A Cache Coherent System Must:

• Manage coherence protocol
  – (0) Determine when to invoke coherence protocol
  – (a) Find info about state of block in other caches to determine action
    - whether need to communicate with other cached copies
  – (b) Locate the other copies
  – (c) Communicate with those copies (invalidate/update)

• (0) is done the same way on all systems
  – state of the line is maintained in the cache
  – protocol is invoked if an access fault occurs on the line

• Different approaches distinguished by how (a) to (c) are handled

Directory-based Cache Coherence

Adapted from slides of David Patterson
University of California, Berkeley, CS 252

Bus-based Coherence

• All of (a), (b), (c) done through broadcast on bus
  – Faulting processor sends out a “search”
  – others respond to the search probe and take necessary action

• Conceptually simple, but broadcast doesn’t scale with p
  – on bus, bus bandwidth doesn’t scale
  – on scalable networks, every fault leads to at least p network transactions

• Scalable coherence:
  – Can use same cache states and state transition diagram
  – Different mechanisms to manage protocol

Scalable Approach: Directories

• Every memory block has associated directory information
  – keeps track of copies of cached blocks and their states
  – on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  – in scalable networks, communication with directory and copies is through network transactions

• Many alternatives for organizing directory information
### Basic Operation of Directory

- **k processors.**
- With each cache-block in memory:
  - k presence-bits, 1 dirty-bit
- With each cache-block in cache:
  - 1 valid bit, and 1 dirty (owner) bit

### Simple Directory Protocol for NUMA/Distributed Memory Arch

- **Similar to Snoopy Protocol:**
  - Three states:
    - **Shared:** ≥ 1 processors have data; memory up-to-date
    - **Uncached:** (no processor has it; not valid in any cache)
    - **Exclusive:** 1 processor (owner) has data; memory out-of-date
  - In addition to cache state, must track which processors have data when in the shared state (usually bit vector, 1 if processor has copy)
  - **Keep it simple:**
    - Writes to non-exclusive data:
      - write miss
    - Processor blocks until access completes
    - Assume messages received and acted upon in order sent

### Simple Directory Protocol for NUMA

- **Terms:** typically 3 processors involved
  - *Local node* where a request originates
  - *Home node* where the memory location of an address resides (and the directory entry for the block)
  - *Remote node* has a copy of a cache block, whether exclusive or shared
  - Note: one processor may hold multiple roles at any given time
- **Communication through explicit protocol messages from processor to processor**
  - Interconnect no longer single arbitration point (scalable)
  - All messages have explicit responses
  - All messages processed in the order received
- **Example messages on next slide:**
  - P = processor number, A = address

---

**Quiz questions:**

What are bit settings under various scenarios:

P1, P2, P3 reading then P3 writes
### Directory Protocol Messages (Fig 4.20)

<table>
<thead>
<tr>
<th>Message type</th>
<th>Source</th>
<th>Destination</th>
<th>Msg Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
</tr>
<tr>
<td>Write miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
</tr>
<tr>
<td>Invalidate</td>
<td>Home directory</td>
<td>Remote caches</td>
<td>A</td>
</tr>
<tr>
<td>Fetch</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td>Fetch/Invalidate</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td>Data value reply</td>
<td>Home directory</td>
<td>Local cache</td>
<td>Data</td>
</tr>
<tr>
<td>Data write back</td>
<td>Remote cache</td>
<td>Home directory</td>
<td>A, Data</td>
</tr>
</tbody>
</table>

- **Read miss**: Processor P reads data at address A; make P a read sharer and request data.
- **Write miss**: Processor P has a write miss at address A; make P the exclusive owner and request data.
- **Invalidate**: Invalidate a shared copy at address A.
- **Fetch, Fetch/Invalidate**: Fetch block at address A and send it to its home directory; if required, invalidate the block in the cache.
- **Data value reply**: Return a data value from the home memory (read miss response).
- **Data write back**: Write back a data value for address A (invalidate response).

### State Transition Diagram for One Cache Block in Directory Based System

- States identical to snoopy case; transactions very similar.
- Transitions caused by read misses, write misses, invalidates, data fetch requests.
- Generates read miss & write miss message to home directory.
- Write misses that were broadcast on the bus for snooping ⇒ explicit invalidate & data fetch requests.

### CPU - Cache State Machine

- State machine for CPU requests for each memory block.
- State transitions for CPU read and write operations.
- States: Invalid, Shared (read/write), Exclusive (read/write), Shared (read only).

### State Transition Diagram for Directory

- Same states & structure as the transition diagram for an individual cache.
- 2 actions: update of directory state & send messages to satisfy requests.
- Tracks all copies of memory block.
- Also indicates an action that updates the sharing set, Sharers, as well as sending a message.
**Directory State Machine**

- State machine for Directory requests for each memory block.

**Directory Protocol Details**

- Message sent to directory causes two actions:
  - Update the directory
  - More messages to satisfy request

- Block is in Uncached state: the copy in memory is the current value; only possible requests for that block are:
  - Read miss: requesting processor sent data from memory & requestor made only sharing node; state of block made Shared.
  - Write miss: requesting processor is sent the value & becomes the Sharing node. The block is made Exclusive to indicate that the only valid copy is cached. Sharers indicates the identity of the owner.

- Block is Shared ⇒ the memory value is up-to-date:
  - Read miss: requesting processor is sent back the data from memory & requesting processor is added to the sharing set.
  - Write miss: requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages, & Sharers is set to identity of requesting processor. The state of the block is made Exclusive.

**Example**

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>Processor 2</th>
<th>Interconnect</th>
<th>Directory Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A1 and A2 map to the same cache block.

3/13/07 CIS 429/529 W’07: Directory-based CC
A1 and A2 map to the same cache block

A1 and A2 map to the same cache block
### Example

A1 and A2 map to the same cache block (but different memory block addresses A1 ≠ A2)

### Implementing a Directory

- We assume operations atomic, but they are not; reality is much harder; must avoid deadlock when run out of buffers in network (see Appendix E)
- Optimizations:
  - E.g. in the case of read miss or write miss in Exclusive: send data directly to requestor from owner vs. 1st to memory and then from memory to requestor

### Basic Directory Transactions

1. Request
2. Directory node for block
3a. Read request to directory
3b. Reply with owner identity
4a. Read req. to owner
4b. Data
5. Message to directory
6a. Inval. req. to owner
6b. Inval. ack
7. Requestor

### Example Directory Protocol (1st Read)

- P1: pA
- D: Read pA
- S: Ring
- P2: pA
- E: E
- S: S
- P2: P2

- Id va -> rd pA
- 3/13/07 CIS 429/529 W’07: Directory-based CC
Example Directory Protocol (Read Share)

Example Directory Protocol (Wr to shared)

Example Directory Protocol (Wr to Ex)

A Popular Middle Ground

- Two-level “hierarchy”
- Individual nodes are multiprocessors, connected non-hierarchically
  - e.g. mesh of SMPs
- Coherence across nodes is directory-based
  - directory keeps track of nodes, not individual processors
- Coherence within nodes is snooping or directory
  - orthogonal, but needs a good interface of functionality