Notes-07

CACHE Coherence

Informal: In a shared memory multiprocessor, the global shared memory and a local cache can become inconsistent when a processor writes to its local cache. Cache coherence techniques solve this problem.

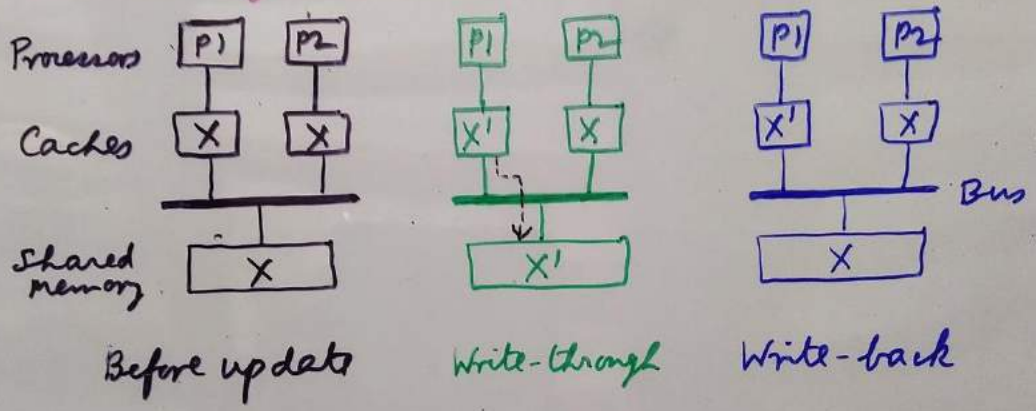
Snoopy protocols monitor bus activity to determine whether a cache has a current copy of the data.

On the other hand, directory based protocols maintain the status of each cache in a directory. This approach is used in scalable multiprocessors.

CACHE COHERENCE

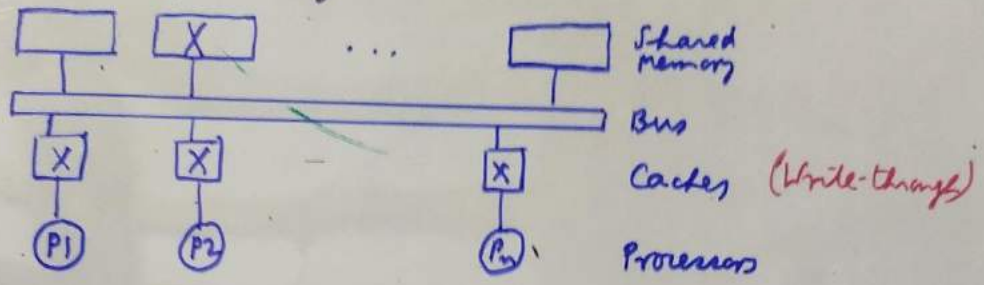
When multiple processors maintain locally cached copies of a unique shared-memory location, any local modification of the location can result in a globally inconsistent view of memory. Cache coherence schemes prevent this problem by maintaining a uniform state for each cached block of data.

Inconsistency in Data Sharing

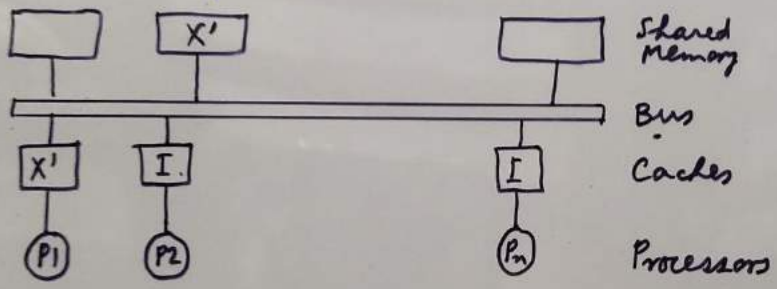


SNOOPY PROTOCOLS

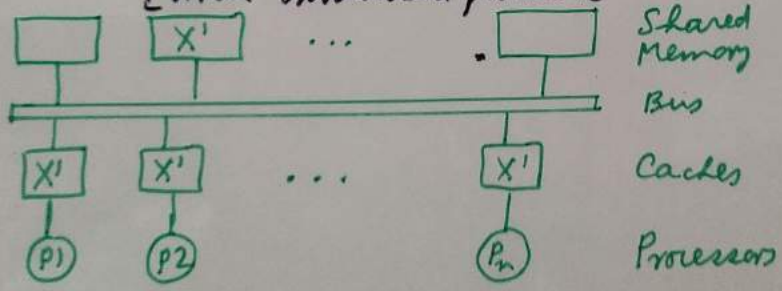
Snoopy protocols achieve data consistency among the caches and shared memory through a bus watching mechanism.



Consistent copies of block X are in shared memory and three processor caches



After a write-invalidate operation by P1.
[Write-invalidate protocol]



After a write-update operation by P1
[Write-update protocol]

Slide 7.2

MSI Write-Back Invalidation Protocol

Three states of a cache:

Modified (M) (also called dirty) means that only this cache has a valid copy of the block, and the copy in main memory is stale.

Shared (S) means the block is present in an unmodified state in this cache, main memory is up-to-date, and zero or more other caches may also have an up-to-date (shared) copy.

Invalid (I) means that the block is not present in cache.

Before a shared or invalid block can be written and placed in the modified state, all the other potential copies must be invalidated via a read-exclusive bus transaction.

The processor issues two types of requests: reads (PrRd) and writes (PrWr). The read or write could be to a memory block that exists in the cache or to one that does not.

MSI Protocol : Bus transactions

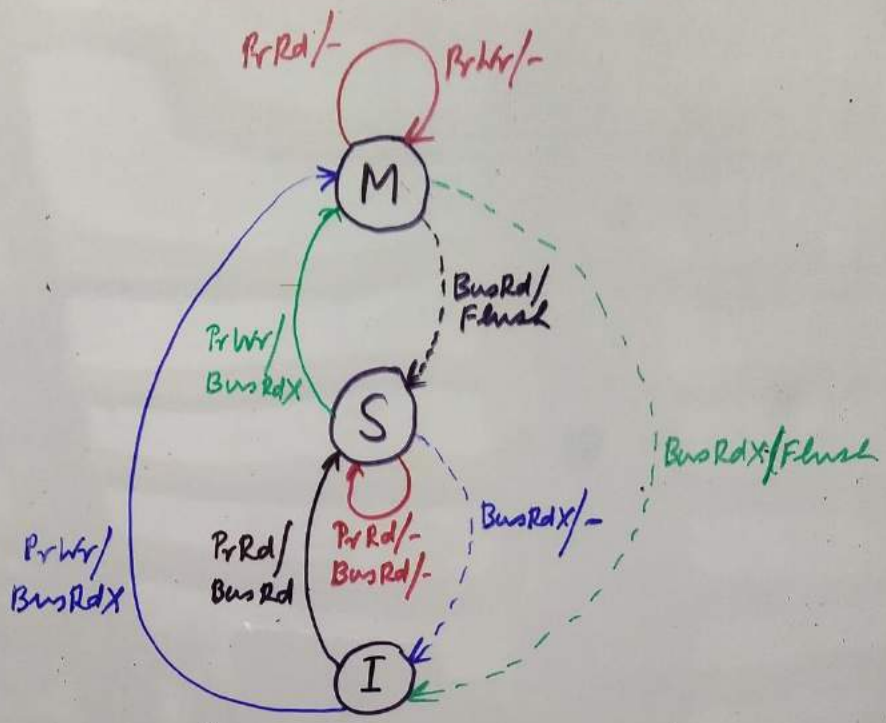
The bus allows the following transactions:

- Bus Read (BusRd): This transaction is generated by a PrRd that misses in the cache, and the processor expects a data response as a result. The cache controller puts the address on the bus and asks for a copy that it does not intend to modify. The memory system (possibly another cache) supplies the data.

- Bus Read Exclusive (BusRdX): This transaction is generated by a PrWr to a block that is either not in the cache or is in the cache but not in the modified state. The cache controller puts the address on the bus and asks for an exclusive copy that it intends to modify. The memory system (possibly another cache) supplies the data. All other caches are invalidated. Only this cache obtains the exclusive copy, the write can be performed in the cache.

- Bus Write Back (BusWB): This transaction is generated by a cache controller on a write back; the processor does not know about it and does not expect a response. The cache controller puts the address and the contents for the memory block on the bus. The main memory is updated with the latest contents.

MSI Protocol: State transitions



Notation A/B: If the controller observes the event A from the processor side or the bus side, then in addition to the state change, it generates the bus transaction or action B.

"—" means null action.

Transitions due to observed bus transactions are shown in dashed arcs, while those due to local processor actions are shown in bold arcs.

Replacements and the write-backs they may cause are not shown in the diagram.

Slide 7.5

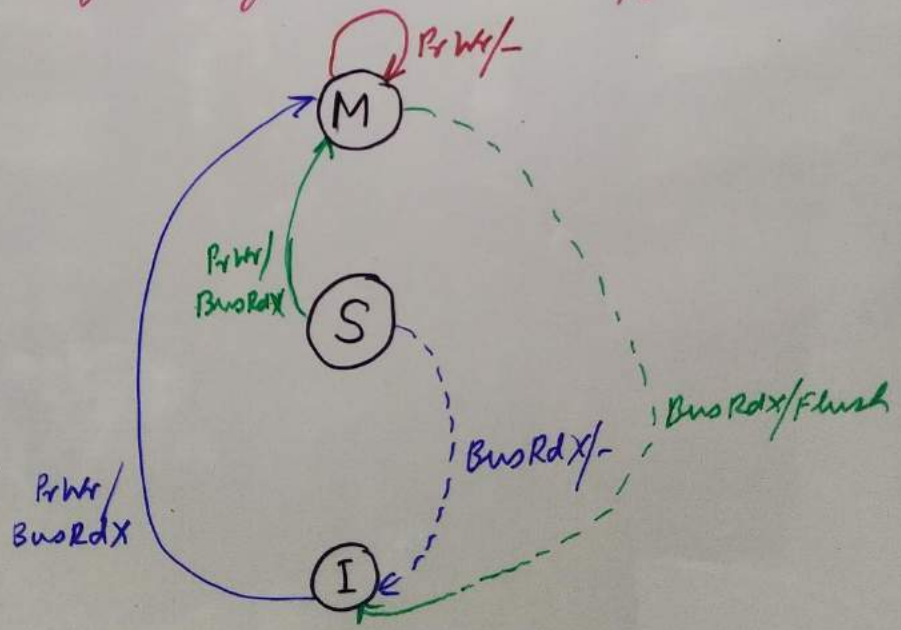
MSI State transitions (cont'd.)

A processor read to a block that is invalid (or not present) causes a BusRd transaction to service the miss. The newly loaded block is promoted, moved up in the state diagram, from invalid to the shared state in the requesting cache, whether or not any other cache holds a copy. Any other caches with the block in the shared state observe the BusRd but take no special action, allowing main memory to respond with the data. However, if a cache has the block in the modified state (there can only be one) and it observes a BusRd transaction on the bus, then it must get involved in the transaction since the copy in main memory is stale. This cache flushes the data onto the bus, in lieu of memory, and demotes its copy of the block to the shared state. The memory and the requesting cache both pick up the block. This can be accomplished either by a direct cache-to-cache transfer across the bus during this BusRd transaction or by signaling an error on the BusRd transaction and generating a write transaction to update memory. In the latter case the original cache will eventually retry its request and obtain the block from memory.

Slide 7.6

MSI State transitions (contd.)

Writing into an invalid block is a write miss, which is serviced by first loading the entire block and then modifying the desired bytes within it. The write miss generates a read-exclusive bus transaction, which causes all other cached copies of the block to be invalidated, thereby granting the requesting cache exclusive ownership of the block. The block of data returned by the read exclusive is promoted to the modified state, and the desired bytes are then written into it. If another cache requests exclusive access, then in response to its BusRdX transaction this block will be invalidated (demoted to the invalid state) after flushing the exclusive copy to the bus.



Slide 7.7

MESI Write-Back Invalidation Protocol

MSI : Drawback

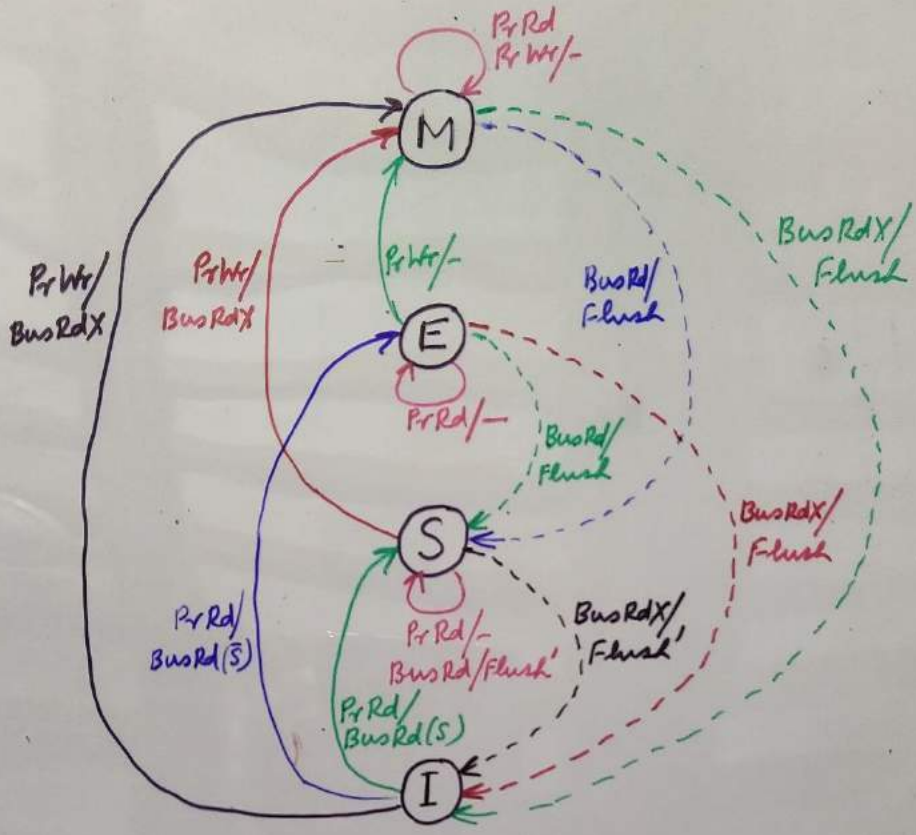
When a process reads in and modifies a data item, two bus transactions are generated:
(i) a BusRd that gets the memory block in the S state.
(ii) a BusRdX (or BusUpgr) that converts the block from S to M state. (takes place even if no other processor is sharing this data)
→ It is not known whether other processors are sharing this block; if they do, those copies must be invalidated.

MESI:

By adding a state that indicates that the block is the only (exclusive) copy but is not modified and by loading the block in this state, we can save the second transaction since the state indicates that no other processor is caching the block.

This new state, called exclusive-clean or exclusive-unowned (or even simply "exclusive"), indicates an intermediate level of binding between shared and modified. It is exclusive, so unlike the shared state, the cache can perform a write and move to the modified state without further bus transactions; but it does not imply ownership (memory has a valid copy), so unlike the modified state, the cache need not reply upon observing a request for the block.

MESI Protocol: State Transitions

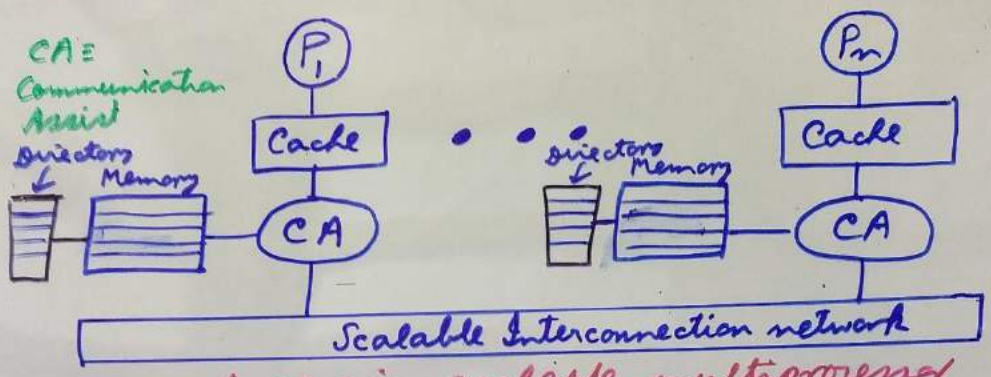


This protocol places a new requirement on the physical interconnect of the bus. An additional signal called the shared signal (S) must be available to the controllers in order to determine on a BusRd if any other cache currently holds the data. During the address phase of the bus transaction all caches determine if they contain the requested block and, if so, assert the shared signal.

Flush': A cache, rather than main memory, supplies data for BusRd and BusRdX. Flush' applies to the cache that supplies the data.

Slide 7.9

DIRECTORY-BASED CACHE-COHERENCE



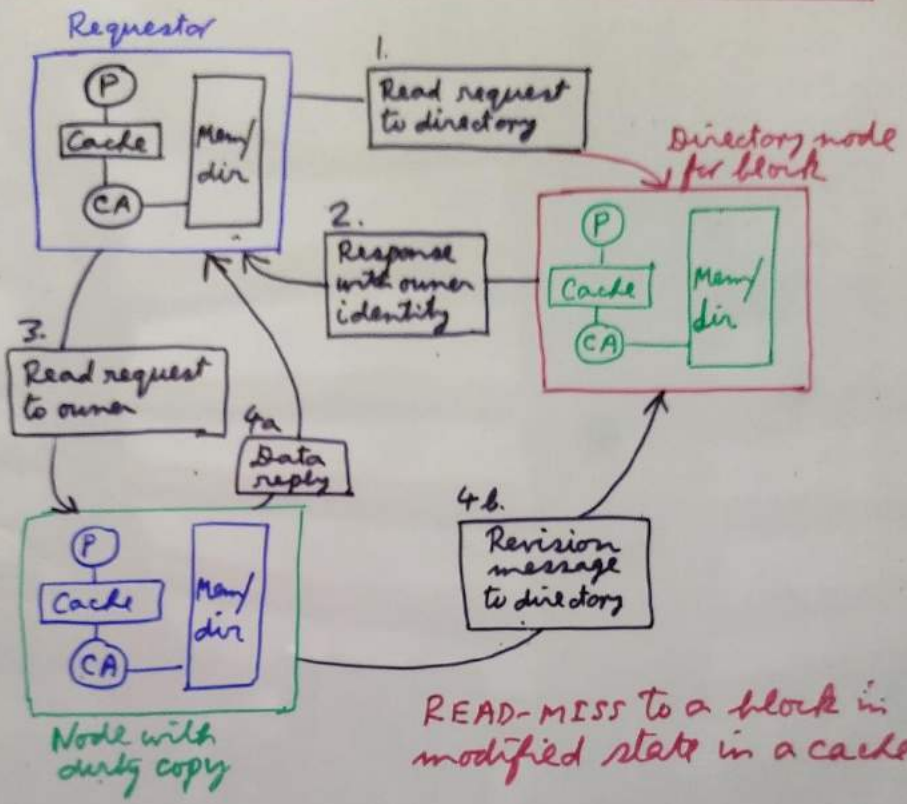
A generic scalable multiprocessor with directories

Scalable cache coherence is typically based on the concept of a directory.

Consider a simple example. Imagine that each cache-line-sized block of main memory has associated with it a record of the caches that currently contain a copy of the block and the state of the block in those caches. This record is called the directory entry for that block.

When a node incurs a cache miss, it first communicates with the directory entry for the block using point-to-point network transactions. From the directory, the node determines where the valid cached copies (if any) are and what further actions to take. It then communicates with the cached copies as necessary using additional network transactions. For example, it may obtain a modified block from another node.

OPERATION OF A SIMPLE DIRECTORY SCHEME



READ-MISS to a block in modified state in a cache.

Home node : node in whose main memory the block is allocated.

(may be the same)

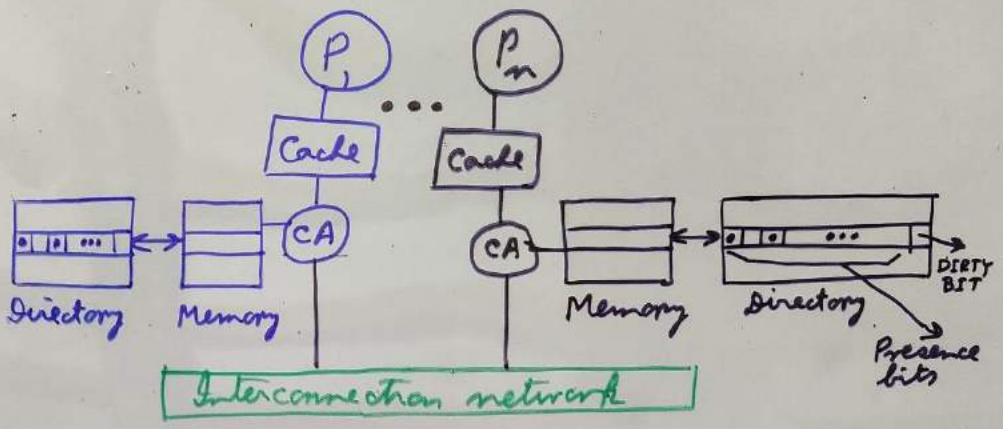
Dirty node : node that has a copy of the block in its cache in modified (dirty) state.

Owner node : node that currently holds the valid (Home/Dirty) copy of a block and must supply the data when needed.

Exclusive node : node that has a copy of the block in its cache in an exclusive state, either dirty or (clean) exclusive as the case may be.

Slide 7.11

DIRECTORY ORGANIZATION



A natural way to organize a directory is to maintain the directory information for a block together with the block in main memory, that is, at the home node for the block.

A simple organization for the directory information for a block is as a bit vector of P presence bits - which indicate for each of the P nodes (uniprocessor or multiprocessor) whether that node has a cached copy of the block - together with one or more state bits. Let us assume for simplicity that there is only one state bit, called the dirty bit, which indicates if the block is dirty in one of the node caches. Of course, if the dirty bit is ON, then only one node (the dirty node) should be caching that block and only that node's presence bit should be ON.

Bit-vector directory organization: Operation

Consider a protocol with three stable cache states (MSI)

On a read miss or write miss at node i, the local communication assist or controller looks up the address of the memory block to determine if the home is local or remote. If it is remote, a network transaction is sent to the home node for the block. There, the directory entry for the block is looked up, and the assist at the home may treat the miss as follows:

- If the dirty bit is OFF, then the assist obtains the block from main memory, supplies it to the requestor in a reply network transaction, and turns the ith presence bit, presence[i], ON.
- If the dirty bit is ON then the assist responds to the requestor with the identity of the node whose presence bit is ON (i.e., the owner or dirty node). The requestor then sends a request network transaction to that owner node. At the owner, the cache changes its state to shared and supplies the block to both the requesting node, which stores the block in its cache in shared state, as well as to main memory at the home node. At memory, the dirty bit is turned OFF, and presence[i] is turned ON.

Bit-vector directory organization: Operation [contd.]

A write-miss by processor i goes to memory and is handled as follows:

- If the dirty bit is OFF, then main memory has a clean copy of the data. Invalidation request transactions must be sent to all nodes j for which presence [j] is ON. Assuming a strict request-response scenario, the home node supplies the block to the requesting node i together with the presence bit vector. The directory entry is cleared, leaving only presence [i] and the dirty bit ON. (If the request is an upgrade (from shared) instead of a read exclusive, an acknowledgment containing the bit vector is returned to the requestor instead of the data itself.) The assist at the requestor sends invalidation requests to the required nodes and waits for invalidation acknowledgment transactions from the nodes, indicating that the write has completed with respect to them. Finally, the requestor places the block in its cache in dirty state.

- If the dirty bit is ON, then the block is first recalled from the dirty node (whose presence bit is ON), using network transactions with the home and the dirty node. That cache changes its state to invalid and then the block is supplied to the requesting processor, which places the block in its cache in dirty state. The directory entry is cleared, leaving only presence [i] and the dirty bit ON.