Preserve Program Order

A read by a processor P to a location X that follows a write by P to X, with no writes to X by another processor occuring between the write and the read by P, always returns the value written by P

Coherent view of Memory

A read by a processor to a location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occurs between the two accesses

Write Serialisation

Writes to the same location are serialised; i.e. two writes to the same location by any two processors are seen in the same order by all the processors

For example, if the values 1 and then 2 are written to a location, processors can never read the value of the location as 2 and then later read it as 1

When a written value will be read? Write Consistency

- If P1 writes to X and then P2 reads X. But the read/write are so close in time that P1 has not updated memory and so P2 gets stale data value
- We deal with this in the section on memory consistency models

Two assumptions before consistency

- We make following two assumptions until we deal with memory consistency:

  (1) A write does not complete, and allow the next write to occur, until all processors have seen the effect of that write

  (2) The processor does not change the order of any write with respect to any other memory access

- e.g. If processor writes to location A followed by location B, any processor that sees the new value of B must also see the new value of A

  => These restrictions allows the processor to reorder reads, but writes finish in program order

Basic schemes for Enforcing Coherence

MIGRATION

Data moved to local cache and accessed there in transparent fashion

+ Reduces latency of
Remote access to main memory

+ Reduces bandwidth demand of shared memory (so that memory is available for others)

REPLICATION

To enable simultaneous read of shared data, caches make a copy of data in local-cache

+ Reduces remote access latency

+ Reduce contention at memory
To read shared data

Easy solution to solve coherence is to avoid sharing at the software level

However, SMPs use hardware implementation of protocols to maintain coherence

Types of Coherence Protocols

Snooping
&
Directory based

Cache Coherence Protocols

- Key to implementation is tracking the state of any sharing of data block
- Two classes of protocol:
  - Snooping-based and directory-based

Snooping vs Directory

Snooping-based

- Sharing states kept in every cached copy of the data. Not at a centralised place
- Easy to implement
- Uses broadcasting which limits the scalability
- All caches accessible via a broadcast medium and the cache controller monitors or snoops the medium to check if the shared (cached) data is needed by someone

Directory-based

- Sharing status of a block in physical memory is kept in just one location, called the directory (separate structure or with the shared LLC)
- Higher implementation overhead
- But can scale for large number of multiprocessors

Basic Implementation Techniques (Architectural Building Blocks)

- All processors continuously snoop on the bus to check if address on the bus is in their cache **[CACHE]**

  State can be changed to valid, invalid, exclusive

- When a write occurs to a shared block --> acquire broadcast medium (i.e. bus) to **send inv** request

  If 2 processors try to write, they have to arbitrate for the bus

- The processor which gets the bus first, writes the value and the second processor has to invalidate its own copy
- The 2nd processor can later get the bus and do the write
- => [**BUS]** enforces write-serialisation
- When new cache miss occurs for an invalidated shared block, where to locate up-to-date copy of data? **[LATEST COPY]**