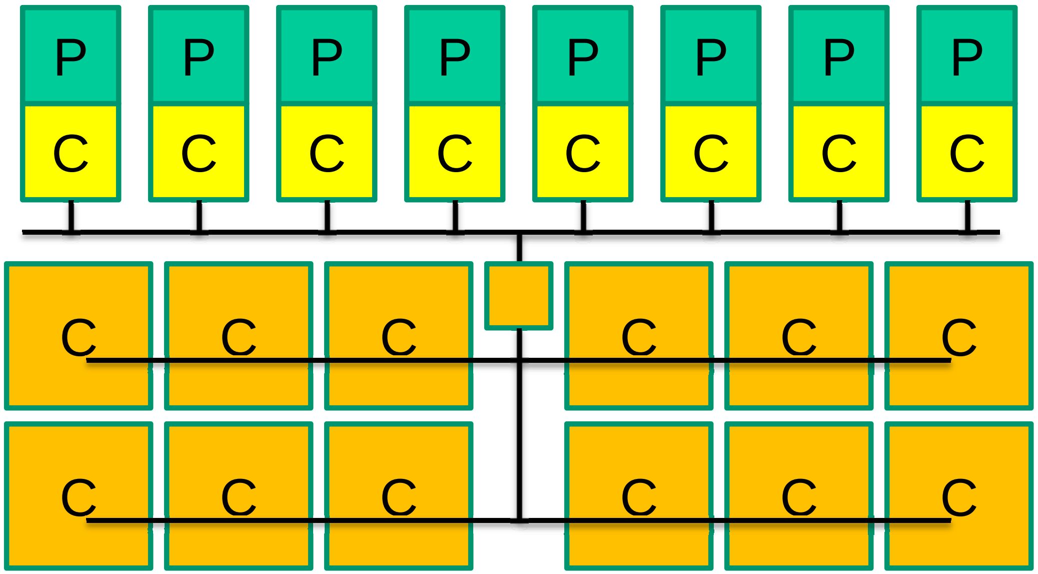
Lecture 3: Directory-Based Coherence

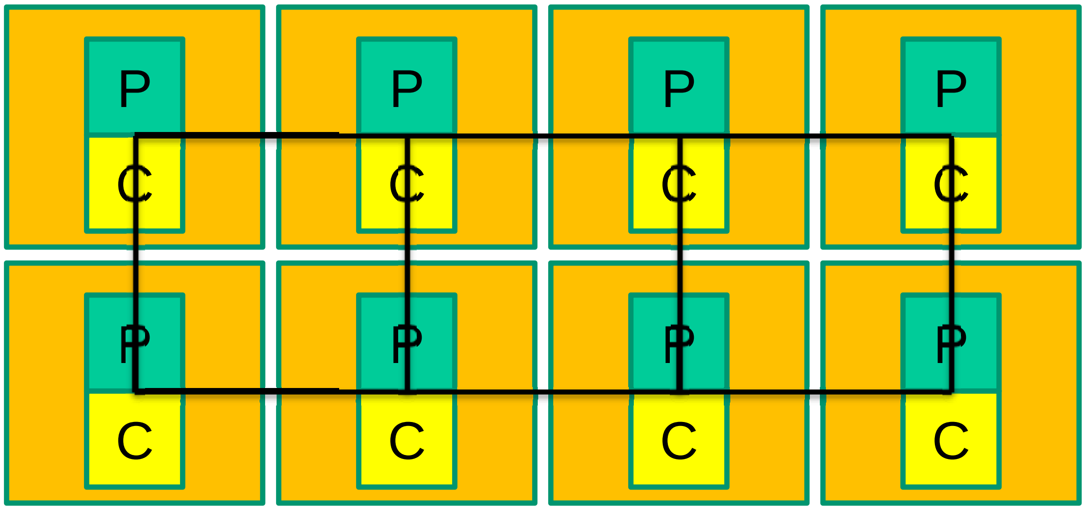
- Basic operations, memory-based and cache-based directories

Multi-Core Cache Organizations



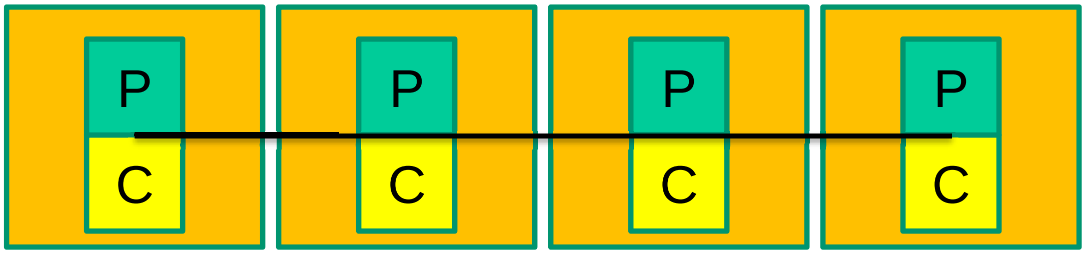
Private L1 caches
Shared L2 cache
Bus between L1s and single L2 cache controller
Snooping-based coherence between L1s

Multi-Core Cache Organizations



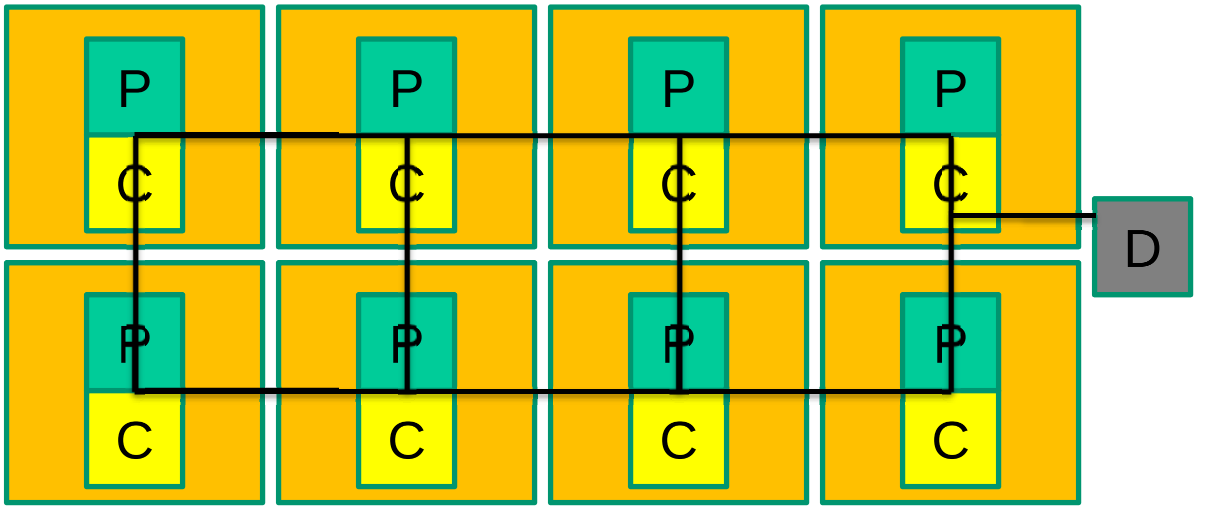
Private L1 caches
Shared L2 cache, but physically distributed
Scalable network
Directory-based coherence between L1s

Multi-Core Cache Organizations



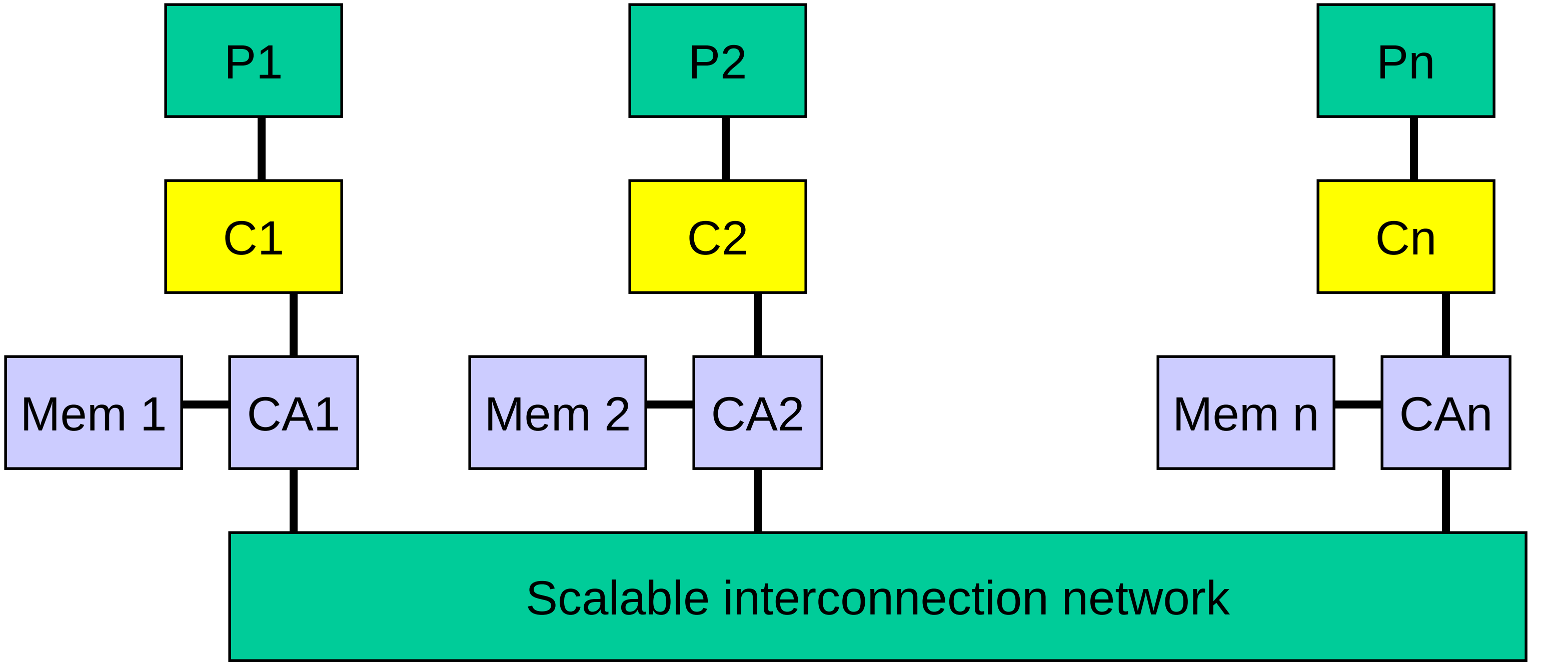
Private L1 caches
Shared L2 cache, but physically distributed
Bus connecting the four L1s and four L2 banks
Snooping-based coherence between L1s

Multi-Core Cache Organizations



Private L1 caches
Private L2 caches
Scalable network
Directory-based coherence between L2s
(through a separate directory)

Scalable Multiprocessors



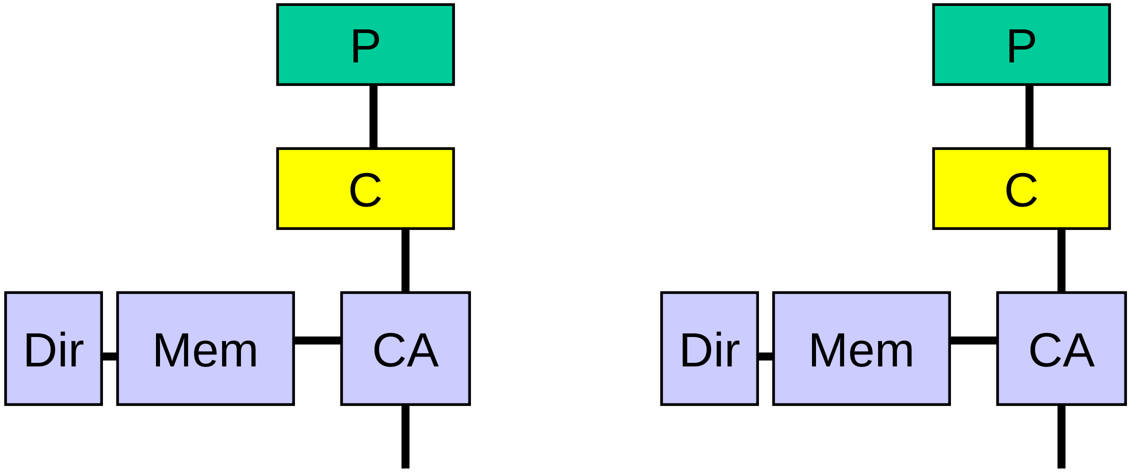
CC NUMA: Cache coherent non-uniform memory access

Directory-Based Protocol

- For each block, there is a centralized “directory” that maintains the state of the block in different caches

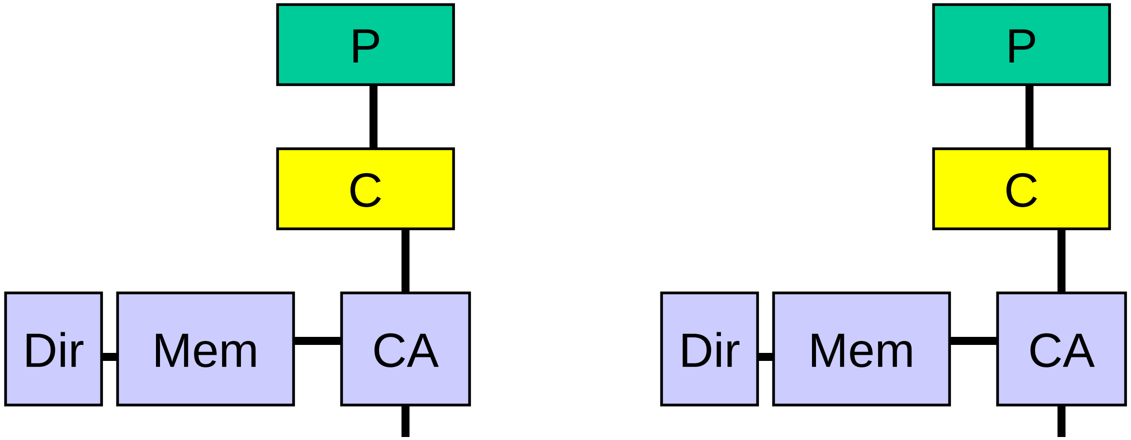
- The directory is co-located with the corresponding memory

- Requests and replies on the interconnect are no longer seen by everyone – the directory serializes writes

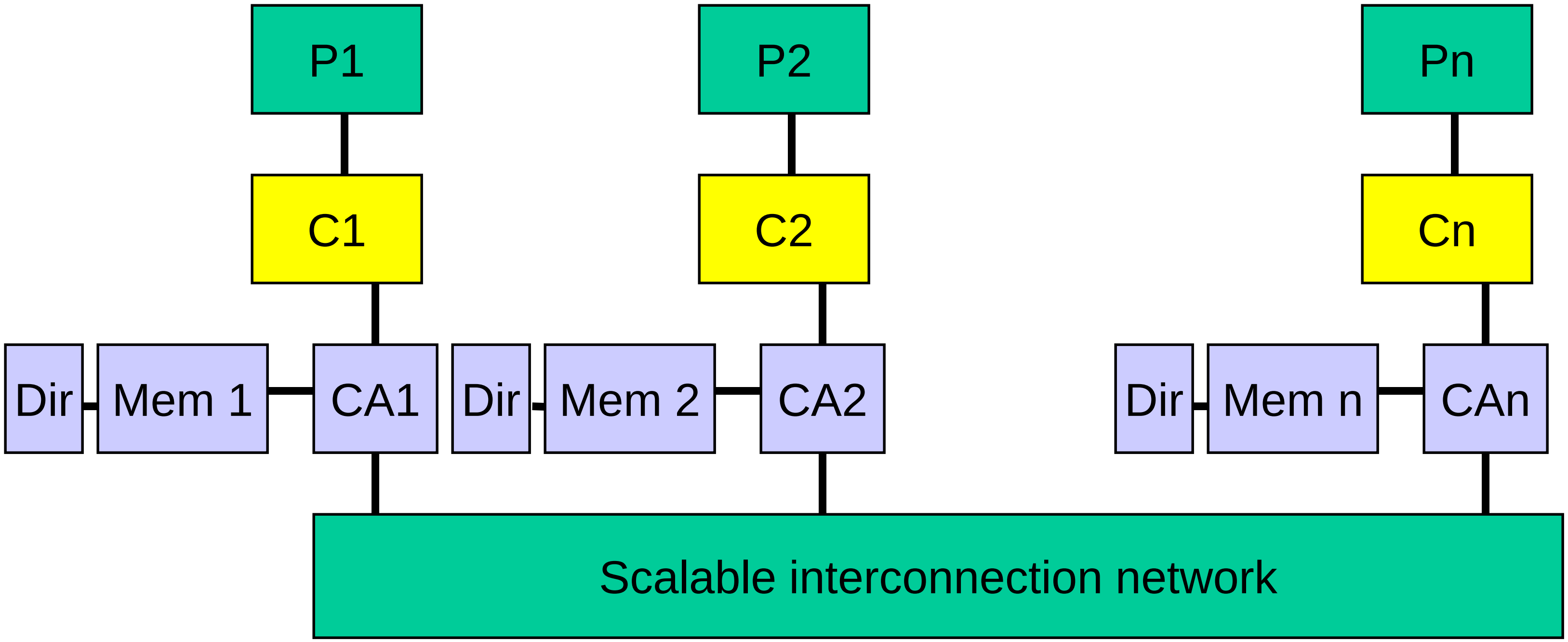


Definitions

- 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

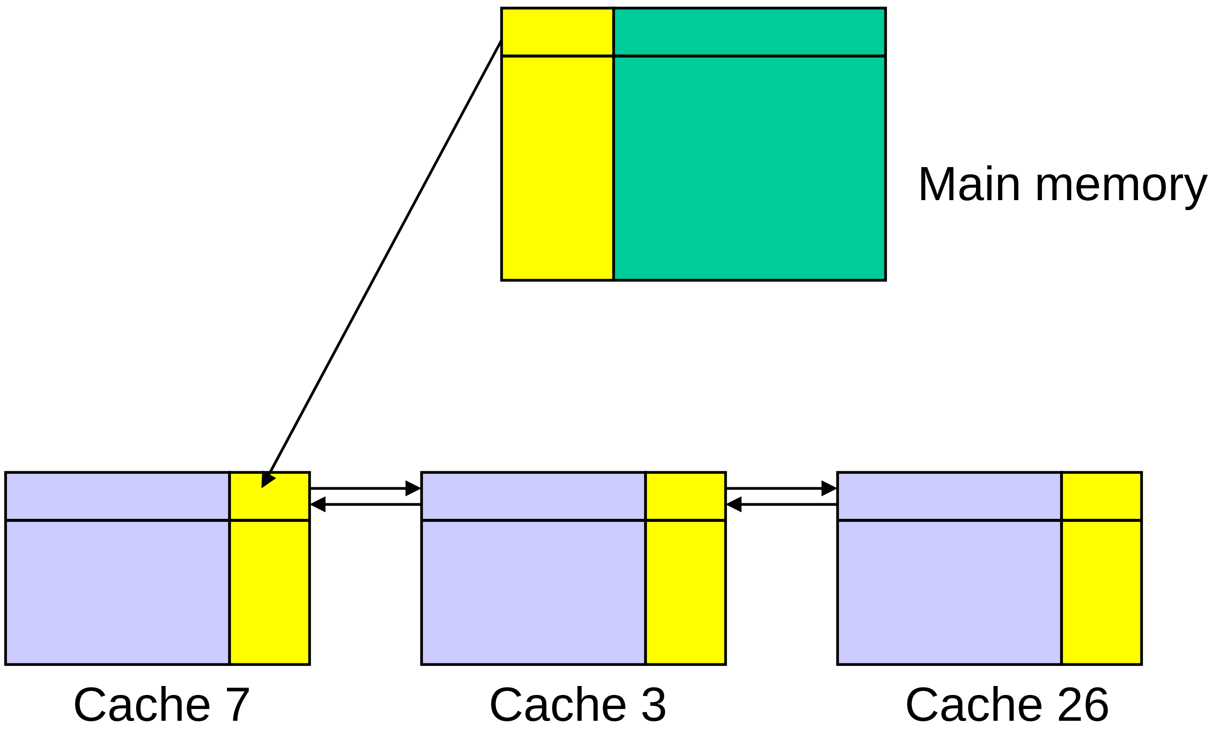
- Directory is associated with memory and stores info for all cache copies

- 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)

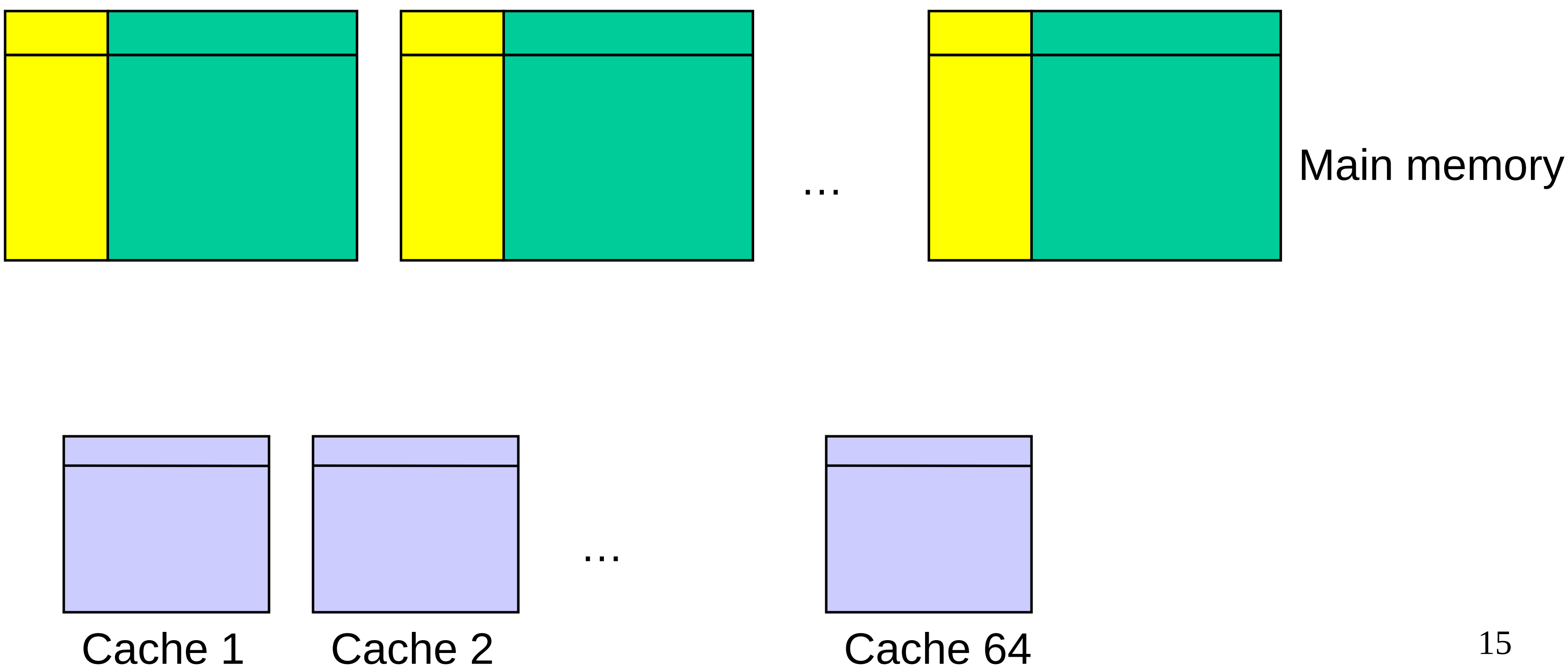
- Potentially lower storage, no bottleneck for network traffic

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

Flat Memory-Based Directories

Block size = 128 B
Memory in each node = 1 GB
Cache in each node = 1 MB

For 64 nodes and 64-bit directory,
Directory size = 4 GB
For 64 nodes and 12-bit directory,
Directory size = 0.75 GB

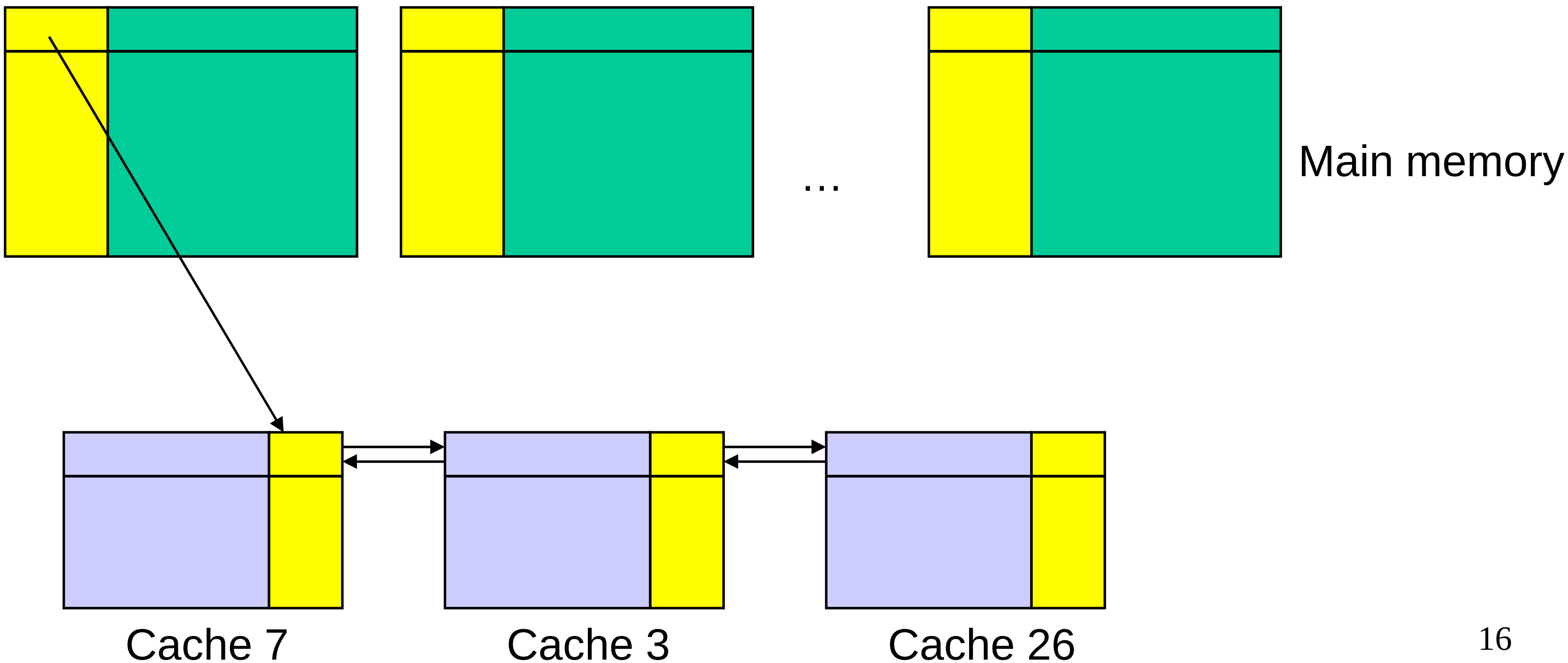


Flat Cache-Based Directories

Block size = 128 B
Memory in each node = 1 GB
Cache in each node = 1 MB

6-bit storage in DRAM for each block;
DRAM overhead = 0.375 GB

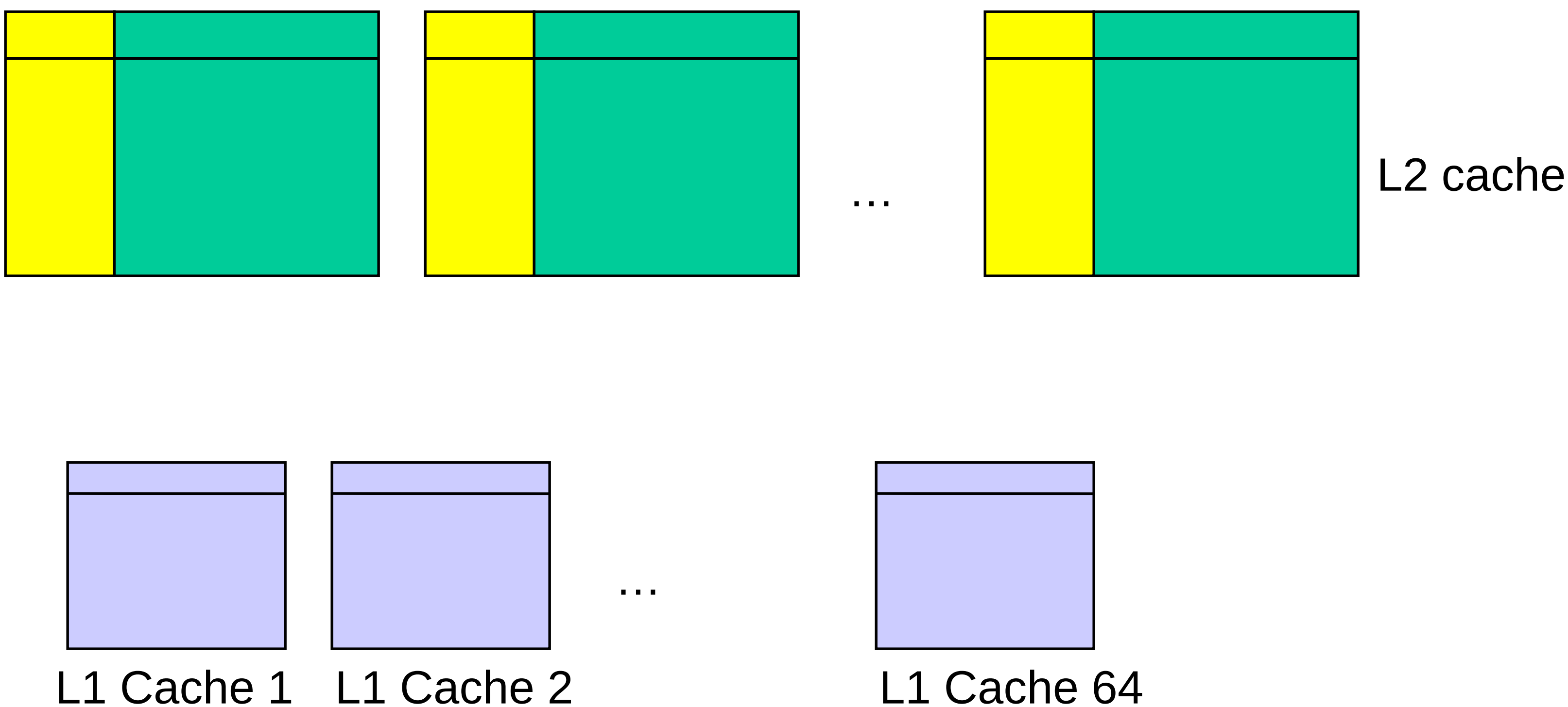
12-bit storage in SRAM for each block;
SRAM overhead = 0.75 MB



Flat Memory-Based Directories

Block size = 64 B
L2 cache in each node = 1 MB
L1 Cache in each node = 64 KB

For 64 nodes and 64-bit directory,
Directory size = 8 MB
For 64 nodes and 12-bit directory,
Directory size = 1.5 MB

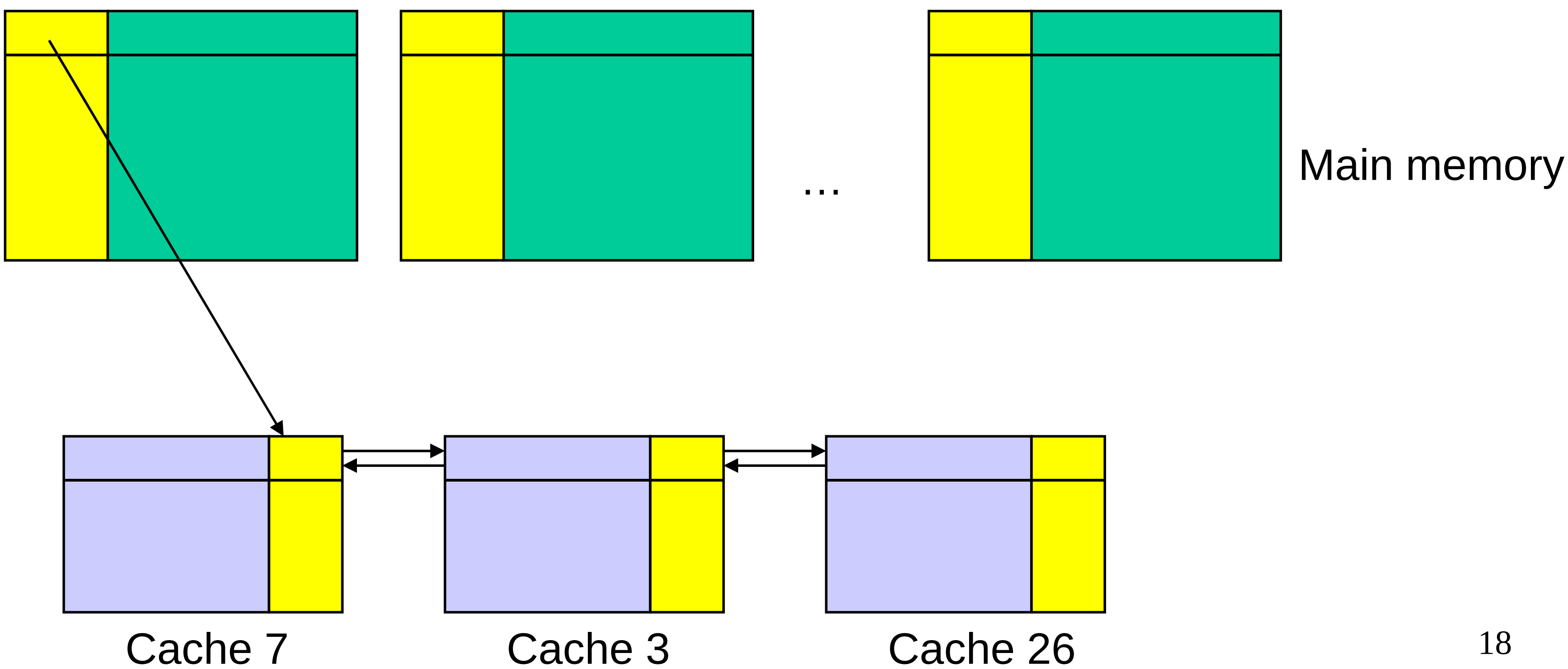


Flat Cache-Based Directories

Block size = 64 B
L2 cache in each node = 1 MB
L1 Cache in each node = 64 KB

6-bit storage in L2 for each block;
L2 overhead = 0.75 MB

12-bit storage in L1 for each block;
L1 overhead = 96 KB

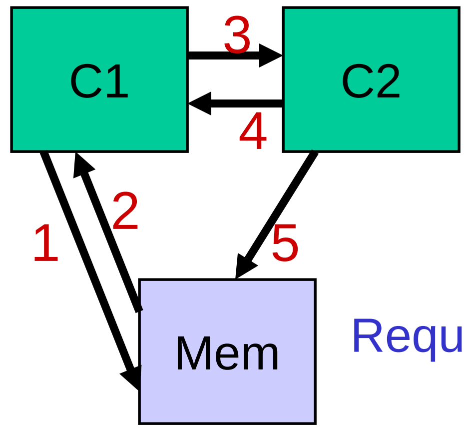


Data Sharing Patterns

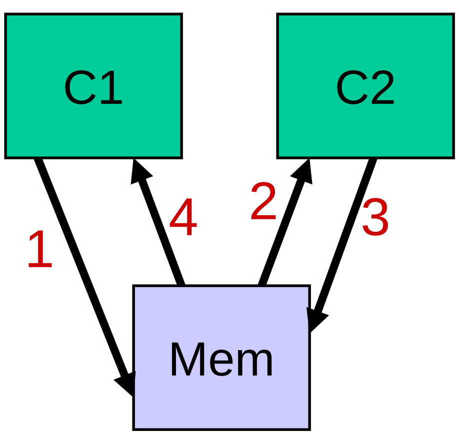
- Two important metrics that guide our design choices: invalidation frequency and invalidation size – turns out that invalidation size is rarely greater than four
- Read-only data: constantly read, never updated (raytrace)
- Producer-consumer: flag-based synchronization, updates from neighbors (Ocean)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

Protocol Optimizations

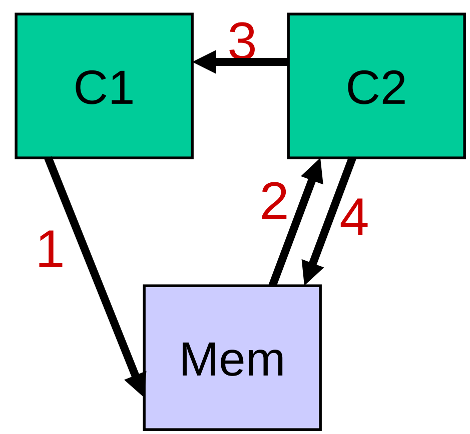
C1 attempts to read a block that is in Modified state in C2



Request Response



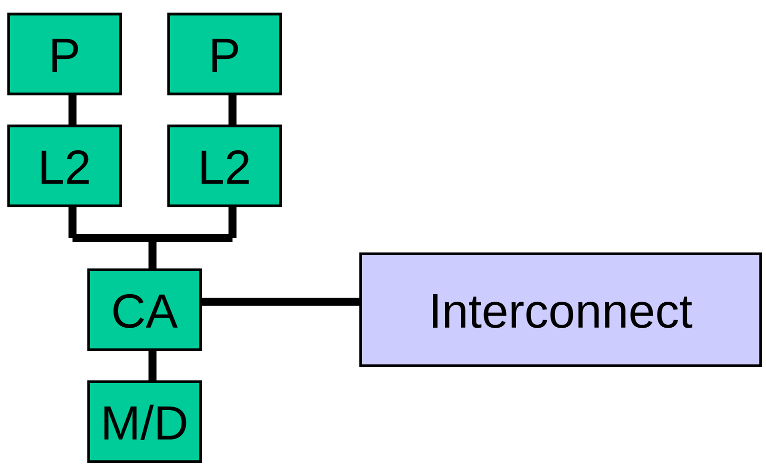
Intervention Forwarding



Reply Forwarding

SGI Origin 2000

- Flat memory-based directory protocol
- Uses a bit vector directory representation
- Two processors per node – combining multiple processors in a node reduces cost



Protocol States

- Each memory block has seven states
- Three stable states: unowned, shared, exclusive (either dirty or clean)
- Three busy states indicate that the home has not completed the previous request for that block (read, read-excl or upgrade, uncached read)
- Poison state – used for lazy TLB shootdown

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

Handling Reads

- When the home receives a read request, it looks up memory (speculative read) and directory in parallel

- Actions taken for each directory state:
  - shared or unowned: memory copy is clean, data is returned to requestor, state is changed to excl if there are no other sharers
  - busy: a NACK is sent to the requestor
  - exclusive: home is not the owner, request is fwded to owner, owner sends data to requestor and home

Inner Details of Handling the Read

- The block is in exclusive state – memory may or may not have a clean copy – it is speculatively read anyway

- The directory state is set to busy-exclusive and the presence vector is updated

- In addition to fwding the request to the owner, the memory copy is speculatively forwarded to the requestor
  - Case 1: excl-dirty: owner sends block to requestor and home, the speculatively sent data is over-written
  - Case 2: excl-clean: owner sends an ack (without data) to requestor and home, requestor waits for this ack before it moves on with speculatively sent data

Inner Details II

- Why did we send the block speculatively to the requestor if it does not save traffic or latency?
  - the R10K cache controller is programmed to not respond with data if it has a block in excl-clean state
  - when an excl-clean block is replaced from the cache, the directory need not be updated – hence, directory cannot rely on the owner to provide data and speculatively provides data on its own

Handling Write Requests

- The home node must invalidate all sharers and all invalidations must be acked (to the requestor), the requestor is informed of the number of invalidates to expect

- Actions taken for each state:
  - shared: invalidates are sent, state is changed to excl, data and num-sharers is sent to requestor, the requestor cannot continue until it receives all acks (Note: the directory does not maintain busy state, subsequent requests will be fwded to new owner and they must be buffered until the previous write has completed)

Handling Writes II

- Actions taken for each state:
  - unowned: if the request was an upgrade and not a read-exclusive, is there a problem?
  - exclusive: is there a problem if the request was an upgrade? In case of a read-exclusive: directory is set to busy, speculative reply is sent to requestor, invalidate is sent to owner, owner sends data to requestor (if dirty), and a “transfer of ownership” message (no data) to home to change out of busy
  - busy: the request is NACKed and the requestor must try again

Title

- Bullet