Lecture 22: Multicore – Coherence

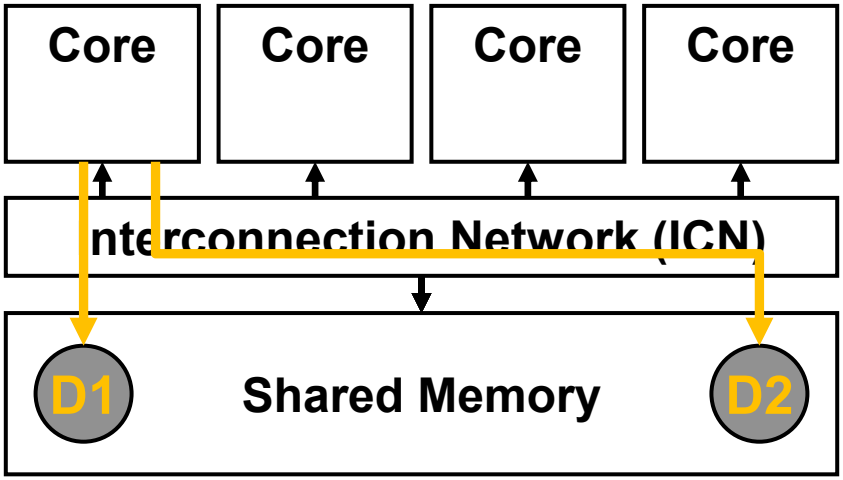
Hunjun Lee
hunjunlee@hanyang.ac.kr

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Shared Memory in CMP: UMA vs. NUMA

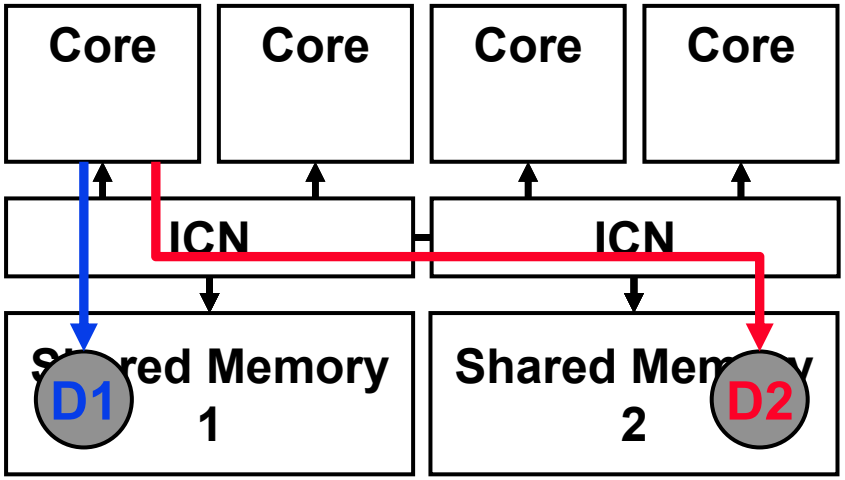
- UMA: bus-based systems where all the cores share a single bus to the shared memory
  - Fast, simple, but not scalable
- NUMA: point-to-point (or not) network-based systems
  - Slow, complex, but more scalable

Same latency for D1 and D2



Uniform Memory Access (UMA)

D2 takes longer than D1

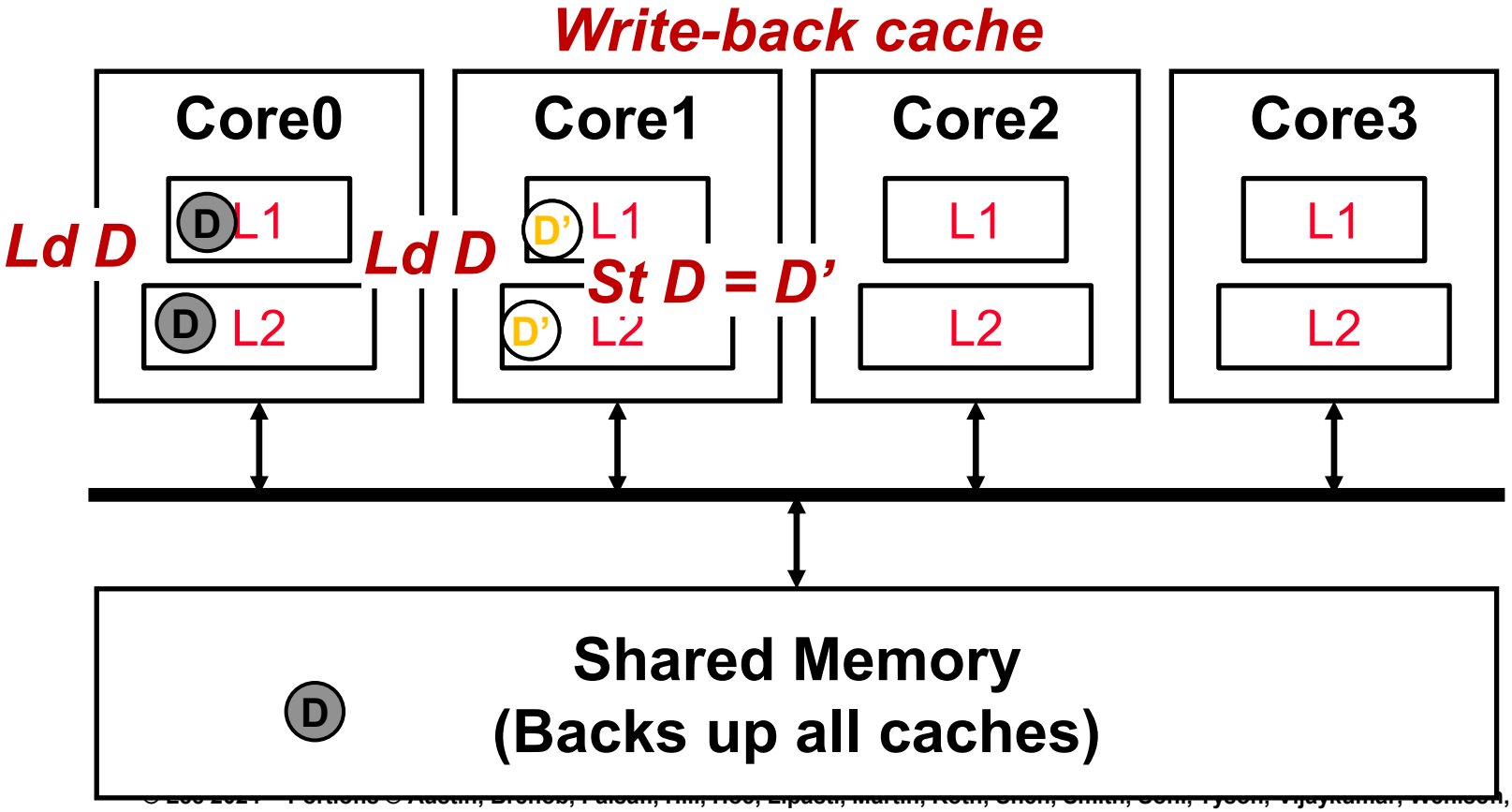


Non-Uniform Memory Access (NUMA)

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

We are missing something: Cache!

- In fact, there are local caches that are not shared among the cores ➔ The data may reside in multiple locations
- The goal of cache coherence is to make all the processors believe they are connected to the same memory



© 2007-2021 Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Solutions to Cache Coherence (CC)

- ~~**Option #1:** why not cache the shared variables?~~

- **Option #2:** Allow multiple copies, but make sure they all have the same value
  - Updates to one copy must be visible to all copies
  - We need multiple readers + writers at once

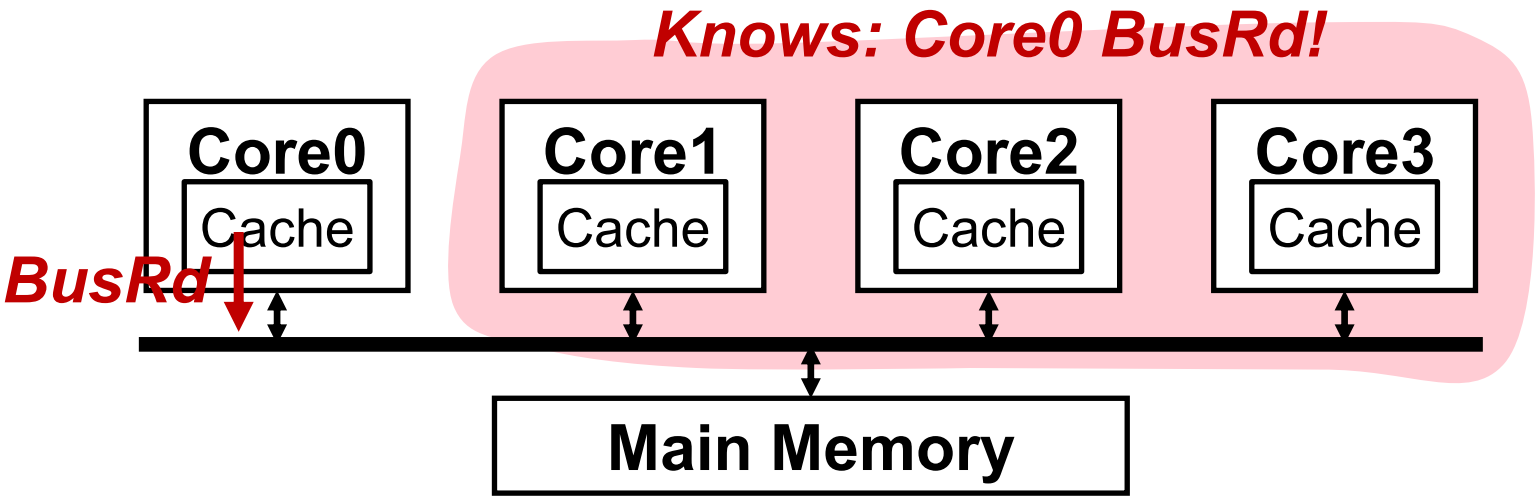
- **Option #3:** Allow only one copy of a memory location
  - If location x is cached in once cache ➔ then, it is **invalid in memory or caches**
  - Another processor must have a way to find out who has location x and **retrieve the value and ownership** to its cache
  - We can have a single reader/writer per location

A cache coherence protocol is the "rule" between caches to enforce a particular policy

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Bus-Based Systems

- Bus is a broadcast medium, bus "snooping" allows every cache to see what everyone else wants to do
- A cache can even intervene in another cache's bus transaction, e.g. a cache might ask another cache to "retry" the transaction later or respond in place of the memory
- Besides the usual status bits, additional information might have to be recorded with each cache line, aka cache coherence states, e.g.



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

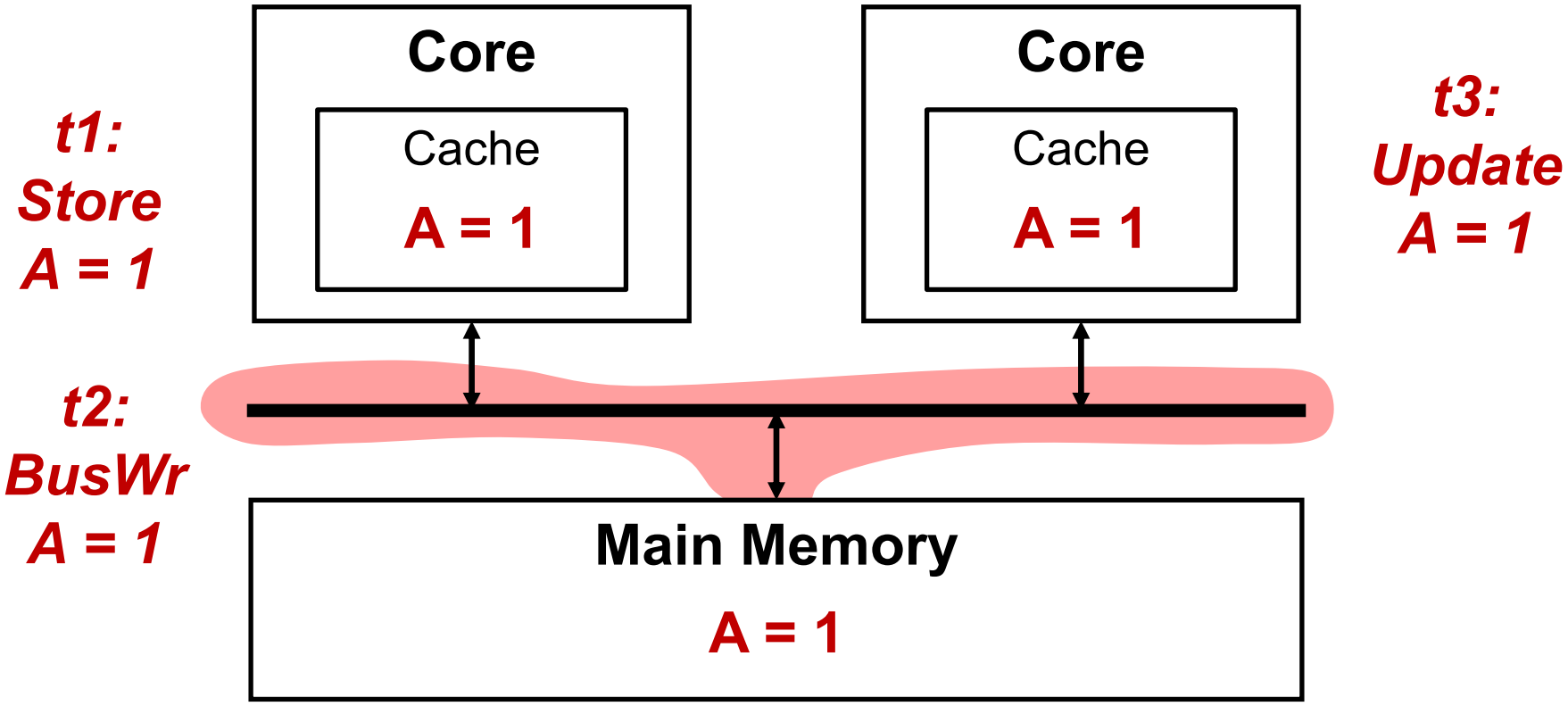
Allowing Multiple Copies (use a write-through cache)

- A cache line can be either **Valid** or **Invalid**
- Enforce a write-through scheme
  - Issues a write transaction to memory whenever the cache line is changed by a processor
  - Does not need to write back when a line is displaced
- Require two mechanisms
  - Write through ➔ a cache is always coherence with memory
  - All caches "snoop" the bus for other's write transactions
    - Check if the write is to a currently cached location
    - **Option #1:** Overwrite the old value with the snooped value (=write update coherence)
    - **Option #2:** delete the old value (=valid-invalid coherence)

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Valid-update Coherence

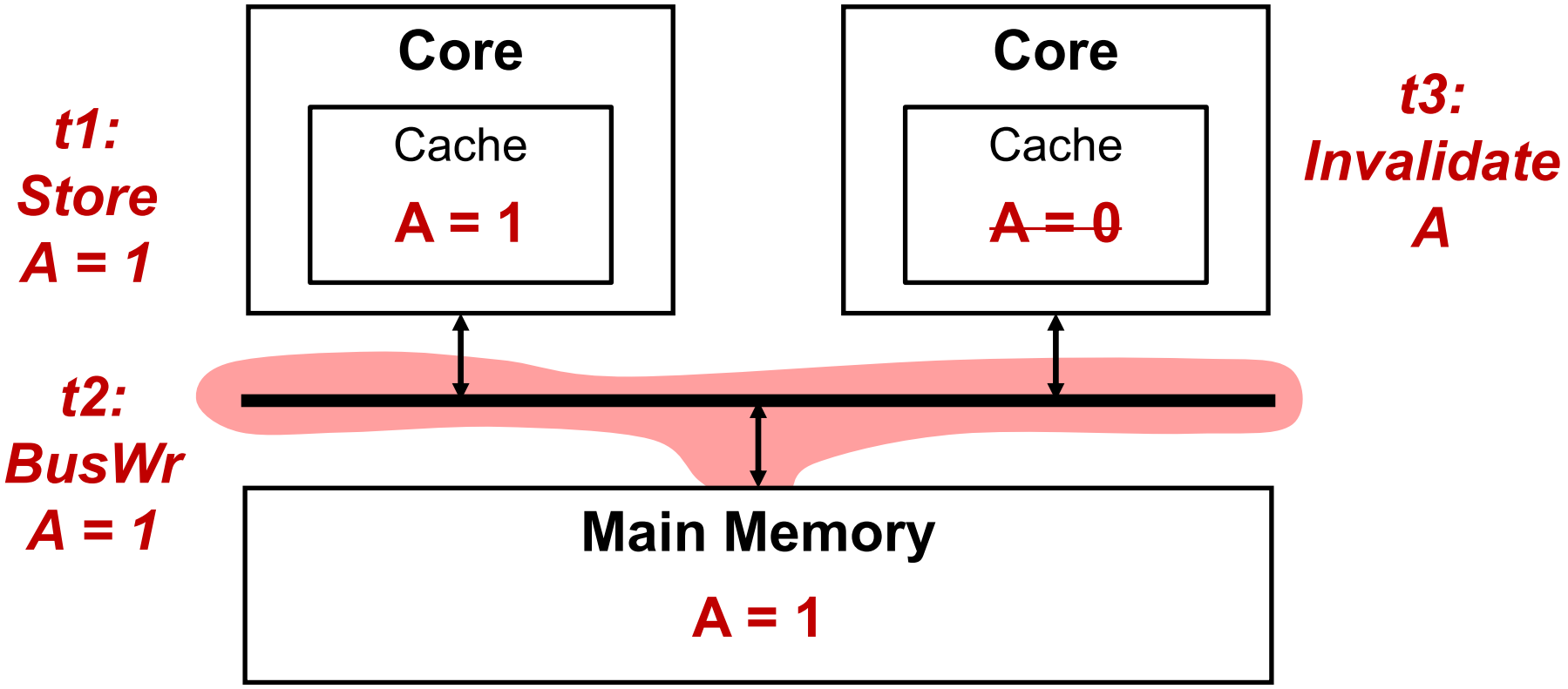
- Update all values of A in caches and memory ➔ support multiple readers & writers
- But, 15% of cache accesses are stores ➔ extremely large amount of cache writes



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Valid-invalid Coherence

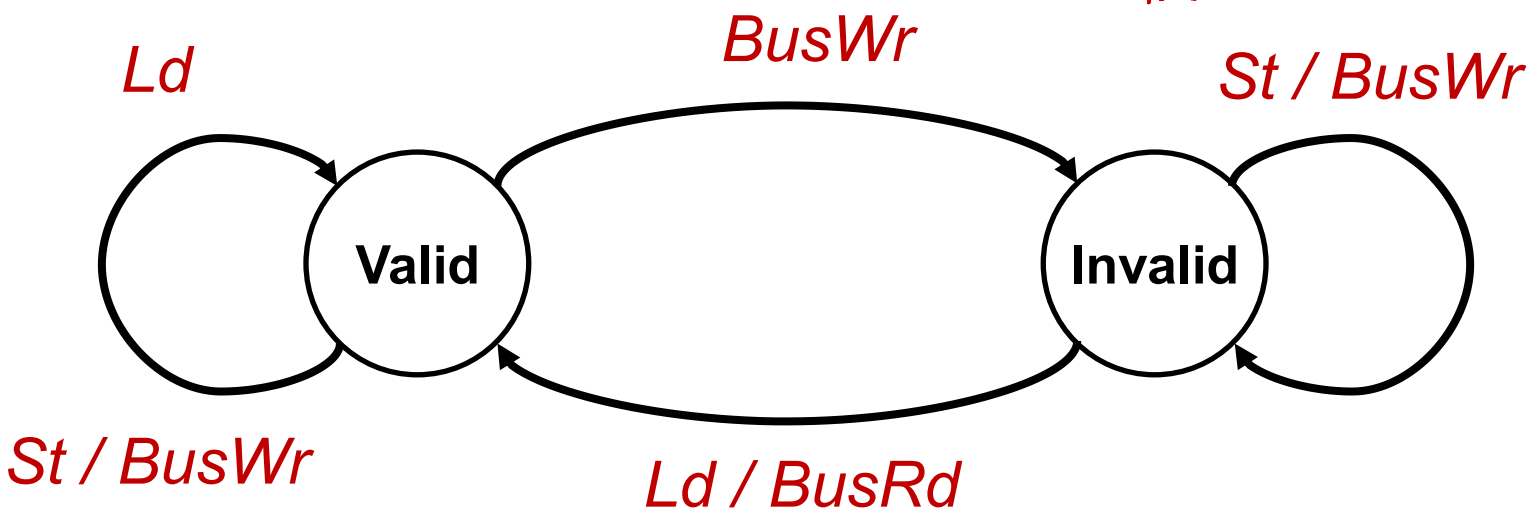
- Allows multiple readers, but **must write through to bus**
  - Write-through + no-write allocate cache
- All caches must **monitor bus traffics (i.e., "snoop")**
  - Simple state machine for each cache line



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Valid-Invalid Snooping

- Assumption: write-through & write-no-allocate cache
- 1-bit additional storage to track valid bits
- Support multiple readers
- Actions: Ld (load), St (store), BusRd (bus read signal), BudWr (bus write signal)



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Allow one copy (use a write-back cache)

- Write-through is too bandwidth intensive … (you need to access the bus for all the store operations …)

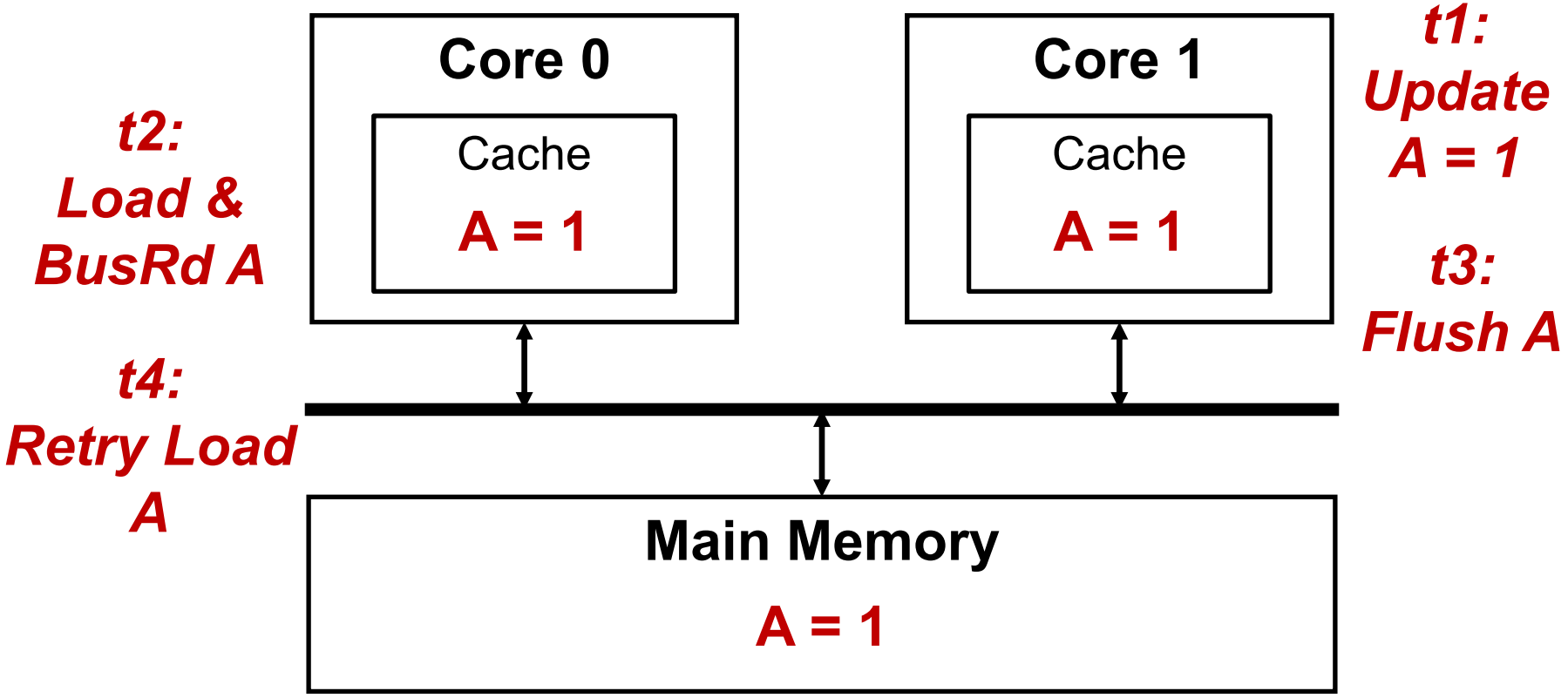
- Why not use the write-back cache w/ write allocate?
  - Cache issues a read transaction on a read or write miss
  - Cache issues a write-back to memory only when a line is displaced

- A cache line can be either Modified or Invalid
- The caches "snoop" the bus for other's read transactions
  - **Option #1:** The cache may flush and ask to retry later on …
  - **Option #2:** If a cache observes a request to a currently cached line ➔ Then, respond with a value cache and mark itself "invalid"

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

One-Copy Mechanism: Flush & Retry

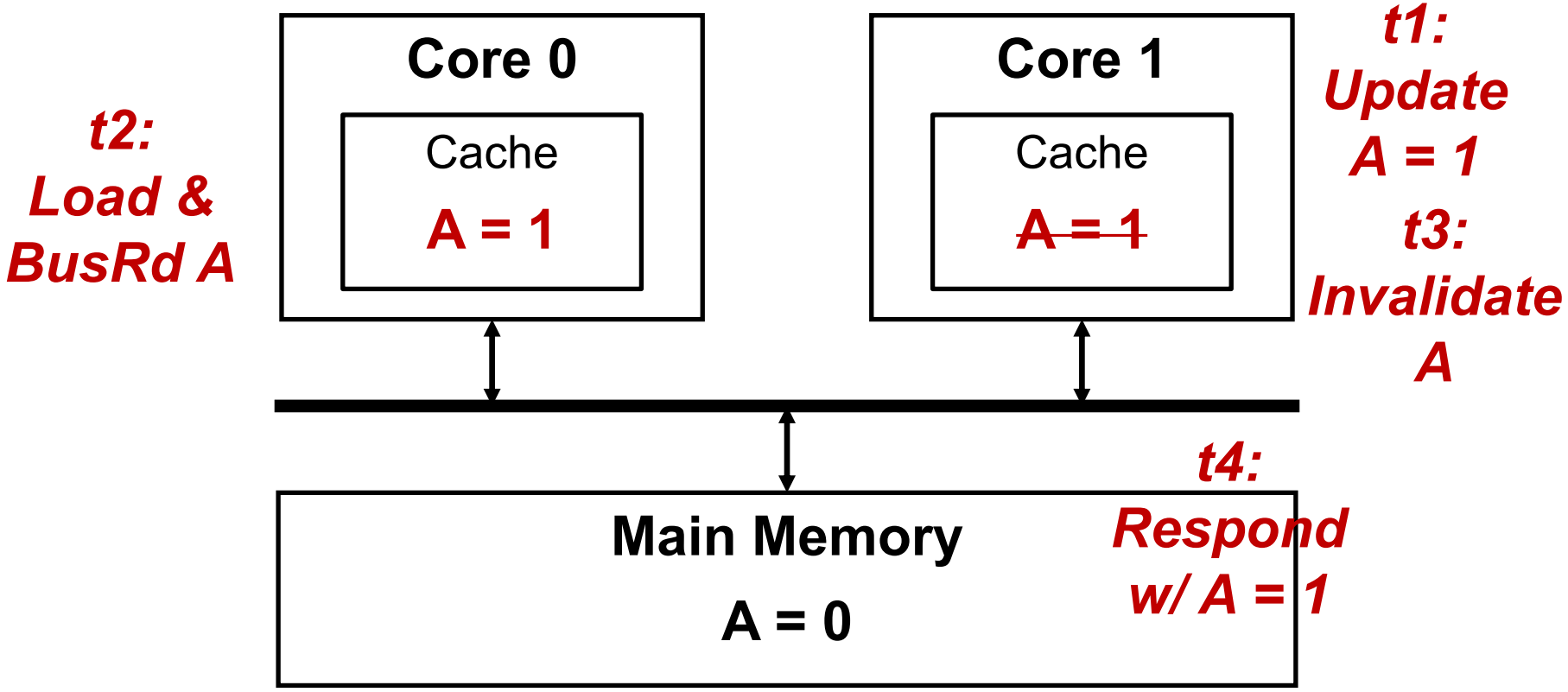
- Flush the modified value upon receiving the snooping signal (for A): flush A = 1 & let Core 0 retry



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

One-Copy Mechanism: Invalidate

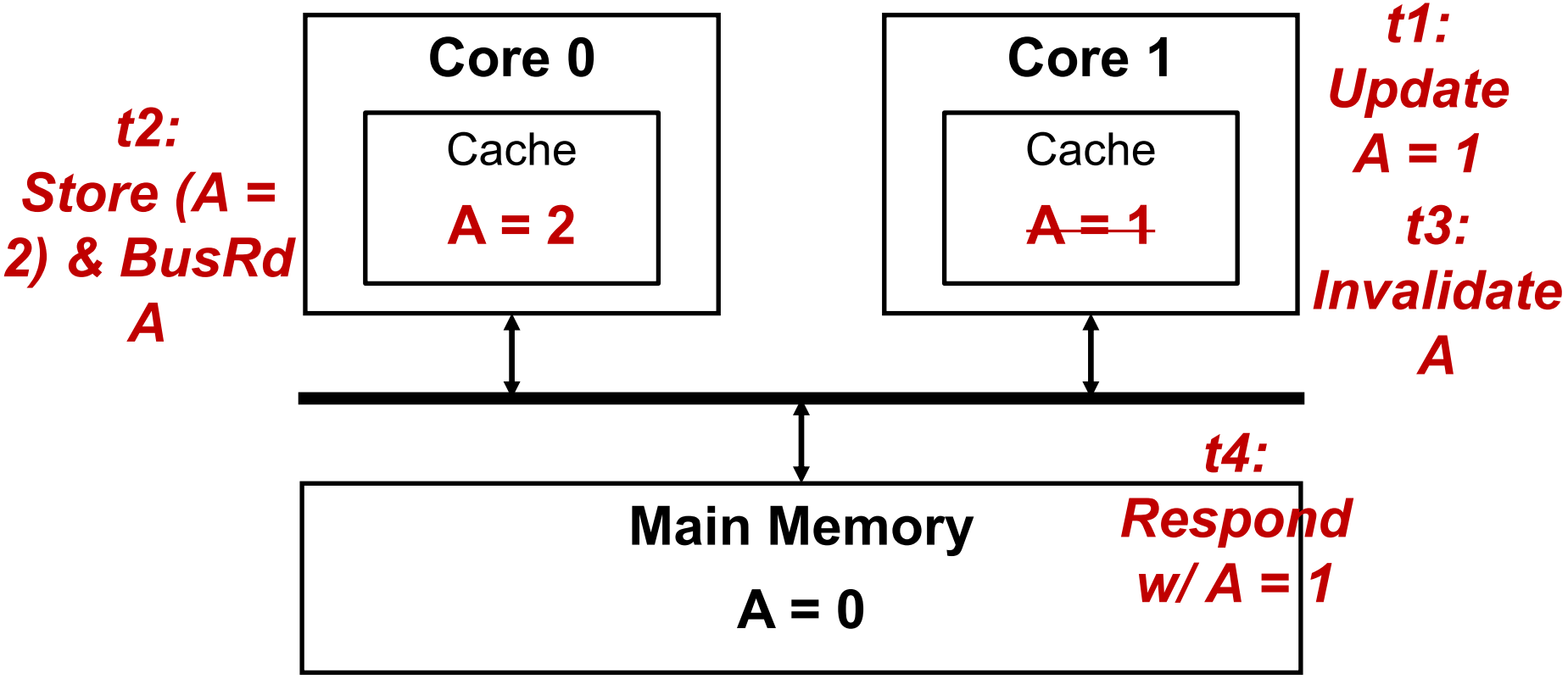
- Flush the modified value upon receiving the snooping signal (for A): flush A = 1 & let Core 0 retry



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

One-Copy Mechanism: Invalidate

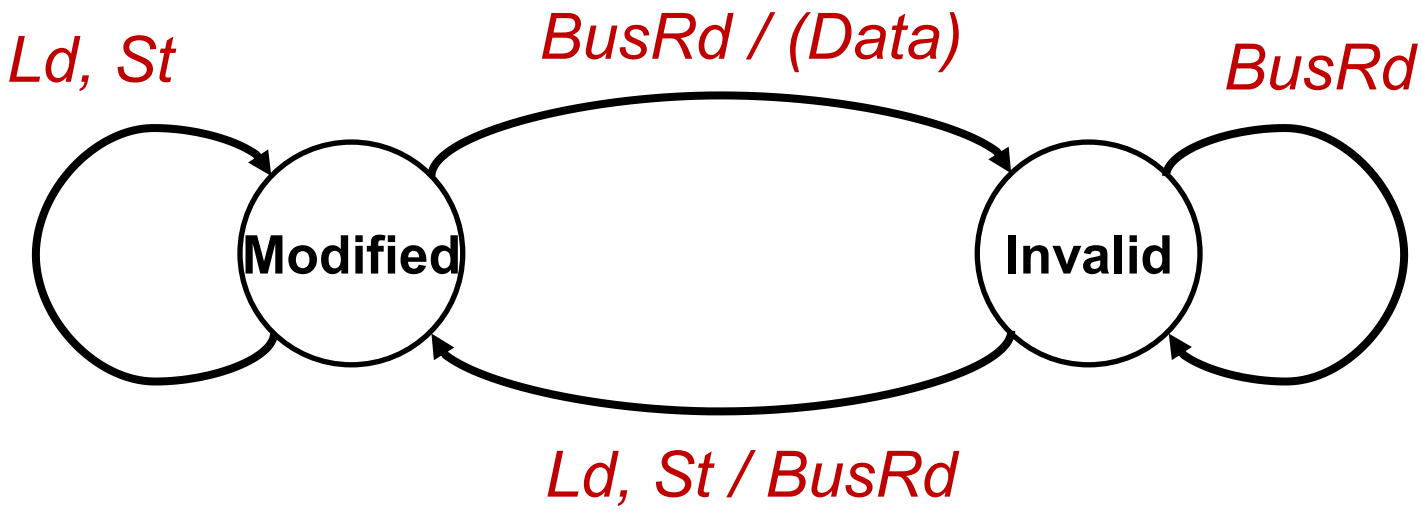
- Flush the modified value upon receiving the snooping signal (for A): flush A = 1 & let Core 0 retry



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Modified-Invalid Snooping Protocol

- Assumption: write-back & write-allocate cache
- 1-bit additional storage to track states
- One reader & One Writer
- Actions: Ld (load), St (store), BusRd (bus read signal)



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

What's the Problem?

Valid & Invalid	Modified & Invalid
Write through & Write-no-allocate	Write back & Write allocate
Multiple Readers (Multiple copies)	**Single Reader** (Single copy)
Excessive Write Overhead (Write through)	**Small Write Overhead** (Write back)

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

How about “Multiple Copies” protocol using “Write-Back” cache?

- Let’s take the benefits of both approaches: MSI protocol

- Key idea: add notion of “ownership” to Valid-Invalid
  - **Mutual exclusion** – when “owner” has only replica of a cache block, it may update it freely
  - **Sharing** – multiple readers are ok, but they may not write without gaining ownership

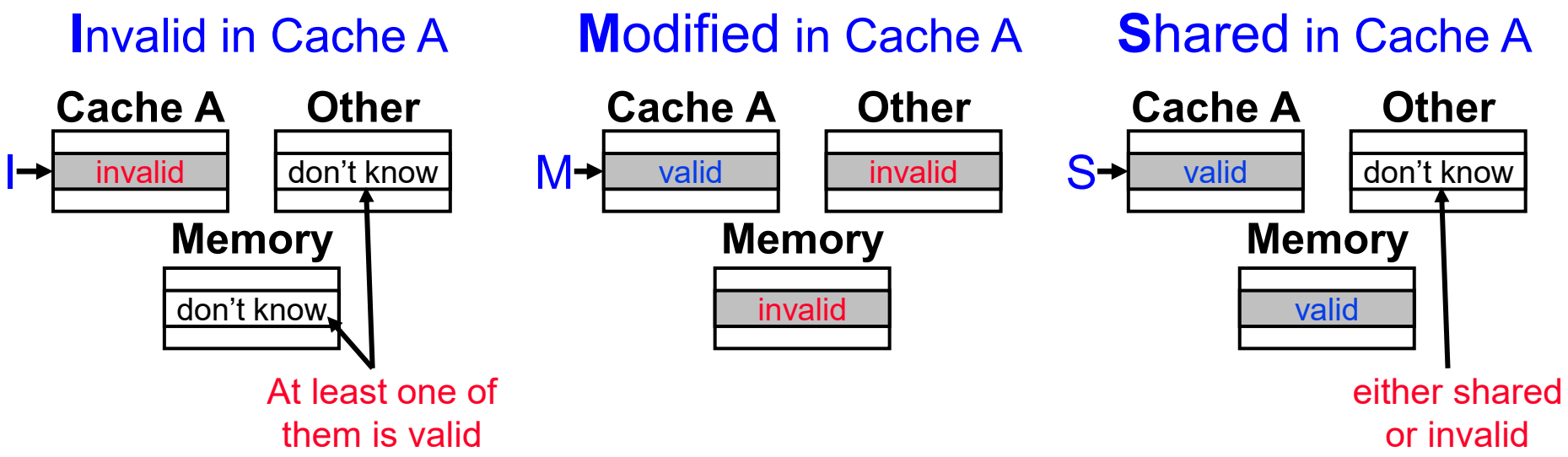
(1) Must find which cache (if any) is an owner on read misses

(2) Must eventually update memory so writes are not lost

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Coherence Protocol for Bus-Based Systems

- You can think of additional states to take the benefits of both approaches
  - **Invalid**: cache line does not have valid data
  - **Modified**: cache line has been written to since it was brought in
  - **Shared**: valid line, but other caches may have copies (presumably all identical and unchanged from memory)
    - Allows multiple readers!



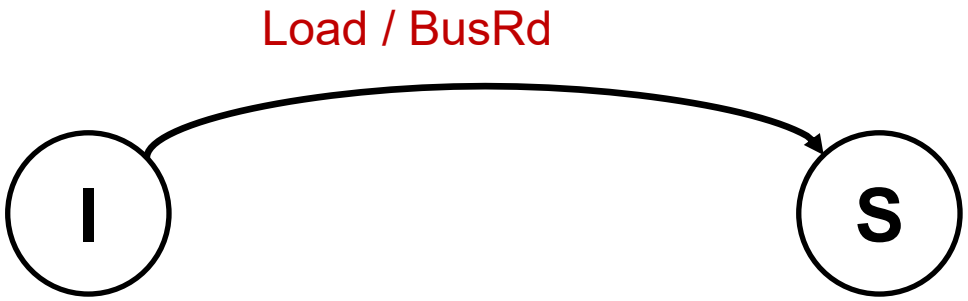
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

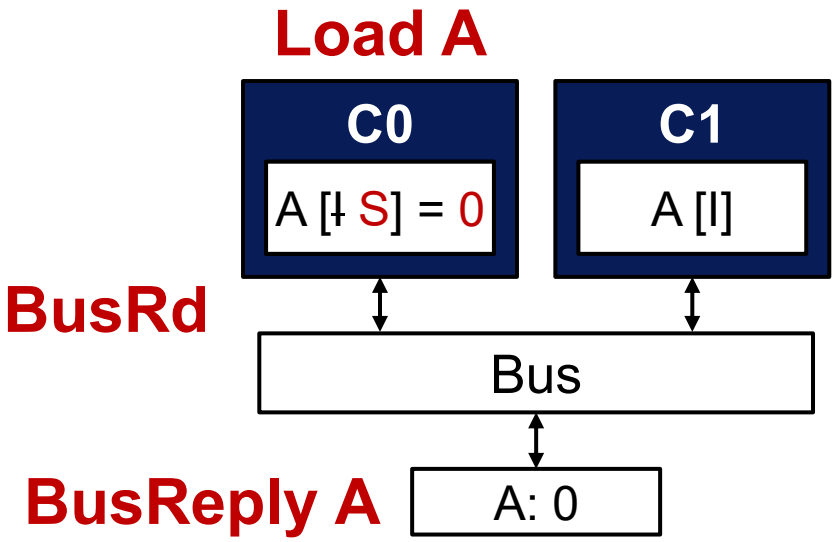
Modified-Shared-Invalid (MSI) Protocol

- Three states tracked per-block at each cache
  - **Invalid** – cache does not have a copy
  - **Shared** – cache has a read-only copy; clean
    - Clean = memory is up to date
    - Some other caches might have this copy as well.
  - **Modified** – cache has a writable, "dirty" copy
    - Dirty = memory is out of date
    - No other cache has this copy
- Three actions **from processor**
  - **Load, Store, Evict**
- Five messages **from bus**
  - **BusRd, BusRdX, BusInv, BusWB, BusReply**
  - Could combine some of these

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

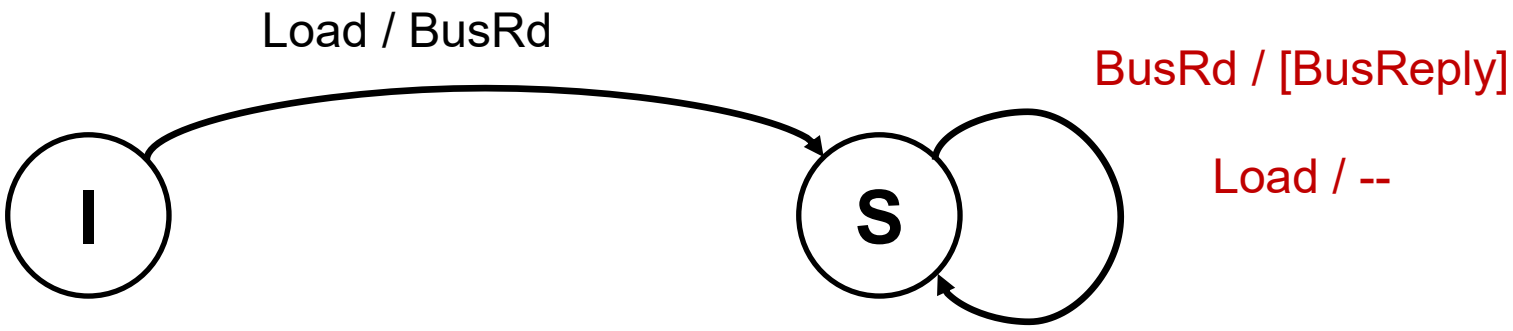
Modified-Shared-Invalid (MSI) Protocol

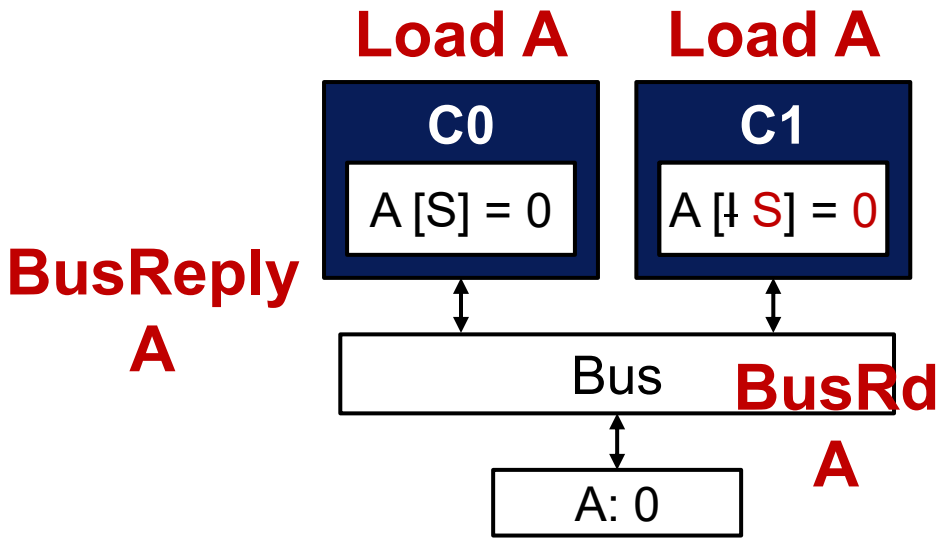




© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

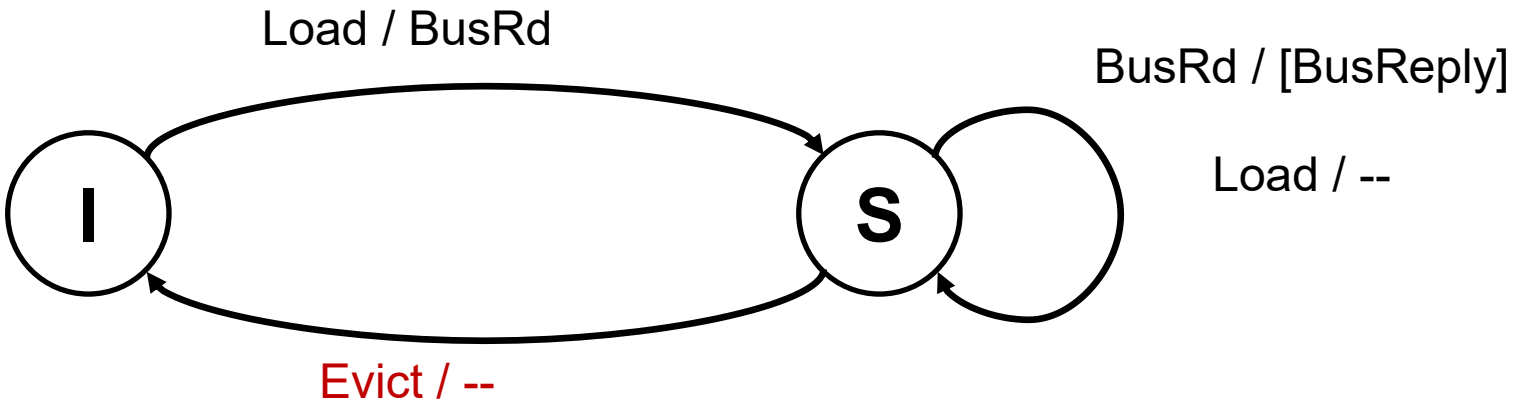
Modified-Shared-Invalid (MSI) Protocol

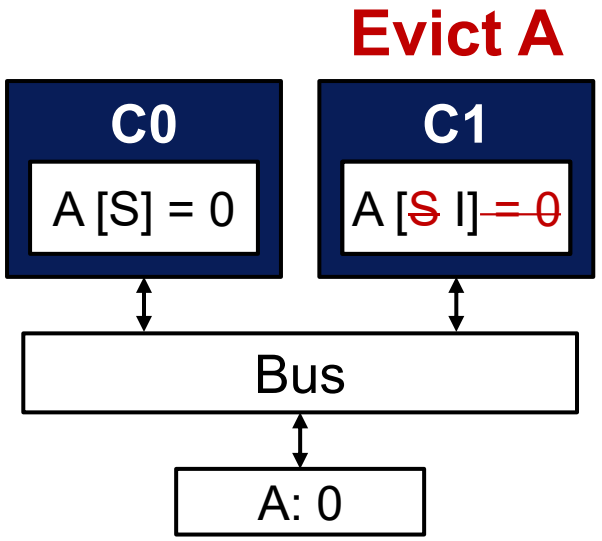




© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

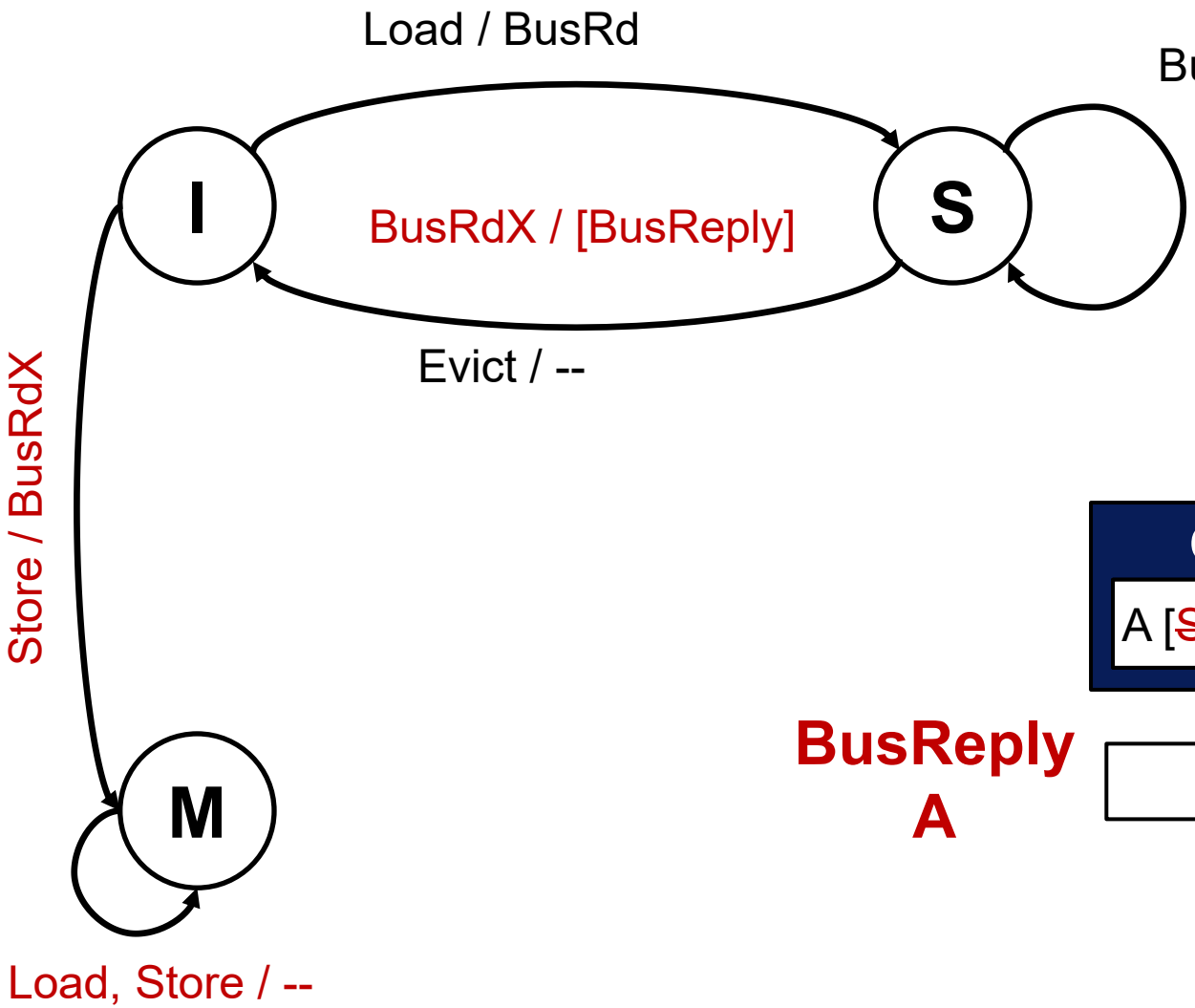
Modified-Shared-Invalid (MSI) Protocol

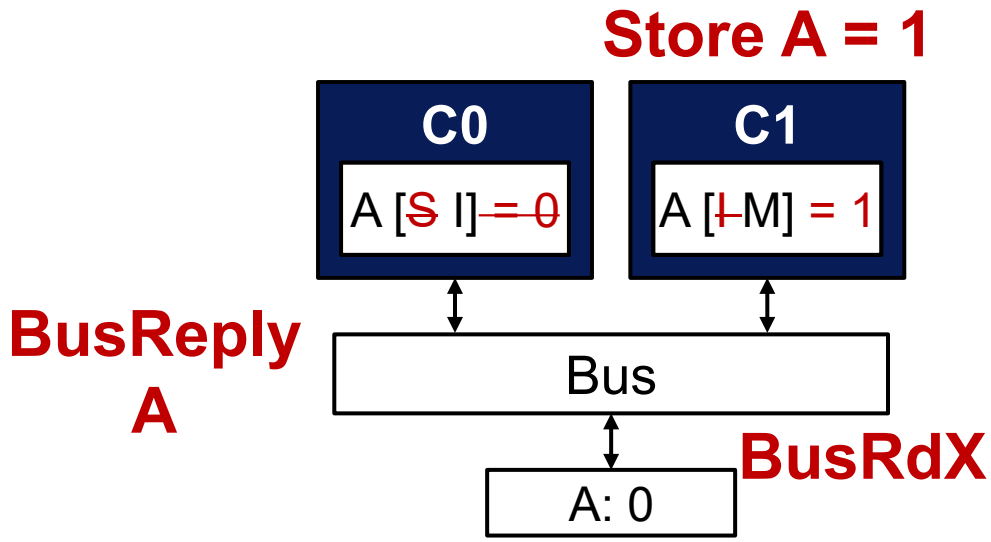




© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

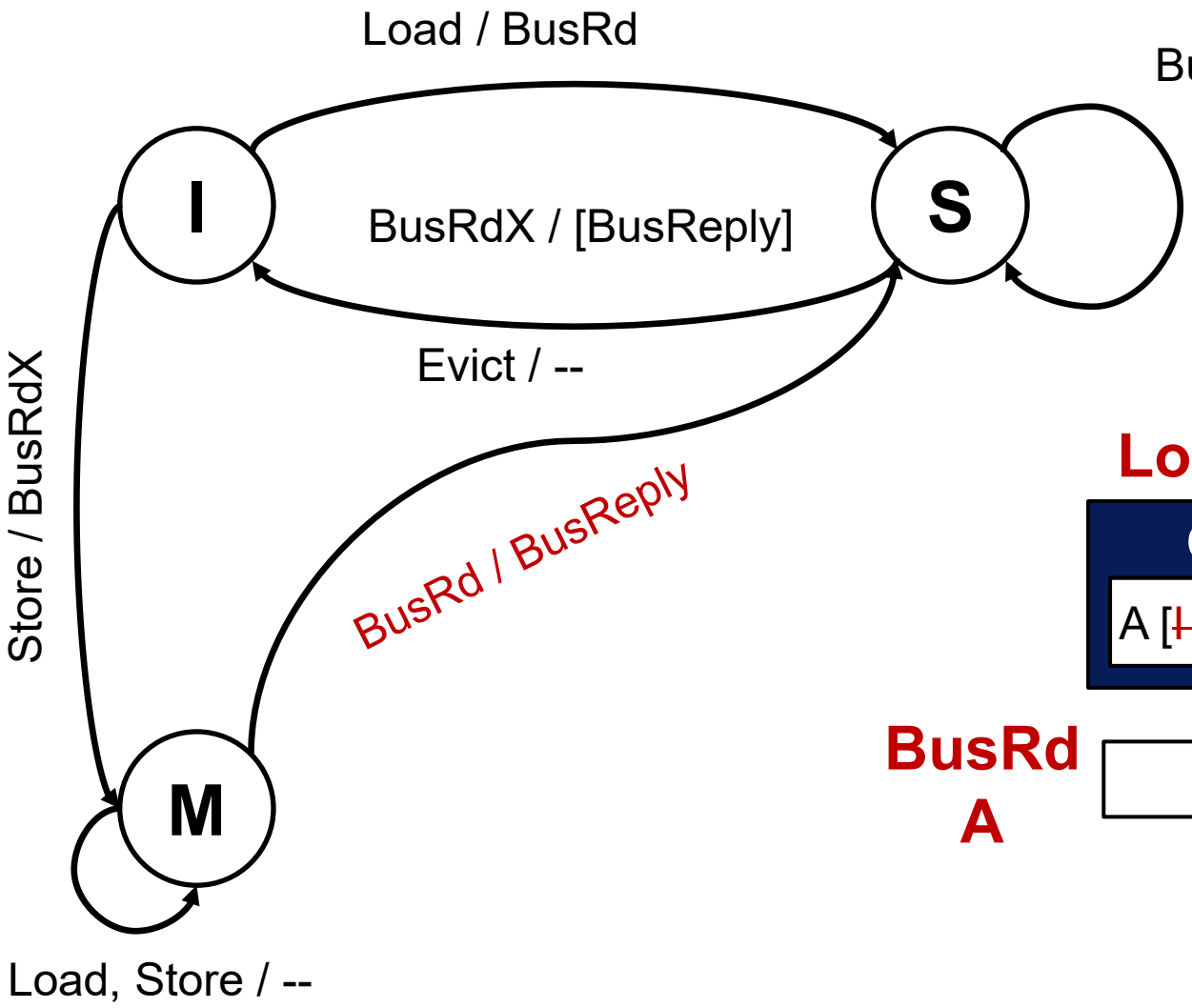
Modified-Shared-Invalid (MSI) Protocol

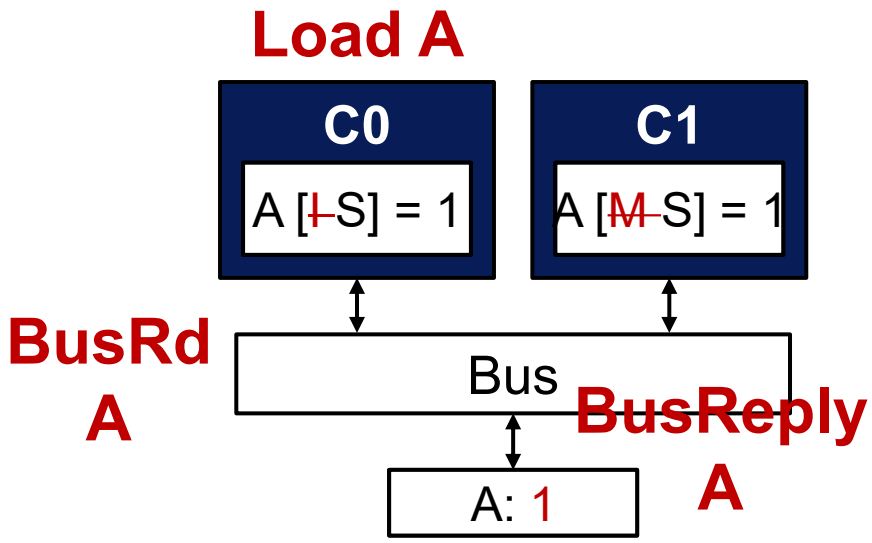




© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

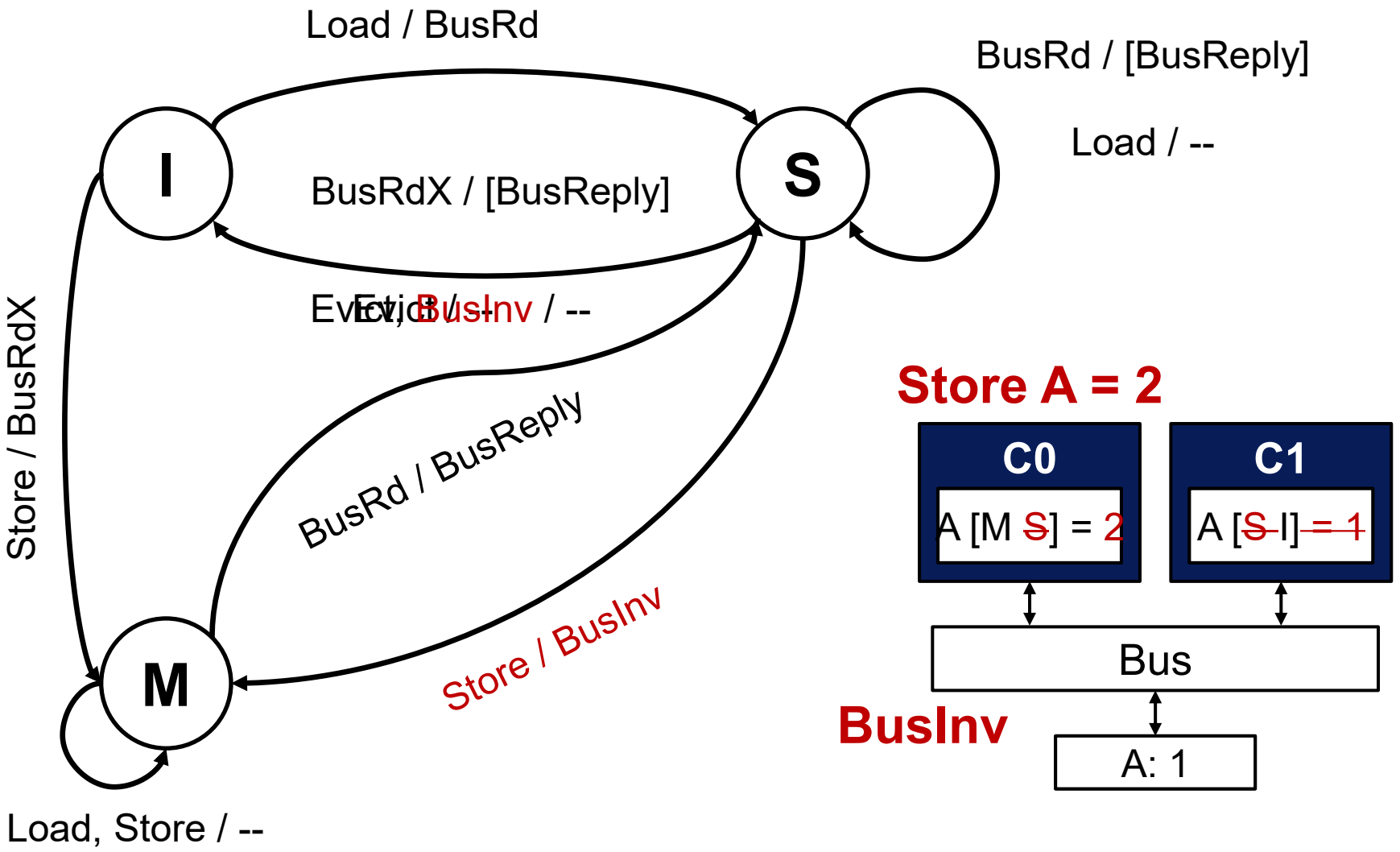
Modified-Shared-Invalid (MSI) Protocol





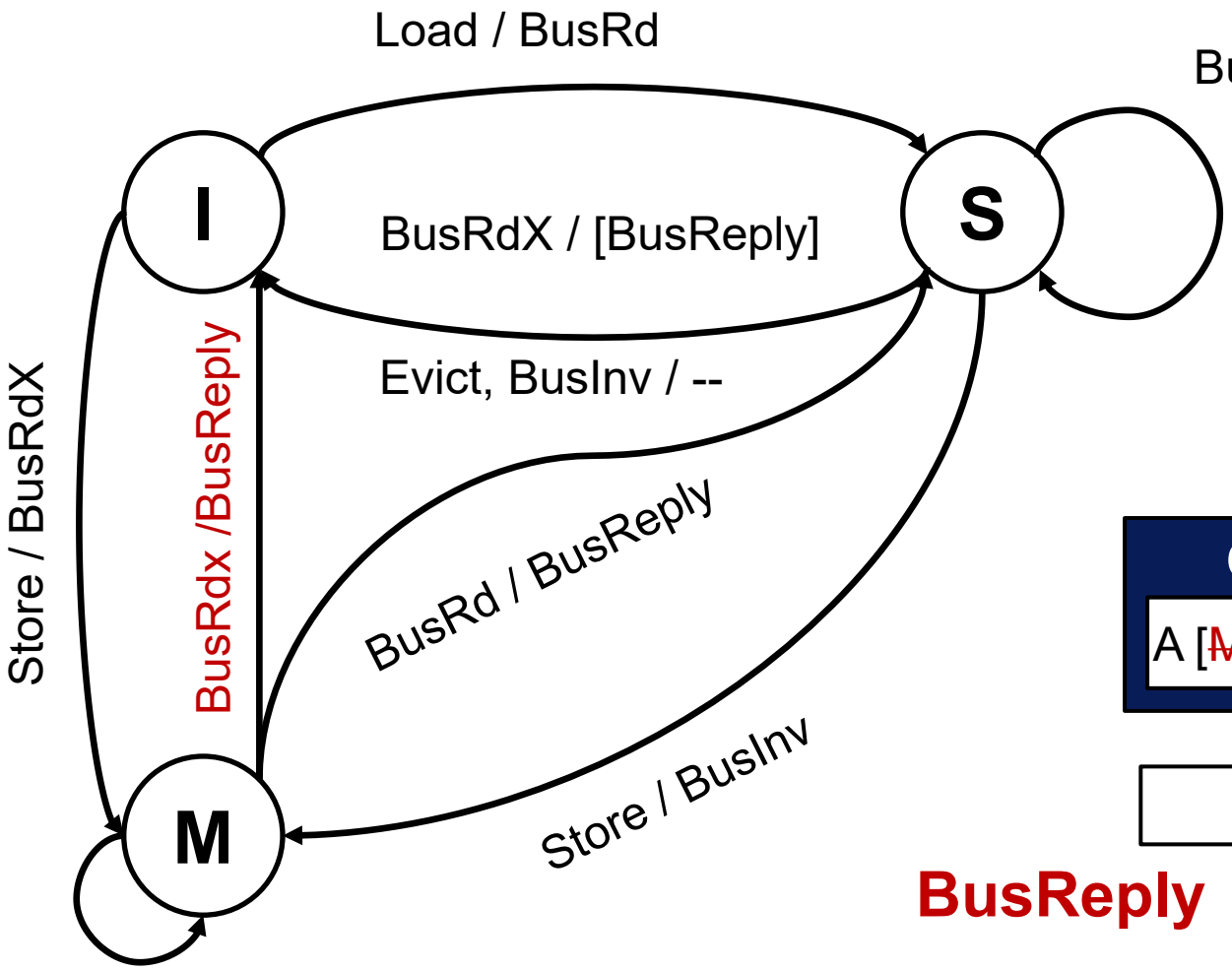
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

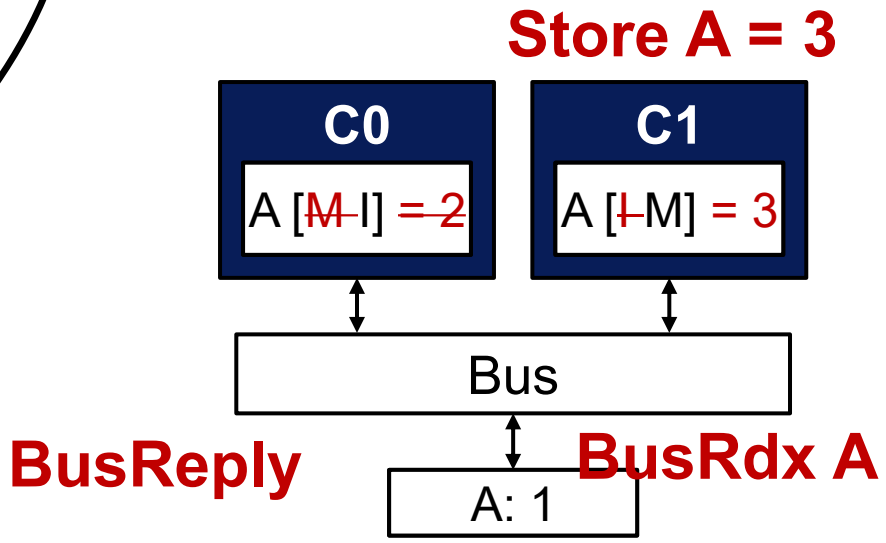
Modified-Shared-Invalid (MSI) Protocol



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

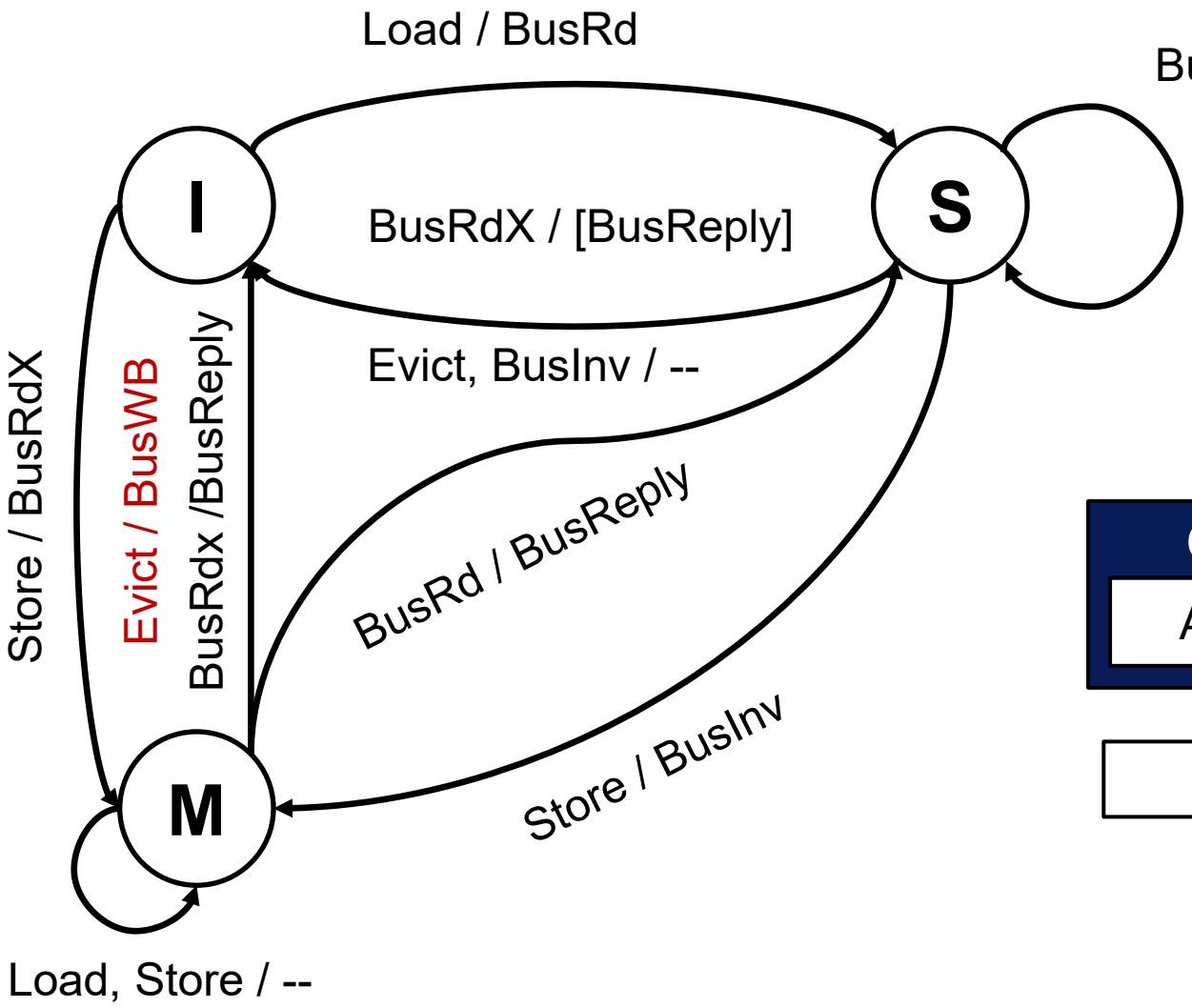
Modified-Shared-Invalid (MSI) Protocol

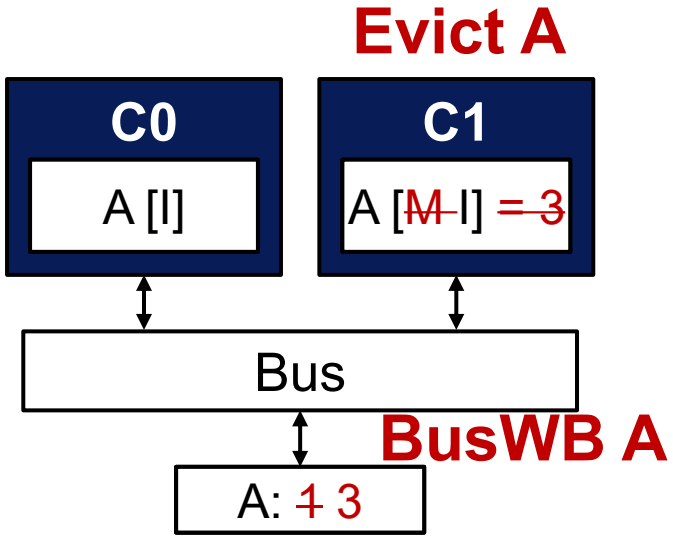




© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

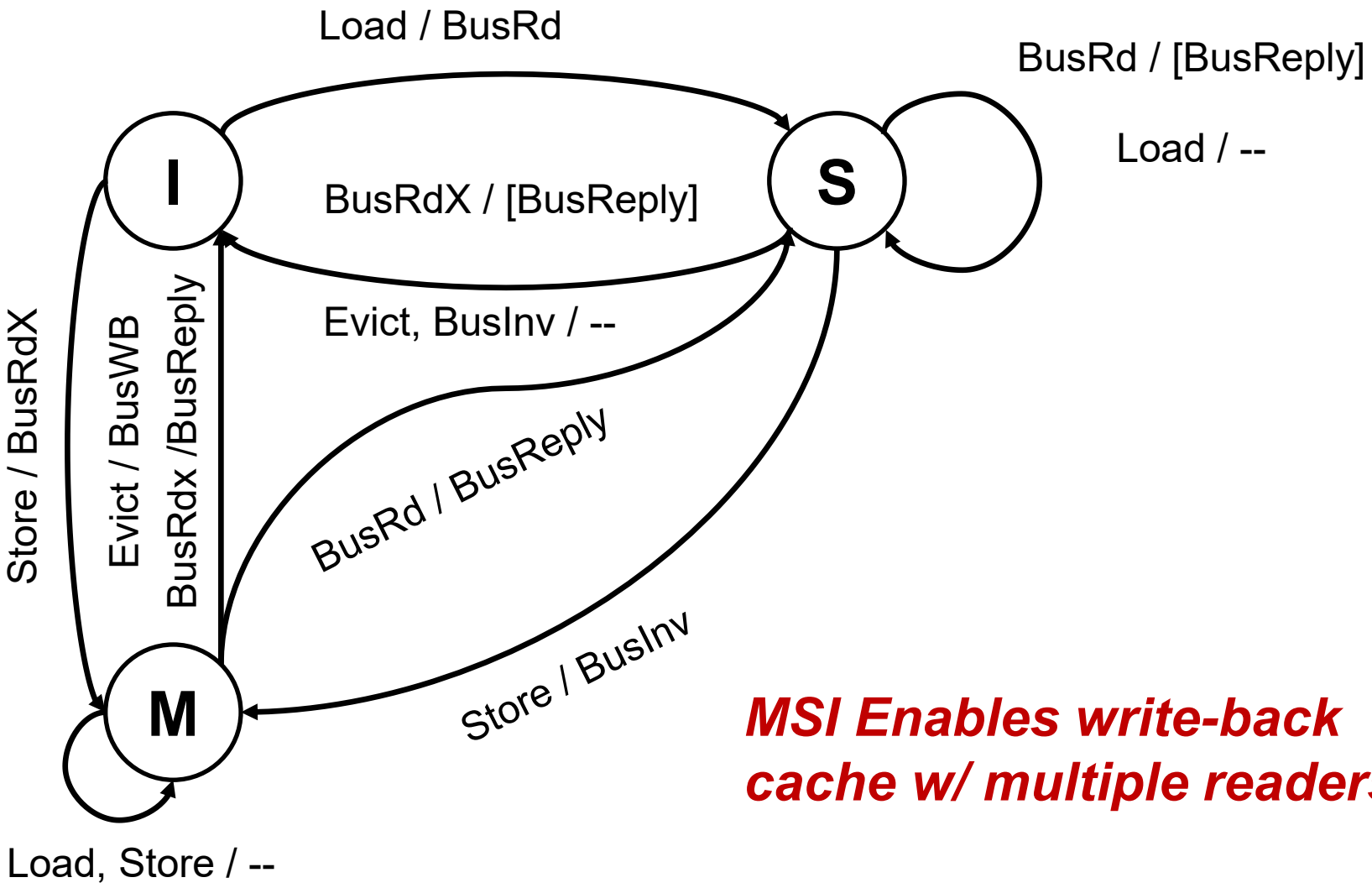
Modified-Shared-Invalid (MSI) Protocol





© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

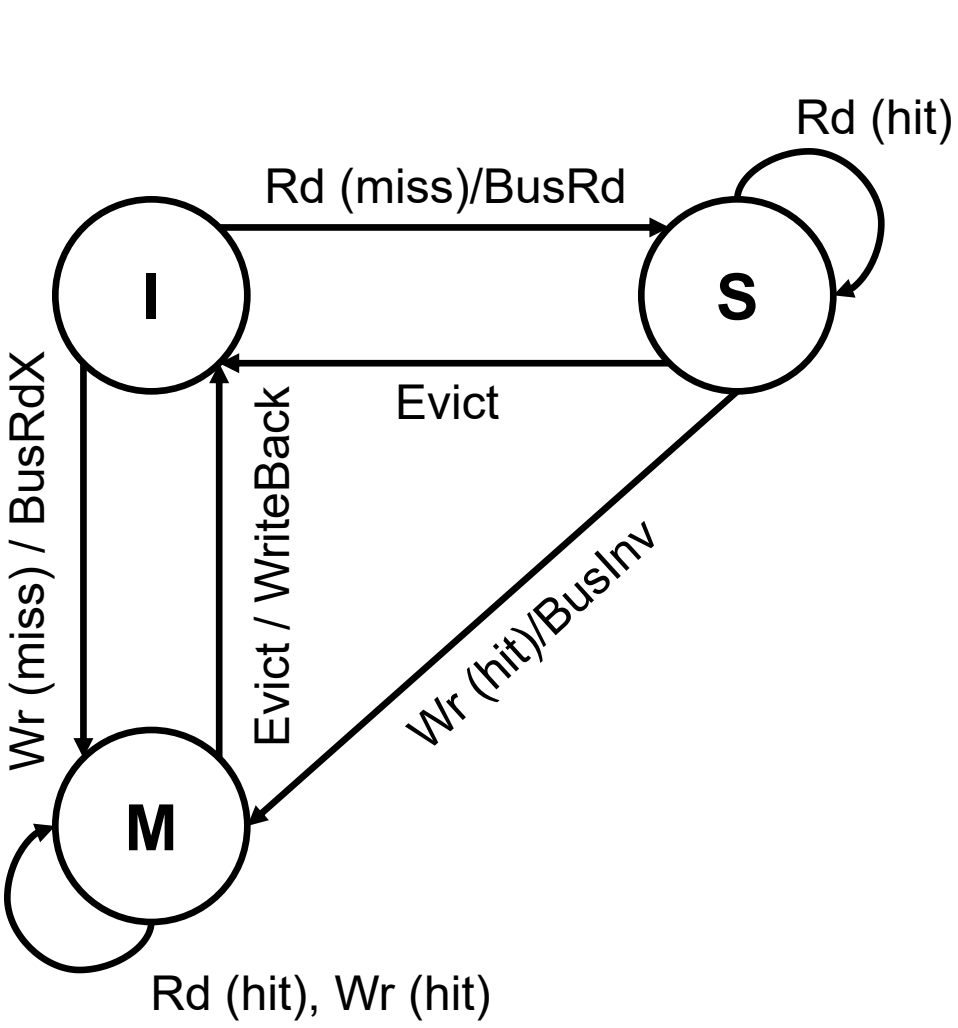
Modified-Shared-Invalid (MSI) Protocol Summary

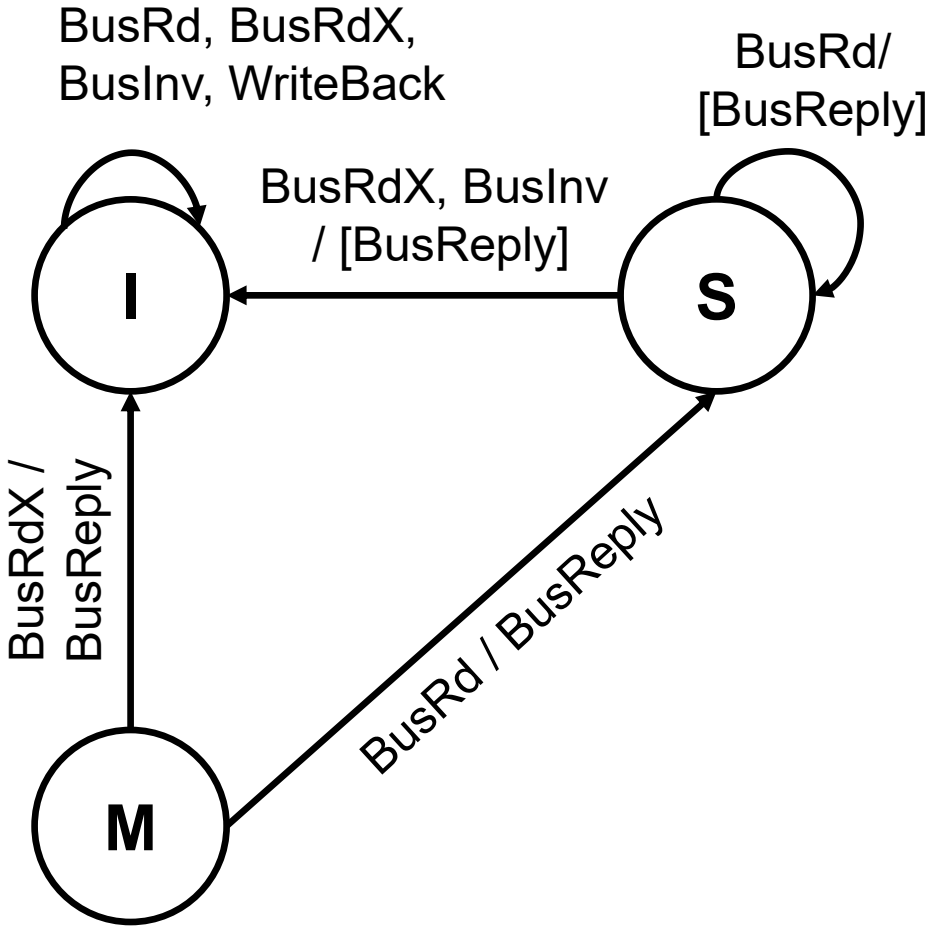


MSI Enables write-back cache w/ multiple readers!!!

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

MSI State Transition





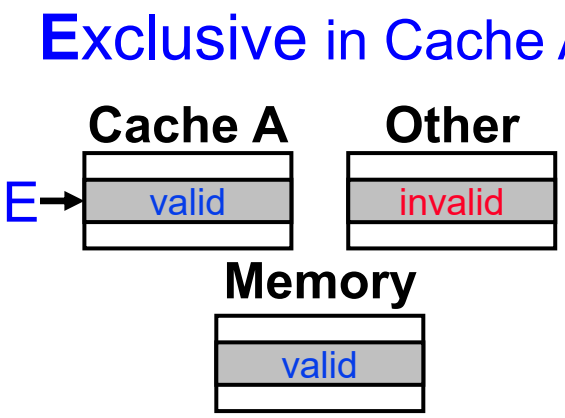
State Transitions Using CPU Signals

State Transitions Using BUS Signals

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

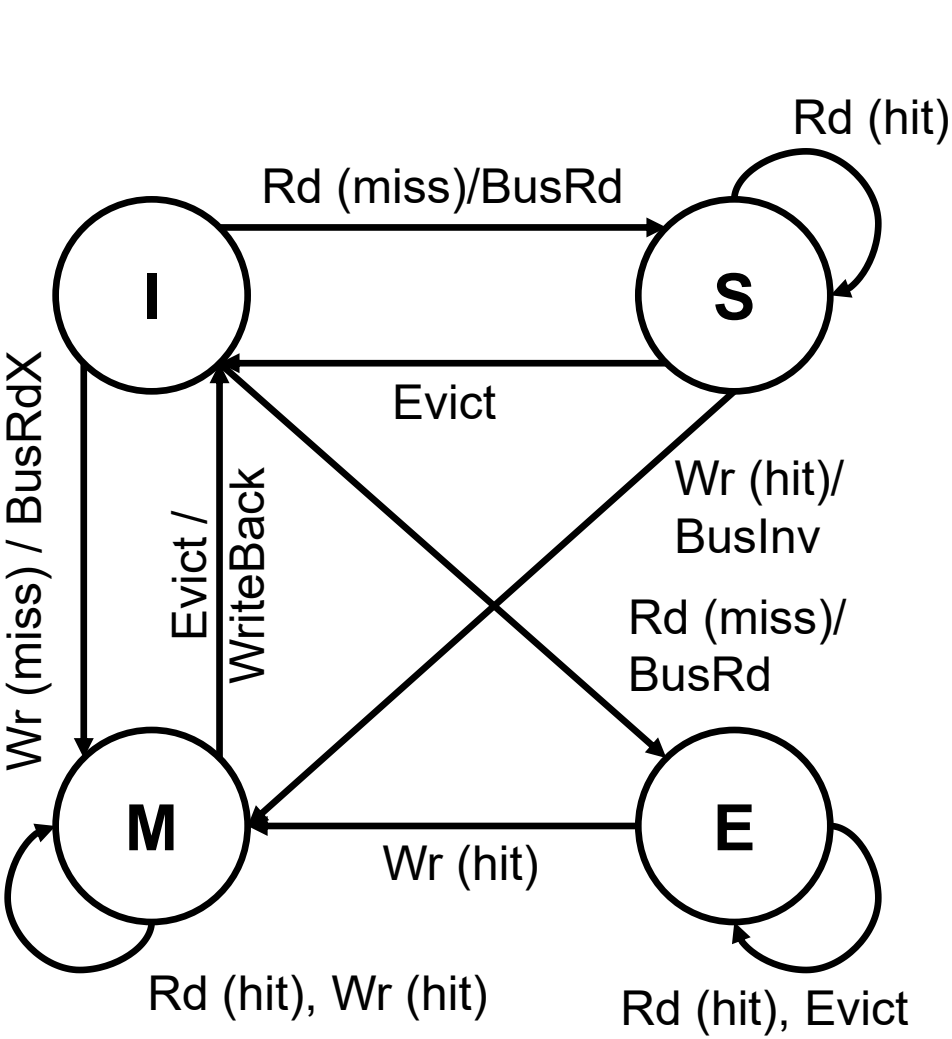
Better Protocol: MESI!

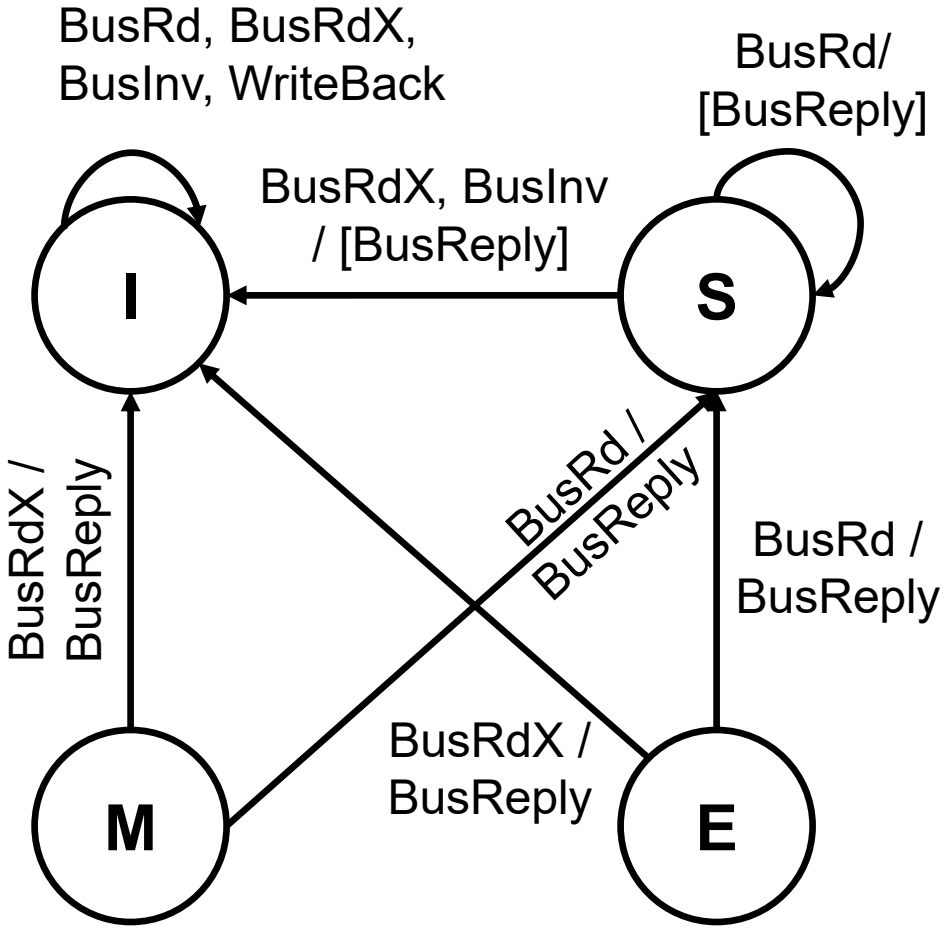
- MSI suffers from frequent read-update sequences
  - Leads to two bus transactions, **even for private blocks** (=one sharer)
    - BusRd : gets the block in S
    - BusRdX (or BusUpgr) : convert S to M
      - Must broadcast useless invalidation messages
  - Uniprocessors don't have this problem
- Solution: add an "**Exclusive**" state
  - Exclusive – only one copy; writable; clean
    - Can detect exclusivity when memory provides reply to a read



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

M“E”SI State Transition





State Transitions Using CPU Signals

State Transitions Using BUS Signals

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Is MESI Protocol the Best?

- There are several optimization opportunities
  - Should a downgrade from M go to S or I?
    - **M → S (current impl):** If the data is likely to be reused
    - **M → I (Maybe …?):** If the data is likely be written by another core
  - Cache-to-cache data transfers
    - When reading a data from the shared state;
      - Read the data from the cache: Faster!
      - Read the data from the memory: Slower, but simpler impl …
  - Writeback on M → S (S ➔ Shared with memory)
    - S state implies that the data is also at the main memory ➔ Should we enforce this???
    - Utilize a new Owner (O) state → Only cache owns the latest data and the cache with O state provides the latest copy (others are in S state)

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

MOESI

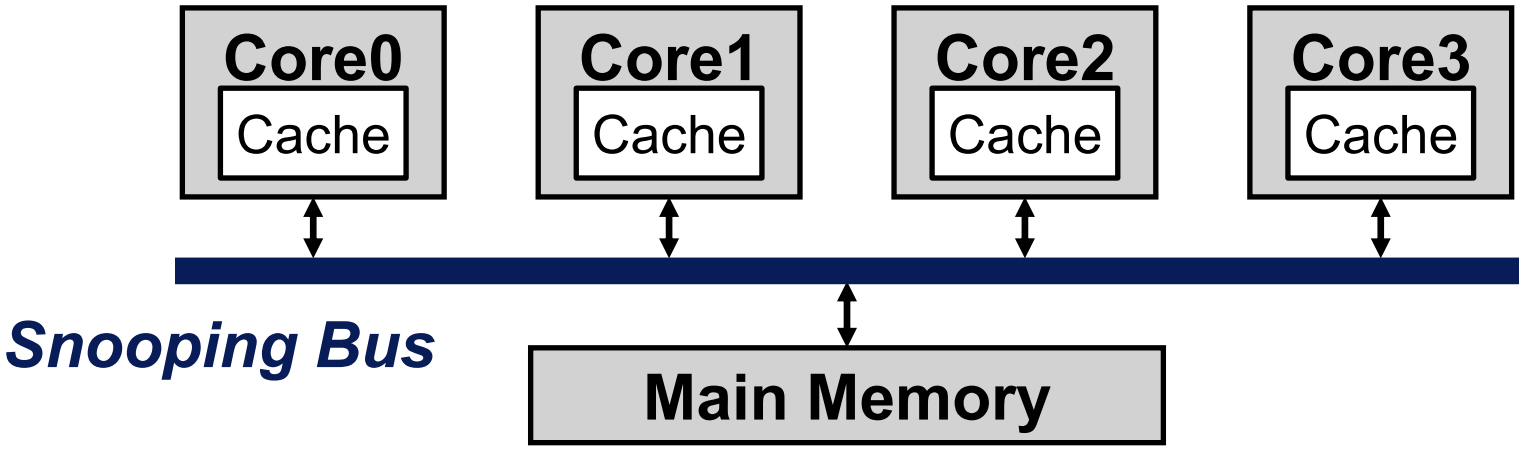
- Key motivation behind MOESI → "I do not want to write back whenever I become M → S"
  - Problem: Shared state applies for only clean data
- Key Idea: "Let's also have shared states for dirty data"
  - Now … we need a dedicate a cache to write-back the data on eviction! ➔ Owner of the dirty data
  - Imagine the FSM by yourself!
- Consider transition from MSI → MESI → MOESI
  - Is it just getting better??

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

How to implement CC? UMA structure + Snooping Bus

- **Using a UMA structure (cc-UMA):** there is a single bus to serialize memory requests

- All the cores listen to the bus requests from other cores! ➔ Snooping Bus!



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Straightforward, but “Not Scalable!”

- **Every bus snoop requires a cache lookup**
  - Needs a dual ported cache or at least an extra tag lookup port
  - In an inclusive L1 and L2 arrangement, snoop only goes to L2 and does not contend with processor for L1 bandwidth

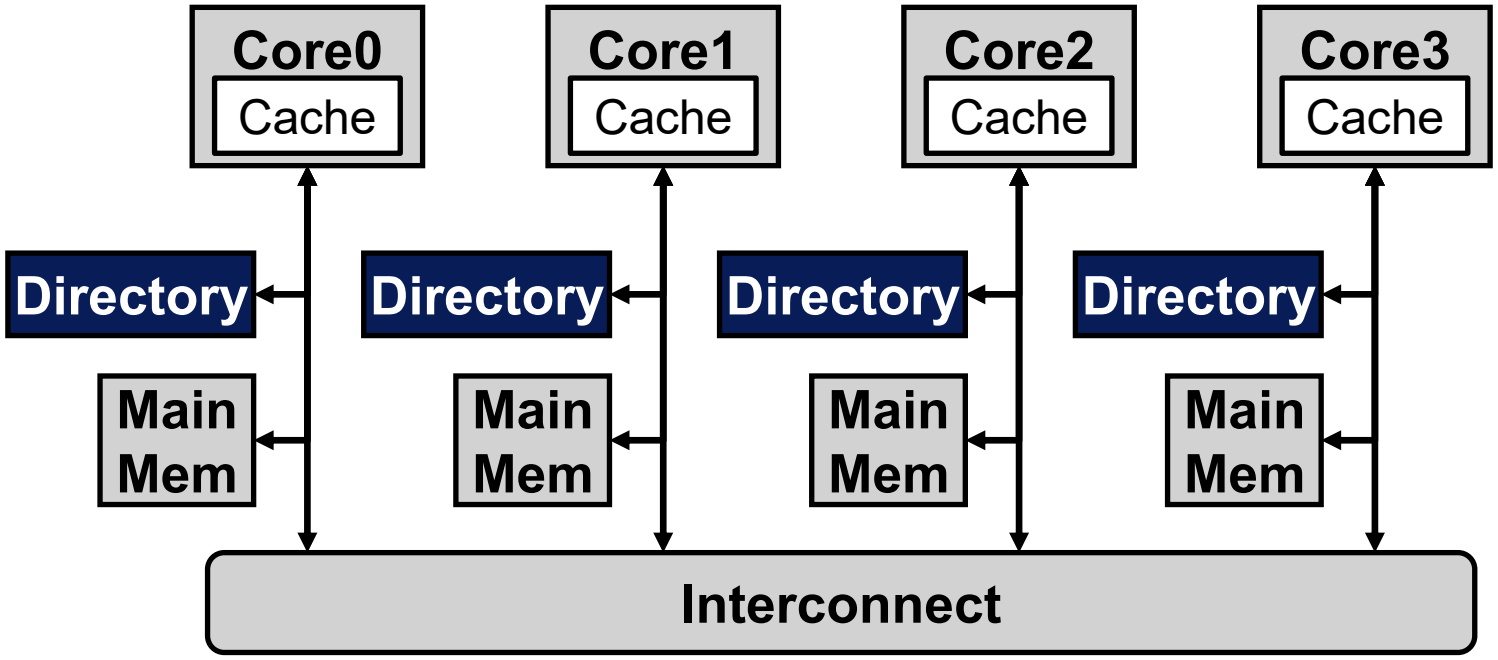
    (True benefits of “inclusive” cache!)
- **Bus is not very scalable**
  - Physical limits (such as number of drops and physical extent of the bus) force bigger buses to clock slower
  - Bus bandwidth is divided when you add more processors
- **Implementing Snoopy protocols can get complicated**
  - MESI state transitions are not really atomic
  - CPU and bus transactions are not atomic
  - …

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

NUMA structure + Directory

- **Use a NUMA structure (cc-NUMA):** the serialization point is distributed across the nodes

- Directory coordinates invalidation, updates, cache sharing …

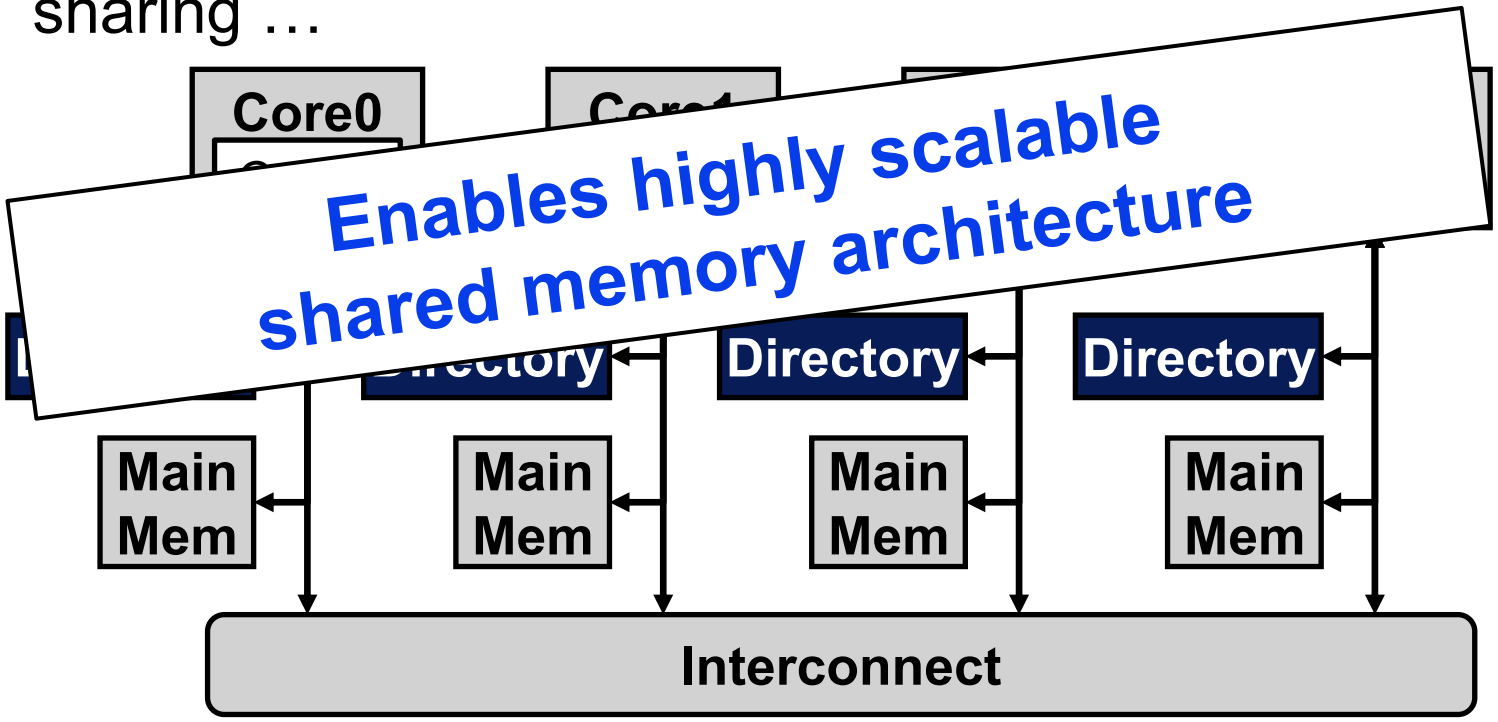


© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

NUMA structure + Directory

- **Use a NUMA structure (cc-NUMA):** the serialization point is distributed across the nodes

- Directory coordinates invalidation, updates, cache sharing …



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Key Benefits of Using Directory

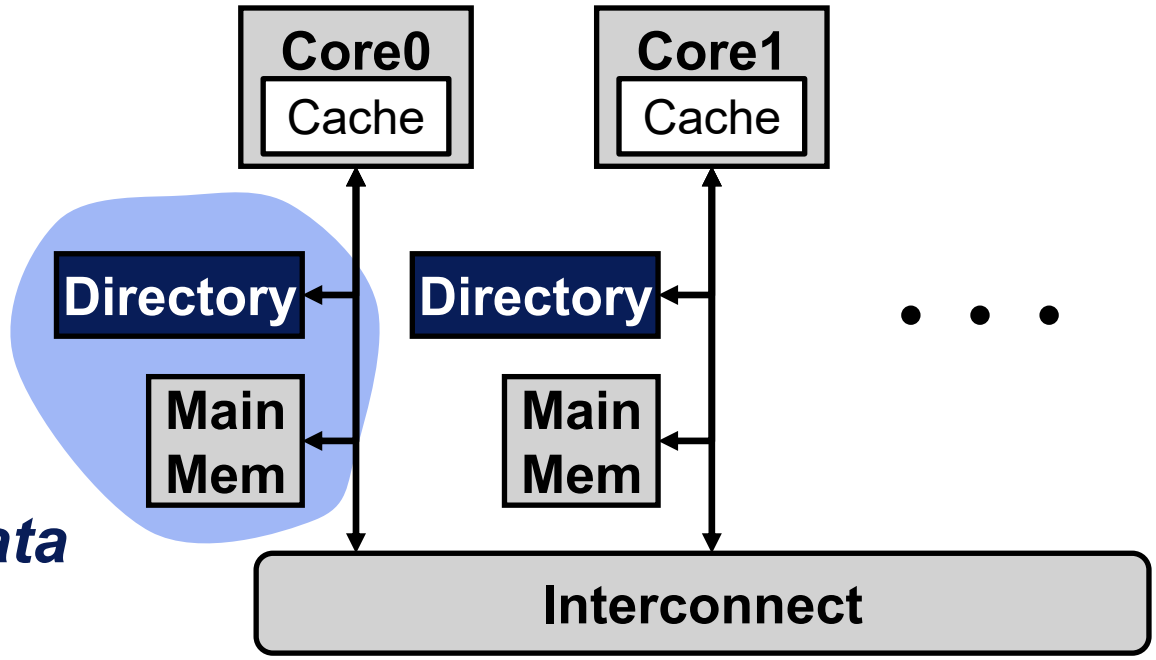
- **Benefit #1:** We can utilize the highly scalable NUMA architecture (the bus bandwidth is not shared)
  - Enables scalable interconnection network

- **Benefit #2:** We can reduce the broadcasting overhead (of snooping bus), the directory actively notifies the processors (caches) that only matters!

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

NUMA structure + Directory

- **Directory:** Extend memory to track caching information
  - Keeps information of only the local (home) main memory (not remote ones)
  - Each processor sends a coherence-related messages to the directory, and the directory acts as a serialization point!

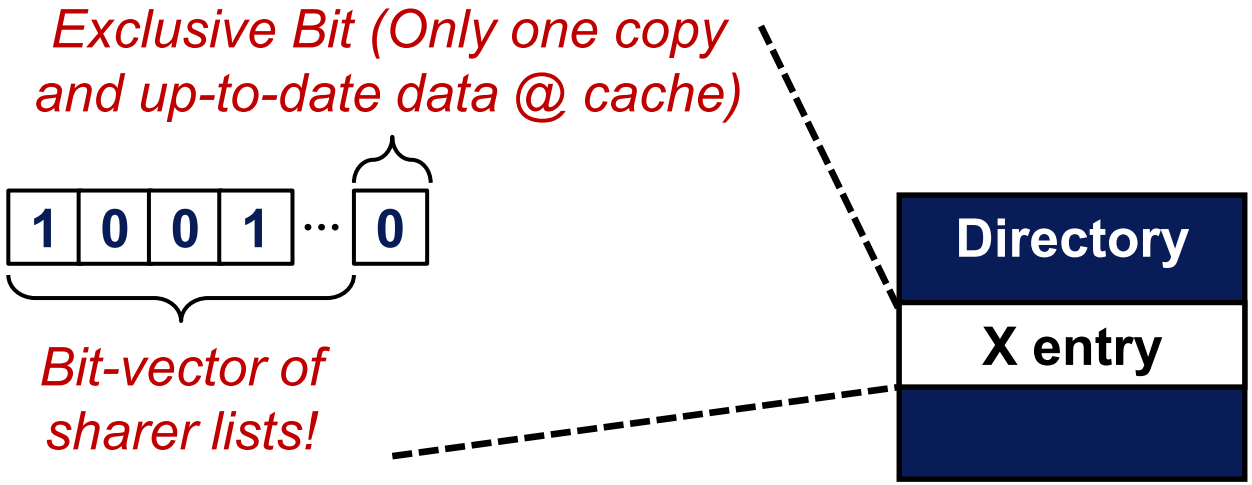


Tracks local data

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

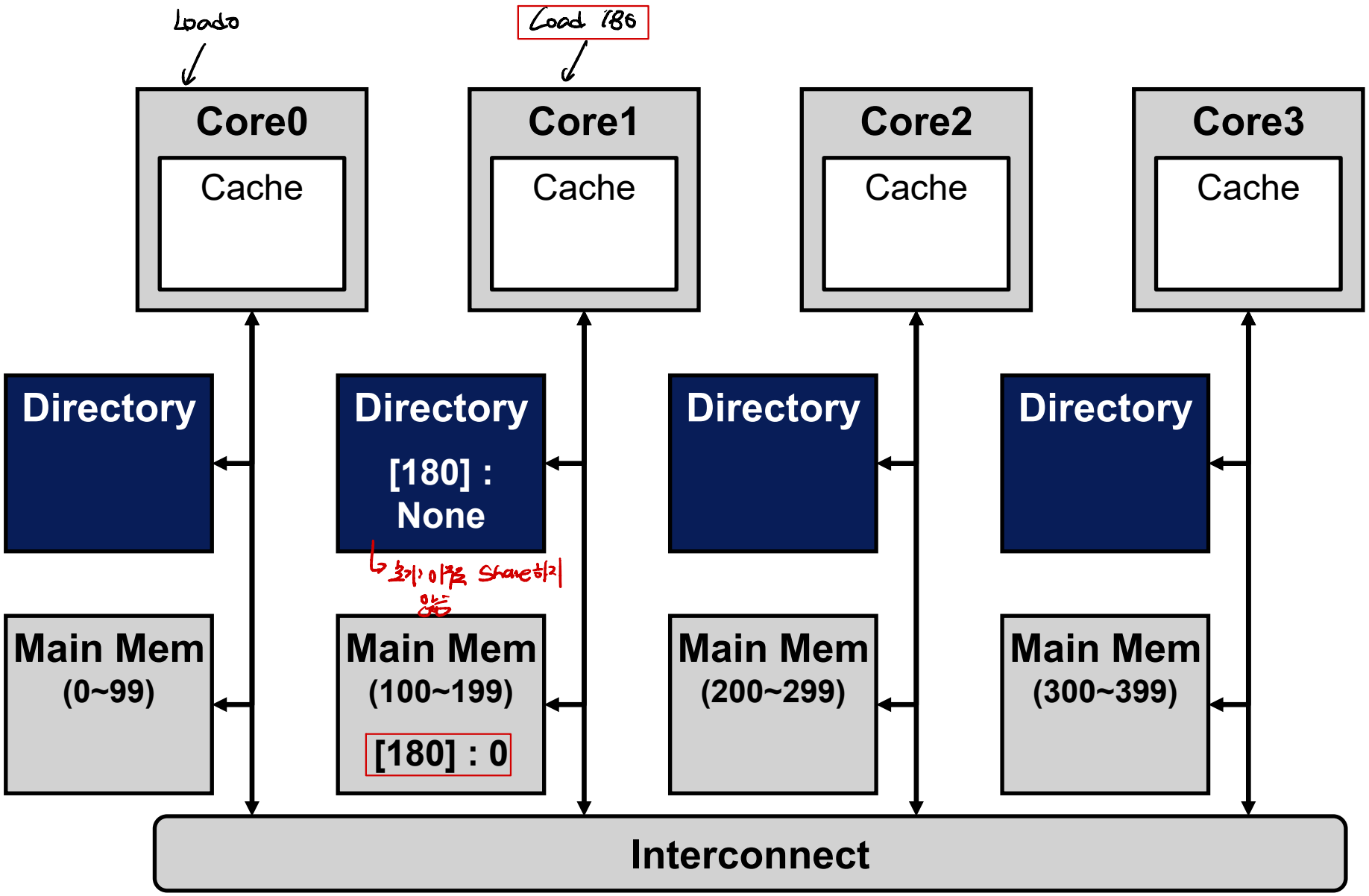
Directory Implementation

- What are the data stored in each directory?
  - Shared-vector: the list of sharer
  - Exclusive bit: there is only a single core that has the data and grant the permission to freely modify the data



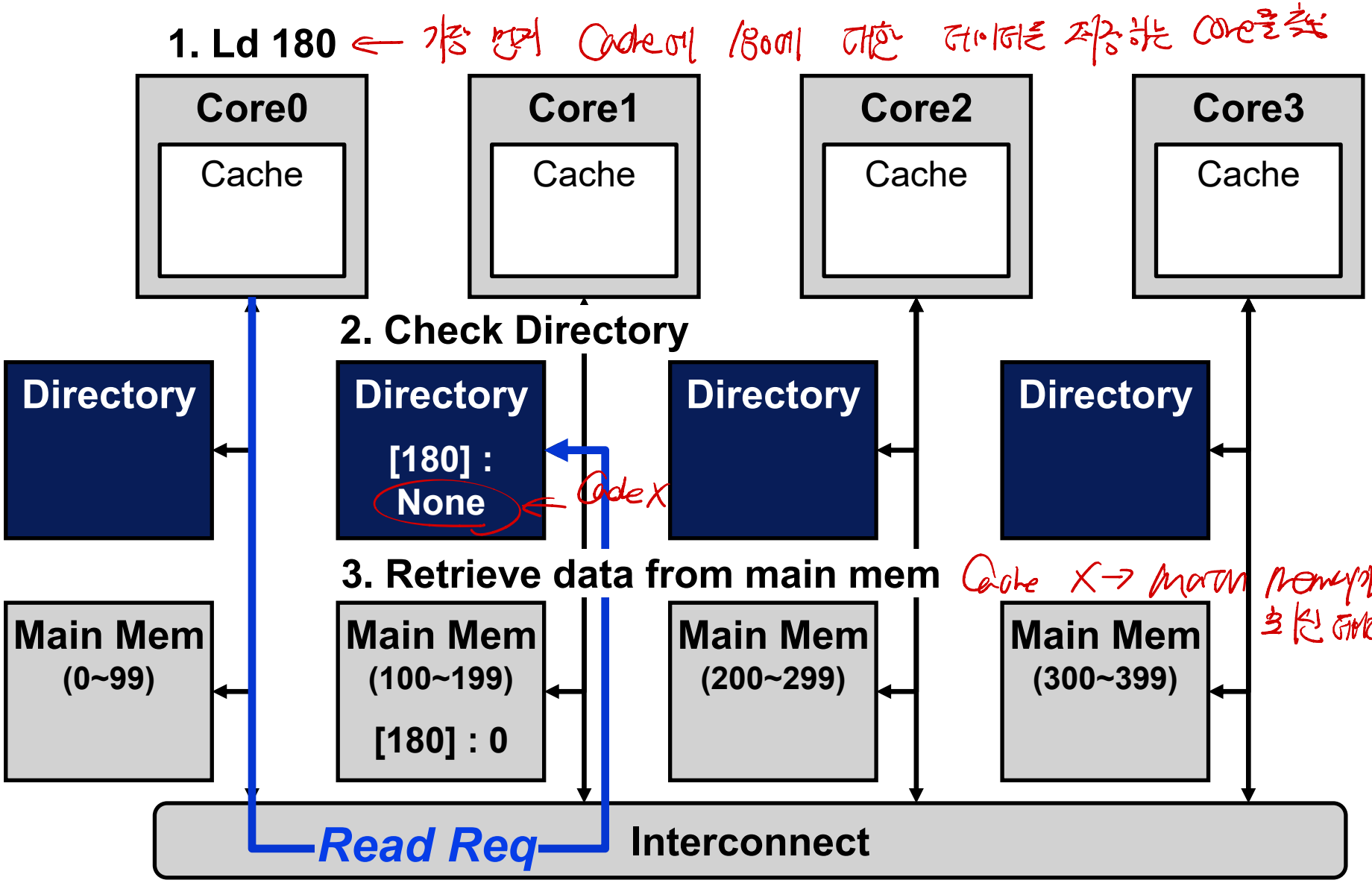
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



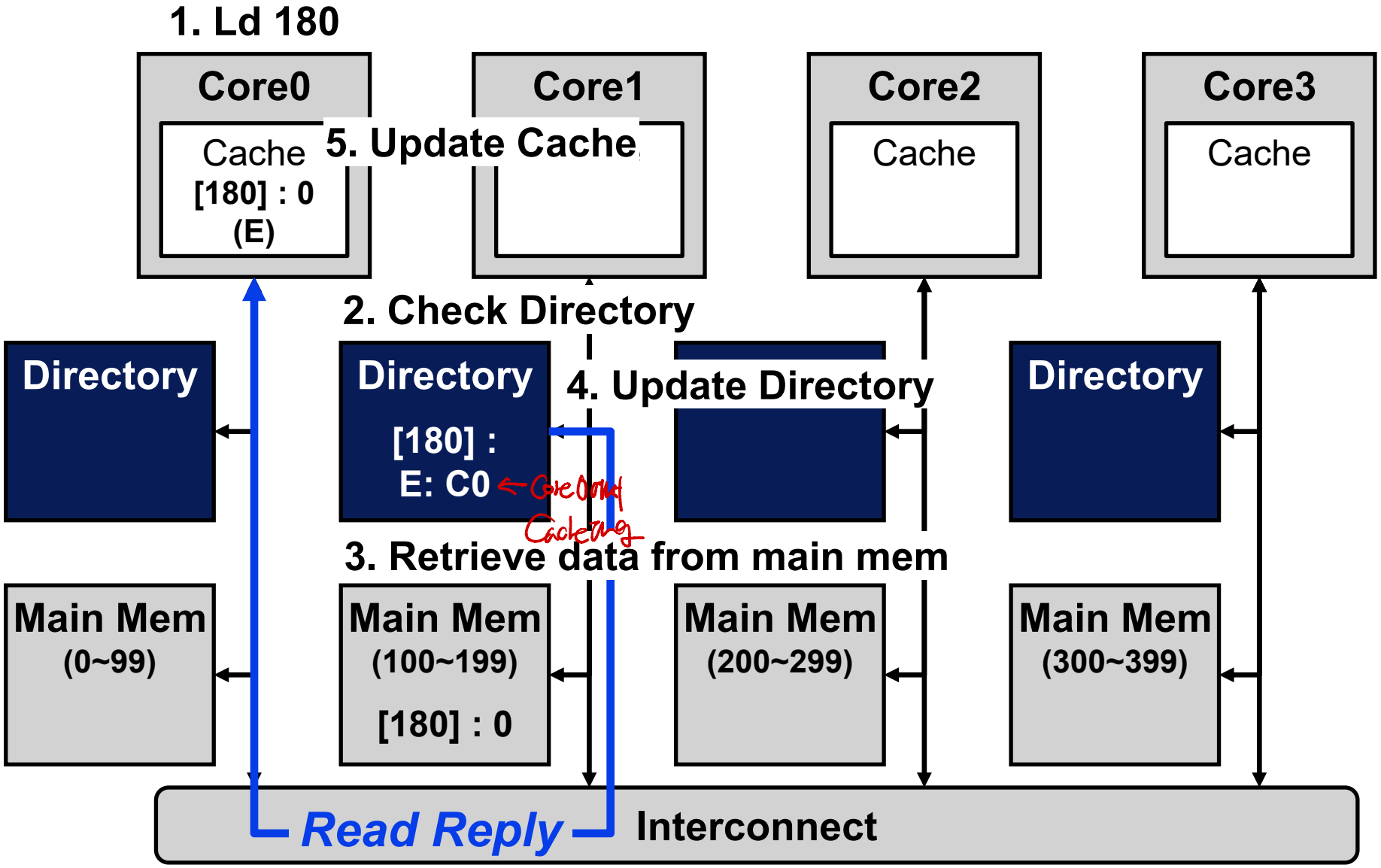
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



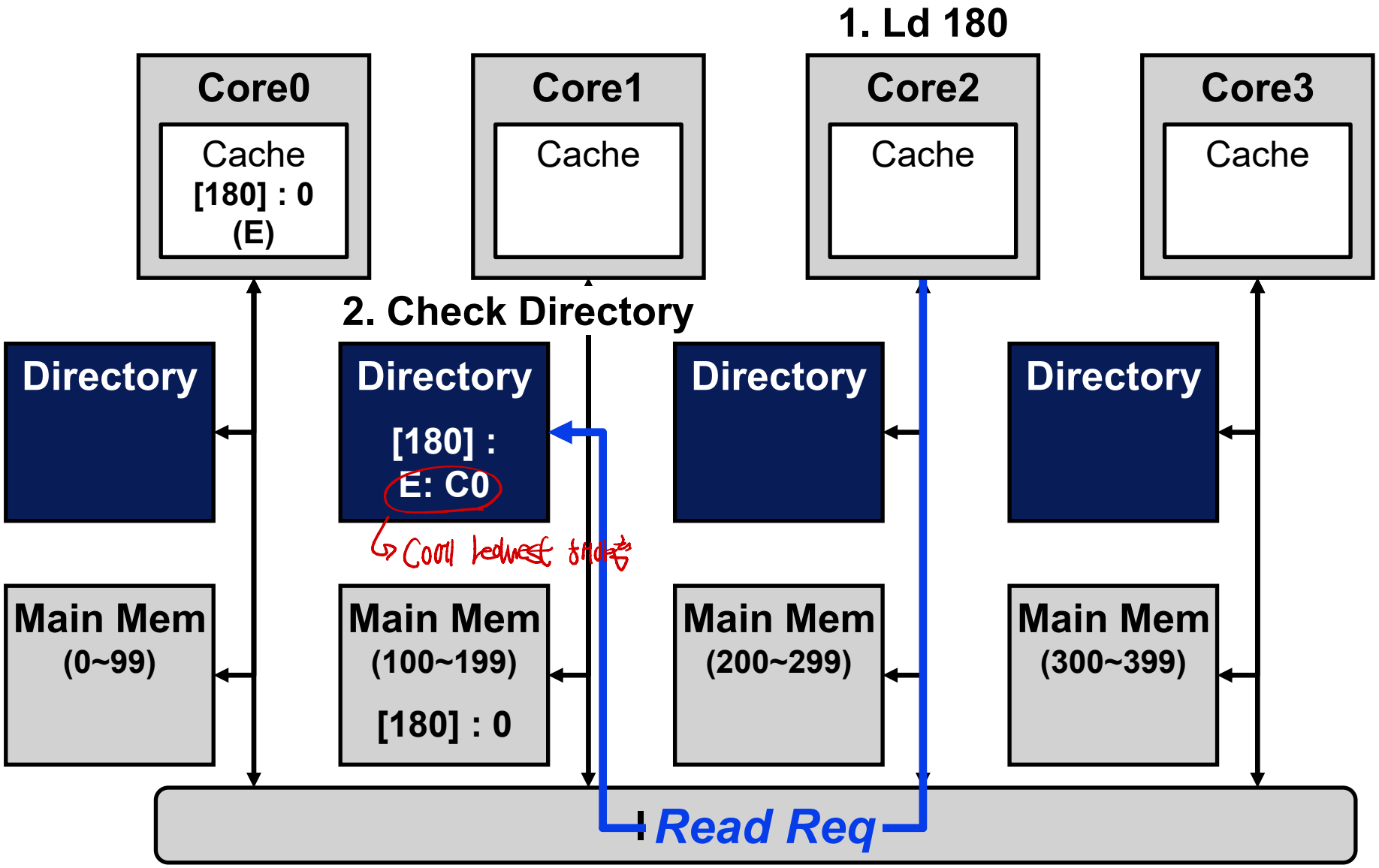
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



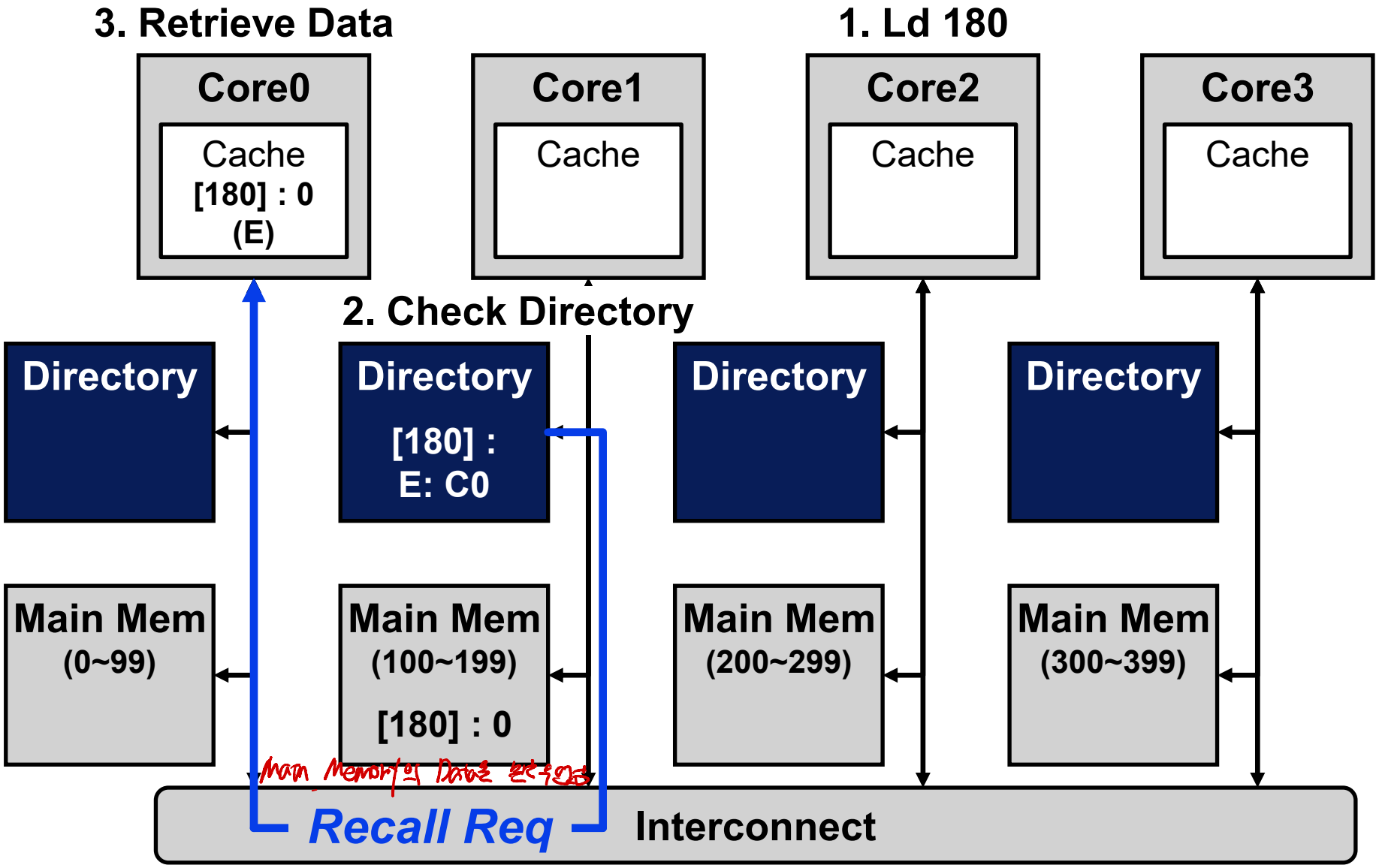
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



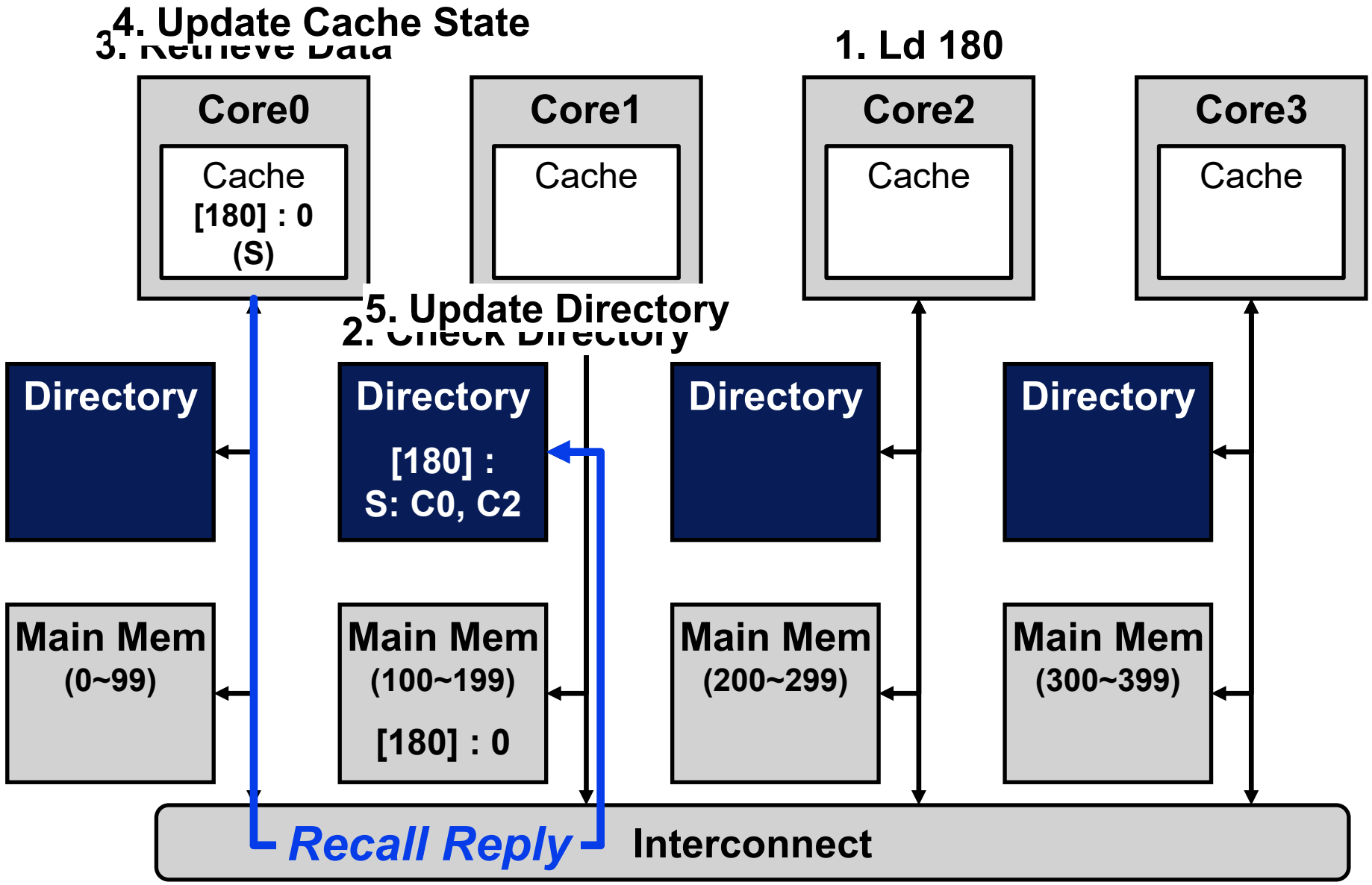
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



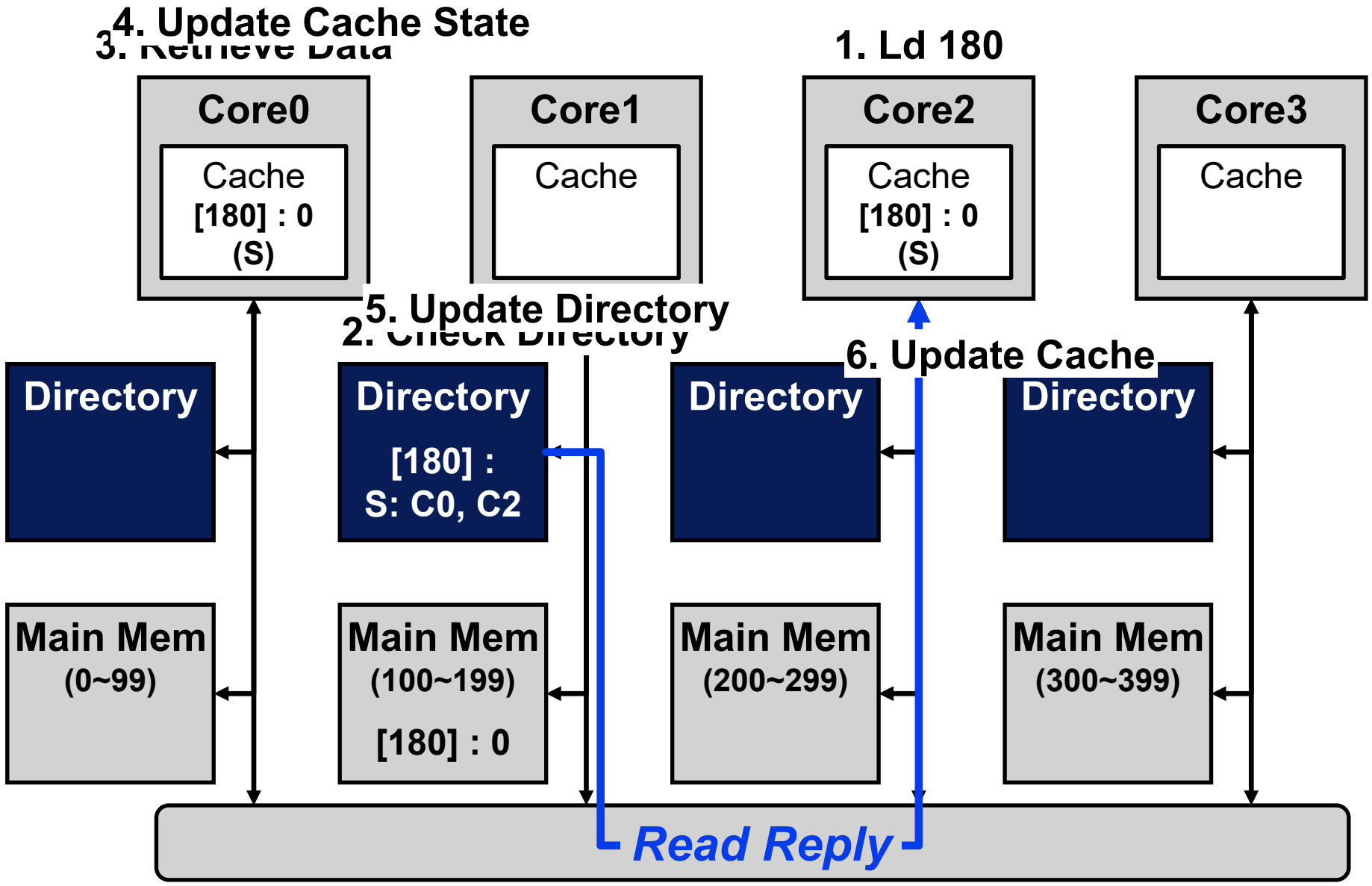
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



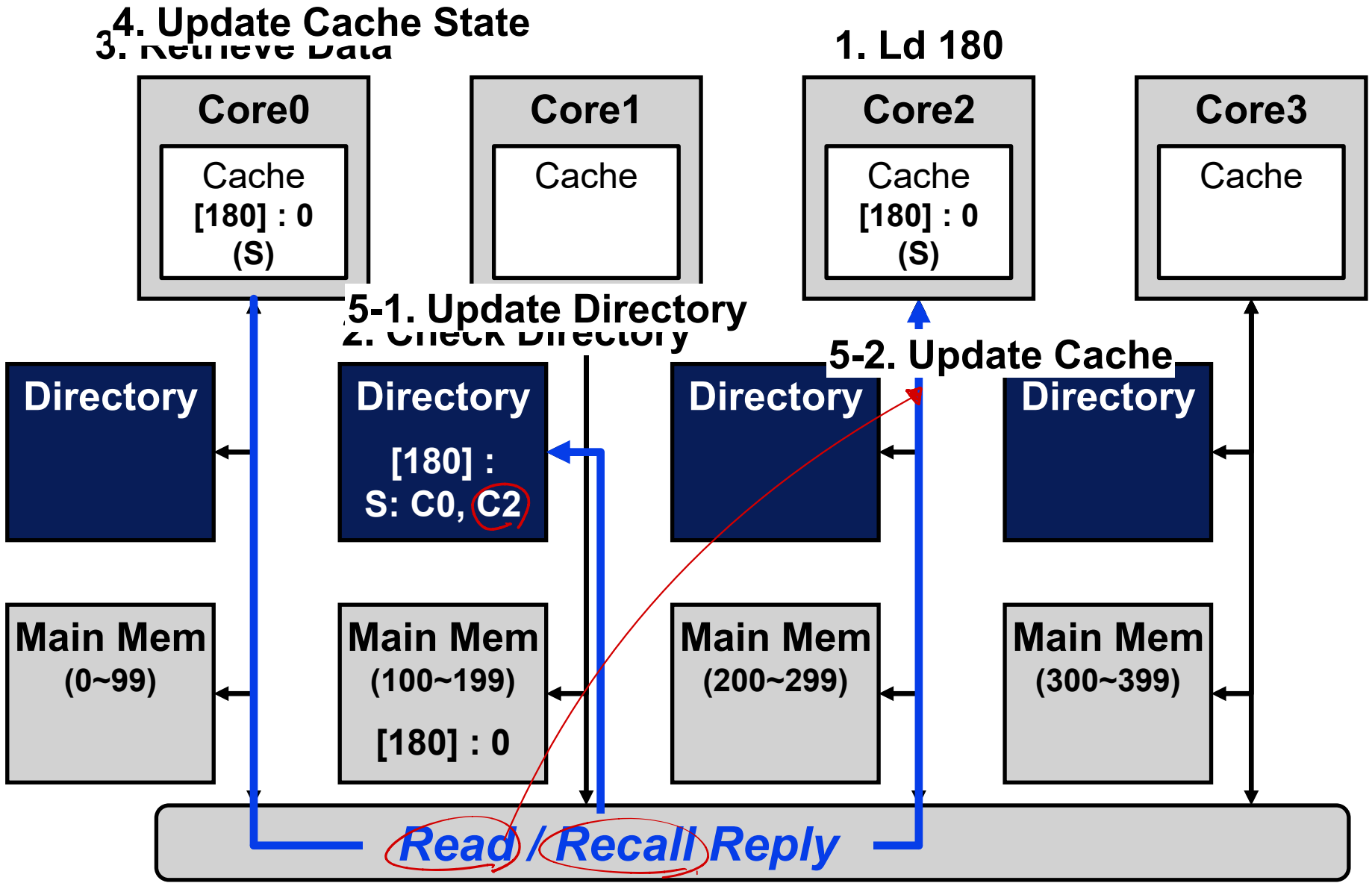
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



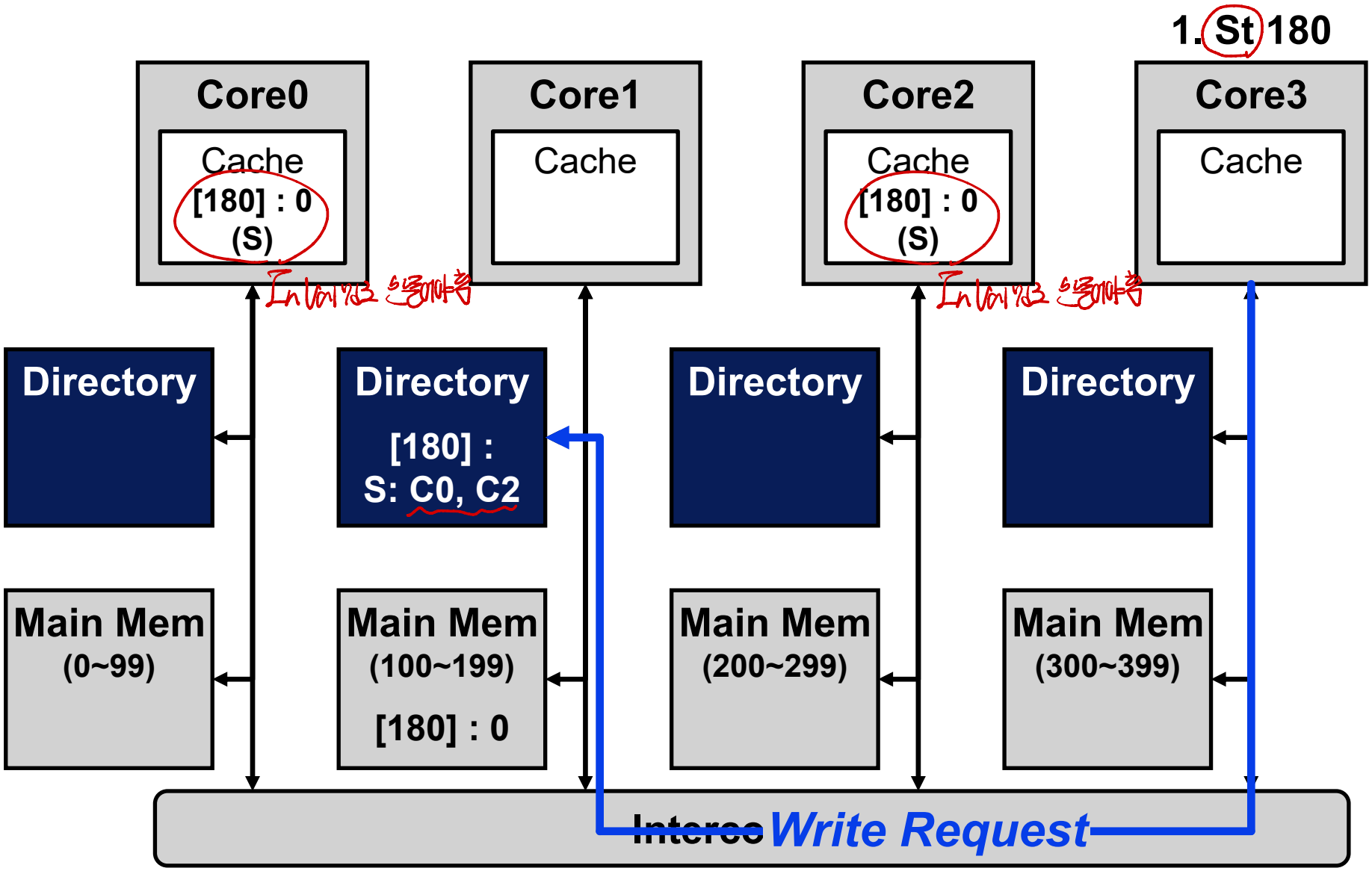
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



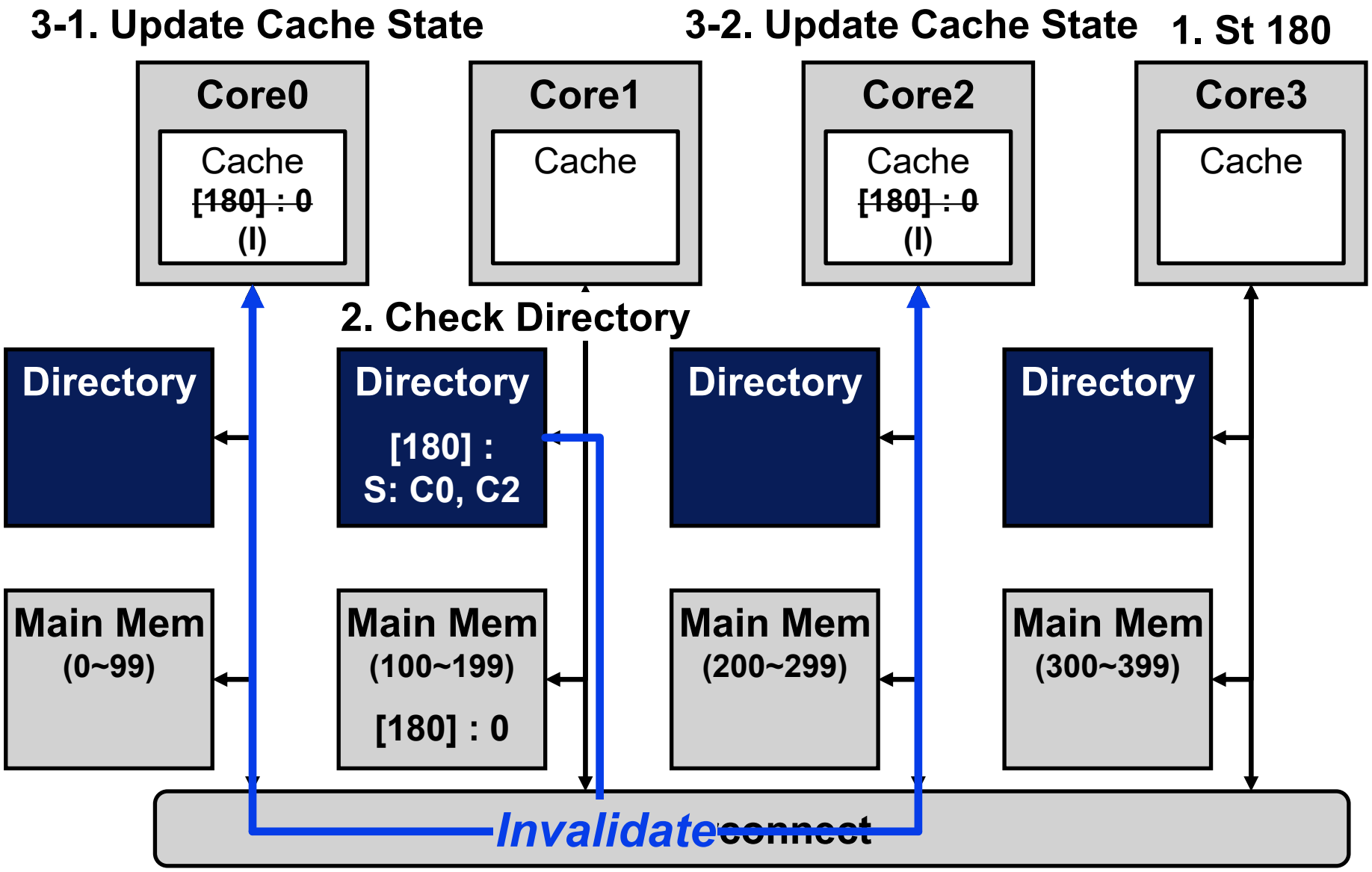
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



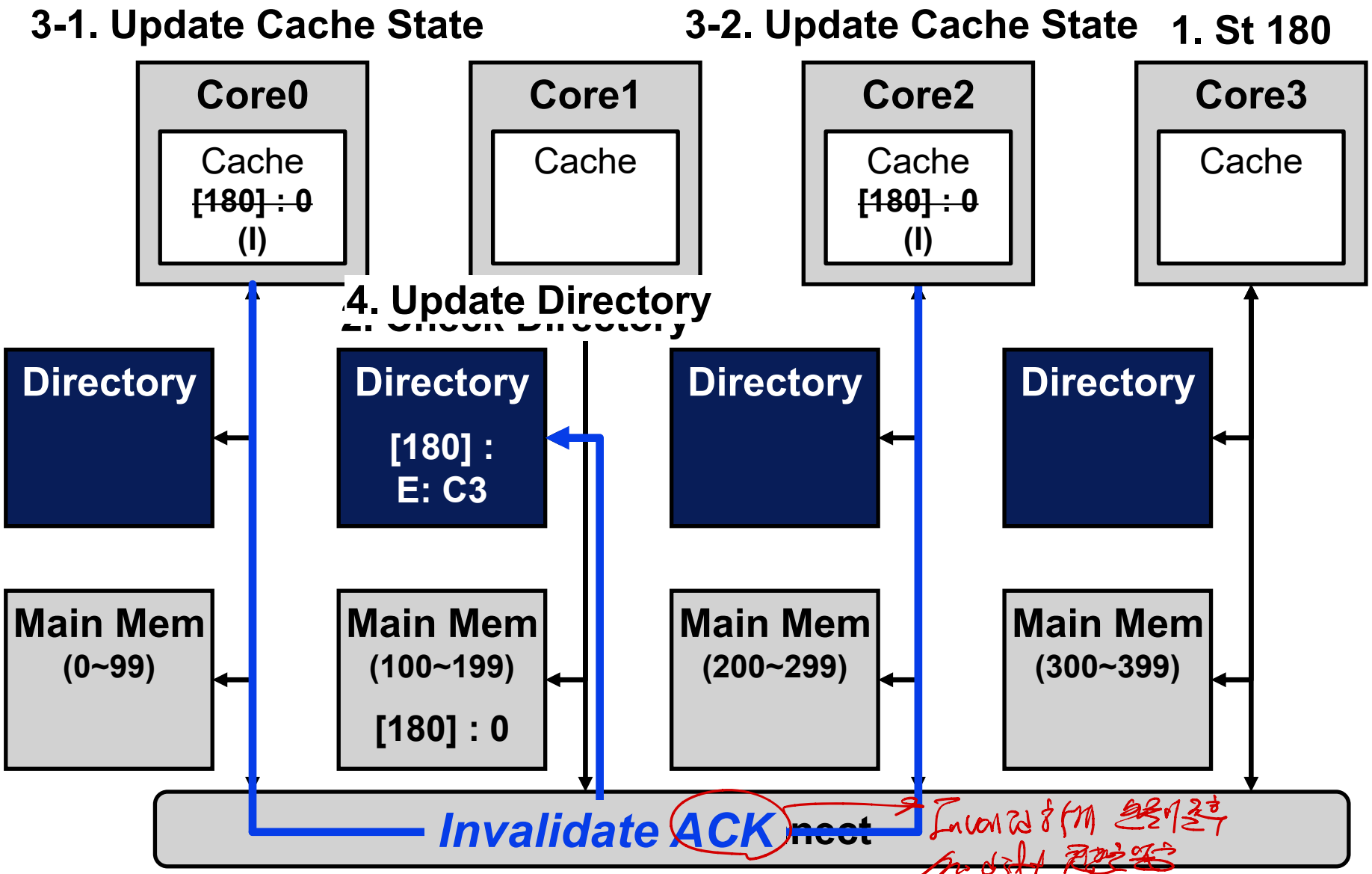
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



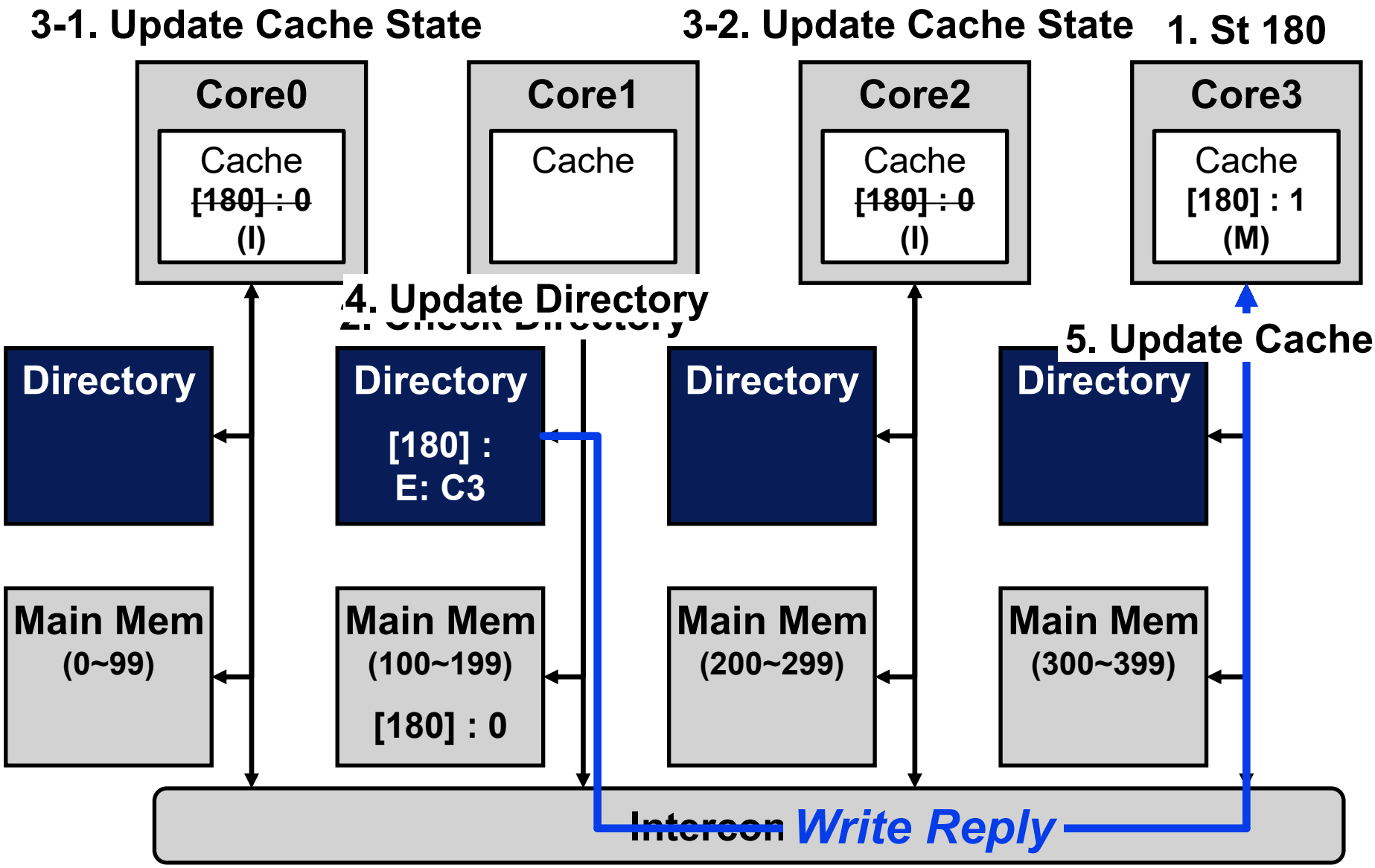
© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Directory Implementation

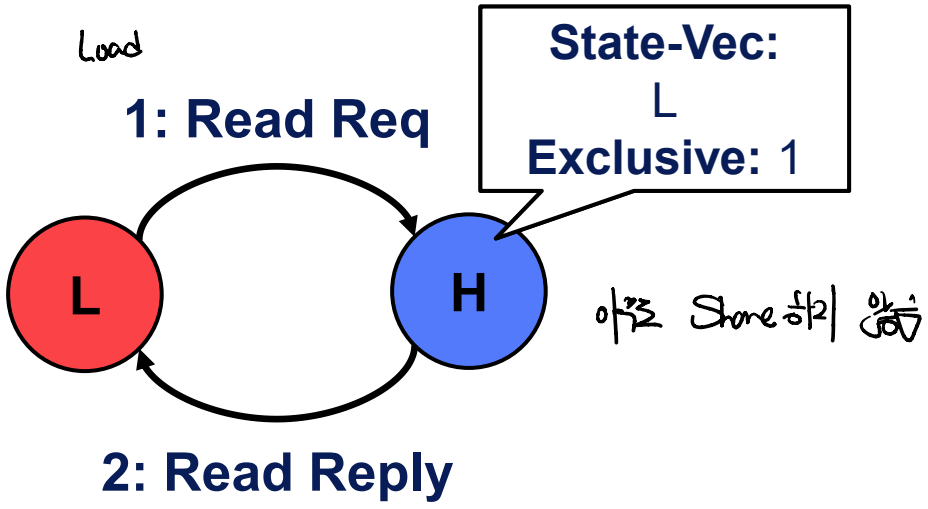


© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Summary: Transition Diagram

- Target nodes:
  - **Local Node (L):** Node initiating the transaction
  - **Home Node (H):** Node having directory/main memory
  - **Remote Node (R):** Any other node

Scenario #1: L has a cache miss on a load instruction

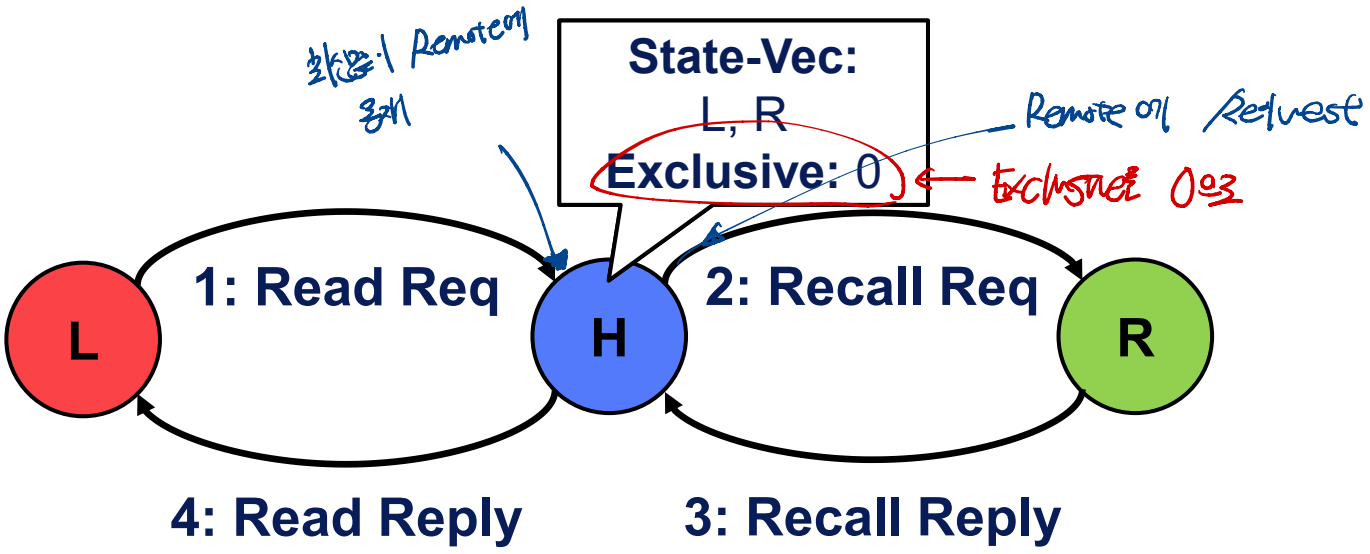


© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Summary: Transition Diagram

- Target nodes:
  - **Local Node (L):** Node initiating the transaction
  - **Home Node (H):** Node having directory/main memory
  - **Remote Node (R):** Any other node

Scenario #2: L has a cache miss on a load instruction (block in modified state at R (4-hop implementation)

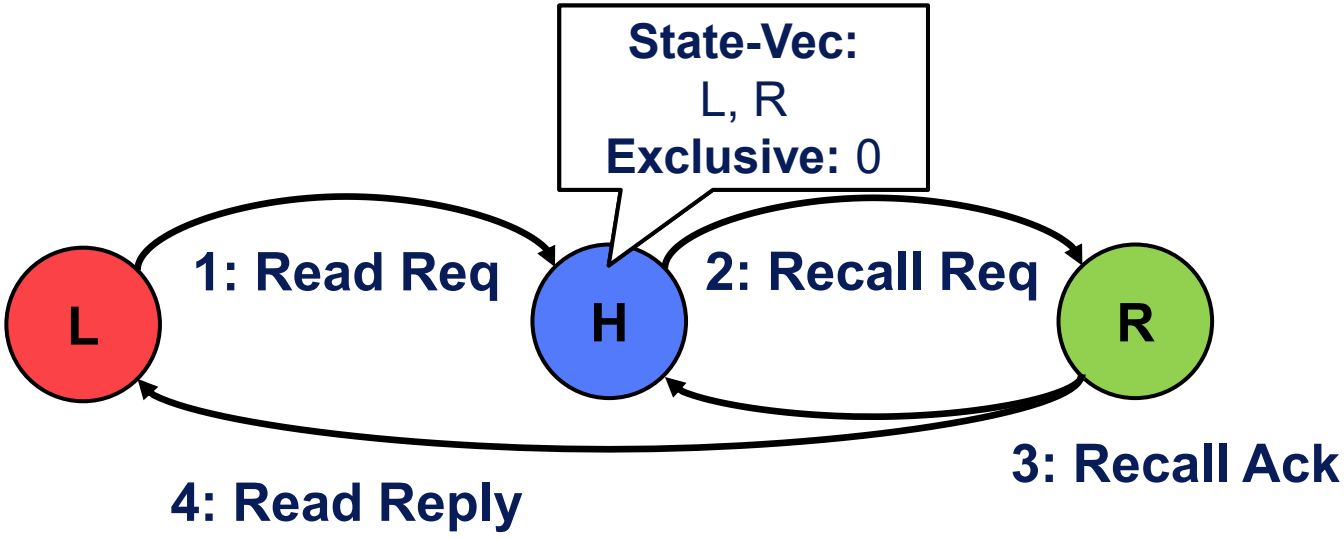


© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Summary: Transition Diagram

- Target nodes:
  - **Local Node (L):** Node initiating the transaction
  - **Home Node (H):** Node having directory/main memory
  - **Remote Node (R):** Any other node

Scenario #2: L has a cache miss on a load instruction (block in modified state at R (3-hop implementation)

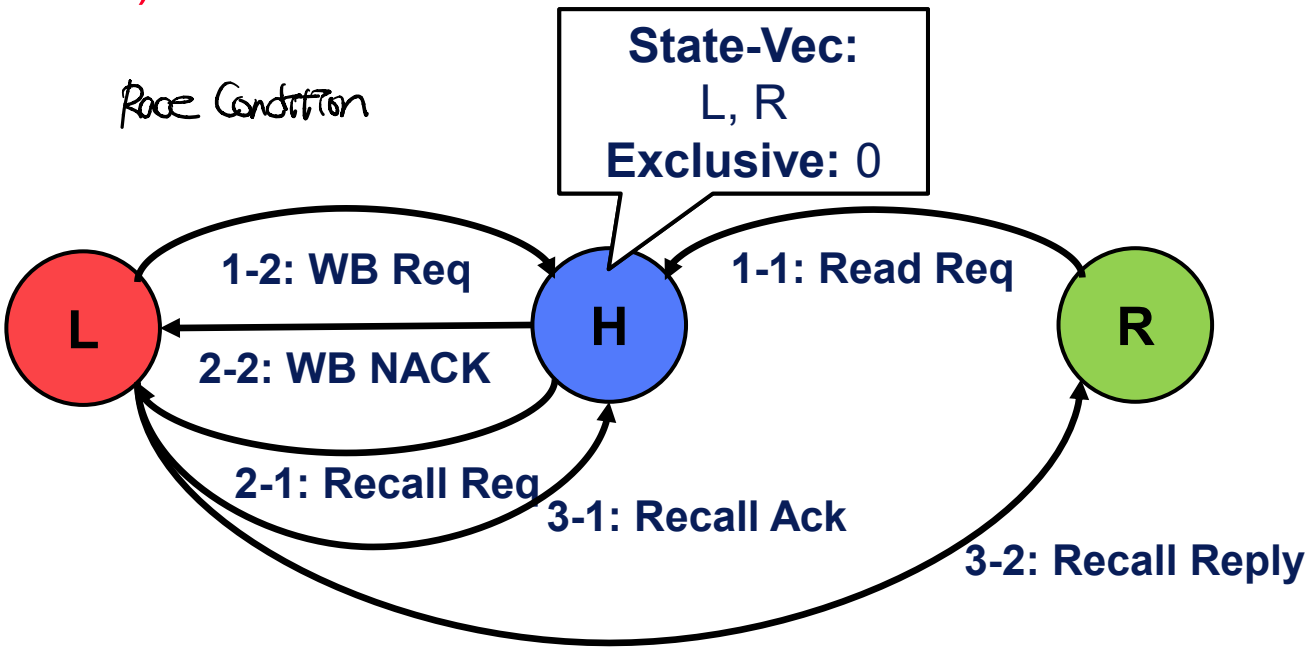


© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Summary: Transition Diagram

- Target nodes:
  - **Local Node (L):** Node initiating the transaction
  - **Home Node (H):** Node having directory/main memory
  - **Remote Node (R):** Any other node

Scenario #3: L writes back to H while R sends a read to H (race condition)

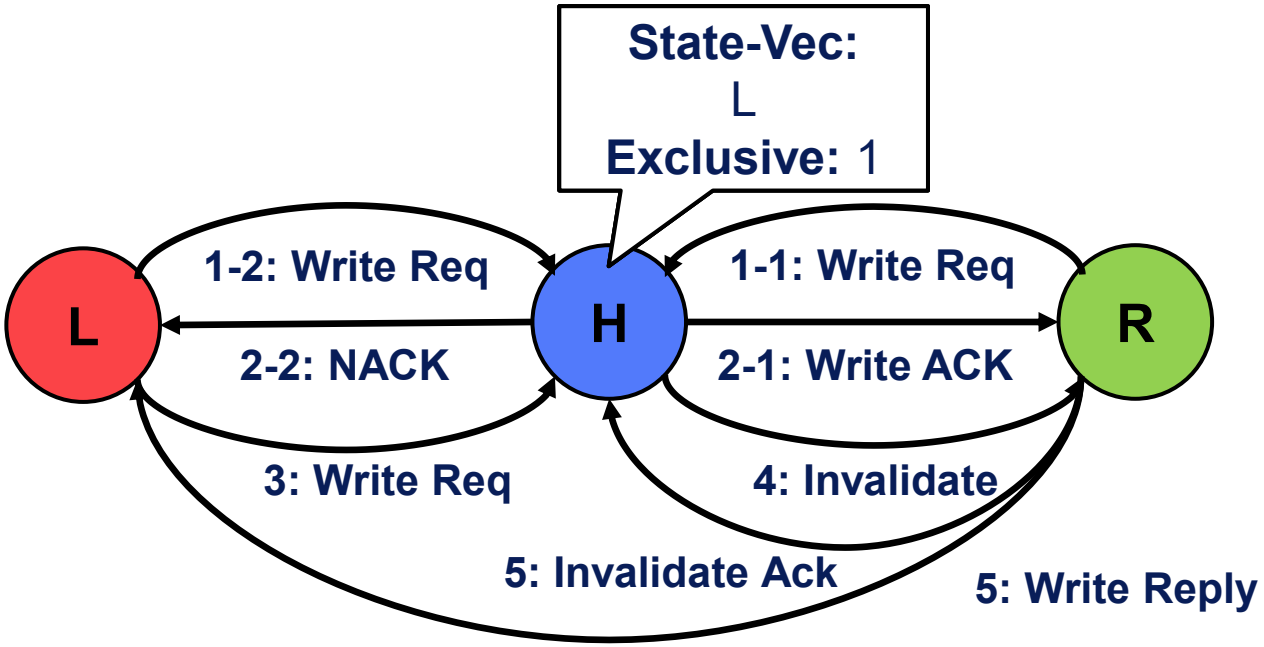


© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Summary: Transition Diagram

- Target nodes:
  - **Local Node (L):** Node initiating the transaction
  - **Home Node (H):** Node having directory/main memory
  - **Remote Node (R):** Any other node

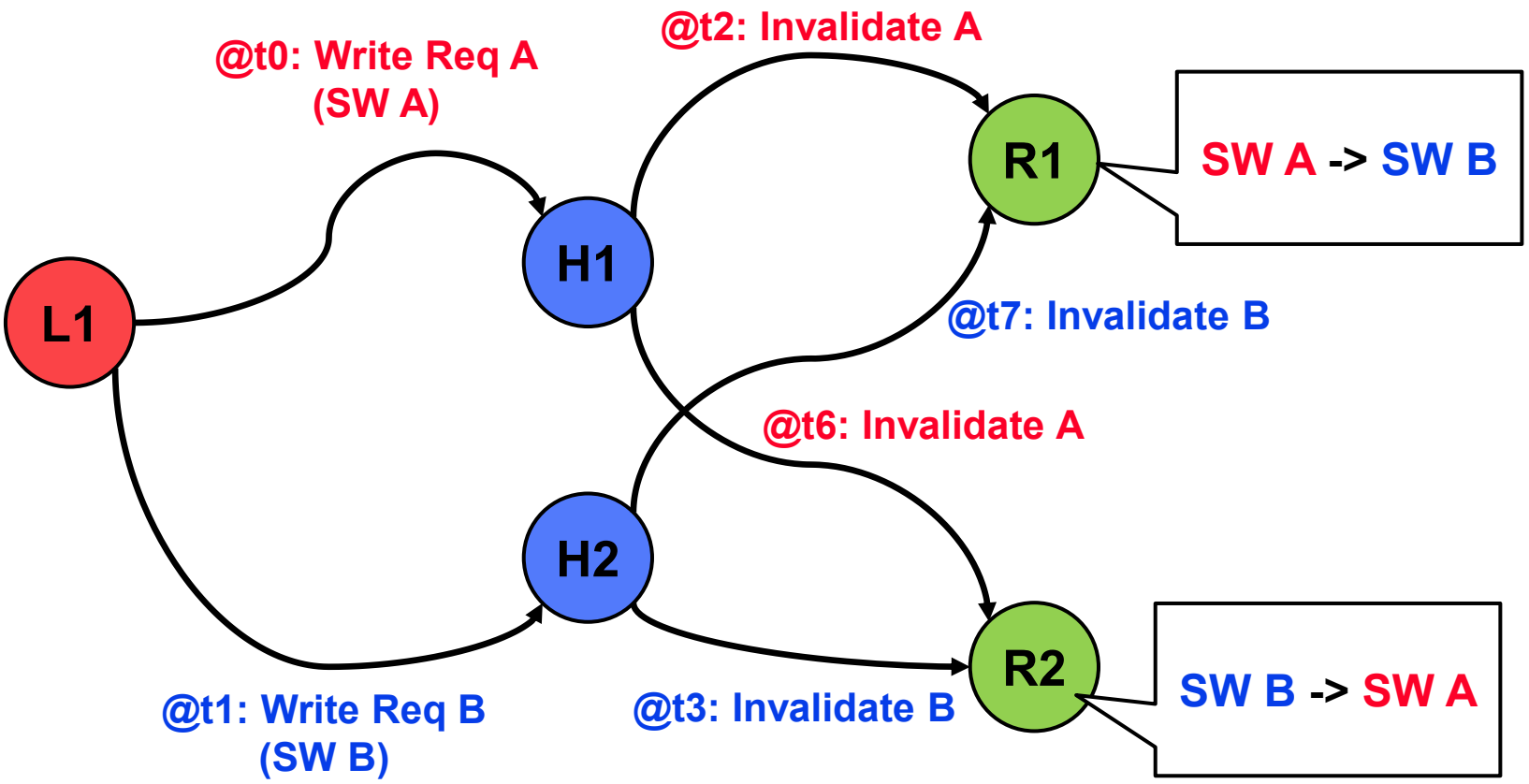
Scenario #3: L and R simultaneously writes data



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Relation to Consistency

- There are various consistency implications behind …



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Now … Let's Talk About Implementation

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

HW Overhead

- This sounds great … but you need too much additional data to maintain the directory

- What is the granularity of the coherence management?
  ➔ Per L3 cache block (typically 512 bit)

- Let's say we have N cores, then you **need "at least" N bit (shared vector) + alpha (exclusive, uncached, …)**
  - 128 cores: 25% overhead
  - 256 cores: 50% overhead
  - 512 cores: 100% overhead
  - 1024 cores: 200% overhead

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

How to reduce the overhead: (Limited) Pointer-Based

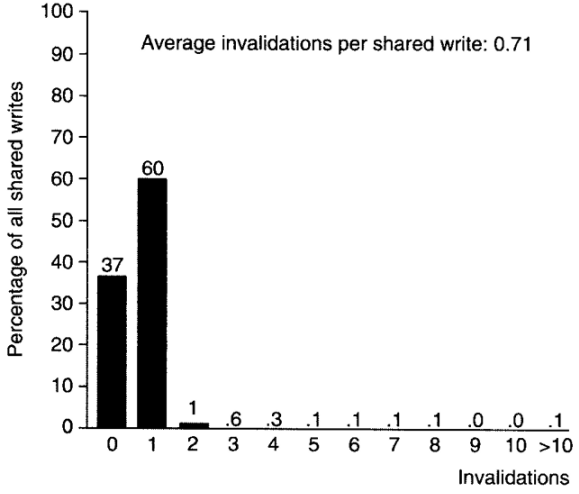
- Motivation: the number of sharers are extremely sparse in most cases
  - Write down the sharers' index in an array
  - E.g., 8-core-system and
    - Bit-vector: 00001000 ➔ index arr: 101
    - Bit-vector: 00000001 ➔ index arr: 111
    - Bit-vector: 00001001 ➔ index arr: 101111

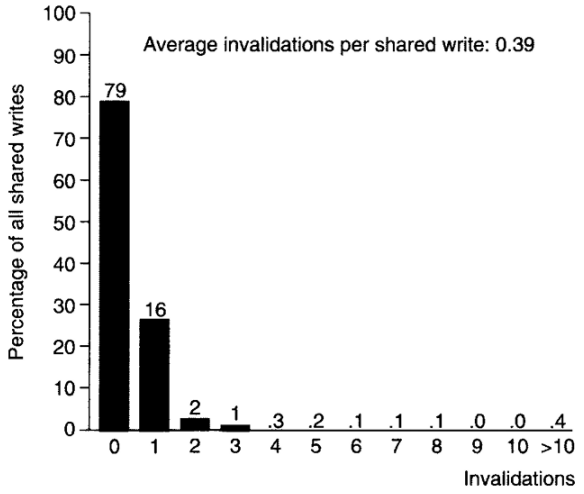
- What about 1024-core-system?
  - 10-bit per index → we still consume less overhead than the bit-vector when having 100 sharers

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

How to reduce the overhead: (Limited) Pointer-Based

- How many pointers to allow? (We need 1024 in the worst-case)
  - Maybe 2~4? There are only a small number of sharers in typical use cases



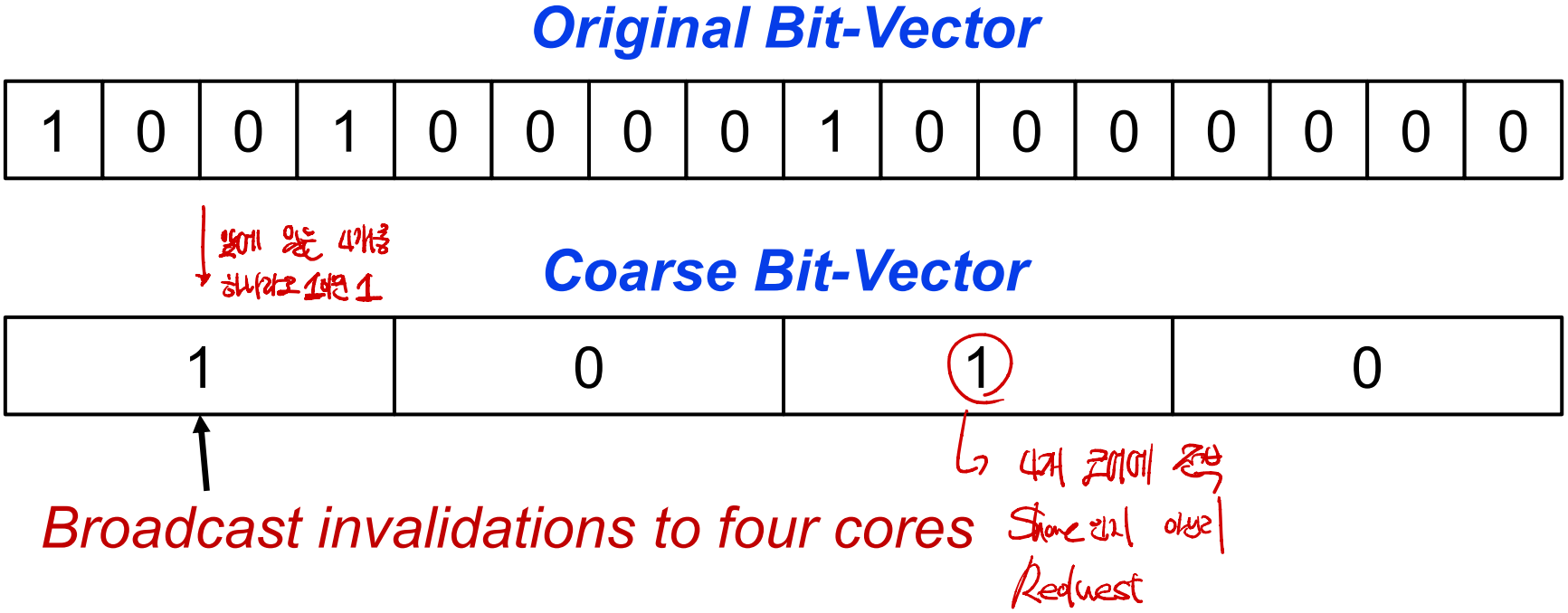


  - What if we have more than 4 sharers? HW Overflow ☹ ➔ Then, just broadcast the data to all the cores
  - The cores can simply filter out invalid requests

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

How to reduce the overhead: Coarse Bit-Vector

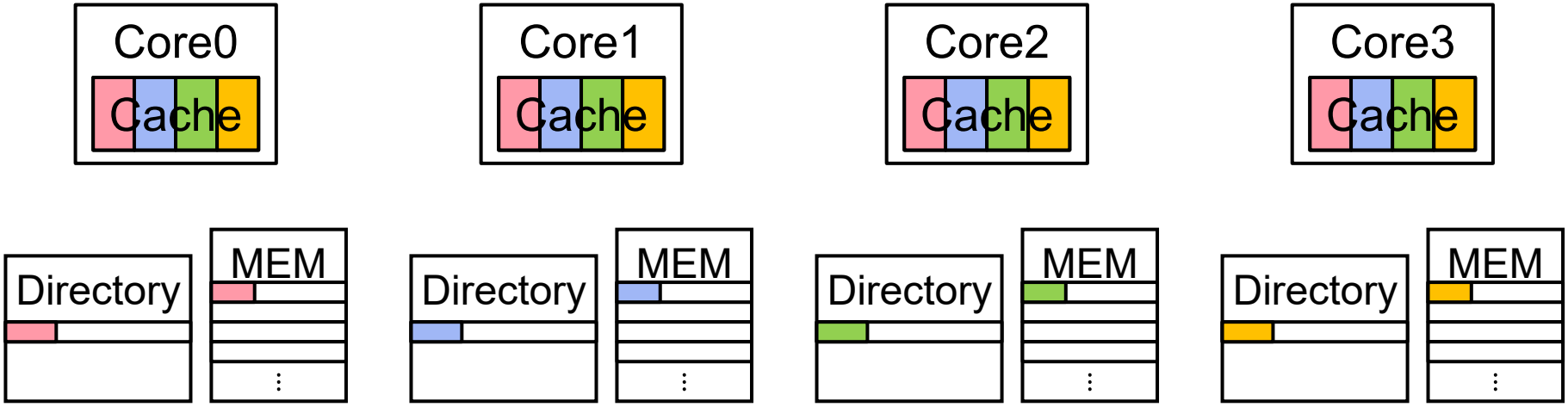
- Hierarchically manage the sharers
- We can combine multiple cores to share a single sharer bit



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

How to reduce the overhead: Sparse Directories

- Need to keep the directories for the entire memory? No!
- Why? Because not all the data are in the cache… We do not need to keep the uncached data to the directories!
  - How many entries? "At most" P * C entries per directory
    - where P: # of processors and C: # of cache lines
  - What happens when we do not have enough space for P * C entries

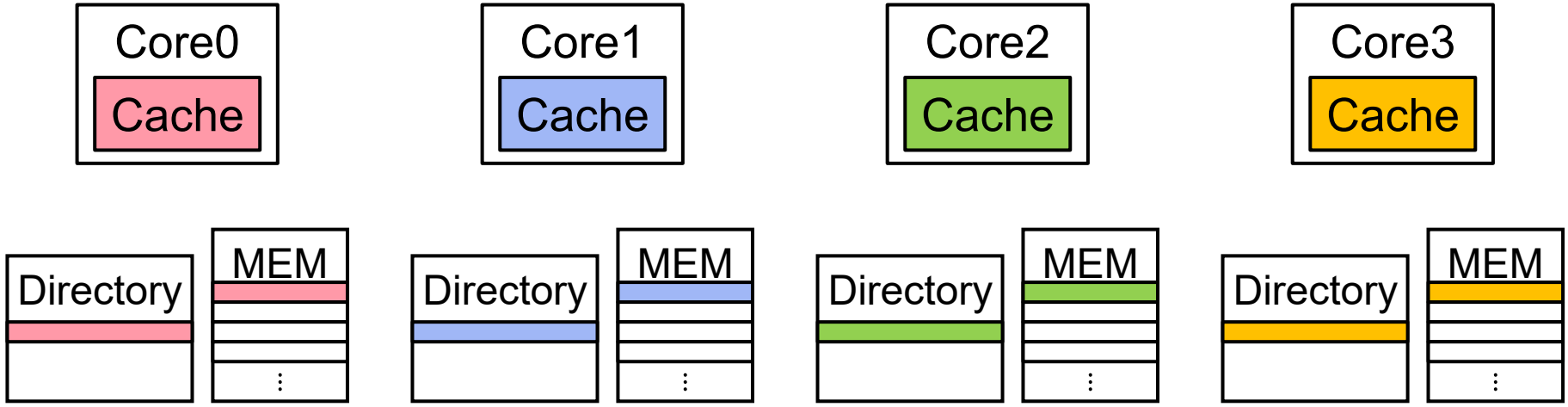


Directory only keeps C/P entries in the best case

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

How to reduce the overhead: Sparse Directories

- Need to keep the directories for the entire memory? No!
- Why? Because not all the data are in the cache… We do not need to keep the uncached data to the directories!
  - How many entries? "At most" P * C entries per directory
    - where P: # of processors and C: # of cache lines
  - What happens when we do not have enough space for P * C entries

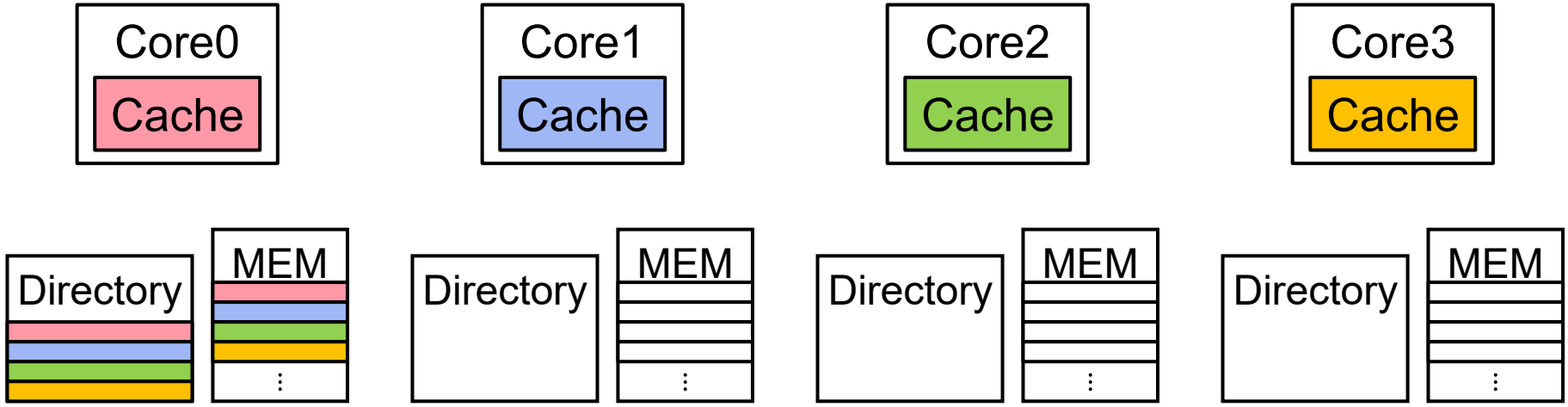


In this case, we need C entries

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

How to reduce the overhead: Sparse Directories

- Need to keep the directories for the entire memory? No!
- Why? Because not all the data are in the cache… We do not need to keep the uncached data to the directories!
  - How many entries? "At most" P * C entries per directory
    - where P: # of processors and C: # of cache lines
  - What happens when we do not have enough space for P * C entries

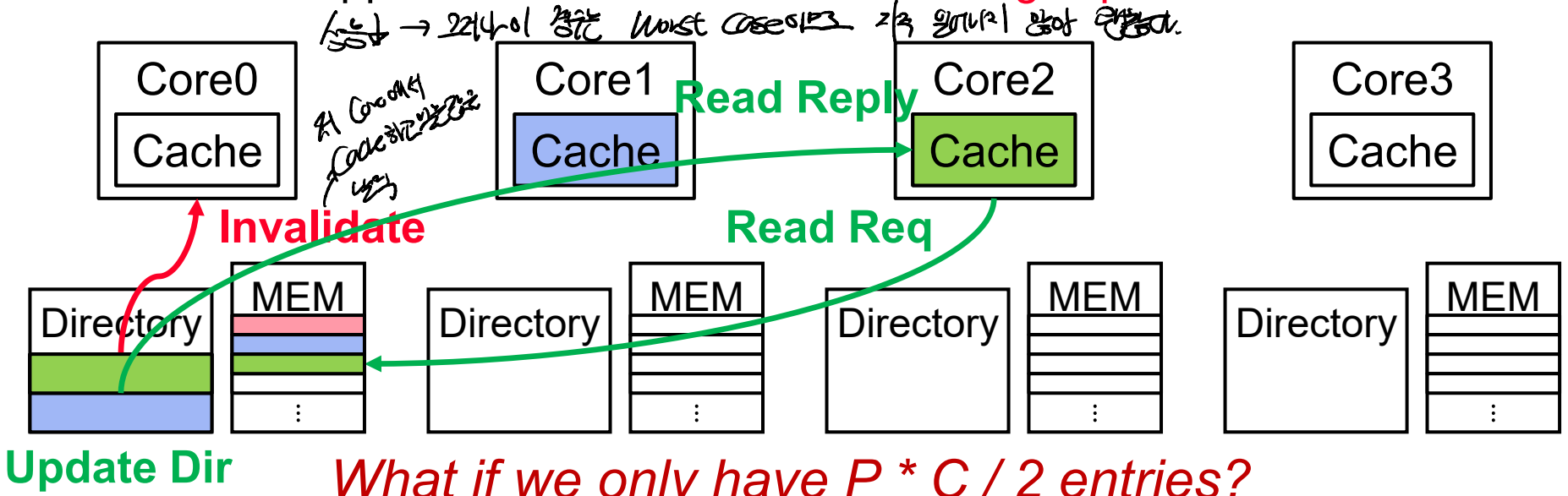


*In the worst case, we need P * C entries*

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

How to reduce the overhead: Sparse Directories

- Need to keep the directories for the entire memory? No!
- Why? Because not all the data are in the cache… We do not need to keep the uncached data to the directories!
  - How many entries? "At most" P * C entries per directory
    - where P: # of processors and C: # of cache lines
  - What happens when we do not have enough space for P * C entries



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Question?

Announcements:

Reading: *read P&H Ch.6*

Handouts: *none*

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim