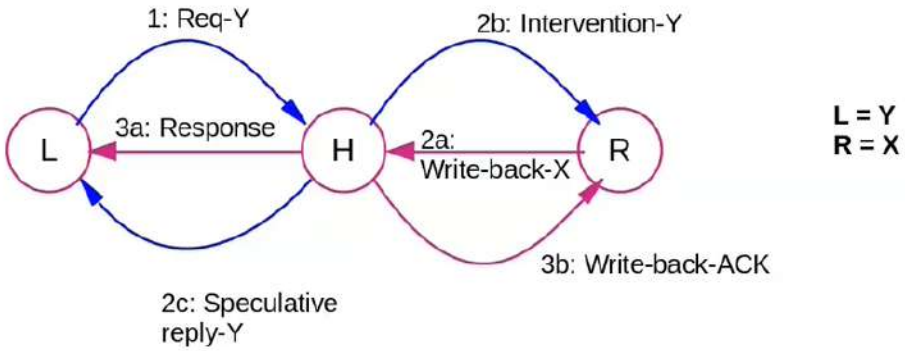
Handling write-back requests

- Here the requestor (node-X) is holding a dirty copy to be written back
  - Directory state cannot be shared or un-owned
    - If another request (=Y) has come which will set the state to shared, this new request will have been forwarded to node-X and the state of Home would be Busy
- State = Exclusive
  - Dir-state is set to un-owned and requestor is ACKed
- State = Busy (interesting race condition)



L = Y
R = X

Handling write-back requests

- State = Busy (interesting race condition)
  - Request Y made dir-state = Busy as directory forwarded request-Y to X-owner as intervention
  - Node-X has sent write-back
    - Request-Y and Req-X have crossed each other
    - Req-Y operation has already started processing at Home
  - We cannot drop the write-back (from X) request else
    - (1)the only valid copy of data will be lost
    - (2) also, we cannot NACK write-back and retry after req-Y completes because, serving req-Y before write-back means Y gets old copy from memory while the dirty copy at X is pending a write-back

Solution: race condition

- Solution to this race = combine the two operations
- (1) when write-back reaches Directory, it changes the state
- New Dir-state = Shared? Exclusive?
  - Shared : if earlier state was busy-shared (i.e. Y asked Read copy)
  - Exclusive: if earlier state was busy-exclusive (i.e. Y asked for ReadEx)
- (2) home forwards writeback to Y and sends WB-ACK to X
- (3) Later when X received intervention (on behalf of Y) it ignores it
  - X knows this as it has an outstanding WB for the same block
- (4) Y's operation completes when it gets the reply
- (5) X's operation completes when it gets write-back-ACK

Replacement of Shared block

- Block in shared state is replaced (node-X) in the local-cache
- Could send replacement-hint to Directory to remove node from sharer's list
- This will eliminate an inv-message being sent to X when some cache-Y wants to write the block
- However, this does not reduce traffic, as either send replacement-hint "now" or incur wasteful inv message "later"
- Origin protocol does not use replacement hints
- Total transaction types:
  - Coherent memory: 9 request, 6 inv / intervention, 39 responses
  - Non-coherent memory: (I/O, synch, special operations): 19 request, 14 reply. No inv/interventions

Correctness

- Serialisation of Operations
  - Need a serialisation agent
    - Home node is a good candidate, as all misses go there first
  - Possible mechanism: Buffer request at home in FIFO order
    - Buffer until previous requests forwarded from home have returned replies to it
    - But input buffer problem becomes acute at the home
- Possible solutions
  - (1) Let input buffer overflow into memory (MIT Alewife)
  - (2) Don't buffer at home, but forward to owner node (Stanford DASH)
    - Serialisation determined by home, when clean-copy and owner when exclusive copy
    - If cannot be satisfied at owner, i.e. block written-back or ownership given-up
      - Then owner NACKs request to requestor without serialising. This request will be retried later
  - (3) Don't buffer at home, use busy state to NACK (Origin)
    - Serialisation order is that in which requests are accepted (not NACKed)
  - (4) FIFO buffer in distributed way (SCI)

Serialising entity is not sufficient

- With multiple copies in a distributed network, simply identifying a serialising entity is not enough
- The problem is that, the home or serialising agent may know when its involvement with a request is completed, BUT this does not mean that a request is completed with respect to other nodes
- Some transactions for the next request to that block may reach other nodes and perform with respect to them, before some remaining transactions for the previous request
- This shows that, in addition to the system providing a global serialising entity for a block, individual nodes (ex requestors) should also preserve a local serialisation wrt each block.
  - Eg: they should not apply incoming transactions to a block while they still have a transaction outstanding for that block

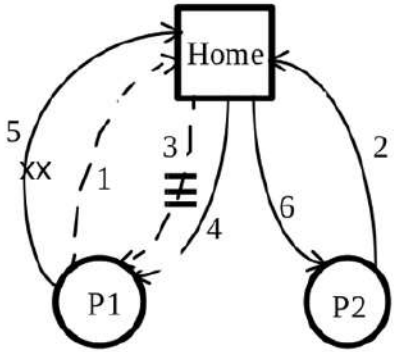
Single entity for serialisation is not enough: EX-1

- As it may not know when all transactions for that operation are done everywhere
- Ex-1:
  - Rd-A (1) and Wr-A should be serialised
  - Rd-A (2) must happen after Wr-A of P2

P1	P2
Rd A --- (1)	Wr A
BARRIER	BARRIER
Rd A ---- (2)	

Single entity for serialisation is not enough

- EX-1: Effect of Wr-A gets LOST !
- **Shows need for local serialisation at requestor**



1. P1 issues read request to home node for A
2. P2 issues read-exclusive request to home corresponding to write of A. But won't process it until it is done with read
3. Home recieves 1, and in response sends reply to P1 (and sets directory presence bit). Home now thinks read is complete. Unfortunately, the reply does not get to P1 right away.
4. In response to 2, home sends invalidate to P1; it reaches P1 before transaction 3 (no point-to-point order among requests and replies).
5. P1 receives and applies invalideate, sends ack to home.
6. Home sends data reply to P2 corresponding to request 2.

Finally, transaction 3 (read reply) reaches P1.

P1 overwrites the invalidated copy with this data.

4-Oct-2021 - Google

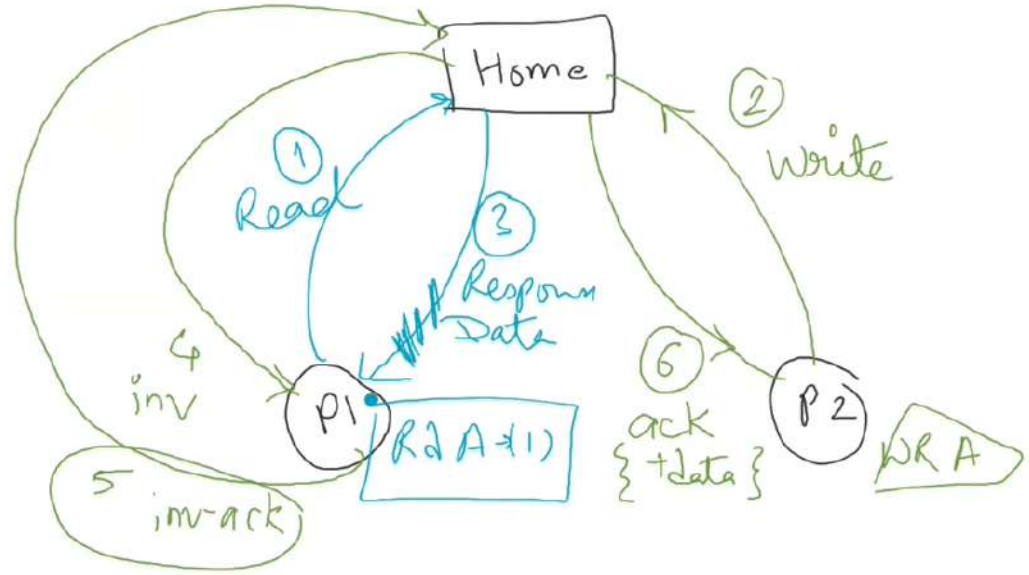
jamboard.google.com/d/1xgN9R4EQq0OZrDzyhb5wMm6ZrOqQYsfUxoR5tdgHi1U/viewer?f=10

4-Oct-2021

11 / 11

Share

Set background | Clear frame



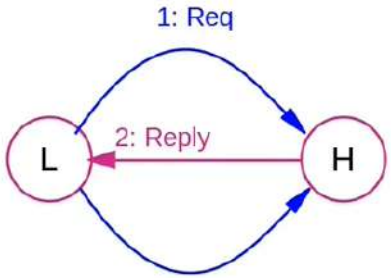
NS NALABOLU SAN...
NS NISHTHA SHARMA
SP SARASWATULA P...
PM PAIDIMARRI MA...
DG DEVANSHI GUPTA
SK SHIVANGI KUMAR
AS ASWATHY N S
RP RAKSHIT RAJEND...
IMLIJUNGLA LON...
KOUSIK RAJESH
SHIVAM BAGHEL
ANNAPURNE KRI...
NARESH BHARAS...
YK YOGESH KUMAR
AM ANSHUL MITTAL
NIHARIKA BHAM...
TANVISH
+2

77% (3:14) Mon, October 11, 15:29

- Effect?
- When P1 reads-A after BARRIER, it should have found an invalid block, but now it reads a stale value
- The effect of write by P2 is lost, as far as P1 is concerned
- Solutions:
- (1) home does not allow P2- request. Home waits for read-ACK
- (2) P1 waits for read to complete before inv

Solutions

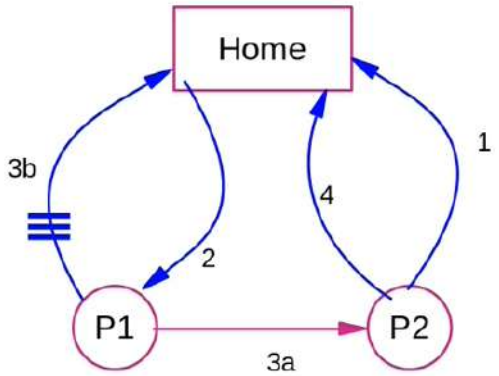
- (1) Home waits for read-ACK
  - Read reply needs to be acknowledged and Home must wait for this ACK before serving next request
  - But this violates strict req-response nature of the protocol
  - Causes buffering and deadlock problems
  - Leads to long delays



- (2) Solution by P1
  - Node that has request outstanding for a block does not allow access by another request (such as inv) to apply on the block until the outstanding request completes
  - P1 may buffer the incoming inv (EX: Origin) and apply it later
  - OR P1 applies inv and when read-reply comes it NACKs it and retry read (Ex: DASH)

Single entity for serialisation is not enough: EX-2

- EX-2 shows need for local serialisation at Home else Directory information corrupted



Initial: Block is Dirty in P1's cache
(1) P2 sends readEx to Home
(2) Home forwards request to P1 (owner node)
(3a) P1 sends data reply to P2
(3b) P1 sends "owndership transfer" revision message to Home so that Home changes owner from P1 to P2 .... this is slow to reach !
(4) P2 gets data and considers write to be complete
P2 writes to block
P2 locally-incurs block replacement and sends write-back to ~~Home~~

Home gets write-back (4) before (3b) revision and so block is updated in memory
Later when revision arries, the Directory makes owner=P2
But his is untrue and our protocol is not correct

4-Oct-2021 - Google

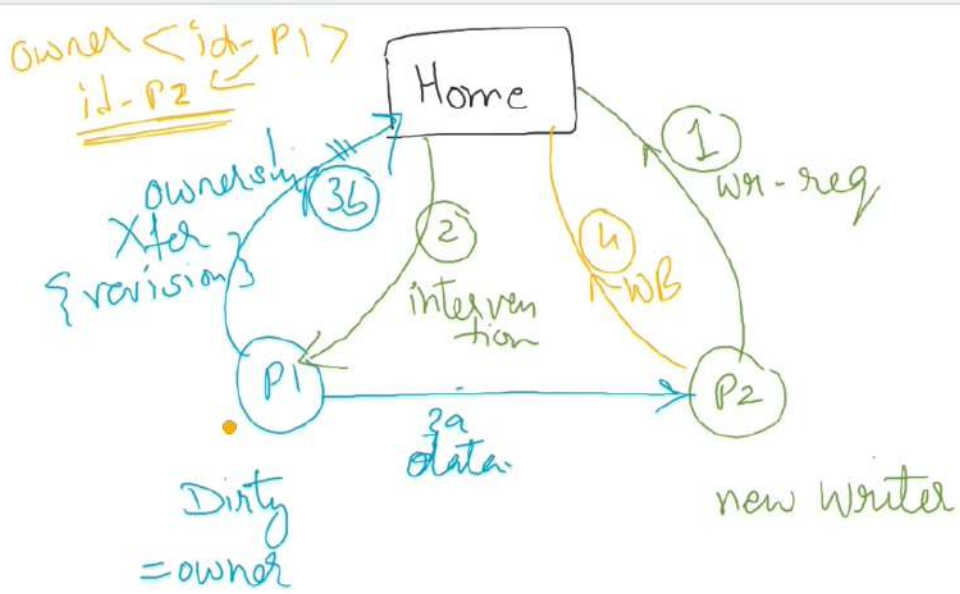
jamboard.google.com/d/1xgN9R4EQq0OZrDzyhb5wMm6ZrOqQYsfUxoR5tdgHi1U/viewer?f=11

4-Oct-2021

12 / 12

Share

Set background | Clear frame

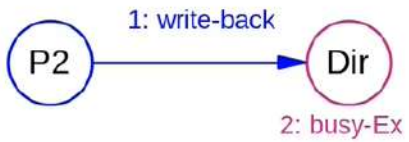


NS NALABOLU SAN...
NS NISHTHA SHARMA
SP SARASWATULA P...
PM PAIDIMARRI MA...
DG DEVANSHI GUPTA
SK SHIVANGI KUMAR
AS ASWATHY N S
RP RAKSHIT RAJEND...
IMLIJUNGLA LON...
KOUSIK RAJESH
SHIVAM BAGHEL
ANNAPURNE KRI...
NARESH BHARAS...
YK YOGESH KUMAR
AM ANSHUL MITTAL
NIHARIKA BHAM...
TANVISH
+2

72% (2:39) Mon, October 11, 15:40

Why? Solution?

- Why this happened?
  - Home does not wait for its involvement in the wr-P2 to complete, before serving new request (write-back)
- Solution
  - The Origin protocol prevents this by using its busy-state
  - The Directory will be busy-exclusive when the write-back (from P2) arrives before the revision message
  - As wr-back is for same node, i.e. which made Dir-busy-ex, so Directory sends NACK to P2



- NOTE:
  - If P2 sends wr-back Dir (and P2's earlier request has made Directory busy) then Dir has to ignore it : send NACK
  - If other node P3 sends wr-back then no-NACK to P3
    - It forwards wr-back block from P3 to P2
    - i.e. the wr-back from P3 is not NACKed but sent as a response to the requestor (P2)

Correctness

- DEADLOCK
  - Origin has finite buffers to keep pending requests but falls-back to strict request-response when it detects contention in network (as it may lead to deadlock !)
  - Therefore if busy and buffers full then NACK new requests
- LIVELOCK
  - Classical problem of two processors trying to write to same block
  - Avoids livelock by NACKing subsequent requests
  - First request to get there makes progress, others NACKed
- STARVATION
  - NACKs can cause starvation
  - If FIFO order then no starvation
  - Solution: Associate priority with each request, which is a function of the number of times the request was NACKed
- Origin Philosophy
  - (1) memory-less: node reacts to incoming events using only its local state (and no history)
  - (2) an operation does not hold shared resources while requesting others. We NACK rather than buffer request when it reaches a busy resource...this prevents deadlock

Priority based starvation freedom

- To avoid starvation we assign priority to requests by assigning a counter to each request and incrementing it each time the request is NACKed.
- The solution to livelock says that the request which reaches first gets serviced and others are NACKed, as the directory is already in Busy state.
- How is the priority implemented?
- The directory entry maintains an extra counter with each block which stores the a threshold value for priority. Whenever a request arrives at the directory, if this request will not make the directory busy then the request is serviced. In case the request will make the directory busy, then the priority of the request is compared with the threshold. If the priority is more than threshold, then it is served else NACKed. this guarantees that requests having higher priority than threshold get serviced solving starvation problem.
- In a situation when there are 2 or more high priority requests than the threshold and one of these succeeds and other NACKed, then the directory threshold is raised to the value of this NACKed request, so that next time this request will surely get serviced.
- To stop the monotonic increase in the value of threshold, the value is reset to zero after one such request is satisfied.