HPCToolkit : Multi-platform Tools for Profile-based Performance Analysis

John Mellor-Crummey

Robert Fowler   Nathan Tallent   Gabriel Marin

Department of Computer Science
Rice University

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

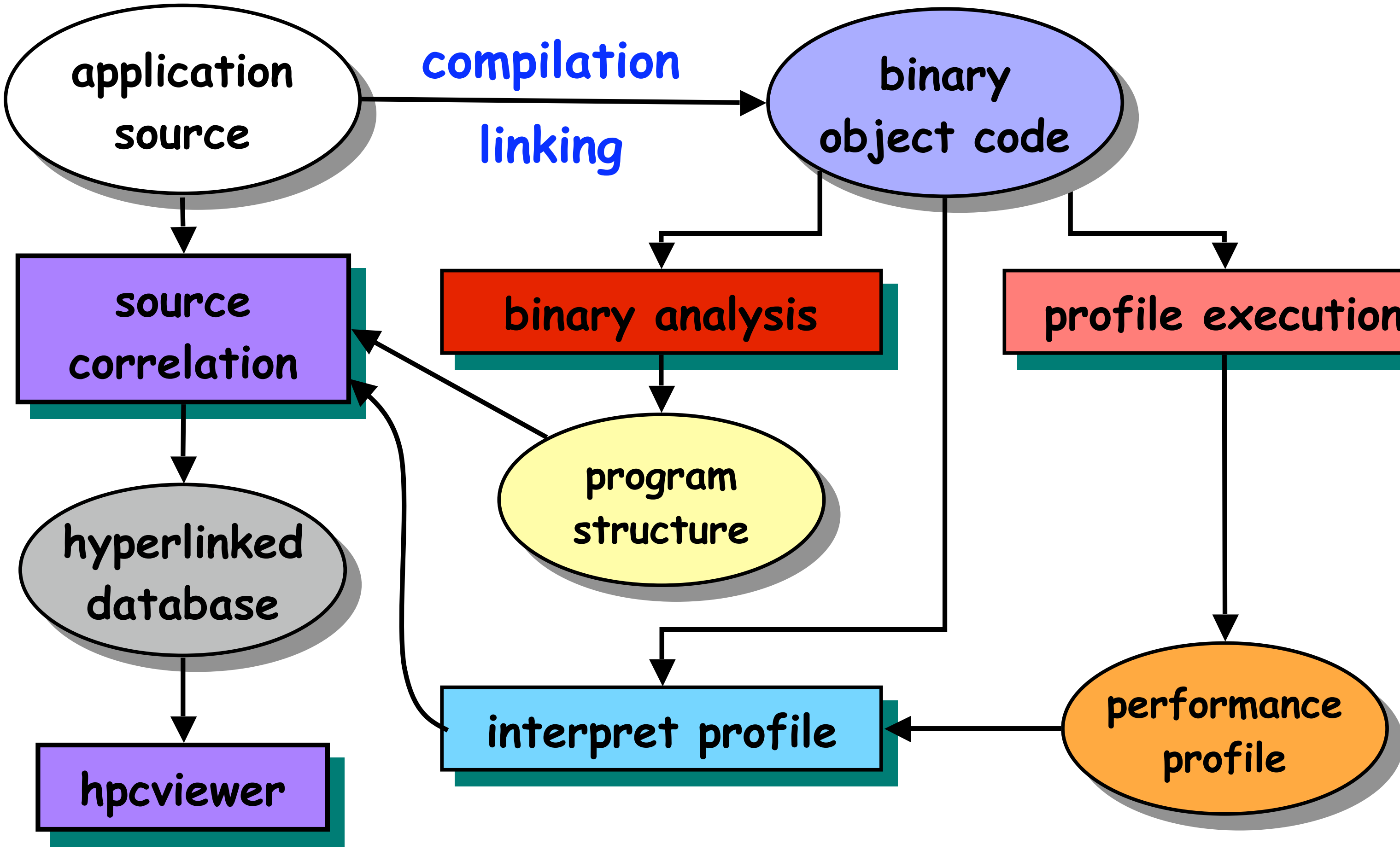
Performance Analysis and Tuning

- **Increasingly necessary**
  - —gap between typical and peak performance is growing
- **Increasingly hard**
  - —complex architectures are harder to program effectively
    - – complex processors
      - VLIW
      - deeply pipelined, out of order, superscalar
    - – complex memory hierarchy
      - non-blocking, multi-level caches
      - TLB
  - —modern scientific applications pose challenges for tools
    - – multi-lingual programs
    - – many source files
    - – complex build process
    - – external libraries in binary-only form

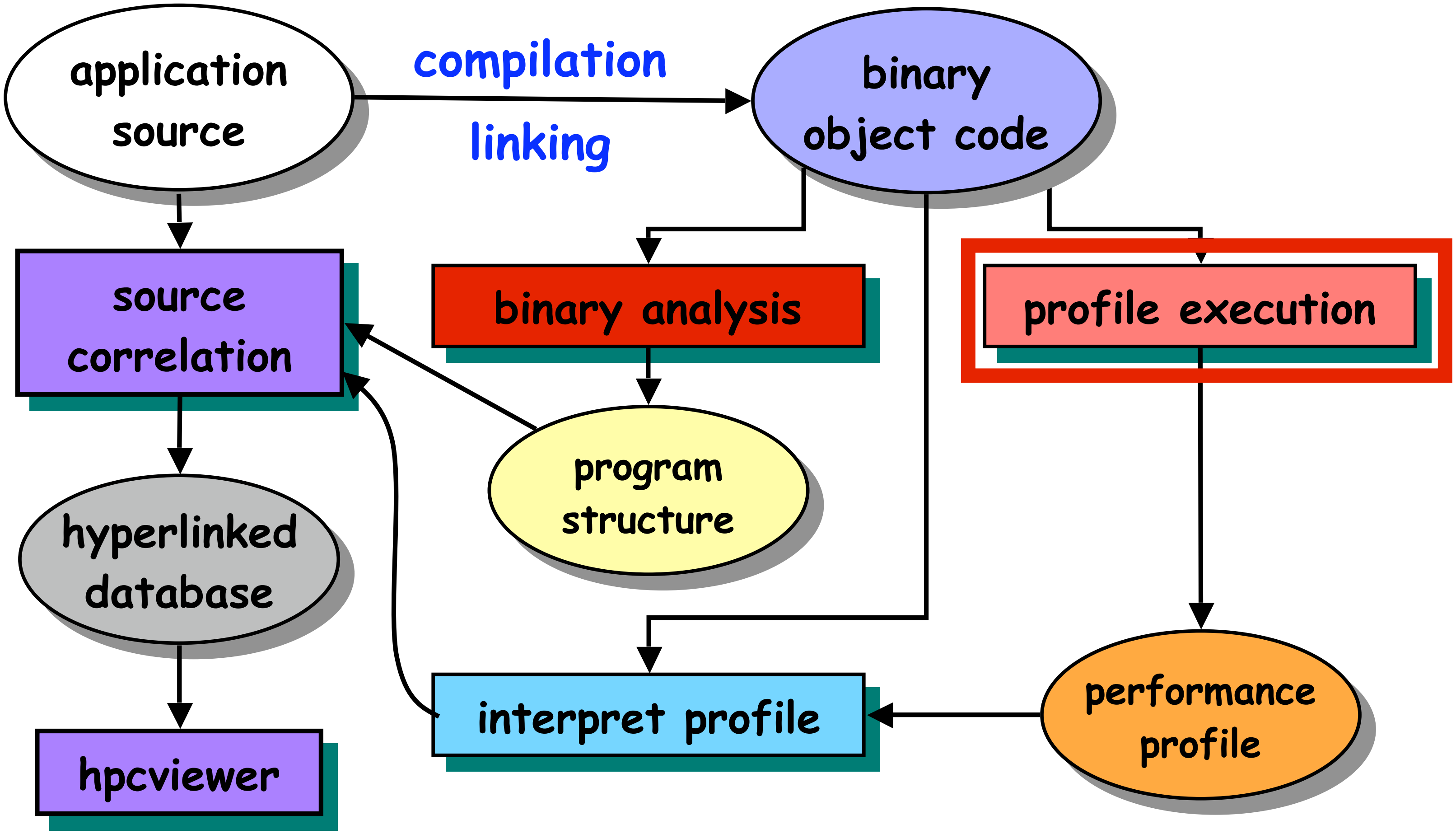
HPCToolkit Goals

- **Support large, multi-lingual applications**
  - —a mix of of Fortran, C, C++
  - —external libraries
  - —thousands of procedures
  - —hundreds of thousands of lines
  - —we must avoid
    - – manual instrumentation
    - – significantly altering the build process
    - – frequent recompilation
- **Multi-platform**
- **Scalable data collection**
- **Analyze both serial and parallel codes**
- **Effective presentation of analysis results**
  - —intuitive enough for physicists and engineers to use
  - —detailed enough to meet the needs of compiler writers

HPCToolkit System Overview

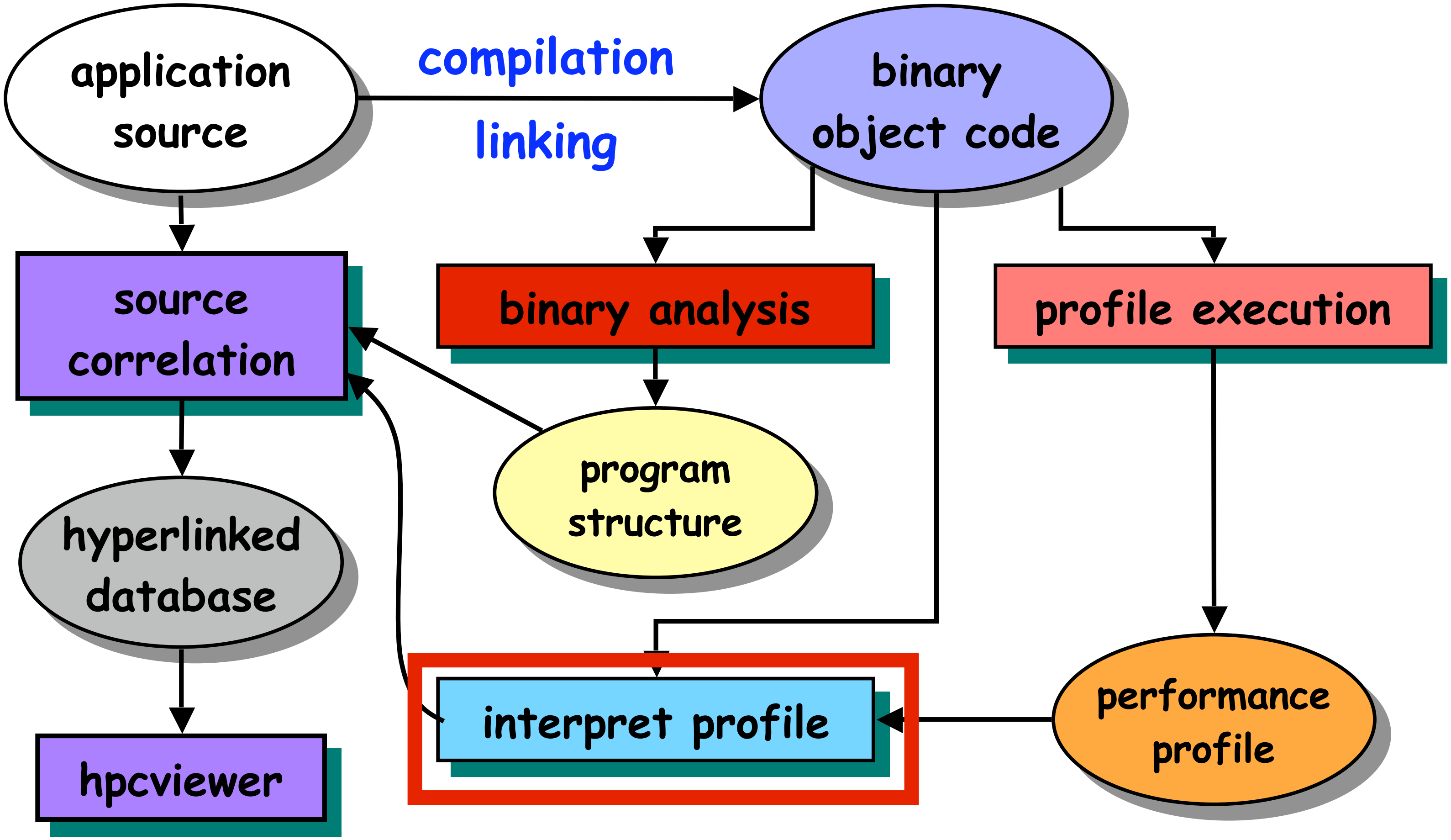


HPCToolkit System Overview



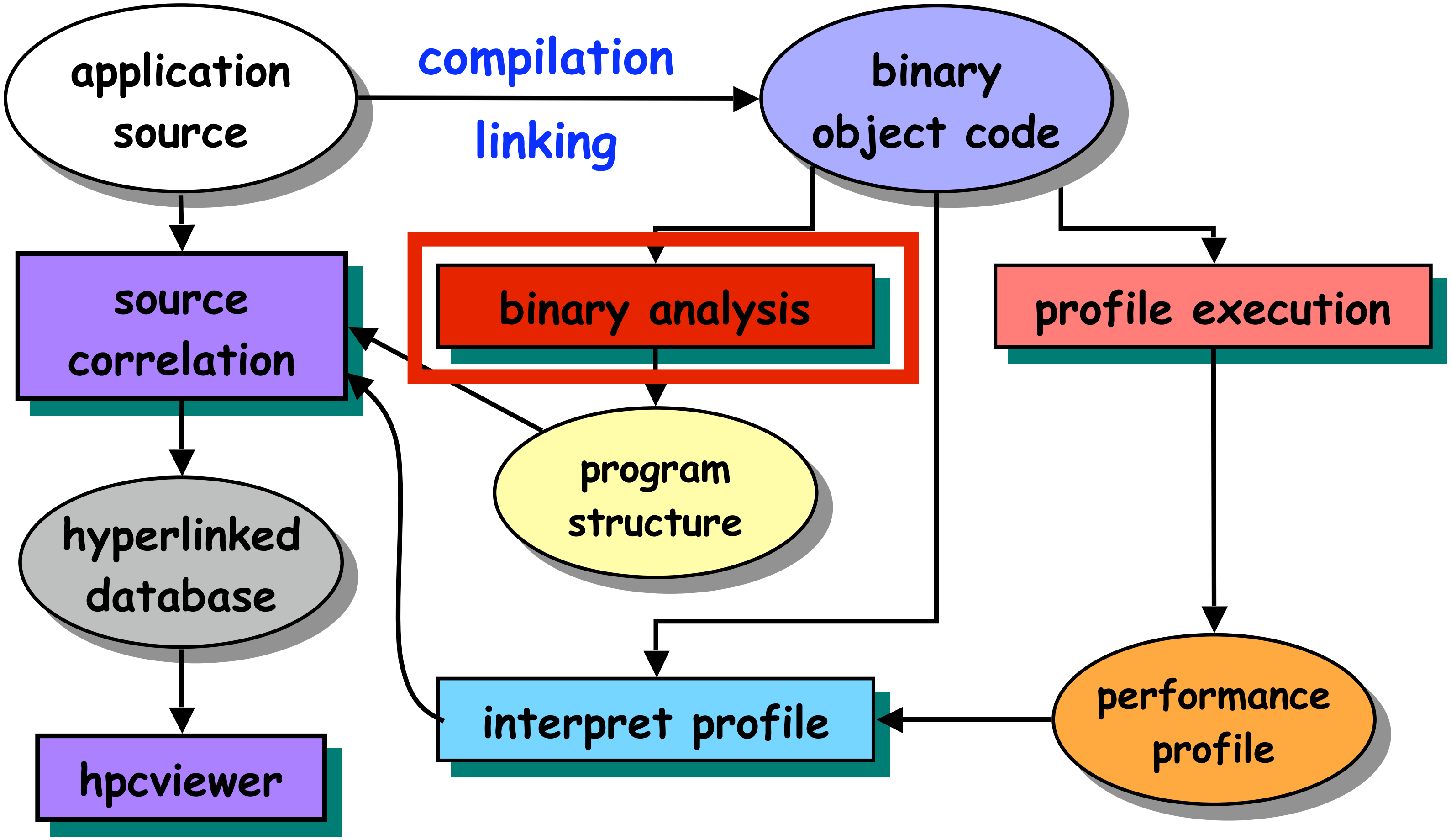
- —launch unmodified, optimized application binaries
- —collect statistical profiles of events of interest

HPCToolkit System Overview



—decode instructions and combine with profile data

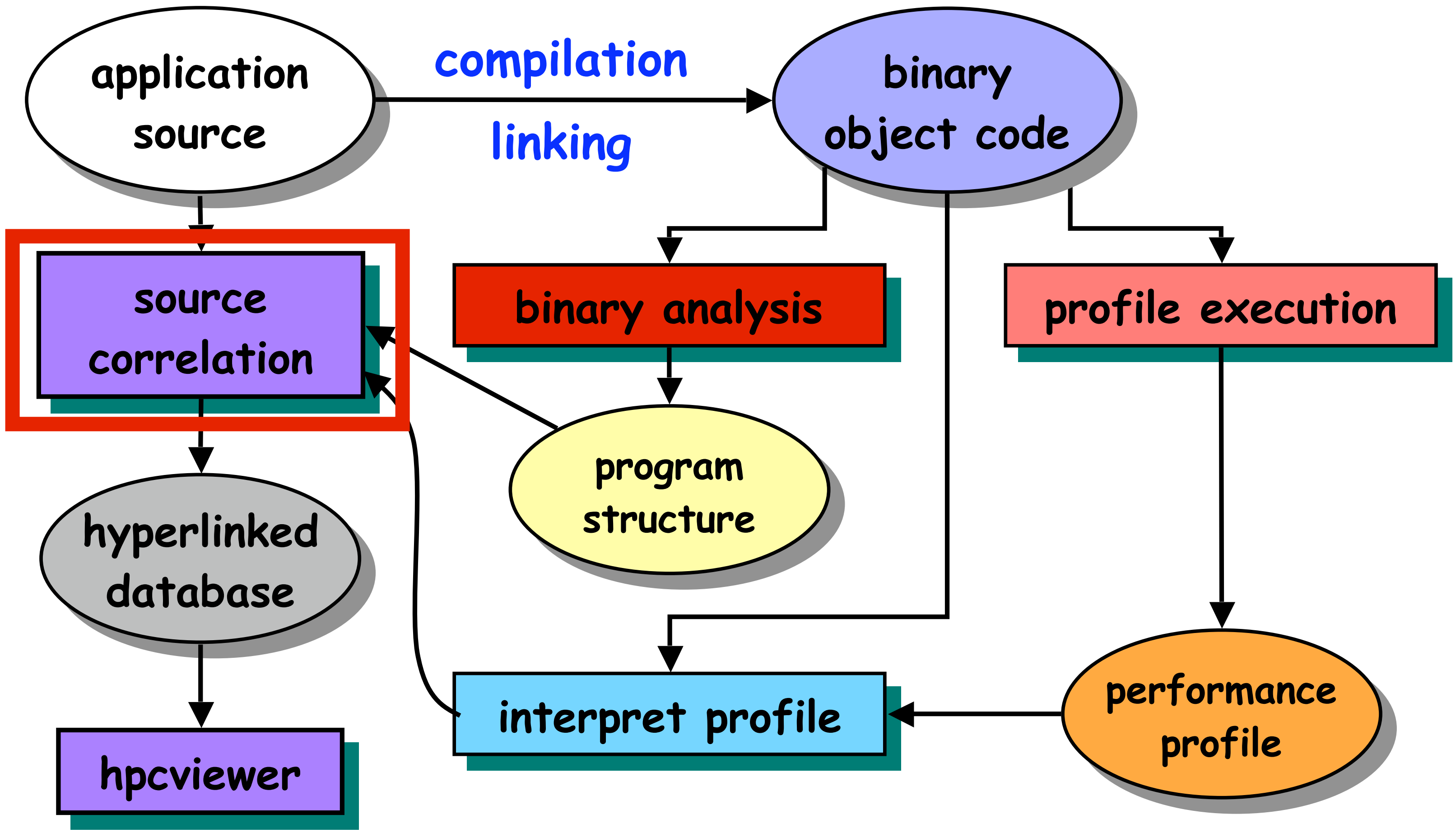
HPCToolkit System Overview



—extract loop nesting information from executables

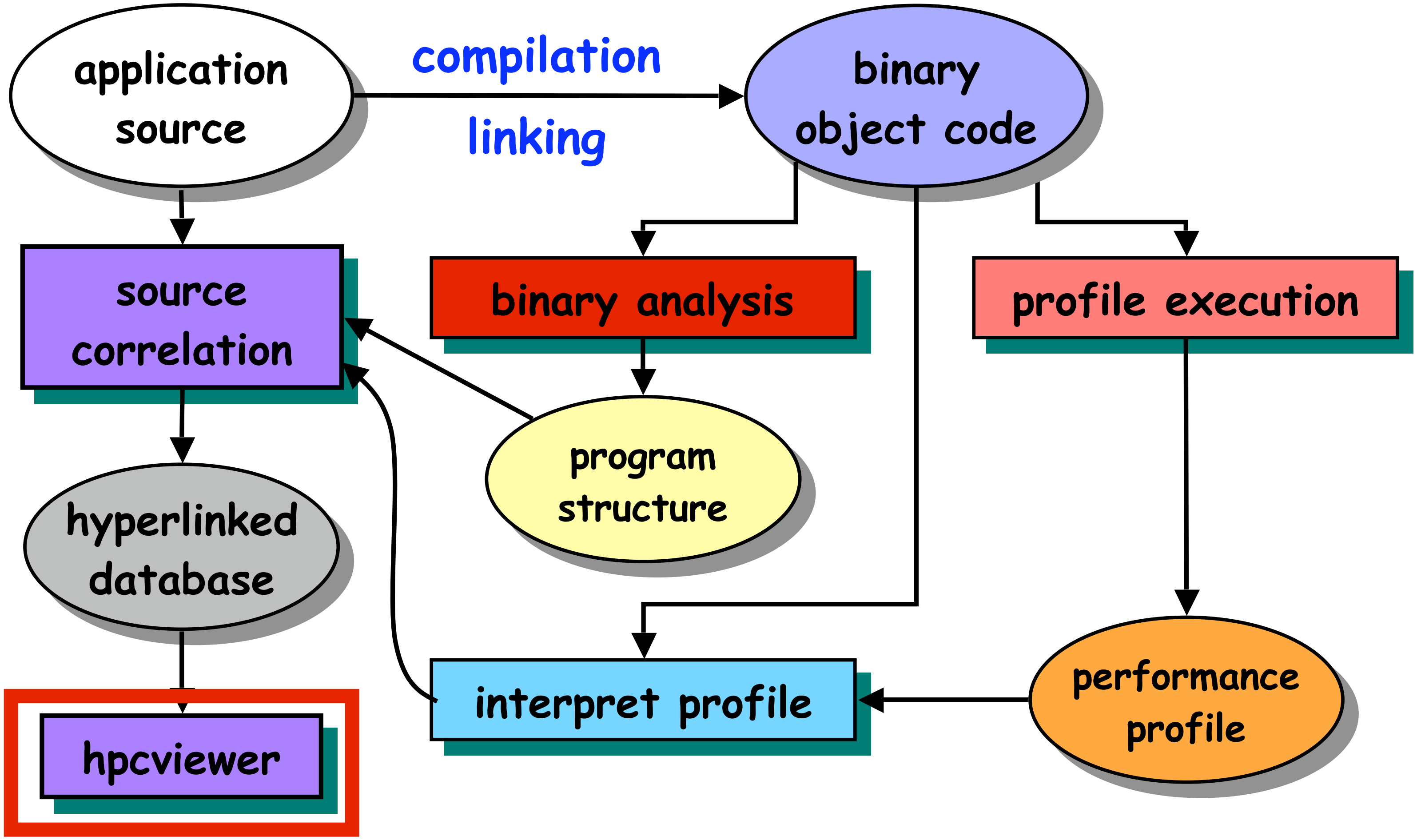
http://hipersoft.cs.rice.edu/hpctoolkit/

HPCToolkit System Overview



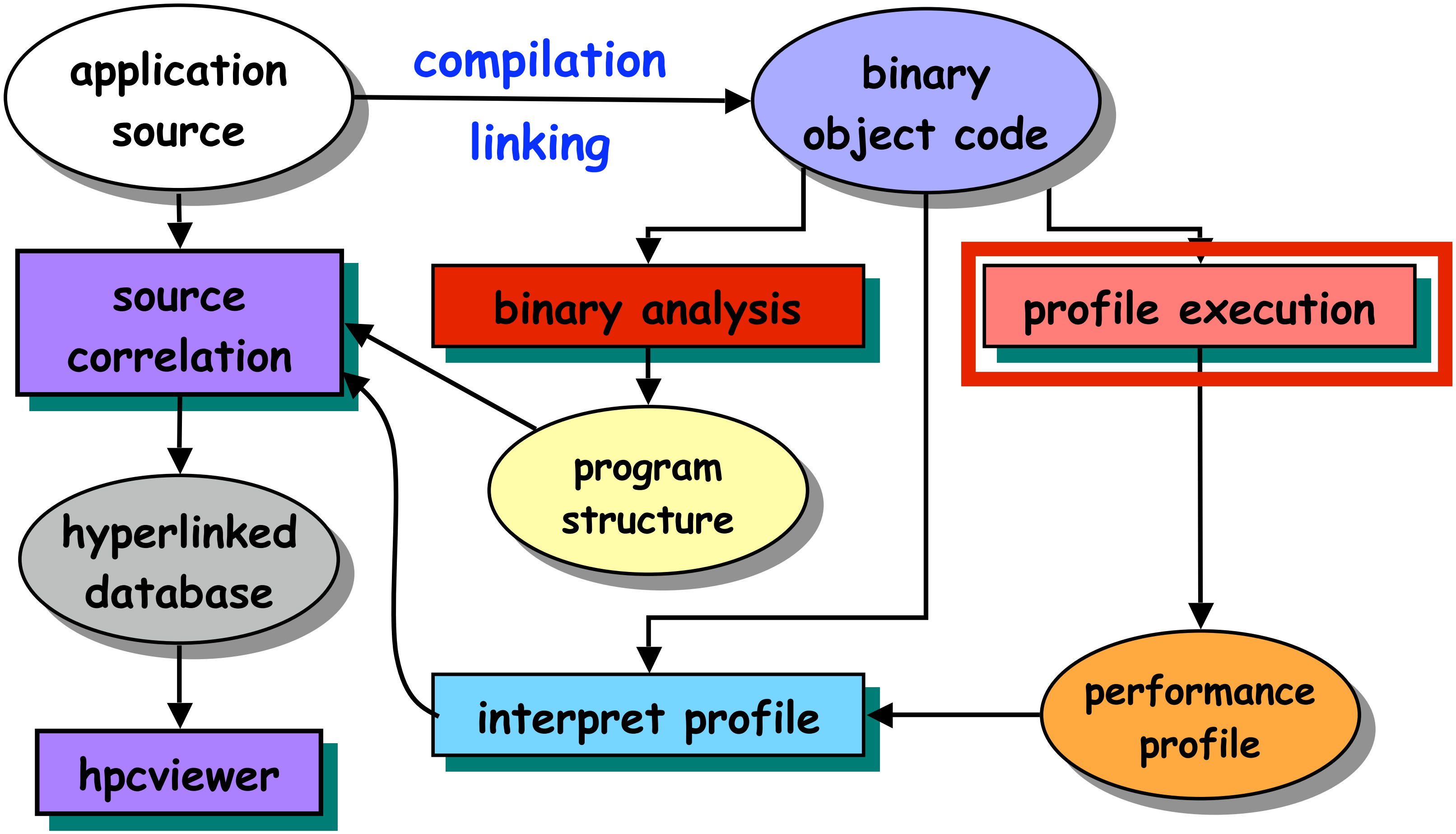
- synthesize new metrics by combining metrics
- relate metrics, structure, and program source

HPCToolkit System Overview



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

HPCToolkit System Overview



http://hipersoft.cs.rice.edu/hpctoolkit/

Data Collection

Support analysis of unmodified, optimized binaries

- **Inserting code to start, stop and read counters has many drawbacks, *so don't do it!***
  - —nested measurements skew results
- **Use hardware performance monitoring to collect statistical profiles of events of interest**
- **Different platforms have different capabilities**
  - —event-based counters: MIPS, IA64, Pentium
  - —ProfileMe instruction tracing: Alpha
- **Different capabilities require different approaches**

Data Collection Tools

Goal: limit development to essentials only

- **MIPS-IRIX:**
  - ssrun + prof → ptran
- **Alpha-Tru64:**
  - uprofile + prof → ptran
  - DCPI/ProfileMe → xprof
- **IA64-Linux and IA32-Linux**
  - papirun/papiprof

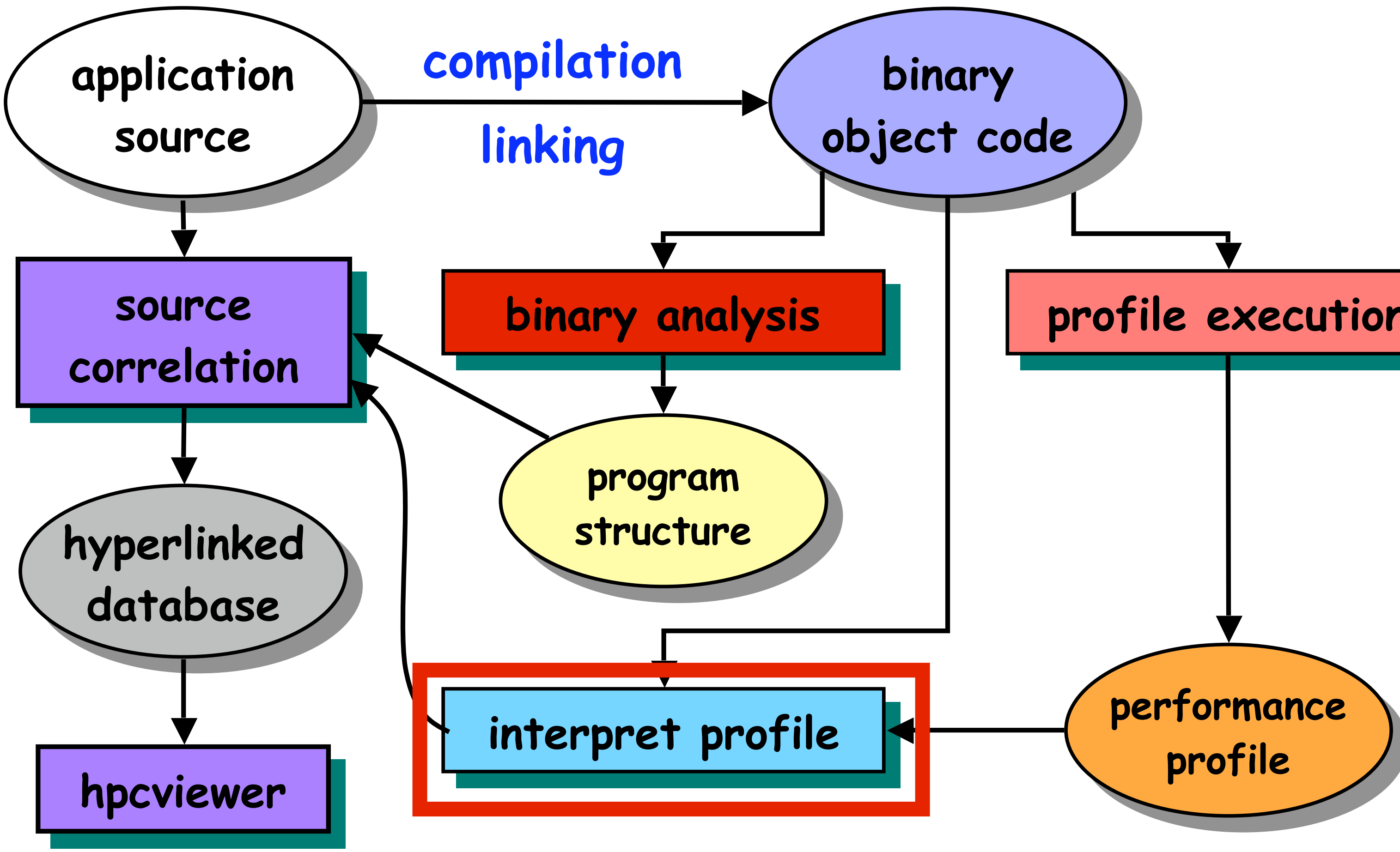
papirun/papiprof

- **PAPI: Performance API**
  - —interface to hardware performance monitors
  - —supports many platforms
- **papirun: open source equivalent of SGI's 'ssrun'**
  - —sample-based profiling of an execution
    - – preload monitoring library before launching application
    - – inspect load map to set up sampling for all load modules
    - – record PC samples for each module along with load map
  - —Linux IA64 and IA32
- **papiprof: 'prof'-like tool**
  - —based on Curtis Janssen's vprof
  - —uses GNU binutils to perform PC ➔ source mapping
  - —output styles
    - – XML for use with hpcview
    - – plain text

DCPI and ProfileMe

- **Alpha ProfileMe**
  - EV67+ records info about an instruction as it executes
    - mispredicted branches, memory access replay traps
    - more accurate attribution of events
- **DCPI: (Digital) Continuous Profiling Infrastructure**
  - sample processor counters and instructions continuously during execution of *all* code
    - all programs
    - shared libraries
    - operating system
  - support both on-line and off-line data analysis
    - to date, we use only off-line analysis

HPCToolkit System Overview

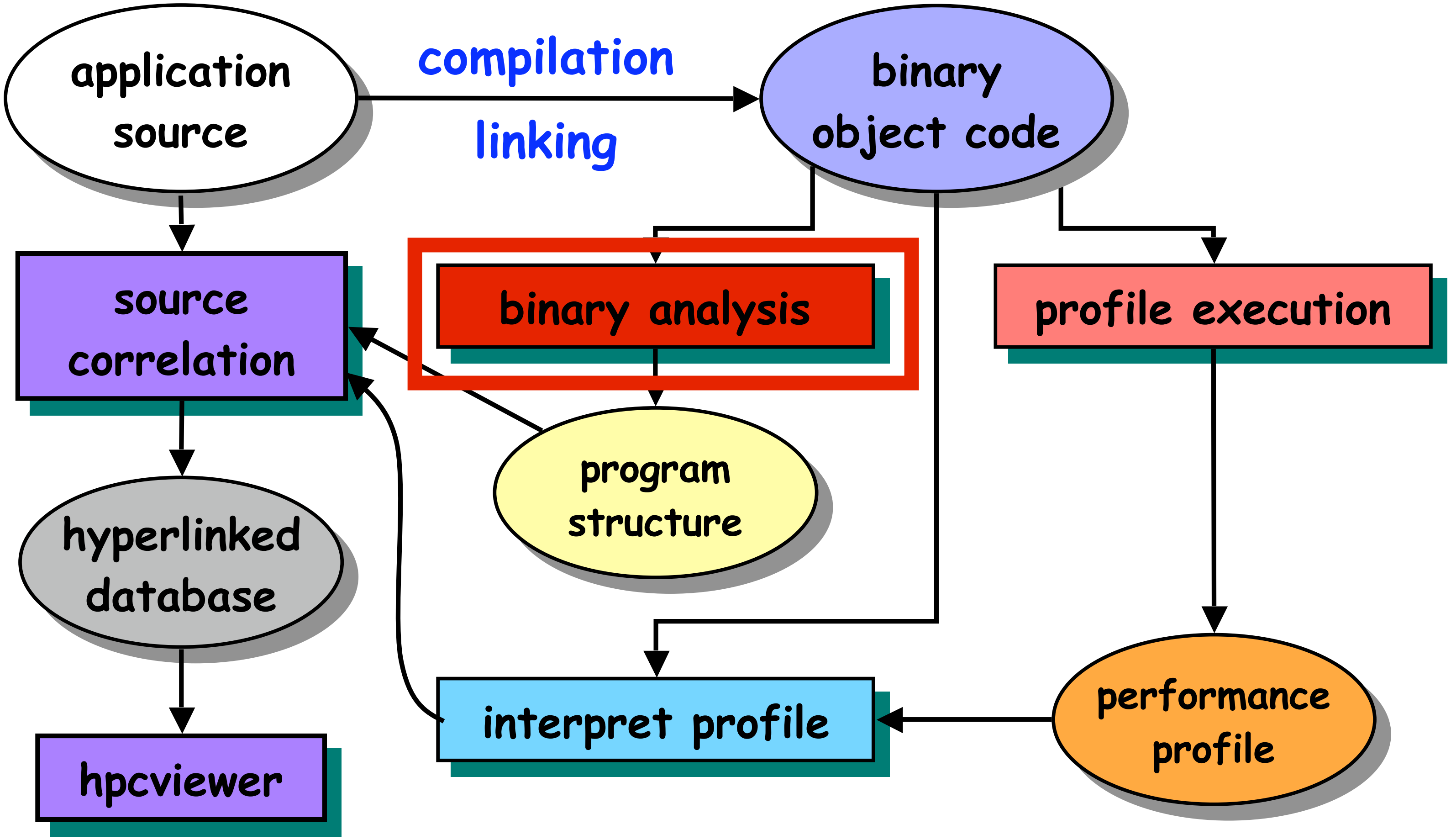


Metric Synthesis with xprof (Alpha)

Interpret DCPI samples into useful metrics

- **Transform low-level data to higher-level metrics**
  - —DCPI ProfileMe information associated with PC values
  - —project ProfileMe data into useful equivalence classes
  - —decode instruction type info in application binary at each PC
    - – FLOP
    - – memory operation
    - – integer operation
  - —fuse the two kinds of information
    - – Retired instructions + instruction type =
      - retired FLOPs
      - retired integer operations
      - retired memory operations
- **Map back to source code like papiprof**

HPCToolkit System Overview

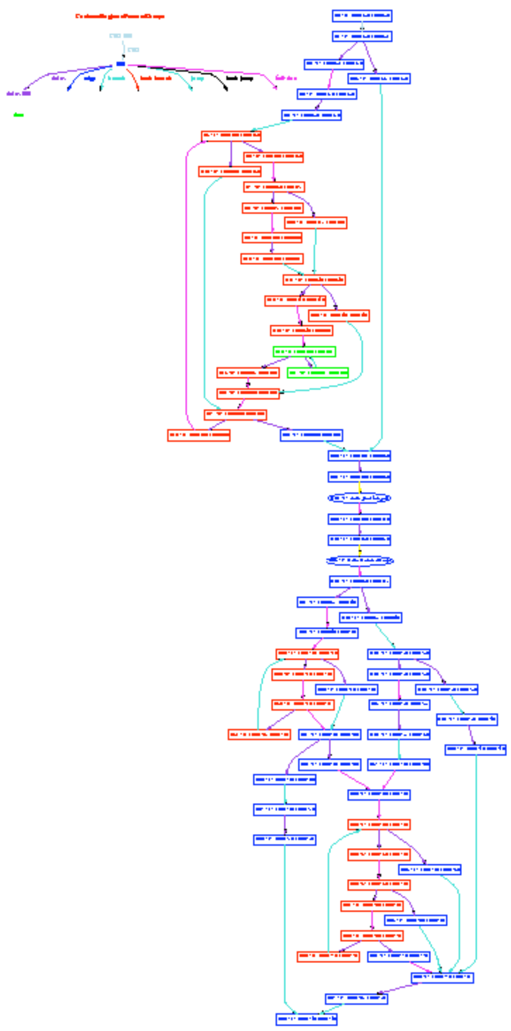


Program Structure Recovery with bloop

- Parse instructions in an executable using GNU binutils
- Analyze branches to identify basic blocks
- Construct control flow graph using branch target analysis
  —be careful with machine conventions and delay slots!
- Use interval analysis to identify natural loop nests
- Map machine instructions to source lines with symbol table
  —dependent on accurate debugging information!
- Normalize output to recover source-level view

Platforms: Alpha+Tru64, MIPS+IRIX, Linux+IA64, Linux+IA32, Solaris+SPARC

Sample Flowgraph from an Executable



Loop nesting structure
- —blue: outermost level
- —red: loop level 1
- —green loop level 2

Observation
optimization complicates program structure!

Normalizing Program Structure

Constraint: each source line must appear at most once

Coalesce duplicate lines

- (1) if duplicate lines appear in different loops
  - find least common ancestor in scope tree; merge corresponding loops along the paths to each of the duplicates

    purpose: re-rolls loops that have been split

- (2) if duplicate lines appear at multiple levels in a loop nest
  - discard all but the innermost instance

    purpose: handles loop-invariant code motion

- apply (1) and (2) repeatedly until a fixed point is reached

Recovered Program Structure

```
<LM n="/apps/smg98/test/smg98">
  ...
  <F n="/apps/smg98/struct_linear_solvers/smg_relax.c">
    <P n="hypre_SMGRelaxFreeARem">
      <L b="146" e="146">
        <S b="146" e="146"/>
      </L>
    </P>
    <P n="hypre_SMGRelax">
      <L b="297" e="328">
        <S b="297" e="297"/>
        <L b="301" e="328">
          <S b="301" e="301"/>
          <L b="318" e="325">
            <S b="318" e="325"/>
          </L>
          <S b="328" e="328"/>
        </L>
        <S b="302" e="302"/>
      </L>
    </P>
    ...
  </F>
</PGM>
```

Load Module

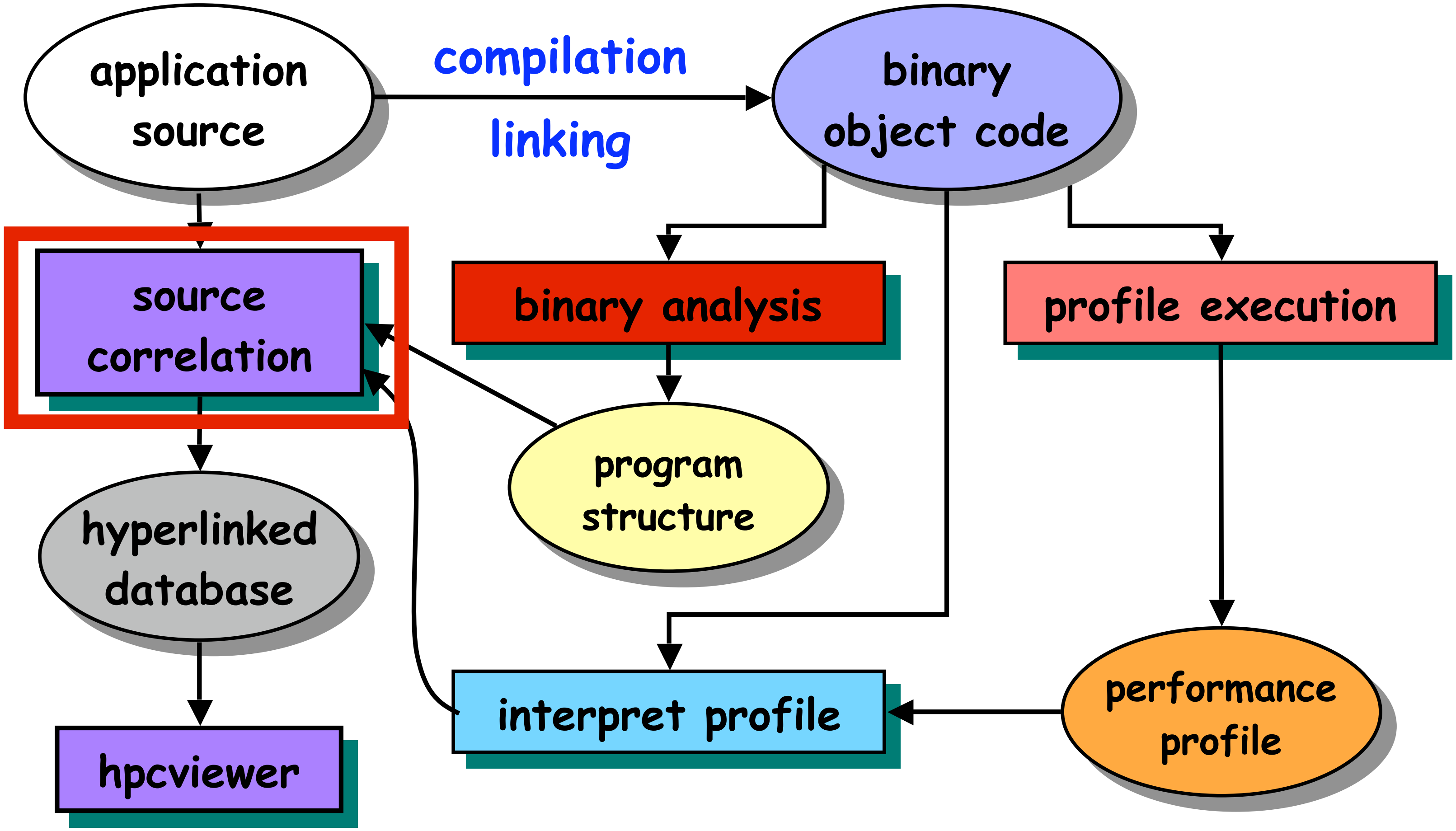
File

Procedure

Loop

Statement

HPCToolkit System Overview

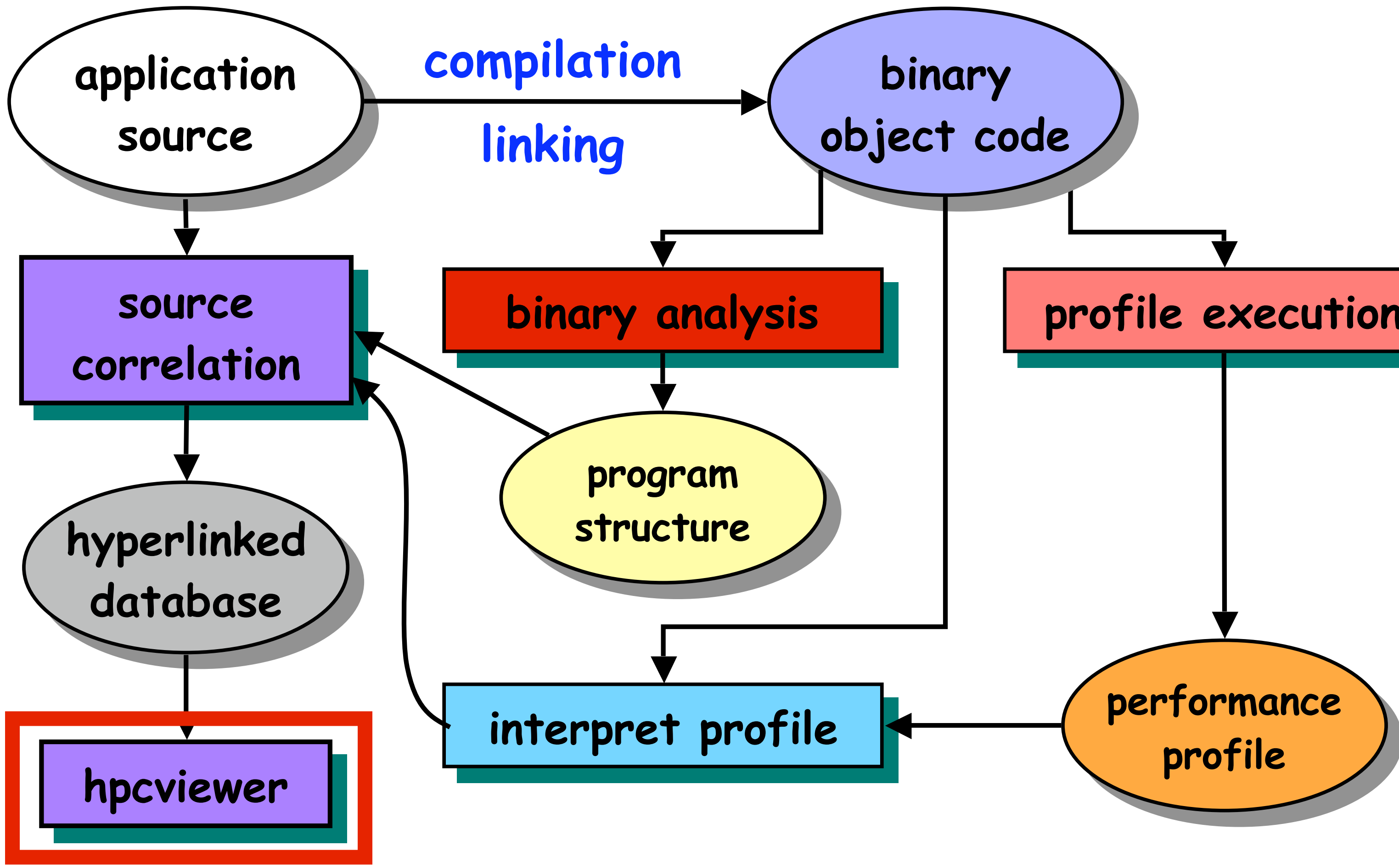


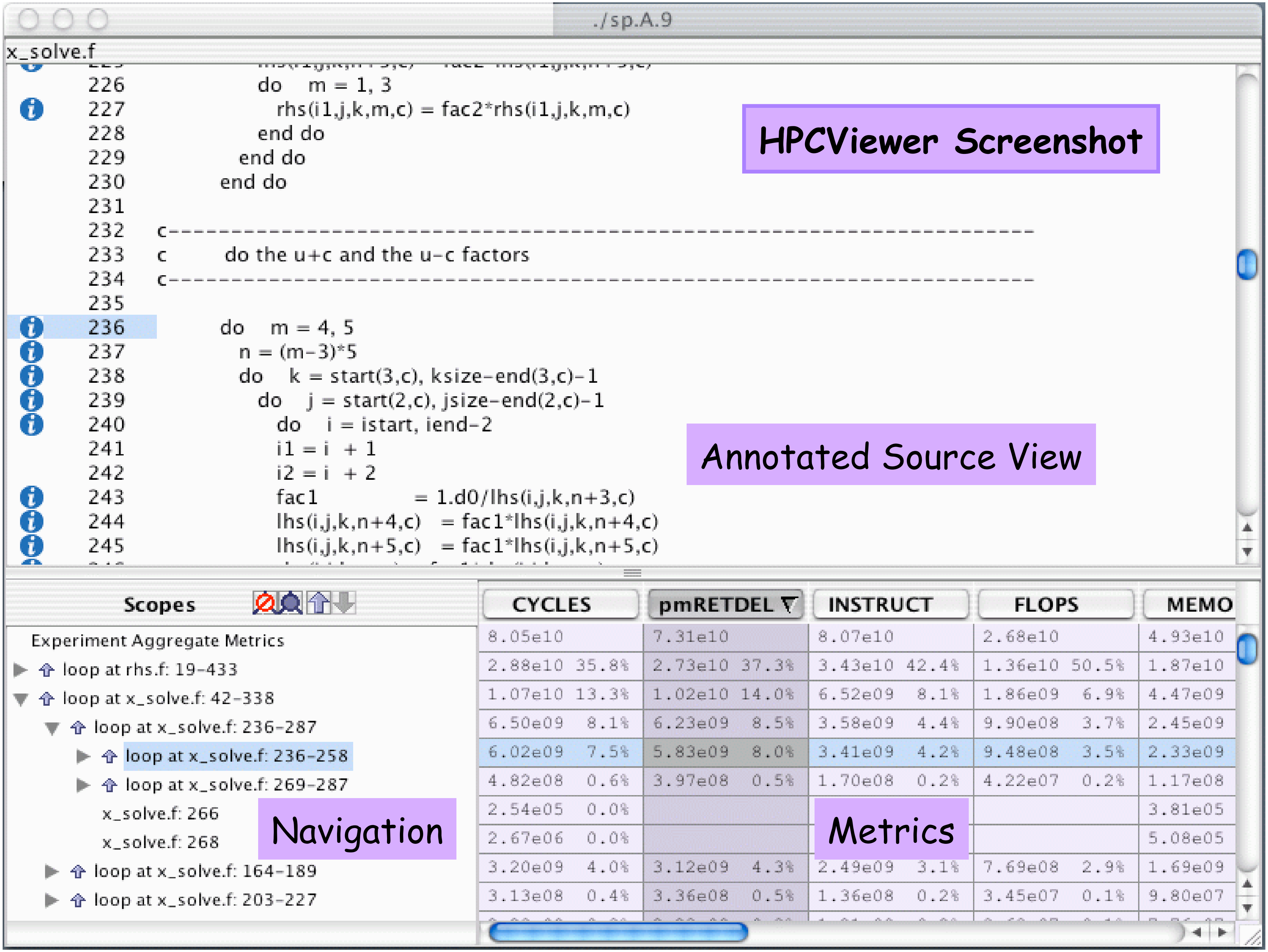
http://hipersoft.cs.rice.edu/hpctoolkit/

Data Correlation

- **Problem**
  - —any one performance measure provides a myopic view
    - – some measure potential *causes* (e.g. cache misses)
    - – some measure *effects* (e.g. cycles)
    - – cache misses not always a problem
  - —event counter attribution is inaccurate for out-of-order processors
- **Approaches**
  - — multiple metrics for each program line
  - —computed metrics, e.g. cycles - FLOPS
    - – eliminate mental arithmetic
    - – serve as a key for sorting
  - —hierarchical structure
    - – line level attribution errors give good loop-level information

HPCToolkit System Overview

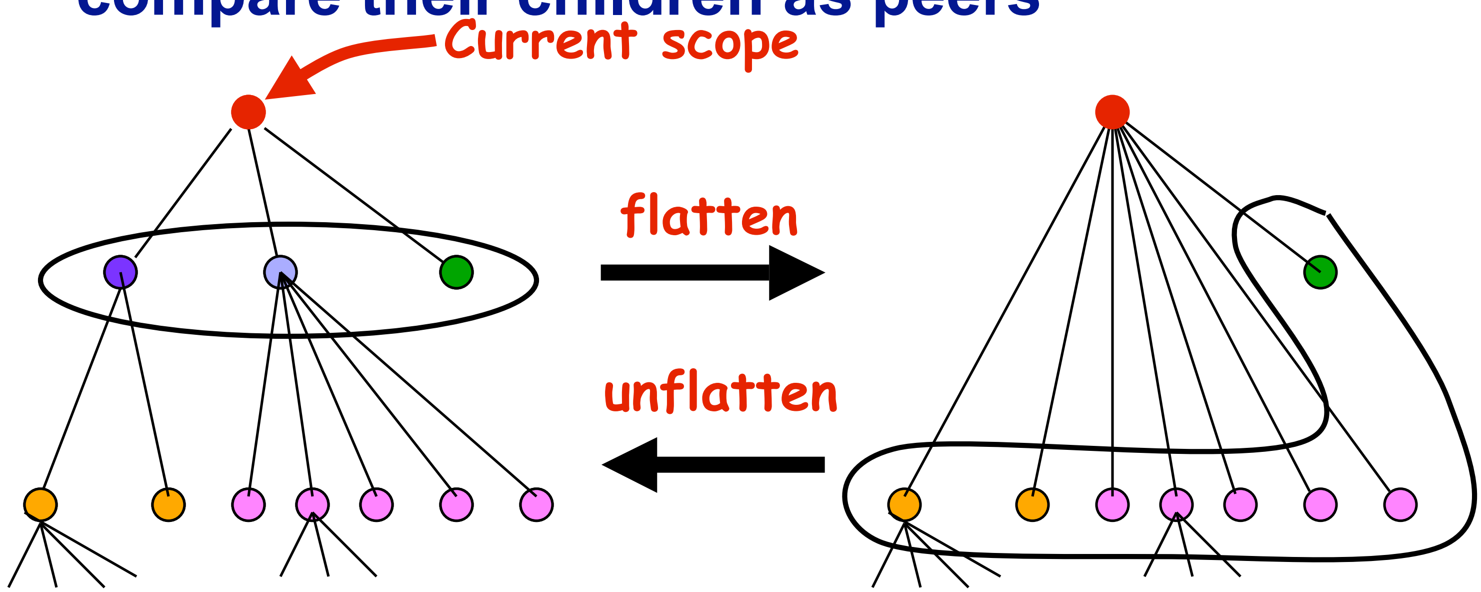




HPCViewer Screenshot
./sp.A.9
x_solve.f
226 do m = 1, 3
227 rhs(i1,j,k,m,c) = fac2*rhs(i1,j,k,m,c)
228 end do
229 end do
230 end do
231
232 c---
233 c do the u+c and the u-c factors
234 c---
235
236 do m = 4, 5
237 n = (m-3)*5
238 do k = start(3,c), ksize-end(3,c)-1
239 do j = start(2,c), jsize-end(2,c)-1
240 do i = istart, iend-2
241 i1 = i + 1
242 i2 = i + 2
243 fac1 = 1.d0/lhs(i,j,k,n+3,c)
244 lhs(i,j,k,n+4,c) = fac1*lhs(i,j,k,n+4,c)
245 lhs(i,j,k,n+5,c) = fac1*lhs(i,j,k,n+5,c)
Annotated Source View
Scopes
CYCLES
pmRETDEL
INSTRUCT
FLOPS
MEMO
Experiment Aggregate Metrics 8.05e10 7.31e10 8.07e10 2.68e10 4.93e10
loop at rhs.f: 19-433 2.88e10 35.8% 2.73e10 37.3% 3.43e10 42.4% 1.36e10 50.5% 1.87e10
loop at x_solve.f: 42-338 1.07e10 13.3% 1.02e10 14.0% 6.52e09 8.1% 1.86e09 6.9% 4.47e09
loop at x_solve.f: 236-287 6.50e09 8.1% 6.23e09 8.5% 3.58e09 4.4% 9.90e08 3.7% 2.45e09
loop at x_solve.f: 236-258 6.02e09 7.5% 5.83e09 8.0% 3.41e09 4.2% 9.48e08 3.5% 2.33e09
loop at x_solve.f: 269-287 4.82e08 0.6% 3.97e08 0.5% 1.70e08 0.2% 4.22e07 0.2% 1.17e08
x_solve.f: 266 2.54e05 0.0% 3.81e05
x_solve.f: 268 2.67e06 0.0% 5.08e05
loop at x_solve.f: 164-189 3.20e09 4.0% 3.12e09 4.3% 2.49e09 3.1% 7.69e08 2.9% 1.69e09
loop at x_solve.f: 203-227 3.13e08 0.4% 3.36e08 0.5% 1.36e08 0.2% 3.45e07 0.1% 9.80e07
Navigation
Metrics

Flattening for Top Down Analysis

- **Problem**
  - —strict hierarchical view of a program is too rigid
  - —want to compare program components at the same level as peers
- **Solution**
  - —enable a scope's descendants to be flattened to compare their children as peers



http://hipersoft.cs.rice.edu/hpctoolkit/

Some Uses for HPCToolkit

- **Identifying unproductive work**
  - —where is the program spending its time not performing FLOPS
- **Memory hierarchy issues**
  - —bandwidth utilization: misses x line size/cycles
  - —exposed latency: ideal vs. measured
- **Cross architecture or compiler comparisons**
  - —what program features cause performance differences?
- **Gap between peak and observed performance**
  - —loop balance vs. machine balance?
- **Evaluating load balance in a parallelized code**
  - —how do profiles for different processes compare

Assessment of HPCToolkit Functionality

- Top down analysis focuses attention where it belongs
  - —sorted views put the important things first
- Integrated browsing interface facilitates exploration
  - —rich network of connections makes navigation simple
- Hierarchical, loop-level reporting facilitates analysis
  - —more sensible view when statement-level data is imprecise
- Binary analysis handles multi-lingual applications and libraries
  - —succeeds where language and compiler based tools can't
- Sample-based profiling, aggregation and derived metrics
  - —reduce manual effort in analysis and tuning cycle
- Multiple metrics provide a better picture of performance
- Multi-platform data collection
- Platform independent analysis tool

What's Next?

Research

- collect and present dynamic content
  - what path gets us to expensive computations?
  - accurate call-graph profiling of unmodified executables
  - analysis and presentation of dynamic content
- communication in parallel programs
- statistical clustering for analyzing large-scale parallelism
- performance diagnosis: why rather than what

Development

- harden toolchain
- new platforms: Opteron and PowerPC
- data collection with oprofile on Linux