Cache Coherency

The issue:
- must guarantee that all processors see correct data despite multiple readers & writers
- in a nutshell, how to make writes by one processor show up in other processor caches

Cache coherent processors
- all reading processors must get the most current value
- most current value for an address is the last write (in program order)
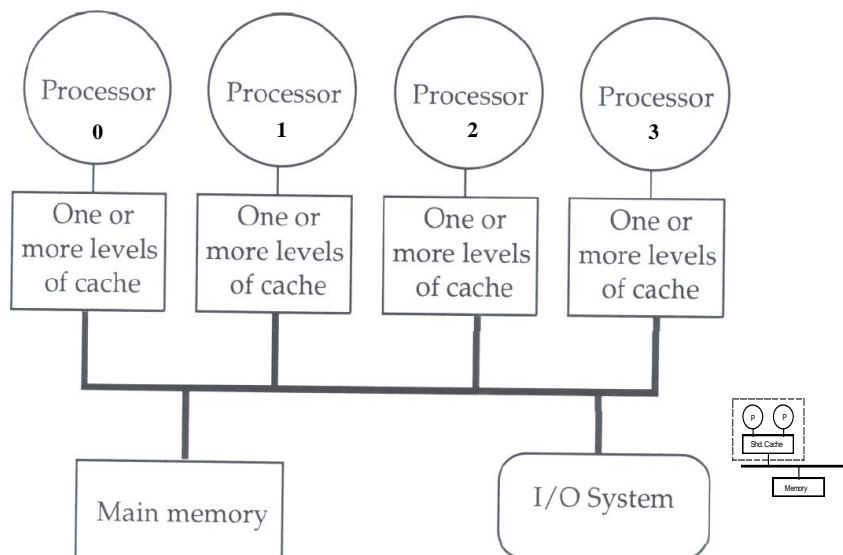
Cache coherency problem
- update from a writing processor is not known to other processors
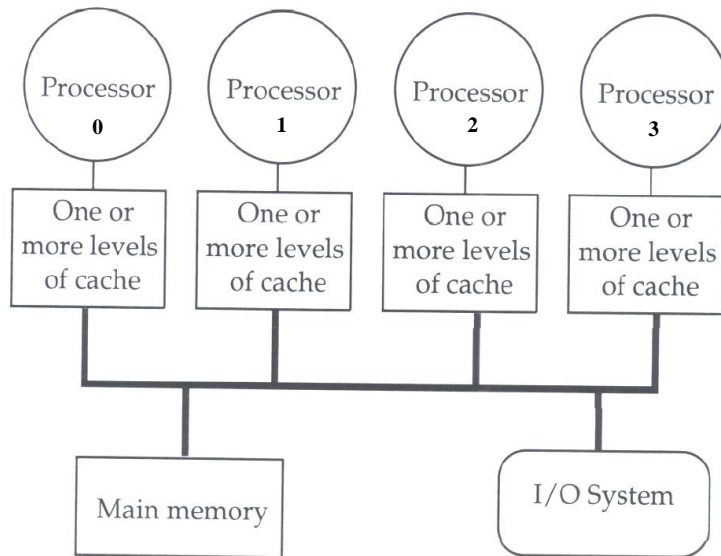
A Low-end MP

Cache Coherency

Cache coherency protocols
- (usually) hardware mechanism for maintaining cache coherency
- coherency state associated with a cache block of data
- operations on shared data change the state
  - for the processor that initiates an operation
  - for other processors that have the data of that operation in their caches
- two general types
  - snooping with a bus
  - directory with a multi-path interconnect

- In sum, hardware implementation:
  - sharing state of each cache block
  - rules for changing this state in response to memory operations
  - implemented as a state transition diagram

Write-Invalidate Protocols

- Processor obtains exclusive access for writes (becomes the "**owner**") by invalidating data in other processors' caches
- When the other processors access the data, they incur a **coherency miss** (invalidation miss)
- Owning processor provides the data in a **cache-to-cache transfer**
- good for:
  - multiple writes to same word or block by one processor
  - exploits **migratory sharing** from processor to processor (also called **processor locality**)

A Low-end MP

Cache Coherency Protocol Implementations

Snooping
- used with low-end MPs
    - few processors
    - centralized memory
    - bus-based (broadcast)
- distributed implementation: responsibility for maintaining coherence lies with each processor cache
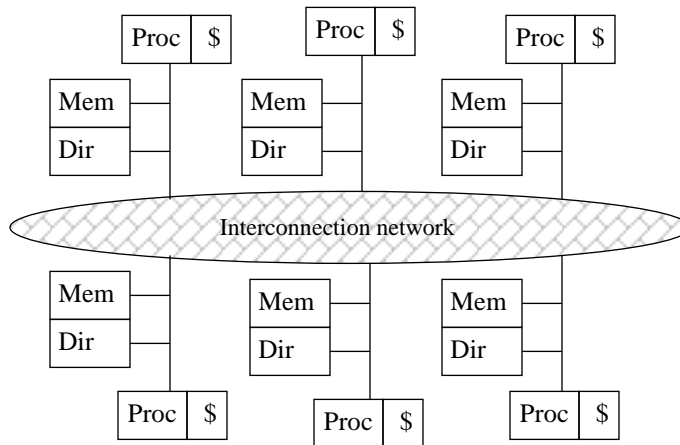
Directory-based
- used with higher-end MPs
    - more processors
    - distributed memory
    - multi-path interconnect (point-to-point)
- distributed implementation: responsibility for maintaining coherence lies with the directory
    - directory structure is distributed with the memory
    - 1 directory entry for each cache-block-size chunk of memory

A High-end MP

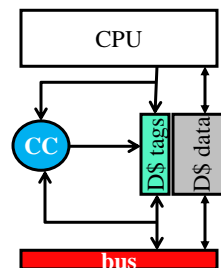
Snooping Implementation

A distributed coherency protocol
- coherency state associated with each cache block
- each cache controller (the "snoop") maintains coherency for its own cache
  - compare address on the bus with address in cache
  - response depends on coherency state

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
        - no processor can initiate another operation while any operation is in progress
        - come to a legal, stable global coherence state before any new operations
    - **split-transaction protocol** :
        - request & response are different phases of an operation
            - Improves bus throughput at the cost of operation latency
        - state values that indicate that an operation is in progress
        - no processor can initiate another operation *for a cache block* that has an operation already 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 must be 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

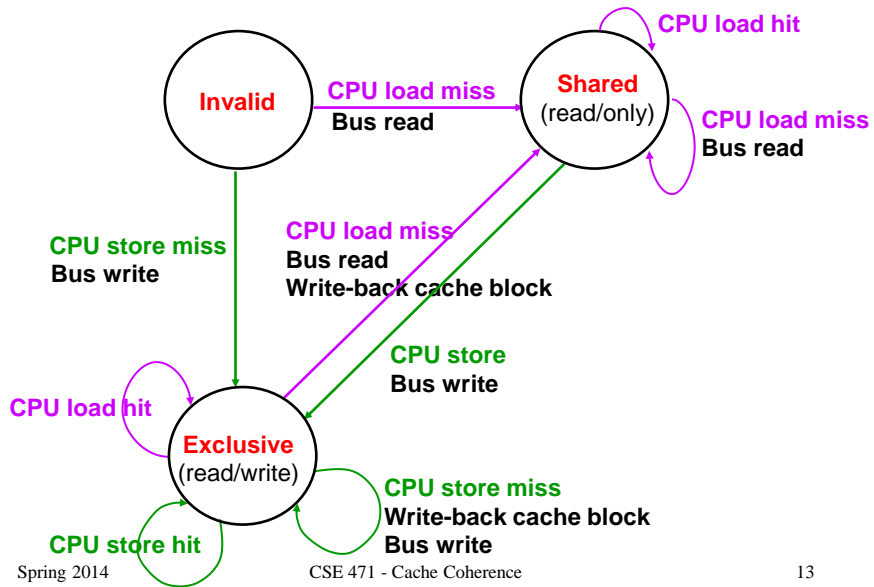Each cache block is in one of three states

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

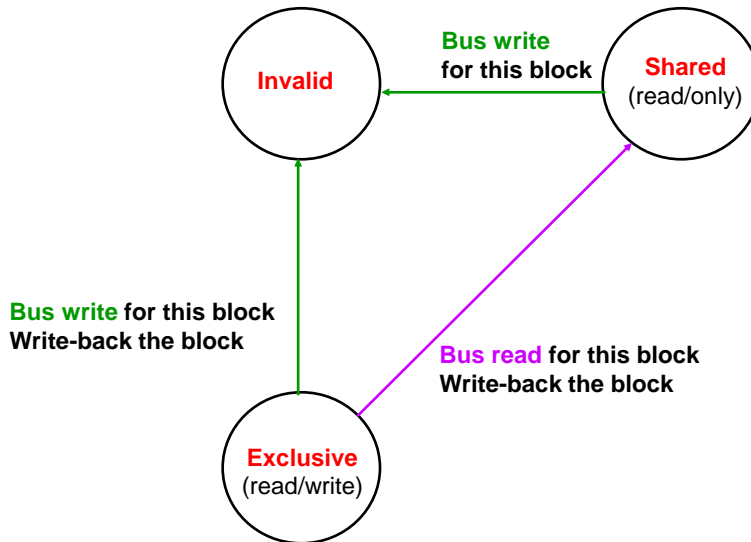## State Transitions for a Given Cache Block

State transitions caused by:

- the **requesting processor**, e.g.,
    - read/load miss (go from invalid to shared)
    - write/store miss (go from invalid to exclusive)
    - write/store to a shared block (go from shared to exclusive)
- **snoops of other caches**, e.g.,
    - read/load miss by P1 makes P2's owned block change from exclusive to shared
    - write/store miss by P1 makes P2's owned block change from exclusive to invalid

State Machine (CPU side)

Invalid → (CPU load miss / Bus read) → Shared (read/only)

Shared (read/only):
- CPU load hit (self loop)
- CPU load miss / Bus read (self loop)

Invalid → (CPU store miss / Bus write) → Exclusive (read/write)

Exclusive (read/write) → (CPU load miss / Bus read / Write-back cache block) → Shared (read/only)

Shared (read/only) → (CPU store / Bus write) → Exclusive (read/write)

Exclusive (read/write):
- CPU load hit (self loop)
- CPU store hit (self loop)
- CPU store miss / Write-back cache block / Bus write (self loop)

State Machine (Bus side: the snoop)

Shared (read/only) → (Bus write for this block) → Invalid

Exclusive (read/write) → (Bus write for this block / Write-back the block) → Invalid

Exclusive (read/write) → (Bus read for this block / Write-back the block) → Shared (read/only)
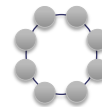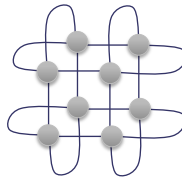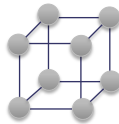
Scalable Cache Coherence

Simple, but is it scalable?

- one operation at a time
- snooping requires broadcasting all operations
- fine for 2 or 4 processors

Alternatives:

- multiple operations at a time
- point-to-point communication (most snooping results in no action)
- hundreds of processors

Directory Implementation

Distributed memory machine

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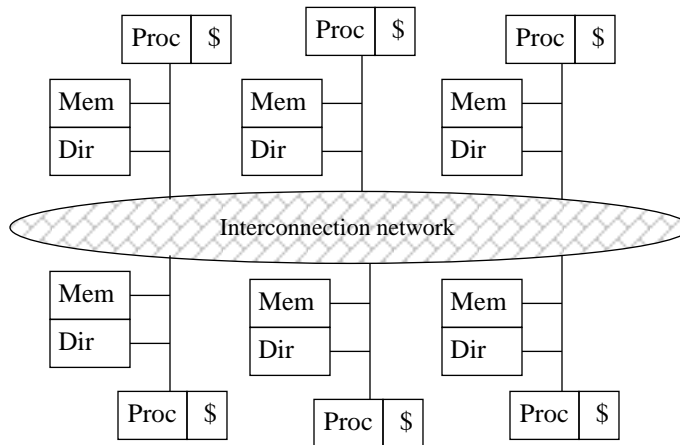   - **NUMA** (non-uniform memory access) machines

A High-end MP

Proc | $

Proc | $

Proc | $

| Mem |
| Dir |

| Mem |
| Dir |

| Mem |
| Dir |

Interconnection network

| Mem |
| Dir |

| Mem |
| Dir |

| Mem |
| Dir |

Proc | $

Proc | $

Proc | $

Directory Implementation

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

- each directory tracks the coherence state of the units in its memory & updates it
  - **uncached** (invalid in snooping):
    - no processor has the data cached & memory is up-to-date
  - **shared**:
    - at least 1 processor has the data cached & memory is up-to-date
    - block can be read by any processor
  - **exclusive**:
    - only 1 processor (the owner) has the data cached & memory is stale
    - only that processor can write to it
- directory tracks which processors share its memory blocks
  - vector of presence bits (1/processor) to indicate which processor(s) has cached the data
  - dirty bit to indicate if exclusive

Directory Implementation

Different nodes have different uses when maintaining coherency
- **local** node: where the memory request initiated
- **home** node: where the memory location of an address resides (and cached data may be there too)
- **remote** node: an alternate location for the data, if a processor has previously requested, cached & updated it

In satisfying a memory request:
- local node is the initiator; home node is identified by the data memory address; a directory knows who the remote node is
- messages sent between the different nodes in point-to-point communication
- messages get explicit replies
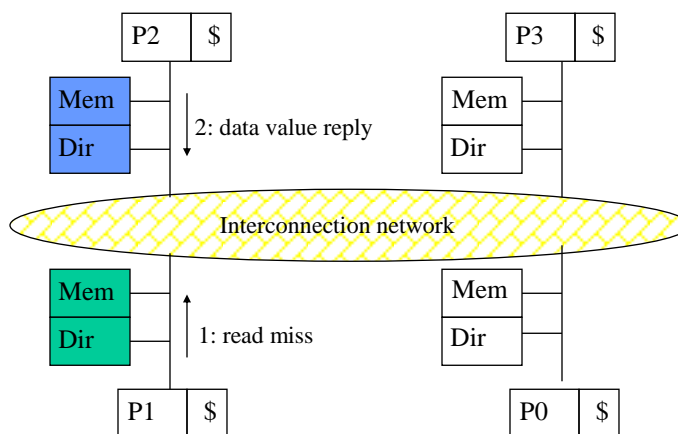
Some simplifying assumptions for using the protocol
- processor blocks until the access is complete
- messages are processed in the order received

Read Miss for an Uncached Block

Read Miss for an Exclusive, Remote Block

P2 $	P3 $
Mem / Dir	Mem / Dir
2: fetch / 4: data value reply	3: data write-back

Interconnection network

Mem / Dir	Mem / Dir
1: read miss	
P1 $	P0 $

Write Miss for an Exclusive, Remote Block

P2 $	P3 $
Mem / Dir	Mem / Dir
2: fetch & invalidate / 4: data value reply	3: data write-back

Interconnection network

Mem / Dir	Mem / Dir
1: write miss	
P1 $	P0 $
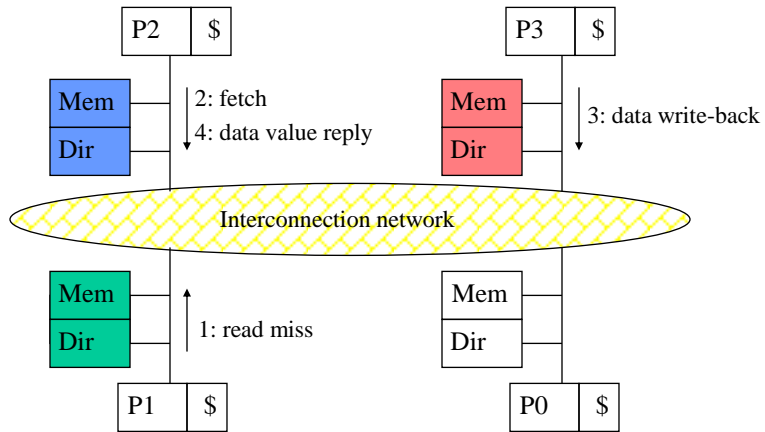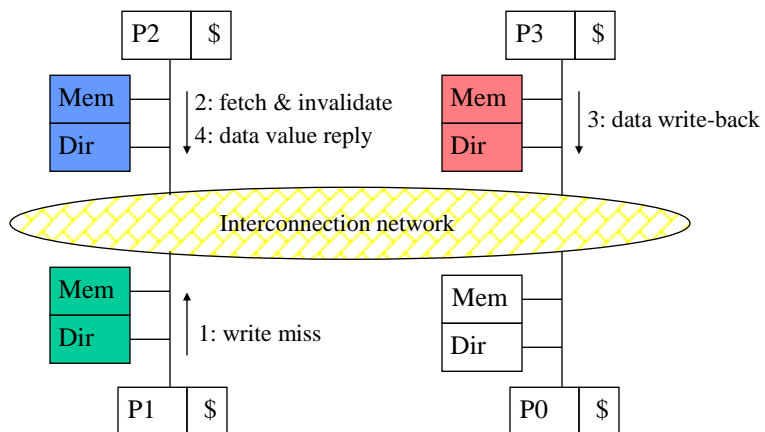
Directory Protocol Messages

Message type	Source	Destination	Message Content
Read miss	Local cache	Home directory	P, A
	– Processor P reads data at address A;		
	make P a read sharer and arrange to send data back		
Write miss	Local cache	Home directory	P, A
	– Processor P writes data at address A;		
	make P the exclusive owner and arrange to send data back		
Invalidate	Home directory	Remote caches	A
	– Invalidate a shared copy at address A.		
Fetch	Home directory	Remote cache	A
	– Fetch the block at address A and send it to its home directory		
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	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 A (invalidate response)		

Evaluating the Performance of Directory Schemes

Greater bandwidth capability
- multiple paths
- not contacting processors not involved in the memory operation

Longer operation latency
- extra hops
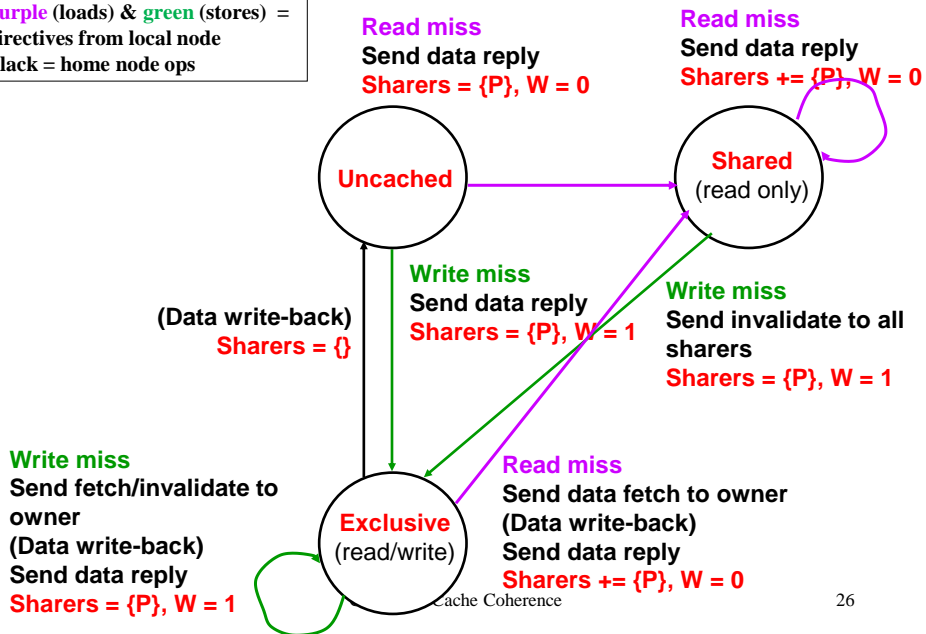- ack'ing

Directory FSM for a Memory Block

Tracks all copies of a memory block

Makes two state changes:

- update coherency state (same as for snooping protocol)
- alter the number of sharers in the sharing set

Directory FSM for a Memory Block (Home)

Purple (loads) & green (stores) = directives from local node
Black = home node ops

Read miss
Send data reply
Sharers = {P}, W = 0

Read miss
Send data reply
Sharers += {P}, W = 0

Uncached

Shared
(read only)

Write miss
Send data reply
Sharers = {P}, W = 1

(Data write-back)
Sharers = {}

Write miss
Send invalidate to all sharers
Sharers = {P}, W = 1

Write miss
Send fetch/invalidate to owner
(Data write-back)
Send data reply
Sharers = {P}, W = 1

Read miss
Send data fetch to owner
(Data write-back)
Send data reply
Sharers += {P}, W = 0

Exclusive
(read/write)
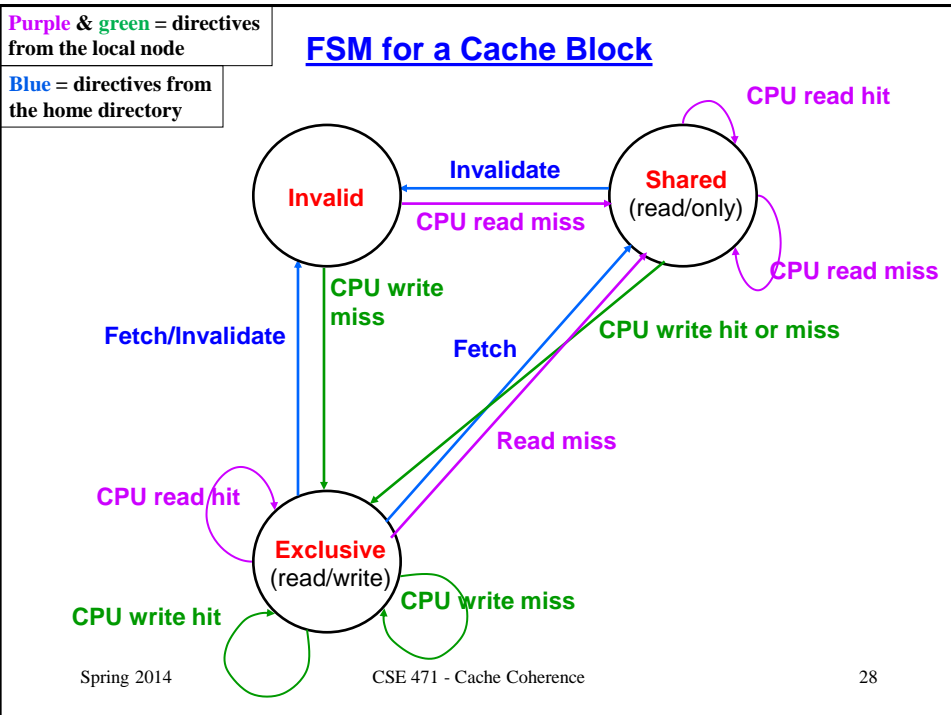
Cache Coherence   26

13

CPU FSM for a Cache Block

Same coherency states as for the directory FSM

Transactions very similar to snooping implementations, except that they are point-to-point

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

---

FSM for a Cache Block

Purple & green = directives from the local node

Blue = directives from the home directory

14

False Sharing

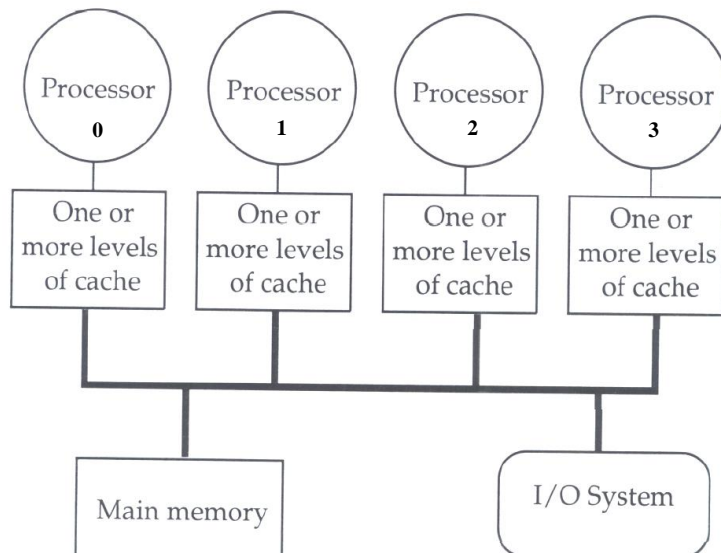Processors read & write to *different* words in a shared cache block
- cache coherency is maintained on a cache block basis
    - processes share cache blocks, not data
    - block ownership bounces between processor caches

A Low-end MP

False Sharing

Impact aggravated by:
- larger block size: why?
- larger cache size: why?
- large miss penalties: why?

Reduced by:
- coherency protocols (coherency 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

Important Issues

Cache coherency:
- its definition
- the hardware support
- write-invalidate protocols
  - how bus-based protocols work
  - how directories work
- how coherency protocols match or take advantage of the MP design

Adding to our knowledge:
- a 4th type of miss (coherency misses)
- a 3rd locality (processor)
- a 2nd application of snooping (bus-based coherency protocol)
- a 2nd use of sub-block placement (eliminate costs of false sharing)
- a 3rd latency vs. throughput trade-off

Important Issues

Anything in red or green:

- 2 bus protocols
- inclusion property
- UMA vs. NUMA
- role of local, home, remote nodes
- bus vs. multipath
- snooping vs. directory
- snooping in a coherency protocol vs. snooping in Tomasulo's algorithm
- false sharing: why it occurs, what makes it worse, how to fix it

Apply What You Know

A 4th coherency state:

- what triggers state transitions
- what are the state changes, given a sequence of memory operations

A protocol that isn't based on invalidations:

- what triggers state transitions
- what are the state changes, given a sequence of memory operations

Apply What You Know

Example:

Assume you have a 4-state, write-invalidate protocol, in which three of the states are those used in the baseline 3-state protocol we studied in class and the fourth state is a new one, called *private clean*. A private clean state means that there is only one cached copy of the data, and that it is a read-only copy (i.e., it has the same value as its backup in memory). Using this new 4-state coherency protocol, fill in the state values for a single cache block in each of the processors (P0, P1, P2), for each of the memory operations listed in the first column. Assume the multiprocessor is bus-based.

Operations	P0	P1	P2
Initially	invalid	invalid	invalid
P1: loads B			
P2: loads B			
P0: stores B			
P1: loads B			
P1: stores B			