Different caches may contain different values for the same memory location.

### Cache coherence in multiprocessors (sec. 5.10)

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Cache Contents for CPU 1</th>
<th>Cache Contents for CPU 2</th>
<th>Memory Contents for location X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>CPU 1 Reads X</td>
<td>X = 1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>CPU 2 reads X</td>
<td>X = 1</td>
<td>X = 1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>CPU 1 stores 0 into X</td>
<td>X = 0</td>
<td>X = 1</td>
<td>0 if write through 1 if write back</td>
</tr>
</tbody>
</table>

Single bus

memory

X = 1

I/O
Approaches to cache coherence

- Do not cache shared data
- Do not cache writeable shared data
- If connected by a bus, use snoopy caches (see following slides)
- If no shared bus, then
  - Use broadcast to emulate shared bus
  - Use directory-based protocols (to communicate only with concerned parties, not with everybody – directory keeps track of concerned parties)

Snooping cache coherence

EXAMPLE: A coherence protocol for **write through** caches

- Snooping → each processor monitors the activity on the bus
- **If a write is posted on the bus**, each cache checks to see if it has a copy of the block. If yes, it either
  - update its copy with the new value (update protocol), or
  - invalidate its copy (invalidation protocol).
A snooping protocol for write back caches

- If a processor writes to a block in its cache, it sends an "invalidate" message on the bus indicating that it will EXCLUSIVELY own this block.
- If another cache has a copy of the block, it invalidates it.
- Subsequent accesses to an exclusive block does not need further invalidation.

A block is EXCLUSIVE in one cache means that it is **dirty** in this cache (its value is different from memory) and no other cache has a copy of that the block.

A block that is cached in multiple caches is said to be in a SHARED state if its value is consistent across the caches and the memory (clean blocks).

Hence:
- On a read miss, a block is brought into a cache and marked as SHARED.
- On a write miss, a block is brought into a cache and marked as EXCLUSIVE
  - Other caches that have the block will notice the read request on the bus and invalidate their copies.
A snooping protocol for write back caches

- If a block is Exclusive in one cache (ex. cache 8).
- And another cache (ex. cache 1) wants to read or write the block
  - It puts a request on the bus
- The cache that has the block in the EXCLUSIVE state will
  - Detect that it has an exclusive block requested by another cache
  - Supply the block by putting it on the bus
  - Set the block to SHARED if the request is to read, otherwise Invalidate it
- Both the memory and the requesting cache will get the correct block

MSI: an invalidation protocol for write back caches

- In each cache, we record the state of each block, B, as one of:
  - Shared: block is clean (and may be also be in other caches)
  - Exclusive (Modified): cache has only copy, it is writeable, and dirty
  - Invalid: block not in cache (entry in cache is either invalid or contains another block)

Protocol actions (when CPUᵢ accesses a block, B, in its cache, Cᵢ):

- On a read hit, the state of B does not change
- On a read miss (block B is invalid in Cᵢ) a “request to read B“ is generated on the bus:
  If another cache has B “exclusively”, it writes it back and B becomes “Shared” in both
  caches. Otherwise the memory supplies B. In all cases, Cᵢ sets the state of B to “Shared”.
- On a write hit
  - If B is Shared, an “invalidate” message is put on the bus -- all other caches that
    have B should invalidate it – B becomes “Exclusive” in the requesting cache, Cᵢ.
  - If B is “Exclusive”, no invalidate message is issued – B stays “Exclusive” in Cᵢ.
- On a write miss (block B is invalid in Cᵢ) a “request to write B“ is generated on the bus:
  - If no other cache has B, the memory will supply B.
  - If another cache has B as “Shared”, it invalidates it (the memory will supply B)
  - If another cache has B in “exclusive”, it writes back B and “invalidates” it in its cache
  - In all cases, B is set to “Exclusive” in Cᵢ.
Example of MSI coherence

- Assumes that blocks A and B map to the same cache location L.
- Block size = one word
- Initially neither A nor B is cached processors P1 and P2 caches’

<table>
<thead>
<tr>
<th>Event</th>
<th>In P1’s cache</th>
<th>In P2’s cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 writes 10 to A (write miss)</td>
<td>P1 requests A (to write)</td>
<td>P1 requests A (to write)</td>
</tr>
<tr>
<td></td>
<td>L ← A = 10 (exclusive)</td>
<td>L = invalid</td>
</tr>
<tr>
<td>P1 reads A (read hit)</td>
<td>L ← A = 10 (exclusive)</td>
<td>L = invalid</td>
</tr>
<tr>
<td>P2 reads A (read miss)</td>
<td>P1 writes A back</td>
<td>P2 requests A (to read)</td>
</tr>
<tr>
<td></td>
<td>L ← A = 10 (shared)</td>
<td>L ← A = 10 (shared)</td>
</tr>
<tr>
<td>P2 writes 20 to A (write hit)</td>
<td>L = invalid</td>
<td>Put invalidate A on bus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L ← A = 20 (exclusive)</td>
</tr>
<tr>
<td>P2 writes 40 to A (write hit)</td>
<td>L = invalid</td>
<td>L ← A = 40 (exclusive)</td>
</tr>
<tr>
<td>P1 write 45 to A (write miss)</td>
<td>P1 requests A (to write)</td>
<td>P2 writes A back</td>
</tr>
<tr>
<td></td>
<td>L ← A = 45 (exclusive)</td>
<td>L = invalid</td>
</tr>
</tbody>
</table>

In P1’s cache

- L = invalid
- L = invalid

In P2’s cache

- L = invalid

Note: False sharing can occur when block size > one word.
Example: - x and y are in the same cache block
- P1 repeatedly write x and not y, P2 repeatedly write y and not x
This 13.5 by 19.6 mm die has 731 million transistors. It contains four processors that each have private 32-KB instruction and 32-KB data caches and a 512-KB L2 cache. The four cores share an 8-MB L3 cache. The coherence is between the L2 caches and the L3.