Technical University of Cluj-Napoca
Computer Science Department

Computer Architecture

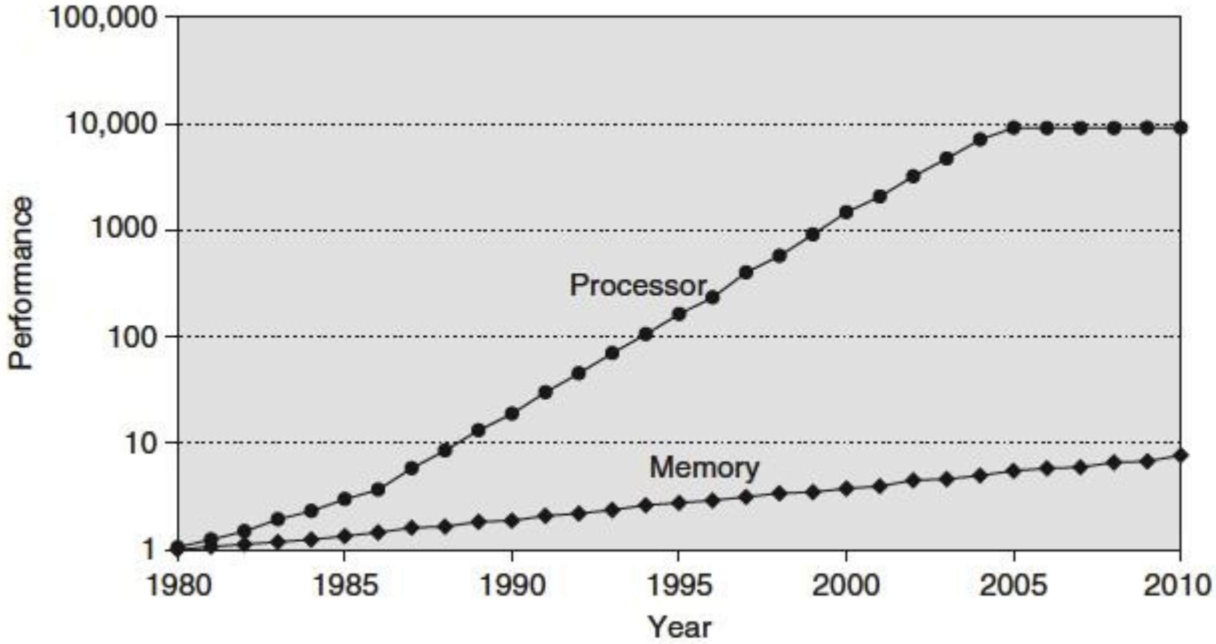
Lecturer: Mihai Negru

2nd Year, Computer Science

Lecture 11: Memory

http://users.utcluj.ro/~negrum/

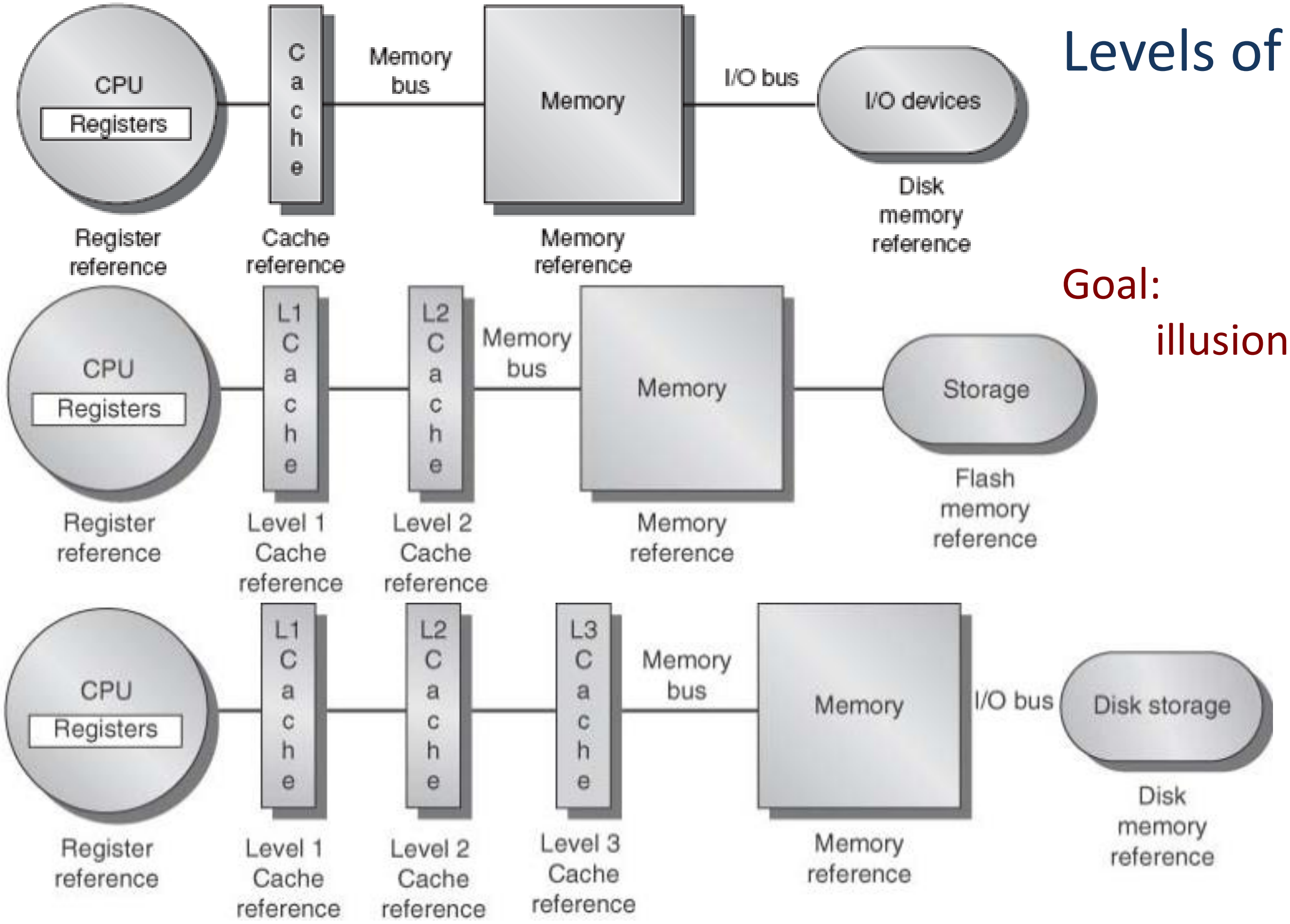
Processor – Memory Performance Gap



Processor (Single Core) vs. Memory (DRAM) Performance Gap [1]

Memory Technology	Typical Access Time	Cost / GB in 2012
SRAM semiconductor memory	0.5 – 2.5 ns	$500 – $1000
DRAM semiconductor memory	50 – 70 ns	$10 – $20
Flash semiconductor memory	5,000 – 50,000 ns	$0.75 – $1.00
Magnetic disk	5,000,000 – 20,000,000 ns	$0.05 – $0.10

Memory Hierarchy



Levels of memory hierarchy [1]

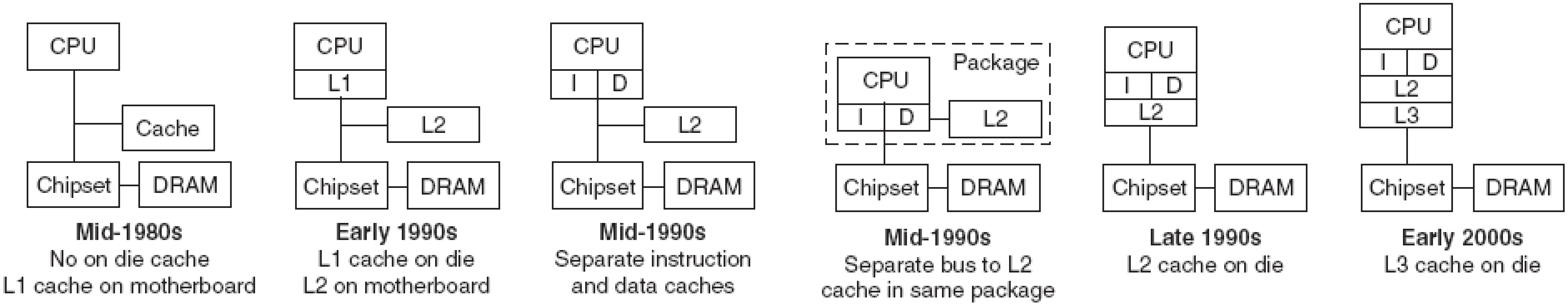
Goal:

illusion of large, fast, cheap memory

Distance from Processor → lower speed but greater size

Cache – a safe place for hiding or storing things!

Memory Hierarchy Evolution



The Evolution of the Memory Hierarchy. Separate Instruction and Data Caches

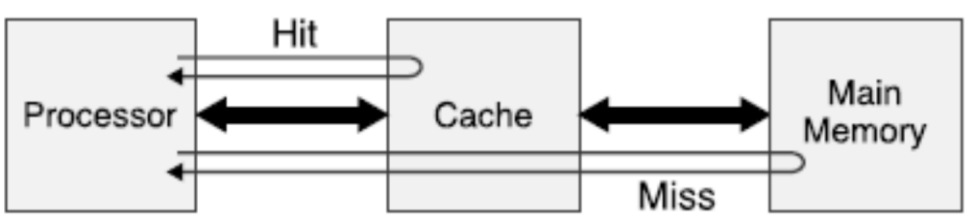
Why hierarchy works? The Principle of Locality:

- Temporal Locality – Locality in Time
  - If a data location is referenced then it will tend to be referenced again soon
  - Keep most recently accessed data items closer to the processor.
- Spatial Locality – Locality in Space
  - If a data location is referenced, nearby addresses will tend to be referenced soon
  - Move blocks consisting of contiguous words to the upper levels.

Cache Memory Terminology

- Hit: data requested is in upper level.
- Miss: data requested is not in upper level.
- Hit rate: fraction of memory accesses that are hits (i.e., found at upper level).
- Miss rate: fraction of memory accesses that are not hits: miss rate = 1 – hit rate
- Hit time: time to determine if the access is a hit + time to access and deliver the data from the upper level to the CPU.
- Miss penalty: time to determine if the access is a miss + time to replace block at upper level with corresponding block at lower level + time to deliver the block to the CPU

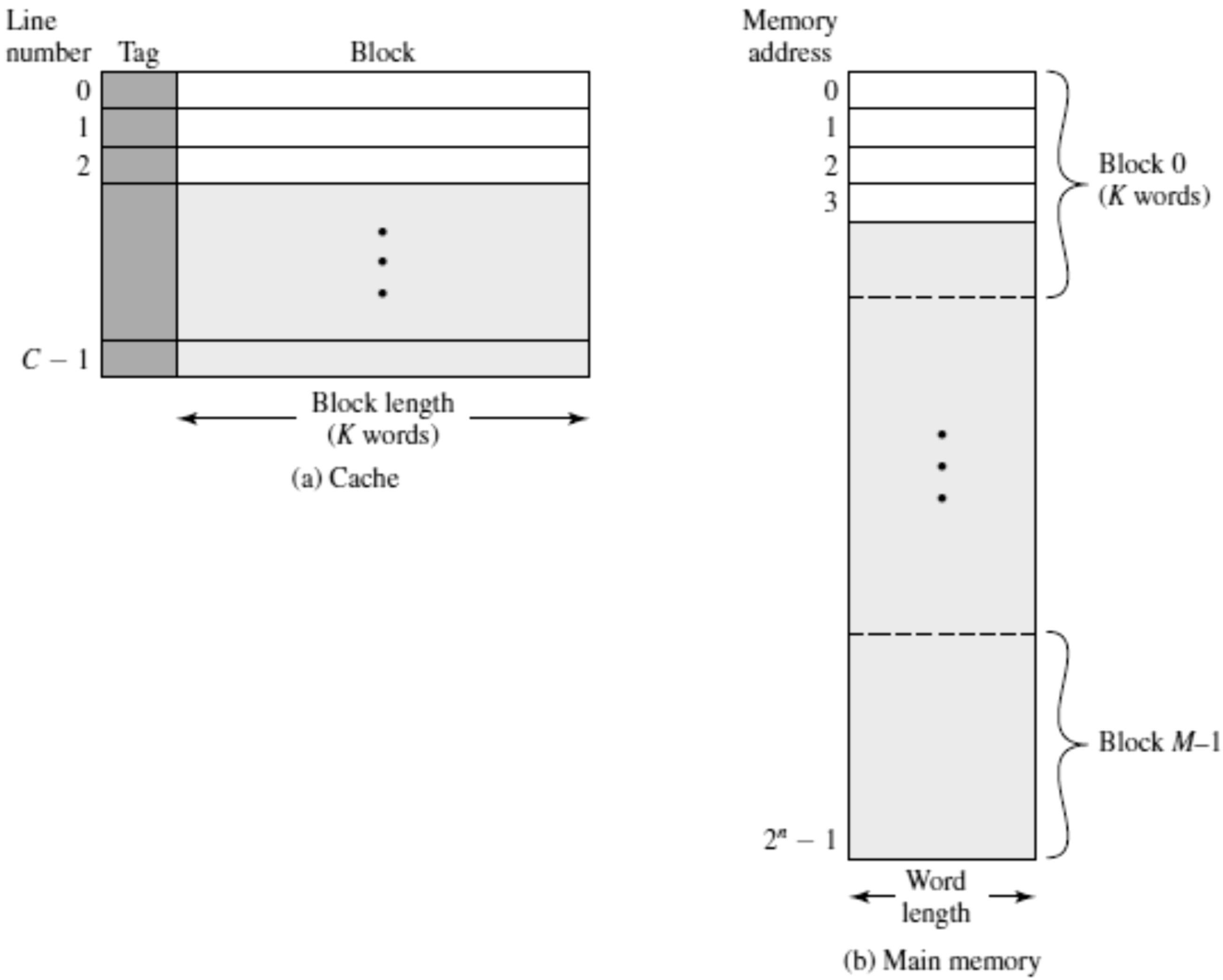
- Average memory access time (AMAT) = Hit time + Miss rate x Miss penalty



[1]

- To improve performance → reduce AMAT
  - Reduce the miss rate, miss penalty, or the hit time

Cache Memories



Block in Cache and Main Memory [1]

Cache Memories

- 4 Problems for Cache Memory Specification

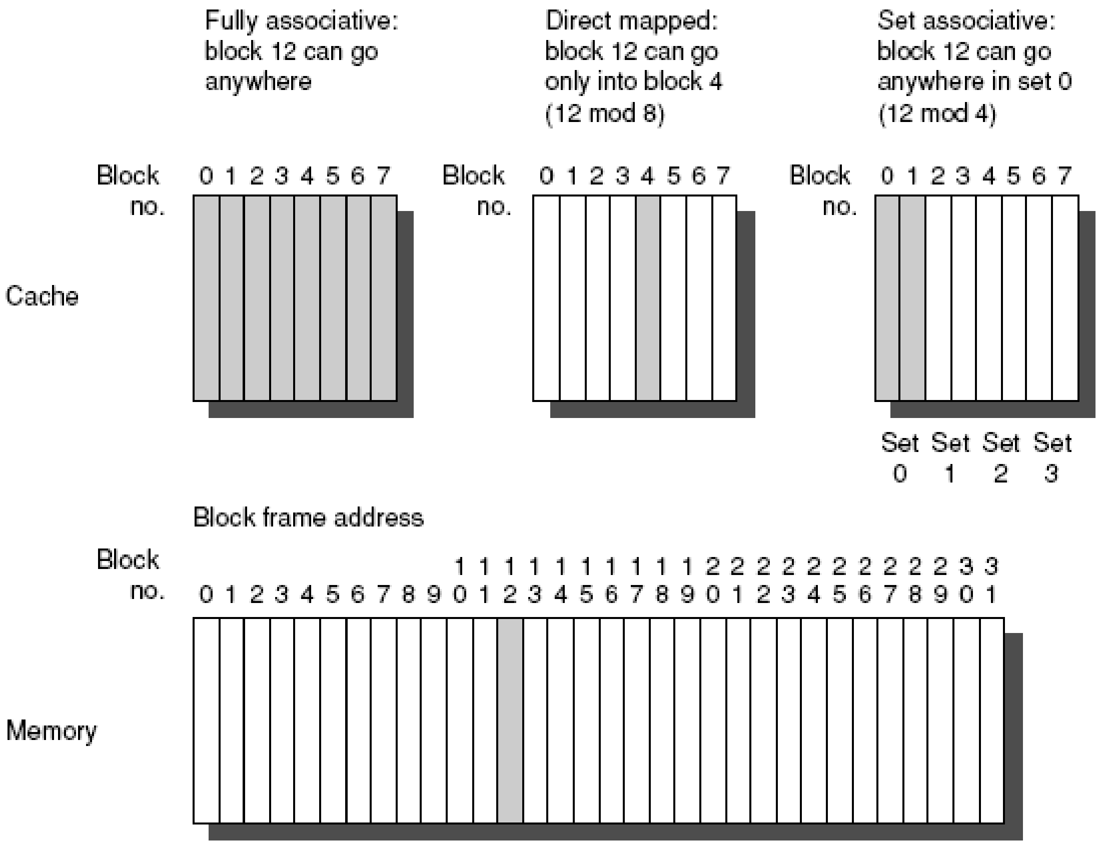
  – Q1: Where can a block be placed in the Cache? (Block placement)
    - Associativity: Fully Associative, Set Associative, Direct Mapped

  – Q2: How is a block found in the Cache? (Block identification)
    - Tag / Index / Block

  – Q3: Which block should be replaced on a Cache miss? (Block replacement)
    - Random, LRU, FIFO, NLRU, FIFO with exception for most recently used

  – Q4: How to write in the Cache? (Write strategy)
    - Write Back or Write Through, Write Buffer

Q1: Block Placement in the Cache Memory



Example: Cache memory with 8 blocks [1]

Cache Memory Organization

- Direct Mapped Cache – each block has only one corresponding place in the cache
- Fully Associative Cache – a block can be placed anywhere in the cache
- Set Associative Cache – a block can be placed in a restricted set of places in the cache. N sets in a cache → N-way Set Associative

Address mapping for set assoc. cache: (Block address) MOD (Nb. sets in the cache)

"The miss rate of a Direct Mapped Cache of size X is about the same as for a 2- to 4-way Set Associative cache of size X/2"

Q1: Block Placement in the Cache Memory

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Direct Mapped Cache
1-way Set Associative Cache

Set = a group of blocks,
Index in the cache memory

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

2-way Set Associative Cache

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

4-way Set Associative Cache

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

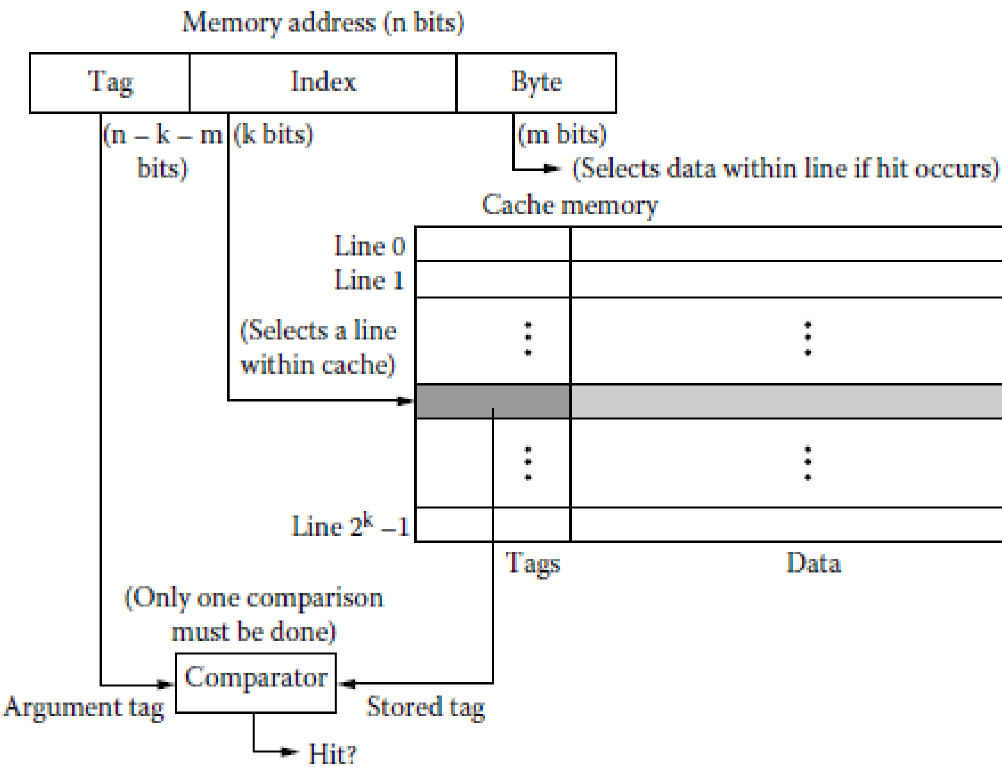
8-way Set Associative (Fully Associative)

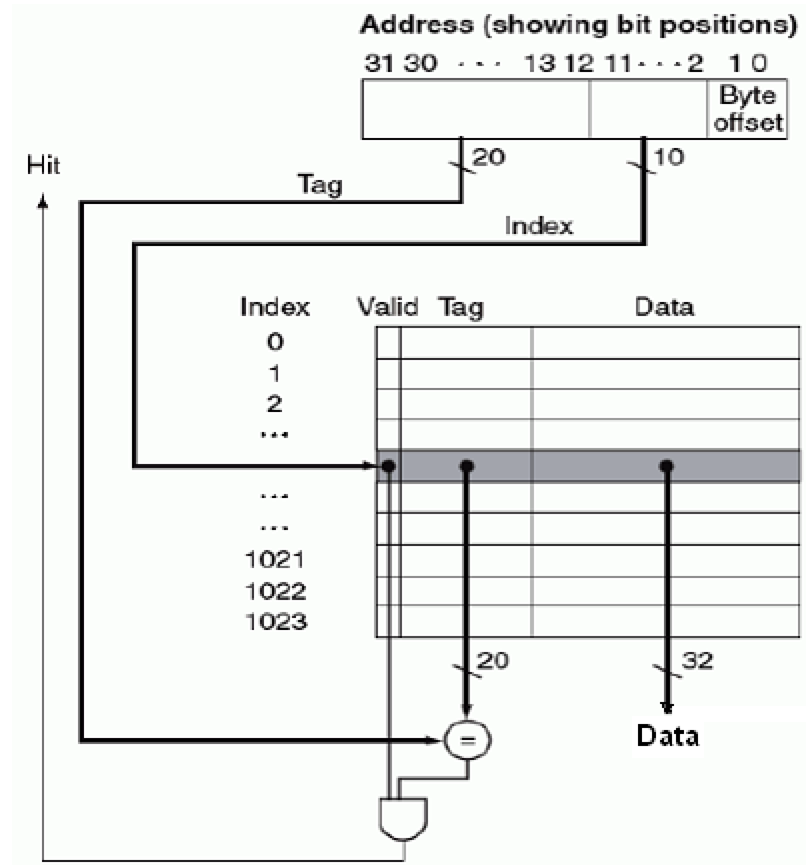
Configurations of an 8-block cache with different degrees of associativity.

Q2: Block Identification in the Cache Memory

Block address		Block offset
Tag	Index	

Address for a Set Associative or Direct Mapped Cache
A fully Associative caches has no index field

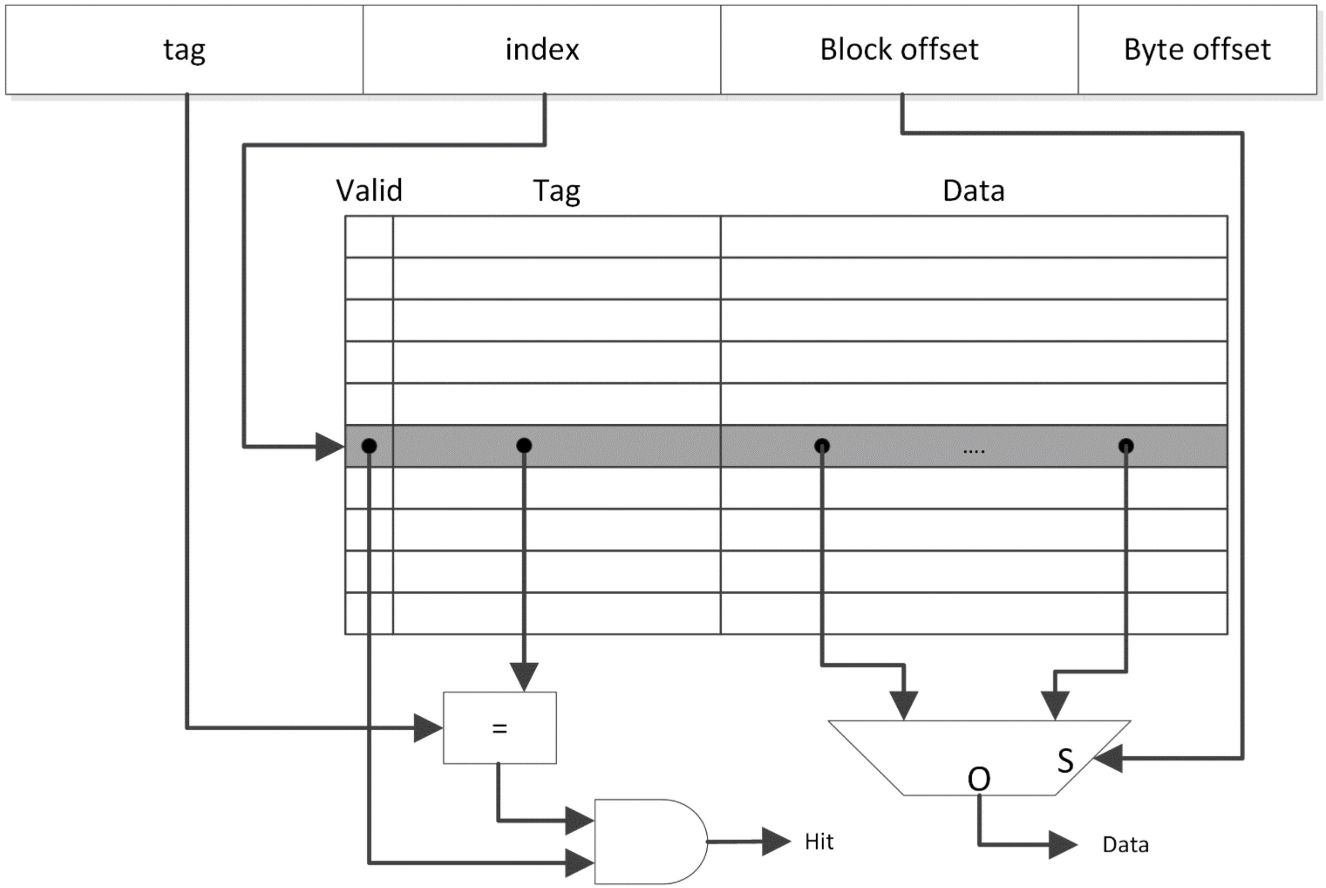




Direct Mapped Cache – general [1]

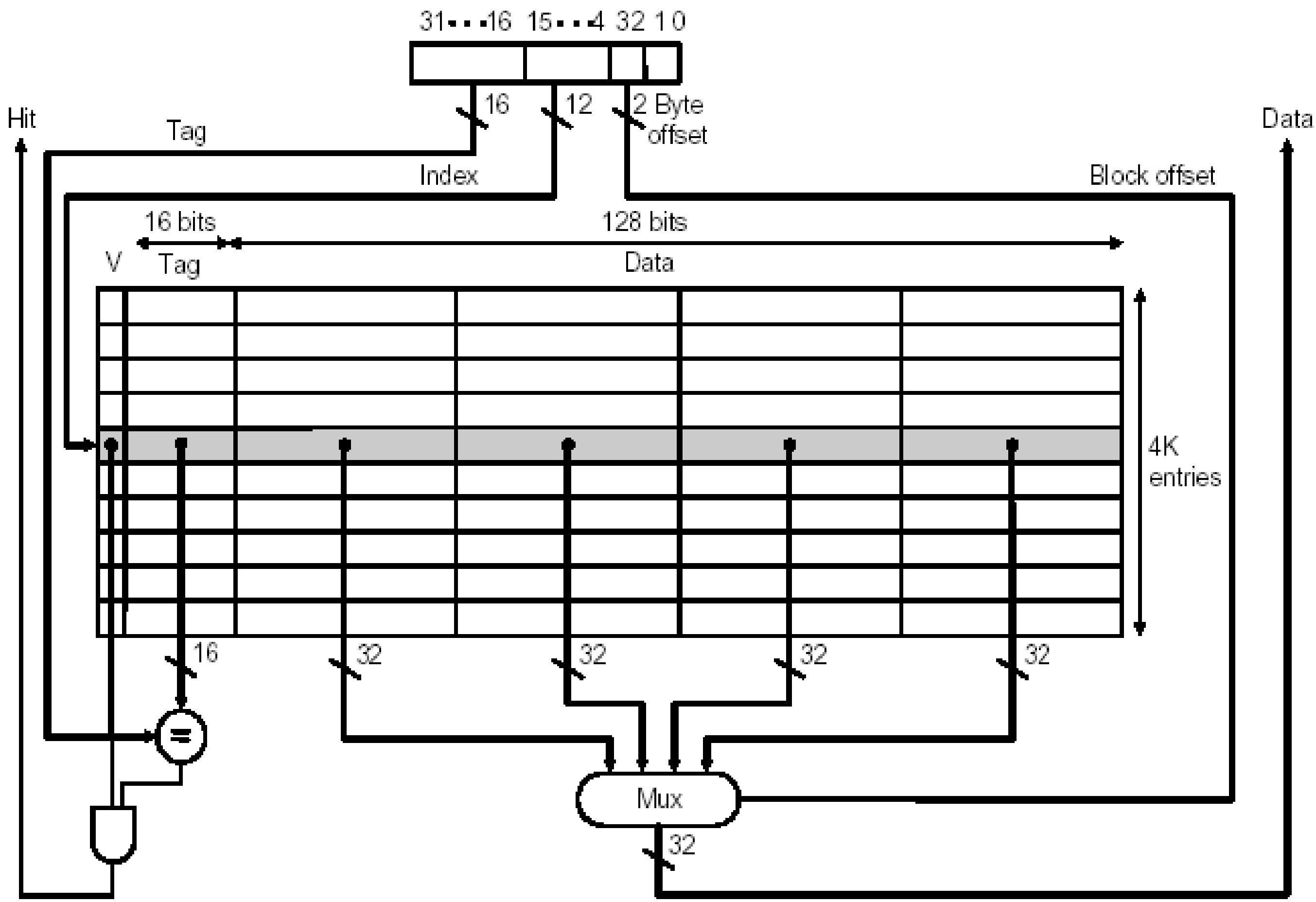
Direct Mapped Cache, 1024, 1-word blocks [1]

Q2: Block Identification in the Cache Memory



Direct Mapped Cache with multiple data/tag. Taking advantage of spatial locality

Q2: Block Identification in the Cache Memory

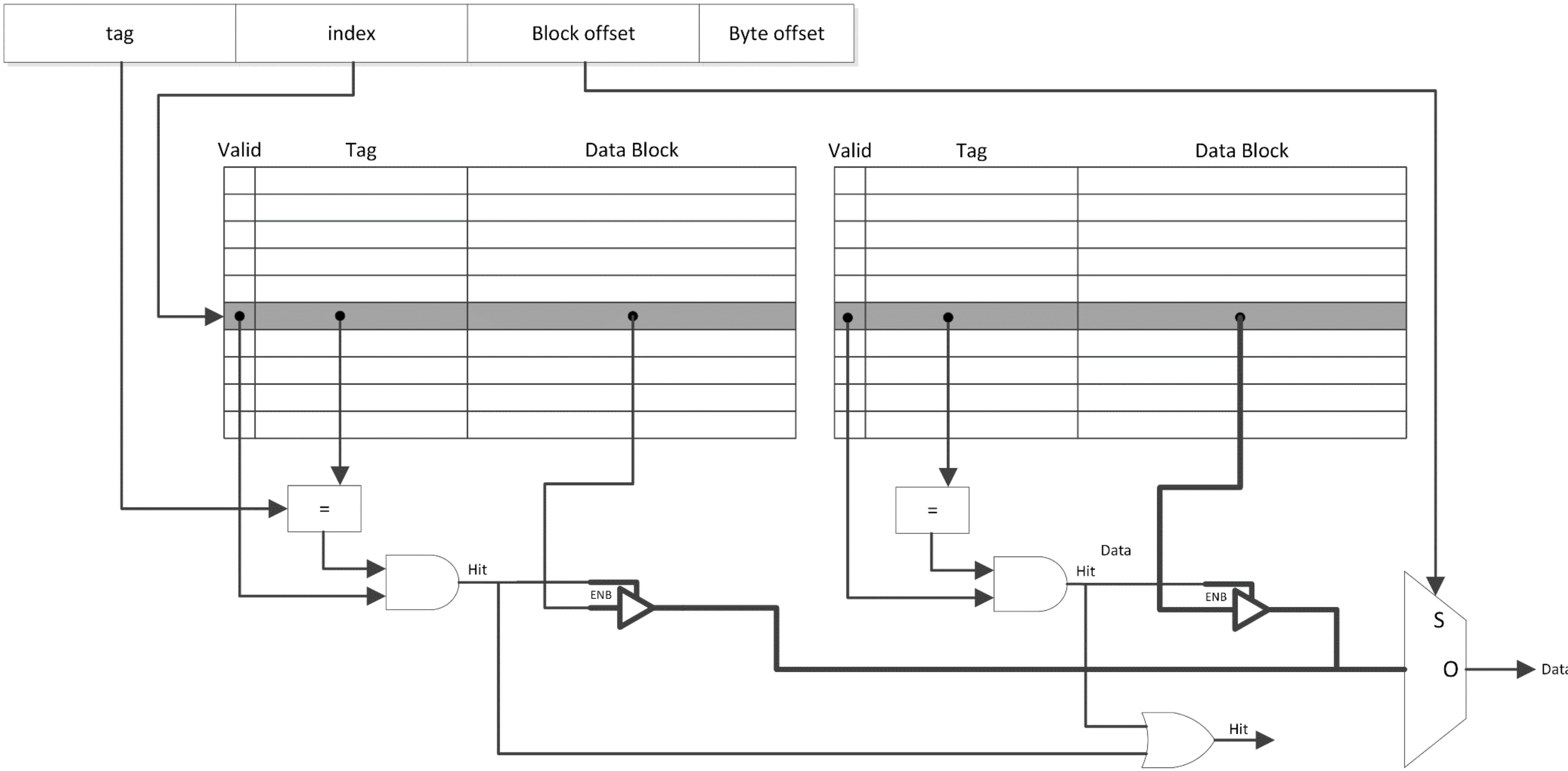


Direct Mapped Cache with multiple data/tag [1]. Taking advantage of spatial locality

- 64 KB cache, 4K blocks, 4 words per block; byte offset ignored – we read words (32 bits) from cache; block offset – which word to read.

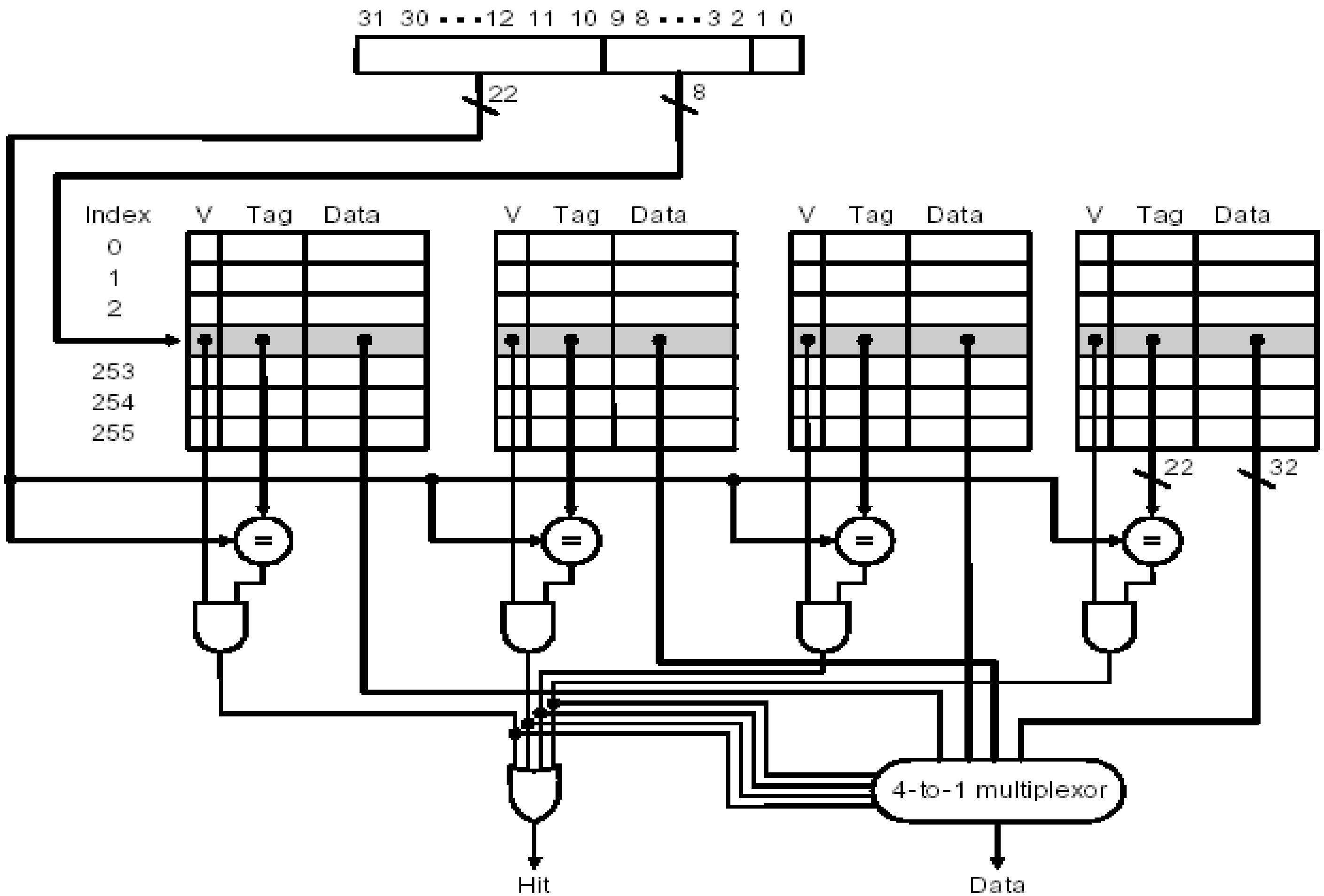
Q2: Block Identification in the Cache Memory

- N-way Set Associative Cache: N entries for each cache index
  - N Direct Mapped Caches that operate in parallel



2-way Set Associative Cache with 2 comparators and 1 multiplexer

Q2: Block Identification in the Cache Memory



4-way Set Associative Cache Memory [1]

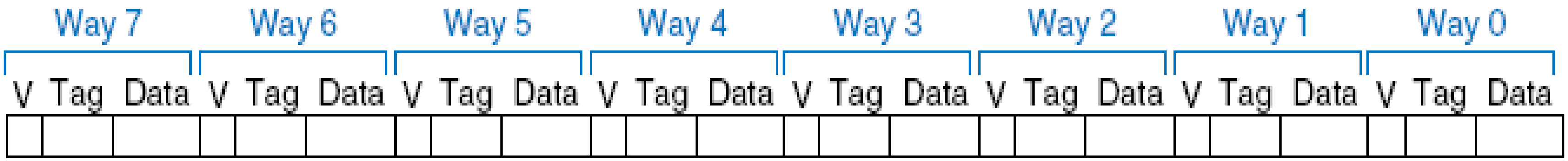
- 4-way set-associative cache with 4 comparators and one 4-to-1 multiplexor
- Size of cache: 4 KB cache, 1K blocks = 256 sets * 4-block/set (4x256 blocks, 1 word per block)

Q2: Block Identification in the Cache Memory

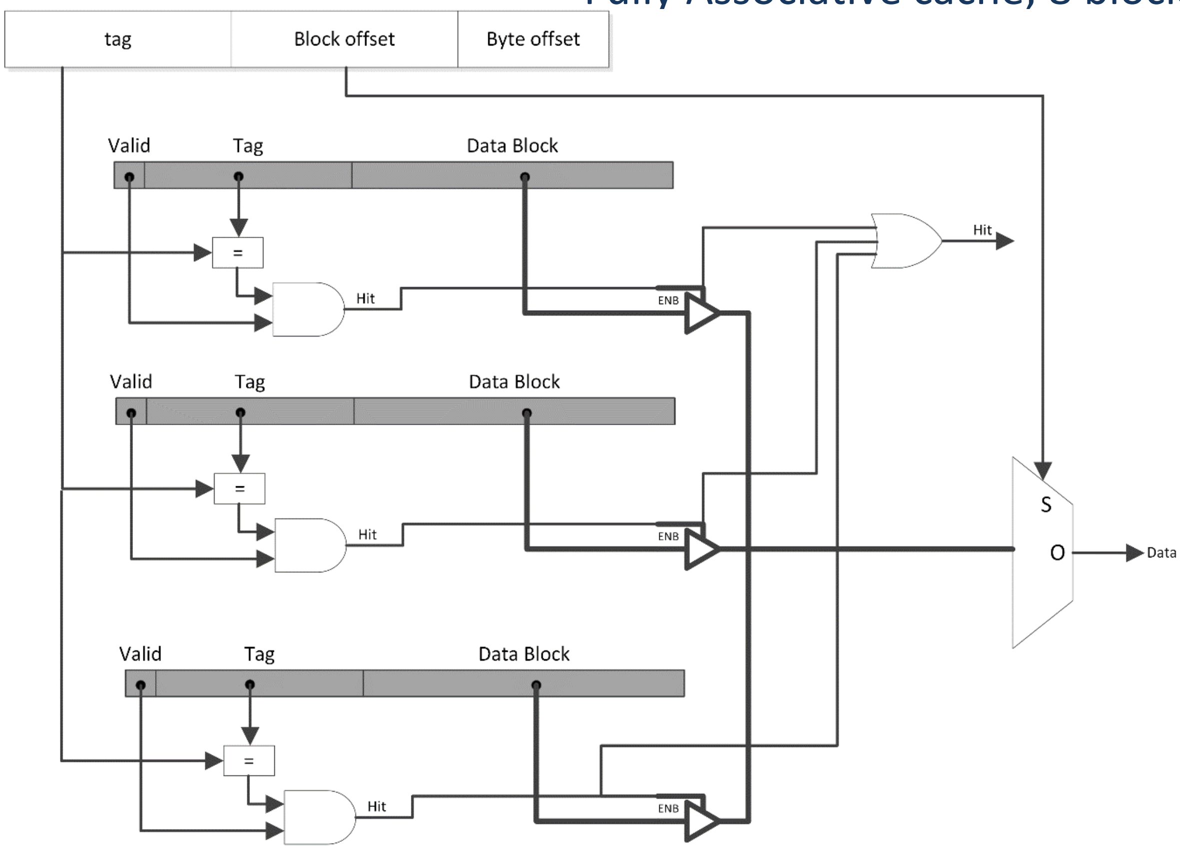
- Advantages of Set Associative Cache
  - Higher Hit rate for the same cache size.
  - Fewer Conflict Misses.

- Disadvantages of Set Associative Cache
  - N-way Set Associative Cache versus Direct Mapped Cache
    - N comparators vs. 1
    - Extra MUX delay for the data
    - Data comes AFTER Hit/Miss decision and set selection
    - In a Direct Mapped Cache, Cache Block is available BEFORE Hit/Miss
    - Possible to assume a hit and continue. Recover later if miss.

Q2: Block Identification in the Cache Memory



Fully Associative cache, 8 blocks [1]



Fully Associative Cache Memory

Fully Associative Cache

No Cache Index.

Compare the Cache Tags of all cache entries in parallel.

Needs a lot of comparators.

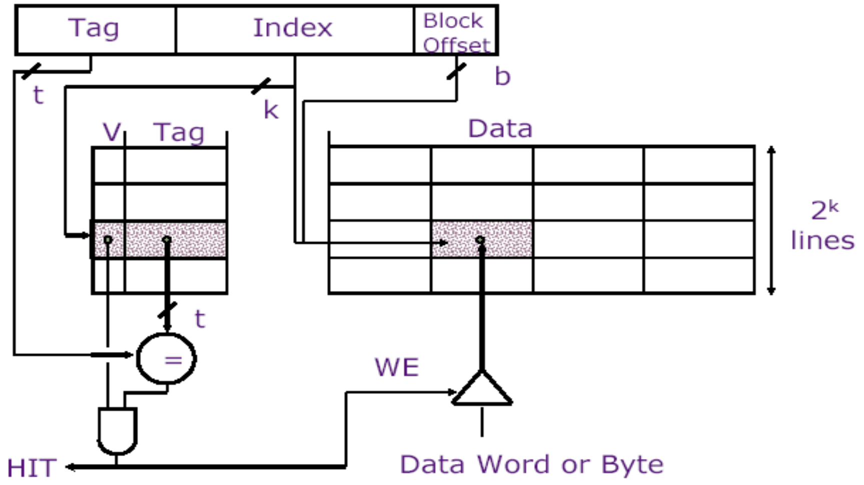
Implemented using content addressable memory (CAM).

Conflict Miss = 0 for a fully associative cache.

Q3: Block Replacement on a Cache Miss

- Direct Mapped Cache – easy: only one possible block to replace
- Associative cache – need a block replacement algorithm
  - Least Recently Used (LRU)
    - Expensive – keeps track when an element in the set was used
    - For 2-way set assoc. – use 1 bit (USE bit)
    - On a block reference Use bit $\leftarrow$ 1; Use Bit of the other block $\leftarrow$ 0
    - For fully associative – keep track of all references. Use a list: the most recently used block is the front of the list. The last block in the list is replaced
  - First-In-First-Out (FIFO)
    - Replace the block that has been in the cache longest
    - Easy to implement as a round-robin or circular buffer reference
  - Least Frequently Used (LFU)
    - Counter per block that increments on reference
    - Block with lowest count is replaced
  - Most Recently Used (MRU)
  - Random
    - Victim blocks are randomly selected
    - Simulations indicate almost as good as LRU

Q4: Write Strategy for Cache Memories



- Cache write
  - Modifying a block cannot begin until the tag is checked to see if the address is a hit.
  - Tag checking is done before the write → writes normally take longer than reads.
  - Write size: 1 – 8 bytes specified; only that portion of a block can be changed.
  - In contrast, reads can access more bytes than necessary
  - Pipelined writes: hold write data for store in single buffer ahead of cache, write cache data during next store's tag check

Q4: Write Strategy for Cache Memories

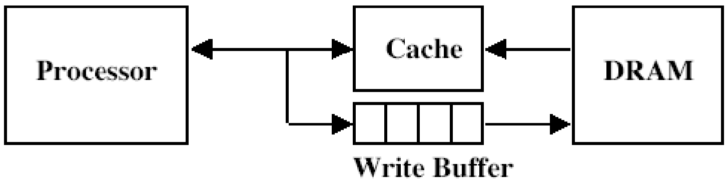
- Write Policy Choices
  - Write-Through (WT)
    - Replaces a block in the cache and low-level memory to avoid inconsistency
    - Write-through is slow because it always requires a write in main memory
    - Performance is improved with a write buffer where blocks are stored while waiting to be written to memory – processor can continue execution until write buffer is full
    - Advantages: read misses do not result in writes and assures data coherency
  - Write-Back (WB)
    - Write the data block only into the cache and write-back the block to main memory only when it is replaced in the cache
    - More efficient than write-through, more complex to implement
    - A dirty bit per block can further reduce the traffic
  - Write Once – first write as write-through, the followings as write back

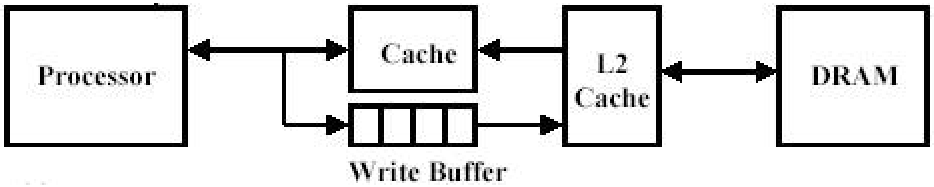
Q4: Write Strategy for Cache Memories

- Write miss actions: allocate block if it's a miss?
  - Write allocate – the block is allocated on a write miss, followed by the write hit.
  - No write allocate – only write to main memory.
  - Common combinations
    - Write through and no write allocate, even if there are subsequent writes to that block, the writes must still go to the lower level memory.
      - WT combined with write buffers so that it doesn't wait for memory.
    - Write back with write allocate, hoping that subsequent writes to that block will be captured by the cache

Q4: Write Strategy for Cache Memories

- Write Buffer
  - Contains evicted dirty lines for WB cache or all writes in WT cache
  - It reduces Read Miss Penalty
    - Processor is not stalled on writes, read misses can go ahead of writes to main memory





  - Implemented as a FIFO (queue) – holds data to be written to memory
  - Memory controller writes contents of the buffer to memory
    - Frees the write buffer data entry after completing memory write
    - Stall the CPU if write buffer is full
  - Problem: Write Buffer may hold a value needed by a read miss!
    - Simple: on a read miss, wait for the write buffer to go empty
    - Faster: check write buffer addresses against read miss addresses
      - if no match, allow read miss to go ahead of writes, else
      - return the value from the write buffer

Cache Memory Performance

- AMAT (average memory access time)
- CPU Time

$$\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

$$\text{CPU time} = (\text{CPU Execution clock cycles} + \text{Memory stall clock cycles}) \times \text{Clock cycle time}$$

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{execution} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}} \right) \times \text{Clock cycle time}$$

$$\text{CPUtime} = \text{IC} \times \left(\text{Miss rate} \times \frac{\text{Memory access}}{\text{Instruction}} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

CPU Execution clock cycles – includes the execution clock cycles and the memory access for a cache hit

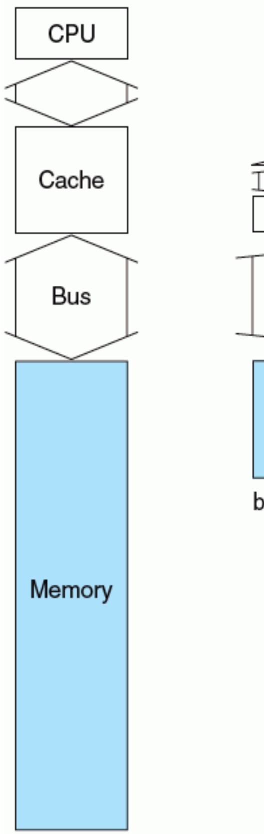
Memory stall clock cycles – includes the auxiliary penalties for working with memory

Causes for Cache Misses

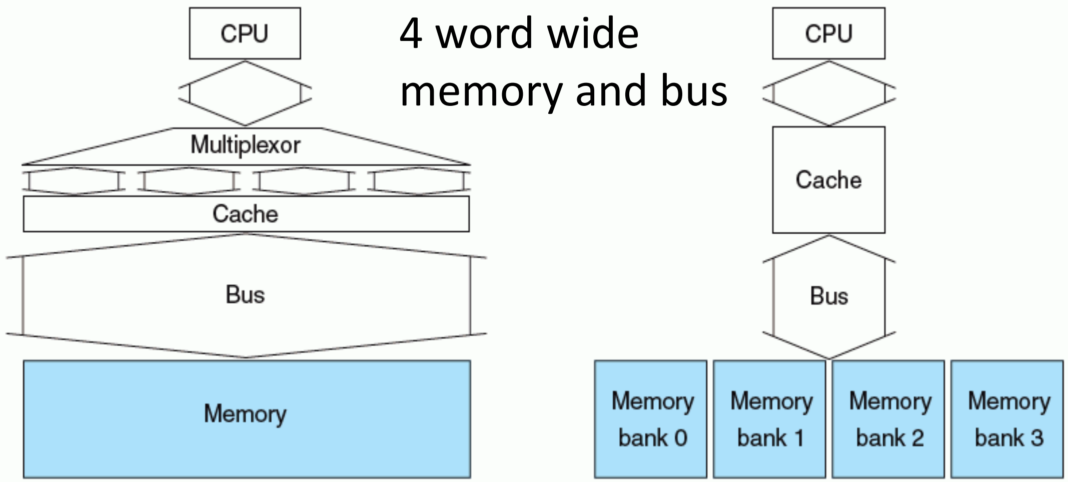
- 3 C's model
  - Compulsory: first-reference to a block (cold start misses)
    - Misses that would occur even with infinite cache
  - Capacity: cache is too small to hold all data needed by the program
    - Misses that would occur even under perfect replacement policy
  - Conflict: misses that occur because of collisions due to block-placement strategy
    - Misses that would not occur with full associativity
- 4th C: Coherence
  - Misses caused by cache coherence (Multiprocessors)

Design change	Effect on miss rate	Possible negative performance effect
Increase cache size	Decreases capacity misses	May increase access time
Increase associativity	Decreases miss rate due to conflict misses	May increase access time
Increase block size	Decreases miss rate due to spatial locality	Increases miss penalty. Very large block size can increase miss rate

Cache Memory Connections



a. One-word-wide memory organization



b. Wide memory organization

c. Interleaved memory organization

4 word wide memory and bus

4 word wide memory
Interleaved memory units compete for bus

Example: we assume

- cache block of 4 words
- 1 clock cycle to send address to memory address buffer (1 bus cycle)
- 15 clock cycles for each memory data access
- 1 clock cycle to send data to memory data buffer (1 bus cycle)

Miss penalties

- a: 1 + 4*15 + 4*1 = 65 cycles
- b: 1 + 1*15 +1*1 = 17 cycles
- c: 1 +1*15 + 4*1 = 20 cycles

Improving Cache Performance by Increasing Bandwidth [1]

Cache Memory Evolution

Processor	Year	Frequency (MHz)	Level 1 Data Cache	Level 1 Instruction Cache	Level 2 Cache
80386	1985	12 – 40	none	none	None
80486	1989	16 – 150	8 KB unified		None on chip
Pentium	1993	60 – 100	8 KB	8 KB	None on chip
Pentium Pro	1995	150 – 200	8 KB	8 KB	256 KB – 1 MB
Pentium II	1997	233 – 450	16 KB	16 KB	256 KB – 512 KB
Pentium III	1999	450 – 1400	16 KB	16 KB	256 KB – 512 KB
Pentium 4	2001	1400 – 3730	8-16 KB	12 KB	256 KB – 2 MB
Pentium M	2003	900 – 2130	32 KB	32 KB	1 – 2 MB on chip
Core Duo	2005	1500 – 2160	32 KB / core	32 KB / core	2 MB shared on chip
Skylake (Core I7)	2015	Up to 4200	32 KB / core	32 KB / core	256 KB / Core (8 M L3 cache)

Evolution of intel IA-32 Microprocessor Cache Memory Systems

Virtual Memory

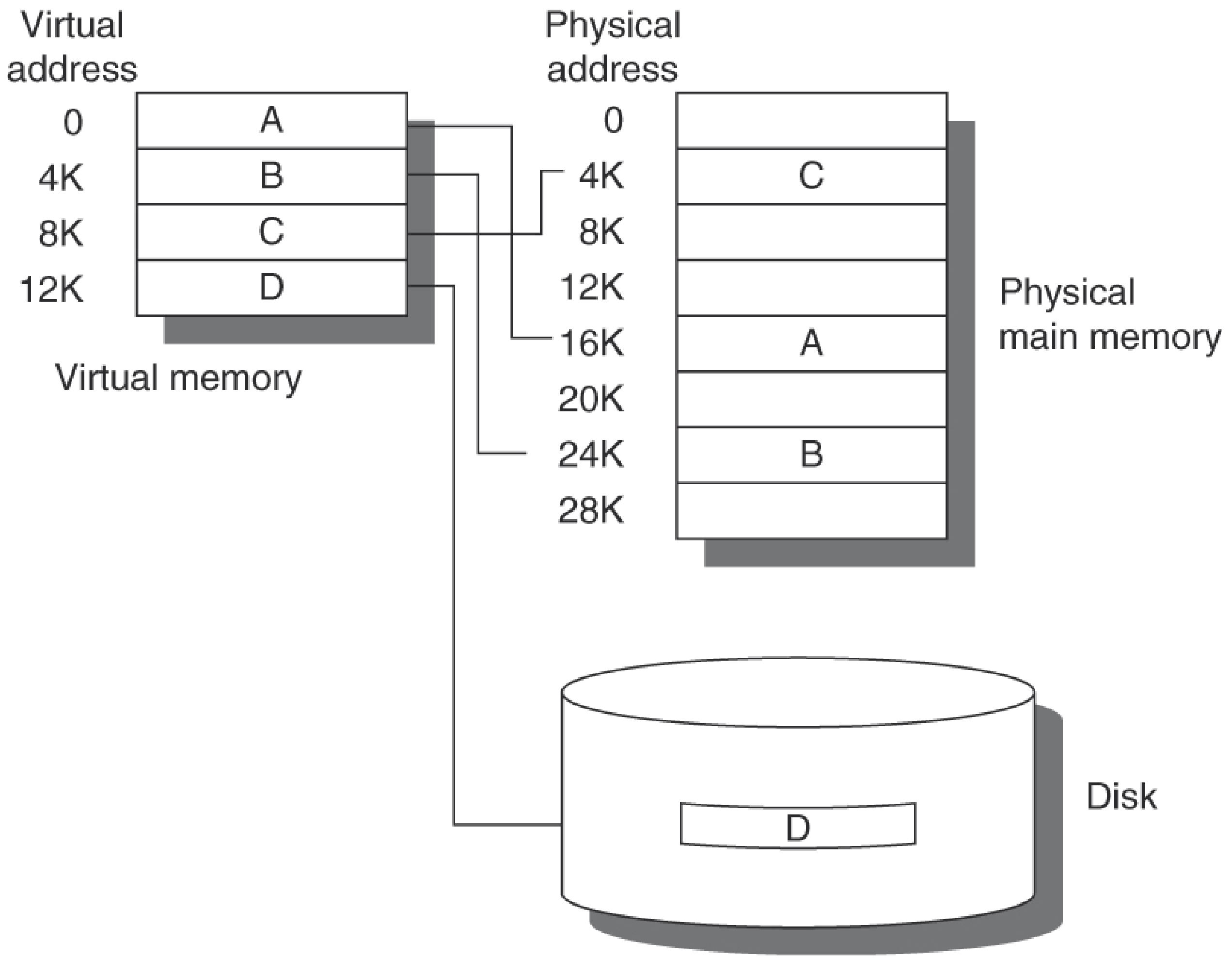
- Virtual address space, i.e., space addressable by a program is determined by ISA
- Main memory size ≤ disk size ≤ virtual address space size
- Virtual memory is organized in fixed-size (power of 2, typically at least 4 KB) blocks, called pages
- Physical memory is considered a collection of pages of the same size
- The unit of data transfer between disk and physical memory is a page
- Advantages of Virtual Memory:
  - Illusion of having more physical memory
  - Program reallocation
  - Protection

Virtual Memory

- Main Memory acts like a cache for the secondary memory (disk)
- Pages: virtual memory blocks

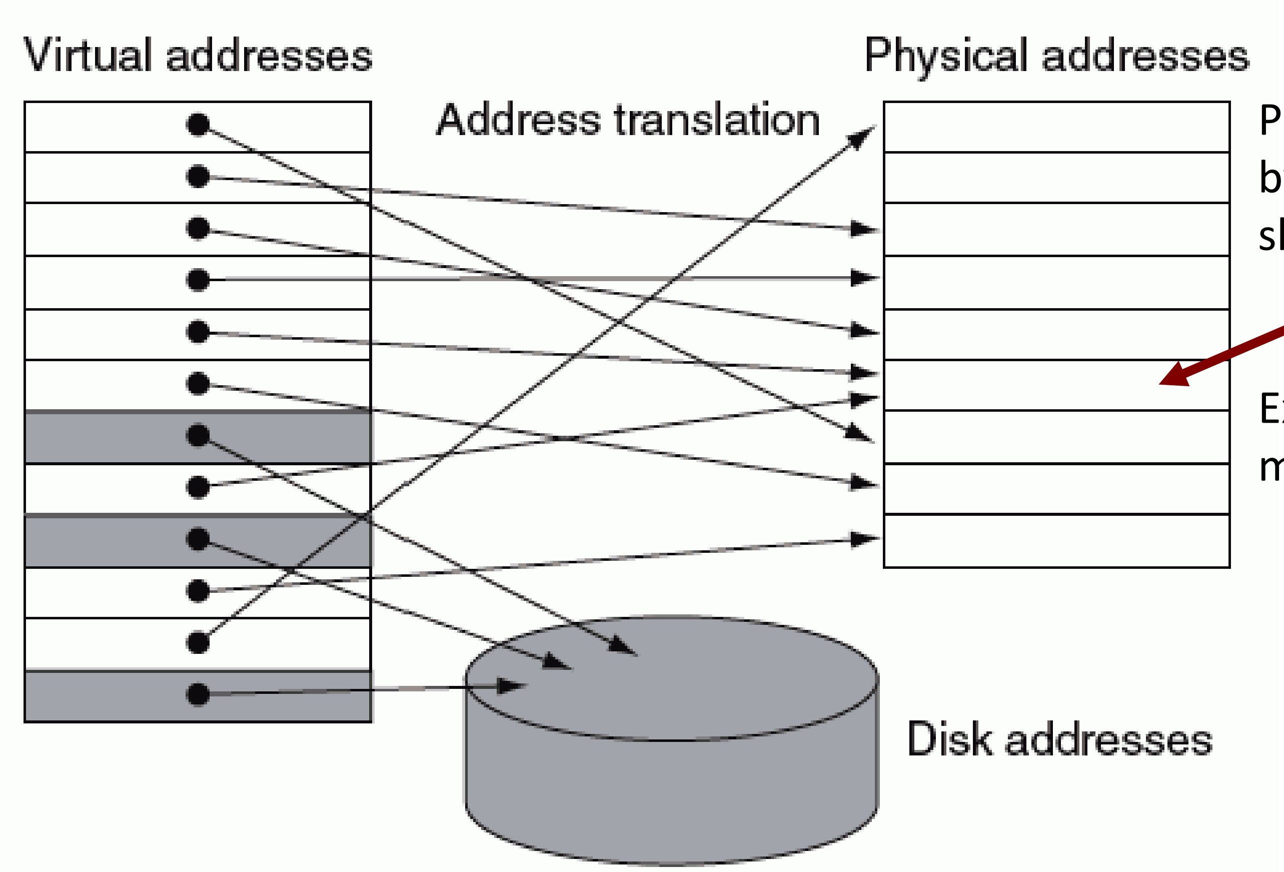
- Page faults
  - The data is not in main memory → retrieve it from disk
  - **Huge miss penalty**, thus pages should be fairly large (e.g., 4 KB)
  - Reducing page faults is important → **LRU** is worth the price
  - Can handle the faults in software instead of hardware
  - Overhead is small compared to the disk access time
  - Using write-through is too expensive so **write back is used**

Virtual Memory



The logical program – contiguous virtual address space: four pages A, B, C, and D. [1]

Virtual Memory



Physical page can be shared by 2 virtual addresses to share data or code.

Example: OS code shared by more programs

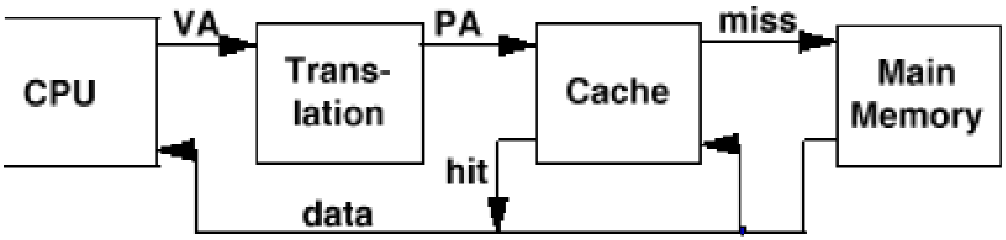
The actual location of the blocks is in physical main memory and on the disk [1]

Mapping of pages from a virtual address to a physical address or disk address:

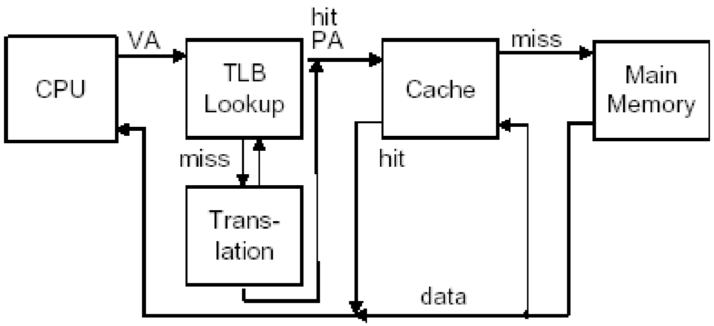
- Main memory acts as cache for secondary storage (disk)

Virtual Memory – Address Translation

- Translation from virtual address (VA) to physical address (PA)

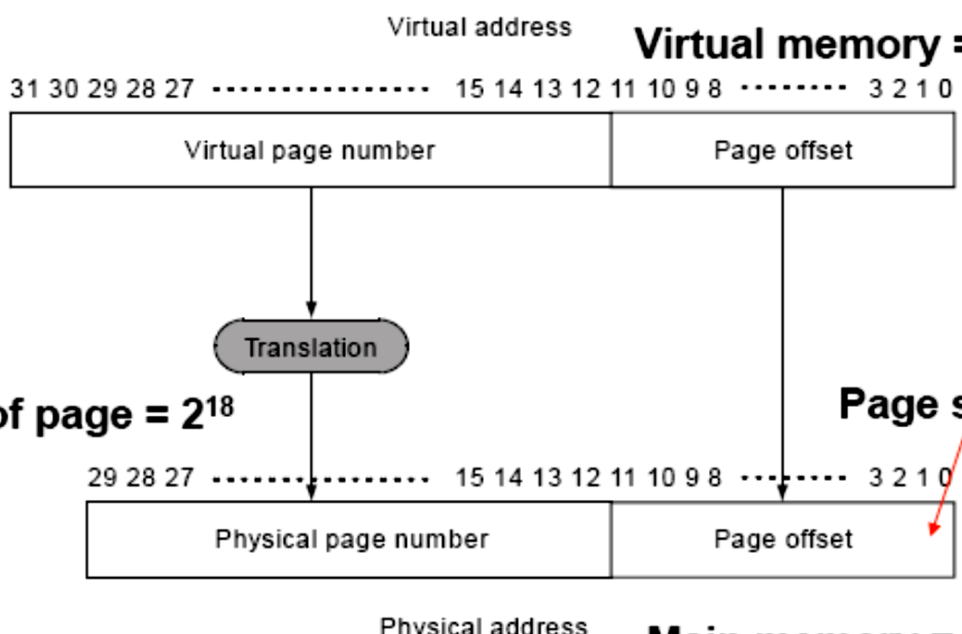


VA – PA translation with Page Tables



VA – PA translation with Page Tables + TLB

Virtual memory = 2^{32} = 4GB



The number of bits in the page offset field determine the page size (4 KB)

No. of page = 2^{18}

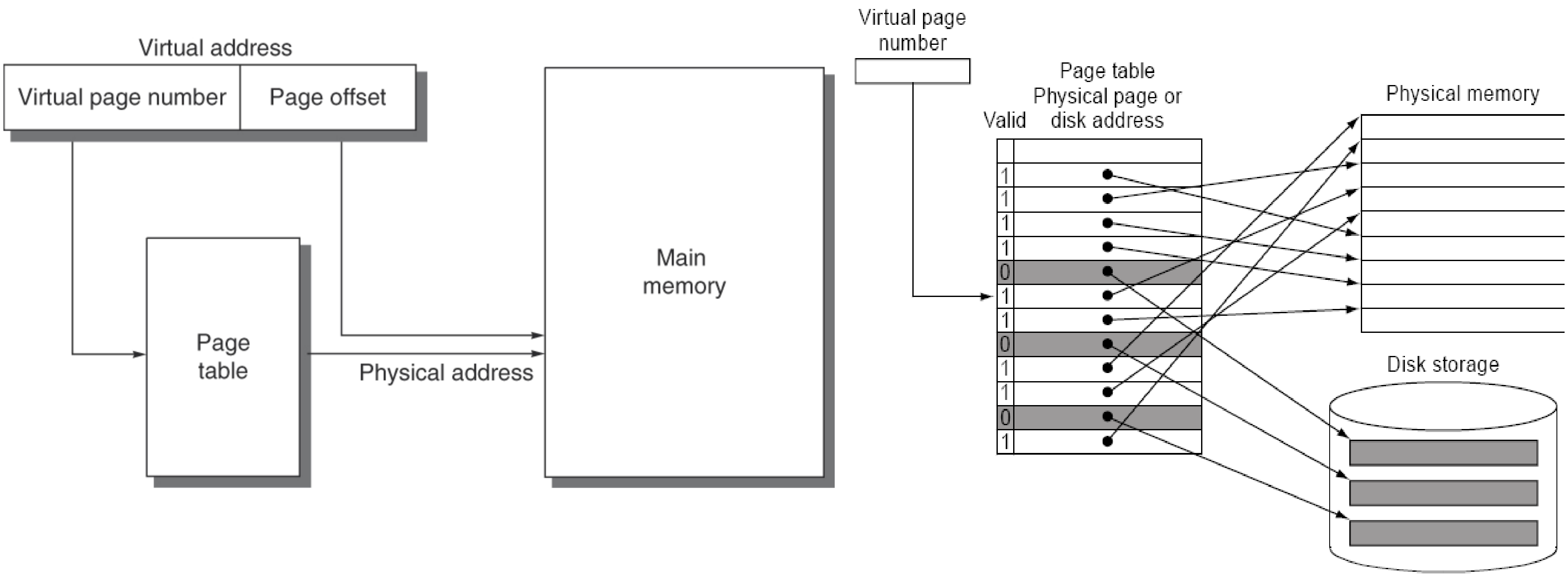
Page size = 2^{12} = 4KB

Usually, number of virtual pages > number of physical pages

[1]

Main memory = 2^{18+12} = 1GB

Virtual Memory – Page Table

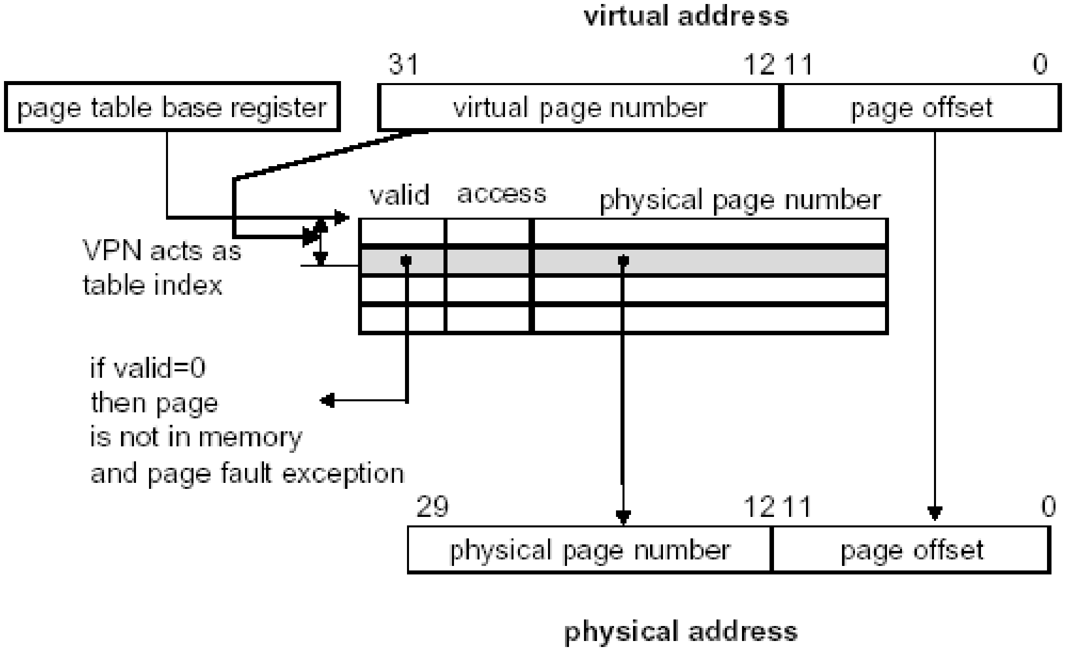


The mapping of a virtual address to a physical address via a page table [1]
Page table maps virtual page to either physical page or disk page

How Do You Place the Page and Find it Again?

- Locate pages by an index table: page table
- Each program has its own page table.
- A register (page table register) points to the start of the page table.
- The Page Table Implements Virtual to Physical Address Translation

Virtual Memory – Page Table



Address translation using the Page Table [1]
page size 4 KB, virtual address space 4 GB, physical memory 1 GB,
virtual page number = 20 bits, physical page number = 18 bits

To avoid large page table sizes:

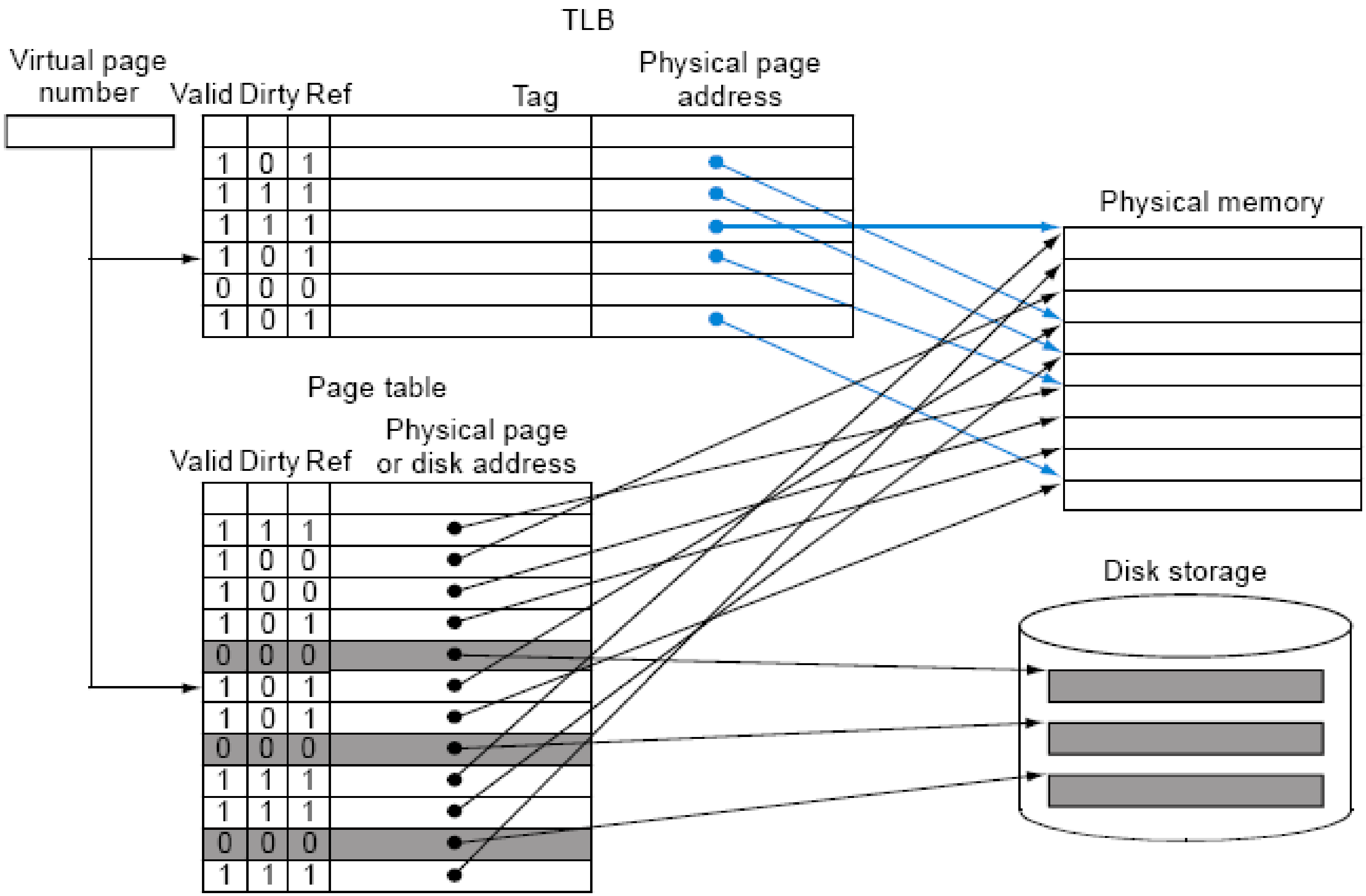
- Each program has its own page table.
- Page table register points to start of program's page table.
- Other techniques – multiple-level page tables, hashing virtual address, etc.

Virtual Memory

- Where to Place the Requested Pages? (in main memory)
  - If some pages are empty, use them
  - If all pages in main memory are in use, choose a page to replace it
  - **LRU replacement (least recently used)**
  - Replaced pages are written to swap space on the disk

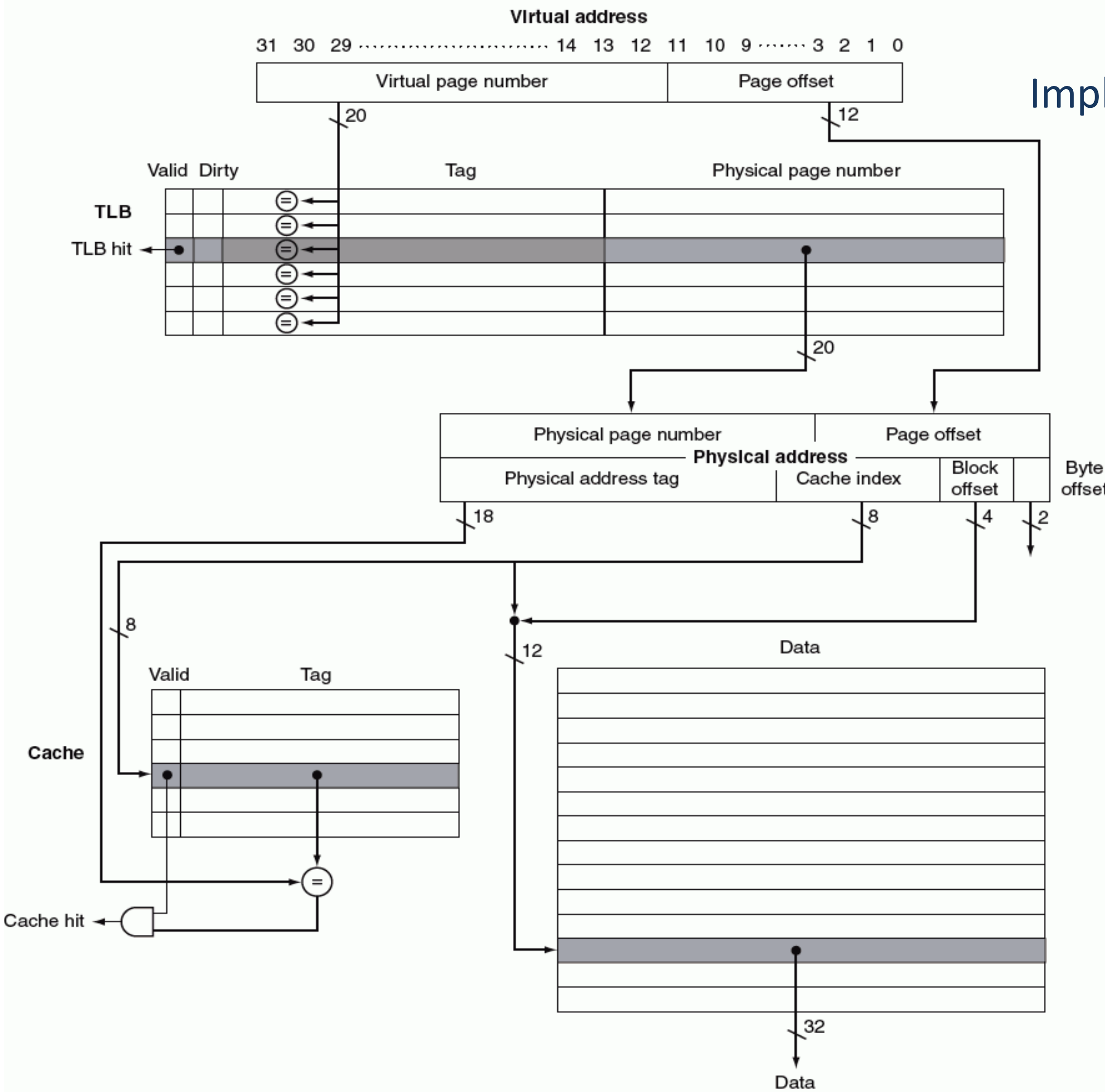
- Making Address Translation Fast: TLB (translation look aside buffer)
  - Address translation mechanism – Slow
  - Two cycle memory access, the page tables are stored in main memory.
  - One memory access to obtain the physical address,
  - Second access to get the data
  - TLB – cache for recently used Page Table Entries

Virtual Memory – TLB



Translation with TLB: TLB – fully associative cache [1]

Virtual Memory – TLB



Fully associative TLB [1]

Implemented as a direct mapped cache

Data read – 16 words in a block

On a page reference, look up the virtual page number in the TLB

If TLB hit:

- Get the physical page number
- Turn on the reference bit
- Turn on the dirty bit if write

If TLB miss:

- Look up the page table
- If miss again then true page fault

TLB, Typical values:

- 16 – 512 entries
- Miss-rate: 0.01% – 1%.
- Miss-penalty: 10 – 100 cycles

Problems – Homework

- A CPU generates 32-bit addresses for a byte addressable memory. Design an 8 KB cache memory for this CPU (8 KB is the cache size only for the data; it does not include the tag). The block size is 32 bytes. Show the block diagram, and the address decoding for
  - direct mapped cache memory
  - 4-way set associative cache memory

References

1. D. A. Patterson and J. L. Hennessy, “Computer Organization and Design: A Quantitative Approach”, 5th edition, ed. Morgan-Kaufmann, 2011.
2. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5th edition, ed. Morgan–Kaufmann, 2013.