Directory-Based Cache Coherence

Mengjia Yan
Computer Science & Artificial Intelligence Lab
M.I.T.

Based on slides from Daniel Sanchez
Valid/Invalid Example

Main Memory

<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

Core 1

1. LD 0xA
Valid/Invalid Example

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

1. LD 0xA

BusRd 0xA

2. LD 0xA
Valid/Invalid Example

1. LD 0xA

2. LD 0xA

Core 0

Core 1

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>

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>
Valid/Invalid Example

1. LD 0xA
2. LD 0xA
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: (e.g. MSI)
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: (e.g. MSI)

write request:
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: (e.g. MSI)

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: (e.g. MSI)

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

read request:
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: (e.g. MSI)

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
(Goodman 1983)

- 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 Protocols  
(Goodman 1983)

- 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 Protocols

$P$ $P$ $P$ $P$

$P$

Interconnect Network

$P$

Mem.

Dir.

April 9, 2020
Directory-Based Coherence
(Censier and Feautrier, 1978)

Snoopy Protocols
(Goodman 1983)

- 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 Protocols

- 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

April 9, 2020
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
MSI Protocol: Caches (1/3)

Transitions initiated by processor accesses:

- M
- S
- I

<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:

M → S
PrRd / ShReq

I

<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:

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

### 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>
**MSI Protocol: Caches (1/3)**

Transitions initiated by processor accesses:

- **PrWr / ExReq**
  - M → S
  - S → M
- **PrRd / --**
  - S → I
  - I → S
- **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 processor accesses:

- 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 processor accesses:

- Processor Read (PrRd)
- Processor Write (PrWr)
- Shared Request (ShReq)
- Exclusive Request (ExReq)
MSI Protocol: Caches (2/3)

Transitions initiated by directory requests:

- Invalidation Request (InvReq)
- Downgrade Request (DownReq)
- Invalidation Response (InvResp)
- Downgrade Response (DownResp)
Transitions initiated by directory requests:

- Invalidation Request (InvReq)
- Downgrade Request (DownReq)
- Invalidation Response (InvResp)
- Downgrade Response (DownResp)
MSI Protocol: Caches (2/3)

Transitions initiated by directory requests:

DownReq / DownResp (with data)

InvReq / InvResp (with data)

Actions

<table>
<thead>
<tr>
<th>Invalidation Request (InvReq)</th>
</tr>
</thead>
<tbody>
<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)
- InvReq / InvResp (without data)

Actions

<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 (3/3)

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>
Transitions initiated by evictions:

- Eviction / WbReq (with data)

Actions

- Writeback Request (WbReq)
Transitions initiated by evictions:

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

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

- Transitions initiated by processor accesses
- Transitions initiated by directory requests
- Transitions initiated by evictions
MSI Protocol: Directory (1/2)

Transitions initiated by data requests:

- Ex
- Sh
- Un
MSI Protocol: Directory (1/2)

Transitions initiated by data requests:

- **Sh**
  - **ShReq / Sharers** = **Sharers** + {P}; **ShResp**

- **Un**
  - **ShReq / Sharers** = {P}; **ShResp**
MSI Protocol: Directory (1/2)

Transitions initiated by data requests:

ExReq / Sharers = \{P\}; ExResp

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

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

ExReq / Sharers = \{P\}; ExResp

Ex

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

Sh

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

Un

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

Sh

Un

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

- Ex
- Sh
- Un
MSI Protocol: Directory (2/2)

Transitions initiated by writeback requests:

Ex

WbReq / Sharers = {}; WbResp

Sh

Un
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 && \(|\text{Sharers}| > 1\) / Sharers = Sharers - {P}; WbResp

- **Un**
  
  WbReq && \(|\text{Sharers}| == 1\) / Sharers = {}; WbResp
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>
</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
MSI Directory Protocol Example

```
1. LD 0xA
```
MSI Directory Protocol Example

1. LD 0xA
MSI Directory Protocol Example

1. LD 0xA

2. ShReq 0xA
MSI Directory Protocol Example

1. LD 0xA
2. ShReq 0xA
MSI Directory Protocol Example

1. LD 0xA
2. ShReq 0xA
3. Mem[0xA] = 3
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>S</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Cache 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td></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>

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 0

LD 0xA

Cache 1

Core 1

Cache 2

Core 2
MSI Directory Protocol Example

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

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

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

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

1. LD 0xA
MSI Directory Protocol Example

The diagram illustrates the MSI (Multi-Stage Interconnect) directory protocol example. The directory contains an entry for tag 0xA with state "Sh" and sharers {0}. The cache memories for different cores are shown with their respective states and data. The protocol steps are:

1. **LD 0xA**: Core 0 requests a load operation on the memory location 0xA.
2. **ShReq 0xA**: Core 0 generates a share request for the memory location 0xA.
MSI Directory Protocol Example

```
Core 0
  Cache 0
    Tag | State | Data |
    0xA | S     | 3    |

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

Core 2
  Cache 2
    Tag | State | Data |
    0xA | I->S  |
```

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. LD 0x10

2. ShReq 0x10
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>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>

Core 1

Cache 2

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

Core 2

LD 0xA

ShReq 0xA

Mem[0xA] = 3
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>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

<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

<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>

Core 2

<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>

1 ST 0xA
MSI Directory Protocol Example

1. **ST 0xA**
   - Core 0
   - Cache 0
   - Tag 0xA, State S, Data 3

2. **ExReq 0xA**
   - Core 0
   - Cache 0
   - Tag 0xA, State Sh, Sharers {0,2}
   - Core 1
   - Cache 1
   - Tag 0xA, State I->M
   - Core 2
   - Cache 2
   - Tag 0xA, State S, Data 3
MSI Directory Protocol Example

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

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;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>S</td>
<td>3</td>
</tr>
</tbody>
</table>

ExReq 0xA

InvReq 0xA

ST 0xA

InvReq 0xA
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>{0,2}</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>

Core 0

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>

Core 1

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 2

1. ST 0xA
2. ExReq 0xA
3. InvReq 0xA
3. InvReq 0xA
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>{0,2}</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>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

ST 0xA

Core 1

InvReq 0xA

InvResp 0xA

Core 2

InvReq 0xA

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

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

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

Core 2
- Cache 2
  - Tag: 0xA
  - State: I
  - Data: 3
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>Ex</td>
<td>{1}</td>
</tr>
</tbody>
</table>

Mem[0xA] = 3

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

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>
MSI Directory Protocol Example

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

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

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

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

Cache 1
- Tag: 0xA
- State: I
- Data: 3

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

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

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

Main Memory

April 9, 2020
MIT 6.823 Spring 2020
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>0xA</td>
<td>M</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>

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>

Main Memory

Mem[0xA] = 3

ExReq 0xA

ExResp 0xA

data = 3

InvReq 0xA

InvResp 0xA

InvResp 0xA

InvResp 0xA

ST 0xA

April 9, 2020
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>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>

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>

1. ST 0xB
MSI Directory Protocol Example

1. ST 0xB

2. WbReq 0xA, data=5
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

1. ST 0xB
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**
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
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

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
7. ExResp 0xB, data=10

```markdown
<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:
- Tag: 0xA, State: I, Data: 3

Cache 1:
- Tag: 0xB, State: I->M

Cache 2:
- Tag: 0xA, State: I, Data: 3
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?
Why are 0xA’s wb and 0xB’s req serialized?

**Structural dependence**
MSI Directory Protocol Example

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

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

Structural dependence
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 Handling 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>
</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 Handling Register

- MSHR – Holds load misses and writes outside of cache

### MSHR entry

- V
- X
- Addr
- Data

### per ld/st slots

- L/S
- Inum
- Block Offset

### 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 Handling 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>
</table>

<table>
<thead>
<tr>
<th>L/S</th>
<th>Inum</th>
<th>Block Offset</th>
</tr>
</thead>
</table>
Miss Status Handling Register

MSHR – Holds load misses and writes outside of cache

Per ld/st slots allow servicing multiple requests with one entry
Miss Status Handling 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>V</th>
<th>L/S</th>
<th>Inum</th>
<th>Block Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>L/S</td>
<td>Inum</td>
<td>Block Offset</td>
</tr>
<tr>
<td>V</td>
<td>L/S</td>
<td>Inum</td>
<td>Block Offset</td>
</tr>
</tbody>
</table>

- On cache load miss
  - Look for matching address is MSHR
    - 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

<table>
<thead>
<tr>
<th>State</th>
<th>Sharer Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sh</td>
<td>01001100</td>
</tr>
</tbody>
</table>

64 bytes 10 bits
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 with State and Sharer Set]

- 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

Directory Entry Format

- Line Address
- State
- Sharer Set

Way 1  Way 2  Way 3  Way 4

```
0xF00  Sh  0  1  0  0  1  1  0  0
```
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

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

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

<table>
<thead>
<tr>
<th>Cache 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td></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>

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 1

Core 2

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

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

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>

Core 0

Core 1

1 LD 0xB

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

### 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>
</tbody>
</table>
```

### 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>
<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>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**
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
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>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>
```

```
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>I-&gt;S</td>
<td></td>
</tr>
</tbody>
</table>
```

**Steps:**
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

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

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

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

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

Main Memory
Dir
- Tag: 0xB
- State: Sh
- Sharers: {2}

Core 0
Mem[0xB] = 5
```

Legend:
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

Main Memory

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
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>
```

How many entries should the directory have?
Inexact Representations of Sharer Sets

- Coarse-grain bit-vectors (e.g., 1 bit per 4 cores)
  
  Sharer Set
  
<table>
<thead>
<tr>
<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>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)
  
  Sharer Set
  
<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>14</th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<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)

```
+------++------++------++------++------++------++------+
|       |       |       |       |       |       |       |       |
| 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
+------++------++------++------++------++------++------+
          0-3    4-7    8-11   12-15   16-19   20-23
```

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

```
+-----++-----++-----++-----++-----++-----++-----++-----+
|     |     |     |     |     |     |     |     |     |
| 0   | 8   | 14  | 33  |     |     |     |     |     |
+-----++-----++-----++-----++-----++-----++-----++-----+
       all  sharer 1  sharer 2  sharer 3
```

- 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

```
<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

<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

<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

![Diagram of main memory and cache states]

- Core 0
  - ST 0xA
- Core 1
  - ST 0xA

Table:

<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:

<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>
<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>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>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>

<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
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

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!
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

<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>M</td>
<td>3</td>
</tr>
</tbody>
</table>

Core 0  | Core 1  | Core 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

<table>
<thead>
<tr>
<th>Main Memory</th>
</tr>
</thead>
</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>Cache 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Core 0

Core 1

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>Main Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
</tr>
<tr>
<td>Tag</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>0xA</td>
<td>M</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core 0</td>
<td>Core 1</td>
<td>Core 2</td>
</tr>
<tr>
<td>ST 0xA</td>
<td></td>
<td></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

### Diagram

**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>

**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>

**Cache 1**

<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>

**Cache 2**

<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>

**Core 0**

**Core 1**

**Core 2**

1. **ST 0xA**
2. **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

---

**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>

---

**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>

**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>
<tbody>
<tr>
<td>0xA</td>
<td>I-&gt;M</td>
<td></td>
</tr>
</tbody>
</table>

---

**Core 0**

**Core 1**

**Core 2**

1. **ST 0xA**
2. **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

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

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

3. **ExFwd 0xA, req=2**

2. **ExReq 0xA**

1. **ST 0xA**

<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>I</td>
<td>3</td>
<td>0xA</td>
</tr>
</tbody>
</table>

April 9, 2020
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

- **Main Memory**
- **Directory**
  - | Tag | State  | Sharers |
  - |-----|--------|---------|
  - | 0xA | Ex->Ex | {2}     |

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

- **Cache 1**
  - | Tag | State | Data |
  - |-----|-------|------|
  - | 0xA |        |      |

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

- **Core 0**
- **Core 1**
- **Core 2**

- **Step 1**: ST 0xA
- **Step 2**: ExReq 0xA
- **Step 3**: ExFwd 0xA, req=2
- **Step 3**: 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

```
<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>
```

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

```
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 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>
</table>

Core 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. ExFwd 0xA, req=2
4. ExAck 0xA
3. 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

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>{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>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</td>
<td>3</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>M</td>
<td>3</td>
</tr>
</tbody>
</table>

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

April 9, 2020
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
  – Need extended directory if non-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
  - Need extended directory if non-inclusive

✓ Avoids tag overheads & separate lookups
✗ Can be inefficient if shared cache size $>> \text{sum(private cache sizes)}$
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)

![Diagram of multi-level memory hierarchy]

- MESIF protocol
  - Snooping (QPI)
- Mesi protocol
  - L3 in-cache directory
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
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
Thank you!

Next Lecture: 
On-chip Networks
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