**Cache Coherency**

**Cache coherent processors**
- most current value for an address is the last write
- all reading processors must get the most current value

**Cache coherency problem**
- update from a writing processor is not known to other processors

**Cache coherency protocols**
- (usually hardware) mechanism for maintaining cache coherency
- coherency state associated with a cache block of data
- bus/interconnect operations on shared data change the state
  - for the processor that initiates an operation
  - for other processors that have the data of the operation resident in their caches

---

**A Low-end MP**

![Diagram of a low-end MP](image)
**Cache Coherency Protocols**

**Write-invalidate**

(most multiprocessors)

- processor obtains exclusive access for writes (becomes the “owner”) by invalidating data in other processors’ caches
- coherency miss (invalidation miss)
- cache-to-cache transfers
- good for:
  - multiple writes to same word or block by one processor
  - exploits migratory sharing from processor to processor, or processor locality

---

**A Low-end MP**

Diagram showing a low-end multiprocessor system with one or more levels of cache per processor, main memory, and an I/O system.
Cache Coherency Protocols

Write-update
(SPARC Center 2000)
- broadcast each write to actively shared data
- each processor with a copy snoops/takes the data
- good for **inter-processor contention**

Competitive
(Alphas)
- switches between them

We will focus on write-invalidate.
Cache Coherency Protocol Implementations

Snooping
- used with low-end MPs
  - few processors
  - centralized memory
  - bus-based
- distributed implementation: responsibility for maintaining coherence lies with each processor cache

Directory-based
- used with higher-end MPs
  - more processors
  - distributed memory
  - multi-path interconnect
- distributed implementation: responsibility for maintaining coherence lies with the directory for each address

Snooping Implementation

A distributed coherency protocol
- coherency state associated with each cache block
- each snoop maintains coherency for its own cache
  - compare address on the bus with address in cache
  - response depends on coherency state
Snooping Implementation

How the bus is used
• broadcast medium
• entire coherency operation is atomic wrt other processors
  • keep-the-bus protocol:
    • master holds the bus until the entire operation has completed
    • do not initiate another operation while one is in progress
  • split-transaction protocol:
    • request & response are different phases
    • state values that indicate that an operation is in progress
    • do not initiate another operation for a cache block that has one in progress

Snooping Implementation

Snoop implementation:
• snoop on the highest level cache
  • another reason L2 is physically-accessed
  • property of inclusion:
    • all blocks in L1 are in L2
    • therefore only have to snoop on L2
    • may need to update L1 state if change L2 state
• separate tags & state for snoop lookups
  • processor & snoop communicate for a state or tag change
**An Example Snooping Protocol**

Invalidation-based coherency protocol
Each cache block is in one of three states
- **shared:**
  - clean in all caches & up-to-date in memory
  - block can be read by any processor
- **exclusive:**
  - dirty in exactly one cache
  - only that processor can write to it
- **invalid:**
  - block contains no valid data

**State Transitions for a Given Cache Block**

State transitions caused by:
- events caused by the **requesting processor**, e.g.,
  - read miss (go from invalid to shared)
  - write miss (go from invalid to exclusive)
  - write on shared block (go to exclusive)
- events caused by **snoops of other caches**, e.g.,
  - read miss by P1 makes P2’s owned block change from exclusive to shared
  - write miss by P1 makes P2’s owned block change from exclusive to invalid
State Machine (CPU side)

Invalid
- CPU write miss
- Place write op on bus
- CPU read hit
- CPU write hit

Shared (read/only)
- CPU read miss
- Place read op on bus
- Write-back block
- CPU read hit

Exclusive (read/write)
- CPU read miss
- Place read op on bus
- CPU write miss
- Place write op on bus
- Write-back cache block
- CPU write hit

State Machine (Bus side: the snoop)

Invalid
- Write miss for this block
- Write-back the block

Shared (read/only)
- Write miss for this block
- Write-back the block

Exclusive (read/write)
- Read miss for this block
- Write-back the block
Directory Implementation

Distributed memory machine
- processor-memory pairs are connected via a multi-path interconnection network
  - point-to-point communication
  - snooping with broadcasting is wasteful of the parallel communication capability
- each processor (or cluster of processors) has its own memory
- a processor has fast access to its local memory & slower access to "remote" memory located at other processors
  - **NUMA** (non-uniform memory access) machines

A High-end MP

![Diagram of a high-end multiprocessor system with processors, memories, directories, and interconnection network.](image-url)
Coherence on High-end Machines

How cache coherency is handled
- no caches (Cray MTA)
- disallow caching of shared data (Cray 3TD)
- software coherence (research machines)
- hardware directories that record cache block state (most others)

Directory Implementation

Coherency state is associated with units of memory that are the size of cache blocks: directory state
- directory tracks the state of cache blocks
  - shared:
    - at least 1 processor has the data cached & memory is up-to-date
    - block can be read by any processor
  - exclusive:
    - 1 processor (the owner) has the data cached & memory is stale
    - only that processor can write to it
  - invalid:
    - no processor has the data cached & memory is up-to-date
    - directory tracks the sharing of memory blocks
      - bit vector in which 1 means the processor has cached the data
      - write bit to indicate if exclusive
## Directory Implementation

Directories assign different uses to different processors
- **home** node: where the memory location of an address resides (and cached data may be there too)
- **local** node: where the memory request initiated
- **remote** node: an alternate location for the data, if this processor has requested & cached it

In satisfying a memory request:
- messages sent between the different nodes in point-to-point communication
- messages get explicit replies

Some simplifying assumptions for using the protocol
- processor blocks until the access is complete
- messages processed in the order received

---

## Read Miss for an Uncached Block

1: read miss
2: data value reply

Interconnection network
Read Miss for an Exclusive, Remote Block

1: read miss
2: fetch
4: data value reply
3: data write-back

Write Miss for an Exclusive, Remote Block

1: write miss
2: fetch & invalidate
4: data value reply
3: data write-back
### Directory Protocol Messages

<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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor P reads data at address A:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>make P a read sharer and arrange to send data back</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor P writes data at address A:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>make P the exclusive owner and arrange to send data back</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalidate</td>
<td>Home directory</td>
<td>Remote caches</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalidate a shared copy at address A.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch the block at address A and send it to its home directory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch/Invalidate</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch the block at address A and send it to its home directory; invalidate the block in the cache</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data value reply</td>
<td>Home directory</td>
<td>Local cache</td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return a data value from the home memory (read or write miss response)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data write-back</td>
<td>Remote cache</td>
<td>Home directory</td>
<td>A, Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write-back a data value for address A (invalidate response)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CPU FSM for a Cache Block

States identical to the snooping protocol

Transactions very similar

- read & write misses sent to home directory
- invalidate & data fetch requests to the node with the data replace broadcasted read/write misses
**CPU FSM for a Cache Block**

- **Invalid**
  - Fetch/Invalidate
  - Send data write-back
  - CPU read
  - Send read miss

- **Shared** (read-only)
  - CPU write
  - Send write miss
  - CPU write
  - Send invalidate (write miss)

- **Exclusive** (read/write)
  - Fetch
  - Send data write-back
  - Read miss
  - Send read miss
  - Send data write-back
  - CPU write miss
  - Send write miss message
  - Data write-back message
  - CPU write hit

**Directory FSM for a Memory Block**

- Same states and structure as for the cache block FSM
- Tracks all copies of a memory block
- Makes two state changes:
  - update coherency state
  - alter the number of sharers in the sharing set
Directory FSM for a Memory Block

- **Uncached**
  - Write miss
    - Send fetch/invalidate to owner
    - (Data write-back)
    - Send data reply
      - Sharers = \{P\}, W = 1
  - Read miss
    - Send data reply
      - Sharers = \{P\}, W = 0

- **Shared**
  - Read miss
    - Send data reply
      - Sharers += \{P\}, W = 0
  - Write miss
    - Send invalidate to all sharers
    - Send data reply
      - Sharers = \{P\}, W = 1

- **Exclusive**
  - Read miss
    - Send data reply
      - Sharers += \{P\}, W = 0
  - Write miss
    - Send fetch/invalidate to owner
      - (Data write-back)
    - Send data reply
      - Sharers = \{P\}, W = 1

False Sharing

Processors read & write to *different* words in a shared cache block
- cache coherency is maintained on a cache block basis
- processes share cache blocks, not data
- block ownership bounces between processor caches
False Sharing

Impact aggravated by:
- block size: why?
- cache size: why?
- large miss penalties: why?

Reduced by:
- coherency protocols (state per subblock)
  - let cache blocks become incoherent as long as there is only false sharing
  - make them coherent if any processor true shares
- compiler optimizations (group & transpose, cache block padding)
- cache-conscious programming wrt initial data structure layout