Lect. 6: Directory Coherence Protocol

- **Snooping coherence**
  - Global state of a memory line is the collection of its state in all caches, and there is no summary state anywhere
  - All cache controllers monitor all other caches’ activities and maintain the state of their lines
  - Requires a broadcast shared medium (e.g., bus or ring) that also maintains a total order of all transactions
  - Bus acts as a serialization point to provide ordering

- **Directory coherence**
  - Global state of a memory line is the collection of its state in all caches, but there is a summary state at the directory
  - Cache controllers do not observe all activity, but interact only with directory
  - Can be implemented on scalable networks, where there is no total order and no simple broadcast, but only one-to-one communication
  - Directory acts as a serialization point to provide ordering
Directory Structure

- Directory information (for every memory line)
  - Line state bits (e.g., not cached, shared, modified)
  - Sharing bit-vector: one bit for each processor that is sharing or for the single processor that has the modified line
  - Organized as a table indexed by the memory line address

- Directory controller
  - Hardware logic that interacts with cache controllers and enforces cache coherence

### Directory Information

<table>
<thead>
<tr>
<th>Line state</th>
<th>Sharing vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0 1</td>
</tr>
</tbody>
</table>

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified

Memory: 4, 9
Directory Structure

- Directory information (for every memory line)
  - Line state bits (e.g., not cached, shared, modified)
  - Sharing bit-vector: one bit for each processor that is sharing or for the single processor that has the modified line
  - Organized as a table indexed by the memory line address

- Directory controller
  - Hardware logic that interacts with cache controllers and enforces cache coherence

Up to 3 processors can be supported
Directory Structure

- **Directory information (for every memory line)**
  - Line state bits (e.g., not cached, shared, modified)
  - Sharing bit-vector: one bit for each processor that is sharing or for the single processor that has the modified line
  - Organized as a table indexed by the memory line address

- **Directory controller**
  - Hardware logic that interacts with cache controllers and enforces cache coherence

Directory information

- **Line state**
  - 00 = not cached
  - 01 = shared
  - 10 = modified

- **Sharing vector**
  - 00 = not cached
  - 01 = shared
  - 10 = modified

- **Memory**
  - 01 = value
  - 10 = modified

Up to 3 processors can be supported

Line is not cached so sharing vector is empty and memory value is valid

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Structure

- **Directory information (for every memory line)**
  - Line state bits (e.g., not cached, shared, modified)
  - Sharing bit-vector: one bit for each processor that is sharing or for the single processor that has the modified line
  - Organized as a table indexed by the memory line address

- **Directory controller**
  - Hardware logic that interacts with cache controllers and enforces cache coherence

Up to 3 processors can be supported

Line is not cached so sharing vector is empty and memory value is valid

Line is shared in P0 and P2 and memory value is valid
Directory Operation

- Example: load with no sharers

- Cache states:
  - 00 = invalid
  - 01 = shared
  - 10 = modified

- Dir. states:
  - 00 = not cached
  - 01 = shared
  - 10 = modified
Directory Operation

- Example: load with no sharers

Load

<table>
<thead>
<tr>
<th>P0</th>
<th>L1</th>
<th>Line state</th>
<th>Sharing vector</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 0</td>
<td>0 0</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>L1</th>
<th>Line state</th>
<th>Sharing vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P2</th>
<th>L1</th>
<th>Line state</th>
<th>Sharing vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Operation

- Example: load with no sharers

Load: P0, L1
Line state: 0 0
Miss: P1, L1
Line state: 0 0
Miss: P2, L1
Line state: 0 0

Line state
Sharing vector
Memory

0 0
0 0 0
4

Cache states:
00 = invalid
01 = shared
10 = modified

Dir. states:
00 = not cached
01 = shared
10 = modified
Directory Operation

- Example: load with no sharers

Load

Line state

P0
L1

0 0

Miss

Line state

P1
L1

0 0

Line state

P2
L1

0 0

Line state

0 1

Sharing vector

0 0 1

Memory

4

Cache states:
00 = invalid
01 = shared
10 = modified

Dir. states:
00 = not cached
01 = shared
10 = modified
Directory Operation

- Example: load with no sharers

Load

Line state

P0

L1

0 0

4

Line state

P1

L1

0 0

Miss

Line state

P2

L1

0 0

Line state

Sharing vector

Memory

0 1

0 0 1

4

Cache states:
00 = invalid
01 = shared
10 = modified

Dir. states:
00 = not cached
01 = shared
10 = modified
Directory Operation

- Example: load with no sharers

Load

Line state

P0

L1

Line state

0 1

4

Miss

Line state

P1

L1

0 0

Line state

P2

L1

0 0

Line state

Memory

Sharing vector

0 0 1

Value

4

Cache states: 00 = invalid
01 = shared
10 = modified

Dir. states: 00 = not cached
01 = shared
10 = modified
Directory Operation

- Example: load with sharers

Line state

<table>
<thead>
<tr>
<th>P0</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Line state

<table>
<thead>
<tr>
<th>P1</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Line state

<table>
<thead>
<tr>
<th>P2</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Line state

<table>
<thead>
<tr>
<th>Sharing vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1</td>
</tr>
</tbody>
</table>

Memory

| 4 |

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Operation

- Example: load with sharers

<table>
<thead>
<tr>
<th>Line state</th>
<th>P0</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line state</th>
<th>P1</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line state</th>
<th>P2</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Line state: 00 = invalid, 01 = shared, 10 = modified

Sharing vector: 00 0 1

Memory: 4

Cache states: 00 = invalid, 01 = shared, 10 = modified

Dir. states: 00 = not cached, 01 = shared, 10 = modified

Load: 4
Directory Operation

- Example: load with sharers

- Line state: 00 = invalid, 01 = shared, 10 = modified
- Dir. states: 00 = not cached, 01 = shared, 10 = modified

Diagram:
- P0 (L1) with line state 01, sharing vector 001, Memory 4
- P1 (L1) with line state 00, Miss
- P2 (L1) with line state 00, Miss

Load operation results in a miss for P1 and P2, as indicated by the line state transitions and the sharing vector.
Directory Operation

- Example: load with sharers

Load
Line state
P0
L1
0 1
4

Sharing vector
0 1 1
Memory
4

Miss
Line state
P1
L1
0 0

P2
L1
0 0

Line state
0 0

Cache states:
00 = invalid
01 = shared
10 = modified

Dir. states:
00 = not cached
01 = shared
10 = modified
Directory Operation

- Example: load with sharers

<table>
<thead>
<tr>
<th>P0</th>
<th>L1</th>
<th>Line state</th>
<th>Sharing vector</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0 1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>L1</th>
<th>Line state</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P2</th>
<th>L1</th>
<th>Line state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- Cache states:
  - 00 = invalid
  - 01 = shared
  - 10 = modified

- Dir. states:
  - 00 = not cached
  - 01 = shared
  - 10 = modified
Directory Operation

- **Example: load with sharers**

  - **Line state**
    - P0: 01
      - L1: 4
  - **Sharing vector**
    - 011
  - **Memory**
    - 4

  - **Load**
    - P1: 01
      - L1: 4
      - Value: 1

  - **Miss**
    - P2: 00
      - L1

  - **Cache states:**
    - 00 = invalid
    - 01 = shared
    - 10 = modified
  - **Dir. states:**
    - 00 = not cached
    - 01 = shared
    - 10 = modified
Directory Operation

- Example: store with sharers

<table>
<thead>
<tr>
<th>P0</th>
<th>L1</th>
<th>Line state</th>
<th>0</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>L1</th>
<th>Line state</th>
<th>0</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>P2</th>
<th>L1</th>
<th>Line state</th>
<th>0</th>
<th>0</th>
<th></th>
</tr>
</thead>
</table>

Line state:  
00 = invalid  
01 = shared  
10 = modified

Sharing vector:  
0 1 1

Memory:  
4

Cache states:
00 = invalid
01 = shared
10 = modified

Dir. states:
00 = not cached
01 = shared
10 = modified
Directory Operation

- Example: store with sharers

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line state</td>
<td>Line state</td>
<td>Line state</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line state</th>
<th>Sharing vector</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
<td>0 1 1</td>
<td>4</td>
</tr>
</tbody>
</table>

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Operation

- Example: store with sharers

Line state | Sharing vector | Memory
-----|----------------|-----
0 1 | 0 1 1 | 4

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Operation

- Example: store with sharers

```
<table>
<thead>
<tr>
<th>Line state</th>
<th>Sharing vector</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
```

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Operation

- Example: store with sharers

<table>
<thead>
<tr>
<th>Cache states:</th>
<th>Dir. states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 = invalid</td>
<td>00 = not cached</td>
</tr>
<tr>
<td>01 = shared</td>
<td>01 = shared</td>
</tr>
<tr>
<td>10 = modified</td>
<td>10 = modified</td>
</tr>
</tbody>
</table>

Line state:
- P0: 01
- P1: 01
- P2: 00

Sharing vector:
- 01

Memory:
- 4

Invalidate

Miss

Reply
Directory Operation

- Example: store with sharers

<table>
<thead>
<tr>
<th>P0</th>
<th>L1</th>
<th>Line state</th>
<th>Sharing vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 4</td>
<td>0 1 1</td>
</tr>
<tr>
<td>P1</td>
<td>L1</td>
<td>0 1 4</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>L1</td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

Memory: 4

Invalidate

Reply

Store

Miss

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Operation

- Example: store with sharers

**Diagram:***
- **P0**: Line state 00, Sharing vector 01, Memory 4
- **P1**: Line state 01, Sharing vector 01, Memory 4
- **P2**: Line state 00, Sharing vector 01, Memory

**Legend:**
- **Miss**
- **Acknowledge**
- **Invalidate**
- **Reply**

**Cache states:**
- 00 = invalid
- 01 = shared
- 10 = modified

**Dir. states:**
- 00 = not cached
- 01 = shared
- 10 = modified

CS4/MSc Parallel Architectures - 2016-2017
Directory Operation

- Example: store with sharers

Store

Line state

Miss

P0

Line state

L1

L1

P1

P2

Line state

Direction states:

00 = not cached
01 = shared
10 = modified

00 = invalid
01 = shared
10 = modified

Sharing vector

0 1 1

Memory

4

Cache states:

Miss

Acknowledge

Invalidate

Reply

Director
Directory Operation

Example: store with sharers

- **Line state**
  - P0: 0 0
  - P1: 1 0
  - P2: 0 0

- **Sharing vector**
  - 0 1 1

- **Memory**
  - 4

- **Cache states:**
  - 00 = invalid
  - 01 = shared
  - 10 = modified

- **Dir. states:**
  - 00 = not cached
  - 01 = shared
  - 10 = modified

- **Operations:**
  - Store
  - Miss
  - Acknowledge
  - Invalidate
  - Reply
Directory Operation

- Example: store with sharers

**Line state**
- P0: L1 (0 0)
- P1: L1 (1 0)
- P2: L1 (0 0)

**Sharing vector**
- Memory: 0 1 0

**Cache states:**
- 00 = invalid
- 01 = shared
- 10 = modified

**Dir. states:**
- 00 = not cached
- 01 = shared
- 10 = modified

**Operations:**
- Miss
- Acknowledge
- Invalidate
- Reply

CS4/MSc Parallel Architectures - 2016-2017
Directory Operation

- Example: load with owner

### Cache States

- **00** = invalid
- **01** = shared
- **10** = modified

### Directory States

- **00** = not cached
- **01** = shared
- **10** = modified

**Line State**

- **P0**: Line state 00, Memory 4
- **P1**: Line state 10, Memory 6
- **P2**: Line state 00, Memory 0

**Sharing Vector**

- 0 1 0

**Memory**

- 4
Directory Operation

- Example: load with owner

<table>
<thead>
<tr>
<th>Line state</th>
<th>P0</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line state</th>
<th>P1</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Load

<table>
<thead>
<tr>
<th>Line state</th>
<th>P2</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line state</th>
<th>Sharing vector</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0</td>
<td>0 1 0</td>
<td>4</td>
</tr>
</tbody>
</table>

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Operation

- Example: load with owner

Line state

\[
\begin{array}{c|c}
0 & 0 \\
4 & \\
\end{array}
\]

P0

L1

Line state

\[
\begin{array}{c|c}
1 & 0 \\
6 & \\
\end{array}
\]

P1

L1

Load

Line state

\[
\begin{array}{c|c}
0 & 0 \\
\end{array}
\]

P2

L1

Miss

Line state

\[
\begin{array}{c|c}
0 & 0 \\
\end{array}
\]

P0

L1

Sharing vector

\[
\begin{array}{c|c|c}
0 & 1 & 0 \\
\end{array}
\]

Memory

\[
\begin{array}{c}
4 \\
\end{array}
\]

Cache states:
- 00 = invalid
- 01 = shared
- 10 = modified

Dir. states:
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Operation

Example: load with owner

| P0 | L1 | Line state | 0 | 0 | 4 |
| P1 | L1 | Line state | 1 | 0 | 6 |
| P2 | L1 | Line state | 0 | 0 |

Line state: 00 = invalid, 01 = shared, 10 = modified

Sharing vector: 01 = shared, 10 = modified, 11 = owned

Forward

Miss

Load

Cache states: 00 = invalid, 01 = shared, 10 = modified
Dir. states: 00 = not cached, 01 = shared, 10 = modified
Directory Operation

- Example: load with owner

<table>
<thead>
<tr>
<th>P0</th>
<th>L1</th>
<th>Line state</th>
<th>Sharing vector</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 0</td>
<td>1 0</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>L1</th>
<th>Line state</th>
<th>Sharing vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 1</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P2</th>
<th>L1</th>
<th>Line state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 0</td>
</tr>
</tbody>
</table>

**Cache states:**
- 00 = invalid
- 01 = shared
- 10 = modified

**Dir. states:**
- 00 = not cached
- 01 = shared
- 10 = modified
Directory Operation

- **Example: load with owner**

```
<table>
<thead>
<tr>
<th>Line state</th>
<th>Sharing vector</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 1 0</td>
<td>4</td>
</tr>
<tr>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- **Cache states:**
  - 00 = invalid
  - 01 = shared
  - 10 = modified

- **Dir. states:**
  - 00 = not cached
  - 01 = shared
  - 10 = modified
Directory Operation

- **Example: load with owner**

  - **Line state**
    - P0: 0 0
    - P1: 0 1
    - P2: 0 1
  - **Sharing vector**
    - 0 1 0
  - **Memory**
    - 4
  - **Cache states:**
    - 00 = invalid
    - 01 = shared
    - 10 = modified
  - **Dir. states:**
    - 00 = not cached
    - 01 = shared
    - 10 = modified

- **Load**
  - Value
  - Forward
  - Miss

- **Value**
  - 0 1
  - 6

CS4/MSc Parallel Architectures - 2016-2017
Directory Operation

- Example: load with owner

- Cache states:
  - 00 = invalid
  - 01 = shared
  - 10 = modified

- Dir. states:
  - 00 = not cached
  - 01 = shared
  - 10 = modified

- Line state
  - P0: L1 0 0
  - P1: L1 0 1
  - P2: L1 0 1

- Sharing vector
  - 0 1 0

- Memory
  - 4

- Forward

- Miss

- Load

- Value

- Acknowledge+Value
Directory Operation

- Example: load with owner

<table>
<thead>
<tr>
<th>Cache states:</th>
<th>Dir. states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 = invalid</td>
<td>00 = not cached</td>
</tr>
<tr>
<td>01 = shared</td>
<td>01 = shared</td>
</tr>
<tr>
<td>10 = modified</td>
<td>10 = modified</td>
</tr>
</tbody>
</table>

Line state: 00 = not cached, 01 = shared, 10 = modified

Sharing vector: 00 = not shared, 01 = shared, 10 = modified

Memory: 0 = not cached, 1 = shared, 2 = modified
Notes on Directory Operation

- On a write with multiple sharers it is necessary to collect and count all the invalidation acknowledgements (ACK) before actually writing.
- On transactions that involve more complex state changes the directory must also receive acknowledgement:
  - To establish the completion of the load or store.
- As with snooping on buses, “the devil is in the details” and we actually need transient states, must deal with conflicting requests, and must handle multi-level caches.
- As with buses, when buffers overflow we need to introduce NACKs.
- Directories should work well if only a small number of processors share common data at any given time (otherwise broadcasts are better).
Quantitative Motivation for Directories

- Number of invalidations per store miss on MSI with infinite caches

- Bottom-line: number of sharers for read-write data is small

Culler and Singh
Fig. 8.9
Example Implementation Difficulties

- Operations have to be serialized locally

  ![Diagram](Diagram.png)

- Operations have to be serialized at directory

  ![Diagram](Diagram.png)
Example Implementation Difficulties

- Operations have to be serialized locally
  1. P0 sends read request for line A.

- Operations have to be serialized at directory
Example Implementation Difficulties

- Operations have to be serialized locally
  1. P0 sends read request for line A.
  2. P1 sends read exclusive request for line A (waits at dir.).

- Operations have to be serialized at directory
Example Implementation Difficulties

- Operations have to be serialized locally
  1. P0 sends read request for line A.
  2. P1 sends read exclusive request for line A (waits at dir.).
  3. Dir. responds to (1), sets sharing vector (message gets delayed).

- Operations have to be serialized at directory
Example Implementation Difficulties

- **Operations have to be serialized locally**
  1. P0 sends read request for line A.
  2. P1 sends read exclusive request for line A (waits at dir.).
  3. Dir. responds to (1), sets sharing vector (message gets delayed).
  4a/b. Dir. responds to (2) to both P0 (sharer) and P1 (new owner).

- **Operations have to be serialized at directory**

```plaintext
1. P0 sends read request for line A.
2. P1 sends read exclusive request for line A (waits at dir.).
3. Dir. responds to (1), sets sharing vector (message gets delayed).
4a/b. Dir. responds to (2) to both P0 (sharer) and P1 (new owner).
```
Example Implementation Difficulties

- **Operations have to be serialized locally**
  1. P0 sends read request for line A.
  2. P1 sends read exclusive request for line A (waits at dir.).
  3. Dir. responds to (1), sets sharing vector (message gets delayed).
  4a/b. Dir. responds to (2) to both P0 (sharer) and P1 (new owner).
  5. P0 invalidates line A and sends acknowledgement

- **Operations have to be serialized at directory**
Example Implementation Difficulties

- **Operations have to be serialized locally**
  1. P0 sends read request for line A.
  2. P1 sends read exclusive request for line A (waits at dir.).
  3. Dir. responds to (1), sets sharing vector (message gets delayed).
  4a/b. Dir. responds to (2) to both P0 (sharer) and P1 (new owner).
  5. P0 invalidates line A and sends acknowledgement

    Problem: when (3) finally arrives at P0 the stale value of line A is placed in the cache. Solution: P0 must serialize transactions locally so that it won’t react to 4b while it has a read pending.

- **Operations have to be serialized at directory**
Example Implementation Difficulties

- Operations have to be serialized locally

  1. P0 sends read request for line A.
  2. P1 sends read exclusive request for line A (waits at dir.).
  3. Dir. responds to (1), sets sharing vector (message gets delayed).
  4a/b. Dir. responds to (2) to both P0 (sharer) and P1 (new owner).
  5. P0 invalidates line A and sends acknowledgement

Problem: when (3) finally arrives at P0 the stale value of line A is placed in the cache. Solution: P0 must serialize transactions locally so that it won’t react to 4b while it has a read pending.

- Operations have to be serialized at directory

  1. P1 sends read exclusive request for line A.

CS4/MSc Parallel Architectures - 2016-2017
Example Implementation Difficulties

- Operations have to be serialized locally

1. P0 sends read request for line A.
2. P1 sends read exclusive request for line A (waits at dir.).
3. Dir. responds to (1), sets sharing vector (message gets delayed).
4a/b. Dir. responds to (2) to both P0 (sharer) and P1 (new owner).
5. P0 invalidates line A and sends acknowledgement

Problem: when (3) finally arrives at P0 the stale value of line A is placed in the cache. Solution: P0 must serialize transactions locally so that it won’t react to 4b while it has a read pending.

- Operations have to be serialized at directory

1. P1 sends read exclusive request for line A.
2. Dir. forwards request to P0 (owner).

CS4/MSc Parallel Architectures - 2016-2017
Example Implementation Difficulties

- **Operations have to be serialized locally**
  1. P0 sends read request for line A.
  2. P1 sends read exclusive request for line A (waits at dir.).
  3. Dir. responds to (1), sets sharing vector (message gets delayed).
  4a/b. Dir. responds to (2) to both P0 (sharer) and P1 (new owner).
  5. P0 invalidates line A and sends acknowledgement

  Problem: when (3) finally arrives at P0 the stale value of line A is placed in the cache. Solution: P0 must serialize transactions locally so that it won’t react to 4b while it has a read pending.

- **Operations have to be serialized at directory**
  1. P1 sends read exclusive request for line A.
  2. Dir. forwards request to P0 (owner).
  3a/b. P0 sends data to P1 and ack. to dir. (ack gets delayed).
Example Implementation Difficulties

- **Operations have to be serialized locally**
  1. P0 sends read request for line A.
  2. P1 sends read exclusive request for line A (waits at dir.).
  3. Dir. responds to (1), sets sharing vector (message gets delayed).
  4a/b. Dir. responds to (2) to both P0 (sharer) and P1 (new owner).
  5. P0 invalidates line A and sends acknowledgement.

  Problem: when (3) finally arrives at P0 the stale value of line A is placed in the cache. Solution: P0 must serialize transactions locally so that it won’t react to 4b while it has a read pending.

- **Operations have to be serialized at directory**
  1. P1 sends read exclusive request for line A.
  2. Dir. forwards request to P0 (owner).
  3a/b. P0 sends data to P1 and ack. to dir. (ack gets delayed).
  4. P1 receives (3a) and considers read excl. complete. A replacement miss sends the updated value back to memory.
Example Implementation Difficulties

- Operations have to be serialized locally
  1. P0 sends read request for line A.
  2. P1 sends read exclusive request for line A (waits at dir.).
  3. Dir. responds to (1), sets sharing vector (message gets delayed).
  4a/b. Dir. responds to (2) to both P0 (sharer) and P1 (new owner).
  5. P0 invalidates line A and sends acknowledgement

Problem: when (3) finally arrives at P0 the stale value of line A is placed in the cache. Solution: P0 must serialize transactions locally so that it won’t react to 4b while it has a read pending.

- Operations have to be serialized at directory
  1. P1 sends read exclusive request for line A.
  2. Dir. forwards request to P0 (owner).
  3a/b. P0 sends data to P1 and ack. to dir. (ack gets delayed).
  4. P1 receives (3a) and considers read excl. complete. A replacement miss sends the updated value back to memory.

Problem: when (4) arrives dir. accepts and overwrites memory. When (3b) finally arrives dir. completes ownership transfer and thinks that P1 is the owner. Solution: dir. must serialize transactions so that it won’t react to 4 while the ownership transfer is pending.
Directory Overhead

- **Problem:** consider a system with 128 processors, 256GB of memory, 1MB L2 cache per processor, and 64byte cache lines
  - 128 bits for sharing vector plus 3 bits for state → ~16bytes
  - Per line: \( \frac{16}{64} = 0.25 \) → 25% memory overhead
  - Total: \( 0.25 \times 256G = 64GB \) of memory overhead!

- **Solution:** **Cached Directories**
  - At any given point in time there can be only \( \frac{128M}{64} = 2M \) lines actually cached in the whole system
  - Lines not cached anywhere are implicitly in state “not cached” with null sharing vector
  - To maintain only the entries for the actively cached lines we need to keep the tags → 64bits = 8bytes
  - Overhead per cached line: \( \frac{8+16}{64} = 0.375 \) → 37.5% overhead
  - Total overhead: \( 0.375 \times 2M = 768KB \) of memory overhead
Scalability of Directory Information

- **Problem:** number of bits in sharing vector limit the maximum number of processors in the system
  - Larger machines are not possible once we decide on the size of the vector
  - Smaller machines waste memory

- **Solution:** Limited Pointer Directories
  - In practice only a small number of processors share each line at any time
  - To keep the ID of up to $n$ processors we need $\log_2 n$ bits and to remember $m$ sharers we need $m$ IDs → $m*\log_2 n$
  - For $n=128$ and $m=4$ → $4*\log_2 128 = 28$ bits = 3.5 bytes
  - Total overhead: $(3.5/64)*256G = 14GB$ of memory overhead

- **Idea:**
  - Start with pointer scheme
  - If more than $m$ processors attempt to share the same line then trap to OS and let OS manage longer lists of sharers
  - Maintain one extra bit per directory entry to identify the current representation
Directories can be used with UMA systems, but are more commonly used with NUMA systems.

- In this case the directory is actually distributed across the system.
- These machines are then called cc-NUMA, for cache-coherent-NUMA, and DSM, for distributed shared memory.
Distributed Directories

- Now each part of the directory is only responsible for the memory lines of its node

- How are memory lines distributed across the nodes?
  - Lines are mapped per OS page to nodes
  - Pages are mapped to nodes following their physical address
  - Mapping of physical pages to nodes is done statically in chunks
  - E.g., 4 processors with 1MB of memory each and 4KB pages (thus, 256 pages per node)
    - Node 0 is responsible (home) for pages [0,255]
    - Node 1 is responsible for pages [256, 511]
    - Node 2 is responsible for pages [512, 767]
    - Node 3 is responsible for pages [768, 1023]
    - Load to address 1478656 goes to page 1478656/4096=361, which goes to node 361/256=1
Distributed Directories

- How is data mapped to nodes?
  - With a single user, OS can map a virtual page to any physical page → OS can place data almost anywhere, albeit at the granularity of pages
  - Common mapping policies:
    - **First-touch:** the first processor to request a particular data has the data’s page mapped to its range of physical pages
      - Good when each processor is the first to touch the data it needs, and other nodes do not access this page often
    - **Round-robin:** as data is requested virtual pages are mapped to physical pages in circular order (i.e., node 0, node 1, node 2, … node N, node 0, …)
      - Good when one processor manipulates most of the data at the beginning of a phase (e.g., initialization of data)
      - Good when some pages are heavily shared (**hot pages**)
    - Note: data that is only private is always mapped locally
  - Advanced cc-NUMA OS functionality
    - Mapping of virtual pages to nodes can be changed on-the-fly (**page migration**)
    - A virtual page with read-only data can be mapped to physical pages in multiple nodes (**page replication**)

Combined Coherence Schemes

- Use bus-based snooping in nodes and directory (or bus snooping) across nodes
  - Bus-based snooping coherence for a small number of processors is relatively straightforward
  - Hopefully communication across processors within a node will not have to go beyond this domain
  - Easier to scale up and down the machine size
  - Two levels of state:
    - Per-node at higher level (e.g., a whole node owns modified data, but Dir. does not know which processor in the node actually has it)
    - Per-processor at lower level (e.g., by snooping inside the node we can find the exact owner and the exact up-to-date value)
References and Further Reading

- **Original directory coherence idea:**

- **Seminal work on distributed directories:**

- **A commercial machine with distributed directories:**

- **A commercial machine with SCI:**

- **Adaptive full/limited pointer distributed directory protocols:**
Probing Further

- **Page migration and replication for ccNUMA**

- **Cache Only Memory Architectures**

- **Recent alternative protocols: token, ring**