Computer Architecture
A Quantitative Approach, Fifth Edition

Cache Coherence

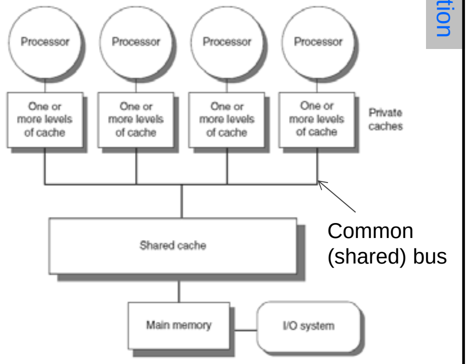
A very brief introduction

1

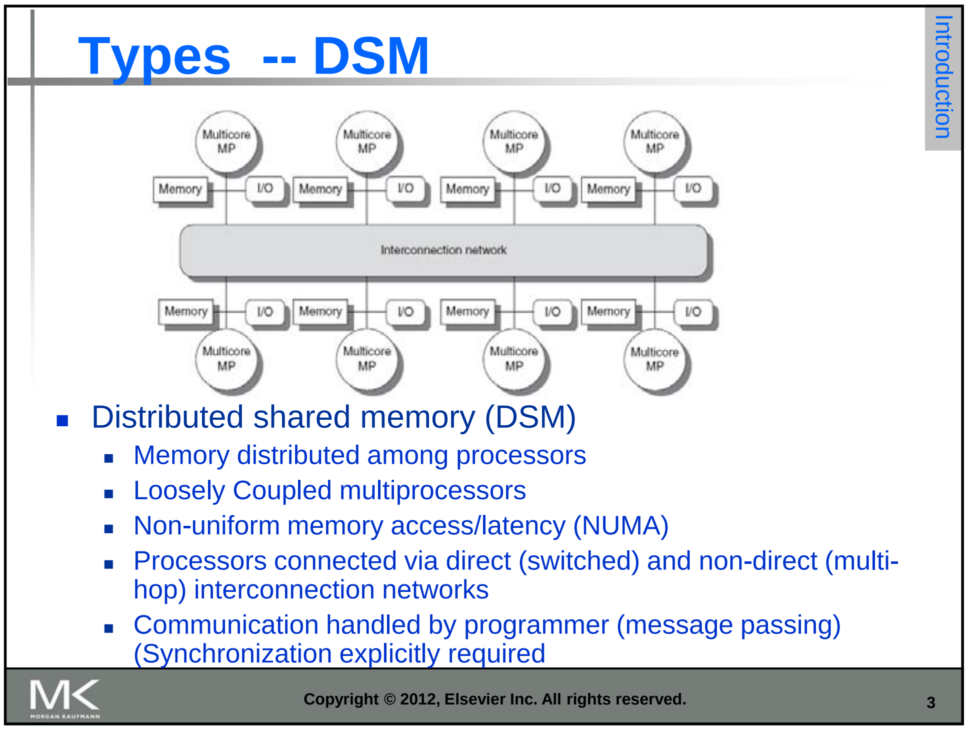
Types -- SMP

Introduction

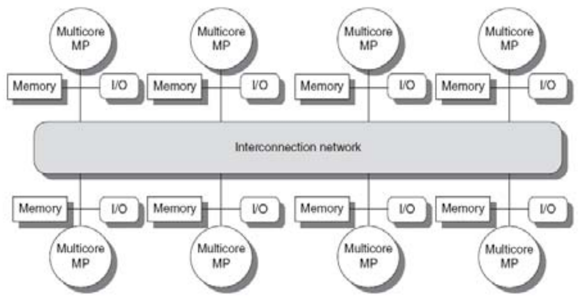
- Symmetric multiprocessors (SMP)
  - Small number of cores
  - AKA Tightly coupled multiprocessors
  - Share single memory with uniform memory latency
  - Bus is a bottleneck
  - Most of the communication is handled by OS/HW
  - Existing multi-core


Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Private caches
Common (shared) bus
Shared cache
Main memory
I/O system

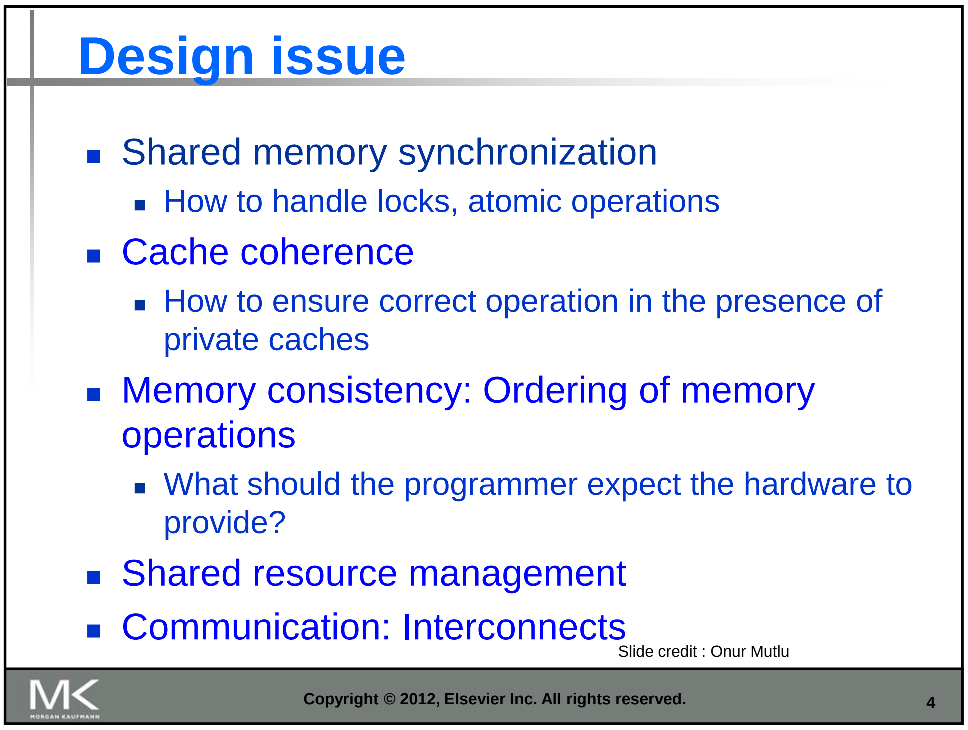
2



Types -- DSM



- Distributed shared memory (DSM)
  - Memory distributed among processors
  - Loosely Coupled multiprocessors
  - Non-uniform memory access/latency (NUMA)
  - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks
  - Communication handled by programmer (message passing) (Synchronization explicitly required



Design issue

- Shared memory synchronization
  - How to handle locks, atomic operations
- Cache coherence
  - How to ensure correct operation in the presence of private caches
- Memory consistency: Ordering of memory operations
  - What should the programmer expect the hardware to provide?
- Shared resource management
- Communication: Interconnects

Slide credit : Onur Mutlu

Programming Issues

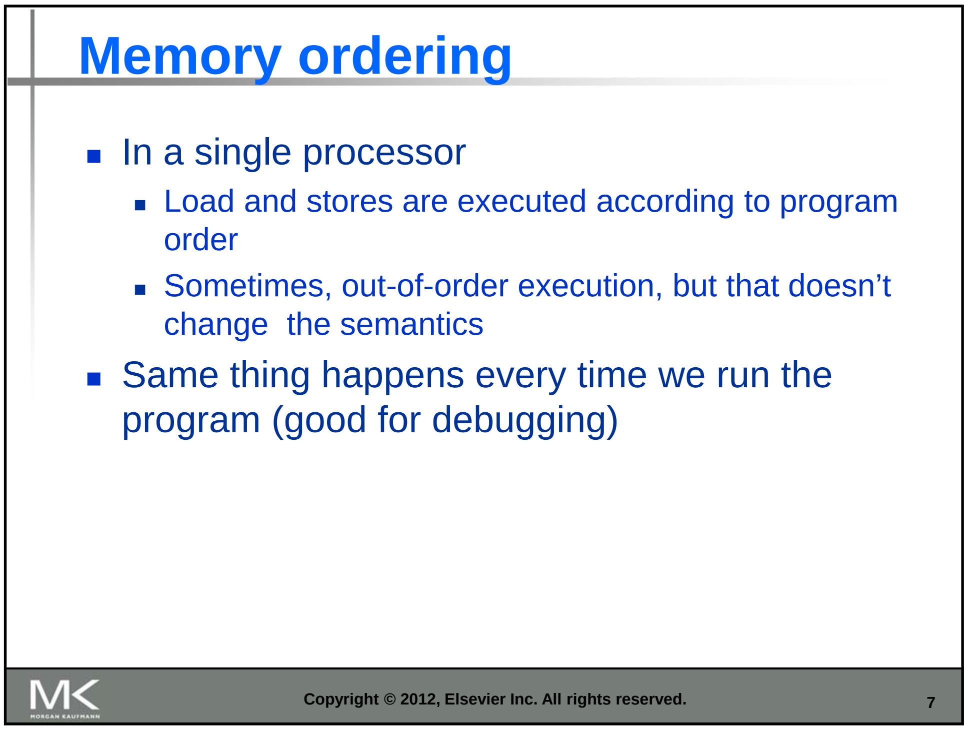
- Load imbalance
  - How to partition a single task into multiple tasks
- Synchronization
  - How to synchronize (efficiently) between tasks
  - How to communicate between tasks
  - Locks, barriers, pipeline stages, condition variables, semaphores, atomic operations, …
- Ensuring correct operation while optimizing for performance

Slide credit : Onur Mutlu

Cache Coherence

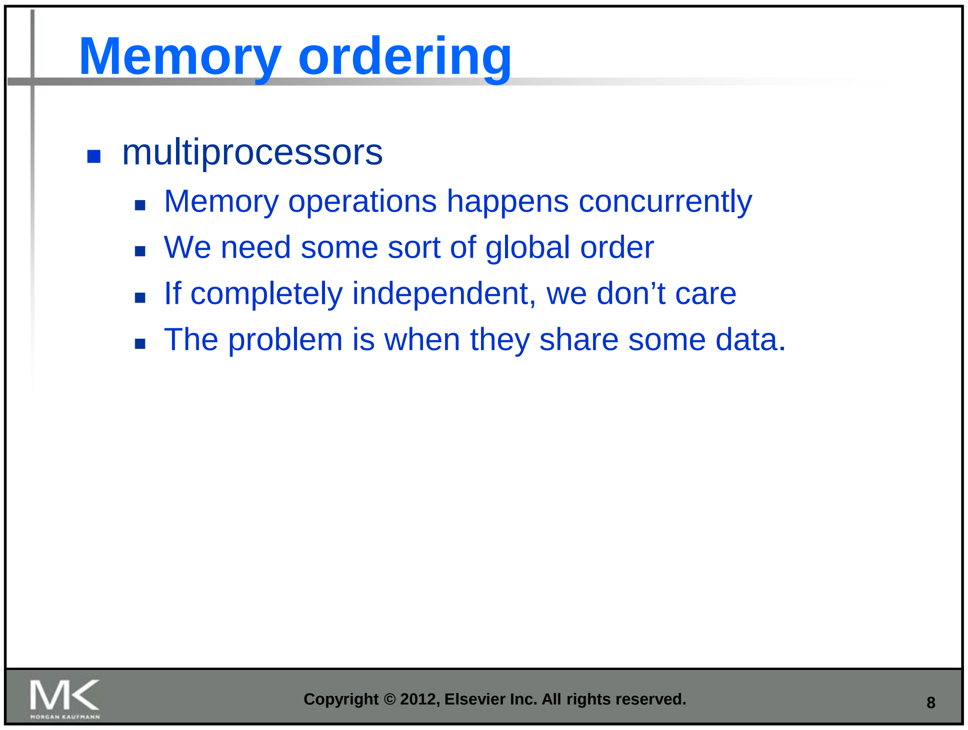
- Processors may see different values through their caches:

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



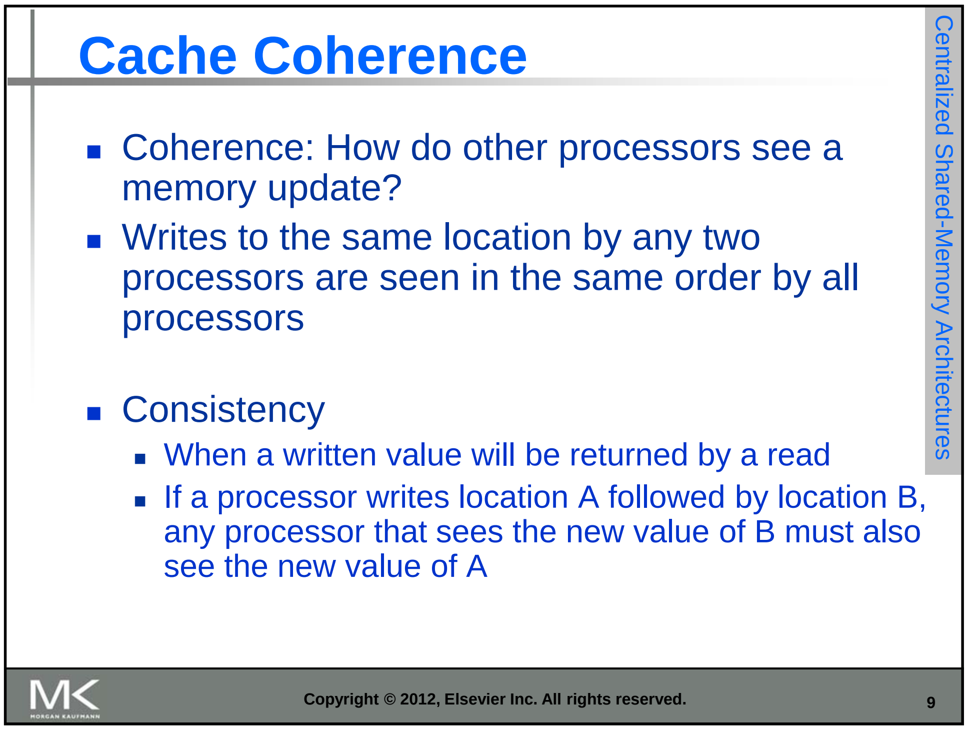
Memory ordering

- In a single processor
  - Load and stores are executed according to program order
  - Sometimes, out-of-order execution, but that doesn't change the semantics
- Same thing happens every time we run the program (good for debugging)



Memory ordering

- multiprocessors
  - Memory operations happens concurrently
  - We need some sort of global order
  - If completely independent, we don't care
  - The problem is when they share some data.

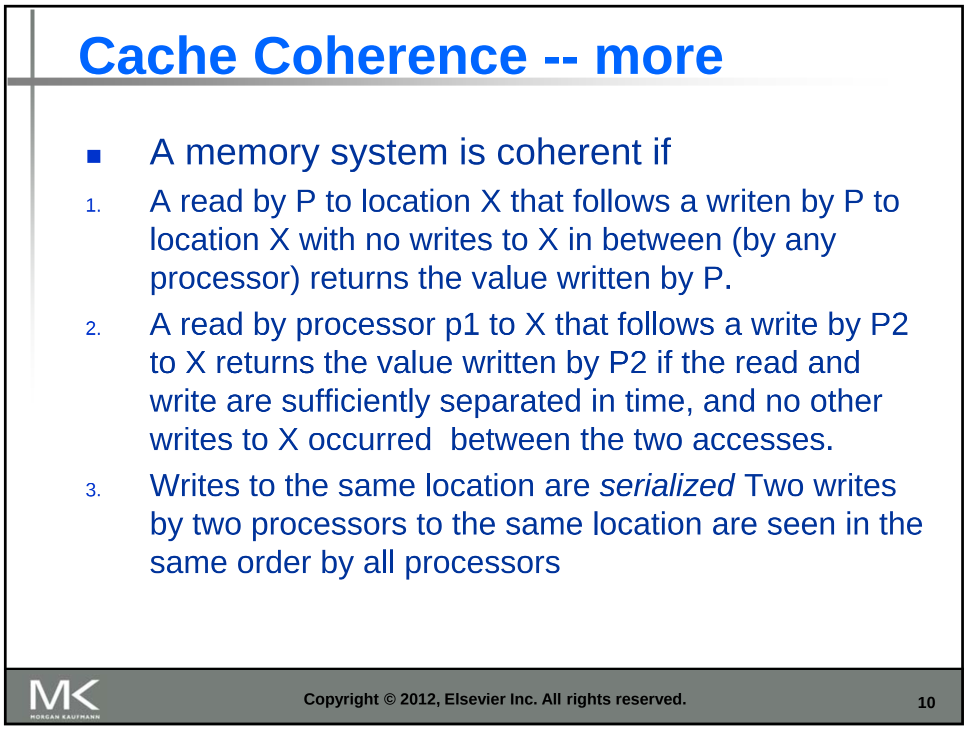


Cache Coherence

Centralized Shared-Memory Architectures

- Coherence: How do other processors see a memory update?
- Writes to the same location by any two processors are seen in the same order by all processors
- Consistency
  - When a written value will be returned by a read
  - If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

9

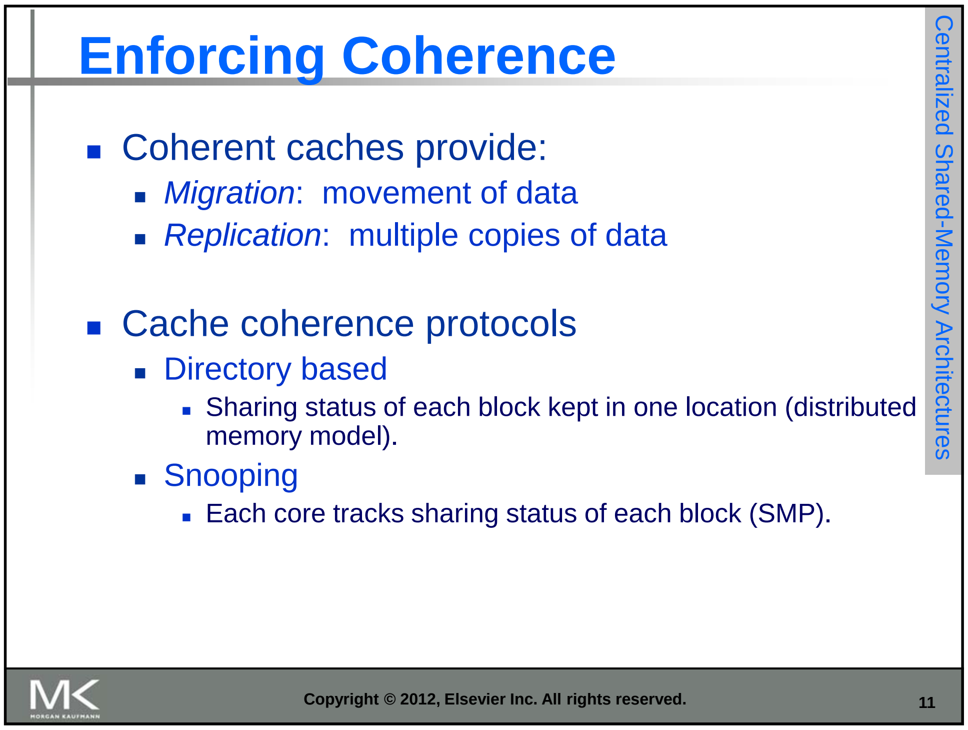


Cache Coherence -- more

- A memory system is coherent if

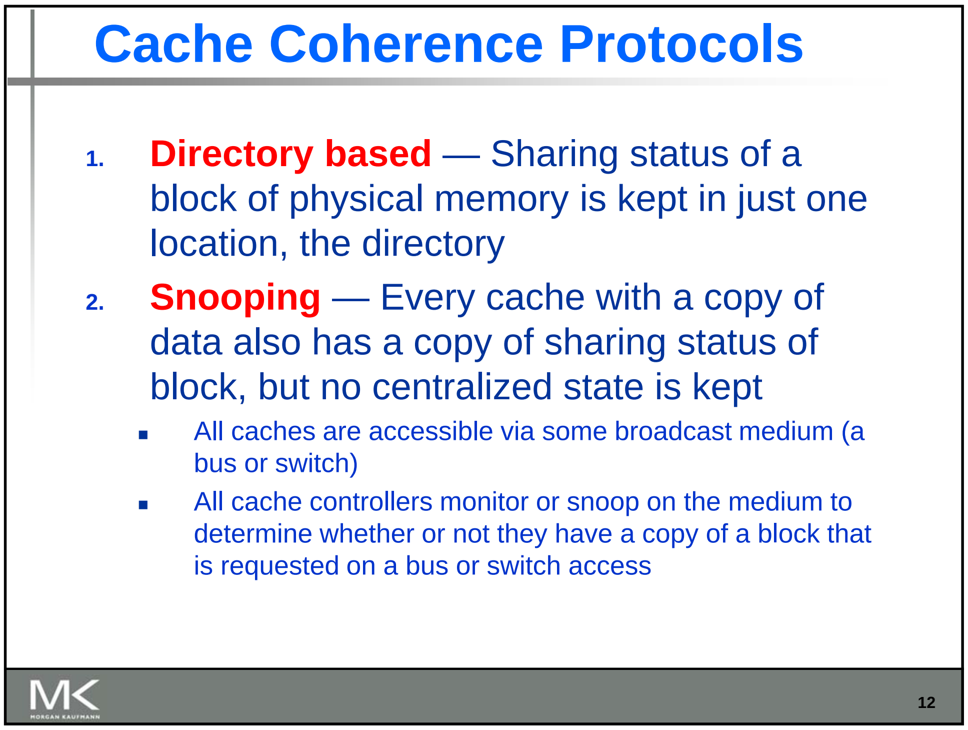
1. A read by P to location X that follows a writen by P to location X with no writes to X in between (by any processor) returns the value written by P.
2. A read by processor p1 to X that follows a write by P2 to X returns the value written by P2 if the read and write are sufficiently separated in time, and no other writes to X occurred between the two accesses.
3. Writes to the same location are *serialized* Two writes by two processors to the same location are seen in the same order by all processors

10



Enforcing Coherence

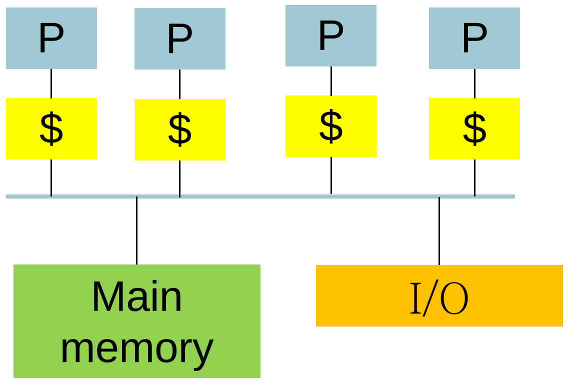
- Coherent caches provide:
  - *Migration*: movement of data
  - *Replication*: multiple copies of data
- Cache coherence protocols
  - Directory based
    - Sharing status of each block kept in one location (distributed memory model).
  - Snooping
    - Each core tracks sharing status of each block (SMP).



Cache Coherence Protocols

1. **Directory based** — Sharing status of a block of physical memory is kept in just one location, the directory
2. **Snooping** — Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept
   - All caches are accessible via some broadcast medium (a bus or switch)
   - All cache controllers monitor or snoop on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access

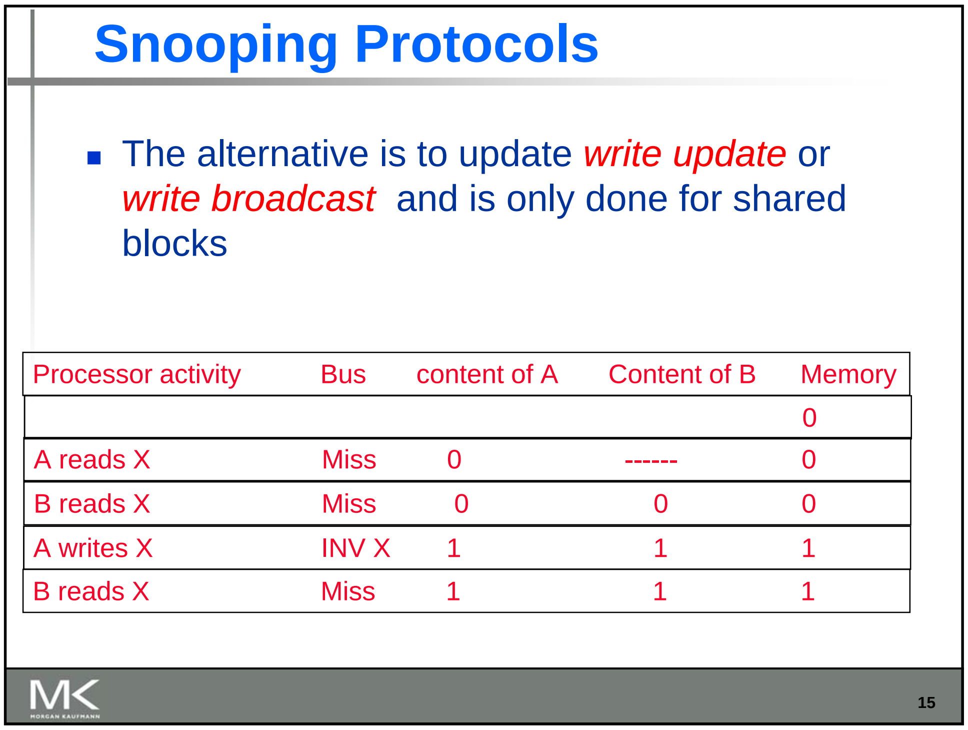
SMP or Centralized Shared Memory



Snooping Protocols

- The processor may have an exclusive access to the data, in this case the processor may change it. This is knows as *write invalidate*

Processor activity	Bus	content of A	Content of B	Memory
				0
A reads X	Miss	0	------	0
B reads X	Miss	0	0	0
A writes X	INV X	1	----	~~0~~
B reads X	Miss	1	1	1

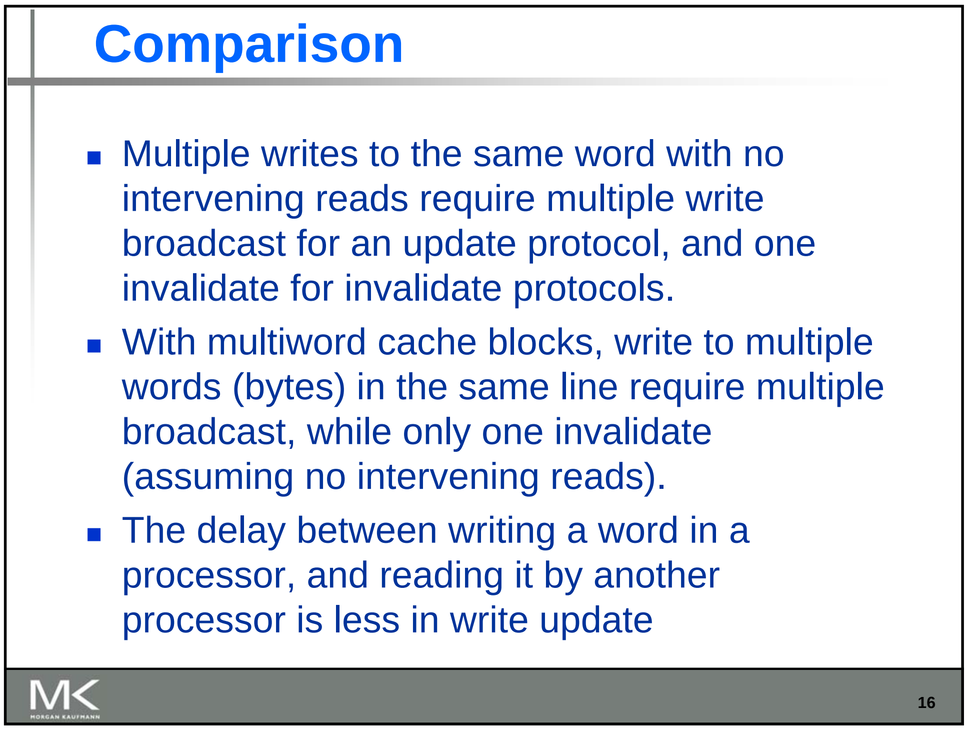


Snooping Protocols

- The alternative is to update *write update* or *write broadcast* and is only done for shared blocks

Processor activity	Bus	content of A	Content of B	Memory
				0
A reads X	Miss	0	------	0
B reads X	Miss	0	0	0
A writes X	INV X	1	1	1
B reads X	Miss	1	1	1

15

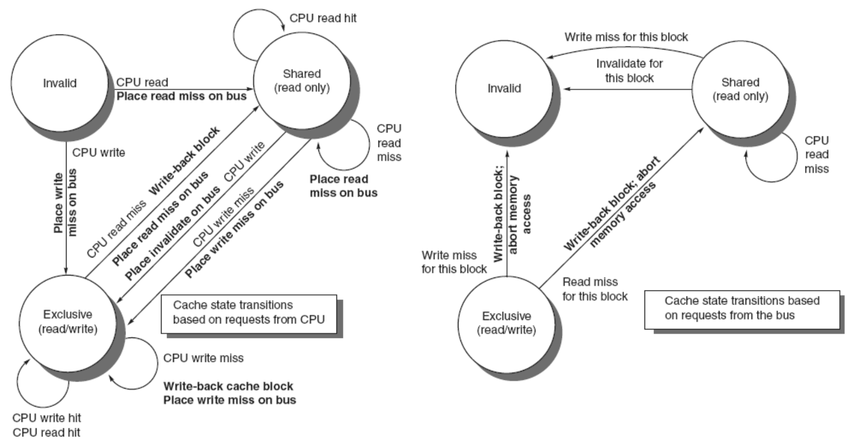


Comparison

- Multiple writes to the same word with no intervening reads require multiple write broadcast for an update protocol, and one invalidate for invalidate protocols.
- With multiword cache blocks, write to multiple words (bytes) in the same line require multiple broadcast, while only one invalidate (assuming no intervening reads).
- The delay between writing a word in a processor, and reading it by another processor is less in write update

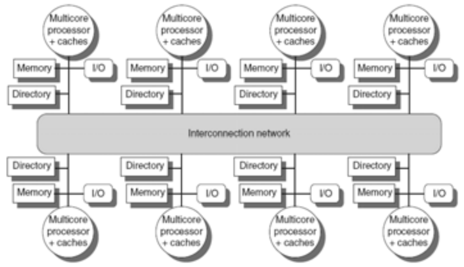
16

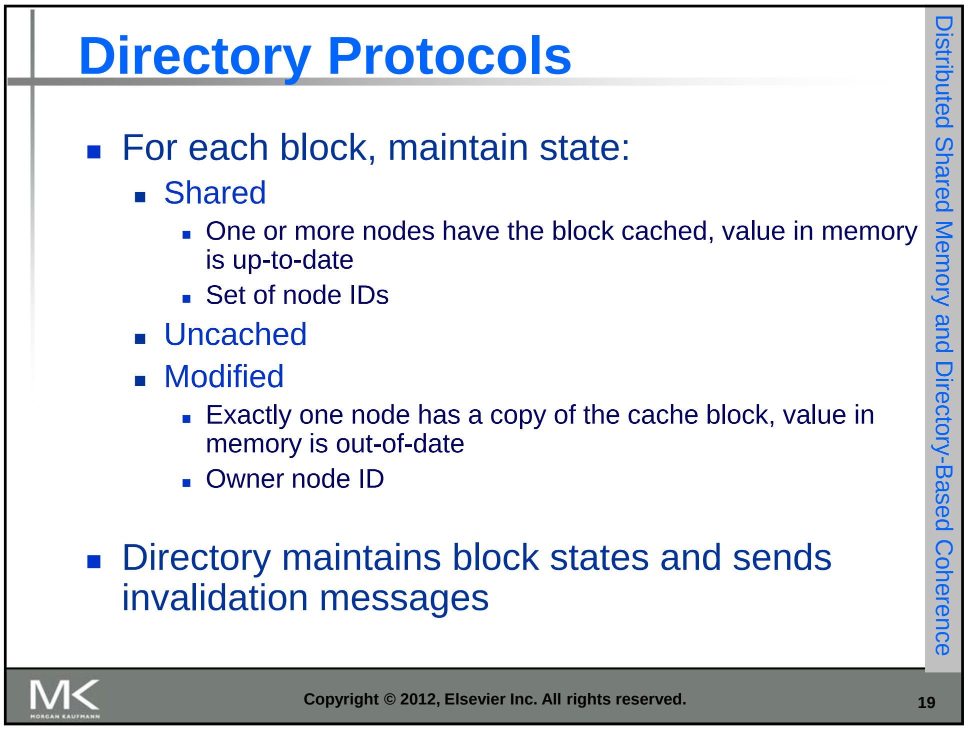
Snooping Coherence Protocols



Directory Protocols

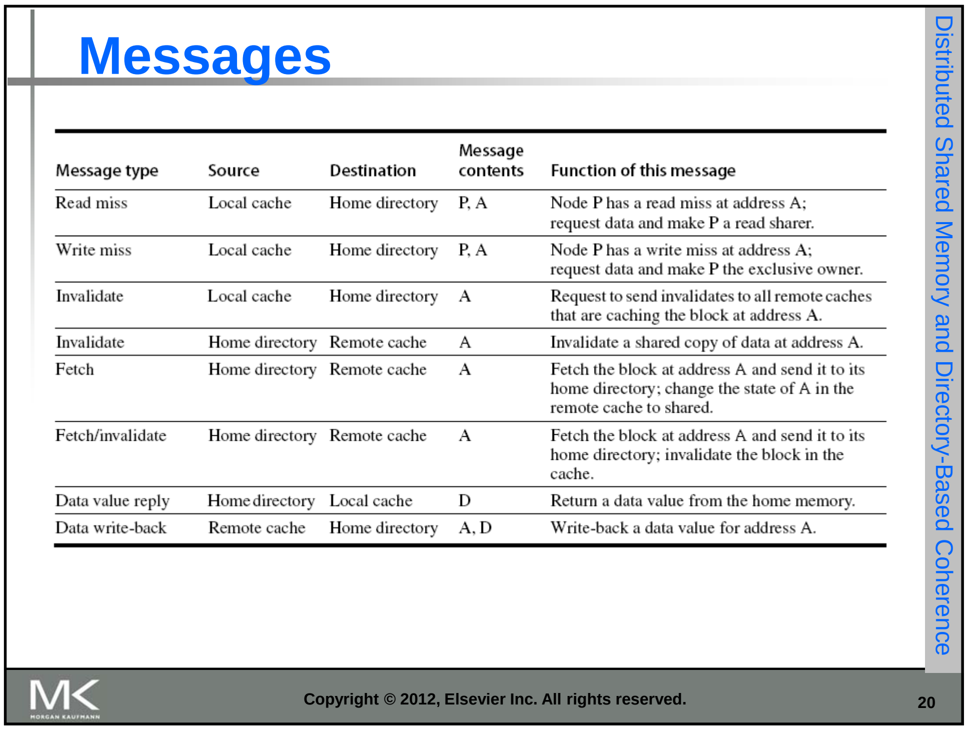
- Directory keeps track of every block
  - Which caches have each block
  - Dirty status of each block
- Implement in shared L3 cache
  - Keep bit vector of size = # cores for each block in L3
  - Not scalable beyond shared L3
- Implement in a distributed fashion:





Directory Protocols

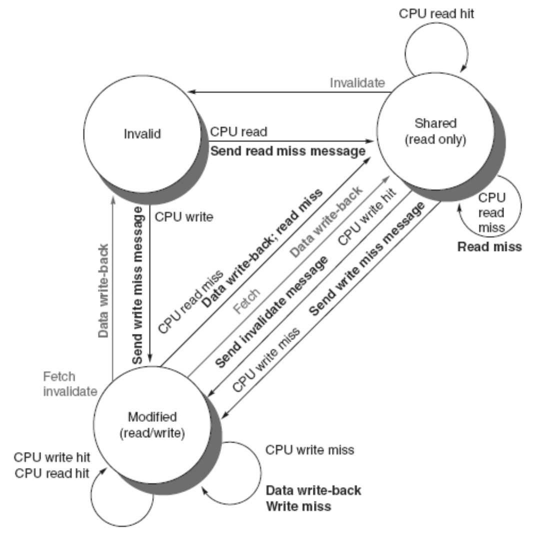
- For each block, maintain state:
  - Shared
    - One or more nodes have the block cached, value in memory is up-to-date
    - Set of node IDs
  - Uncached
  - Modified
    - Exactly one node has a copy of the cache block, value in memory is out-of-date
    - Owner node ID
- Directory maintains block states and sends invalidation messages



Messages

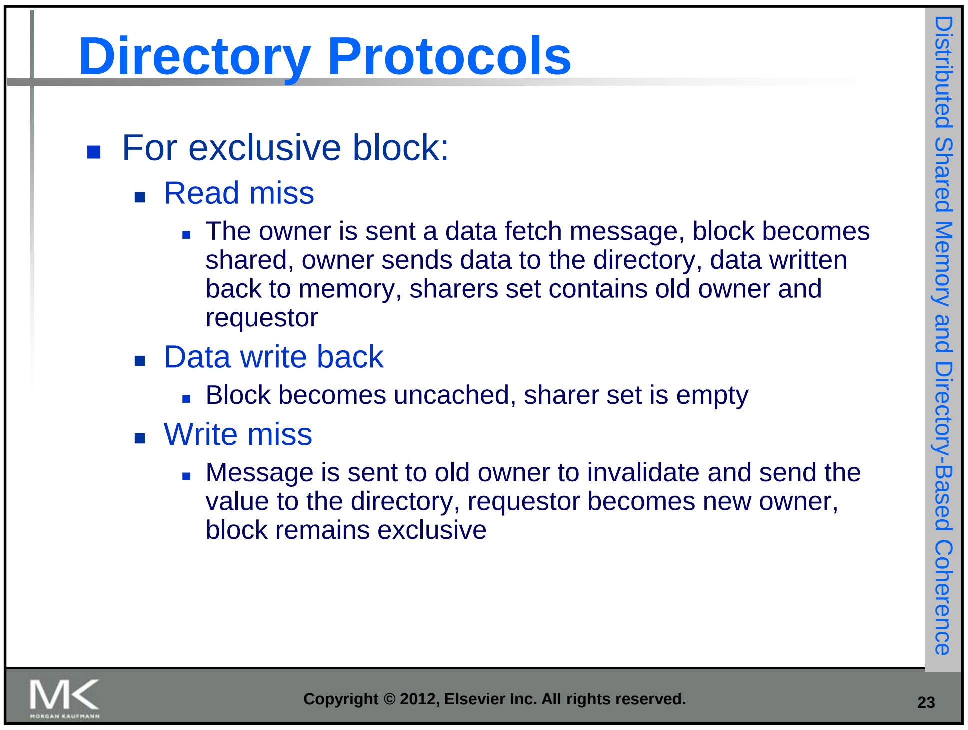
Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Node P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Node P has a write miss at address A; request data and make P the 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.

Directory Protocols



Directory Protocols

- For uncached 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

Distributed Shared Memory and Directory-Based Coherence