TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Distributed Operating Systems
Cache Coherence

Marcus Völp

(slides Julian Stecklina, Marcus Völp)

SS2014

Concurrent programs

global variables:

```
int i;
int k;
```

```
i = 1;
if (i > 1) k = 3;
```

||

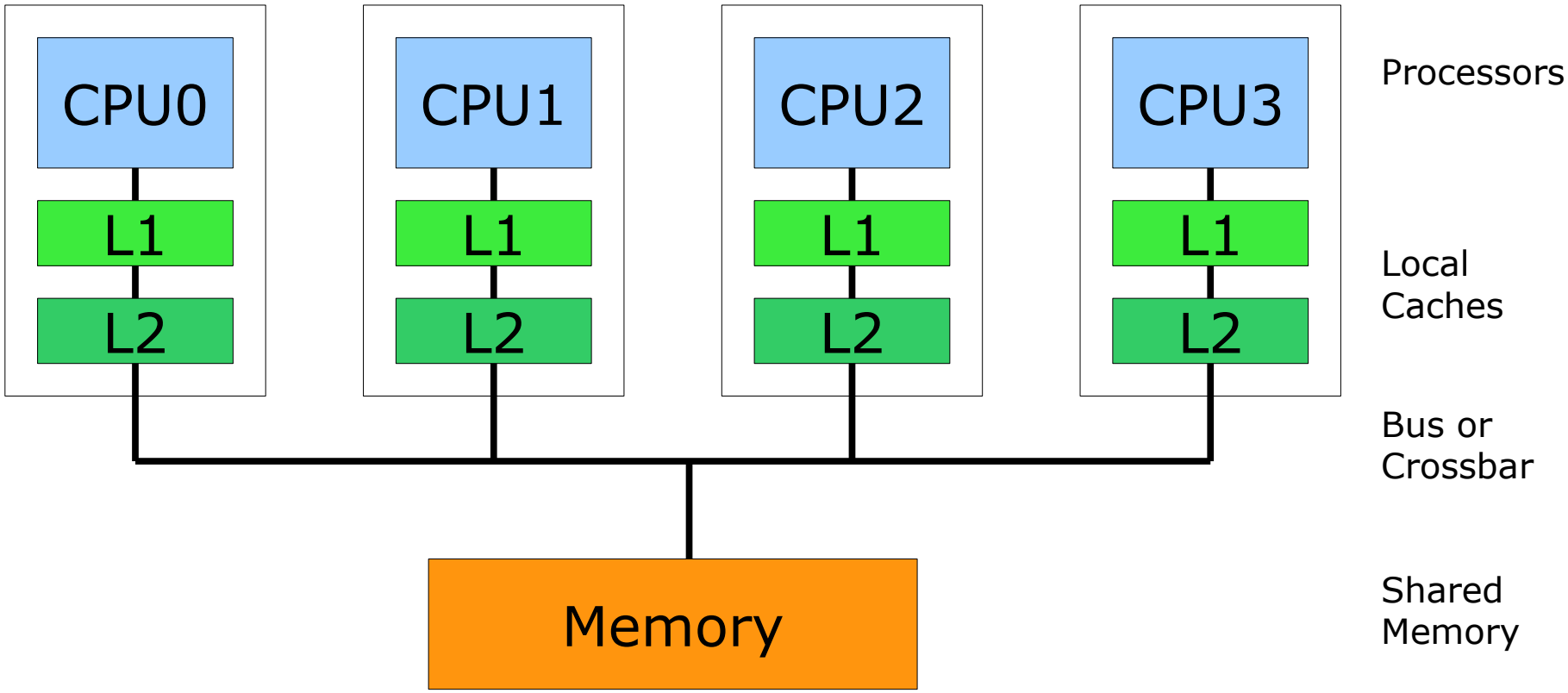
```
i = i + 1;
if (k == 0) k = 4;
```

```
mov $1, [%i]
cmp [%i], $1
jgt   end
mov $3, [%k]
end:
```

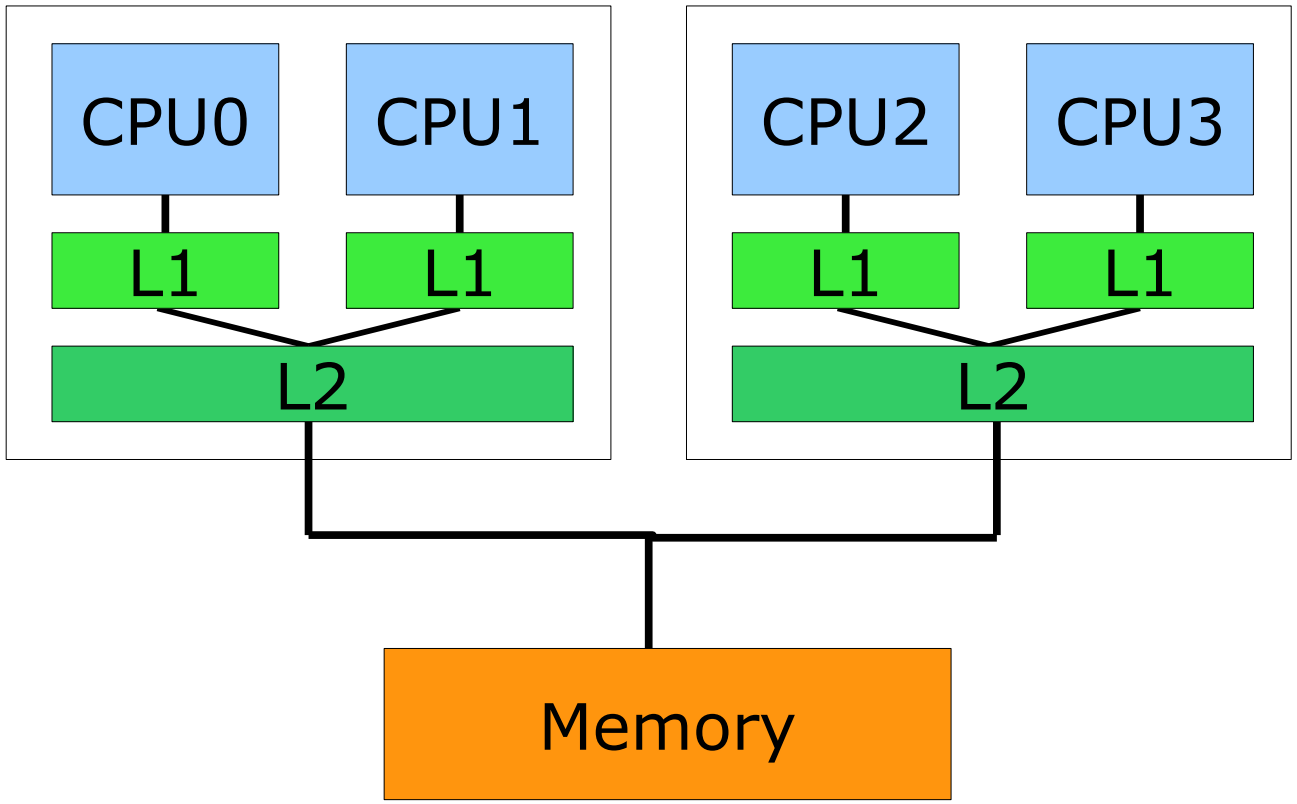
||

```
lock;inc   [%i]
     cmp [%k], $0
     jne   end
     mov $4, [%k]
     end:
```

Symmetric Multiprocessor (SMP)



Chip Multiprocessor (CMP), Multicore



Processors

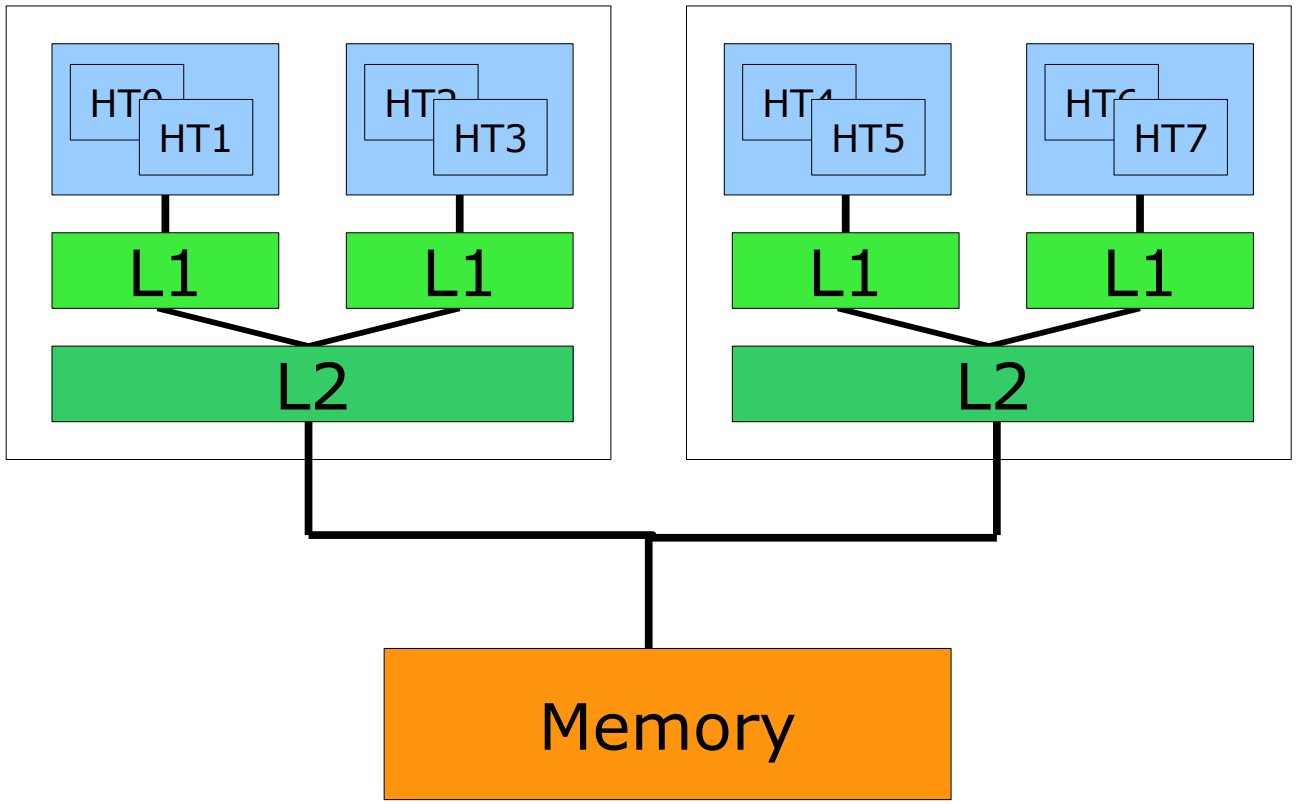
Local Cache

Shared LL-Cache

Bus or Crossbar

Shared Memory

Symmetric Multithreading (SMT), Hyperthreading



Processors

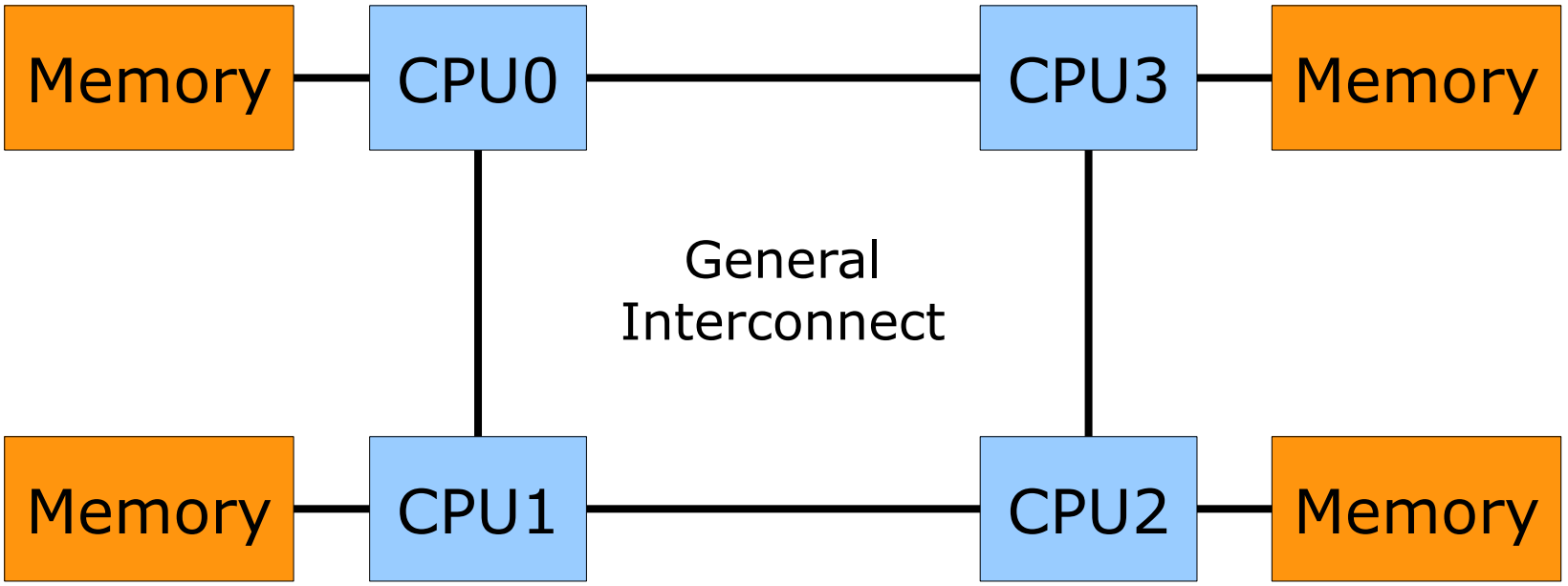
Local Cache

Shared LL-Cache

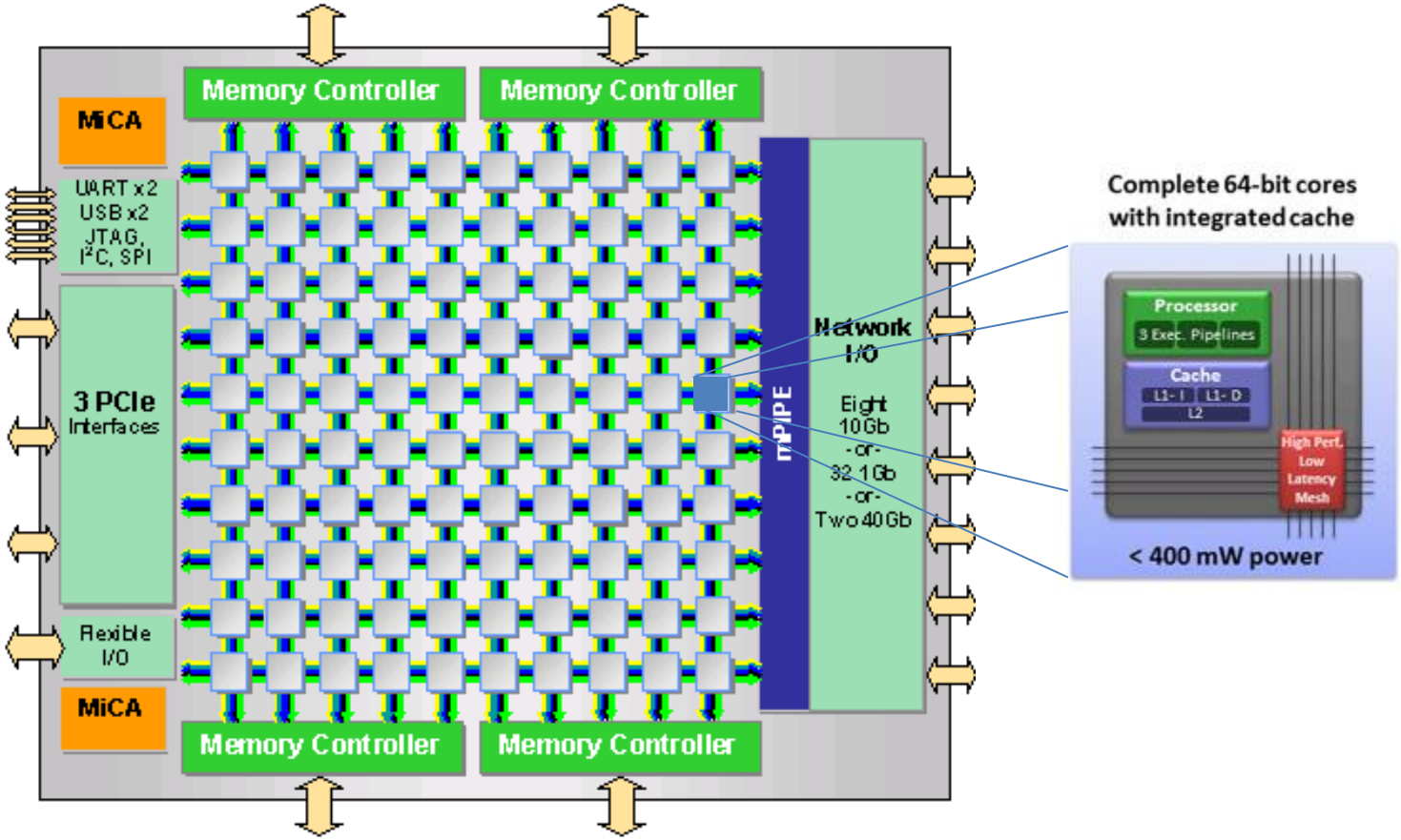
Bus or Crossbar

Shared Memory

Non-Uniform Memory Access (NUMA)



NUMA Example: Tilera Gx

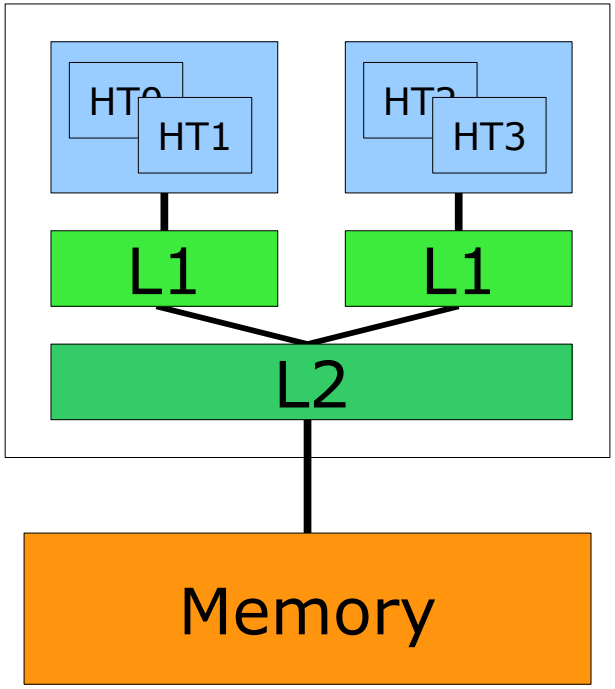


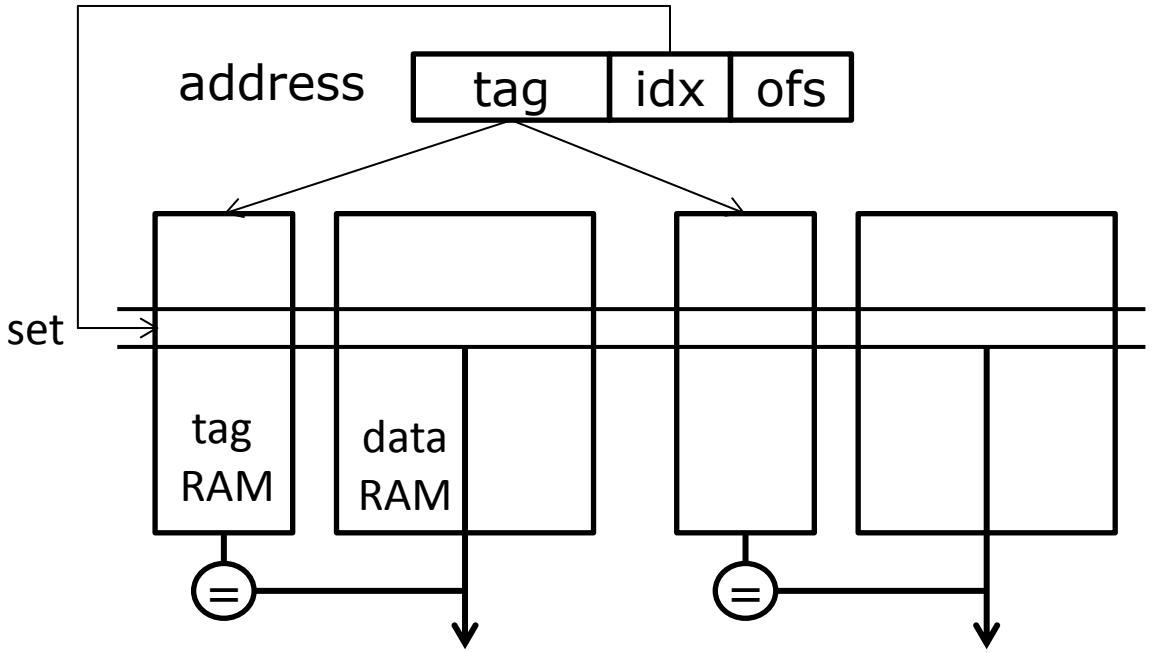
Source: http://tilera.com

Summary: Memory Organization

- Multiple processors share memory
- Memory access paths through one or more controllers
  - UMA (Uniform Memory Access)
  - NUMA (Non-Uniform Memory Access)
- Caches / store buffers hold memory content near accessing CPUs.

Cache Coherency





Cache Coherency

- Caches lead to multiple copies for the content of a single memory location
- Cache Coherency keeps copies "consistent"
  - locate all copies
  - invalidate/update content
- **Write Propagation**
  writes must eventually become visible to all processors.
- **Write Serialization**
  every processor should see writes to the **same** location in the same order.

Alternative Definition: SWMR

Single-Writer, Multiple-Reader Invariant

For any memory location A, at any given time,

either only a single core may write (or read-modify-write) the content of A

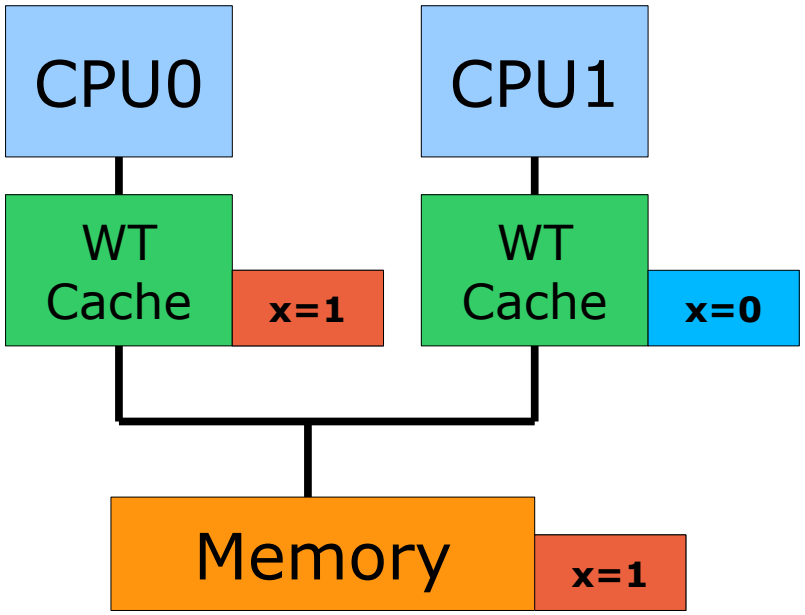
or any number of cores may read the content of A.

Data-Value Invariant

The value of a memory location at the start of an operation is the same as the value at the end of its ***last*** write (read-modify-write) operation.

[based on Sorin et al., 2011]

Attempt 1: write through all caches



CPU0: read x
x=0 stored in cache

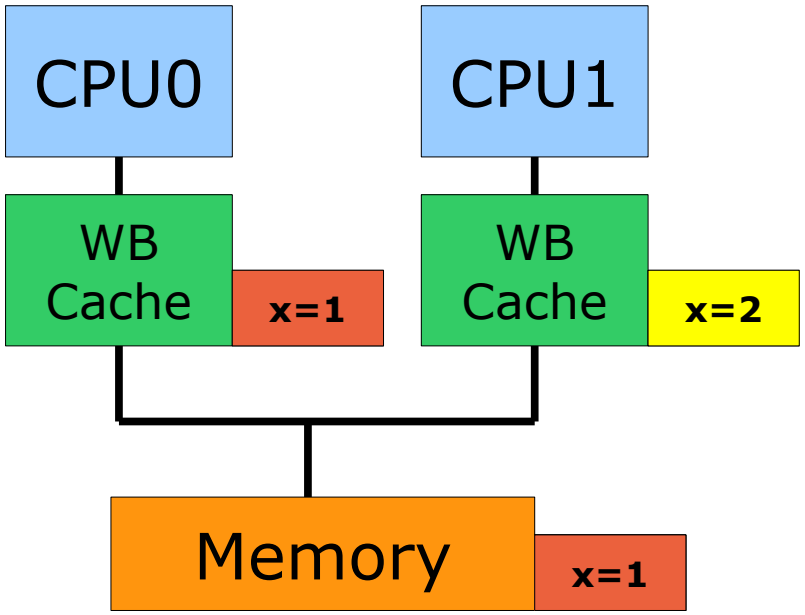
CPU1: read x
x=0 stored in cache

CPU0: write x=1
x=1 stored in cache
x=1 stored in memory

CPU1: read x
x=0 retrieved from cache

Write not visible to CPU1!

Attempt 2: write back



Later store x=2 lost!

CPU0: read x
x=0 stored in cache

CPU1: read x
x=0 stored in cache

CPU0: write x=1
x=1 stored in cache

CPU1: write x=2
x=2 stored in cache

CPU1: writeback
x=2 stored in memory

CPU0: writeback
x=1 stored in memory

Coherency Problems & Solutions

Both examples violate SWMR!

Problem 1

CPU1 used stale value that had already been modified by CPU0.

- Solution: Invalidate copies before write proceeds!

Problem 2

Incorrect writeback order of modified cache lines.

- Solution: Disallow more than one modified copy!

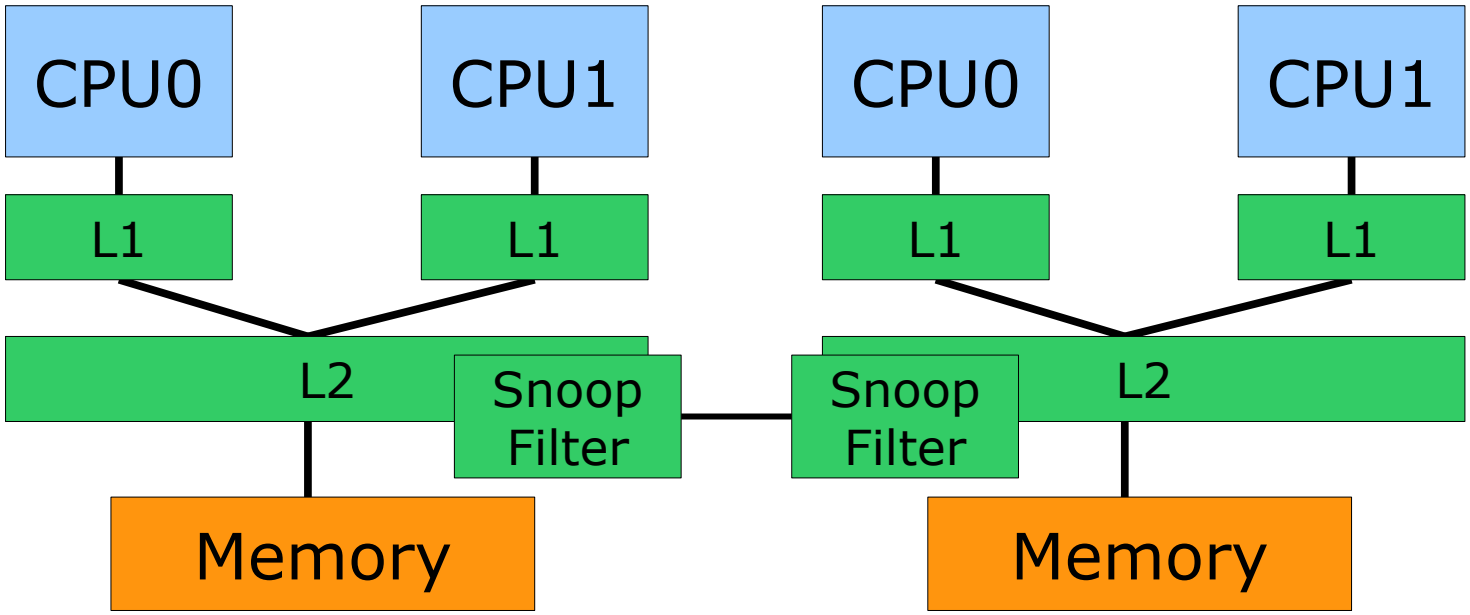
Coherency Protocol Design Space

- **Snooping-based**
  - All coherency related traffic broadcasted to all CPUs
  - Each processor snoops and acts accordingly:
    - Invalidate lines written by other CPUs
    - Signal sharing for lines currently in cache
  - Straightforward for bus-based systems
  - Suited for small-scale systems

- **Directory-based**
  - Uses central directory to track cache line owner
  - Update copies in other caches
    - Can update all CPUs at once
      (less traffic for alternating reads and writes)
    - Multiple writes need multiple updates
      (more traffic for subsequent writes)
  - Suited for large-scale systems

Coherency Protocol Design Space

- **Snooping-based vs. Directory-based**



Invalidation vs. Update Protocols

- **Invalidation-based**
  - Only write misses hit bus (suited for WB caches)
  - Subsequent writes are write hits
  - Good for multiple writes to same cache line by same CPU
- **Update-based**
  - All shares of a cache line continue to hit in the cache after a write by one CPU
  - Otherwise lots of useless updates (wastes bandwidth) → Rarely used!

- Hybrid forms are possible!

A Basic Coherency Protocol: MSI

- Modified (M)
  - No copies on other caches; local copy modifed
  - Memory is stale
- Shared (S)
  - Unmodified copies in one or more caches
  - Memory is up-to-date
- Invalid (I)
  - Not in cache

- States tracked from the view of the cache controller. Sees events from:
  - Local processor → processor transactions
  - Other processors → snoop transactions

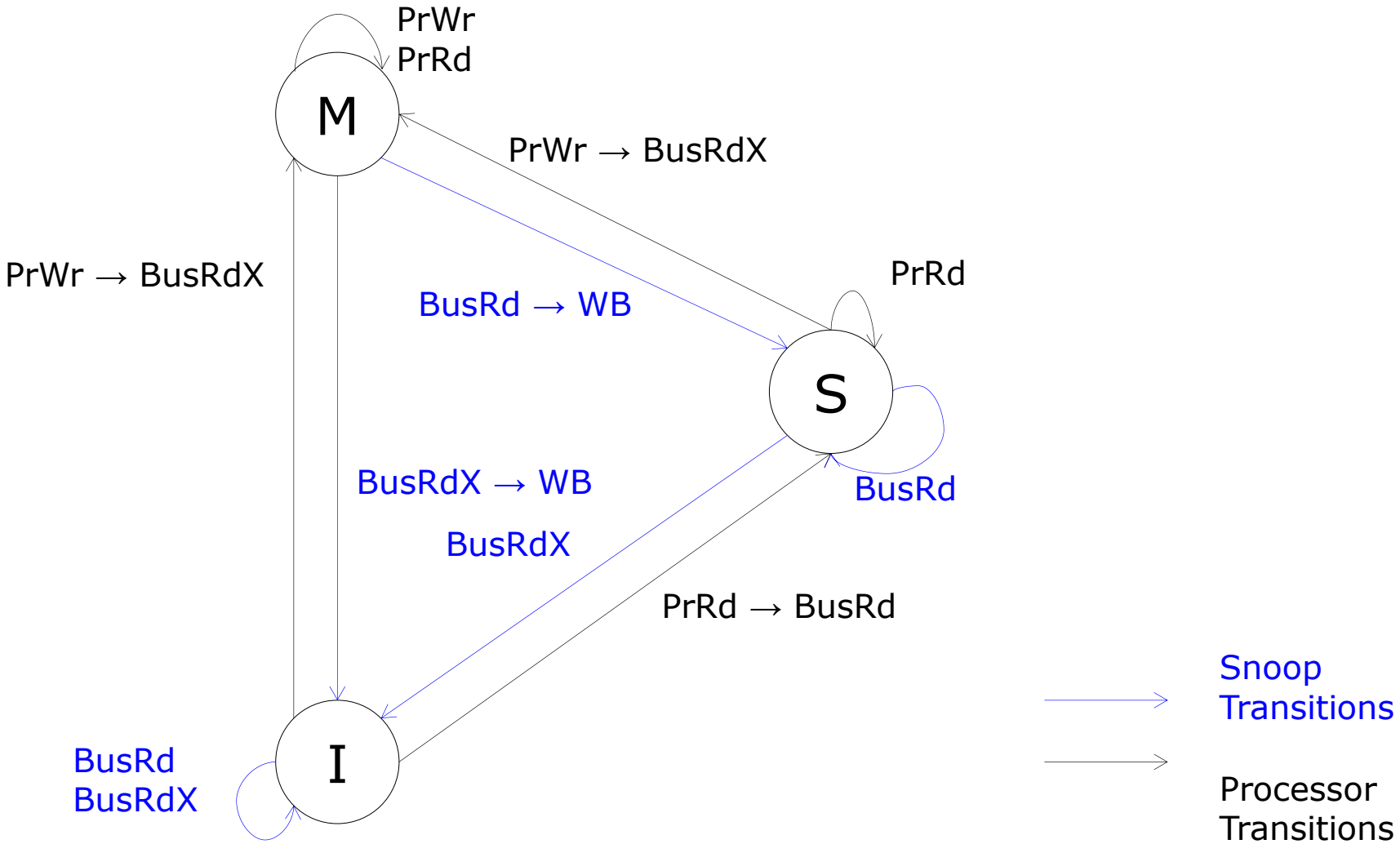
MSI: Processor Transitions

- State is I, CPU reads (PrRd)
  - Generate bus read request (BusRd)
  - Go to S
- State is S or M, CPU reads (PrRd)
  - No transition
- State is S, CPU writes (PrWr)
  - Upgrade cache line for exclusive ownership (BusRdX)
  - Go to M
- State is M, CPU writes (PrWr)
  - No transition

MSI: Snoop Transitions

- Receiving a read snoop (BusRd) for a cache line
  - If M, write cache line back to memory (WB), transition to S
  - If S, no transition
- Receiving a exclusive ownership snoop (BusRdX)
  - If M, write cache line back to memory (WB), discard it, transition to I
  - If S, discard cache line, transition to I

MSI State Transitions



Problems in MSI

A common usecase is to:

- read variable A: S
- Modify A: BusRdX sent, S → M

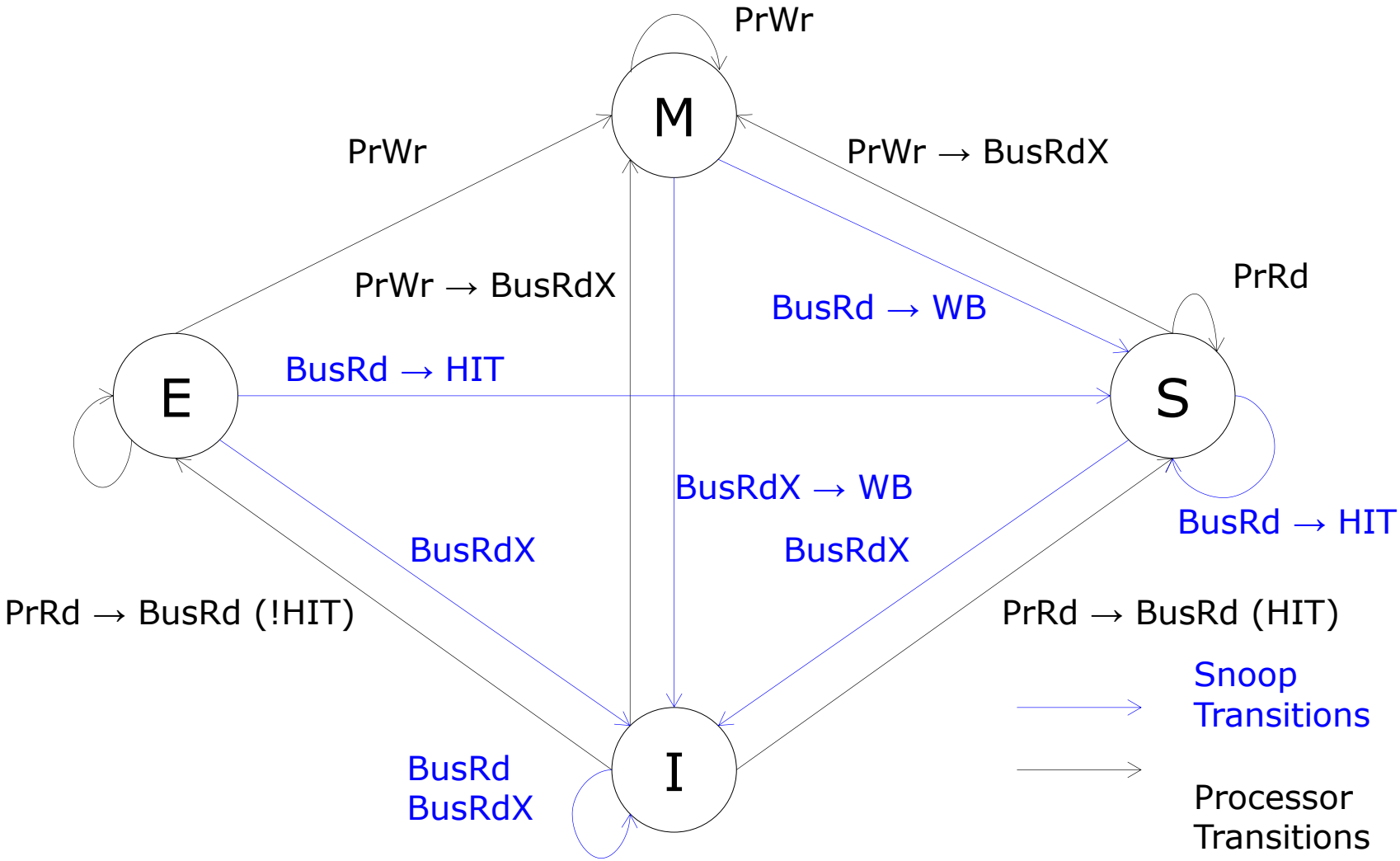
Invalidation message pointless, if no other cache holds A.

Solved by adding Exclusive (E) state:

- No copies exist in other caches
- Memory is up-to-date

Variants of MESI are used by most popular microprocessors.

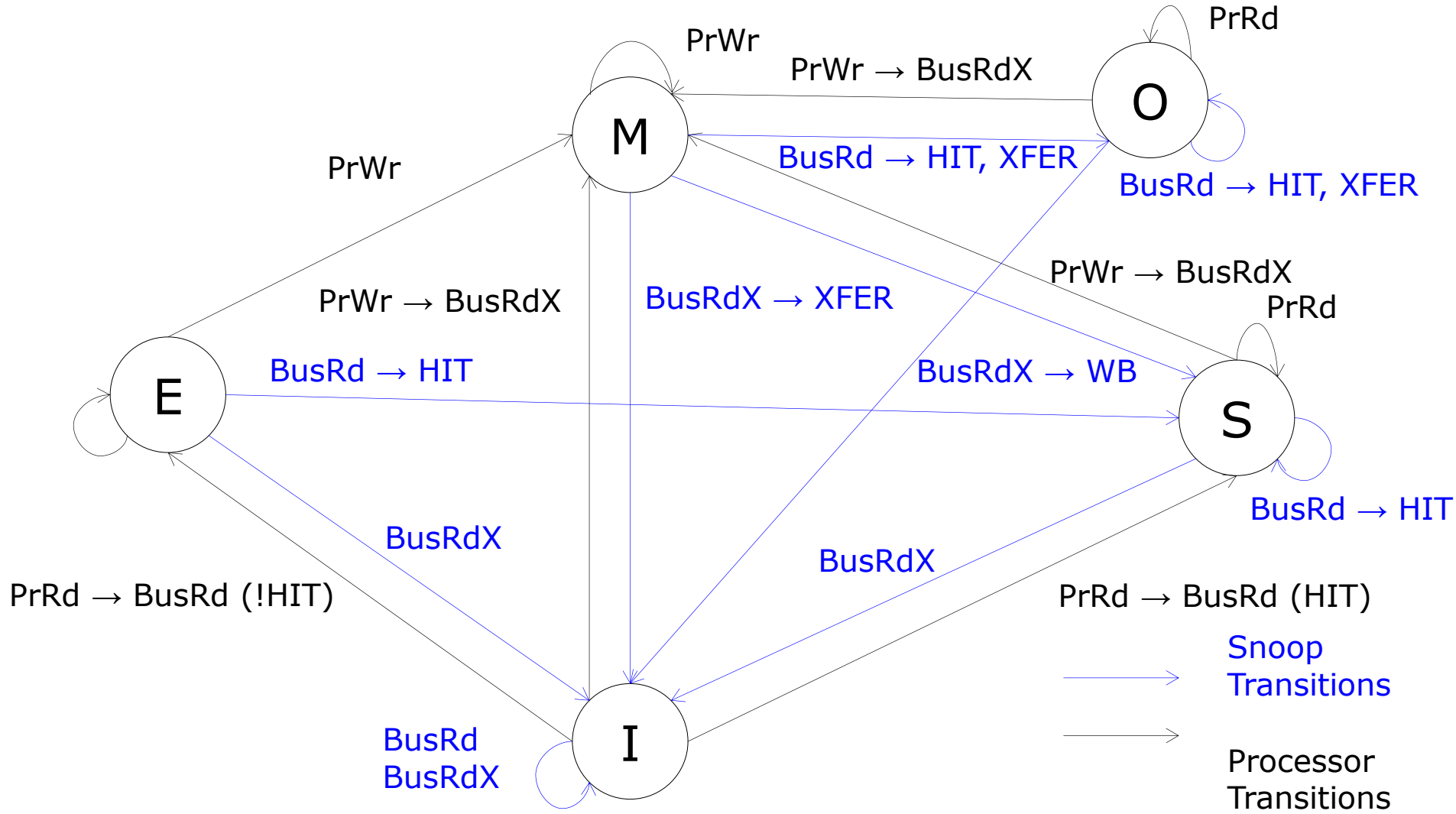
MESI State Transitions



MOESI: Adding Owned to MESI

- Similar to MESI, with some extensions
- Cache-to-Cache transfers of modified cache lines
  - Modified cache lines not written back to memory, but supplied by to other CPUs on BusRd
  - CPU that had initial modified copy becomes “owner”
- Avoids writeback to memory when another CPU accesses cache line
  - Beneficial when cache-to-cache latency/bandwidth is better than cache-to-memory latency/bandwidth
- Used by AMD Opteron

MOESI State Transitions



Coherency in Multi-Level Caches

- Bus only connected to last-level cache (e.g. L2)
  - Snoop requests are relevant to inner-level caches (e.g. L1)
  - Modifications in L1 may not be visible to L2 (and the bus)
- Idea: L2 forwards filters transactions for L1:
  - On BusRd check if line is M/O in L1 (may be S or E in L2)
  - On BusRdX, send invalidate to L1
- Only easy for inclusive caches!

- **Inclusion property**

  Outer cache contains a superset of the content of its inner caches.

References

- A Primer on Memory Consistency and Cache Coherence
  Sorin, Hill, Wood; 2011
- atomic<> Weapons: The C++ Memory Model and Modern Hardware (Video)
  Sutter; 2013
- Shared memory consistency models: a tutorial
  Adve, Gharachorloo; 1996
- IA Memory Model
  Richard Hudson; Google Tech Talk 2008
- Memory Ordering in Modern Microprocessors
  McKenney; Linux Journal 2005
- How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs
  Lamport, 1979
- PowerPC Storage Model

References

Scheduler-Conscious Synchronization
Leonidas Kontothanassis, Robert Wisniewski, Michael Scott

Scalable Reader- Writer Synchronization for Shared-Memory Multiprocessors
John M. Mellor-Crummey, Michael L. Scottt

Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors
John Mellor-Crummey, Michael Scott

Concurrent Update on Multiprogrammed Shared Memory Multiprocessors
Maged M. Michael, Michael L. Scott

Scalable Queue-Based Spin Locks with Timeout
Michael L. Scott and William N. Scherer III

References

Reactive Synchronization Algorithms for Multiprocessors
B. Lim, A. Agarwal

Lock Free Data Structures
John D. Valois (PhD Thesis)

Reduction: A Method for Proving Properties of Parallel Programs
R. Lipton - Communications of the ACM 1975

Decoupling Contention Management from Scheduling (ASPLOS 2010)
F.R. Johnson, R. Stoica, A. Ailamaki, T. Mowry

Corey: An Operating System for Many Cores (OSDI 2008)
Silas Boyd-Wickizer, Haibo Chen, Rong Chen, Yandong Mao, Frans Kaashoek, Robert Morris, Aleksey Pesterev, Lex Stein, Ming Wu, Yuehua Dai, Yang Zhang, Zheng Zhang