Memory Issues in Parallel Processing

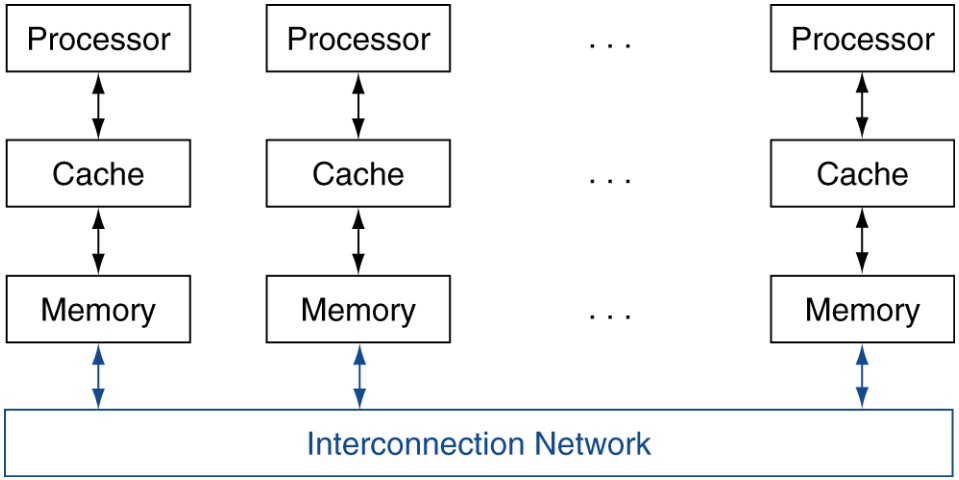
Lecture for CPSC 5155
Edward Bosworth, Ph.D.
Computer Science Department
Columbus State University

Sharing Data

- **Multiprocessors** are computing systems in which all programs share a single address space. This may be achieved by use of a single memory or a collection of memory modules that are closely connected and addressable as a single unit.
- **Multicomputers** are computing systems in which a collection of processors, each with its private memory, communicate via some dedicated network. Programs communicate by use of specific message and receive message primitives. Examples: Clusters and **MPP** (**M**assively **P**arallel **P**rocessors).

Message Passing

- Each processor has private physical address space
- Hardware sends/receives messages between processors



Loosely Coupled Clusters

- Network of independent computers
  - Each has private memory and OS
  - Connected using I/O system
    - E.g., Ethernet/switch, Internet
- Suitable for applications with independent tasks
  - Web servers, databases, simulations, …
- High availability, scalable, affordable
- Problems
  - Administration cost (prefer virtual machines)
  - Low interconnect bandwidth
    - c.f. processor/memory bandwidth on an SMP

Grid Computing

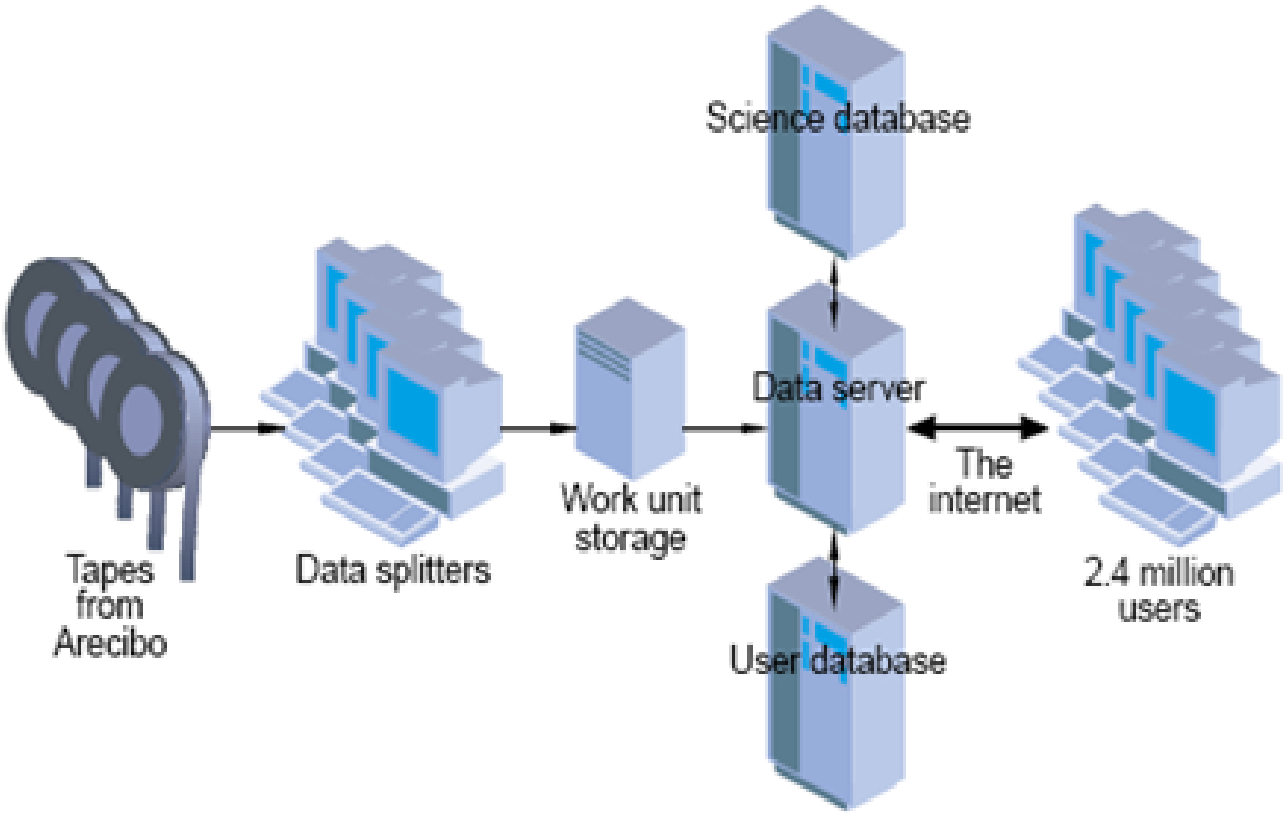
- Separate computers interconnected by long-haul networks
  - E.g., Internet connections
  - Work units farmed out, results sent back
- Can make use of idle time on PCs
  - E.g., SETI@home, World Community Grid

Radio SETI

- The SETI antennas regularly detect signals from a species that is reputed to be intelligent; unfortunately that is us on this planet. A great deal of computation is required to filter the noise and human–generated signals from the signals detected, possibly leaving signals from sources that might be truly extraterrestrial.
- Radio SETI was started under a modest grant and involved the use of dedicated radio antennas and supercomputers (the Cray–1 ?) located on the site.
- In 1995, David Gedye proposed a cheaper data–processing solution: create a virtual supercomputer composed of large numbers of computers connected by the global Internet. SETI@home was launched in May 1999.

A Radio Telescope

The Radio SETI Process



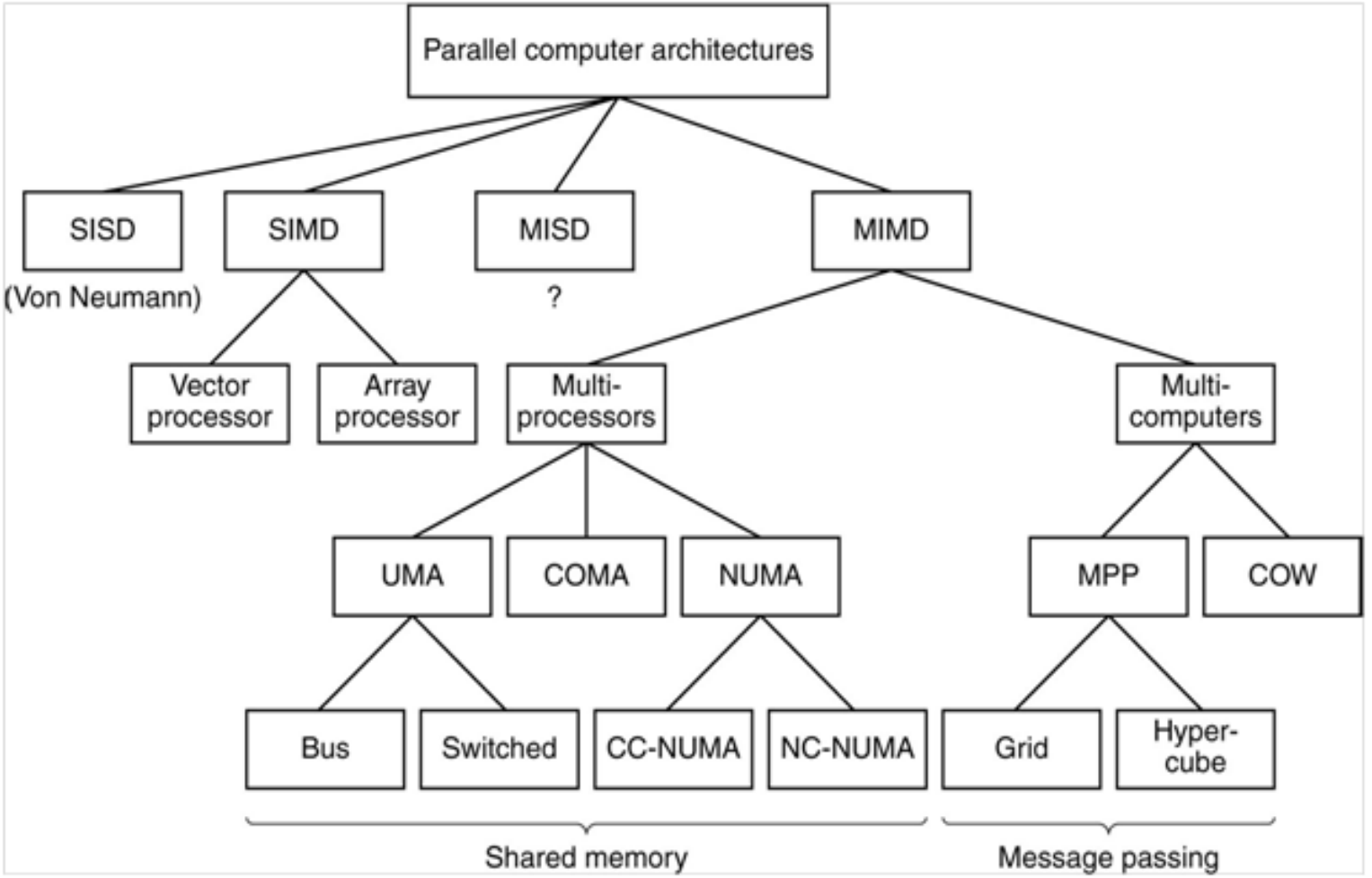
Multiprocessors

- There are two major variants of multiprocessors.
- In **UMA** (**U**niform **M**emory **A**ccess) multiprocessors, often called **SMP** (**S**ymmetric **M**ulti**p**rocessors), each processor takes the same amount of time to access every memory location.
- In **NUMA** (**N**on–**U**niform **M**emory **A**ccess) multiprocessors, some memory accesses are faster than others. This model presents interesting challenges to the programmer in that race conditions become a real possibility, but offers increased performance.

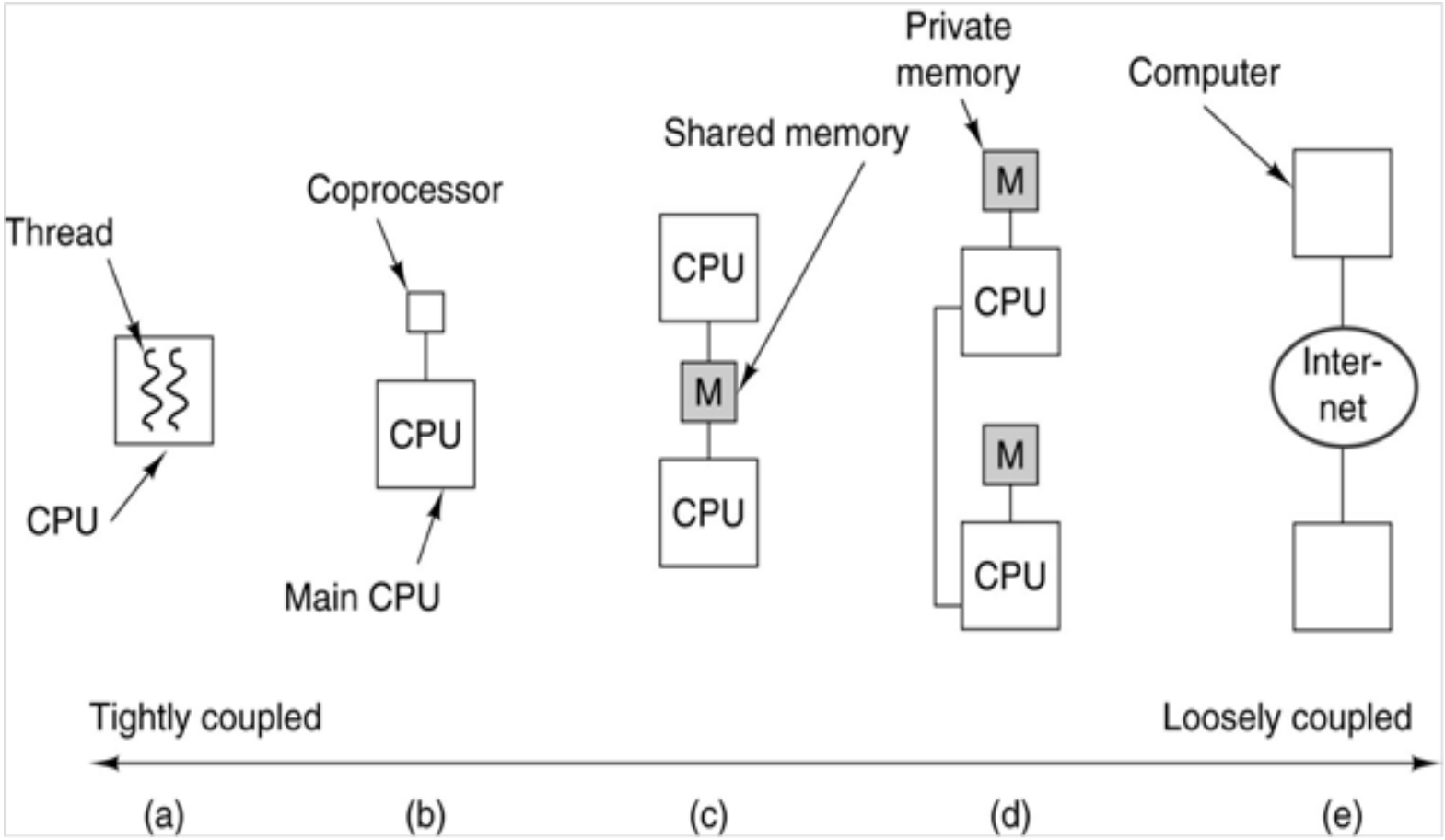
Coordination of Processes

- **Multiprocessors**
  One of the more common mechanisms for coordinating multiple processes in a single address space multiprocessor is called a **lock**. This feature is commonly used in databases accessed by multiple users, even those implemented on single processors.
- **Multicomputers**
  These must use explicit synchronization messages in order to coordinate the processes. One method is called **"barrier synchronization"**, in which there are logical spots, called "barriers" in each of the programs. When a process reaches a barrier, it stops processing and waits until it has received a message allowing it to proceed.

Parallel Processor Taxonomy

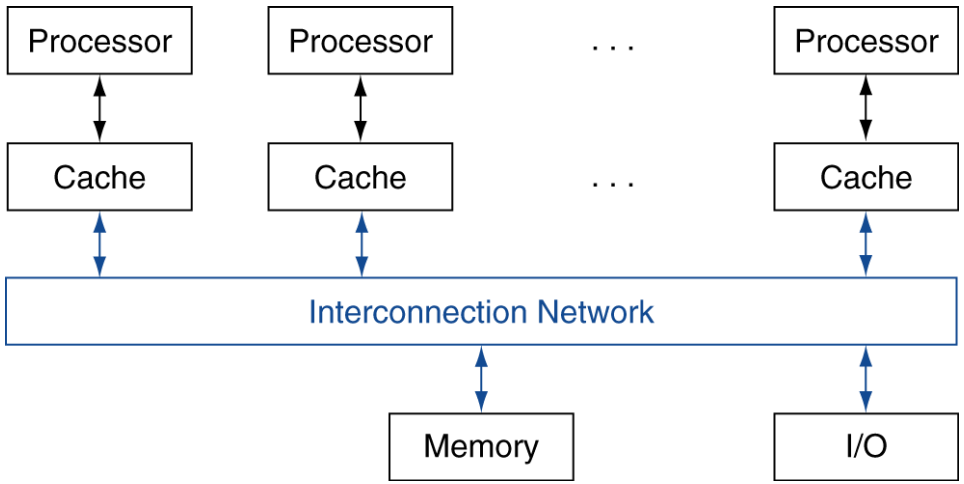


Levels of Parallelism



Shared Memory

- SMP: shared memory multiprocessor
  - Hardware provides single physical address space for all processors
  - Synchronize shared variables using locks
  - Memory access time
    - UMA (uniform) vs. NUMA (nonuniform)



What is the Problem?

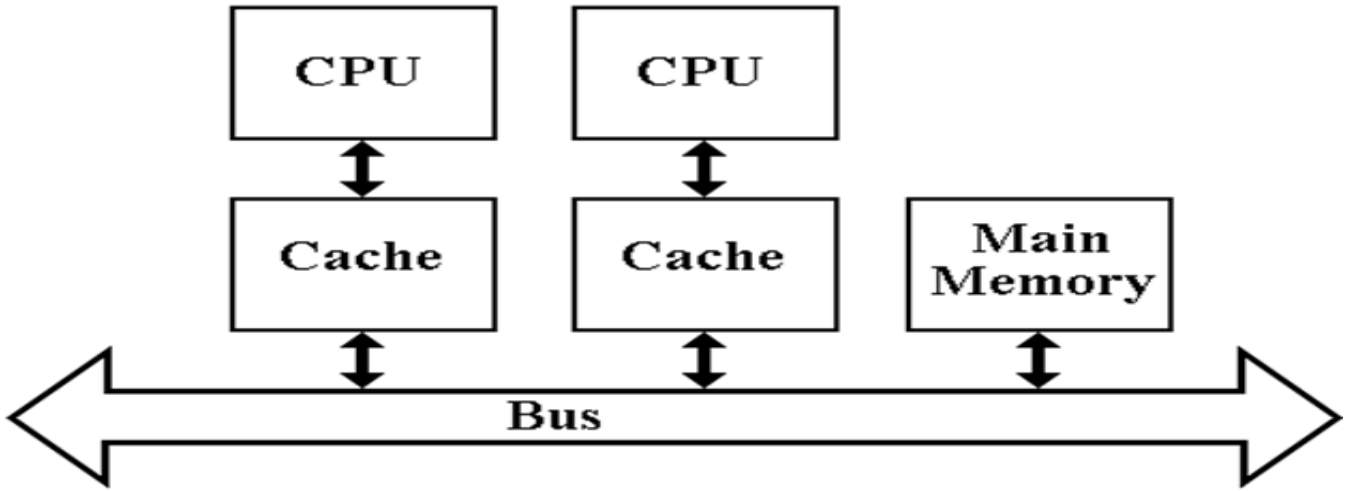
- From a simplicity viewpoint, it would be better to avoid cache memory.
- In this model, each CPU on the shared bus would access main memory directly.
- This is not feasible. The shared memory bus could not handle the data traffic.
- Also, there are speed advantages to each CPU having its own caches (L1 and L2).

The Cache Write Problem

- The problem in uniprocessors is quite simple. If the cache is updated, the main memory must be updated at some point so that the changes can be made permanent if needed.
- If the CPU writes to the cache and then reads from it, the data are available, even if they have not been written back to main memory.

The Cache Write Problem (Part 2)

- The cache coherence problem arises from the fact that the same block may be in 2 or more caches.
- There is no problem with reading shared data. As soon as one processor writes to a cache block that is found in another processor's cache, the possibility of a problem arises.



Introductory Comments

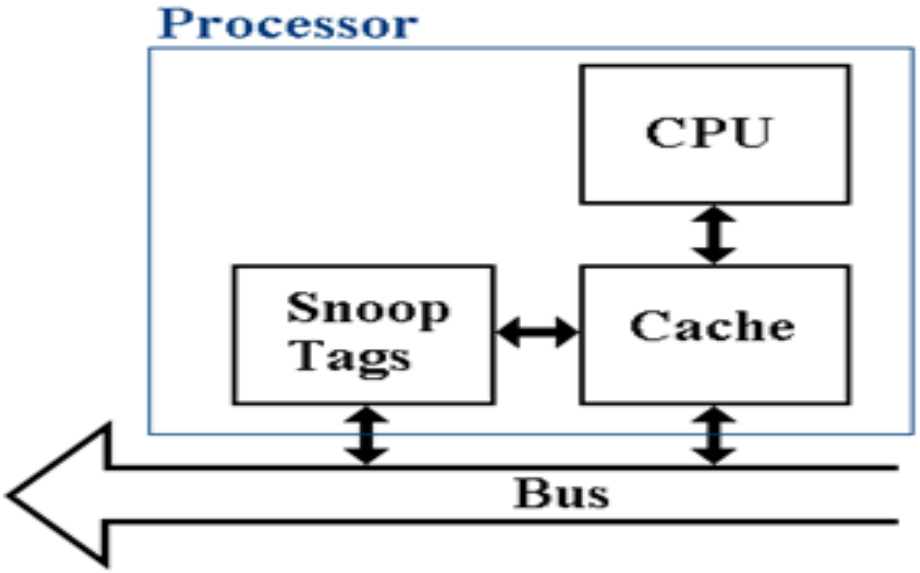
- Access to a cache by a processor involves one of two processes: read and write. Each process can have two results: a cache hit or a cache miss.
- Recall that a **cache hit** occurs when the processor accesses its private cache and finds the addressed item already in the cache. Otherwise, the access is a **cache miss**.
- **Read hits** occur when the individual processor attempts to read a data item from its private cache and finds it there. There is no problem with this access, no matter how many other private caches contain the data.
- The problem of processor receiving stale data on a read hit, due to updates by other independent processors, is handled by the cache write protocols.

Cache Coherence: Snoop Tags

- Cache blocks are identified and referenced by their memory tags.
- To maintain coherence, each cache monitors the address traffic on the shared bus.
- This is done by a **snooping cache** (or **snoopy cache**, after the Peanuts comic strip), which is just another port into the cache memory from the shared bus.
- The function of the snooping cache is to "**snoop the bus**", watching for references to memory blocks that have copies in the associated data cache.

The Snoopy Cache

- The goal of the snoopy cache is to generate a hit when there is a write to a cache line in that processor's private cache.

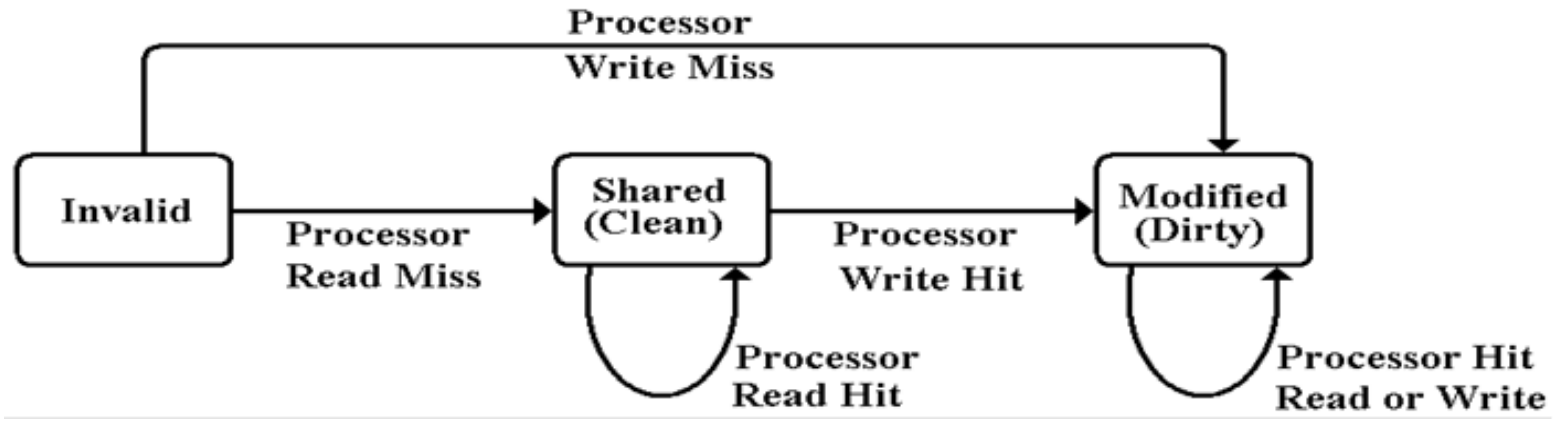


A Simple Coherence Protocol

- The cache line may be in one of three states.

1. Invalid – the cache line has no valid data.
2. Shared (Read Only) – the cache line has been loaded for a read request. The data are not changed, and may be in other caches.
3. Modified – the CPU has changed the data in the cache line. The memory block may not be shared; other caches have stale data.

A First Look at the Simple Protocol



- In a **read miss**, the individual processor acquires the bus and requests the block. When the block is read into the cache, it is labeled as "not dirty" and the read proceeds.
- In a **write miss**, the individual processor acquires the bus, requests the block, and then writes data to its copy in the cache. This sets the dirty bit on the cache block.
- Note that the processing of a write miss exactly follows the sequence that would be followed for a read miss followed by a write hit, referencing the block just read.

Cache Misses

- On either a read miss or a write miss, the processor must acquire the shared bus and request the memory block.
- If the requested block is not held by the cache of any other processor, the transition takes place as described above.
- If the requested block is held by another cache and that copy is labeled as “Modified”, then a sequence of actions must take place:
  1) the modified copy is written back to the shared primary memory,
  2) the requesting processor fetches the block just written back to the shared memory, and
  3) both copies are labeled as “Shared”.

Write Hits and Misses

- As we have noted above, the best way to view a write miss is to consider it as a sequence of events: first, a read miss that is properly handled, and then a write hit.
- This is due to the fact that the only way to handle a cache write properly is to be sure that the affected block has been read into memory.
- As a result of this two–step procedure for a write miss, we may propose a uniform approach that is based on proper handling of write hits.
- At the beginning of the process, it is the case that no copy of the referenced block in the cache of any other individual processor is marked as “Modified”.

Write Hits and Misses (Part 2)

- If the block in the cache of the requesting processor is marked as “Shared”, a write hit to it will cause the requesting processor to send out a “Cache Invalidate” signal to all other processors. Each of these other processors snoops the bus and responds to the Invalidate signal if it references a block held by that processor. The requesting processor then marks its cache copy as “Modified”.
- If the block in the cache of the requesting processor is already marked as “Modified”, nothing special happens. The write takes place and the cache copy is updated.

The MESI Protocol

- This is a commonly used cache coherency protocol. Its name is derived from the fours states in its FSM representation: **M**odified, **E**xclusive, **S**hared, and **I**nvalid.
- Each line in an individual processors cache can exist in one of the four following states:

1. Invalid — The cache line does not contain valid data.
2. Shared — Multiple caches may hold the line; the shared memory is up to date.
3. Exclusive — No other cache holds a copy of this line; the shared memory is up to date.
4. Modified — The line in this cache is valid; no copies of the line exist in other caches; the shared memory is not up to date.

MESI: The Exclusive State

- The main purpose of the Exclusive state is to prevent the unnecessary broadcast of a Cache Invalidate signal on a write hit. This reduces traffic on a shared bus.
- Recall that a necessary precondition for a successful write hit on a line in the cache of a processor is that no other processor has that line with a label of Exclusive or Modified.
- As a result of a successful write hit on a cache line, that cache line is always marked as Modified.

The MESI: Write Hit

- Suppose a requesting processor processing a write hit on its cache. By definition, any copy of the line in the caches of other processors must be in the Shared State. What happens depends on the state of the cache in the requesting processor.

1. Modified The protocol does not specify an action for the processor.
2. Shared The processor writes the data, marks the cache line as Modified, and broadcasts a Cache Invalidate signal to other processors.
3. Exclusive The processor writes the data and marks the cache line as Modified.

Processor P1 has the Cache Line

- Here is a scenario:

1. P1 has written to the Cache Line; it is Modified.
2. Another processor attempts to fetch the same block from the shared main memory.
3. P1's snoop cache detects the memory request. P1 broadcasts a message "Dirty" on the shared bus, causing the other processor to abandon its memory fetch.
4. P1 writes the block back to the shared memory and the other processor can access it.

Local Events in the MESI Protocol

- **Local Read** The individual processor reads from its cache memory.
- **Local Write** The individual processor writes data to its cache memory.
- **Local Eviction** The individual processor must remove a block from its cache in order to free up a cache line for a newly requested block. Shared memory is updated, if needed.

Bus Events in the MESI Protocol

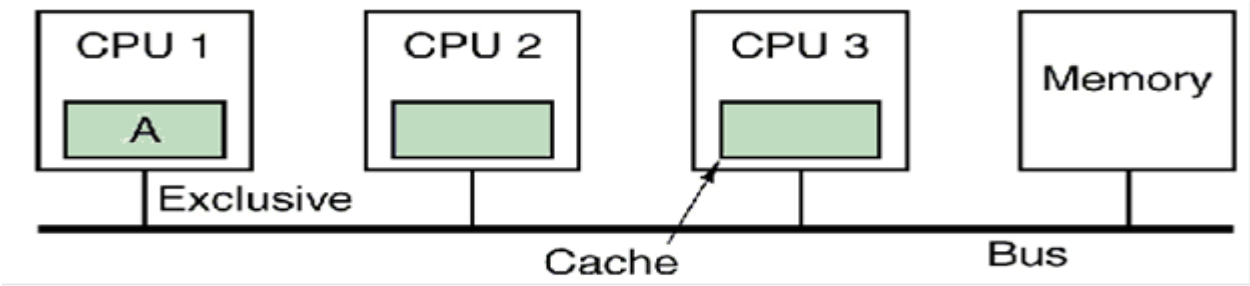
- **Bus Read** Another processor issues a read request to the shared primary memory for a block that is held in this processors individual cache.
  This processor's snoop cache detects the request.
- **Bus Write** Another processor issues a write request to the shared primary memory for a block that is held in this processors individual cache.
- **Bus Upgrade** Another processor signals that a write to a cache line that is shared with this processor. The other processor will upgrade the status of the cache line from "Shared" to "Modified".

FSM for MESI

PS	Local Read	Local Write	Local Eviction	BR Bus Read	BW Bus Write	BU – Bus Upgrade
I Invalid	Issue BR Do other caches have this line. Yes: NS = S No: NS = E	Issue BW NS = M	NS = I	Do nothing	Do nothing	Do nothing
S Shared	Do nothing	Issue BU NS = M	NS = I	Respond Shared	NS = I	NS = I
E Exclusive	Do nothing	NS = M	NS = I	Respond Shared NS = S	NS = I	Error Should not occur.
M Modified	Do nothing	Do nothing	Write data back. NS = I.	Respond Dirty. Write data back NS = S	Respond Dirty. Write data back NS = I	Error Should not occur.

MESI Illustrated

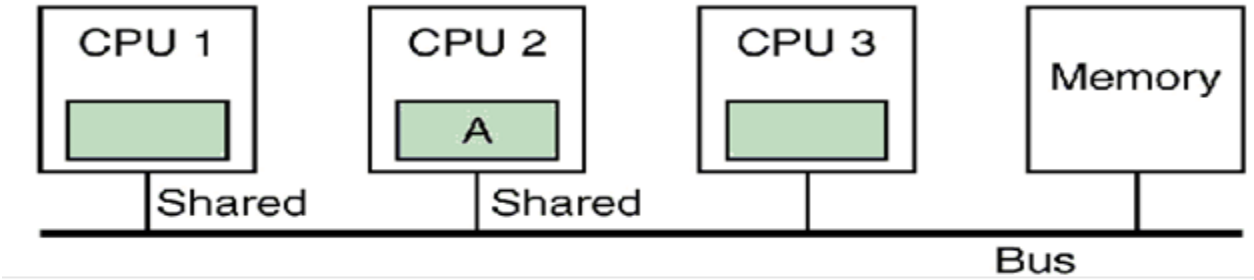
- When the multiprocessor is turned on, all cache lines are marked invalid.



- CPU 1 requests block A from the shared memory.
- It issues a BR (Bus Read) for the block and gets its copy.
- The cache line containing block A is marked Exclusive.
- Subsequent reads to this block access the cached entry and not the shared memory.
- Neither CPU 2 nor CPU 3 respond to the BR.

MESI Illustrated (Step 2)

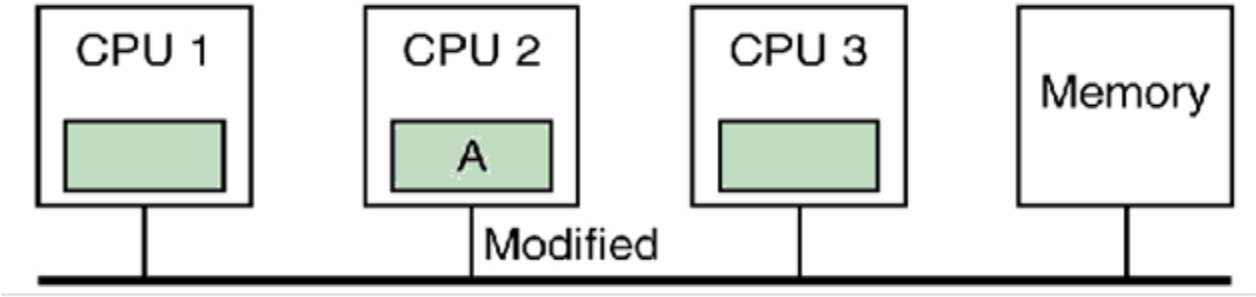
- CPU 2 requests the same block. The snoop cache on CPU 1 notes the request and CPU 1 broadcasts "Shared", announcing that it has a copy of the block.



- Both copies of the block are marked as shared.
- This indicates that the block is in two or more caches for reading and that the copy in the shared primary memory is up to date.
- CPU 3 does not respond to the BR.

MESI Illustrated (Step 3)

- Suppose that CPU 2 writes to the cache line it is holding in its cache. It issues a BU (Bus Upgrade) broadcast, marks the cache line as Modified, and writes the data to the line.
- CPU 1 responds to the BU by marking the copy in its cache line as Invalid.



- CPU 3 does not respond to the BU.
- Informally, CPU 2 can be said to “own the cache line”.

MESI Illustrated (Step 4)

- Now suppose that CPU 3 attempts to read block A .
- For CPU 1, the cache line holding that block is marked as Invalid. CPU 1 does not respond to the BR (Bus Read).
- CPU 2 has the cache line marked as Modified. It asserts the signal “Dirty” on the bus, writes the data in the cache line back to the shared memory, and marks the line “Shared”.
- Informally, CPU 2 asks CPU 3 to wait while it writes back the contents of its modified cache line to the shared primary memory. CPU 3 waits and then gets a correct copy. The cache line in each of CPU 2 and CPU 3 is marked as Shared.