EEL 5764 Computer Architecture

Lecture 32-33: Multiprocessing

Sandip Ray
Department of Electrical and Computer Engineering
University of Florida

Announcements

- **Midterm Structure Announced in Canvas**
  - → There will be 6 regular problems and two bonus problems
  - → Regular problems only from materials after Midterm 1
  - → Bonus problems from anywhere
  - → Bonus points added to lower of the two midterm grades
  - → Up to a maximum of 70 (nobody will receive more than 70 on any midterm even with bonus)
  - → Bonus scores will not be distributed across midterms
- **I have recorded one lecture for missed classes, two others coming Wednesday (11/20) and Monday (11/25)**
- **Announcements on Project Report structure will be posted on Canvas**

Cache Coherence Problem

- Processors may see different values through their private caches

Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

Cache Coherence

- Coherence – what values returned for reads
  - A read by a processor A to a location X that follows a write to X by A returns the value written by A if no other processors write in between
  - A read by processor A to location X after a write to X by processor B returns the written value if the read and write are sufficiently separated, and no other writes occur in between
  - Writes to the same location are serialized

- Consistency – when a written value seen by a read
  - Concerns reads & writes to different memory locations from multiple processors

Enforcing Coherence

- Coherent caches provide:
  - *Migration*: movement of data – reduce latency
  - *Replication*: multiple copies of data – reduce latency & memory bandwidth demand

- Cache coherence protocols
  - Snooping
    - Every cache tracks sharing status of each cache block
    - Mem requests are broadcast on a bus to all caches
    - Writes serialized naturally
  - Directory based
    - Sharing status of each cache block kept in one location

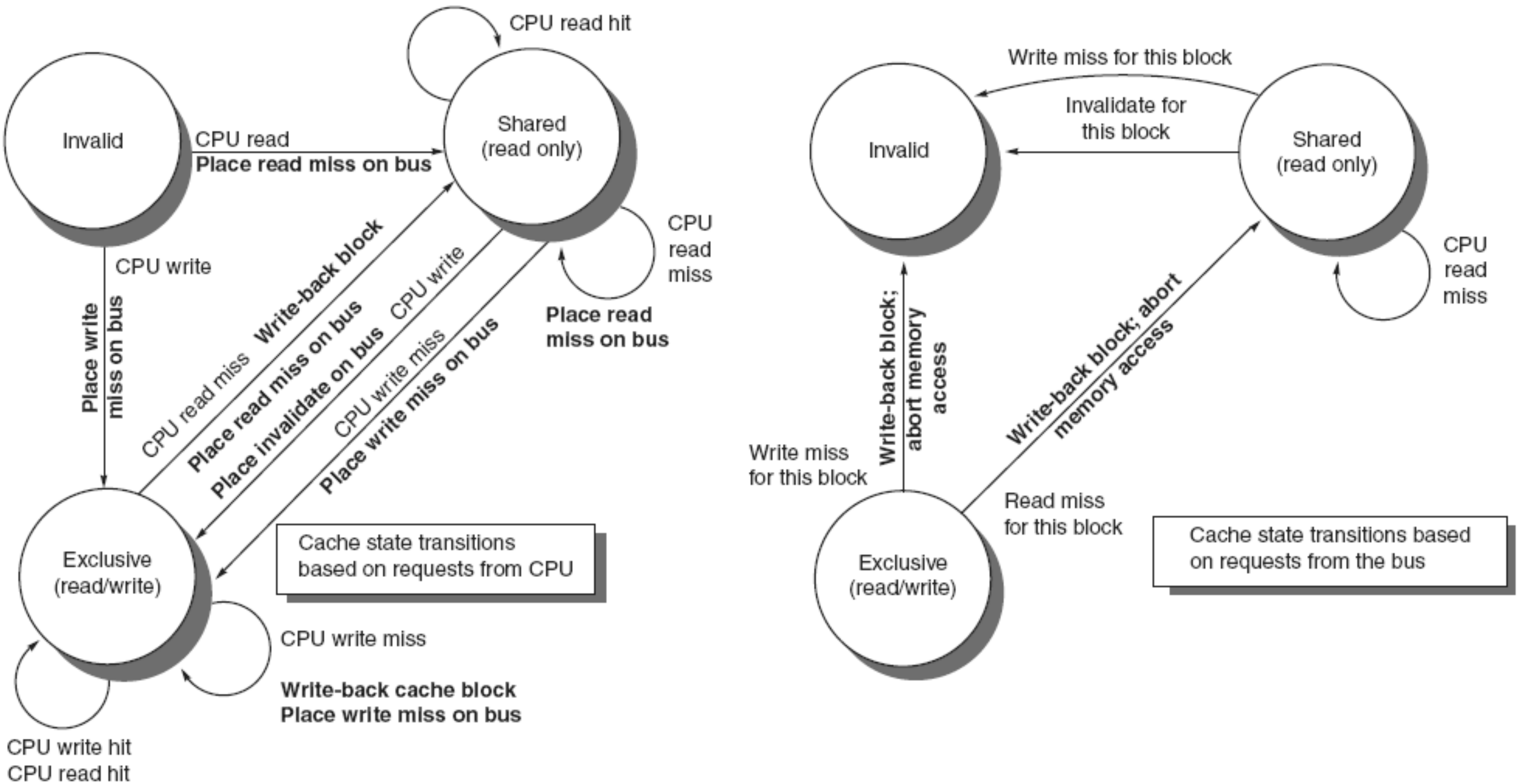
Snoopy Coherence Protocols

- Each cache block is in one of following states
  - Invalid (I)
  - Shared (S)
  - Modified (M) – implies exclusion or not shared

- Locating an item when a read miss occurs
  - In write-back cache, the updated value must be sent to the requesting processor

- Cache lines marked as shared or exclusive/modified
  - Only writes to shared lines need an invalidate broadcast
    - After this, the line is marked as exclusive

Snoopy Coherence Protocols



Snoopy Coherence Protocols

Request	Source	State of addressed cache block	Type of cache action	Function and explanation
Read hit	Processor	Shared or modified	Normal hit	Read data in local cache.
Read miss	Processor	Invalid	Normal miss	Place read miss on bus.
Read miss	Processor	Shared	Replacement	Address conflict miss: place read miss on bus.
Read miss	Processor	Modified	Replacement	Address conflict miss: write-back block, then place read miss on bus.
Write hit	Processor	Modified	Normal hit	Write data in local cache.
Write hit	Processor	Shared	Coherence	Place invalidate on bus. These operations are often called upgrade or *ownership* misses, since they do not fetch the data but only change the state.
Write miss	Processor	Invalid	Normal miss	Place write miss on bus.
Write miss	Processor	Shared	Replacement	Address conflict miss: place write miss on bus.
Write miss	Processor	Modified	Replacement	Address conflict miss: write-back block, then place write miss on bus.
Read miss	Bus	Shared	No action	Allow shared cache or memory to service read miss.
Read miss	Bus	Modified	Coherence	Attempt to share data: place cache block on bus and change state to shared.
Invalidate	Bus	Shared	Coherence	Attempt to write shared block; invalidate the block.
Write miss	Bus	Shared	Coherence	Attempt to write shared block; invalidate the cache block.
Write miss	Bus	Modified	Coherence	Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache.

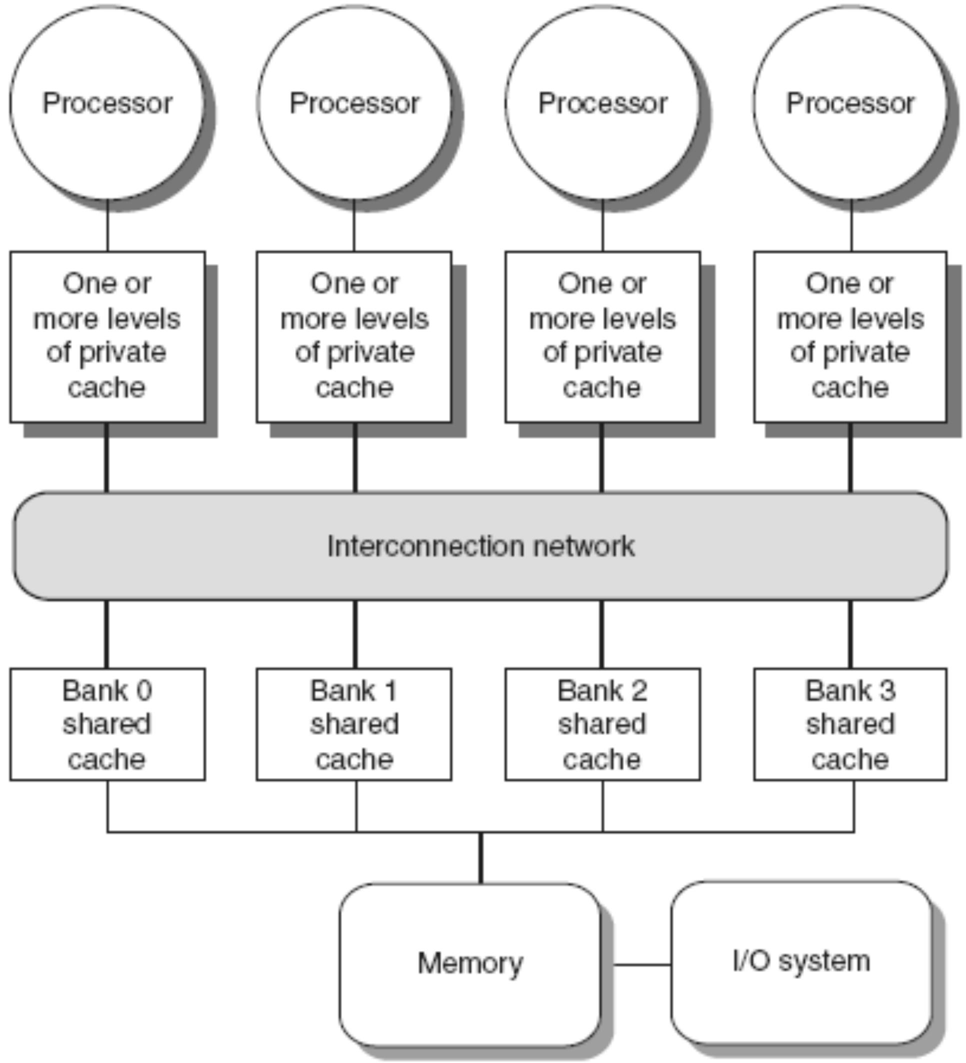
Snoopy Coherence Protocols

- Complications for the basic MSI protocol:
  - Operations are not atomic
    - E.g. detect miss, acquire bus, receive a response
    - Creates possibility of deadlock and races
    - One solution: processor that sends invalidate can hold bus until other processors receive the invalidate

- Extensions – optimize performance
  - Add exclusive (E) state to indicate clean block in only one cache (MESI protocol)
    - No need to write invalidate on a write to a block in E state.
  - MOESI – add Owned (O) state
    - Dirty block is in local caches, but not in the shared cache

Coherence Protocols: Extensions

- Shared memory bus and snooping bandwidth is bottleneck for scaling symmetric multiprocessors
  - Use crossbars or point-to-point networks with banked memory



Distributed Shared-Memory Architecture, Directory based Protocols

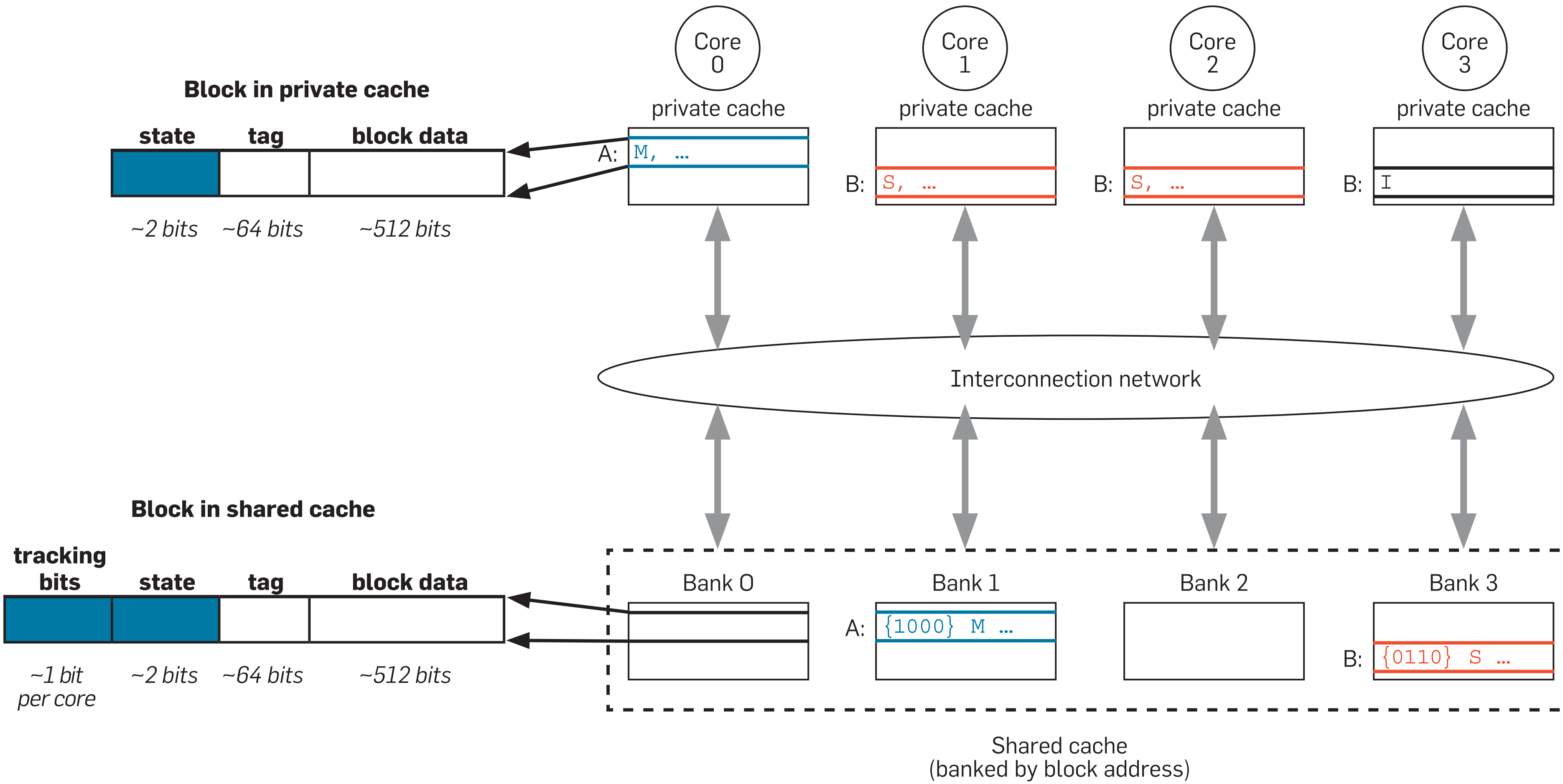
Directory Protocols

- Directory maintains block states and sends invalidation messages
  - Tracks states of all local memory blocks (simplest sol.)

- For each block, maintain state:
  - Shared
    - One or more nodes have the block cached, value in memory is up-to-date
  - Uncached – invalid
  - Modified – Exclusive
    - Exactly one node has a copy of the cache block, value in memory is out-of-date
    - This node is the owner

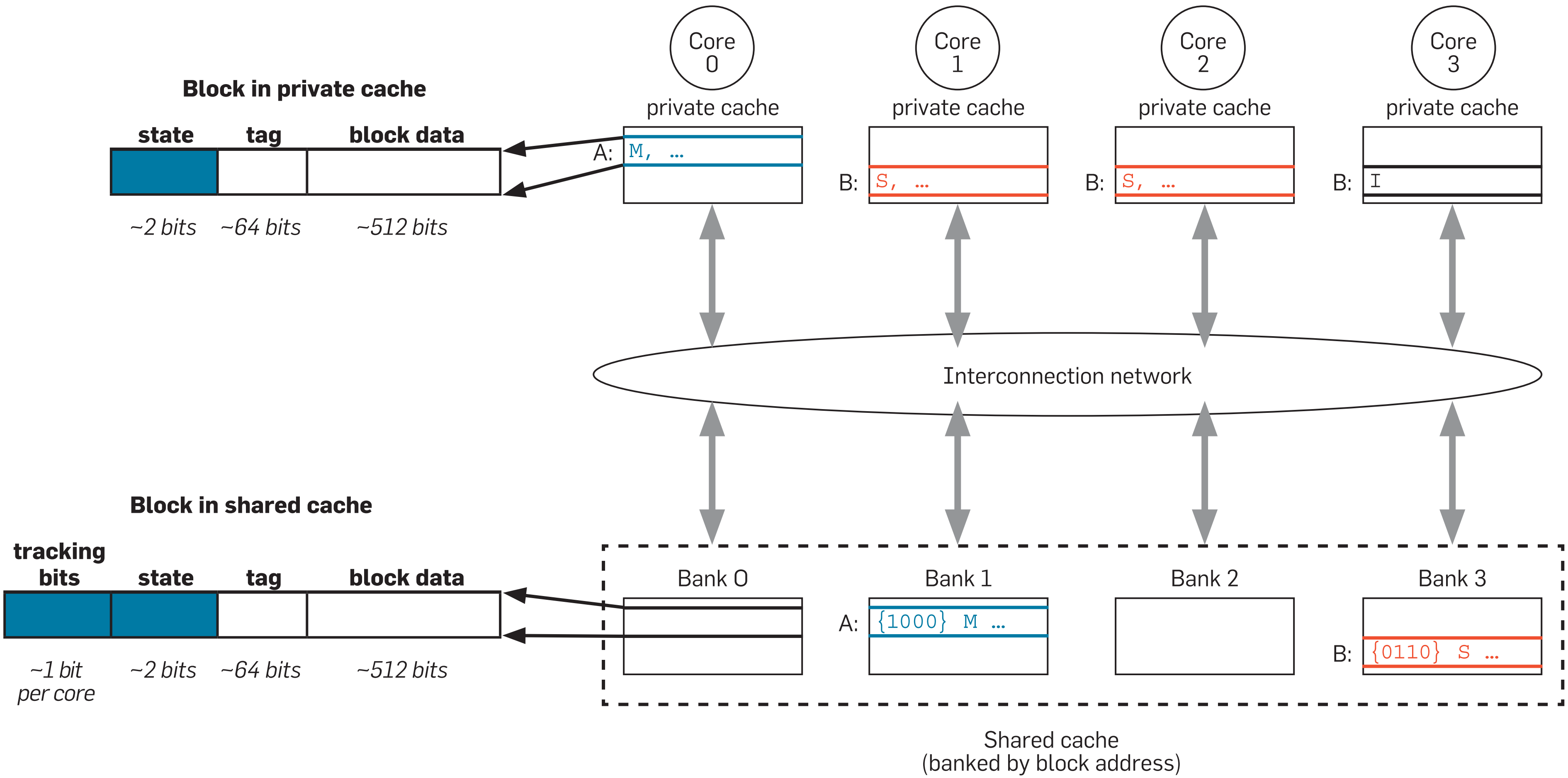
Directory Protocols

- Directory keeps track of every block
  - Sharing caches and dirty status of each block



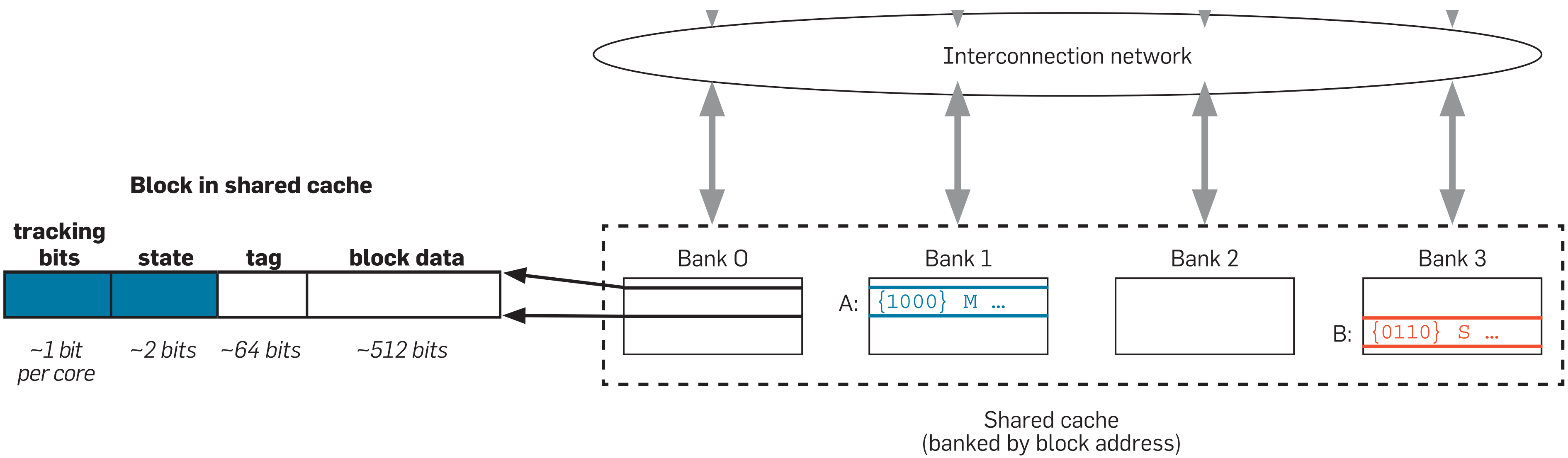
Directory Protocols

- Implement in shared L3 cache
  - Status bit vector, its size = # cores for each block in L3

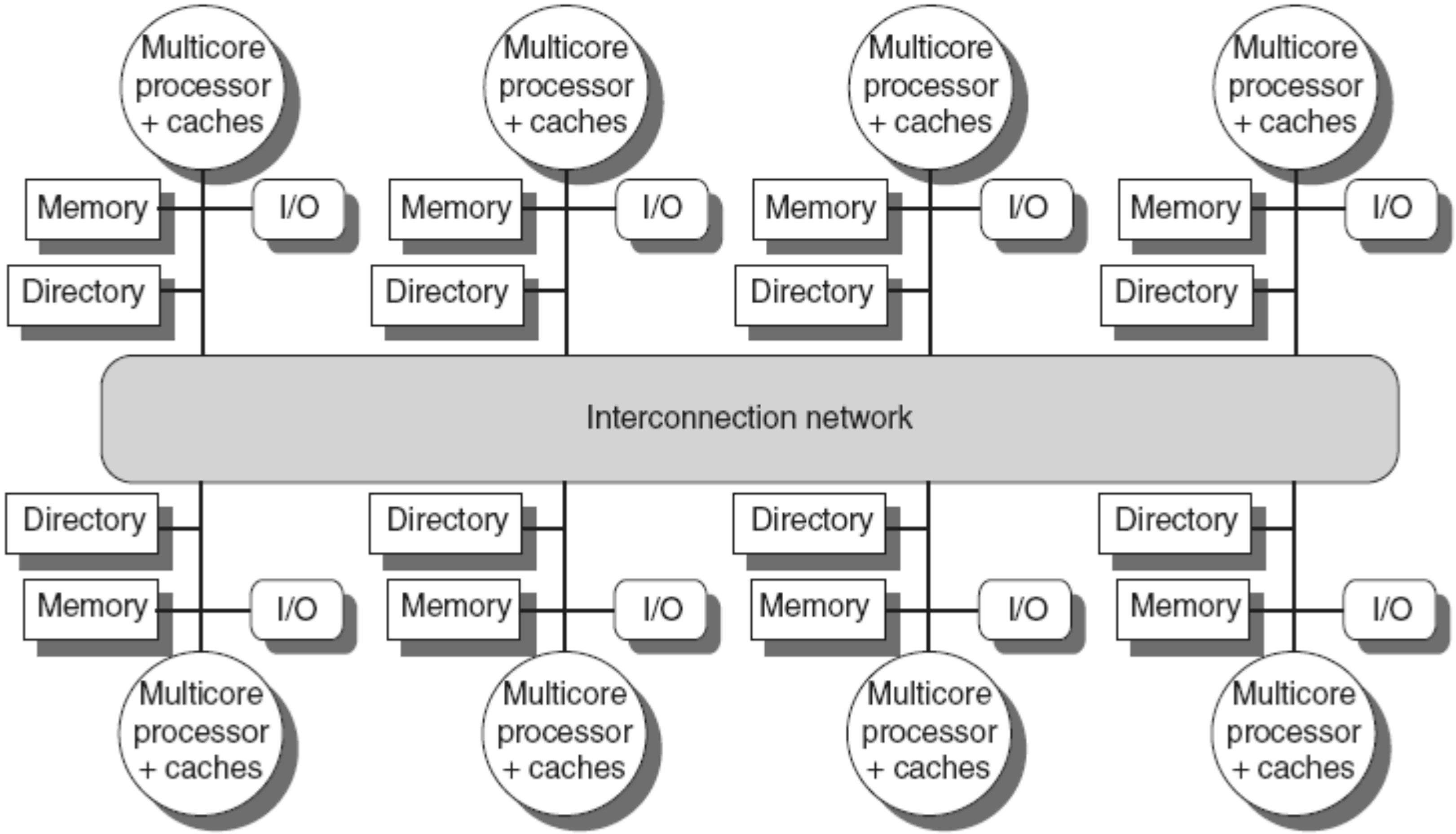


Directory Protocols

- Not scalable beyond L3 cache – centralized cache is bottleneck
- Must be implemented in a distributed fashion
  - Each distributed memory has a directory
  - size = # memory blocks X # nodes



Directory Protocols in DSM



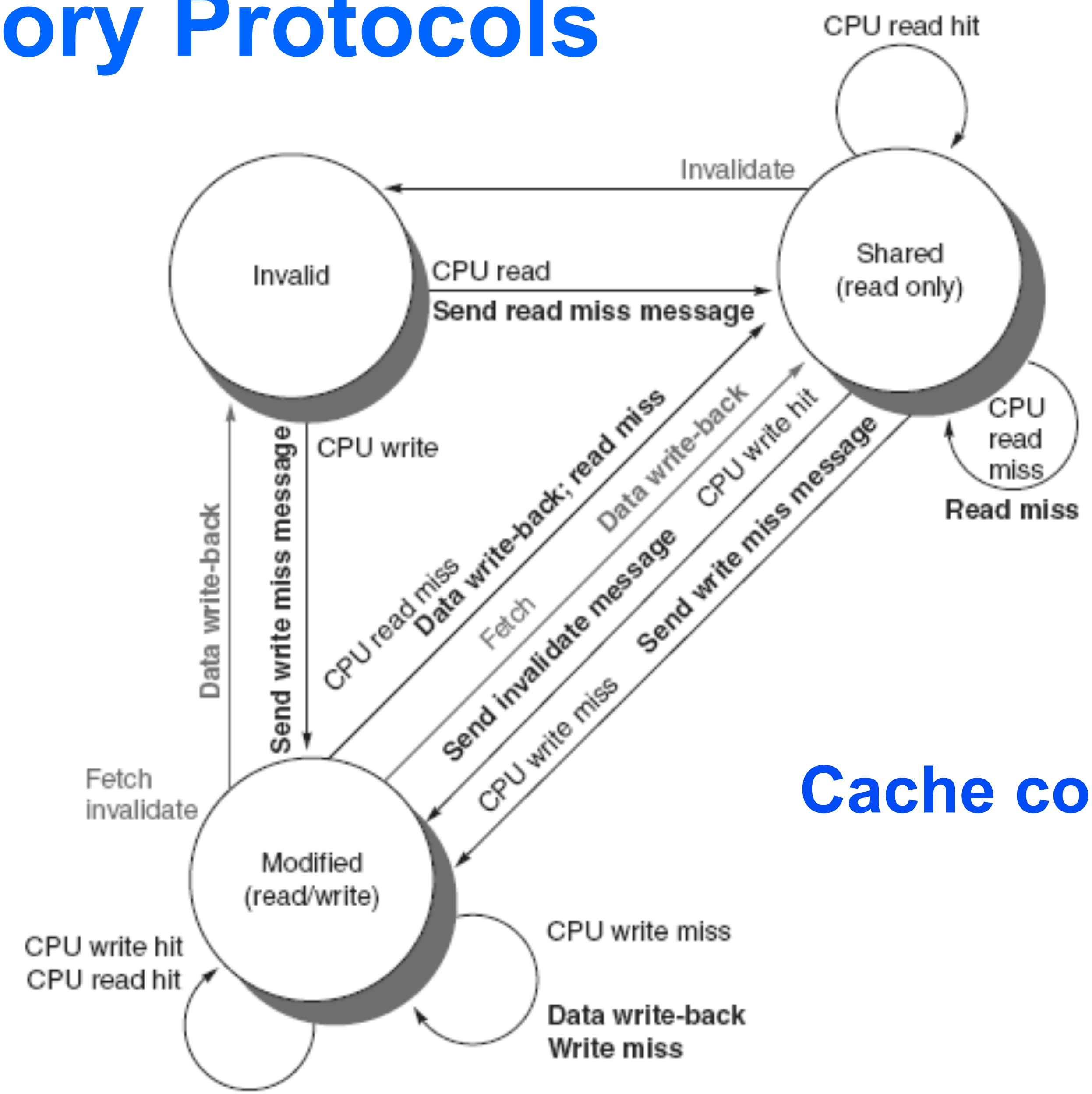
Local directory only stores coherence information of cache blocks in local memory

Messages

Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Node P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Node P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	Local cache	Home directory	A	Request to send invalidates to all remote caches that are caching the block at address A.
Invalidate	Home directory	Remote cache	A	Invalidate a shared copy of data at address A.
Fetch	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
Fetch/invalidate	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache.
Data value reply	Home directory	Local cache	D	Return a data value from the home memory.
Data write-back	Remote cache	Home directory	A, D	Write-back a data value for address A.

Local node: source of requests
Home node: destination of the requests

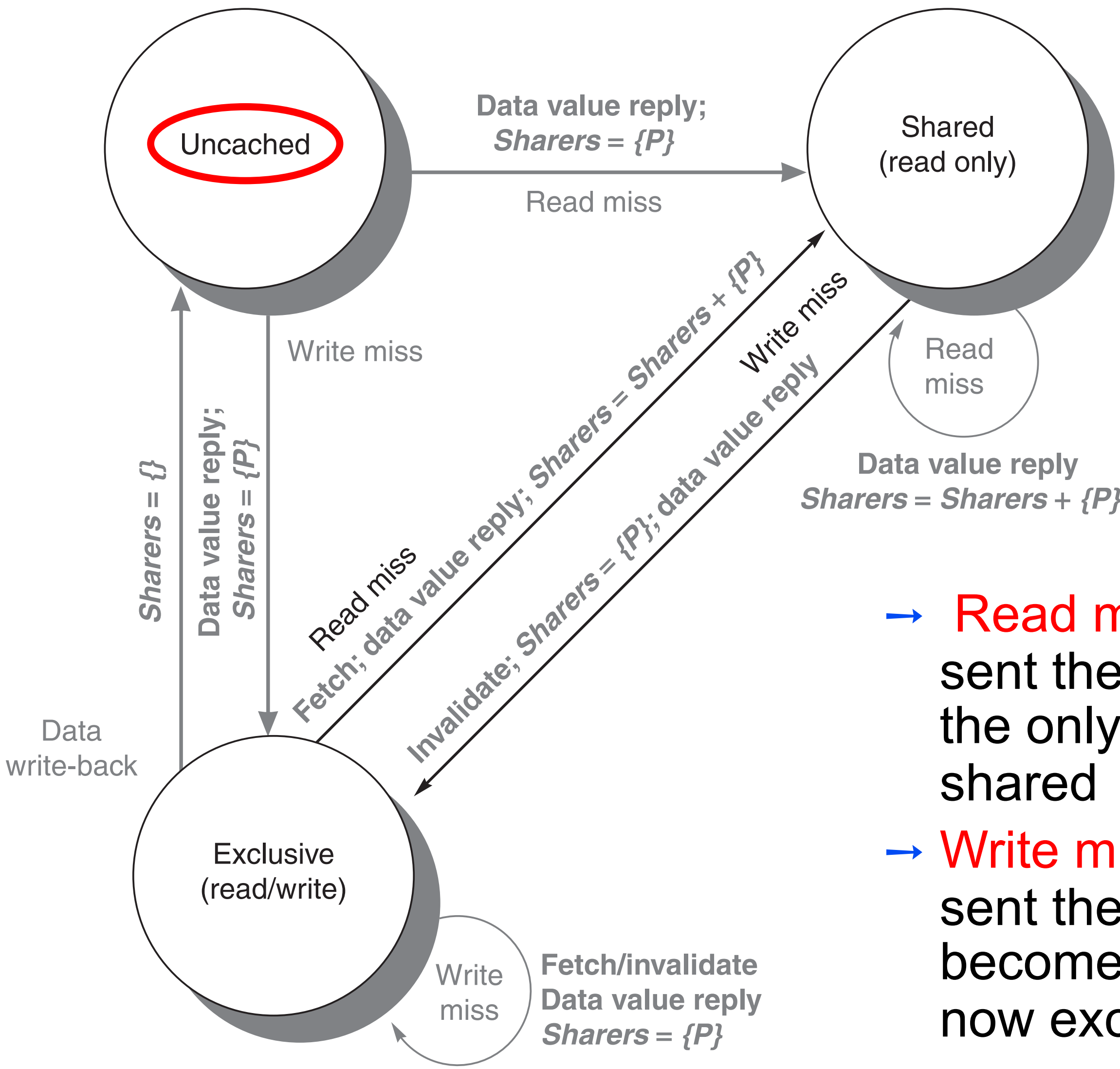
Directory Protocols



Cache controller

Directory Protocols

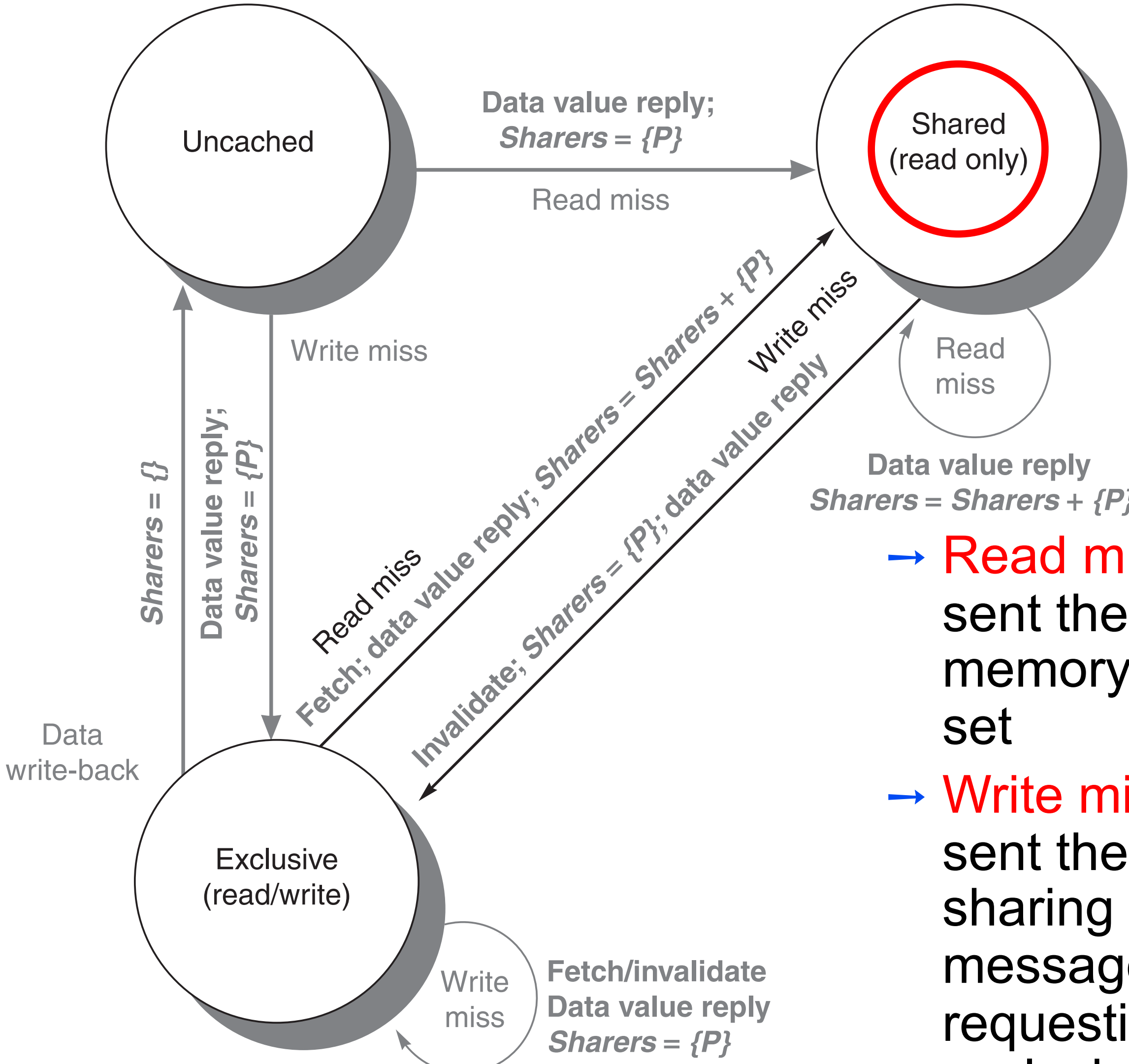
Directory controller



- **Read miss** – Requesting node is sent the requested data and is made the only sharing node, block is now shared
- **Write miss** – The requesting node is sent the requested data and becomes the sharing node, block is now exclusive

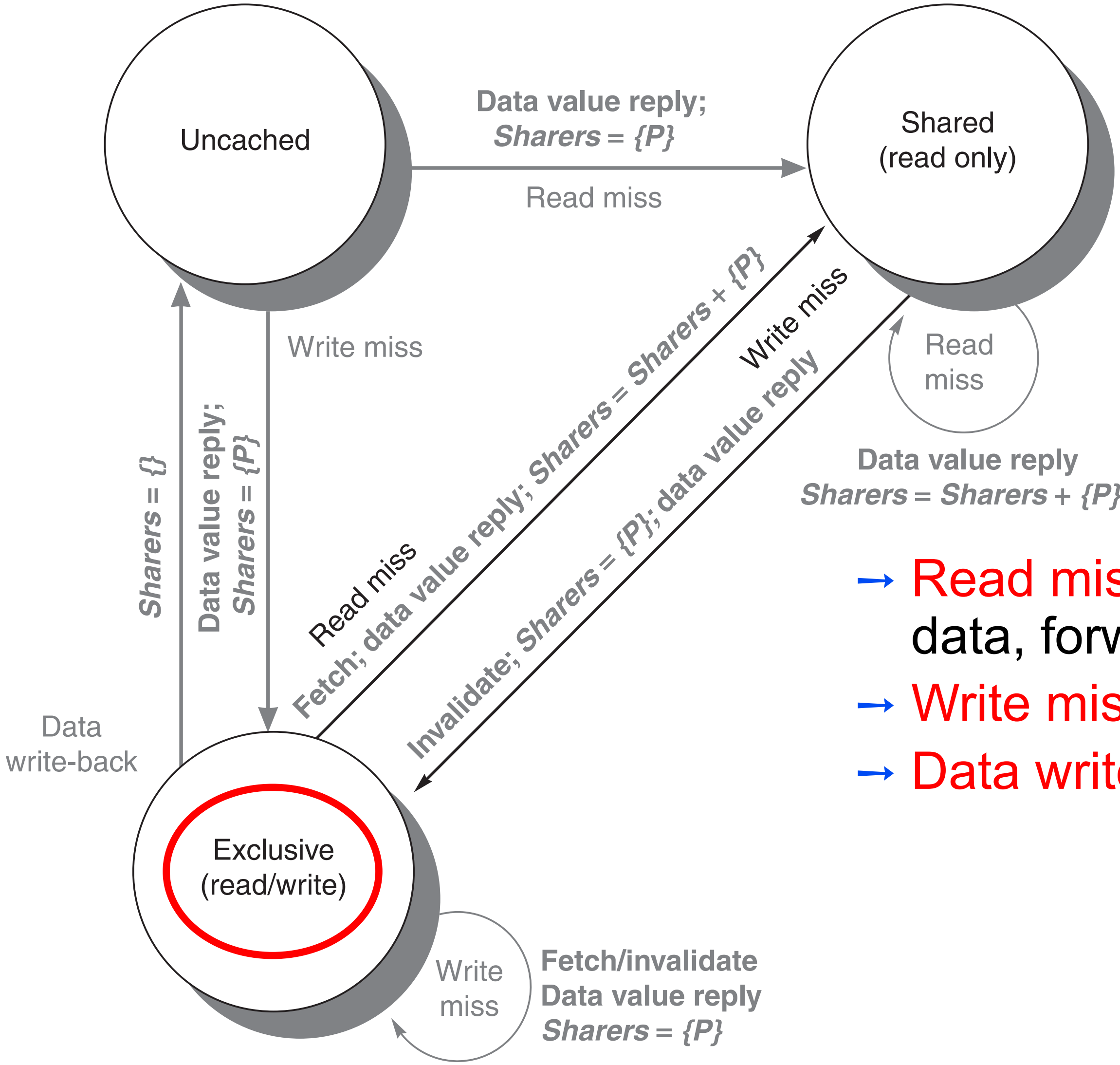
Directory Protocols

Directory controller



- Read miss – The requesting node is sent the requested data from memory, node is added to sharing set
- Write miss – The requesting node is sent the value, all nodes in the sharing set are sent invalidate messages, sharing set only contains requesting node, block is now exclusive

Directory Protocols   Directory controller



- Read miss – ask owner to send data, forward to the requesting node
- Write miss – invalidate owner's copy
- Data write back

Directory Protocols

- For uncached (Invalid) block:
  - Read miss
    - Requesting node is sent the requested data and is made the only sharing node, block is now shared
  - Write miss
    - The requesting node is sent the requested data and becomes the sharing node, block is now exclusive
- For shared block:
  - Read miss
    - The requesting node is sent the requested data from memory, node is added to sharing set
  - Write miss
    - The requesting node is sent the value, all nodes in the sharing set are sent invalidate messages, sharing set only contains requesting node, block is now exclusive

Directory Protocols

- For exclusive block:
  - Read miss
    - The owner is sent a data fetch message, block becomes shared, owner sends data to the directory, data written back to memory, sharers set contains old owner and requestor
  - Data write back
    - Block becomes uncached, sharer set is empty
  - Write miss
    - Message is sent to old owner to invalidate and send the value to the directory, requestor becomes new owner, block remains exclusive

Synchronization

- Basic building blocks:
  - Atomic exchange
    - Swaps register with memory location
  - Test-and-set
    - Sets under condition
  - Fetch-and-increment
    - Reads original value from memory and increments it in memory
  - Requires memory read and write in uninterruptable instruction

  - load linked/store conditional
    - If the contents of the memory location specified by the load linked are changed before the store conditional to the same address, the store conditional fails

Implementing Locks

- Spin lock
  - If no coherence:

```
        DADDUI  R2,R0,#1
lockit: EXCH    R2,0(R1)    ;atomic exchange
        BNEZ    R2,lockit   ;already locked?
```

  - If coherence:

```
lockit: LD      R2,0(R1)    ;load of lock
        BNEZ    R2,lockit   ;not available-spin
        DADDUI  R2,R0,#1    ;load locked value
        EXCH    R2,0(R1)    ;swap
        BNEZ    R2,lockit   ;branch if lock wasn’t 0
```

Implementing Locks

- Advantage of this scheme: reduces memory traffic

Step	P0	P1	P2	Coherence state of lock at end of step	Bus/directory activity
1	Has lock	Begins spin, testing if lock = 0	Begins spin, testing if lock = 0	Shared	Cache misses for P1 and P2 satisfied in either order. Lock state becomes shared.
2	Set lock to 0	(Invalidate received)	(Invalidate received)	Exclusive (P0)	Write invalidate of lock variable from P0.
3		Cache miss	Cache miss	Shared	Bus/directory services P2 cache miss; write-back from P0; state shared.
4		(Waits while bus/ directory busy)	Lock = 0 test succeeds	Shared	Cache miss for P2 satisfied
5		Lock = 0	Executes swap, gets cache miss	Shared	Cache miss for P1 satisfied
6		Executes swap, gets cache miss	Completes swap: returns 0 and sets lock = 1	Exclusive (P2)	Bus/directory services P2 cache miss; generates invalidate; lock is exclusive.
7		Swap completes and returns 1, and sets lock = 1	Enter critical section	Exclusive (P1)	Bus/directory services P1 cache miss; sends invalidate and generates write-back from P2.
8		Spins, testing if lock = 0			None

Models of Memory Consistency

Processor 1:	Processor 2:
A=0	B=0
…	…
A=1	B=1
if (B==0) …	if (A==0) …

- Should be impossible for both if-statements to be evaluated as true
  - Delayed write invalidate?
- Sequential consistency:
  - Result of execution should be the same as long as:
    - Accesses on each processor were kept in order
    - Accesses on different processors were arbitrarily interleaved

Relaxed Consistency Models

- Rules:
  - X → Y
    - Operation X must complete before operation Y is done
  - Sequential consistency requires:
    - R → W, R → R, W → R, W → W
  - Relax W → R
    - "Total store ordering"
  - Relax W → W
    - "Partial store order"
  - Relax R → W and R → R
    - "Weak ordering" and "release consistency"

Relaxed Consistency Models

- Consistency model is multiprocessor specific
- Programmers will often implement explicit synchronization
- Speculation gives much of the performance advantage of relaxed models with sequential consistency
  - Basic idea: if an invalidation arrives for a result that has not been committed, use speculation recovery

Implementing Locks

- To implement, delay completion of all memory accesses until all invalidations caused by the access are completed
  - Reduces performance!

- Alternatives:
  - Program-enforced synchronization to force write on processor to occur before read on the other processor
    - Requires synchronization object for A and another for B
      - “Unlock” after write
      - “Lock” after read

Course Evaluation for TA

- **Course Name:** EEL 5764 Computer Architecture
- **PI:** Kshitij Raj

- **If you are not physically attending this class today (and for EDGE students), please fill out the form in Canvas and submit**
  - Email sandip@ece.ufl.edu
  - Slide under my office door (BEN 323)
  - Put in my mailbox (LAR 216, ask for the location of faculty mailboxes)