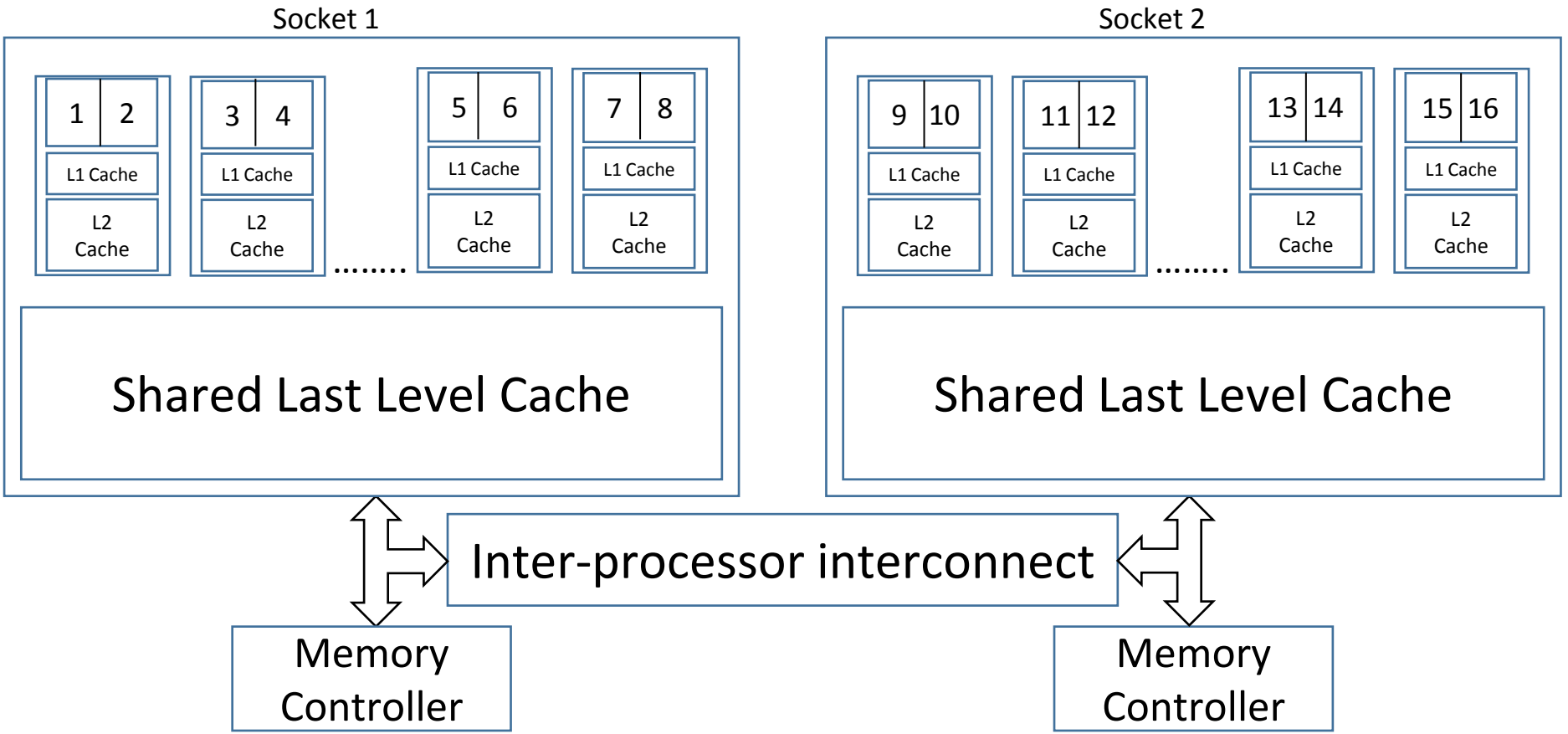
Coherence Stalls or Latency Tolerance: Informed CPU Scheduling for Socket and Core Sharing

Sharanyan Srikanthan
Sandhya Dwarkadas
Kai Shen

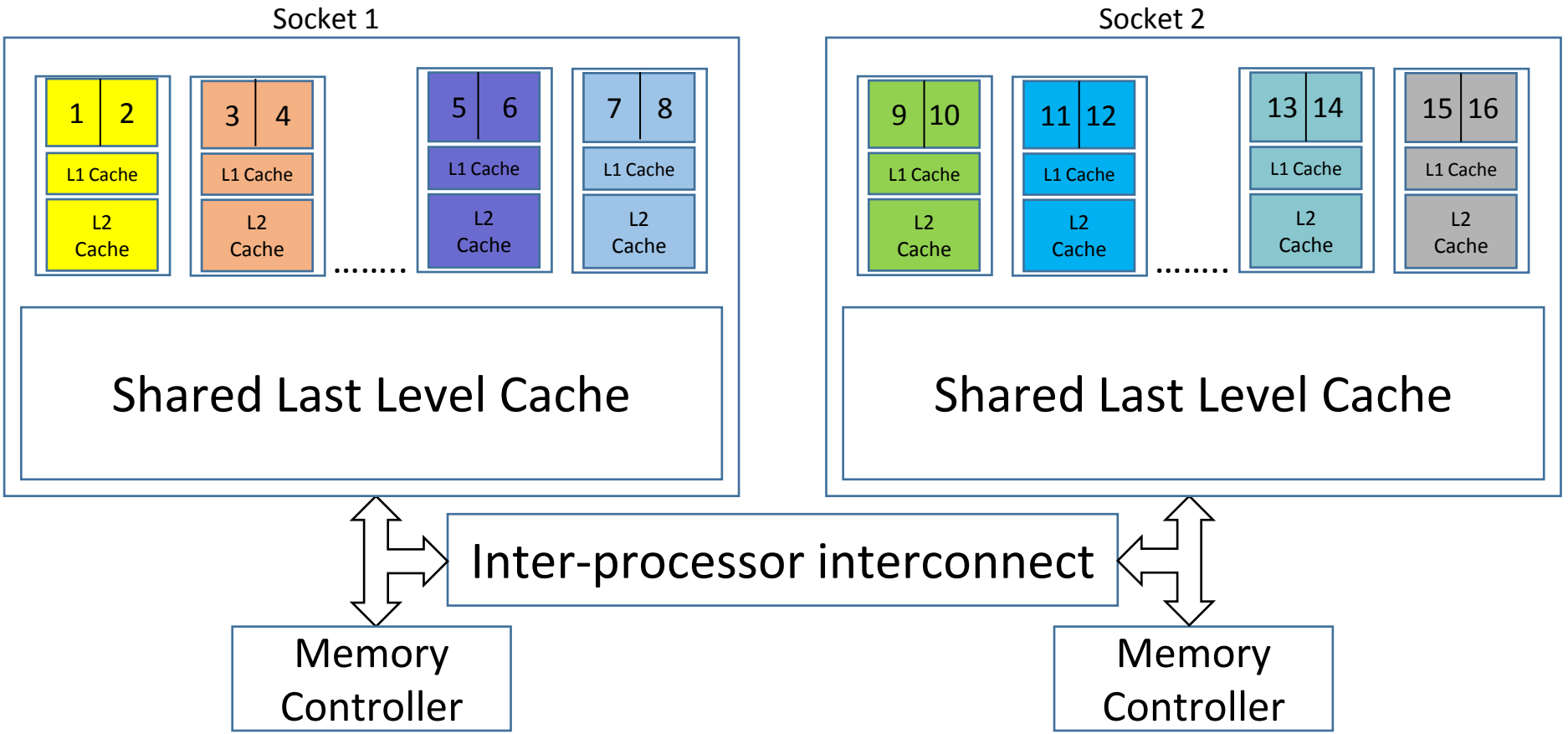
Department of Computer Science
University of Rochester

Performance Transparency Challenge: Modern Multi-core Machines



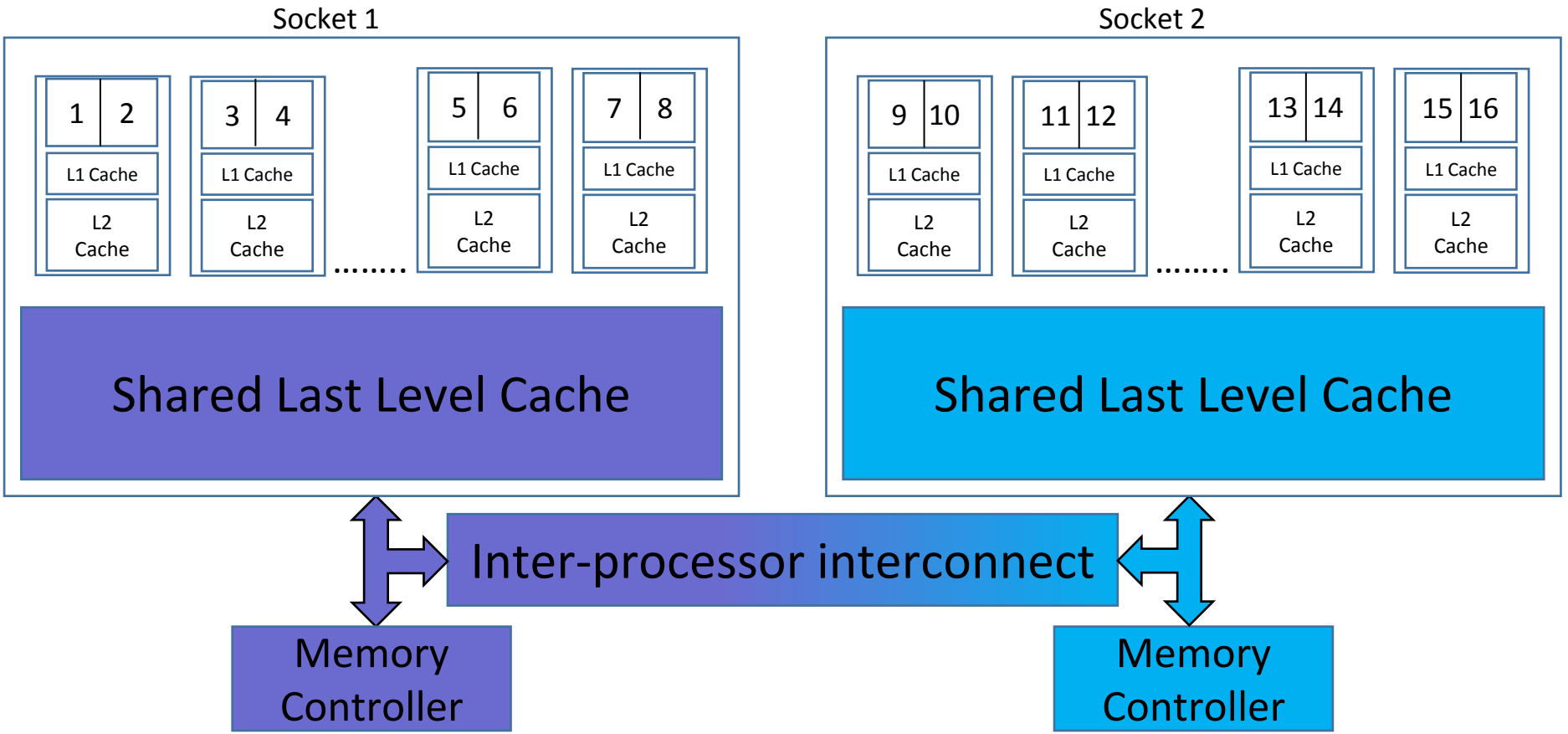
Performance Transparency Challenge: Resource Sharing

Problem: Simultaneous multi-threading



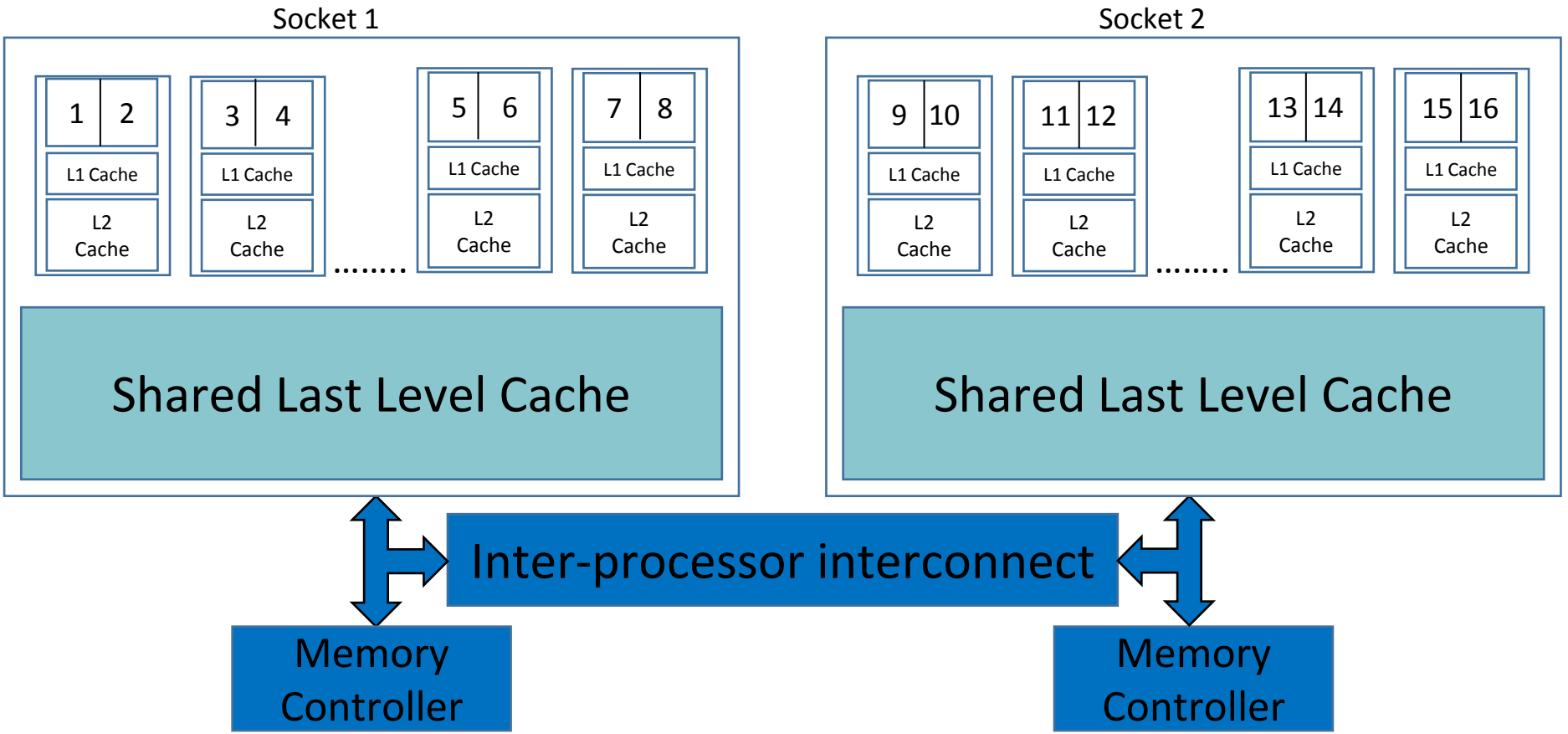
Performance Transparency Challenge: Resource Sharing

Problem: Intra-processor resource sharing



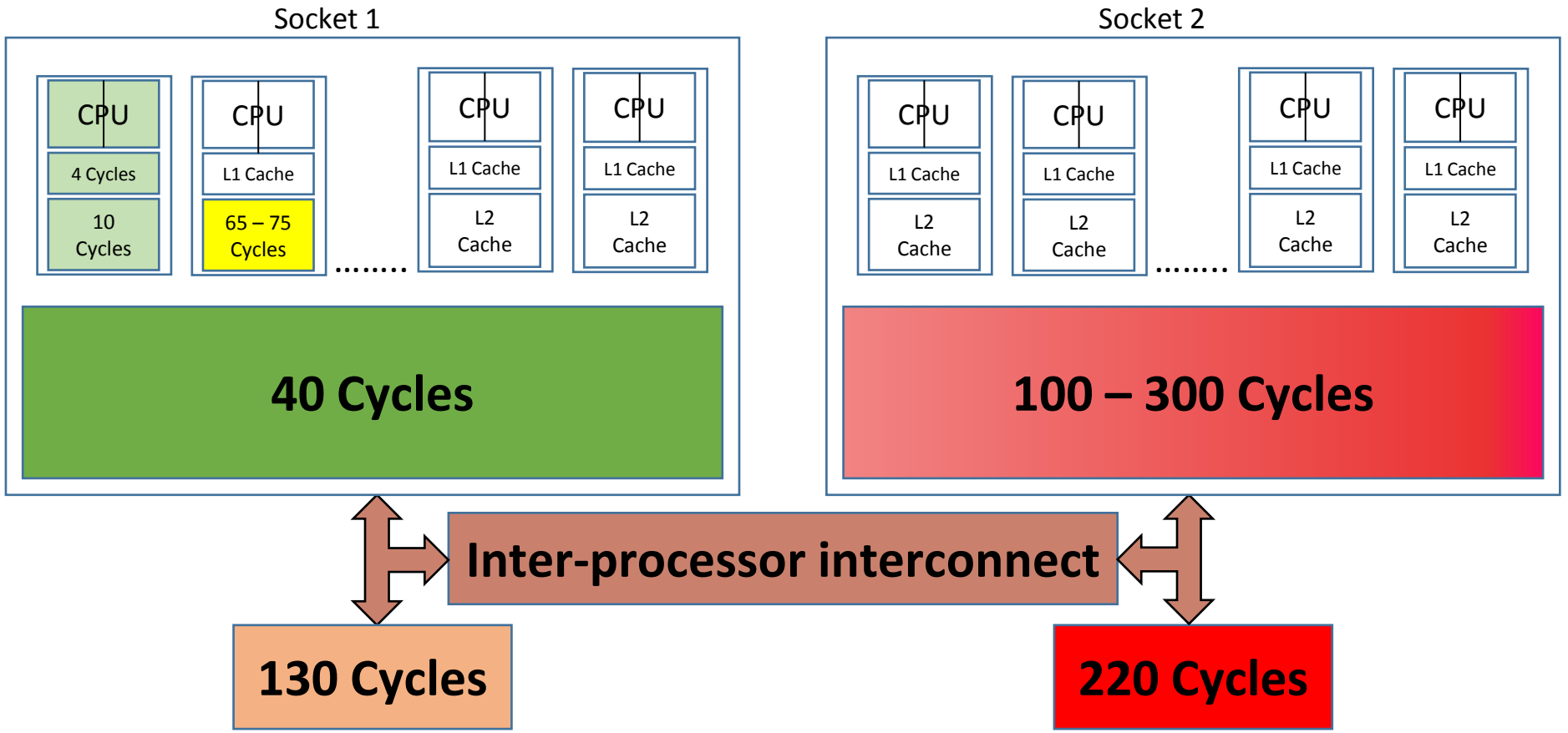
Performance Transparency Challenge: Resource Sharing

Problem: Inter-processor resource sharing



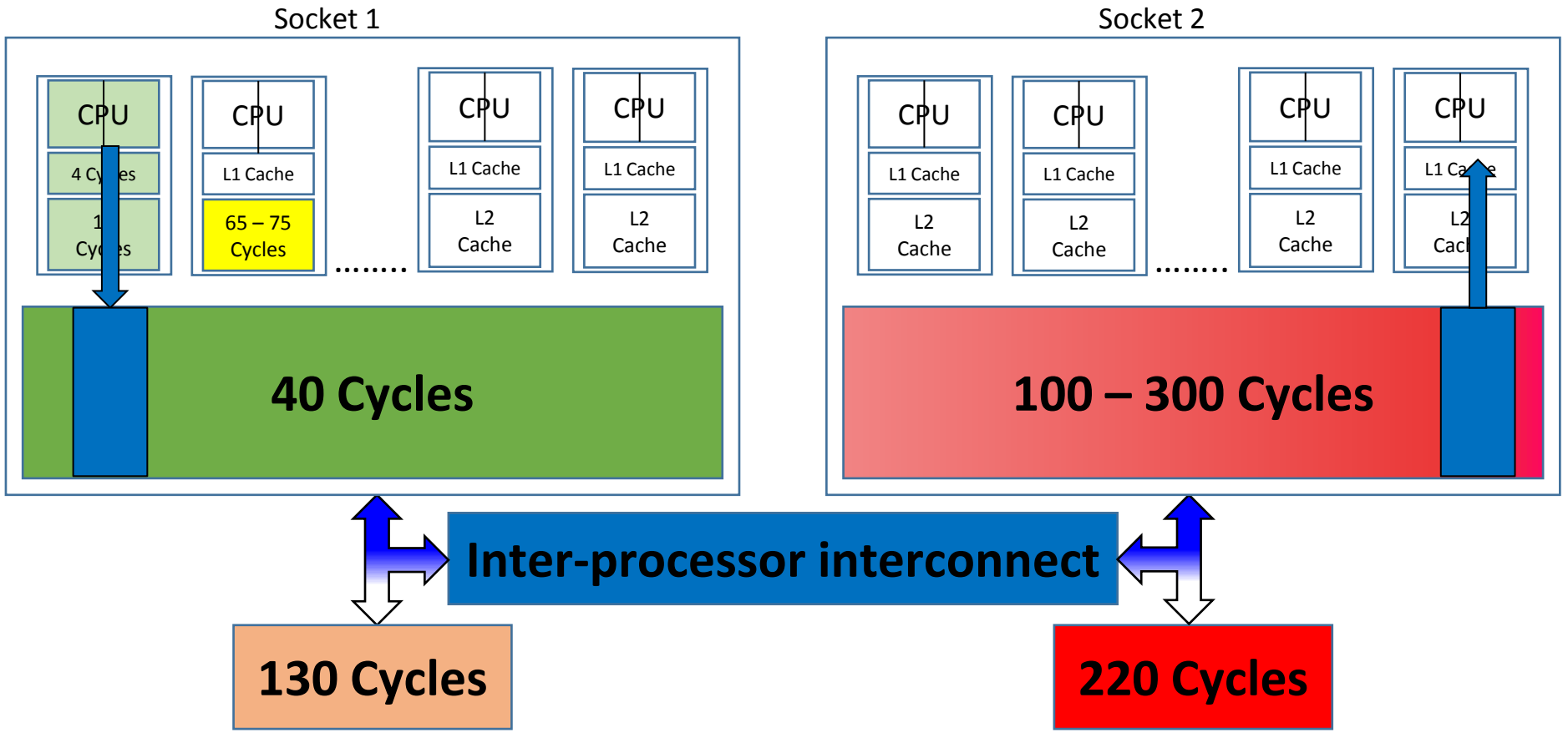
Performance Transparency Challenge: Non-Uniform Access Latencies

Problem: Communication costs a function of thread/data placement



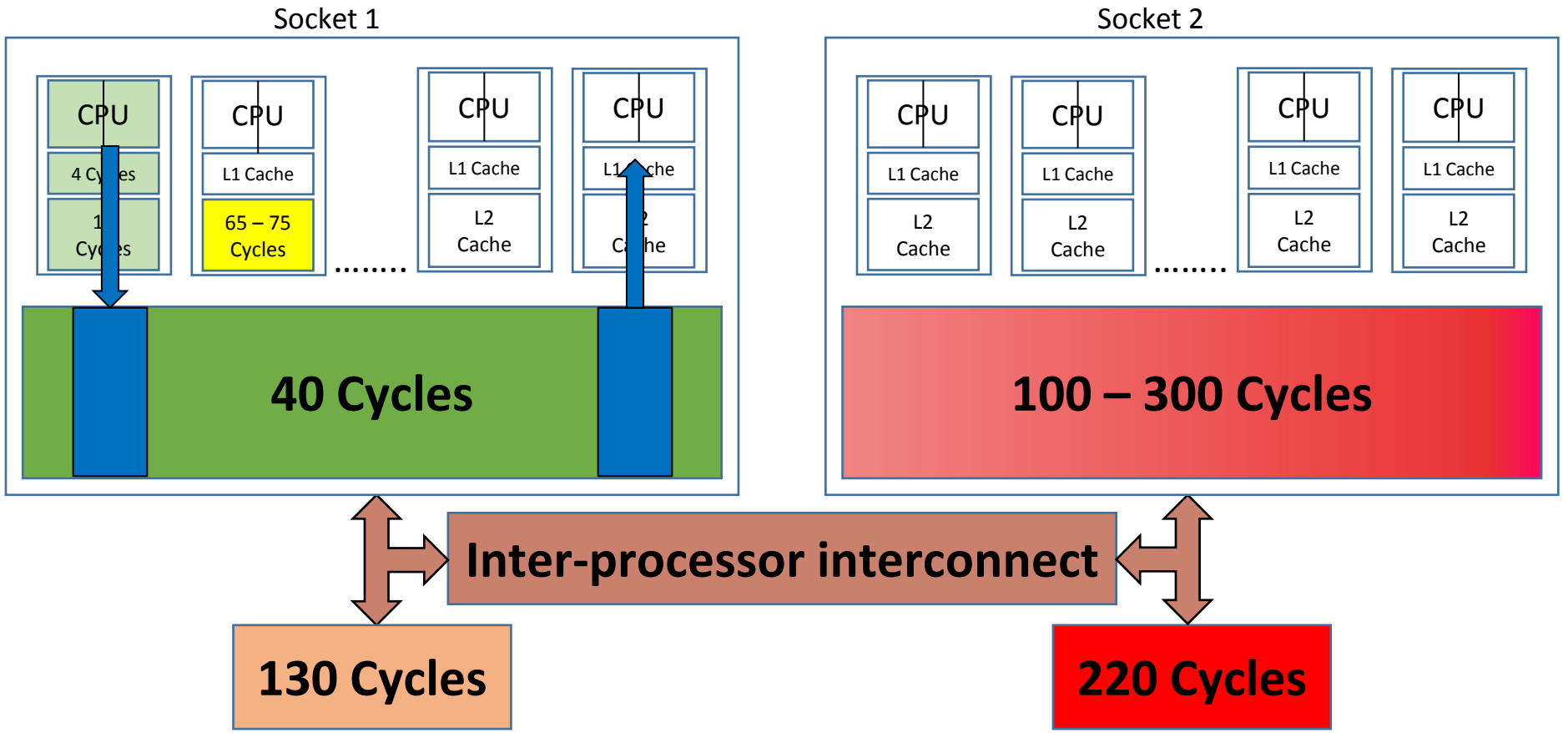
Ref: https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf

Impact of Thread Placement on Data Sharing Costs



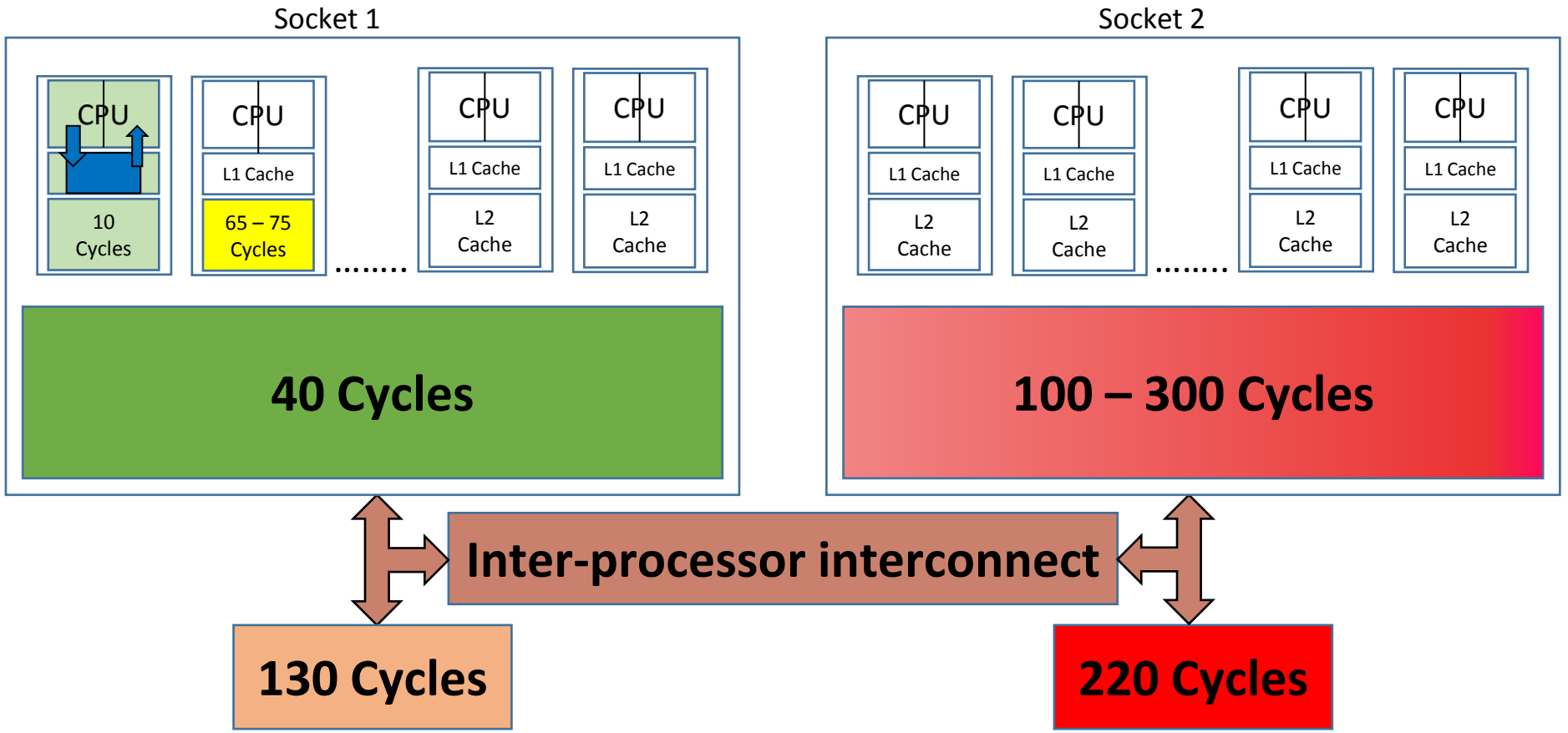
Ref: https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf

Impact of Thread Placement on Data Sharing Costs



Ref: https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf

Impact of Thread Placement on Data Sharing Costs



Ref: https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf

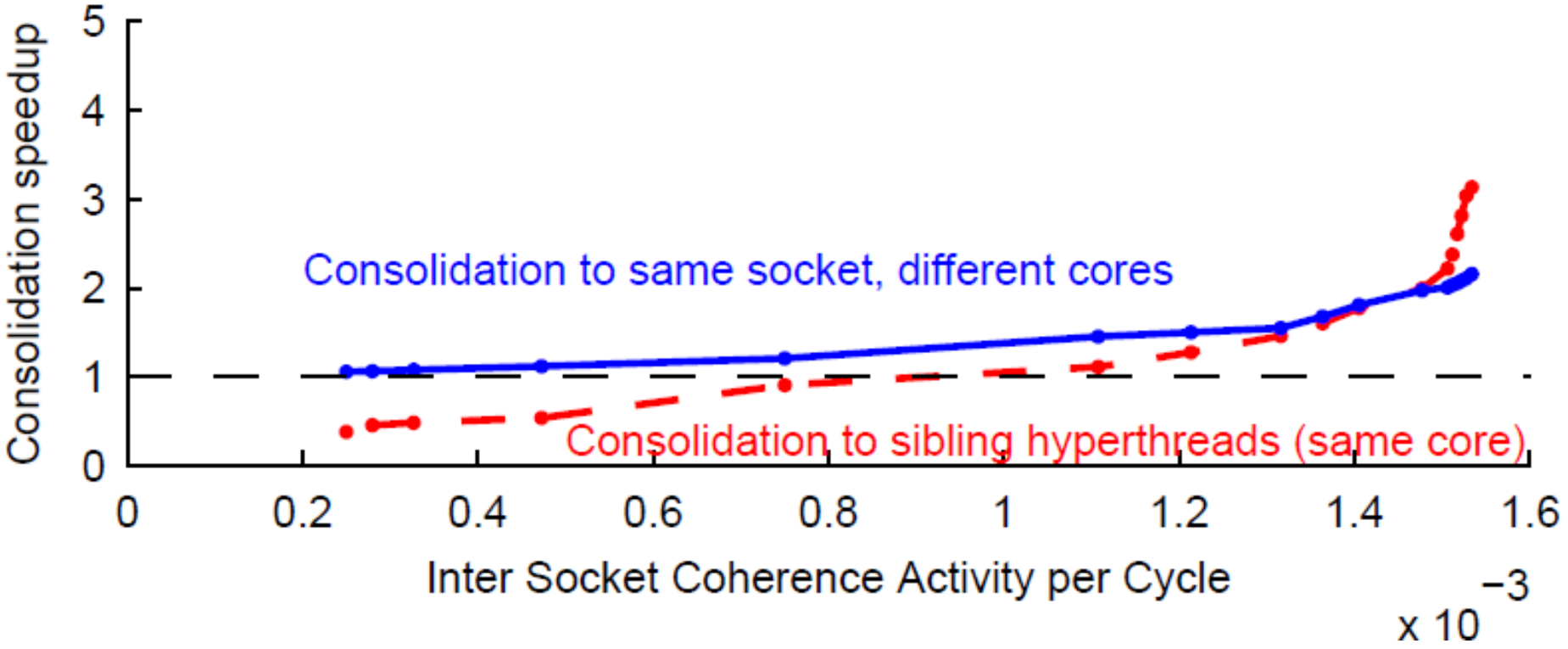
Sharing Aware Mapper (SAM)

- Srikanthan et al. [USENIX ATC 2015]
- Uses low overhead performance counters to identify data sharing, resource demand
- Map processes to CPUs to minimize
  - Communication cost due to data sharing
  - Resource contention
  - Memory access latency

Sharing Aware Mapper (SAM-MPH)

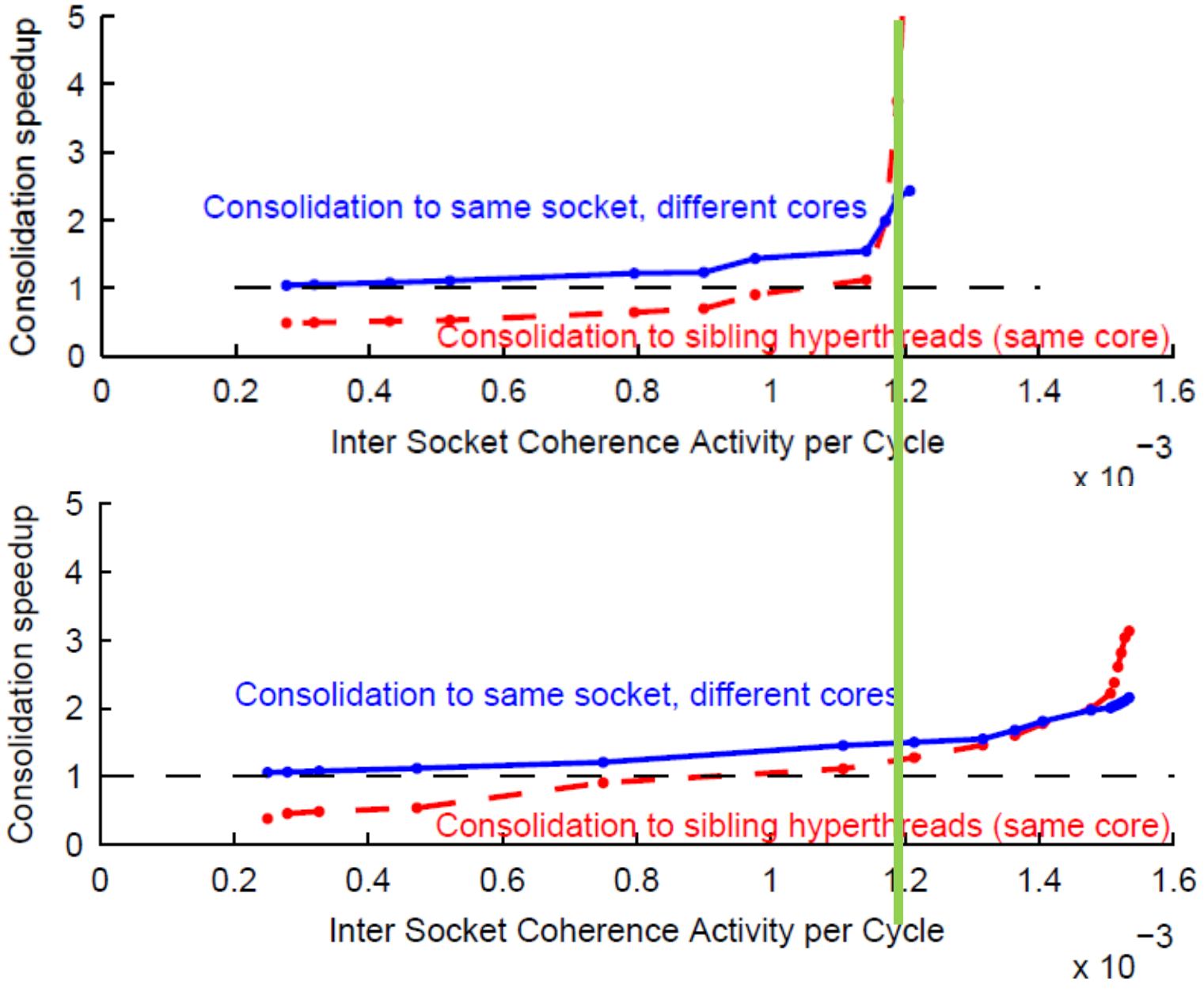
- Remaining challenges:
  - Understand impact of coherence: Execution stalls or latency tolerance
  - Analyze impact of hyper-threading
- Our approach:
  - M: Metrics to identify and use latency tolerance for data sharing cost to prioritization
  - P: Phase detection adds hysteresis to recognize and avoid reacting to transient phases
  - H: Hyper-threading related cost/benefits

Importance of Task Placement

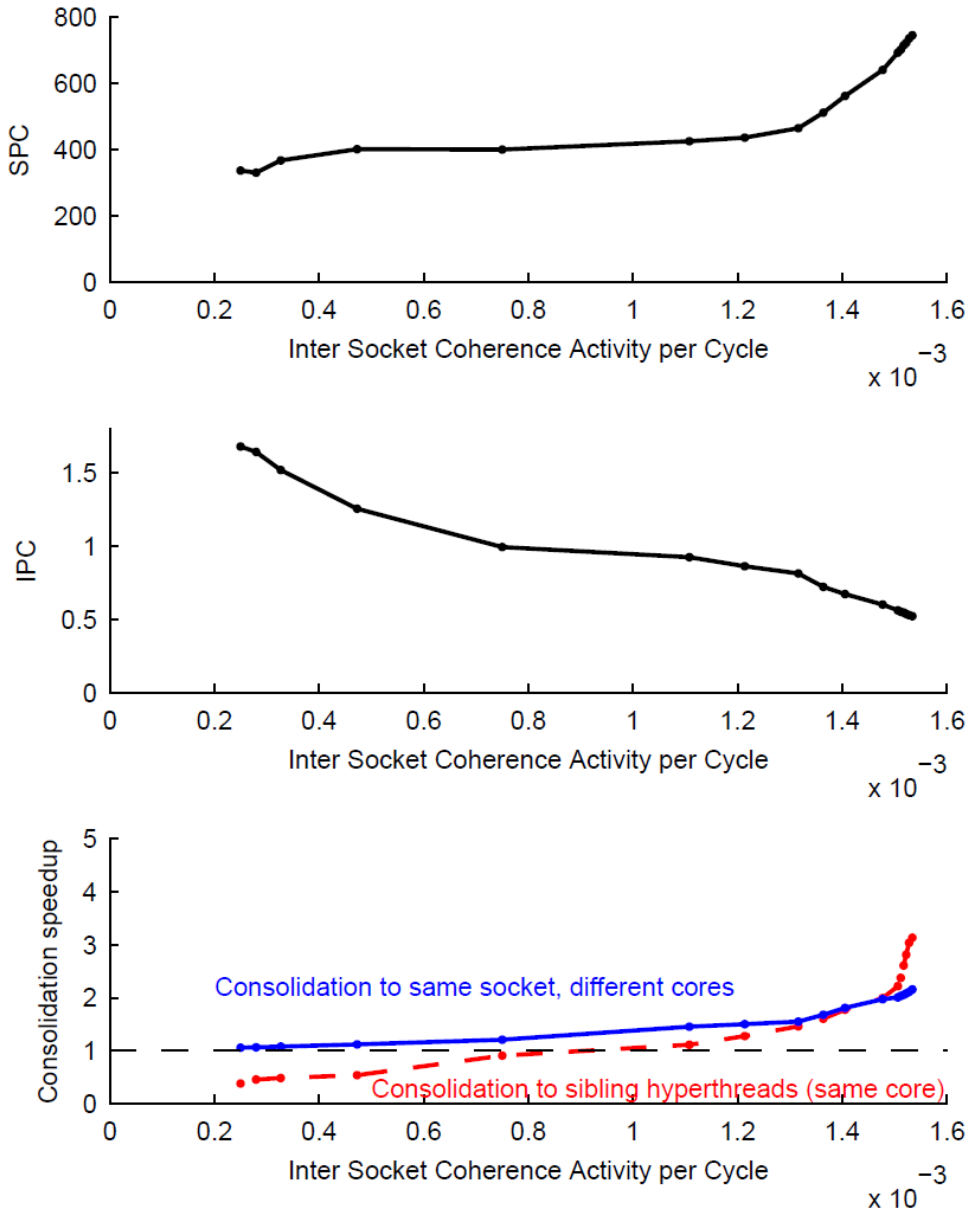


- Micro-benchmark forces data to move from one task to the other, generating coherence activity
- Rate of coherence activity varied by varying ratio of private to shared variable access

Prioritizing Applications – Coherence Activity?



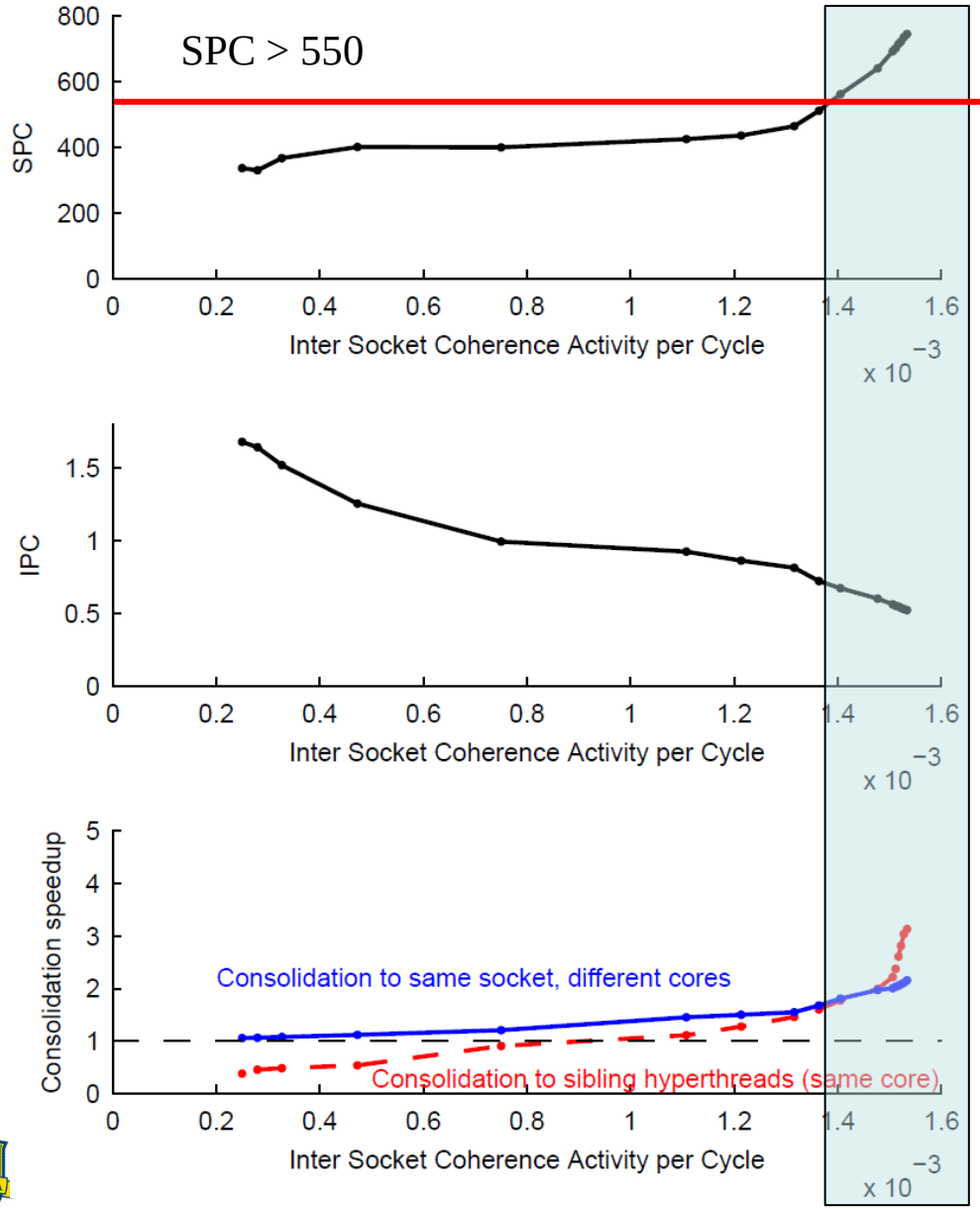
Metrics: SPC and IPC

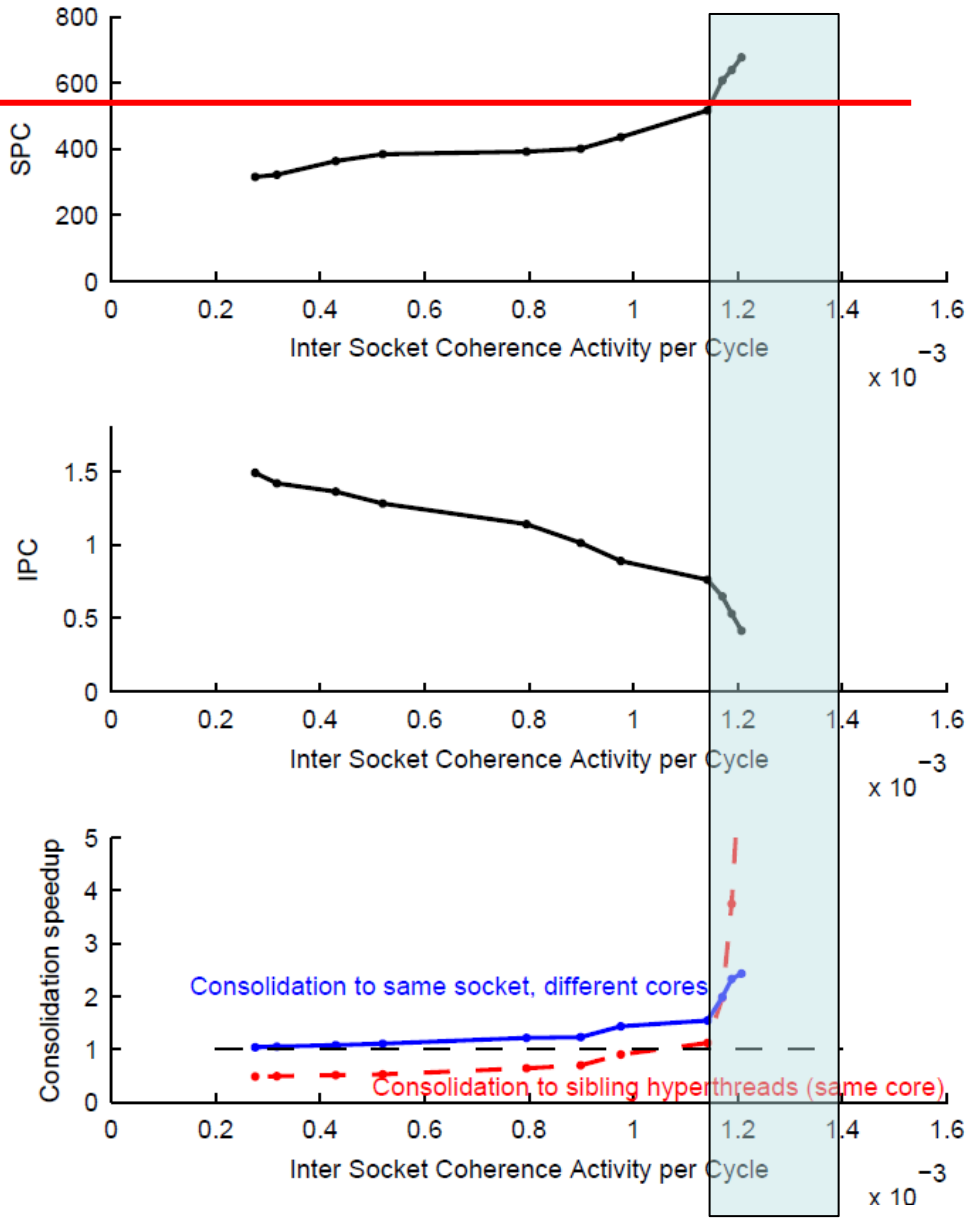


- SPC – Stalls per inter-socket coherence activity
- SPC helps identify latency hiding capability of application

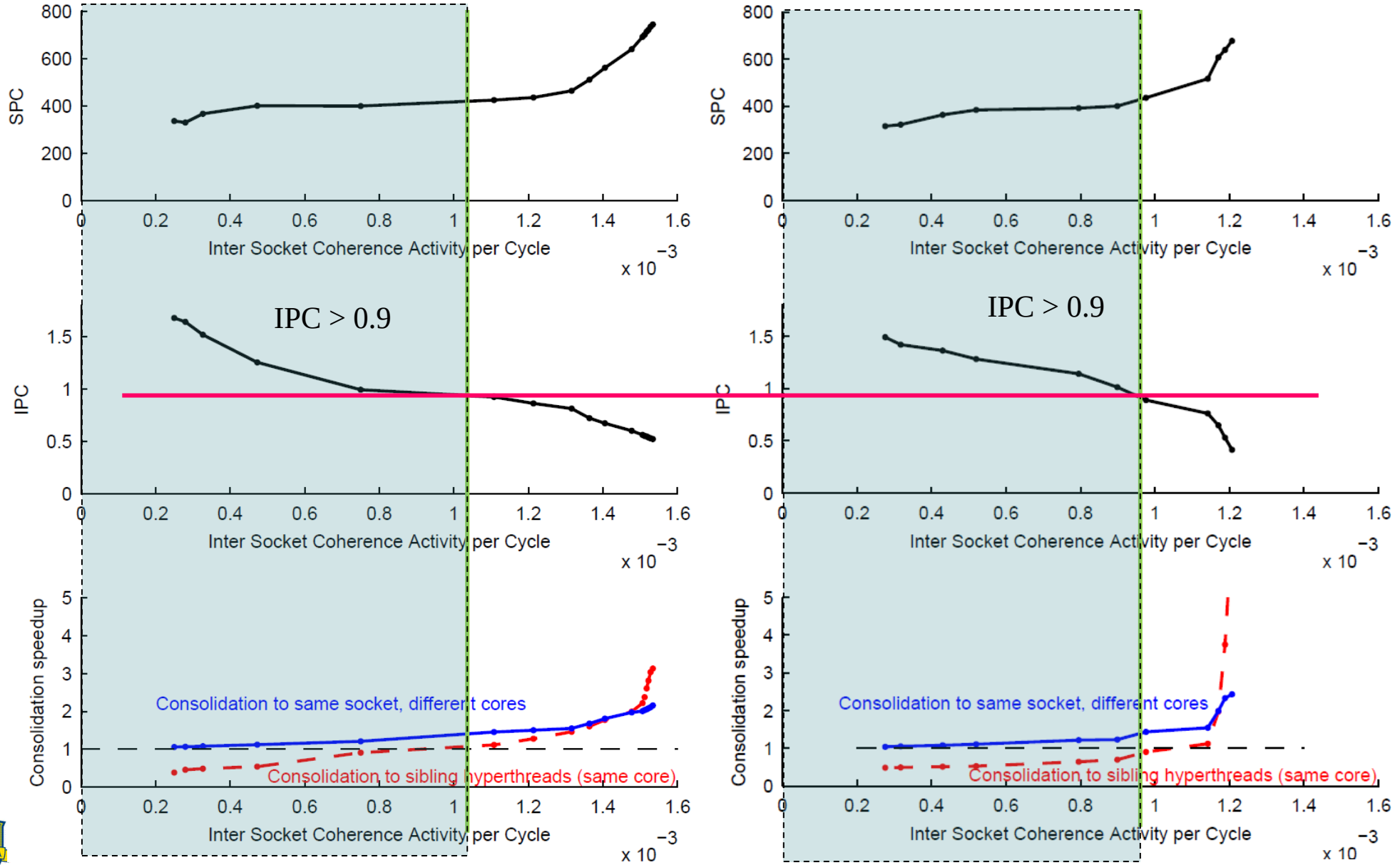
- IPC – Instructions per cycle
- IPC helps identify computational contention on processor core

Take Placement: Importance of SPC

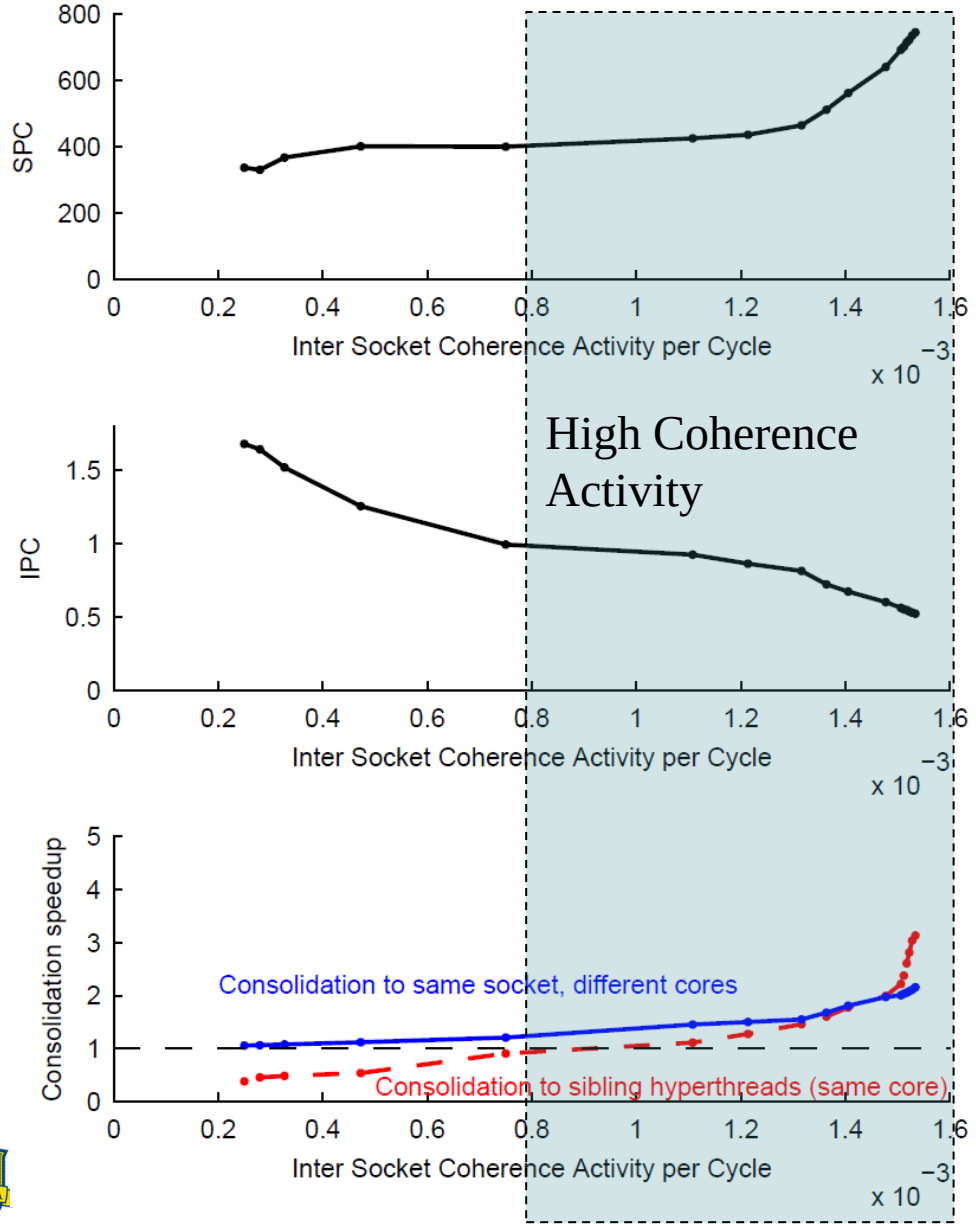




Task Placement: Importance of IPC

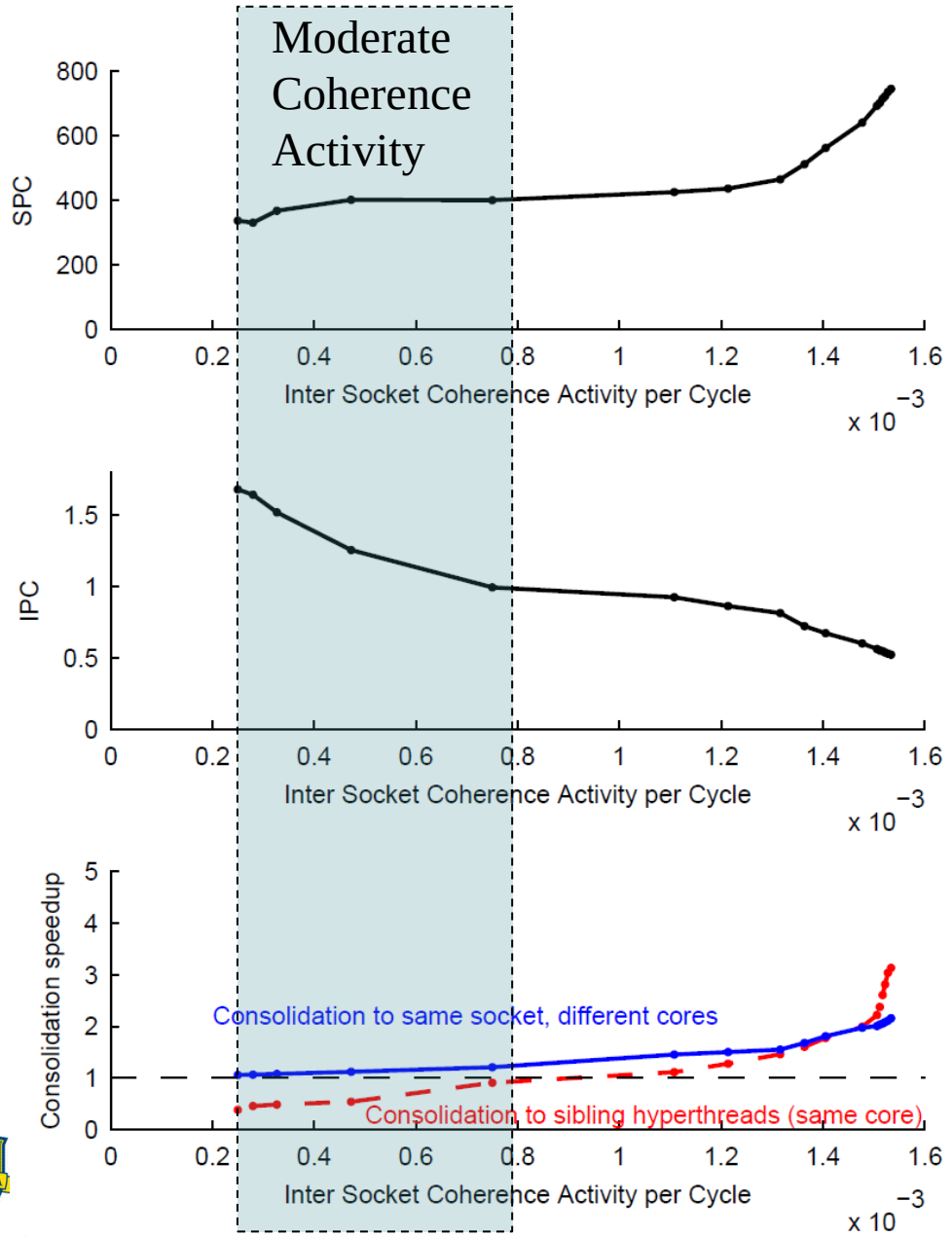


Prioritizing Applications: High Coherence Activity



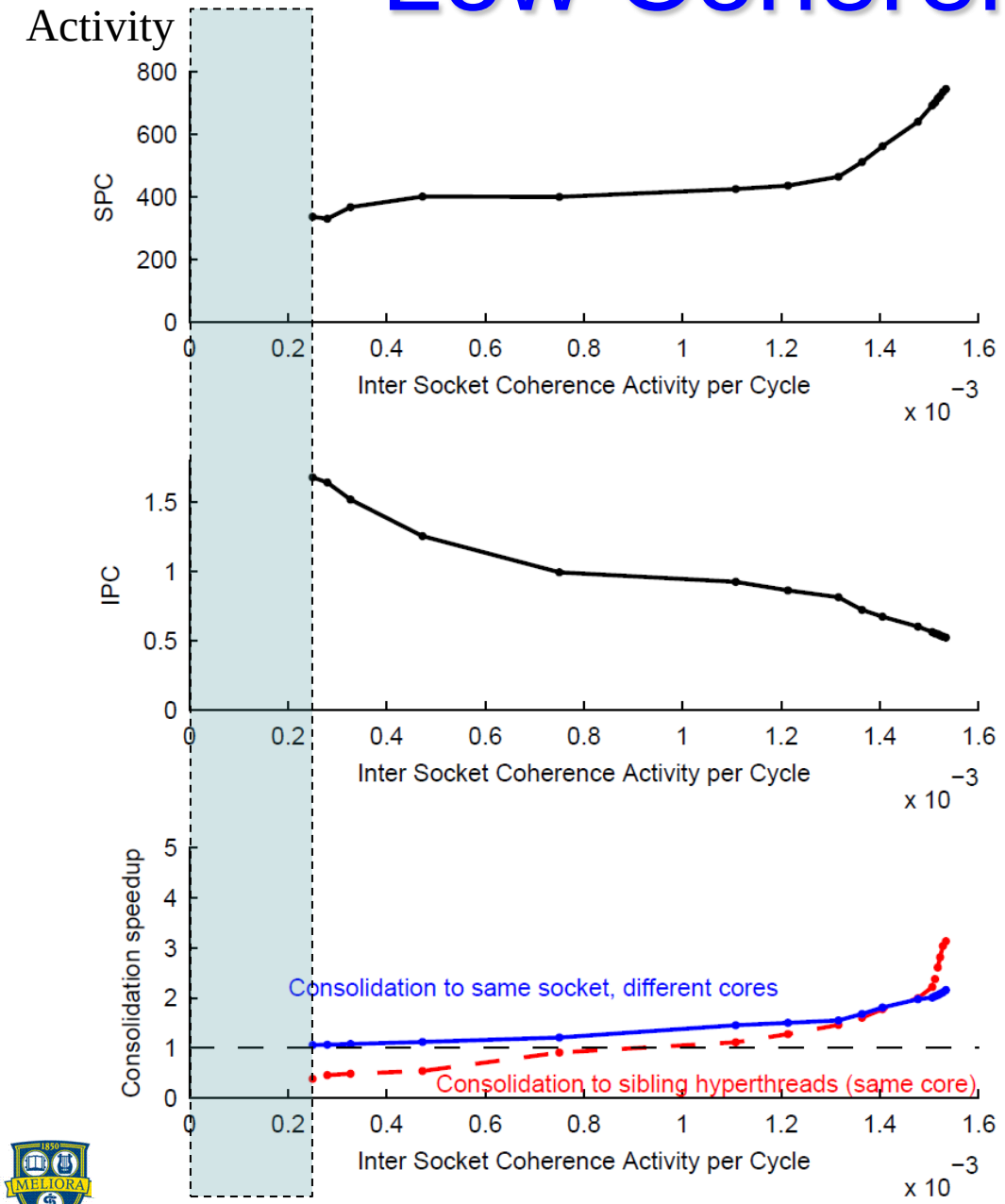
- ⇧ SPC, ⇧ Priority
- SPC ~ $\frac{\text{Stalls on cache accesses}}{\text{Inter-socket coherence activity}}$
- SPC > 550
  - Prioritize logical thread co-location
- IPC > 0.9
  - Avoid co-location on hyper-threads even if it results in distributing across sockets

Prioritizing Applications: Moderate Coherence Activity



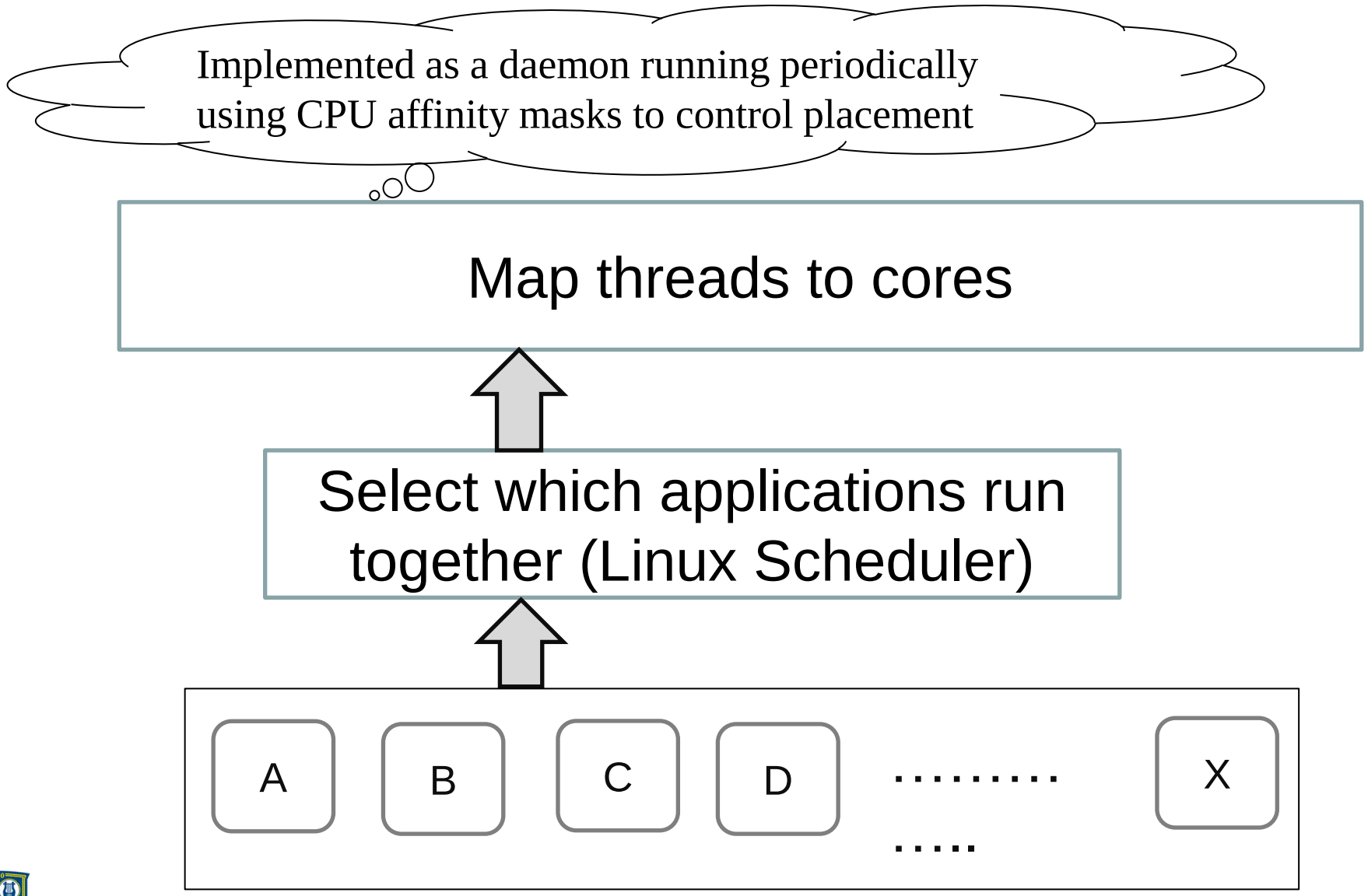
- ⇧ IPC, ⇩ Priority
- IPC > 0.9
  - Avoid co-location on hyper-threads even if it results in distributing across sockets

Prioritizing Applications: Low Coherence Activity



- No preference to be distributed across sockets or consolidated within a socket
- IPC > 0.9
  - Avoid co-locating on hyper-threads even if it results in distributing across sockets

Implementation Context



Monitoring Using Performance Counters

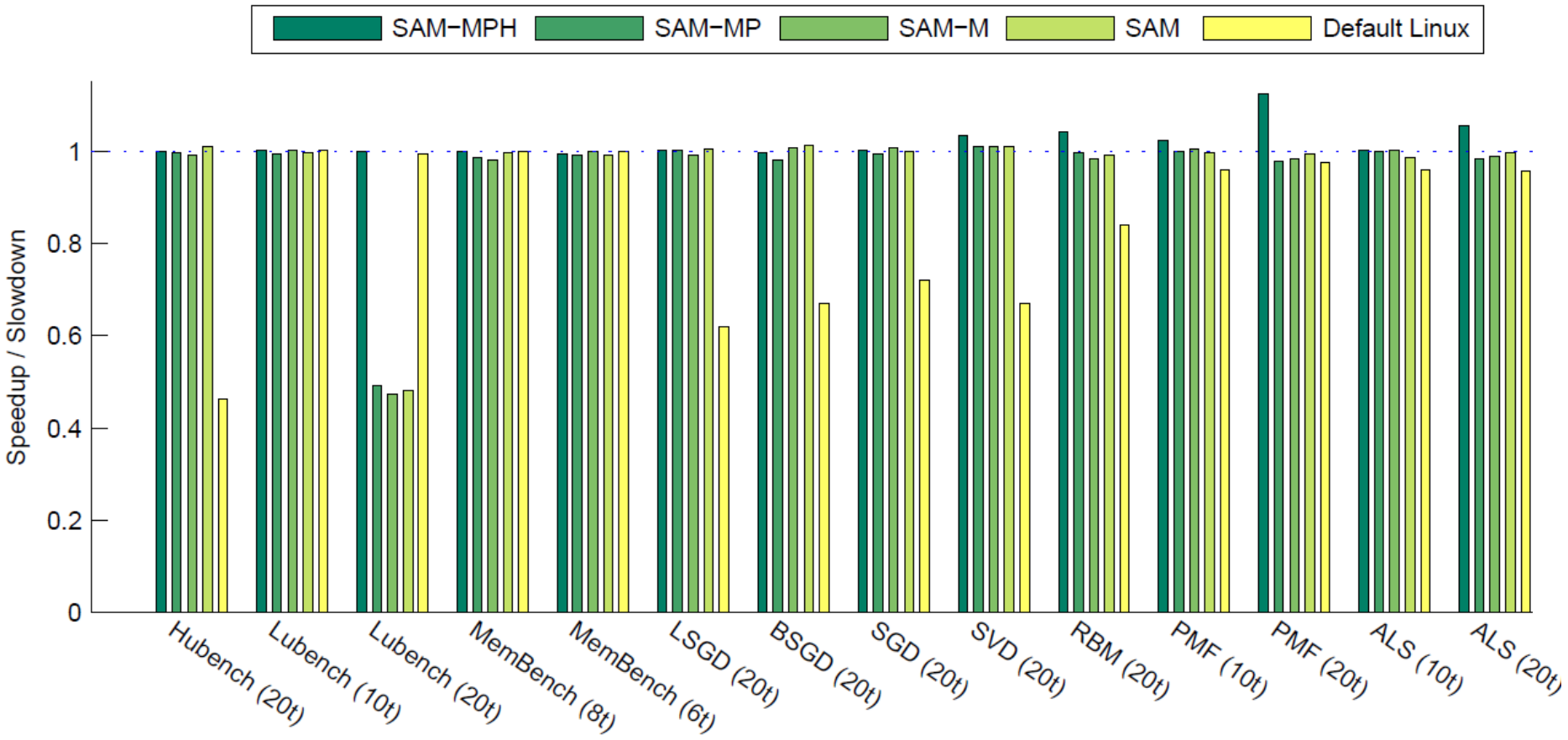
- 5 metrics identified: 8 counter events need to be monitored
  - Inter-socket coherence activity
    - Last level cache misses served by remote cache
  - Intra-socket coherence activity
    - Last private level cache misses – (sum of hits and misses in LLC)
  - Stalled cycles on coherence activity
  - Local Memory Accesses
    - Approximated by LLC misses
  - Remote Memory Accesses
- 4 hardware programmable counters available: requires multiplexing

Experimental Environment

- Fedora 19, Linux 3.14.8
- Dual socket machine – “IvyBridge” processor (40 logical cores, 2.20 GHz)
- Quad socket machine – “Haswell” processor (80 logical cores, 1.90 GHz)
- Benchmarks
  - Microbenchmarks
  - SPECCPU ’06 (CPU & Memory bound workloads)
  - PARSEC 3.0 (Parallel workloads – light on data sharing)
  - Machine learning and data mining algorithms like: ALS, Stochastic Gradient Descent, Single Value Decomposition, etc
  - Service Oriented – MongoDB

Standalone Applications

Dual Socket “IvyBridge”



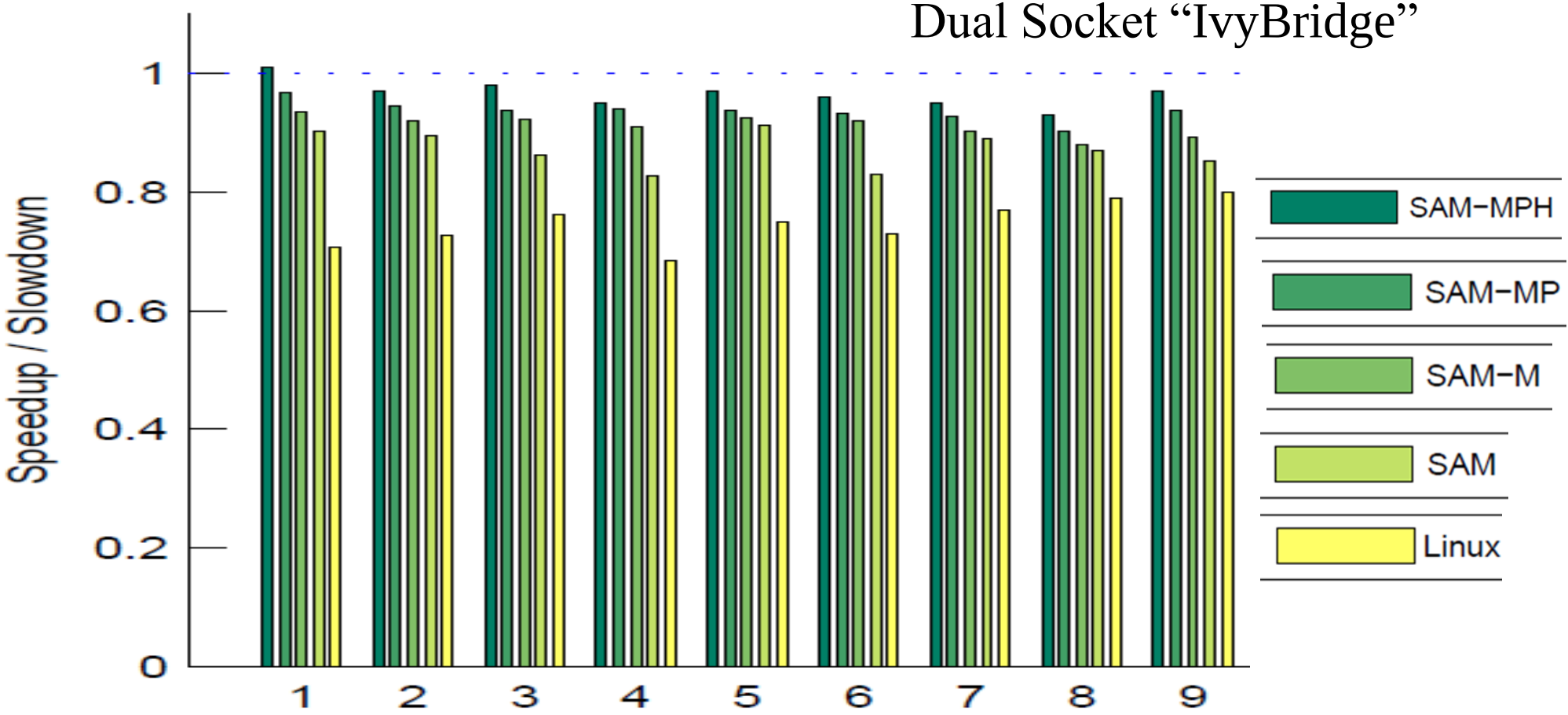
Baseline (Normalization factor): Best static mapping determined by exhaustive search

Improvement over

- Linux: Mean = 20%
- SAM: Mean = 6%

Multiple Applications

Dual Socket "IvyBridge"



Improvement over

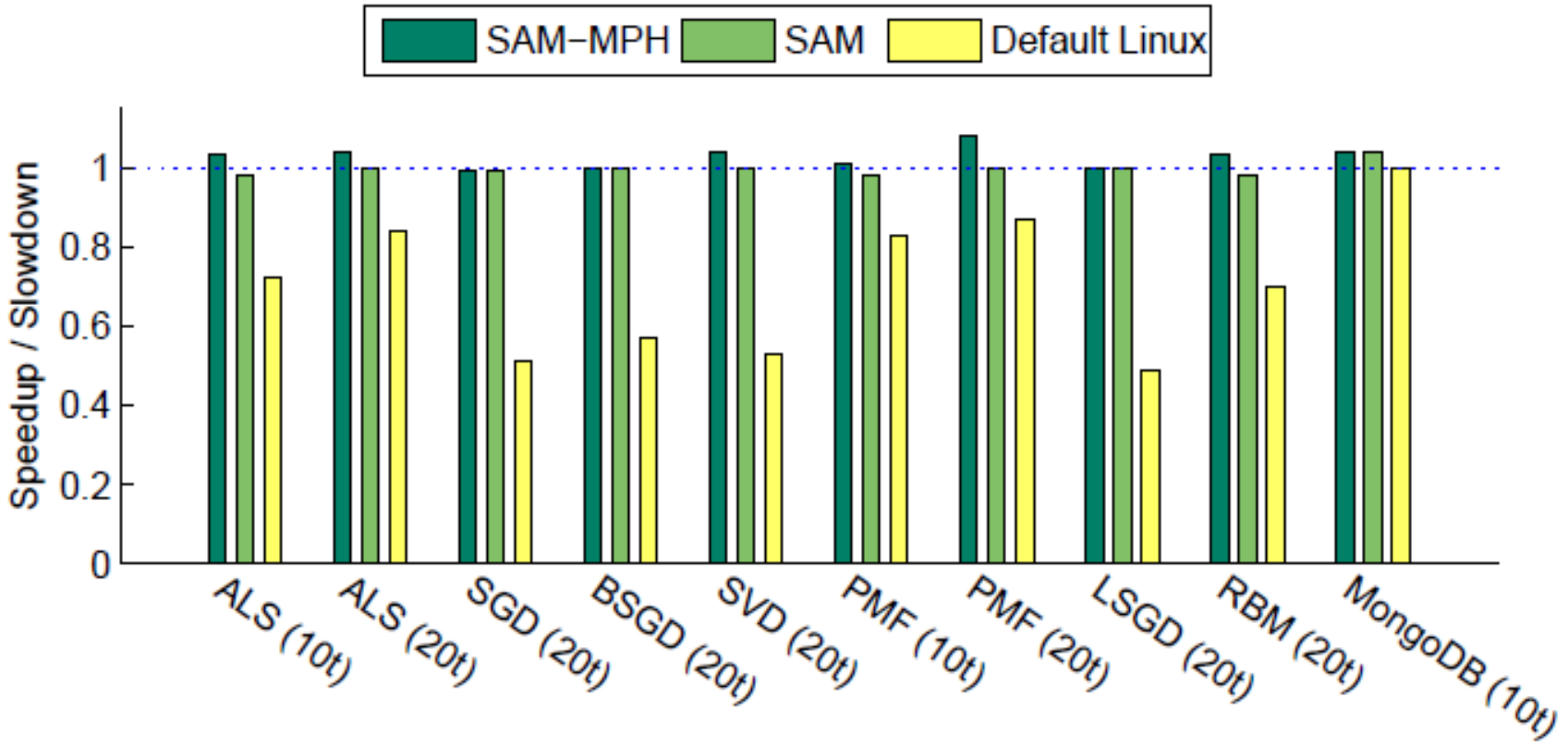
- Linux: Mean = 27% (Max: 43%)
- SAM: Mean = 9% (Max: 24%)

Improvement in fairness:

- Linux: Avg min speedup: 0.71, Avg max speedup: 0.84
- SAM: Avg min speedup: 0.86, Avg max speedup: 0.93
- SAM-MPH: Avg min speedup: 0.95, Avg max speedup: 1.0003

Standalone Applications

Quad Socket "Haswell"



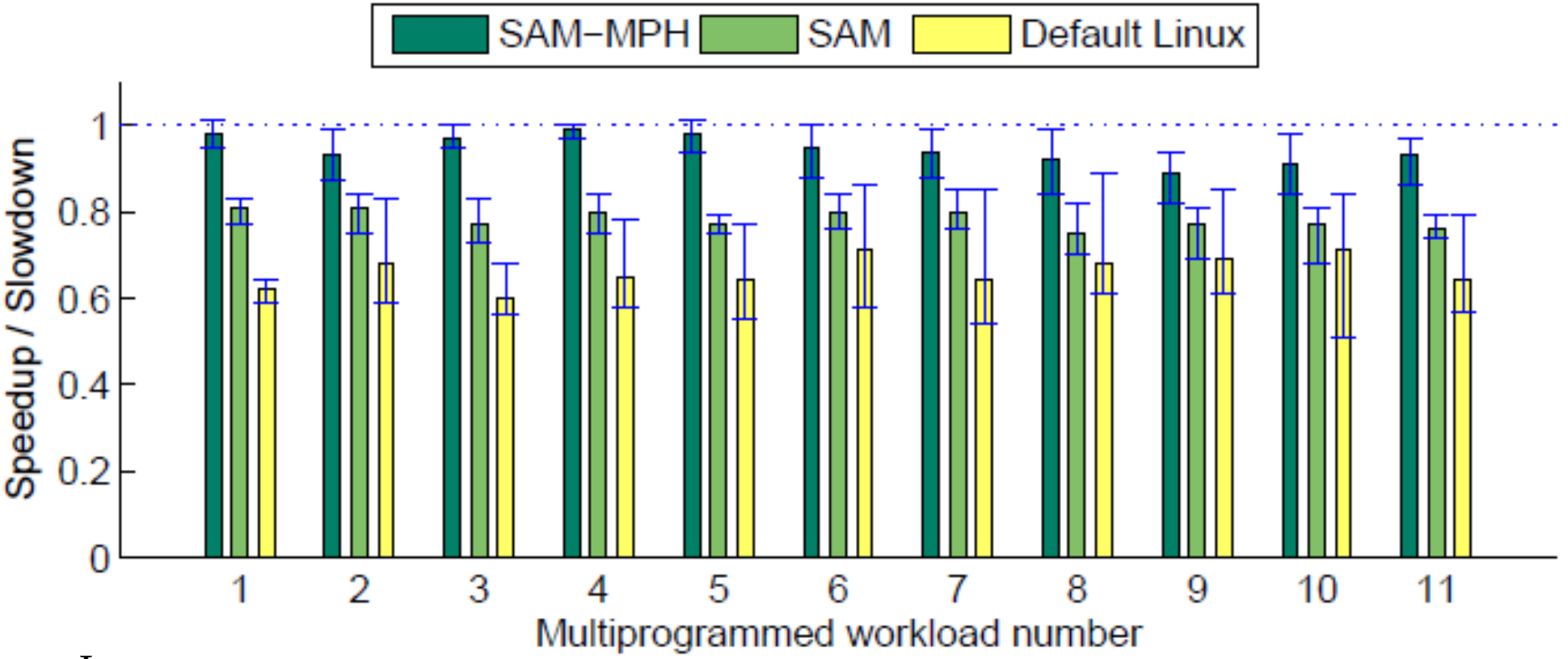
Baseline (Normalization factor): Best static mapping determined by exhaustive search

Improvement over

- Linux: Mean = 45%
- SAM: Mean = 3%

Multiple Applications

Quad Socket "Haswell"



Improvement over

- Linux: Mean = 43% (Max: 61%)
- SAM: Mean = 21% (Max: 27%)

Improvement in fairness:

- Linux: Avg min speedup: 0.57, Avg max speedup: 0.79
- SAM: Avg min speedup: 0.73, Avg max speedup: 0.82
- SAM-MPH: Avg min speedup: 0.89, Avg max speedup: 0.99

Overheads and Scaling

- Overall overhead (40 cores)
  - Performance counter reading
    - Invoked every tick – 1msec – 8.9 µs per tick
    - Constant time overhead
- Data consolidation and decision making is centralized
  - Data consolidation
    - Demon invoked every 100ms
    - SAM-MPH: 230us (worst case), 14us (best case)
    - SAM: 9.9us
  - Decision making
    - Invoked every 100ms – negligible time related to data consolidation
    - $O(n^2)$ complexity, time spent is within measurement error

Conclusions

- Information from performance counters is sufficient to
  - Identify and prioritize latency intolerant applications
  - Separate data sharing from resource contention
- Significant contributors to improving performance:
  - Minimizing expensive communication due to data sharing
  - Identifying impact of data sharing on performance to prioritize applications
  - Identifying resource contention within the core, outside of the core, and outside of the chip

Thank you

Questions?

Coherence Stalls or Latency Tolerance: Informed CPU Scheduling for Socket and Core Sharing

Sharanyan Srikanthan
Sandhya Dwarkadas
Kai Shen

Department of Computer Science
University of Rochester

UNIVERSITY of ROCHESTER