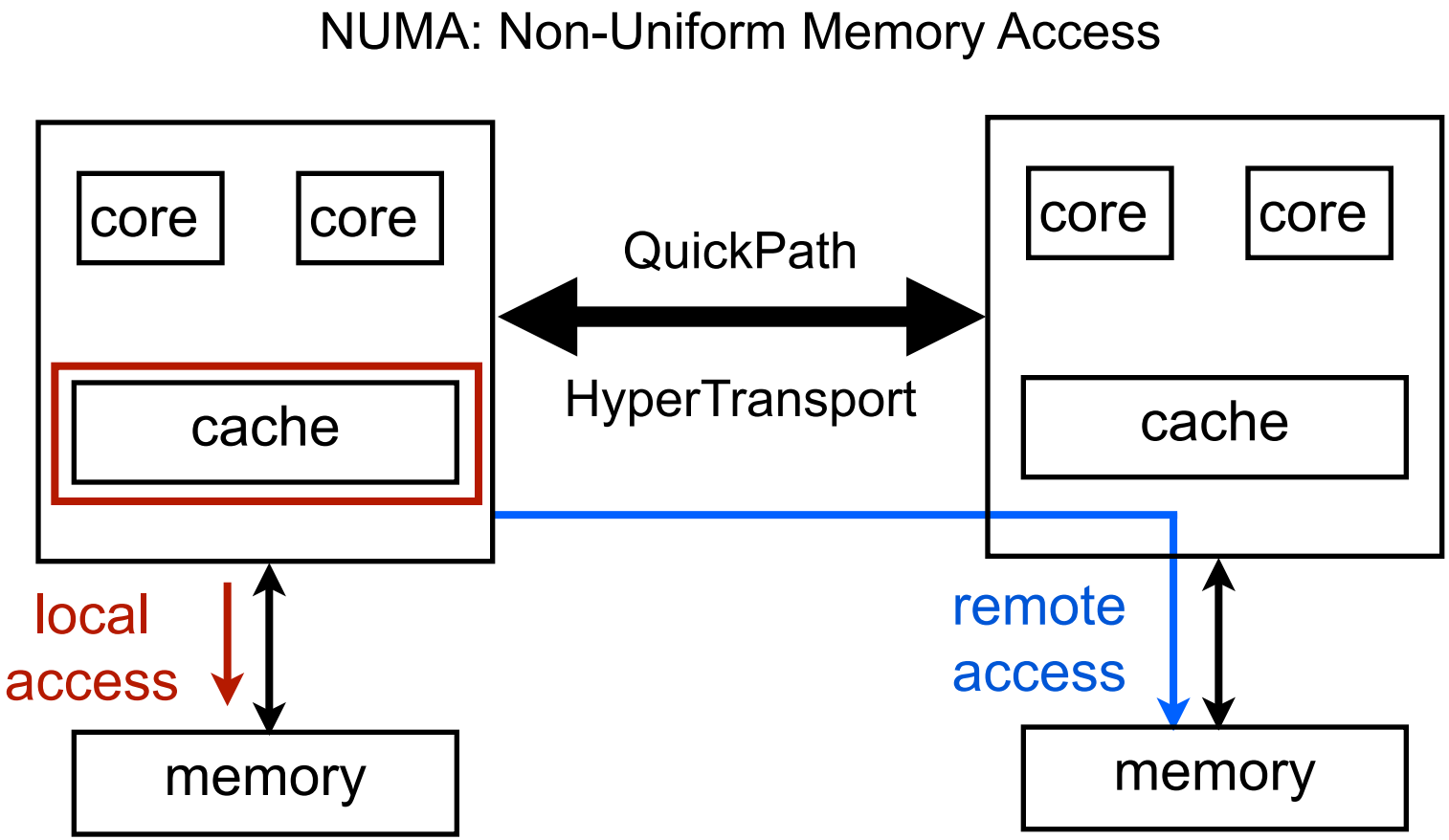
Leveraging Hardware Address Sampling for Memory Performance Insights

Xu Liu

Department of Computer Science
College of William and Mary
xl10@cs.wm.edu

Motivation: Memory is the Bottleneck



State of the Arts

simulation methods

deep insights

weaknesses:

- 2-5x overhead
- not real machines

low overhead with deep insights

deep insights with low overhead

measurement methods

low overhead

Hardware Address Sampling

- Features of address sampling
  - sample memory-related events (memory accesses, NUMA events)
  - capture effective addresses
  - record precise IP of sampled instructions or events
- Support in modern processors
  - AMD Opteron 10h and above: instruction-based sampling (IBS)
  - IBM POWER 5 and above: marked event sampling (MRK)
  - Intel Itanium 2: data event address register sampling (DEAR)
  - Intel Pentium 4 and above: precise event based sampling (PEBS)
  - Intel Nehalem and above: PEBS with load latency (PEBS-LL)
- Efficient memory measurement (SC'13)
  - code-centric analysis
  - data-centric analysis

Code-centric vs. Data-centric

- Code-centric attribution
  - problematic code sections
    - instruction, loop, function

```
1: #pragma omp parallel for num_threads(4)
2: for (i = 0; i < n; i++) {
3:    for(j = 0; j < n; j++) {
4:       for(k = 0; k < n; k++) {
5:          A[i, j, k] = A[i, j, k,] + B[j, i, k] + C[k, j, i];
6:       }
7:    }
8: }
```

code-centric profiling

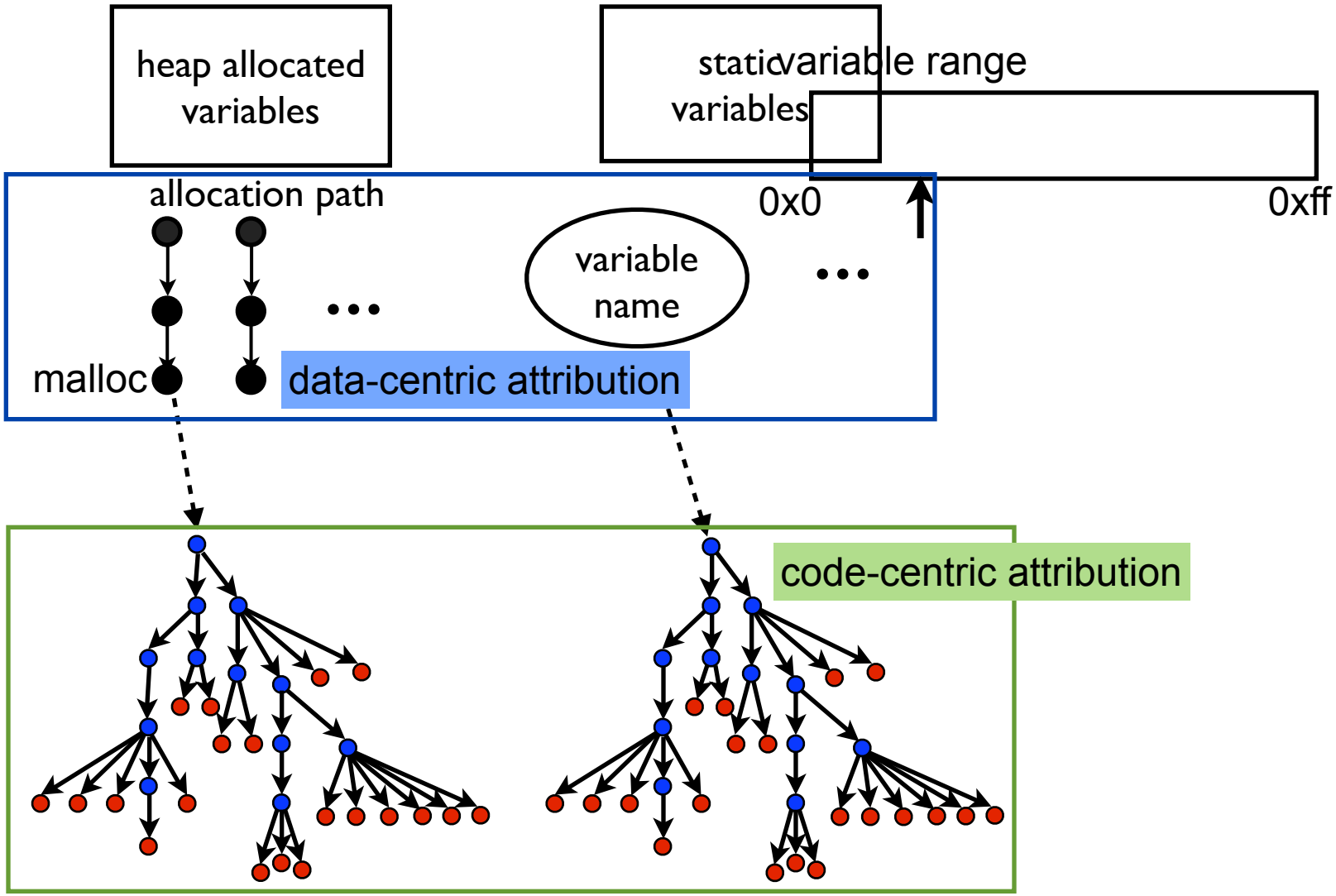
line 5: 100% latency

data-centric profiling

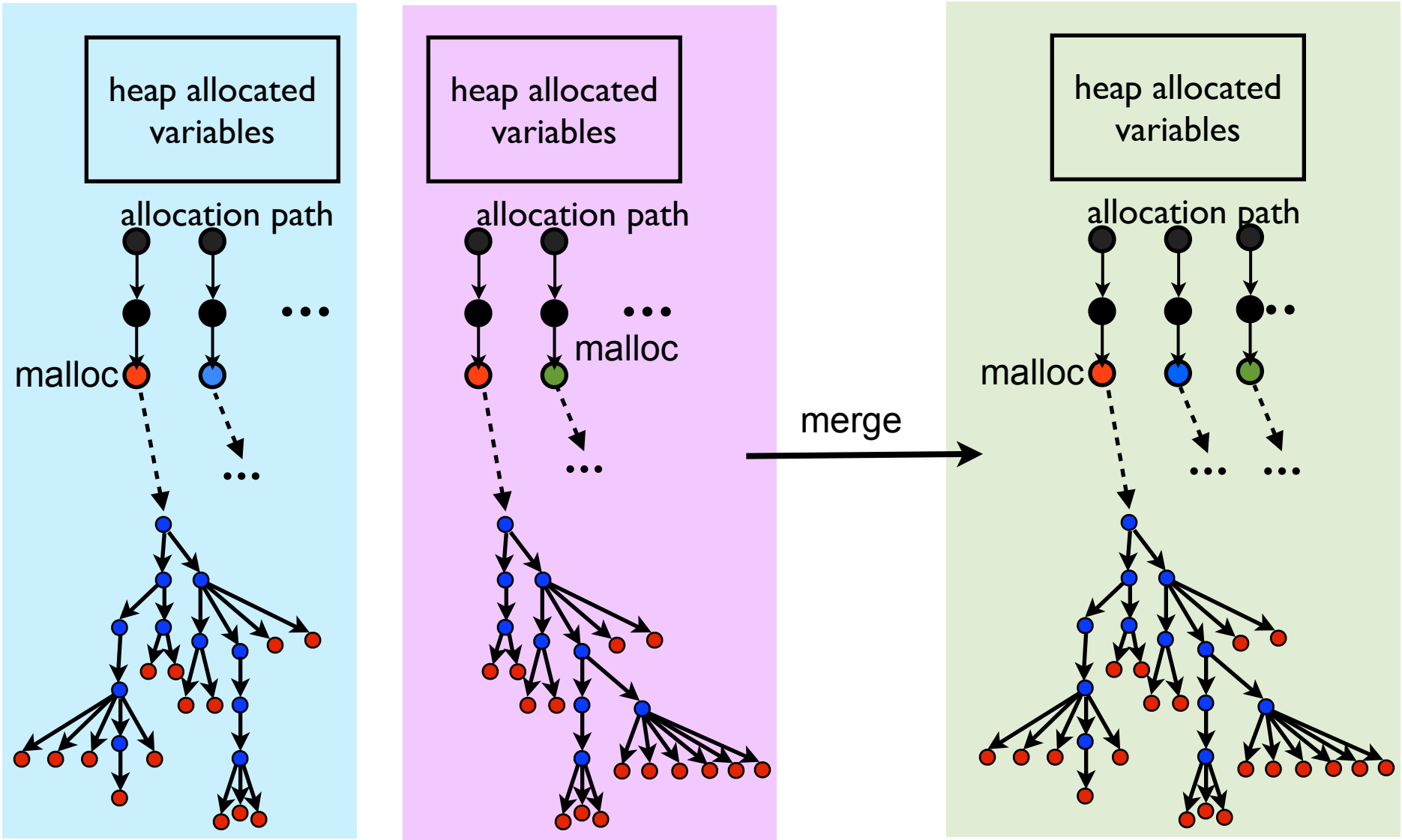
array A:
line 5: 1% latency
array B
line 5: 10% latency
array C
line 5: 89% latency

Combining code-centric and data-centric attribution provides additional insight

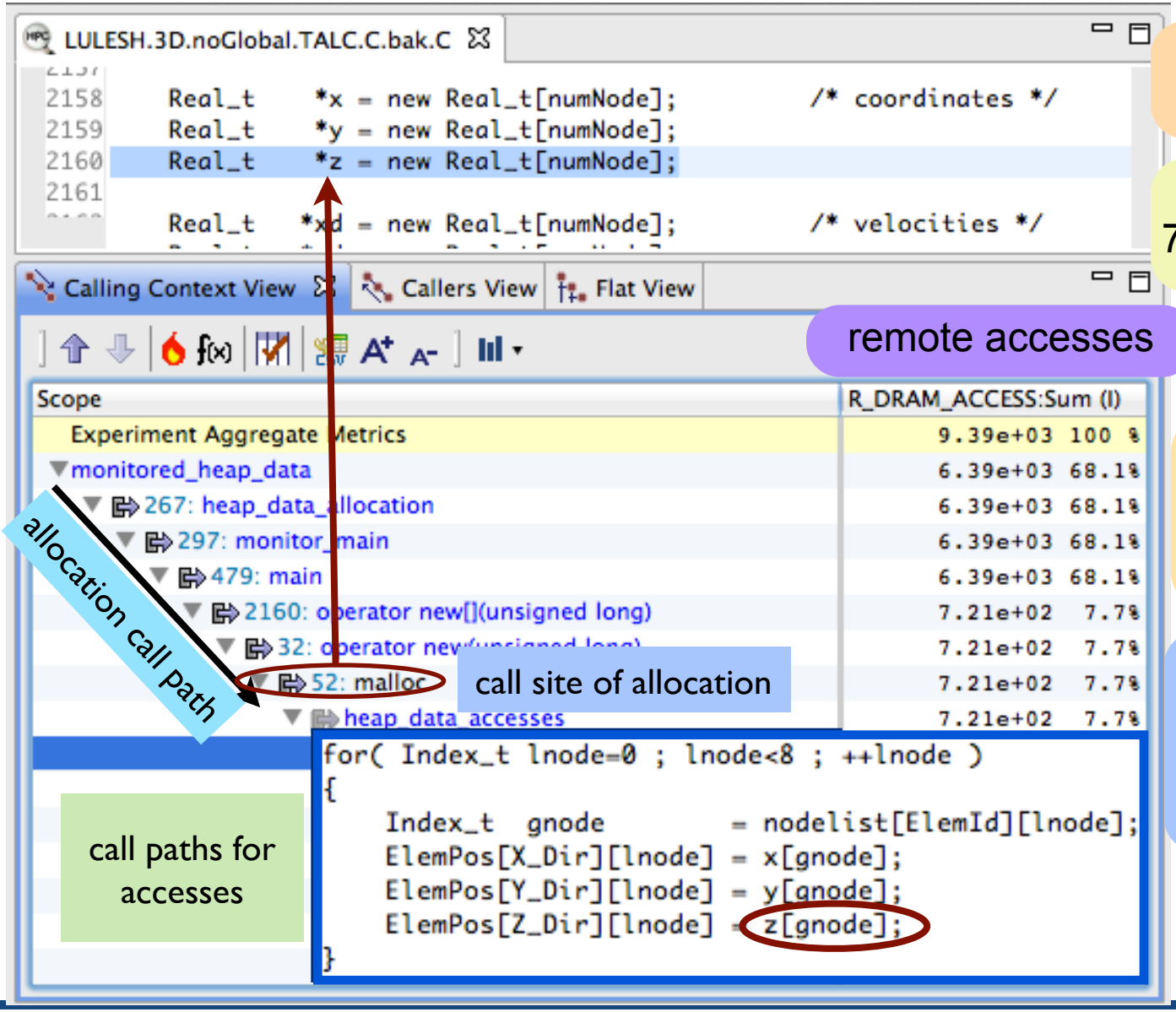
Attributing Samples



Aggregating Profiles



LULESH on Platform of 8 NUMA Domains



heap data:68% remote accesses

z accounts for 7.7% remote accesses

z is allocated in a NUMA domain but accessed by others

interleave pages of z across NUMA nodes 13% improvement in running time

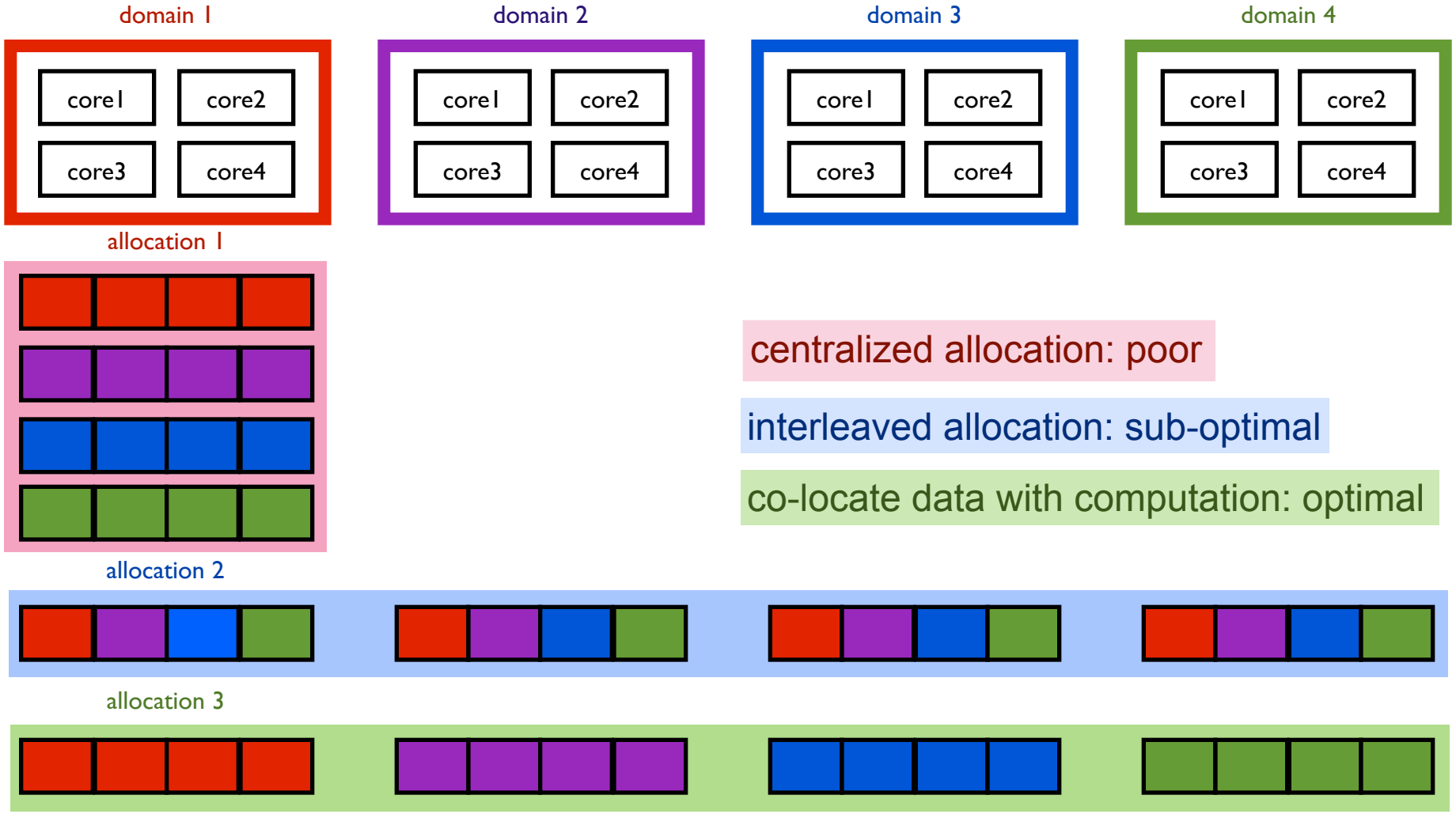
Existing Measurement is Inadequate

- Data collection + attribution ≠ optimal optimization
  - know problematic data objects but not know why
  - need more insights for optimization guidance
- Challenges for address sampling
  - very sparse memory access samples
  - not monitoring continuous memory accesses
- Opportunities for address sampling
  - effective addresses: analyze memory access patterns
  - data sources: understand where inefficiencies come from
  - latency: derive new latency metrics to quantify inefficiencies.

Beyond Data Collection and Attribution

- Published work
  - HPCToolkit-NUMA: analyzing NUMA bottlenecks (PPoPP'14)
  - ArrayTool: guiding array regrouping for better locality (PACT'14)
  - ScaAnalyzer: identifying memory scaling issues (SC'15)
  - StructSlim: guiding structure splitting (CGO'16)
  - Cheetah: detecting false sharing (CGO'16)
  - SMTAnalyzer: identifying SMT-aware optimization (HPDC'16)

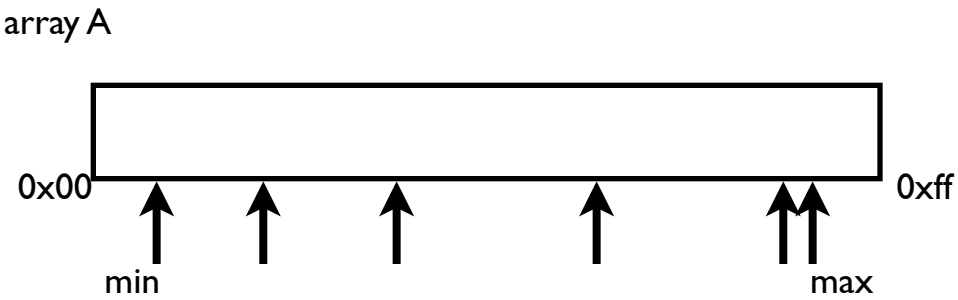
Interleaved Allocation is NOT Always Best



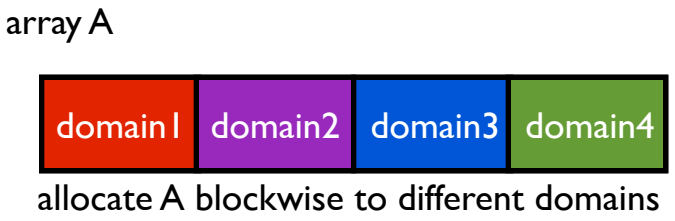
Goal: identify the best data distribution for a program

Memory Access Pattern Analysis

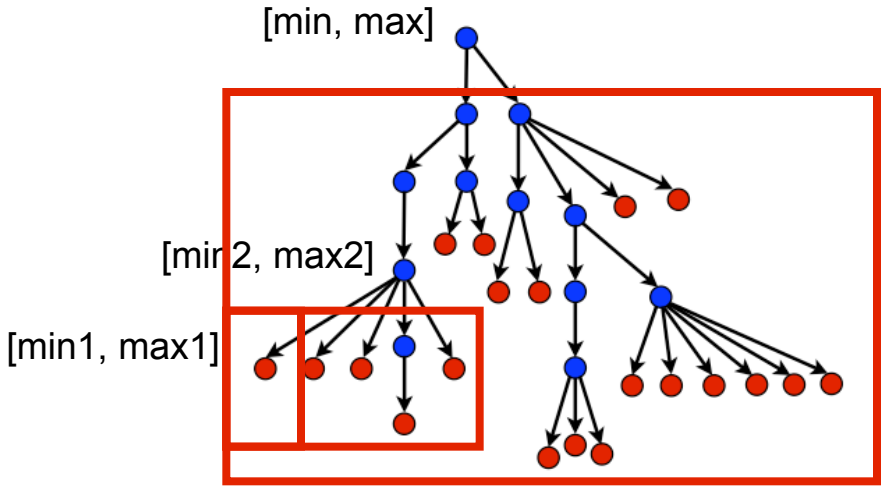
- Online data collection

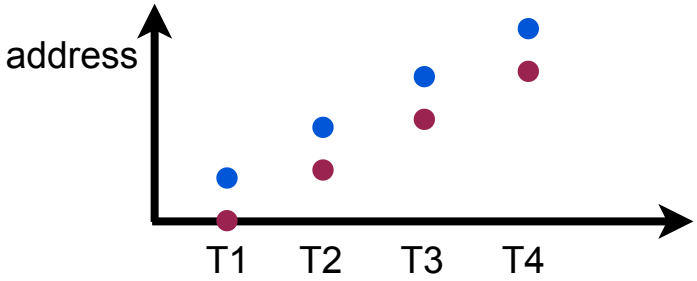


[min, max] per sampled memory access



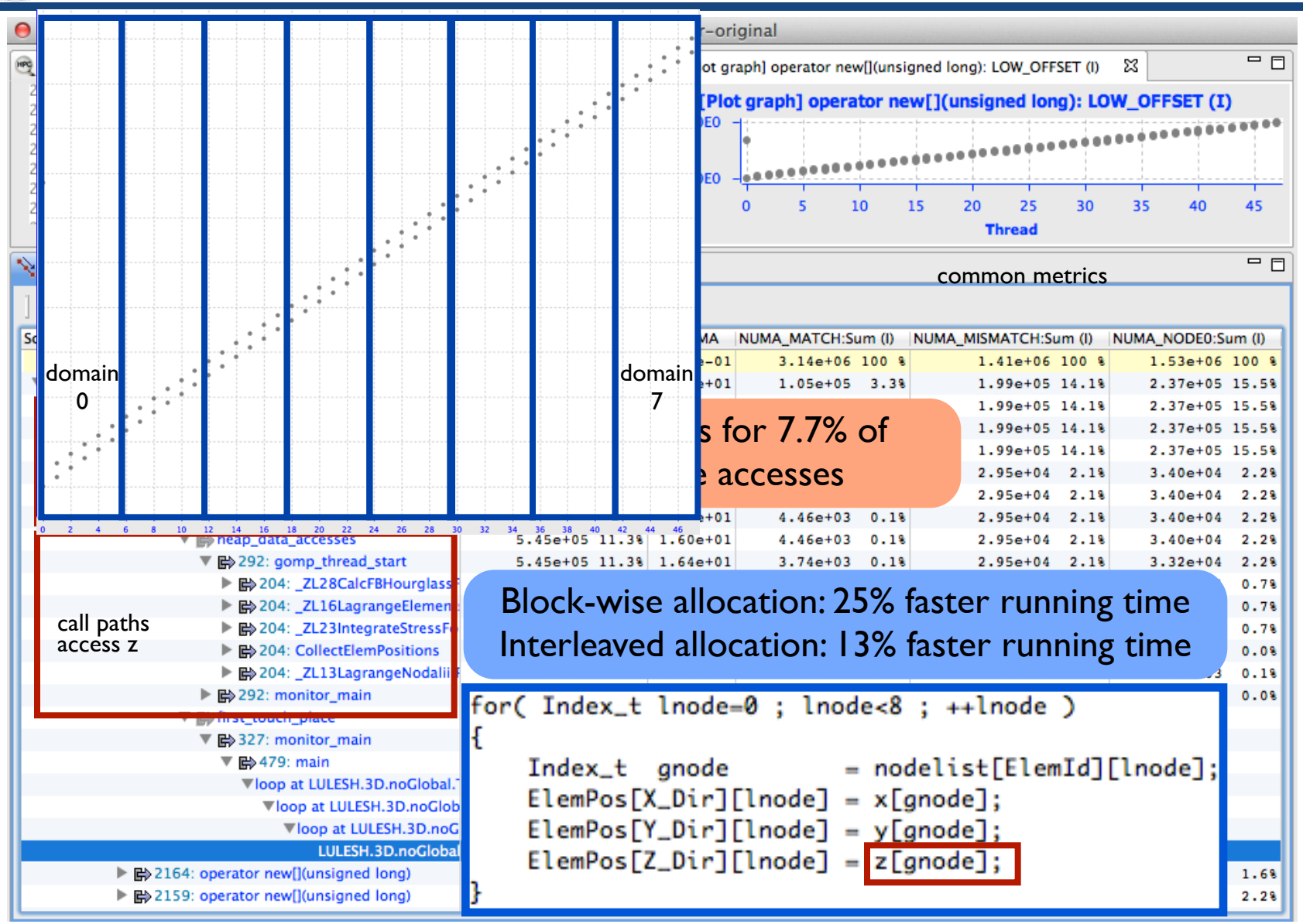
allocate A blockwise to different domains





balanced allocation + maximum locality

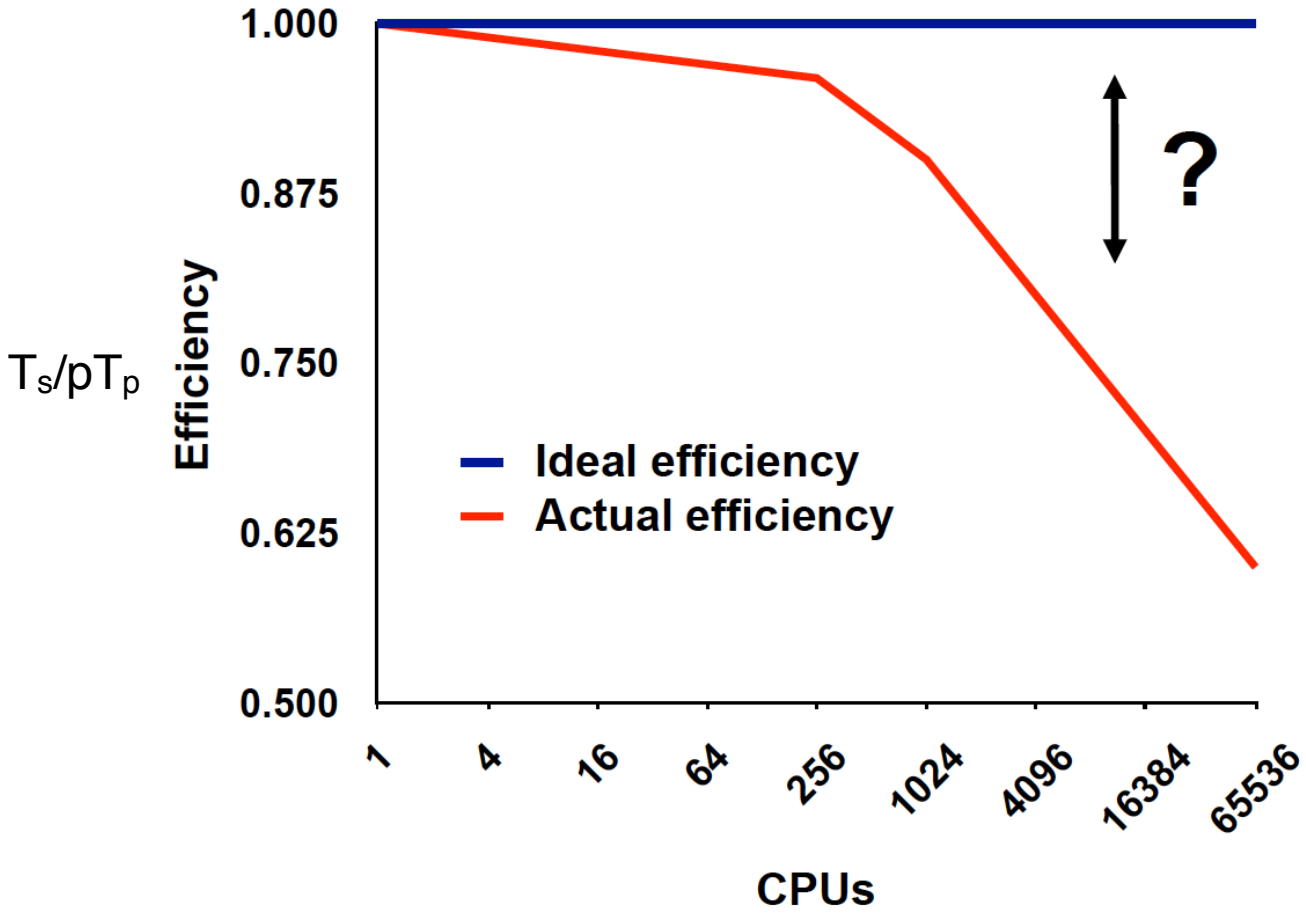
LULESH on Platform of 8 NUMA Domains



Beyond Data Collection and Attribution

- Published work
  - HPCToolkit-NUMA: analyzing NUMA bottlenecks (PPoPP'14)
  - ArrayTool: guiding array regrouping for better locality (PACT'14)
  - ScaAnalyzer: identifying memory scaling issues (SC'15)
  - StructSlim: guiding structure splitting (CGO'16)
  - Cheetah: detecting false sharing (CGO'16)
  - SMTAnalyzer: identifying SMT-aware optimization (HPDC'16)

The Problem of Scaling



Note: higher is better

Quantifying Scaling Loss

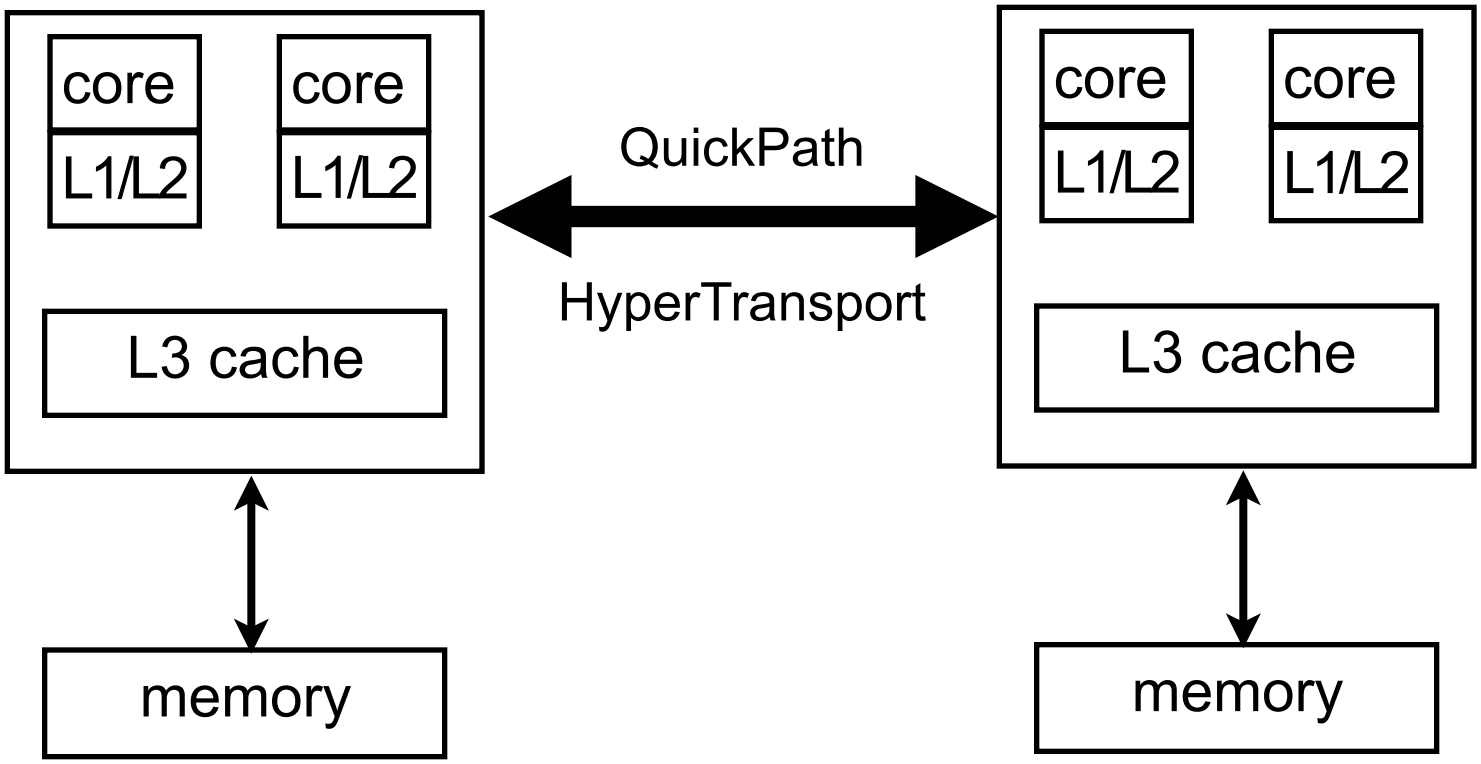
- Quantifying scaling loss with execution time (parallel efficiency)
  - it is straightforward with expectation: the same program with the same workloads

$$T_{2n} = T_n/2$$

    - $T_{2n} = T_n/2$: perfect scaling — expected
    - $T_{2n} > T_n/2$: scaling with loss
    - $T_{2n} < T_n/2$: superlinear scaling

  - execution time cannot help us focus on memory scaling
    - can we have an expectation for the memory performance for scaling?

Memory Scaling Expectation



The average latency of memory accesses should be at least the same with perfect scaling

Derived Metrics

- Average latency l_p over all memory accesses on p cores

  expectation: **scaling factor** = $l_p / l_q = 1\ (p > q)$

  - if $l_p / l_q \gg 1$
    - there is significant scaling loss in memory
  - which memory layer has the most memory scaling loss
  - which part of the source code has the memory scaling loss

Root Cause Analysis for Memory Scaling

- Which memory layer causes the scaling loss?
  - private layers: L1, L2 caches
  - shared layers: L3 cache and main memory
  - NUMA layers: remote L3 cache and remote memory

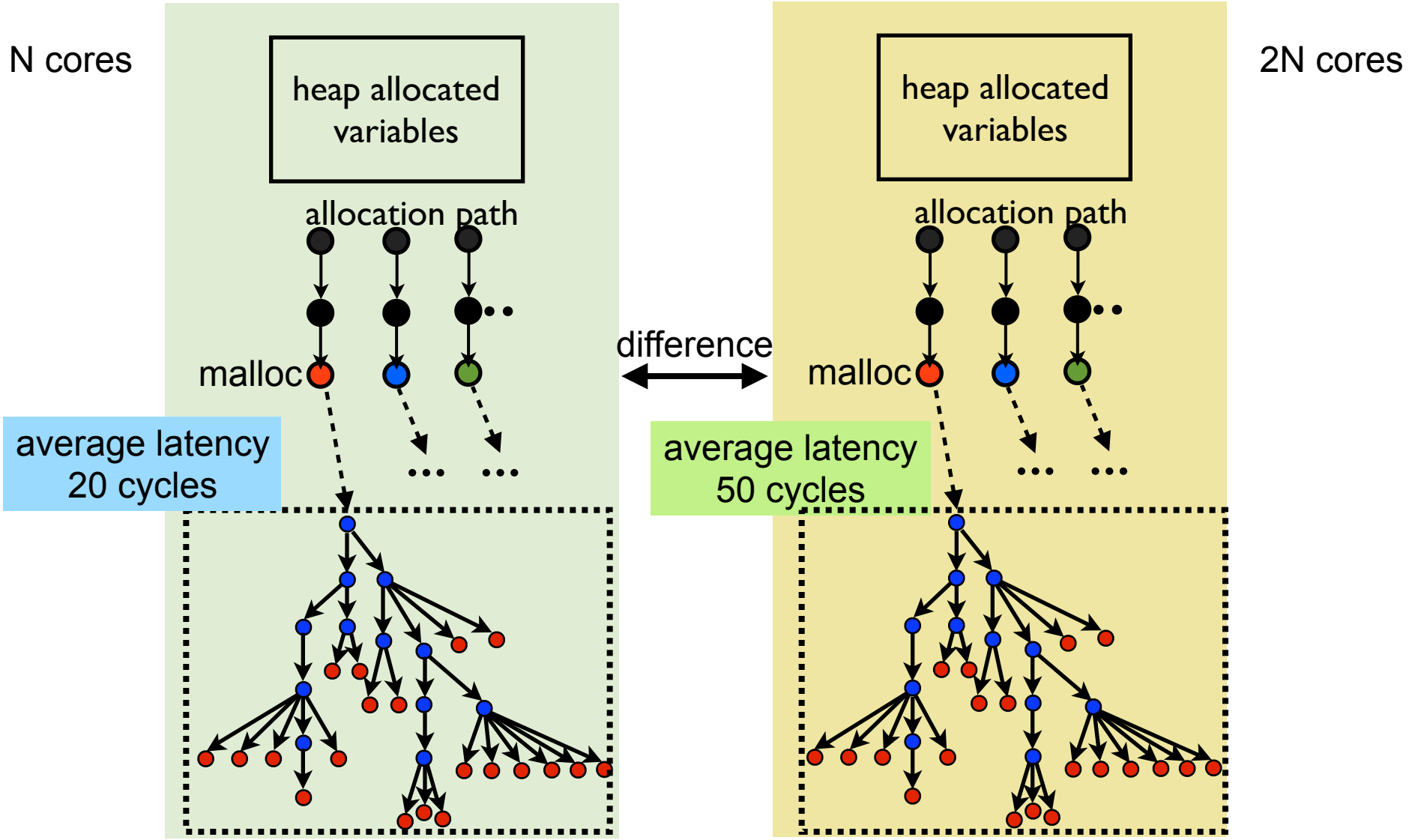
Provide optimization guidance: eliminating false sharing, mitigating contention, or addressing NUMA bottlenecks

- Which data objects and memory accesses cause the scaling loss?
  - problematic arrays with problematic accesses

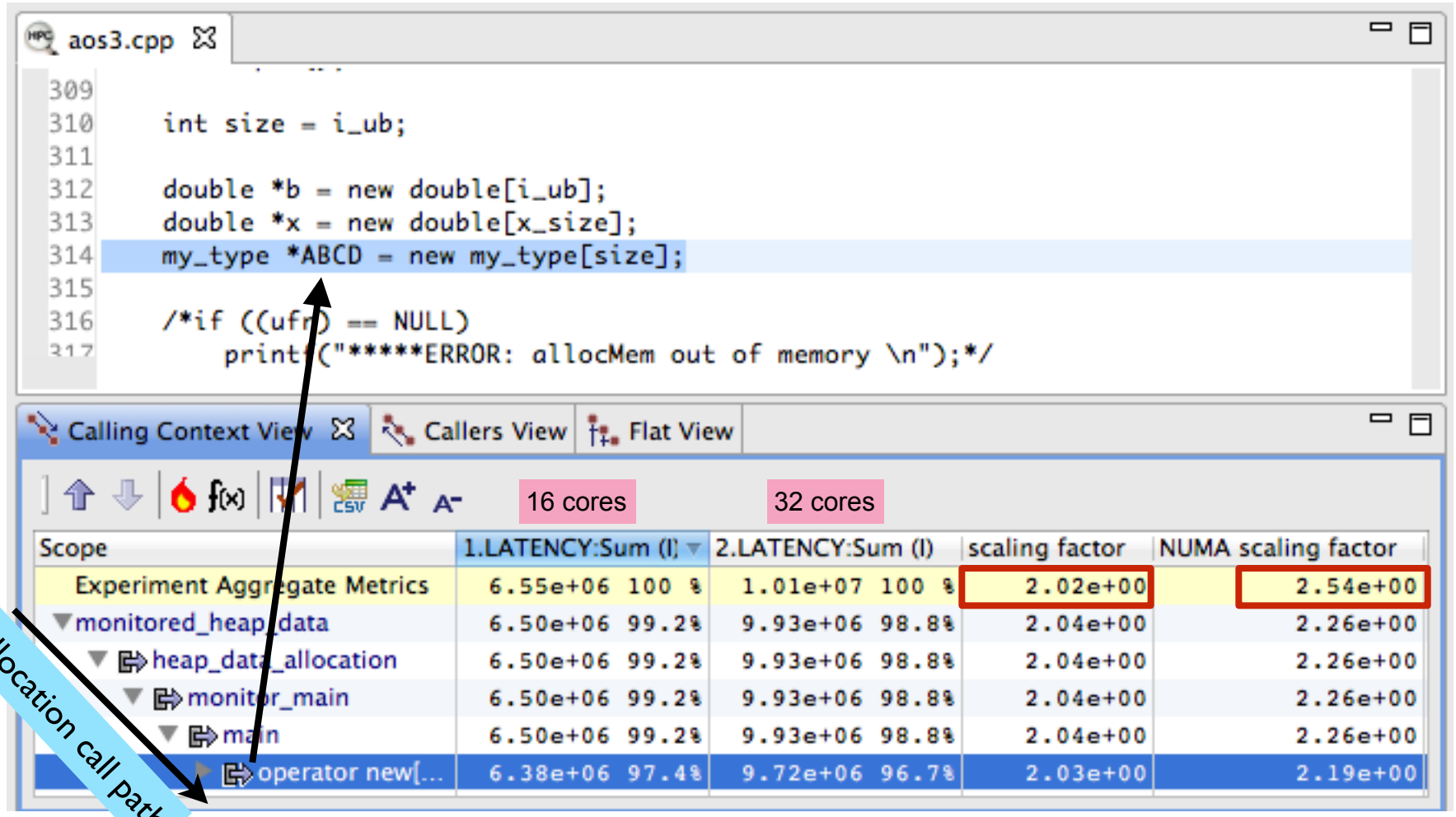
Pinpointing problematic source code for optimization: high-level feedback for programmers

ScaAnalyzer extends HPCToolkit: lightweight analysis

Differential Analysis on Aggregate Profiles



IRSmk: An Important DOE Benchmark



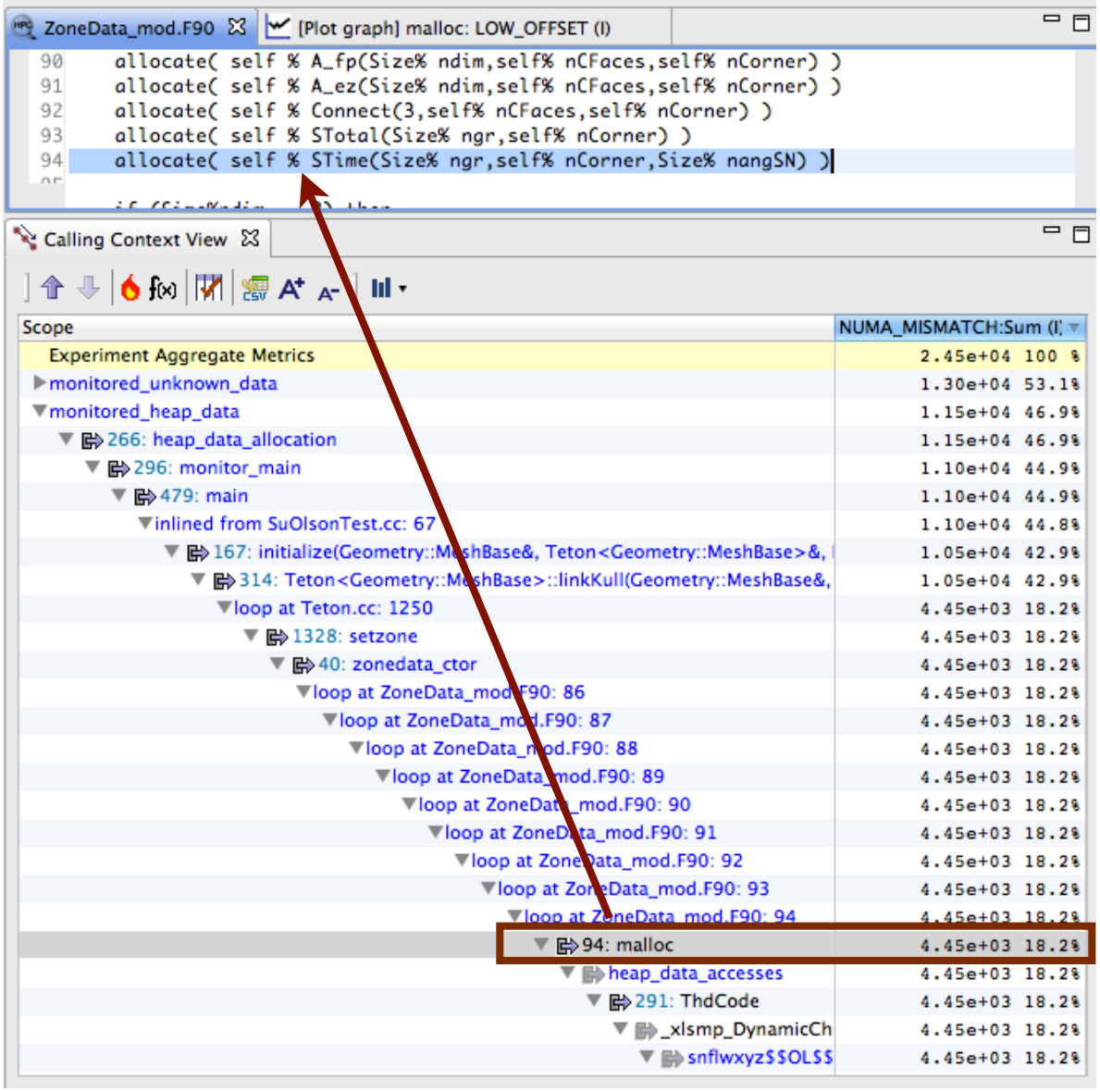
NUMA layer impedes the scalability

Conclusions and Future Work

- Hardware address sampling
  - widely supported in modern architectures
  - powerful in monitoring memory behaviors
  - further analysis of the samples provides deeper performance insights
- On-going work
  - automatic page migration for NUMA architectures
  - program optimization for heterogenous memories
    - with memif support (goto memif talk at ASPLOS)
- Future work: profilers for Big Data workloads
  - understand memory contention, memory usage, and SMT effects
  - multiple programs co-running
  - go beyond code-centric and data-centric analysis

Backup Slides

UMT2013 on Quad-socket POWER7 Node



sample off-chip accesses

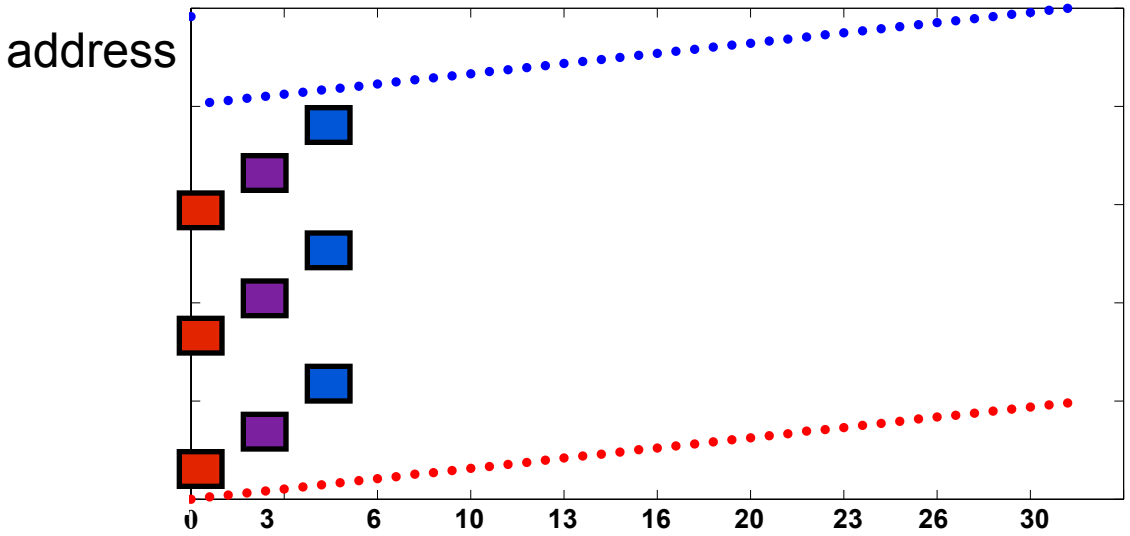
self%STime

18.2% of remote accesses

allocated in one domain accessed by everyone

Optimize *self%STime* for UMT2013

address-centric analysis for *self%STime*



self%STime's address space

optimization: let each thread initialize its own data

result: all threads have data locally -- 7% faster