Lecture 4: Directory Protocols

- Topics: directory-based cache coherence implementations

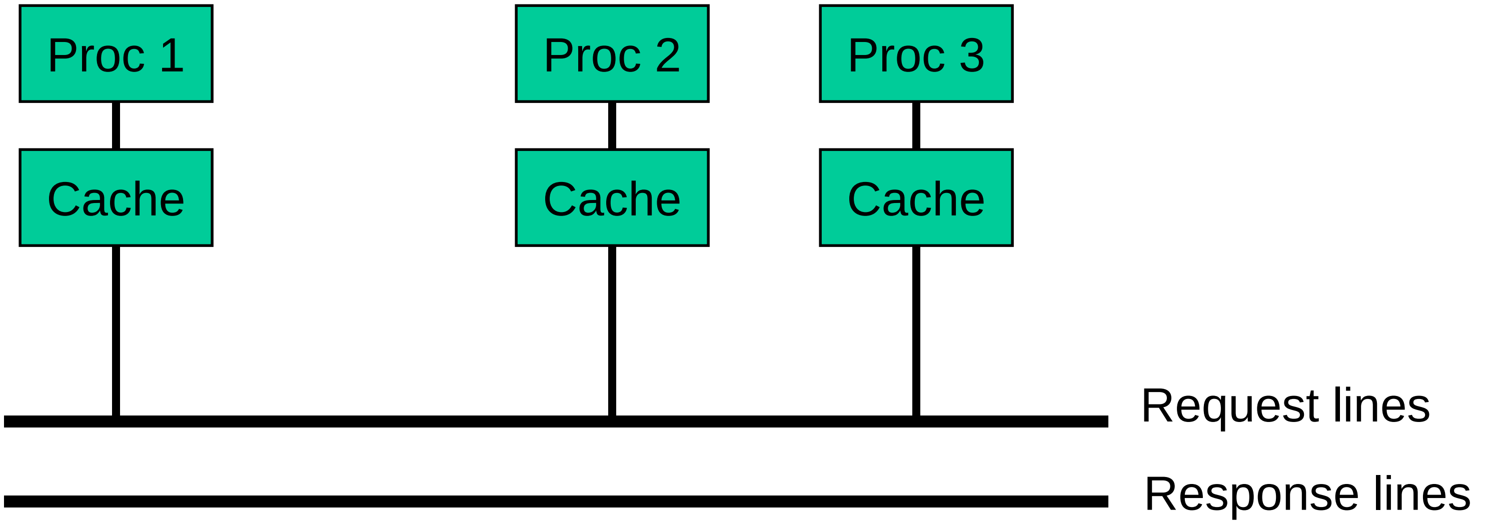
Split Transaction Bus

- What would it take to implement the protocol correctly while assuming a split transaction bus?

- Split transaction bus: a cache puts out a request, releases the bus (so others can use the bus), receives its response much later

- Assumptions:
  - only one request per block can be outstanding
  - separate lines for addr (request) and data (response)

Split Transaction Bus



Design Issues

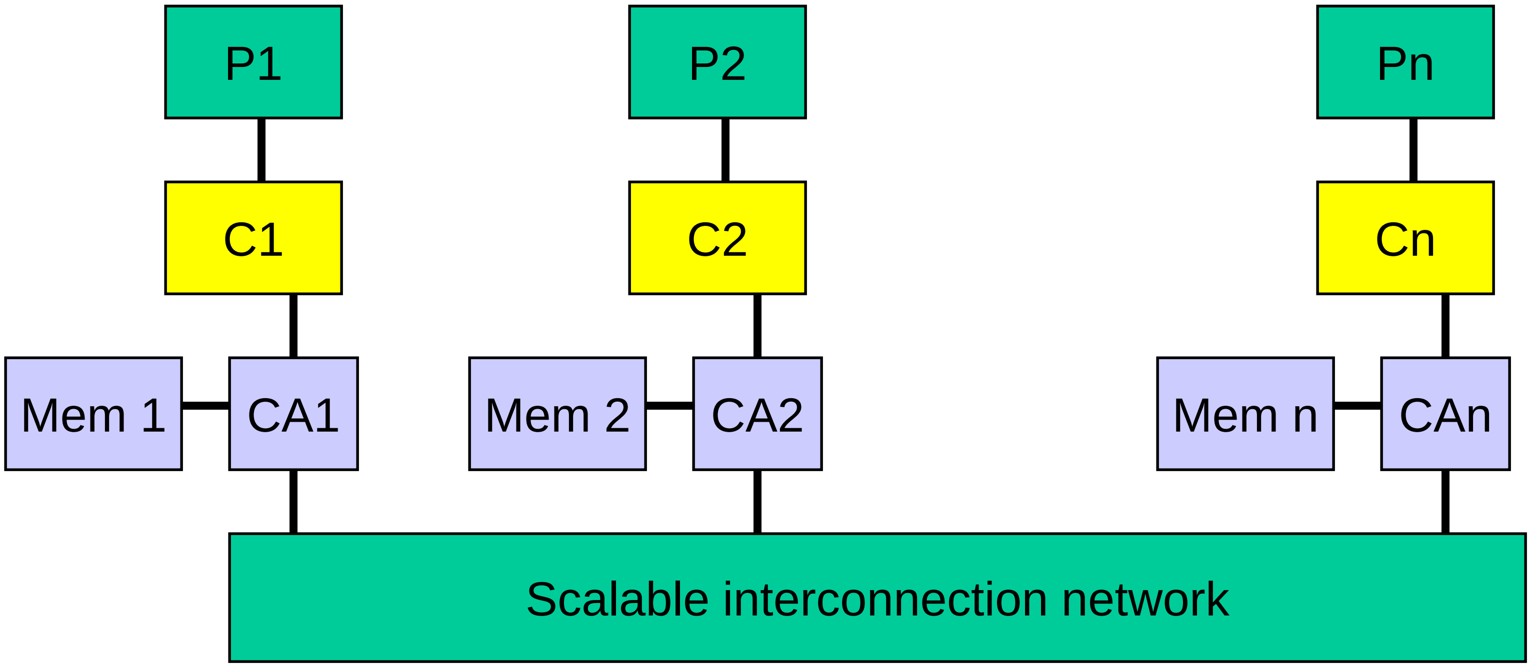
- When does the snoop complete? What if the snoop takes a long time?
- What if the buffer in a processor/memory is full? When does the buffer release an entry? Are the buffers identical?
- How does each processor ensure that a block does not have multiple outstanding requests?
- What determines the write order – requests or responses?

Design Issues II

- What happens if a processor is arbitrating for the bus and witnesses another bus transaction for the same address?

- If the processor issues a read miss and there is already a matching read in the request table, can we reduce bus traffic?

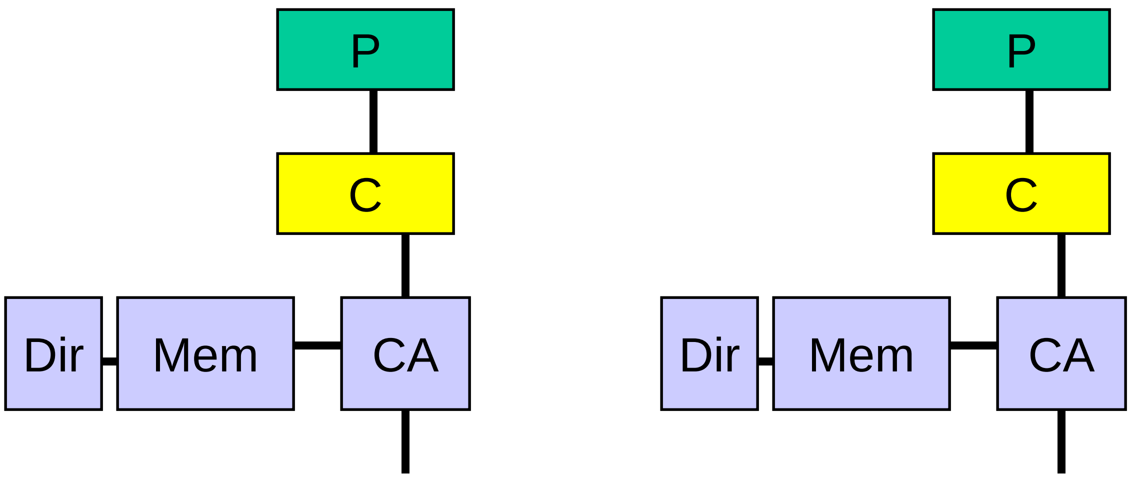
Scalable Multiprocessors



CC NUMA: Cache coherent non-uniform memory access

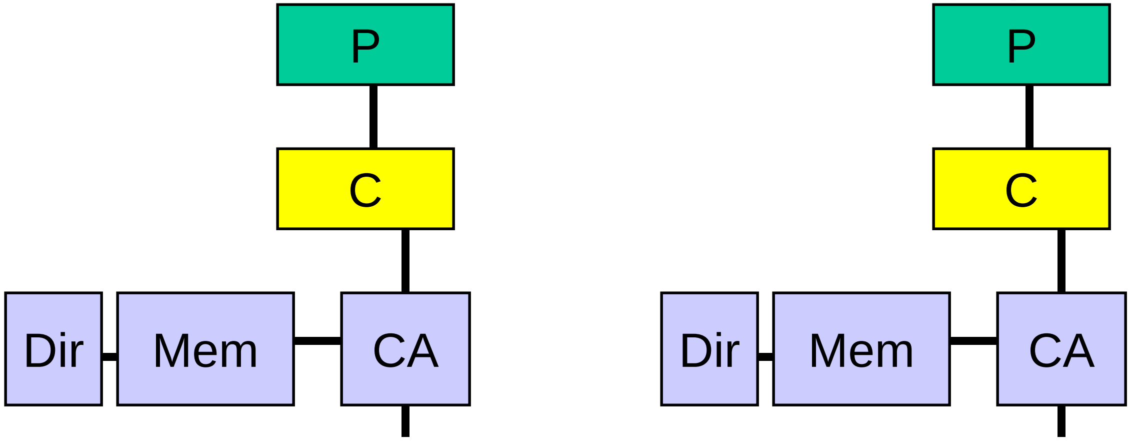
Directory-Based Protocol

- For each block, there is a centralized "directory" that maintains the state of the block in different caches
- The directory is co-located with the corresponding memory
- Requests and replies on the interconnect are no longer seen by everyone – the directory serializes writes

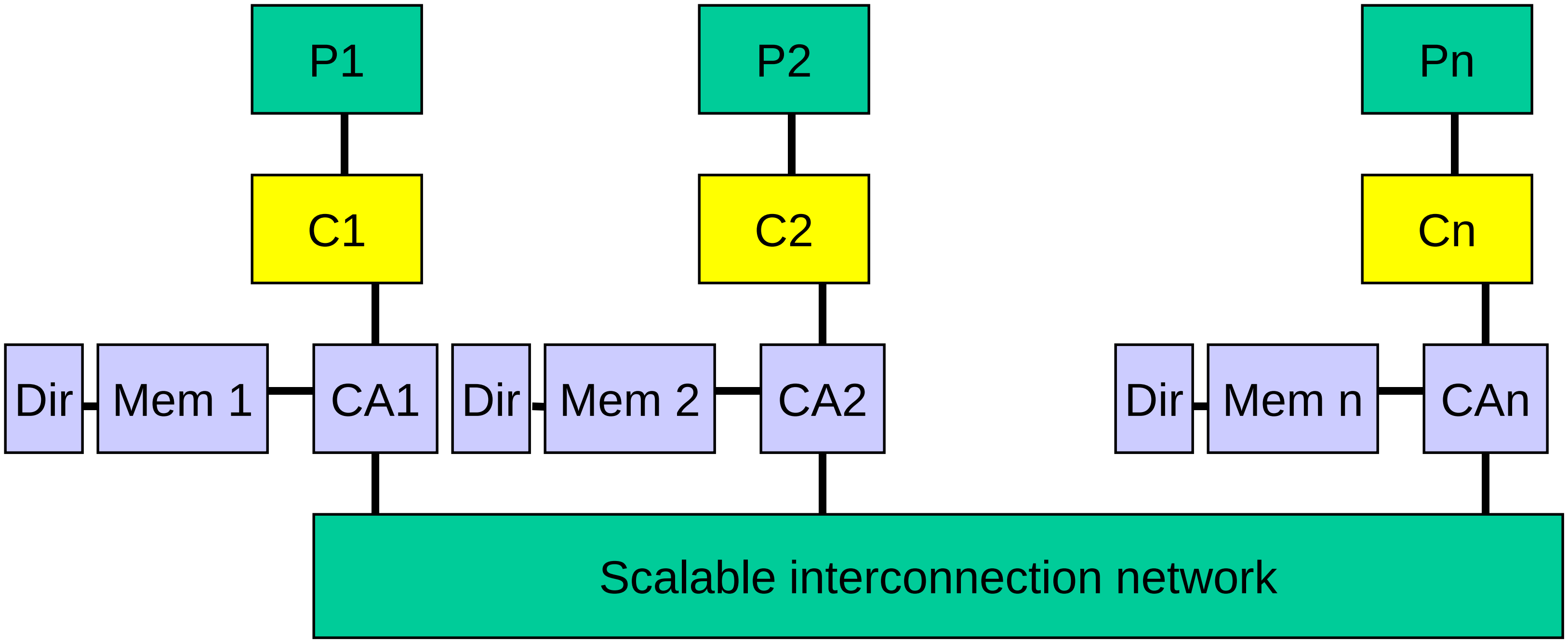


Definitions

- Home node: the node that stores memory and directory state for the cache block in question
- Dirty node: the node that has a cache copy in modified state
- Owner node: the node responsible for supplying data (usually either the home or dirty node)
- Also, exclusive node, local node, requesting node, etc.



Protocol Steps



- What happens on a read miss and a write miss?
- How is information stored in a directory?

Directory Organizations

- Centralized Directory: one fixed location – bottleneck!
- Flat Directories: directory info is in a fixed place, determined by examining the address – can be further categorized as memory-based or cache-based
- Hierarchical Directories: the processors are organized as a logical tree structure and each parent keeps track of which of its immediate children has a copy of the block – less storage (?), more searching, can exploit locality

Flat Memory-Based Directories

- Directory is associated with memory and stores info for all cache copies

- A presence vector stores a bit for every processor, for every memory block – the overhead is a function of memory/block size and #processors

- Reducing directory overhead:

Flat Memory-Based Directories

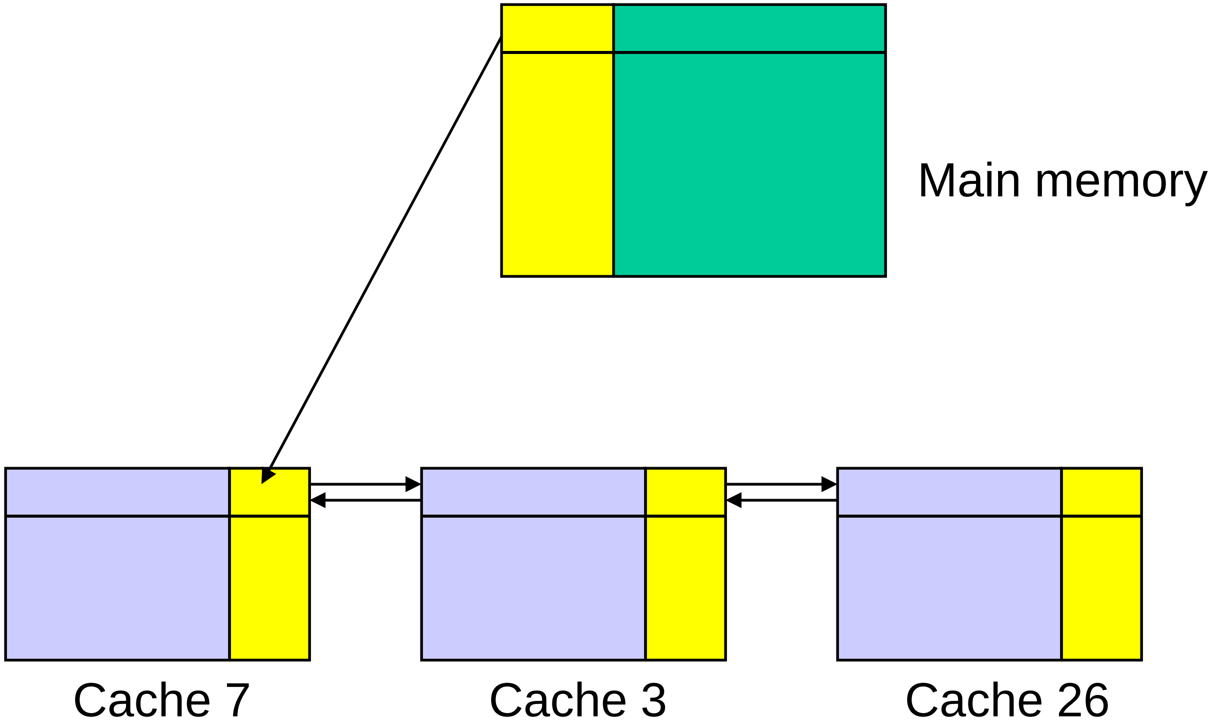
- Directory is associated with memory and stores info for all cache copies

- A presence vector stores a bit for every processor, for every memory block – the overhead is a function of memory/block size and #processors

- Reducing directory overhead:
  - Width: pointers (keep track of processor ids of sharers) (need overflow strategy), 2-level protocol to combine info for multiple processors
  - Height: increase block size, track info only for blocks that are cached (note: cache size << memory size)

Flat Cache-Based Directories

- The directory at the memory home node only stores a pointer to the first cached copy – the caches store pointers to the next and previous sharers (a doubly linked list)



Flat Cache-Based Directories

- The directory at the memory home node only stores a pointer to the first cached copy – the caches store pointers to the next and previous sharers (a doubly linked list)

- Potentially lower storage, no bottleneck for network traffic,

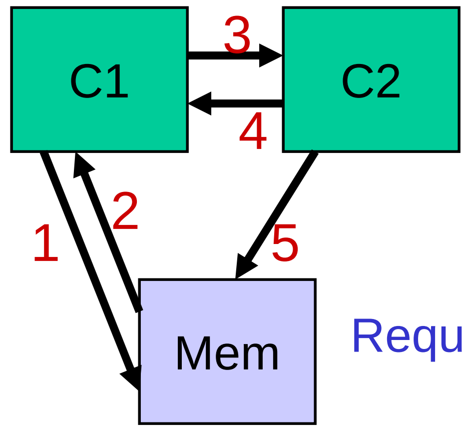
- Invalidates are now serialized (takes longer to acquire exclusive access), replacements must update linked list, must handle race conditions while updating list

Data Sharing Patterns

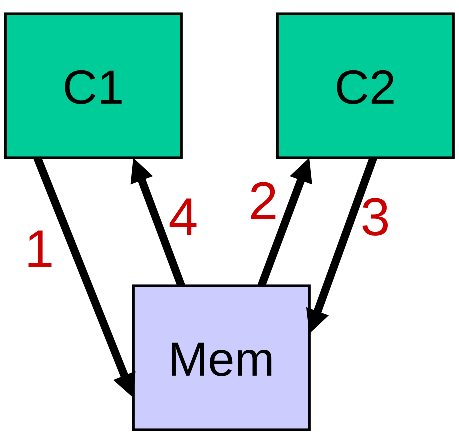
- Two important metrics that guide our design choices: invalidation frequency and invalidation size – turns out that invalidation size is rarely greater than four
- Read-only data: constantly read, never updated (raytrace)
- Producer-consumer: flag-based synchronization, updates from neighbors (Ocean)
- Migratory: reads and writes from a single processor for a period of time (global sum)
- Irregular: unpredictable accesses (distributed task queue)

Protocol Optimizations

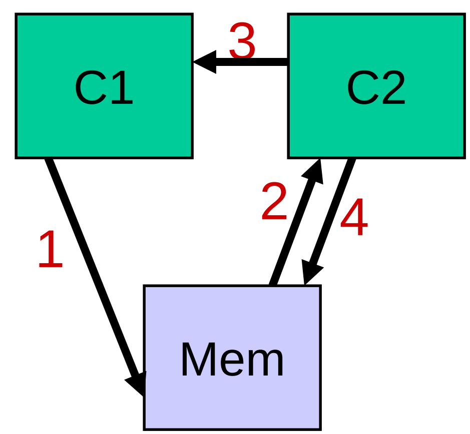
C1 attempts to read a block that is in Modified state in C2



Request Response



Intervention Forwarding



Reply Forwarding

Serializing Writes for Coherence

- Potential problems: updates may be re-ordered by the network; General solution: do not start the next write until the previous one has completed

- Strategies for buffering writes:
  - buffer at home: requires more storage at home node
  - buffer at requestors: the request is forwarded to the previous requestor and a linked list is formed
  - NACK and retry: the home node nacks all requests until the outstanding request has completed

Title

- Bullet