Request	Source	State of addressed cache block	Type of cache action	Function and explanation
Read hit	processor	shared or modified	normal hit	Read data in cache.
Read miss	processor	invalid	normal miss	Place read miss on bus.
Read miss	processor	shared	replacement	Address conflict miss: place read miss on bus.
Read miss	processor	modified	replacement	Address conflict miss: write-back block, then place read miss on bus.
Write hit	processor	modified	normal hit	Write data in cache.
Write hit	processor	shared	coherence	Place invalidate on bus. These operations are often called upgrade or *ownership* misses, since they do not fetch the data but only change the state.
Write miss	processor	invalid	normal miss	Place write miss on bus.
Write miss	processor	shared	replacement	Address conflict miss: place write miss on bus.
Write miss	processor	modified	replacement	Address conflict miss: write-back block, then place write miss on bus.
Read miss	bus	shared	no action	Allow memory to service read miss.
Read miss	bus	modified	coherence	Attempt to share data: place cache block on bus and change state to shared.
Invalidate	bus	shared	coherence	Attempt to write shared block; invalidate the block.
Write miss	bus	shared	coherence	Attempt to write block that is shared; invalidate the cache block.
Write miss	bus	modified	coherence	Attempt to write block that is exclusive elsewhere: write-back the cache block and make its state invalid.

FIGURE e5.12.9 The cache coherence mechanism receives requests from both the processor and the bus and responds to these based on the type of request, whether it hits or misses in the cache, and the state of the cache block specified in the request. The fourth column describes the type of cache action as normal hit or miss (the same as a uniprocessor cache would see), replacement (a uniprocessor cache replacement miss), or coherence (required to maintain cache coherence); a normal or replacement action may cause a coherence action depending on the state of the block in other caches. For read misses, write misses, or invalidates snooped from the bus, an action is required only if the read or write addresses match a block in the cache and the block is valid. Some protocols also introduce a state to designate when a block is exclusively in one cache but has not yet been written. This state can arise if a write access is broken into two pieces: getting the block exclusively in one cache and then subsequently updating it; in such a protocol this "exclusive unmodified state" is transient, ending as soon as the write is completed. Other protocols use and maintain an exclusive state for an unmodified block. In a snooping protocol, this state can be entered when a processor reads a block that is not resident in any other cache. Because all subsequent accesses are snooped, it is possible to maintain the accuracy of this state. In particular, if another processor issues a read miss, the state is changed from exclusive to shared. The advantage of adding this state is that a subsequent write to a block in the exclusive state by the same processor need not acquire bus access or generate an invalidate, since the block is known to be exclusively in this cache; the processor merely changes the state to modified. This state is easily added by using the bit that encodes the coherent state as an exclusive state and using the dirty bit to indicate that a block is modified. The popular MESI protocol, which is named for the four states it includes (modified, exclusive, shared, and invalid), uses this structure. The MOESI protocol introduces another extension: the "owned" state.