

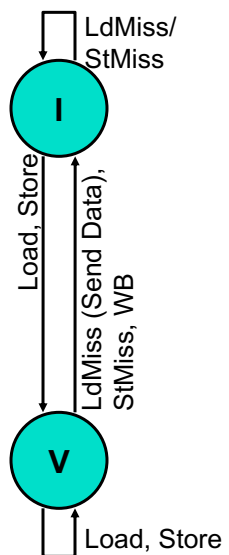
Review



- ❑ Last Class:
 - Thread-level parallelism
 - Multicore/manycore processors
- ❑ Today's class:
 - Cache coherence continued.
- ❑ Announcement and reminder
 - Midterm grade released.

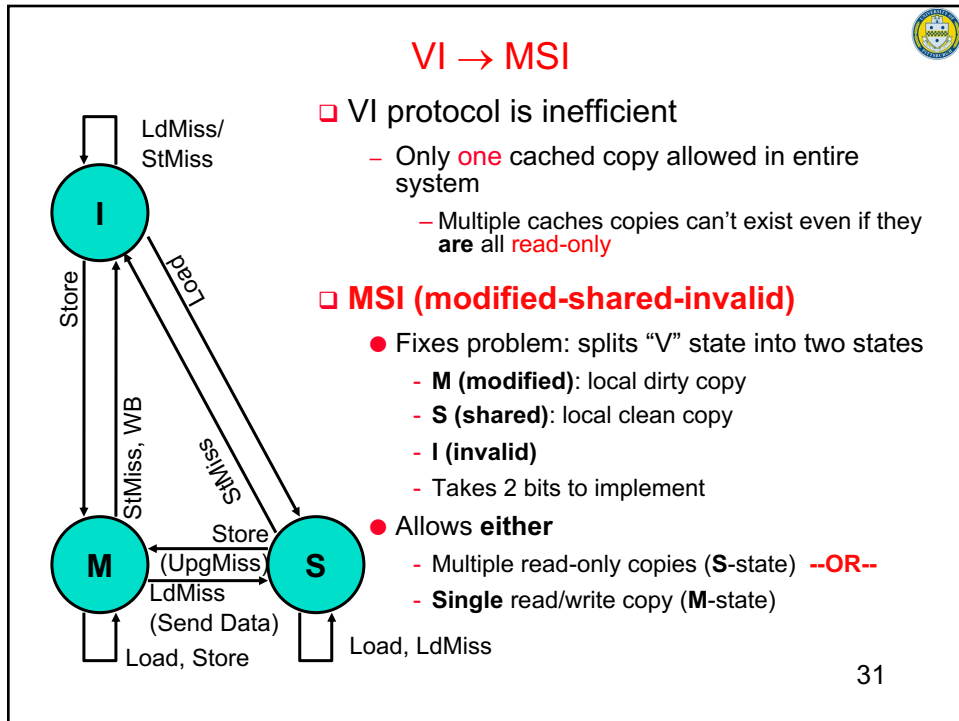
29


Review: VI (MI) coherence protocol



- ❑ Each cacheline has its own state machine
- ❑ VI (valid-invalid) protocol:
 - Two states (per block in cache)
 - V (valid): have block
 - I (invalid): don't have block
 - + Can implement with one bit (the valid bit)
- ❑ Protocol diagram (left)
 - If anyone **else** wants to read/write block
 - Give it up: transition to I state
 - WriteBack (WB) if your own copy is dirty
- ❑ This is an **write-invalidate** protocol
- ❑ **Write-update** protocol: copy data (write-through), don't invalidate
 - Sounds good, but wastes a lot of bus bandwidth

30





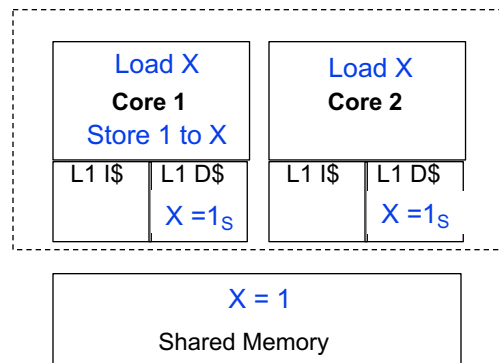
MSI protocol state transition table

State	<i>From This Core</i>		<i>From Other Cores</i>	
	Load	Store	Load Miss	Store Miss
Invalid (I)	Miss → S	Miss → M	---	---
Shared (S)	Hit	Upg Miss → M	---	→ I
Modified (M)	Hit	Hit	Send Data → S	Send Data → I

❑ M → S transition on a LdMiss also updates memory (Sends Data) since the block is Dirty

32

Example of MSI Snooping Protocol



LdMiss seen by Core 1 (no action required)

Core 1 marks X as modified and sends out an UpgMiss

StMiss (UpgMiss) seen by Core 2 so Core 2 invalidates its copy

LdMiss seen by Core 1, so Core 1 does a Send Data of X and marks its copy as shared

- When the second miss by Core 2 occurs, Core 1 responds with the value canceling the response from the shared memory (and also updating the value of X in the shared memory and marking its own copy as shared)

33

Other Coherence Protocols



- Another write-invalidate protocol used in the Pentium 4 (and many other processors) is **MESI** with four states:
 - **Modified** – same (local dirty copy)
 - **Exclusive** – only one copy of the shared data is allowed to be cached (local clean copy)
 - Since there is only one copy of the data, write hits don't need to send UpgMiss broadcast
 - **Shared** – same (multiple copies of the shared data may be cached (i.e., data permitted to be cached with more than one core))
 - **Invalid** – load miss is E if no other core is caching the data, else its S
- Other variations include MOSI, MOESI, MERSI, MESIF, Berkeley, Firefly and Dragon
- Snooping protocols (and buses) don't scale well to many-cores
 - Alternative is directory based, non-broadcast coherence protocols where a directory in memory keeps track of data ownership

34

Directory-based Coherence (§ 5.4)



- ❑ Idea: Implement a “directory” that keeps track of where each copy of a block is cached and its state in each cache (note that with snooping, the state of a block was kept only in the cache).
- ❑ Processors must consult the directory before caching blocks from memory. If block is “exclusive or modified”, then its “owner” should provide the most up-to-date copy.
- ❑ When a block in memory is updated (written), the directory is consulted to either update or invalidate other cached copies.
- ❑ Eliminates the overhead of broadcasting/snooping (bus bandwidth) – Hence, scales up with the numbers of processors that would saturate a single bus.
- ❑ latency??

35

Directory-Based Coherence

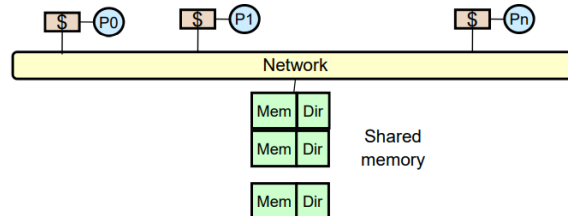


- ❑ No centralized bus and no broadcast for coherence traffic!
- ❑ Each cache/memory contains a portion of the directory
- ❑ Directory entry tells
 - Whether the block is in the cache
 - If so, where (which cache) and its status
- ❑ **Algorithm:**
 - Use physical address to go to the directory
 - Check whether the block is in the cache
 - If yes:
 - Go to the cache that contains the block
 - If no:
 - Issue a memory request {we have a cache miss}
 - The block is brought to the requesting core/cache
 - The directory is updated
- ❑ There are many ways to build a directory (e.g., distributed vs shared)

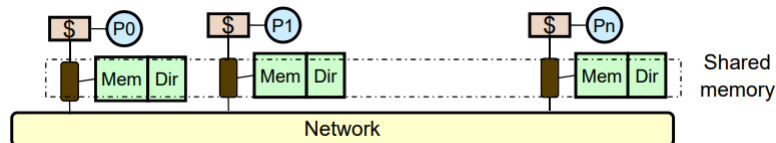
36

Directory-based Coherence

- ❑ The memory and the directory can be centralized



- ❑ Or distributed



- ❑ Alternatively, the memory may be distributed but the directory can be centralized.
- ❑ Or the memory may be centralized but the directory can be distributed.

37

Directory-based Coherence (§ 5.4)

- ❑ The location (home) of each memory block is determined by its address.
- ❑ A controller decides if access is Local or Remote
- ❑ As in snooping caches, the state of every block in every cache is tracked in that cache (modified/dirty, shared/clean, invalid).
- ❑ In addition, with each block in memory, a directory entry keeps track of where the block is cached. Accordingly, a block can be in one of the following states:
 - Uncached: no processor has it (not valid in any cache)
 - Shared/clean: cached in one or more processors and memory is up-to-date
 - modified/dirty: one processor (owner) has data; memory out-of-date

38

Review



- ❑ Last Class:
 - Directory based cache coherence
- ❑ Today's class:
 - Cache coherence continued.
- ❑ Announcement and reminder
 - Second reading assignment will be distributed today.

39

Enforcing coherence



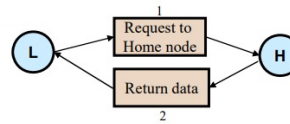
- ❑ Coherence is enforced by exchanging messages between nodes/PEs/cores.
- ❑ Three types of nodes may be involved
 - Local requestor node (L): the node that reads or write the cache block
 - Home node (H): the node that stores the block (and its directory entry) in its memory -- may be the same as L
 - Remote nodes (R): other nodes that have a cached copy of the requested block.
- ❑ When L encounters a Read Hit, it just reads the data
- ❑ When L encounters a Read Miss, it sends a message to the home node, H, of the requested block – three cases may arise:
 - The directory indicates that the block is “not cached”
 - The directory indicates that the block is “shared/clean”
 - The directory indicates that the block is “modified”

40

What happens on a read miss? (when block is invalid in local cache)

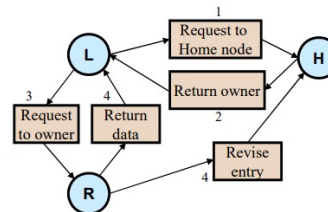
❑ Read miss (if block is shared or uncached)

- L sends request to H
- H sends the block to L
- state of block is "shared" in directory
- state of block is "shared" in L



❑ Read miss (if block is modified in another cache)

- L sends request to H
- H informs L about the block owner, R
- L requests the block from R
- R send the block to L
- L and R set the state of block to "shared"
- R informs H that it should change the state of the block to "shared"



41

What happens on a write miss? (when block is invalid in local cache)

❑ Write miss to an uncached block

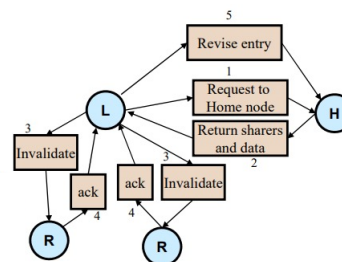
- similar to a read miss to an uncached block except that the state of the block is set to "modified"

❑ Write miss to a block that is modified in another cache

- similar to a read miss to an modified block except that the state of the block is set to "modified" in H and L and to "Invalid" in R.

❑ Write miss to a shared block

- L sends request to H
- H sets the state to "modified"
- H sends the block to L
- H sends to L the list of other sharers
- L sets the block's state to "modified"
- L sends invalidating messages to each sharers (R)
- Each R sets block's state to "invalid"



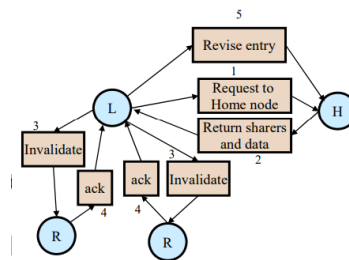
42

What happens on a write hit? (when block is shared or modified in local cache)

- ❑ If the block is “modified” in L, just write the data

- ❑ If the block is “shared” in L

- L sends a request to H to have the block as “modified”
- H sets the state to “modified”
- H informs L of the block’s other sharers
- L sets the block’s state to “modified”
- L sends invalidating messages to each sharers (R)
- R sets block’s state to “invalid”

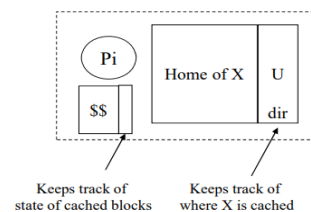
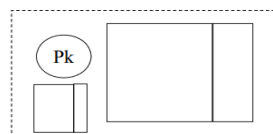
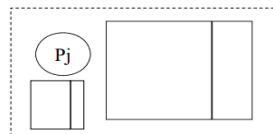


- ❑ Additional complexity:

- We need a “busy” state to handle simultaneous requests to the same block. For example, if there are two writes to the same block – it has to be serialized. Reason: order of events depends on message orders, which is non-deterministic.
- Book to read: A Primer on Memory Consistency and Cache Coherence, Second Edition. Vijay Nagarajan, Daniel J. Sorin, Mark D. Hill, David A. Wood February 2020

MSI Directory-based coherence - example

- ❑ Three cores P_i , P_j , P_k
- ❑ P_i is the home node of X (i.e., the memory address of X is in the memory partition on node P_i)
- ❑ X is uncached (U) in the home directory
- ❑ P_j reads X



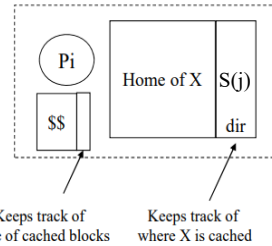
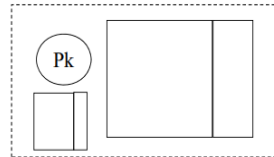
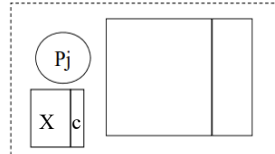
Possible scenario:

- P_j reads X

44

MSI Directory-based coherence - example

- Directory holds X is shared by Pj
- Pj has the copy of X and it is a clean copy.



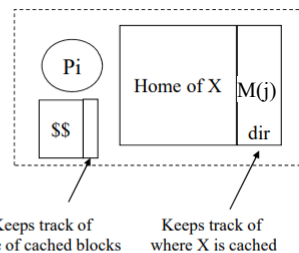
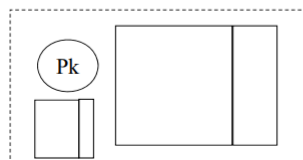
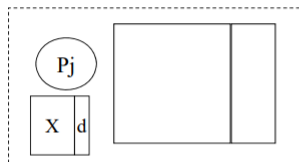
Possible scenario:

- Then Pj writes to X

45

MSI Directory-based coherence - example

- Directory holds X is modified by Pj
- Pj has the copy of X and it is a dirty copy.

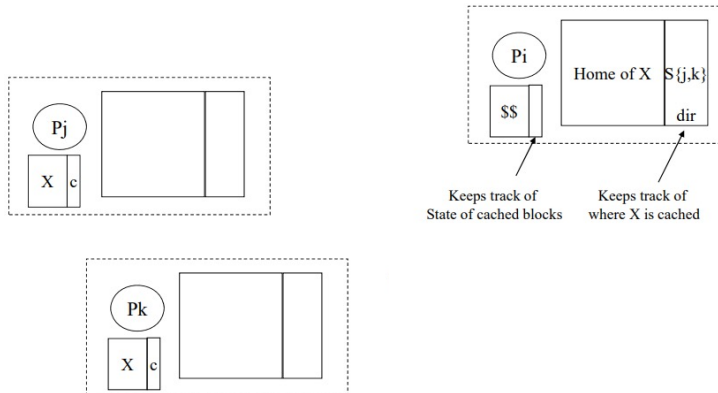


Trace the state of X if:

- Pk reads X

MSI Directory-based coherence - example

- ❑ Directory holds X is shared by Pj and Pk
- ❑ Pj and Pk has the clean copy of X.



47

The MESI protocol

- ❑ As described earlier, in MSI, a cache block can be in one of three states
 - Invalid (uncached): not in the cache (not valid in any cache)
 - Shared/clean: cached in one or more processors and memory is up-to-date
 - Modified/dirty: one processor (owner) has data; memory out-of-date
- ❑ The MESI protocol regroup the Shared and Modified states into three states:
 - Invalid (uncached): same as in MSI
 - Shared: cached in more than one processors and memory is up-to-date
 - Exclusive: one processor (owner) has data and it is clean (clean but not shared)
 - Modified: one processor (owner) has data, but it is dirty
- ❑ If MESI is implemented using a directory, then the information kept for each block in the directory is the same as the three state protocol:
 - Shared in MESI = shared/clean but more than one sharer
 - Exclusive in MESI = shared/clean but only one sharer
 - Modified in MESI = Modified/dirty
- ❑ However, at each cached copy, a distinction is made between shared, exclusive and modified (rather than only shared and modified).

48

The MESI protocol



- ❑ On a read miss (local block is invalid), load the block and change its state to
 - “exclusive” if it was uncached in memory
 - “shared” if it was already shared, modified or exclusive
 - if it was modified, the owner will send you a clean copy
 - if it was modified or exclusive, the previous owner will change the state of the block to “shared” in its cache.
- ❑ On a write miss: same as read miss, except set the state to “modified”
 - copies in other caches (if any) are invalidated
- ❑ On a write hit to a “modified” block, do nothing
- ❑ On a write hit to an “exclusive” block change the block to “modified”
 - no need for invalidation.
- ❑ On a write hit to a “shared” block change the block to “modified” and invalidate the other cached copies.
- ❑ When a modified block is evicted, write it back.

49

The MESI protocol



- ❑ If MESI is implemented as a snooping protocol, then the main advantage over the three state protocol is when a read to an uncached block is followed by a write to that block.
 - After the uncached block is read, it is marked “exclusive” – (need a scheme to know that it was uncached).
 - Note that, when writing to a shared block, the transaction has to be posted on the bus so that other sharers invalidate their copies.
 - But when writing to an exclusive block, there is no need to post the transaction on the bus. Hence, by distinguishing between shared and exclusive states, we can avoid bus transactions when writing on an exclusive block.
 - However, now a cache that has an “exclusive” block has to monitor the bus for any read to that block. Such a read will change the state to “shared”.
 - What if a block was shared in two caches and is evicted from one of them. Should we detect this case and set the block to Exclusive in the other cache?

50

Coherence Misses



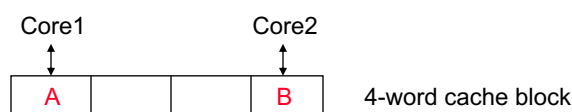
1. **True-sharing misses** arise from communication of data through cache-coherence mechanism
 - Invalidates due to write to shared word
 - Reads by another core of modified block in different cache
 - This miss would still occur if block size were 1 word
2. **False-sharing misses** when a block is invalidated because some word in the block, other than the one being read, is written into
 - Invalidation does not cause a new value to be communicated, but only causes an extra cache miss
 - Fundamental reason: coherence protocol is applied at cache line/block granularity!
 - Block is shared, but no word in block is actually shared
 - This miss would not occur if block size were 1 word

51

Block Size Effects



- ❑ Writes to one word in a multi-word block mean that the full block is invalidated
- ❑ Multi-word blocks can also result in **false sharing**: when two cores are writing to two different variables that happen to fall in the same cache block
 - False sharing increases cache miss rates



- ❑ Compilers can help reduce false sharing by allocating highly correlated data to the same cache block
 - ❑ How to know?
 - ❑ Padding? Good or bad?

52

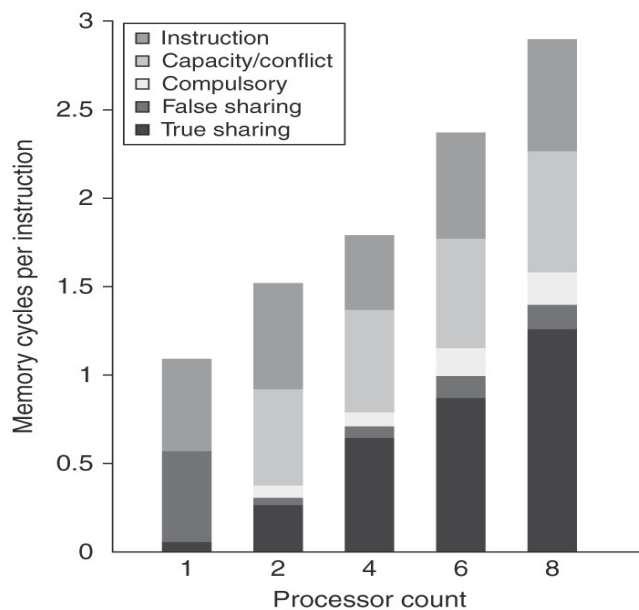
Example: True Sharing vs False Sharing

Assume d1 and d2 in same cache block.
P1 and P2 both read d1 and d2 before.

Step	P1	P2	True, False, Hit? Why?
1	Write d1		Invalidate the cache block in P2
2		Read d2	False miss; d1 irrelevant to P2
3	Write d1		Invalidate the cache block in P2
4		Write d2	Invalidate the cache block in P1
5	Read d2		True miss; they are sharing d2

53

Memory Cycle Distribution



55

