Big Picture

- Issue 1: Fundamental concepts and principles
  - What is computer, CSE, computer architecture?
- Issue 2: ISA (HW-SW interface) design
  - Ch. 1: computer performance
  - Ch. 2: language of computer; ISA
  - Ch. 3: data representation and ALU
- Issue 3: implementation of ISA (internal design)
  - Ch. 4: processor (data path, control, pipelining)
  - Ch. 5: memory system (cache memory)
- Short introduction to parallel processors

MIPS Pipeline

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

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

Big Picture

- Part 3: implementation of ISA
  - 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
    1) Memory systems: physical and virtual
    2) Memory hierarchy
    3) Cache memory: structure, operation and performance
    4) Cache and virtual memory

Chapter 5

Large and Fast: Exploiting Memory Hierarchy

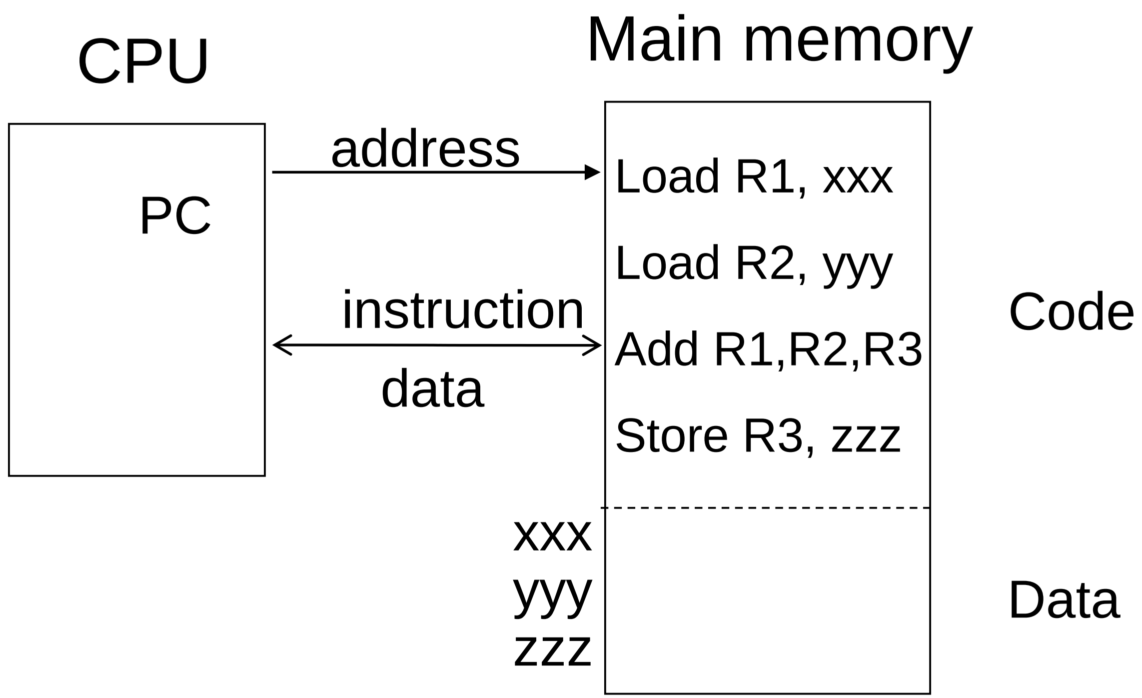
Part 1

Some of authors' slides are modified

Physical and Virtual Memory Systems

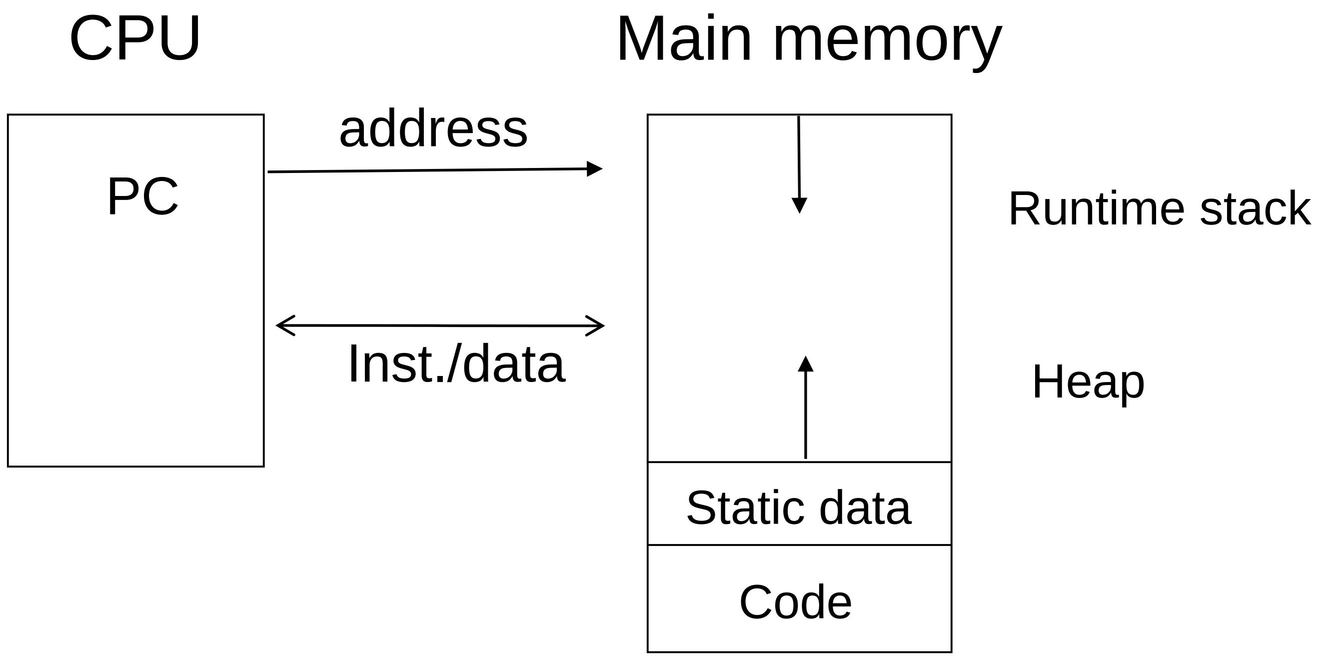
Physical Memory Model

- CPU address is the address of main memory



- Small embedded systems: assembly programming
  - Programmers allocate memory

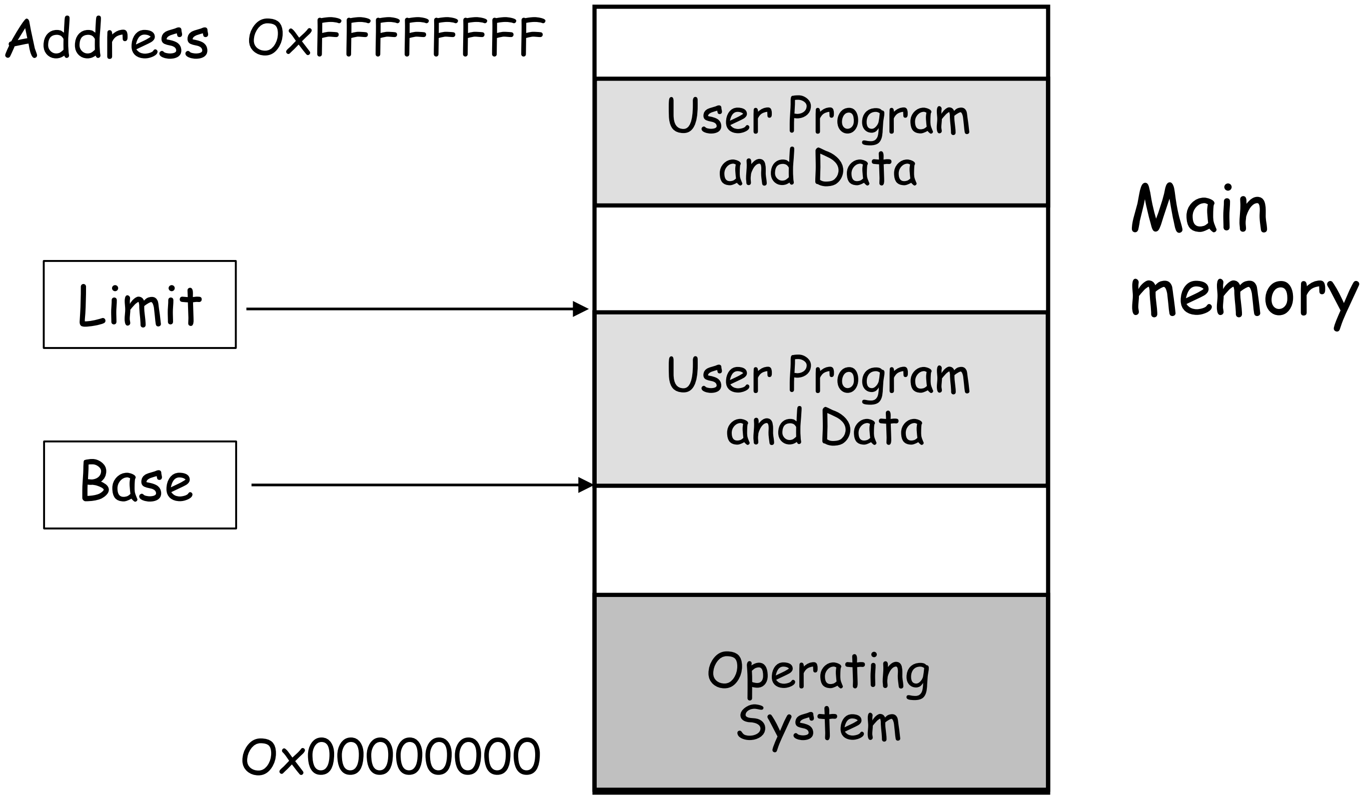
Physical Memory Model



- Small embedded systems: C programming
  - Compilers allocate memory

Physical Memory: General-Purpose Computers

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

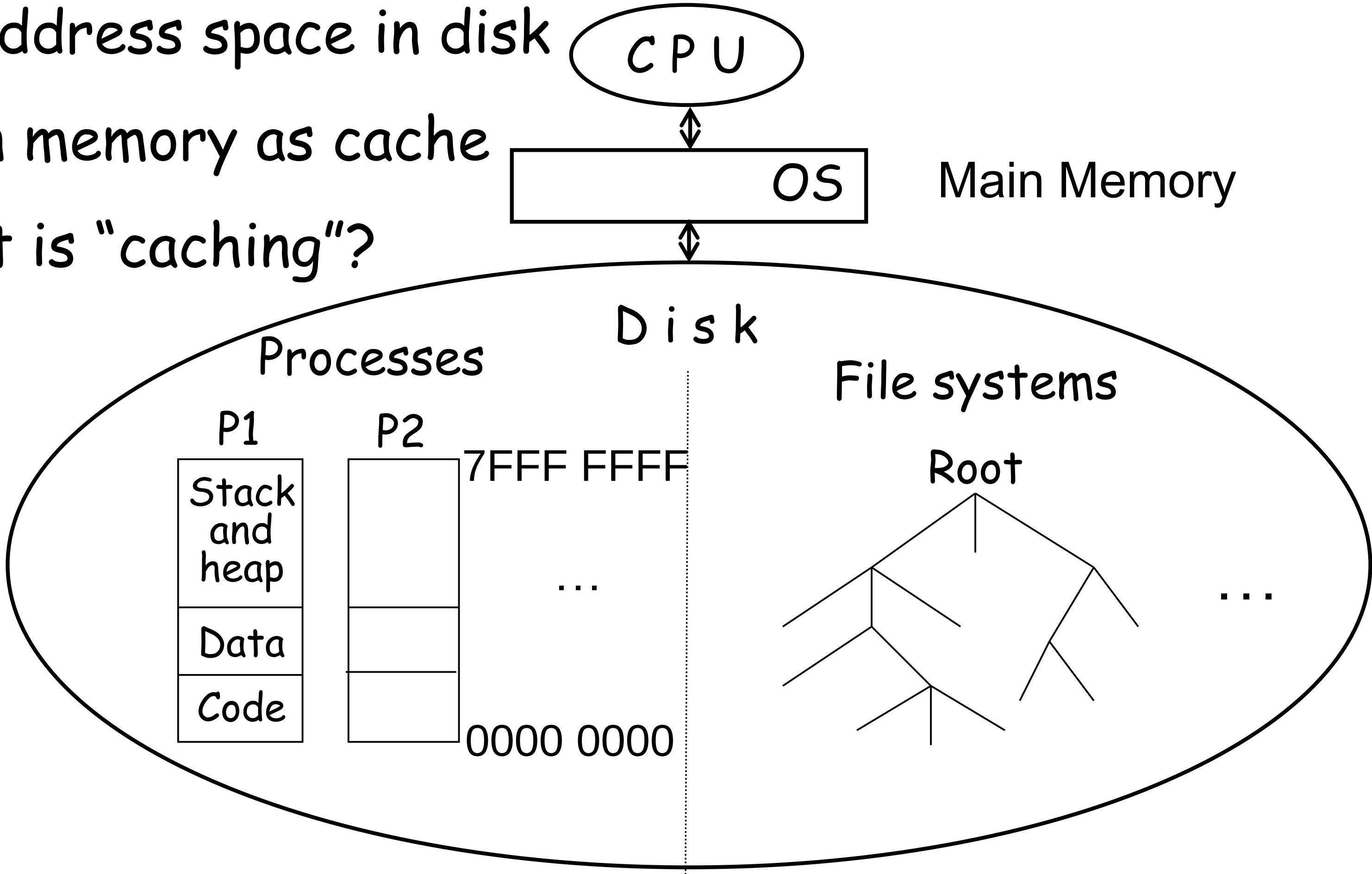


Motivations for VM (OS Topic; 참고)

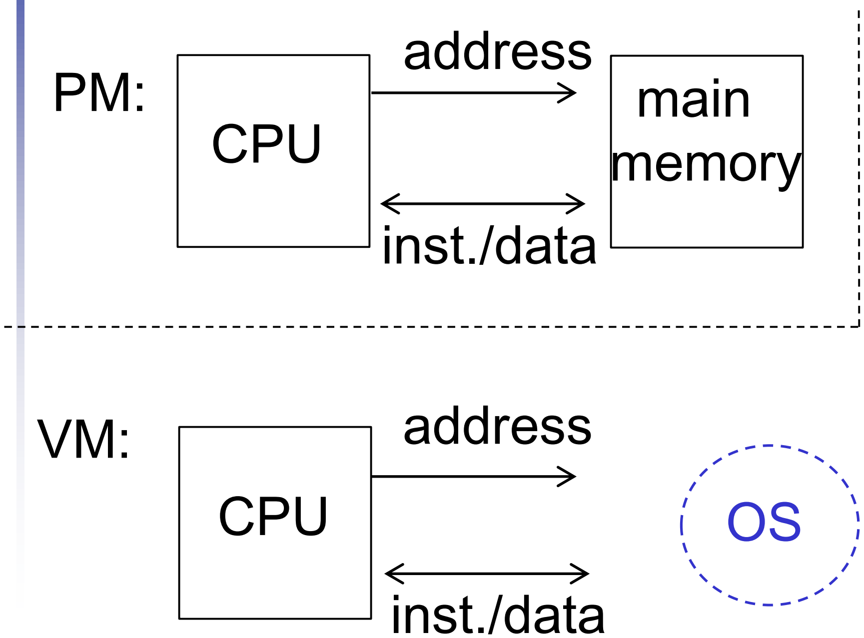
- What if single program exceed the size of main memory
  - Formerly, programmers divide program into pieces
    - Group them into overlays (modules)
  - Serious burden to programmers
- Decouple main memory and process address space
  - 프로그램 컴파일 및 실행이 main memory 와 무관하도록
- Multiple user programs share main memory
  - Protected from other programs
- Simplify loading of the program for execution

Virtual Memory: General-Purpose Computers

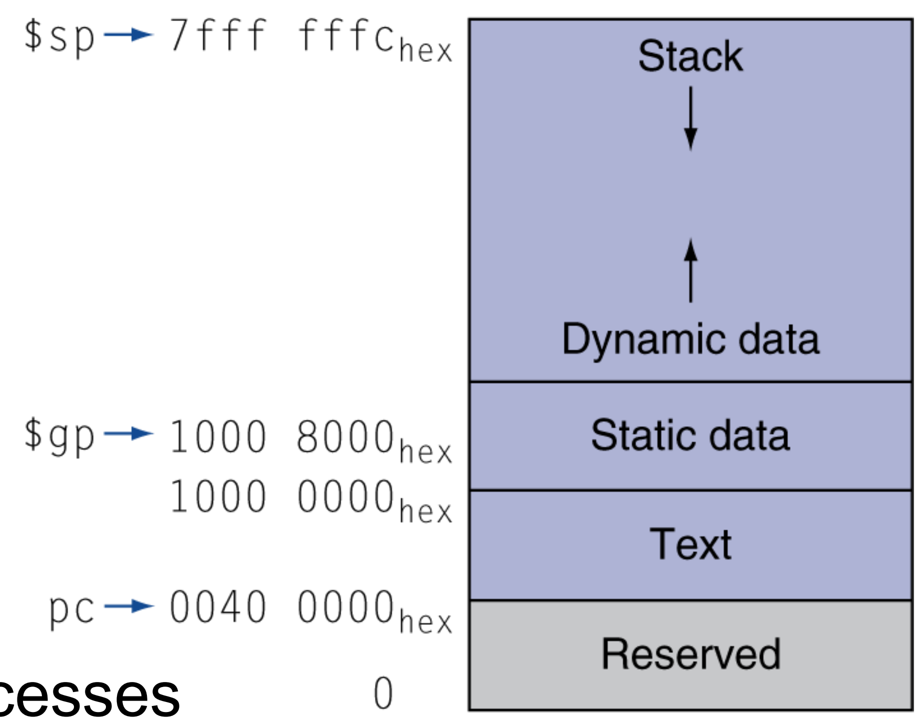
- Decouple main memory and user process address space
  - Virtual address space in disk
  - Use main memory as cache
    - What is "caching"?



Virtual Memory: General-Purpose Computers



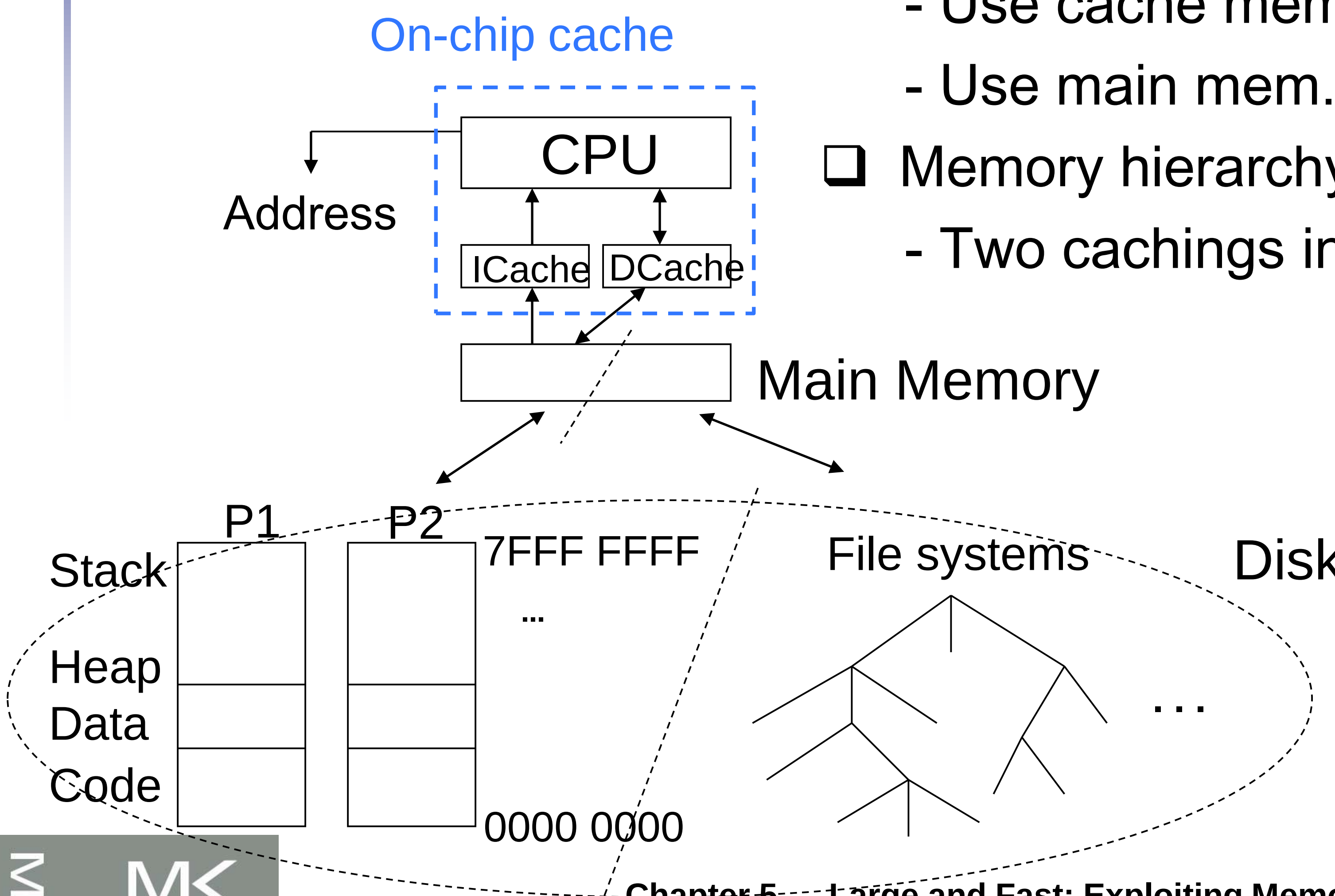
Virtual address space



- Computer system design
  - Compiler: compile and link
  - OS: create and manage processes
  - CPU: designed ISA

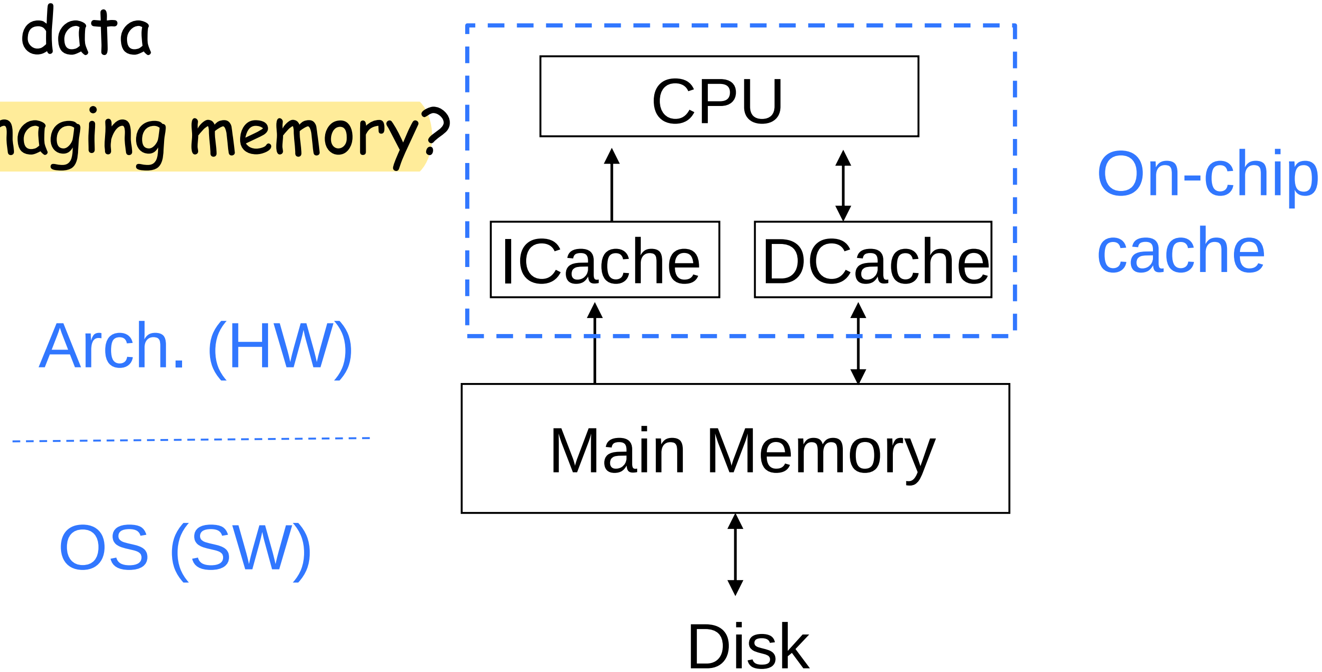
General-Purpose Computers

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



Split vs. Unified Memory

- Why split cache?
  - To avoid structural hazards
- Why unified main memory?
  - Better; if lots of data are accessed, allocate more area for data
- Who is managing memory?



Memory Hierarchy

Memory Technology

- Ideal memory: capacity, cost, speed

Speed	Processor	Size	Cost ($/bit)		Current technology	
Fastest	Memory	Smallest	Highest	Fast	SRAM	(Cache memory)
	Memory				DRAM	(Main memory)
Slowest	Memory	Biggest	Lowest	Slow	Magnetic disk	

Memory Technology (참고)

- Current technology: SRAM, DRAM, disk (flash mem.)
  - Survival of the fittest

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

- Ideal memory: capacity, speed, cost
- How to build (illusion of) "ideal memory"?

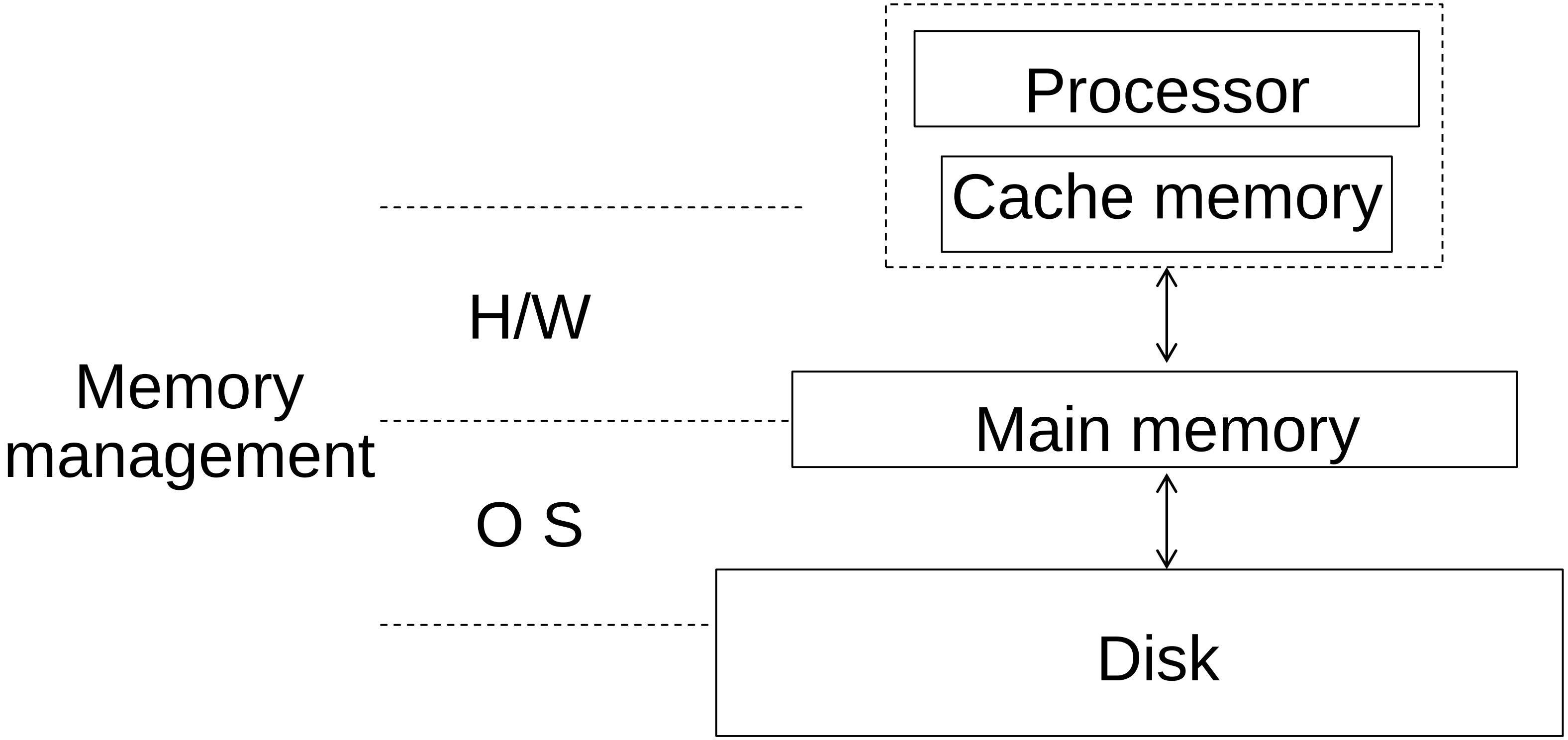
Memory management		Access time (cycles)	Capacity (B)
	Processor	0.5	< 1 K's
H/W	Cache memory	1's, 10's	10K's, M's
	Main memory	100's	G's
O S	Magnetic disk	>10,000,000's	T's

Principle of Locality

- Programs access a small proportion of their address space at any time (e.g., notion of working set)
  - What make memory hierarchy a good idea
- If an item (instruction or data) is referenced
  - Temporal locality: likely to reference it again soon
    - e.g., instructions in a loop, induction variables
  - Spatial locality: likely to reference nearby items soon
    - e.g., sequential instruction access, array data
- Given locality, how do we manage memory hierarchy?

Memory Hierarchy (반복)

- Ideal memory: capacity, speed, cost
- How to build (illusion of) “ideal memory”?



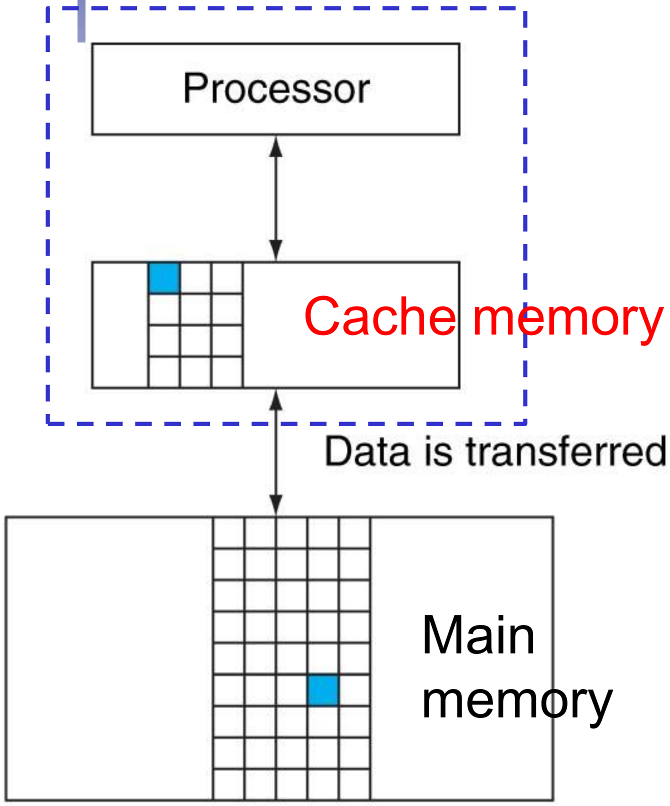
Memory Management

- Moving items between two adjacent levels
  - Two different layers: cache and virtual memory
- When do we move?
  - $\uparrow$: on-demand (vs. prediction)
  - $\downarrow$: to make space for new entry
- 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 (요약)

- 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 inside CPU

Memory Hierarchy: Cache Memory Level

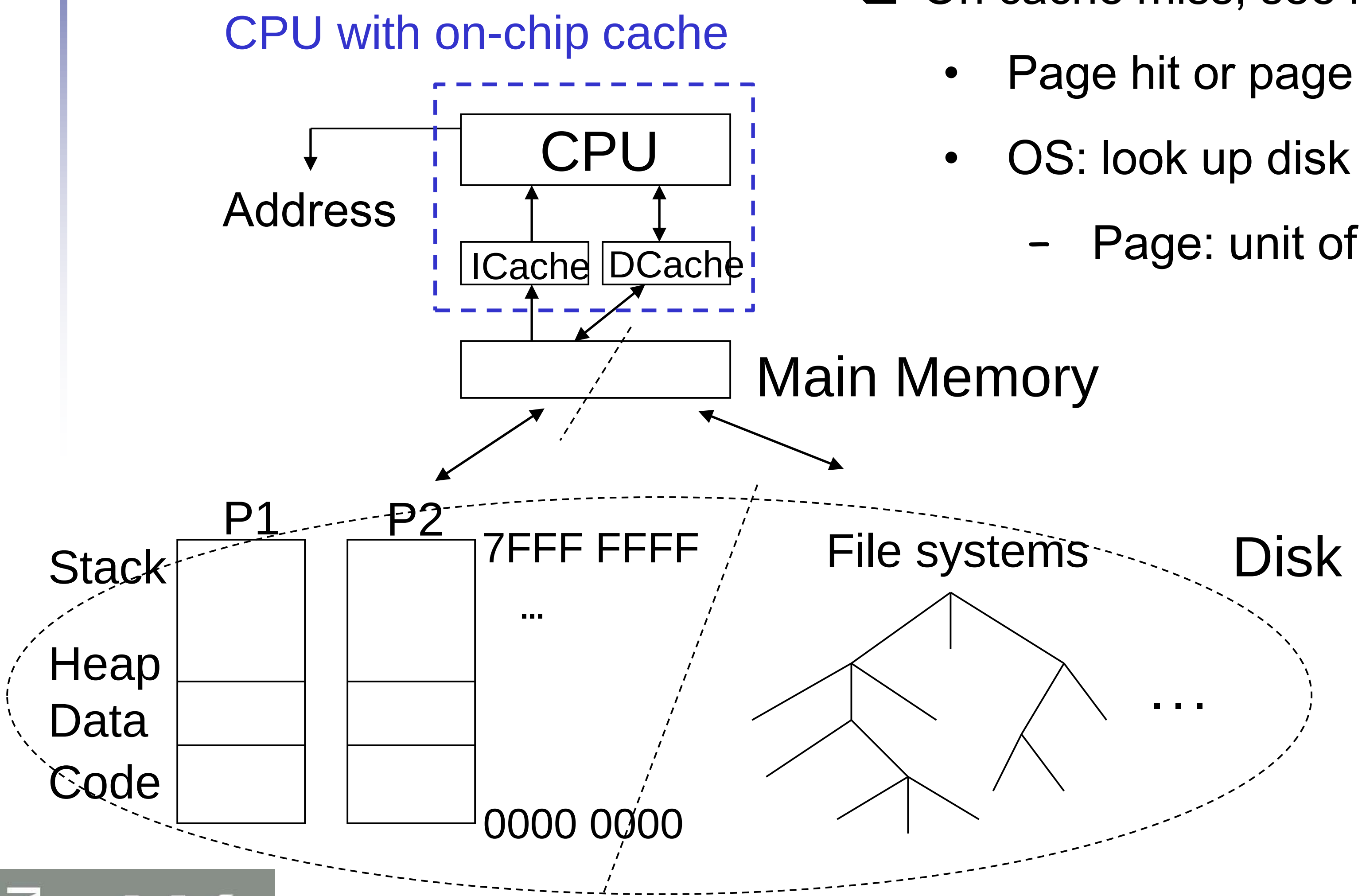


Work done by hardware!
(OS not know about cache)

- Cache block (or line): unit of copying
  - May be multiple words
- If requested data in cache memory
  - Cache hit
    - Cache hit rate: hits/accesses
- If requested data is absent
  - Cache miss
    - Time taken: miss penalty
    - Cache miss rate
      = 1 – cache hit rate
  - 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



Memory Management

- Cache memory management (Architecture topic)
  - Cache parts of main memory
  - Implemented by hardware: fast, simple
    - Part of processor (on-chip cache)
    - Hardware accelerator: SW not know about it
- Virtual memory management (OS topic)
  - Use main memory as cache for disk
  - Implemented by software
    - Disk access is already slow (10 ms)
- Same principles (caching, locality, management) for both
  - Usage: independent of each other

Cache Memory

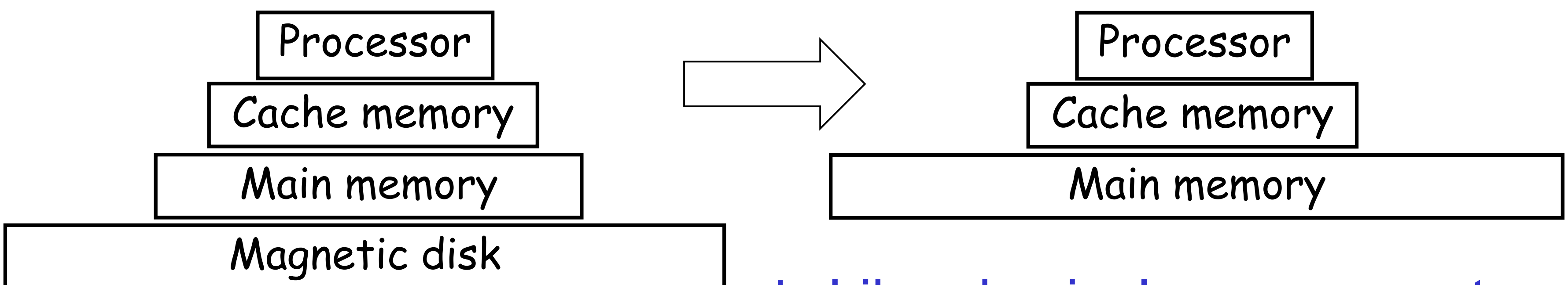
Cache Memory

- 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 means that we assume main memory is ideal
  - Assume that size of main memory is infinite

Cache Designer's Perspective

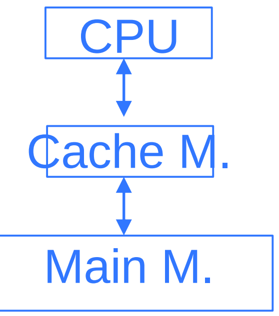
- Page fault and associated performance loss
  - Not something that cache designer can control
    - OS and I/O design issue
- Separation of concern

† Cache memory designer's view



† Like physical memory system with everything in it

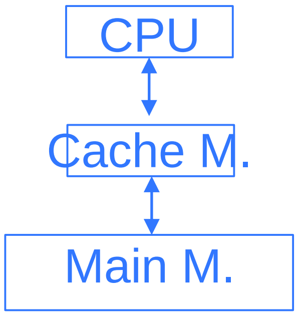
Cache Memory in Operation



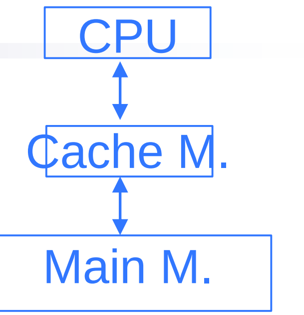
- Read hits
  - This is what we want
- Read misses (memory stalls)
  - Stall CPU (pipe), fetch block from memory, restart
- Write hits
  - Update data in cache and memory (write-through)
  - Update data in cache (write-back)
- Write misses (memory stalls)
  - Stall CPU, fetch block from memory, write,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

Cache Memory Performance

- Suppose that cache memory access time is 1 clock cycle
  - Miss penalty: 50 cycles, miss rate: 0.02
- Average memory access time
  - $1 + 0.02 \times 50 = 2$ cycles
  - At every IF or DM, pipeline stalls 1 cycle
- CPI increase due to memory
  - Frequency of load and store: 20%
  - CPI: $1 \rightarrow 2.2$ (80%: lose 1 cycle, 20%: lose 2 cycles)
- What if there is no cache memory?
- Memory is slow – much more damaging than hazards



Cache Memory Performance



- Cache hit/miss
  - Hit rate, miss rate
- Average memory access time (performance model)
  - 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)

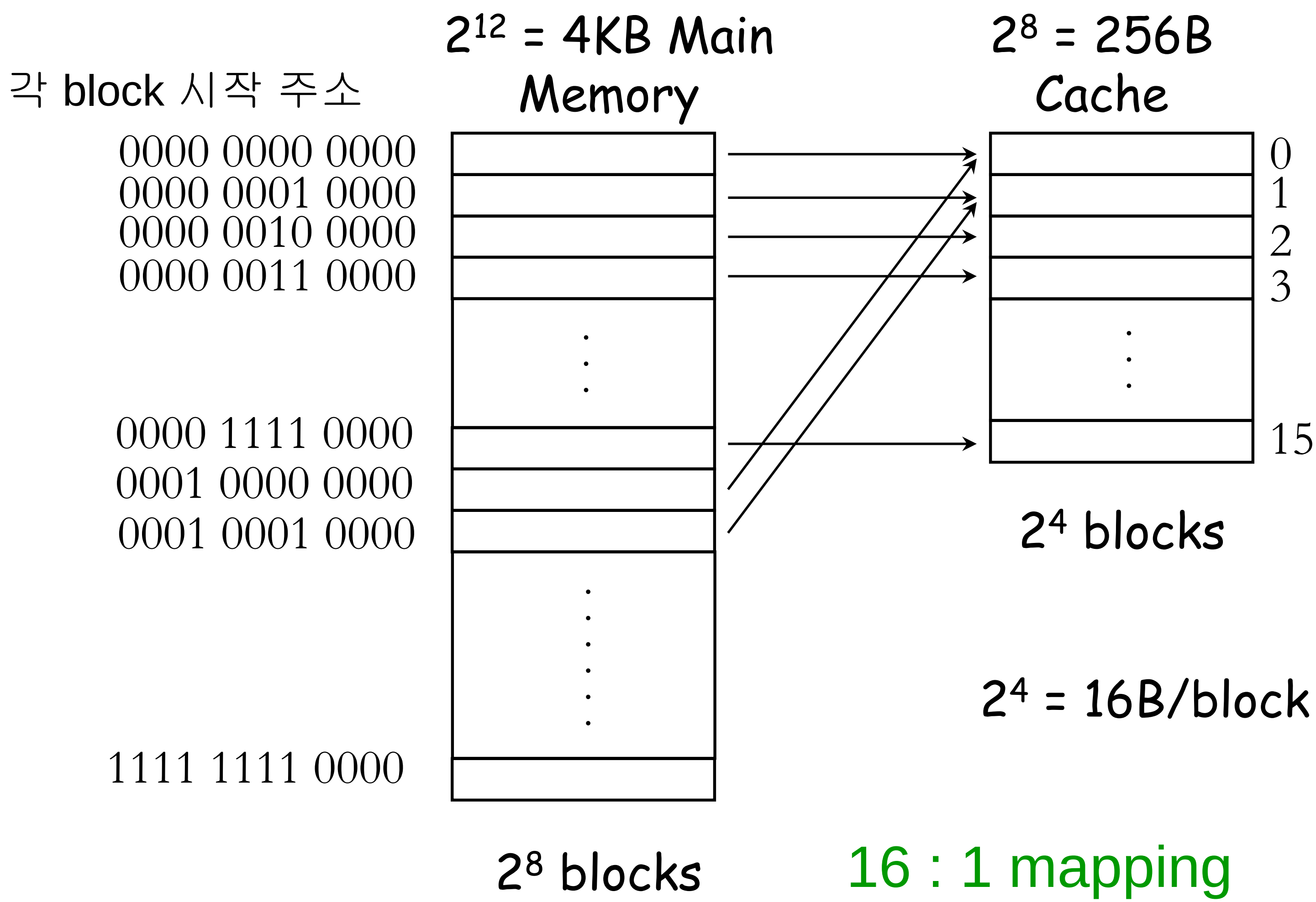
Cache Memory:

Structures and Operations

Cache Memory Design

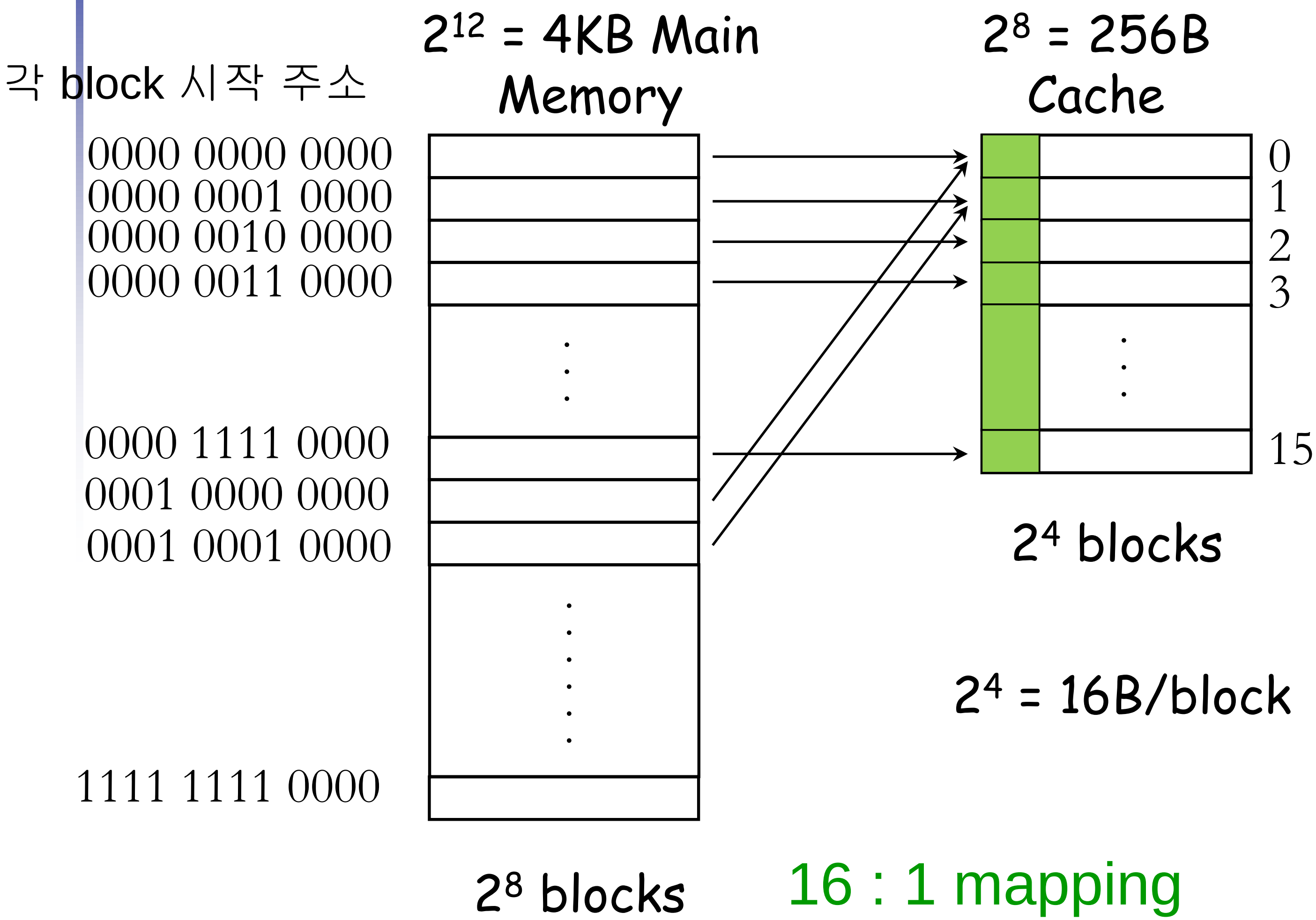
- Cache memory: smaller than main memory
  - Where to place a block (placement issue)
  - How to find it later (identification issue)

Direct Map Cache: Placement



16 : 1 mapping

Direct Map Cache: Identification



16 : 1 mapping

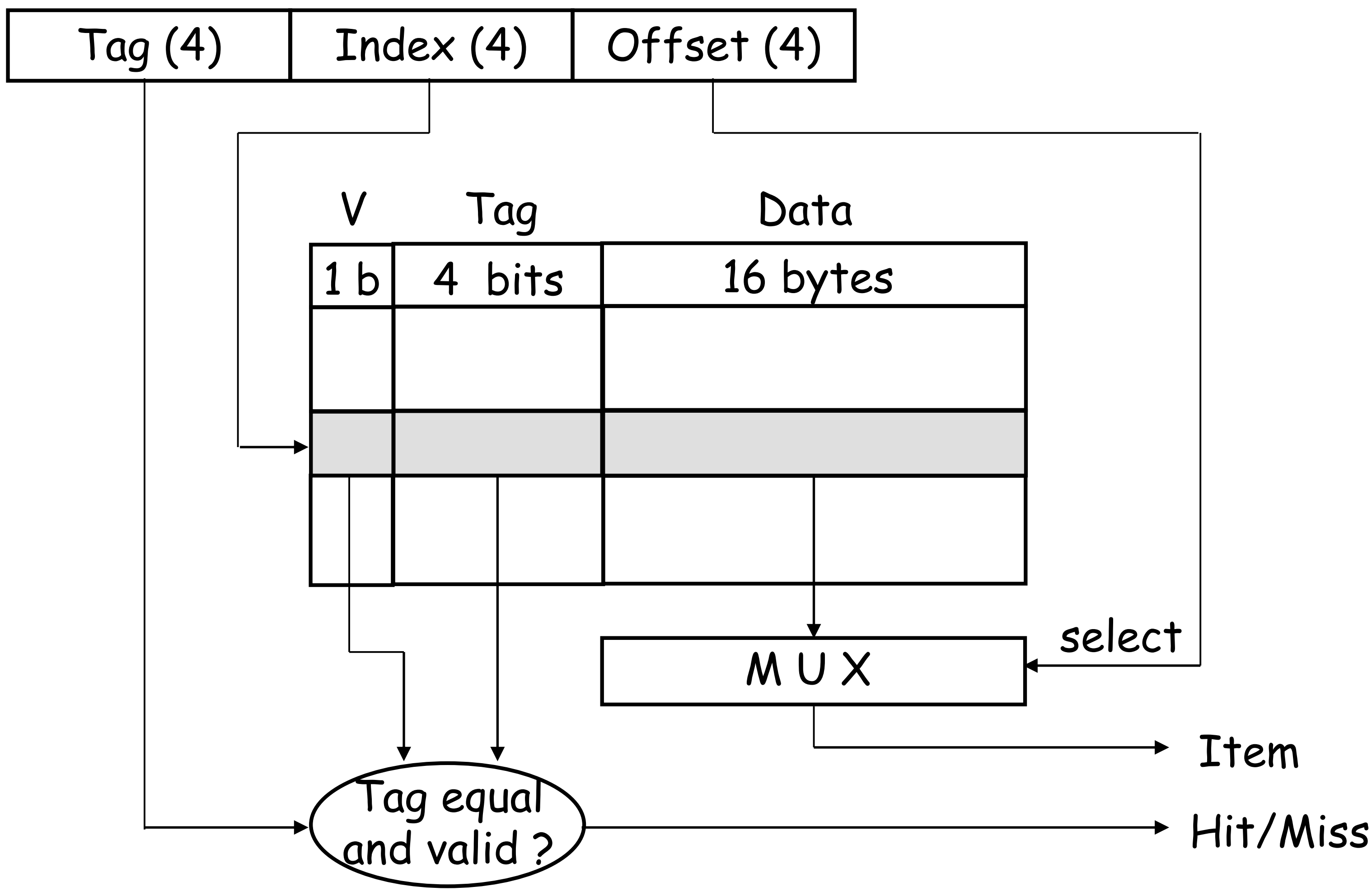
Tag Index Byte offset

0000 0000 0000
0000 0000 0001
0000 0000 1111

0000 0001 xxxx
0000 0010 xxxx
0000 1111 xxxx

0001 0000 xxxx
1111 0000 xxxx

Direct Map Cache: Identification



Quiz

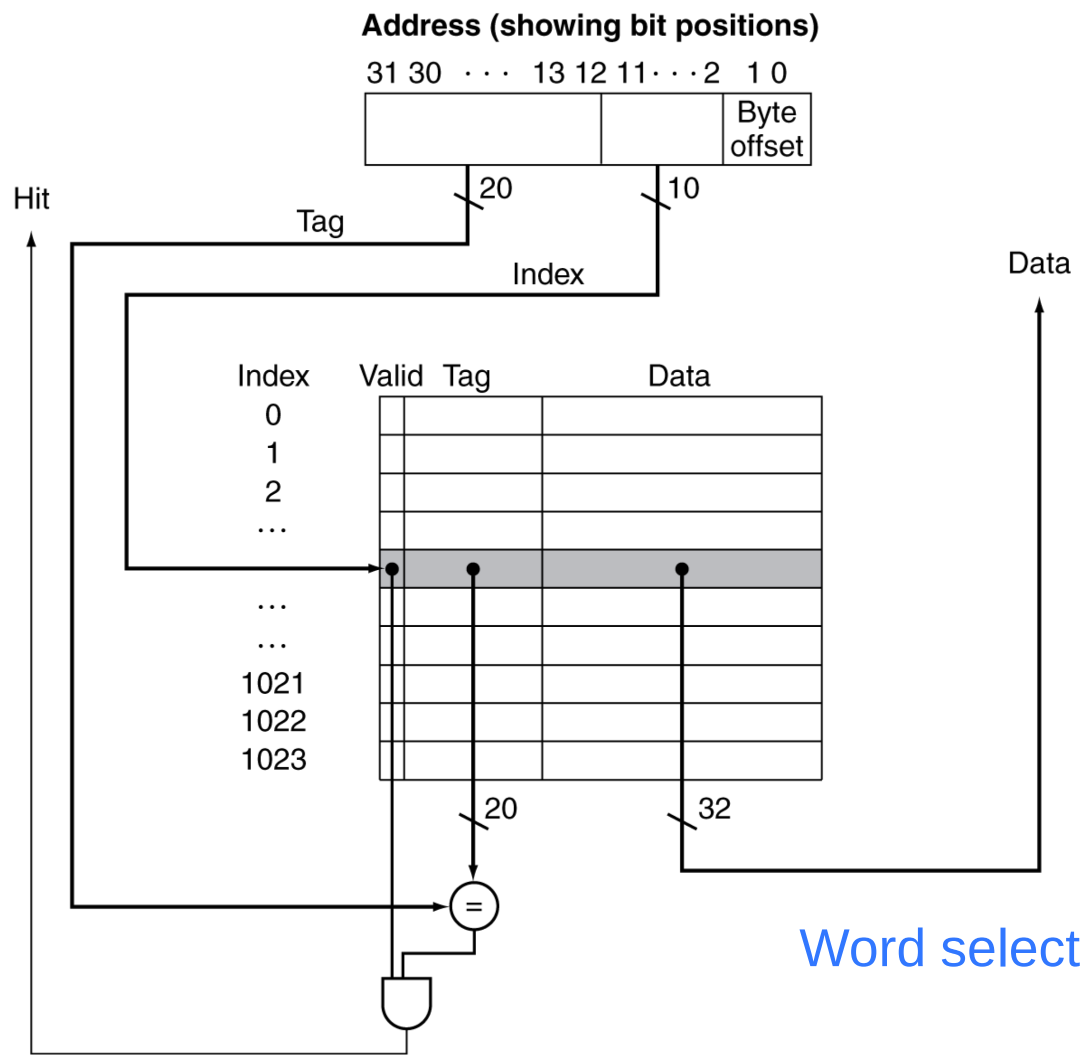
- In our example, tag: 4 bits (main is 2^4 times larger),
  index: 4 bits (2^4 blocks in cache),
  byte offset: 4 bits (2^4 bytes/block)
- What if we reduce the size of cache memory to half? (all other parameters do not change)
  Tag++, index--
  - Number of bits for tag, index, and byte offset
- What if we reduce the block size to half?
  Index++, BOffset--
- What if the size of main memory is doubled?
  Tag++

Real Direct Mapped Cache

- What kind of locality are we taking advantage of?

4KB direct map cache with block size of 4 bytes

$2^{10} * 2^{2} = 4K$



Word select

Real Direct Mapped Cache

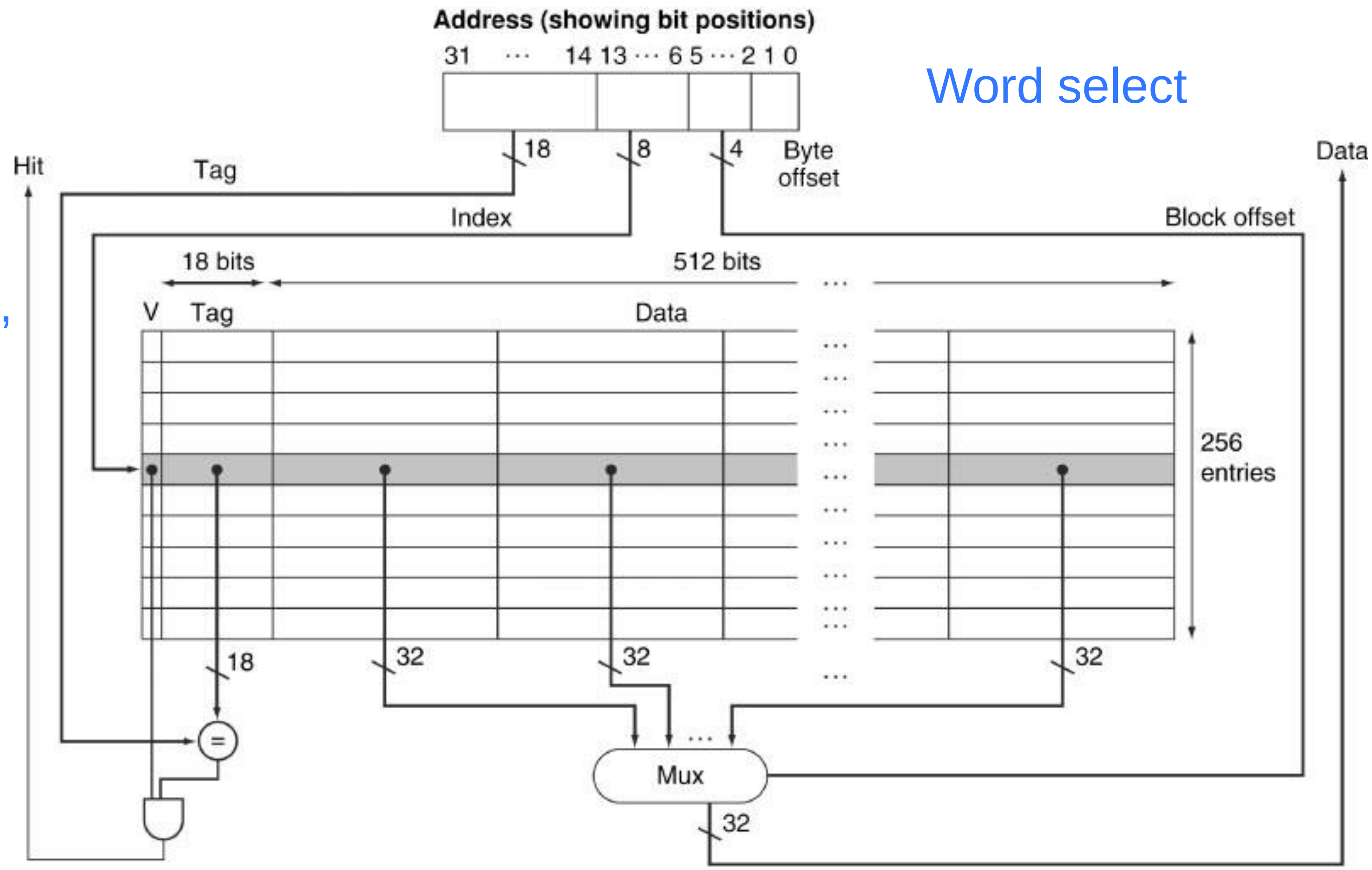
- Taking advantage of spatial locality

Word select

16KB DM cache,

block size: 64 B

$2^8 * 2^6 = 16K$



Cache Simulation

Cache Simulation

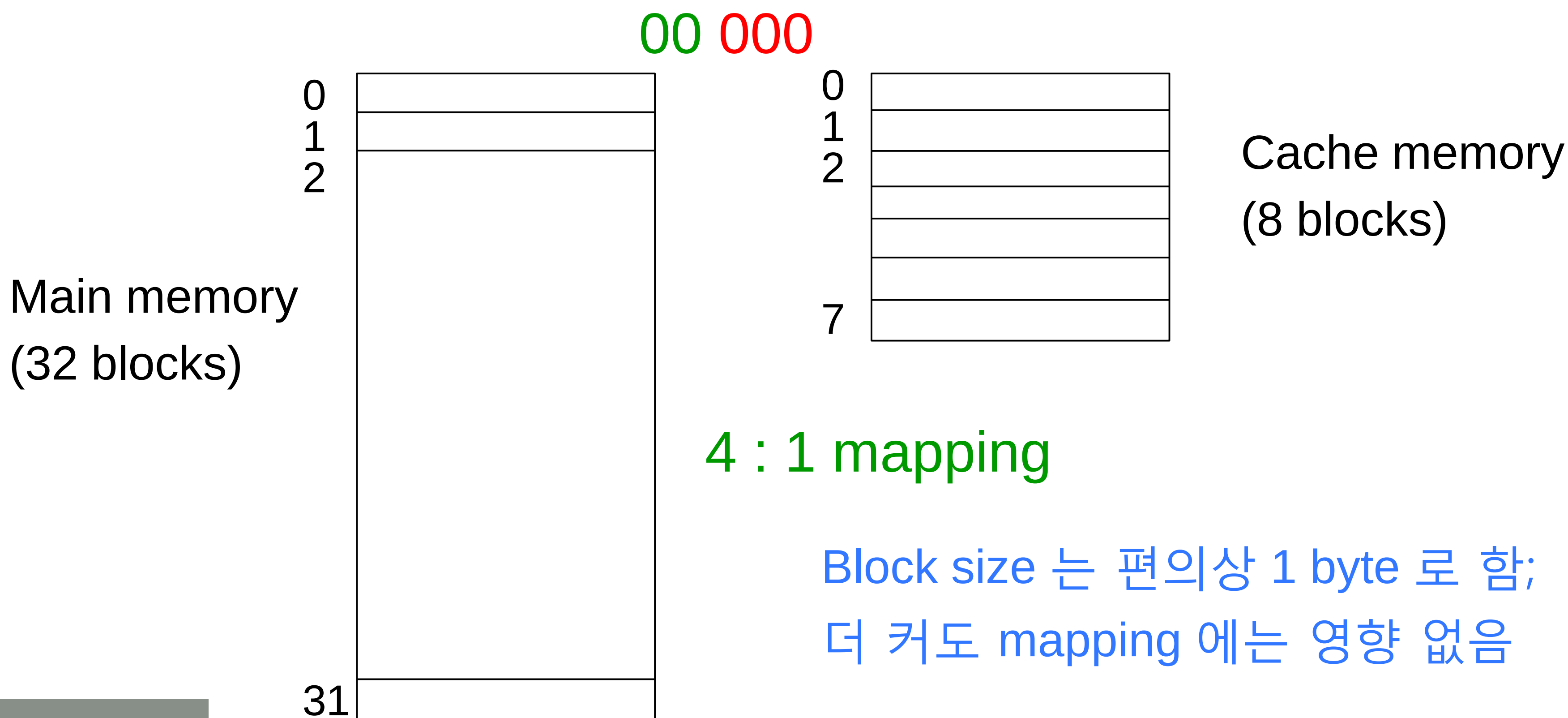
- Now let's look into a small cache memory system
  - Input: addresses from CPU,

    cache configuration (size, mapping, block size)
  - Output: cache hit rate

Cache Simulation

- 5-bit address, 8-byte direct map cache, 1 byte/block

  (tag: 2 bits, index: 3 bits, byte offset: 0 bit)



Block size 는 편의상 1 byte 로 함;
더 커도 mapping 에는 영향 없음

Cache Simulation

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

Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
10 110	miss	110
11 010	miss	010
10 110	hit	110
11 010	hit	010
10 000	miss	000
00 011	miss	011
10 000	hit	000
10 010	miss	010

Address trace

Address trace:

10 110
11 010
10 110
11 010
10 000
00 011
10 000
10 010

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	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Dlata
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memorlly (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

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

e. After handling a miss of address (00011_{two})

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

f. After handling a miss of address (10010_{two})

Cache Simulation

- In a real memory system design
  - Use address traces from benchmarks
    - Size of address trace
    - Need specialized HW tools
  - Simulation done by software (cache simulator)
    - Input: cache size, block size, mapping
    - Output: miss rate
  - Cache simulation homework
    - You may develop a cache simulator

Cache Memory: Performance

Cache Performance (How to Improve?)

- Average access time (앞에서의 예, $1 + 0.02 \times 50 = 2$)

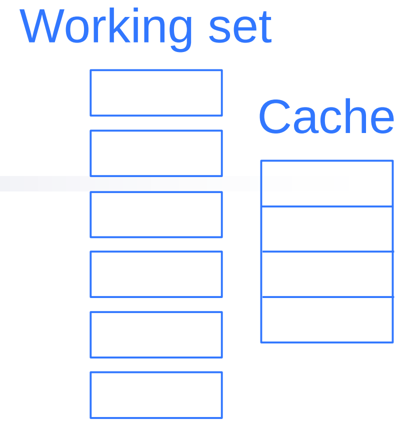
  $= \text{Cache hit time} + \text{miss rate} \times \text{miss penalty}$
- Three ways of improving performance
  - Decreasing hit time
  - Decreasing miss rate
  - Decreasing miss penalty
- What if you increase total cache size?
- What if you increase block size (from 1 to ∞)?
- What if miss penalty is too high?

Hit Time

- What if we increase the size of cache memory?
  - Which factor is improved?
    - Any side effect?
- Always consider the average memory access time!

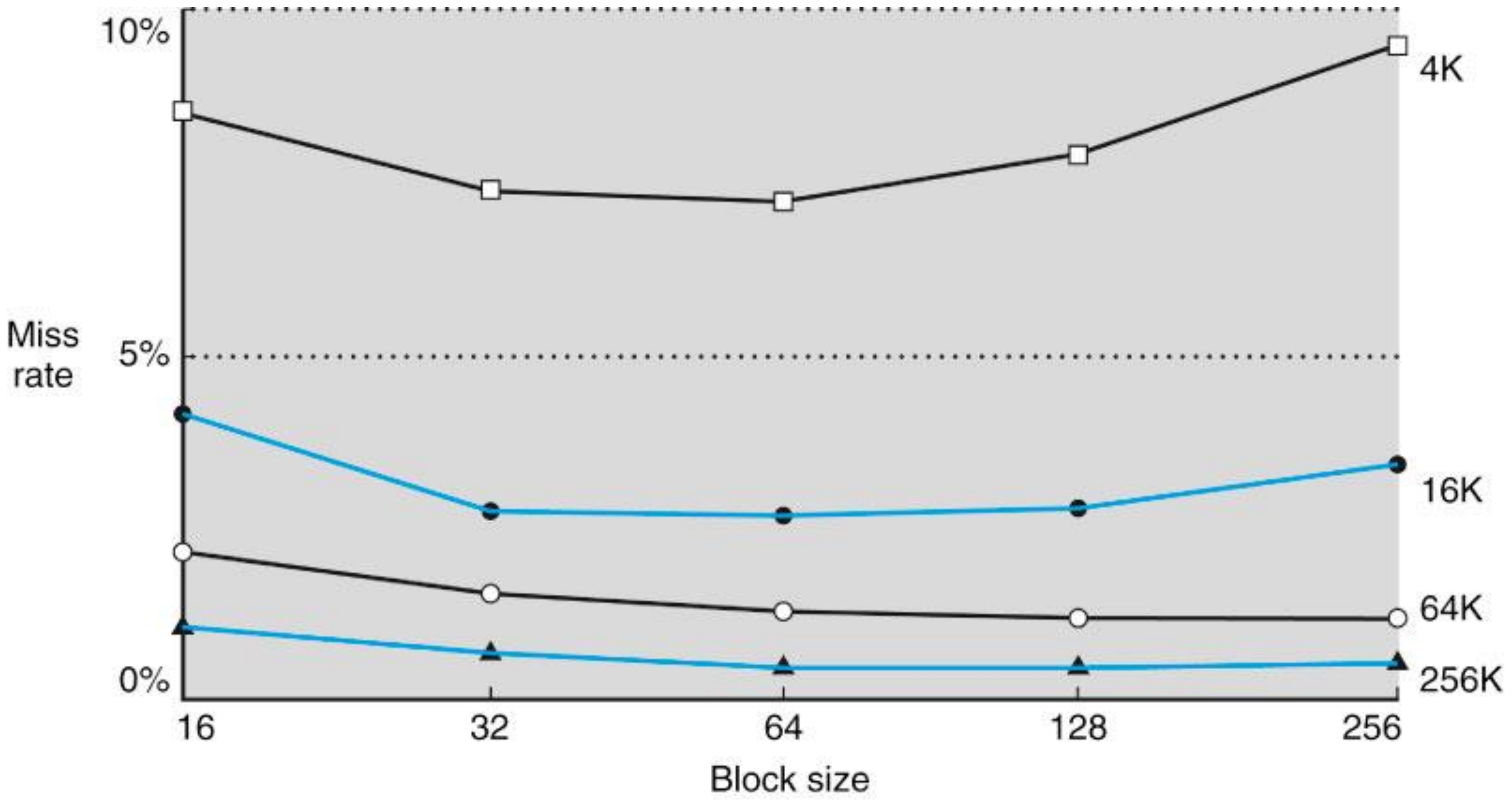
- Let's consider the block size
  - Changing block size not affect cost, still large impact

Block Size Considerations



- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache (notion of working set)
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate
- Larger block: larger miss penalty
  - May override benefit of reduced miss rate
  - Early restart and critical-word-first can help (skip)
- Always consider the average memory access time!

Block Size

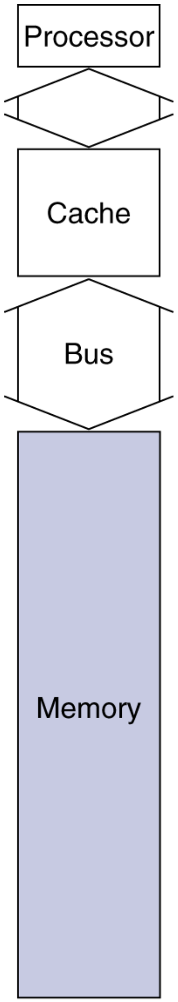


- Sweet spots exist - find it with cache simulation

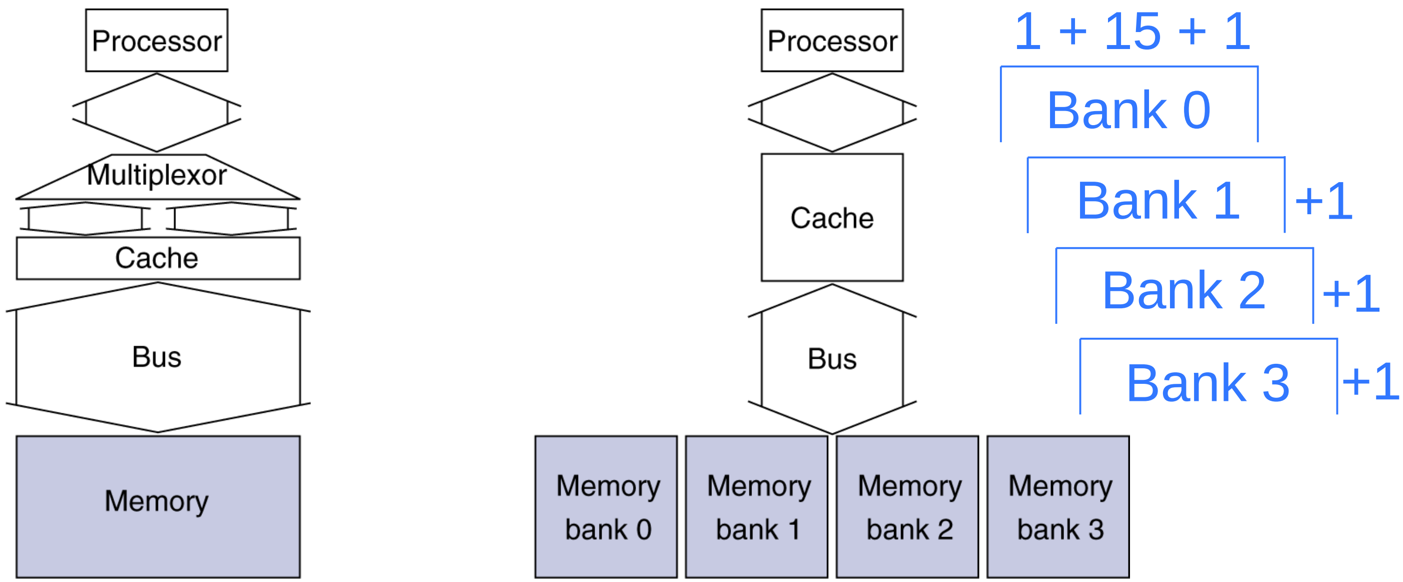
Miss Penalty and Main Memory

- Use DRAMs for main memory
  - Connected by fixed-width (e.g., 1 word) clocked bus
- 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 = $1 + 4 \times (15 + 1) = 65$ bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

Increasing Memory Bandwidth



a. One-word-wide memory organization

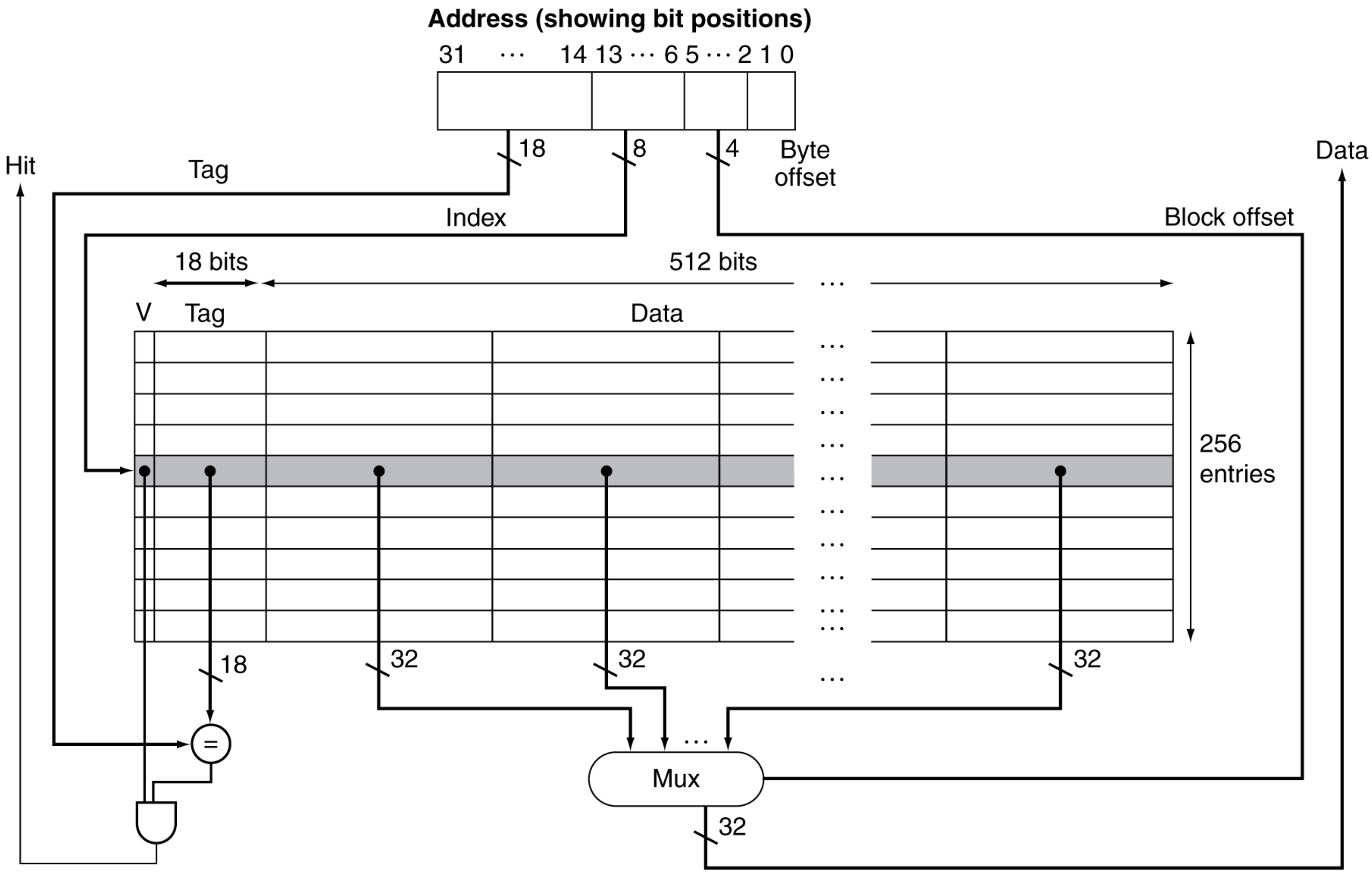


b. Wider memory organization

c. Interleaved memory organization

- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Example: Intrinsity FastMATH (skip)



Example: Intrinsity FastMATH

- Embedded MIPS processor
  - 12-stage pipeline
- 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%

Where We Are

- ❑ Physical memory and virtual memory
- ❑ Memory hierarchy
- ❑ Cache memory
  - Concepts and terminology
  - Direct-map cache: structures and operations
  - Performance
    - Hit time, miss rate, miss penalty
    - Consider average memory access time
    - Cache simulation
  - ✓ Different mappings (miss rate)

Homework #13 (see Class Homepage)

1) Write a report summarizing the materials discussed in Topic 5-1 (이번 주 수업 내용)

** 문장으로 써도 좋고 파워포인트 형태의 개조식 정리도 좋음

- Due: see Blackboard
  - Submit electronically to Blackboard

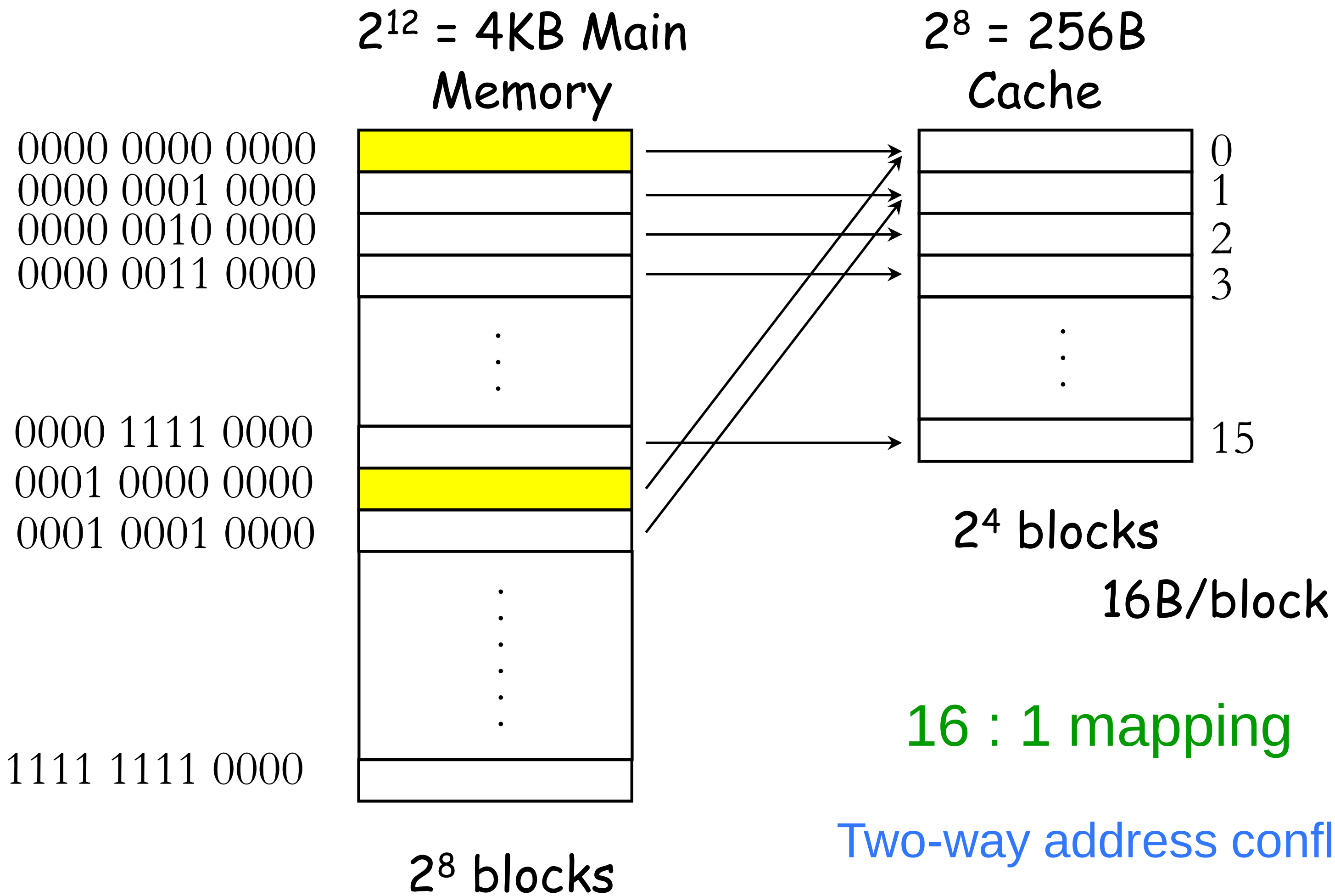
Cache Memory:

Set-Associative Mapping (1)

Direct Mapping and Address Conflict

- Simplest and fastest mapping
  - No choice in placement, identification, replacement
  - Shorter clock cycle, widely used
- Problem of miss rate
  - Address conflict: 2-way conflict, 4-way conflict, …
    - Miss rate may go up
    - Less a problem when cache is large enough
  - Set-associative mapping can be a solution

Direct Map Cache: Placement (반복)

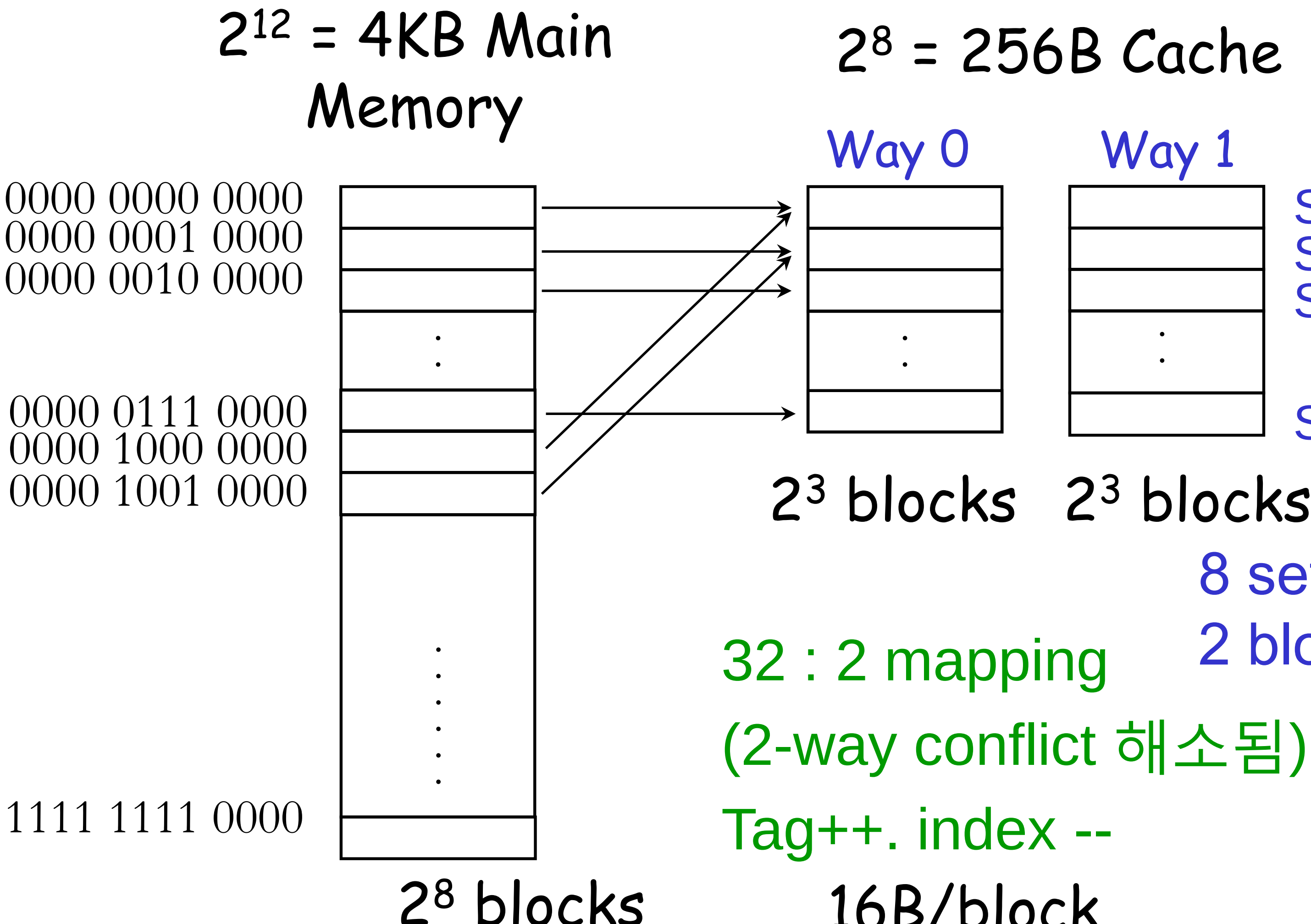


16 : 1 mapping

Two-way address conflict, real industry story

Two-Way Set-Associative Cache

(Why the name?)



8 sets,
2 blocks/set

32 : 2 mapping

(2-way conflict 해소됨)

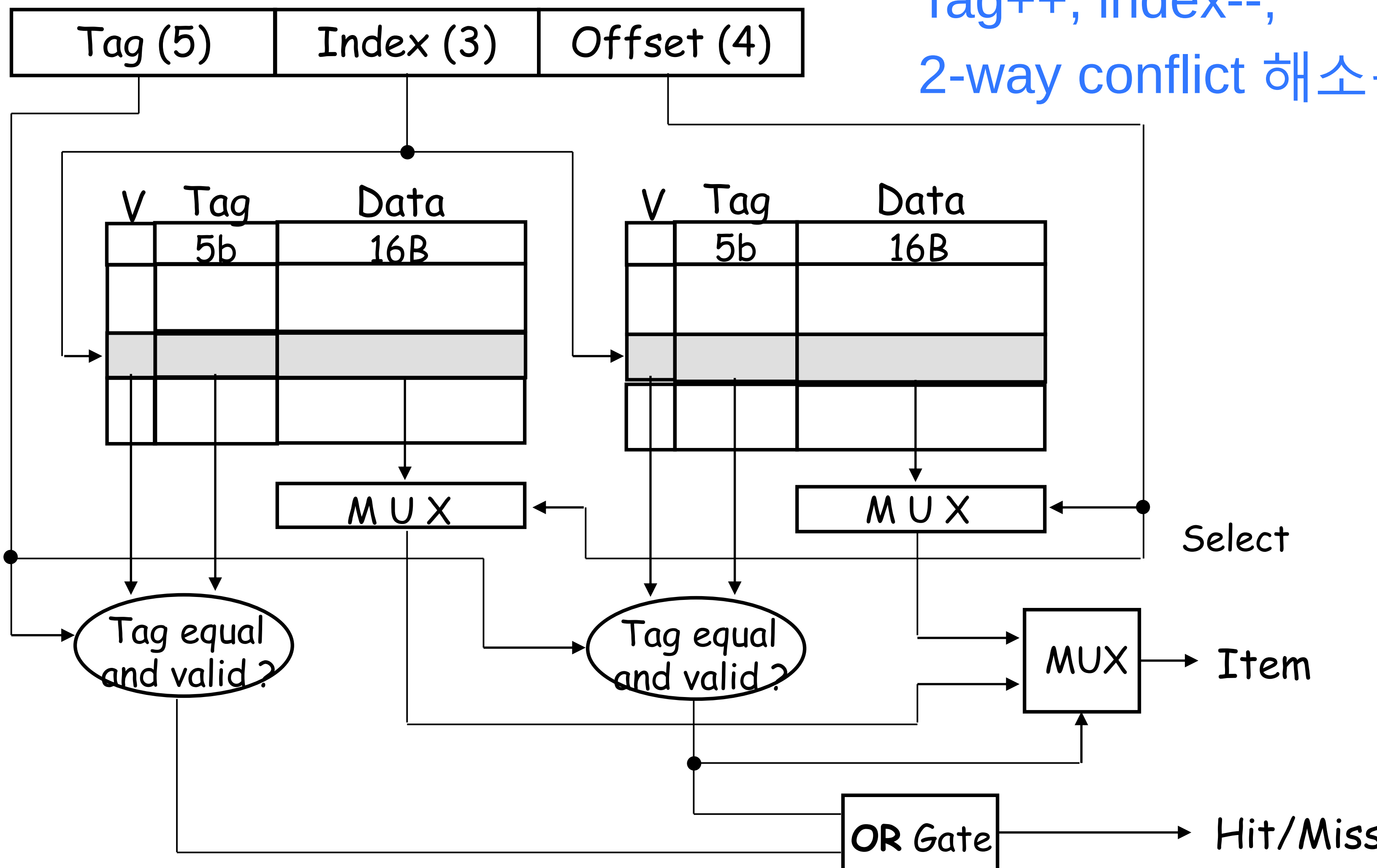
Tag++. index --

16B/block

Two-Way SA Cache: Identification

Tag++, index--;

2-way conflict 해소됨



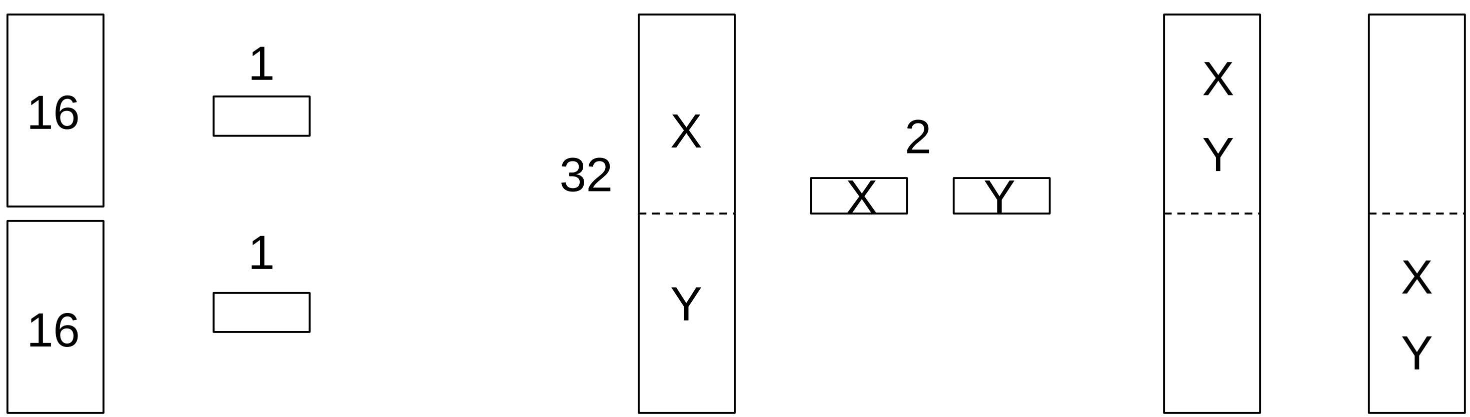
Two-Way SA Cache

- How do we use an address?

12-bits

5	3	4
Tag	Index	Byte offset

- 32-to-2 mapping; can you see more freedom?



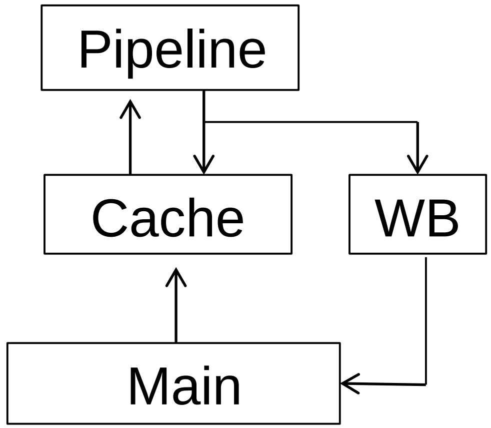
- What do we gain? What do we lose?

Four Issues in Memory Management

- Q1: placement (mapping)
- Q2: identification
- Q3: write strategy
  - Write-through: simple and consistent
  - Write-back: may reduce memory traffic
- Q4: replacement policy
  - Least recently used (LRU) and reference bit
    - LRU is costly and approximated

Write-Through

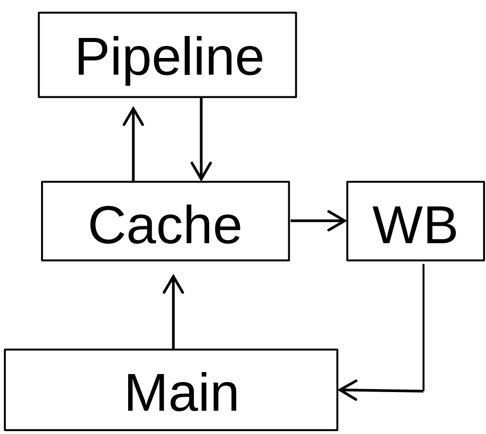
- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer (high miss penalty)
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 50 cycles
    - Effective CPI = $1 + 0.1 \times 50 = 6$
- Solution: write buffer
  - Only stalls on write if write buffer is already full



Write-Back

- On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
    - Dirty bit for each block (dirty vs. clean)

V	D	R	Tag	Data block
1	1	1		



- When a dirty block is replaced
  - Write it back to memory
  - Can use write buffer to reduce miss penalty

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
- Write through is easier to implement
  - Misses are simpler and cheaper (no write to lower memory)
- Think about shared-bus multiprocessors and write back

Write Allocation (참고)

- 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

Cache Memory:

Set-Associative Mapping (2)

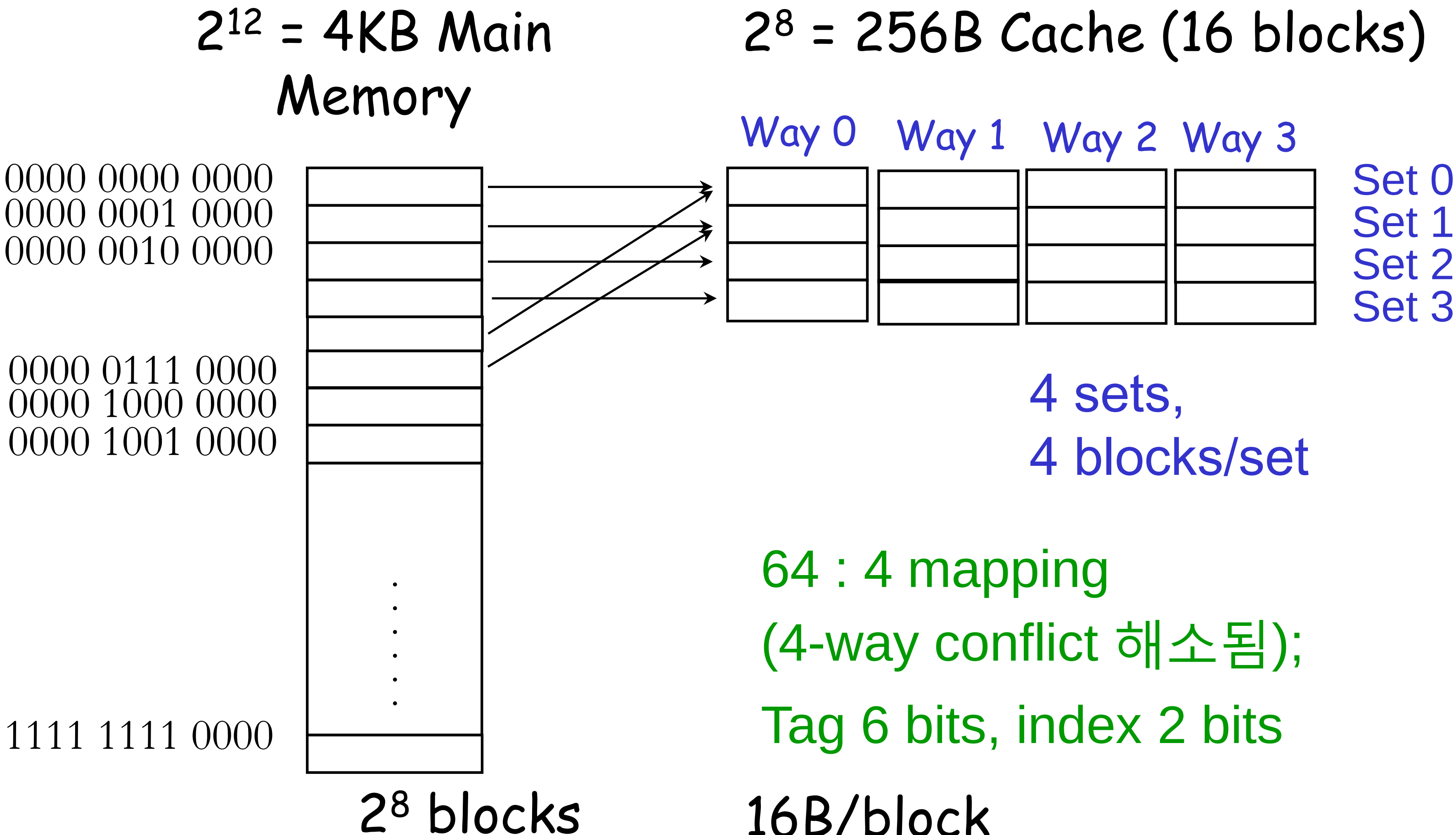
4, 8, 16-Way Set-Associative Cache

- Can you imagine more freedom: 4-way, 8-way, … ?

	Tag	Index	B. Offset	mapping
DM	4	4	4	16:1
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

(16-way means complete freedom in our example)

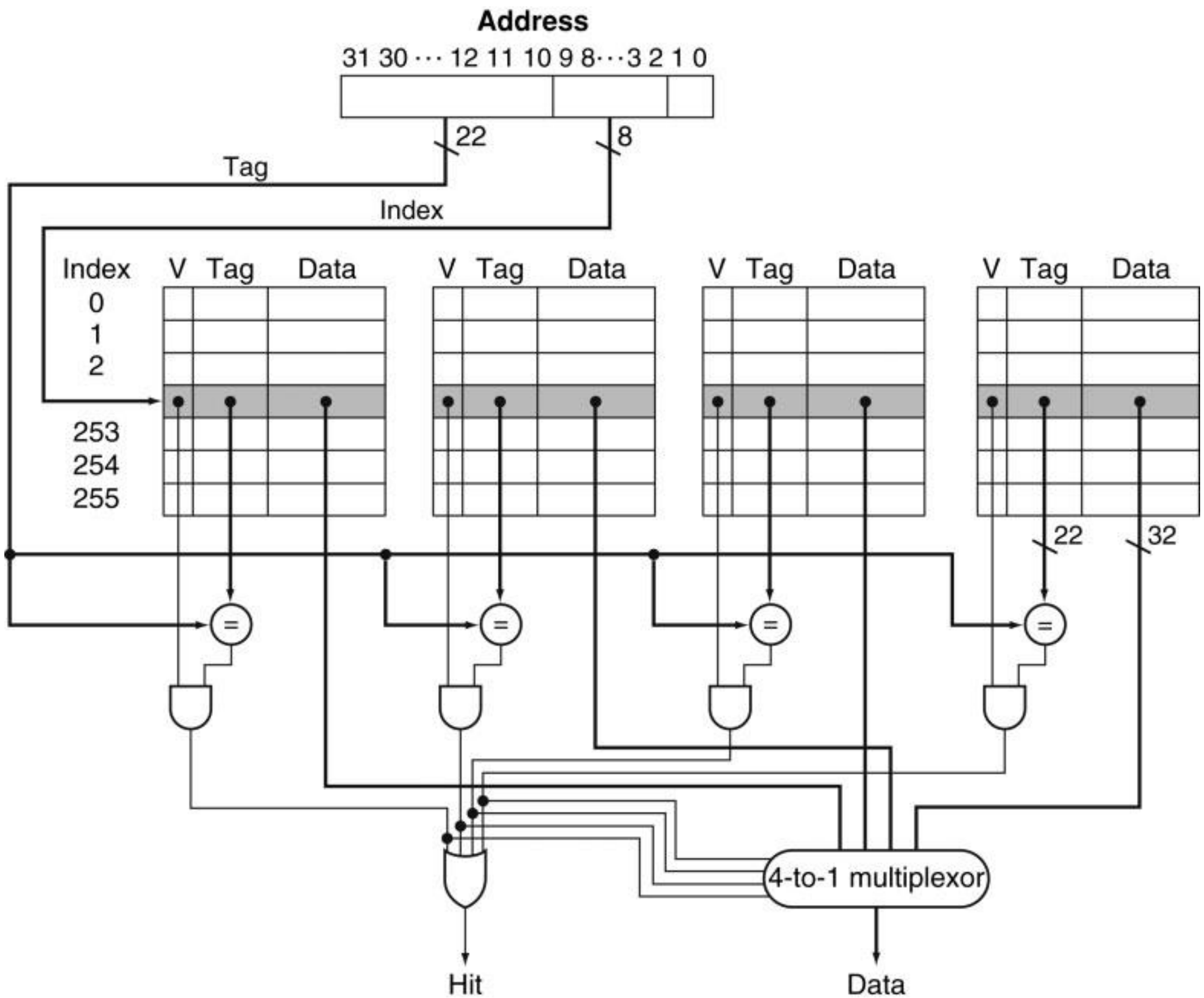
Four-Way Set-Associative Cache



4 sets,
4 blocks/set

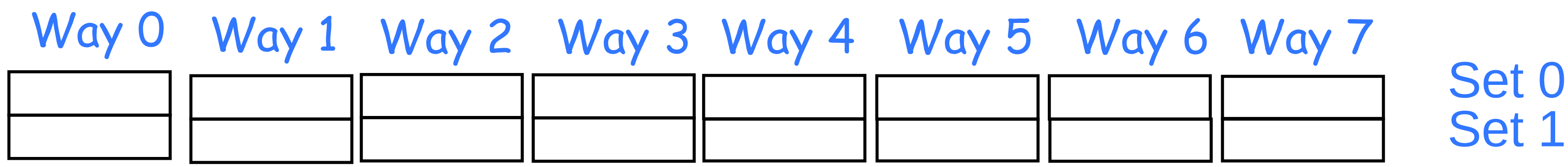
64 : 4 mapping
(4-way conflict 해소됨);
Tag 6 bits, index 2 bits

4-Way SA Cache: Identification(참고)



8-Way Set-Associative Cache

256 blocks in main → 16 blocks in cache

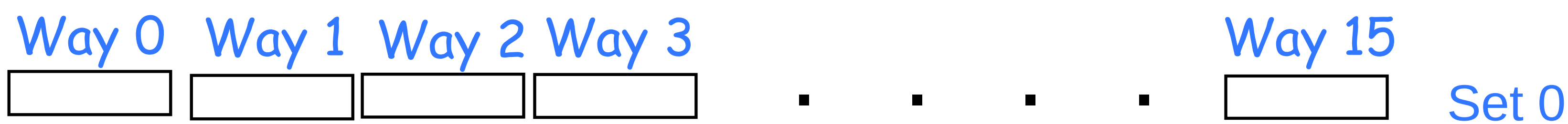


2 sets, 8 blocks/set

	Tag	Index	B. Offset	mapping
DM	4	4	4	16:1
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

16-Way Set-Associative Cache

256 blocks in main → 16 blocks in cache



1 set, 16 blocks/set

	Tag	Index	B. Offset	mapping
DM	4	4	4	16:1
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

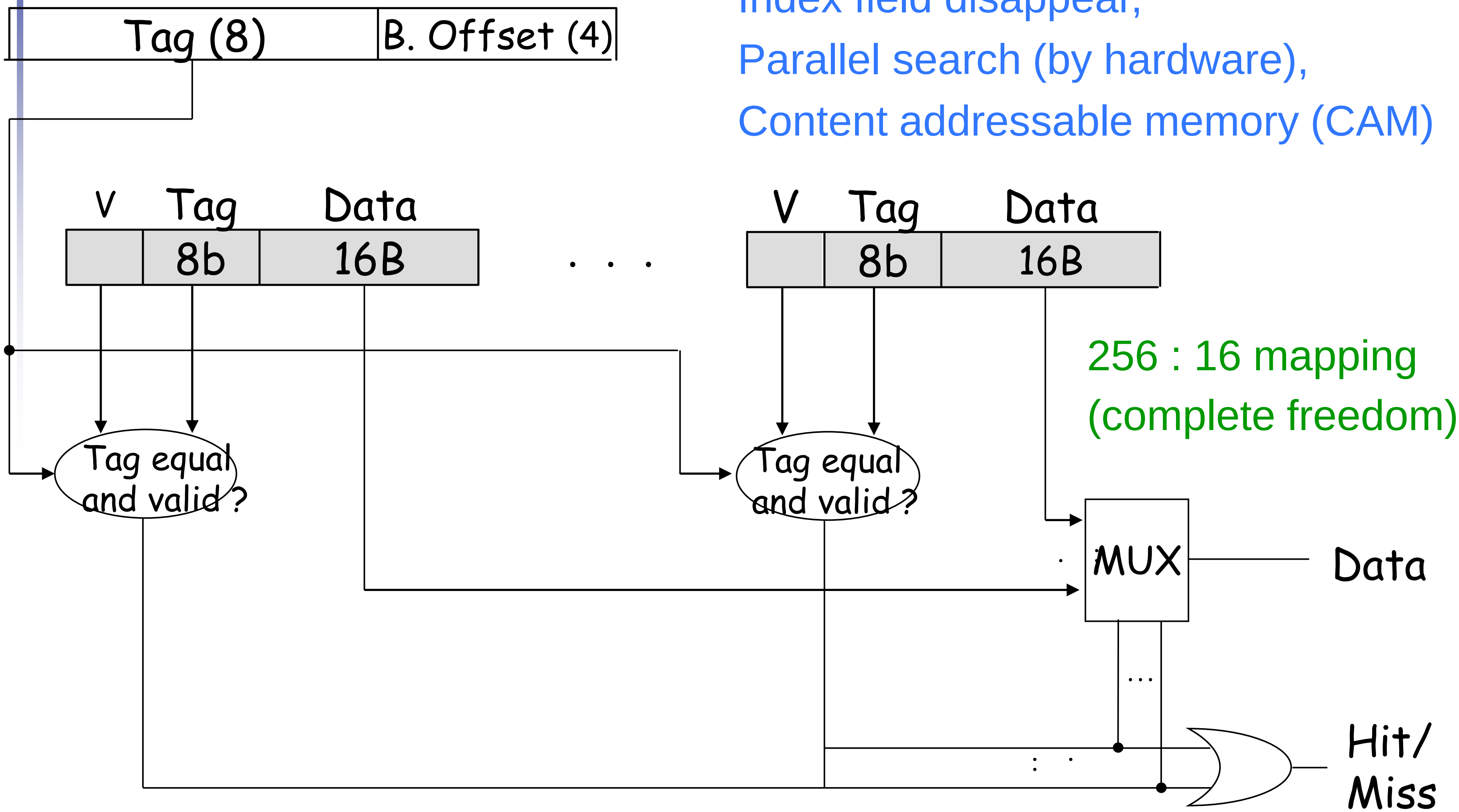
(16-way means complete freedom in our example)

Fully-Associative Mapping

Index field disappear,

Parallel search (by hardware),

Content addressable memory (CAM)



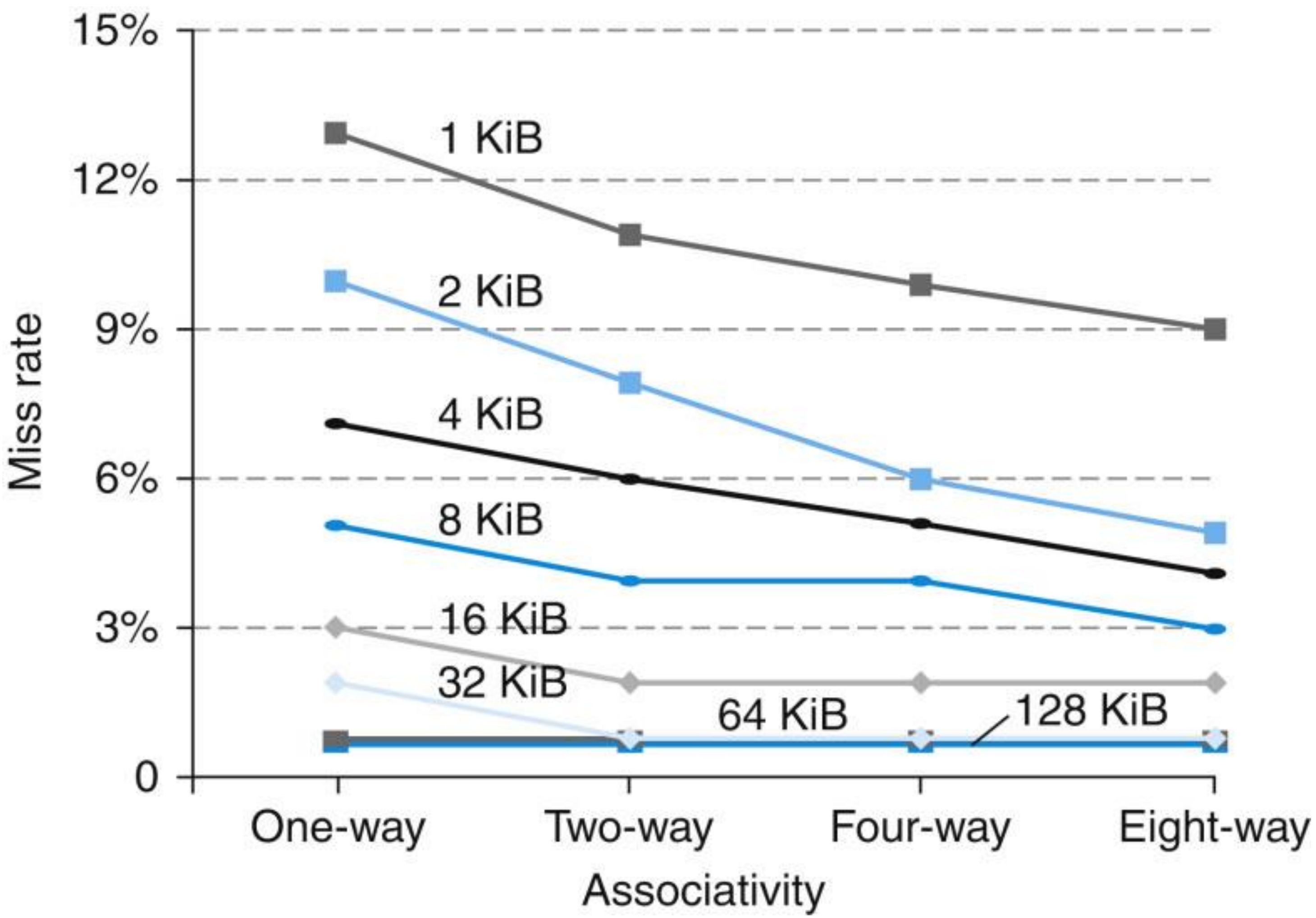
256 : 16 mapping

(complete freedom)

4, 8, 16-Way Set-Associative Cache

- Can you imagine more freedom: 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 hit time, good hit rate
    - Can be used in small cache (e.g., TLB)

Performance

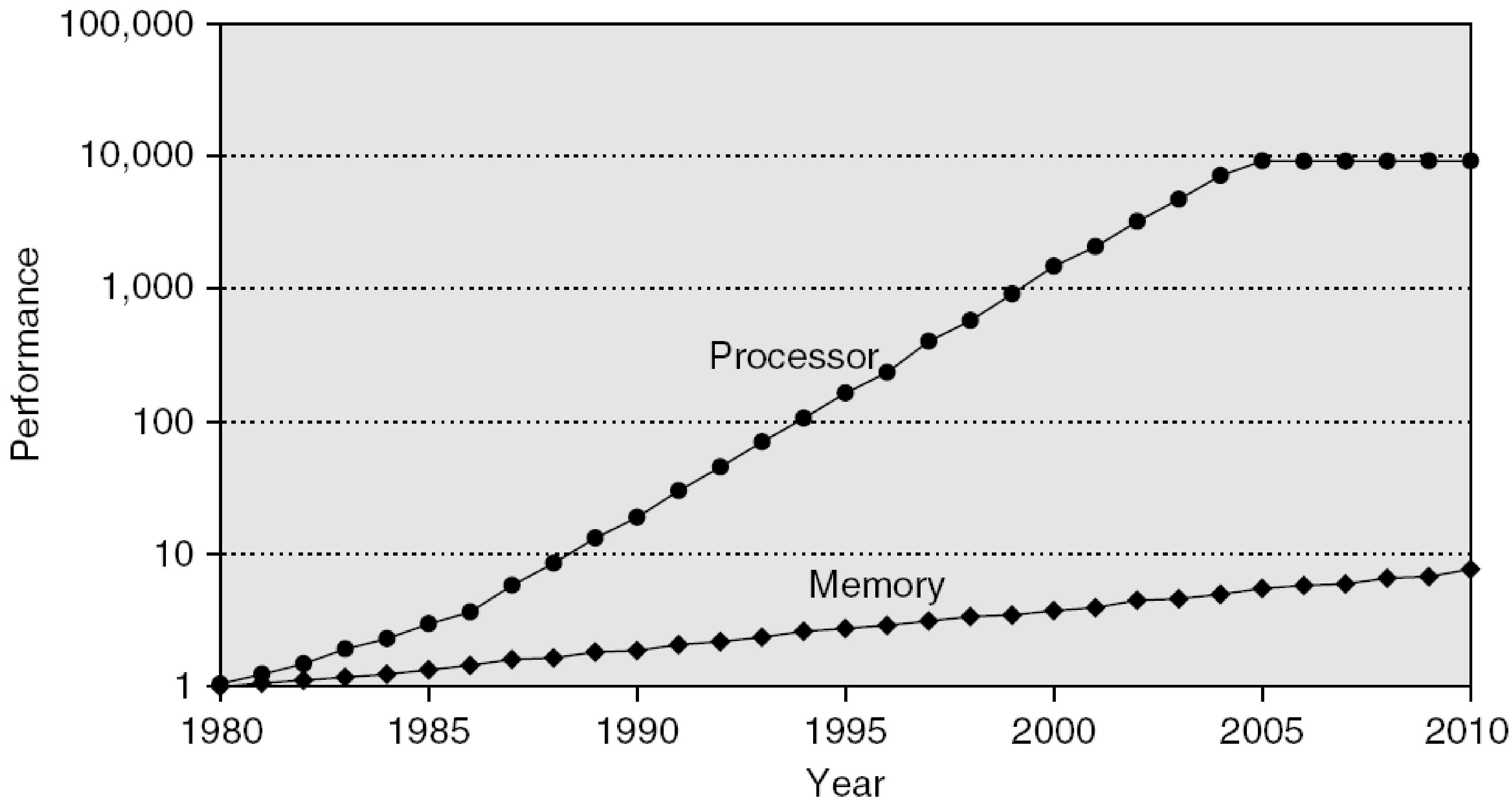


Performance

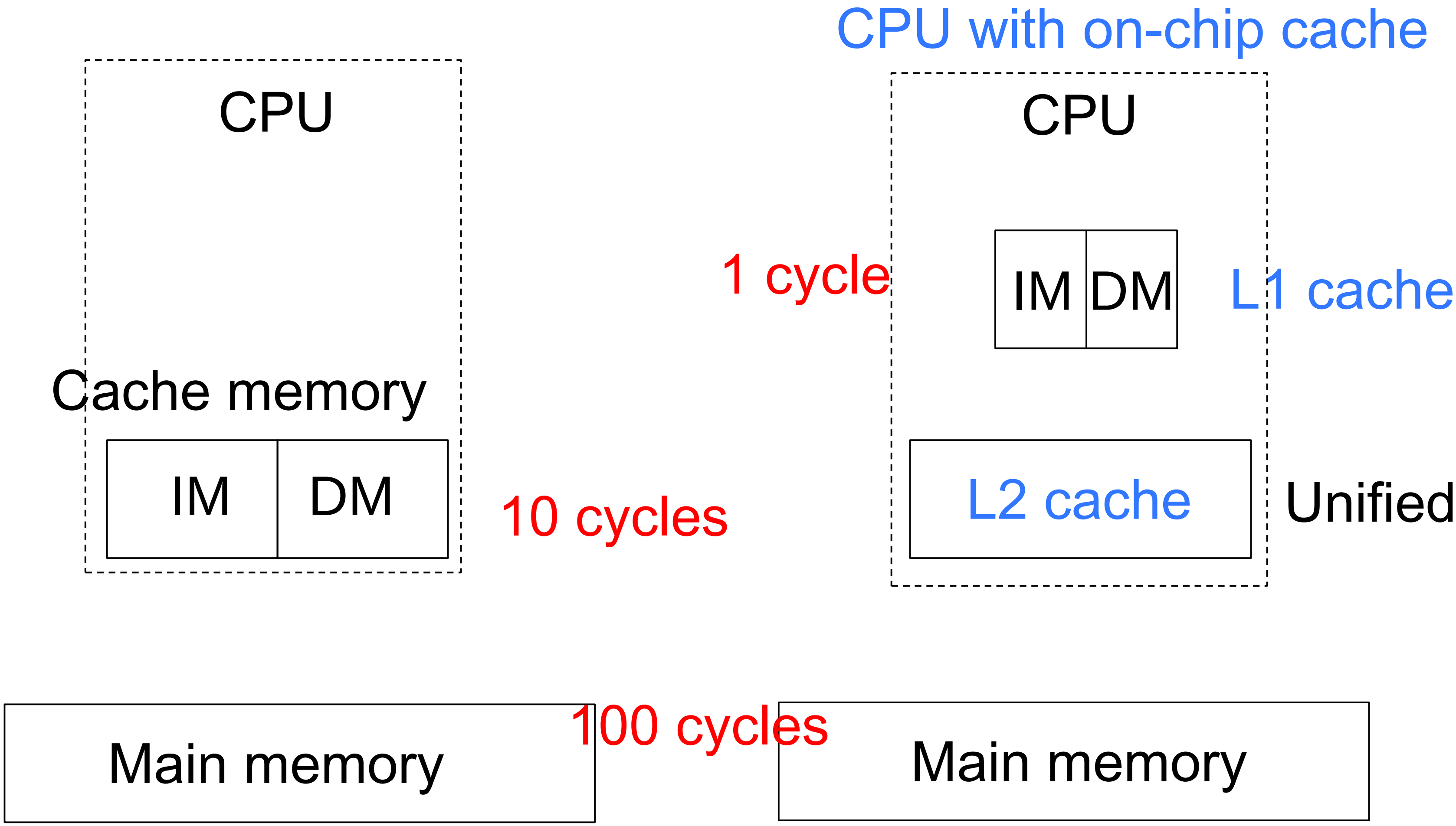
- As associativity increases, miss rate drops
  - Largest gain: from direct map to 2-way SA
- As cache size increases
  - Miss rate drops
  - Impact of associativity becomes smaller
- May use Large DM cache for higher clock speed

Multilevel Caches

CPU-Memory Performance Gap



Multilevel Caches (Two-Level)



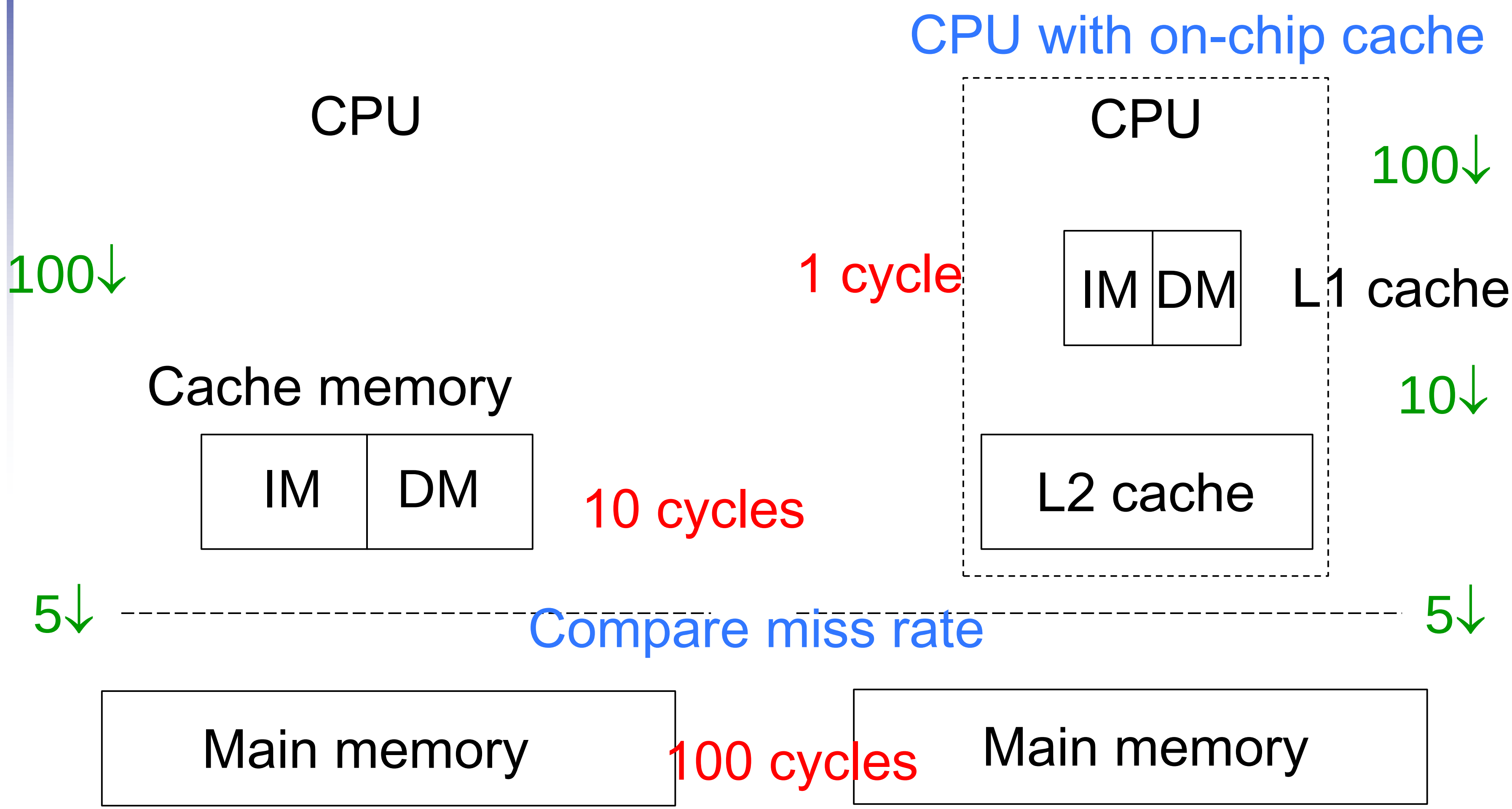
- Why not "split" L2 cache or main memory?

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

Multilevel Caches (Two-Level)



Multilevel Caches

- Performance: why use them? (Fast hit and what else?)
  - 1-level versus 2-level cache (with same last level size)
    - If cache is reasonably big, show similar miss rates
- 1-level caches

  Average memory access time (AMAT)

  $= \text{hit time} + \text{miss rate} \times \text{miss penalty}$

L1 miss penalty
or AMAT at L2

- 2-level caches

  $\text{AMAT} = \text{L1 hit time} + \text{L1 miss rate} \times (\text{L2 hit time}$

  $+ \text{L2 miss rate} \times \text{L2 miss penalty})$

Multilevel Caches

- Average access time in 1-level caches
  - L2 only: $10 + 0.05 \times 100 = 15$ cycles
- Average access time in 2-level caches
  - L1 and L2: $1 + 0.1 \times (10 + 0.5 \times 100) = 7$ cycles

Multilevel Caches

- Two-level caches
  - Faster hit time
  - Smaller average access time
    - Can use larger L2 cache to reduce miss rate without extending hit time
- Two-level 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 increases
  - 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 (e.g., address conflict and next slide)

Understanding Program Performance (참고자료)

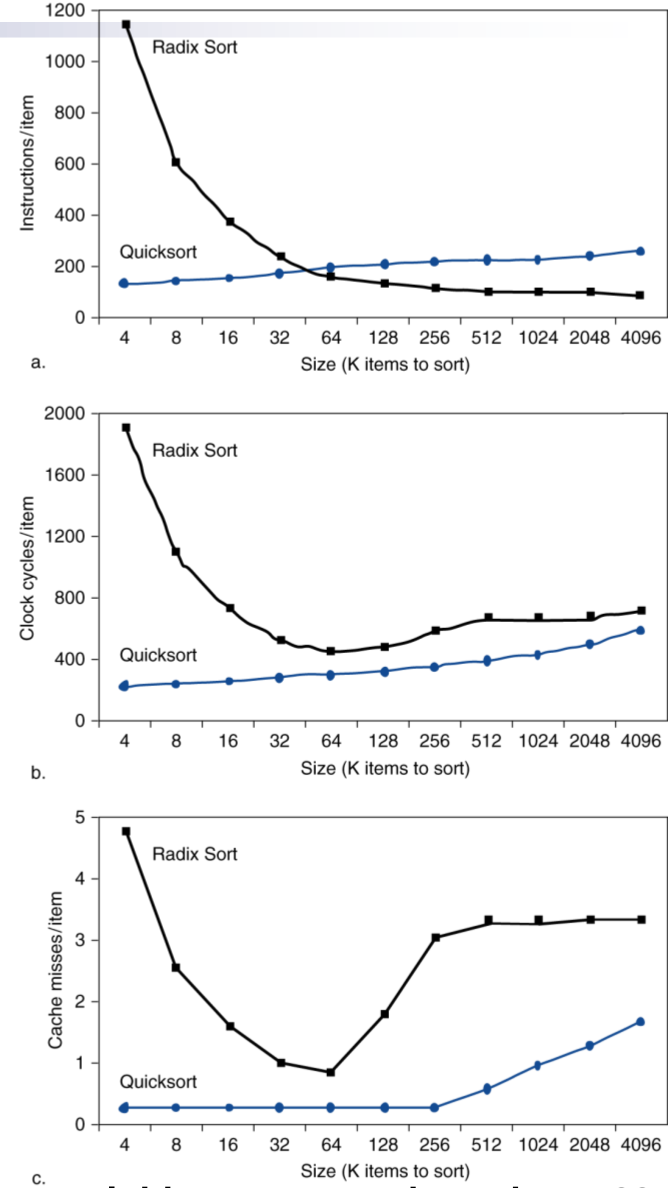
Interactions with Advanced CPUs

- 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 analyze
  - Use system simulation

Interactions with Software

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access
- Using memory hierarchy well is critical to high performance

Locality in data access!



Software Optimization via Blocking

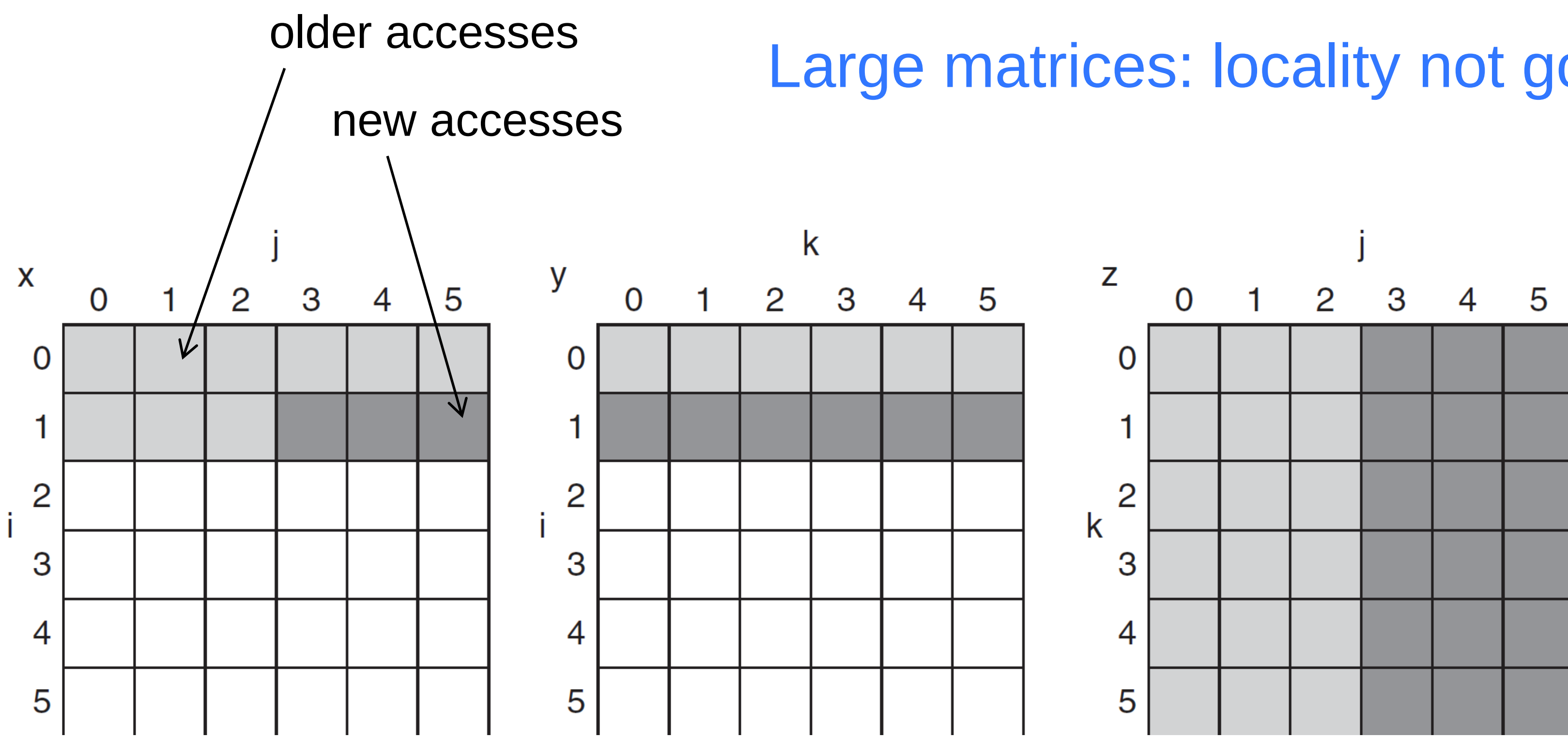
- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

```
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;
}
```

Double precision general matrix multiply

DGEMM Access Pattern

- C, A, and B arrays

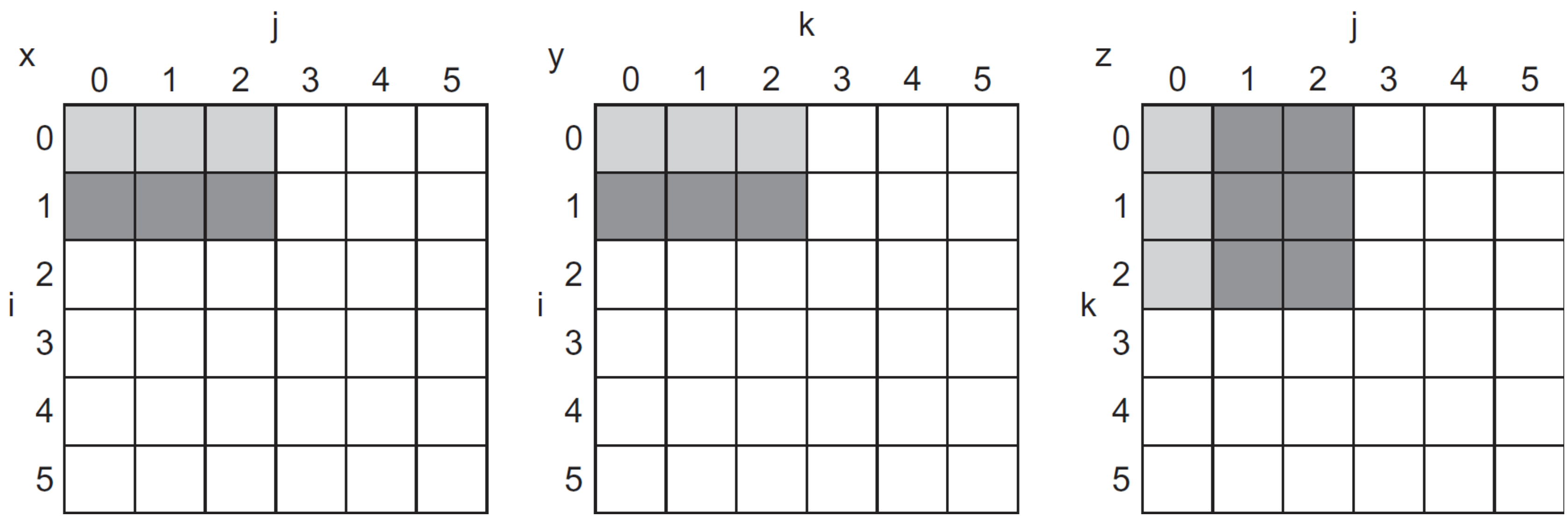


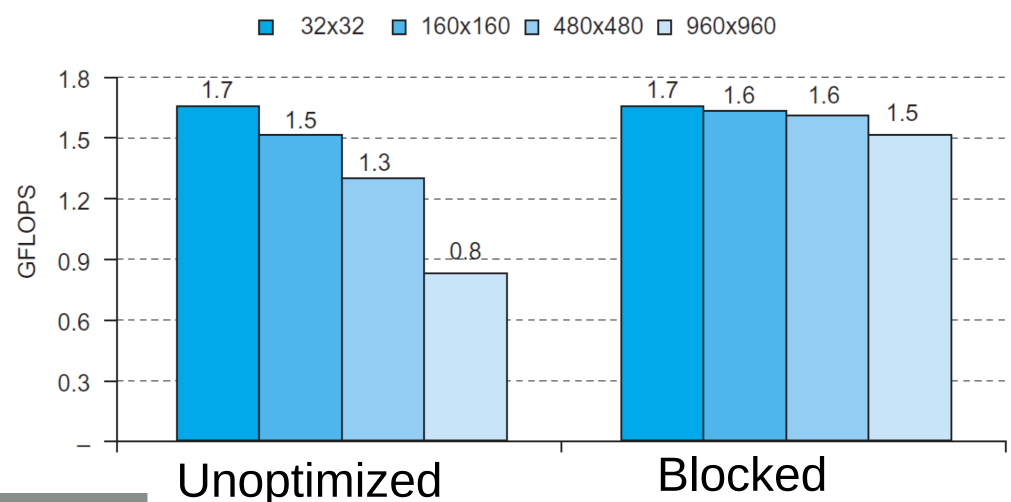
Large matrices: locality not good

Cache Blocked DGEMM

```
#define BLOCKSIZE 32
void do_block (int n, int si, int sj, int sk, double *A, double
*B, double *C)
{
 for (int i = si; i < si+BLOCKSIZE; ++i)
  for (int j = sj; j < sj+BLOCKSIZE; ++j)
  {
   double cij = C[i+j*n];/* cij = C[i][j] */
   for( int k = sk; k < sk+BLOCKSIZE; k++ )
    cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
   C[i+j*n] = cij;/* C[i][j] = cij */
  }
}
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int sj = 0; sj < n; sj += BLOCKSIZE )
  for ( int si = 0; si < n; si += BLOCKSIZE )
   for ( int sk = 0; sk < n; sk += BLOCKSIZE )
    do_block(n, si, sj, sk, A, B, C);
}
```

Blocked DGEMM Access Pattern





Multiply with smaller submatrices

Big Picture

- Part 3: implementation of ISA
  - 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
    1) Memory systems: physical and virtual
    2) Memory hierarchy
    3) Cache memory: structure, operation and performance
    4) Cache and virtual memory

