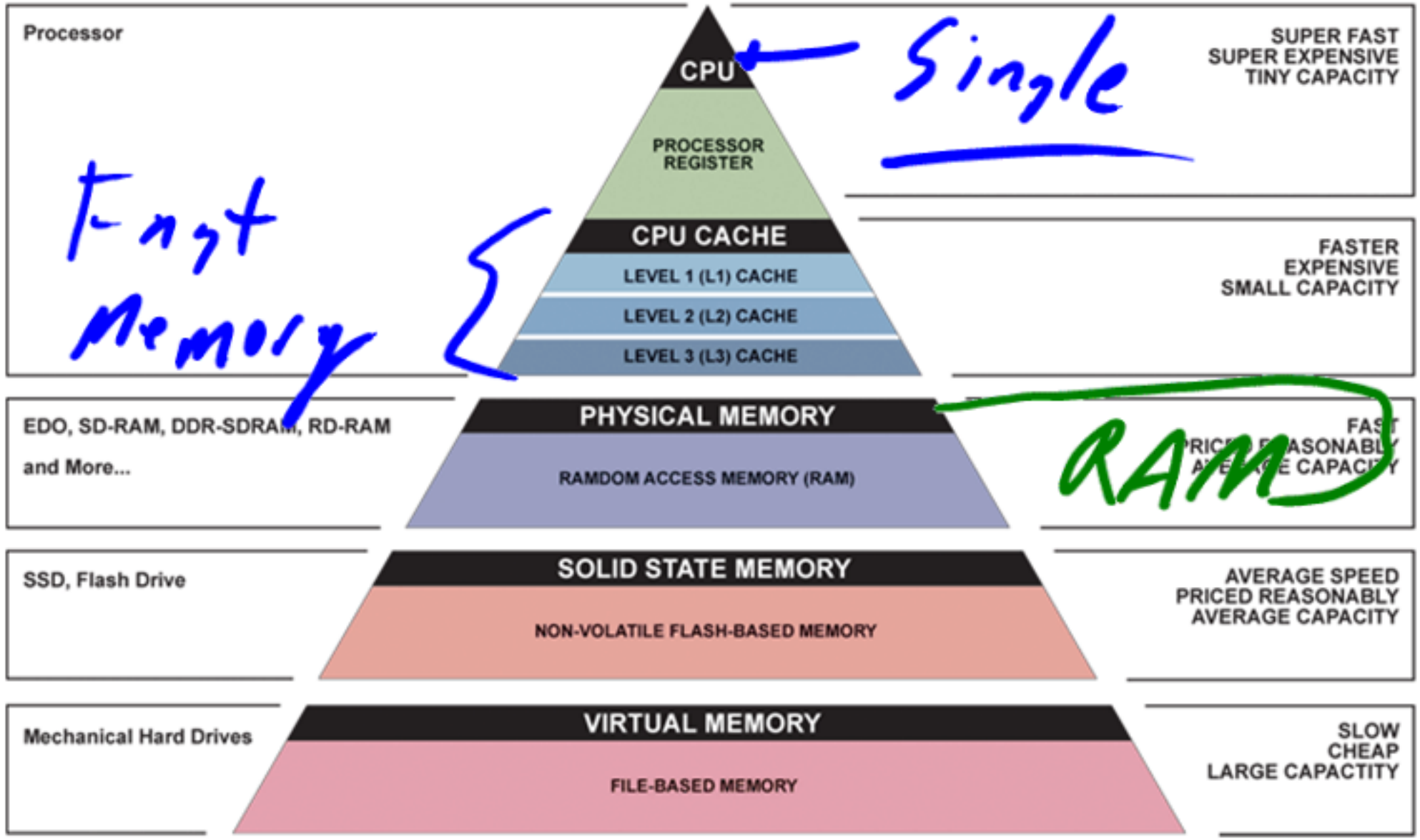
Cache Coherence

Lecture Objectives:

1) Explain the difference between write back and write through caching
2) Explain the difference between write allocate and no-write allocate
3) Explain the problem of cache coherence
4) Explain how cache coherence can be dealt with
5) Explain snooping based cache coherence
6) Explain false sharing

snoopy

The Memory Hierarchy



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

*http://www.bit-tech.net/hardware/memory/2007/11/15/the_secrets_of_pc_memory_part_1/3

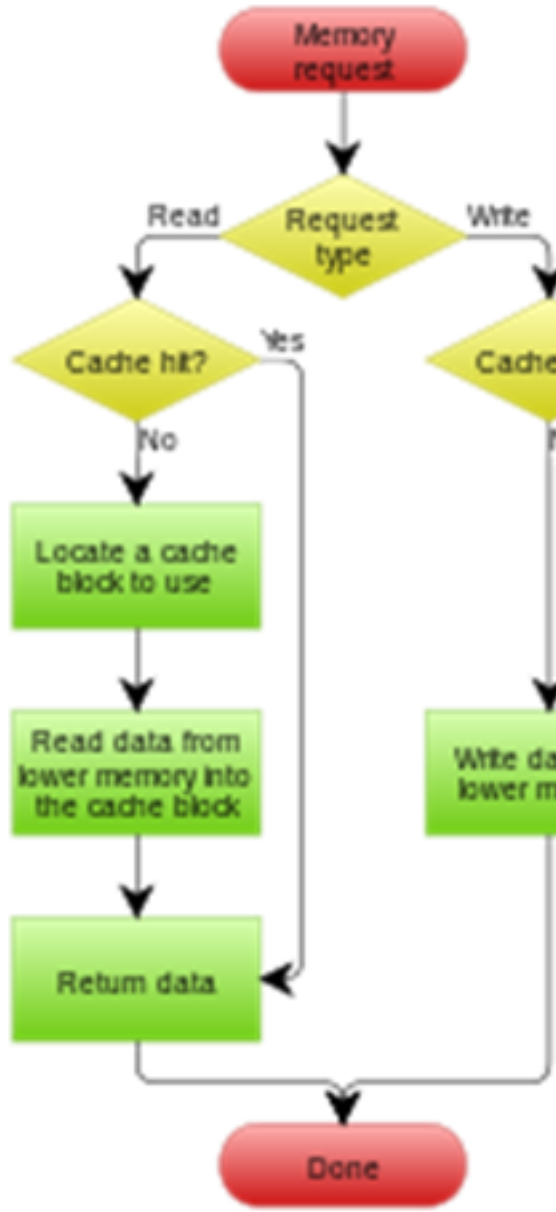
Writing Policies

- Write Through Cache
  - Writing is done simultaneously to both the cache and the backing store
- Write back (write behind)
  - Initially, writing is done only to the cache. The write to the backing store is postponed until the blocks are to be replaced by new content
  - Generally used to improve performance

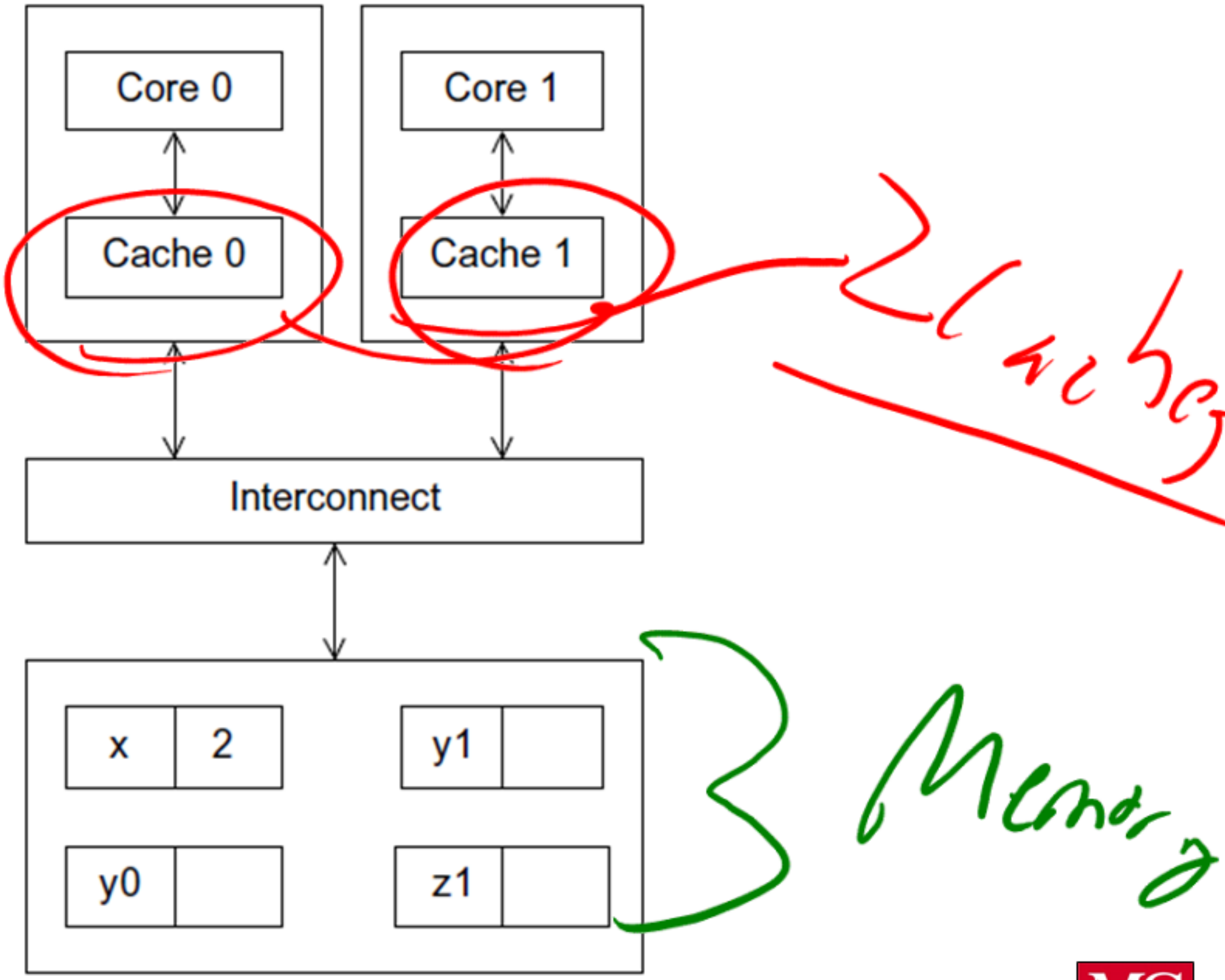
What if a memory location is not loaded into cache?

- **Write allocate** (aka **Fetch on write**) - Datum at the missed-write location is loaded to cache, followed by a write-hit operation. In this approach, write misses are similar to read-misses.
- **No-write allocate** (aka **Write-no-allocate**, **Write around**) - Datum at the missed-write location is not loaded to cache, and is written directly to the backing store. In this approach, only system reads are being cached.

Write Cache Flowcharts (Wikipedia)



One example system



Write Invalidate

- processor obtains exclusive access for writes (becomes the "owner") by invalidating data in other processors' caches coherency miss (invalidation miss) cache-to-cache transfers
- good for:
  - multiple writes to same word or block by one processor
  - migratory sharing from processor to processor
  - a problem is false sharing
  - processors read & write to different words in a shared cache block
  - cache coherency is maintained on a cache block basis
  - block ownership bounces between processor caches

Write Update

- broadcast each write to actively shared data
- each processor with a copy snarfs the data
- good for inter-processor contention

Snooping Implementation

A distributed coherency protocol

- coherency state associated with each cache block
- each snoop maintains coherency for its own cache

How the bus is used

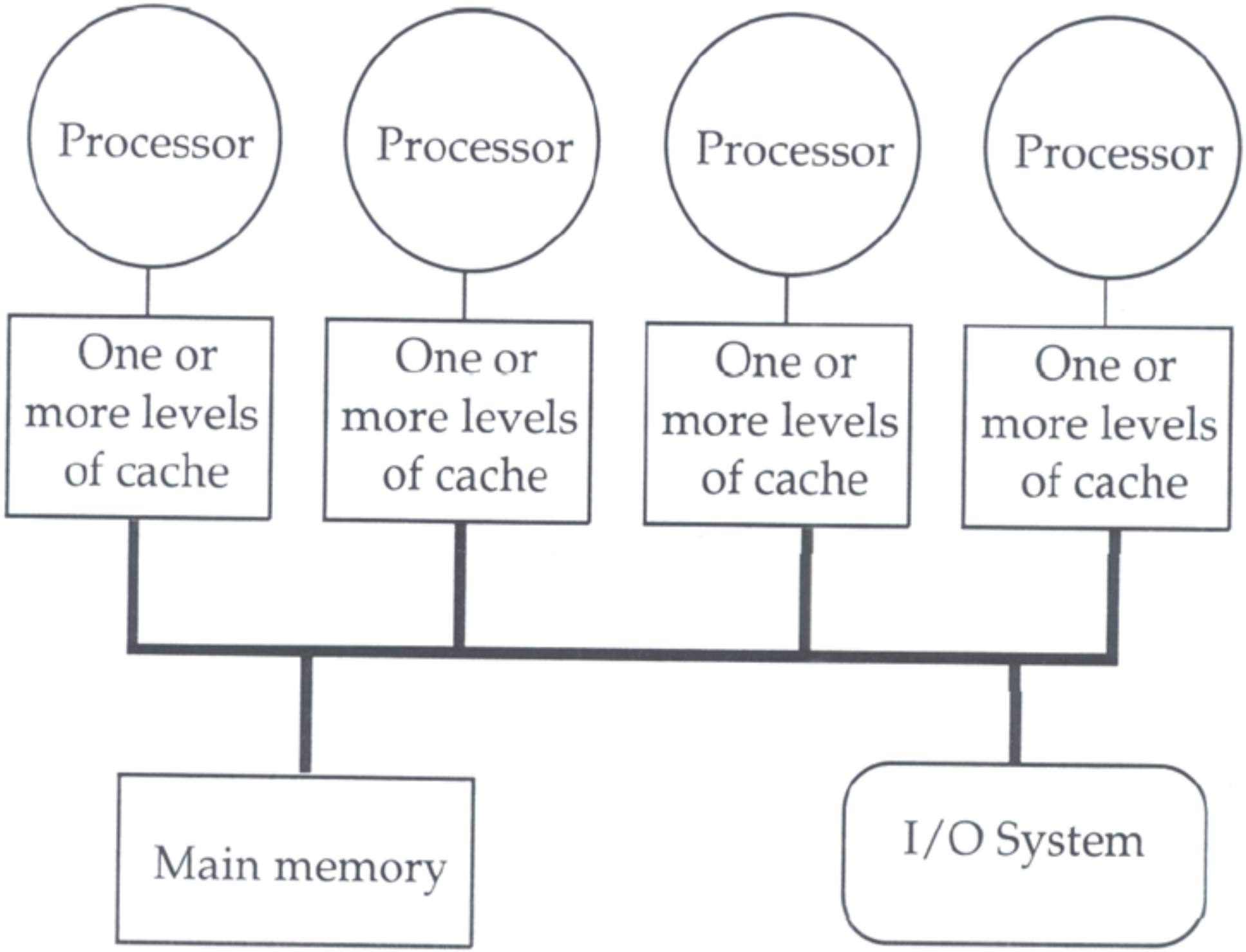
- broadcast medium
- entire coherency operation is atomic wrt other processors
- **keep-the-bus protocol**: master holds the bus until the entire operation has completed
- **split-transaction buses**:
  - request & response are different phases
  - state value that indicates that an operation is in progress
  - do not initiate another operation for a cache block that has one in progress

Snooping Implementation

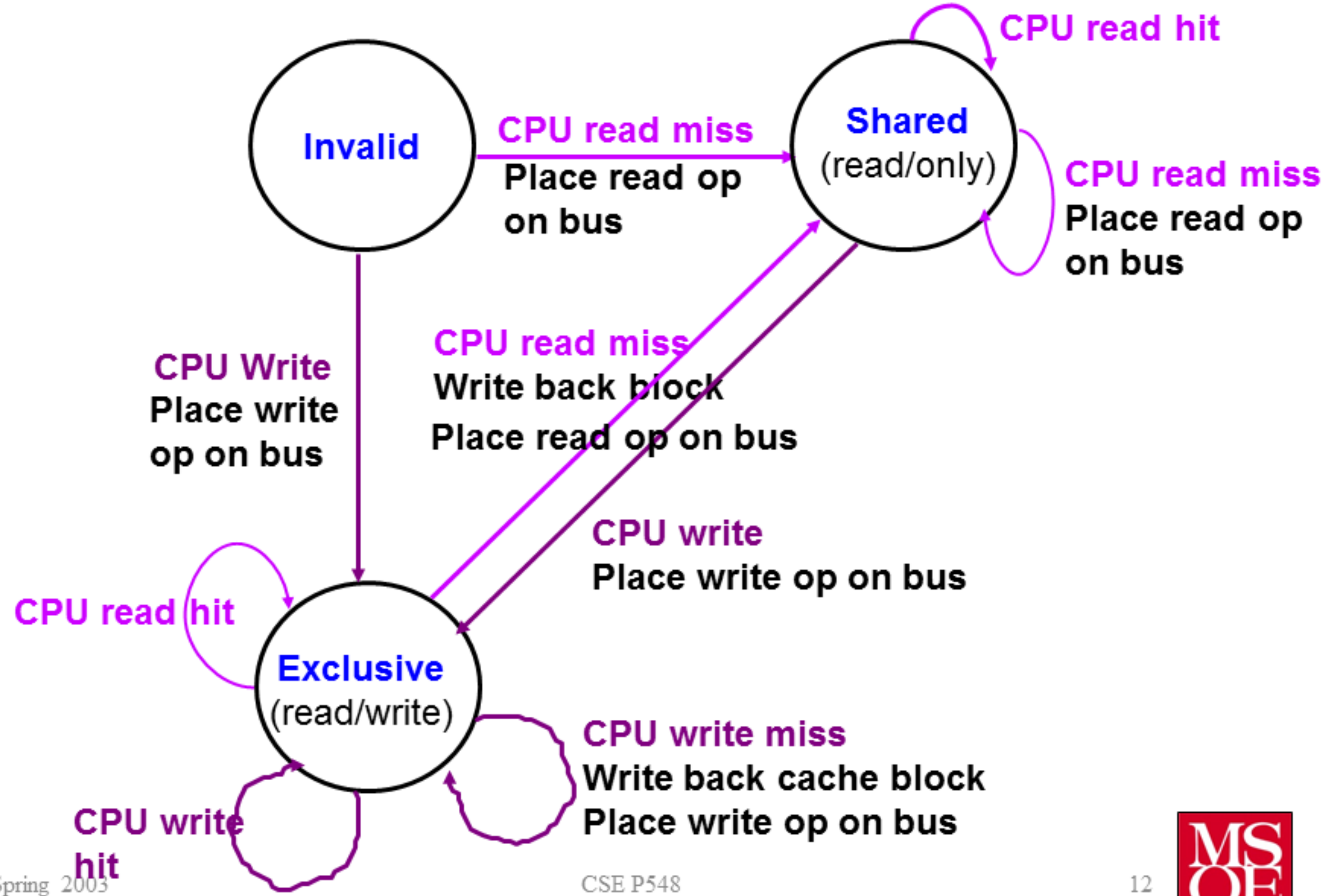
Snoop implementation:

- separate tags & state for snoop lookups
  - processor & snoop communicate for a state or tag change
- snoop on the highest level cache
  - another reason it is a physically-accessed cache
  - property of inclusion:
    - all blocks in L1 are in L2
    - therefore only have to snoop on L2
    - may need to update L1 state if change L2 state

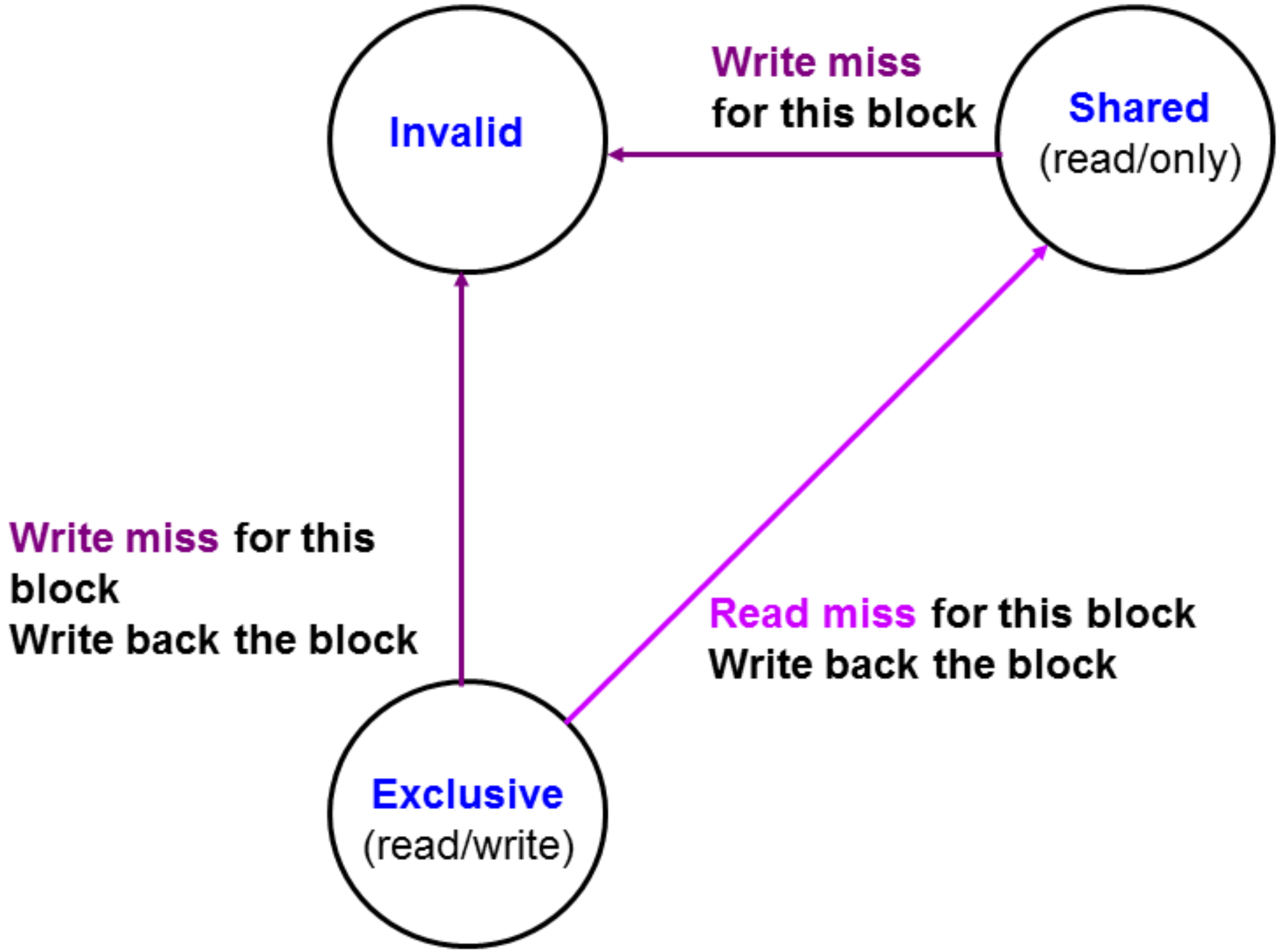
A Low-end MP


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
Main memory
I/O System

State Machine (CPU side)



State Machine (Bus side)



Directory Implementation

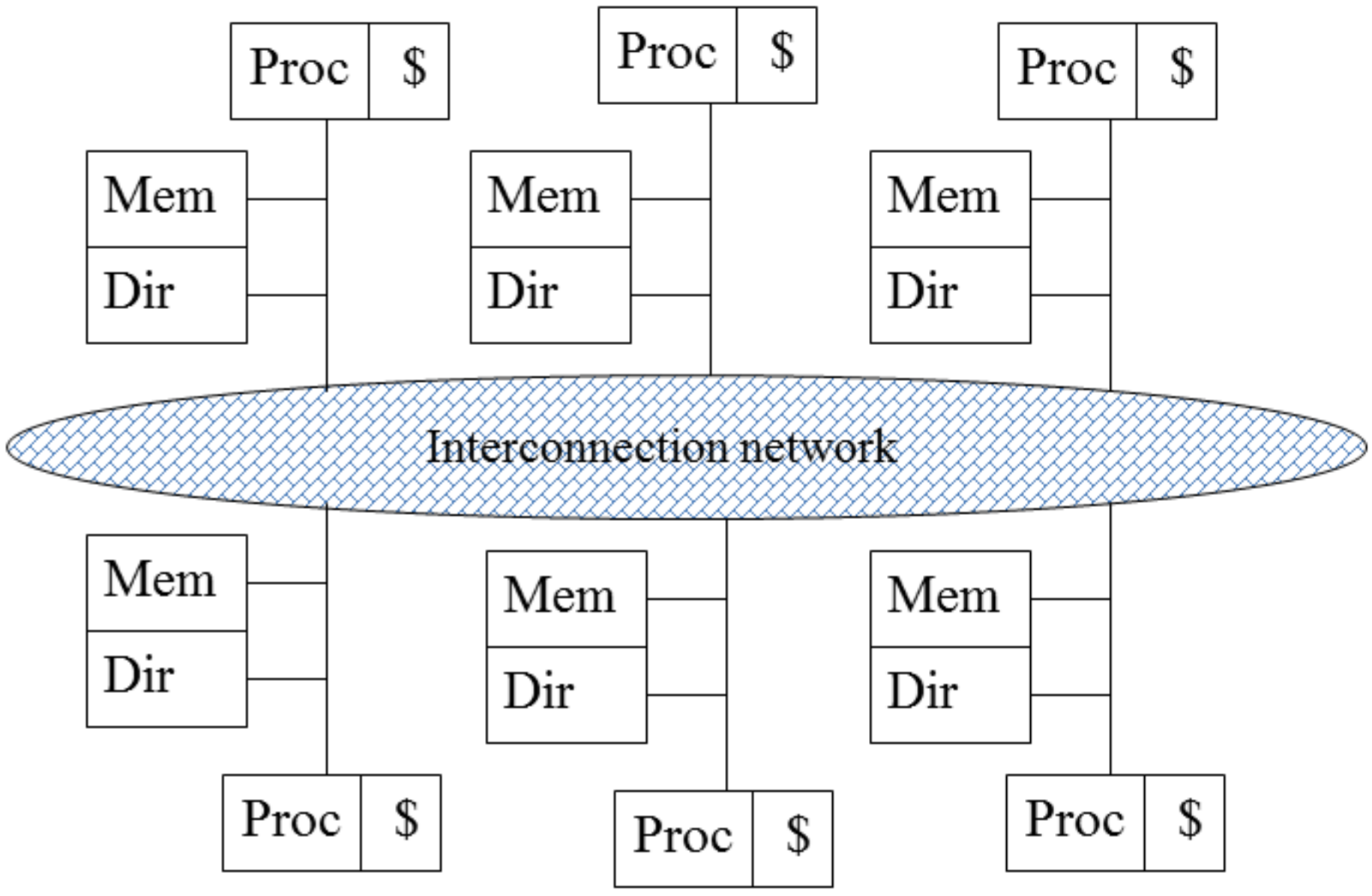
Distributed memory

- each processor (or cluster of processors) has its own memory
- processor-memory pairs are connected via a multi-path interconnection network
  - snooping with broadcasting is wasteful
  - **point-to-point communication** instead
- a processor has fast access to its local memory & slower access to "remote" memory located at other processors
  - **NUMA** (non-uniform memory access) machines

How cache coherency is handled

- no caches (Tera (Cray) MTA)
- disallow caching of shared data (Cray 3TD)
- hardware directories that record cache block state

A High-end MP



Directory Implementation

Coherency state is associated with memory blocks that are the size of cache blocks

- cache state
  - **shared**:
    - at least 1 processor has the data cached & memory is up-to-date
    - block can be read by any processor
  - **exclusive**:
    - 1 processor (the owner) has the data cached & memory is stale
    - only that processor can write to it
  - **invalid**:
    - no processor has the data cached & memory is up-to-date
- which processors have read/write access
  - bit vector in which 1 means the processor has the data
    - optimization: space for 4 processors & trap for more
  - write bit

Directory Implementation

Directories have different meanings (& therefore uses) to different processors

- **home** node: where the memory location of an address resides (and cached data may be there too) (static)
- **local** node: where the request initiated (relative)
- **remote** node: alternate location for the data if this processor has requested it (dynamic)

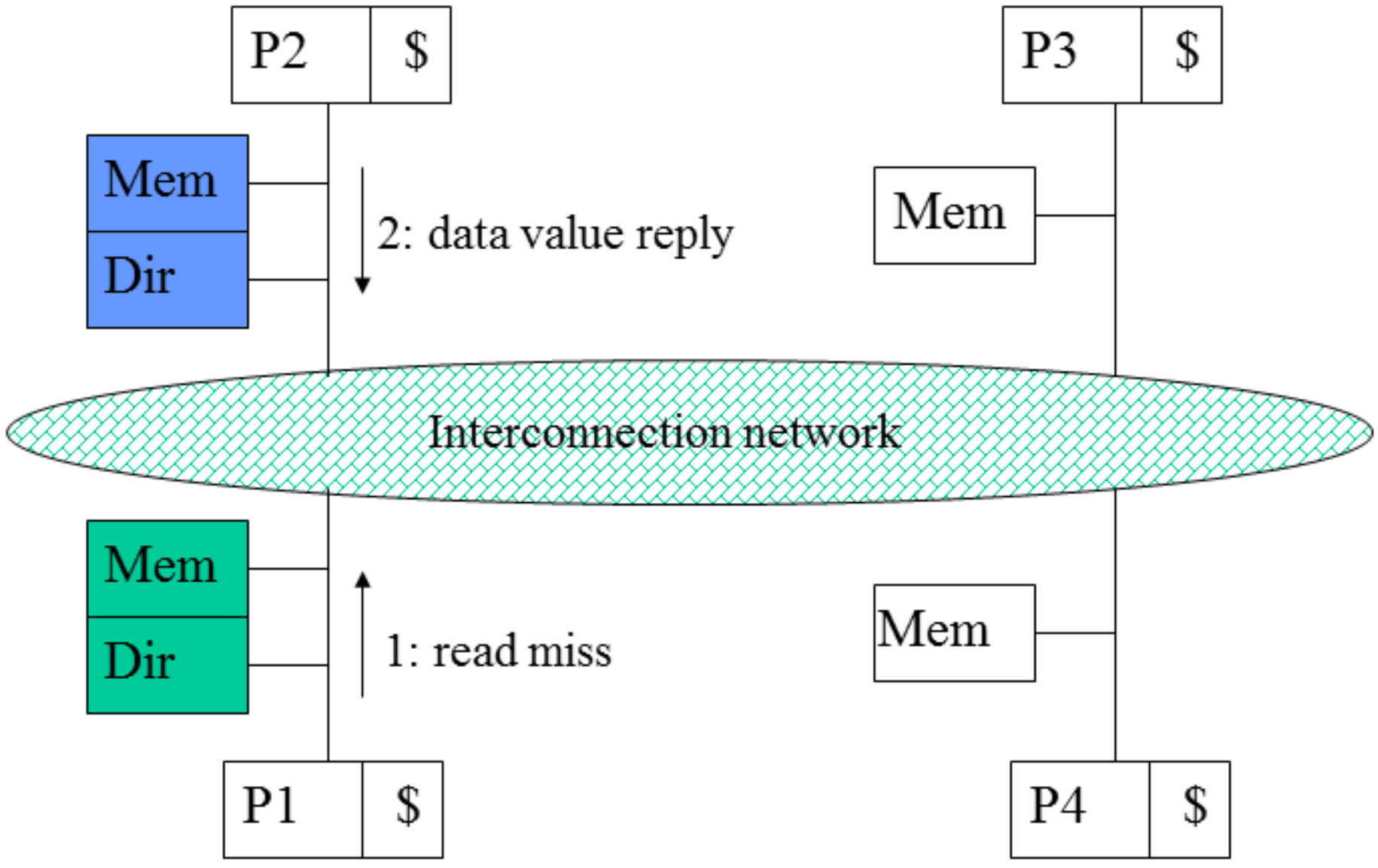
In satisfying a memory request:

- messages sent between the different nodes in point-to-point communication
- messages get explicit replies

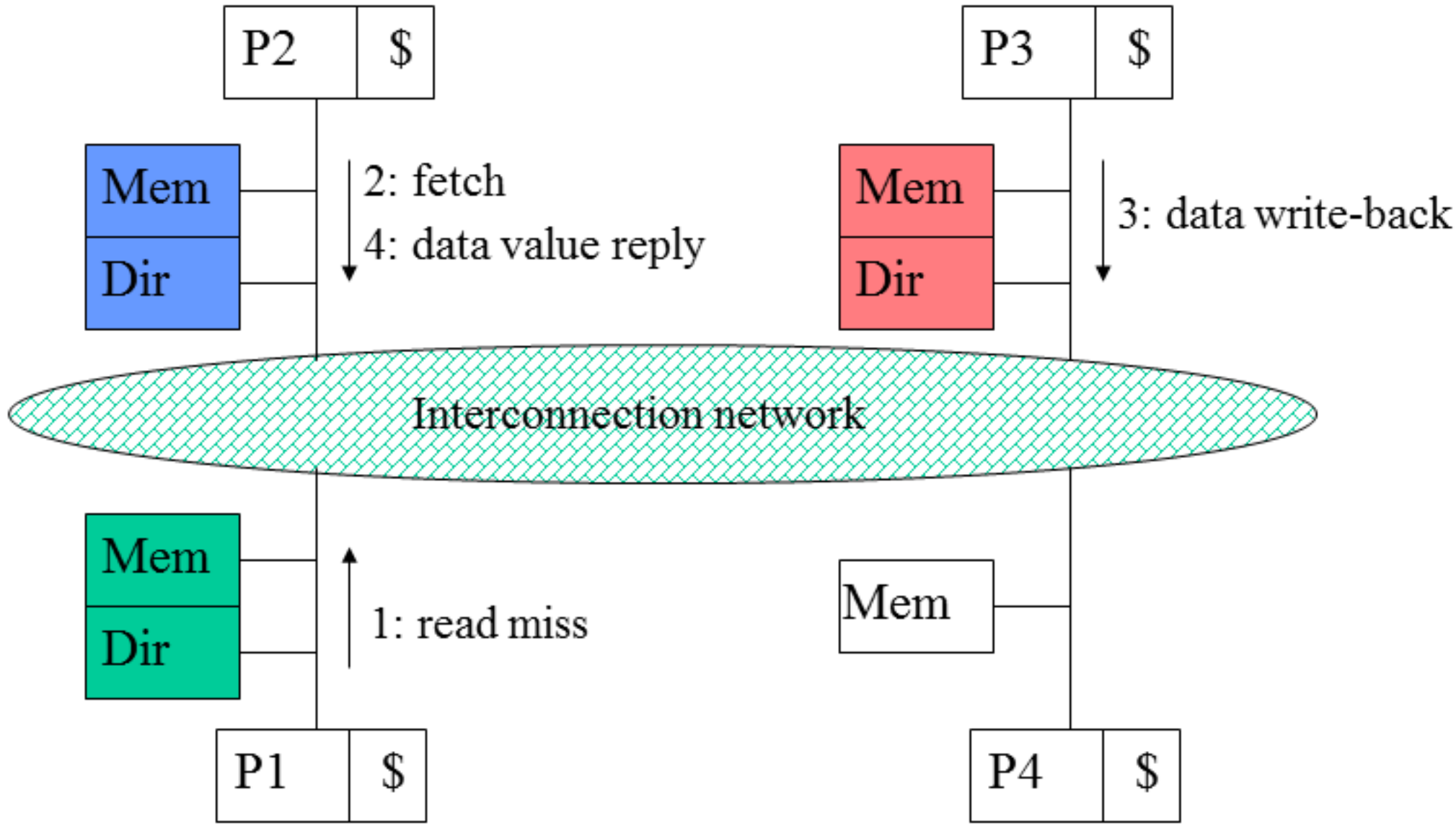
Some simplifying assumptions for using the protocol

- processor blocks until the access is complete
- messages processed in the order received

Read Miss for an Uncached Block



Read Miss for an Exclusive, Remote Block



Write Miss for an Exclusive, Remote Block