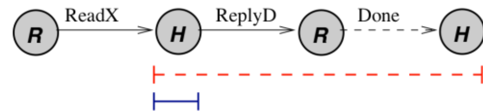
Handling races: non-atomic messages (cont.)

Last time, we saw how to deal with read requests when another message (e.g., an invalidation) arrived while the read was being processed.

Now we want to consider what happens when a write request arrives.

Case 1: ReadX to a block in state U

ReadX to Unowned block:



Legend:

processing period (home–centric)

processing period (requestor–assisted)

Home-centric approach

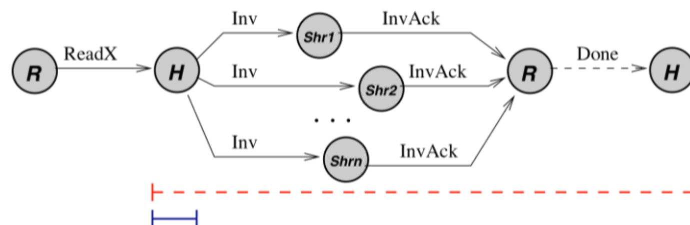
- Requester sends ReadX request
- Home responds with data
- Requester sends Ack
- Home closes transaction.

Requester-assisted approach

- Requester sends ReadX request
- Home sends

Case 2: ReadX to block in state S

ReadX to Shared block:



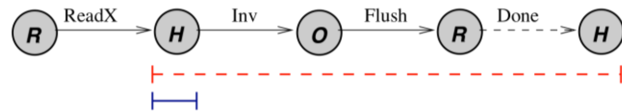
Home-centric approach

- Requester sends ReadX request
- Home enters transient state and sends Inv msgs.
- InvAcks must be
  - collected at Requester, which notifies Home, or
  - collected at Home
- Home closes transaction

Requester-assisted approach

- Requester sends ReadX request to home.
- Home sends Invs and closes the transaction
- InvAcks collected ____________
- ______________________________

Case 3: ReadX to EM block



Home-centric approach

- Requester sends ReadX request to home
- Home enters transient state and sends Inv message.
- InvAck must be
  - awaited at Requester, which notifies home, or
  - awaited at Home.
- Owner flushes block to home and requester.
- Upon receiving the block from owner, home closes transaction

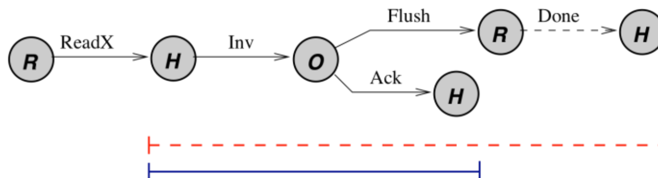
Requester-assisted approach

- Requester sends ReadX request to home.
- Home sends Inv message to owner and closes transaction.
- Owner flushes block to requester.
- Requester buffers/NACKs new requests

Case 4: ReadX to EM block with data race

Is this different from Case 3 for home-centric approach?

For the requester-assisted approach?



- What if the current owner no longer has the block?
  - Either it had it in state M and
  - or it had it in state E and
- Home cannot close the transaction yet, as it may have to supply the block.
- Hence, it can close the transaction late, after it receives Ack from the owner.

Dealing with Imprecise Directory Information

[§10.5.1] Why does directory information get stale over time?

Why isn't the directory always notified?

What problems does stale directory information cause?

1. Increased trouble (power consumption, latency) in locating a block
2. Storage overhead
3. Extra blocks invalidated when directory gets full
4. Increase in invalidation traffic

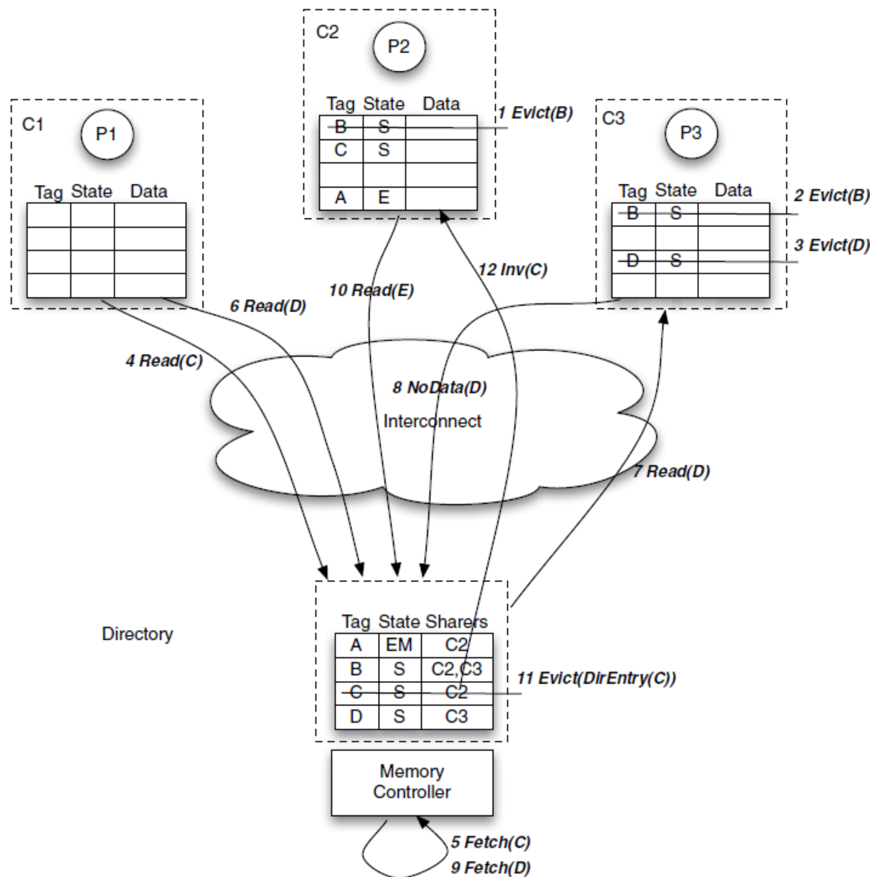
Let's consider these in order.

Problem 1 is caused by three evictions. After the evictions, list the tags of the directory entries that are incorrect.

When a core C1 wants to fetch C, it hedges because the directory info might be incorrect.

- If the directory info is correct, where should it get the data from?
- If the directory info is incorrect, where should it go for the data?

- Which choice has the least latency?
- Which choice takes the least power?
- Which steps in the diagram illustrate the hazard (in terms of power and/or latency) in making the wrong choice?
- A "compromise" is to look in both places. Is this better from the standpoint of latency and/or power?



Problem 2 (storage overhead) is caused by unneeded directory entries occupying space in the directory. Which directory entries above are unneeded at the end of Step 9?

Problem 3 is illustrated by which steps in the above diagram?

How can it cause an unnecessary cache miss?

Problem 4 is higher invalidation traffic. But why might traffic not be higher when stale directory entries are allowed?

Solihin suggests two antidotes.

- Aggregating notification messages on clean-block purges.
- Predicting when directory blocks are invalid, based on # of cache misses from a particular LLC.

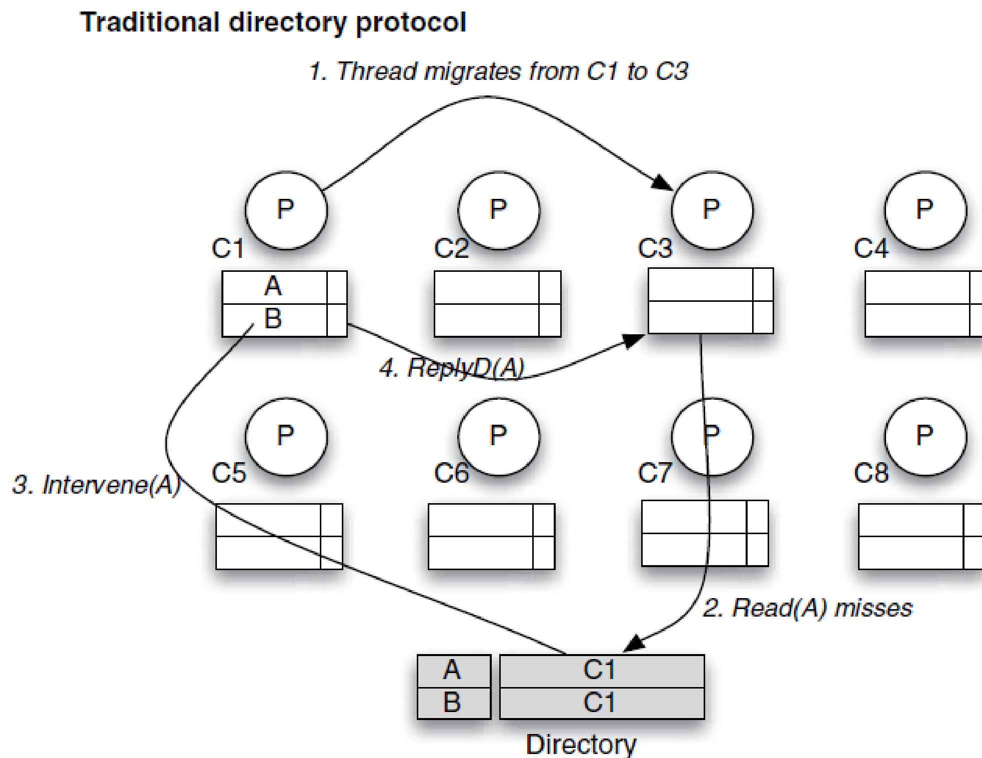
Accelerating thread migration

Ordinarily, a directory keeps track of which processor has cached a copy of a block.

If a thread moves from one processor to another, it will suffer a lot of cold misses.

What are the steps in servicing such a miss?

- P3 references block A, but A is not in its cache (C3).
- So P3 consults the directory, and finds that the block is cached in C1.
- It sends a request to C1, which responds by sending a ReplyD to the requester, C3.



Is there a way to avoid repeated references to the directory for each cache block that needs to move?

Solihin suggests adding a level of indirection to the directory.

- Instead of saying the block is cached in C1, it would say that it's cached in _______________.
- Initially, V1 is set to point to ___, because that's where the block is cached.
- When the thread migrates to a new processor, the OS adds the new processor's cache to ___
- This effectively says that any block cached in C1 can also be cached in ______
- When a miss occurs, the corresponding line in C1 is consulted, and transfers the block.
    - This saves _______________ per miss.