Shared memory

Caches, Cache coherence and Memory consistency models

Diego Fabregat-Traver and Prof. Paolo Bientinesi

HPAC, RWTH Aachen
fabregat@aices.rwth-aachen.de

WS15/16

Shared memory

Caches, Cache coherence and Memory consistency models

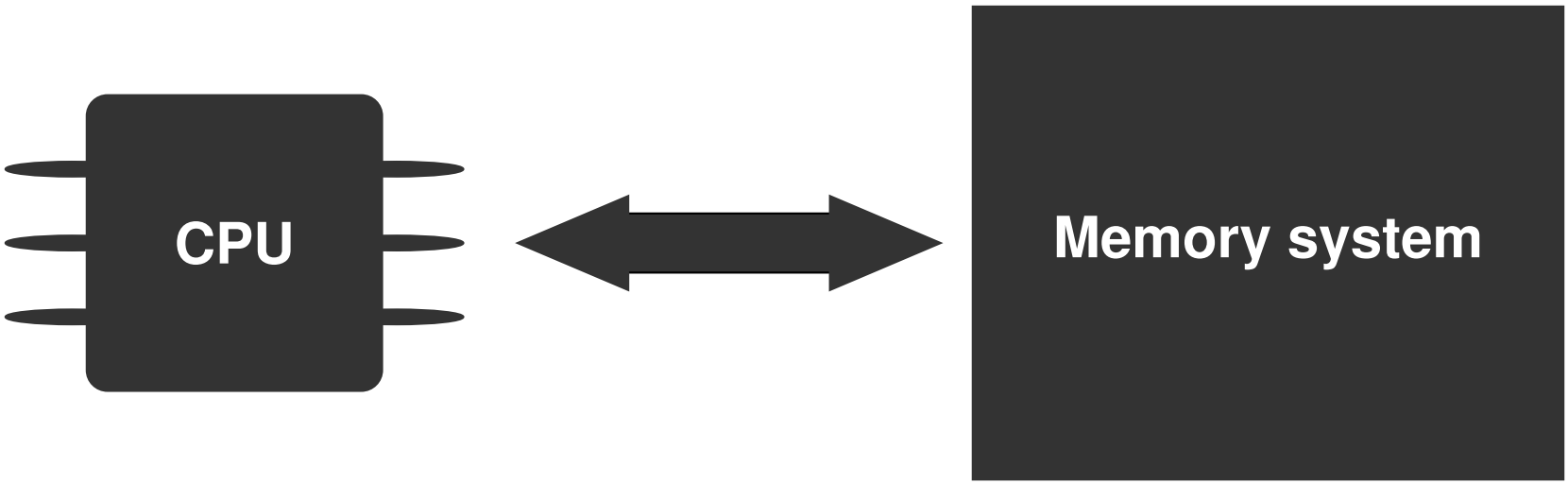
References

- Computer Organization and Design. *David A. Patterson, John L. Hennessy*. Chapter 5.
- Computer Architecture: A Quantitative Approach. *John L. Hennessy, David A. Patterson*. Appendix 4.
- Only if interested in much more detail on cache coherence and memory consistency: A Primer on Memory Consistency and Cache Coherence. *Daniel J. Sorin, Mark D. Hill, David A. Wood*.

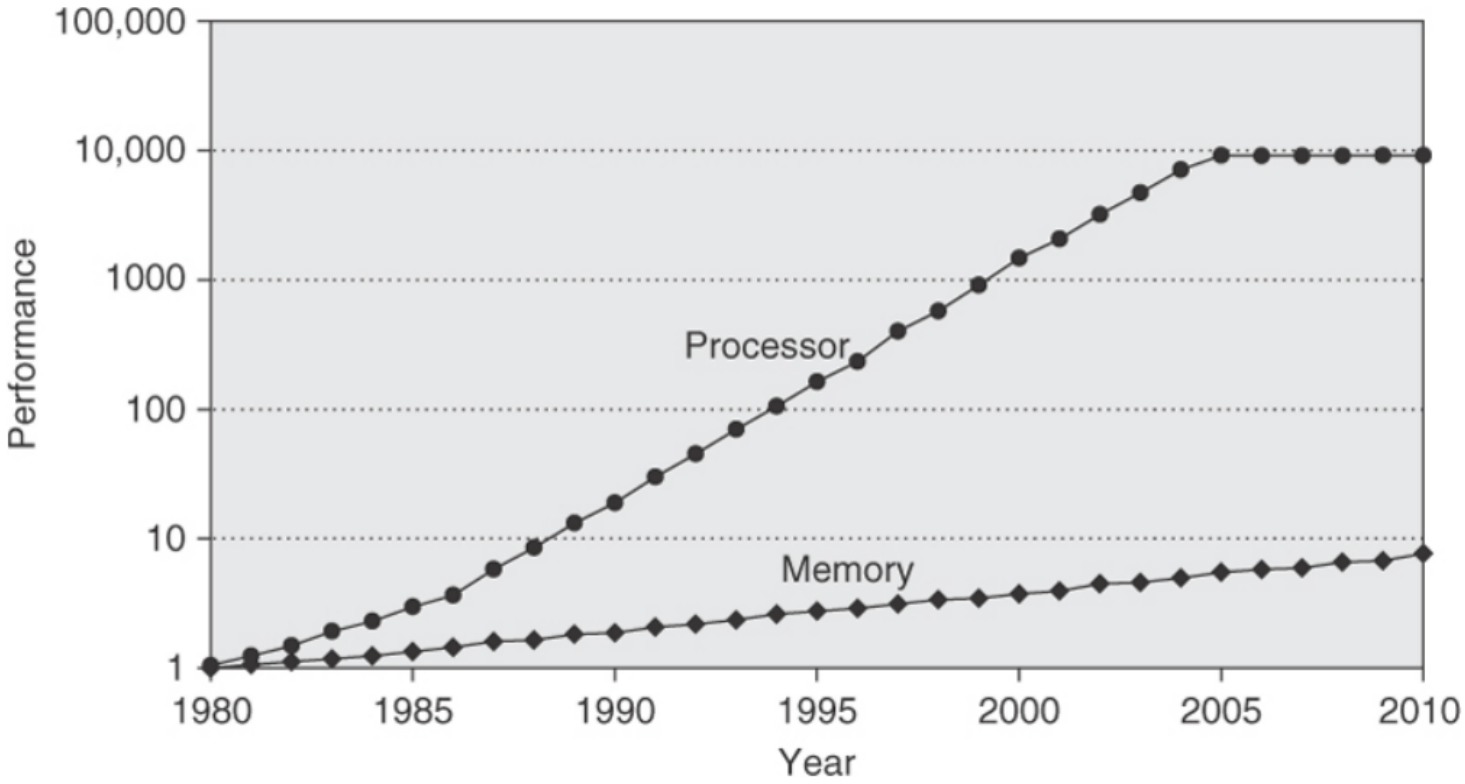
Outline

The basic picture

Programmers dream: Unlimited amount of fast memory



Speed divergence

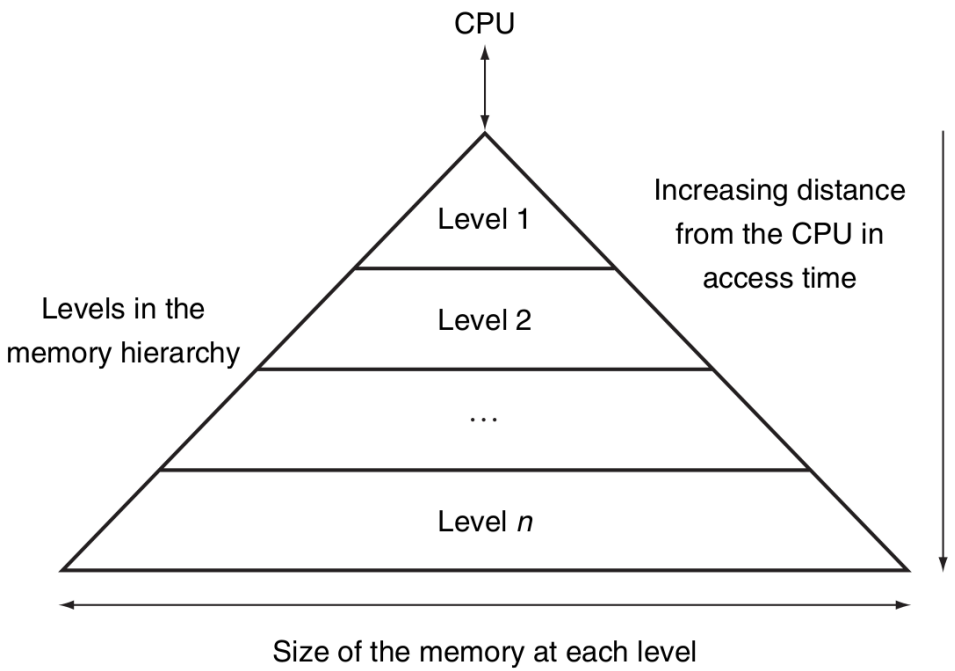


Gap in performance between CPU and main memory.

Computer organization and design. Patterson, Hennessy.

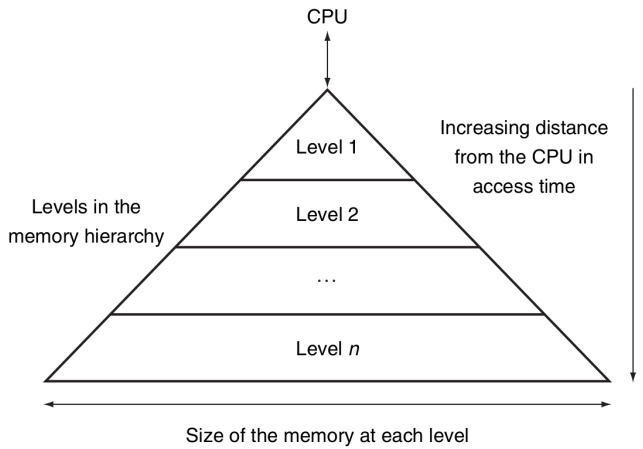
Memory hierarchy

Illusion of large and fast memory



Memory hierarchy

Illusion of large and fast memory



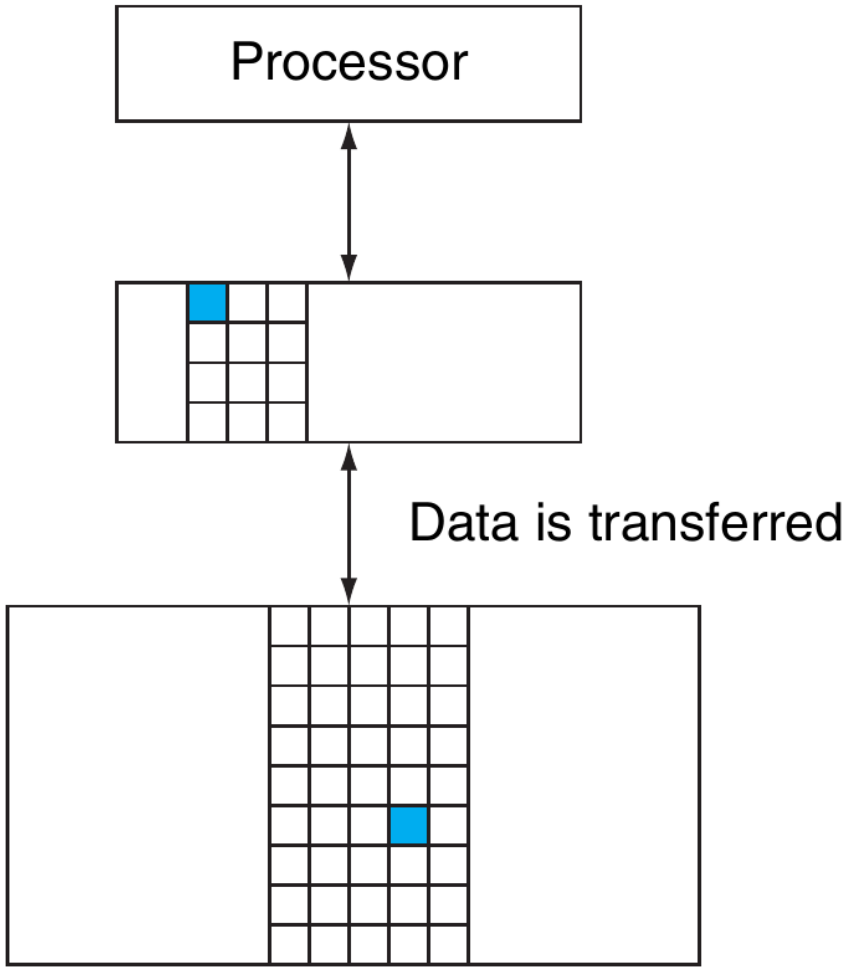
Underlying idea: *principle of locality*

- Temporal locality: if an item is referenced, it will tend to be referenced again soon (e.g., instructions and data in a loop).
- Spatial locality: if an item is referenced, items whose addresses are close by will tend to be referenced soon (e.g., arrays).

Outline

Cache memories

- Cache hierarchy may consist of multiple levels
- Data copied between two adjacent levels
- Cache block (or line): unit of data transfer
- Cache consists of multiple such blocks



Cache memories (II)

- Cache hit: data requested by the processor is present in some block of the upper level of cache
- Cache miss: data requested by the processor is *not* present in any block of the upper level of cache

In a cache miss the lower levels in the hierarchy are accessed to retrieve the block containing the requested data.

The data is then supplied from the upper level.

Line placement

- How do we know whether a line is in cache?
- Where do we look for it?

Line placement

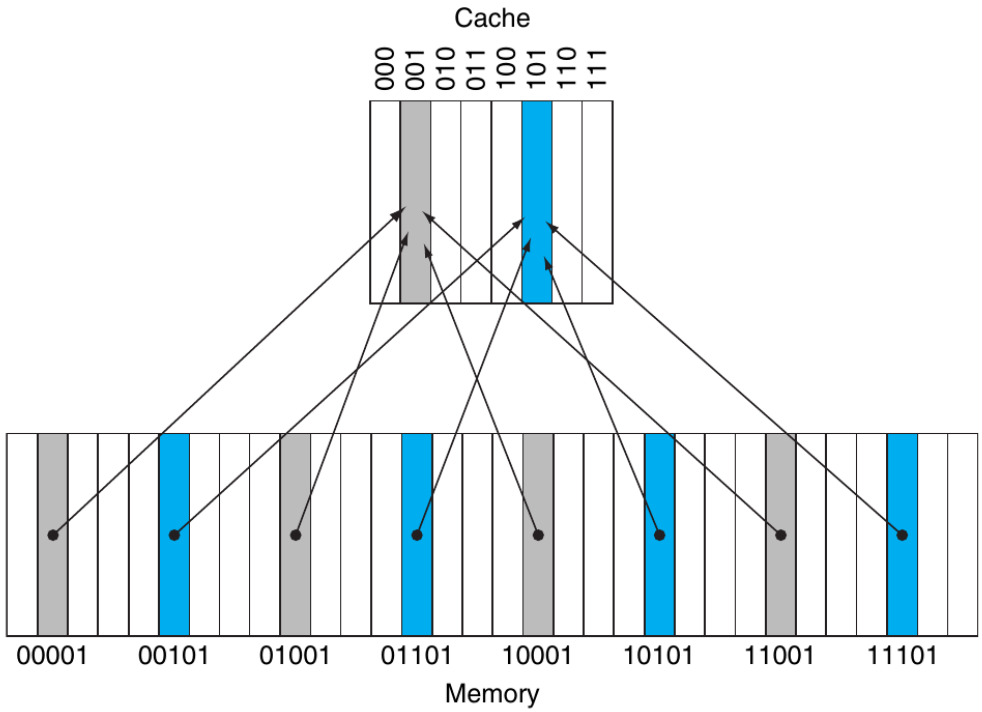
- How do we know whether a line is in cache?
- Where do we look for it?

- Direct-mapped: Each block can go in exactly one place in the cache
- Fully associative: Each block can go anywhere in the cache
- Set associative: Each block can go anywhere within a restricted set of entries in the cache

Line placement

Direct-mapped

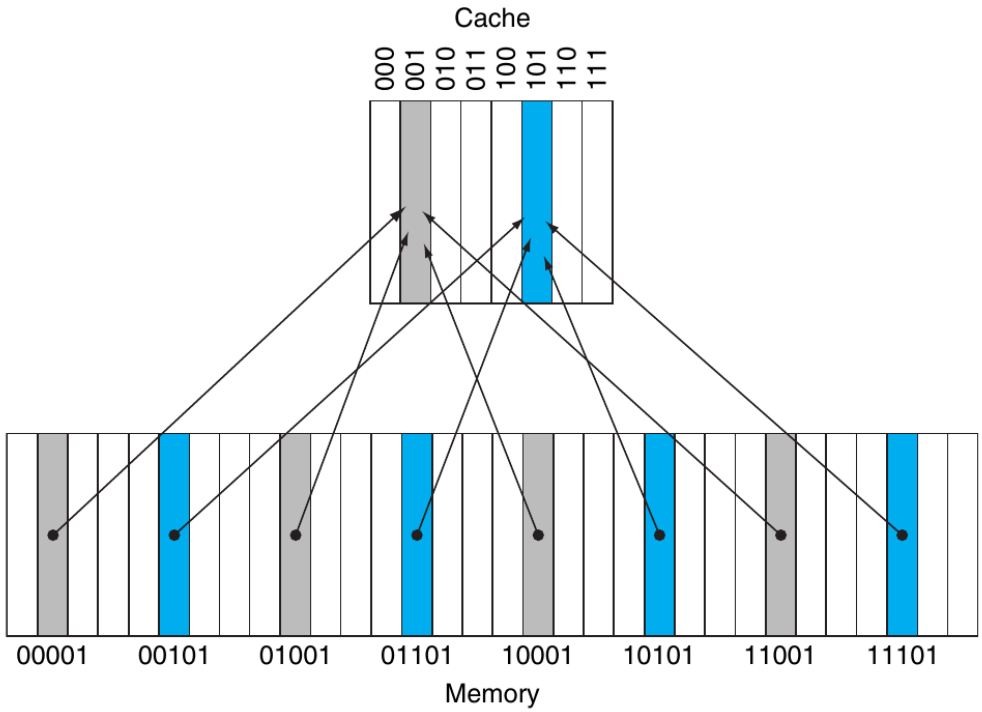
Mapping: (Block address) modulo (Number of blocks in the cache)



Line placement

Direct-mapped

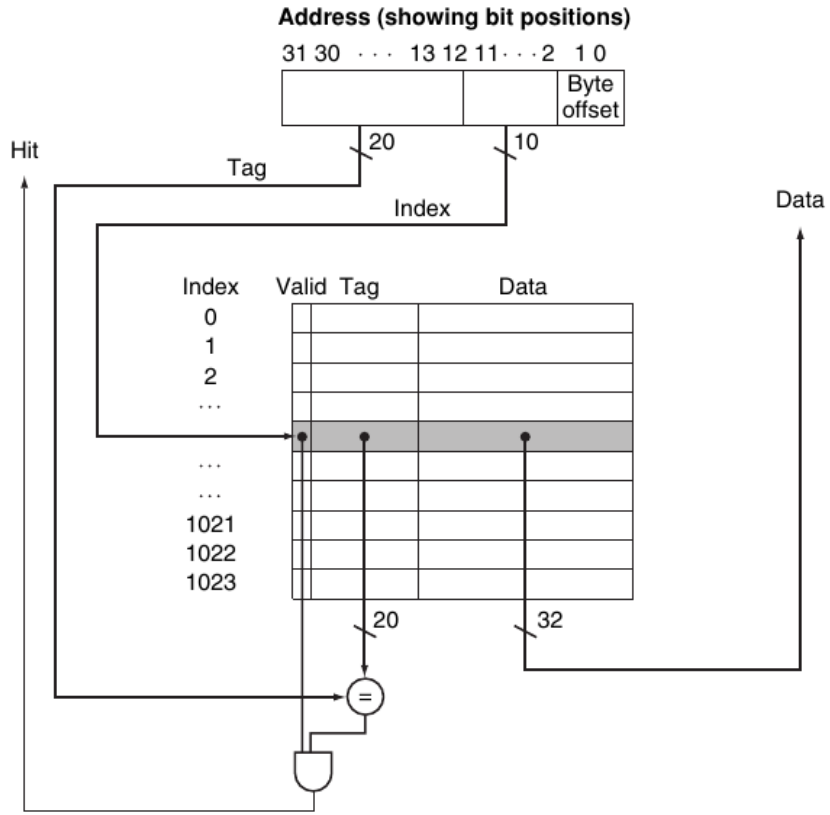
Mapping: (Block address) modulo (Number of blocks in the cache)



How do we know whether cache block `001` contains `MEM[00001]`, `MEM[01001]`, `MEM[10001]`, ...?

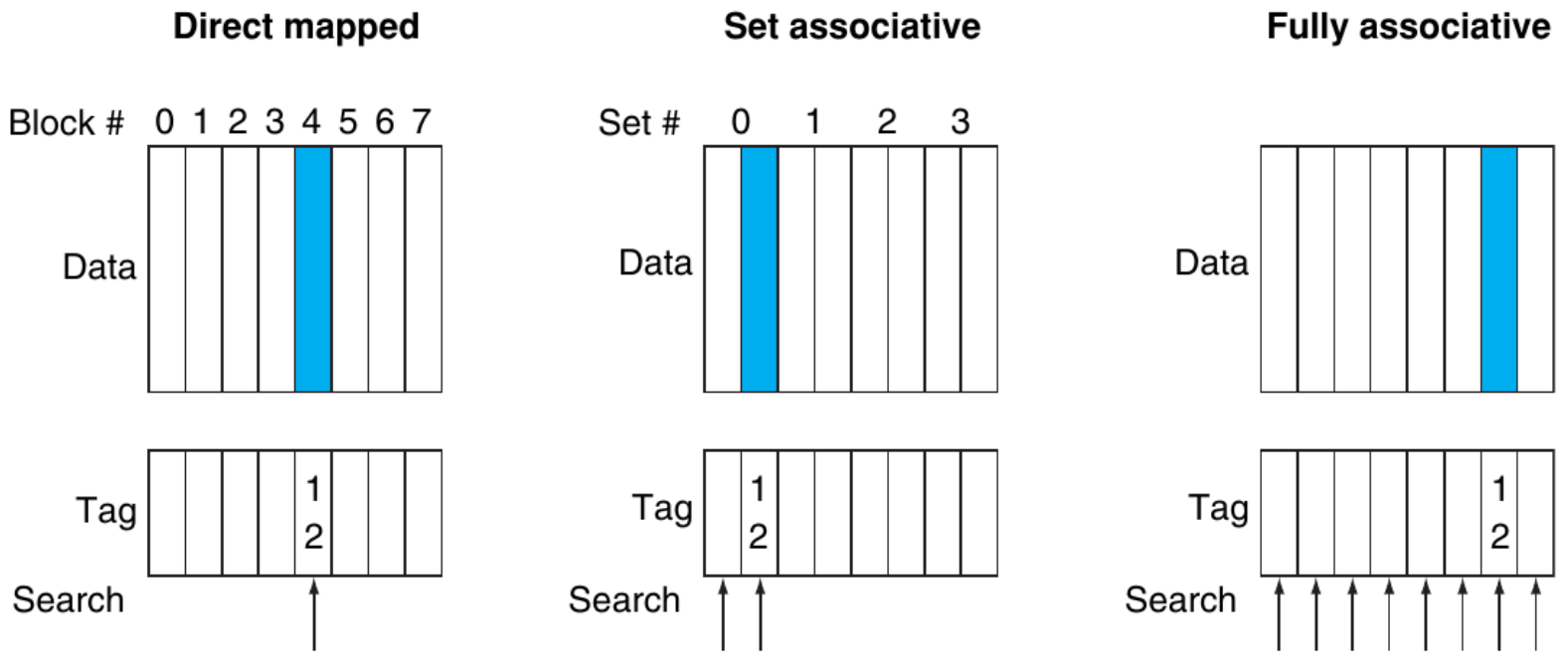
Line placement

Direct-mapped



- Index: block selection
- Tag: compare block with memory address
- Valid bit: whether block contains valid data

Line placement



- Direct-mapped: Simple scheme, simple tag comparison, high #misses
- Fully associative: Complex tag comparison, low #misses
- *n-way* associative: Compromise, multiple tags, lower #misses than DM

Line replacement policies

- Random: all lines are candidates to be evicted (replaced)
- Least recently used (LRU): replace the block that has been unused for the longest time.
  Ok for 2-way, maybe 4-way. Too hard beyond.
- Approximated LRU: “Hierarchical” LRU. Example 4-way:
  1 bit determines the pair with the oldest (LRU) pair of blocks. 1 bit determines the LRU block within the pair.

Writing data

Write-through

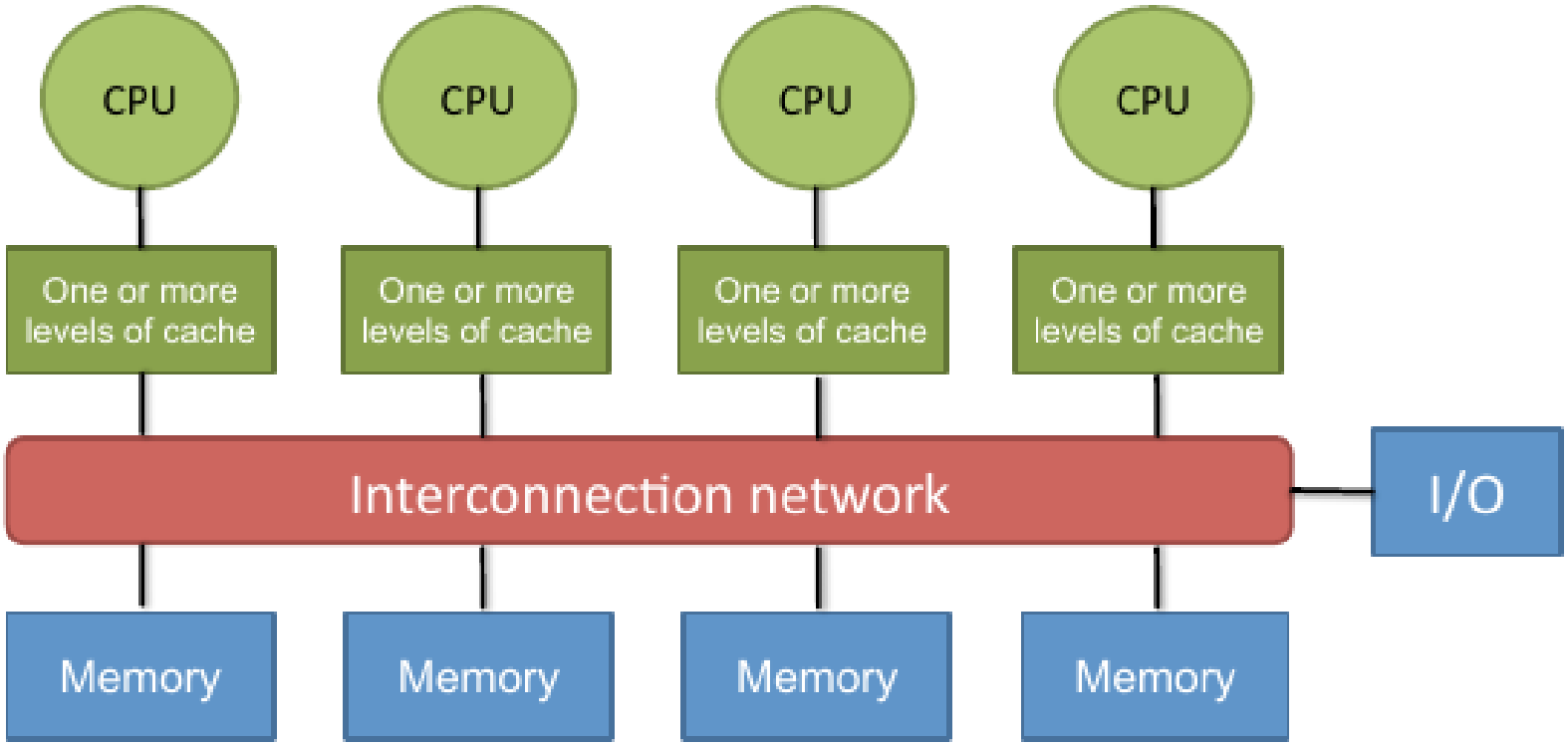
- On a hit, processor could simply update only the block in the cache.
  - *Inconsistency problem*: cache and memory hold different values.
- Simplest solution: write-through, i.e., update both cache and memory.
- On a miss: fetch block into cache, update, write to memory.
- Problem: Low performance. A write may take at least 100 clock cycles, slowing down the processor. Can be alleviated with write buffers.

Write-back

- On a hit, update only the block in cache.
  - Keep track of whether each block is *dirty*.
- When a dirty block is replaced, write to the lower level of the hierarchy.
- On a miss: typically, also fetch block into cache and update.

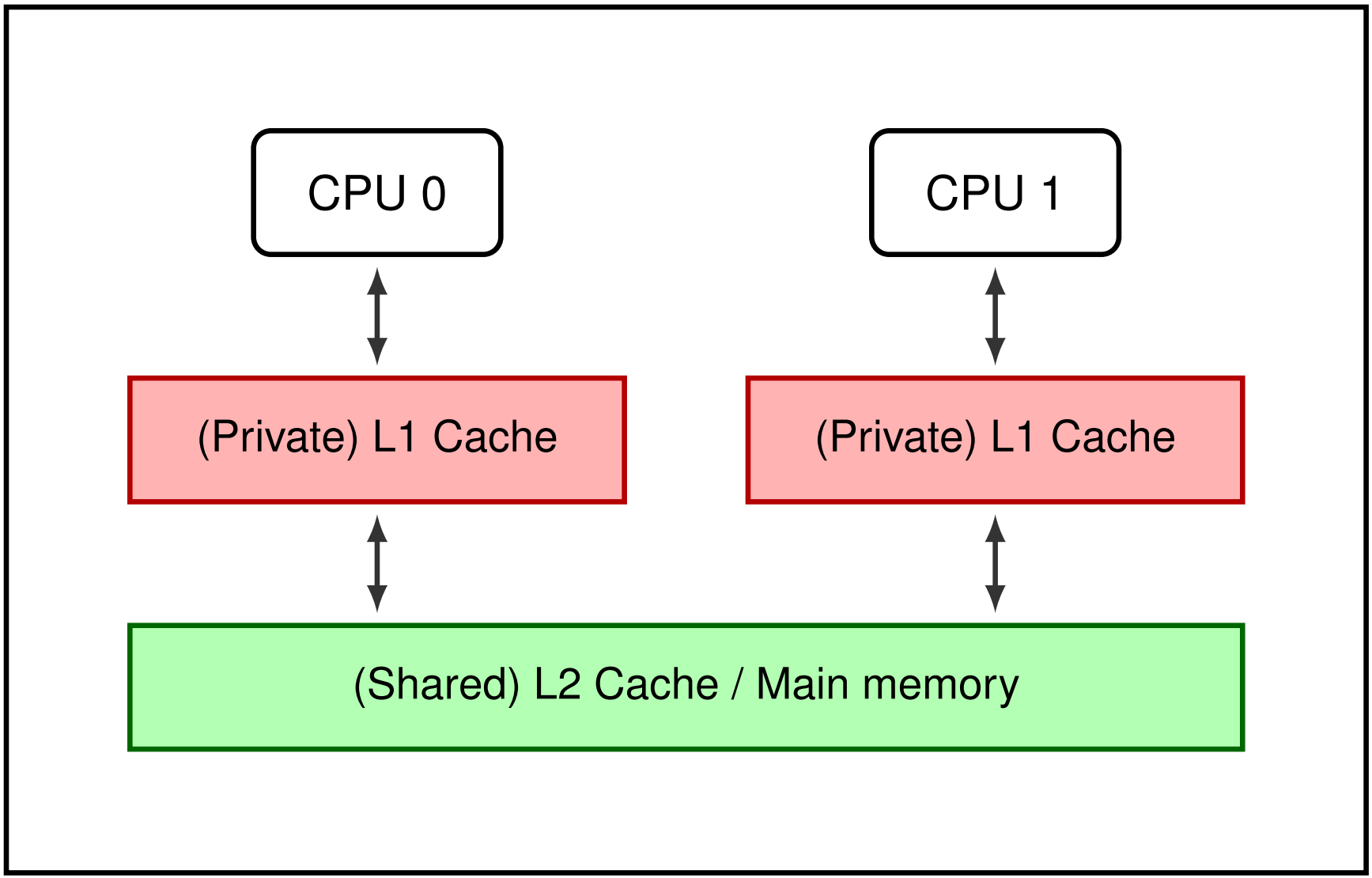
Outline

Shared-memory parallel architectures



Shared-memory parallel architectures

Not every memory is shared!



Cache coherence

- View of memory of two different processors is through their caches.

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

Cache coherence

- View of memory of two different processors is through their caches.

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

- *Cache coherence problem*: Two different processors can have two different values for the same location.
- *Coherence*: Which value will be returned by a read?

Coherence definition

Idea: Cache coherence aims at making the caches of a shared-memory system as functionally invisible as the caches of a single-core system.

Properties:

- Processor P writes to location X, then P reads from X (with no other processor writing to X in between). The read must return the value written by P.

  Preserves program order.

- A read by processor P_1 from X after a write by processor P_2 to X returns the value written by P_2 if no other writes to X occur between the two accesses.

  Defines the notion of coherent view of the memory.

- Two writes to the same location by any two processors are seen in the same order by all processors. Prevents two processors to end up with two different values of X.

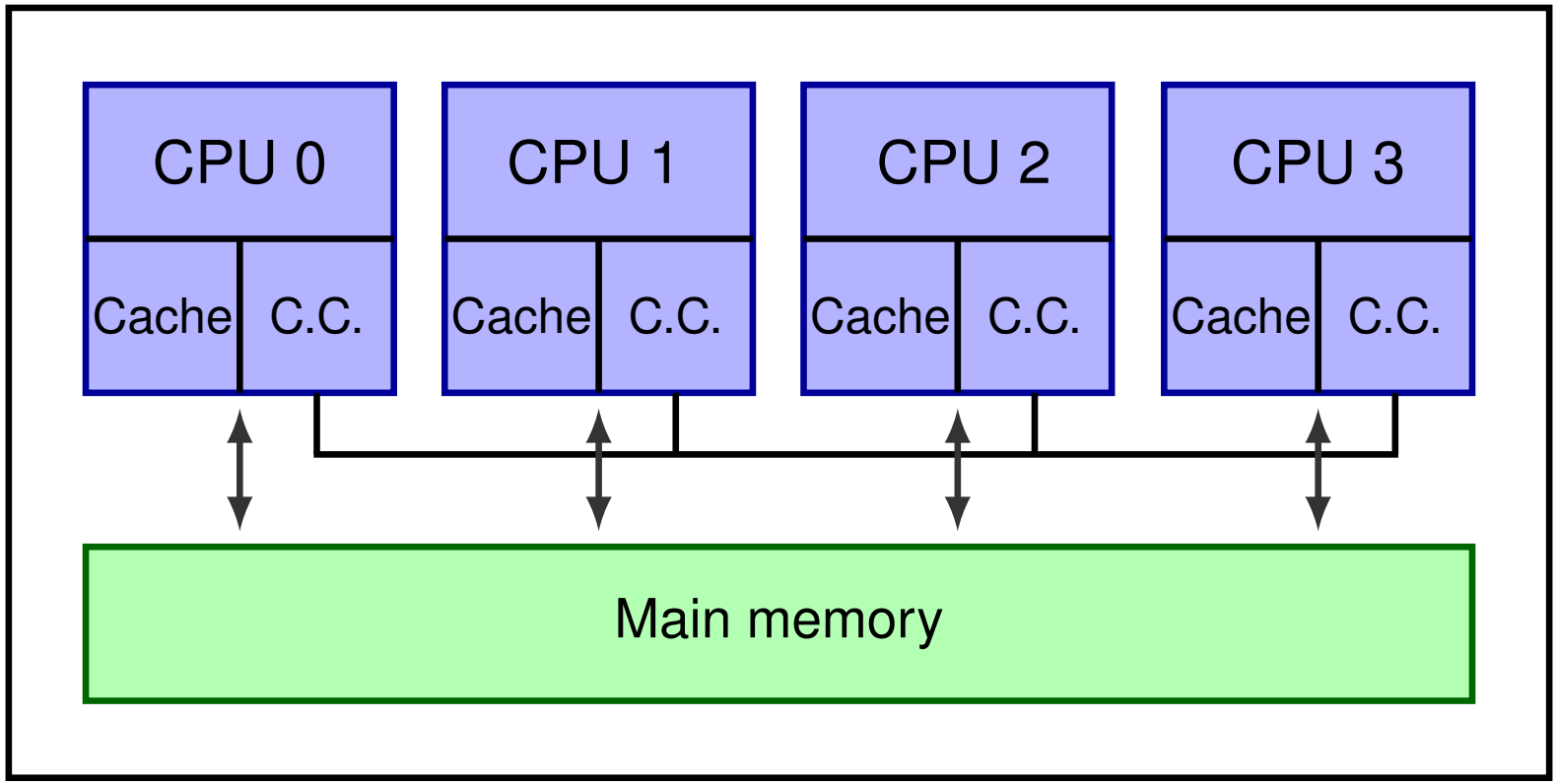
  Write serialization.

Cache coherence protocols

- Snooping protocols
  - Decentralized approach
  - Each cache line with a copy of a memory location has a label indicating the sharing state of the line
  - All caches connected via a broadcast medium (e.g., bus)
  - Cache controllers snoop (listen) on the medium to determine whether they hold a copy of a line that is requested
- Directory-based protocols
  - The state of a block of memory is kept in one location (centralized)
  - More scalable

Snooping protocol

Cache controllers (C.C.) connected via a bus which they monitor (snoop).



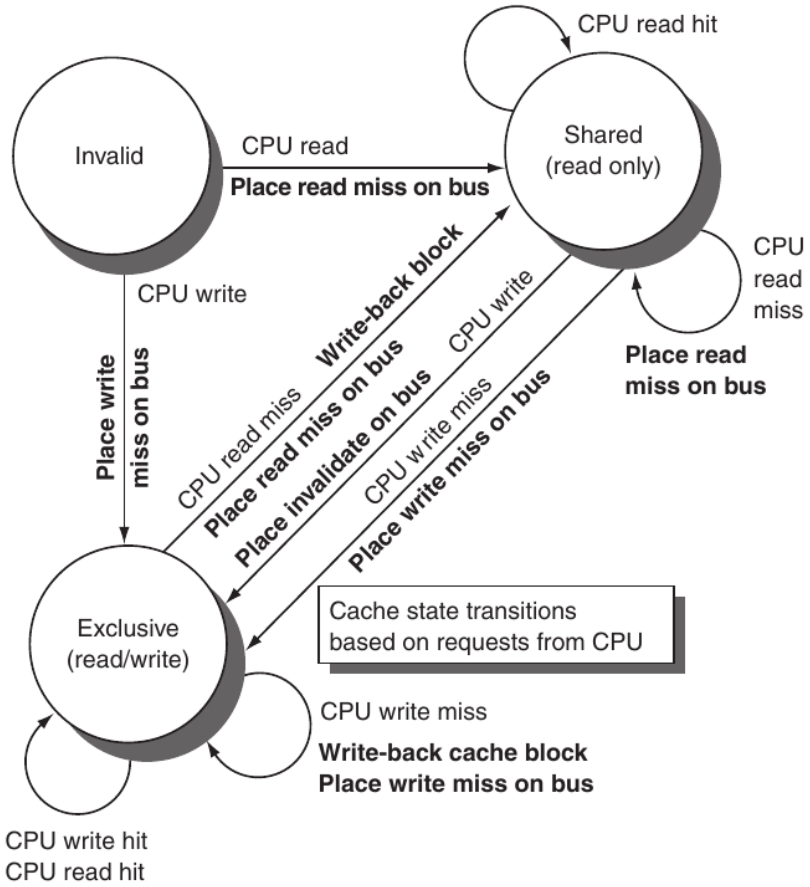
Snooping protocol

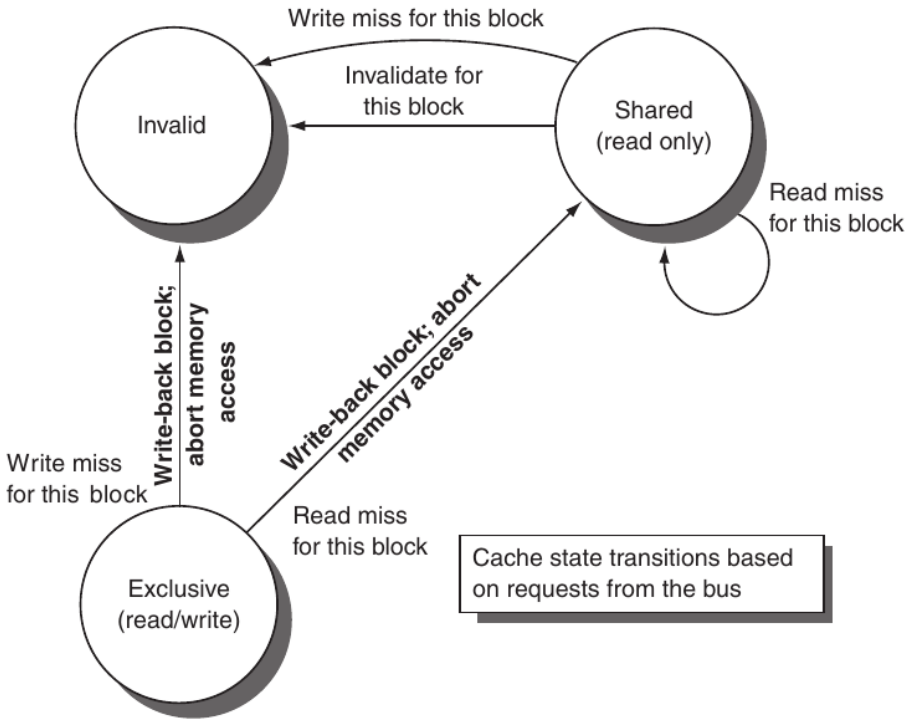
- Three states: exclusive (read/write), shared (read-only), invalid
- On a write, all other copies in cache are invalidated
- Guarantees exclusive access

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

Snooping protocol

Finite-state machine controller





Note 1: True and False sharing

- True sharing: A cache miss occurs because a block was invalidated by another processor writing the same word.
  - Necessary to keep coherence
- False sharing: A cache miss occurs because a block was invalidated by another processor writing a different word.
  - Would not occur is block size were one word

Note 2: Data races and synchronization

- Beware, CC does not “solve” race condition situations

CPU 0	CPU 1
t1 = load(X)	
t1 = t1+1	t2 = load(X)
X = store(t1)	t2 = t2 + 1
	X = store(t2)

- We need critical sections to ensure mutual exclusion

Outline

Memory Consistency Models

- Coherence guarantees that (i) a write will eventually be seen by other processors (ii) all processors see writes to the *same location* in the same order
- Consistency model defines the (allowed) ordering of reads and writes to *different* memory locations. The hardware guarantees a certain consistency model and the programmer attempts to write correct programs with those assumptions.

Memory Consistency Models

Sequential consistency

- Within a thread of execution, ordering is preserved
- Instructions from different threads may interleave arbitrarily
- Example:

CPU 0	CPU 1
instr a	instr A
instr b	instr B
instr c	instr C
instr d	instr D

Valid executions: abcdABCD, aAbBcCdD, abAcBCDd, ...

Memory Consistency Models

Sequential consistency

- Why is preserving the ordering within a thread so important?
- Consider the following code:

```
    CPU 0           |        CPU 1
--------------------------------------------
S1: data = NEW;     |
S2: flag = SET;     |    L1: r1 = flag;
                    |    B1: if (r1 != SET) goto L1;
                    |    L2: r2 = data;
```

Assume: initially, `data = OLD` and `flag = UNSET`

Memory Consistency Models

Sequential consistency

- Why is preserving the ordering within a thread so important?
- Consider the following code:

```
    CPU 0           |        CPU 1
--------------------------------------------
S1: data = NEW;     |
S2: flag = SET;     |    L1: r1 = flag;
                    |    B1: if (r1 != SET) goto L1;
                    |    L2: r2 = data;
```

  Assume: initially, `data = OLD` and `flag = UNSET`

- As a result, should `r2` always be set to `NEW`?

Memory Consistency Models

Sequential consistency

- Why is preserving the ordering within a thread so important?
- Consider the following code:

```
    CPU 0           |        CPU 1
--------------------------------------------
S1: data = NEW;     |
S2: flag = SET;     |    L1: r1 = flag;
                    |    B1: if (r1 != SET) goto L1;
                    |    L2: r2 = data;
```

Assume: initially, `data = OLD` and `flag = UNSET`

- As a result, should `r2` always be set to `NEW`?
- What if order is `S2, L1, L2, S1`?

Memory Consistency Models

Sequential consistency

- Very intuitive
- Easy for programmers
- Problems:
  - Limited room for optimization
  - Slower execution

Memory Consistency Models

Relaxed consistency models

- We, as programmers, care about ordering only in a few cases (e.g., synchronization)
- Relax ordering constraints in general, use fences (barriers) to complete all previous memory accesses before moving on (sequential consistency)
- Programmers are responsible to set these fences or synchronization points.

Summary

- Memory hierarchy: illusion of large and fast memory
- Cache coherence:
  - Behavior of the memory system when a single memory location is accessed by multiple threads
  - Coherence protocols: snooping, directory-based
- Memory consistency model:
  - Define the allowed orderings of reads and writes to different memory locations
  - Most multicore processors implement some kind of relaxed model
  - Most programs rely on predefined synchronization primitives