Cooperative Caching for Chip Multiprocessors

Jichuan Chang

Guri Sohi

University of Wisconsin-Madison

ISCA-33, June 2006

Motivation

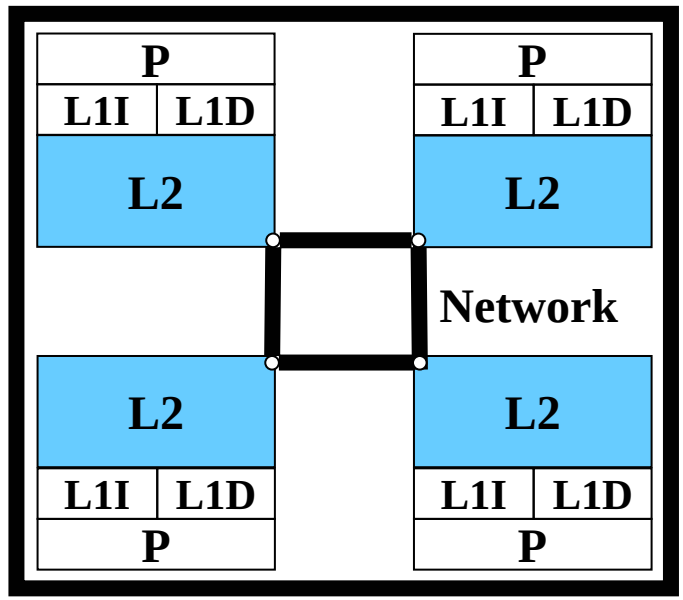
- Chip multiprocessors (CMPs) both require and enable innovative on-chip cache designs
- Two important demands for CMP caching
  - Capacity: reduce off-chip accesses
  - Latency: reduce remote on-chip references
- Need to combine the strength of both private and shared cache designs

Yet Another Hybrid CMP Cache - Why?

- Private cache based design
  - Lower latency and per-cache associativity
  - Lower cross-chip bandwidth requirement
  - Self-contained for resource management
  - Easier to support QoS, fairness, and priority
- Need a unified framework
  - Manage the aggregate on-chip cache resources
  - Can be adopted by different coherence protocols

CMP Cooperative Caching

- Form an aggregate global cache via cooperative private caches
  - Use private caches to attract data for fast reuse
  - Share capacity through cooperative policies
  - Throttle cooperation to find an optimal sharing point
- Inspired by cooperative file/web caches
  - Similar latency tradeoff
  - Similar algorithms



Outline

- Introduction
- CMP Cooperative Caching
- Hardware Implementation
- Performance Evaluation
- Conclusion

Policies to Reduce Off-chip Accesses

- Cooperation policies for capacity sharing
  - (1) Cache-to-cache transfers of clean data
  - (2) Replication-aware replacement
  - (3) Global replacement of inactive data
- Implemented by two unified techniques
  - Policies enforced by cache replacement/placement
  - Information/data exchange supported by modifying the coherence protocol

Policy (1) - Make use of all on-chip data

- Don’t go off-chip if on-chip (clean) data exist
- Beneficial and practical for CMPs
  - Peer cache is much closer than next-level storage
  - Affordable implementations of “clean ownership”
- Important for all workloads
  - Multi-threaded: (mostly) read-only shared data
  - Single-threaded: spill into peer caches for later reuse

Policy (2) – Control replication

- Intuition – increase # of "singlets"
- Latency/capacity tradeoff
  - Evict singlets only when no replications exist
  - Modify the default cache replacement policy
- "Spill" an evicted singlet into a peer cache
  - Can further reduce on-chip replication
  - Randomly choose a recipient cache for spilling

Policy (3) - Global cache management

- Approximate global-LRU replacement
  - Combine global spill/reuse history with local LRU
- Identify and replace globally inactive data
  - First become the LRU entry in the local cache
  - Set as MRU if spilled into a peer cache
  - Later become LRU entry again: evict globally
- 1-chance forwarding (1-Fwd)
  - Blocks can only be spilled once if not reused

Cooperation Throttling

- Why throttling?
  - Further tradeoff between capacity/latency
- Two probabilities to help make decisions
  - Cooperation probability: control replication
  - Spill probability: throttle spilling

Outline

- Introduction
- CMP Cooperative Caching
- Hardware Implementation
- Performance Evaluation
- Conclusion

Hardware Implementation

- Requirements
  - Information: singlet, spill/reuse history
  - Cache replacement policy
  - Coherence protocol: clean owner and spilling
- Can modify an existing implementation
- Proposed implementation
  - Central Coherence Engine (CCE)
  - On-chip directory by duplicating tag arrays

Information and Data Exchange

- Singlet information
  - Directory detects and notifies the block owner
- Sharing of clean data
  - PUTS: notify directory of clean data replacement
  - Directory sends forward request to the first sharer
- Spilling
  - Currently implemented as a 2-step data transfer
  - Can be implemented as recipient-issued prefetch

Outline

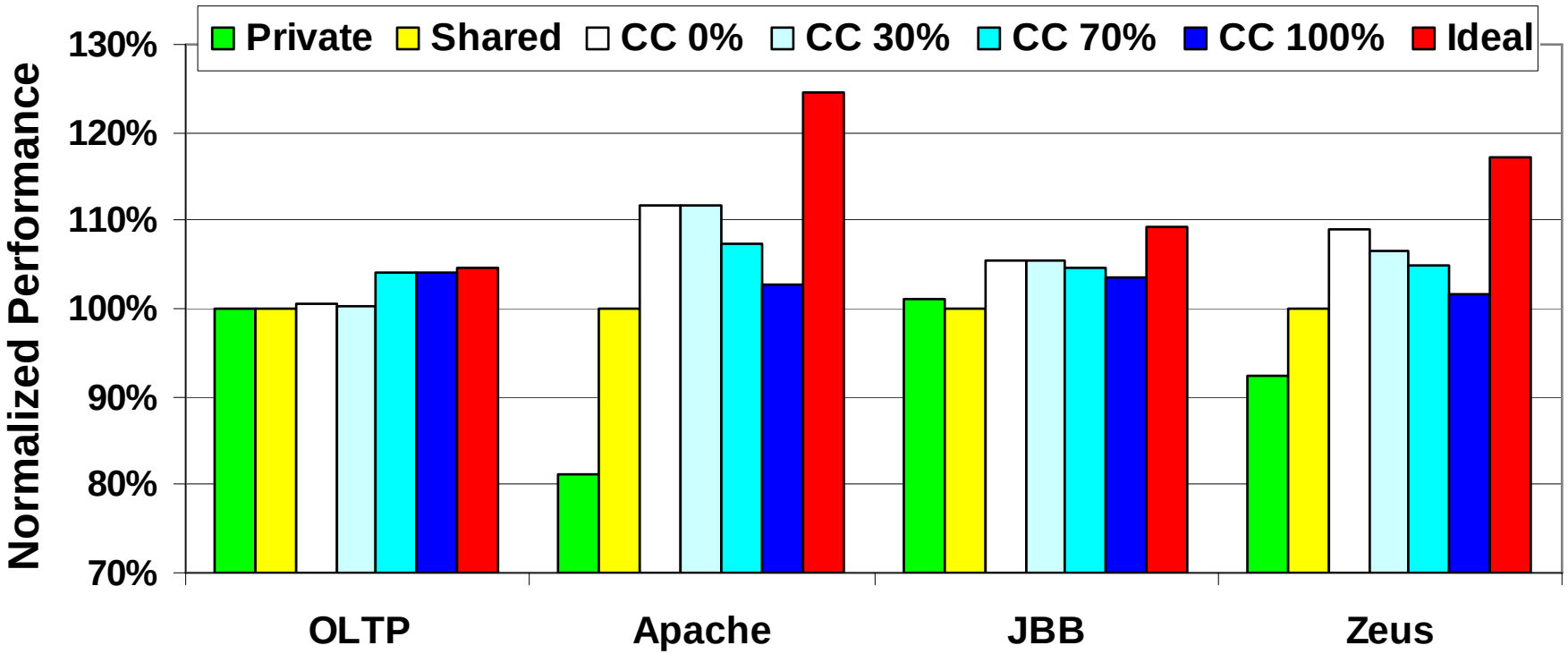
- Introduction
- CMP Cooperative Caching
- Hardware Implementation
- Performance Evaluation
- Conclusion

Performance Evaluation

- Full system simulator
  - Modified GEMS Ruby to simulate memory hierarchy
  - Simics MAI-based OoO processor simulator
- Workloads
  - Multithreaded commercial benchmarks (8-core)
    - OLTP, Apache, JBB, Zeus
  - Multiprogrammed SPEC2000 benchmarks (4-core)
    - 4 heterogeneous, 2 homogeneous
- Private / shared / cooperative schemes
  - Same total capacity/associativity

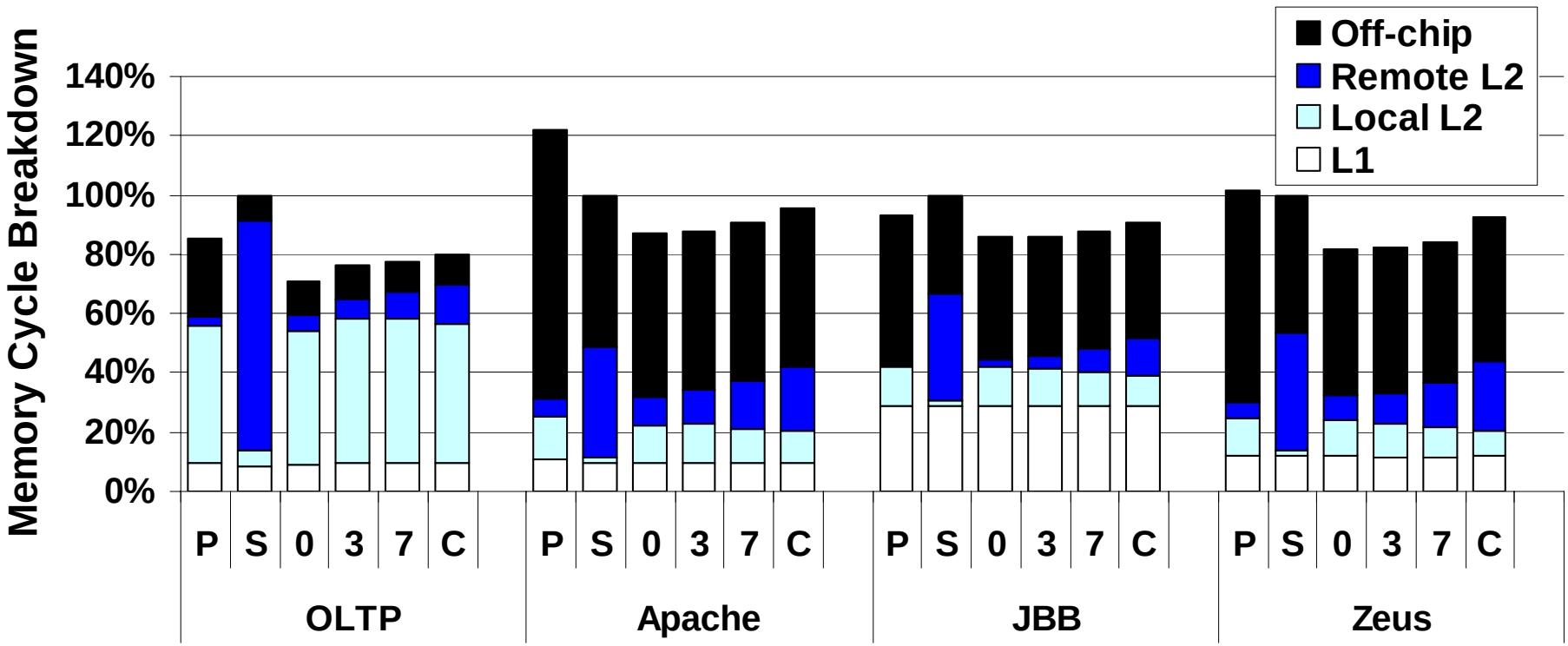
Multithreaded Workloads - Throughput

- CC throttling - 0%, 30%, 70% and 100%
- Ideal – Shared cache with local bank latency

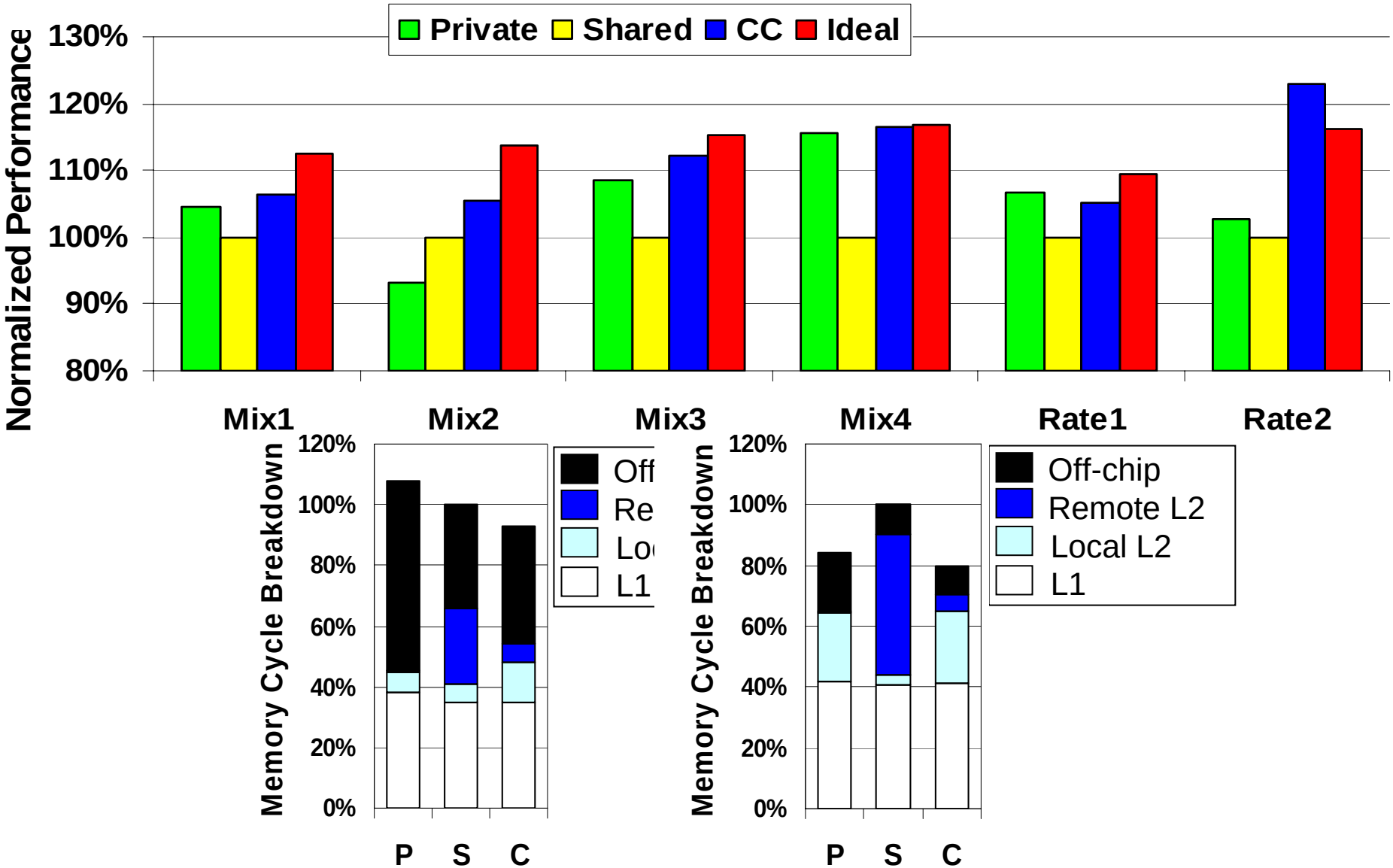


Multithreaded Workloads - Avg. Latency

- Low off-chip miss rate
- High hit ratio to local L2
- Lower bandwidth needed than a shared cache

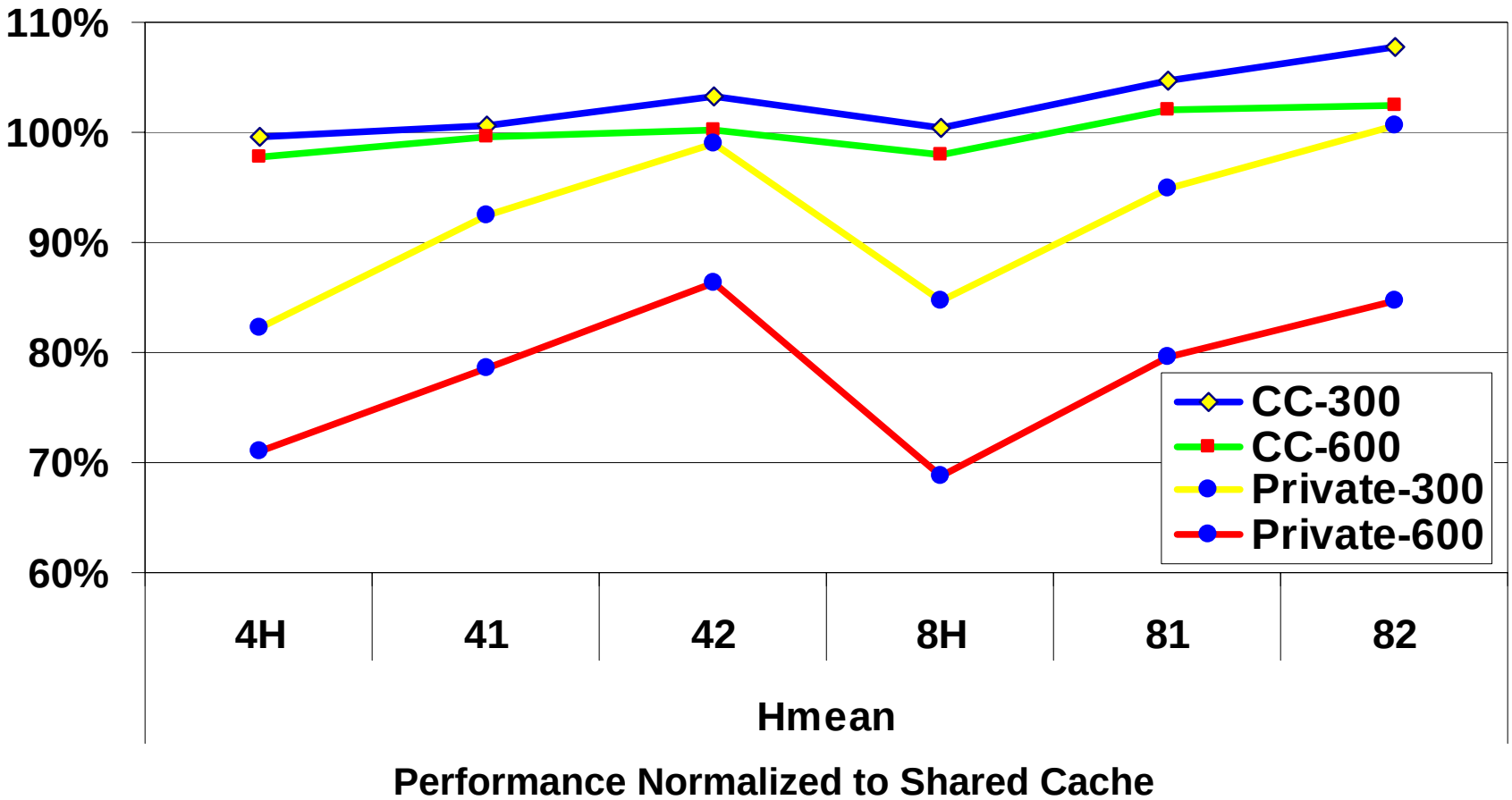


Multiprogrammed Workloads

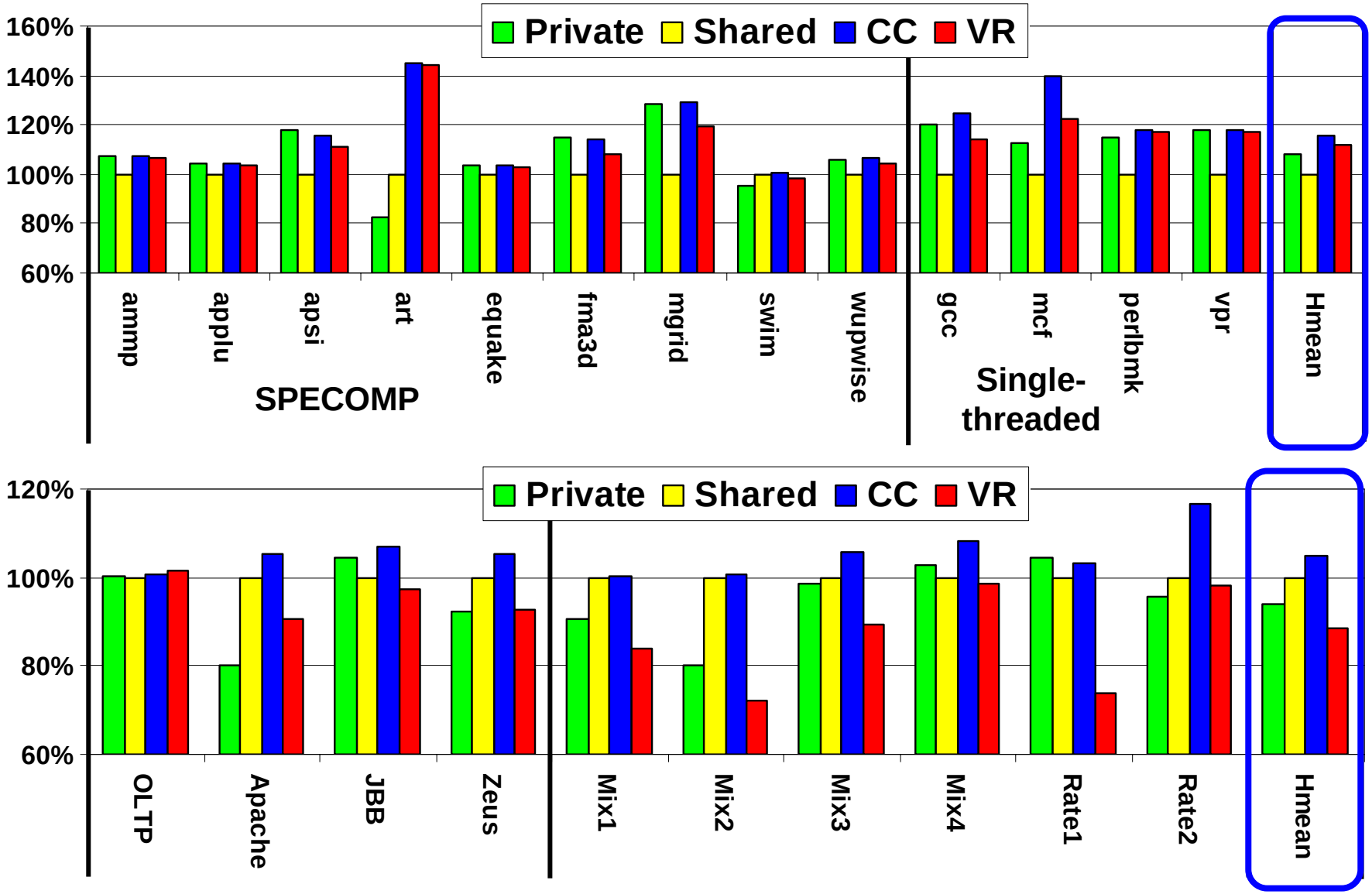


Sensitivity - Varied Configurations

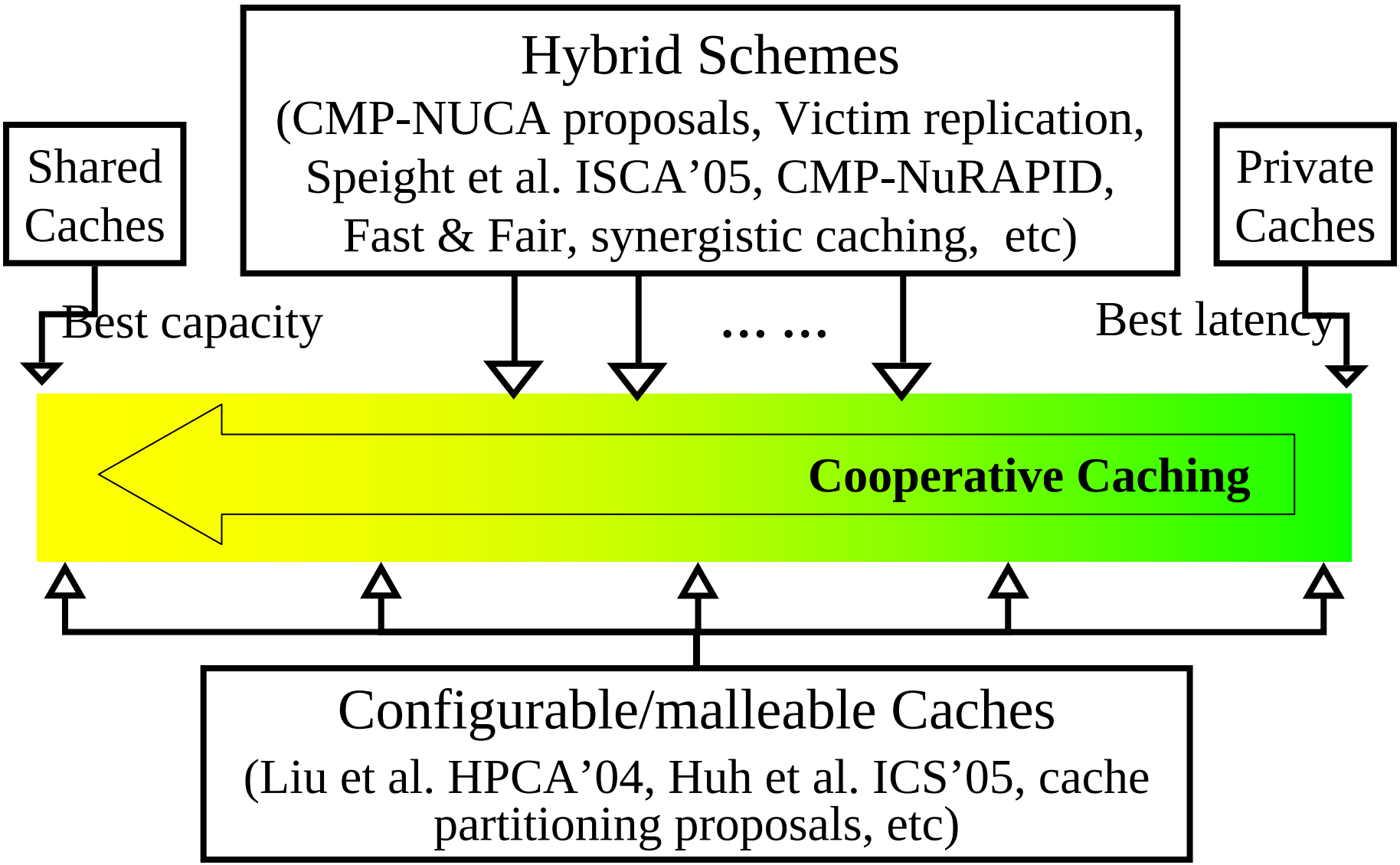
- In-order processor with blocking caches



Comparison with Victim Replication



A Spectrum of Related Research



Conclusion

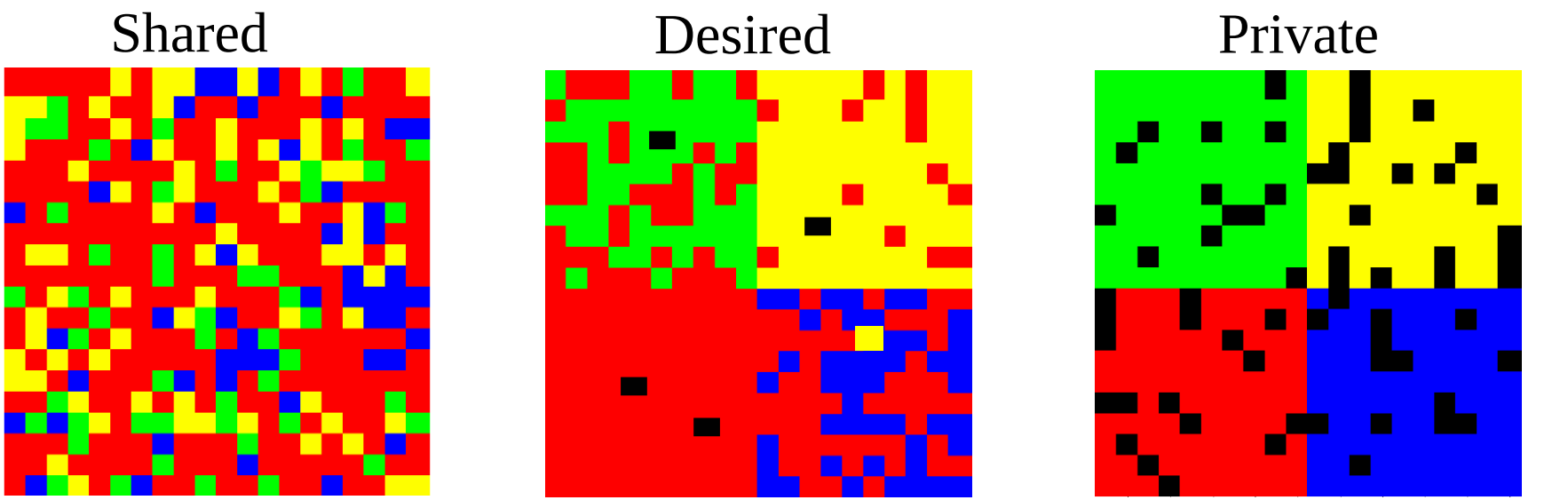
- CMP cooperative caching
  - Exploit benefits of private cache based design
  - Capacity sharing through explicit cooperation
  - Cache replacement/placement policies for replication control and global management
  - Robust performance improvement

Thank you!

Backup Slides

Shared vs. Private Caches for CMP

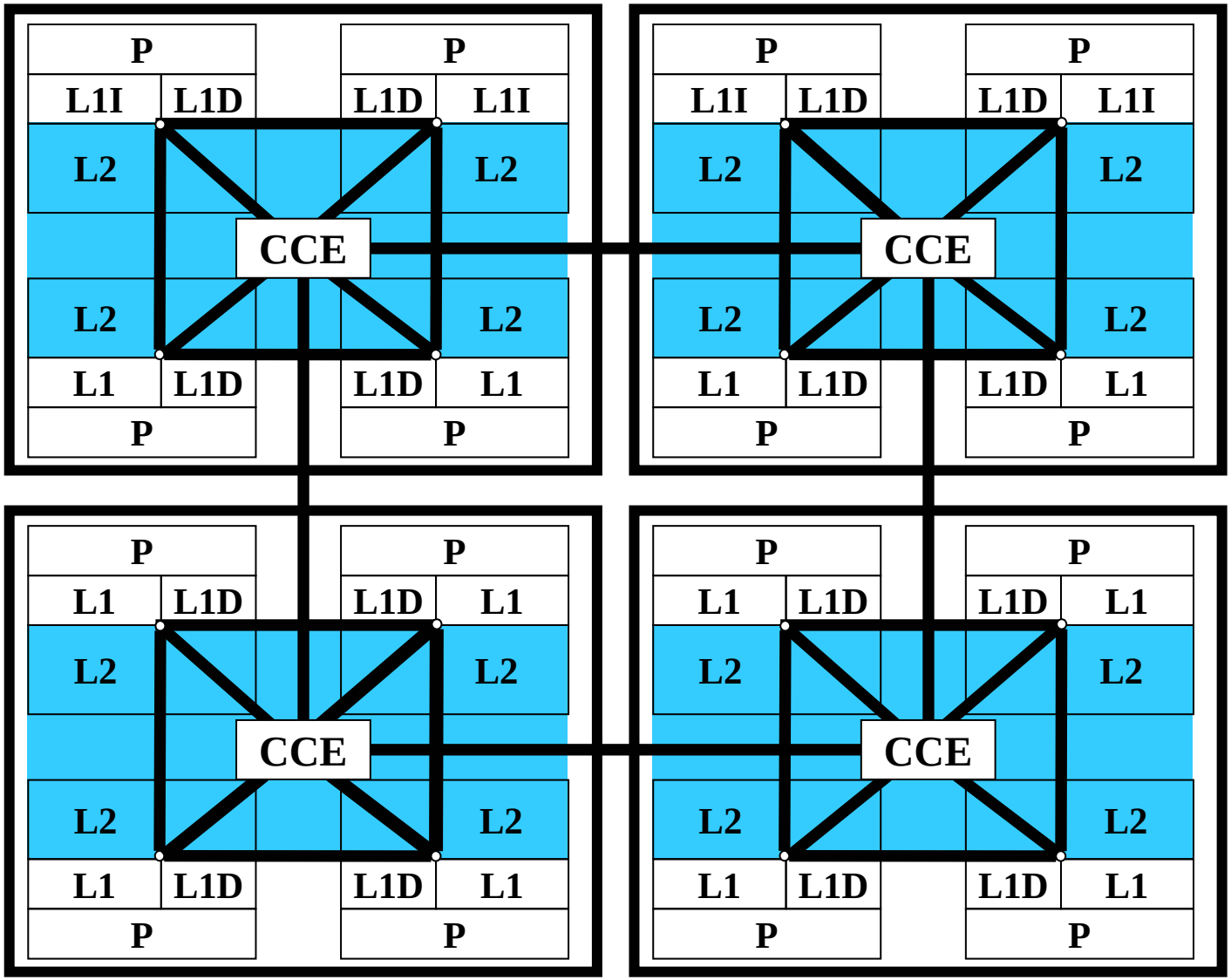
- Shared cache – best capacity
  - No replication, dynamic capacity sharing
- Private cache – best latency
  - High local hit ratio, no interference from other cores
- Need to combine the strengths of both



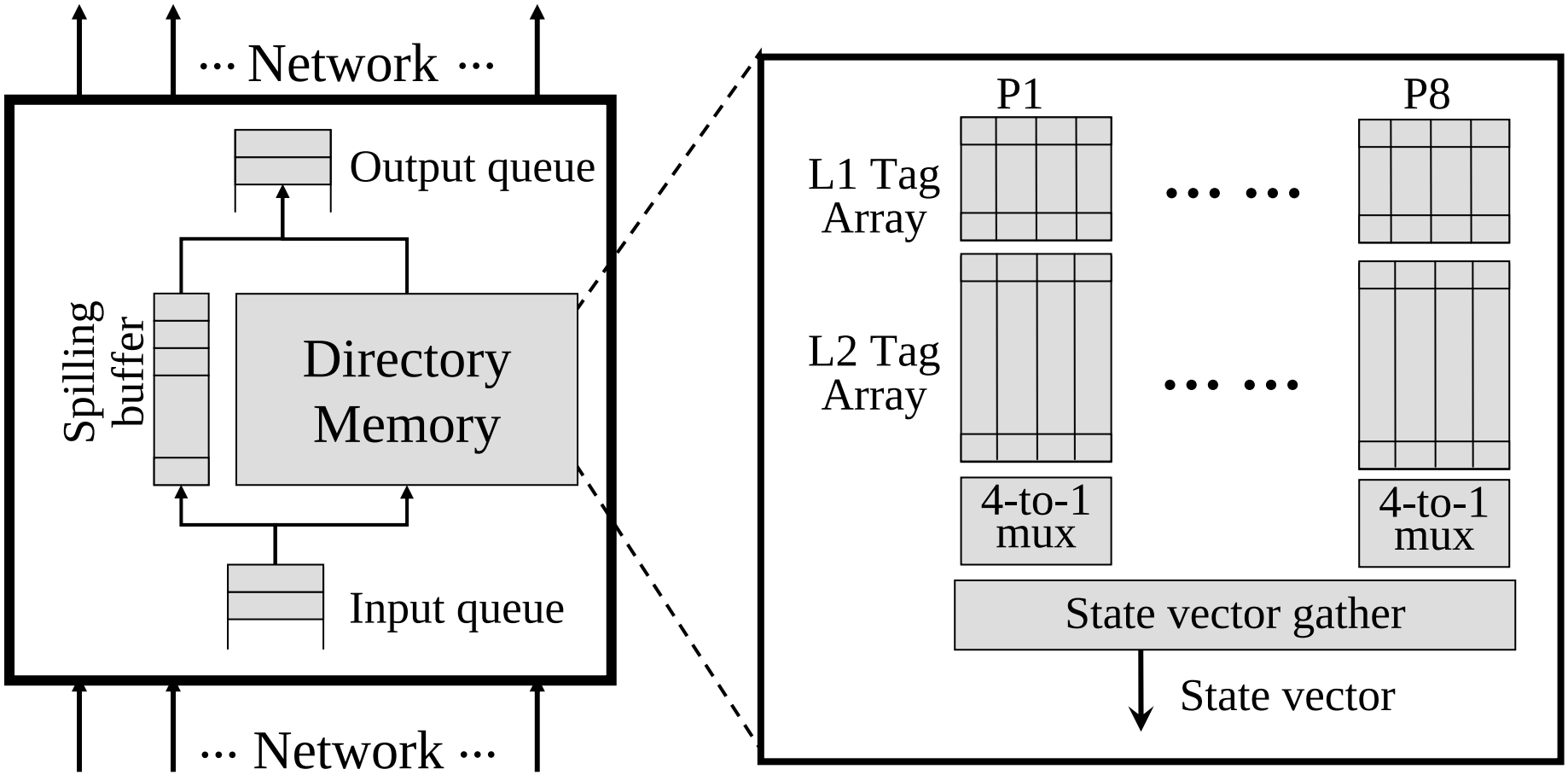
CMP Caching Challenges/Opportunities

- Future hardware
  - More cores, limited on-chip cache capacity
  - On-chip wire-delay, costly off-chip accesses
- Future software
  - Diverse sharing patterns, working set sizes
  - Interference due to capacity sharing
- Opportunities
  - Freedom in designing on-chip structures/interfaces
  - High-bandwidth, low-latency on-chip communication

Multi-cluster CMPs (Scalability)



Central Coherence Engine (CCE)



Cache/Network Configurations

- Parameters
  - 128-byte block size; 300-cycle memory latency
  - L1 I/D split caches: 32KB, 2-way, 2-cycle
  - L2 caches: Sequential tag/data access, 15-cycle
  - On-chip network: 5-cycle per-hop
- Configurations
  - 4-core: 2X2 mesh network
  - 8-core: 3X3 or 4X2 mesh network
  - Same total capacity/associativity for private/shared/cooperative caching schemes