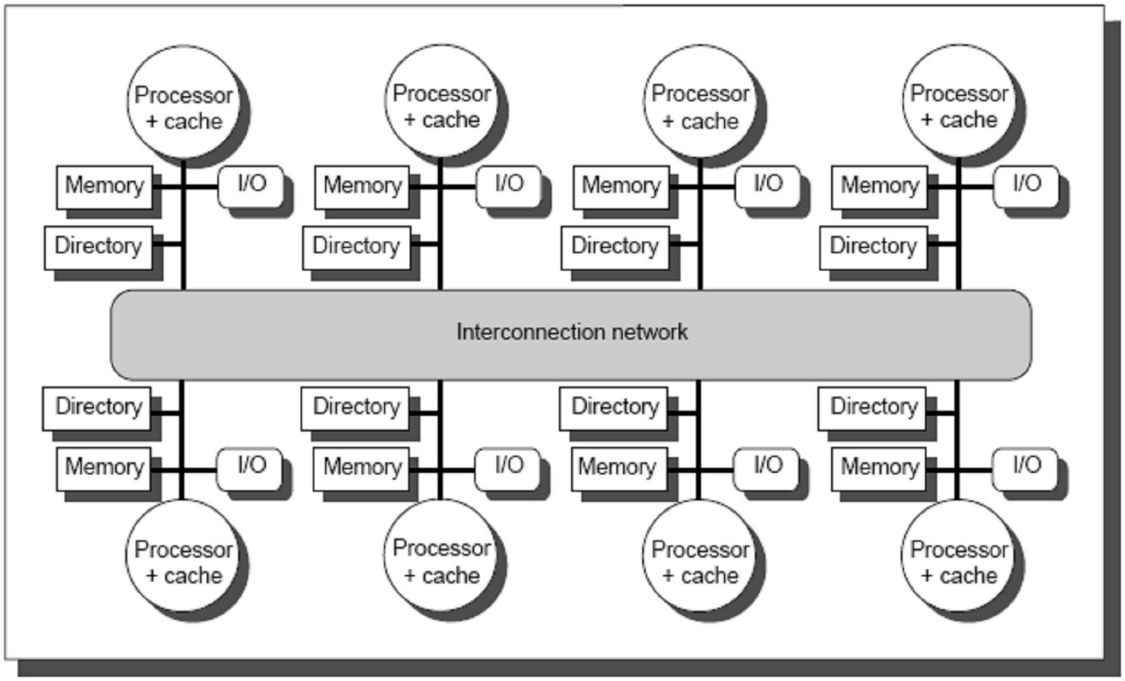
Directory-Based Cache-Coherence Protocols

Just as with a snooping protocol, there are two primary operations that a directory protocol must implement: handling a read miss and handling a write to a shared, cache block.



A directory is added to each node to implement cache coherence in a distributed-memory multiprocessor. Each directory is responsible for tracking the caches that share the memory addresses of the portion of memory in the node. The directory may communicate with the processor and memory over a common bus, as shown, or it may have a separate port to memory, or it may be part of a central node controller through which all intra node and inter node communications pass.

To implement these operations, a directory must track the state of each cache block. In a simple protocol, these states could be the following:

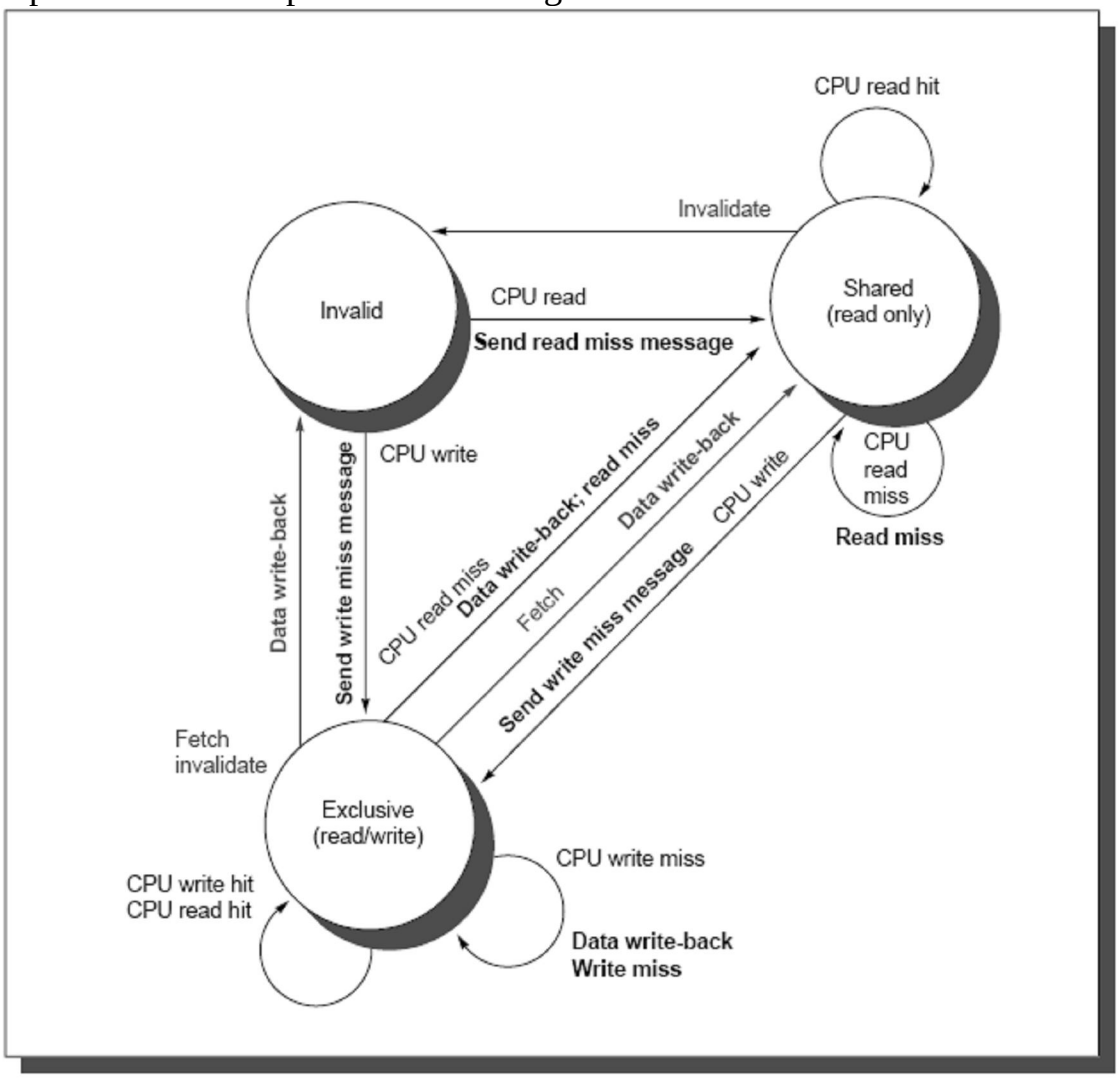
* *Shared*—One or more processors have the block cached, and the value in memory is up to date (as well as in all the caches).
* *Uncached*—No processor has a copy of the cache block.
* *Exclusive*—Exactly one processor has a copy of the cache block and it has written the block, so the memory copy is out of date. The processor is called the *owner* of the block.

In addition to tracking the state of each cache block, we must track the processors that have copies of the block when it is shared, since they will

need to be invalidated on a write. The simplest way to do this is to keep a bit vector for each memory block. When the block is shared, each bit of the vector indicates whether the corresponding processor has a copy of that block. We can also use the bit vector to keep track of the owner of the block when the block is in the exclusive state. For efficiency reasons, we also track the state of each cache block at the individual caches.

The basic states of a cache block in a directory-based protocol are exactly like those in a snooping protocol, and the states in the directory are also analogous to those we showed earlier. Thus we can start with simple state diagrams that show the state transitions for an individual cache block and then examine the state diagram for the directory entry corresponding to each block in memory. As in the snooping case, these state transition diagrams do not represent all the details of a coherence protocol; however, the actual controller is highly dependent on a number of details of the multiprocessor (message delivery properties, buffering structures, and so on). In this section

we present the basic protocol state diagrams



State transition diagram for an individual cache block in a directorybased

system. Requests by the local processor are shown in black and those from the home directory are shown in gray. The states are identical to those in the snooping case, and the transactions are very similar, with explicit invalidate and write-back requests replacing the write misses that were formerly broadcast on the bus. As we did for the snooping controller, we assume that an attempt to write a shared cache block is treated as a miss; in practice, such a transaction can be treated as an ownership request or upgrade request and can deliver ownership without requiring that the cache block be fetched. We use the same notation as in the last section, with requests coming from outside the node in gray and actions in bold. The state transitions for an

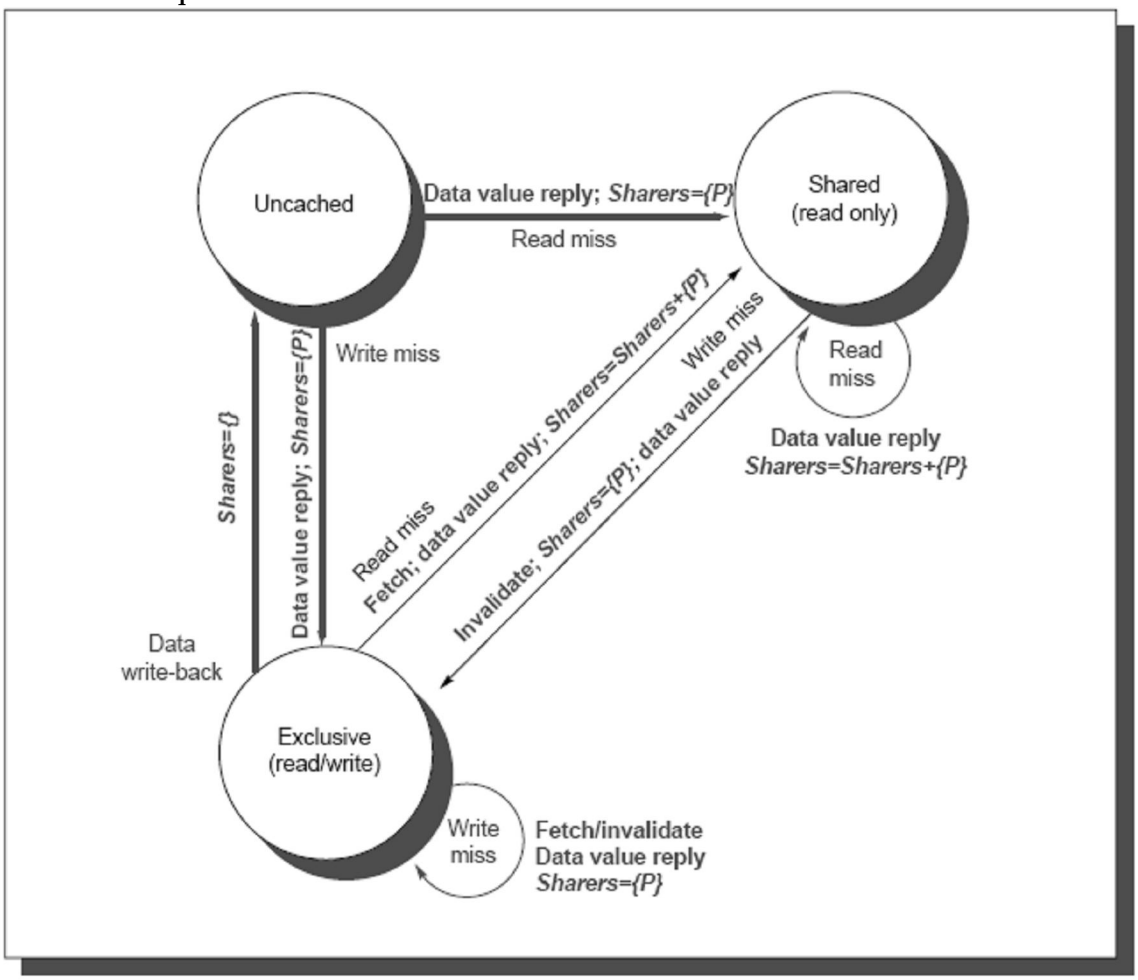
individual cache are caused by read misses, write misses, invalidates, and data fetch requests; these operations are all shown in Figure above.
An individual cache also generates read and write miss messages that are sent to the home directory. Read and write misses require data value replies, and these events wait for replies before changing state.
The operation of the state transition diagram for a cache block in Figure above is essentially the same as it is for the snooping case: the states are identical, and the stimulus is almost identical. The write miss operation, which was broadcast on the bus in the snooping scheme, is replaced by the data fetch and invalidate operations that are selectively sent by the directory controller. Like the snooping protocol, any cache block must be in the exclusive state when it is written and any shared block must be up to date in memory.
In a directory-based protocol, the directory implements the other half of the coherence protocol. A message sent to a directory causes two different types of actions: updates of the directory state, and sending additional messages to satisfy the request. The states in the directory represent the three standard states for a block; unlike in a snoopy scheme, however, the directory state indicates the state of all the cached copies of a memory block, rather than for a single cache block. The memory block may be uncached by any node, cached in multiple nodes and readable (shared), or cached exclusively and writable in exactly one node. In addition to the state of each block, the directory must track the set of processors that have a copy of a block; we use a set called *Sharers* to perform this function. In multiprocessors with less than 64 nodes (which may represent 2-4 times as many processors), this set is typically kept as a bit vector. In larger multiprocessors, other techniques, which we discuss in the Exercises, are needed. Directory requests need to update the set Sharers and also read the set to perform invalidations.

Figure below shows the actions taken at the directory in response to messages received. The directory receives three different requests: read miss, write miss, and data write back. The messages sent in response by the directory are shown in bold, while the updating of the set Sharers is shown in bold italics. Because all the stimulus messages are external, all actions are shown in gray. Our simplified protocol assumes that some actions are atomic, such as requesting a value and sending it to another node; a realistic implementation cannot use this assumption. To understand these directory operations, let’s examine the requests received and actions taken state by

state. When a block is in the uncached state the copy in memory is the current value, so the only possible requests for that block are

* *Read miss*—The requesting processor is sent the requested data from memory and the requestor is made the only sharing node. The state of the block is made shared.
* *Write miss*—The requesting processor is sent the value and becomes the Sharing node. The block is made exclusive to indicate that the only valid copy is cached. Sharers indicates the identity of the owner.

When the block is in the shared state the memory value is up-to-date, so the same two requests can occur:



The state transition diagram for the directory has the same states and structure as the transition diagram for an individual cache. All actions are in gray because they are all externally caused. Bold indicates the action taken by the directory in response to the request. Bold italics indicate an action that updates the sharing set, Sharers, as opposed to sending a message.

* *Read miss*—The requesting processor is sent the requested data from memory and the requesting processor is added to the sharing set.
* *Write miss*—The requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages, and the Sharers set is to contain the identity of the requesting processor. The state of the block is made exclusive. When the block is in the exclusive state the current value of the block is held in the cache of the processor identified by the set sharers (the owner), so there are three possible directory requests:
* *Read miss*—The owner processor is sent a data fetch message, which causes the state of the block in the owner's cache to transition to shared and causes the owner to send the data to the directory, where it is written to memory and sent back to the requesting processor. The identity of the requesting processor is added to the set sharers, which still contains the identity of the processor that was the owner (since it still has a readable copy).
* *Data write-back*—The owner processor is replacing the block and therefore must write it back. This write-back makes the memory copy up to date (the home directory essentially becomes the owner), the block is now uncached, and the sharer set is empty.
* *Write miss*—The block has a new owner. A message is sent to the old owner causing the cache to invalidate the block and send the value to the directory, from which it is sent to the requesting processor, which becomes the new owner.

Sharers is set to the identity of the new owner, and the state of the block remains exclusive.