Update-Based Cache Coherence

In the class, we covered invalidation-based cache coherence protocols for write-back caches. Another class of protocols is update-based protocols. In an update-based protocol, whenever a cache write happens, the writer's cache sends an update to other private caches with the new data. All the private caches that contain the updated address replace their old contents with the new data. Such a design ensures that all the cache blocks in private caches contain the latest data. The main memory is only updated upon the eviction of a cache block.

This handout describes a snoopy, update-based protocol for write-back caches over a synchronous bus. To simplify the protocol, transient states are not considered.

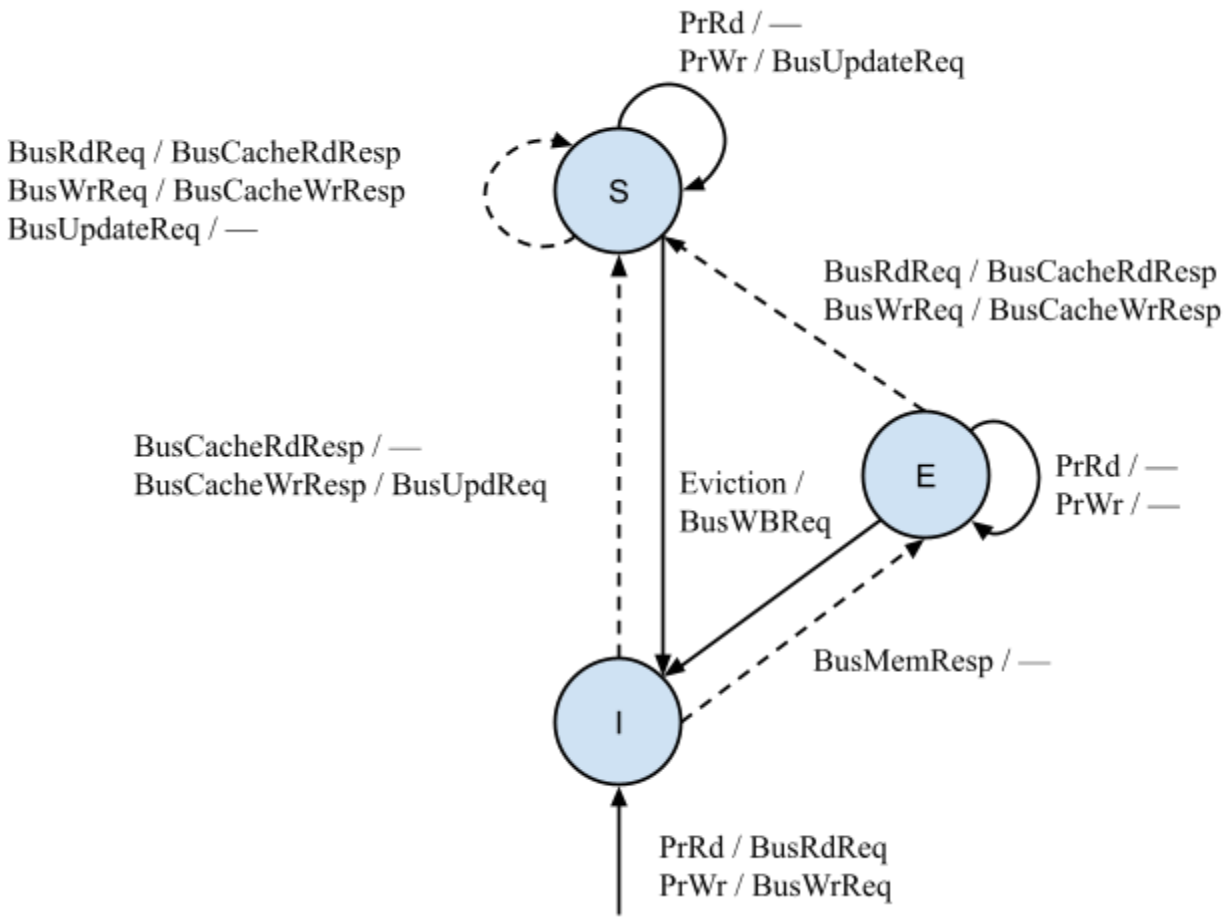
The protocol has the following 3 states:
Invalid (I): The block is not present in the cache.
Exclusive (E): The block has a single sharer and the sharer has both read and write permissions.
Shared (S): The block may have two or more sharers and all the sharers have both read and write permissions.

This protocol can be triggered by 3 processor events:
PrRd: A core issues a read request to a cache block.
PrWr: A core issues a write request to a cache block.
Eviction: A cache block is evicted due to cache conflicts.

The protocol has 4 bus requests:
BusRdReq and **BusWrReq:** Both bus requests are used to fetch the latest data from either the memory or another private cache. The requests will prioritize fetching data from private caches over the memory, and the corresponding response will indicate where the data comes from (see below). BusRdReq is issued upon a read miss, and BusWrReq is issued upon a write miss.
BusUpdReq: A bus request that is used to forward the update of a cache block to the other sharers in the cache if they exist. *BusUpdReq will not update the data in the memory.*
BusWBReq: A bus request to write back the data from the private cache to memory. In this baseline protocol, anytime a cache block is evicted and the block is not Invalid (either in Shared or Exclusive state), the data needs to be written back to memory by issuing a BusWBReq.

The protocol has 3 bus responses:
BusCacheRdResp and **BusCacheWrResp:** The two responses return the latest data and indicate that the data comes from another cache. BusCacheRdResp is the response for a BusRdReq, and BusCacheWrResp is the response for a BusWrReq.
BusMemResp: This bus response returns the latest data from memory. This response can be used to reply for both BusRdReq and BusWrReq.

The transition diagram is shown below. In this transition diagram, solid lines represent transitions that are triggered by processor events, and dashed lines represent transitions that are triggered by bus requests or responses.



Memory Barriers

Consider a system which uses Weak Ordering(WO), meaning that a read or a write may complete before a read or a write that is earlier in program order if they are to different addresses and there are no data dependencies. The WO system introduces four fine-grained memory fence instructions to allow programmers to explicitly express which reorderings of reads and writes should be prevented.

Below is the description of these instructions:

- **FENCE$_{RR}$** guarantees that all read operations initiated before the FENCE$_{RR}$ will be seen before any read operation initiated after it.
- **FENCE$_{RW}$** guarantees that all read operations initiated before the FENCE$_{RW}$ will be seen before any write operation initiated after it.
- **FENCE$_{WR}$** guarantees that all write operations initiated before the FENCE$_{WR}$ will be seen before any read operation initiated after it.
- **FENCE$_{WW}$** guarantees that all write operations initiated before the FENCE$_{WW}$ will be seen before any write operation initiated after it.