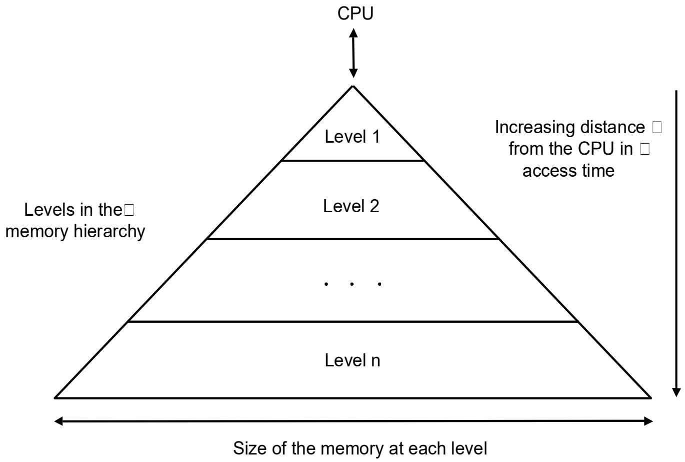
Chapter 5

Large and Fast: Exploiting Memory Hierarchy



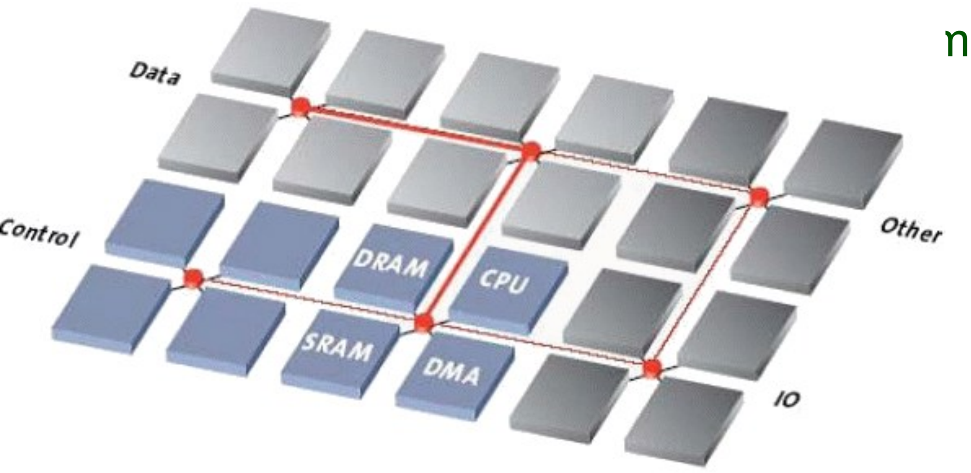
Kuei-Chung Chang

Email: changkc@fcu.edu.tw

Ext. 3753, Office 211

nation Engineering and Computer Science

Feng Chia Univ.



Spring 2019

Outline

- 5.1 Introduction
- 5.3 The Basic of Caches
- 5.4 Measuring and Improving Cache Performance
- 5.7 Virtual Memory
- 5.8 A Common Framework for Memory Hierarchies
- 5.10 Parallelism and Memory Hierarchies: Cache Coherence
- 5.16 Concluding Remarks

Outline

5.1 Introduction

5.3 The Basic of Caches

5.4 Measuring and Improving Cache Performance

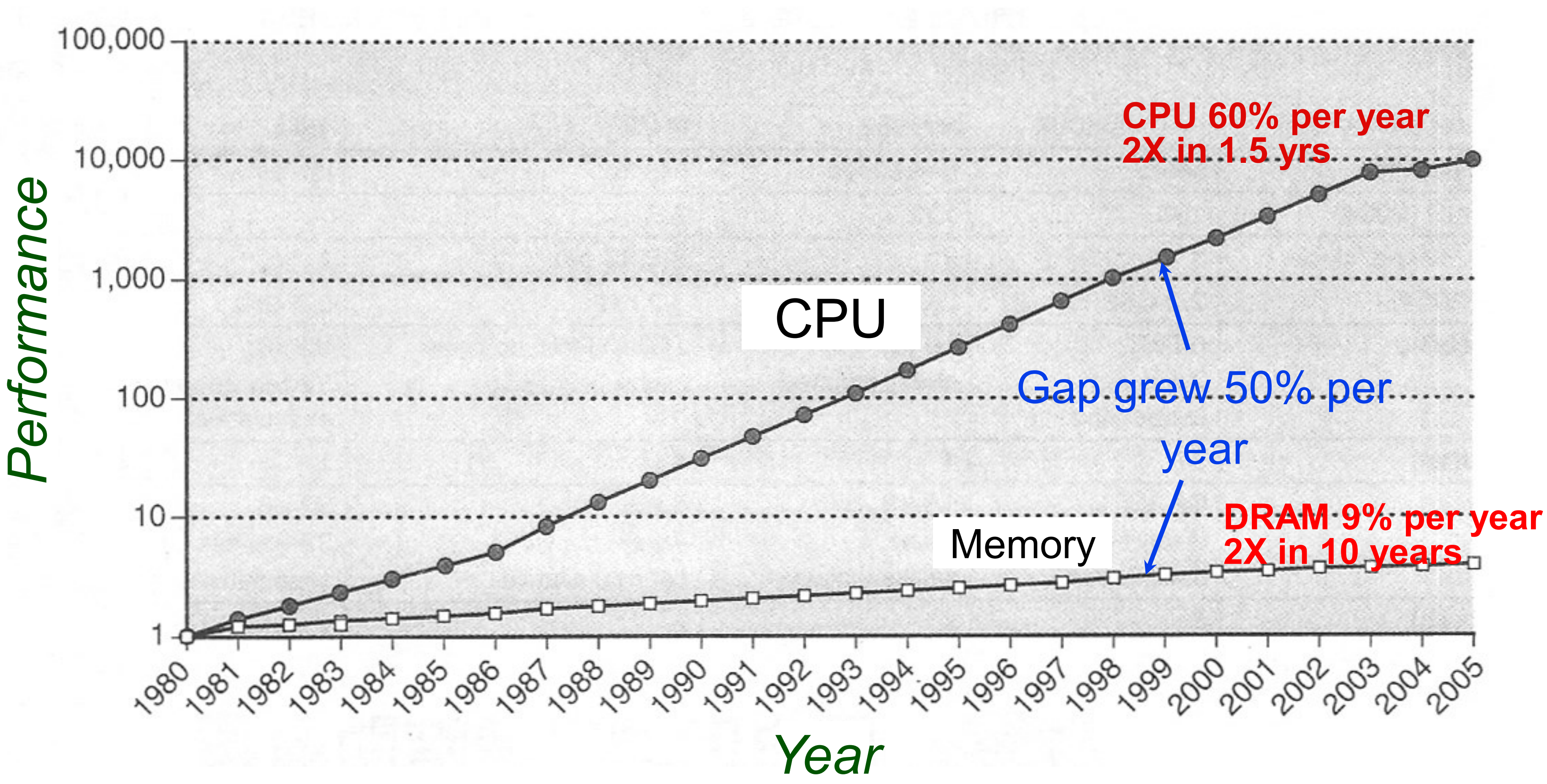
5.7 Virtual Memory

5.8 A Common Framework for Memory Hierarchies

5.10 Parallelism and Memory Hierarchies: Cache Coherence

5.16 Concluding Remarks

Since 1980, CPU has outpaced DRAM ...

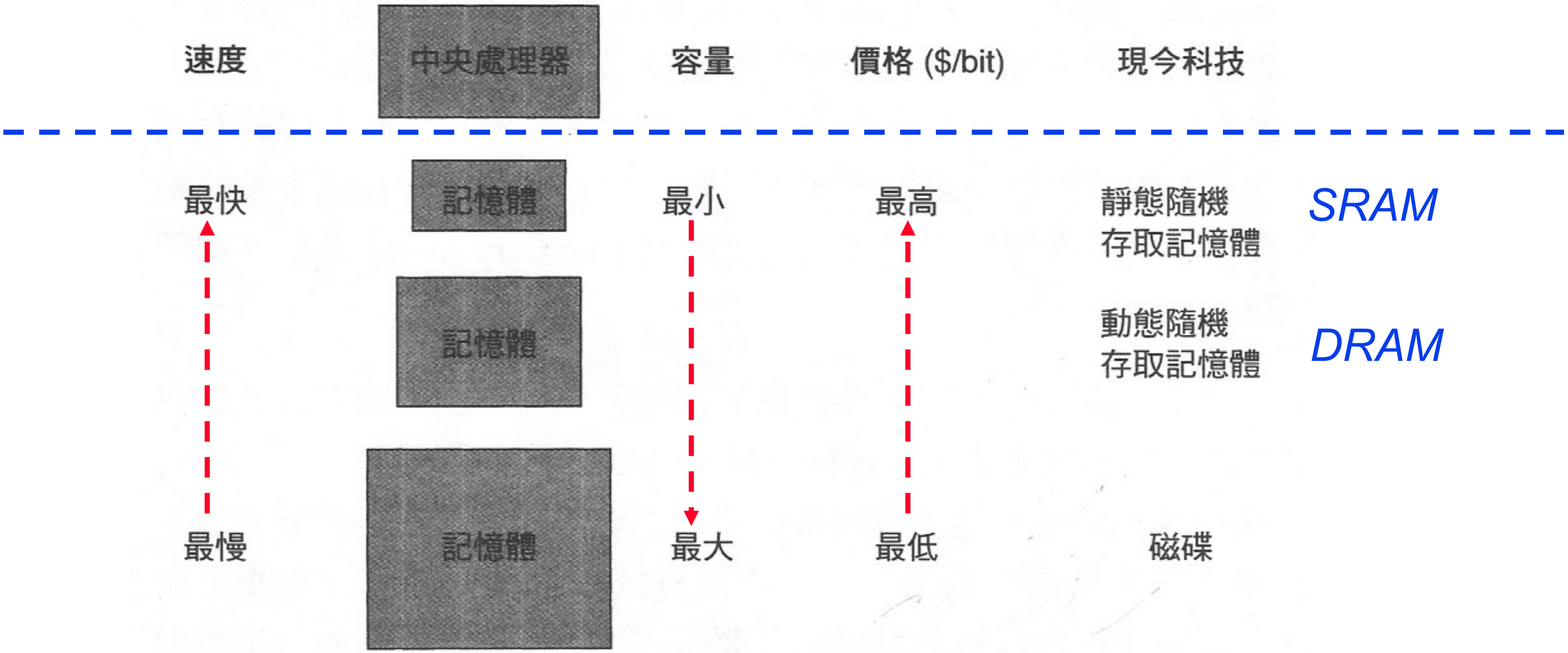


Q. How do architects address this gap?

A. 1. Put smaller, faster "***cache***" memories between CPU and DRAM.

2. Create a "***memory hierarchy***".

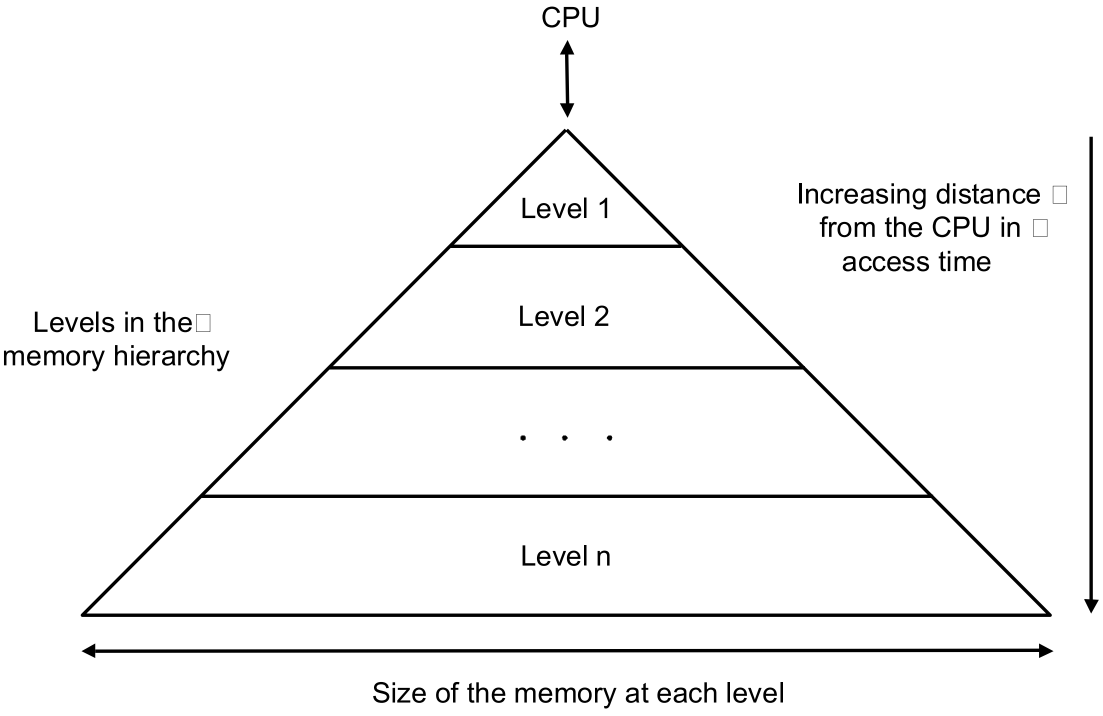
Speed vs. Cost



- The figure shows the faster memory is close to the processor and the slower, less expensive memory is below it.
- It can provide the user with *as much memory as is available in the cheapest technology*, while *providing access at the speed offered by the fastest memory*.

Structure of memory hierarchy

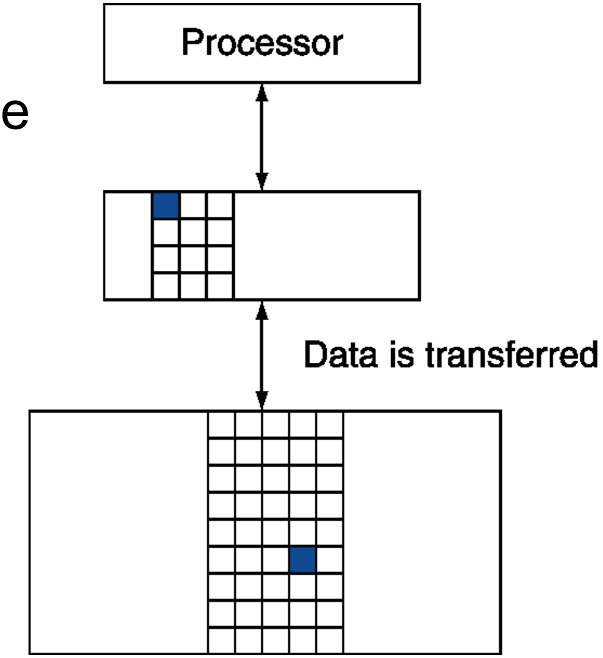
- Access that ***miss*** go to lower levels of the hierarchy, which are larger but slower.
- *If the hit rate is high enough*, the memory hierarchy has an *effective access* time close to that of the highest level and a *size* equal to that of the lowest (and largest) level.


CPU
Level 1
Level 2
. . .
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level

Levels in the memory hierarchy

- The minimum unit of information that can be either present or not present in the two-level hierarchy is called a **block** or a **line**.
- Data is copied between only *two adjacent levels* at a time.

- **Hit**：If the data requested by the processor appears in some block in the upper level.
- **Miss**：If the data is not found in the upper leve
- **Hit rate**：The fraction of memory access is found in the upper level.
- **Miss rate**：1 - Hit rate
- **Hit time**：Determining time + Access time
- **Miss Penalty**：deliver time to transfer block from lower to upper + replace block time



Memory Hierarchy

- *A program does not access all of its code or data at once* with equal probability. Otherwise, it would be impossible to make most memory accesses fast and still have large memory in computers.

- Taking advantage of **Locality**

  - ***Temporal locality***:

    it will tend to be referenced

    again soon

  - ***Spatial locality***:

    nearby items will tend to be

    referenced soon.

```
For (i=0;i<10;i++) {
     inst. 1
     inst. 2
     ….
     inst. N
}
```

2004 Memory Hierarchy: Apple iMac G5

Managed by compiler

Managed by hardware

Managed by OS, hardware, application

	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency (cycles)	1	3	3	11	88	1e7

iMac G5
1.6 GHz
$1299.00

Goal: Illusion of large, fast, cheap memory

Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access

Outline

5.1 Introduction

5.3 The Basic of Caches

5.4 Measuring and Improving Cache Performance

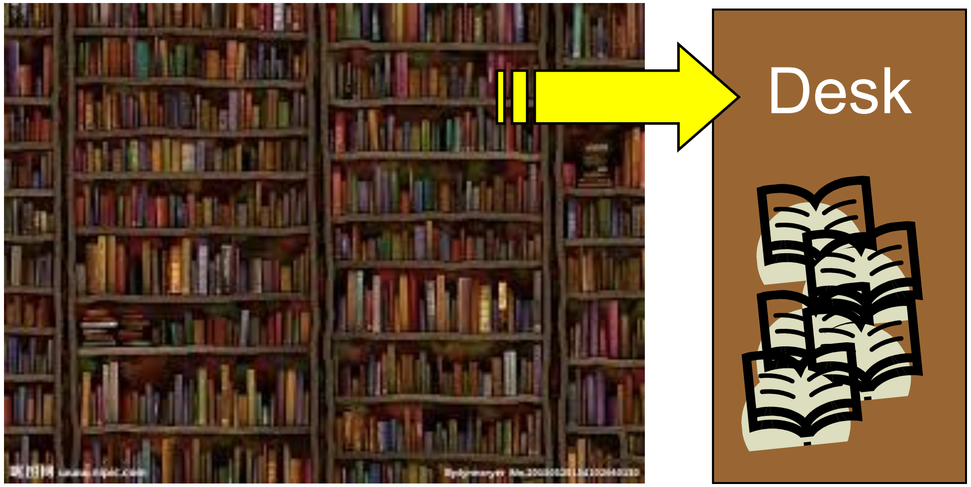
5.7 Virtual Memory

5.8 A Common Framework for Memory Hierarchies

5.10 Parallelism and Memory Hierarchies: Cache Coherence

5.16 Concluding Remarks

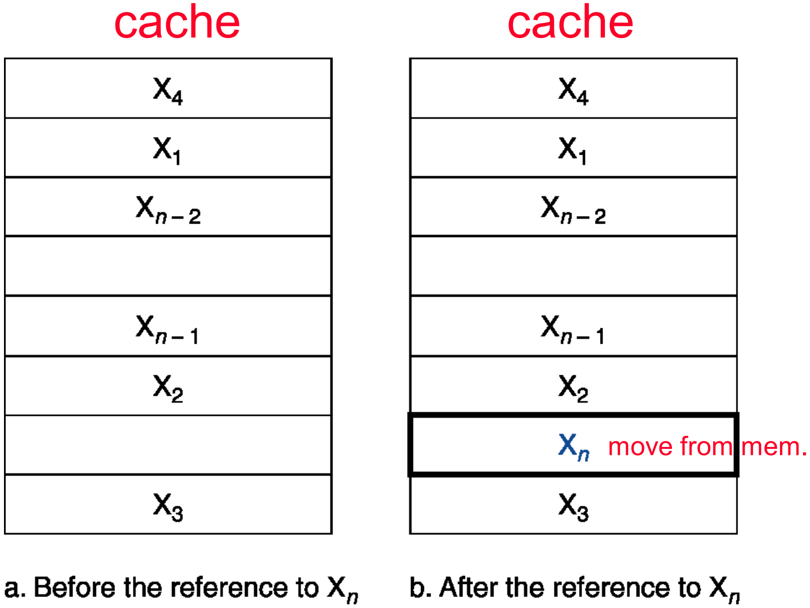
The basic concept of cache



1. Before the request, the cache contains *a collection of recent references* X1, X2, …Xn-1

2. Then, processor requests Xn → ***miss***

3. The word Xn is brought *from memory into cache*

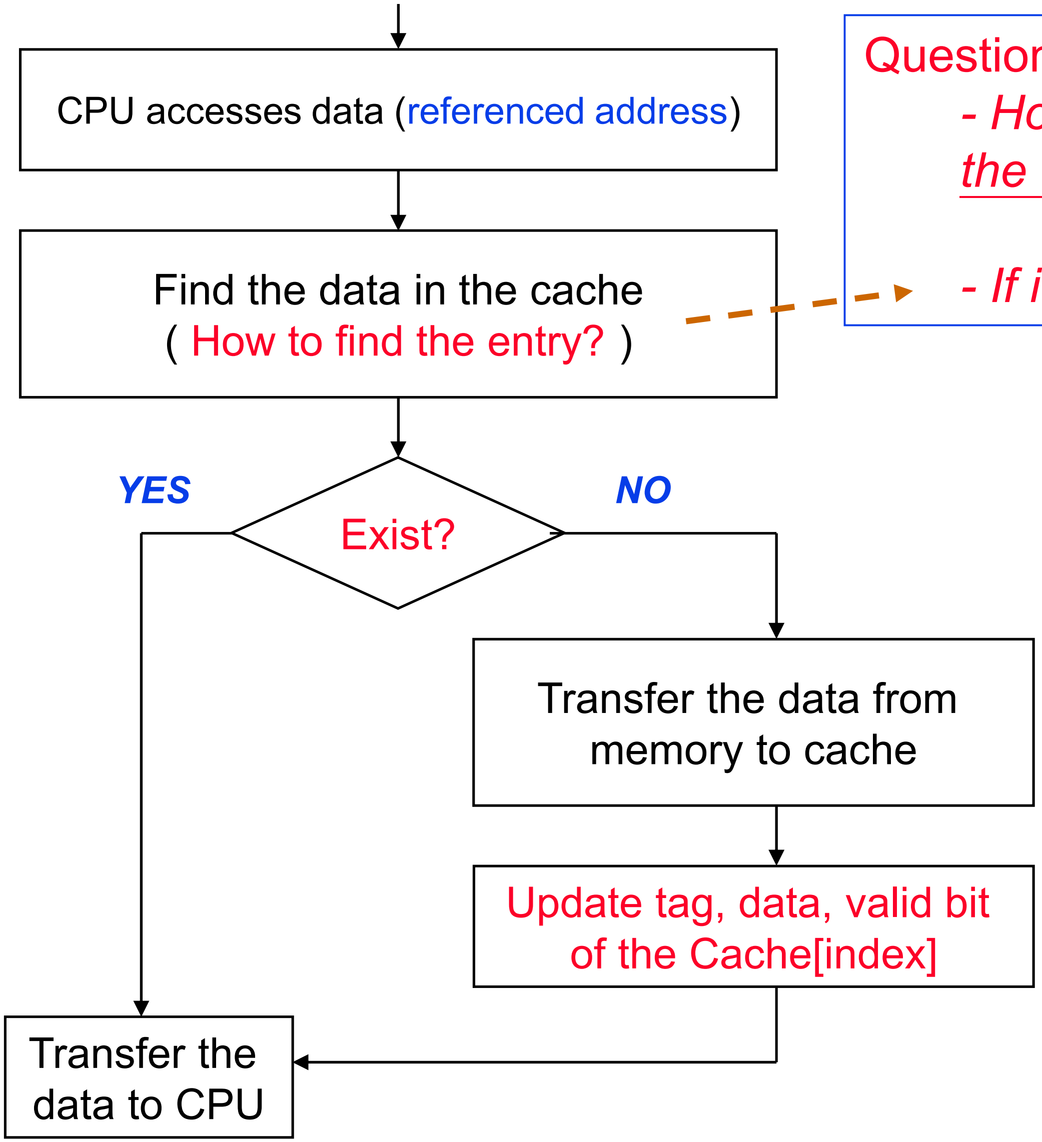


cache	cache
X_4	X_4
X_1	X_1
X_{n-2}	X_{n-2}
X_{n-1}	X_{n-1}
X_2	X_2
	X_n move from mem.
X_3	X_3

a. Before the reference to X_n

b. After the reference to X_n

Cache access flow



CPU accesses data (referenced address)

Find the data in the cache
(How to find the entry?)

Exist?

YES

NO

Transfer the data from memory to cache

Update tag, data, valid bit of the Cache[index]

Transfer the data to CPU

Questions:

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

Direct mapped cache

Direct-mapped Cache

- Accessing a cache - ***direct mapped***
  - a **referenced address** is divided
    - a cache index (***lower bits***)
      - to place the block into cache
    - a tag field (***upper bits***)
      - to identify the block (N:1)
    - a valid bit (***additional bit in Cache***)
      - to validate the block

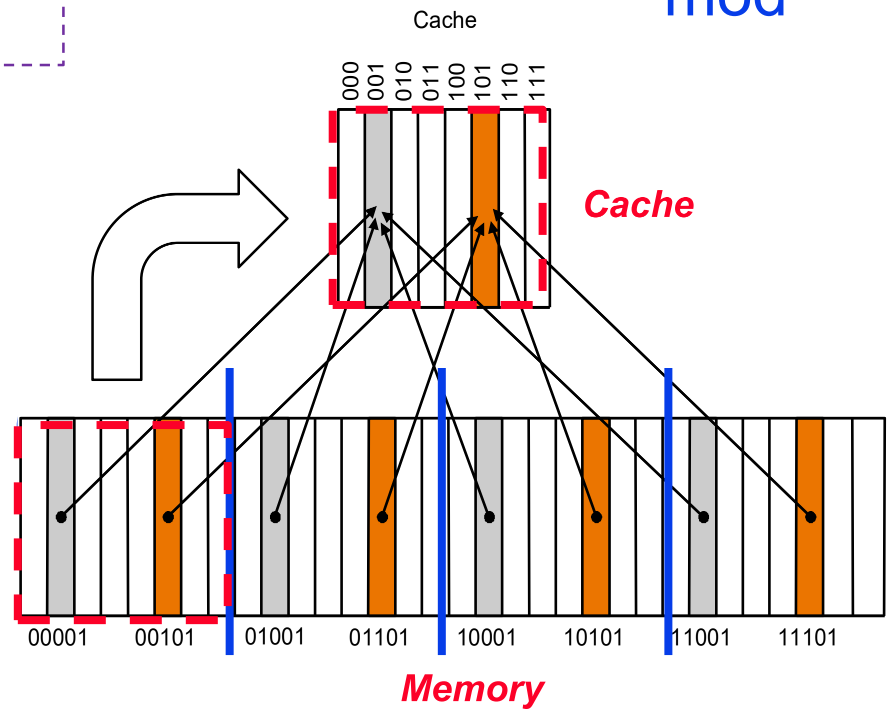
32 words (Mem) ***map to*** *8 entries (Cache)*

5 bits → 3 bits for index + 2 bits for tag

00001
01001
10001
11001

001

mod

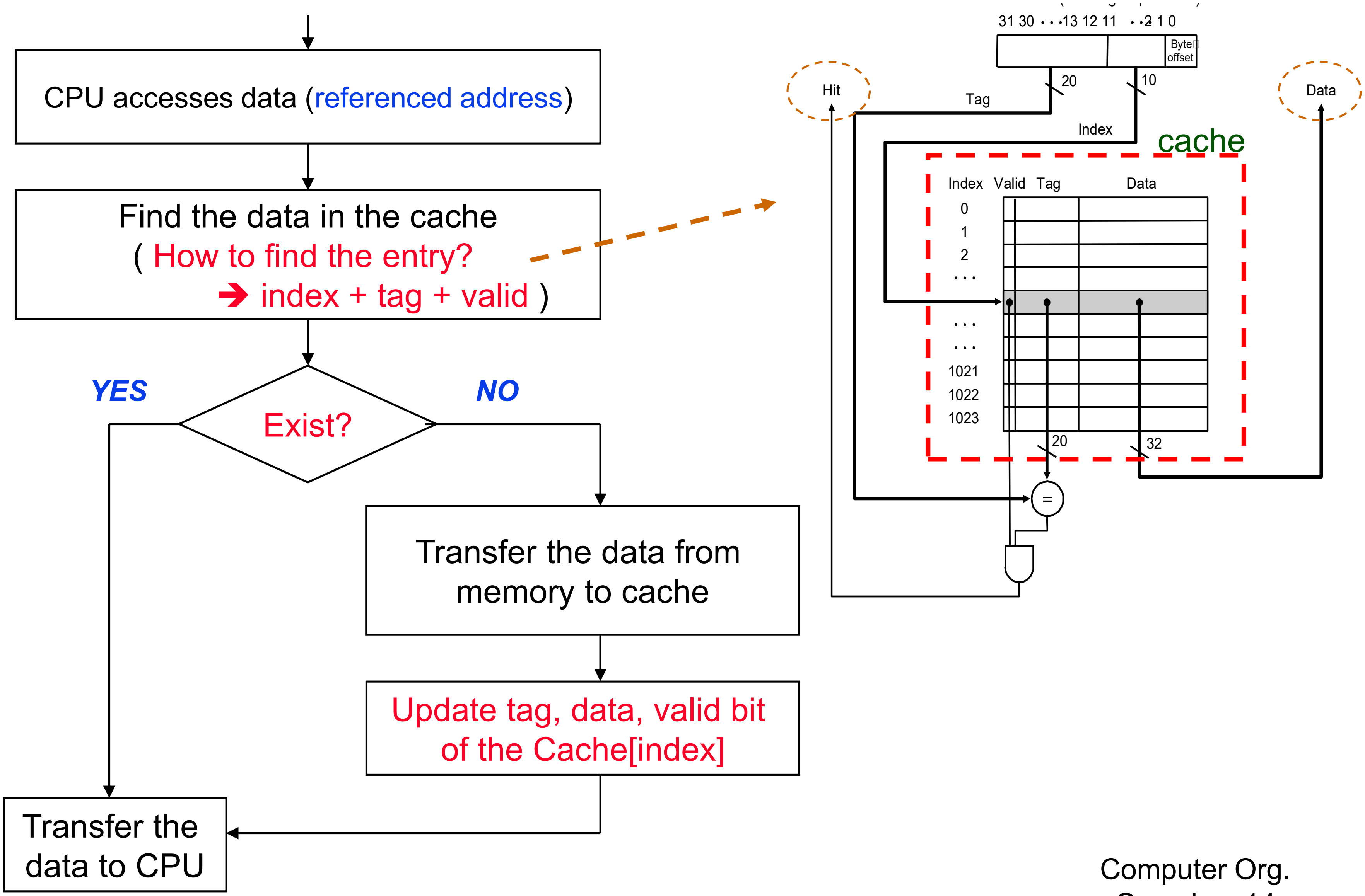


Data address format

Tag	Index	offset

Location in a cache block/line

Direct-mapped Cache



CPU accesses data (referenced address)

Find the data in the cache
(How to find the entry?
➔ index + tag + valid)

Exist?

YES

NO

Transfer the data from memory to cache

Update tag, data, valid bit of the Cache[index]

Transfer the data to CPU

Cache Example (direct mapped cache)

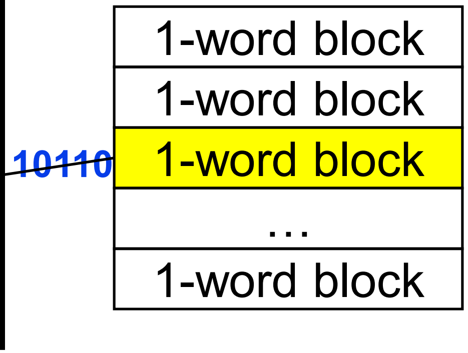
- 8-blocks, 1 byte/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	**Miss**	**110**

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		



Word addr	Binary addr	Hit/miss	Cache block
26	11 010	**Miss**	010

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	**Hit**	110
26	11 010	**Hit**	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	**Miss**	010

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010] (replace 26)**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Size

- *Total bits* of a direct-mapped cache =

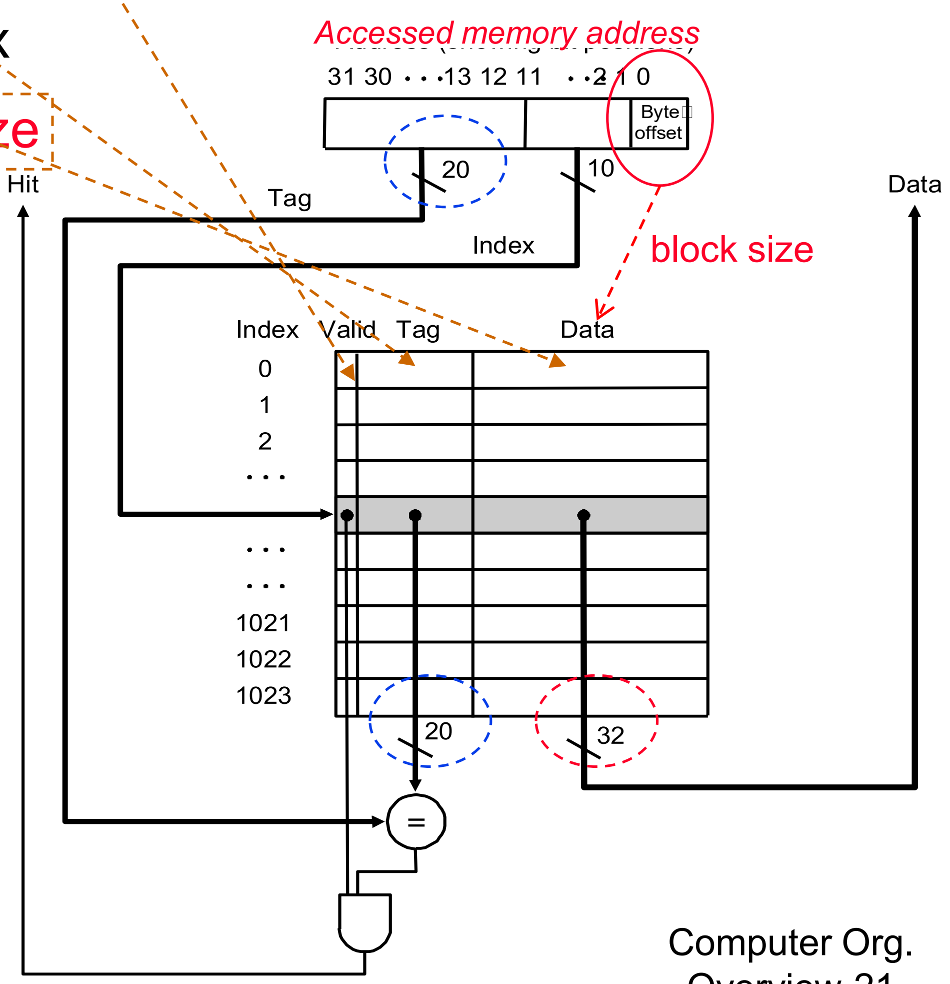
2^n * (block size + tag size + valid field)

- n bits are used for index
- entry number x entry size

Address bit format

↕

Cache size



Example P.378 – Bits in a Cache

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

0	4-word block
1	4-word block
2	4-word block
	…
x=?	4-word block

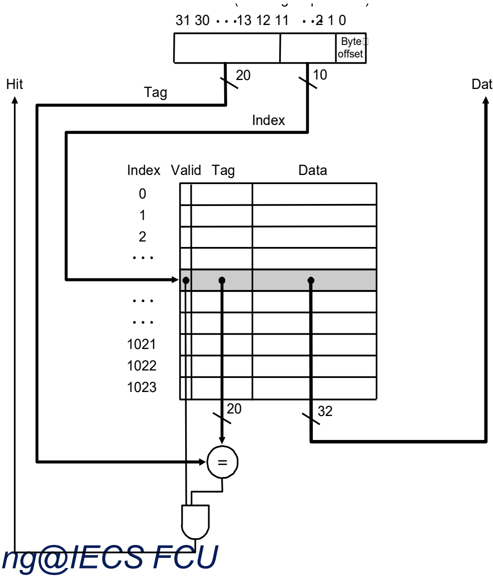
Offset = 4 words ➔ 4 x 4 bytes

block size = 4 x 32 = 128 bits (16 Bytes)

16 KB → 4 K words

x = 4KW / (4) = 1024 = 2^{10} blocks (1 K)

So, **index** needs 10 bits



Tag bits = 32 – 10 – 2 – 2 = 18 bits

Valid bit = 1 bit

4 **words** / block

4 x **4 bytes** / block

Total bits for the cache =

2^{10} x (128 + 18 + 1) = 147 Kbit

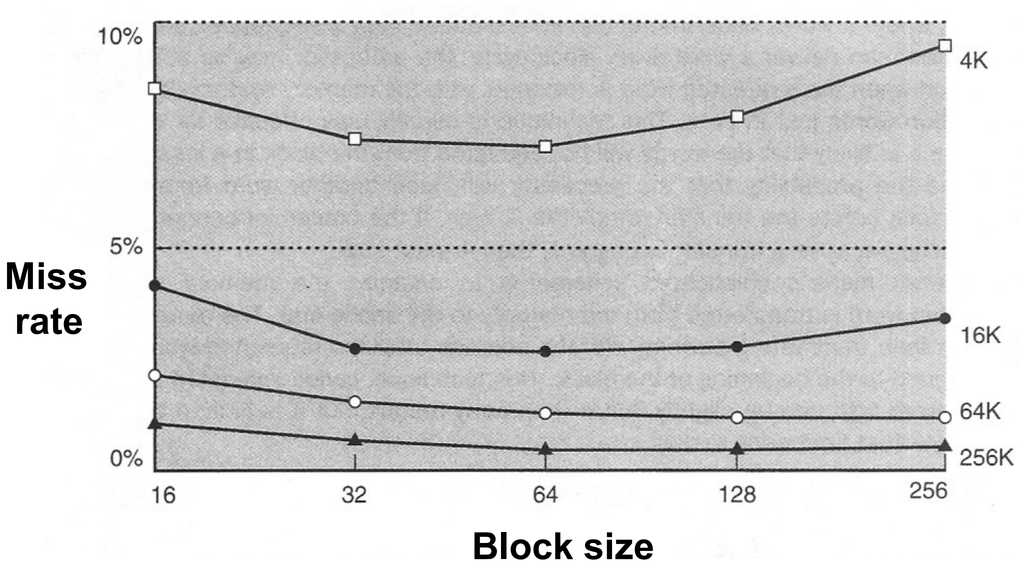
Data + Tag + Valid

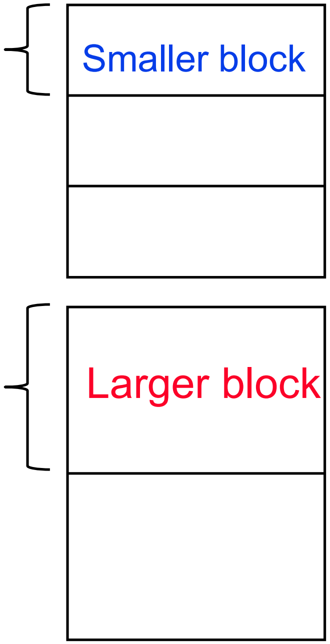
Cache block size VS. Cache entry

+ *Larger blocks* exploit spatial locality to lower miss rates.

- The ***miss rate*** may go up eventually if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the cache will become small, and there will be a *great deal of competition* for those blocks.

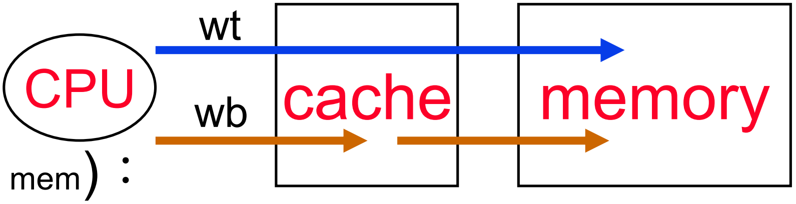
- A more serious problem associated with just increasing the block size is that *the cost of a miss increases*.





Handling Writes

- Inconsistent：
  - When we write the data into only the ***data cache***, ***memory*** would have a different value from that in the cache.
- Approaches to keep the cache and memory consistent
  - ***Write-through***
    - To always write the data into both the memory and the cache.



wt

CPU

wb

cache

memory

    - Low performance
    - Example (10% store, 100 cycle process mem)：
      - CPI (1 → 11) (write memory)
    - Use write buffer to improve performance
  - ***Write-back***
    - When a write occurs, the new value is written only to the block in the cache. *The modified block is written to the lower level of the hierarchy when it is replaced*.

Hits vs. Misses

- Read hits
    - That is good!
- **Read misses**
    - stall the CPU
    - fetch block from memory
    - deliver to cache
    - restart
- Write hits:
    - can replace data in cache and memory (**write-through**)
    - write the data only into the cache (**write-back** the cache later)
- **Write misses**:
    - read the entire block into the cache
    - then write the word

Outline

5.1 Introduction

5.3 The Basic of Caches

5.4 Measuring and Improving Cache Performance

5.7 Virtual Memory

5.8 A Common Framework for Memory Hierarchies

5.10 Parallelism and Memory Hierarchies: Cache Coherence

5.16 Concluding Remarks

Cache Performance Evaluation

- *Simplified Model*

CPU execution time =

(execution cycles + memory stall cycles) x cycle time

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

read-stall cycles = # of reads x miss rate x miss penalty

write-stall cycles =

(# of writes x miss rate x miss penalty) + write buffer stalls

Example P.388 – cache performance

- **Assume**： *loads and stores are 36% in SPECint2000*
  - Inst. cache miss rate = 2%, data cache miss rate = 4%
  - Miss penalty = 100 cycles
  - CPI=2 without any memory stalls
  - *Determine how much faster a processor would run with a perfect cache that never missed???*

Inst. miss (stall) cycles = i x 2% x 100 = 2i
Data miss (stall) cycles = i x **36%** x 4% x 100 = 1.44i
loads and stores are 36% in SPECint2000

Total memory-stall cycles = 2i + 1.44i = 3.44 i (CPI=3.44i/i=3.44)

CPI with memory stall = 2 + 3.44 = 5.44

CPU time with stalls = I x $\mathbf{CPI}_{\mathbf{stall}}$ x clock cycle time
CPU time with **perfect cache** = I x $\mathbf{CPI}_{\mathbf{perfect}}$ x clock cycle time

Performance speedup = (CPU time with stalls) / (CPU time with perfect cache)
= 5.44 / 2 = 2.72 (memory stall is too bad for cpu …)

Example – cache performance

- Assume：Derived from the previous example
  - **Doubling CPU's clock rate**, the main memory speed is unlikely to change
  - *How much faster will the computer be???*

Miss penalty = 200 cycles (because main memory speed is unchanged)
Inst. miss cycles = i x 2% x 200 = 4i
Data miss cycles = i x **36%** x 4% x 200 = 2.88i
loads and stores are 36% in SPECint2000

Total memory-stall cycles = 4i + 2.88i = 6.88 i (CPI=6.88i/i=6.88)

CPI with memory stall = 2 + 6.88 = **8.88**

CPU time with stalls = I x CPI_{stall} x clock cycle time
CPU time with faster CPU = I x CPI_{new_stall} x new clock cycle time

Performance speedup = (CPU time with **slow clock**) / (CPU time with **fast clock**)
= 5.44 / 8.88x(**1/2**) = 1.23 (**NOT** 2 times faster)

Example P.390 – Avg. Memory Access Time

- AMAT = Time for a hit + Miss rate x Miss penalty

- ***Example***:
  - 1 ns clock cycle time
  - Miss penalty = 20 clock cycles
  - Miss rate = 0.05 misses per inst.
  - Cache access time = 1 clock cycle
  - Assume the read and write miss penalties are the same and ignore other write stalls.

AMAT = 1 + 0.05 x 20 = 2 clock cycles (or 2ns)

Improve Cache Performance

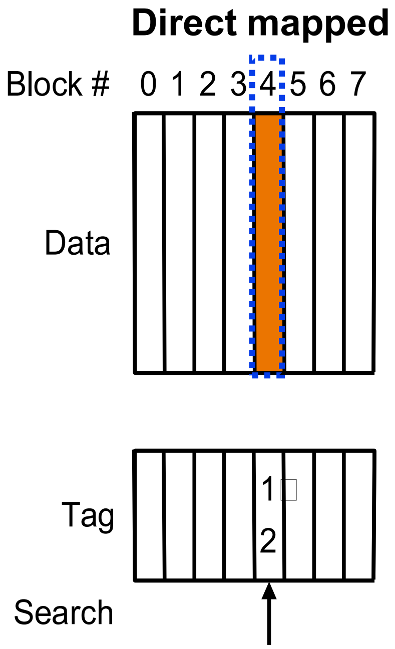
- Reservations from about 2 examples
  - The lower the CPI, the more pronounced the impact of stall cycles (relative penalty will be great)
  - A higher processor clock rate leads to a larger miss penalty

- Two ways of improving performance:
  - decreasing the ***miss rate***
  - decreasing the ***miss penalty*** (improve memory access time)

read-stall cycles = # of reads x miss rate x miss penalty

The problem of direct-mapped cache

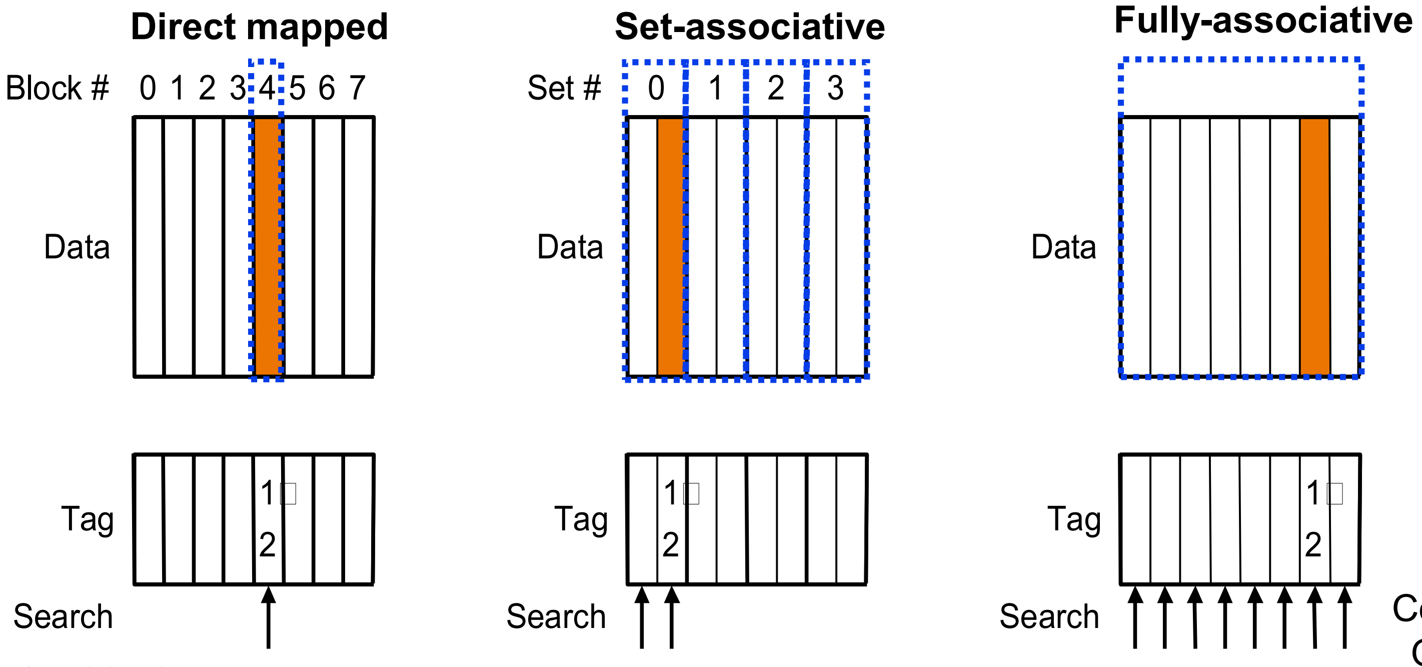
- **Conflictions** for several hot data in the same cache location.
    - Replace data in and out frequently



Decreasing miss ratio with associativity

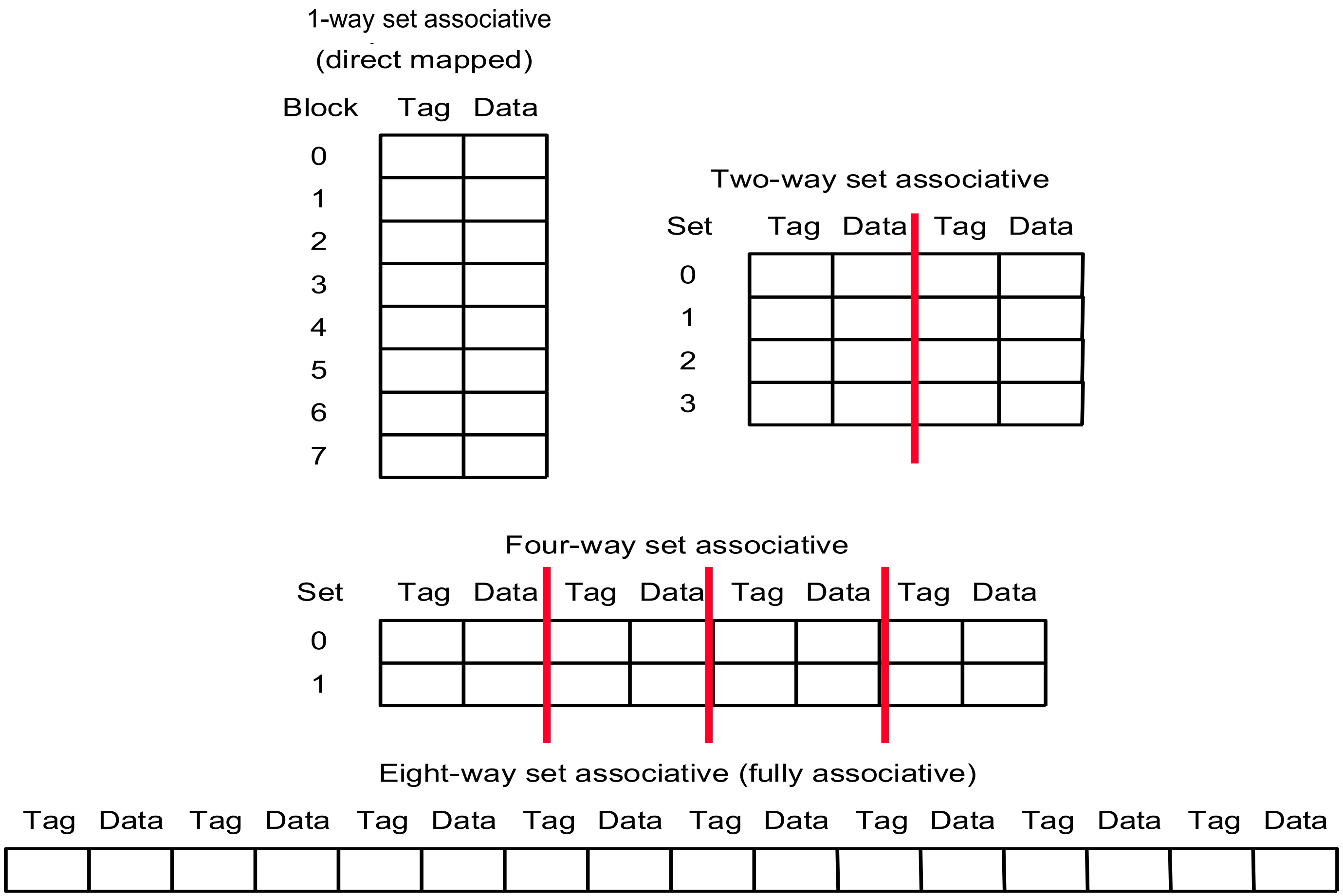
° By more flexible placement of blocks

- *Fully-associative cache*：
  - a block can be placed in any location in the cache.
- *Set-associative cache*：
  - There are a fixed number of locations where each block can be placed.



Set-associative cache

- The position of a memory block is given by：
  - (Memory block number) modulo (Number of cache blocks)
  - *(Memory block number) modulo (Number of **sets** in the cache)*



1-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

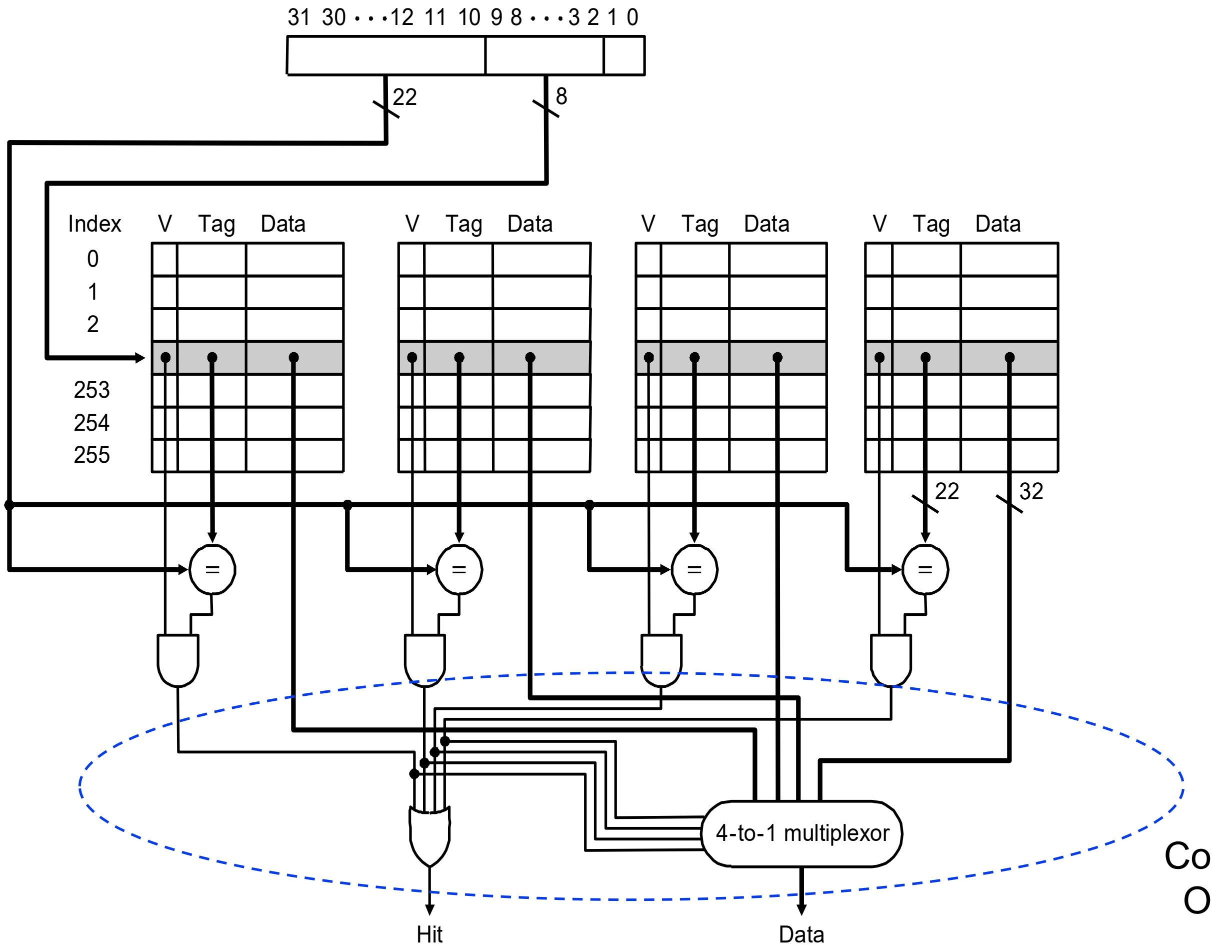
Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

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

Set-associative cache

- The advantage of increasing the degree of associativity is that it usually *decreases the miss rate (more records for the same index)*.
- ***What's the disadvantage?*** **Hit time …**



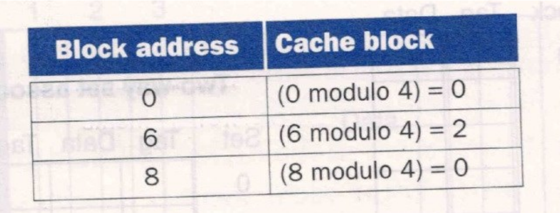
Computer Org. Overview-35

Example P.393

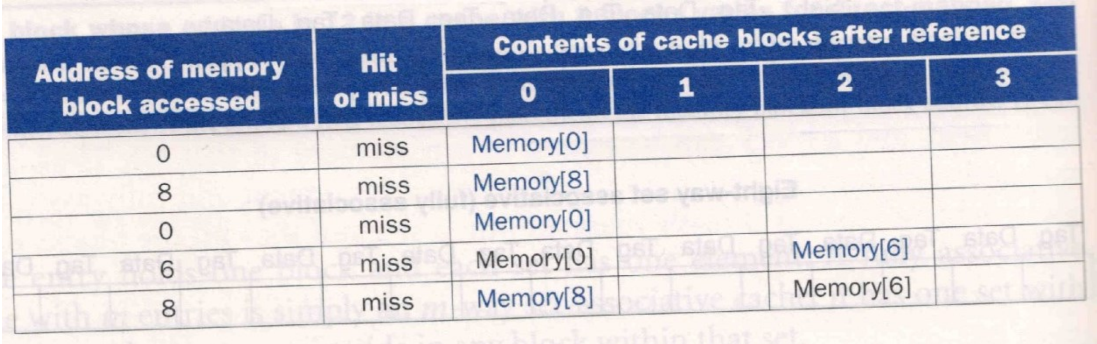
- Cache：4 one-word blocks for 3 caches

- Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, 8

(1). direct mapped：5 misses



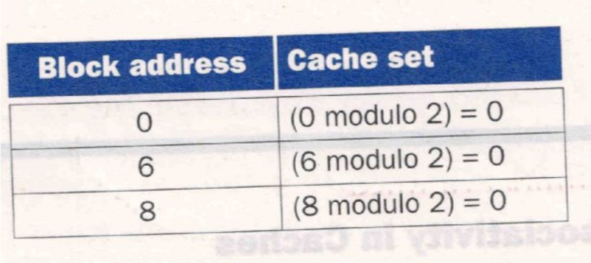
Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0



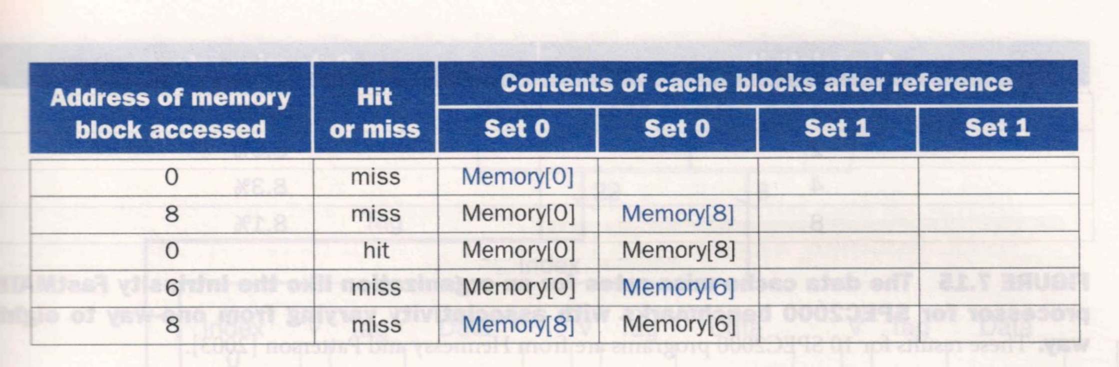
Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

Example P.393

(2). 2-way Set-associative：4 misses

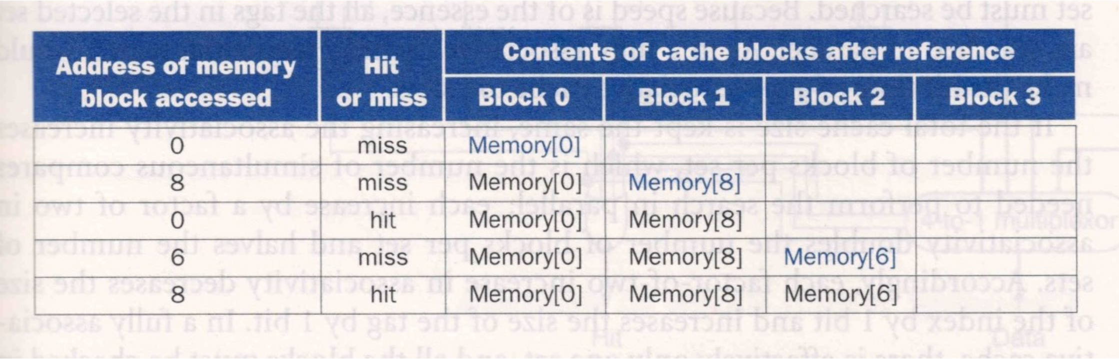


Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0



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]		

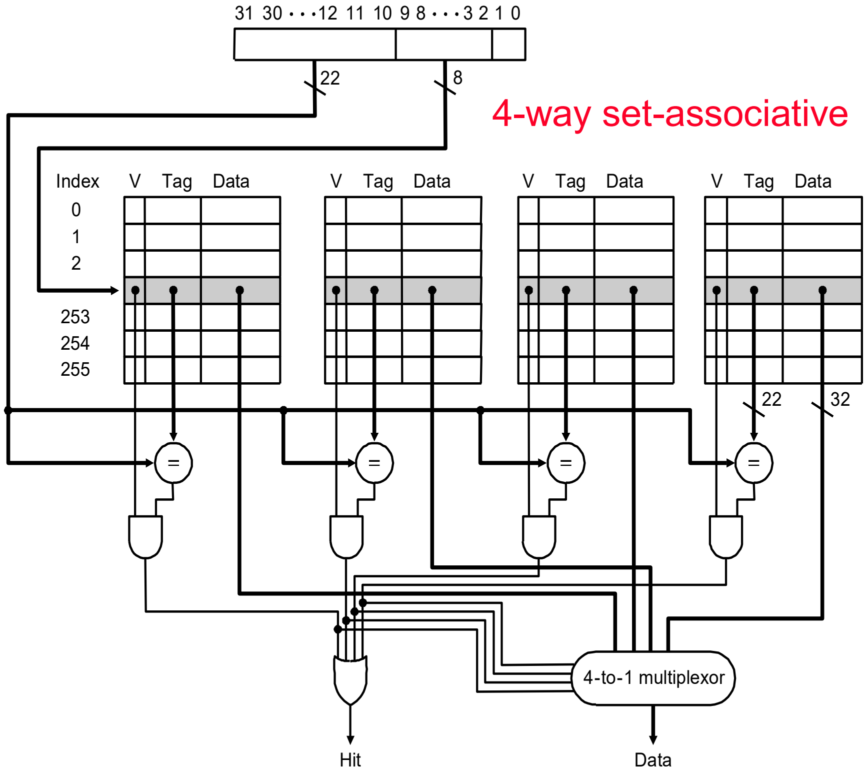
(3). Fully-associative：3 misses



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]	

Locating a Block in the Cache

- Set-associative：
  - *The tag of every cache block within the appropriate set is checked* to see if it matches the block address from the processor.
- Fully-associative：
  - There is effectively only one set, and all the blocks must be checked in parallel.
  - In other words, *we search the entire cache without any indexing*.

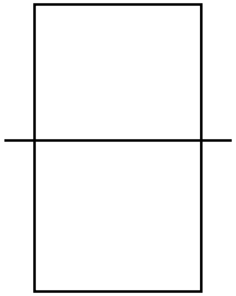


Example P. 397 – Size of Tags vs. Set Associativity

- Assume：32-bit address, 4K cache blocks, 4-word block size
  - Find the total number of sets and the total number of **tag bits** for caches that are directed mapped, 2-way and 4-way set associative, and fully associative.

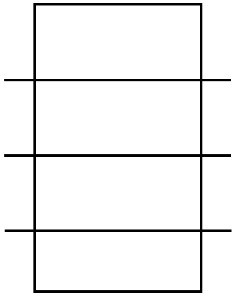
Direct mapped：

block offset = 4 bits (4 word per block)
index bits = 12 (4K blocks)
tag bits = 32 – 4 – 12 = 16
Total number of tag bits = 16 x 4K = 64Kbits



Fully associative：

block offset = 4 (16 bytes per block)
tag bits = 32 – 4 = 28
Total number of tag bits = 28 x 4K = 112Kbits



2-way, (4-way)：

2 set → index bits -1 (4 set → index bits -2)
index bits = 12 – 1 = 11 (10)
tag bits = 32 – 4 – 11(10) = 17 (18)
Total number of tag bits = 17 (18)x 2K x 2 = 68Kbits (72)

Block replacement

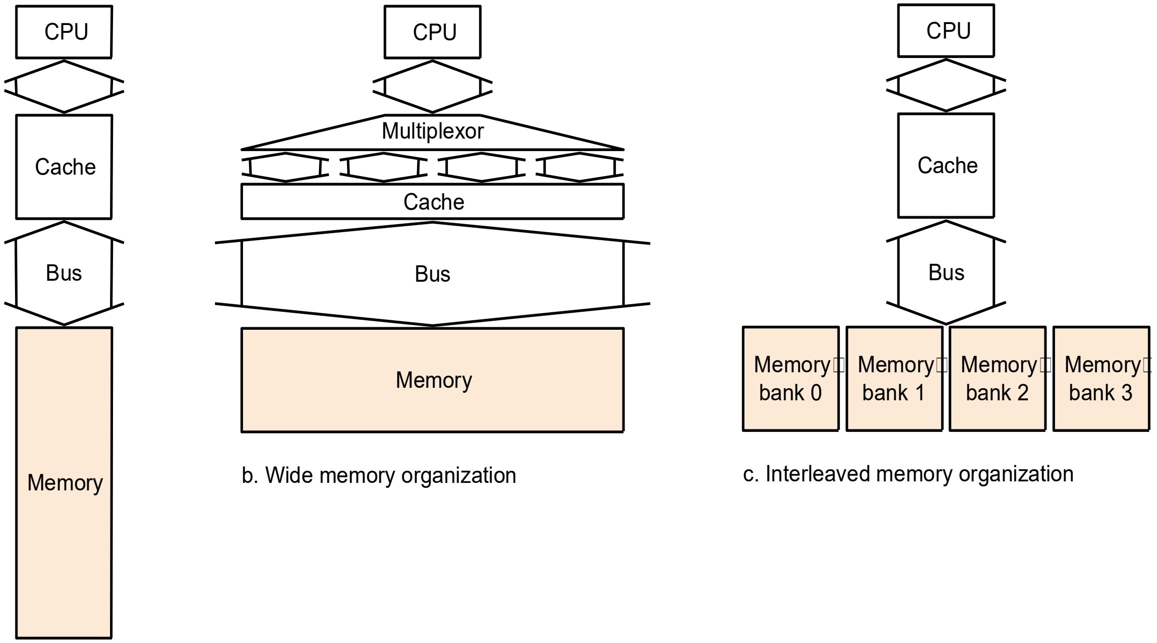
◦ ***Least recently used (LRU)***

- *The block replaced is the one that has been unused for the longest time*.
- Implemented by keeping track of when each element in a set was used relative to the other elements in the set.
- Example：2-way set associative
  - 1 bit in each set and setting the bit to indicate an element whenever that element is referenced.

Designing the memory system to support cache

◦ We can *reduce the* ***miss penalty*** if we increase the bandwidth from the memory to the cache.

- *Widening* the memory and the buses between the processor and memory
- *Interleaving*：sending to multiple banks and read data at the same time



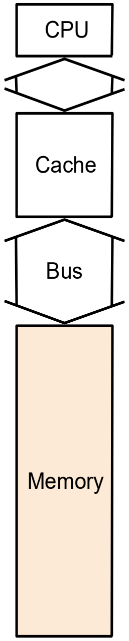
b. Wide memory organization

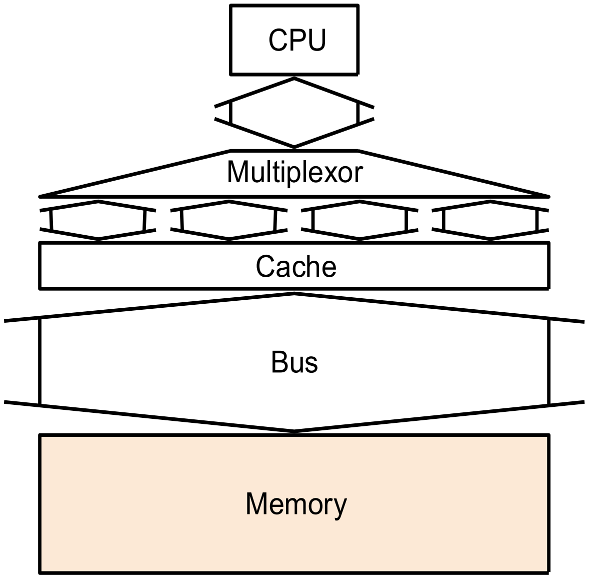
c. Interleaved memory organization

a. One-word-wide memory organization

Increasing Bandwidth – widening bus and memory

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





b. Wide memory organization

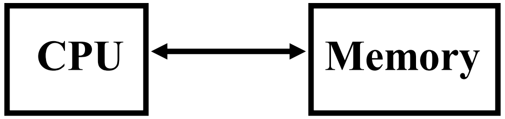
4-word-wide cache block, 1-word-wide bank of DRAM
Miss penalty = 1 + 4x15 + 4x1 = 65 cycles

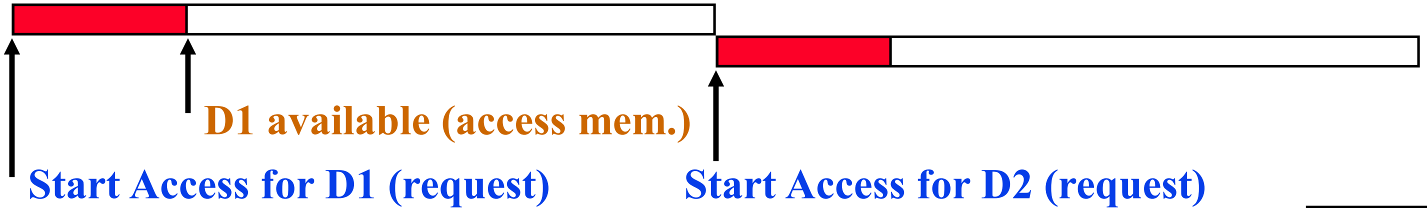
4-word-wide cache block, 2-word-wide bank of DRAM
Miss penalty = 1 + 2x15 + 2x1 = 33 cycles

4-word-wide cache block, 4-word-wide bank of DRAM
Miss penalty = 1 + 1x15 + 1x1 = 17 cycles

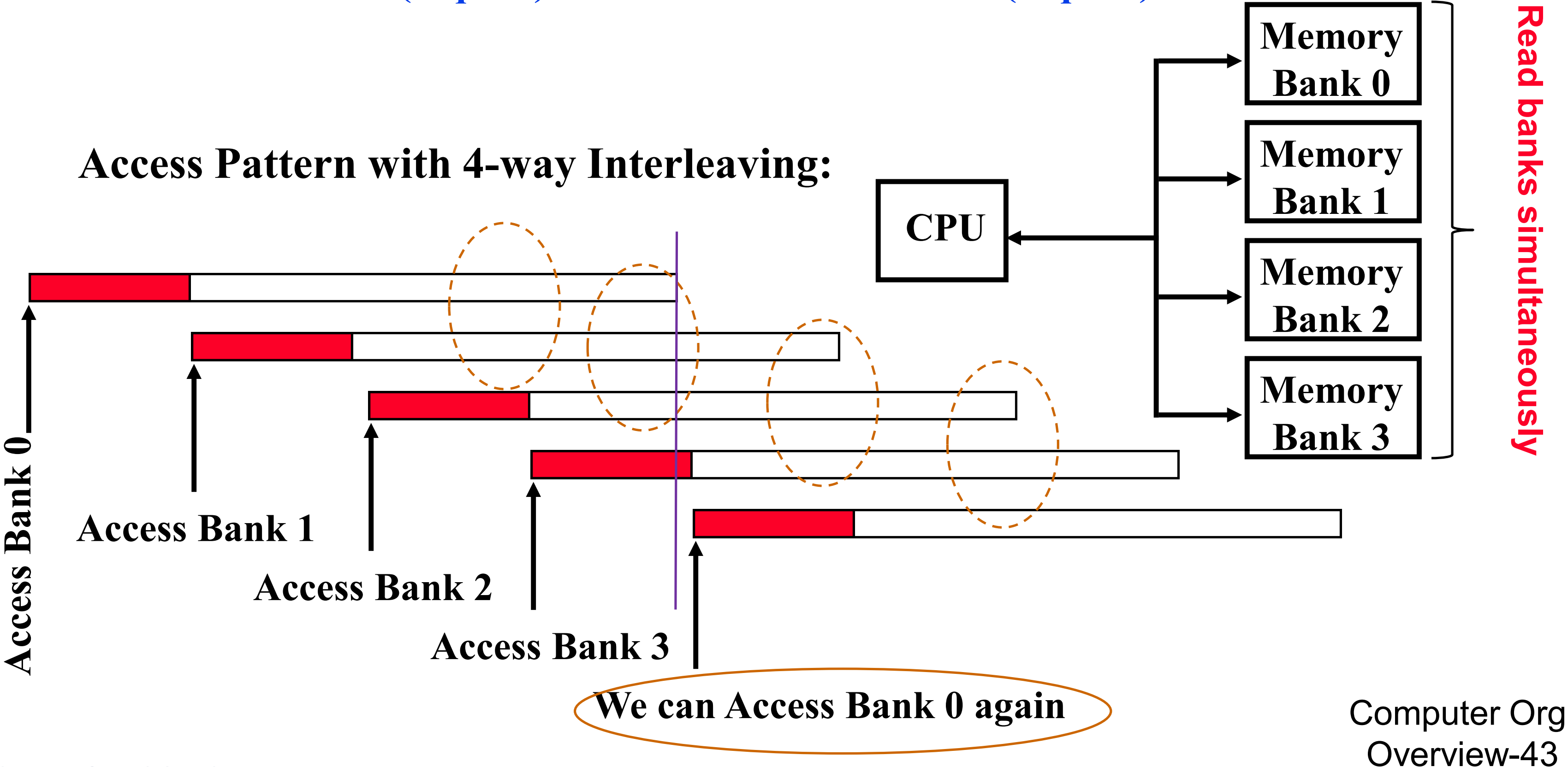
Increasing Bandwidth - Interleaving

Access Pattern without Interleaving:




D1 available (access mem.)
Start Access for D1 (request)
Start Access for D2 (request)

Access Pattern with 4-way Interleaving:



Decreasing miss penalty with multilevel caches

- Add a second level cache (L2 Cache):
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - *miss penalty goes down if data is in 2nd level cache*
- ***Example*** (Page 398):
  - CPI of 1.0 on a 4GHz machine with a 2% miss rate, 100ns DRAM access
  - Adding large enough 2nd level cache with 5ns access time decreases miss rate to 0.5%
  - **Result ➔** The processor with L2 cache is faster by 2.6
- *Optimize two caches separately*
  - 1st cache: unified cache, smaller block size, small associativity
  - 2nd cache: split cache, larger block size, larger associativity

Q & A ?