Computer Organization and Structure

Bing-Yu Chen
National Taiwan University

Large and Fast: Exploiting Memory Hierarchy

- The Basic of Caches
- Measuring & Improving Cache Performance
- Virtual Memory
- A Common Framework for Memory Hierarchies

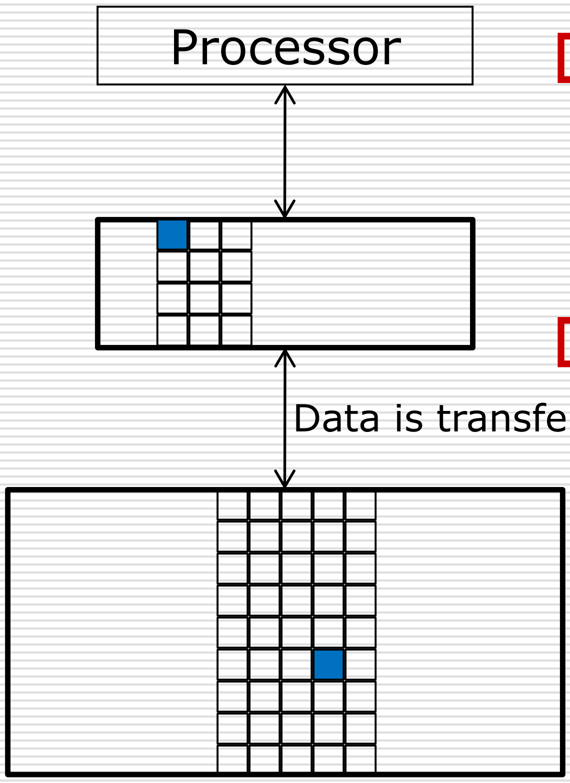
Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - e.g., sequential instruction access, array data

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Memory Hierarchy Levels



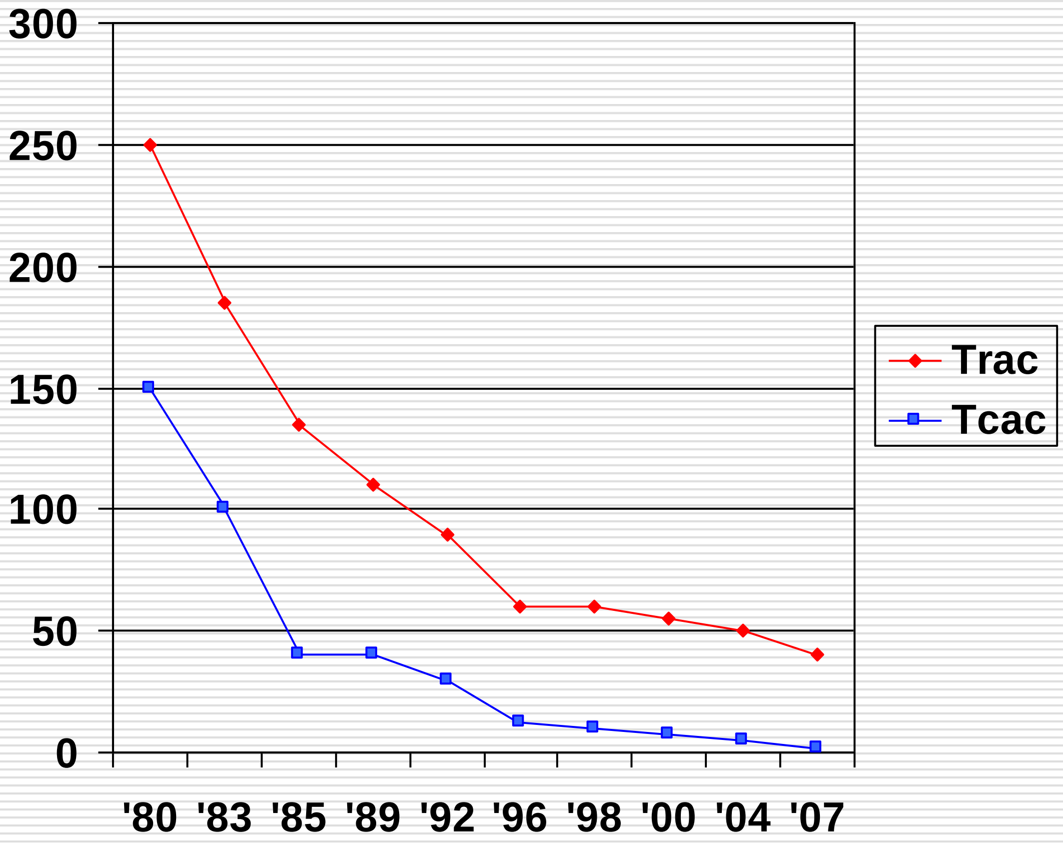
- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses = 1 – hit ratio
  - Then accessed data supplied from upper level

Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, \$20 – \$75 per GB
- Magnetic disk
  - 5ms – 20ms, \$0.20 – \$2 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

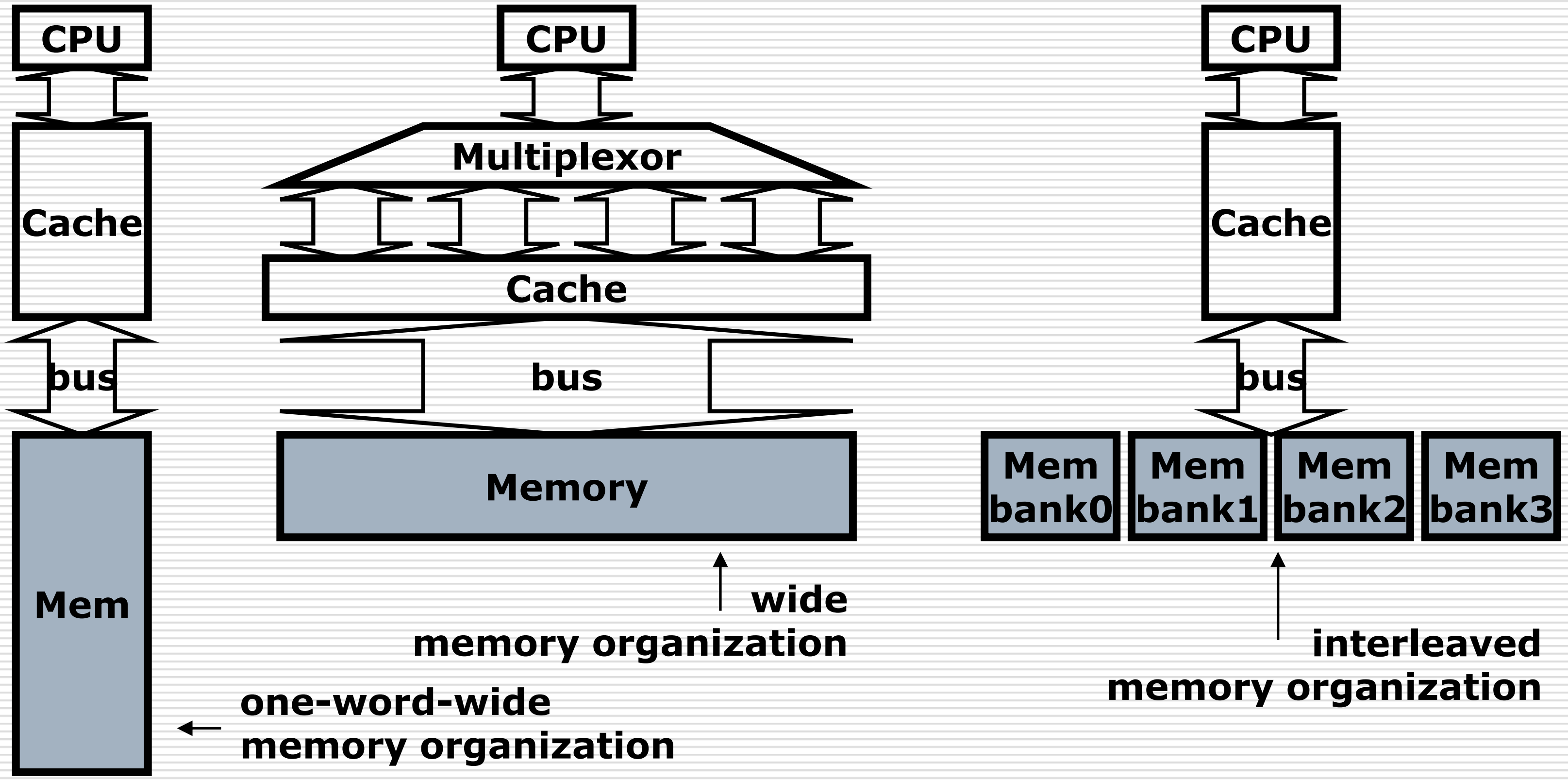
DRAM Generations

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



Increasing Memory Bandwidth

- Make reading multiple words easier by using banks of memory



- It can get a lot more complicated…

Increasing Memory Bandwidth

- Assume a set of hypothetical memory access times:
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

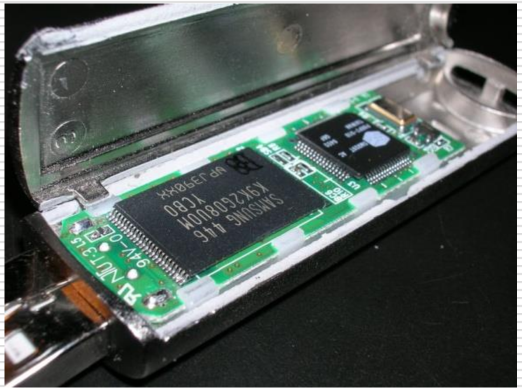
Increasing Memory Bandwidth

- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle

- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4 × 1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Flash Storage

- Non-volatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)

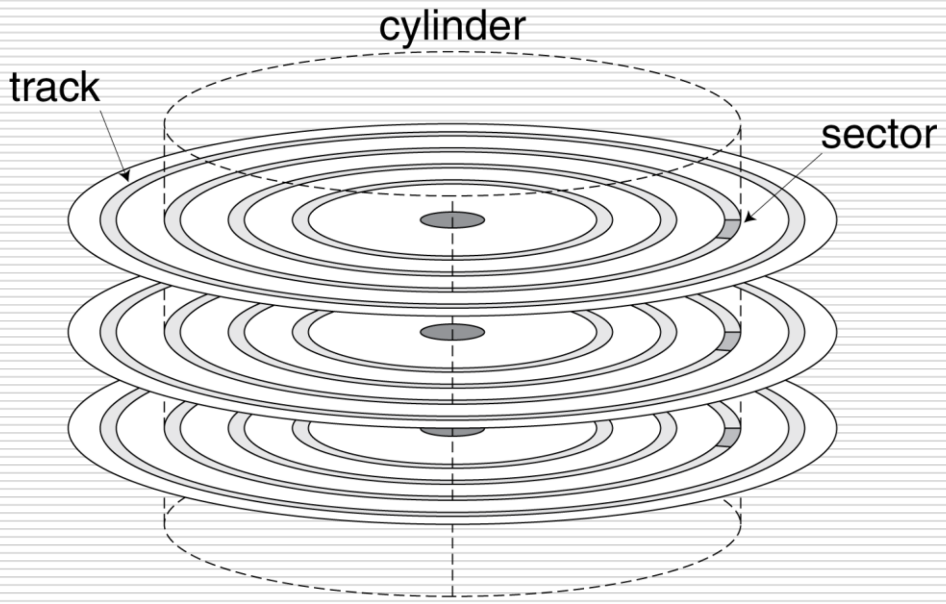


Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems
- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, …
- Flash bits wears out after 1000’s of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear levelling: remap data to less used blocks

Disk Storage

- Nonvolatile, rotating magnetic storage



Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
  - Synchronization fields and gaps
- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead

Disk Access Example

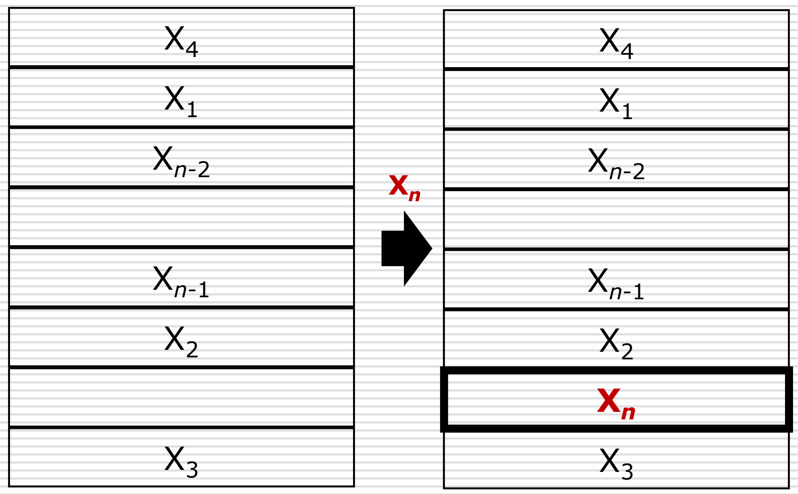
- Given
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average read time
  - 4ms seek time
    + ½ / (15,000/60) = 2ms rotational latency
    + 512 / 100MB/s = 0.005ms transfer time
    + 0.2ms controller delay
    = 6.2ms
- If actual average seek time is 1ms
  - Average read time = 3.2ms

Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times
- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - SCSI, ATA, SATA
- Disk drives include caches
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay

Cache Memory

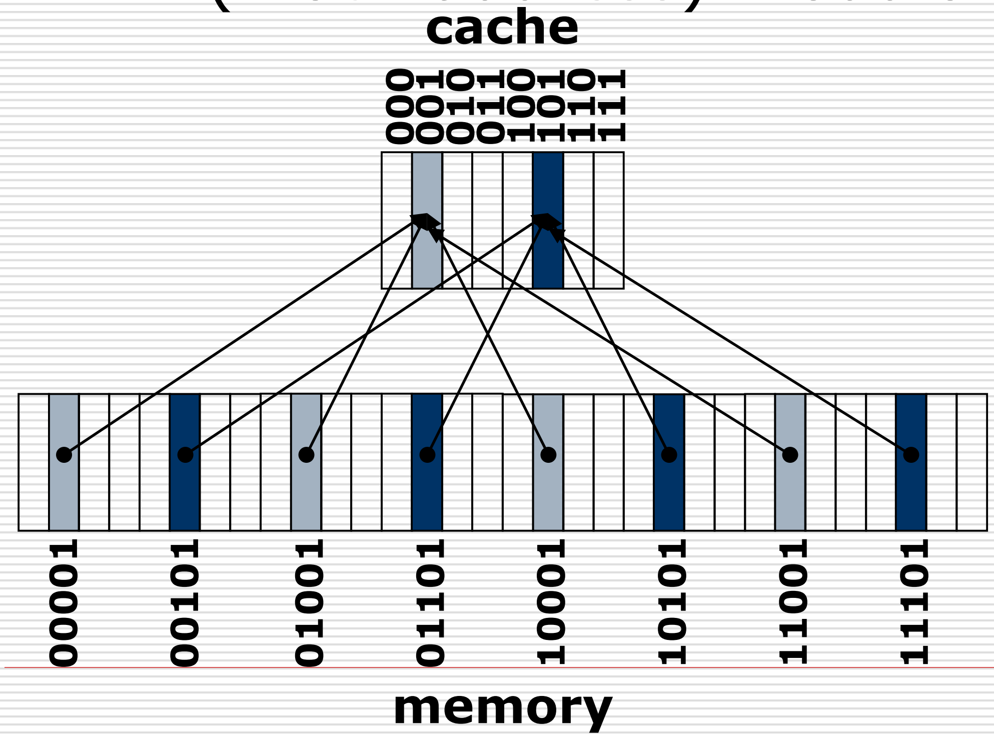
- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$



  - How do we know if the data is present?
  - Where do we look?

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Accessing a Cache (initial)

index	V	tag	data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

8-blocks, 1 word/block, direct mapped

Accessing a Cache (22 miss)

index	V	tag	data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

Accessing a Cache (26 miss)

index	V	tag	data
000	N		
001	N		
010	Y	11_{two}	Memory(11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

Accessing a Cache (22 hit)

index	V	tag	data
000	N		
001	N		
010	Y	11_{two}	Memory(11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

Accessing a Cache (26 hit)

index	V	tag	data
000	N		
001	N		
010	Y	11_{two}	Memory(11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

Accessing a Cache (16 miss)

index	V	tag	data
000	Y	10_{two}	Memory(10000_{two})
001	N		
010	Y	11_{two}	Memory(11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

Accessing a Cache (3 miss)

index	V	tag	data
000	Y	10_{two}	Memory(10000_{two})
001	N		
010	Y	11_{two}	Memory(11010_{two})
011	Y	00_{two}	Memory(00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

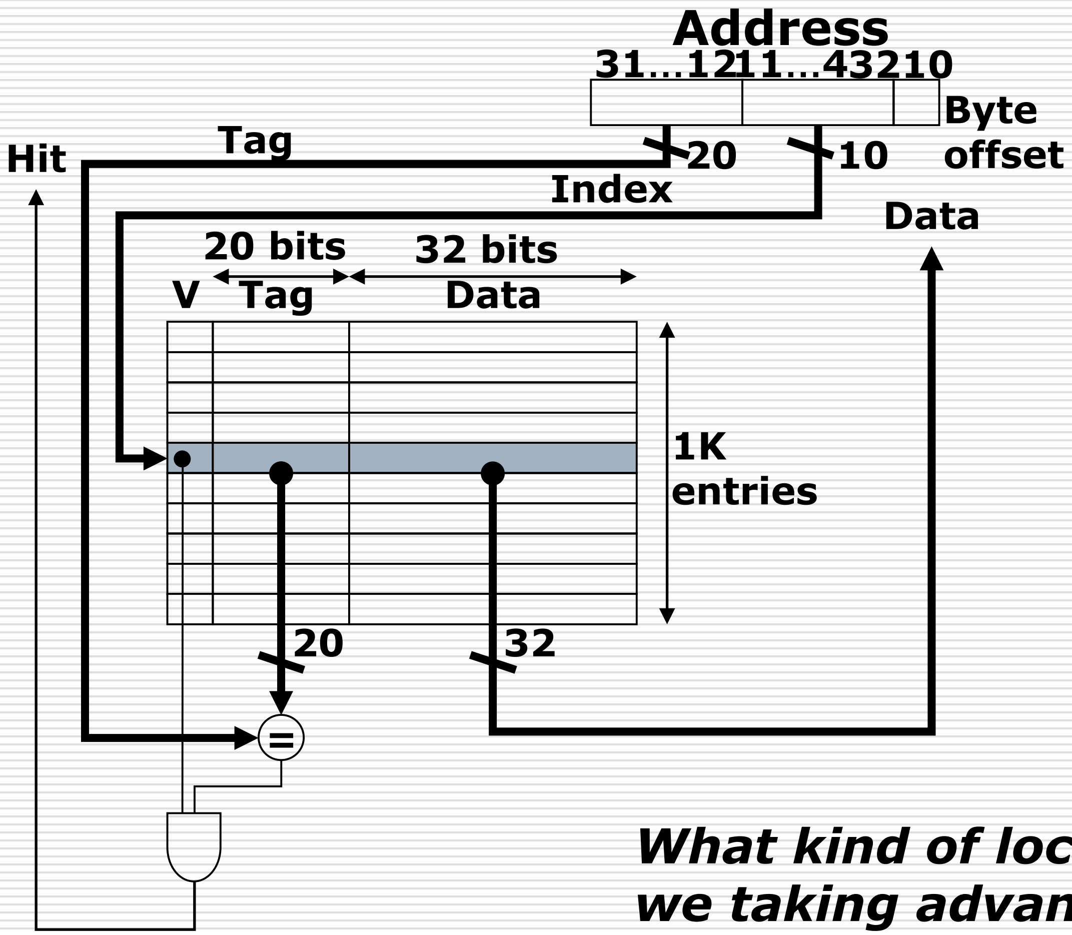
Accessing a Cache (16 hit)

index	V	tag	data
000	Y	10_{two}	Memory(10000_{two})
001	N		
010	Y	11_{two}	Memory(11010_{two})
011	Y	00_{two}	Memory(00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

Accessing a Cache (18 miss)

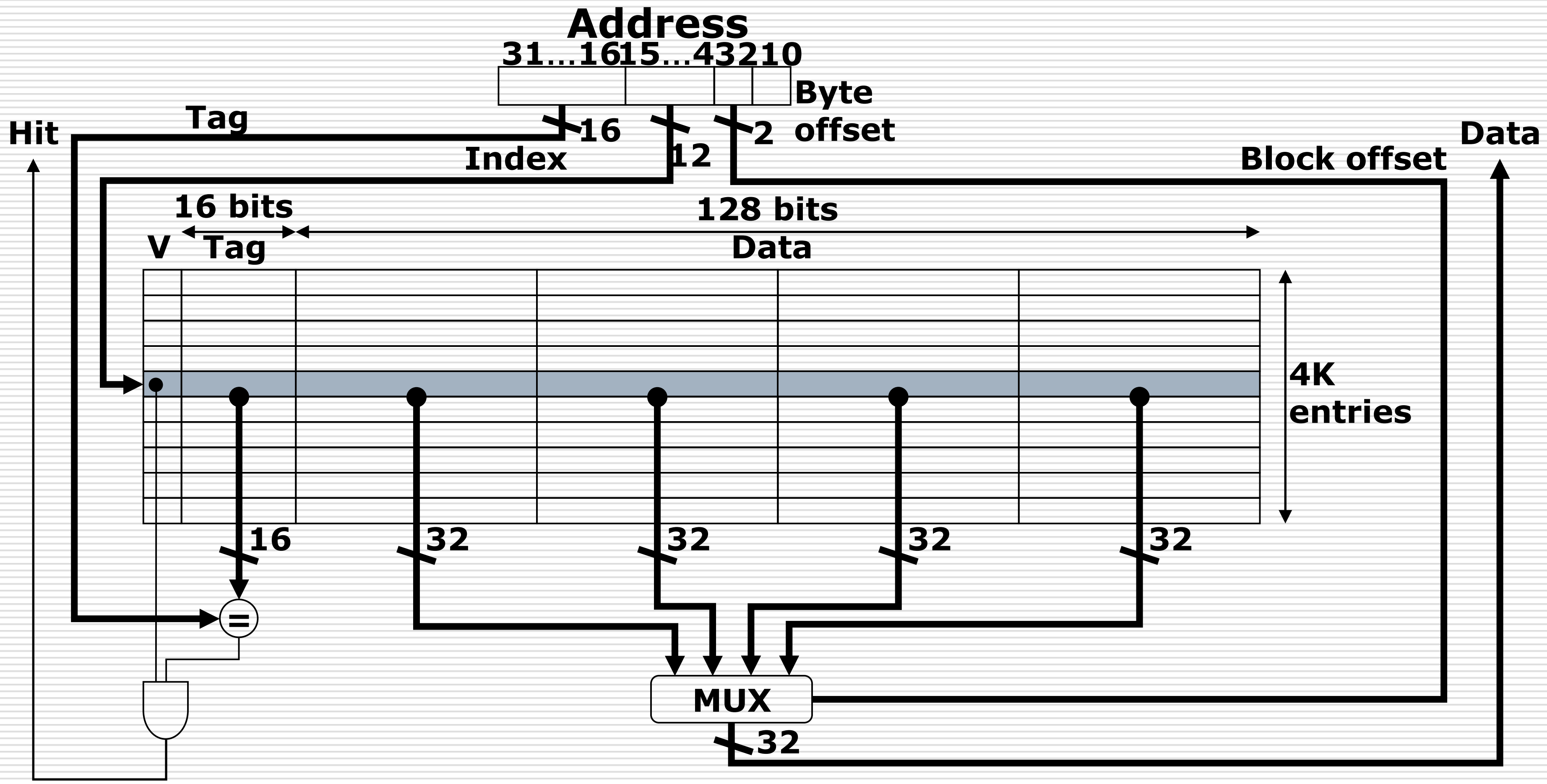
index	V	tag	data
000	Y	10_{two}	Memory(10000_{two})
001	N		
010	Y	10_{two}	Memory(10010_{two})
011	Y	00_{two}	Memory(00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

Direct Mapped Cache



What kind of locality are we taking advantage of?

Spatial Locality



Bits in a Cache

- Assuming the 32-bit byte address, a direct-mapped cache of size 2^n blocks with 2^m-word (2^{m+2}-byte) blocks will require a tag field whose size is $32-(n+m+2)$ bits
  - n bits are used for the index
  - m bits are used for the word within the block
  - 2 bits are used for the byte part of the address

- The total number of bits in a direct-mapped cache is
  - 2^nx(block size + tag size + valid field size)
  - = 2^nx$(2^m$x$32+(32-n-m-2)+1)$

Bits in a Cache

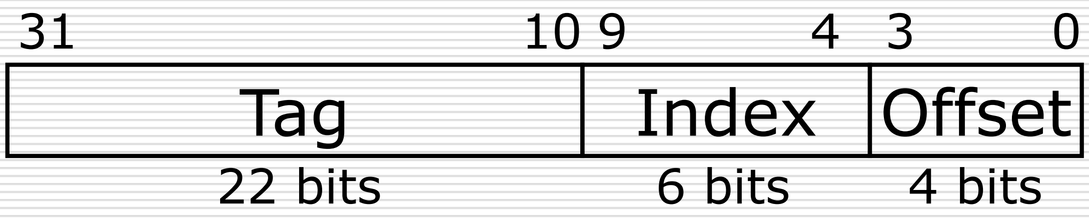
- How many total bits are required for a direct-mapped cache with 16KB of data and 4-word blocks, assuming a 32-bit address?

- We know that 16KB is 4K words, which is 2^{12} words, and, with a block size of 4 words, 2^{10} blocks. Each block has 4x32 bits of data plus a tag, which is 32-10-2-2 bits, plus a valid bit. Thus the total cache size is

  $2^{10}x(128+(32-10-2-2)+1)=$
  $2^{10}x147=147Kbits=18.4KB$

Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$
- Block number = 75 modulo 64 = 11



Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks $\Rightarrow$ fewer of them
    - More competition $\Rightarrow$ increased miss rate
  - Larger blocks $\Rightarrow$ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

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

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = $1 + 0.1 \times 100 = 11$
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don’t fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Measuring Cache Performance

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

$$\text{Memory stall cycles}$$

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

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

Calculating Cache Performance

- assume
  - instruction cache[†] miss rate = 2%
  - data cache[‡] miss rate = 4%
  - miss penalty = 100 cycles for all misses
  - base CPI (ideal cache) = 2
  - frequency of all loads & stores = 36%
- *how much faster a machine would run with a perfect cache that never missed?*

[†]I-Cache
[‡]D-Cache

Calculating Cache Performance

- If Instruction count = I
  - instruction miss cycles = I x 2% x 100
  - data miss cycles = I x 36% x 4% x 100
  - ⇨ memory-stall cycles = 2I+1.44 I = 3.44I
  - ⇨ CPI with memory-stall = 2+3.44 = 5.44
- so $$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times \text{CPI}_{\text{stall}} \times \text{Clock cycle}}{I \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}}$$

$$= \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2} = 2.72$$

Cache Performance with Increased Clock Rate

- *how about doubling the clock rate?*
  - assume the time to handle the cache miss does not change
  - ⇨ miss penalty = 200
  - ⇨ total miss cycles per instruction = (2% x 200) + 36% x (4% x 200) = 6.88
  - ⇨ CPI with memory-stall = 2+6.88 = 8.88
- so $$\frac{\text{Performance with fast clock}}{\text{Performance with slow clock}} = \frac{I \times CPI_{slow} \times \text{Clock cycle}}{I \times CPI_{fast} \times \frac{\text{Clock cycle}}{2}}$$

$$= \frac{5.44}{8.88 \times {}^{1}\!/_{2}} = 1.23$$

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty

- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

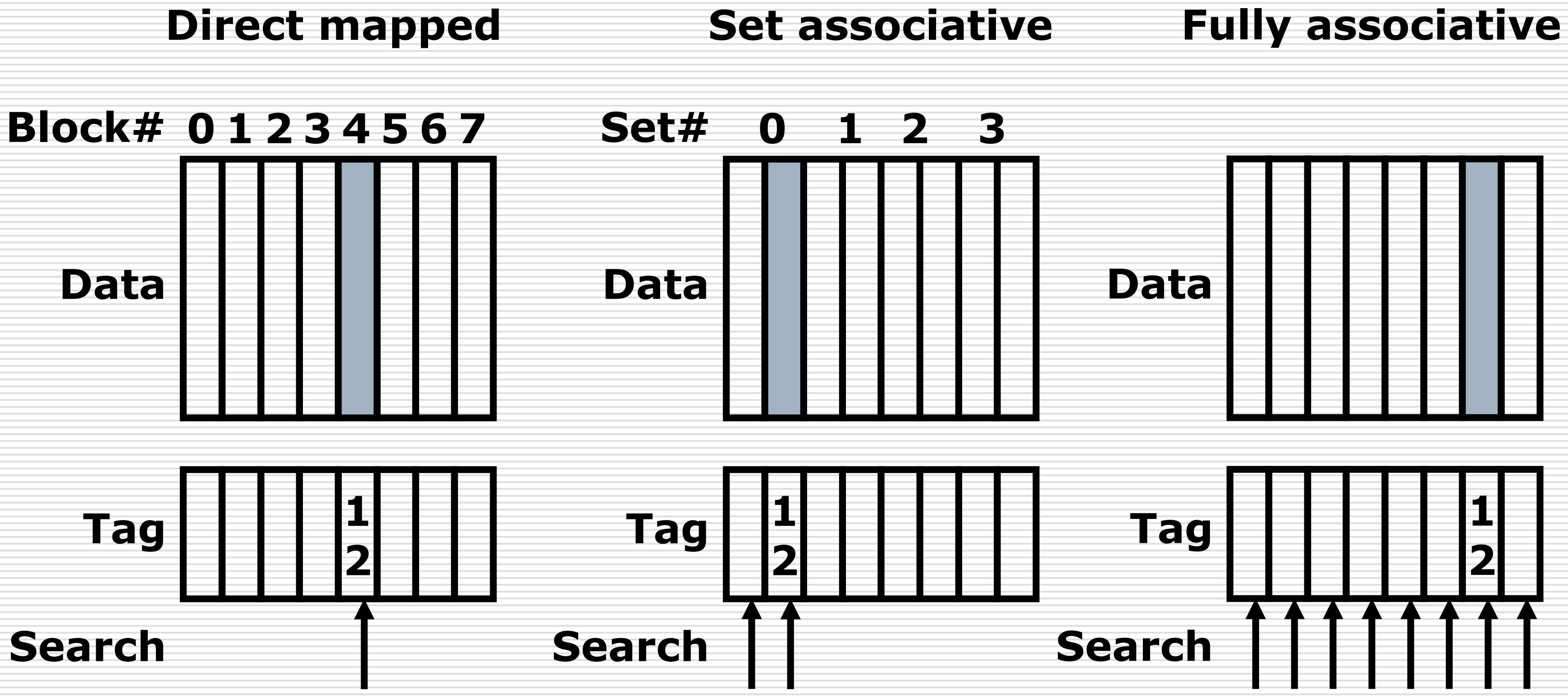
Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- 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

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)

Direct-Mapped, Set-Associative, and Fully Associative Placements



Spectrum of Associativity

One-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

For a cache with 8 entries

Associativity in Caches

- There are 3 small caches, each consisting of 4 1-word blocks with fully associative / 2-way set associative / direct mapped. Find the number of misses for the sequence of block addresses:
  - 0,8,0,6,8

block address	direct-mapped cache block	2-way set associative cache set
0	(0 mod 4)=0	(0 mod 2)=0
6	(6 mod 4)=2	(6 mod 2)=0
8	(8 mod 4)=0	(8 mod 2)=0

Assuming we use the “Least Recently Used” replacement strategy

Direct-Mapped Case

addr.	hit / miss	contents of cache blocks after ref.			
		0	1	2	3
0	miss	Mem[0]			
8	miss	Mem[8]			
0	miss	Mem[0]			
6	miss	Mem[0]		Mem[6]	
8	miss	Mem[8]		Mem[6]	

Two-Way Set-Associative Case

add.	hit / miss	contents of cache blocks after ref.			
		set 0	set 0	set 1	set 1
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	Mem[0]	Mem[8]		
6	miss	Mem[0]	Mem[6]		
8	miss	Mem[8]	Mem[6]		

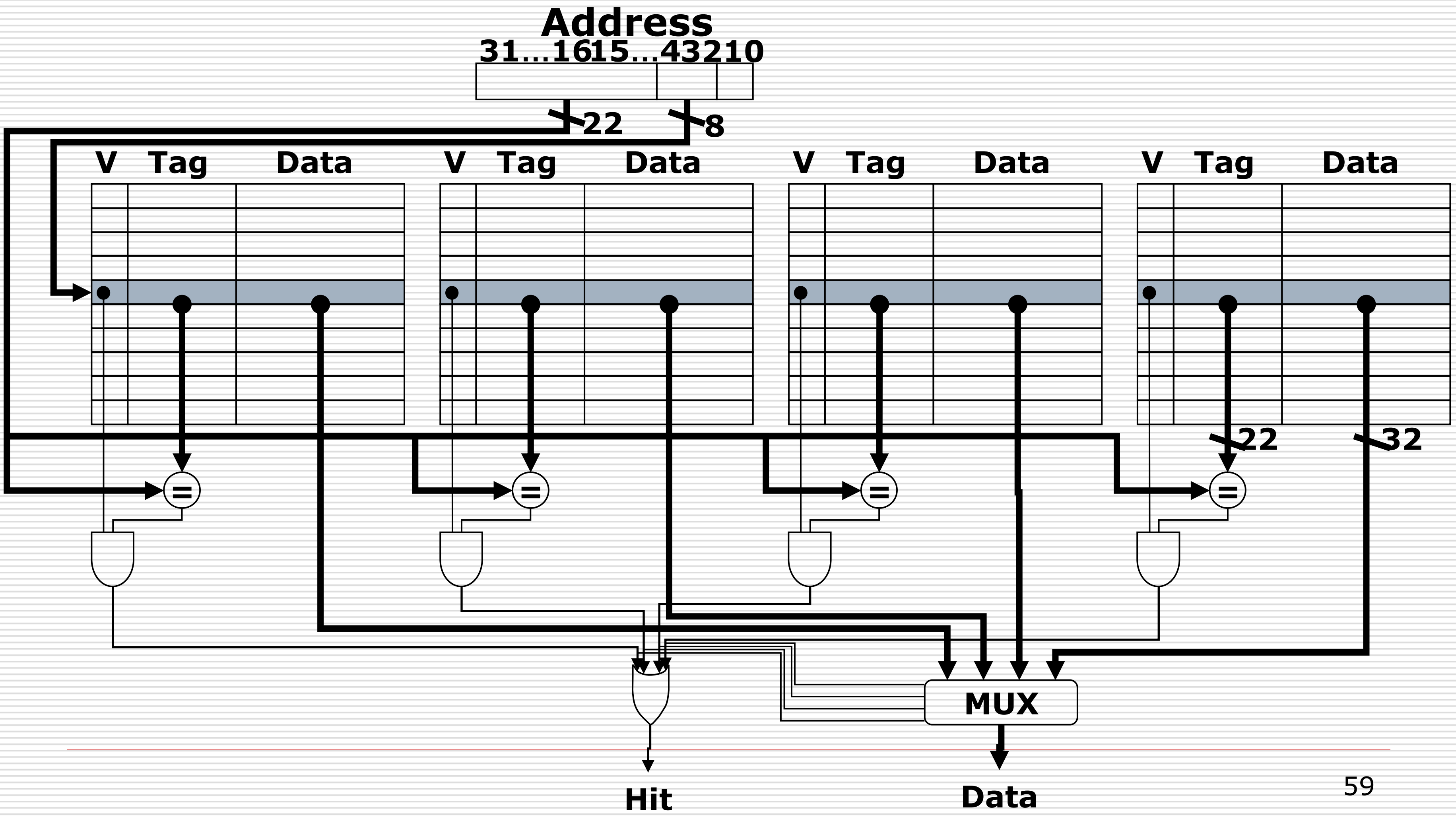
Fully Associative Case

add.	hit / miss	contents of cache blocks after ref.			
		block 0	block 1	block 2	block 3
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]	

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%

Set Associative Cache Organization



Replacement Policy

- Direct mapped: no choice
- 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

Size of Tags vs. Set Associativity

- Increasing associativity requires more comparators, as well as more tag bits per cache block.
- Assuming a cache of 4K blocks, a 4-word block size, and a 32-bit address, find the total number of sets and the total number of tag bits for caches that are direct mapped, 2-way and 4-way set associative, and fully associative.

Size of Tags vs. Set Associativity

- direct mapped
  - 4K sets need $\log_2(4K)$=12 bits for index
  - (28-12) x 1 x 4K = 64K tag bits
- 2-way set associative
  - 2K sets
  - (28-11) x 2 x 2K = 68K tag bits
- 4-way set associative
  - 1K sets
  - (28-10) x 4 x 1K = 72K tag bits
- fully associative
  - 1 set with 4K blocks
  - 28 x 4K x 1 = 112K tag bits

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary 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 just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

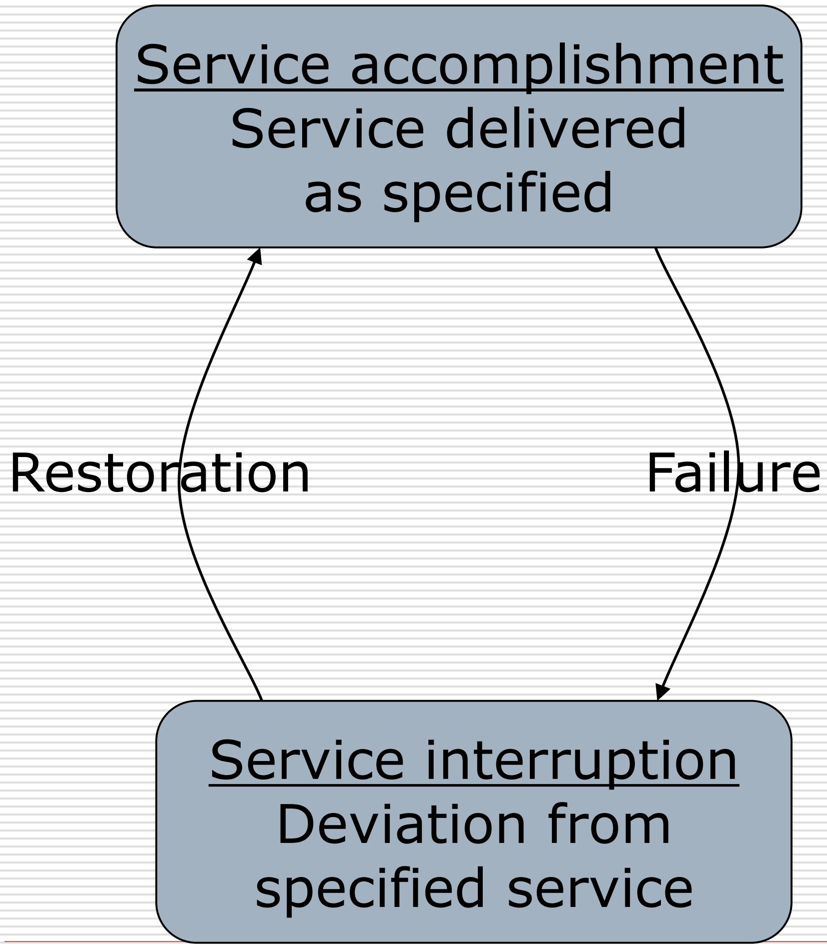
Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

Multilevel Cache Considerations

- Primary 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

Dependability


Service accomplishment
Service delivered as specified
Restoration
Failure
Service interruption
Deviation from specified service

- Fault: failure of a component
  - May or may not lead to system failure

Dependability Measures

- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair

Virtual Machines

- Host computer emulates guest operating system and machine resources
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources
- Virtualization has some performance impact
  - Feasible with modern high-performance computers
- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Microsoft Virtual PC

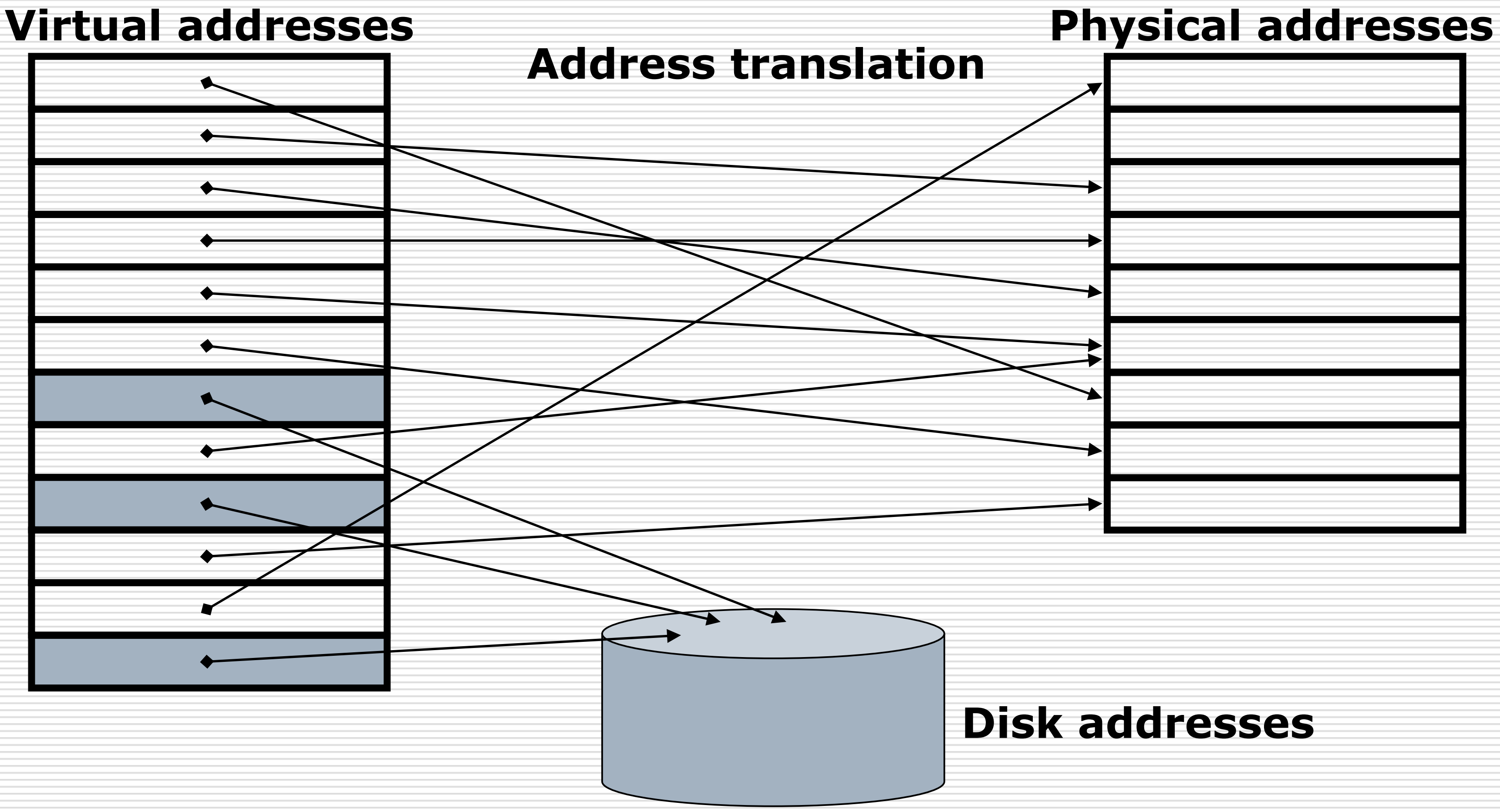
Virtual Machine Monitor

- Maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest

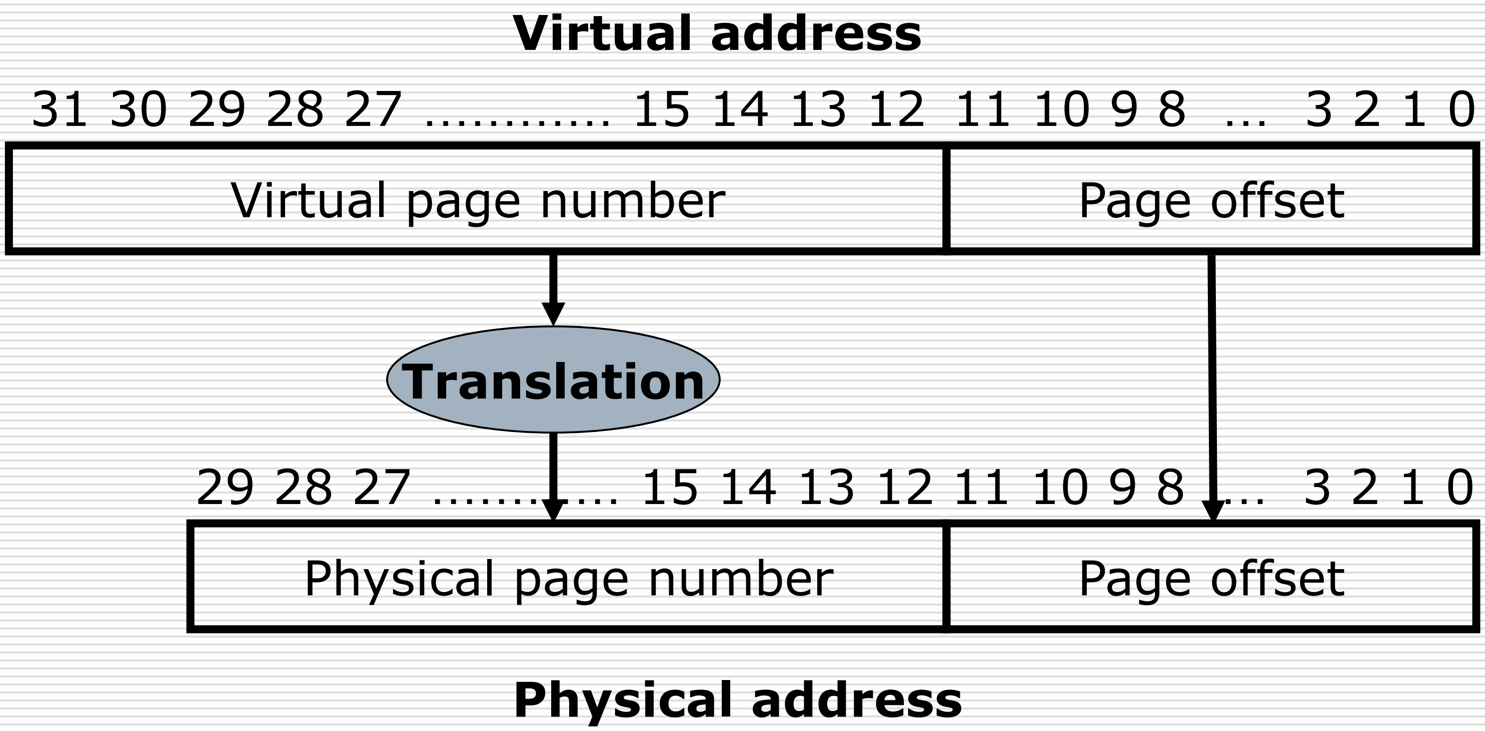
Virtual Memory

- 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 translation “miss” is called a page fault

Virtual Memory



Mapping from a Virtual to a Physical Address



Page Fault Penalty

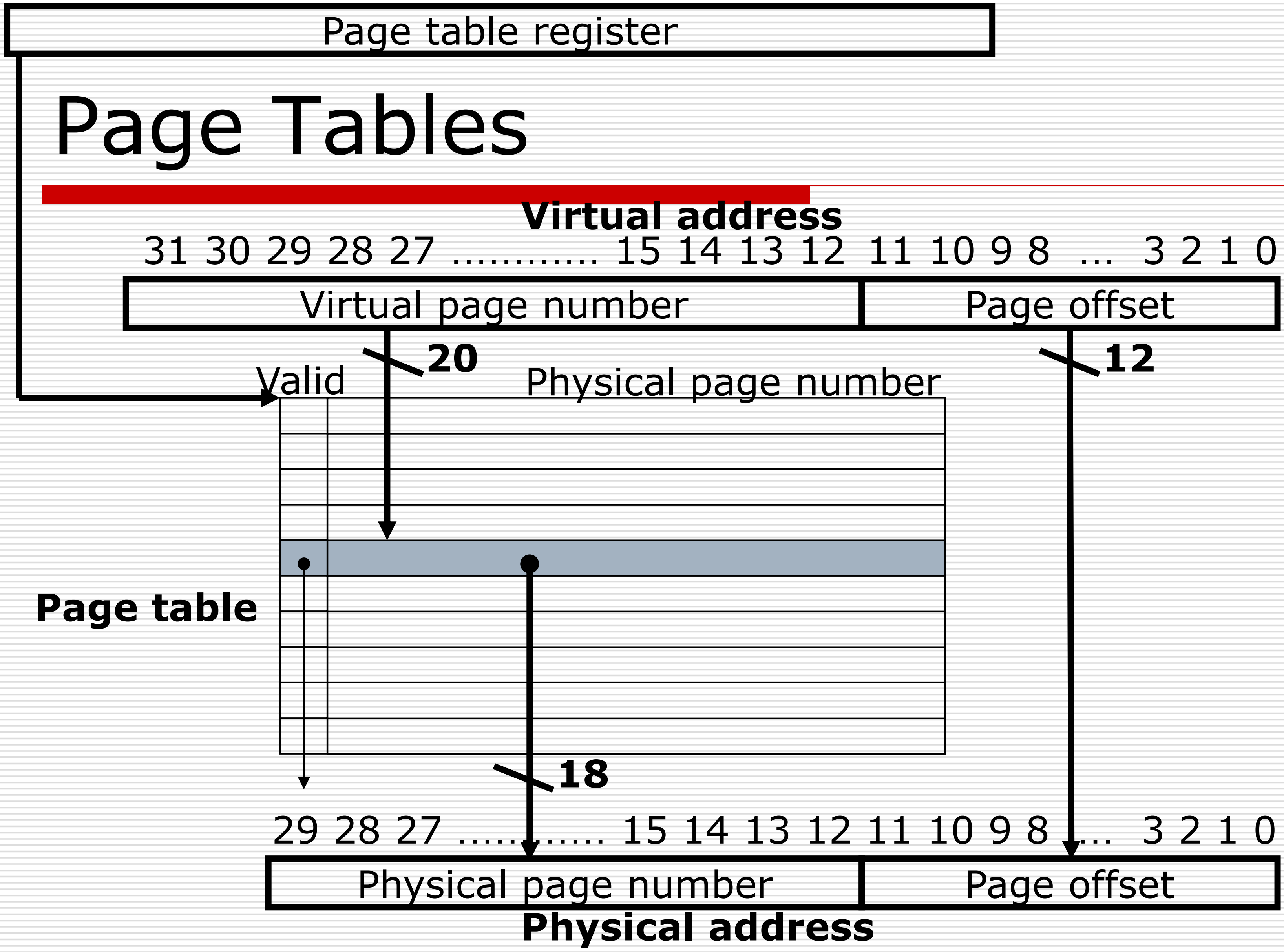
- On page fault, the page[†] must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms

[†]Page: Virtual Memory Block

Page Tables

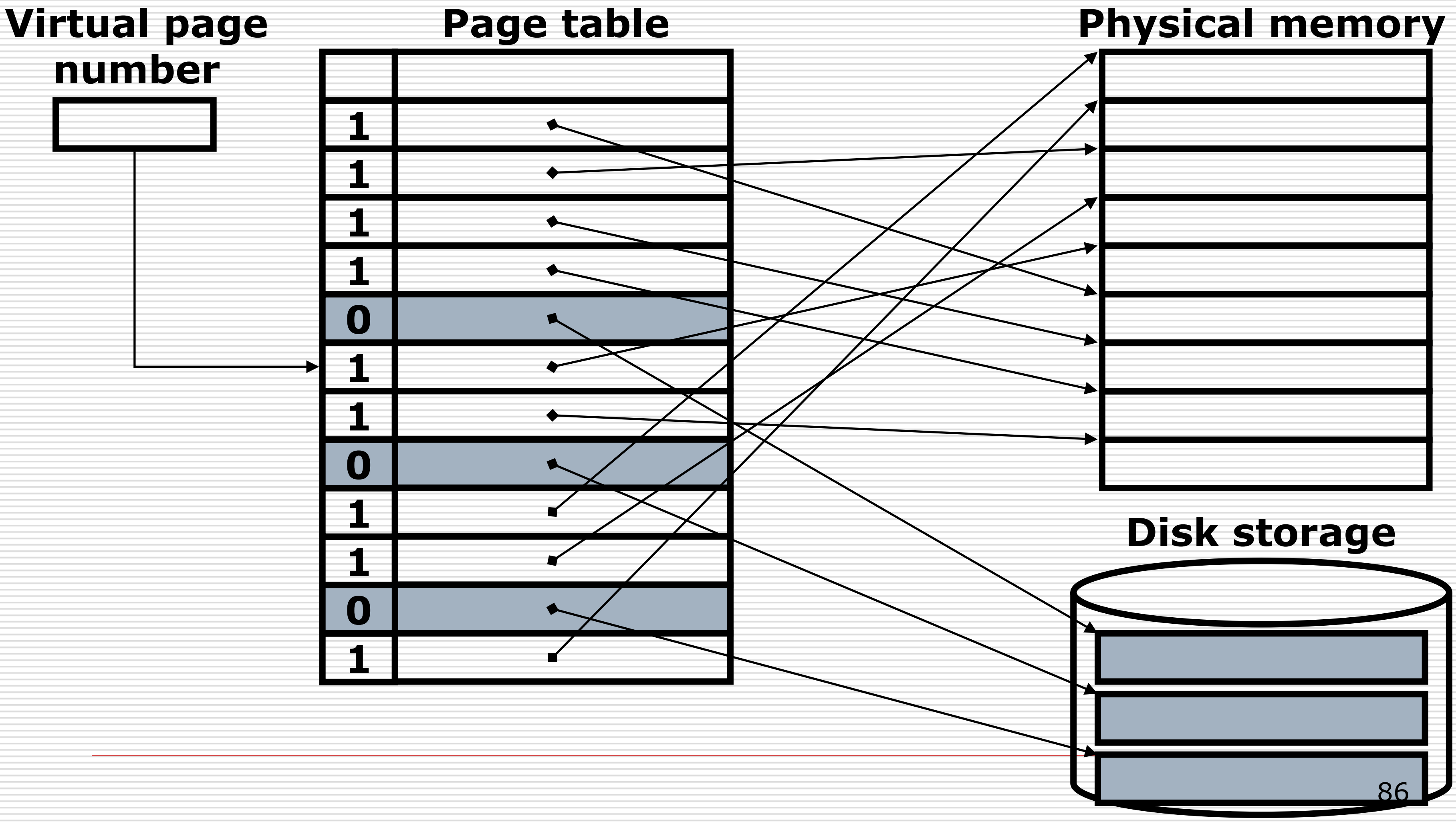
- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to location in swap space on disk

Page Tables



Page table register
Virtual address
31 30 29 28 27 ………… 15 14 13 12 11 10 9 8 … 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Physical page number
Page table
18
29 28 27 ………… 15 14 13 12 11 10 9 8 … 3 2 1 0
Physical page number
Page offset
Physical address

Mapping Pages to Storage



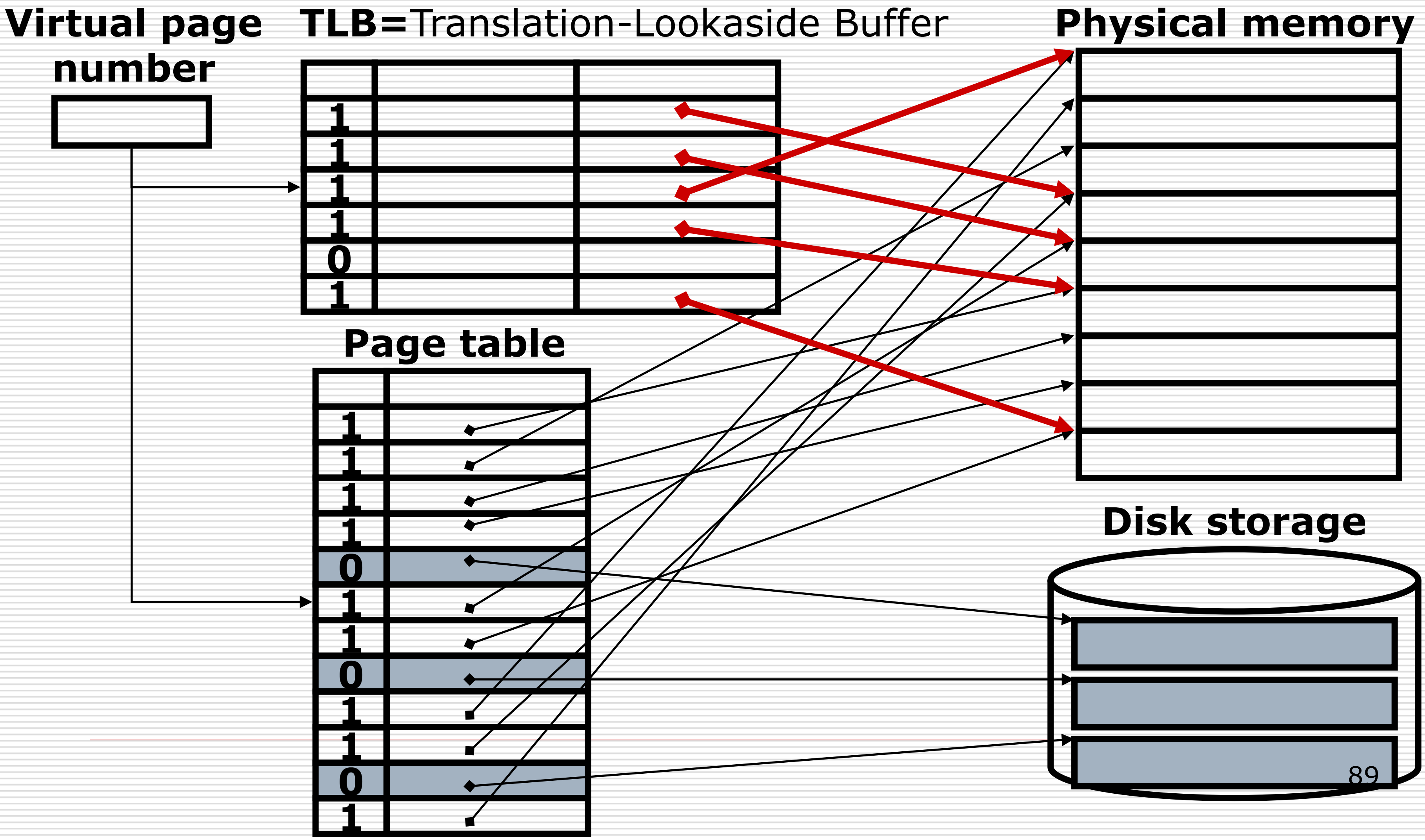
Replacement and Writes

- 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
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - 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

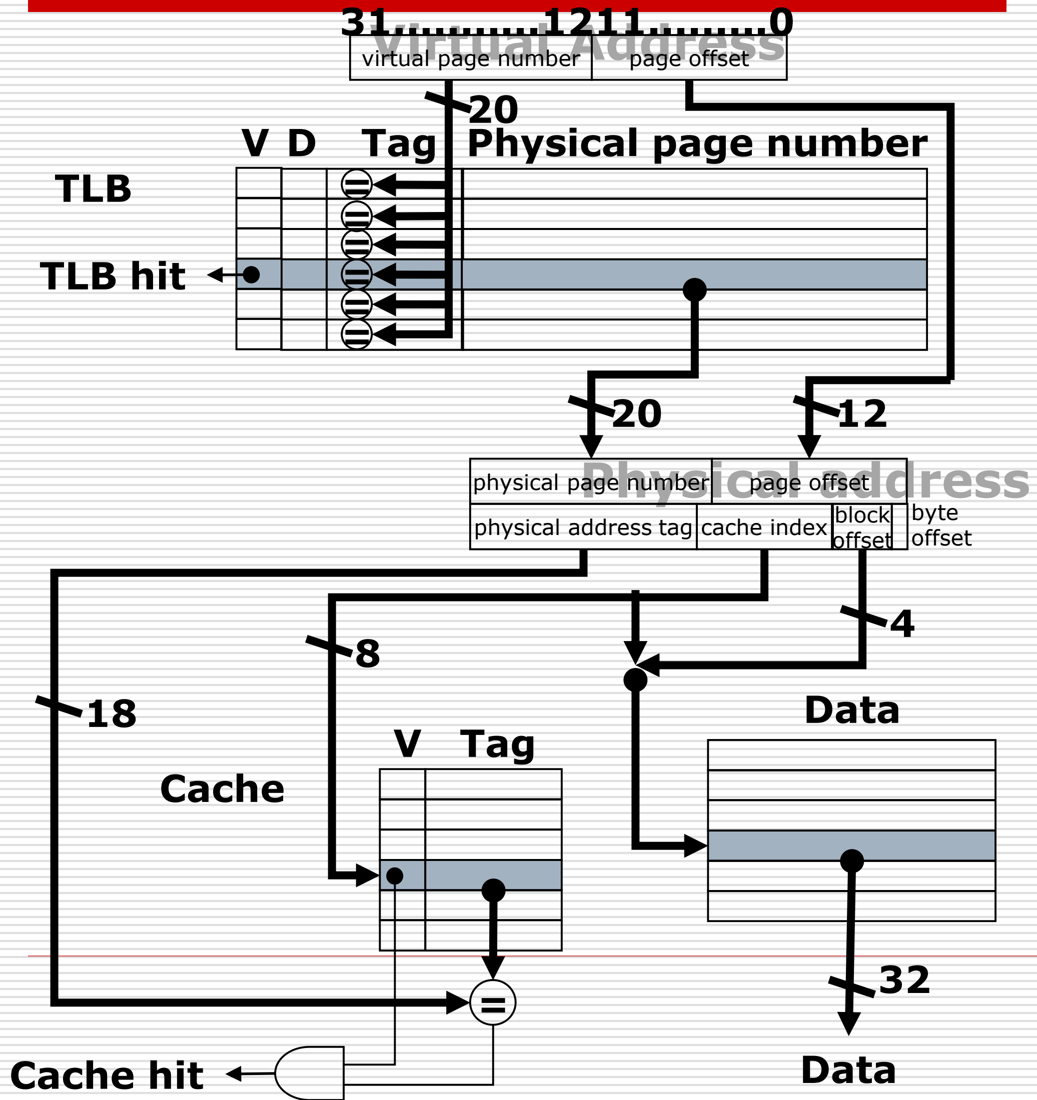
TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not preset
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

Page Fault Handler

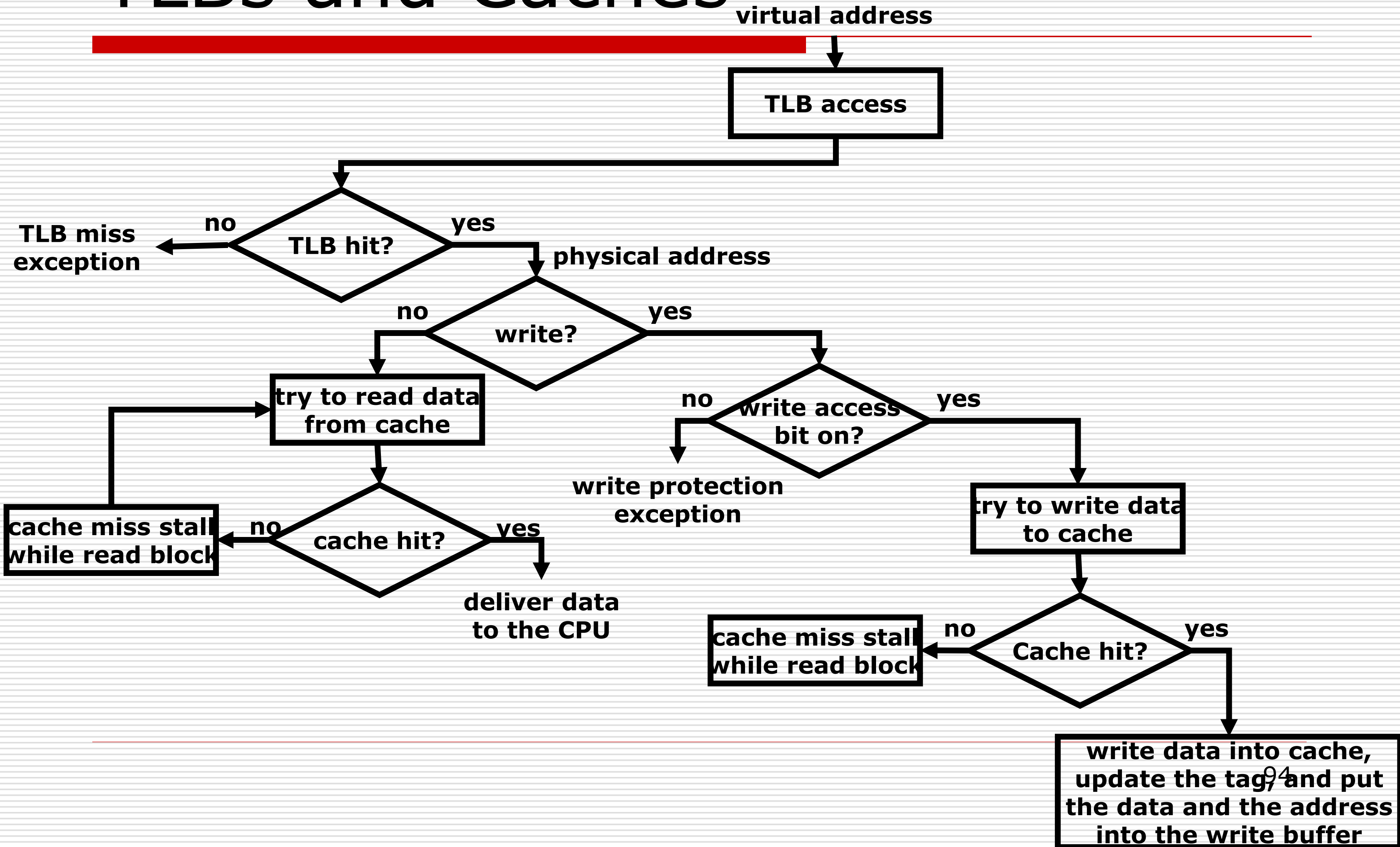
- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

TLB and Cache Interaction



- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address

TLBs and Caches


virtual address
TLB access
TLB hit?
no
TLB miss exception
yes
physical address
write?
no
yes
try to read data from cache
write access bit on?
no
write protection exception
yes
try to write data to cache
cache hit?
no
cache miss stall while read block
yes
deliver data to the CPU
Cache hit?
no
cache miss stall while read block
yes
write data into cache, update the tag, and put the data and the address into the write buffer

Possible Combinations of Events in TLB / Page Table / Cache

TLB	Page table	Cache	possible?
hit	hit	miss	possible, although the page table is never really checked if TLB hits
miss	hit	hit	TLB misses, but entry found in page table; after retry, data is found in cache
miss	hit	miss	TLB misses, but entry found in page table; after retry, data misses in cache
miss	miss	miss	TLB misses and is followed by a page fault; after retry, data must miss in cache
hit	miss	miss	impossible: cannot have a translation in TLB if page is not present in memory
hit	miss	hit	impossible: cannot have a translation in TLB if page is not present in memory
miss	miss	hit	impossible: data cannot be allowed in cache if the page is not in memory

The Memory Hierarchy

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Where can a Block be Placed?

scheme name	number of sets	blocks per set
direct mapped	number of blocks in cache	1
set associative	$\frac{\text{number of blocks in cache}}{\text{associativity}}$	associativity
fully associative	1	number of blocks in cache

- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

How is a Block Found?

associativity	location method	comparisons required
direct mapped	index	1
set associative	index the set, search among elements	degree of associativity
fully associative	search all cache entries	size of the cache
	separate lookup table	0

- Hardware caches
  - Reduce comparisons to reduce cost
- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

Which Block should be Replaced on a Cache Miss?

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

What Happens on a Write?

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

Sources of Misses

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.