نموذج رقم (2)

COURSE TITLE

Course Code	:30102315
Credit Hours	:3
Prerequisite	:30102214

Instructor Information

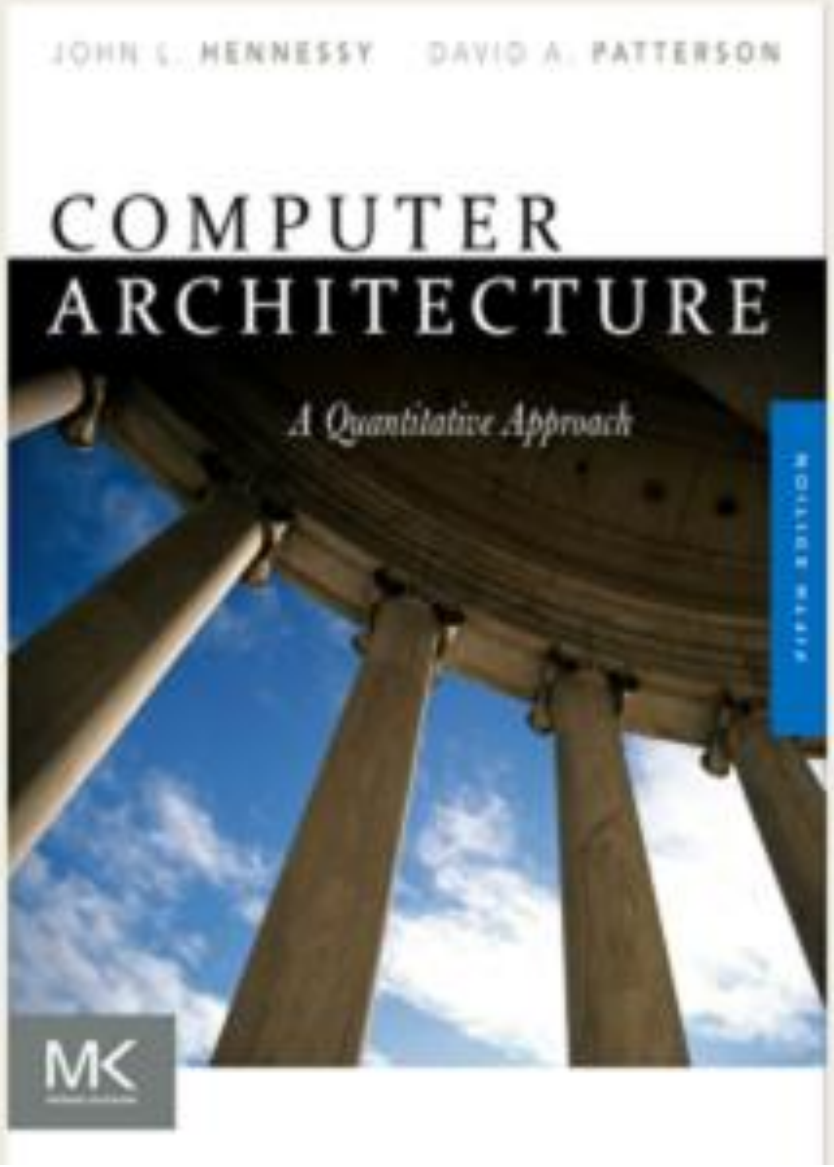
Name	: Dr. Rushdi Saleem Abu Zneit				
Office No.	**18, building 17 floor 3**				
Tel (Ext)	**Contact Telephone(Department Telephone)**				
E-mail	dr.rushdizneit@bau.edu.jo				
Office Hours	**Online**				
Class Times	Building	Day	Start Time	End Time	Room No.
		Sun,Tue, Thu	10:00	11:00	**Online**

Course description {From course plan}

Course Title; Computer Architecture
Credit Hour(3)

[Pre-req. 30102214

Textbook: Textbook Title


JOHN L. HENNESSY DAVID A. PATTERSON

COMPUTER ARCHITECTURE

A Quantitative Approach

FIFTH EDITION

MK

COURSE OBJECTIVES

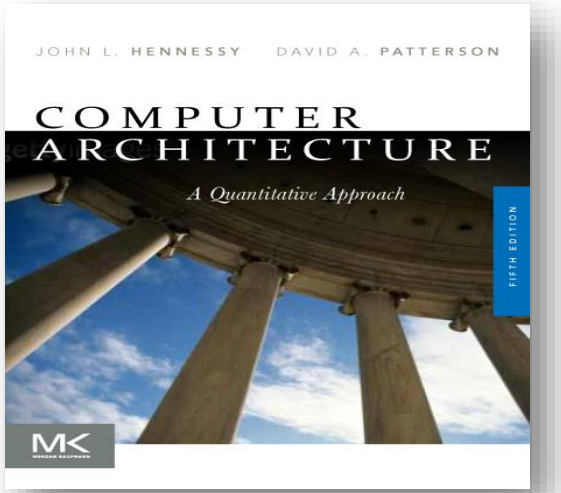
The module aims to provide students with a fundamental knowledge of computer hardware and computer systems, with an emphasis on system design and performance. The module concentrates on the principles underlying systems organization, issues in computer system design, and contrasting implementations of modern systems. The module is central to the aims of the Computing Systems degree course, for which it is core.

COURSE SYLLABUS

Week	Course Topic
Week 1	Course overview & Introduction
Week 2	Performance, Benchmarks, Measurements
Week 3	Pipelining Review
Week 4	Review: Instruction Set Design & Memory Hierarchy Design
Week 5	Memory Hierarchy Design
Week 6	Pipeline hazards, Instruction Re-scheduling
Week 7	ILP – Static: Loop Unrolling
Week 8	Midterm Exam
Week 9	Introduction to Branch Predictors
Week 10	Dynamic Instruction Level Parallelism: Scoreboarding Technique
Week 11	Dynamic Instruction Level Parallelism: Tomoslus' technique
Week 12	Data-Level Parallelism in Vector, SIMD, and GPU Architectures
Week 13	Data-Level Parallelism in Vector, SIMD, and GPU Architectures
Week 14	Thread-Level Parallelism
Week 15	Warehouse-Scale Computers to Exploit Request-Level and Data-Level Parallelism
Week 16	Final Exam

Course Topic
Introduction
Performance, Benchmarks, Measurements
Review: pipelining
Review: Instruction Set Design
Memory Hierarchy Design
Pipeline hazards, Instruction Re-scheduling
ILP – Static: Loop Unrolling
Midterm Exam
Introduction to Branch Predictors
Dynamic Instruction Level Parallelism: Scoreboarding Technique
Dynamic Instruction Level Parallelism: Tomoslus' technique
Data-Level Parallelism in Vector, SIMD, and GPU Architectures
Thread-Level Parallelism
Warehouse-Scale Computers to Exploit Request-Level and Data-Level Parallelism
Final Exam

Week 5



Chapter 2

Memory Hierarchy Design

Week 5

Introduction

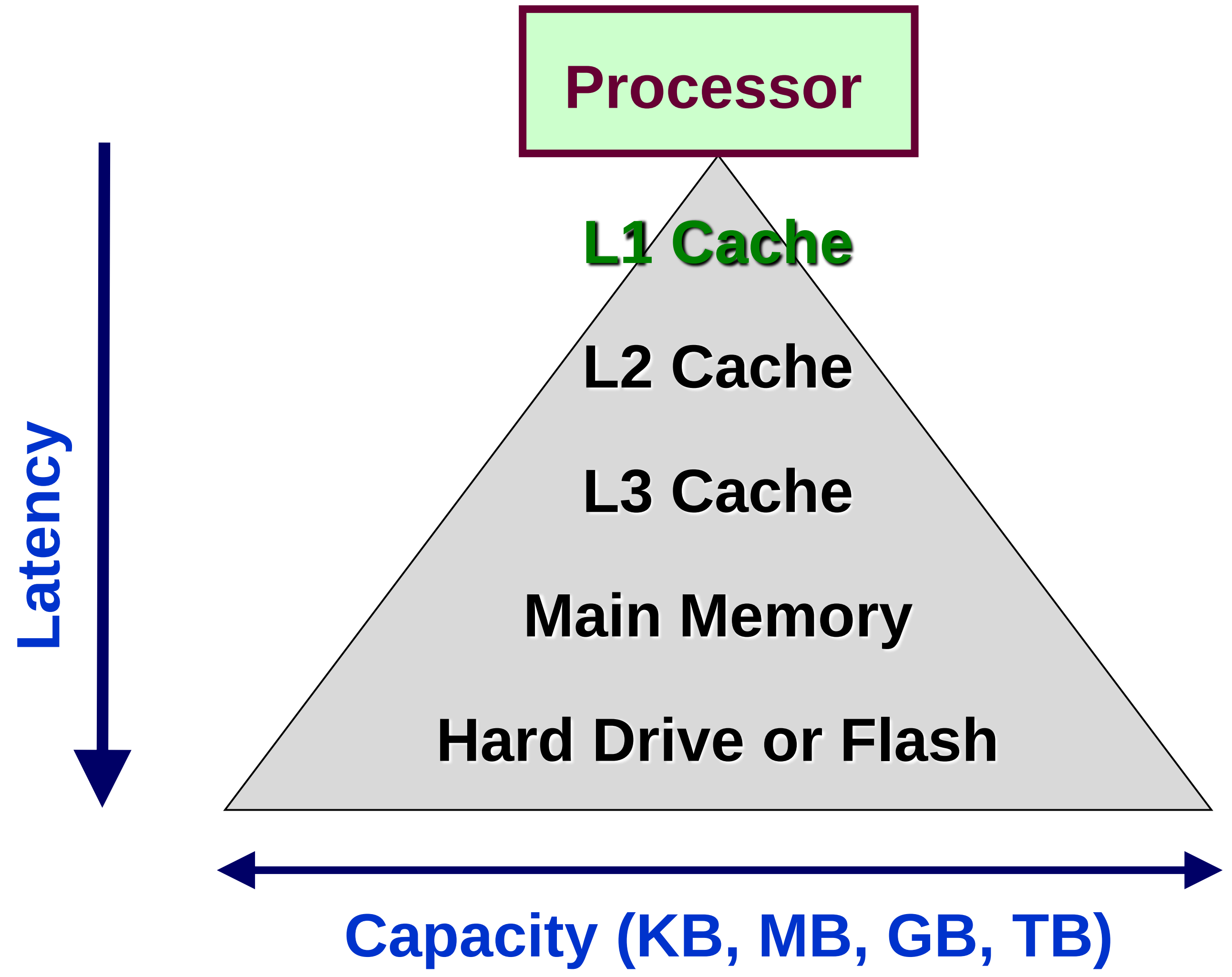
- Programmers want very large memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, **each containing a subset of the memory below it**, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

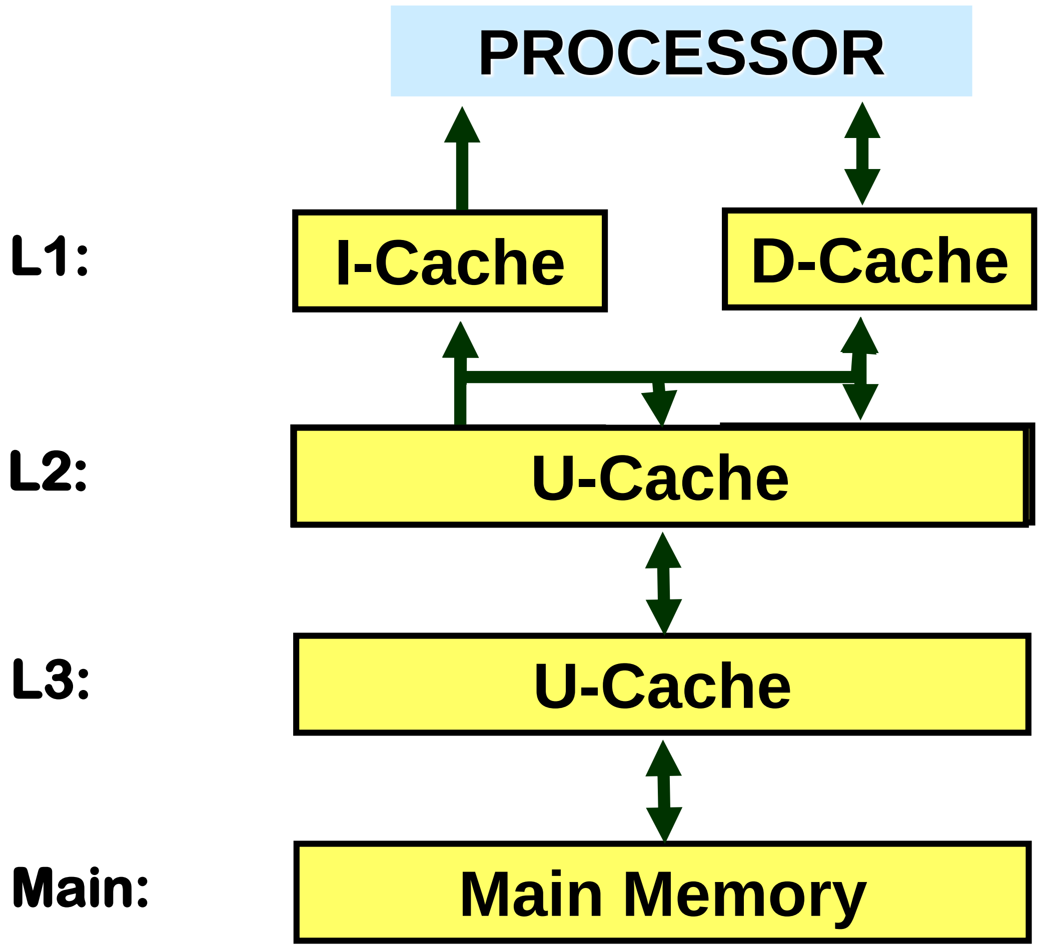
Temporal Locality :
Temporal Locality means that a instruction which is recently executed have high chances of execution again. So the instruction is kept in cache memory such that it can be fetched easily and takes no time in searching for the same instruction.

Spatial Locality :

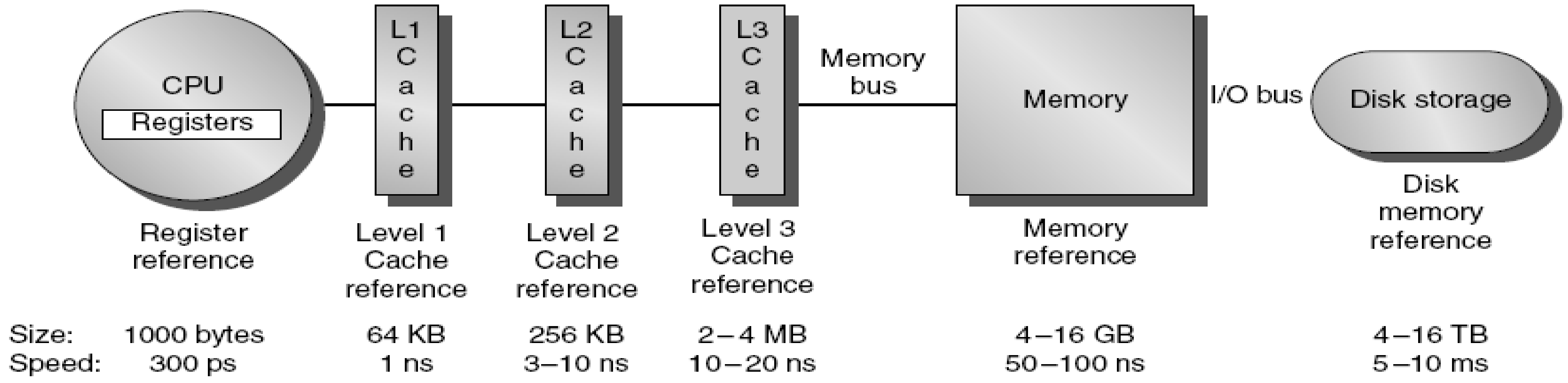
Spatial Locality means that all those instructions which are stored nearby to the recently executed instruction have high chances of execution. It refers to the use of data elements(instructions) which are relatively close in storage locations.

Memory Hierarchy

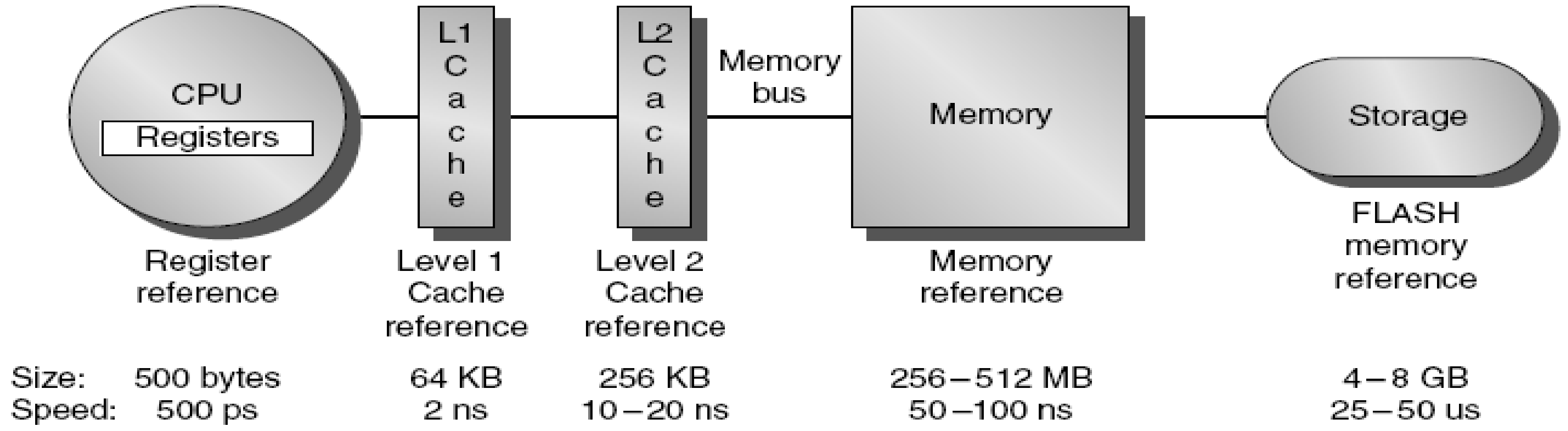



PROCESSOR
L1:
I-Cache
D-Cache
L2:
U-Cache
L3:
U-Cache
Main:
Main Memory

Memory Hierarchy

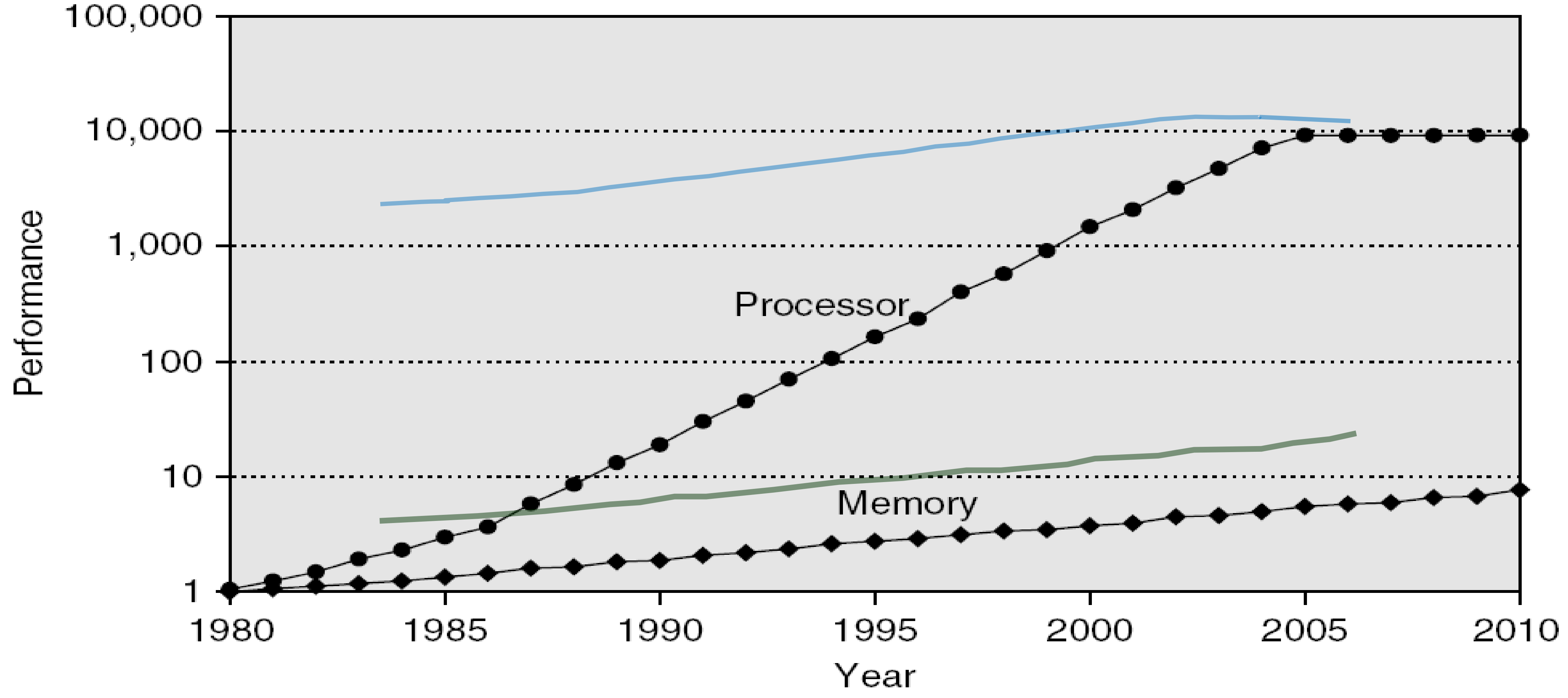


(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Memory Performance Gap



Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references
      - = 409.6 GB/s!
    - DRAM bandwidth is only 6% of this (25 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

Intel Processors (3rd Generation Intel Core)

- **Intel Core i7** *
  - 4 cores 8 threads
  - 2.5-3.5 GHz (Normal); 3.7 or 3.9GHz (Turbo)
- **Intel Core i5**
  - 4 cores 4 threads (or 2 cores 4 threads)
  - 2.3-3.4GHz (Normal); 3.2-3.8Ghz (Turbo)
- **Intel Core i3**
  - 2 cores 4 threads
  - 3.3 or 3.4 GHz

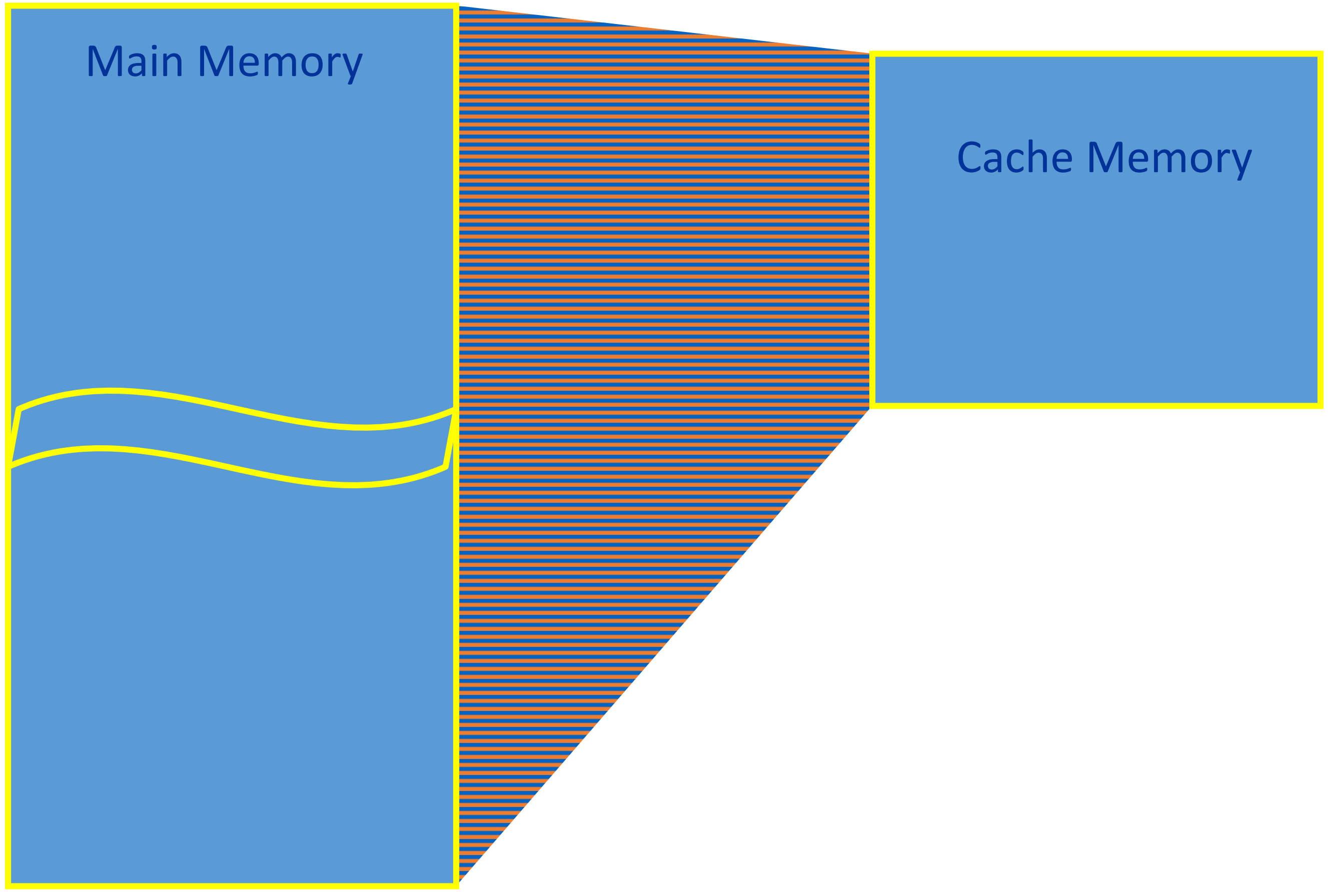
Performance and Power

- High-end microprocessors have >10 MB on-chip cache
  - Consumes large amount of area and power budget

Memory Hierarchy Basics

- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets

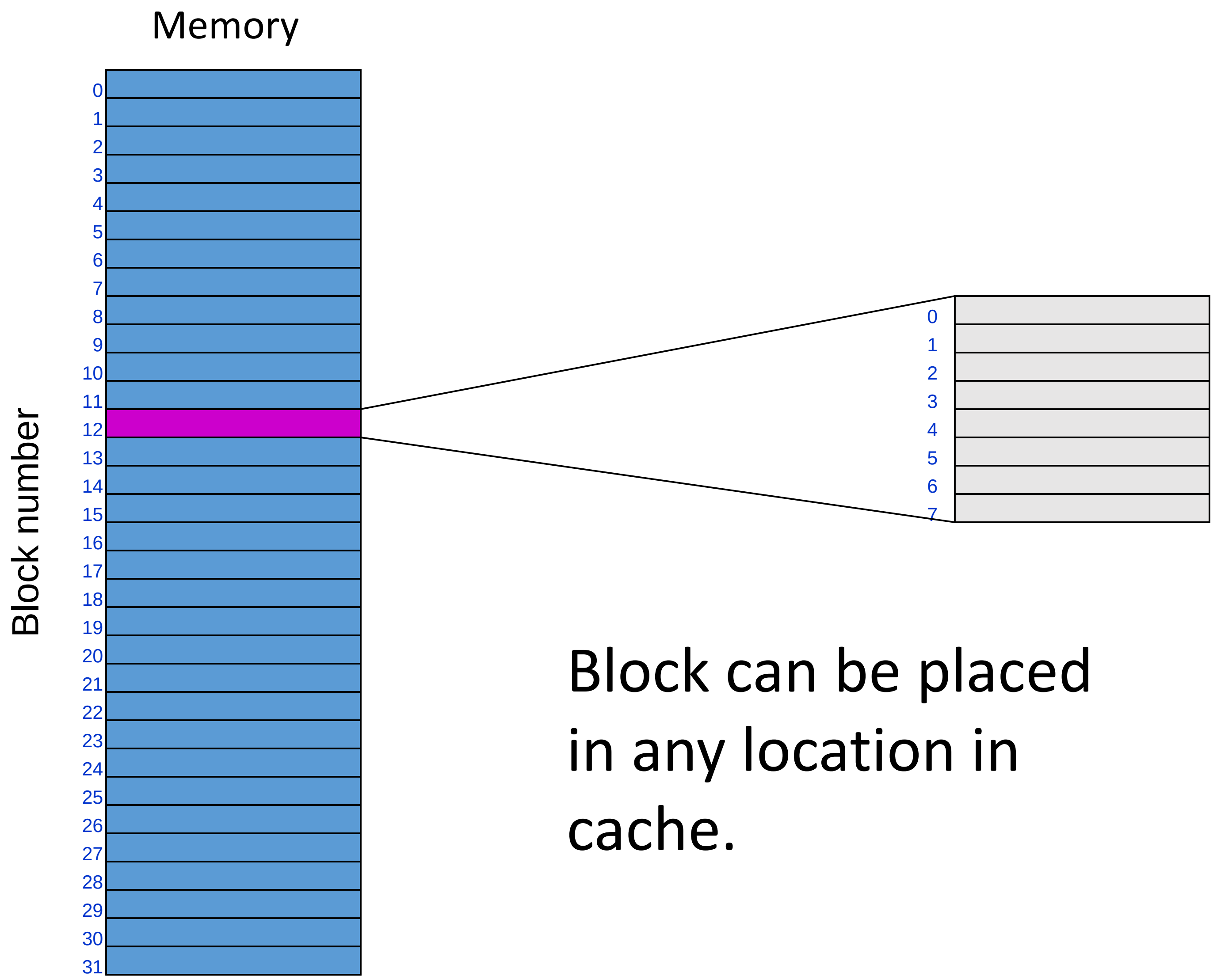
Placement Problem



Placement Policies

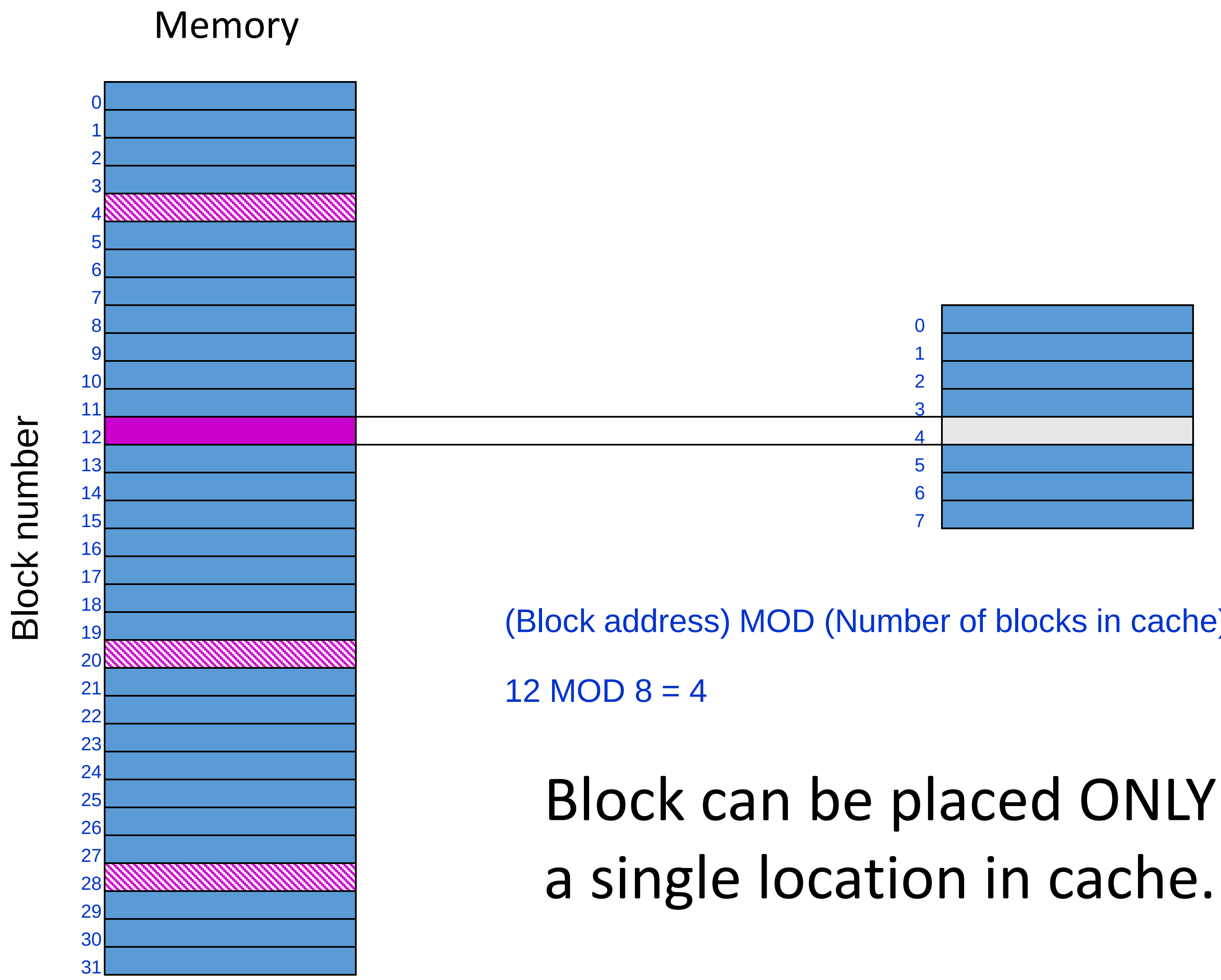
- WHERE to put a **<u>block</u>** in cache
- **Mapping** between main and cache memories.
- Main memory has a much larger capacity than cache memory.

Fully Associative Cache



Block can be placed in any location in cache.

Direct Mapped Cache

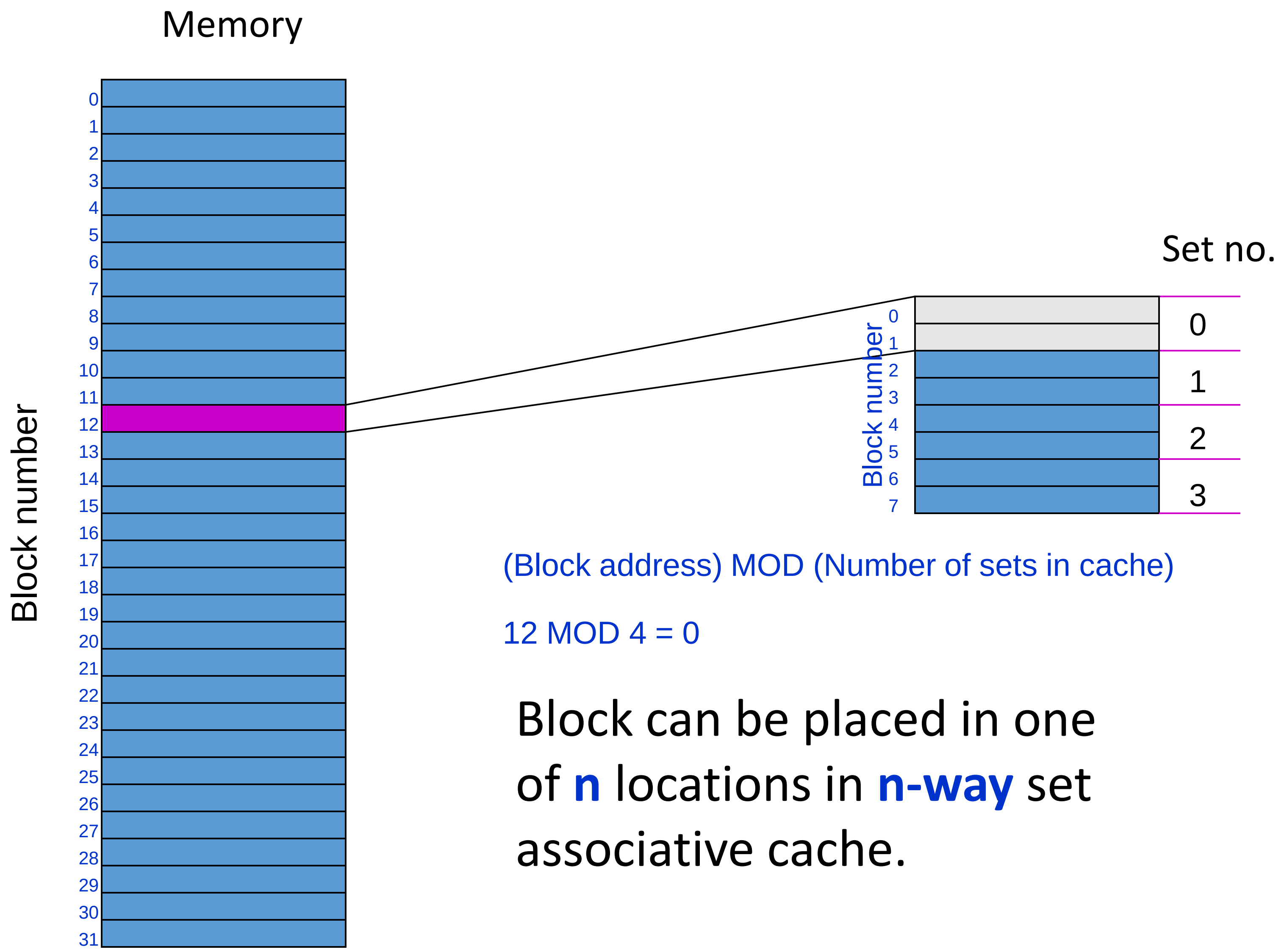


(Block address) MOD (Number of blocks in cache)

12 MOD 8 = 4

Block can be placed ONLY in a single location in cache.

Set Associative Cache



(Block address) MOD (Number of sets in cache)

12 MOD 4 = 0

Block can be placed in one of **n** locations in **n-way** set associative cache.

Memory Hierarchy Basics

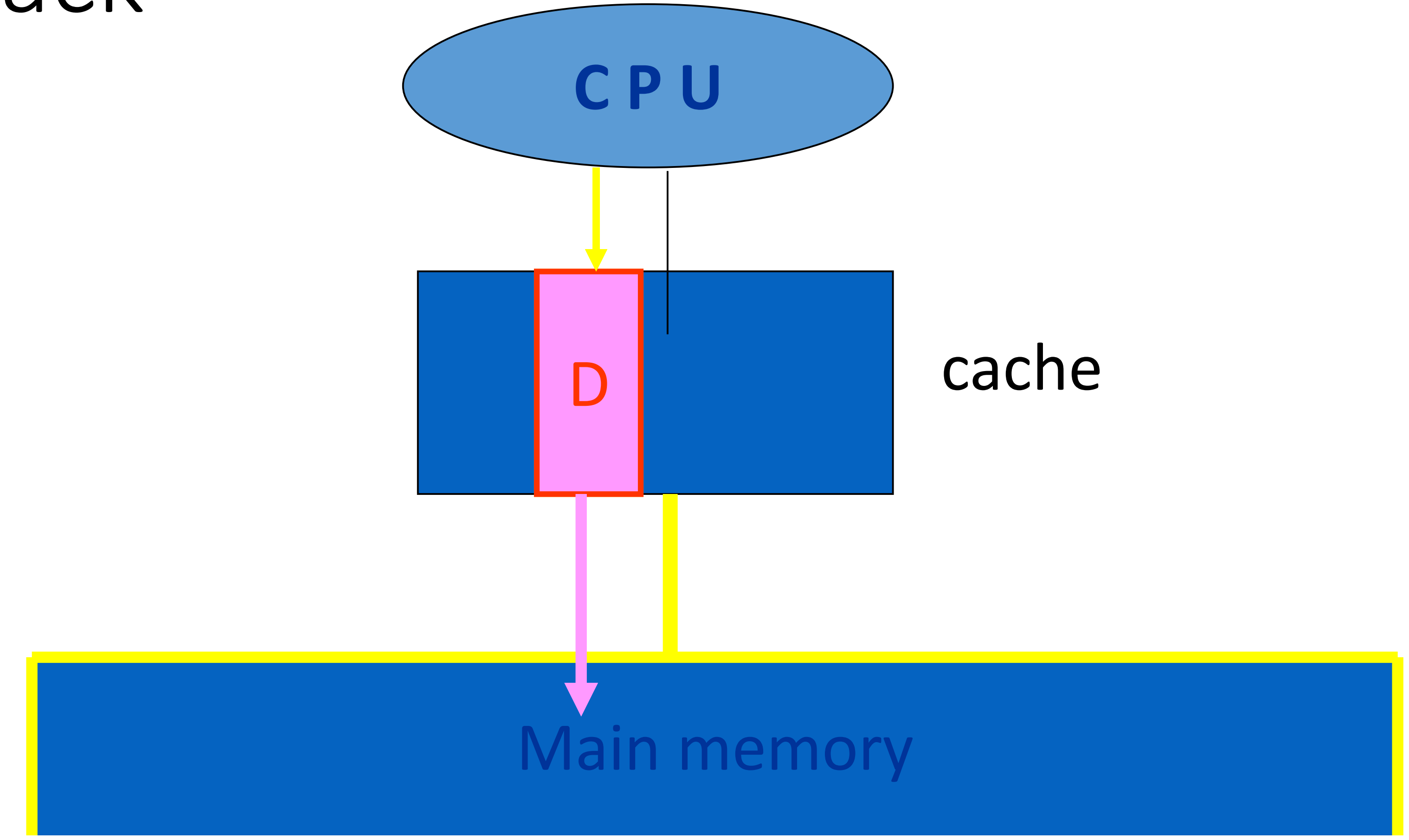
- *n* sets => *n-way set associative*
  - *Direct-mapped cache =>* one block per set
  - *Fully associative* => one set

- Writing to cache: two strategies
  - *Write-through*
    - Immediately update lower levels of hierarchy
  - *Write-back*
    - Only update lower levels of hierarchy when an updated block is replaced
  - Both strategies use *write buffer* to make writes asynchronous

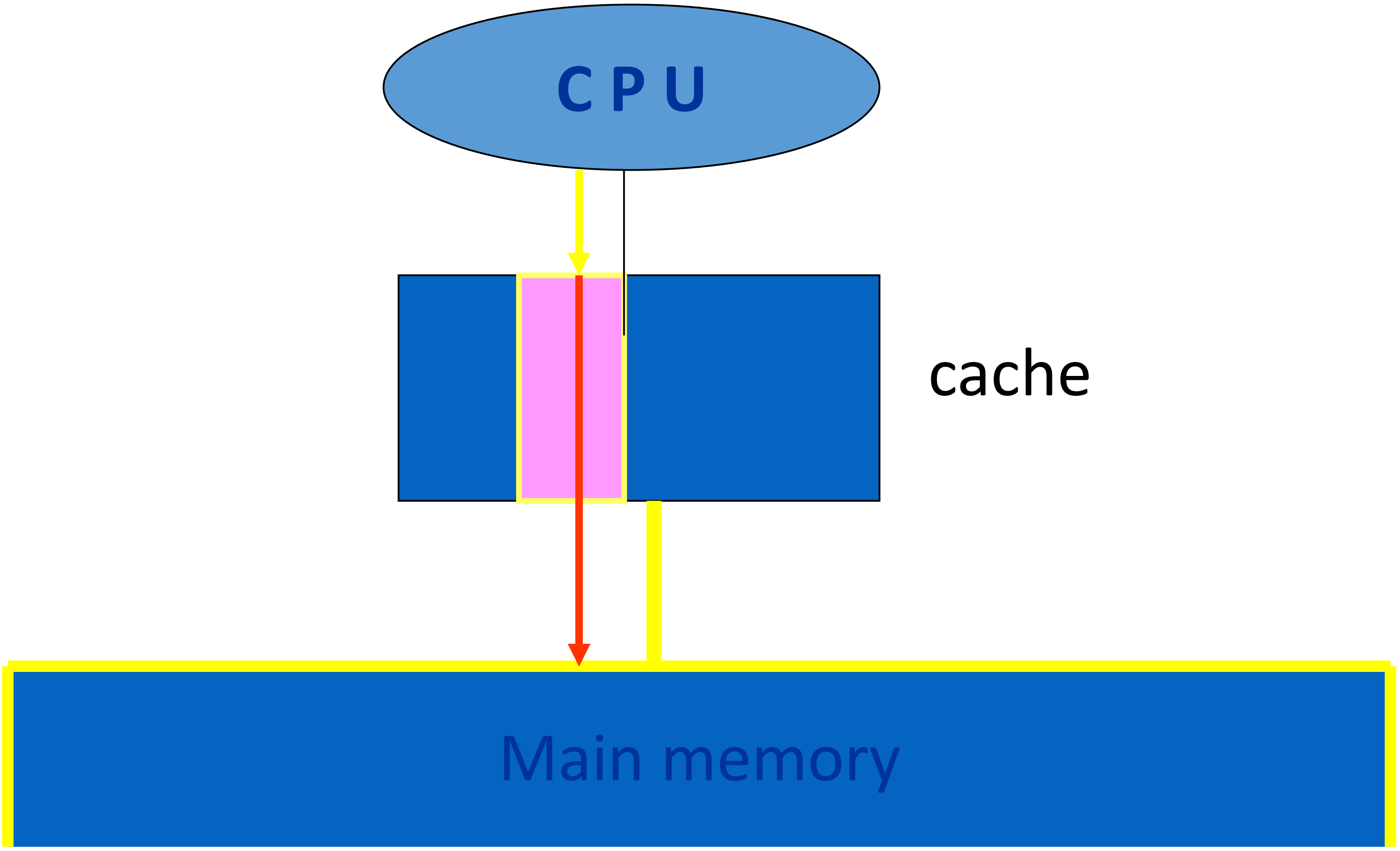
Dirty bit(s)

- Indicates if the block has been written to.
  - No need in I-caches.
  - No need in write through D-cache.
  - Write back D-cache needs it.

Write back



Write through

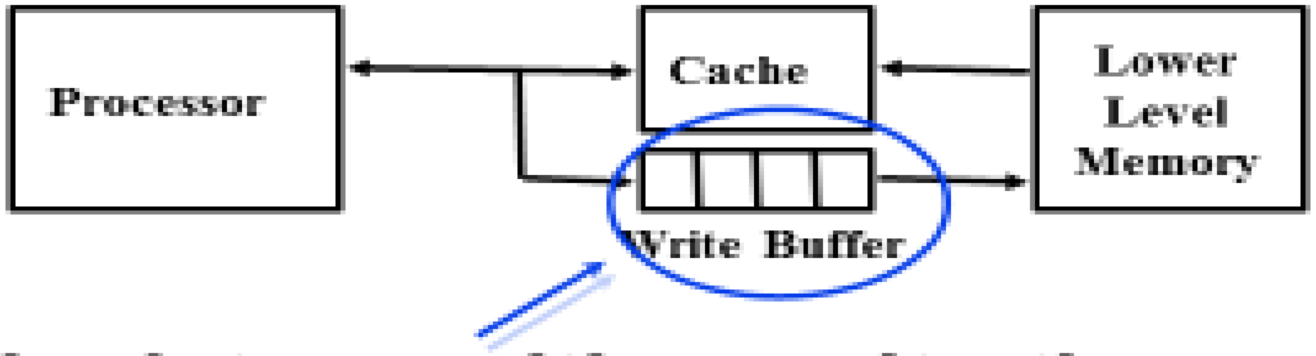


Q4: What happens on a write?

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option (on miss)-- let writes to an un-cached address; allocate a new cache line ("write-allocate").

Write Buffers for Write-Through Caches



Holds data awaiting write-through to lower level memory

Q. Why a write buffer ? A. So CPU doesn't stall

Q. Why a buffer, why not just one register ? A. Bursts of writes are common.

Q. Are Read After Write (RAW) hazards an issue for write buffer? A. Yes! Drain buffer before next read, or send read 1st after check write buffers.

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses (Three Cs)
  - **Compulsory**
    - First reference to a block
  - **Capacity**
    - Blocks discarded and later retrieved
  - **Conflict**
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Memory Hierarchy Basics

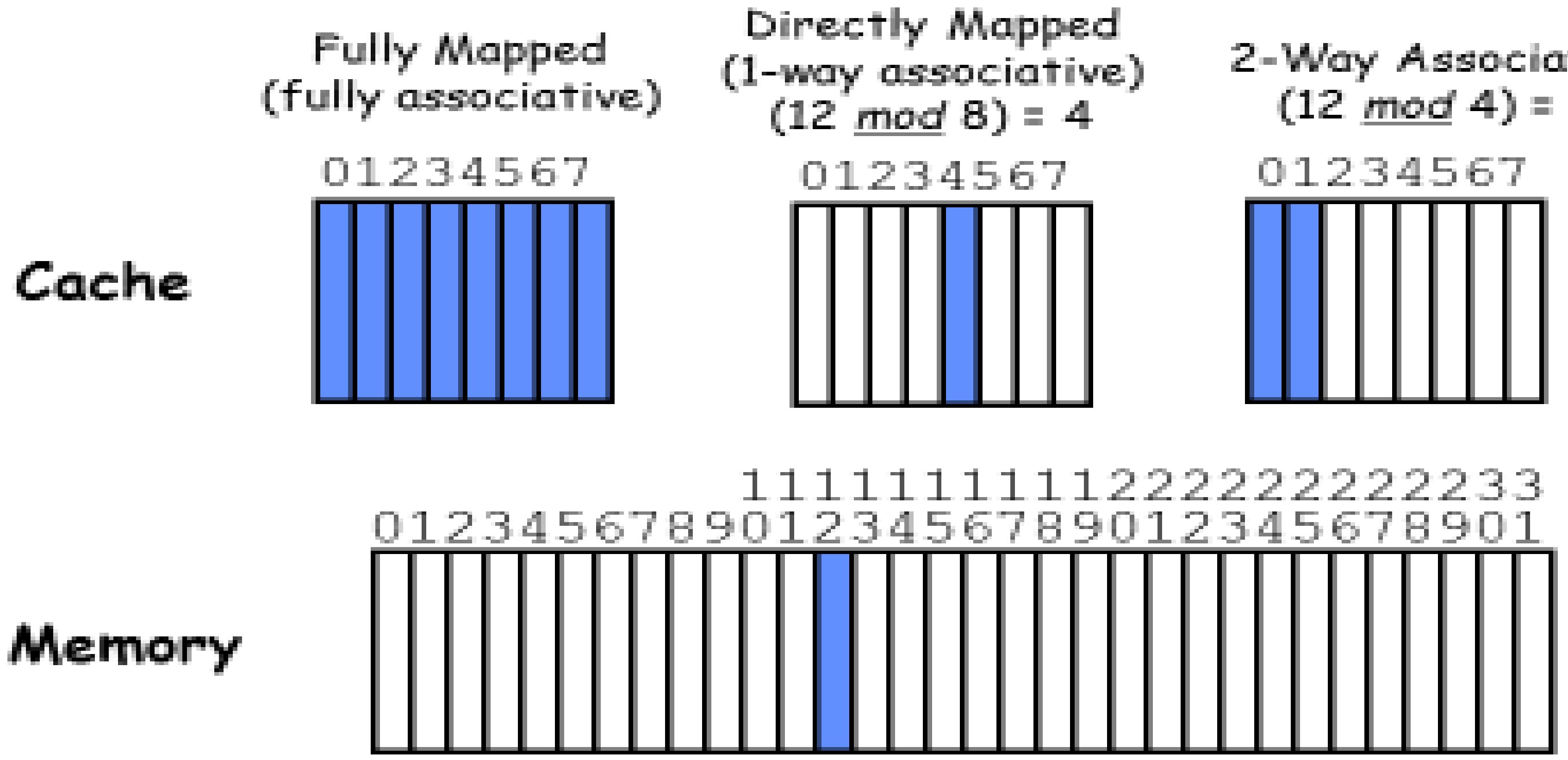
$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

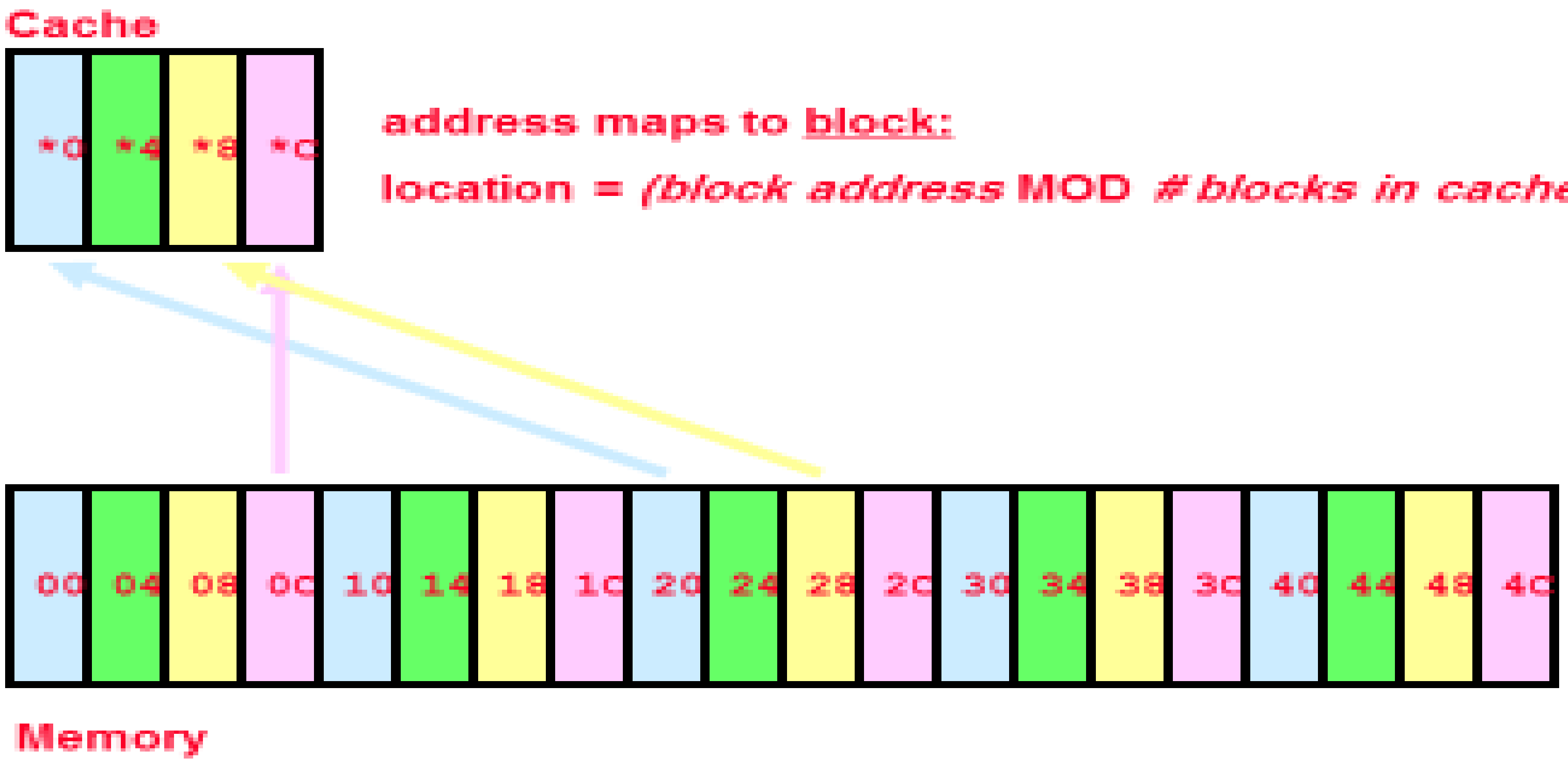
- Note that speculative and multithreaded processors may execute other instructions during a miss
  - Reduces performance impact of misses

Q1: Where can a block be placed in the upper level?

- **Block 12 placed in an 8-block cache:**
  - **Fully associative, direct mapped, 2-way set associative**
  - **S.A. Mapping = (Block Number) *Modulo* (Number Sets)**



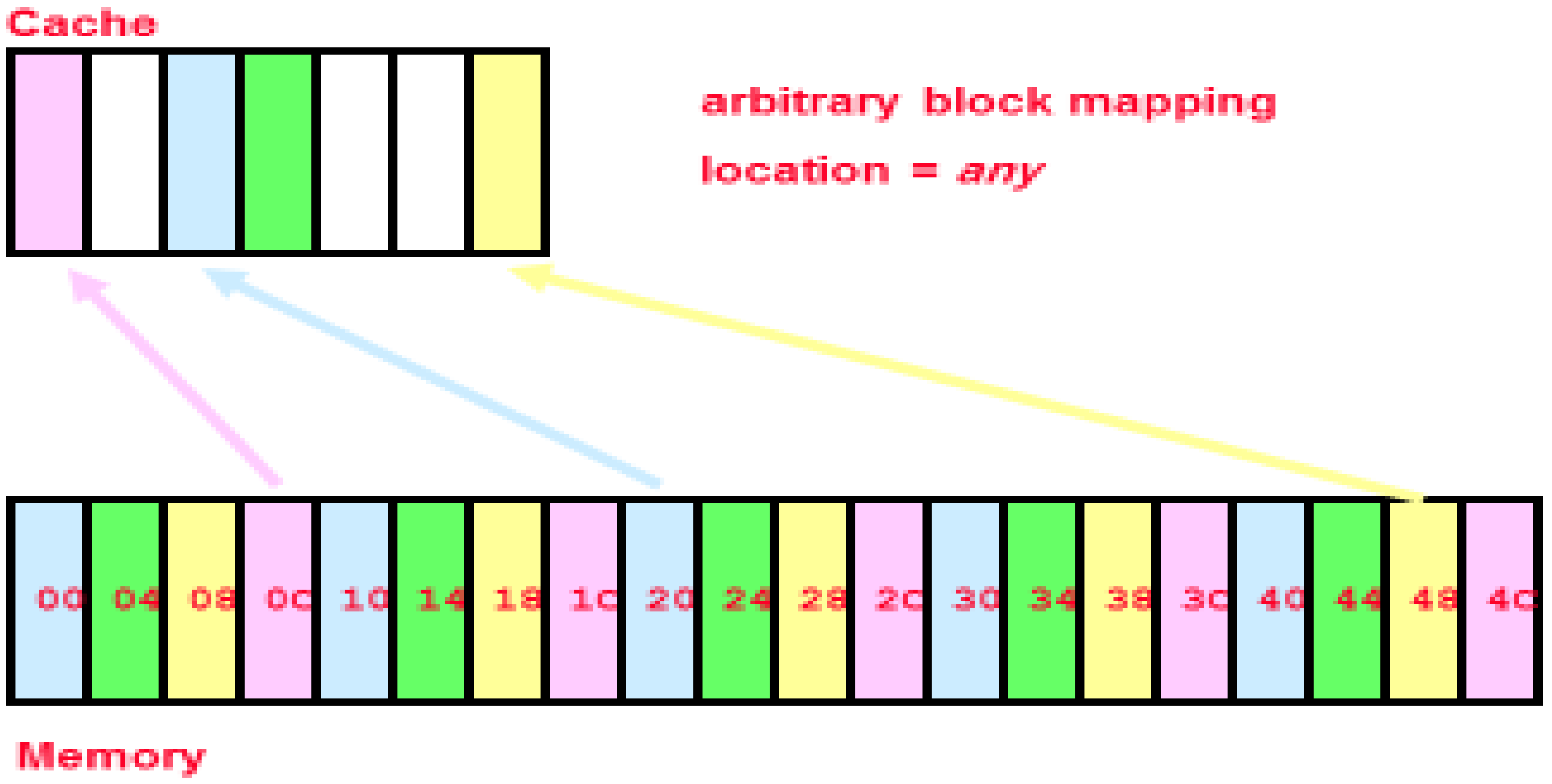
Direct Mapped Block Placement



address maps to <u>block:</u>

location = *(block address* MOD *# blocks in cache)*

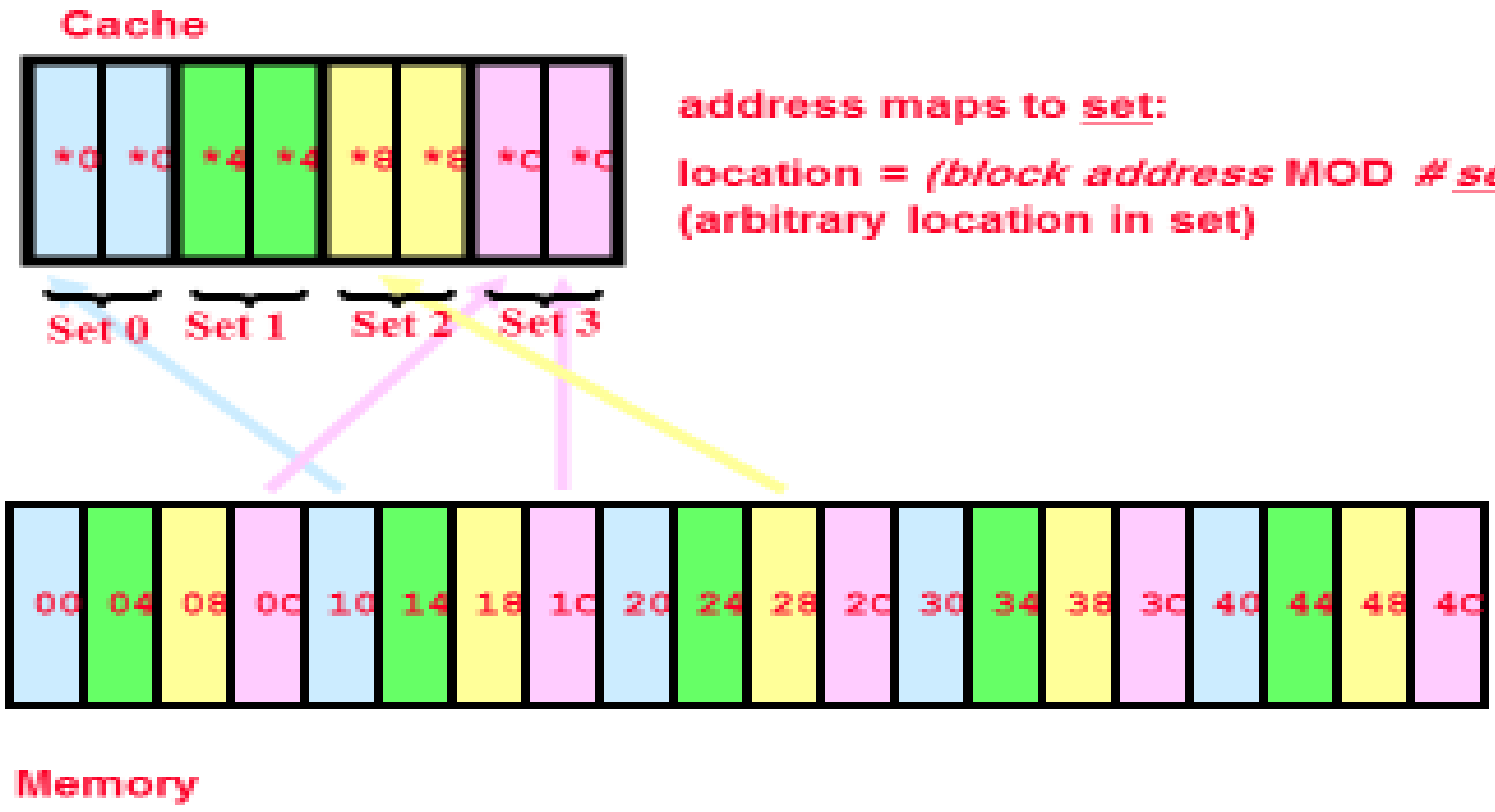
Fully Associative Block Placement



arbitrary block mapping

location = *any*

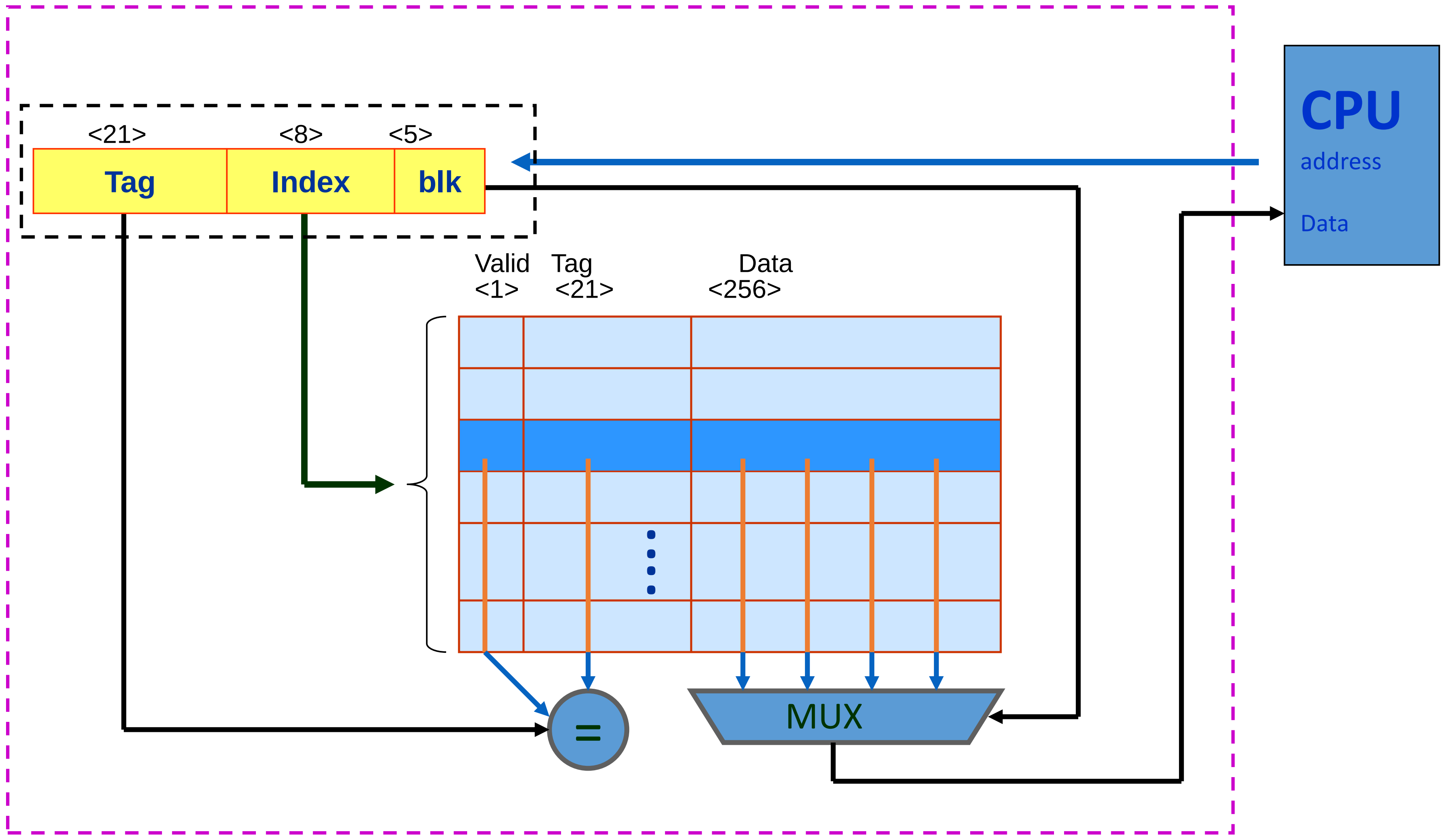
Set-Associative Block Placement



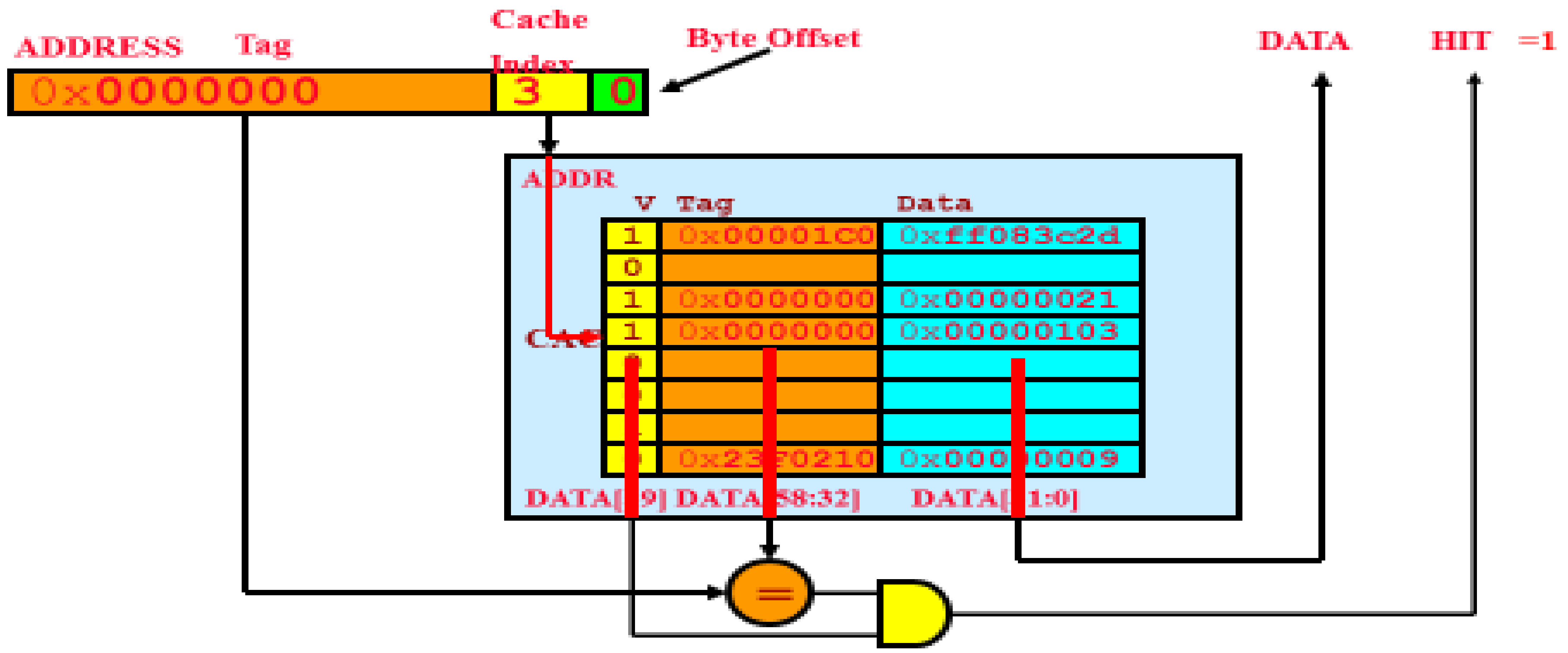
address maps to set:

location = *(block address* MOD *# sets in cache)*
(arbitrary location in set)

Cache organization Direct Mapp.



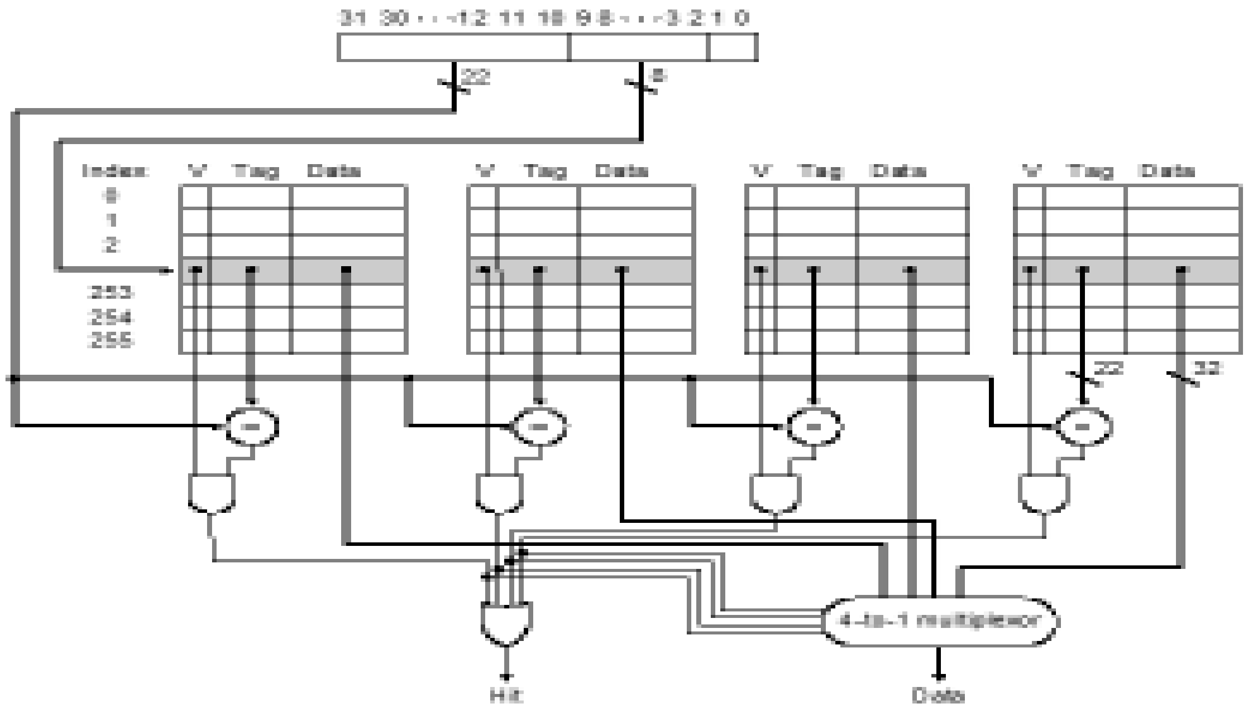
Direct-Mapped Cache Design



Four-way cache

Set Associative Cache Design

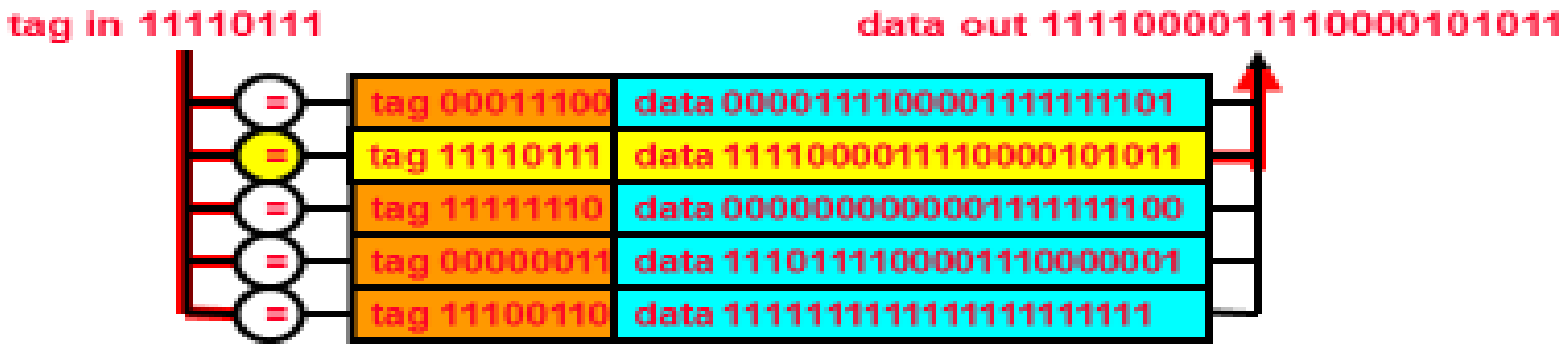
- **Key idea:**
  - Divide cache into sets
  - Allow block anywhere in a set
- **Advantages:**
  - Better hit rate
- **Disadvantage:**
  - More tag bits
  - More hardware
  - Higher access time



A Four-Way Set-Associative Cache

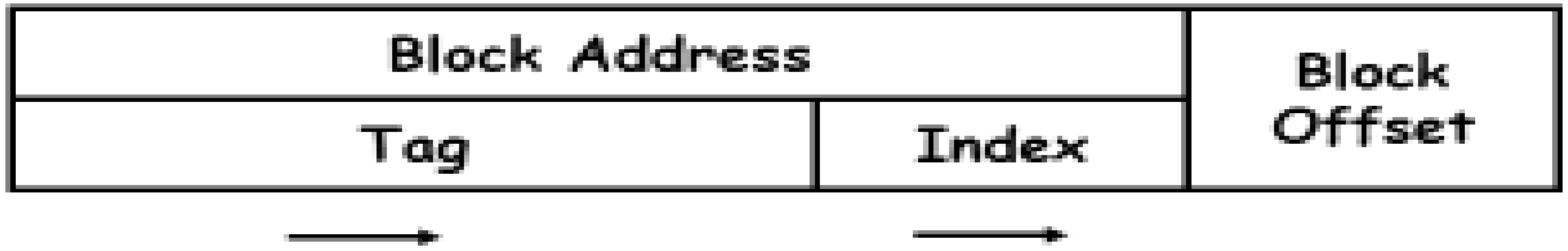
Fully Associative Cache Design

- Key idea: set size of one block
  - 1 comparator required for each block
  - No address decoding
  - Practical only for small caches due to hardware demands


tag in 11110111
data out 1111000011110000101011
=
tag 00011100
data 0000111100001111111101
=
tag 11110111
data 1111000011110000101011
=
tag 11111110
data 0000000000001111111100
=
tag 00000011
data 1110111100001110000001
=
tag 11100110
data 1111111111111111111111

Q2: How is a block found if it is in the upper level?

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag



In-Class Exercise

- Given the following requirements for cache design for a 32-bit-address computer: (1) cache contains 16KB of data, and (2) each cache block contains 16 words. (3) Placement policy is 4-way set-associative.
  - What are the lengths (in bits) of the *block offset field* and the *index field* in the address?
  - What are the lengths (in bits) of the *index field* and the *tag field* in the address if the placement is 1-way set-associative?

Solution

Cache size=16KB=2^{10}x2^4=2^{14}Bytes

Block size=16 words=4*16=$2^2 x 2^4$=2^6 $Bytes$ (word length 32 bit, 4 bytes)

Number of blocks in cahe=$\frac{2^{14}}{2^6}$=2^8=256 blocks

4-Way set associative cache then each set has=$\frac{2^8}{2^2} = 2^6$ blocks

Number of bits in offset=$lg2^6 = 6\ bits$

Number of bits in index field=$lg2^6$=6 bits

Number of bits in tag field=32-6-6=20 bits

<u>In one set:</u>

Number of block from above =256 block

Number of bits in index=$lg2^8 = 8\ bits$

Number of bits in tag field=32-8=24 bits

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q3: After a cache read miss, if there are no empty cache blocks, which block should be removed from the cache?

The Least Recently Used (LRU) block? Appealing, but hard to implement for high associativity

A randomly chosen block? Easy to implement, how well does it work?

Miss Rate for 2-way Set Associative Cache

Size	Random	LRU
16 KB	5.7%	5.2%
64 KB	2.0%	1.9%
256 KB	1.17%	1.15%

Also, try other LRU approx.

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

Average memory access time = Hit time + Miss rate × Miss penalty

Ex: miss rate=0.05, miss penalty=10CC, hit penalty=3CC

AMAT=(1-0.05)*3+0.05*10=0.95*3+0.05*10=2.85+0.5=3.35CC

If the frequency of CPU=2.5GHz

AMAT=3.35*1/(2.5*10^9)=3.35*0.4*10^-9=1.34*10^-9=1.34ns

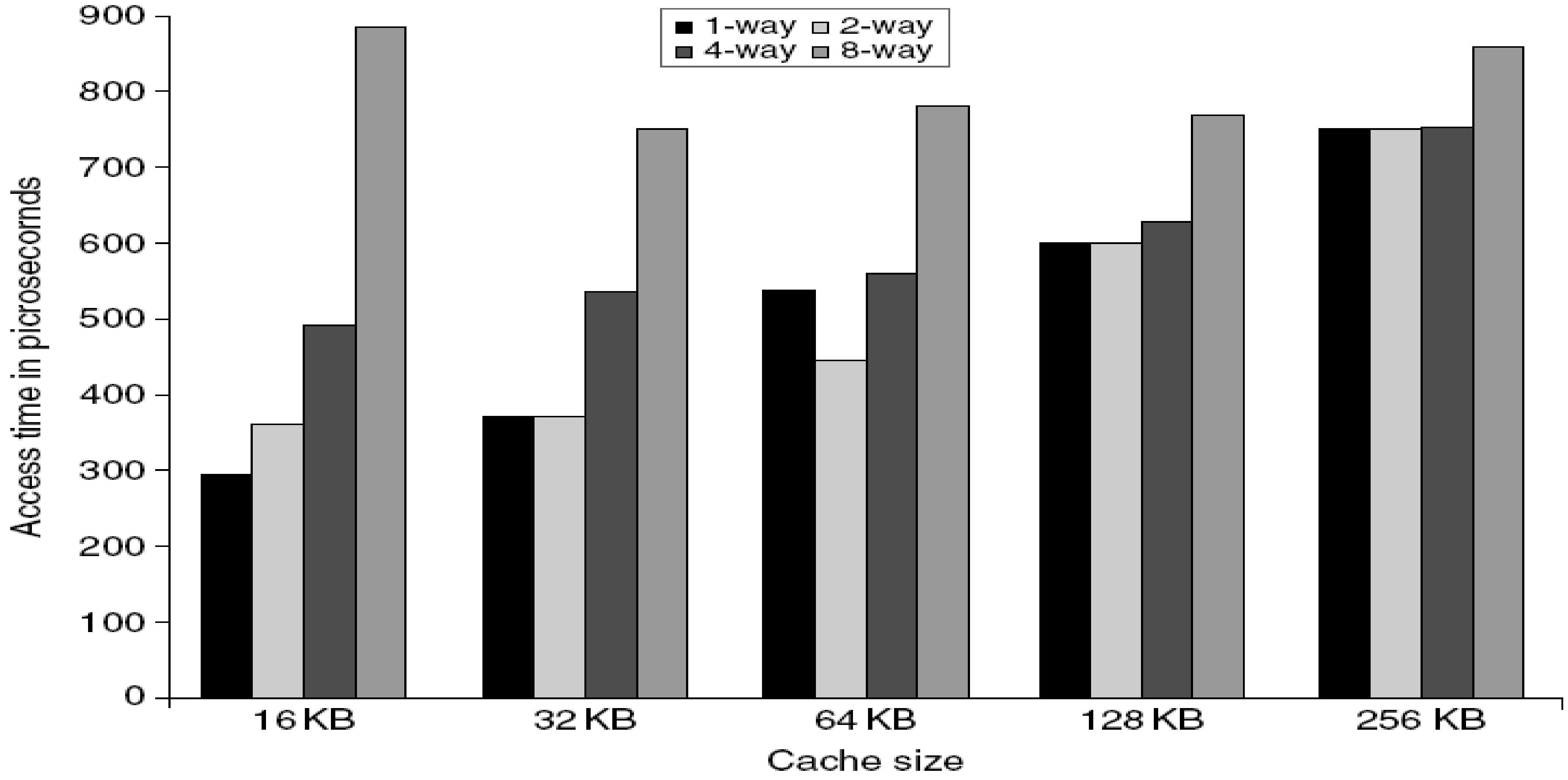
Ten Advanced Optimizations

- Metrics:
  - Reducing the hit time
  - Increase cache bandwidth
  - Reducing miss penalty
  - Reducing miss rate
  - Reducing miss penalty or miss rate via parallelism

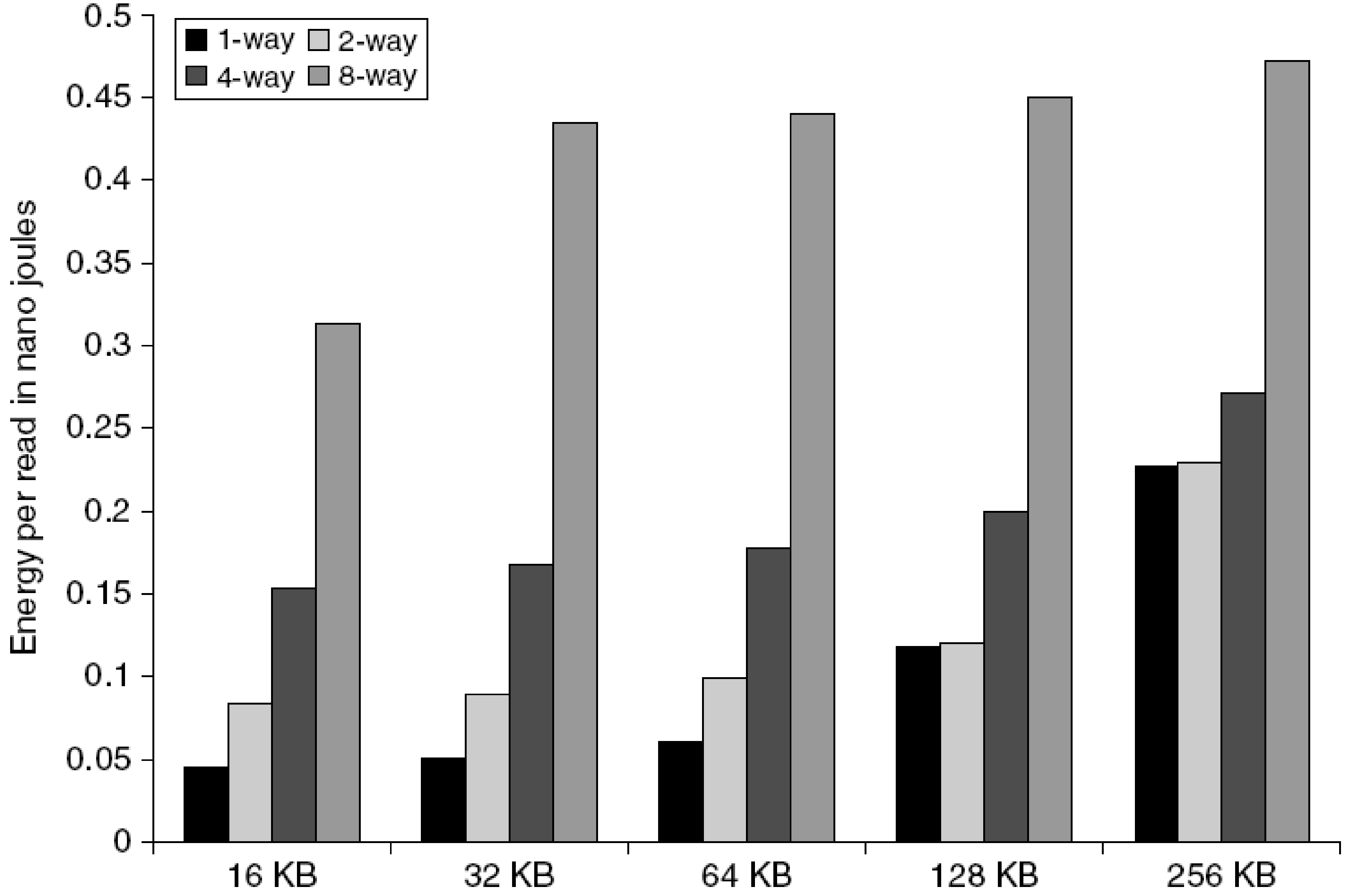
1) Small and simple L1 caches

- Critical timing path:
  - addressing tag memory, then
  - comparing tags, then
  - selecting correct set
- Direct-mapped caches can overlap tag compare and transmission of data
- Lower associativity reduces power because fewer cache lines are accessed

L1 Size and Associativity



L1 Size and Associativity


Energy per read in nano joules
1-way
2-way
4-way
8-way
16 KB
32 KB
64 KB
128 KB
256 KB