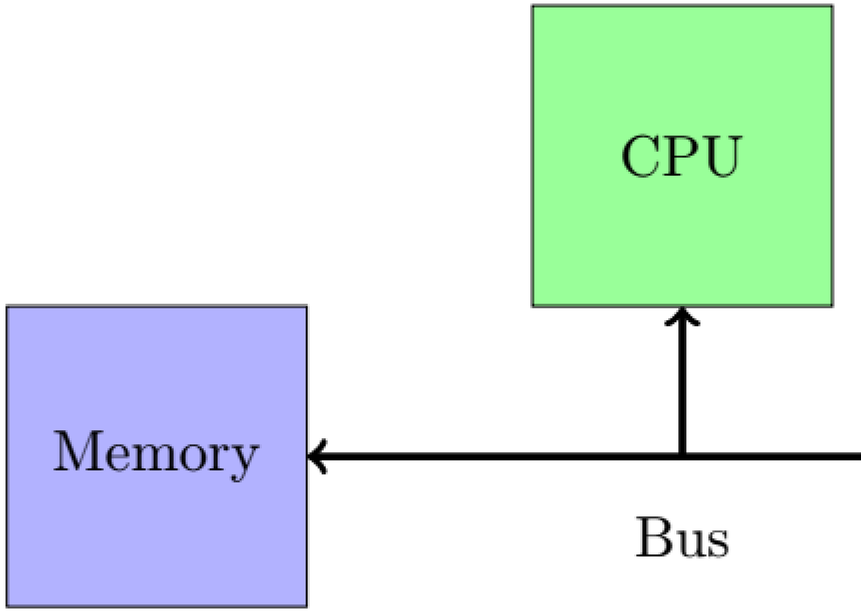
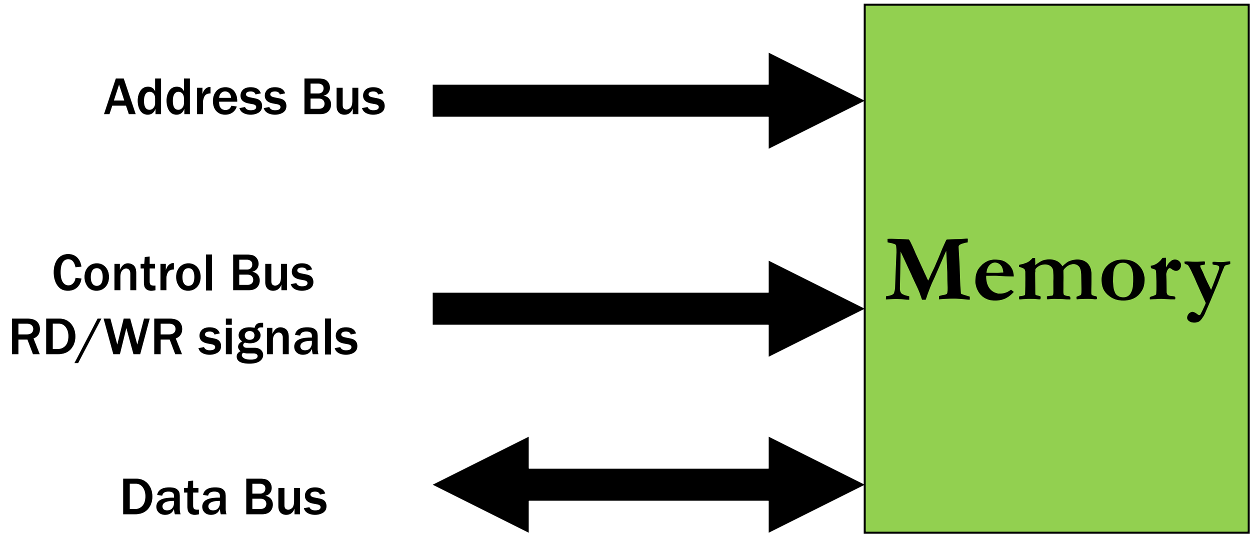
CS2053 Computer Architecture

Memory Hierarchy II

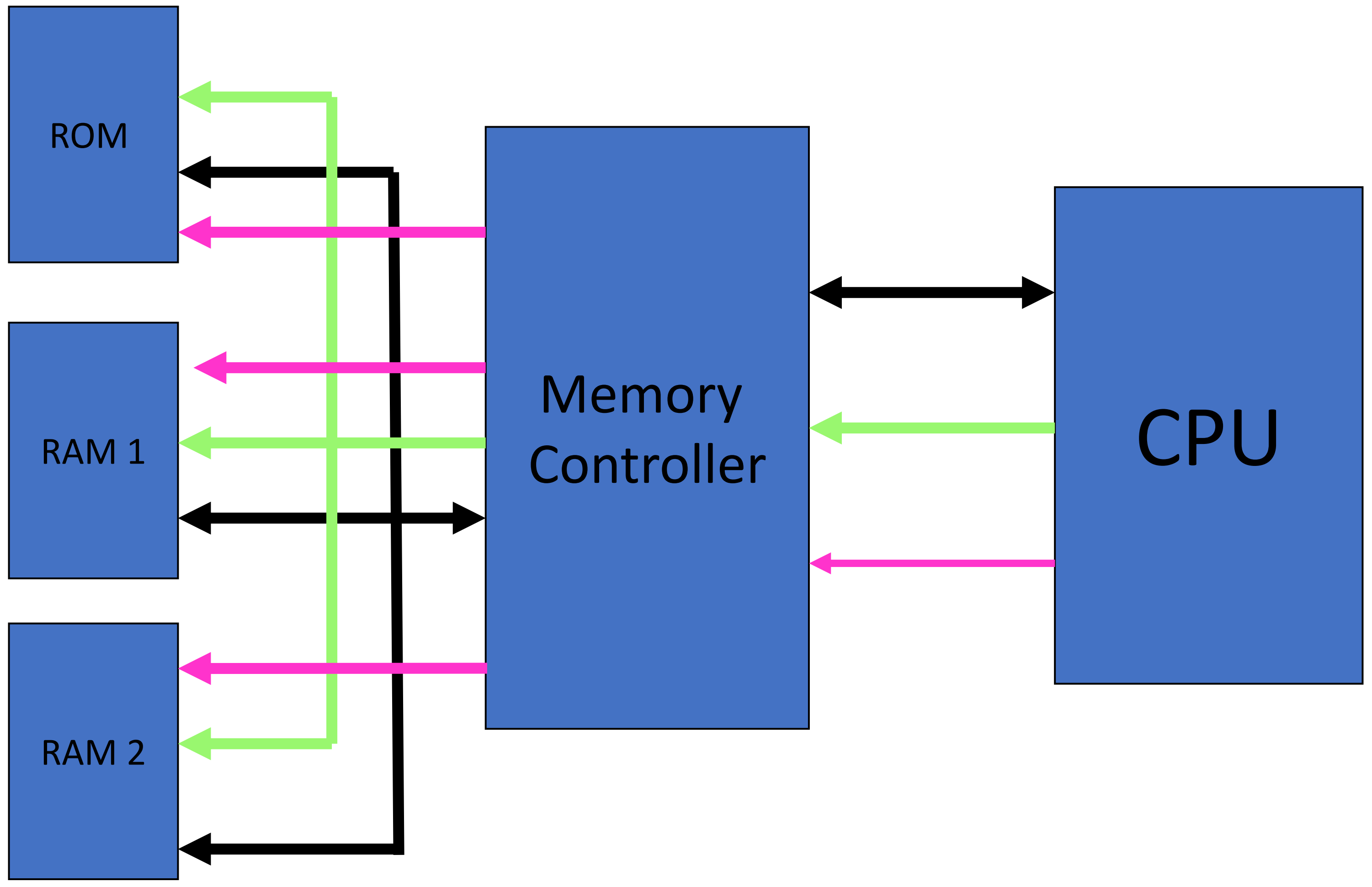
Dr. Sulochana Sooriyaarachchi

Accessing Memory

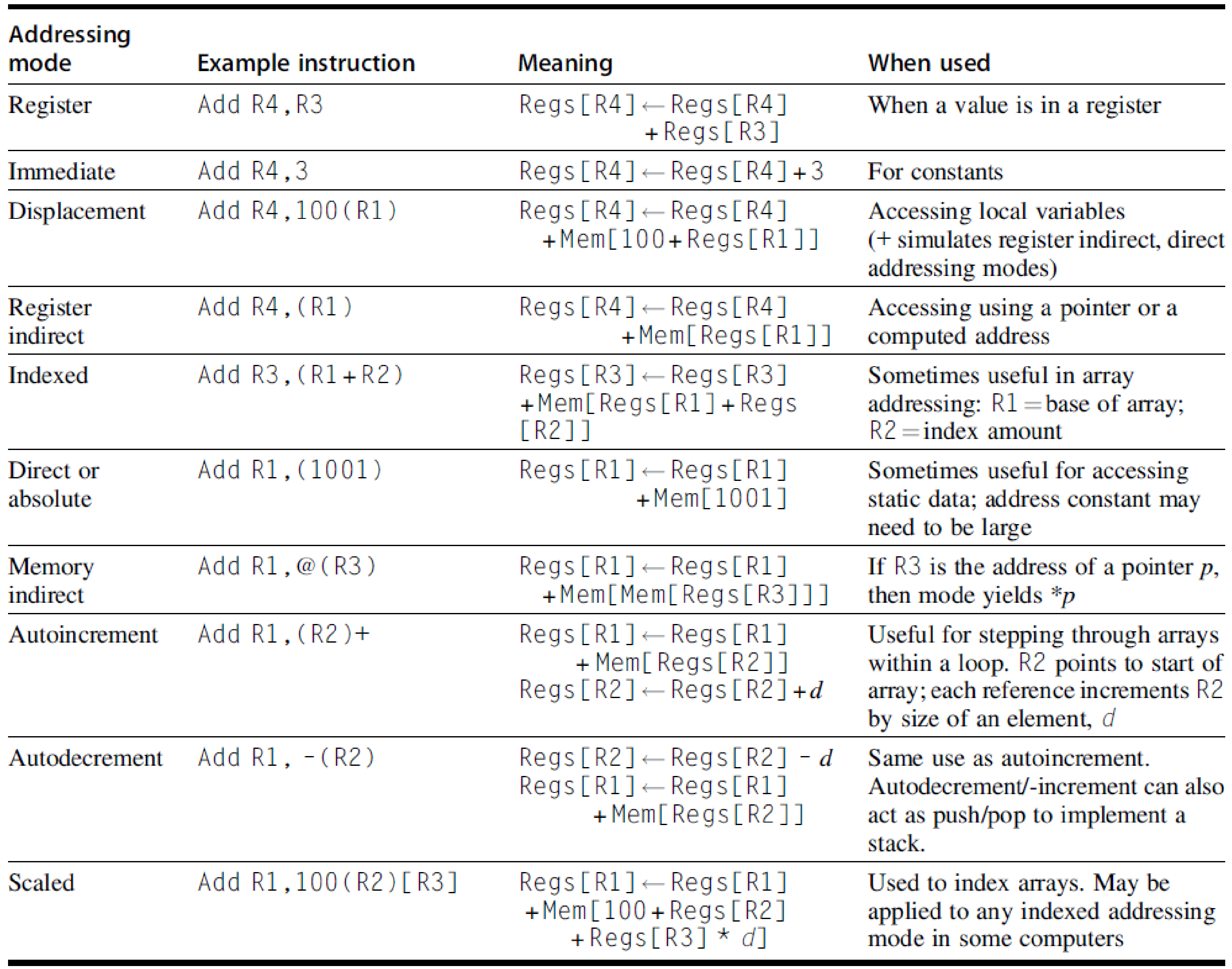




Connecting Memory & CPU



Example Addressing Modes



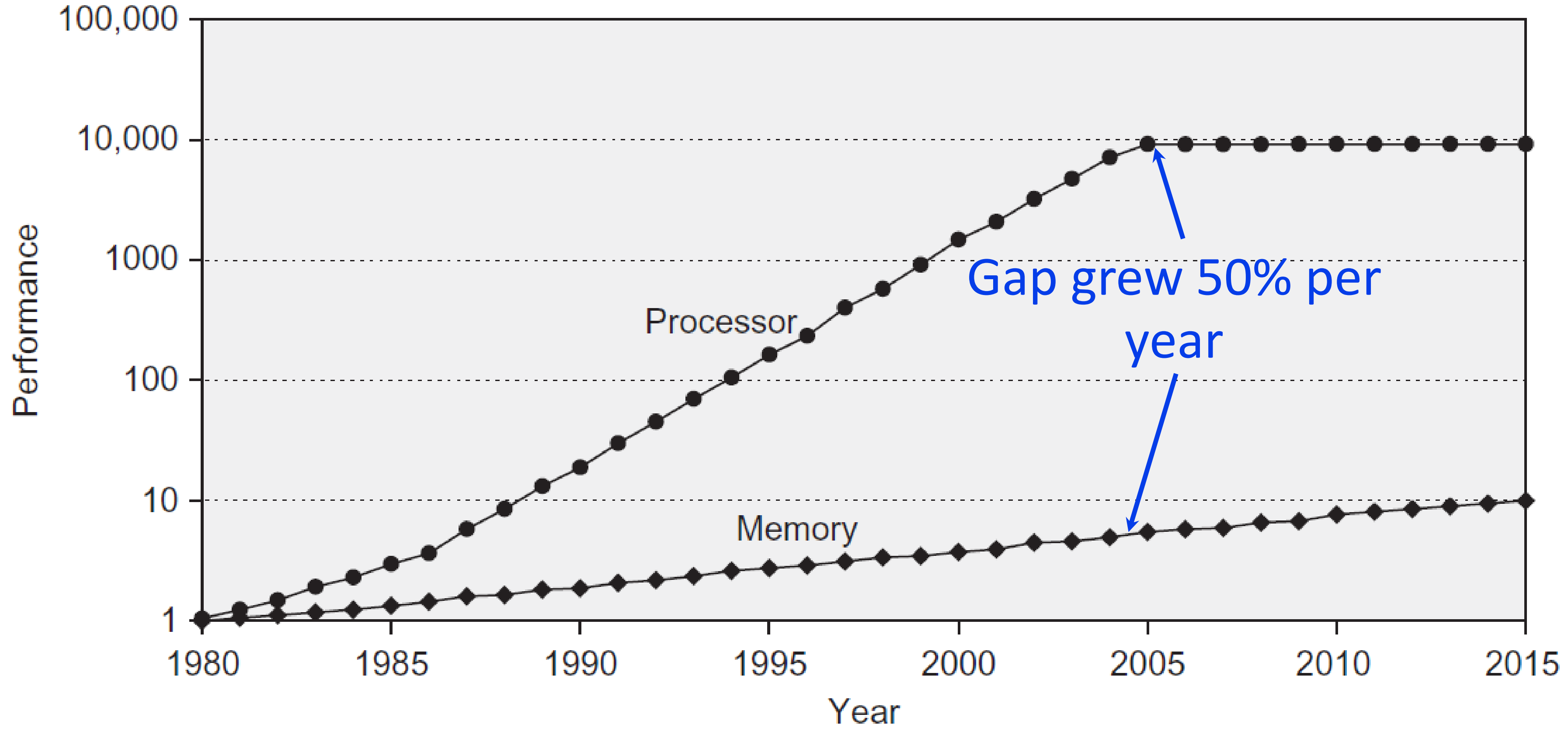
Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4]←Regs[R4] +Regs[R3]	When a value is in a register
Immediate	Add R4,3	Regs[R4]←Regs[R4]+3	For constants
Displacement	Add R4,100(R1)	Regs[R4]←Regs[R4] +Mem[100+Regs[R1]]	Accessing local variables (+ simulates register indirect, direct addressing modes)
Register indirect	Add R4,(R1)	Regs[R4]←Regs[R4] +Mem[Regs[R1]]	Accessing using a pointer or a computed address
Indexed	Add R3,(R1+R2)	Regs[R3]←Regs[R3] +Mem[Regs[R1]+Regs [R2]]	Sometimes useful in array addressing: R1=base of array; R2=index amount
Direct or absolute	Add R1,(1001)	Regs[R1]←Regs[R1] +Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large
Memory indirect	Add R1,@(R3)	Regs[R1]←Regs[R1] +Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer *p*, then mode yields **p*
Autoincrement	Add R1,(R2)+	Regs[R1]←Regs[R1] +Mem[Regs[R2]] Regs[R2]←Regs[R2]+*d*	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, *d*
Autodecrement	Add R1, -(R2)	Regs[R2]←Regs[R2] - *d* Regs[R1]←Regs[R1] +Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/-increment can also act as push/pop to implement a stack.
Scaled	Add R1,100(R2)[R3]	Regs[R1]←Regs[R1] +Mem[100+Regs[R2] +Regs[R3] * *d*]	Used to index arrays. May be applied to any indexed addressing mode in some computers

Take home assignment 1

- Study the addressing modes in the processor architecture in your Lab Series.
  - PC-relative: via *auipc, jal* and *br** instructions
  - Register-offset: via the *jalr, addi* and all memory instructions.
  - Absolute: via the *lui* instruction
- Read relevant sections in: RISCV ISA SpecificationsURL
  - https://online.uom.lk/mod/url/view.php?id=343622
    - Volume 1, Unprivileged Specification version 20191213 [PDF]

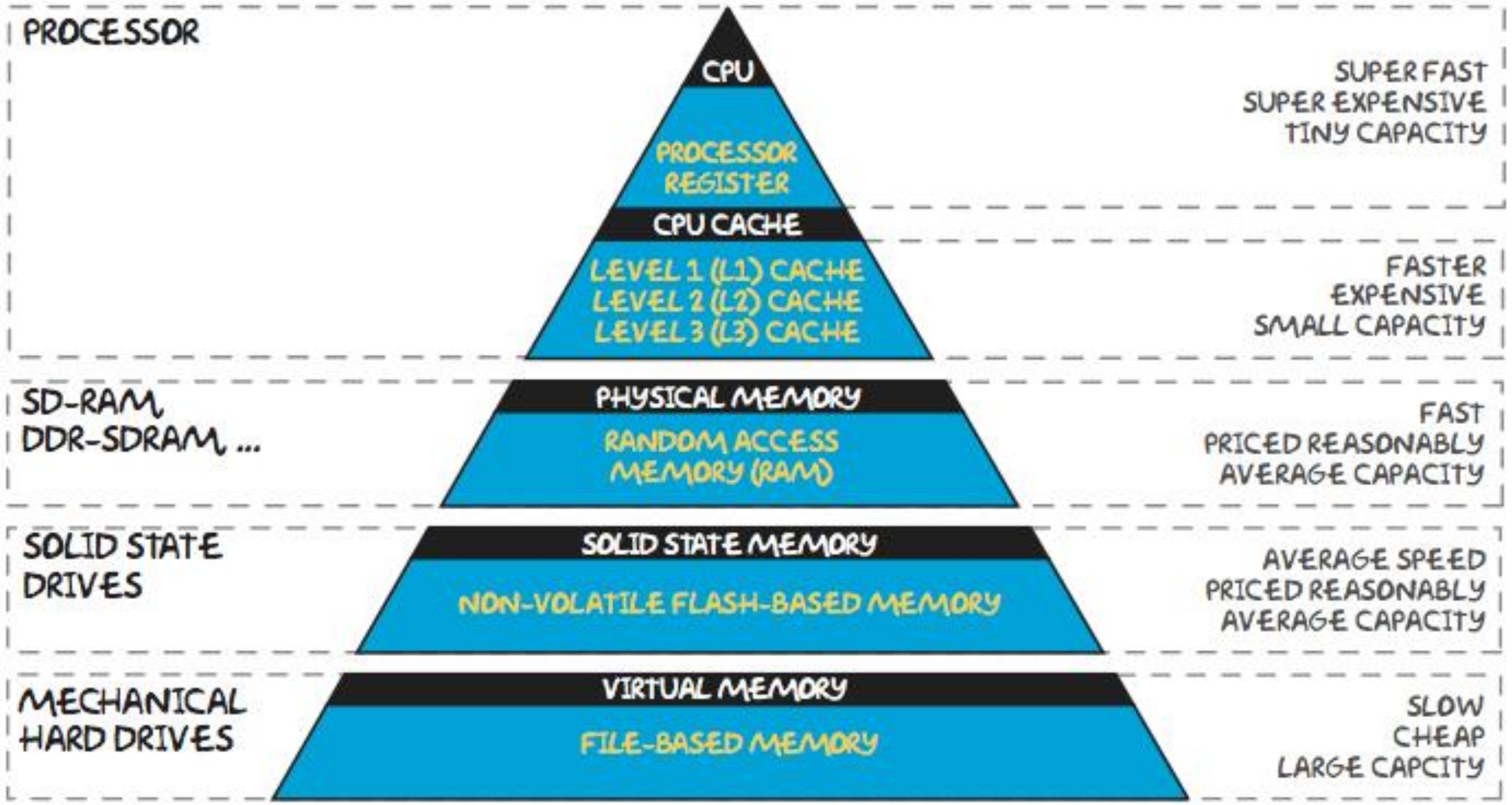
Submit a short note for each of above instructions with an illustration of addressing involved

Processor Memory Gap



Source: Computer Architecture, A Quantitative Approach by John L. Hennessy and David A. Patterson

Modern Memory Hierarchy



Source: http://blog.teachbook.com.au/index.php/2012/02/memory-hierarchy/

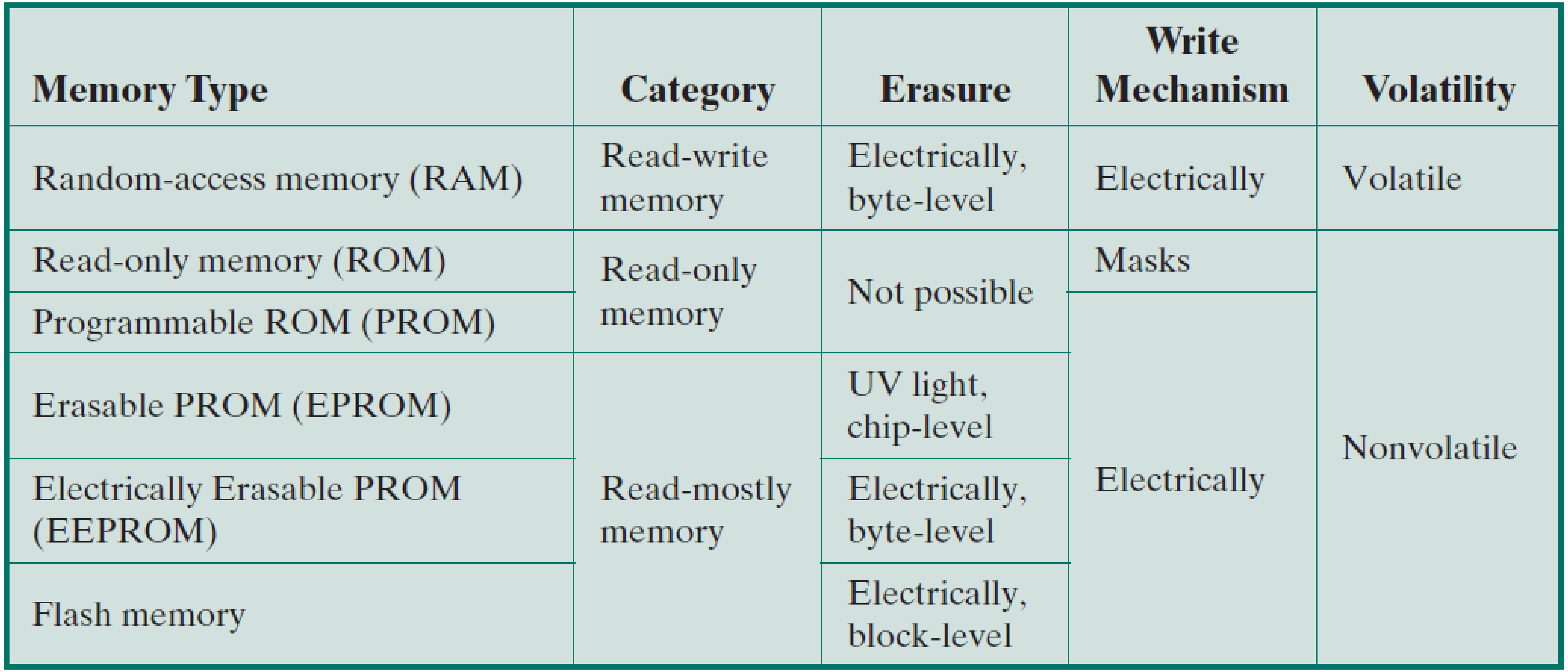
Principle of Locality

- Programs tend to reuse data & instructions that are close to each other or they have used recently
- Temporal locality
  - Recently referenced items are likely to be referenced in the near future
  - A block tend to be accessed again & again
- Spatial locality
  - Items with nearby addresses tend to be referenced close together in time
  - Near by blocks tend to be accessed

Summary

- Memory hierarchy - Illusion of a large amount of fast memory
- All data and instructions in memory are not accessed at once with equal probability
- Principle of Locality – accessing small portion of address space at any instance of time
    - Temporal locality – refer the same item again and again
    - Spatial locality – refer the items stored close to each other

Memory types



Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level		
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

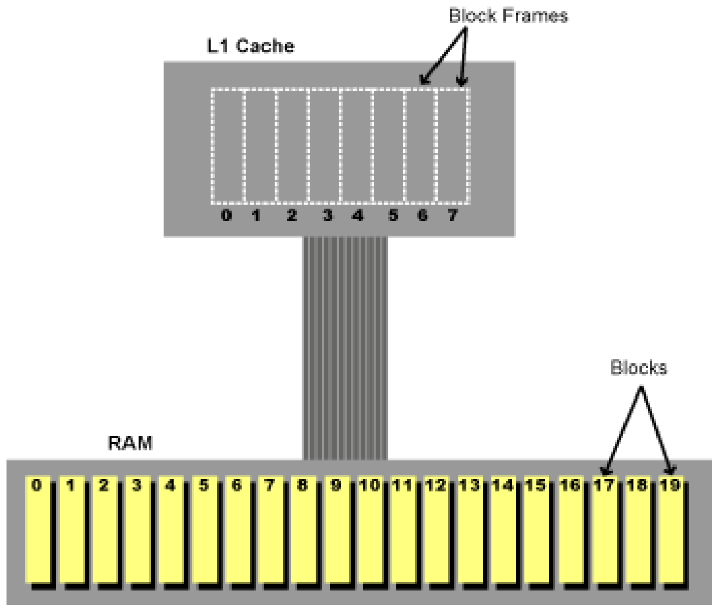
Take home assignment 2

Read Section 5.2 of

Patterson, David. "Computer organization and design RISC-V edition: the hardware." (2017) and

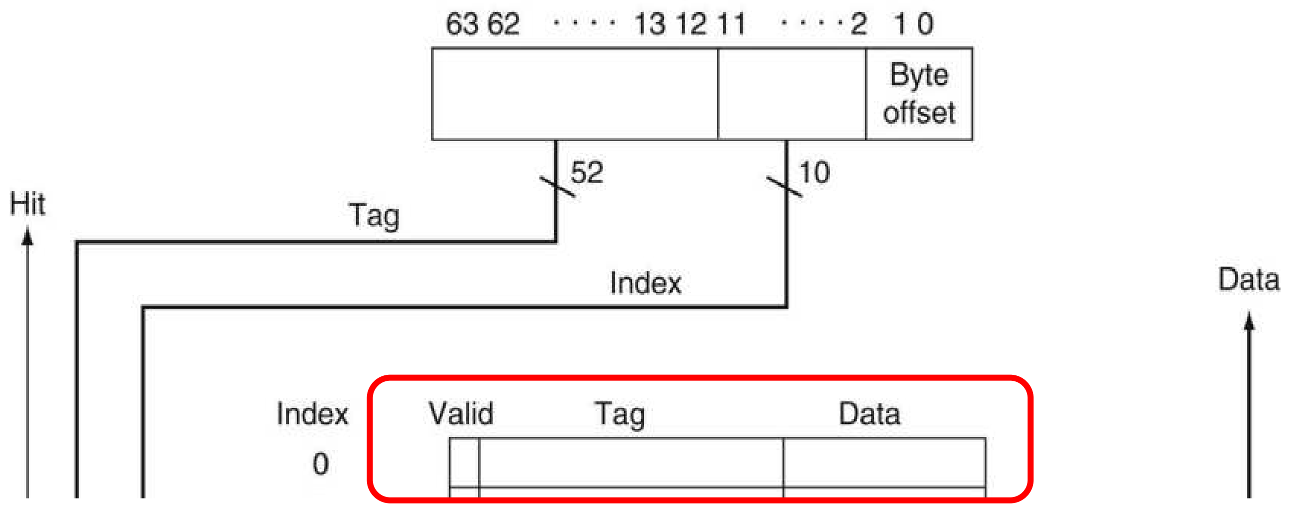
Answer the Quiz during mid semester quiz.

Cache

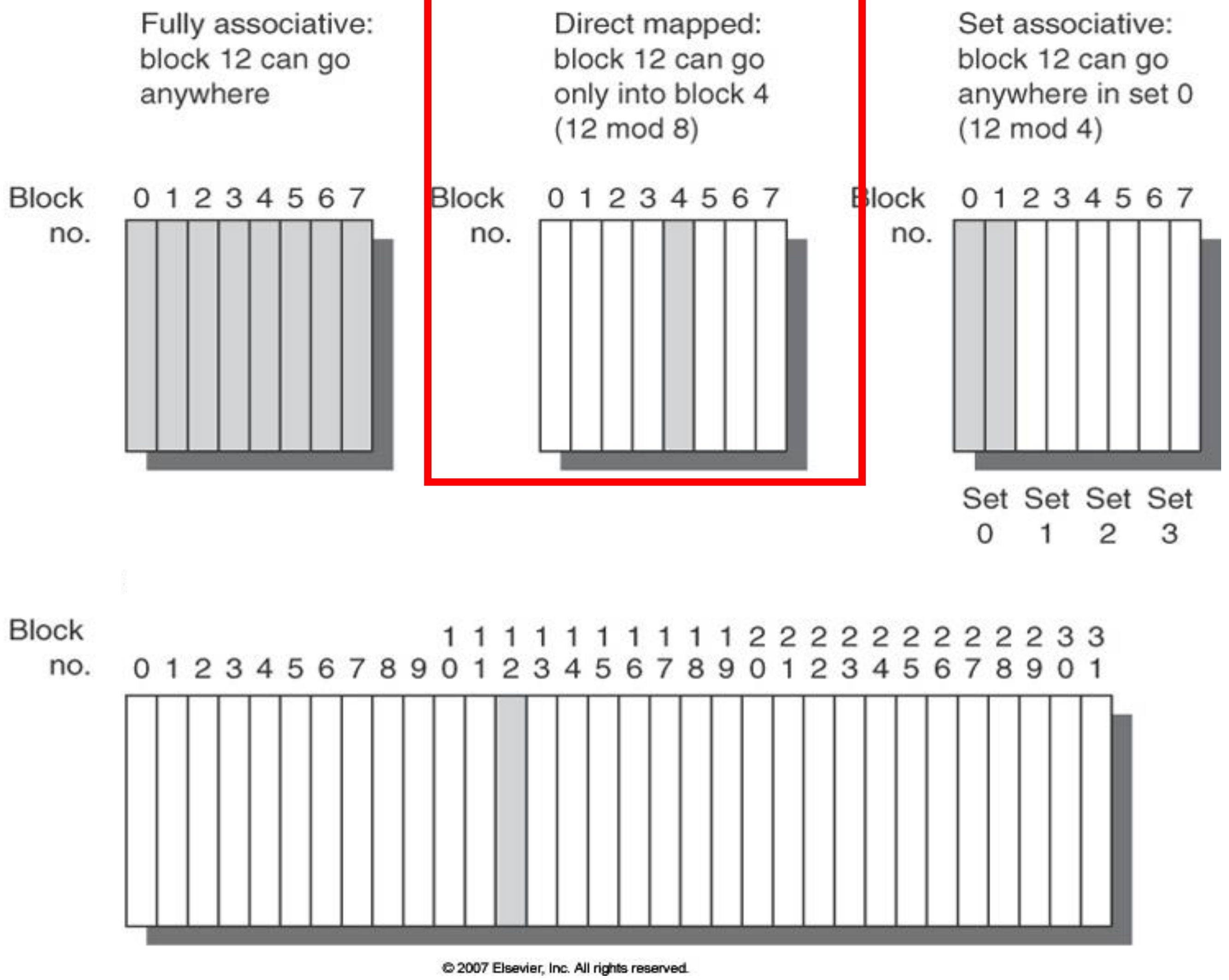


Source: http://archive.arstechnica.com/paedia/c/caching/m-caching-5.html

- Cache hits/misses
- Average memory access time
- Accessing cache
- Cache organization



Cache Associativity

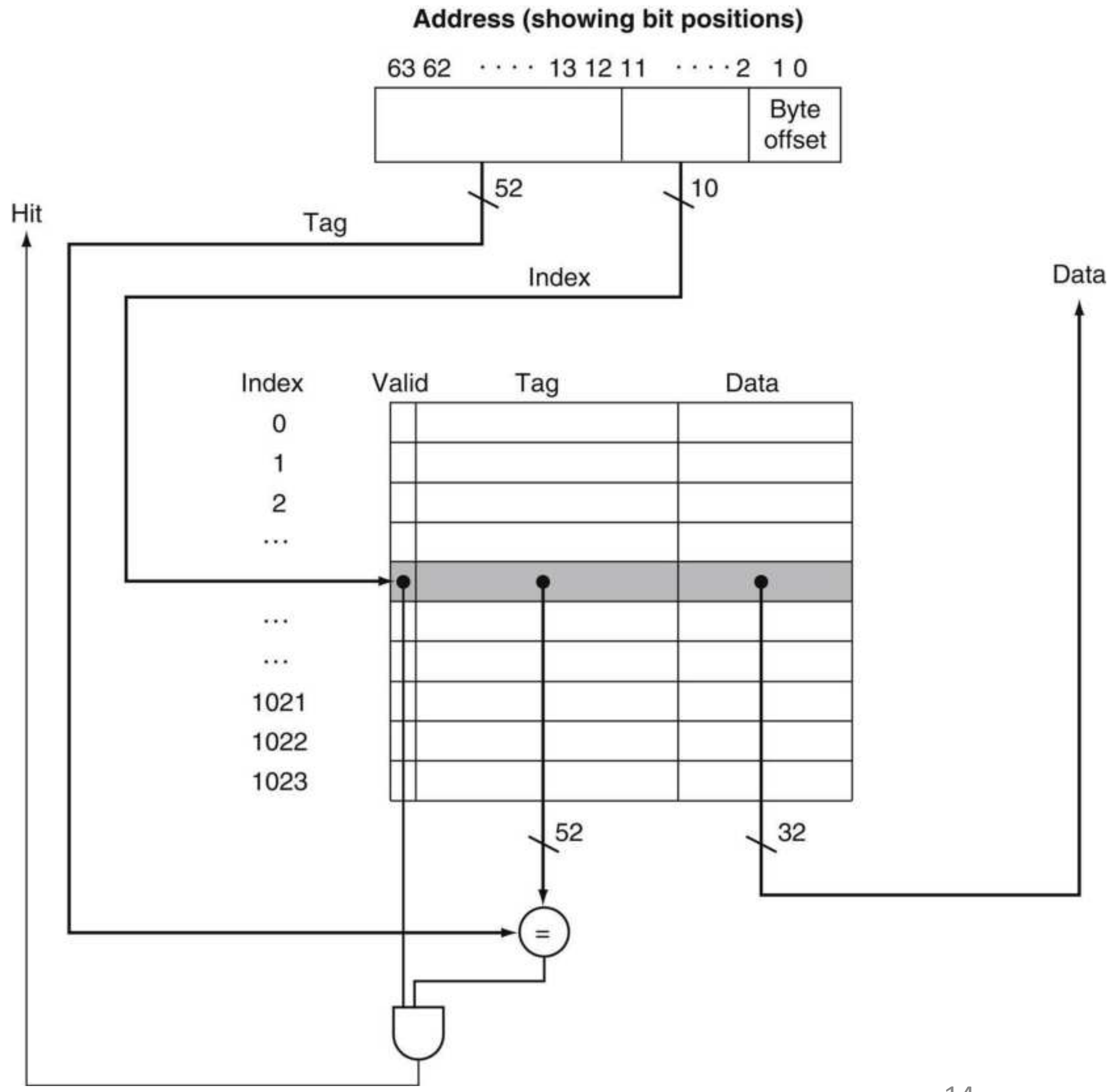

Fully associative: block 12 can go anywhere
Direct mapped: block 12 can go only into block 4 (12 mod 8)
Set associative: block 12 can go anywhere in set 0 (12 mod 4)
Block no.
0 1 2 3 4 5 6 7
Set 0 Set 1 Set 2 Set 3
Block no.
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
© 2007 Elsevier, Inc. All rights reserved.

Source: Computer Architecture, A Quantitative Approach by John L. Hennessy and David A. Patterson

Accessing Cache

- Tag = upper portion of address
- Index = (Memory block address) modulo #Cache blocks
- Valid bit
- Example:

1. How many bits are required for a direct-mapped cache with 16KiB of data and four-word blocks (addresses are 64bit)



Handling Cache Misses - Read

- Cache miss → processor stall
  - Freeze the register content & wait for memory in in-order processors (opposite: out-of-order processors)
- Steps:
  1. Processor generates the address to access the memory
  2. Cache miss occurs (=tag not matched)
  3. Instruct memory to do a read operation
  4. Wait for memory action to complete
  5. Write cache entry: data→data portion, upper bits of address → tag, valid bit →1
  6. Processor resume from where stalled

Handling Cache Misses - Write

- Processor writes ONLY to the cache (no change in main memory)

  ➔ Cache Inconsistency
- Write-through: write both to Cache and Memory at the same time

  → poor performance
  - Write-buffer – temporarily store data while waiting to write to memory
  - Frees the write-buffer only after memory write successfully completes
  - Otherwise, write-buffer is full → next write-through operation causes processor stall
- Write-back: new value written only to cache, memory is written only when the block is replaced
  - Performs better but complex to implement

Cache Replacement Policies

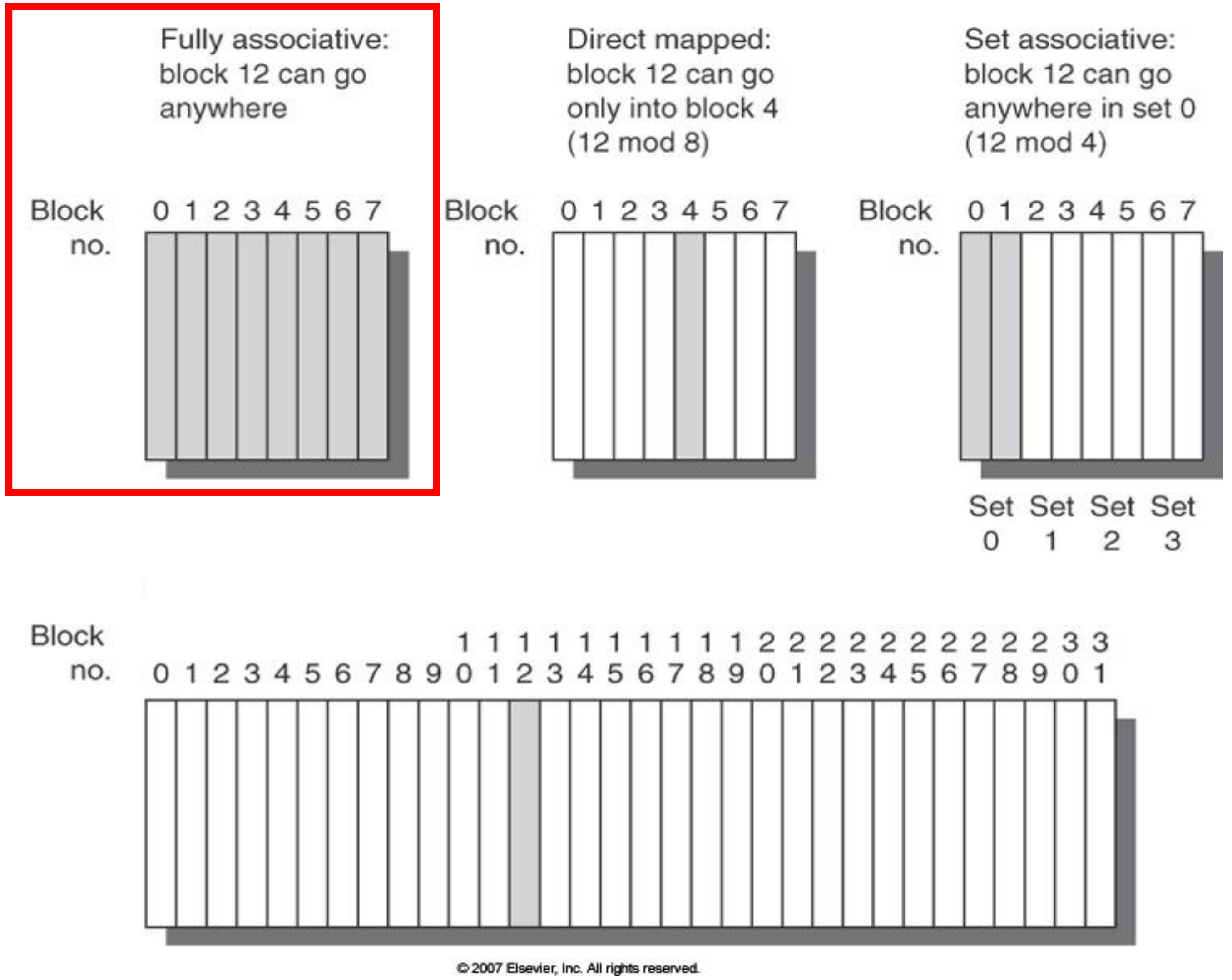
- When cache is full some of the cached blocks need to be removed before bringing new ones in
    - If cached blocks are dirty (written/updated), then they need to be written to RAM
- Cache replacement policies
    - Random: simple to build in hardware
    - Least Recently Used (LRU)
        - Need to track last access time
    - Least Frequently Used (LFU)
        - Need to track no of accesses
    - First In First Out (FIFO)

Fully Associative Cache

- Parallel tag search using hardware comparators

→ Hardware costs high

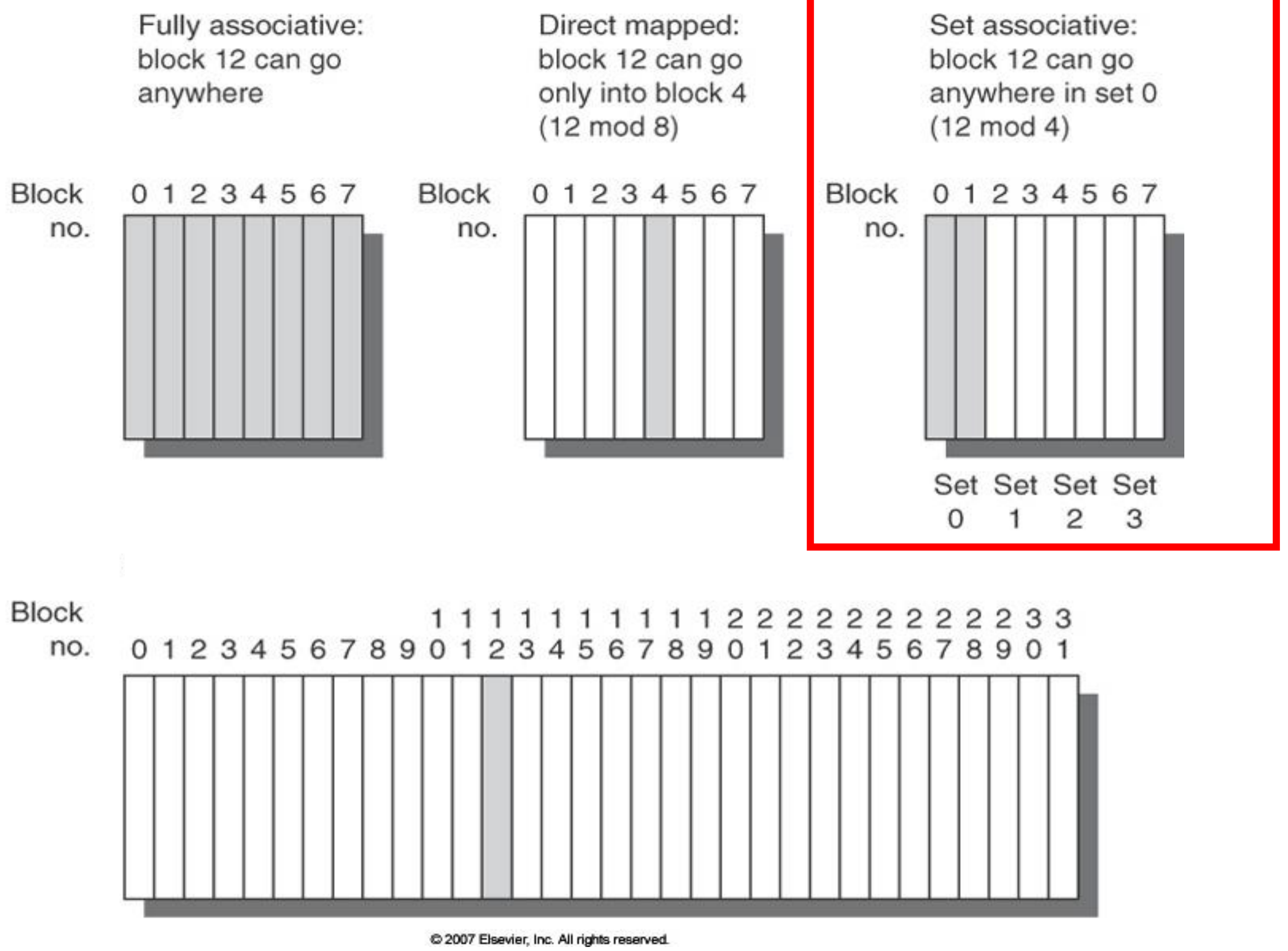
- Practical when #blocks in cache is less


Fully associative: block 12 can go anywhere
Block no.
0 1 2 3 4 5 6 7
Direct mapped: block 12 can go only into block 4 (12 mod 8)
Block no.
0 1 2 3 4 5 6 7
Set associative: block 12 can go anywhere in set 0 (12 mod 4)
Block no.
0 1 2 3 4 5 6 7
Set 0 Set 1 Set 2 Set 3
Block no.
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
© 2007 Elsevier, Inc. All rights reserved.

Source: Computer Architecture, A Quantitative Approach by John L. Hennessy and David A. Patterson

Set Associative Cache

- A block in main memory can go to any block in a set of cache blocks
- Combines Direct-mapping (Sets) and Fully associativity (Blocks)

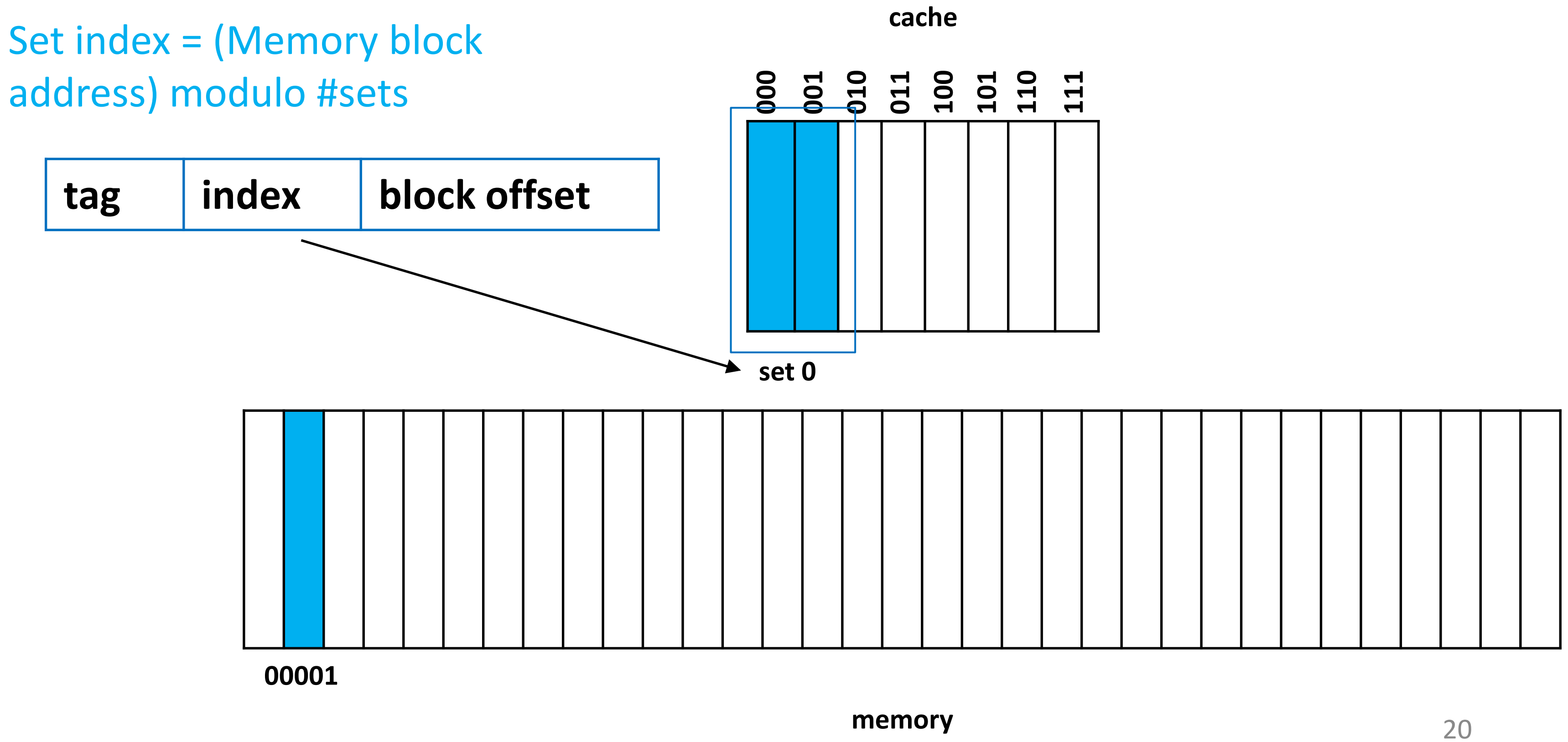


Fully associative: block 12 can go anywhere
Direct mapped: block 12 can go only into block 4 (12 mod 8)
Set associative: block 12 can go anywhere in set 0 (12 mod 4)
Block no.
0 1 2 3 4 5 6 7
Set 0
Set 1
Set 2
Set 3
Block no.
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
© 2007 Elsevier, Inc. All rights reserved.

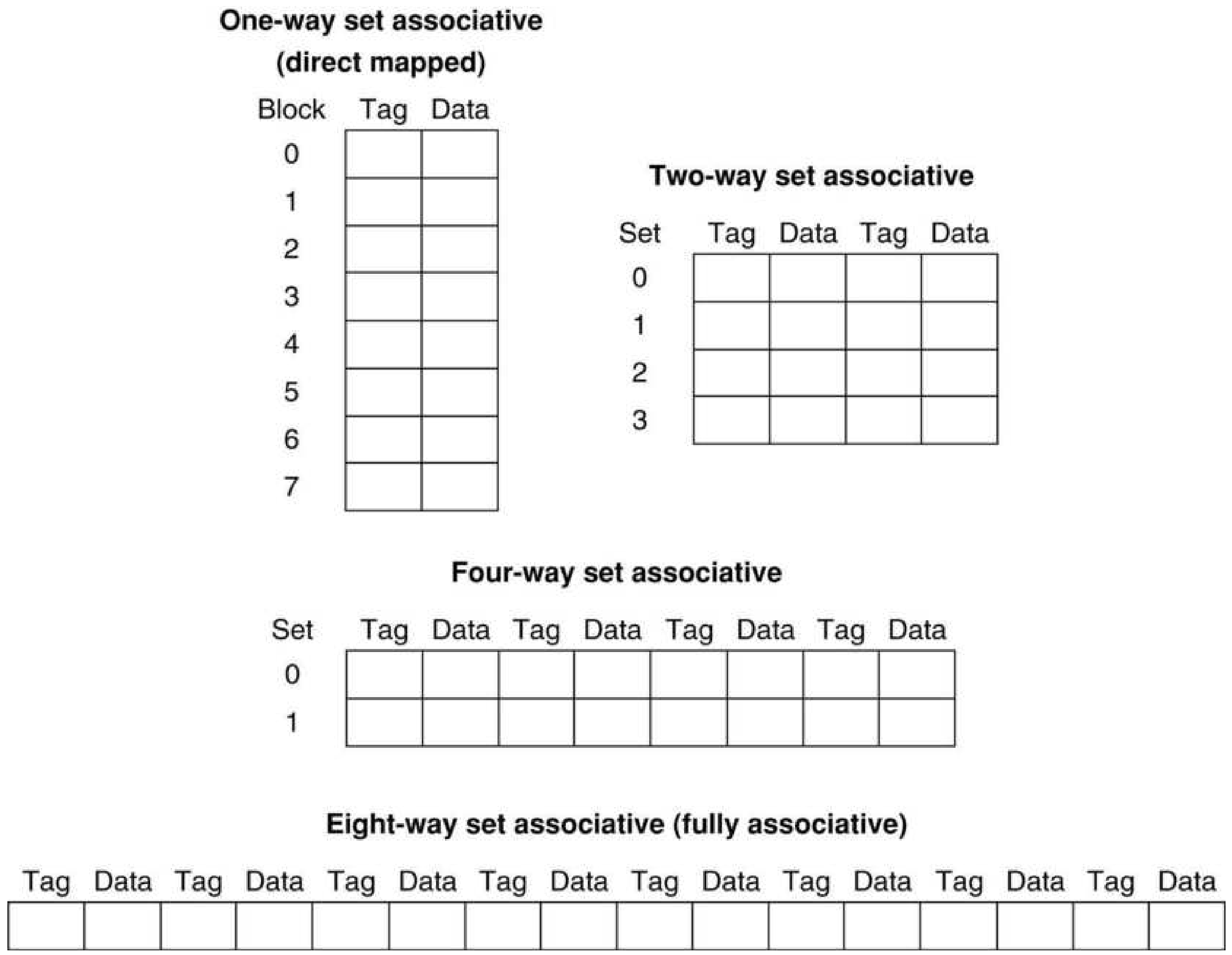
Source: Computer Architecture, A Quantitative Approach by John L. Hennessy and David A. Patterson

Set associative

- *n-way set associative*: *n* blocks for a set
- All tags of all elements in a set must be searched



Cache configuration examples

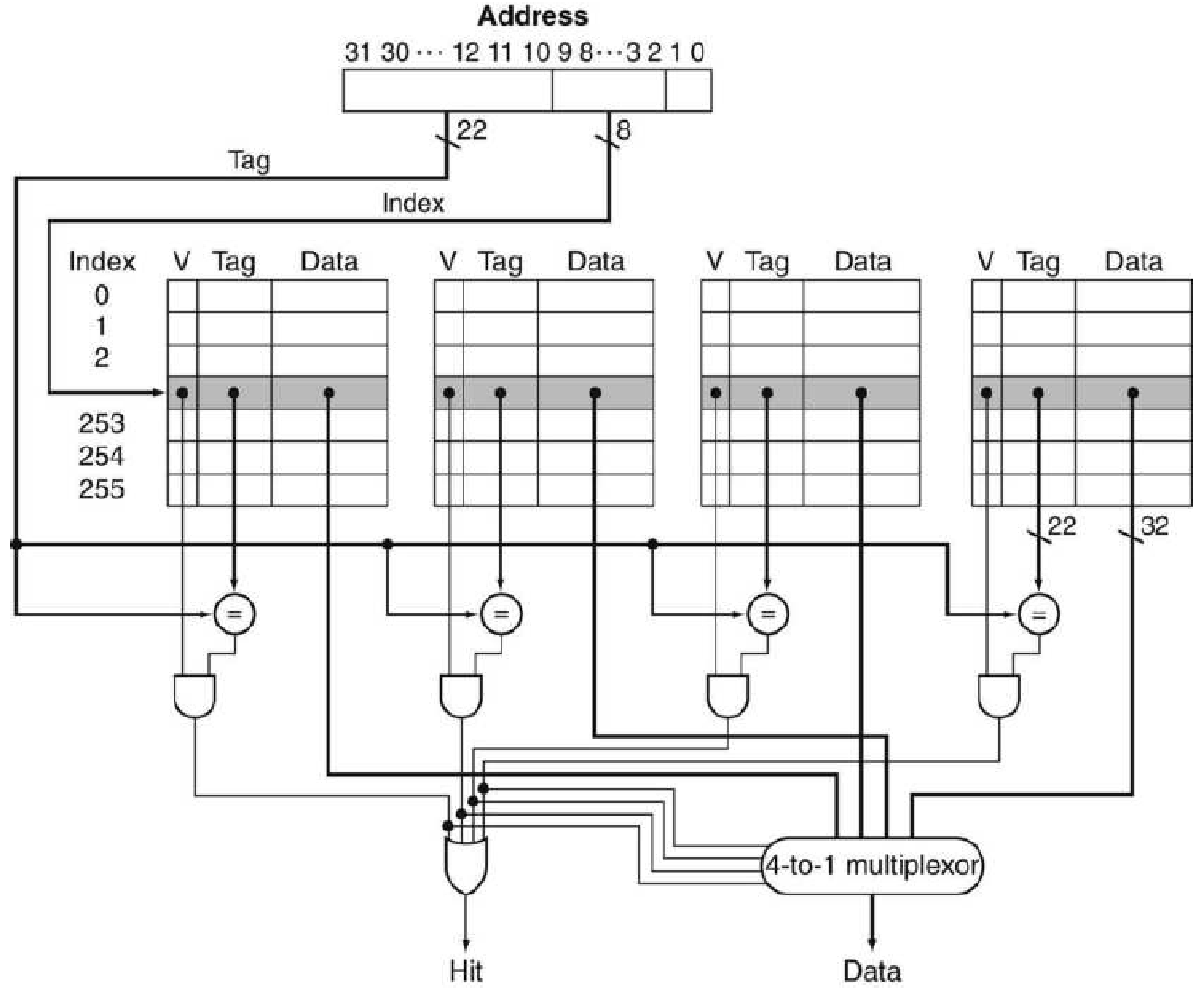


Associativity Vs Performance

- Refer the example in pages 785-787 and upload a similar analysis where block address sequence is A,B,C,X,Y,Z (ABCXYZ is the numerical part of your index number) – replace digit 9 with 8

- Take home assignment 3!

Accessing Set-Associative Cache



Size of Tags Vs Set Associativity - Exercise

- Cache size = 4096 Blocks, Block size = 4 words, Address size = 64 bits
- Calculate
    1. Total number of sets
    2. Total number of tag bits

    Where the cache is

    (a) Direct mapped

    (b) Two-way set associative

    (c) Fully associative

Least Recently Used (LRU)

- Replaces the block unused for the longest time
- Keeps track of usage of blocks
  - Expensive when #blocks increases
- Relies on temporal locality

Increasing Cache Performance

- Large cache capacity
- Multiple-levels of cache
- Prefetching
    - a block of data is brought into the cache before it is actually referenced
- Fully associative cache

Thank you