National Tsing Hua University

COMPUTER ARCHITECTURE

Memory Hierarchy

國立清華大學資訊工程學系

黃婷婷教授

Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Memory Technology

- **Random access:**
  - **Access time same for all locations**
  - **SRAM: *Static Random Access Memory***
    - **Low density, high power, expensive, fast**
    - **Static: content will last (forever until lose power)**
    - **Address not divided**
    - **Use for caches**
  - **DRAM: *Dynamic Random Access Memory***
    - **High density, low power, cheap, slow**
    - **Dynamic: need to be refreshed regularly**
    - **Addresses in 2 halves (memory as a 2D matrix):**
      RAS/CAS (Row/Column Access Strobe)
    - **Use for main memory**
- **Magnetic disk**

Comparisons of Various Technologies

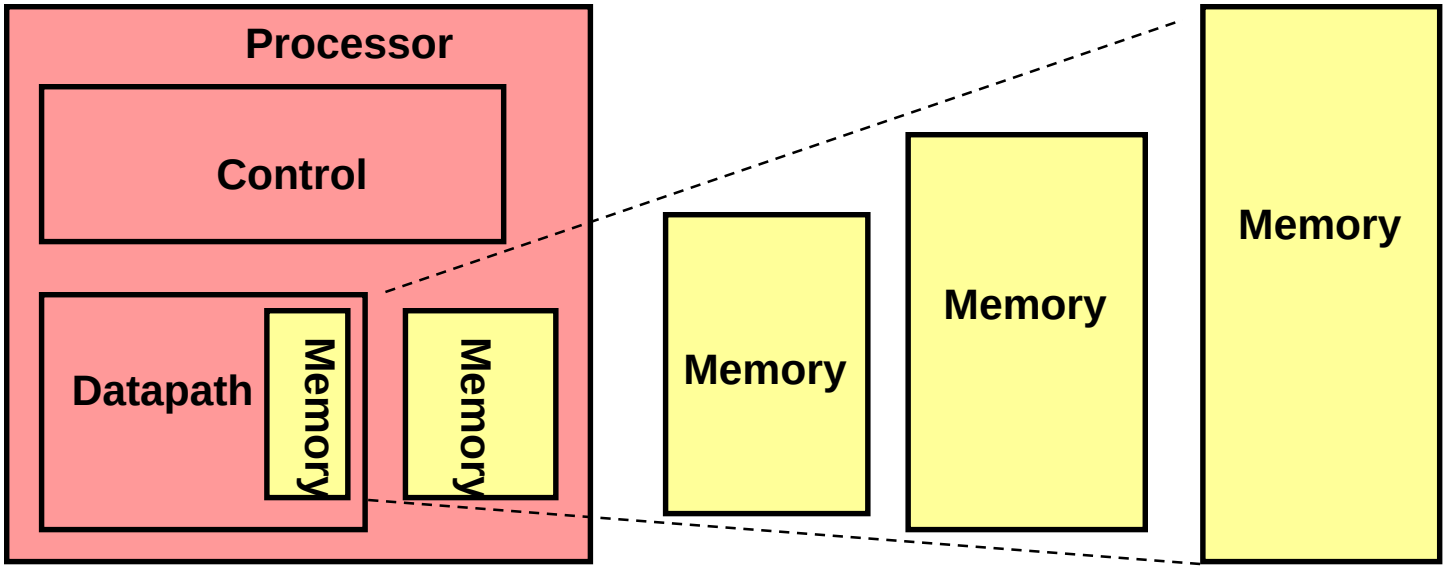
Memory technology	Typical access time	$ per GB in 2008
SRAM	0.5 – 2.5 ns	$2000 – $5,000
DRAM	50 – 70 ns	$20 – $75
Magnetic disk	5,000,000 – 20,000,000 ns	$0.20 – $2

Ideal memory

- Access time of SRAM
- Capacity and cost/GB of disk

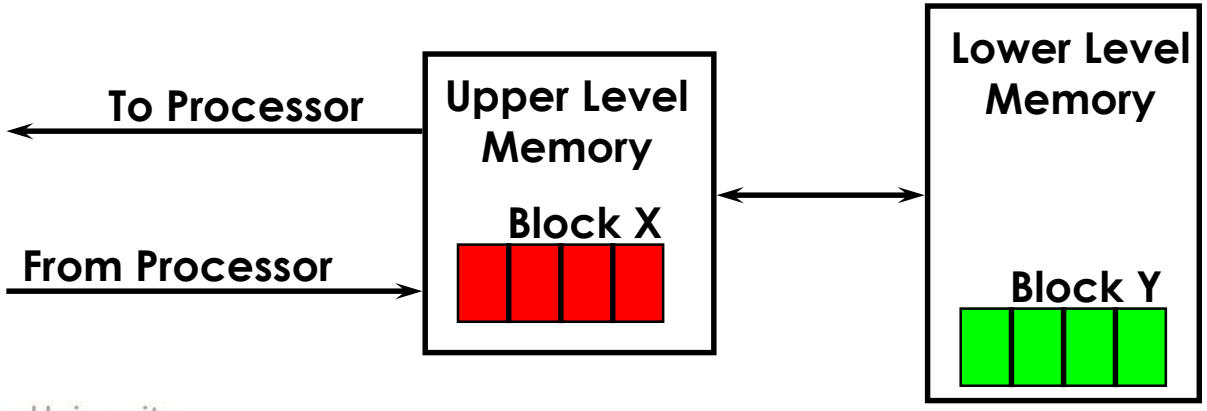
Solution: Memory Hierarchy

- An Illusion of a large, fast, cheap memory
  - Fact: Large memories slow, fast memories small
  - How to achieve: hierarchy, parallelism
- An expanded view of memory system:



Memory Hierarchy: Principle

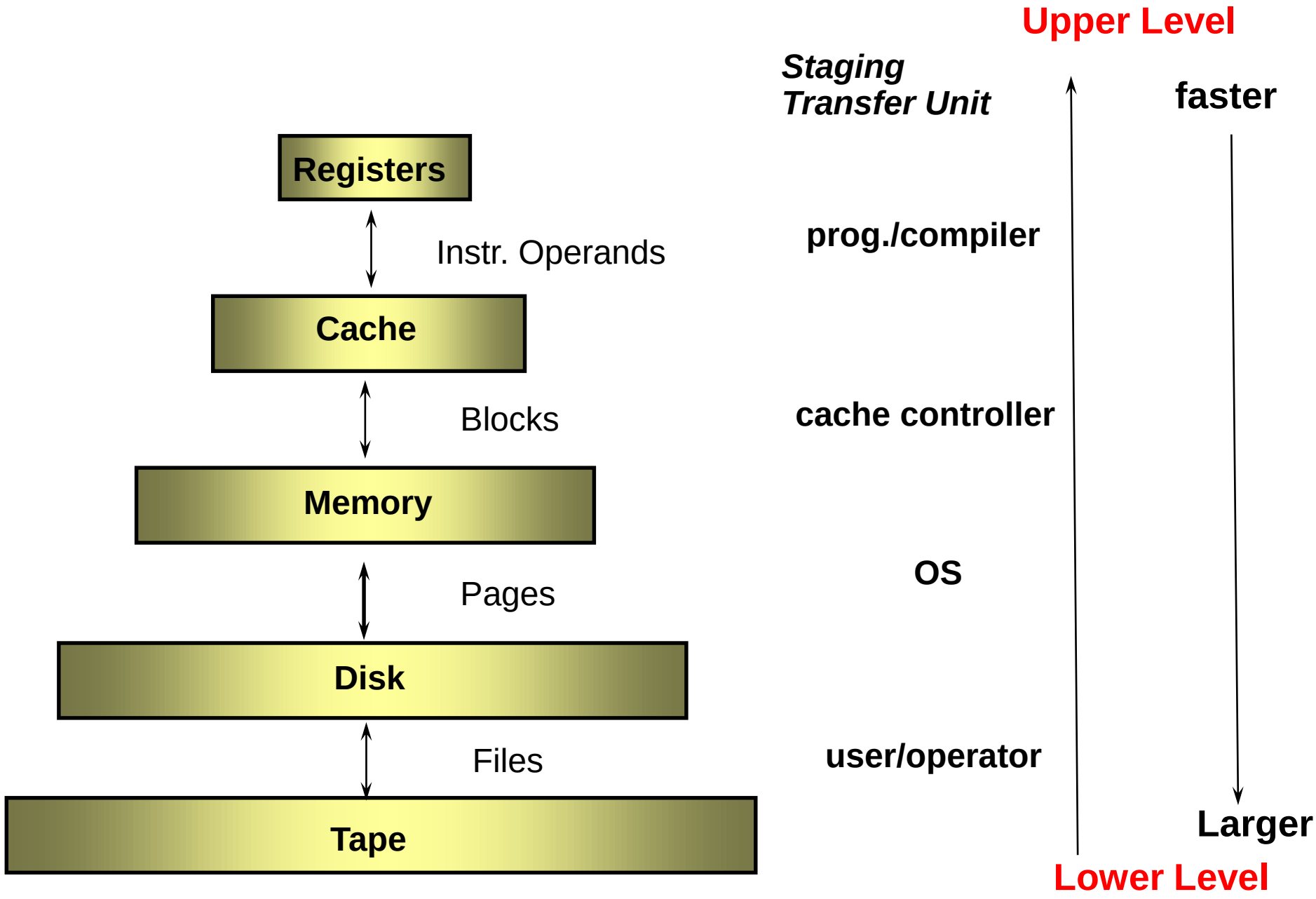
- At any given time, data is copied between only two adjacent levels:
  - Upper level: the one closer to the processor
    - Smaller, faster, uses more expensive technology
  - Lower level: the one away from the processor
    - Bigger, slower, uses less expensive technology
- *Block*: basic unit of information transfer
  - Minimum unit of information that can either be present or not present in a level of the hierarchy



Why Hierarchy Works?

- *Principle of Locality*:
  - Program access a relatively small portion of the address space at any instant of time
  - 90/10 rule: 10% of code executed 90% of time
- Two types of locality:
  - Temporal locality: if an item is referenced, it will tend to be referenced again soon, e.g., loop
  - Spatial locality: if an item is referenced, items whose addresses are close by tend to be referenced soon., e.g., instruction access, array data structure



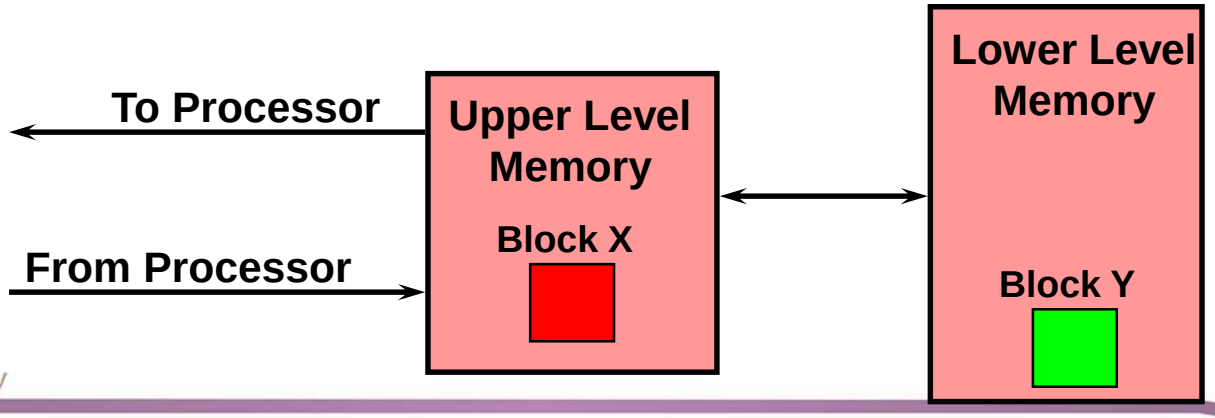

Upper Level
Staging
Transfer Unit
faster
Registers
Instr. Operands
prog./compiler
Cache
Blocks
cache controller
Memory
OS
Pages
Disk
Files
user/operator
Tape
Larger
Lower Level

How Is the Hierarchy Managed?

- Registers <-> Memory
  - by compiler (programmer?)
- cache <-> memory
  - by the hardware
- memory <-> disks
  - by the hardware and operating system (virtual memory)
  - by the programmer (files)

Memory Hierarchy: Terminology

- Hit: data appears in upper level (Block X)
  - Hit rate: fraction of memory access found in the upper level
  - Hit time: time to access the upper level
    - RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieved from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: time to replace a block in the upper level + time to deliver the block to the processor (latency + transmit time)
- Hit Time << Miss Penalty



4 Questions for Hierarchy Design

Q1: Where can a block be placed in the upper level?
=> *block placement*

Q2: How is a block found if it is in the upper level?
=> *block finding*

Q3: Which block should be replaced on a miss?
=> *block replacement*

Q4: What happens on a write?
=> *write strategy*

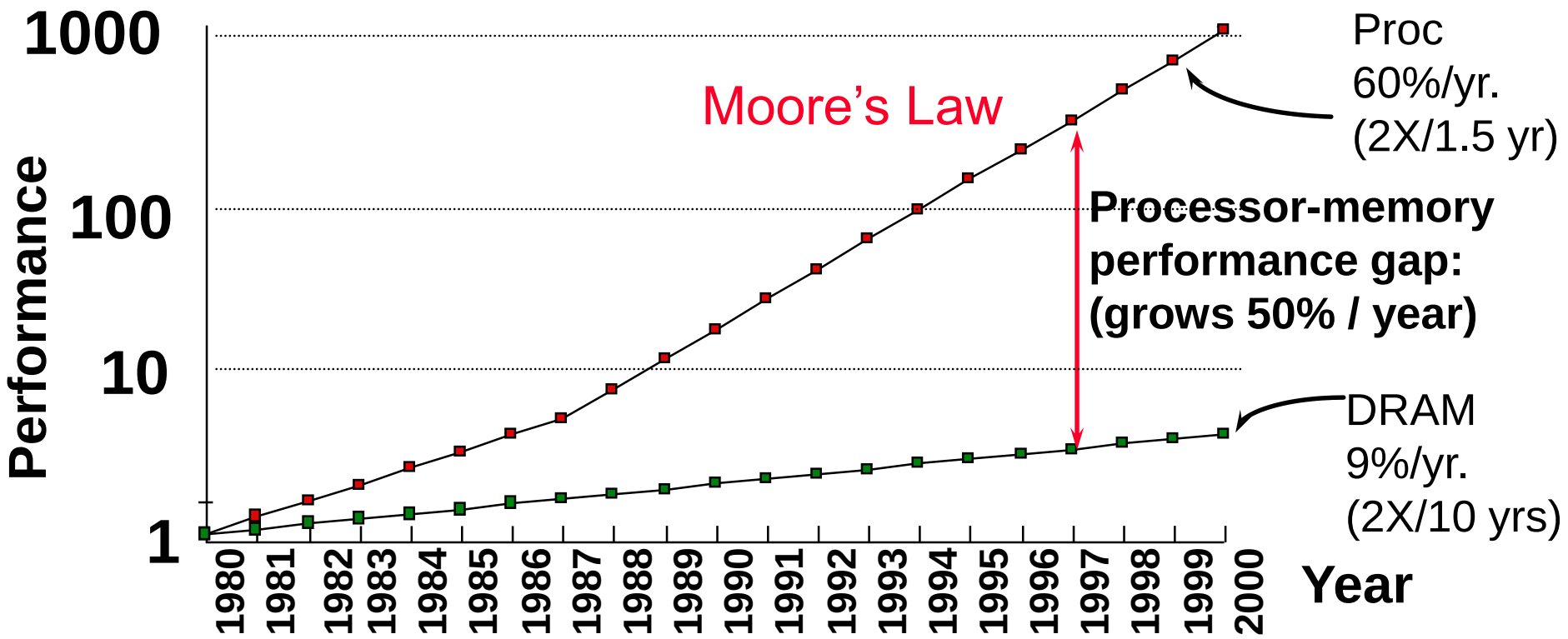
Summary of Memory Hierarchy

- Two different types of locality:
  - Temporal Locality (Locality in Time)
  - Spatial Locality (Locality in Space)
- Using the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.
- DRAM is slow but cheap and dense:
  - Good for presenting users with a BIG memory system
- SRAM is fast but expensive, not very dense:
  - Good choice for providing users FAST accesses

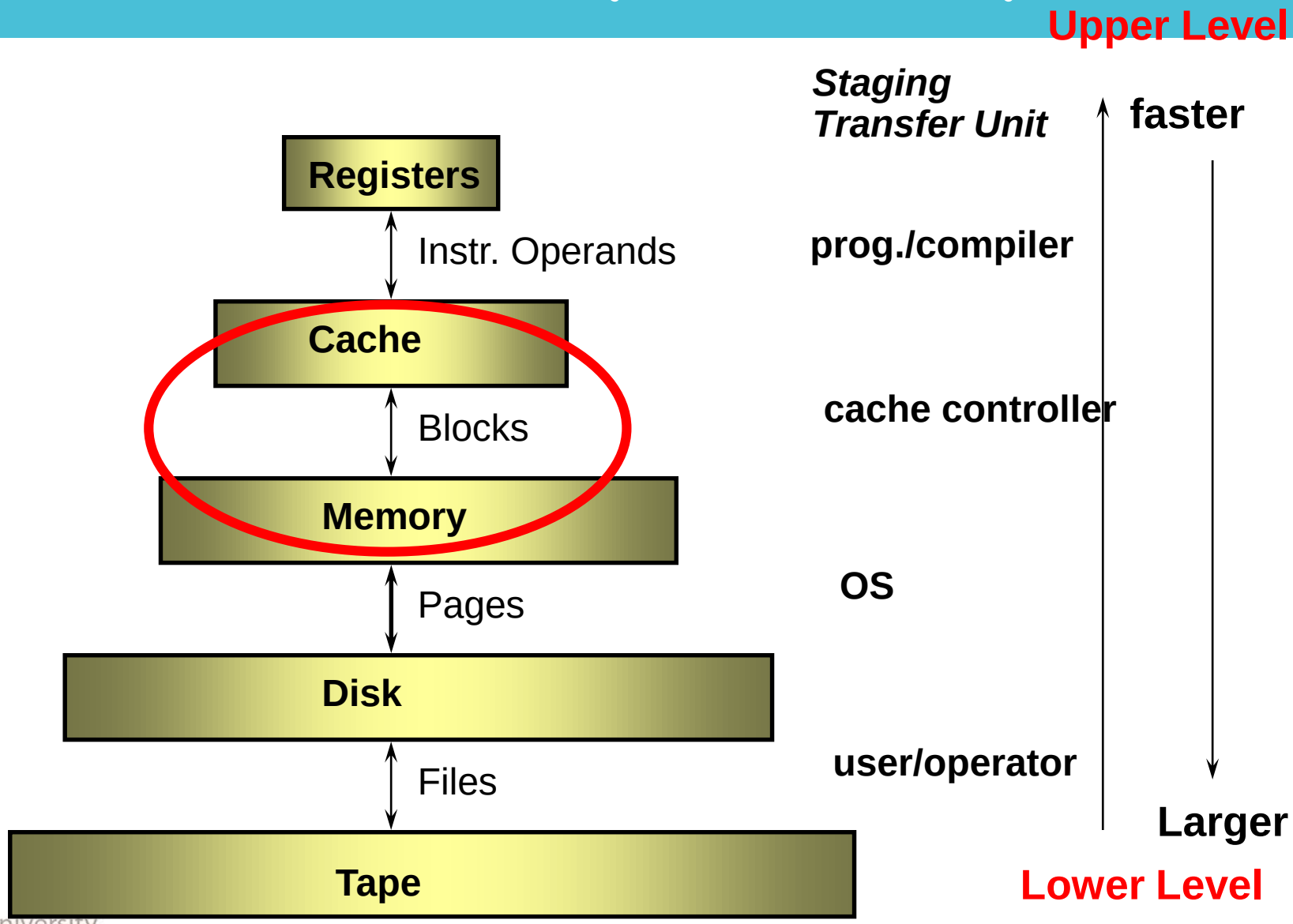
Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Processor Memory Latency Gap

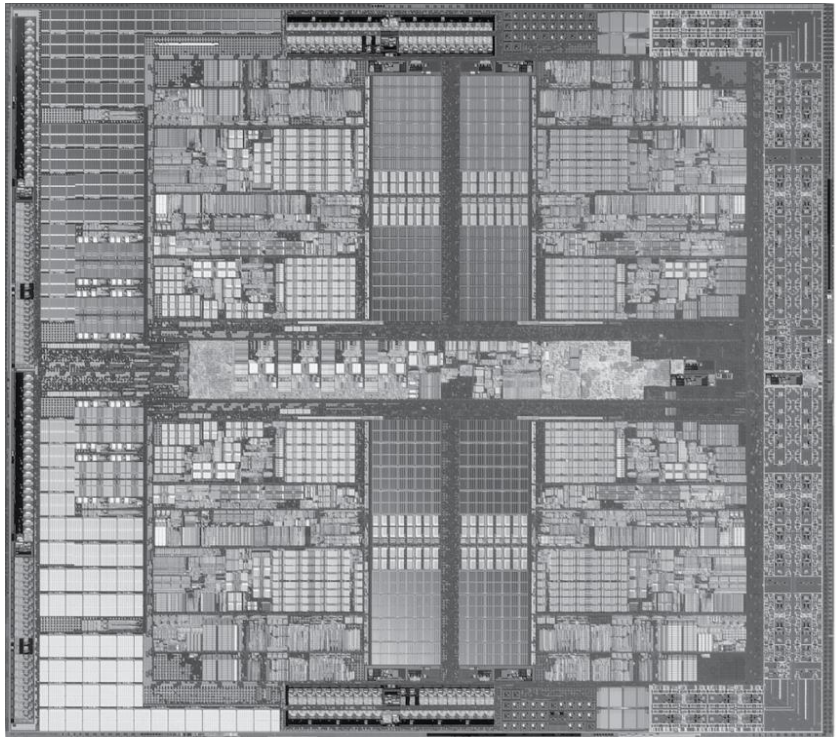


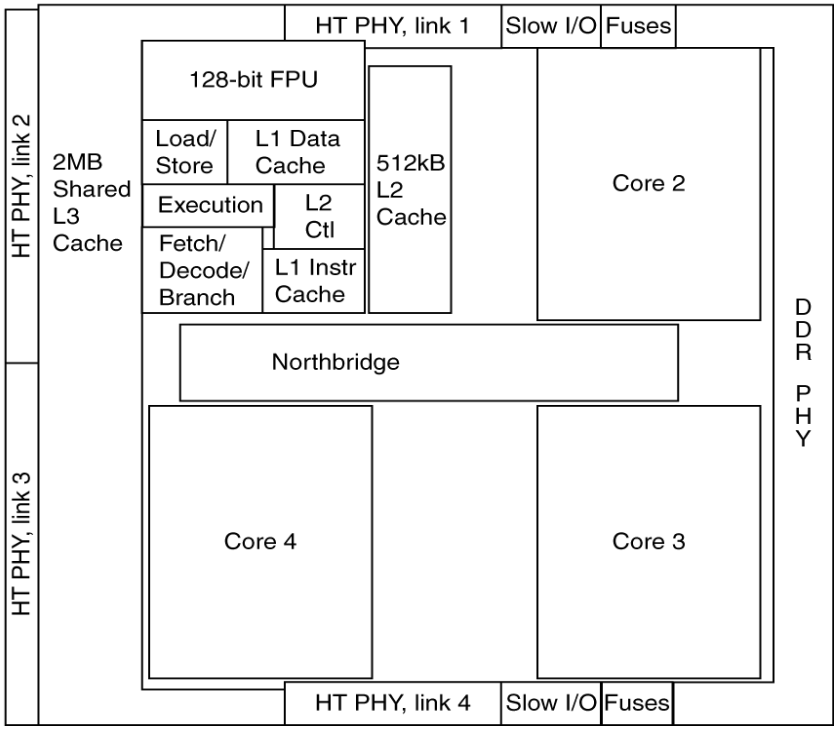
Levels of Memory Hierarchy



Inside the Processor

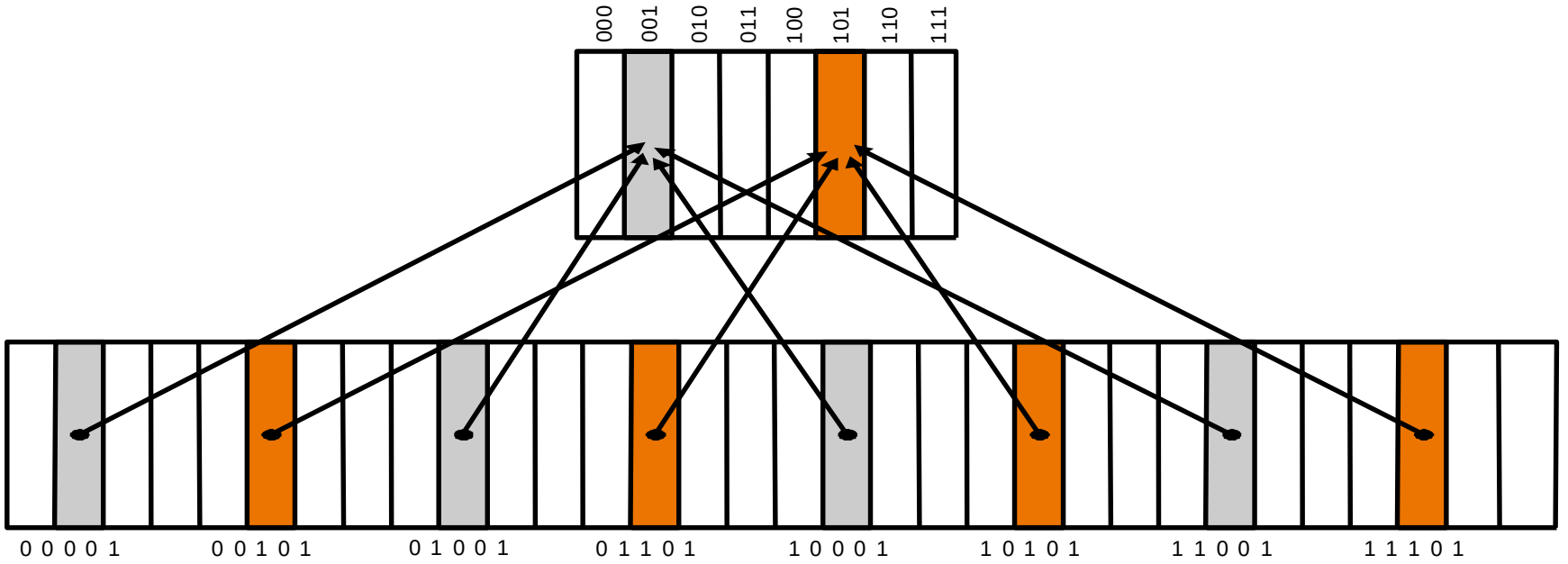
- **AMD Barcelona: 4 processor cores**





Basics of Cache

- Our first example: *direct-mapped cache*
- Block Placement :
  - For each item of data at the lower level, there is exactly one location in cache where it might be
  - Address mapping: modulo number of blocks



Tags and Valid Bits: Block Finding

- 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

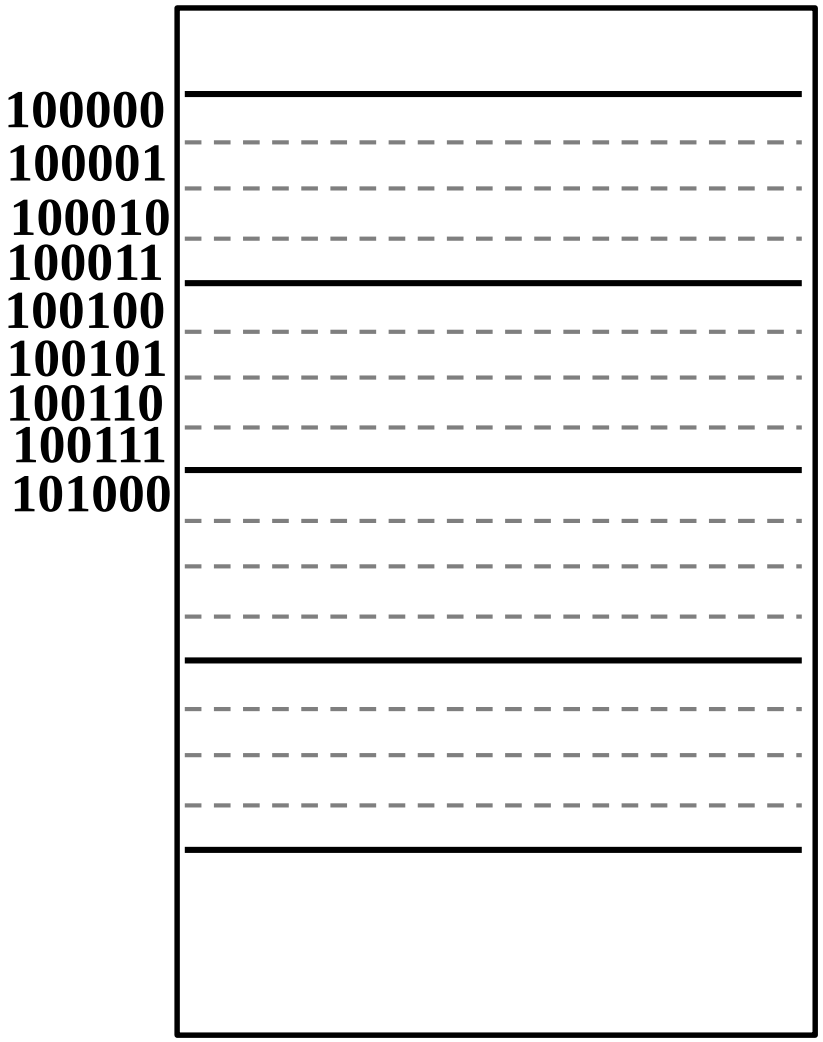
Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11->10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Outline

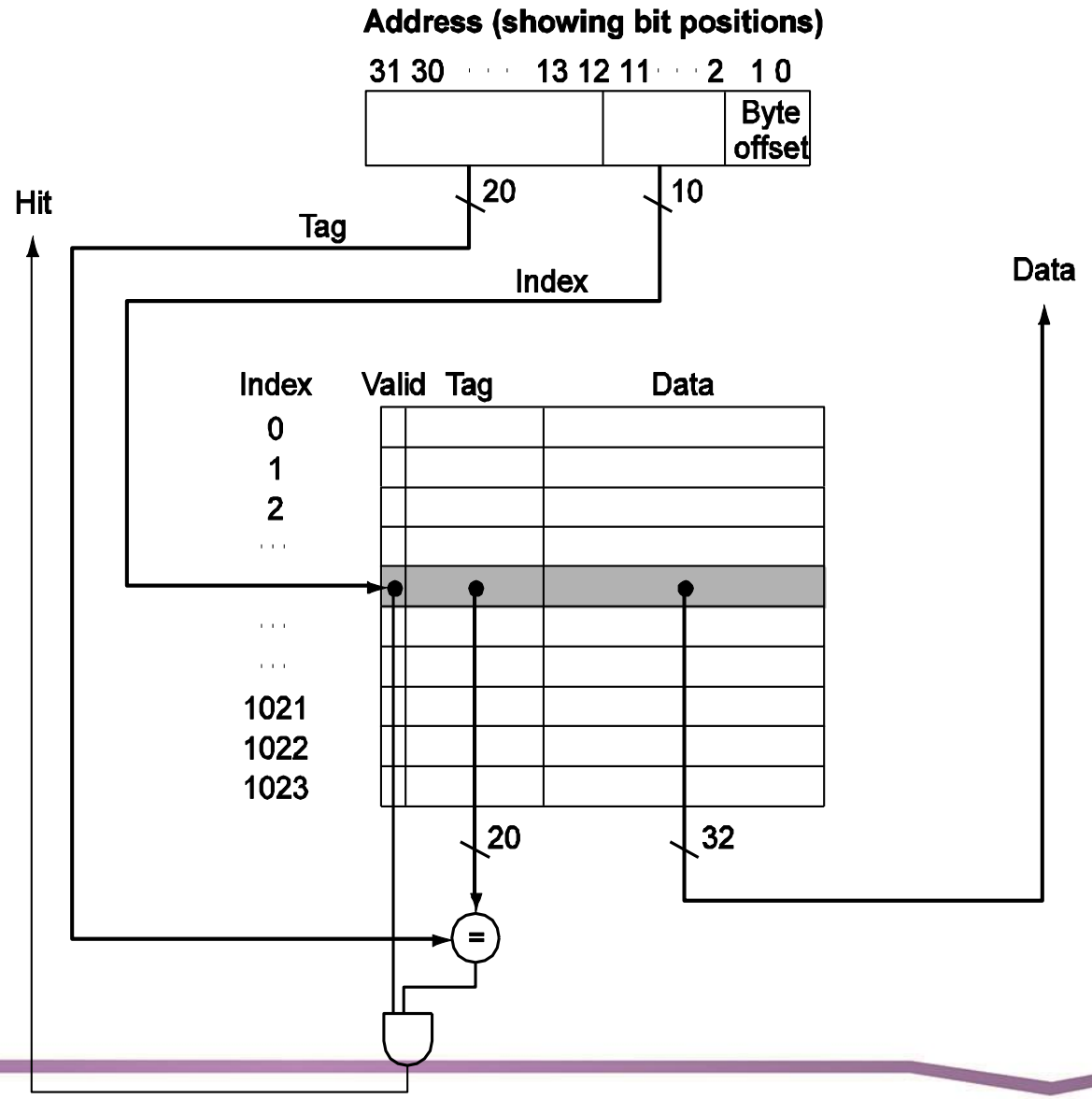
- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Memory Address



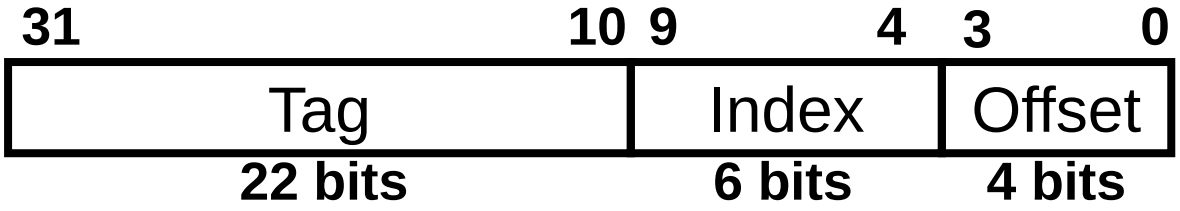
Address Subdivision

- Cache:
  1K words,
  1-word block:
  - Cache index: lower 10 bits
  - Cache tag: upper 20 bits
  - Valid bit (When start up, valid is 0)



Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
- Block address (memory) = ⌊1200/16⌋ = 75
- Block number (cache index) = 75 modulo 64 = 11
- $1200=00\ldots0010010110000_2 / 10000_2 \Rightarrow 00\ldots0010010111_2$
- $1001011_2 \Rightarrow 001011_2$

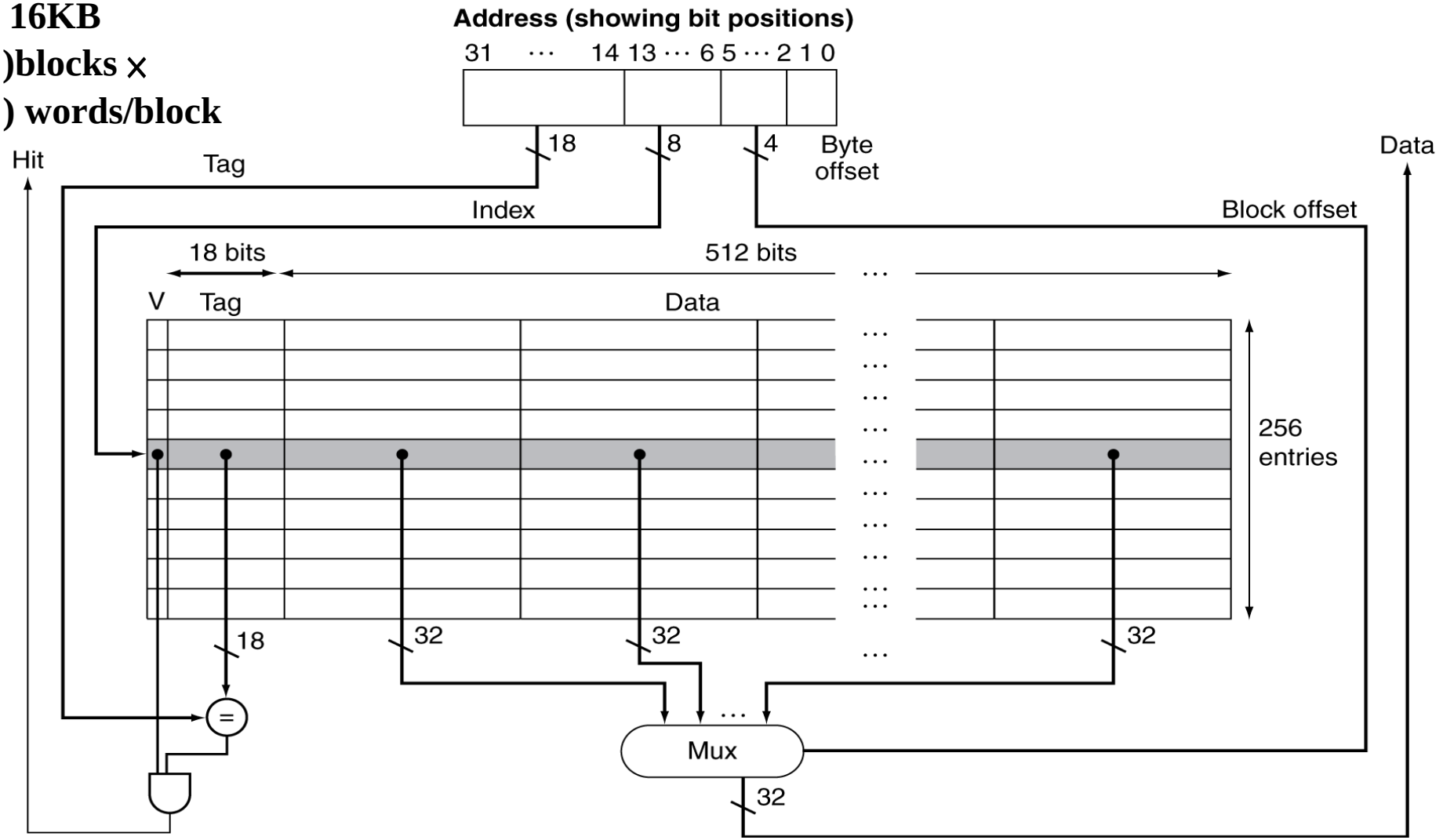


Example: Intrinsity FastMATH

- 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

Example: Intrinsity FastMATH

Cache: 16KB
256(2^8)blocks x
16 (2^4) words/block

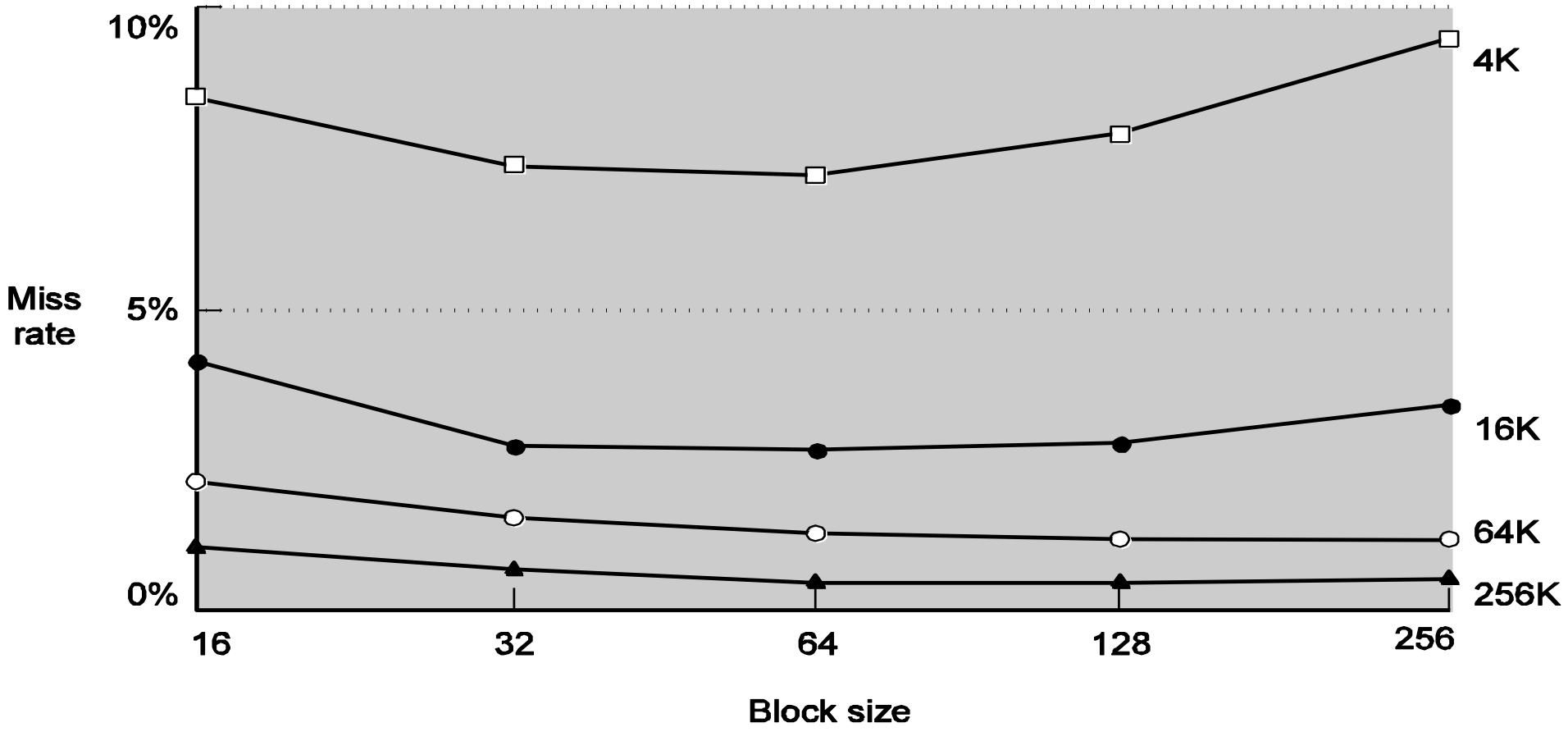


Block Size Considerations

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

Block Size on Performance

- Increase block size tends to decrease miss rate



Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Cache Hit and Miss

- Cache read
  - Read hit
  - Read miss
- Cache write
  - Write hit
    - Write-through
    - Write-back
  - Write miss

Read Cache

Read Hit:

- On cache hit, CPU proceeds normally

Read Miss:

- 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

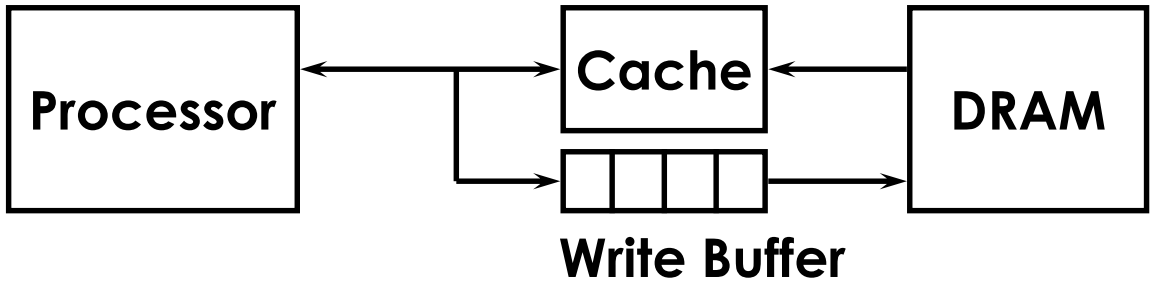
Write Hit: Write-Through

There are two copies of data: one in cache and one in memory.

Write Hit:

- Write through: also update memory
- Increase the traffic to memory
- Also makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 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

Avoid Waiting for Memory in Write Through



- Use a *write buffer* (WB):
  - Processor: writes data into cache and WB
  - Memory controller: write WB data to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
- Memory system designer's nightmare:
  - Store frequency > 1 / DRAM write cycle
  - Write buffer saturation => CPU stalled

Write Hit: 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
- Data in cache and memory is inconsistent

Write Miss: Write Allocation

Write Miss:

- **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

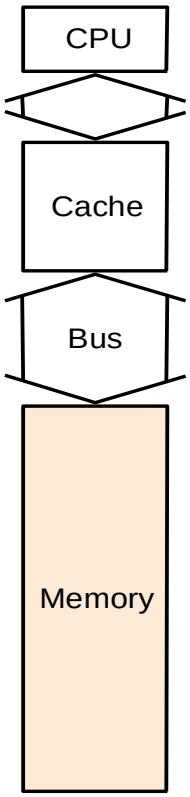
- Embedded MIPS processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: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

Outline

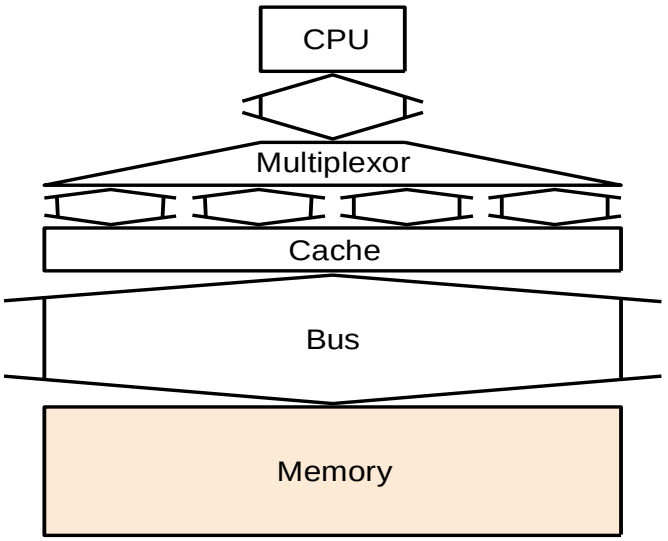
- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Memory Design to Support Cache

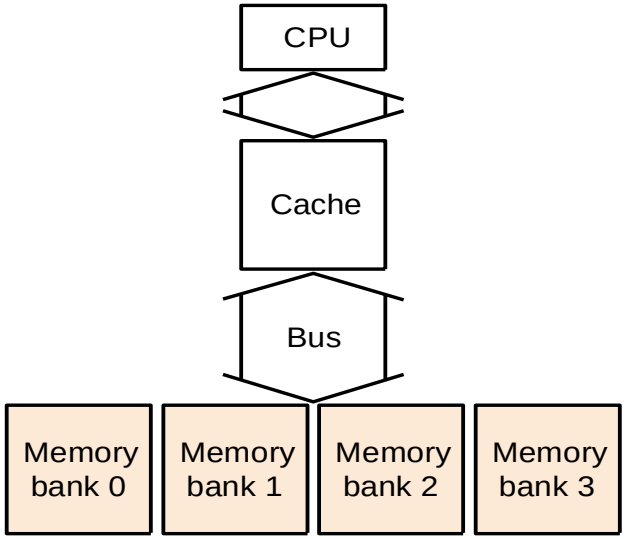
- **How to increase memory bandwidth to reduce miss penalty?**



a. One-word-wide memory organization



b. Wide memory organization

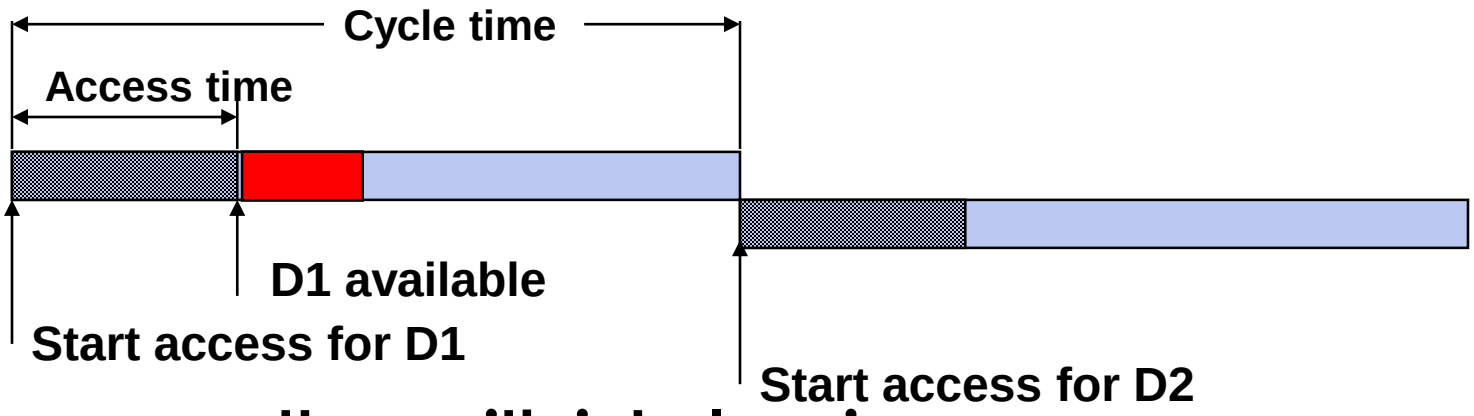


c. Interleaved memory organization

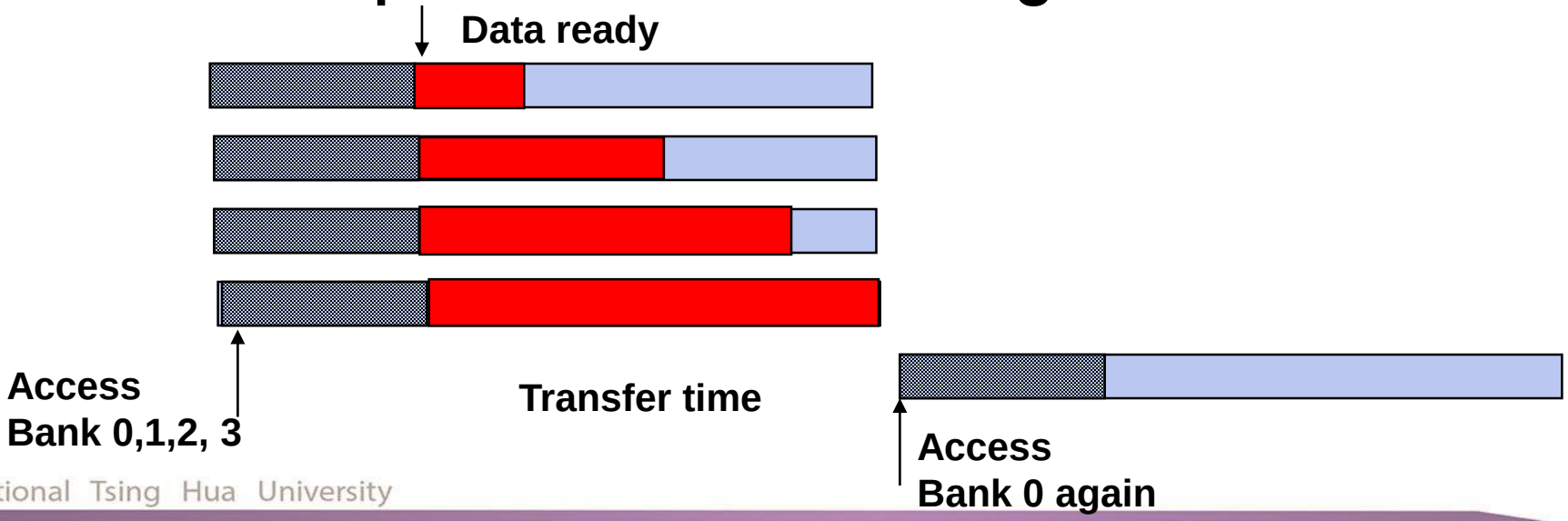
Fig. 5.11

Interleaving for Bandwidth

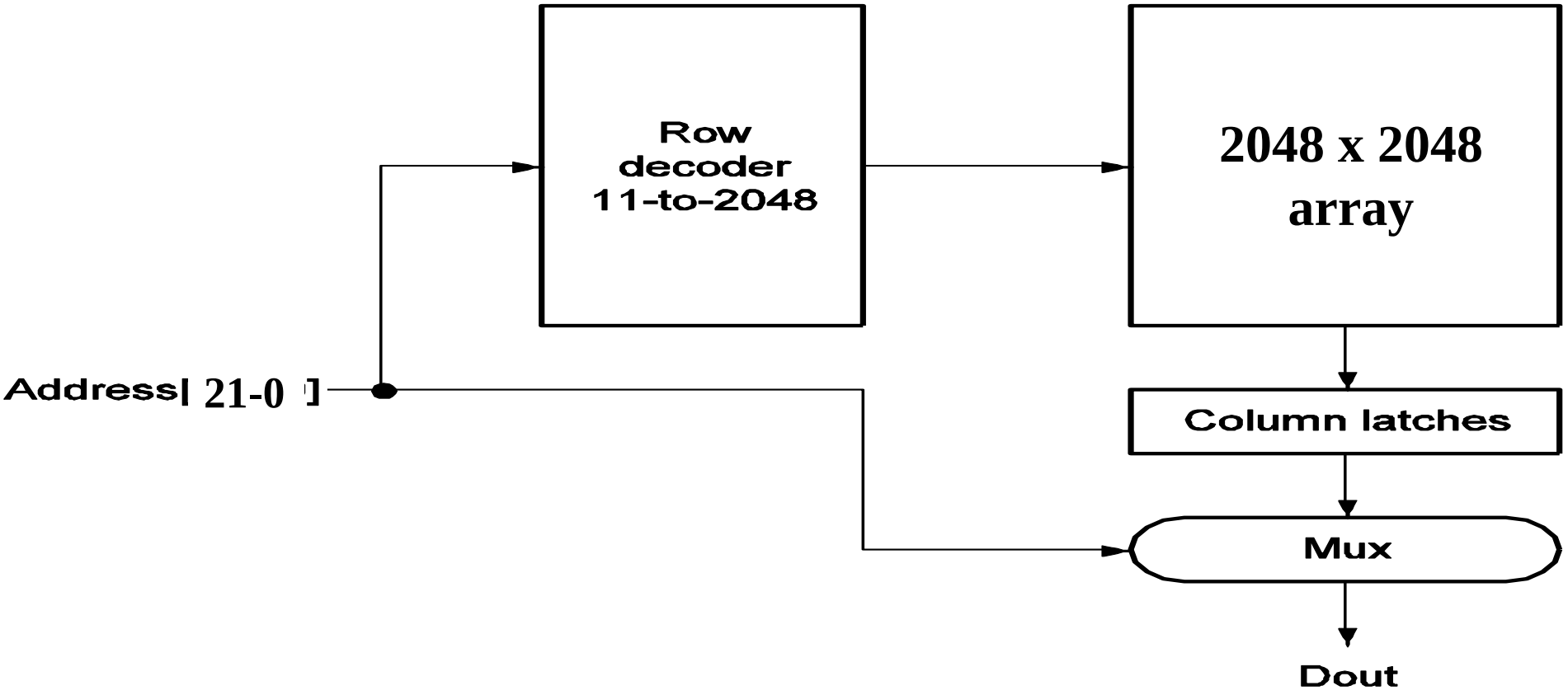
- Access pattern without interleaving:



- Access pattern with interleaving

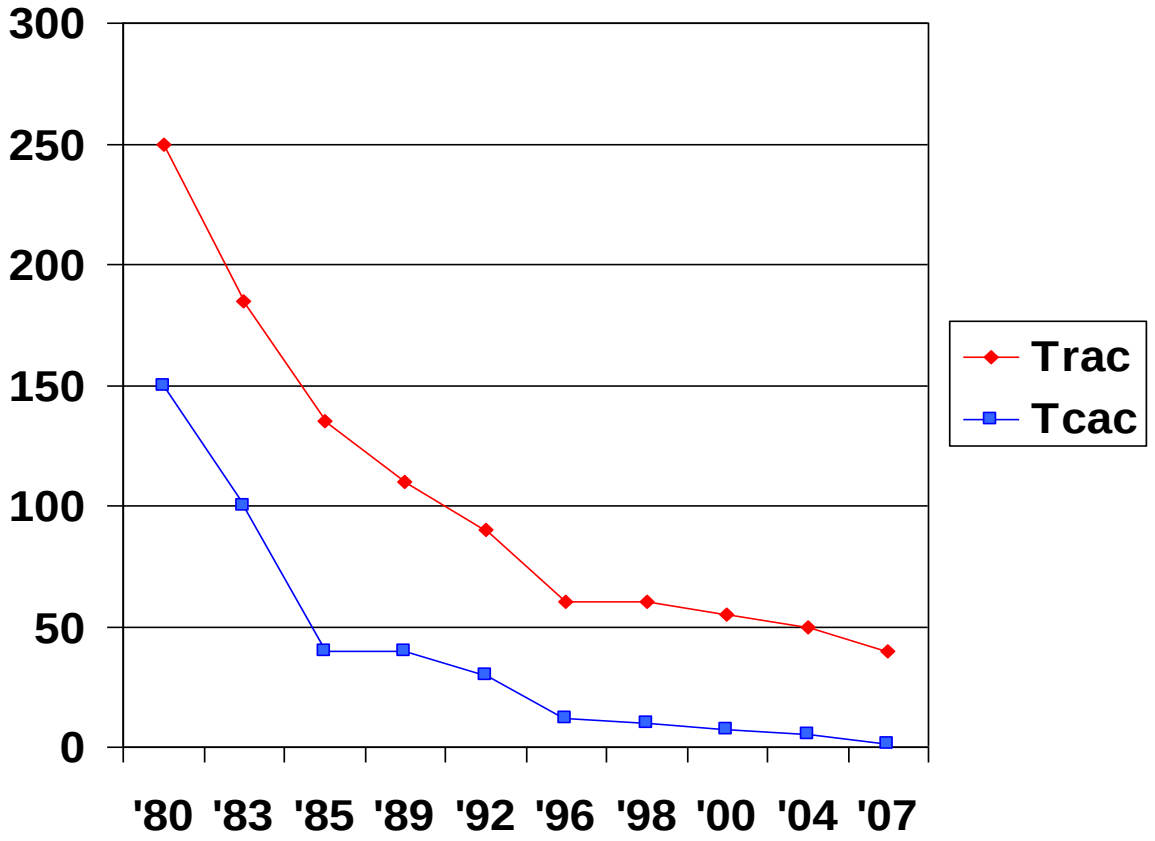


Access of DRAM



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



Trac:access time to a new row
Tcac:column access time to existing row

Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Measuring Cache Performance

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

$$\text{Memory stall cycles}$$

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

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

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
  - Compute the actual CPI?

- 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
- 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

Improving Cache Performance

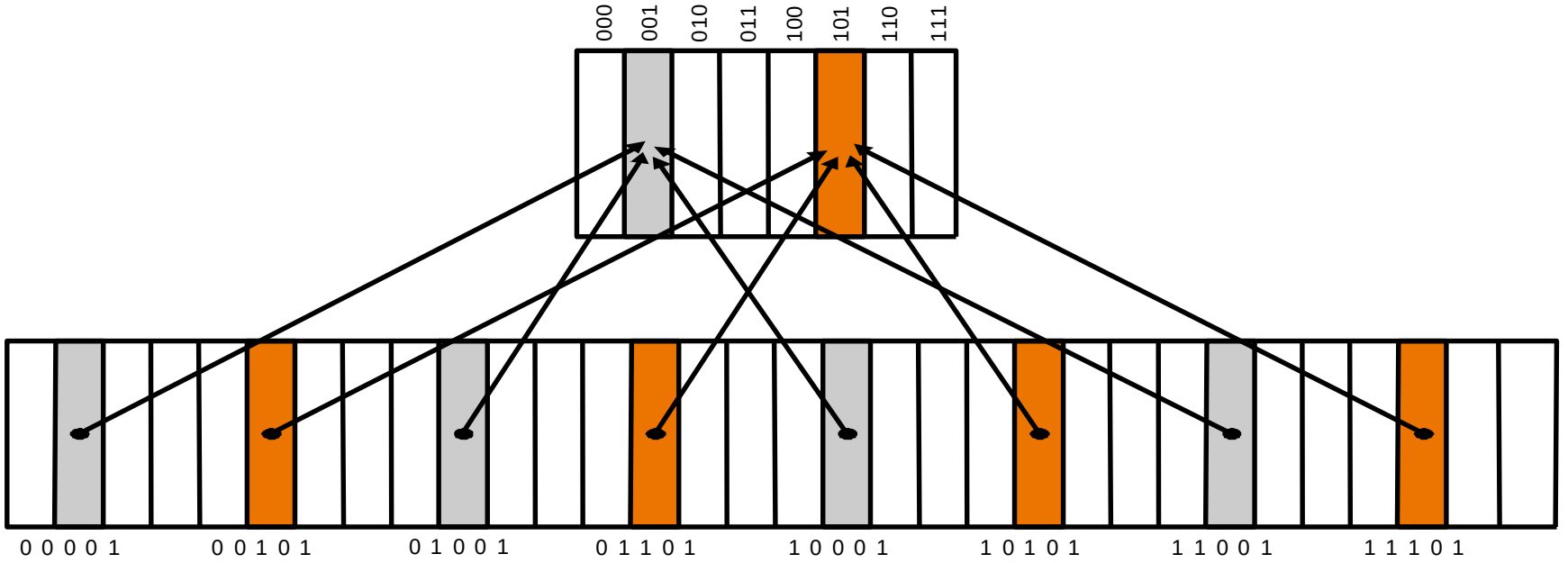
- Reduce the time to hit in the cache
- Decreasing the miss ratio
- Decreasing the miss penalty

Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Direct Mapped

- **Block Placement :**
  - **For each item of data at the lower level, there is exactly one location in cache where it might be**
  - **Address mapping: modulo number of blocks**

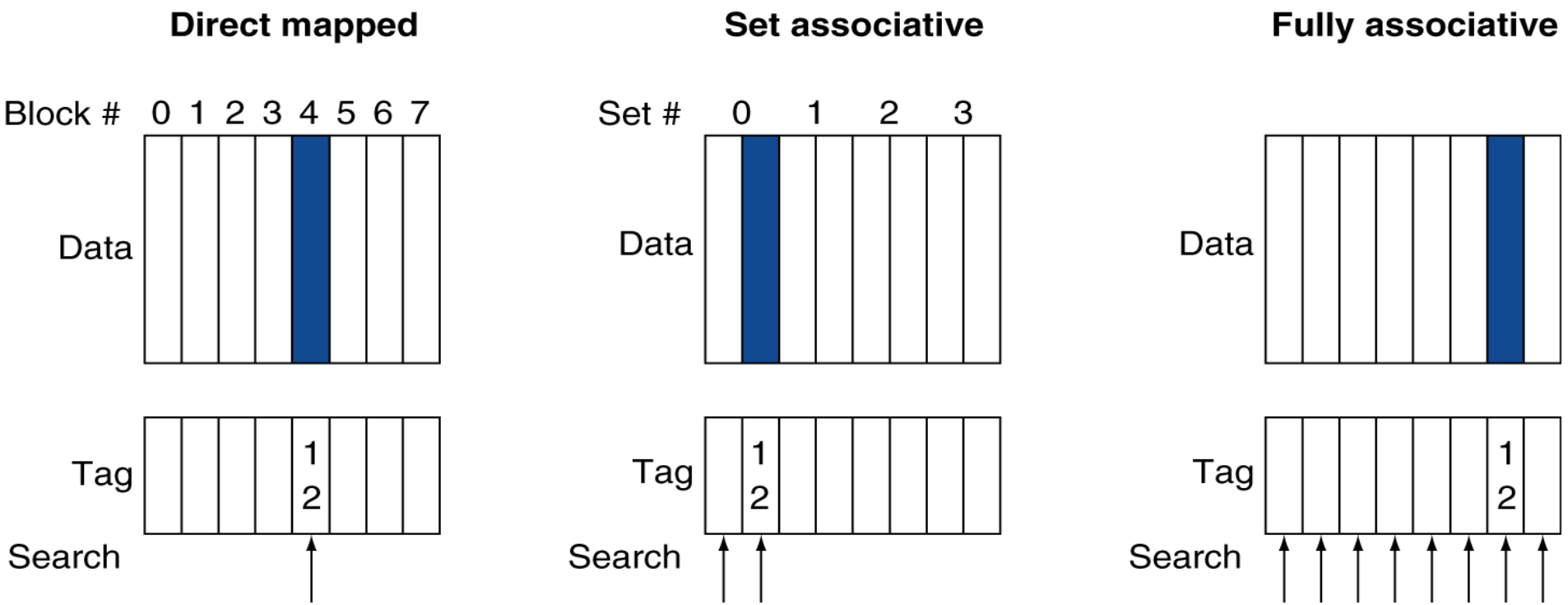


Associative Caches

- 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

- **Placement of a block whose address is 12:**



Possible Associativity Structures

An 8-block cache

(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

Associativity Example

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

time ↓

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

time ↓

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

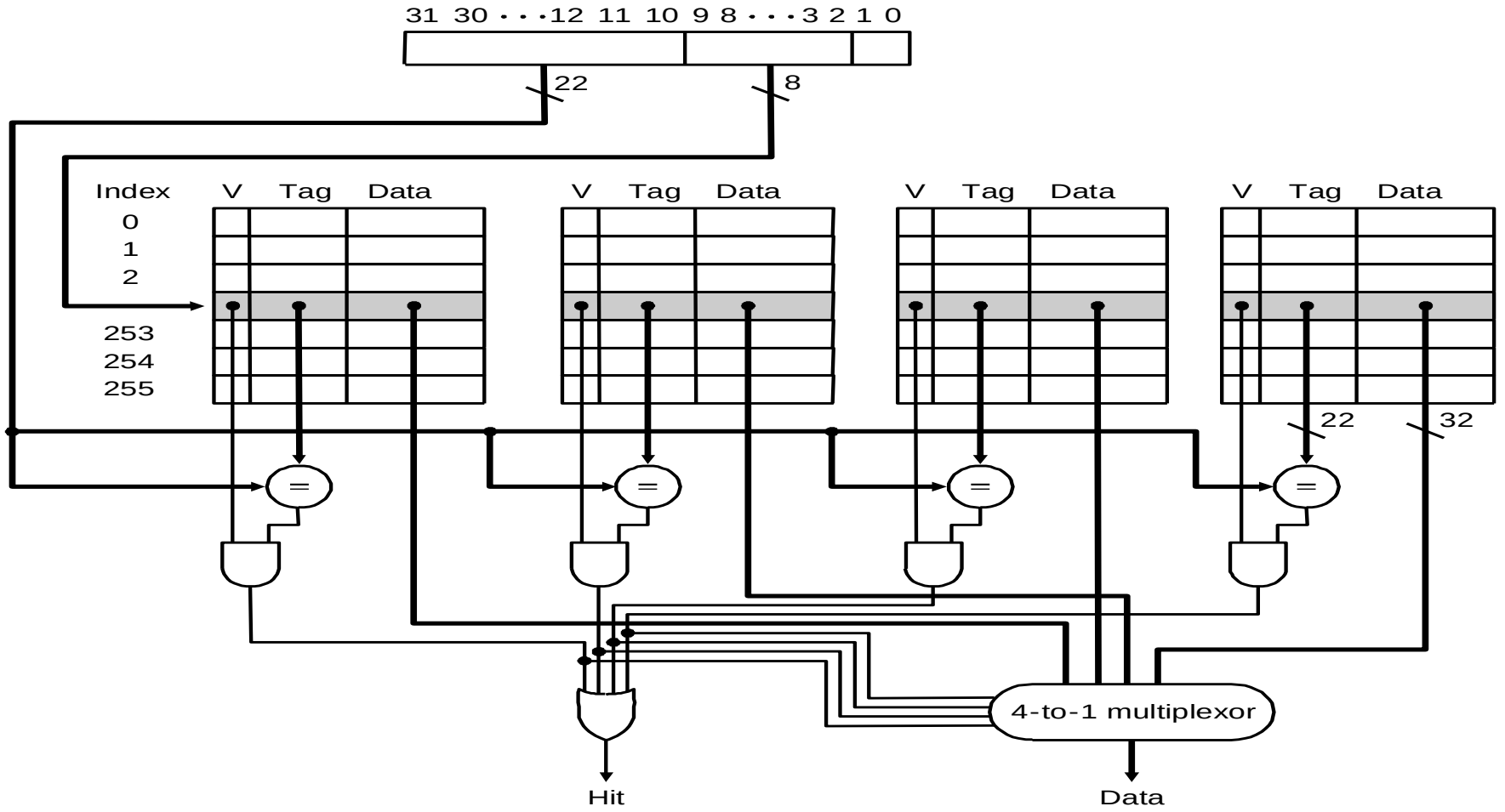
time ↓

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]	

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%

A 4-Way Set-Associative Cache



- **Increasing associativity shrinks index, expands tag**

Comparing Data Available Time

- Direct mapped cache
  - Cache block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue, recover later if miss

- N-way set-associative cache
  - Data comes AFTER Hit/Miss decision
  - Extra MUX delay for the data

Data Placement Policy

- Direct mapped cache:
  - Each memory block mapped to one location
  - No need to make any decision
  - Current item replaces previous one in location
- N-way set associative cache:
  - Each memory block has choice of N locations
- Fully associative cache:
  - Each memory block can be placed in ANY cache location
- Misses in N-way set-associative or fully associative cache:
  - Bring in new block from memory
  - Throw out a block to make room for new block
  - Need to decide on which block to throw out

Cache Block Replacement

- Direct mapped: no choice
- Set associative or fully associative:
  - Random
  - LRU (Least Recently Used):
    - Hardware keeps track of the access history and replace the block that has not been used for the longest time
  - An example of a pseudo LRU (for a two-way set associative) :
    - use a pointer pointing at each block in turn
    - whenever an access to the block the pointer is pointing at, move the pointer to the next block
    - when need to replace, replace the block currently pointed at

Comparing the Structures

- N-way set-associative cache
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection
- Direct mapped cache
  - Cache block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue, recover later if miss

Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

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
  - L-1 miss rate to L-2 = 2% (with one cache: to M)
  - L-2 access time = 5ns (to M: 100ns)
  - L-2 miss rate to main memory = 0.5%
- Primary miss with L-2 hit (2%)
  - Penalty to access L-2 = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss (0.5%)
  - Penalty to access memory = 400 cycles
- CPI = $1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance ratio = 9/3.4 = 2.6

Multilevel Cache Considerations

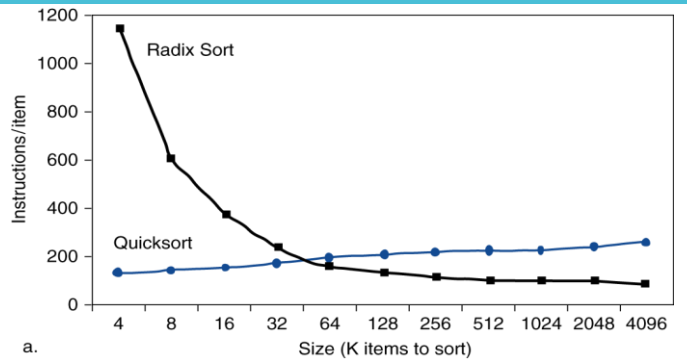
- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single-level cache
  - L-1 block size smaller than L-2 block size

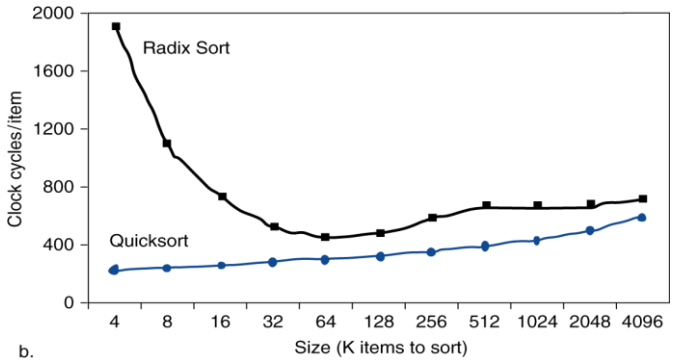
Interactions with Advanced CPUs

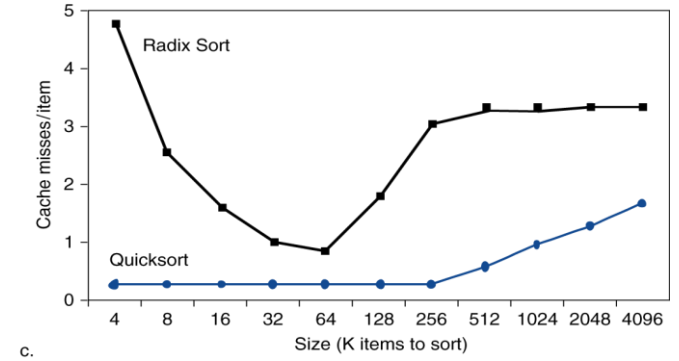
- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyze
  - Use system simulation

Interactions with Software

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



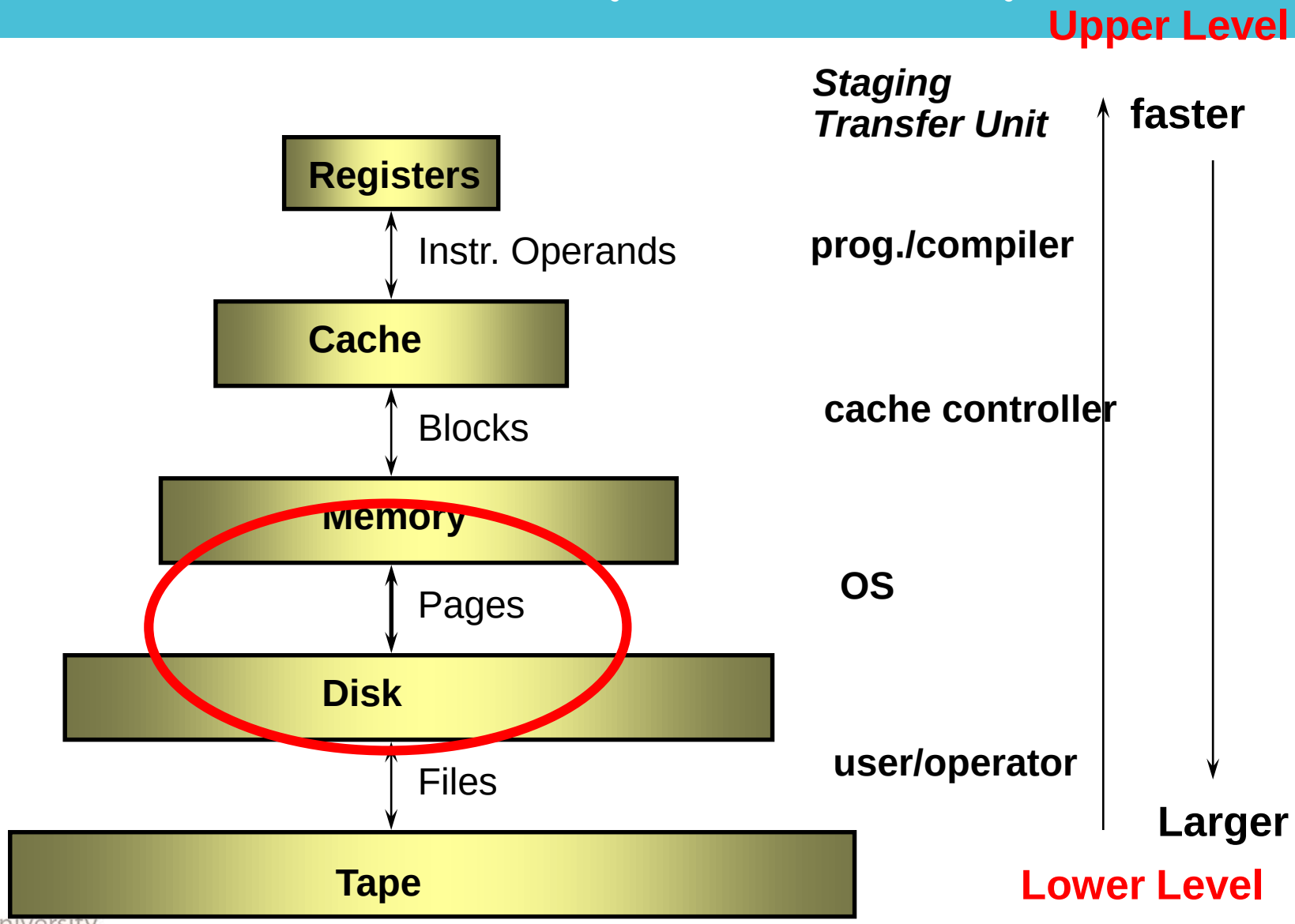




Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Issues in virtual memory
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Levels of Memory Hierarchy



Virtual Memory

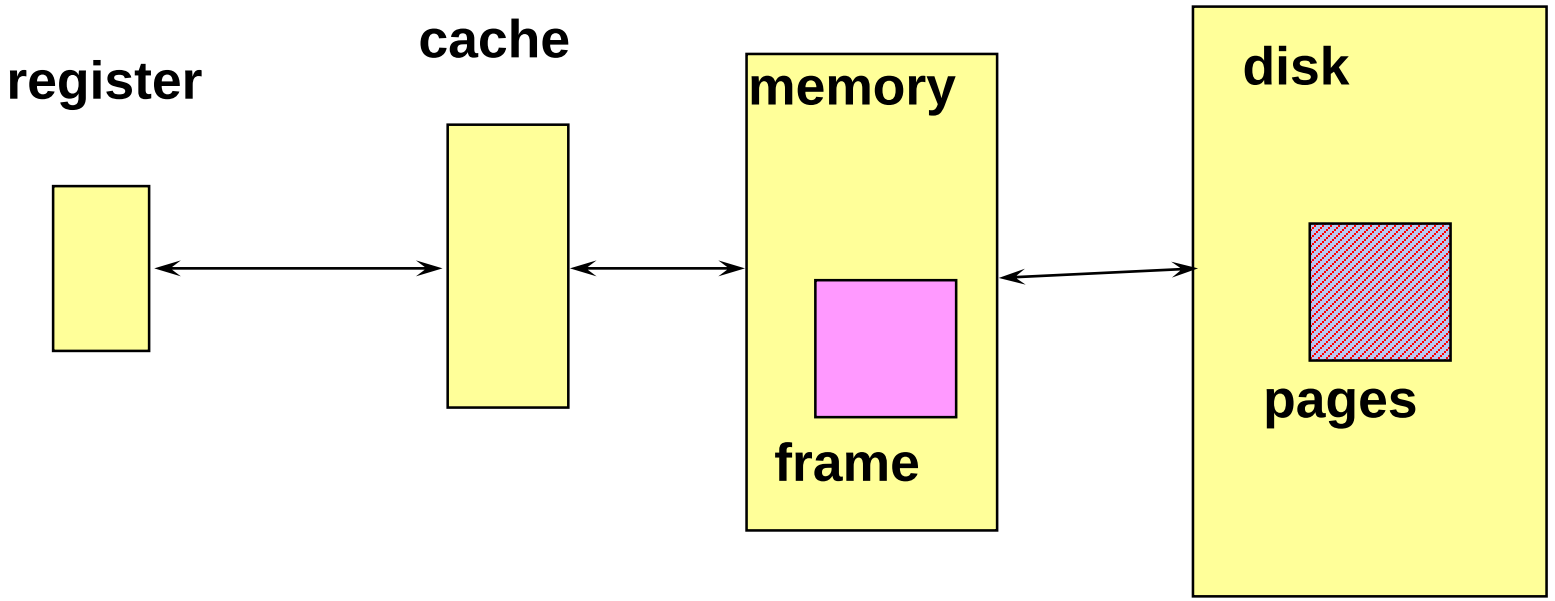
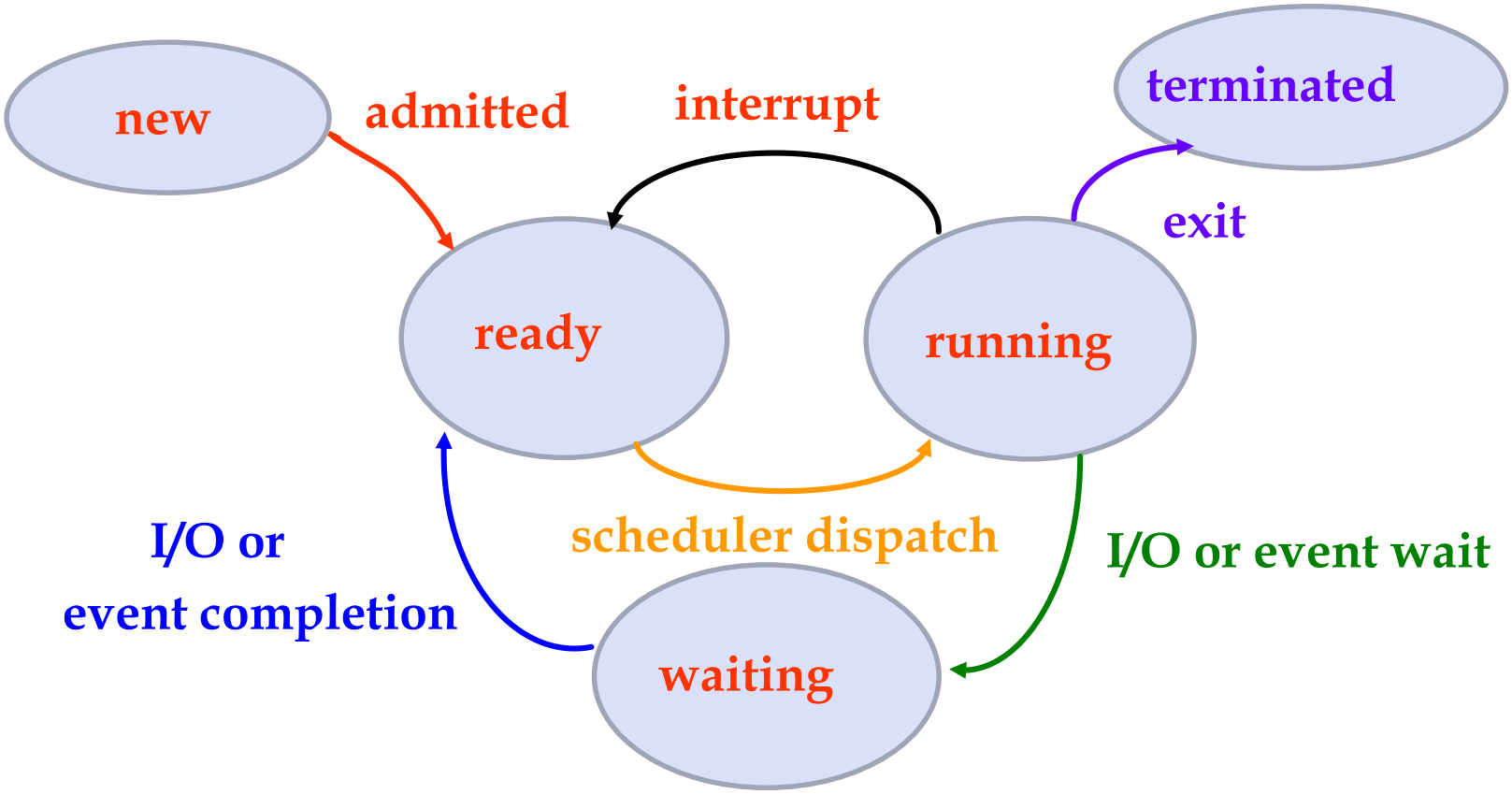
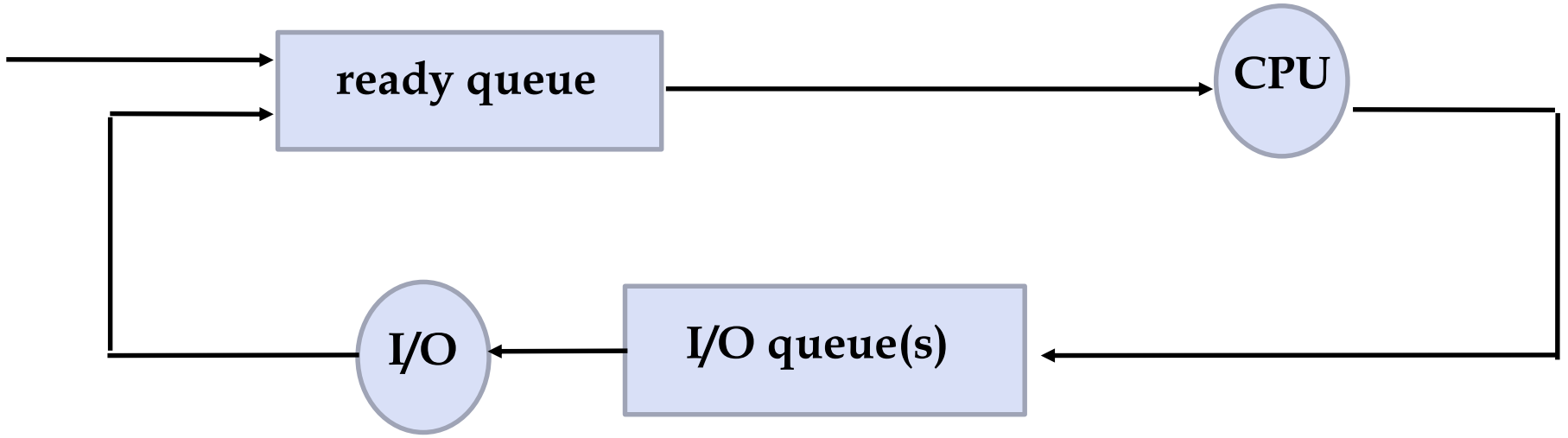


Diagram of Process State



CPU Scheduler

- **selecting** one of the ready processes, and allocates the CPU to it.



Virtual Memory

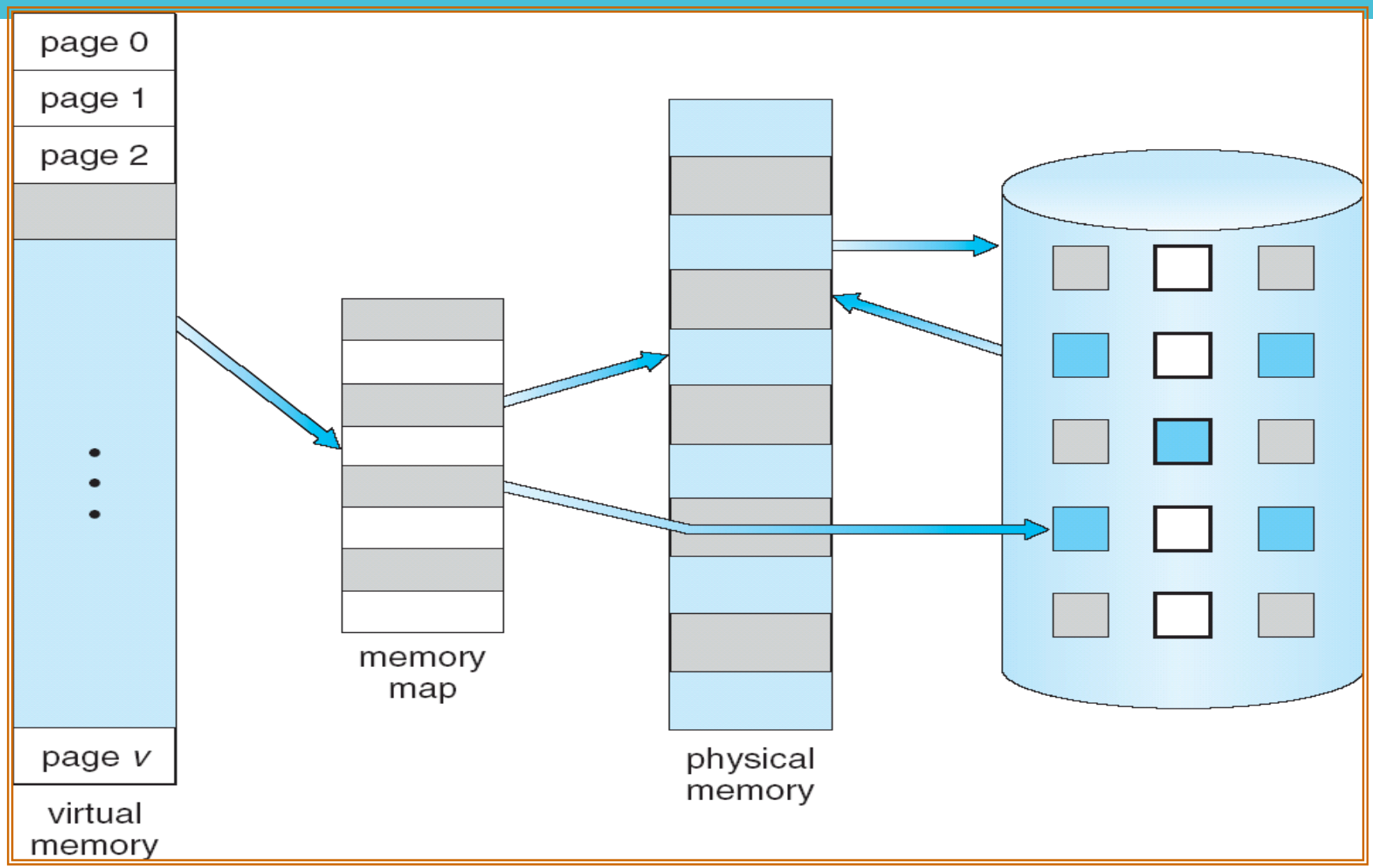
- Programs share main memory (Multi-programming and I/O)
  - Program executed in a name space (virtual address space) different from memory space (physical address space)
  - Each gets a private virtual address space holding its frequently used code and data, starting at address 0, only accessible to itself
    - yet, any can run anywhere in physical memory
    - virtual memory implements the translation from virtual space to physical space
  - Every program has lots of memory (> physical memory)
  - Protected from other programs

- Use main memory as a "cache" for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)

Virtual Memory - Continued

- 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

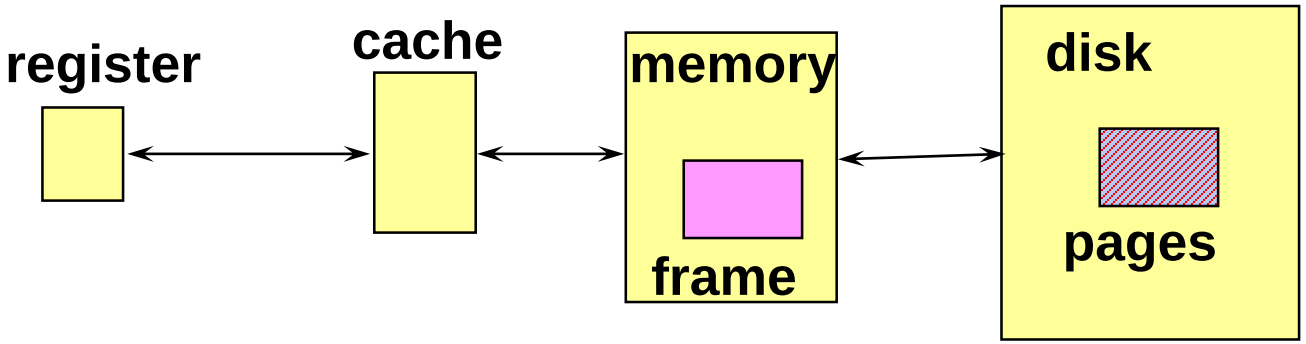


Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Issues in virtual memory
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Basic Issues in Virtual Memory

- Size of data blocks that are transferred from disk to main memory
- Which region of memory to hold new block
  => Placement policy (how to find the page?)
- When memory is full, then some region of memory must be released to make room for the new block
  => replacement policy
- When to fetch missing items from disk?
  - Fetch only on a fault => demand load policy

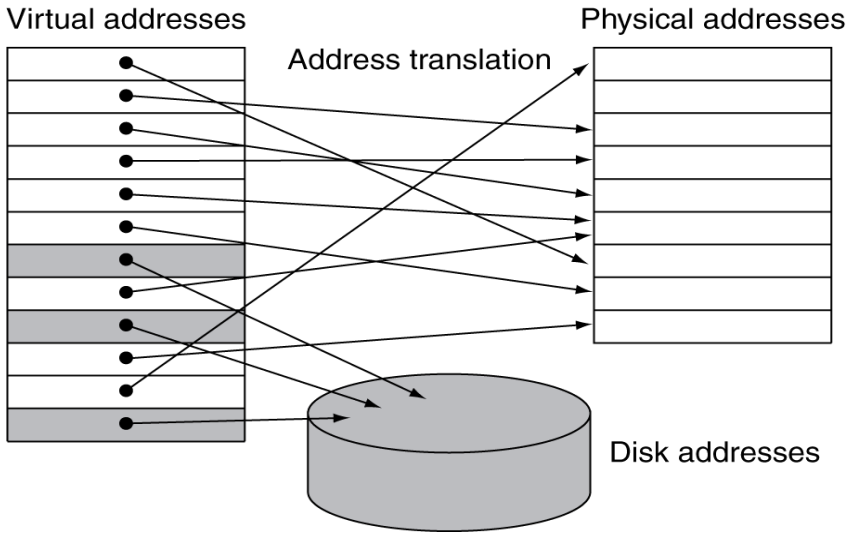


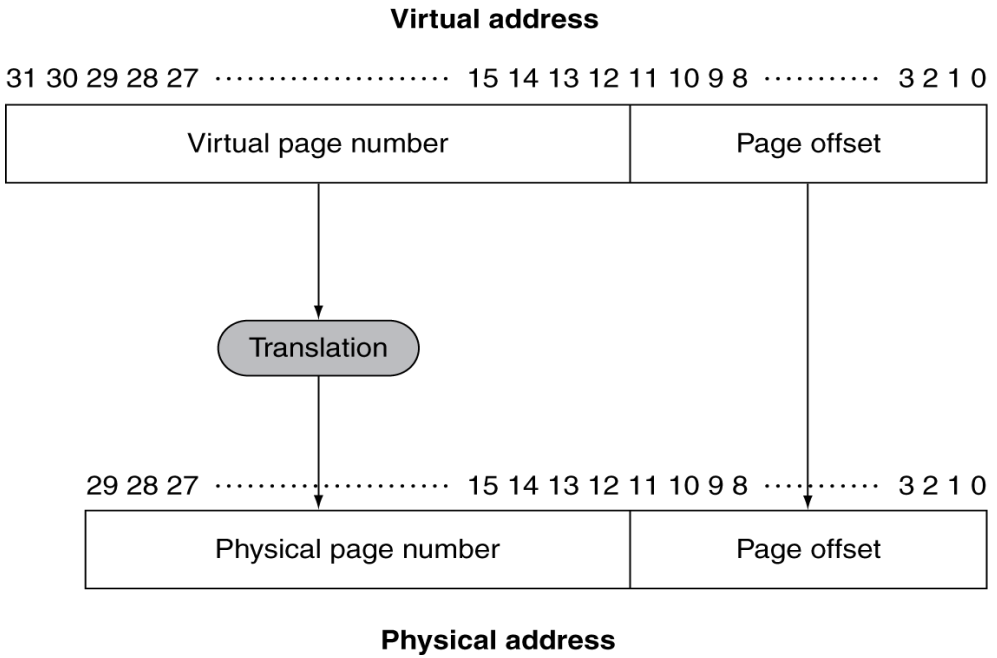
Block Size and Placement Policy

- Huge miss penalty: a page fault may take millions of cycles to process
  - Pages should be fairly large (e.g., 4KB) to amortize the high access time
  - Reducing page faults is important
    - fully associative placement
      => use page table (in memory) to locate pages

Address Translation

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





Page Tables

- Stores placement information
  - Page table stored in main memory
  - 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

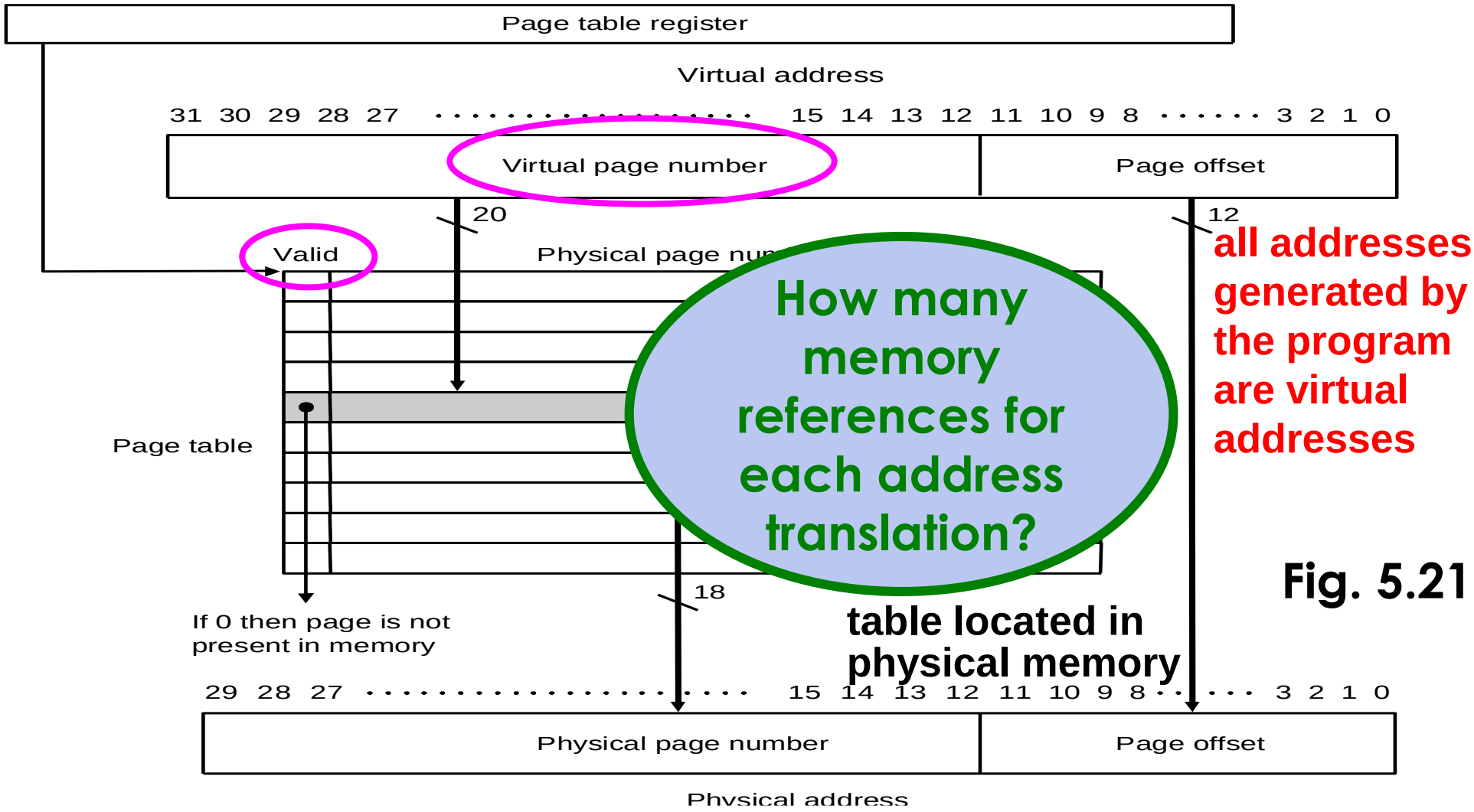


Fig. 5.21

Page Fault: What Happens When You Miss?

- Page hit, proceed normally
- Page fault means that page is not resident in memory
- Huge miss penalty: a page fault may take millions of cycles to process
- Hardware must detect situation but it cannot remedy the situation
- Can handle the faults in software instead of hardware, because handling time is small compared to disk access
  - the software can be very smart or complex
  - the faulting process can be context-switched

OS Handling Page Faults

- Hardware must trap to the operating system so that it can remedy the situation
  - Pick a page to discard (may write it to disk)
  - Load the page in from disk
  - Update the page table
  - Resume to program so HW will retry and succeed!

- In addition, OS must know where to find the page
  - Create space on disk for all pages of process (swap space)
  - Use a data structure to record where each valid page is on disk (may be part of page table)
  - Use another data structure to track which process and virtual addresses use each physical page
    => for replacement purpose

When to Fetch Missing Items From Disk

- **Fetch only on a fault**
  => Demand load policy

Page 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

Problems of Page Table

- Page table is too big
- Access to page table is too slow (needs one memory read)

Problems of Page Table

- Page table is too big
- Access to page table is too slow (needs one memory read)

Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Issues in virtual memory
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Impact of Paging: Huge Page Table

- Page table occupies storage
  32-bit VA, 4KB page, 4bytes/entry
  => 2^{20} PTE, 4MB table
- Possible solutions:
  - Use bounds register to limit table size; add more if exceed
  - Let pages to grow in both directions
    => 2 tables, 2 limit registers, one for hash, one for stack
  - Use hashing => page table same size as physical pages
  - Multiple levels of page tables
  - Paged page table (page table resides in virtual space)

Problems of Page Table

- **Page table is too big**
- **Access to page table is too slow (needs one memory read)**

Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Issues in virtual memory
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Impact of Paging: More Memory Access !

- Each memory operation (instruction fetch, load, store) requires a page-table access!
  - Basically double number of memory operations
  - One to access the PTE
  - Then the actual memory access
- access to page tables has good locality

Fast Translation Using a TLB

- Access to page tables has good locality
  - Fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)

Fast Translation Using TLB (Translation Lookaside Buffer)

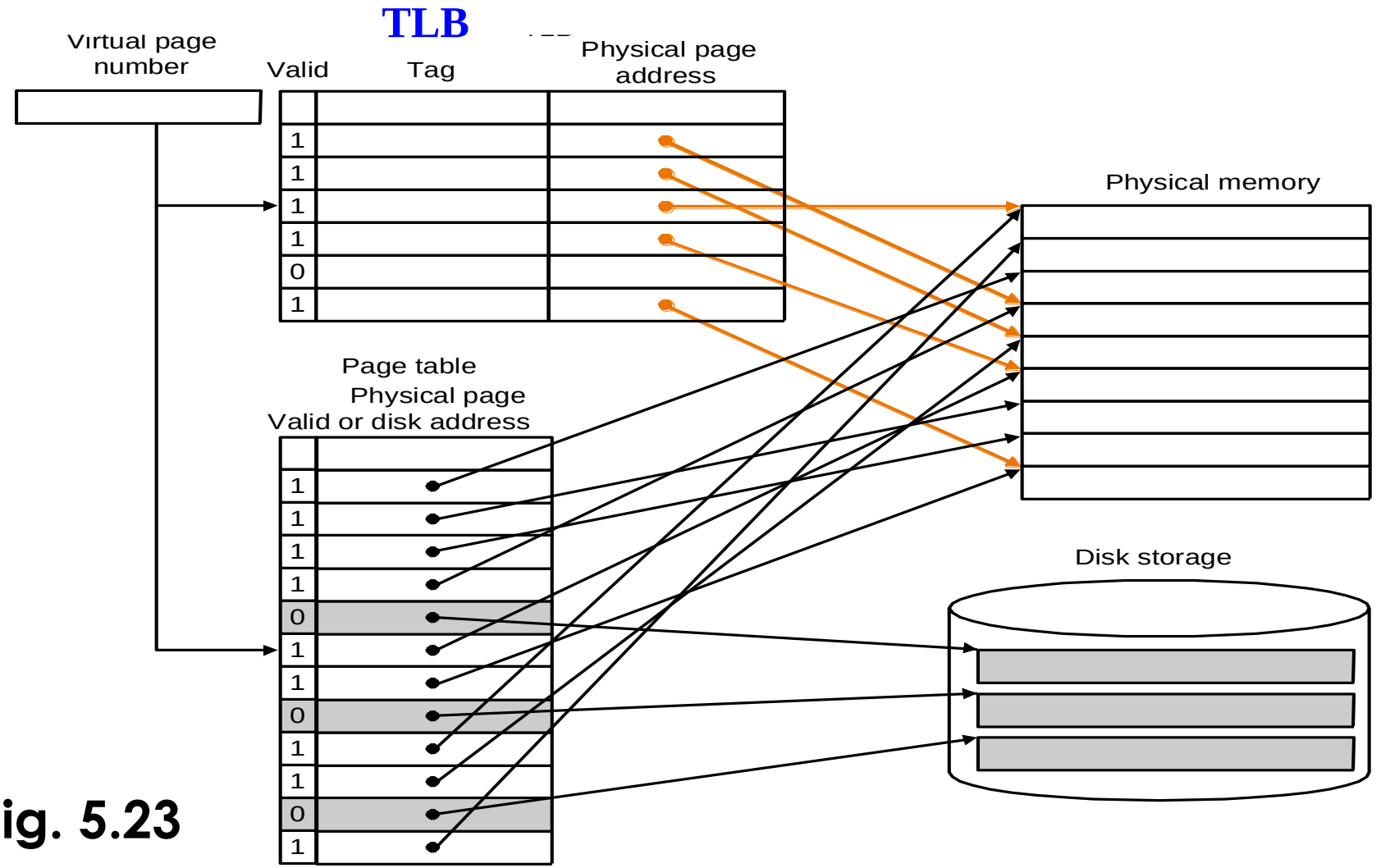


Fig. 5.23

Translation Lookaside Buffer

- Typical RISC processors have *memory management unit* (MMU) which includes TLB and does page table lookup
  - TLB can be organized as fully associative, set associative, or direct mapped
  - TLBs are small, 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

TLB Hit

- TLB hit on read
- TLB hit on write:
  - Toggle dirty bit (write back to page table on replacement)

TLB Miss

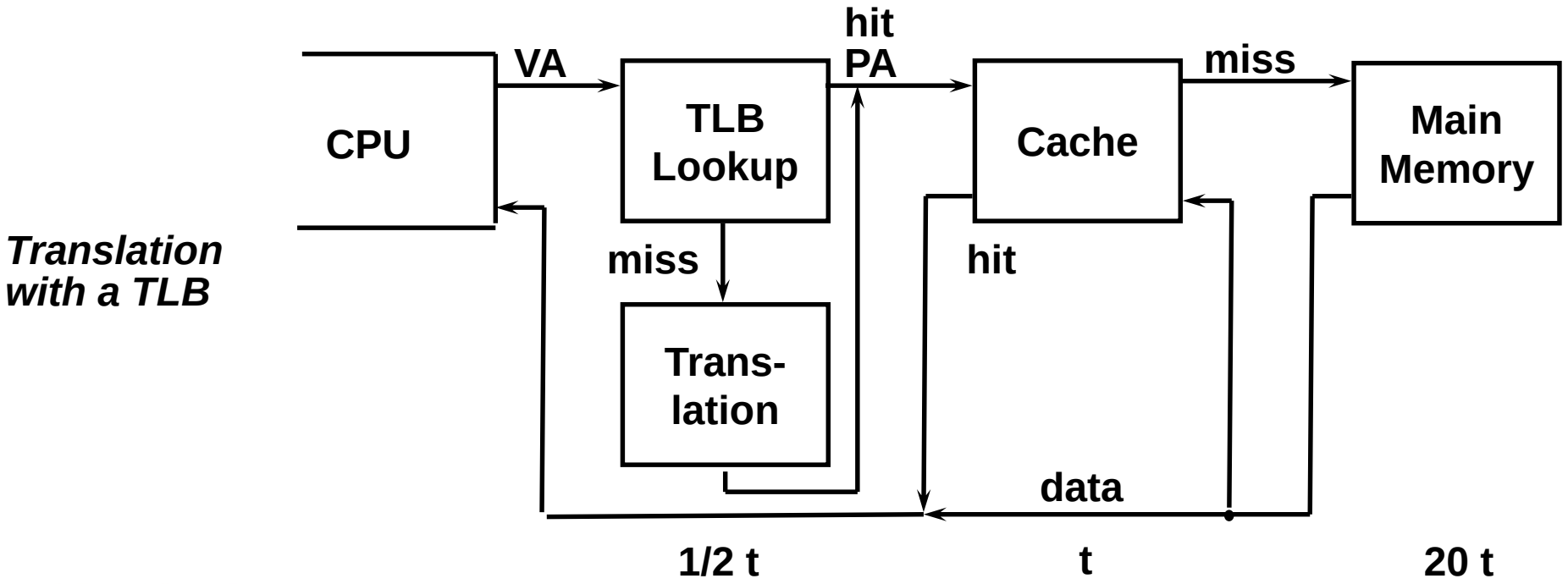
- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the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

Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Issues in virtual memory
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

Making Address Translation Practical

- In VM, memory acts like a cache for disk
  - Page table maps virtual page numbers to physical frames
  - Use a page table cache for recent translation
    => *Translation Lookaside Buffer* (TLB)



Integrating TLB and Cache

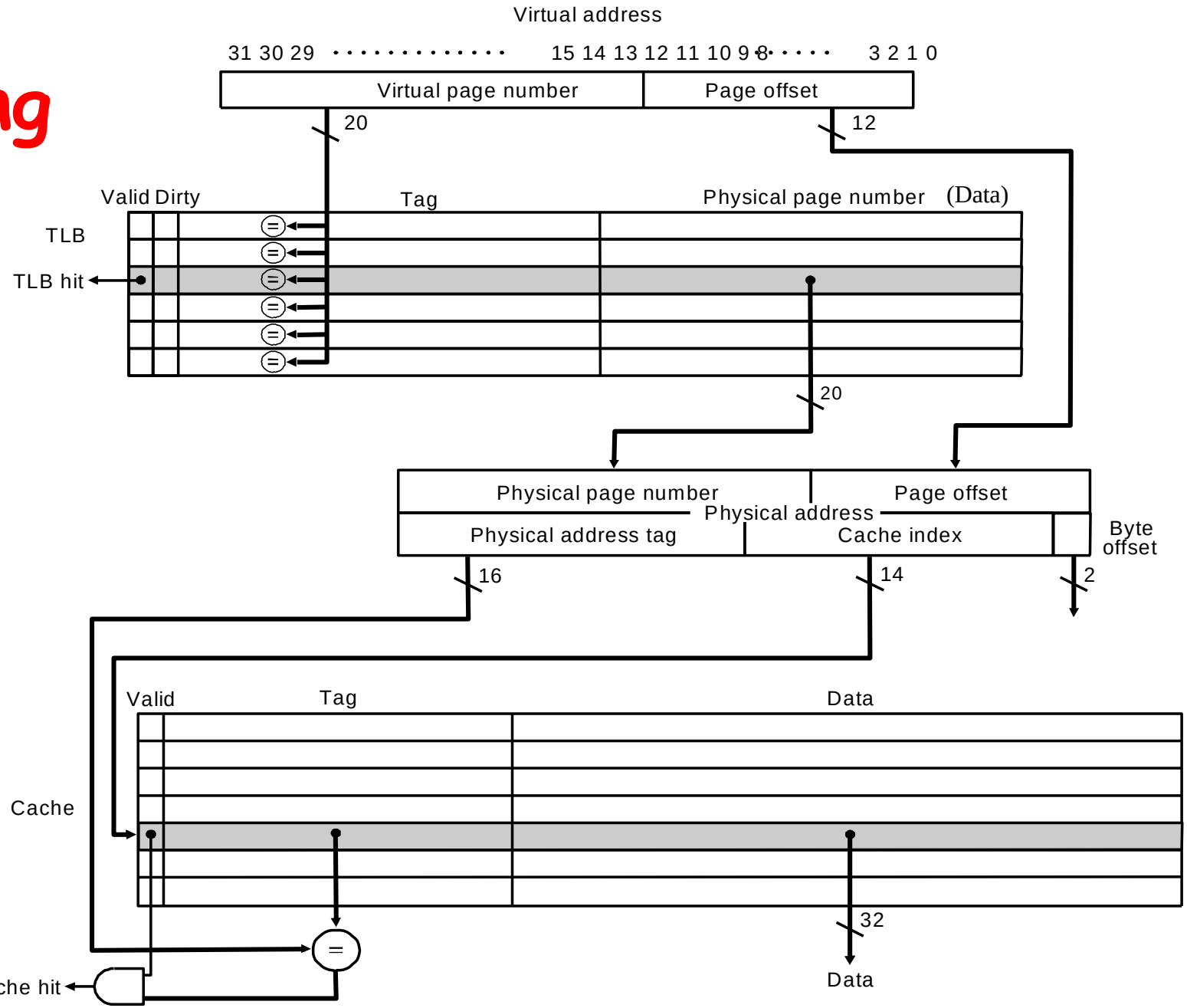
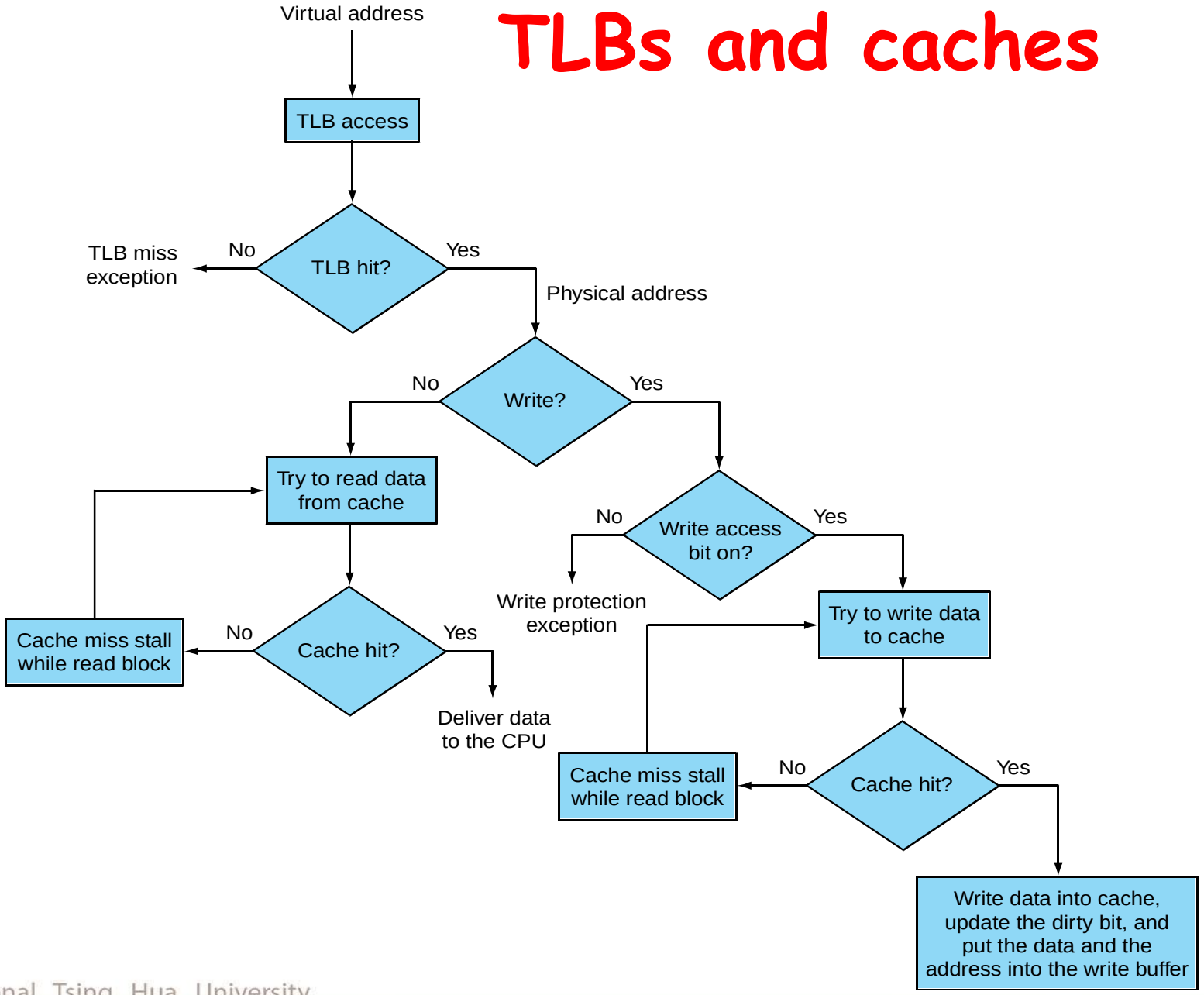


Fig. 5.24

TLBs and caches



Possible Combinations of Events

Cache	TLB	Page table	Possible? Conditions?
Miss	Hit	Hit	Yes; but page table never checked if TLB hits
Hit	Miss	Hit	TLB miss, but entry found in page table;after retry, data in cache
Miss	Miss	Hit	TLB miss, but entry found in page table; after retry, data miss in cache
Miss	Miss	Miss	TLB miss and is followed by a page fault; after retry, data miss in cache
Miss	Hit	Miss	impossible; not in TLB if page not in memory
Hit	Hit	Miss	impossible; not in TLB if page not in memory
Hit	Miss	Miss	impossible; not in cache if page not in memory

Outline

- Memory hierarchy
- The basics of caches
  - Direct-mapped cache
  - Address sub-division
  - Cache hit and miss
  - Memory support
- Measuring cache performance
- Improving cache performance
  - Set associative cache
  - Multiple level cache
- Virtual memory
  - Basics
  - Issues in virtual memory
  - Handling huge page table
  - TLB (Translation Lookaside Buffer)
  - TLB and cache
- A common framework for memory hierarchy

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 (1-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 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

Challenge in Memory Hierarchy

- Every change that potentially improves miss rate can negatively affect overall performance

Design change	Effects on miss rate	Possible effects
size ↑	capacity miss ↓	access time ↑
associativity ↑	conflict miss ↓	access time ↑
block size ↑	spatial locality ↑	miss penalty ↑

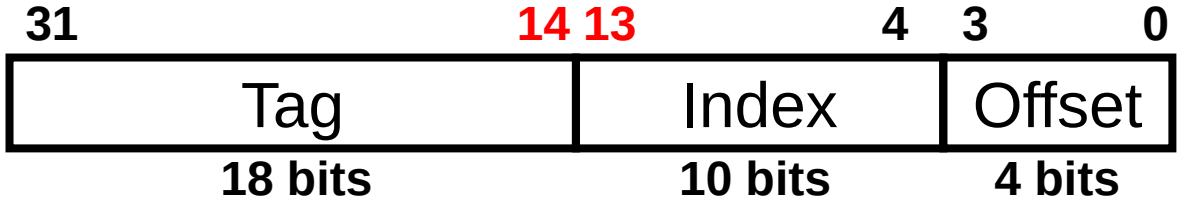
- Trends:
  - Synchronous SRAMs (provide a burst of data)
  - Redesign DRAM chips to provide higher bandwidth or processing
  - Restructure code to increase locality
  - Use prefetching (make cache visible to ISA)

Outline

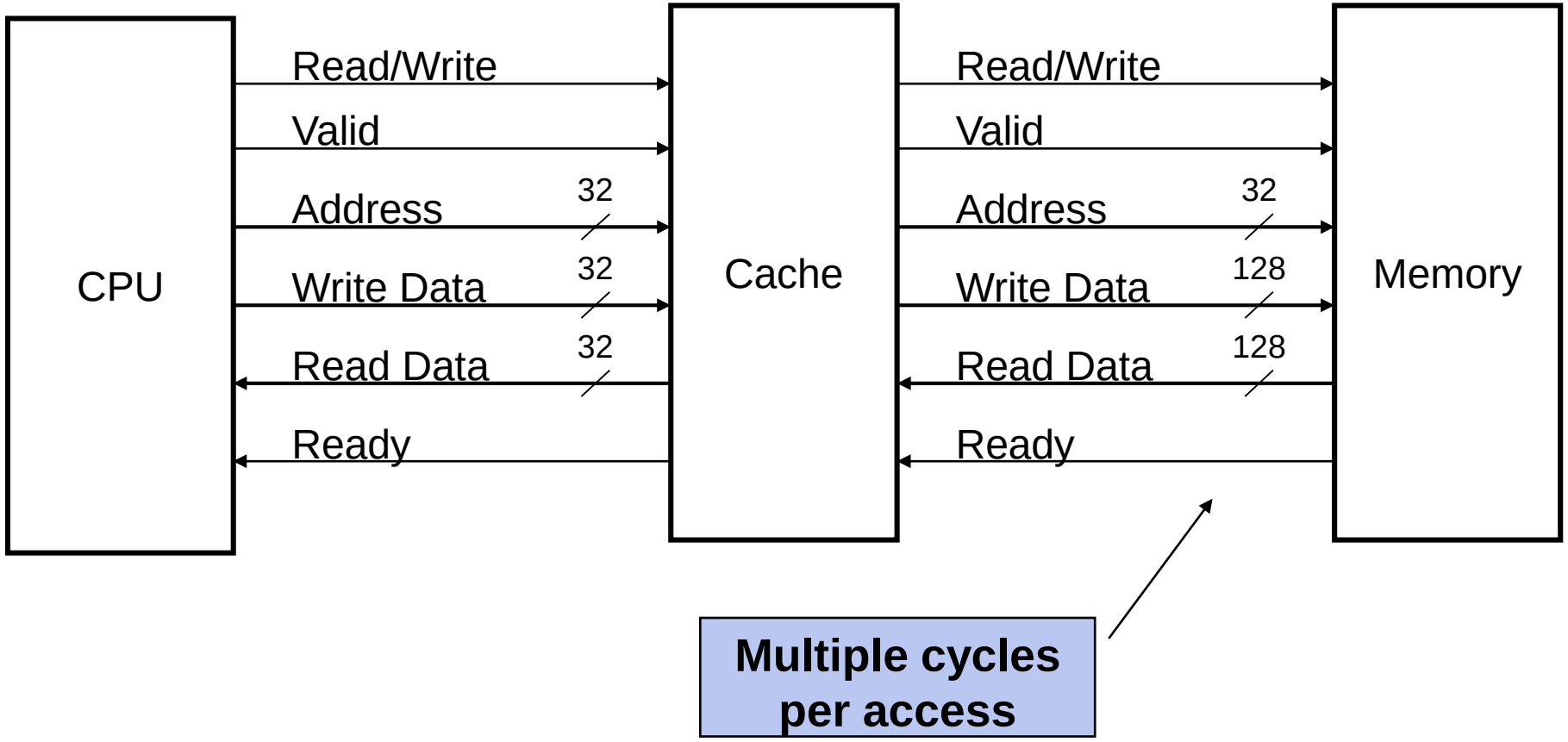
- Memory hierarchy
- The basics of caches
- Measuring and improving cache performance
- Virtual memory
- A common framework for memory hierarchy
- Using a Finite State Machine to Control a Simple Cache
- Parallelism and Memory Hierarchies: Cache Coherence

Cache Control

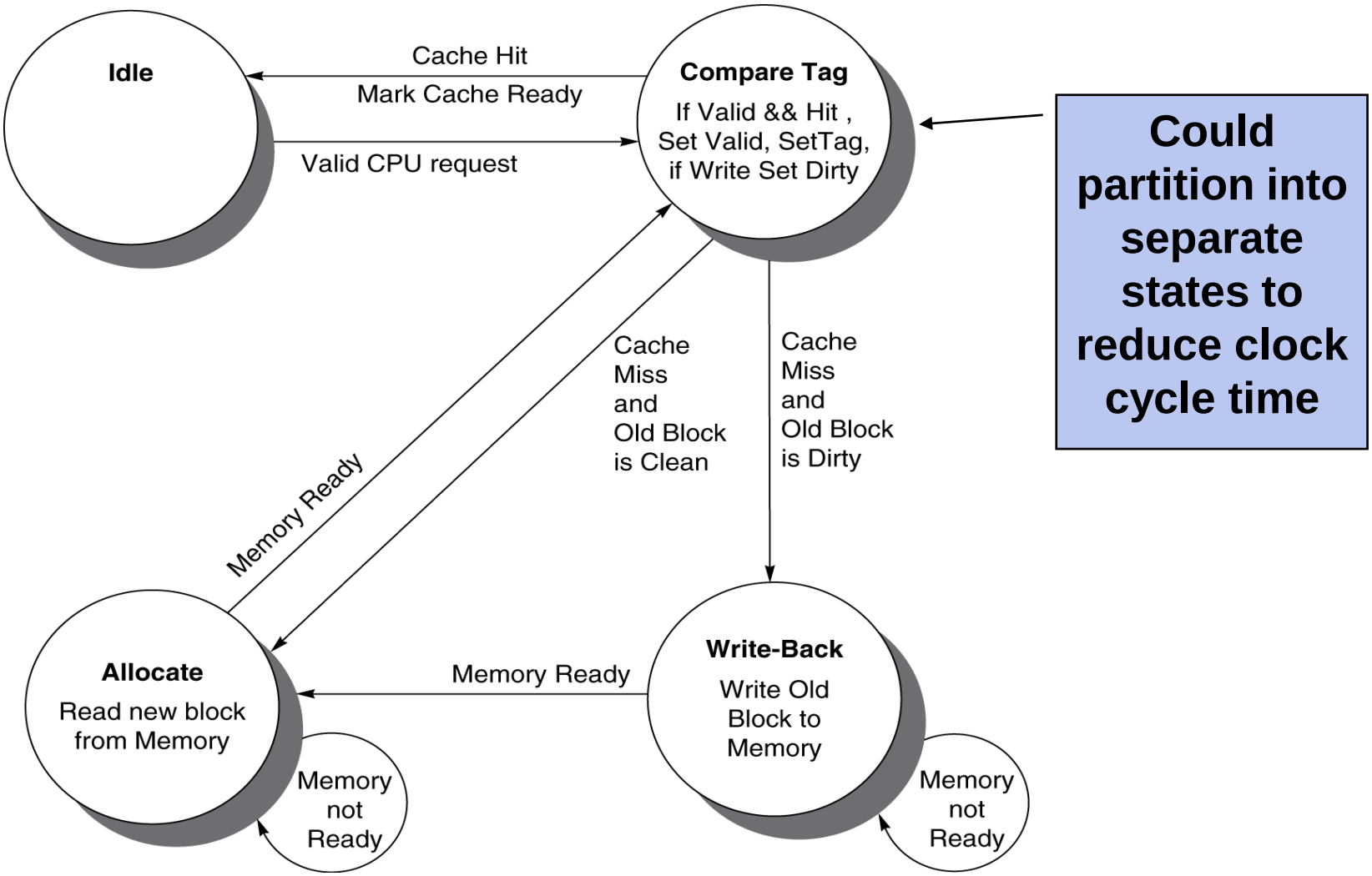
- Example cache characteristics
  - Direct-mapped, write-back, write allocate
  - Block size: 4 words (16 bytes)
  - Cache size: 16 KB (1024 blocks)
  - 32-bit byte addresses
  - Valid bit and dirty bit per block
  - Blocking cache
    - CPU waits until access is complete



Interface Signals

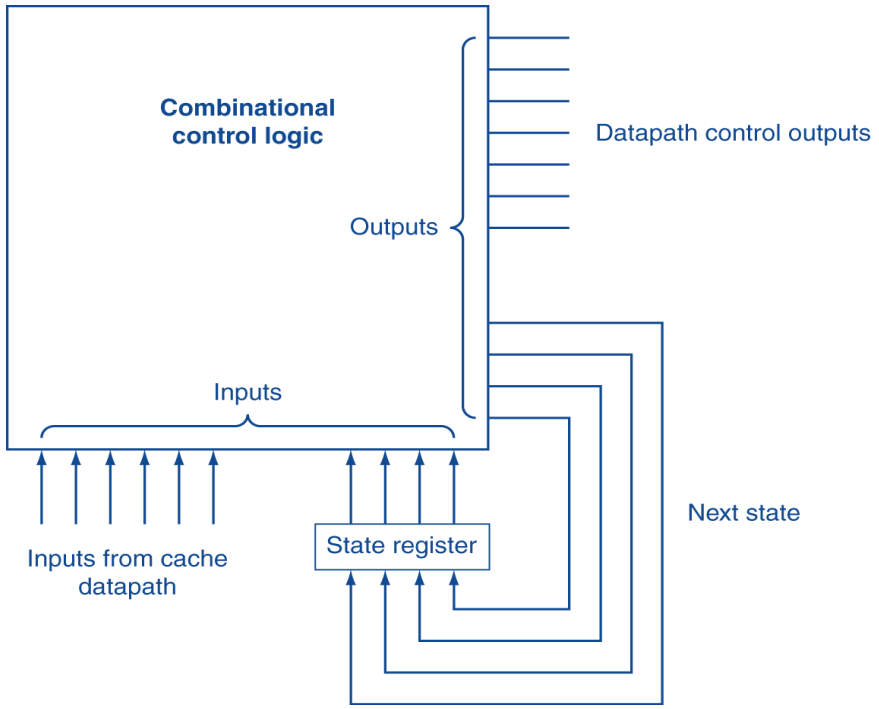


Cache Controller FSM



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)



Outline

- Memory hierarchy
- The basics of caches
- Measuring and improving cache performance
- Virtual memory
- A common framework for memory hierarchy
- Using a Finite State Machine to Control a Simple Cache
- Parallelism and Memory Hierarchies: Cache Coherence

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

- Two different processors have two different values for the same location

Cache Coherence Protocols

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

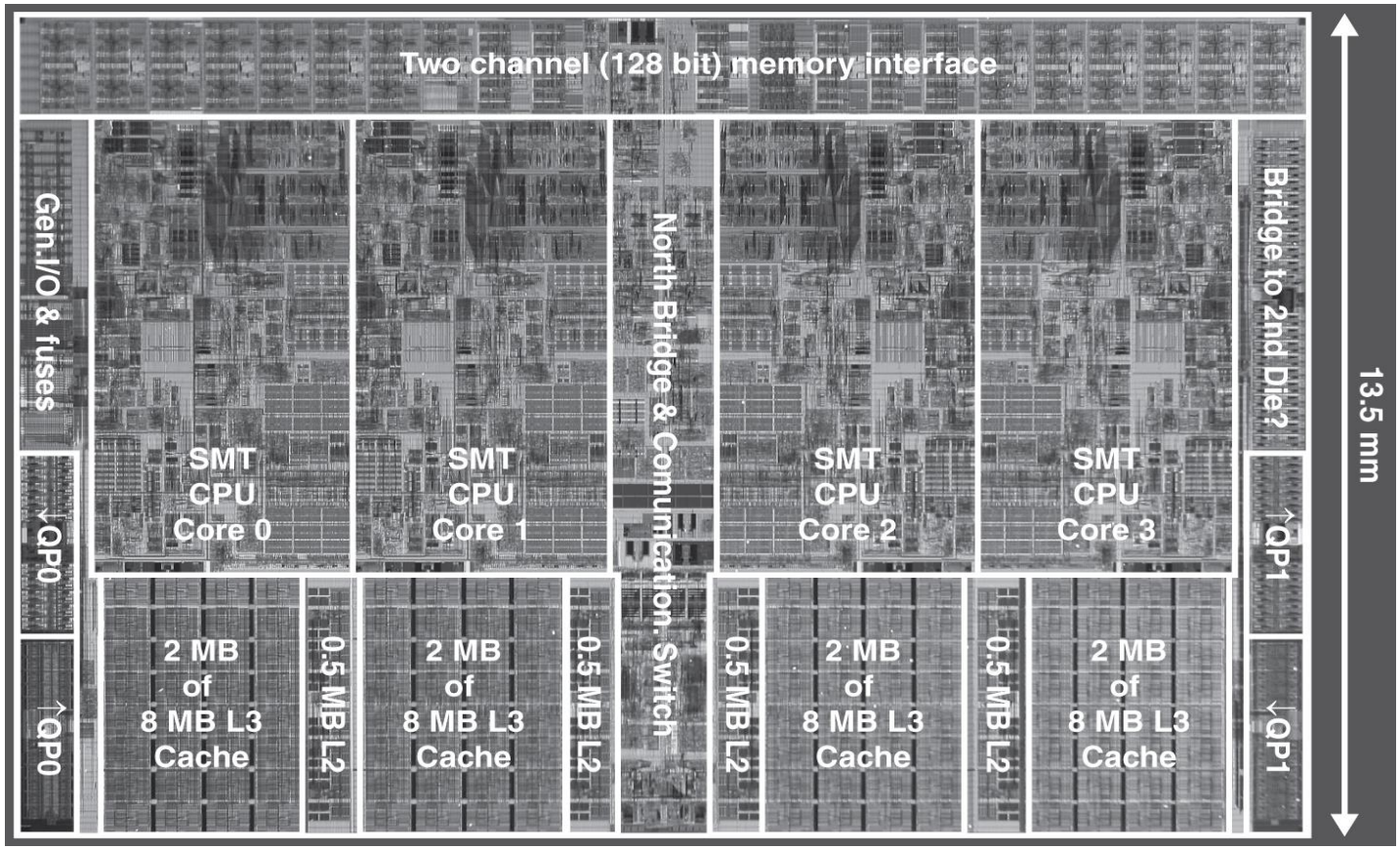
Invalidating Snooping Protocols

- 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

Multilevel On-Chip Caches

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache

2-Level TLB Organization

	Intel Nehalem	AMD Opteron X4
Virtual addr	48 bits	48 bits
Physical addr	44 bits	48 bits
Page size	4KB, 2/4MB	4KB, 2/4MB
L1 TLB (per core)	L1 I-TLB: 128 entries for small pages, 7 per thread (2x) for large pages L1 D-TLB: 64 entries for small pages, 32 for large pages Both 4-way, LRU replacement	L1 I-TLB: 48 entries L1 D-TLB: 48 entries Both fully associative, LRU replacement
L2 TLB (per core)	Single L2 TLB: 512 entries 4-way, LRU replacement	L2 I-TLB: 512 entries L2 D-TLB: 512 entries Both 4-way, round-robin LRU
TLB misses	Handled in hardware	Handled in hardware

3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available

Pitfalls

- Byte vs. word addressing
  - Example: 32-byte direct-mapped cache, 4-byte blocks
    - Byte 36 maps to block 1
    - Word 36 maps to block 4
- Ignoring memory system effects when writing or generating code
  - Example: iterating over rows vs. columns of arrays
  - Large strides result in poor locality

Pitfalls

- In multiprocessor with shared L2 or L3 cache
  - Less associativity than cores results in conflict misses
  - More cores ⇒ need to increase associativity
- Using AMAT (Average Memory Access Time) to evaluate performance of out-of-order processors
  - Ignores effect of non-blocked accesses
  - Instead, evaluate performance by simulation

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 ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors