

Chapter 5

Large and Fast: Exploiting Memory Hierarchy

Recap

- Memory hierarchy
- Storage technologies
- Direct mapped cache
- Write policies

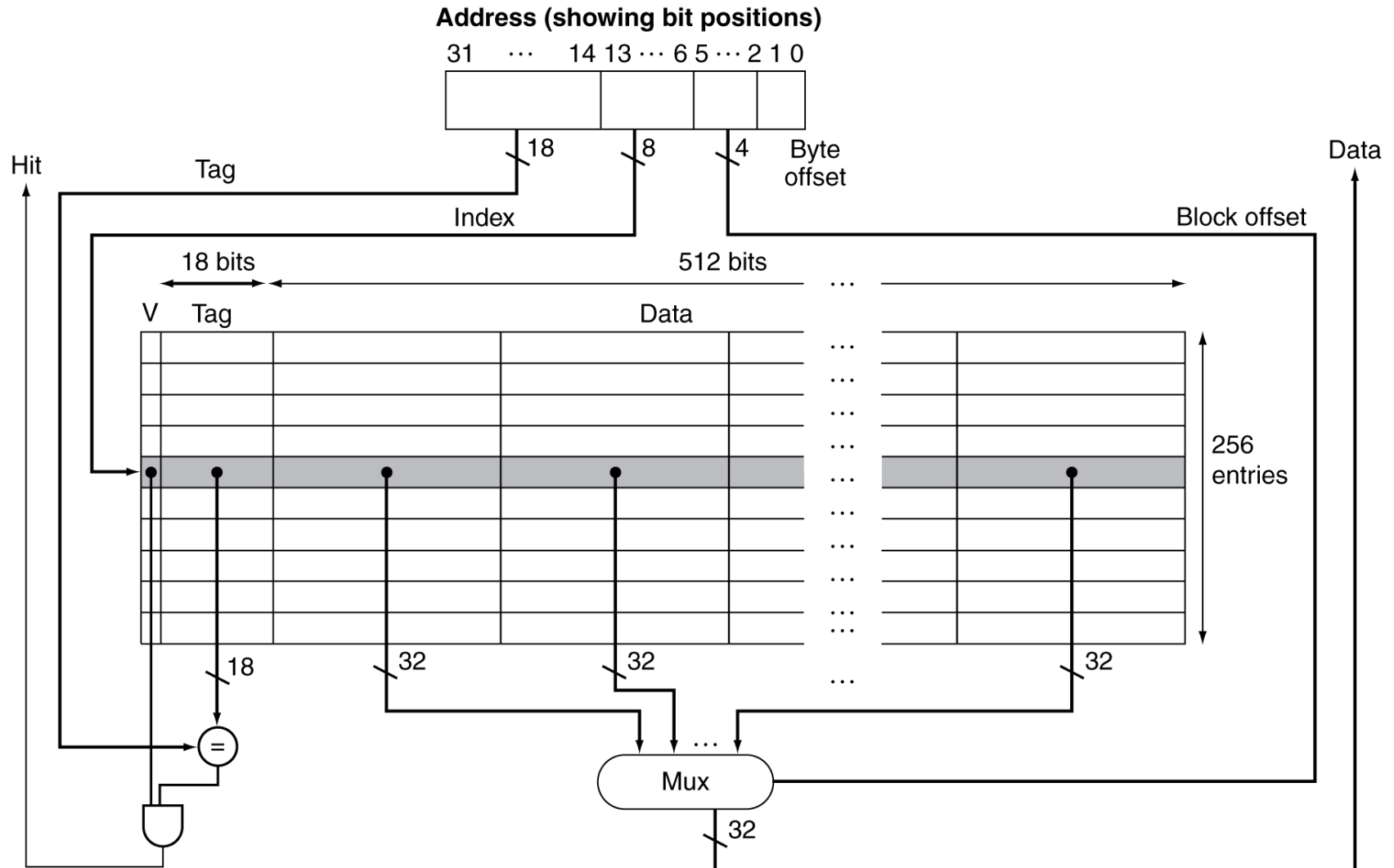
Write Policies Summary

- If that memory location is in the cache?
 - ◆ Send it to the cache
 - ◆ Should we also send it to memory right away?
(write-through policy)
 - ◆ Wait until we kick the block out (write-back policy)
- If it is not in the cache?
 - ◆ Allocate the line (put it in the cache)?
(write allocate policy)
 - ◆ Write it directly to memory without allocation?
(no write allocate policy)

Example: Intrinsity FastMATH

- Embedded MIPS processor
 - ◆ 12-stage pipeline
 - ◆ Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
 - ◆ Each 16KB: $256 \text{ blocks} \times 16 \text{ words/block}$
 - ◆ D-cache: write-through or write-back
- SPEC2000 miss rates
 - ◆ I-cache: 0.4%
 - ◆ D-cache: 11.4%
 - ◆ Weighted average: 3.2%

Example: Intrinsity FastMATH



Measuring Cache Performance

- Components of CPU time
 - ◆ Program execution cycles
 - Includes cache hit time
 - ◆ Memory stall cycles
 - Mainly from cache misses
- With simplifying assumptions:

Memory stall cycles

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

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Example

- Calculate actual CPI, given that
 - ◆ I-cache miss rate = 2%
 - ◆ D-cache miss rate = 4%
 - ◆ Miss penalty = 100 cycles
 - ◆ Base CPI (ideal cache) = 2
 - ◆ Load & stores are 36% of instructions
- Miss cycles per instruction (assume N ins. In total)
 - ◆ I-cache: $N \times 0.02 \times 100/N = 2$
 - ◆ D-cache: $N \times 0.36 \times 0.04 \times 100/N = 1.44$
- Actual CPI = $2 + 2 + 1.44 = 5.44$
 - ◆ Ideal CPU is $5.44/2 = 2.72$ times faster

Average Access Time

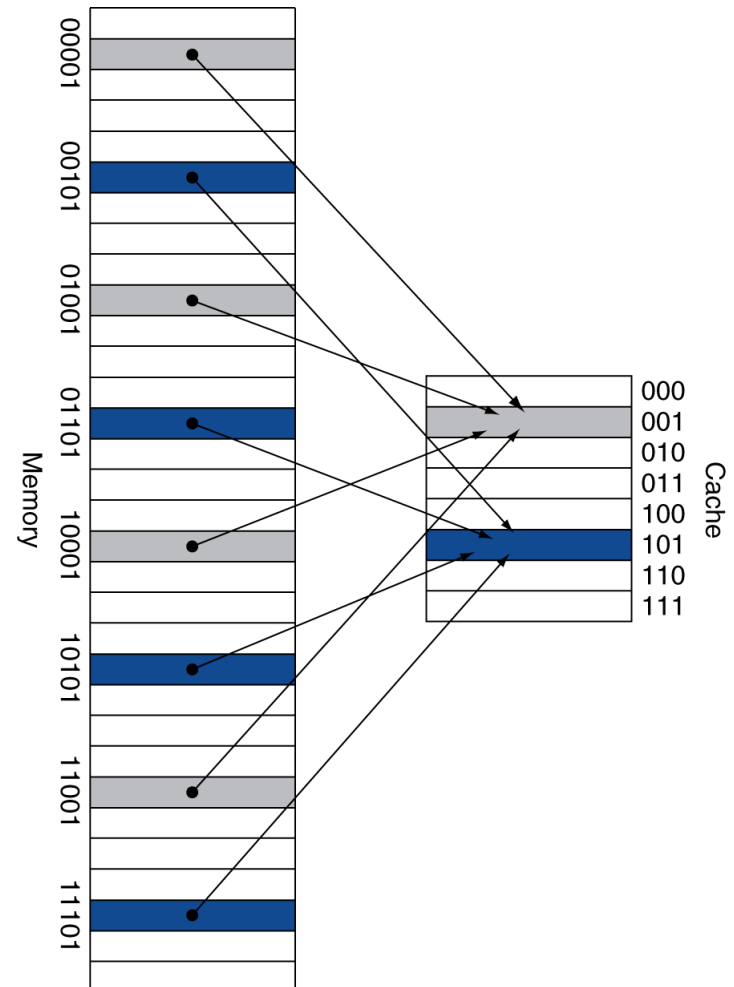
- Hit time is also important for performance
- Average memory access time (AMAT)
 - ◆ $\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
 - ◆ CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%
 - ◆ $\text{AMAT} = 1 + 0.05 \times 20 = 2\text{ns}$
 - 2 cycles per instruction

Performance Summary

- When CPU performance increased
 - ◆ Miss penalty becomes more significant
 - ◆ $\text{CPI}=2$, $\text{Miss}=3.44$, % of memory stall: $3.44/5.44=63\%$
 - ◆ $\text{CPI}=1$, $\text{Miss}=3.44$, % of memory stall: $3.44/4.44=77\%$
- Decreasing base CPI
 - ◆ Greater proportion of time spent on memory stalls
- Increasing clock rate
 - ◆ Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance

Recall: Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
 - ◆ Capacity of cache is not fully exploited
 - ◆ Miss rate is high



Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010
26	11 010	Miss	010
18	10 010	Miss	010

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

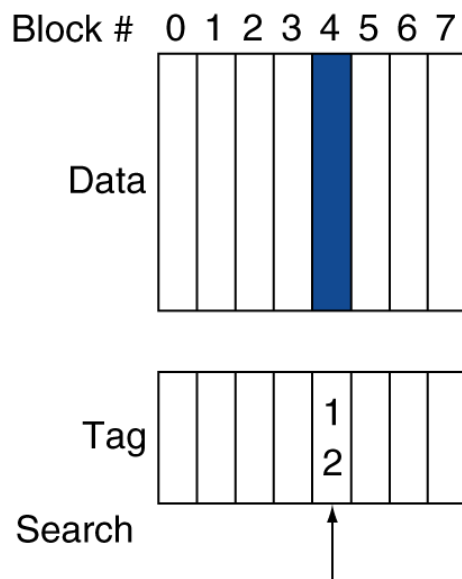
Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010
26	11 010	Miss	010
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		

Associative Cache Example

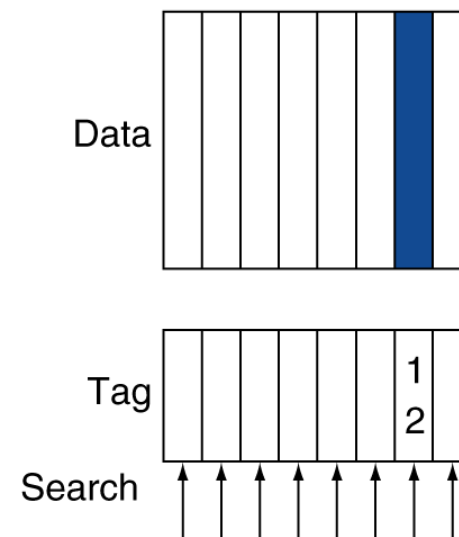
Direct mapped



Set associative

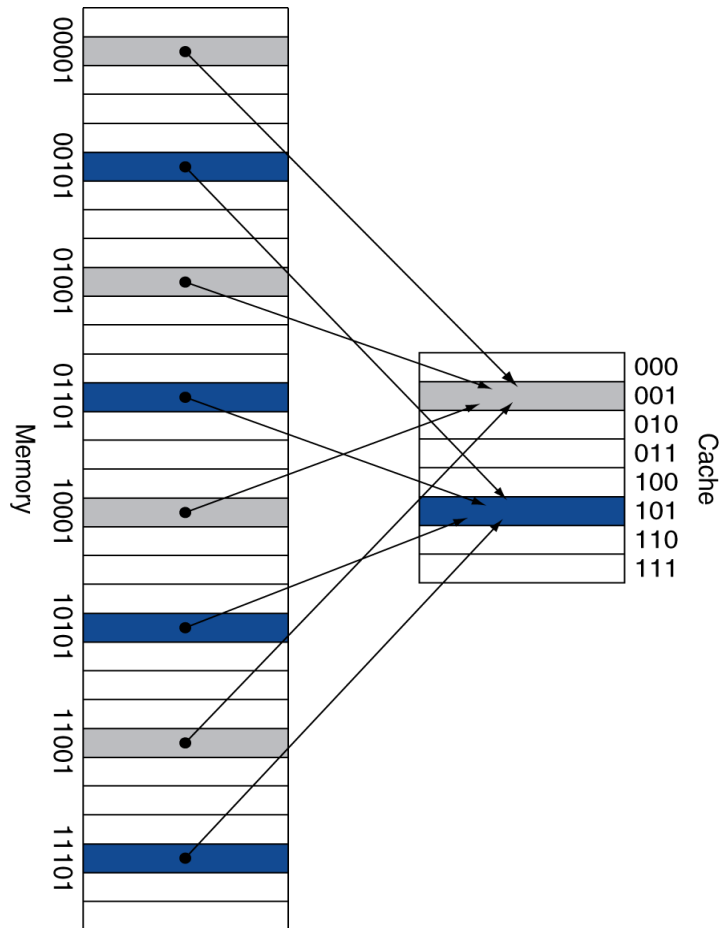


Fully associative

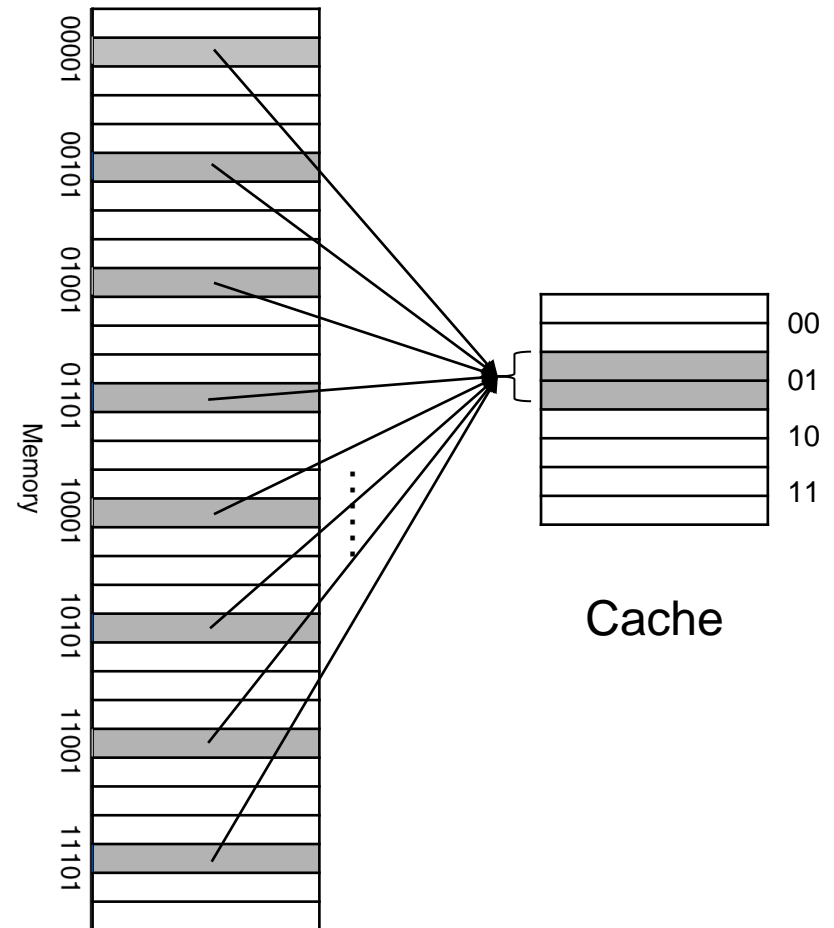


2-Way Set Associative Cache

Direct mapped cache



2-way set associative cache



Associative Caches

- Fully associative
 - ◆ Allow a given block to go in any cache entry
 - ◆ Requires all entries to be searched at once
 - ◆ Comparator per entry (expensive)
- n -way set associative
 - ◆ Each set contains n entries
 - ◆ Block number determines which set
 - (Block number) modulo (#Sets in cache)
 - ◆ Search all entries in a given set at once
 - ◆ n comparators (less expensive)

- For a cache with 8 blocks

Block	Tag	Data
-------	-----	------

Two-way set associative

Four-way set associative

Eight-way set associative (fully associative)

[illegible]

Associativity Example

- Assume a cache has 4 blocks
 - ◆ Compare three kinds of cache: Direct mapped, 2-way set associative, fully associative
 - ◆ Assume block size is 8 bytes, we access address 0, 64, 0, 48, 64 sequentially, then, the block sequence we accessed should be: 0, 8, 0, 6, 8
- Direct mapped

Tag: 27 bits	Index: 2 bits	Offset: 3 bits
--------------	---------------	----------------

 - ◆ Index: 2-bit (because the cache has 4 blocks)

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	

Associativity Example

■ 2-way set associative

Tag: 28 bits

Index: 1 bits

Offset: 3 bits

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

■ Fully associative

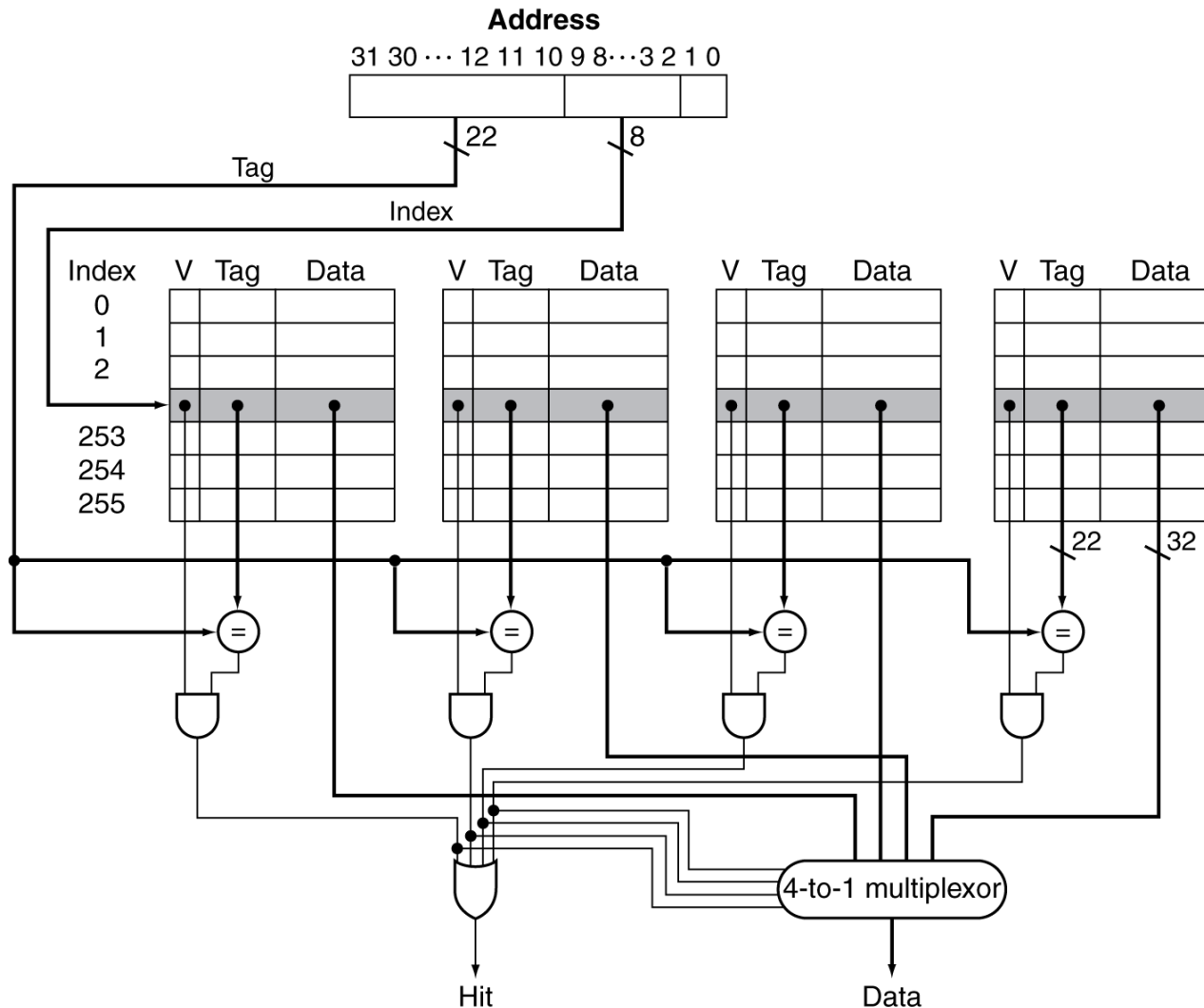
Tag: 29 bits

Index: 0 bits

Offset: 3 bits

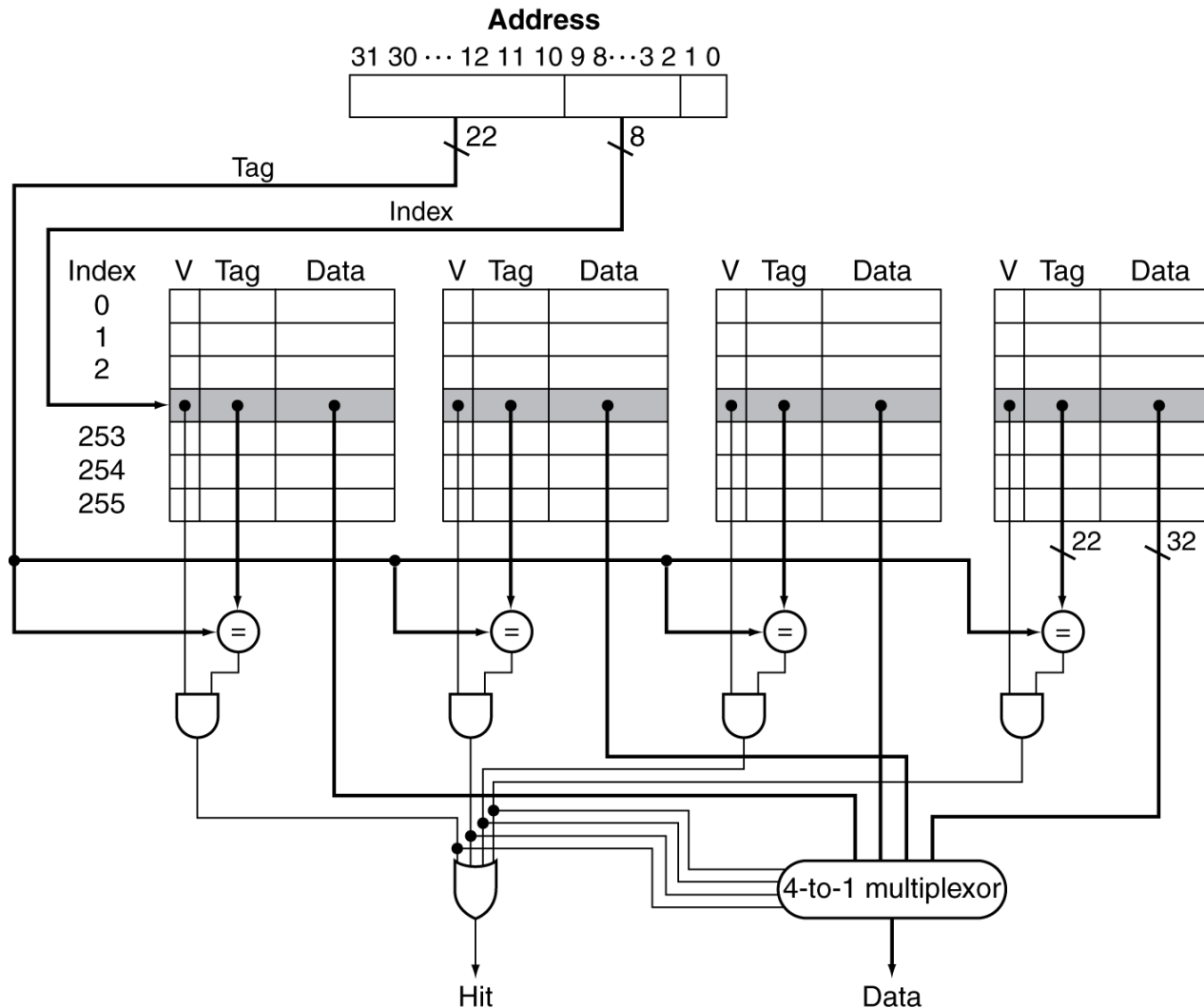
Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[8]	Mem[6]	

Set Associative Cache Organization



?-way set associative? Block size? How many sets in cache? How many tags?

Set Associative Cache Organization



4-way set associative. Block size: 4 bytes. 256 sets. 1024 tags.

How Much Associativity

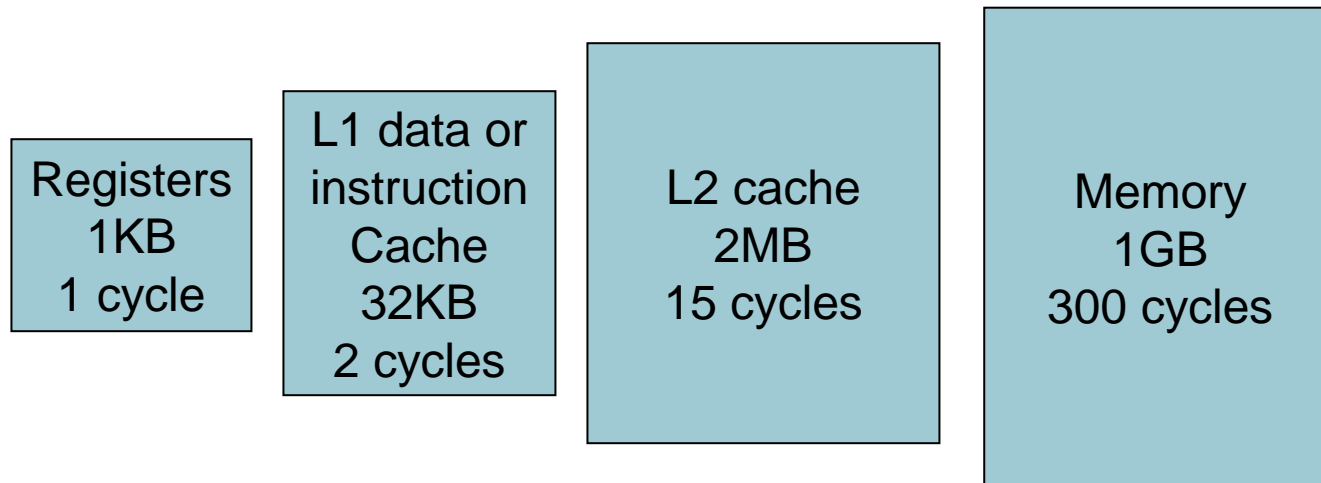
- Increased associativity decreases miss rate
 - ◆ But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
 - ◆ 1-way: 10.3%
 - ◆ 2-way: 8.6%
 - ◆ 4-way: 8.3%
 - ◆ 8-way: 8.1%

Replacement Policy

- Direct mapped: no choice, no replacement policy needed
- Set associative
 - ◆ Prefer non-valid entry, if there is one
 - ◆ Otherwise, choose among entries in the set
- Least-recently used (LRU)
 - ◆ Choose the one unused for the longest time
 - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
 - ◆ Gives approximately the same performance as LRU for high associativity

Multilevel Caches

- Primary cache (level-1 cache) attached to CPU
 - ◆ Small, but fast
- Level-2 cache services misses from level-1 cache
 - ◆ Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache



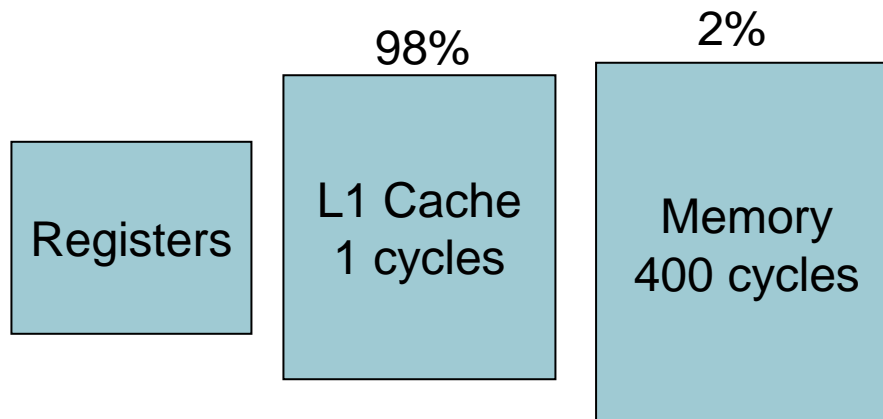
Multilevel Cache Example

- Given

- ◆ CPU base CPI = 1, clock rate = 4GHz
- ◆ Miss rate/instruction = 2%
- ◆ Main memory access time = 100ns

- With only L1 cache

- ◆ Miss penalty = $100\text{ns} / 0.25\text{ns} = 400$ cycles
- ◆ Effective CPI = $1 + 0.02 \times 400 = 9$



Example (cont.)

- Now add L-2 cache, calculate the new CPI, given

- ◆ Access time = 5ns
- ◆ Global miss ratio of L2 cache = 0.5%
(Local miss ratio of L2 = $0.5\% / 2\% = 25\%$)

- L-1 miss with L-2 hit

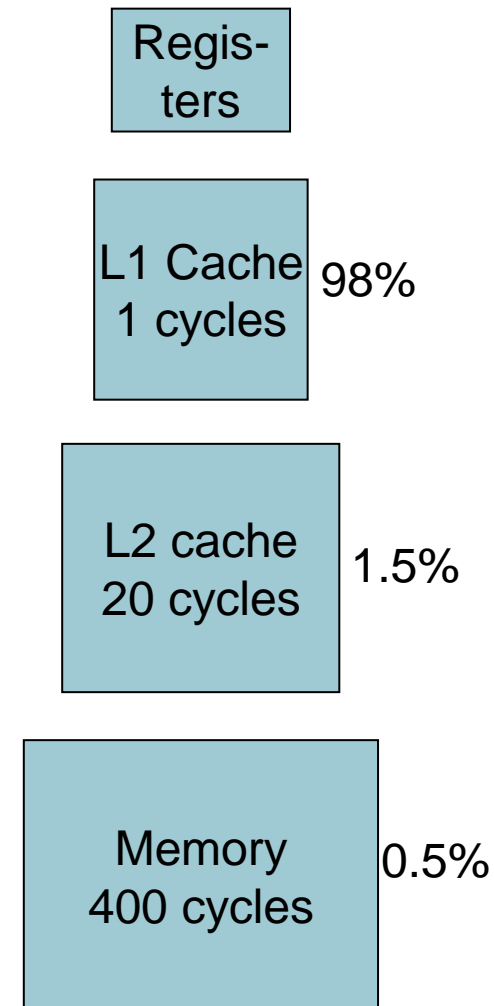
- ◆ Penalty = $5\text{ns} / 0.25\text{ns} = 20$ cycles

- L-1 miss with L-2 miss

- ◆ Extra penalty = 400 cycles

- $\text{CPI} = 0.98 \times 1 + 0.015 \times 21$
 $+ 0.005 \times 421 = 3.4$

- Performance ratio = $9 / 3.4 = 2.6$



Multilevel Cache Considerations

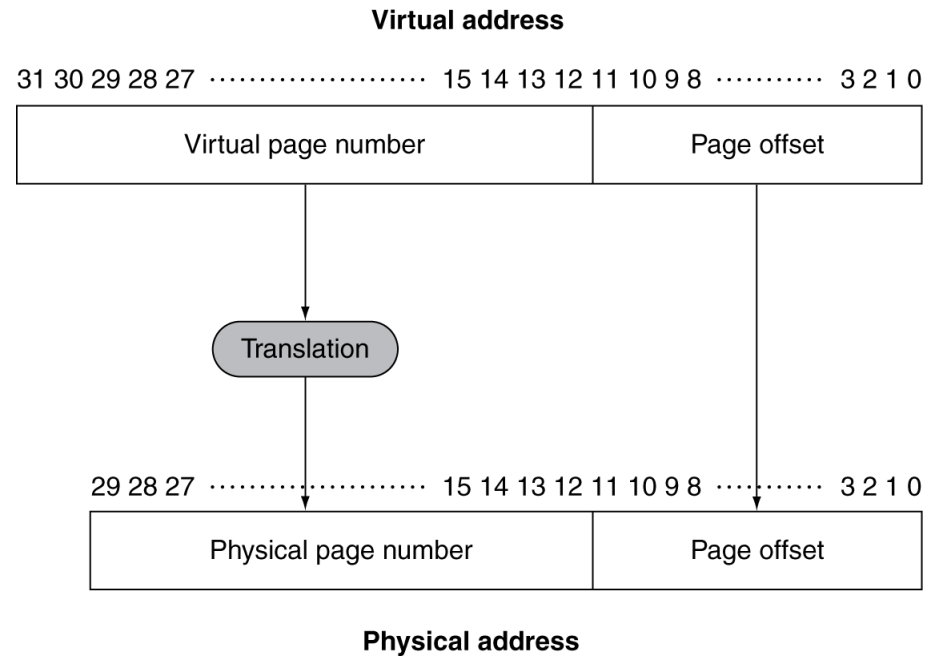
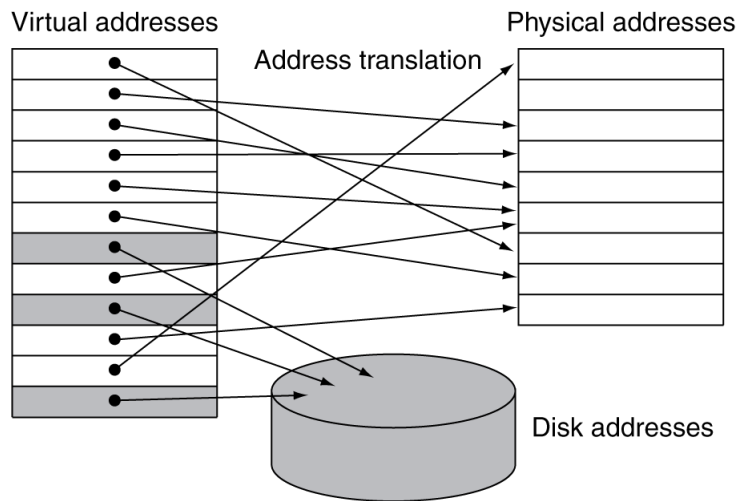
- L-1 cache
 - ◆ Focus on minimal hit time
- L-2 cache
 - ◆ Focus on low miss rate to avoid main memory access
 - ◆ Hit time has less overall impact
- Results
 - ◆ L-1 cache usually smaller than a single cache
 - ◆ L-1 block size smaller than L-2 block size

Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
 - ◆ Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
 - ◆ Each gets a private virtual address space holding its frequently used code and data
 - ◆ Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
 - ◆ VM “block” is called a page
 - ◆ VM “miss” is called a page fault

Address Translation

- Fixed-size pages (e.g., 4K)



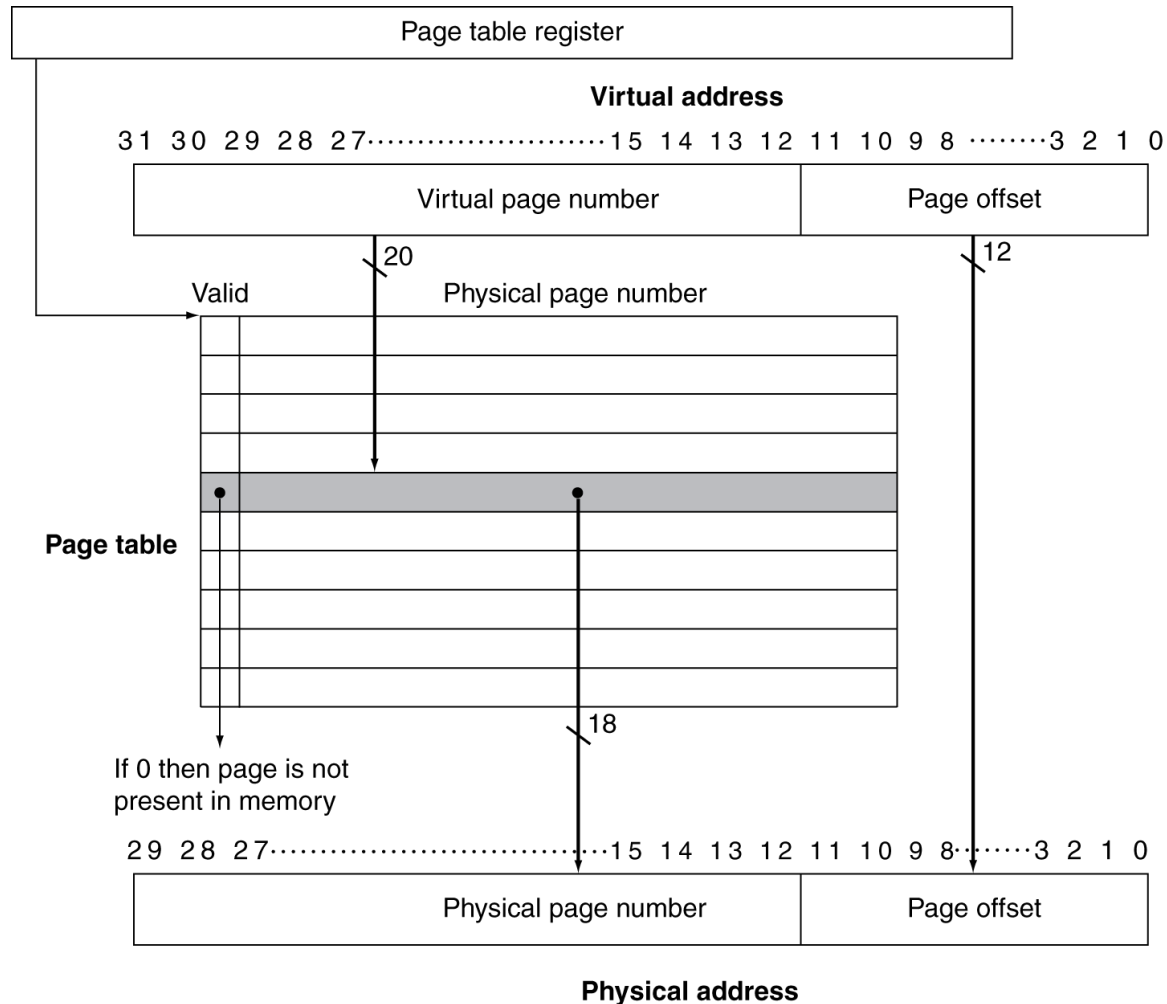
Page Fault Penalty

- On page fault, the page must be fetched from disk
 - ◆ Takes millions of clock cycles
 - ◆ Handled by OS code
- Try to minimize page fault rate
 - ◆ Fully associative placement
 - ◆ Smart replacement algorithms

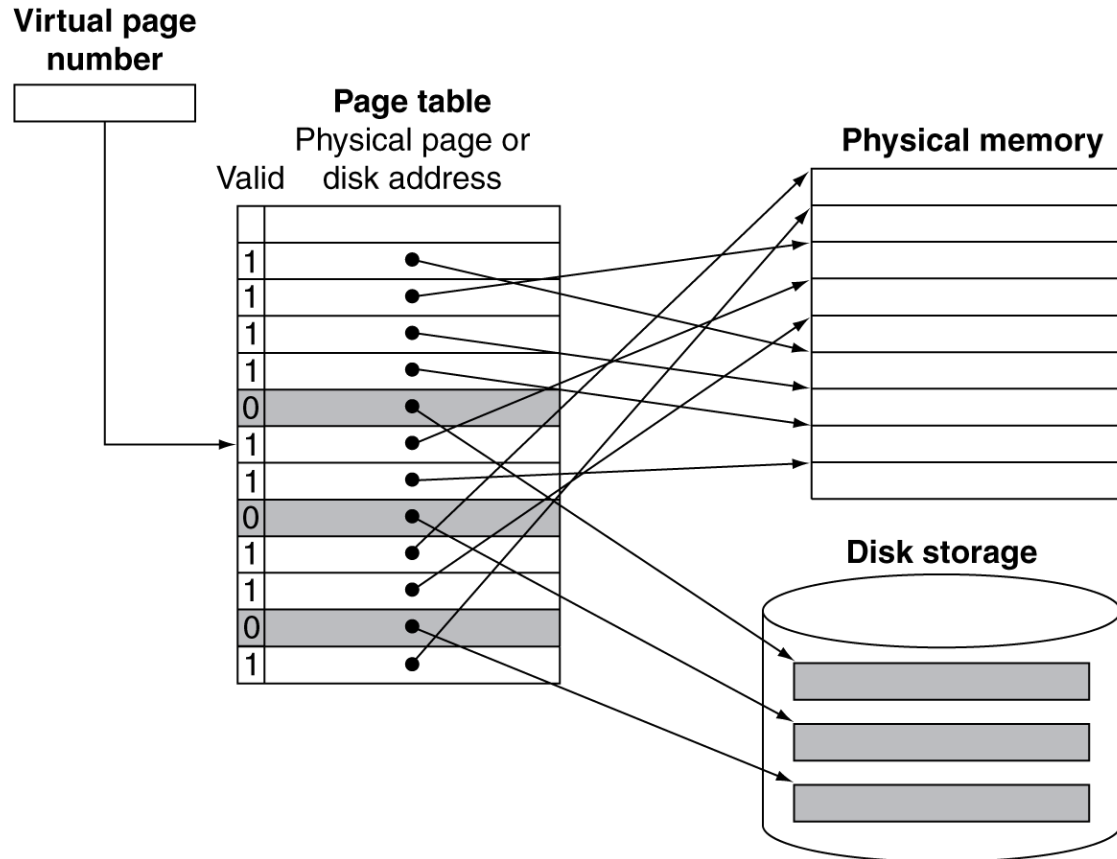
Page Tables

- Where is the placement information? Page Table
 - ◆ Array of page table entries (PTE), indexed by virtual page number
 - ◆ Page table register in CPU points to page table in physical memory
- Each program has its page table. Page table is in memory
- If page is present in memory
 - ◆ PTE stores the physical page number
 - ◆ Plus other status bits (referenced, dirty, ...)
- If page is not present
 - ◆ PTE can refer to location in swap space on disk

Translation Using a Page Table



Mapping Pages to Storage



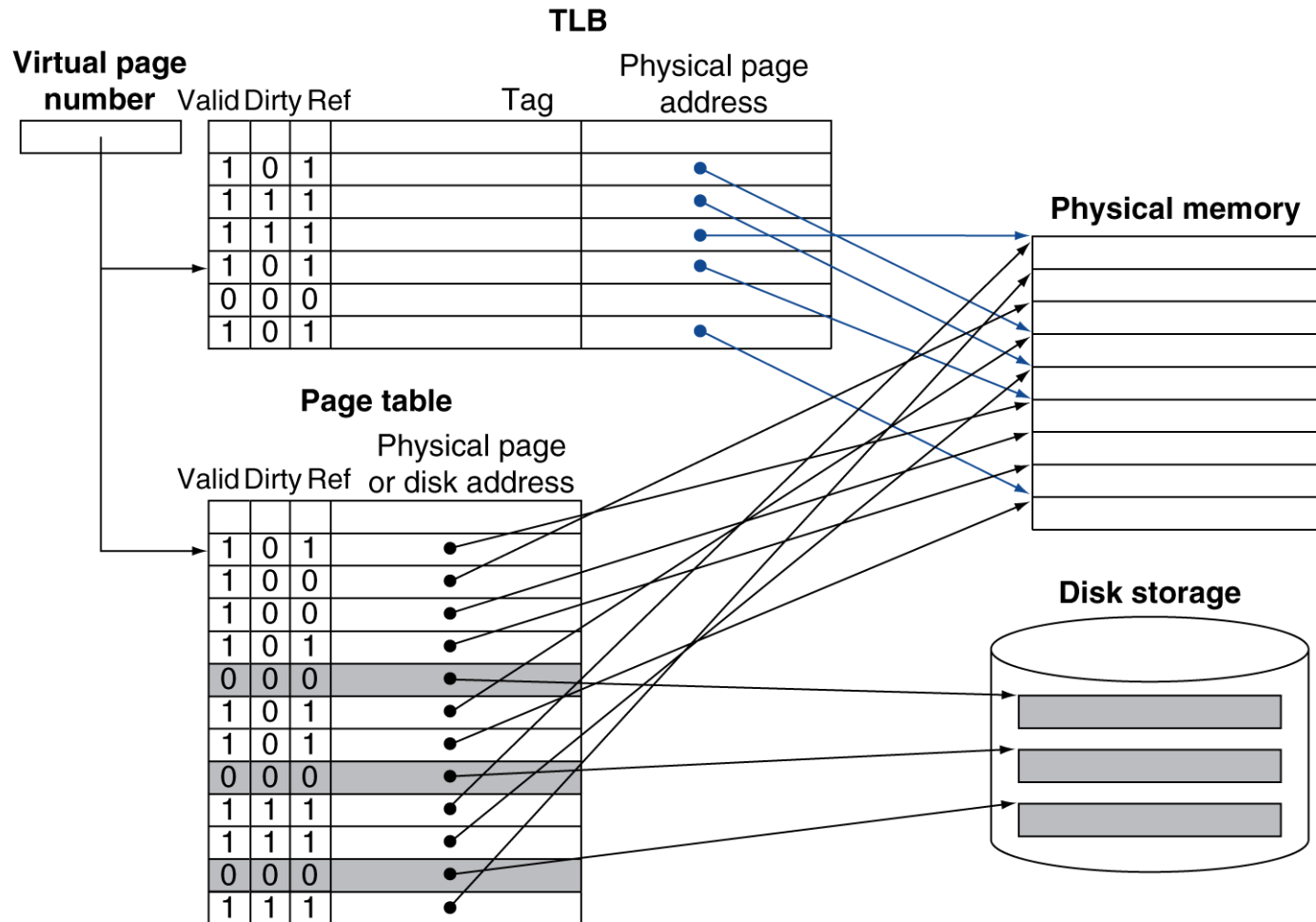
Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
 - ◆ Reference bit (aka use bit) in PTE set to 1 on access to page
 - ◆ Periodically cleared to 0 by OS
 - ◆ A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
 - ◆ Block at once, not individual locations
 - ◆ Use write-back, because write through is impractical
 - ◆ Dirty bit in PTE set when page is written

Fast Translation Using a TLB

- Since page table is in memory, every memory access by a program requires two memory accesses
 - ◆ One to access the page table entry
 - ◆ Then the actual memory access
- Can we move the page table to CPU?
 - ◆ Yes, use a fast cache in CPU to store recently used PTEs, because access to page tables has good locality
 - ◆ Called a Translation Look-aside Buffer (TLB)
 - ◆ Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
 - ◆ Misses could be handled by hardware or software

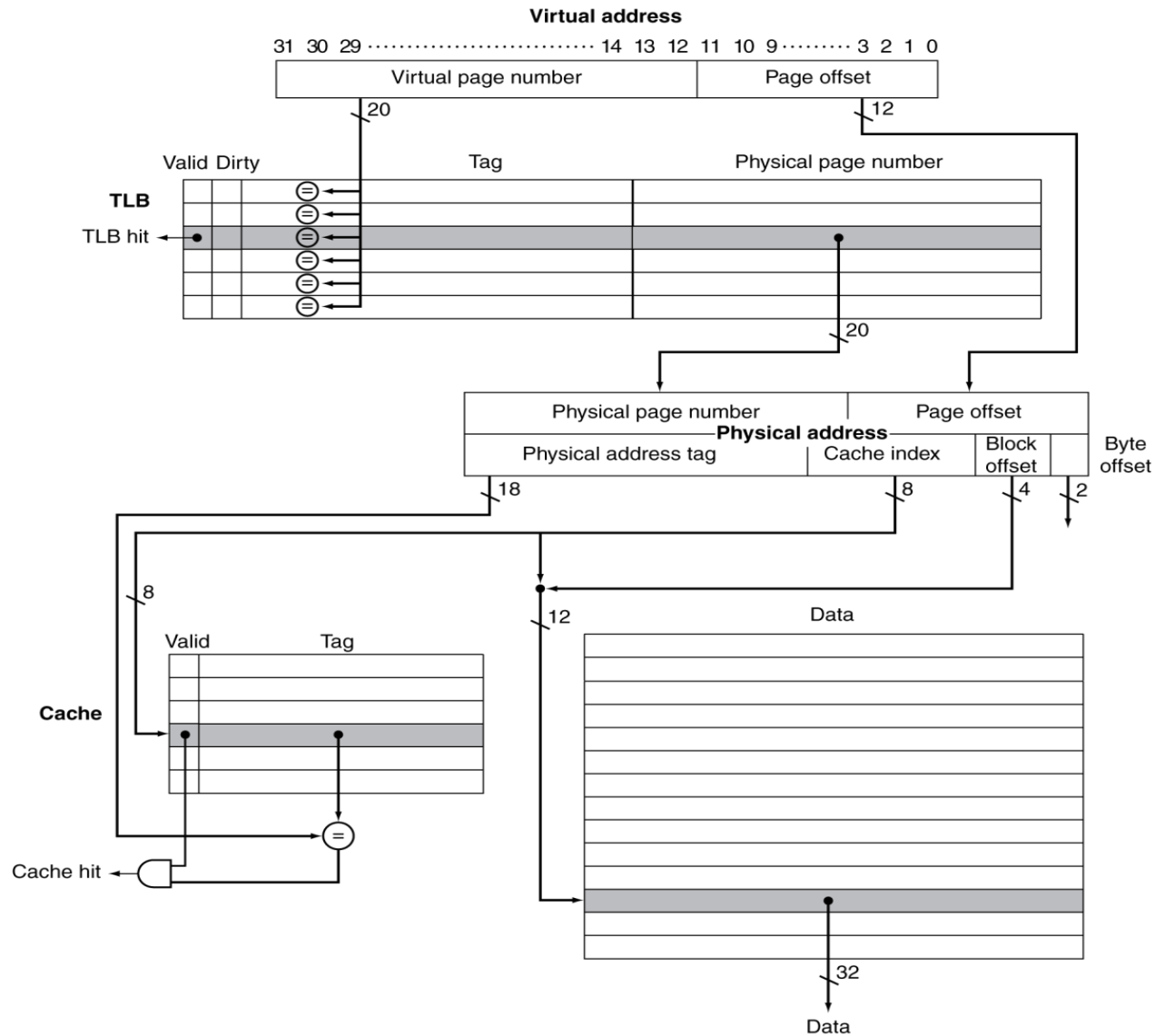
Fast Translation Using a TLB



TLB Misses

- If page is in memory
 - ◆ Load the PTE from memory and retry
 - ◆ Could be handled in hardware
 - Can get complex for more complicated page table structures
 - ◆ Or in software
 - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
 - ◆ OS handles fetching the page and updating the page table
 - ◆ Then restart the faulting instruction

TLB and Cache Interaction



Memory Protection

- Different tasks can share parts of their virtual address spaces
 - ◆ But need to protect against errant access
 - ◆ Requires OS assistance
- Hardware support for OS protection
 - ◆ Privileged supervisor mode (aka kernel mode)
 - ◆ Privileged instructions
 - ◆ Page tables and other state information only accessible in supervisor mode
 - ◆ System call exception (e.g., syscall in MIPS)

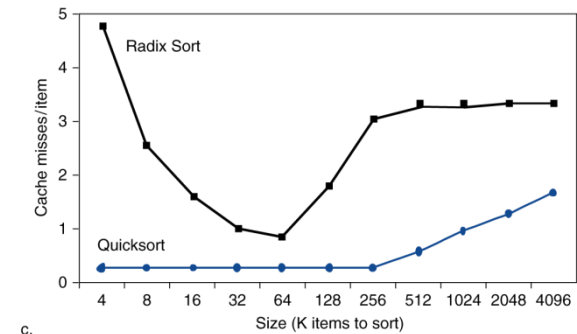
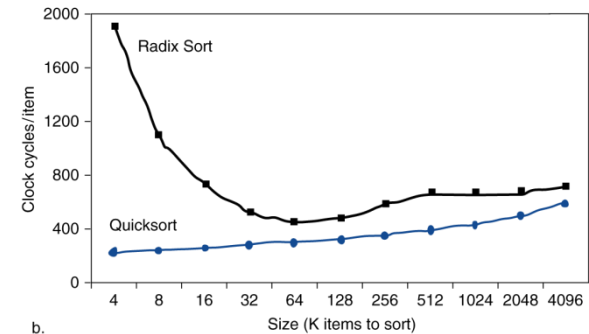
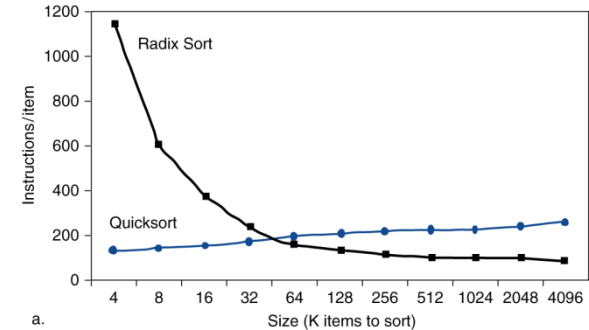
Check Yourself

- Match the definitions between left and right

- | | | |
|---------------|------------------|----------------------------------|
| ➤ L1 cache | ————— | ➤ A cache for a cache |
| ➤ L2 cache | ————— | ➤ A cache for disks |
| ➤ Main memory | ————— | ➤ A cache for a main memory |
| ➤ TLB | ————— | ➤ A cache for page table entries |

Interactions with Software

- Compare two algorithms:
Radix sort & Quicksort
- When size is large,
 - ◆ Radix sort has less instructions
 - ◆ But quicksort has less clock cycles
 - ◆ Because miss rate of radix sort is higher



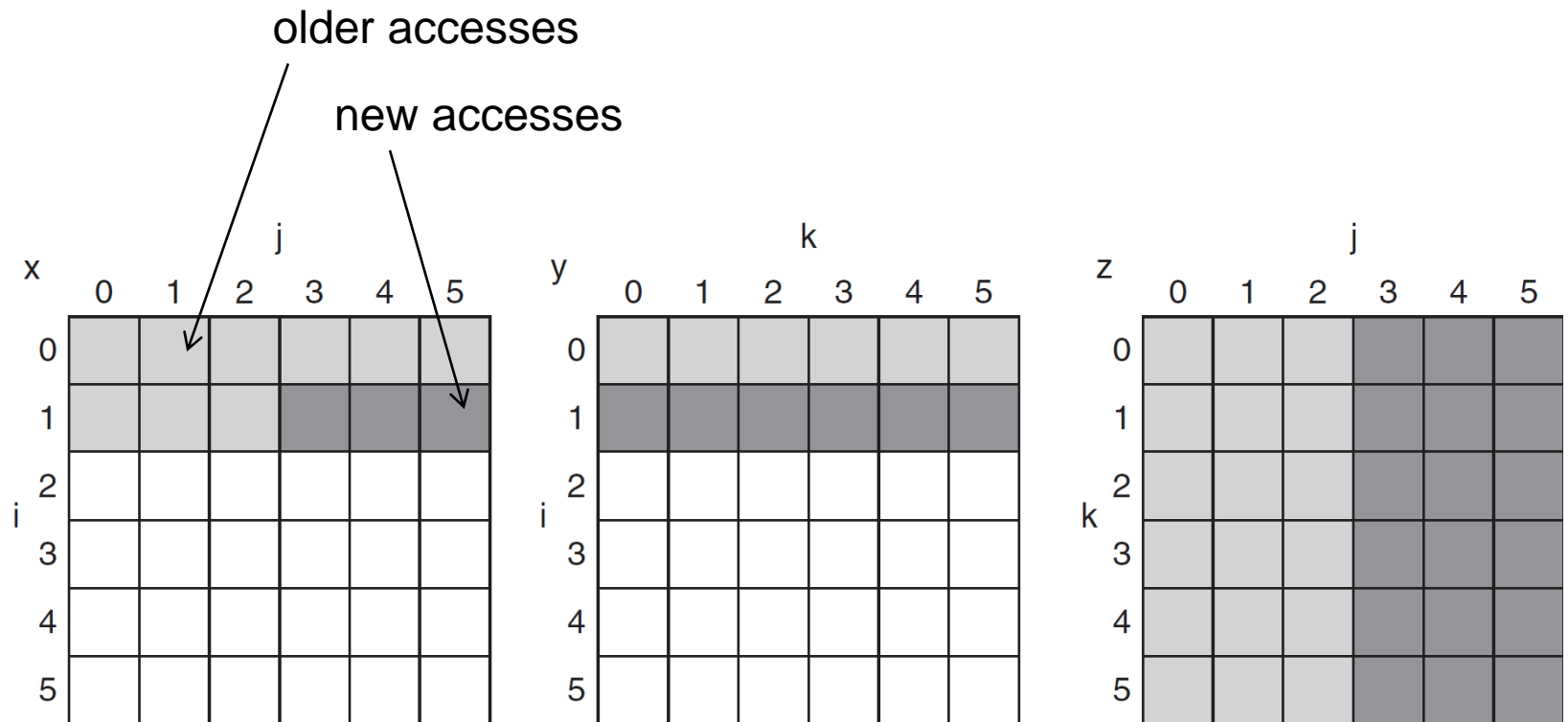
Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

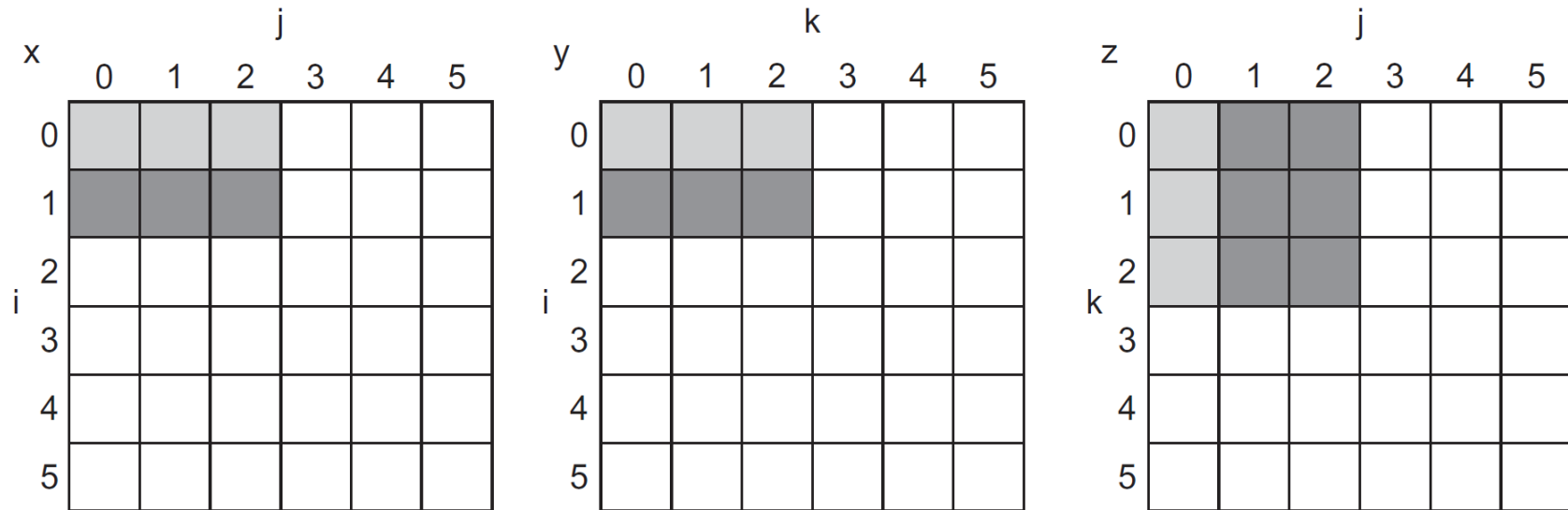
```
for (int j = 0; j < n; ++j)
{
    double cij = C[i+j*n];
    for( int k = 0; k < n; k++ )
        cij += A[i+k*n] * B[k+j*n];
    C[i+j*n] = cij;
}
```

DGEMM Access Pattern

- C, A, and B arrays



Blocked DGEMM Access Pattern



■ 32x32 ■ 160x160 ■ 480x480 ■ 960x960

