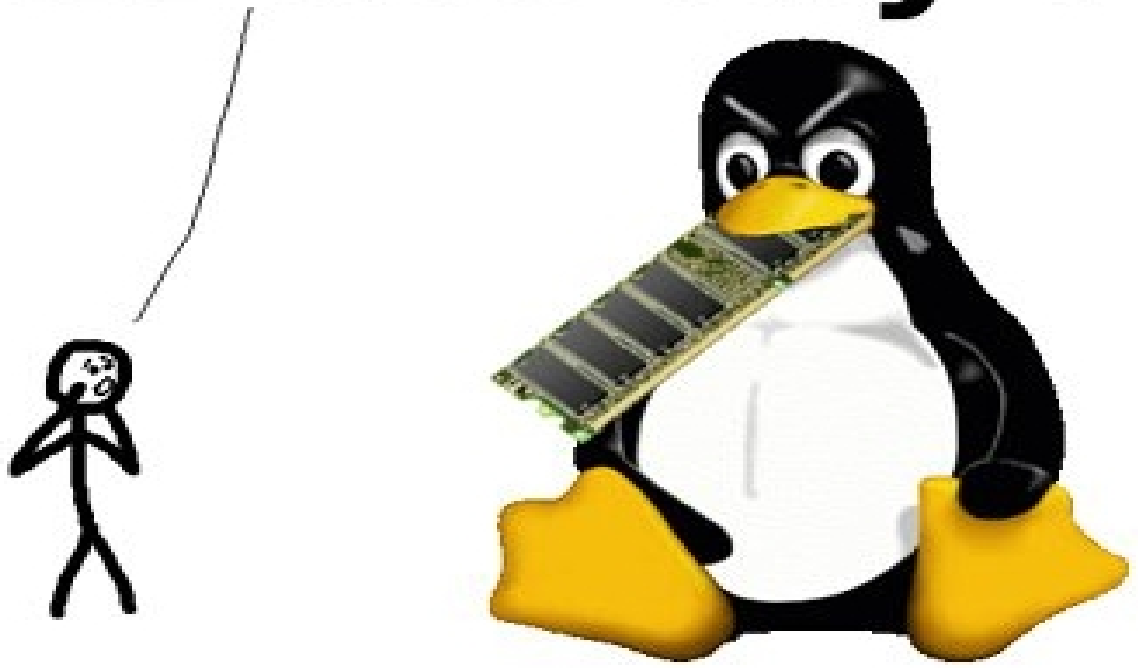
CSEN3103 – Sec A- Memory Management (by NB)

Linux ate my ram!

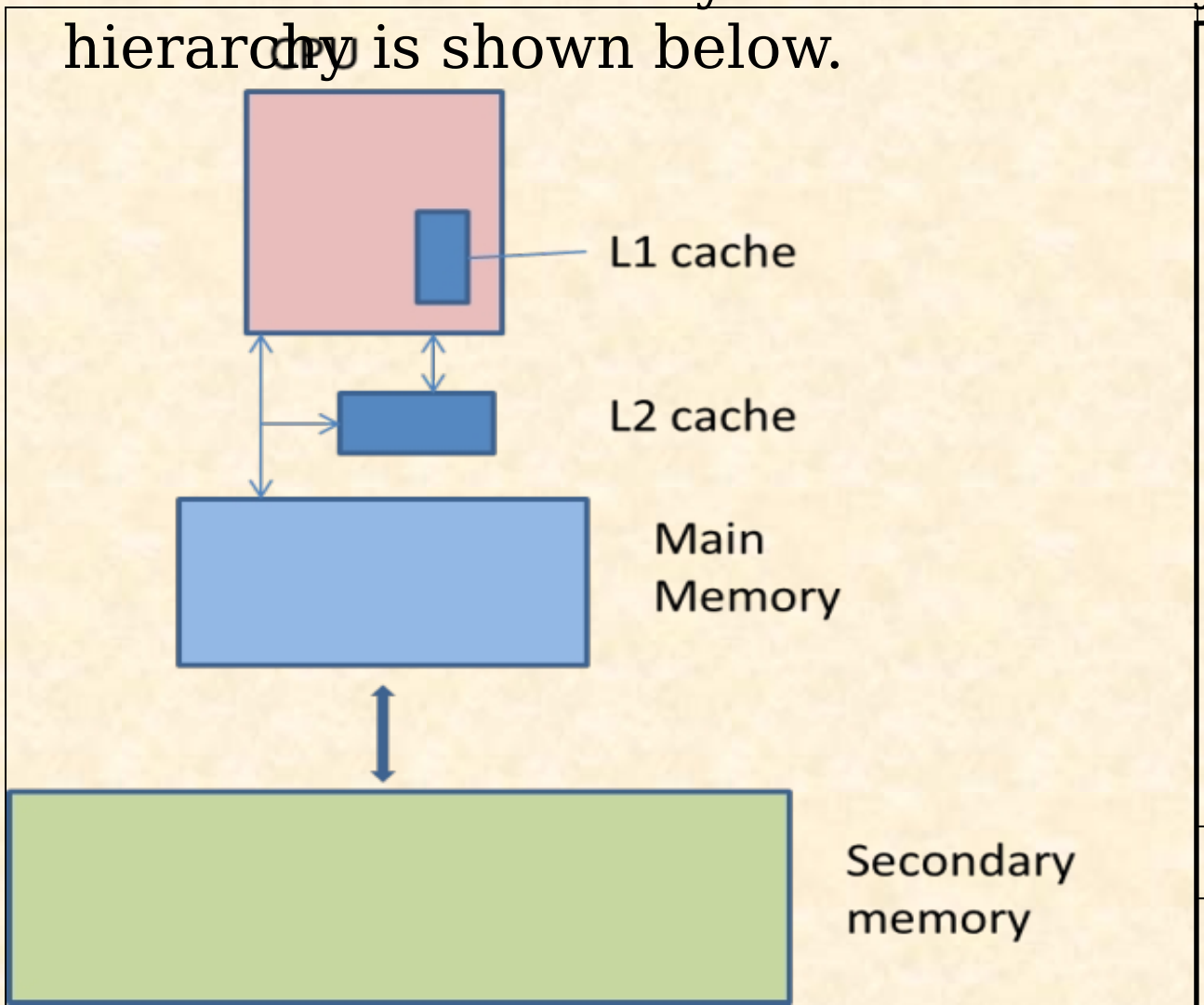


Why is it doing this?

Disk caching makes the system much faster! There are no downsides, except for confusing newbies. It does not take memory away from applications in any way, ever!

Introduction

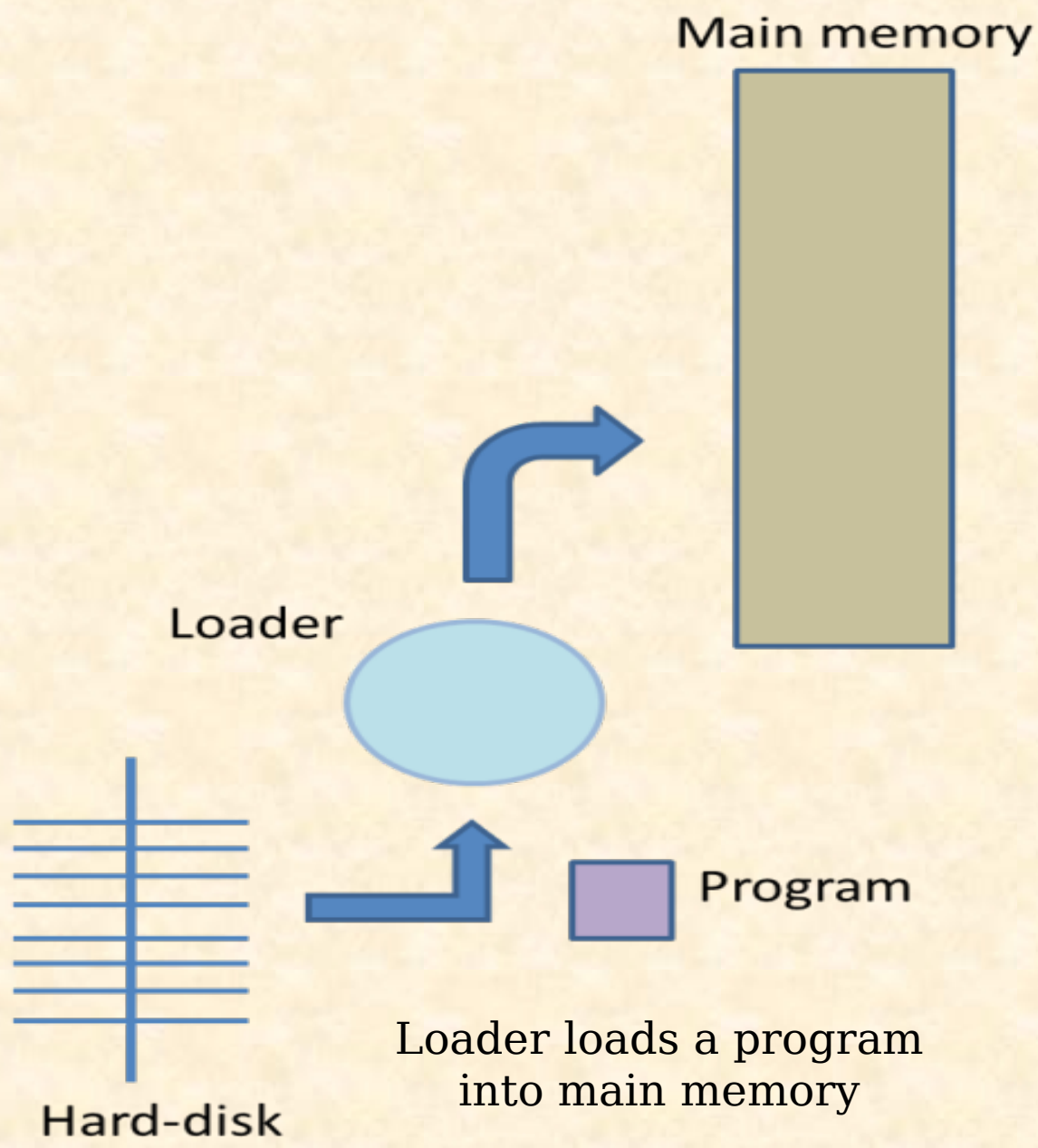
- In a computer system, there is a hierarchy of storage elements organized such that the fastest memory is closest and directly accessible by the CPU. The hierarchy is shown below.



Memory hierarchy

On-chip cache (L1) is the closest and fastest memory. The secondary memory is largest, slowest and farthest from the CPU. Cache and main memory are directly accessible by the CPU

Loader

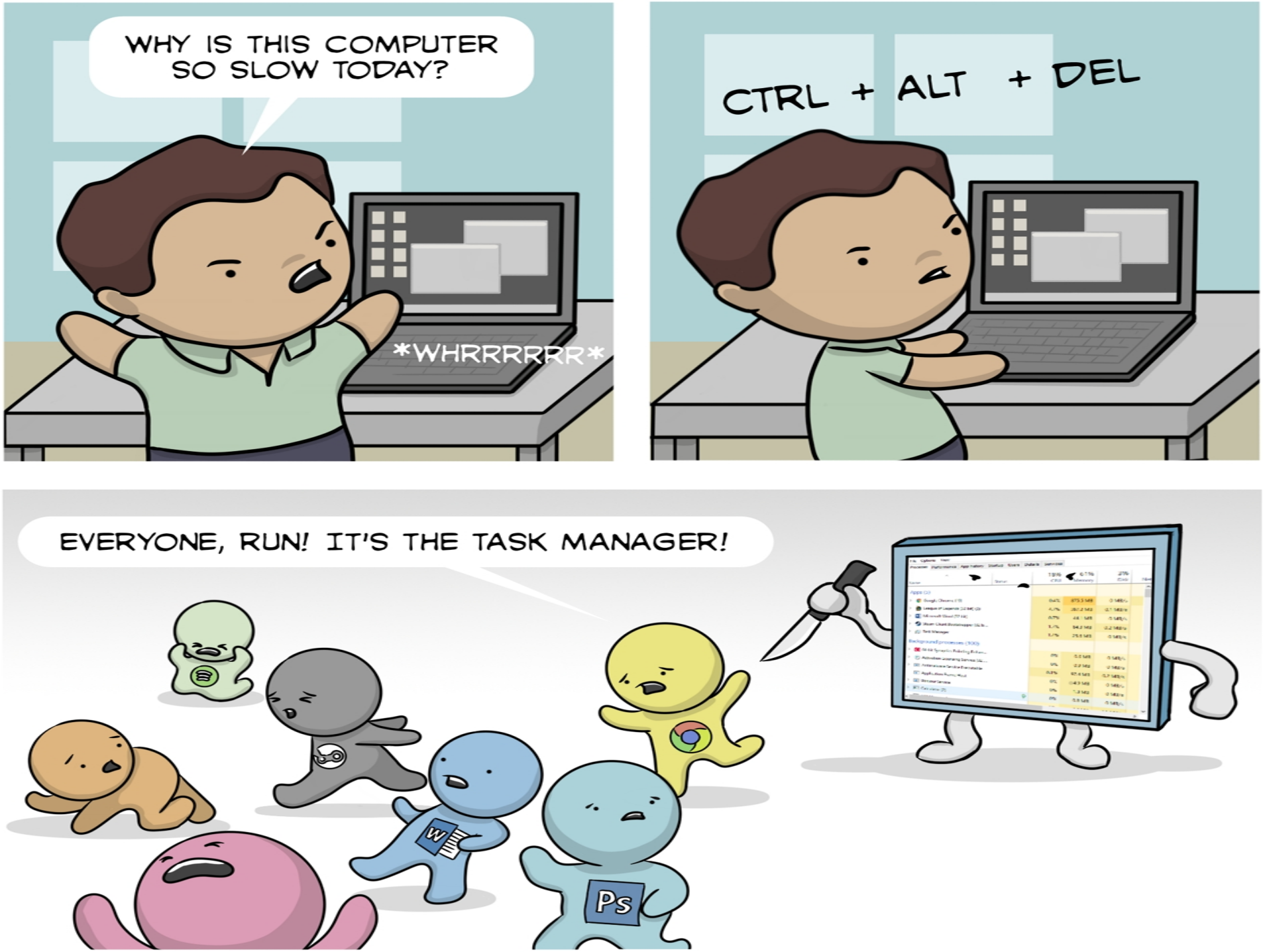


Loader loads a program into main memory

The secondary memory is the permanent memory of the memory system. Therefore, the files are stored in secondary memory. At the time of execution, the programs are loaded into main memory from secondary memory by a system program called **loader**.

Memory Management

- The issues concerning efficient memory management are as follows:
  - Allocation
  - Relocation
  - Sharing
  - Protection


WHY IS THIS COMPUTER SO SLOW TODAY?
WHRRRRRR
CTRL + ALT + DEL
EVERYONE, RUN! IT'S THE TASK MANAGER!
W
Ps

Memory Management

- **Allocation**
  - The operating system must open a process for the program being loaded by the loader.
  - It must also allocate main memory space for the incoming process and its associated data so that it becomes a resident process.

Programs and services are assigned with a specific memory as per their requirements when they are executed. Once the program has finished its operation or is idle, the memory is released and

Memory allocation has two core types:
Static Memory Allocation: The program is allocated memory at compile time.
Dynamic Memory Allocation: The programs are allocated with memory at run time.

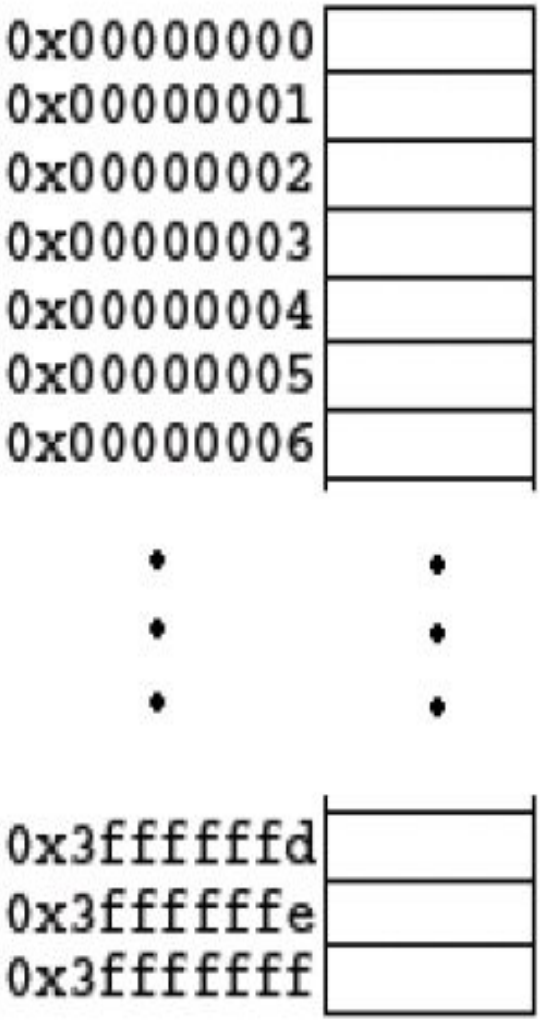
Computer memory can be considered as a very large array of bytes.

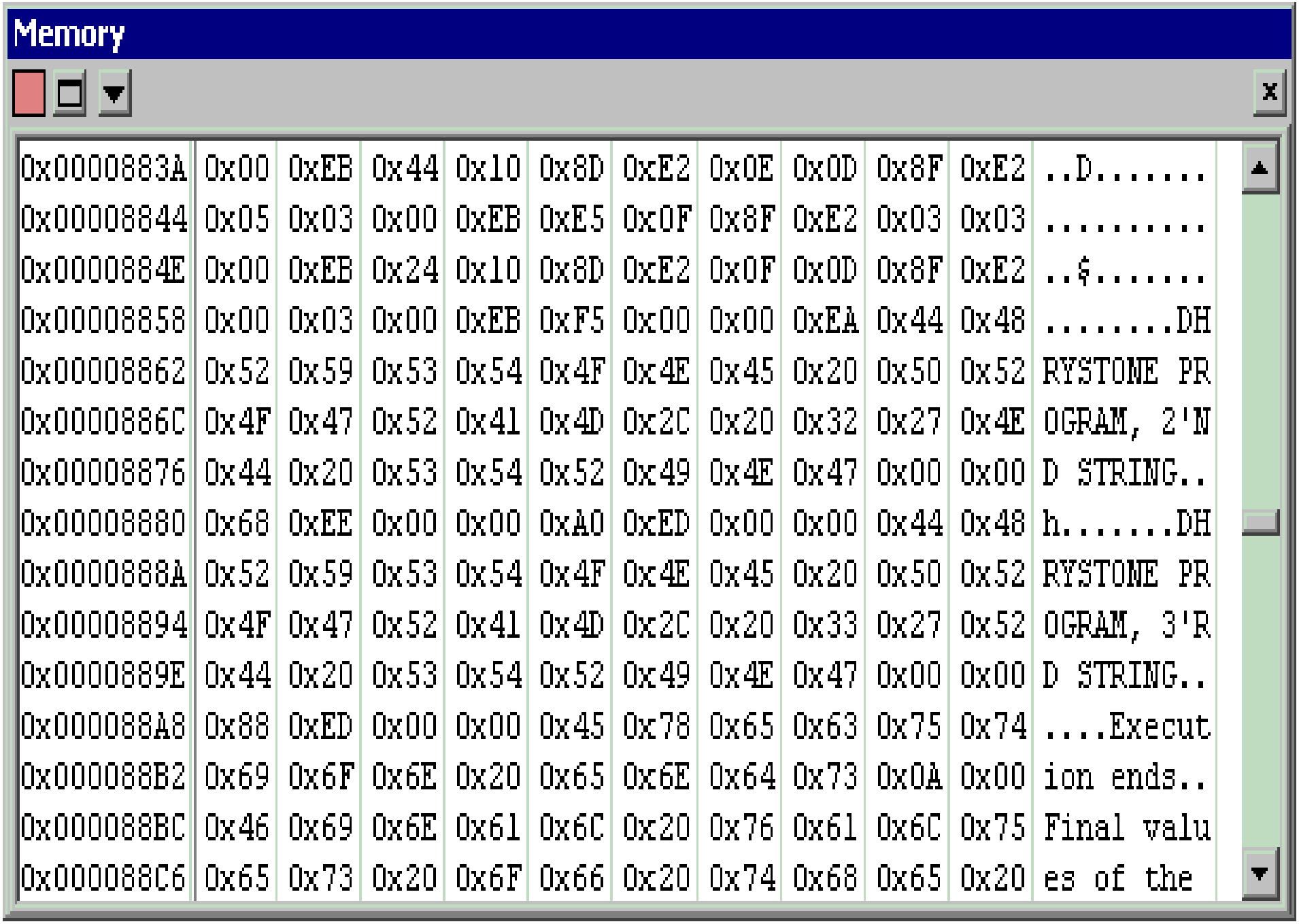
For example a computer with 1GB of RAM actually contains an array of

1024×1024×1024= 1,073,741,824 Bytes

0=0x00000000

1,073,741,824=0x3fffffff

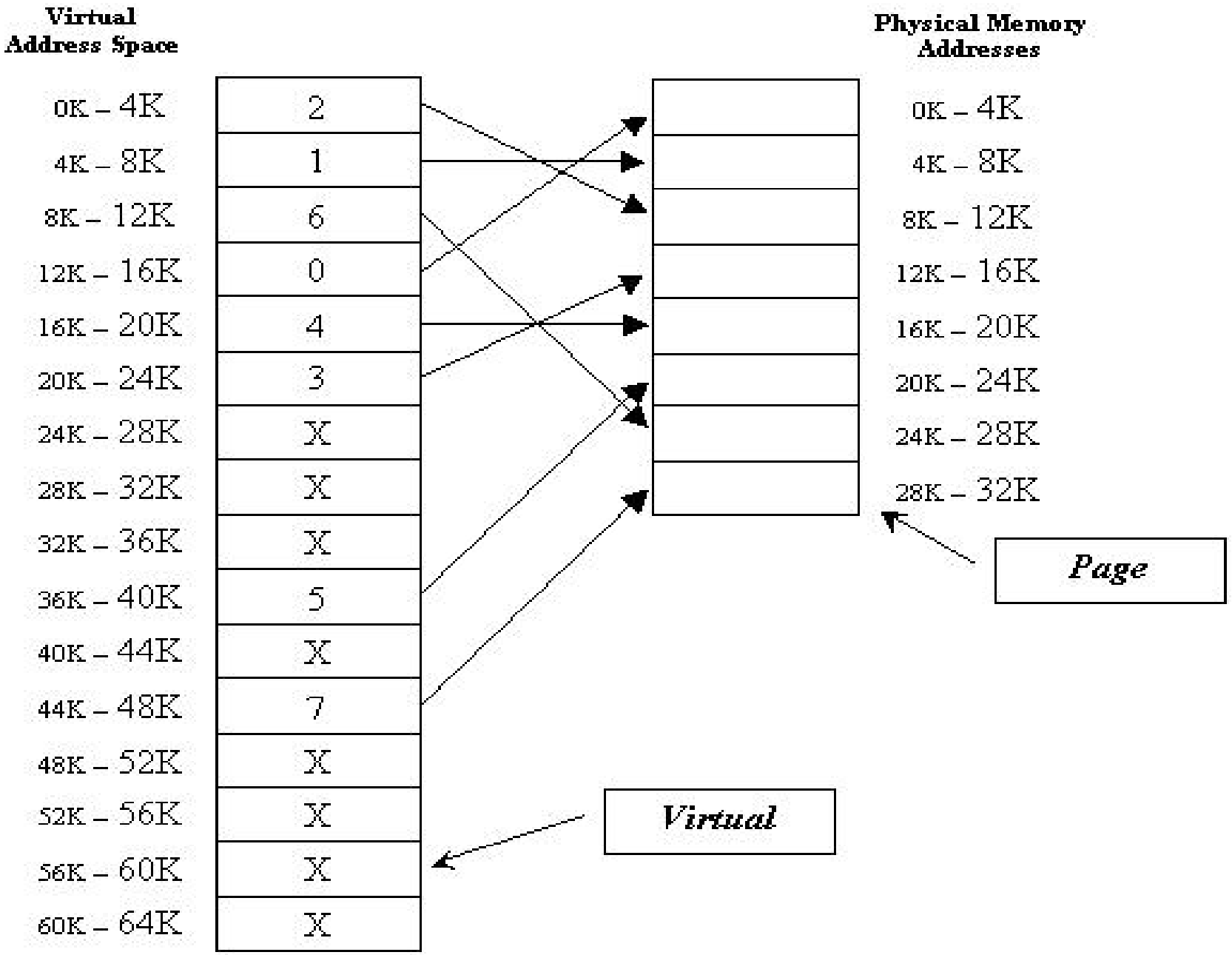



Memory
x
0x0000883A 0x00 0xEB 0x44 0x10 0x8D 0xE2 0x0E 0x0D 0x8F 0xE2 ..D.....
0x00008844 0x05 0x03 0x00 0xEB 0xE5 0x0F 0x8F 0xE2 0x03 0x03 ....
0x0000884E 0x00 0xEB 0x24 0x10 0x8D 0xE2 0x0F 0x0D 0x8F 0xE2 ..$.....
0x00008858 0x00 0x03 0x00 0xEB 0xF5 0x00 0x00 0xEA 0x44 0x48 ....DH
0x00008862 0x52 0x59 0x53 0x54 0x4F 0x4E 0x45 0x20 0x50 0x52 RYSTONE PR
0x0000886C 0x4F 0x47 0x52 0x41 0x4D 0x2C 0x20 0x32 0x27 0x4E OGRAM, 2'N
0x00008876 0x44 0x20 0x53 0x54 0x52 0x49 0x4E 0x47 0x00 0x00 D STRING..
0x00008880 0x68 0xEE 0x00 0x00 0xA0 0xED 0x00 0x00 0x44 0x48 h.....DH
0x0000888A 0x52 0x59 0x53 0x54 0x4F 0x4E 0x45 0x20 0x50 0x52 RYSTONE PR
0x00008894 0x4F 0x47 0x52 0x41 0x4D 0x2C 0x20 0x33 0x27 0x52 OGRAM, 3'R
0x0000889E 0x44 0x20 0x53 0x54 0x52 0x49 0x4E 0x47 0x00 0x00 D STRING..
0x000088A8 0x88 0xED 0x00 0x00 0x45 0x78 0x65 0x63 0x75 0x74 ....Execut
0x000088B2 0x69 0x6F 0x6E 0x20 0x65 0x6E 0x64 0x73 0x0A 0x00 ion ends..
0x000088BC 0x46 0x69 0x6E 0x61 0x6C 0x20 0x76 0x61 0x6C 0x75 Final valu
0x000088C6 0x65 0x73 0x20 0x6F 0x66 0x20 0x74 0x68 0x65 0x20 es of the

Memory Organization Example

- **Example memory contents:**
  - A memory with 3 address bits & 8 data bits has:
  - k = 3 and n = 8 so $2^3 = 8$ addresses labeled 0 to 7.
  - $2^3 = 8$ words of 8-bit data

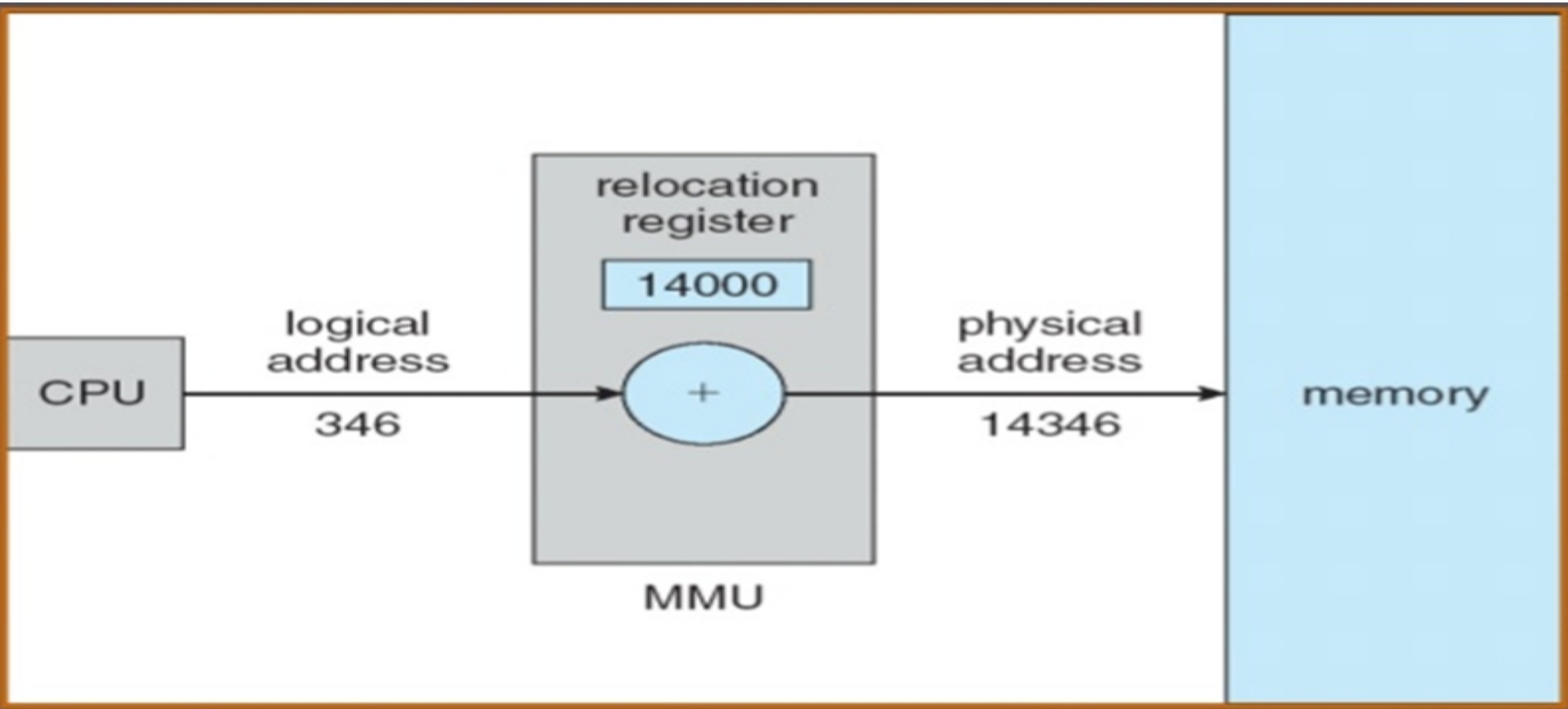
Memory Address Binary	Decimal	Memory Content
0 0 0	0	1 0 0 0 1 1 1 1
0 0 1	1	1 1 1 1 1 1 1 1
0 1 0	2	1 0 1 1 0 0 0 1
0 1 1	3	0 0 0 0 0 0 0 0
1 0 0	4	1 0 1 1 1 0 0 1
1 0 1	5	1 0 0 0 0 1 1 0
1 1 0	6	0 0 1 1 0 0 1 1
1 1 1	7	1 1 0 0 1 1 0 0

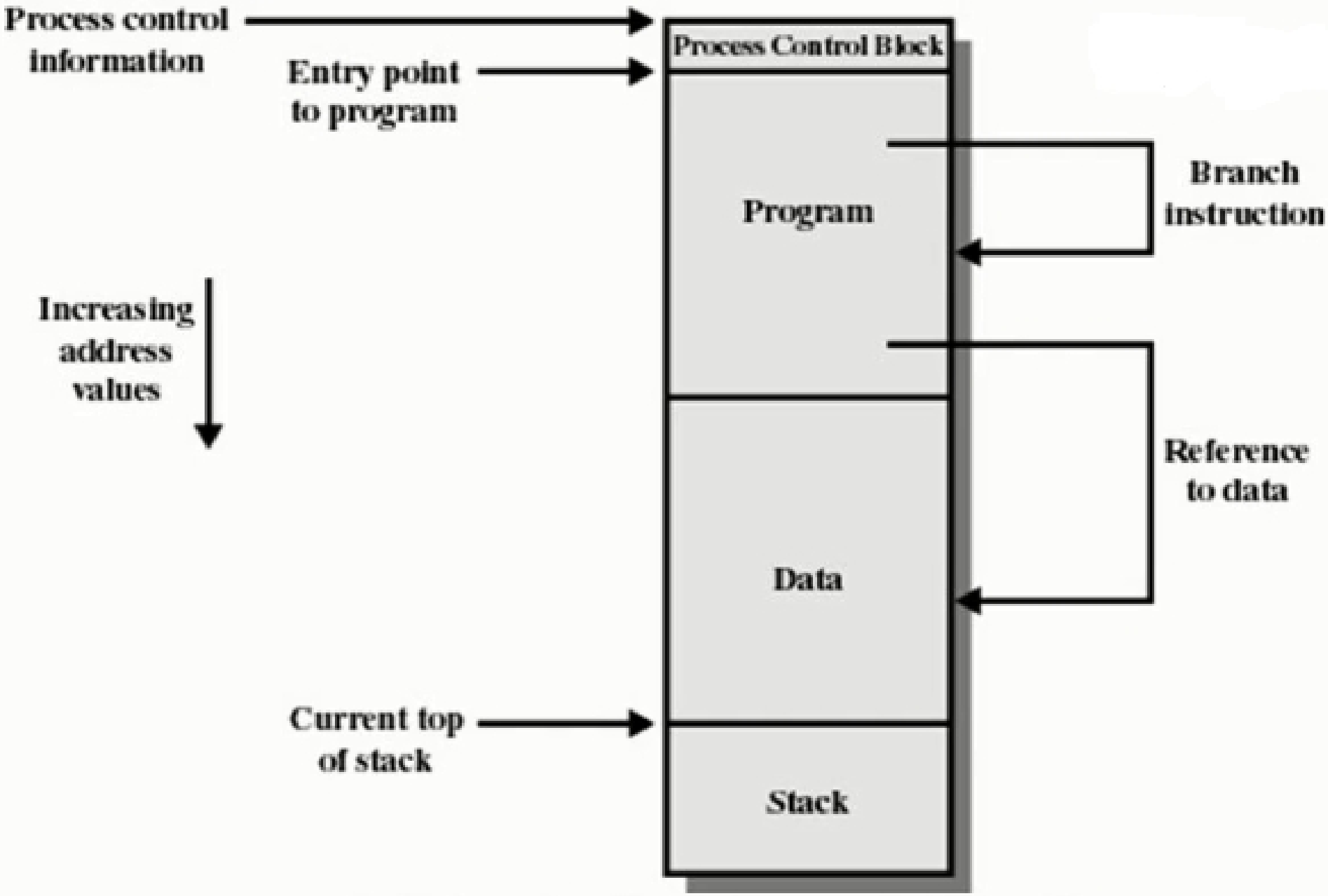

Virtual Address Space
0K – 4K
4K – 8K
8K – 12K
12K – 16K
16K – 20K
20K – 24K
24K – 28K
28K – 32K
32K – 36K
36K – 40K
40K – 44K
44K – 48K
48K – 52K
52K – 56K
56K – 60K
60K – 64K
2
1
6
0
4
3
X
X
X
5
X
7
X
X
X
X
Physical Memory Addresses
0K – 4K
4K – 8K
8K – 12K
12K – 16K
16K – 20K
20K – 24K
24K – 28K
28K – 32K
Page
Virtual

Memory Management

- **Relocation**
  - In a multiprogramming/multitasking environment, the processes are allocated memory space as and when they are loaded into memory.
  - A relocatable program is compiled and linked-in internal references are made relative to the address of the statement at its entry point.
  - The program also contains a table of information about its statements which are ‘address sensitive’. This table is called the **relocation table.**

- **Relocation**
  - Programmer does not know where the program will be placed in memory when it is executed
  - While the program is executing, it may be swapped to disk and returned to main memory **at a different location** (relocated)
  - Memory **references must be translated** in the code to actual physical memory address



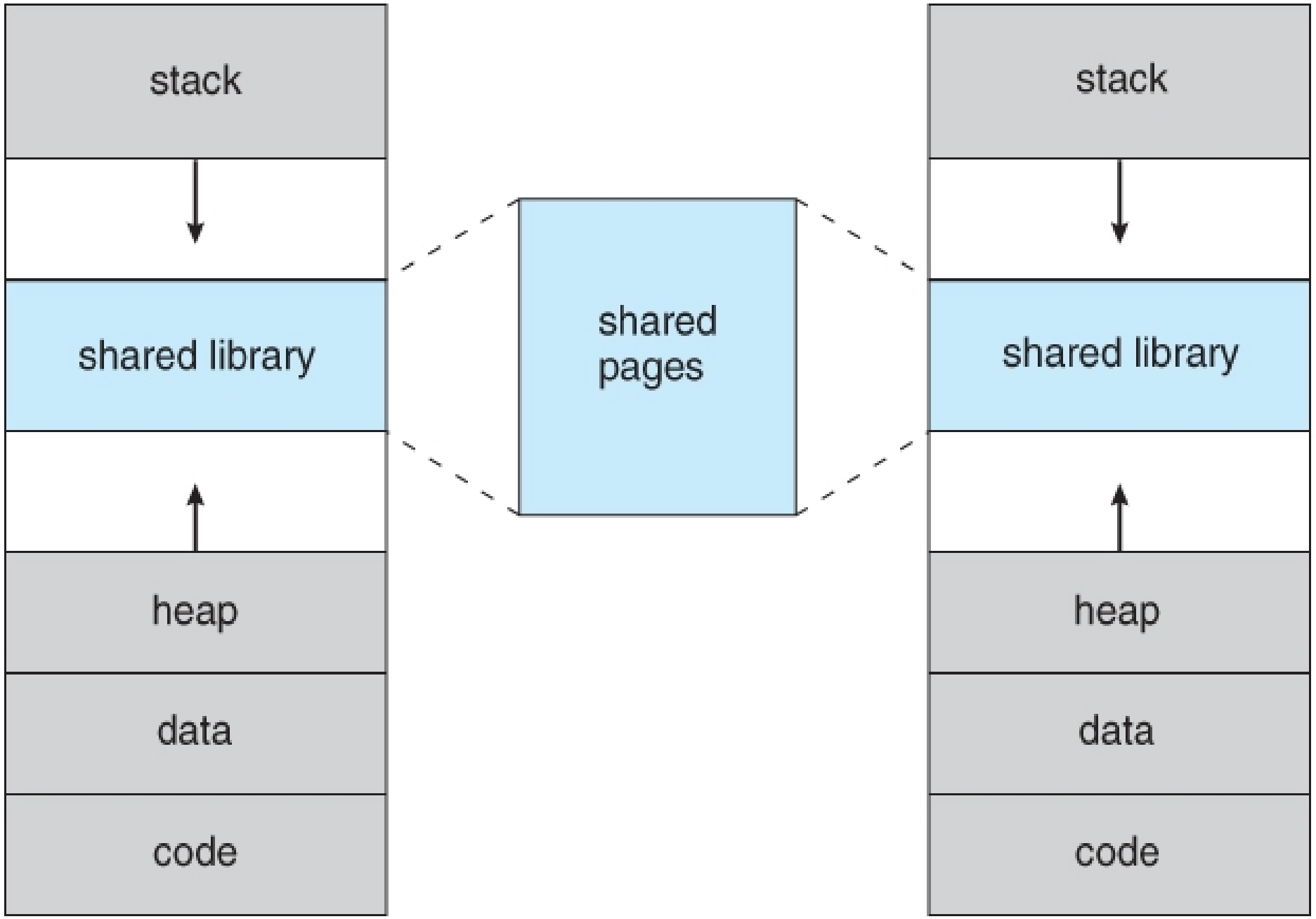


Addressing Requirements for a Process

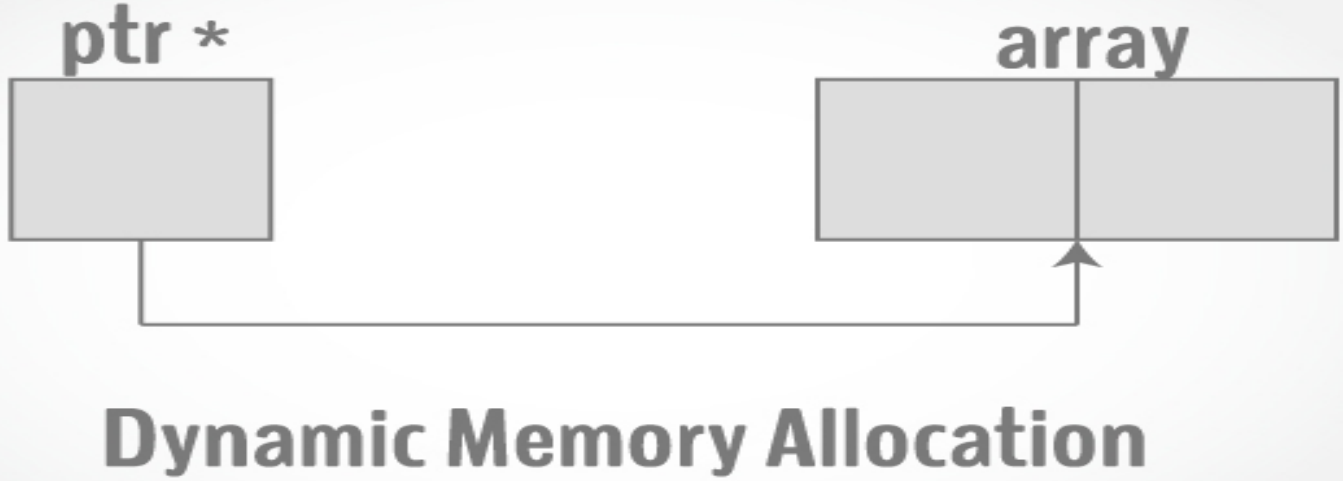
Memory Management

- **Sharing**
  - The **system programs such as compilers and their libraries (stdio.h)** need to be shared by multiple processes.
  - The operating system ensures that access to shared applications, pages or databases is allowed to processes belonging to different users or groups, without compromising the integrity of the data, information or pages of the participating documents.

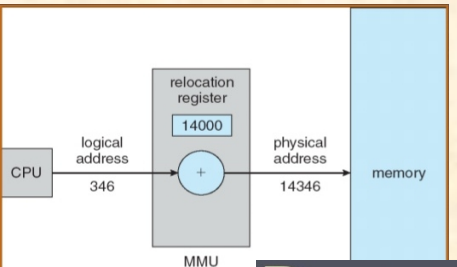
1. **Sharing –** A protection mechanism that have to allow several processes to access the same portion of main memory.
2. Allowing each processes access to the same copy of the program rather than have their own separate copy has an advantage.
   For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes [eg., *header files in C*]
3. It is the task of Memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.


stack
shared library
heap
data
code
shared pages
stack
shared library
heap
data
code

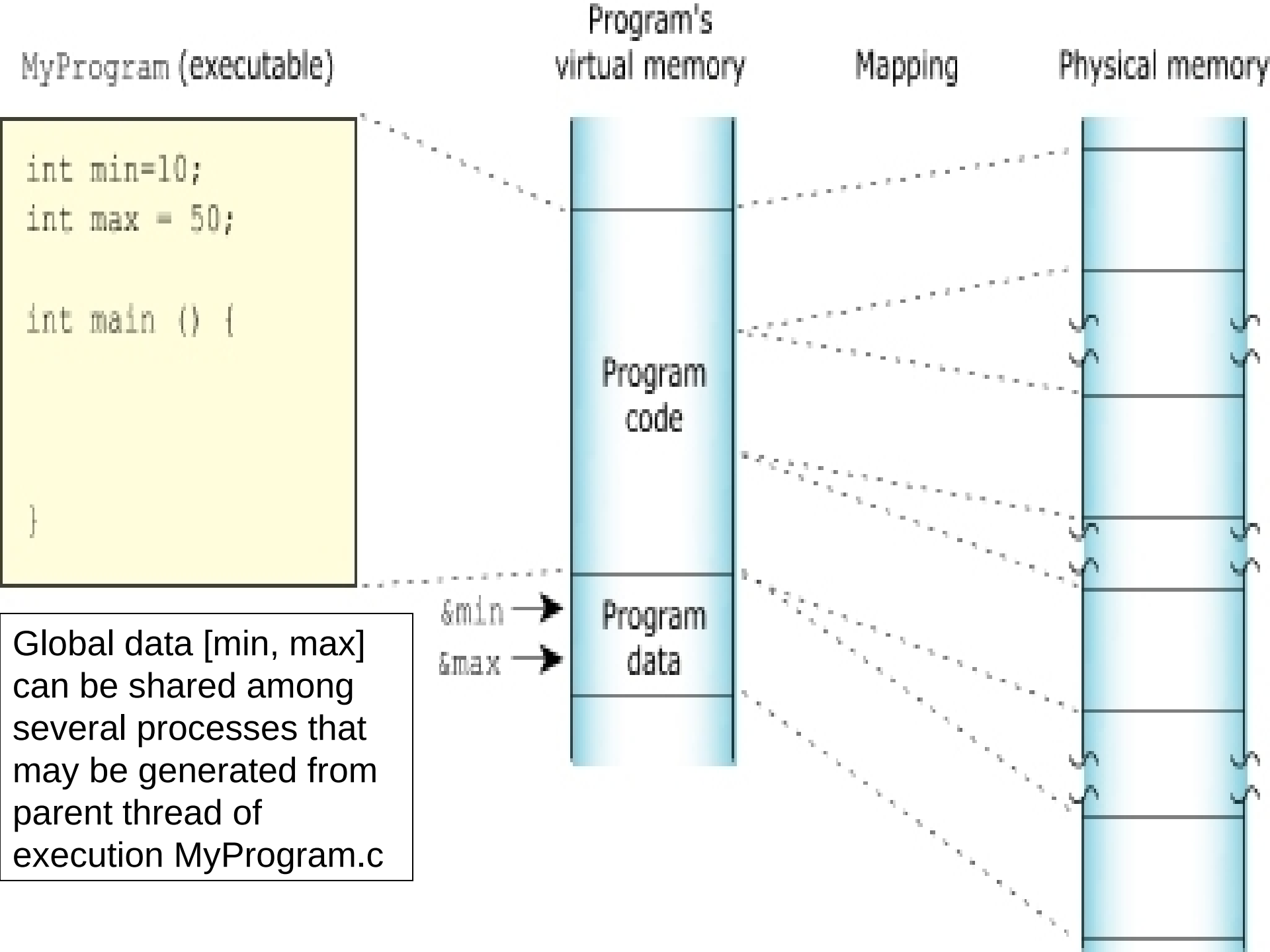
Dynamic memory allocation



Dynamic memory allocation means to allocate the memory at run time.
dynamic memory allocation is possible by 4 functions of stdlib.h header file: malloc(), calloc(), realloc(), free()



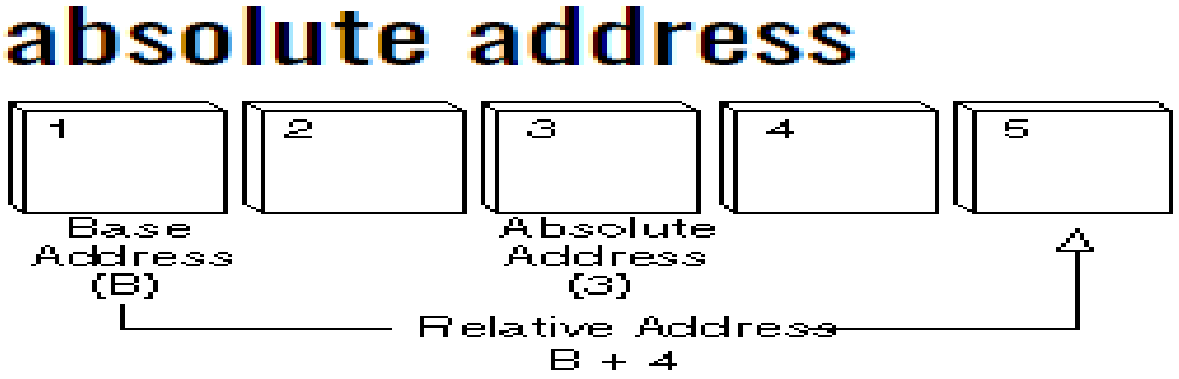
Program relocation



Global data [min, max] can be shared among several processes that may be generated from parent thread of execution MyProgram.c

Memory Management

- **Protection**
  - The multiple resident processes frequently access their files and other resources.
  - It becomes the responsibility of the operating system to manage the memory space in such a manner that no process, accidentally or otherwise, reads from or writes into the pages allotted to other processes.



A fixed address in memory. The term *absolute* distinguishes it from a *relative address*, which indicates a location by specifying a distance from another location. Absolute addresses are also called *real addresses* and *machine addresses*.

Next slide explains what is absolute and relative memory addresses

Logical and physical addresses

- The processor at compile time generates some addresses known as ***logical addresses***. The set of all logical addresses generated by the compilation of the process is known as ***logical address space***.

- Logical addresses need to be converted into ***absolute addresses*** at the time of execution of the ***process***. These absolute addresses are known as ***physical addresses***. The set of physical addresses generated corresponding to all logical

1. The main memory accommodates :

a) operating system **b) CPU**

c) user processes **d) All of these**

Ans: a & c

2. Which of the following is/are the requirements of memory management.

i) Relocation ii) Protection iii) Sharing iv) Memory organization

A) i, ii and iii only **B) ii, iii and iv only**

C) i, iii and iv only **D) All i, ii, iii and iv**

Ans: A)

3. Main memory in a computer system is ……. as a linear or one dimensional, address space, consisting of a sequence of bytes or words.

A) relocated **B) protected** **C) shared** **D) organized**

Ans: d)

4. Once a program is compiled, it can be loaded for execution

a. Only from the compiler generated starting address

b. Any where in the main memory

c. User needs to specify where the compiled code is to be loaded

d. It is loaded starting from address 0 in the main memory.

Ans: a)

Memory Management

1. CPU fetches the instruction from memory according to the value of

a) program counter
b) status register
c) instruction register
d) program status word

Answer: a

2. Which one of the following is the address generated by CPU?

a) physical address
b) absolute address
c) logical address
d) none of the mentioned

Answer: c

Memory Management

4. Run time mapping from virtual to physical address is done by

a) memory management unit

b) CPU

c) PCI

d) none of the mentioned

Answer: a

5. The main memory accommodates :

a) operating system

b) CPU

c) user processes

d) All of these

Answer: a and c

Memory Management

5. Program always deals with
a) logical address
b) absolute address
c) physical address
d) relative address

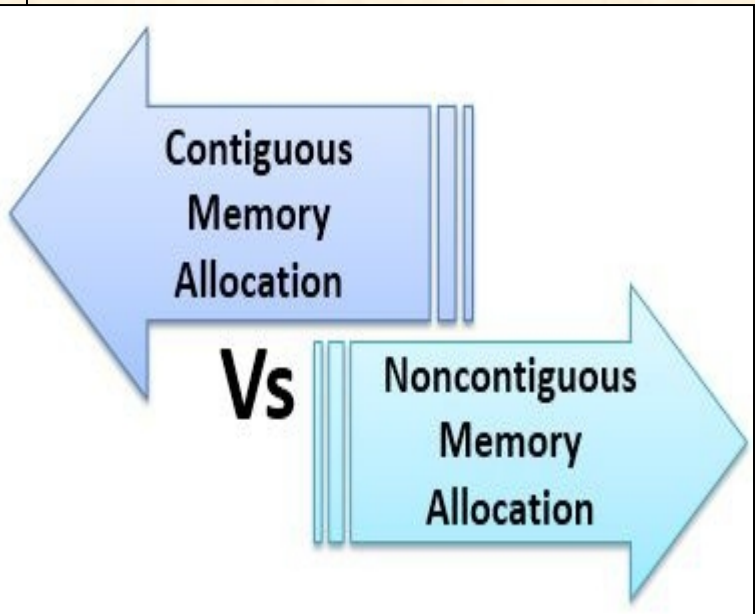
Answer: a

6. The operating system and the other processes are protected from being modified by an already running process because :
a) they are in different memory spaces
b) they are in different logical addresses
c) they have a protection algorithm
d) every address generated by the CPU is being checked against the relocation and limit registers

Answer: d

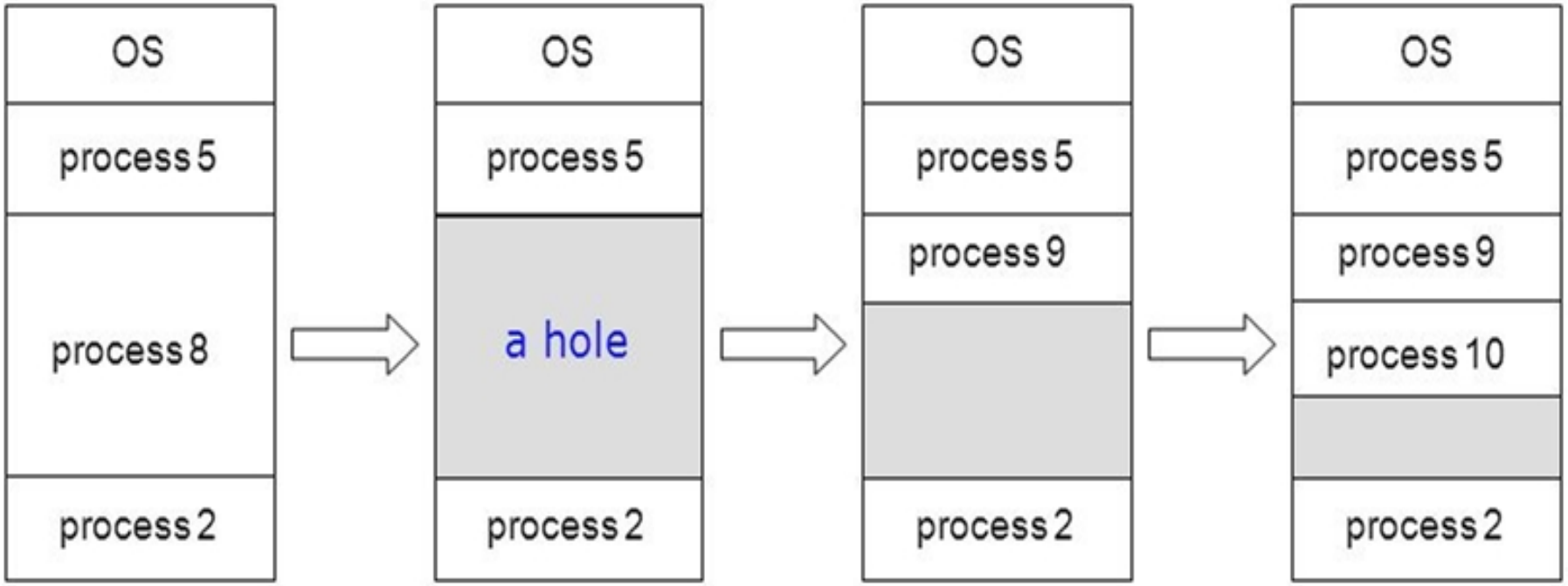
Contiguous memory allocation

- The contiguous allocation method was implemented by partitioning the memory into various regions.
- The memory partition which is free to allocate is known as a *hole*. Thus an is searched for allocating cess.

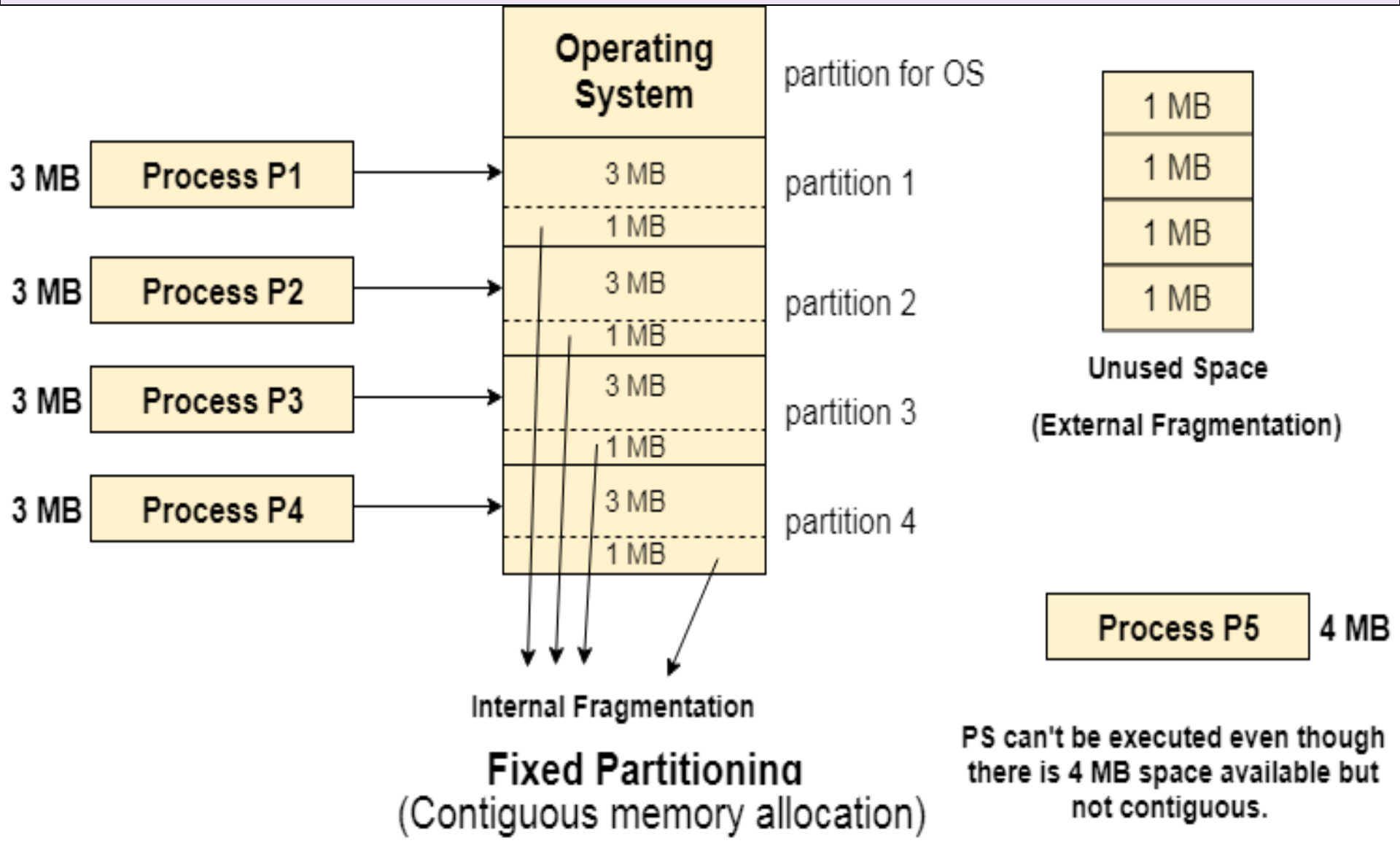


Contiguous Memory Allocation (Cont)

- Multiple-partition allocation
  - Hole – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Operating system maintains information about: a) allocated partitions b) free partitions (holes)



- The simplest way is to divide the memory into partitions:
  - Fixed sized partitions
  - Variable sized partitions

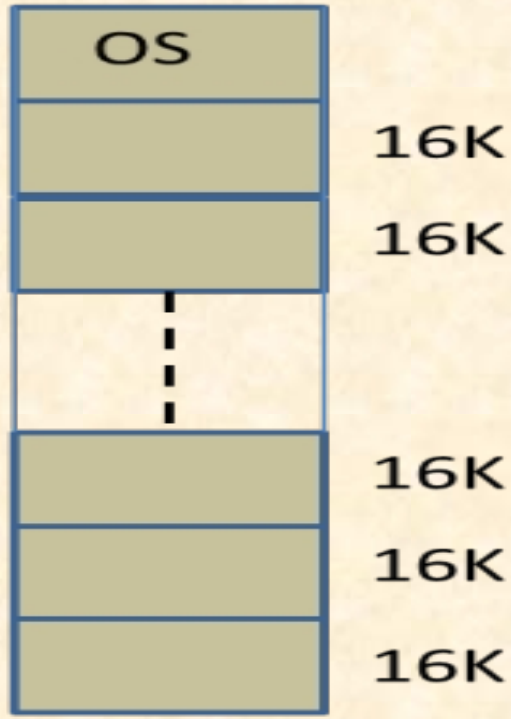


Contiguous allocation with fixed partitioning

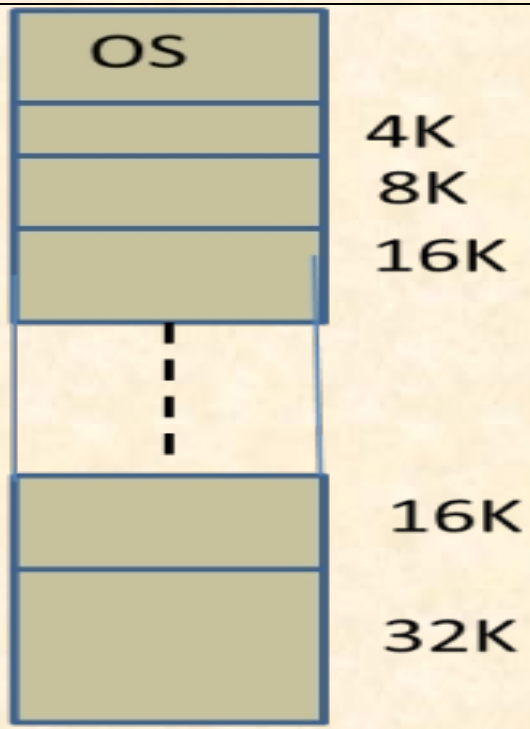
- Fixed partitioning is the method of partitioning the memory at the time of system generation.
- The partition size can be of fixed as well as variable but once fixed cannot be changed.

Fixed Sized Partitions

- Fixed size partitions can be of two types:
  - Equal sized partitions
  - Unequal sized partitions



(a) Equal sized



(b) Unequal sized

Equal Sized Partitions

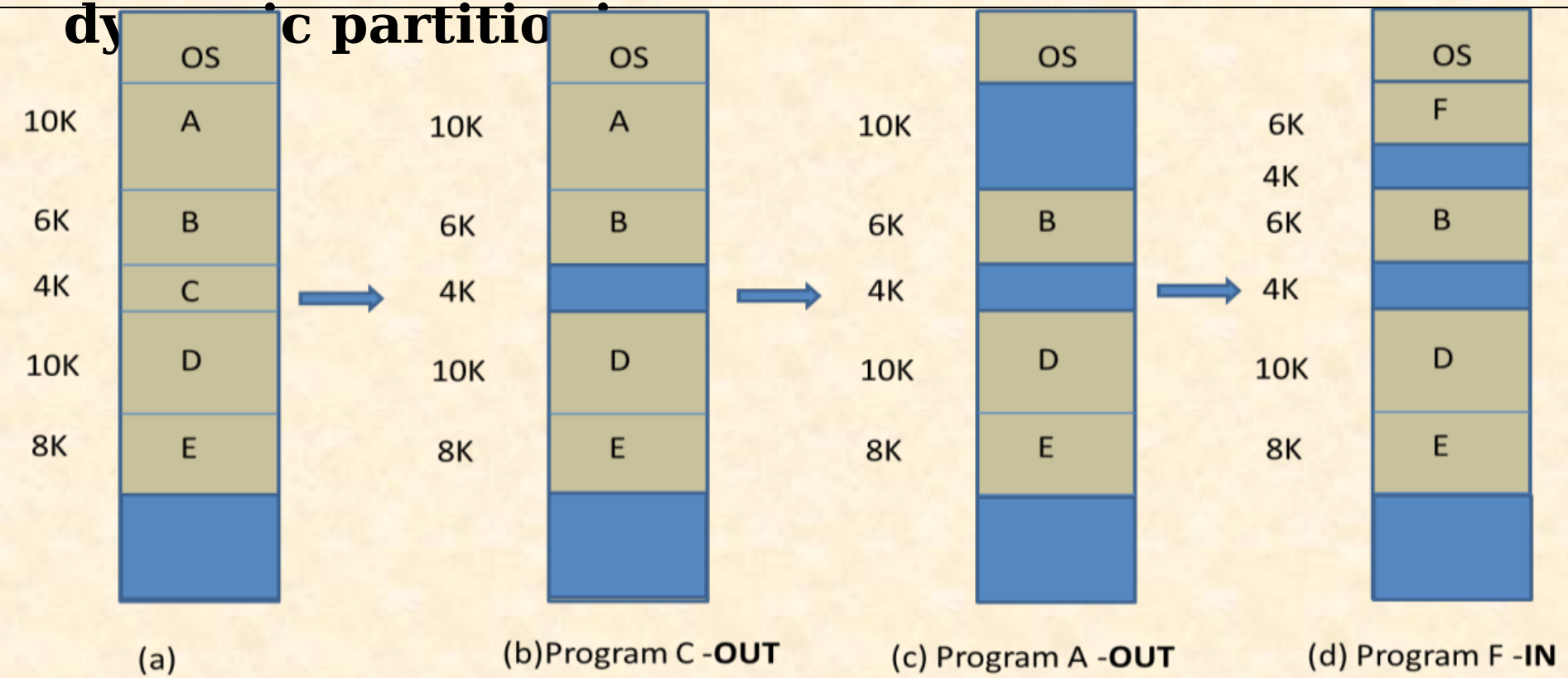
- Here, the operating system divides the main memory into equal sized partitions.
  - **Disadvantages**
    - The smaller the size of the program to be loaded, the greater the wastage of memory.
    - This wastage of memory within a partition is called **internal fragmentation**.

How to Reduce Internal Fragmentation?

- **Method 1**: Reduce the partition size
  - The reduced partition size places a limit on the size of program that can be loaded into the partition.
- **Method 2**: Divide the memory into unequal sized partitions
  - In this case, the operating system uses the **best fit policy** to load processes into main memory so that there is least internal fragmentation.

Variable Sized Partitions

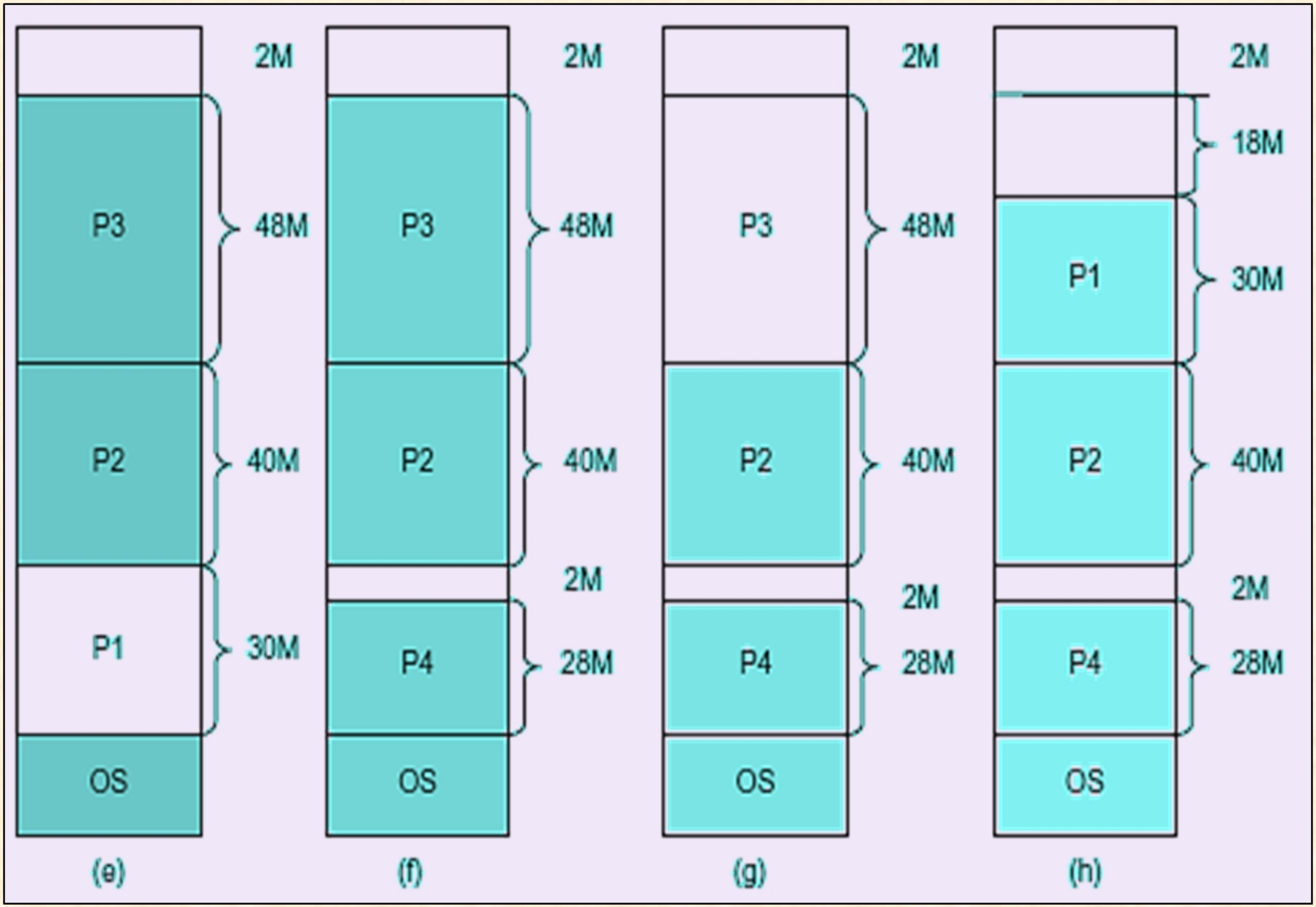
- Here, the programs are loaded into the memory in the order of their arrival.
- As the programs are of different sizes, the memory is dynamically divided into variable sized partitions. Therefore, this scheme of partitioning is also called **dy c partitio**



Contiguous allocation with dynamic/variable partitioning

- Instead of having static partitions in the memory the memory partition will be allocated to a process dynamically.
- As the processes dynamically occupy and release the memory, a time is reached when there are small holes generated in the memory partitions. These memory partitions at a certain time cause the ***external fragmentation***.

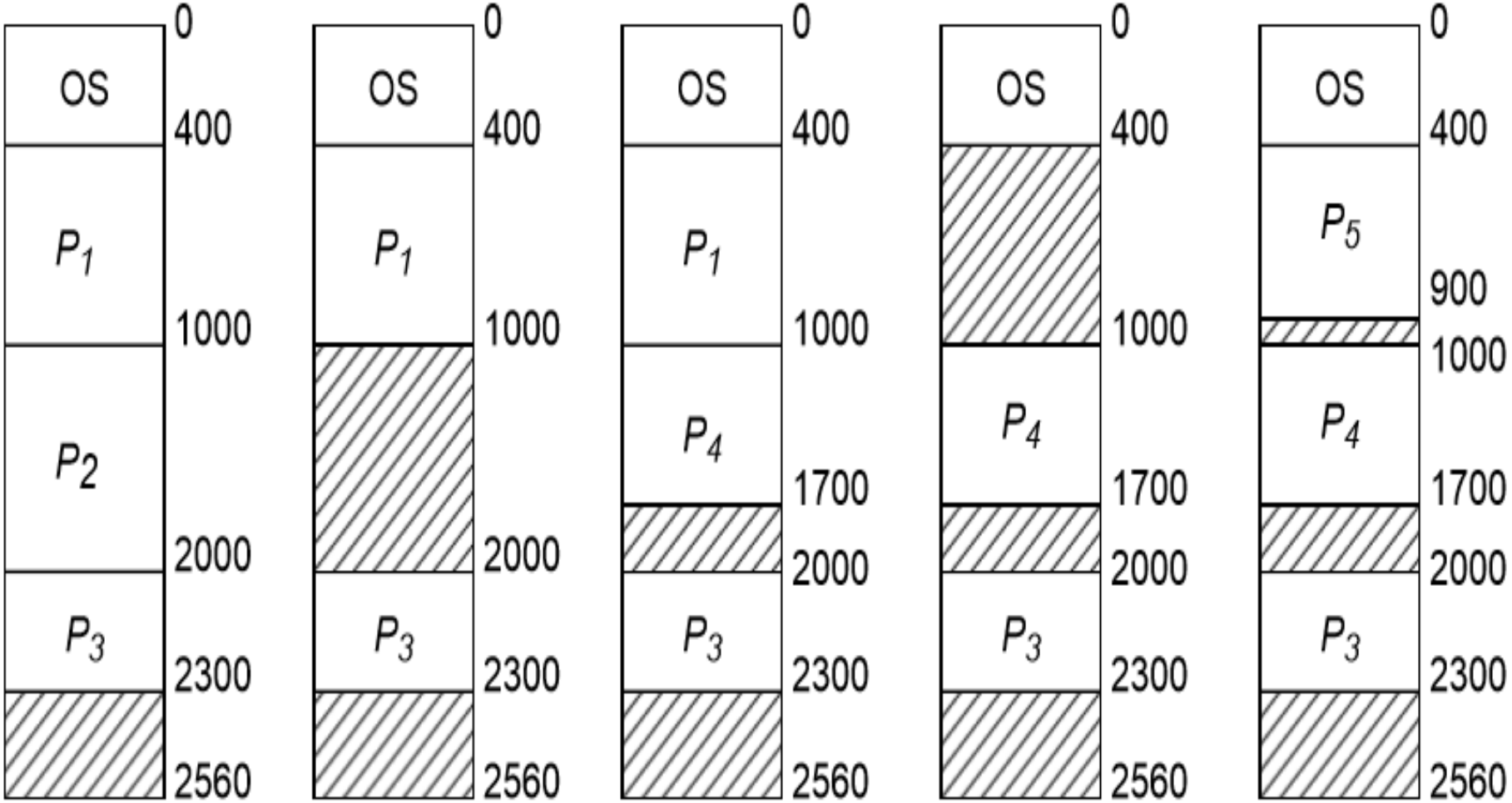
Variable Partitioning with external fragmentation

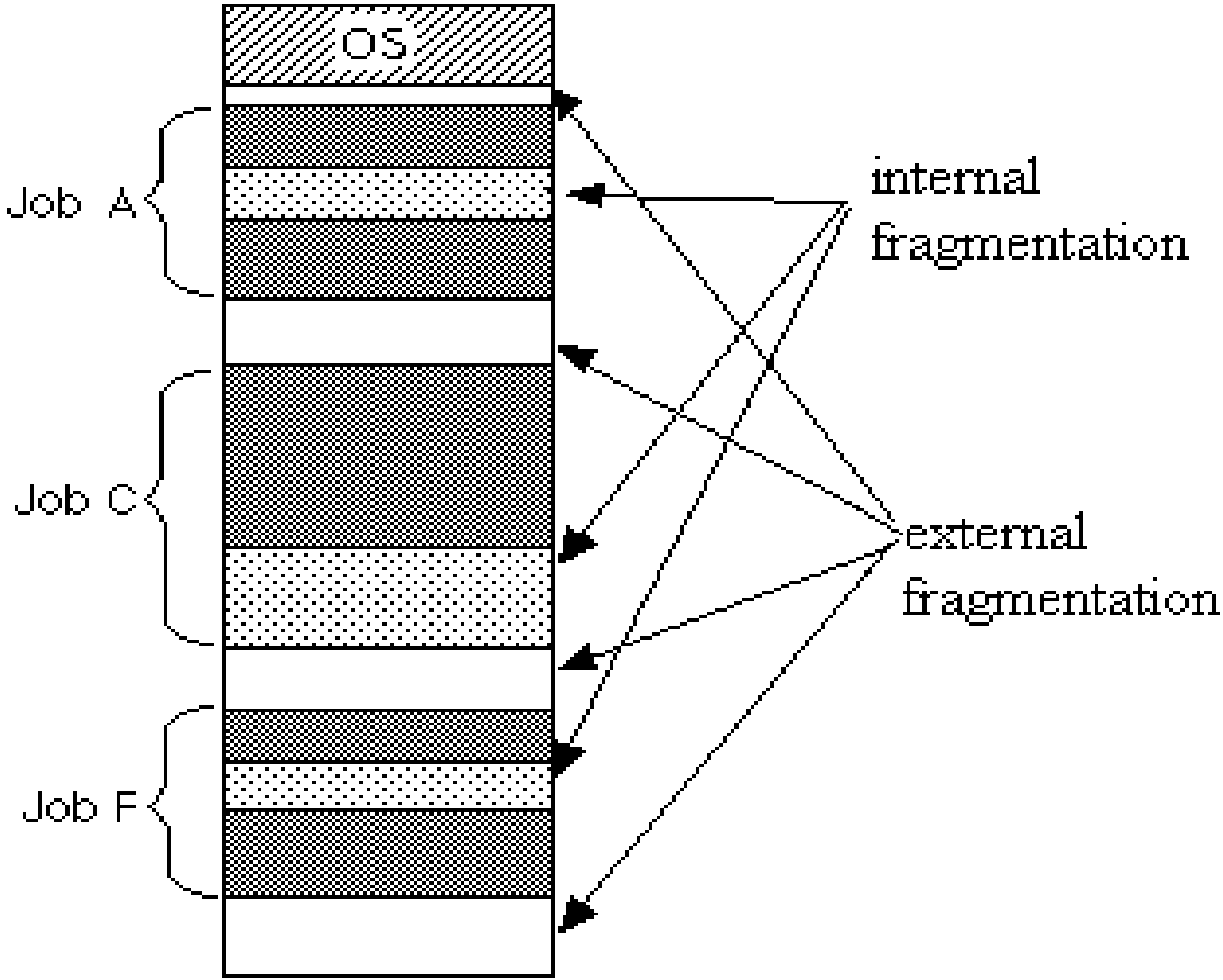


Example:

Process sizes:

P_1 600 P_2 1000 P_3 300 P_4 700 P_5 500



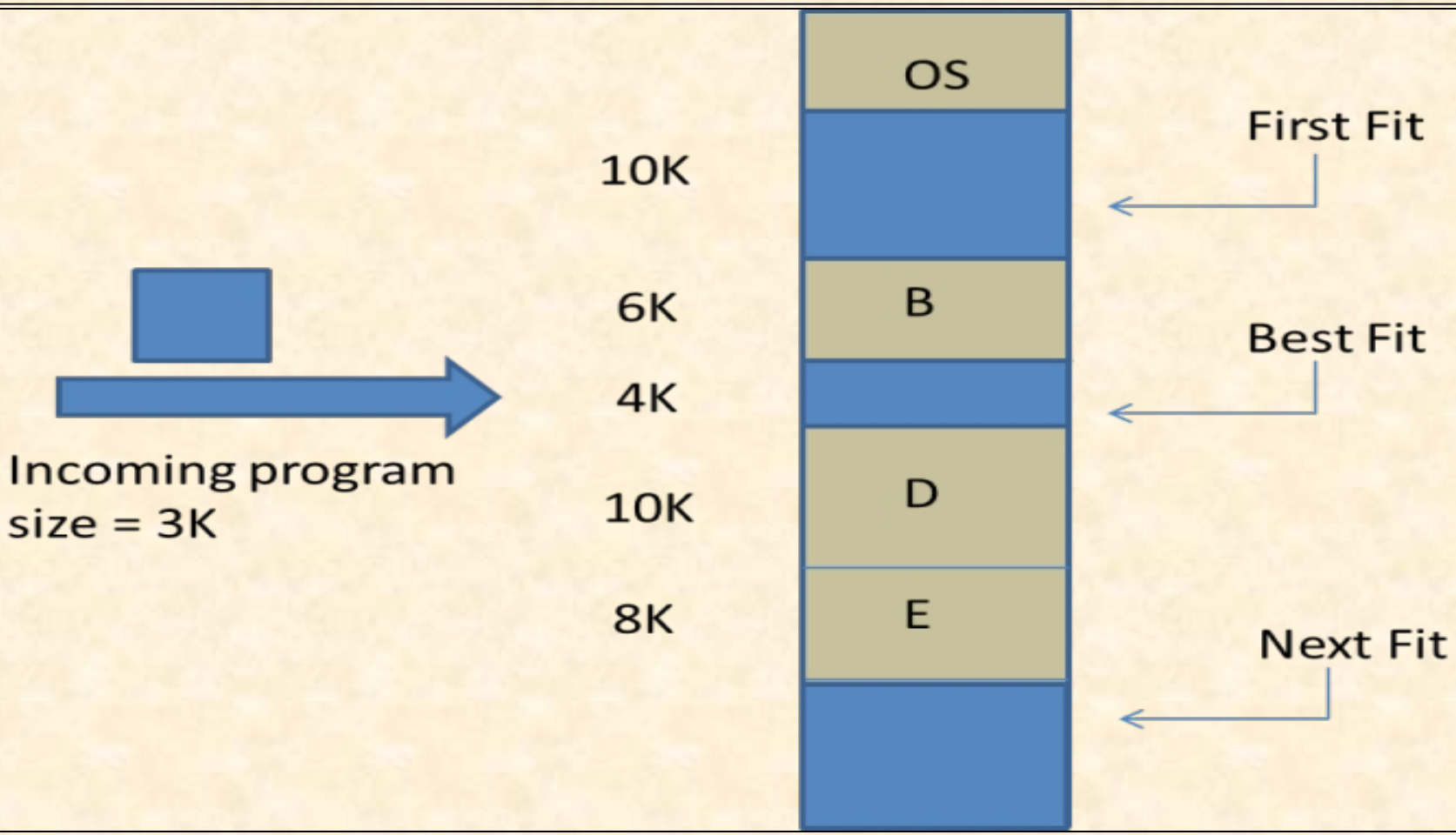

OS
Job A
Job C
Job F
internal
fragmentation
external
fragmentation

Loading Policies

- External fragmentation happens and adapts using any of the following loading policies to load the incoming programs in MM:
  - First fit policy
  - Best fit policy
  - Next fit policy

Loading Policies: First Fit Policy

- The first free memory block, from the start of the memory, whose size is more than equal to the size of incoming program is found.



Loading Policies: Best Fit Policy

- The operating system searches for a free memory block whose size is just more than or equal to the size of the incoming program.
- As this policy spends some time searching for the best suited memory block, the system becomes slow.

Loading Policies: Next Fit Policy

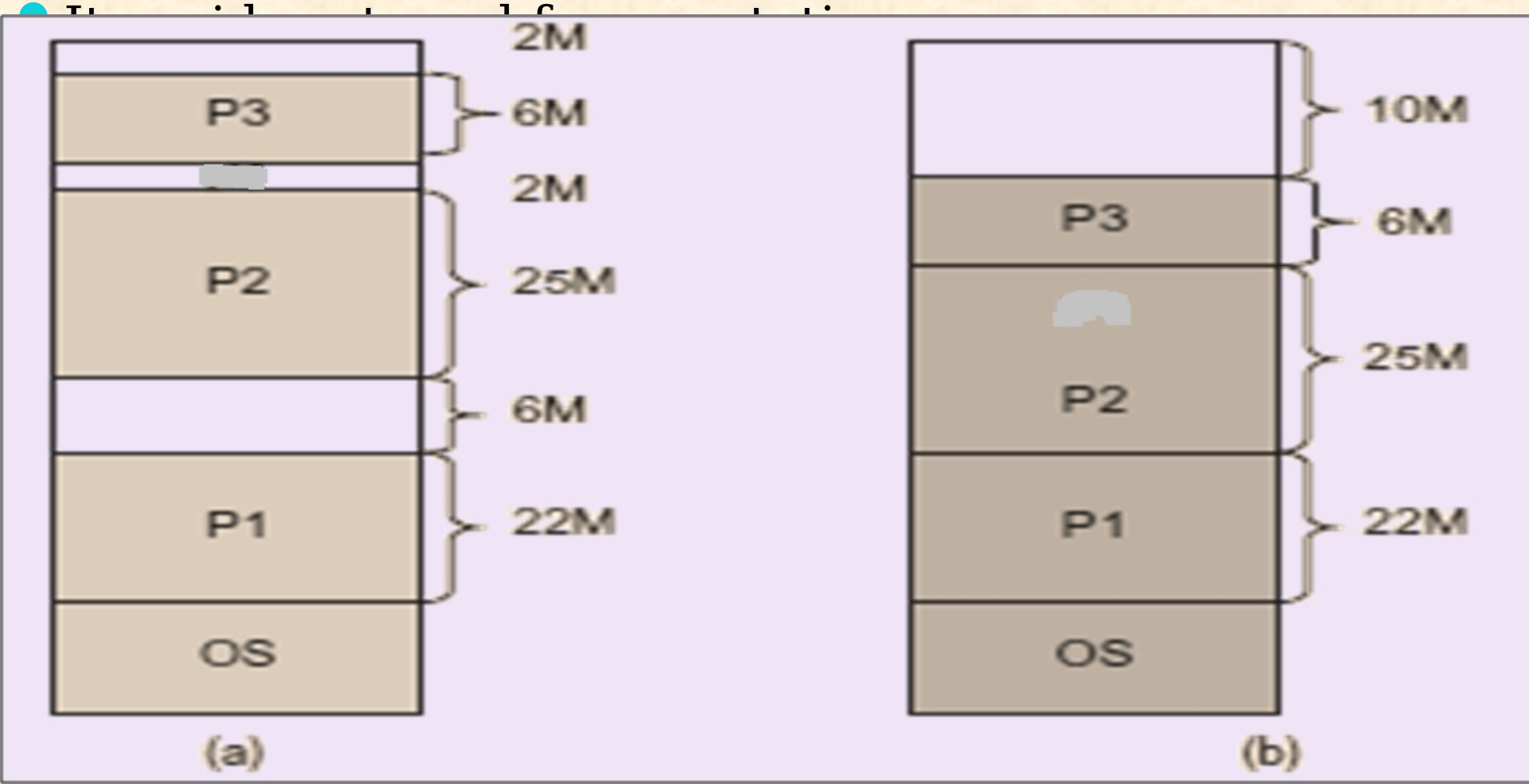
- The operating system keeps a pointer to the location where the last program was loaded.
- From that pointer onwards, it finds the next free memory block and loads the incoming program.

Variable Sized Partitions

- The type of memory fragmentation created by outgoing programs is called as **external fragmentation**.
- **Solution**: The problem of external fragmentation is solved by moving the resident programs into a big chunk of contiguous programs and the fragmented memory into another chunk of free memory. This activity is known as **storage compaction.**

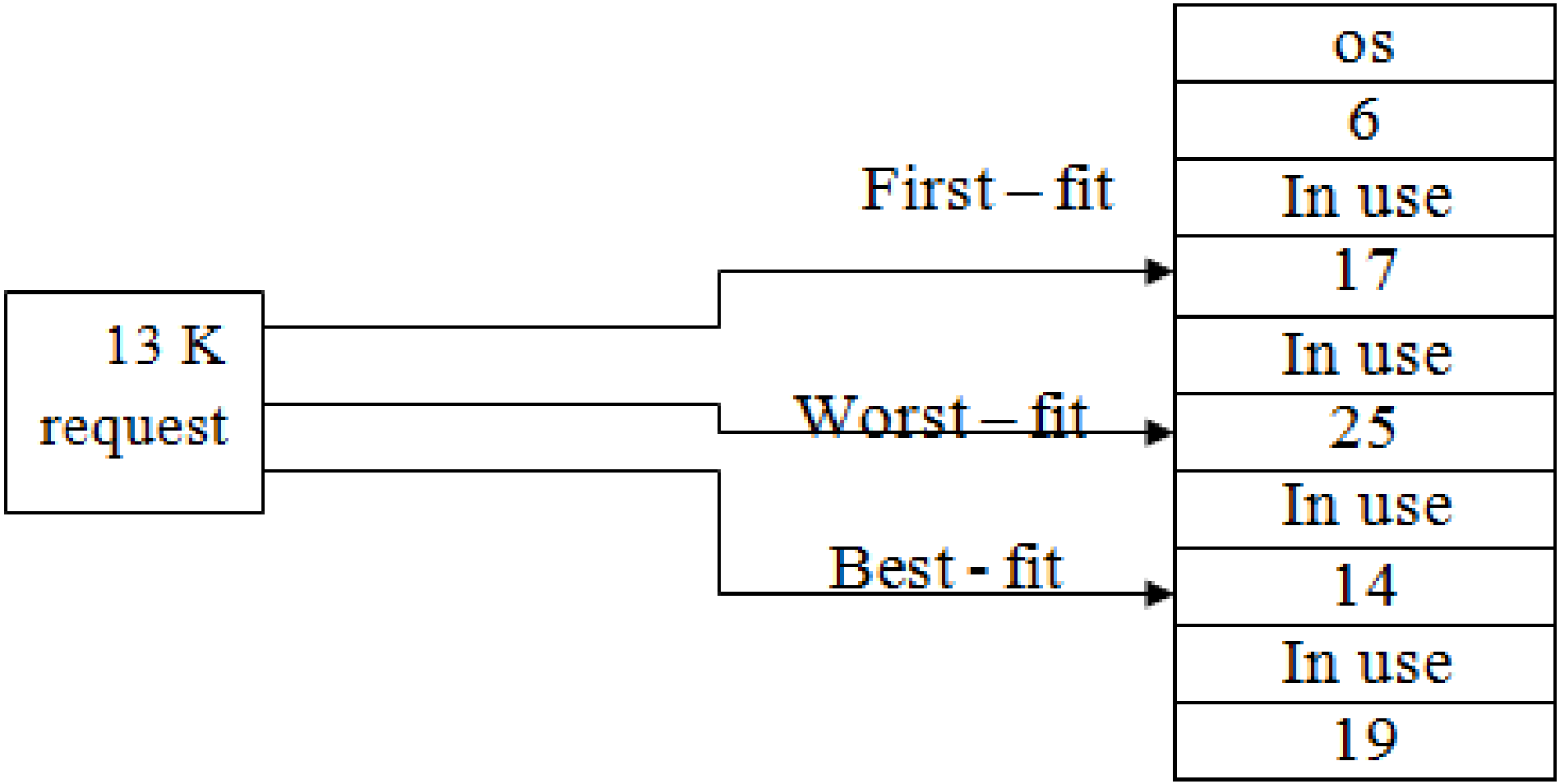
Storage Compaction

- Incoming programs can be accommodated in the free memory space created by storage compaction.
- It is a time-consuming exercise, leading to slowing of the system.



Assignments

Suppose that we have free segments with sizes: 6, 17, 25, 14, and 19. Place a program with size 13kB in the free segment using first-fit, best-fit and worst fit?



Assignments

Consider a segmentation-based system. At some point in the system operation, the main memory has the following ***free segment blocks*** in this order:
21K, 5K, 90K, 54K, 10K, 25K and 56K. There are three new requests for memory of sizes 10K, 7K and 22K. The system follows FCFS service for memory-allocation requests.
Explain which free segment blocks will be taken for (i) First-Fit and (ii) Best-Fit memory-allocation schemes.

Ans: **First-Fit**: The first two requests will be satisfied from the free segment block 21K, and the third request from the 90K free segment block. Final free segment block configuration is: **4K, 5K, 68K, 54K, 10K, 25K, and 56K**.
Best Fit: The first request will be satisfied from the 10K free segment block, the second request from the 21K free segment block, and the third request from the 25K free segment block. Final free segment blocks configuration is: **14K, 5K, 90K, 54K, 3K, and 56K.**

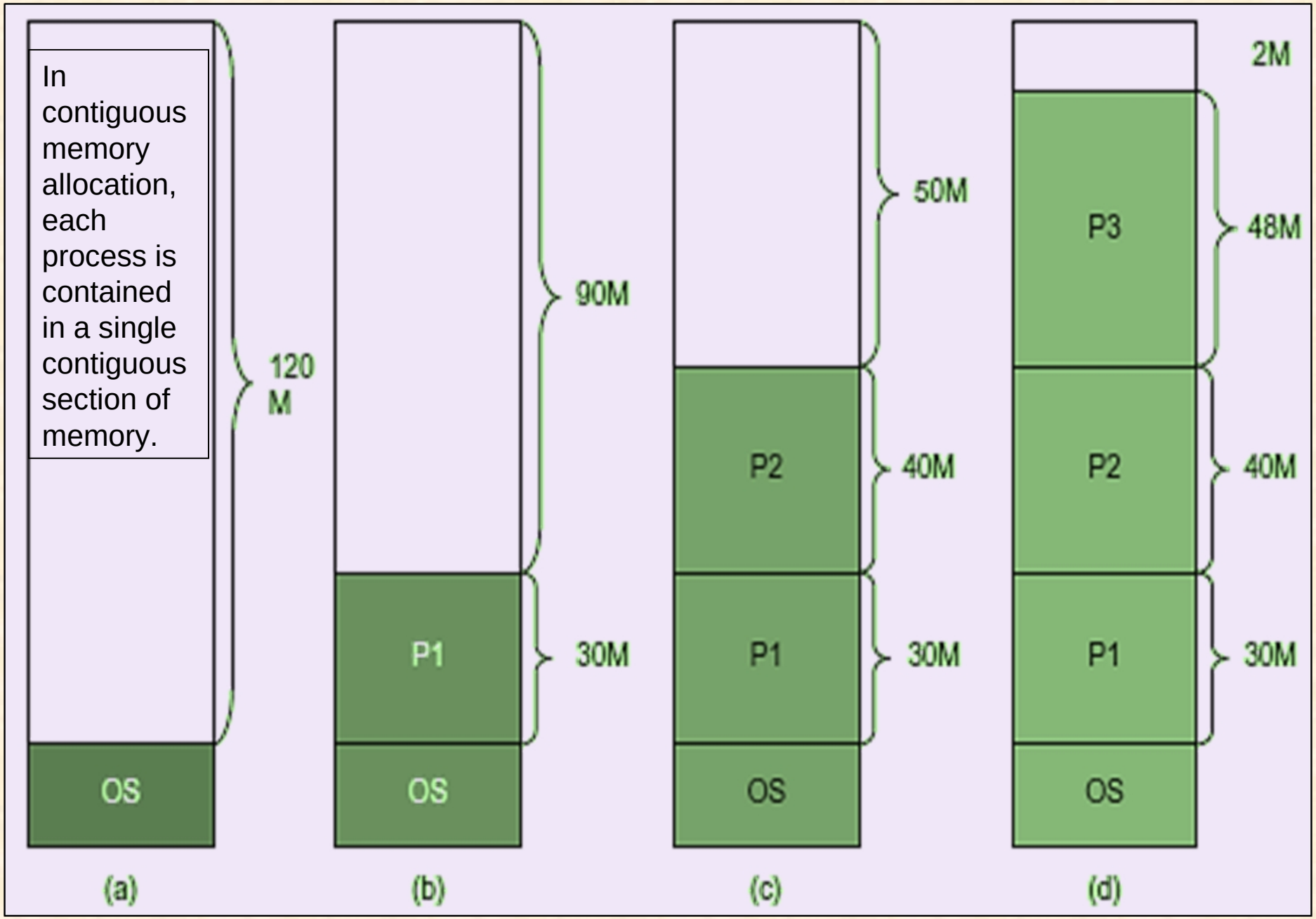
How and why is data in RAM partitioned?

Single contiguous allocation is the simplest memory management technique. All the computer's memory, usually with the exception of a small portion reserved for the operating system, is available to the single application. MS-DOS is an example of a system which allocates memory in this way.

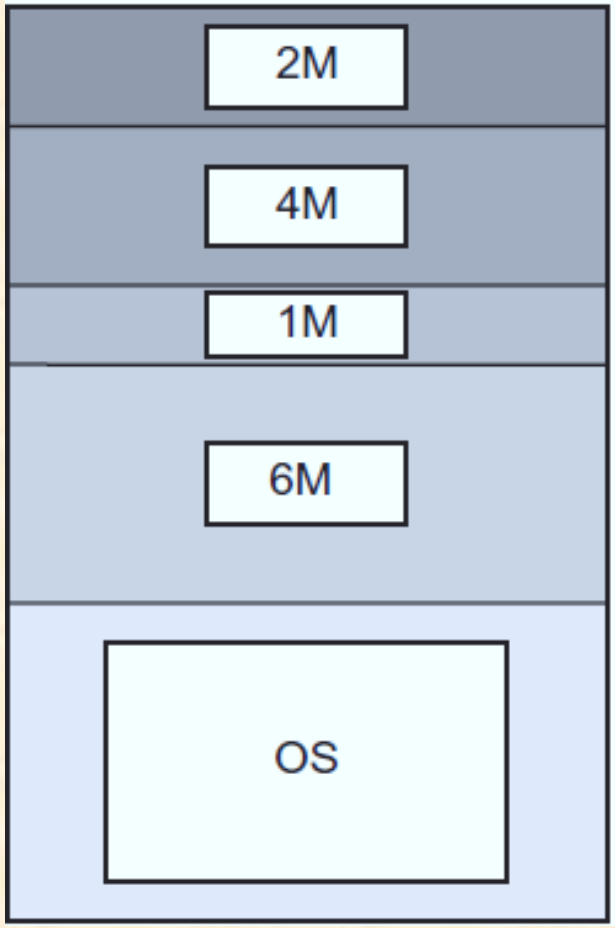
The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. We can place the operating system in either low memory or high memory.

We usually want several user processes to reside in memory at the same time. We therefore need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory.

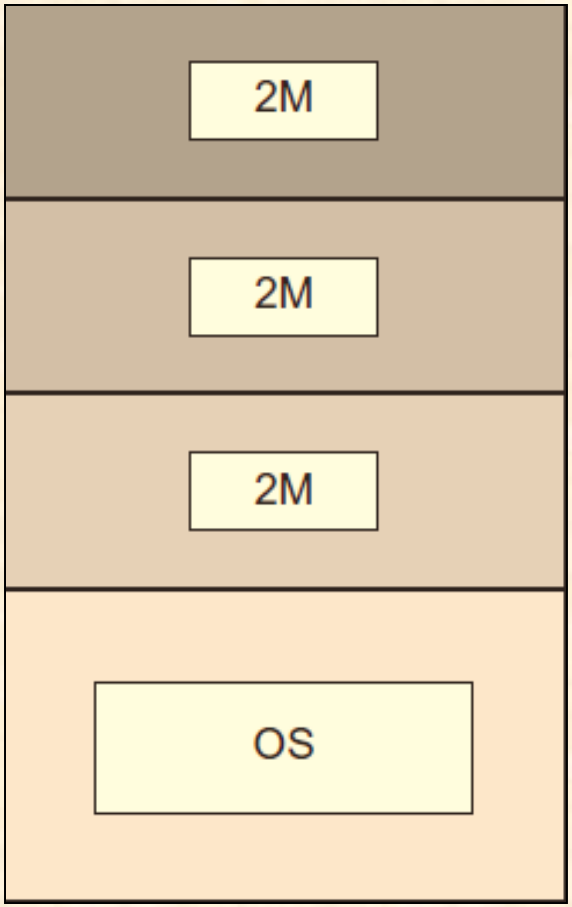
Single contiguous allocation



Fixed unequal sized partitioning based contiguous memory allocation



Fixed equal sized partitioning based contiguous memory allocation

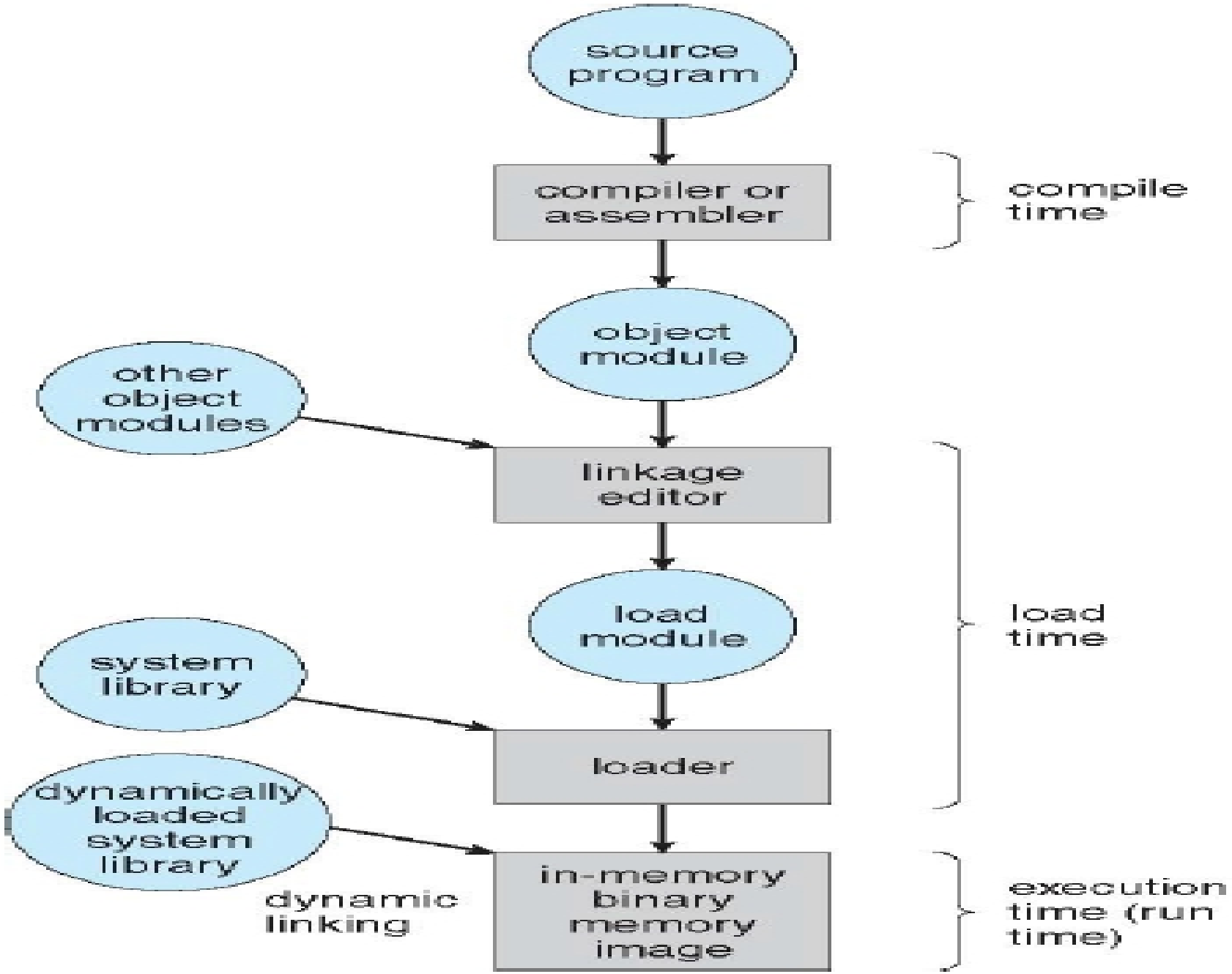


Protection – *There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program.*

1. *So every process must be protected against unwanted interference when other process tries to write in a process whether accidental or incidental.*
2. *Prediction of the location of a program in main memory is not possible, that's why it is impossible to check the absolute address at compile time to assure protection*.
3. *Most of the programming language allows the dynamic calculation of address at run time.*
4. *The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system can hardly control a process when it occupies the processor.*
5. Thus it is possible to check the validity of memory references.

https://en.wikipedia.org/wiki/Memory_management_(operating_systems)#Memory_management_techniques

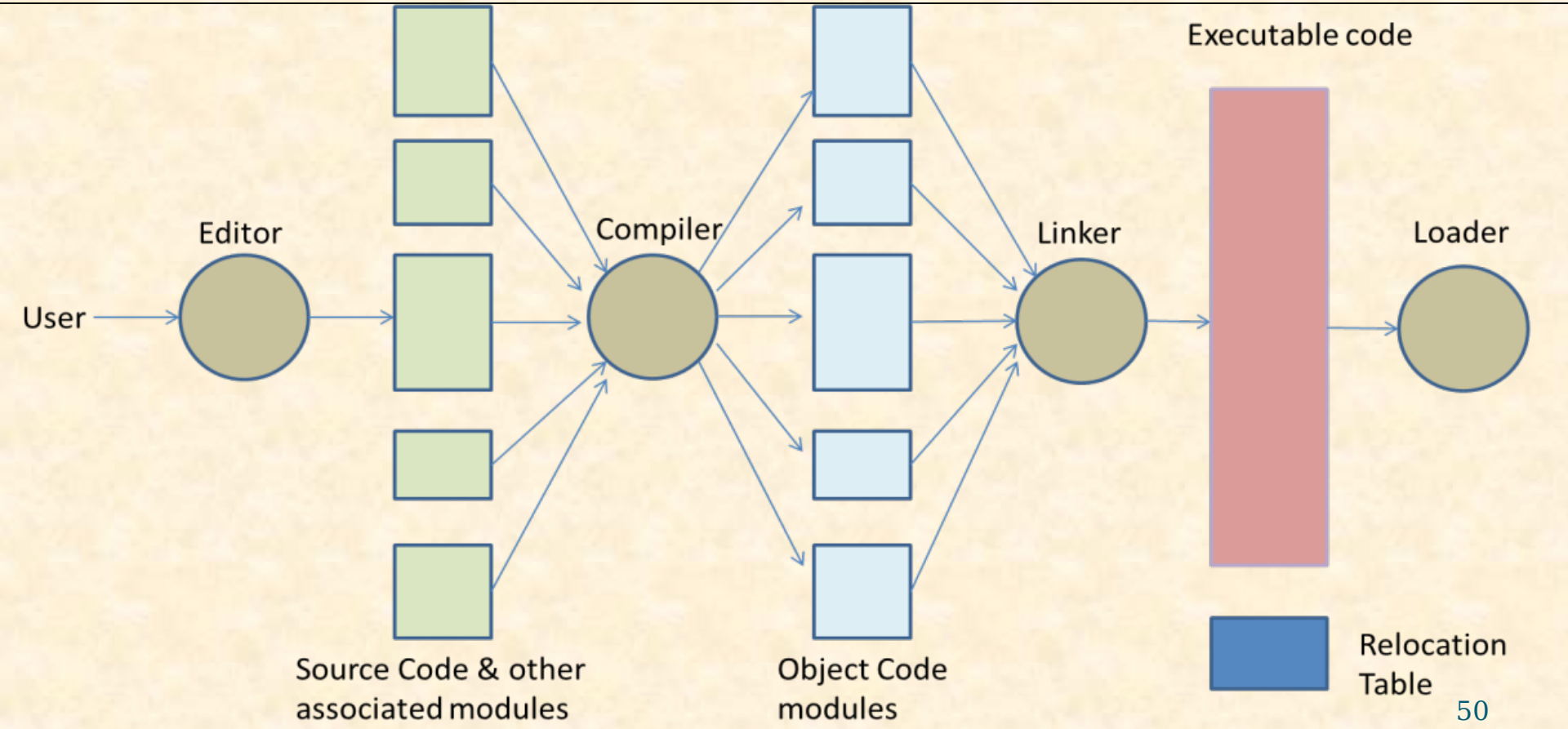
Multistep Processing of a User Program



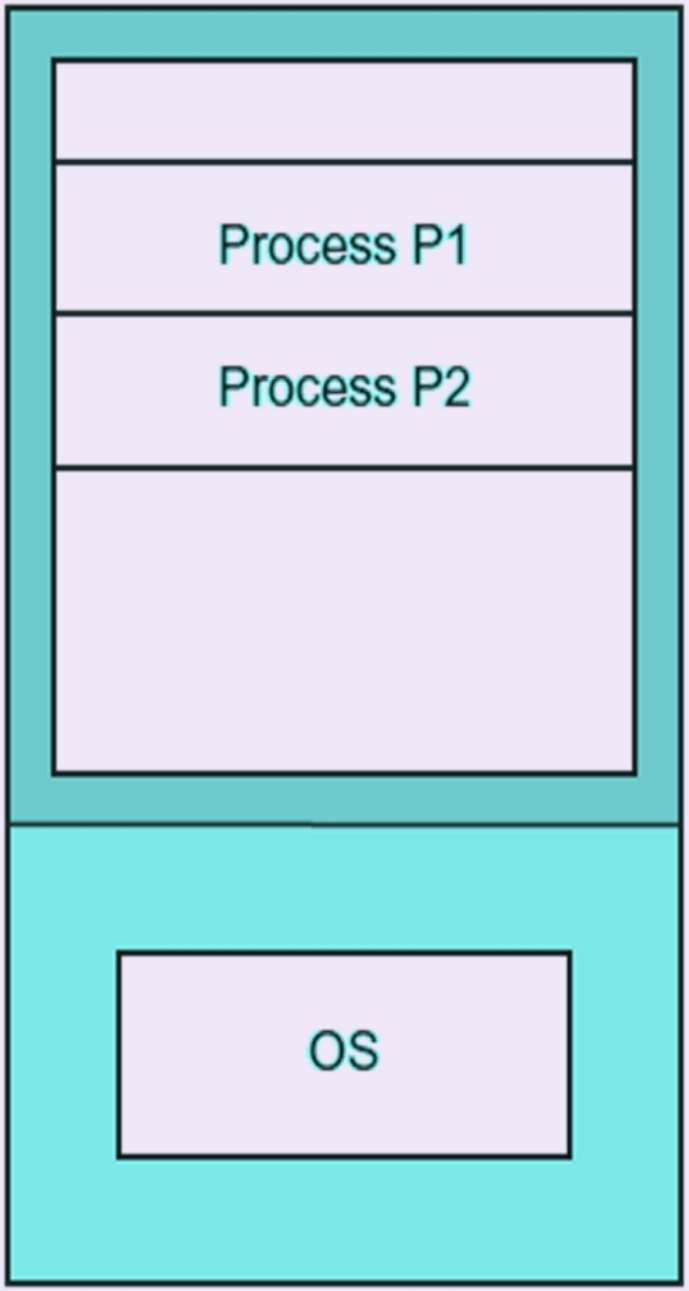
Program Relocation

- **Relocation table** that contains information about statements that are ‘address sensitive’, such as branch and function call statement, is created.
- Relocation factor (RF) = (Load time address − Compile time address)

Linking and loading of a program



Relocation with base and limit register



10200

10400

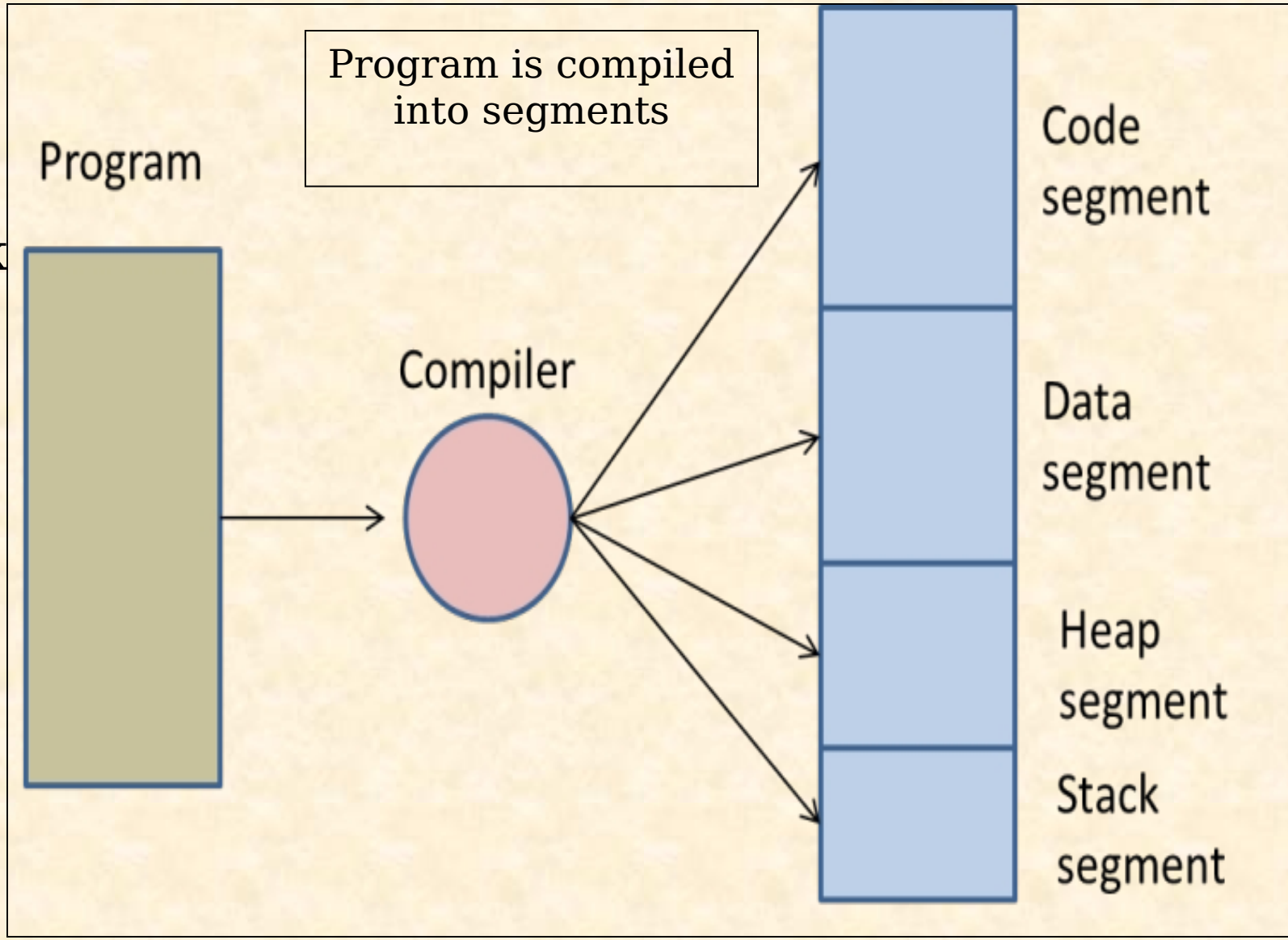
10600

Base register 10200

Limit register 10400

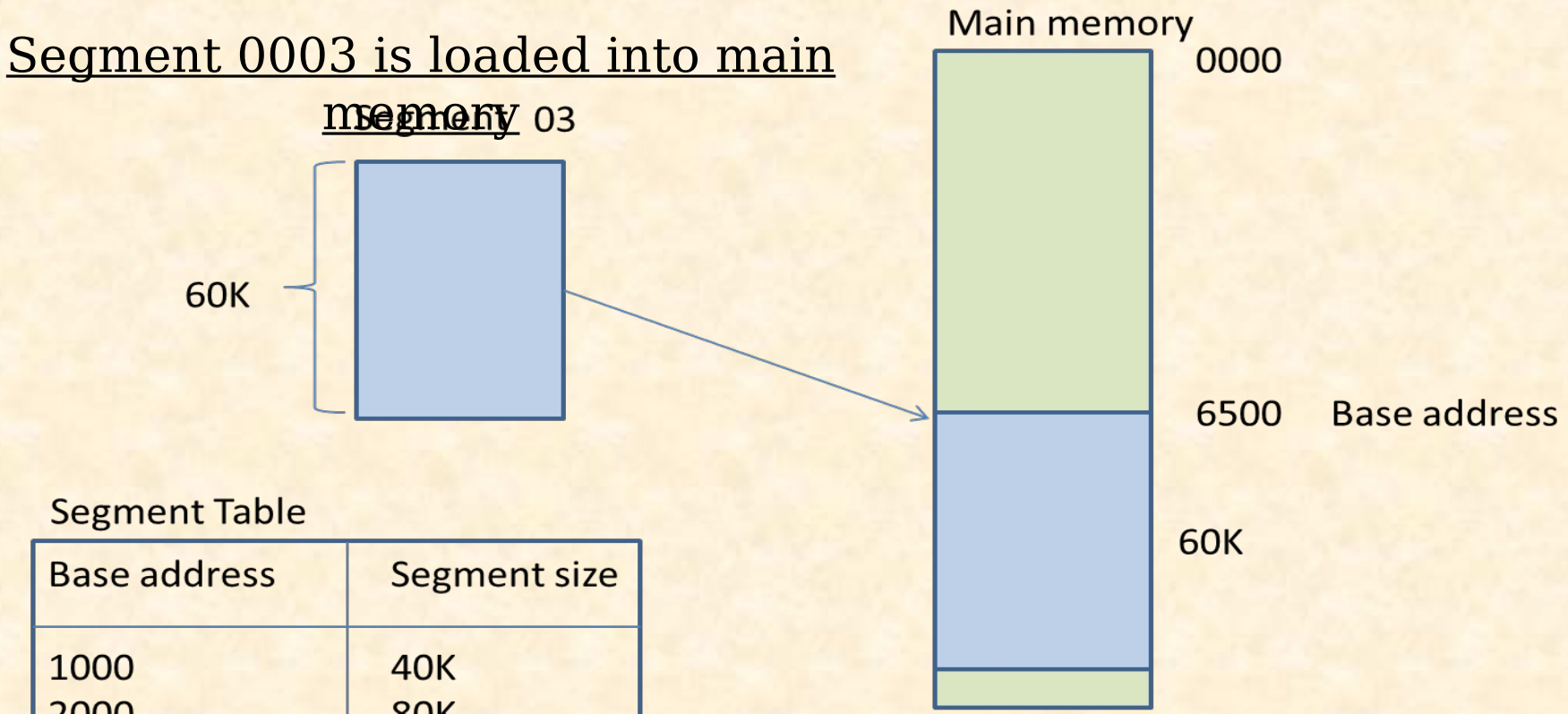
Segmentation

- A program consists of the following components:
  - Code
  - Data
  - Heap
  - Stack



Segmentation

- When a segment is loaded into main memory, its starting main memory address is called the base address.
- The size of the segment is stored in a table called the segment table.

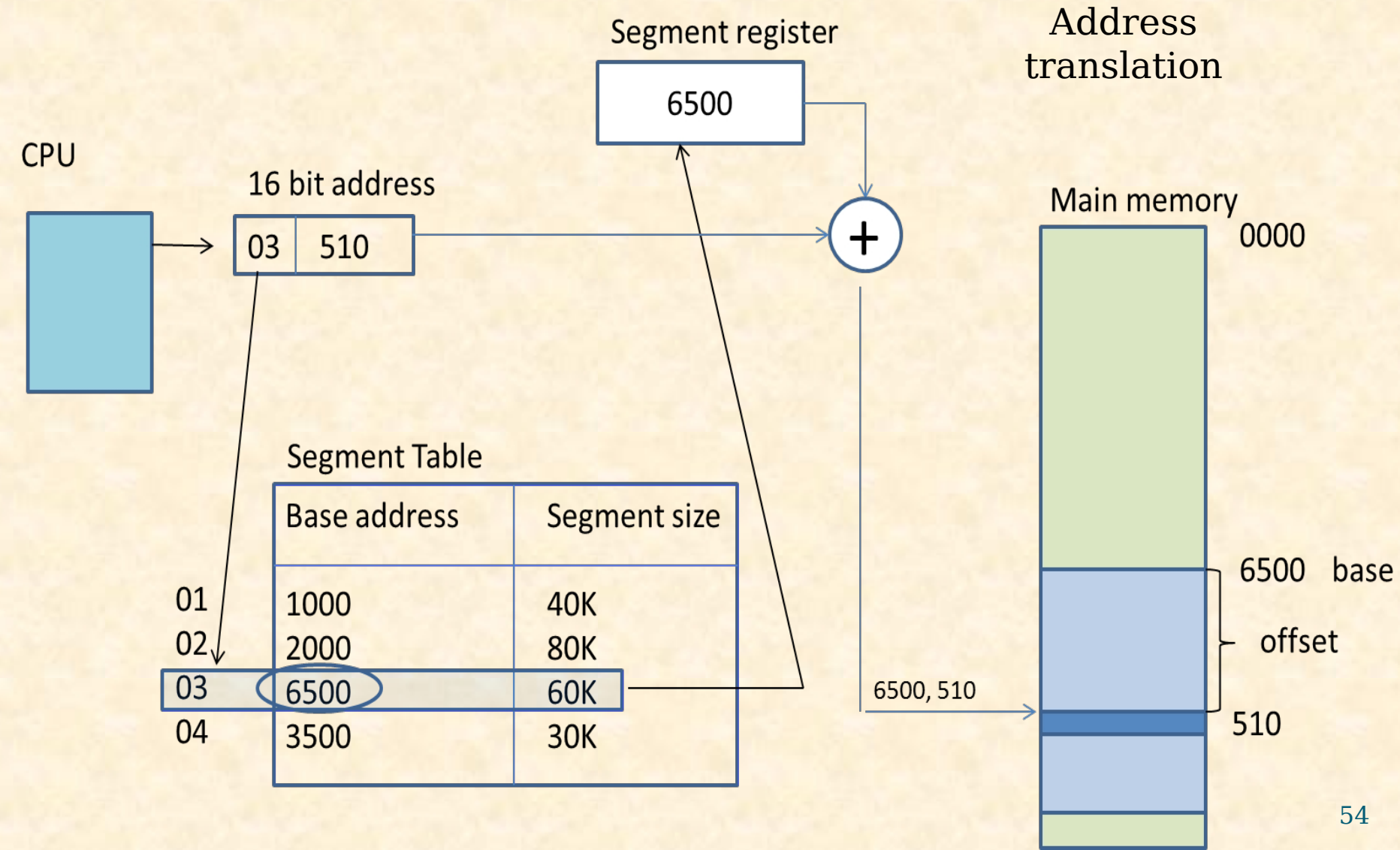


Segment Table

	Base address	Segment size
01	1000	40K
02	2000	80K
03	6500	60K
04	3500	30K

Segmentation: Address Translation

- Segmentation and variable sized partitioning suffer from external fragmentation.

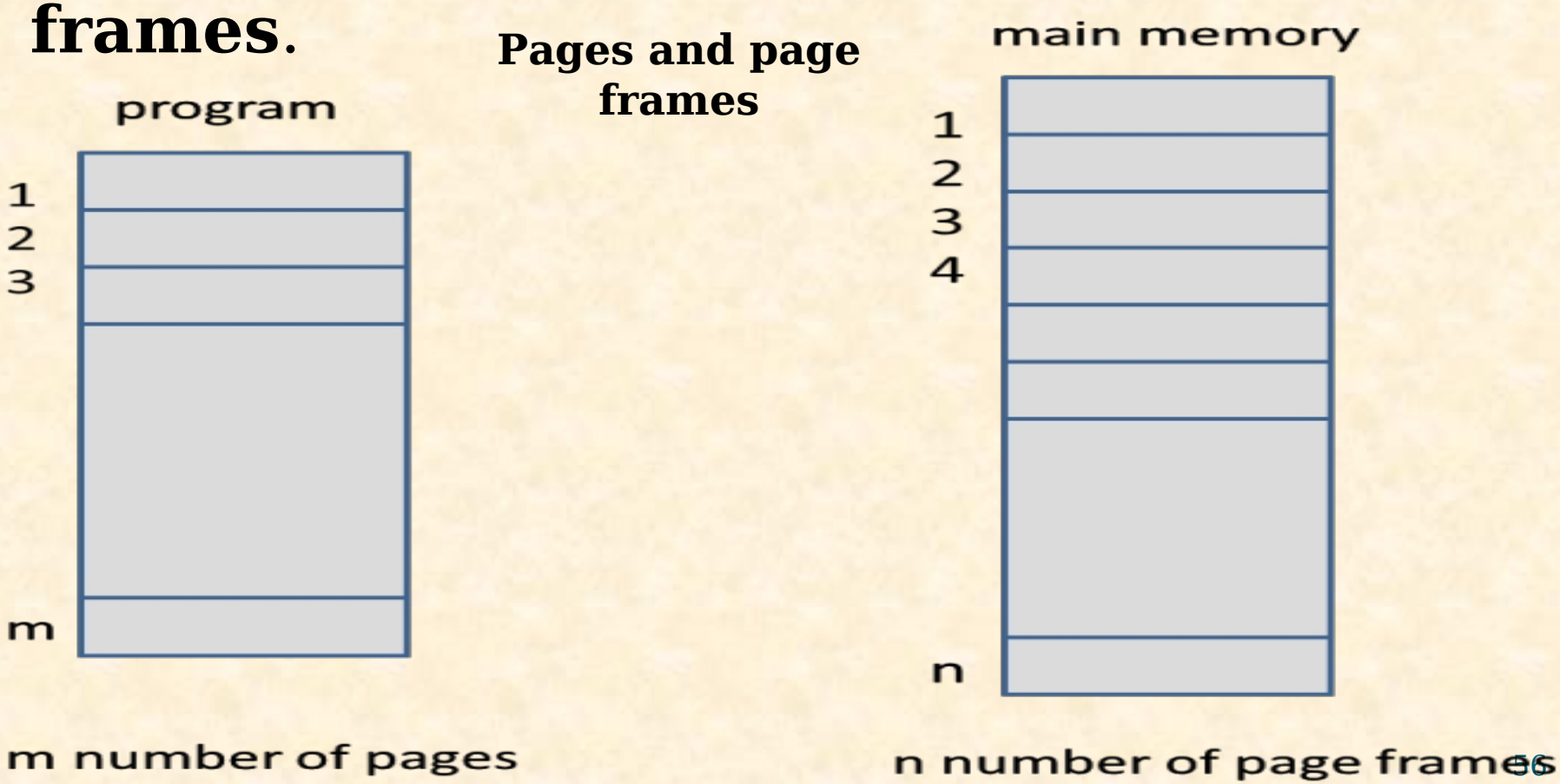



START

Paging

- The program and main memory are divided into equal sized partitions. The partitions of a program are called **pages**.
- The partitions of main memory are called **page frames**.

Pages and page frames



Paging concept

- Memory is equal-sized partitioned
- Partitions are known as *frames*
- The logical memory of a process is also divided into small fixed-size chunks or blocks of the same size as of frames. These chunks of processes are called *pages* of the process.
- The hard disk is also divided into the *blocks* which are of same size as frames.
- Thus, *paging is a logical concept that divides the logical address space of a process into fixed sized partitions known as pages and is implemented in physical memory through the frames*.
- Instead of a base register for every page, the start addresses of pages are stored in the form of a table known as *page table*.

Logical and physical addresses

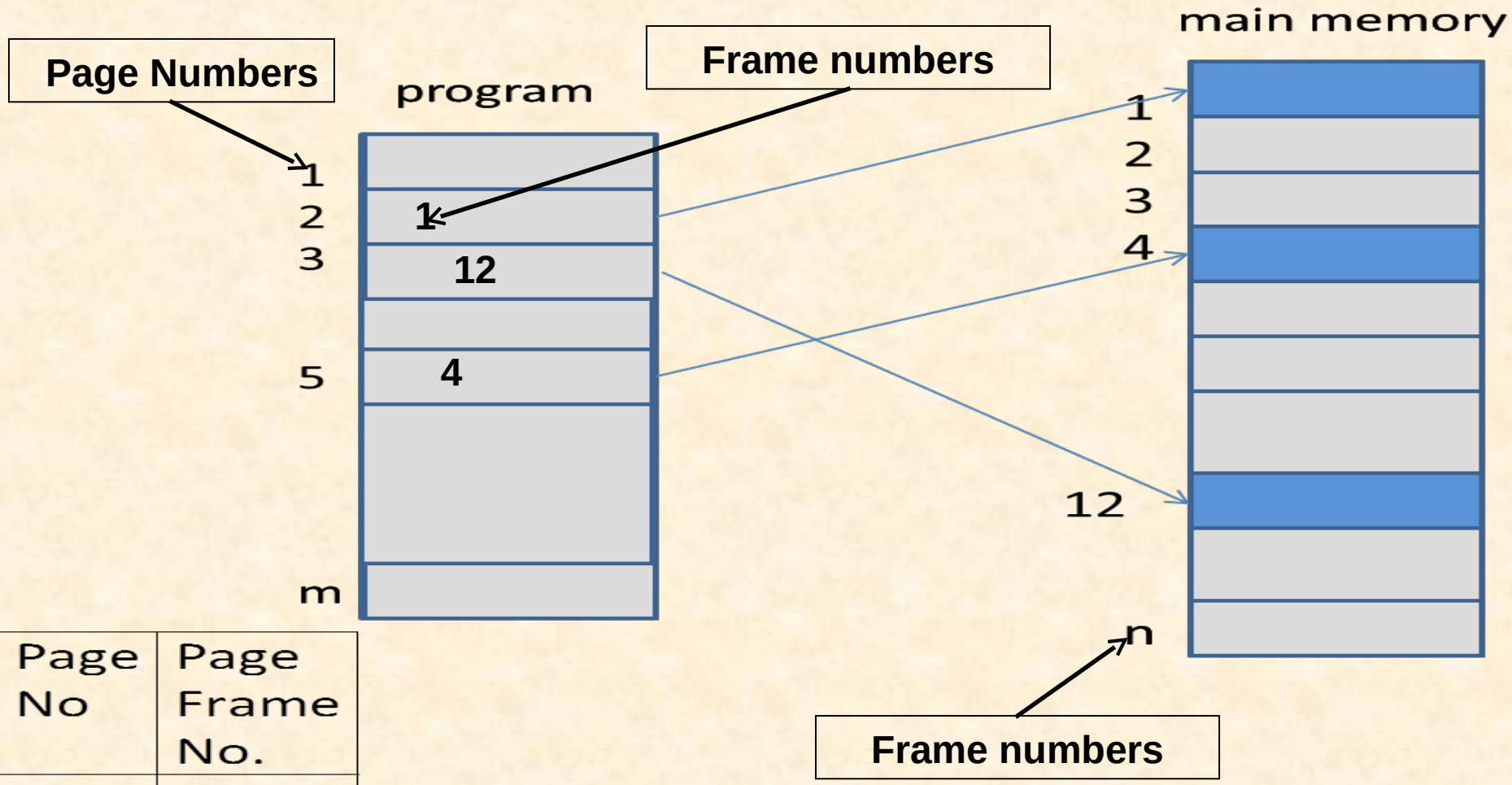
- The processor at compile time generates some addresses known as ***logical addresses***. The set of all logical addresses generated by the compilation of the process is known as ***logical address space***.

- Logical addresses need to be converted into absolute addresses at the time of execution of the process. These absolute addresses are known as ***physical addresses***. The set of physical addresses generated corresponding to all logical addresses during process execution is known as ***physical address space***

Paging

- Pages $= \frac{\text{Size of program}}{\text{size of page}} = \frac{\text{Size of logical memory}}{\text{size of page}}$
- No. of page frames $= \frac{\text{Size of main memory}}{\text{size of page}} = \frac{\text{Size of physical memory}}{\text{size of page}}$

Direct Mapping



Page No	Page Frame No.
3	12
5	4
2	1

The page table keeps the entries for the pages loaded into the main memory

Paging concept

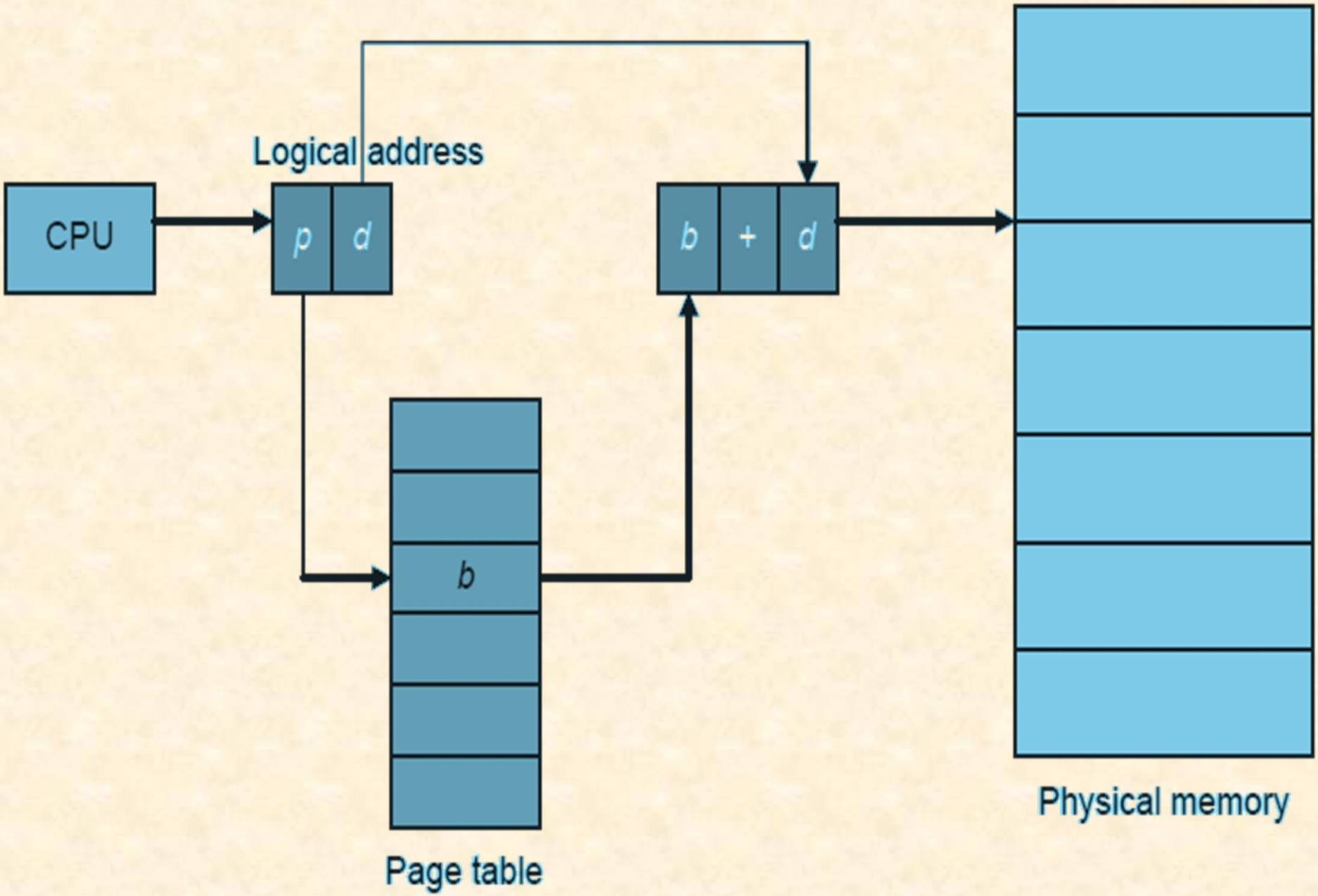
Paging generates the logical address in the form:

(Page number p, Offset d)

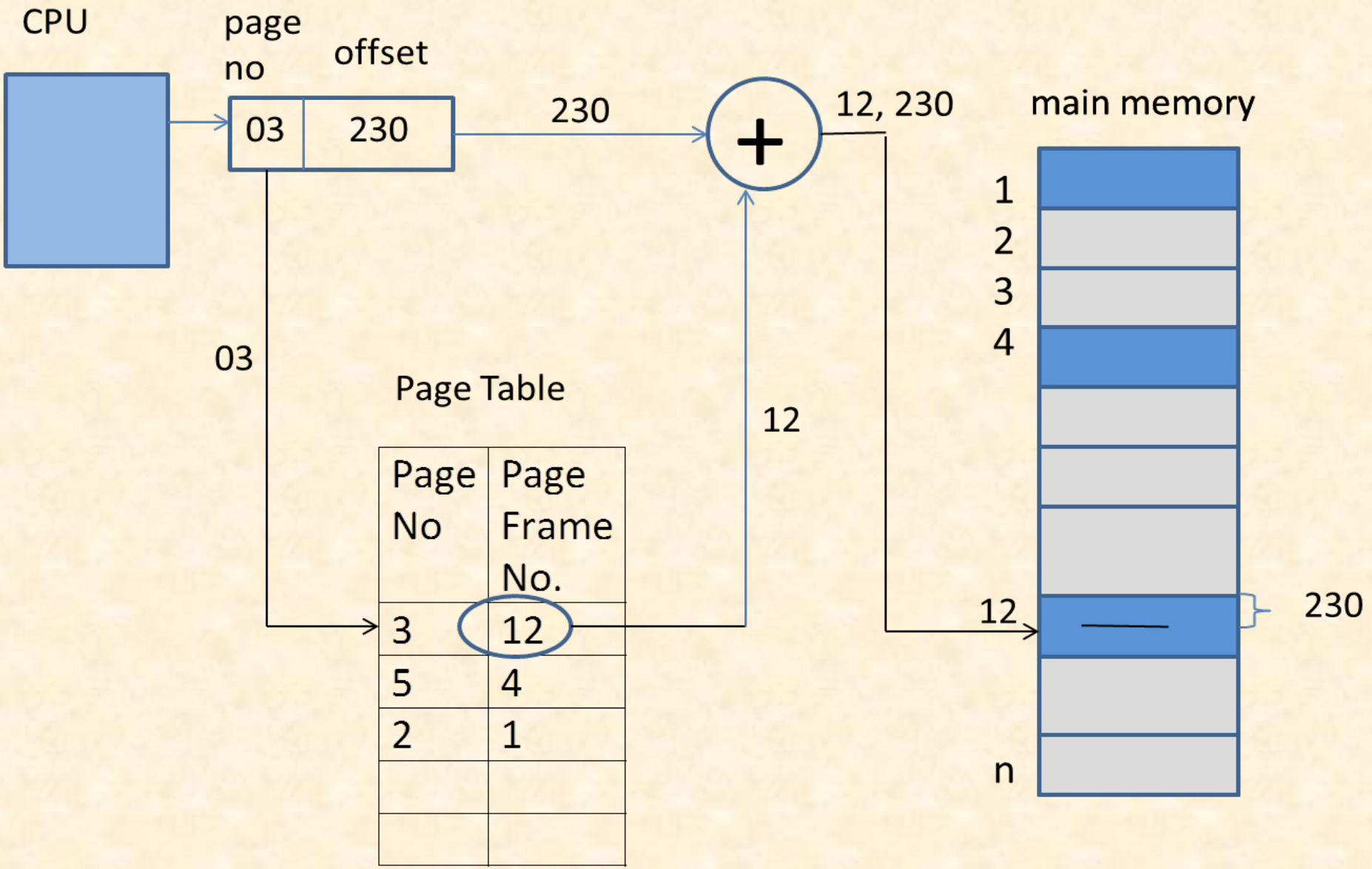
The logical address is converted into physical address by the memory management unit. page table is stored in memory only in contemporary operating systems. The hardware support needed in this case is to have a register just like the base register so that the page table address can be stored. This base register is now known as *page table base register* (PTBR).

some of the page table entries are cached in a high speed cache memory. A high speed associative cache memory known as *translation look aside buffer* (TLB) is used for this purpose.

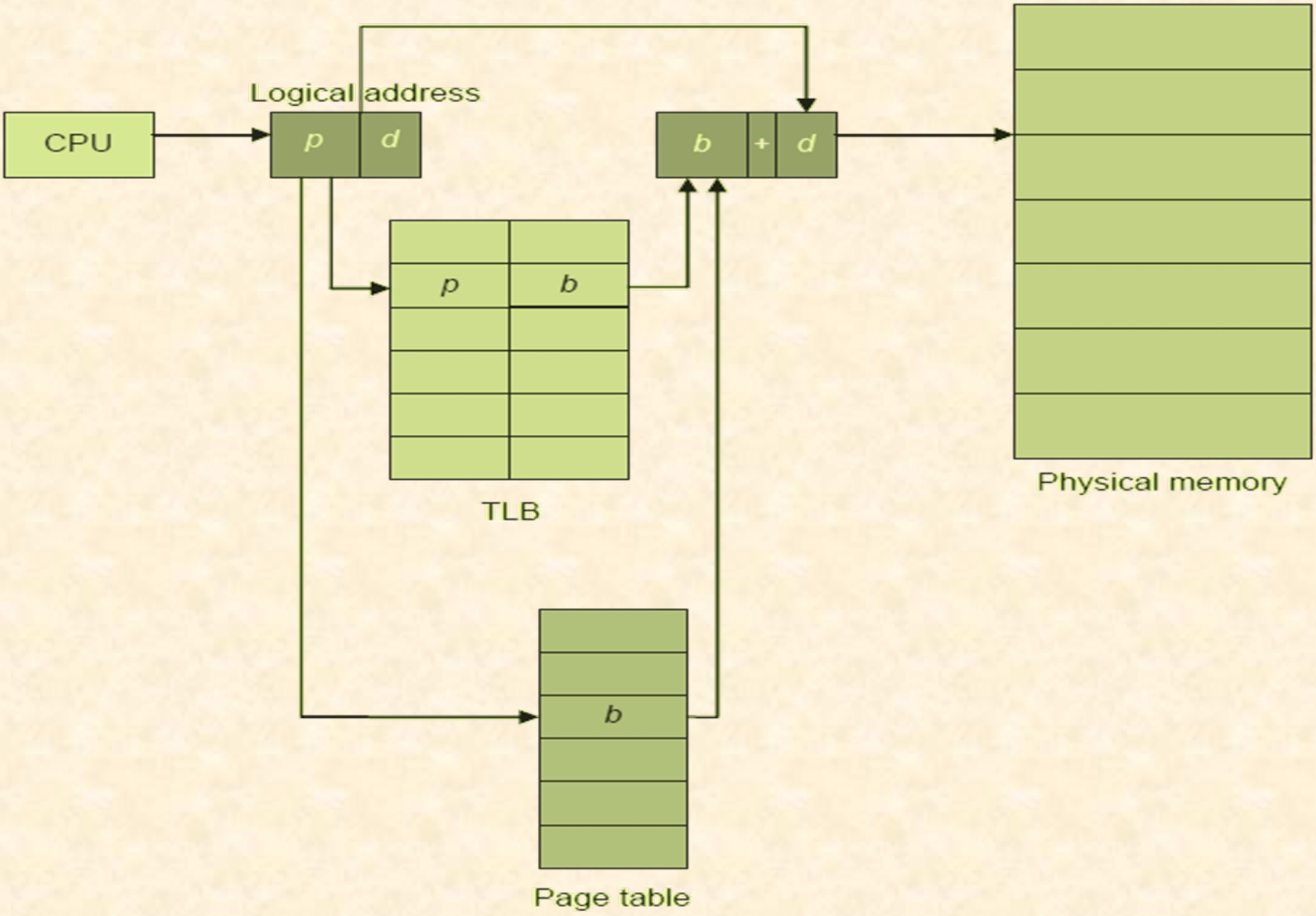
Address translation in paging - 1



Address Translation in Paging - 2



Paging implementation with TLB



Hit Ratio and Miss Ratio

- Hit ratio: If a page table entry is found, then it is called a **hit**. In case of a **miss,** the page is bought from the hard disk.

$$\text{Hit ratio} = \frac{No\ of\ hits}{Total\ page\ references}$$

- Miss ratio: Ratio of number of misses to total number of references.

$$\text{Miss ratio} = \frac{No\ of\ misses}{Total\ page\ references} = \textbf{1 – Hit ratio}$$

Q. If a process has 32 k bytes logical address space and the page size is 2048 bytes then the number of frames of that process is

a) 4 b) 8 c) 16 d) 32

Logical address = 32 k = $2^5 \times 2^{10} = 2^{15}$, i.e., logical address space can fit into 15 bits;

Given Page size = 2048 bytes = 2^{11};

How many pages can fit into main memory?

Number of page frames = $2^{15} / 2^{11} = 2^4 = 16$ frames;

Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 page frames.

Answer the following:

- How many bits are there in the logical address?
- How many bits are there in the physical address?

Ans: (i) Given: No of pages = 8.

Size of each page = 1024 words;

Size of logical memory = $8 * 1024 = 2^3 * 2^{10} = 2^{13}$ words

Thus the number of bits in the logical address = 13 bits

(ii) Given: No of page frames = 32.

Size of each page frame = size of page = 1024 words.

Size of physical memory = $32 * 1024 = 2^5 * 2^{10} = 2^{15}$ words.

Thus the number of bits in the physical address = 15 bits

What is a memory fragmentation? What are internal and external memory fragmentations? When is an external fragmentation said to occur?

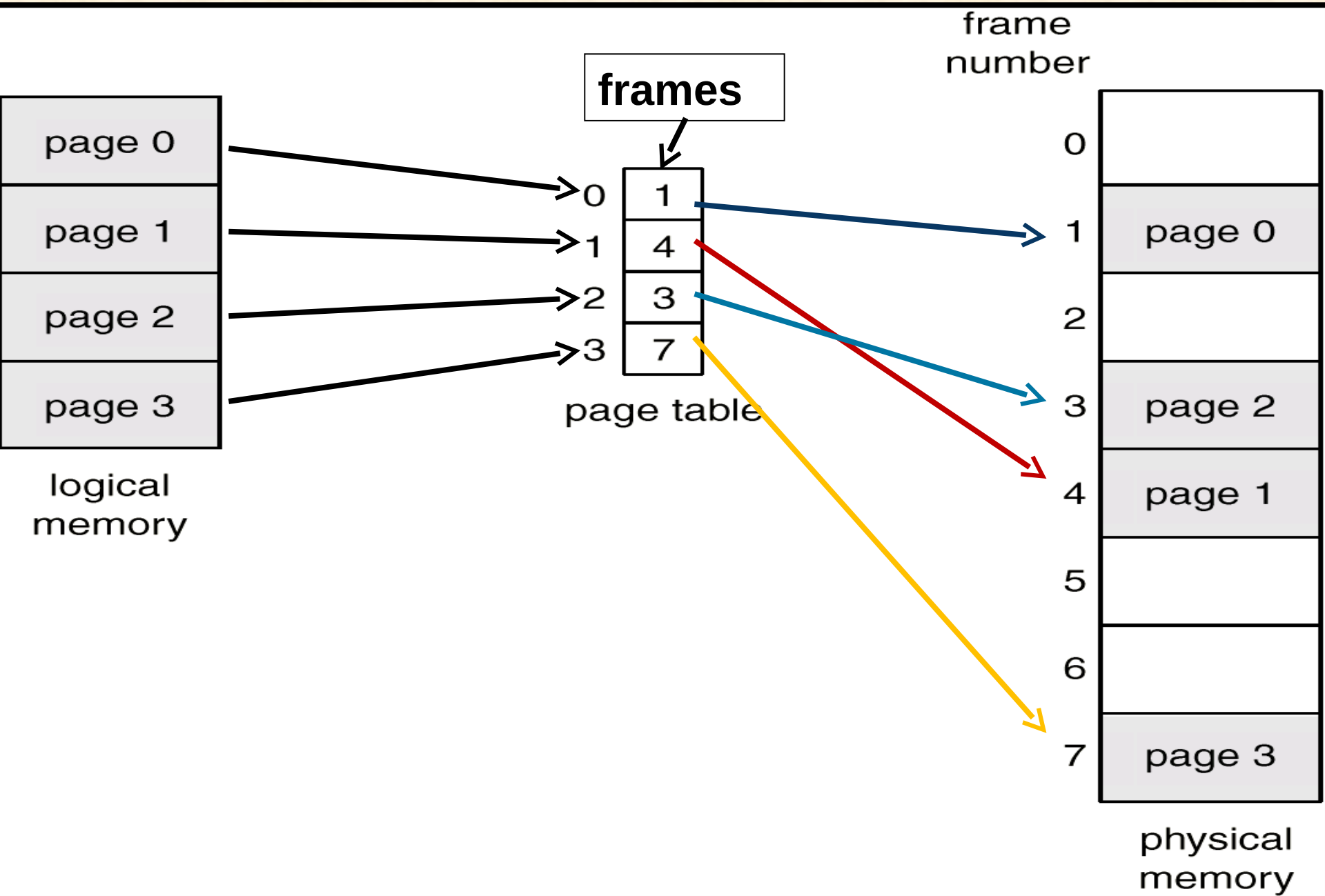
Ans: We say a memory fragmentation has occurred if the allocated memory is not needed, or the available memory cannot be used to satisfy memory allocation requests. In the former case, it is an internal fragmentation, and in the latter case an external fragmentation. If we have a memory request for n bytes and we have n or more bytes of free memory but not consecutive to satisfy the request, we say an external fragmentation has occurred.

How many pages of size 512 words each, are contained in a program with a logical address having 16 bits?

Ans: Size of logical memory = 2^{16} words.
Size of page = 512 words = 2^9 words.

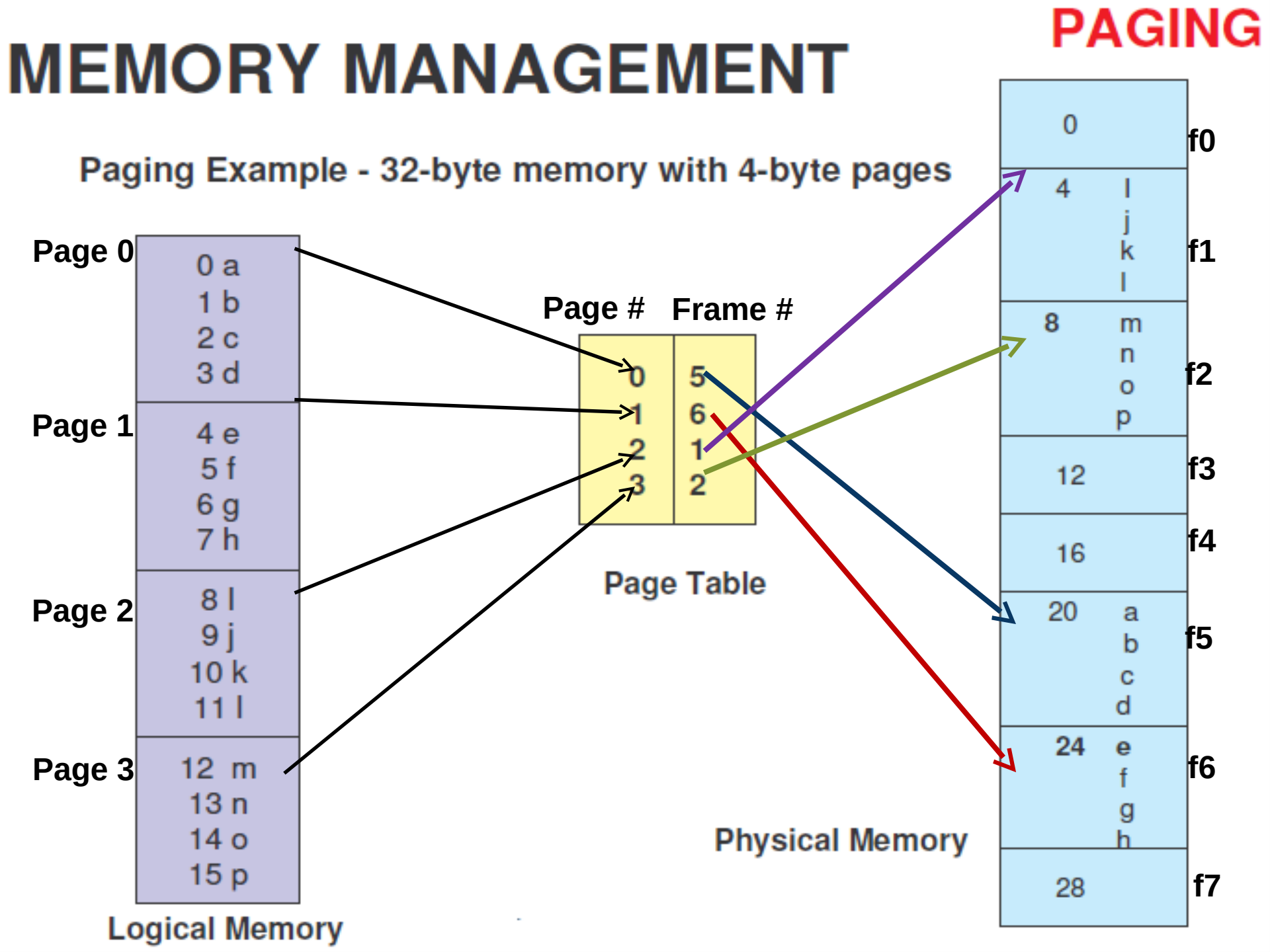
Number of pages = size of logical memory / size of page = $2^{16} / 2^9 = 2^{16-9} = 2^7$

Paging Example



MEMORY MANAGEMENT

PAGING

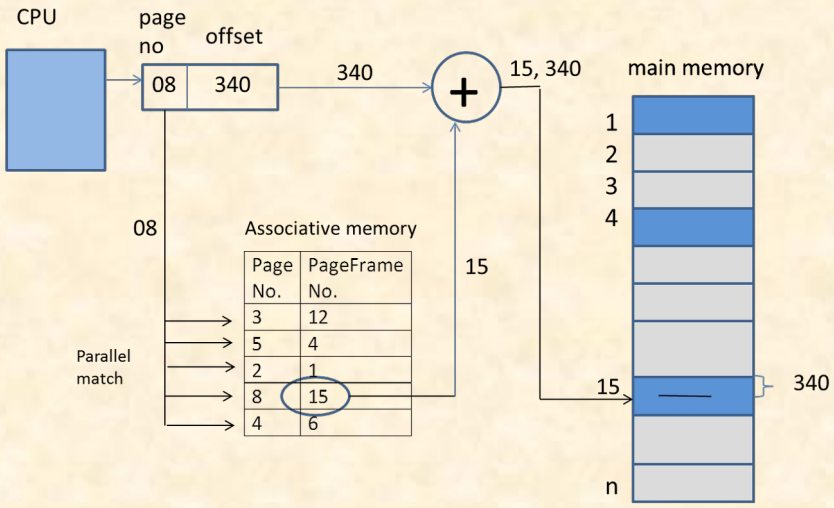


Address Translation – Associative Memory

- The memory cycle cannot be eliminated because for a virtual memory address translation, the related page table has to be accessed.

Associative Mapping

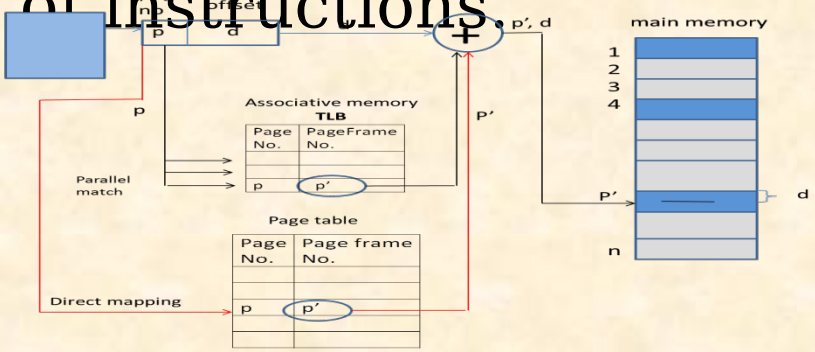
- When a complete page table is kept in associative memory, the address translation is done by pure associative mapping.

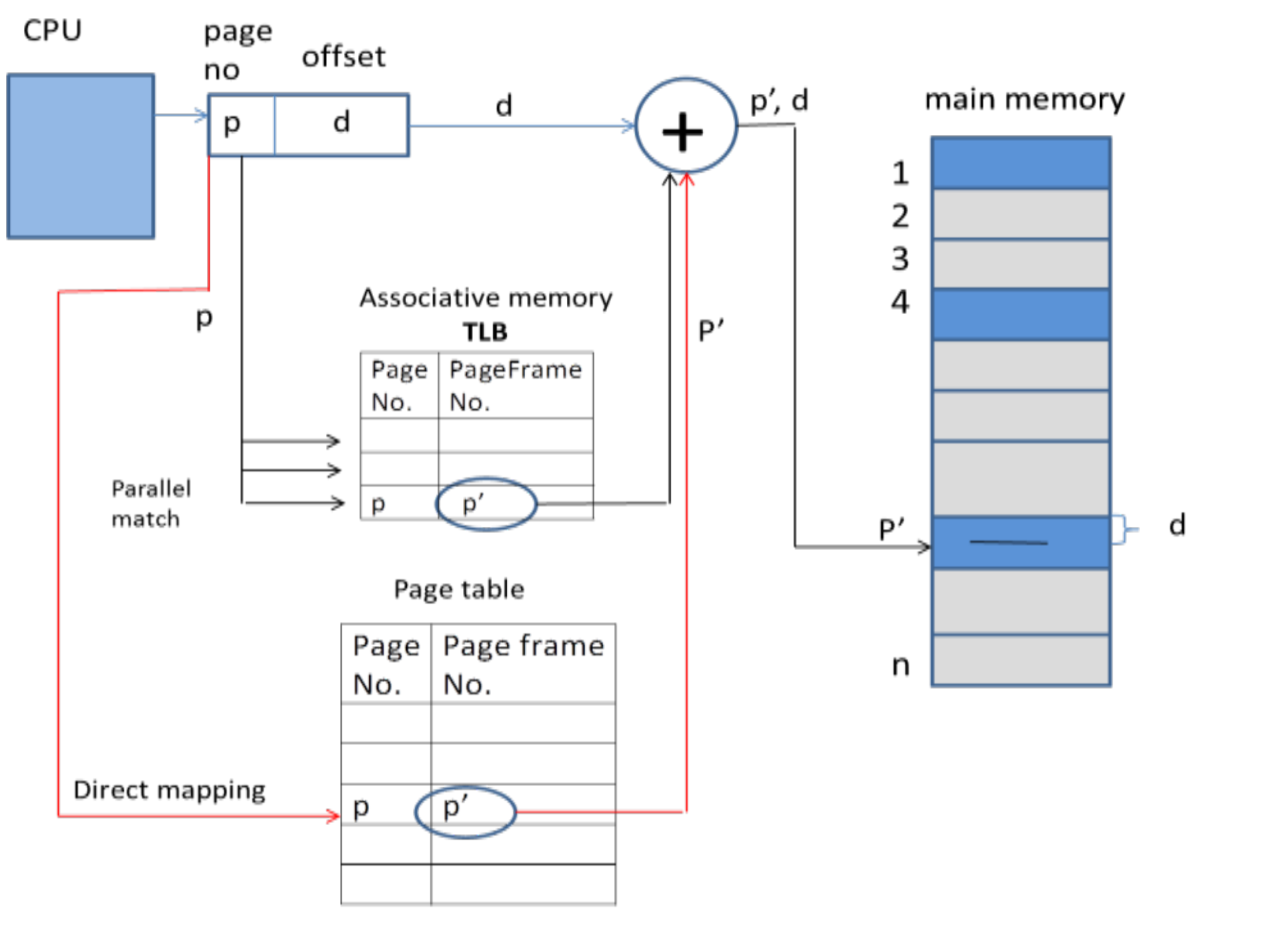


Pure associative mapping

Combined Direct and Associative Mapping

- It uses the ‘principle of locality of reference’ to accept a recently referred block of instructions that will be referred to again.
- Associative memory stores a portion of the page table.
- It keeps the page entries of the most recently referred pages of instructions.




CPU
page no
offset
p
d
d
+
p', d
main memory
1
2
3
4
n
p
Associative memory
TLB
Page No.
PageFrame No.
p
p'
P'
Parallel match
P'
d
Page table
Page No.
Page frame No.
p
p'
Direct mapping

Combined Direct and Associative Mapping

- Associative memory acts as a look-ahead buffer to keep the portion of the page table containing entries about most recently and most frequently used pages. Associative memory is also called **translation look-ahead buffer (TLB)**.

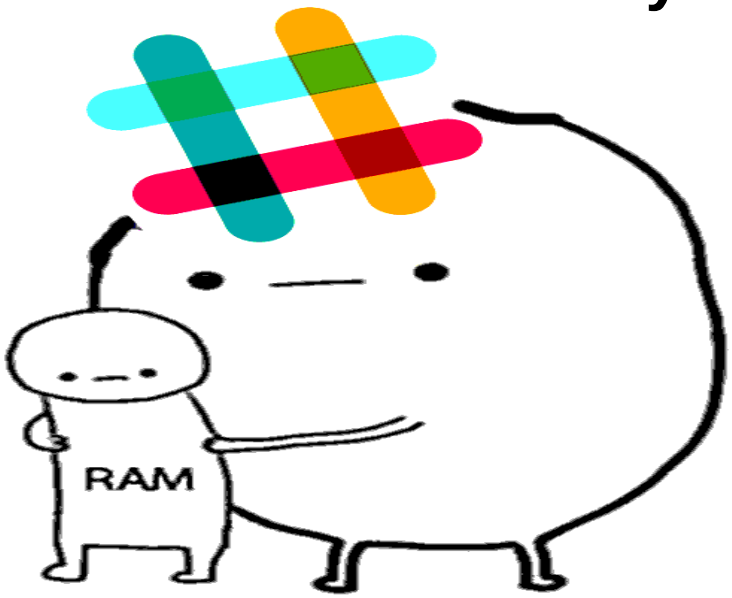
$$\text{Hit ratio (TLB)} = \frac{\text{No of hits}}{\text{Total page references}}$$

T_{eff} = Hit ratio (TLB) * (t1 + t2) + Miss ratio * (t1 + 2 *t2)

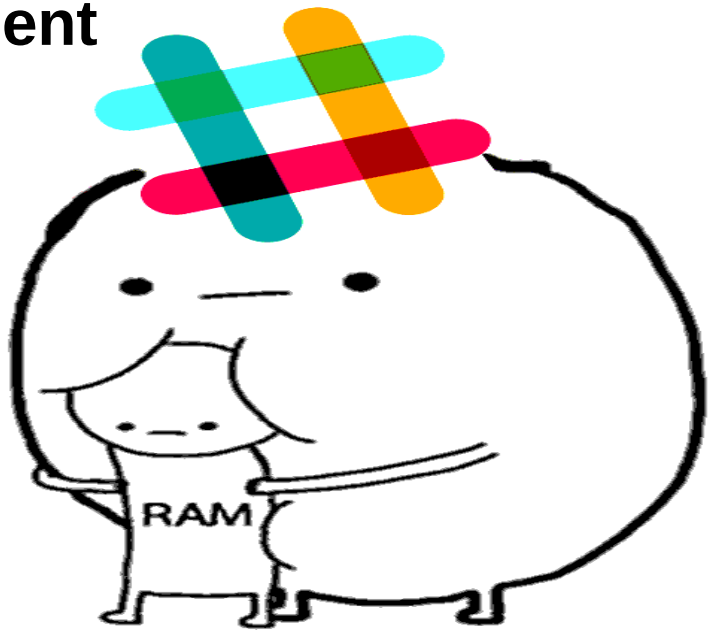
T_{eff} = Effective reference time

t1 and t2 = Access time of TLB and main memory

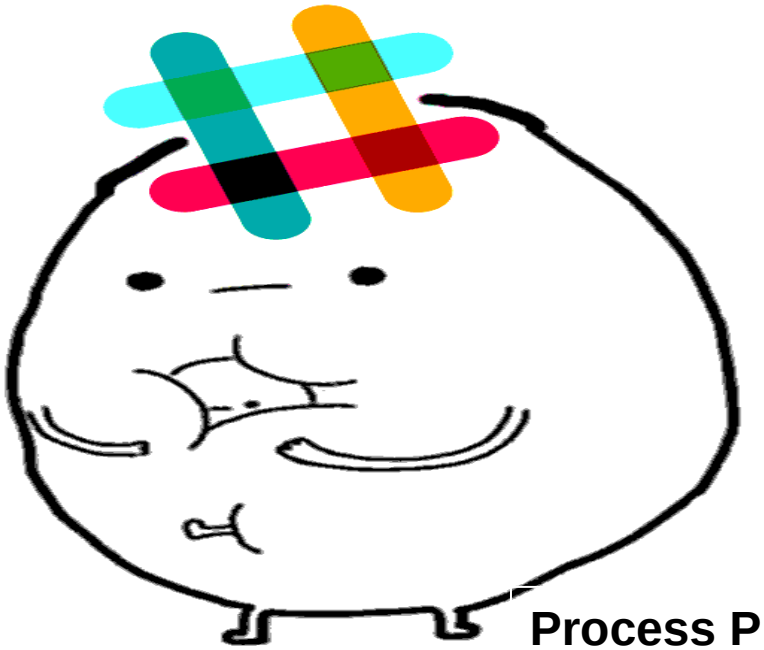
Memory Management



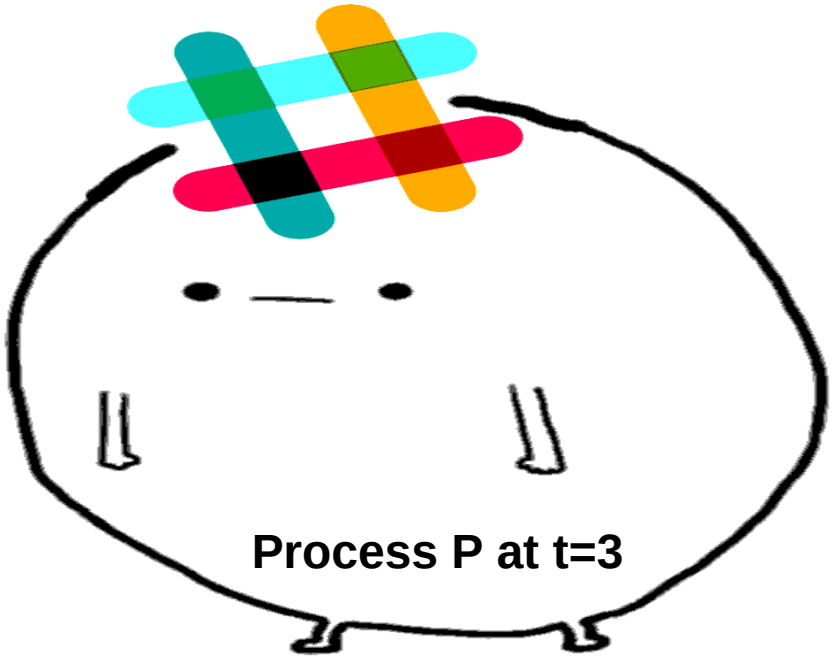
Process P at t=0



Process P at t=1



Process P at t=2



Process P at t=3