
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

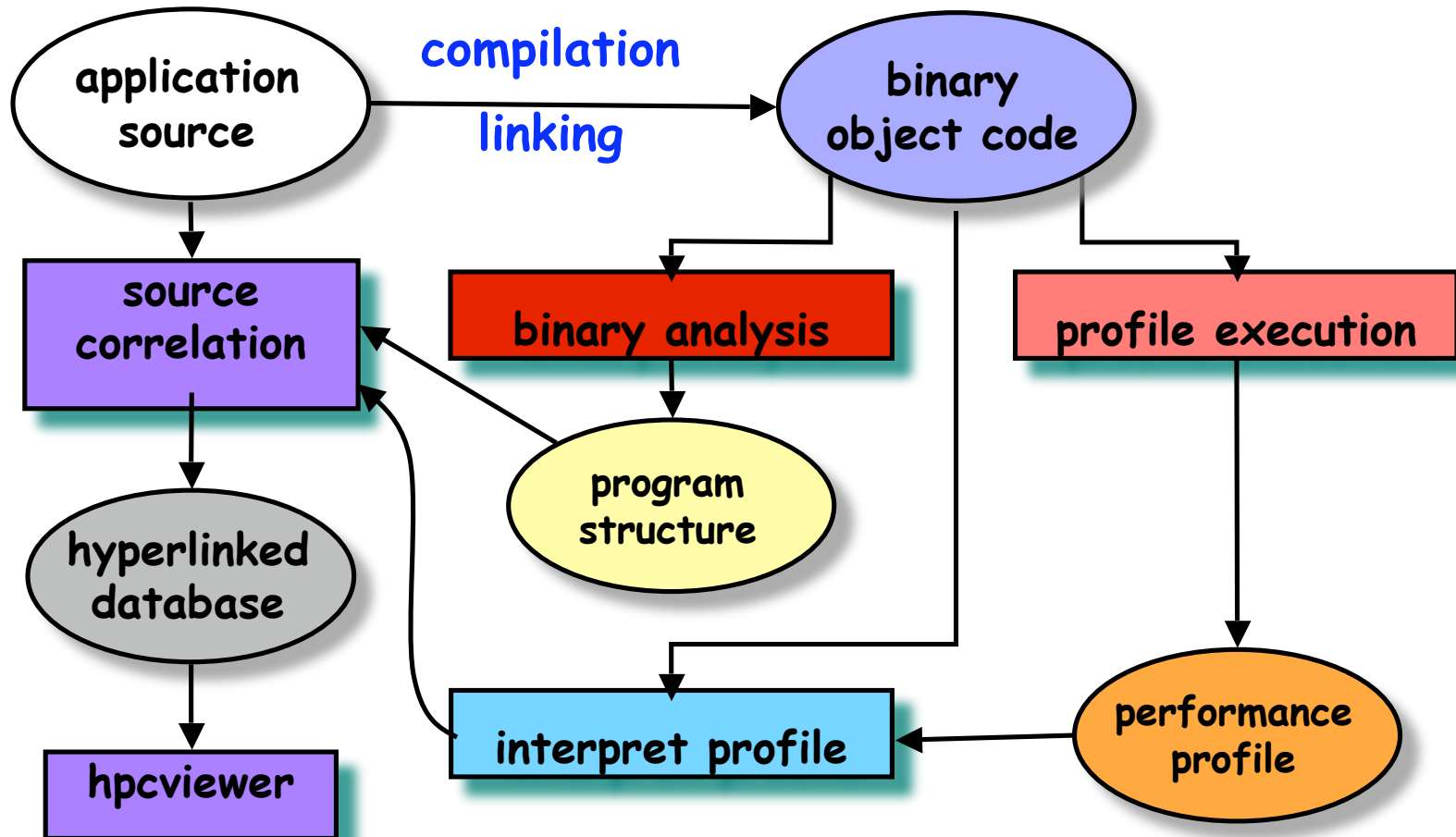
Rice's HPCToolkit Performance Tool Goals

- **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

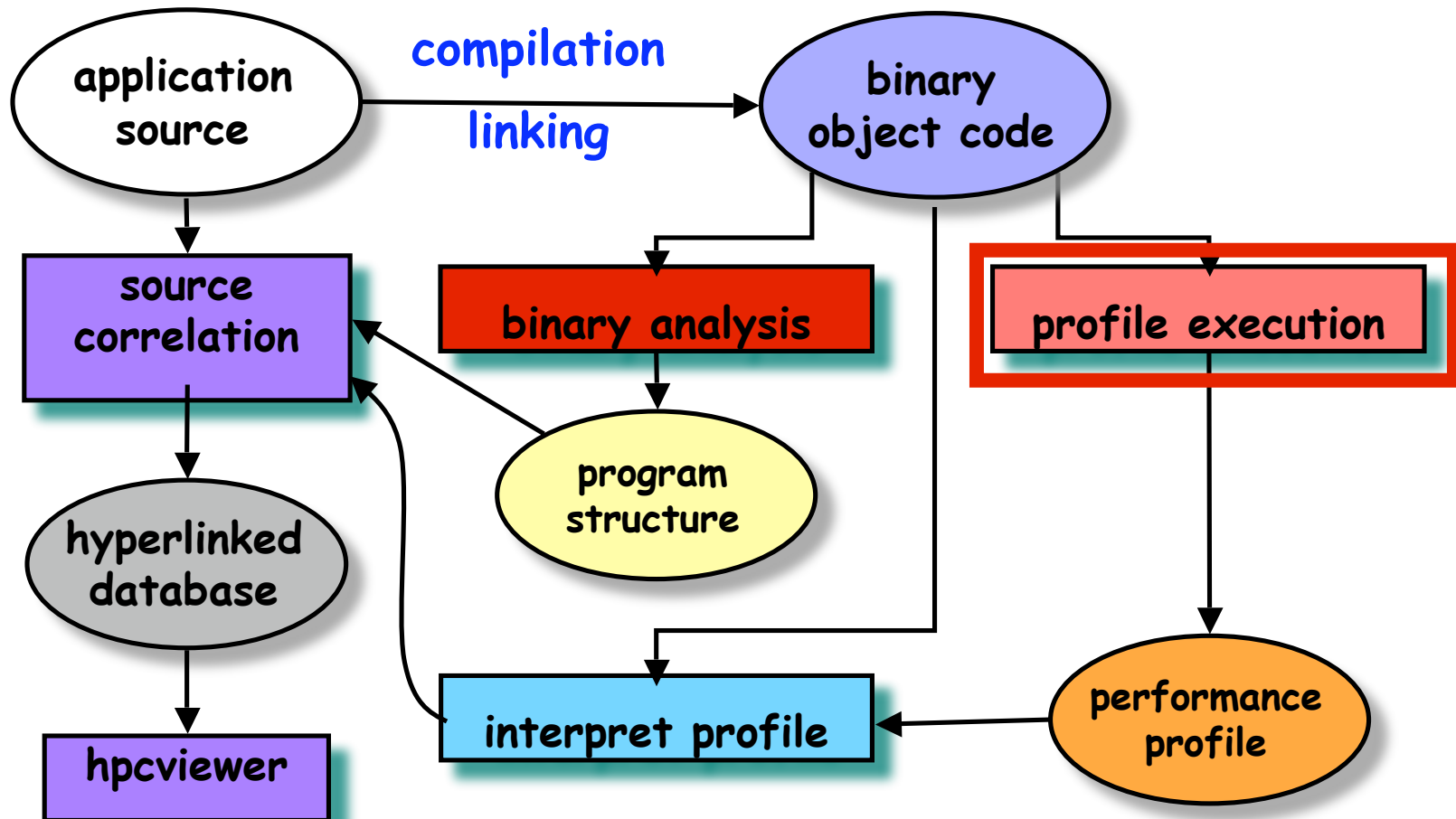
HPCToolkit Design Principles

- **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

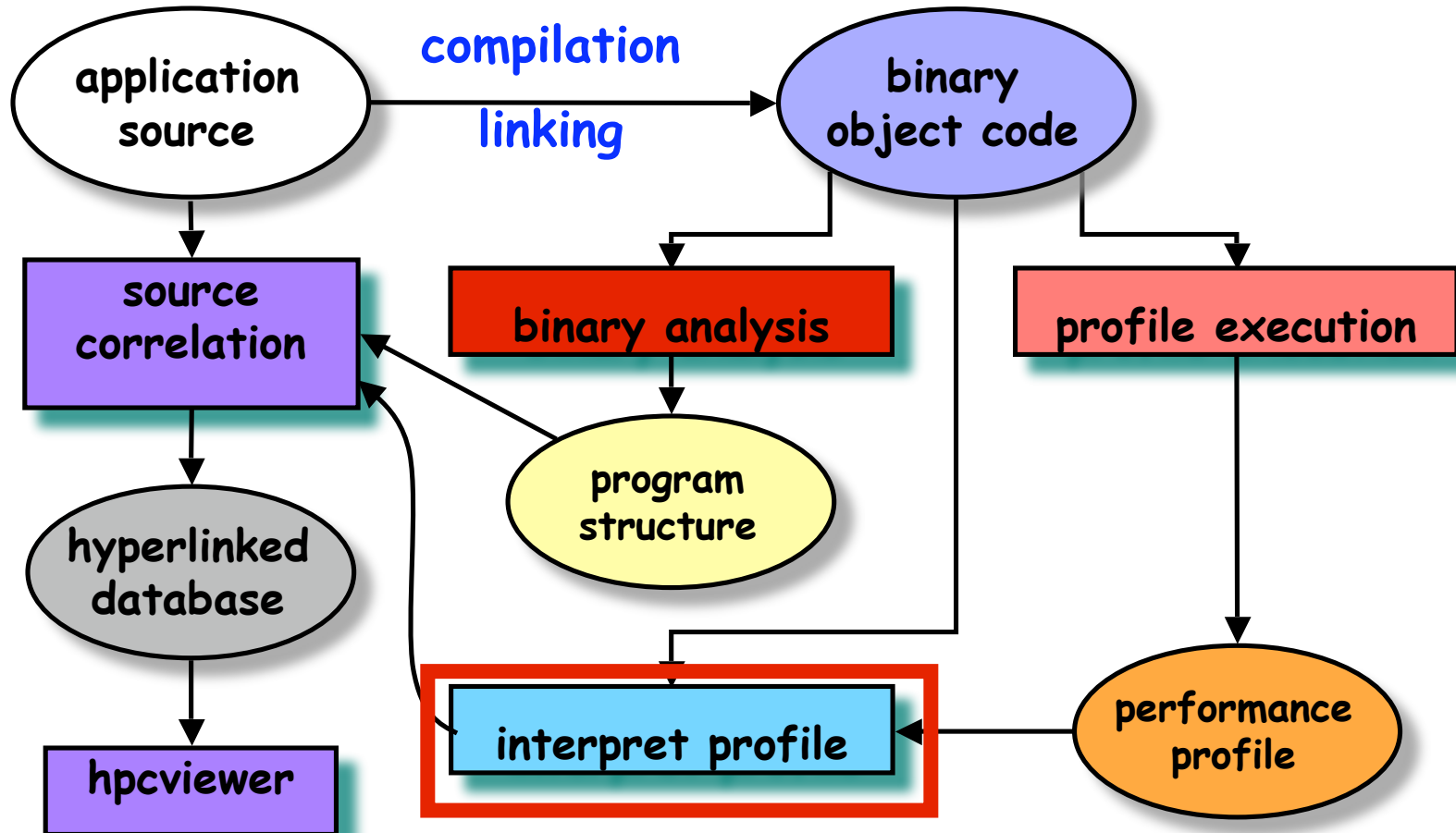


HPCToolkit Workflow



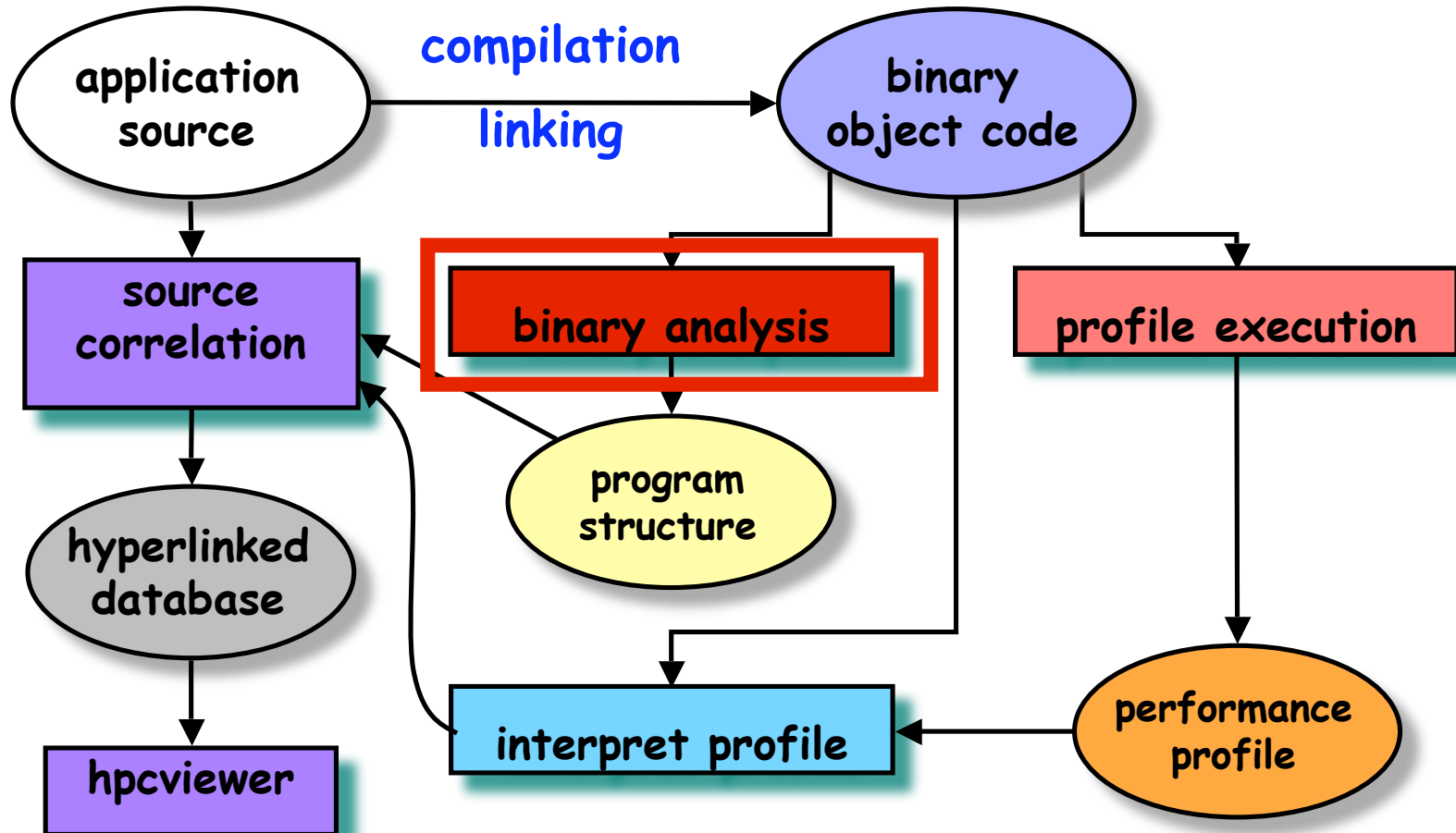
- launch unmodified, optimized application binaries
- collect statistical profiles of events of interest

HPCToolkit Workflow



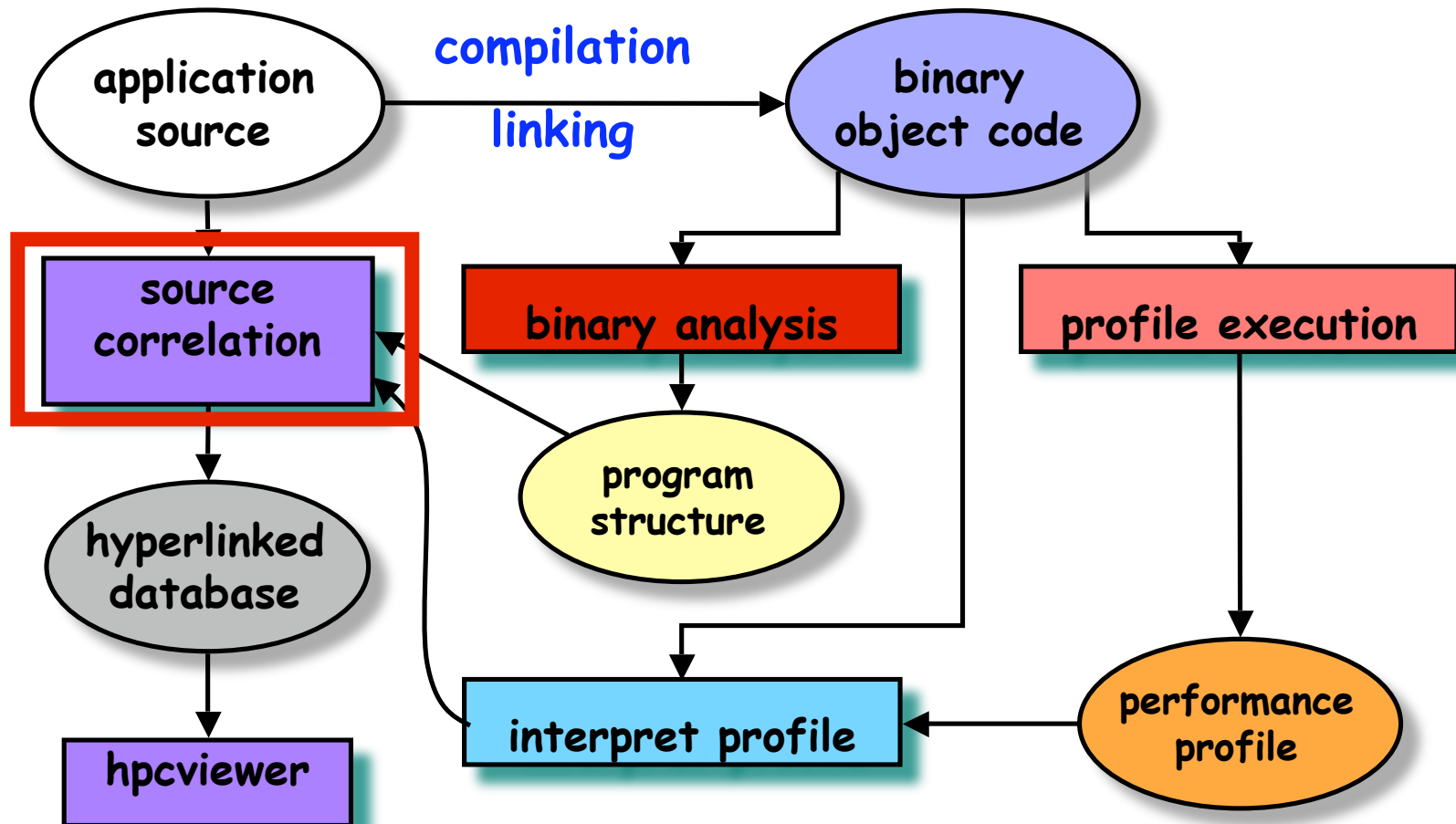
—decode instructions and combine with profile data

HPCToolkit Workflow



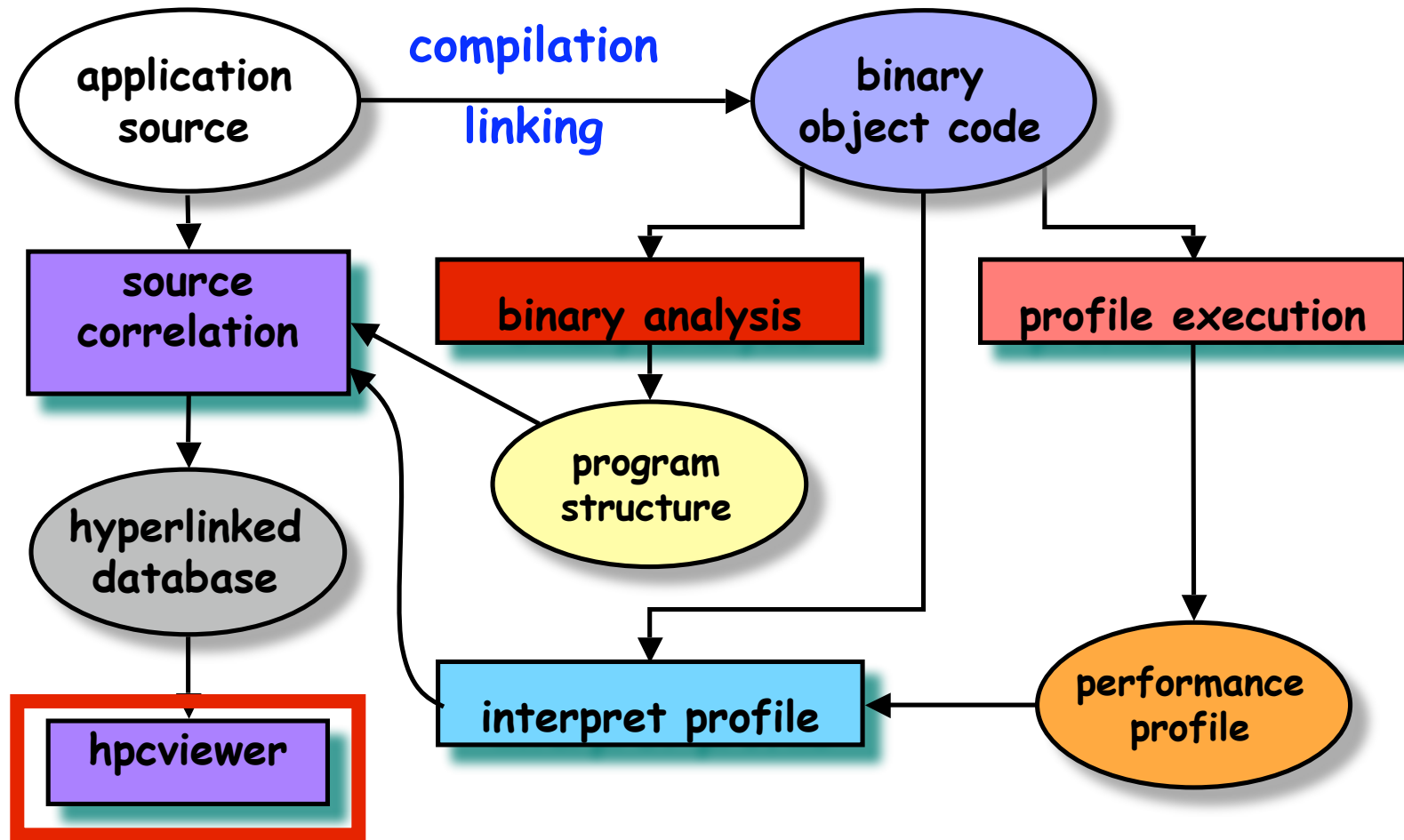
—extract loop nesting & inlining from executables

HPCToolkit Workflow



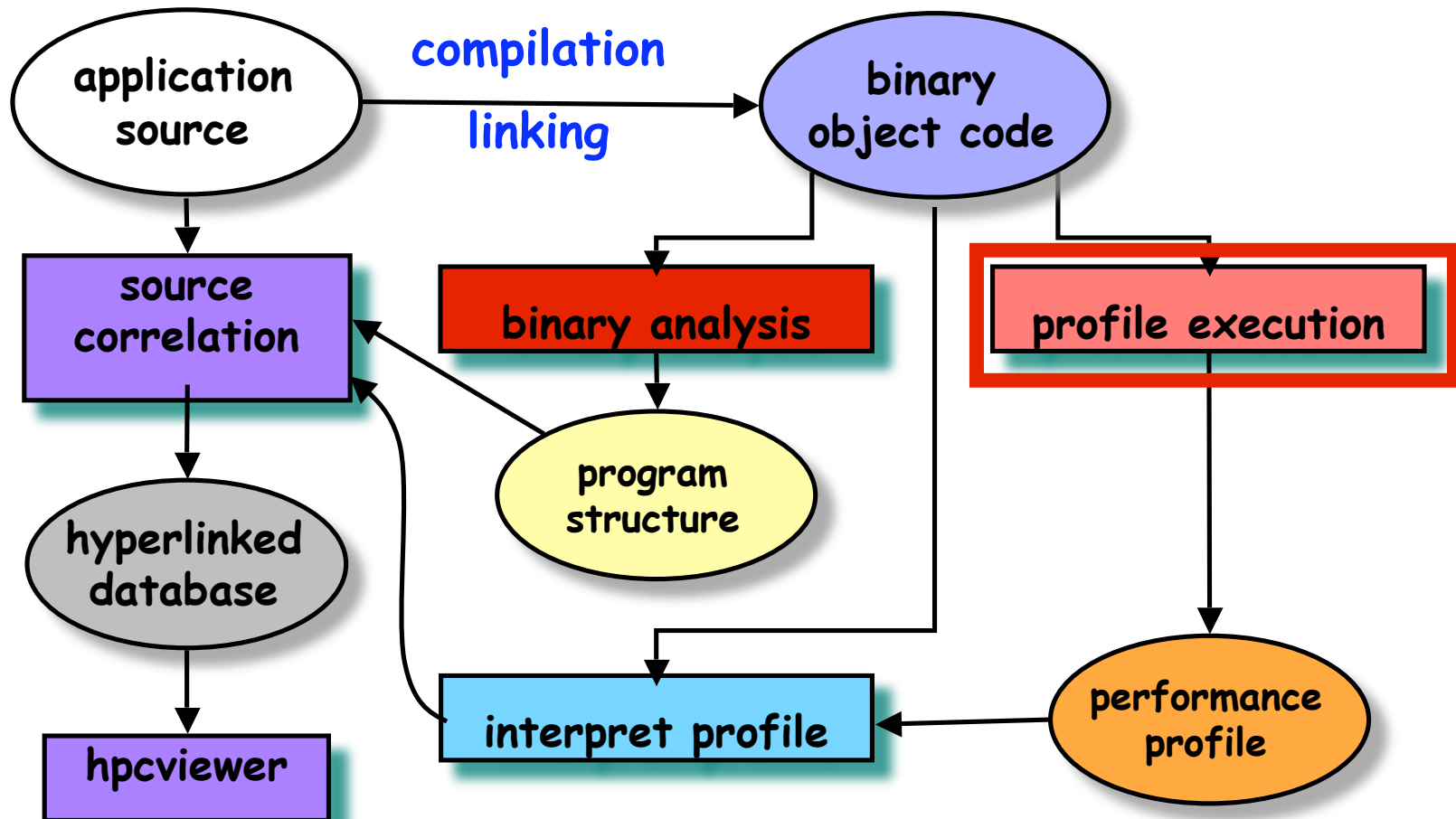
- synthesize new metrics by combining metrics
- relate metrics and structure to program source

HPCToolkit Workflow



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

HPCToolkit Workflow



Measurement Challenges

Performance often depends upon context

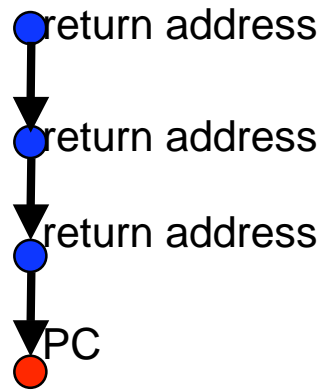
- **Layered design**
 - application frameworks, math libraries, communication libraries
- **Generic programming, e.g. C++ templates**
 - both data structures and algorithms
- **Context-sensitive optimization**
 - e.g. inlining
- **Goals**
 - identify and quantify context-sensitive behavior
 - differentiate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

Call Path Profiling

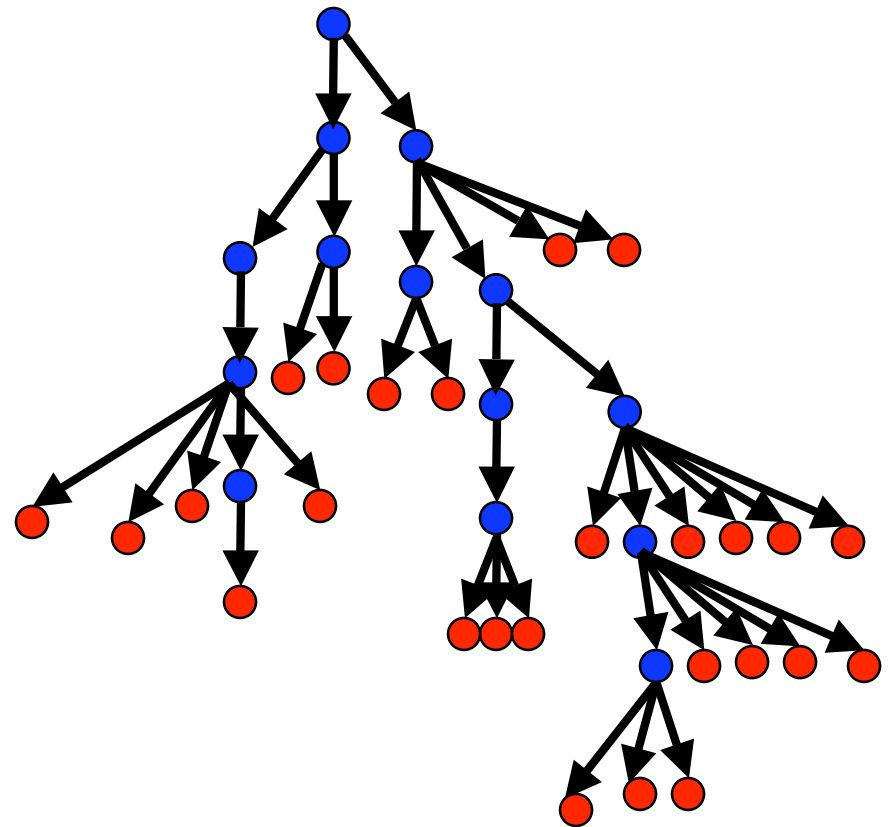
- **No instrumentation**
 - statistical sampling of hardware performance counter overflows
 - gather calling context information using stack unwinding
 - overhead proportional to sampling frequency
 - not calling frequency
- **Capture samples in full calling context**
 - attribute sample to individual 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

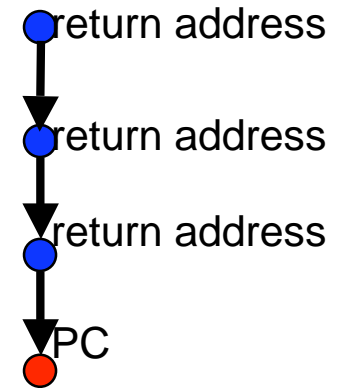


Calling Context Tree (CCT)



Unwinding Optimized Code

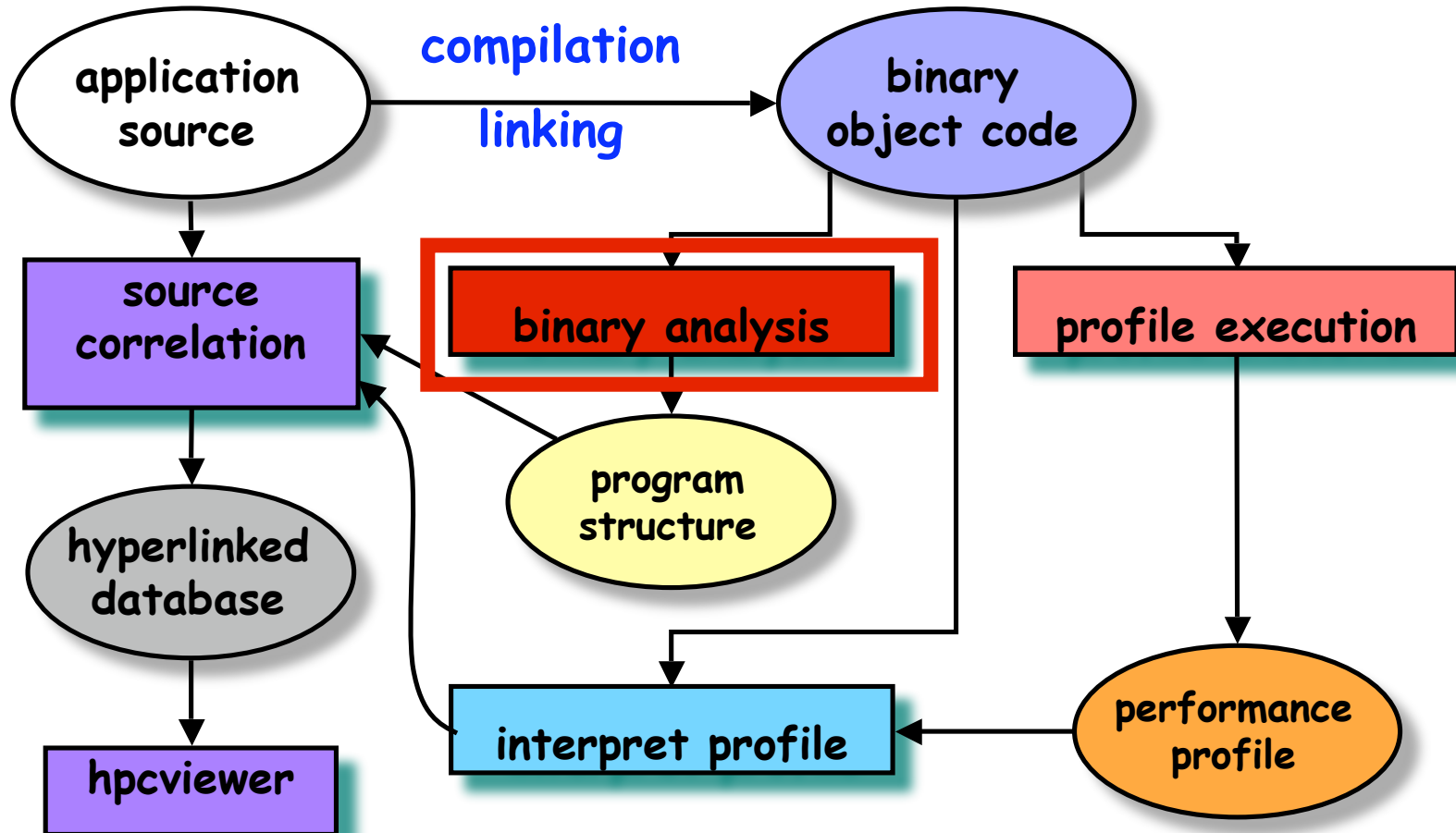
- **Information needed for unwinding**
 - where is the return address of the current frame?
 - where is the BP for the caller's frame?
 - what is the caller's stack pointer value?
- **Unwinding optimized code is challenging**
 - often lacks frame pointers
 - little or no compiler information available
 - 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

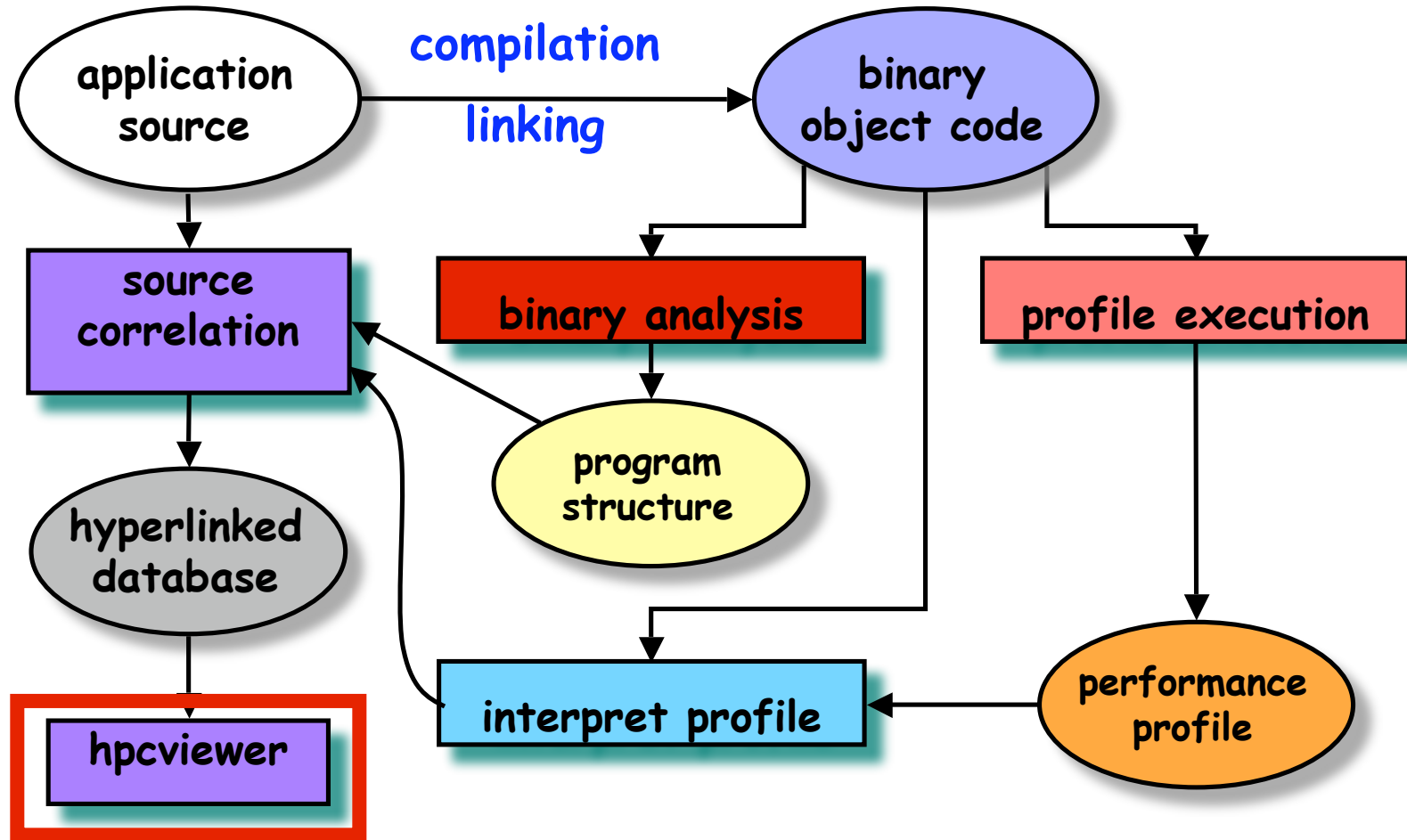


Program Structure Recovery

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

The screenshot displays the hpcviewer application window titled 'hmc'. The main area is the source code editor for 'hmc.cc', showing C++ code for a Hybrid Monte Carlo program. A red box labeled 'source pane' highlights the code area. Below the code editor is a 'view control' bar with three buttons: 'Calling Context View', 'Callers View', and 'Flat View'. A red box labeled 'view control' points to this bar. Below the view control is the 'navigation pane' (Scopes) showing a tree view of the execution context, including 'main', 'void Chroma::doHMC', and various loops and function calls. A red box labeled 'navigation pane' points to this area. To the right of the navigation pane is the 'metric pane', which displays a table of performance metrics. A red box labeled 'metric pane' points to this table.

# samples (I)	# samples (E)
5.80e05	97.5%
5.70e05	95.8%
5.65e05	95.0%
5.65e05	95.0%
5.25e05	88.2%
4.25e05	71.4%
4.25e05	71.4%
3.30e05	55.5%

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

calling context view

```
947 template<class T>
948 inline typename UnaryReturn<OLattice<T>, FnNorm2>::Type_t
949 norm2(const multi1d< OLattice<T> >& s1, const OrderedSubset& s)
950 {
951     typename UnaryReturn<OLattice<T>, FnNorm2>::Type_t d;
952
953     #if defined(QDP_USE_PROFILING)
954     static QDPProfile_t prof(d, OpAssign(), FnNorm2(), s1[0]);
955     prof.time -= getClockTime();
956     #endif
957
958     // Possibly loop entered
959     zero_rep(d.elem());
960
961     for(int n=0; n < s1.size(); ++n)
962     {
963         const OLattice<T>& ss1 = s1[n];
964         for(int i=s.start(); i <= s.end(); ++i)
965             d.elem() += localNorm2(ss1.elem(i));
966     }

```

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

Calling Context View Callers View Flat View

Scopes	# samples (I)	# samples (E)
void Chroma::doHMC<Chroma::HMCTryParams>(QDP::multi1d<QDP::OLattice<T>>& s1, const OrderedSubset& s)	5.70e05 95.81	
loop at hmc.cc: 311-435	5.65e05 95.01	
Chroma::AbsHMCTry<QDP::multi1d<QDP::OLattice<QDP::PScalar<Q>>& s1, const OrderedSubset& s>::operator()(QDP::multi1d<QDP::OLattice<T>>& s1, const OrderedSubset& s)	5.65e05 95.01	
Chroma::LatColMatSTSLeapfrogRecursiveIntegrator::operator()(Chroma::AbsHMCTry<QDP::multi1d<QDP::OLattice<QDP::PScalar<Q>>& s1, const OrderedSubset& s>& hmc, const QDP::multi1d<QDP::OLattice<T>>& s1, const OrderedSubset& s)	5.25e05 88.21	
loop at lcm_sts_leapfrog_recursive.cc: 129-131	4.85e05 81.51	
Chroma::LatColMatExpSdtIntegrator::operator()(Chroma::AbsHMCTry<QDP::multi1d<QDP::OLattice<QDP::PScalar<Q>>& s1, const OrderedSubset& s>& hmc, const QDP::multi1d<QDP::OLattice<T>>& s1, const OrderedSubset& s)	4.25e05 71.41	
loop at lcm_exp_sdt.cc: 85-88	4.25e05 71.41	
Chroma::LCMMDIntegratorSteps::leapP(QDP::multi1d<Chroma::AbsHMCTry<QDP::multi1d<QDP::OLattice<QDP::PScalar<Q>>& s1, const OrderedSubset& s>& hmc, const QDP::multi1d<QDP::OLattice<T>>& s1, const OrderedSubset& s)	4.25e05 71.41	
Chroma::TwoFlavorExactWilsonTypeFermMonomial<QDP::OLattice<T>>::operator()(Chroma::AbsHMCTry<QDP::multi1d<QDP::OLattice<QDP::PScalar<Q>>& s1, const OrderedSubset& s>& hmc, const QDP::multi1d<QDP::OLattice<T>>& s1, const OrderedSubset& s)	3.30e05 55.51	
loop at lcm_integrator_leaps.cc: 42-47	8.00e04 13.41	
Chroma::GaugeMonomial::dsdq(QDP::multi1d<QDP::OLattice<T>>& s1, const OrderedSubset& s)	8.00e04 13.41	5.00e03 0.81
Chroma::PlaqGaugeAct::deriv(QDP::multi1d<QDP::OLattice<T>>& s1, const OrderedSubset& s)	7.50e04 12.61	1.00e04 1.71

Chroma Lattice QCD Library

caller's view

```
sse_su3dslash_w.c
686  /* the basic operations in this routine include loading the halfspinor
687  * from memory, multiplying it by the appropriate gauge field, doing the
688  * spin reconstruction, and summing over directions, and saving the partial
689  * sum over directions */
690
691 void mvv_recons_plus(size_t lo,size_t hi, int id, const void *ptr)
692 {
693     DECL_COMMON_ALIASES_TEMPS;
694
695     const Arg_s *a =(Arg_s *)ptr;
696     int low = (int)lo;
697     int high = (int)hi;
698
699     MY_SPINOR* spinor_field = a->spinfun;
700
701     MY_SSE_VECTOR* chia = a->chifun; /* a 1-d map of a 2-d array */
```

show attribution of procedure costs to calling contexts

Calling Context View Callers View Flat View

Scopes

mvv_recons_plus	
# samples (I)	# samples (E)
5.50e04 9.2%	5.50e04 9.2%
5.50e04 9.2%	5.50e04 9.2%
5.50e04 9.2%	5.50e04 9.2%
3.50e04 5.9%	3.50e04 5.9%
3.00e04 5.0%	3.00e04 5.0%
5.00e03 0.8%	5.00e03 0.8%
2.00e04 3.4%	2.00e04 3.4%
1.50e04 2.5%	1.50e04 2.5%
5.00e03 0.8%	5.00e03 0.8%
5.00e04 8.4%	5.00e04 8.4%

S3D Solver for Turbulent, Reacting Flows

flat view

```
mixavg_transport_m.f90
734 diffFlux(:,:,:,n_spec,:) = 0.0
735 DIRECTION: do m=1,3
736 SPECIES: do n=1,n_spec-1
737
738 if (baro_switch) then
739 ! driving force includes gradient in mole fraction and baro-diffusion:
740 diffFlux(:,:,:,n,m) = - Ds_mixavg(:,:,:,n) * ( grad_Ys(:,:,:,n,m) &
741 + Ys(:,:,:,n) * ( grad_mixMW(:,:,:,m) &
742 + (1 - molwt(n)*avmolwt) * grad_P(:,:,:,m)/Press))
743 else
744 ! driving force is just the gradient in mole fraction:
745 diffFlux(:,:,:,n,m) = - Ds_mixavg(:,:,:,n) * ( grad_Ys(:,:,:,n,m) &
746 + Ys(:,:,:,n) * grad_mixMW(:,:,:,m) )
747 endif
748
749 ! Add thermal diffusion:
750 if (thermDiff_switch) then
751 diffFlux(:,:,:,n,m) = diffFlux(:,:,:,n,m) &
752 - Ds_mixavg(:,:,:,n) * Rs_therm_diff(:,:,:,n) * molwt(n) &
753 * avmolwt * grad_T(:,:,:,m) / Temp
754 endif
755
```

attribution of costs to loops
implicit with F90 vector syntax

fine grain attribution to loops
within a loop nest

Calling Context View Callers View Flat View

Scopes

- loop at mixavg_transport_m.f90: 735-760
 - loop at mixavg_transport_m.f90: 736-758
 - loop at mixavg_transport_m.f90: 745
 - loop at mixavg_transport_m.f90: 758
 - loop at mixavg_transport_m.f90: 740
 - loop at mixavg_transport_m.f90: 751

# samples (I)		# samples (E)	
2.17e07	11.3%	2.17e07	11.3
2.17e07	11.3%	2.17e07	11.3
1.54e07	8.0%	1.54e07	8.0
6.32e06	3.3%	6.32e06	3.3

S3D Solver for Turbulent, Reacting Flows

flat view

```
rhsf.f90
199 ! grad_Y - Species mass fraction gradients may be required in transport
200 !     evaluation as well as for boundary conditions.
201 !
202 !notes by ramanan - 01/05/05
203 !The array dimensioning can be misleading
204 !For grad_u, 4th dimension is the direction and 5th dimension is the velocity component
205 !For grad_Ys, 4th dimension is the species and 5th dimension is the direction
206
207 call computeVectorGradient( u, grad_u )
208 call computeScalarGradient( temp, grad_T )
209 do n=1,n_spec
210   call computeScalarGradient( yspecies(:,:,:,n), grad_Ys(:,:,:,n) )
211 enddo
212
213 !Added by Ramanan - 01/05/05
214 !Store the boundary grad values
215 if(vary_in_x==1)then
216   if (xid==0) then
217     grad_u_x0 = grad_u(1,:,:,1,:)
218     grad_Ys_x0 = grad_Ys(1,:,:,1)
219     h_spec_x0 = h_spec(1,:,:,)
220   end if
```

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

Calling Context View Callers View Flat View

Scopes

- Experiment Aggregate Metrics
 - ~~~s3d_f90.x: <unknown-file>~~~: 0
 - loop at mixavg_transport_m.f90: 735-760
 - loop at rhsf.f90: 209-210
 - loop at rhsf.f90: 210
 - loop at mixavg_transport_m.f90: 1004-1011

# samples (I)	# samples (E)
1.91e08 100.0	1.91e08 100.0
2.60e07 13.6%	2.60e07 13.6%
2.17e07 11.3%	2.17e07 11.3%
2.03e07 10.6%	1.04e07 5.4%
2.03e07 10.6%	1.04e07 5.4%
8.94e06 4.7%	8.94e06 4.7%

S3D Solver for Turbulent, Reacting Flows

```

mixavg_transport_m.f90
740  SPECIES: do n=1,n_spec-1
741
742    if (baro_switch) then
743      ! driving force includes gradient in mole fraction and baro-diffusion:
744      diffFlux(i,j,k,n,m) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,m) &
745                    + Ys(i,j,k,n) * ( grad_mixMW(i,j,k,m) &
746                    + (1 - molwt(n)*avmolwt) * grad_P(i,j,k,m)/Press))
747    else
748      ! driving force is just the gradient in mole fraction:
749      diffFlux(i,j,k,n,m) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,m) &
750                    + Ys(i,j,k,n) * grad_mixMW(i,j,k,m) )
751    endif
752
753    ! Add thermal diffusion:
754    if (thermDiff_switch) then
755      diffFlux(i,j,k,n,m) = diffFlux(i,j,k,n,m) &
756                    - Ds_mixavg(i,j,k,n) * Rs_therm_diff(i,j,k,n) * molwt(n) &
757                    + avmolwt * grad_T(i,j,k,m) / Temp
            
```

flat view

waste metric
peak FLOPs -
actual FLOPS

highlights memory
hierarchy problems here

Flat View

Scopes	Cycles	Instructions	FP Instructions	LI Accesses	Waste	Relative Waste
Experiment Aggregate Metrics	5.82e11 100.0	3.92e11 100.0	1.79e11 100.0	2.03e11 100.0	1.57e12 100.0	0.90e00
loop at mixavg_transport_m.f90: 739-764	5.98e10 10.3%	3.34e10 8.5%	7.30e09 4.1%	1.83e10 9.0%	1.72e11 11.0%	0.96e00
loop at mixavg_transport_m.f90: 740-762	5.98e10 10.3%	3.34e10 8.5%	7.30e09 4.1%	1.83e10 9.0%	1.72e11 11.0%	0.96e00
loop at mixavg_transport_m.f90: 749	4.04e09 0.69%	2.08e09 5.2%	4.07e09 2.26%	1.00e10 4.9%	1.27e11 7.9%	0.98e00
loop at mixavg_transport_m.f90: 762	1.94e10 3.3%	1.66e10 4.2%	2.70e09 1.5%	8.28e09 4.1%	5.54e10 3.5%	0.95e00
mixavg_transport_m.f90: 747	7.00e06 0.0%				2.10e07 0.0%	1.00e00
mixavg_transport_m.f90: 751	3.00e06 0.0%				9.00e06 0.0%	1.00e00
mixavg_transport_m.f90: 740	1.00e06 0.0%				3.00e06 0.0%	1.00e00
mixavg_transport_m.f90: 758	1.00e06 0.0%	1.00e06 0.0%			3.00e06 0.0%	1.00e00
_fmth_i_dexp	7.47e10 12.8%	8.02e10 20.4%	5.86e10 32.8%	3.70e10 18.3%	1.66e11 10.6%	0.74e00
loop at rhs.f90: 209-210	3.05e10 5.2%	1.33e10 3.4%		6.81e09 3.4%	9.16e10 5.8%	1.00e00
loop at mixavg_transport_m.f90: 1025-1029	2.70e10 4.6%	1.25e10 3.2%	7.01e09 3.9%	4.66e09 2.3%	7.41e10 4.7%	0.91e00

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