Protocol Races

[§10.4] We have assumed—

- Directory state reflects the most up-to-date state of caches.
- Messages due to a request are processed atomically.

In reality, one of or both conditions may be violated

- *Protocol races* can occur
- Some protocol races can be handled in a simple way; others are trickier.

We will discuss how protocol races can be handled.

- Purpose of discussion: illustrate approaches for dealing with protocol races.
- Discussing *all* possible races is not the goal.

Handling races: out-of-sync directory

[§10.4.1] Suppose the home sends an invalidation to a node that has replaced the block silently.

- The node can reply with

Suppose that the home receives a read request from a node that is already a sharer from the home point of view.

- The directory can reply with data

Suppose that the home receives a read/write request from a node that the home thinks is the owner.

- (In the directory, what state is this block in?)
- What might have happened to the block?
    - If the block was clean,
    - If the block was dirty,
- What should the home do? (Why will neither of these work?)
    - Wait?

- Reply with data?

- The directory alone cannot resolve this. Coherence controllers at other nodes must participate in the solution.

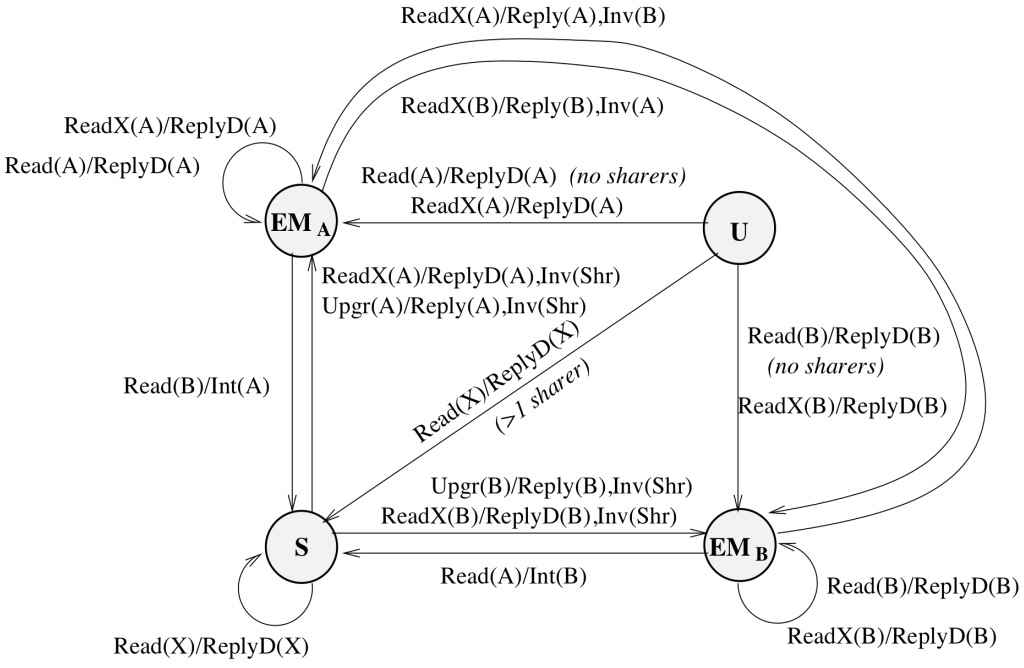
- What does the coherence controller at a node *n* need to do when a flush or writeback occurs?
  - Maintain an *outstanding transaction buffer* (OTB) for flush messages.
  - Require the home to acknowledge the receipt of a flush

- These two steps allow node *n* to delay a Read/ReadX request to a block that is still being written back.

- Hence, the home only receives Read/ReadX to a block that is not being written back.
  - When it does, it can send a

Protocol modification

Here is a modified state-transition diagram.



What is the meaning of “owner” in a directory protocol?

The meaning of “owner” is ambiguous here …

- because the directory may be out of sync with cache states,
- the directory may get a Read or ReadX from a node it *thinks* is the owner (but actually isn’t).

(This isn’t permitted by the protocol.)

What do we do about it?

- Split EM into two states (EM_A and EM_B) to reflect this situation.
- EM_A means the directory thinks the current owner is A.
- EM_B means the directory thinks the current owner is B.

Transitions from state U

Suppose the block is in state U in the directory.

- What happens on a ReadX request?
  - The system fetches the block from the local memory, sends a ReplyD to the Requester, and moves to state

- What happens on a Read request?
  - The system
  - What state does the requesting cache transition to?
  - What state does the directory transition to?

Transitions from state S

Suppose the directory state is S.

- What happens on a Read request?
  - The directory knows it has a valid block in the local memory.
  - It sends a _____ to the Requester and updates the sharing vector.
  - Directory state

- What happens on a ReadX request?
  - Directory sends _____ to the Requester.
  - Directory sends _____ to all (other) sharers.
  - State changes to
  - But, if it's an upgrade, it just

Transitions from state EM

Suppose WLOG the directory state is EM_A.

Suppose a Read request (from a different node B) is received.

- The state is set to
- An _____ is sent to the owner (A) to change its state to ___

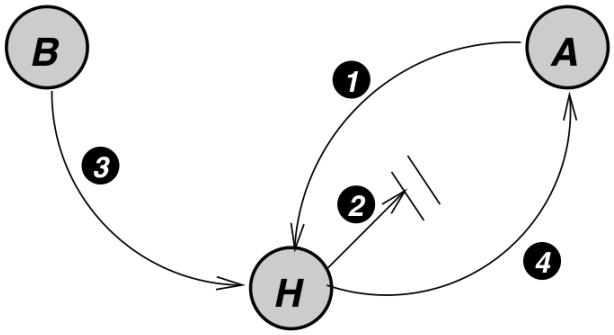
Suppose a ReadX request (from a different node B) is received.

- Directory sends an invalidation message to
- This message also says to send the data to
- Directory sends a reply message to B, saying that __ will supply the data.
- State transitions to
- (Note that it doesn't matter whether owner is in state E or M.)

Suppose the directory has an out-of-sync view of cache states, and is in state EM_A.

- Suppose it receives a Read or ReadX from A.
  - This means A's block must've been replaced due to a cache miss.
- The directory knows that A is really the owner.
- Thus, it can just respond with

Handling races: non-atomic messages



1. [§10.4.2] A sends a read request to home.
2. Home replies with data (but the message gets delayed).
3. B sends a write request to home.
4. Home sends invalidation to A, and it arrives before the ReplyD

Why is this a problem?

This is called an “early invalidation” race.

How should A respond to the invalidation?

Two incorrect ways to respond:

- A replies with InvAck.
  - B thinks that its write propagation is complete
  - A receives a ReplyD and places the block in its cache (the block that should have been invalidated).
- A ignores the invalidation message
  - The message is lost; write propagation has failed to occur

Solution:

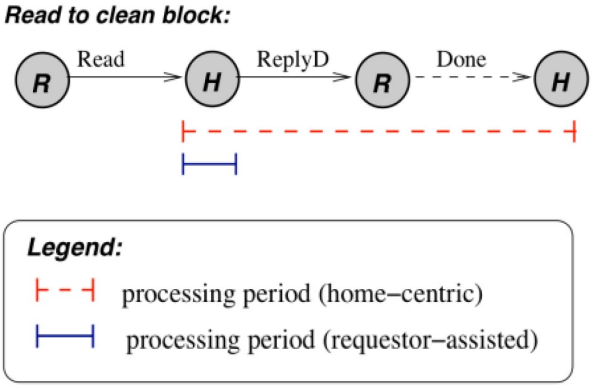
- Brute force (avoids overlapped handling of requests):
  - Home waits until it receives ack from all parties (home-centric)
- Allow overlapping but ask nodes to participate (requester-assisted)
  - Node keeps an OTB

- It does not entertain requests (to the same block) until the current transaction is completed

Exercise: *Explain how each of these scenarios would play out using the four-step diagram above.*

Processing a Read Request

Case 1: Read to clean block



Home-centric approach

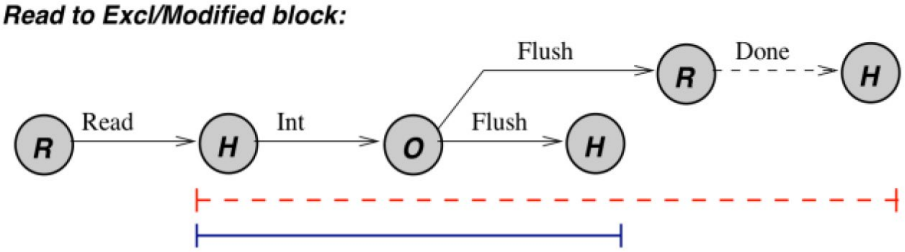
- Directory enters a transient state.
- Home replies with data
- Requester receives data, sends ack to home.
- Home closes transaction (transitions to a stable state, update sharing vector).

Cons: too much serialization at home, transaction closed late, and it requires ack

Requester-assisted approach

- Directory sends ReplyD, then closes transaction
- Requester buffers/nacks all new requests until ReplyD received (i.e., till the current Read transaction is completed)

Case 2: Read to block in EM state



Home-centric approach

- Requester sends Read to home
- Home enters a transient state, sends intervention to owner
- Owner flushes block to home and requester
- Requester sends ack back to home
- Home closes transaction (transitions to shared state, updates sharing vector)

Requester-assisted approach

- Requester sends Read to home
- Home enters a transient state, sends intervention to owner
- Home cannot close the transaction yet, because in the final state (Shared), it must have a clean copy of the block
- Owner flushes block to home and requester
- Upon receiving the block from owner, home closes transaction

Processing a write (ReadX) request

We will cover this in the next class.

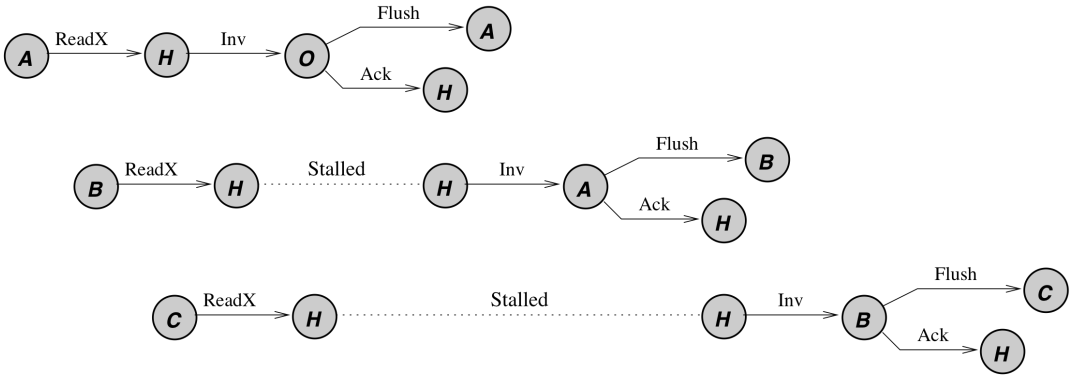
Write Propagation and Serialization

[§10.4.3] In a directory-based protocol,

- Write propagation is achieved through invalidation.
- Multiple writes to a block are serialized by the protocol.
  - Transaction closes after the ack from current owner is received by home.

- A new ReadX request is not served until the previous ReadX request is closed.
- This provides write serialization

Here is a diagram of serializing writes by A, B, and C.



Is it using the home-centric or requester-assisted scheme?

Memory consistency models

[§10.4.5] Implementing sequential consistency:

All memory accesses by a processor must be issued and completed in program order.

Which of the two (issuing or completion) is hardest to assure?

- Write completion detected when all InvAcks are collected
- When does read completion occur?

- Prefetching and load speculation can be used.

As the number of processors grows,

- Average latency of a cache miss increases
- Harder to hide it
- What does this do to the viability of SC?