Chapter 5

Large and Fast: Exploiting Memory Hierarchy

Outline

- Memory Technologies
- Introduction to Caches
- **The Basics of Caches**
- Measuring and Improving Cache Performance
- Dependable Memory Hierarchy

Memory Technology

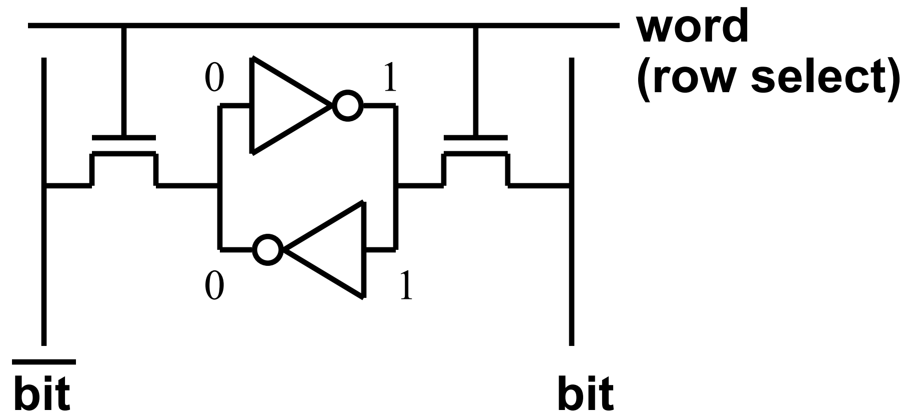
- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $1000 – $2000 per GB
- Dynamic RAM (DRAM)
  - 10ns – 40ns, $3 – $7 per GB
- Magnetic disk
  - 3ms – 12ms, $0.018 per GB (about half of 2018)
  - Data transfer rate: 200MB/s
- SSD (TLC)
  - 0.1ms – 1.5ms (write), $0.15 per GB (also about half of 2018); 8X of HDD
  - Data transfer rate: 200-2500MB/s

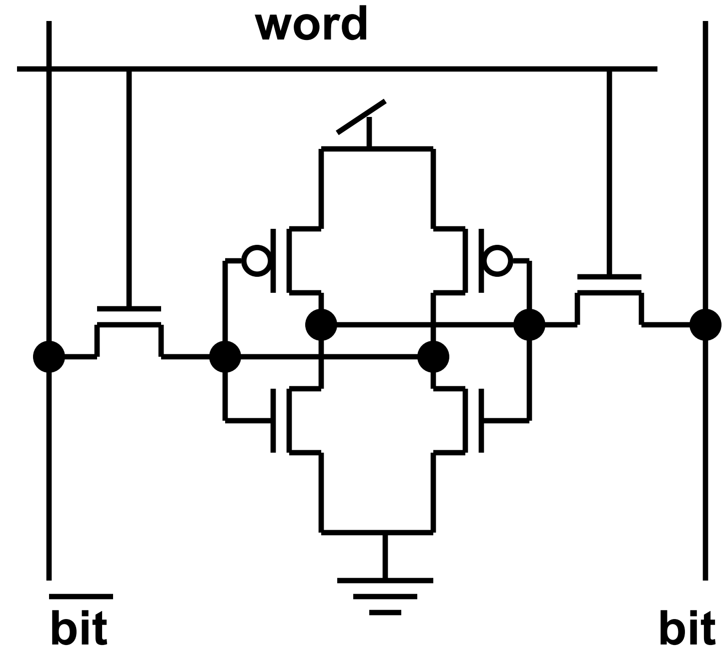
https://en.wikipedia.org/wiki/Solid-state_drive#Hard_disk_drives

https://venturebeat.com/2019/09/02/the-death-of-disk-hdds-still-have-an-important-role-to-play/

Static RAM Cell

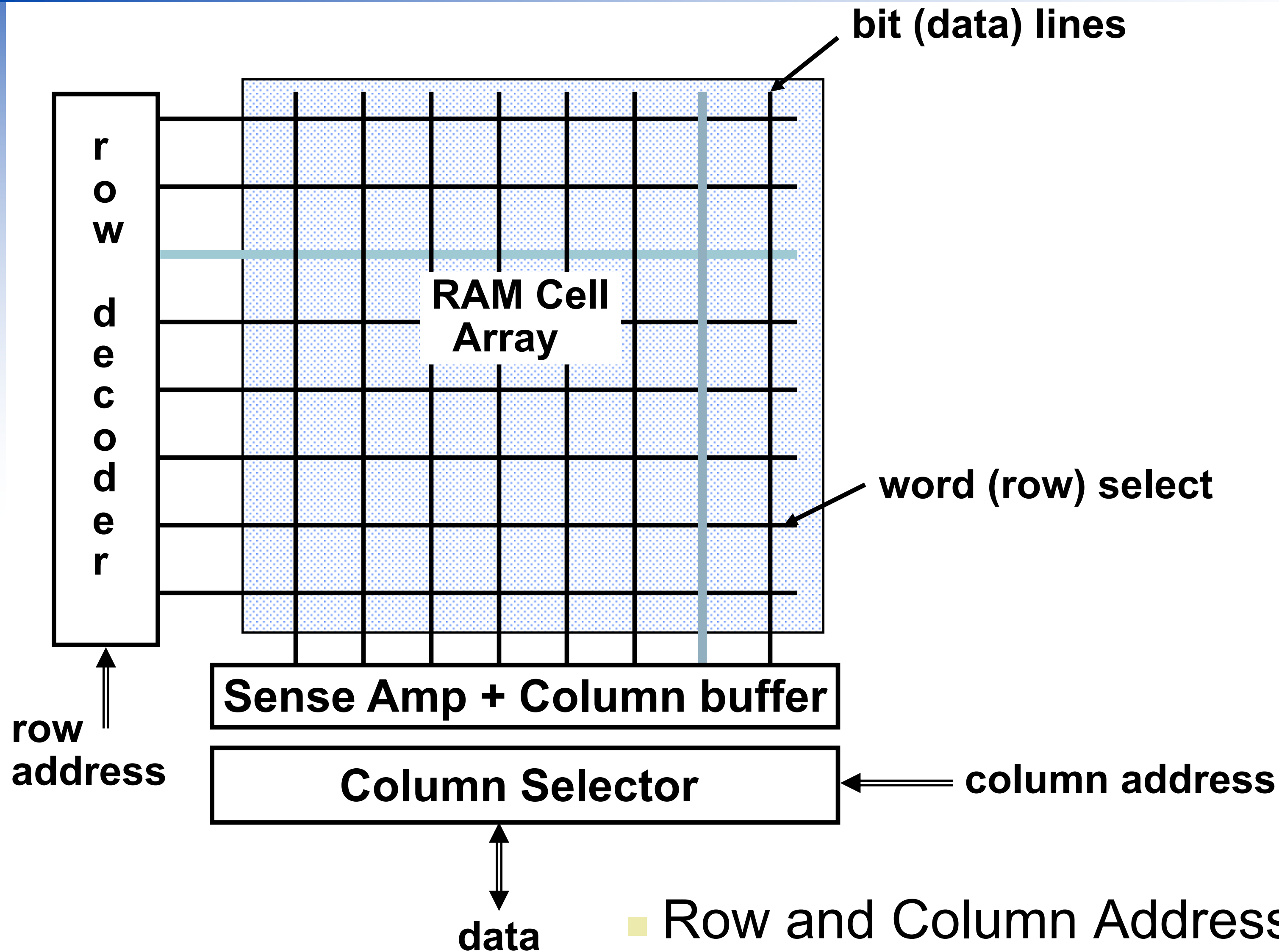
6-Transistor SRAM Cell





- Write:
  1. Drive bit lines (bit=1, bit'=0)
  2. Select row
- Read:
  1. Precharge bit and bit' to Vdd
  2. Select row
  3. Cell pulls one line low
  4. Sense amp on column detects difference between bit and bit'

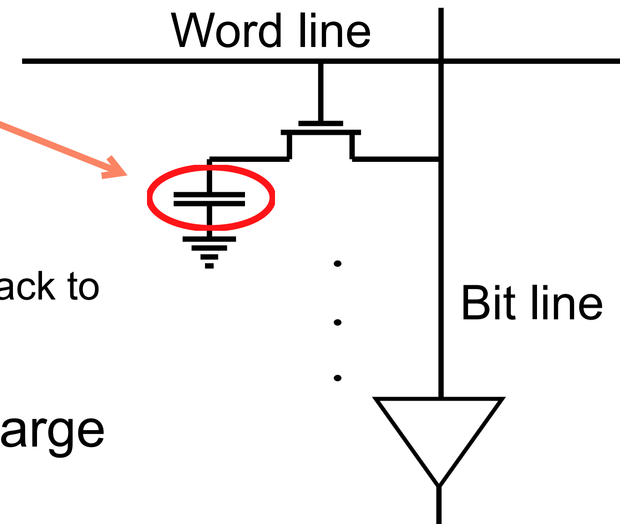
Classical DRAM Organization



- Row and Column Address together:
  - Select 1 bit a time

1-Transistor DRAM Cell

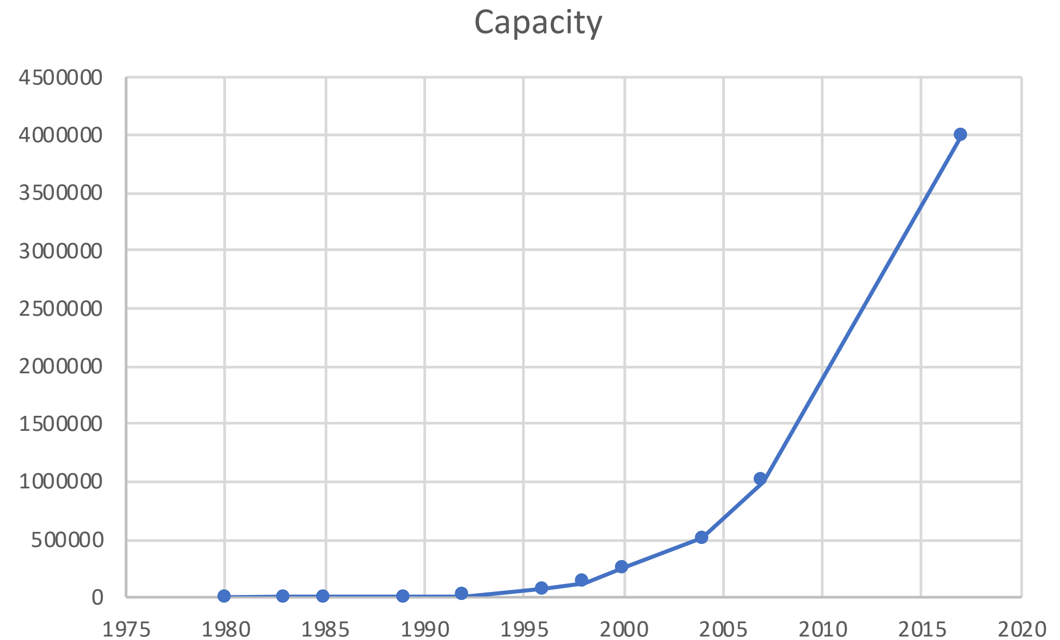
- Write
  - Charge bit line HIGH or LOW and set word line HIGH
- Read
  - Bit line is precharged to Vdd/2, and then the word line is set HIGH.
  - Depending on the charge in the cap, the precharged bitline is pulled slightly higher or lower.
  - Sense Amp Detects change
  - Read is destructive
    - So need to design circuit to write original value back to restore charge
    - Therefore read refreshes memory
- Must periodically be refreshed to prevent charge leakage
  - Read contents and write back
  - Performed on a DRAM "row"



DRAM的操作時間會比SRAM慢很多

DRAM Generations

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50
2017	4Gbit	$5



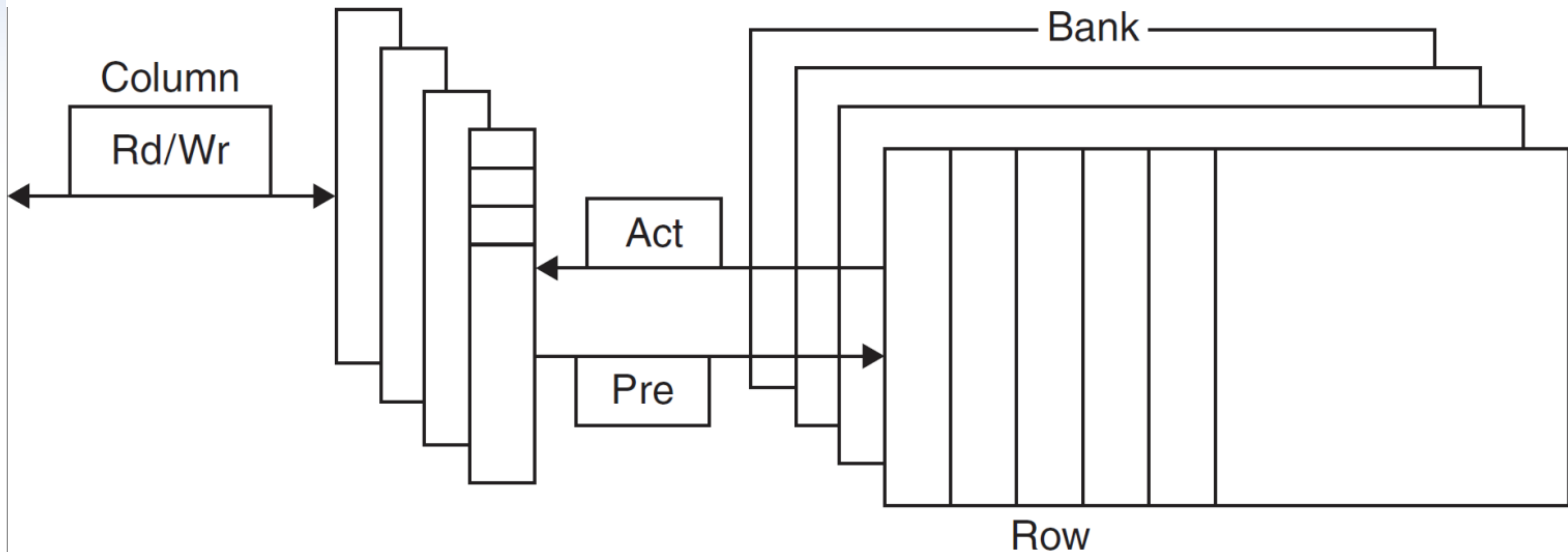
DRAM also exhibits ”moor’s law” in the past decades.

DRAM Performance Factors

- Row buffer
  - Allows several words to be read and refreshed in parallel
- Synchronous DRAM
  - Allows for consecutive accesses in **bursts** without needing to send each address
  - Improves bandwidth
- DRAM banking
  - Allows simultaneous access to multiple DRAMs
  - Improves bandwidth

DRAM Organizations

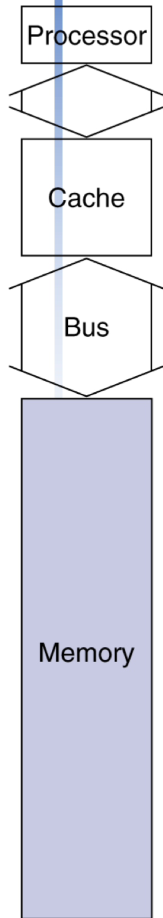
- Multiple banks to output rows of data simultaneously
  - Increase throughputs



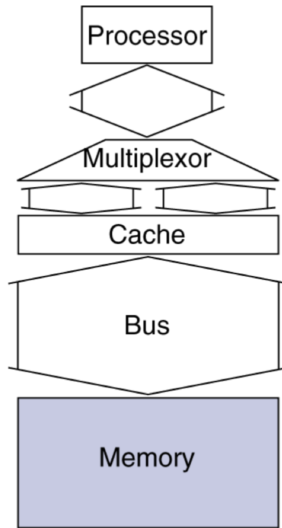
Main Memory Supporting Caches

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

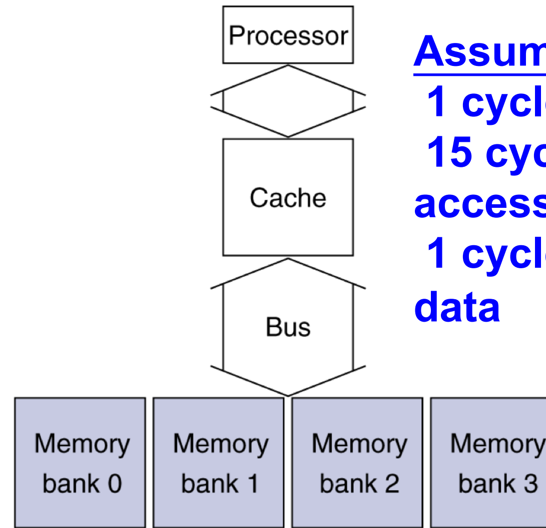
Increasing Memory Bandwidth



a. One-word-wide memory organization



b. Wider memory organization



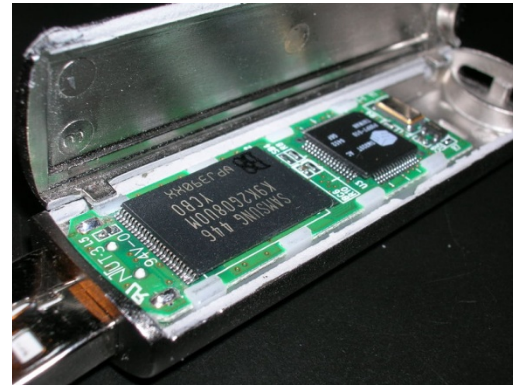
c. Interleaved memory organization

Assumptions:
- **1 cycle to send address**
- **15 cycles for each DRAM access initiated**
- **1 cycle to send a word of data**

- 1-word wide memory
  - Miss penalty = 1 + 4 x (15 + 1) = 65 cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle
- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles (BW=0.94 B/cycle)
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles (BW=0.8 B/cycle)

Flash Storage

- Nonvolatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)

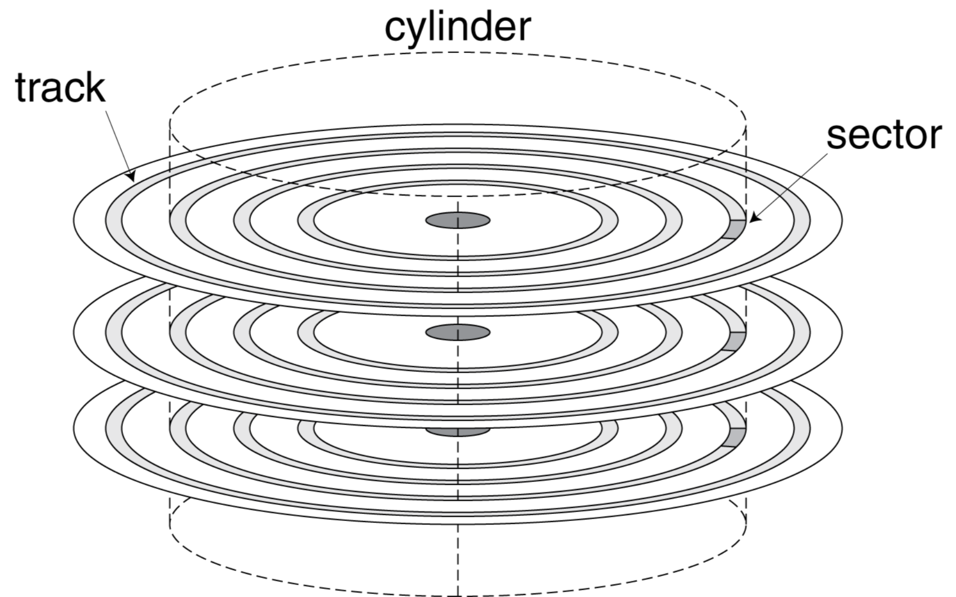


Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems
- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, …
- Flash bits wears out after 1000's of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks

Disk Storage

- Nonvolatile, rotating magnetic storage



Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
  - Synchronization fields and gaps
- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead

Disk Access Example

- Given
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average read time
  - 4ms seek time
    + ½ / (15,000/60) = 2ms rotational latency
    + 512 / 100MB/s = 0.005ms transfer time
    + 0.2ms controller delay
    = 6.2ms
- If actual average seek time is 1ms
  - Average read time = 3.2ms

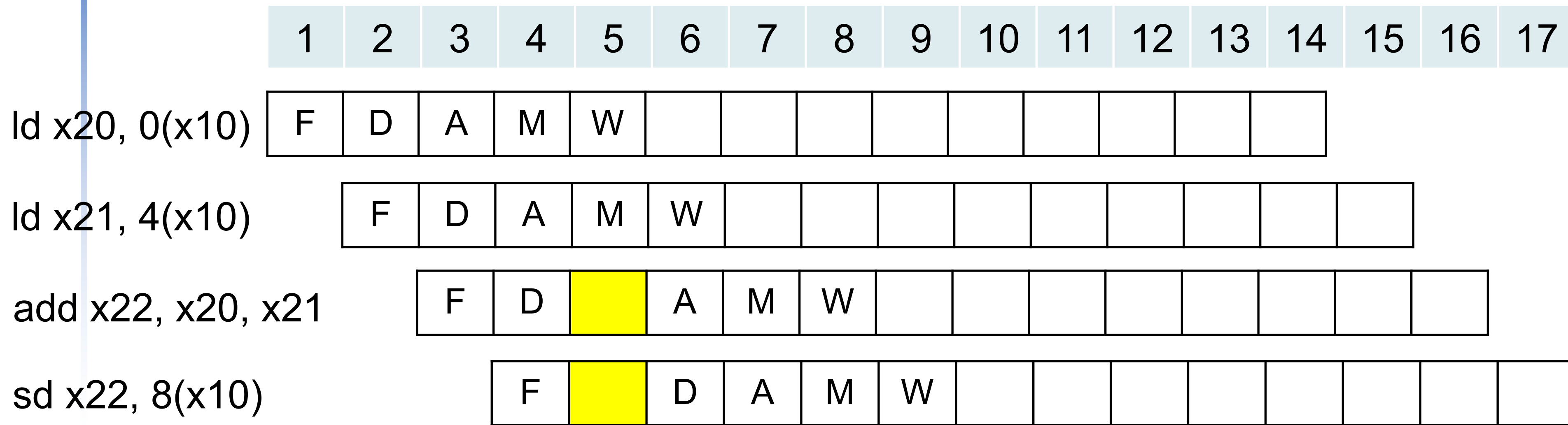
Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times
- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - SCSI, ATA, SATA
- Disk drives include caches
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay

Bridging Gap Between External DRAM and On-Chip SRAM

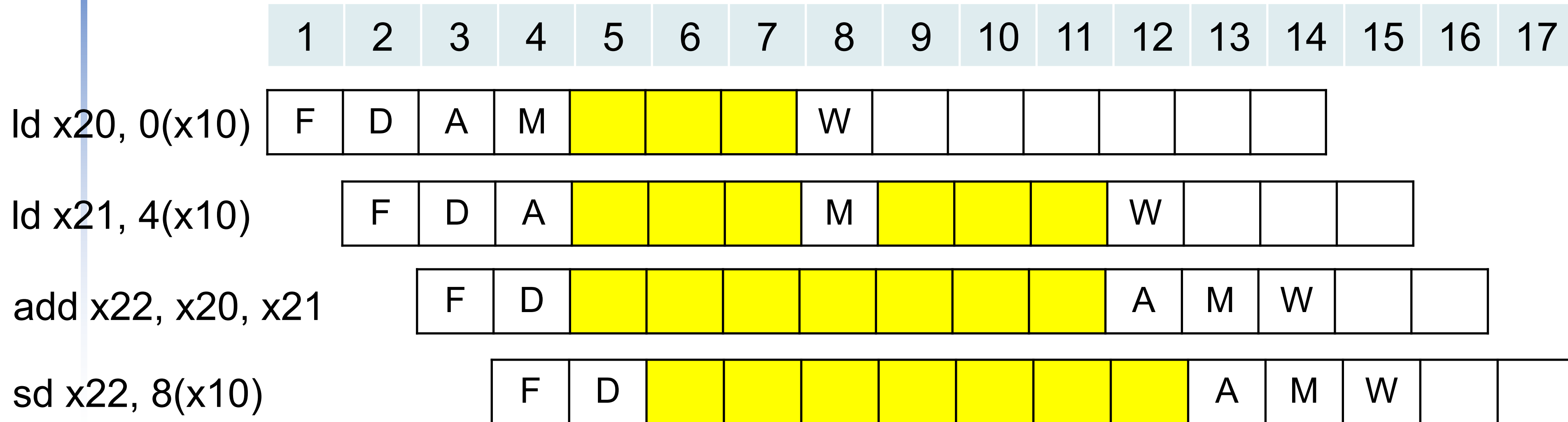
- Read access to DRAM can take 50-200 CPU cycles
- Access to SRAM is typically 1-2 cycles
- The impact to CPI is huge if we directly access instructions or data from DRAM
  - As shown in the following example
- Solution
  - Data locality

Example 1-Cycle Memory



- Total 9 cycles
- CPI=9/4=2.25
- Ignore instruction fetch from memory

Example 4-Cycle Memory



- Total 15 cycles
- CPI=15/4=3.75
- Assume memory write buffer

Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality

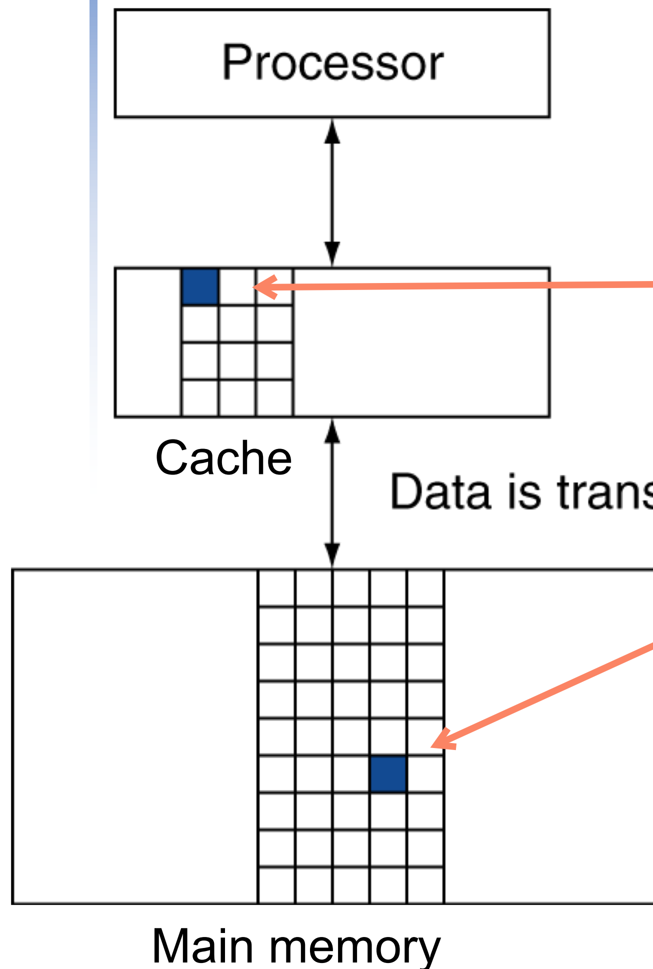
Array或是影像資料

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

Taking Advantage of Locality

- Apply memory hierarchy
  - All data are available on disk
  - Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Copy more recently accessed (and nearby) items from DRAM to smaller SRAM Cache memory

Memory Hierarchy Levels



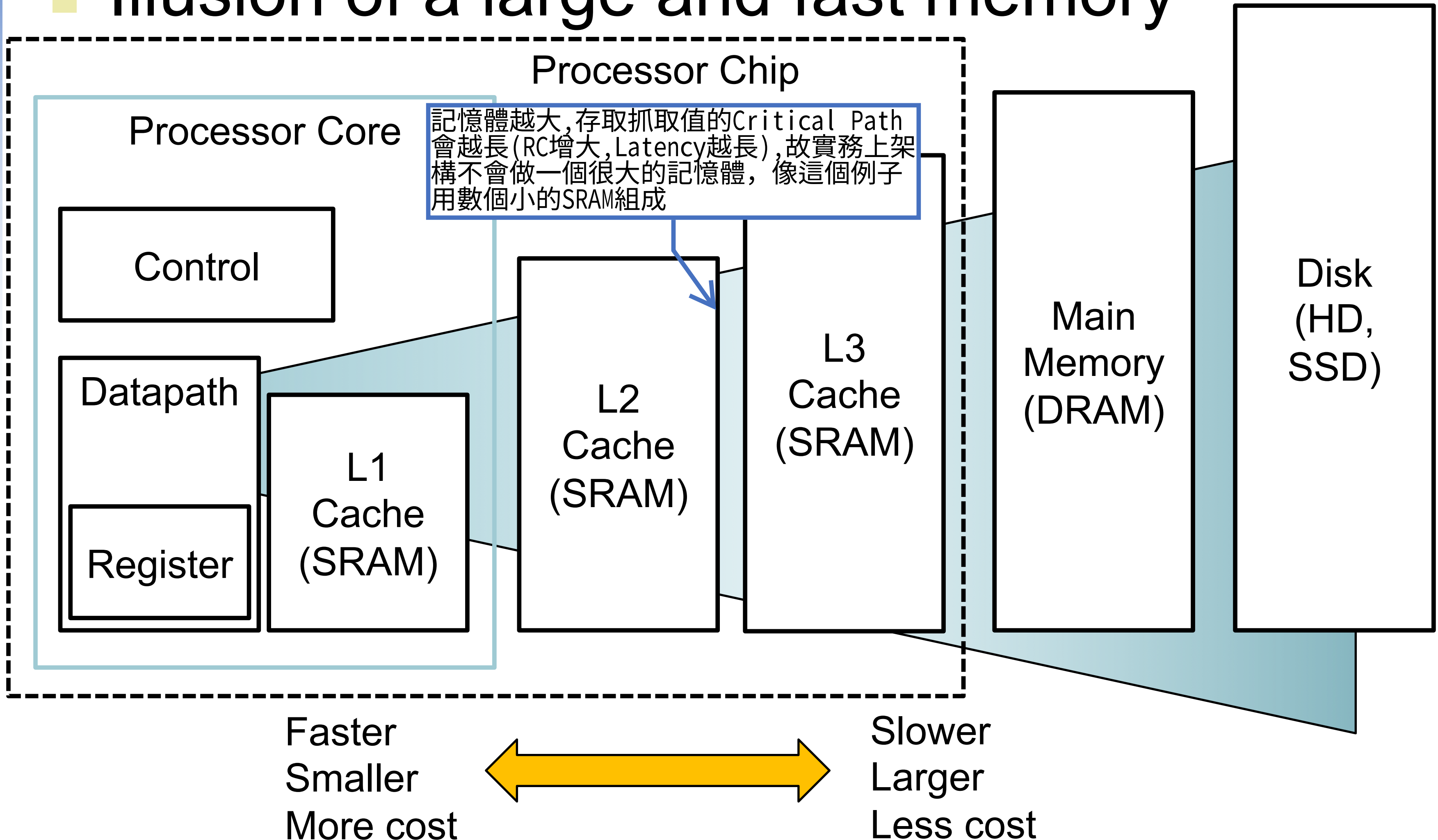
- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - **Hit**: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - **Miss**: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: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

直接能從Cache得到

必須從記憶體讀取

Memory Hierarchy Organization

- Illusion of a large and fast memory



Cache Memory

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

a. Before the reference to X_n	b. After the reference to X_n
X_4	X_4
X_1	X_1
X_{n-2}	X_{n-2}
X_{n-1}	X_{n-1}
X_2	X_2
	X_n
X_3	X_3

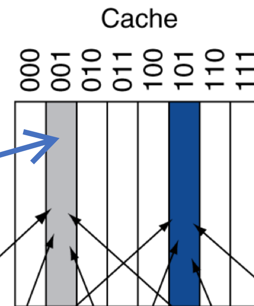
- How do we know if the data is present?
- Where do we look for data?

Direct Mapped Cache

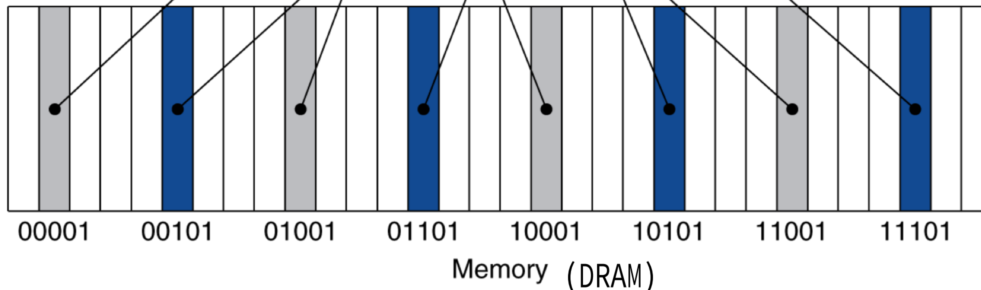
- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)

一個Cache空間會有額外的空間去紀錄是哪個Memory存進去
Ex:001->(00)001
多的兩個bits為tag

另外存一個boolean去判斷Cache裡面由沒有資料



記憶體各自對應到Cache的位置，同一個Cache會由多個Memory佔有



- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

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

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

Cache Example

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

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

Cache Example

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

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

Cache Example

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

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

Cache Example

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

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

Cache Example

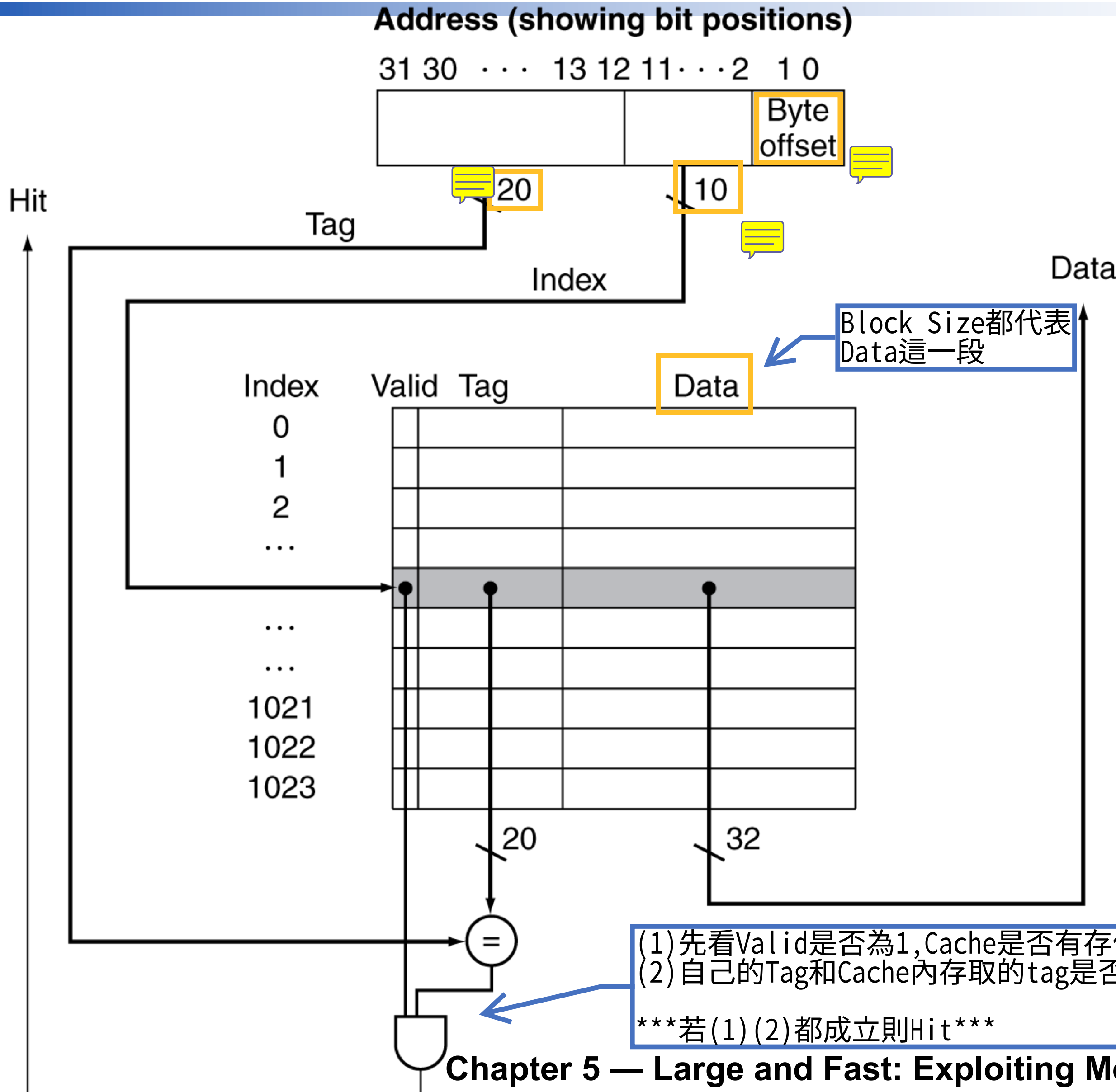
Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss **Replace**	010

需要指標去判斷是否要Replace

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

Address Subdivision

很重要！此圖
與(P44)



(1)先看Valid是否為1,Cache是否有存值
(2)自己的Tag和Cache內存取的tag是否相符合

若(1)(2)都成立則Hit

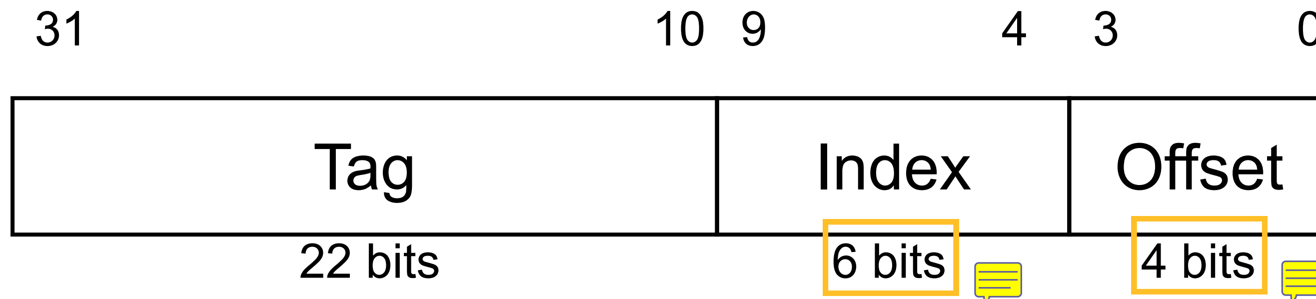
Example: Larger Block Size

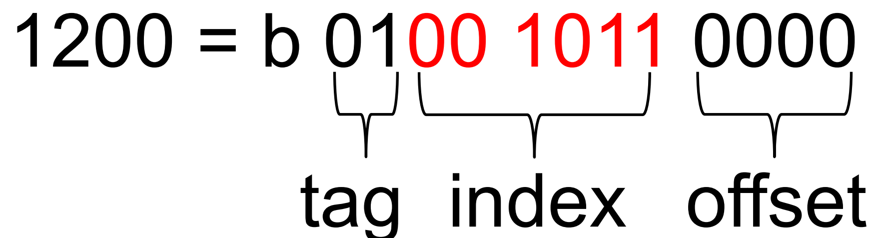
- A direct-mapped cache has 64 blocks, 16 bytes/block

Memory Address: Tag+Index+Offset

- Block number of address 1200?
- Block address = ⌊1200/16⌋ = 75
- Block number = ~~75 modulo 64~~ = 11

通常不會用Modulo去判斷Block Number,因為會增加Latency (Cache目的本是加速)





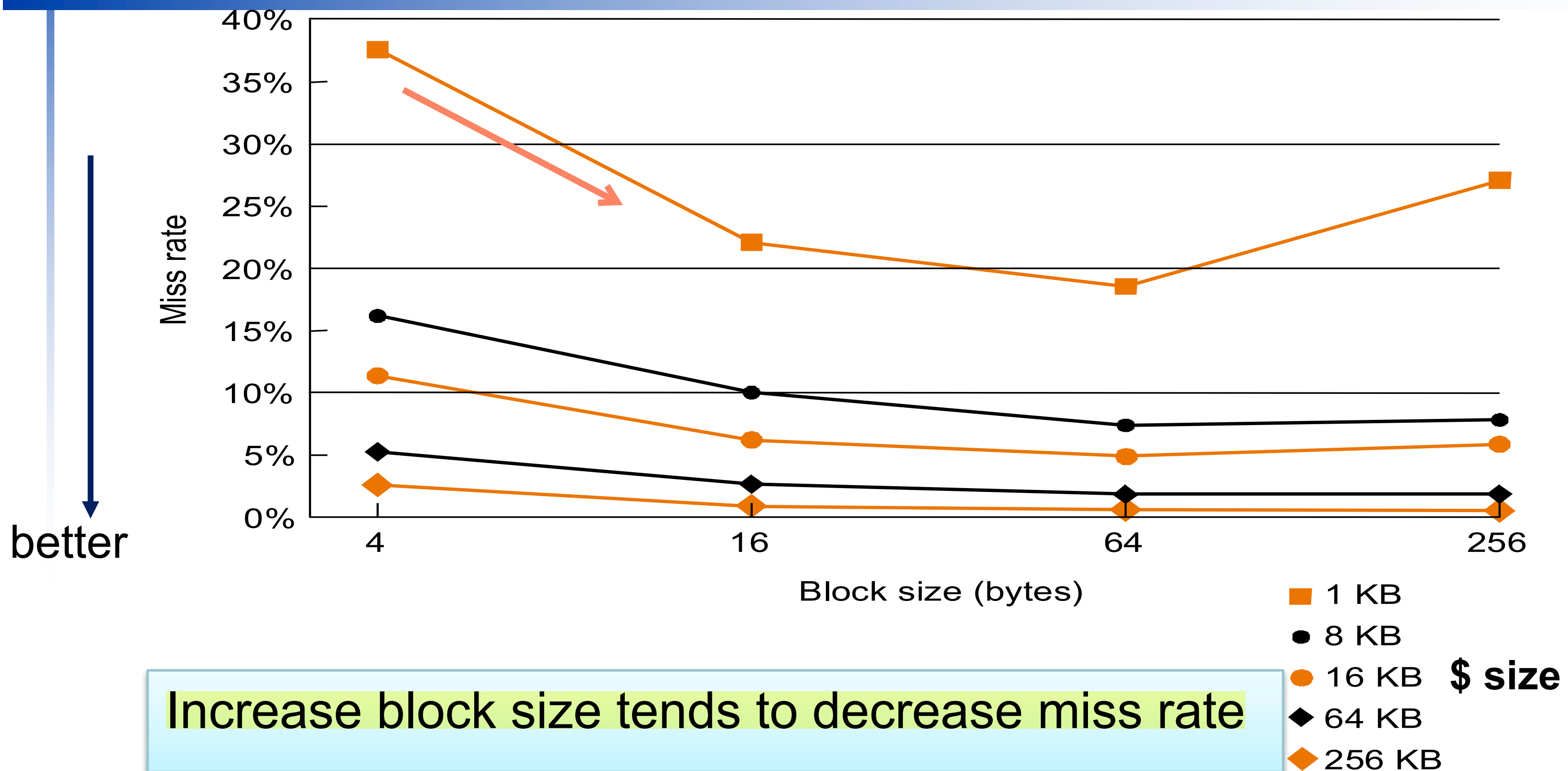
Block Size Considerations

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

> - Early restart – send arrived words in cache to CPU as early as possible
> - Critical-word-first – retrieve needed words from memory first

從記憶體讀取資料送到CPU後,再處理Cache的問題

Example of Block Size on Performance

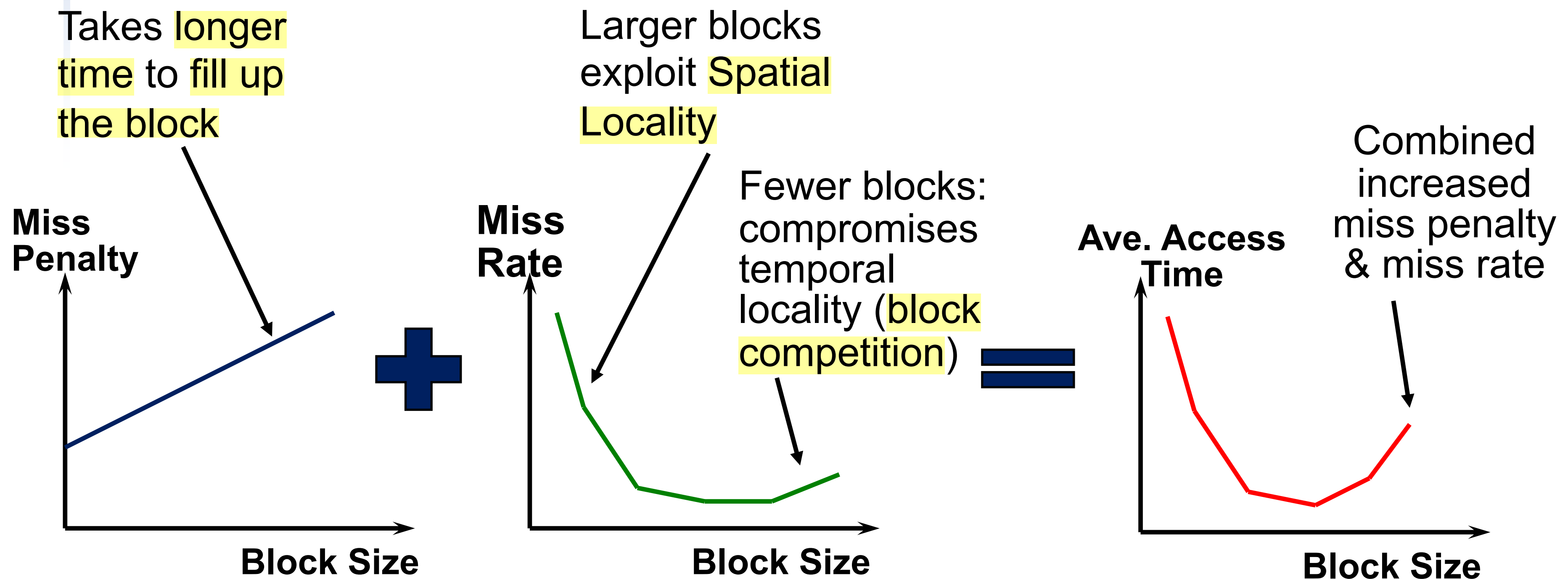


Increase block size tends to decrease miss rate

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Memory Access Time v.s. Block Size

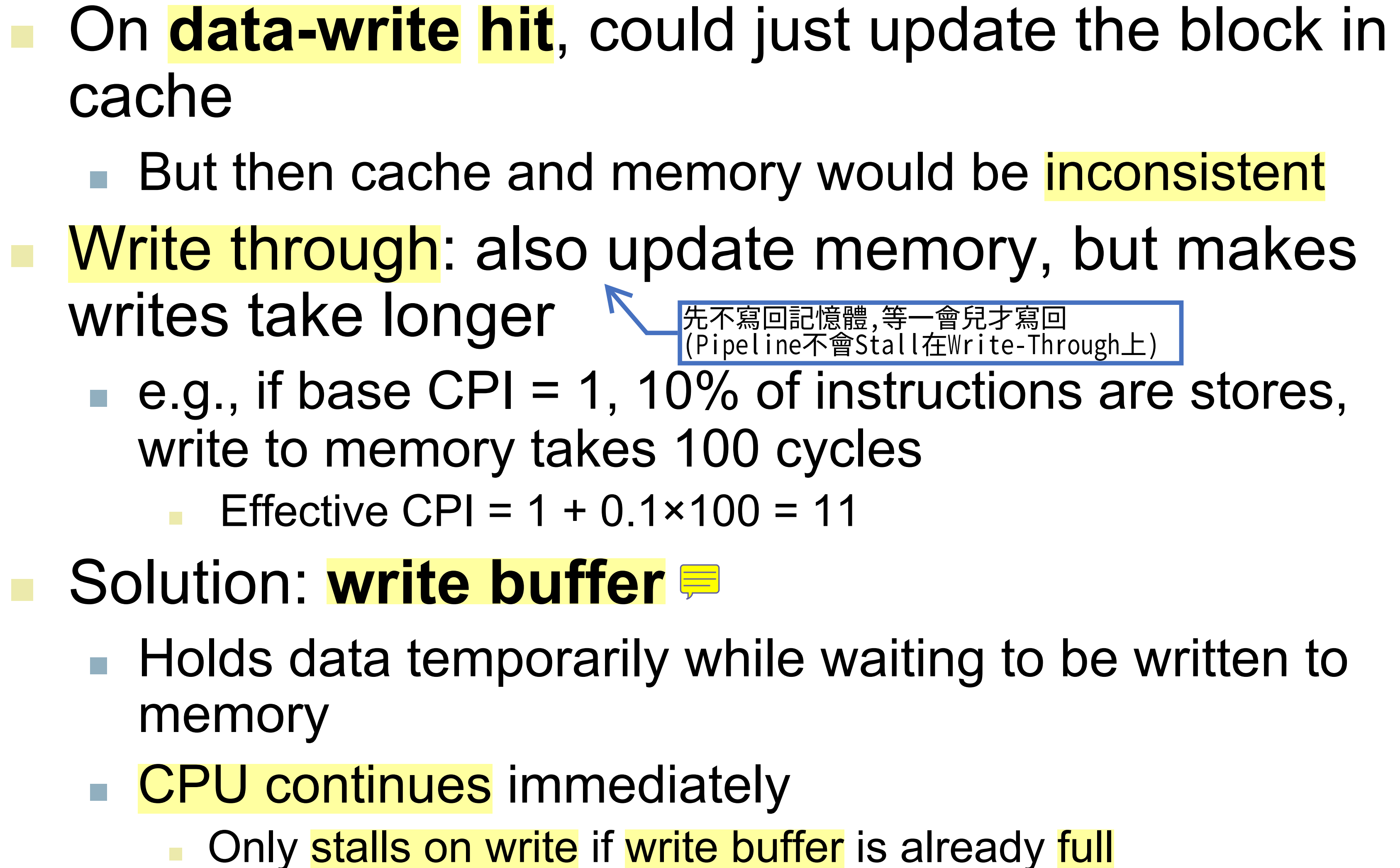
- Average Memory Access Time
  = Hit Time + Miss Penalty x Miss Rate



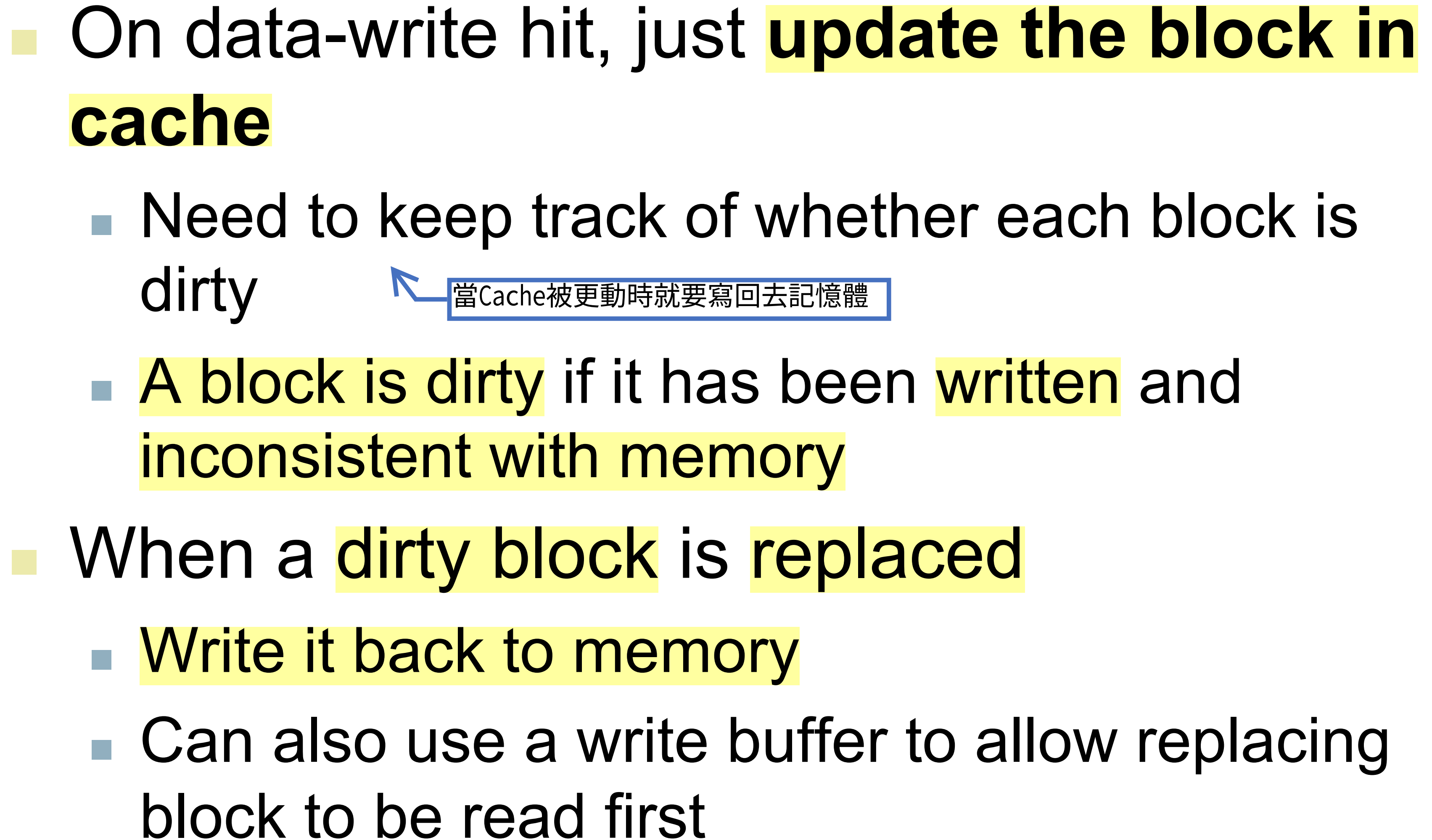
Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Write-Through



Write-Back



Write Allocation

Write Hit : Write Back
Write Miss : Write Allocation

- On a write miss
- Two methods 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)

整個Cache做更新
Ex:初始化Array

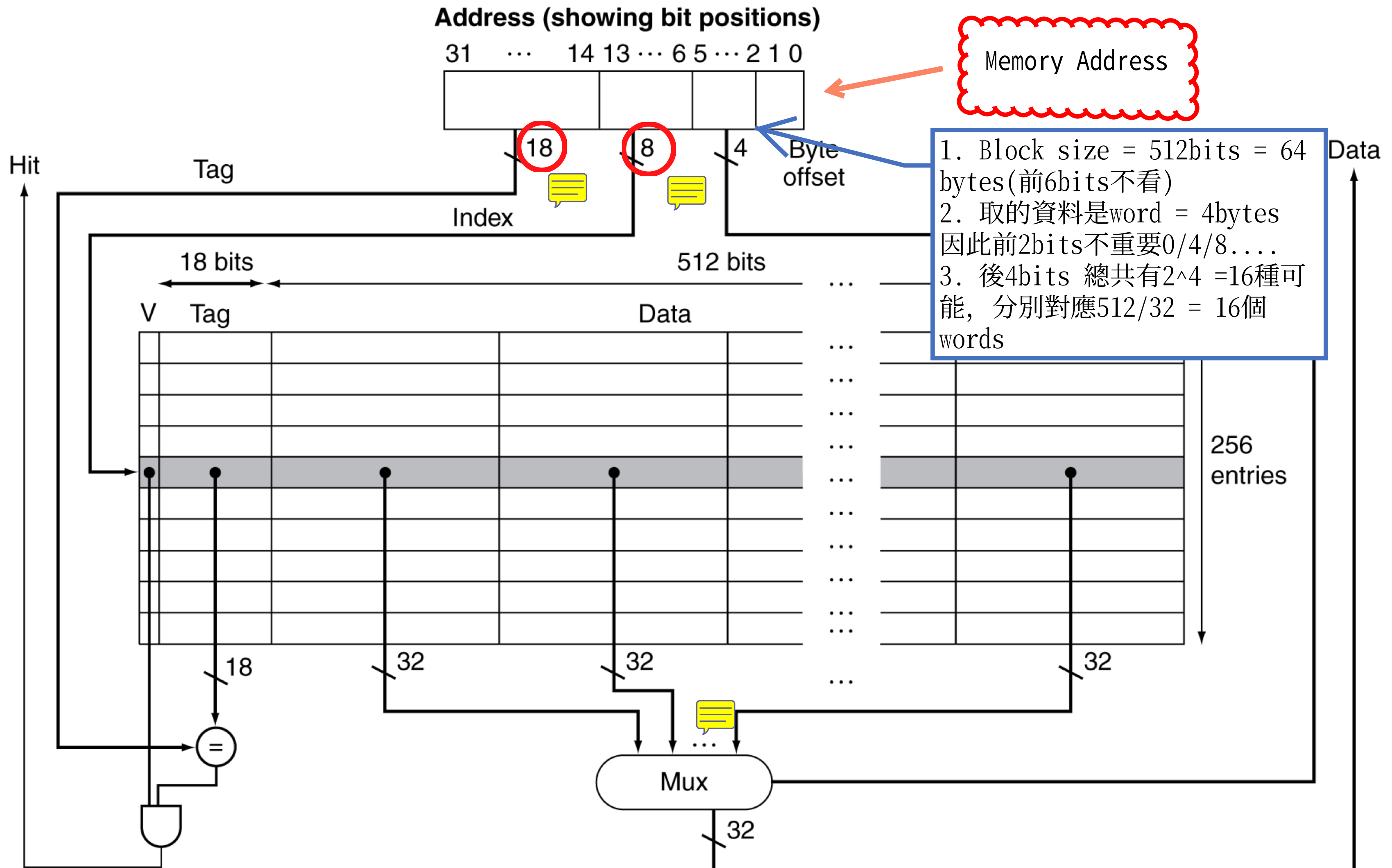
- For write-back
  - Usually fetch the block

. 當cache hit時，若CPU要寫入資料到某一位址時，可分為二種方式：一種是write through，此種方式資料會立刻寫到cache及主記憶體中；另一種是write back，此種方式會先將資料寫入cache中，然後再將同一位址的資料整批一起寫入主記憶體中（非立即寫入）。
. 當cache miss時，若CPU要寫入資料到某一位址時，可分為二種方式：一種是no write allocate，此種方式會直接將資料寫到主記憶體中，不會再從記憶體中載入到cache，另一種方式是write allocate，此種方式會先將資料從主記憶體中載入到cache，然後再依cache hit的規則，將資料寫出。

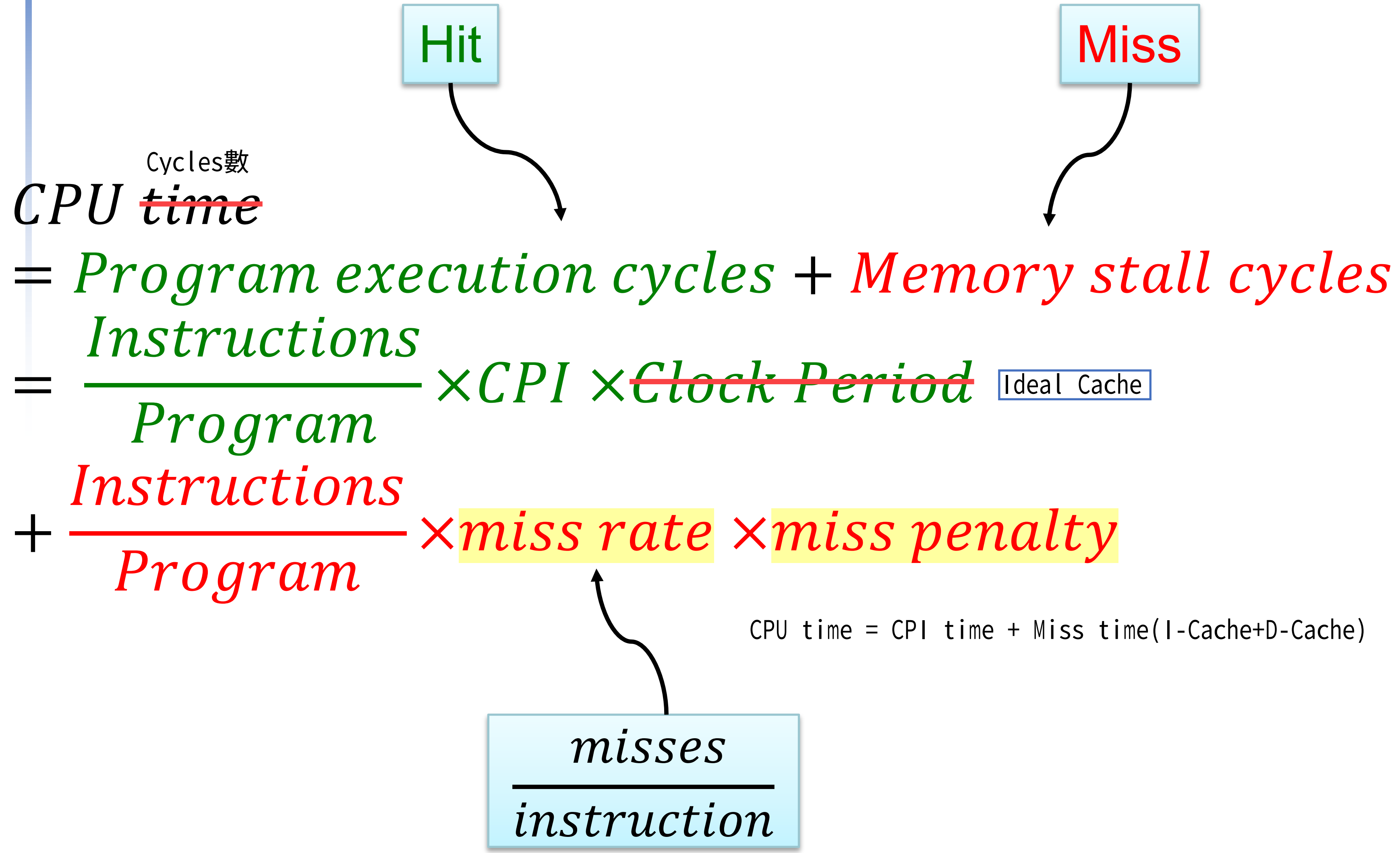
Example: Intrinsity FastMATH

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

Example: Intrinsity FastMATH

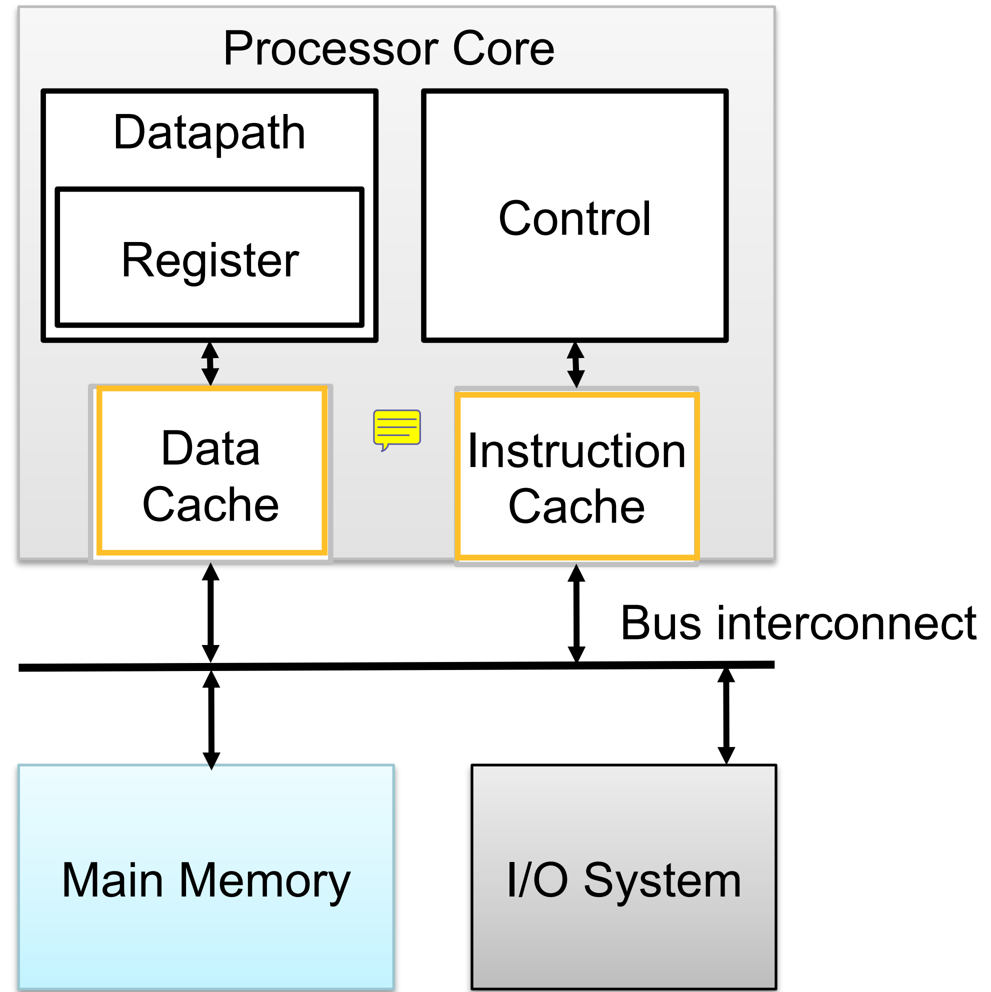


Measuring Cache Performance



Instruction and Data Cache

- Most architectures separate instruction and data caches
- Different I-Cache and D-Cache architectures to optimize performance and increase bandwidth.



I-Cache and D-Cache Performance

$$Memory\ stall\ cycles$$

$$= Instruction\ cache\ miss + data\ cache\ miss$$

$$= \frac{Instructions}{Program} \times Icache\ miss\ rate \times miss\ penalty$$

$$+ \frac{Load\&Store\ Instructions}{Program} \times Dcache\ miss\ rate \times miss\ penalty$$

讀取Instruction的Miss

Memory存取位置是共有的

讀取Data的Miss

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Average Access Time

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

Performance Summary

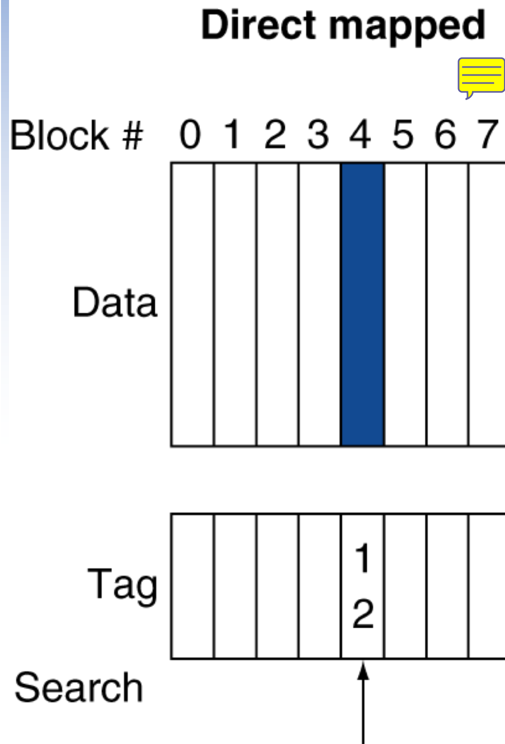
- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can’t neglect cache behavior when evaluating system performance

Associative Caches

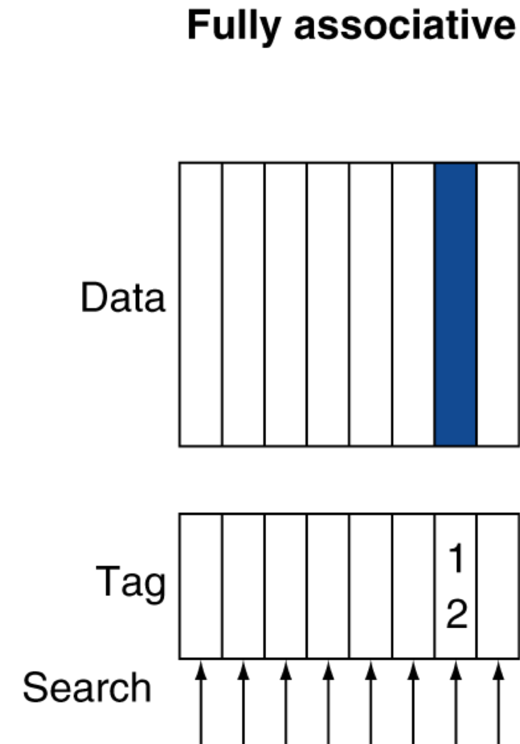
no index，offset剩下的全部拿來做tag

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Different Cache Architectures



必須比對兩個Tag
其中一個



Spectrum of Associativity

- For a cache with 8 entries

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

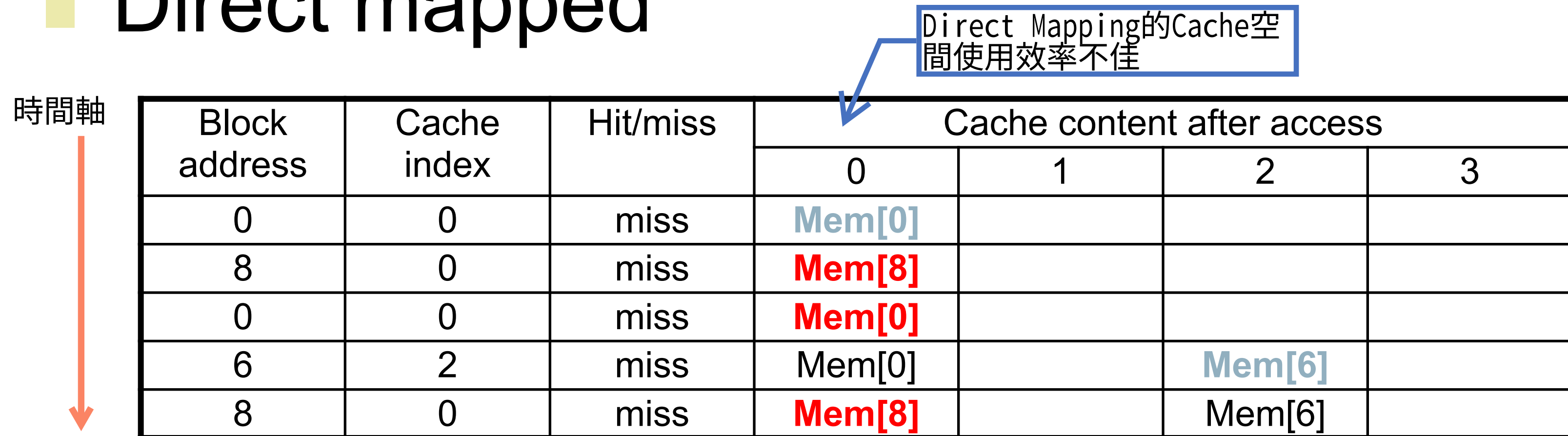
Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Associativity Example

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



Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example

- 2-way set associative

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

- Fully associative

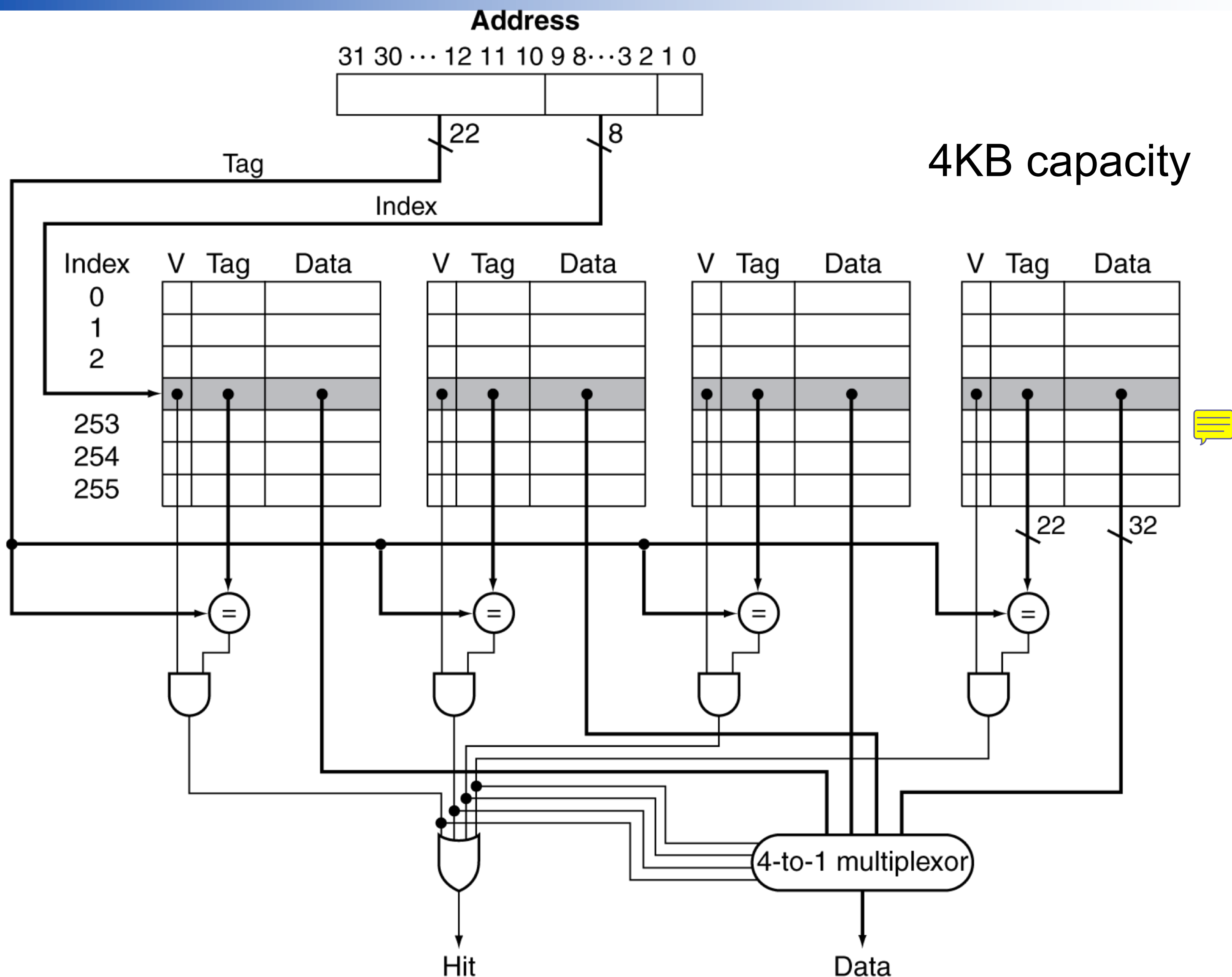
看出Hit比例較高

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

How Much Associativity

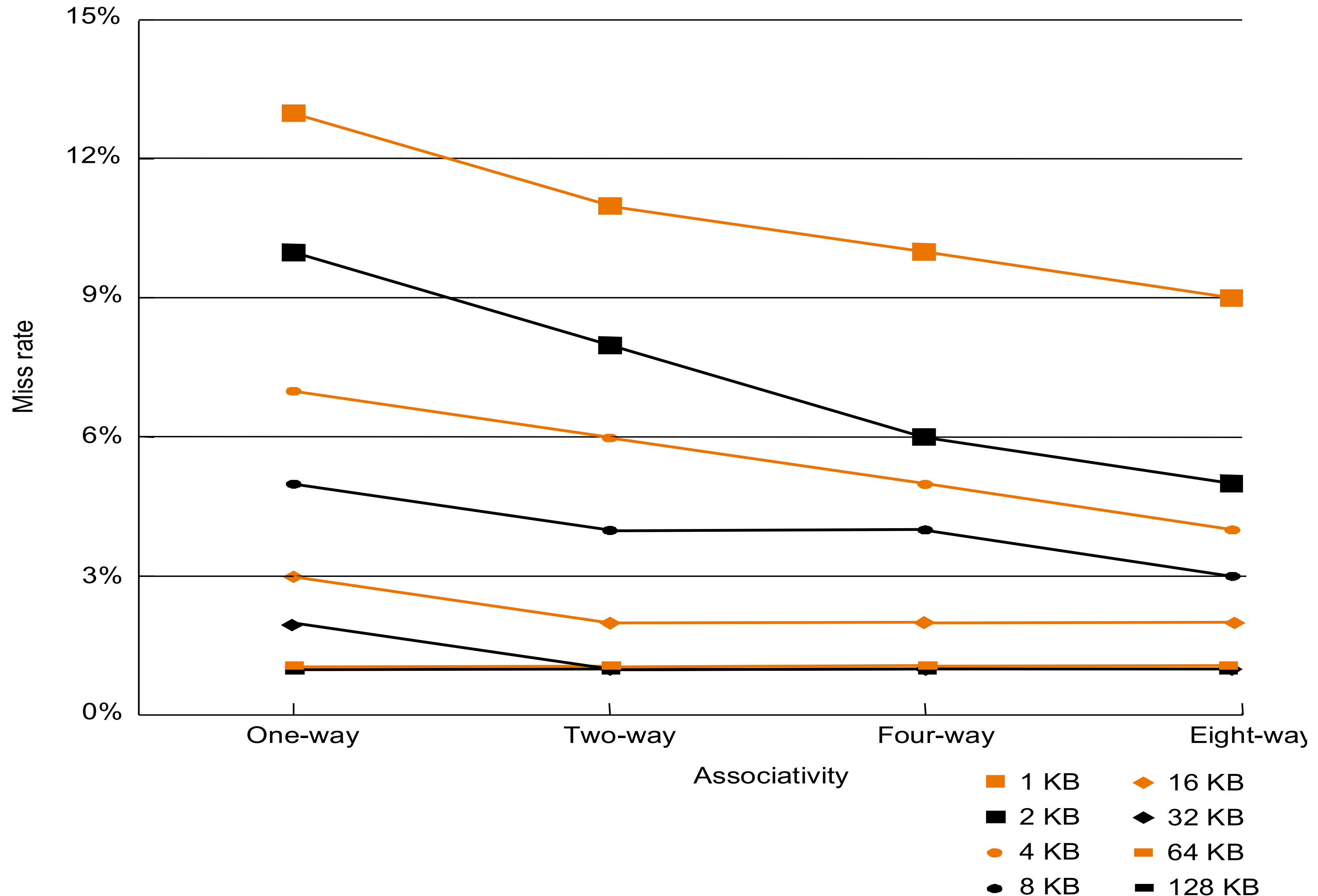
- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

4-way Set Associative Cache Organization



4KB capacity

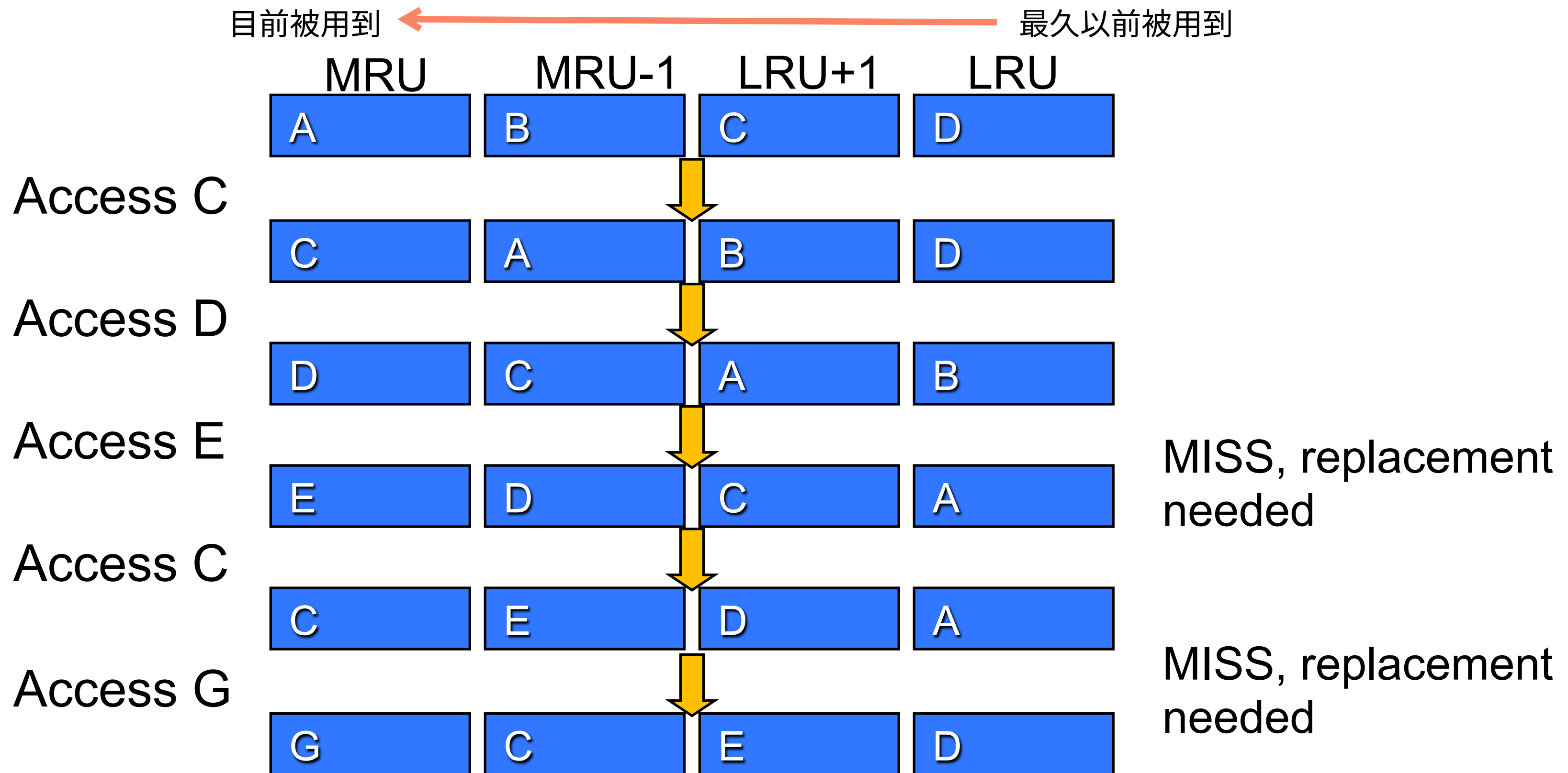
Example of Multi-Way Caches



Replacement Policy for Associative Caches

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, a single bit to set if one block is accessed, unset if the other is accessed
    - Manageable for 4-way, too hard beyond that
- Random (randomly pick any block)
  - Gives approximately the same performance as LRU for high associativity

LRU Example

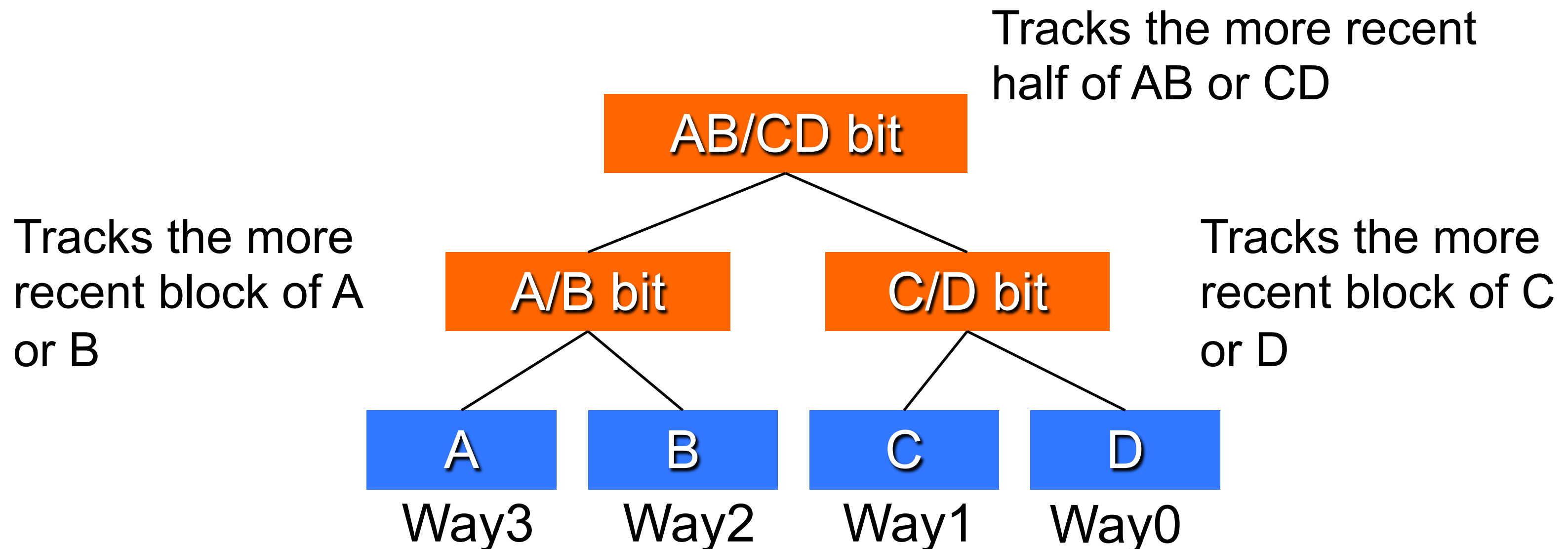


Pseudo LRU

- Most caches use approximate LRU
  - A popular approach uses S-1 bits for an S-way cache
    - The blocks are hierarchically divided into a binary tree
    - At each level of the tree, one bit is used to track the least recently used
  - For a 4-way set associate cache, blocks are first divided into two halves, each half has two blocks
    - The 1st bit tracks the more recently used half
    - The 2nd bit (3rd) tracks the more recently block in the first (second) half
    - The one to replace is the less recently used block in the less recently used half
- Used in many commercial processors

Example Pseudo LRU

- Tree-based
  - O(N): 3 bits for 4-way
  - Cache ways are the leaves of the tree
  - Combine ways as we proceed towards the root of the tree



Comparing Random and LRU Policy

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

Sources of Misses

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

Cache Design Trade-offs

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

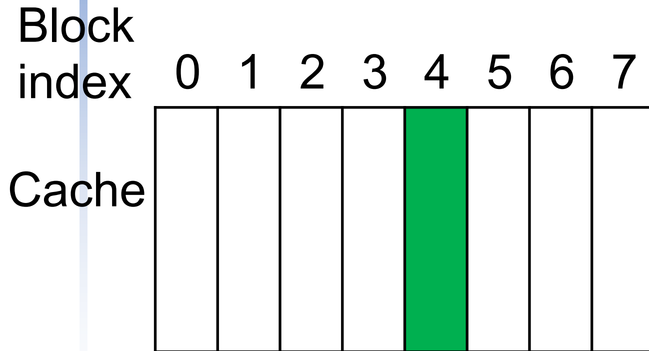
More Cache Examples

- Block placement
- Block identification and address formats

Block Placement

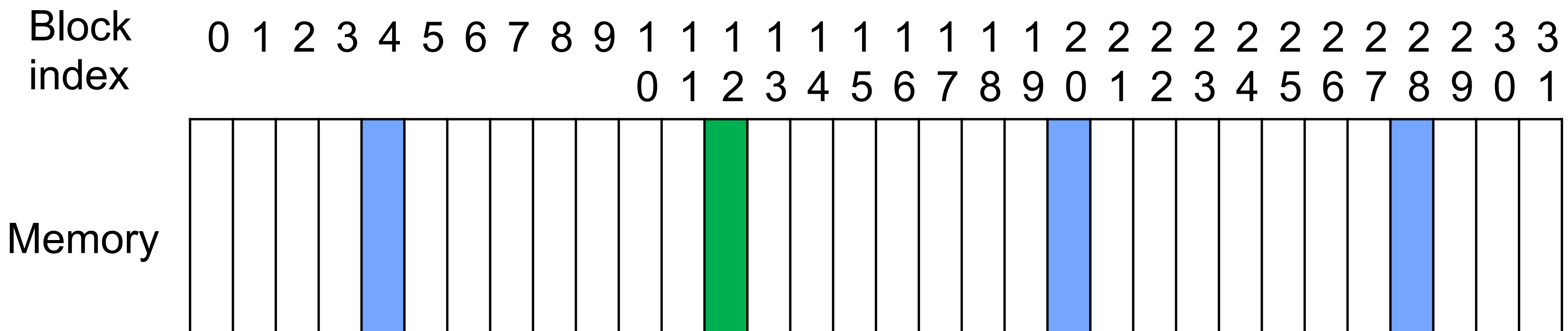
- Which location to put a block?
- Direct-mapped
  - location= (block addr) mod (# block in cache)
- Fully-associative
  - Any place in cache
  - Search complete cache for exact block location
- Set-associative
  - location= (block addr) mod (# set in cache)
  - Search # way for exact block location in a set

Block Placement Example (Direct-mapped)

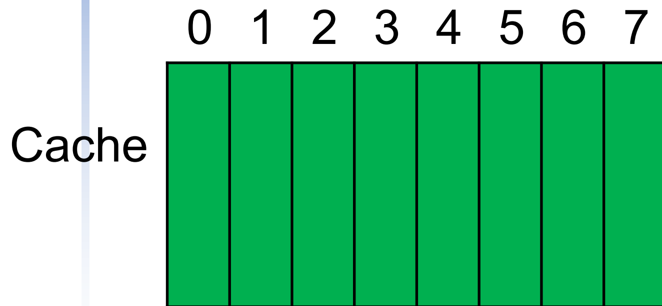


block location of block #12
= 12 mod 8 = 4

Also block #4, #20, #28 will be placed at the same cache location
(to replace #12).

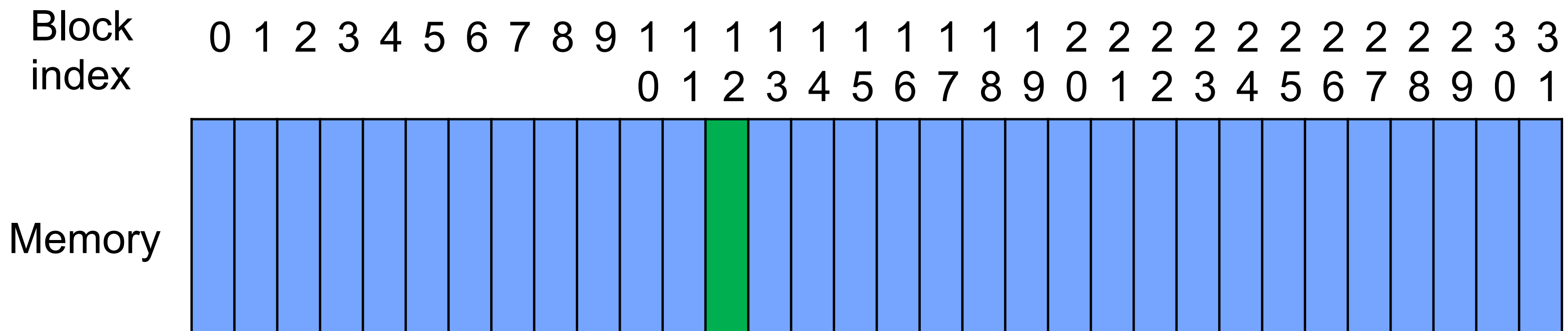


Block Placement Example (Fully-associative)

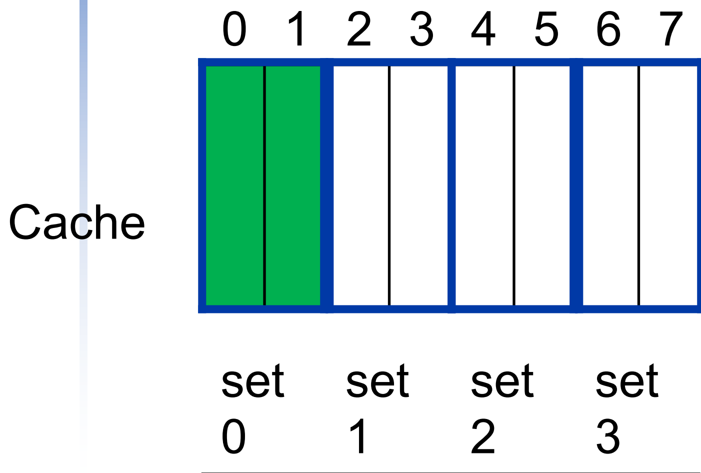


block location of block #12
= any where in the cache

Also other 31 blocks can be placed at the same cache location.



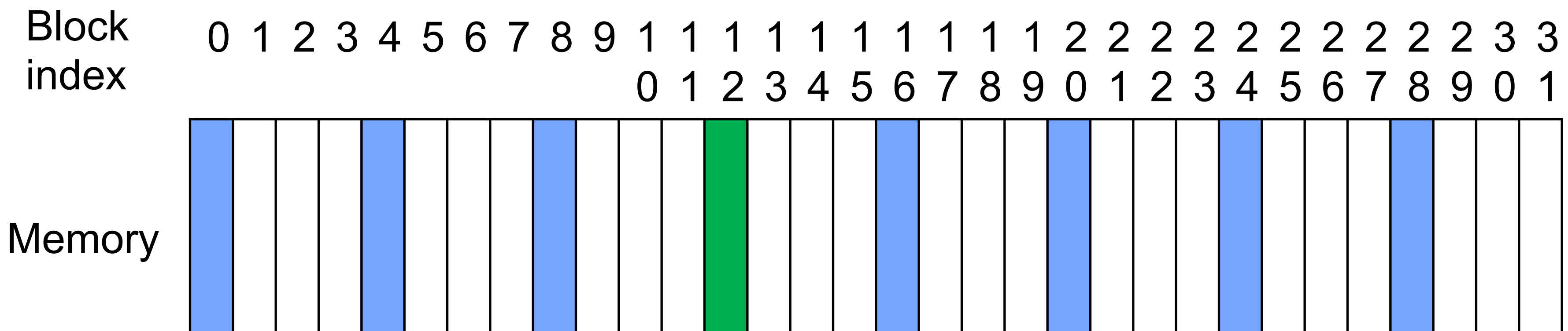
Block Placement Example (Set-associative)



block location of block #12
= 12 mod 4 = 0

Block #12 can go into either block in set 0.

Also block #0, #4, #8, #16, #20, #24, #28 will be placed at the same set (to replace #12).



Block Identification (Direct-mapped)

- Find where we put a block in cache?
- Direct-mapped address format

r = address tag	m = block index	n = byte offset

  - address = r + m + n
  - 2^n bytes in a block
  - 2^m blocks in a cache
  - cache size = $2^m * 2^n$
  - index = (block addr) mod (2^m)
  - Check valid bit at index location
  - Compare r address tag to confirm match

Block Identification (Fully-associative)

- Find where we put a block in cache?
- Fully-associative address format

  - address = r + n
  - 2^n bytes in a block
  - Compare r address tag to all cache blocks to confirm match

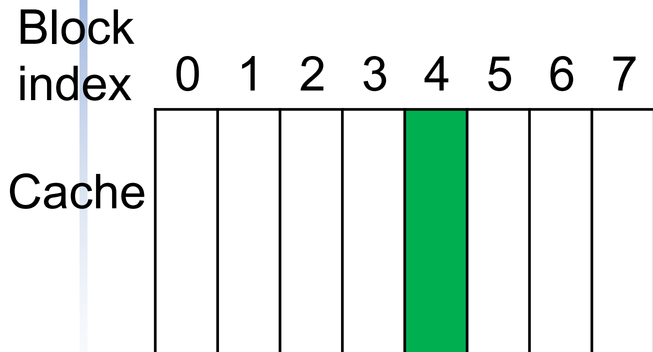
Block Identification (Set-associative)

- Find where we put a block in cache?
- Set-associative address format

r = address tag	m = set index	n = byte offset

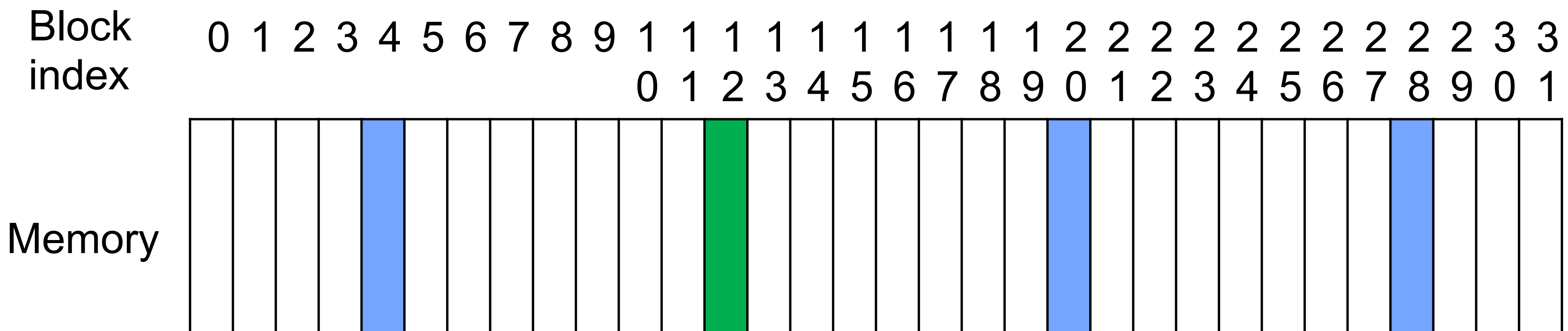
  - address = r + m + n
  - 2^n bytes in a block
  - 2^m sets in a cache
  - cache size = 2^m * 2^n * # way
  - set index = (block addr) mod (2^m)
  - Compare r address tag of all ways to confirm match
  - Check valid bit at set location

Direct-mapped Address Example

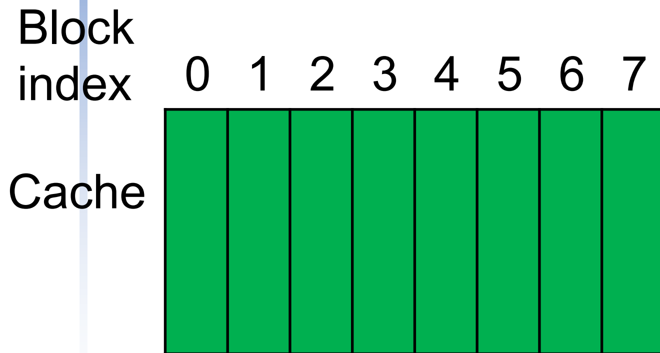


block size = 2 bits (2^2=4bytes)
total memory = 5+2 bits (2^7=128bytes)
cache block size = 3 bits (2^3=8blocks)
tag = 2 bits (5-3=2)

block #12= 01100
block index = 100 mod 8 = 100
tag = 01

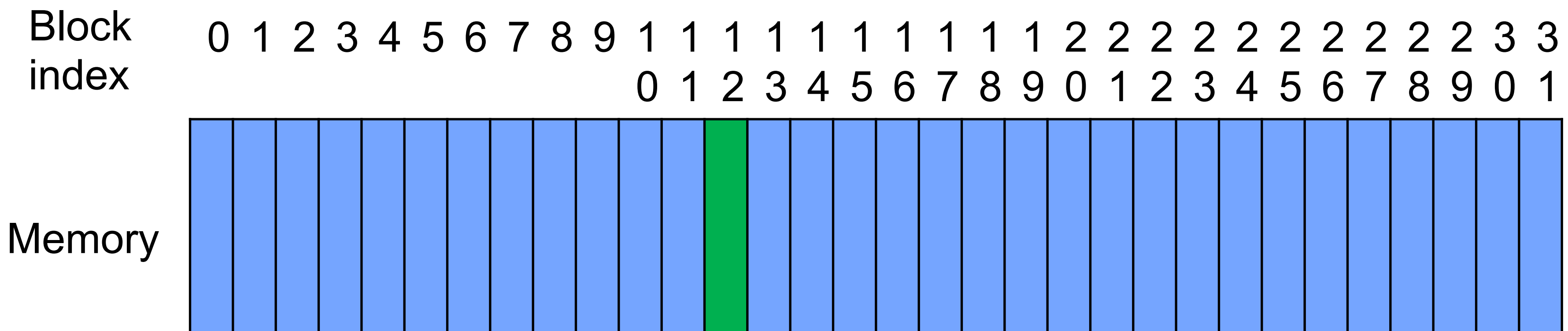


Fully-associative Address Example

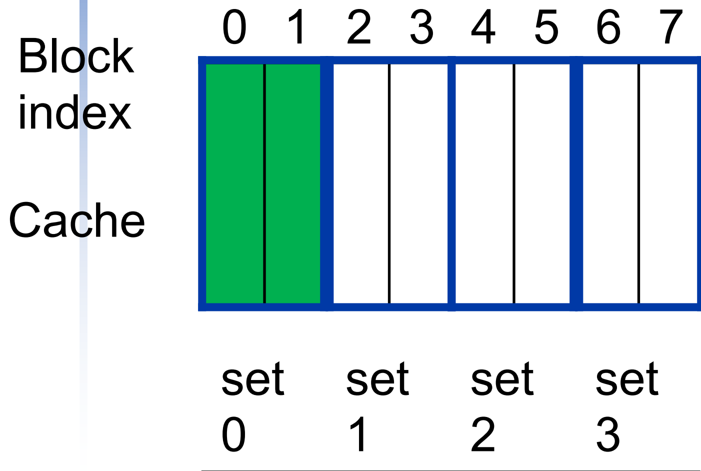


block size = 2 bits (2^2=4bytes)
total memory = 5+2 bits (2^7=128bytes)
cache size = 3 bits (2^3=8blocks)
tag = 5 bits

block #12= 01100
tag = 01100



2-way Set-associative Address Example



block size = 2 bits (2^2=4bytes)

total memory = 5+2 bits (2^7=128bytes)

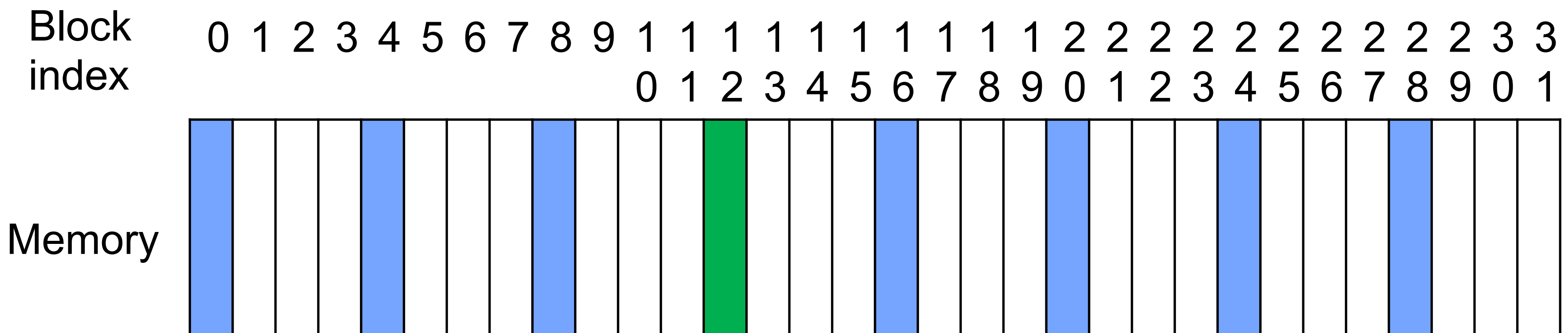
cache set size = 2 bits (2^2=4sets)

tag = ~~2~~ 3 bits (5-3=~~2~~)

block #12= 01100

set index = 100 mod 4 = 00

tag = 011



Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

Level 2 Cache的Access time 較高，但是Miss Rate有效降低

Multilevel Cache Considerations

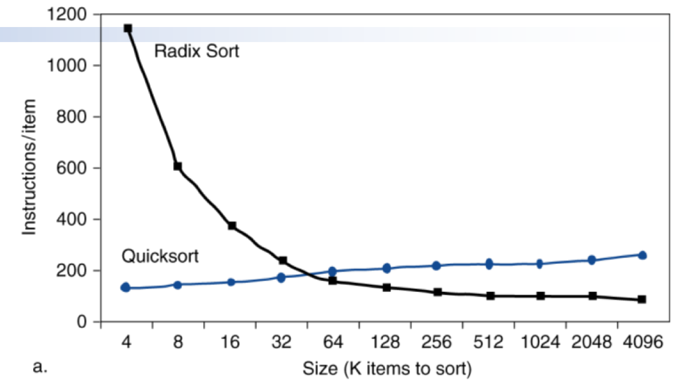
- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

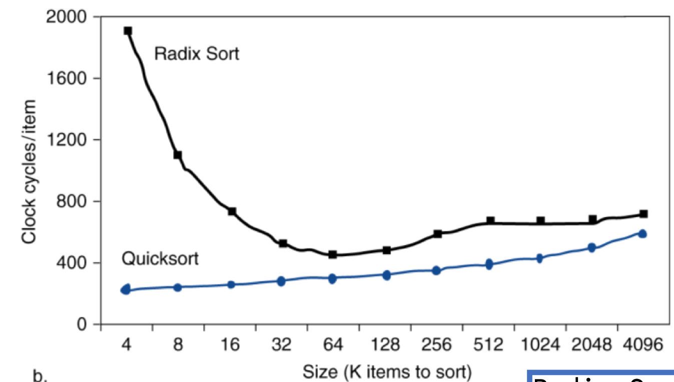
Interactions with Advanced CPUs

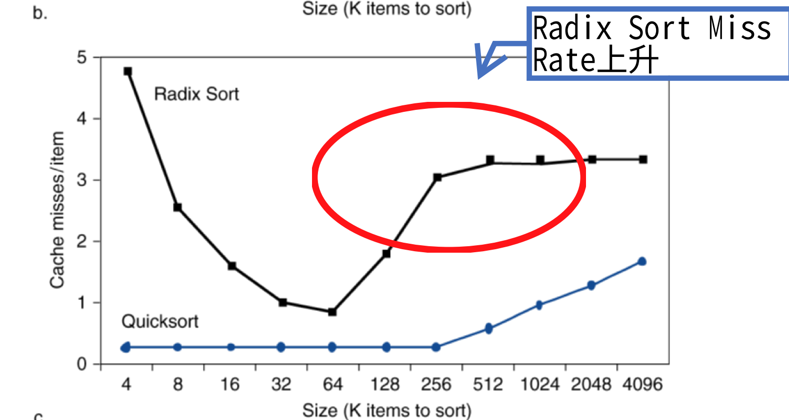
- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

Interactions with Software

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







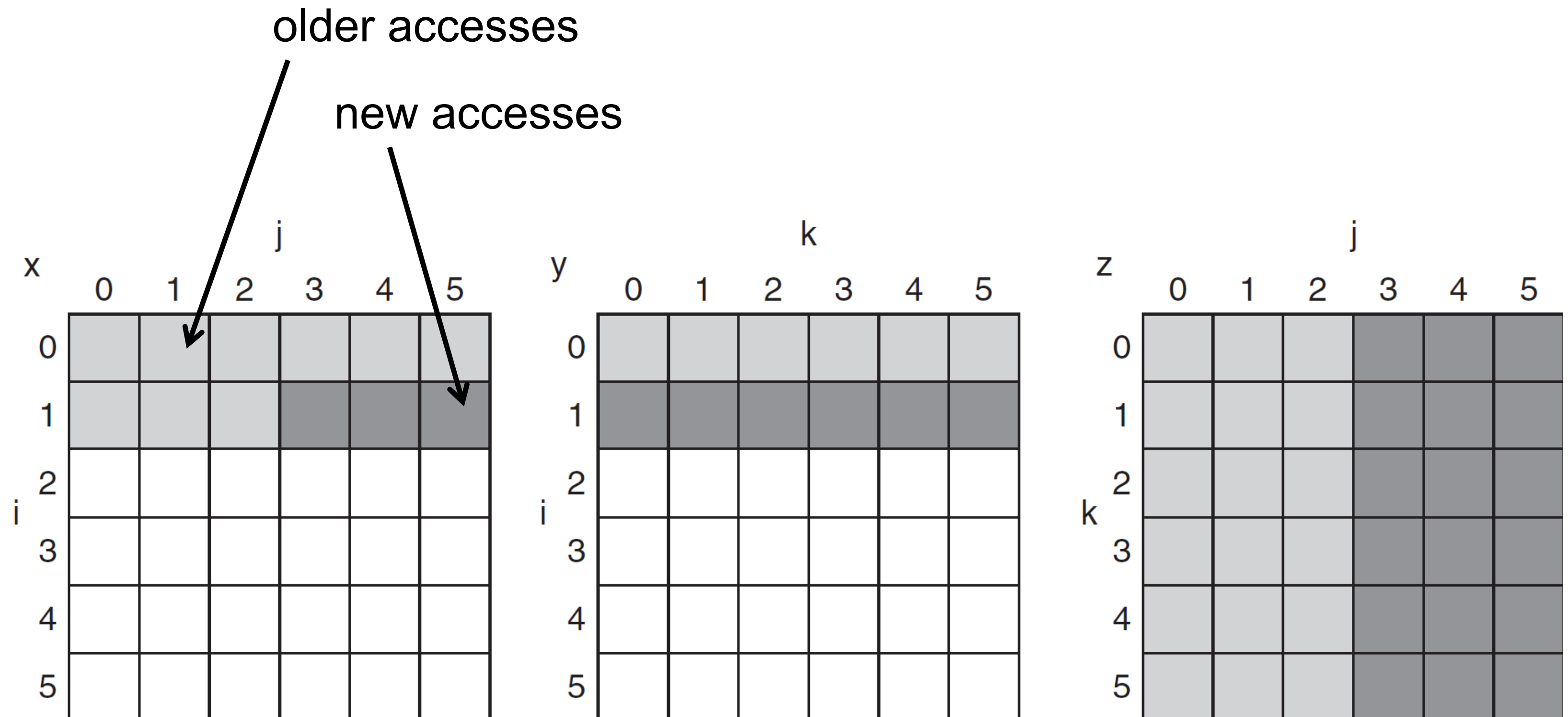
Software Optimization via Blocking

- 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;
}
```

DGEMM Access Pattern

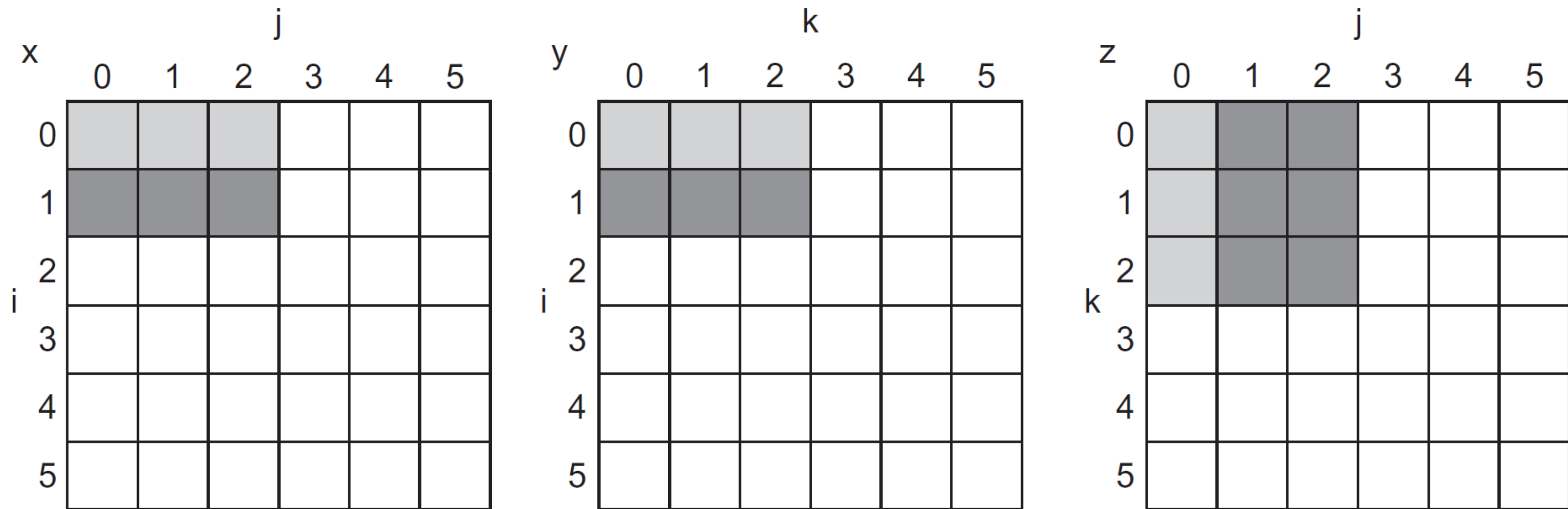
- C, A, and B arrays

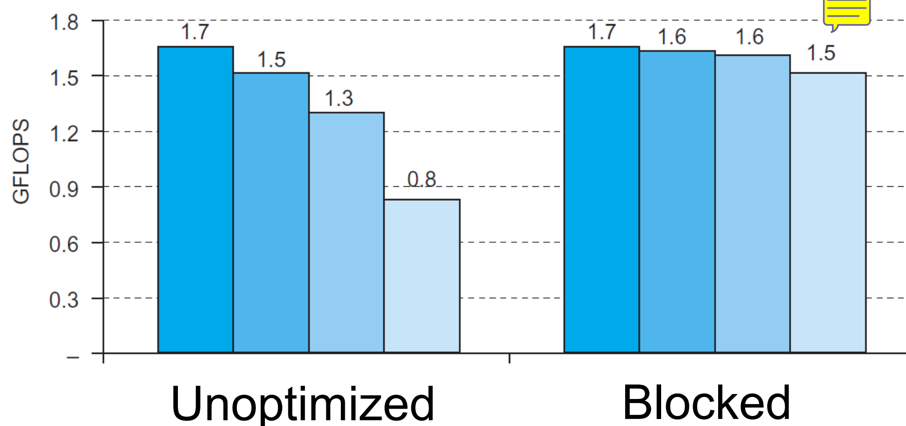


Cache Blocked DGEMM

```
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 }
```

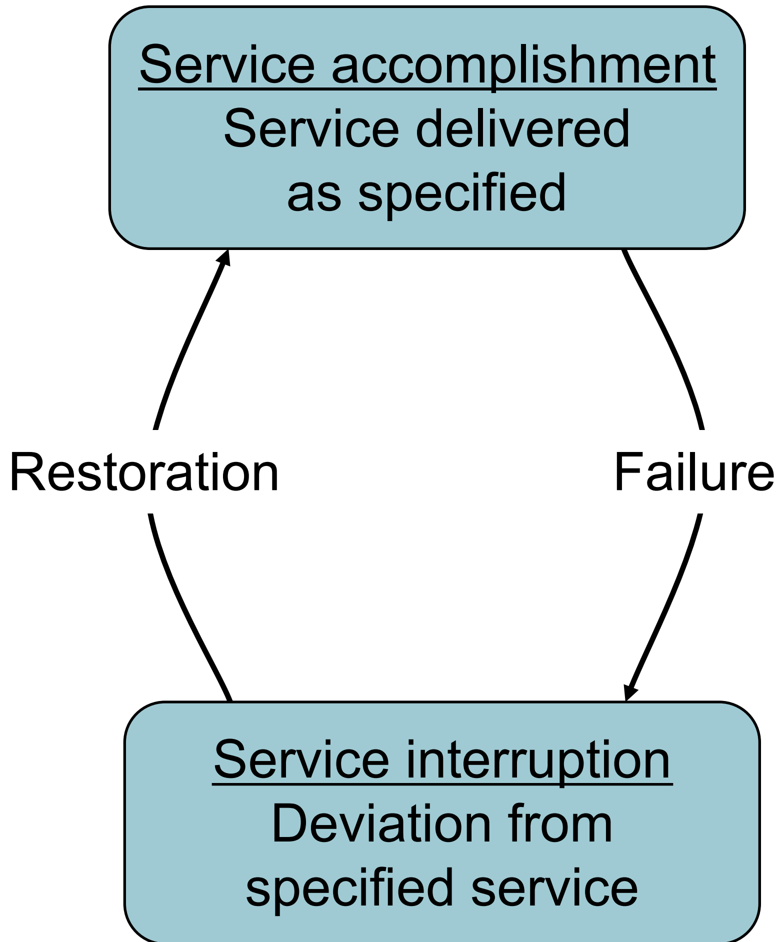
Blocked DGEMM Access Pattern





利用Cache Optimization去優化演算法效率

Dependability



- Fault: failure of a component
  - May or may not lead to system failure

Dependability Measures

- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair