18-742:
Computer Architecture & Systems

Why On-chip Cache Coherence is Here To Stay

Prof. Phillip Gibbons

Spring 2025, Lecture 4

What is Memory Consistency?

- **A model which describes the behavior of memory operations (reads, writes) in terms of visibility and ordering.**
- **Not a problem specific to multi-processors**
  - Out-of-order execution
- **But multi-processors make it more complicated**

Memory Consistency in Practice

```
/* initial A = B = 0 */
P1          P2
A = 1;      B = 1
r1 = B;     r2 = A;
```

```
/* initially all 0 */
P1          P2
A = 1;      while (flag == 0);
B = 1;      r1 = A;
flag = 1;   r2 = B;
```

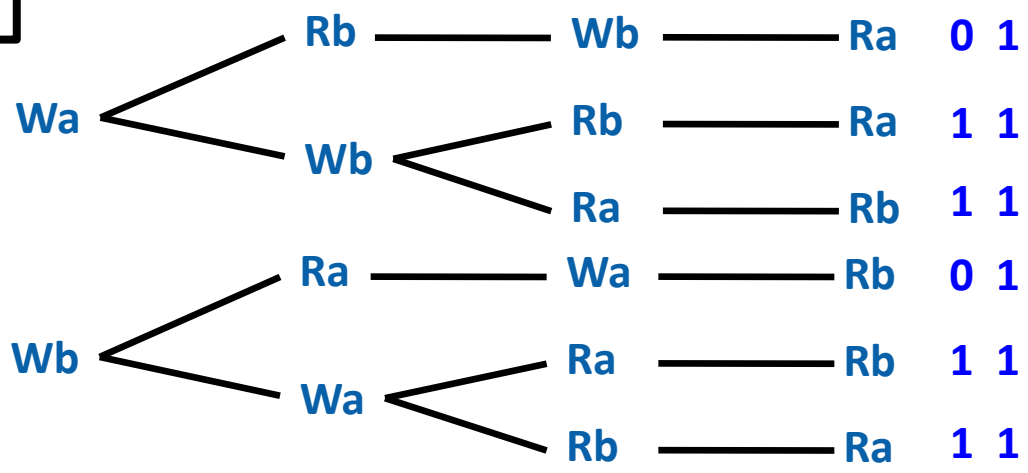
What are the final values of r1 and r2?

Sequential Consistency (SC) Model

As if only one operation at a time, in an order consistent with the order of operations within each thread

```
/* initial A = B = 0 */
P1              P2
A = 1;          B = 1
r1 = B;         r2 = A;
```

What are the final values of r1 and r2 under SC?



[Lamport, 1979]

Sequential Consistency (SC) Model

As if only one operation at a time, in an order consistent with the order of operations within each thread

```
/* initially all 0 */
P1                  P2
A = 1;              while (flag == 0);
B = 1;              r1 = A;
flag = 1;           r2 = B;
```

What are the final values of r1 and r2 under SC?

What Is Wrong with SC?

- **Performance Overhead**
  - No reordering
  - Reduced concurrency

```
/* initially all 0 */
P1                  P2
A = 1;              while (flag == 0);
B = 1;              r1 = A;
flag = 1;           r2 = B;
```

Consequence: No popular architecture supporting SC

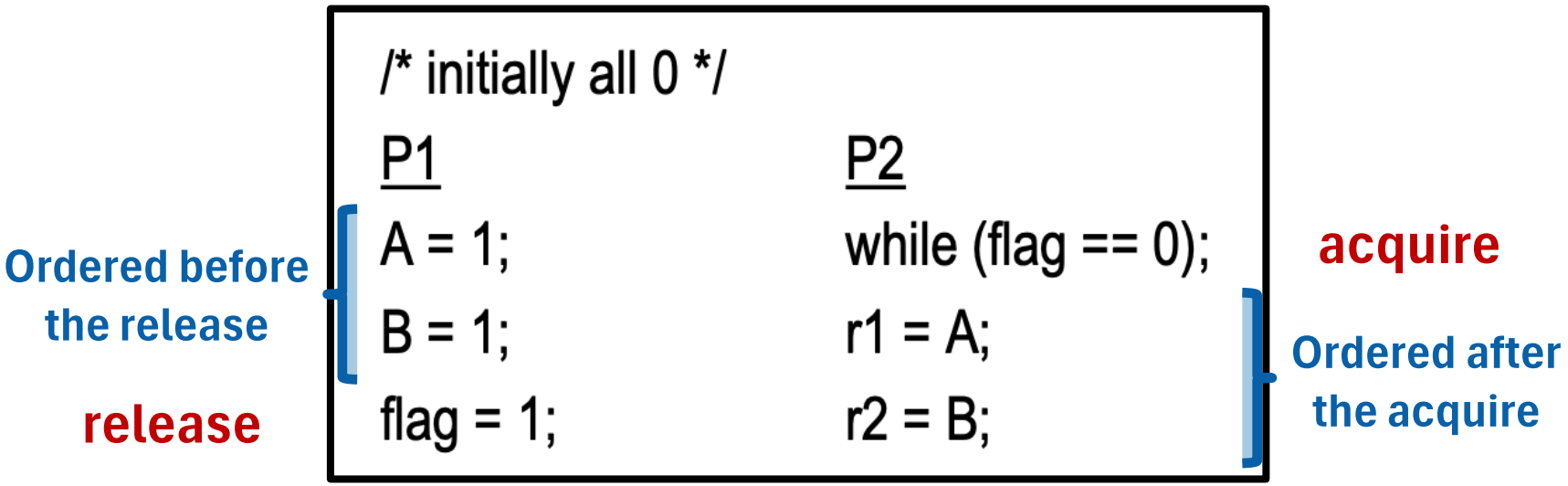
Relaxed Consistency

- **Relaxed memory models allow more flexibility in the ordering of read and write operations.**

- **This can lead to improved performance but requires more careful synchronization in the program to maintain data integrity.**
  - The system doesn't guarantee that a write by one processor will be immediately visible to other processors.
  - Must use **Fences/Barriers** or **Synchronization** to enforce order.

Release Consistency (1990)

- **Programs should be free of data races**
  - Use synchronization to avoid data races



- **Provides SC for data-race free programs**

Weak Consistency in Practice

```
/* initially all 0 */
 P1                      P2
A = 1;                   while (FENCE flag == 0);
B = 1;                   r1 = A;
FENCE flag = 1;          r2 = B;
```

All order is preserved across a FENCE instruction

Other Memory Consistency Models

- **Total Store Order (TSO)**
  - The middle-ground approach
  - Some reordering is allowed
  - Writes can be delayed (buffered) but are seen by all processors in the same order

- **SUN Microsystems SPARC V8 was first to support TSO (1992)**

- **Intel x86 only committed to TSO c2008**

Memory Consistency, Now

- **Well-handled in modern systems**
  - Modern programming languages and hardware architectures have evolved to handle many aspects of memory consistency more transparently.

```
#include <atomic>
std::atomic<int> counter = 0;

void increment() {
    counter++;
}
```

Memory Consistency, Now

```
std::atomic<int> flag = 0;

void thread1() {
    flag.store(1, std::memory_order_release);
}

void thread2() {
    while (flag.load(std::memory_order_acquire) == 0);
    // Proceeds only after flag is set to 1
}
```

Memory Consistency, Now

```cpp
#include <mutex>
std::mutex mu;
int shared_data = 0;

void safe_increment() {
    std::lock_guard<std::mutex> guard(mu);
    shared_data++;
    // Mutex is automatically released when guard goes out of scope
}
```

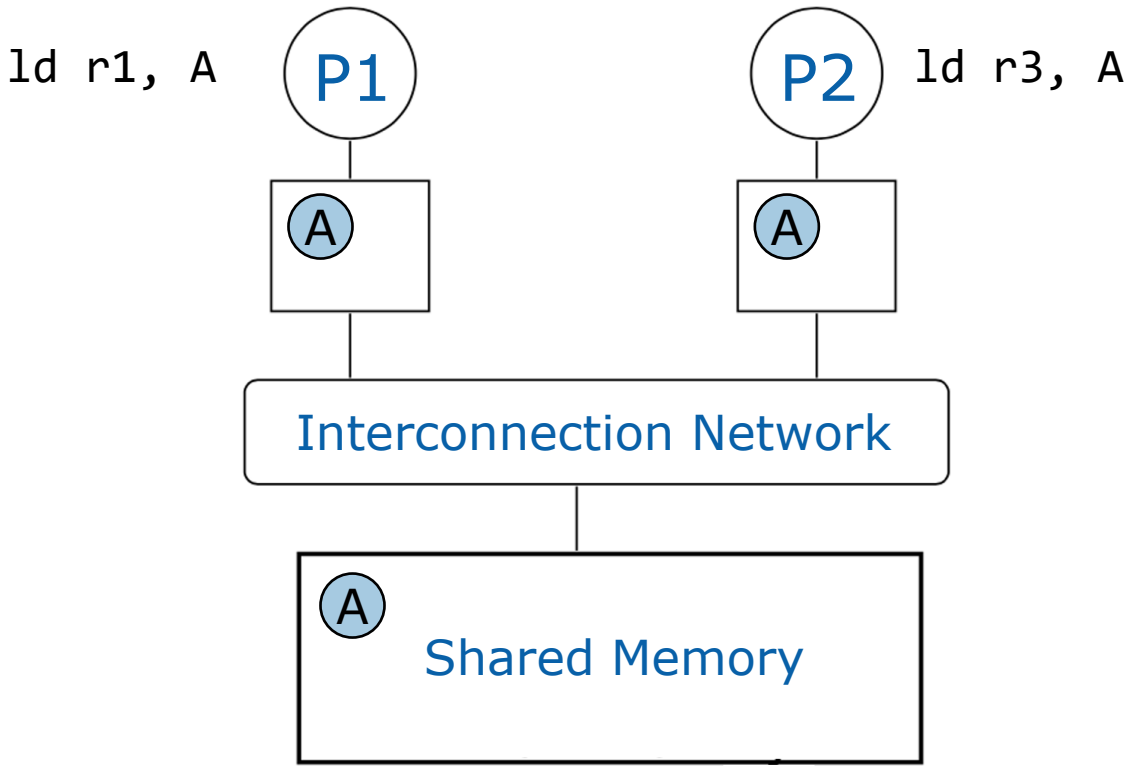
```cpp
#include <condition_variable>
std::mutex mu;
std::condition_variable cv;
bool ready = false;

void worker_thread() {
    std::unique_lock<std::mutex> lock(mu);
    cv.wait(lock, []{ return ready; });
    // Do work after 'ready' is true
}
```

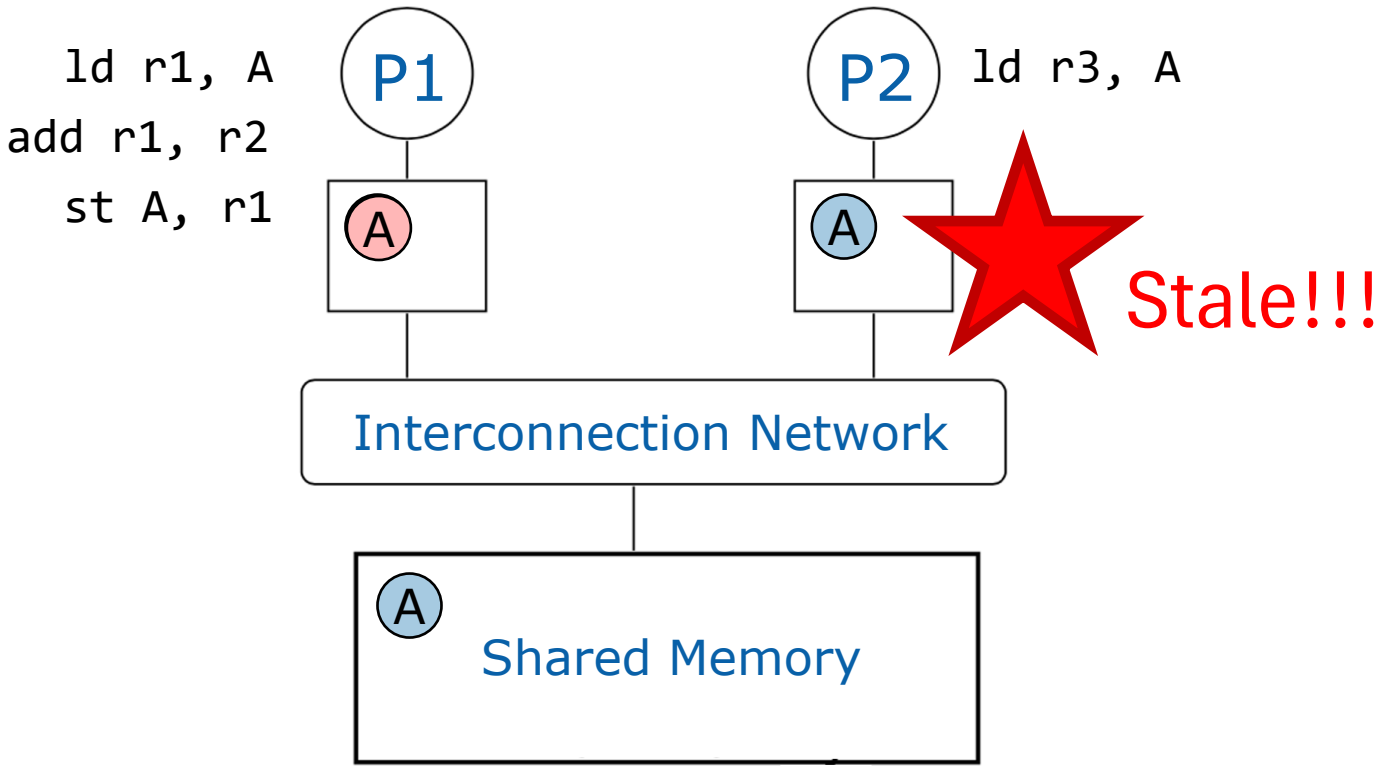
Memory/Cache Coherence

- **Coherency of shared data in multi-processor systems**
- **All processors see the same value for a particular address**
  - Any changes made to shared data in one processor's cache are promptly reflected in other processors' caches
- **Memory coherence is akin to memory consistency on one location**

Memory Coherence in Practice



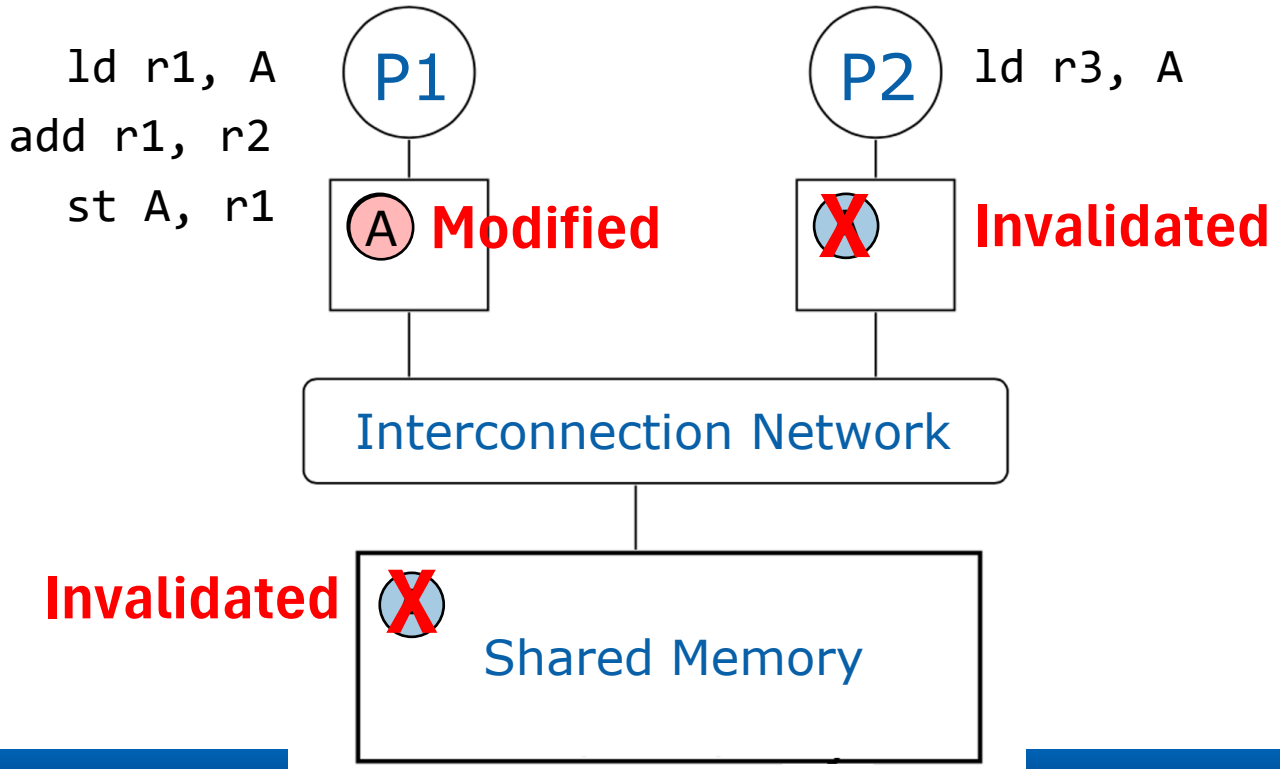
Memory Coherence in Practice



Solution: HW Coherence Protocol

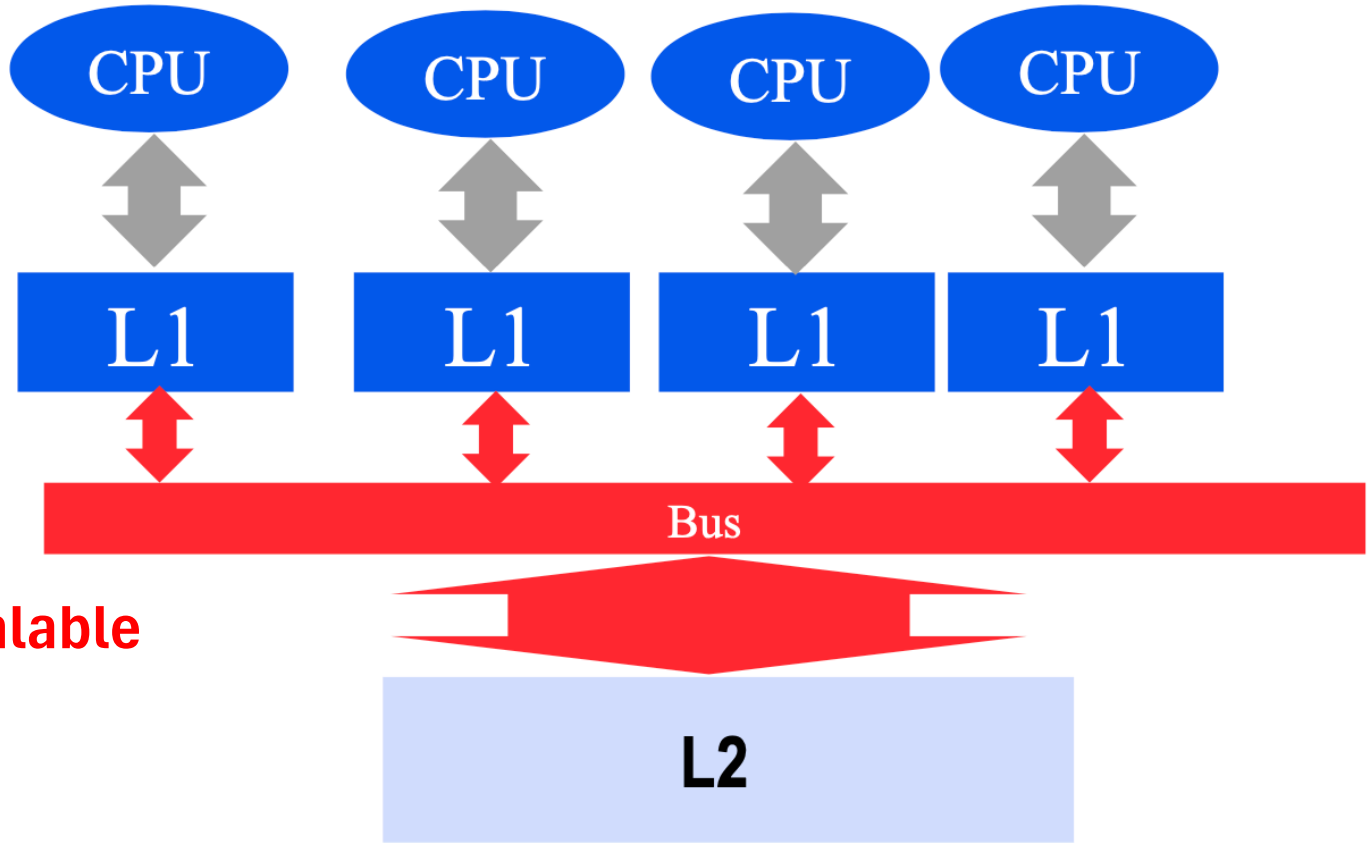
- Ensures that any changes made to data in one cache are properly reflected in all other caches that contain a copy of that data.

- Examples: MSI, MESI, MOESI



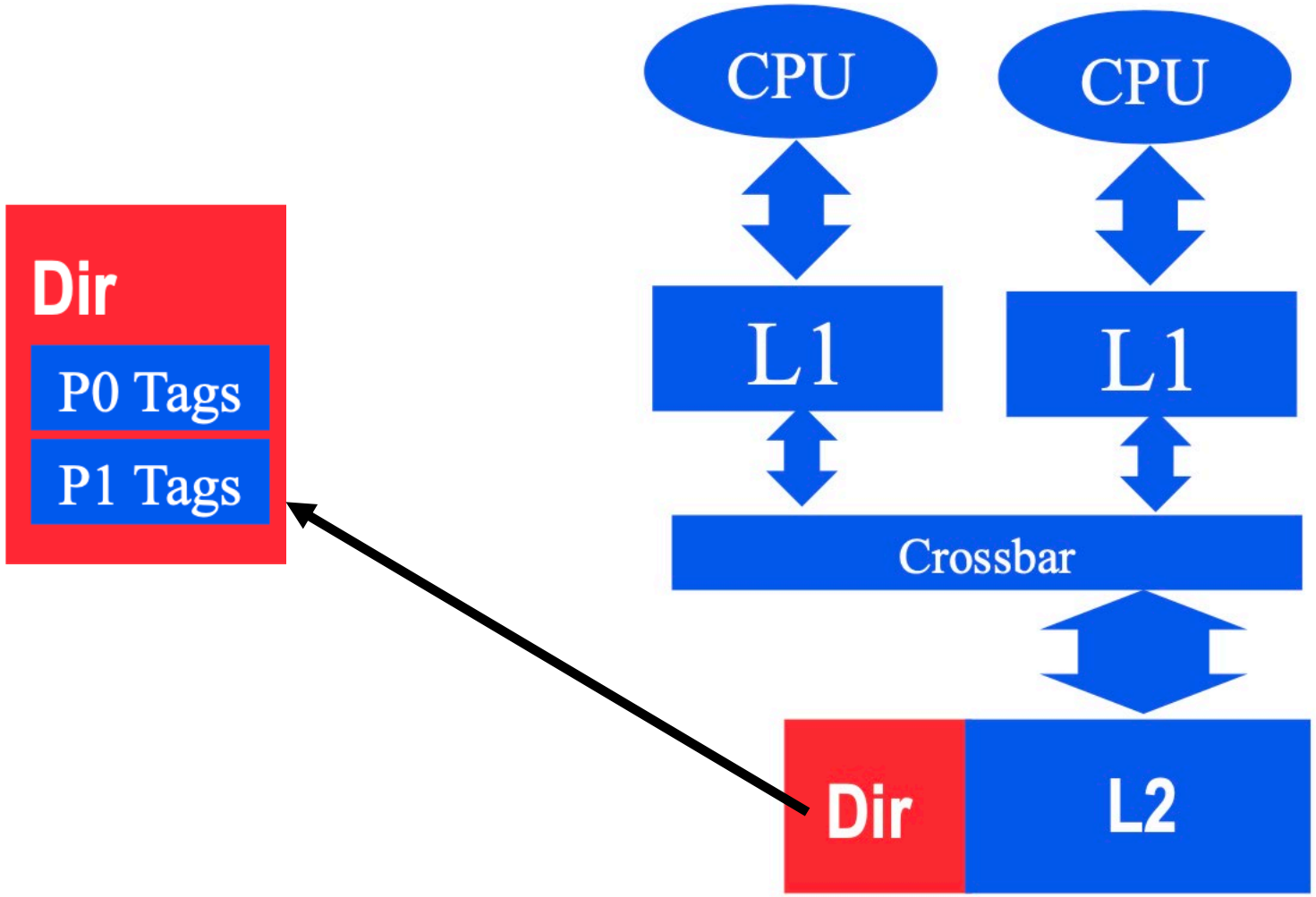
Bus-Based Implementation

- A shared bus establishes communications

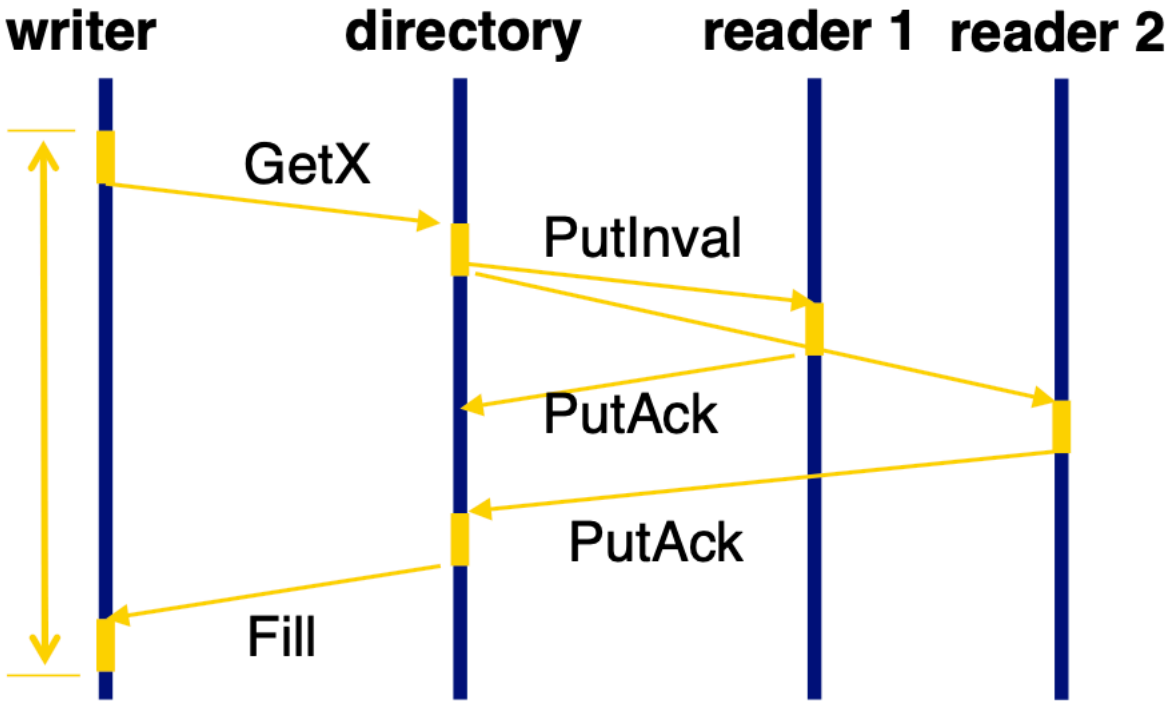


Not Scalable

Directory-Based Implementation



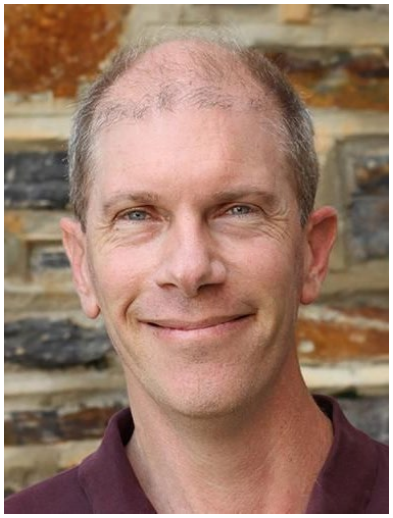
Example: Writer with Two Prior Readers



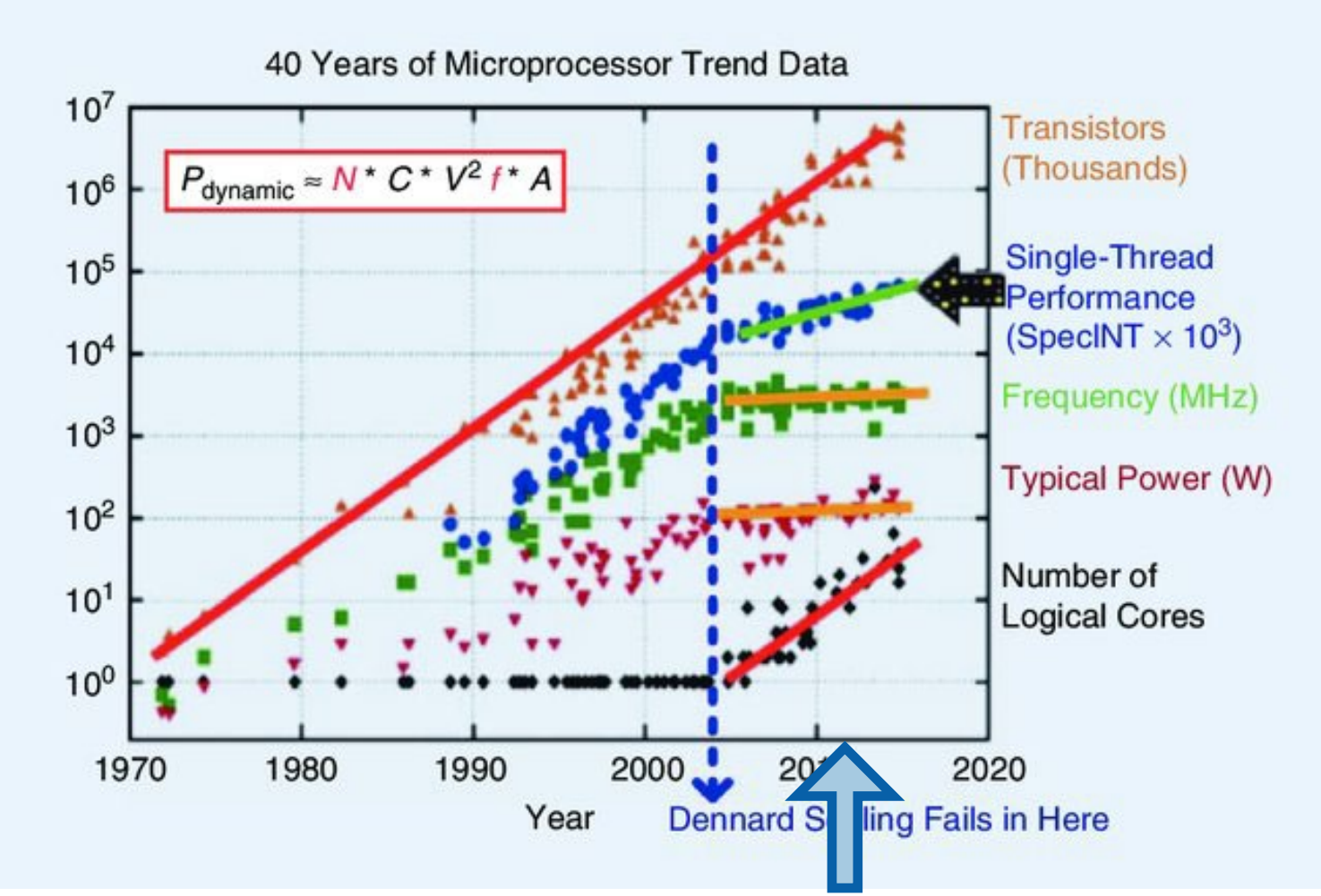
"Why On-chip Cache Coherence is Here To Stay"

Milo M.K. Martin, Mark D. Hill, Daniel J. Sorin 2012

- **Milo:** U Penn prof, now Engineering Director@Google
- **Mark:** Wisconsin prof, now emeritus
  - Eckert-Mauchly Award 2019. AAAS/ACM/IEEE Fellow
- **Daniel:** Duke prof
  - ISCA'25 PC Chair



Moore's Law w/o Dennard Scaling



"Why On-chip Cache Coherence is Here To Stay"

Milo M.K. Martin, Mark D. Hill, Daniel J. Sorin 2012

- **Hardware-based cache coherence will remain.**
  - Get used to it!
  - Despite prevailing sentiment that it's just too costly.
- **Alternative approaches will continue to exist, but on-chip coherence will continue to dominate in mainstream multicore chips.**
- **Forcing software to use software-managed coherence or explicit message passing only shifts complexity from hardware to software.**

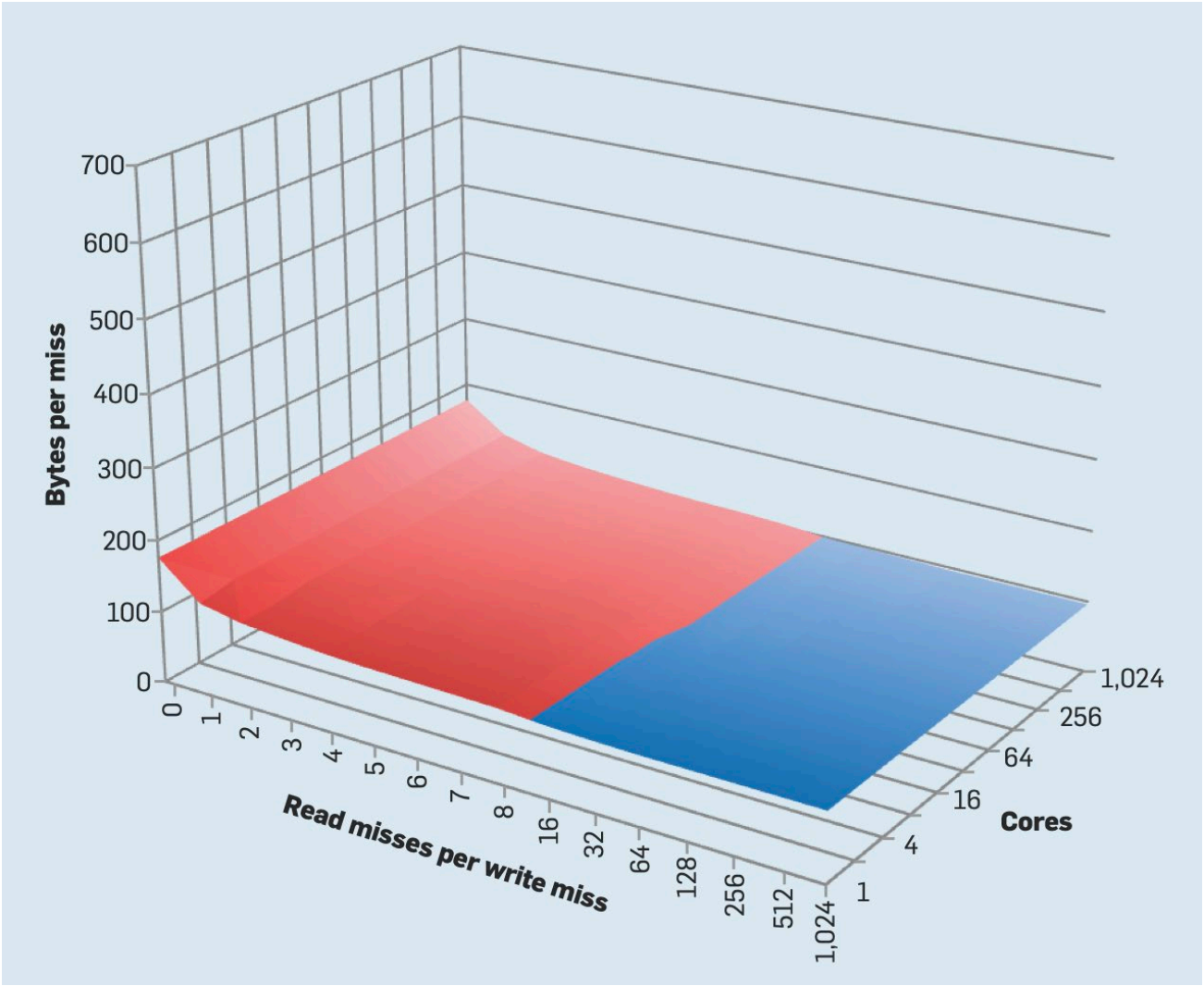
Discussion: Summary Question #1

What Did the Paper Get Right?

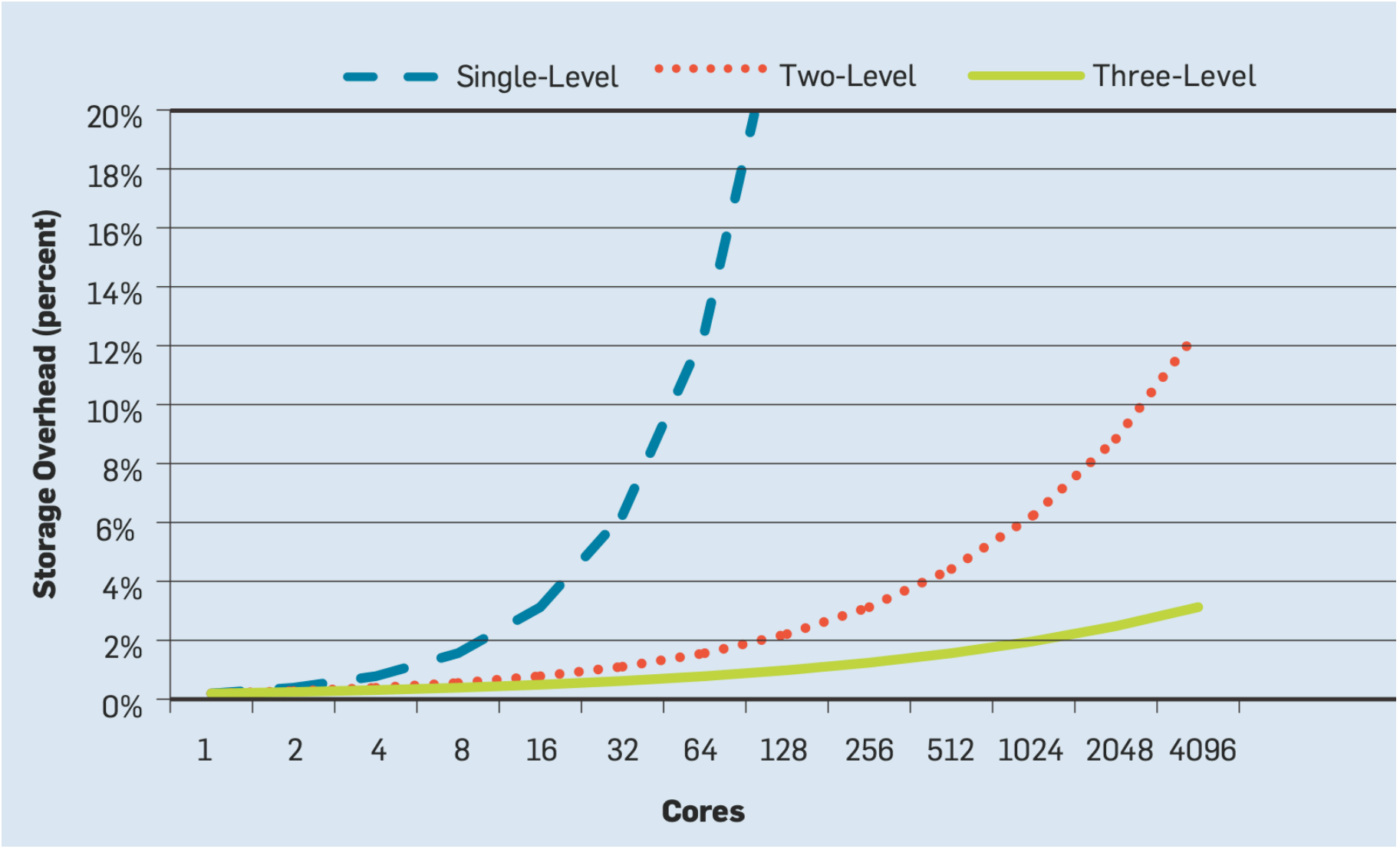
State the 3 most important things the paper says.

These could be some combination of the motivations, observations, interesting parts of the design, or clever parts of the implementation.

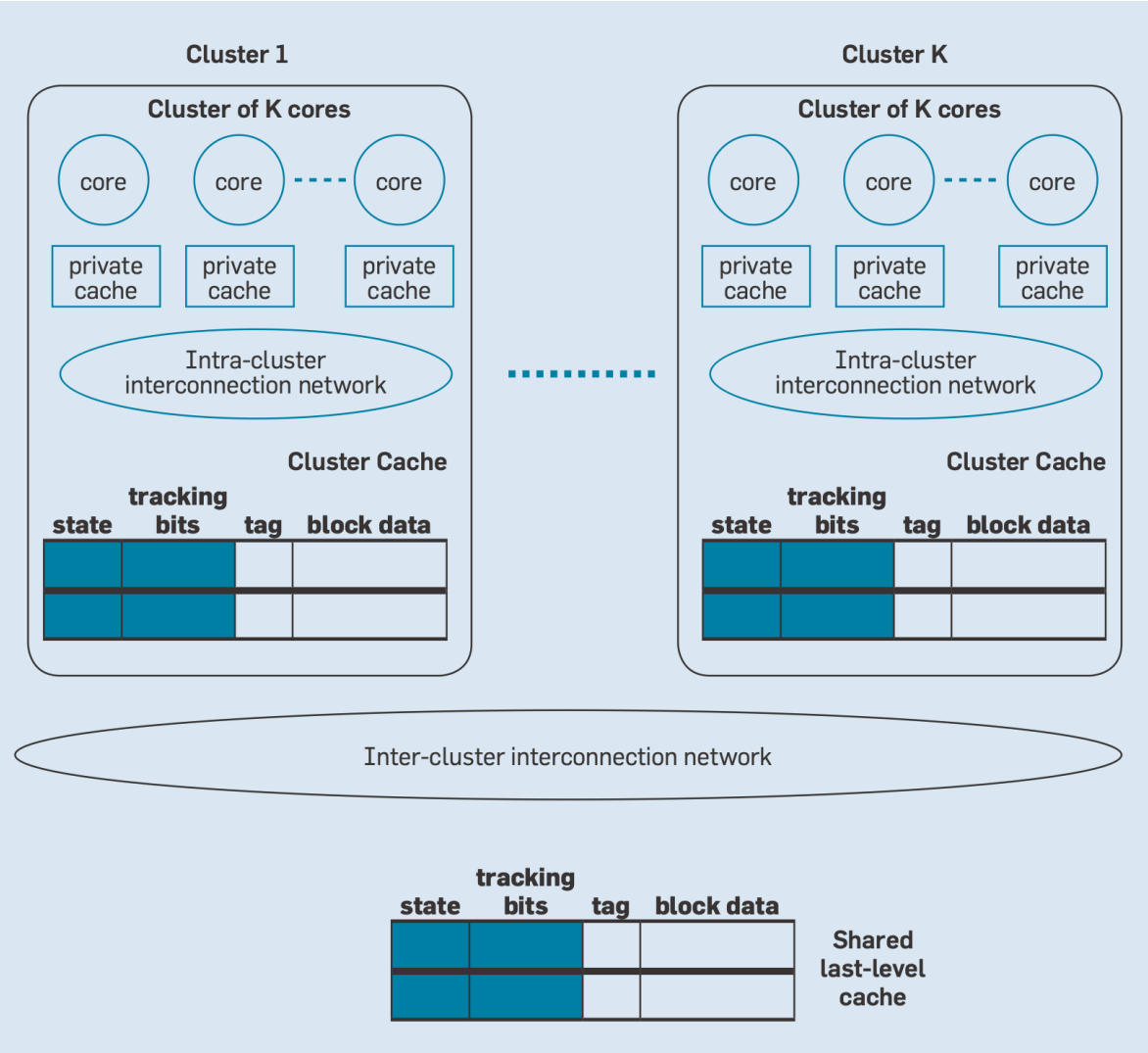
Traffic Overhead



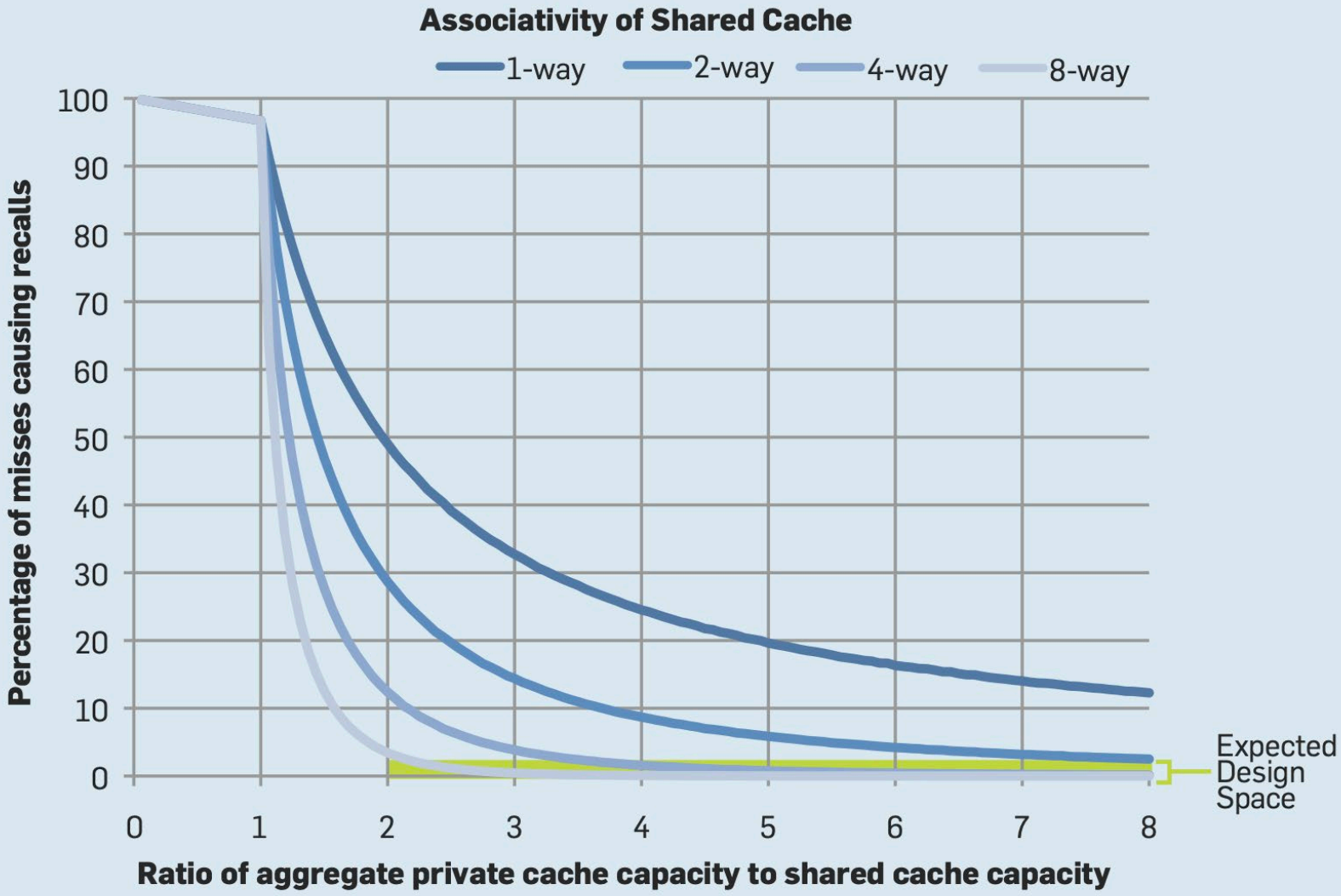
Storage Overhead



Hierarchical?



Invalidations



Discussion: Summary Question #2

What Did the Paper Get Wrong?

Describe the paper's single most glaring deficiency.

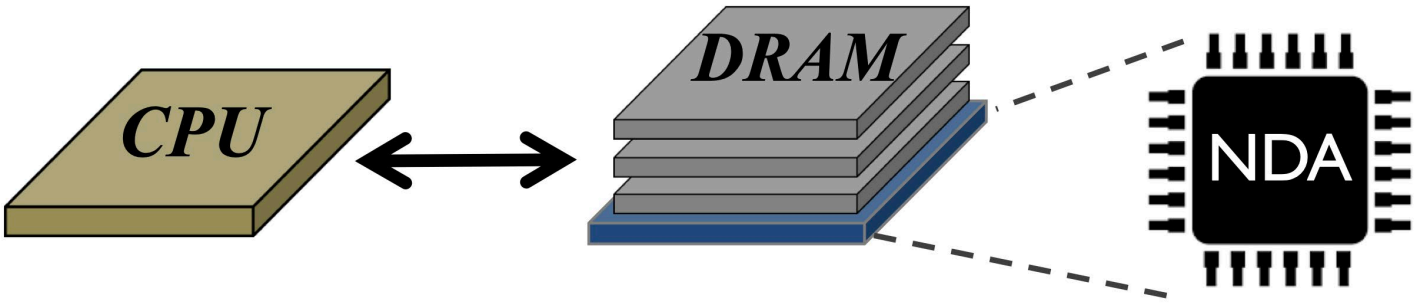
Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Other Scalability Issues?

- Latency?
- Energy?

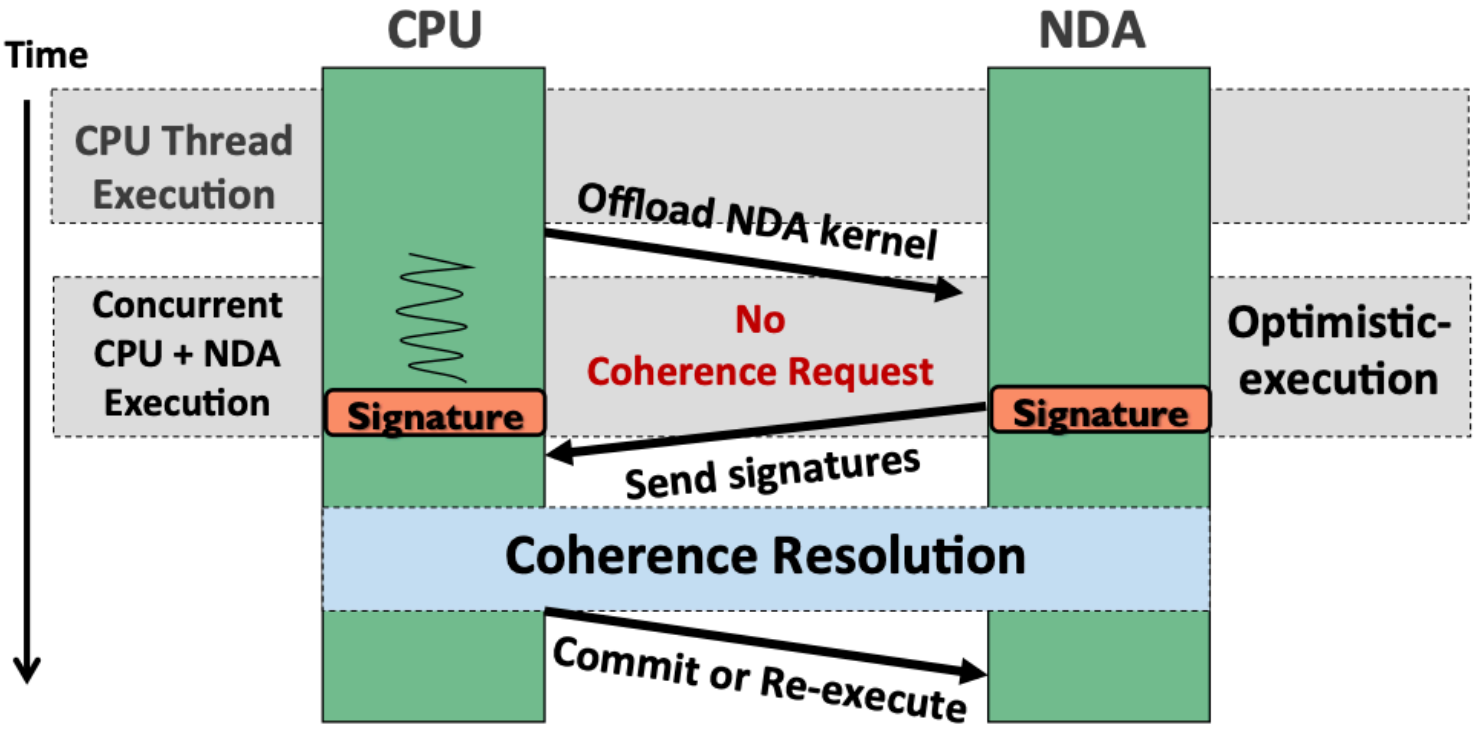
“CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators”

Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T. Malladi, Hongzhong Zheng, Onur Mutlu 2019



"CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"

Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T. Malladi, Hongzhong Zheng, Onur Mutlu 2019



To Read for Friday

"Cohmeleon: Learning-Based Orchestration of Accelerator Coherence in Heterogeneous SoCs"

Joseph Zuckerman, Davide Giri, Jihye Kwon, Paolo Mantovani, Luca P. Carloni 2021

Optional Further Reading:

"Cuckoo Directory: A Scalable Directory for Many-Core Systems"

Michael Ferdman, Pejman Lotfi-Kamran, Ken Balet, Babak Falsafi 2011