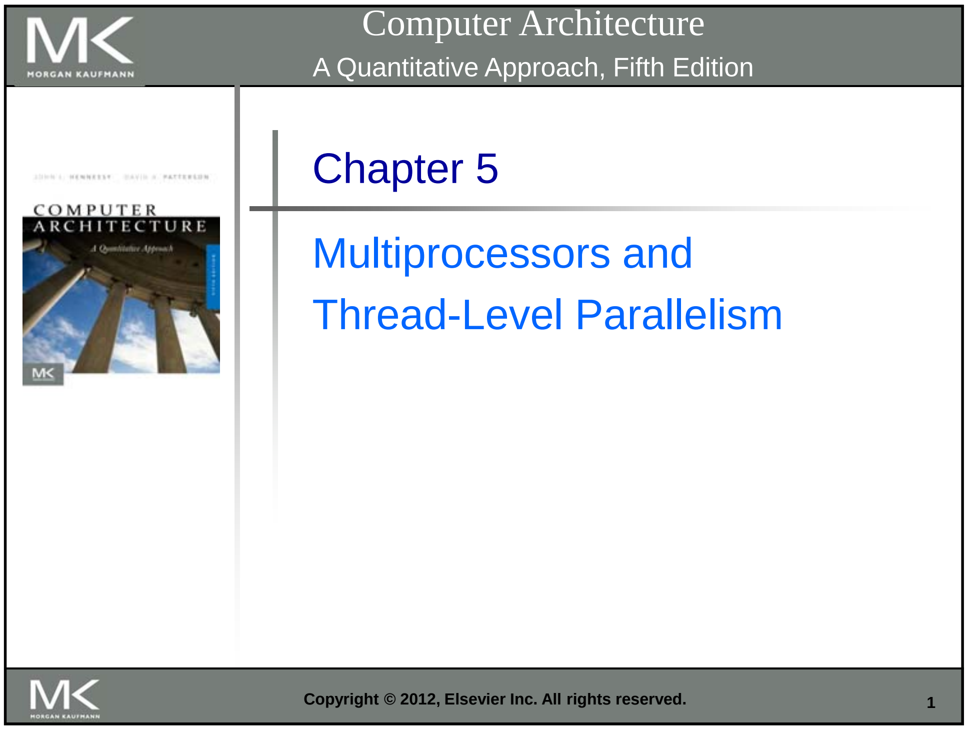


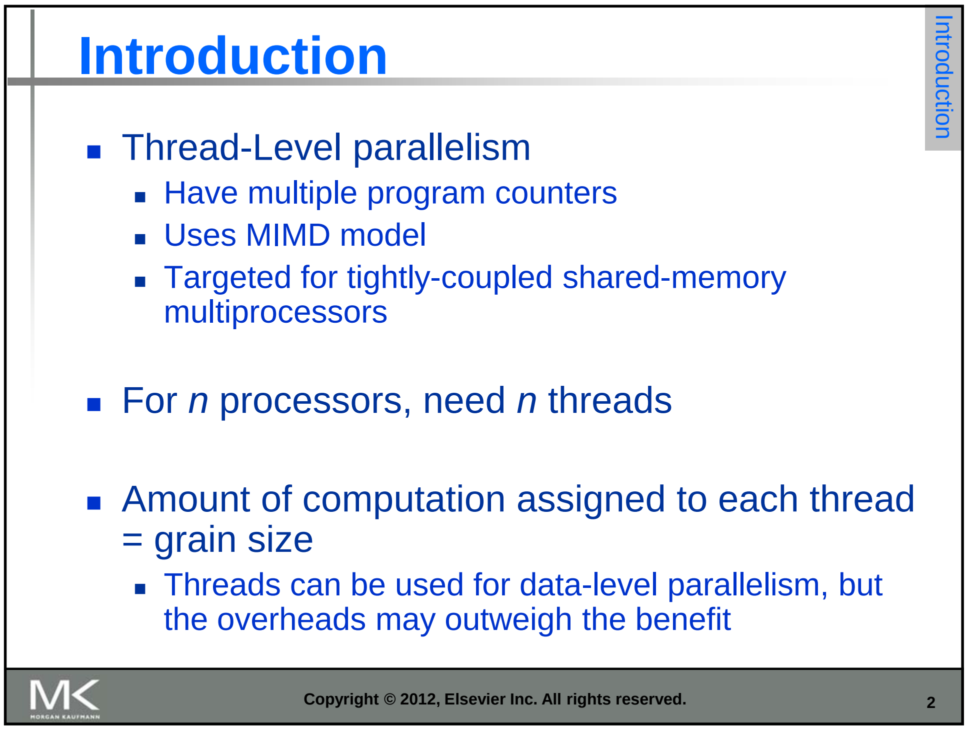
Computer Architecture

A Quantitative Approach, Fifth Edition

# Chapter 5

## Multiprocessors and Thread-Level Parallelism

1



## Introduction

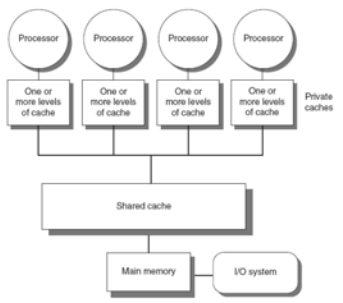
Introduction

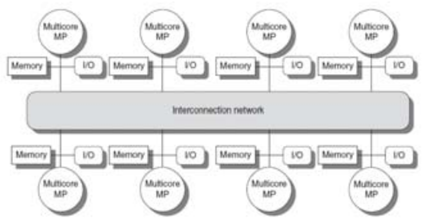
- Thread-Level parallelism
  - Have multiple program counters
  - Uses MIMD model
  - Targeted for tightly-coupled shared-memory multiprocessors
- For *n* processors, need *n* threads
- Amount of computation assigned to each thread = grain size
  - Threads can be used for data-level parallelism, but the overheads may outweigh the benefit

2

## Types

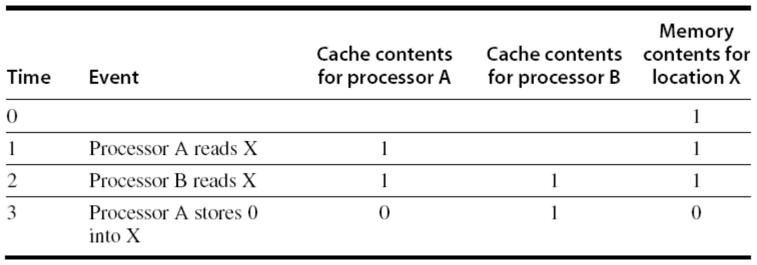
- Symmetric multiprocessors (SMP)
  - Small number of cores
  - Share single memory with uniform memory latency
- Distributed shared memory (DSM)
  - Memory distributed among processors
  - Non-uniform memory access/latency (NUMA)
  - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks


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
Shared cache
Main memory
I/O system

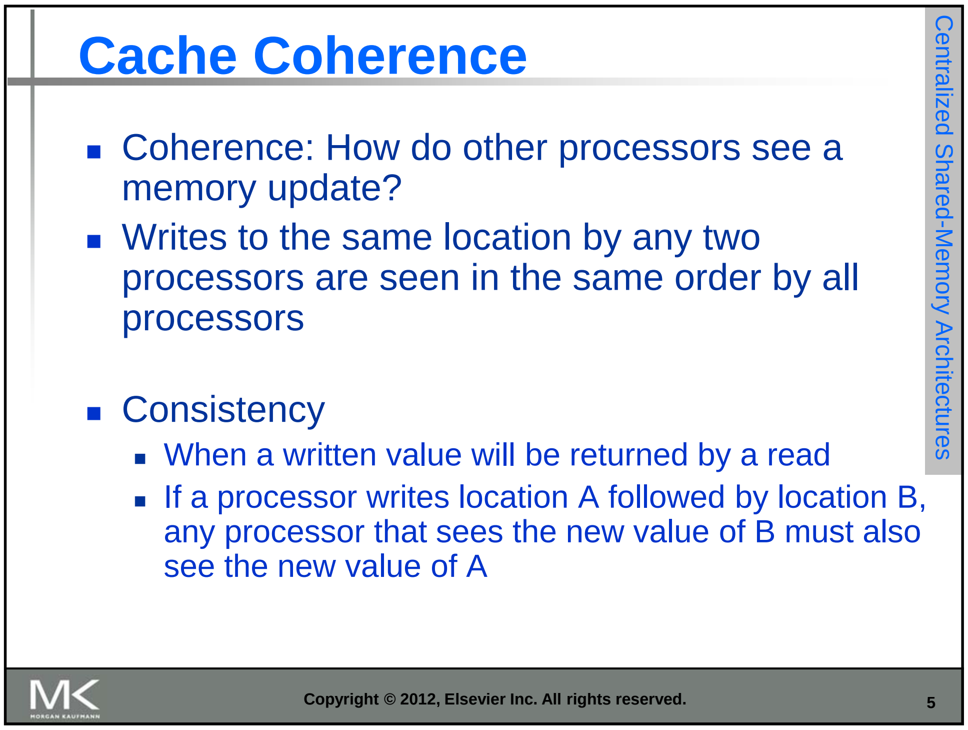


## Cache Coherence

- 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 |

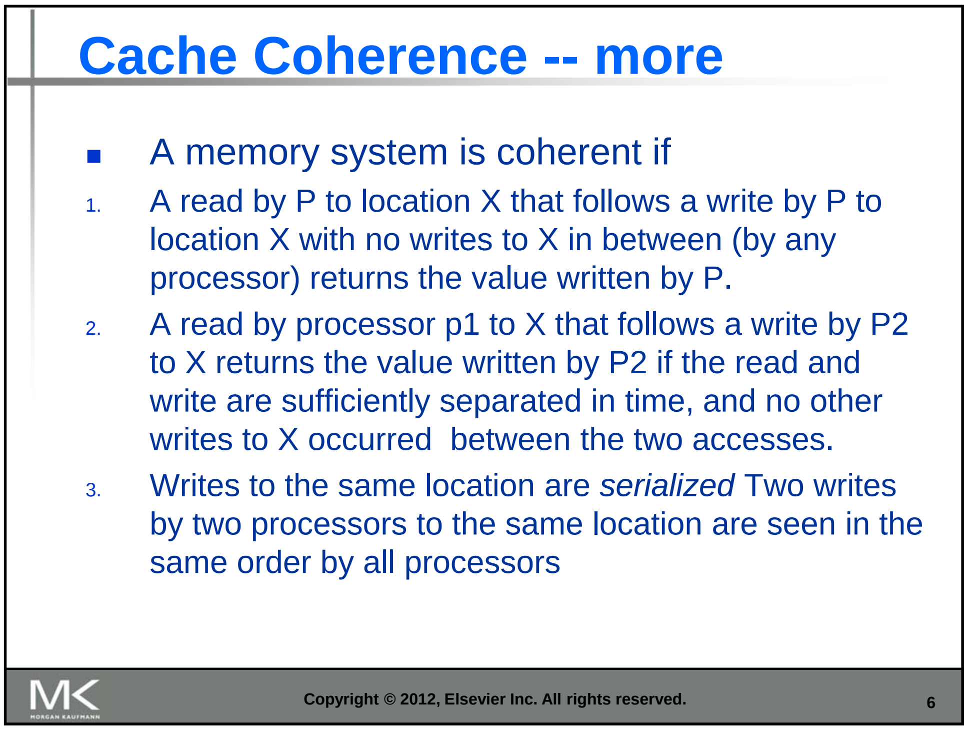


## Cache Coherence

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

5

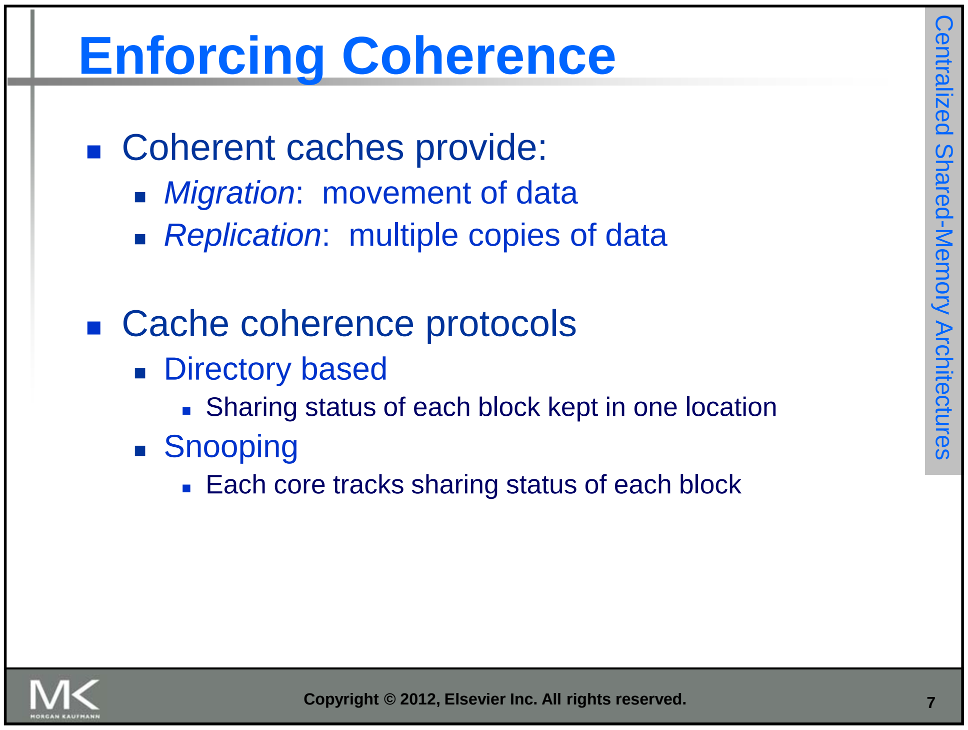


## Cache Coherence -- more

- A memory system is coherent if

1. A read by P to location X that follows a write 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

6

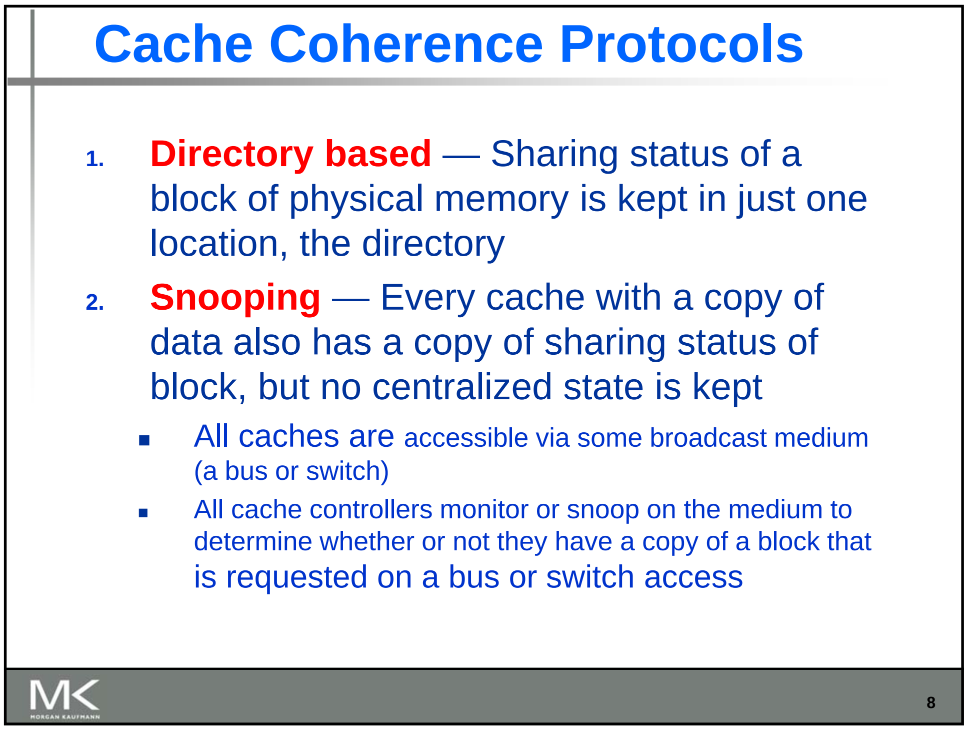


## Enforcing Coherence

Centralized Shared-Memory Architectures

- 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
  - Snooping
    - Each core tracks sharing status of each block

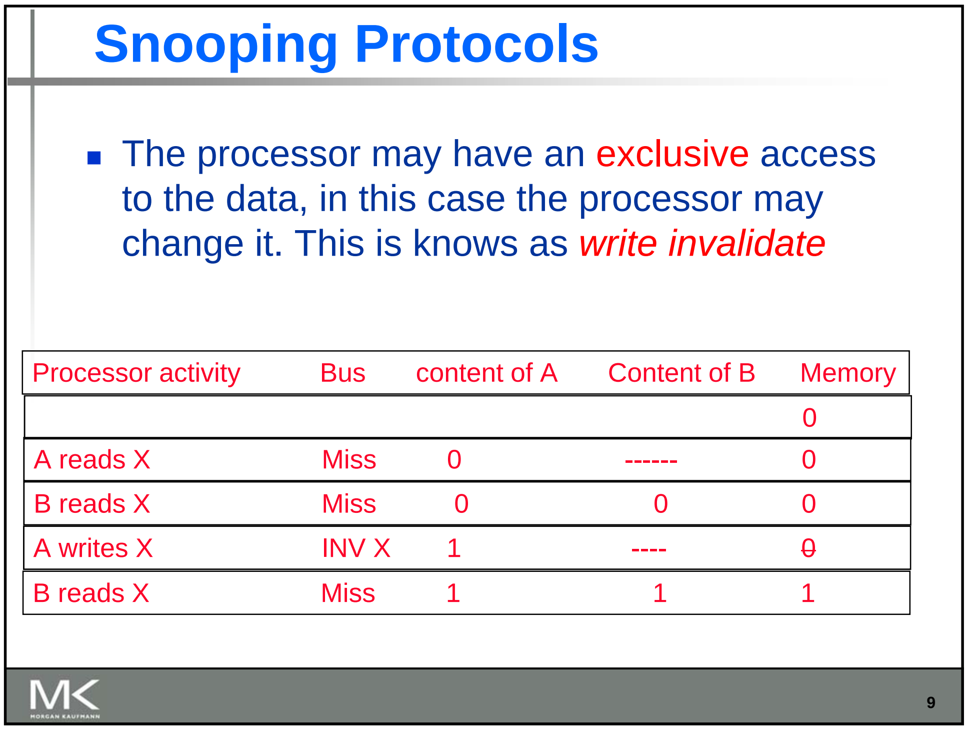
7



## 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

8

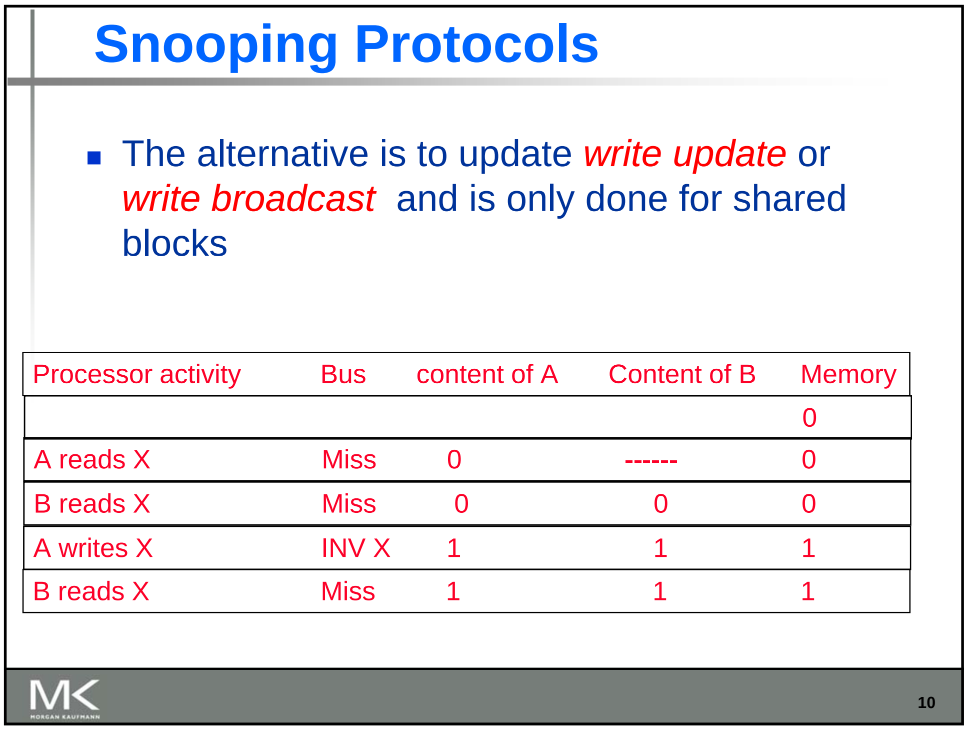


## Snooping Protocols

- 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 |

9

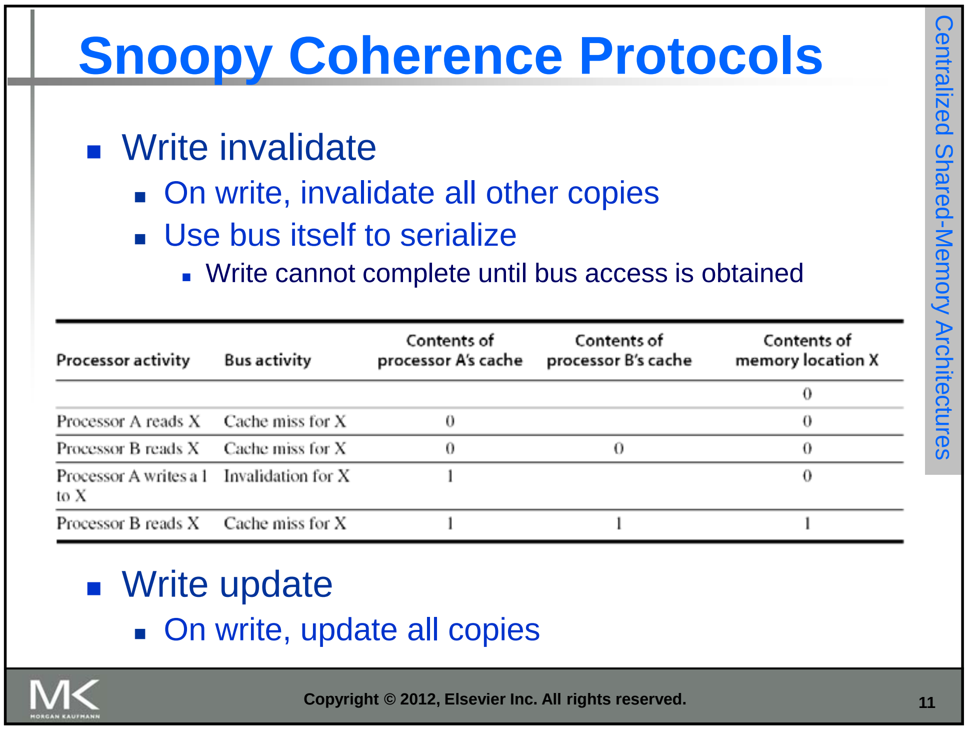


## Snooping Protocols

- 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 |

10

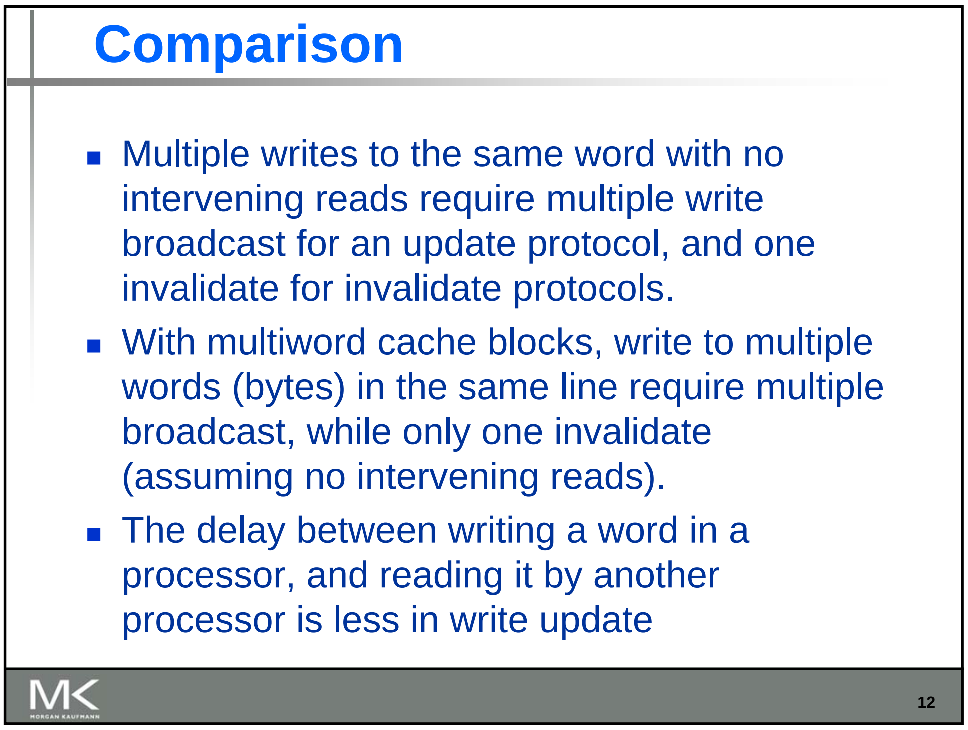


## Snoopy Coherence Protocols

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
    - Write cannot complete until bus access is obtained

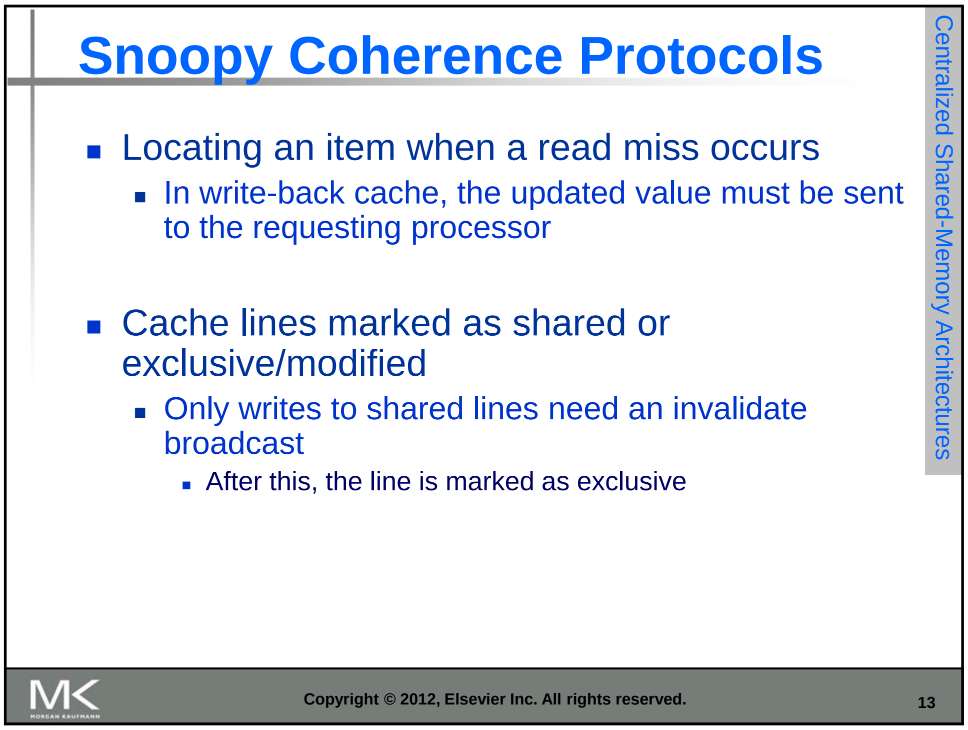
| Processor activity | Bus activity | Contents of processor A's cache | Contents of processor B's cache | Contents of memory location X |
|---|---|---|---|---|
| | | | | 0 |
| Processor A reads X | Cache miss for X | 0 | | 0 |
| Processor B reads X | Cache miss for X | 0 | 0 | 0 |
| Processor A writes a 1 to X | Invalidation for X | 1 | | 0 |
| Processor B reads X | Cache miss for X | 1 | 1 | 1 |

- Write update
  - On write, update all copies



## 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

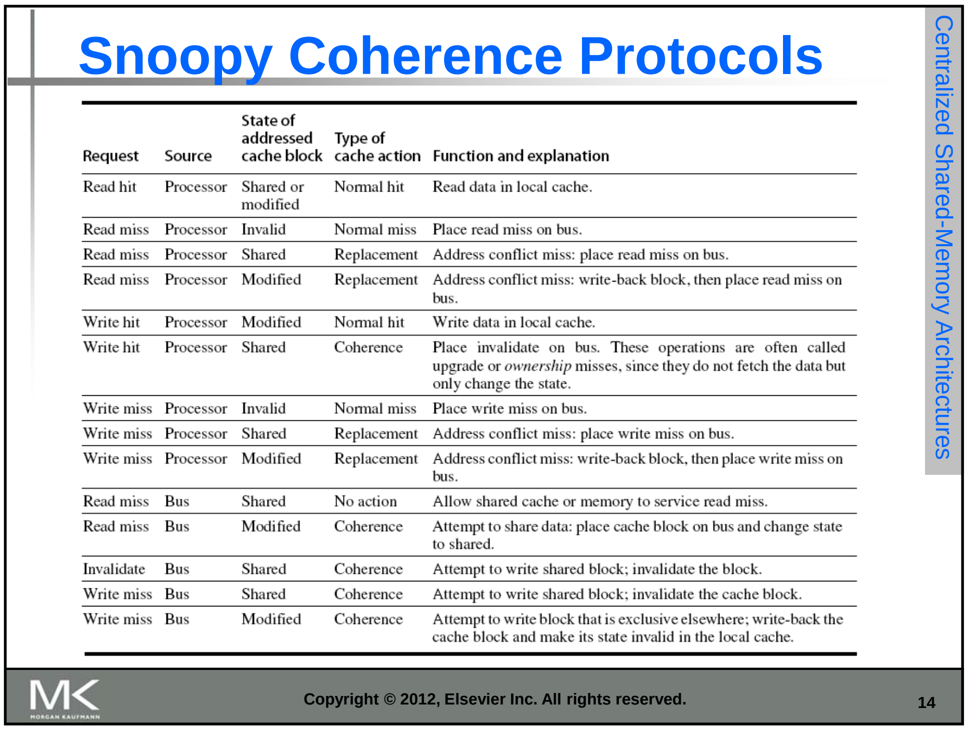


## Snoopy Coherence Protocols

- 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

Centralized Shared-Memory Architectures

13



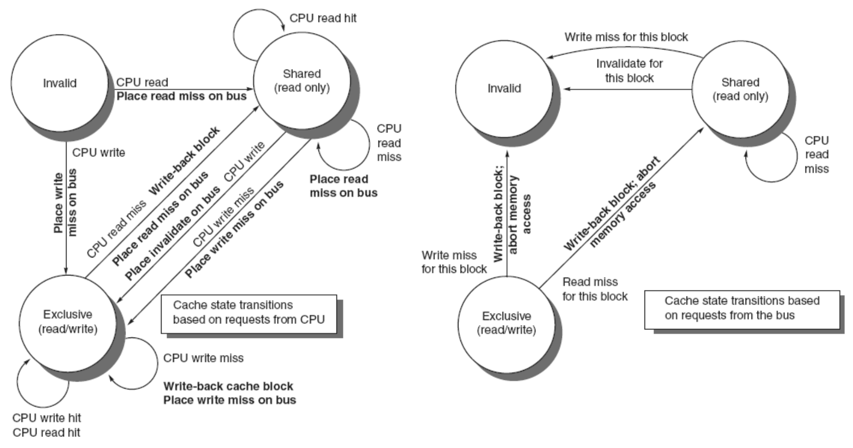
## Snoopy Coherence Protocols

| 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. |

Centralized Shared-Memory Architectures

14

## Snoopy Coherence Protocols



## 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:
  - Add exclusive state to indicate clean block in only one cache (MESI protocol)
    - Prevents needing to write invalidate on a write
  - Owned state