

# Chapter 5: Memory Systems

## Issues:

- How long does IF or MEM take?
  - Memory: performance bottleneck
- Cache memory
  - Suppress CPI increase
  - Part of processor design

(problem solving to develop powerful machine)

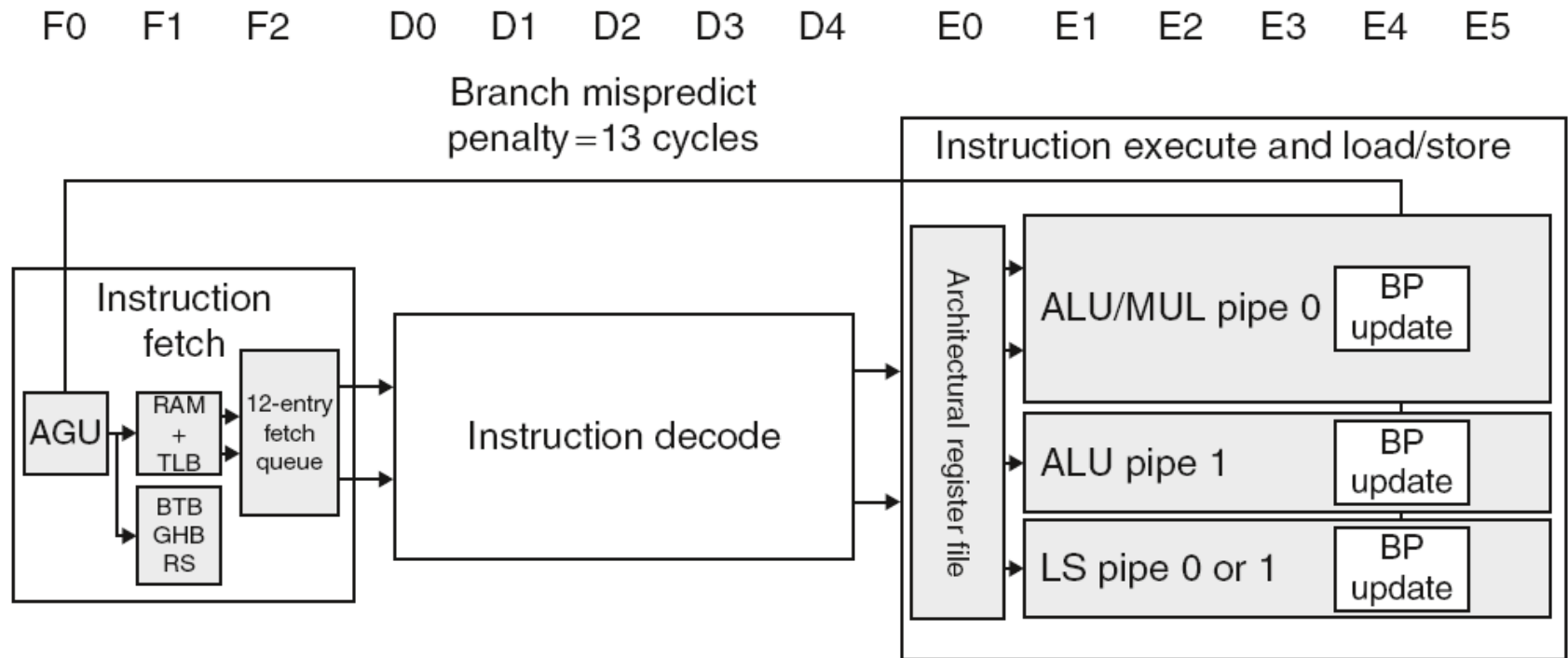
(contain OHPs by H&P, Morgan Kaufmann)

# MIPS Pipeline

- How long does a memory access take?
  - Can IF or MEM be done in one clock cycle?
    - Main memory is slow cache !

Instr	Instr fetch (IF)	Register read (ID)	ALU op (EX)	Memory access (MEM)	Register write (WB)	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

# ARM Cortex-A8 Pipeline (반복)



# Big Picture

---

- ❑ Part 1: what is computer, CSE, computer architecture?
  - Fundamental concepts and principles
- ❑ Part 2: ISA ([externals](#))
  - Ch. 1: performance
    - Exe. time, benchmark, model, RISC, power, multicore
  - Ch. 2: language of computer; ISA
    - What is a good ISA? Today's RISC-style ISA (MIPS)
  - Ch. 3: computer arithmetic
    - Data representation and ALU, ISA – data perspective
- ❑ Part 3: implementation of ISA ([internals](#))
  - Ch. 4: processor
  - Ch. 5: memory system

# Big Picture

---

- ❑ Part 3: implementation of ISA ([internals](#))
  - High-level organization, not circuits design
  - Ch. 4: processor
    - Given ISA, what is a good implementation?
    - Datapath and control, pipelining
  - Ch. 5: memory system design
    - Memory system: physical and virtual
    - Memory hierarchy
    - Cache memory management (main topic)
    - Cache and virtual memory

# Physical and Virtual Memory Systems

physical memory : cpu -> memory embedded computer

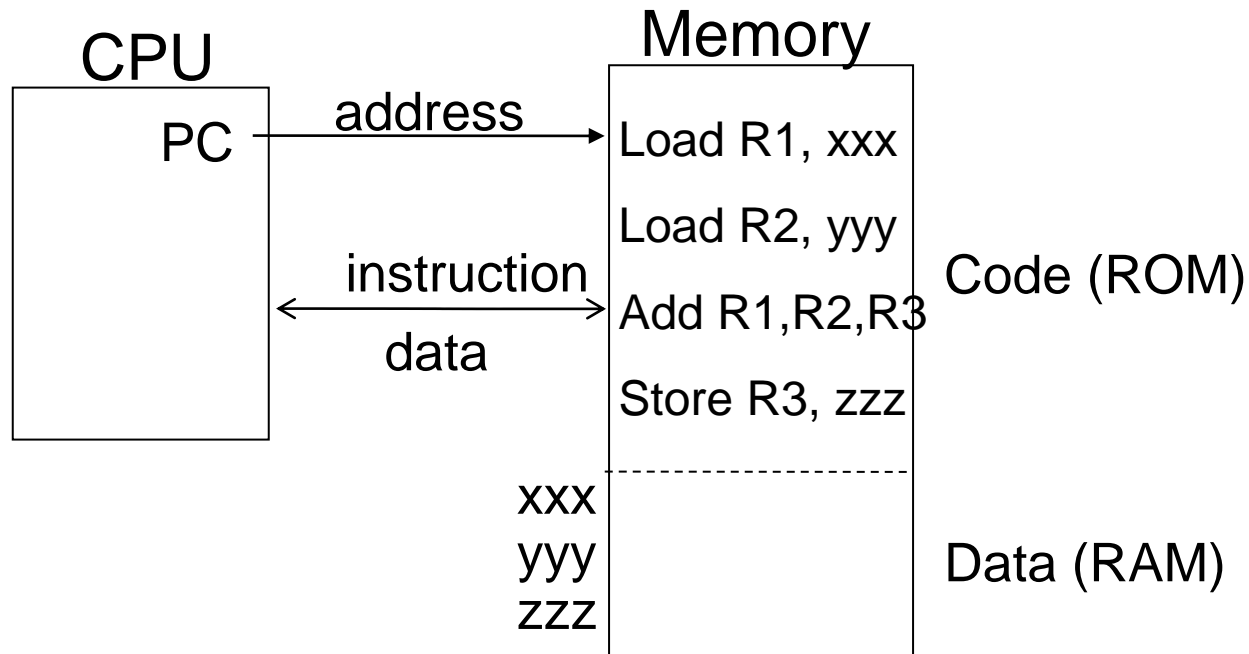
가

virtual memory : cpu -> memory      memory      program size가      !.  
copy program      가

linux concept 1. process (concurrency, virtual memory) 2. file

# Physical Memory Model

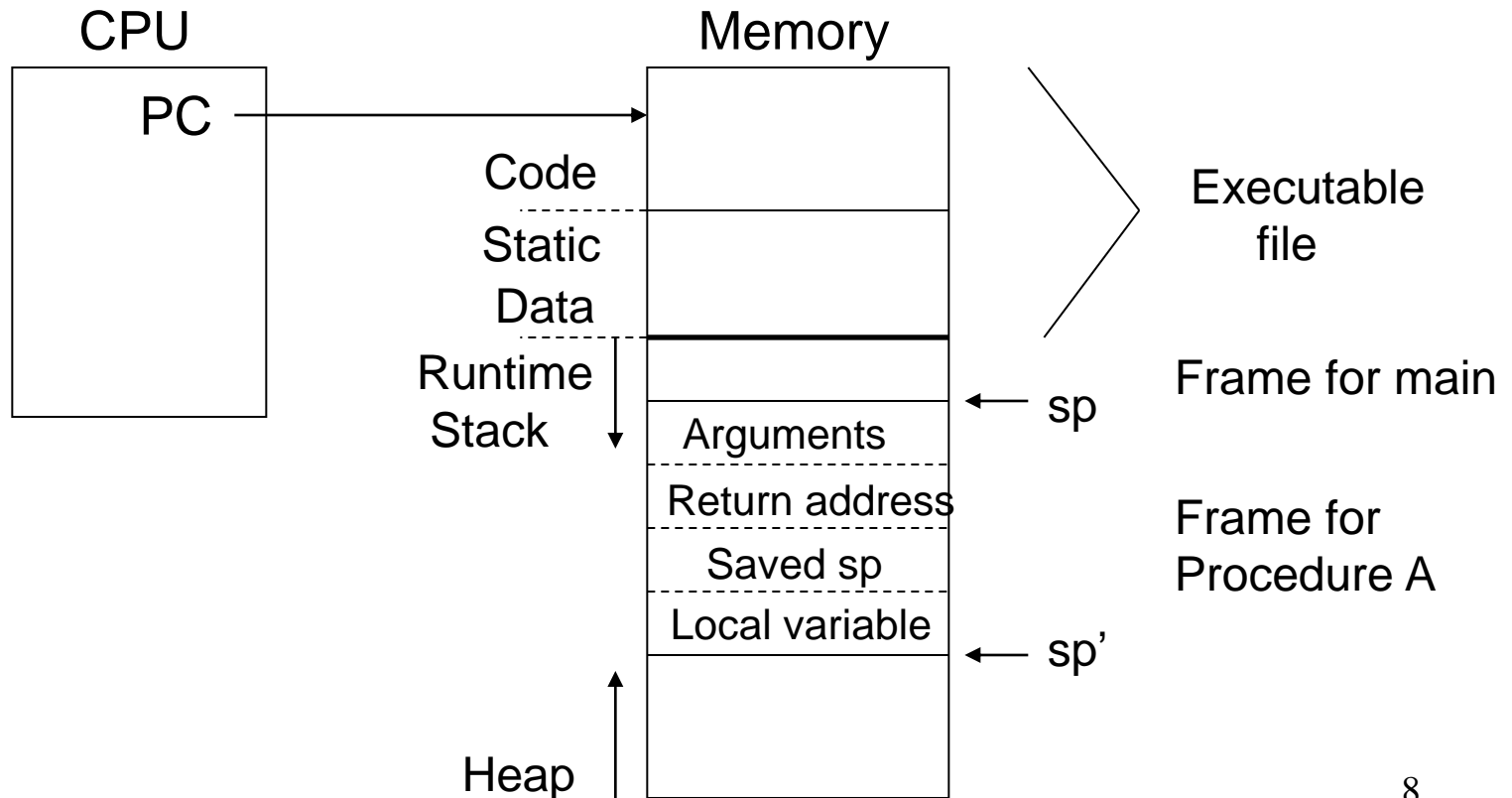
- ❑ Small embedded systems: assembly programming
  - Programmer allocate memory



† Fetch, decode, execute (Von Neumann)

# Physical Memory Model

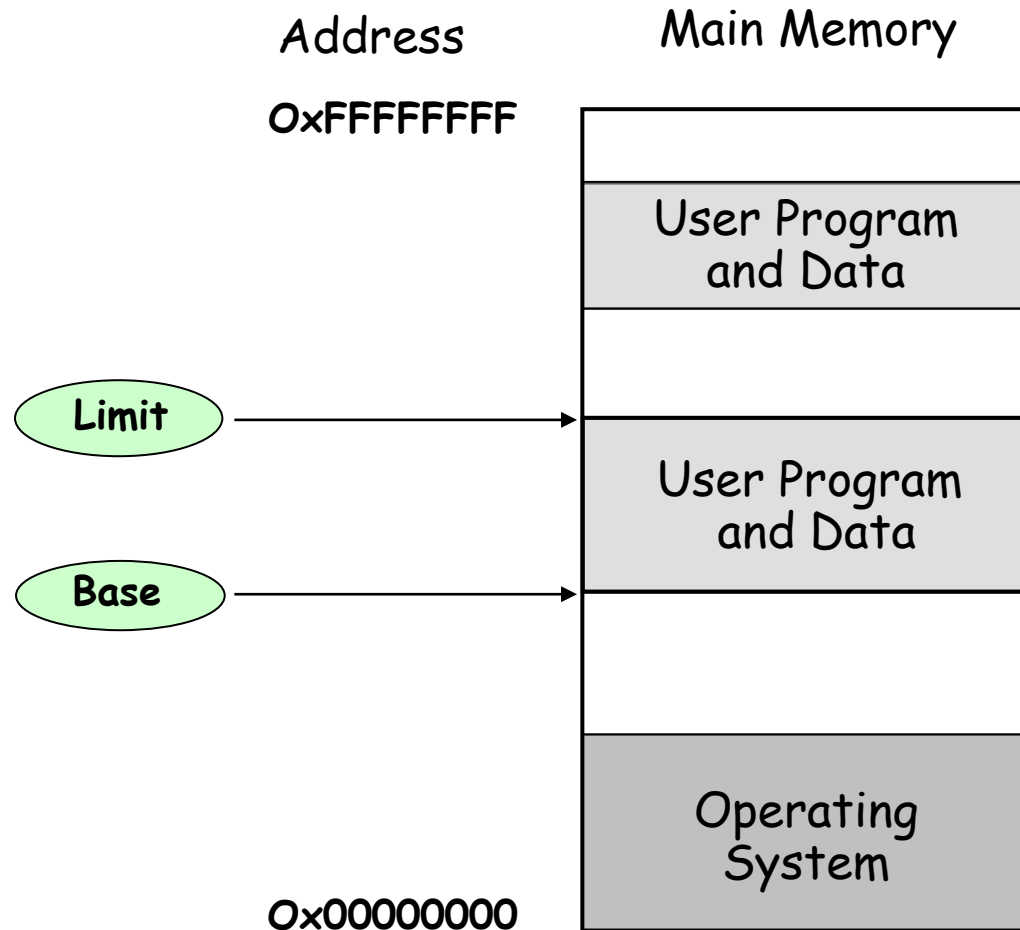
- ❑ Small embedded systems: C programming
  - Compiler allocate memory





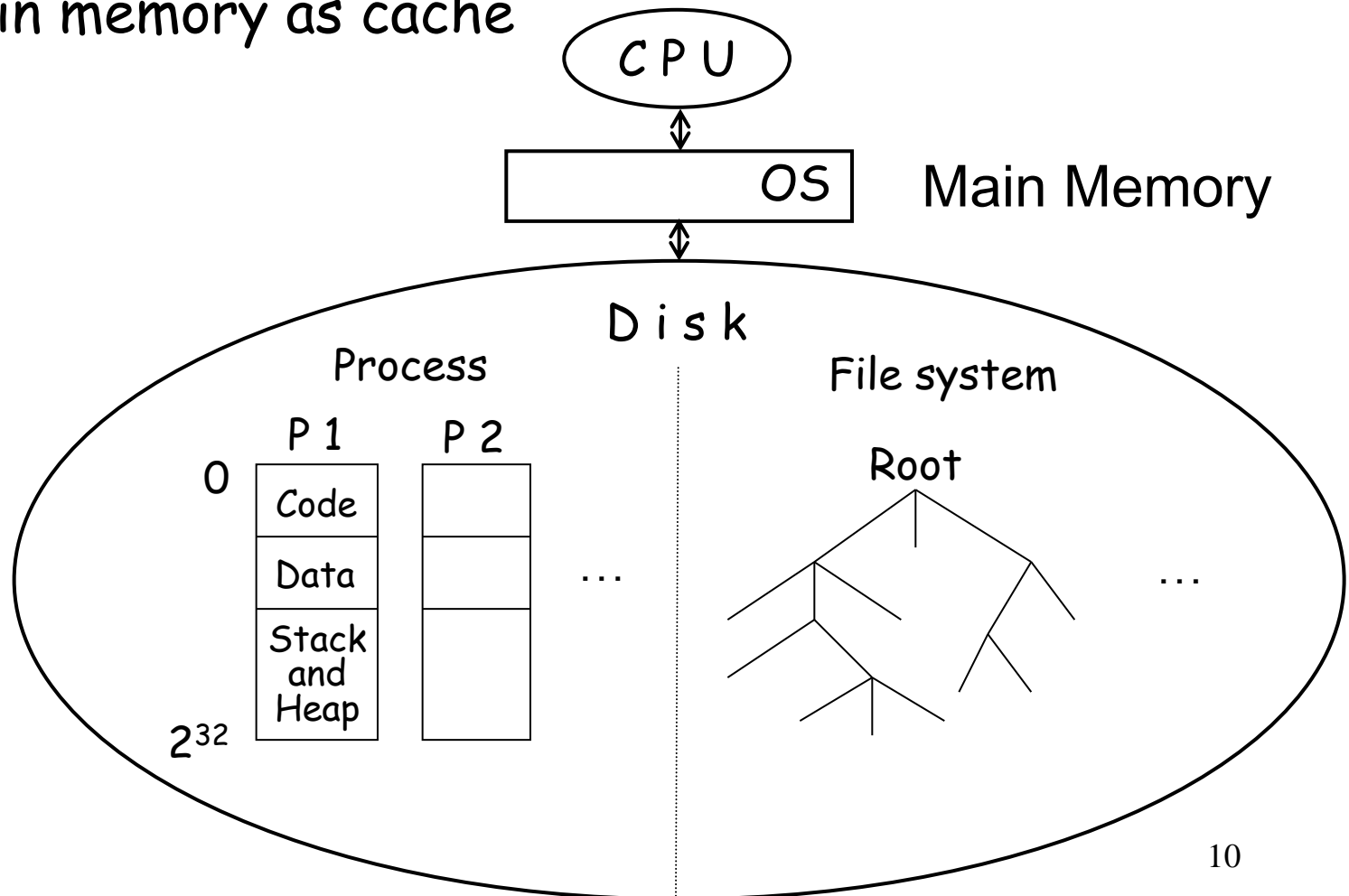
# Physical Memory: General-Purpose Computer

- ❑ Management issues in early OS
  - Size/number of user processes, size of main memory



# Virtual Memory: General-Purpose Computer

- ❑ Decouple main memory and user process address space
- ❑ Use main memory as cache



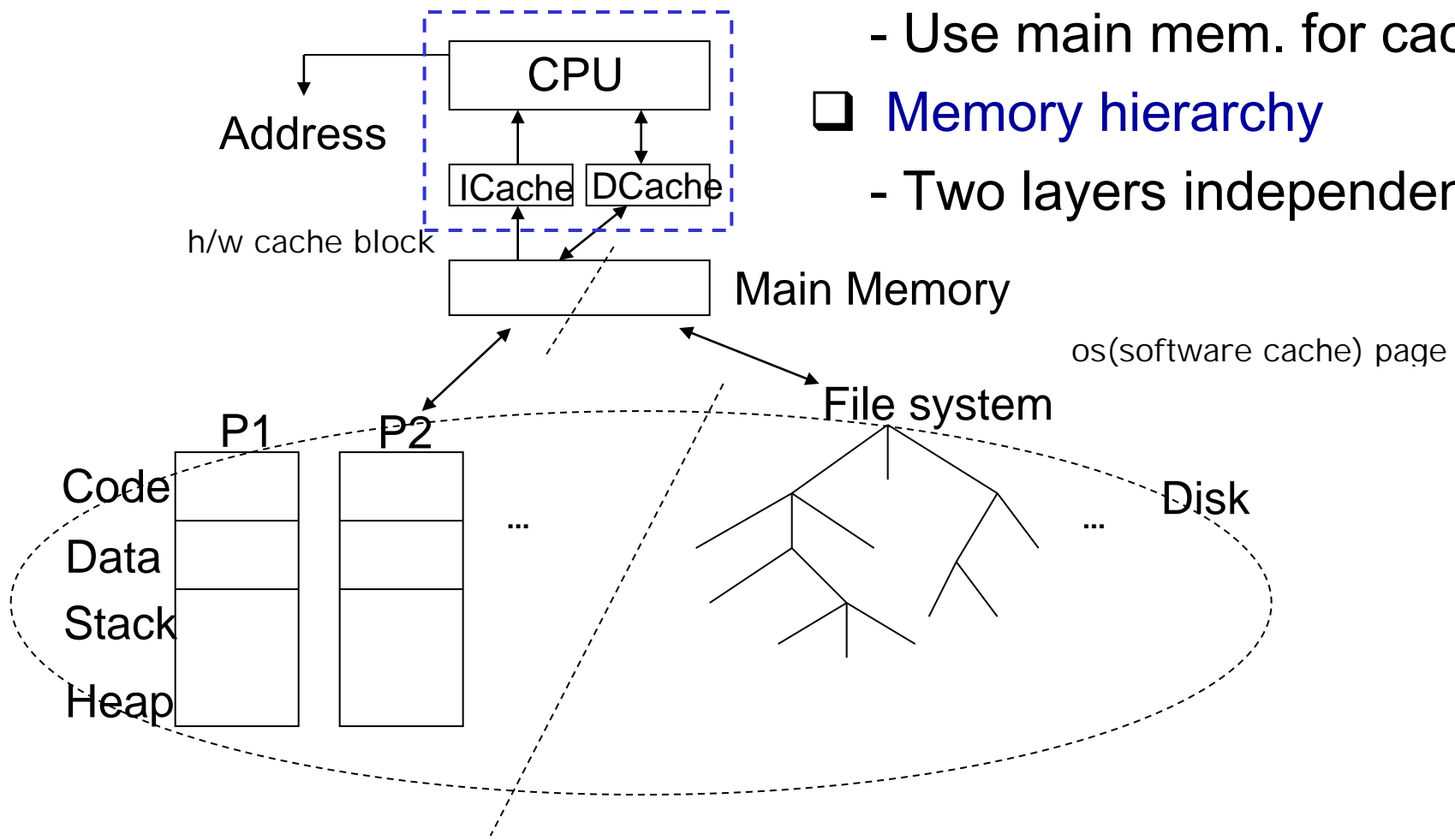
# Motivations for VM

---

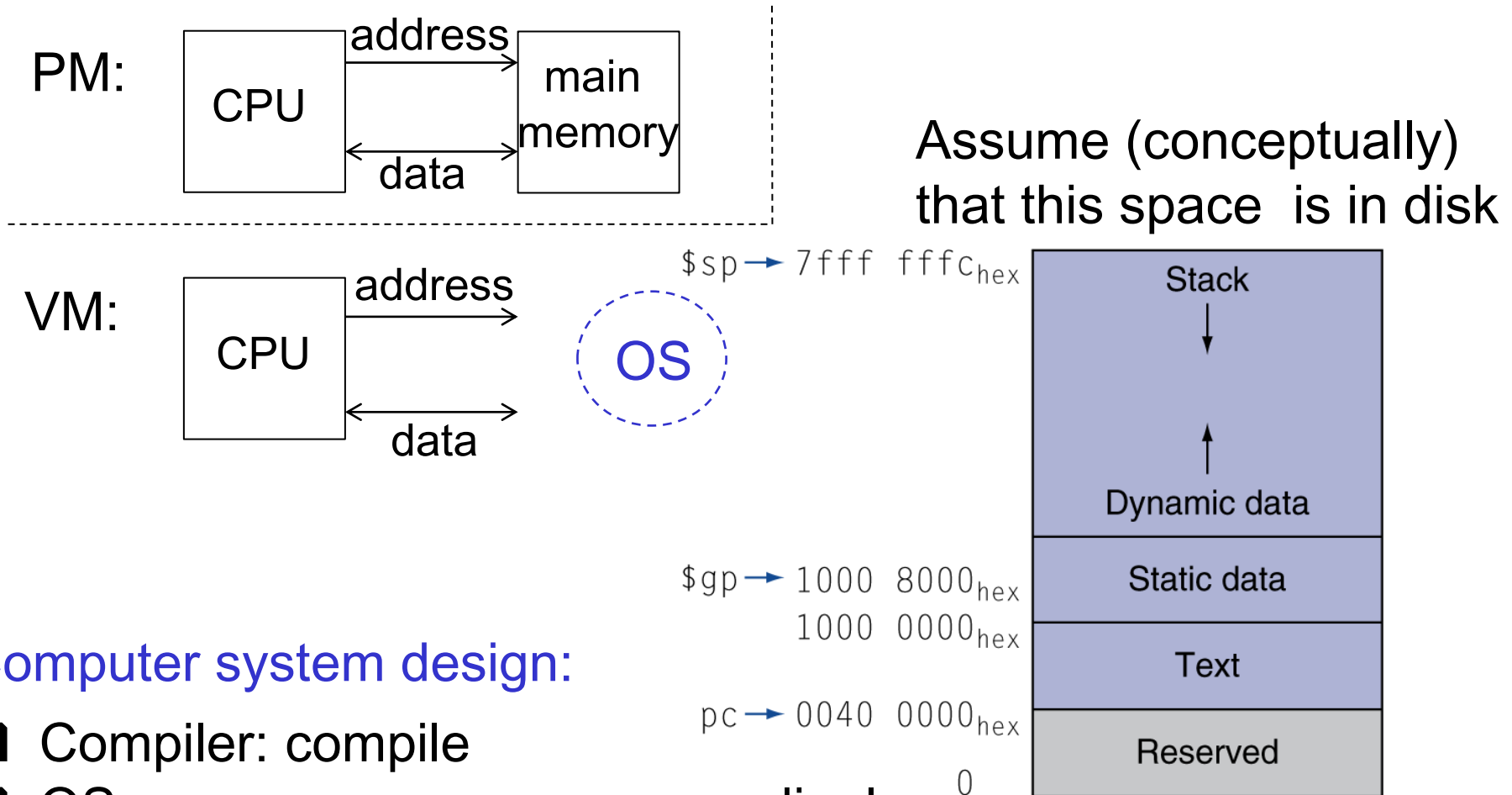
- ❑ Allow single program to exceed the size of main memory
  - Formerly, programmers divide program into pieces
    - Group them into overlays (modules)
  - Serious burden to programmers
    - † Today PM can be larger than VM, but there can be hundreds of processes
- ❑ Decouple main memory and process address space
  - Use main memory as cache
- ❑ Sharing of main memory among multiple programs
  - Efficient and safe (protection issue - more later)
- ❑ Simplify loading of the program for execution

# General-Purpose Computer

- ❑ Speeding up memory access
  - Use cache mem. for caching
  - Use main mem. for caching
- ❑ **Memory hierarchy**
  - Two layers independent



# Virtual Memory: General-Purpose Computer



## Computer system design:

- ❑ Compiler: compile
- ❑ OS: manage processes accordingly
  - Given address, know disk location & main memory location
- ❑ CPU: designed accordingly

# Memory Hierarchy

# Memory Technology

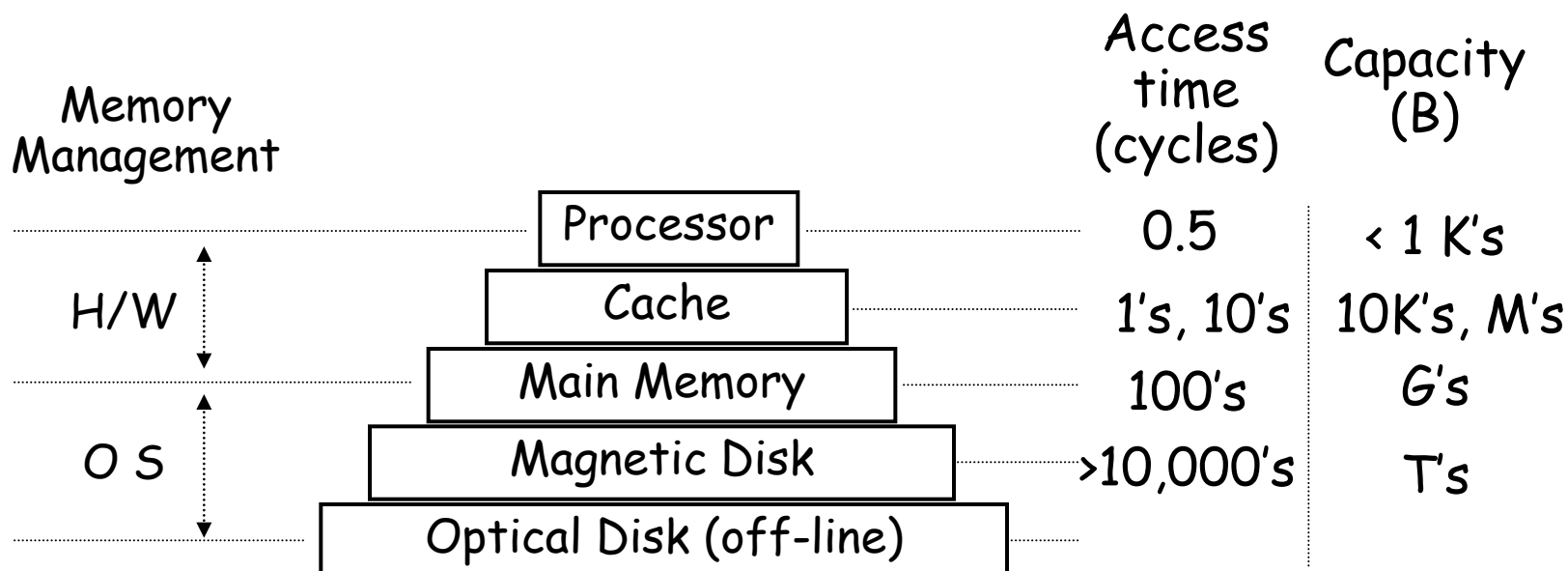
---

- ❑ Ideal memory: capacity, speed, cost
  - Access time of SRAM
  - Capacity and cost/GB of disk

Memory technology	Typical access time	\$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	\$500–\$1000
DRAM semiconductor memory	50–70 ns	\$10–\$20
Flash semiconductor memory	5,000–50,000 ns	\$0.75–\$1.00
Magnetic disk	5,000,000–20,000,000 ns	\$0.05–\$0.10

# Memory Hierarchy

- ❑ Memory: performance bottleneck
- ❑ How to build (illusion of) "ideal memory"
  - Current technology: SRAM, DRAM, disk (flash mem.)
- † Survival of the fittest





# Effect of Cache Memory (미리 보기)

## □ Given

- Main memory access time: 10 clock cycles
- Cache memory access time: 1 cycle
- Miss rate: 0.1

## □ Average memory access time (simple-minded calculation)

- Without cache: 10 cycles
- With cache:  $1 + 0.1 * 10 = 2$  cycles

† How long IF and DM takes

† Caching, pipeline: two key (general) speedup techniques

# Principle of Locality

---

- ❑ What make memory hierarchy a good idea
- ❑ If an item (instruction or data) is referenced
  - Temporal locality: likely to reference it again soon
  - Spatial locality: likely to reference nearby items soon
- ❑ Can you imagine
  - Locality in code
  - Locality in data
- ❑ Given locality, how do we manage memory hierarchy?

# Memory Systems

---

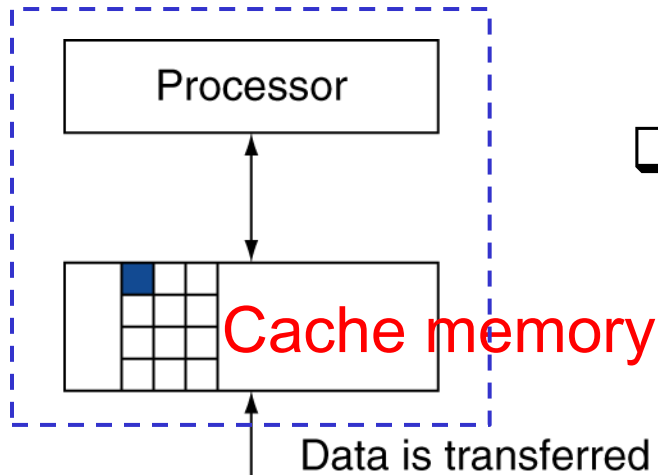
- ❑ Memory management
  - Moving items between two adjacent levels
  - Two different layers: cache and virtual memory
- ❑ When do we move?
  - On-demand (vs. prediction)
- ❑ How to utilize temporal and spatial locality
  - Do you move a single word?
    - Block (or line), page
  - Do you remove block or page right after access?
- ❑ Inclusion property

# 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: Cache Memory Level



- ❑ **Cache block** (aka **line**): unit of copying
  - May be multiple words
- ❑ If CPU-requested data is present in cache memory
  - **Cache hit** 95%
    - **Cache hit ratio**: hits/accesses
- ❑ If requested data is absent
  - **Cache miss**: copy block from main
    - Time taken: **miss penalty** main memory block 가
    - **Cache miss ratio** pipeline
    - = 1 - cache hit ratio

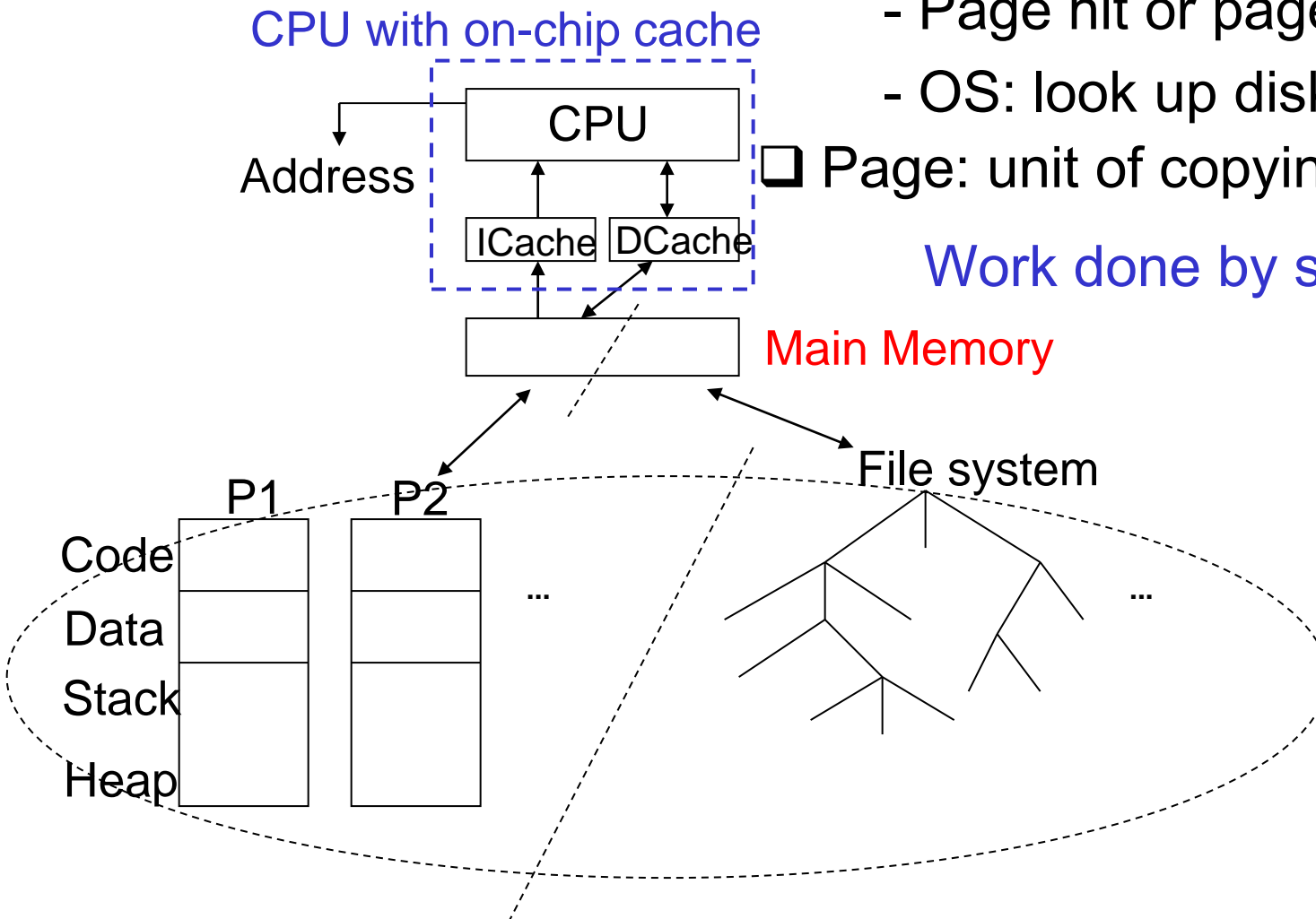
Work done by hardware!  
(OS not know about cache)

Then data block from main memory  
supplied to cache memory

# Memory Hierarchy: Main Memory Level

- ❑ On cache miss, see main memory
  - Page hit or page miss (or fault)
  - OS: look up disk on page miss
- ❑ Page: unit of copying

Work done by software (OS)!



# Memory Management

---

## ❑ Cache memory management (Architecture topic)

- Cache part of main memory
- Implemented by hardware: fast, simple

† May think it as part of processor, i.e., on-chip cache

- Hardware accelerator: OS not know about it

## ❑ Virtual memory management (OS topic)

- Use main memory as cache for disk
- Implemented by software
  - Disk access is already slow (10ms)

† Principles (caching, locality, management) same for both

- Usage: independent of each other

# Cache Memory



# Cache Memory

---

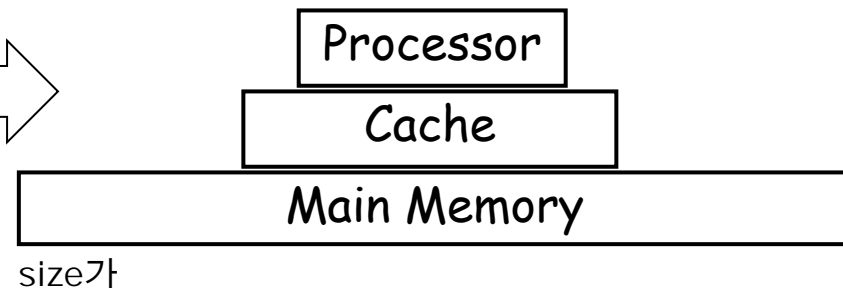
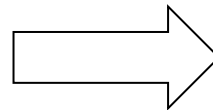
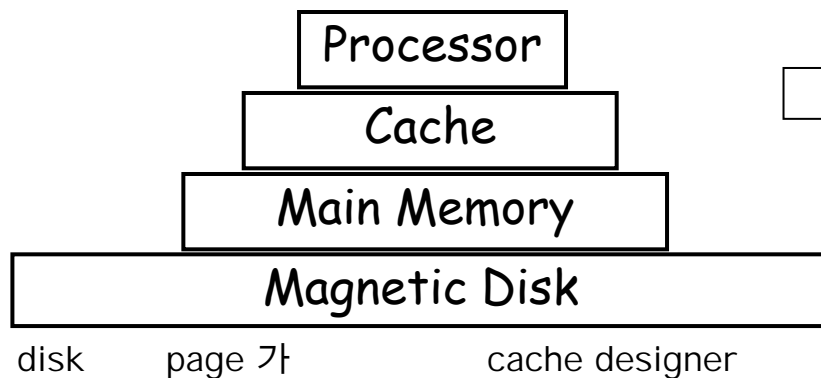
- ❑ Forget about virtual memory for now
  - Cache memory: independent of virtual memory
    - † Will show how it works with VM later
- ❑ Focus on how to cache parts of main memory
  - To speed-up main memory access
- ❑ This mean that we assume main memory is ideal
  - Assume that size of main memory is infinite

# Cache (CPU) Designer's Perspective

- ❑ Page fault and associated performance loss
  - Not something that cache designer can control
  - OS, I/O design issue (not cache memory design issue)

† Separation of concern

† Cache memory designer's view



† Like physical memory system with everything in main<sup>26</sup>

# Cache Memory in Operation

- ❑ Read hits
  - This is what we want
- ❑ Read misses
  - Stall CPU, fetch block from memory, restart
- ❑ Write hits
  - Update data in cache and memory (write-through)
  - Update data in cache (write-back)
- ❑ Write misses
  - Stall CPU, fetch block from memory, write, restart

# Cache Memory Performance

---

## ❑ Hit and miss

- Cache hit/miss, page hit/miss
- Hit rate (or ratio), miss rate

## ❑ Average access time = hit time\*hit rate + (hit time+miss penalty)\*miss rate

- Hit time + miss rate \* miss penalty
  - From perspective of cache access
  - Can you see that cache is a good idea?
  - Can you see increase in CPI due to cache miss?

## ❖ Performance model and three key factors (more later)

# Cache Memory Performance

## □ Given

- Cache memory access time: 1 clock cycle
- Miss penalty: 10 cycles, miss rate: 0.1

## □ Average memory access time

- $1 + 0.1 * 10 = 2$  cycles
- At every IF or DM, pipeline stall 1 cycle

## □ CPI increase due to memory

- Frequency of load and store: 20%
- CPI: 1  $\rightarrow$  2.2 (80%: lose 1 cycle<sup>IF</sup>, 20%: lose 2 cycles<sup>IF/MEM</sup>)

$$2.2 = 0.8 * 2 + 0.2 * 3$$

## □ What if there is no cache memory?

## □ Memory is slow - much more damaging than hazards<sup>29</sup>

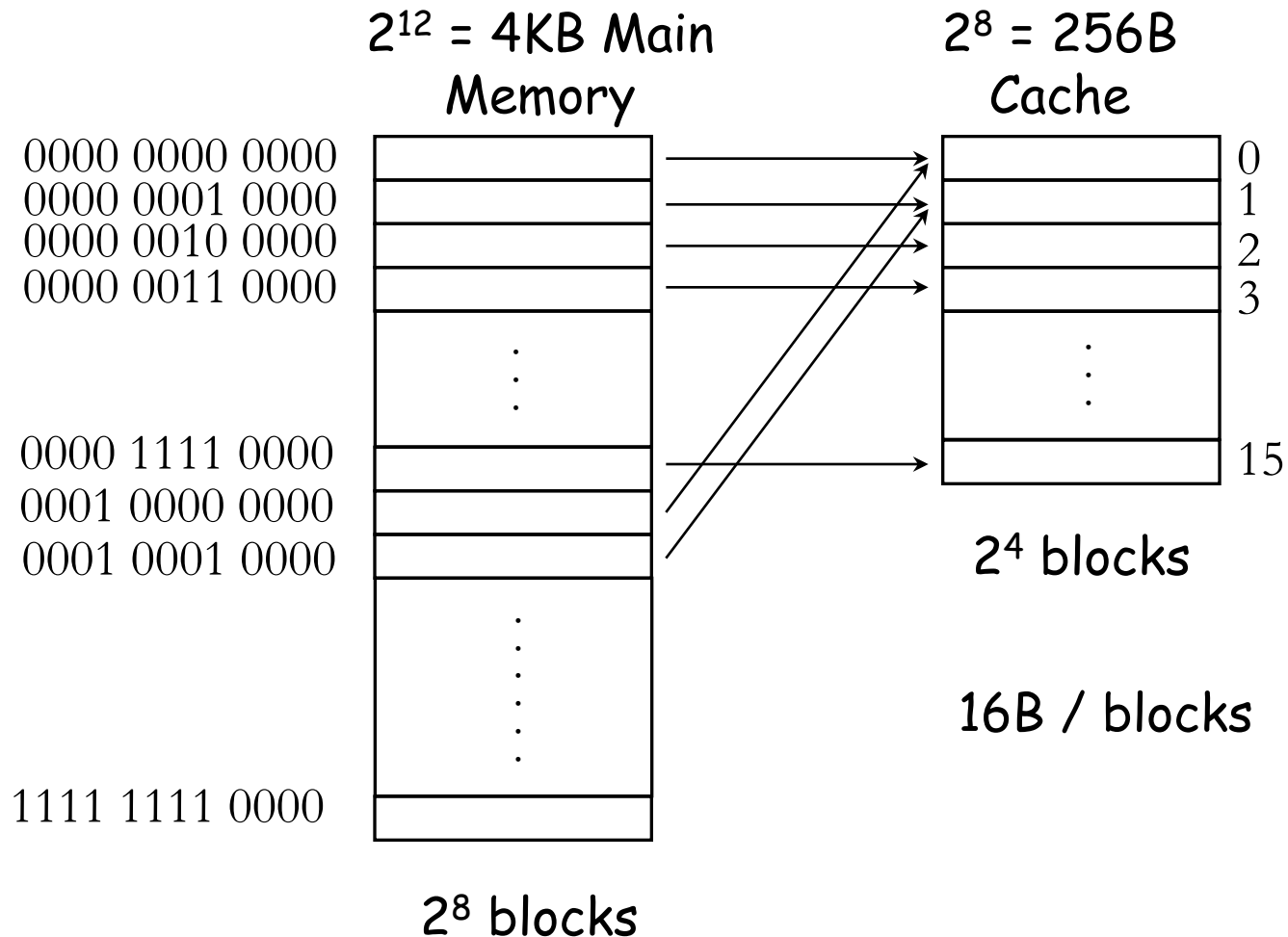
# Cache Memory: Structure and Operation

# Cache Memory Design

---

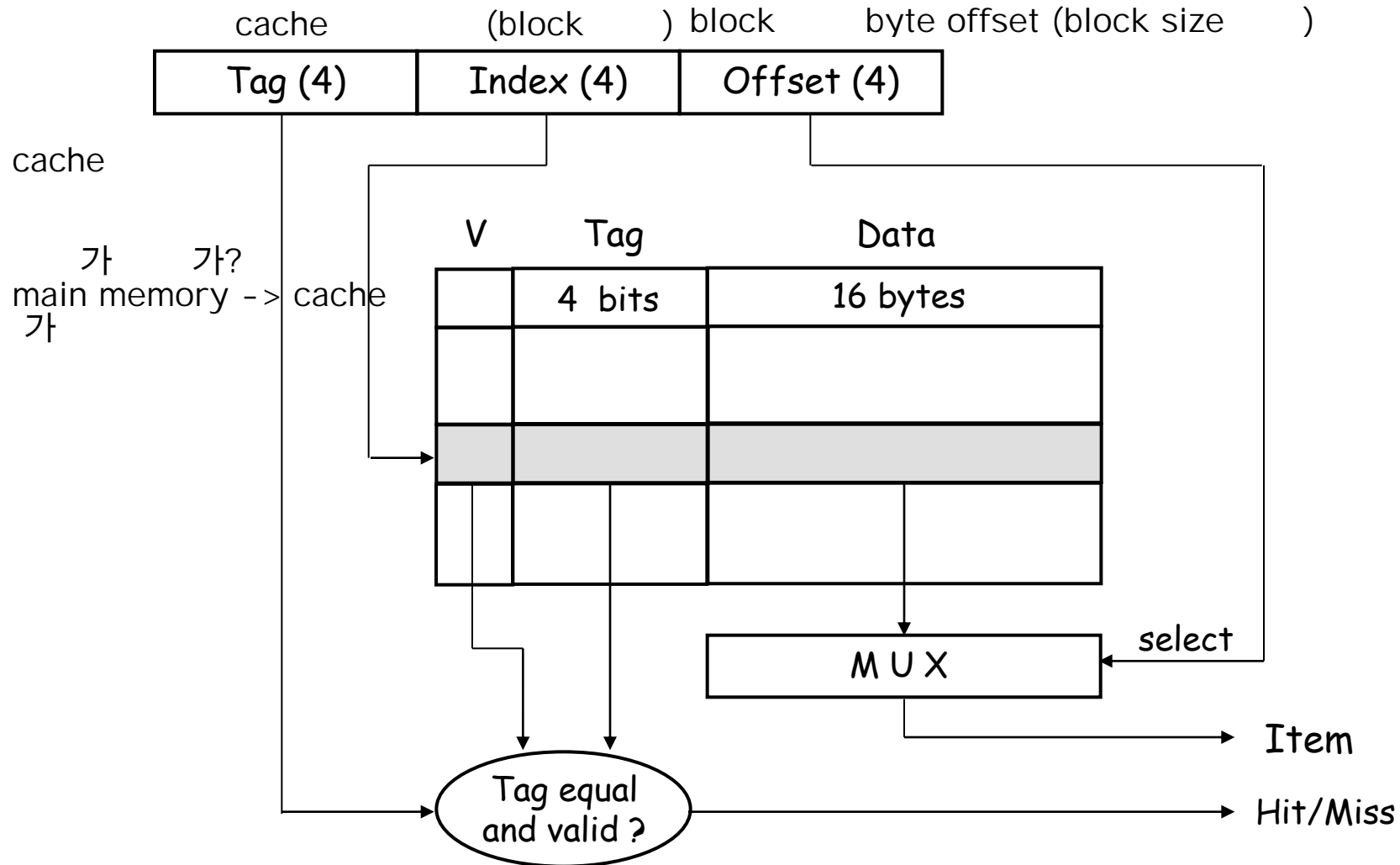
- ❑ Cache memory: smaller than main memory
  - Where to place a block (placement issue)
  - How to find it later (identification issue)

# Direct Map Cache: Placement





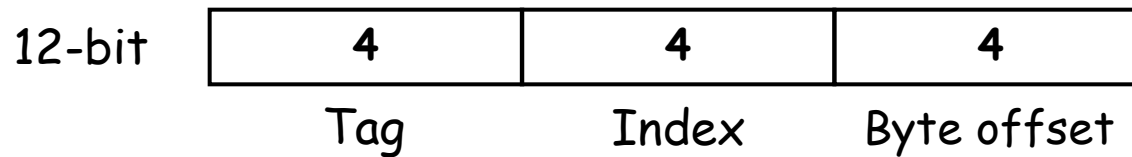
# Direct Map Cache: Identification



# Direct Map Cache

---

- ❑ How do we use an address?



- ❑ 16-to-1 compaction
  - How do we know it is a right block

- ❑ Valid bit

V	Tag	Data
	4 bits	16 bytes
⋮	⋮	⋮

- ❑ Cache Utilization:  $(16 \times 8) / (16 \times 8 + 4 + 1)$

# Quiz

---

❑ Given all others do not change

- What if we reduce the size of cache memory to half?  
Tag(5) Index(3) offset(4)
  - How many bits for tag, index, and byte offset
- What if we reduce the block size to half?  
Tag(4) Index(5) offset(3)
- What if the size of main memory is doubled?  
Tag(5) Index(4) offset(4)

❑ Now let's look into a small cache memory system:

- You become a human cache simulator
  - Address trace, cache configuration → cache hit rate

† Then you can design a software cache simulator

# Quiz

- 8-byte direct map cache, 5-bit address, 1 byte/block (tag: 2 bits, index: 3 bits, byte offset: 0 bit)

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	$10110_{\text{two}}$	miss (7.6b)	$(10\mathbf{110}_{\text{two}} \bmod 8) = \mathbf{110}_{\text{two}}$
26	$11010_{\text{two}}$	miss (7.6c)	$(11\mathbf{010}_{\text{two}} \bmod 8) = \mathbf{010}_{\text{two}}$
22	$10110_{\text{two}}$	hit	$(10\mathbf{110}_{\text{two}} \bmod 8) = \mathbf{110}_{\text{two}}$
26	$11010_{\text{two}}$	hit	$(11\mathbf{010}_{\text{two}} \bmod 8) = \mathbf{010}_{\text{two}}$
16	$10000_{\text{two}}$	miss (7.6d)	$(10\mathbf{000}_{\text{two}} \bmod 8) = \mathbf{000}_{\text{two}}$
3	$00011_{\text{two}}$	miss (7.6e)	$(00\mathbf{011}_{\text{two}} \bmod 8) = \mathbf{011}_{\text{two}}$
16	$10000_{\text{two}}$	hit	$(10\mathbf{000}_{\text{two}} \bmod 8) = \mathbf{000}_{\text{two}}$
18	$10010_{\text{two}}$	miss (7.6f)	$(10\mathbf{010}_{\text{two}} \bmod 8) = \mathbf{010}_{\text{two}}$

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

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	Y	11 <sub>two</sub>	Memory (11010 <sub>two</sub> )
011	N		
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory (10110 <sub>two</sub> )
111	N		

c. After handling a miss of address (11010<sub>two</sub>)

Index	V	Tag	Data
000	Y	10 <sub>two</sub>	Memory (10000 <sub>two</sub> )
001	N		
010	Y	11 <sub>two</sub>	Memory (11010 <sub>two</sub> )
011	Y	00 <sub>two</sub>	Memory (00011 <sub>two</sub> )
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory (10110 <sub>two</sub> )
111	N		

e. After handling a miss of address (00011<sub>two</sub>)

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory(10110 <sub>two</sub> )
111	N		

b. After handling a miss of address (10110<sub>two</sub>)

Index	V	Tag	Data
000	Y	10 <sub>two</sub>	Memory (10000 <sub>two</sub> )
001	N		
010	Y	11 <sub>two</sub>	Memory (11010 <sub>two</sub> )
011	N		
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory (10110 <sub>two</sub> )
111	N		

d. After handling a miss of address (10000<sub>two</sub>)

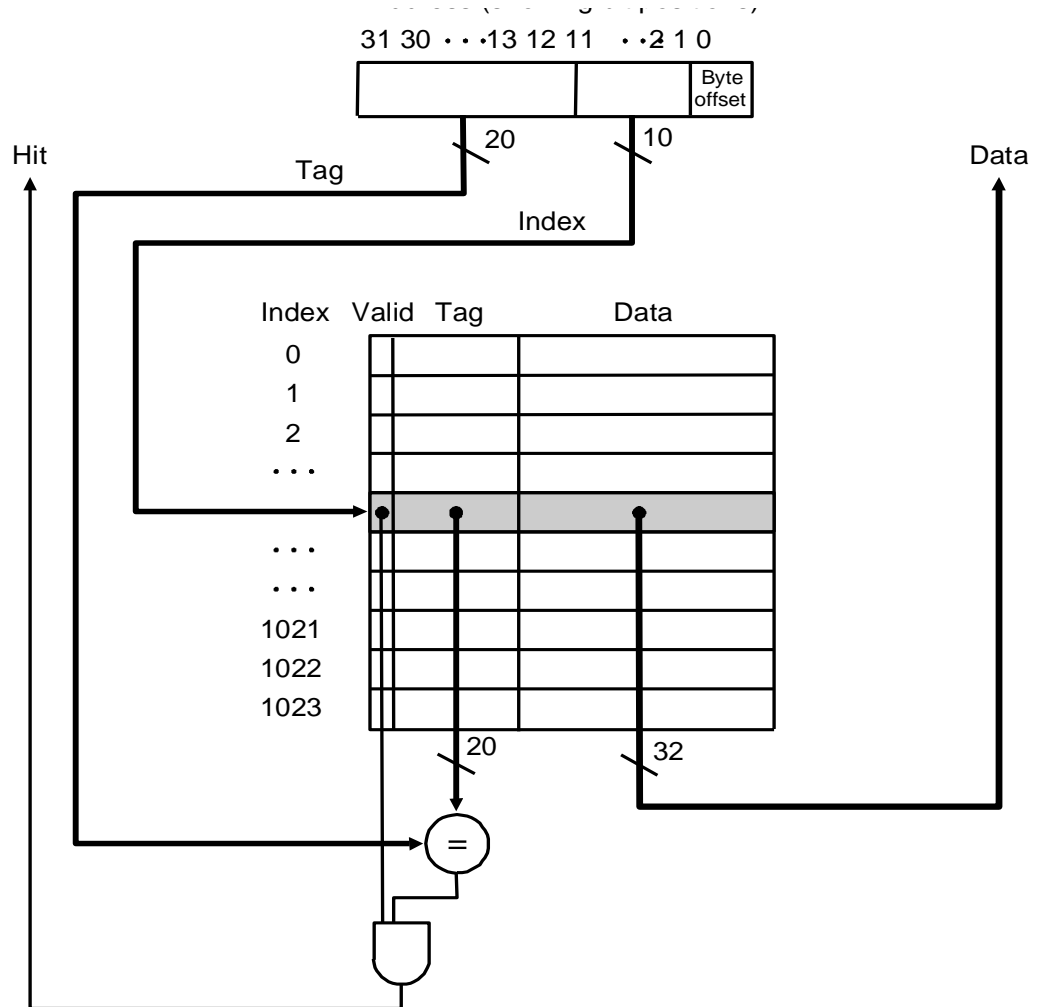
Index	V	Tag	Data
000	Y	10 <sub>two</sub>	Memory (10000 <sub>two</sub> )
001	N		
010	Y	10 <sub>two</sub>	Memory (10010 <sub>two</sub> )
011	Y	00 <sub>two</sub>	Memory (00011 <sub>two</sub> )
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory (10110 <sub>two</sub> )
111	N		

f. After handling a miss of address (10010<sub>two</sub>)

# Real Direct Mapped Cache

□ For MIPS:

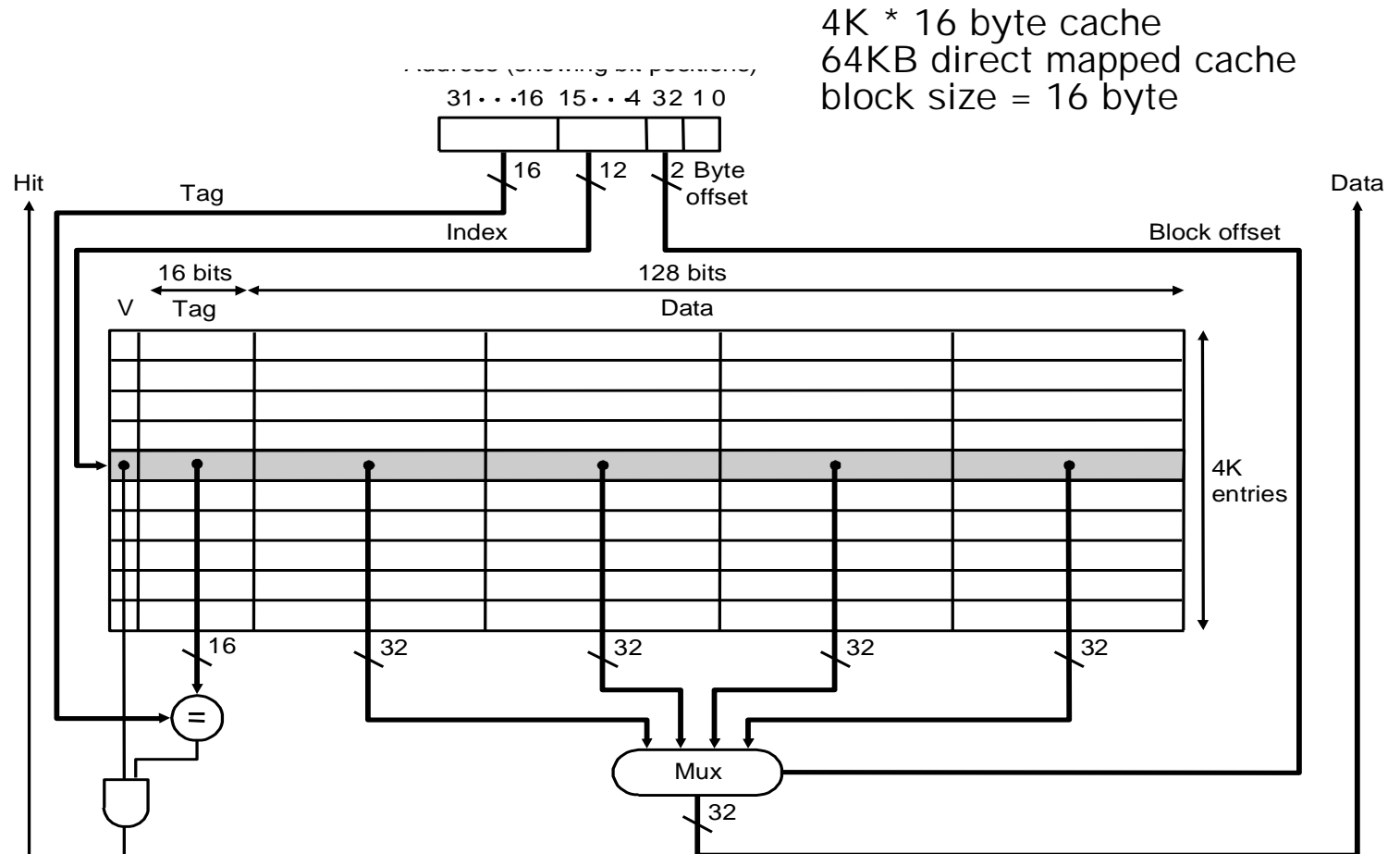
1K \* 4 byte cache  
4KB direct mapped cache  
block size = 4 byte



† What kind of locality are we taking advantage of?<sup>38</sup>

# Real Direct Mapped Cache

- Taking advantage of spatial locality



# Cache Memory: Performance



# Cache Performance (how to improve?)

- ❑ Average access time
  - = Cache hit time + miss rate \* miss penalty
- ❑ Three ways of improving performance
  - Decreasing hit time
  - Decreasing miss rate
  - Decreasing miss penalty
- ❑ What if
  - You increase block size (from 1 to  $\infty$ )
  - You increase total cache size
  - You use multi-level cache

# Quiz

---

- ❑ What if we reduce/increase the size of cache memory?
  - Which factor is improved?
    - Any side effect?
  - † Always consider the average memory access time!
  
- ❑ Let's consider the block size
  - Changing block size not affect cost, still large impact

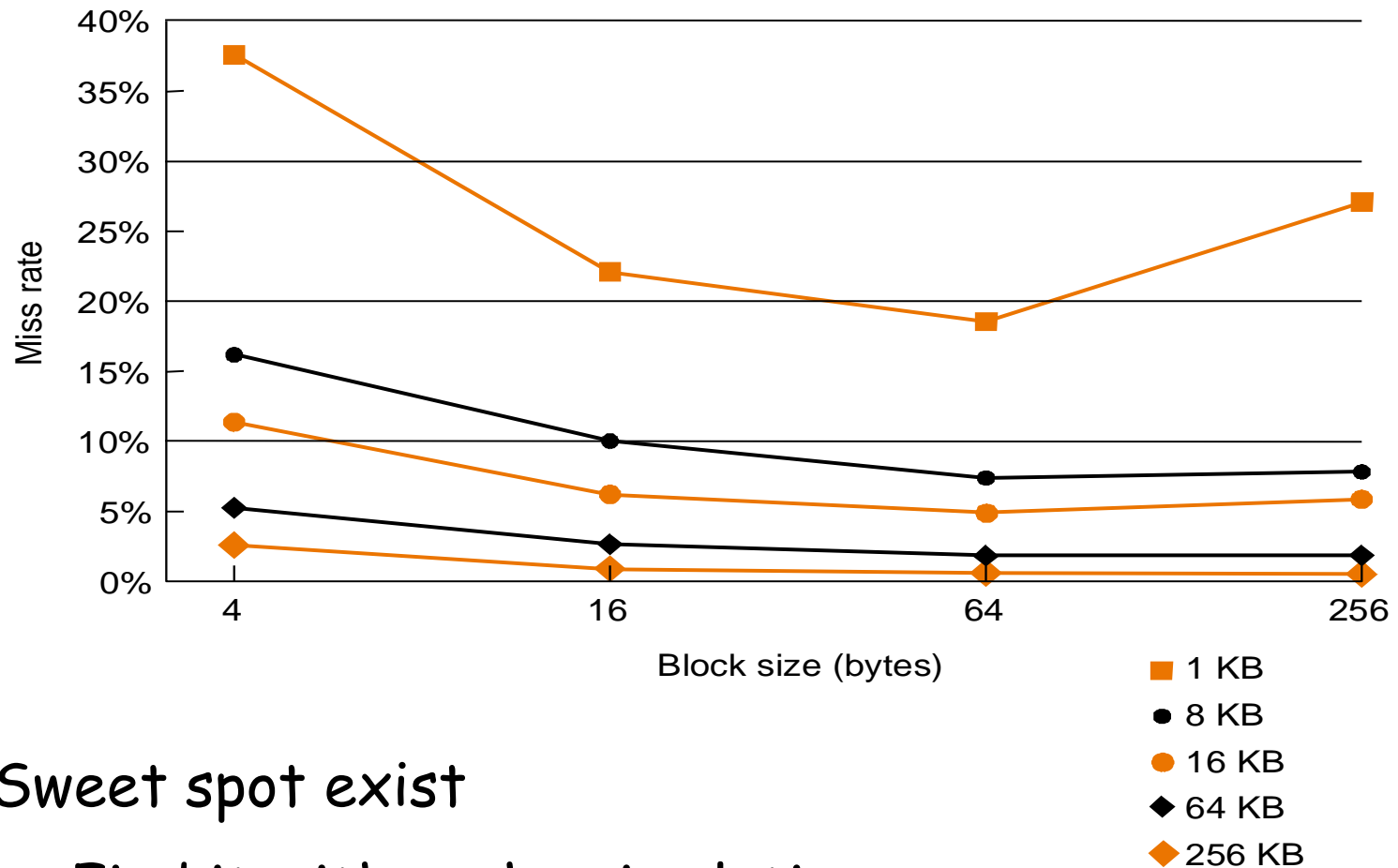
# Block Size Considerations

---

- ❑ Larger blocks should reduce miss rate
    - Due to spatial locality
  - ❑ But in a fixed-sized cache
    - Larger blocks  $\Rightarrow$  fewer of them
      - More competition  $\Rightarrow$  increased miss rate
    - Larger blocks  $\Rightarrow$  pollution
  - ❑ Larger block: larger miss penalty memory data transfer time/DRAM access time 가
    - Can override benefit of reduced miss rate
    - Early restart and critical-word-first can help
- † Always consider the average memory access time!

# Performance

## □ Increasing block size



† Sweet spot exist

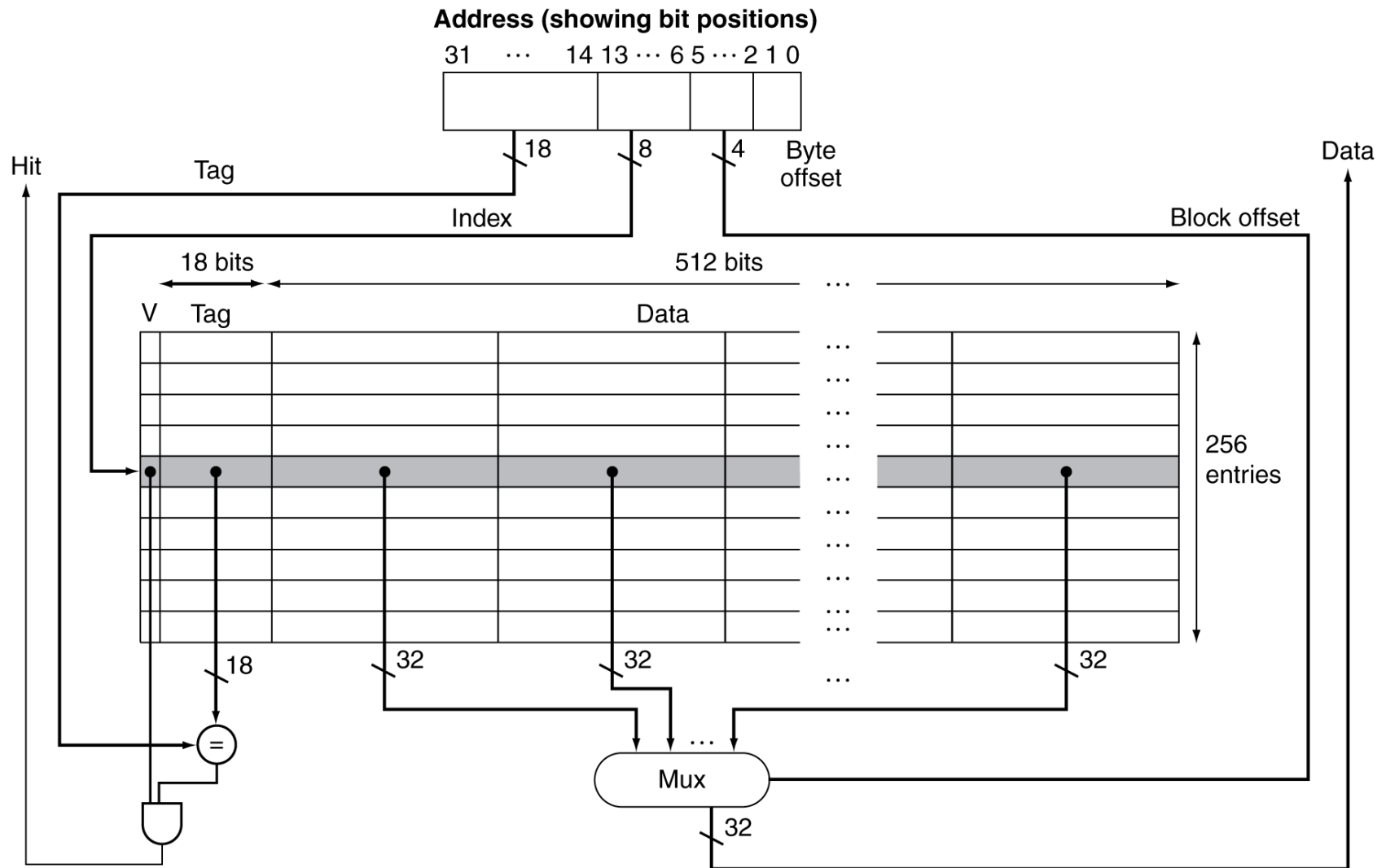
- Find it with cache simulation

# Example: Intrinsity FastMATH

---

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

# Example: Intrinsity FastMATH

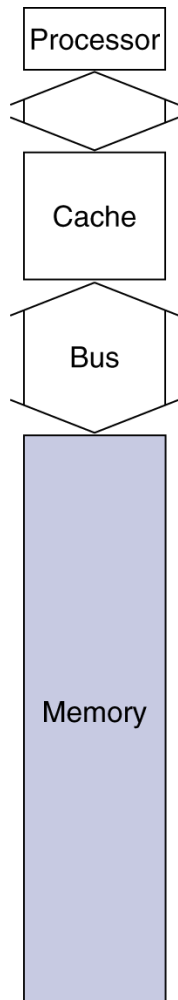


# Main Memory – Miss Penalty

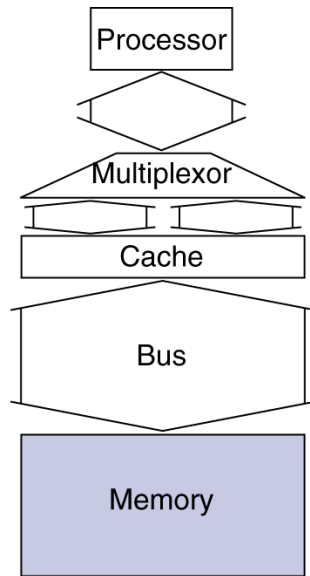
---

- ❑ 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 bytes / 65 cycles = 0.25 B/cycle

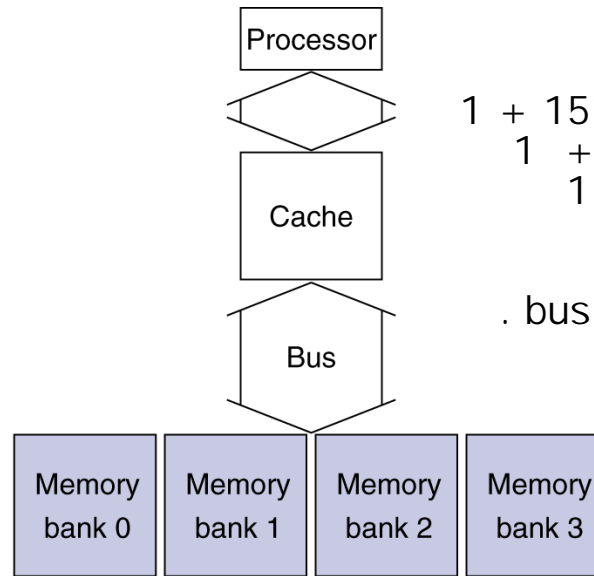
# Increasing Memory Bandwidth



a. One-word-wide memory organization



b. Wider memory organization



c. Interleaved memory organization

$$1 + 15 \cdot 1 + 1$$

$$1 + 15 \cdot 1 + 1$$

$$1 + 15 \cdot 1 + 1$$

$$1 + 15 \cdot 1 + 1 = 20$$

parallel  
. bus

- 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 =  $16 \text{ bytes} / 20 \text{ cycles} = 0.8 \text{ B/cycle}$



# Where are we?

---

- ❑ Physical memory and virtual memory
- ❑ Memory hierarchy
- ❑ Cache memory
  - Concepts and terminology
  - Direct-map cache: structure and operation
  - Performance: consider average access time
    - Hit time (small cache, direct map)
    - Miss rate (block size: sweet spot, cache simulation)
    - Miss penalty (block size, memory bandwidth)
- ✓ Different mappings (miss rate)

# Cache Memory: Set-Associative Mapping

# Direct Map Cache

---

## ❑ Simplest and fastest mapping

- No choice in placement, identification, replacement
- Most widely used
  - Shorter clock cycle

## ❑ Problem of miss rate

- **Address conflict:** 2-way conflict, 4-way conflict, ...

conflict가  
memory  
가

- Miss rate may go up

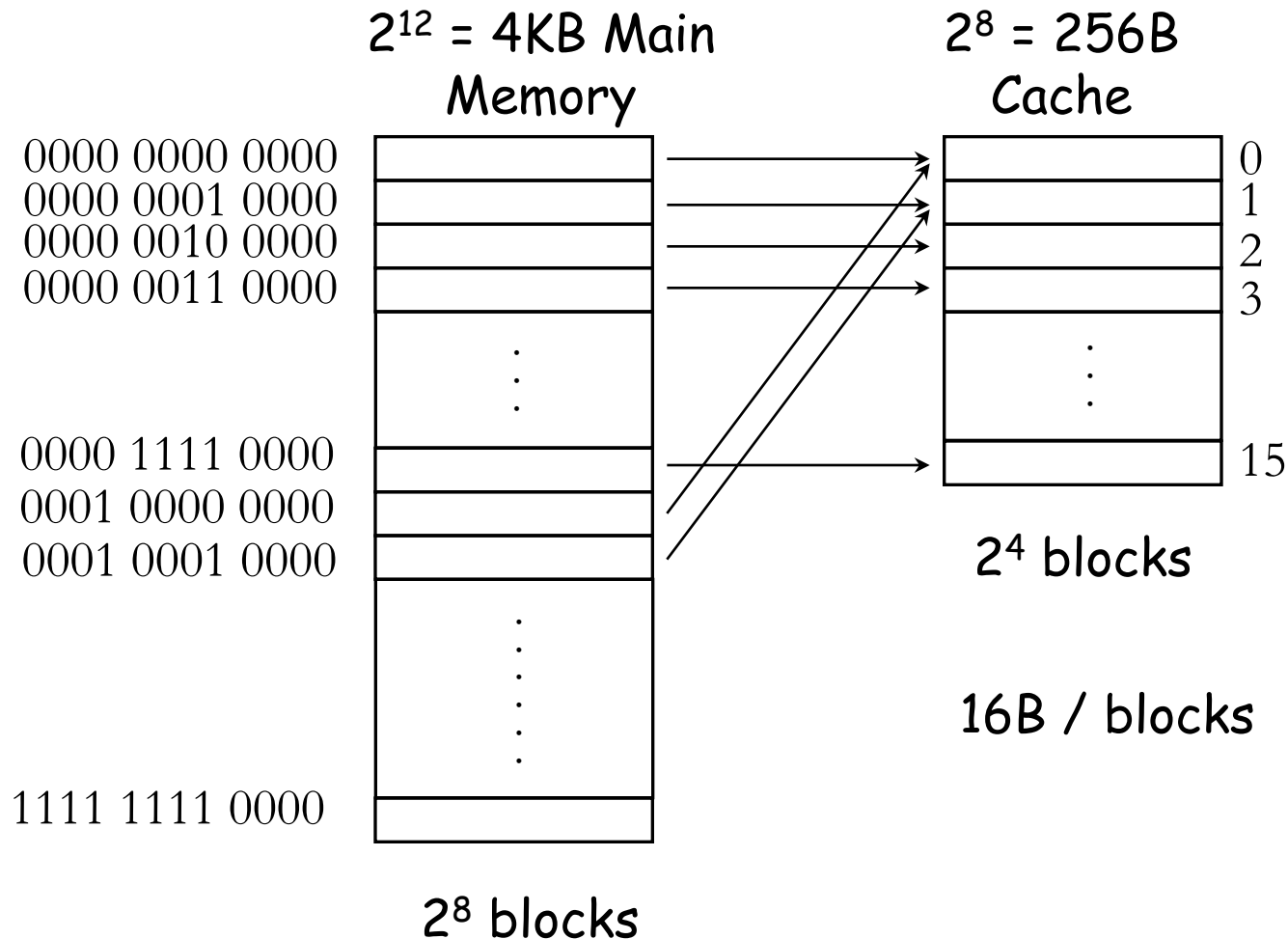
working set  
(cache block mapping .)

cache  
performance  
!

- Less a problem when cache is large enough

- Set-associative mapping can be a solution

# Direct Map Cache: Placement (반복)



set

# Two-Way Set-Associative Cache

two - way conflict 가 !

$2^{12} = 4\text{KB}$  Main

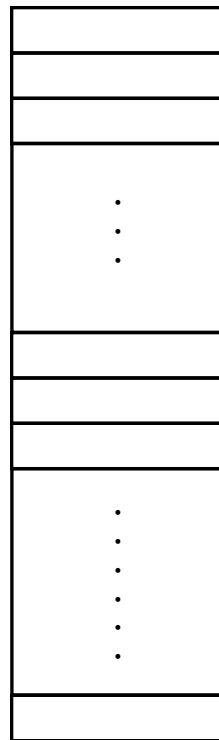
$2^8 = 256\text{B}$  Cache

Memory

0000 0000 0000  
0000 0001 0000  
0000 0010 0000

0000 0111 0000  
0000 1000 0000  
0000 1001 0000

1111 1111 0000

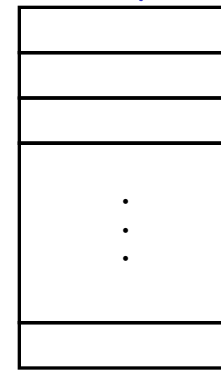
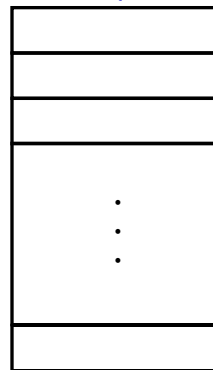


Way 0

Way 1

Set 0  
Set 1  
Set 2

Set 7



$2^3$  blocks  $2^3$  blocks

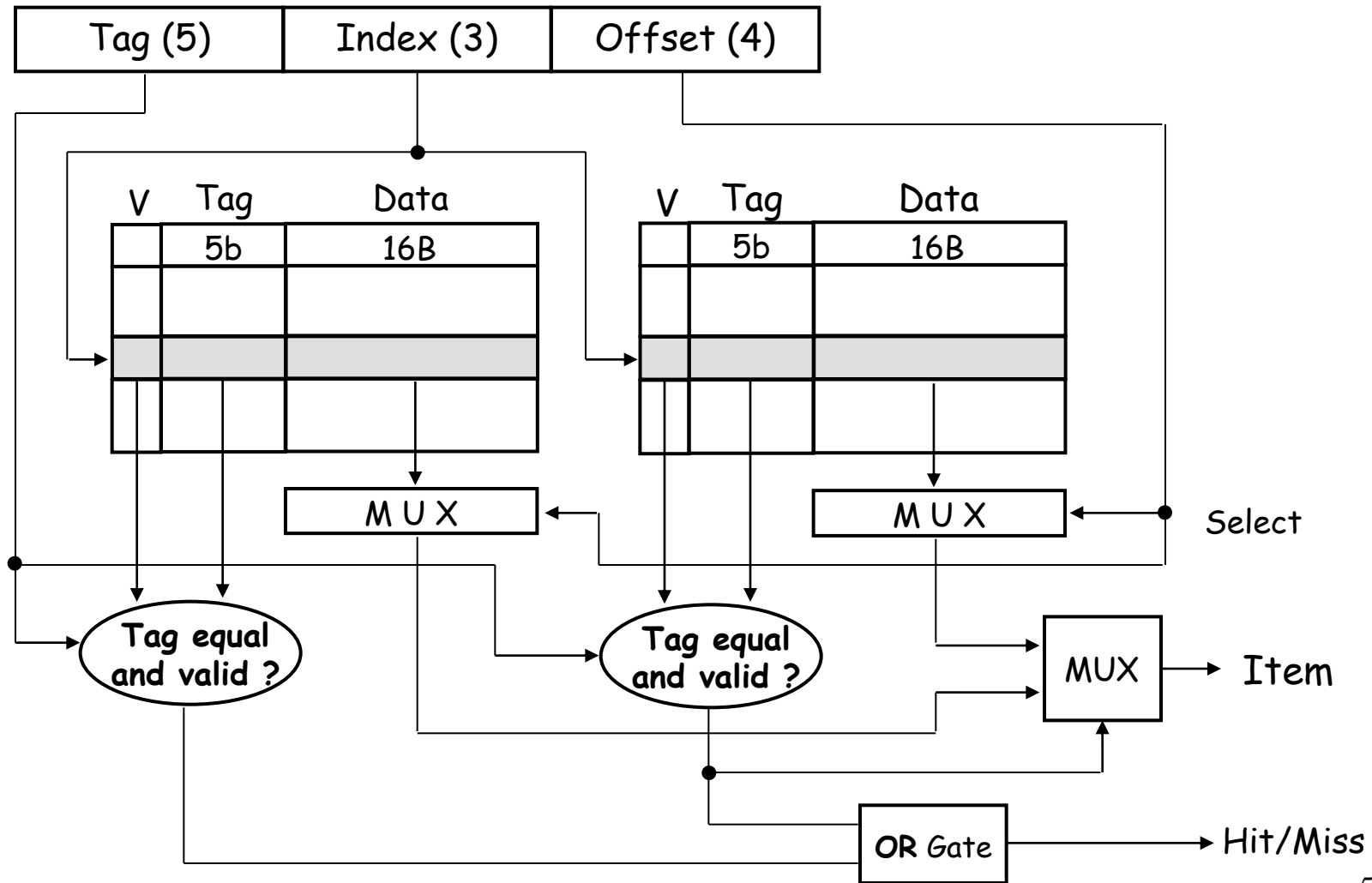
direct mapped      two-way  
Tag index offset -> Tag index offset  
4    4    4      5    3    4  
16:1      32:2  
16 가      32 가    가  
1          2      가

8 sets in cache,  
2 blocks per set

$2^8$  blocks

16B / blocks

# Two-Way SA Cache: Identification



hit time

가  
cct

hit time

가  
가

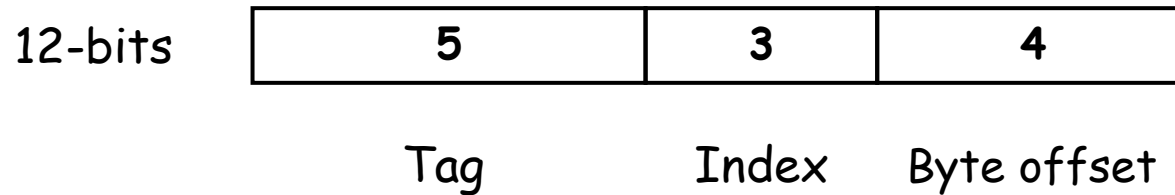
miss rate

(two-way conflict

)

# Two-Way SA Cache

- How do we use an address?



- 32-to-2 compaction; can you see more freedom?
  - Tag size increases

V	Tag	Data
1b	5b	16B
⋮	⋮	⋮
⋮	⋮	⋮

V	Tag	Data	
1b	5b	16B	0
⋮	⋮	⋮	
⋮	⋮	⋮	7

- What do we gain? What do we lose?

# Four Issues in Memory Management

---

❑ Q1: placement (mapping)

❑ Q2: identification hit/miss

❑ Q3: write strategy

- Write-through: simple and consistent, write buffer
- Write-back: may reduce memory traffic
  - Dirty bit

❑ Q4: replacement policy direct 가 , set associative cache 가 way

- Least recently used (LRU) and reference bit temporary locality - reference bit(1 bit) 가 1 0 (implementation algorithm)
  - LRU too costly (For even four-way, LRU is approximated)
  - Can use random (use free-running counter)
    - For large cache, difference from LRU become small

† Can you see why clock cycle time increases?



# Write-Through

---

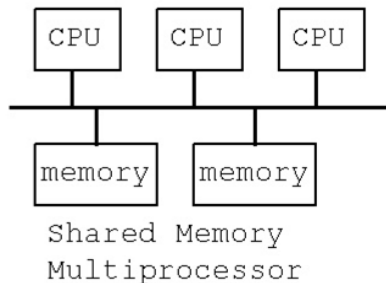
- ❑ On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- ❑ Write through: also update memory
- ❑ But makes writes take longer (high miss penalty)
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI =  $1 + 0.1 \times 100 = 11$
- ❑ Solution: **write buffer** cpu가                  memory
  - 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
    - Dirty bit for each block
- ❑ When a dirty block is replaced
  - Write it back to memory
  - Can use write buffer to reduce miss penalty

shared memory multi processor



write back  
why? bus traffic

# Write through vs. Write-Back

## ❑ Advantages of write-back

- Individual words can be written at cache speed
- Multiple writes within a block result in one write to lower memory
- Since entire block is written, can effectively use high-bandwidth transfer

## ❑ Advantages of write through

- Misses are simpler and cheaper (no write to lower memory)
- Easier to implement than write-back

† Think about shared-bus multiprocessor and write back

# Write Allocation (can skip)

---

- ❑ 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

# 4, 8, 16-Way Set-Associative Cache

□ Can you imagine

way 가 hit time 가 / hit rate 가

- 4-way, 8-way, ...

Tag	Index	B. Offset
-----	-------	-----------

DM	4	4	4	16:1	mapping
2-way	5	3	4	32:2	
4-way	6	2	4	64:4	
8-way	7	1	4	128:8	
16-way	8	0	4	256:16	
					fully associative mapping

(16-way mean complete freedom in mapping)

# 4, 8, 16-Way Set-Associative Cache

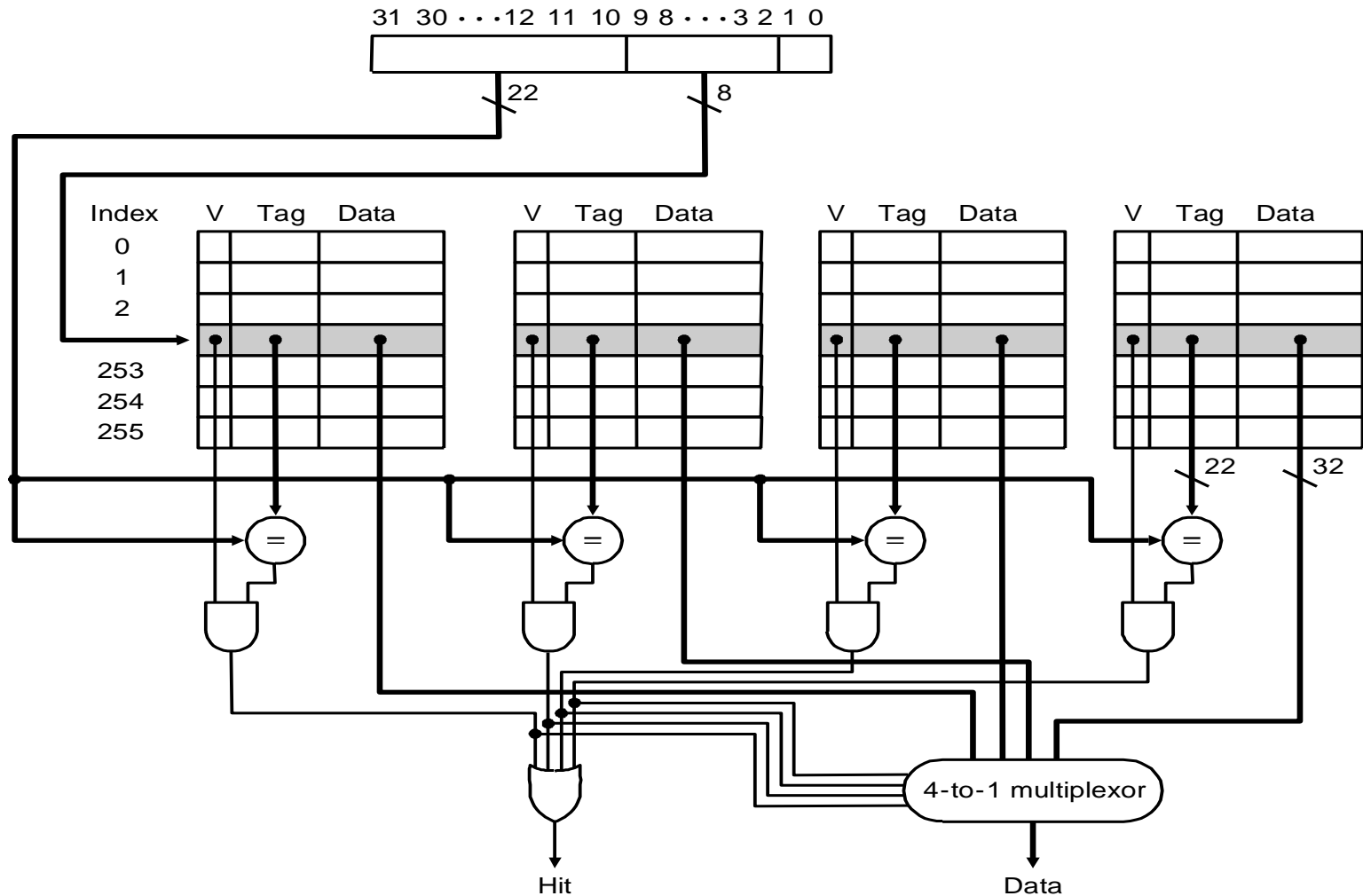
## ❑ Can you imagine

- 4-way
- 8-way

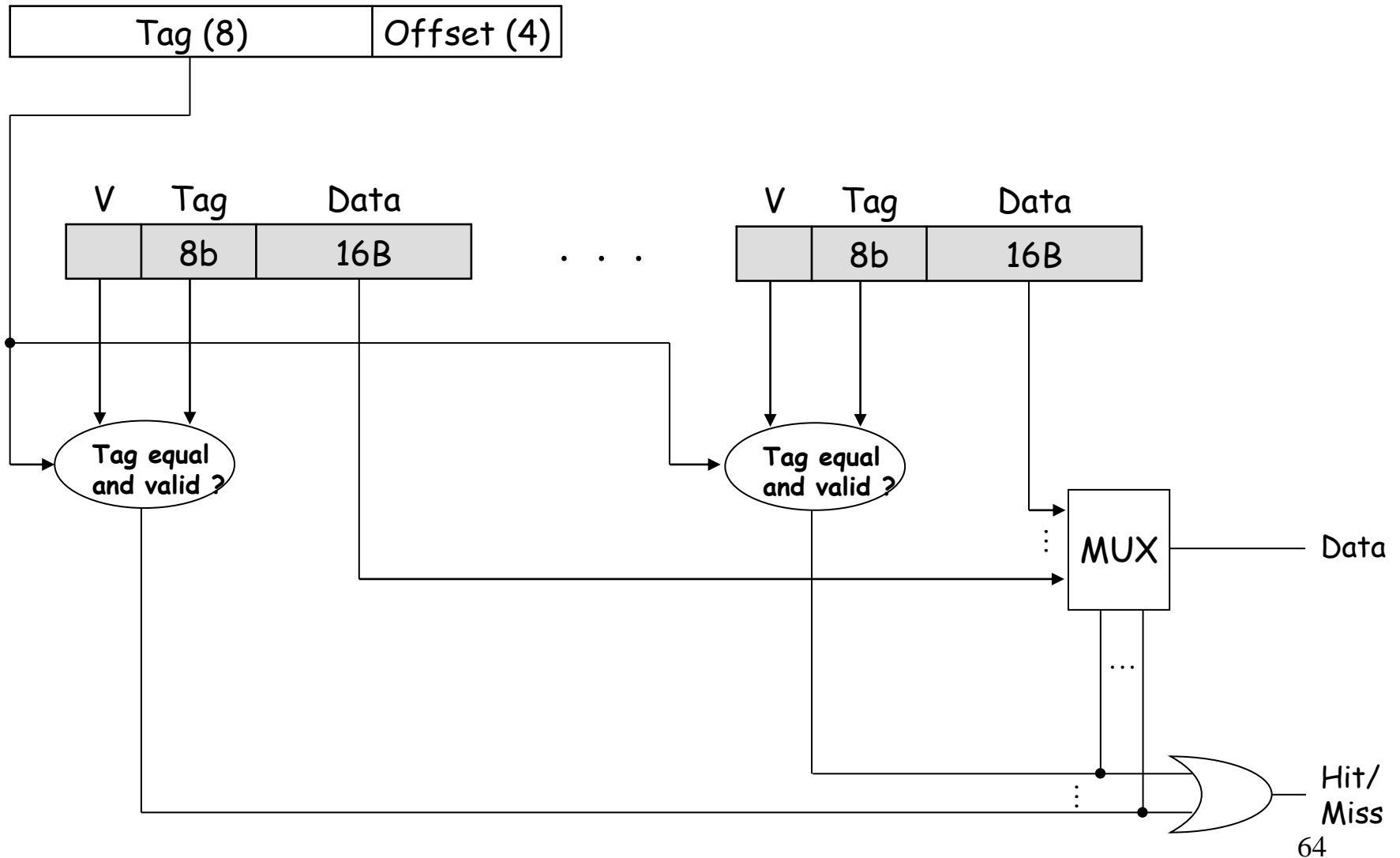
## ❑ 16-way set-associative mapping

- Fully-associative mapping, in this example
  - Index field disappear
  - Parallel search (by hardware)
  - Content addressable memory (CAM)
- More hardware, longer clock cycle time, good hit rate
  - Can be used in small, specialized cache (e.g., TLB)

# Real 4-Way SA Cache

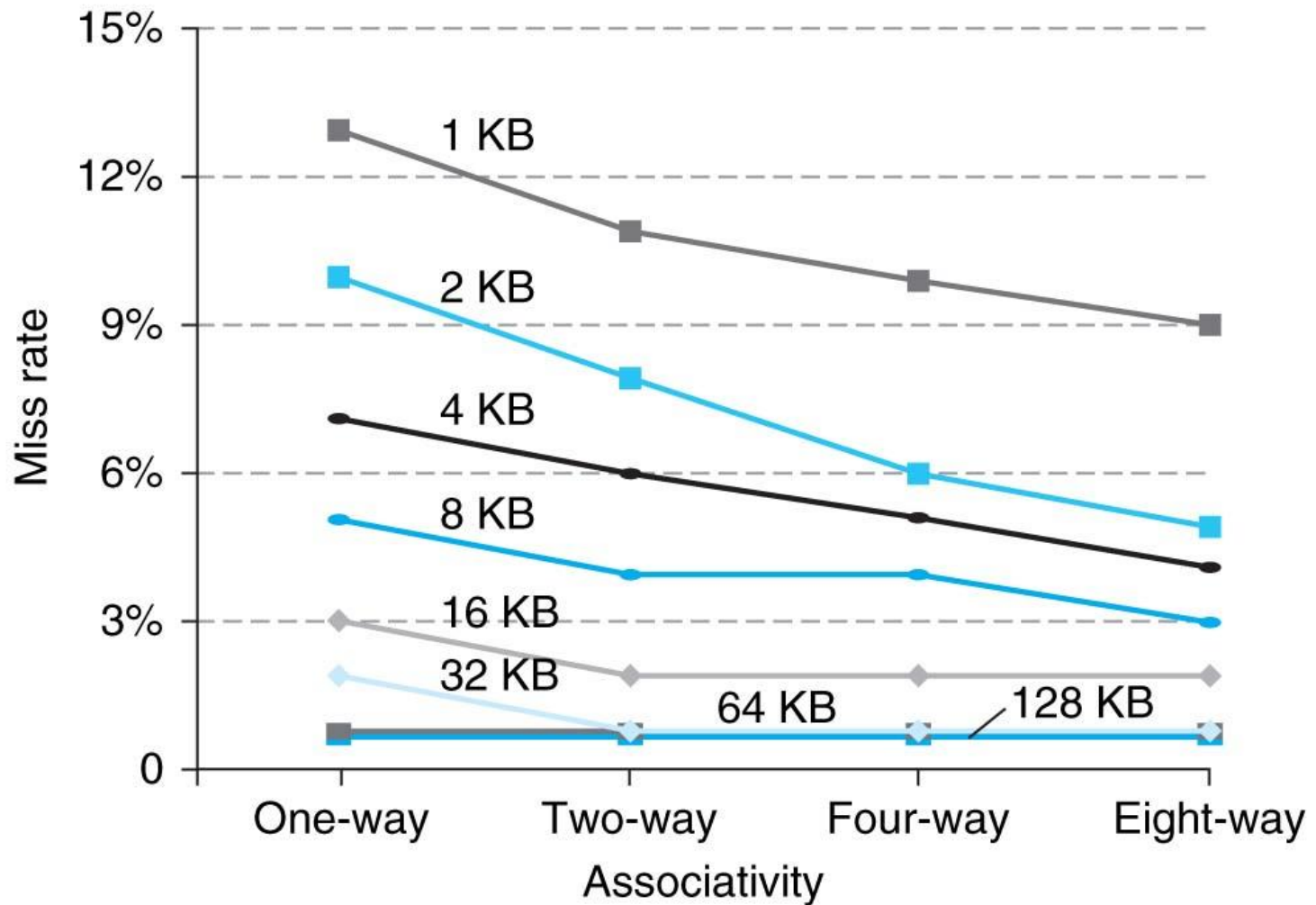


# Fully-Associative Mapping





# Performance



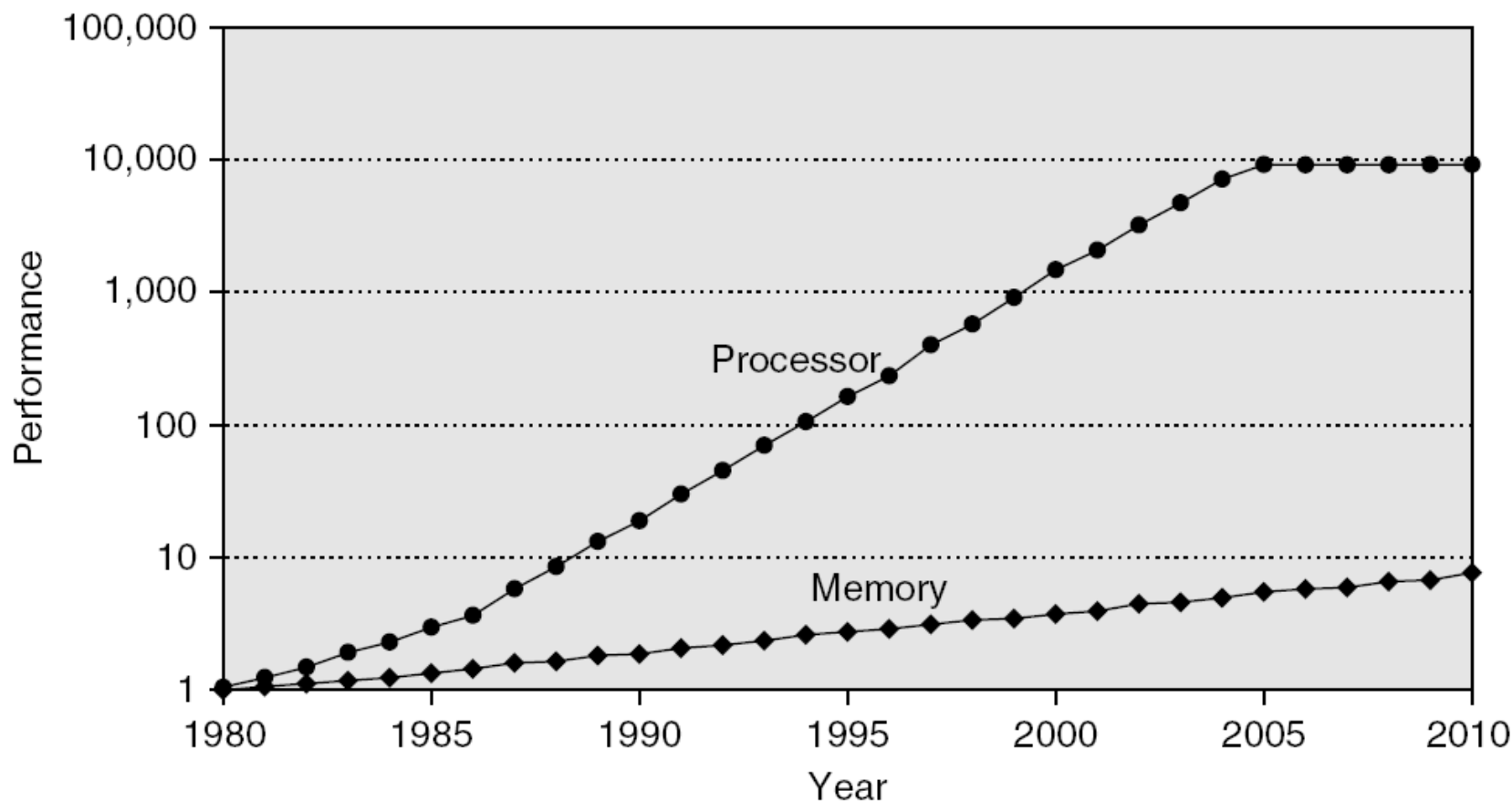
# Performance

---

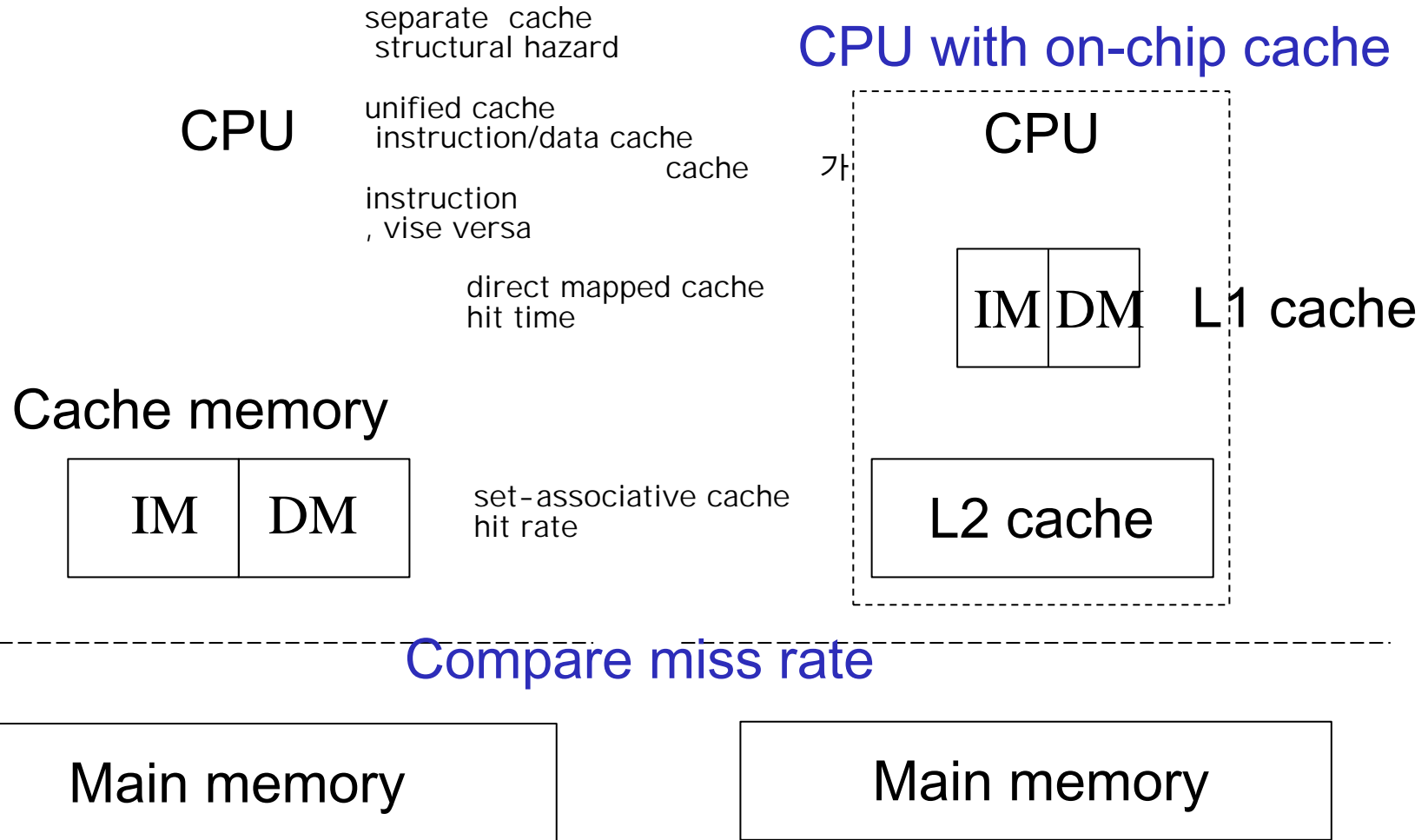
- ❑ As associativity increases, miss rate drops
  - Largest gain: from direct map to 2-way SA
- ❑ As cache size increases
  - Miss rate drops
  - Impact of associativity becomes smaller
- † May use Large DM cache for higher clock speed

# Multi-Level Cache

# CPU-Memory Performance Gap



# Multilevel Caches



# Multilevel Caches

---

- ❑ Small primary cache (level 1 or L1 cache)
    - Cache hit becomes faster
      - Small miss penalty if data in 2nd level cache
  - ❑ Level-2 (L2) cache services misses from primary cache
    - Larger, slower, but still faster than main memory
  - ❑ Main memory services L2 cache misses
- † L1 and L2 caches inside processor chip
- † Some high-end systems use L3 cache also

# Multi-Level Caches

---

- ❑ Performance: why use them?
  - 1-level versus 2-level cache (with same last level size)
    - If cache is reasonably big, show similar miss rates
- ❑ Using multilevel caches:
  - Try and optimize hit time on L1 cache
    - Size of L1 cache has been growing slowly, if at all
  - Try and optimize miss rate on L2 cache
    - Size of L2 cache has been growing steadily

# Performance Summary

---

- ❑ As CPU performance increase
  - 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



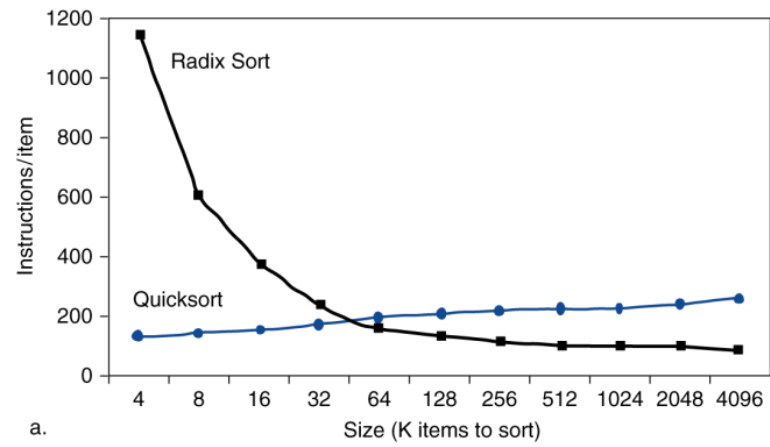
# Understanding Program Performance (참고자료)

# Interactions with Advanced CPUs

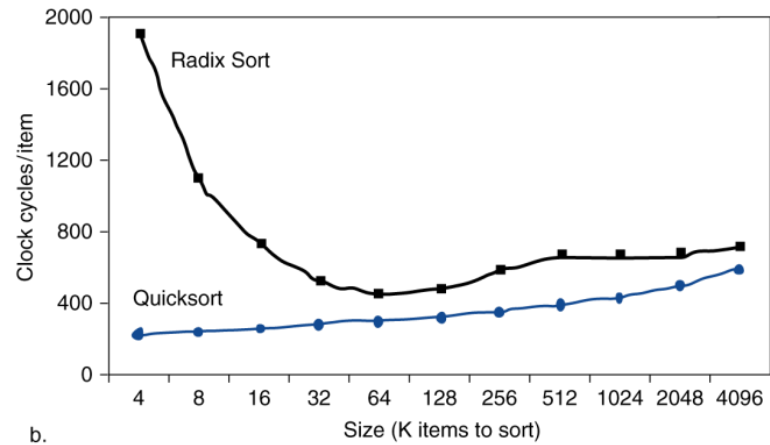
- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

# Interactions with Software

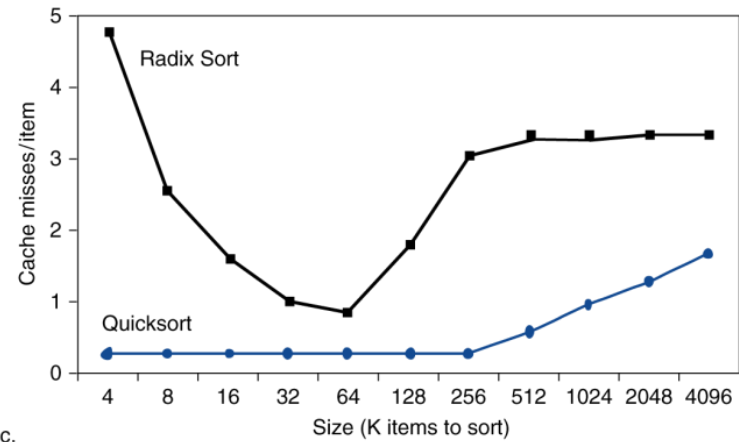
- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access
- Radix sort vs. Quick sort
  - Radix sort has algorithmic advantage
    - But slower due to cache miss – next slide
  - New versions of Radix sort invented
- Using memory hierarchy well critical to high performance



a.



b.



c.

# Software Optimization via Blocking

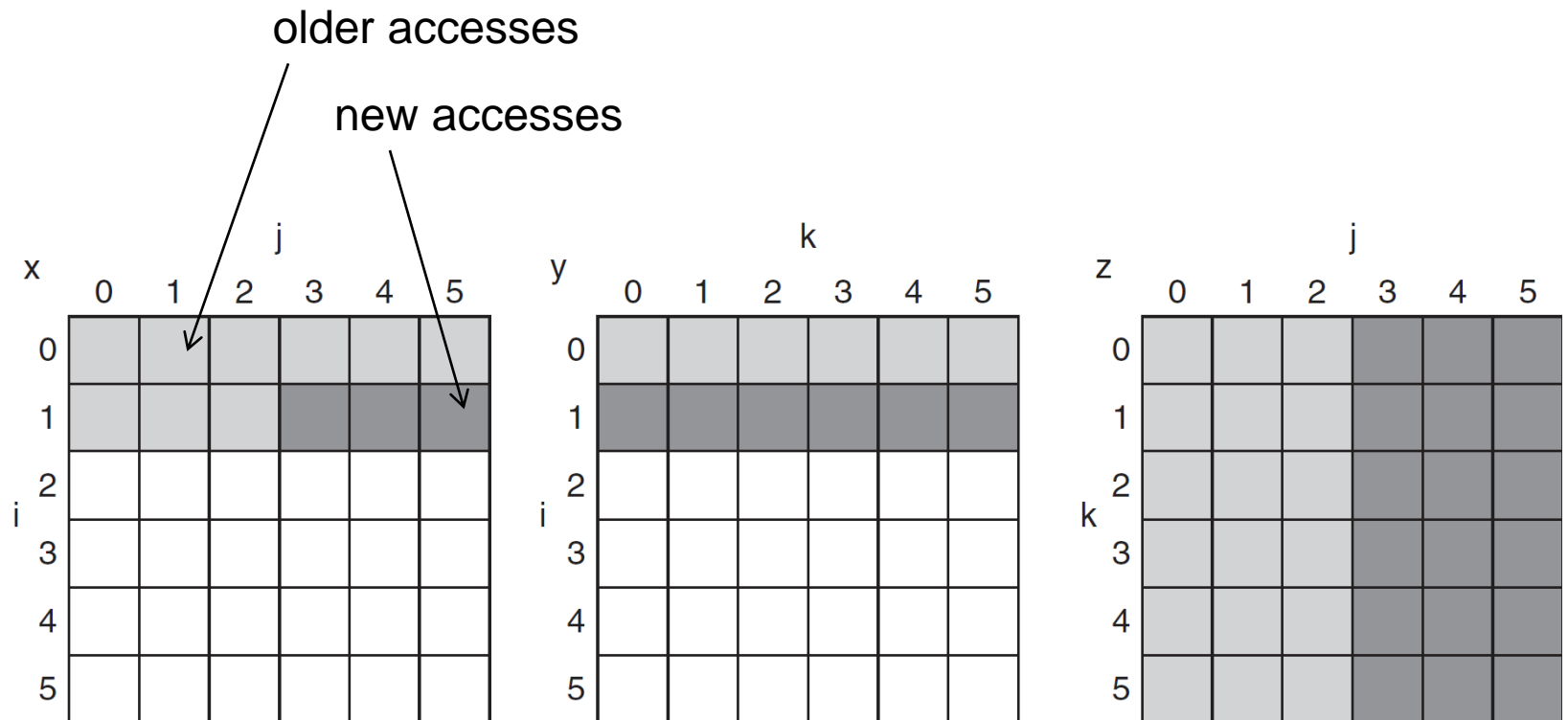
- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

cache miss      - memory

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

# DGEMM Access Pattern

- C, A, and B arrays

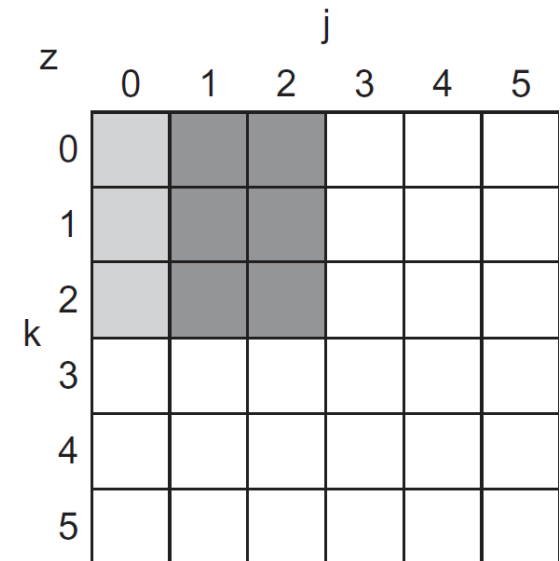
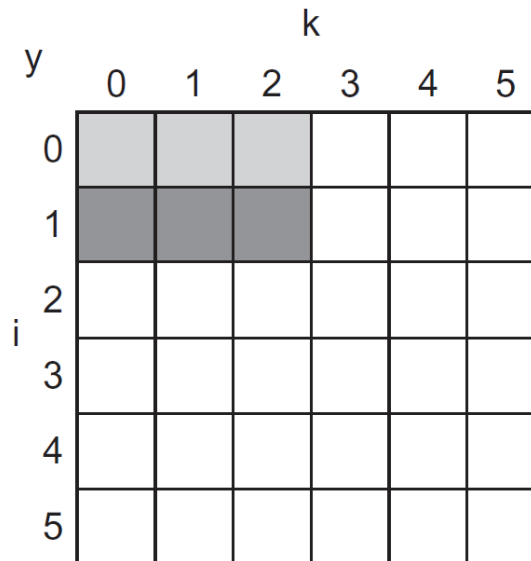
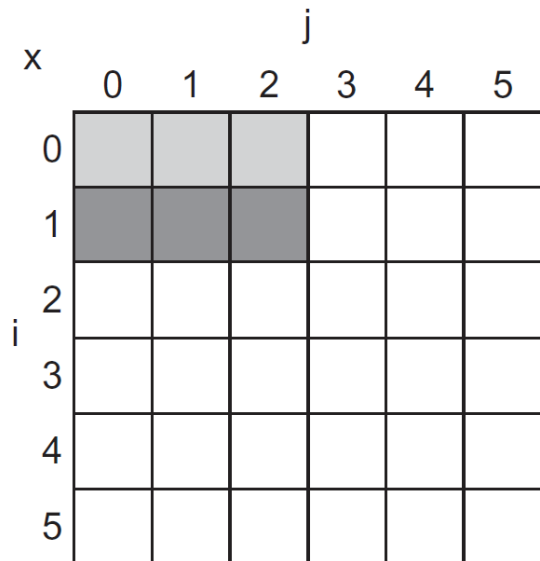


# Cache Blocked DGEMM

performance programming

```
1 #define BLOCKSIZE 32
2 void do_block (int n, int si, int sj, int sk, double *A, double
3 *B, double *C)
4 {
5   for (int i = si; i < si+BLOCKSIZE; ++i)
6     for (int j = sj; j < sj+BLOCKSIZE; ++j)
7       {
8         double cij = C[i+j*n]; /* cij = C[i][j] */
9         for( int k = sk; k < sk+BLOCKSIZE; k++ )
10          cij += A[i+k*n] * B[k+j*n]; /* cij+=A[i][k]*B[k][j] */
11        C[i+j*n] = cij; /* C[i][j] = cij */
12      }
13 }
14 void dgemm (int n, double* A, double* B, double* C)
15 {
16   for ( int sj = 0; sj < n; sj += BLOCKSIZE )
17     for ( int si = 0; si < n; si += BLOCKSIZE )
18       for ( int sk = 0; sk < n; sk += BLOCKSIZE )
19         do_block(n, si, sj, sk, A, B, C);
20 }
```

# Blocked DGEMM Access Pattern





# Blocked DGEMM Access Pattern

