



# Large and Fast: Exploiting Memory Hierarchy

Instructor: Huzefa Rangwala, PhD

# 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

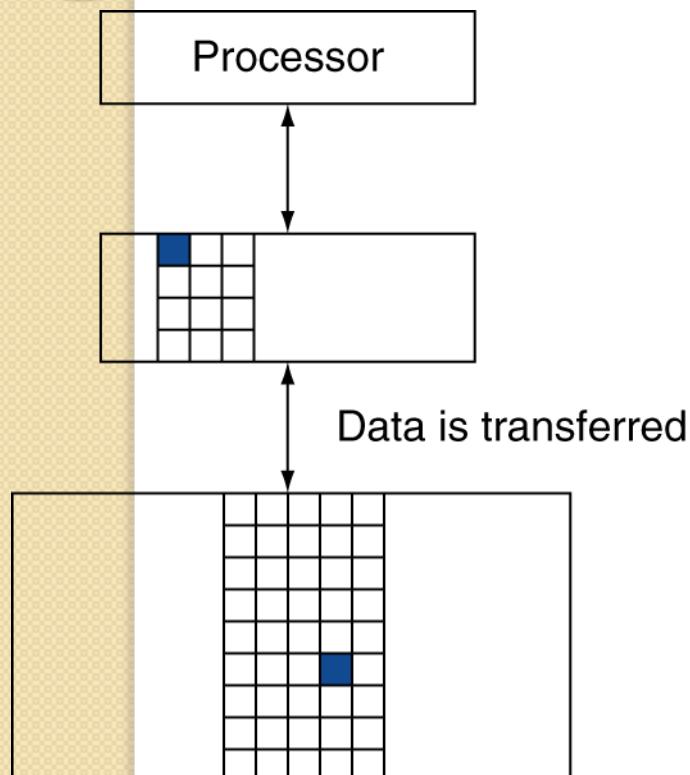
# Principle of Locality

- 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

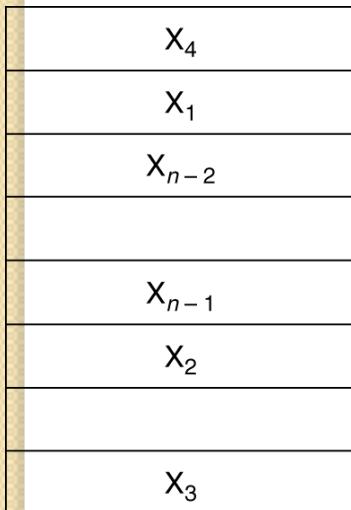
# Memory Hierarchy Levels



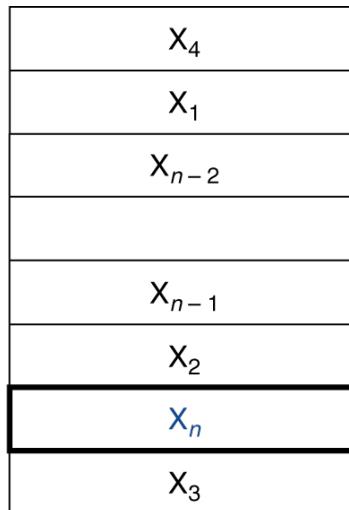
- **Block (aka line):** unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: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

# Cache Memory

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



a. Before the reference to  $X_n$

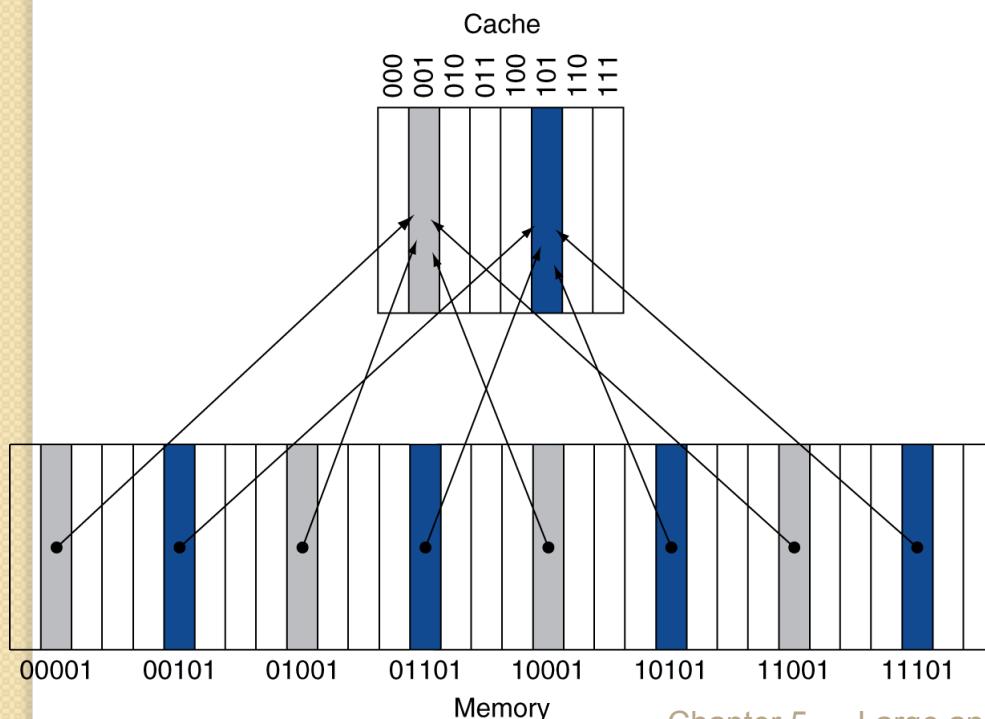


b. After the reference to  $X_n$

- 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

# Cache Example

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

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

# Cache Example

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

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

# Cache Example

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

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

# Cache Example

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

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

# Cache Example

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

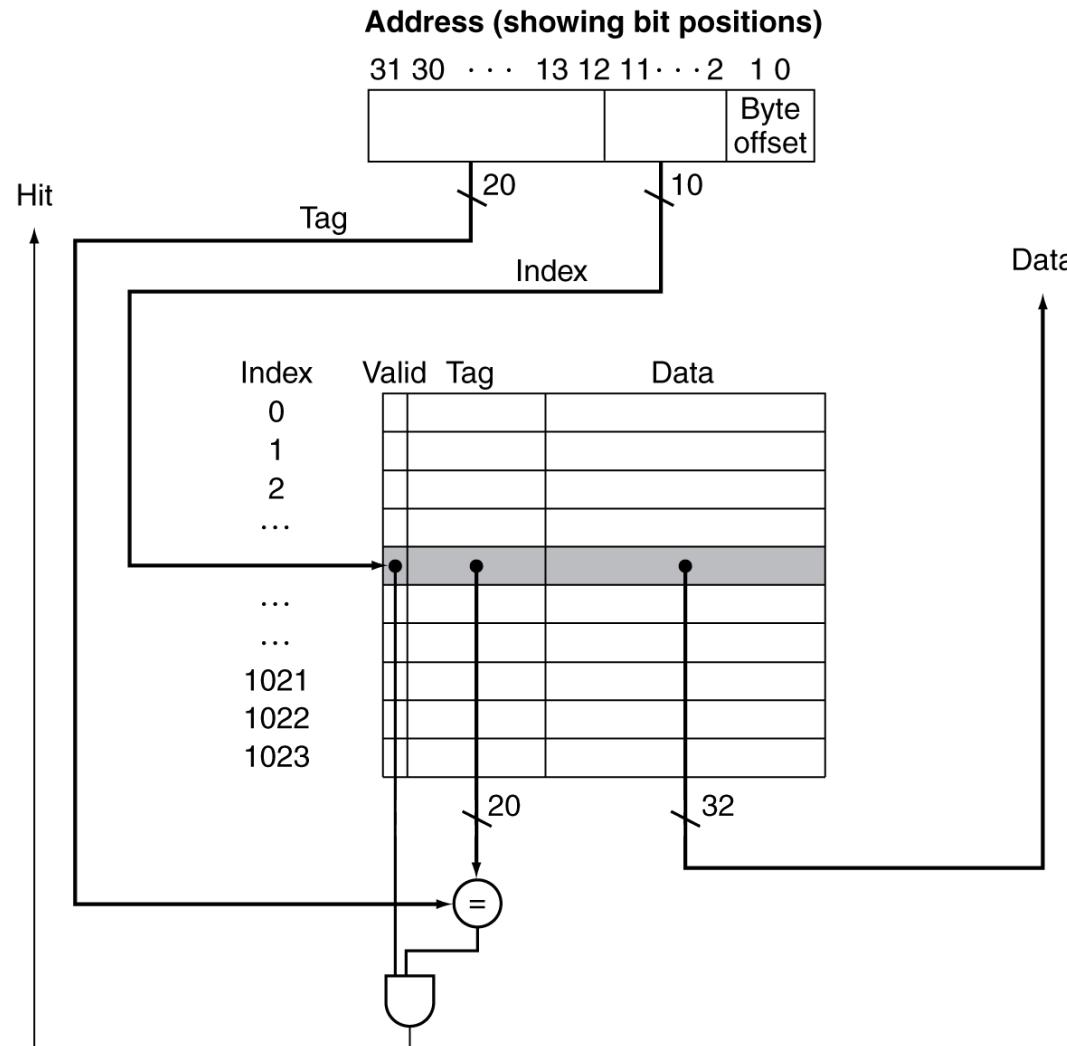
Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

# Cache Example

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

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

# Address Subdivision



# Example (WPS)

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



# What is the effect of block size ?

# 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 miss penalty
  - Can override benefit of reduced miss rate

# 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
  - write to memory takes 100 cycles
- 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

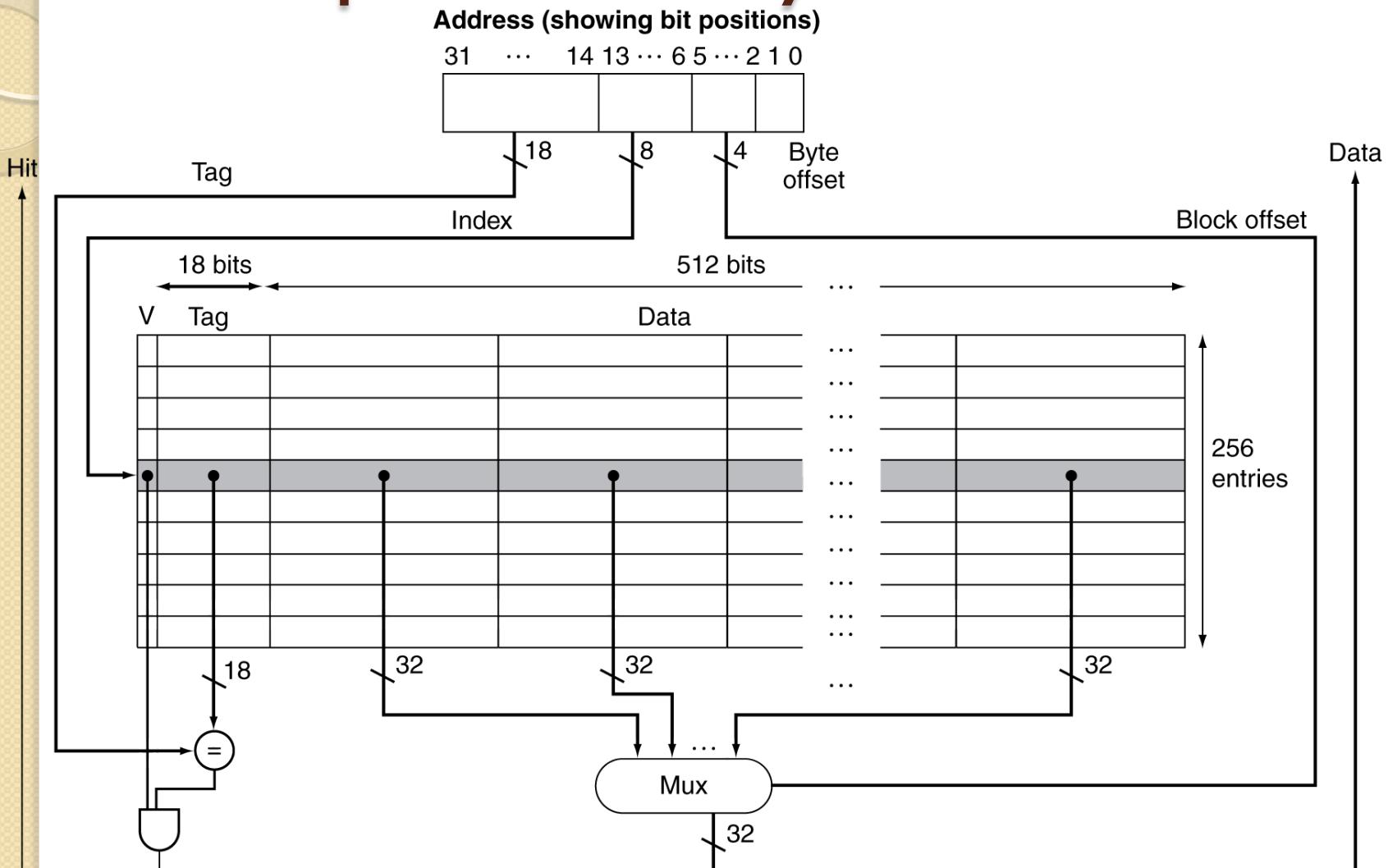
# Write Allocation

- 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

# 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 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

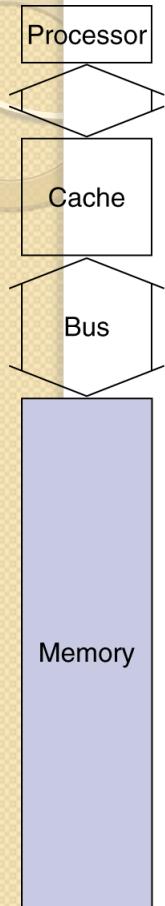
# Example: Intrinsity FastMATH



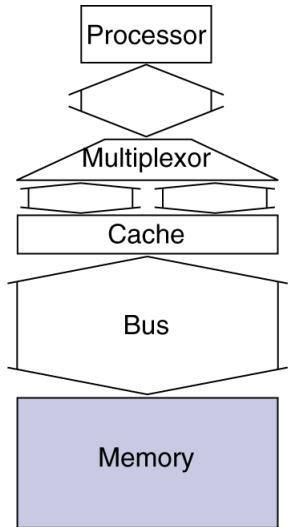
# Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example cache block read
  - 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 + 4×15 + 4×1 = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

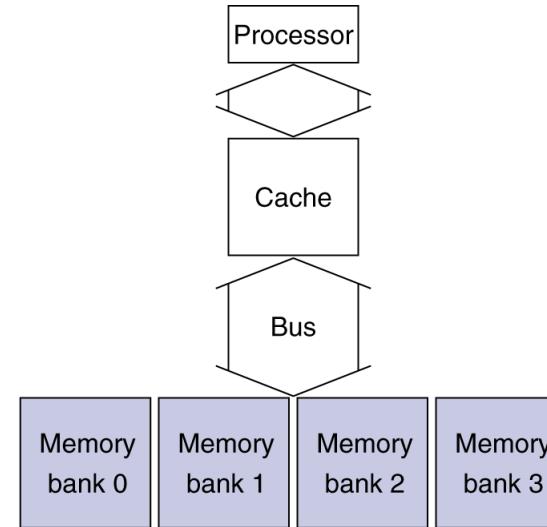
# Increasing Memory Bandwidth



a. One-word-wide  
memory organization



b. Wider memory organization



c. Interleaved memory organization

## ■ 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 = ??
- Bandwidth = ??

# Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

# 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

# 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

- Given
  - 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
  - I-cache:  $0.02 \times 100 = 2$
  - D-cache:  $0.36 \times 0.04 \times 100 = 1.44$
- 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 (Why neglect it ?)
- Average memory access time (AMAT)
  - $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%
  - $AMAT = 1 + 0.05 \times 20 = 2\text{ns}$ 
    - 2 cycles per instruction

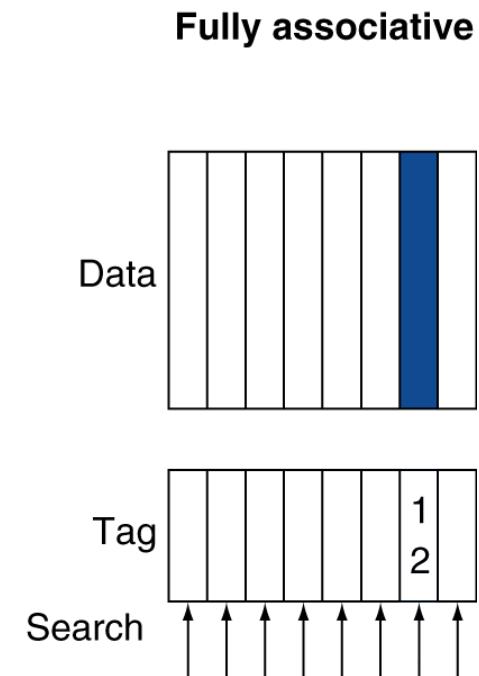
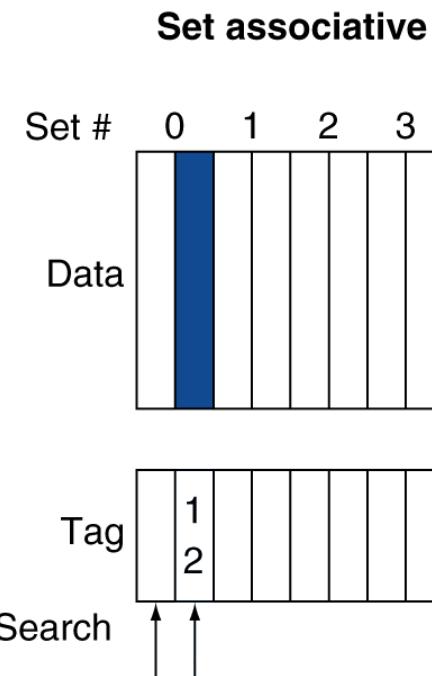
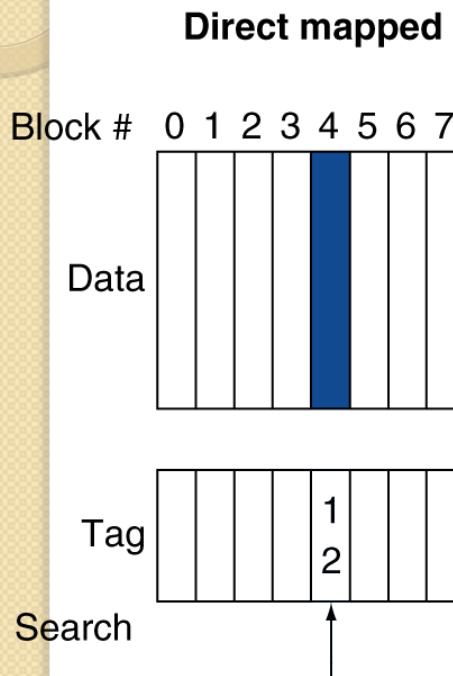
# Performance Summary

- 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)

# Associative Cache Example



# Spectrum of Associativity

- For a cache with 8 entries

**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														

# Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

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

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]

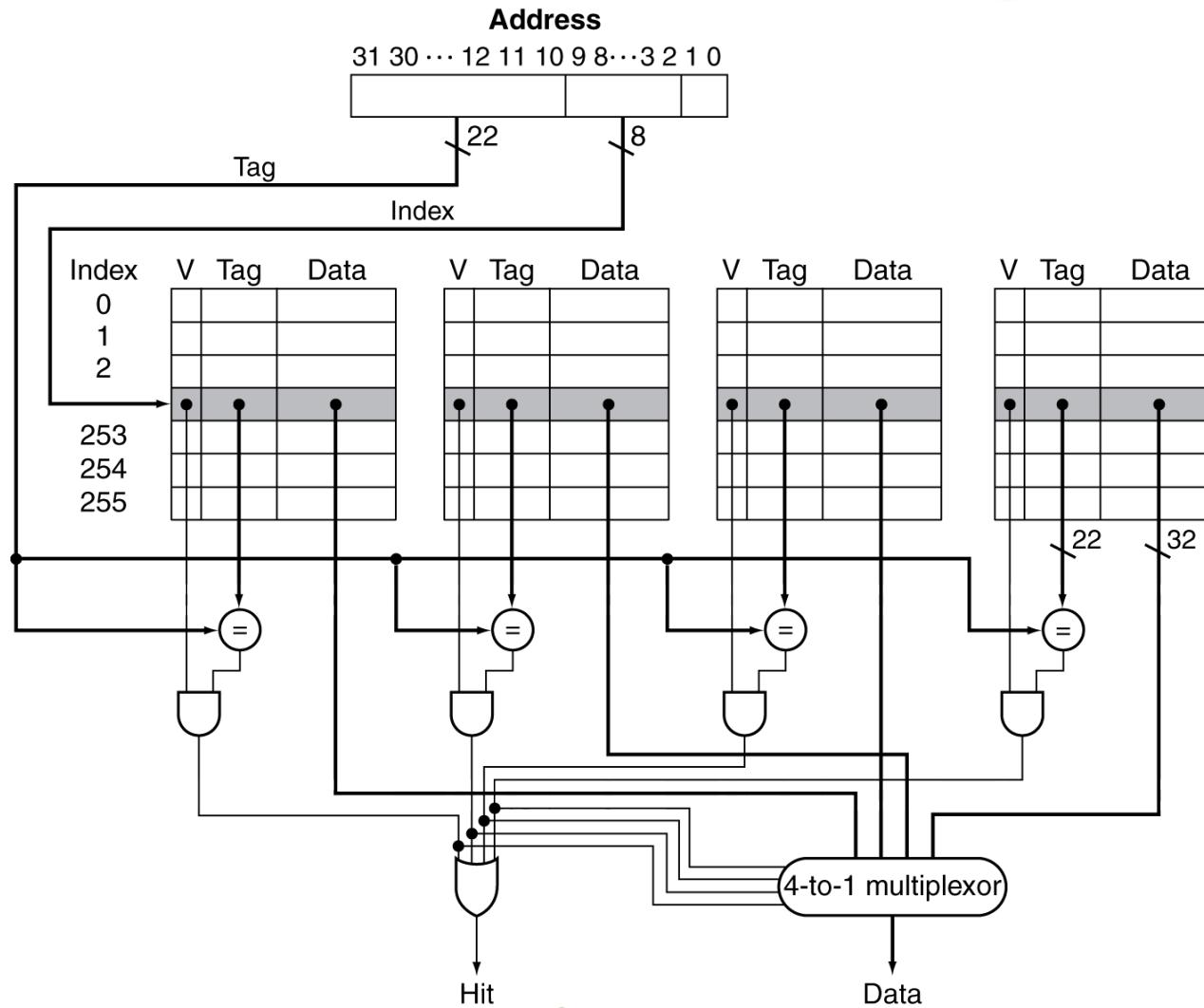
- Fully associative

Block address		Hit/miss	Cache content after access		
			Set 0	Set 1	Set 2
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

# 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 = ?

# 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 =  $5\text{ns}/0.25\text{ns} = 20 \text{ cycles}$
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI =  $I + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance ratio =  $9/3.4 = 2.6$

# Multilevel Cache Considerations

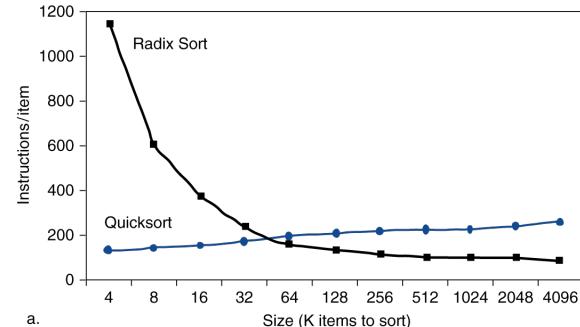
- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

# Interactions with Advanced CPUs

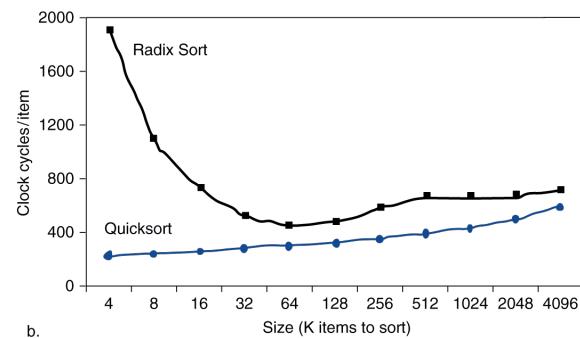
- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

# Interactions with Software

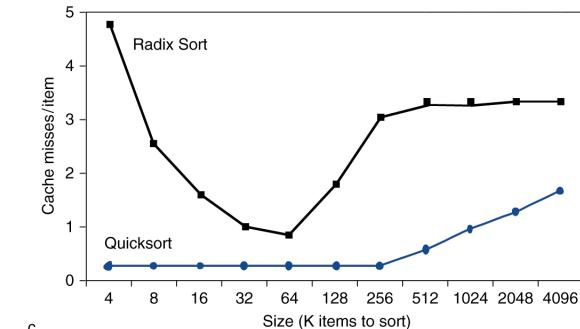
- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



a.



b.



c.



# Virtual Memory

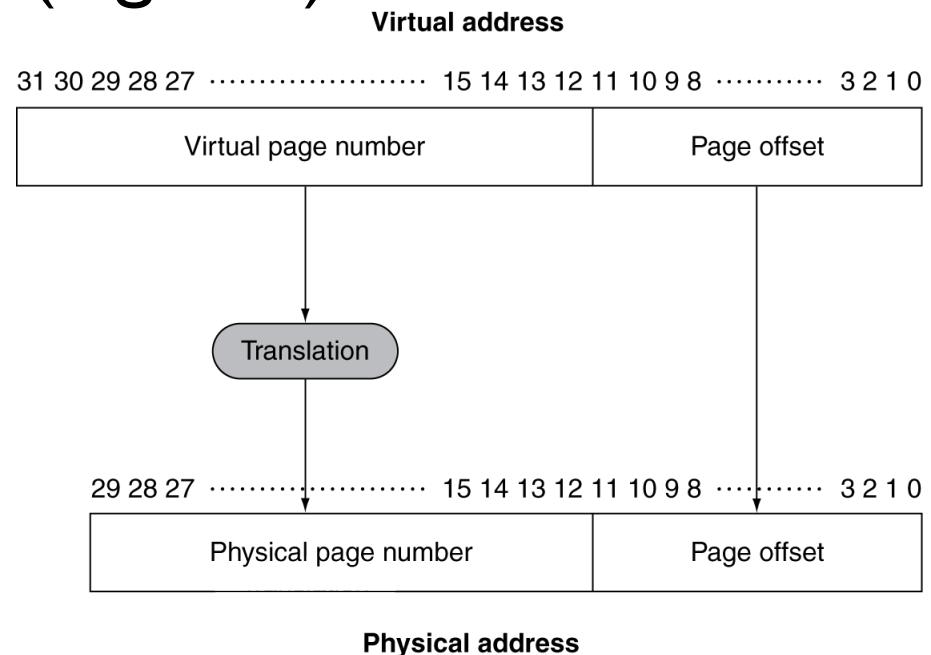
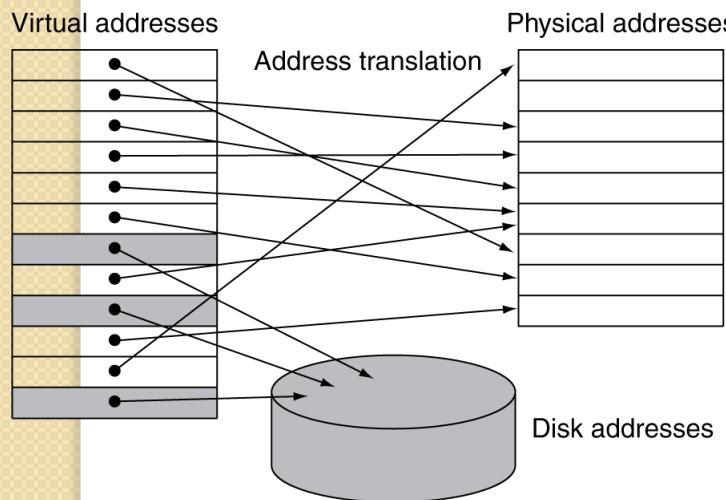
Instructor: Huzefa Rangwala, PhD

# Virtual Memory: Concept

- 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.
- Handles programs that may not fit in the physical memory

# Address Translation

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



# Realizing Virtual Memory

## Paging

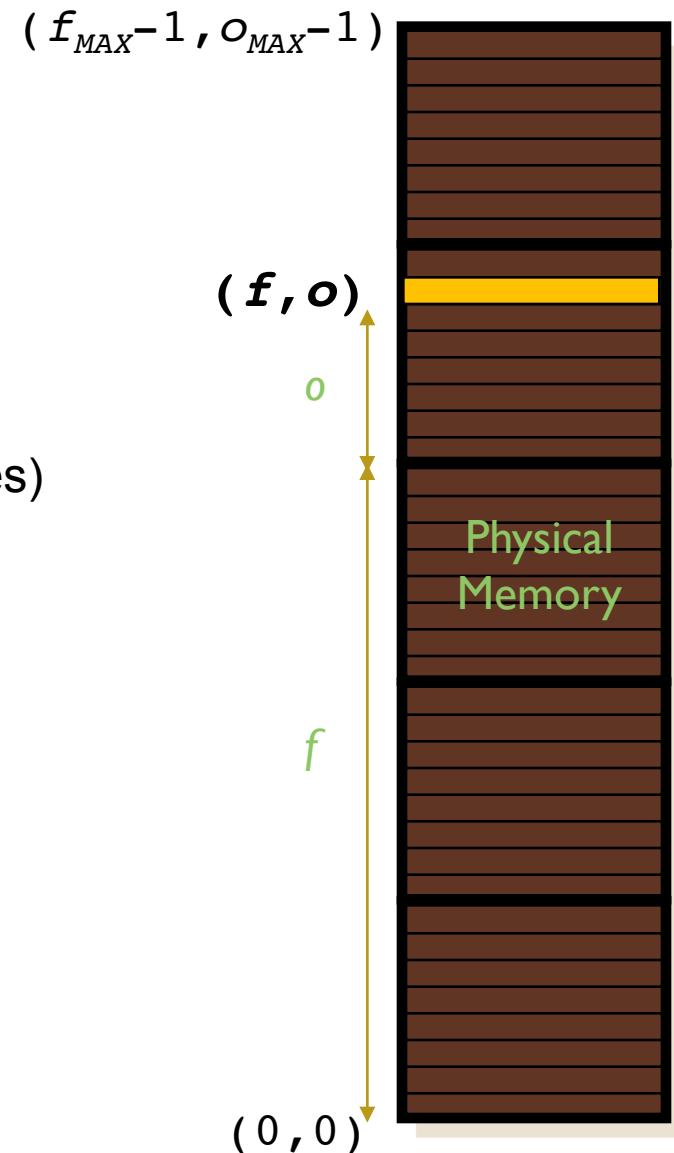
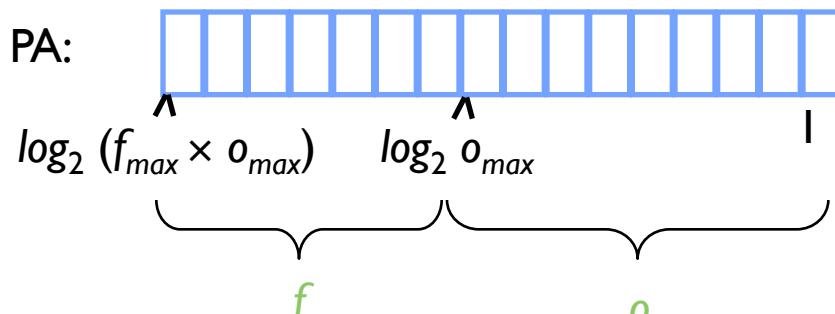
- Physical memory partitioned into equal sized *page frames or just pages*
  - Page frames avoid external fragmentation.

A memory address is a pair  $(f, o)$

$f$  — frame number ( $f_{max}$  frames)

$o$  — frame offset ( $o_{max}$  bytes/frames)

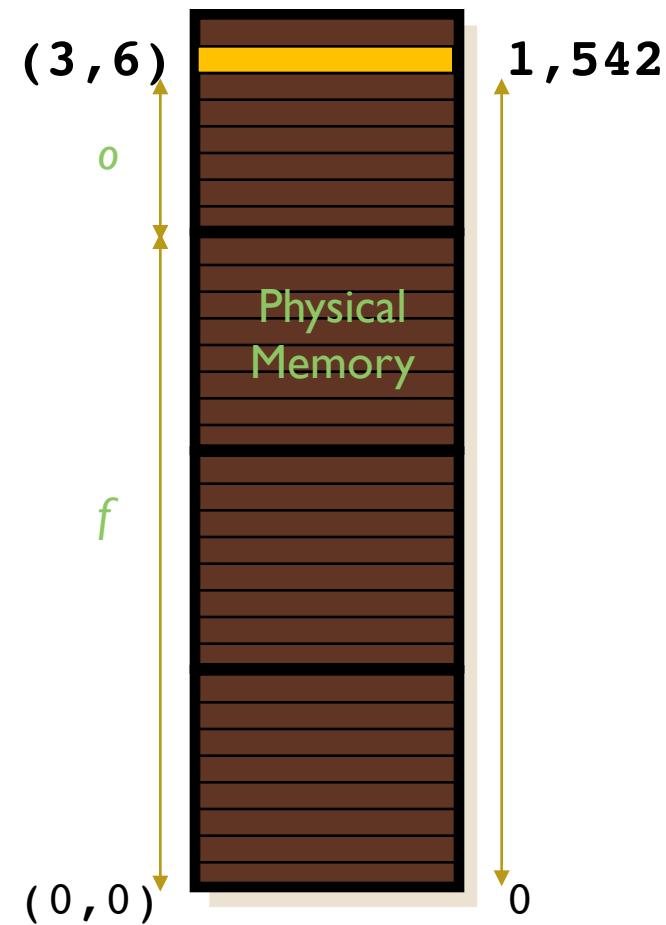
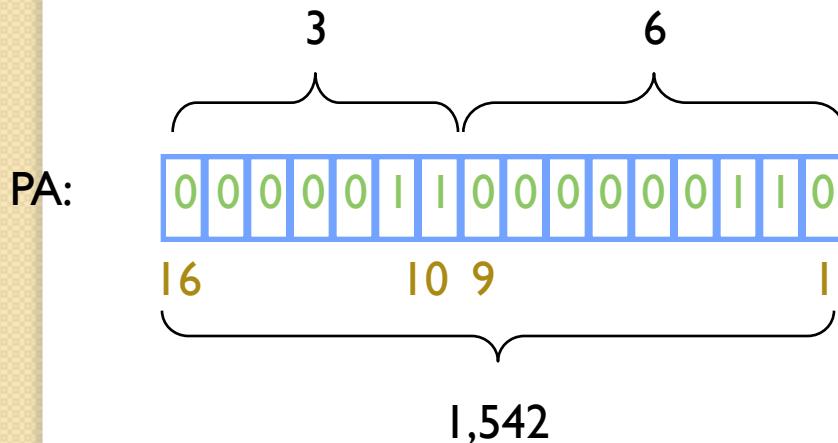
$$\text{Physical address} = o_{max} \times f + o$$



# Physical Address Specifications

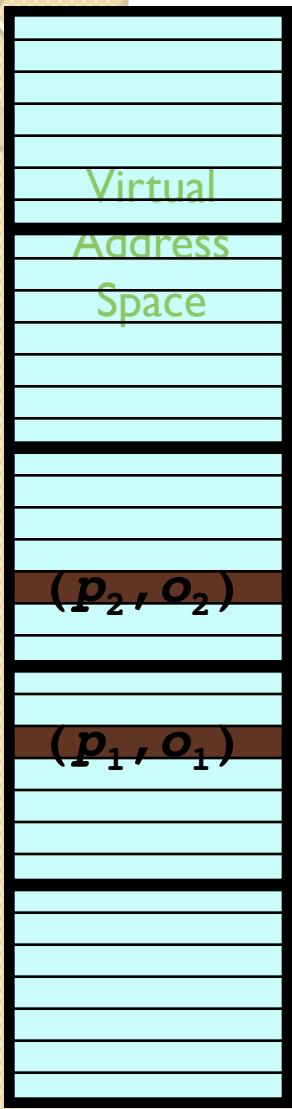
Frame/Offset pair v. An absolute index

- Example: A 16-bit address space with ( $o_{max} =$ ) 512 byte page frames
  - Addressing location (3, 6) = 1,542

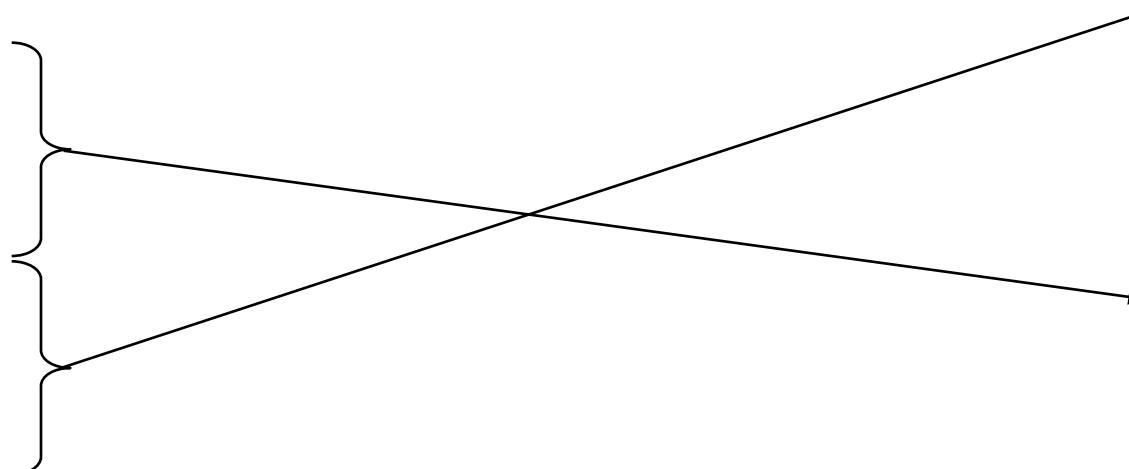
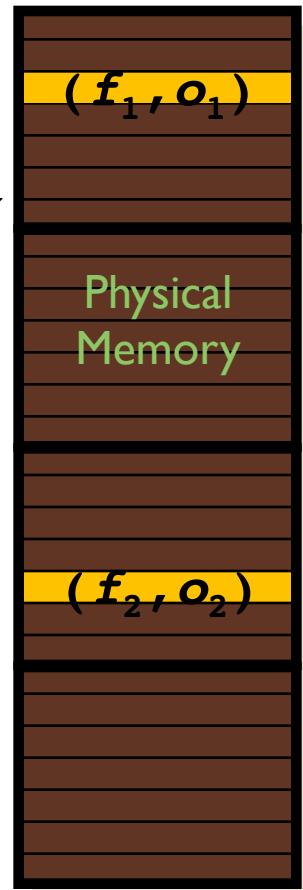


# Paging (Virtual Addresses)

Mapping virtual addresses to physical addresses

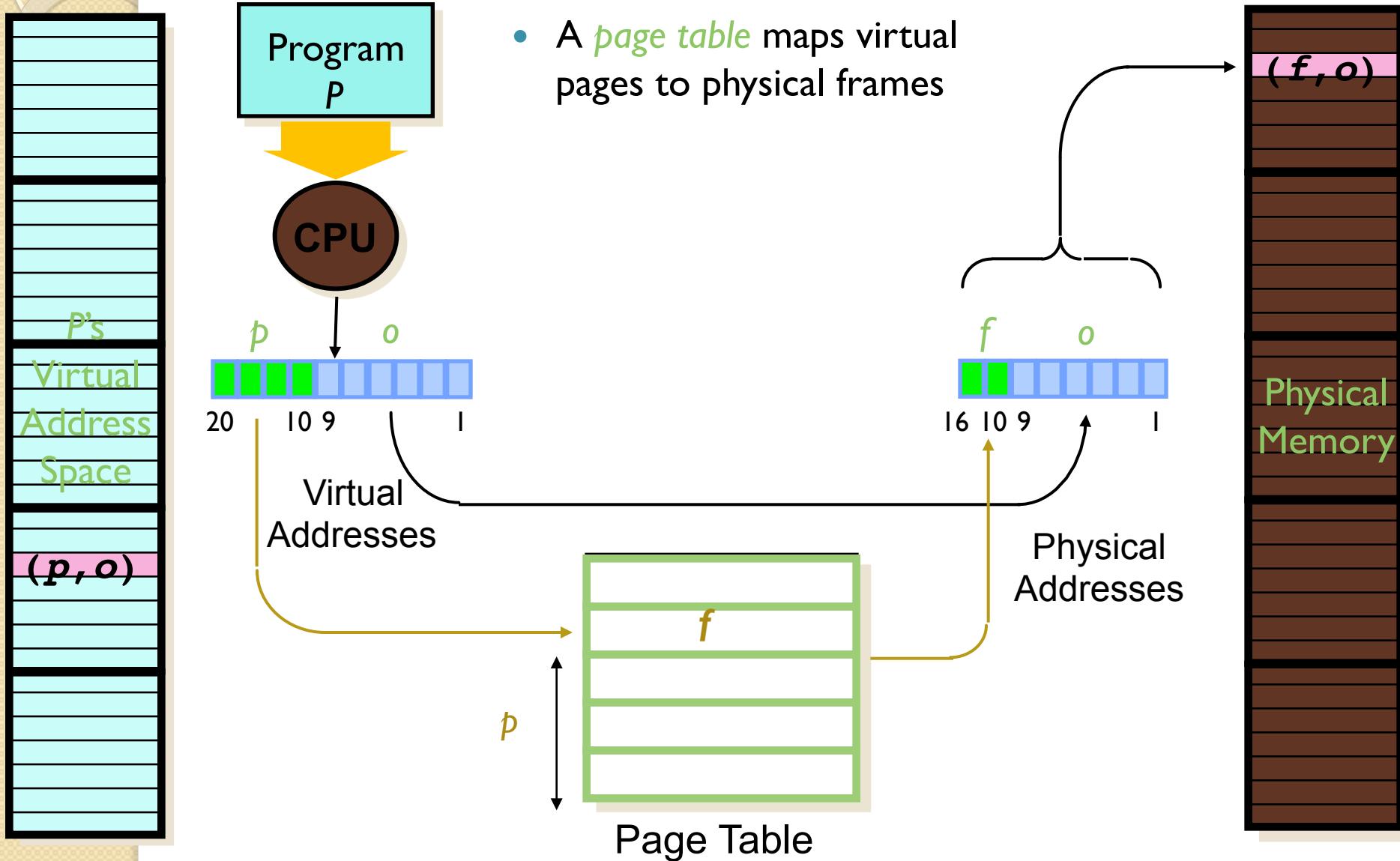


- Pages map to frames (pages) in PM
- Pages are contiguous in a VAS...
  - But pages are arbitrarily located in physical memory, and
  - Not all pages mapped at all times



# Paging

Virtual address translation

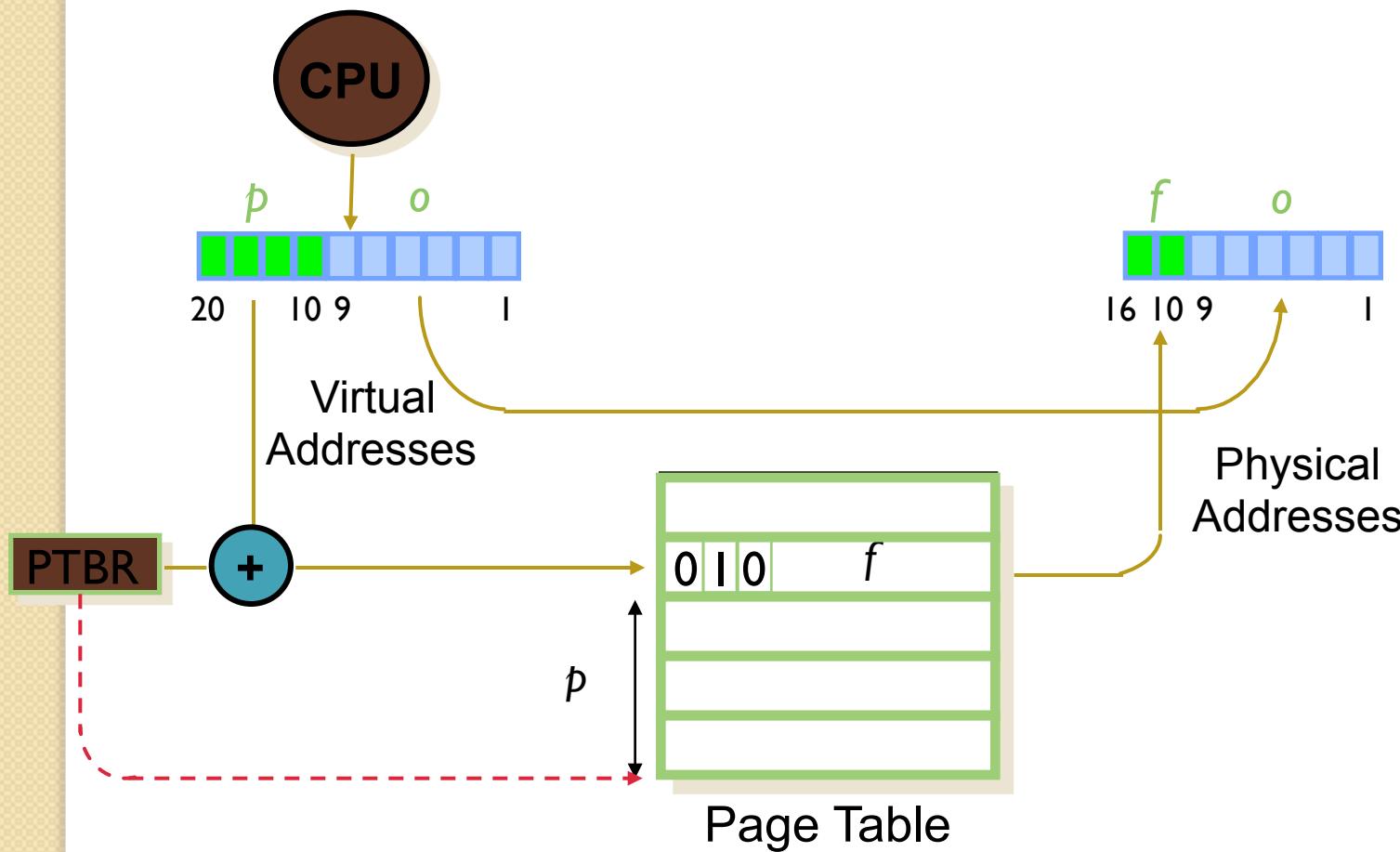


# Virtual Address Translation Details

## Page table structure

1 table per process  
Part of process's state

- Contents:
  - Flags — dirty bit, resident bit, clock/reference bit
  - Frame number



# Virtual Address Translation

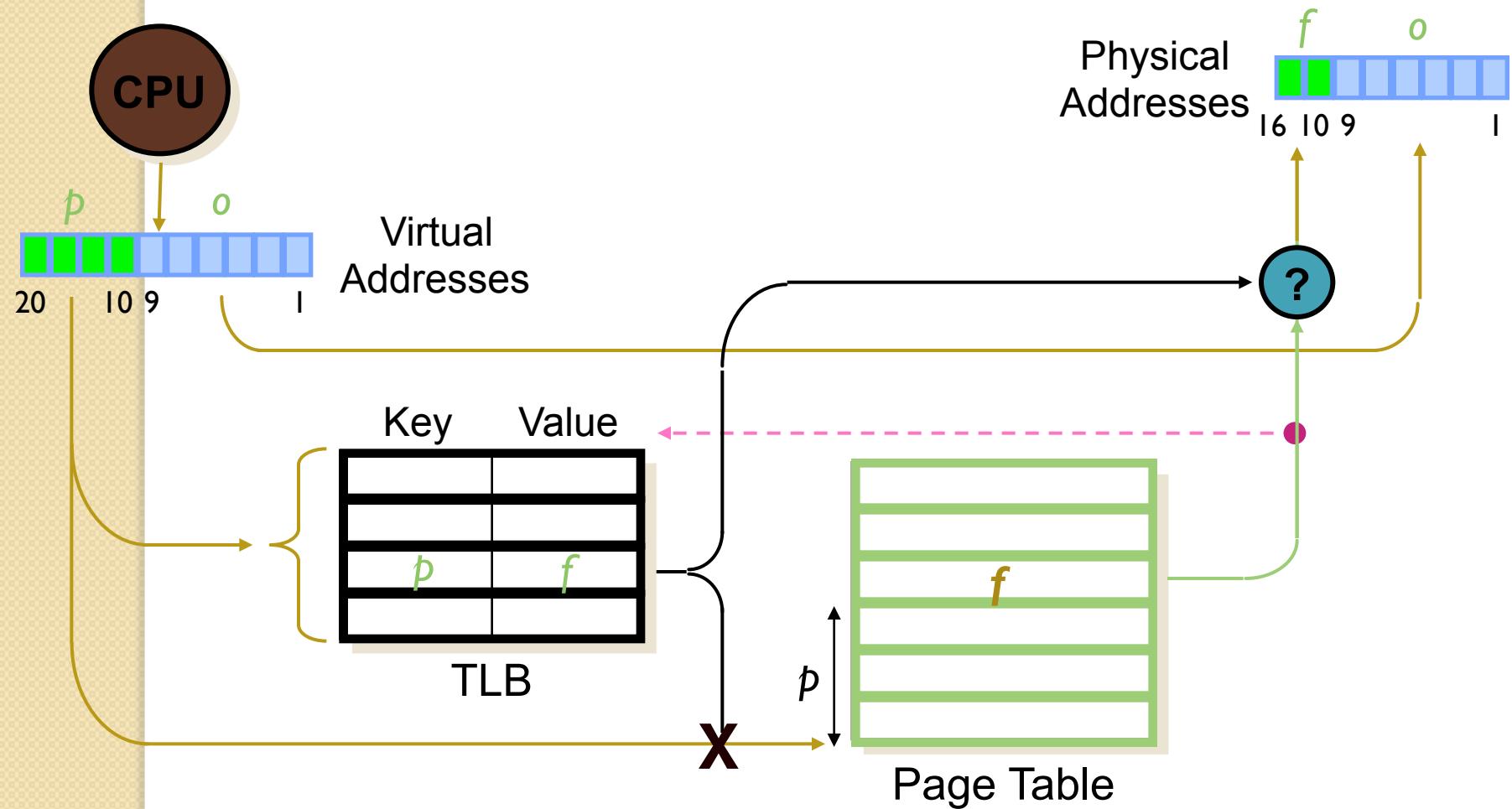
## Performance Issues

- Problem — VM reference requires 2 memory references!
  - One access to get the page table entry
  - One access to get the data
- Page table can be very large; a part of the page table can be on disk.
- What to do?
  - Most computing problems are solved by some form of...
    - Caching
    - Indirection

# Virtual Address Translation

## Using TLBs to Speedup Address Translation

- Cache recently accessed page-to-frame translations in a TLB
  - For TLB hit, physical page number obtained in 1 cycle
  - For TLB miss, translation is updated in TLB
  - Has high hit ratio (why?)

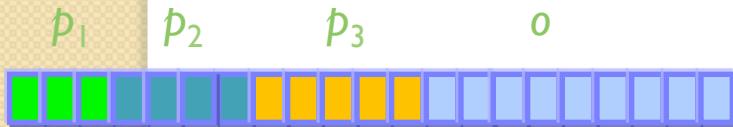


# Dealing With Large Page Tables

## Multi-level paging

- Add additional levels of indirection to the page table by sub-dividing page number into  $k$  parts
  - Create a “tree” of page tables
  - TLB still used, just not shown
  - The architecture determines the number of levels of page table

Virtual Address



$p_1$

$p_2$

$p_3$

$o$

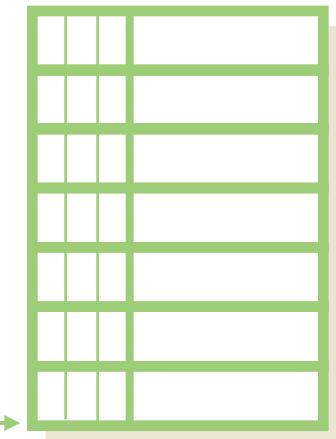
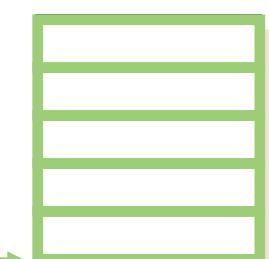
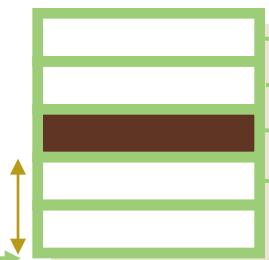
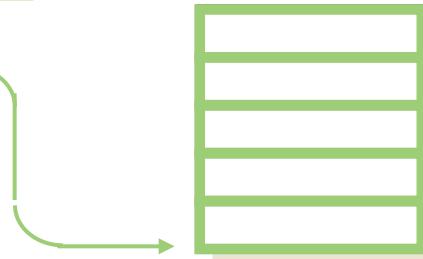
First-Level  
Page Table

Second-Level  
Page Tables

$p_2$

$p_3$

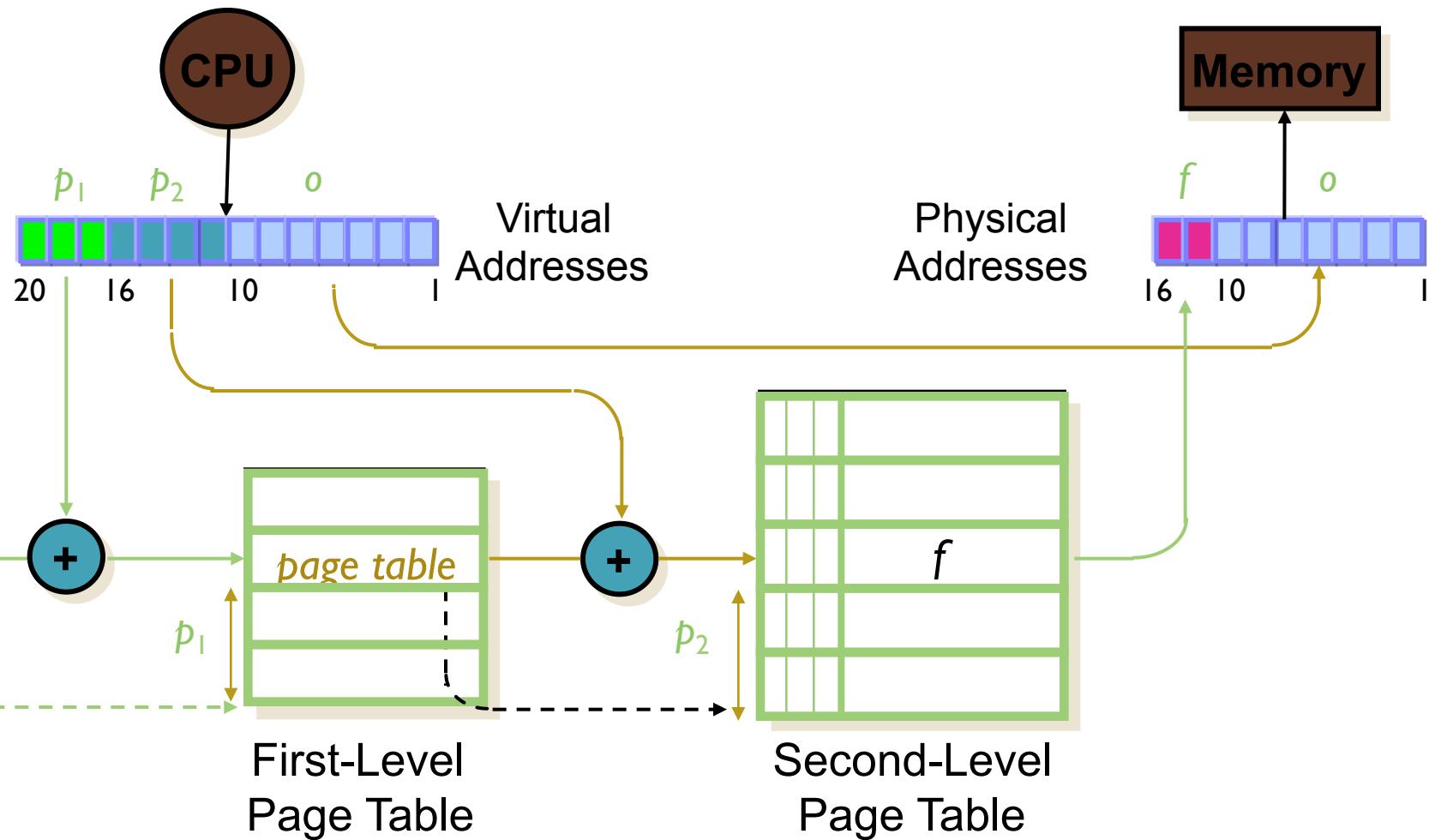
Third-Level  
Page Tables



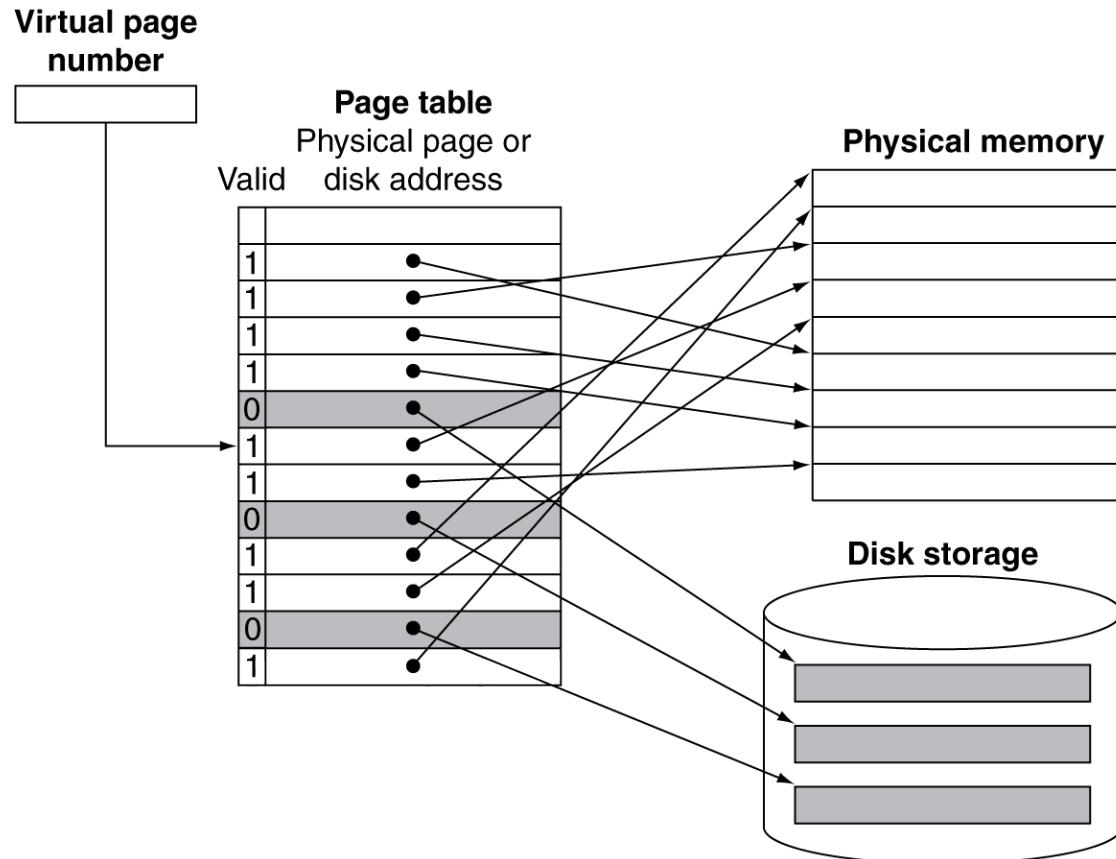
# Dealing With Large Page Tables

Multi-level paging

- Example: Two-level paging



# Mapping Pages to Storage



# 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

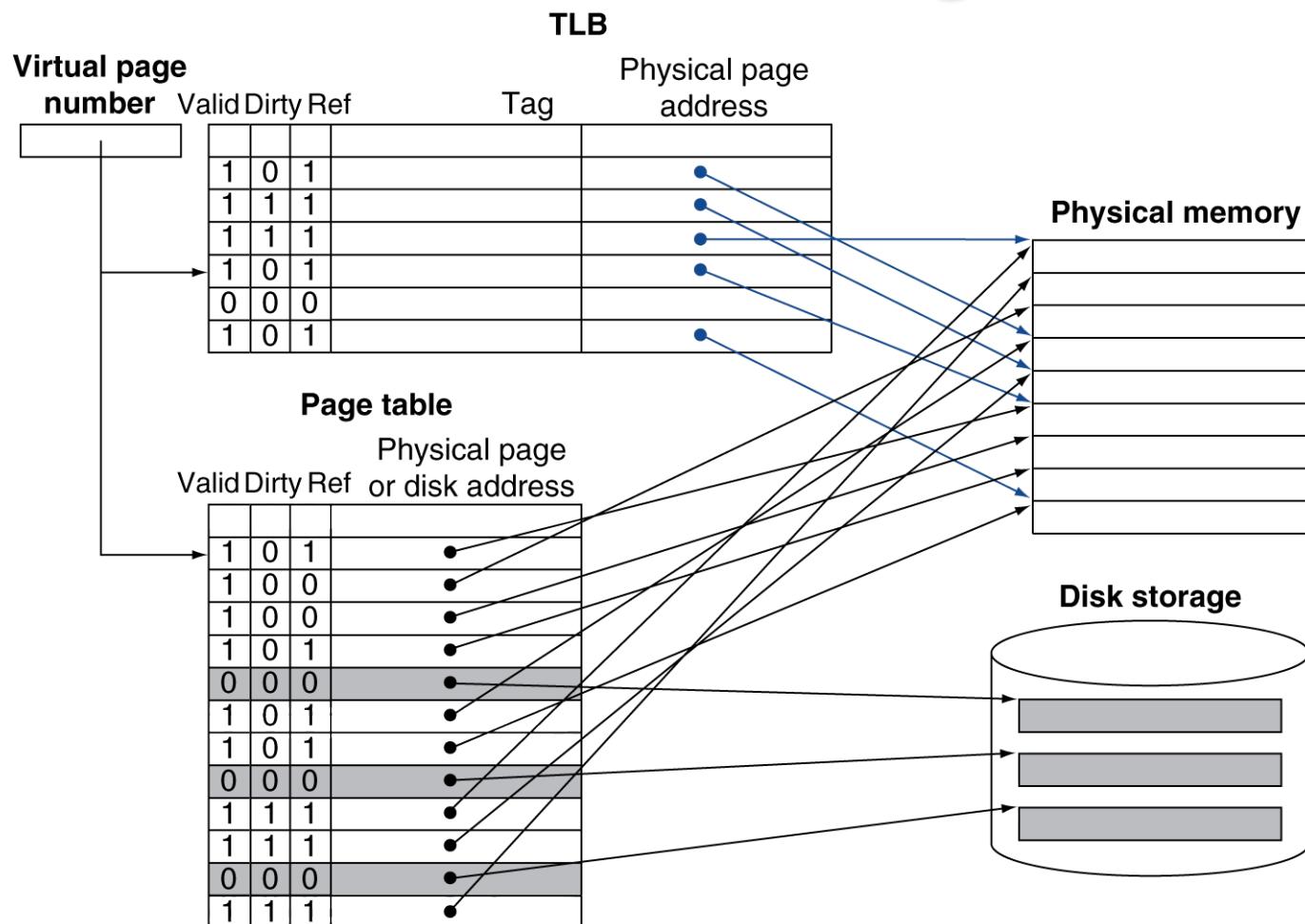
# Page Tables

- 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

# 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

# Virtual Memory

## Page fault handling

- References to non-mapped pages generate a *page fault*

### Page fault handling steps:

Processor runs the interrupt handler

OS blocks the running process

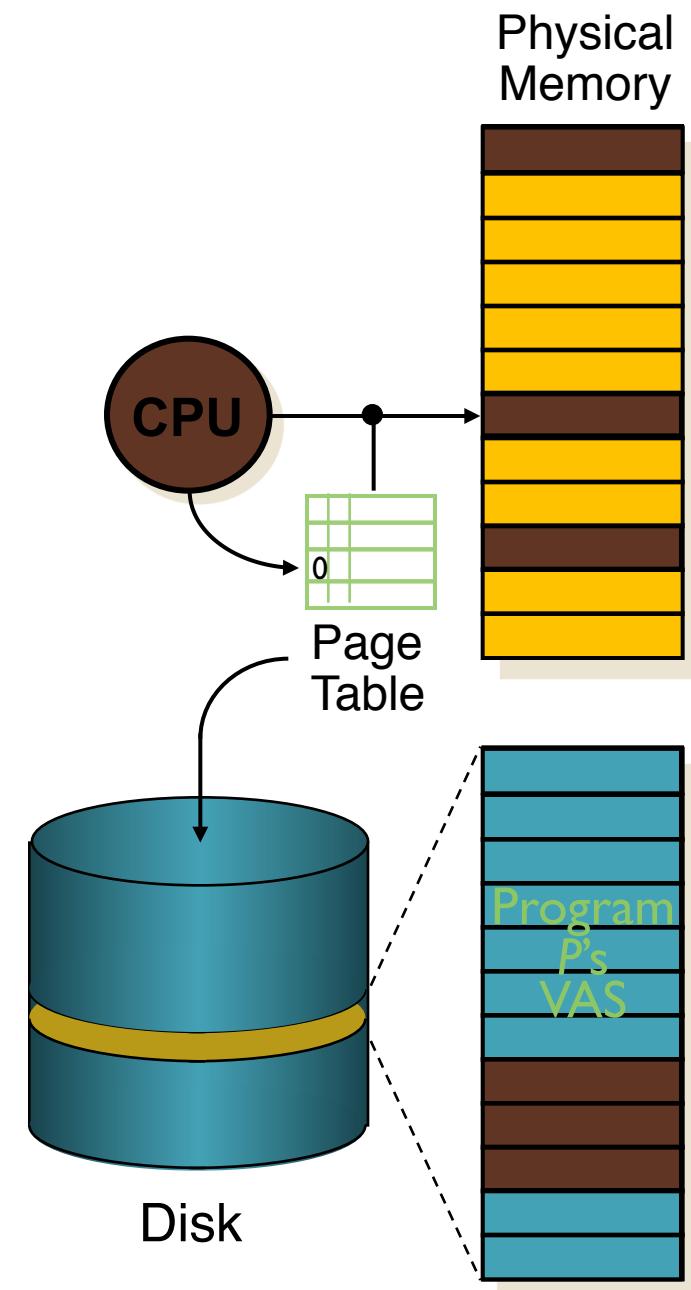
OS starts read of the unmapped page

OS resumes/initiates some other process

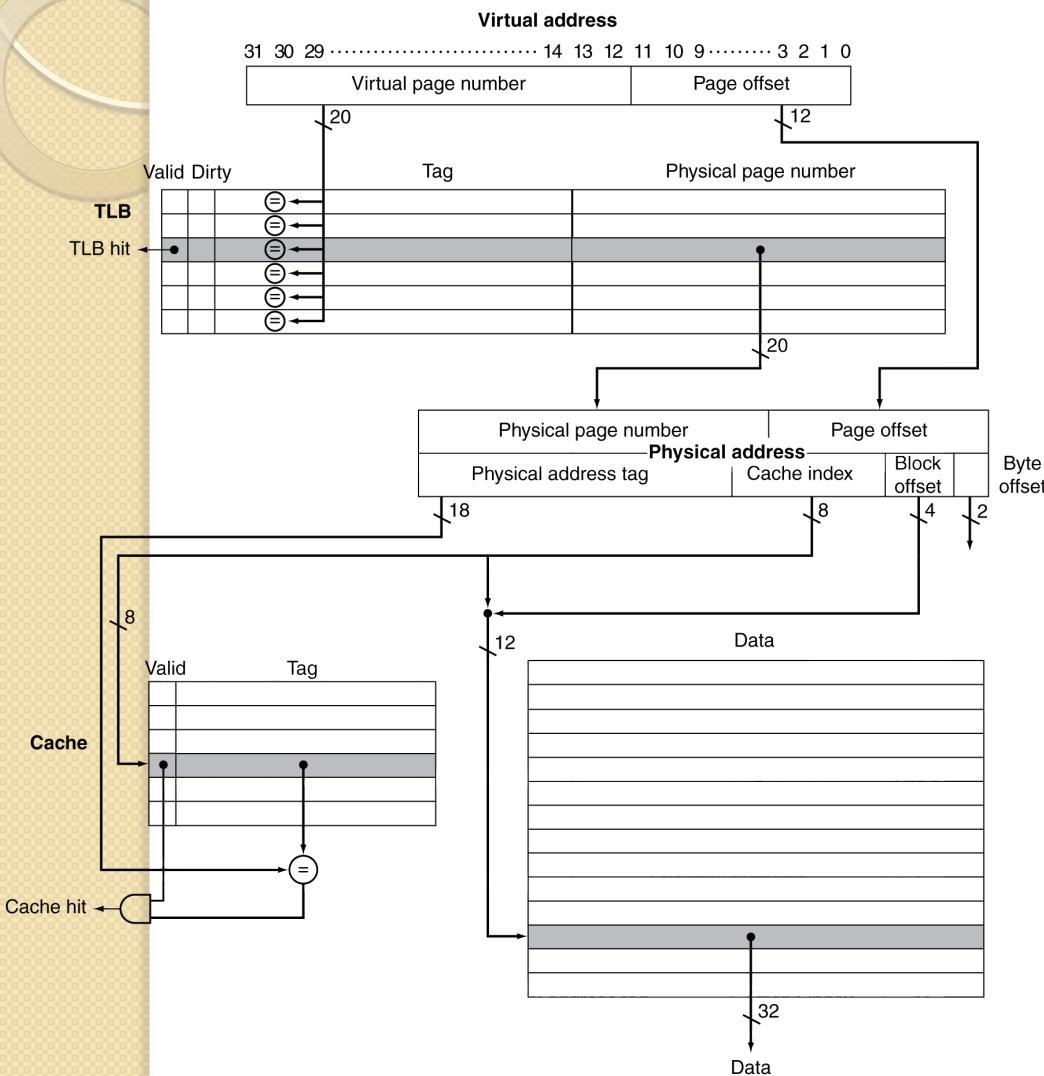
Read of page completes

OS maps the missing page into memory

OS restarts the faulting process



# 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

# 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)

# The Memory Hierarchy

## The BIG Picture

- 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

# Block Placement

- Determined by associativity
  - Direct mapped (one-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

# Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

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

# Replacement

- 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

# Write Policy

- 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 state information
- Virtual memory
  - Only write-back is feasible, given disk write latency

# Sources of Misses

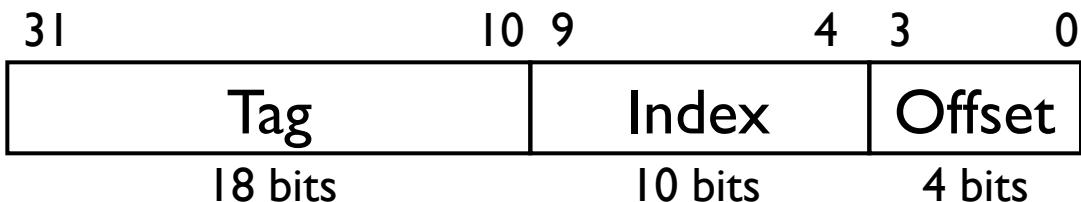
- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur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

# Cache Design Trade-offs

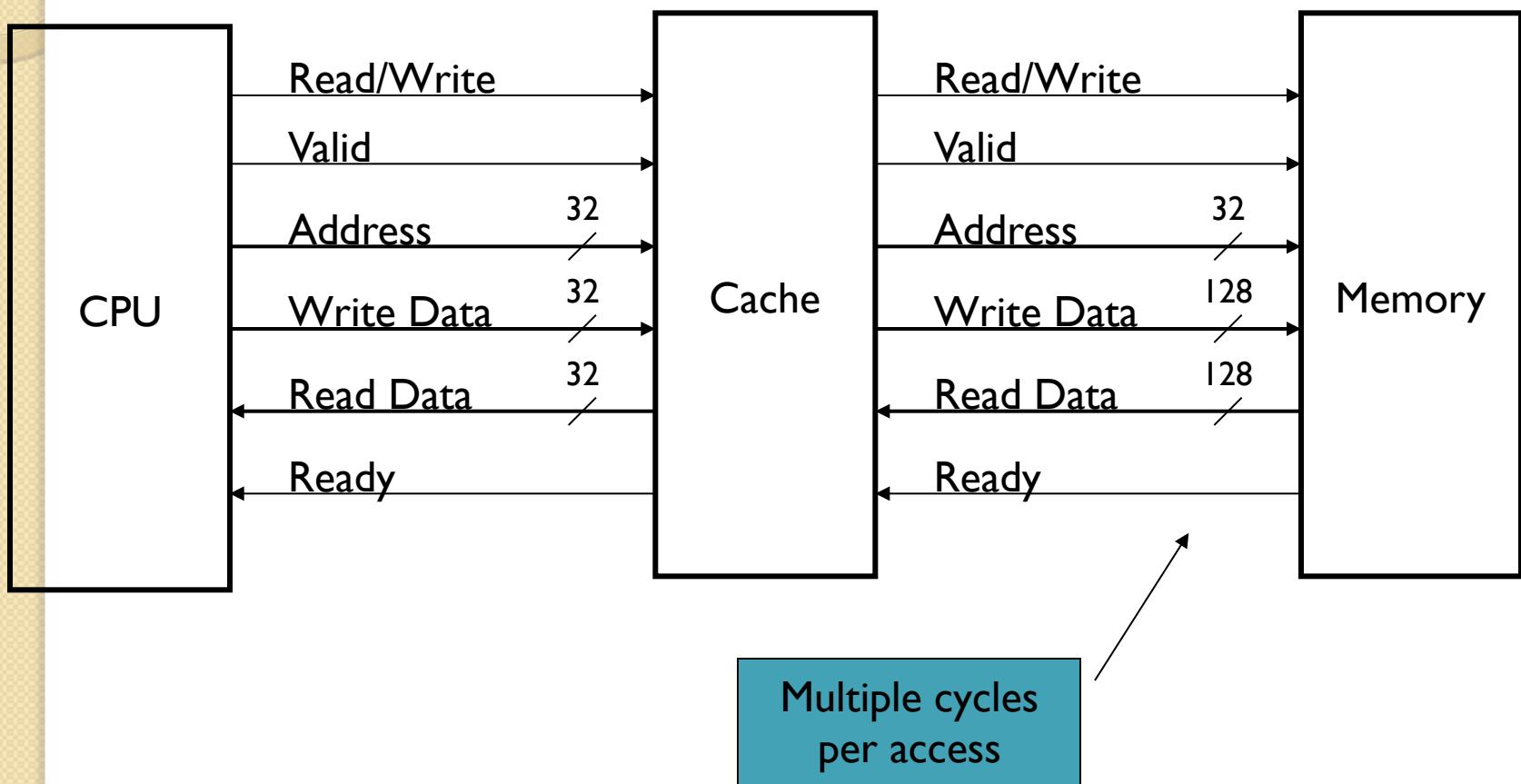
Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size,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

# Cache Control

- Example cache characteristics
  - Direct-mapped, write-back, write allocate
  - Block size: 4 words (16 bytes)
  - Cache size: 16 KB (1024 blocks)
  - 32-bit byte addresses
  - Valid bit and dirty bit per block
  - **Blocking cache**
    - CPU waits until access is complete

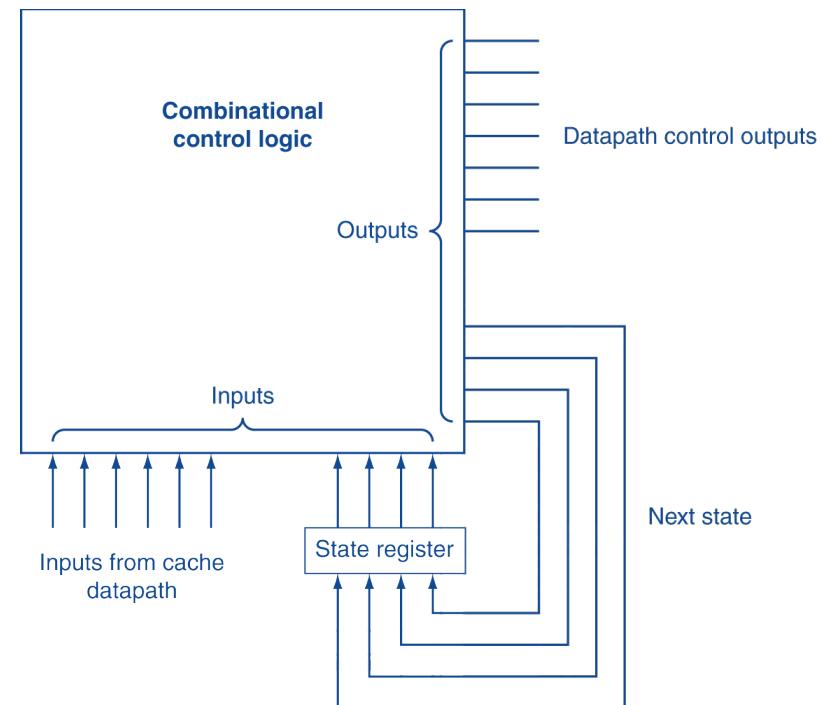


# Interface Signals

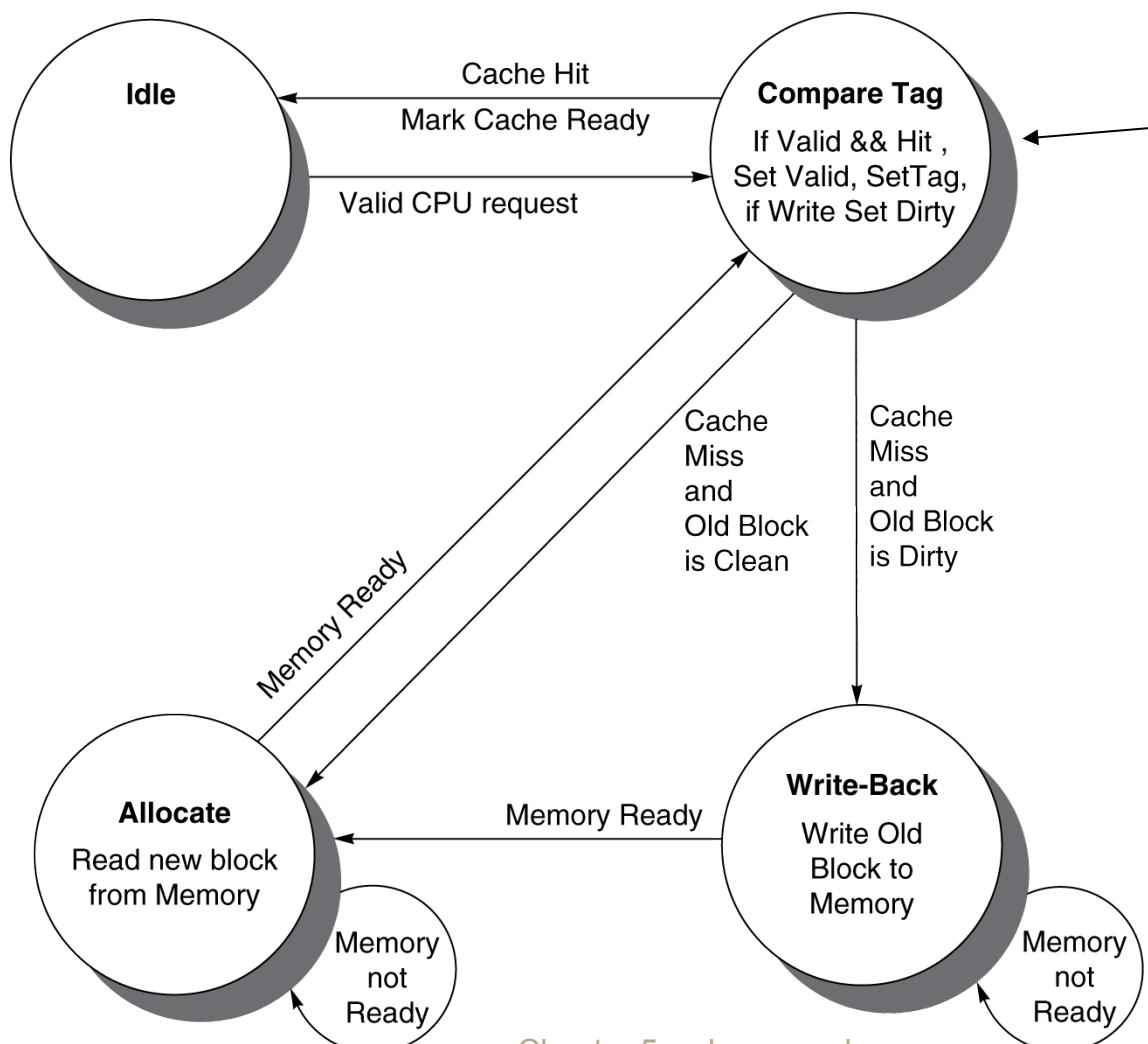


# Finite State Machines

- Use an FSM to sequence control steps
- Set of states, transition on each clock edge
  - State values are binary encoded
  - Current state stored in a register
  - Next state =  $f_n$  (current state, current inputs)
- Control output signals =  $f_o$  (current state)



# Cache Controller FSM



Could partition into separate states to reduce clock cycle time

# Cache Coherence Problem

- Suppose two CPU cores share a physical address space
  - Write-through caches

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

# Coherence Defined

- Informally: Reads return most recently written value
- Formally:
  - P writes X; P reads X (no intervening writes)  
⇒ read returns written value
  - P<sub>1</sub> writes X; P<sub>2</sub> reads X (sufficiently later)  
⇒ read returns written value
    - c.f. CPU B reading X after step 3 in example
  - P<sub>1</sub> writes X, P<sub>2</sub> writes X  
⇒ all processors see writes in the same order
    - End up with the same final value for X

# Cache Coherence Protocols

- Operations performed by caches in multiprocessors to ensure coherence
  - Migration of data to local caches
    - Reduces bandwidth for shared memory
  - Replication of read-shared data
- Snooping protocols
  - Each cache monitors bus reads/writes
- Directory-based protocols
  - Caches and memory record sharing status of blocks in a directory

# Invalidating Snooping Protocols

- Cache gets exclusive access to a block when it is to be written
  - Broadcasts an invalidate message on the bus
  - Subsequent read in another cache misses
    - Owning cache supplies updated value

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidate for X	1		0
CPU B read X	Cache miss for X	1	1	1

# Memory Consistency

- When are writes seen by other processors
  - “Seen” means a read returns the written value
  - Can’t be instantaneously
- Assumptions
  - A write completes only when all processors have seen it
  - A processor does not reorder writes with other accesses
- Consequence
  - P writes X then writes Y  
⇒ all processors that see new Y also see new X
  - Processors can reorder reads, but not writes

# Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache  $\leftrightarrow$  L2 cache  $\leftrightarrow \dots \leftrightarrow$  DRAM memory  $\leftrightarrow$  disk
- Memory system design is critical for multiprocessors