

Chapter 5

Memory Hierarchy

Tien-Fu Chen

Dept. of Computer Science and
Information Engineering

National Chiao Tung Univ.

Material source:
RISC-V slides

Since 1980, CPU has outpaced DRAM ...

Q. How do architects address this gap?

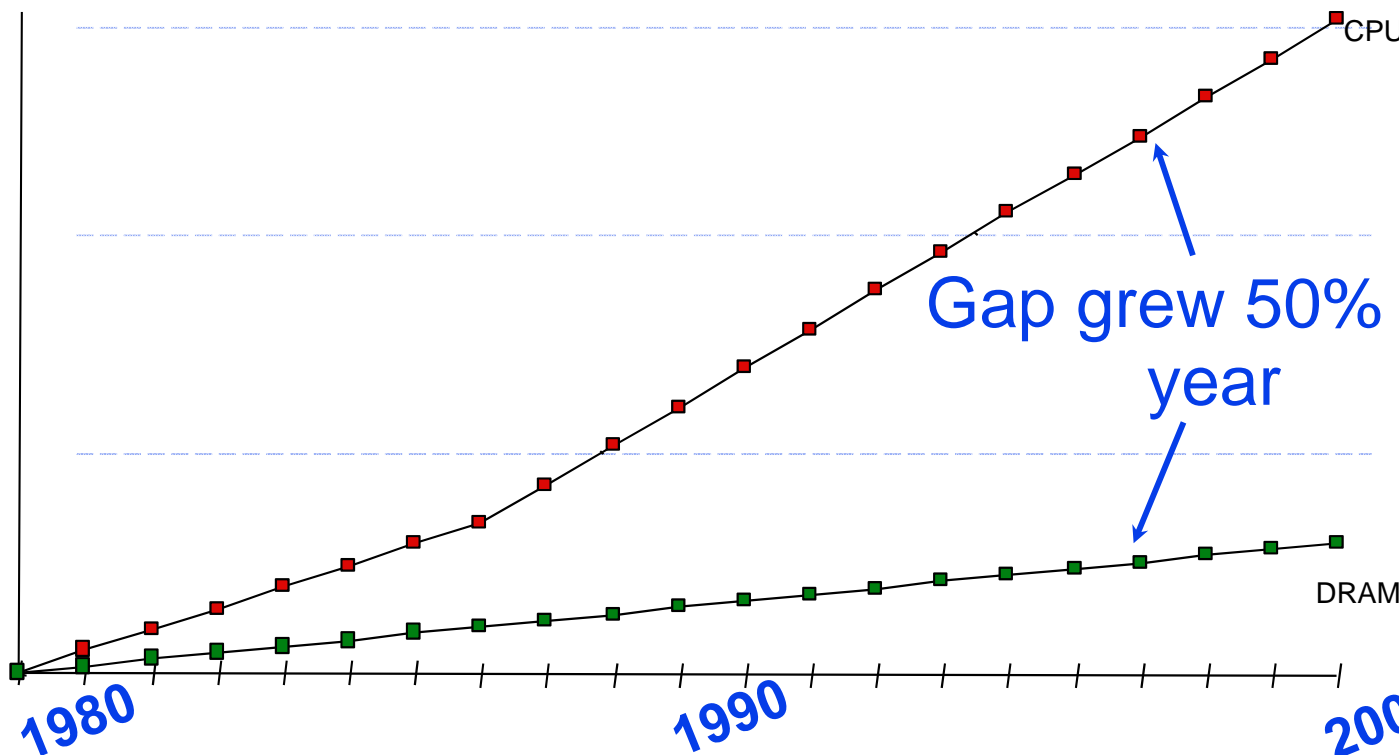
A. Put smaller, faster “cache” memories between CPU and DRAM.
Create a “memory hierarchy”.

Performance
(1/latency)

1000

100

10



CPU

60% per yr
2X in 1.5 yrs

Gap grew 50% per
year

DRAM

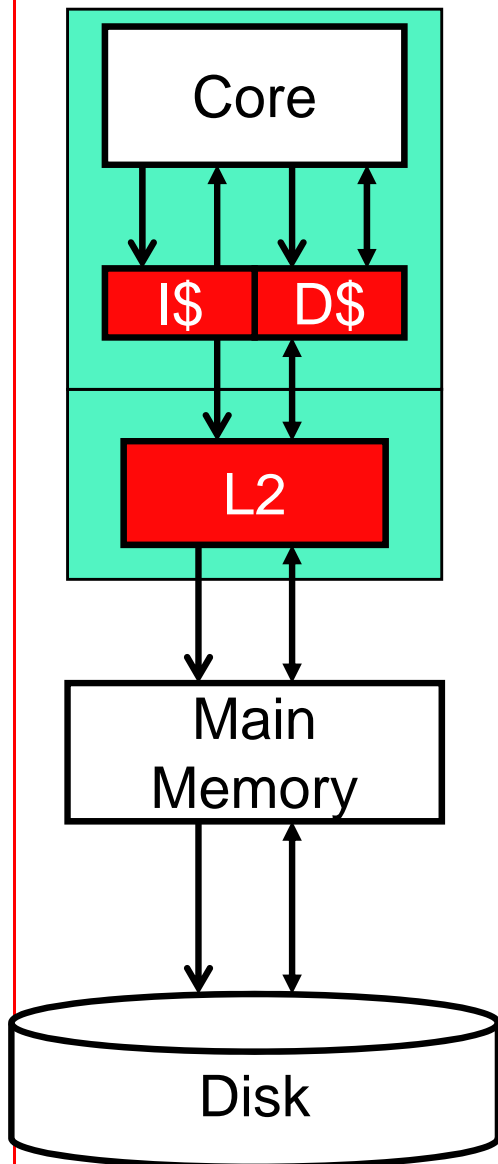
9% per yr
2X in 10 yrs

DRAM

2000

Year

Key concepts in this chapter



- ❑ “Cache”: hardware managed
 - Hardware automatically retrieves missing data
 - Built from fast on-chip SRAM
 - In contrast to off-chip, DRAM “main memory”
- ❑ **Average access time** of a memory component
 - $latency_{avg} = latency_{hit} + (\%_{miss} * latency_{miss})$
 - Hard to get low $latency_{hit}$ and $\%_{miss}$ in one structure → memory hierarchy
- ❑ Cache ABCs (**associativity, block size, capacity**)
- ❑ **Performance optimizations**
 - Prefetching & data restructuring
- ❑ **Handling writes**
 - Write-back vs. write-through
- ❑ **Memory hierarchy**
 - Smaller, faster, expensive → bigger, slower, cheaper

Memory Hierarchy

❑ Speed vs. cost

SRAM access times 0.5ns – 2.5ns, \$2000 – \$5000 per GB.

DRAM access times are 50ns – 70ns, \$20 – \$75 per GB.

Disk access times are 5ms – 20ms, \$0.20 – \$2 per GB.

❑ build a memory hierarchy

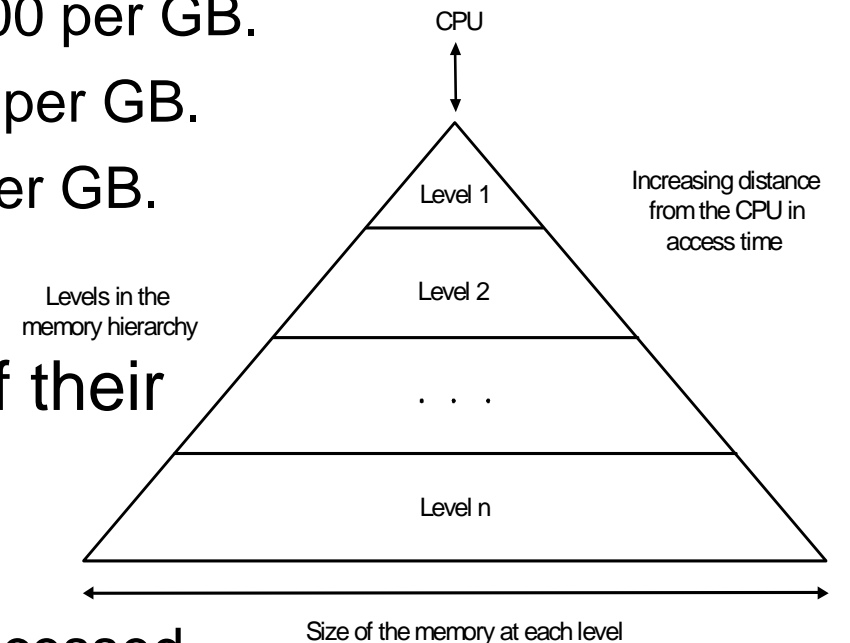
❑ Programs access a small proportion of their address space at any time

❑ Temporal locality

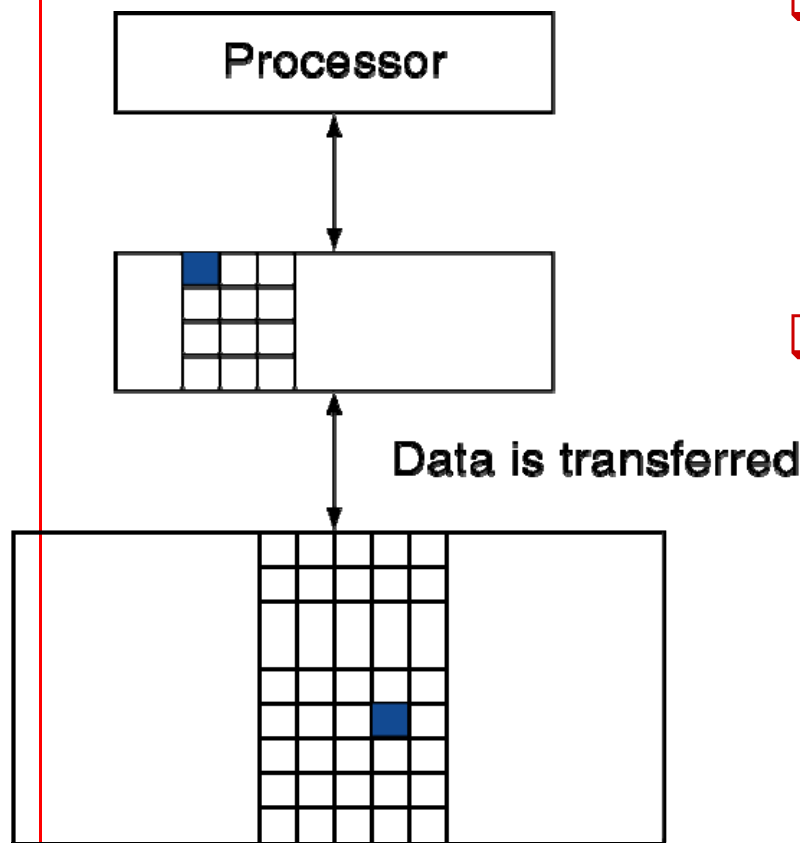
- Items accessed recently are likely to be accessed again soon
- e.g., instructions in a loop, induction variables

❑ Spatial locality

- Items near those accessed recently are likely to be accessed soon
- E.g., sequential instruction access, array data



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
 $= 1 - \text{hit ratio}$
 - Then accessed data supplied from upper level

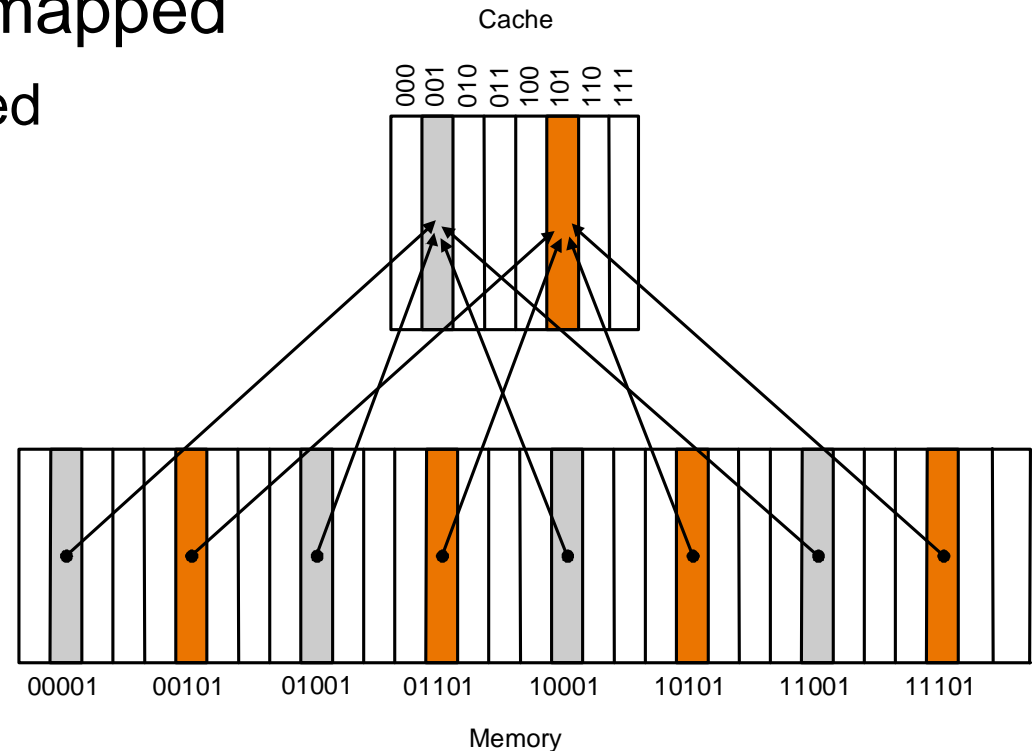
Basics of caches

□ Terms:

- block: minimum unit of data
- hit: data requested is in the upper level
- miss: data requested is not in the upper level

□ Accessing a cache - direct mapped

- a referenced address is divided
 - a cache index
 - a tag field
- taking advantage of locality?
 - Enlarge block size to capture spatial locality



Hits vs. Misses

- ❑ Read hits

- That is good!

- ❑ Read misses

- stall the CPU, fetch block from memory, deliver to cache, restart

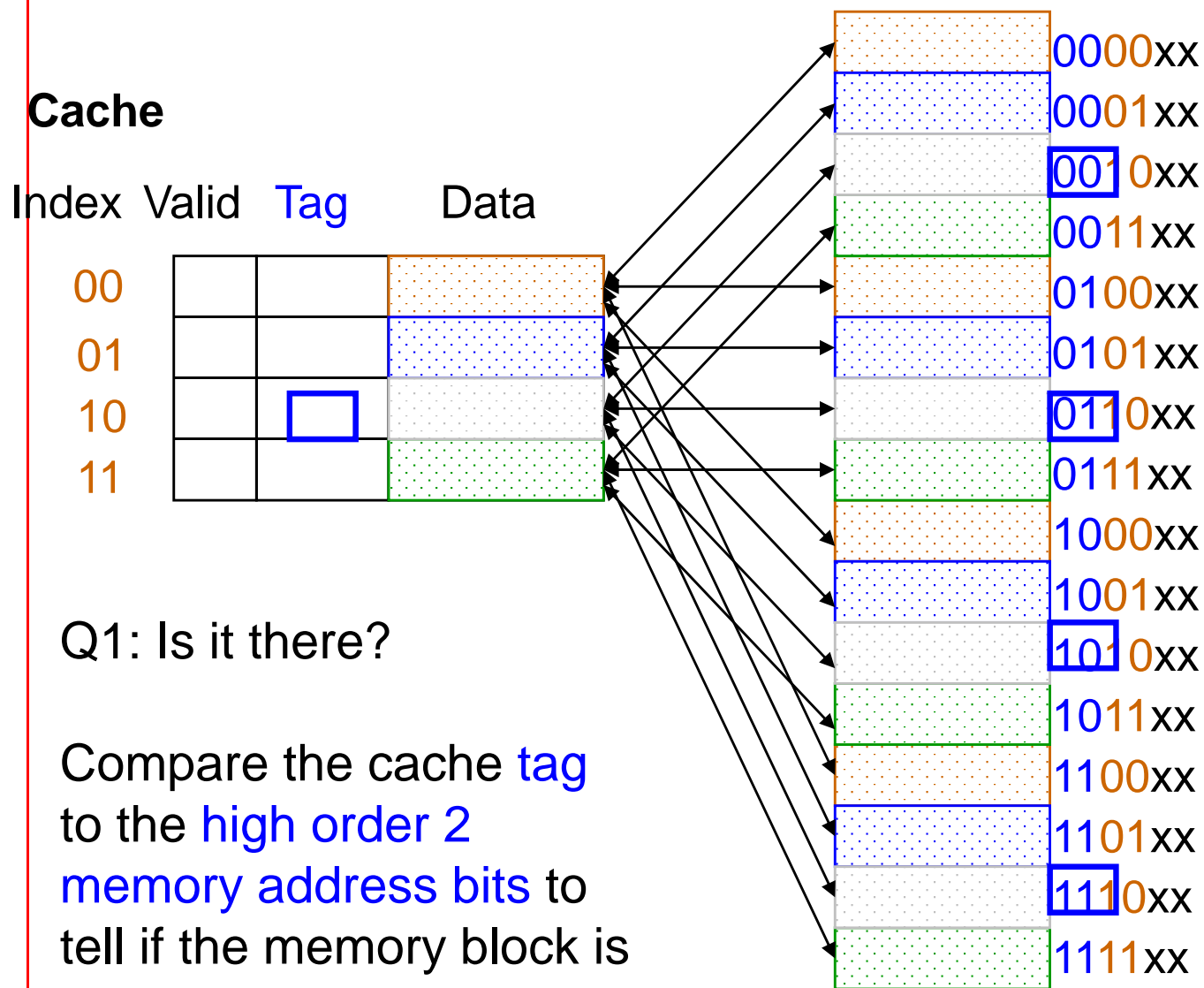
- ❑ Write hits:

- 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

Caching: A Simple First Example



Main Memory

One word blocks

Two low order bits define the byte in the word (32b words)

Q2: How do we find it?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

(block address) modulo (# of blocks in the cache)

Cache Example –

- 8-blocks, 1 word (4B)/block, direct mapped
- Accessing 22, 26, 22, 26, 16, 3, 16, 18, 16

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

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

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

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

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 –

- ❑ 8-blocks, 1 word/block, direct mapped

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

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	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

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		

Four Questions for Caches and Memory Hierarchy

❑ 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 identification

❑ Q3: Which block should be replaced on a miss?

Block replacement

❑ Q4: What happens on a write?

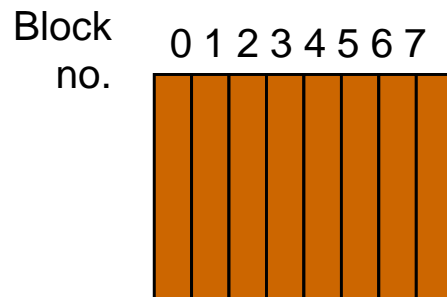
Write strategy

Q1: Where can a block be placed in the upper level?

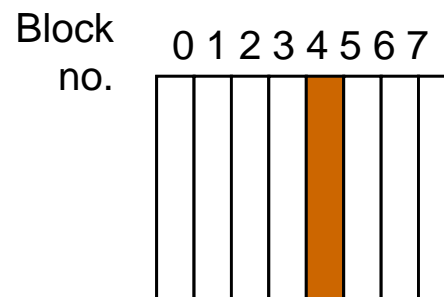
❑ Block 12 placed in 8 block cache:

- Fully associative, direct mapped, 2-way set associative
- S.A. Mapping = Block Number Modulo Number Sets

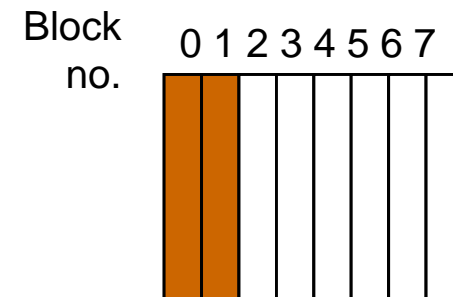
Fully associative:
block 12 can go
anywhere



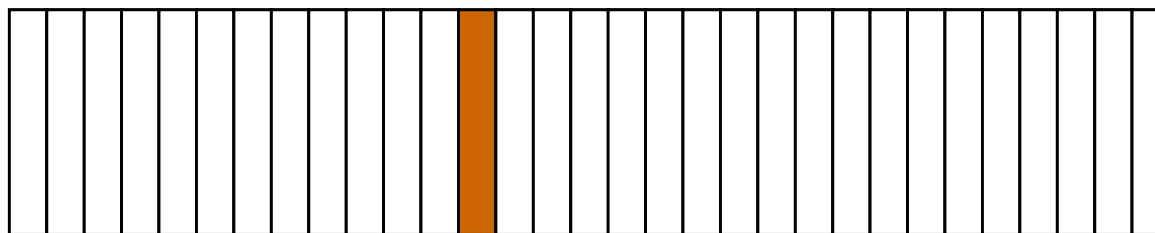
Direct mapped:
block 12 can go only
into block 4 ($12 \bmod 8$)



Set associative:
block 12 can go
anywhere in set 0
($12 \bmod 4$)



Block-frame address



Block no.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3

Associativity choices with 8-block cache

0	tag	Data
1	tag	Data
2	tag	Data
3	tag	Data
4	tag	Data
5	tag	Data
6	tag	Data
7	tag	Data

***One Way Associative
(Direct Mapped)***

0	tag	Data	tag	Data
1	tag	Data	tag	Data
2	tag	Data	tag	Data
3	tag	Data	tag	Data

Two Way Associative

0	tag	Data	tag	Data	tag	Data	tag	Data
1	tag	Data	tag	Data	tag	Data	tag	Data

Four Way Associative

tag	Data	tag	Data	tag	Data	tag	Data	tag	Data	tag	Data	tag	Data	tag	Data
-----	------	-----	------	-----	------	-----	------	-----	------	-----	------	-----	------	-----	------

***Eight Way Associative
(Fully Associative)***

Cache Strategies – Block Placement

❑ Direct Mapped

- Each block has only one place to go in a cache, typically
 - $\text{Address} \% \text{Num-Blocks-In-Cache}$
 - Usually lower n bits corresponds to the offset in the block, where $2^n = \text{Block size}$, and then another m bits corresponding to the cache block, where $2^m = \text{Num blocks in the cache}$

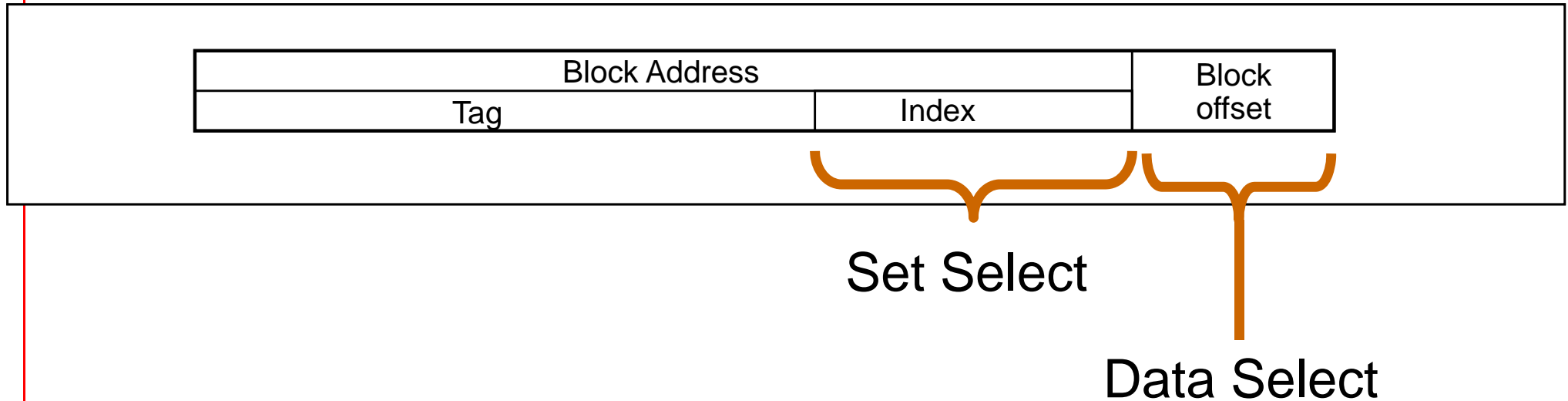
❑ Fully Associative

- Block can be placed anywhere in the cache
- Implies we must be able to search for it associatively

❑ Set Associative

- A **set** is a group of arbitrary blocks in the cache
- An address in memory maps to a set, then maps into a block within that set
- Usually lower n bits corresponds to the offset in the block, where $2^n = \text{Block size}$, and then another m bits corresponding to the set, where $2^m = \text{Num sets}$

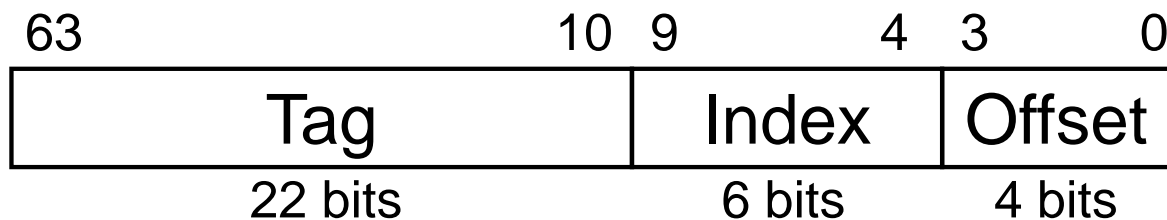
Q2: How is a block found if it is in the upper level?



- ❑ Direct indexing (using index and block offset), tag compares, or combination
- ❑ Increasing associativity \Rightarrow shrinks index \Rightarrow expands tag

Example: 1K-byte Cache

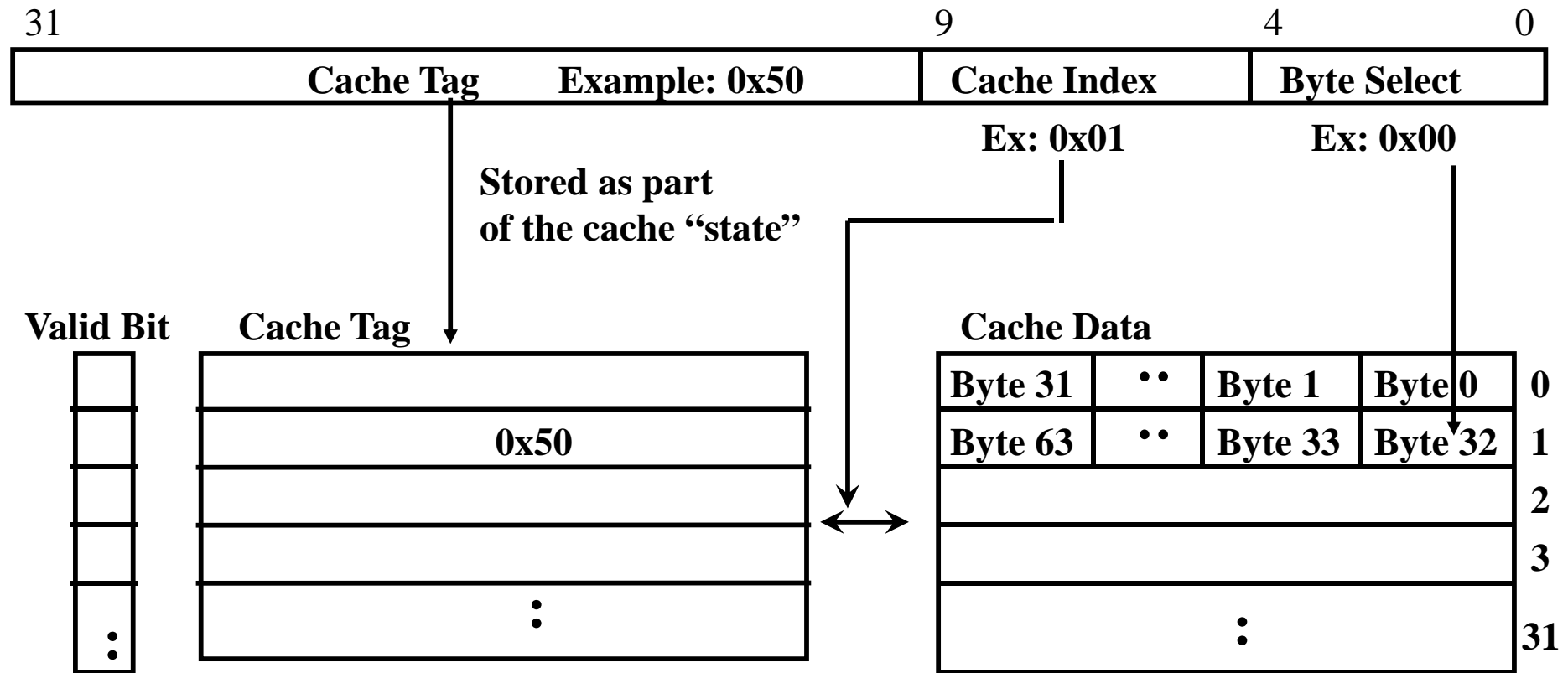
- ❑ 64 blocks, 16 bytes/block
 - To what block number does address 1200 map?
- ❑ Block address = $\lfloor 1200/16 \rfloor = 75$
- ❑ Block number = 75 modulo 64 = 11



addr	tag	index	offset
1200	0001	11d	0
0x4B0			
0x12346			

Example: 1 KB Direct Mapped Cache with 32 B Blocks

- ❑ For a 2^N byte cache:
 - The uppermost $(32 - N)$ bits are always the Cache Tag
 - The lowest M bits are the Byte Select (Block Size = 2^M)



- ❑ Total bits = $2^n * (\text{block size} + \text{tag size} + \text{valid field})$

Total # of Bits in a Cache

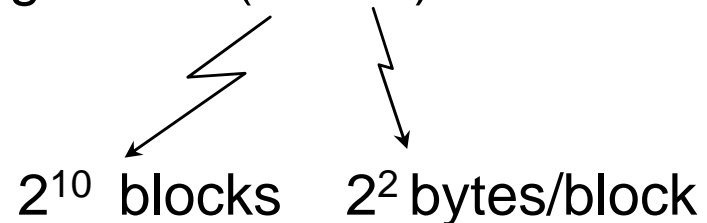
Total # of bits = # of blocks x (# of bits of a tag + # of bits of a block +
of bits in valid field)

For the example:

❑ **Direct mapped Cache** with 4KB of data, 4B blocks and 32 bit address

→ 4kB = 1k 4B words = 2^{10} 4B words = 2^{10} blocks

of bits of tag = $32 - (10 + 2) = 20$


 2^{10} blocks 2^2 bytes/block

Total # of bits = $2^{10} \times (20 + 32 \times 1 + 1) = 53 \times 2^{10} = 53 \text{ kbits} = 6.625 \text{ kBytes}$

❑ **How about 4-way Cache** with 4KB of data, 1-word blocks and 32 bit address?

Q3: Which block should be replaced on a miss?

- ❑ Easy for Direct Mapped
- ❑ Set Associative or Fully Associative:
 - Random
 - LRU (Least Recently Used)

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

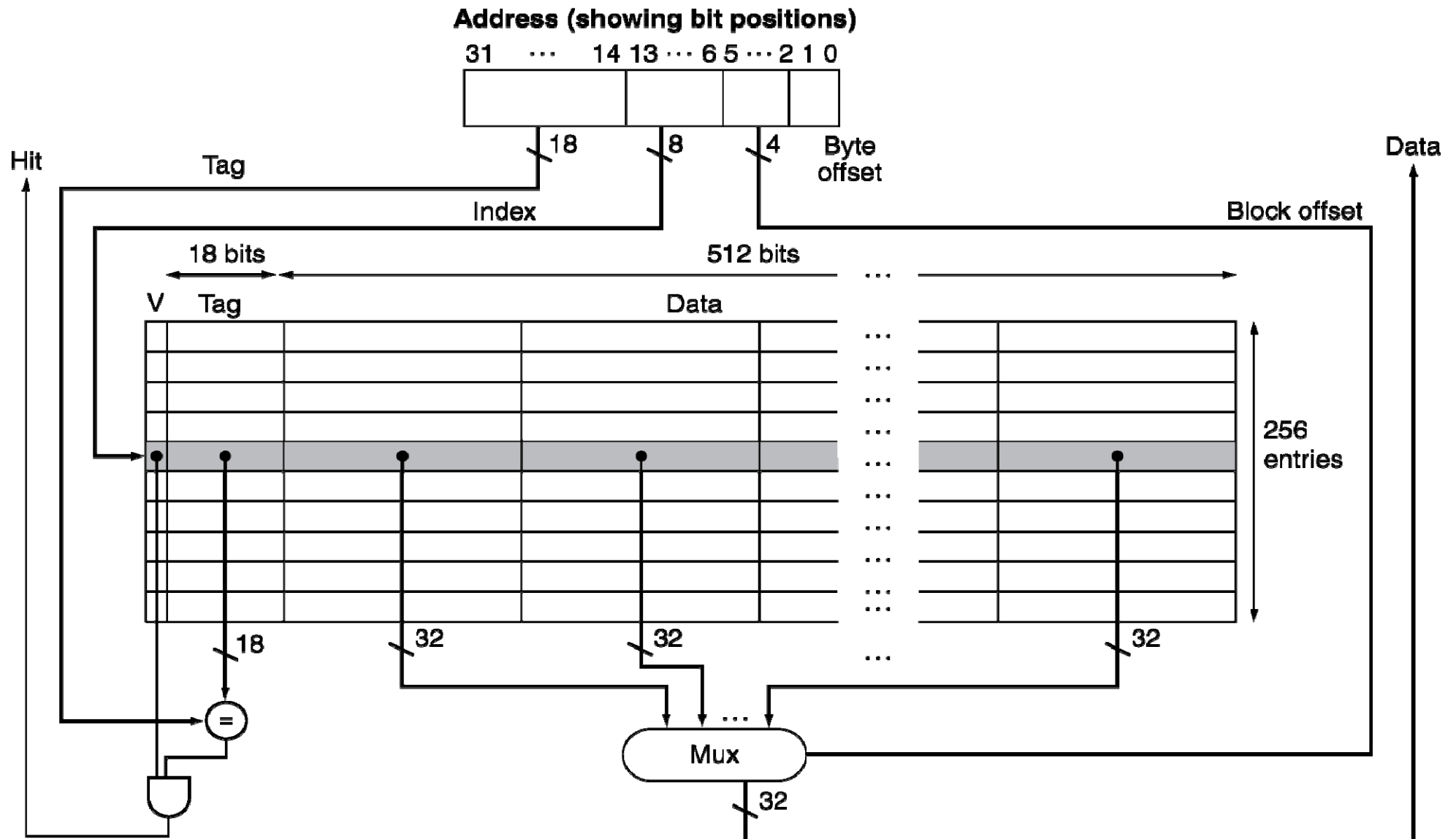
Q4:Write Policy: Write-Through vs Write-Back

	Write-Through	Write-Back
Policy	<ul style="list-style-type: none">❑ Data written to cache block❑ also written to lower-level memory	<ul style="list-style-type: none">❑ Write data only to the cache❑ Update lower level when a block falls out of the cache
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No
Advantages	<ul style="list-style-type: none">❑ Read misses are cheaper.❑ Simpler to implement.❑ Requires a write buffer to pipeline writes	<ul style="list-style-type: none">❑ Reduced traffic to memory❑ Individual writes performed at the processor rate

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 \times 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: Intrinsic FastMATH



Cache Performance by Average memory access time (AMAT)

Average memory access time
= hit time + miss rate \times miss penalty

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

□ Given (example in textbook)

- 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

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

□ 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

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 = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 65 \text{ cycles} = 0.25 \text{ B/cycle}$

Average Access Time

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

Improve Cache Performance

❑ Simplified Model

execution time = (execution cycles + memory stall cycles) x cycle time

stall cycles = # of refs x miss ratio x miss penalty

❑ Two ways of improving performance:

- decreasing the miss ratio
- decreasing the miss penalty

❑ Reducing Cache Misses

- Larger Block Size
- Higher Associativity

❑ Decreasing cache penalty

- Multilevel cache
- Larger memory bandwidth

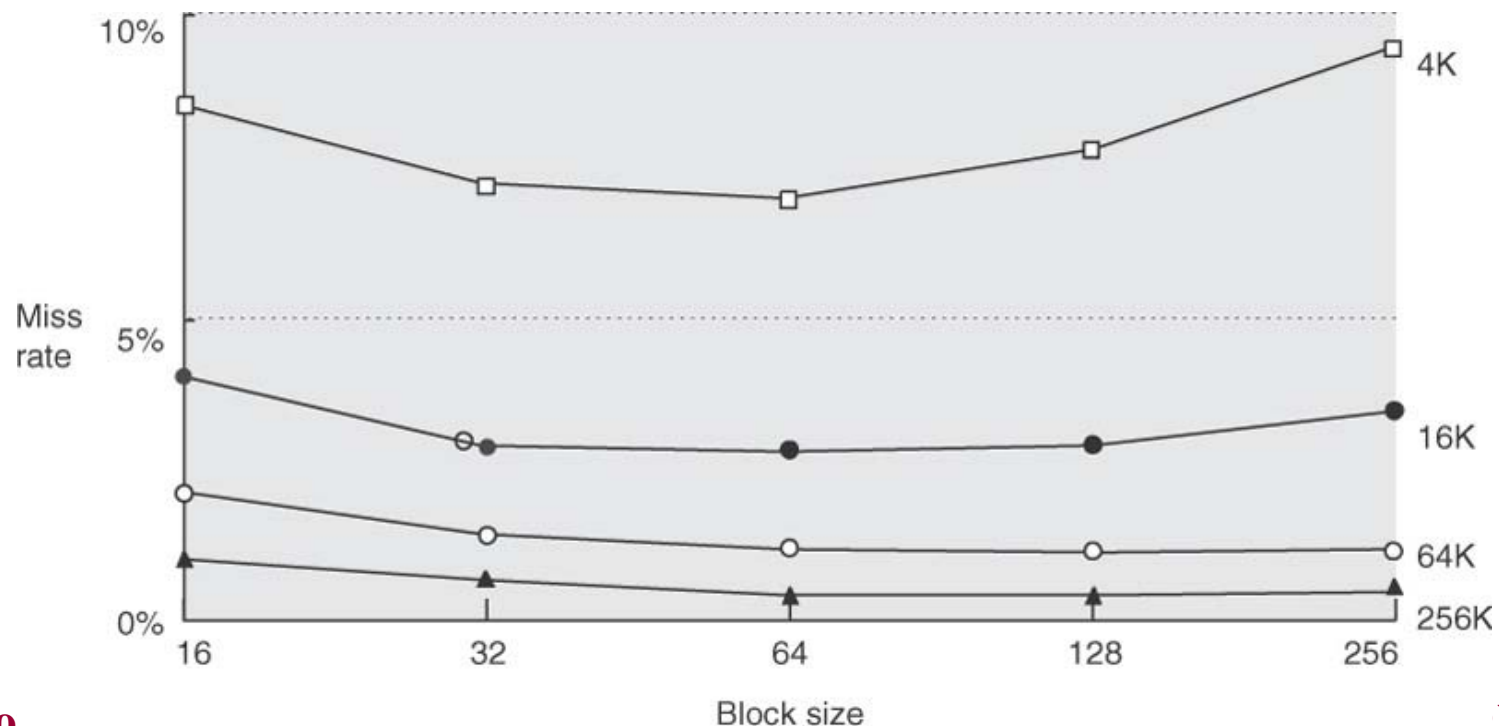
Sol 1: Increased Block Size

❑ Advantages

- Reduce compulsory misses due to spatial locality

❑ Disadvantages

- Larger block takes longer to move, so higher penalty for miss
- More conflicts now though, because there are fewer blocks in the cache, so more memory blocks map to the same cache blocks



Taking Advantage of Spatial Locality

- Let cache block hold more than one word

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15

0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss

01	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

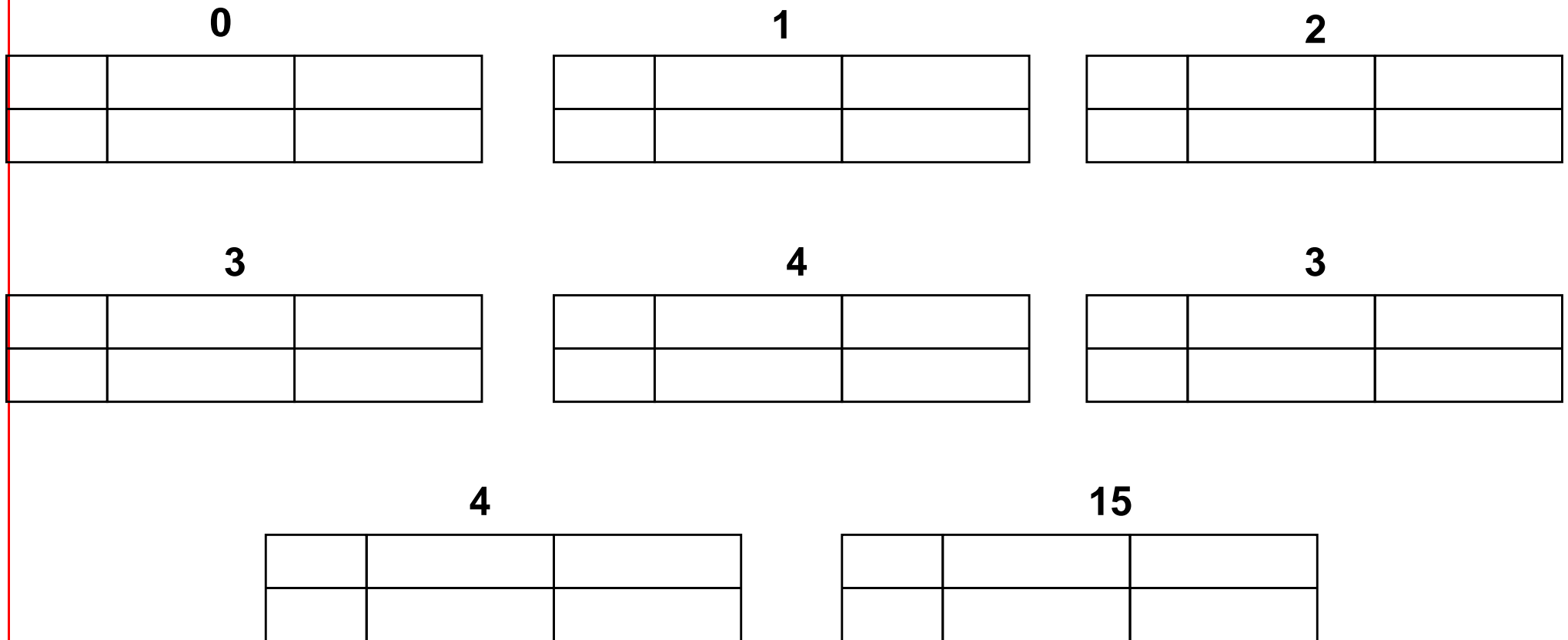
➤ 8 requests, 4 misses

Taking Advantage of Spatial Locality

- ❑ Let cache block hold more than one word

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15



➤ 8 requests, 4 misses

Sol 2: Higher Associativity

❑ Advantage

- Higher associativity should reduce conflicts

❑ Disadvantage

- Higher associativity can reduce number of sets, if we keep the same cache size
- There is overhead with higher associativity in the hardware, increases the basic clock cycle for all instructions

Cache size (KB)	Associativity			
	One-way	Two-way	Four-way	Eight-way
1	7.65	6.60	6.22	5.44
2	5.90	4.90	4.62	4.09
4	4.60	3.95	3.57	3.19
8	3.30	3.00	2.87	2.59
16	2.45	2.20	2.12	2.04
32	2.00	1.80	1.77	1.79
64	1.70	1.60	1.57	1.59
128	1.50	1.45	1.42	1.44

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				
8	0	miss				
0	0	miss				
6	2	miss				
8	0	miss				

Associativity Example: access sequence: 0, 8, 0, 6, 8

❑ 2-way set associative

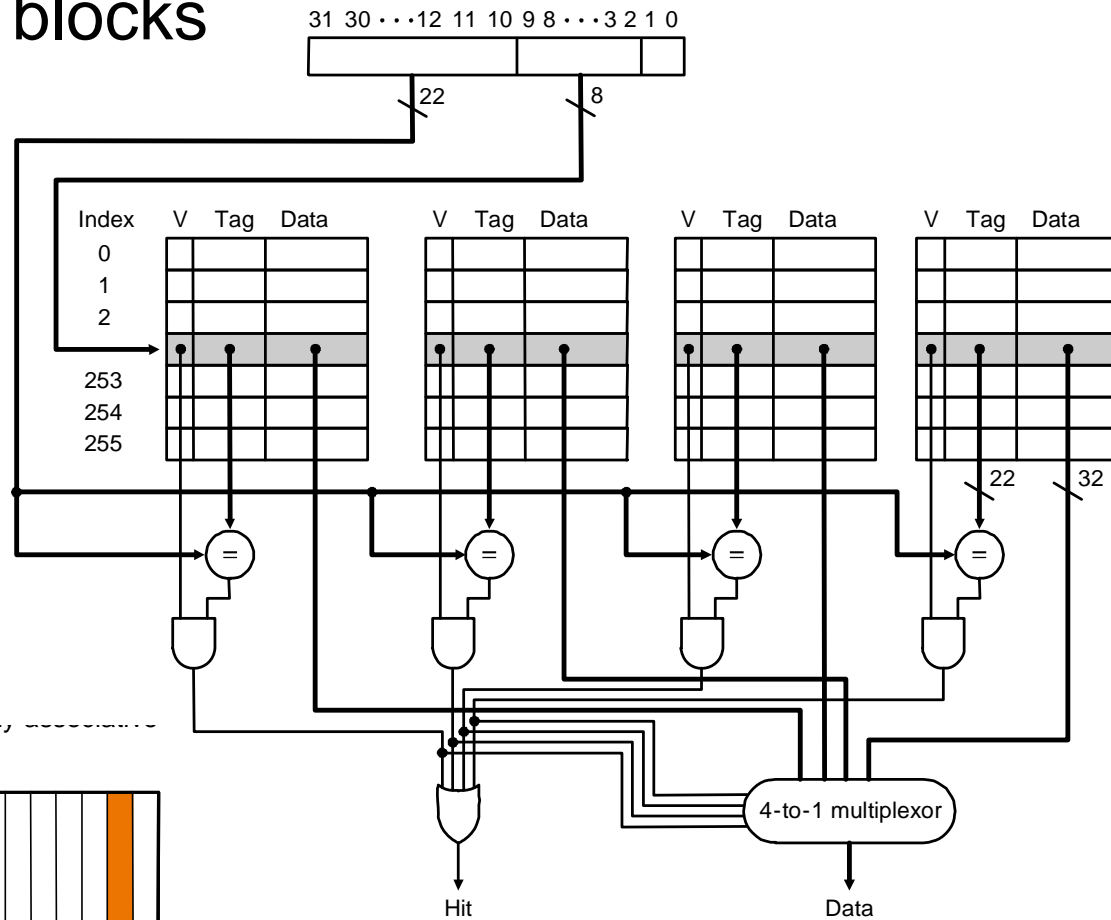
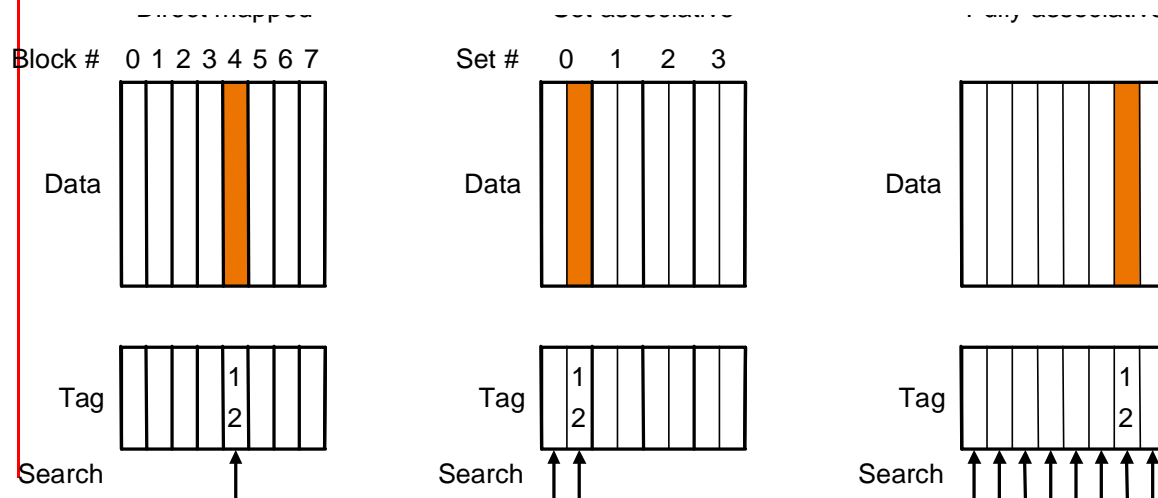
Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss				
8	0	miss				
0	0	hit				
6	0	miss				
8	0	miss				

❑ Fully associative

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

Decreasing miss ratio with associativity

- By more flexible placement of blocks



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.

Sol 3: Decreasing miss penalty with multilevel caches

❑ Add a second level cache:

- often primary cache is on the same chip as the processor
- use SRAMs to add another cache above primary memory (DRAM)
- miss penalty goes down if data is in 2nd level cache

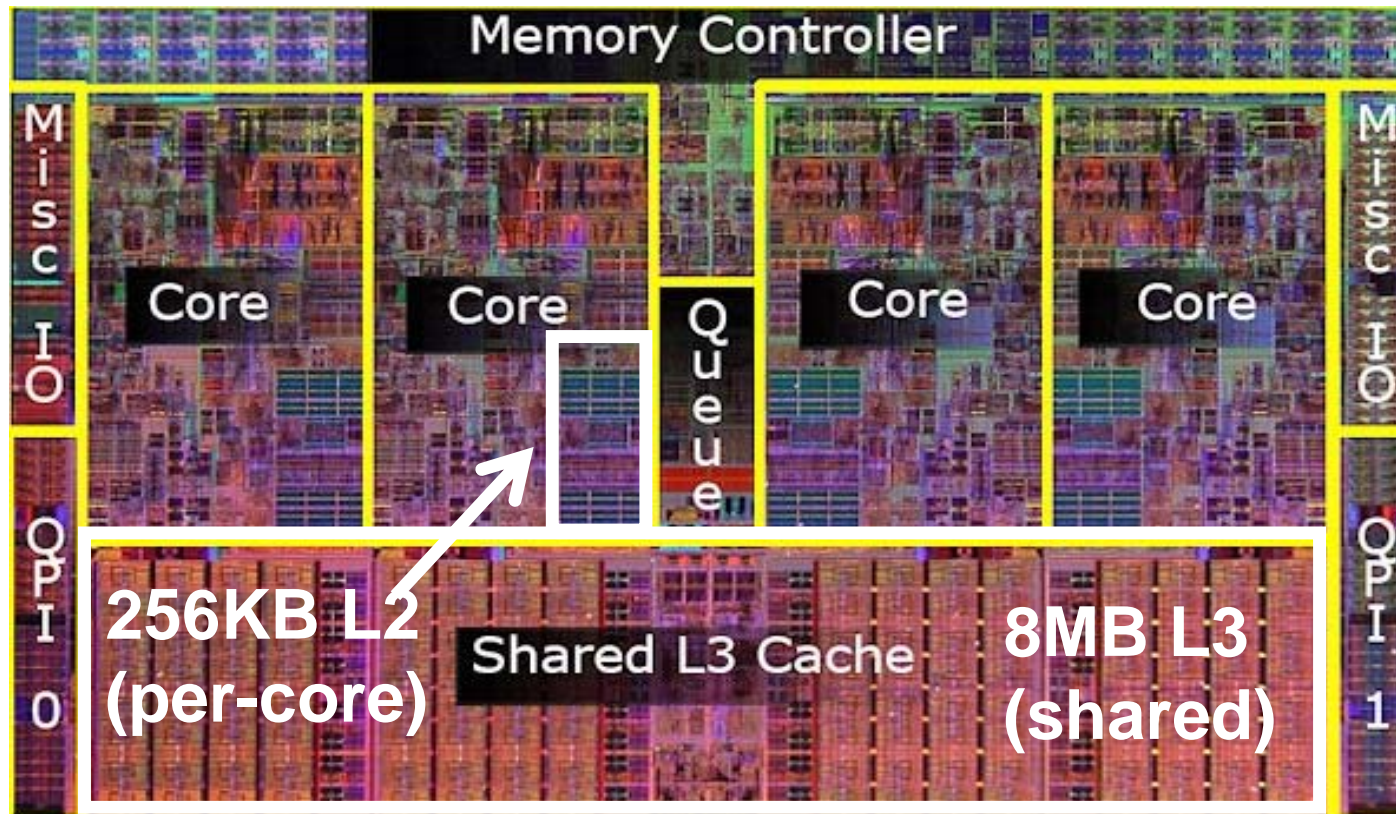
❑ Example:

- CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
- Adding 2nd level cache with 20ns access time decreases miss rate to 2%

❑ Optimize two caches separately

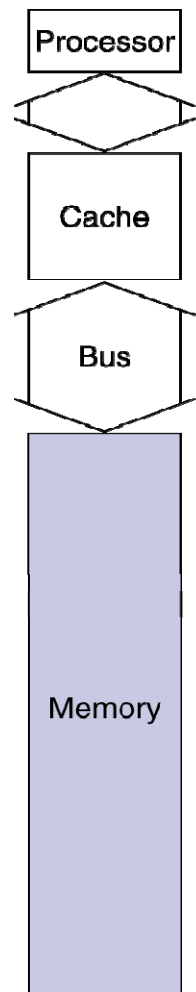
- 1st cache: unified cache, smaller block size, small associativity
- 2nd cache: split cache, larger block size, larger associativity

Example Cache Hierarchy: Core i7

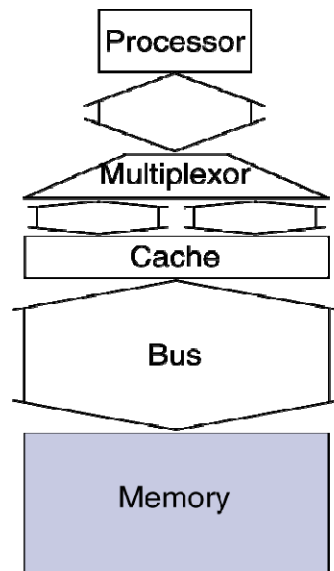


- ❑ Each core:
 - 32KB insn & 32KB data, 8-way set-associative, 64-byte blocks
 - 256KB second-level cache, 8-way set-associative, 64-byte blocks
- ❑ 8MB shared cache, 16-way set-associative

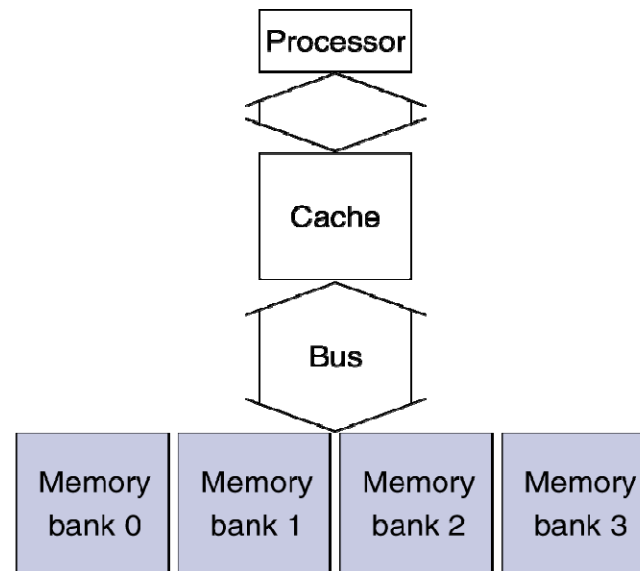
Sol 4: Increasing Memory Bandwidth



a. One-word-wide memory organization



b. Wider memory organization



c. Interleaved memory organization

❑ Interleaving

- sending to multiple banks and read data at the same time

❑ 4-word wide memory

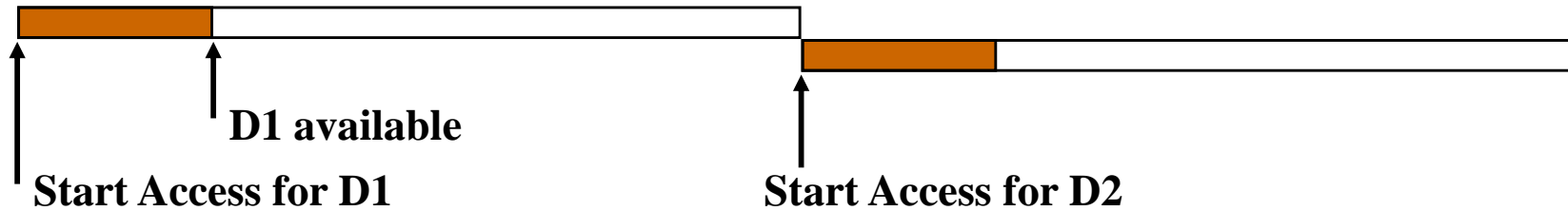
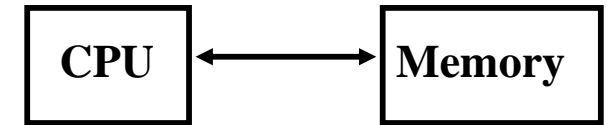
- Miss penalty = $1 + 15 + 1 = 17$ bus cycles
- Bandwidth = $16 \text{ bytes} / 17 \text{ cycles} = 0.94 \text{ B/cycle}$

❑ 4-bank interleaved memory

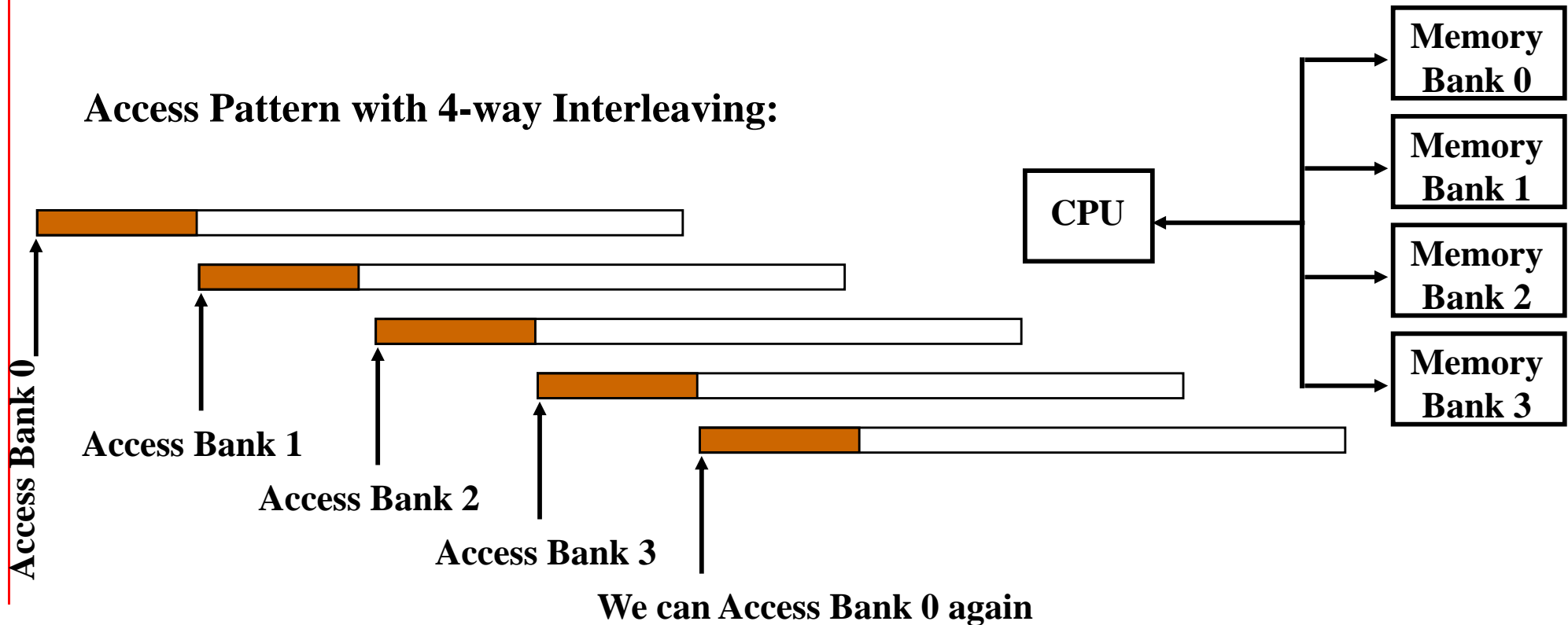
- Miss penalty = $1 + 15 + 4 \times 1 = 20$ bus cycles
- Bandwidth = $16 \text{ bytes} / 20 \text{ cycles} = 0.8 \text{ B/cycle}$

Increasing Bandwidth - Interleaving

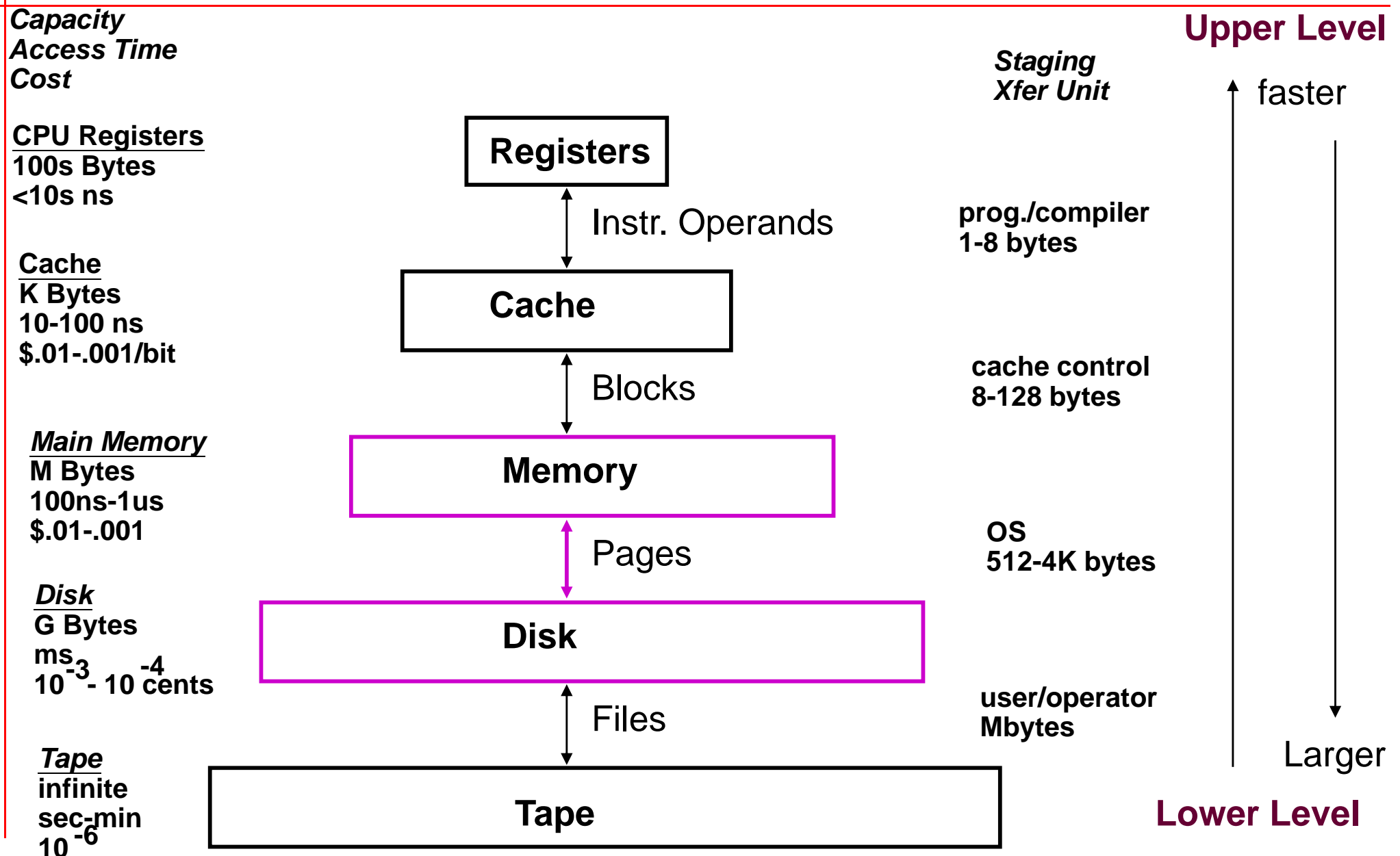
Access Pattern without Interleaving:



Access Pattern with 4-way Interleaving:

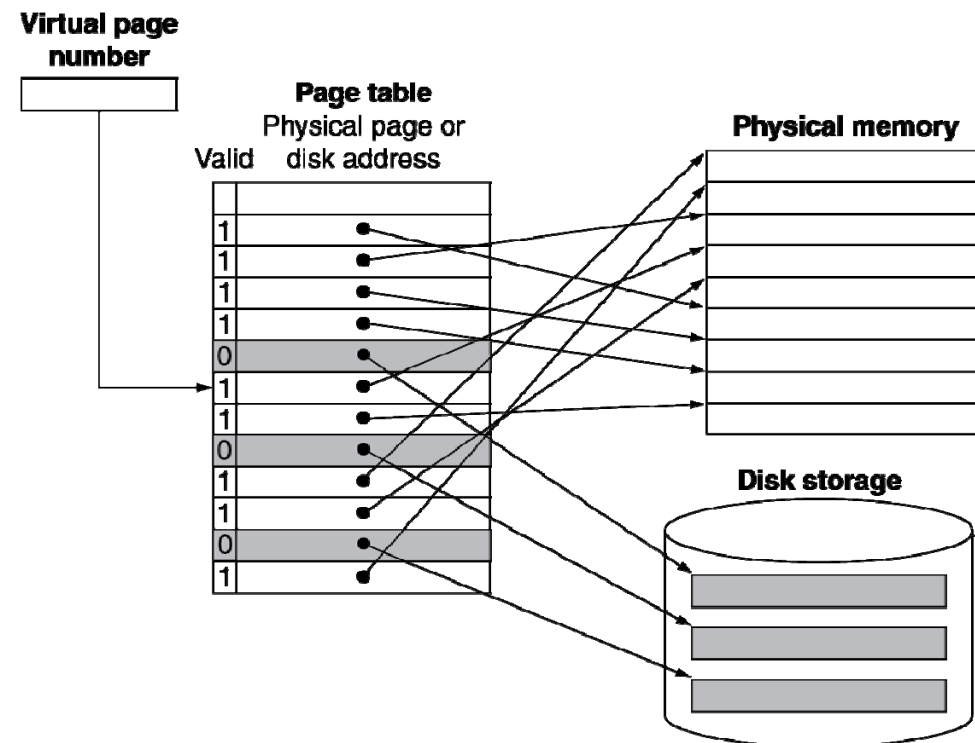


Levels of the Memory Hierarchy



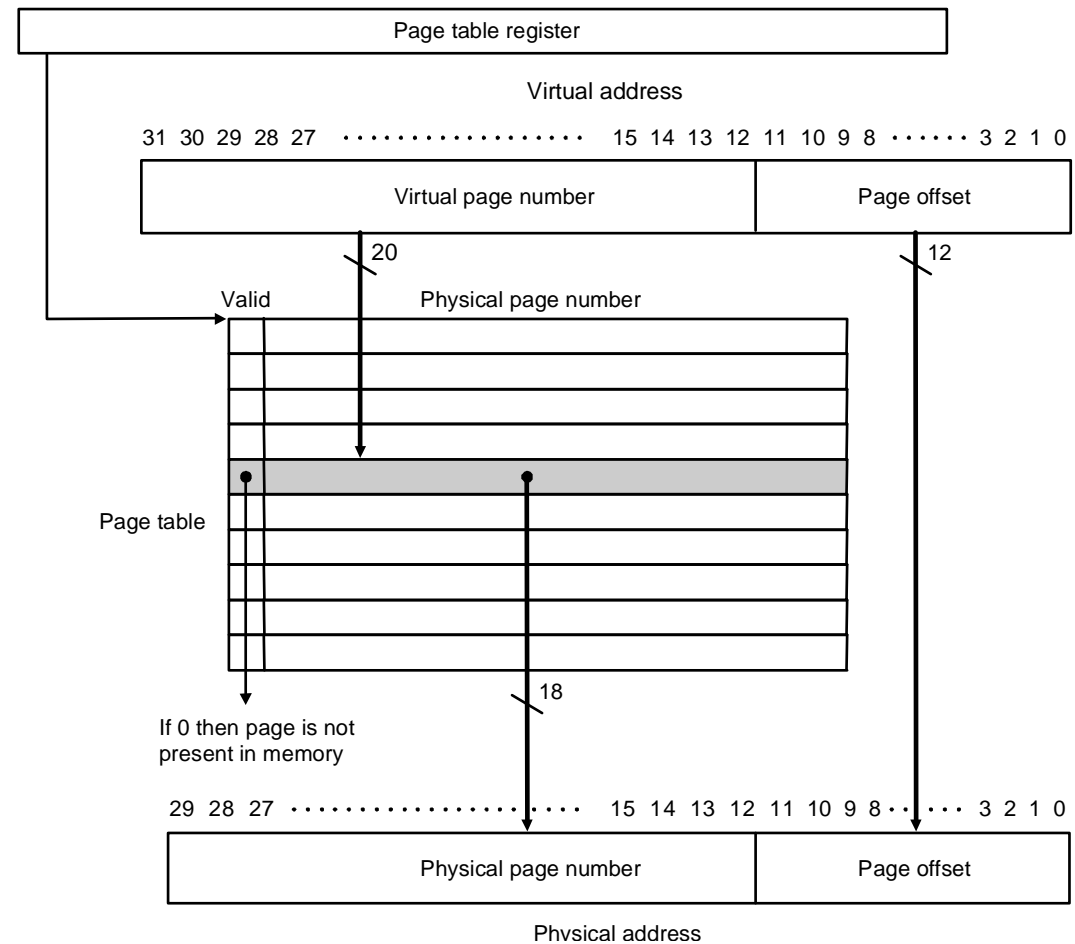
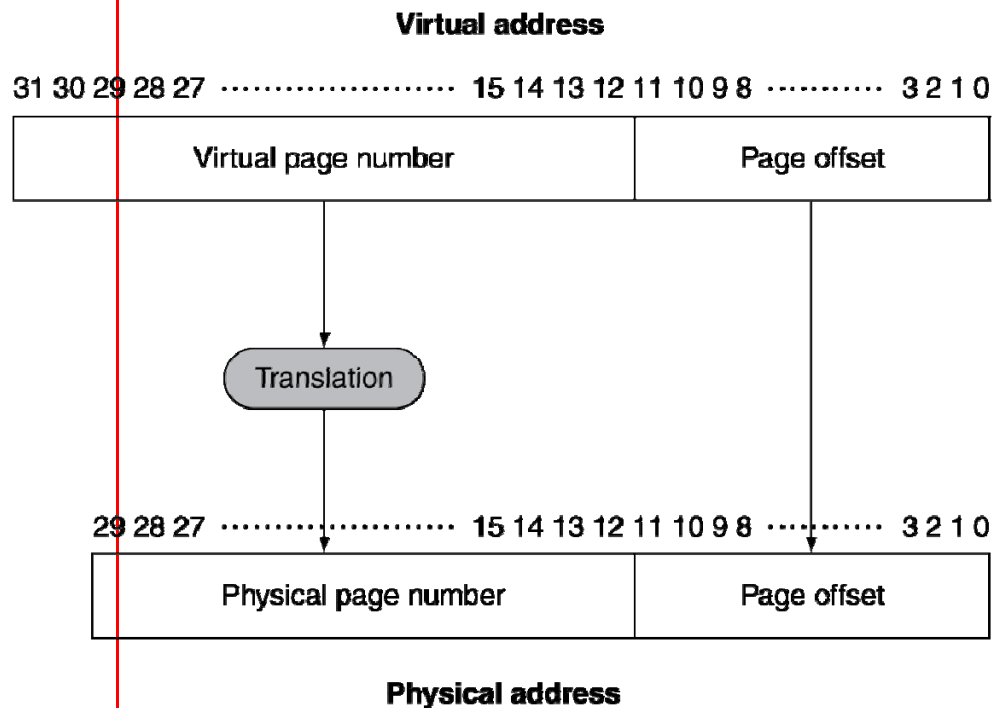
Virtual Memory

- ❑ Use main memory as a “cache” for secondary 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 page vs cache block
 - VM page fault vs cache miss

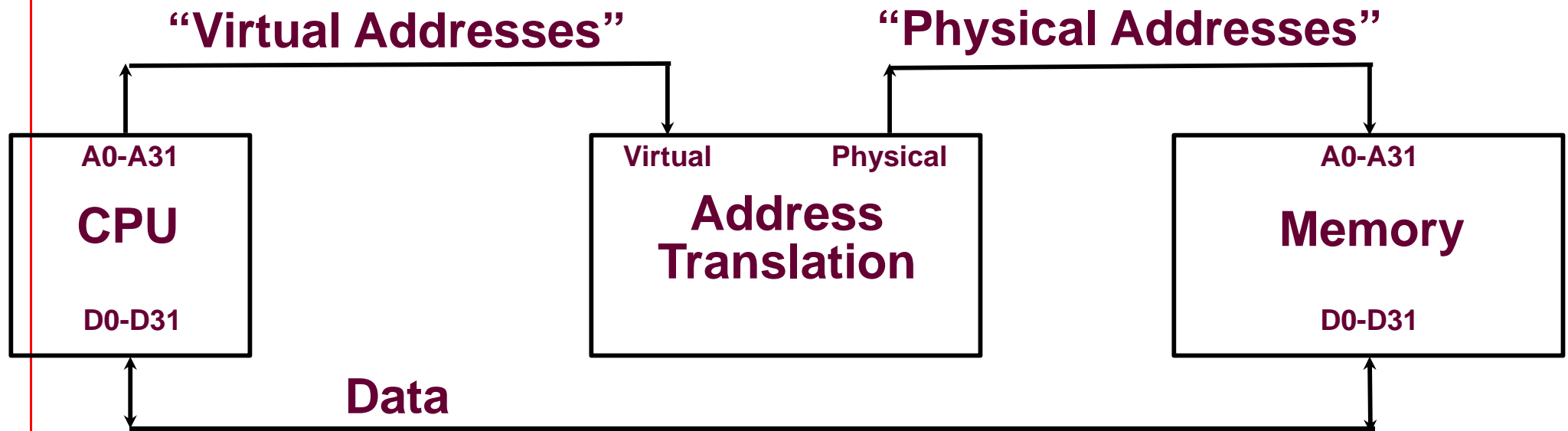


Address Translation for Virtual Memory

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



Virtual memory: memory indirection



- ❑ User programs run in an standardized **virtual** address space
- ❑ **Address Translation** hardware managed by the operating system (OS) maps virtual address to physical memory
- ❑ Hardware supports “modern” OS features: **Protection, Translation, Sharing**

Pages: blocks of memory

- ❑ Page faults: a miss from physical memory
 - very huge penalty - require millions of cycles to process
- ❑ Choice
 - huge miss penalty, thus pages should be fairly large (e.g., 4KB or larger)
 - Organization that reducing page faults is important
 - allow fully associative, LRU is worth the price
 - approximate LRU by using a reference bit
 - can handle the faults in software instead of hardware
 - using write-through is too expensive so we use writeback
 - a dirty bit is added in the page table

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

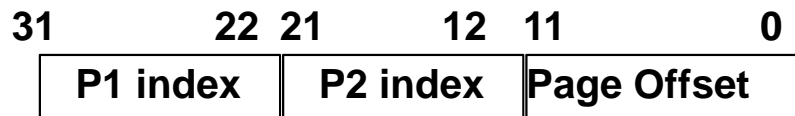
Multiple level of Page tables

A table for 4KB pages for a 32-bit address space has 1M entries

Each process needs its own address space!

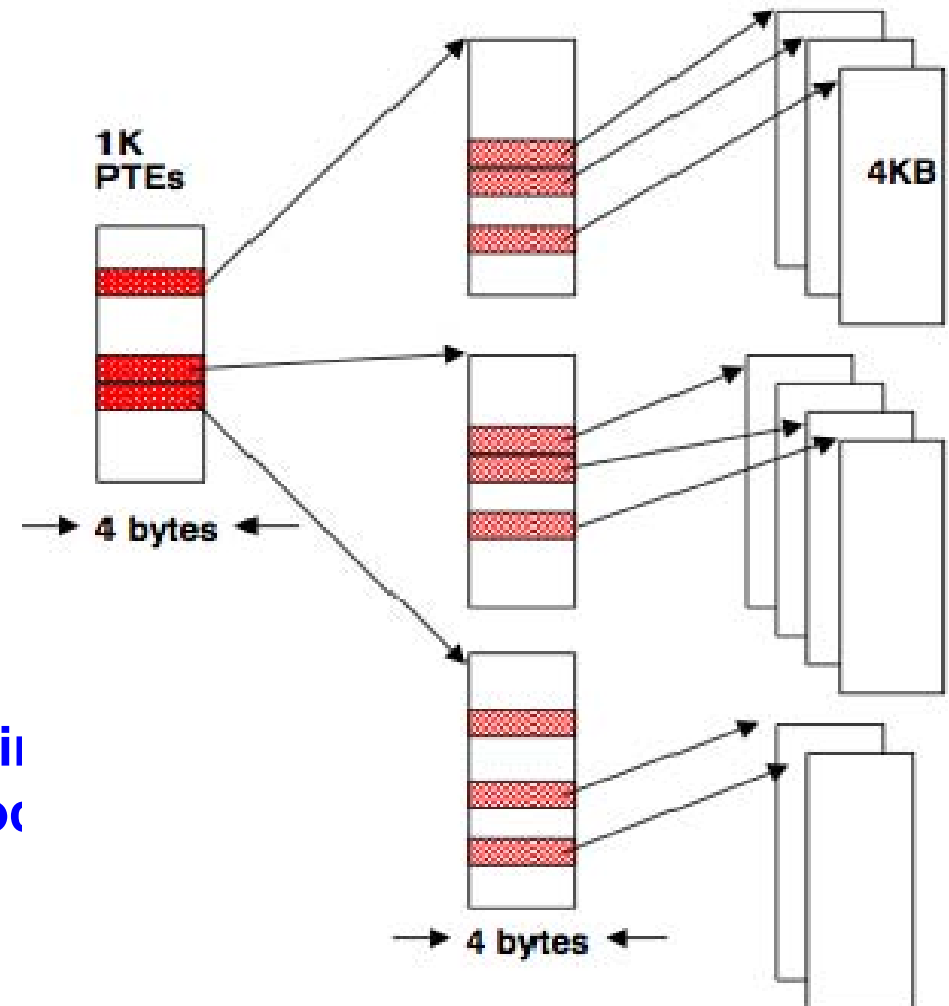
Two-level Page Tables

32 bit virtual address



Top-level table wired in main memory

Subset of 1024 second-level tables in memory; rest are on disk or unallocated



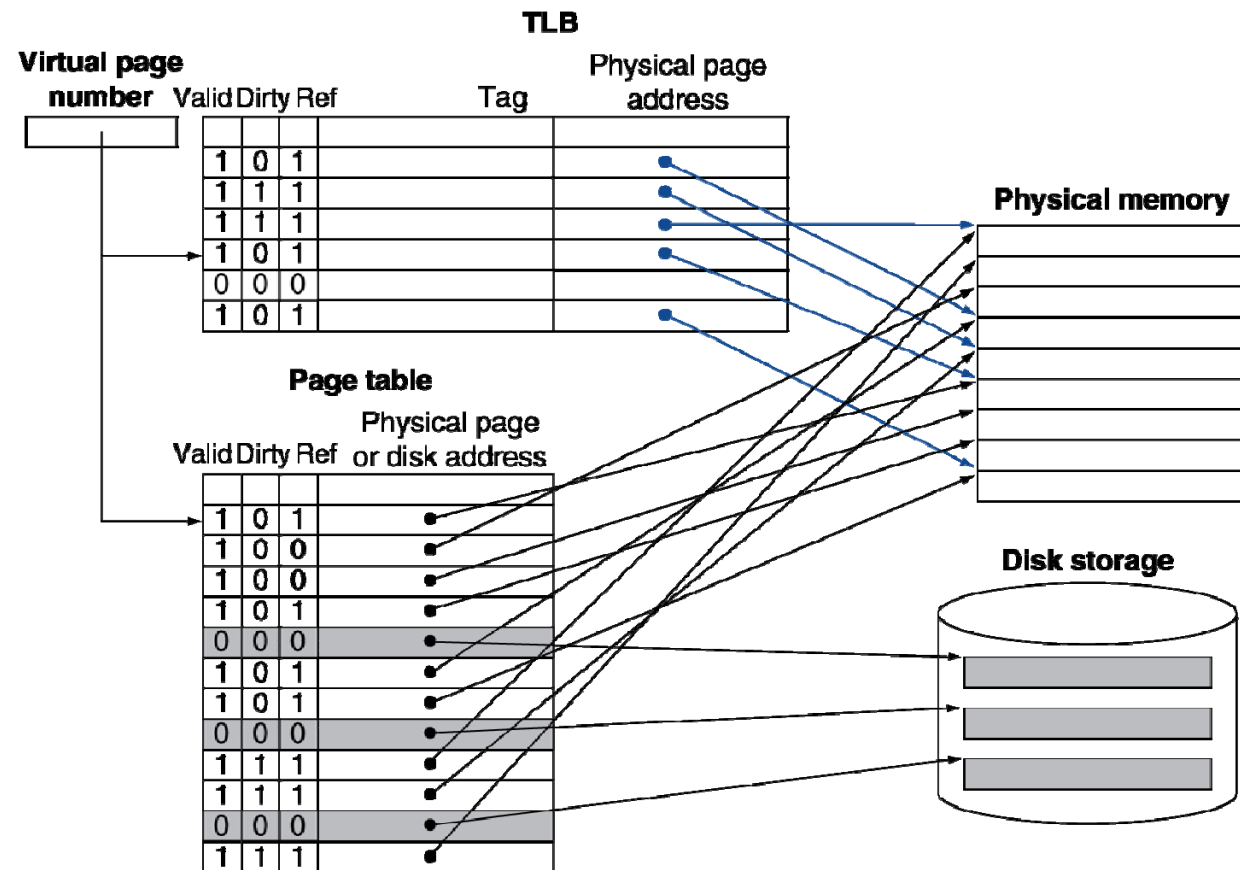
TLB: translation-lookaside buffer

❑ Making address translation fast

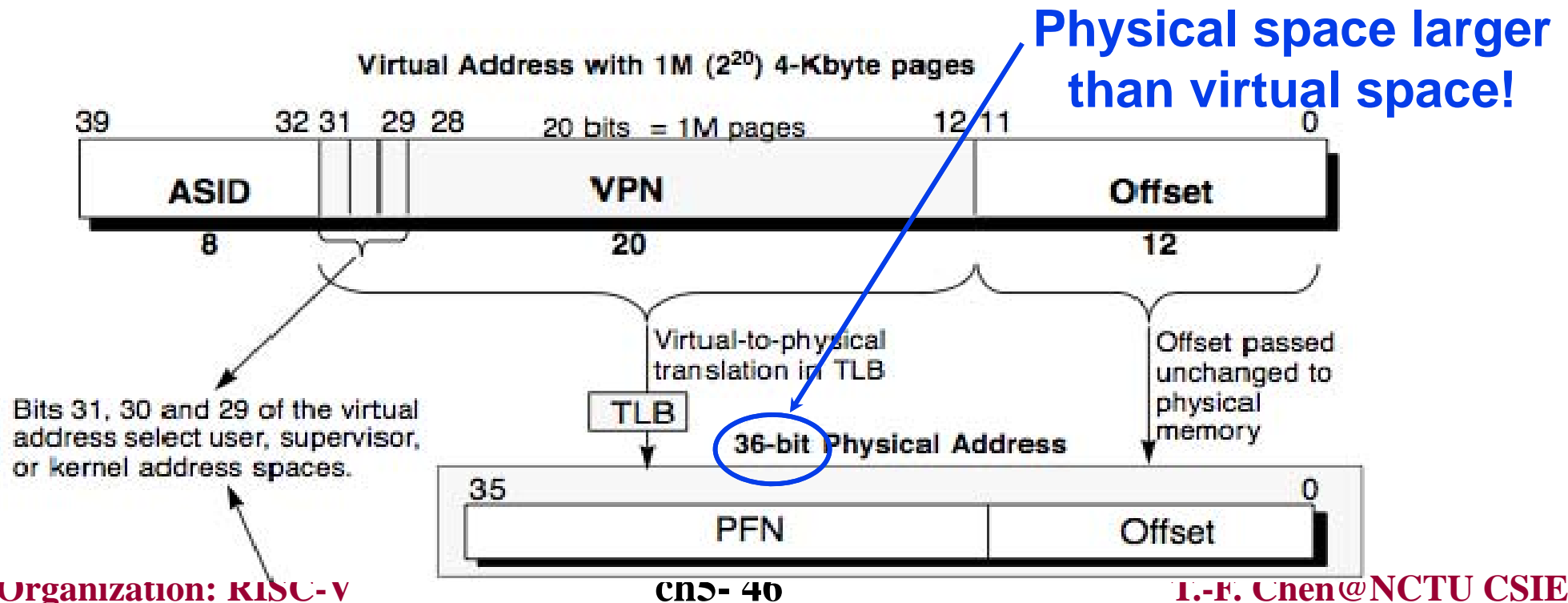
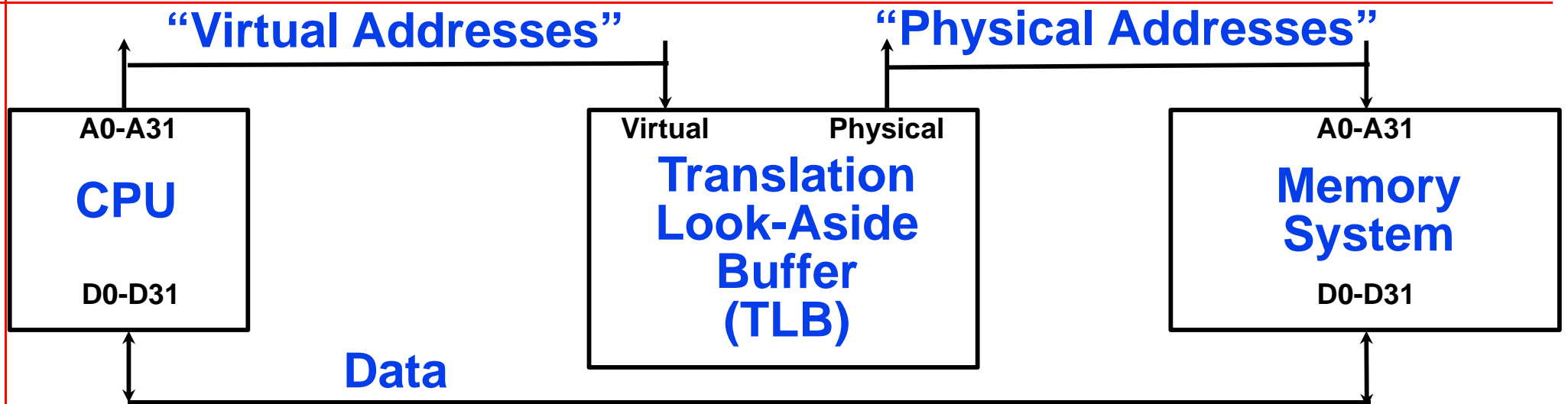
- A special fully-associative cache for recent translation
- Tag: virtual address
- Data: physical page frame number, protection field, valid bit, use bit, dirty bit

❑ Translation

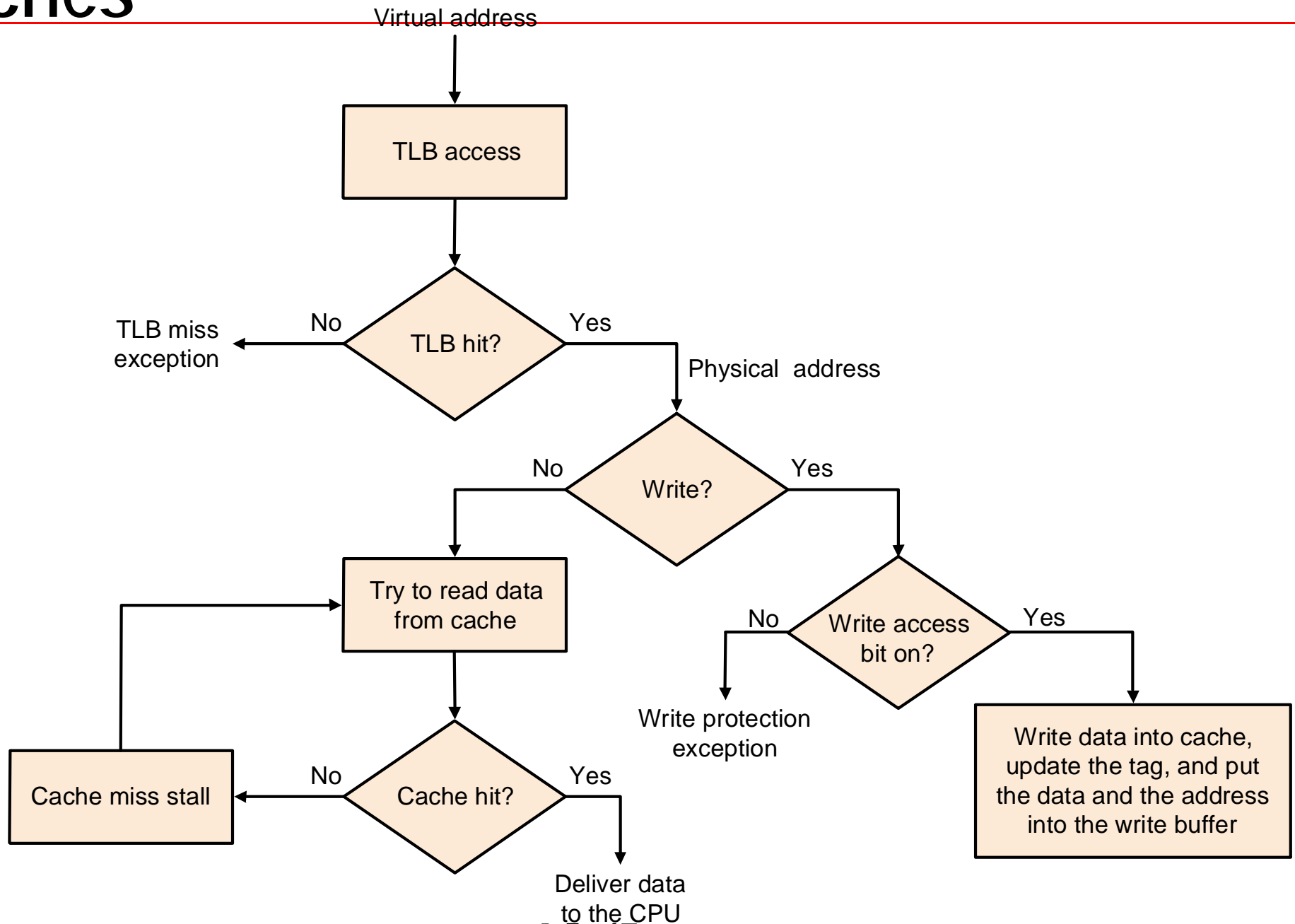
- Send virtual address to all tags
- Check violation
- Matching tag send physical address
- Combine offset to get full physical address



MIPS R4000 TLB: A closer look ...

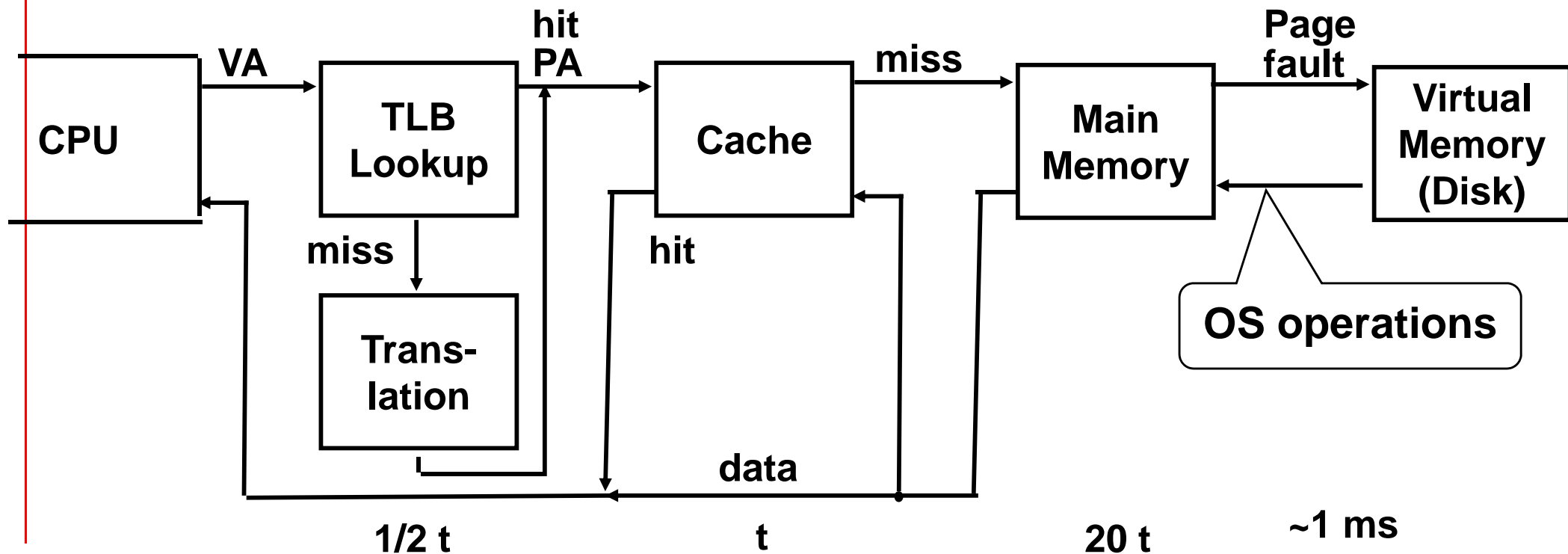


Integrating virtual memory, TLBs and caches



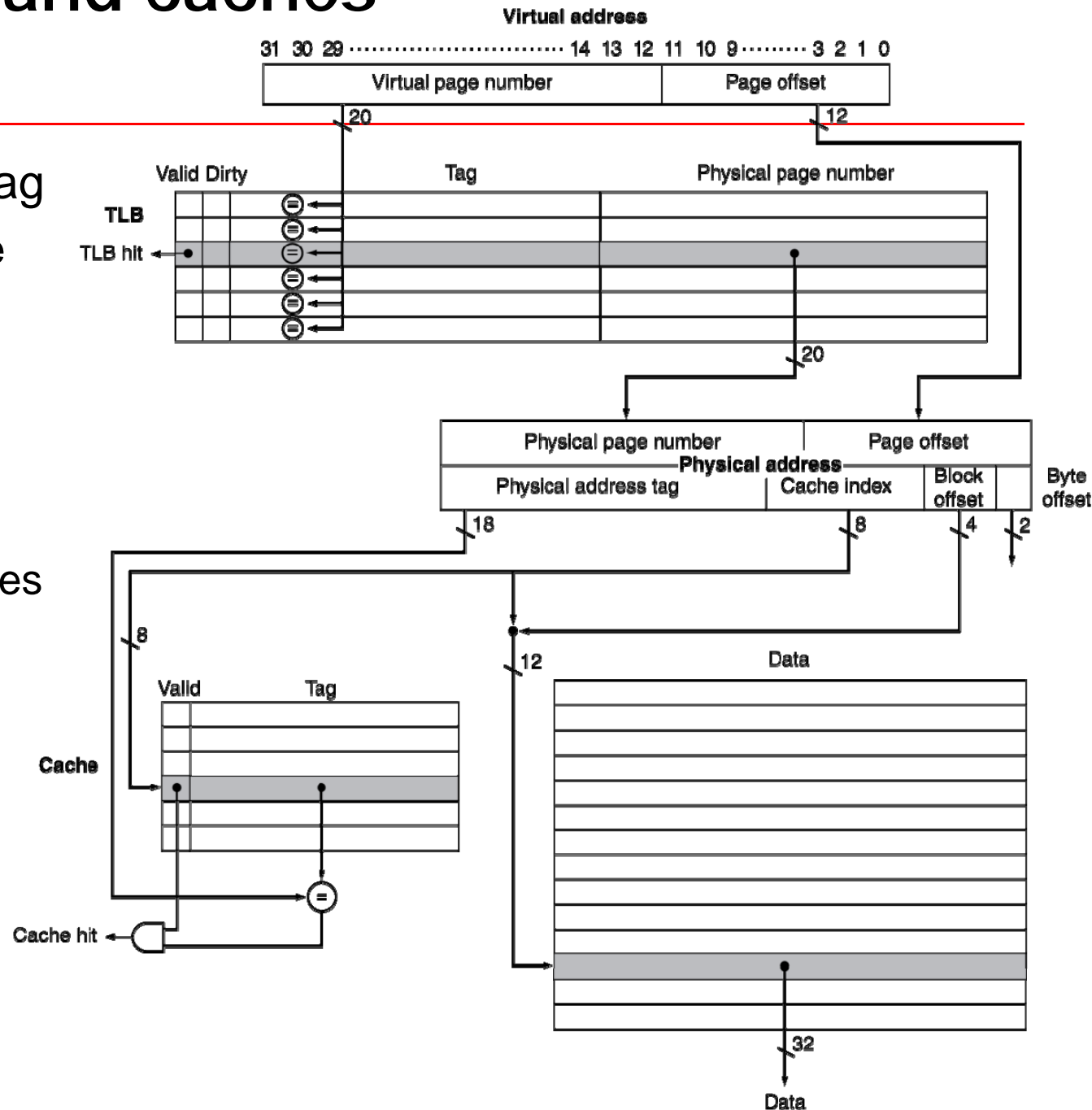
Put everything together

- ❑ It takes an extra memory access to translate VA to PA
- ❑ Be aware of the latency to handle various miss events
- ❑ Compare the differences of policy between cache and virtual memory



Integrating VM, TLBs and caches

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



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 present
- ❑ 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

Multilevel On-Chip Caches

Characteristic	ARM Cortex-A53	Intel Core i7
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	Configurable 16 to 64 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	Two-way (I), four-way (D) set associative	Four-way (I), eight-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, variable allocation policies (default is Write-allocate)	Write-back, No-write-allocate
L1 hit time (load-use)	Two clock cycles	Four clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 2 MiB	256 KiB (0.25 MiB)
L2 cache associativity	16-way set associative	8-way set associative
L2 replacement	Approximated LRU	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	12 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles