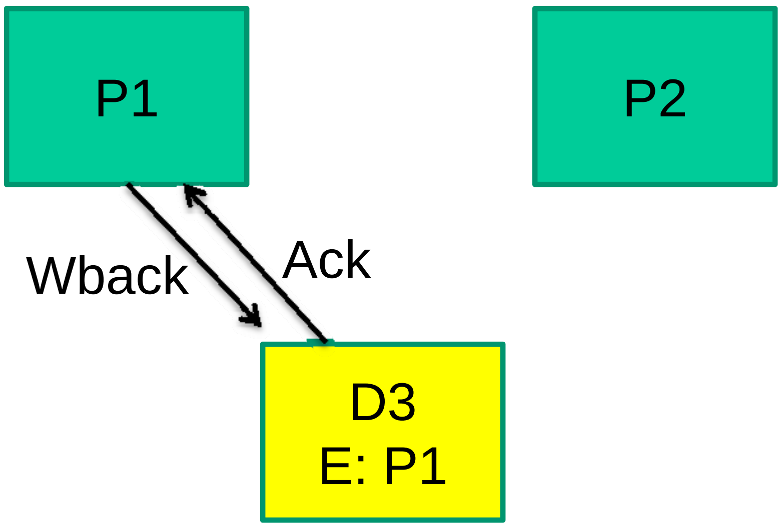
Lecture 6: Directory Protocols

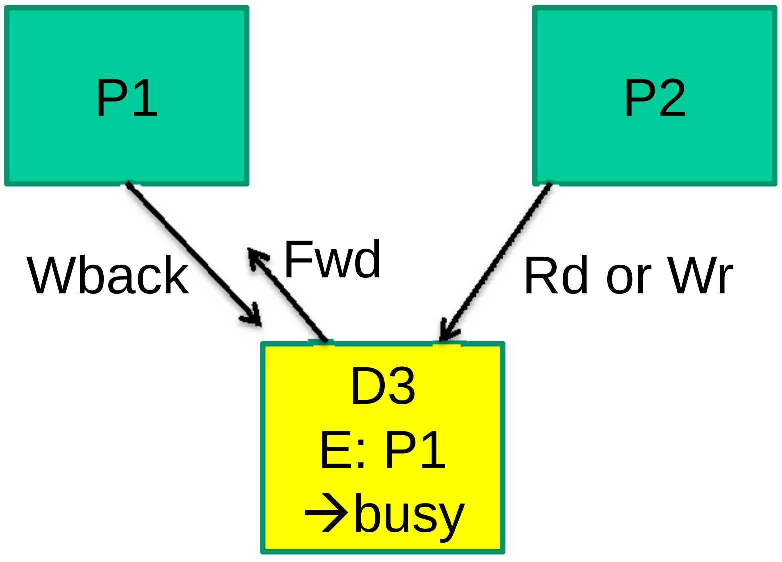
- Topics: directory-based cache coherence implementations (wrap-up of SGI Origin and Sequent NUMA case study)

Writeback Cases



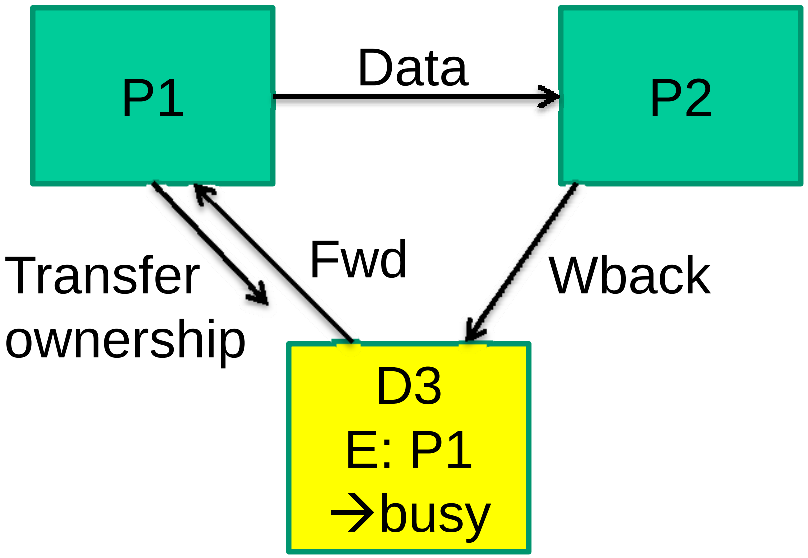
This is the “normal” case
D3 sends back an Ack

Writeback Cases



If someone else has the block in exclusive, D3 moves to busy
If Wback is received, D3 serves the requester
If we didn't use busy state when transitioning from E:P1 to E:P2,
D3 may not have known who to service
(since ownership may have been passed on to P3 and P4…)
(although, this problem can be solved by NACKing the Wback
and having P1 buffer its "strange" intervention requests)

Writeback Cases



If Wback is from new requester, D3 sends back a NACK
Floating unresolved messages are a problem
Alternatively, can accept the Wback and put D3 in some new busy state

Conclusion: could have got rid of busy state between E:P1 → E:P2, but with Wback ACK/NACK and other buffering
could have kept the busy state between E:P1 → E:P2, could have got rid of ACK/NACK, but need one new busy state

Directory Structure

- The system supports either a 16-bit or 64-bit directory (fixed cost)
- For small systems, the directory works as a full bit vector representation
- For larger systems, a coarse vector is employed – each bit represents p/64 nodes
- State is maintained for each node, not each processor – the communication assist broadcasts requests to both processors

Page Migration

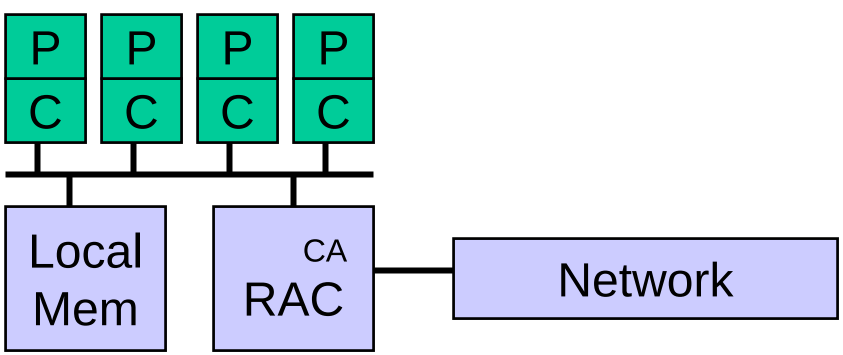
- Each page in memory has an array of counters to detect if a page has more misses from a node other than home

- When a page is moved to a different physical memory location, the virtual address remains the same, but the page table and TLBs must be updated

- To reduce the cost of TLB shootdown, the old page sets its directory state to poisoned – if a process tries to access this page, the OS intervenes and updates the translation

Sequent NUMA-Q

- Employs a flat cache-based directory protocol between nodes – IEEE standard SCI (Scalable Coherent Interface) protocol
- Each node is a 4-way SMP with a bus-based snooping protocol
- The communication assist includes a large "remote access cache" – the directory protocol tries to keep the remote caches coherent, while the snooping protocol ensures that each processor cache is kept coherent with the remote access cache



Directory Structure

- The physical address identifies the home node – the home node directory stores a pointer to the head of a linked list – each cache stores pointers to the next and previous sharer

- A main memory block can be in three directory states:
  - Home: (similar to unowned) the block does not exist in any remote access cache (may be in the home node's processor caches, though)
  - Fresh: (similar to shared) read-only copies exist in remote access caches and memory copy is up-to-date
  - Gone: (similar to exclusive) writeable copy exists in some remote cache

Cache Structure

- 29 stable states and many more pending/busy states!
- The stable states have two descriptors:
  - position in linked list: ONLY, HEAD, TAIL, MID
  - state within cache: dirty, clean, fresh, etc.
- SCI defines and implements primitive operations to facilitate linked list manipulations:
  - List construction: add a new node to the list head
  - Rollout: remove a node from a list
  - Purging: invoked by the head to invalidate all other nodes

Handling Read Requests

- On a read miss, the remote cache sets up a block in busy state and other requests to the block are not entertained

- The requestor sends a "list construction request" to the home and the steps depend on the directory state:
    - Home: state updated to fresh, head updated to requestor, data sent to requestor, state at requestor is set to ONLY_FRESH
    - Fresh: head updated to requestor, home responds with data and pointer to old head, requestor moves to a different busy state, sends list construction request to old head, old head moves from HEAD_FRESH to MID_VALID, sends ack, requestor → HEAD_FRESH

Handling Read Requests II

- Gone: home does not reply with data, it remains in Gone state, sends old head pointer to requestor, requestor moves to a different busy state, asks old head for data and "list construction", old head moves from HEAD_DIRTY to MID_VALID, returns data, requestor moves to HEAD_DIRTY (note that HEAD_DIRTY does not mean exclusive access; the head can write without talking to the home, but sharers must be invalidated)

- Home keeps forwarding requests to head even if head is busy – this results in a pending linked list that is handled as transactions complete

Handling Write Requests

- At all times, the head of a list is assumed to have the latest copy and only the head is allowed to write
- The writer starts by moving itself to the head of the list; actions depend on the state in the cache:
    - HEAD_DIRTY: the home is already in GONE state, so home is not informed, sharing list is purged (each list element invalidates itself and informs the requestor of the next element – simple, but slow – works well for small invalidation sizes)

Handling Write Requests II

- HEAD_FRESH: home directory is updated from FRESH to GONE, sharing list is purged; if the home directory is not in FRESH state, some other node's request is in flight – the requestor will have to move to the head again and retry

- ONLY_DIRTY: the write happens without generating any interconnect traffic

Writeback & Replacement

- Replacements are no longer “quiet” as the linked lists have to be updated – the “rollout” operation is used

- To rollout, a node must set itself to pending, inform the neighbors, and set itself to invalid – to prevent deadlock in the case of two neighbors attempting rollout, the node closer to the tail is given priority

- If the node is the head, it makes the next element the head and informs home

Writeback & Replacement II

- If the head is attempting a rollout, it sends a message home, but the home is pointing to a different head: the old head will eventually receive a request from the new head – at this point, the writeback is complete, and the new head is instead linked with the next node

- To reduce buffering needs, the writeback happens before the new block is fetched

Serialization

- The home serves as the point of serialization – note that requests are almost never NACKed – requests are usually re-directed to the current head – helps avoid race conditions

- Since requests get queued in a pending list and buffers are rarely used, the protocol is less prone to starvation, unfairness, deadlock, and livelock problems

Title

- Bullet