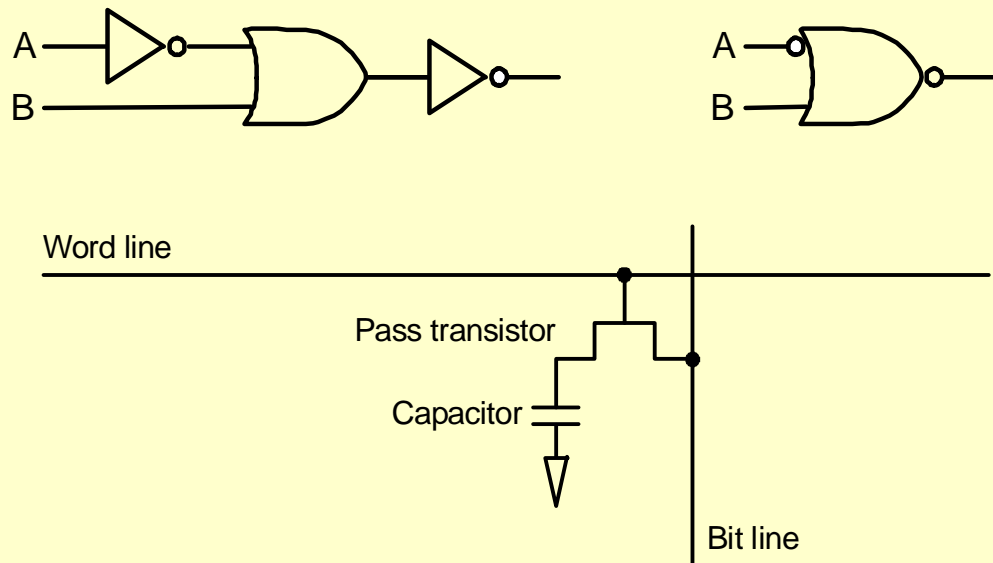


Memories: Review

- **SRAM:**
 - value is stored on a pair of inverting gates
 - very fast but takes up more space than DRAM (4 to 6 transistors)
- **DRAM:**
 - value is stored as a charge on capacitor (must be refreshed)
 - very small but slower than SRAM (factor of 5 to 10)



Exploiting Memory Hierarchy

(See Figure 7.3)

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

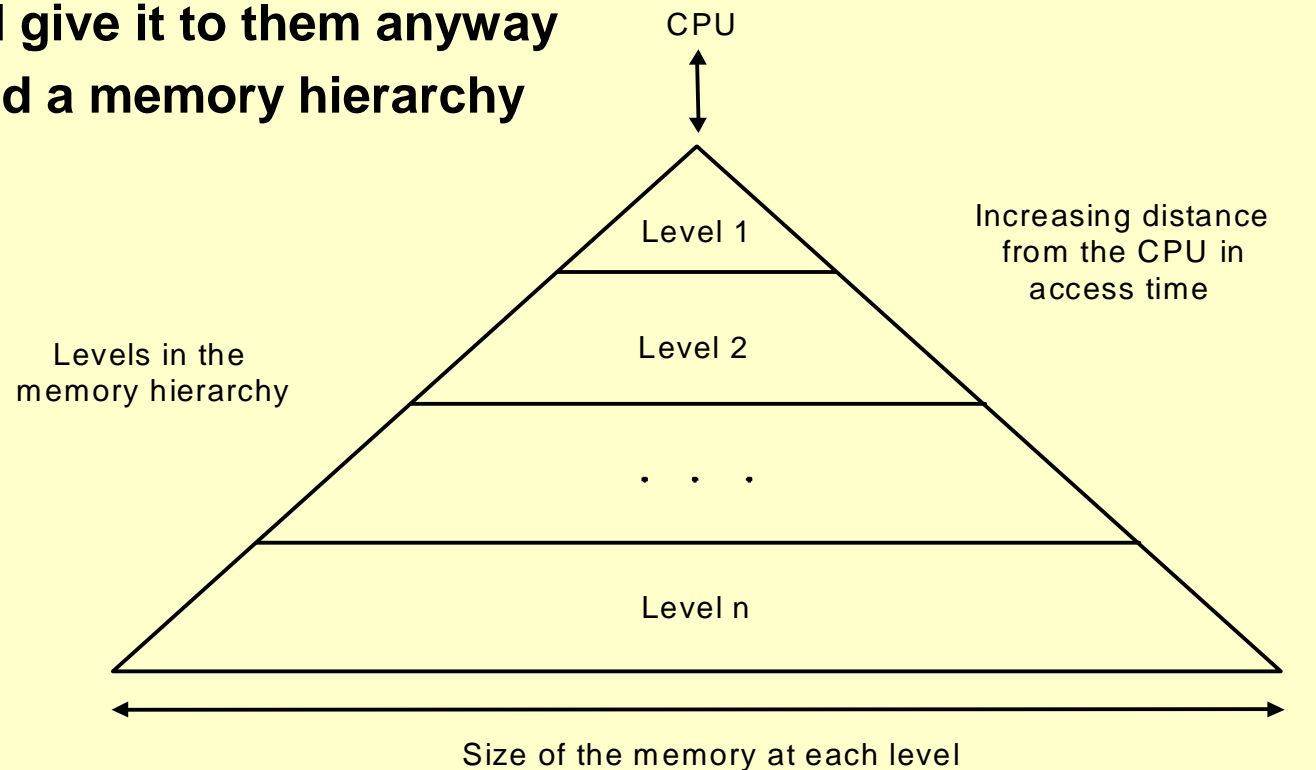
SRAM access times are 2 - 25ns at cost of \$100 to \$250 per Mbyte.

DRAM access times are 60-120ns at cost of \$5 to \$10 per Mbyte.

Disk access times are 10 to 20 million ns at cost of \$.10 to \$.20 per Mbyte.

1997

- **Try and give it to them anyway**
 - **build a memory hierarchy**



Locality

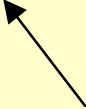
- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,
 - temporal locality*: it will tend to be referenced again soon
 - spatial locality*: nearby items will tend to be referenced soon.

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

Cache

- **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:**
 - **block size is one word of data**
 - **"direct mapped"**

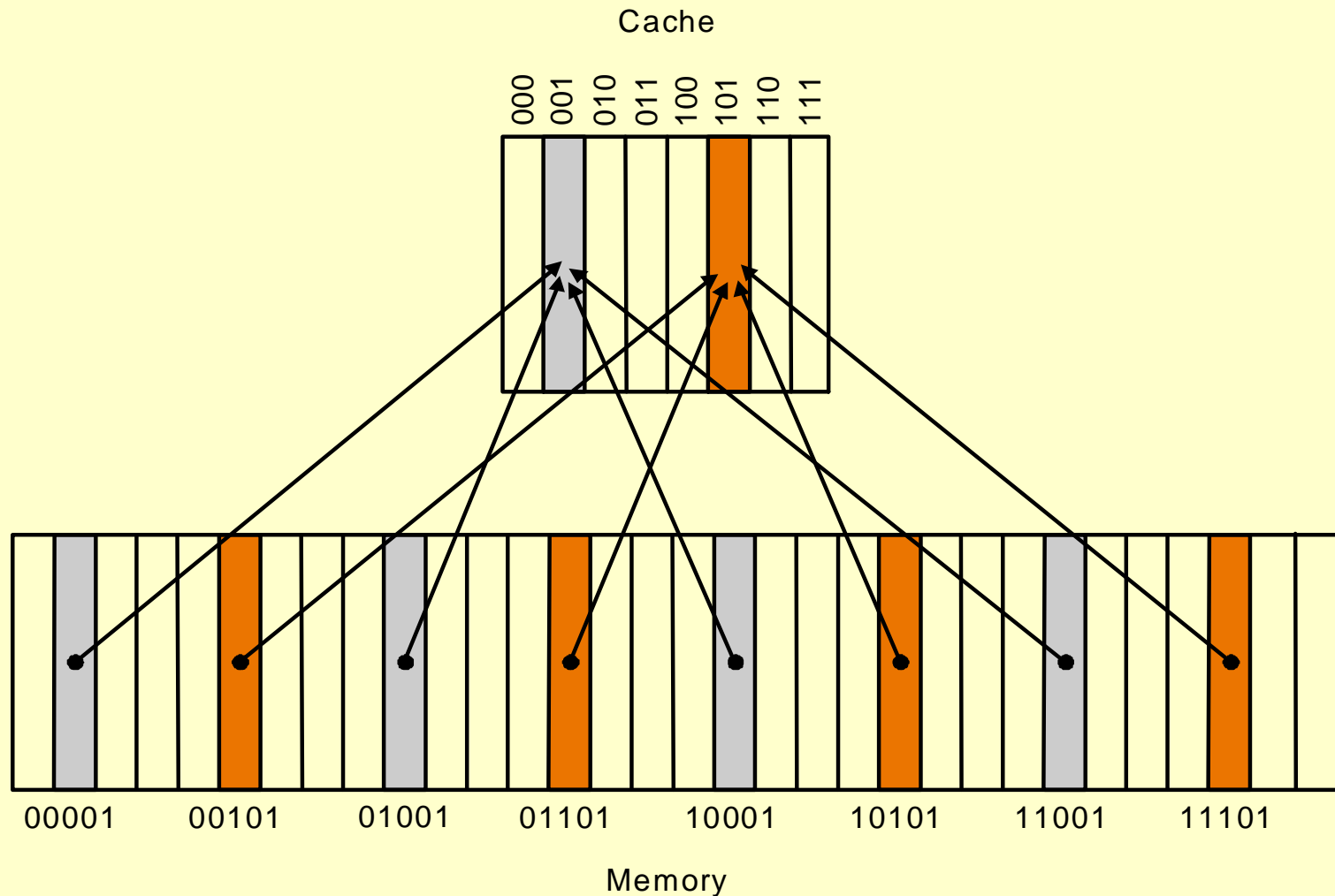
**For each item of data at the lower level,
there is exactly one location in the cache where it might be.**

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

Direct Mapped Cache

(See Figure 7.5)

- Mapping: address is modulo the number of blocks in the cache



Example: Accessing a Cache

(See Figure 7.6)

- Initial state of an eight-word direct-mapped cache:

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

Example: Accessing a Cache

(See Figure 7.6)

- Handling reference #1 to address 22_{10} (10110_2):

Index	Valid	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Memory[10110 ₂]
111	N		

miss #1

Example: Accessing a Cache

(See Figure 7.6)

- Handling reference #2 to address 26_{10} (11010_2):

Index	Valid	Tag	Data
000	N		
001	N		
010	Y	11	Memory[11010_2]
011	N		
100	N		
101	N		
110	Y	10	Memory[10110_2]
111	N		

miss #2

Example: Accessing a Cache

(See Figure 7.6)

- Handling reference #3 to address 22_{10} (10110_2):

Index	Valid	Tag	Data
000	N		
001	N		
010	Y	11	Memory[11010 ₂]
011	N		
100	N		
101	N		
110	Y	10	Memory[10110 ₂]
111	N		

hit

Example: Accessing a Cache

(See Figure 7.6)

- Handling reference #4 to address 26_{10} (11010_2):

Index	Valid	Tag	Data
000	N		
001	N		
010	Y	11	Memory[11010 ₂]
011	N		
100	N		
101	N		
110	Y	10	Memory[10110 ₂]
111	N		

hit

Example: Accessing a Cache

(See Figure 7.6)

- Handling reference #5 to address 16_{10} (10000_2):

Index	Valid	Tag	Data
000	Y	10	Memory[10000 ₂]
001	N		
010	Y	11	Memory[11010 ₂]
011	N		
100	N		
101	N		
110	Y	10	Memory[10110 ₂]
111	N		

miss #3

Example: Accessing a Cache

(See Figure 7.6)

- Handling reference #6 to address 3_{10} (00011_2):

Index	Valid	Tag	Data
000	Y	10	Memory[10000 ₂]
001	N		
010	Y	11	Memory[11010 ₂]
011	Y	00	Memory[00011 ₂]
100	N		
101	N		
110	Y	10	Memory[10110 ₂]
111	N		

miss #4

Example: Accessing a Cache

(See Figure 7.6)

- Handling reference #7 to address 16_{10} (10000_2):

Index	Valid	Tag	Data
000	Y	10	Memory[10000 ₂]
001	N		
010	Y	11	Memory[11010 ₂]
011	Y	00	Memory[00011 ₂]
100	N		
101	N		
110	Y	10	Memory[10110 ₂]
111	N		

hit

Example: Accessing a Cache

(See Figure 7.6)

- Handling reference #8 to address 18_{10} (10010_2):

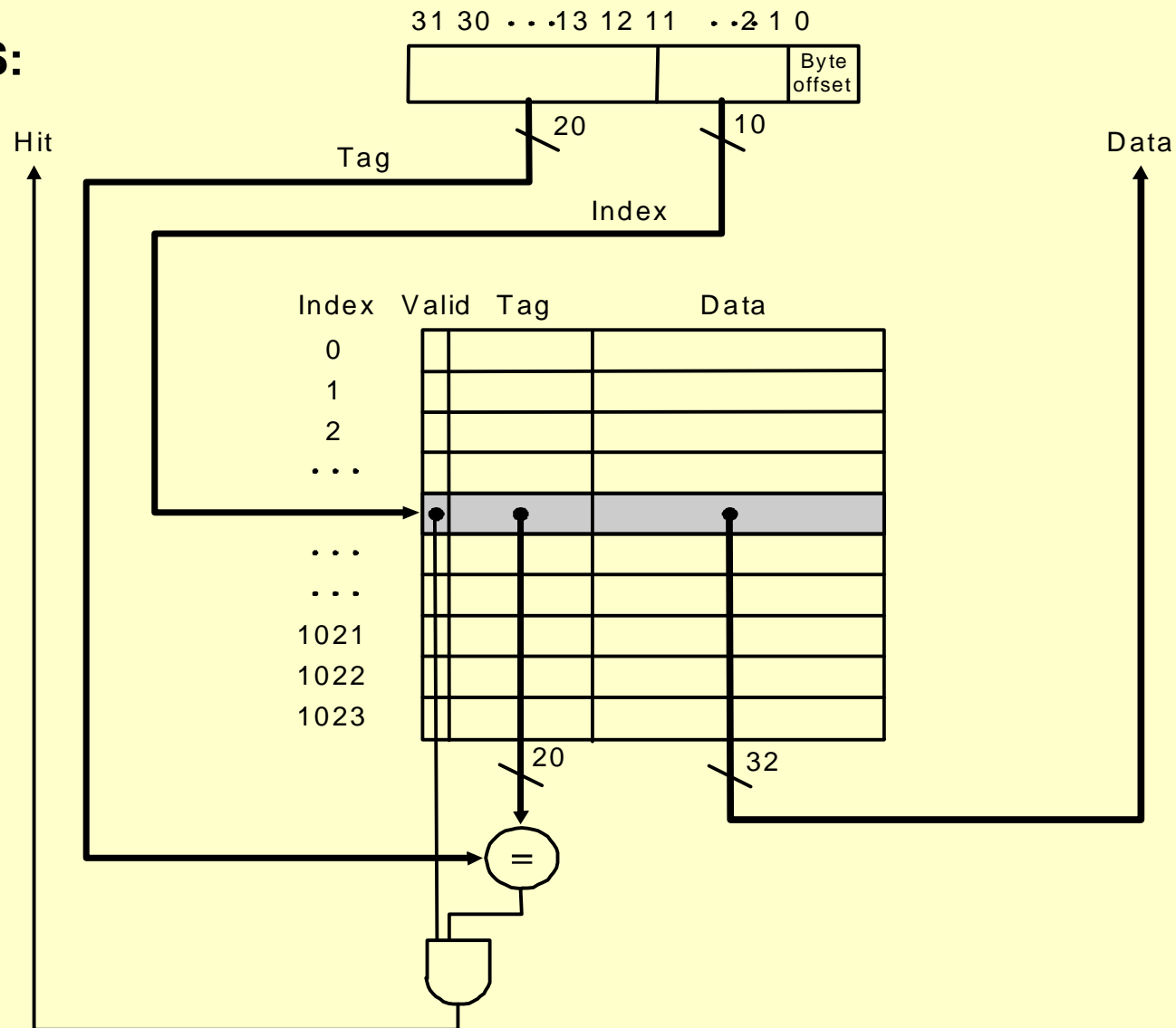
Index	Valid	Tag	Data
000	Y	10	Memory[10000 ₂]
001	N		
010	Y	10	Memory[10010 ₂]
011	Y	00	Memory[00011 ₂]
100	N		
101	N		
110	Y	10	Memory[10110 ₂]
111	N		

miss #5

Direct Mapped Cache

(See Figure 7.7)

- For MIPS:

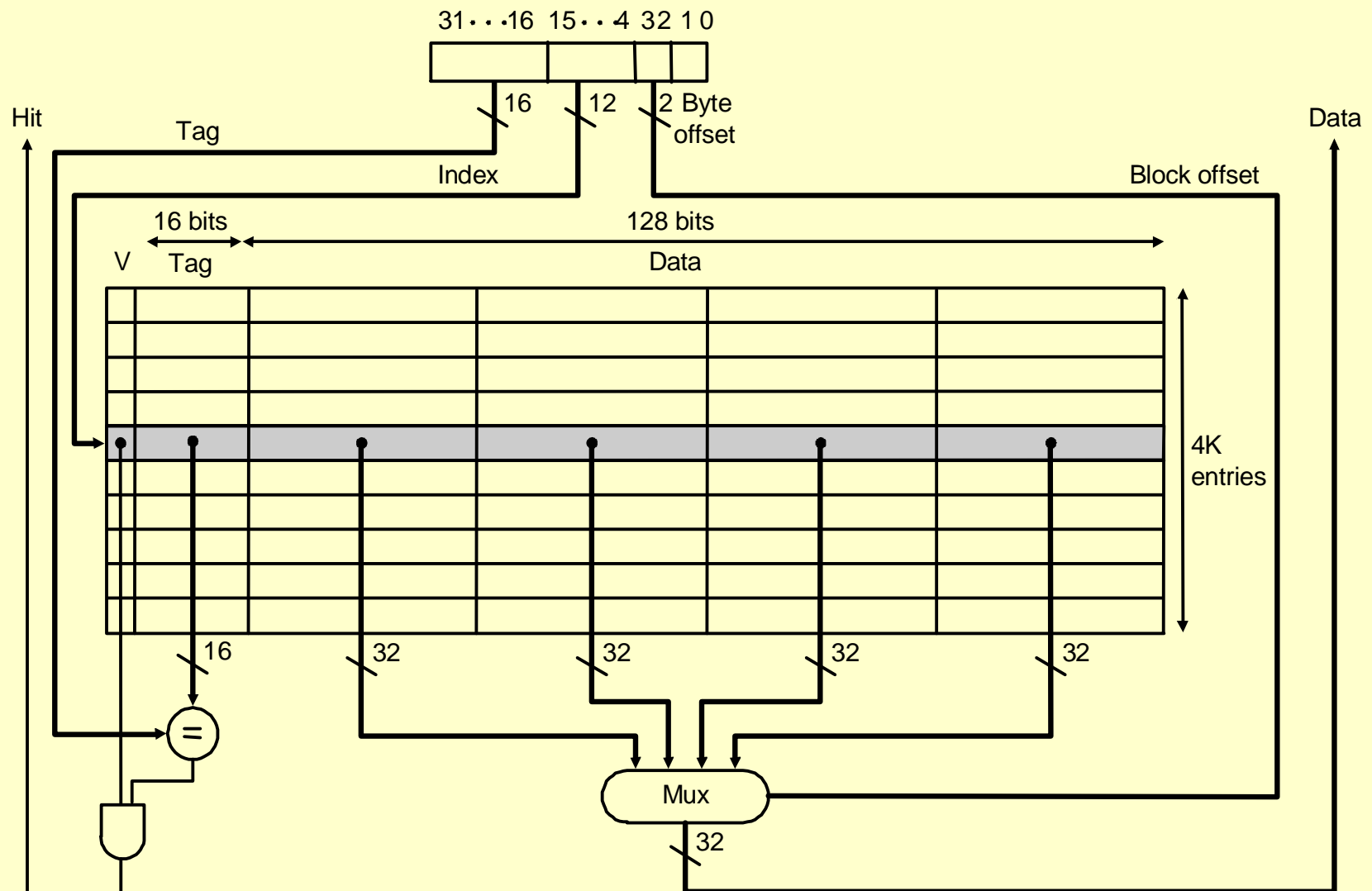


This only takes care of *temporal locality* ...

Direct Mapped Cache

(See Figure 7.10)

- Taking advantage of *spatial locality*:

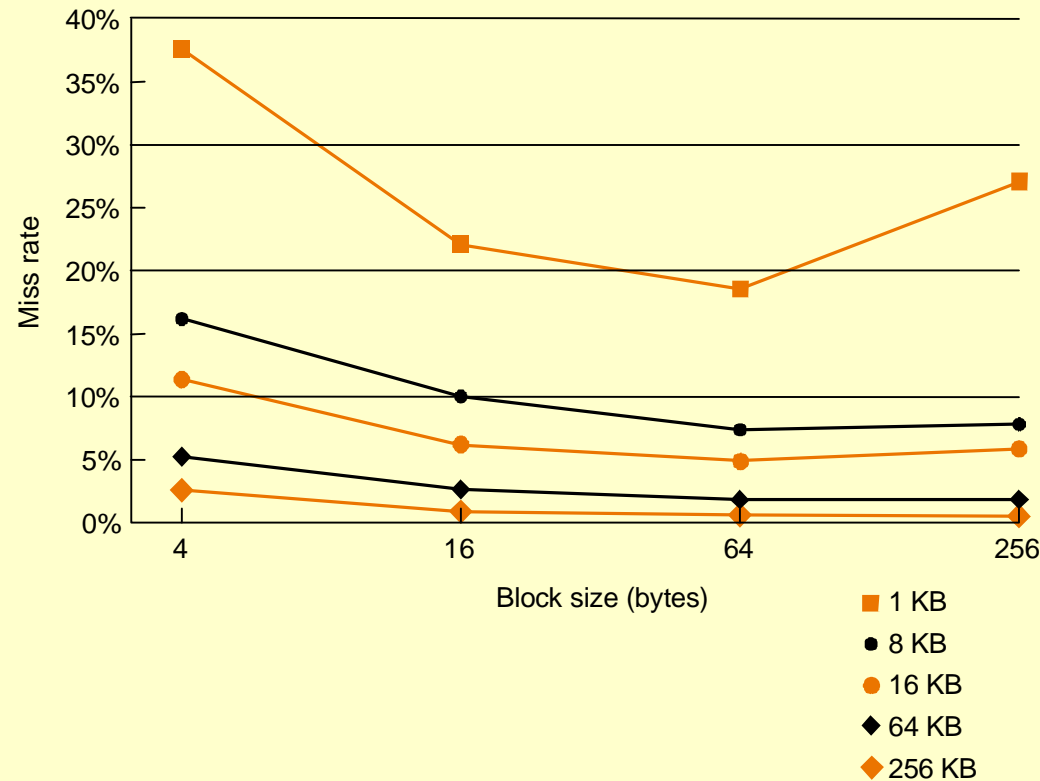


Hits vs. Misses

- Read *hits*
 - this is what we want!
- 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

Performance

- Increasing the block size tends to decrease miss rate:



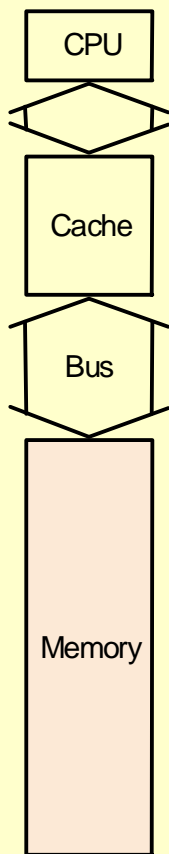
- Use *split caches* because there is more spatial locality in code:

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%

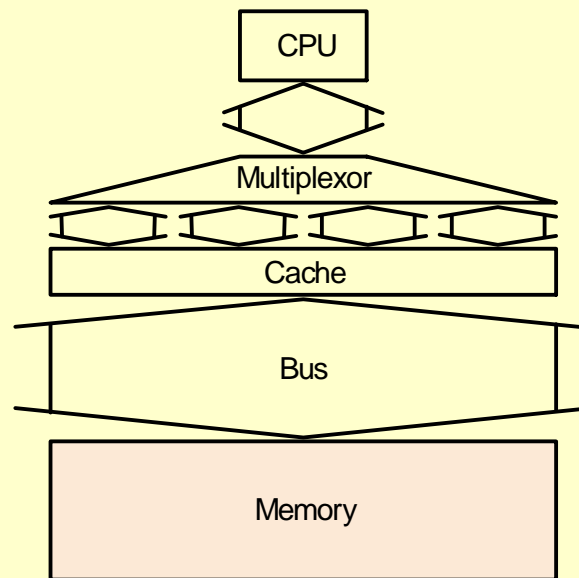
Hardware Issues

(See Figure 7.13)

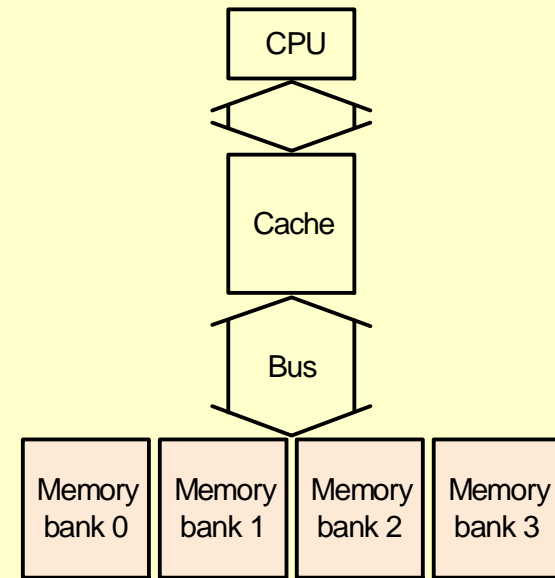
- **Make reading multiple words easier by using banks of memory**



a. One-word-wide memory organization



b. Wide memory organization



c. Interleaved memory organization

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

Elaboration: Hardware Issues

(See Figure 7.13)

- **Four-word cache block with a one-word-wide memory bank:**

- **Given:**

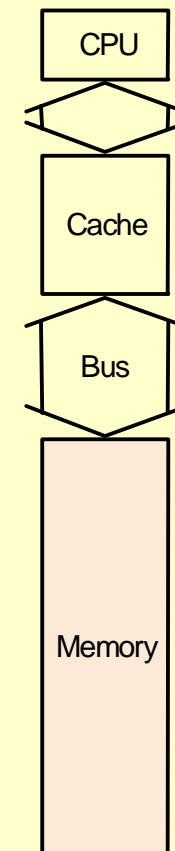
- 1 clock cycle / address sent
- 15 clock cycles / DRAM access
- 1 clock cycle / data word sent

- **Miss penalty:**

- $1 + 4 \cdot 15 + 4 \cdot 1 = 65$ clock cycles

- **Bytes transferred per clock cycle:**

- $(4 \text{ bytes/word} \cdot 4 \text{ words}) \cdot 65 \text{ cycles} = 0.25$



a. One-word-wide memory organization

Elaboration: Hardware Issues

(See Figure 7.13)

- **Four-word cache block with a two- and four-word-wide memory bank:**

- **Given:**

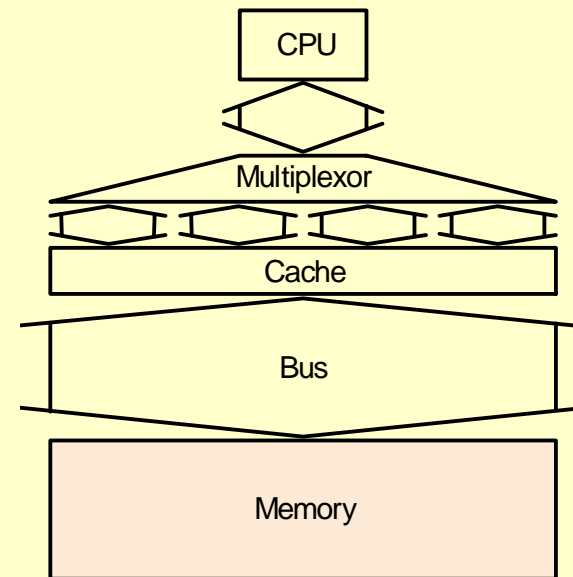
- 1 clock cycle / address sent
- 15 clock cycles / DRAM access
- 1 clock cycle / data word sent

- **Miss penalty:**

- $1 + 2 \cdot 15 + 2 \cdot 1 = 33$ clock cycles
- $1 + 1 \cdot 15 + 1 \cdot 1 = 17$ clock cycles

- **Bytes transferred per clock cycle:**

- $(4 \text{ bytes/word} \cdot 4 \text{ words}) \cdot 33 \text{ cycles} = 0.48$
- $(4 \text{ bytes/word} \cdot 4 \text{ words}) \cdot 17 \text{ cycles} = 0.94$



b. Wide memory organization

Elaboration: Hardware Issues

(See Figure 7.13)

- **Four-word cache block with a four-bank *interleaved* memory:**

- **Given:**

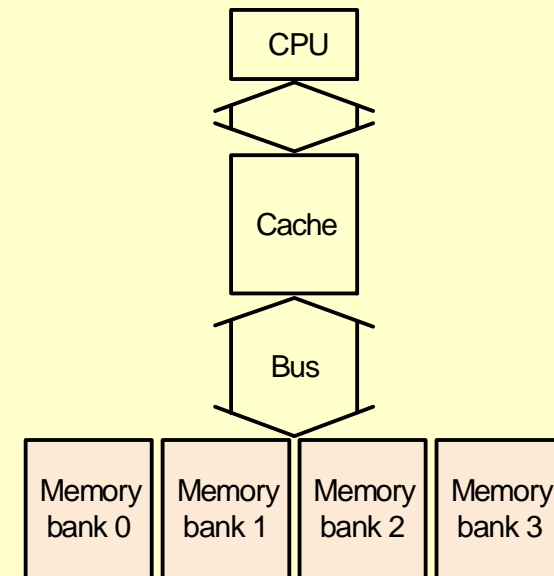
- 1 clock cycle / address sent
- 15 clock cycles / DRAM access
- 1 clock cycle / data word sent

- **Miss penalty:**

- $1 + 1 + 15 + 4 + 1 = 20$ clock cycles

- **Bytes transferred per clock cycle:**

- $(4 \text{ bytes/word} \times 4 \text{ words}) \div 20 \text{ cycles} = 0.80$



c. Interleaved memory organization

Measuring Performance

- Simplified model:

execution time = (execution cycles + stall cycles) ? clock cycle time

stall cycles = # of instructions ? *miss ratio* ? *miss penalty*

- Therefore, there are two ways to improve performance:
 - decreasing the *miss ratio*
 - decreasing the *miss penalty*

Recall what happens if we increase block size?

Measuring Performance

(See pages 565-566)

- **Example:** Assume an instruction cache miss for `gcc` of 2% and a data cache miss rate of 4%. If a machine has a CPI of 2 without any memory stalls and the miss penalty is 40 cycles for all misses, determine how much faster a machine would run with a perfect cache that never missed.
(Use the instruction frequencies for `gcc` from Chapter 4, Figure 4.54, on page 311.)

- **Answer** (let I be the instruction count; frequency of loads/stores in `gcc` is 36%):

$$\text{Instruction miss cycles} = I \cdot 2\% \cdot 40 = 0.80I$$

$$\text{Data miss cycles} = I \cdot 36\% \cdot 4\% \cdot 40 = 0.56I$$

$$\text{Total memory-stall cycles} = 0.80I + 0.56I = 1.36I$$

$$\text{CPI with memory-stalls} = 2 + 1.36 = 3.36$$

$$\frac{\text{CPU-time}_{\text{stall}}}{\text{CPU-time}_{\text{perfect}}} = \frac{I \cdot \text{CPI}_{\text{stall}} \cdot \text{Clock cycle}}{I \cdot \text{CPI}_{\text{perfect}} \cdot \text{Clock cycle}} = \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{3.36}{2} = 1.68$$

Measuring Performance

(See pages 565-566)

- **Question:** What happens if the processor is made faster, but the memory system stays the same?
- **Note:** By Amdahl's Law, if the processor is made faster, the amount of time spent on memory stalls will take up an increasing fraction of the execution time.
- **Answer** (assume that the speedup results in a new CPI of 1 from a CPI of 2):

$$\text{CPI}_{\text{stall}} = 1 + 1.36 = 2.36$$

So, the system with a perfect cache would be

$$\frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{2.36}{1} = 2.36 \text{ times faster}$$

The amount of execution time spent on memory stalls would have risen from

$$\frac{\text{Memory stall cycles}_{\text{old}}}{\text{CPI}_{\text{old}}} = \frac{1.36}{3.36} = 41\%$$

to

$$\frac{\text{Memory stall cycles}_{\text{new}}}{\text{CPI}_{\text{new}}} = \frac{1.36}{2.36} = 58\%$$

Measuring Performance

(See page 567)

- **Example:** Suppose we increase the performance of the machine in the previous example by doubling its clock rate. Since the main memory speed is unlikely to change, assume that the absolute time to handle a cache miss does not change. How much faster will the machine be with the faster clock, assuming the same miss rate as the previous example?
- **Answer** (doubling the clock rate, miss penalty of faster machine is 80 clock cycles):

$$\text{Total miss cycles per instruction} = 2\% \cdot 80 + 36\% \cdot 4\% \cdot 80 = 2.75$$

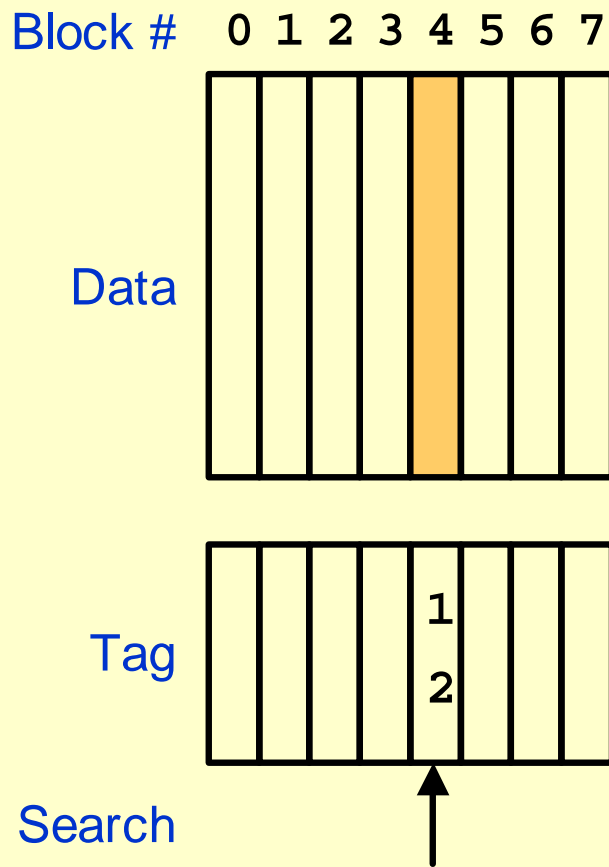
$$\text{Recall, } CPI_{\text{slow}} = CPI_{\text{stall}} = 3.36$$

$$CPI_{\text{fast}} = 2 + 2.75 = 4.75$$

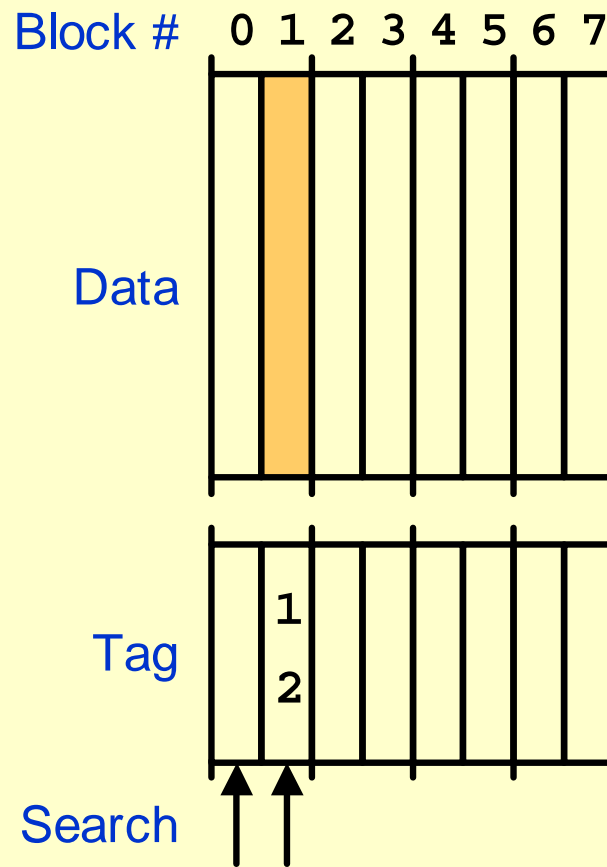
$$\frac{\text{CPU-time}_{\text{slow}}}{\text{CPU-time}_{\text{fast}}} = \frac{I \cdot CPI_{\text{slow}} \cdot \text{Clock cycle}}{I \cdot CPI_{\text{fast}} \cdot \frac{\text{Clock cycle}}{2}} = \frac{3.36}{4.75 \cdot \frac{1}{2}} = 1.41$$

Variations in Placement of Blocks

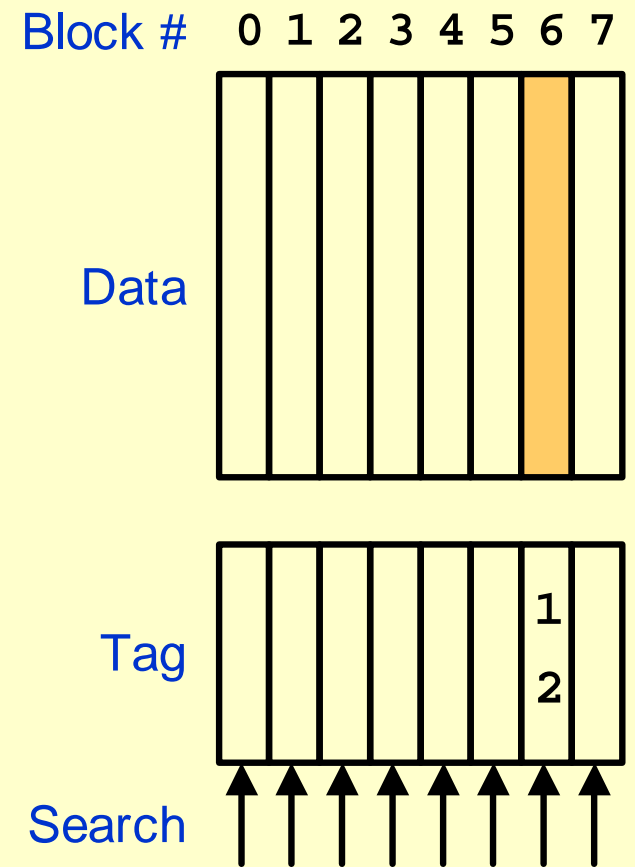
(See Figure 7.15)



Direct mapped



Set associative



Fully associative

Decreasing miss ratio with associativity (See Figure 7.16)

(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

Compared to direct mapped, give a series of references that:

- results in a lower miss ratio using a 2-way set associative cache
- results in a higher miss ratio using a 2-way set associative cache

assuming we use the “least recently used” replacement strategy

Associativity in Caches

(See pages 571-572)

- Referencing 0, 8, 0, 6, 8 from a *direct-mapped*, four one-word block cache:

– Mapping:

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

– Cache contents:

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]	

– Total number of cache misses = 5

Associativity in Caches

(See pages 571-572)

- Referencing 0, 8, 0, 6, 8 from a *two-way, set-associative*, four one-word block cache:

– Mapping:

Block address	Cache set
0	$(0 \text{ modulo } 2) = 0$
6	$(6 \text{ modulo } 2) = 0$
8	$(8 \text{ modulo } 2) = 0$

– Cache contents:

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]		

– Total number of cache misses = 4

Associativity in Caches

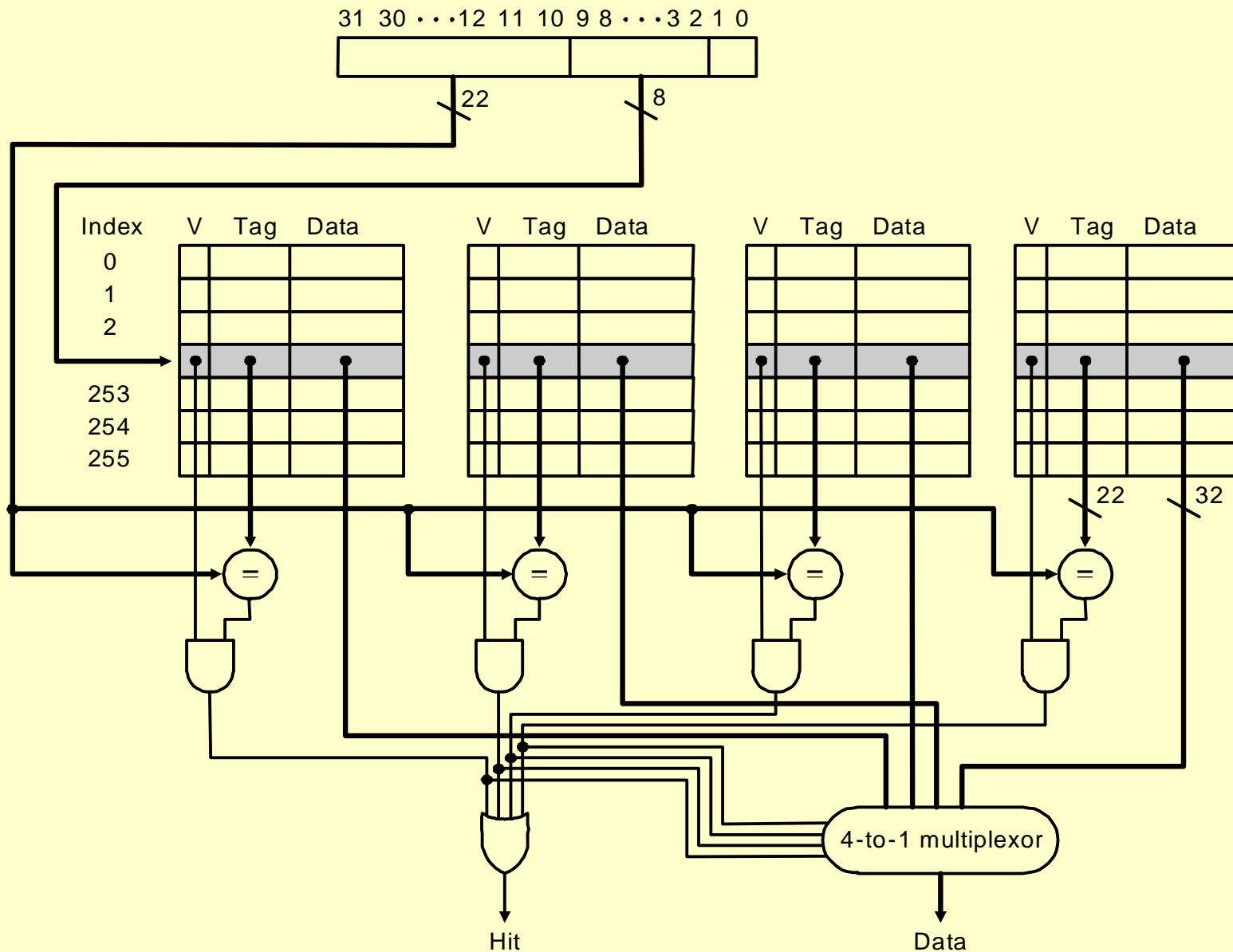
(See pages 571-572)

- Referencing 0, 8, 0, 6, 8 from a *fully associative*, four one-word block cache:
 - **Mapping:** *Any memory block can be stored in any cache block.*
 - **Cache contents:**

Address of memory	Hit	Contents of cache blocks after reference			
block accessed	or miss	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]	

- **Total number of cache misses = 3**

An Implementation of an Associative Cache



Tag Size versus Set Associativity

(See page 575)

- **Example:** Increasing associativity requires more comparators, as well as more tag bits per cache block. Assuming a cache of 4K blocks, a four-word block size, and a 32-bit address, find the *total number of sets*, n_S , and the *total number of tag bits*, n_B , for caches that are direct mapped, two-way and four-way set associative, and fully associative.

- **Answer:**

- direct mapped:

$n_S = \text{number of blocks} = 4K$, so 12 bits of index

$$n_B = (32-4-12) \cdot 4K = 64K \text{ bits}$$

- two-way and four-way set associative:

$n_S = \text{no. of blocks} \cdot \text{no. of blocks per set} = 4K \cdot 2 = 2K$ [$4K \cdot 4 = 1K$]

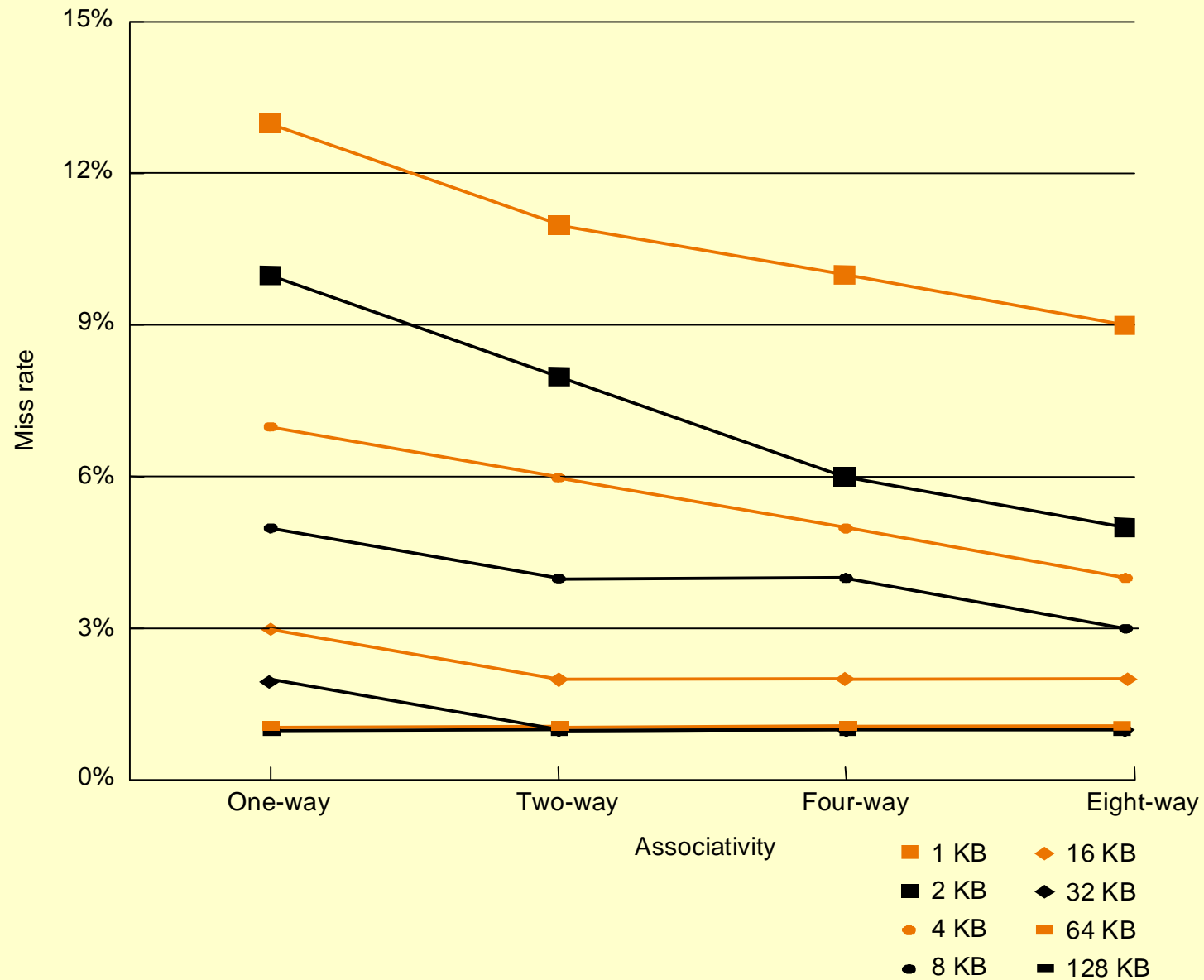
$$n_B = (32-4-11) \cdot 2 \cdot 2K = 68K \text{ bits} \quad [(32-4-10) \cdot 4 \cdot 1K = 72K \text{ bits}]$$

- fully associative:

$n_S = 1$ (with 4K blocks)

$$n_B = (32-4-0) \cdot 1 \cdot 4K = 112K \text{ bits}$$

Performance: miss rate and associativity (See Figure 7.29)



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 500Mhz machine with a 5% miss rate, 200ns DRAM access
 - Adding 2nd level cache with 20ns access time decreases miss rate to 2%
- **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

Performance: Multilevel Caches

(See pages 576-577)

- **Example:** Suppose we have a processor with a base CPI of 1.0, assuming all references hit in the primary cache, and a clock rate of 500 MHz. Assume a main memory access time of 200ns, including all miss handling. Suppose the miss rate per instruction at the primary cache is 5%. How much faster will the machine be if we add a secondary cache that has a 20ns access time for either a hit or a miss and is large enough to reduce the miss rate to main memory to 2%?

- **Answer:**

- For one level of caching:

- miss penalty (in cycles) to main memory

$$\frac{200 \text{ ns}}{500 \text{ MHz}} = \frac{200 \text{ ns}}{5 \times 10^8 \text{ cycles}} = \frac{200 \text{ ns}}{2 \times 10^{-9} \frac{\text{seconds}}{\text{cycle}}} = 100 \text{ cycles}$$

- effective CPI

$$1.0 + 5\% \times 100 = 6.0$$

base CPI memory-stall cycles per instruction

Performance: Multilevel Caches

(See pages 576-577)

- **Answer** (continued):

- With two levels of cache:

- miss penalty (in cycles) to second-level cache

$$\frac{20 \text{ ns}}{2 \times 10^{-9} \frac{\text{seconds}}{\text{cycle}}} = 10 \text{ cycles}$$

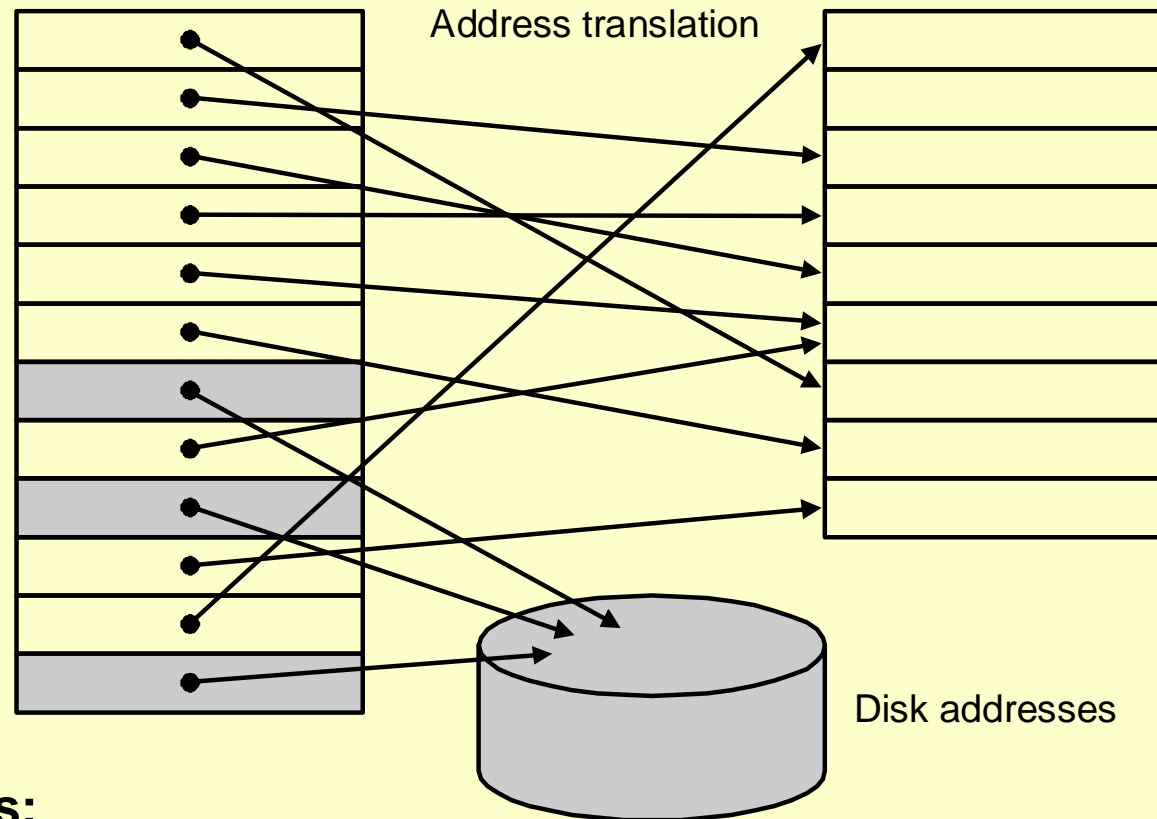
- effective CPI

$$\begin{aligned} \text{Total CPI} &= \text{base CPI} + \text{Primary-stall cycles per instruction} \\ &\quad + \text{Secondary-stall cycles per instruction} \\ &= 1.0 + ((5\% - 2\%) \times 10) \\ &\quad + (2\% \times (10 + 100)) \\ &= 1.0 + 0.3 + 2.2 = 3.5 \end{aligned}$$

Virtual Memory

(See Figure 7.20)

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

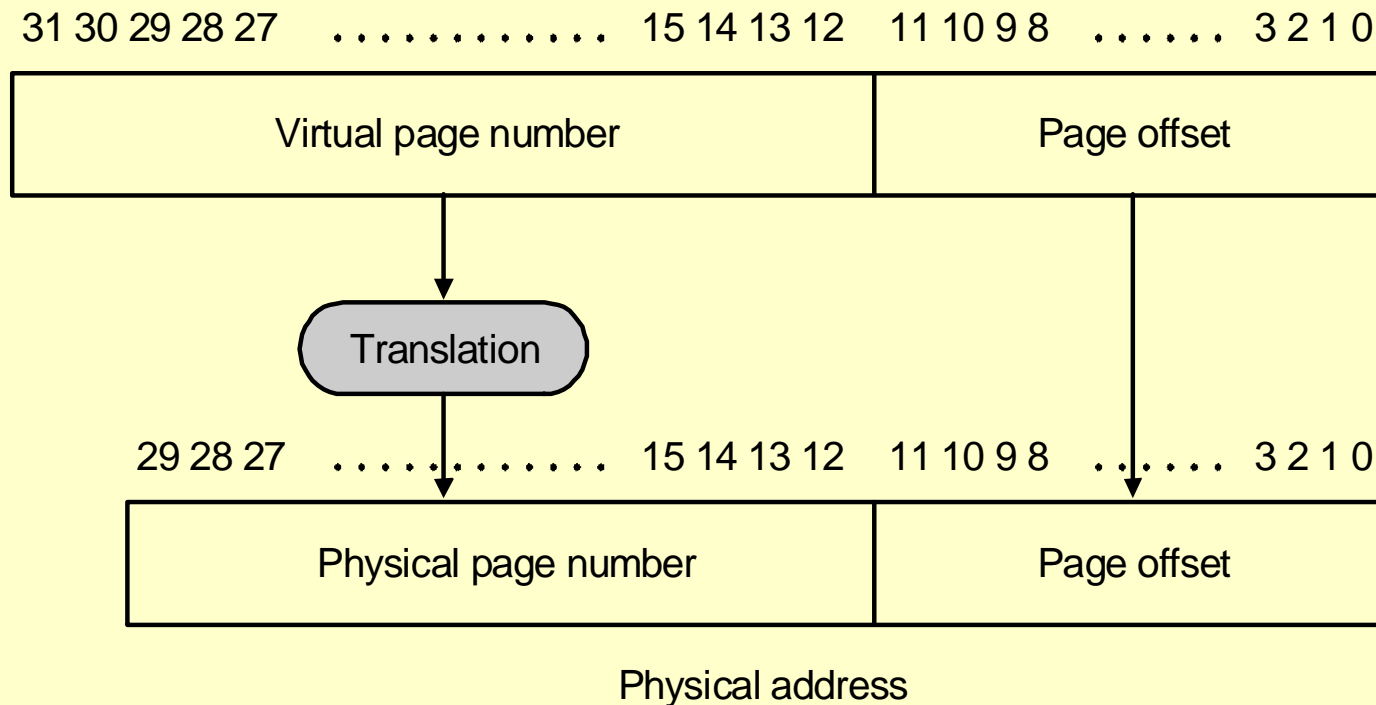


- Advantages:
 - illusion of having more physical memory
 - program relocation
 - protection

Pages: virtual memory blocks

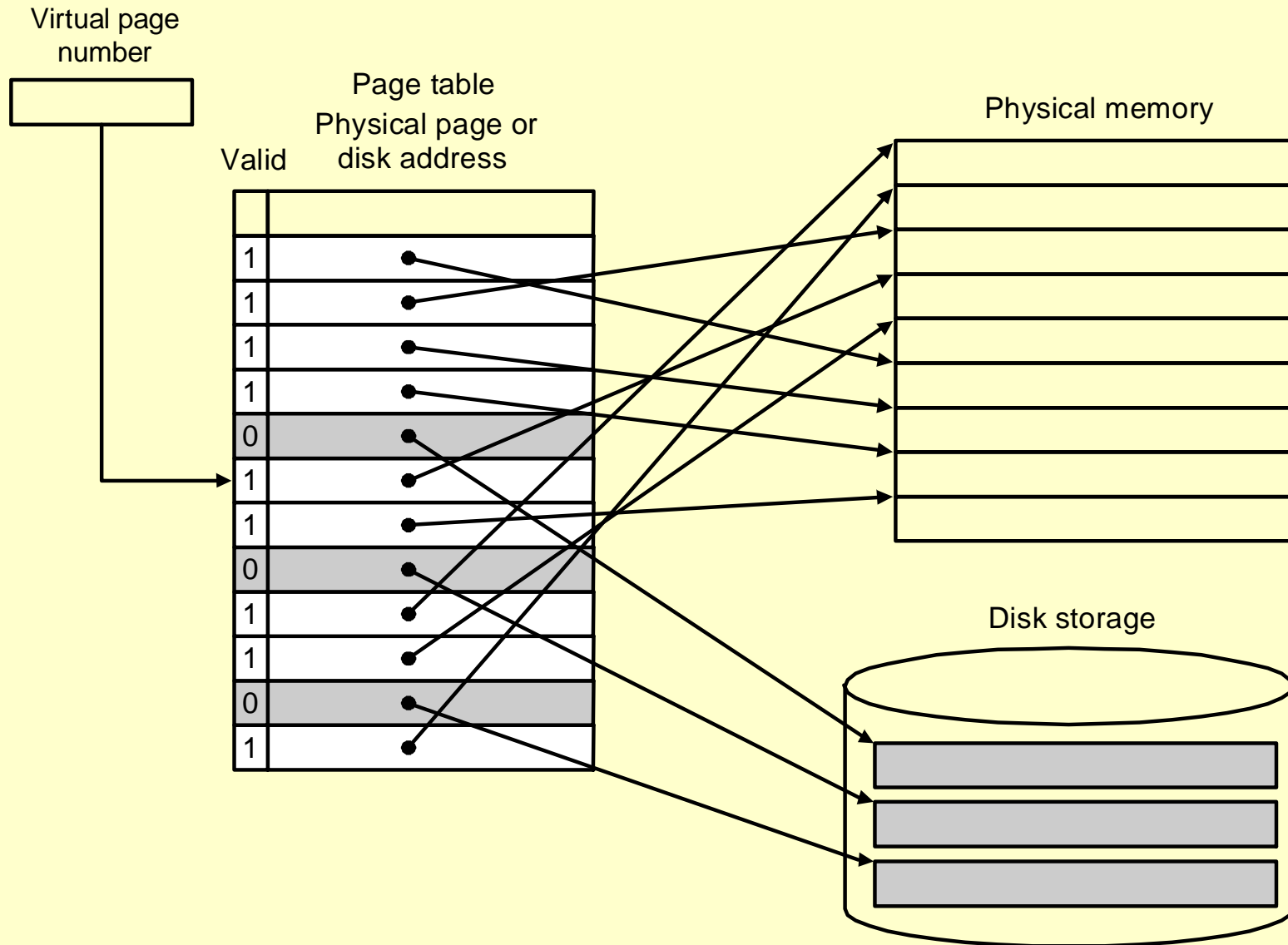
(See Figure 7.21)

- **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 *writeback*



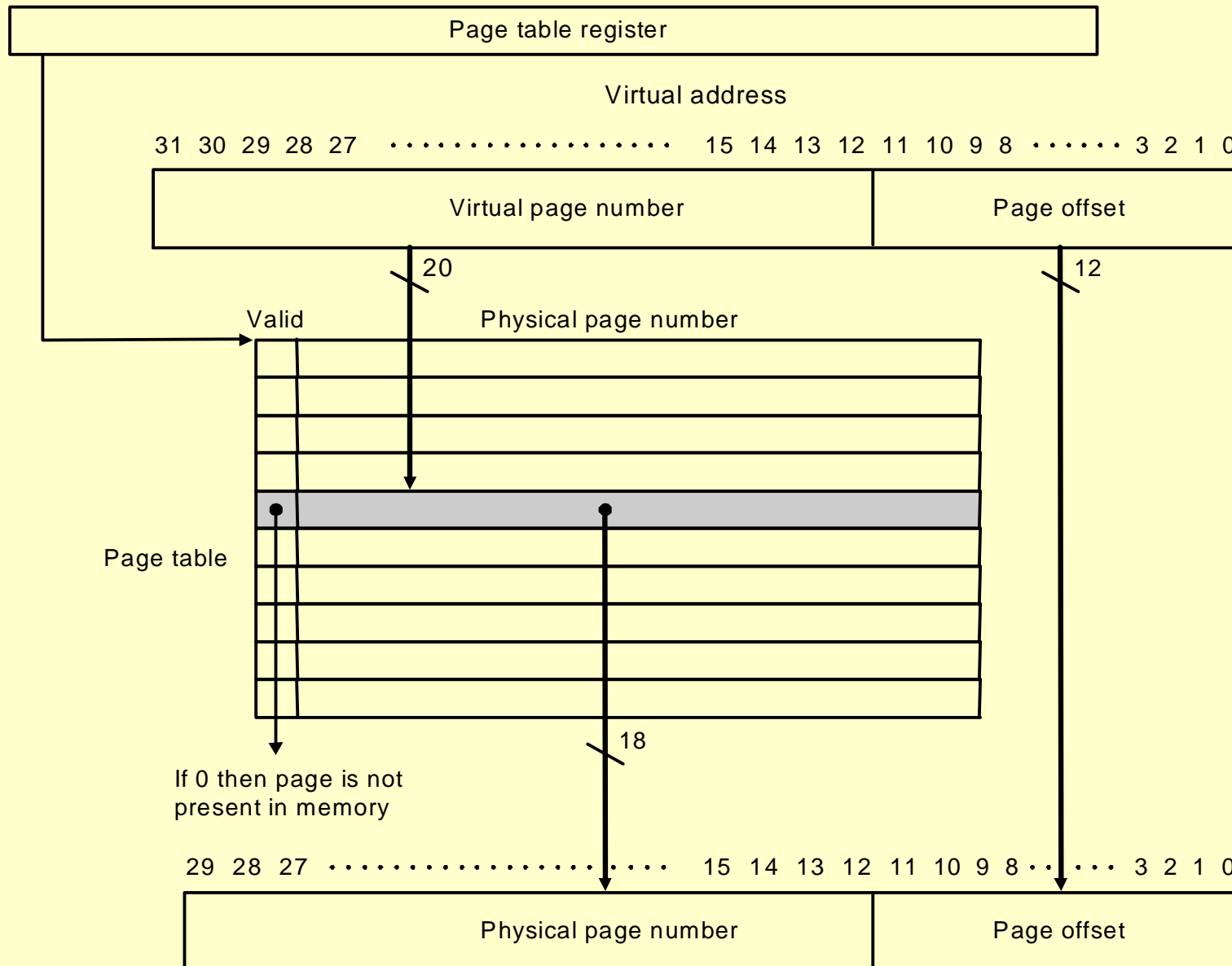
Page Tables

(See Figure 7.23)



Page Tables

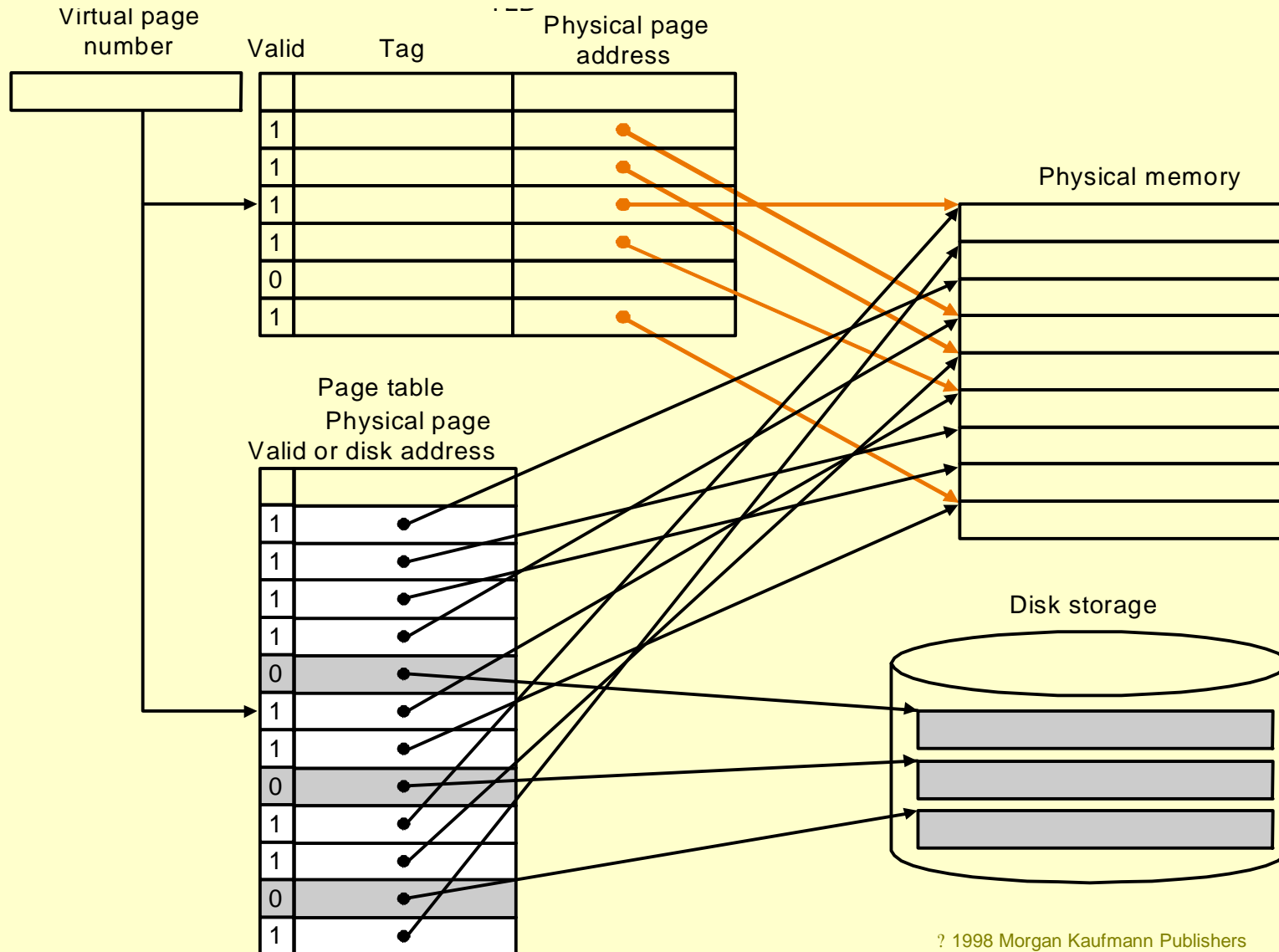
(See Figure 7.22)



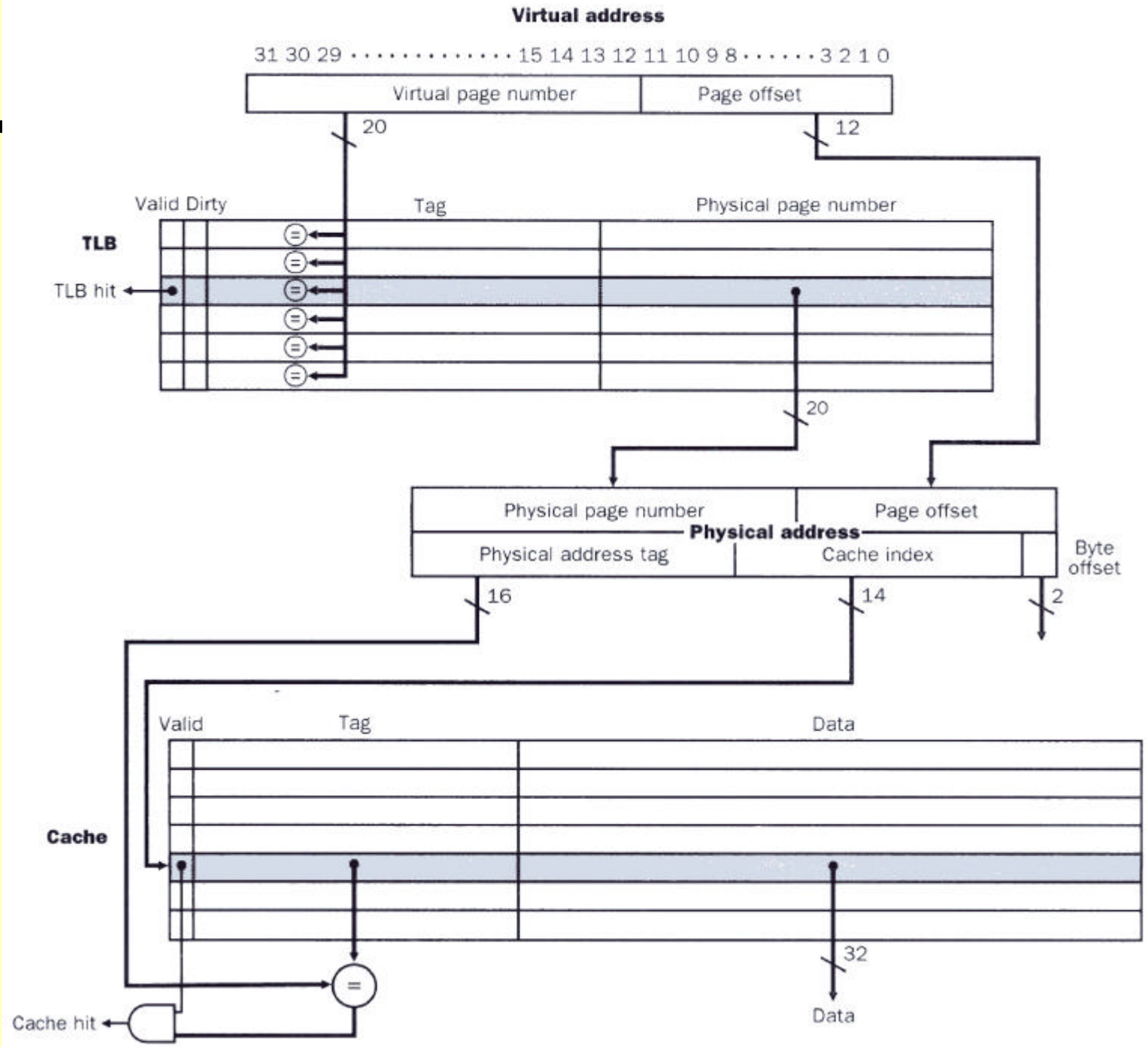
Making Address Translation Fast

(See Figure 7.24)

- A cache for address translations: *translation lookaside buffer* (TLB)



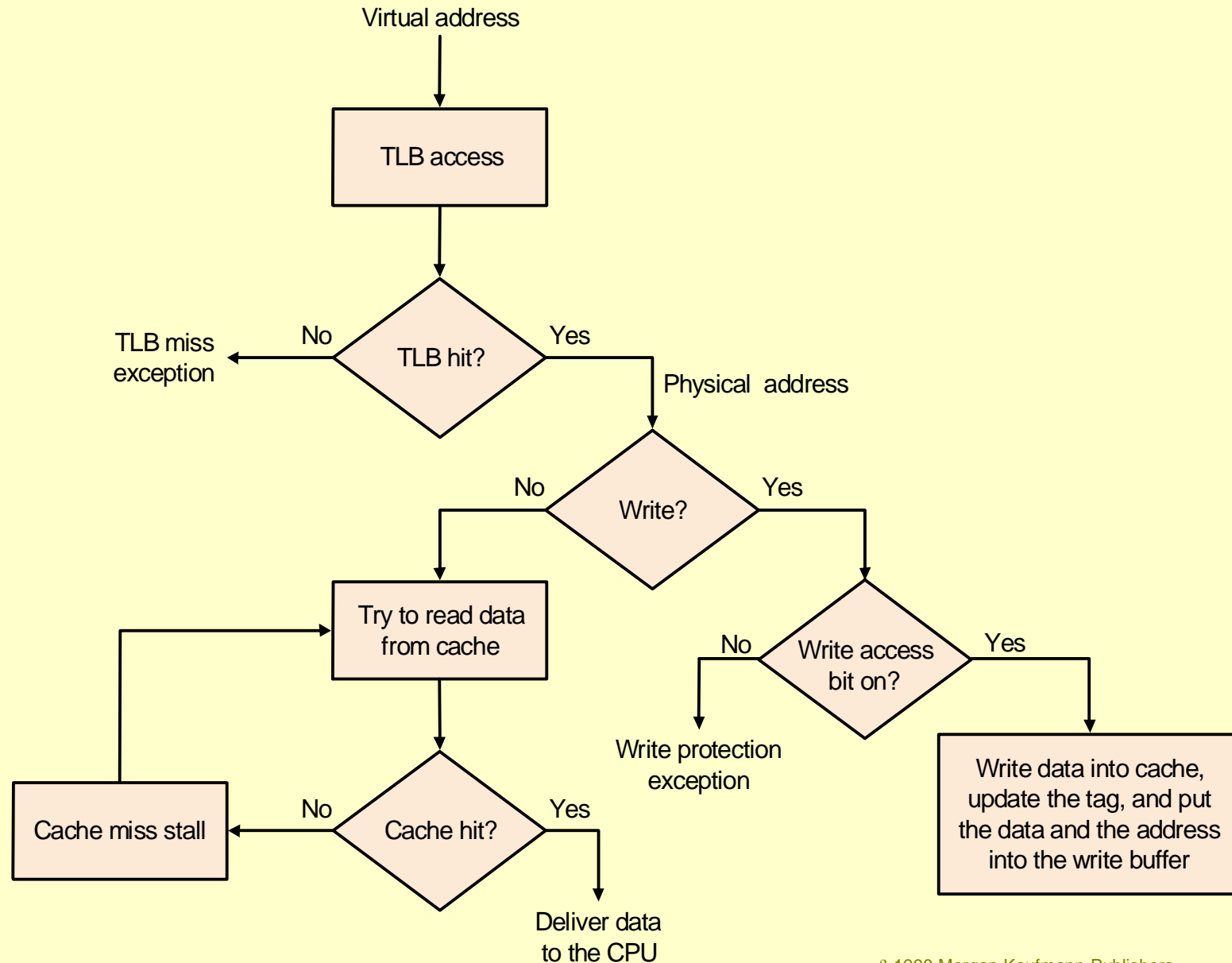
MIPS R2000 TLB



(See Figure 7.25)

TLBs and caches

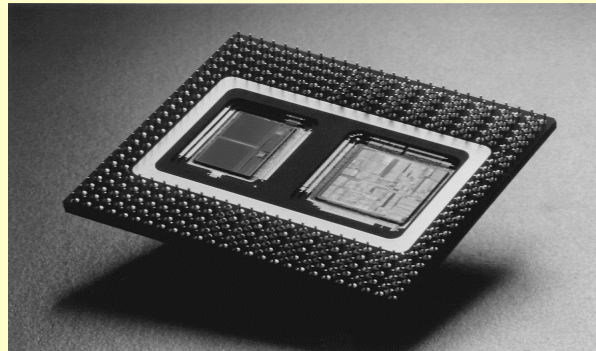
(See Figure 7.26)



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 instruction 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)