Chapter Seven

Large and Fast: Exploiting Memory Hierarchy

Qingsong Shi

Email: zjsqs@zju.edu

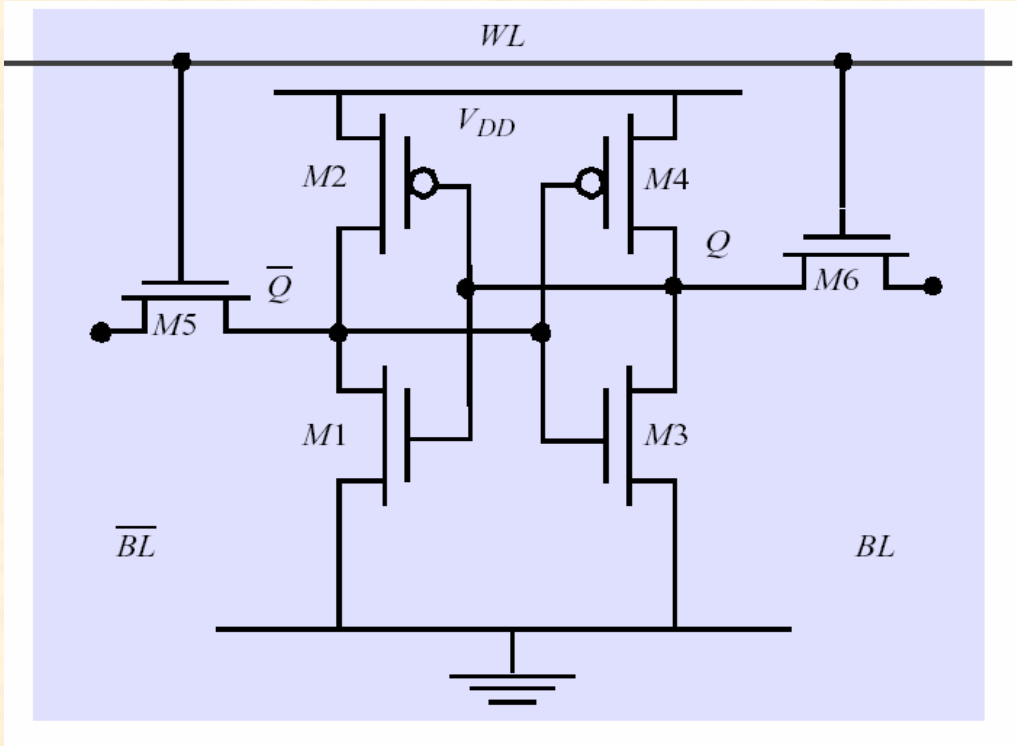
Website: http://zjsqs.hzs.cn

Zhejiang University'2006

7.1 Introduction

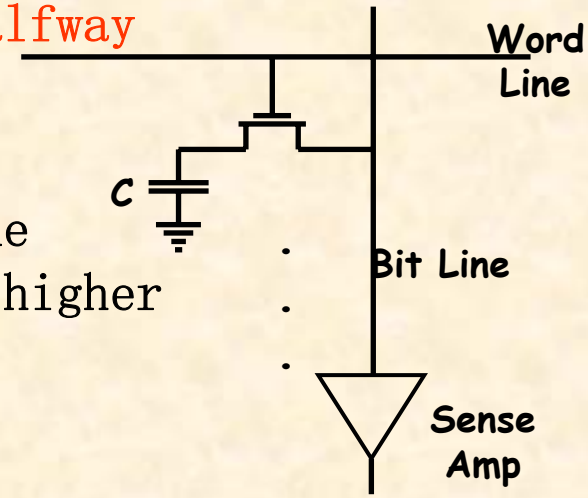
Memories: Review

- **SRAM:**
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)

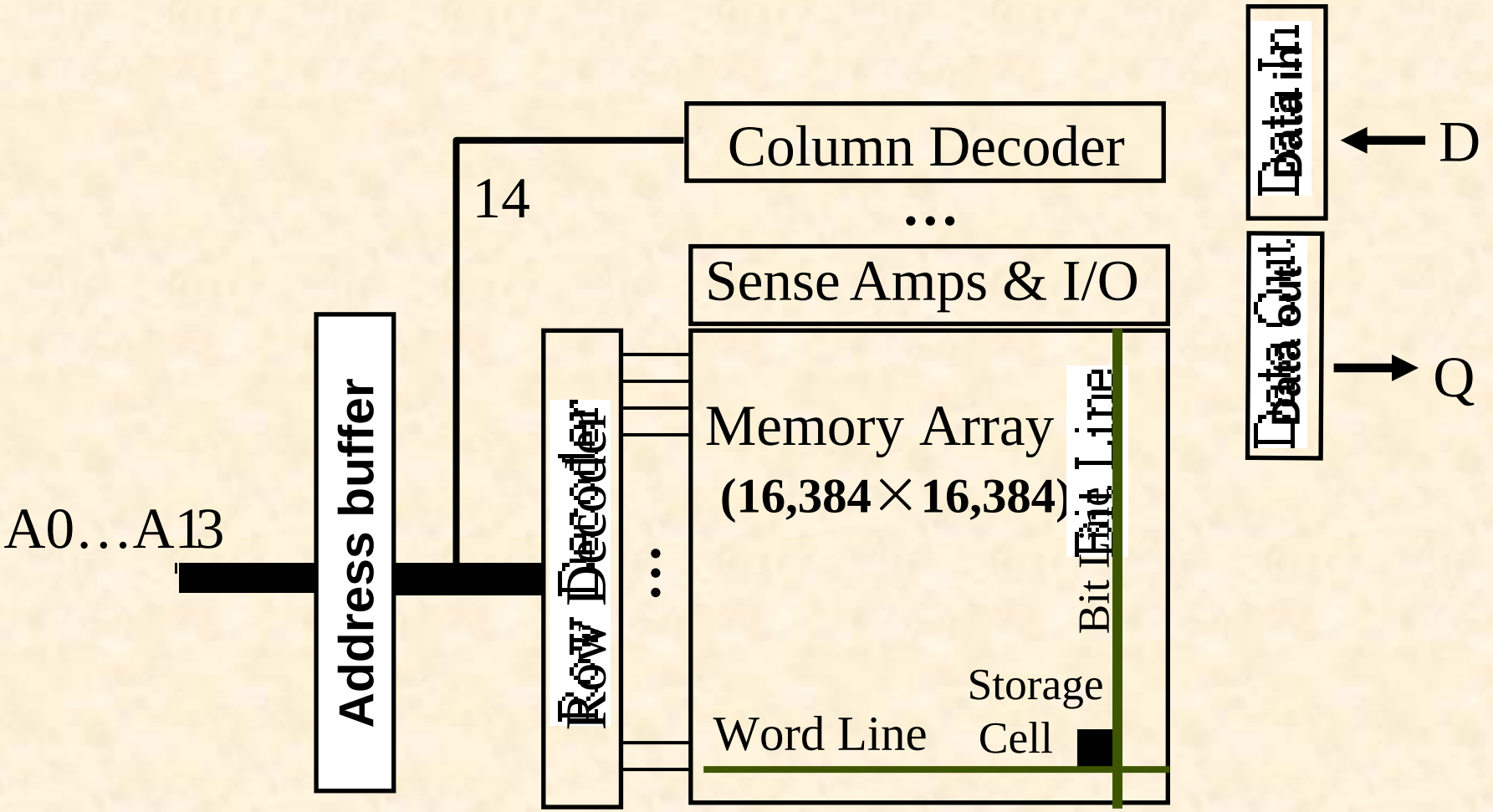


Memories: Review

- **DRAM:**
  - value is stored as a charge on capacitor (must be refreshed)
  - very small but slower than SRAM (factor of 5 to 10)
- **Write**
  - Charge bitline HIGH or LOW and set wordline HIGH
- **Read**
  - Bit line is precharged to a voltage halfway between HIGH and LOW, and then the word line is set HIGH.
  - Depending on the charge in the cap, the precharged bitline is pulled slightly higher or lower.
  - Sense Amp Detects change



DRAM logical organization (64 Mbit)



- **Square root of bits per RAS/CAS**

Problems in memory designing

- **In fact**

Memory technology	Typical access time	Cost per GByte (2004)
SRAM	0.5-5ns	$4000-$10,000
DRAM	50-70ns	$100-$200
Magnetic disk	5,000,000-20,000,000ns	

- **Users want large and fast memories!**

Locality---- two important concepts

1. temporal locality (locality in time):

If an item is referenced, it will tend to be referenced again soon.

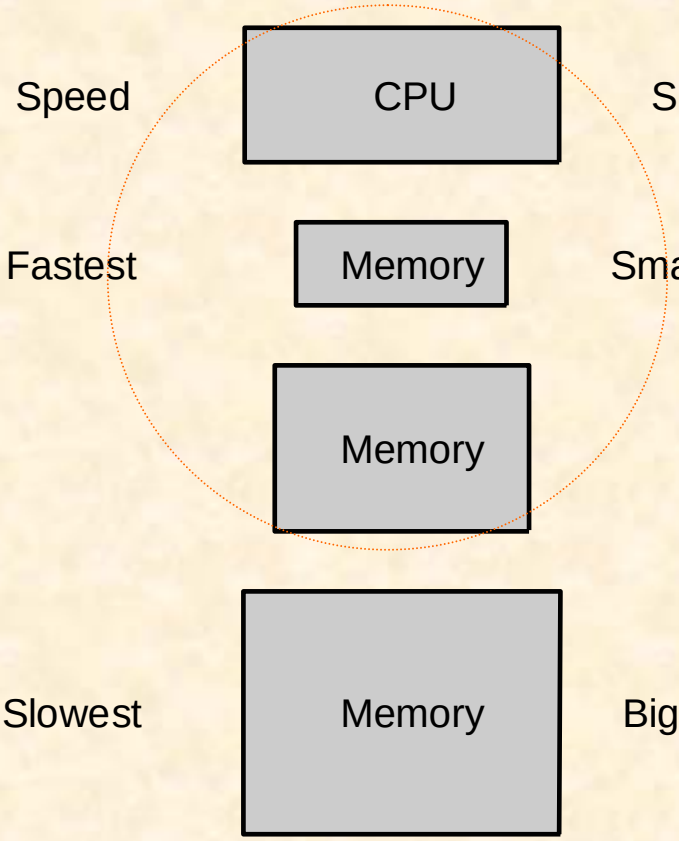
2. spatial locality (locality in space):

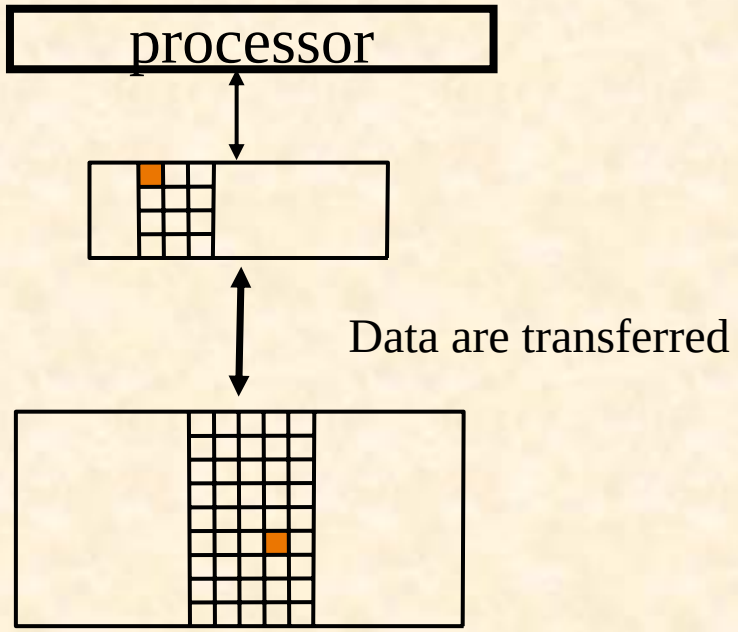
If an item is referenced, items whose addresses are close by will tend to be referenced soon.

- **As we know, these tow principles actually exists in most programs.**
  - *Why does code have locality?*
- **Our initial focus: two levels (upper, lower)**
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level

Solutions

- **Build a memory hierarchy**





Some important items

hit: **The CPU accesses the upper level and succeeds.**

Miss: **The CPU accesses the upper level and fails.**

Hit time:

The time to access the upper level of the memory hierarchy, which includes the time needed to determine whether the access is a hit or a miss.

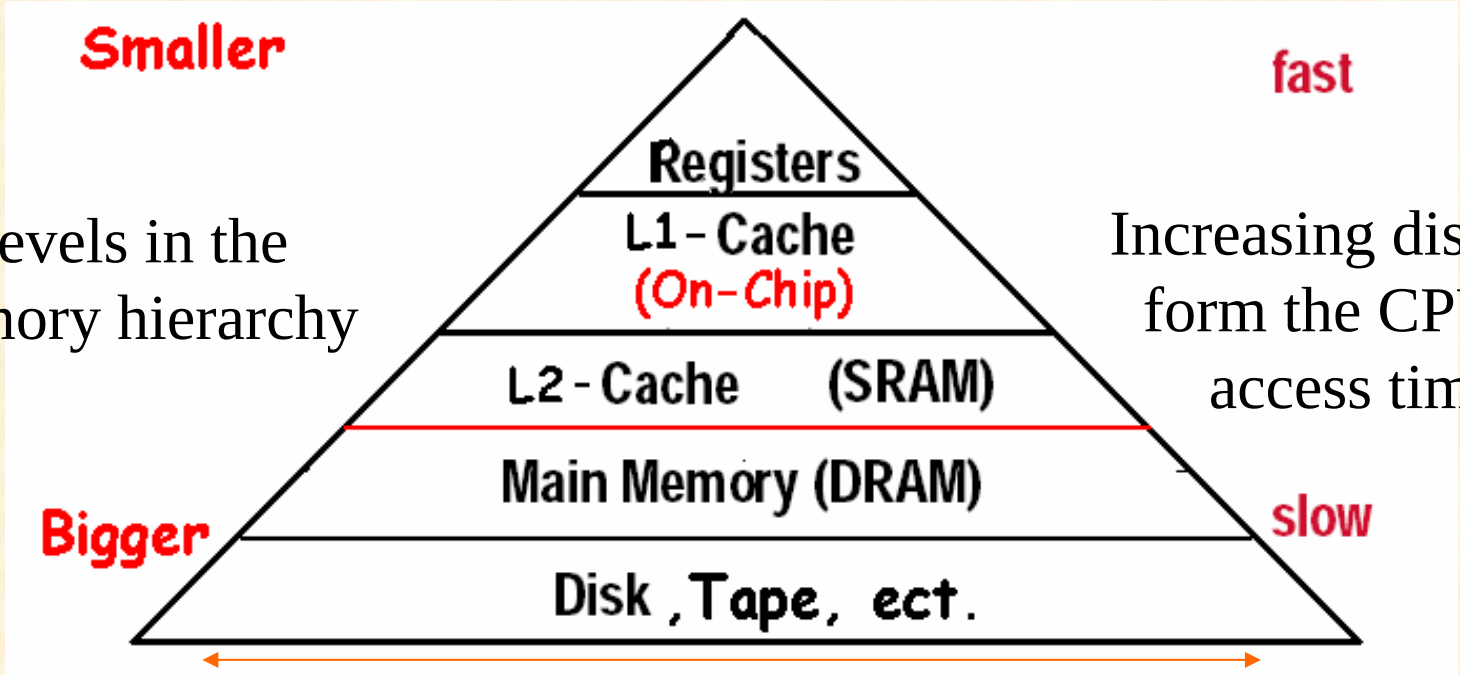
miss penalty:

The time to replace a block in the upper level with the corresponding block from the lower level, plus the time to deliver this block to the processor.

Exploiting Memory Hierarchy

The method

- Hierarchies bases on memories of different speeds and size
  - The more closely CPU the level is,the faster the one is.
  - The more closely CPU the level is,the smaller the one is.
  - The more closely CPU the level is,the more expensive



There has been exploited Memory Hierarchy

1. The basics of Cache: SRAM and DRAM (main memory)

The solution is in speed

2. Visual Memory: DRAM and DISK

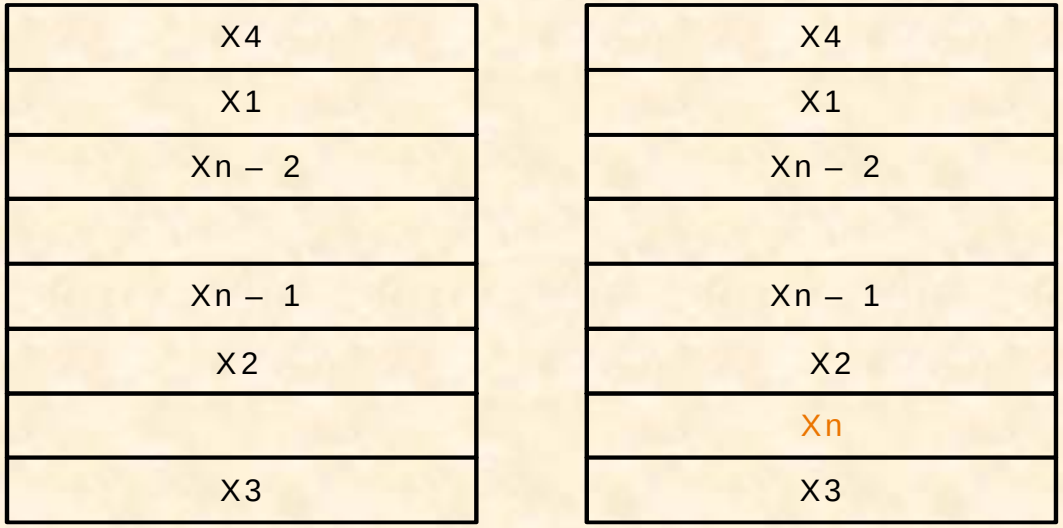
The solution is in size

7.2 The basics of Cache

Simple implementations

• For each item of data at the lower level, there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level



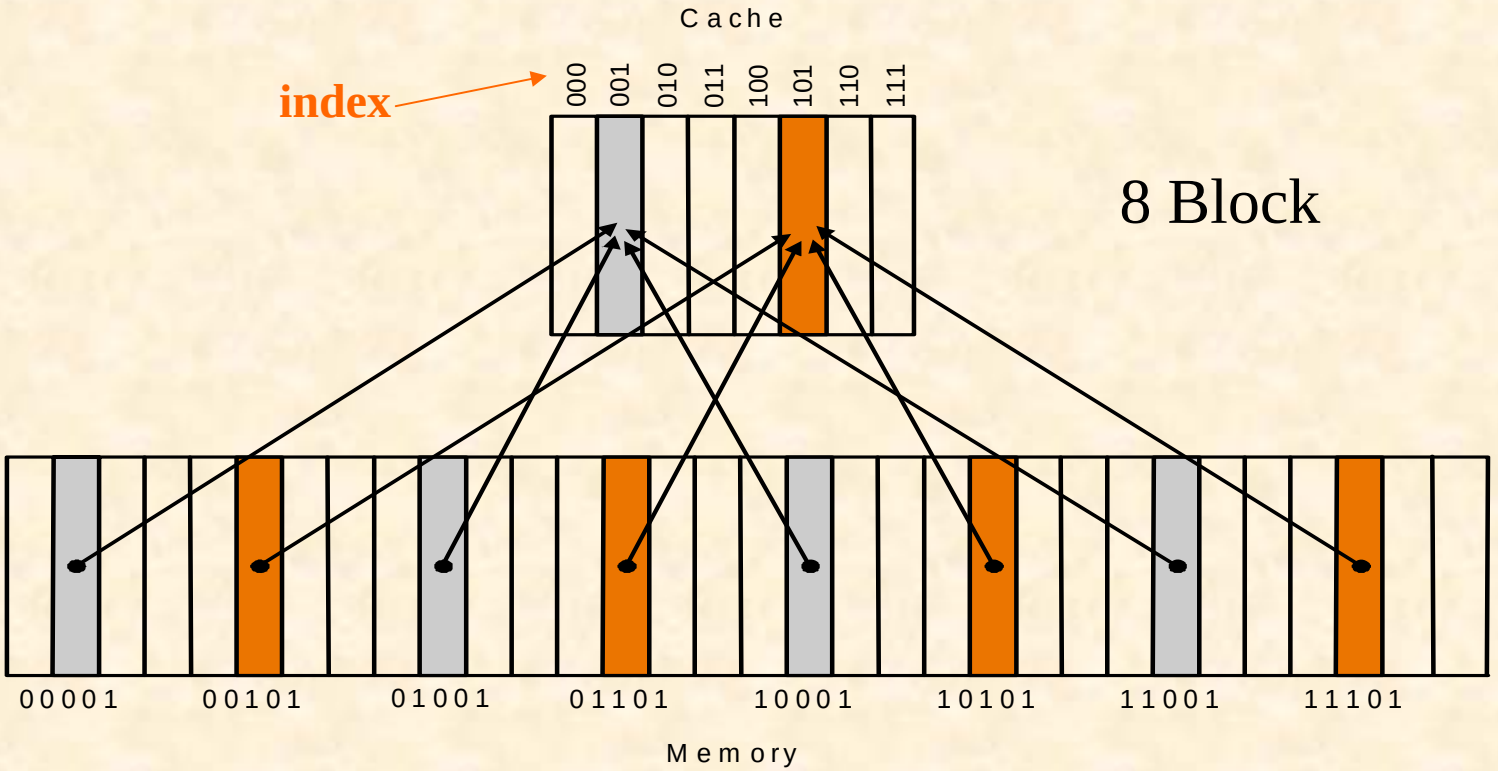
a. Before the reference to Xn

b. After the reference to Xn

- **Two issues:**
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- **Our first example: "direct mapped"**
  - block size is one word of data

Direct Mapped Cache

- **Where can a block be placed in the upper level?**



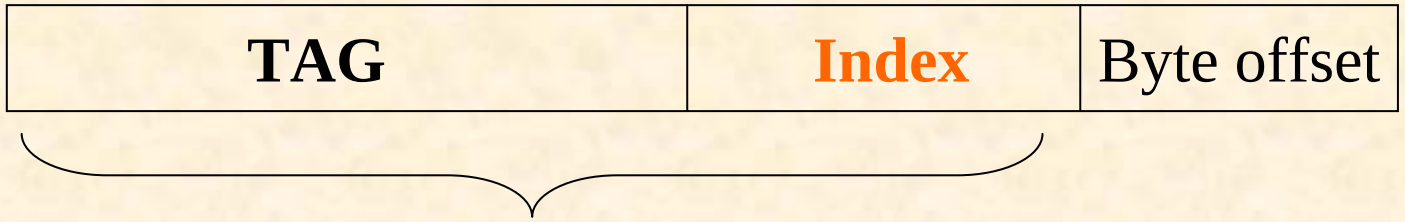
- Direct-mapping algorithm.

  memory address is modulo the number of blocks in the cache

- Fortunately, while the cache has 2^n blocks, the orresponding index is equal to the lowest n bits of memory block address. Here n=3. Let's check

Accessing a cache---how do we find it?

- **Memory block address is larger than cache block address**



valid bit

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

a. The initial state of the cache after power-on

Access sequence

- 10110,11010,10110,11010,10000,00011,10000,10010

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

b. After handling a miss of address(10110)

Index	V	Tag	Data
000	N		
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

c. After handling a miss of address(11010)

Index	V	Tag	Data
000	N		
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

d. After handling a hit of address(10110)

Index	V	Tag	Data
000	N		
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

e. After handling a hit of address(11010)

Access sequence-2

- 10110,11010,10110,11010,10000,00011,10000,10010

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

f. After handling a miss of address(10000)

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11)_2$	Memory(11010)
011	Y	$(00)_2$	Memory(00011)
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

g. After handling a miss of address(00011)

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11)_2$	Memory(11010)
011	Y	$(00)_2$	Memory(00011)
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

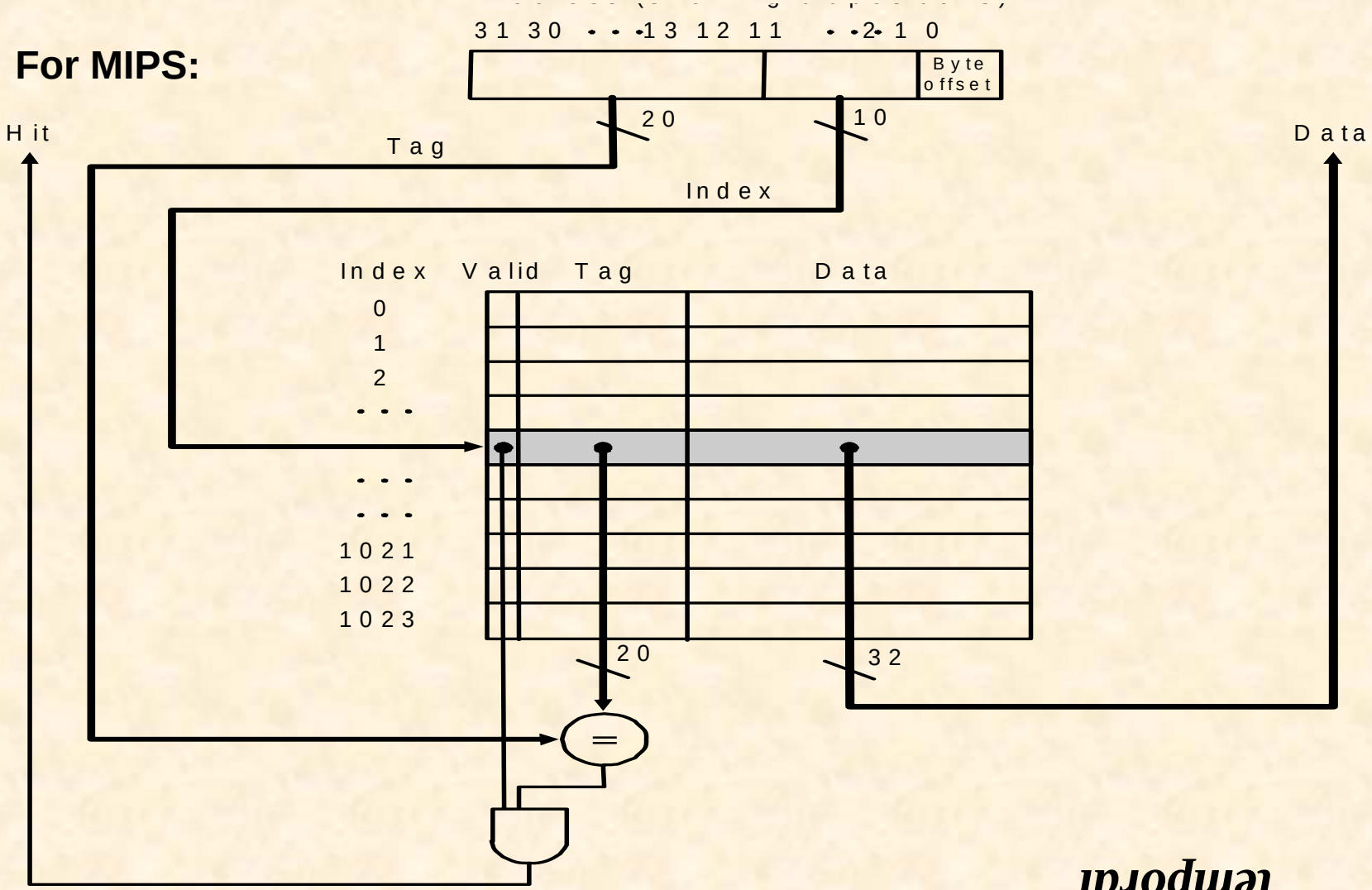
h. After handling a hit of address(10000)

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11) \rightarrow (10)_2$	Memory(10010)
011	Y	$(00)_2$	Memory(00011)
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

i. After handling a miss of address(10010)

Direct Mapped Cache construction

- **For MIPS:**



temporal

What kind of locality are we taking advantage of?

Bits in Cache

Example

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

Answer

- 16KB=4KWord=2^{12} words
- One block=4 words = 2^2 words
- Number of blocks (index bit) = $2^{12} \div 2^2 = 2^{10}$ blocks
- Data bits of block =4×32=128 bits
- Tag bits = address – index-block size =32-10–2-2 =18 bits
- Valid bit = 1 bit

- Total Cache size = $2^{10} \times (128+18+1) = 2^{10} \times 147$= 147 Kbits

  = 18.4KB
- It is about 1.15 times as many as needed just for the data

Mapping an Address to Multiword Cache Block

Example

- **Consider a cache with 64 blocks and a block size of 16 bytes.**
- **What block number does byte address 1200 map to?**

Answer

(Block address) modulo (Number of cache blocks)

Where the address of the block is

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor = \left\lfloor \frac{1200}{16} \right\rfloor = 75$$

75 modulo 64 =11

Notice!!!

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Byte per block} \longleftrightarrow \left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Byte per block} + (\text{Byte per block} - 1)$$

Here: 1200 ⟷ 1215

Handling Cache reads hit and Misses

- **Read hits**
  - this is what we want!
- **Read misses—two kinds of misses**
  - instruction cache miss
  - data cache miss
- **let's see main steps taken on an instruction cache miss**
  - stall the CPU, fetch block from memory, deliver to cache, restart CPU read

1. Send the original PC value (current PC-4) to the memory.

2. Instruct main memory to perform a read and wait for the memory to complete its access.

3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field,and turning the valid bit on.

4. Restart the instruction execution at the first step, which will refetch the instruction again, this time finding it in the cache.

Handling Cache Writes hit and Misses

- **Write hits:** Difference Strategy
  - write-back: Cause Inconsistent
    - Wrote the data into only the data cache
    - Strategy ---- write back data from the cache to memory later Fast
  - write-through: Ensuring Consistent
    - Write the data into both the memory the cache
    - Strategy ---- writes always update both the cache and the memory
    - Slower----write buffer

- **Write misses:**
  - read the entire block into the cache, then write the word

Deep concept in Cache

Four Questions for Memory Hierarchy Designers

Caching is a general concept used in processors, operating systems, file systems, and applications.

There are Four Questions for Memory Hierarchy Designers

- **Q1: Where can a block be placed in the upper level?**
  (Block placement)
  - Fully Associative, Set Associative, Direct Mapped
- **Q2: How is a block found if it is in the upper level?**
  (Block identification)
  - Tag/Block
- **Q3: Which block should be replaced on a miss?**
  (Block replacement)
  - Random, LRU,FIFO
- **Q4: What happens on a write?**
  (Write strategy)
  - Write Back or Write Through (with Write Buffer)

Q1: Block Placement

- **Direct mapped**
  - Block can only go in one place in the cache

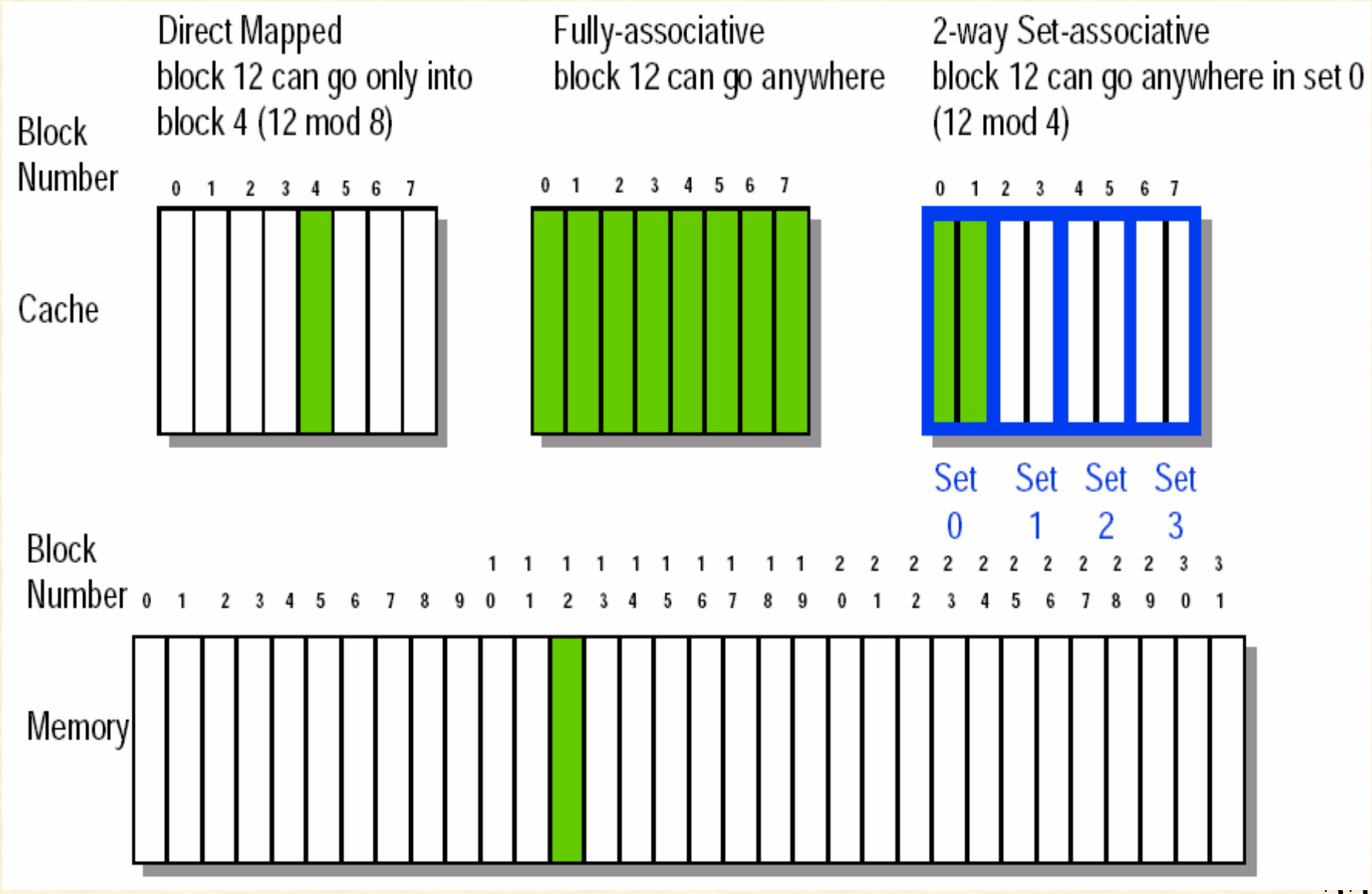
    Usually address MOD Number of blocks in cache
- **Fully associative**

  Block can go anywhere in cache.
- **Set associative**
  - Block can go in one of a set of places in the cache.
  - A set is a group of blocks in the cache.

    Block address MOD Number of *sets* in the cache
  - If sets have n blocks, the cache is said to be n-way set associative.

•Note that direct mapped is the same as 1-way set associative, and fully associative is m-way set-associative (for a cache with m blocks).

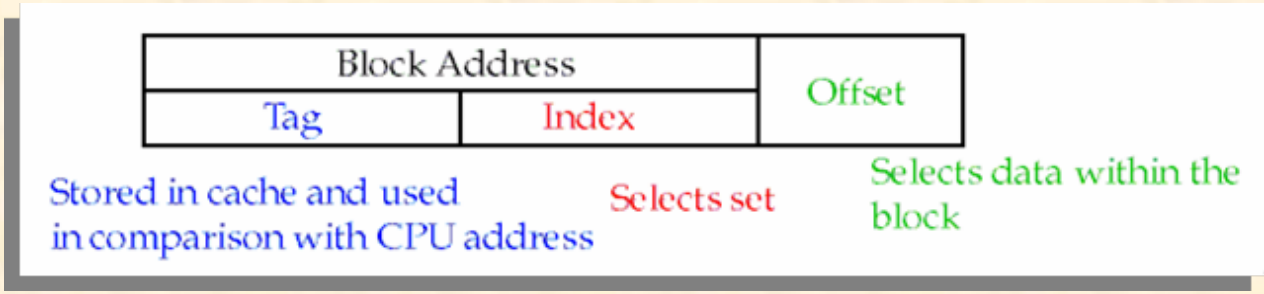
Figure 8-32 Block Placement



Q2: Block Identification

- Every block has an address tag that stores the main memory address of the data stored in the block.
- When checking the cache, the processor will compare the requested memory address to the cache tag -- if the two are equal, then there is a cache hit and the data is present in the cache
- Often, each cache block also has a valid bit that tells if the contents of the cache block are valid

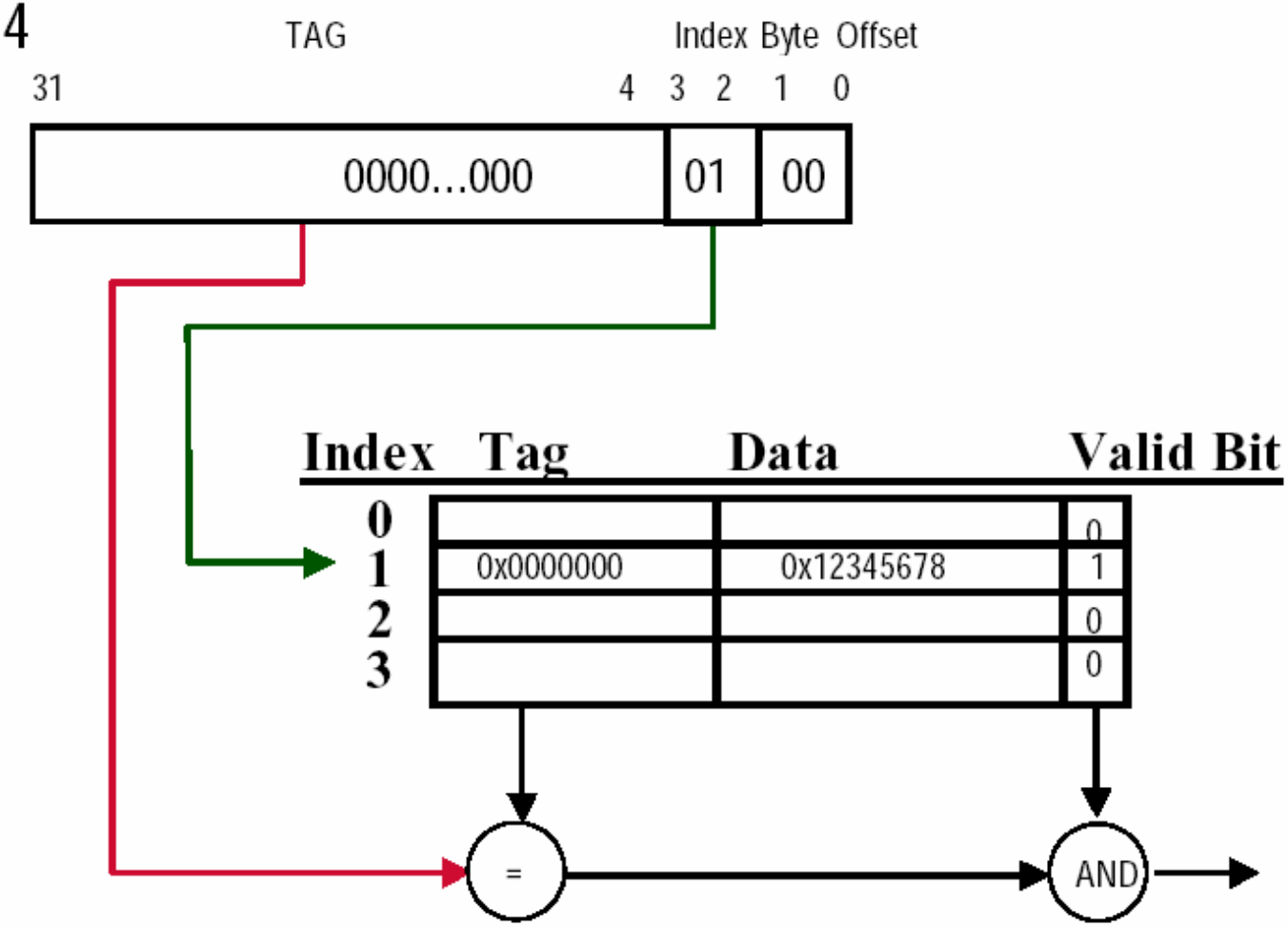
The Format of the Physical Address



- **The Index field selects**
  - The set, in case of a set-associative cache
  - The block, in case of a direct-mapped cache
  - Has as many bits as log2(#sets) for set-associative caches, or log2(#blocks) for direct-mapped caches
- **The Byte Offset field selects**
  - The byte within the block
  - Has as many bits as log2(size of block)
- **The Tag is used to find the matching block within a set or in the cache**
  - Has as many bits as Address_size - Index_size - Byte_Offset_Size

Direct-mapped Cache Example (1-word Blocks)

LOAD R1, 0x04

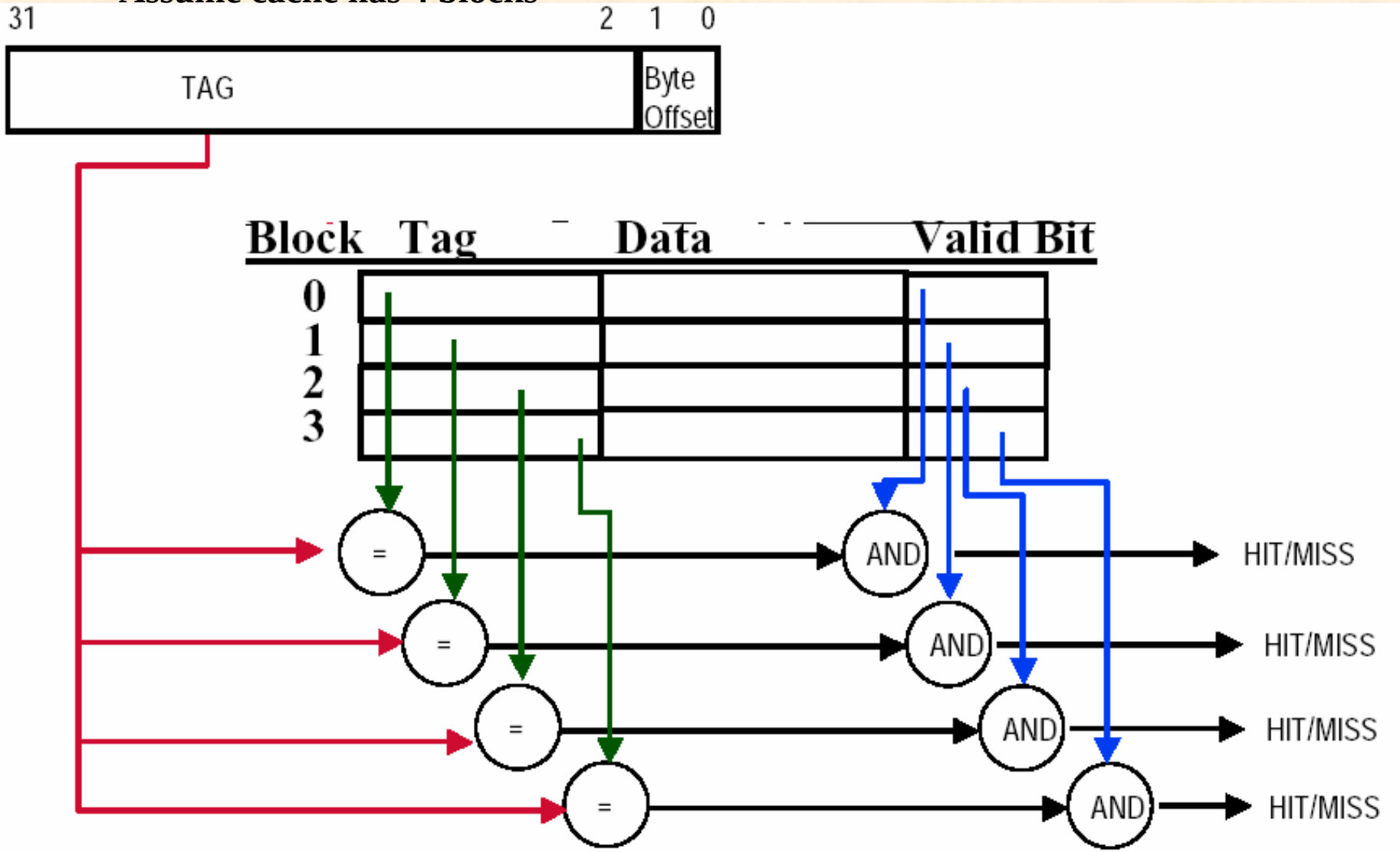


MEMORY

Address	Data
0x00	0x00000000
0x04	0x12345678
0x08	0x87654321
0x0C	0x11111111
0x10	0x22222222
0x14	0x33333333
0x18	0x44444444
0x1C	0x55555555
0x20	0x10101010

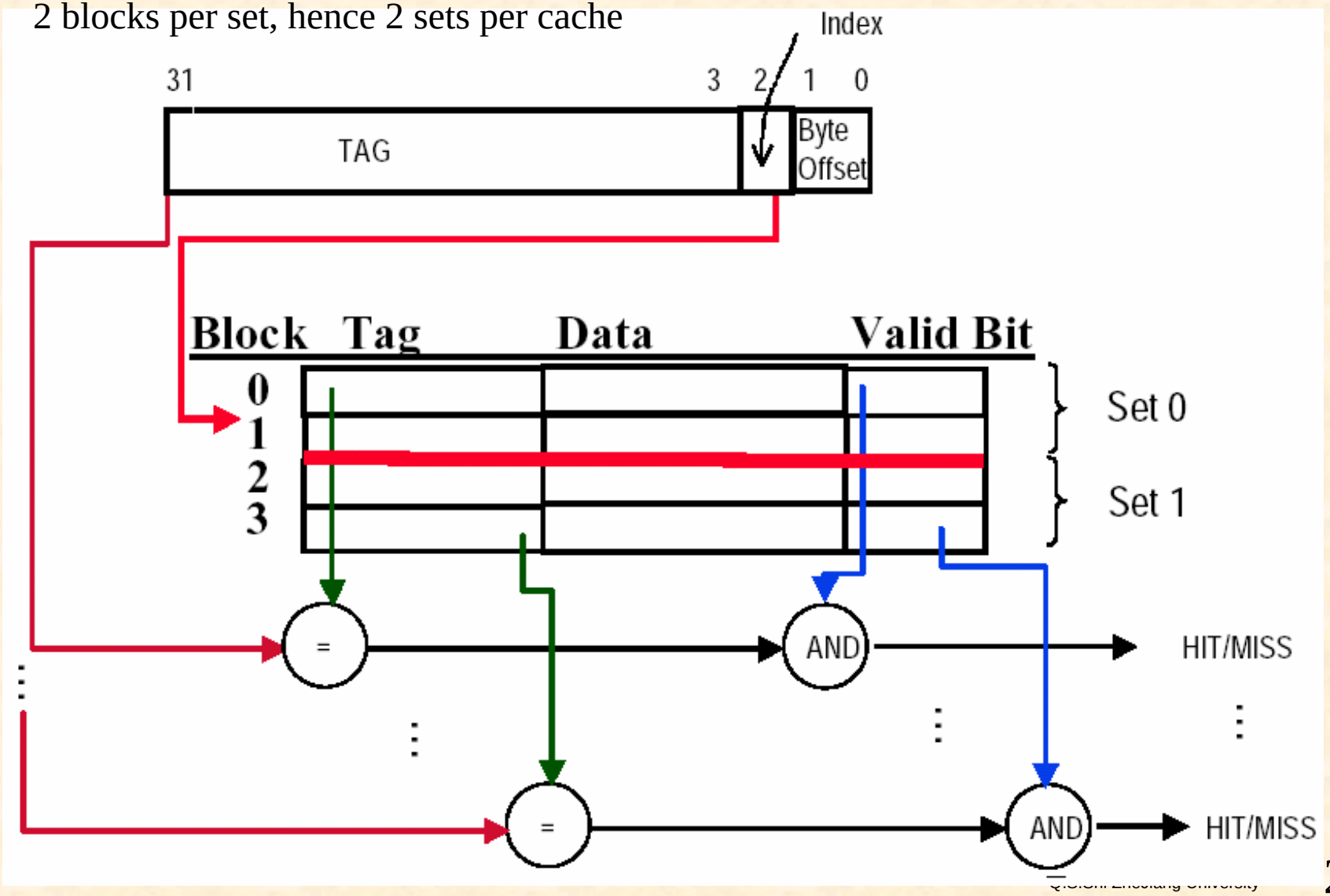
Fully-Associative Cache example (1-word Blocks)

- **Assume cache has 4 blocks**



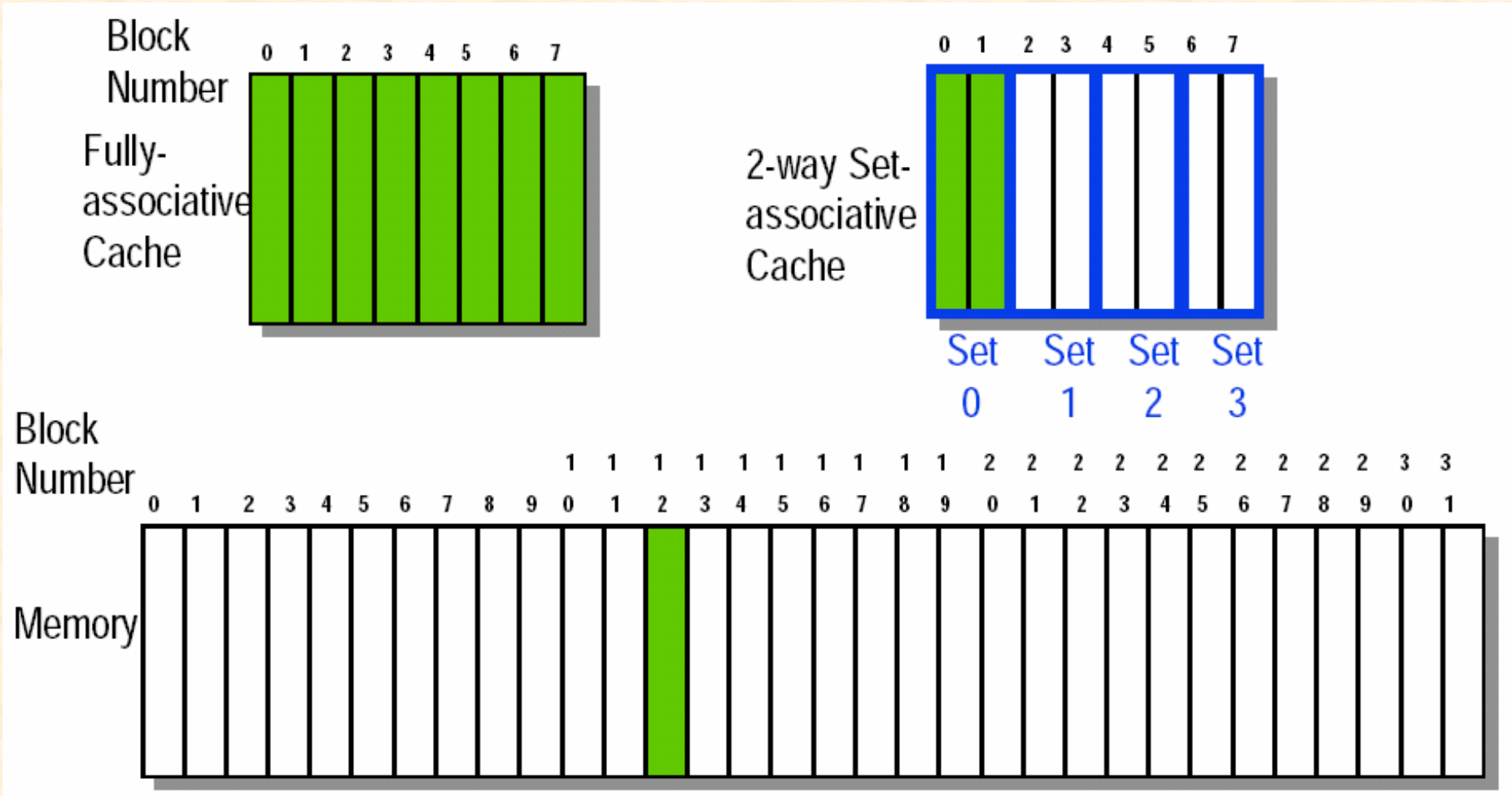
2-Way Set-Associative Cache

- Assume cache has 4 blocks and each block is 1 word
- 2 blocks per set, hence 2 sets per cache



Q3: Block Replacement

- In a direct-mapped cache, there is only one block that can be replaced
- In set-associative and fully-associative caches, there are N blocks (where N is the degree of associativity



Strategy of block Replacement

- Several different replacement policies can be used
  - Random replacement – *randomly pick any block*
    - **Easy to implement in hardware, just requires a random number generator**
    - **Spreads allocation uniformly across cache**
    - **May evict a block that is about to be accessed**
  - Least-recently used (LRU) – *pick the block in the set which was least recently accessed*
    - **Assumed more recently accessed blocks more likely to be referenced again**
    - **This requires extra bits in the cache to keep track of accesses.**
  - First in, first out(FIFO) –*Choose a block from the set which was first came into the cache*

Q4: Write Strategy

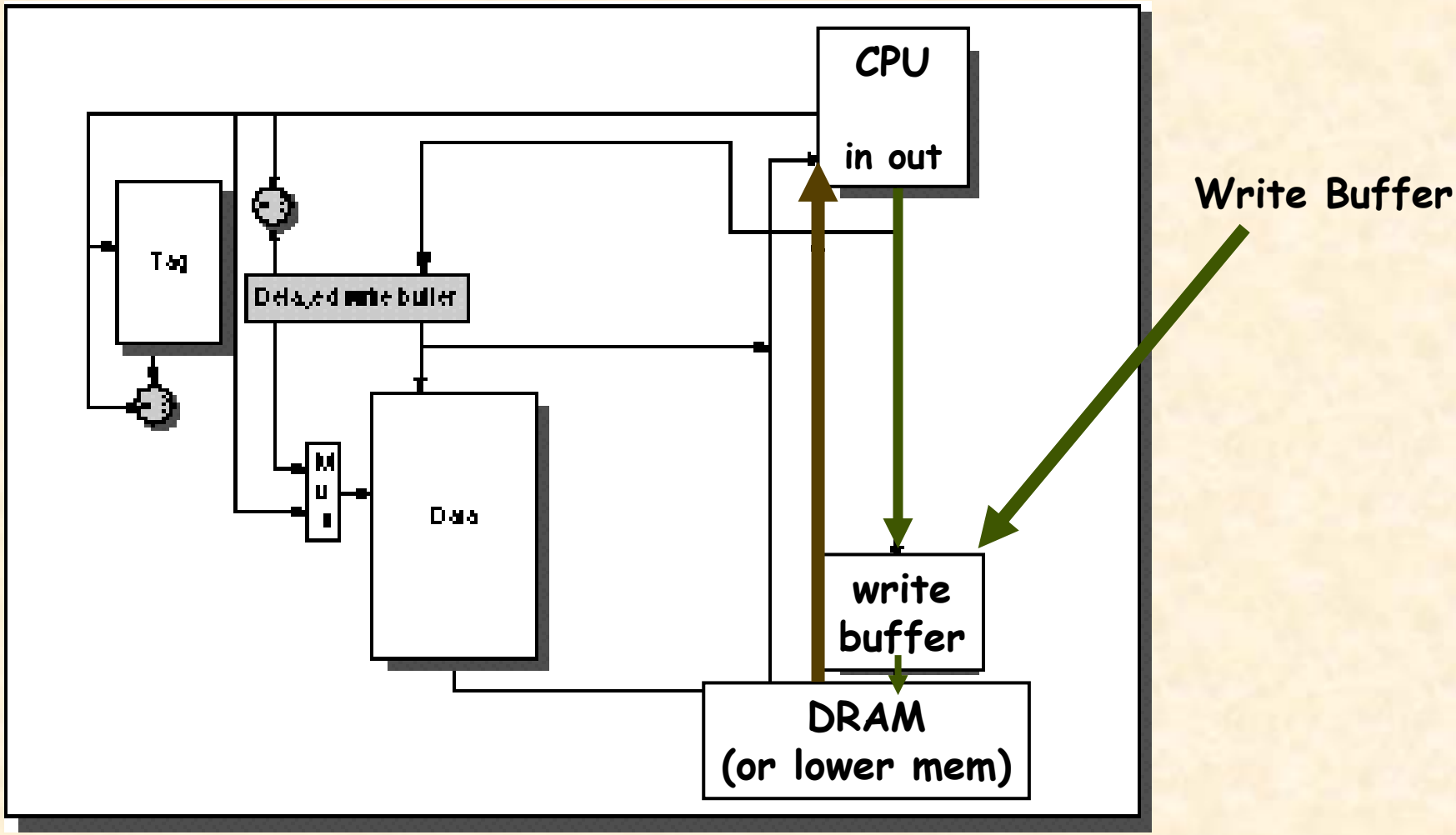
- When data is written into the cache (on a store), is the data also written to main memory?
  - If the data is written to memory, the cache is called a *write-through cache*
    - Can always discard cached data - most up-to-date data is in memory
    - Cache control bit: only a *valid* bit
    - memory (or other processors) always have latest data
  - If the data is NOT written to memory, the cache is called a *write-back cache*
    - Can't just discard cached data - may have to write it back to memory
    - Cache control bits: both *valid* and *dirty* bits
    - much lower bandwidth, since data often overwritten multiple times
- Write-through adv: Read misses don't result in writes, memory hierarchy is consistent and it is simple to implement.
- Write back adv: Writes occur at speed of cache and main memory bandwidth is smaller when multiple writes occur to the same block.

Write stall

- **Write stall ---When the CPU must wait for writes to complete during write through**
- **Write buffers**
  - A small cache that can hold a few values waiting to go to main memory.
  - *To avoid stalling on writes, many CPUs use a write buffer.*

  - This buffer helps when writes are clustered.
  - It does not entirely eliminate stalls since it is possible for the buffer to fill if the burst is larger than the buffer.

Write buffers

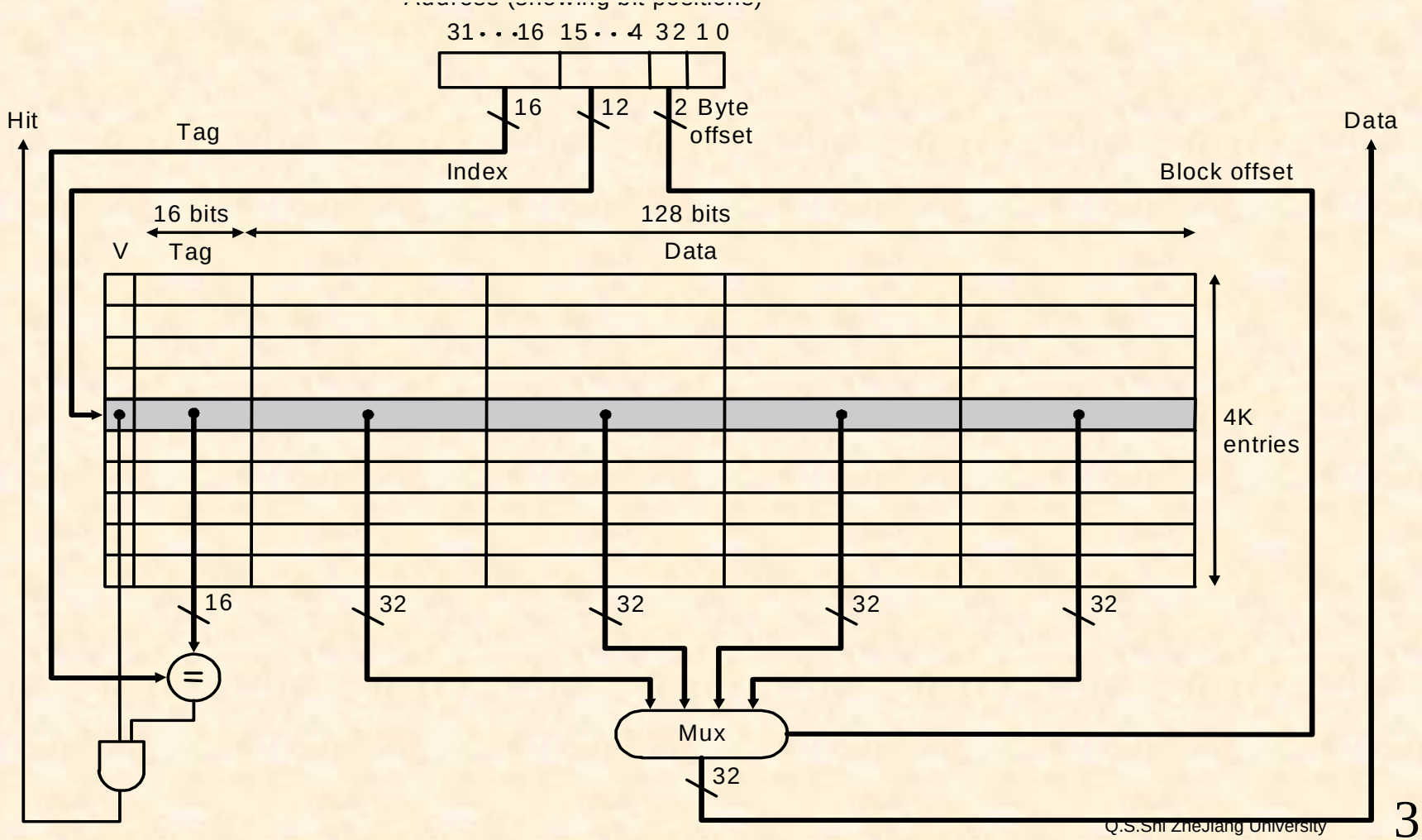


Write misses

- **Write misses**
  - *If a miss occurs on a write (the block is not present), there are two options.*
  - *Write allocate*
    - ***The block is loaded into the cache on a miss before anything else occurs.***
  - *Write around (no write allocate)*
    - ***The block is only written to main memory***
    - ***It is not stored in the cache.***
  - *In general, write-back caches use write-allocate , and write-through caches use write-around .*

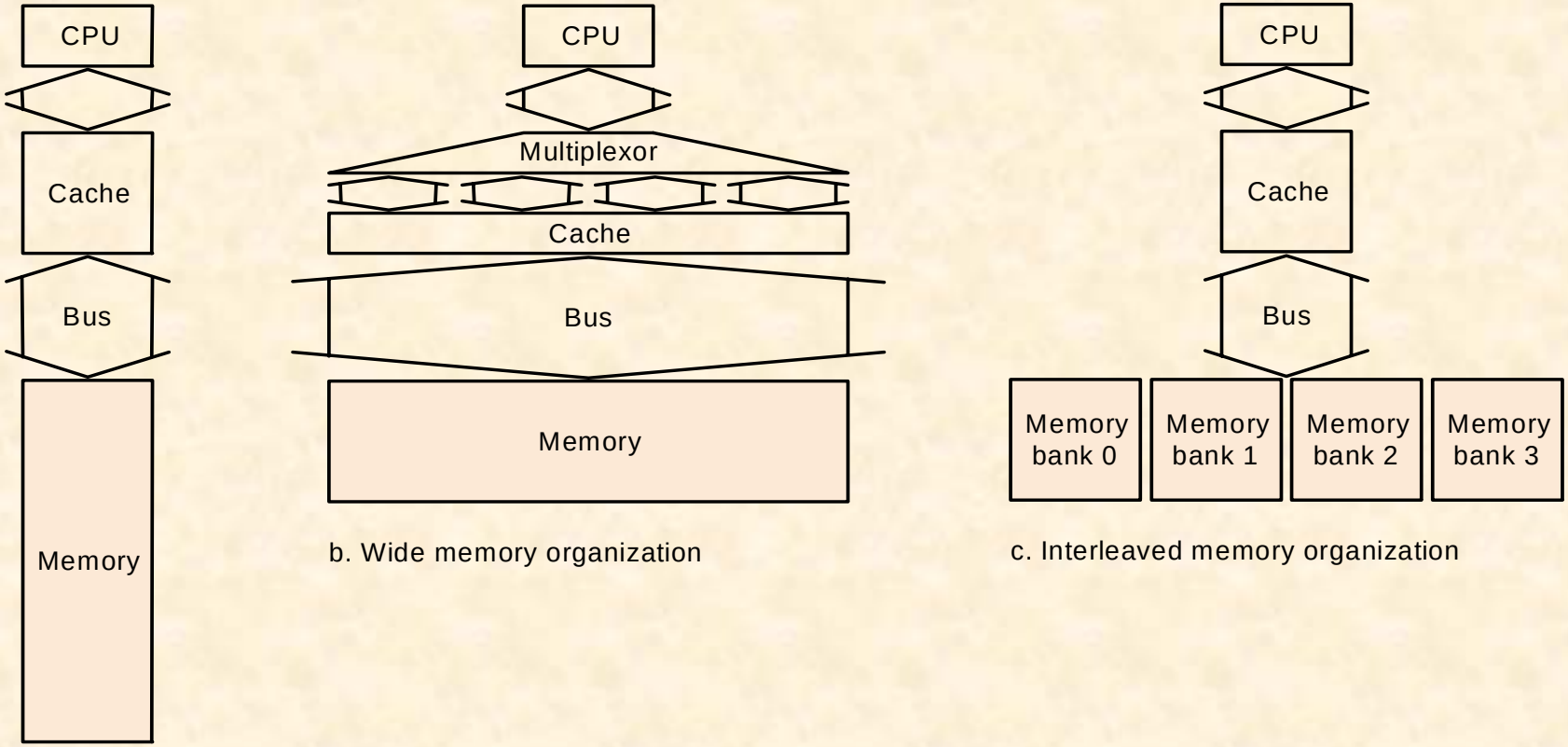
Larger blocks exploit spatial locality

- **Taking advantage of spatial locality to lower miss rates with many word in the block:**



Designing the Memory system to Support Cache

- **Make reading multiple words easier by using banks of memory**



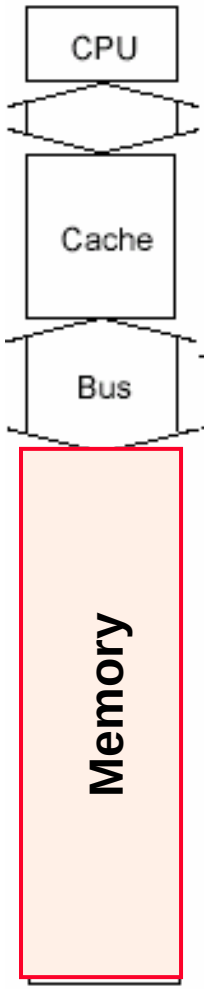
a. One-word-wide memory organization

b. Wide memory organization

c. Interleaved memory organization

- **It can get a lot more complicated...**

Performance basic memory organization



Assume

1 clock cycles to send the address

15 memory bus clock cycles for each DRAM access initiated

1 bus clock cycles to send a word of data

Block size is 4 words

Every word is 4 bytes

The time to transfer one word is 1+15+1=17

The miss penalty (The time to transfer one block is):

$$1+4\times(1+15)=65 \text{ CLKs}$$

Bandwidth : $\frac{4\times 4}{65} \approx \frac{1}{4}$

Only one word is useful,and three other words may be useless. So, for caches using four-word blocks, this memory system is not viable.

Performance in Wider Main Memory

- With a main memory width of 2 words(64bits)

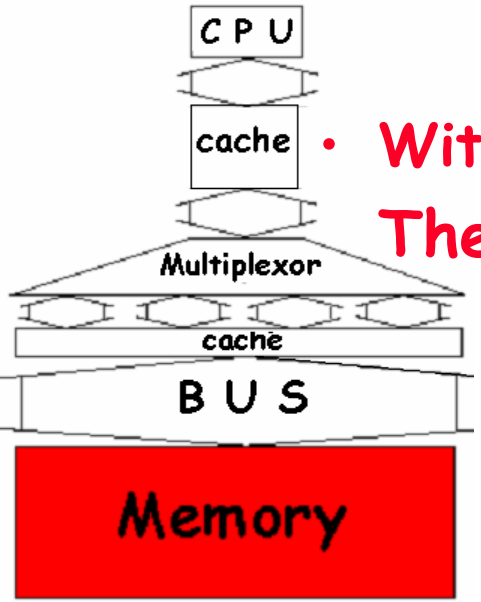
The miss penalty: 4words/Block

$$1+2\times(15+1)=33 \text{ CLKs}$$

Bandwidth :

$$\frac{4\times4}{33}=\frac{16}{33}\approx0.48$$

only two times that needed to transfer one word.



- With a main memory width of 4 words(128bits)

The miss penalty: 4words/Block

$$1+1\times(15+1)=17 \text{ CLKs}$$

Bandwidth :

$$\frac{4\times4}{17}=\frac{16}{17}\approx0.98$$

Equal to time to transfer one word.

Performance in Four-way interleaved memory

- With 4 banks Interleaved Memory

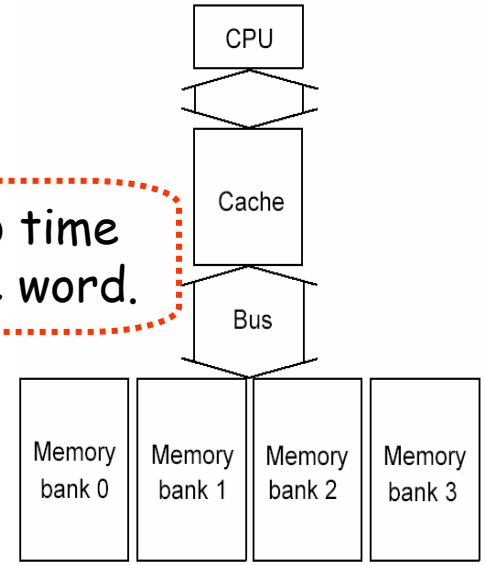
  The miss penalty: 4words/Block

$$1+15+(4 \times 1)=20$$

Almost equal to time to transfer one word.

Bandwidth :

$$\frac{4 \times 4}{20} = 0.8$$



Four-way interleaved memory

Parallel access

Word address	Bank 0	Word address	Bank 1	Word address	Bank 2	Word address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

Optimizes sequential address access patterns

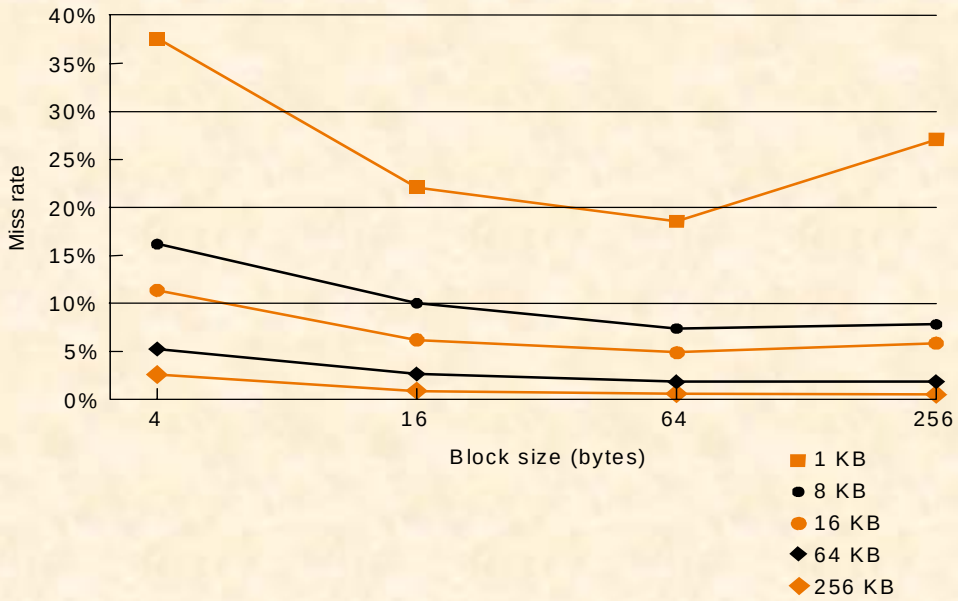
DRAM developed

Year introduced	Chip size	$ per MB	Total access time to a new row/column	Columm access time to existing row
1980	64Kbit	$1500	250ns	150ns
1983	128Kbit	$500	185ns	100ns
1985	1Mbit	$200	135ns	40ns
1989	4Mbit	$50	110ns	40ns
1992	16Mbit	$15	90ns	30ns
1996	64Mbit	$10	60ns	12ns
1998	128Mbit	$4	60ns	10ns
2000	256Mbit	$1	55ns	7ns
2002	512Mbit	$0.25	50ns	5ns
2004	1024Mbit	$0.10	45ns	3ns

DRAM size increased by multiples of four approximately once every three year until 1996,and thereafter doubling approximately every two years.

Performance in different block size

- **Increasing the block size tends to decrease miss rate:**



- **Use split caches because there is more spatial locality in code:**

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

7.3 Measuring and improving cache performance

- **In this section, we will discuss two questions:**
  1. How to measure cache performance?
  2. How to improve performance?

- **The main contents are the following:**
  1. Measuring cache performance
  2. Reducing cache misses by more flexible placement of blocks
  3. Reducing the miss penalty using multilevel caches

Average Memory Assess time = hit time + miss time

= **hit rate** × **Cache time + miss rate** ×**memory time**

= 99% × 5 + (1-99%) × 45 =5.5ns

Measuring cache performance

- We use CPU time to measure cache performance.

CPU time=

$$CPU_{time} = I \times CPI \times Clock\ \ cycle\ time$$

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

Memory-stall clock cycles = # of instructions ´ miss ratio ´ miss penalty

= Read-stall cycles + Write-stall cycles

For Read-stall

$$\text{Read-stall cycles} = \frac{\text{Read}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

- For a write-through plus write buffer scheme:

$$\text{Write-stall cycles} = \left(\frac{\text{Read}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty} \right)$$

+ Write buffer stalls

- If the write buffer stalls are small, we can safely ignore them .
- If the cache block size is one word, the write miss penalty is 0.

Combine the reads and writes

- **In most write-through cache organizations, the read and write miss penalties are the same**
  - the time to fetch the block from memory.
- **If we neglect the write buffer stalls, we get the following equation:**

  Memory-stall clock cycles =

$$\frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

We can also write this as:

$$\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instructions}} \times \text{Miss penalty}$$

Calculating cache performance

- **Assume:**

  instruction cache miss rate 2%

  data cache miss rate 4%

  CPI without any memory stalls 2

  miss penalty 100 cycles

  The frequency of all loads and stores in gcc is 36%,as we see in Figure 3.26, on page 288.

- **Question: How faster a processor would run with a perfect cache?**
- **Answer:**

  Instruction miss cycles = I×2%×100 =2.00I

  Data miss cycles = I×36%×4%×100 =1.44I

  Total memory-stall cycles= 2.00I+ 1.44I =3.44 I

  CPI with stall = CPI with perfect cache + total memory-stalls

  = (2 + 3.44)I = 5.44I

How faster a processor for ideal

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times CPI_{stall} \times \text{Clock cycle}}{I \times CPI_{perfect} \times \text{Clock cycle}}$$

$$= \frac{CPI_{stal}}{CPI^{I}_{perfect}} = \frac{5.44}{2} = 2.72$$

- **What happens if the processor is made faster?**

Assume CPI reduces from 2 to 1

CPI with stall = CPI with perfect cache + total memory-stalls

=(1+3.44)I = 4.44I

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{CPI_{stal}}{CPI^{I}_{perfect}} = \frac{4.44}{1} = 4.44$$

Ratio time for Memory stalls

$$\text{from } \frac{3.44}{5.44} = 63\% \quad \text{to } \frac{3.44}{4.44} = 77\%$$

Calculating cache performance with Increased Clock Rate

- ***Suppose we increase the performance of the computer in the previous example by doubling its clock rate for same memory system.***

- ***Question : How much faster will the computer be with the faster clock to slow clock?***

- ***Answer***

Total miss cycles per instruction = (2% × 200) + 36% × (4% × 200)=6.88

CPI with cache misses = 2 + 6.88 =8.88

$$\frac{\text{Performance with fast clock}}{\text{Performance with slow clock}} = \frac{\text{Execution time with slow clock}}{\text{Execution time with fast clock}}$$

$$= \frac{IC \times CPI_{slow\ clock} \times \text{Clock cycle}}{IC \times CPI_{fast\ clock} \times \text{Clock cycle/2}} = \frac{5.44}{8.88 \times 1/2} = 1.23$$

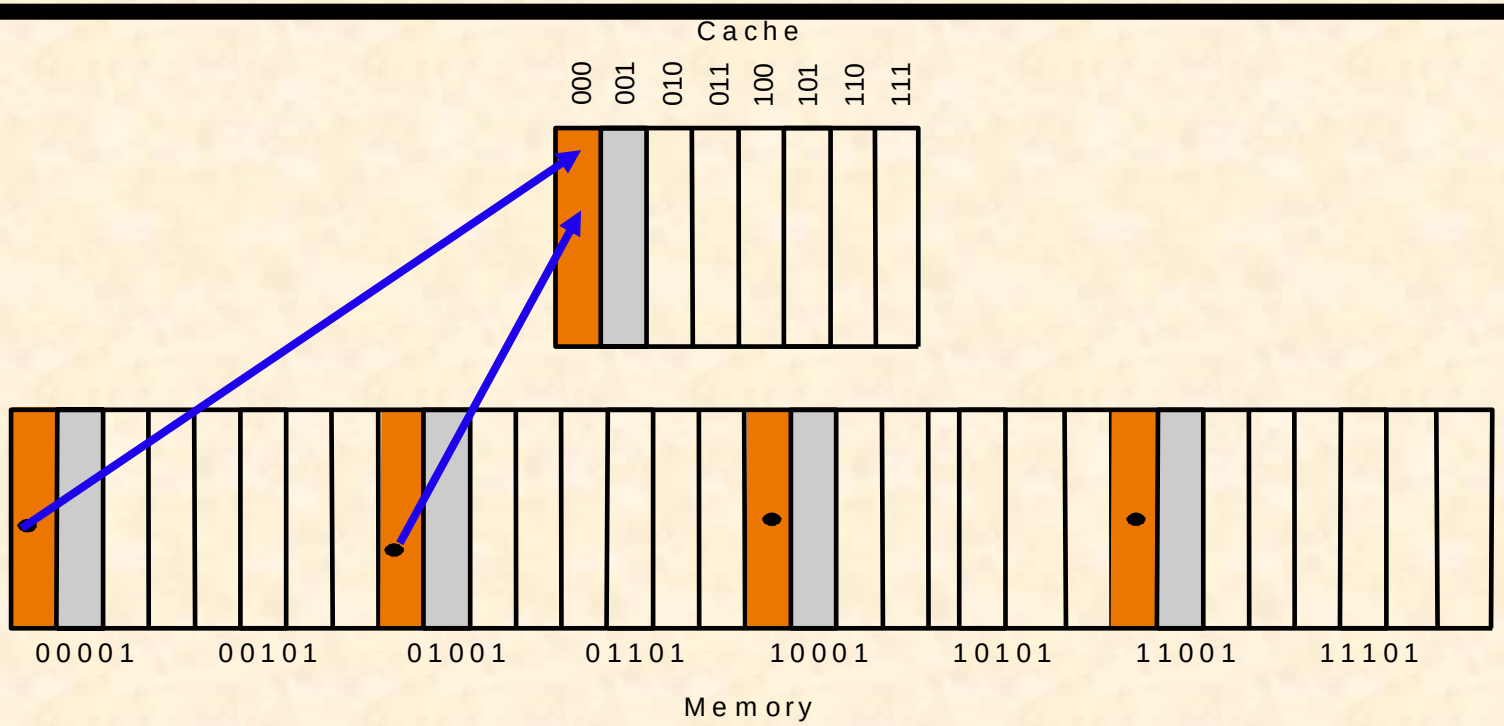
This, the computer with the faster clock is about 1.2 times faster rather than 2 time faster.

Solution 1

Reducing cache misses by more flexible placement of blocks

(1) The disadvantage of a direct-mapped cache
(2) The basics of a set-associative cache
(3) Miss rate versus set-associative
(4) Locating a block in the set-associative cache
(5) Size of tags versus set associative
(6) Choosing which block to replace

The disadvantage of a direct-mapped cache



- **If the CPU requires the following memory units sequentially: word 0,word 8 and word 0. Word 0 and word 8 both are mapped to cache block 0, so the third access will be a miss.**
- **But obviously, if one memory block can be placed in any cache block , the miss can be avoided. So, there is possibility that the miss rate can be improved.**

The basics of a set-associative cache

Decreasing miss ratio with associativity

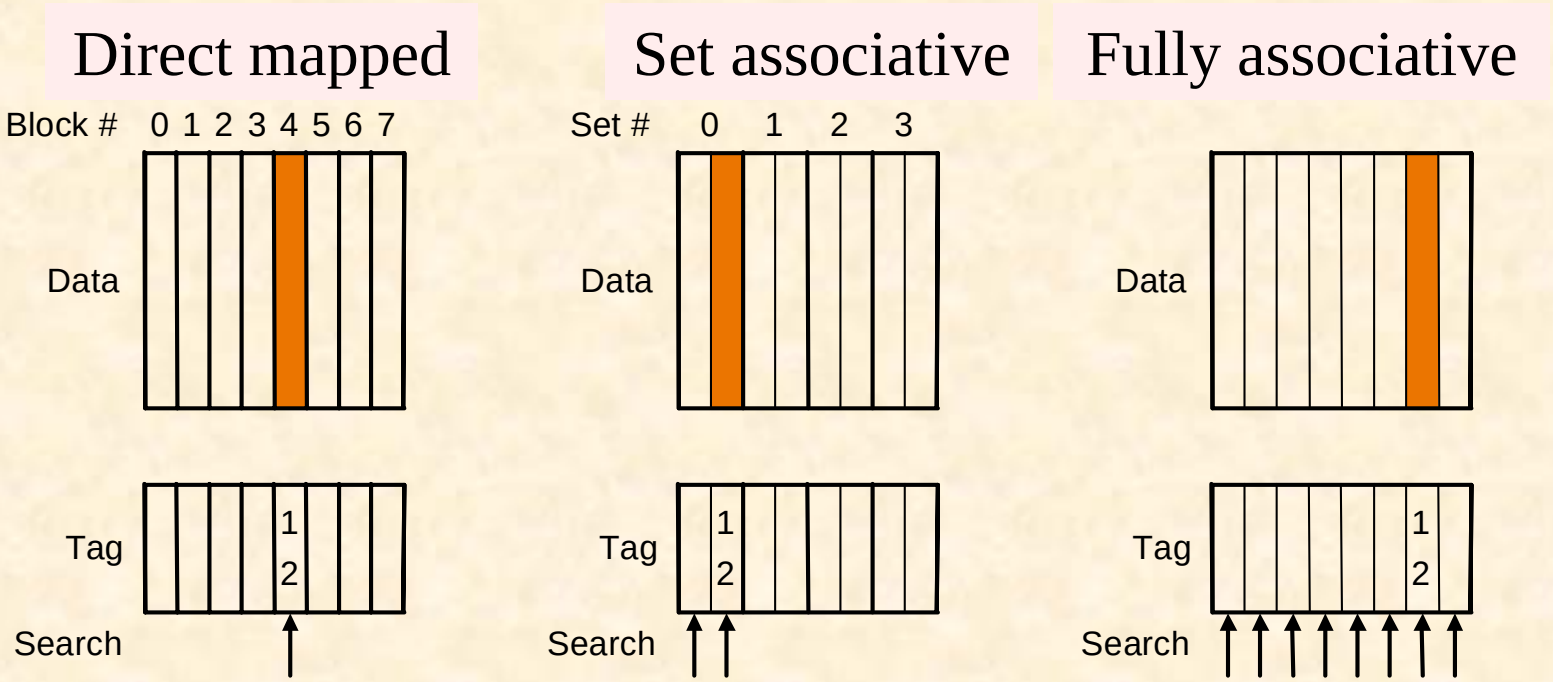
- **A set-associative cache is divided into some sets. A set contains several blocks.**
- **A memory block is mapped to a set in the cache through a mapping algorithm.**
  - The memory block can be placed in any block in the corresponding set.
- **The mapping algorithm is: (set with direct-mapped)**

Set number (Index) =

(Memory block number) modulo (Number of sets in the cache)

- If a set has only one block, this set-associative cache is actually a direct-mapped cache.
- If a set-associative cache has only one set, this set-associative cache is called a fully-associative cache.

Memory block whose address is 12 in a cache with 8 blocks for different mapped



An eight-block cache configured as variety-way

(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

Miss rate versus set-associativity

Assume: there are three small caches, each consisting of four one-word blocks.

One cache is direct-mapped,

the second is two-way set associative

and the third is fully associative.

Question: Given the following sequence of block addresses: 0,8,0,6,8, find the number of misses for each cache organization.

Answer: for direct-mapped 6 misses

Memory block	Hit or miss	Contents after each reference			
		Set 0	Set 1	Set 2	Set 3
		Block 0	Block 1	Block 2	Block 3
0	Miss	M[0]			
8	Miss	M[8]			
0	Miss	M[0]			
6	Miss	M[0]		M[6]	
8	Miss	M[8]		M[6]	

Second, for the two-way set associative cache. **5 misses**

Memory block	Hit or miss	Contents after each reference			
		Set 0		Set 1	
		Block 0	Block 1	Block 2	Block 3
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[6]		
8	Miss	M[8]	M[6]		

Finally, for the fully associative cache. **3 misses**

Memory block	Hit or miss	Contents after each reference			
		Only one set			
		Block 0	Block 1	Block 2	Block 3
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[8]	M[6]	
8	Hit	M[0]	M[8]	M[6]	

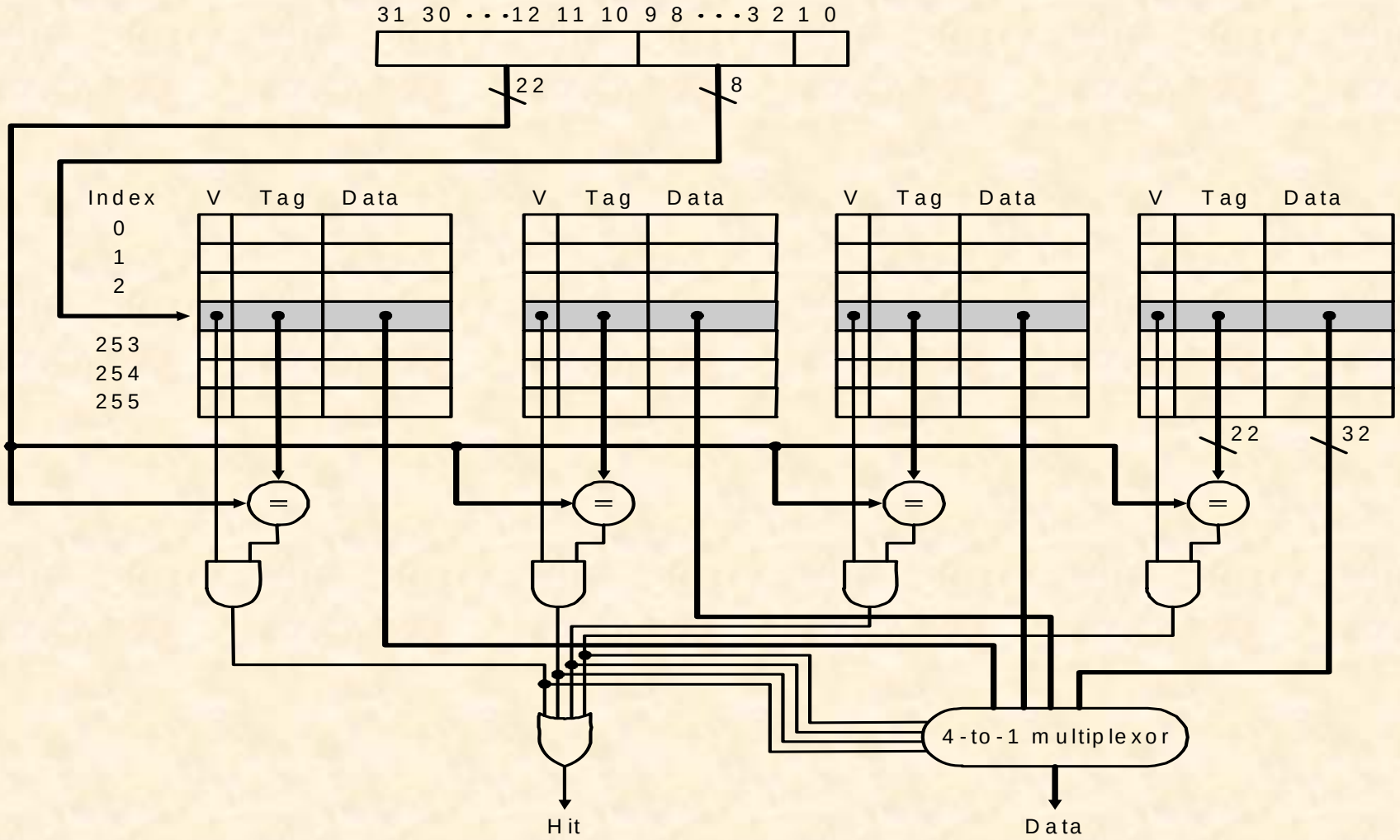
How much of a reduction in the miss rate is achieved by associativity?

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

The data cache miss rates for organization like the Intrinsuty FastMATH processor for SPEC2000 benchmarks with associativity varying form one-way to eight-way .

- Data cache organization is 64KB data cache and 16-word block

Locating a block in the set-associative cache



- The implementation of a four-way set-associative cache requires four comparators and a 4-to-1 multiplexor.

Size of tags versus set associativity

Assume

Cache size is 4K Block

Block size is 4 words

Physical address is 32bits

Question

Find the total number of set and total number of tag bits for variety associativity

Answer

Offset size (Byte) = 16= 2^4 — 4 bits for address

Number of memory block = $2^{32} \div 2^4 = 2^{28}$ — 28 bits for Block address

Number of cache block = 2^{12} — 12 bits for Block address

For direct-mapped

Bits of index = 12 bits

bits of Tag = (28-12) $\times$4K=16$\times$4K=64 Kbits

For two-way associative

Number of cache set = $2^{12} \div 2 = 2^{11}$

Bits of index = 12-1=11 bits

Bits of Tag = (28-11) ×2×2K=17×2×2K=68 Kbits

For four-way associative

Number of cache set = $2^{12} \div 4 = 2^{10}$

Bits of index = 12-1=10 bits

Bits of Tag = (28-10) ×4×1K=18×4×1K=72 Kbits

For full associative

Number of cache set = $2^{12} \div 2^{12} = 2^{0}$

Bits of index = 12-12=0 bits

Bits of Tag = (28-0) ×4K×1=128 Kbits

	Direct	2-way	4-way	Fully
Index(bit)	12	11	10	0
Tag(bit)	16	17	18	28

Choosing which block to replace

- In an associative cache, we must decide which block to replace when a miss happens and the corresponding set is full.
- The most commonly used scheme is least recently used (LRU), which we used in the previous example. In an LRU scheme, the block replaced is the one that has been unused for the longest time.
- For a two-way set associative cache, the LRU can be implemented easily. We could keep a single bit in each set. We set the bit whenever a specific block in the set is referenced, and reset the bit whenever another block is referenced.
- As associativity increases, implementing LRU gets harder.

Decreasing miss penalty with multilevel caches

- **Add a second level 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:**
  - CPI of 1.0 on a 5GHz machine with a 2% miss rate, 100ns DRAM access
  - Adding 2nd level cache with 20ns access time decreases miss rate to 2%
- **Miss penalty to main memory is**

$$\frac{\textbf{100ns}}{\textbf{0.2}} = 500 \text{ clock cycles}$$

- **The CPI with one level of caching**

Total CPI = 1.0 + Memory-stall cycles per instruction

$= 1.0 + 2\% \times 500 = 11.0$

Miss penalty with levels of cache without access main memory

$$\frac{\textbf{5ns}}{\textbf{0.2}} = 25 \text{ clock cycles}$$

- **The CPI with Two level of cache with 0.5% miss rate for main memory**

Total CPI = 1.0 + Primary stalls per instruction + Secondary stalls per instruction

= 1 + 2% ×25 + 0.5% × 500

= 1.0 + 0. 5 +2.5 = 4.0

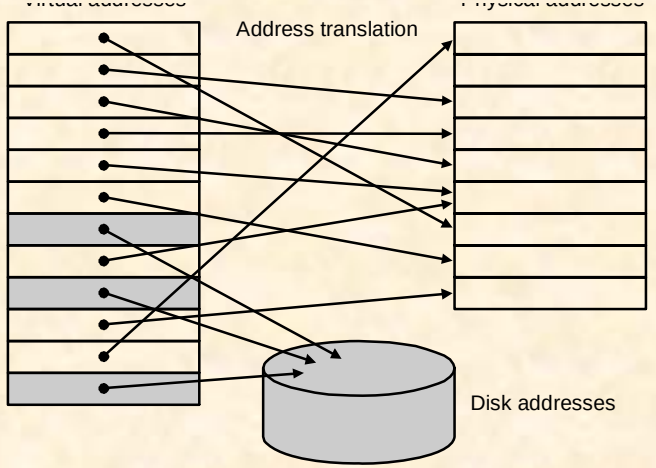
- **The processor with secondary cache is faster by**

$$\frac{\textbf{11.0}}{\textbf{4.0}} = 2.8$$

- **Using multilevel caches:**
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache

7.4 Virtual Memory

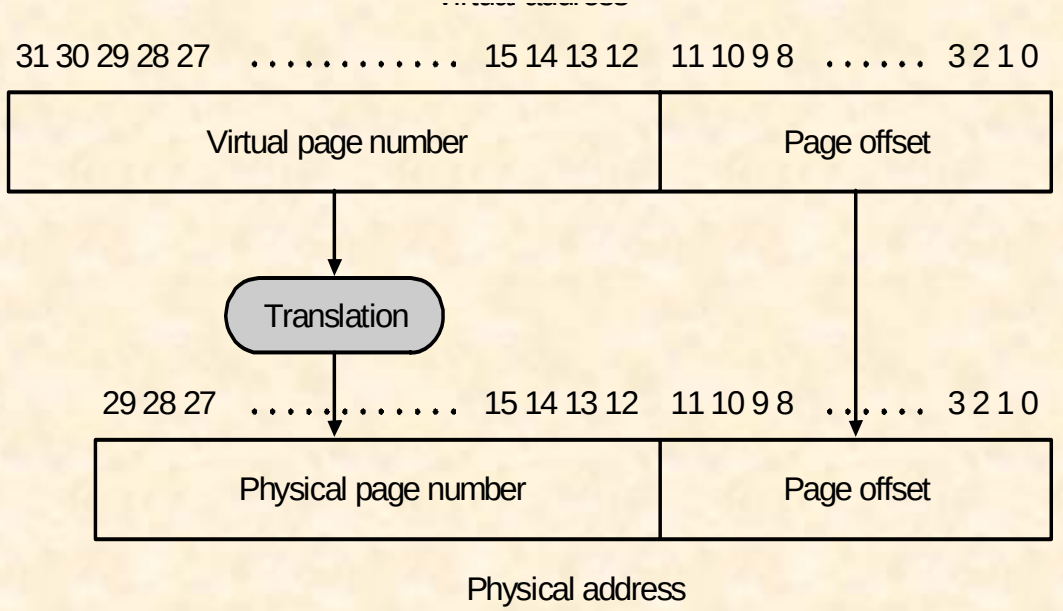
- **Main memory can act as a cache for the secondary storage (disk)**



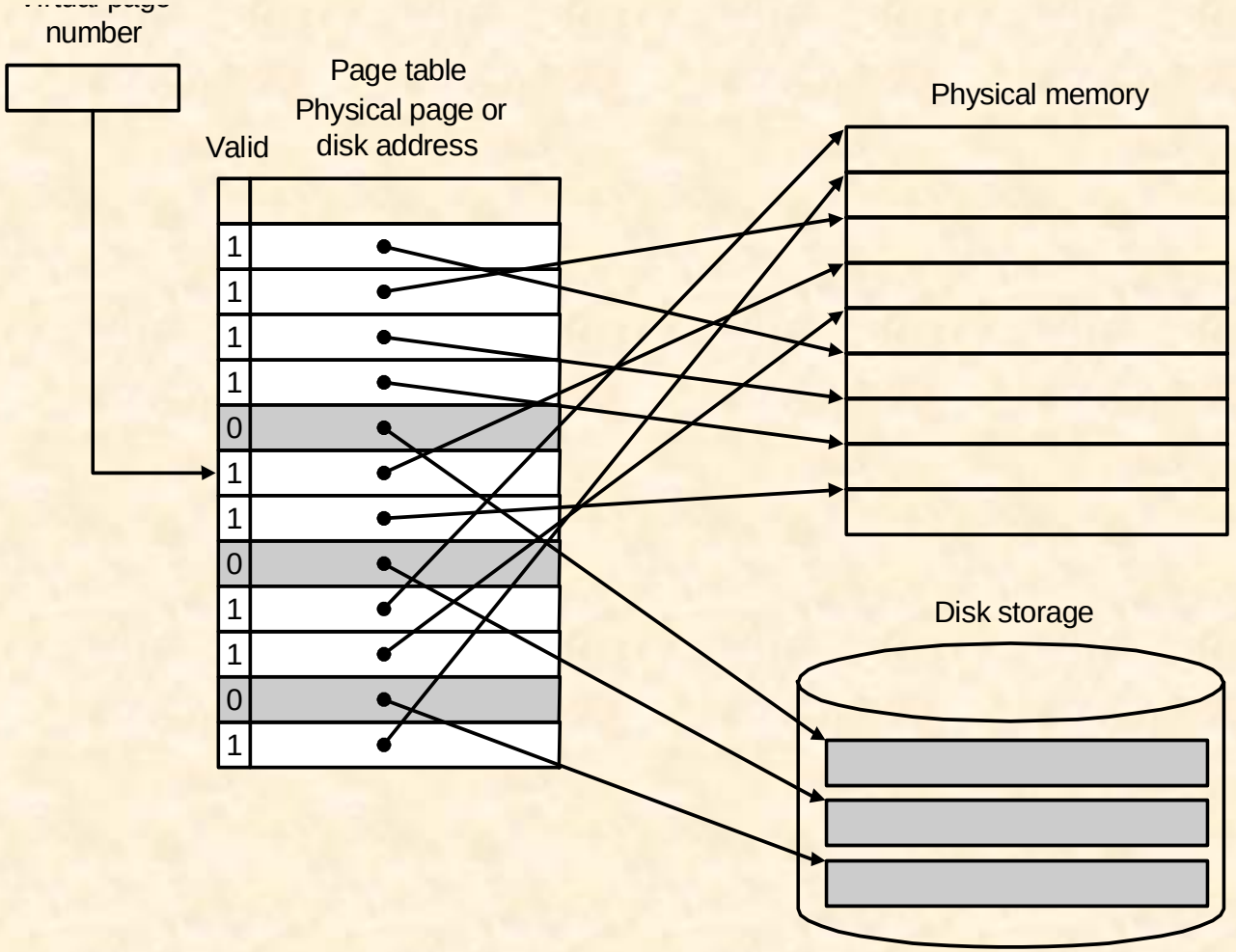
- **Advantages:**
  - illusion of having more physical memory
  - program relocation
  - protection

Pages: virtual memory blocks

- **Page faults: the data is not in memory, retrieve it from disk**
  - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important (LRU is worth the price)
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use write back

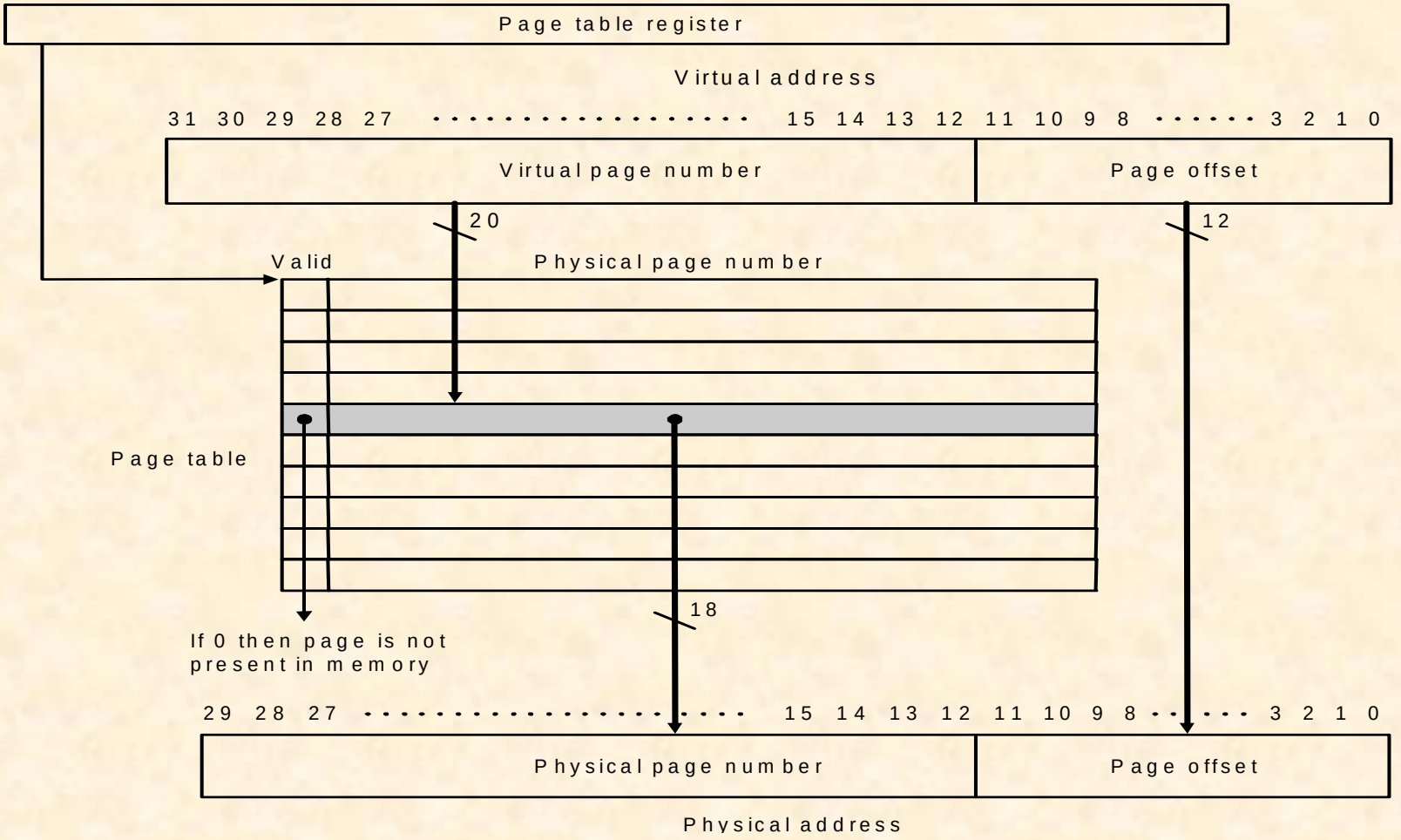


Page Tables


number
Page table
Physical page or
disk address
Valid
1
1
1
1
0
1
1
0
1
1
0
1
Physical memory
Disk storage

Placing a page and finding it again ----Page Tables

Virtual memory systems use fully associative mapping method



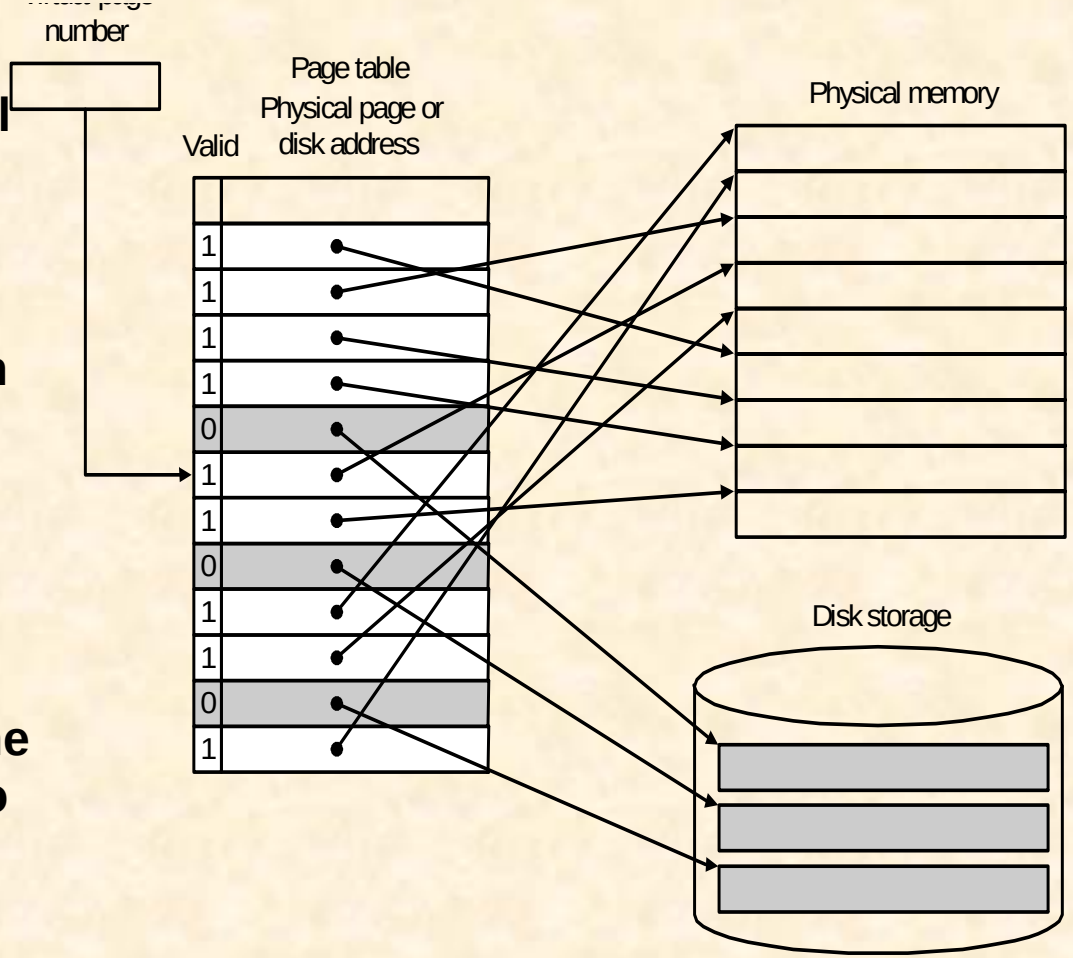
Page faults

- When the OS creates a process, it usually creates the space on disk for all the pages of a process.

- When a gage fault occurs, the OS will be given control through exception mechanism.

- The OS will find the page in the disk by the page table.

- Next, the OS will bring the requested page into main memory. If all the pages in main memory are in use, the OS will use LRU strategy to choose a page to replace

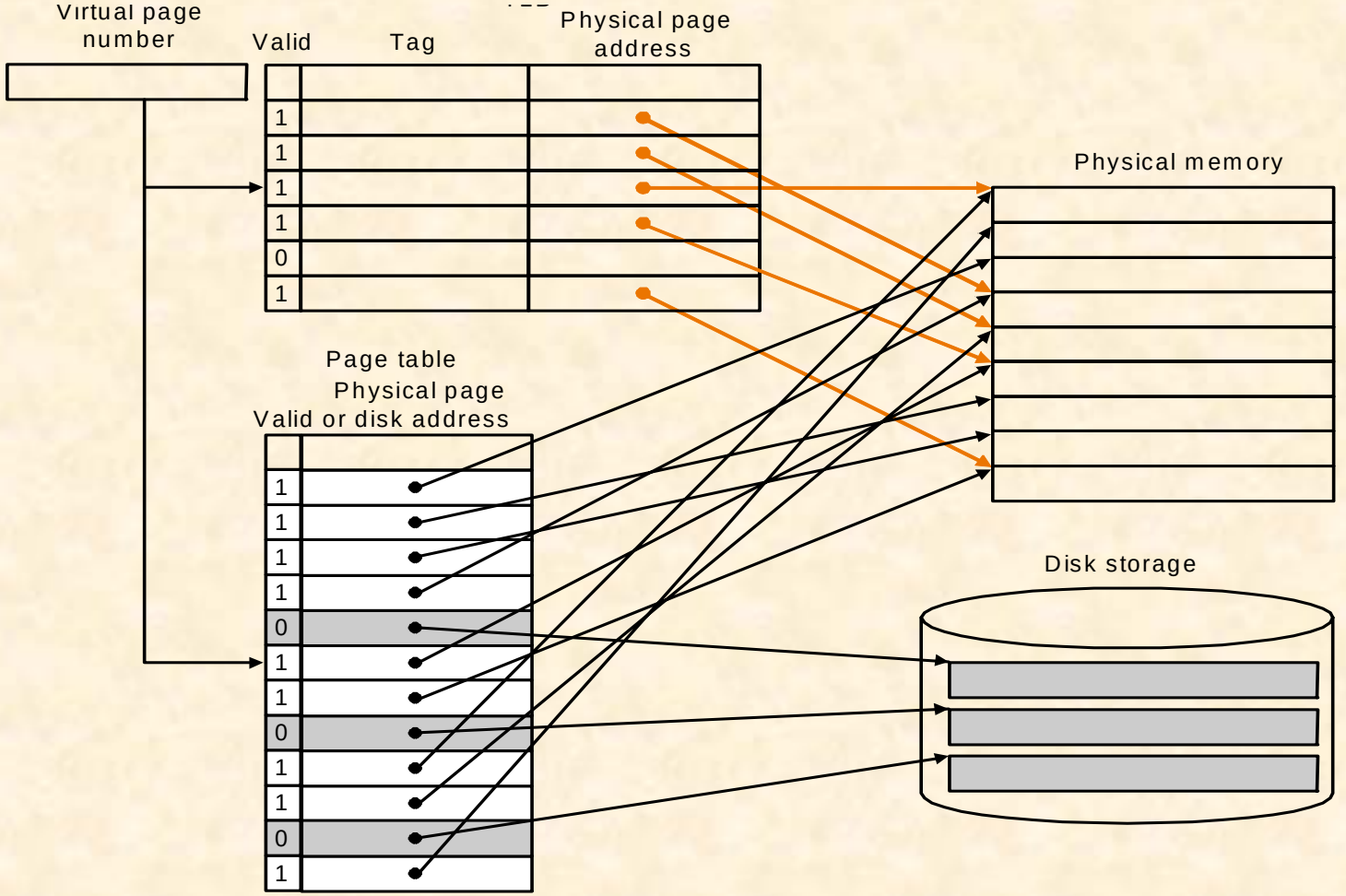


What about writes?

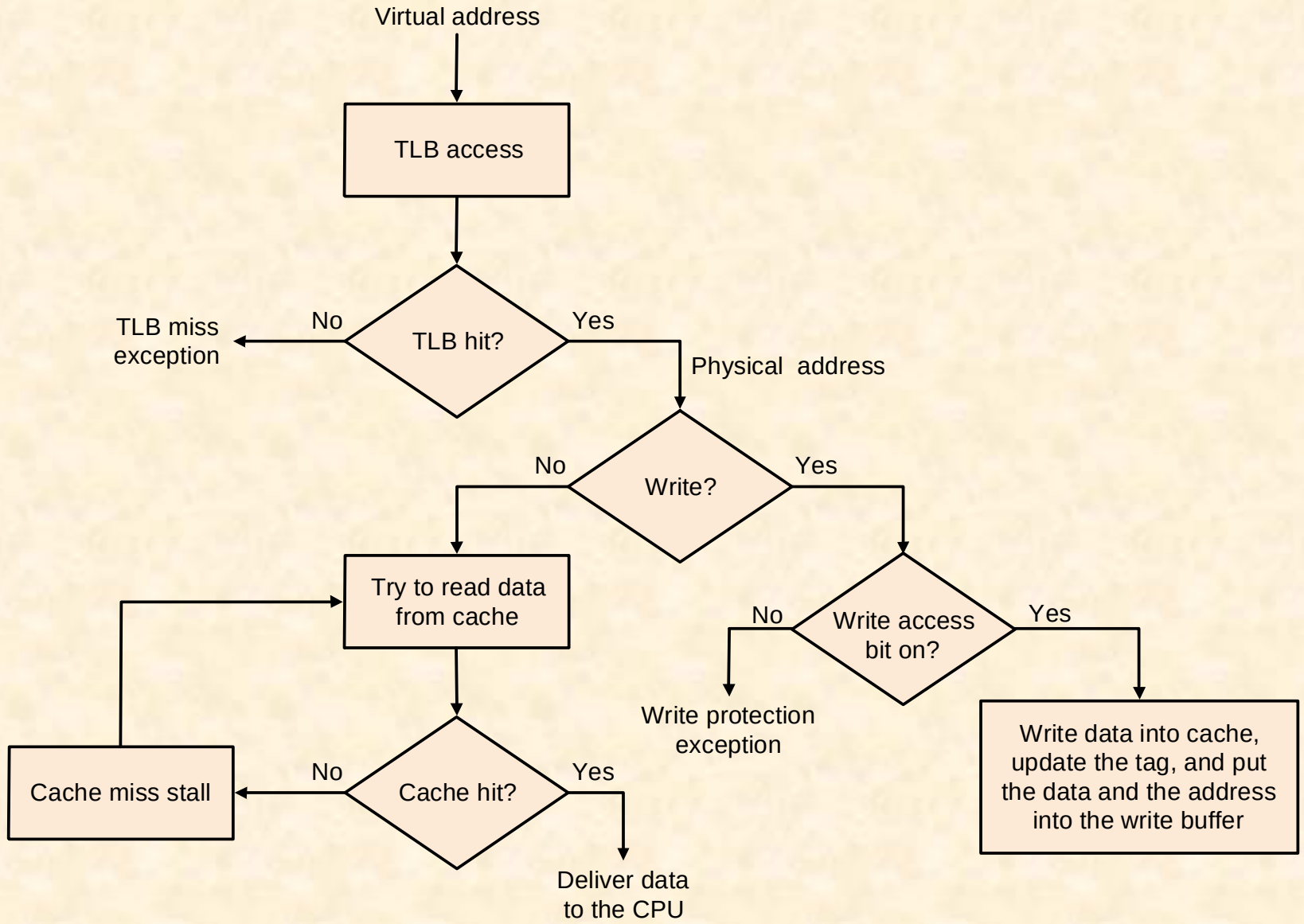
- **Because disk accesses are too slow,virtual memory systems can not use write-through strategy.**
- **Instead, they must use write-back strategy. To do so, the machines need add a dirty bit to the entry of page table.**
- **The dirty bit is set when a page is first written. If the dirty bit of a page is set, the page must be written back to disk before being replaced.**

Making Address Translation Fast----TLB

- **A cache for address translations: translation lookaside buffer**



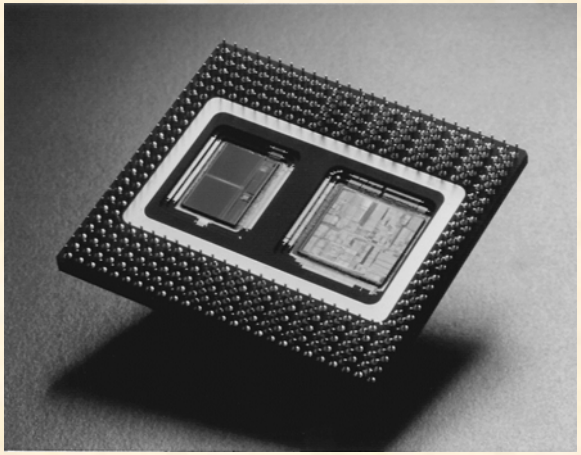
TLBs and caches



Modern Systems

- **Very complicated memory systems:**

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical address	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organization	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware



Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split intruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through

Some Issues

- **Processor speeds continue to increase very fast**
  — much faster than either DRAM or disk access times

- **Design challenge: dealing with this growing disparity**

- **Trends:**
  - synchronous SRAMs (provide a burst of data)
  - redesign DRAM chips to provide higher bandwidth or processing
  - restructure code to increase locality
  - use prefetching (make cache visible to ISA)