Directory-Based Cache Coherence

Daniel Sanchez
Computer Science and Artificial Intelligence Lab
M.I.T.
Maintaining Cache Coherence

It is sufficient to have hardware such that

- only one processor at a time has write permission for a location
- no processor can load a stale copy of the location after a write
Maintaining Cache Coherence

It is sufficient to have hardware such that
• only one processor at a time has write permission for a location
• no processor can load a stale copy of the location after a write

⇒ A correct approach could be:
Maintaining Cache Coherence

It is sufficient to have hardware such that
• only one processor at a time has write permission for a location
• no processor can load a stale copy of the location after a write

⇒ A correct approach could be:

write request:
  The address is invalidated in all other caches before the write is performed
Maintaining Cache Coherence

It is sufficient to have hardware such that
• only one processor at a time has write permission for a location
• no processor can load a stale copy of the location after a write

⇒ A correct approach could be:

write request:
The address is *invalidated* in all other caches *before* the write is performed

read request:
If a dirty copy is found in some cache, a write-back is performed before the memory is read
Directory-Based Coherence
[Censier and Feautrier, 1978]

Snoopy Protocols

- Snoopy schemes broadcast requests over memory bus
- Difficult to scale to large numbers of processors
- Requires additional bandwidth to cache tags for snoop requests
Directory-Based Coherence
[Censier and Feautrier, 1978]

- Snoopy schemes broadcast requests over memory bus
- Difficult to scale to large numbers of processors
- Requires additional bandwidth to cache tags for snoop requests
Directory-Based Coherence
[Censier and Feautrier, 1978]

- Snoopy schemes broadcast requests over memory bus
- Difficult to scale to large numbers of processors
- Requires additional bandwidth to cache tags for snoop requests

- Directory schemes send messages to only those caches that might have the line
- Can scale to large numbers of processors
- Requires extra directory storage to track possible sharers
An MSI Directory Protocol

- Cache states: Modified (M) / Shared (S) / Invalid (I)
An MSI Directory Protocol

- Cache states: Modified (M) / Shared (S) / Invalid (I)
- Directory states:
  - Uncached (Un): No sharers
  - Shared (Sh): One or more sharers with read permission (S)
  - Exclusive (Ex): A single sharer with read & write permissions (M)
An MSI Directory Protocol

- Cache states: Modified (M) / Shared (S) / Invalid (I)
- Directory states:
  - Uncached (Un): No sharers
  - Shared (Sh): One or more sharers with read permission (S)
  - Exclusive (Ex): A single sharer with read & write permissions (M)
- Transient states not drawn for clarity; for now, assume no racing requests
Transitions initiated by processor accesses:

- **M**
- **S**
- **I**

### Actions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Read</td>
<td>Processor Read</td>
</tr>
<tr>
<td>Processor Write</td>
<td>Processor Write</td>
</tr>
<tr>
<td>Shared Request</td>
<td>Shared Request</td>
</tr>
<tr>
<td>Exclusive Request</td>
<td>Exclusive Request</td>
</tr>
</tbody>
</table>
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

- **M**
- **S**
- **I**

PrRd / ShReq

<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>Shared Request (ShReq)</td>
</tr>
<tr>
<td>Exclusive Request (ExReq)</td>
</tr>
</tbody>
</table>
Transitions initiated by processor accesses:

<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>Shared Request (ShReq)</td>
</tr>
<tr>
<td>Exclusive Request (ExReq)</td>
</tr>
</tbody>
</table>
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

- Processor Read (PrRd)
- Processor Write (PrWr)
- Shared Request (ShReq)
- Exclusive Request (ExReq)
Transitions initiated by processor accesses:

- Processor Read (PrRd)
- Processor Write (PrWr)
- Shared Request (ShReq)
- Exclusive Request (ExReq)
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

- PrRd / --
- PrWr / --
- PrWr / ExReq
- PrRd / --
- PrRd / ShReq

**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>Shared Request (ShReq)</td>
</tr>
<tr>
<td>Exclusive Request (ExReq)</td>
</tr>
</tbody>
</table>
Transitions initiated by directory requests:

- **M**
- **S**
- **I**

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidation Request (InvReq)</td>
</tr>
<tr>
<td>Downgrade Request (DownReq)</td>
</tr>
<tr>
<td>Invalidation Response (InvResp)</td>
</tr>
<tr>
<td>Downgrade Response (DownResp)</td>
</tr>
</tbody>
</table>
Transitions initiated by directory requests:

InvReq / InvResp (with data)

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidation Request (InvReq)</td>
</tr>
<tr>
<td>Downgrade Request (DownReq)</td>
</tr>
<tr>
<td>Invalidation Response (InvResp)</td>
</tr>
<tr>
<td>Downgrade Response (DownResp)</td>
</tr>
</tbody>
</table>
MSI Protocol: Caches (2/3)

Transitions initiated by directory requests:

- DownReq / DownResp (with data)
- InvReq / InvResp (with data)

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidation Request (InvReq)</td>
</tr>
<tr>
<td>Downgrade Request (DownReq)</td>
</tr>
<tr>
<td>Invalidation Response (InvResp)</td>
</tr>
<tr>
<td>Downgrade Response (DownResp)</td>
</tr>
</tbody>
</table>
Transitions initiated by directory requests:

- **M**
  - DownReq / DownResp (with data)
  - InvReq / InvResp (with data)

- **S**
  - InvReq / InvResp (without data)

- **I**

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidation Request</td>
</tr>
<tr>
<td>(InvReq)</td>
</tr>
<tr>
<td>Downgrade Request</td>
</tr>
<tr>
<td>(DownReq)</td>
</tr>
<tr>
<td>Invalidation Response</td>
</tr>
<tr>
<td>(InvResp)</td>
</tr>
<tr>
<td>Downgrade Response</td>
</tr>
<tr>
<td>(DownResp)</td>
</tr>
</tbody>
</table>
Transitions initiated by evictions:

- M
- S
- I

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writeback Request (WbReq)</td>
</tr>
</tbody>
</table>
MSI Protocol: Caches (3/3)

Transitions initiated by evictions:

Eviction / WbReq (with data)

Actions

Writeback Request (WbReq)
MSI Protocol: Caches (3/3)

Transitions initiated by evictions:

- From M to S: Eviction / WbReq (with data)
- From S to I: Eviction / WbReq (without data)

Actions:
- Writeback Request (WbReq)
MSI Protocol: Caches

- Transitions initiated by processor accesses
- Transitions initiated by directory requests
- Transitions initiated by evictions
Transitions initiated by data requests:

Ex
Sh
Un
Transitions initiated by data requests:

- **Ex**
  - **Sh**
    - **Un**
    - **ShReq / Sharers = Sharers + {P}; ShResp**
    - **ShReq / Sharers = {P}; ShResp**
Transitions initiated by data requests:

ExReq / Sharers = \{P\}; ExResp

Ex

ShReq / Sharers = Sharers + \{P\}; ShResp

Sh

Un

ShReq / Sharers = \{P\}; ShResp
Transitions initiated by data requests:

- **ExReq / Sharers = \{P\}; ExResp**
- **ExReq / Inv(Sharers – \{P\}); Sharers = \{P\}; ExResp**
- **ShReq / Sharers = Sharers + \{P\}; ShResp**
- **ShReq / Sharers = \{P\}; ShResp**
MSI Protocol: Directory (1/2)

Transitions initiated by data requests:

ExReq / Sharers = {P}; ExResp

Ex

ShReq / Down(Sharer); Sharers = Sharer + {P}; ShResp

Ex

ExReq / Inv(Sharers − {P}); Sharers = {P}; ExResp

Sh

ShReq / Sharers = Sharers + {P}; ShResp

Un

ShReq / Sharers = {P}; ShResp
Transitions initiated by writeback requests:

- Ex
- Sh
- Un
Transitions initiated by writeback requests:

\[
\text{WbReq} / \text{Sharers} = \{\}; \text{WbResp}
\]
Transitions initiated by writeback requests:

Ex

WbReq / Sharers = {}; WbResp

Sh

WbReq && |Sharers| > 1 /
Sharers = Sharers - {P}; WbResp

Un
Transitions initiated by writeback requests:

- **Ex**
  - WbReq / Sharers = {}; WbResp

- **Sh**
  - WbReq && |Sharers| > 1 /
    - Sharers = Sharers - {P}; WbResp
  - WbReq && |Sharers| == 1 /
    - Sharers = {}; WbResp

- **Un**
MSI Directory Protocol Example

![Diagram of MSI Directory Protocol Example]

- **Main Memory**
- **Directory**
  - Tag
  - State
  - Sharers
- **Cache 0**
  - Tag
  - State
  - Data
- **Cache 1**
  - Tag
  - State
  - Data
- **Cache 2**
  - Tag
  - State
  - Data
- **Core 0**
- **Core 1**
- **Core 2**
MSI Directory Protocol Example

1. LD 0xA
MSI Directory Protocol Example

Main Memory

Directory

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

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>

Core 0

Cache 1

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

Core 1

Cache 2

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

Core 2

1. LD 0xA
MSI Directory Protocol Example

1. **LD 0xA**

2. **ShReq 0xA**

![Diagram of MSI Directory Protocol Example]

- **Cache 0**
  - Tag: 0xA
  - State: I->S

- **Cache 1**
  - Tag: [empty]
  - State: [empty]
  - Data: [empty]

- **Cache 2**
  - Tag: [empty]
  - State: [empty]
  - Data: [empty]
MSI Directory Protocol Example

1. **LD 0xA**

2. **ShReq 0xA**

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0}</td>
</tr>
</tbody>
</table>

Core 0

- **Cache 0**

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>

Core 1

- **Cache 1**

Core 2

- **Cache 2**
MSI Directory Protocol Example

1. **LD 0xA**

2. **ShReq 0xA**

3. **Mem[0xA] = 3**

### Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0}</td>
</tr>
</tbody>
</table>

### Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>

### Cache 1

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

### Cache 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
</table>
MSI Directory Protocol Example

1. LD 0xA

2. ShReq 0xA

3. Mem[0xA] = 3

4. ShResp 0xA, data=3
MSI Directory Protocol Example

1. LD 0xA
2. ShReq 0xA
3. Mem[0xA] = 3
4. ShResp 0xA, data=3
MSI Directory Protocol Example

<table>
<thead>
<tr>
<th>Main Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
</tr>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>0xA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache 0</th>
<th>Cache 1</th>
<th>Cache 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>State</td>
<td>Data</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>
MSI Directory Protocol Example

Main Memory

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0}</td>
</tr>
</tbody>
</table>

Core 0

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1

Cache 1

Core 2

Cache 2

1 LD 0xA

Core 0

Core 1

Core 2

October 25, 2021
MSI Directory Protocol Example

Main Memory

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0}</td>
</tr>
</tbody>
</table>

Core 0

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>

Core 2

Cache 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>

1. LD 0xA
MSI Directory Protocol Example

Core 0

Main Memory

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0}</td>
</tr>
</tbody>
</table>

Core 1

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>

Cache 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>

1. LD 0xA

2. ShReq 0xA
MSI Directory Protocol Example

1. LD 0xA
2. ShReq 0xA

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0,2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Cache 0</th>
<th>Cache 1</th>
<th>Cache 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>State</td>
<td>Data</td>
<td>Tag</td>
</tr>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
<td>0xA</td>
</tr>
</tbody>
</table>

Main Memory

Directory
MSI Directory Protocol Example

1. LD 0xA
2. ShReq 0xA
3. Mem[0xA] = 3
MSI Directory Protocol Example

1. **LD 0xA**
   - Core 0 requests to load the memory location at 0xA.

2. **ShReq 0xA**
   - Core 0 sends a sharer request (ShReq) to the directory for memory location 0xA.
   - The directory updates its state to indicate that 0xA is sharable (Sh) with sharers 0 and 2.

3. **Mem[0xA] = 3**
   - The main memory updates the value at 0xA to 3.

4. **ShResp 0xA, data=3**
   - The directory responds with the sharer request and the data value.
   - Core 0 updates its cache with the value 3 and sets the state to S.

The diagram illustrates the interaction between the core, cache, directory, and main memory, showing the process of loading a value into the cache and the steps involved in managing shared memory access.
MSI Directory Protocol Example

1. LD 0xA
2. ShReq 0xA
3. Mem[0xA] = 3
4. ShResp 0xA, data=3
MSI Directory Protocol Example

Core 0

Core 1

Core 2

ST 0xA
MSI Directory Protocol Example

1. ST 0xA
2. ExReq 0xA
MSI Directory Protocol Example

- **Core 0**: Cache 0 (Tag 0xA, State S, Data 3)
  - Directory: Tag 0xA, State Sh, Sharers {0,2}

- **Core 1**: Cache 1 (Tag 0xA, State I->M, Data)

- **Core 2**: Cache 2 (Tag 0xA, State S, Data 3)

- **Main Memory**

- **Directory**

1. **ST 0xA**
2. **ExReq 0xA**
3. **InvReq 0xA**

October 25, 2021
MSI Directory Protocol Example

Core 0
- Cache 0
  - Tag: 0xA
  - State: I
  - Data: 3

Core 1
- Cache 1
  - Tag: 0xA
  - State: I->M

Core 2
- Cache 2
  - Tag: 0xA
  - State: I
  - Data: 3

Main Memory

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0,2}</td>
</tr>
</tbody>
</table>

1. ST 0xA
2. ExReq 0xA
3. InvReq 0xA
4. InvReq 0xA
MSI Directory Protocol Example

- **Core 0**: Cache 0
  - Tag 0xA
  - State I
  - Data 3

- **Core 1**: Cache 1
  - Tag 0xA
  - State I->M
  - Data

- **Core 2**: Cache 2
  - Tag 0xA
  - State I
  - Data 3

**Directory**

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0,2}</td>
</tr>
</tbody>
</table>

**Main Memory**

- **ExReq 0xA**
- **InvReq 0xA**
- **InvResp 0xA**

Steps:

1. **ST 0xA**
2. **ExReq 0xA**
3. **InvReq 0xA**
4. **InvResp 0xA**
3. **InvReq 0xA**
4. **InvResp 0xA**
MSI Directory Protocol Example

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

1. **ST 0xA**

2. **ExReq 0xA**

3. **InvReq 0xA**

4. **InvResp 0xA**

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I-&gt;M</td>
<td></td>
</tr>
</tbody>
</table>

Cache 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 0

Core 1

Core 2
 MSI Directory Protocol Example

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

1. ST 0xA
2. ExReq 0xA
3. InvReq 0xA
4. InvResp 0xA
5. Mem[0xA] = 3

Core 0
Cache 0
Tag | State | Data
---|-------|-----
0xA | I     | 3   

Core 1
Cache 1
Tag | State | Data
---|-------|-----
0xA | I->M  |     

Core 2
Cache 2
Tag | State | Data
---|-------|-----
0xA | I     | 3   

October 25, 2021
L13-13
MSI Directory Protocol Example

Core 0
- Tag: 0xA
- State: I
- Data: 3

Core 1
- Tag: 0xA
- State: I->M
- Data:

Core 2
- Tag: 0xA
- State: I
- Data: 3

Directory
- Tag: 0xA
- State: Ex
- Sharers: {1}

Main Memory
- Mem[0xA] = 3

1. ST 0xA
2. ExReq 0xA
3. InvReq 0xA
4. InvResp 0xA
5. Mem[0xA] = 3
6. ExResp 0xA data = 3
MSI Directory Protocol Example

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

1. **ST 0xA**
2. **ExReq 0xA**
3. **InvReq 0xA**
4. **InvResp 0xA**
5. **Mem[0xA] = 3**
6. **ExResp 0xA**

Data flow:
- Core 0: Cache 0
- Core 1: Cache 1
- Core 2: Cache 2

- **Core 0**: Cache 0
  - Tag: 0xA
  - State: I
  - Data: 3

- **Core 1**: Cache 1
  - Tag: 0xA
  - State: M
  - Data: 5

- **Core 2**: Cache 2
  - Tag: 0xA
  - State: I
  - Data: 3
MSI Directory Protocol Example

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

Core 0

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>M-&gt;I</td>
<td>5</td>
</tr>
</tbody>
</table>

Core 2

Cache 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

ST 0x(B)
MSI Directory Protocol Example

Core 0

Cache 0

Tag | State | Data
---|---|---
0xA | I | 3

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

Core 1

Cache 1

Tag | State | Data
---|---|---
0xA | M->I | 5

Core 2

Cache 2

Tag | State | Data
---|---|---
0xA | I | 3

2 WbReq 0xA, data=5

1 ST 0xB
MSI Directory Protocol Example

1. ST 0xB

2. WbReq 0xA, data=5

3. Mem[0xA] = 5
MSI Directory Protocol Example

1. ST 0xB

2. WbReq 0xA, data=5

3. Mem[0xA] = 5
MSI Directory Protocol Example

### Diagram

- **Main Memory**
  - Mem[0xA] = 5

- **Directory**
  - Tag | State | Sharers
  - 0xA | Un    | {}

- **Cache 0**
  - Tag | State | Data
  - 0xA | I     | 3

- **Cache 1**
  - Tag | State | Data
  - 0xA | M->I  | 5

- **Cache 2**
  - Tag | State | Data
  - 0xA | I     | 3

### Instructions

1. **ST 0xB**
2. **WbReq 0xA, data=5**
3. **Mem[0xA] = 5**
4. **WbResp 0xA**
MSI Directory Protocol Example

1. ST 0xBB
2. WbReq 0xA, data=5
3. Mem[0xA] = 5
4. WbResp 0xA
MSI Directory Protocol Example

1. ST 0xB
2. WbReq 0xA, data=5
3. Mem[0xA] = 5
4. WbResp 0xA
5. ExReq 0xB
MSI Directory Protocol Example

1. ST 0xB
2. WbReq 0xA, data=5
3. Mem[0xA] = 5
4. WbResp 0xA
5. ExReq 0xB

Main Memory

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>I-&gt;M</td>
<td></td>
</tr>
</tbody>
</table>

Cache 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 0

Core 1

Core 2
MSI Directory Protocol Example

1. ST 0xB
2. WbReq 0xA, data=5
3. Mem[0xA] = 5
4. WbResp 0xA
5. ExReq 0xB
6. Mem[0xB] = 10

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
<tr>
<td>0xB</td>
<td>I-&gt;M</td>
<td></td>
</tr>
</tbody>
</table>

Core 0

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
<tr>
<td>0xB</td>
<td>I-&gt;M</td>
<td></td>
</tr>
</tbody>
</table>

Core 2

Cache 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>
MSI Directory Protocol Example

1. ST 0xB
2. WbReq 0xA, data=5
3. Mem[0xA] = 5
4. WbResp 0xA
5. ExReq 0xB
6. Mem[0xB] = 10
7. ExResp 0xB, data=10
MSI Directory Protocol Example

1. ST 0xB
2. WbReq 0xA, data=5
3. Mem[0xA] = 5
4. WbResp 0xA
5. ExReq 0xB
6. Mem[0xB] = 10
7. ExResp 0xB, data=10
MSI Directory Protocol Example

Why are 0xA’s wb and 0xB’s req serialized?
MSI Directory Protocol Example

Core 0

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>M</td>
<td>10</td>
</tr>
</tbody>
</table>

Core 2

Cache 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Mem[0xA] = 5

WbReq 0xA, data=5

WbResp 0xA

Mem[0xB] = 10

ExReq 0xB

ExResp 0xB, data=10

Why are 0xA’s wb and 0xB’s req serialized?

Structural dependence
Why are 0xA’s wb and 0xB’s req serialized? Structural dependence
Possible solutions?
MSI Directory Protocol Example

Why are 0xA’s wb and 0xB’s req serialized?  
Structural dependence

Possible solutions?  
Buffer outside of cache to hold write data
Miss Status Holding Register

MSHR – Holds load misses and writes outside of cache

MSHR entry

<table>
<thead>
<tr>
<th>V</th>
<th>X</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
</table>

- On eviction/writeback
  - No free MSHR entry: stall
  - Allocate new MSHR entry
  - When channel available send WBReq and data
  - Deallocate entry on WBResp
**Miss Status Holding Register**

MSHR – Holds load misses and writes outside of cache

<table>
<thead>
<tr>
<th>MSHR entry</th>
<th></th>
<th>per ld/st slots</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>X</td>
<td>Addr</td>
</tr>
</tbody>
</table>

- **On cache load miss**
  - No free MSHR entry: stall
  - Allocate new MSHR entry
  - Send ShReq (or ExReq)
  - On *Resp forward data to CPU and cache
  - Deallocate MSHR
Miss Status Holding Register

MSHR – Holds load misses and writes outside of cache

<table>
<thead>
<tr>
<th>V</th>
<th>X</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>L/S</td>
<td>Inum</td>
<td>Block Offset</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

```
MSHR entry

V X Addr Data

L/S Inum Block Offset
```
## Miss Status Holding Register

**MSHR** – Holds load misses and writes outside of cache

**MSHR entry**

<table>
<thead>
<tr>
<th>V</th>
<th>X</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
</table>

**Per ld/st slots**

<table>
<thead>
<tr>
<th></th>
<th>V</th>
<th>L/S</th>
<th>Inum</th>
<th>Block Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>V</td>
<td>L/S</td>
<td>Inum</td>
<td>Block Offset</td>
</tr>
<tr>
<td>2</td>
<td>V</td>
<td>L/S</td>
<td>Inum</td>
<td>Block Offset</td>
</tr>
<tr>
<td>3</td>
<td>V</td>
<td>L/S</td>
<td>Inum</td>
<td>Block Offset</td>
</tr>
</tbody>
</table>

Per ld/st slots allow servicing multiple requests with one entry
Miss Status Holding Register

MSHR – Holds load misses and writes outside of cache

**MSHR entry**

<table>
<thead>
<tr>
<th>V</th>
<th>X</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>per ld/st slots</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
</tr>
<tr>
<td>V</td>
</tr>
<tr>
<td>V</td>
</tr>
</tbody>
</table>

- **On cache load miss**
  - Look for matching address in MSHRs
    - If not found
      - If no free MSHR entry: stall
      - Allocate new MSHR entry and fill in
    - If found, just fill in per ld/st slot
      - Send ShReq (or ExReq)
      - On *Resp forward data to CPU and cache
      - Deallocate MSHR

Per ld/st slots allow servicing multiple requests with one entry
Directory Organization

- **Requirement:** Directory needs to keep track of all the cores that are sharing a cache block.

- **Challenge:** For each block, the space needed to hold the list of sharers grows with number of possible sharers...
Flat, Memory-based Directories

- Dedicate a few bits of main memory to store the state and sharers of every line
- Encode sharers using a bit-vector

Main Memory

64 bytes 10 bits

State

Sharer Set

Sh 0 1 0 0 1 1 0 0
Flat, Memory-based Directories

- Dedicate a few bits of main memory to store the state and sharers of every line
- Encode sharers using a bit-vector

![Diagram of main memory and bit-vector]

- **Simple**
- **Slow**
- Very inefficient with many processors (~P bits/line)
Sparse Full-Map Directories

- Not every line in the system needs to be tracked – only those in private caches!
- Idea: Organize directory as a cache
Sparse Full-Map Directories

- Not every line in the system needs to be tracked – only those in private caches!
- Idea: Organize directory as a cache

### Directory Entry Format

<table>
<thead>
<tr>
<th>Way 1</th>
<th>Way 2</th>
<th>Way 3</th>
<th>Way 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Directory Entry Format**

- **Line Address**: 0xF00
- **State**: Sh
- **Sharer Set**: 0 1 0 0 1 1 0 0

- ✓ Low latency, energy-efficient
- ✗ Bit-vectors grow with # cores → Area scales poorly
- ✗ Limited associativity → Directory-induced invalidations
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0}</td>
<td>0xF</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

Main Memory

Directory

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF</td>
<td>M</td>
<td>1</td>
</tr>
</tbody>
</table>

Cache 2

Core 0

Core 1

Core 2
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

<table>
<thead>
<tr>
<th>Main Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache 0</td>
<td>Cache 1</td>
<td>Cache 2</td>
</tr>
<tr>
<td></td>
<td>Tag</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>0xA</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>Tag</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>0xF</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Tag</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>0xB</td>
<td>I-&gt;S</td>
</tr>
</tbody>
</table>
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0}</td>
<td>0xF</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

Core 0
- Cache 0:
  - Tag: 0xA
  - State: S
  - Data: 3

Core 1
- Cache 1:
  - Tag: 0xF
  - State: M
  - Data: 1

Core 2
- Cache 2:
  - Tag: 0xB
  - State: I->S

1. LD 0xB
2. ShReq 0xB
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

### Diagram

**Main Memory**

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0}</td>
<td>0xF</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

**Cache 0**

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>

**Cache 1**

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF</td>
<td>M</td>
<td>1</td>
</tr>
</tbody>
</table>

**Cache 2**

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>

1. LD 0xB
2. ShReq 0xB
3. InvReq 0xA
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

### Main Memory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0}</td>
</tr>
<tr>
<td>0xF</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

### Directory

- **InvReq 0xA**
- **InvResp 0xA**
- **ShReq 0xB**
- **LD 0xB**
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>Sh</td>
<td>{2}</td>
</tr>
<tr>
<td>0xF</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

1. LD 0xB
2. ShReq 0xB
3. InvReq 0xA
4. InvResp 0xA
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

1. LD 0xB
2. ShReq 0xB
3. InvReq 0xA
4. InvResp 0xA
5. Mem[0xB] = 5
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

```
<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>Sh</td>
<td>{2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>
```

```
Core 0
Cache 0
<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1
Cache 1
<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF</td>
<td>M</td>
<td>1</td>
</tr>
</tbody>
</table>

Core 2
Cache 2
<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>
```

1. LD 0xB
2. ShReq 0xB
3. InvReq 0xA
4. InvResp 0xA
5. Mem[0xB] = 5
6. ShResp 0xB, data=5
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>Sh</td>
<td>{2}</td>
</tr>
<tr>
<td>0xF</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

Main Memory

Directory

- InvReq 0xA
- InvResp 0xA
- ShReq 0xB
- ShResp 0xB, data=5
- Mem[0xB] = 5
- LD 0xB

Core 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF</td>
<td>M</td>
<td>1</td>
</tr>
</tbody>
</table>

Core 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>S</td>
<td>5</td>
</tr>
</tbody>
</table>
Directory-Induced Invalidations

- To retain inclusion, must invalidate all sharers of an entry before reusing it for another address
- Example: 2-way set-associative sparse directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>Sh</td>
<td>{2}</td>
</tr>
<tr>
<td>0xF</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

```
Main Memory

Directory

- Mem[0xB] = 5

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>Sh</td>
<td>{2}</td>
</tr>
<tr>
<td>0xF</td>
<td>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF</td>
<td>M</td>
<td>1</td>
</tr>
</tbody>
</table>

Cache 2

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB</td>
<td>S</td>
<td>5</td>
</tr>
</tbody>
</table>

Core 0

Core 1

Core 2

LD 0xB

How many entries should the directory have?

October 25, 2021

MIT 6.823 Fall 2021

L13-21
Inexact Representations of Sharer Sets

- Coarse-grain bit-vectors (e.g., 1 bit per 4 cores)

<table>
<thead>
<tr>
<th>Sharer Set</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-3</td>
<td>4-7</td>
<td>8-11</td>
<td>12-15</td>
<td>16-19</td>
<td>20-23</td>
<td></td>
</tr>
</tbody>
</table>

- Limited pointers: Maintain a few sharer pointers, on overflow mark ‘all’ and broadcast (or invalidate another sharer)

<table>
<thead>
<tr>
<th>Sharer Set</th>
<th>0</th>
<th>8</th>
<th>14</th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>all</td>
<td>sharer 1</td>
<td>sharer 2</td>
<td>sharer 3</td>
</tr>
</tbody>
</table>

- Allow false positives (e.g., Bloom filters)
Inexact Representations of Sharer Sets

• Coarse-grain bit-vectors (e.g., 1 bit per 4 cores)

<table>
<thead>
<tr>
<th>Sharer Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0-3 4-7 8-11 12-15 16-19 20-23</td>
</tr>
</tbody>
</table>

• Limited pointers: Maintain a few sharer pointers, on overflow mark ‘all’ and broadcast (or invalidate another sharer)

<table>
<thead>
<tr>
<th>Sharer Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 8 14 33</td>
</tr>
<tr>
<td>all sharer 1 sharer 2 sharer 3</td>
</tr>
</tbody>
</table>

• Allow false positives (e.g., Bloom filters)

✓ Reduced area & energy
✗ Overheads still not scalable (these techniques simply play with constant factors)
✗ Inexact sharers → Broadcasts, invalidations or spurious invalidations and downgrades
Protocol Races

- Directory serializes multiple requests for the same address
  - Same-address requests are queued or NACKed and retried
- But races still exist due to conflicting requests
- Example: Upgrade race

```
Main Memory

ReqQ

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Sh</td>
<td>{0,2}</td>
</tr>
</tbody>
</table>

Core 0 Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1 Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S</td>
<td>3</td>
</tr>
</tbody>
</table>
```
Protocol Races

- Directory serializes multiple requests for the same address
  - Same-address requests are queued or NACKed and retried
- But races still exist due to conflicting requests
- Example: Upgrade race

<table>
<thead>
<tr>
<th>ReqQ</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>State</td>
</tr>
<tr>
<td>0xA</td>
<td>Sh</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache 0</th>
<th>Cache 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>State</td>
</tr>
<tr>
<td>0xA</td>
<td>S</td>
</tr>
</tbody>
</table>

Core 0 1  ST 0xA

Core 1 1' ST 0xA
Protocol Races

- Directory serializes multiple requests for the same address
  - Same-address requests are queued or NACKed and retried
- But races still exist due to conflicting requests
- Example: Upgrade race

<table>
<thead>
<tr>
<th>Main Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReqQ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>0xA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>0xA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ST 0xA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>0xA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1′ ST 0xA</td>
</tr>
</tbody>
</table>

October 25, 2021
MIT 6.823 Fall 2021
Protocol Races

- Directory serializes multiple requests for the same address
  - Same-address requests are queued or NACKed and retried
- But races still exist due to conflicting requests
- Example: Upgrade race

Caches 0 and 1 issue simultaneous ExReqs

1. ST 0xA

2. ExReq 0xA

2'. ExReq 0xA

Core 0

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S-&gt;M</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 1

Cache 1

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>S-&gt;M</td>
<td>3</td>
</tr>
</tbody>
</table>
Protocol Races

- Directory serializes multiple requests for the same address
  - Same-address requests are queued or NACKed and retried
- But races still exist due to conflicting requests
- Example: Upgrade race

Caches 0 and 1 issue simultaneous ExReqs
Directory starts serving cache 0’s ExReq, queues cache 1’s
Protocol Races

- Directory serializes multiple requests for the same address
  - Same-address requests are queued or NACKed and retried
- But races still exist due to conflicting requests
- Example: Upgrade race

**Diagram:**

- Caches 0 and 1 issue simultaneous ExReqs
  - Directory starts serving cache 0’s ExReq, queues cache 1’s ExReq
- Cache 1 expected ExResp, but got InvReq!
Protocol Races

- Directory serializes multiple requests for the same address
  - Same-address requests are queued or NACKed and retried
- But races still exist due to conflicting requests
- Example: Upgrade race

Caches 0 and 1 issue simultaneous ExReqs
Directory starts serving cache 0’s ExReq, queues cache 1’s

Cache 1 expected ExResp, but got InvReq!
Cache 1 should transition from S->M to I->M and send InvResp
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex</td>
<td>{0}</td>
</tr>
</tbody>
</table>

Main Memory

Directory

Cache 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>M</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 0

Cache 1

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

Core 1

Cache 2

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

Core 2

1 ST 0xA
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

![Diagram of cache and directory states]

- **Core 0**
  - Cache 0:
    - Tag: 0xA
    - State: M
    - Data: 3
  - Core 0

- **Core 1**
  - Cache 1:
    - Tag: 0xA
    - State: I->M
    - Data
  - Core 1

- **Core 2**
  - Cache 2:
    - Tag: 0xA
    - State: Ex
    - Sharers: {0}
  - Core 2

1. ST 0xA
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

```
<table>
<thead>
<tr>
<th>Core 0</th>
<th>Cache 0</th>
<th>Cache 1</th>
<th>Cache 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>State</td>
<td>Tag</td>
<td>State</td>
</tr>
<tr>
<td>0xA</td>
<td>M</td>
<td>0xA</td>
<td>I-&gt;M</td>
</tr>
<tr>
<td>Data</td>
<td>3</td>
<td>Data</td>
<td>Data</td>
</tr>
</tbody>
</table>

ST 0xA

ExReq 0xA

Main Memory
Directory
<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex</td>
<td>{0}</td>
</tr>
</tbody>
</table>
```
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

```
Core 0

0x10: M 3

Main Memory

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex-&gt;Ex</td>
<td>{2}</td>
</tr>
</tbody>
</table>

Core 1

0x10: I->M

Cache 0

0x10: M 3

Cache 1

0x10: I->M

Cache 2

0x10: I->M

Core 2

ST 0xA

ExReq 0xA
```
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

**Diagram:**

1. **ST 0xA**
2. **ExReq 0xA**
3. **ExFwd 0xA, req=2**
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

1. ST 0xA
2. ExReq 0xA
3. ExFwd 0xA, req=2
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

```
Core 0

+------------------+
| Cache 0          |
+------------------+
| Tag  | State | Data |
| 0xA  | I     | 3    |
+------------------+

Core 1

+------------------+
| Cache 1          |
+------------------+
| Tag  | State | Data |
| 0xA  | I>->M |      |
+------------------+

Core 2

+------------------+
| Cache 2          |
+------------------+
| Tag  | State | Data |
| 0xA  | I>->M |      |
+------------------+

Main Memory

Directory

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex-&gt;Ex</td>
<td>{2}</td>
</tr>
</tbody>
</table>
```

3. ExFwd 0xA, req=2
2. ExReq 0xA
3. ExResp 0xA, data=3
1. ST 0xA
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

**Diagram:**

- **Core 0:**
  - Cache 0
    - Tag: 0xA, State: I, Data: 3

- **Core 1:**
  - Cache 1
    - Tag: 0xA, State: M, Data: 3

- **Core 2:**
  - Cache 2
    - Tag: 0xA, State: M, Data: 3

**Directory:***

<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex-&gt;Ex</td>
<td>{2}</td>
</tr>
</tbody>
</table>

**Steps:**

1. **ST 0xA**
2. **ExReq 0xA**
3. **ExFwd 0xA, req=2**
4. **ExResp 0xA, data=3**
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

```
Core 0
  Main Memory
  Directory
<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Sharers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Ex-&gt;Ex</td>
<td>{2}</td>
</tr>
</tbody>
</table>

Core 1
  Cache 0
<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>I</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 2
  Cache 1
<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>M</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 3
  Cache 2
<table>
<thead>
<tr>
<th>Tag</th>
<th>State</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>M</td>
<td>3</td>
</tr>
</tbody>
</table>
```

1. ST 0xA
2. ExReq 0xA
3. ExResp 0xA, data=3
4. ExAck 0xA
3. ExFwd 0xA, req=2
Extra Hops and 3-Hop Protocols
Reducing Protocol Latency

- Problem: Data in another cache needs to pass through the directory, adding latency
- Optimization: Forward data to requester directly

Diagram:

1. ST 0xA
2. ExReq 0xA
3. ExResp 0xA, req=2
4. ExAck 0xA
3. ExFwd 0xA, req=2

Core 0
- Tag: 0xA
- State: I
- Data: 3

Core 1
- Tag: 0xA
- State: M
- Data: 3

Core 2
- Tag: 0xA
- State: Ex
- Sharers: {2}
Coherence in Multi-Level Hierarchies

- Can use the same or different protocols to keep coherence across multiple levels
- Key invariant: Ensure sufficient permissions in all intermediate levels
- Example: 8-socket Xeon E7 (8 cores/socket)
In-Cache Directories

- Common multicore memory hierarchy:
  - 1+ levels of private caches
  - A shared last-level cache
  - Need to enforce coherence among private caches

- Idea: Embed the directory information in shared cache tags
  - Shared cache must be inclusive
In-Cache Directories

- Common multicore memory hierarchy:
  - 1+ levels of private caches
  - A shared last-level cache
  - Need to enforce coherence among private caches

- Idea: Embed the directory information in shared cache tags
  - Shared cache must be inclusive

✔ Avoids tag overheads & separate lookups
✗ Can be inefficient if shared cache size >> sum(private cache sizes)
Avoiding Protocol Deadlock

- Protocols can cause deadlocks even if network is deadlock-free! *(more on this later)*
Avoiding Protocol Deadlock

- Protocols can cause deadlocks even if network is deadlock-free! (*more on this later*)

Example: Both nodes saturate all intermediate buffers with requests to each other, blocking responses from entering the network.

- **Solution:** Separate *virtual networks*
  - Different sets of virtual channels and endpoint buffers
  - Same physical routers and links
Avoiding Protocol Deadlock

• Protocols can cause deadlocks even if network is deadlock-free! (*more on this later*)

  Example: Both nodes saturate all intermediate buffers with requests to each other, blocking responses from entering the network

• Solution: Separate *virtual networks*
  – Different sets of virtual channels and endpoint buffers
  – Same physical routers and links

• Most protocols require at least 2 virtual networks (for requests and replies), often >2 needed
Coherence and Synchronization

Processor 1
R ← 1
L: swap (mutex), R;
if <R> then goto L;
critical section
M[mutex] ← 0;
cache

Processor 2
R ← 1
L: swap (mutex), R;
if <R> then goto L;
critical section
M[mutex] ← 0;
cache

Processor 3
R ← 1
L: swap (mutex), R;
if <R> then goto L;
critical section
M[mutex] ← 0;
mutex=1

CPU-Memory Bus
Cache coherence protocols will cause *mutex* to *ping-pong* between P1’s and P2’s caches.
Cache coherence protocols will cause mutex to ping-pong between P1’s and P2’s caches.
Cache coherence protocols will cause mutex to ping-pong between P1’s and P2’s caches.

Ping-ponging can be reduced by first reading the mutex location (non-atomically) and executing a swap only if it is found to be zero (test&test&set).
Implementing Atomic Instructions

• In general, an *atomic read-modify-write* instruction requires two memory operations without intervening memory operations by other processors
Implementing Atomic Instructions

• In general, an atomic read-modify-write instruction requires two memory operations without intervening memory operations by other processors

• Implementation options:
  • With snoopy coherence, lock the bus → expensive
Implementing Atomic Instructions

• In general, an atomic read-modify-write instruction requires two memory operations without intervening memory operations by other processors.

• Implementation options:
  • With snoopy coherence, lock the bus → expensive
  • With directory-based coherence, lock the line in the cache (prevent invalidations or evictions until atomic op finishes) → complex
Implementing Atomic Instructions

• In general, an atomic read-modify-write instruction requires two memory operations without intervening memory operations by other processors.

• Implementation options:
  • With snoopy coherence, lock the bus → expensive
  • With directory-based coherence, lock the line in the cache (prevent invalidations or evictions until atomic op finishes) → complex

• Modern processors often use
  load-reserve
  store-conditional
Load-reserve & Store-conditional

Special register(s) to hold reservation flag and address, and the outcome of store-conditional

Load-reserve R, (a):
<flag, adr> ← <1, a>;
R ← M[a];

Store-conditional (a), R:
if <flag, adr> == <1, a>
then cancel other procs’ reservation on a;
M[a] ← <R>;
status ← succeed;
else status ← fail;

If the cache receives an invalidation to the address in the reserve register, the reserve bit is set to 0

- Several processors may reserve ‘a’ simultaneously
- These instructions are like ordinary loads and stores with respect to the bus traffic
Load-Reserve/Store-Conditional

Swap implemented with Ld-Reserve/St-Conditional

#     Swap(R1, mutex):

L:     Ld-Reserve R2, (mutex)
       St-Conditional (mutex), R1
       if (status == fail) goto L
       R1 <- R2
Performance: Load-reserve & Store-conditional

The total number of coherence transactions is not necessarily reduced, but splitting an atomic instruction into load-reserve & store-conditional:

- *increases utilization* (and reduces processor stall time), especially in split-transaction buses and directories

- *reduces cache ping-pong effect* because processors trying to acquire a semaphore do not have to perform stores each time
Thank you!

Next Lecture: Consistency and Relaxed Memory Models