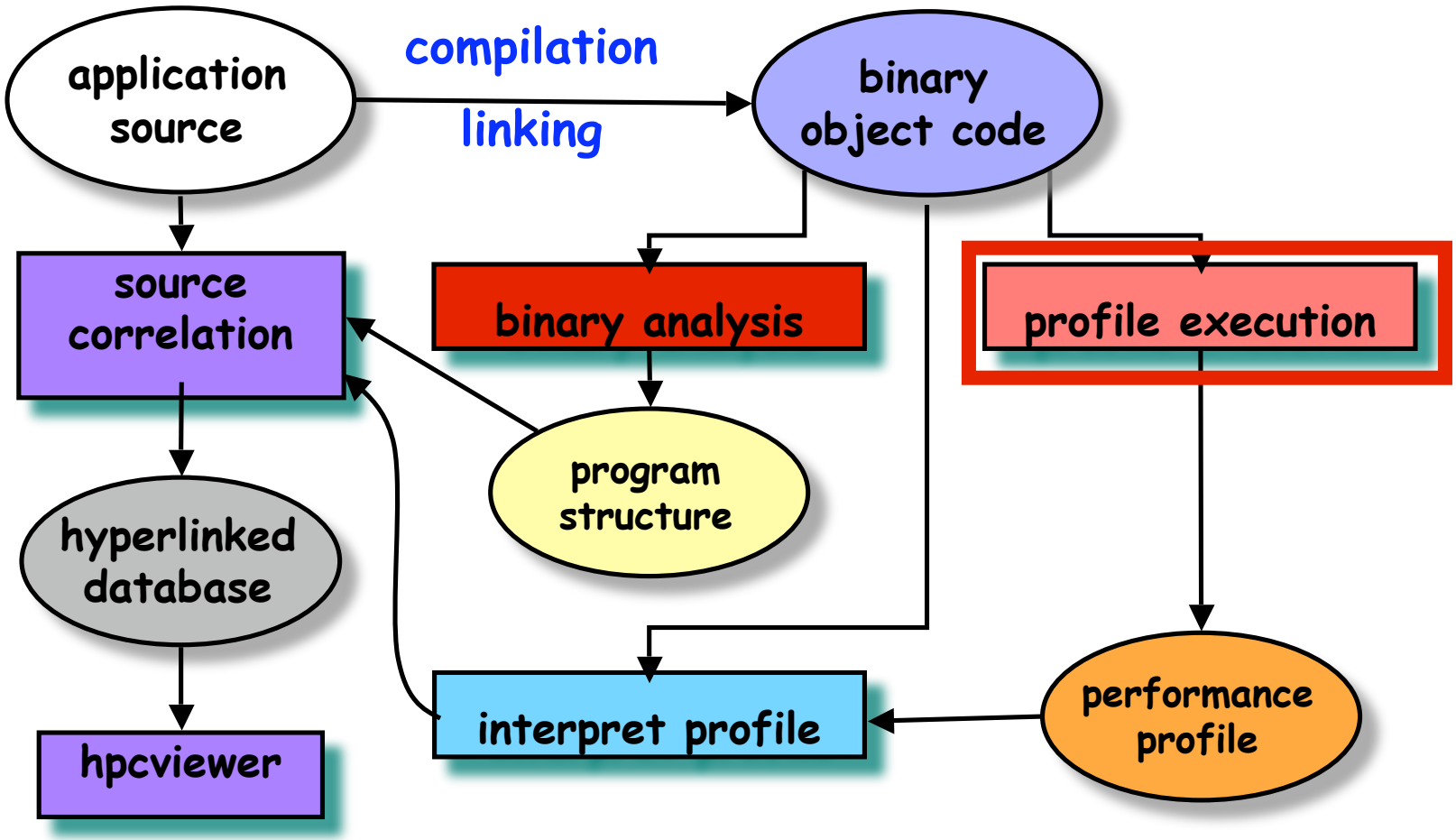
HPCToolkit Design Principles

- **Work at binary level for language independence**
  - —support multi-lingual codes with external binary-only libraries
- **Profile rather than adding code instrumentation**
  - —minimize measurement overhead and distortion
  - —enable data collection for large-scale parallelism
- **Collect and correlate multiple performance measures**
  - — can’t diagnose a problem with only one species of event
- **Compute derived metrics to aid analysis**
- **Associate costs with both static and dynamic context**
  - —loop nests, procedures, inlined code, calling context
- **Support top down performance analysis**
  - —intuitive enough for scientists and engineers to use
  - —detailed enough to meet the needs of compiler writers

HPCToolkit Workflow

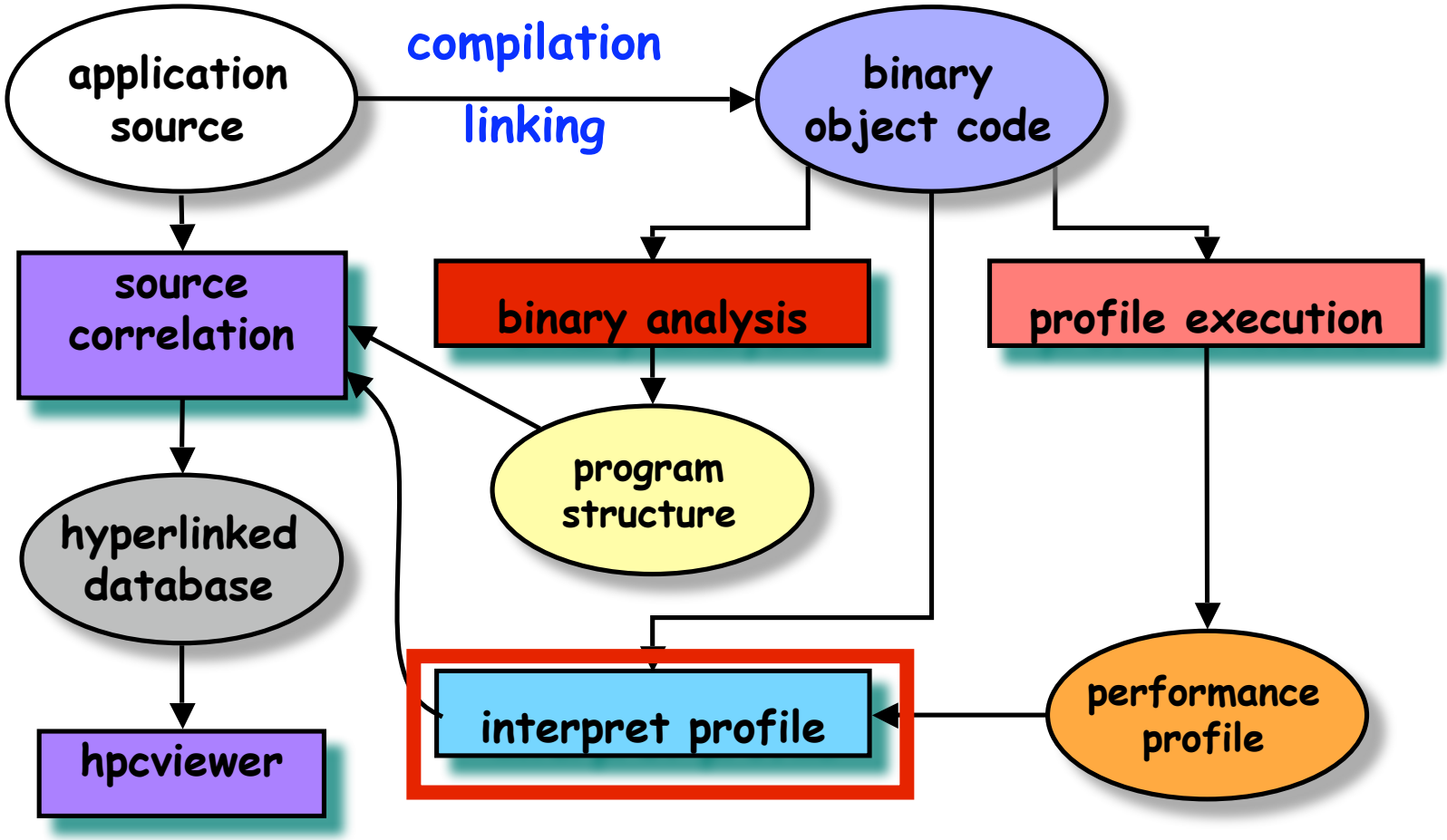


HPCToolkit Workflow



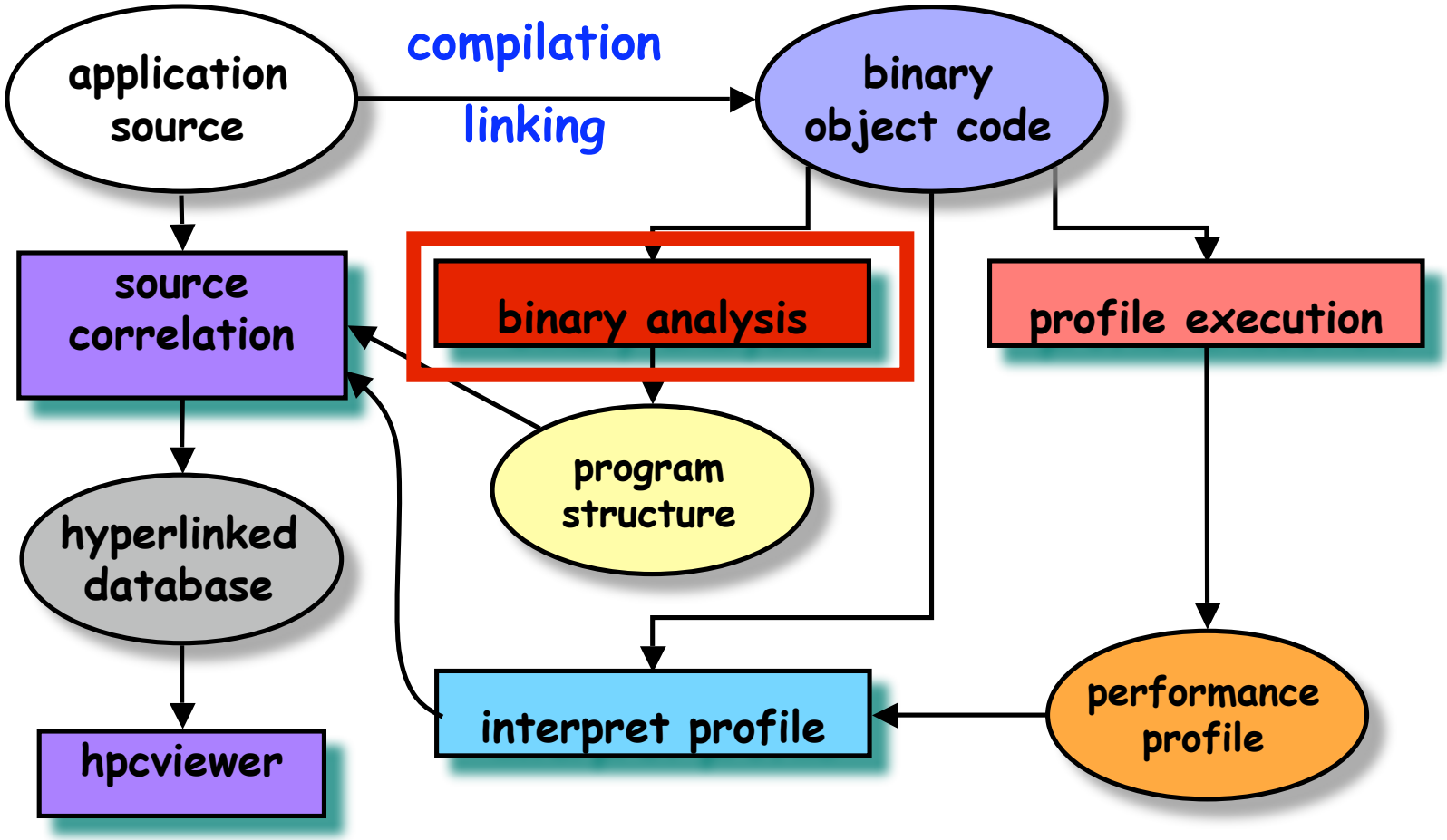
—launch unmodified, optimized application binaries
—collect statistical profiles of events of interest

HPCToolkit Workflow



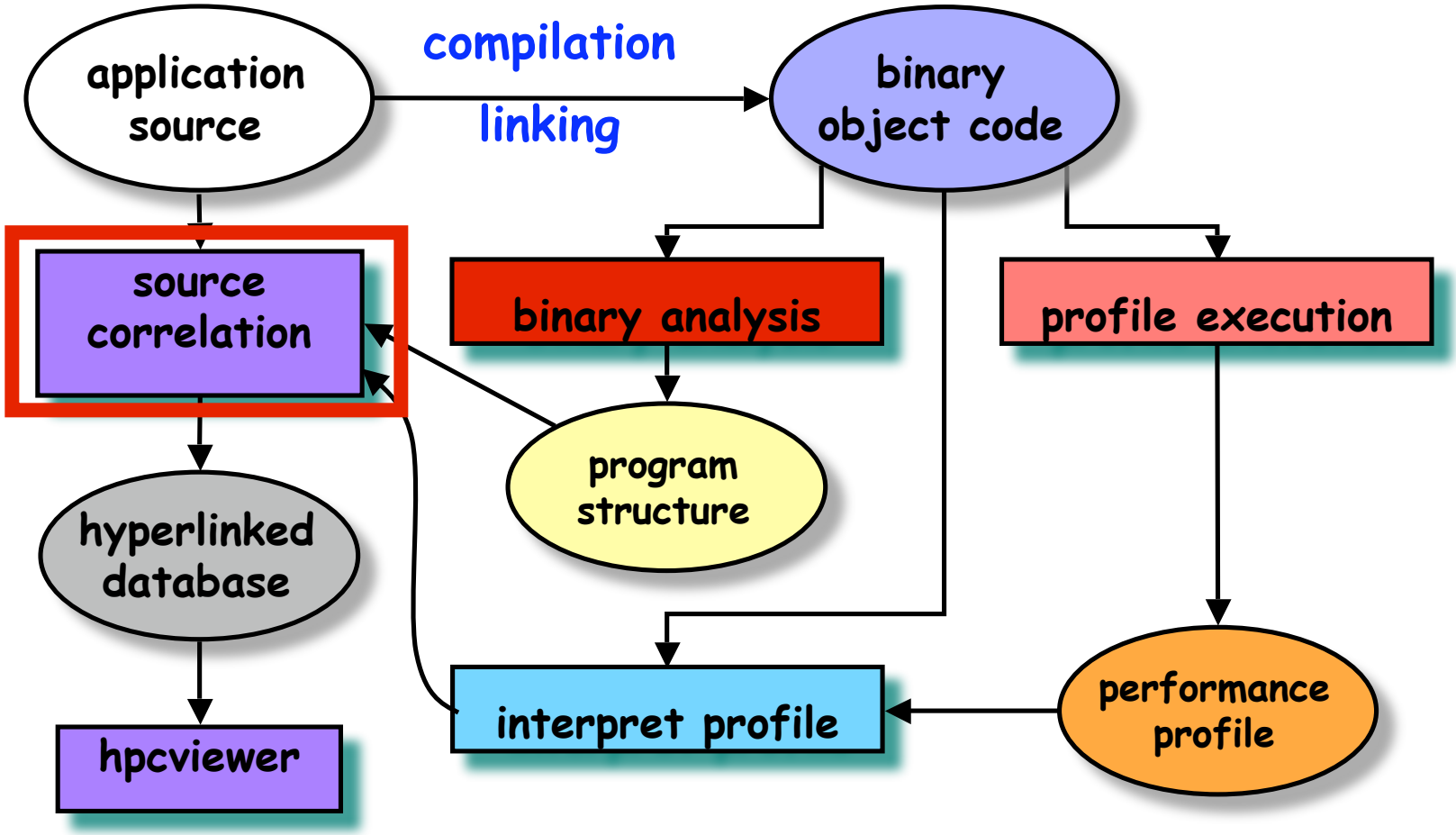
—decode instructions and combine with profile data

HPCToolkit Workflow



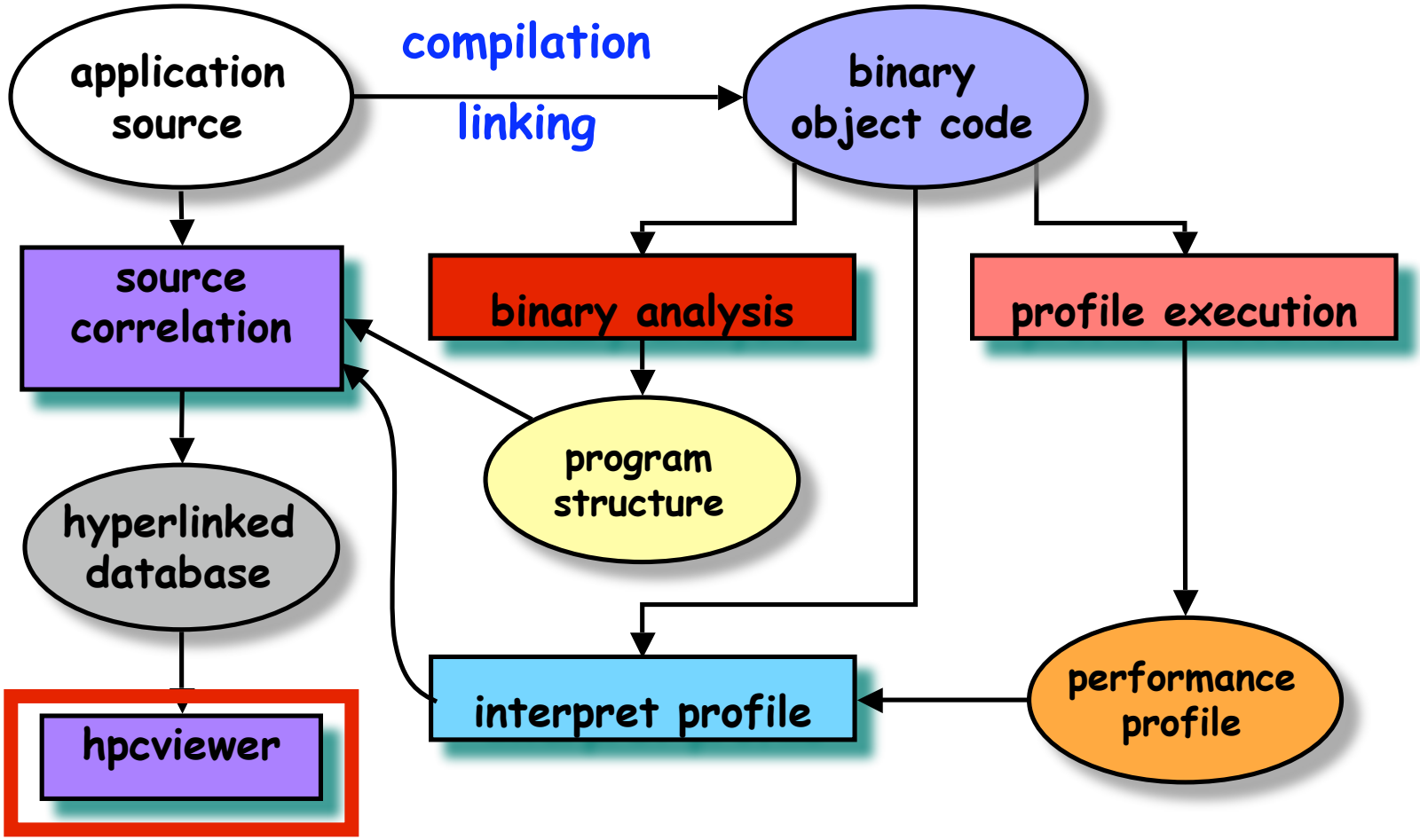
—extract loop nesting & inlining from executables

HPCToolkit Workflow



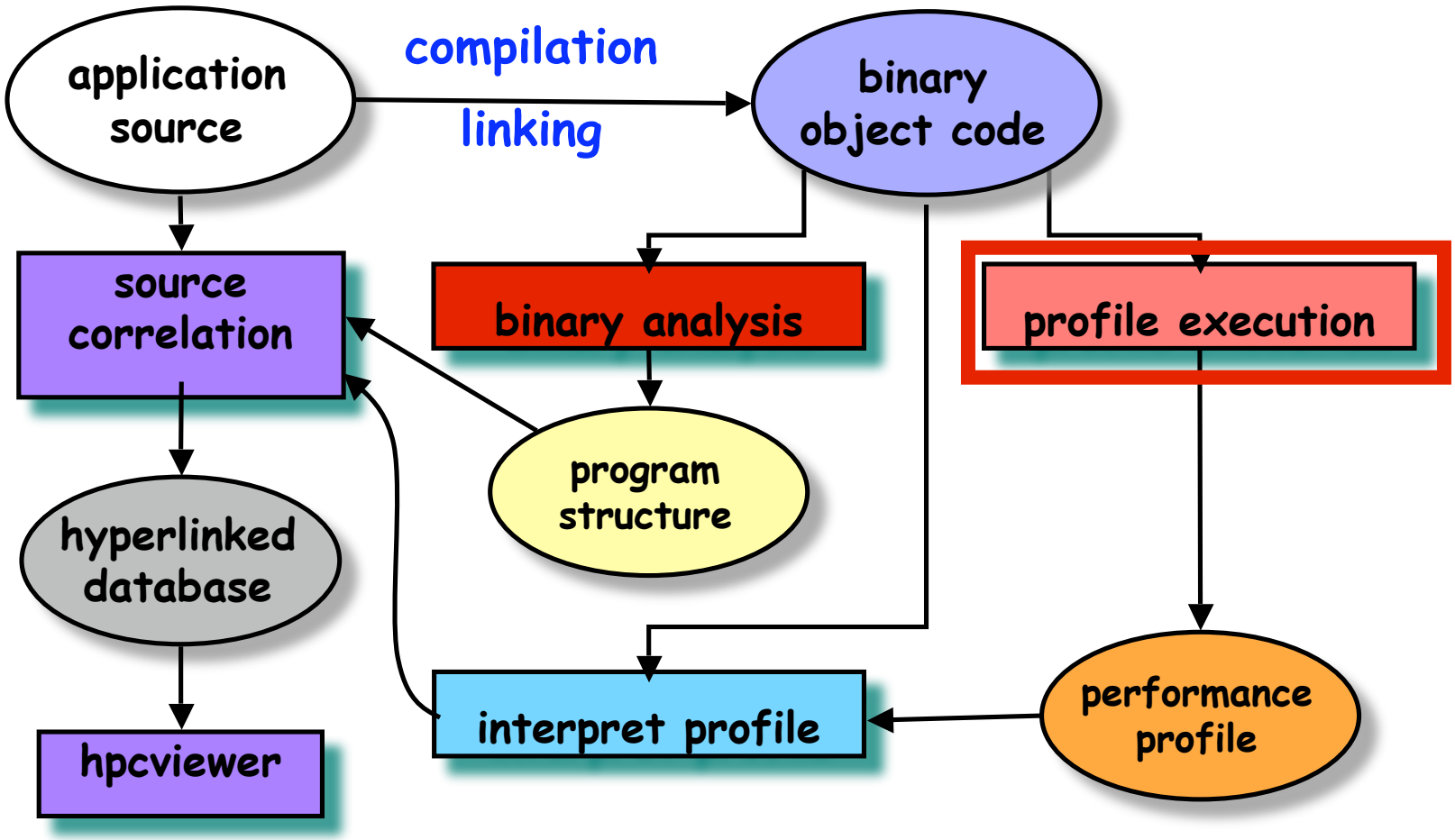
—synthesize new metrics by combining metrics
—relate metrics and structure to program source

HPCToolkit Workflow



—support top-down analysis with interactive viewer
—analyze results anytime, anywhere

HPCToolkit Workflow



Measurement Challenges

Performance often depends upon context

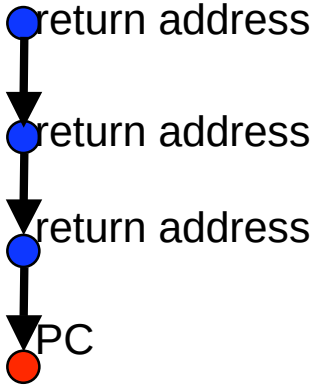
- **Layered design**
  - —application frameworks, math libraries, communication libraries
- **Generic programming, e.g. C++ templates**
  - —both data structures and algorithms
- **Context-sensitive optimization**
  - —e.g. inlining
- **Goals**
  - —identify and quantify context-sensitive behavior
  - —differentiate between types of performance problems
    - cheap procedure called many times
    - expensive procedure called few times

Call Path Profiling

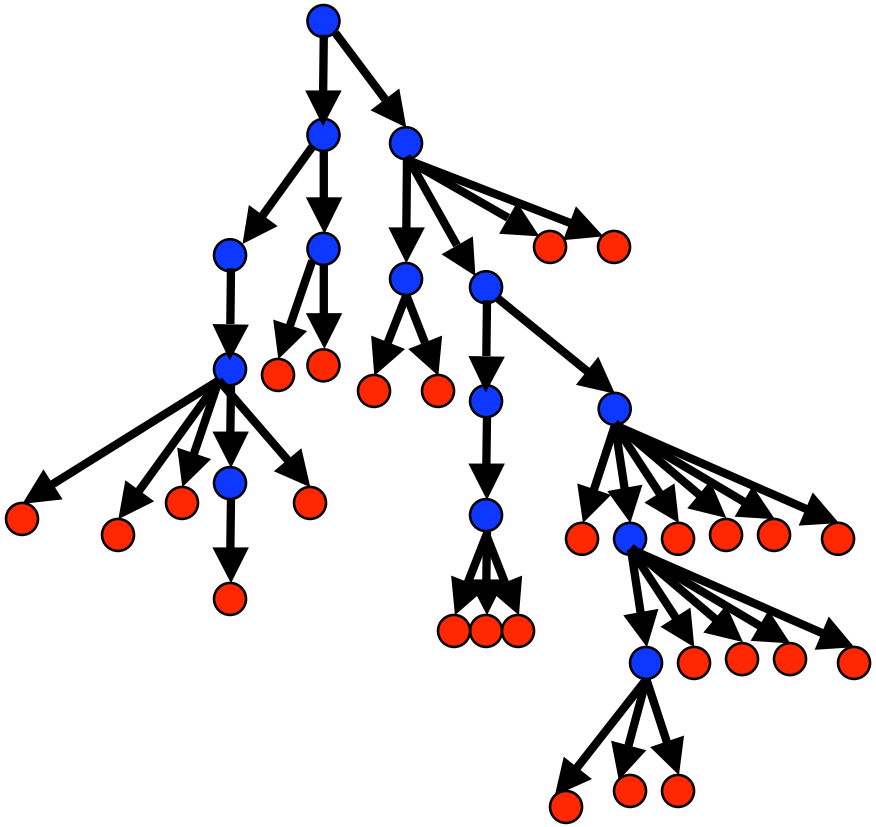
- No instrumentation
  - —statistical sampling of hardware performance counter overflows
  - —gather calling context information using stack unwinding
  - —overhead proportional to sampling frequency
    - not calling frequency
- Capture samples in full calling context
  - —attribute sample to individual PC and source line
  - —associate costs with full calling context
    - call sites too, not just callers
- Measurement overhead is only a few percent

A Call Path Profile

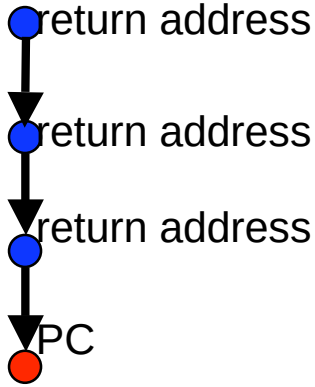
A call path sample



Calling Context Tree (CCT)



Unwinding Optimized Code

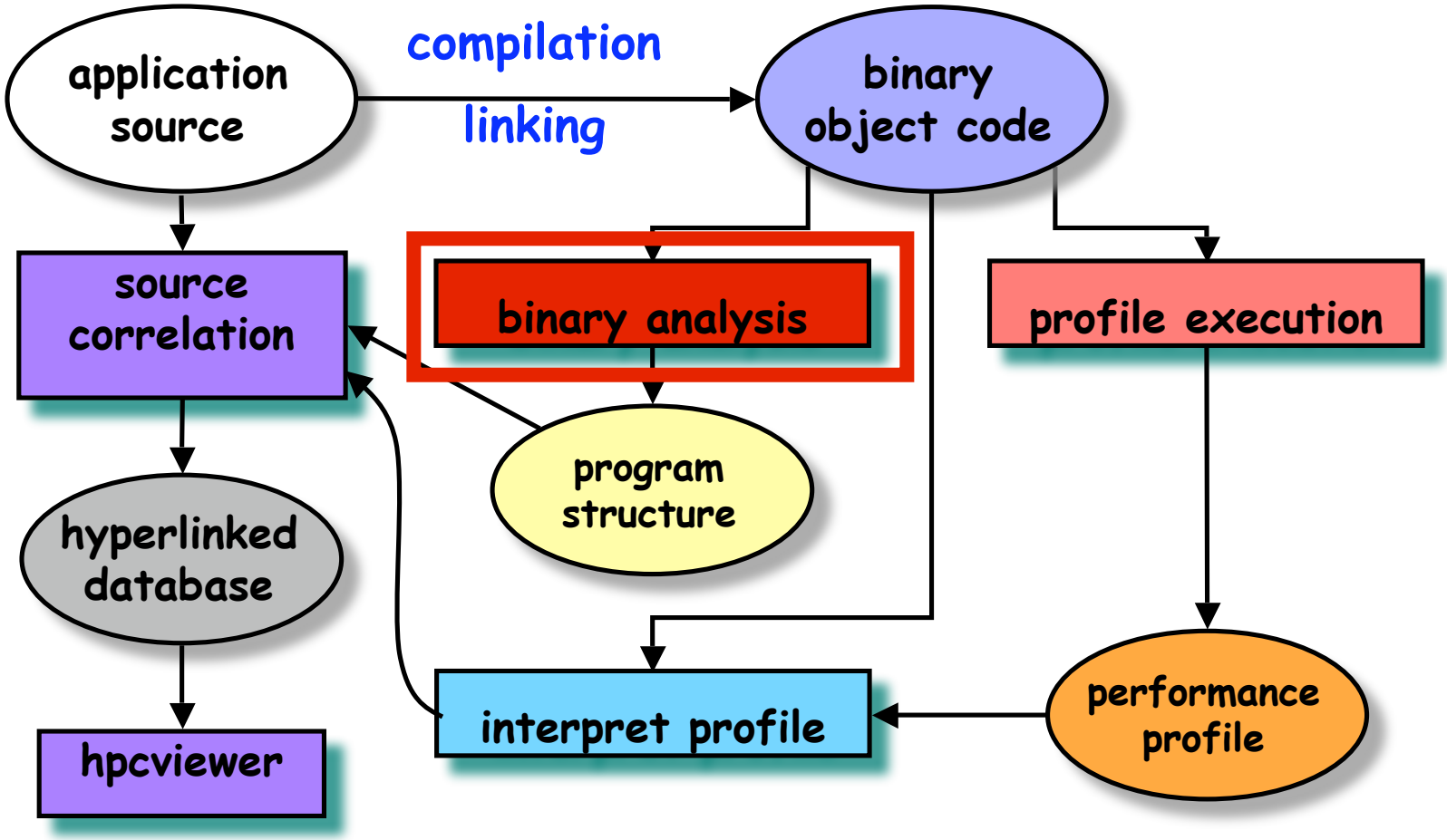


- **Information needed for unwinding**
  - —where is the return address of the current frame?
  - —where is the BP for the caller's frame?
  - —what is the caller's stack pointer value?
- **Unwinding optimized code is challenging**
  - —often lacks frame pointers
  - —little or no compiler information available
    - code may be partially stripped
  - —recipe for extracting required information varies
    - even within a single routine!
- **Approach: use binary analysis**
  - —when a module is loaded
    - locate routines and compute bounds, even for stripped code
  - —when a procedure is sampled
    - analyze instructions, determine unwind recipes, cache for later

Call Stack Unwinding Effectiveness

- **Test cases using SPEC CPU 2006 benchmarks**
  - **—combination of spec train and ref tests**
  - **—AMD Opteron**
  - **—compilers: Intel 10.0.23, Pathscale 3.1, PGI 7.0.3**
- **11M samples, dropped 234**
  - **—binary analysis is effective for unwinding optimized code**
- **Overhead: averages < 1% @ 200 samples/second**
  - **—even with on-the-fly binary analysis**

HPCToolkit Workflow

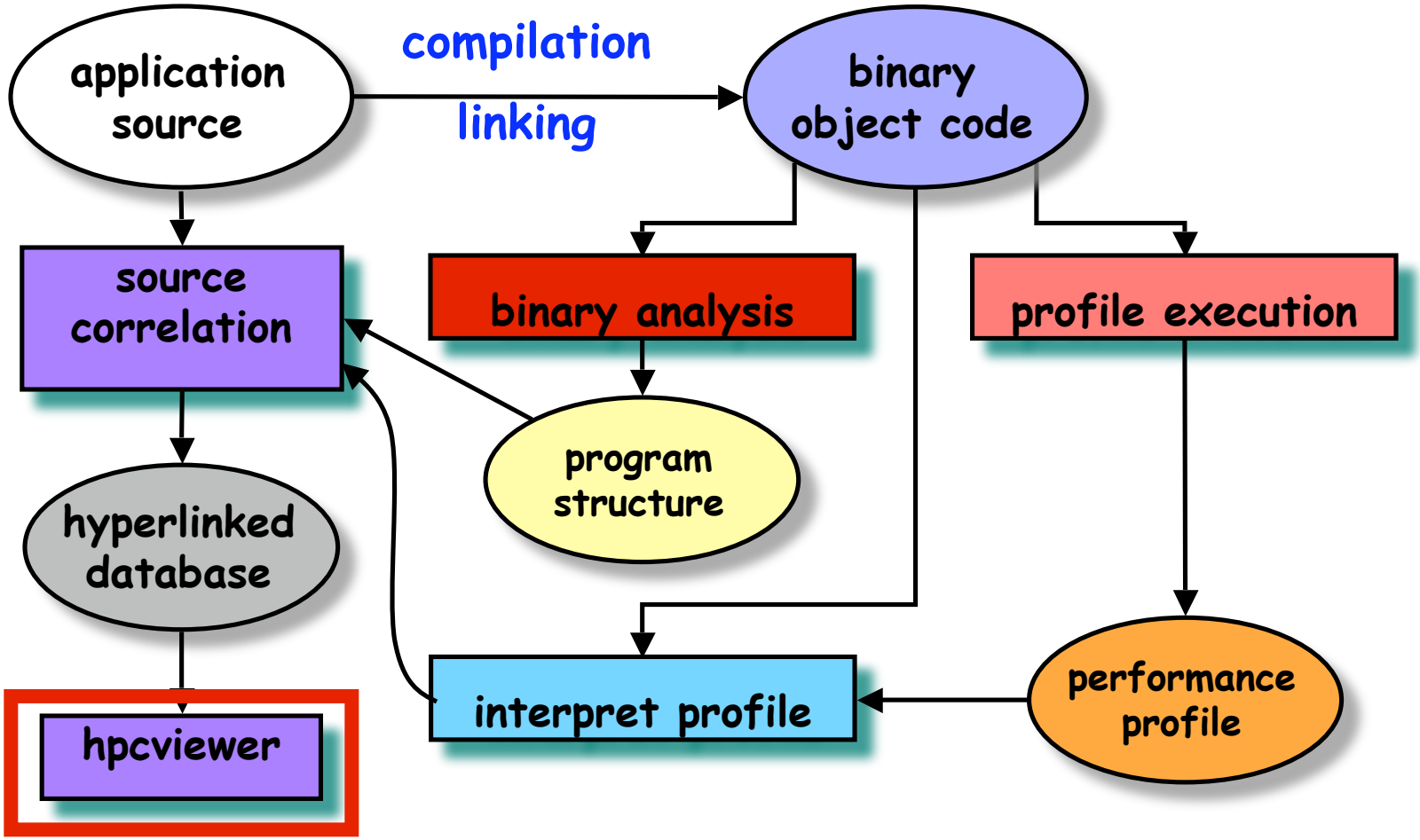


Program Structure Recovery

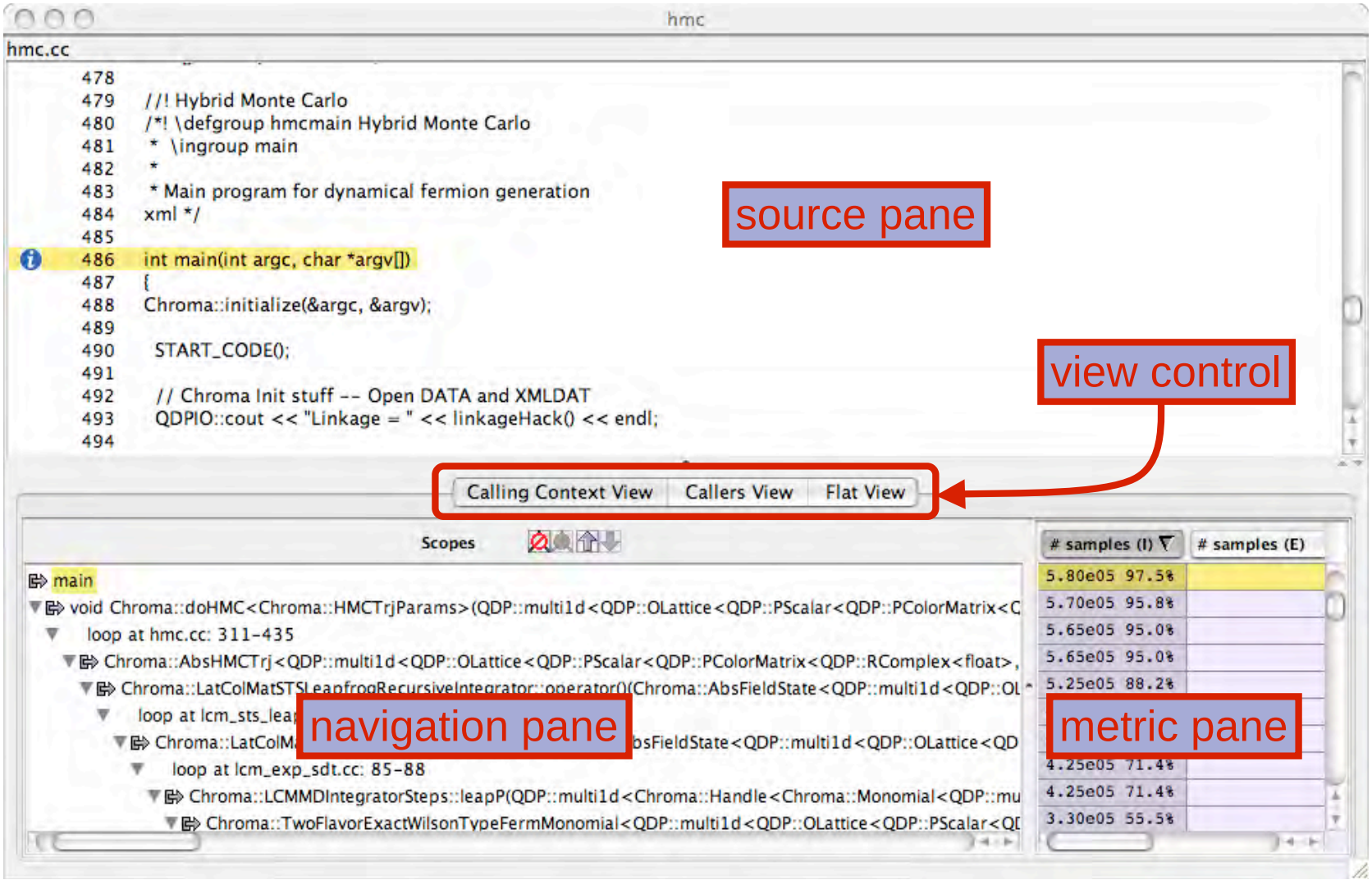
Analyze an application binary

- **Decode machine instructions**
- **Construct control flow graph from branches**
- **Identify natural loop nests using interval analysis**
- **Map instructions to source lines, procedures**
  - **—leverage line map + DWARF debugging information**
- **Discover inlined code**
- **Normalize output to recover source-level view**

HPCToolkit Workflow



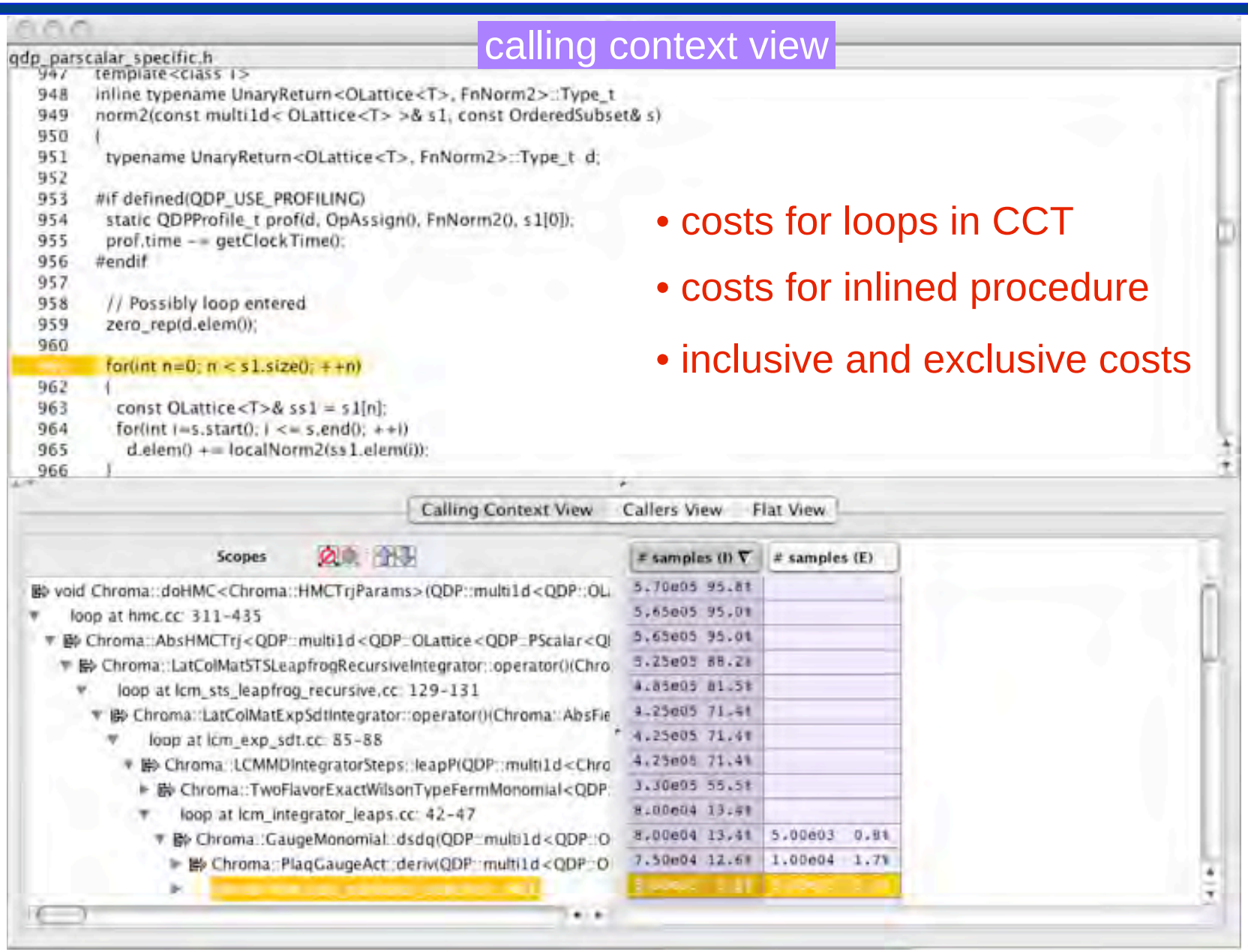
hpcviewer User Interface



Principal `hpcviewer` Views

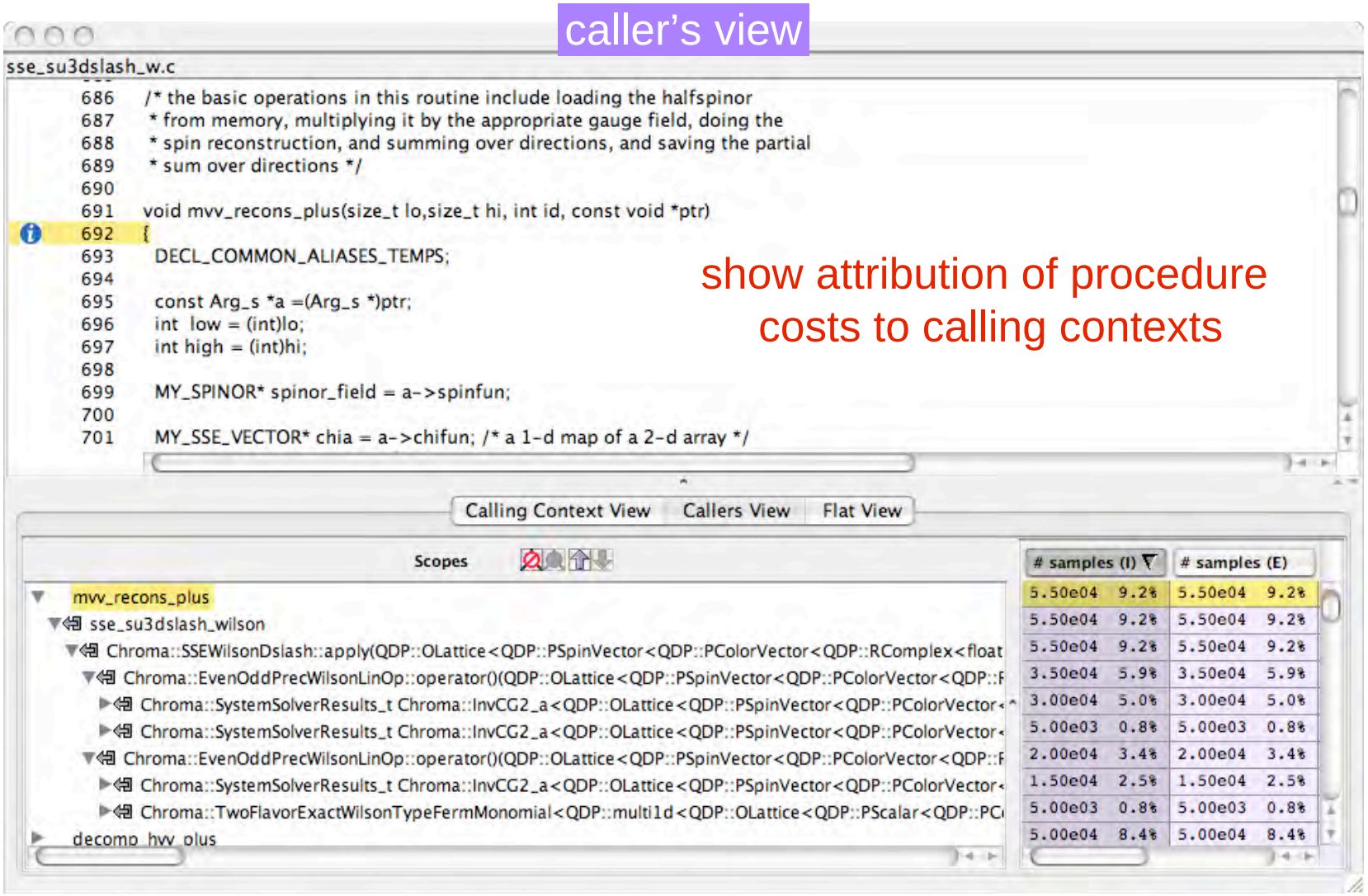
- **Calling context tree view**
  - —***top-down* view shows dynamic *calling contexts* in which costs were incurred**
- **Caller's view**
  - —***bottom-up* view apportions costs incurred in a routine to the routine's dynamic calling contexts**
- **Flat view**
  - —**aggregates all costs incurred by a routine *in any context* and shows the details of where they were incurred within the routine**

Chroma Lattice QCD Library

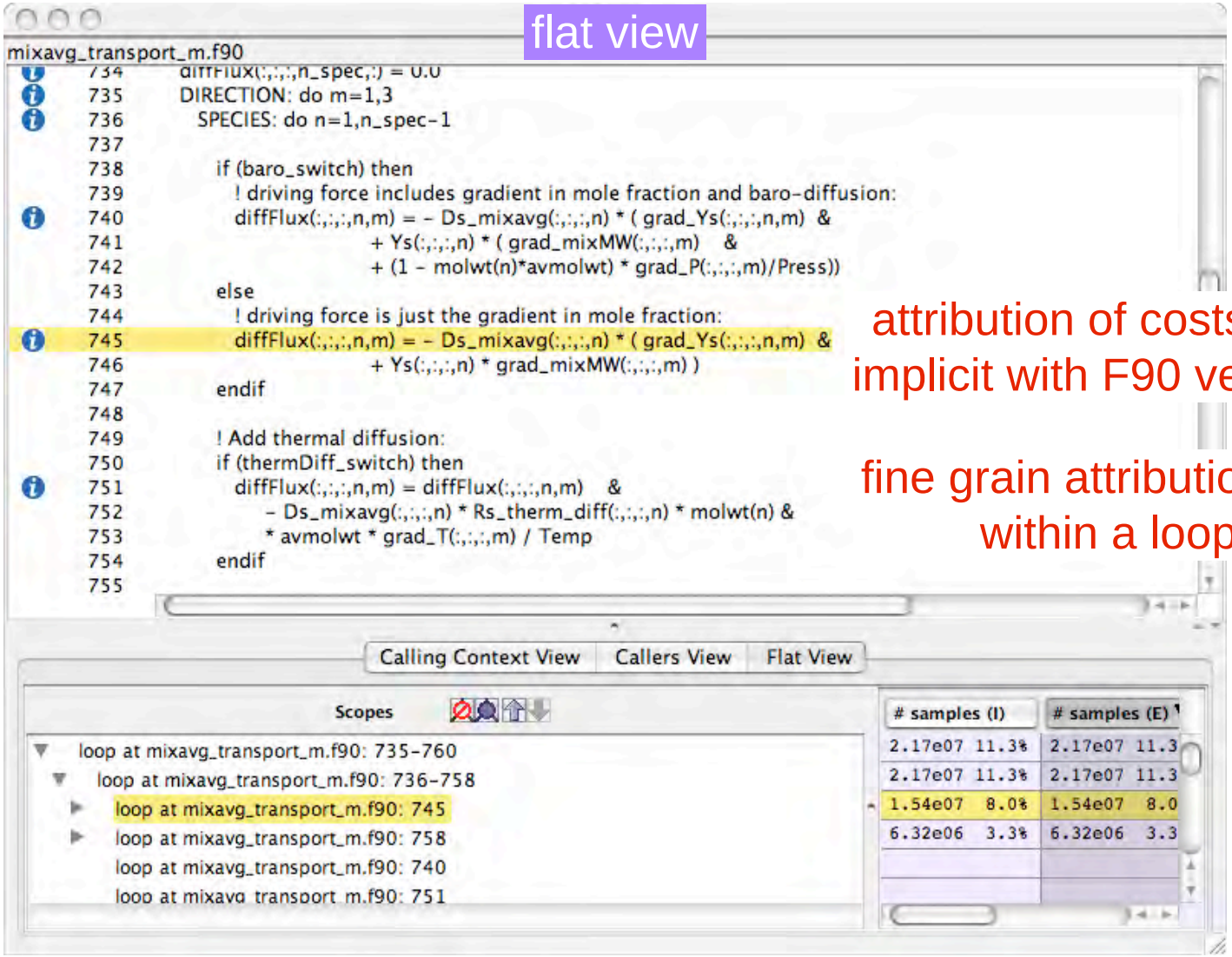


- costs for loops in CCT
- costs for inlined procedure
- inclusive and exclusive costs

Chroma Lattice QCD Library



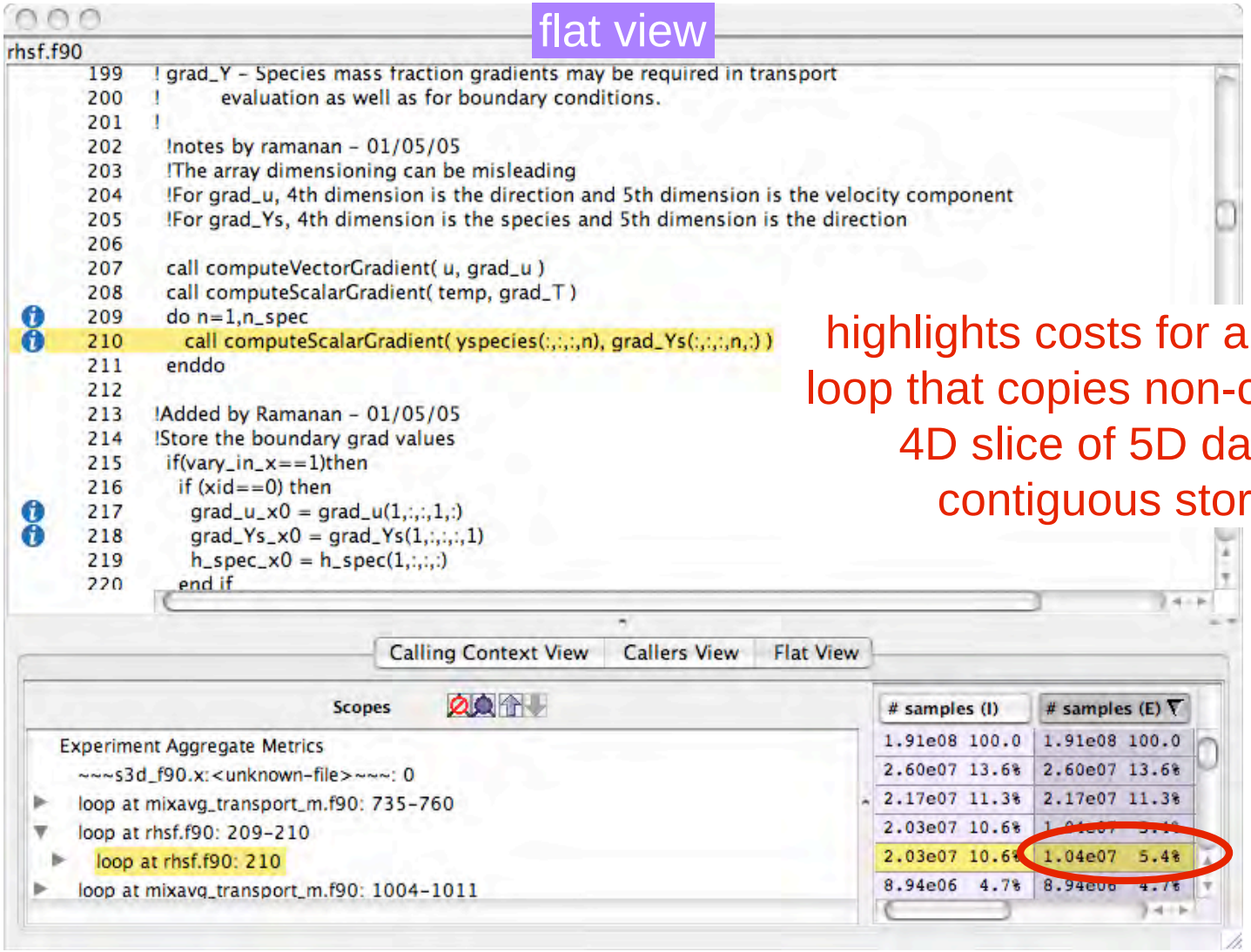
S3D Solver for Turbulent, Reacting Flows



attribution of costs to loops implicit with F90 vector syntax

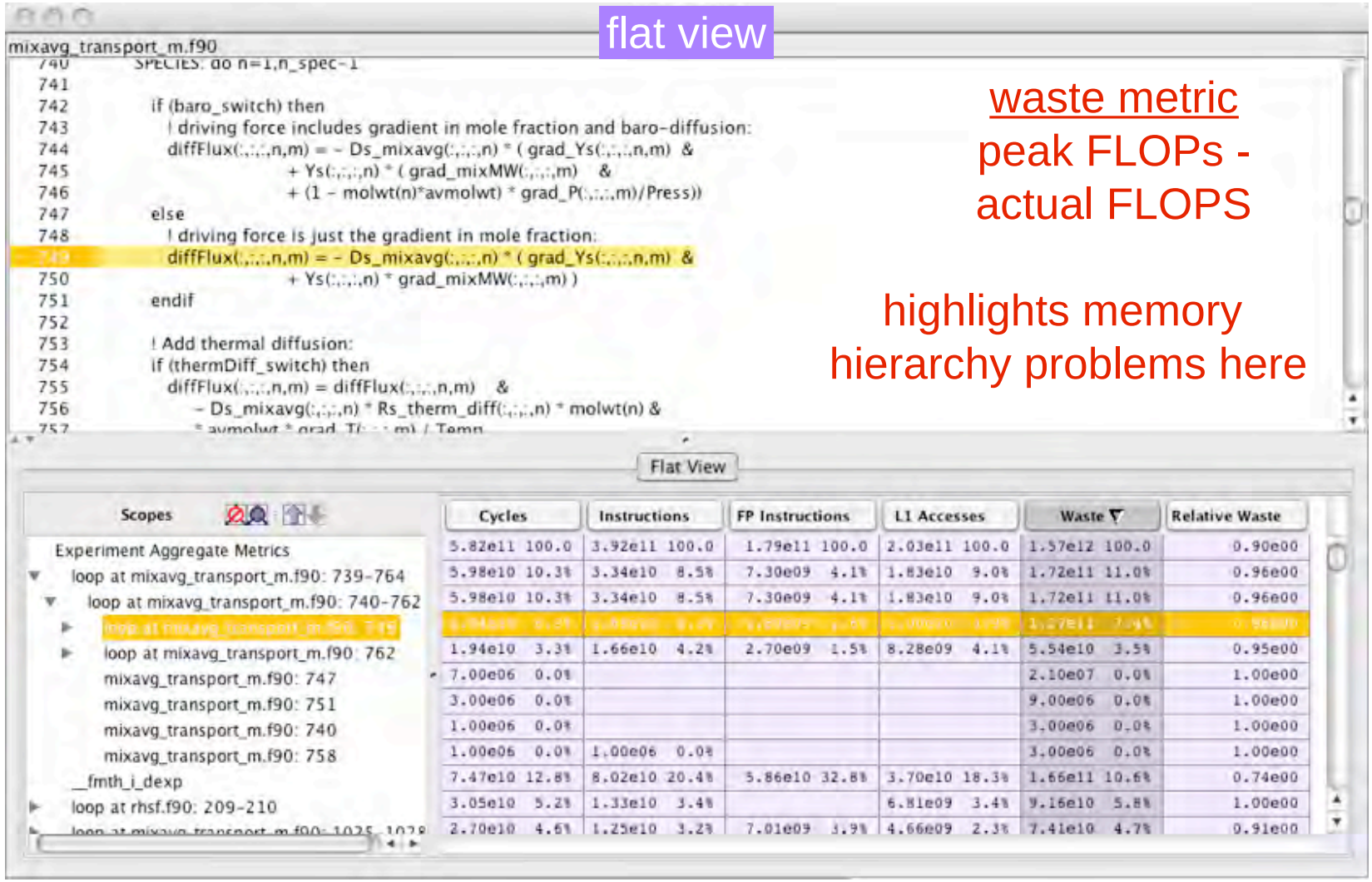
fine grain attribution to loops within a loop nest

S3D Solver for Turbulent, Reacting Flows



highlights costs for an implicit loop that copies non-contiguous 4D slice of 5D data to contiguous storage

S3D Solver for Turbulent, Reacting Flows



Ongoing Analysis of a Migration Code

- Proprietary code from a partner in the oil and gas industry
- Characteristics
  - —dynamic loading of application modules (shared libraries)
  - —partially stripped thread libraries (boost)
  - —multi-threaded for multi-core
- Collected call path profiles of unmodified code
- Integrated views of multiple threads
- Analyzed multiple metrics with hardware counters
  - —cycles, floating point operations, cache, address translation
  - —clear understanding of costs in context
- Assessment
  - —overall machine utilization is high
  - —identified some potential opportunities for improvement
    - next step: discussion with the application experts

HPCToolkit Status

- **Toolkit available as open source**
  - **—flat profiling today**
  - **—call path profiling being prepared for release this spring**
  - **—build system overhaul**
- **System requirements**
  - **—Linux OS**
  - **—hardware performance counter kernel patches necessary**