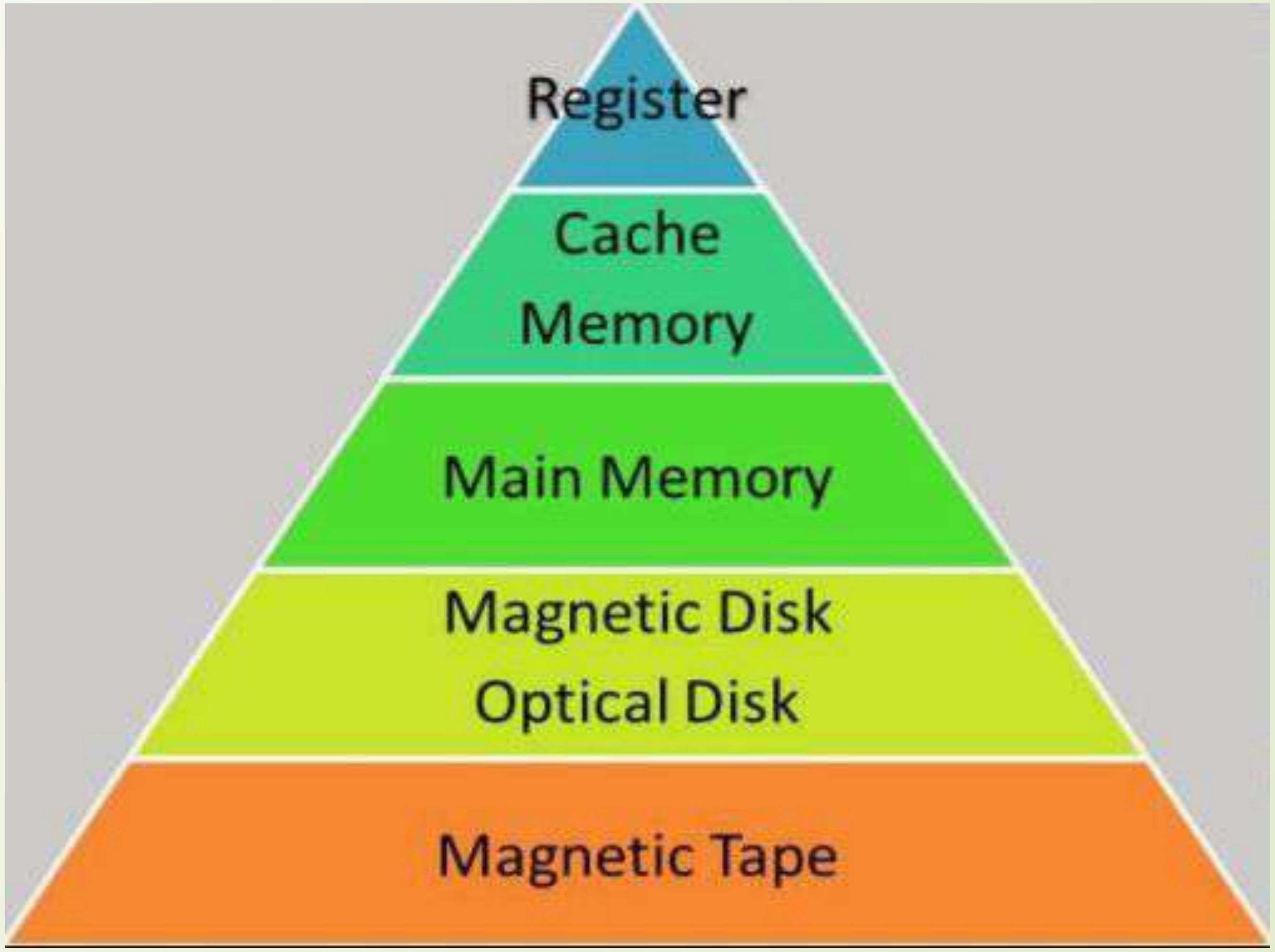
CHAPTER-7

Hierarchical memory system



Contd…

- The design constraints on computer memory depends on:
  - capacity
  - speed / access time
  - cost
- However, there is a trade-off among these three characteristics and the following relationship is observed.
  => lesser access time, greater cost per bit.
  => greater capacity, smaller cost per bit.
  => greater capacity, greater access time.
- Desire=> greater capacity, faster access time, low cost per bit.
- Cannot be achieved with single memory component, solution is to organize computer memory into hierarchy.
- While moving down the hierarchy, we can observe the following:
  => Increasing capacity.
  => Decreasing cost per bit.
  => Increasing access time / Decreasing speed.

Associative memory(Content Addressable Memory)

- This type of memory is accessed simultaneously & in parallel on the basis of data content rather than by specific address or location
- When a word is written in an associative memory, no address is given. The CAM checks for location whose valid bit is 0; stores data in that location, sets valid bit to 1.
  => if no location found, clears location(some algorithm) to store data.
- When a word is to be read from an associative memory, the content of word or part of word is specified. The CAM locates all words which match the specified content and mark them for reading.
- Uniquely suitable for parallel searches.
- More expensive than RAM. Therefore, CAM are used in application where search time is very critical and must be very short.

Contd…

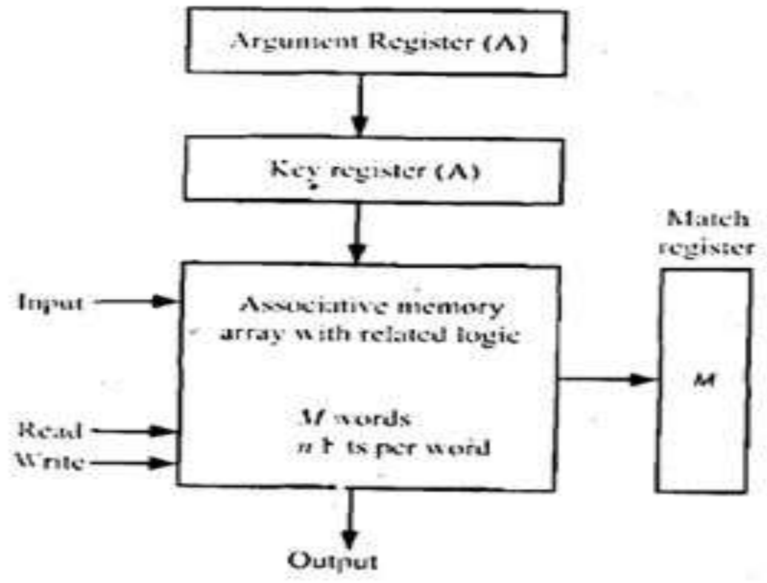


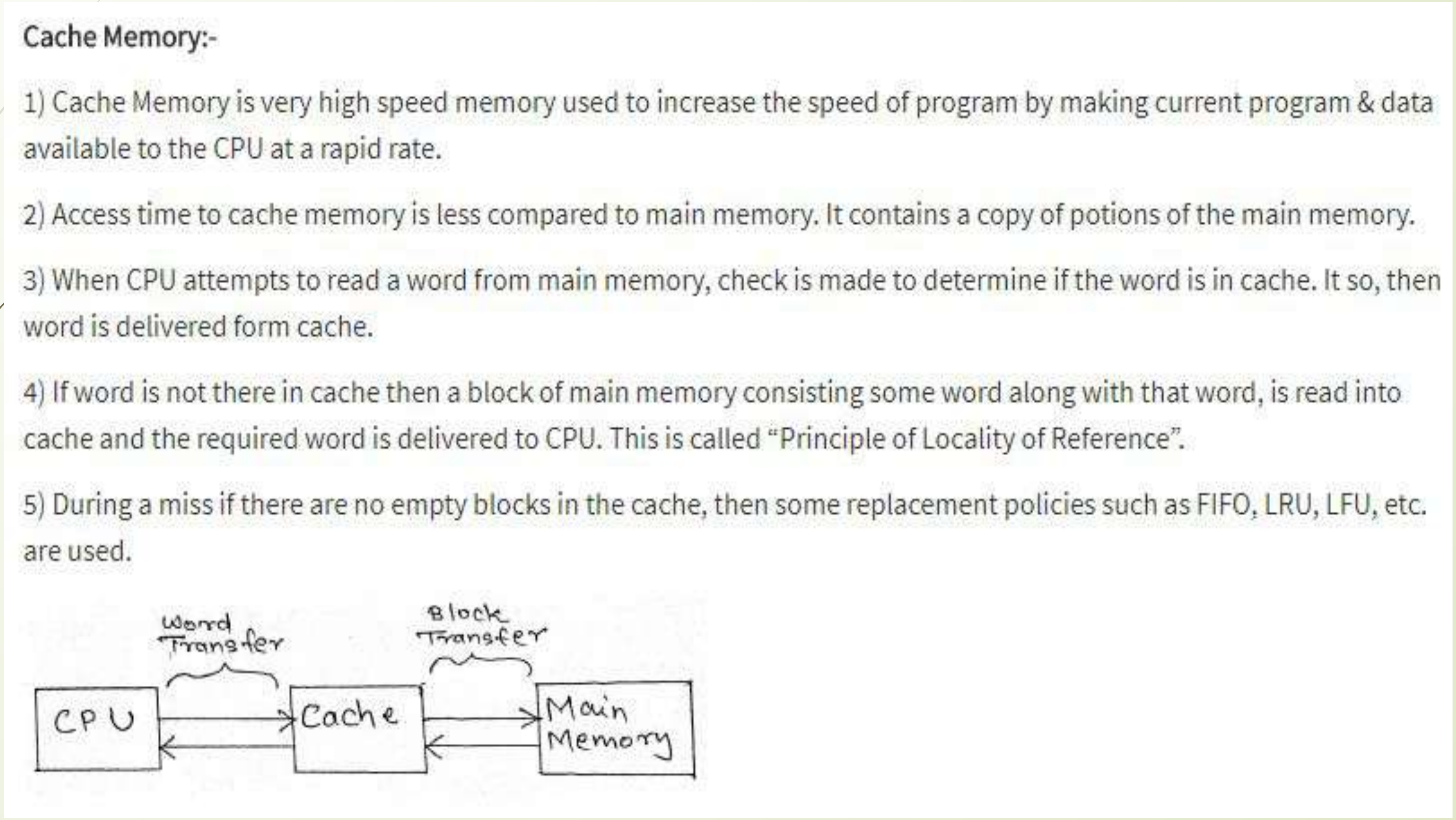
Figure: Associative Memory - Block Diagram

To explain with a numerical illustration assume that argument register A and key register K have the bit configuration displayed below. Only three leftmost bits of a compared with memory words since K has 1's on these positions.

A	101 111100	
K	111 000000	
Word 1	100 111100	no match
Word 2	101 000001	match

Word 2 matches unmasked argument field since the three leftmost bits of argument and word are equal.

Cache memory:



Cache Memory:-

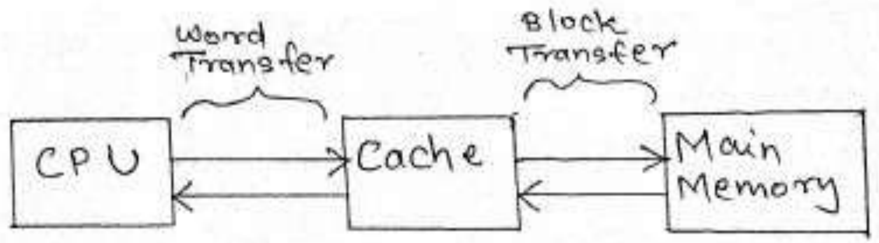
1) Cache Memory is very high speed memory used to increase the speed of program by making current program & data available to the CPU at a rapid rate.

2) Access time to cache memory is less compared to main memory. It contains a copy of potions of the main memory.

3) When CPU attempts to read a word from main memory, check is made to determine if the word is in cache. It so, then word is delivered form cache.

4) If word is not there in cache then a block of main memory consisting some word along with that word, is read into cache and the required word is delivered to CPU. This is called "Principle of Locality of Reference".

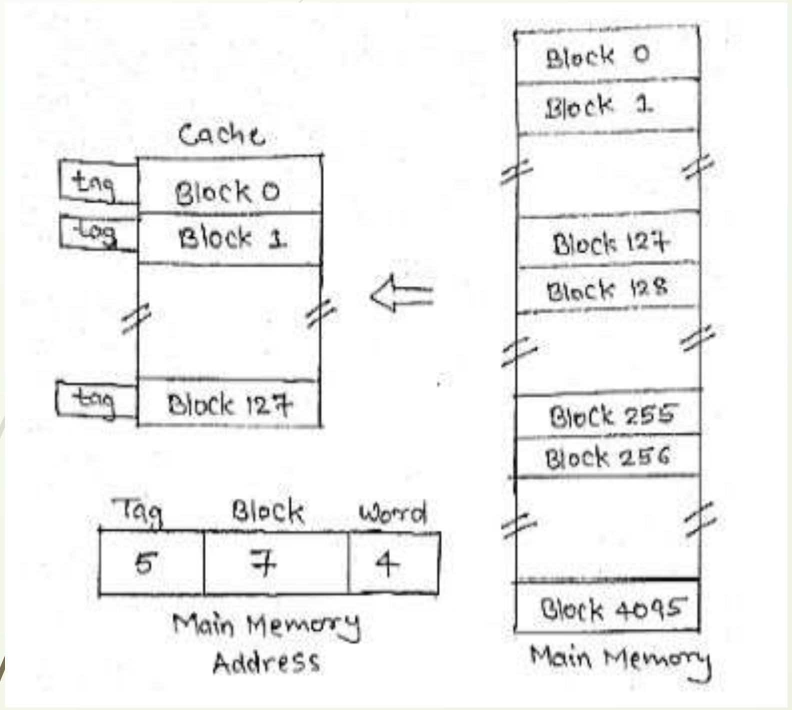
5) During a miss if there are no empty blocks in the cache, then some replacement policies such as FIFO, LRU, LFU, etc. are used.

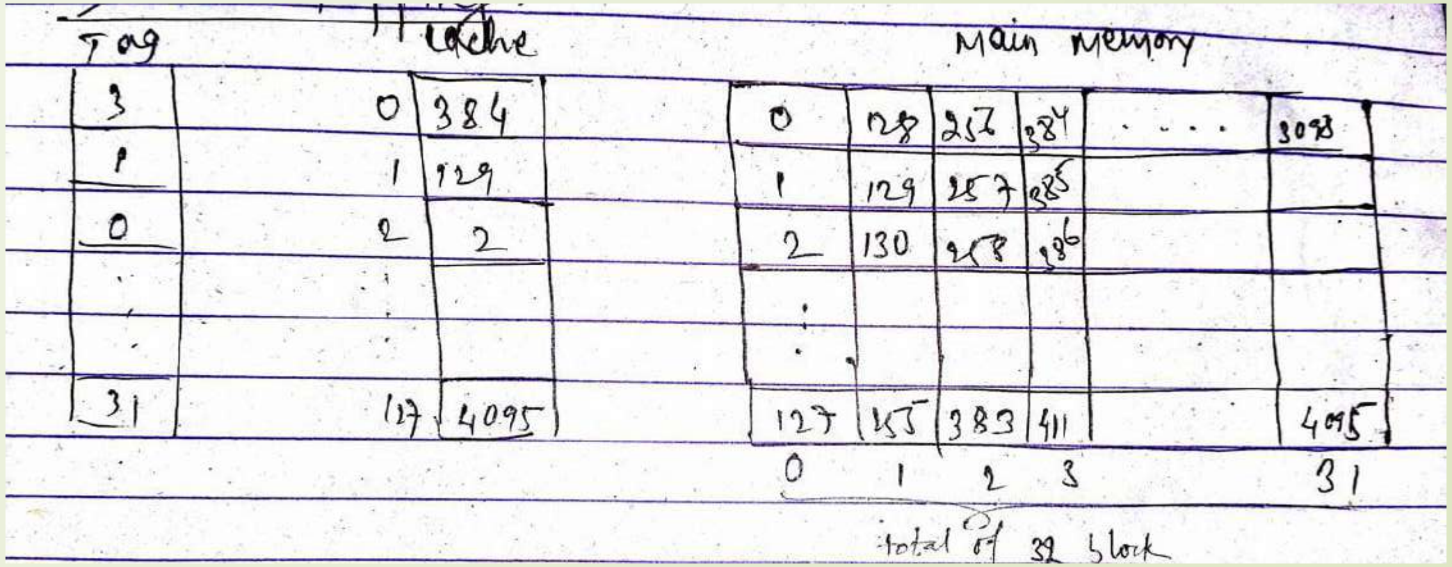


Cache Mapping Techniques:

- The different cache mapping techniques are as follows:
  => Direct Mapping
  => Associative mapping
  => Set Associative mapping
- Consider 64K main memory which will be viewed as 4K blocks of 16 word each.
  i.e. 64K = $2^6 \times 2^{10} = 2^{16}$ = 16 – bit address
- Consider a cache consisting of 128 blocks of 16 words each, for total of 2048(2K) words.

Direct Mapping





Contd…

- Mapping function: i = j % m

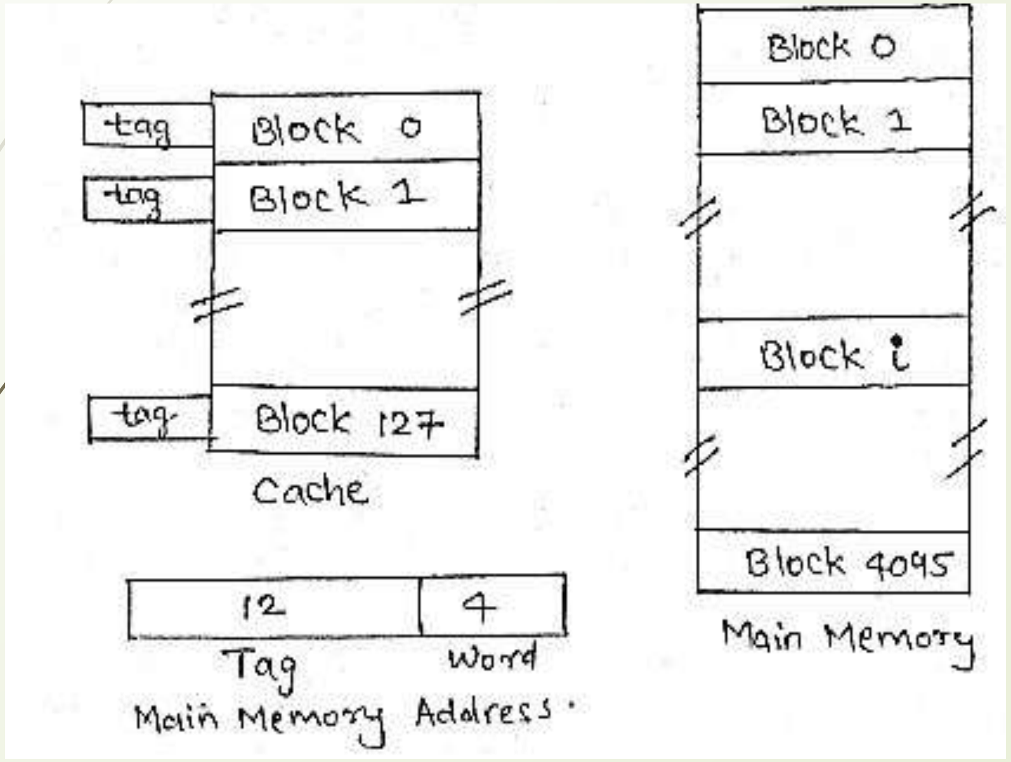
  j = main memory block number

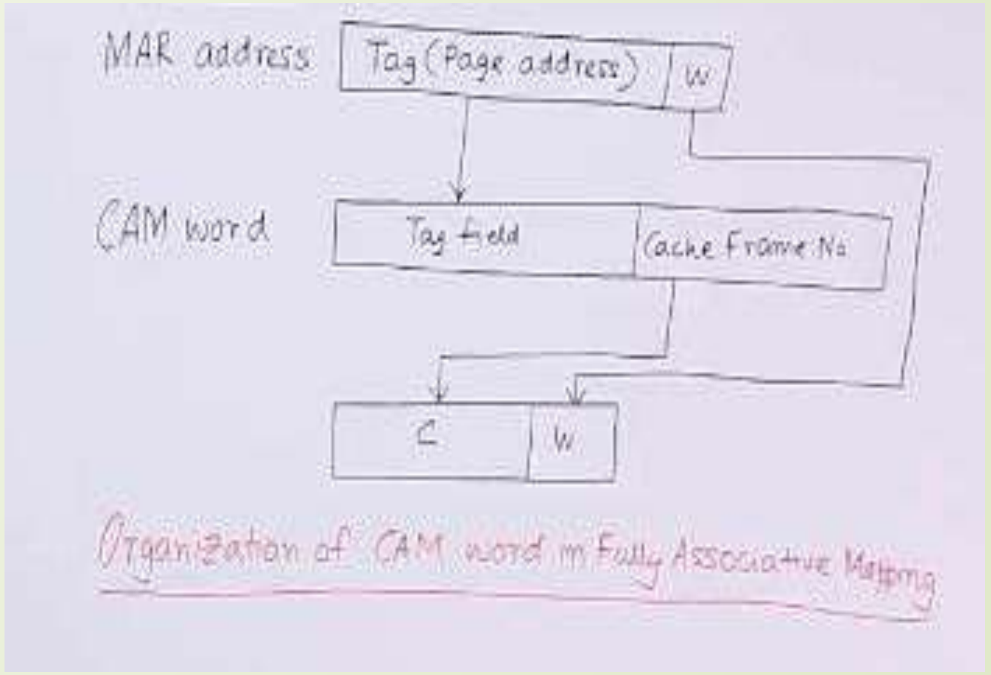
  m = number of cache lines

  i = cache line number

  i.e. main memory blocks 0, 128, 256,… is loaded into cache and stored at block 0. Block 1, 129, 257,…are stored at block 1 and so on.
- Lower 4-bits select one of the 16 words in a block.
- The 7-bit cache block determines the cache line in which the block must be stored.
- The leftmost 5-bit is used to identify one of 32 memory block when it is resident in cache.

Associative mapping/ Fully associative mapping:





Contd…

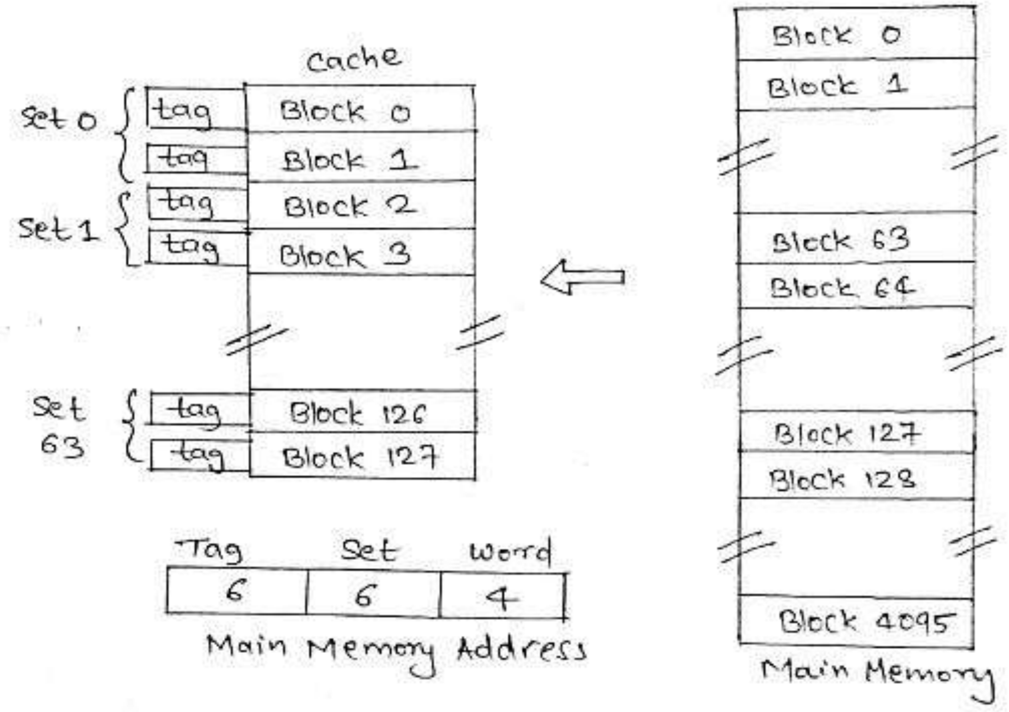
- Flexible mapping method, in which main memory block can be placed into any cache block position.
- In this, 12 tag bits are required to identify a memory block when it is resident in the cache.
- CAM contains so many blocks with their respective cache location. It takes 12-bit tag(fig) from the memory address to find whether a particular block is in memory.
- The searching is done in all blocks within CAM and in parallel.
- Cost of associated mapped cache is higher than the cost of direct mapped because of the need to search all 128 tag pattern to determine whether a block is in cache. This is known as associative search.

Set associative mapping:

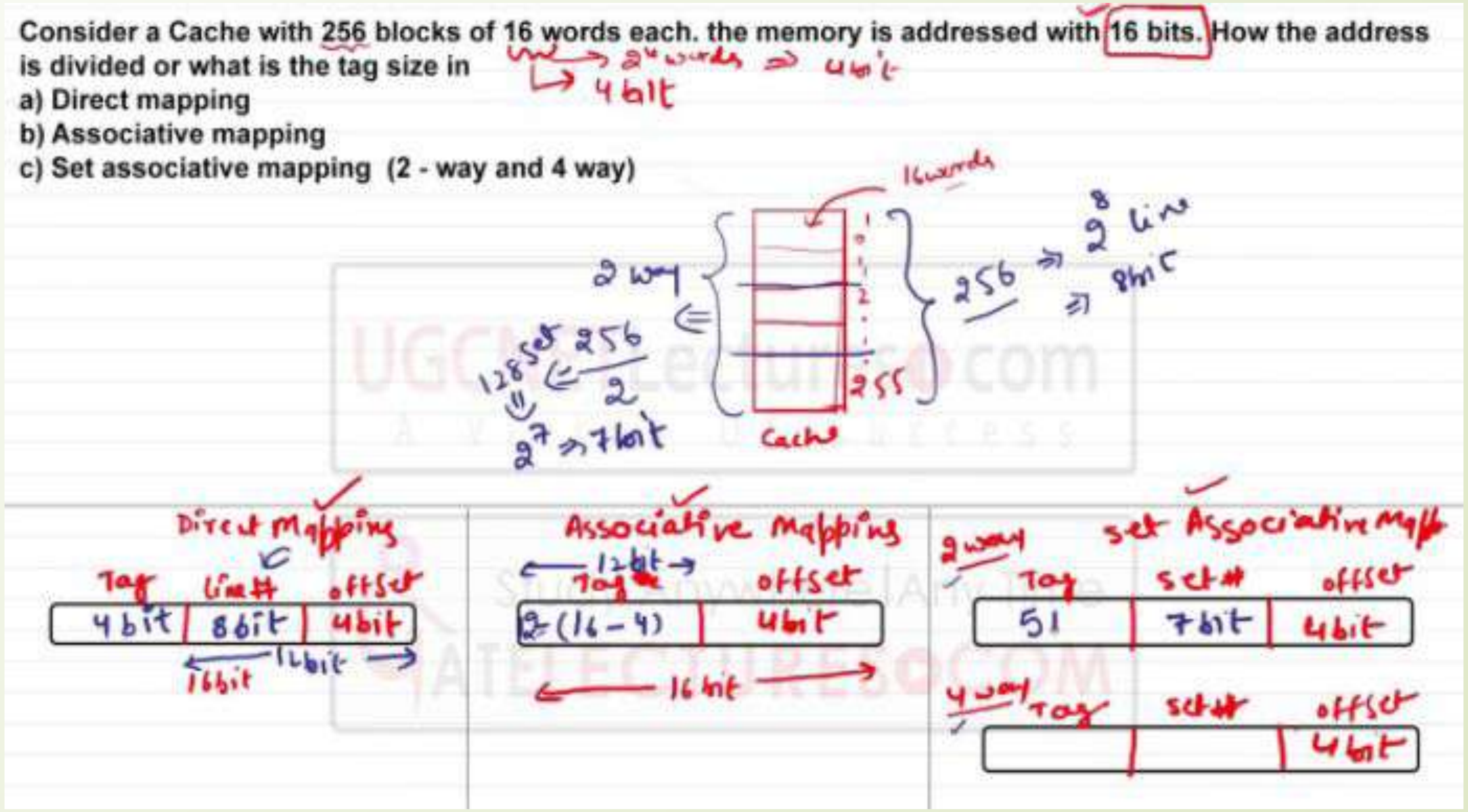
=> It is the combination of direct and associative mapping technique.

●) For a cache with two blocks per set. In this case, memory block 0, 64, 128,…..,4032 map into cache set 0 and they can occupy any two block within this set.

●) Having 64 sets means that the 6 bit set field of the address determines which set of the cache might contain the desired block. The tag bits of address must be associatively compared to the tags of the two blocks of the set to check if desired block is present. This is two way associative search.



Example



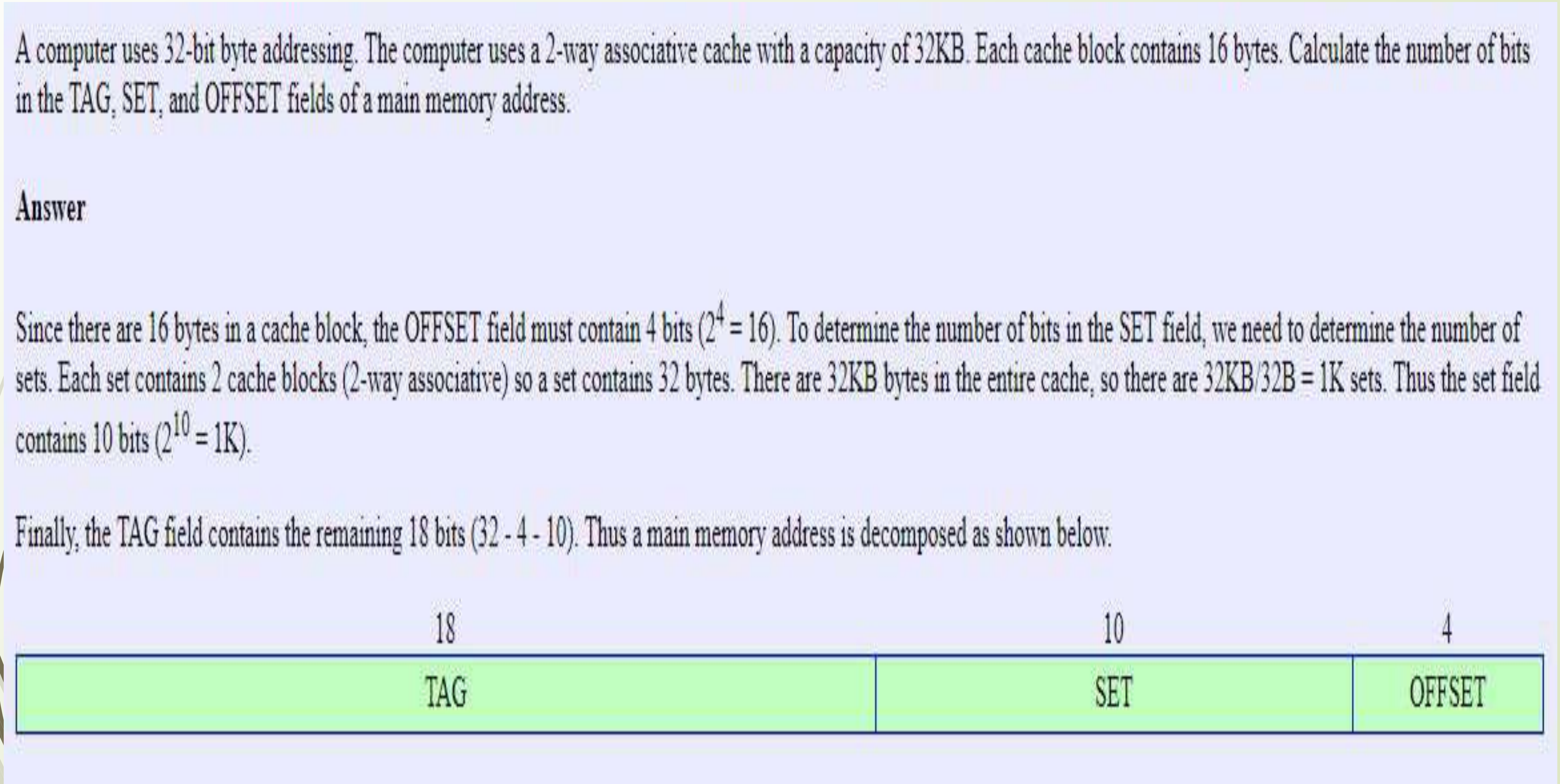
Consider a Cache with 256 blocks of 16 words each. the memory is addressed with 16 bits. How the address is divided or what is the tag size in

a) Direct mapping

b) Associative mapping

c) Set associative mapping (2 - way and 4 way)

Example contd…



A computer uses 32-bit byte addressing. The computer uses a 2-way associative cache with a capacity of 32KB. Each cache block contains 16 bytes. Calculate the number of bits in the TAG, SET, and OFFSET fields of a main memory address.

Answer

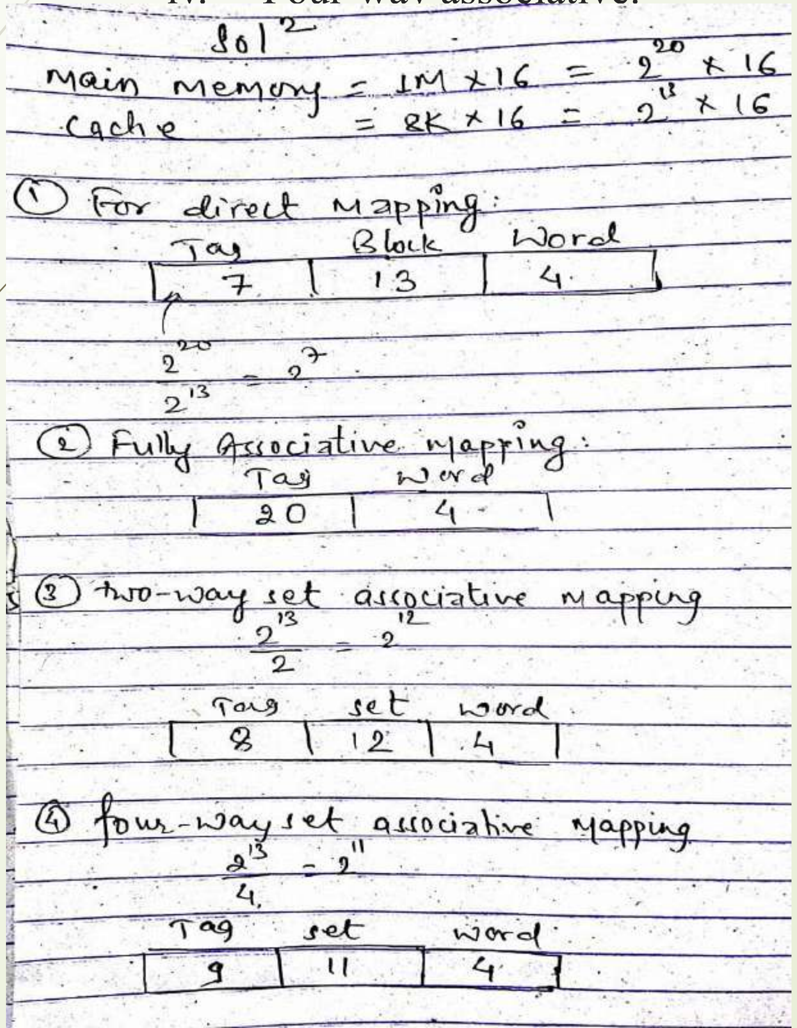
Since there are 16 bytes in a cache block, the OFFSET field must contain 4 bits ($2^4 = 16$). To determine the number of bits in the SET field, we need to determine the number of sets. Each set contains 2 cache blocks (2-way associative) so a set contains 32 bytes. There are 32KB bytes in the entire cache, so there are 32KB/32B = 1K sets. Thus the set field contains 10 bits ($2^{10} = 1K$).

Finally, the TAG field contains the remaining 18 bits (32 - 4 - 10). Thus a main memory address is decomposed as shown below.

18	10	4
TAG	SET	OFFSET

Q. Show the layout of the cache for a CPU that can address 1M x 16 of memory; the cache holds 8K x 16 of data and has the following mapping strategies. Given the number of bits per location and total number of locations as well.

i. Fully Associative
ii. Direct Mapped
iii Two-way associative.
iv. Four-wav associative.



Sol^{2}

Main memory = 1M × 16 = $2^{20} \times 16$
Cache = 8K × 16 = $2^{13} \times 16$

① For direct mapping:

Tag	Block	Word
7	13	4

$\frac{2^{20}}{2^{13}} = 2^{7}$

② Fully Associative mapping:

Tag	Word
20	4

③ two-way set associative mapping

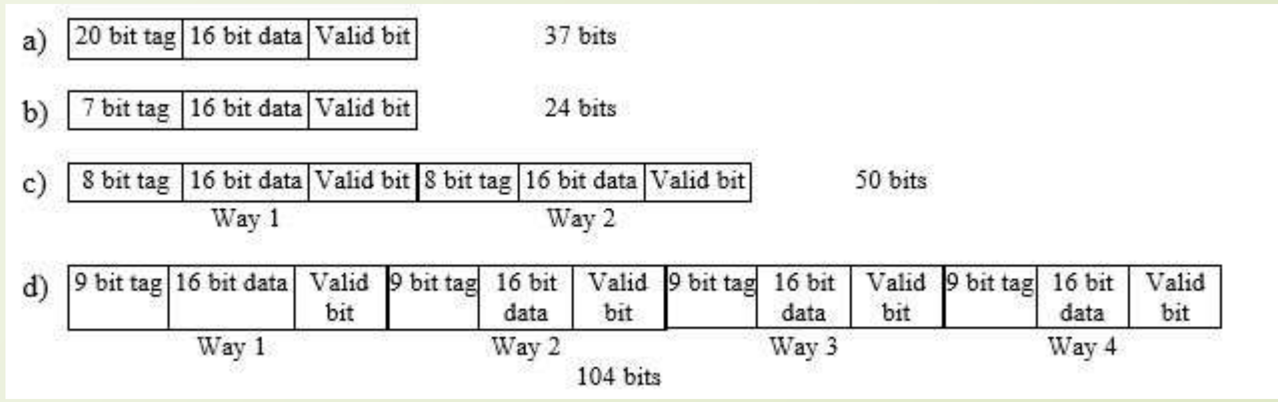
$\frac{2^{13}}{2} = 2^{12}$

Tag	set	word
8	12	4

④ four-way set associative mapping

$\frac{2^{13}}{4} = 2^{11}$

Tag	set	word
9	11	4



a)

20 bit tag	16 bit data	Valid bit	37 bits

b)

7 bit tag	16 bit data	Valid bit	24 bits

c)

8 bit tag	16 bit data	Valid bit	8 bit tag	16 bit data	Valid bit	50 bits
	Way 1			Way 2		

d)

9 bit tag	16 bit data	Valid bit	9 bit tag	16 bit data	Valid bit	9 bit tag	16 bit data	Valid bit	9 bit tag	16 bit data	Valid bit
	Way 1			Way 2			Way 3			Way 4	

104 bits

Replacing data in the cache:

- When a computer begins to execute program, it fetches instructions and data from memory, it loads these values into cache.(when particular data is not in cache)
- If the computer needs to move data into cache locations that are already occupied, problem becomes deciding which data to move out of cache and how to preserve that data in physical memory.
- Direct mapping offers easiest solution to these problem. Since each address in physical memory maps to one specific location in cache, there is no choice to be made about which value will be replaced.
- The associative cache allows any location in physical memory to be mapped to any location in the cache. Some replacement policies to select are:
  => FIFO
  => LRU
  => Random
- In set-associative cache, each location in physical memory is mapped to one specific location of cache; however, a replacement strategy is needed to determine which way within the location to use.

Writing data to the cache

Two methodologies can be used to write data to the cache.

- In **Write-through,** every time value is written from the CPU into a location in the cache, it is also written to the corresponding location in physical memory.
  => This guarantees that physical memory always contains the correct value, but it requires additional time for the writes to physical memory.
- In **Write-back,** the value written to cache is not always written to physical memory. The value is written to physical memory only once, when data is removed from cache.
- Another situation that must be addressed is how to write data to locations not currently loaded into the cache, a **write miss.**
- One possibility is to load the location into cache and then write the new value to cache using either write-back or write-through. This is a **write-allocate policy.**
- A **write-no allocate policy,** updates the value in physical memory without loading it into the cache.

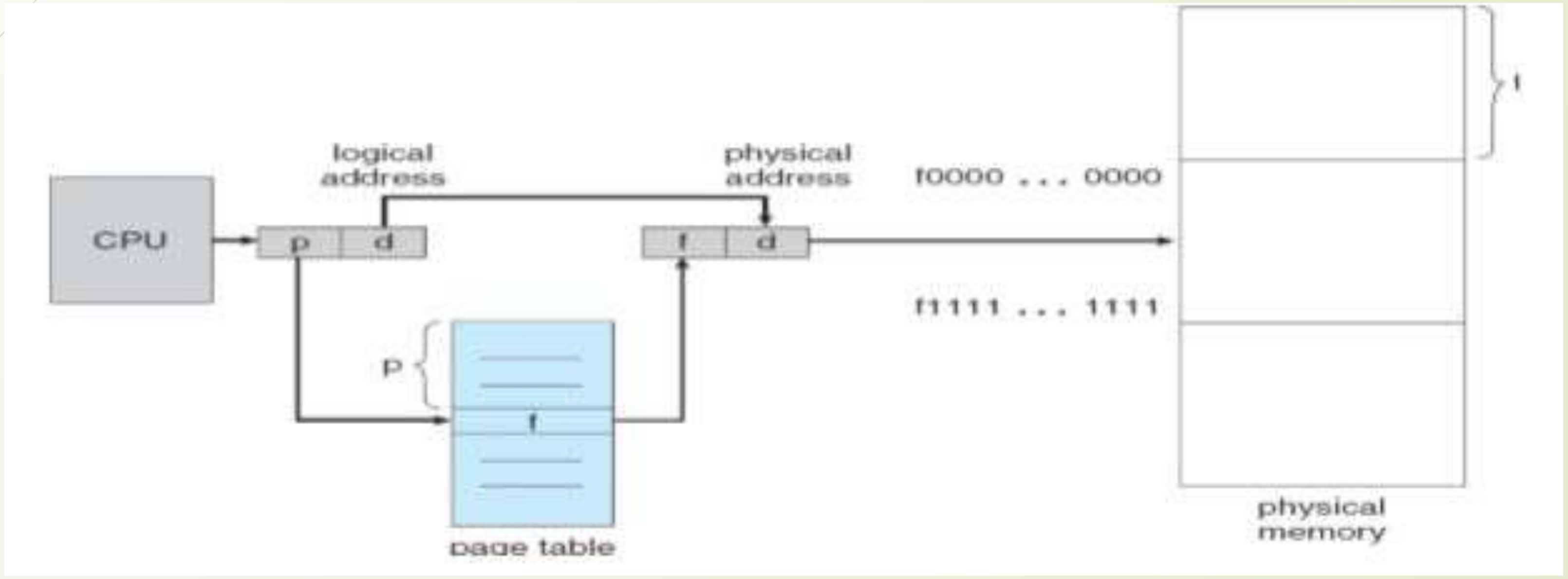
Cache performances:

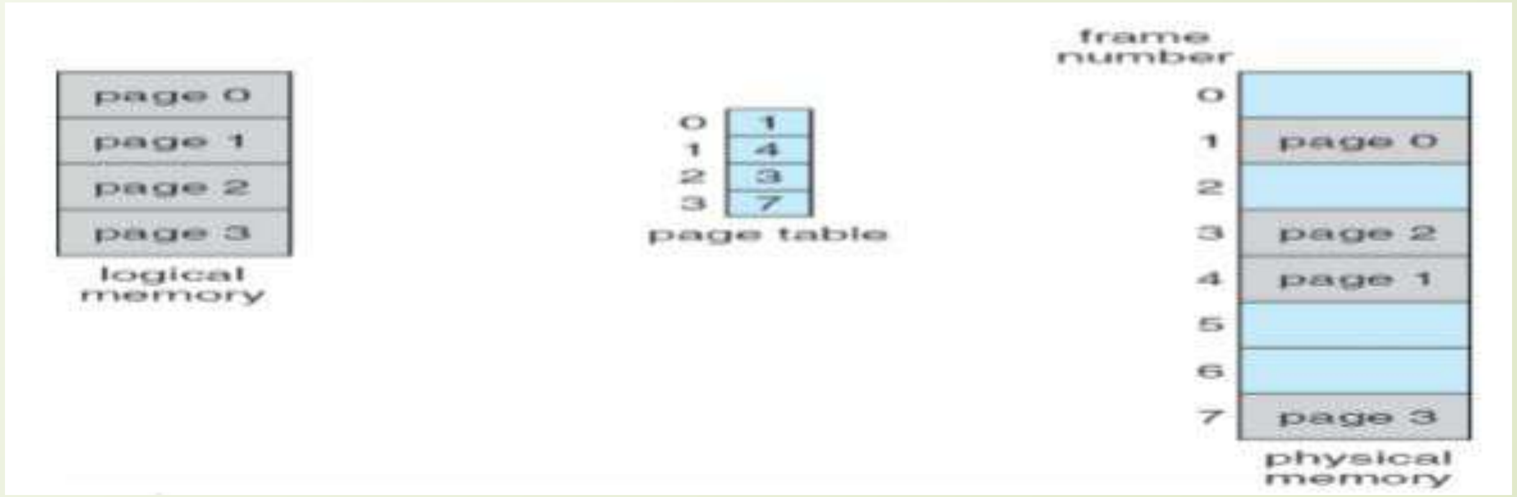
-primary components of cache performances are cache hits and cache misses.

- Every time the CPU accesses memory, it checks the cache. If the requested data is in the cache, it accesses data from cache rather than physical memory, this is a **cache hit.**
- If the data is not in the cache, the CPU accesses the data from main memory (and usually writes the data into the cache as well). This is a **cache miss.**
- **Hit ratio** is the ratio of total number of hits to the total number of CPU references.
- **Hit latency** is the time taken to find whether required element is in cache or not.
- The **average memory access time,** $T_M = hT_c + (1\text{-}h)T_p$
  where, h = hit ratio
  T_c = cache access time
  T_p = physical access time

Paging

(Avoids external fragmentation, any free frames can be allocated to a process that needs it.)





Contd...

- Physical memory is broken into blocks of the same size of pages, called memory frames.
- When a process is to be executed, its pages are loaded into any available memory frames.
- Every logical address generated by CPU is divided into any two parts: page number(p) and offset(d).
- The page number is used as an index for page table, that contains base address. The base address is combined with offset address to define physical address.
- If there are n pages in process, there must be at least n available frames to allocate the arriving process.
- The first page is loaded into one of the memory frame and the corresponding frame number is put into the page table. This process is repeated for all the pages.

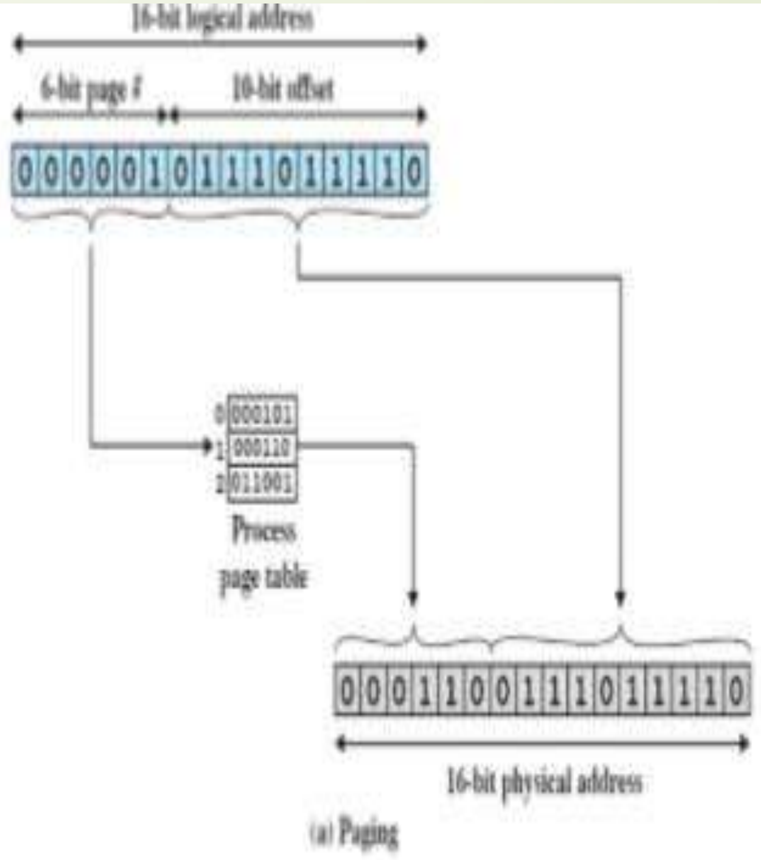
Contd…

Example:

In this example, we have the logical address 0000010111011110, which is page number 1, offset 478.

Suppose that this page is residing in main memory frame 6 = binary 000110.

Then the physical address is frame number 6, offset 478 = 0001100111011110



Segmentation

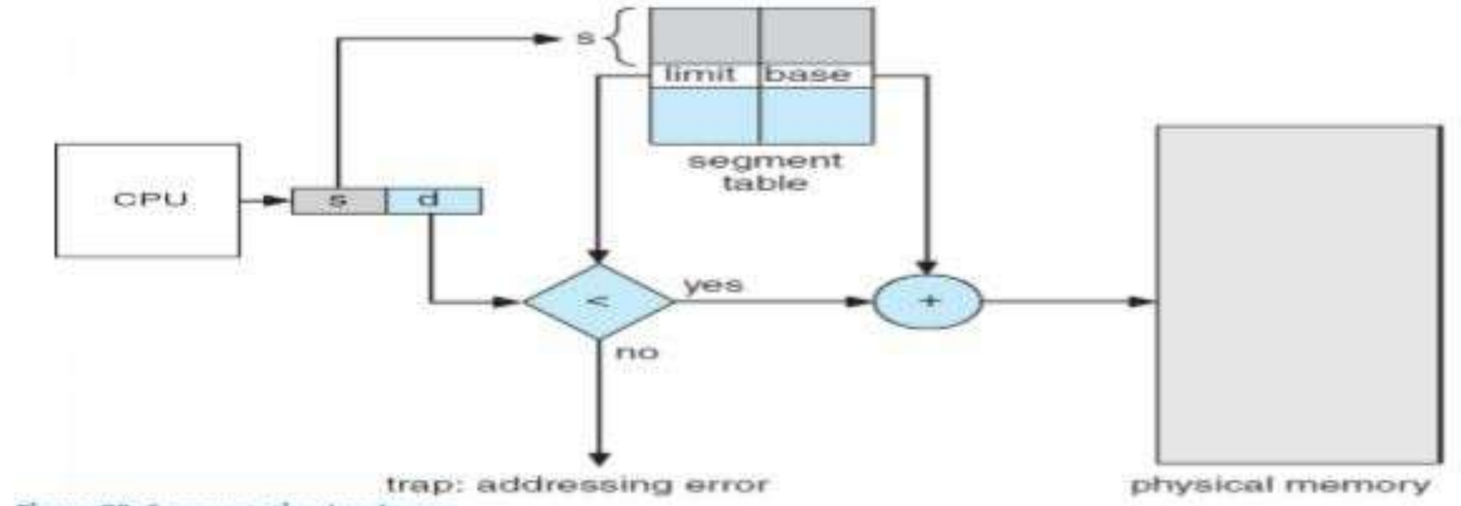
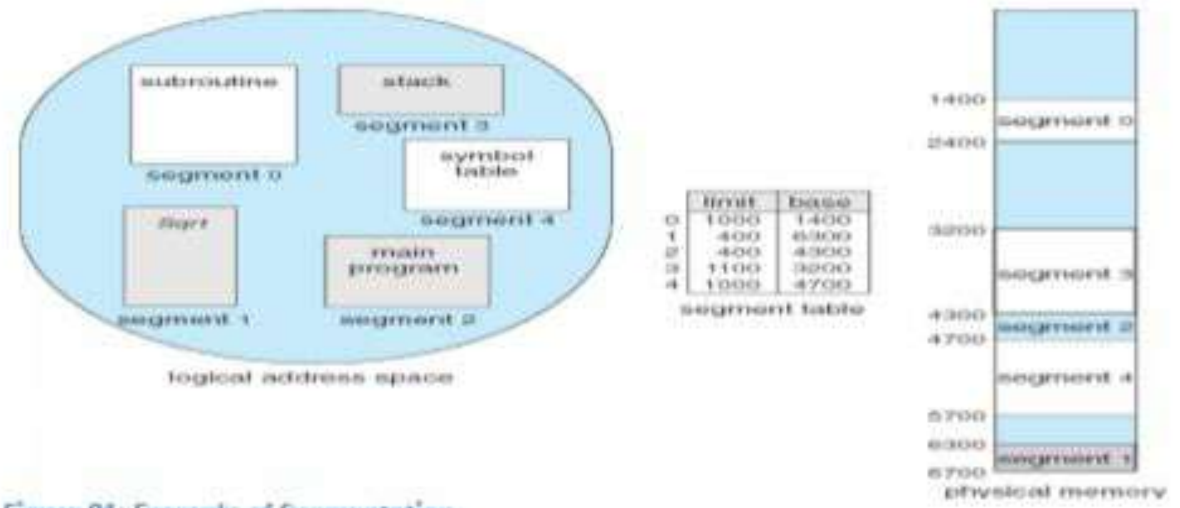


Figure 80: Segmentation hardware



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Figure 81: Example of Segmentation

Contd…

- In segmentation, program and its associated data are divided into number of segments such as: main program, procedure, function, local variables, global variables, stack, arrays, etc.
- The segments of program are of variable length.
- The logical address generated by CPU using segmentation consists of two parts, segment number and offset.
- Eliminates internal fragmentation, but like dynamic partitioning, suffers from external fragmentation.
- Segmentation contains the segment table containing starting address of segment. It also contains length of segment to assure that invalid addresses are not used.
- When the process enters the running state, the address of the segment table is loaded into memory.
- If segment number(s) < segment table length register(STLR), physical address is obtained, else addressing error.

Virtual memory

- Is a technique that allows execution of process that may not be completely in memory.
- Main advantage of this scheme is that program can be larger than physical memory.
- In virtual memory, program need not be fully loaded into main memory, only those portion execution of program is loaded.
- Example:
  => error handling routines are used only when error occurs in the data or computation.
  => certain feature of program is used rarely.
  =>small portion of table is loaded instead of whole table.
- Advantages:
  => less number of I/O would be required to load or swap program into memory.
  => program would no longer be constrained by the amount of physical memory available.
  => takes less physical memory, so many program can be run at the same time.

Demand paging:

- similar to paging system with swapping.

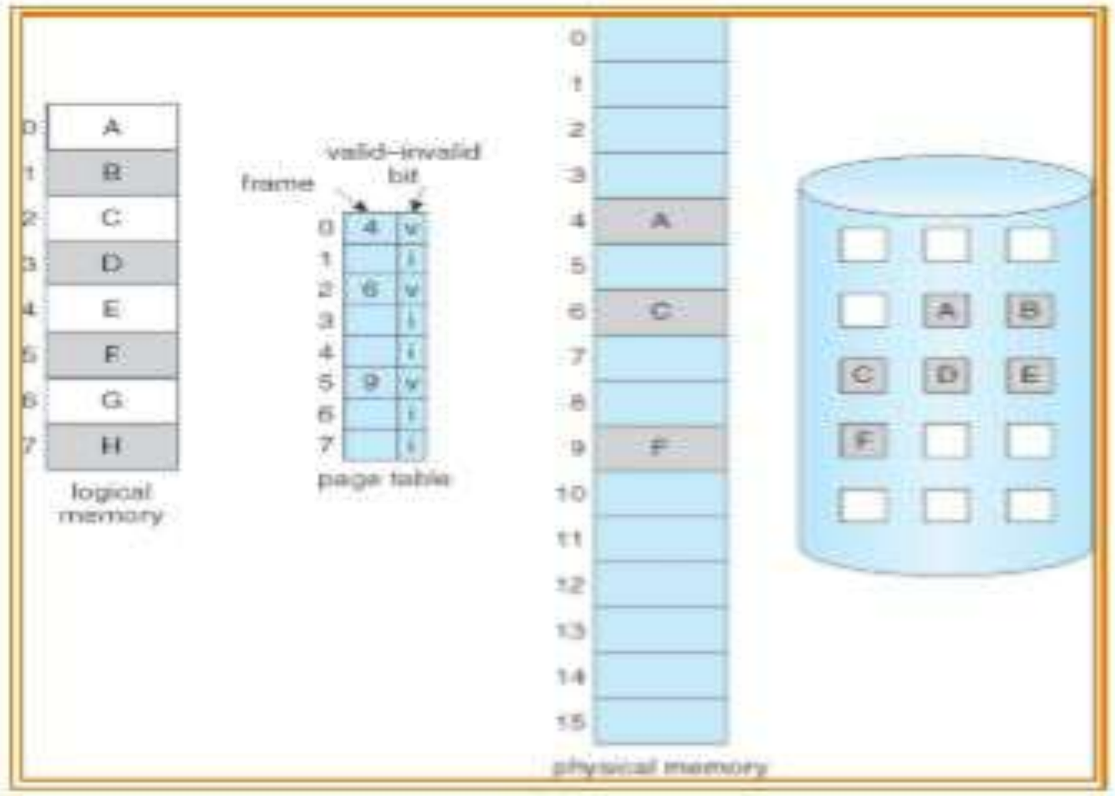


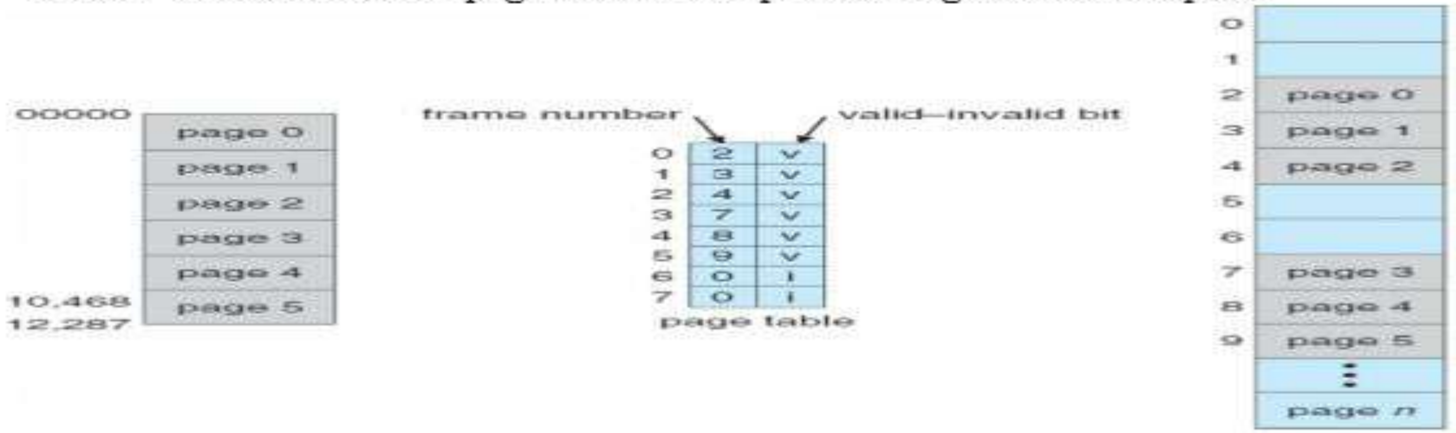
Figure [redacted] The Picture When All Pages Are Not In Memory

Contd…

- When we want to execute a process, we swap it into memory. Rather than swapping entire process, we only swap a page needed for execution. This is done by lazy swapper.
- Thus, some pages of process will be in memory and the other pages in the disk.
- Hence, a hardware support is required to distinguish between those pages that are on disk and on memory. This is done by using valid bit-invalid bit scheme.
- When the process tries to use the pages that is in memory, execution proceeds normally and when it tries to use the page that was not brought in memory, it results page fault trap.

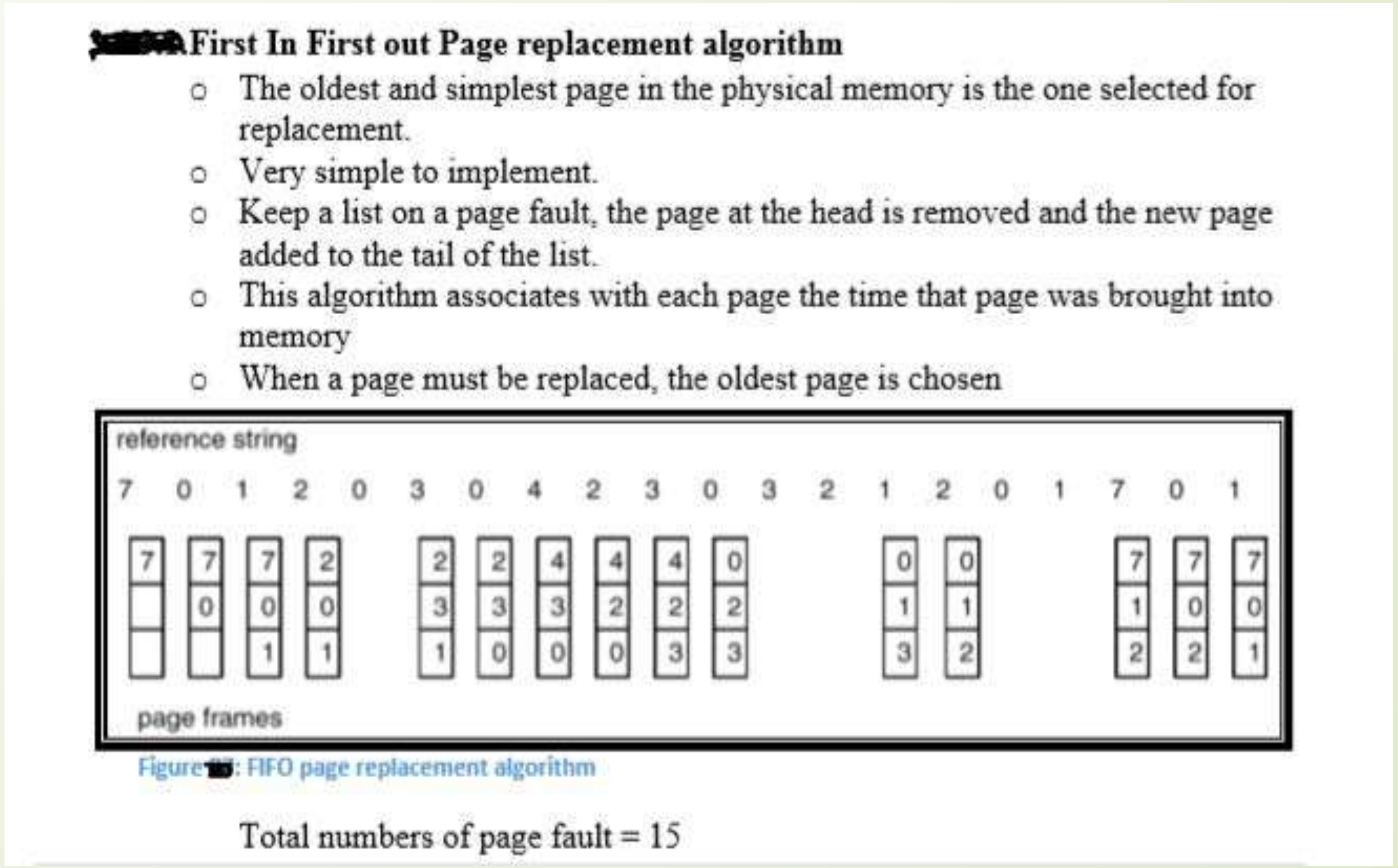
Memory protection:

- Main memory must accommodate both the operating system and the user processes
- Main memory is usually divided into two partitions:
    - Resident operating system, usually held in low memory
    - User processes then held in high memory
- In this case memory protection must be done.
- *Memory protection* means protecting the operating system from user processes and protecting user processes form one another.
- Memory protection implemented by associating protection bit with each frame
- **Valid-invalid** bit attached to each entry in the page table:
- "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
- "invalid" indicates that the page is not in the process' logical address space



Page Replacement Algorithm

- when there are more pages than the memory frames, then the page from the frame must be loaded out so as to load the unloaded page into the memory.
- replacing the pages in the memory follows certain algorithm.



[redacted] First In First out Page replacement algorithm

- The oldest and simplest page in the physical memory is the one selected for replacement.
- Very simple to implement.
- Keep a list on a page fault, the page at the head is removed and the new page added to the tail of the list.
- This algorithm associates with each page the time that page was brought into memory
- When a page must be replaced, the oldest page is chosen

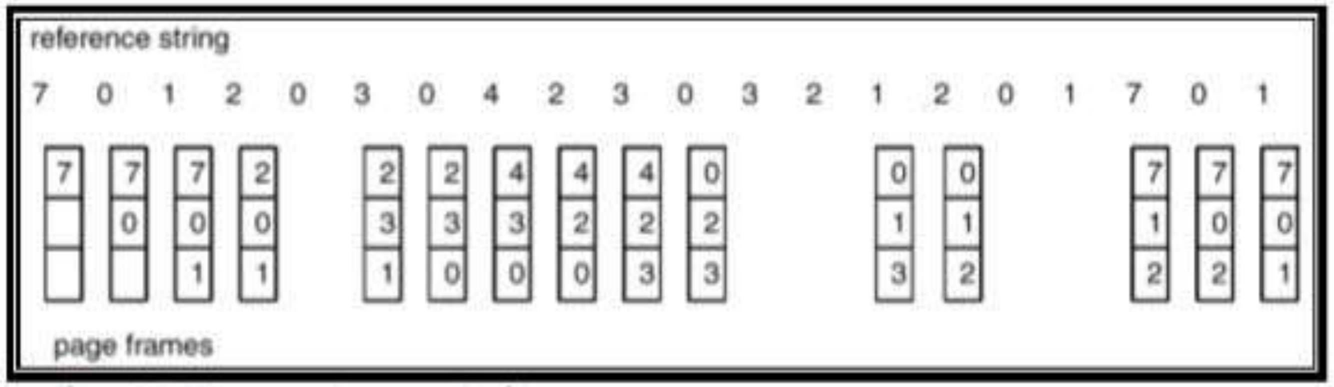
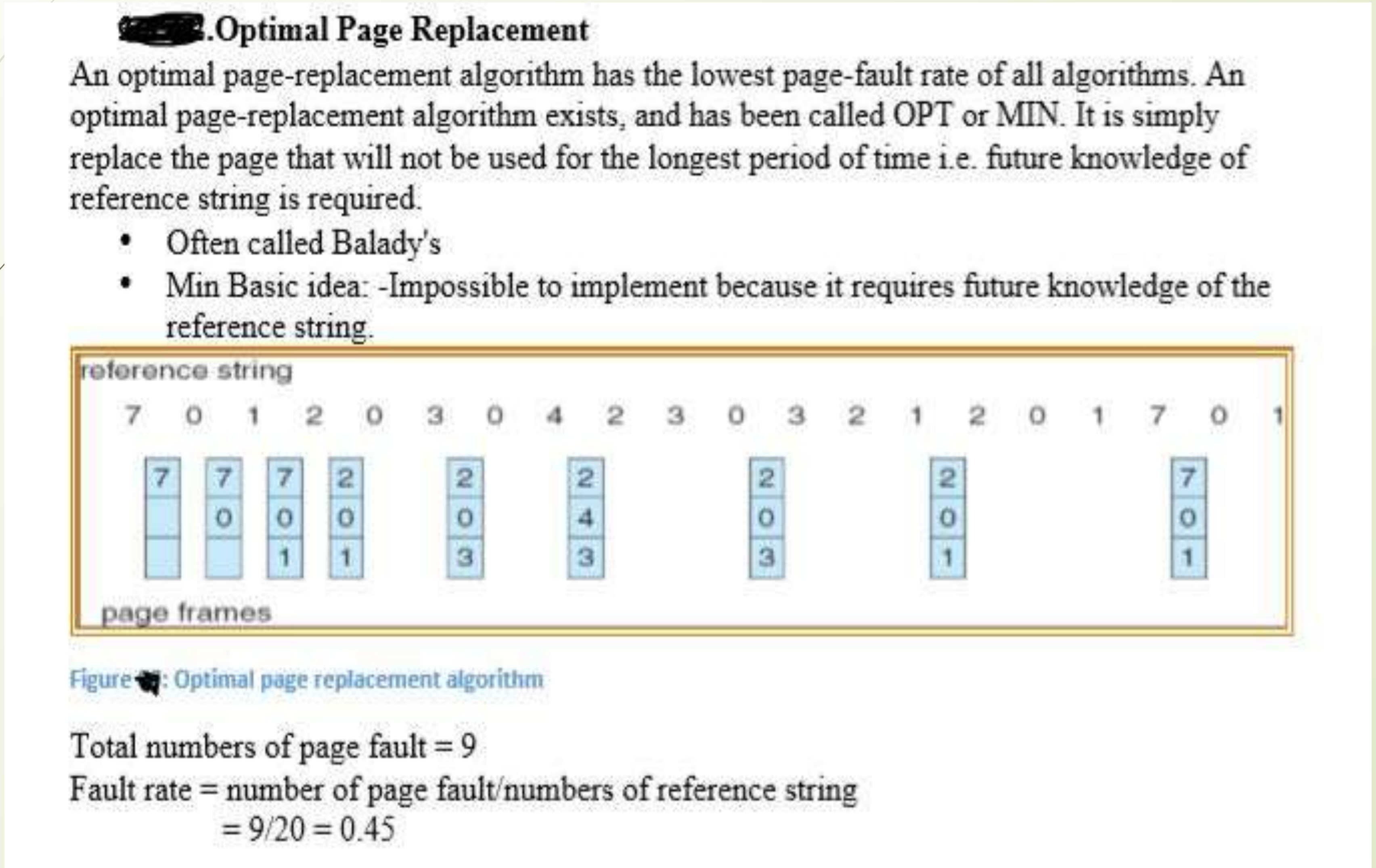


Figure [redacted]: FIFO page replacement algorithm

Total numbers of page fault = 15

Contd…



[redacted].Optimal Page Replacement

An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN. It is simply replace the page that will not be used for the longest period of time i.e. future knowledge of reference string is required.

- Often called Balady's
- Min Basic idea: -Impossible to implement because it requires future knowledge of the reference string.

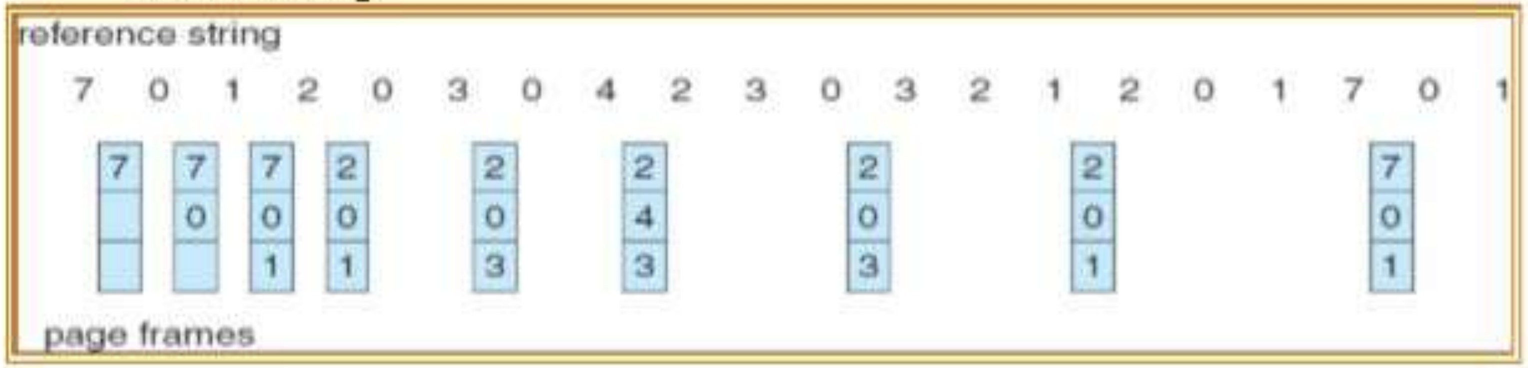


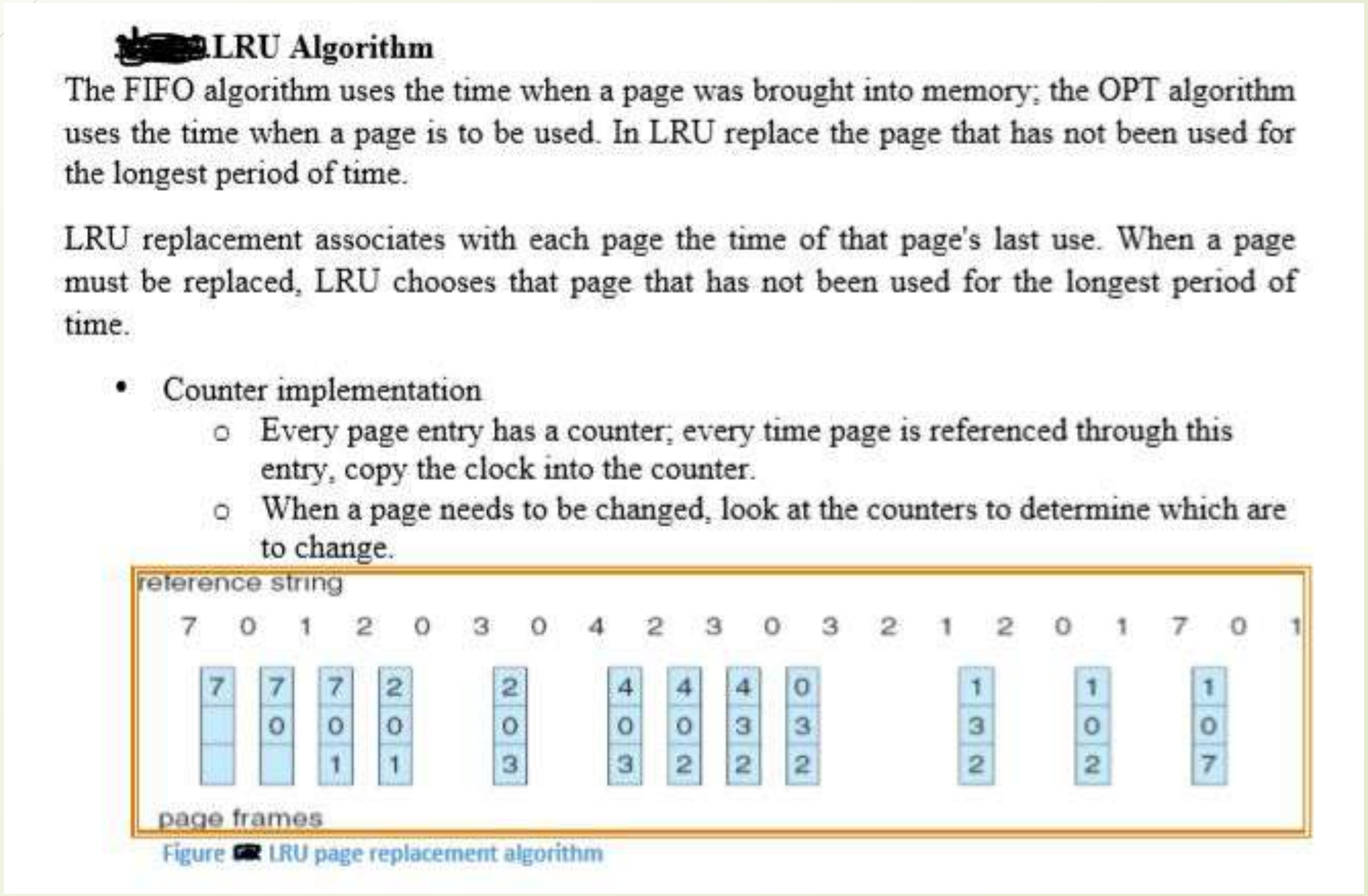
Figure [redacted]: Optimal page replacement algorithm

Total numbers of page fault = 9

Fault rate = number of page fault/numbers of reference string

= 9/20 = 0.45

Contd…



[redacted]LRU Algorithm

The FIFO algorithm uses the time when a page was brought into memory; the OPT algorithm uses the time when a page is to be used. In LRU replace the page that has not been used for the longest period of time.

LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time.

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be changed, look at the counters to determine which are to change.

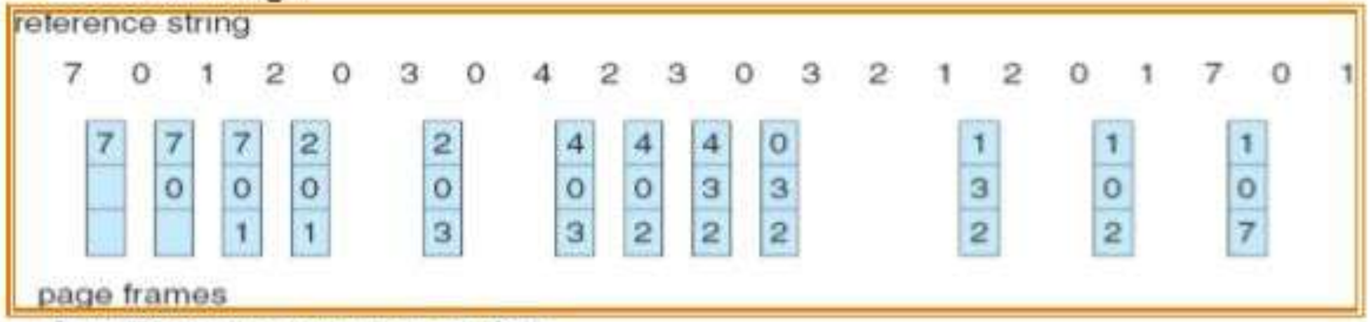


Figure [redacted] LRU page replacement algorithm