

Cache Coherency

Cache coherent processors

- most current value for an address is the last write
- all reading processors must get the most current value

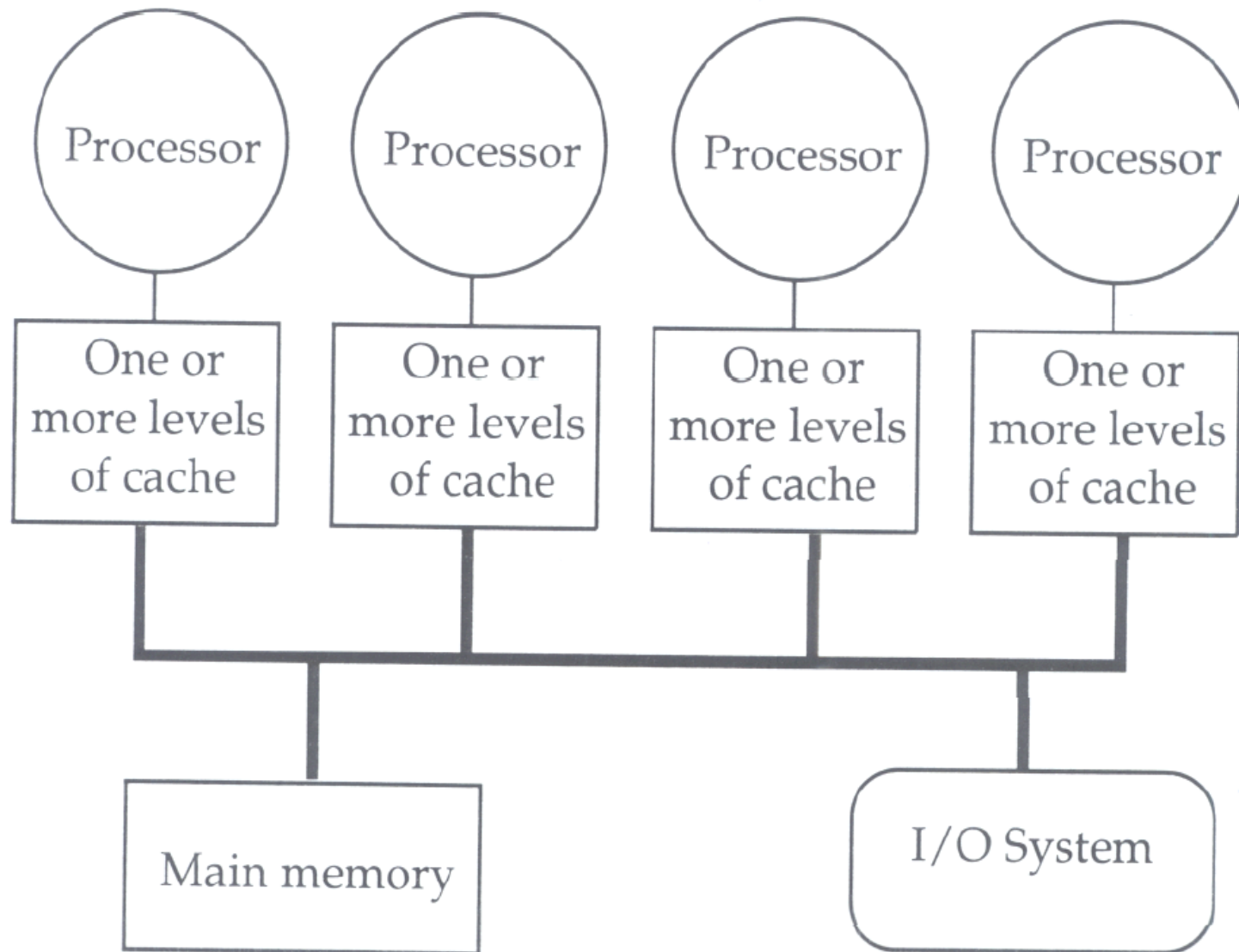
Cache coherency problem

- update from a writing processor is not known to other processors

Cache coherency protocols

- mechanism for maintaining cache coherency
- coherency state associated with a block of data
- bus/interconnect operations on shared data change the state
 - for the processor that initiates an operation
 - for other processors that have the data of the operation resident in their caches

A Low-end MP



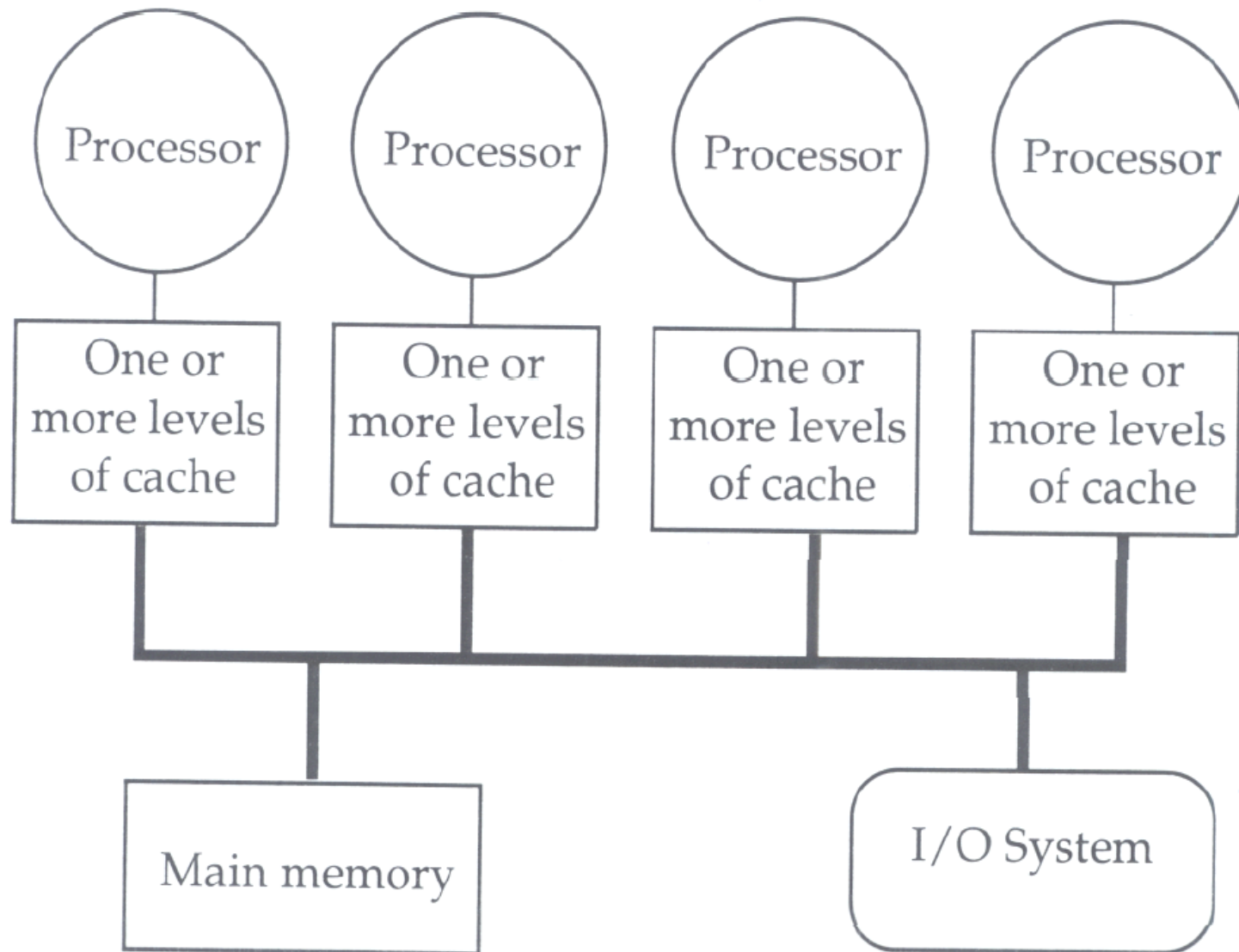
Cache Coherency Protocols

Write-invalidate

(Sequent Symmetry, SGI Power/Challenge, SPARCCenter 2000)

- 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 Low-end MP



Cache Coherency Protocols

Write-update

(SPARCCenter 2000)

- broadcast each write to actively shared data
- each processor with a copy snoops/takes the data
- good for **inter-processor contention**

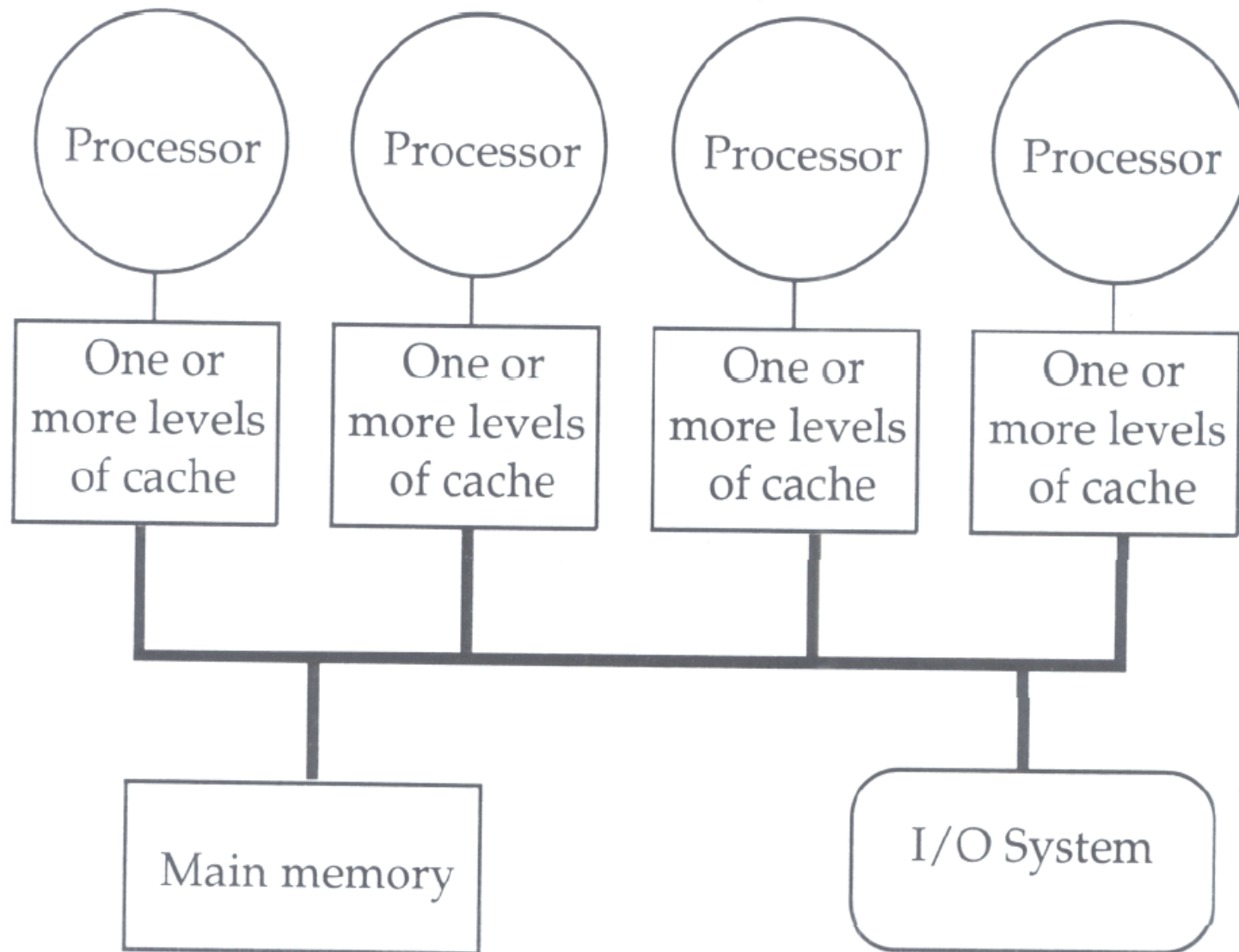
Competitive

(Alphas)

- switches between them

We will focus on write-invalidate.

A Low-end MP



Cache Coherency Protocol Implementations

Snooping

- used with low-end MPs
 - few processors
 - centralized memory
 - bus-based
- distributed implementation: responsibility for maintaining coherence lies with each cache

Directory-based

- used with higher-end MPs
 - more processors
 - distributed memory
 - multi-path interconnect
- centralized for each address: responsibility for maintaining coherence lies with the directory for each address

Snooping Implementation

A distributed coherency protocol

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

Snooping Implementation

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:

- snoop on the highest level cache
 - another reason L2 is physically-accessed
 - 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
- separate tags & state for snoop lookups
 - processor & snoop communicate for a state or tag change

An Example Snooping Protocol

Invalidation-based coherency protocol

Each cache block is in one of three states

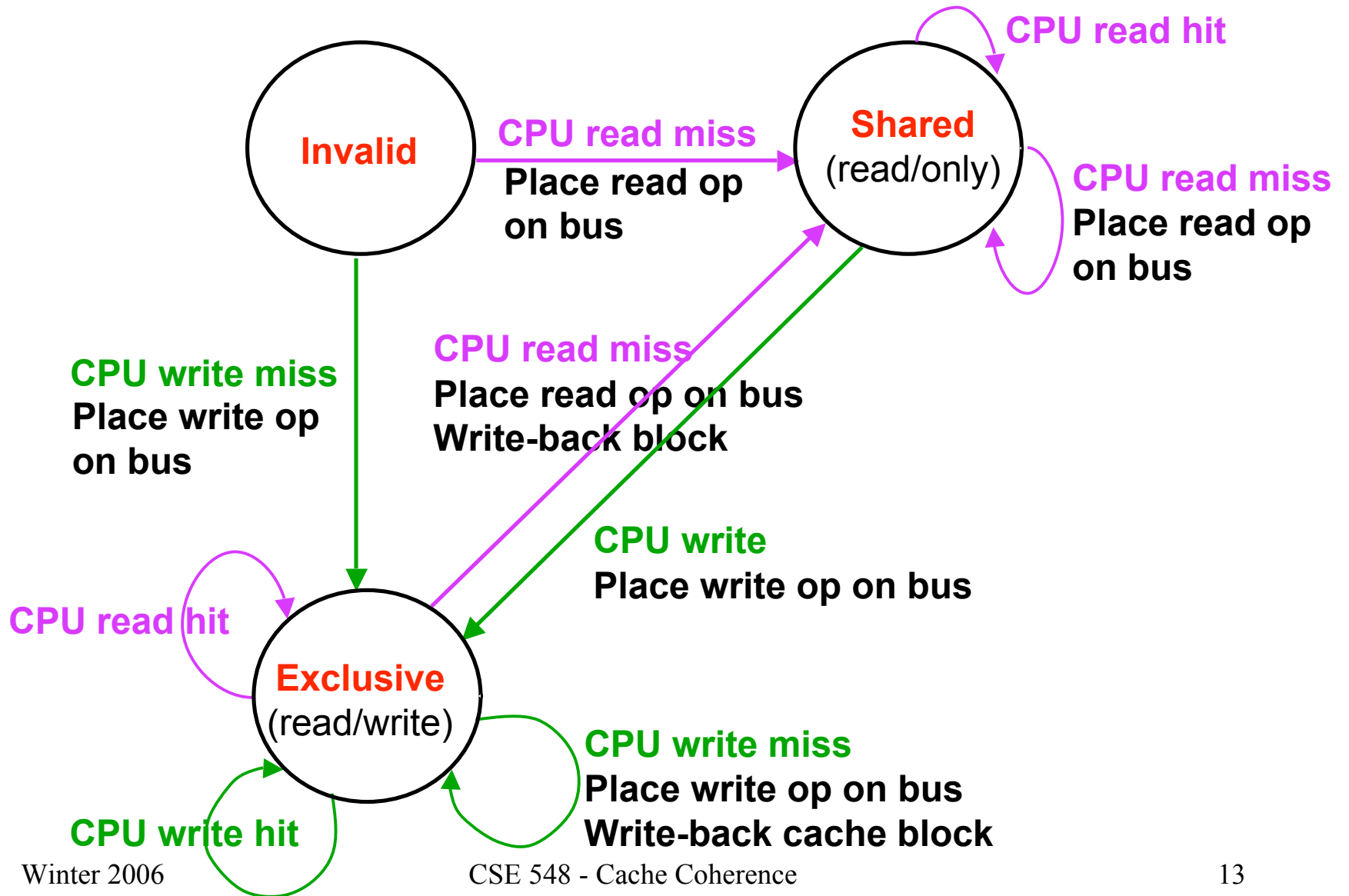
- **shared:**
 - clean in all caches & up-to-date in memory
 - block can be read by any processor
- **exclusive:**
 - dirty in exactly one cache
 - only that processor can write to it
- **invalid:**
 - block contains no valid data

State Transitions for a Given Cache Block

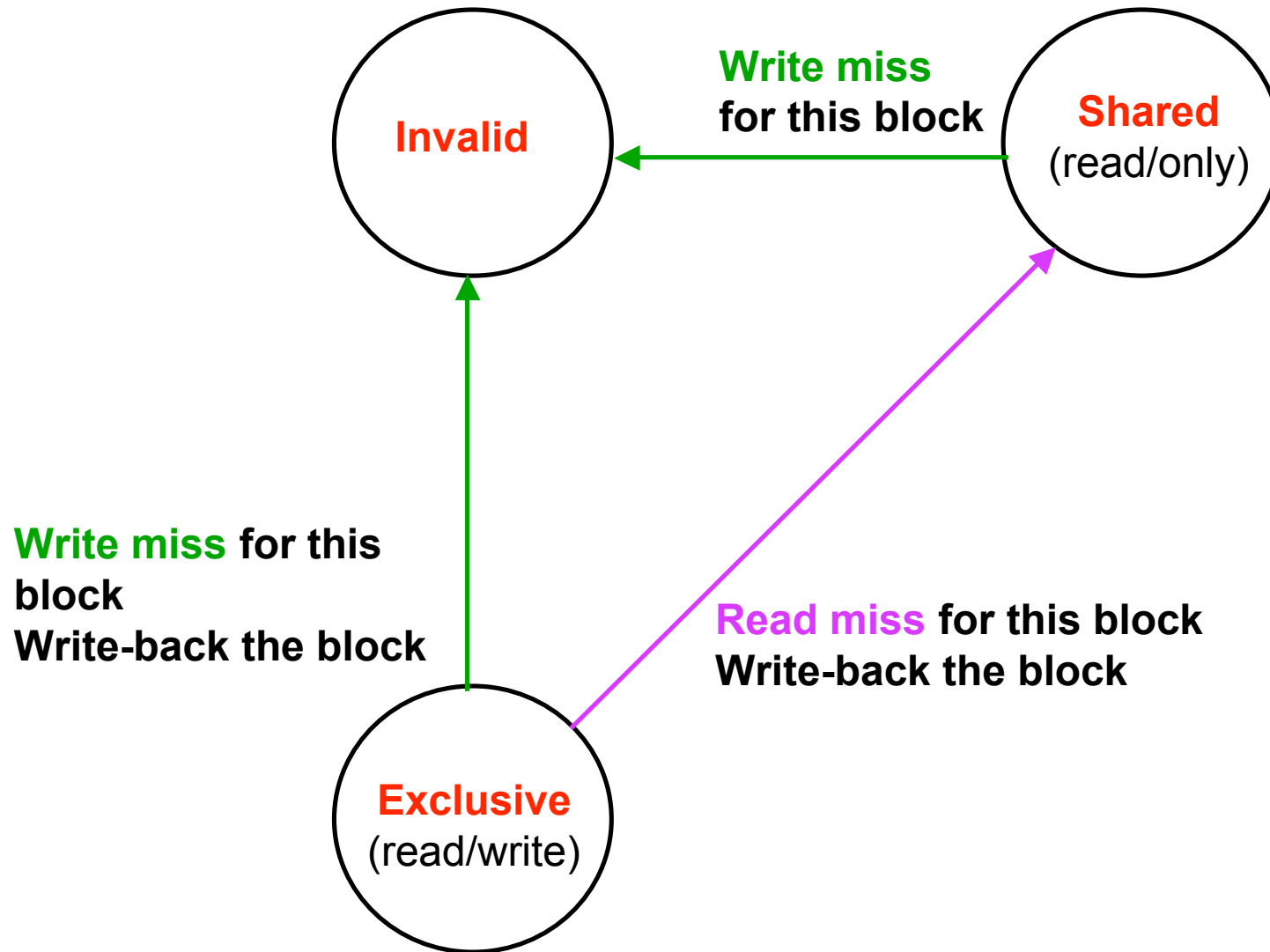
State transitions caused by:

- events caused by the **requesting processor**, e.g.,
 - read miss, write miss, write on shared block
- events caused by **snoops of other caches**, e.g.,
 - read miss by P1 makes P2's owned block change from exclusive to shared
 - write miss by P1 makes P2's owned block change from exclusive to invalid

State Machine (CPU side)



State Machine (Bus side: the snoop)

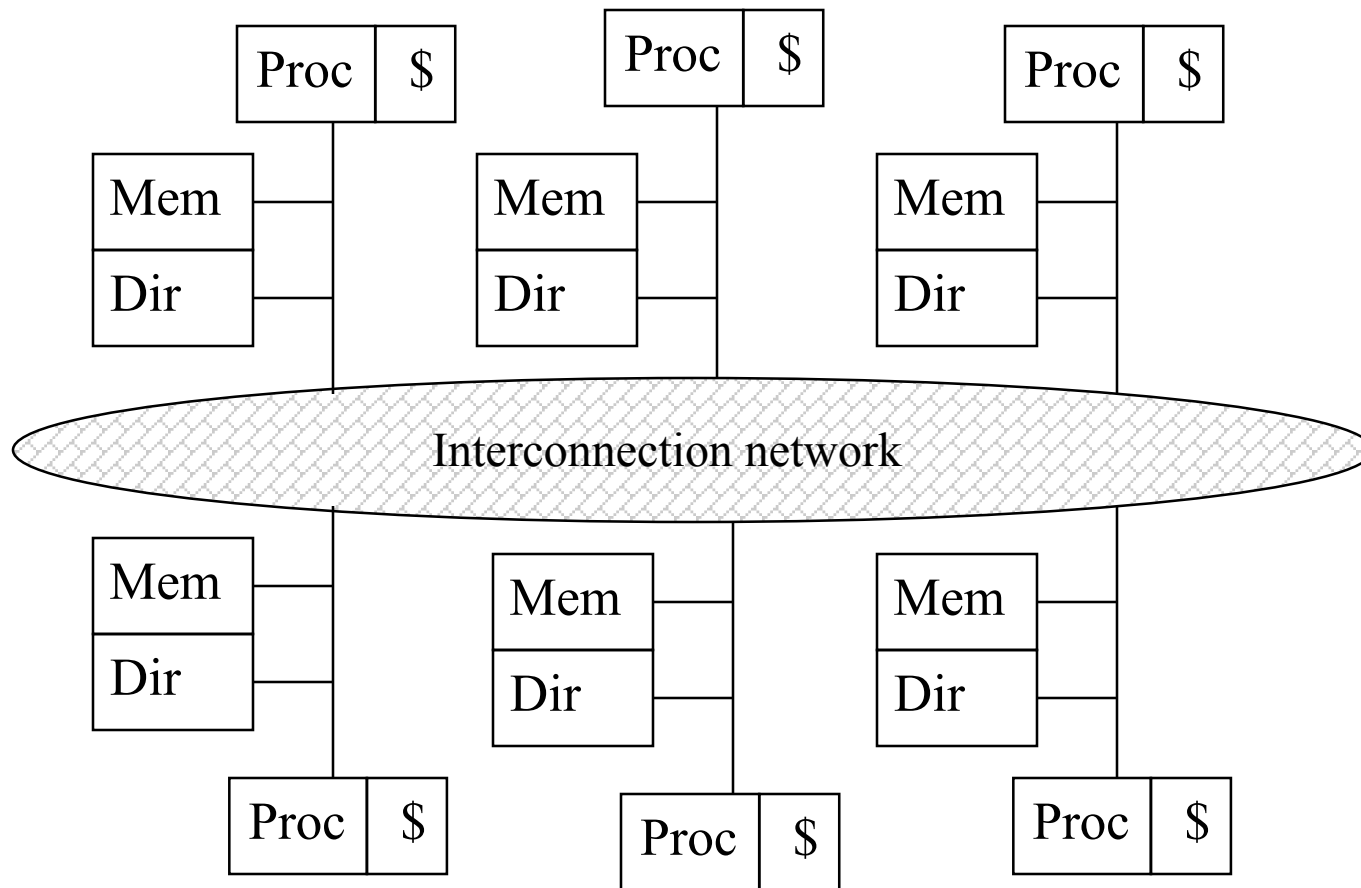


Directory Implementation

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

A High-end MP



Directory Implementation

How cache coherency is handled

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

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
- directory state
 - bit vector in which 1 means the processor has cached the data
 - write bit to indicate if exclusive

Directory Implementation

Directories have different uses to different processors

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

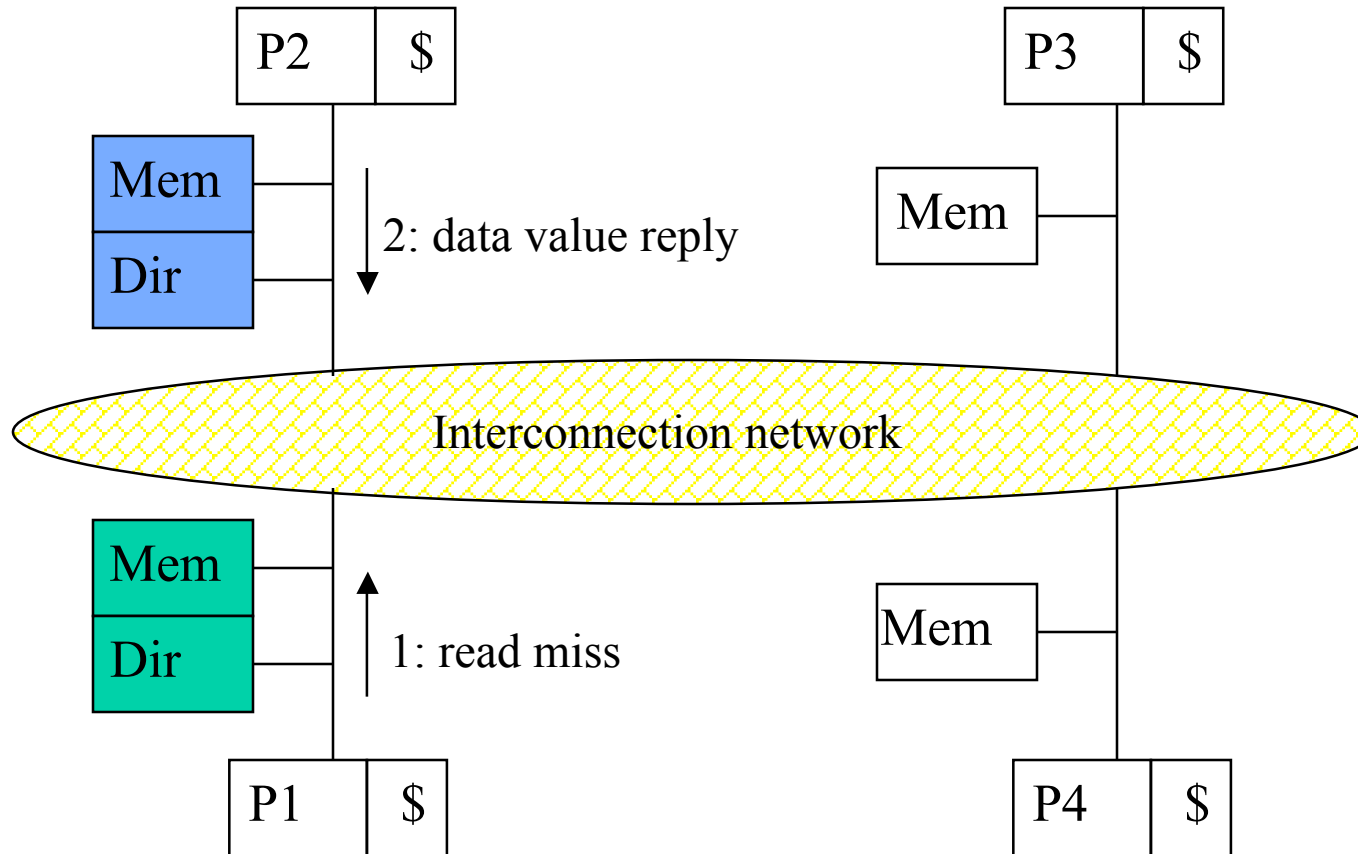
In satisfying a memory request:

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

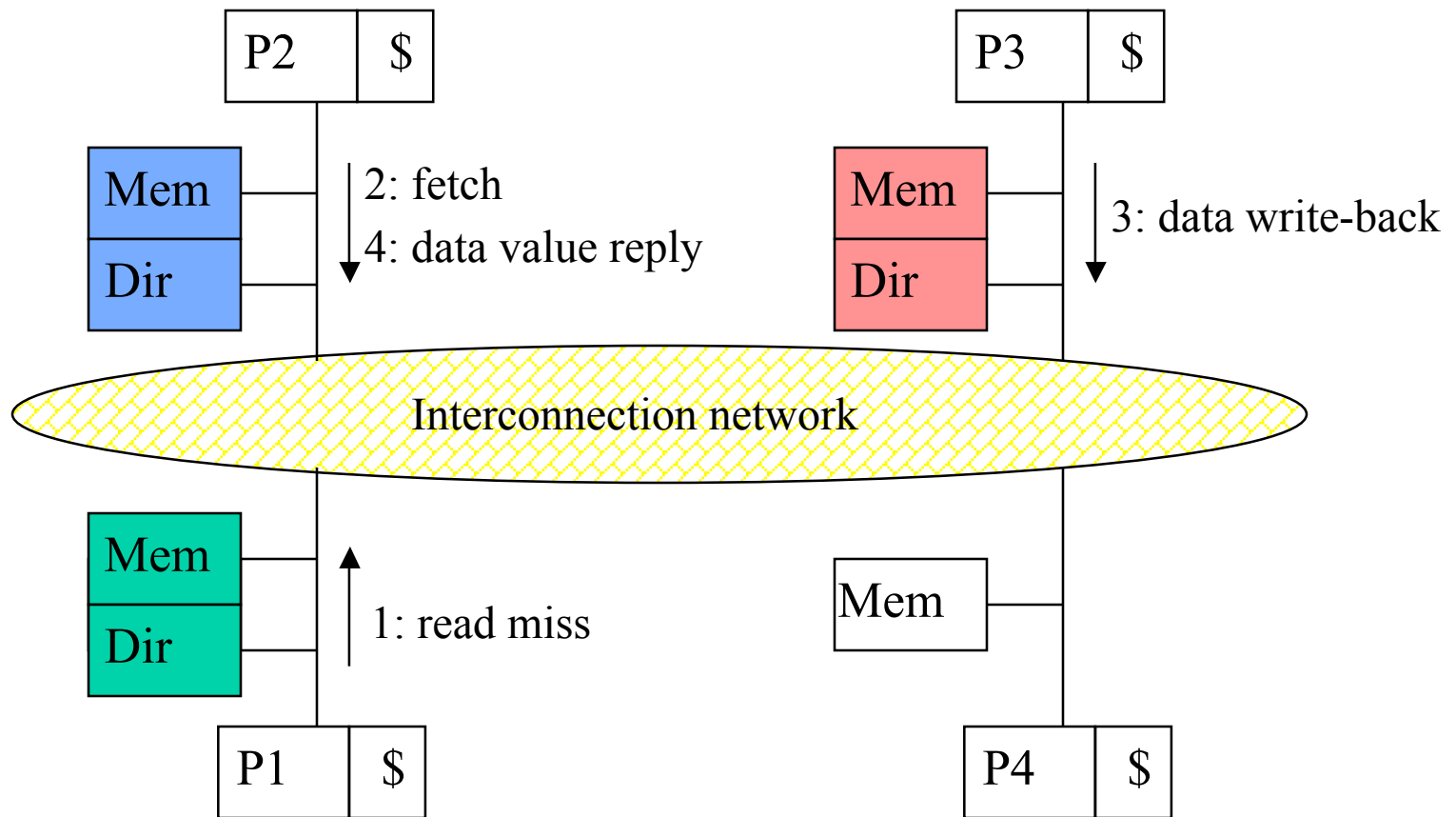
Some simplifying assumptions for using the protocol

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

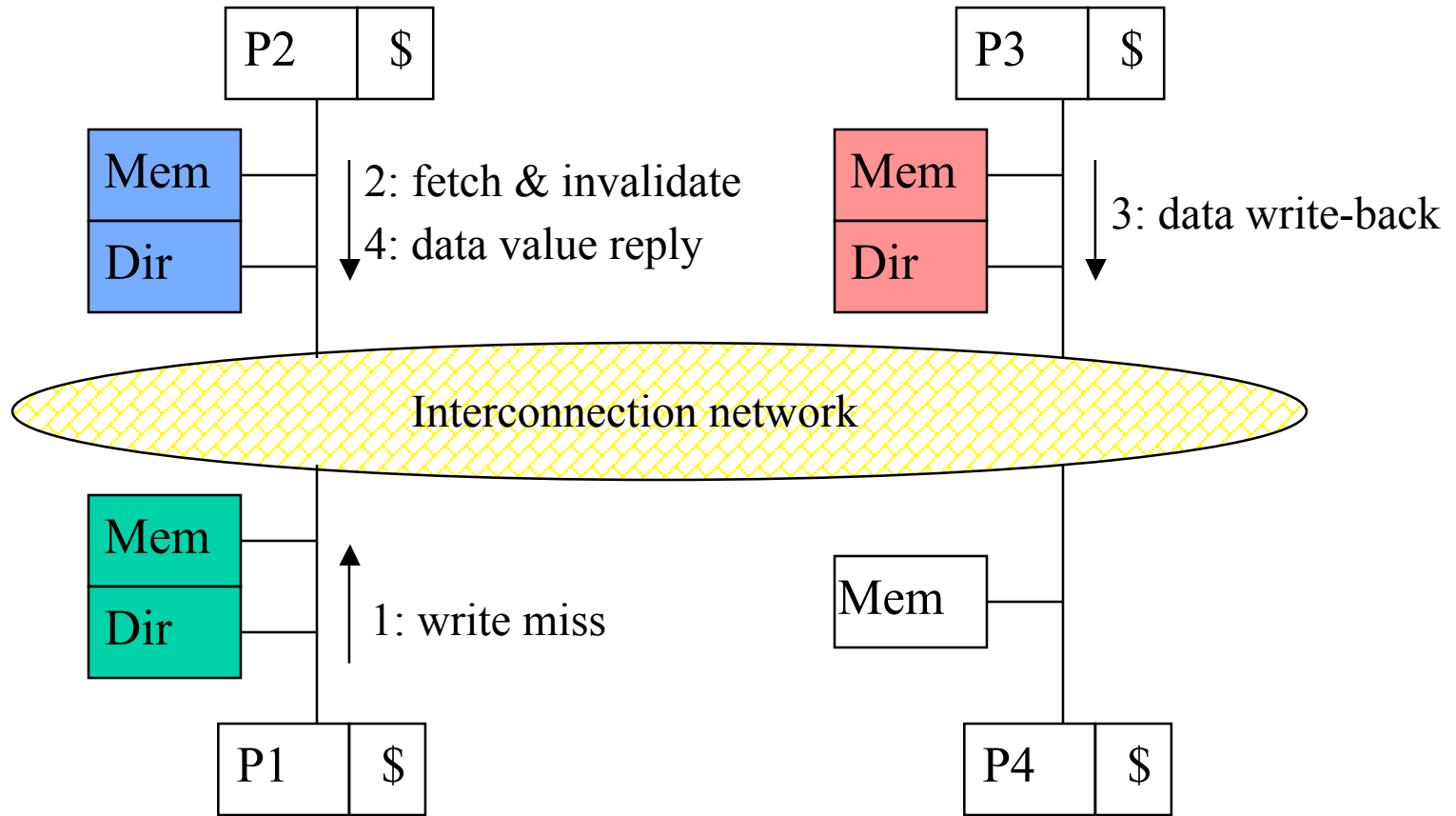
Read Miss for an Uncached Block



Read Miss for an Exclusive, Remote Block



Write Miss for an Exclusive, Remote Block



Directory Protocol Messages

<i>Message type</i>	<i>Source</i>	<i>Destination</i>	<i>Msg Content</i>
Read miss	Local cache	Home directory	P, A
	– Processor <i>P</i> reads data at address <i>A</i> ; make <i>P</i> a read sharer and arrange to send data back		
Write miss	Local cache	Home directory	P, A
	– Processor <i>P</i> writes data at address <i>A</i> ; make <i>P</i> the exclusive owner and arrange to send data back		
Invalidate	Home directory	Remote caches	A
	– Invalidate a shared copy at address <i>A</i> .		
Fetch	Home directory	Remote cache	A
	– Fetch the block at address <i>A</i> and send it to its home directory		
Fetch/Invalidate	Home directory	Remote cache	A
	– Fetch the block at address <i>A</i> and send it to its home directory; invalidate the block in the cache		
Data value reply	Home directory	Local cache	Data
	– Return a data value from the home memory (read or write miss response)		
Data write-back	Remote cache	Home directory	A, Data
	– Write-back a data value for address <i>A</i> (invalidate response)		

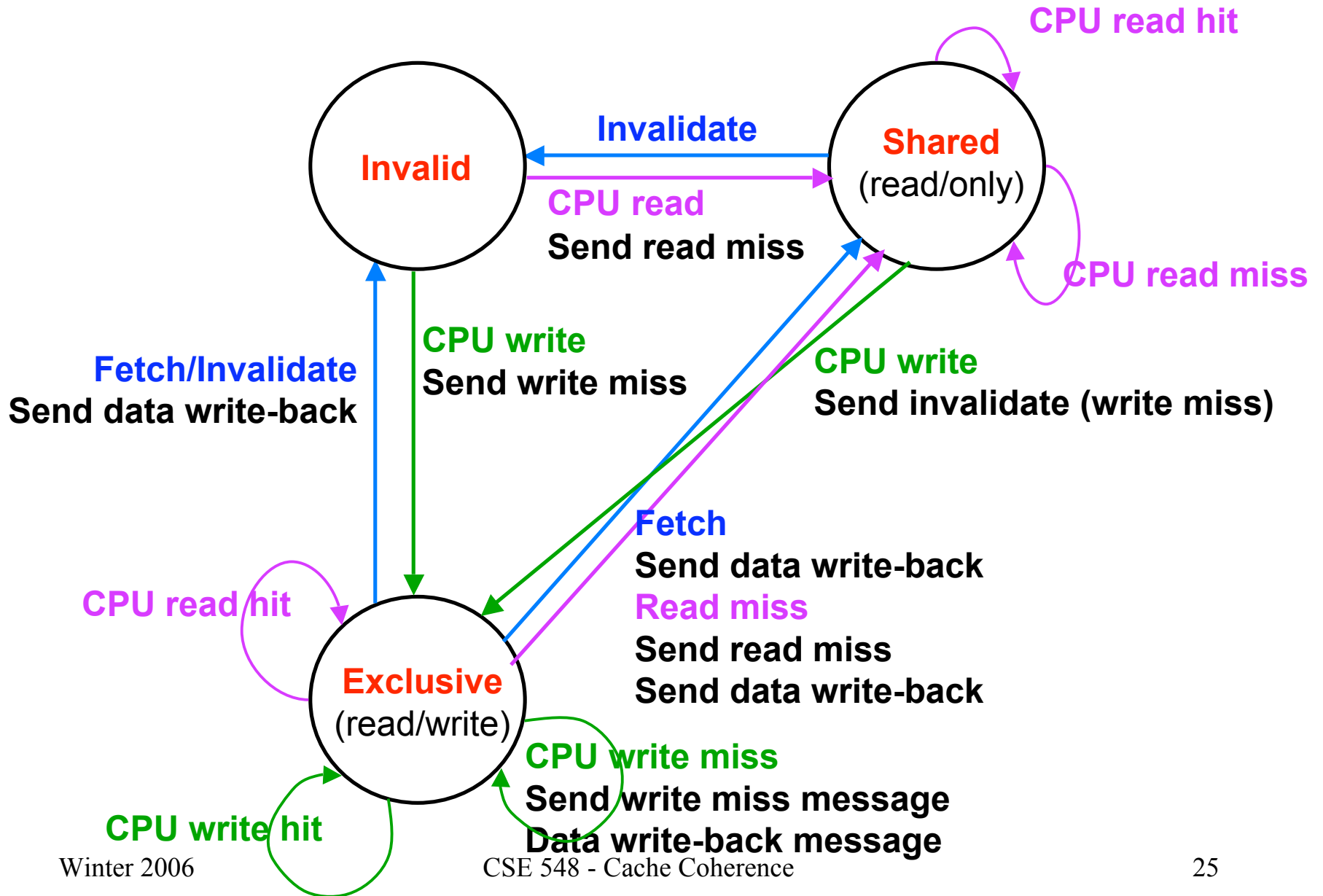
CPU FSM for a Cache Block

States identical to the snooping protocol

Transactions very similar

- read & write misses sent to home directory
- invalidate & data fetch requests to the node with the data replace
broadcasted read/write misses

CPU FSM for a Cache Block



Directory FSM for a Memory Block

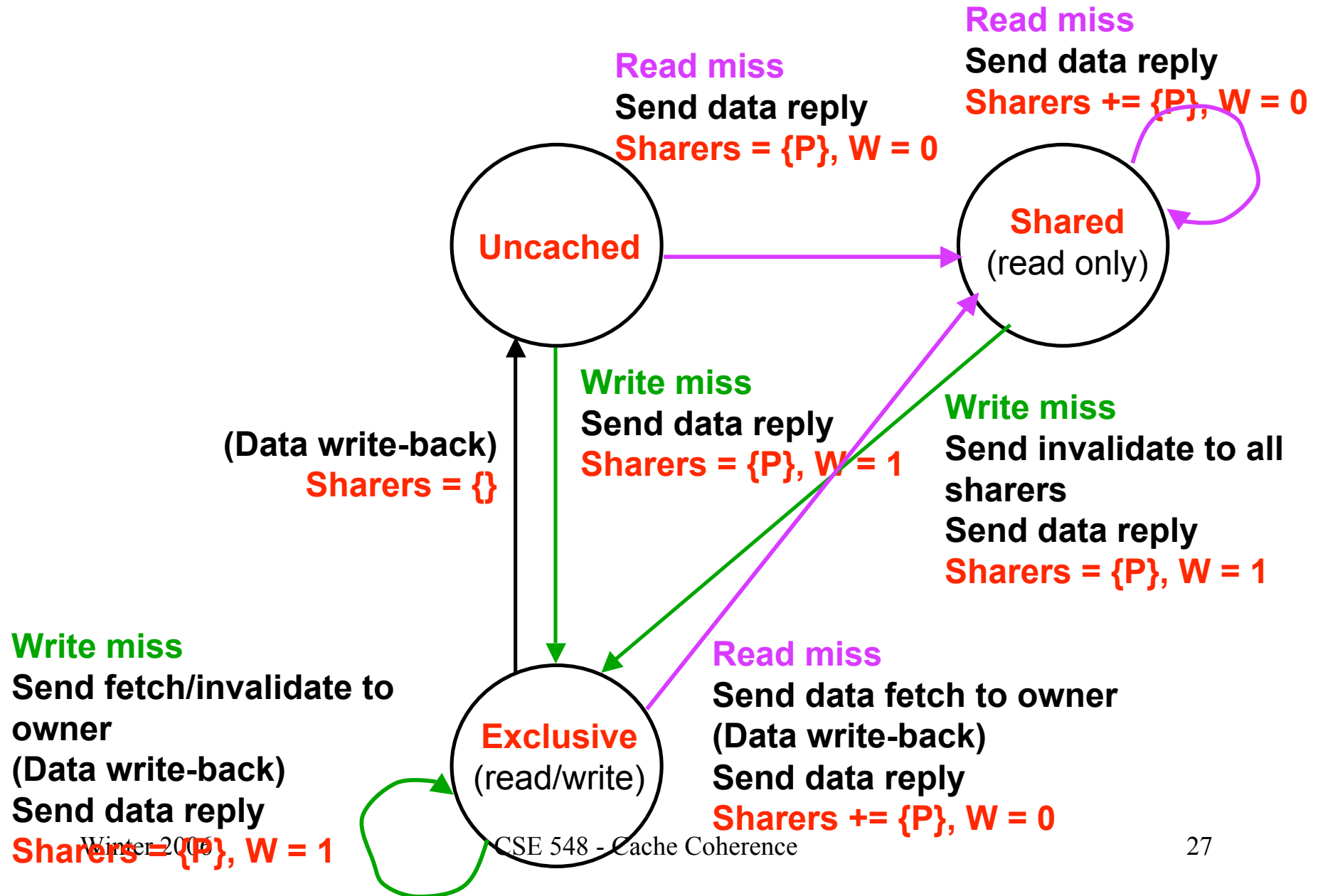
Same states and structure as for the cache block FSM

Tracks all copies of a memory block

Two state changes:

- update coherency state
- alter the number of sharers in the sharing set

Directory FSM for a Memory Block

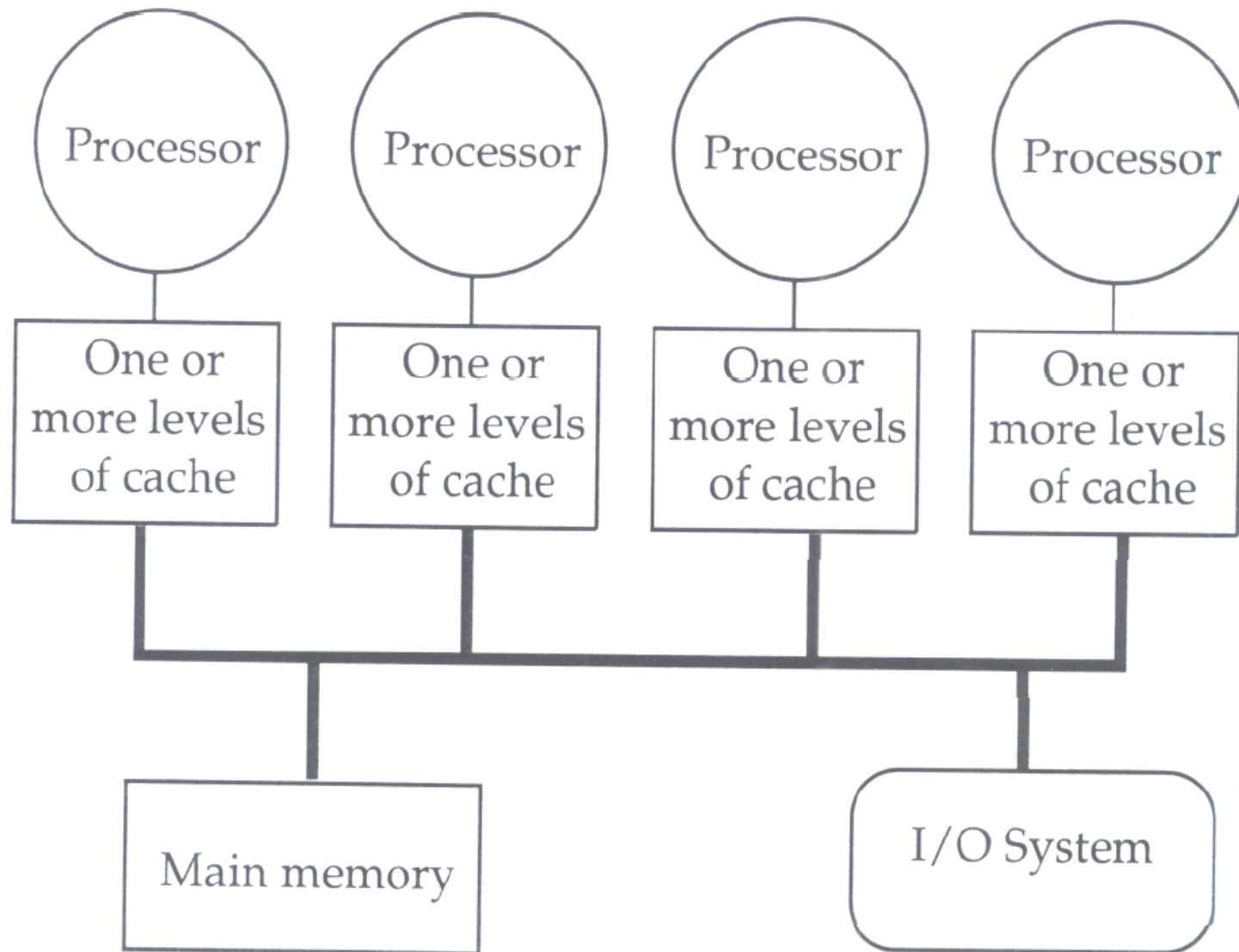


False Sharing

Processors read & write to *different* words in a shared cache block

- cache coherency is maintained on a cache block basis
 - processes share cache blocks, not data
 - block ownership bounces between processor caches

A Low-end MP



False Sharing

Impact aggravated by:

- block size: why?
- cache size: why?
- large miss penalties: why?

Reduced by:

- coherency protocols (state per subblock)
 - let cache blocks become incoherent as long as there is only false sharing
 - make them coherent if any processor true shares
- compiler optimizations (group & transpose, cache block padding)
- cache-conscious programming wrt initial data structure layout