# Chapter 3

# **Memory Management**

- Memory is an important resource that must be carefully managed.
- Most computers have memory hierarchy with RAM, ROM and Non volatile Disk storage. The operating system present in each system coordinates these memories.
- A part of operating system that manages memory hierarchy is called **memory manager**.

**ROLE OF MEMORY MANAGER:**

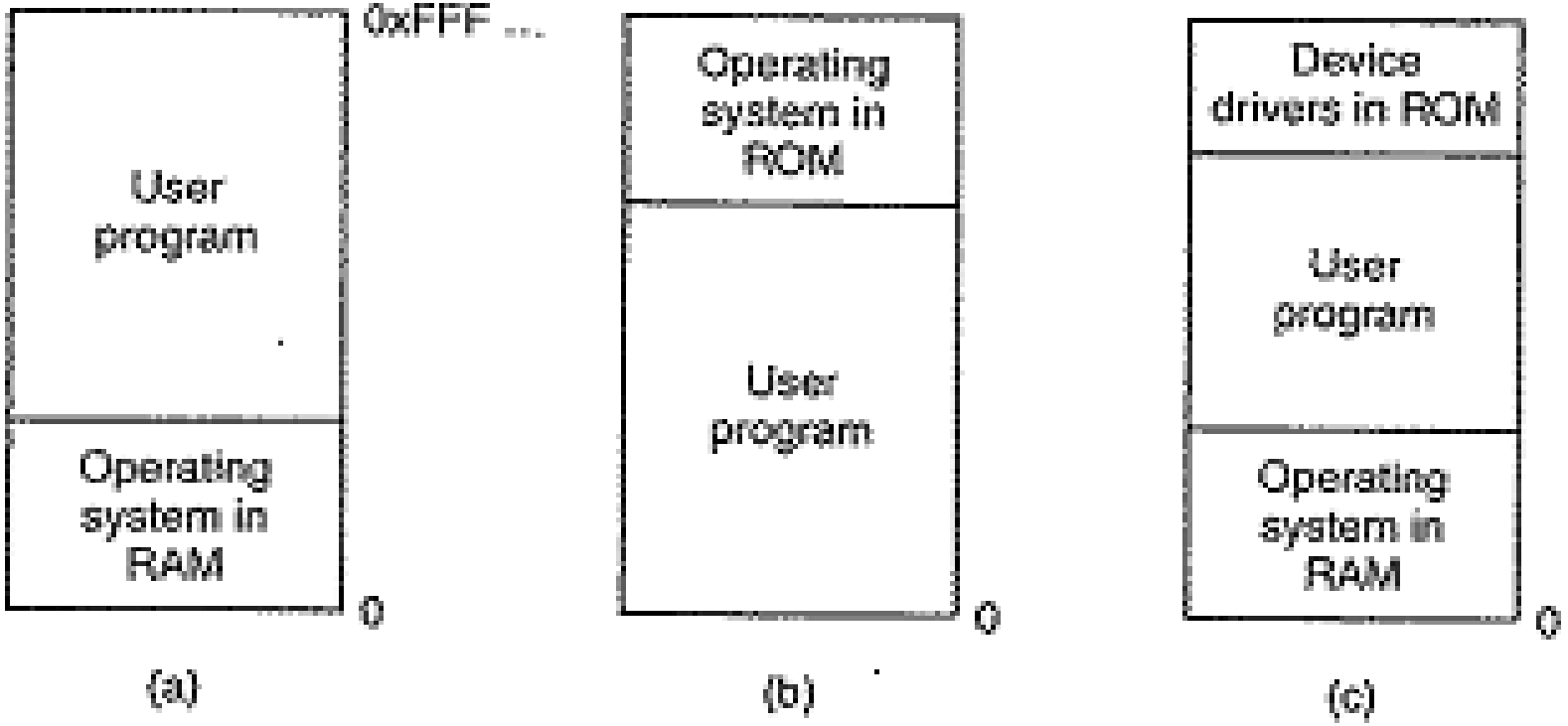
1. Tracks which parts are in use and which parts are not in use.
2. Allocates and deallocates memory to processes

3. Manages swapping between main memory and disk.

## MEMORY MANAGEMENT:

### Monoprogramming without swapping or paging:

- Here only **one program at a time**, so entire memory is shared by program and Operating system alone.
- Different ways in which memory can be organized to be shared by One process and Operating system is shown in below diagrams.
- Here requests a program for run, Operating system copies the requested program from disk to memory and executes it.




The above figure illustrates different ways of organizing memory between program and operating system

- When the process finishes running, O.S waits for new request from user.

## Multiprogramming with fixed partitions:

- In order to enable multiprogramming facility in modern systems, **many process needs to be placed in memory**. By placing multiple programs in memory , when one process is blocked, another process can be run.
- **So in order to keep N process in memory, here the entire memory is divided in N partitions.**
- After partitioning memory, an input queue can be maintained for each partition or for all partitions collectively a single input queue can be maintained.

- When a job gets to head of the queue they are loaded into partition.
- By maintain only one queue for all partitions, the **wait time** that occurs by maintain input queue for each partition is reduced.
- **Memory utilization is worst**

**Problems caused by Multiprogramming:**

- Multiprogramming introduces two essential problems that must be solved

  1. Relocation
  2. Protection

**RELOCATION PROBLEM:**

if a program is loaded in partition 1 at address 100K, and if the instruction is a call to procedure at absolute address 100 within binary file, what happens is system will jump to a location 100 inside operating system and not to location 100K+100.

- To solve this relocation problem, we use two special hardware registers called **Base and Limit registers**.
- So **every memory address generated has base register content added** to it before being sent to memory. Thus if base register contains value 100k, a call to $100^{th}$ instruction is effectively turned into a call 100k+100 instruction. The **limit register is loaded with the maximum length** of that partition so instruction beyond this limit cannot be accessed.

**PROTECTION PROBLEM:**

- In multiuser system, it is undesirable to let processes read and write memory belonging to other user.

- So the solution was to **assign a 4-bit protection code to each block.** The **PSW(program status word)** contains a 4-bit key.
- Any attempt by a running process to **access the memory is checked**, if **PSW matches** they are allowed to access else the access are denied.

**SWAPPING:**

- In batch system, **many jobs can be kept in memory** to keep CPU busy all time.
- In **time sharing or Personal computers**, since there is not enough main memory to hold all active process, **excess process must be kept on disk** & brought in whenever there is necessity to run.

- Two methods by means of which above theory can be realized are
  1. swapping
  2. virtual memory.

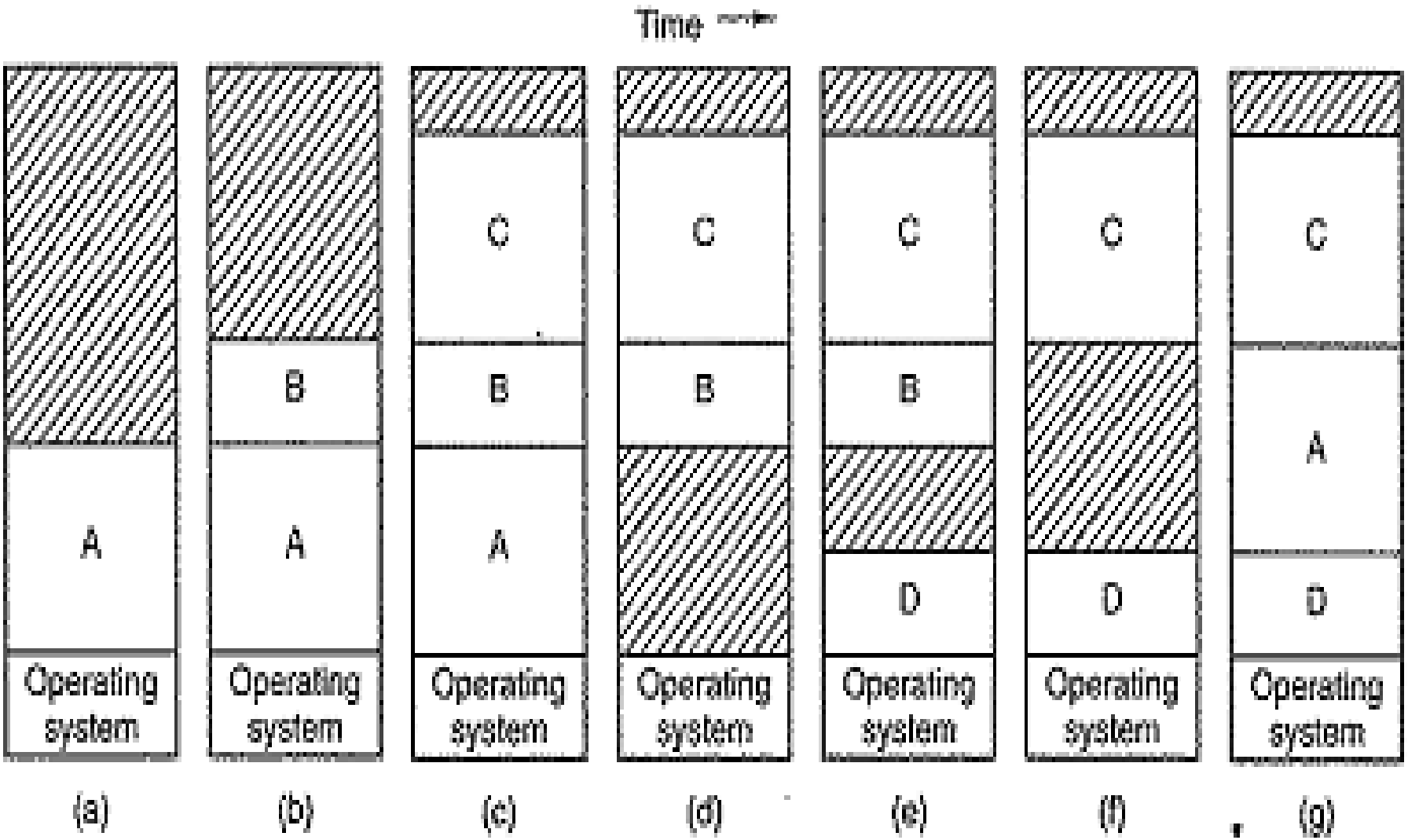


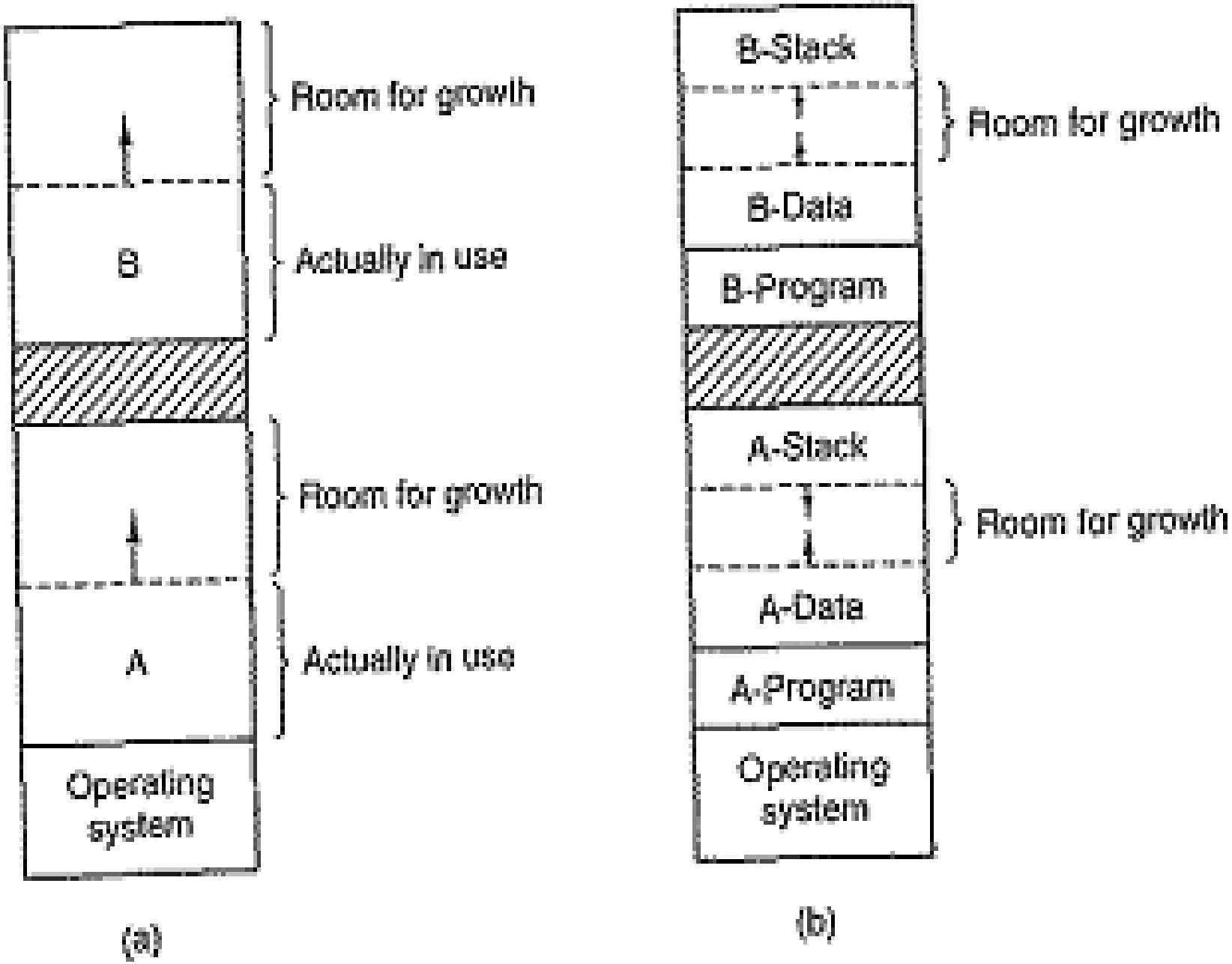

**figure depicts, Memory allocation changes and address contained changes as processes come into memory and leaves it**

- Here the **number, location, size of partition vary dynamically**, allowing good memory utilization.
- When memory is dynamically allocated, to keep track of memory usage two methods used are

    1. Bitmap
    2. Linked list.

Disadvantage: Allocating and De-allocating memory is complex, memory compaction is needed and as data segment grows, problem rises.

The below diagram shows allocating space for growing data segment, and allocating space for growing stack and growing data segment.

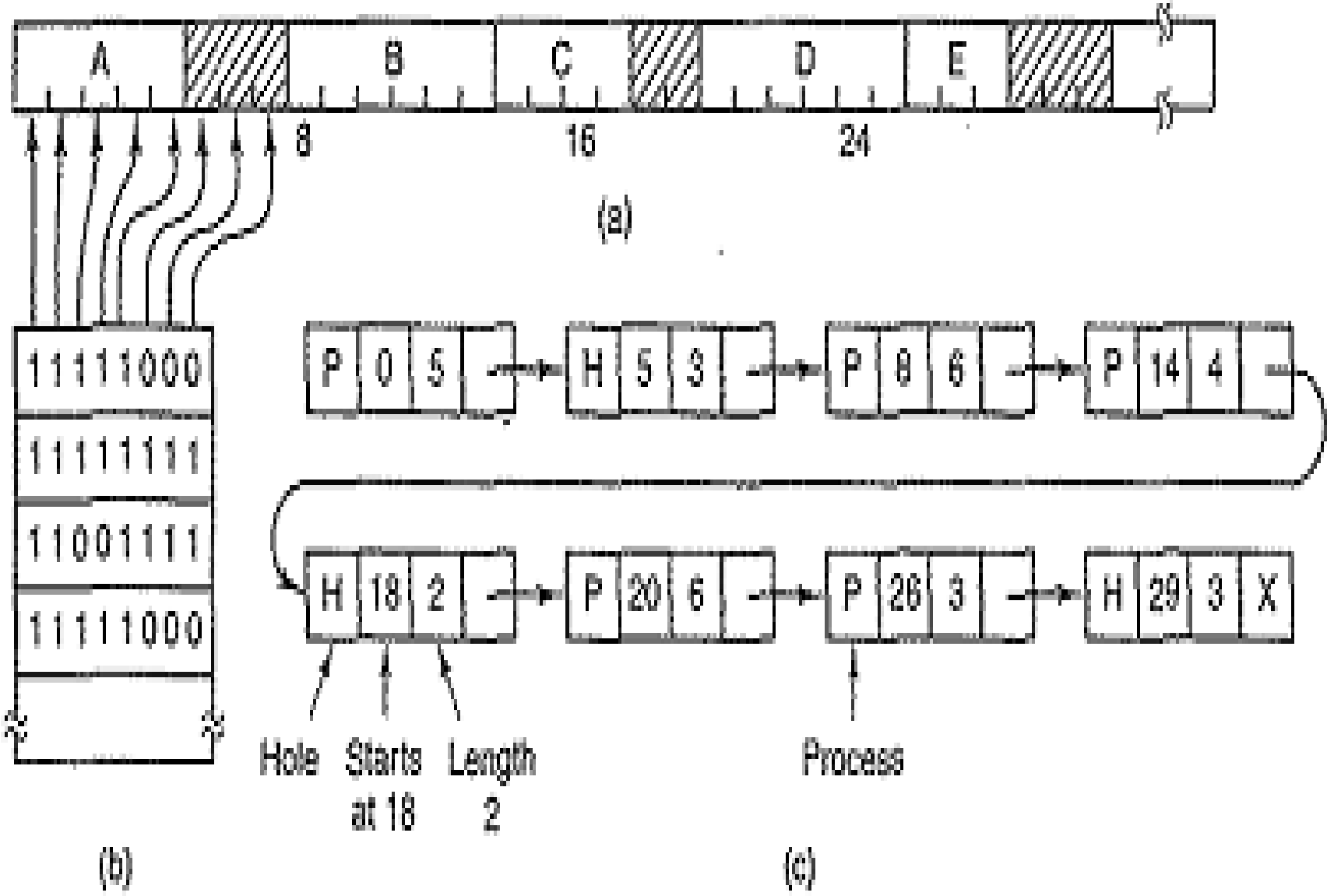

Room for growth
B
Actually in use
Room for growth
A
Actually in use
Operating system
(a)
B-Stack
Room for growth
B-Data
B-Program
A-Stack
Room for growth
A-Data
A-Program
Operating system
(b)

## Memory management with bit map:

- Memory is divided into allocation units,
- For each allocation unit, a bit is assigned. **If bit=0 -unit is free, If bit is 1- unit is occupied.**
- The diagram of memory management with bitmap and linked list is shown below.

Design issue in this case,

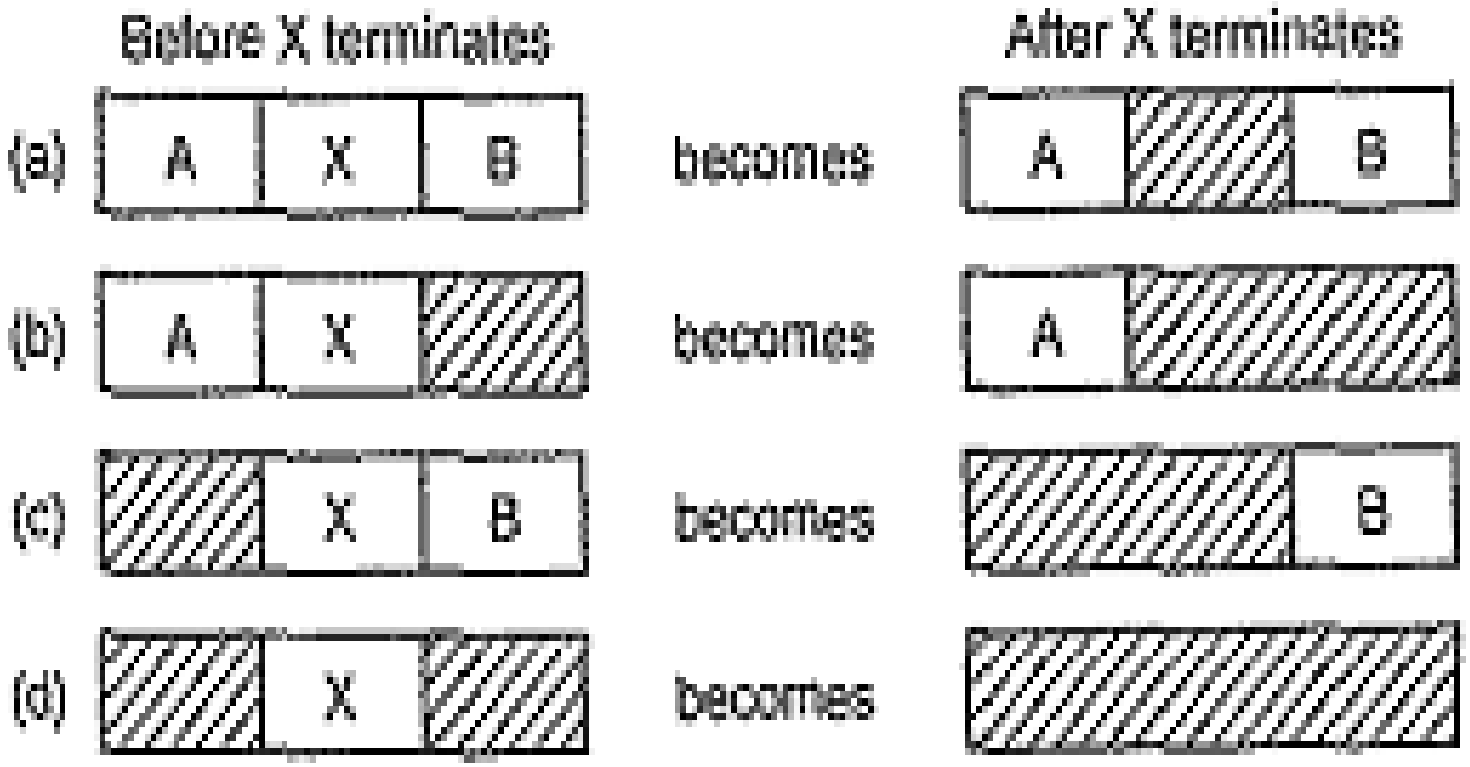
- The smaller the allocation unit is chosen, bitmap will be larger or vice versa.
- The drawback is searching a bit map for a run is time consuming.


A
B
C
D
E
8
16
24
(a)
11111000
11111111
11001111
11111000
P 0 5
H 5 3
P 8 6
P 14 4
H 18 2
P 20 6
P 26 3
H 29 3 X
Hole
Starts at 18
Length 2
Process
(b)
(c)

## Memory management with linked list.

- A linked list of allocated & free memory segment is maintained.
- Free running segment can be a process or hole.
- Each entry in list specifies a hole (H) or process(P) & the address at which it starts, the length and a pointer to next entry.
- The segment list is kept sorted by address.
- Advantage is when process terminates or is swapped out, updating list is straightforward.

- A terminating process has two neighbours either process or hole leading to 4 combinations as shown in below diagram.



- Several algorithms exists to allocate memory for newly created process if process & holes are kept on a list.
- **First fit**: Memory manager searches for list of segments until it finds holes. Then hole is broken into two pieces for both process and hole.
- **Next fit**: keeps track of where it found hole previously. So when again a new hole is to be found, it starts from previous hole location.
- **Best fit**: searches entry list, chooses hole that is adequate. But slower in working.
- **Worst fit**: Largest hole is chosen first.

- **Quick fit**: maintains separate list for some of common size requested. Finding a hole of required size is fast. But finding whether merge is possible or not is expensive.

## VIRTUAL MEMORY:

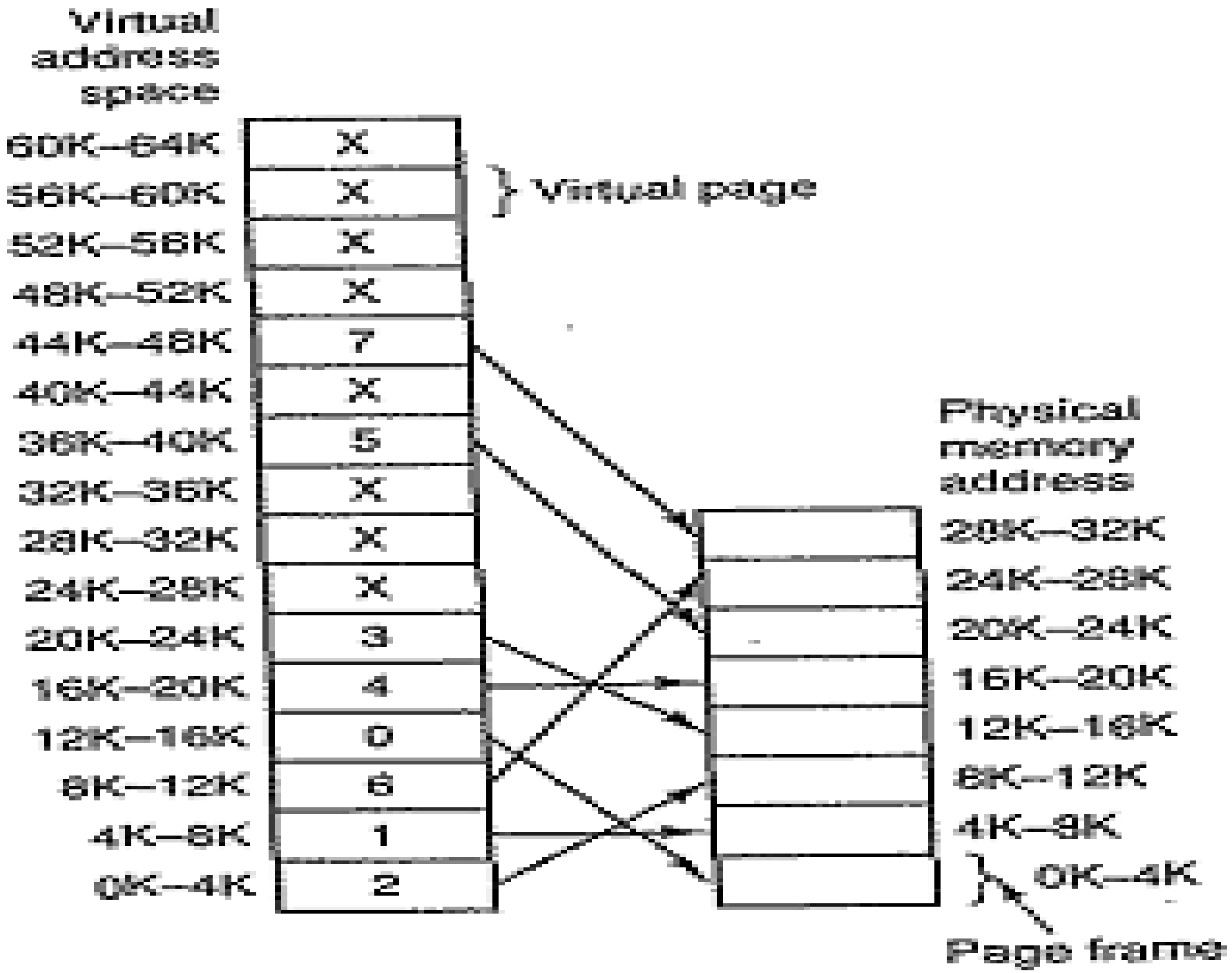
- Some programs are too big to fit in available memory. So they are split into pieces called **Overlays**.
- The overlays are kept on disk and are swapped in & out of memory by O.S dynamically.
- Although work of swapping is done by system, splitting up large programs into small pieces has to be done by programmer.
- Virtual memory is implemented with the help of paging concept

**PAGING:**

- **Virtual address** refers to address in virtual memory where program will be placed.
- This entire memory with virtual address is called collectively as **virtual address space**.
- If there is no virtual memory , the virtual address just points directly to the physical memory location.
- If there is virtual memory, virtual address is given to **MMU(memory management unit)** which maps virtual address to physical memory address.
- The virtual address space is divided into units called **page's.**

- Physical memory is also divided into units called **page frame**.
- **Size of page and page frame are always same**.

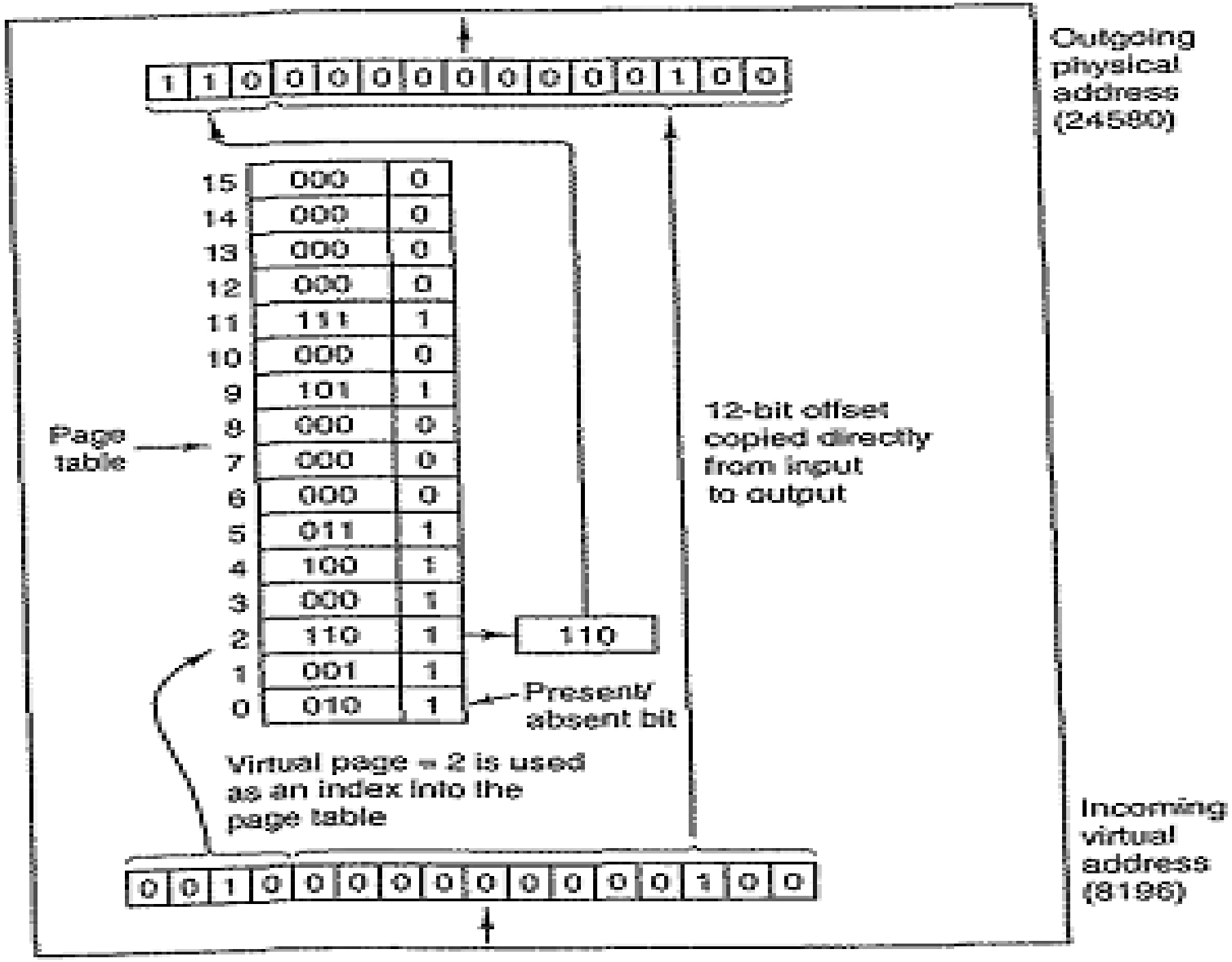
  *e.g.* *MOV REG, 0*
- When this instruction is executed, The virtual address is sent to MMU,
- MMU sees now that virtual address 0 corresponds to Page 0, as page 0 was mapped to page frame 2 (8192 to 12287).
- So MMU tracks this mapping and outputs address 8192 from where actual execution is done.


Virtual address space
60K–64K
56K–60K
52K–56K
48K–52K
44K–48K
40K–44K
36K–40K
32K–36K
28K–32K
24K–28K
20K–24K
16K–20K
12K–16K
8K–12K
4K–8K
0K–4K
X
X
X
X
7
X
5
X
X
X
3
4
0
6
1
2
Virtual page
Physical memory address
20K–32K
24K–28K
20K–24K
16K–20K
12K–16K
8K–12K
4K–8K
0K–4K
Page frame

- The above diagram shows relation between virtual address and physical memory address.
- In the above diagram, **virtual memory is big and most of its page are not mapped to page frame**. So when reference to unmapped location is made, a trap occurs. This condition is called **page fault.**
- So what operating system does is, it writes little used page frame into disk and fetches the page just referenced into page frame just freed and changes the map and restarts the trapped instruction.

**PAGE TABLE:**

- Purpose of page table **is to map virtual page into page frame.**
- Virtual address is split into **virtual page number** (the higher order bits) and **offset** (the lower order bits).
- **Virtual page number is used as index** into page table to find entry for that virtual page.
- For that page table entry, the corresponding page frame number is found.
- This Page frame number is appended to high order end of offset. Thus forming a physical address.
- Below diagram shows internal operation of MMU with 16 4 –KB page


Outgoing physical address (24590)
1 1 0 0 0 0 0 0 0 0 0 0 1 0 0
15 000 0
14 000 0
13 000 0
12 000 0
11 111 1
10 000 0
9 101 1
8 000 0
7 000 0
6 000 0
5 011 1
4 100 1
3 000 1
2 110 1
1 001 1
0 010 1
Page table
110
Present/ absent bit
12-bit offset copied directly from input to output
Virtual page = 2 is used as an index into the page table
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
Incoming virtual address (8196)

**Issues:**

- Page table can be extremely large, so **mapping would take considerably long time**. So by some mechanism there is a need to make mapping faster.

**Remedy:**

- In order to get rid of the problem of storing huge page table in memory all the time, the alternative solution is to **use multilevel page table.**

**Disadvantages of page table:**

- Since page table itself is large & it consumes memory.
- With paging, additional memory reference are needed to access page table.

- Reduces overall execution speed and performance.
- The solution to this is to introduce a small hardware device called **TLB (Translational look aside buffer)** also called associate memory for mapping virtual address to physical address going through page table.
- TLB is present inside the MMU.
- TLB consists of entries and
- Each entry contains **information about one page including virtual page number , the protection code, the physical page frame in which page is loaded.**

## SEGMENTATION:

- The **virtual memory is one dimensional** and is limited in-terms of accessibility.
- In some situations, it is desirable to have two or more separate virtual address space rather than having one.
- In virtual memory, the entire virtual address pace is **divided into many chunks of virtual address space and each chunks are assigned to each process that request for the chunk.** But the main drawback of this concept was, as the program grows or shrinks lot of process's are involved as the entire address space has to be maintained.

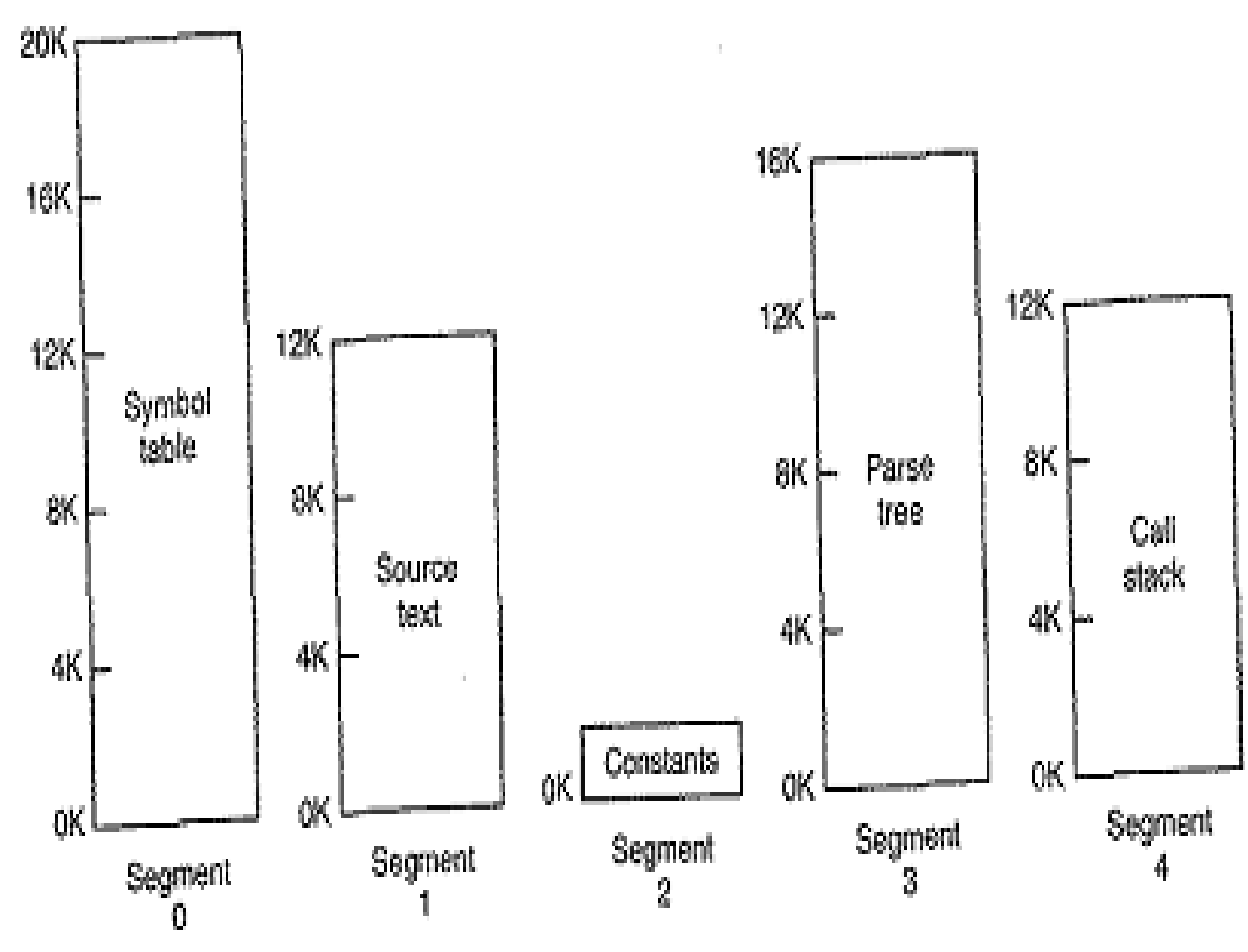
- So the ultimate solution was to provide the machine with many independent address space called **Segment**.
- Each segment consists of linear sequence of address from 0 to some maximum.
- Unlike paging, here different segments may have different length.
- And segment length may change i.e. grow or shrink independently during execution as each segment constitutes a separate address.
- In-order to specify an address among segment, **two part addressing mechanism** is used,

the **first part** (i.e Segment number), segment number denotes the segment within which the address could be found and the **second part** (i.e. the address that follows segment number) denotes the exact address within the segment.

Diagram illustrating how a segment memory allows each table to grow or shrink independently is shown in below diagram

**Advantages:**

- Sharing of procedures between programmers is facilitated as each segment forms a logical entity of which each programmer is aware.
- Different segments can have different kinds of protection

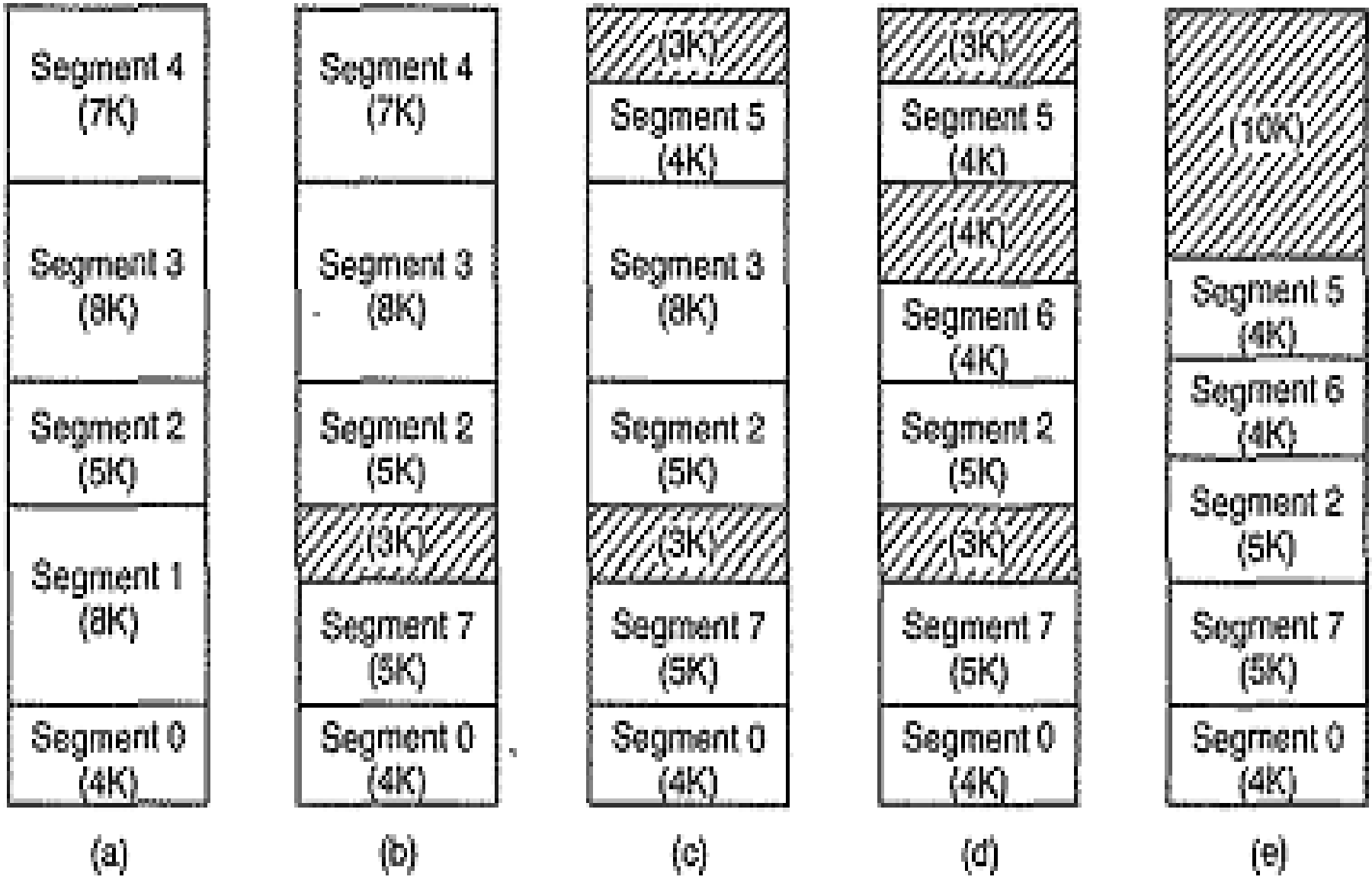

20K
16K
12K
8K
4K
0K
Symbol table
Segment 0
12K
8K
4K
0K
Source text
Segment 1
0K
Constants
Segment 2
16K
12K
8K
4K
0K
Parse tree
Segment 3
12K
8K
4K
0K
Call stack
Segment 4

**Disadvantage:**

- Segmentation also introduces the phenomenon called **checker boarding** or **external fragmentation** as shown in below diagram from fig (a) to fig (d).

**Remedy:**

- **Memory compaction technique** is needed to get rid of external fragmentation.
- The below diagram illustrating external fragmentation phenomenon shows how the memory compaction technique has been done resulting in fig (e).


Segment 4 (7K)
Segment 3 (8K)
Segment 2 (5K)
Segment 1 (8K)
Segment 0 (4K)
(a)
Segment 4 (7K)
Segment 3 (8K)
Segment 2 (5K)
(3K)
Segment 7 (5K)
Segment 0 (4K)
(b)
(3K)
Segment 5 (4K)
Segment 3 (8K)
Segment 2 (5K)
(3K)
Segment 7 (5K)
Segment 0 (4K)
(c)
(3K)
Segment 5 (4K)
(4K)
Segment 6 (4K)
Segment 2 (5K)
(3K)
Segment 7 (5K)
Segment 0 (4K)
(d)
(10K)
Segment 5 (4K)
Segment 6 (4K)
Segment 2 (5K)
Segment 7 (5K)
Segment 0 (4K)
(e)