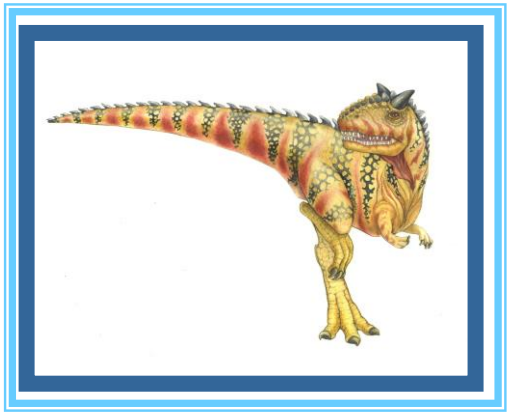
Shell programming



```
-rw-r--r-- 1 kiran kiran   14 Mar  6 11:06 test.java
-rwxr-xr-x 1 kiran kiran   17 Mar  6 12:52 test.sh
kiran@CMKL-kiranw:~/OS$ chmod +x T1.sh
kiran@CMKL-kiranw:~/OS$ ./T1.sh
Welcome to Shell Programming !
What is your name?
Kiran
Hello, Kiran
Hello, Name
kiran@CMKL-kiranw:~/OS$ cat T1.sh
#!/bin/bash

echo "Welcome to Shell Programming !"

echo "What is your name?"
read Name
echo "Hello, $Name"
echo "Hello, Name"
kiran@CMKL-kiranw:~/OS$
```

Conditionals in bash:

---

if -else:

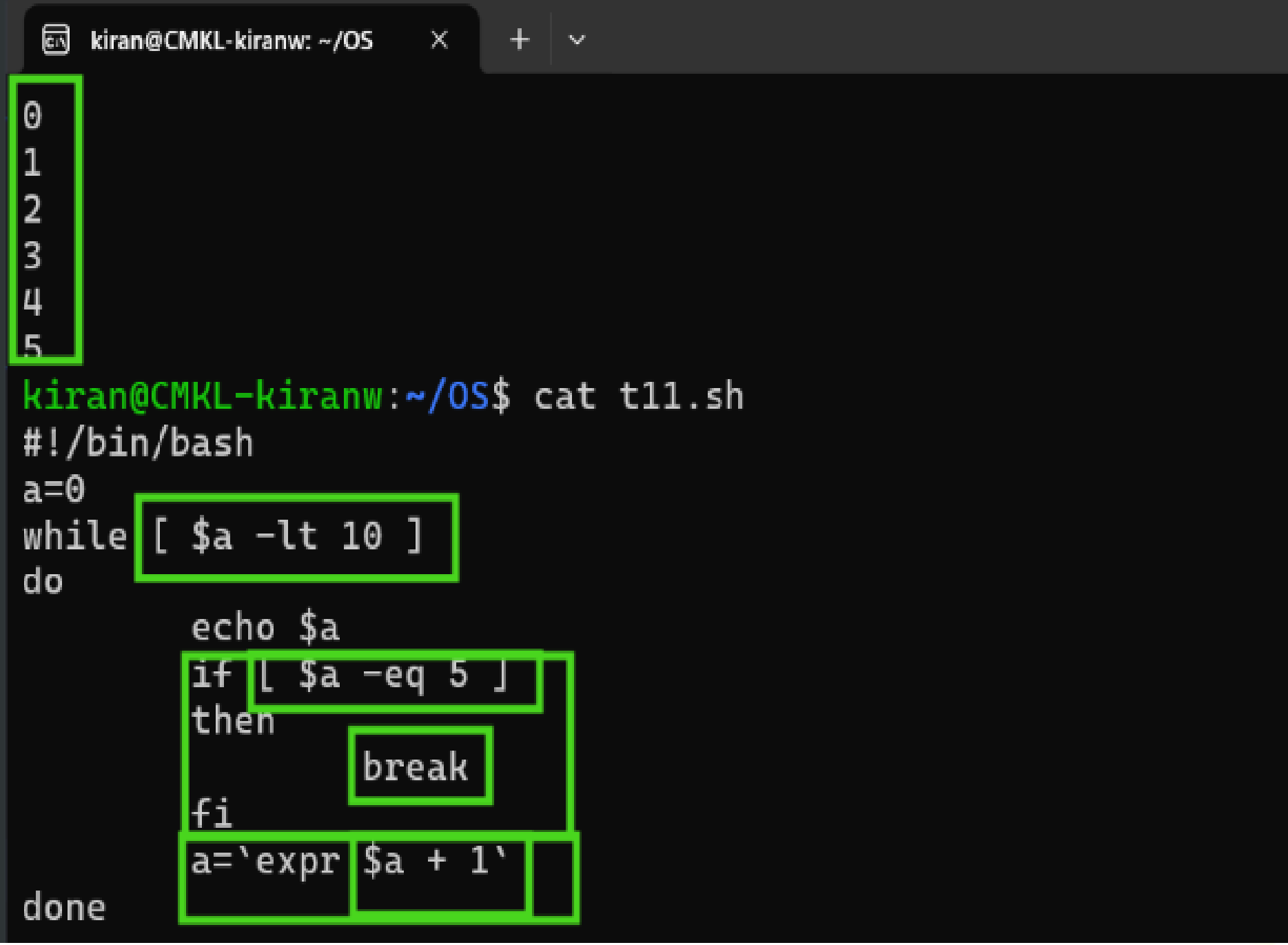
```
if [[ $x -eq 5 ]]
then
    #statement
fi
```

```
#!/bin/bash

echo " Enter number"
read num

if [[ ( $num -lt 10 ) && ( $num%2 -eq 0 ) ]];
then
        echo " Even number"
else
        echo " Odd number"
fi
~
~
~
~
~
~
~
~
~
-- INSERT --
```

```
read x
for i in \`seq 1 $x\`
```

kiran@CMKL-kiranw: ~/OS

```
#!/bin/bash

for var1 in 1 2 3
do
        for var2 in 0 5
        do
                if [[ $var1 -eq 2 -a $var2 -eq 0 ]]
                then
                        break 2
                else
                        echo "$var1 $var2"
                fi
        done
done
~
~
~
~
~
-- INSERT --                                    14,5
```



```
#!/bin/bash
Hello (){
        echo "Hello, Please take break !!!"
}

Hello
kiran@CMKL-kiranw:~/OS$
```

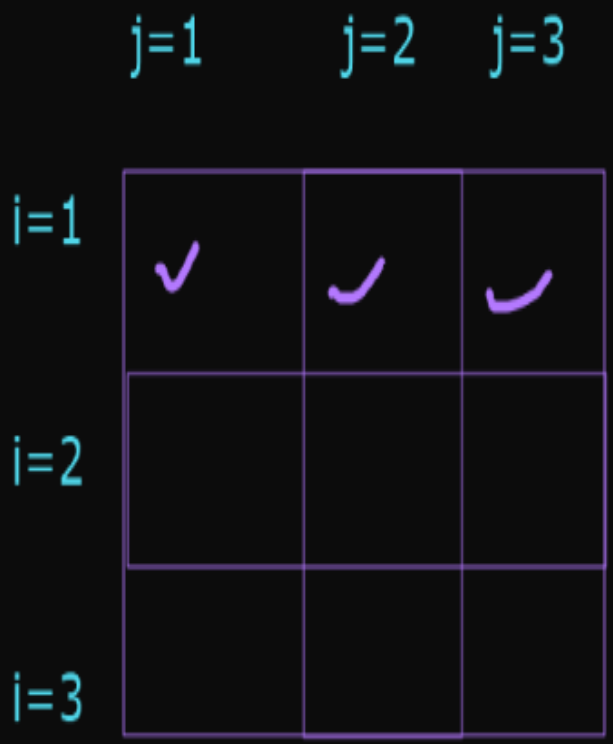
Function definition

Function calling-> function gets invoked

```
#!/bin/bash
echo "Enter number:"

read num

case $num in
        100) echo " Let's have something !";;
        200) echo " Let's have Lunch !";;
        *) echo "No-No we will complete shell programming first!";;
esac

kiran@CMKL-kiranw:~/OS$
```

kiran@CMKL-kiranw: ~/OS

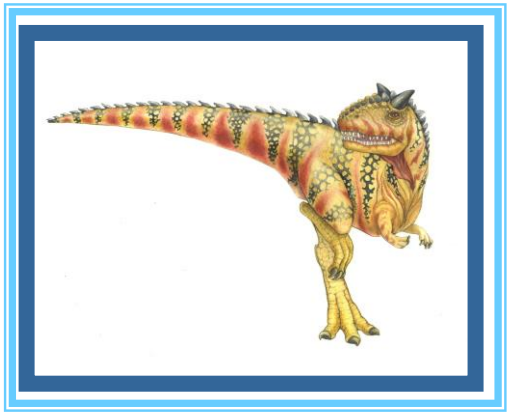
```
1
1
1
2
2
2
3
3
3
```




```
kiran@CMKL-kiranw:~/OS$ cat t15.sh
#!/bin/bash

for var1 in 1 2 3
do
        for var2 in 1 2 3
        do
                echo "$var1"
        done
done
kiran@CMKL-kiranw:~/OS$
```

Memory Management
Day5: Sep 2021

Kiran Waghmare



- Memory Management
  - continuous & dynamic
  - best fit, worst fit, first fitexternal internal fragmentation
  - segmentation
  - paging
  - concept of dirty bit
  - shared pages & reentrant
  - Throttling

Memory Management:

Memory : collection of some amount of data.

-It is represented in the form of binary bits (format)

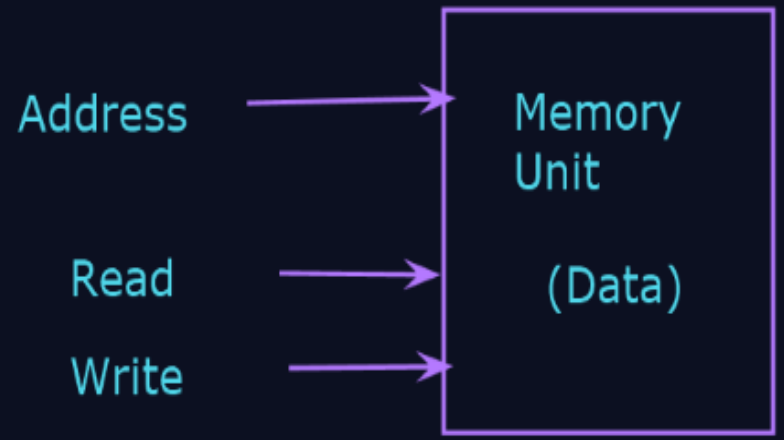
-Bit = 0/1

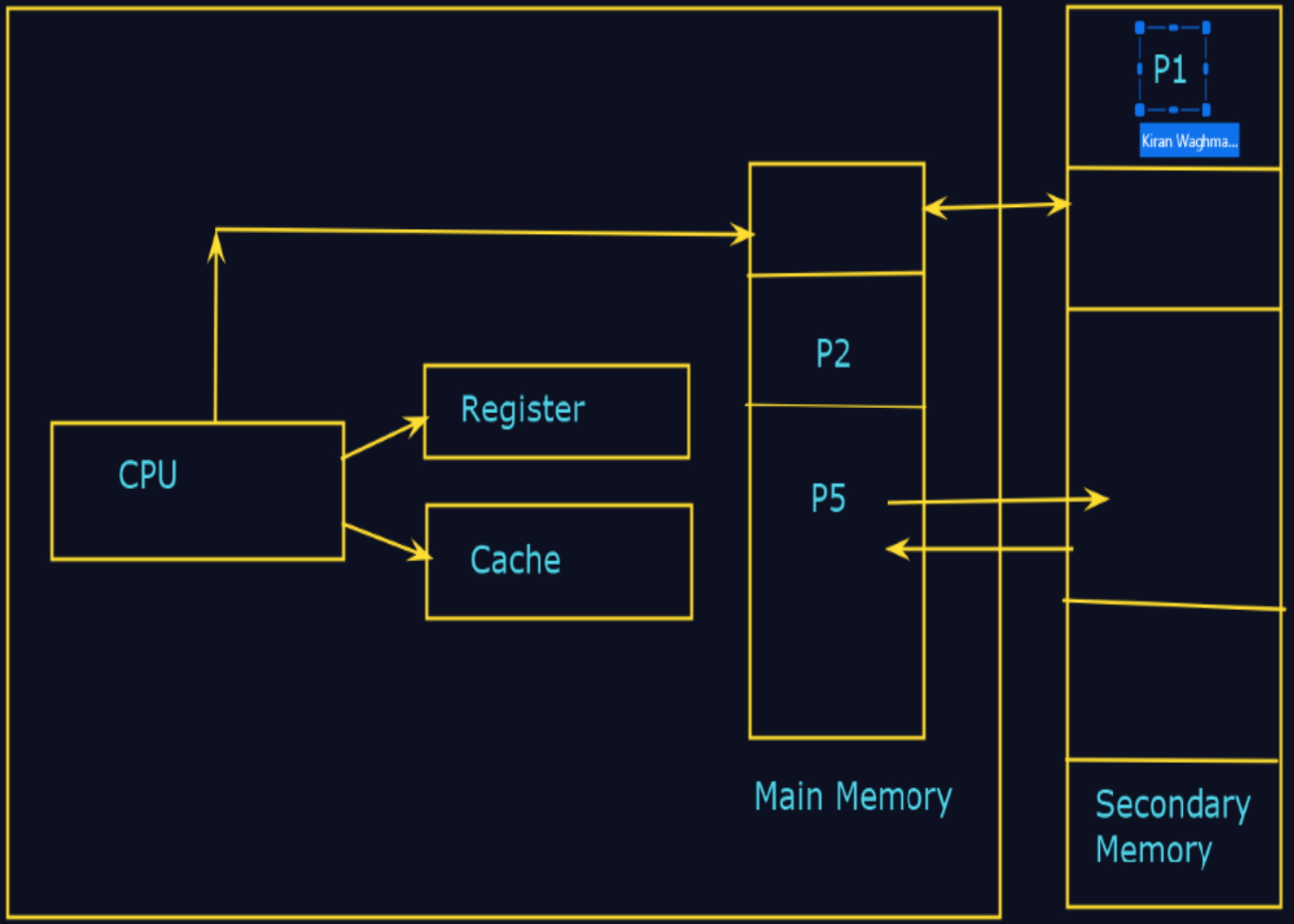
-Nibble = 4bits

-Byte = 8bits

-Word = 16bits

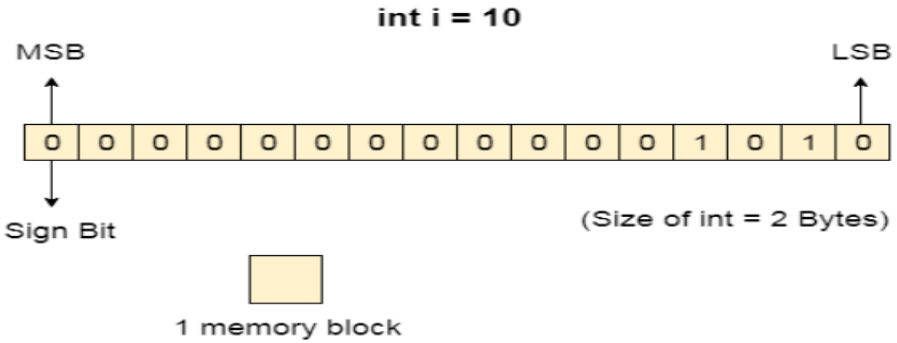
1 0 0 1 1 1 1 1 1 0 1 0 1 0





What is Memory?

- Computer memory can be defined as a **collection of some data represented in the binary format.**
- On the basis of various functions, memory can be classified into various categories.
- Machine **understands only binary language that is 0 or 1**.
- Computer converts every data into binary language first and then stores it into the memory.

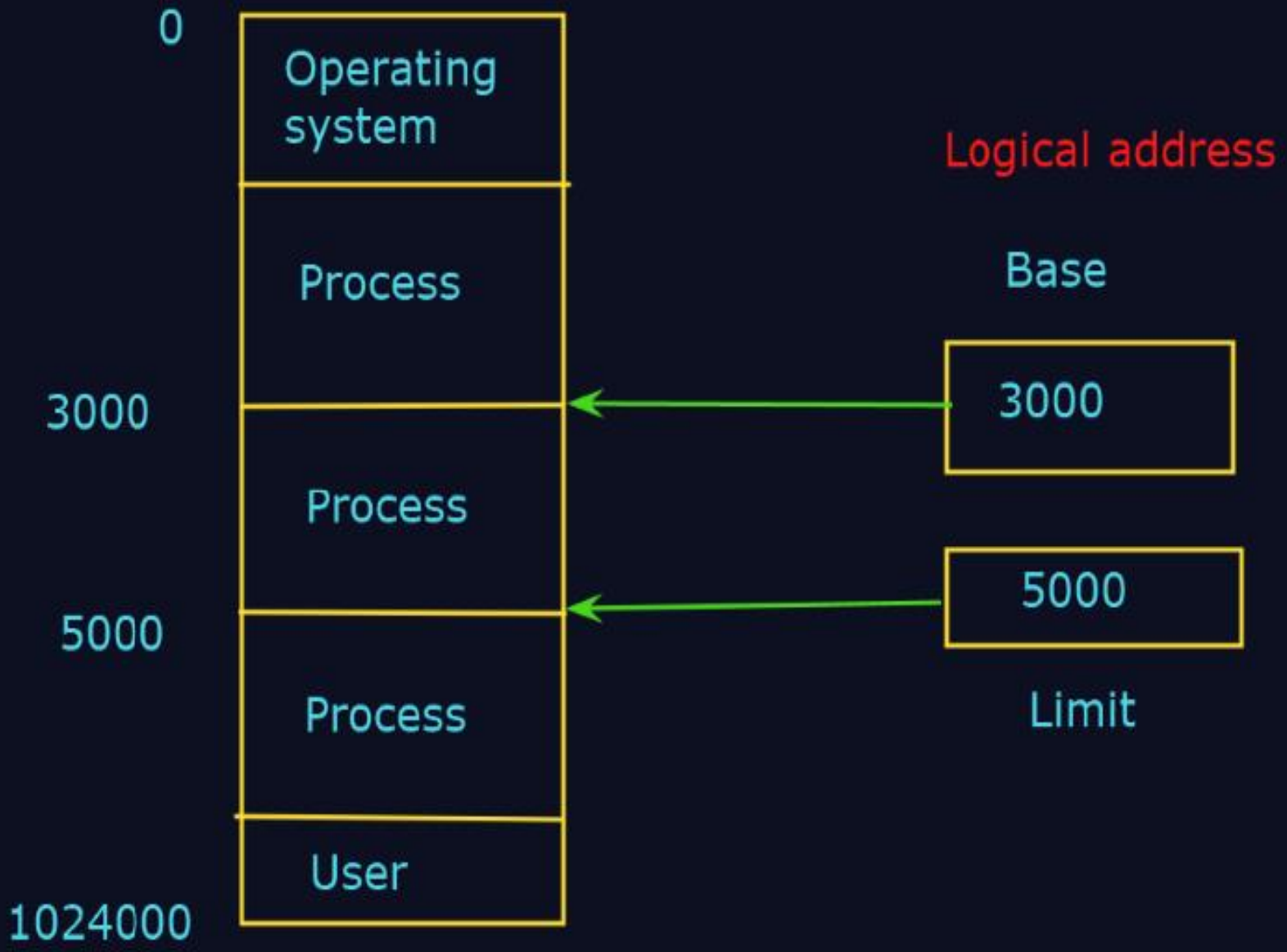


What is Main Memory:

- The main memory is **central to the operation of a modern computer**.
- Main Memory is a **large array of words or bytes, ranging in size from hundreds of thousands to billions.**
- Main memory is a **repository of rapidly available information** shared by the CPU and I/O devices.
- Main memory is the place where **programs and information are kept** when the processor is effectively utilizing them.
- Main memory is **associated with the processor, so moving instructions and information i**nto and out of the processor is extremely fast.
- Main memory is also known **as RAM(Random Access Memory).**
- This memory is a **volatile memory**.
- RAM lost its data when a power interruption occurs.

Background

- Program must **be brought (from disk) into memory** and placed within a process for it to be run
- **Main memory and registers are only storage CPU** can access directly
- Register access in **one CPU clock** (or less)
- Main memory **can take many cycles**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation


0
Operating system
Process
3000
Process
5000
Process
User
1024000
Logical address
Base
3000
5000
Limit

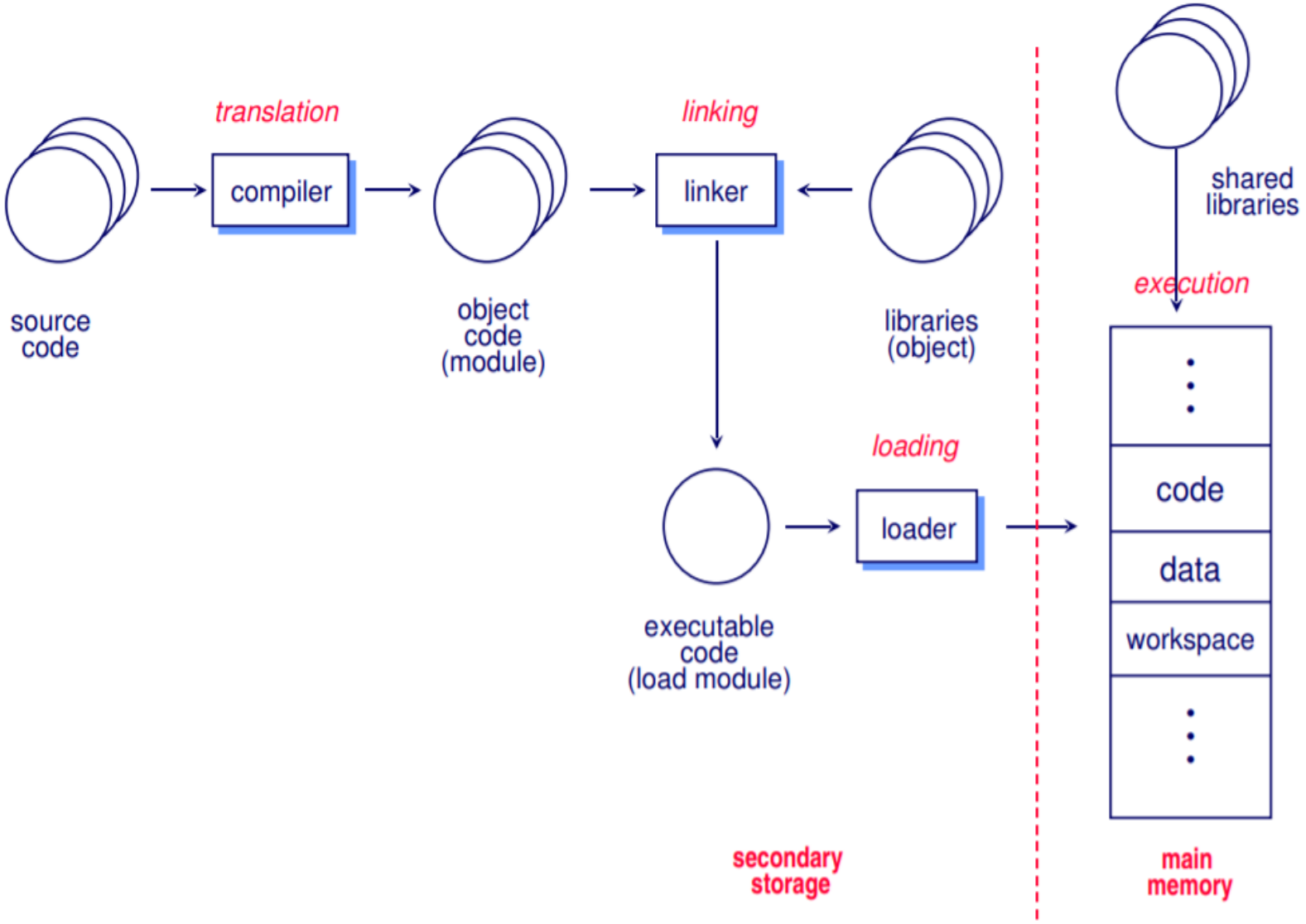
Need for Memory Management in OS

- Memory management technique is needed for the following reasons:
  - This technique **helps in placing the programs in memory** in such a way so that memory is utilized at its fullest extent.
  - This technique **helps to protect different processes from each other** so that they do not interfere with each other's operations.
  - It helps to **allocate space to different application routines**.
  - This technique allows you **to check how much memory needs** to be allocated to processes that decide which processor should get memory at what time.
  - It **keeps the track of each memory location** whether it is free or allocated.
  - This **technique keeps the track of inventory whenever memory gets freed or unallocated** and it will update the status accordingly.

Why Memory Management is required:

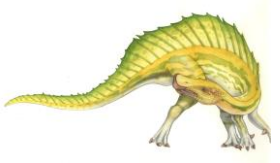
- **Allocate and de-allocate memory** before and after process execution.
- To **keep track of used memory** space by processes.
- To **minimize fragmentation** issues.
- To **proper utilization** of main memory.
- To **maintain data integrity** while executing of process.

From *source* to *executable* code

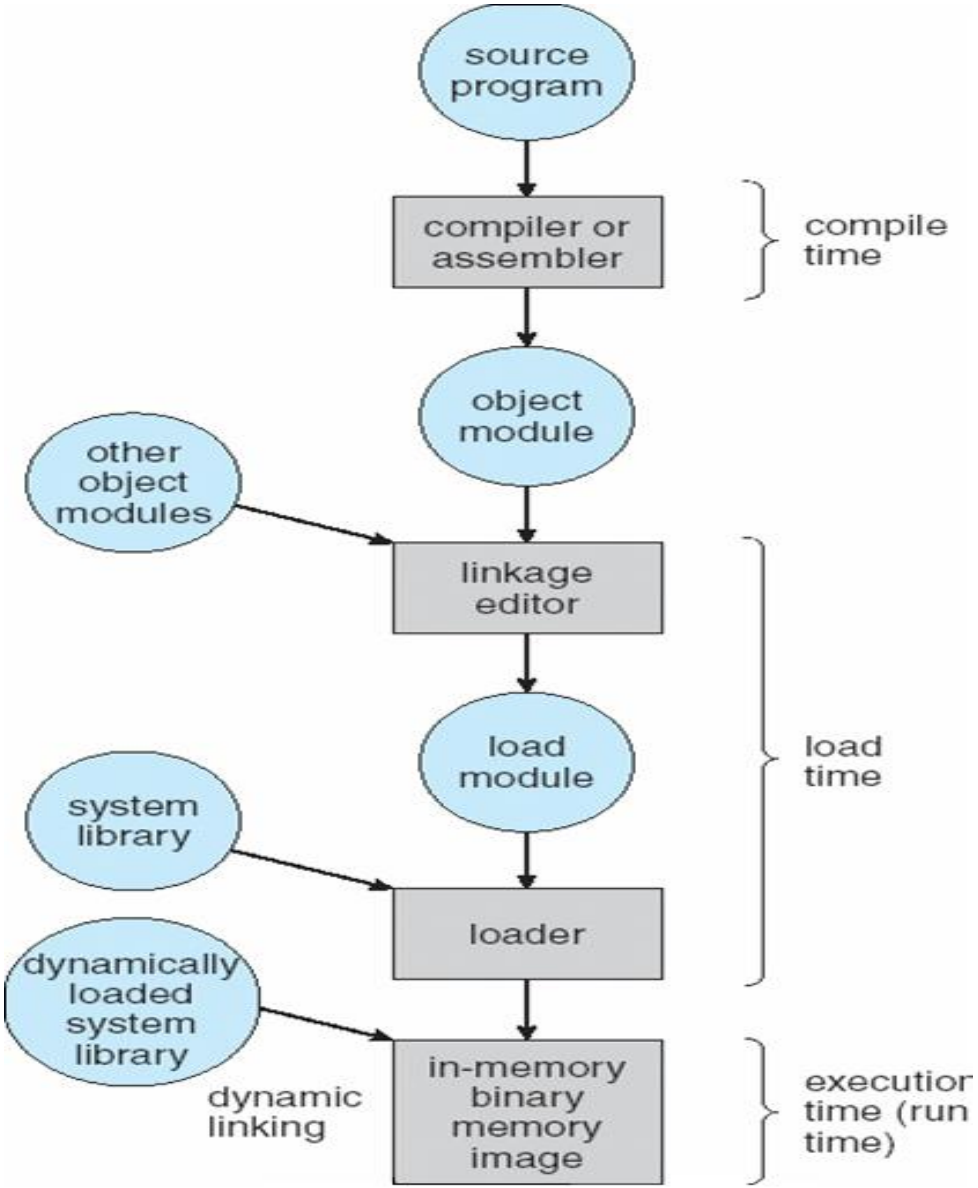


Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers)

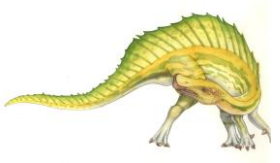


Multistep Processing of a User Program



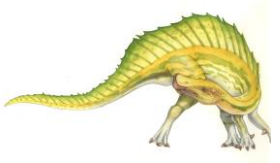
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

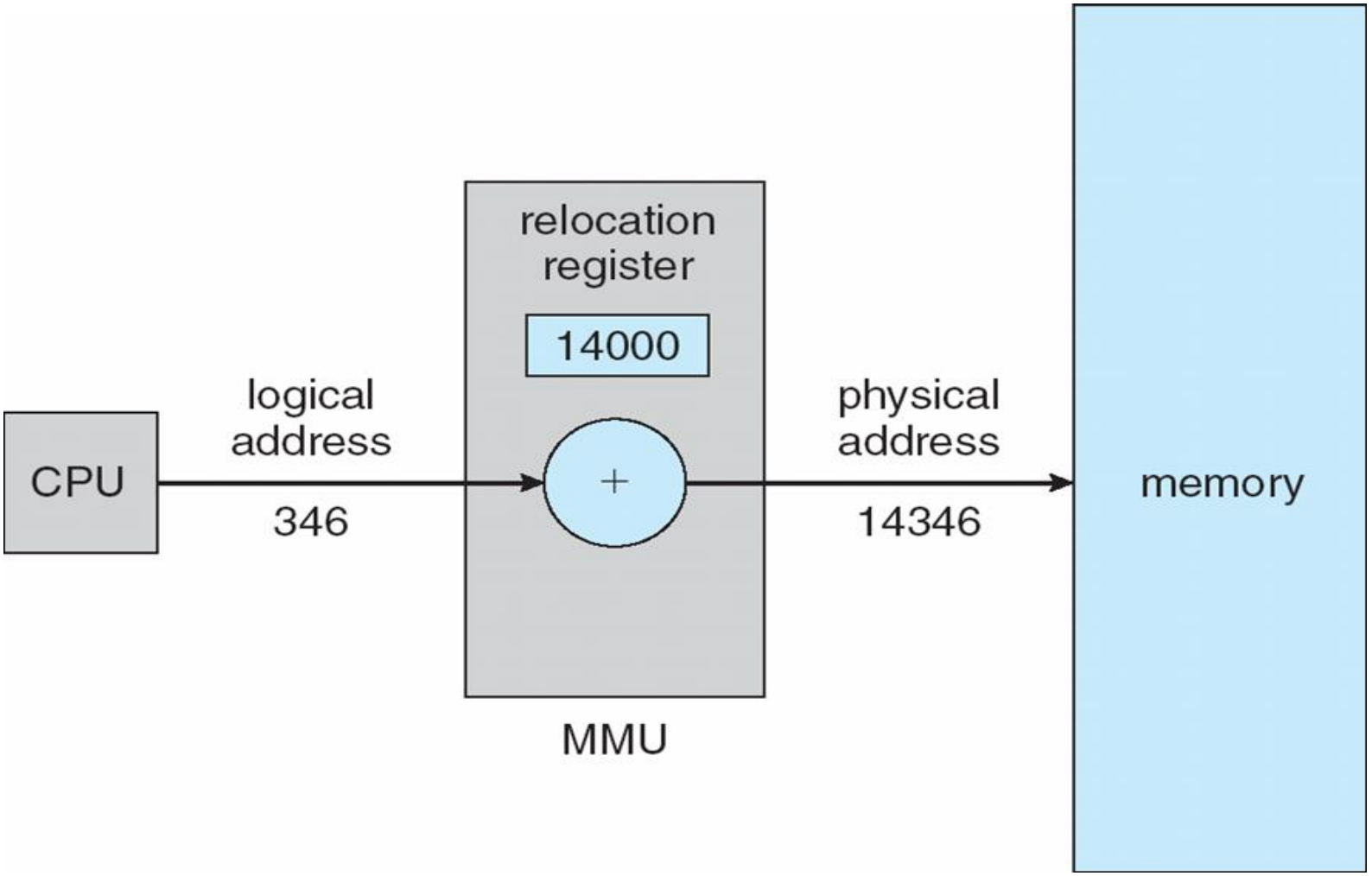


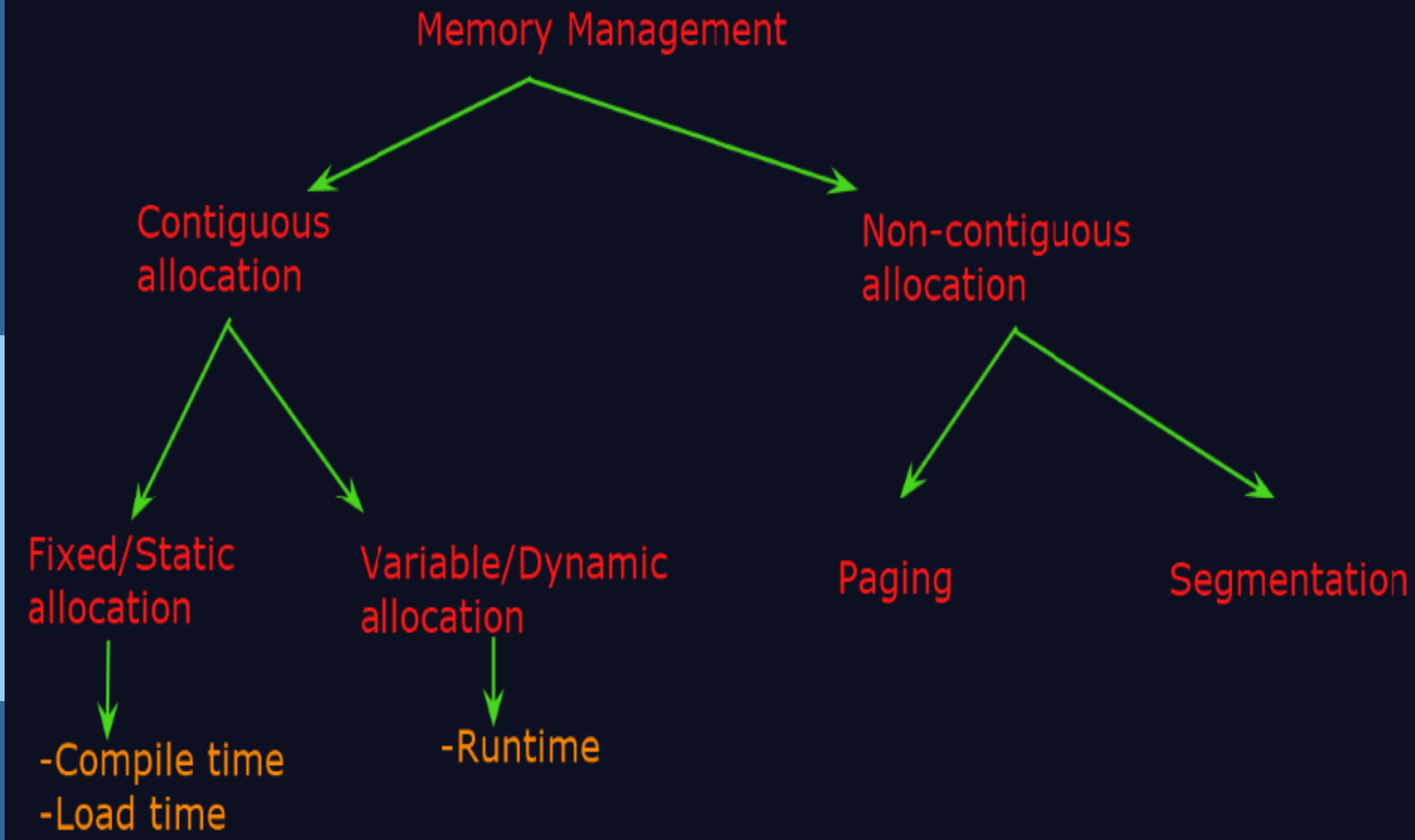
Memory-Management Unit (MMU)

- Hardware device that **maps virtual to physical address**
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with ***logical* addresses;** it never sees the *real* physical addresses



Dynamic relocation using a relocation register




Memory Management
Contiguous allocation
Non-contiguous allocation
Fixed/Static allocation
Variable/Dynamic allocation
Paging
Segmentation
-Compile time
-Load time
-Runtime

Contiguous Allocation

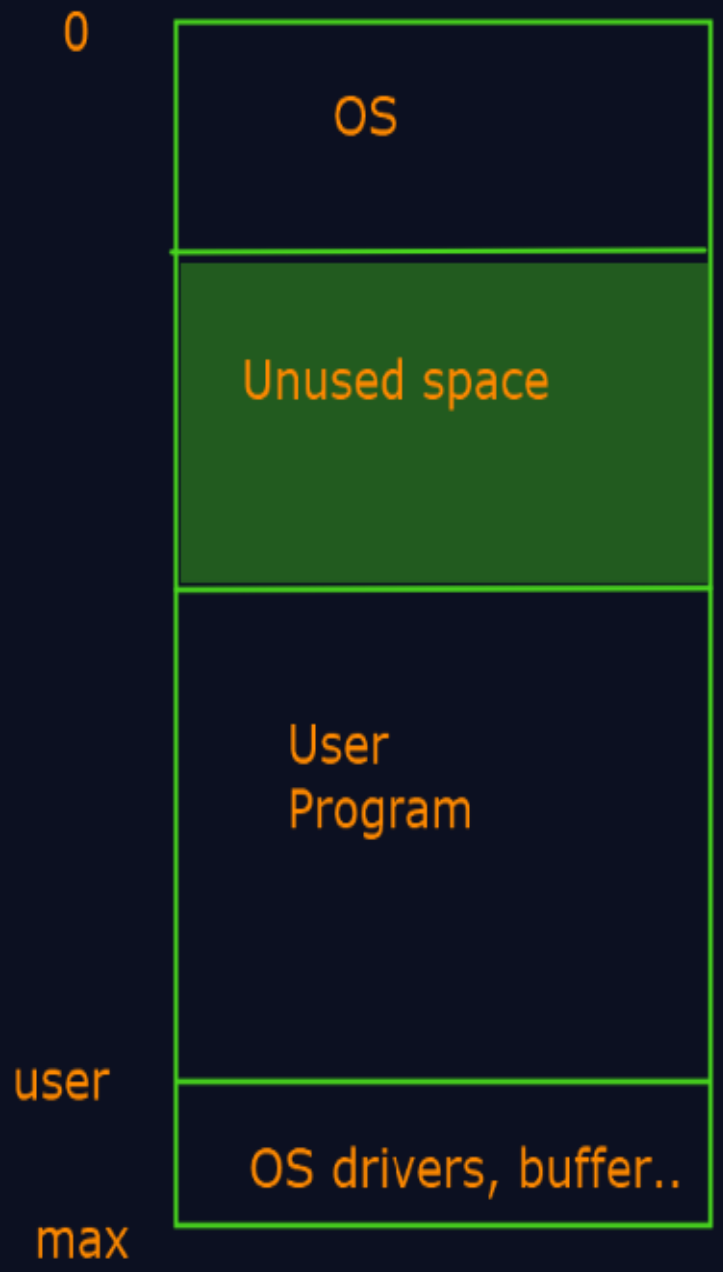
- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*

Partitions

Static

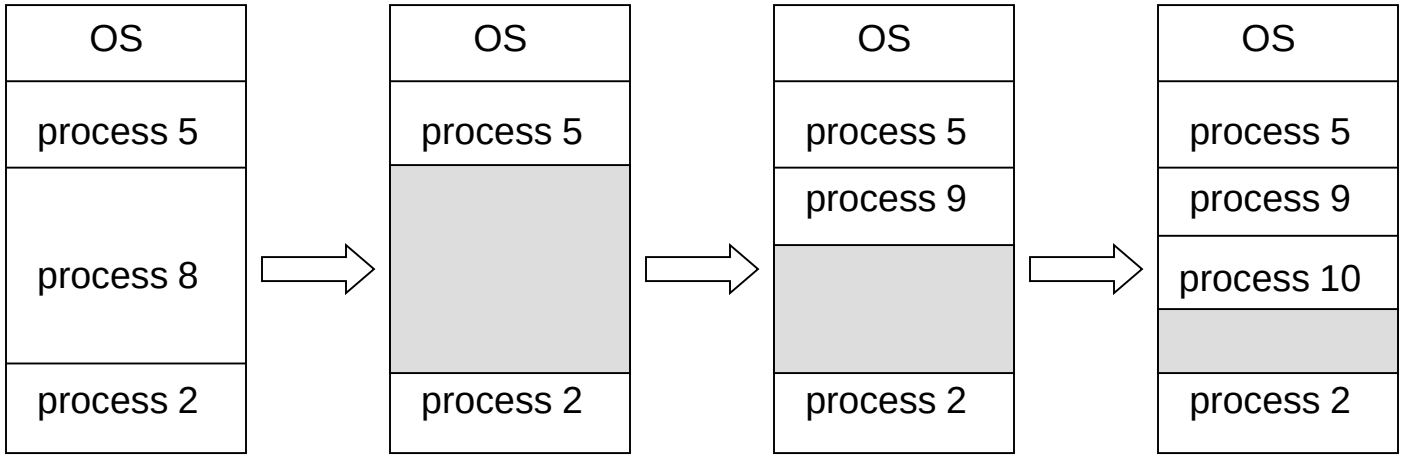
Dynamic

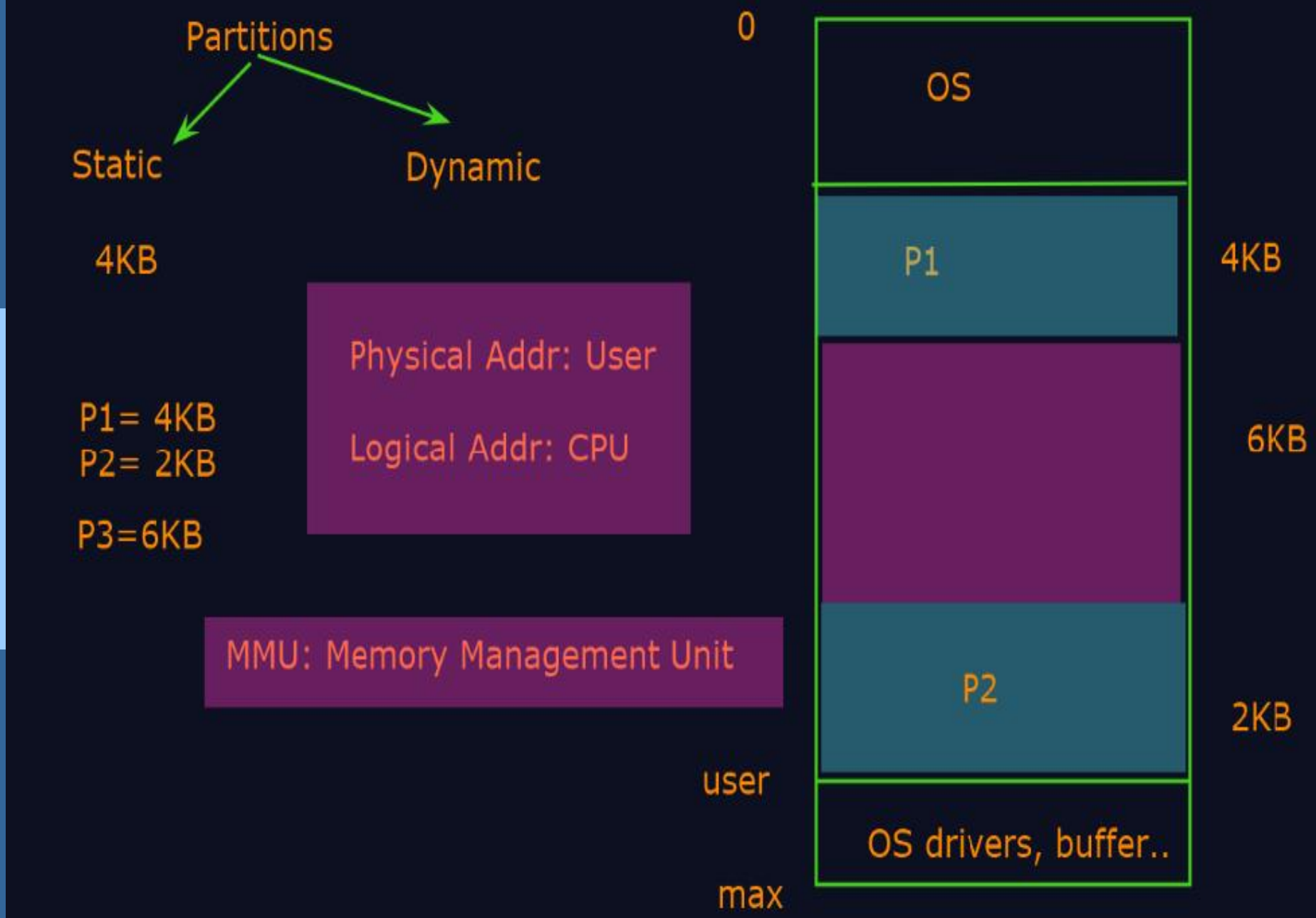
4KB

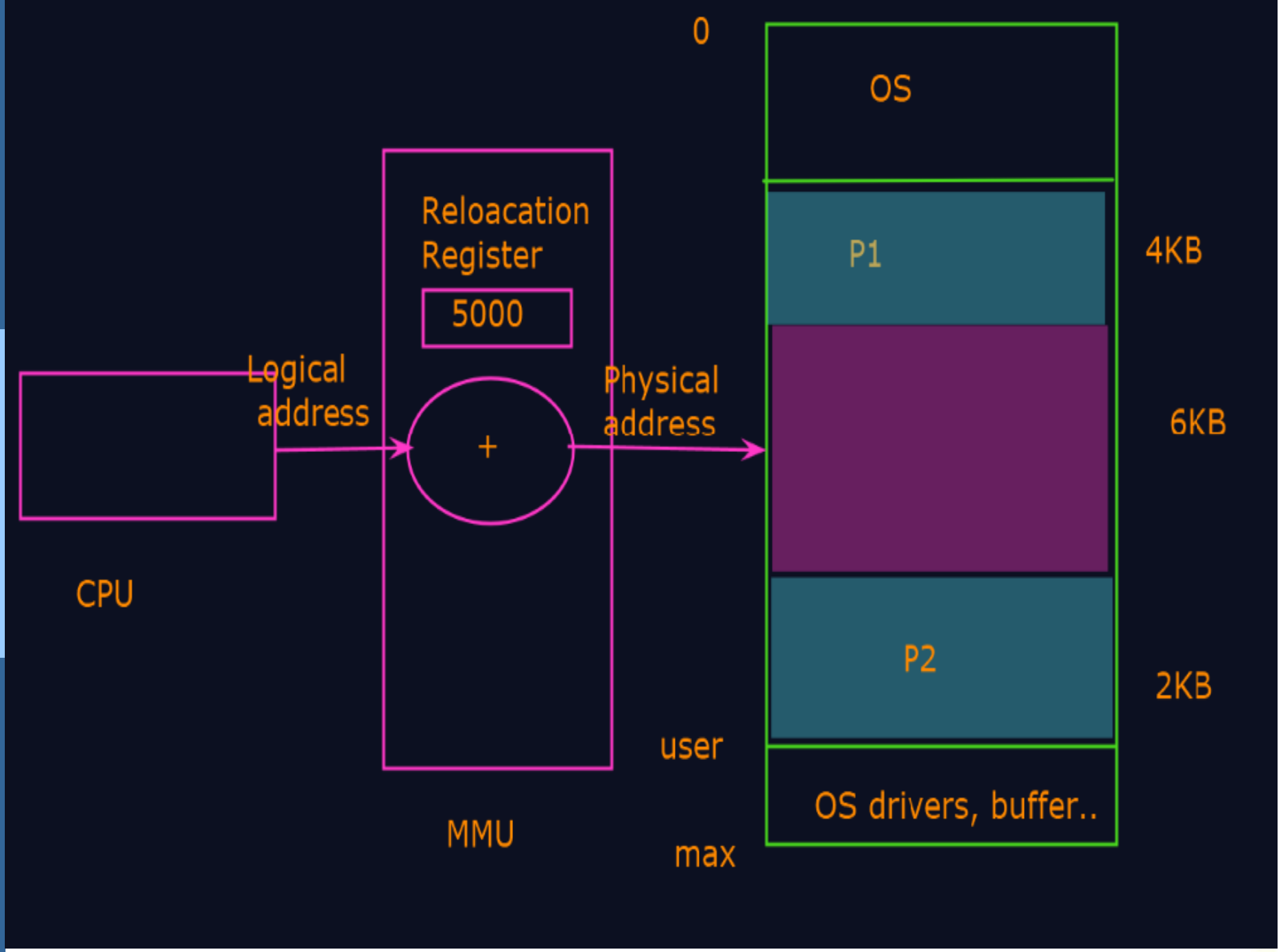


Contiguous Allocation (Cont)

- 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 (hole)





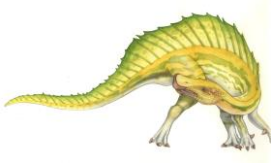


Dynamic Storage-Allocation Problem

How to satisfy a request of size *n* from a list of free holes

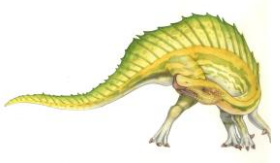
- **First-fit**: Allocate the *first* hole that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the *largest* hole; must also search entire list
  - Produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization



Contiguous Memory Allocation :

- The main memory should **oblige** both the operating system and the different client processes.
- Therefore, the **allocation of memory** becomes an important task in the operating system.
- The memory is **usually divided into two partitions**: one for the resident operating system and one for the user processes.
- We **normally need several user processes** to reside in memory simultaneously.
- Therefore, we need to consider how to allocate available memory to the processes



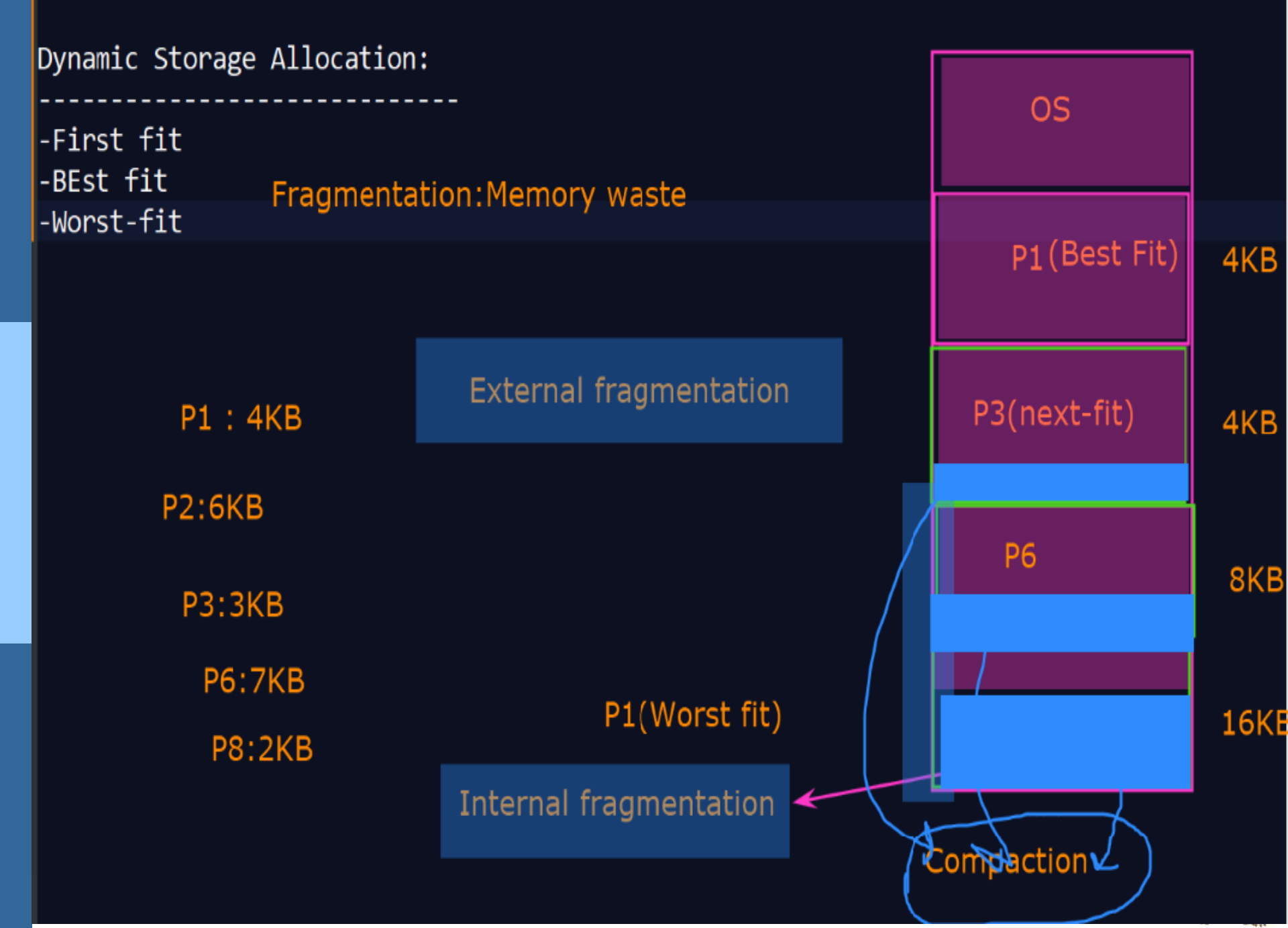
Dynamic Storage Allocation:

-First fit

-BEst fit

-Worst-fit

Fragmentation:Memory waste



Fixed Partitioning

The earliest and one of the **simplest technique** which can be used to load more than one processes into the main memory is Fixed partitioning or Contiguous memory allocation.

In this technique, the main memory is divided into partitions of equal or different sizes.

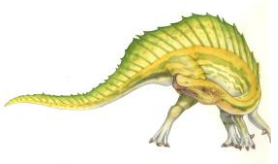
The operating system always resides in the first partition while the other partitions can be used to store user processes.

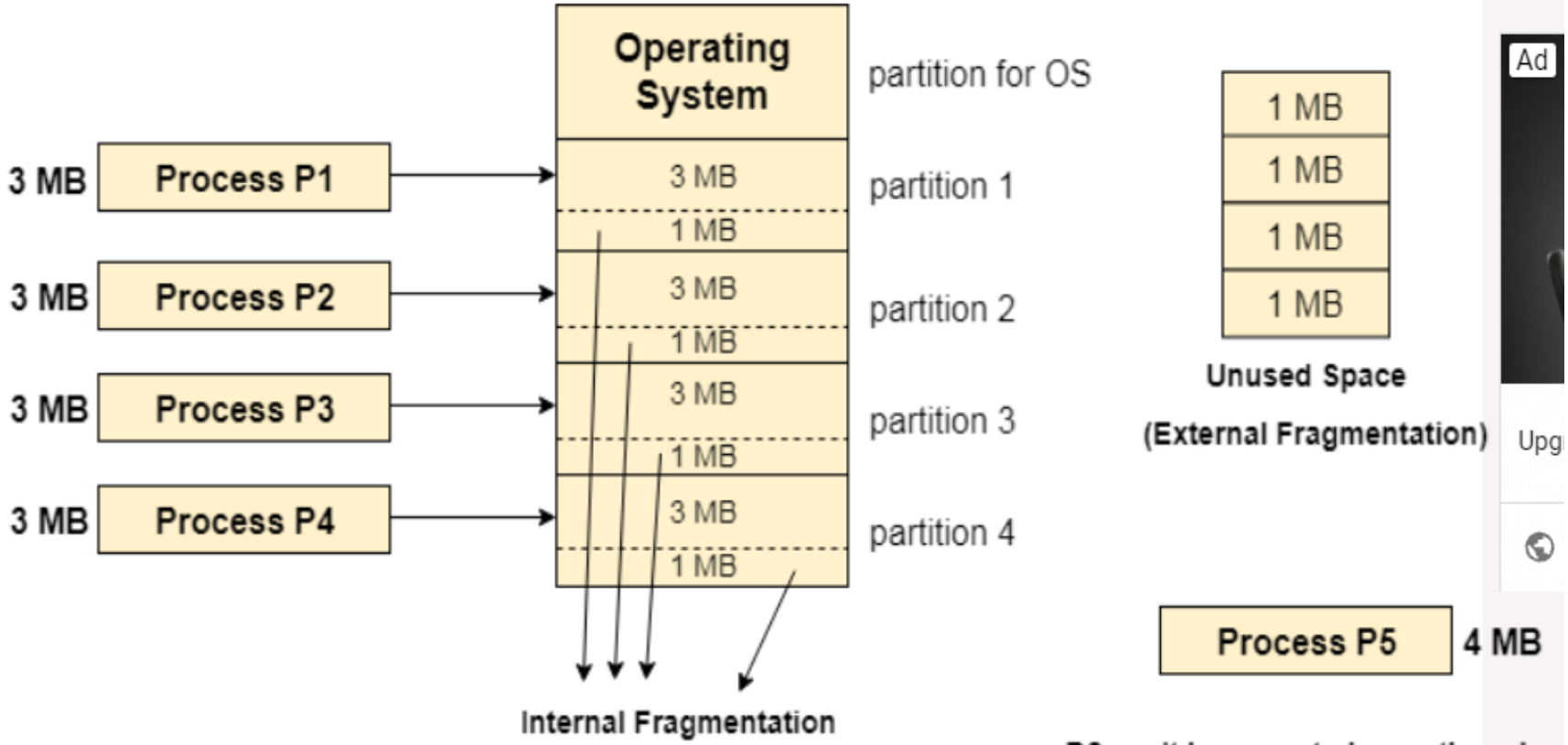
The memory is assigned to the processes in contiguous way.

In fixed partitioning,

1. The partitions cannot overlap.
2. A process must be contiguously present in a partition for the execution.

There are various cons of using this technique.





Fixed Partitioning

(Contiguous memory allocation)

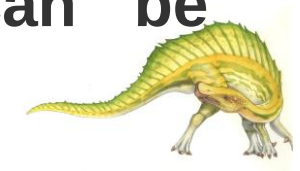
Dynamic Partitioning

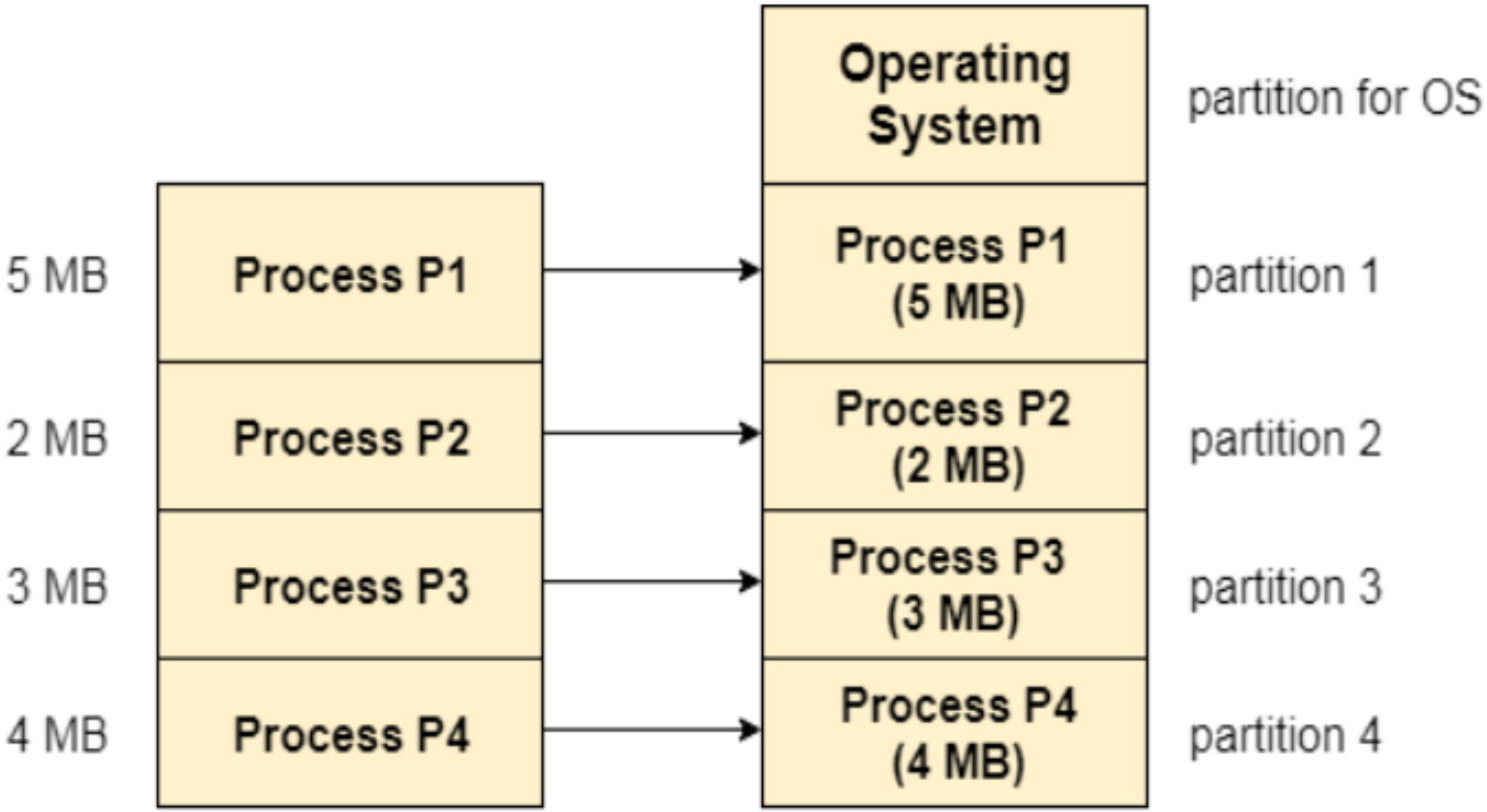
Dynamic partitioning **tries to overcome the problems caused by fixed partitioning.**

In this technique, the **partition size is not declared initially.**

It is **declared at the time of process loading**.

The first partition is **reserved for the operating system.**
The remaining space is divided into parts.
The size of each partition will be equal to the size of the process.
The partition size varies according to the need of the process so that the **internal fragmentation can be avoided.**





Dynamic Partitioning

(Process Size = Partition Size)

Dynamic Storage Allocation:

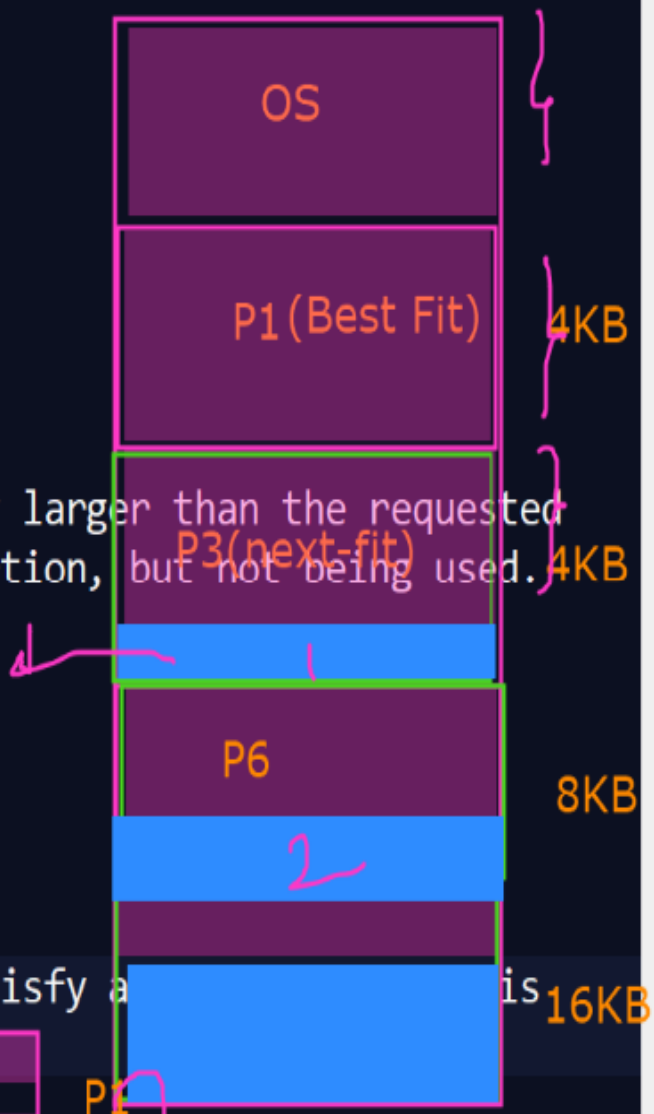
-First fit

-BEst fit

-Worst-fit

Fragmentation:

Internal fregmentation: allocated memory may be slightly larger than the requested memory, this size difference is memory internal to a partion, but not being used.



External Fragmentation: total memory space exists to satisfy a is not contiguous

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is **possible *only* if relocation is dynamic**, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Fragmentation

Fragmentation refers to the unused memory that the memory management system cannot allocate.

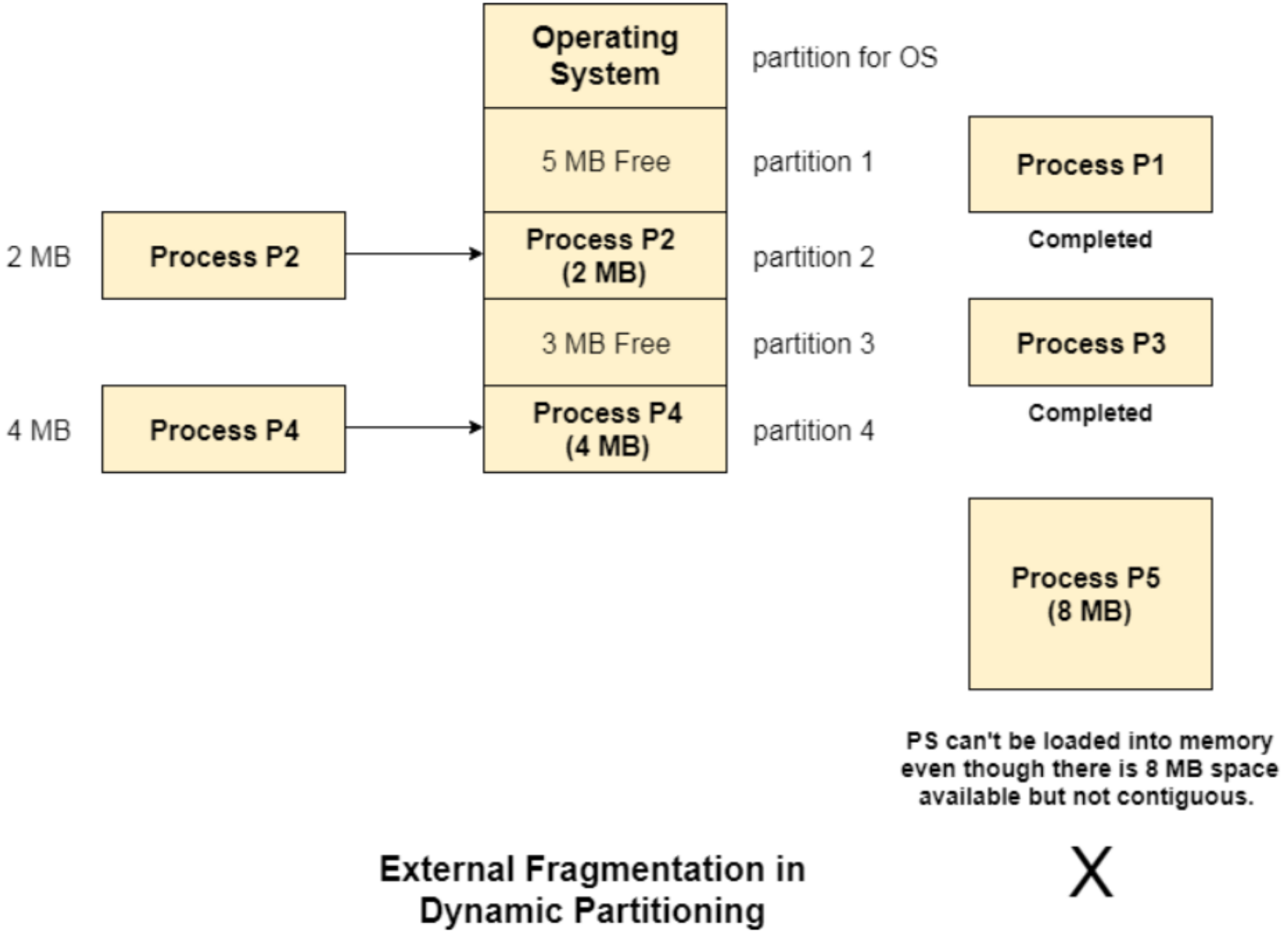
- *Internal fragmentation*

  Waste of memory *within* a partition, caused by the difference between the size of a partition and the process loaded. Severe in static partitioning schemes.

- *External fragmentation*

  Waste of memory *between* partitions, caused by scattered noncontiguous free space. Severe in dynamic partitioning schemes.

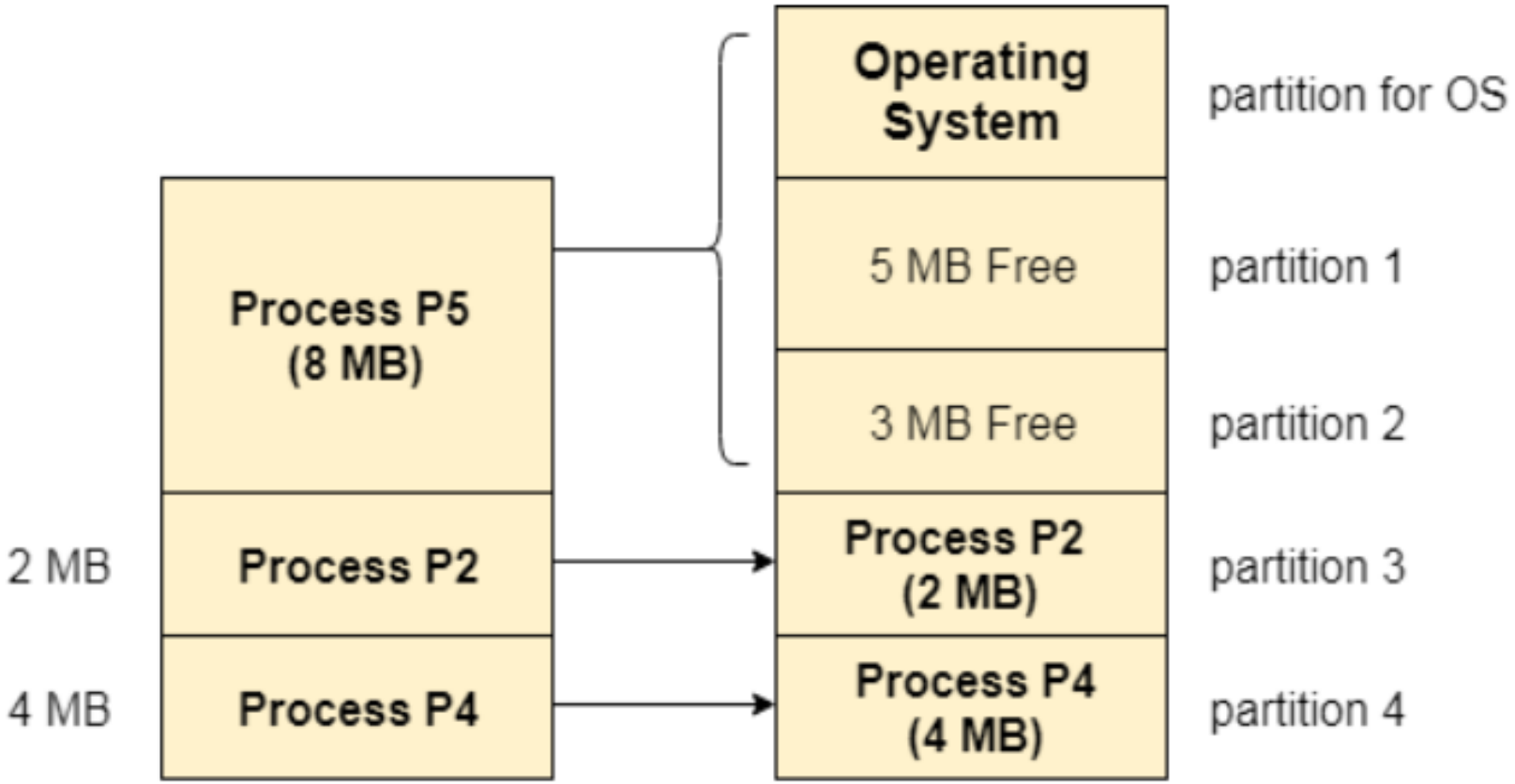
Compaction (aka relocation) is a technique that is used to overcome external fragmentation.



External Fragmentation in Dynamic Partitioning

Compaction

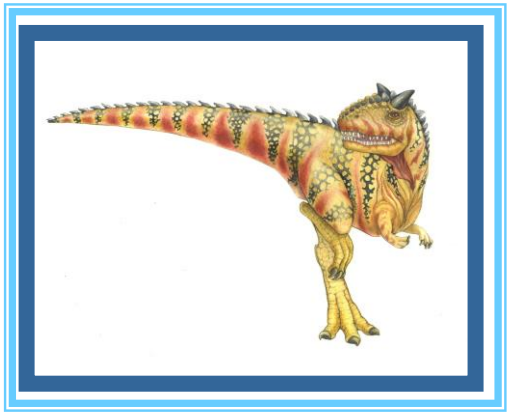
- We got to know that the **dynamic partitioning suffers from external fragmentation**. However, this can cause some serious problems.
- To avoid compaction, we need to change the rule which says that the process can't be stored in the different places in the memory.
- We can also **use compaction to minimize the probability of external fragmentation**. In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together.
- By applying this technique, we can store the bigger processes in the memory.
- The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called defragmentation.

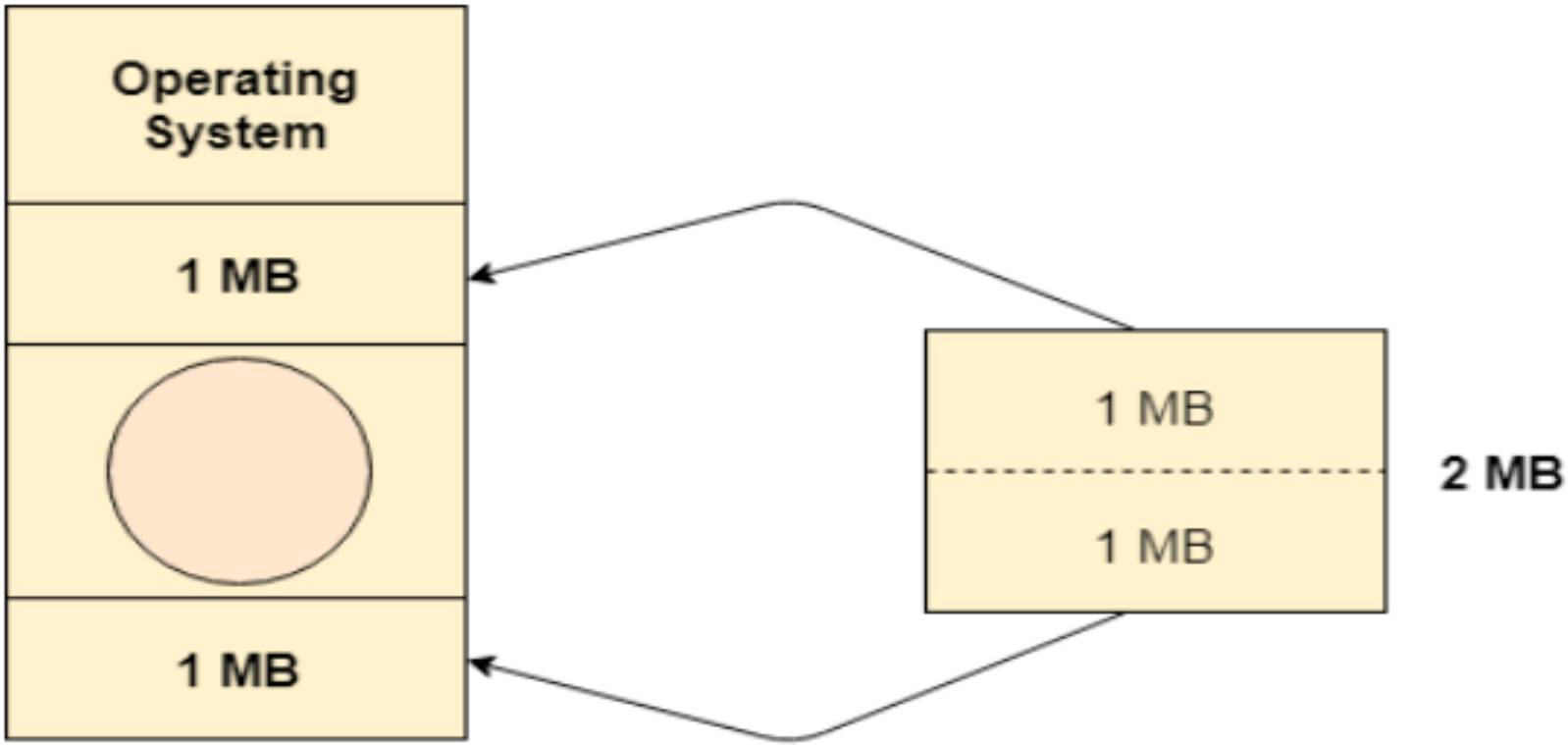


Now PS can be loaded into memory because the free space is now made contiguous by compaction

Compaction

Paging and Segmentation

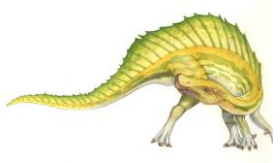


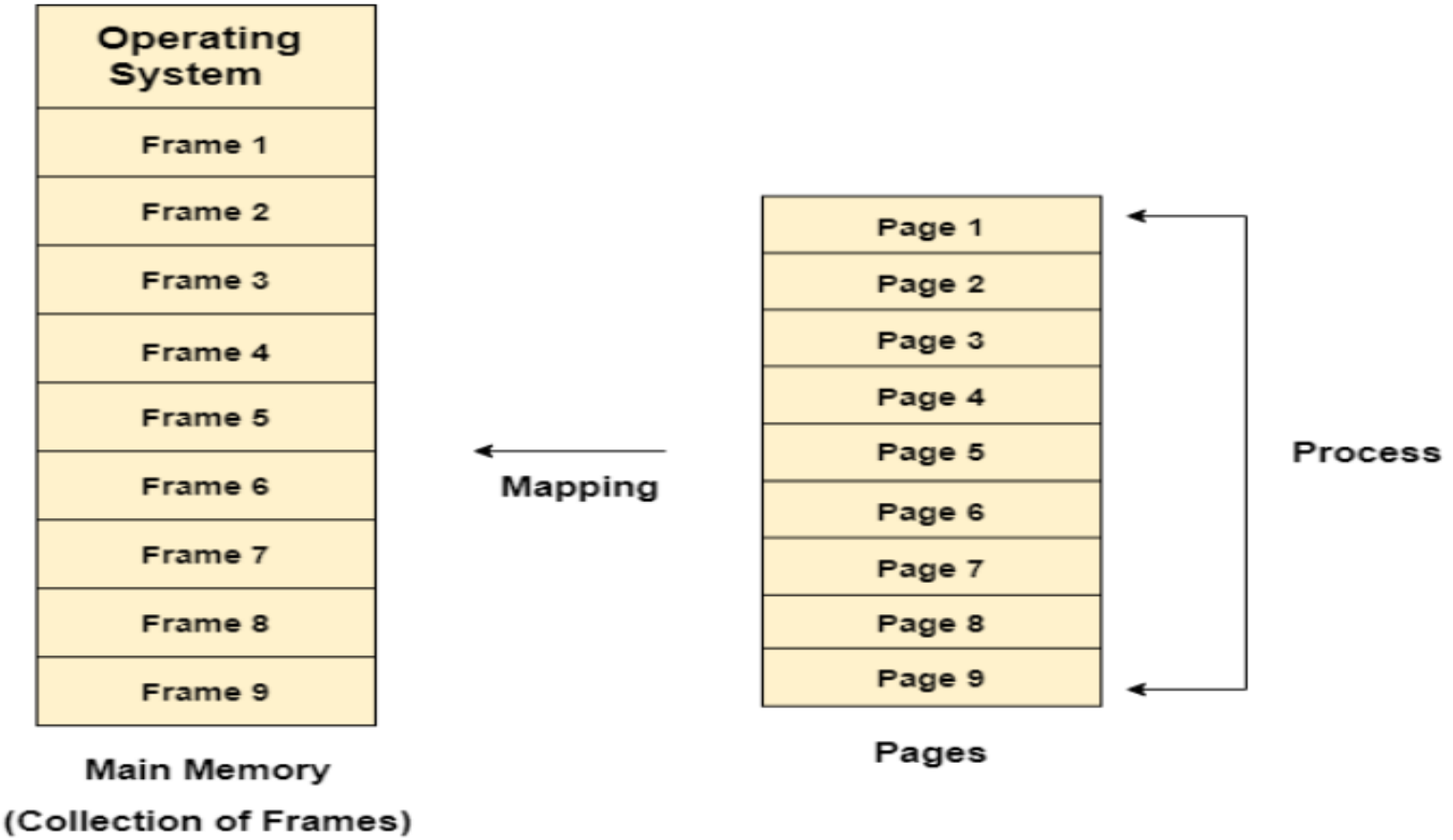


The process needs to be divided into two parts to get stored at two different places.

Need for Paging

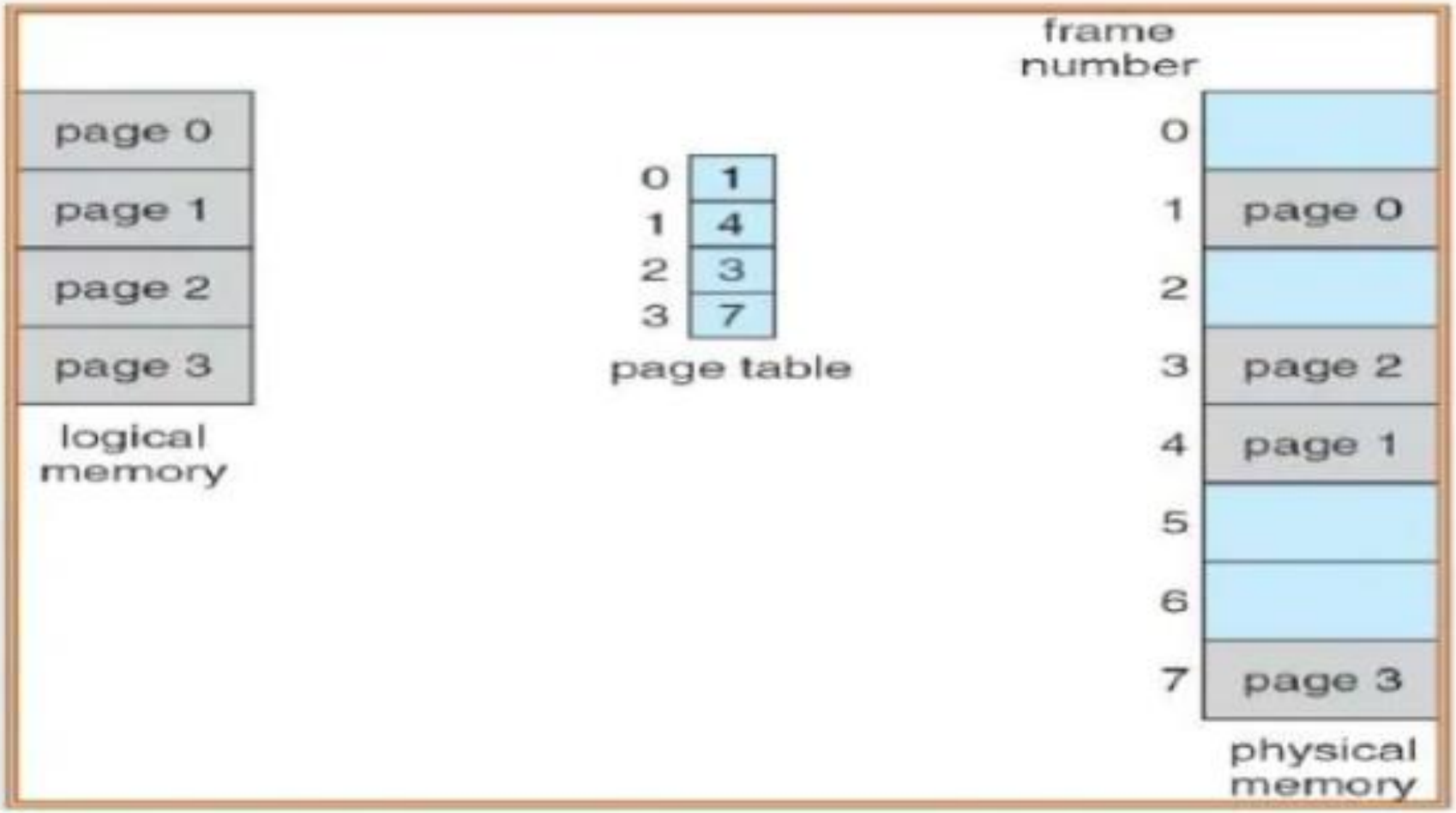
- Disadvantage of Dynamic Partitioning
  - The main disadvantage of Dynamic Partitioning is **External fragmentation.** Although, this can be **removed by Compaction** but as we have discussed earlier, the compaction makes the system inefficient.
  - We **need to find out a mechanism which can load the processes in the partitions** in a more optimal way. Let us discuss a dynamic and flexible mechanism **called paging.**
- **Need for Paging**
- Lets consider a process P1 of size 2 MB and the main memory which is divided into three partitions. Out of the three partitions, two partitions are holes of size 1 MB each.
- P1 needs 2 MB space in the main memory to be loaded. **We have two holes of 1 MB each but they are not contiguous.**

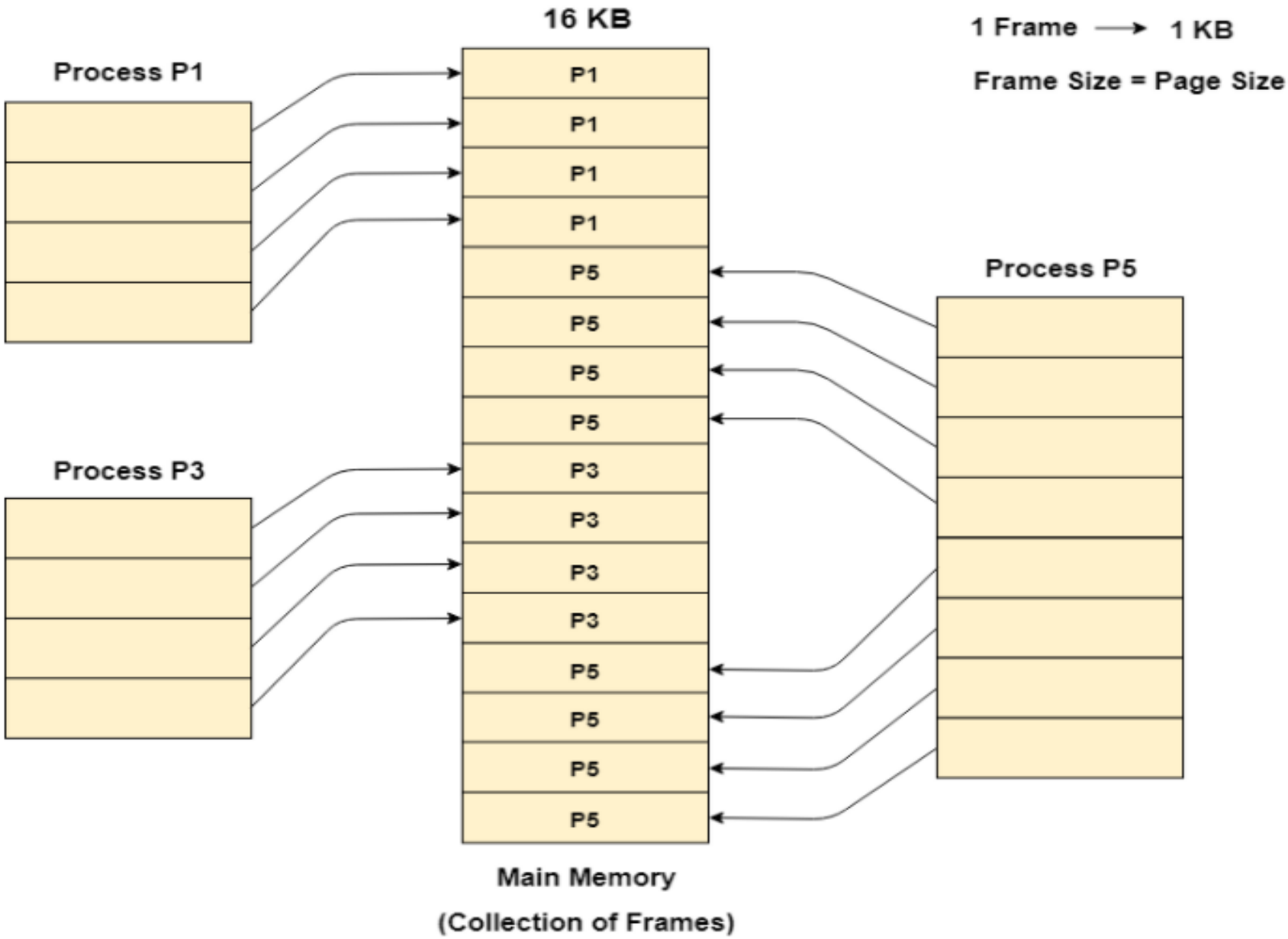




- The main **idea behind the paging is to divide each process** in the **form of pages**. The main memory will also be **divided in the form of frames**.
- **One page of the process is to be stored in one of the frames of the memory**. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.
- Pages of the process are **brought into the main memory** only when they are required otherwise they reside in the secondary storage.

Paging Example




16 KB
1 Frame → 1 KB
Frame Size = Page Size
Process P1
Process P3
Process P5
P1
P1
P1
P1
P5
P5
P5
P5
P3
P3
P3
P3
P5
P5
P5
P5
Main Memory
(Collection of Frames)

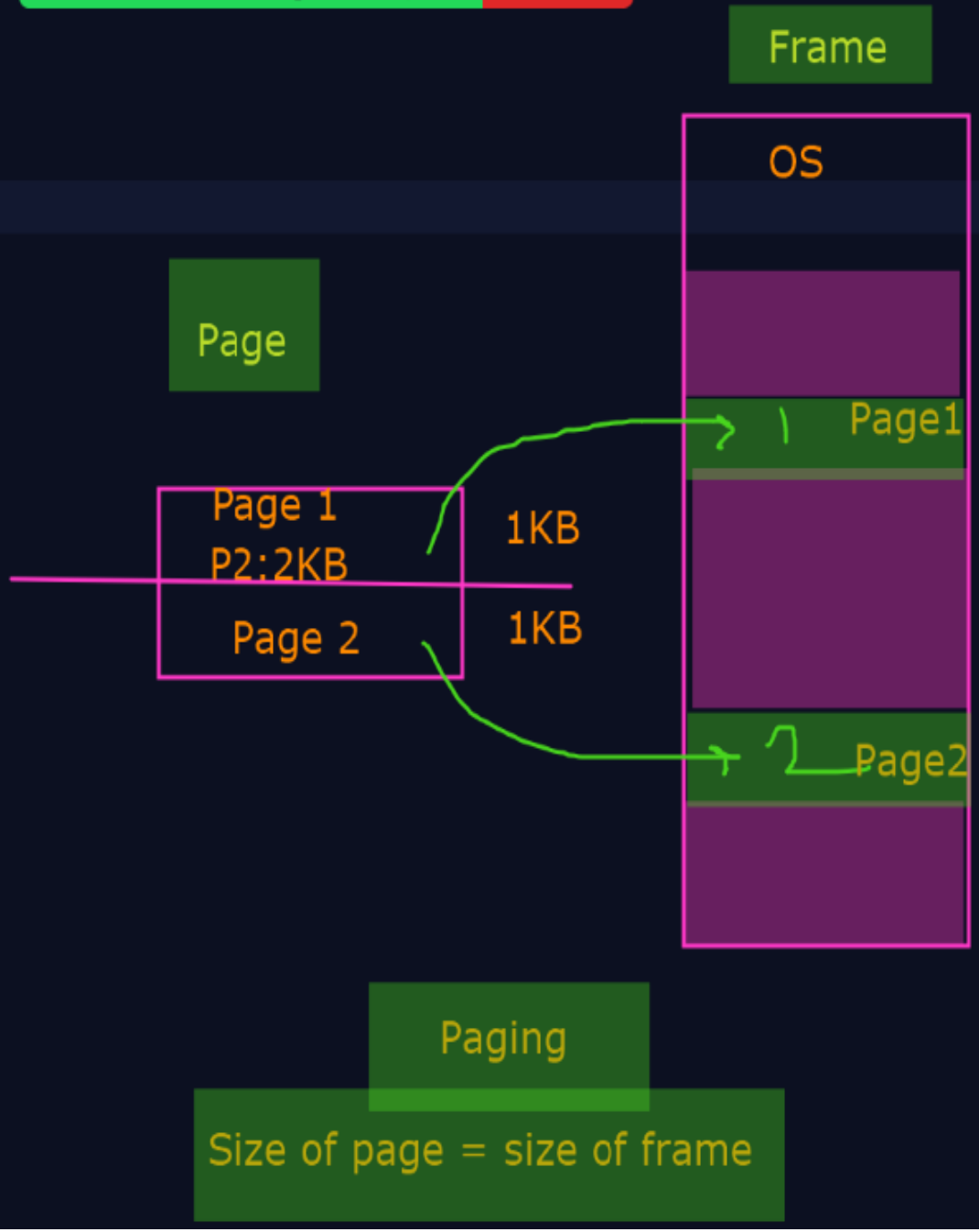
Paging

not contiguous

Non-Contiguous allocation:

---

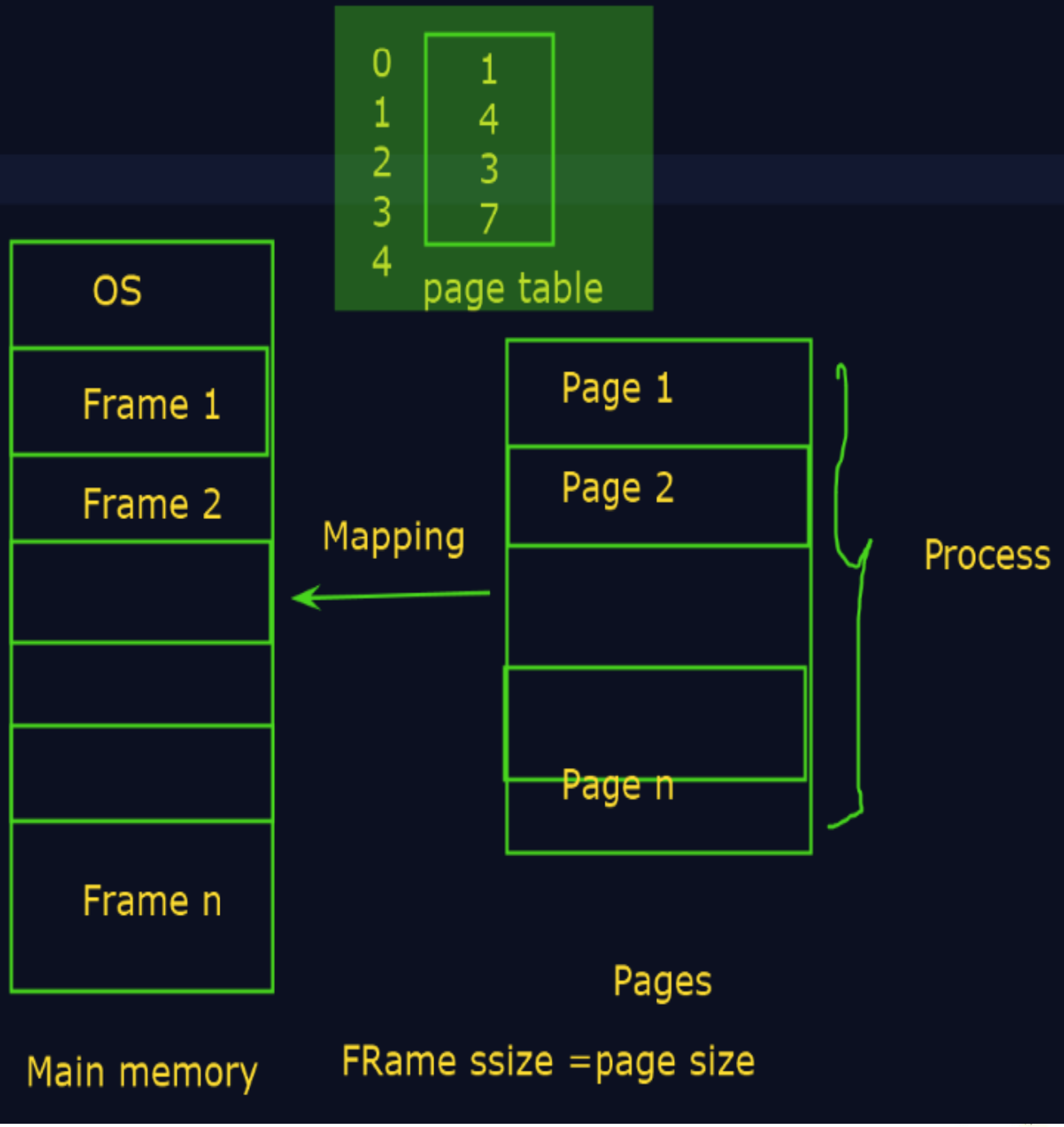
1. Paging
2. Segmentation



Non-Contiguous allocation:

---

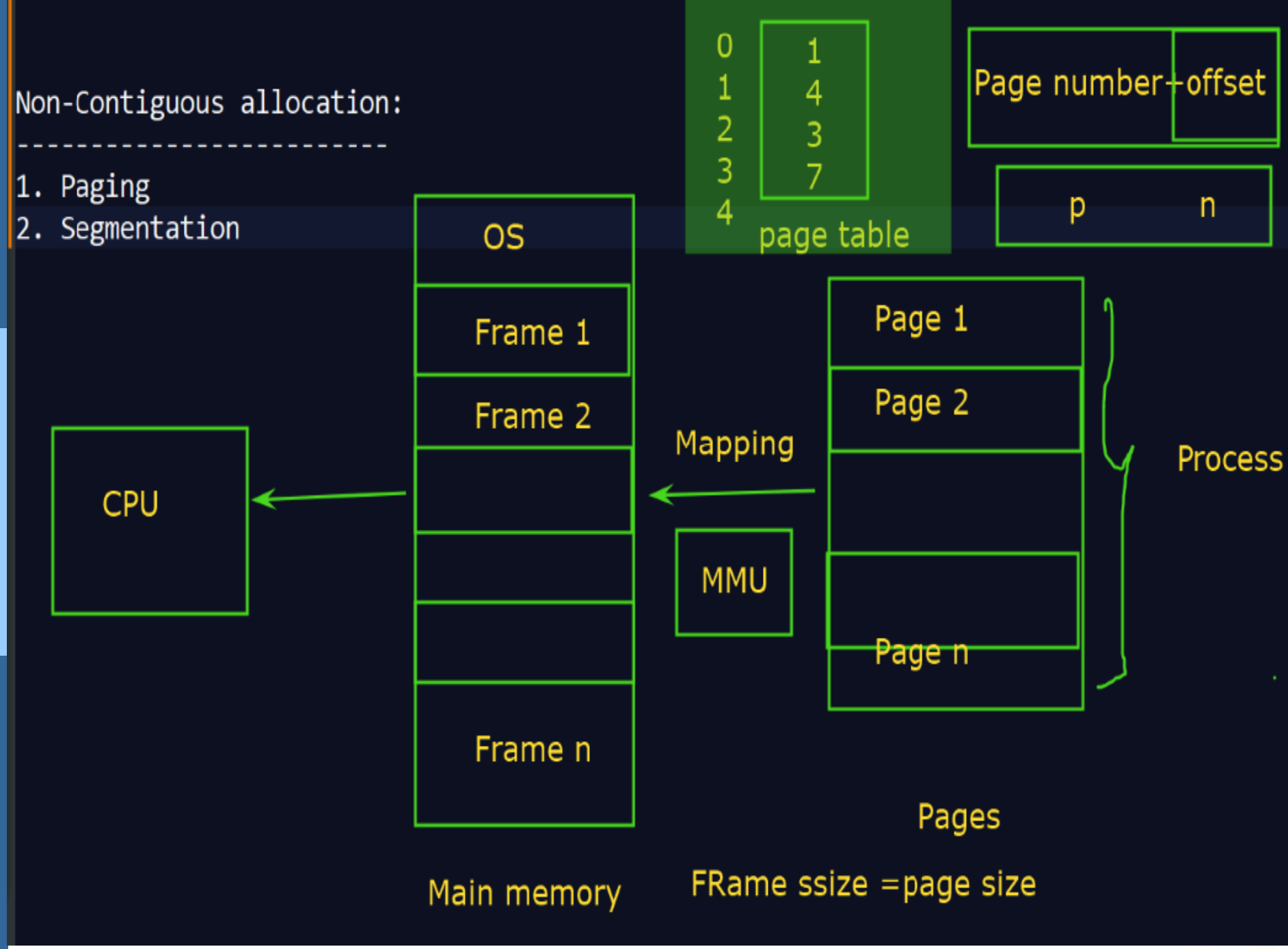
1. Paging
2. Segmentation



Non-Contiguous allocation:

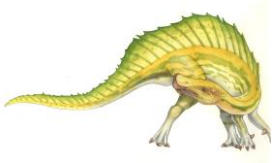
---

1. Paging
2. Segmentation

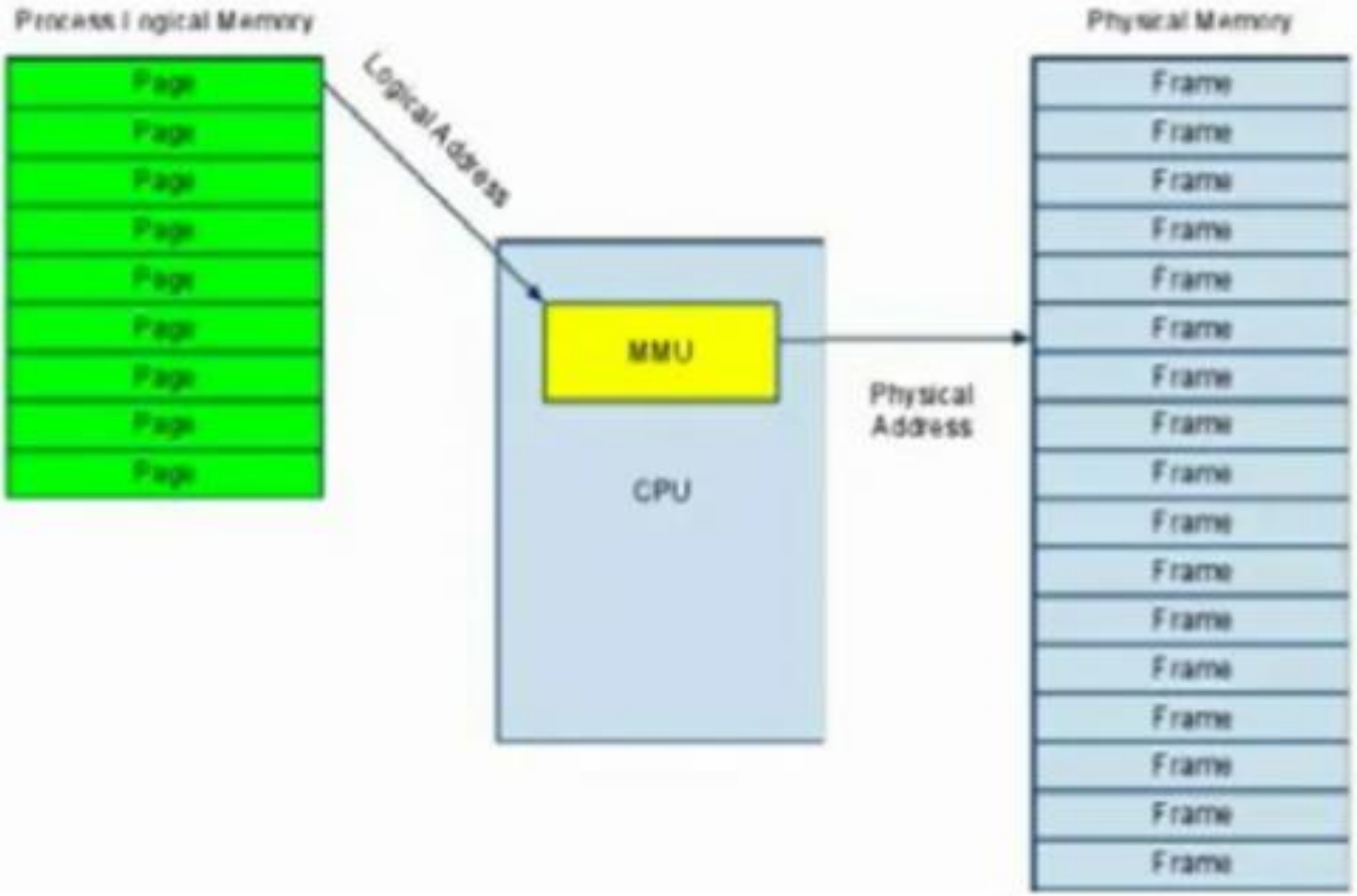


Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***n*** pages, need to find *n* free frames and load program
- Set up a page table to translate logical to physical addresses
- Internal fragmentation



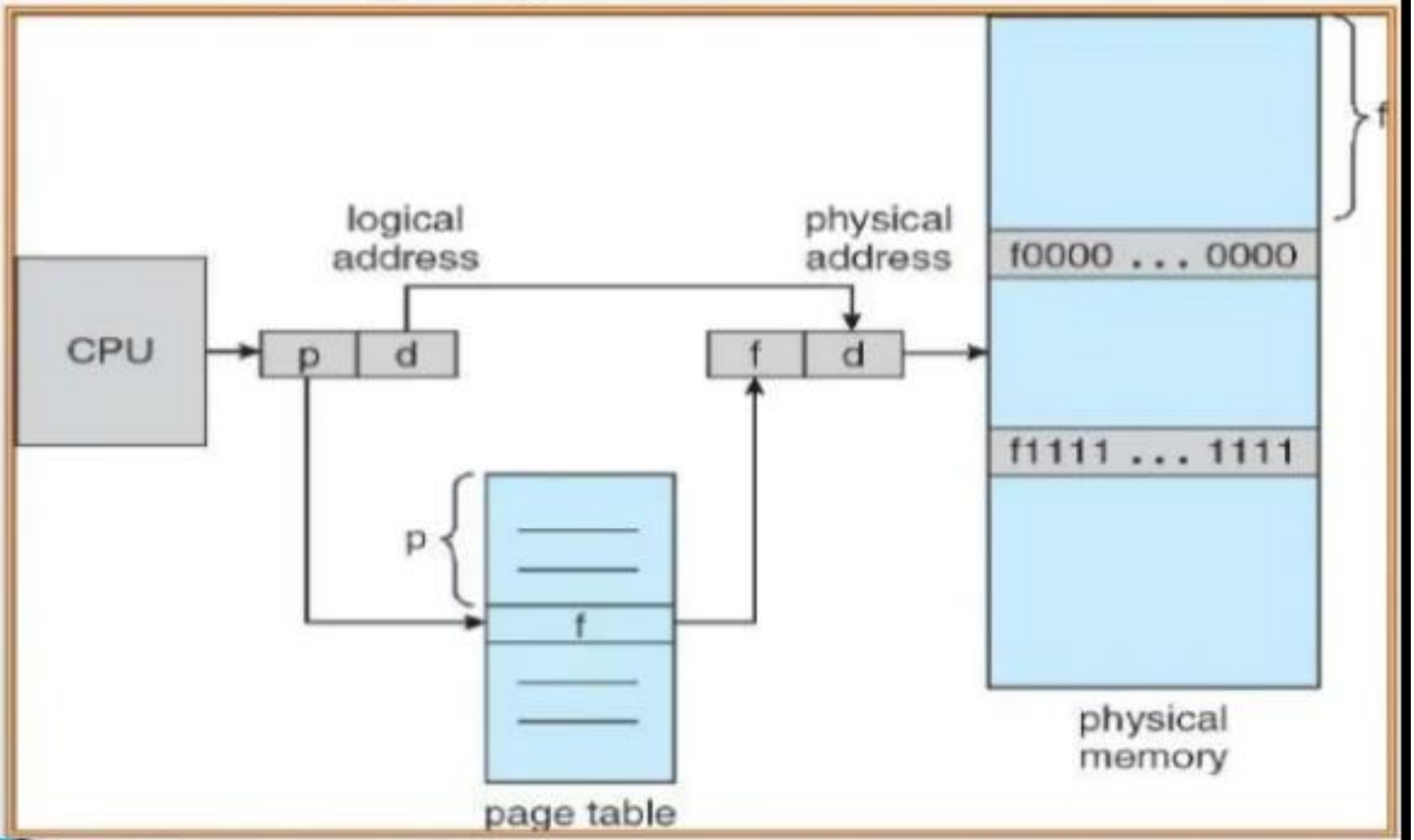
Page Translation



Memory Management Unit

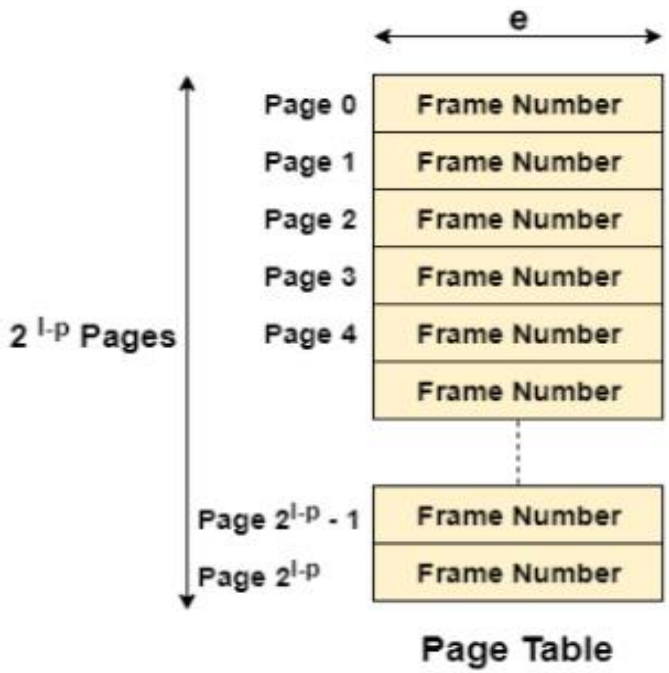
- The purpose of Memory Management Unit (MMU) is **to convert the logical address into the physical address.**
- The logical address is the address generated by the CPU for every page while the physical address is the actual address of the frame where each page will be stored.
- When a page is to be accessed by the CPU by using the logical address, the operating system needs to obtain the physical address to access that page physically.
- The logical address has two parts.
  - Page Number
  - Offset
- Memory management unit of OS needs to convert the page number to the frame number.

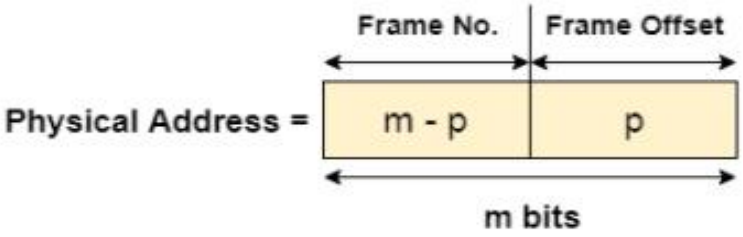
Paging Hardware

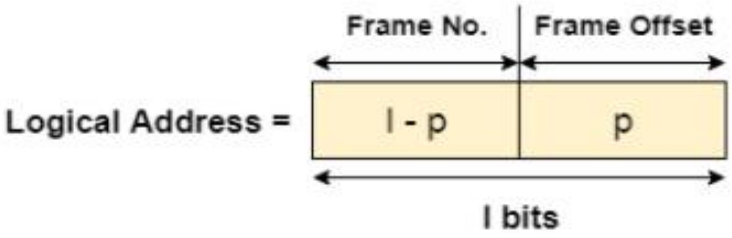


Page Table

- Page Table is a data structure used by the virtual memory system to store the mapping between logical addresses and physical addresses.



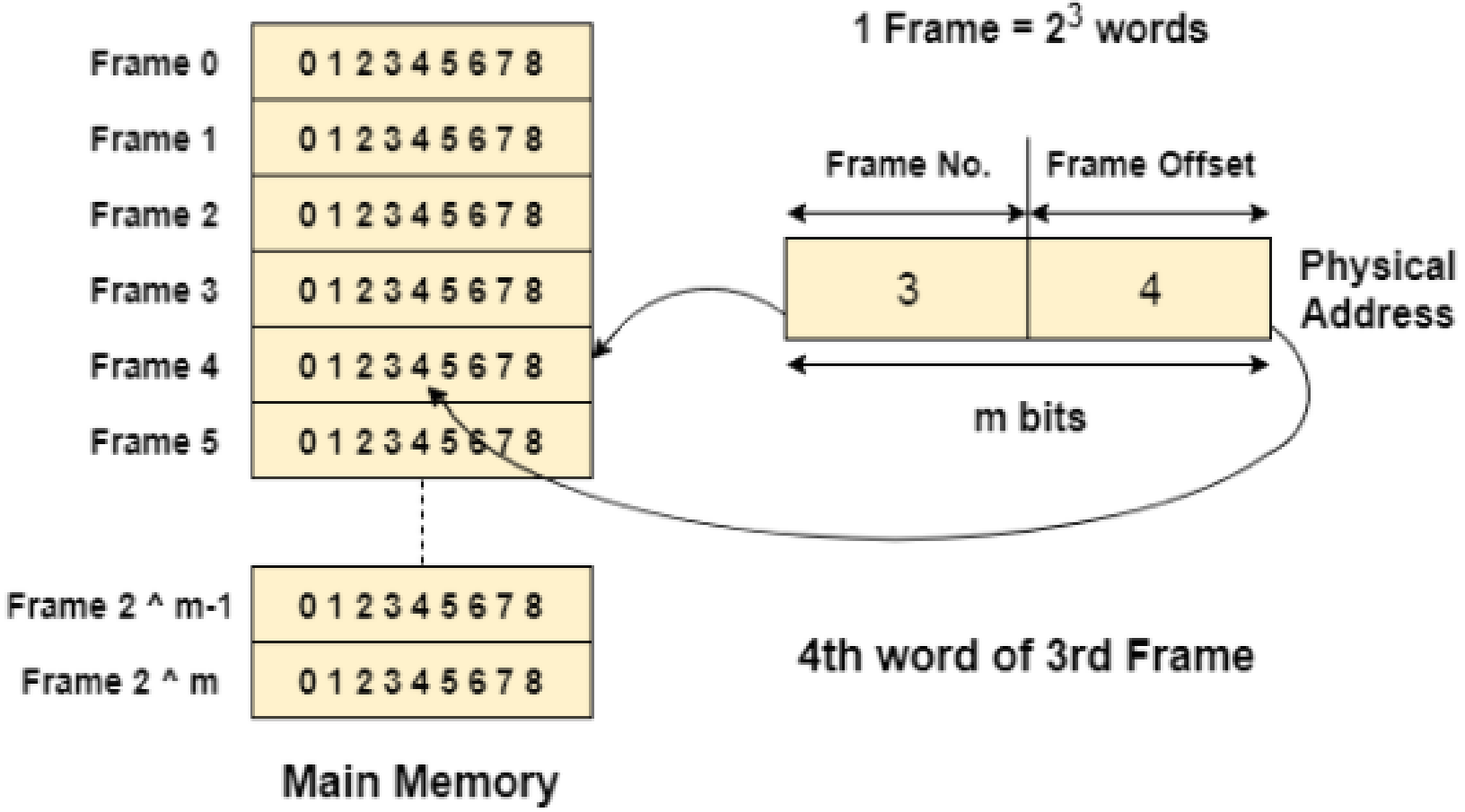


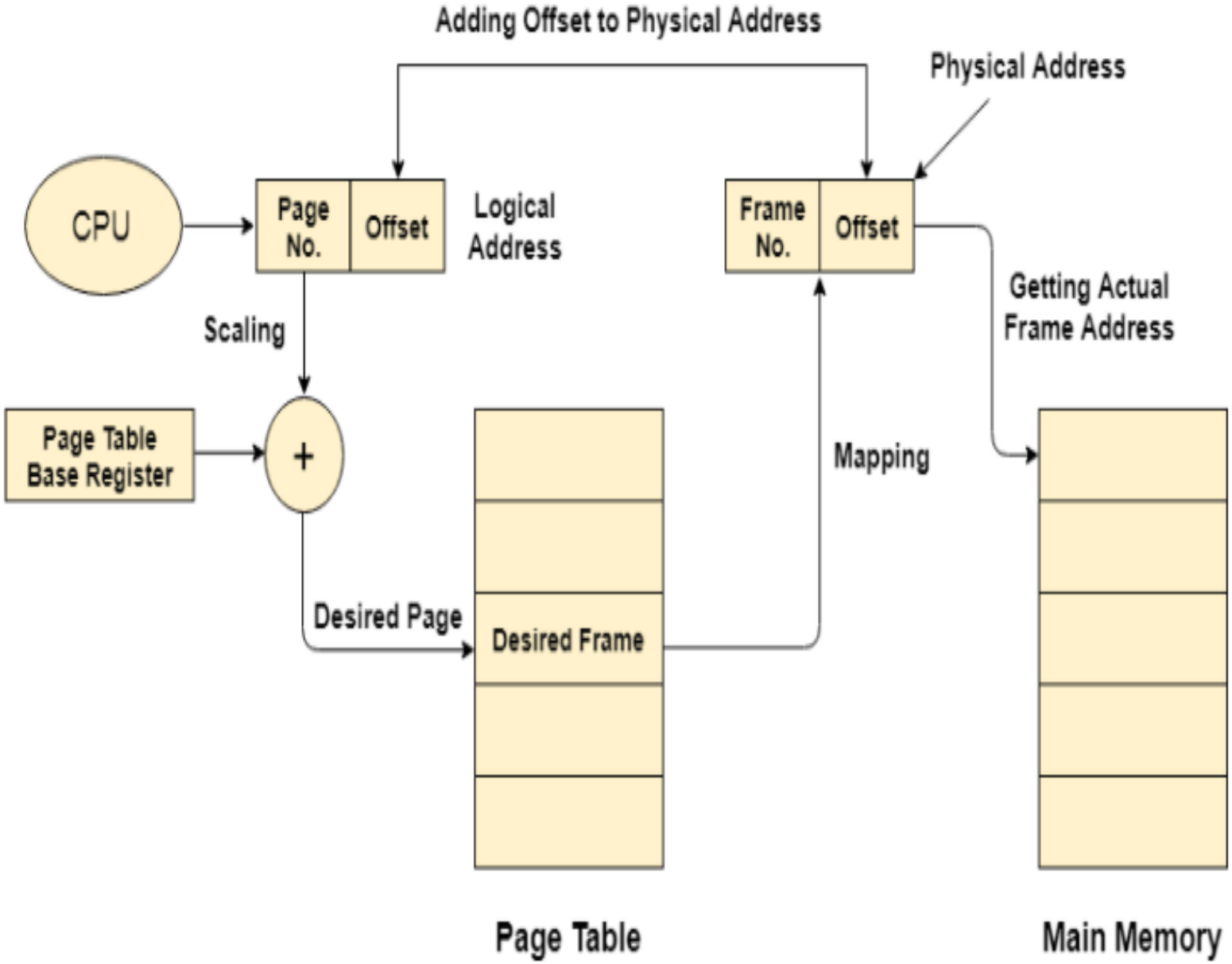


No. of entries in Page Table = No. of the pages in the process

Page Table Size = 2^{l-p} X e bytes

e = m-p (Frame Size) bits



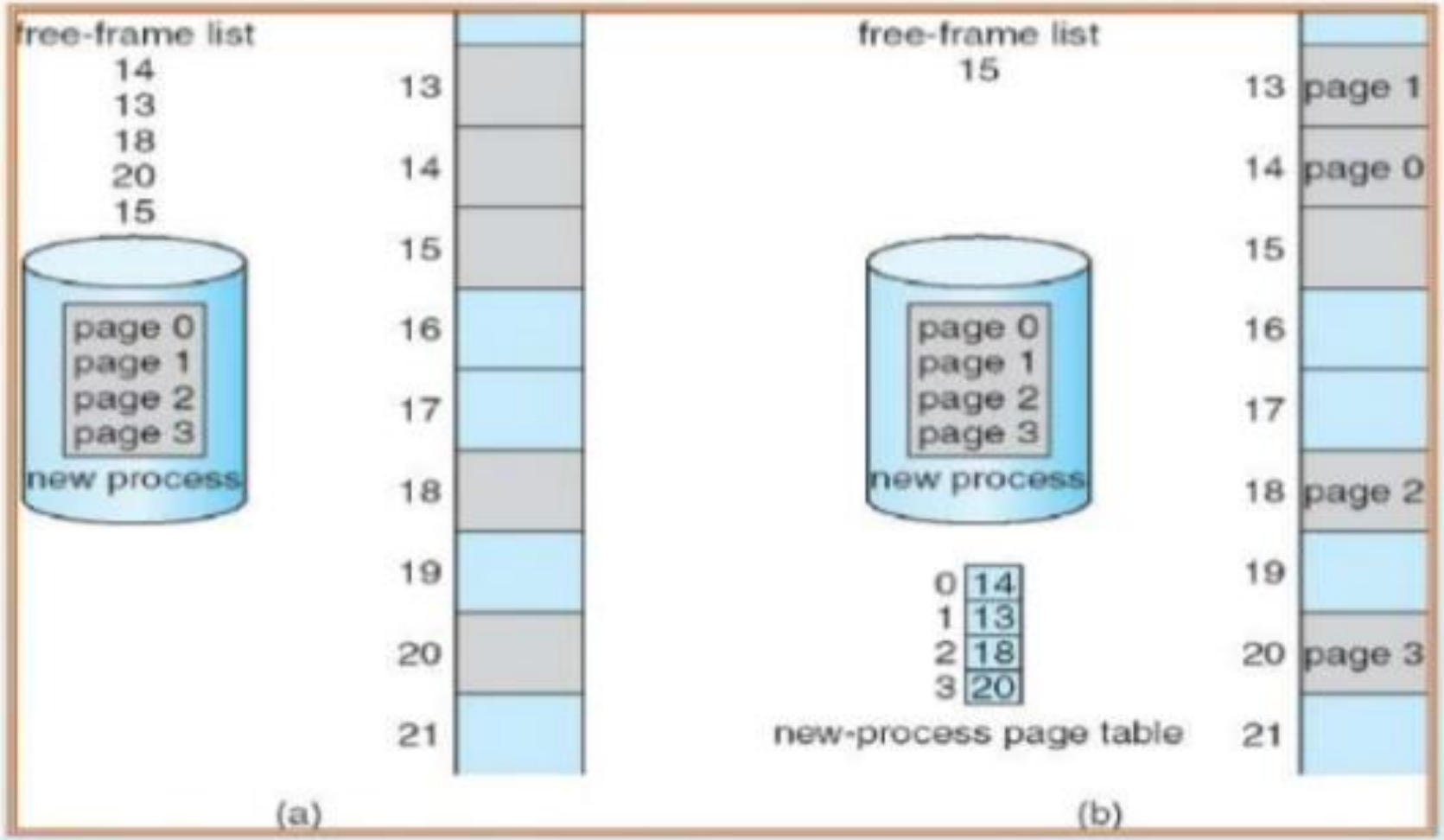


Paging Example

- When we use a paging scheme, we have no external fragmentation: ANY free frame can be allocated to a process that needs it.
- However, we may have internal fragmentation

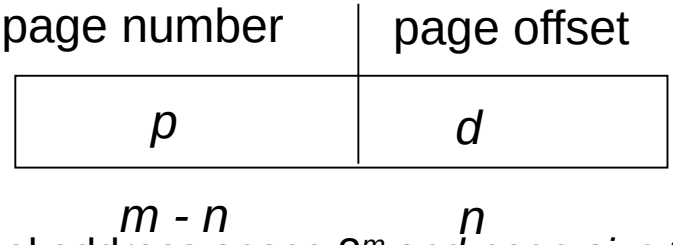
- If the process requires n pages, at least n frames are required
- The first page of the process is loaded into the first frame listed on free-frame list, and the frame number is put into page table

Paging Example

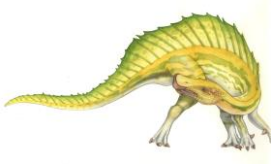


Address Translation Scheme

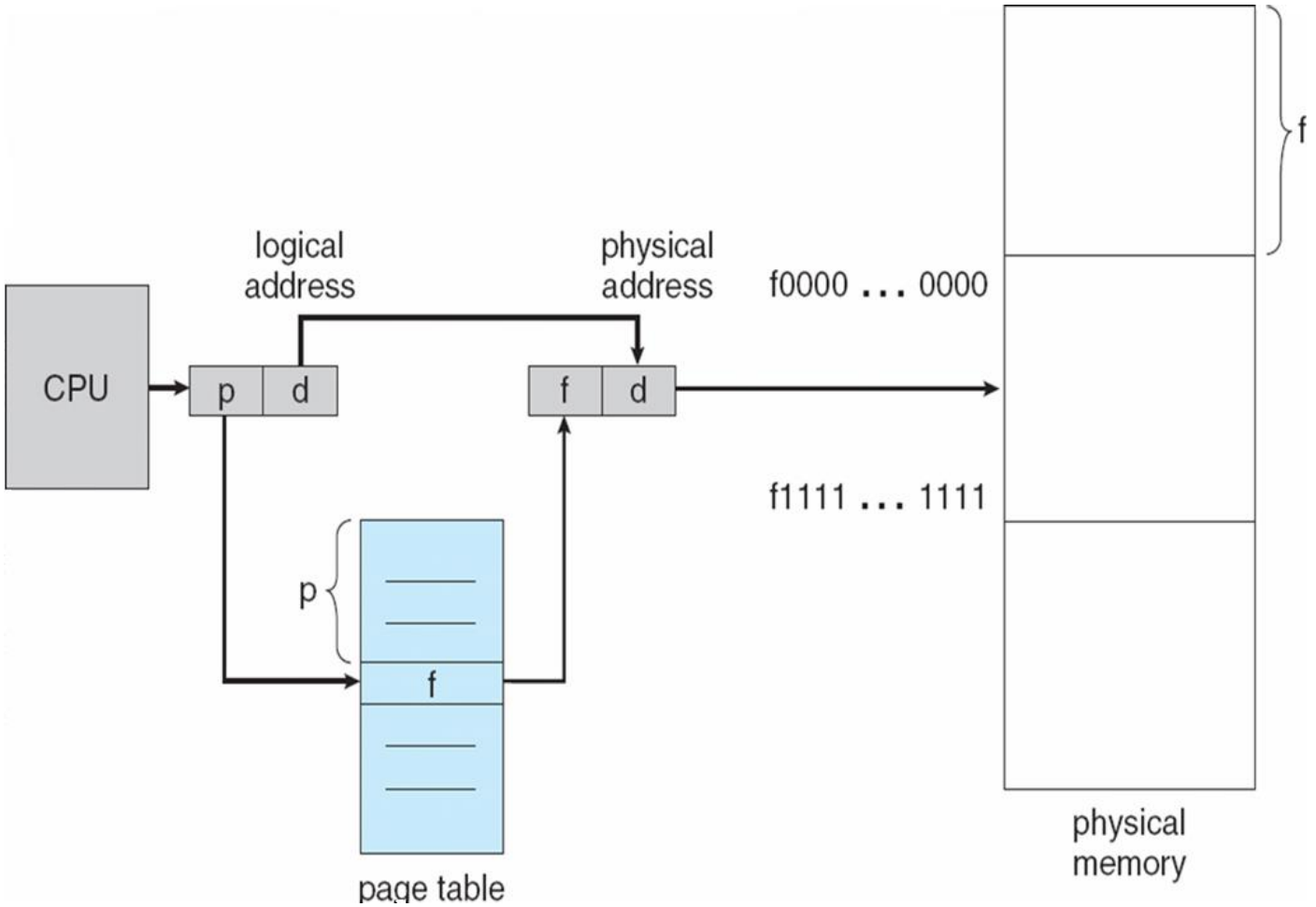
- Address generated by CPU is divided into:
  - **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory
  - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit



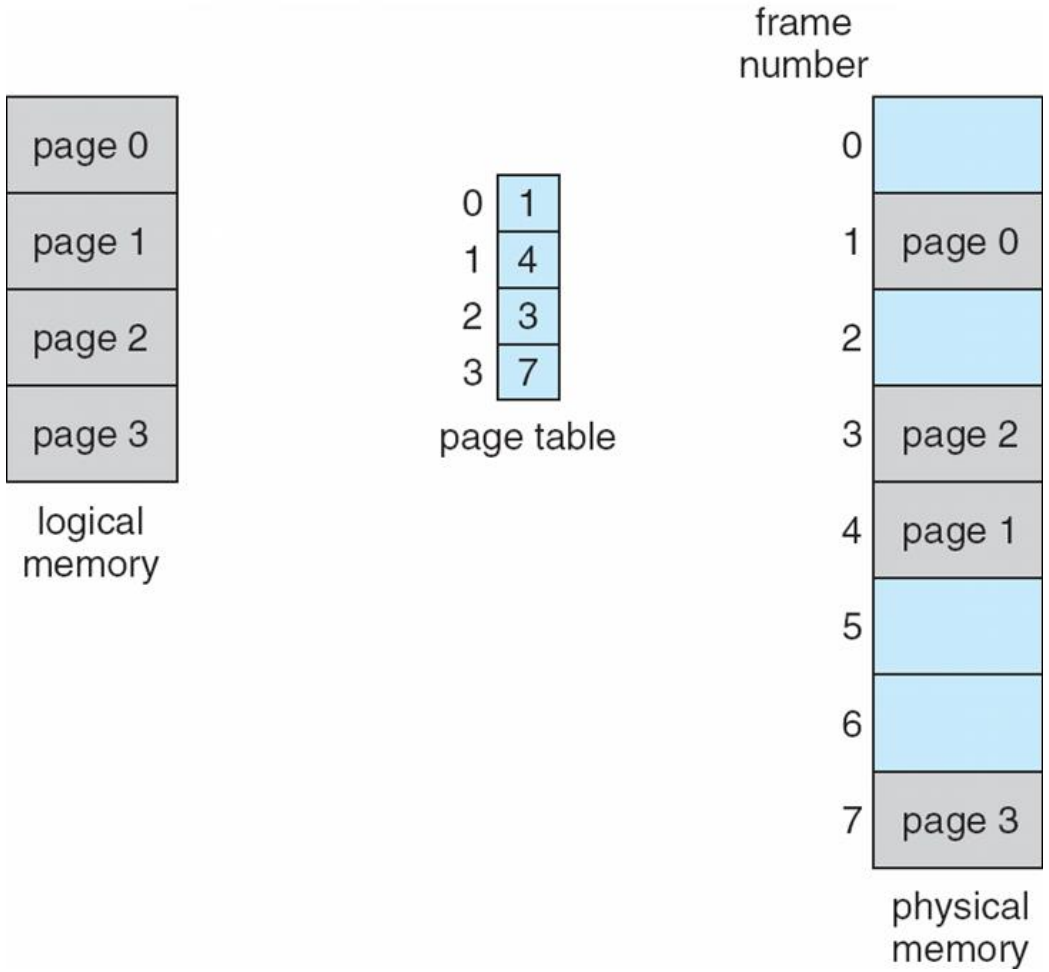
- For given logical address space 2^m *and page size* 2^n



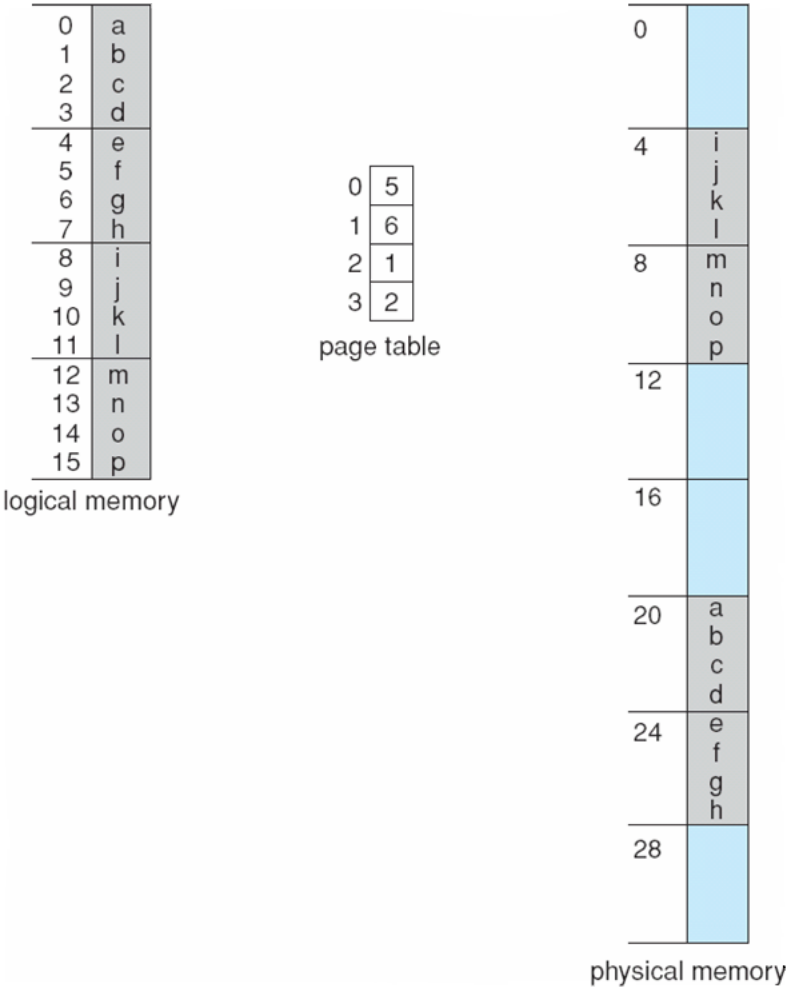
Paging Hardware



Paging Model of Logical and Physical Memory

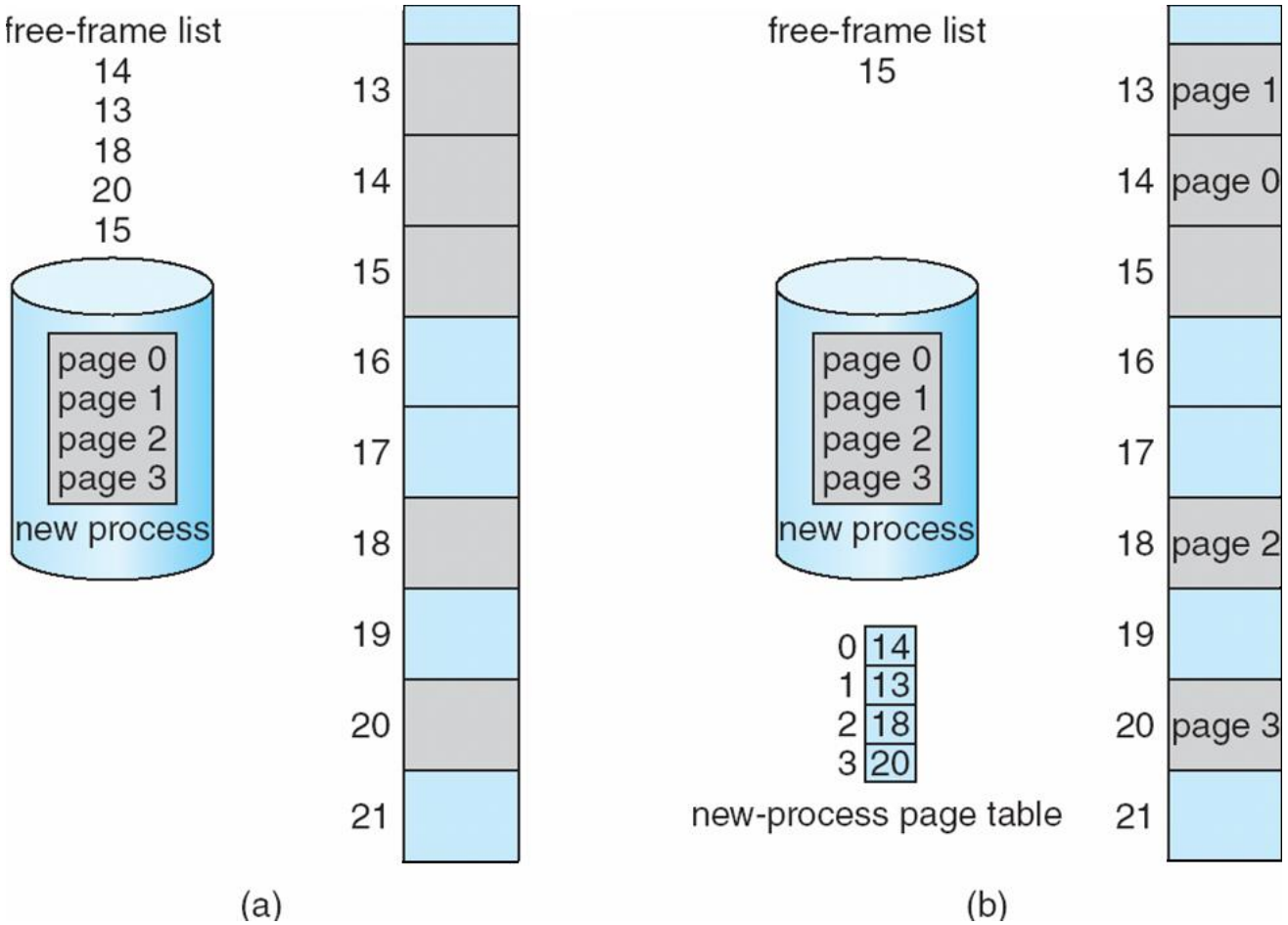


Paging Example



32-byte memory and 4-byte pages

Free Frames



Before allocation

After allocation

Non-contiguous memory allocation:

Paging:

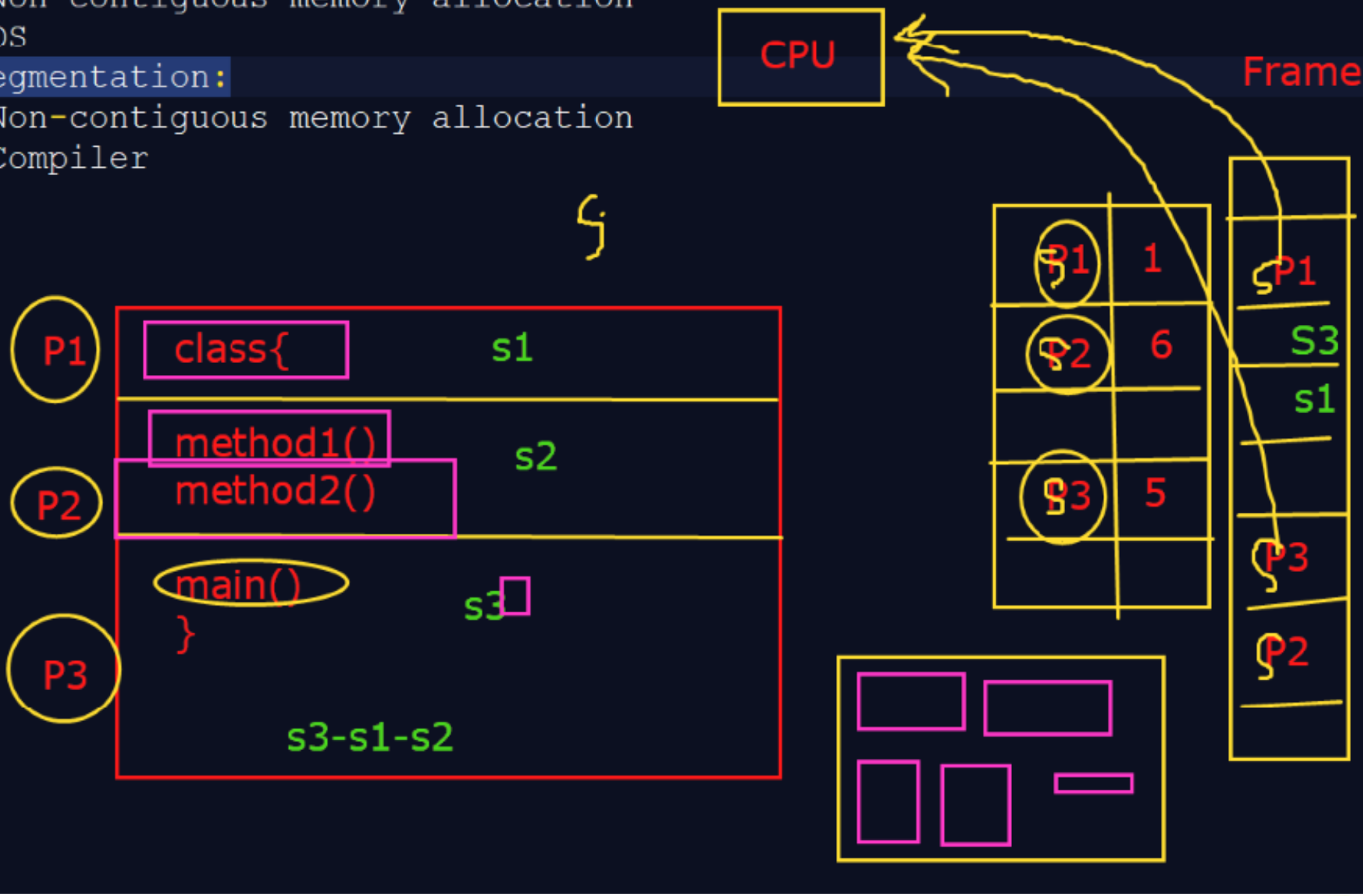
-Non-contiguous memory allocation

-OS

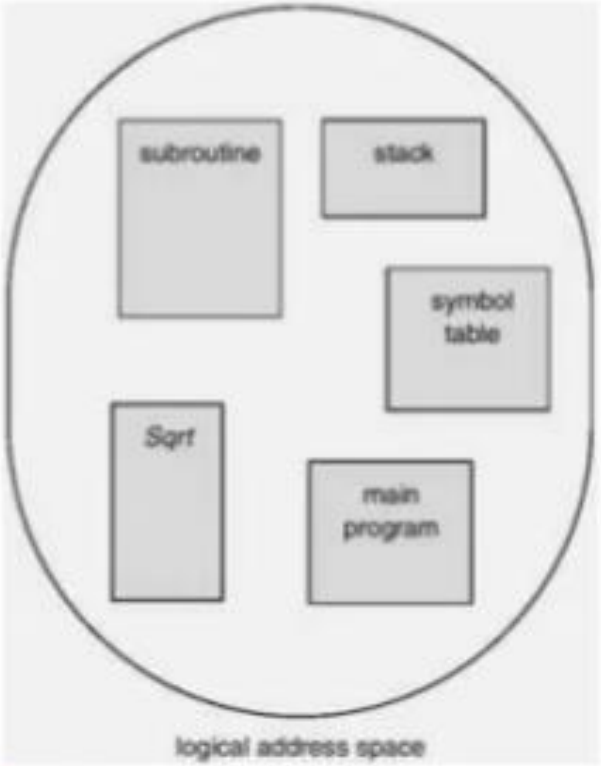
Segmentation:

-Non-contiguous memory allocation

-Compiler

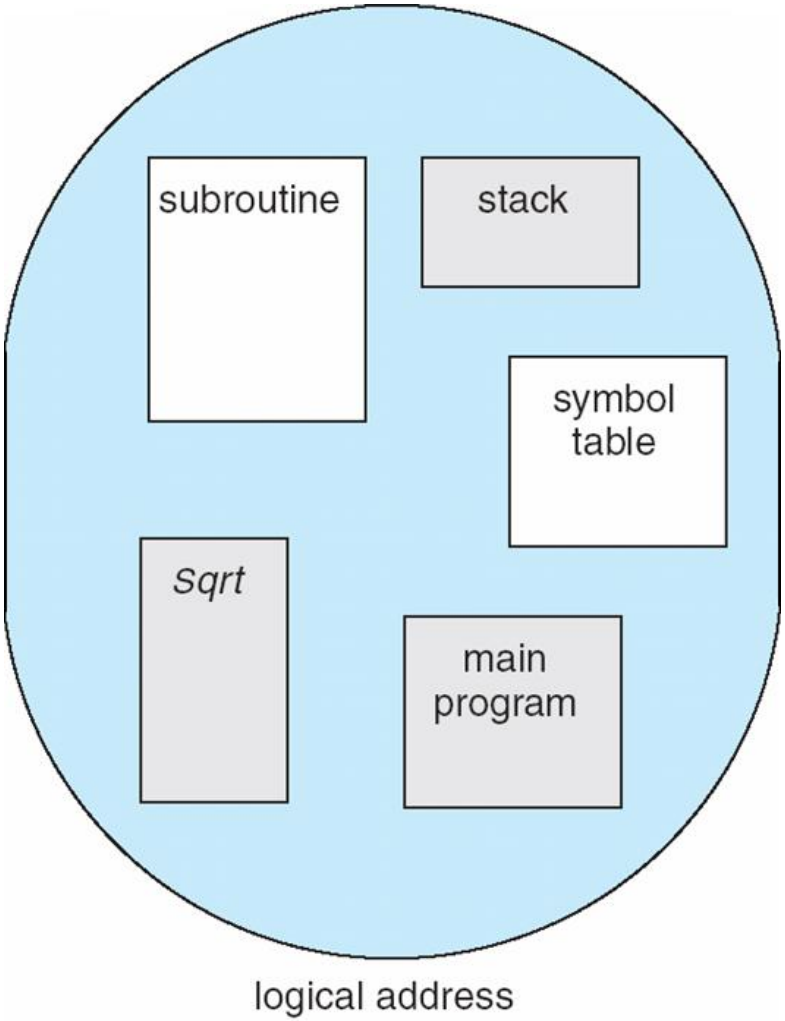


Segmentation

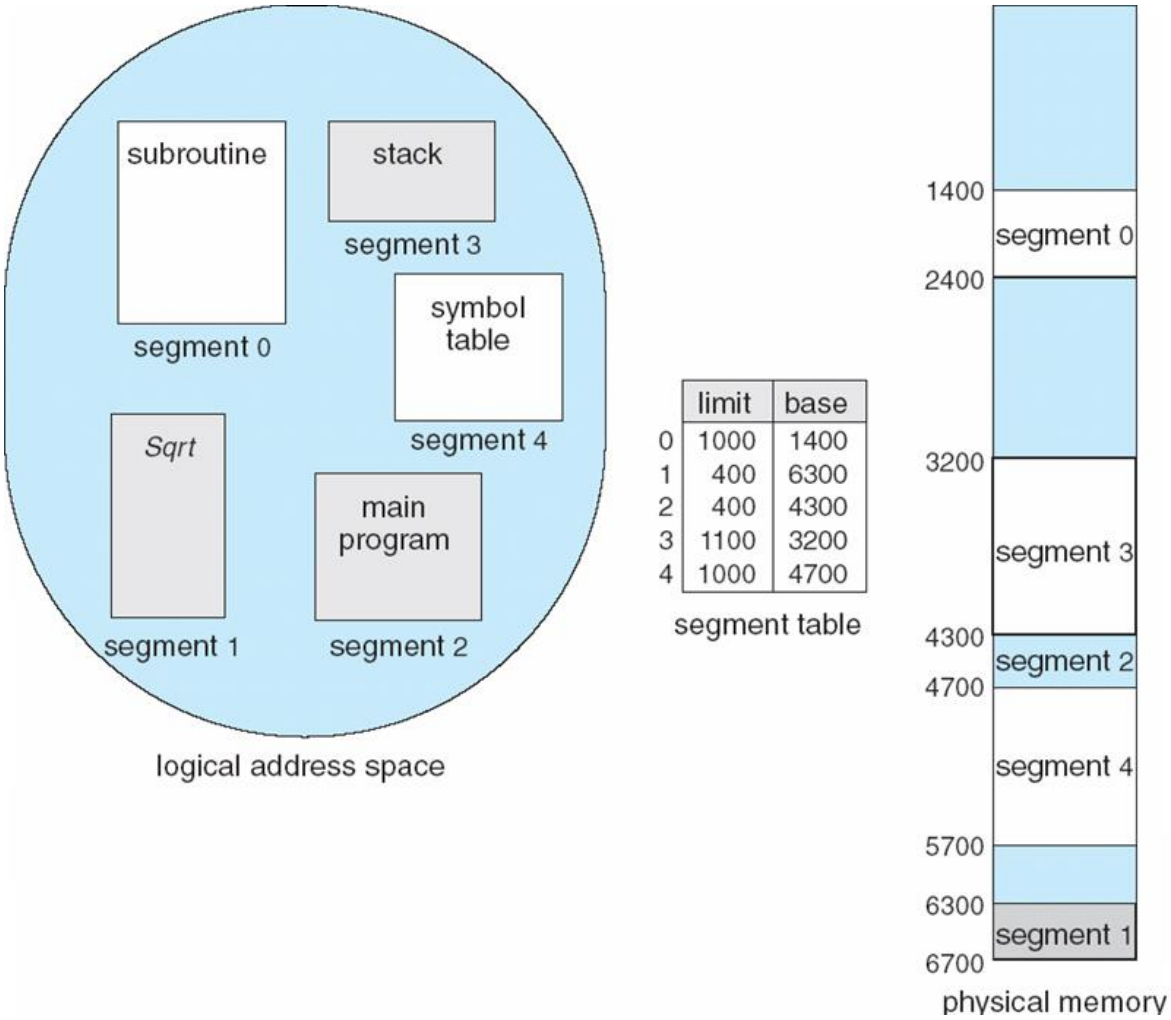


- User View of logical memory
  - Linear array of bytes
    - Reflected by the 'Paging' memory scheme
  - A collection of variable-sized entities
    - User thinks in terms of "subroutines", "stack", "symbol table", "main program" which are somehow located somewhere in memory.]
- Segmentation supports this user view. The logical address space is a collection of segments.

User's View of a Program

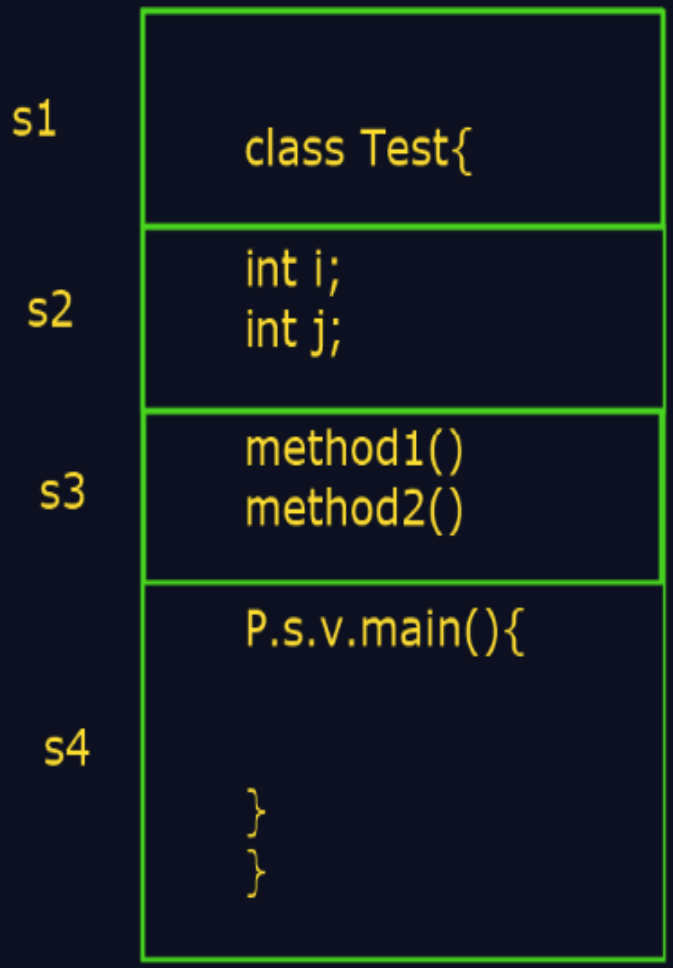


Example of Segmentation



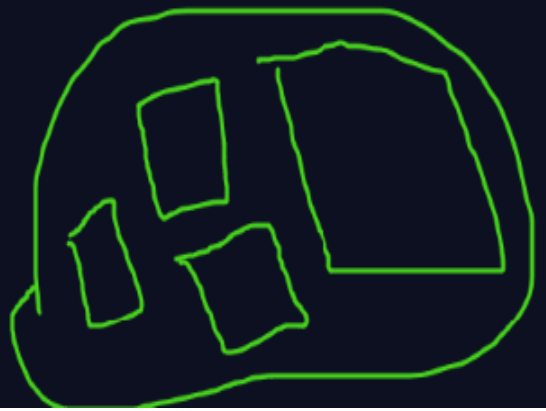
Non-Contiguous allocation:

1. Paging
2. Segmentation



Table

s1	1
s2	6
s3	9
s4	3



M/M

Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:

    main program

    procedure

    function

    method

    object

    local variables, global variables

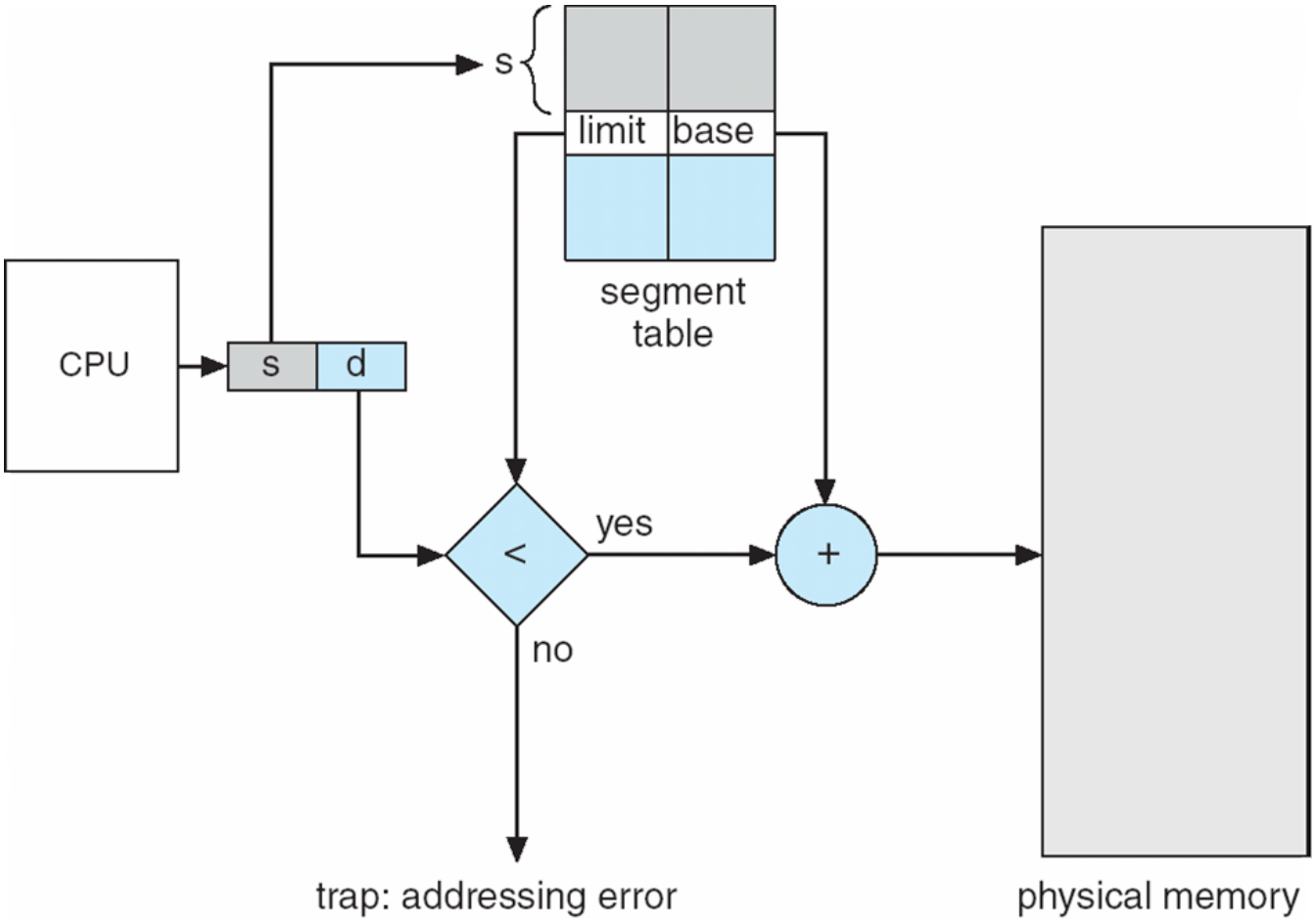
    common block

    stack

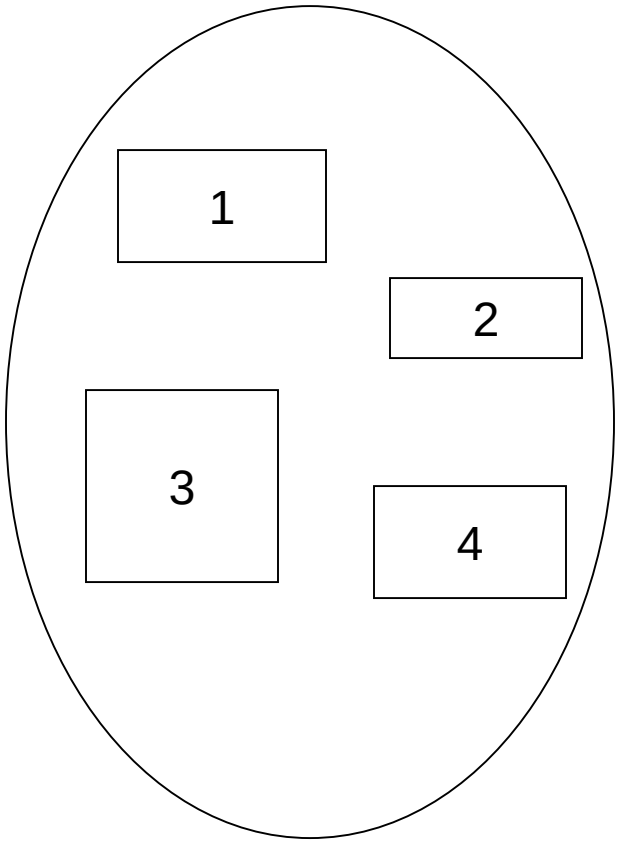
    symbol table

    arrays

Segmentation Hardware



Logical View of Segmentation



user space

physical memory space

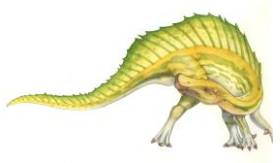
Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**



End of Chapter 8