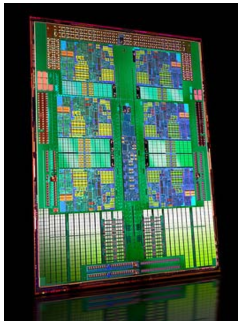
COHESION: A Hybrid Memory Model for Accelerators

John H. Kelm, Daniel R. Johnson, William Tuohy, Steven S. Lumetta and Sanjay J. Patel

University of Illinois at Urbana-Champaign

Chip Multiprocessors Today

- General-purpose + accelerators (e.g., GPUs)
- General-purpose CMP Challenges:
  1. Programmability
  2. Power/perf density of ILP-centric cores
  3. **Scalability of HW coherence, strict memory models**

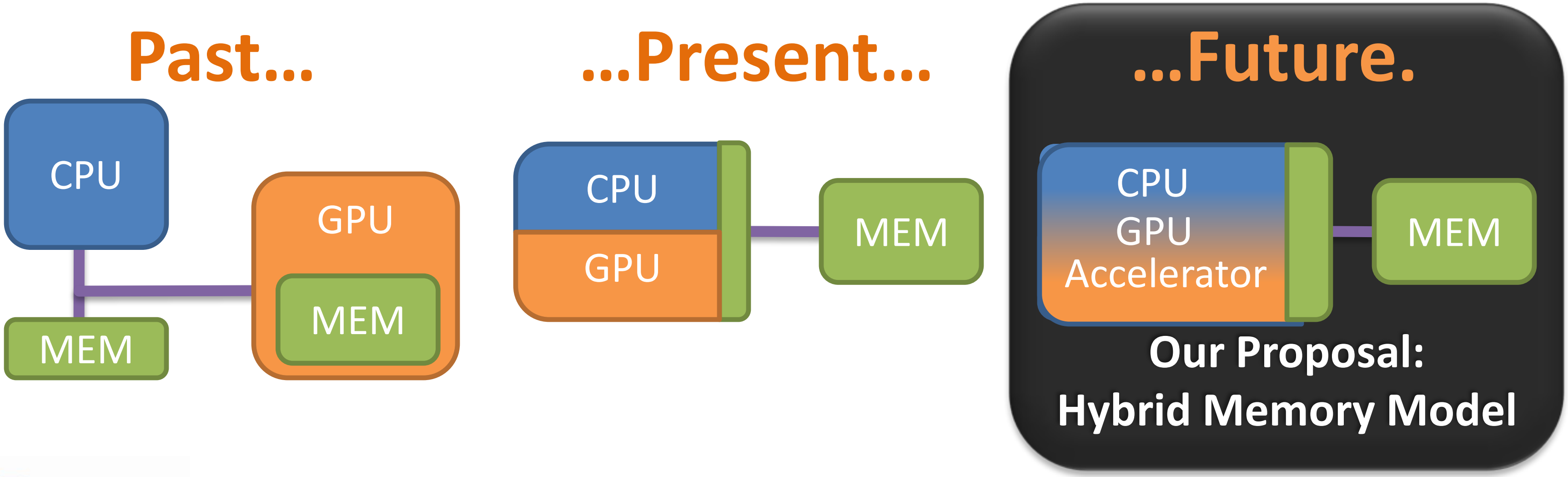


- Accelerator Challenges:
  1. Inflexible programming/execution models
  2. Hard to scale irregular parallel apps
  3. **Lack of conventional memory model**

Chip Multiprocessors Tomorrow

- Industry Trend: Integration over time
- Hybrids: Accelerators + CPUs together on die
- More core/compute heterogeneity but…

  …more homogeneity in memory model



Past…

…Present…

…Future.

Our Proposal:
Hybrid Memory Model

CMP Memory Model Choices

Conventional Multicore CPU

- Ex: Intel i7, Sun Niagara
- Optimized for:
  - Minimal latency
  - Tightly coupled sharing
  - Fine-grained synchronization
  - Minimal programmer effort
- Provides:
  - **Single address space**
  - **Hardware caching**
  - Strong ordering
  - HW-managed coherence

Contemporary Accelerator

- Ex: NVIDIA GPU, IBM Cell
- Optimized for:
  - **Maximum throughput**
  - **Loosely coupled sharing**
  - **Coarse-grained synchronization**
  - Short silicon design cycle
- Provides:
  - Multiple address spaces
  - Scratchpad memories
  - Relaxed ordering
  - SW-managed coherence

Roadmap

- Motivation and context
- Problem statement
- COHESION design
- Use cases and programming examples

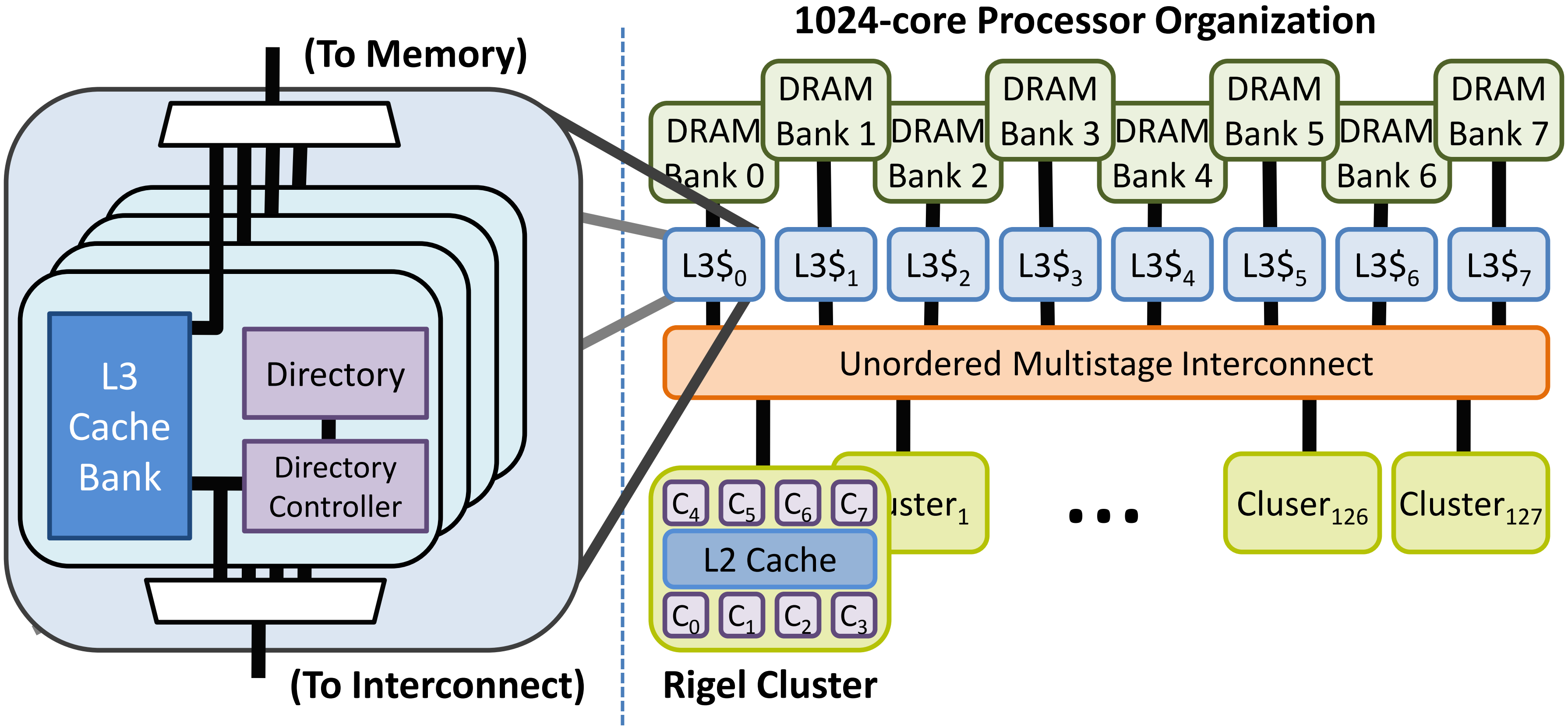
Addressed in this talk:

1. **Opportunity**: Is combining protocols worthwhile?
2. **Feasibility**: How does one implement hybrid memory models?
3. **Tradeoffs**: What are the tradeoffs in HW_{cc} v. SW_{cc}?
4. **Benefit**: What does hybrid coherence get you?

Problem: Scalable Coherence

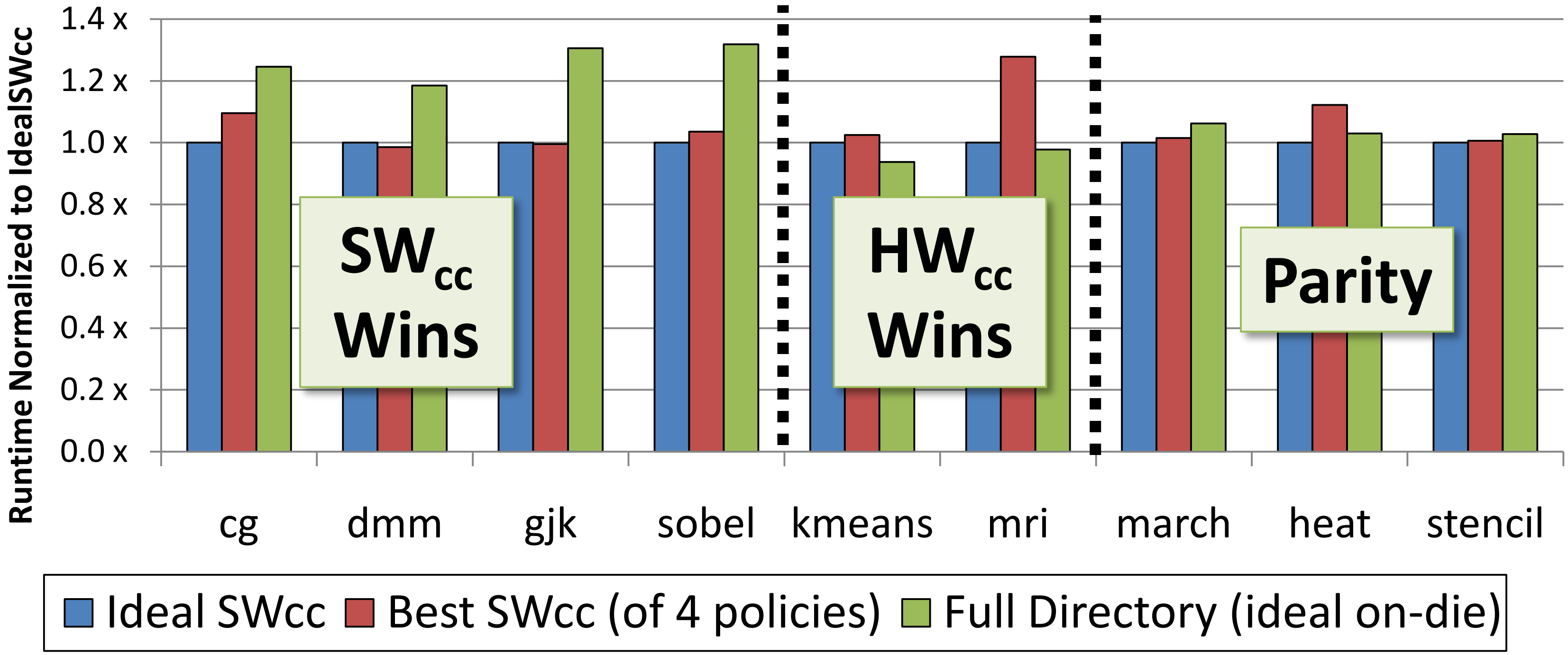
- Available architectures:
  - **Accelerators**: 100s of cores, TFLOPS, no coherence
  - **CMPs**: <10s of cores, GFLOPS, HW coherence
  - Multiple memory models on-die
- What devs want in heterogeneous CMPs:
  - Hardware caches (locality)
  - Single address space (marshalling)
  - Minimal changes to current practices
- Accelerator scalability w/CMP memory model

Baseline Architecture



- Variant of the Rigel Architecture [Kelm et al. ISCA'09]
- 1024-core CMP, HW caches, single address space, MIMD

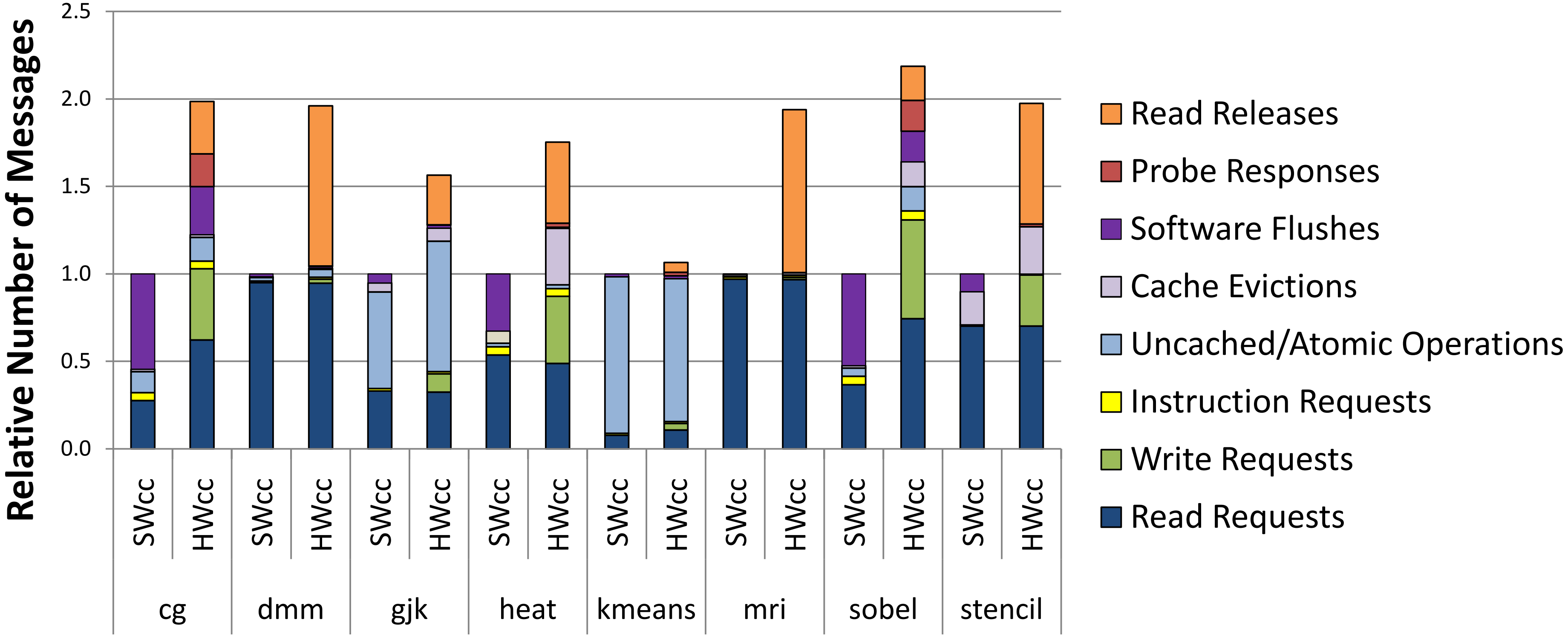
Opportunity: HW_{cc} v. SW_{cc} Shootout



- Note: Lower bars are better
- **Question**: Can we leverage both HW+SW protocols?

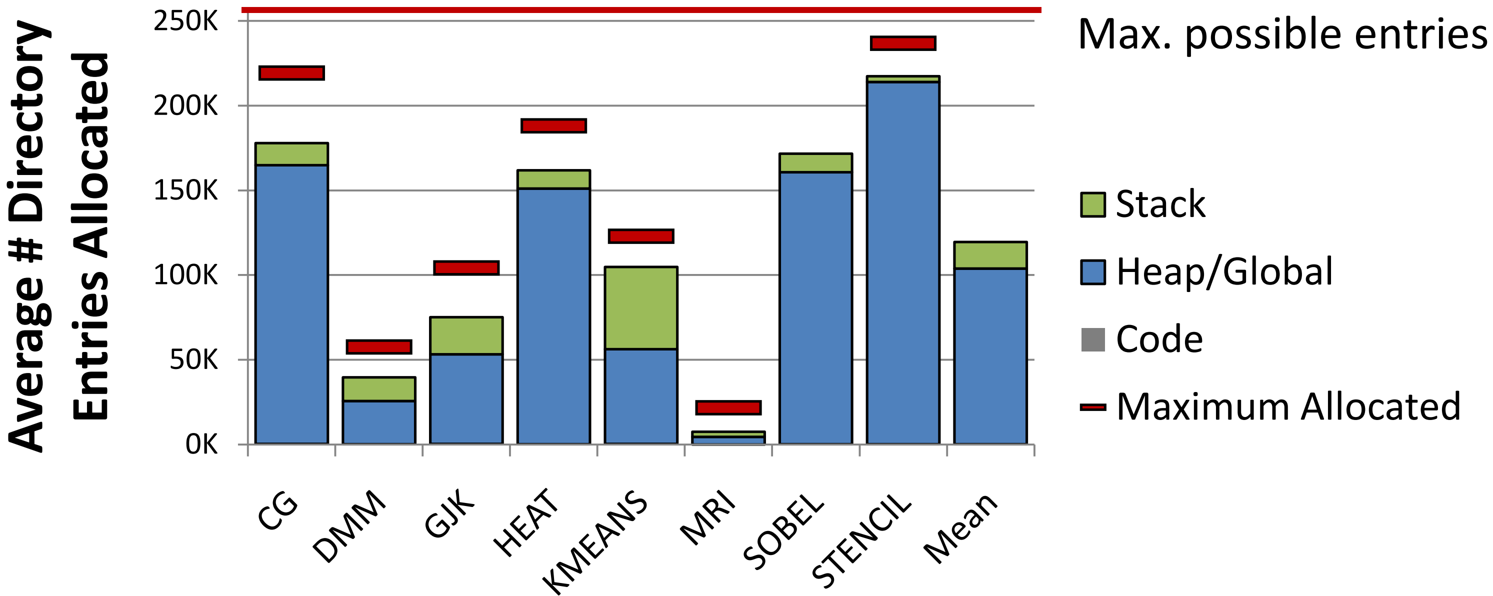
* SWcc based on the Task-centric Memory Model [Kelm et al. PACT'09][Kelm et al. IEEEMicro'10]

Opportunity: Network Traffic Reduction



- **SW$_{cc}$** w/baseline arch (**left**), HW$_{cc}$ ww/DIR$_{FULL}$ (**right**)
- **SW$_{cc}$**: Fewer L2 messages in network, some flush overhead
- **HW$_{cc}$**: Extraneous msgs for unshared data (Wr$_{Request}$, Rd$_{Release}$)

Opportunity: Reduce Directory Utilization

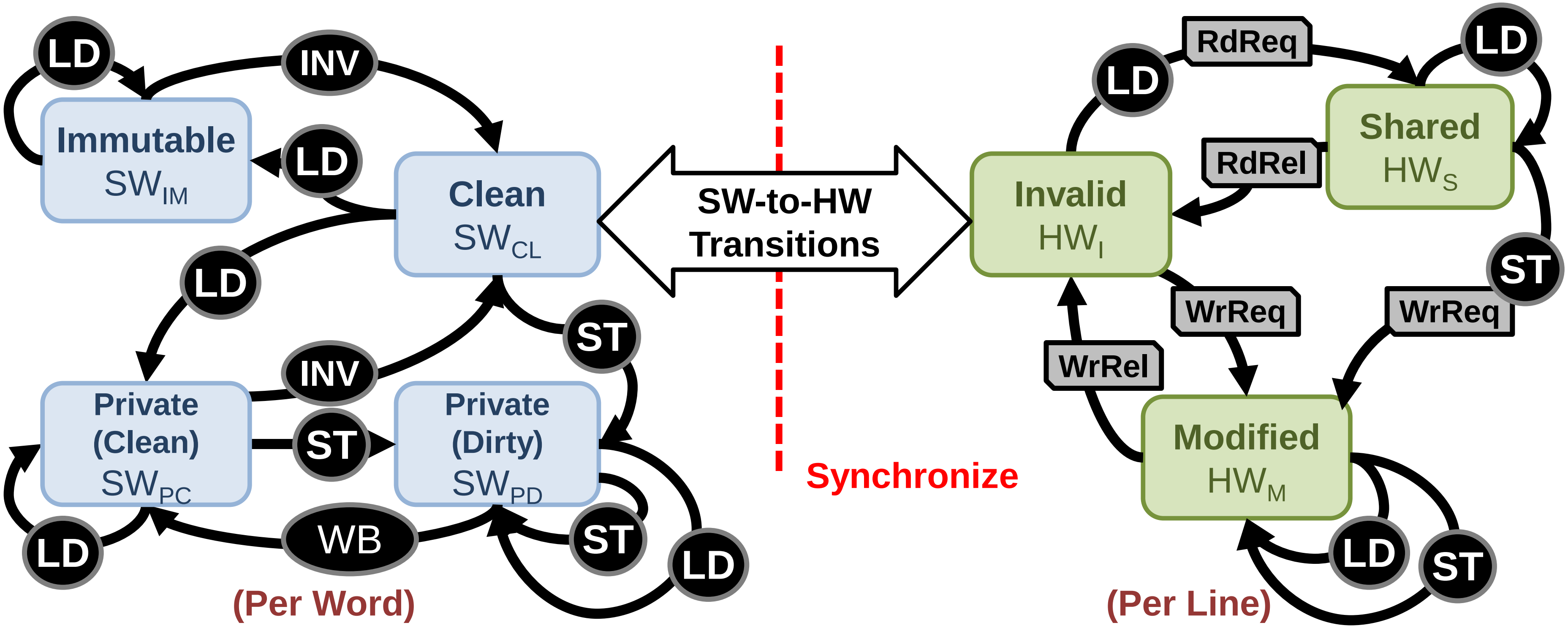


- Not all entries used →Wasted die area
- For many, 256K maximum never reached (red line)
- Observations:
  1. Use SW_{cc} when possible to reduce network traffic
  2. Build smaller sparse directory for common case

COHESION: Toward a Hybrid Memory Model

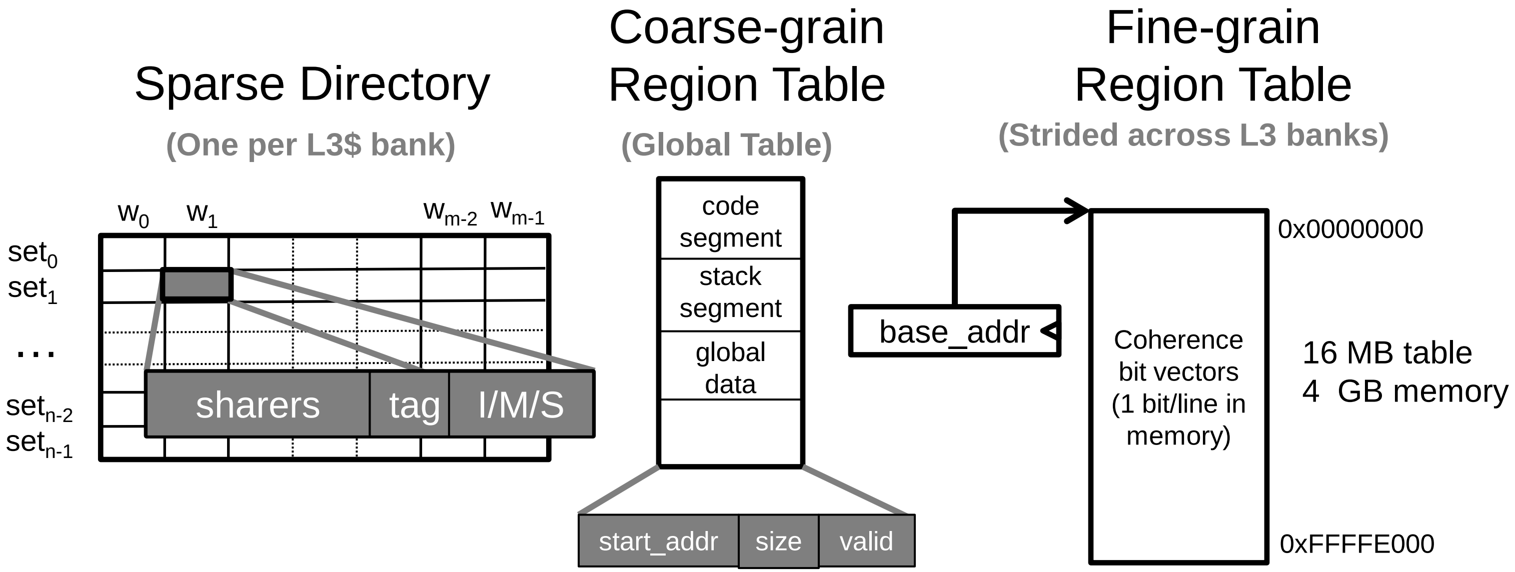
- Support for coherence domain transitions
  1. Protocol for safe migration $\text{SW}_{cc} \Leftrightarrow \text{HW}_{cc}$
  2. Minor architecture extensions
- Motivation:
  - ↑ HW_{cc}: Supports arbitrary sharing, no SW overhead
  - ↓ HW_{cc}: Area + message overhead
  - ↑ SW_{cc}: Removes HW overheads + design complexity
  - ↓ SW_{cc}: Flush overhead + coherence burden on SW

Protocol Synthesis



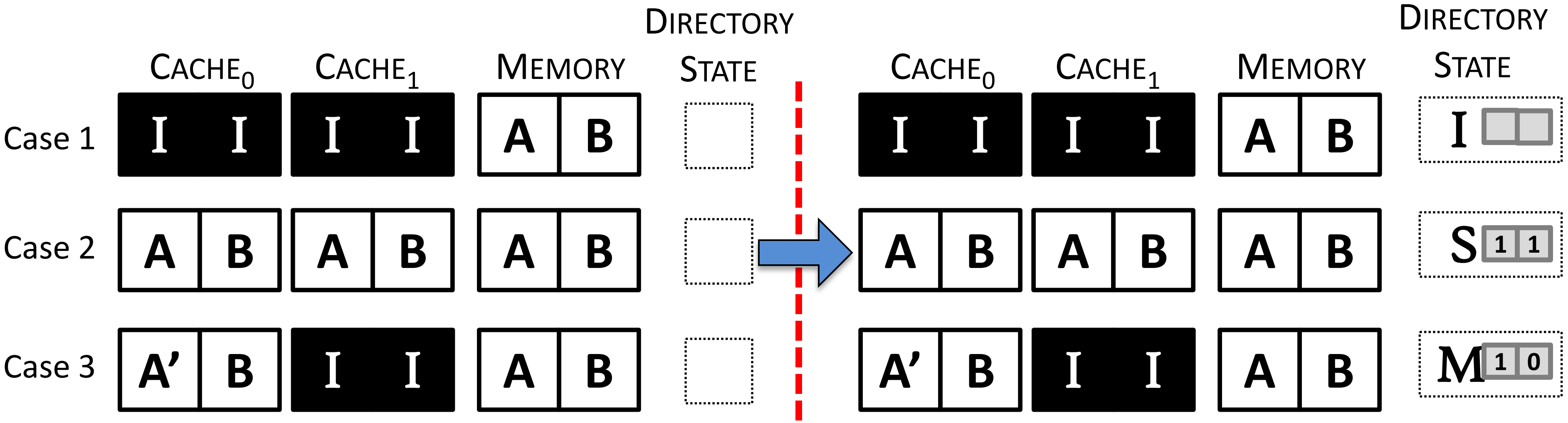
- Create a bridge between SW_{cc} and HW_{cc}
- Leverage existing HW_{cc} and SW_{cc} techniques

COHESION Architecture



- Extension to baseline directory protocol
  - Addition 1: Region table/bit vector in memory
  - Addition 2: One bit/line in the L2 cache (not shown)
- SW writes table → COHESION controller exec's transition

Example Software → Hardware Transitions

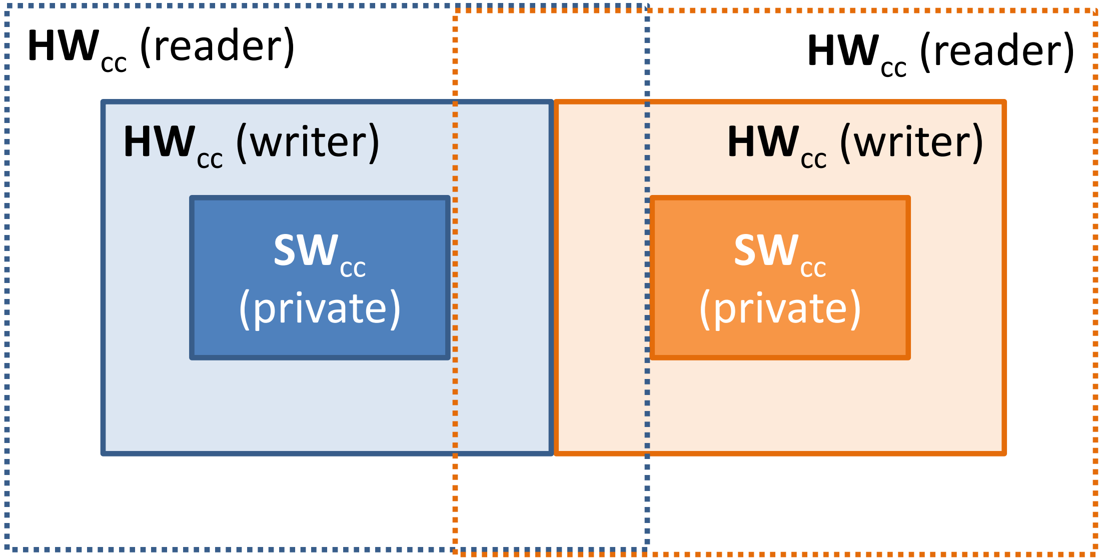


	CACHE_0	CACHE_1	MEMORY	DIRECTORY STATE		CACHE_0	CACHE_1	MEMORY	DIRECTORY STATE
Case 1	I I	I I	A B			I I	I I	A B	I
Case 2	A B	A B	A B		→	A B	A B	A B	S 1 1
Case 3	A’ B	I I	A B			A’ B	I I	A B	M 1 0

- App. initiates transitions between SW_{cc} and HW_{cc}
- COHESION controller probes L2’s to reconstruct state
- *See paper for other cases and HW_{cc} → SW_{cc}*

Static COHESION Example (1 of 3)

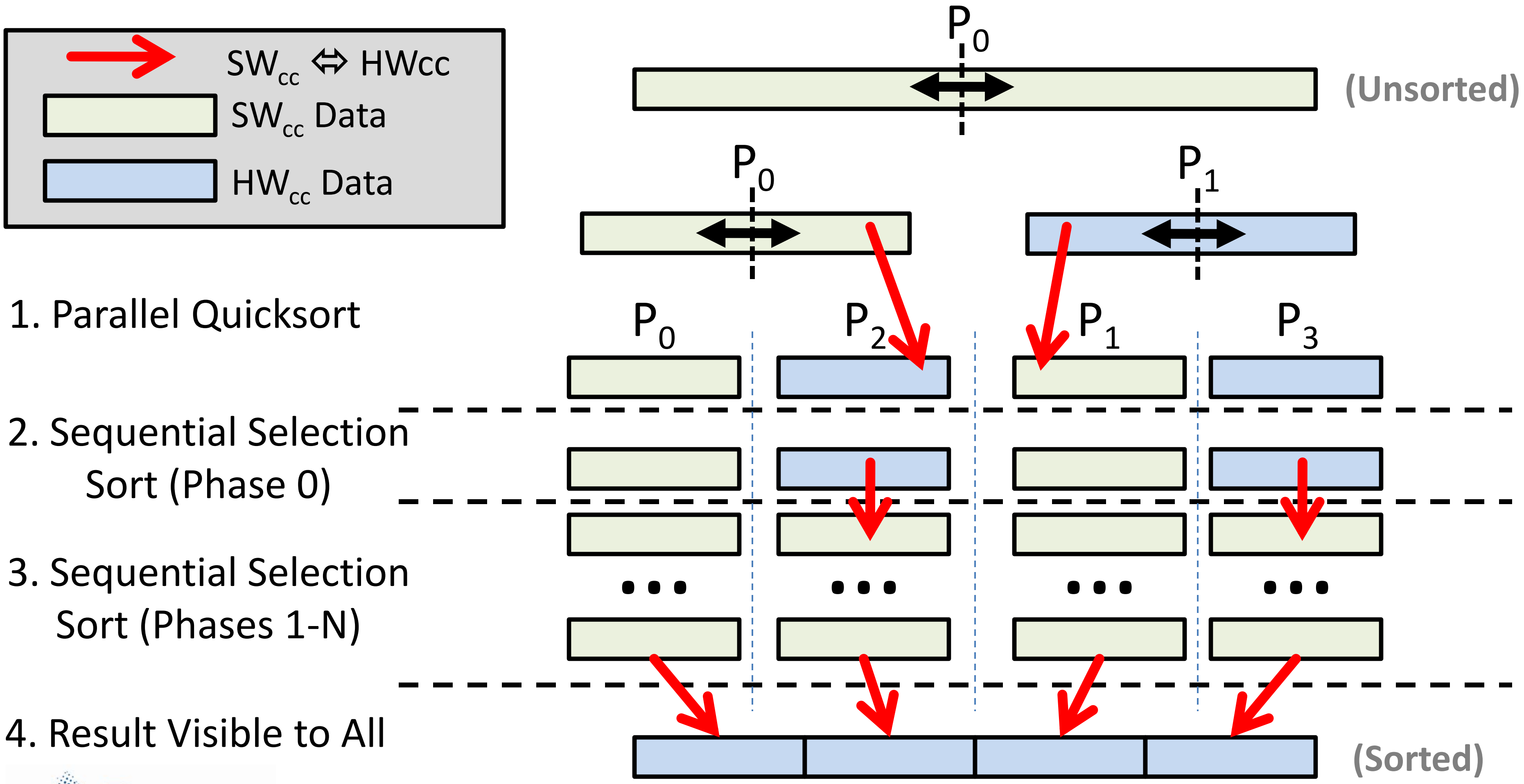
Data regions for two grid blocks from a 2D stencil computation


$\textbf{HW}_{cc}$ (reader)
$\textbf{HW}_{cc}$ (reader)
$\textbf{HW}_{cc}$ (writer)
$\textbf{HW}_{cc}$ (writer)
$\textbf{SW}_{cc}$ (private)
$\textbf{SW}_{cc}$ (private)

- COHESION provides static partitioning of data
- (Large) read-only/private regions SW_{cc}
- (Small) shared regions HW_{cc}

Dynamic Cohesion Example (2 of 3)

Parallel Sort (on four cores)



1. Parallel Quicksort

2. Sequential Selection Sort (Phase 0)

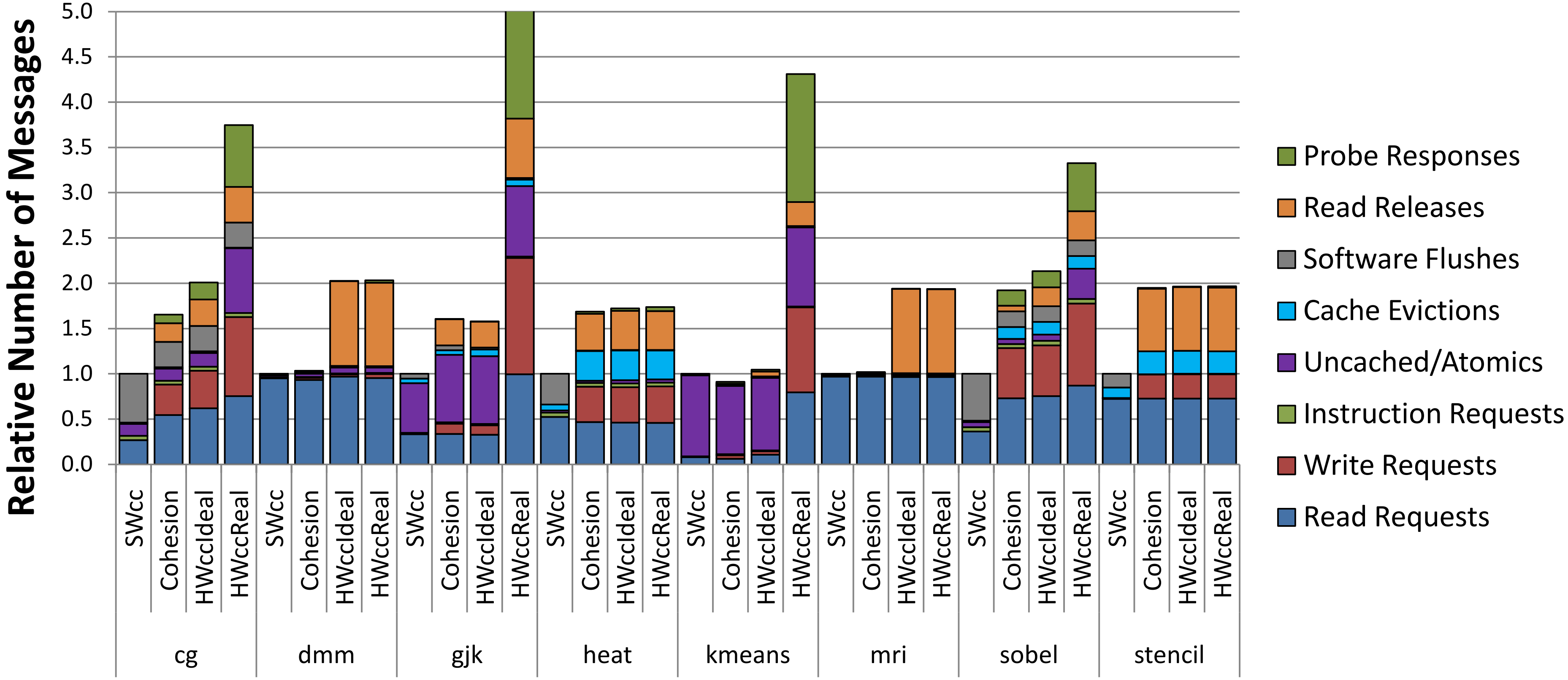
3. Sequential Selection Sort (Phases 1-N)

4. Result Visible to All

System SW Cohesion Example (3 of 3)

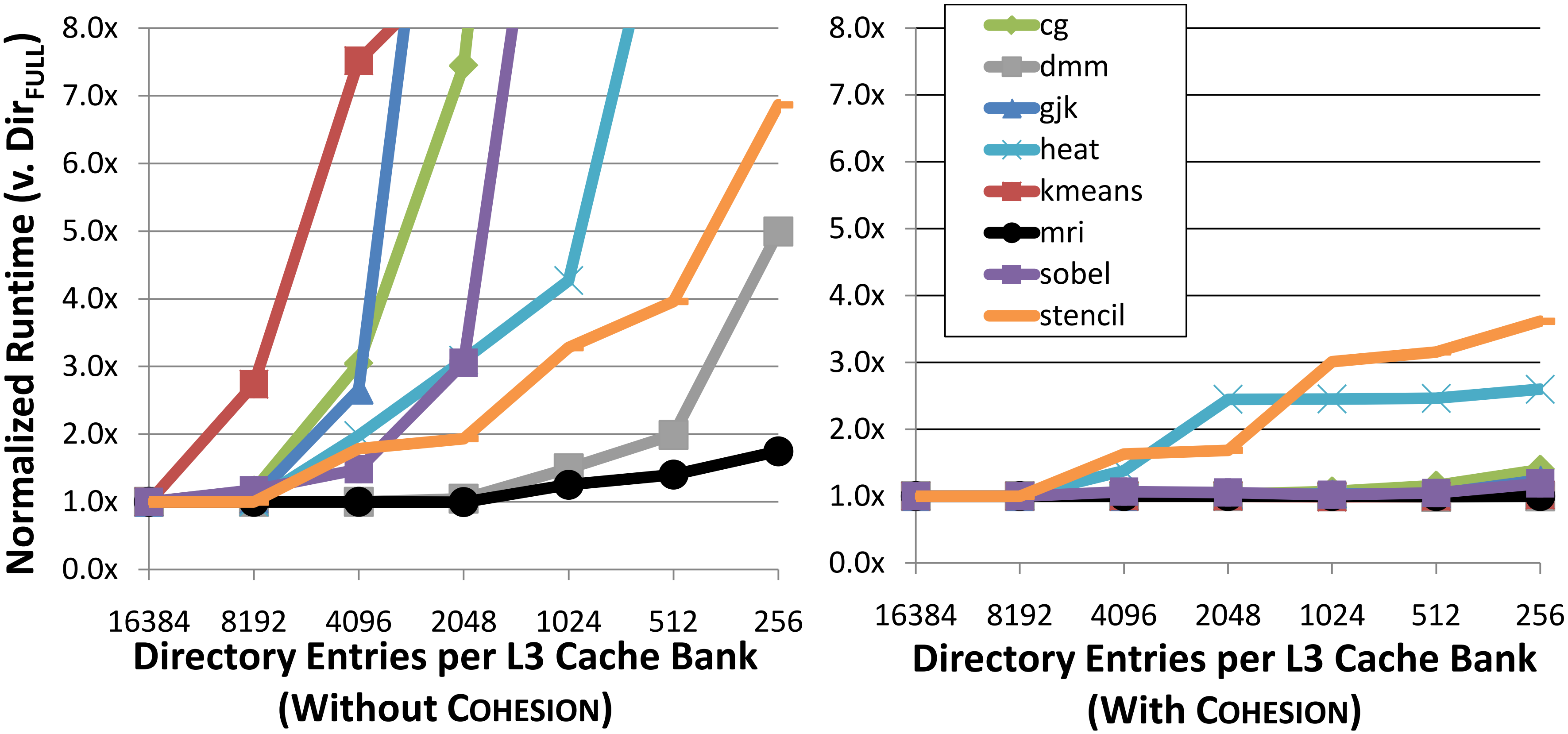
- Problem: Supporting multitasking w/SW_{cc}
- OS process creation workflow
  1. Runtime allocates proc’s memory HW_{cc}
  2. Start new process
  3. Process runs, migrates, SW_{cc} ⇔ HW_{cc} transitions
  4. Exit process
  5. Runtime makes allocated memory HW_{cc}
- Cohesion enables: Migration, isolation, cleanup

Network Message Reductions



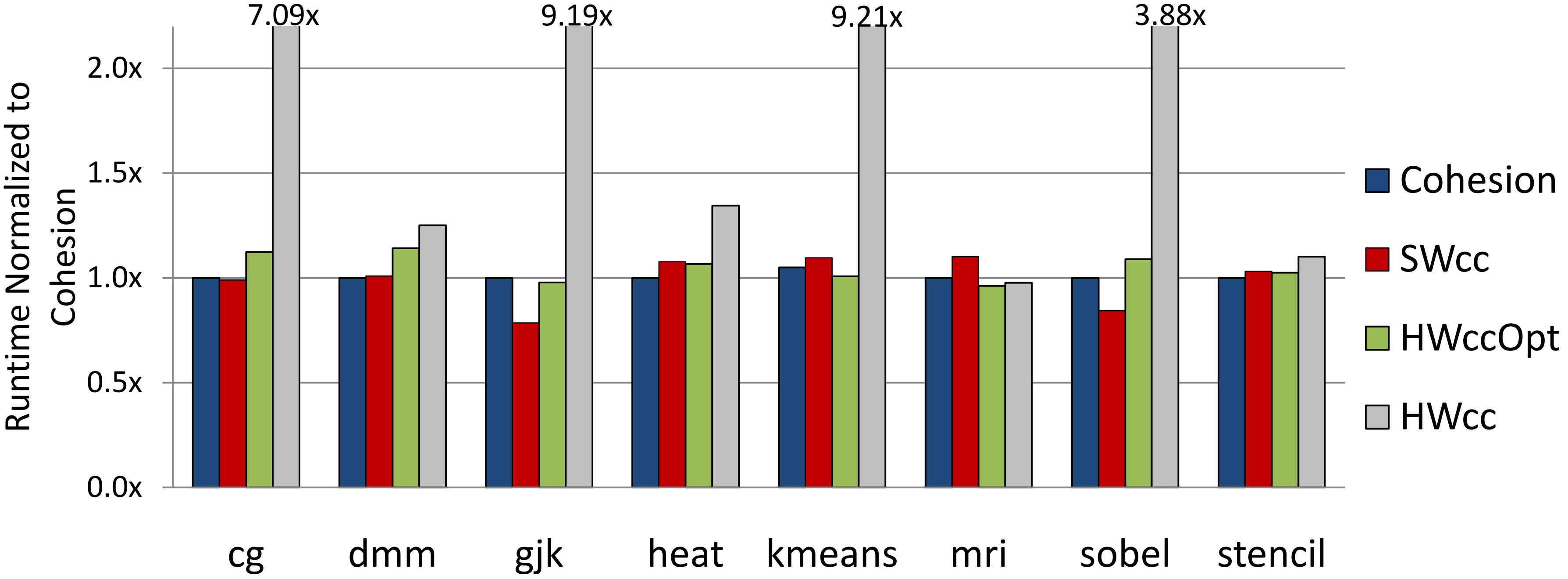
- HW_{cc}Real: HW_{cc}-only w/sparse directory used by COHESION
- HW_{cc}Ideal: Full on-die directory
- **Benefit**: lessens constraints on network design

Directory Size Sensitivity



- Reduces perf. cliffs in sparse directory designs
- **Benefit**: Smaller on-die coherence structures

Runtime: COHESION, $\mathrm{SW_{cc}}$, $\mathrm{HW_{cc}}$



- Perf. close to $\mathrm{SW_{cc}}$ and full-directory $\mathrm{HW_{cc}}$
- Reduce network/directory overhead w/o perf. loss
- Further $\mathrm{HW_{cc}}$→$\mathrm{SW_{cc}}$ optimizations possible

Conclusions

- Why COHESION? CMPs w/multiple mem. models
- Usage scenarios identified
  - System software/migratory tasks w/SW_{cc}
  - App uses: Static, dynamic, and host+accel
  - Optimization Path: Piecemeal HW_{cc}→SW_{cc}
- Hybrid memory model has potential
  - Reduces strain on HW_{cc} implementation
  - Reduces network constraints
  - Competitive performance