William Stallings
Computer Organization and Architecture
7th Edition

Chapter 5
Memory

SYLLABUS

Characteristics of memory system and hierarchy,

Main memory ,ROM, Types of ROM, RAM,

SRAM, DRAM, Flash memory, High speed memories

Cache Memory Organization: Address mapping, Replacement Algorithms

Cache Coherence, MESI protocol, Interleaved and associative memories

Virtual memory, main memory allocation, segmentation paging, secondary storage ,RAID levels

Characteristics of Memory

1. Location
2. Capacity
3. Unit of transfer
4. Access method
5. Performance(SRAM,DRAM)
6. Physical type
7. Physical characteristics
8. Organisation
    1. Direct Mapping,Associative Mapping

1. Location

- CPU
- Internal
- External

2. Capacity

- Word size
  - The natural unit of organisation
- Number of words
  - or Bytes

Types of various Units of Memory-

Byte

Kilo Byte

Mega Byte

Giga Byte

Tera Byte

Peta Byte

Exa Byte

Zetta Byte

Yotta Byte

NAME	EQUAL TO	SIZE(IN BYTES)
Bit	1 bit	1/8
Nibble	4 bits	1/2 (rare)
Byte	8 bits	1
Kilobyte	1024 bytes	1024
Megabyte	1, 024kilobytes	1, 048, 576
Gigabyte	1, 024 megabytes	1, 073, 741, 824
Terrabyte	1, 024 gigabytes	1, 099, 511, 627, 776
Petabyte	1, 024 terrabytes	1, 125, 899, 906, 842, 624
Exabyte	1, 024 petabytes	1, 152, 921, 504, 606, 846, 976
Zettabyte	1, 024 exabytes	1, 180, 591, 620, 717, 411, 303, 424
Yottabyte	1, 024 zettabytes	1, 208, 925, 819, 614, 629, 174, 706, 176

3. Unit of Transfer

- **INTERNAL**
  - Usually governed by **data bus width**
- **EXTERNAL**
  - Usually a **block** which is much larger than a word
- **Addressable unit**
  - **Smallest location which can be uniquely addressed**

4. Access Methods (1)

- **Sequential**
  - —Start at the beginning and read through in order
  - —Access time depends on location of data and previous location
  - —e.g. tape

- **Direct**
  - —Individual blocks have unique address
  - —Access is by jumping to vicinity plus sequential search
  - —Access time depends on location and previous location
  - —e.g. disk

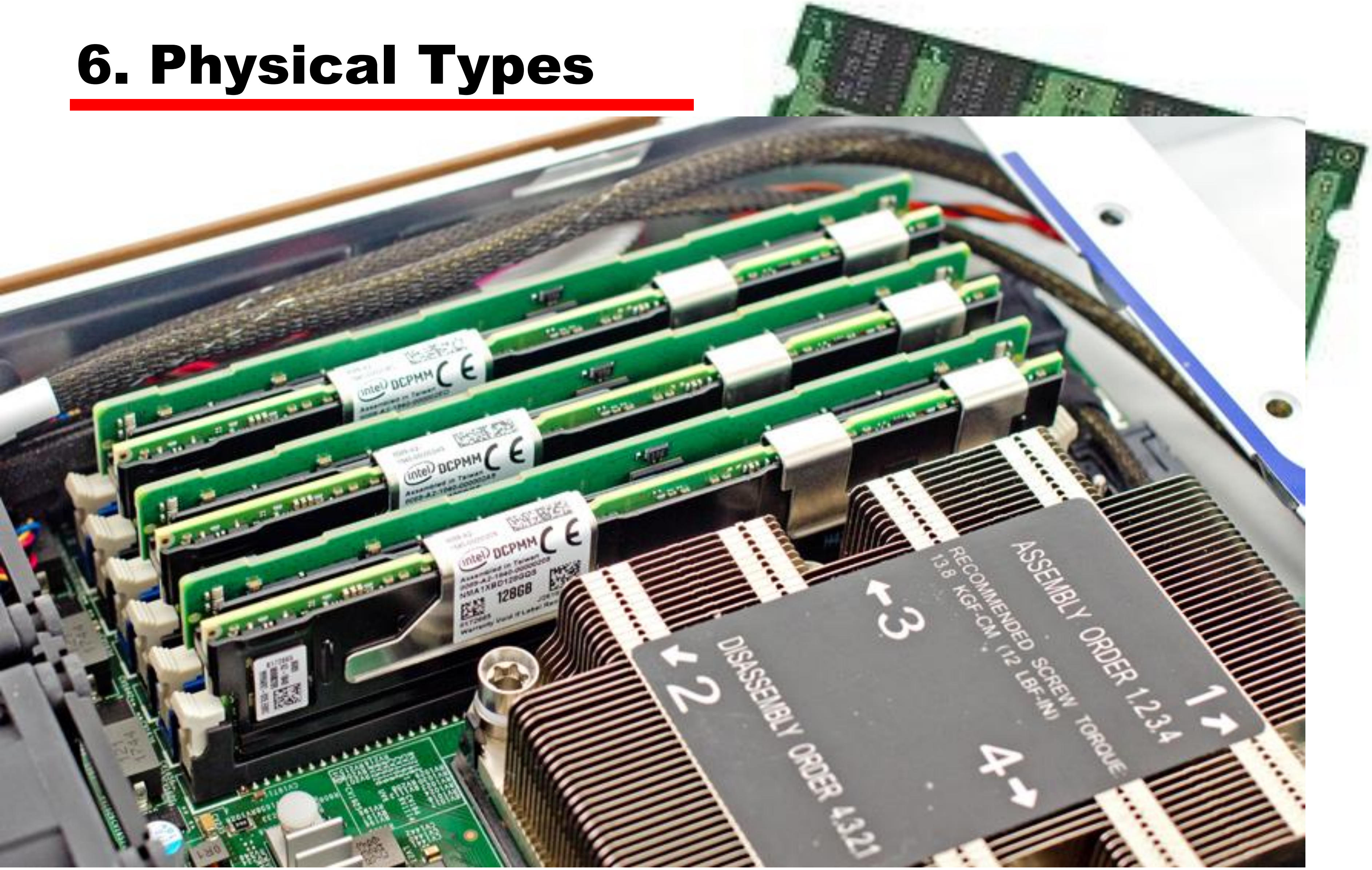
Access Methods (2)

- **Random**
  - Individual addresses identify locations exactly
  - Access time is independent of location or previous access
  - e.g. RAM
- **Associative**
  - Data is located by a comparison with contents of a portion of the store
  - Access time is independent of location or previous access
  - e.g. cache

5. Performance

- **Access time**
  - Time between requesting for operation and the time it is made available at the required location
- **Memory Cycle time**
  - Minimum time elapsed between two **consecutive read requests**
- **Transfer Rate**
  - Rate at which data can be moved

6. Physical Types

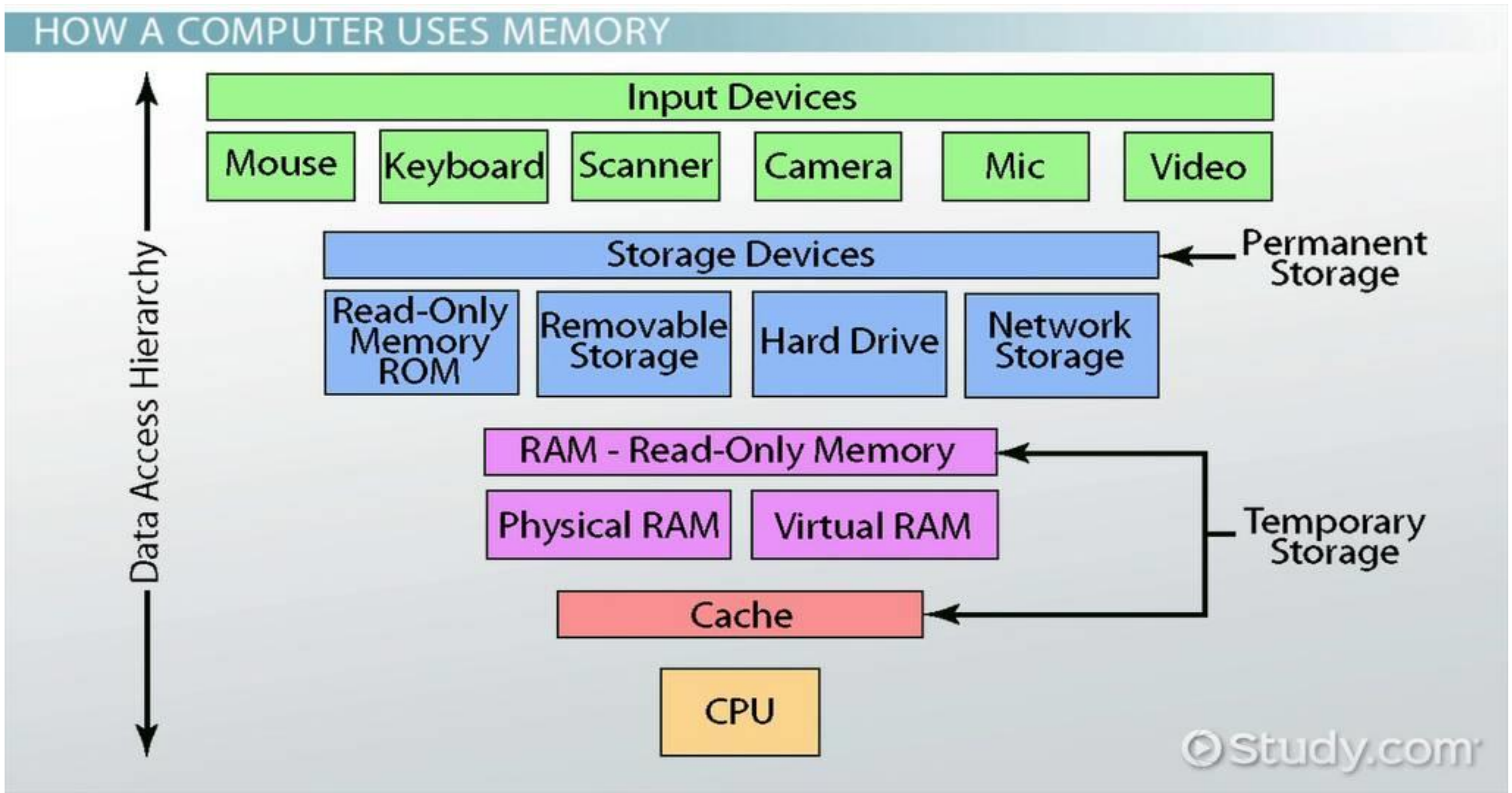


7. Physical Characteristics

- Decay
- Volatility
- Erasable
- Power consumption

8. Organisation

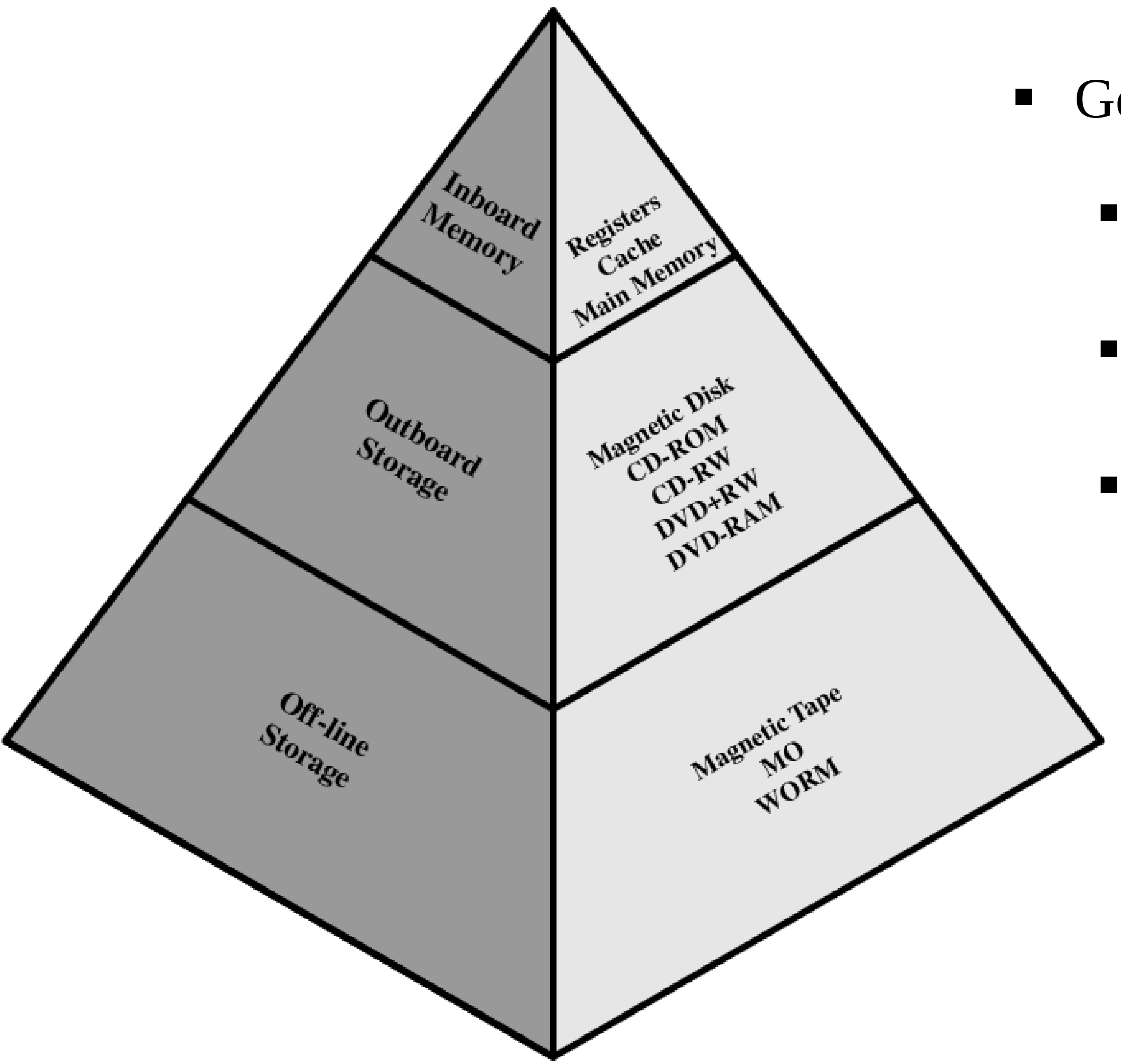
- Physical arrangement of bits into words
- Not always obvious
- e.g. interleaved



Memory Hierarchy

- Registers
  - In CPU
- Internal or Main memory
  - May include one or more levels of cache
  - “RAM”
- External memory
  - Backing store

Memory Hierarchy - Diagram

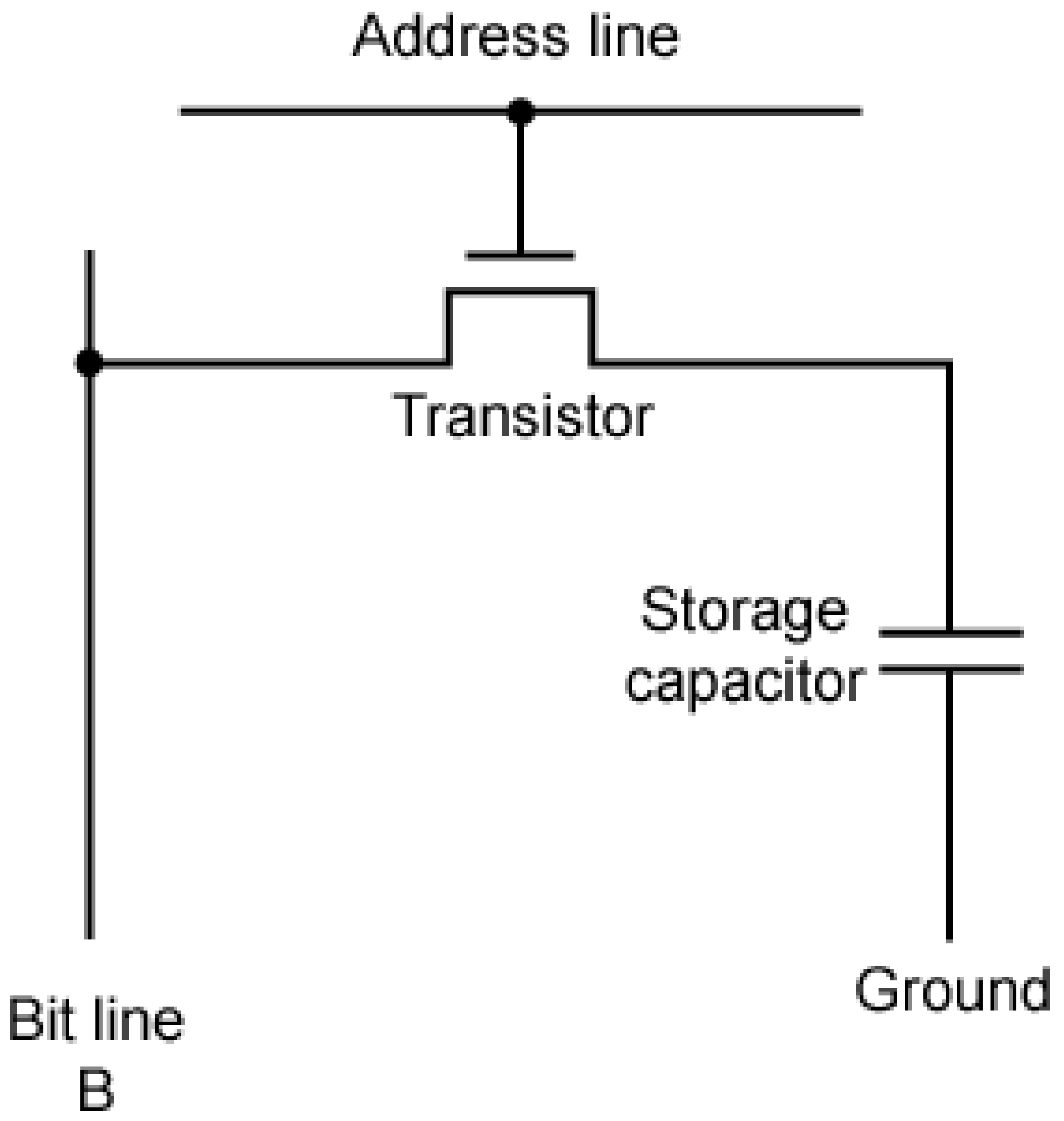


- Going down the hierarchy
  - Decreasing **Cost**
  - Increase **Capacity**
  - Increase **Access Time**

RAM- Random Access Memory

- RAM
  - random access
  - Read/Write
  - Volatile
  - Temporary storage
  - Static or dynamic

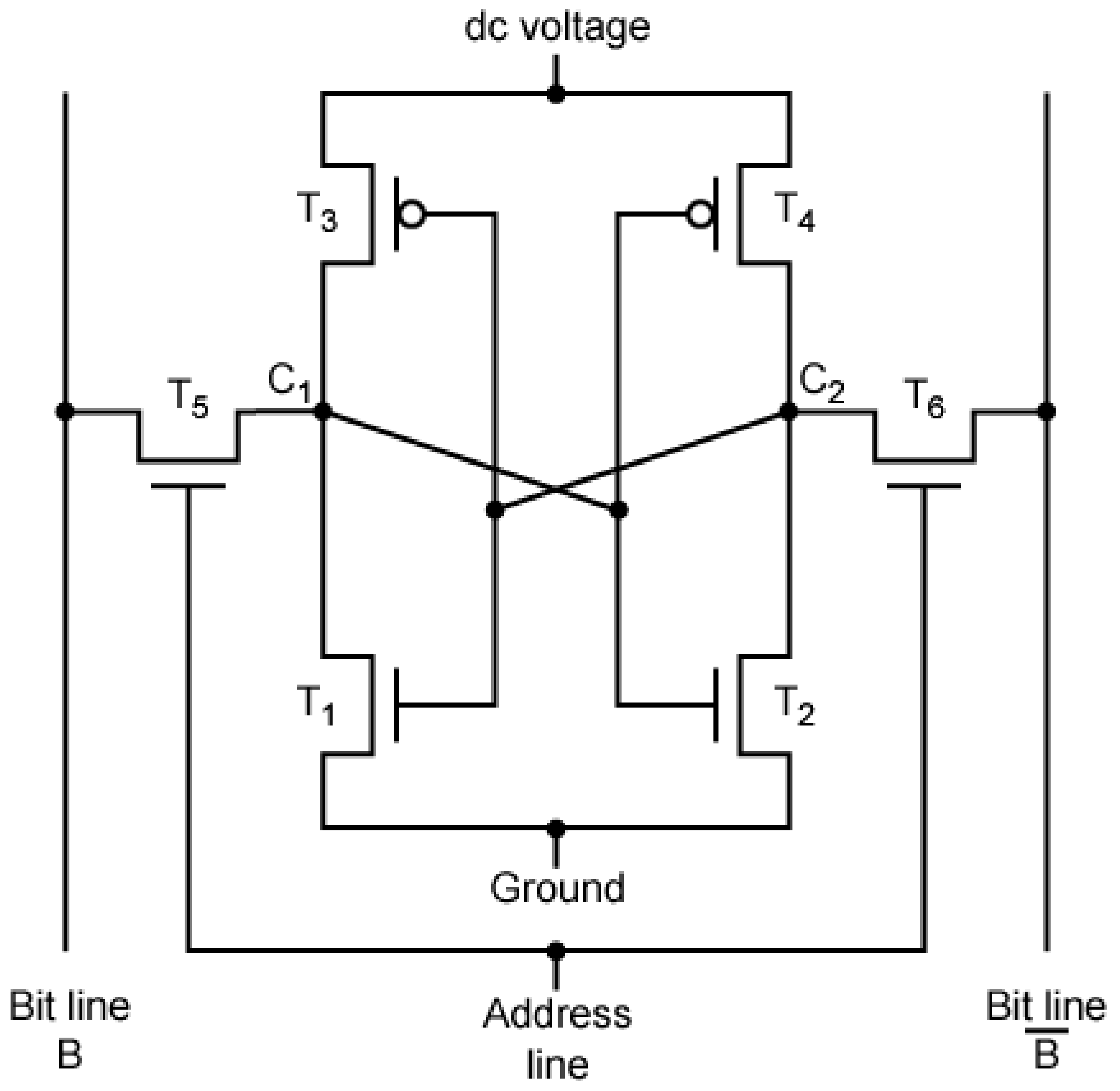
Dynamic RAM Structure



Dynamic RAM

- Bits stored as charge in capacitors
- Charges leak
- Need **refreshing** even when powered
- Simpler construction
- Less expensive
- Need refresh circuits
- Slower
- Main memory
- Essentially analog
    - Level of charge determines value

Static RAM Structure

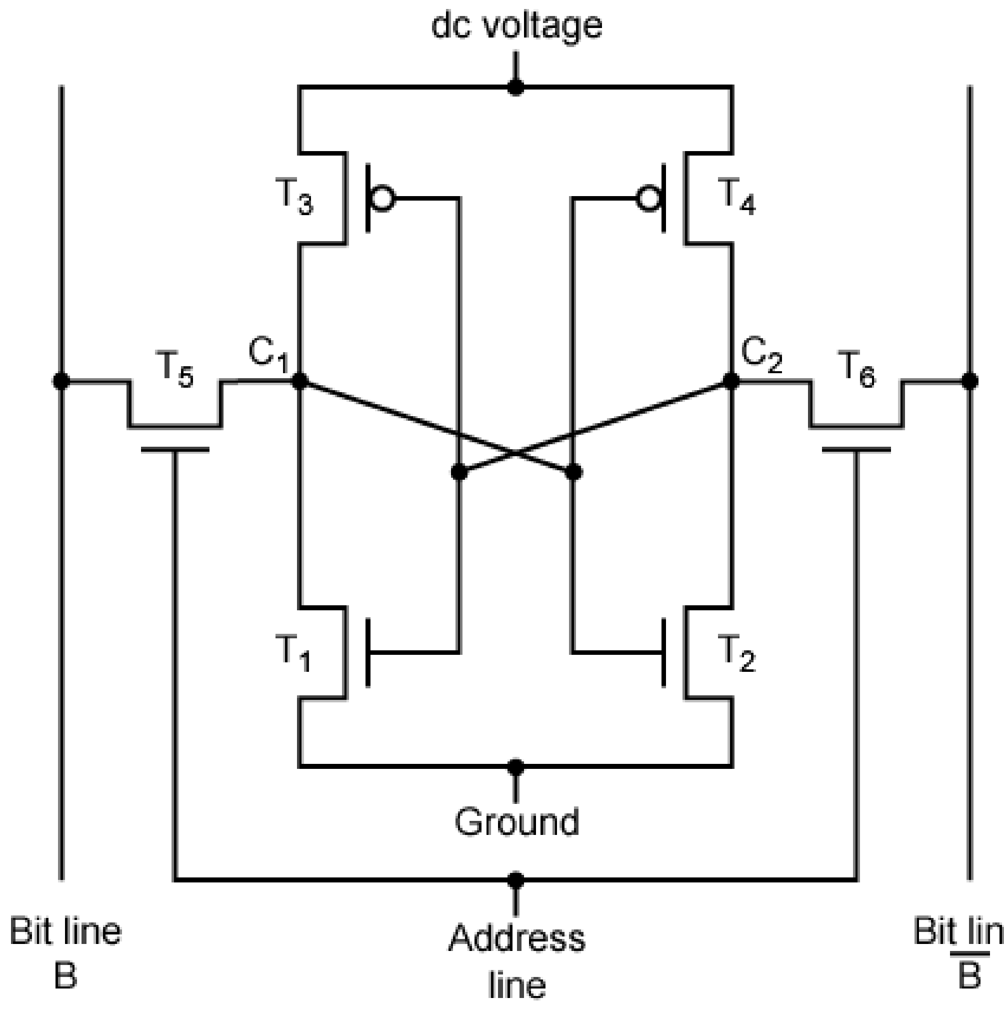


Static RAM

- Bits stored as on/off switches
- No charges to leak
- No refreshing needed when powered
- More complex construction
- Larger information per bit
- More expensive
- Does not need refresh circuits
- Faster
- Cache
- Digital
    - Uses flip-flops

Static RAM Operation

- Transistor arrangement gives stable logic state
- State 1
  - C_1 high, C_2 low
  - T_1 T_4 off, **T_2 T_3 on**
- State 0
  - C_2 high, C_1 low
  - T_2 T_3 off, **T_1 T_4 on**



SRAM v DRAM

- Both volatile
  - Power needed to preserve data
- Dynamic cell
  - Simpler to build, smaller
  - More dense
  - Less expensive
  - Needs refresh
  - Larger memory units
- Static
  - Faster
  - Cache

Read Only Memory (ROM)

- Permanent storage
  - Non volatile
  - Can read a ROM but cant write new data into it
- TYPES OF ROM
  - PROM
  - EPROM
  - EEPROM

PROM-Programmable ROM

- Written during manufacture
- Programmable (“once”)
  - **PROM**
  - Small amount of data to be written
  - Less expensive
  - Non volatile, written only once
  - Writing performed electrically at the time of chip fabrication

Read "mostly"

- Erasable Programmable (**EPROM**)
  - Erased by UV
- Electrically Erasable (**EEPROM**)
  - Takes much longer to write than read
- Flash memory
  - Erase whole memory electrically

EPROM

- Read and written electrically
- All storage cells should be erased electrically to **initial state** by exposure to UV radiation
- Can be altered multiple times and holds data virtually indefinitely
- More expensive than PROM

EEPROM

- Can be written anytime without erasing prior contents
- Write operation takes longer than read
- More expensive than EPROM, less dense

Types of ROM

1. Programmable Read Only Memory (PROM)

- Empty of data when manufactured
- May be permanently programmed by the user

2. Erasable Programmable Read Only Memory (EPROM)

- Can be programmed, erased and reprogrammed
- The EPROM chip has a small window on top allowing it to be erased by shining ultra-violet light on it
- After reprogramming the window is covered to prevent new contents being erased
- Access time is around 45 – 90 nanoseconds

Types of ROM

3. Electrically Erasable Programmable Read Only Memory (EEPROM)

- Reprogrammed electrically **without** using ultraviolet light
- Must be removed from the computer and placed in a special machine to do this
- Access times between 45 and 200 nanoseconds

4. Flash ROM

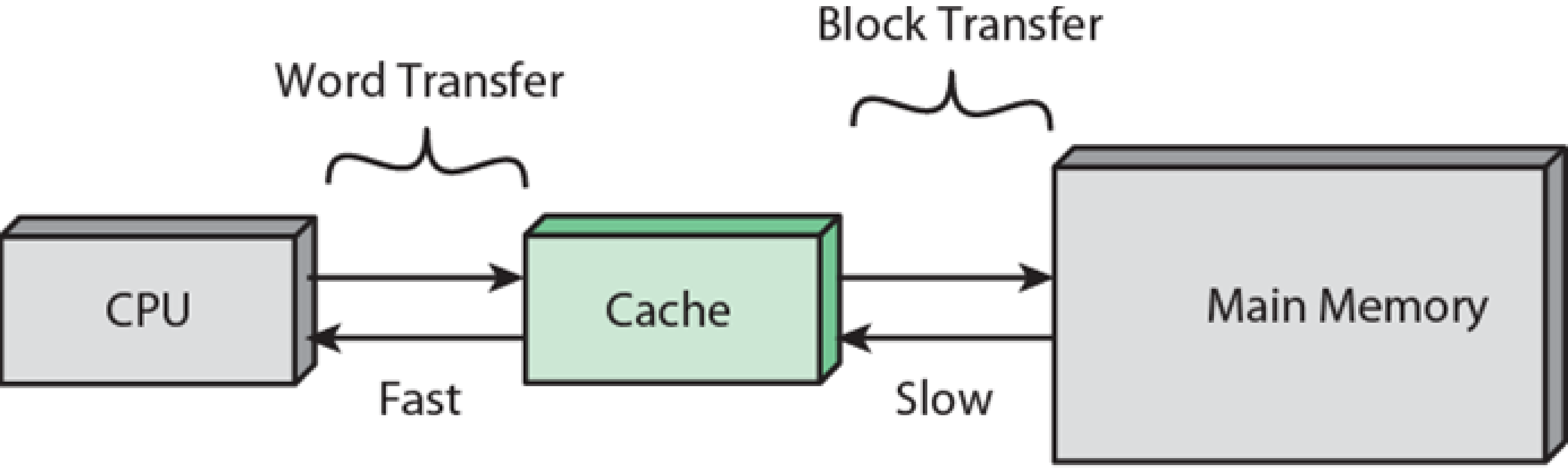
- Similar to EEPROM
- However, can be reprogrammed while still in the computer
- Easier to upgrade programs stored in Flash ROM
- Used to store programs in devices e.g. modems
- Access time is around 45 – 90 nanoseconds

5. ROM cartridges

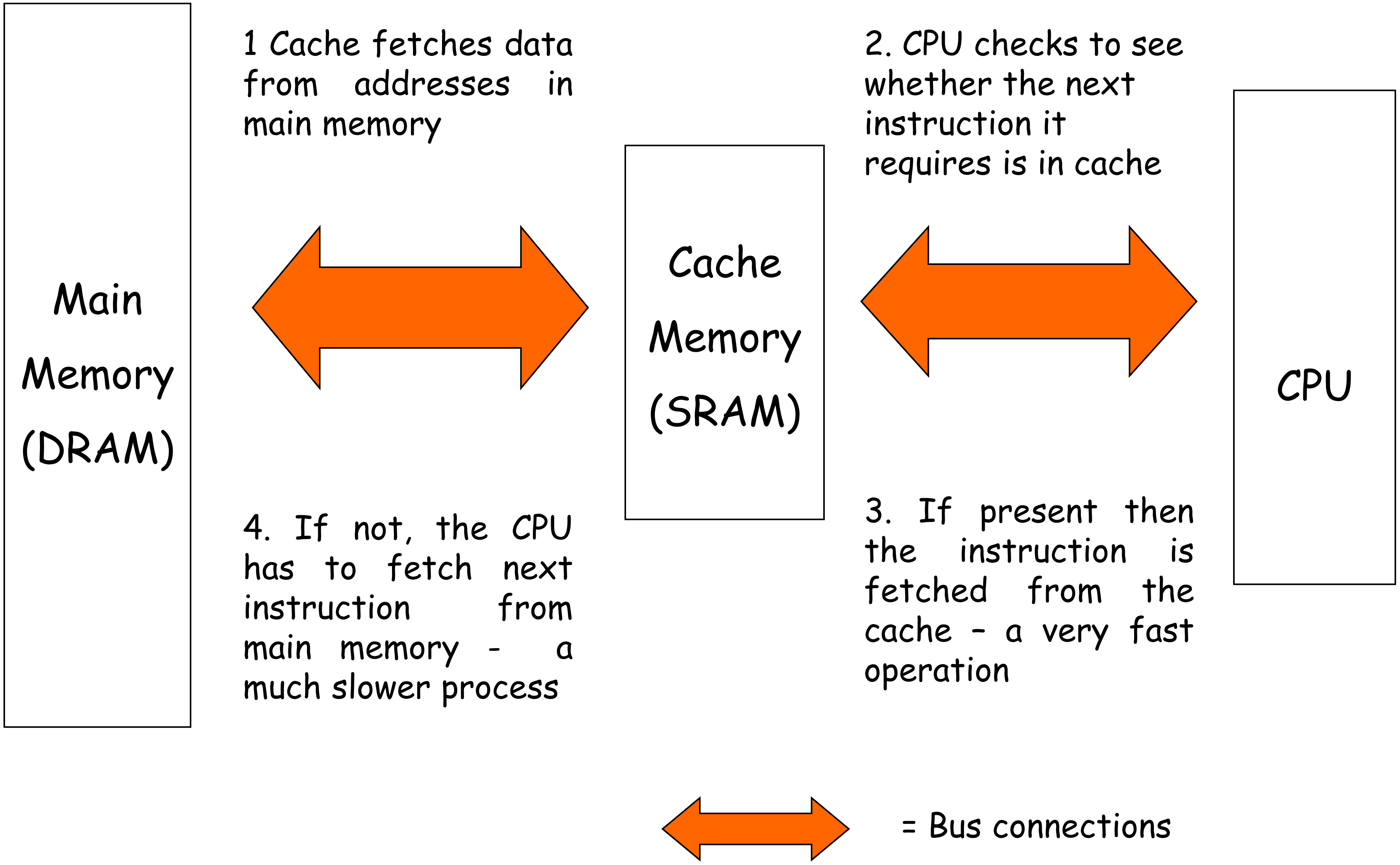
- Commonly used in games machines
- Prevents software from being easily copied

Cache

- **Small** amount of fast memory
- Sits between main memory and CPU
- May be located on CPU chip or module



The operation of cache memory



1 Cache fetches data from addresses in main memory

2. CPU checks to see whether the next instruction it requires is in cache

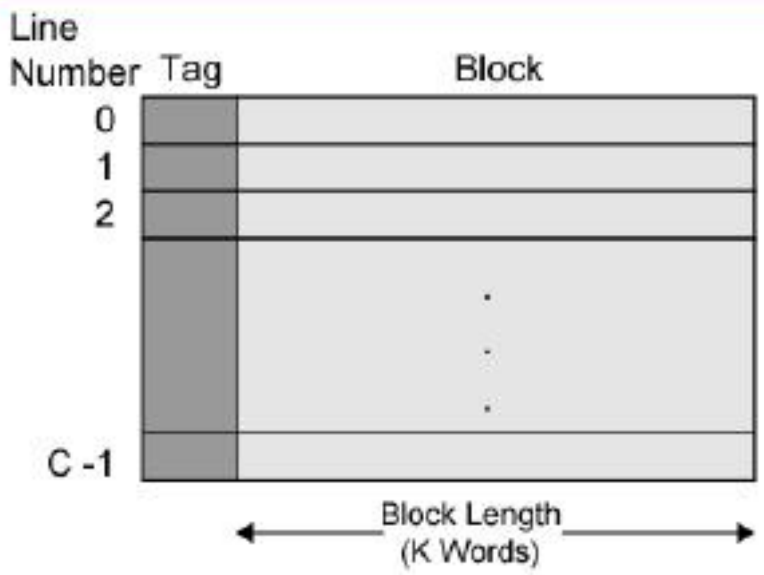
3. If present then the instruction is fetched from the cache – a very fast operation

4. If not, the CPU has to fetch next instruction from main memory - a much slower process

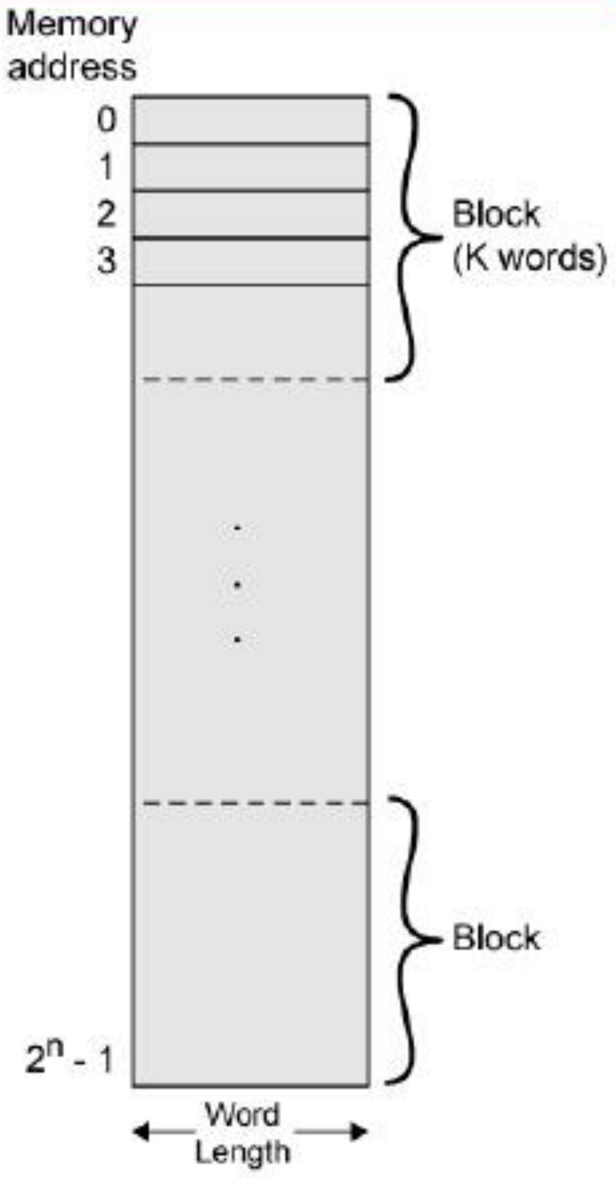
Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes **tags** to identify which block of main memory is in each cache slot

Cache/Main Memory Structure



(a) Cache



(b) Main memory

TAG - **A unique identifier for a group of data. Since different regions of memory may be mapped into a block, the tag is used to differentiate between them.**

- CPU cache is divided into three main 'Levels', L1, L2, and L3.
  - The hierarchy is according to the speed, and thus, the size of the cache.

- **Level 1 (L1) cache** -fast small, **embedded** in the processor chip (CPU).
- **Level 2 (L2) cache** more capacity than L1; **located** on the CPU or on a separate chip or coprocessor.
- **Level 3 (L3) cache** specialized memory to improve the performance of L1 and L2.
  - Slower than L1 or L2, double the speed of RAM

Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

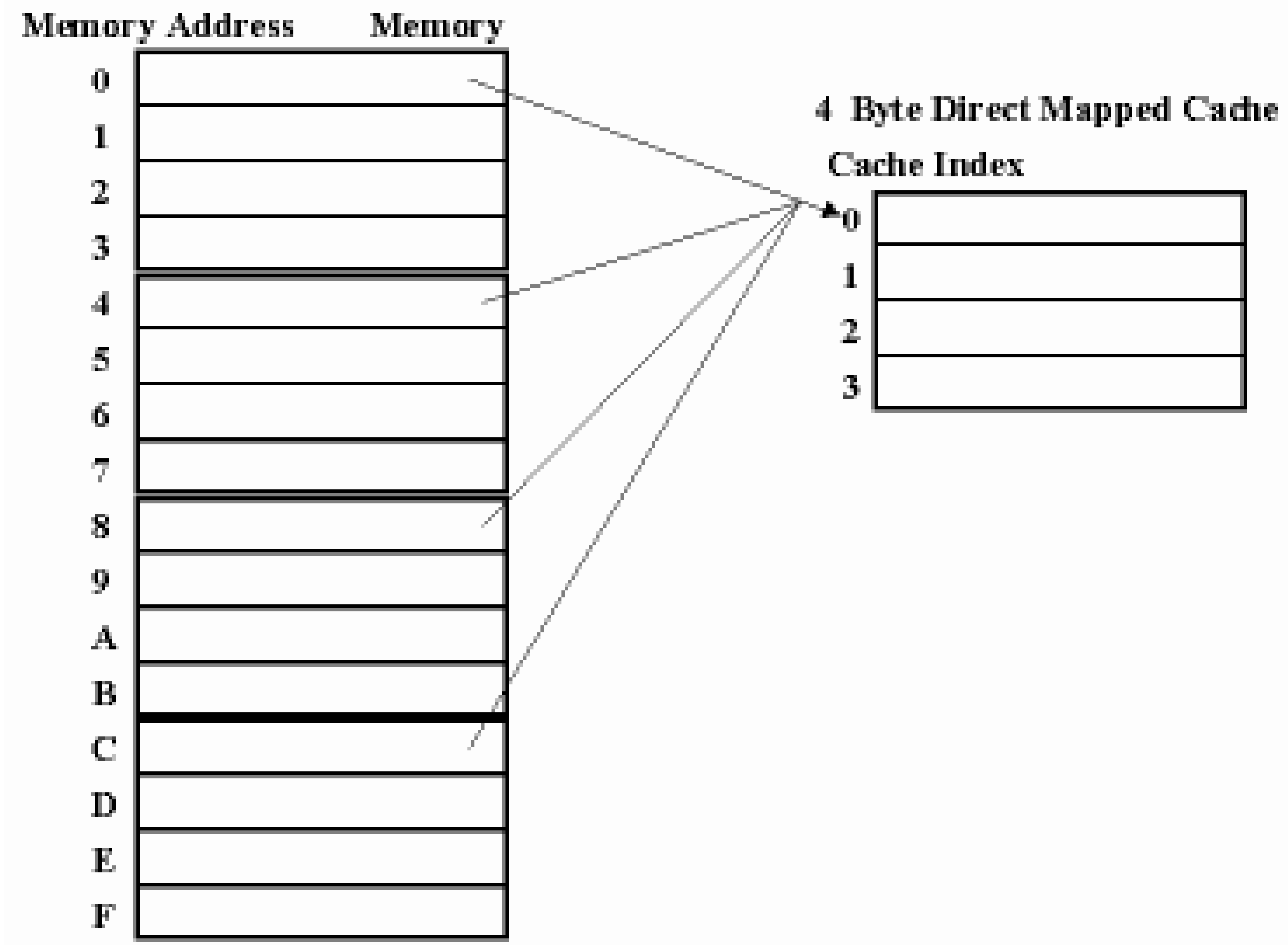
Size does matter

- Cost
  - More cache is expensive
- Speed
  - More cache is faster (up to a point)
  - Checking cache for data takes time

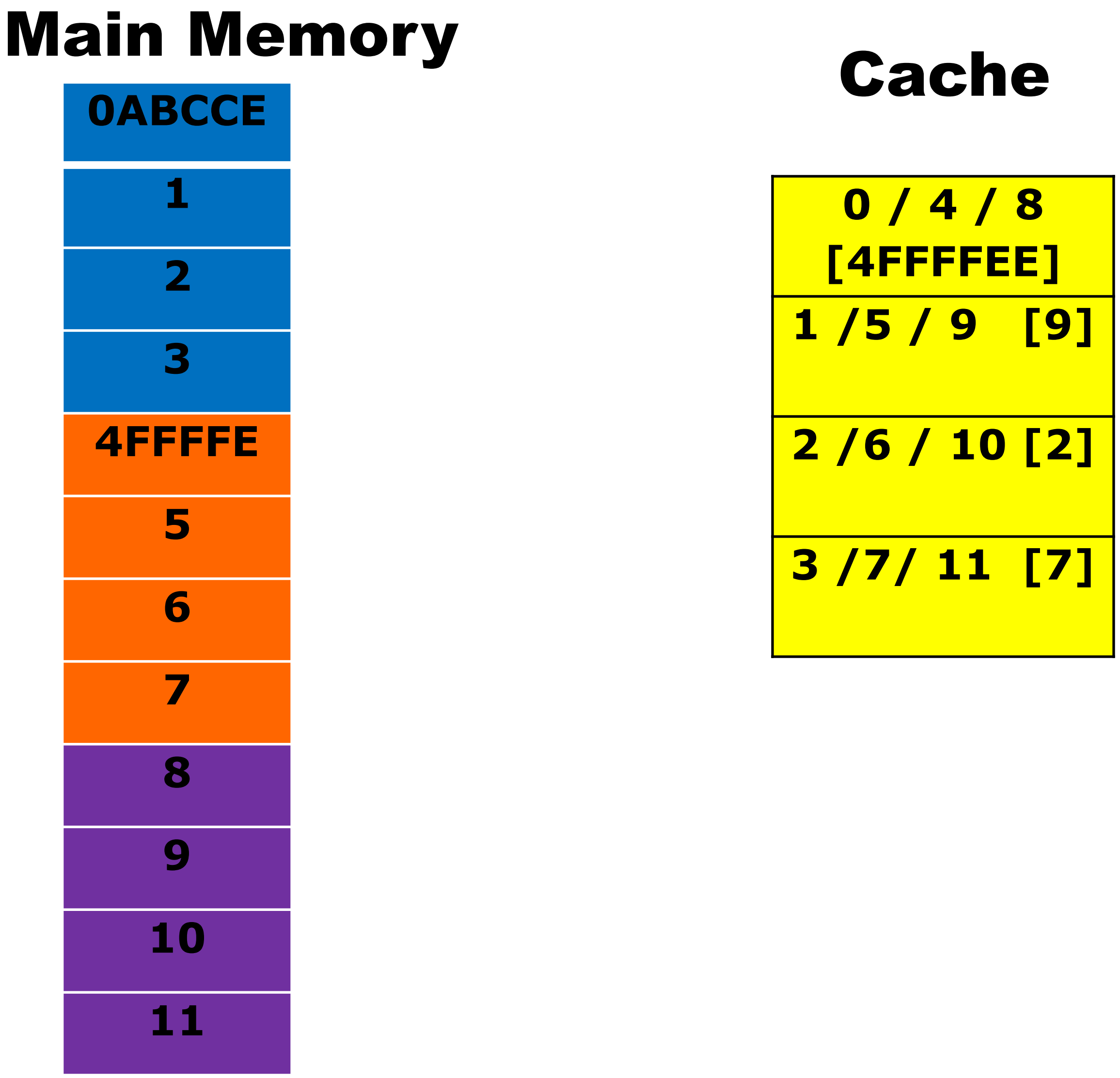
MAPPING TECHNIQUES

- **DIRECT MAPPING**
- ASSOCIATIVE MAPPING
  - FULLY ASSOCIATIVE MAPPING
  - SET ASSOCIATIVE MAPPING
    - **2-WAY SET ASSOCIATIVE MAPPING**

DIRECT MAPPING CONCEPT



Direct Mapping



DIRECT MAPPING

- Cache- 128 blocks of 16 words each
  - 128 * 16 = 2048 **approx 2 KB**

0	16 words
1	16 words
…	16 words
127	16 words

- Main Memory – 4K blocks of 16 words each
  - 4K * 16 = 64000 **approx 64 KB**

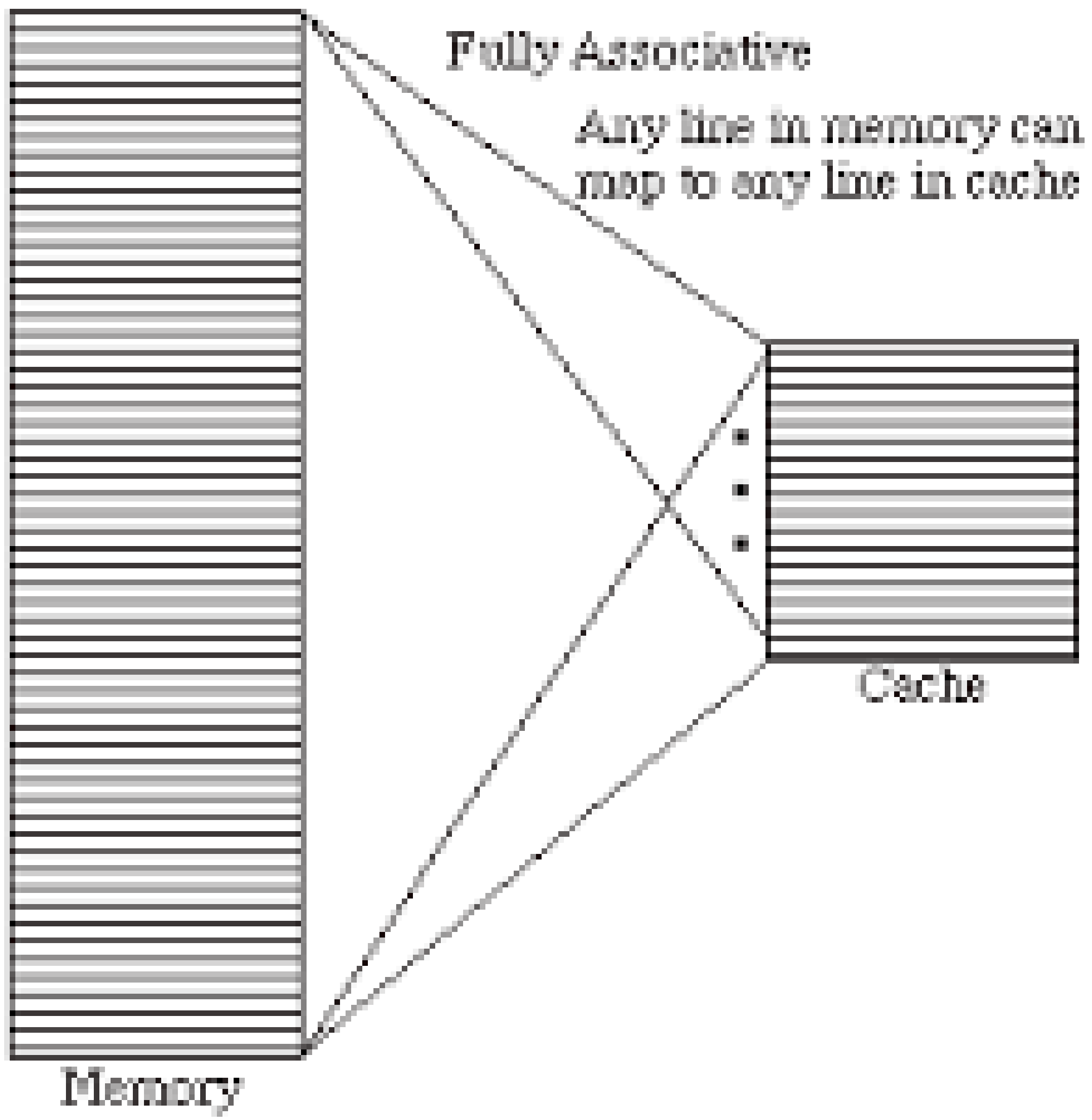
- **TOTAL ADDRESS SIZE 16 bit**

Tag	Block/Line(128)	Word(16)
5(16-(7+4))	**7(2^7)**	**4 (2^4)**

Working of Direct Mapping

- **Word**" field selects one from among the 16 addressable words in a line.
- The "**Line**" field defines the cache line where this memory line should reside.
- The "**Tag**" field of the address is then compared with that cache line's 5-bit tag to determine whether there is a hit or a miss.
  - If there's a miss, we need to swap out the memory line that occupies that position in the cache and replace it with the desired memory line.

FULLY ASSOCIATIVE MAPPING



ASSOCIATIVE MAPPING

- Cache- 128 blocks of 16 words each
- 128 * 16 = 2048 **approx 2 KB**

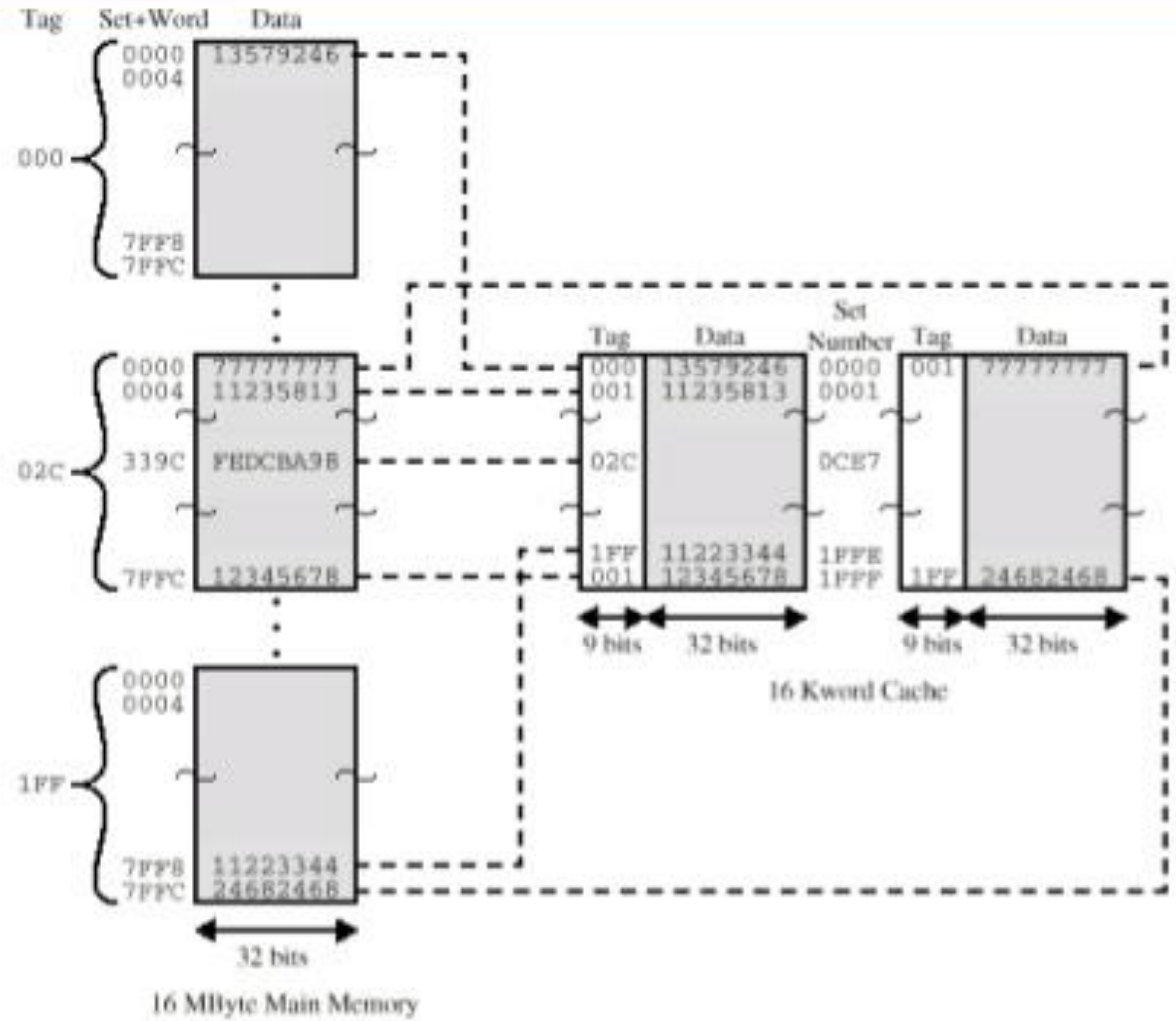
- Main Memory – 4K blocks of 16 words each
- 4K * 16 = 64000 **approx 64 KB**
- **TOTAL ADDRESS SIZE 16 bit**

Tag	Word
12(16-4)	**4 (2^4)**

Working of Fully Associative Mapping

- "Tag" field identifies one of the $2^{12} = 4096$ memory lines;
- All the cache tags are searched to find out whether or not the Tag field matches one of the cache tags.
- If so, we have a hit, and if not there's a miss and we need to replace one of the cache lines by this line before reading or writing into the cache.
- (The "Word" field again selects one from among 16 addressable words (bytes) within the line.)

Two Way Set Associative Mapping Example



SET ASSOCIATIVE MAPPING(2-way)

- Cache- 128 blocks of 16 words each
- 128 * 16 = 2048 **approx 2 KB**
- Set divided into 2 (128 /2) =64
- Main Memory – 4K blocks of 16 words each
- 4K * 16 = 64000 **approx 64 KB**

Tag	Set	Word
6(16-(4+6))	**6(2 ^{6})**	**4 (2 ^{4})**

Working

- "Tag" field identifies one of the $2^6 = 64$ different memory lines in each of the $2^6 = 64$ different "Set" values.
- Since each cache set has room for only two lines at a time, the search for a match is limited to those two lines (rather than the entire cache).
- If there's a match, we have a hit and the read or write can proceed immediately.
- Otherwise, there's a miss and we need to replace one of the two cache lines by this line before reading or writing into the cache. (The "Word" field again select one from among 16 addressable words inside the line.)
- In set-associative mapping, when the number of lines per set is *n*, the mapping is called *n*-way associative. For instance, the above example is 2-way associative.

Direct Mapping Cache Line Table

Cache line	Main Memory blocks held
0	0, m, 2m, 3m…2s-m
1	1,m+1, 2m+1…2s-m+1
m-1	m-1, 2m-1,3m-1…2s-1

Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

Associative Mapping

- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line’s tag is examined for a match
- Cache searching gets expensive

Set Associative Mapping

- Cache is divided into a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

Problem statement

Consider a cache consisting of 256 blocks of 16 words each for a total of 4096(4KB) words and assume that the main memory is addressable by a 16 bit address and it consists of 4KB blocks of 16 words.

TAG :- 4
Block:- 8
Word:- 4

How many bits are there in each of the TAG,BLOCK/SET and WORD field for Direct Mapping , Fully Associative and 2-way set associative techniques?

Problem statement

A block set associative cache memory consists of 128 blocks divided into four block sets. The main memory consists of 16,384 blocks and each block contains 256 words.

i) How many bits are required for addressing the main memory?

ii) How many bits are needed to represent the TAG, SET and WORD fields?

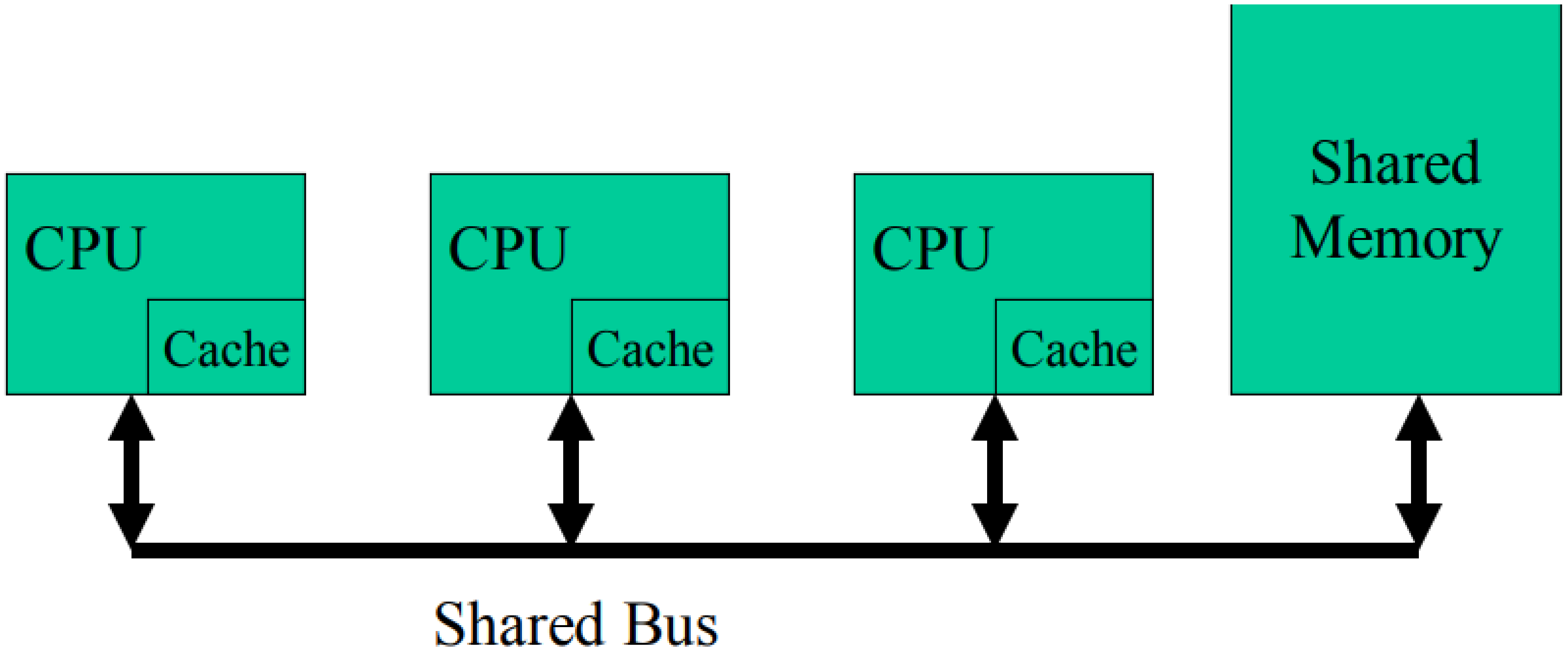
Problem statement

A block set associative cache memory consists of 64 blocks divided into four block sets. The main memory consists of 4096 blocks and each block contains 128 words.

i) How many bits are there in main memory?

ii) How many bits are needed to represent the TAG, SET and WORD fields?

CACHE COHERENCE



Cache Coherence

- Problem - multiple copies of same data in different caches
- Can result in an **inconsistent** view of memory
  - Write through
  - Write back policy
  - Write invalidate
  - Write Update

Software Solutions

- **Compiler and operating system** deal with problem
- Overhead transferred to compile time
- Complier marks data likely to be changed and OS prevents such data from being cached
- Design complexity transferred from hardware to software

Hardware Solution

- Cache coherence protocols
- Dynamic recognition of potential problems
- Run time
- More **efficient** use of cache
- **Transparent** to programmer
- **Snoopy protocols**(to maintain cache consistency)

Snoopy Protocols

- Distribute cache coherence responsibility among **cache controllers**
- Cache recognizes that a line is shared
- Updates announced to other caches
- Suited to bus based multiprocessor
- Increases bus traffic

Write through

- **All writes go to main memory as well as cache**
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic
- Slows down writes

Write back

- **Updates initially made in cache only**
- Update bit for cache slot is set when update occurs
- If block is to be replaced, write to main memory only if **update bit is set**
- Other caches get out of sync
- I/O must access main memory through cache

Write Update

- Multiple readers and writers
- Updated word is distributed to all other processors
- Some systems use an adaptive mixture of both solutions

Write Invalidate

- **Multiple readers, one writer**
- When a write is required, all other caches of the line are invalidated
- Writing processor then has **exclusive access** until line required by another processor
- State of every line is marked as modified, exclusive, shared or invalid
- MESI protocol

MESI Protocol

Commonly implemented for Cache coherence

MESI protocol -four states that a cache line may be in:

- **Modified**
- **Exclusive**
- **Shared**
- **Invalid**

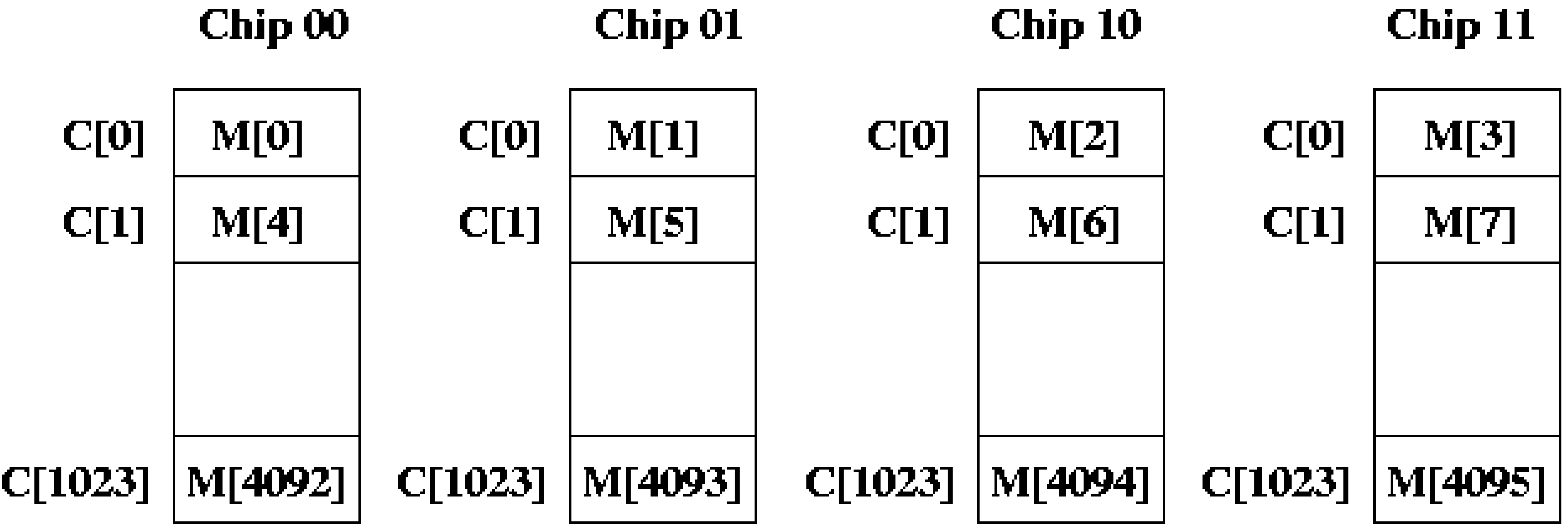
MESI protocol

- **I**nvalid: This cache line is not valid
- **E**xclusive: This **cache** has the only copy of the data. The **memory** is valid.
- **S**hared: More than one cache is holding a copy of this line. The memory copy is valid.
- **M**odified: The line has been modified. The **memory** copy is invalid.

Interleaved
And
Associative Memory

Interleaved Memory

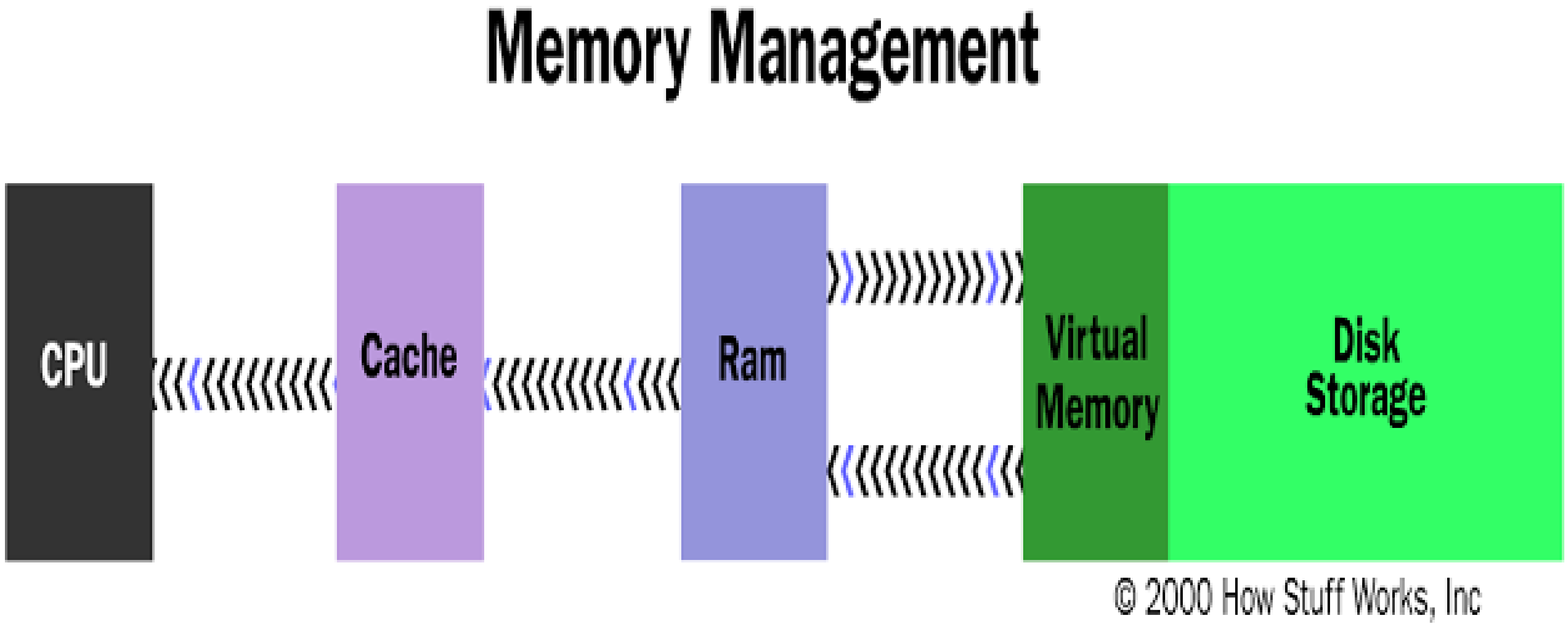
- **Interleaved memory** is a design made to compensate for the relatively slow speed of DRAM
- **Spreads memory** addresses evenly across
- Contiguous memory reads and writes
- Resulting in **higher memory throughputs** due to reduced waiting.

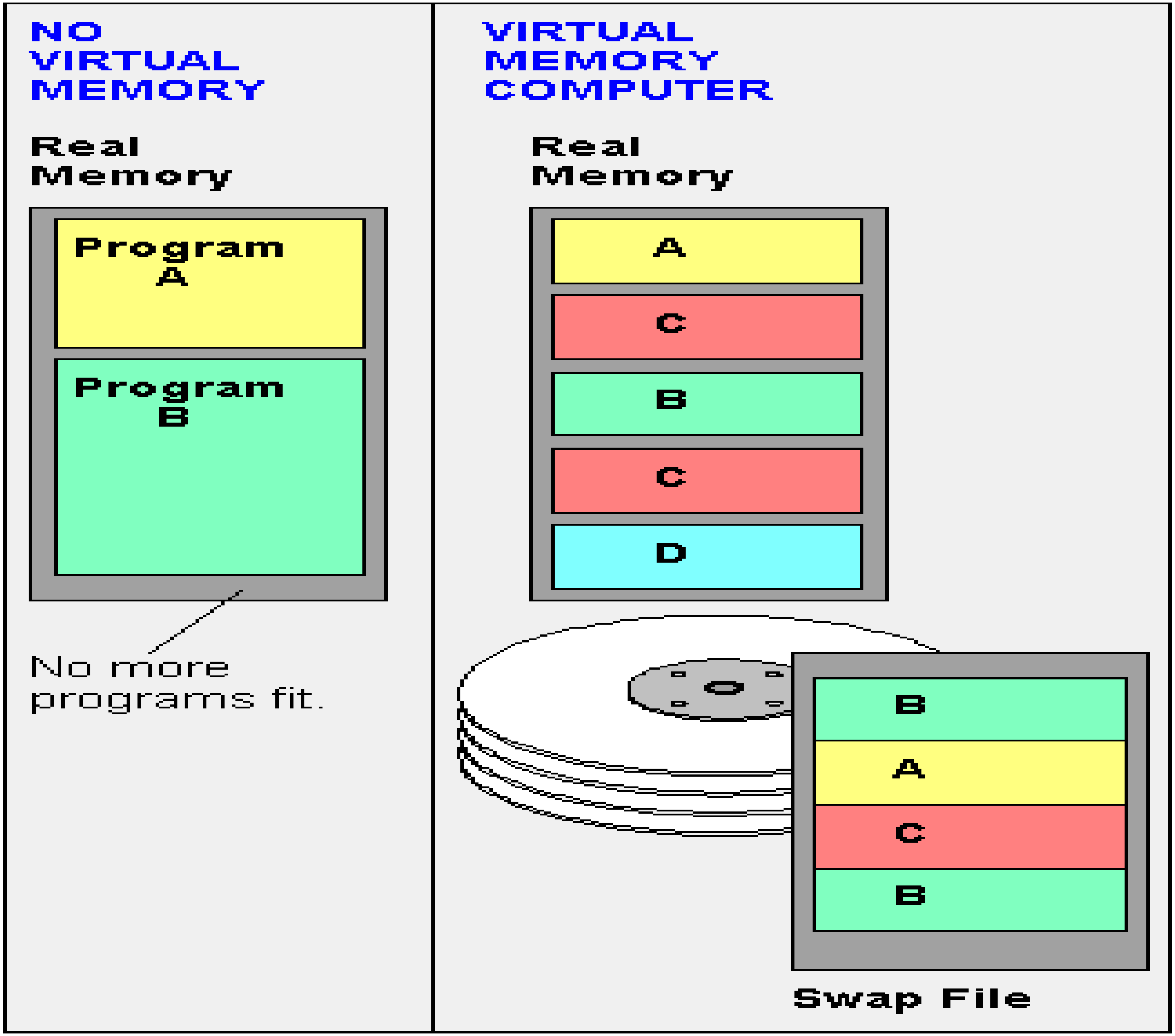

Chip 00
Chip 01
Chip 10
Chip 11
C[0]
M[0]
C[1]
M[4]
C[1023]
M[4092]
C[0]
M[1]
C[1]
M[5]
C[1023]
M[4093]
C[0]
M[2]
C[1]
M[6]
C[1023]
M[4094]
C[0]
M[3]
C[1]
M[7]
C[1023]
M[4095]

Associative Memory

- Content-addressed or associative memory-memory is **accessed by its content** (as opposed to an explicit address).
- **Reference clues** are "associated" with actual memory contents until a desirable match (or set of matches) is found.
- Humans retrieve information best when it can be linked to other related information.

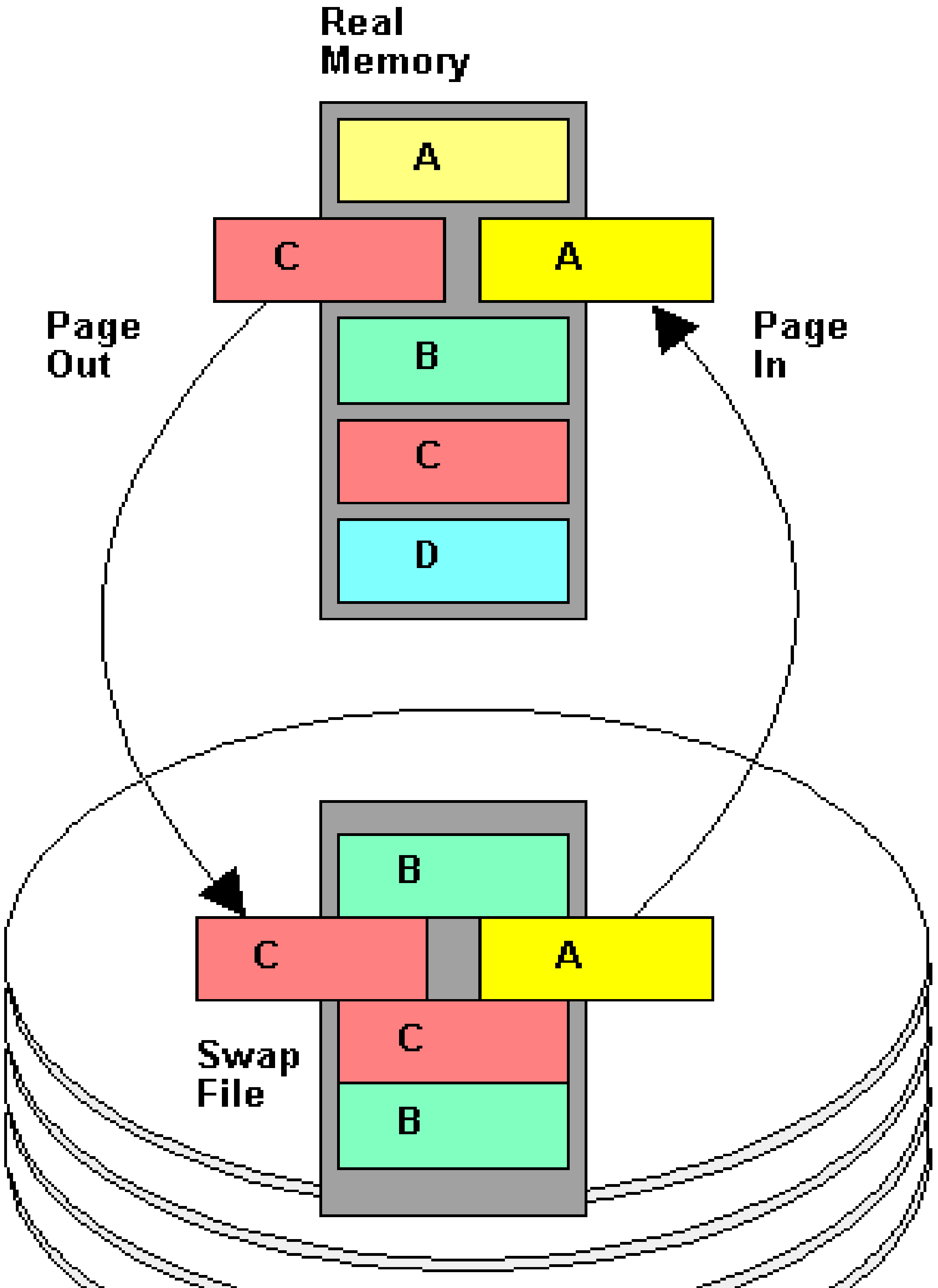
Virtual Memory




NO VIRTUAL MEMORY
Real Memory
Program A
Program B
No more programs fit.
VIRTUAL MEMORY COMPUTER
Real Memory
A
C
B
C
D
B
A
C
B
Swap File

- Virtual memory

Allows more programs to be opened simultaneously by using the hard disk as temporary storage of memory pages.



VIRTUAL MEMORY

- 32 or 64MB of RAM available for CPU usage.
- Users expect all their programs to run at once.
- Ex Email program,a Web browser and word processor(all in RAM simultaneously)
- Find RAM for areas that have not been used recently and copy them onto the hard disk

VIRTUAL MEMORY

- Frees up space in RAM to load the new application.
- **Copying** happens automatically(feels like unlimited RAM space)
- Hard disk space is much cheaper than RAM chips, thus has a economic benefit.
- Read/write speed of a hard drive & technology is not geared toward accessing small pieces of data at a time.

VIRTUAL MEMORY

- **Operating system** has to constantly swap information back and forth between RAM and the hard disk.
- **Thrashing-** computer feels incredibly slow.

PAGING

- Unequal fixed size /Variable Size partitions(Inefficient)
- Primary memory is divided into **small equal fixed sized partitions** (256, 512, 1K) called **page frames**.
- Process are divided into **same sized blocks(pages)** called **paging.**
- Recently referenced pages in the memory.
- Need a page table to this management.

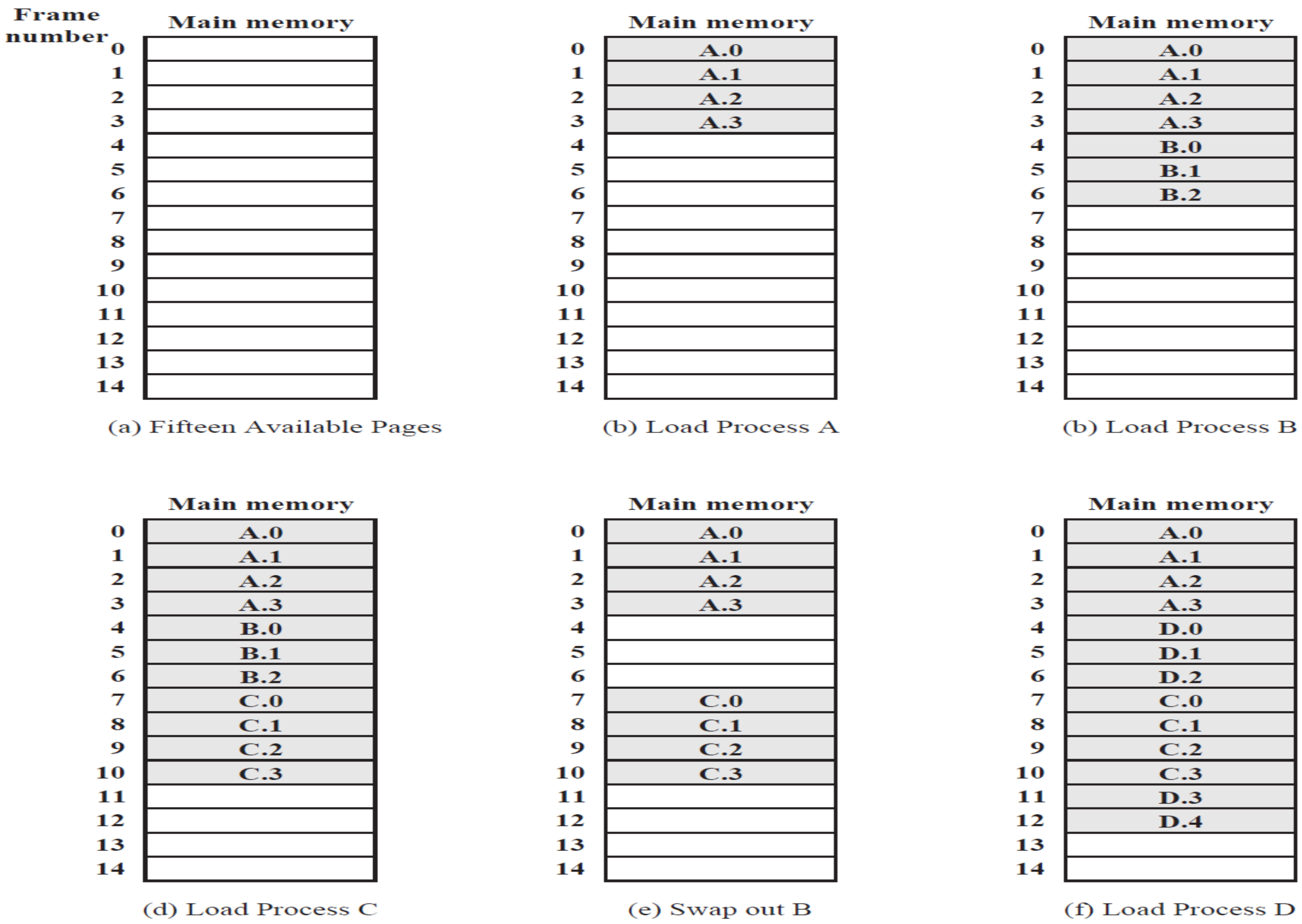


Figure 7.9 Assignment of Process Pages to Free Frames

Page Table Sample

Figure 5-8. Format of a Linear Address

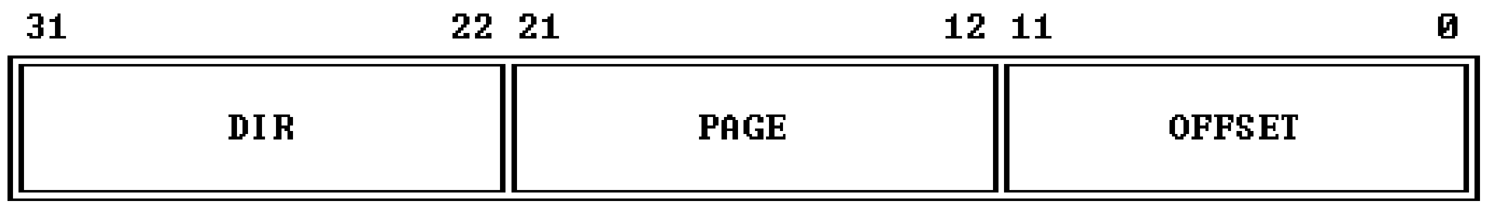
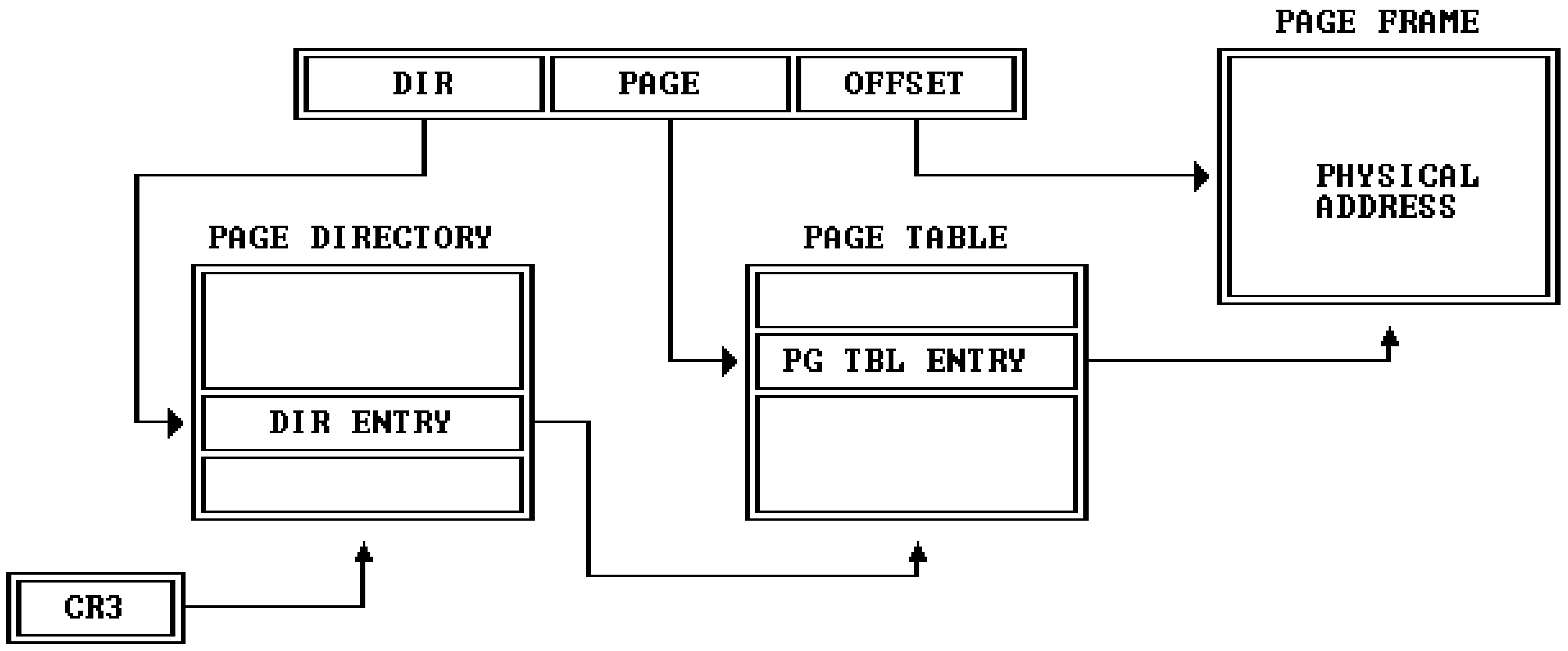


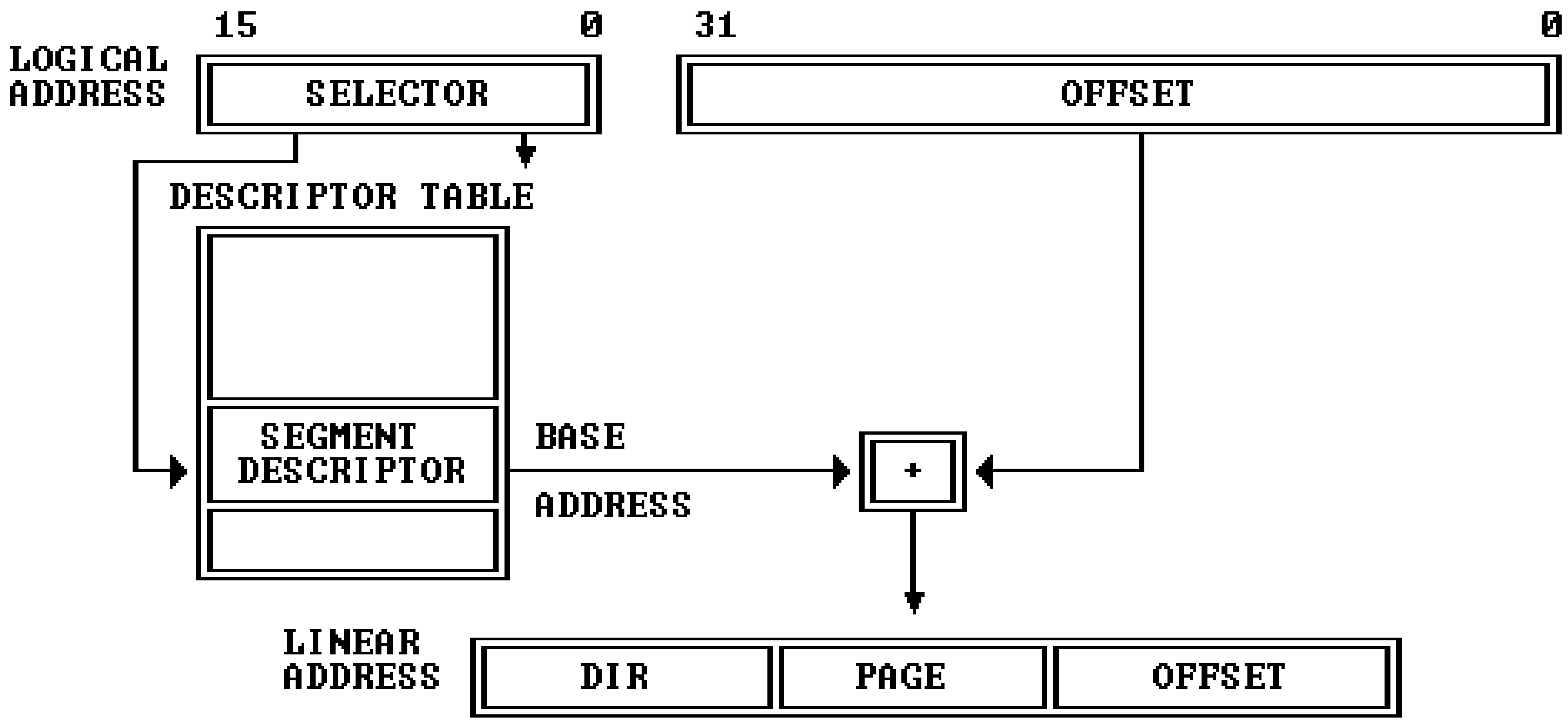
Figure 5-9. Page Translation



SEGMENTATION

- Paging → **internal fragmentation**.
- Segmentation maps segments representing data structures, modules, etc. into **variable partitions.**
- Nor contiguous memory blocks neither all segments of a process are loaded at a time.
- We need a segment table very much like a page table.

Figure 5-2. Segment Translation

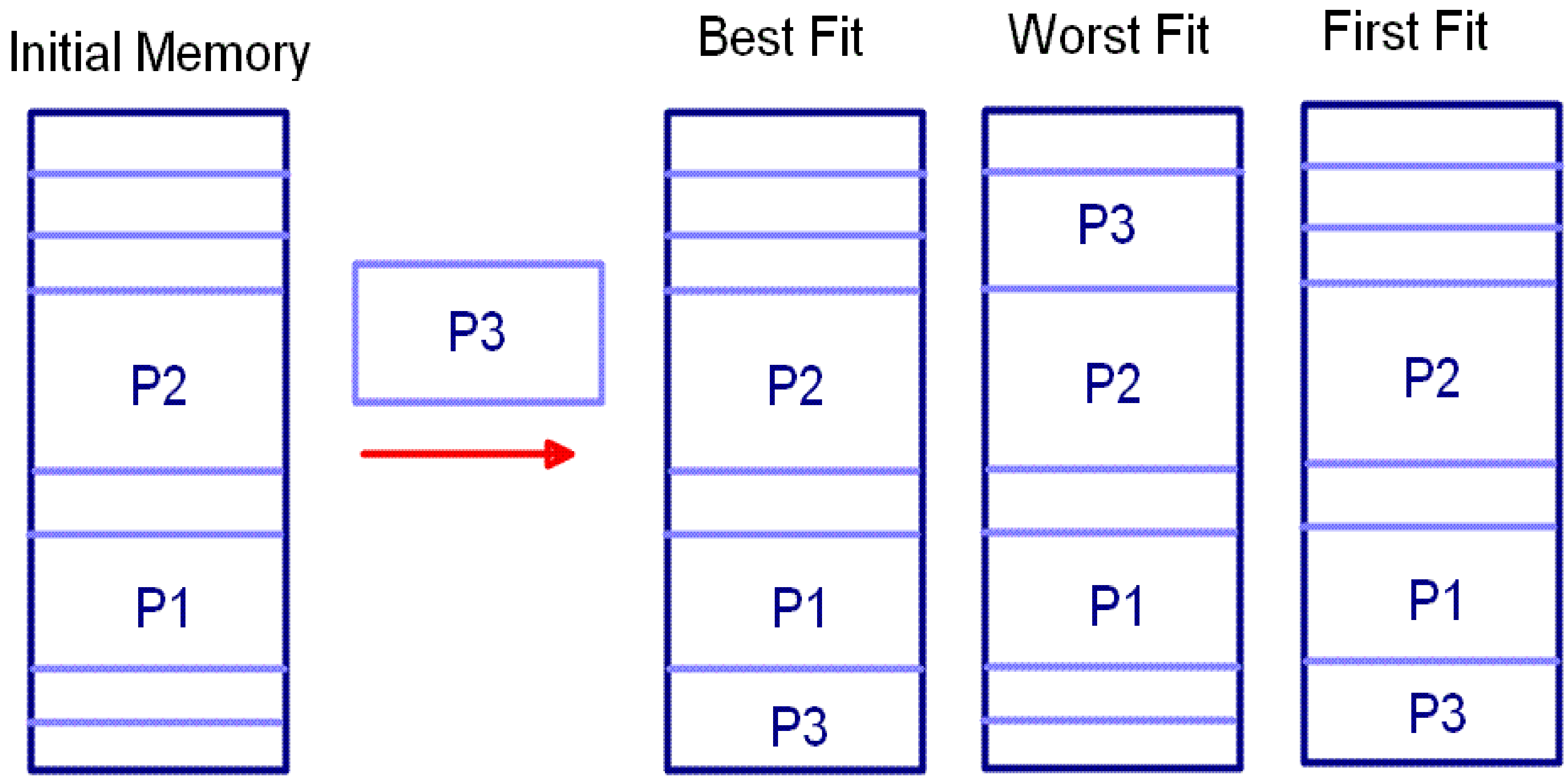


Main Memory Allocation

- Memory is divided into set of contiguous locations called regions/segments/pages
- Store blocks of data
- Placement of blocks of information in memory is called **Memory Allocation**
- **Memory Management Systems** keeps information in a table containing available and free slots

Allocation is done only as per needs

- First Fit
- Best Fit



Replacement Algorithms

- Hardware implemented algorithm (speed)
- Least Recently used (LRU)
  - Pick the slot that hasn't been used in the longest time.
- First in first out (FIFO)
  - replace block that has come into cache first
- Random
- OPT-Optimal(Future)

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

FIFO,LRU,OPT

1) 1 , 6 ,4 , 5 , 1 ,4, 3, 2, 1, 2, 1, 4,6,7,4

FIFO→ 7 – 4 – 6

LRU→ 4 - 6 - 7

OPT→ 7 – 6 -4 (Conflict resolved using LRU)

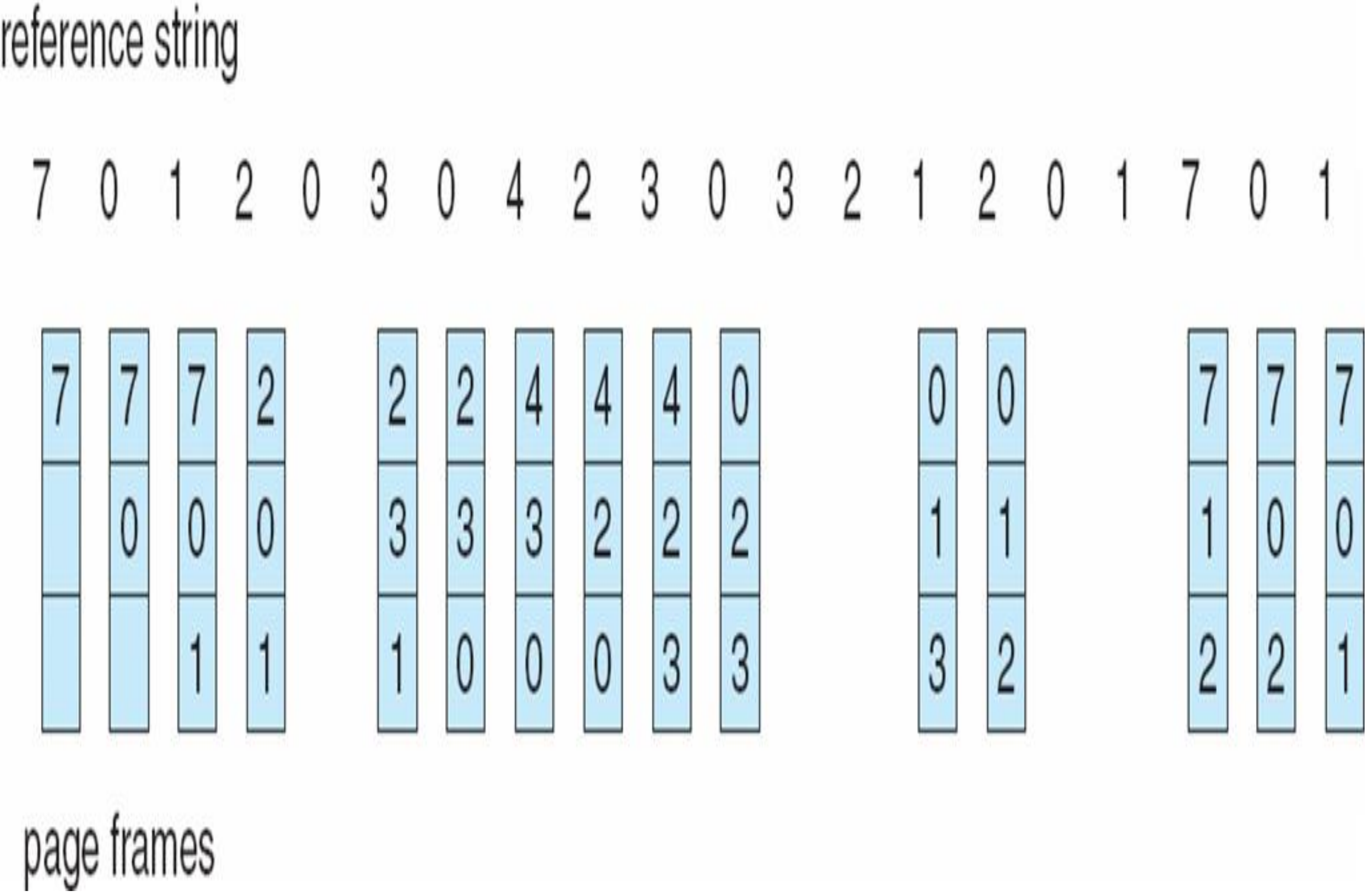
2) 2 , 3 , 2 , 1 , 5 , 2 , 4 , 5 , 3 , 2 , 5 , 2

FIFO→ 3 - 2 - 5

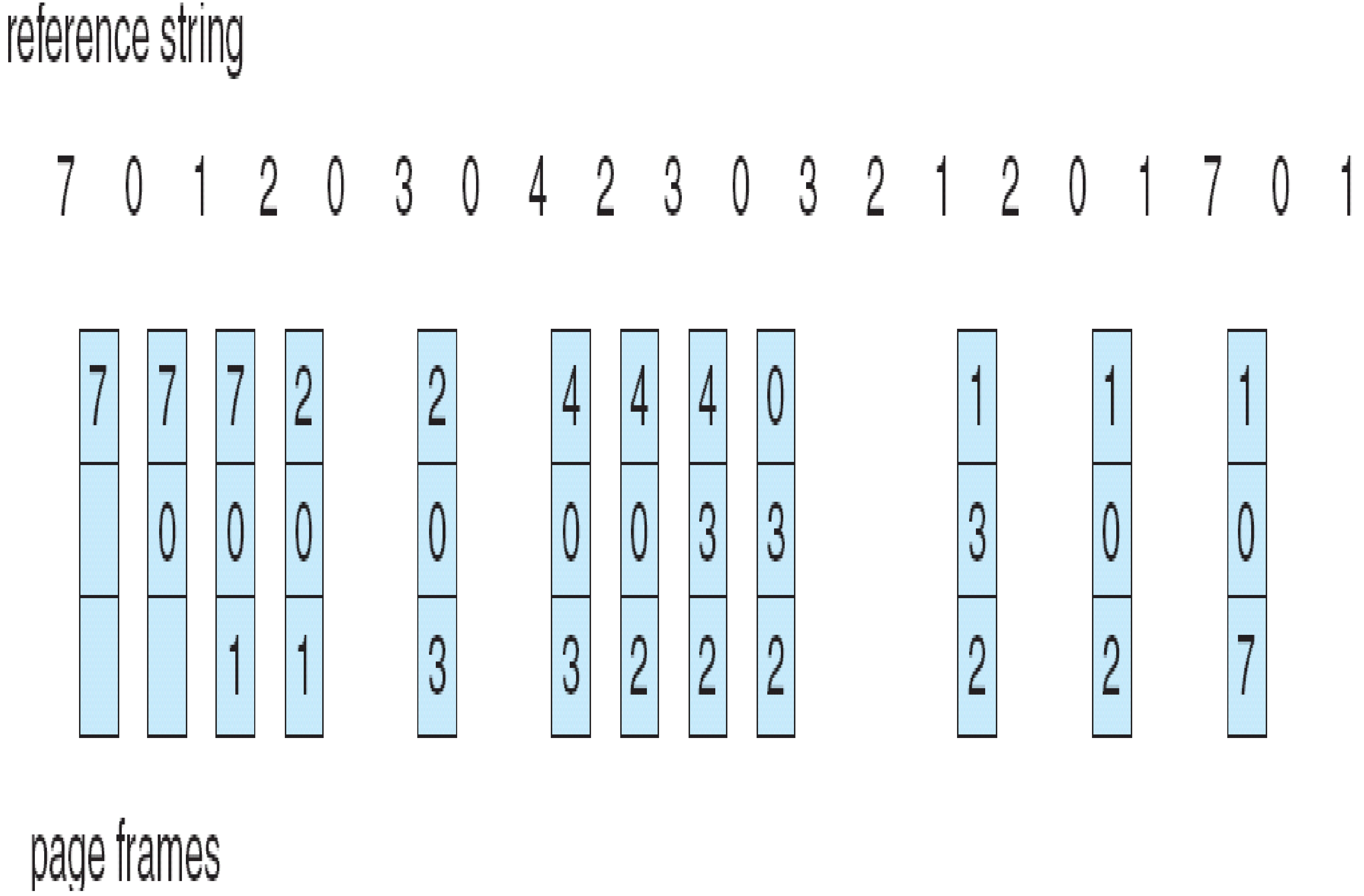
LRU→ 3 – 5 - 2

OPT→ 2 - 3 - 5

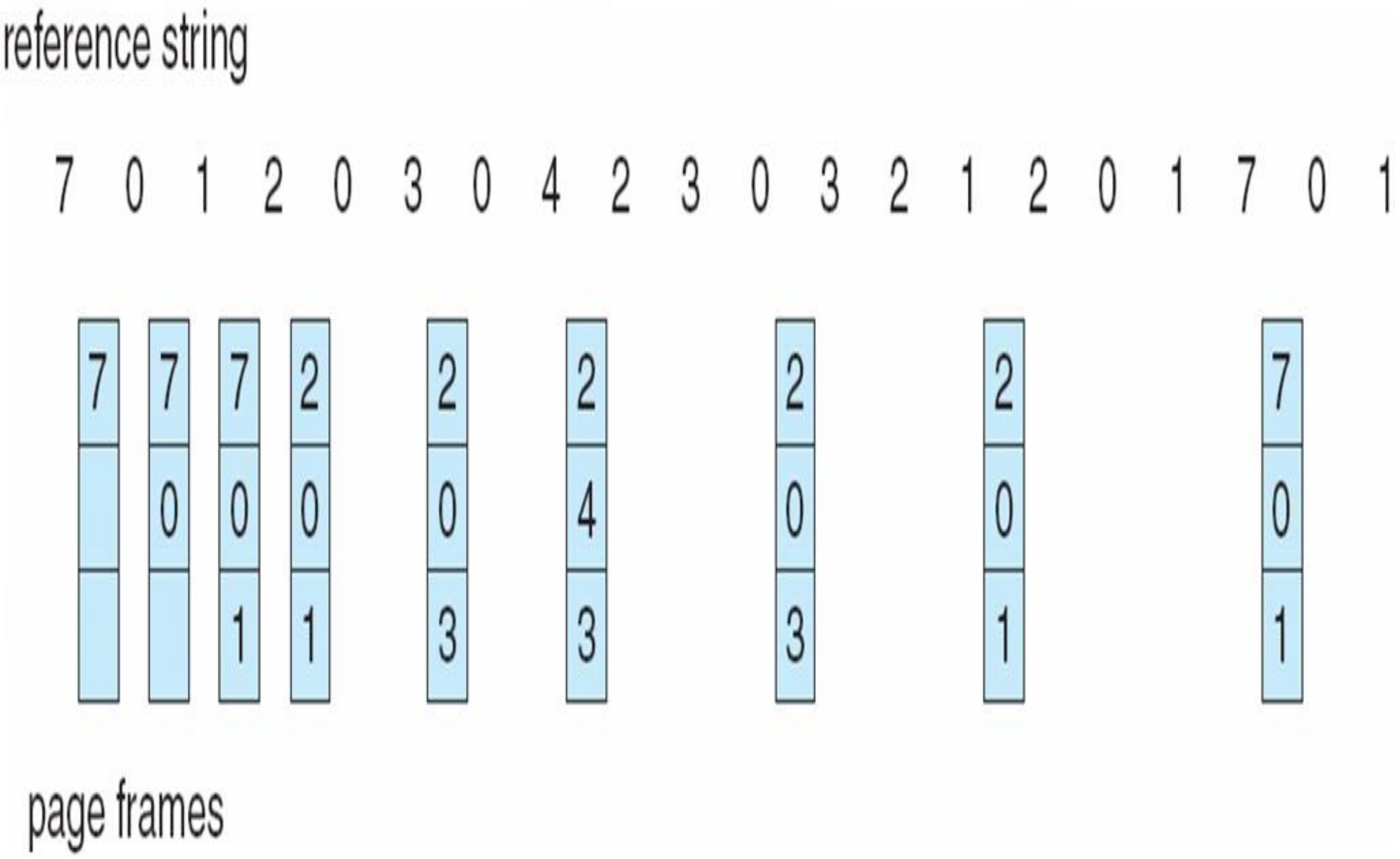
FIFO Page Replacement



LRU Page Replacement



Optimal Page Replacement



Secondary Storage

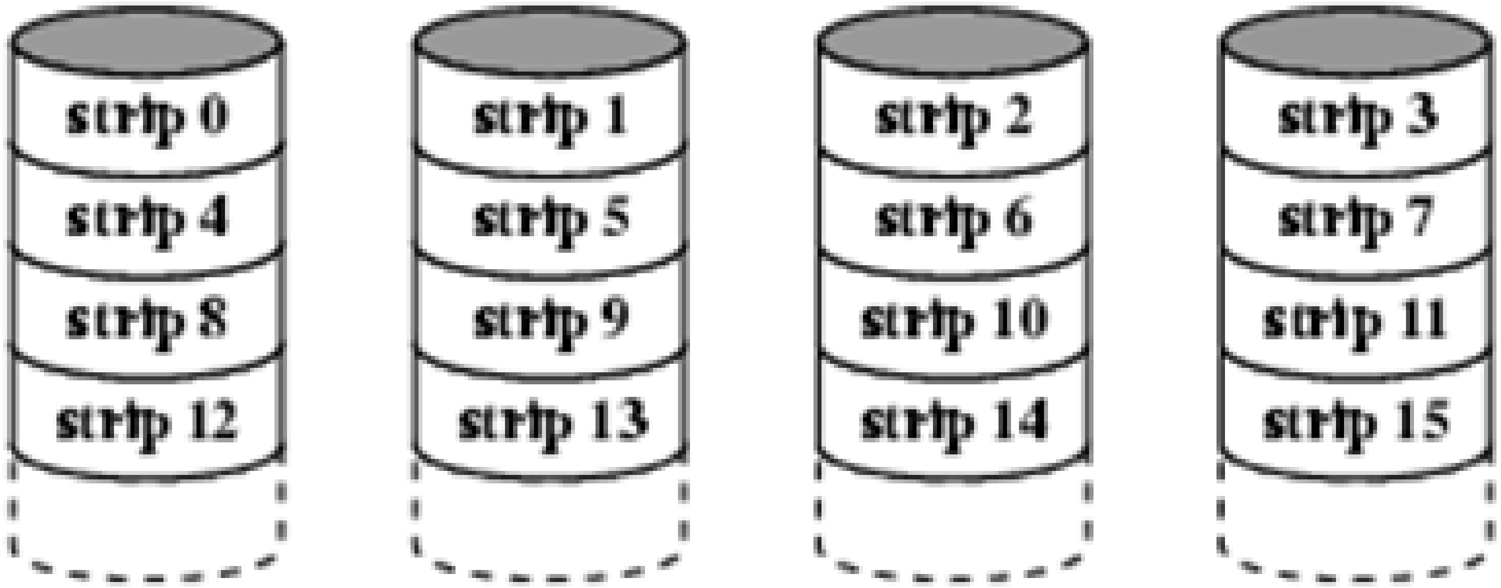
- Magnetic disks
- Floppy disks
- Magnetic Tape
- RAID
- Optical Memory
- CD-ROM
- DVD

RAID Levels 0 - 6

REDUNDANT ARRAY OF INDEPENDENT DISKS

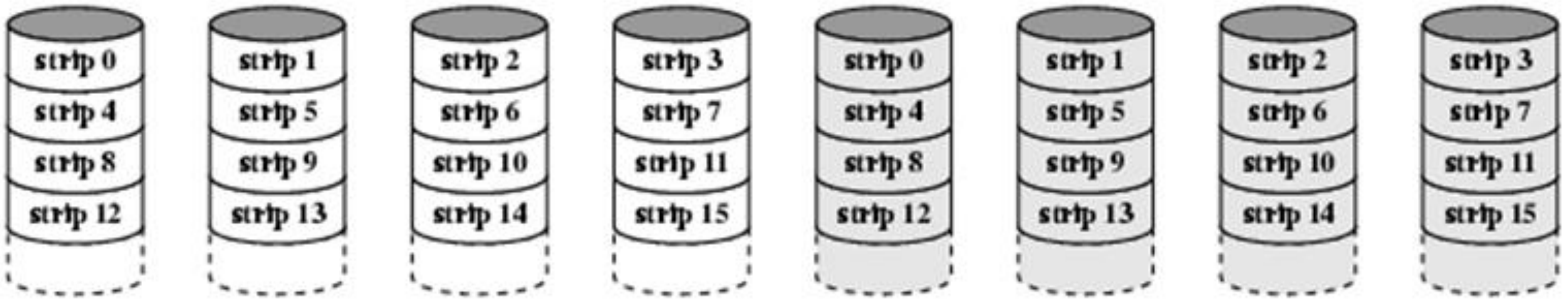
- Storage is an important consideration when setting up a server.
- Almost all of the important information that you and your users car about will at one point be written to a storage device to save for later retrieval.
- Single disks can serve you well if your needs are straight forward.
- However, if you have more complex redundancy or performance requirements, solutions like RAID can be helpful.

RAID Level 0- Non Redundant



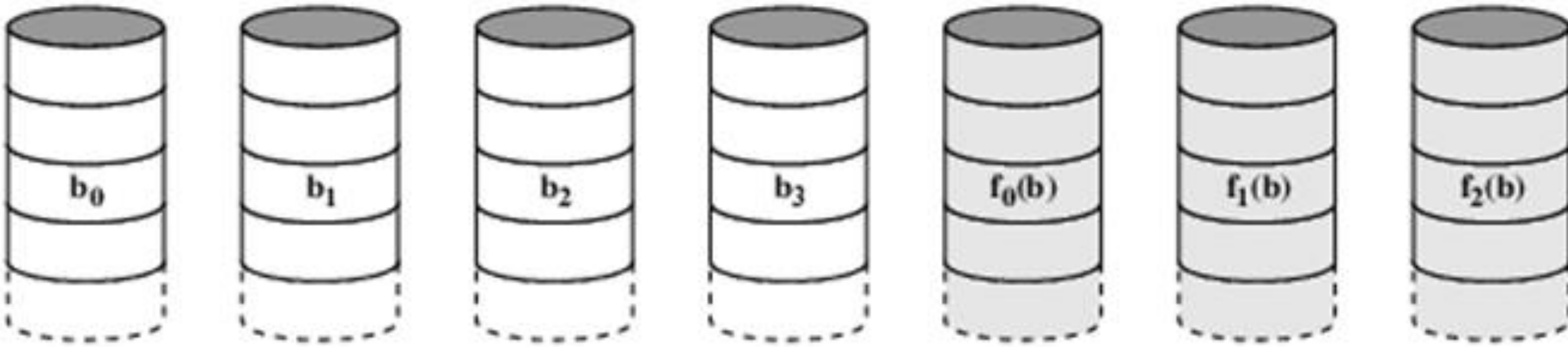
(a) RAID 0 (non-redundant)

RAID Level -1 Mirrored



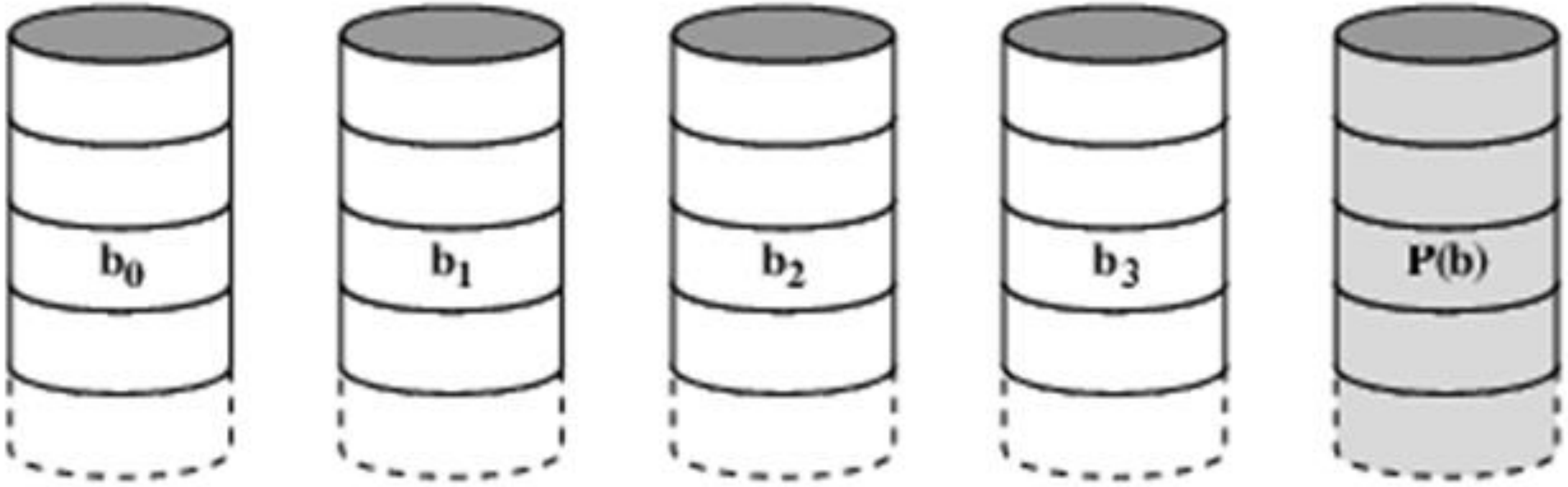
(b) RAID 1 (mirrored)

RAID Level 2- Hamming Code



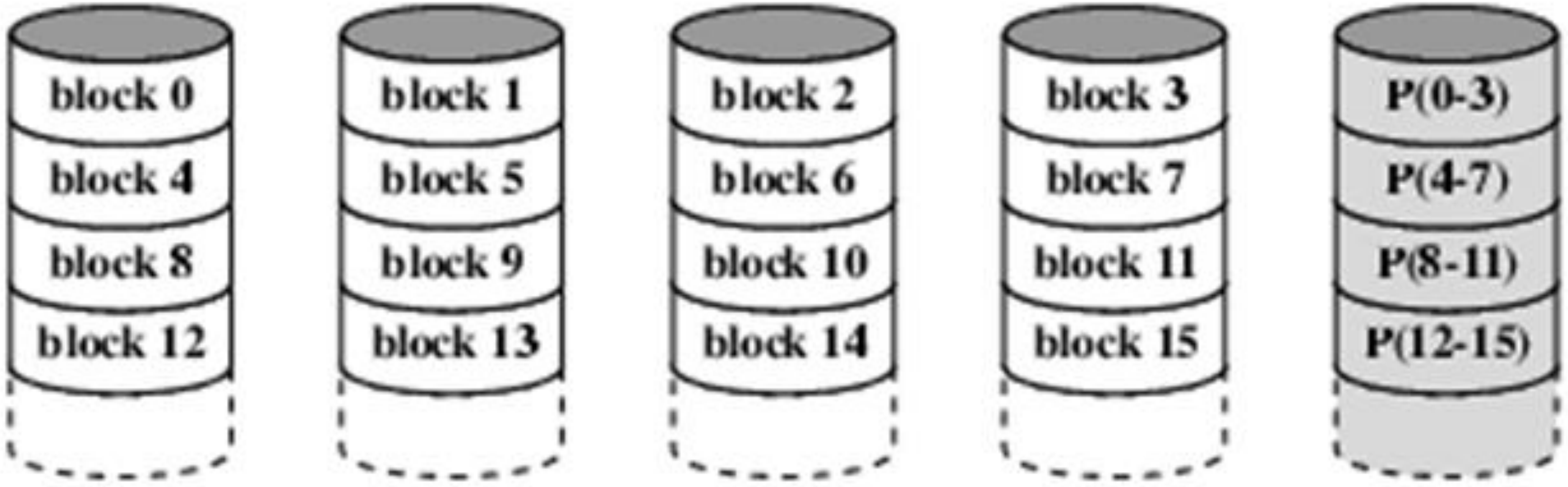
(c) RAID 2 (redundancy through Hamming code)

RAID Level 3 –Bit Interleaved Parity



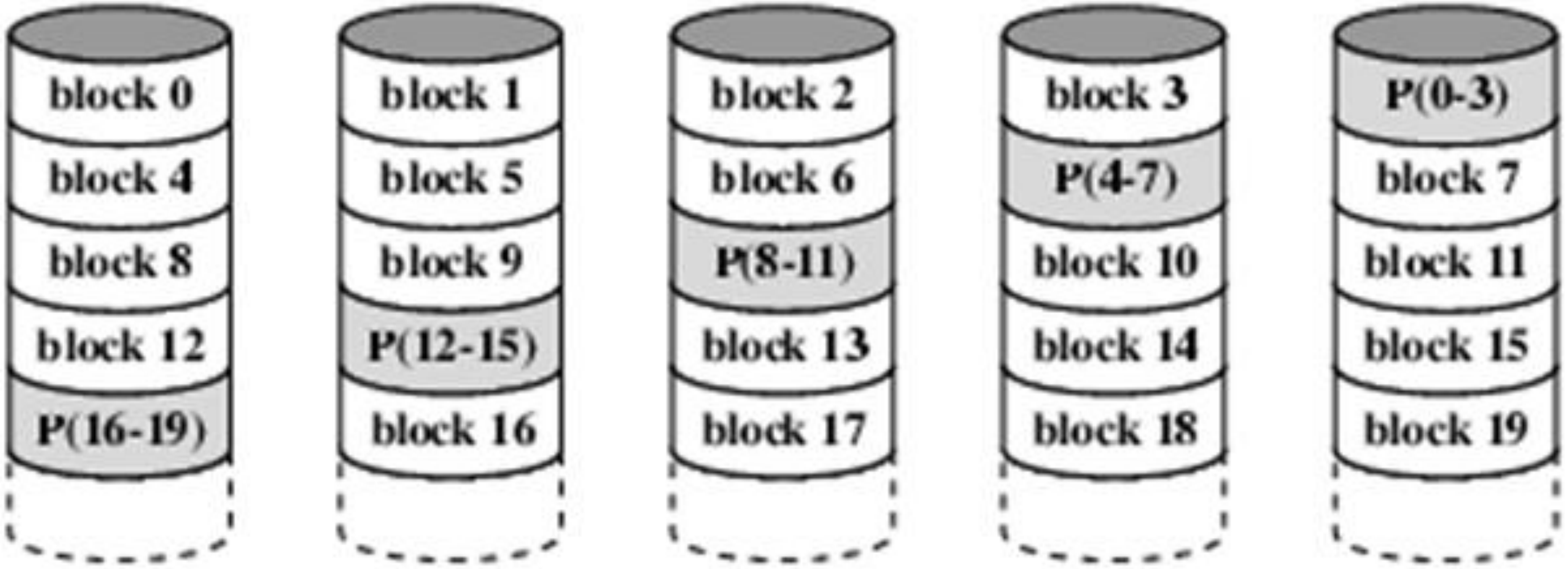
(d) RAID 3 (bit-interleaved parity)

RAID Level 4- Block level parity



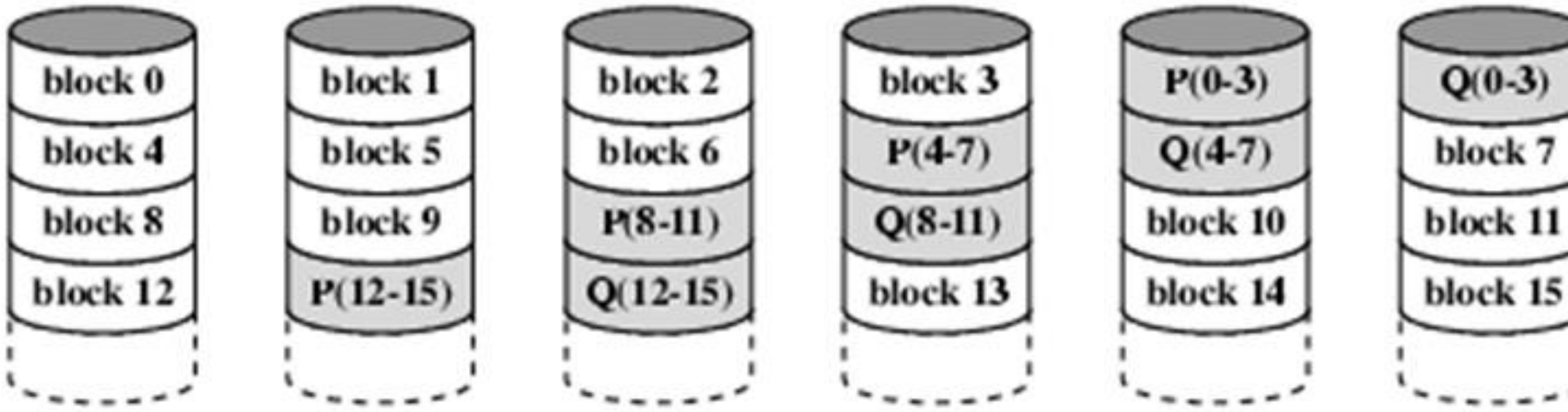
(e) RAID 4 (block-level parity)

RAID Level 5- Block level Distributed Parity



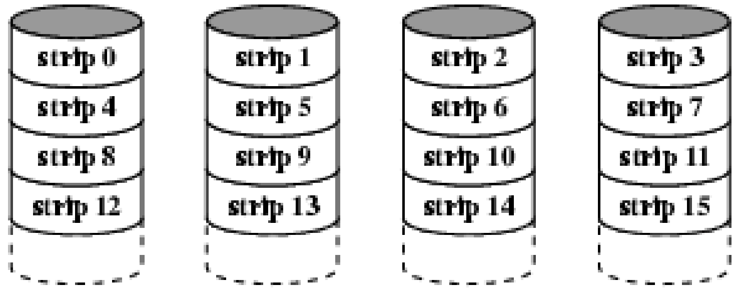
(f) RAID 5 (block-level distributed parity)

RAID Level 6- Dual Redundancy

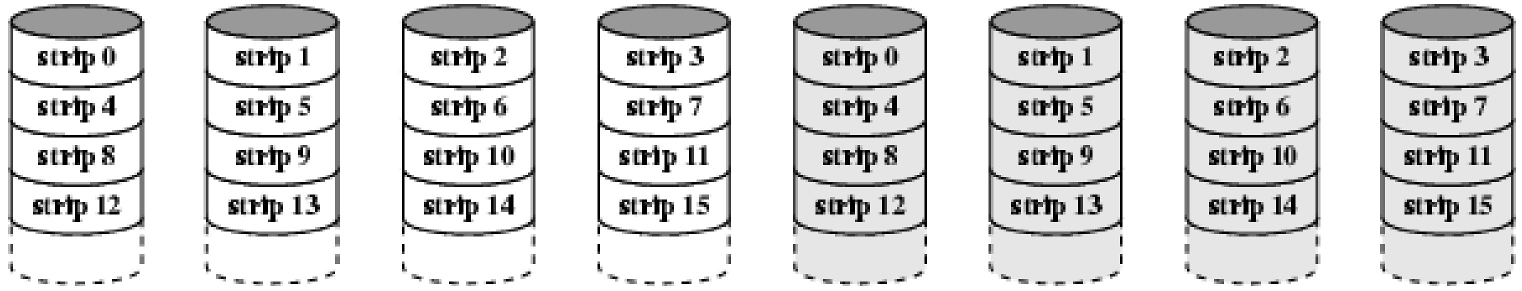


(g) RAID 6 (dual redundancy)

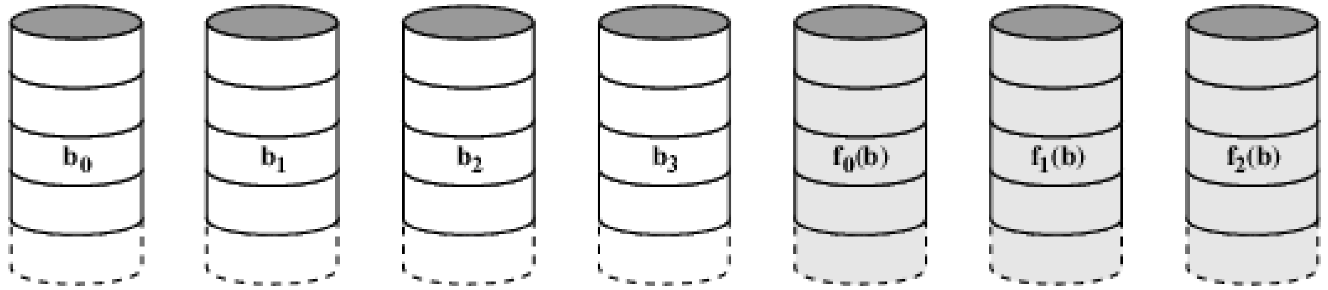
RAID 0, 1, 2 – Redundant Array of Independent Disks



(a) RAID 0 (non-redundant)

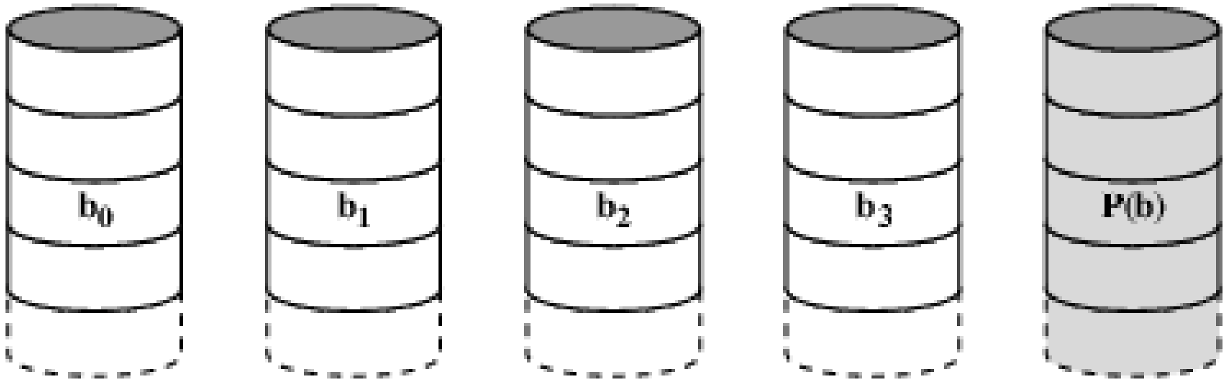


(b) RAID 1 (mirrored)

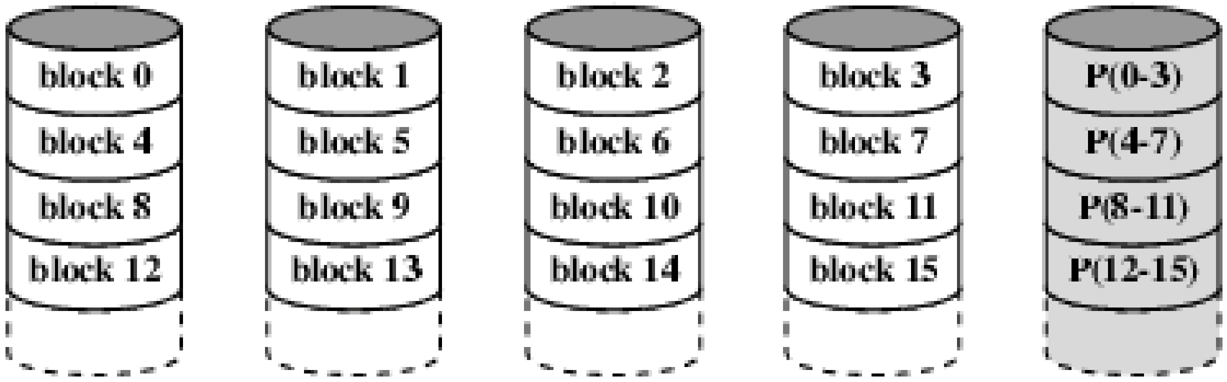


(c) RAID 2 (redundancy through Hamming code)

RAID 3 & 4

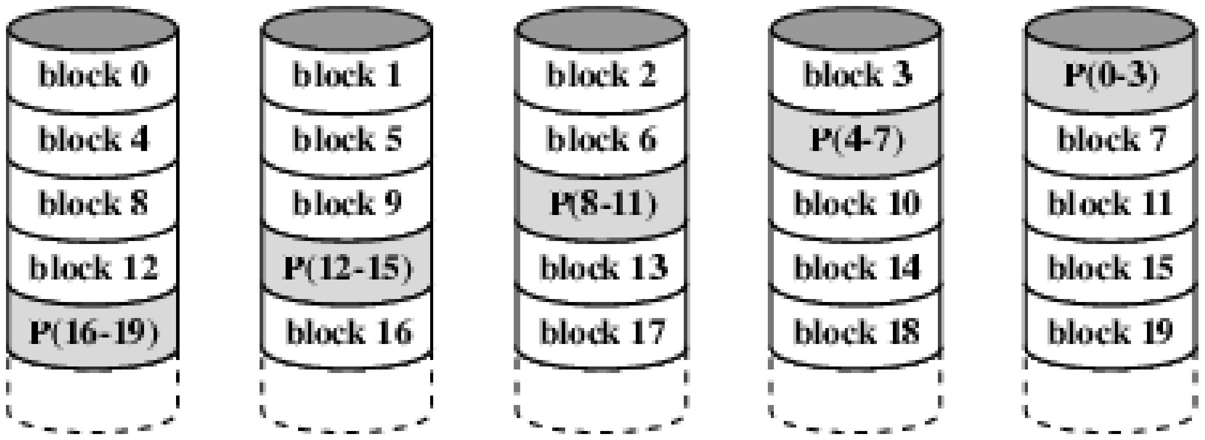


(d) RAID 3 (bit-interleaved parity)

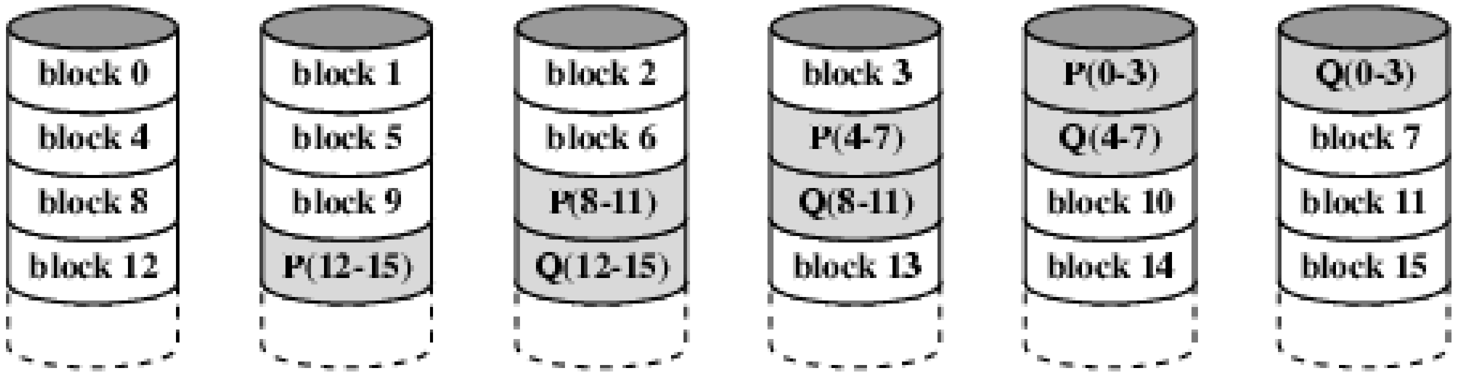


(e) RAID 4 (block-level parity)

RAID 5 & 6



(f) RAID 5 (block-level distributed parity)



(g) RAID 6 (dual redundancy)

Solve using FIFO , LRU and OPT page replacement algorithms

- Given page reference string: 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6
- FIFO - 6 1 3
- LRU - 2 3 6
- OPT - 6 2 3