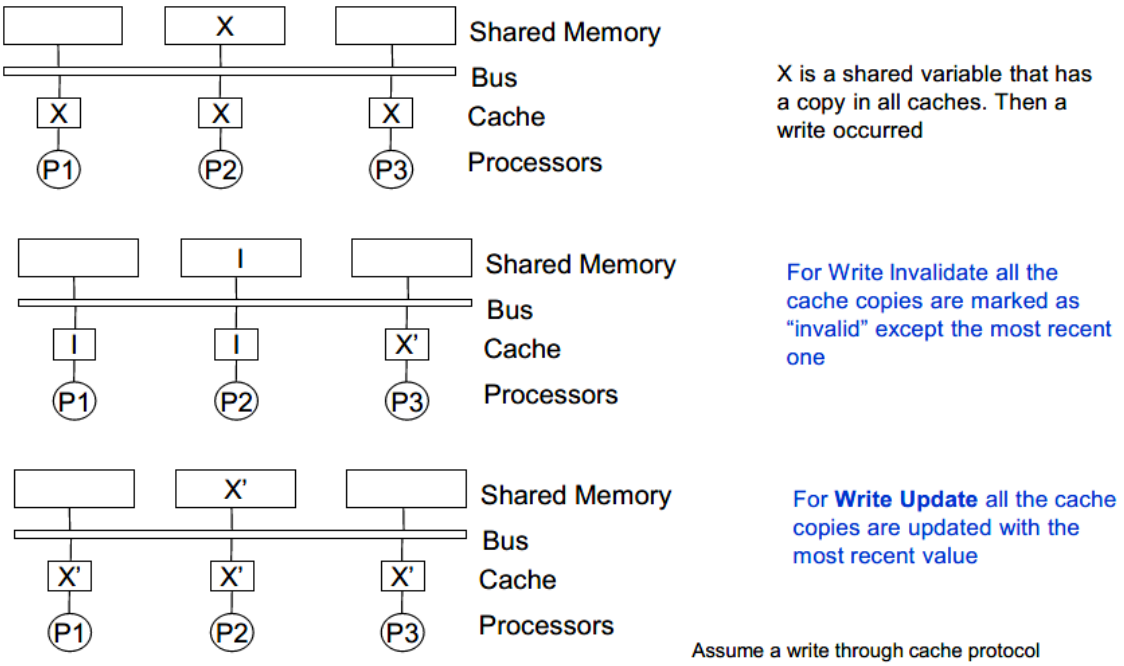
10 Memory Consistency and Cache Coherence

10.1 Cache Coherence

Problem Multiple threads share the same data. If one thread changes the data, how should the other copies of data change in order to retain coherence?

Bus Based Snooping Protocol



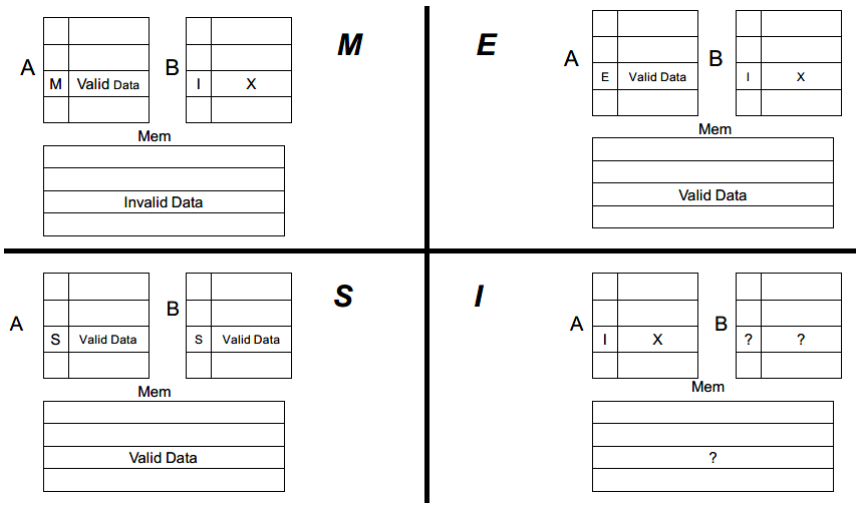
- **Write Invalidate** (commonly-used, aka **Valid-Invalid Protocol**): if data X is modified, then all the copies of X are marked as "invalid" except the most recent one.
- **Write Update**: if data X is modified all the cache copies are updated with the most recent value.

Modified-Shared-Invalid Protocol

- Two state bits in every cache line.
  - **Modified**: the cache copy is the only valid copy in the system. Memory is stale.
  - **Shared**: the cached copy is valid and it may or may not be shared by other caches (Initial state after first loaded). Memory is up-to-date.
  - **Invalid**: the cache copy is not existence.
- Operations are not atomic, but the protocol has possible extensions.

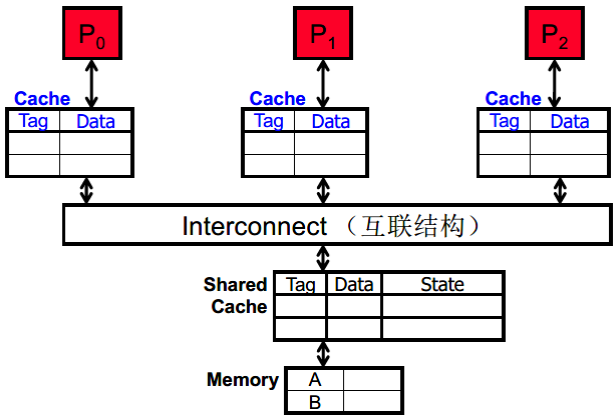
MESI Protocol and MOESI Protocol

- **MESI Protocol**



  - **Exclusive**: Main memory's value is up-to-date, and no other cache possesses a copy (only one possesses).
  - Separate **Shared** into **Shared** and **Exclusive**.
  - Do not have to inform others when changing value in state Exclusive. Reduce the number of busses transactions when a value is read exclusively and it may be modified in the future.
- **MOESI Protocol**
  - **Owner**: when the block is in **Modified** state and other processor want to read the data, change the state into **Owner**.
  - Separate **Modified** into **Owner** and **Modified**.
  - Reduce the number of busses transactions by delaying the update to the memory.

Interconnect Architecture: use interconnect architecture to send the data instead of the bus.



MESIF Protocol

- **Forward**: when the block is in **Modified** state and other processor want to read the data, change the state into **Shared** and the request processor's state into **Forward**.
- Similar to **MOESI**, but use more locality (don't have to read from the same processor).

False Sharing（假共享） exists in all protocols. We need to avoid false sharing when writing the parallel program.

10.2 Memory Consistency

Compare with Cache Coherence

- Memory Consistency: focus on the semantics of load/store operations;
- Cache Coherence: focus on the same data block in different caches.

Sequential Consistency (SC) is the model that programmer wants the machine to provide:

- Processor execute the instructions exactly in the order that the programmer writes, not only from its own view, but from other processor's view.

Total Store Order (TSO) uses a FIFO store buffer to temporarily store the data that needs to write to the cache. (used in x86.)

- Store buffer write the cache in the backstage in order;
- Load operation will also read from store buffer;
- Eliminate the stall caused by store buffer;
- May cause problems to memory-shared multi-processor (other processor may access the wrong data in the cache, but the right data is in the store buffer);
- In the processor's view, the order is correct. In other processor's view, the order may be different from the code.
- **Compare with Write Back Buffer (WBB)**: write back buffer is used to hide the stall between cache and next-level memory; store buffer is used to hide store miss.

Release Consistency (RC) use out-of-order and merging store buffer, the order of store may be different from the program. (used in ARM.)

Guarantee the order: insert **fences (memory barriers)** and so on.