Cache Coherence

<table>
<thead>
<tr>
<th>Refresh Type</th>
<th>Example Shortcuts</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soft Refresh</td>
<td>GMail [Refresh] button</td>
<td>Requests update within JavaScript</td>
</tr>
<tr>
<td>Normal Refresh</td>
<td>F5, Ctrl-R, ⌘R</td>
<td>Refreshes page</td>
</tr>
<tr>
<td>Hard Refresh</td>
<td>Ctrl-F5, Ctrl-Ø, ⌘ØR</td>
<td>Refreshes page including cached files</td>
</tr>
<tr>
<td>Harder Refresh</td>
<td>Ctrl-Ø+Hyper-ESC-R-F5</td>
<td>Remotely cycles power to datacenter</td>
</tr>
<tr>
<td>Hardest Refresh</td>
<td>Ctrl-Ø+⃝-R-F5-F-5-ESC-0-Ø-Ø-Scroll Lock</td>
<td>Internet starts over from ARPANET</td>
</tr>
</tbody>
</table>

- Cache coherence
- Snooping-based protocols
- Implementing synchronization

http://xkcd.com/1854/
Cache Coherence Avoids Stale Data

- Multicores have **multiple private caches** for performance
- Need to provide the illusion of a single shared memory
- Problem:

  ![Diagram of cache coherence](image)

  1. LD 0xA → 2
  2. ST 3 → 0xA
  3. LD 0xA → 2 (stale!)

- Solution: A **cache coherence protocol** controls cache contents to avoid stale lines
  - e.g., invalidate core 0's copy of A before letting core 2 write to it
Coherence vs. Consistency

• Cache coherence makes private caches invisible to software
  – Concerns reads/writes to a single memory location

• Memory consistency models precisely specify how memory behaves with respect to read and write operations from multiple processors
  – Concerns reads/writes to multiple memory locations
Implementing Cache Coherence

- Coherence protocols must enforce two rules:
  - **Write propagation**: Writes eventually become visible to all processors
  - **Write serialization**: Writes to the same location are serialized (all processors see them in the same order)
- How to ensure write propagation?
  - **Write-invalidate protocols**: Invalidate all other cached copies before performing the write
  - **Write-update protocols**: Update all other cached copies after performing the write
- How to ensure write serialization?
  - **Snooping-based protocols**: All caches observe each other’s actions through a shared bus
  - **Directory-based protocols**: A coherence directory tracks contents of private caches and serializes requests
Snooping-Based Coherence
[Goodman 1983]

Caches watch (snoop on) bus
to keep all processors’ view of memory coherent
Snooping-Based Coherence

- Bus provides serialization point
  - Broadcast, totally **ordered**
  - Each cache controller “snoops” all bus transactions
  - Controller updates state of cache in response to processor and snoop events and generates bus transactions

- Snoopy protocol (FSM)
  - State-transition diagram
  - Actions

![State, Tag, Data Table]

**Snoop** (observed bus transaction)
A Simple Protocol: Valid/Invalid (VI)

- Assume write-through caches

**Actions**

<table>
<thead>
<tr>
<th>Processor Read (PrRd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Write (PrWr)</td>
</tr>
<tr>
<td>Bus Read (BusRd)</td>
</tr>
<tr>
<td>Bus Write (BusWr)</td>
</tr>
</tbody>
</table>
Valid/Invalid Example

Main Memory

BusRd 0xA

Cache

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>V</td>
<td>2</td>
</tr>
</tbody>
</table>

Core 0

1 LD 0xA

Cache

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
</table>

Core 1
Valid/Invalid Example

1. LD 0xA
2. LD 0xA

Additional loads satisfied locally, without BusRd
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
Valid/Invalid Example

**VI Problems?**
Every write updates main memory
Every write requires broadcast & snoop

**Step 1:** LD 0xA
**Step 2:** LD 0xA
**Step 3:** ST 0xA
**Step 4:** LD 0xA

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Cache</th>
<th>Core 1</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="tag_0xA" alt="" /></td>
<td><img src="tag_0xA" alt="" /></td>
<td><img src="tag_0xA" alt="" /></td>
</tr>
<tr>
<td><img src="state_valid" alt="" /></td>
<td><img src="data_3" alt="" /></td>
<td><img src="state_valid" alt="" /></td>
<td><img src="data_3" alt="" /></td>
</tr>
<tr>
<td><img src="tag_0xA" alt="" /></td>
<td><img src="state_valid" alt="" /></td>
<td><img src="tag_0xA" alt="" /></td>
<td><img src="state_valid" alt="" /></td>
</tr>
<tr>
<td><img src="data_3" alt="" /></td>
<td></td>
<td><img src="data_3" alt="" /></td>
<td></td>
</tr>
</tbody>
</table>

Bus Rd 0xA
Modified/Shared/Invalid (MSI) Protocol

- Allows writeback caches + satisfies writes locally

### Actions

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Read (PrRd)</td>
</tr>
<tr>
<td>Processor Write (PrWr)</td>
</tr>
<tr>
<td>Bus Read (BusRd)</td>
</tr>
<tr>
<td>Bus Read Exclusive (BusRdX)</td>
</tr>
<tr>
<td>Bus Writeback (BusWB)</td>
</tr>
</tbody>
</table>
MSI Example

LD 0xA

Core 0

BusRd 0xA

Main Memory

Core 1

Cache

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>2</td>
</tr>
</tbody>
</table>

Cache

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MSI Example

Main Memory

BusRd 0xA

Cache

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>2</td>
</tr>
</tbody>
</table>

Core 0

Core 1

Additional loads satisfied locally, without BusRd (like in VI)
MSI Example

Additional loads and stores from core 0 satisfied locally, without bus transactions (unlike in VI)
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
Cache Interventions

- MSI lets caches serve writes without updating memory, so main memory can have stale data
  - Core 0’s cache needs to supply data
  - But main memory may also respond!
- Cache must override response from main memory
MSI Example

1. LD 0xA
2. LD 0xA
3. ST 0xA
4. ST 0xA
5. LD 0xA
MSI Optimizations: Exclusive State

• Observation: Doing read-modify-write sequences on private data is common
  – What’s the problem with MSI?

• Solution: E state (exclusive, clean)
  – If no other sharers, a read acquires line in E instead of S
  – Writes silently cause E→M (exclusive, dirty)
MESI: An Enhanced MSI protocol
increased performance for private read-write data

*Each* cache line has a tag

- **M**: Modified Exclusive
- **E**: Exclusive, unmodified
- **S**: Shared
- **I**: Invalid

**State bits**

**Address tag**

- **PrWr** / --
- **PrRd** / --
- **BusRd** / **BusWB**
- **PrWr** / **BusRdX**
- **BusRdX** / --

**Transitions**

- **PrRd** / **BusRd** if no other sharers
- **PrRd** / **BusRd** if other sharers

**Diagram**

[Diagram showing transitions between states M, E, S, and I with corresponding actions like PrWr, PrRd, BusRd, and BusRdX.]
Cache Coherence and False Sharing

**Performance Issue #1**

- A cache line contains more than one word, and cache coherence is done at line granularity

| state | line addr | word0 | word1 | ... | wordN |

- Suppose $P_1$ writes $\text{word}_i$ and $P_2$ writes $\text{word}_k$ and both words have the same line address
- What can happen?

  The line may be invalidated (ping-pong) many times unnecessarily because addresses are in the same line.
Hardware Support for Synchronization

- New instructions support atomic read-modify-write sequences without intervening memory operations by other processors.
- Example: Atomic swap

```
SWAP(Ra, literal, Rc)
PC ← PC + 4
EA ← Reg[Ra] + literal
TMP ← MEM[EA]
MEM[EA] ← Reg[Rc]
Reg[Rc] ← TMP
```

Cache coherence guarantees atomicity

- Can easily implement mutual exclusion (=binary semaphore)

```
CMOVE(0, R0)
loop: SWAP(R31, lock, R0)
BEQ(R0, loop)
    ... critical section ...
CMOVE(1, R0)
ST(R0, lock, R31)
```
Cache Coherence and Synchronization

**Performance Issue #2**

Cache coherence protocols will cause `lock` to *ping-pong* between P1’s and P2’s caches.

Ping-ponging can be reduced by first reading the `lock` location *(non-atomically)* and executing a swap only if it is found to be 1 *(test & test & set)*.
Summary

• Multiple cores for multi-threading performance
  – Each core has a private cache for performance
  – Issue: stale cache data for memory locations shared by multiple cores

• Cache coherence protocols ensure all cache agree on current value of a memory location
  – Write propagation to other processors
    • Write invalidate or write update protocols
  – Write serialization (writes seen in same order by all cores)
    • Snooping-based protocols (watch a shared transaction bus)
    • Directory-based protocols (track contents of caches, serializes requests)

• Snooping protocols
  – VI, MSI, MESI, … Additional cache line states for increased performance for common use cases.