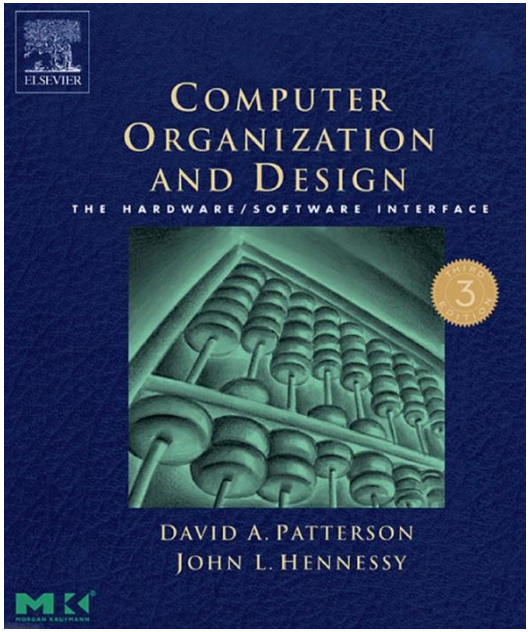
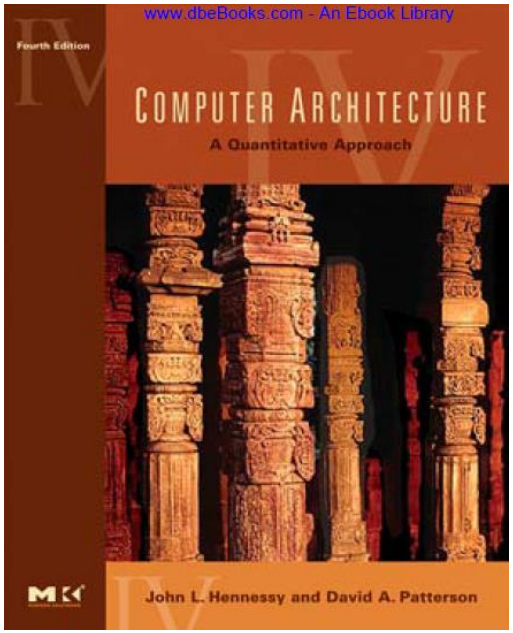
Memory Hierarchies

prepared and Instructed by
Shmuel Wimer
Eng. Faculty, Bar-Ilan University





Amdahl's Law

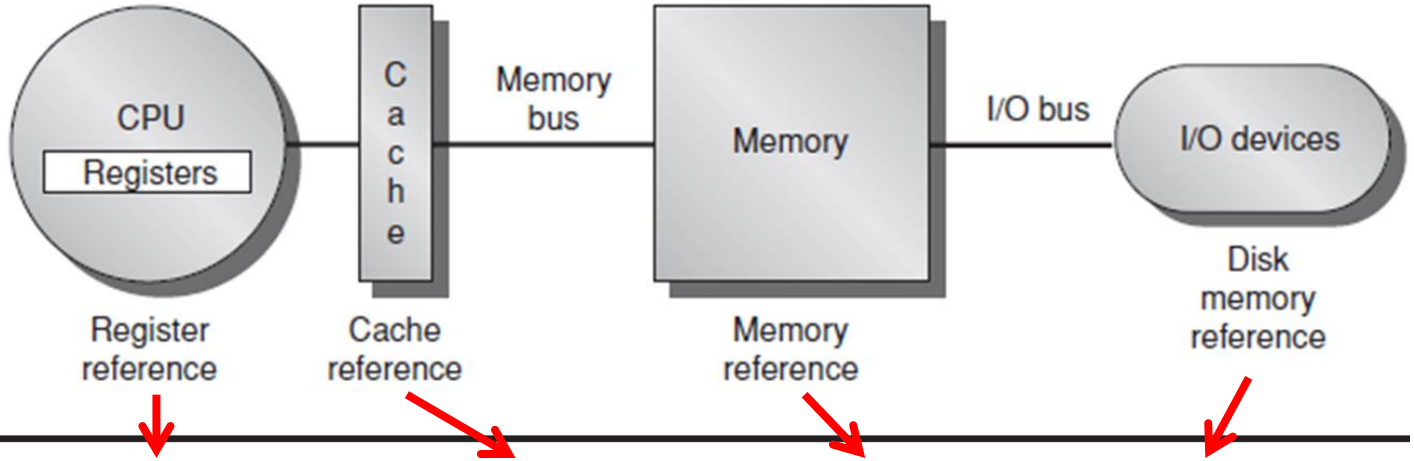
Speedup: How much faster a task will run on the computer with an enhancement, compared to the original computer.

Amdahl's Law: The performance improvement gained from using a faster mode of execution is limited by the fraction of the time the faster mode can be used.

Principle of Locality

- **Temporal locality** (locality in time): If an item is referenced, it will tend to be referenced again soon.
- **Spatial locality** (locality in space): If an item is referenced, items whose addresses are close will tend to be referenced soon.
- locality in programs
  - loops - temporal
  - instructions are usually accessed sequentially - spatial
  - Data access of array - spatial

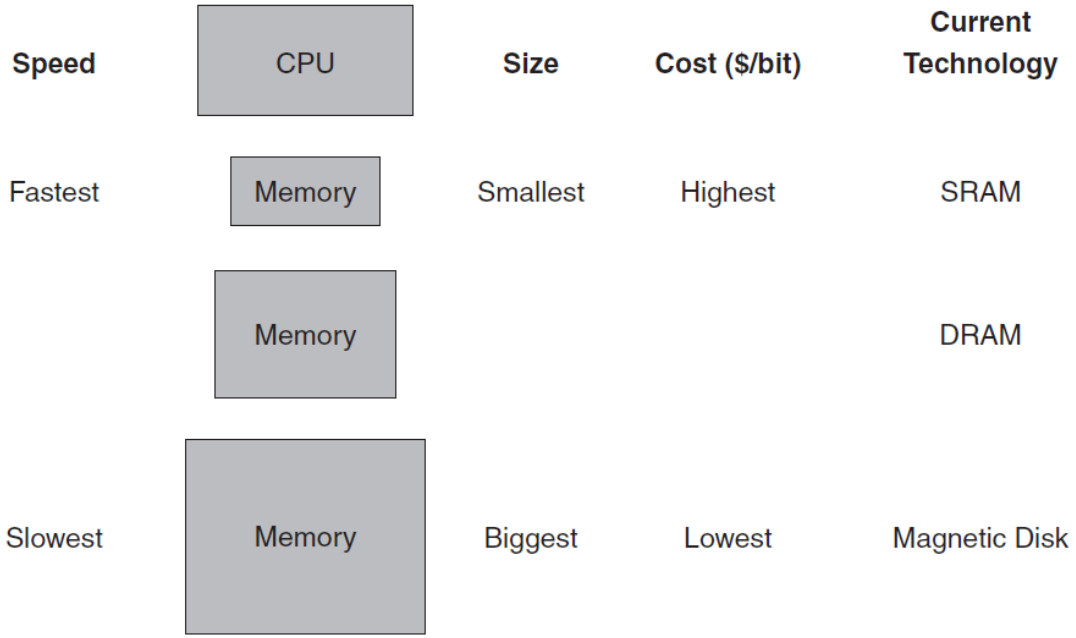
Memory Hierarchy



Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	< 16 MB	< 512 GB	> 1 TB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25–0.5	0.5–25	50–250	5,000,000
Bandwidth (MB/sec)	50,000–500,000	5000–20,000	2500–10,000	50–500
Managed by	compiler	hardware	operating system	operating system/ operator
Backed by	cache	main memory	disk	CD or tape

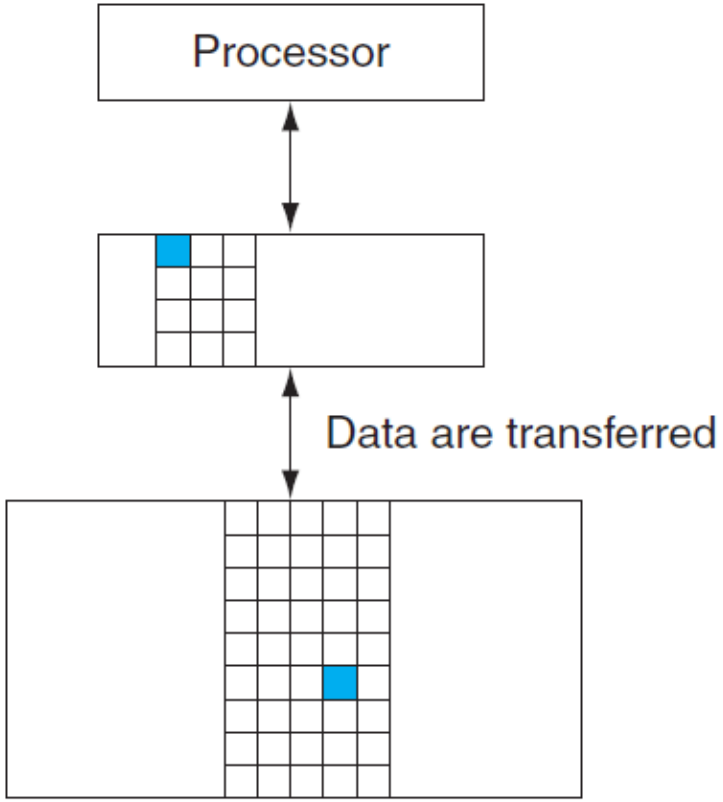
Memory Hierarchy

- The memory system is organized as a hierarchy
  - A level closer to the processor is a subset of any level further away.
  - All the data is stored at the lowest level.



- **Hierarchical implementation makes the illusion of a memory size as the largest, but can be accessed as the fastest.**

Hit and Miss



- In a pair of levels one is **upper** and one is **lower**.
- The unit within each level is called a **block**.
- We transfer an entire block when we copy something between levels.

Hit rate, or **hit ratio**, is the fraction of memory accesses found in the upper level. **Miss rate** = 1 – hit rate.

Hit time: the time required to access a level of the memory hierarchy.

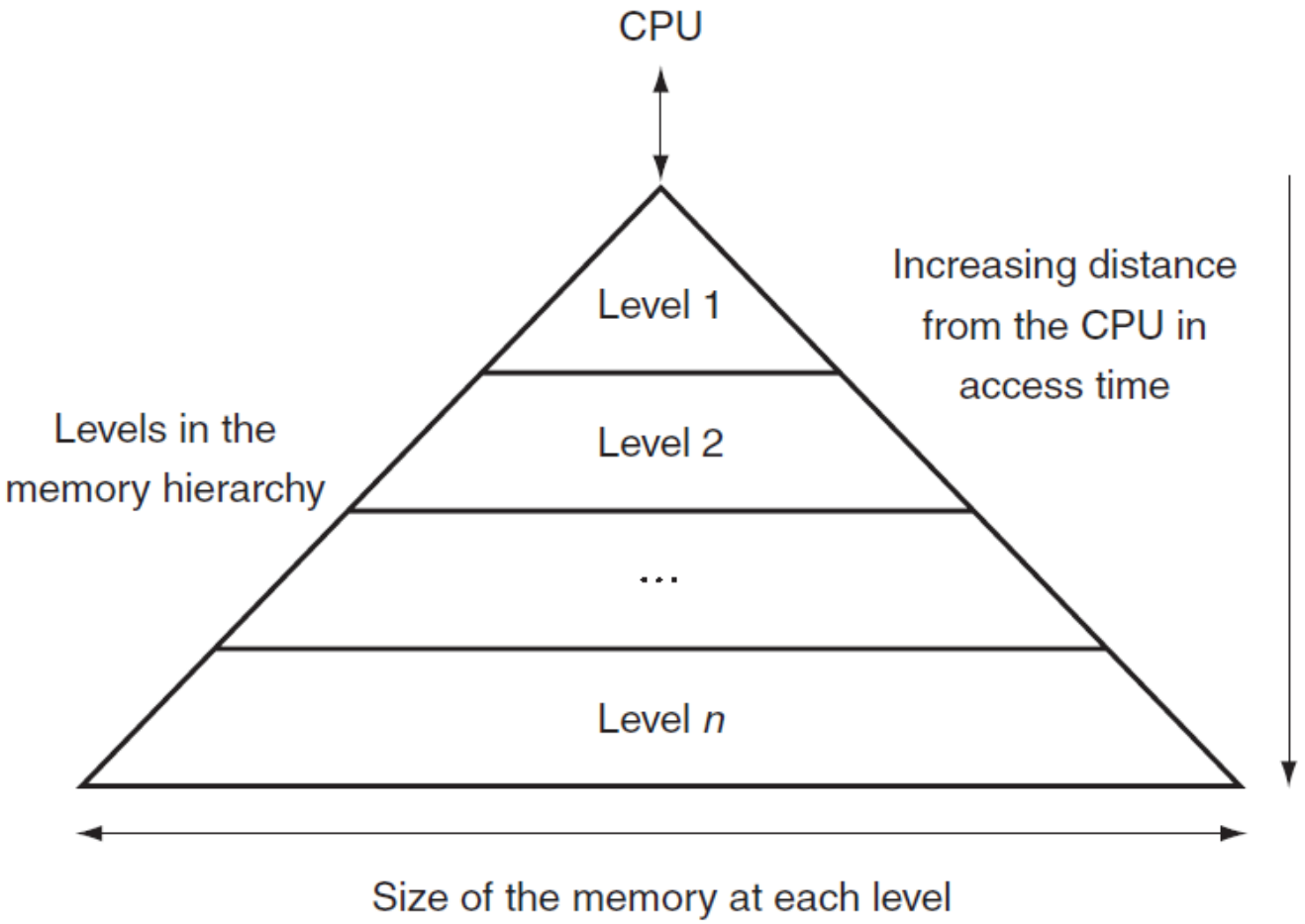
- Includes the time needed to determine whether hit or miss.

Miss penalty: the time required to fetch a block into the memory hierarchy from the lower level.

- Includes the time to access the block, transmit it from the lower level, and insert it in the upper level.

The memory system affects many other aspects of a computer:

- How the **operating system** manages memory and I/O
- How **compilers** generate code
- How **applications** use the computer

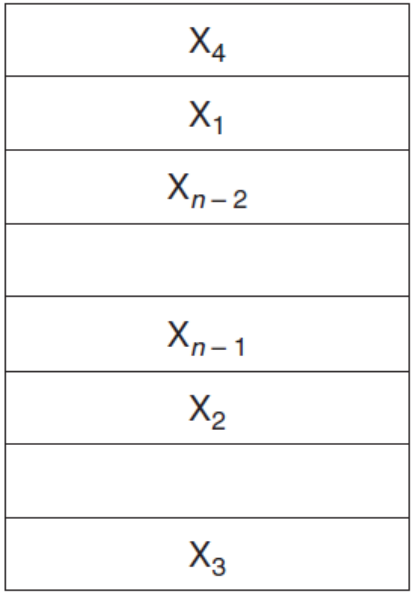


This structure allows the processor to have an access time that is determined primarily by level 1 of the hierarchy and yet have a memory as large as level *n*.

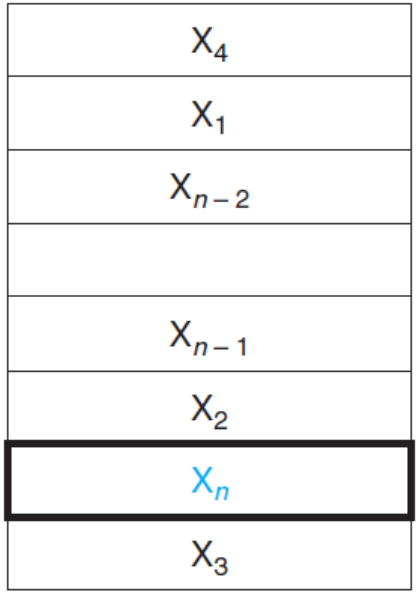
Requesting data from the cache

The processor requests a word X_n that is **not** in the cache

Before reference to X_n



After reference to X_n



Two questions :

- How do we know if a data item is in the cache?
- If it is, how do we find it?

Direct-Mapped Cache

Each memory location is mapped to **one** cache location

Mapping between addresses and cache locations:

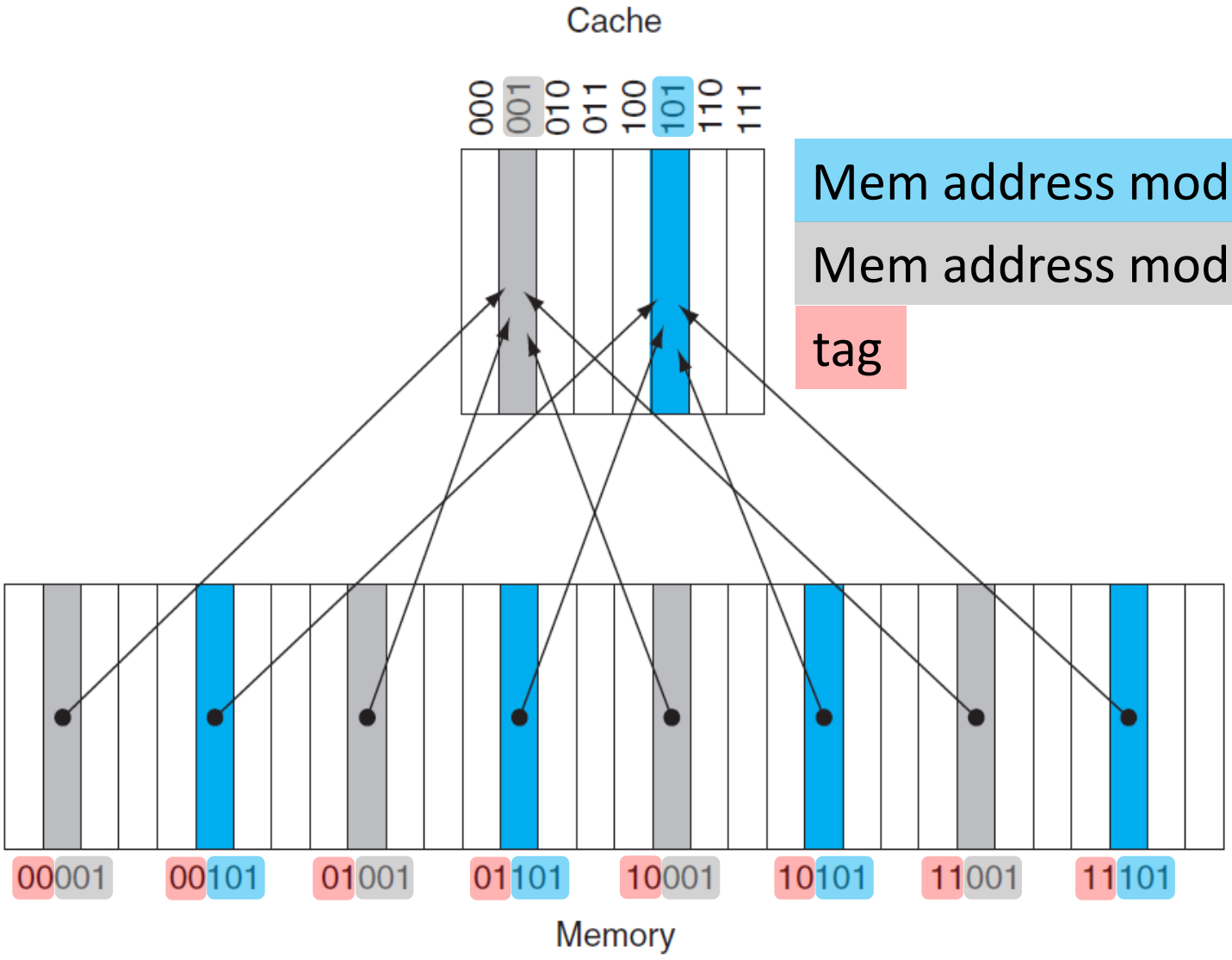
(Block address in Mem) % (# of blocks in cache)

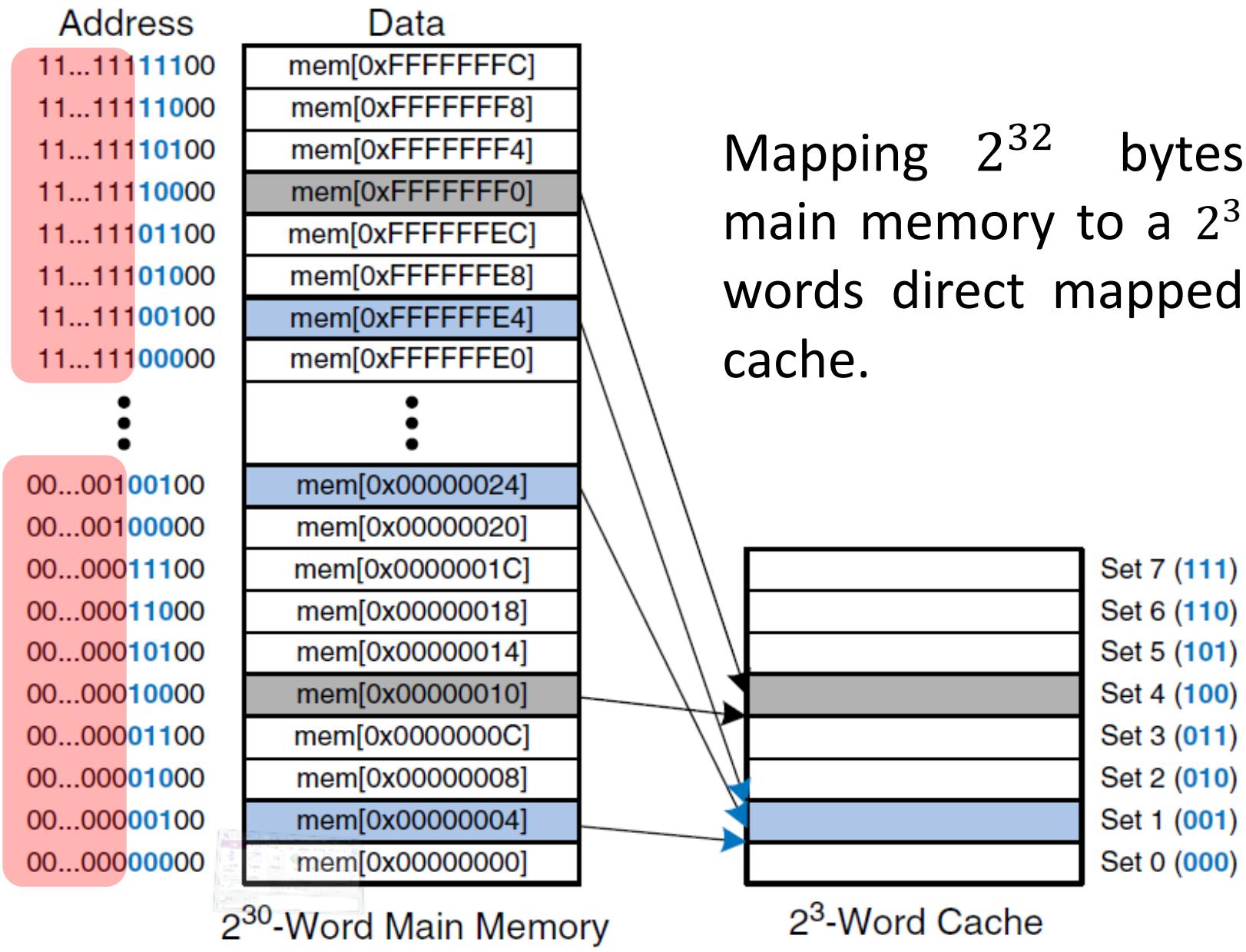
Modulo is computed by using **$\log_2$(cache size in blocks)** LSBs of the address.

The cache is accessed directly with the LSBs of the requested memory address.

Problem: this is a many-to-one mapping.

A **tag** field in a **table** containing the MSBs to identify whether the block in the hierarchy corresponds to a requested word.


Cache
000
001
010
011
100
101
110
111
Mem address mod 8 = 101
Mem address mod 8 = 001
tag
00001
00101
01001
01101
10001
10101
11001
11101
Memory



Mapping 2^{32} bytes main memory to a 2^{3} words direct mapped cache.

Some of the cache entries may still be empty.

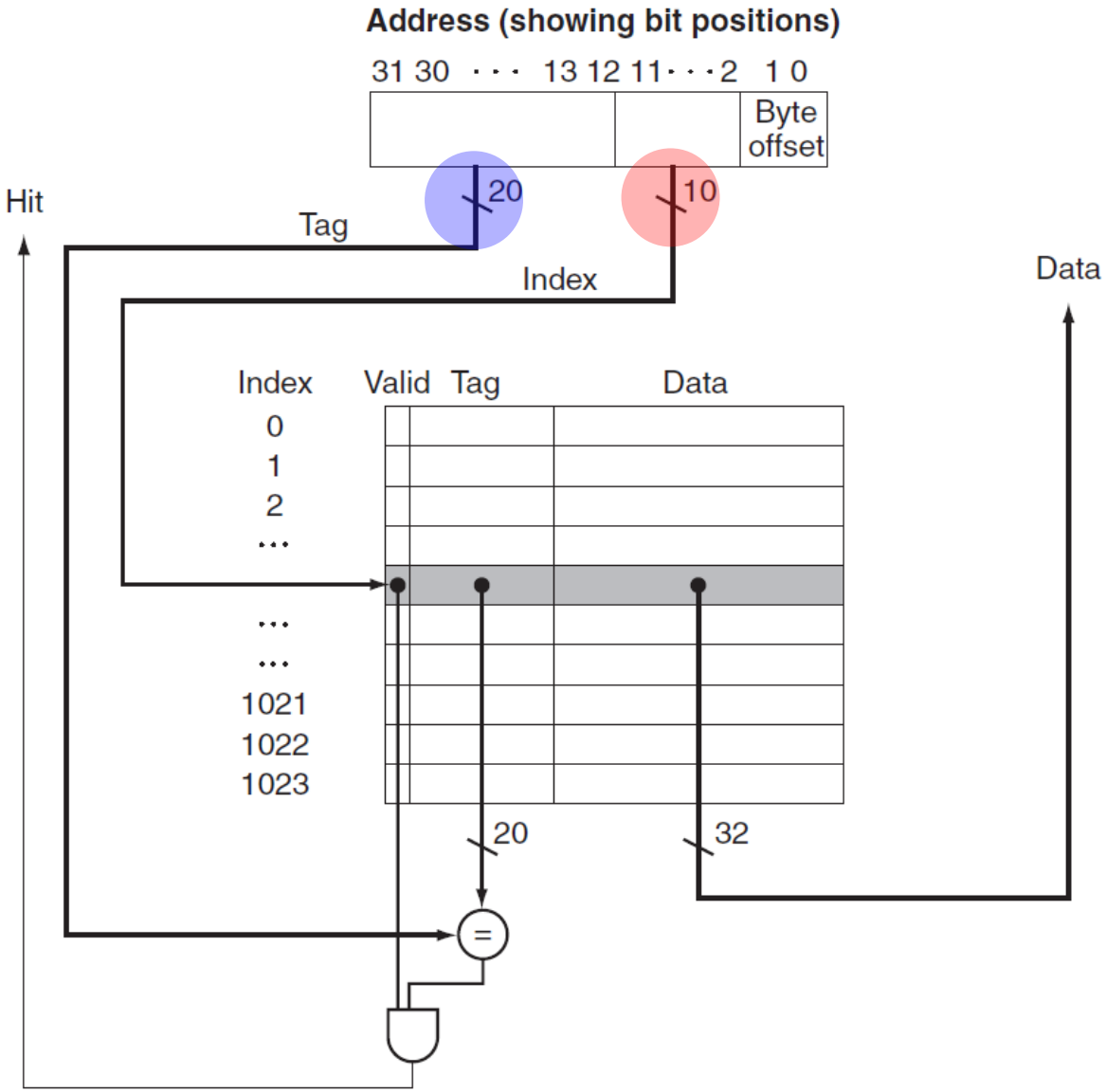
We need to know that the tag should be ignored for such entries.

We add a **valid bit** to indicate whether an entry contains a valid address.

Cache Access Sequence

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (7.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (7.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (7.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (7.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (7.6f)	$(10010_{two} \bmod 8) = 010_{two}$

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		



Referenced address is divided into

- a cache index, used to select the block
- a tag field, compared with the value of the tag field of the cache

Cache Size

The cache includes both the storage for the data and the tags. The size of the block is normally several words.

For 32-bit byte address, a direct-mapped cache of 2^n blocks size with 2^m words (2^{m+2} bytes) in a block, will require a tag field which size is $32 - (n + m + 2)$ bits.

The total number of bits in a direct-mapped cache is therefore 2^n x (block size + tag size + valid field size).

Since the block size is 2^m 32-bit words (2^{m+5} bits), and the address size is 32 bits, the number of bits in a direct-mapped cache is $2^n \times [2^{m+5} + (32 - n - m - 2) + 1] =$

$$2^n \times (2^{m+5} + 31 - n - m)$$

The convention is to count only the size of the data.

Example: How many total bits are required for a direct-mapped cache with 16 KB of data and 4-word blocks, assuming a 32-bit address?

16 KB is 4K words, which is 2^{12} words, and, with a block size of 4 words (2^2), there are 2^{10} blocks.

Each block has 4 x 32 = 128 bits of data, plus a tag of 32 - 10 - 2 - 2 bits, plus a valid bit. The total cache size is therefore

2^{10} x (128 + (32 - 10 - 2 - 2) + 1) = 147 Kbits = 18.4 KB

For a 16 KB cache it is about 1.15 times as many as needed just for data storage.

Example: Find the cache block location that byte 1200 in Mem maps to, in a 64-blocks cache with 16-byte block size.

Cache block locations
=(Mem block address)%(#blocks in cache)

Mem block address $=$ ⌊Mem byte address/bytes per block⌋ $=$

$$\lfloor 1200/16 \rfloor = 75$$

Block address contains all the bytes in range

From: ⌊Mem byte address/bytes per block⌋ $\times$ bytes per block
$= 75 \times 16 = 1200$

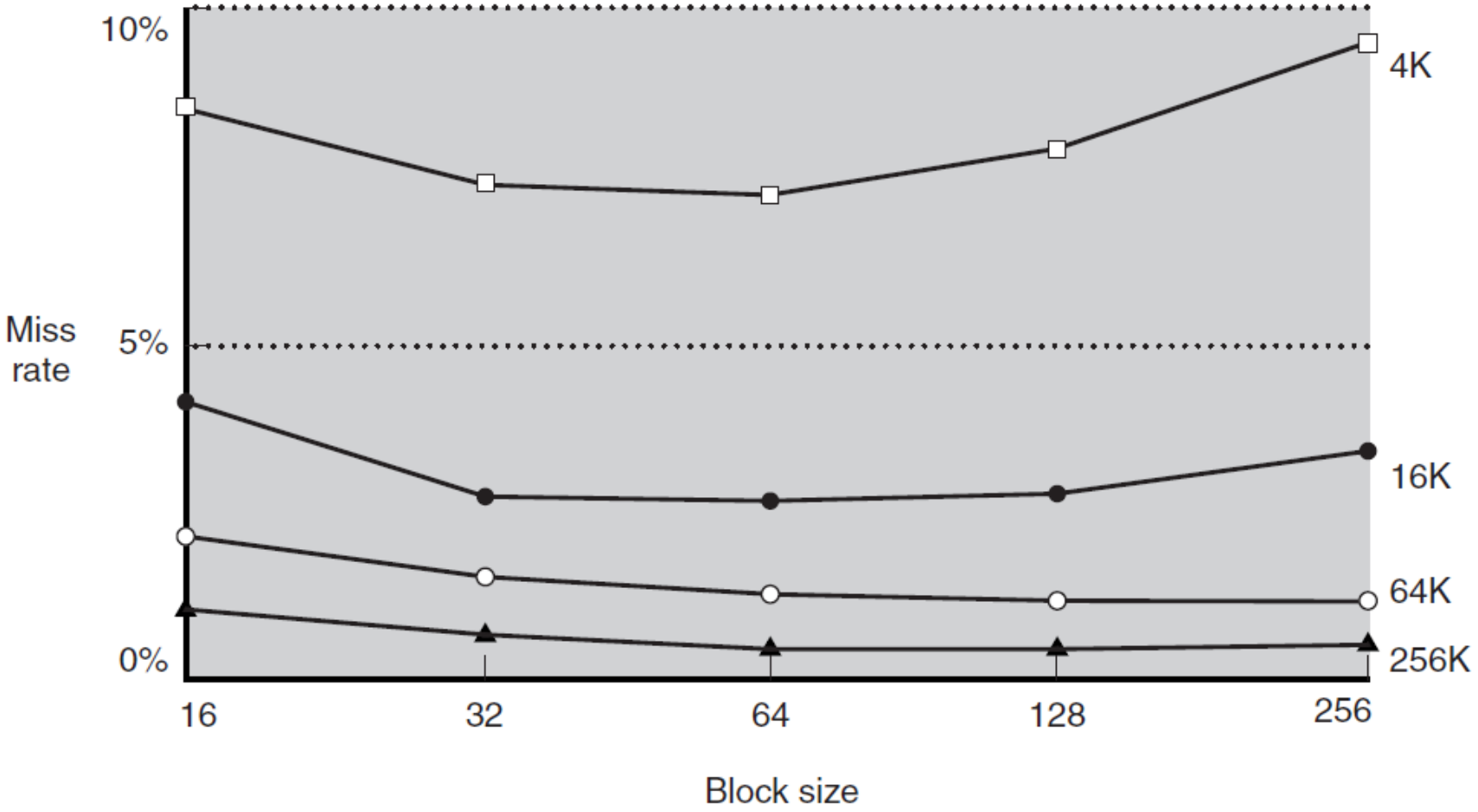
To: $(\lfloor$Mem byte address/bytes per block$\rfloor + 1) \times$ bytes per block $- 1 = 76 \times 16 - 1 = 1215$

It maps to cache block number $(75\ \%\ 64)\ =\ 11$, containing all bytes addresses between 1200 and 1215.

Block Size Implications

- Larger blocks exploit spatial locality to lower miss rates.
- Block increase will eventually increase miss rate
- Spatial locality among the words in a block decreases with a very large block.
  - The number of blocks held in the cache will become small.
  - There will be a big competition for those blocks.
  - A block will be thrown out of the cache before most of its words are accessed.

Miss rate versus block size



A more serious issue in block size increase is the increase of miss cost.

- Determined by the time required to fetch the block and load it into the cache.

Fetch time has two parts:

- the latency to the first word, and
- the transfer time for the rest of the block.

Transfer time (miss penalty) increases as the block size grows.

The increase in the miss penalty overwhelms the decrease in the miss rate for large blocks, thus decreasing cache performance.

- Shortening transfer time is possible by **early restart**, resuming execution once the word is returned.
  - Useful for instruction, that are largely sequential.
  - Requires that the memory delivers a word per cycle.
  - Less effective for data caches. High probability that a word from different block will be requested soon.
  - If the processor cannot access the data cache because a transfer is ongoing, it must stall.
- Requested word first
  - starting with the address of the requested word and **wrapping around**.
  - Slightly faster than early restart.

Handling Cache Misses

Modifying the control of a processor to handle a hit is simple.

Misses require extra work done with the processor's control unit and a separate controller.

Cache miss creates a stall by freezing the contents of the pipeline and programmer-visible registers, while waiting for memory.

Steps taken on an **instruction cache miss**:

1. Send to the memory the original PC value.
2. Instruct main memory to perform a read and wait for the memory to complete its access.
3. Write the cache entry: memory's data in the entry's data portion, upper bits of the address into the tag field, turn the valid bit on.
4. Restart the instruction execution at the first step, which will re-fetch the instruction, this time finding it in the cache.

The control of the data cache is similar: miss stalls the processor until the memory responds with the data.

Handling Writes

After a hit writes into the cache, memory has a different value than the cache. Memory is **inconsistent**.

We can always write the data into both the memory and the cache, a scheme called **write-through**.

Write miss first fetches block from memory. After it is placed into cache, we overwrite the word that caused the miss into the cache block and also write it to the main memory.

Write-through is simple but has bad performance. Write is done both to cache and memory, taking many clock cycles (e.g. 100).

If 10% of the instructions are stores and the CPI without misses was 1.0, new CPI is 1.0 + 100 x 10% = 11, a **10x slowdown**!

Speeding Up

A **write buffer** is a queue holding data waiting to be written to memory, so the processor can continue working. When a write to memory completes, the entry in the queue is freed.

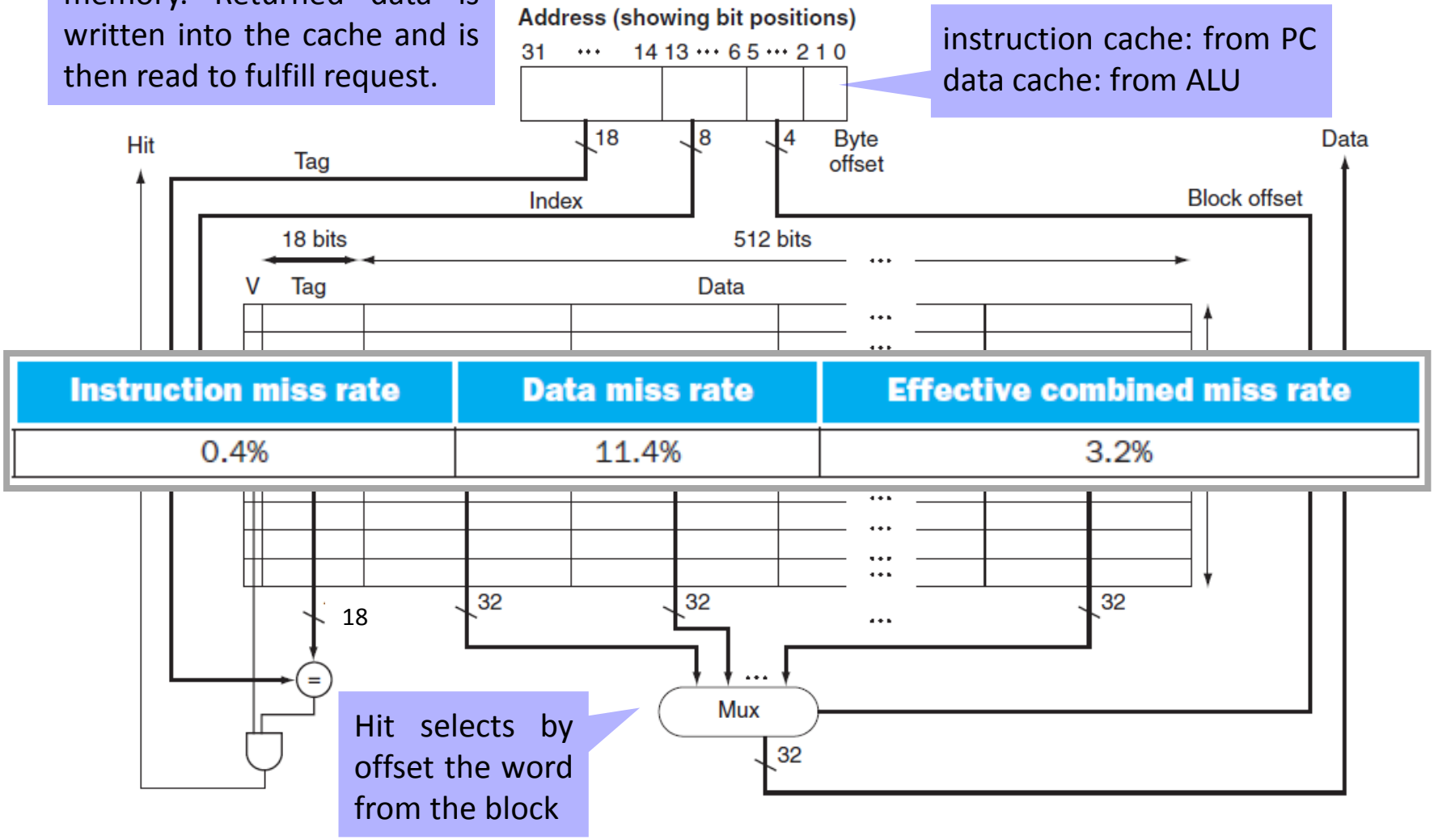
If the queue is full when the processor reaches a write, it must **stall** until there is an empty position in the queue.

An alternative to **write-through** is **write-back.** At write, the new value is written only to the cache. The modified block is written to the main memory when it is replaced.

Write-back improves performance when processor generates writes faster than the writes can be handled by main memory. Implementation is more complex than **write-through**.

Cache Example (Data and Instruction)

Miss sends the address to memory. Returned data is written into the cache and is then read to fulfill request.



Main Memory Design Considerations

Cache misses are satisfied from DRAM main memory, designed for density rather than access time.

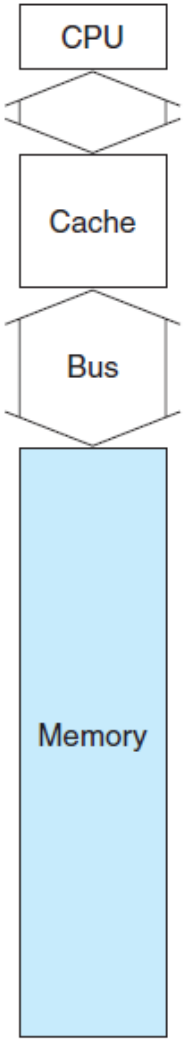
Miss penalty can be reduced by increasing bandwidth from the memory to the cache.

Bus clock rate is 10x slower than processor, affecting the miss penalty. Assume

- 1 memory bus clock cycle to send the address
- 15 memory bus clock cycles for each DRAM access initiated
- 1 memory bus clock cycle to send a word of data

For a cache block of 4 words and a one-word-wide bank of DRAM, $\text{miss penalty} = 1 + 4 \times 15 + 4 \times 1 = 65$ memory bus clock cycles.

Bytes transferred per **bus** clock cycle $= (4 \times 4)/65 = 0.25$.



a. One-word-wide memory organization

$\text{Miss penalty} = 1 + 1 \times 15 + 1 = 17$ cycles. Bytes transferred per cycle $= (4 \times 4)/17 = 0.94$. **Wide bus (area) and MUX (latency) are expensive.**

$\text{Miss penalty} = 1 + 1 \times 15 + 4 \times 1 = 20$ cycles. Bytes transferred per cycle $= 0.8$.

Cache Performance

Two techniques to reduce miss rate:

- Reducing the probability that two different memory blocks will contend for the same cache location by **associativity**.
- Adding a level to the hierarchy, called **multilevel caching**.

CPU Time

CPU time = (CPU execution clock cycles + Memory-stall clock cycles) x Clock cycle time

Memory-stall clock cycles = Read-stall cycles + Write-stall cycles

Read-stall cycles = Reads/Program x Read miss rate x Read miss penalty

Write-stall cycles = Writes/Program x Write miss rate x Write miss penalty + Write buffer stall cycles (write-through)

Write buffer term is complex. It can be ignored for buffer depth > 4 words, and a memory capable of accepting writes at > 2x rate than the average write frequency.

Write-back also has additional stalls arising from the need to write a cache block back to memory when it is replaced.

Write-through has about the same read and write miss penalties (fetch time of block from memory). Ignoring the write buffer stalls, the miss penalty is:

Memory-stall clock cycles (simplified) =

Memory accesses/Program x Miss rate x Miss penalty =

Instructions/Program x Misses/Instruction x Miss penalty

Example: impact of an ideal cache

A program is running I instructions. 2% instruction cache miss, 4% data cache miss, 2 CPI without any memory stalls, and 100 cycles penalty for all misses.

How faster is a processor with a never missed cache?

Instruction miss cycles = $I \times 2\% \times 100 = 2.0 \times I$

With 36% loads and stores,
Data miss cycles = $I \times 36\% \times 4\% \times 100 = 1.44 \times I$

CPI with memory stalls = $2 + 2 + 1.44 = 5.44$

$$\text{Speedup} = \text{CPI}_{\text{stall}} / \text{CPI}_{\text{perfect}} = 5.44/2 = 2.77$$

Example: Accelerating processor but not memory. Memory stalls time fraction is increased.

CPI reduced from 2 to 1 (e.g. deeper pipeline), system with cache misses have $\mathrm{CPI} = 1 + 3.44 = 4.44$. System with perfect cache is $4.44/1 = 4.44$ faster.

The execution time spent on memory stalls increases from $3.44/5.44 = 63\%$ to $3.44/4.44 = 77\%$. ☹

Processor's clock cycle reduced by 2x, but memory bus not, $\mathrm{CPI}_{\mathrm{stall}} = 2 + 2\% \times 200 + 36\% \times 4\% \times 200 = 8.88$

$\mathrm{Perf}_{\mathrm{slow}}/\mathrm{Perf}_{\mathrm{fast}} = 5.44/(8.88 \times 1/2) = 1.22$, rather than 2x. ☹

Relative cache penalties increase as a processor becomes faster.

If a processor improves both CPI and clock rate

- The smaller the CPI, the more impact of stall cycles is.
- If the main memories of two processors have the same absolute access times, higher processor's clock rate leads to larger miss penalty.

The importance of cache performance for processors with small CPI and faster clock is greater.

Reducing Cache Misses

Direct map scheme places a block in a **unique** location.

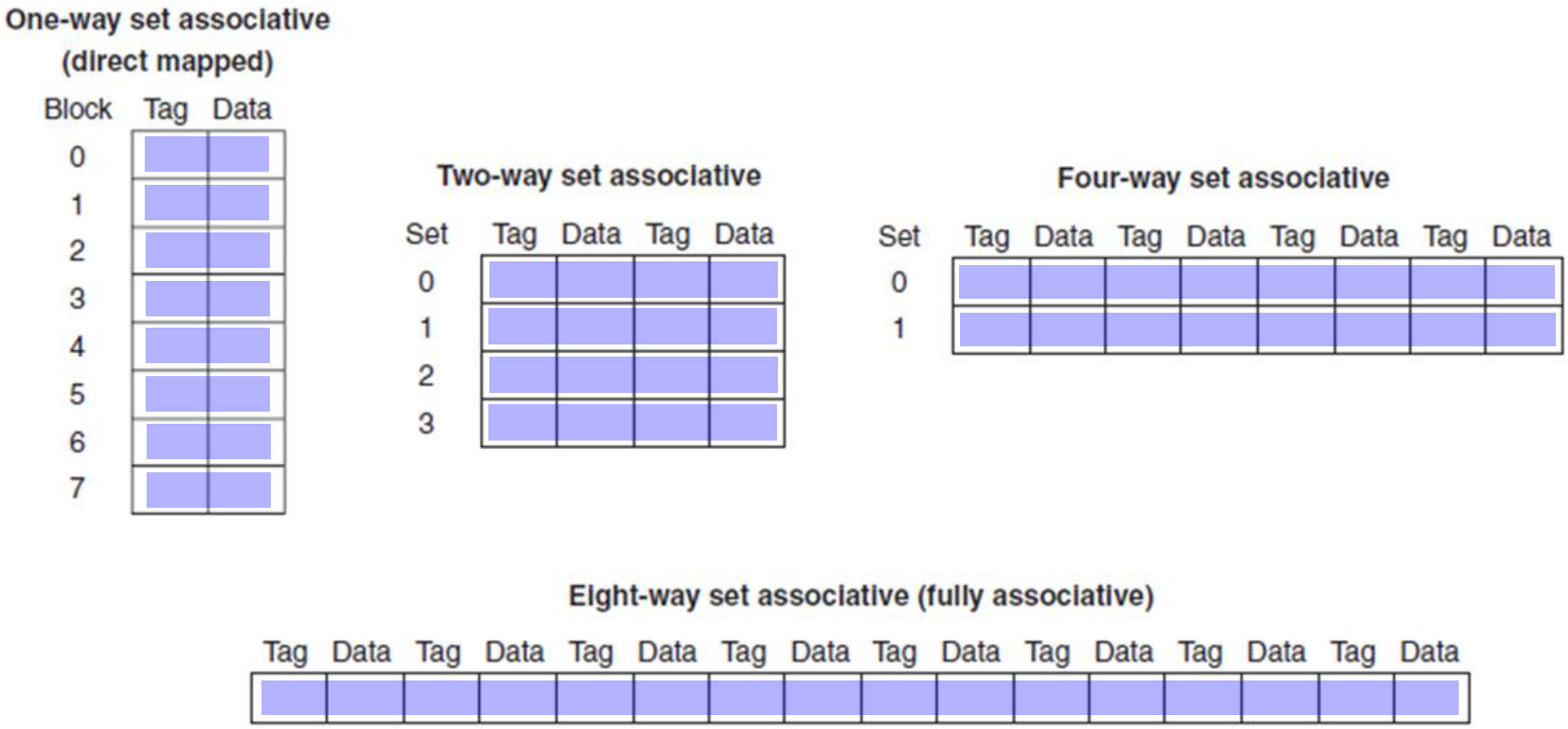
Fully associative scheme places a block in **any** location.

- All cache's entries must be searched.
- Expensive: done in parallel with a comparator for each entry.
- Practical only for caches with a small number of blocks.

A middle solution is called ***n*-way set-associative** map.

- Fixed number (***n***) of locations where a block can be placed.
- A number of sets, each of which consists of ***n*** blocks.
- A memory block maps to a unique **set** in the cache given by the index field. A block is placed in **any** element of that set.


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
Cache
Set
0
Set
1
Set
2
Set
3
Block frame address
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
Memory



Cache size (blocks) = number of sets x associativity.

For fixed cache size, increasing the associativity decreases the number of sets.

Example: Misses and associativity in caches.

Three caches of 4 1-word blocks, fully associative, two-way set associative, and direct mapped.

For the sequence of block addresses: 0, 8, 0, 6, 8, what is the number of misses for each cache?

direct mapped

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

5 misses

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

two-way set associative

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

4 misses

fully associative

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

3 misses

Size and associativity are dependent in determining cache performance.

For 8 blocks in the cache, there are no replacements in the two-way set-associative cache. (why?)

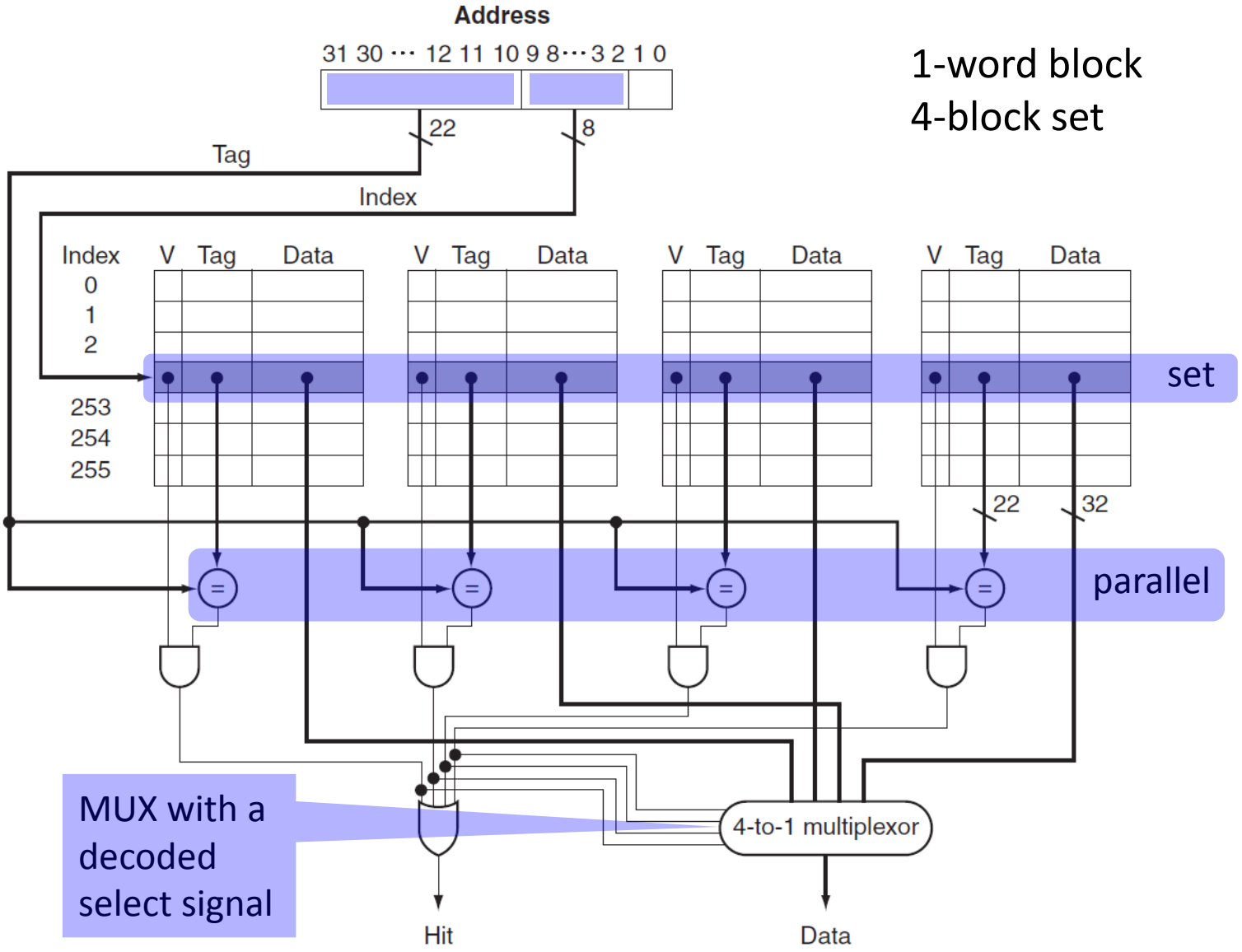
There are same number of misses as the fully associative cache.

For 16 blocks, all three caches would have the same number of misses.

Benchmarks of a 64 KB data cache with a 16-word block

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

Four-way set-associative cache



1-word block
4-block set

Locating a Block in the Cache

Set is found by the index. **Tag** of a block within the appropriate set is checked for matching. **Block offset** is the address of the word within the block.

For **speed** all the tags in a set are searched in **parallel**.

In a fully associative cache, we search the entire cache without any indexing. **Huge HW overhead**.

The choice among direct-mapped, set-associative, or fully associative depends on the miss (performance) cost versus HW cost (power, area).

Example: Size of tags versus set associativity

Given cache of $4K=2^{12}$ blocks, a 4-word block size, and a 32-bit address. What is the total number of sets tag bits?

There are $16=2^4$ bytes / block. 32-bit address yields 32-4 =28 bits for index and tag.

Direct-mapped cache has $12=\log_2(4K)$ bits of index. Tag is 28-12=16 bits, yielding a total of 16 x 4K = 64 Kbits of tags.

For a 2-way set-associative cache, there are $2K = 2^{11}$ sets, and the total number of tag bits is $(28 - 11) \times 2 \times 2K = 34 \times 2K = 68$ Kbits.

For a 4-way set-associative cache, there are $1K = 2^{10}$ sets, and the total number of tag bits is $(28 - 10) \times 4 \times 1K = 72 \times 1K = 72$ Kbits.

Fully associative cache has one set with 4K blocks, and the total number of tag bits is $28 \times 4K \times 1 = 112K$ bits.

Which Block to Replace?

In a direct-mapped cache the requested block can go in exactly one position.

In a set-associative cache, we must choose among the blocks in the selected set.

The most commonly used scheme is **least recently used** (**LRU**), where the block replaced is the one that has been unused for the longest time.

For a two-way set-associative cache, tracking when the two elements were used can be implemented by keeping a single bit in each set.

As associativity increases, implementing LRU gets harder.

Random

- Spreads allocation uniformly.
- Blocks are randomly selected.
- System generates pseudorandom block numbers to get reproducible behavior (useful for HW debug).

First in, first out (**FIFO**) Because LRU can be complicated to calculate, this approximates LRU by determining the **oldest** block rather than the LRU.

Multilevel Caches

Used to reduce miss penalty.

Many µP support a 2^{nd}-level (L2) cache, which can be on the same die or in separate SRAMs (old days).

L2 is accessed whenever a miss occurs in L1.

If L2 contains the desired data, the miss penalty for L1 is the access time of L2, much less than the access time of main memory.

If neither L1 nor L2 contains the data, main memory access is required, and higher miss penalty incurs.

Example: performance of multilevel caches

Given a 5 GHz processor with a base CPI of 1.0 if all references hit in the L1.

Main memory access time is 100 ns, including all the miss handling.

L1 miss rate per instruction is 2%.

How faster the processor is if we add a L2 that has a 5 ns access time for either a hit or a miss, reducing the miss rate to main memory to 0.5% ?

Miss penalty to main memory (memory-stall):

5GHz x 100 ns = 500 cycles.

The effective CPI with L1:

Base CPI + Memory-stall cycles per instruction =

1 + 500 x 2% = 11

The effective CPI with L2:

1 + 25 x (2% - 0.5%) + (500 + 25) x 0.5% = 4

The processor with L2 is faster by:

11 / 4 = 2.8

דוגמא: נתונה מערכת זיכרון במעבד הפועל בתדר 500 MHz בעלת שתי רמות של זיכרון מטמון.

L1-data cache הינו direct-mapped, write-through, בגודל כולל של 8KByte וגודל בלוק של 8Byte.

מניחים שחוצץ הכתיבה שלו מושלם ואין אף פעם stalls. miss-rate הינו 15%.

L1-instruction cache הינו direct-mapped, בגודל כולל של 4KByte וגודל בלוק של 8Byte. נתון שה miss-rate הינו 2%.

L2 הינו יחיד ומשותף, 2-way set associative, write-back, בגודל כולל של 2MByte וגודל בלוק של 32 Byte. miss-rate הינו 10%.

בממוצע 50% מהבלוקים ב L2 הינם "מלוכלכים", כלומר רשום בהם מידע שאיננו כרגע בזיכרון הראשי.

כמה סיביות בכל אחד מזיכרונות המטמון משמשות לאינדקס?

L1 Data: 8Kbyte/8Byte = 1024 blocks => 10 bits
L1 Instruction: 4Kbyte/8Byte = 512 blocks => 9 bits
L2: 2MByte/32Bytes = 64K blocks = 32K sets => 15 bits

איזה אחוז מתוך גישות הנתונים לזיכרון מגיע לזיכרון הראשי?

(L1 miss rate) x (L2 miss rate) = 0.15 X 0.1 = 1.5%

40% מהפקודות הינן פקודות גישה לזיכרון, 60% מהן קריאה (LOAD) ו 40% מהן כתיבה (STORE). L1 hits אינם גורמים ל stalls.

זמן גישה ל L2 הינו 20 ננו שניות.

זמן גישה לזיכרון הראשי הינו 0.2 מיקרו שניה, ומרגע זה מספר מילים כרוחב ה memory bus נשלחות כל מחזור שעון. רוחב ה bus המחבר בין L2 לזיכרון הראשי הינו 128 סיביות.

מה מספר מחזורי השעון המרבי שעשוי להידרש בעת גישה לזיכרון הראשי? מהו רצף האירועים המתרחש במצב קיצוני כזה?

Maximum clock cycles occur when L1 missed first, then L2 missed, then write-back takes place.

L2 access cycles: (20 nSec) / (2 nSec) = 10 cycles
Main memory access cycles: (0.2 µSec) / (2 µSec) = 100 cycles

Block is 32 Bytes and memory bus is 128 bits (16 Bytes), two bus transactions of 16 Bytes each are required. The first 16 bytes take 100 cycles, the next 16 bytes takes one cycle.

Getting a new block from the memory may evict a block from L2, which is a write-back. In that case the evicted block must be written into the memory, requiring a total of L2-memory write-back 2 x (100 + 1) = 202 cycles.

Summing all
L1 miss + L2 miss + write-back = 1 + 10 + 202 = 213 cycles

מהו מספר מחזורי השעון הממוצע בגישה לזיכרון (AMAT) כולל פקודות ונתונים ?

The weight of instruction accesses to memory is 1/(1 + 0.4), while the weight of data accesses is 0.4/(1 + 0.4). Therefore
$AMAT_{total} = 1/1.4\ AMAT_{inst} + 0.4/1.4\ AMAT_{data}$

For any 2-level cache system there is
AMAT = (L1 hit time) + (L1 miss rate) x (L2 hit time) + (L1 miss rate) x (L2 miss rate) x (main memory transfer time).

AMAT must account for the average percentage of L2 dirty blocks, which for the given L2 means that 50% of the blocks must be updated in main memory upon L2 miss, yielding a factor of 1.5 multiplying (100 +1).

$AMAT_{inst} = 1 + 0.02 \times 10 + 0.02 \times 0.1 \times 1.5 \times (100 + 1) = 1.503$

$AMAT_{data} = 1 + 0.15 \times 10 + 0.15 \times 0.1 \times 1.5 \times (100 + 1) = 4.7725$

$AMAT_{total} = 1/1.4 \times 1.503 + 0.4/1.4 \times 4.7725 = 2.44$

Summary – Four Questions

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