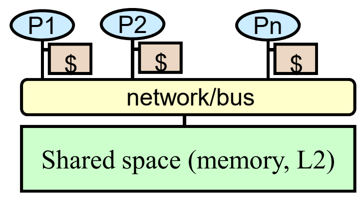
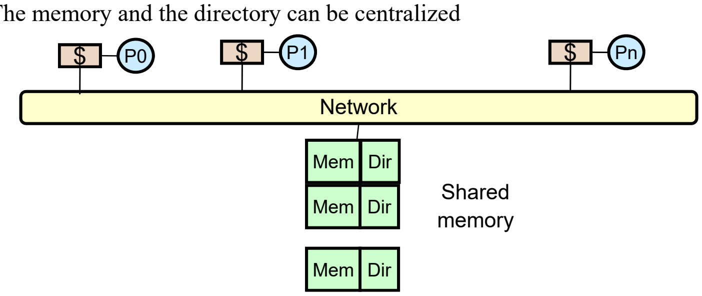
Directory-based Coherence (§ 5.4)

- **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", 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.
- Slower in terms of latency??

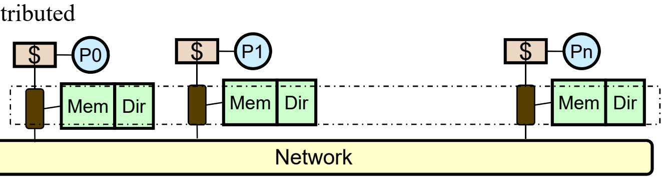


Directory-based Coherence

- The memory and the directory can be centralized



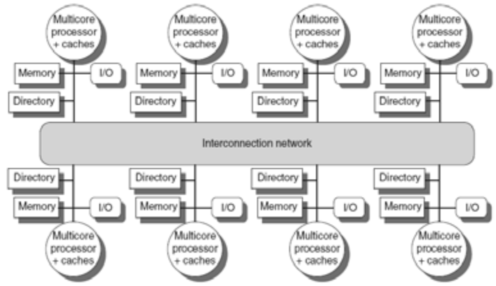
- Or distributed



- Alternatively, the memory may be distributed but the directory can be centralized.
- Or the memory may be centralized but the directory can be distributed (as we will discuss in the case of CMP with private L2 caches)

Distributed directory-based coherence

- The location (home) of each memory block is determined by its address.
- A controller decides if access is Local or Remote



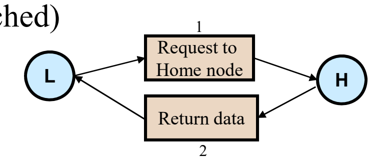
- As in snooping caches, the state of every block in every cache is tracked in that cache (exclusive/dirty, shared/clean, invalid) – to avoid the need for write through and unnecessary write back.
- In addition, with each block in memory, a directory entry keeps track of where the block is cached. Accordingly, 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
  - *Exclusive/modified/dirty*: one processor (owner) has data; memory out-of-date

Enforcing coherence

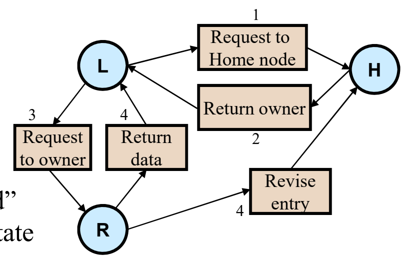
- Coherence is enforced by exchanging messages between nodes
- Three types of nodes may be involved
  - Local requestor node (L): the node that reads or write the cache block
  - Home node (H): the node that stores the block (and its directory entry) in its memory -- may be the same as L
  - Remote nodes (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 “exclusive/modified”

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

(a)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



(b) Read miss (if block is exclusive 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”

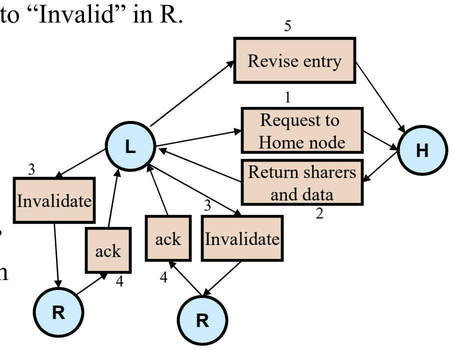


What happens on a write miss?
(when block is invalid in local cache)

(a) 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 “exclusive”

(b) Write miss to an block that is exclusive in another cache
- -- similar to a read miss to an exclusive block except that the state of the block is set to “exclusive” in H and L and to “Invalid” in R.

(c) Write miss to a shared block
- -- L sends request to H
- -- H sets the state to “exclusive”
- -- H sends the block to L
- -- H sends to L the list of other sharers
- -- L sets the block’s state to “exclusive”
- -- L sends invalidating messages to each sharers (R)
- -- Each R sets block’s state to “invalid”



What happens on a write hit?

(when block is shared or exclusive in local cache)

(a) If the block is "exclusive" in L, just write the data

(b) If the block is "shared" in L

-- L sends a request to H to have the block as "exclusive"

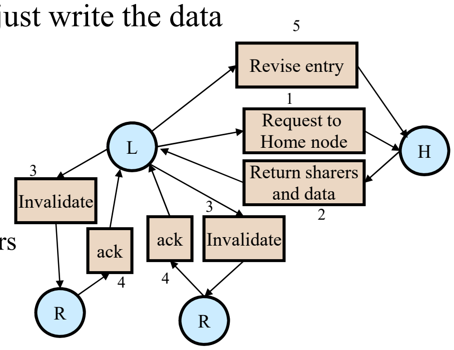
-- H sets the state to "exclusive"

-- H informs L of the block's other sharers

-- L sets the block's state to "exclusive"

-- L sends invalidating messages to each sharers (R)

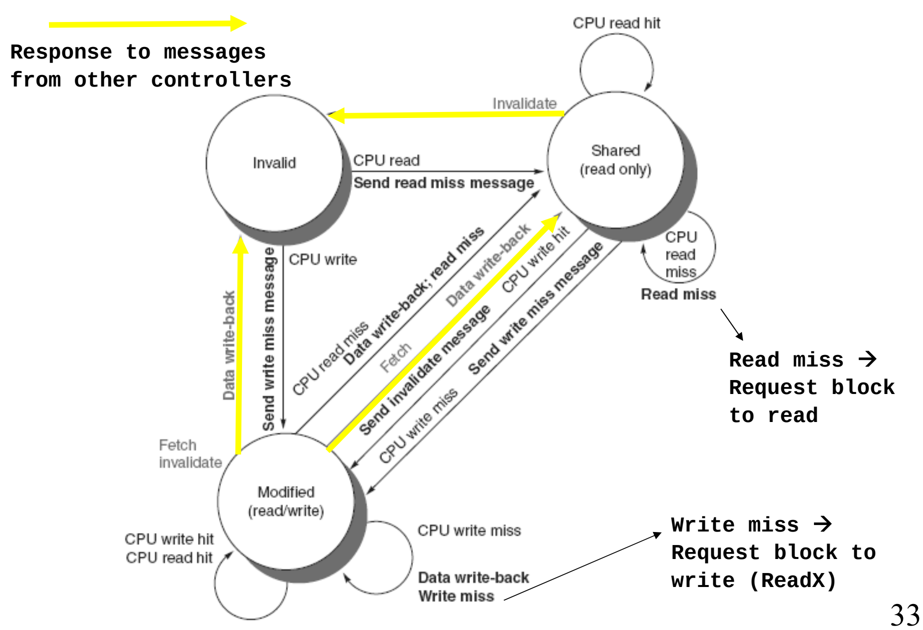
-- R sets block's state to "invalid"



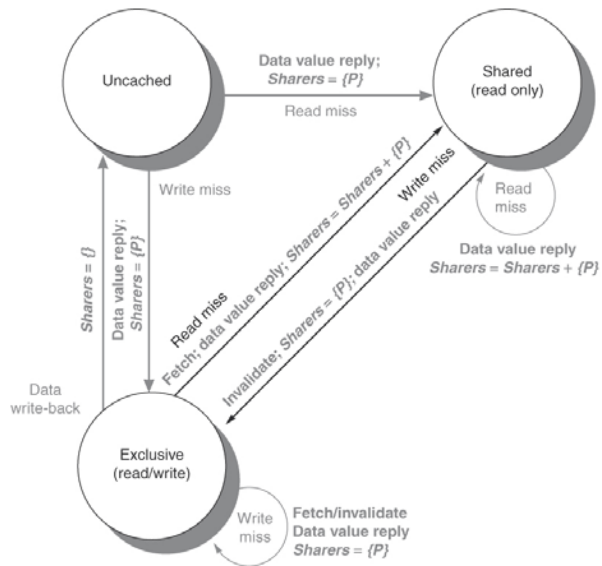
Additional complexity:

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.

The coherence protocol at a node's cache controller

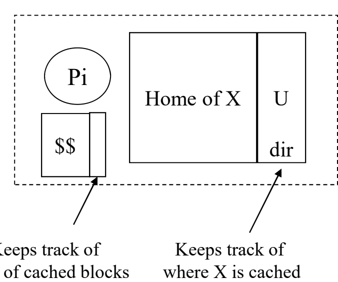


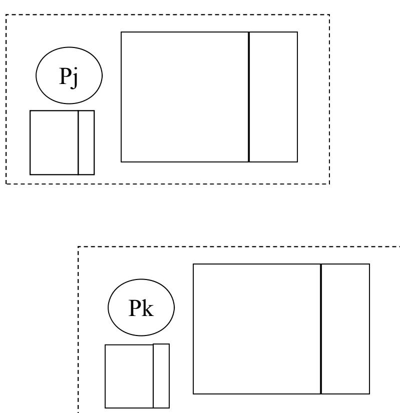
The coherence protocol
(Directory response to a coherence message)



MSI Directory-based coherence - example

Case 1:
X is in the *uncached* (U) state in home directory



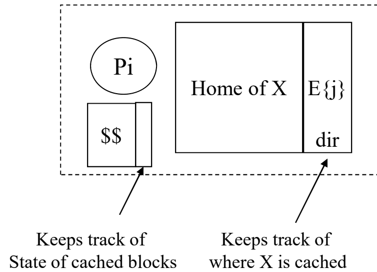


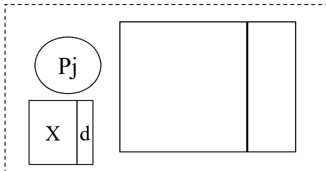
Possible scenario:

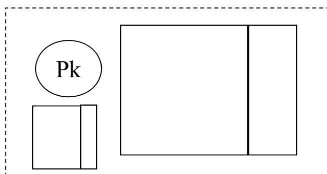
- Pj reads X
- Then Pj writes to X

MSI Directory-based coherence - example

Case 2:
X is *exclusive* (E) in home directory
and owned by Pj (dirty, d, in Pj)





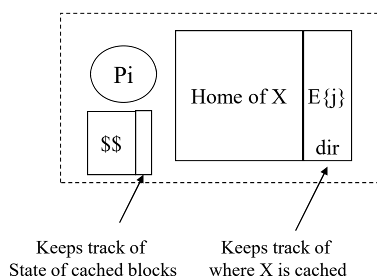


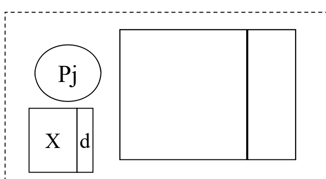
Trace the state of X if:

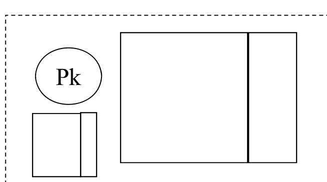
- Then Pk reads X

MSI Directory-based coherence - example

Case 3:
X is *exclusive* (E) in home directory and owned by Pj (dirty, d, in Pj)





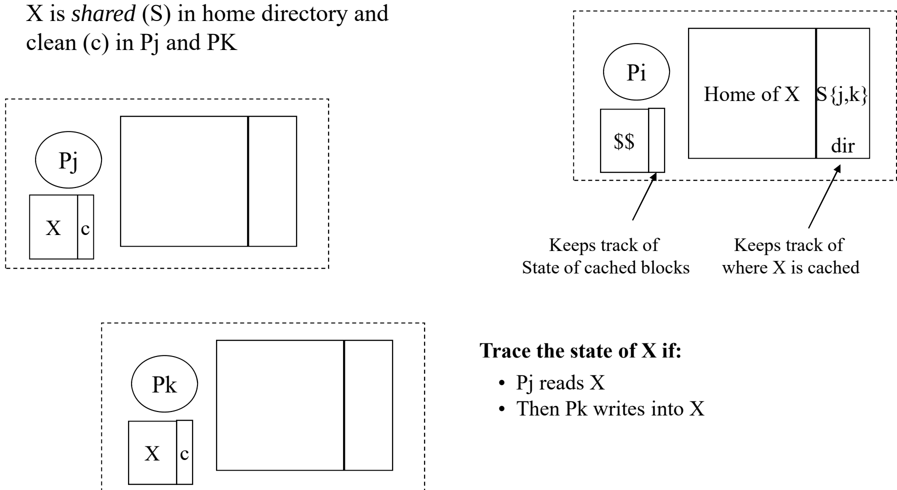


Trace the state of X if:

- Pk writes to X

MSI Directory-based coherence - example

Case 4:
X is *shared* (S) in home directory and clean (c) in Pj and PK



Trace the state of X if:

- Pj reads X
- Then Pk writes into X