Evolving HPCToolkit

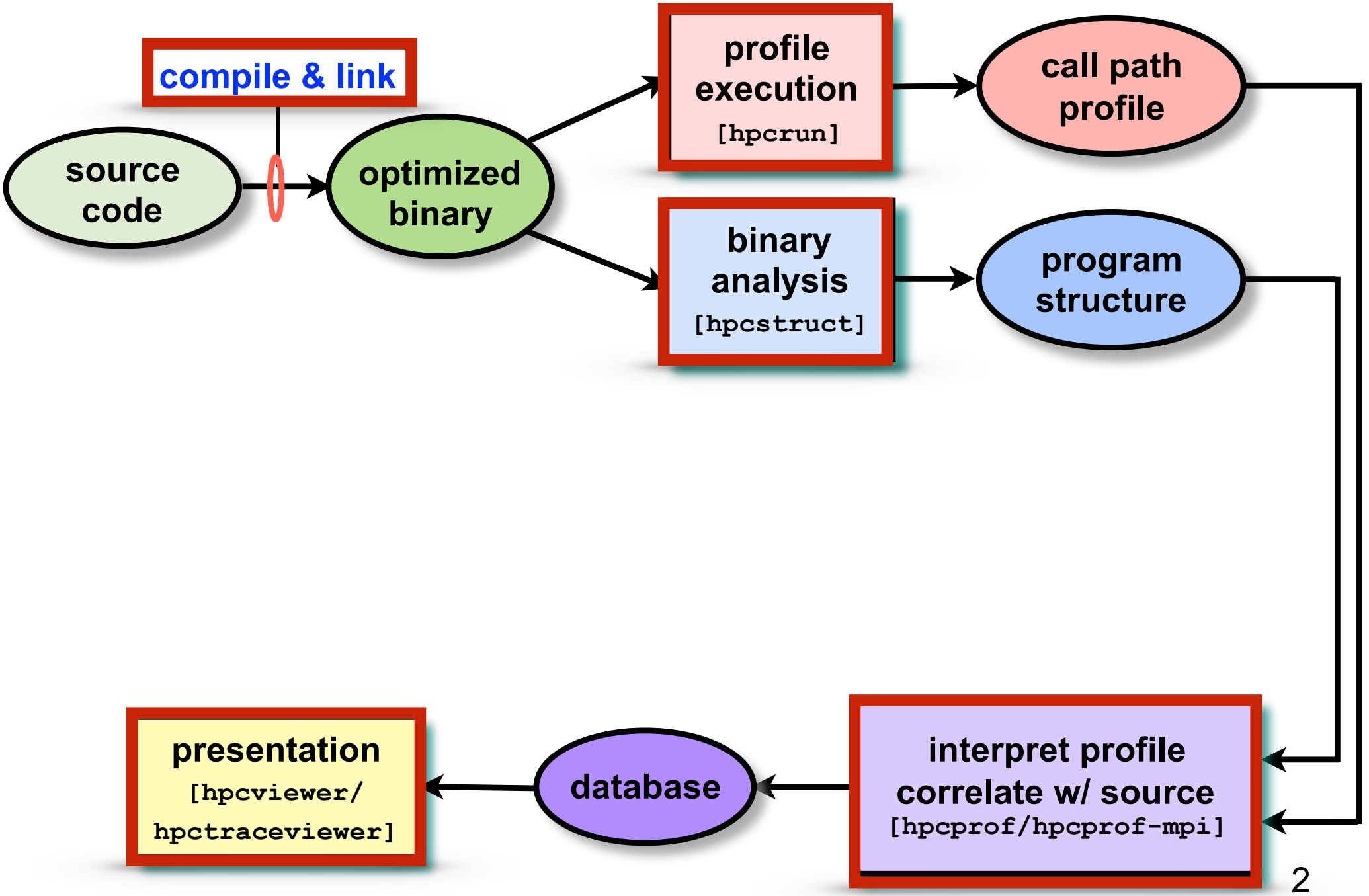
John Mellor-Crummey
Department of Computer Science
Rice University

RICE

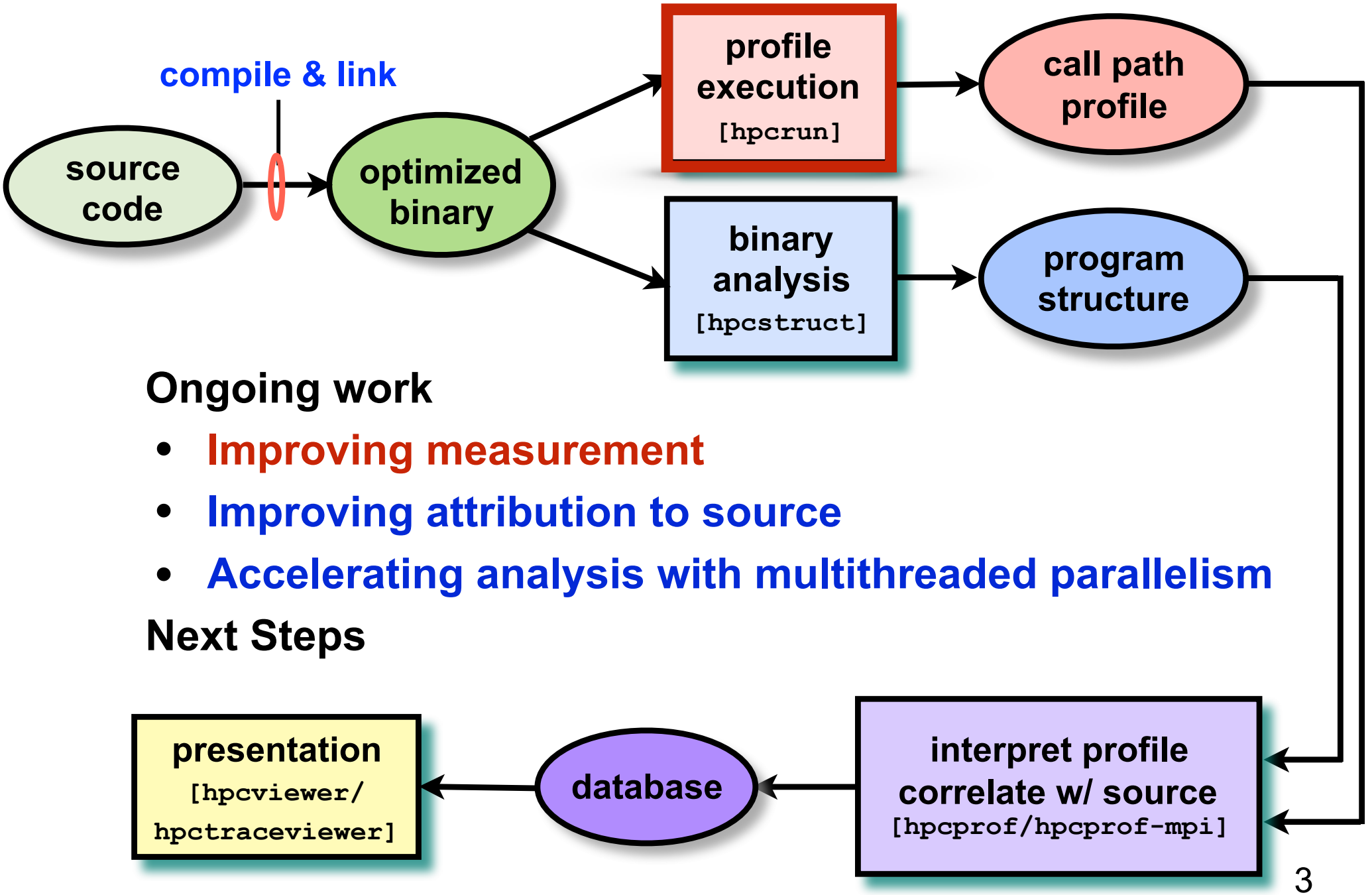
http://hpctoolkit.org

HPCToolkit

HPCToolkit Workflow



HPCToolkit Workflow



Ongoing work

- Improving measurement
- Improving attribution to source
- Accelerating analysis with multithreaded parallelism

Next Steps

Call Path Profiling of Optimized Code

- **Optimized code presents challenges for stack unwinding**
  - **optimized code often lacks frame pointers**
  - **routines may have multiple epilogues, multiple frame sizes**
  - **code may be partially stripped: no info about function bounds**
- **HPCToolkit's approach for nearly a decade**
  - **use binary analysis to compute unwinding recipes for intervals**
    - **often, no compiler information to assist unwinding is available**
  - **cache unwind recipes for reuse at runtime (more about this later)**

Nathan R. Tallent, John Mellor-Crummey, and Michael W. Fagan. Binary analysis for measurement and attribution of program performance. Proceedings of ACM *PLDI. ACM*, New York, NY, USA, 2009, 441–452. Distinguished Paper. (doi:10.1145/1542476.1542526)

Challenges for Unwinding

- **Binary analysis of optimized multithreaded applications has become increasingly difficult**
  - **previously: procedures were typically contiguous**
  - **today: procedures are often discontiguous**

Code generated by Intel's OpenMP compiler

```
void f(…) {
…
#pragma omp parallel
{
  …
}
…
}
```

New Unwinding Approach in HPCToolkit

- Use libunwind to unwind procedure frames where compiler-provided information is available
- Use binary analysis for procedure frames where no unwinding information is available
- Transition seamlessly between the two approaches
- Status:
  - first implementation for x86_64 completed on Friday
  - under evaluation

Surprises

- libunwind sometimes unwound incorrectly from signal contexts [our fixes are now in libunwind git]
- On Power, register frame procedures are not only at call chain leaves [unwind fixes in an hpctoolkit branch]

Caching Unwind Recipes in HPCToolkit

Concurrent Skip Lists

- **Two-level data structure: concurrent skip list of binary trees**
  - **maintain a concurrent skip list of procedure intervals**
    - **[proc begin, proc end)**
  - **associate an immutable balanced binary tree of unwind recipes with each procedure interval**
- **Synchronization needs**
  - **scalable reader/writer locks [Brandenburg & Anderson; RTS '10]**
    - **read lock: find, insert**
    - **write lock: delete**
  - **MCS queuing locks [Mellor-Crummey & Scott; ACM TOCS '91]**
    - **lock skip-list predecessors to coordinate concurrent inserts**

Validating Fast Synchronization

- **Used C++ weak atomics in MCS locks and phase-fair reader/ writer synchronization**
  - **against Herb Sutter's advice**
    - **C++ and Beyond 2012: atomic<> Weapons (bit.ly/atomic_weapons)**
  - **as Herb predicted: we got it wrong!**
- **Wrote small benchmarks that exercised our synchronization**
- **Identified bugs with CDS checker - model checker for C11 and C++11 Atomics**
  - **http://plrg.eecs.uci.edu/software_page/42-2/**

Brian Norris and Brian Demsky. CDSchecker: checking concurrent data structures written with C/C++ atomics. Proceedings of the 2013 ACM SIGPLAN OOPSLA. 2013. ACM, New York, NY, USA, 131-150. (doi: 10.1145/2509136.2509514)

- **Fixed them**
- **Validated the use of C11 atomics by our primitives**

We recommend CDS checker to others facing similar issues

Understanding Kernel Activity and Blocking

- Some programs spend a lot of time in the kernel or blocked
- Understanding their performance requires measurement of kernel activity and blocking

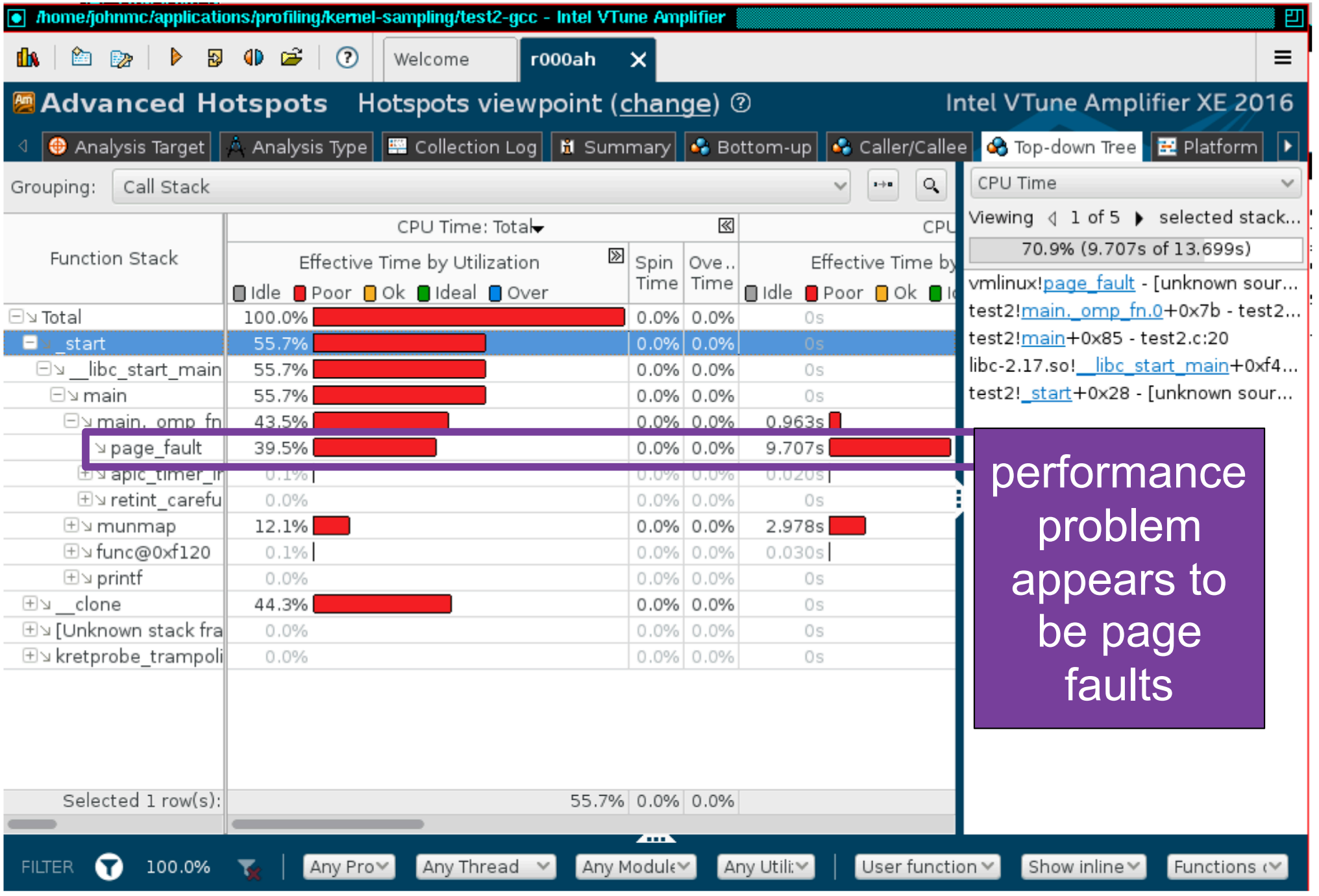
```
…
#define N  1000000000
#define REPS 32

int main()
{
        omp_set_num_threads(2);
        int pagesize = sysconf(_SC_PAGESIZE);
        printf("page size = %d\n", pagesize);
        int nextpage = pagesize/sizeof(int);
        for (int i=0; i < REPS; i++) {
                int *v = malloc(sizeof(int) * N);
#pragma omp parallel
                {
#pragma omp for
                        for(int j = 0; j < N; j += nextpage) {
                                v[j] = 5;
                        }
                }
                free(v);
        }
        return 0;
}
```

Measuring Kernel Activity and Blocking

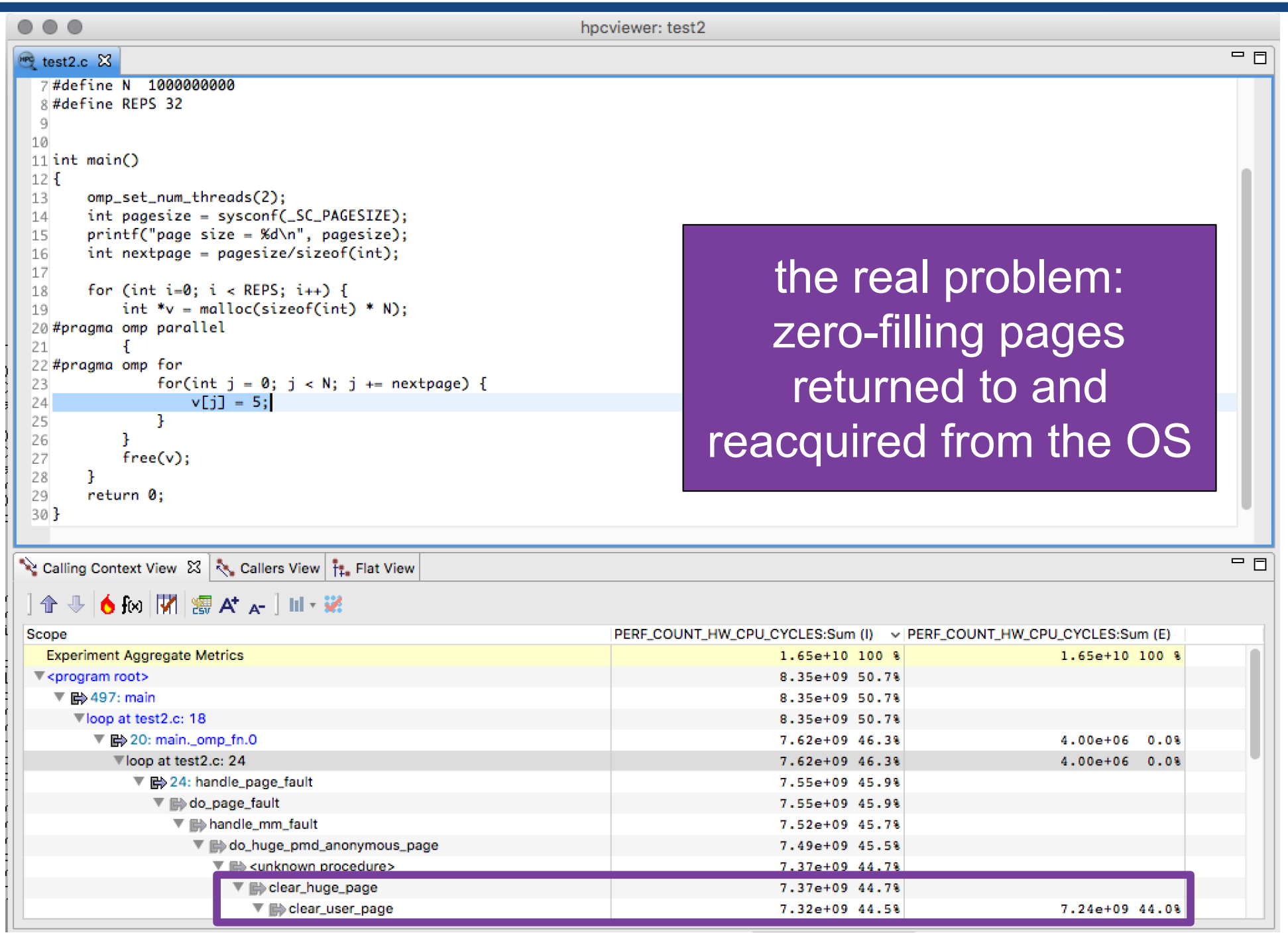
- **Problem**
  - Linux timers and PAPI are inadequate
    - neither measure nor precisely attribute kernel activity
- **Approach**
  - layer HPCToolkit directly on top of Linux perf_events
  - also sample kernel activity: perf_events collect kernel call stack
  - use sampling in conjunction with Linux CONTEXT_SWITCH events to measure and attribute blocking

VTune Result

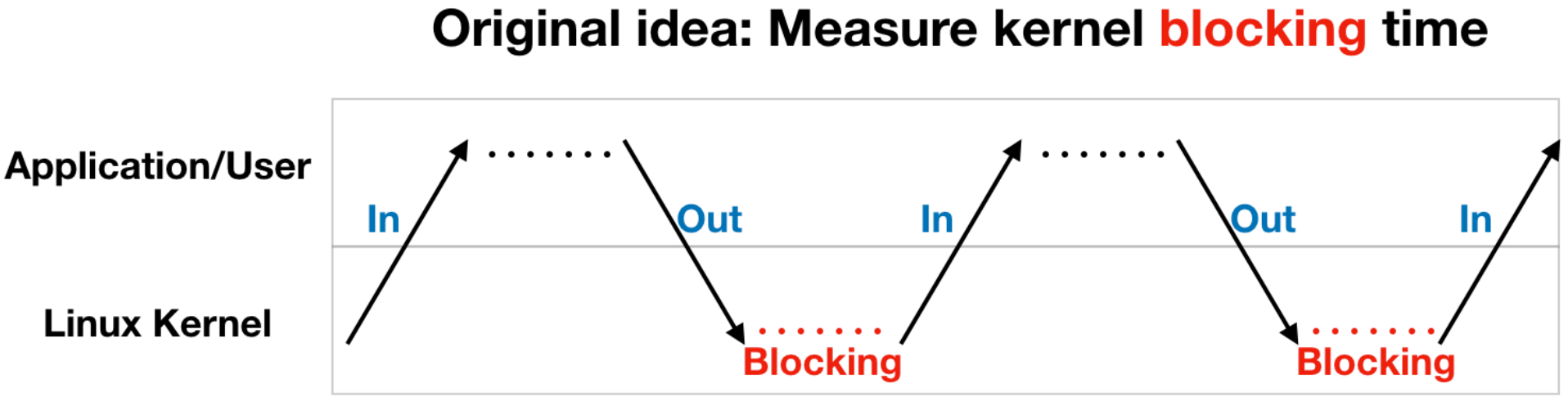


performance problem appears to be page faults

Understanding Kernel Activity with HPCToolkit



Kernel Blocking

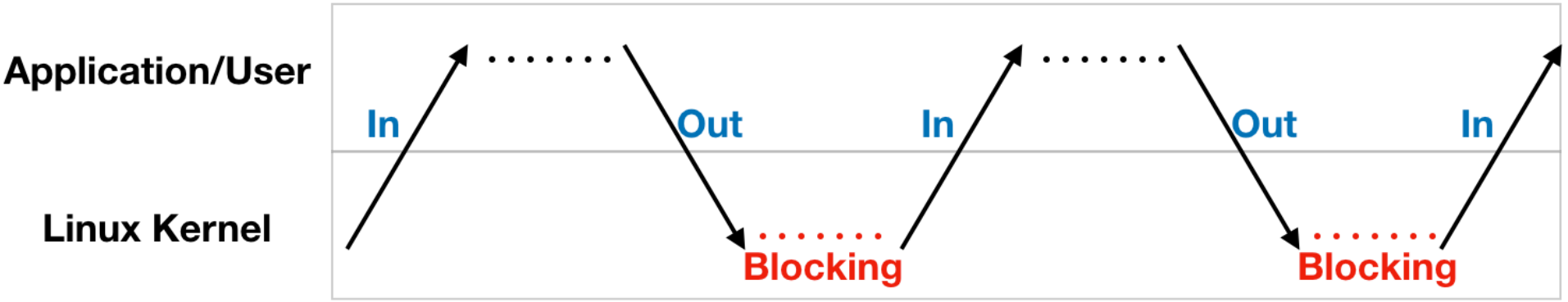


Surprise

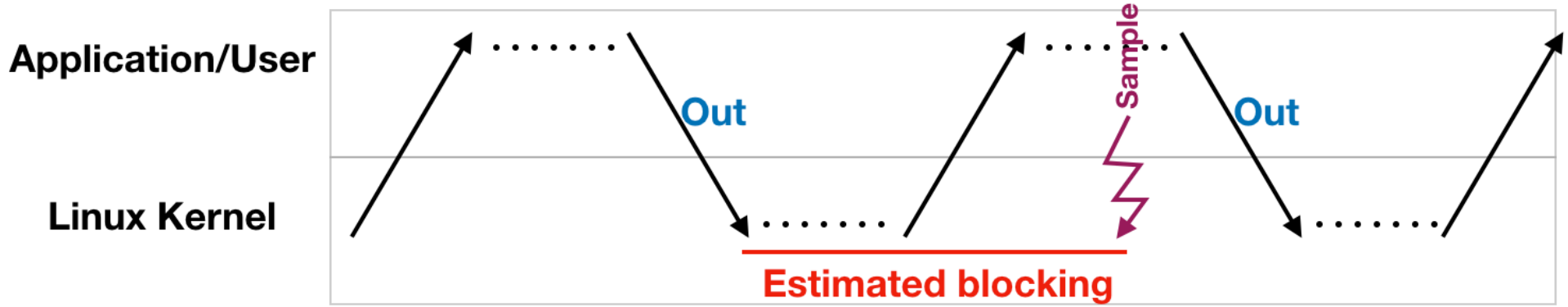
- **Third-party monitoring: SWITCH_OUT & SWITCH_IN**
- **First party monitoring: SWITCH_OUT only**
- **IBM Linux team working to upstream a fix**

Kernel Blocking

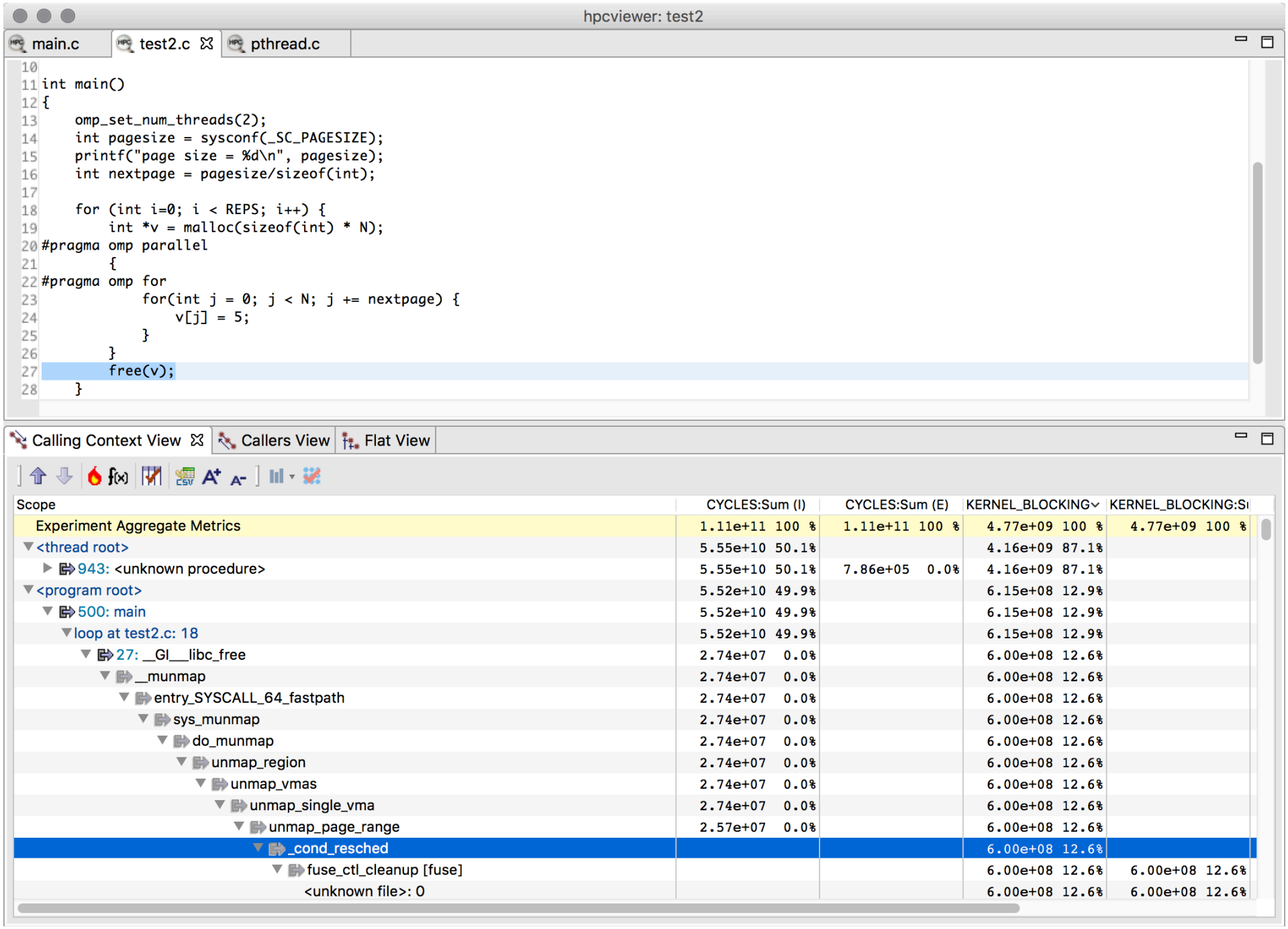
Original idea: Measure kernel blocking time



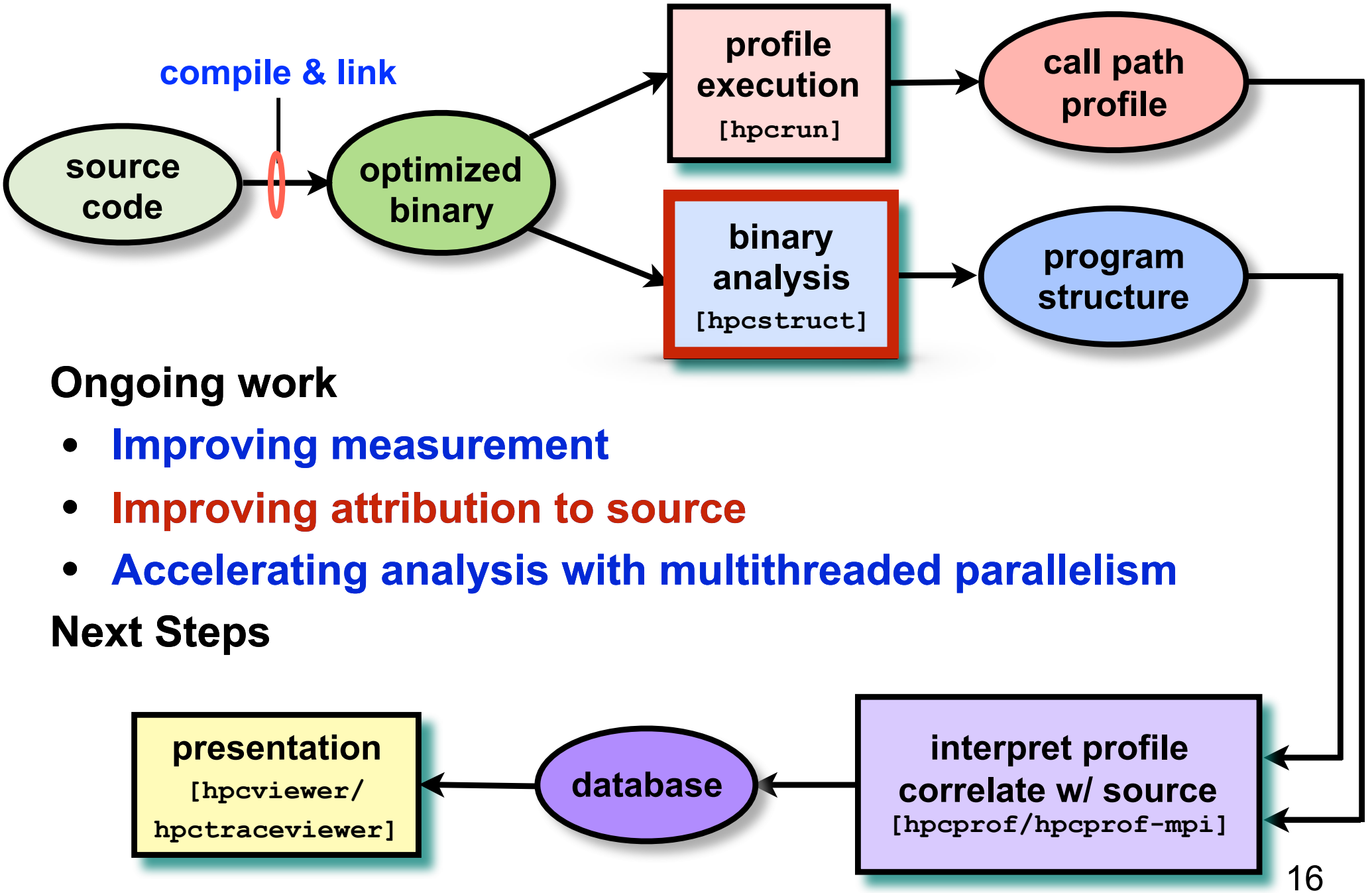
Approximation: Estimate kernel blocking time



Measuring Kernel Blocking



HPCToolkit Workflow

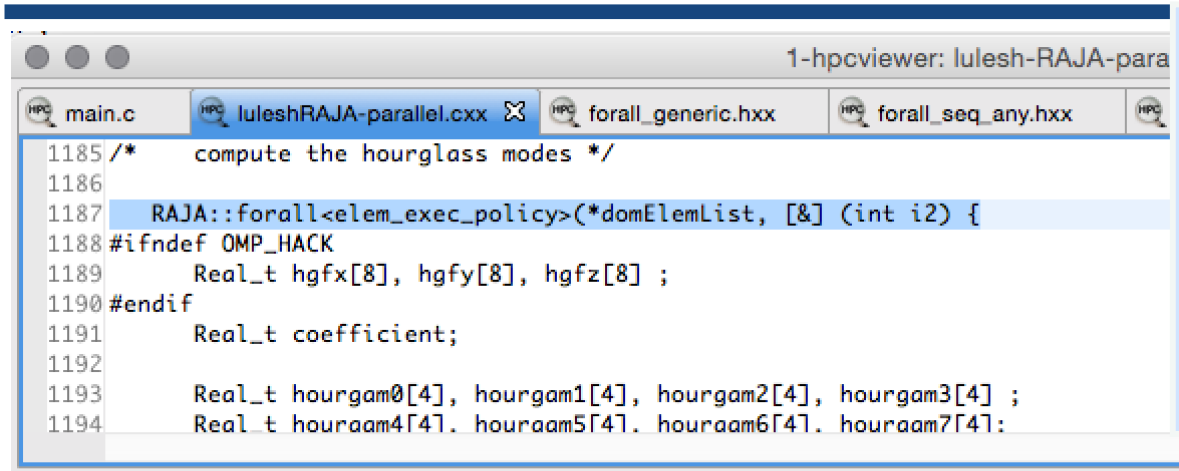


Ongoing work

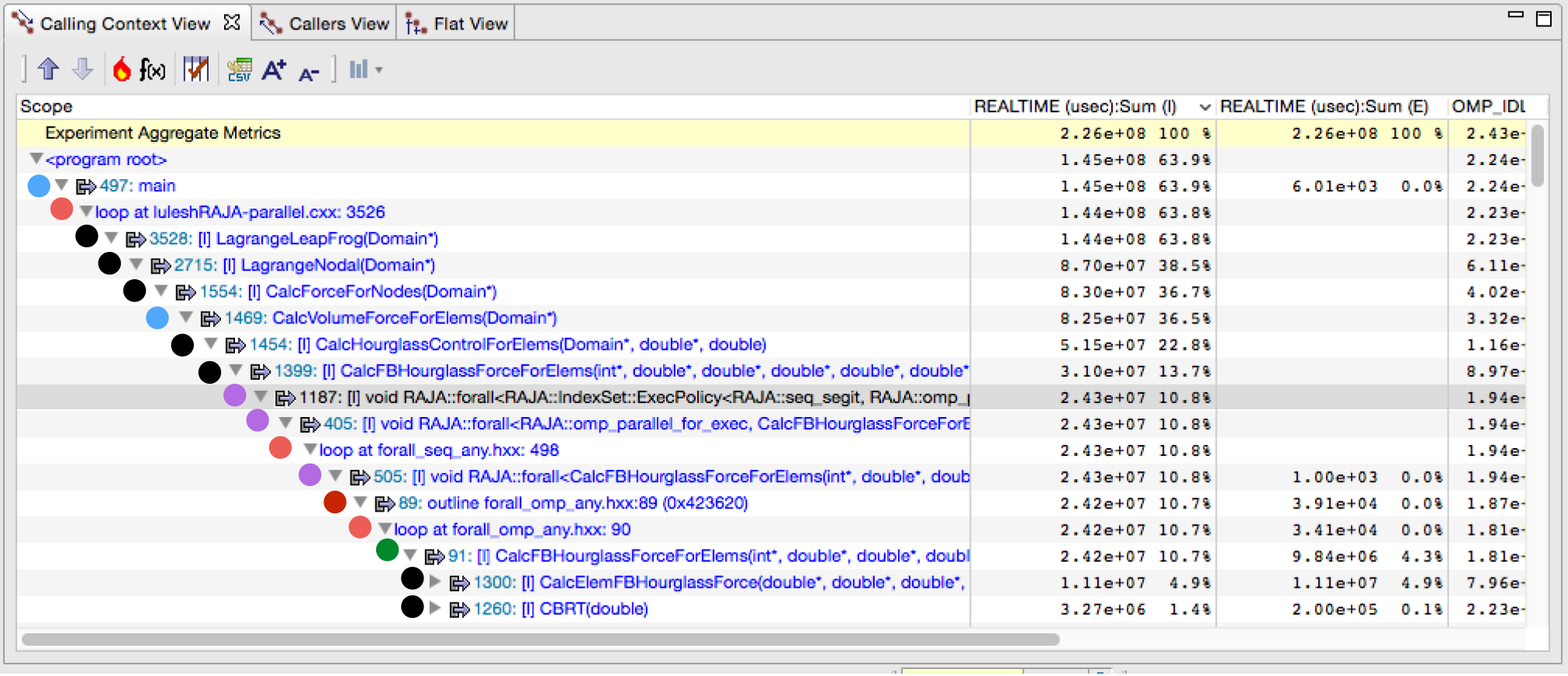
- Improving measurement
- Improving attribution to source
- Accelerating analysis with multithreaded parallelism

Next Steps

Binary Analysis with hpcstruct



- function calls
- inlined functions
- inlined RAJA templates
- loops
- outlined OMP loop
- lambda function



Binary Analysis of GPU Code

- Challenge: NVIDIA is very closed about their code
  - has not shared any CUBIN documentation even through NDA
- Awkward approach: reverse engineer CUBIN binaries
- Findings
  - each GPU function is in its own text segment
  - all text segments begin at offset 0
  - result: all functions begin at 0 and overlap
- Goal
  - use Dyninst to analyze CUBINs in hpcstruct
- Challenge
  - Dyninst SymtabAPI and ParseAPI are not equipped to analyze overlapping functions and regions
- Approach
  - memory map CUBIN load module
  - relocate text segments, symbols, and line map in hpcstruct prior to analysis using Dyninst inside

Binary Analysis of CUBINs: Preliminary Results

```
32 __device__ __forceinline__ float g(const float a, const float b)
33 {
34         return a+b;
35 }
36
37 __device__ __forceinline__ float f(const float a, const float b)
38 {
39   return g(a,b);
40 }
41
42 /**
43  * CUDA Kernel Device code
44  *
45  * Computes the vector addition of A and B into C. The 3 vectors have the same
46  * number of elements numElements.
47  */
48 __global__ void
49 vectorAdd(const float *A, const float *B, float *C, int numElements)
50 {
51     int i = blockDim.x * blockIdx.x + threadIdx.x;
52
53     if (i < numElements)
54     {
55         C[i] = f(A[i],B[i]);
56     }
57 }
```

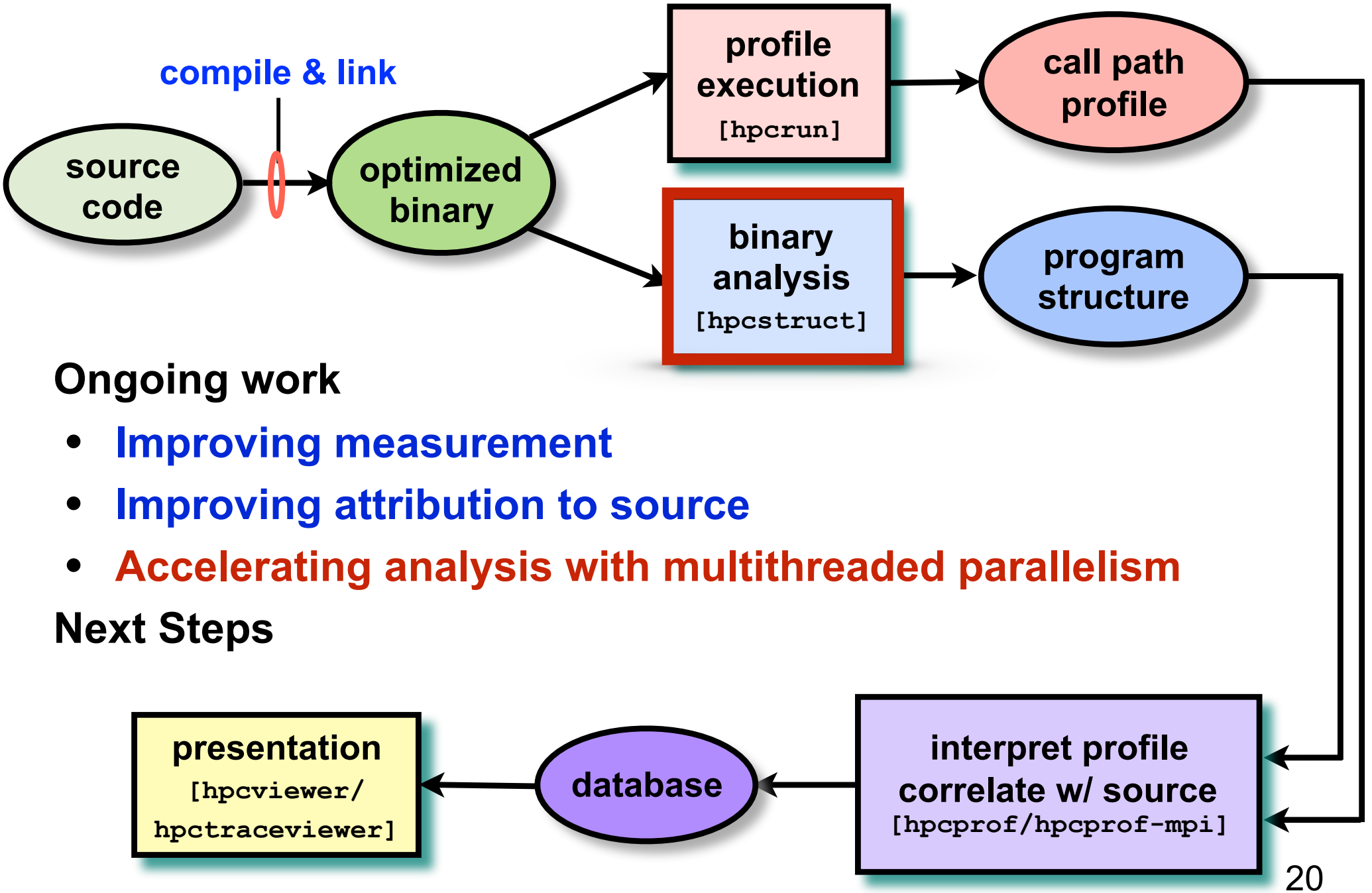
Limitation: CUBINs currently only have inlining information for unoptimized code

Next step: full analysis of heterogeneous binaries

- **host binary with GPU load modules embedded as segments**

```
<P i="32" n="vectorAdd(float const*, float const*, float*, int)" ln="_Z9vectorAddPKfS0_Pfi" l="0" v="{[0xc1e0-0xc1e1)}">
  <S i="33" l="0" v="{[0xc1e0-0xc760)}"/>
  <S i="34" l="0" v="{[0xc1e0-0xc318)}"/>
  <A i="35" l="39" f="/home/johnmc/llnl-visit/cuda-test/vectorAdd/vectorAdd.cu" n="&lt;inline&gt;" v="{}">
    <S i="36" l="39" v="{[0xc5e8-0xc5f0)}"/>
    <S i="37" l="51" v="{[0xc318-0xc370)}"/>
    <S i="38" l="53" v="{[0xc370-0xc398)}"/>
    <S i="39" l="55" v="{[0xc398-0xc5e8) [0xc618-0xc710)}"/>
    <S i="40" l="57" v="{[0xc710-0xc760)}"/>
  </A>
  <A i="41" l="55" f="&lt;unknown&gt;" n="" v="{}">
    <A i="42" l="39" f="/home/johnmc/llnl-visit/cuda-test/vectorAdd/vectorAdd.cu" n="f(float, float)" v="{}">
      <S i="43" l="39" v="{[0xc608-0xc618)}"/>
      <A i="44" l="39" f="/home/johnmc/llnl-visit/cuda-test/vectorAdd/vectorAdd.cu" n="" v="{}">
        <A i="45" l="34" f="/home/johnmc/llnl-visit/cuda-test/vectorAdd/vectorAdd.cu" n="g(float, float)" v="{}">
          <S i="46" l="34" v="{[0xc5f0-0xc608)}"/>
        </A>
      </A>
    </A>
  </A>
</P>
```

HPCToolkit Workflow



Ongoing work

- Improving measurement
- Improving attribution to source
- Accelerating analysis with multithreaded parallelism

Next Steps

Parallel Binary Analysis: Why?

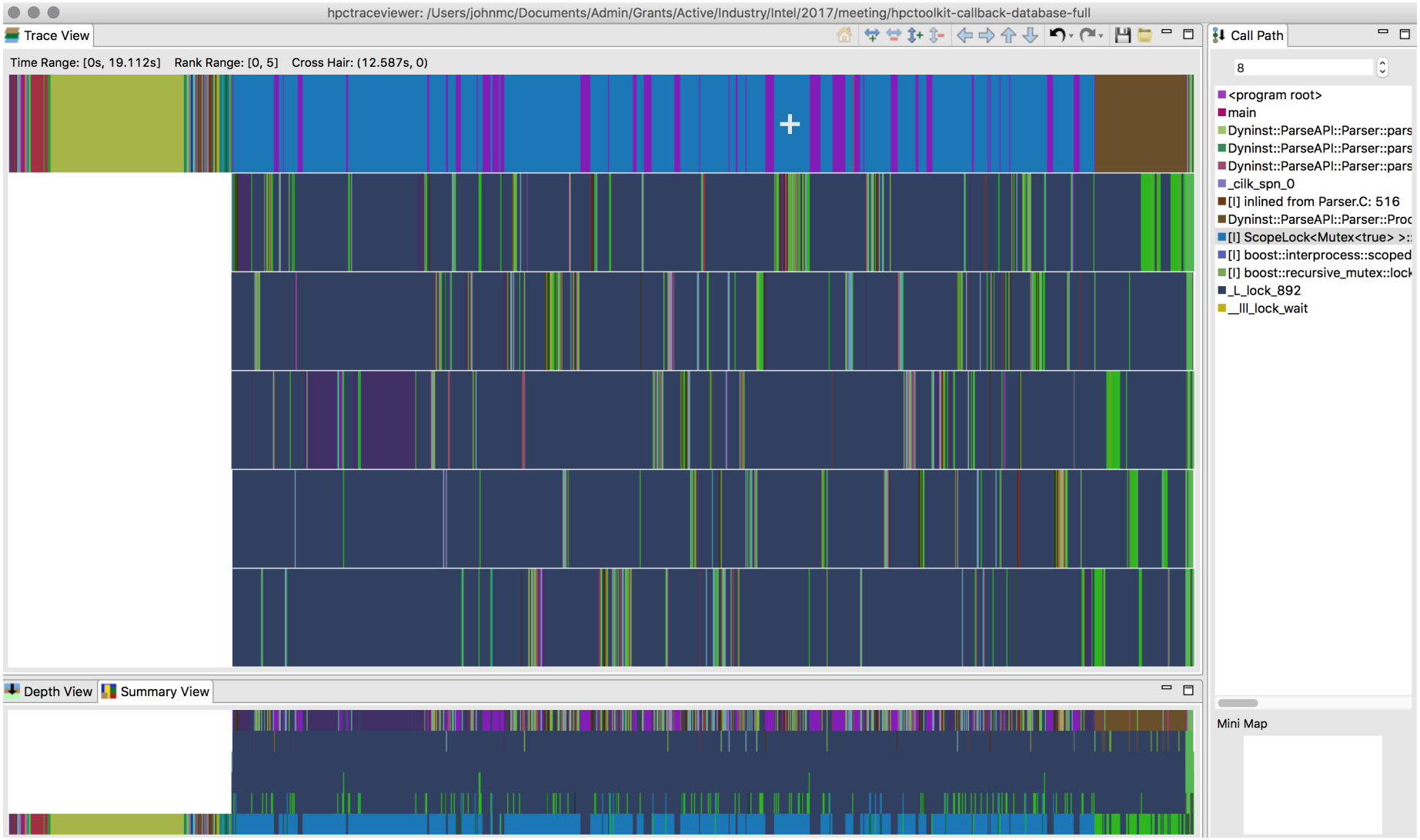
- Static binaries on DOE Cray systems are big
- Binary analysis of large application binaries is too slow
  - NWchem binary from Cray platform at NERSC (Edison)
    157M (104M text)
  - serial hpcstruct based on Dyninst v9.3.2
    Intel Westmere @ 2.8GHz: 10 minutes
    KNL @ 1.4GHz: 28 minutes
- Tests user patience and is an impediment to tool use

Parallelizing hpcstruct: Two Approaches

- **Light**
  - — approach
    - – parse the binary with Dyninst's ParseAPI, SymtabAPI
    - – parallelize hpcstruct's binary analysis, which runs atop Dyninst APIs

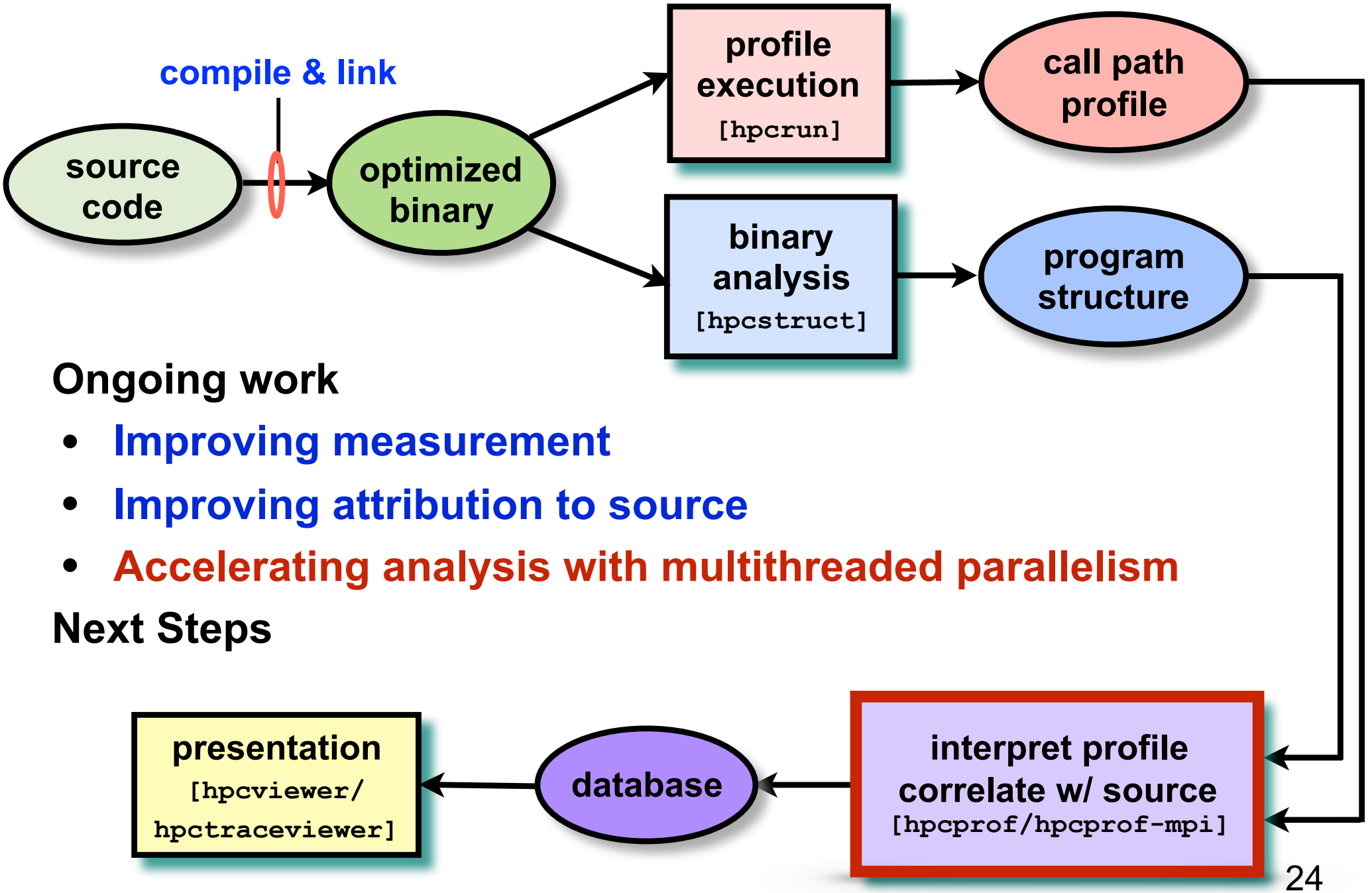
- **Full**
  - — approach
    - – parallelize parsing of the binary with Dyninst
    - – Dyninst supports a callback when a procedure parse is finalized
      - register callback to perform hpcstruct analysis at that time
  - — potential benefits
    - – opportunity for speedup as much as number of procedures

Parallel Binary Parsing with Dyninst



Added parallelism using CilkPlus constructs

HPCToolkit Workflow



Ongoing work

- Improving measurement
- Improving attribution to source
- Accelerating analysis with multithreaded parallelism

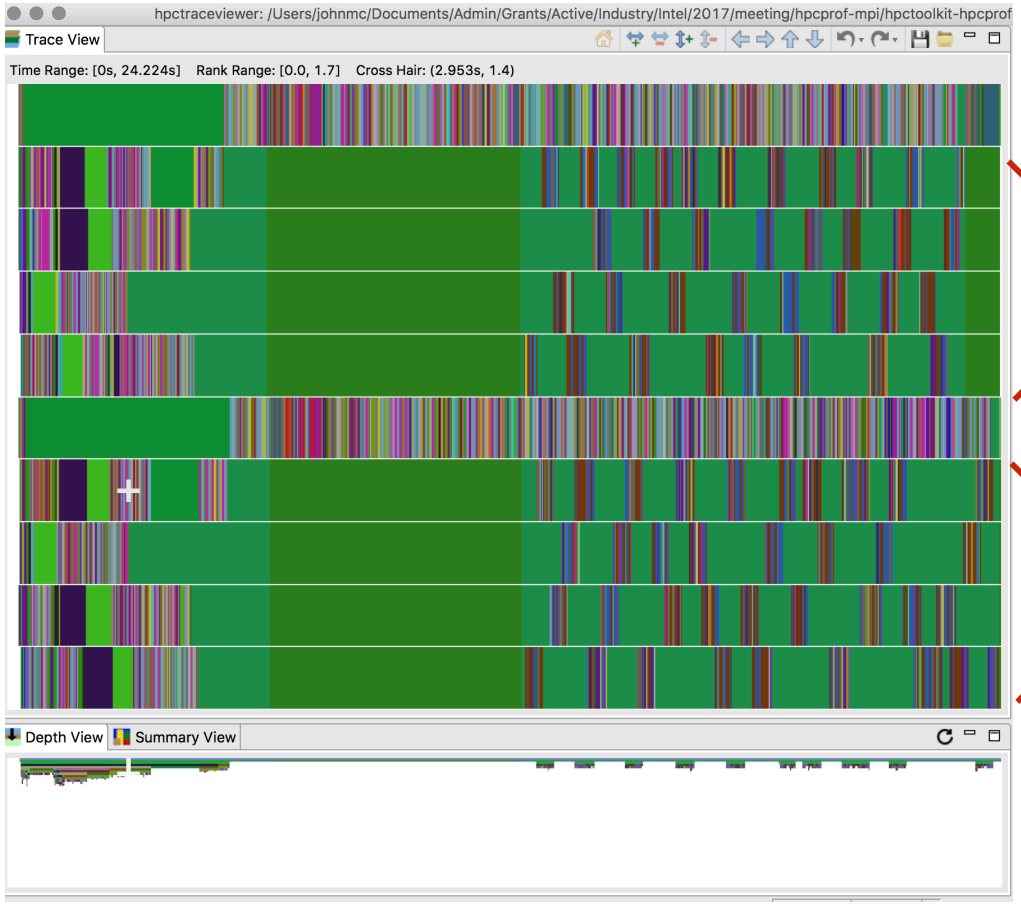
Next Steps

Accelerating Data Analysis

- **Problem**
  - need massive parallelism to analyze large-scale measurements
  - MPI-everywhere is not the best way to use Xeon Phi
- **Approach**
  - add thread-level parallelism to hpcprof-mpi
    - threads collaboratively process multiple performance data files

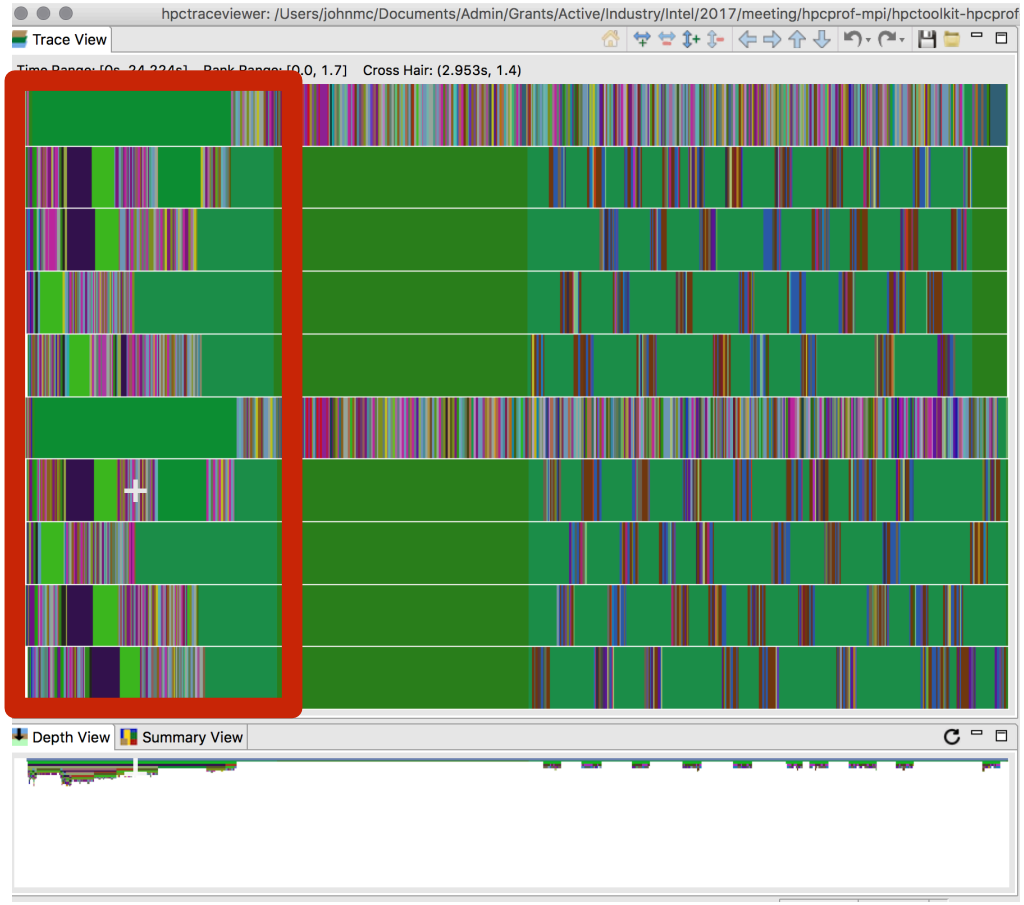
hpcprof-mpi with Thread-level Parallelism

- **Add thread-level parallelism with OpenMP**
  - **program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus**



hpcprof-mpi with Thread-level Parallelism

- **Add thread-level parallelism with OpenMP**
  - **— program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus**



merge profiles using a parallel reduction tree

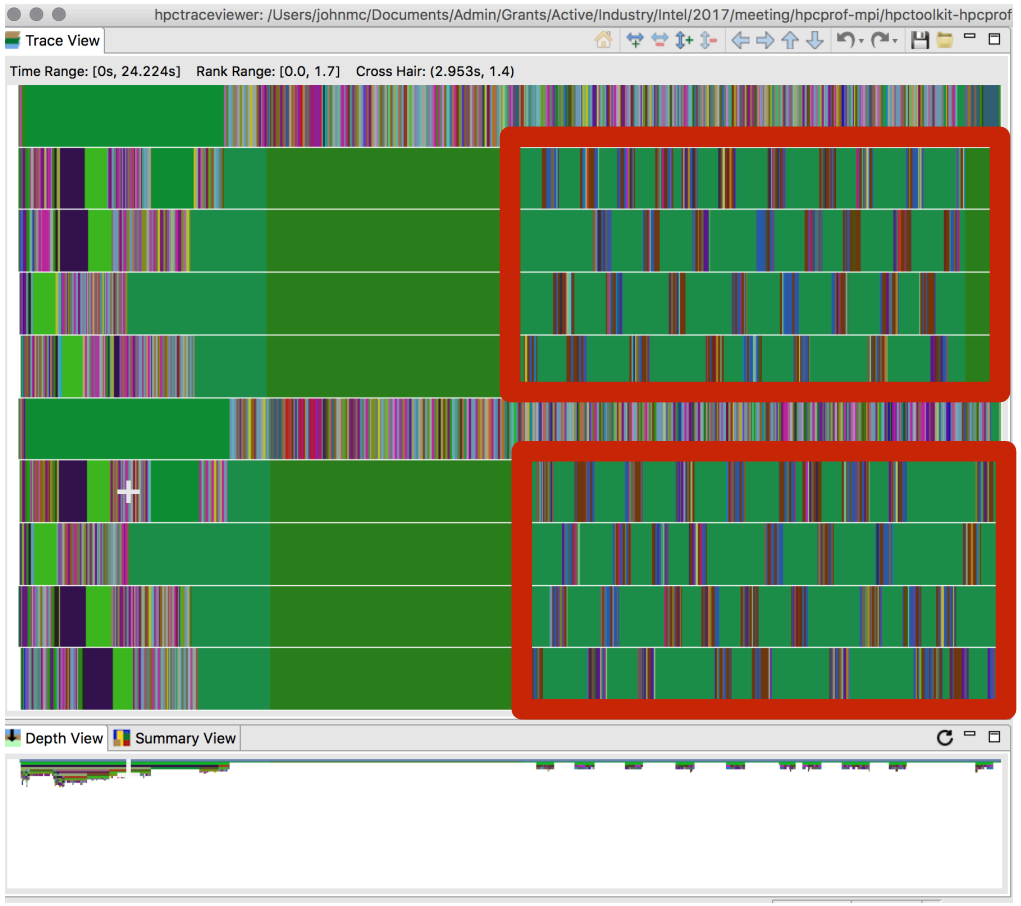
```
  if (upper == lower) { // singleton profile
    Prof::CallPath::Profile* p = readSingle(profileFiles, groupMap, lower, rFlags);
    return p;
  } else {  // multiple profiles
    Prof::CallPath::Profile* left = 0;
    Prof::CallPath::Profile* right = 0;
    uint mid = lower + (upper - lower)/2;
#pragma omp task shared(left, profileFiles, groupMap) \
  firstprivate(mergeTy, rFlags, mrgFlags, lower, mid)
    {
      left = readSet(profileFiles, groupMap, mergeTy, rFlags, mrgFlags, lower, mid);
    }
    {
      right = readSet(profileFiles, groupMap, mergeTy, rFlags, mrgFlags, mid + 1, upper);
    }
#pragma omp taskwait

    left->merge(*right, mergeTy, mrgFlags);
```

```
 Prof::CallPath::Profile* prof = 0;
#pragma omp parallel shared(prof)
{
#pragma omp master
{
  prof = readSet(profileFiles, groupMap, mergeTy,
                 rFlags, mrgFlags, 0, profileFiles.size() - 1);
}
}
#endif
```

hpcprof-mpi with Thread-level Parallelism

- **Add thread-level parallelism with OpenMP**
  - **— program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus**



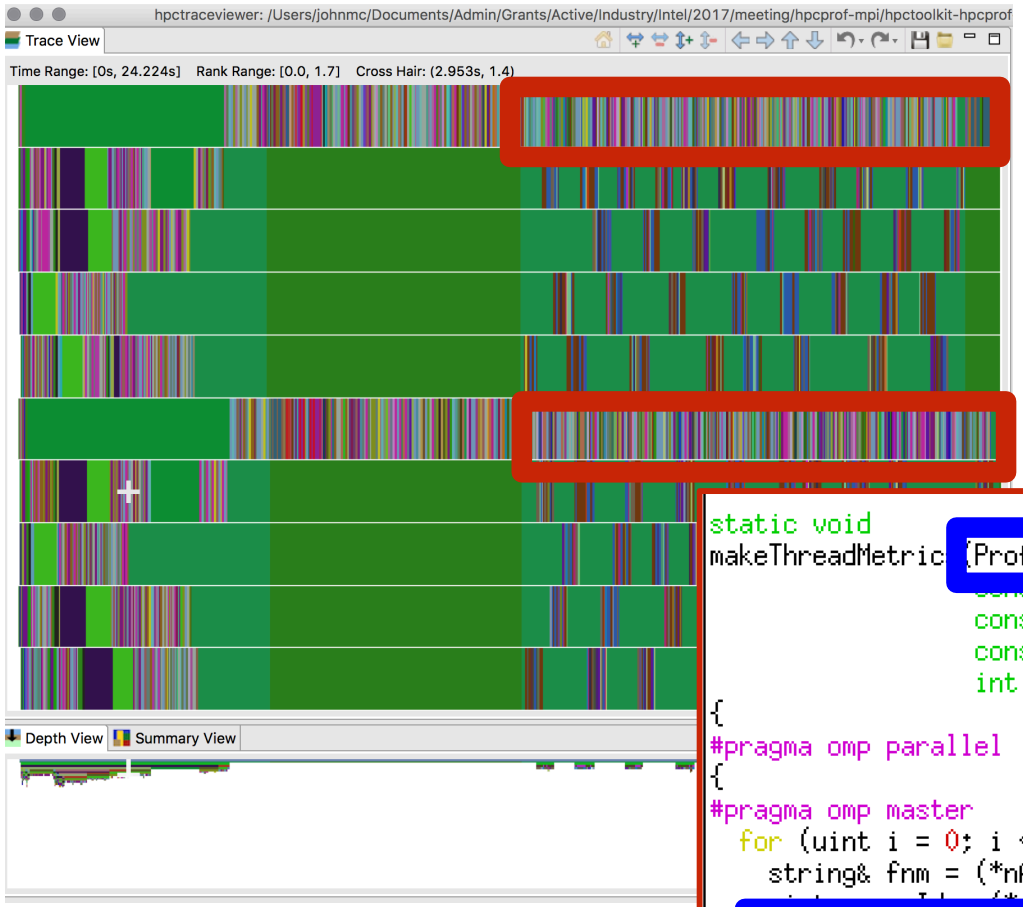
update traces asynchronously

```
void
Profile::merge_fixTrace(const CCT::MergeEffectList* mrgEffects)
{
  string *file = new string(m_traceFileName);
#pragma omp task firstprivate(file)
{
  doFixTrace(*file, mrgEffects);
  delete mrgEffects;
  delete file;
}
```

```
static void
makeThreadMetrics(Prof::CallPath::Profile& profGbl,
                  const Analysis::Args& args,
                  const Analysis::Util::NormalizeProfileArgs_t& nArgs,
                  const vector<uint>& groupIdToGroupSizeMap,
                  int myRank, int numRanks, int rootRank)
{
#pragma omp parallel
{
#pragma omp master
  for (uint i = 0; i < nArgs.paths->size(); ++i) {
    string& fnm = (*nArgs.paths)[i];
    uint groupId = (*nArgs.groupMap)[i];
    makeThreadMetrics_Lcl(profGbl, fnm, args, groupId, nArgs.groupMax, myRank);
  }
}
}
```

hpcprof-mpi with Thread-level Parallelism

- **Add thread-level parallelism with OpenMP**
  - **— program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus**

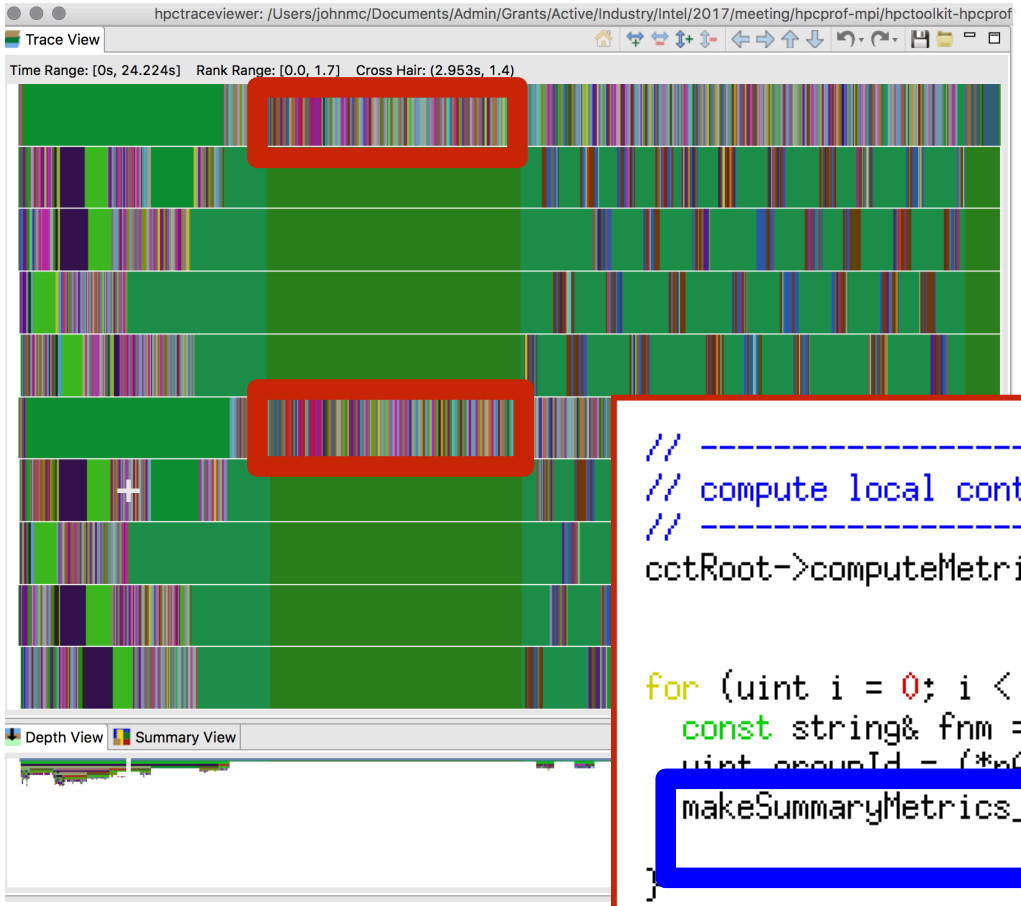


compute thread metrics locally using a global variable

```
static void
makeThreadMetric(Prof::CallPath::Profile& profGbl,
                 const Analysis::Util::NormalizeProfileArgs_t& nArgs,
                 const vector<uint>& groupIdToGroupSizeMap,
                 int myRank, int numRanks, int rootRank)
{
#pragma omp parallel
{
#pragma omp master
  for (uint i = 0; i < nArgs.paths->size(); ++i) {
    string& fnm = (*nArgs.paths)[i];
    makeThreadMetrics_Lcl(profGbl, fnm, args, groupId, nArgs.groupMax, myRank);
  }
}
}
```

hpcprof-mpi with Thread-level Parallelism

- **Add thread-level parallelism with OpenMP**
  - **— program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus**



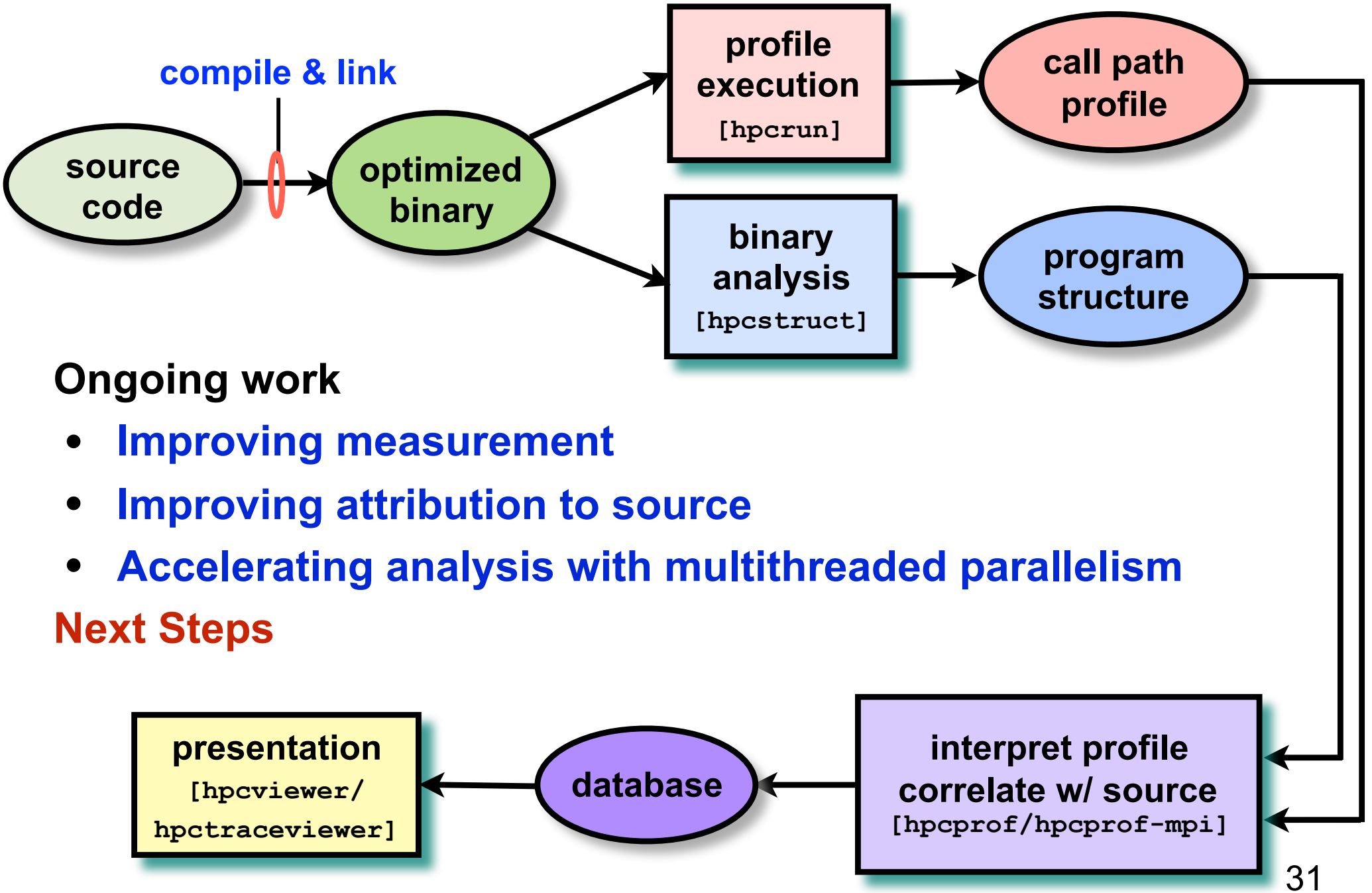
accumulate metric values locally into a global variable

```
// ------------------------------------------------------
// compute local contribution summary metrics (accumulate function)
// ------------------------------------------------------
cctRoot->computeMetricsIncr(mMgrGbl, mDrvdBeg, mDrvdEnd,
                            Prof::Metric::AExprIncr::FnInit);

for (uint i = 0; i < nArgs.paths->size(); ++i) {
  const string& fnm = (*nArgs.paths)[i];
  uint groupId = (*nArgs.groupMap)[i];

  makeSummaryMetrics_Lcl(profGbl, fnm, args, groupId, nArgs.groupMax,
                         groupIdToGroupMetricsMap, myRank);
}
```

HPCToolkit Workflow



Ongoing work

- Improving measurement
- Improving attribution to source
- Accelerating analysis with multithreaded parallelism

Next Steps

Next Steps

- Integrate hpcstruct and perf_events improvements into trunk
- Data-centric measurement with perf_events
- Continue work with Wisconsin on parallelization of hpcstruct
- Work with OpenMP community to finalize OMPT and OpenMP 5
  - test and validate new LLVM OMPT host-side implementation
  - integrate OMPT support for libomptarget into LLVM trunk
- Finish OpenMP 5 and CUDA support in HPCToolkit
- Improve support for measurement and analysis at scale
  - reduce file counts
  - improve multithreaded parallel analysis
- Explore GUI enhancements to improve developer workflows
- Add support for top-down models for architecture analysis