CS/ECE 5381/7381 Computer Architecture Spring 2023

Dr. Manikas

Computer Science

Lecture 12: Mar. 7, 2023

Review of Memory Hierarchy

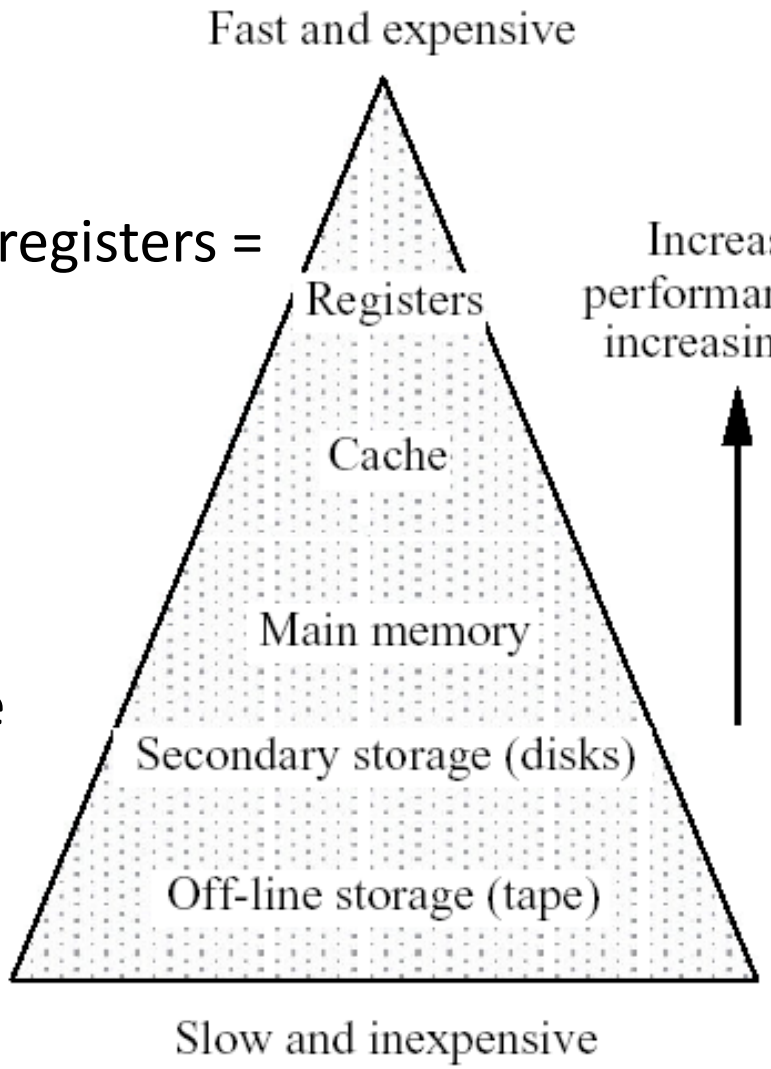
(Appendix B, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Outline

- B.1 Introduction
- B.2 Cache Performance
- B.3 Basic Cache Optimizations
- B.4 Virtual Memory

Memory Hierarchy



32 32-bit (4 B) registers = 128 B

8 GB RAM

1 TB hard drive (disk or SSD)

15 TB tape reel

Make this small to reduce costs

This can be large since cheap

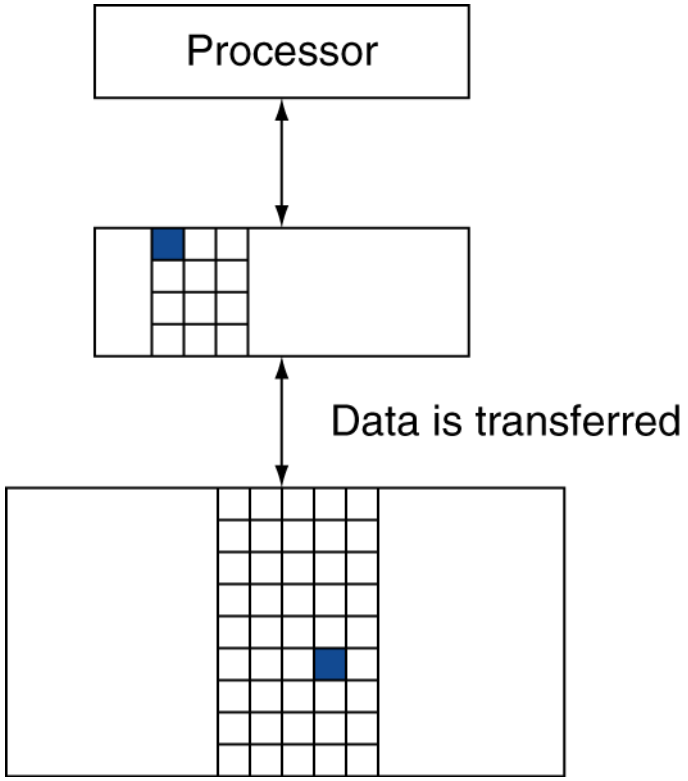
Memory Hierarchy

- Ideal memory
  - Fast access time, large capacity, low cost
- We don't have this in any *one* component
- But we can use a *combination* of components to "appear" as "ideal memory" to CPU
  - This combination is called a *memory hierarchy*
  - Use the "principle of locality"

Principle of Locality

- Programs access a small proportion of their address space at any time
- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, instruction variables
- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

Memory Hierarchy Levels



- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 – hit ratio
  - Then accessed data supplied from upper level

Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper (primary) level
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of:

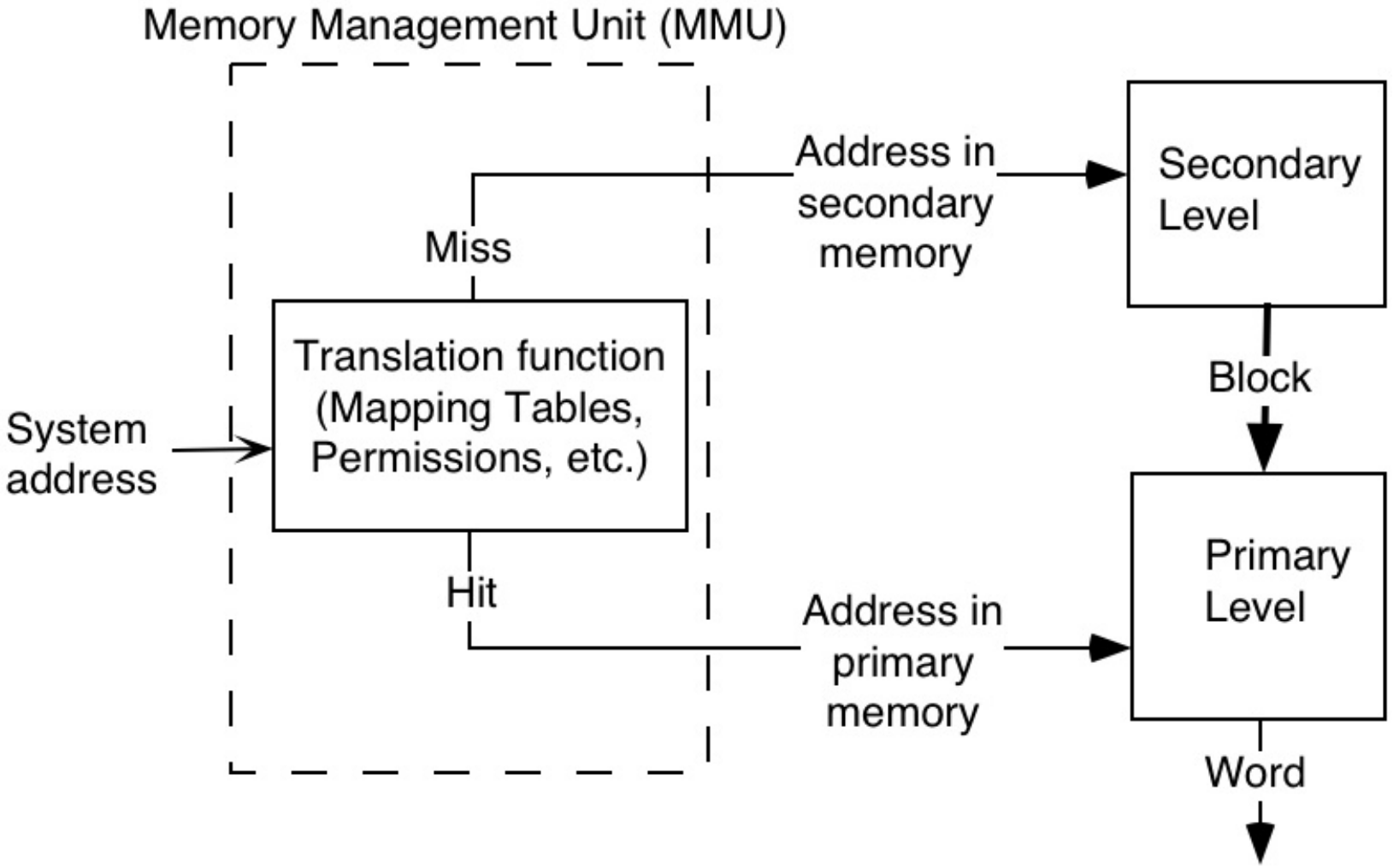
    Primary level access time + Time to determine hit/miss

Memory Hierarchy: Terminology (cont)

- Miss: data needs to be retrieved from a block in the lower (secondary) level
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level +

    Time to deliver the block the processor
- Hit Time << Miss Penalty

Addressing and Accessing a 2-Level Hierarchy



Hits and Misses

- **Miss ratio** = #misses / total # of refs in the same program-execution period
- **Hit ratio** = 1 - miss ratio
  - Let t_p = primary memory access time,
  - t_s = secondary memory access time,
  - h = hit ratio
- Then avg access time for two-level memory hierarchy $t_a = ht_p + (1-h)t_s$
- The qty $t_s > t_p$, so design goal is to minimize miss ratio

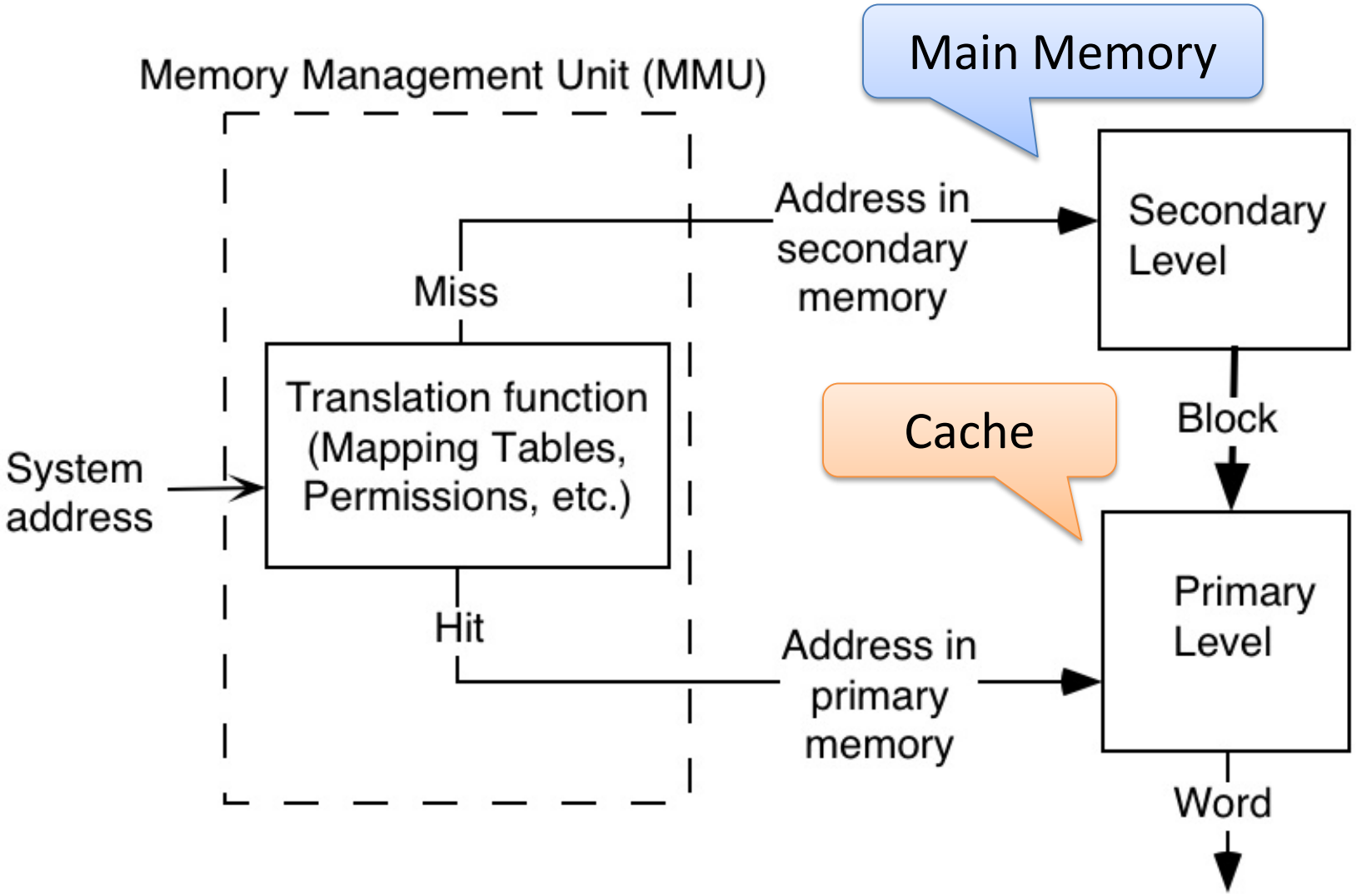
Example B.1-1

- Assume that we have a two-level memory hierarchy with the following memory access times:
  - Primary = 10 ns
  - Secondary = 100 ns (10 times slower)
- What is the **average access time** for this memory hierarchy if the hit rate is **90%** (0.9)?
- What is the **average access time** for this memory hierarchy if the hit rate is **75%** (0.75)?

Cache

- A fast memory (RAM) used to hold active part of program and data
- Speeds up operation of active program
- Often incorporated into CPU
- Operates on *Principle of Locality* (recall from earlier)
  - Part of *2-level memory hierarchy*

Addressing and Accessing a 2-Level Hierarchy



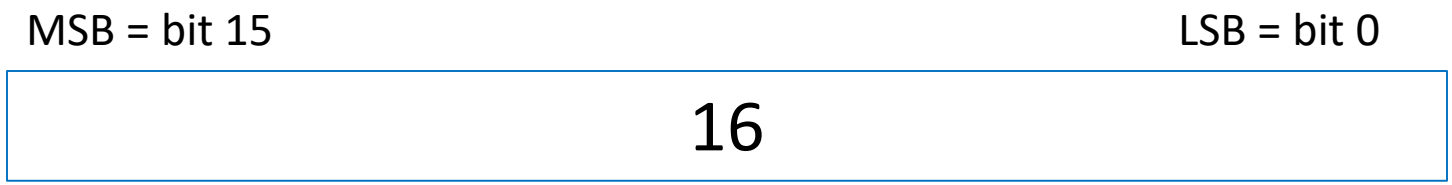
Interpreting Memory Addresses

- Memory size
  - 2^{10} = 1 K (kilo)
  - 2^{20} = 1 M (mega)
  - 2^{30} = 1 G (giga)
  - 2^{40} = 1 T (tera)
  - 2^{50} = 1 P (peta)
- 1 B (byte) = 8 b (bits)

CPU Memory Address (no cache)

Assume we have a main memory with 64K bytes. If the processor accesses main memory directly (no cache), then it needs an address bus with enough lines to access all 64K bytes.

$64K = 2^6 2^{10} = 2^{16}$ bytes => 16-bit address



CPU Memory Address (with cache)

Now add a cache between the processor and main memory.

Cache views main memory as blocks of bytes. For our example, assume that each block contains 8 bytes.

Then, the cache views our 64K byte main memory as follows:

$$\frac{64KB}{8\frac{B}{block}} = \frac{2^{16}}{2^3} = 2^{13} \text{ blocks}$$

So the main memory is viewed as having $2^3 2^{10}$ = 8K blocks, or 8192 blocks.

Cache Size

Cache views main memory as a group of blocks. For our example, we have a group of 8K (8192) blocks in main memory.

The cache is much smaller than main memory (cache is faster, so more cost/bit). For our example, let the cache size be 256 (2^8) blocks.

CPU Memory Address: Cache Mapping

- The CPU memory address is mapped to the cache
- Mapping scheme depends on the type of cache
  - Associative
  - Direct
  - N-way Set-Associative

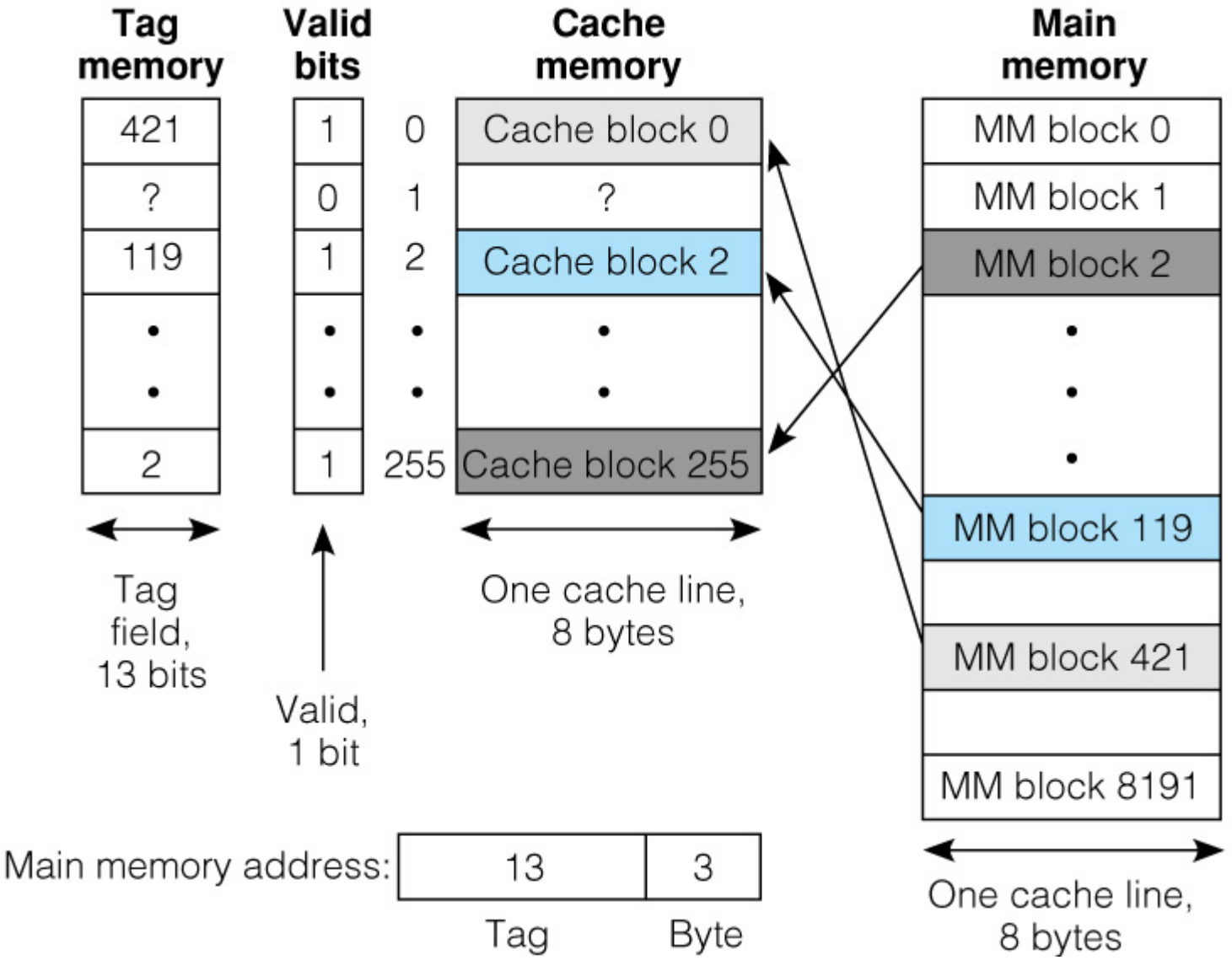
Associative Cache

- Cache views main memory as a list of blocks
- Blocks from main memory may be placed anywhere in cache
- Need tags to identify original main memory block location
- We have 8192 = 2^{13} main memory blocks
  - Need 13 bits for tag field
  - Other 3 bits identify byte in block (8 bytes/block)

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

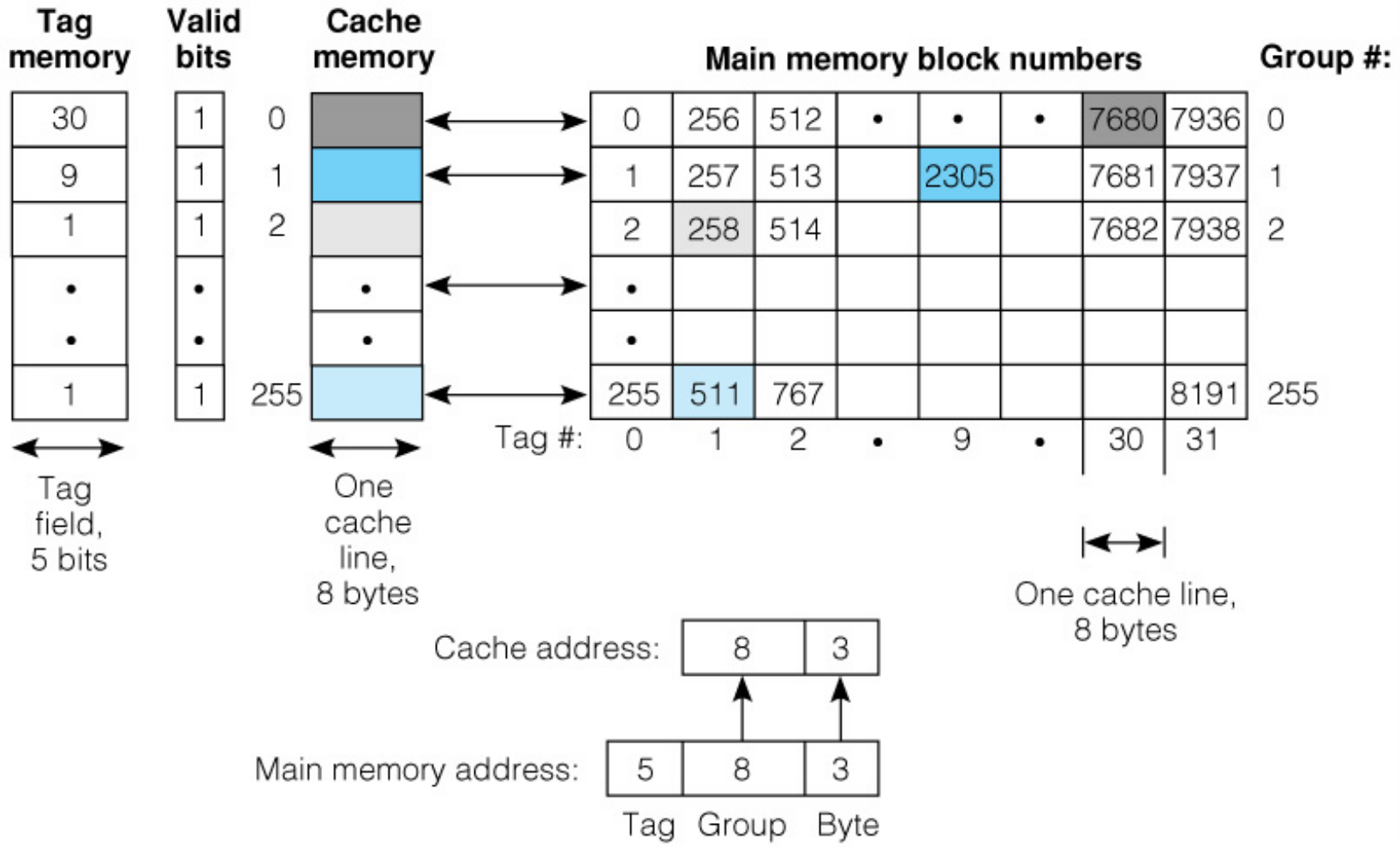
Associative mapped caches



Direct-mapped Cache

- Cache views main memory as an array of blocks
- Blocks from main memory are assigned to specific cache lines, based on array row
  - Number of cache lines = number of rows
- Group field identifies array row
- Tag field identifies array column

Direct mapped cache



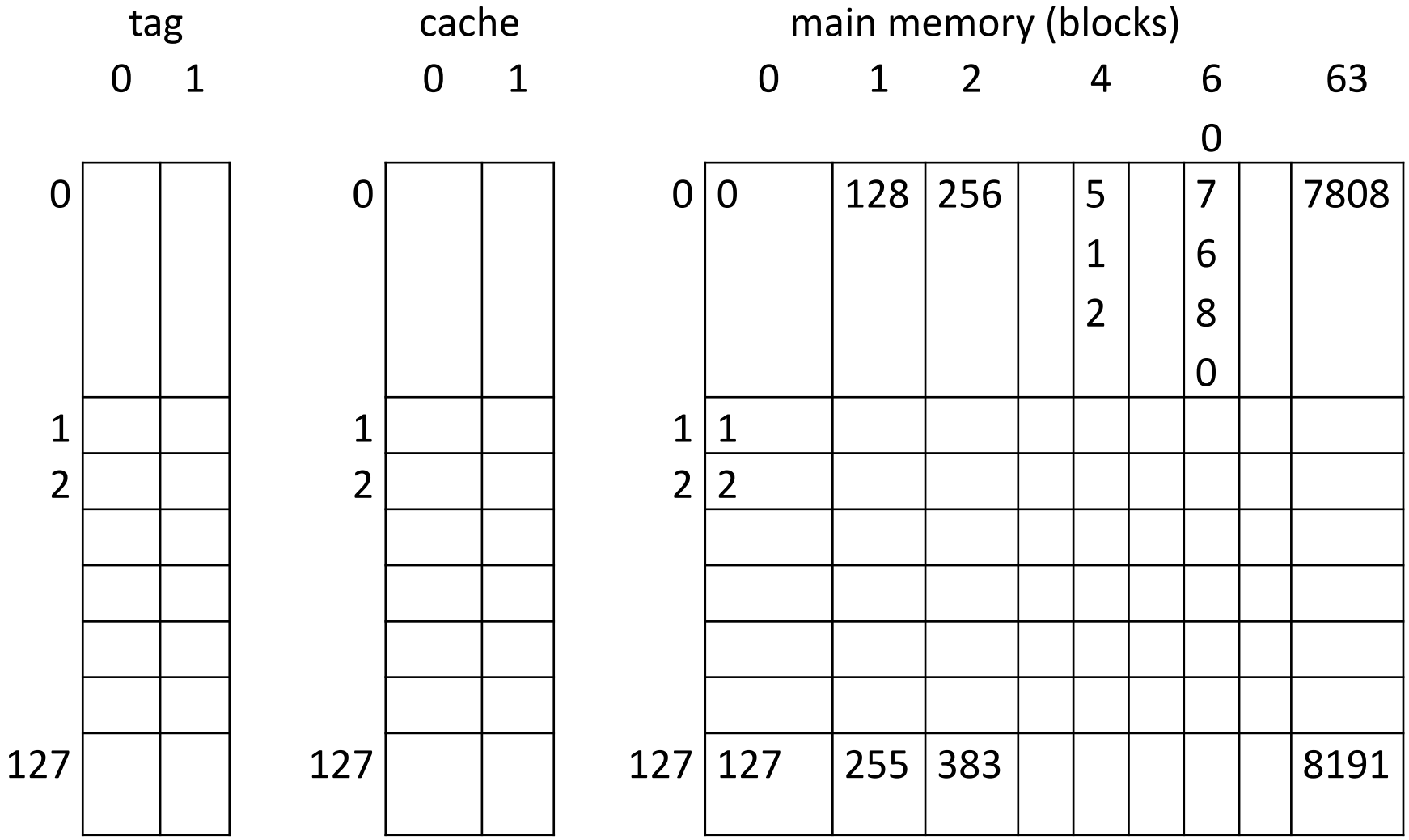
Set-Associative Cache

- Cache views main memory as an array of blocks, similar to direct-mapped cache
- But, now cache is also arranged as an array of blocks
- Set field identifies main memory array row
- Tag field identifies main memory array column

N-way Set-Associative Cache

- N = 2, 4, 8 … (power of 2)
- Number of sets (main memory rows) = number of cache lines / N
  - If N = 1, we have a direct-mapped cache
- For our example, if N=2, the 256-line cache becomes a 128 x 2 cache array

2-Way Set Associative Cache

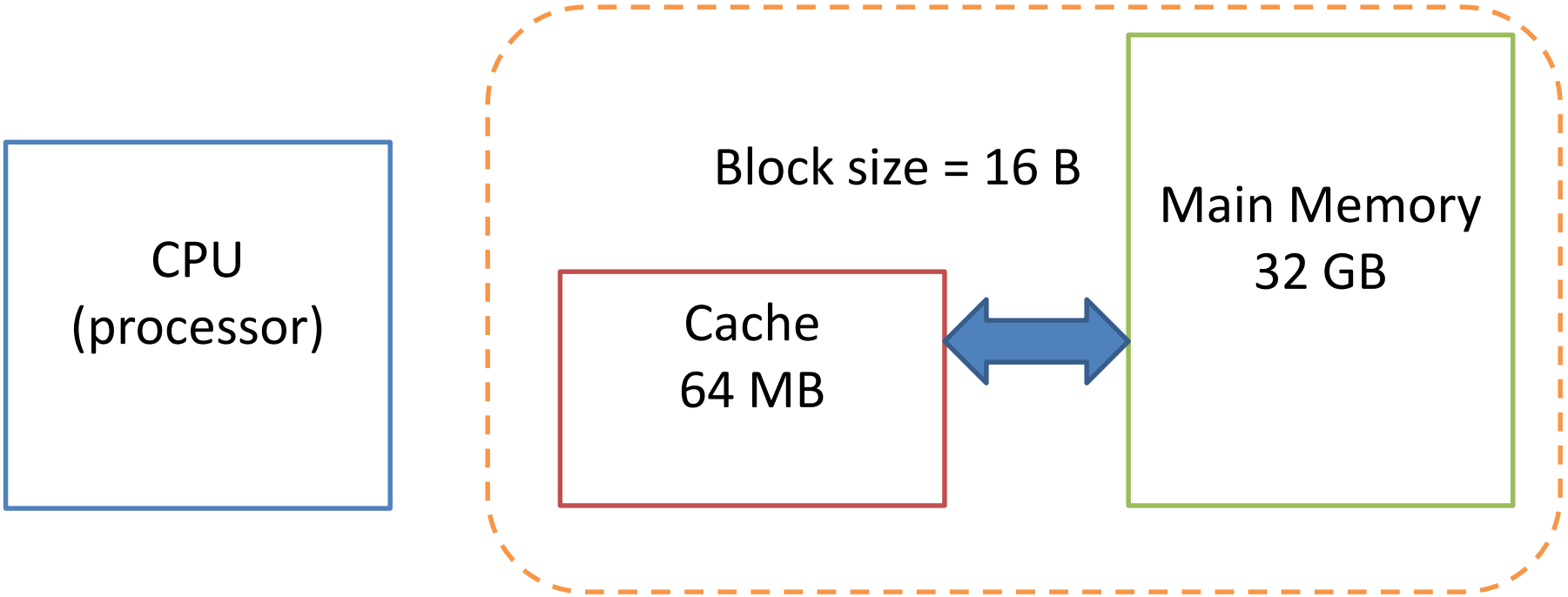


Cache Fields

- Fields are Tag, Set, Byte
- Byte field is 3 bits as before (8 bytes/block)
- Set field identifies main memory row
  - 128 rows = 2^7, so 7 bits
- Tag field identifies main memory column
  - 64 columns = 2^6, so 6 bits

Example B.1-2

- We have the following configuration for our memory system:



Example B.1-2 (cont.)

a) What is the size of the memory address issued by the CPU (number of bits)?

b) Show the fields in the memory address for an **associative** cache.

c) Show the fields in the memory address for a **direct-mapped** cache.

d) Show the fields in the memory address for a **4-way set-associative** cache.

Outline

- B.1 Introduction
- B.2 Cache Performance
- B.3 Basic Cache Optimizations
- B.4 Virtual Memory

Cache Measures

- *Hit rate*: fraction found in that level
  - So high that usually talk about *Miss rate*
- Average memory-access time
  = Hit time + Miss rate x Miss penalty
  (ns or clocks)

Cache Measures (cont)

- *Miss penalty*: time to replace a block from lower level, including time to replace in cache
  - *access time*: time to lower level

    = f(latency to lower level)
  - *transfer time*: time to transfer block

    =f(BW between upper & lower levels)

Memory Access Time

- Bandwidth (BW) (throughput) – transmission rate of bits/bytes (data)
  - Common units: bits/sec (bps)
- Latency (response time) – time (sec) required to access (find) data in specified location
- Access time: latency + (block size)/bandwidth

Cache Performance Metrics

In general:

CPU Execution Time =

(CPU clock cycles) (clock cycle time)

Now, what if we have stalls due to memory access latency (e.g., load, store)? These add cycles, so:

CPU Execution Time =

(CPU clock cycles + memory stall cycles)

(clock cycle time)

Memory Stall Cycles

Memory stall cycles =

(number of cache misses)(miss penalty)

Number of cache misses =

(instruction count)(misses/instruction)

Misses/instruction =

(memory accesses/instruction)(miss rate)

Memory stall cycles = (instruction count) (memory accesses/instruction)(miss rate)(miss penalty)

Instructions Affected by Memory Stalls

- Memory stalls due to cache hits and misses
- Only instructions that access main memory are affected
- MIPS Datapath
  - Instruction memory has I-cache
    - Accessed for EVERY instruction fetch
  - Data memory has D-cache
    - Accessed ONLY during load and store operations

Ideal Cache

Ideal cache: hit rate = 1.00 (100%)

Miss rate = 1 – hit rate = 0

Memory stall cycles = 0

CPU execution time =

(CPU clock cycles)(clock cycle time)

= (IC)(CPI)(clock cycle time)

Real Cache

Real cache: hit rate = h, where $0 < h < 1$

Miss rate = $1 - h$

Memory stall cycles = (instruction count) (memory accesses/instruction)(miss rate)(miss penalty)

CPU execution time = [(IC)(CPI) + memory stall cycles](clock cycle time)

Example B.2-1

We are given a memory system with separate caches for instructions (I-cache) and data (D-cache). The system has the following specifications:

- Base CPI = 2
- Load and stores are 20% of instructions
- I-cache miss rate = 4%
- D-cache miss rate = 2%
- Miss penalty = 100 cycles

What is the **actual CPI** for this system if we consider the effects of cache misses?

4 Questions for Memory Hierarchy

- Q1: Where can a block be placed in the upper level (cache)? *(Block placement)*
- Q2: How is a block found if it is in the upper level? *(Block identification)*
- Q3: Which block should be replaced on a miss? *(Block replacement)*
- Q4: What happens on a write? *(Write strategy)*