《计算机系统结构》课程直播

2020. 3.17

请将ZOOM名称改为“姓名”；

听不到声音请及时调试声音设备；签到将在课间休息进行

本次讲课:

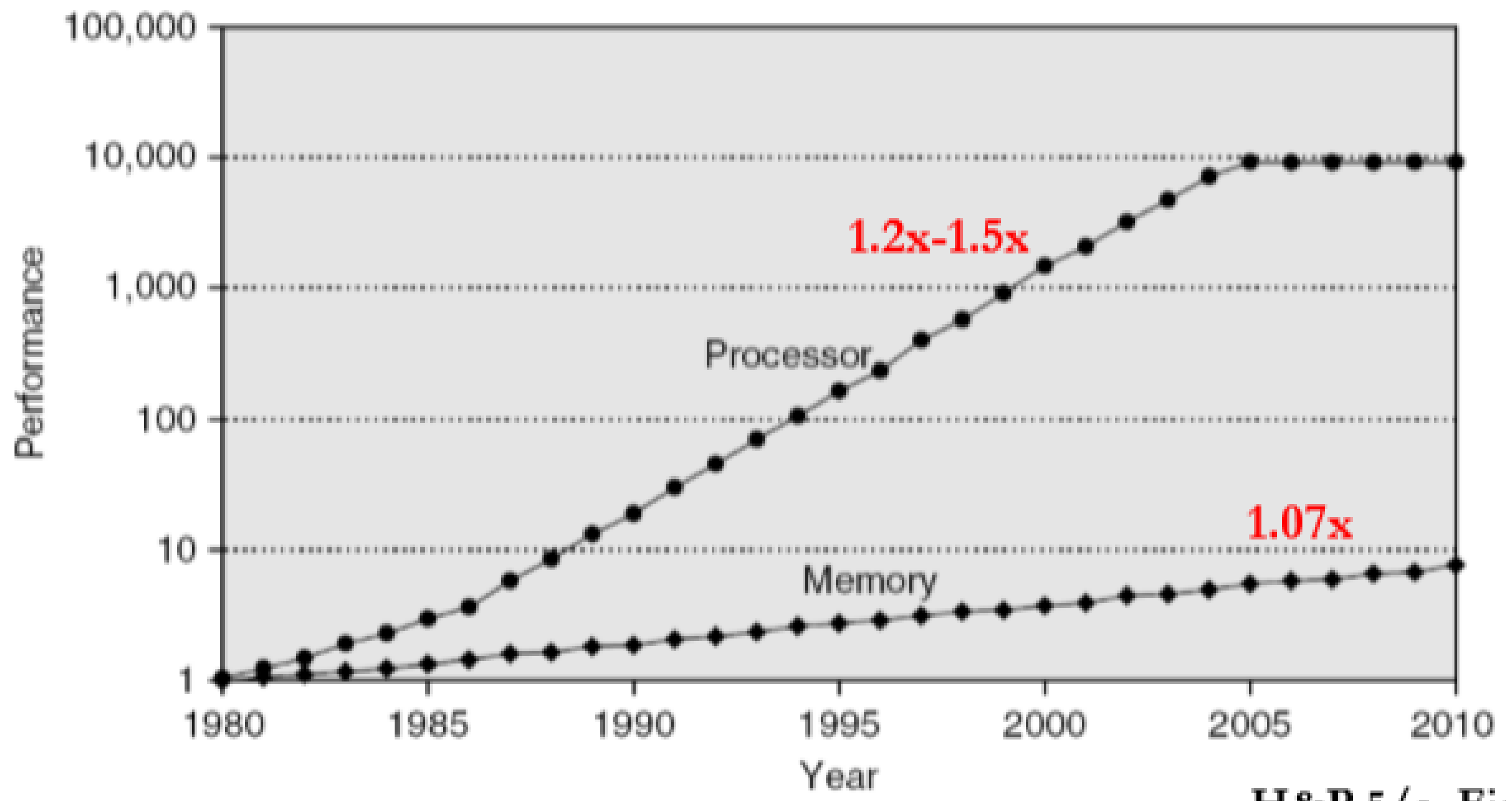
1 Memory Hierarchy Issues

2 Discussion

Cache

Memory Hierarchy Issues

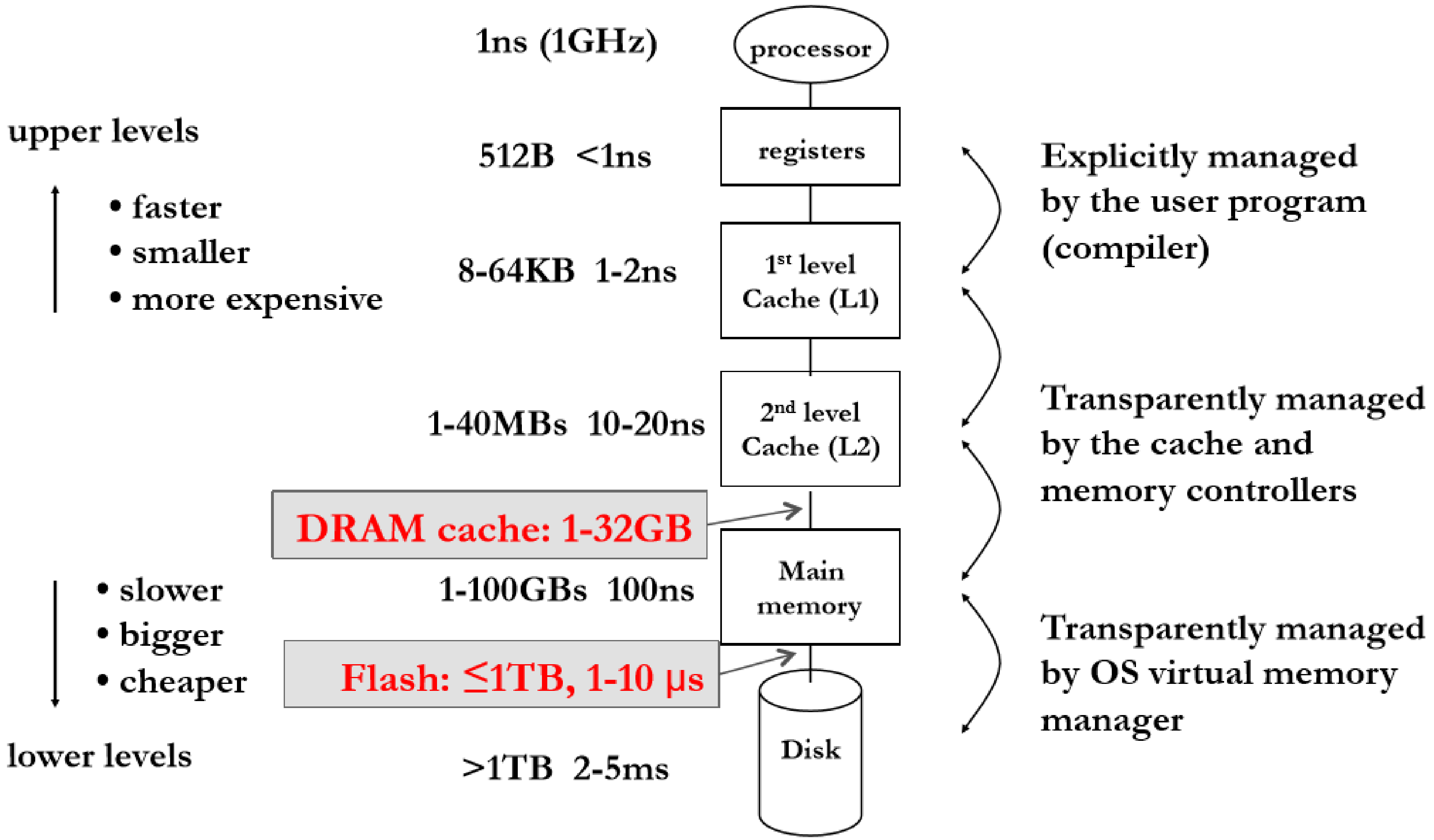
Processor-Memory Performance Gap



H&P 5/e, Fig. 2.2

Memory subsystem design increasingly important!

Memory Hierarchy



Why Does Memory Hierarchy Work?

- Temporal Locality (Locality in Time): Recently-accessed data elements tend to be accessed again.
  ⇒ Keep most recently accessed data items closer to the processor

- Spatial Locality (Locality in Space): Accesses tend to cluster
  ⇒ Move blocks consisting of contiguous words to the upper levels
- Sequential locality - Instructions tend to be accessed sequentially.

- Why does locality exist in programs?
  - Instruction reuse: loops, functions
  - Data working sets: arrays, temporary variables, objects

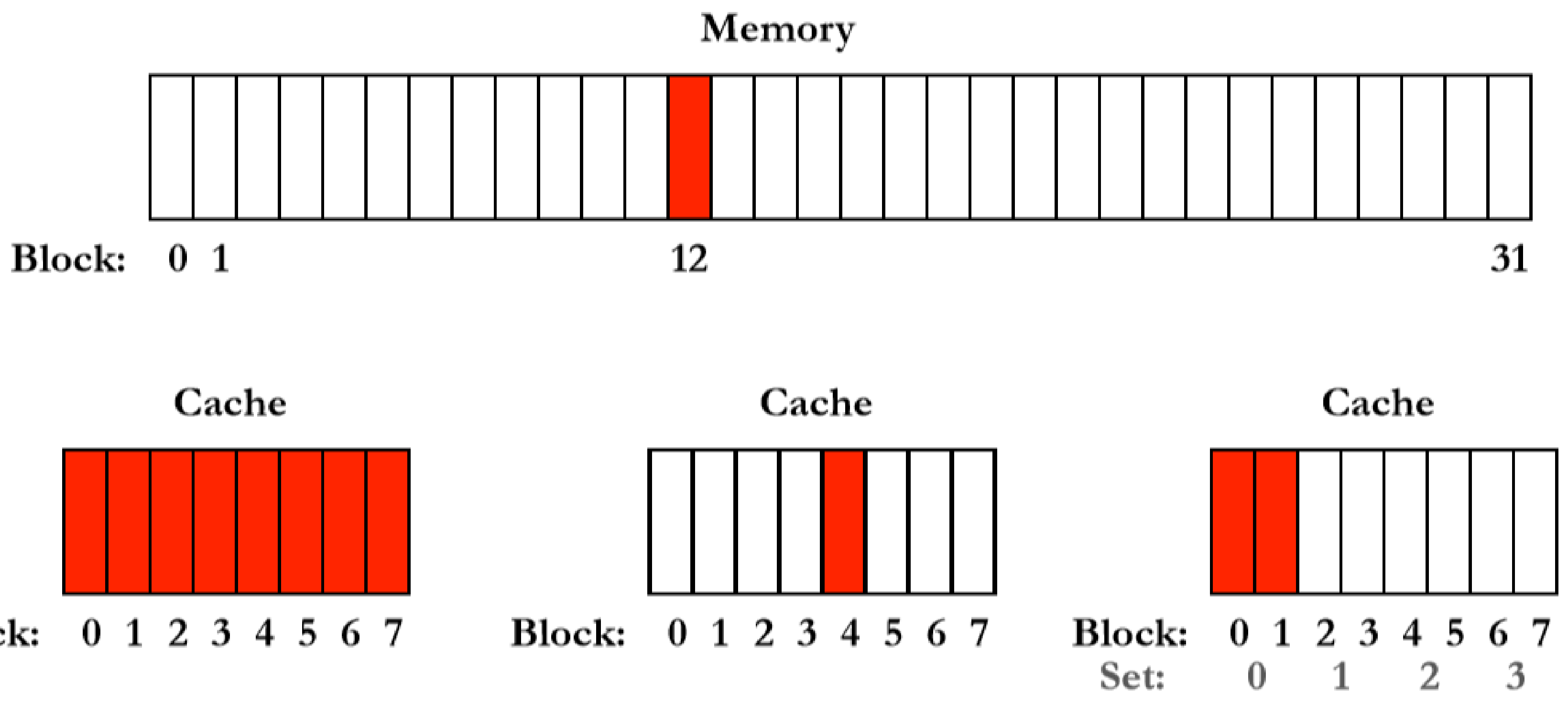
How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler (programmer)
- **cache ↔ main memory**
  - **by the cache controller hardware**
- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB)
  - by the programmer (files)

Memory Hierarchy Issues

- Block size: E.g., 64B for cache lines, 4KB for memory pages
- Block placement: Where can a block be placed?
  - E.g., direct mapped (exactly 1 location), fully associative (any location)
- Block identification: How can a block be identified?
  - E.g., hardware tag matching (e.g., cache), OS page table
- Block replacement: Which block should be replaced?
  - E.g., Random, Least recently used (LRU), Not recently used (NRU)
- Write strategy: What happens on a write?
  - E.g., write-through, write-back, write-allocate
- Inclusivity: whether next lower level contains all the data found in the current level
  - Inclusive, exclusive

Cache Block Placement



Fully associative:

block 12 can go anywhere in the cache

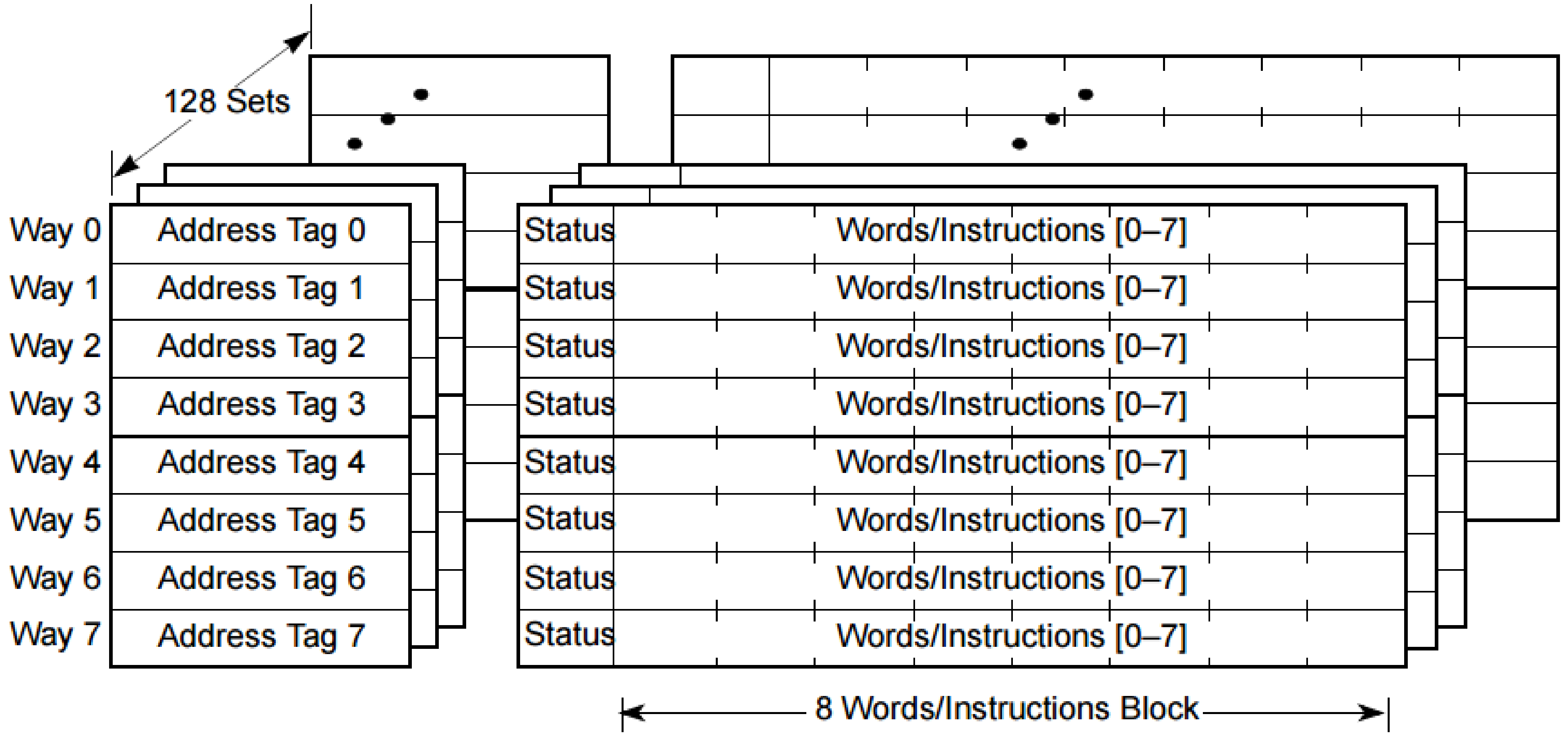
Direct mapped:

block 12 can only go into block 4 (12 mod 8)

Set associative:

block 12 can go anywhere in set 0 (12 mod 4)

How is a Block Found



L1 Cache Organization

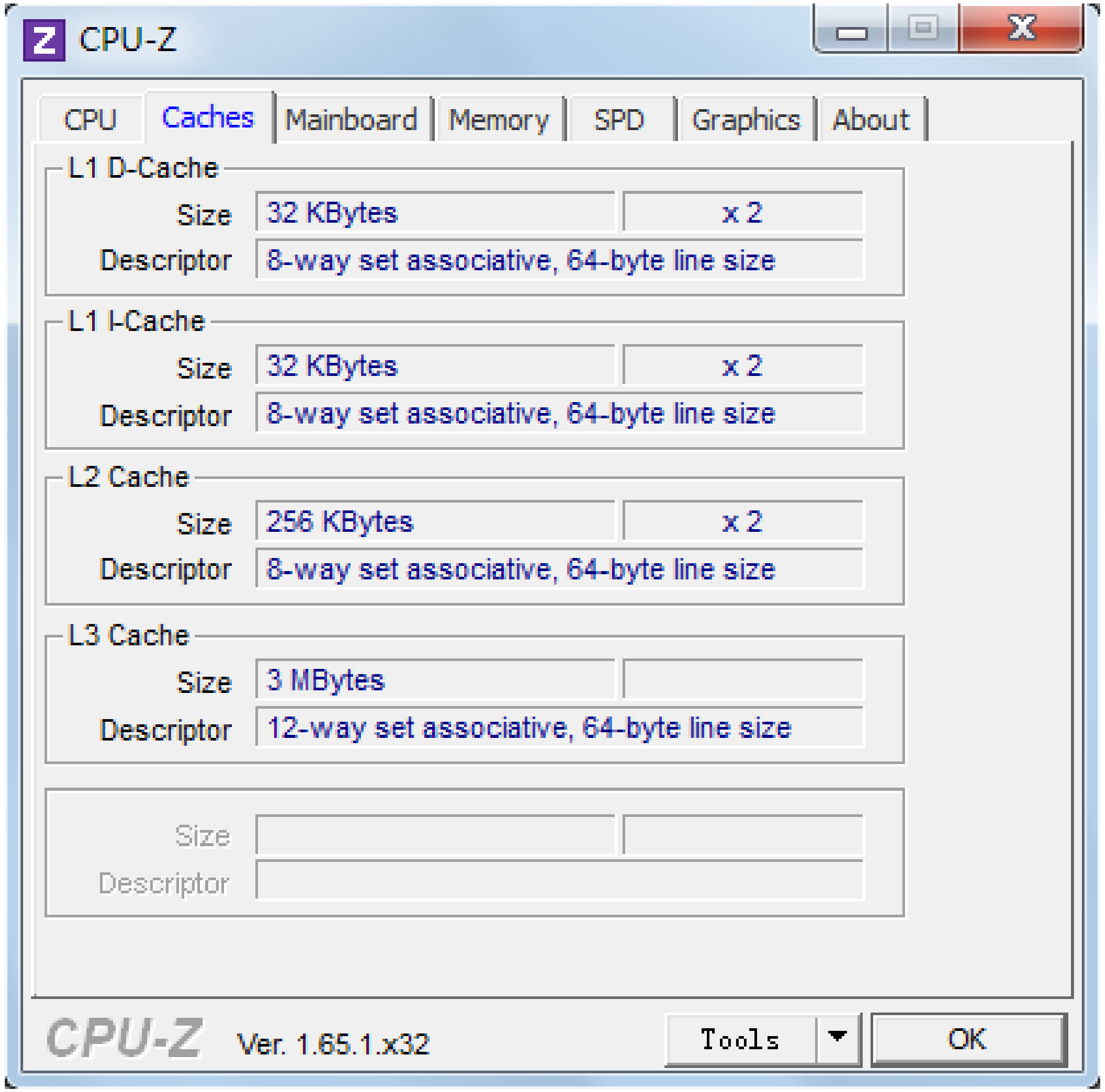
Logical Organization of Cache

- The cache is said to be **direct mapped,** if each block has only one place it can appear in the cache
  - e.g., each cache line is a “set”
- The cache is said to be **fully associative,** if a block can be placed anywhere in the cache
  - e.g., all the cache lines belong to one “set”
- The cache is said to be **set associative**, if a block can be placed in a restricted set of places in the cache
  - e.g., 4-way set associative: 4 lines per set

Summary of Cache Associativity

- Fully associative
  - The block from the lower level can go into any location in the cache
  - Most flexible approach → lowest miss rate
  - Must search the whole cache to find the block
    → increased access time and high power consumption
- Direct mapped
  - The block from the lower level can only go into one location in the cache
  - Simplest approach to implement
  - Blocks mapping to the same location → increased miss rates
- Set associative
  - Split the cache into groups of *m* blocks (sets) → m-way set-associative
  - The block from the lower level can only go into one set, but within that set it can go anywhere
  - What's a good degree of associativity?
    - Higher level caches: 2- or 4-way common
    - Lower level caches: 8- to 32-way common

单击此处编辑母版标题样式



L1 D-Cache
Size: 32 KBytes x 2
Descriptor: 8-way set associative, 64-byte line size

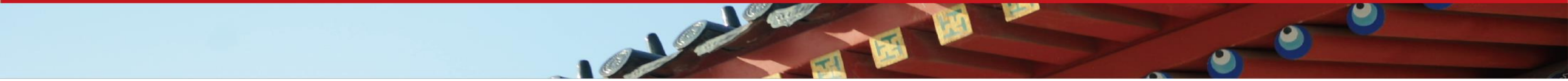
L1 I-Cache
Size: 32 KBytes x 2
Descriptor: 8-way set associative, 64-byte line size

L2 Cache
Size: 256 KBytes x 2
Descriptor: 8-way set associative, 64-byte line size

L3 Cache
Size: 3 MBytes
Descriptor: 12-way set associative, 64-byte line size

Size
Descriptor

CPU-Z Ver. 1.65.1.x32



Cache Block Identification

- Every block is identified by a name or tag, which is part of the memory address
- Block tag is stored alongside the block data in the cache
- Block tags in the cache are compared with the tag of the requested block → often in parallel for set-associative caches, for speed
- Block tag usually comprises the high-order bits of the memory address

Full memory address:

Tag	Index	Block offset

- Data address: the address of the byte being referenced → 32 bits for MIPS
- Offset: the byte within the block; e.g., 6 bits for a 64B block
- Index: the set where the block can be found; e.g., 8 bits for a 4-way 64KB cache
- Tag: the “ID” of the block; e.g., 32-8-6=18 bits

Address Mapping Example

- Cache: 32 KBytes, 2-way, 64 byte lines
- Address: 32 bits

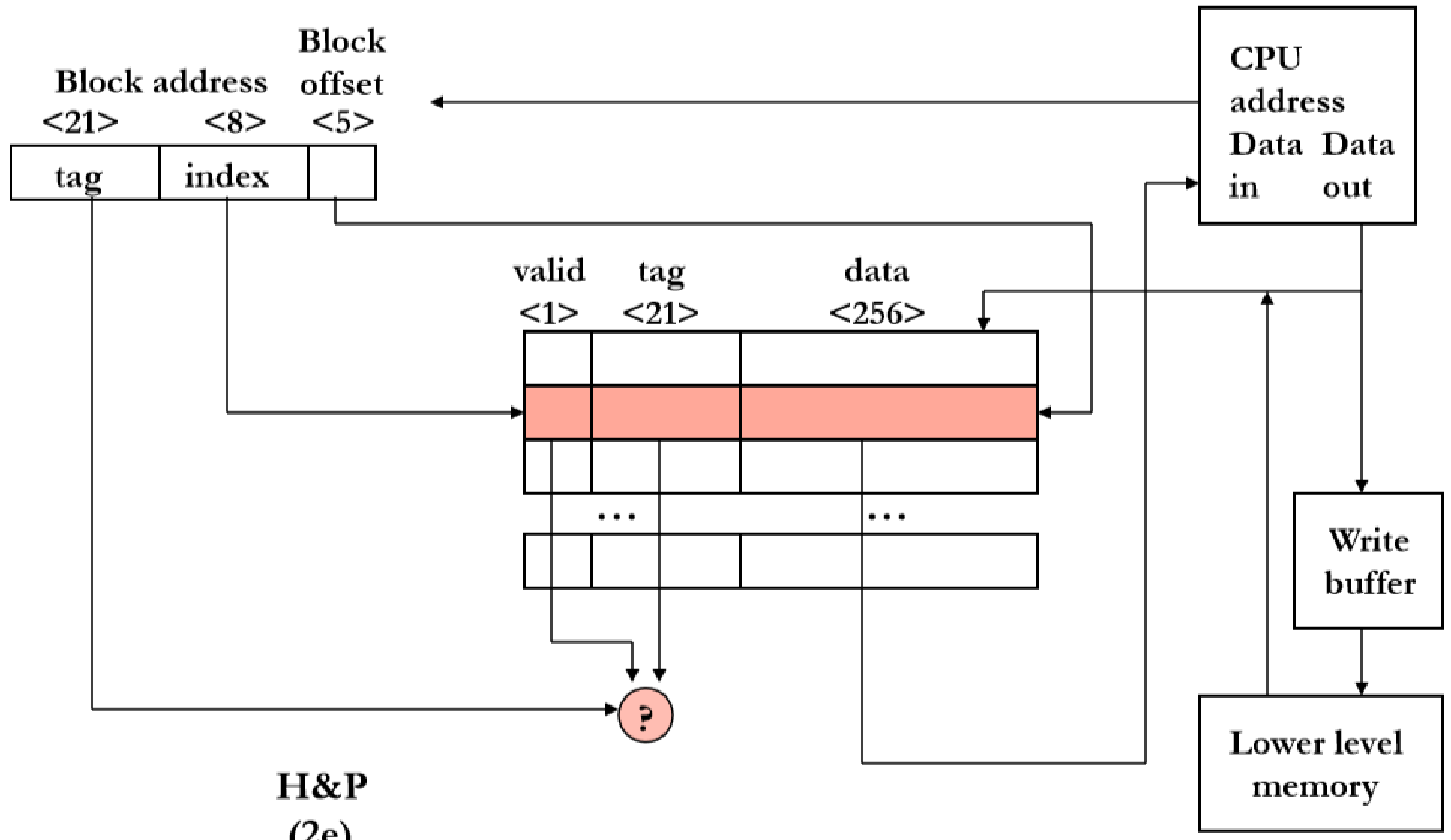
- Example: 0x000249F0 = $(0000\ 0000\ 0000\ 0010\ 0100\ 1001\ 1111\ 0000)_2$

Address Mapping Example

- Cache: 32 KBytes, 2-way, 64 byte lines
- Address: 32 bits

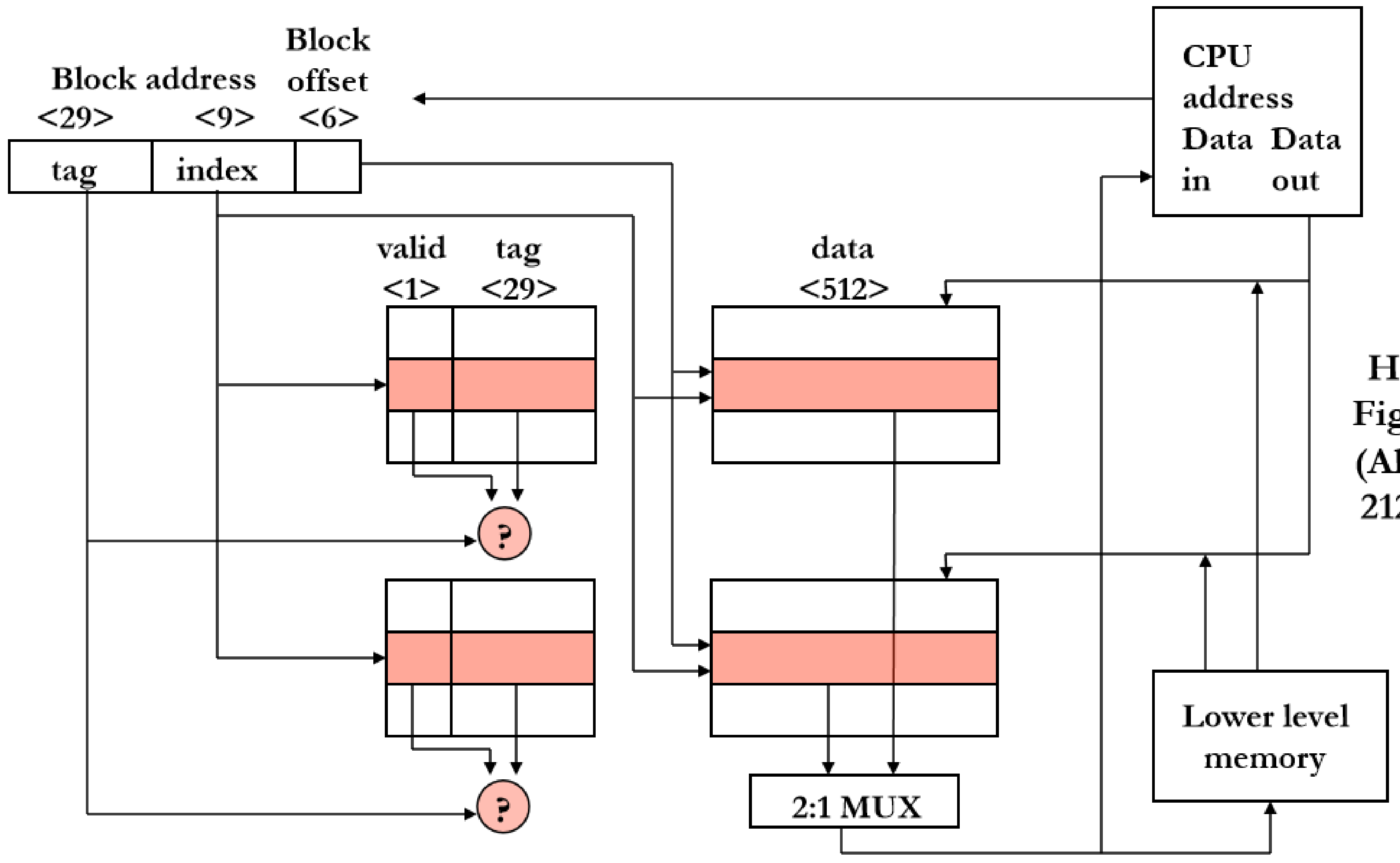
- Example: 0x000249F0 = $(\underbrace{0000\ 0000\ 0000\ 0010\ 01}_{\text{tag}}\underbrace{00\ 1001\ 11}_{\text{index}}\underbrace{11\ 0000}_{\text{offset}})_2$
  - Byte offset
    64 bytes → 6 bits ⇒ 11 0000
  - Index: 32K/64 = 512 lines in the cache
    512/2 = 256 sets in the cache
    256 sets → 8 bits ⇒ 00 1001 11 = 39
  - Tag:
    0000 0000 0000 0010 01

Cache Organization: Direct mapped



H&P (2e) Fig. 5.5 (Alpha 21064)

Cache Organization: 2-way set associative



H&P Fig. 5.7 (Alpha 21264)

Cache Block Replacement

- To bring a new block in the cache, another block must be evicted
- Direct mapped caches: there is only one choice of block to evict
- Associative caches: how to choose a "victim"?
  - Random: select a victim block in the set randomly
  - Least-recently-used (LRU): select the block that has not been used for the longest period of time
    → works well in practice because of the principle of locality
  - Not-recently-used (NRU): select a block other than the most-recently used block.
    - Need less storage than LRU and performs better than random
  - Ideal: select the block that will not be used for the longest period of time
    → requires knowledge of the future → unrealistic!

Cache Write Strategies

How are the writes handled (i.e, when do stores reach a lower level of the memory hierarchy)?

- **Write through**: write to lower level as cache is modified
  - Writes will perform at the speed of the lower level of memory hierarchy ☹
  - Generates more traffic ☹
  - Lower level is kept **coherent** with higher level (particularly important for multi-processors) ☺
- **Write back**: only write to lower level when the block is evicted
  - Writes will perform at the speed of the higher level ☺
  - Generates less traffic ☺
  - Lower level can have stale data for some time (**cache-coherency** problem) ☹

Cache Write Strategies

What happens if the block is not found in the cache?

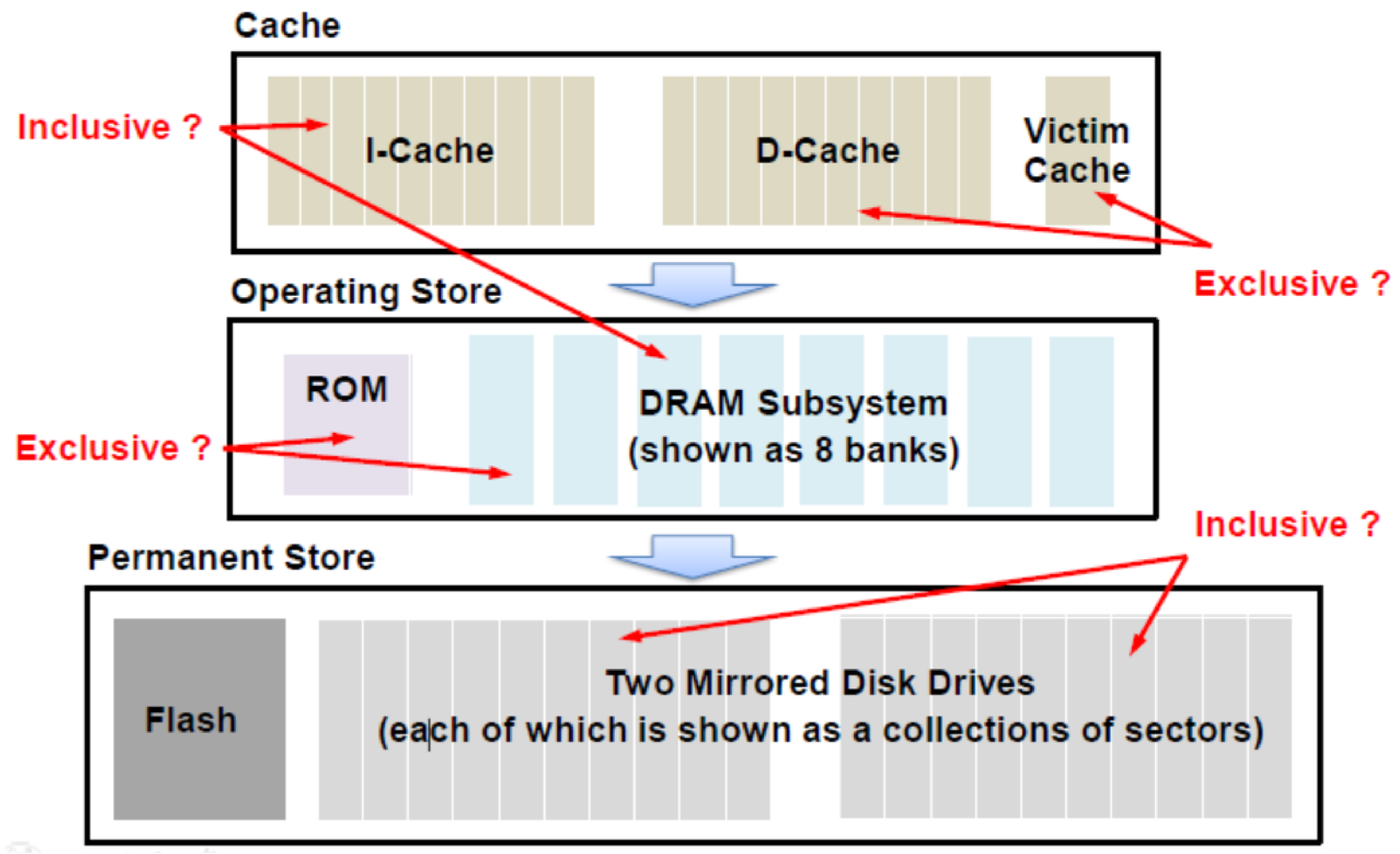
- **Write allocate**: bring the block into the cache and write to it
  - Good if block will soon be used by another memory access (locality)
  - Usually used with write back
- **Write no-allocate**: do not bring block into cache and modify data in the lower level
  - Good if no memory access to the same block occur in the near future
  - Usually used with write through

Multi-Level Caches

Do lower-level caches keep a copy of the block that’s brought into a higher-level cache?

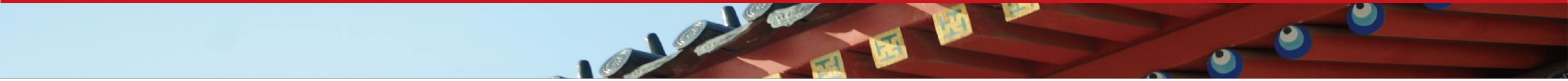
- Inclusive caches:
  - Lower-level cache has a copy of every block in higher-level caches
  - Wastes capacity of lower-level caches ☹
  - Simplifies finding a cache block by another entity (e.g., by other processors) ☺
- Exclusive caches:
  - A block may reside in only one level of the cache hierarchy
  - Maximizes aggregate capacity of the cache hierarchy ☺
  - Requires a uniform block size for all cache levels ☹

A Specific Example of a Memory System

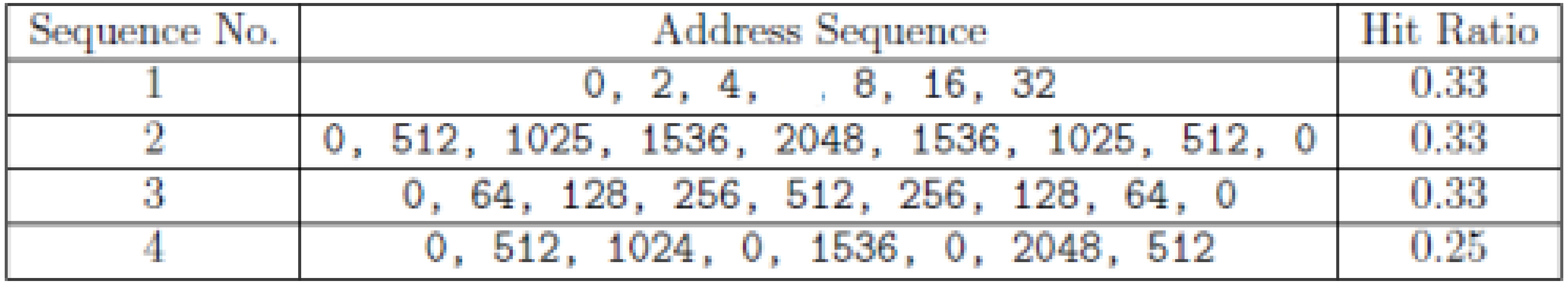


课堂分组讨论

将解除全体静音，请打开你的语音



- We have given you four different sequences of addresses generated by a program running on a processor with a data cache. Cache hit ratio for each sequence is also shown below. Assuming that the cache is initially empty at the beginning of each sequence, find out the following parameters of the processor's data cache:
  - Associativity (1, 2 or 4 ways)
  - Block size (1, 2, 4, 8, 16, or 32 bytes)
  - Total cache size (256 B, or 512 B)
  - Replacement policy (LRU or FIFO)
  - Assumptions: all memory accesses are one byte accesses. All addresses are byte addresses.



Sequence No.	Address Sequence	Hit Ratio
1	0, 2, 4, 8, 16, 32	0.33
2	0, 512, 1025, 1536, 2048, 1536, 1025, 512, 0	0.33
3	0, 64, 128, 256, 512, 256, 128, 64, 0	0.33
4	0, 512, 1024, 0, 1536, 0, 2048, 512	0.25

讨论2：

- We have a 4 Kbyte cache operating with 256 byte lines. Consider two designs for this cache.
- The first is a direct mapped design. The second is a fully associative cache with an LRU replacement policy. Physical addresses are 16 bits.
- Provide a sequence of physical addresses (in hexadecimal notation) that when repeated indefinitely will result in a lower miss rate in the direct mapped cache than in the fully associative cache. The address stream should be accompanied by an explanation of this behavior.

Summary

- Cache Basics
- Concepts
  - Block size: E.g., 64B for cache lines, 4KB for memory pages
  - Block placement: Where can a block be placed?
    - E.g., direct mapped (exactly 1 location), fully associative (any location)
  - Block identification: How can a block be identified?
    - E.g., hardware tag matching (e.g., cache),
  - Block replacement: Which block should be replaced?
    - E.g., Random, Least recently used (LRU), Not recently used (NRU)
  - Write strategy:
    - Write back, write through

下一节

- 周四 8：00
- Cache Performance
  - Optimizations

- 请做好准备

再见