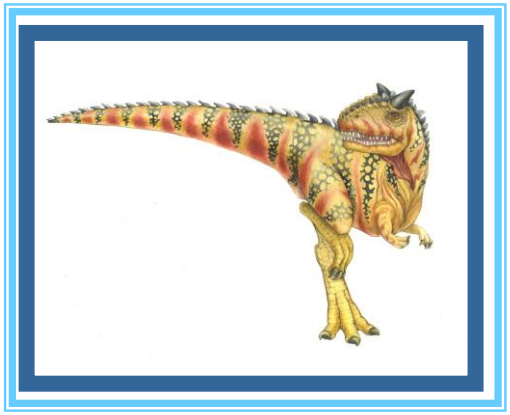
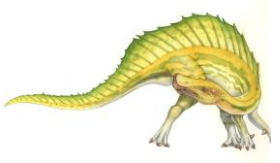
Memory Management
Day5: Sep 2022

Kiran Waghmare



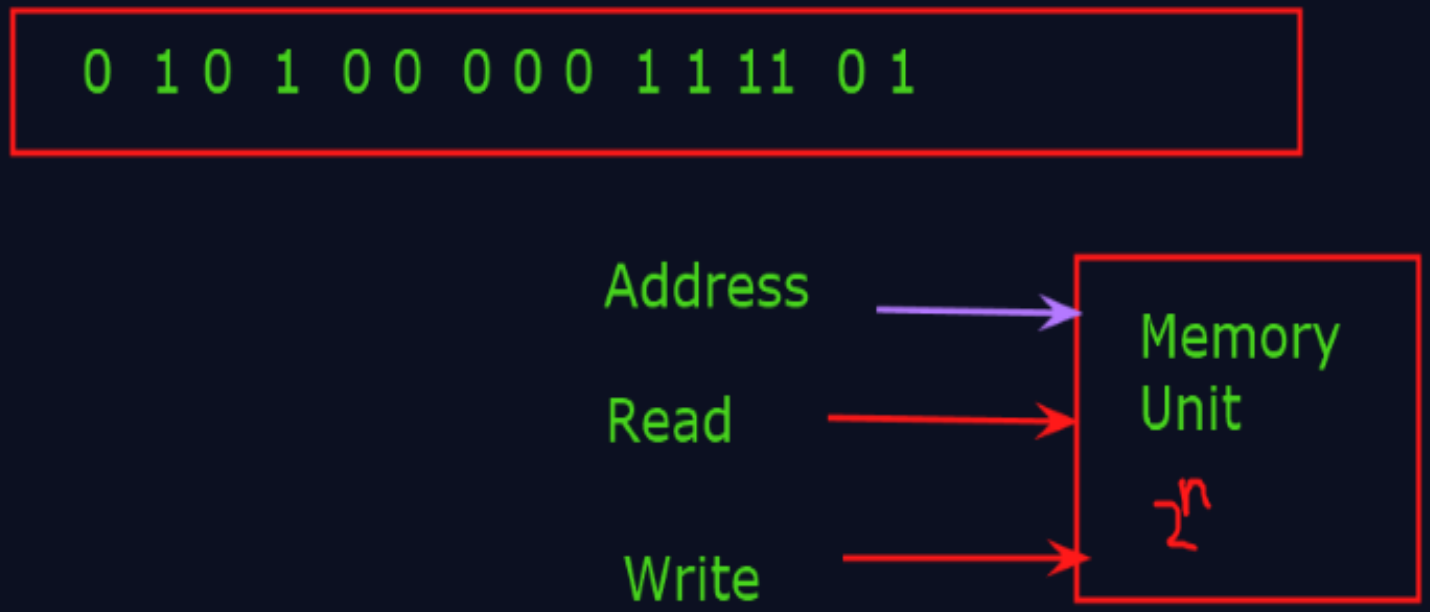
- Memory Management
  - continuous & dynamic
  - best fit, worst fit, first fitexternal internal fragmentation
  - segmentation
  - Paging
  - Segmentation



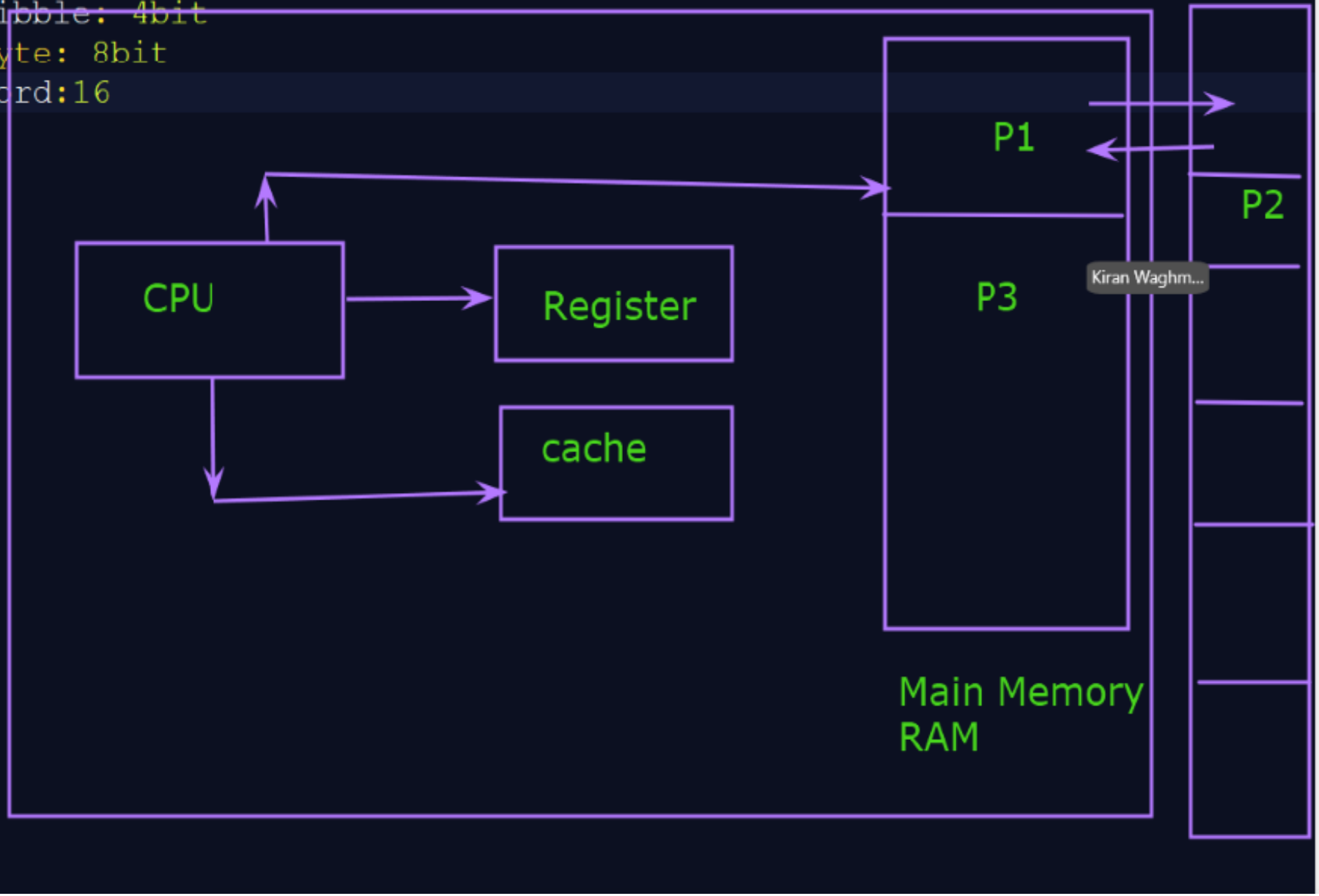
Topics:

Memory Management

-Memory: collection of some amount of data represented in the binary format.

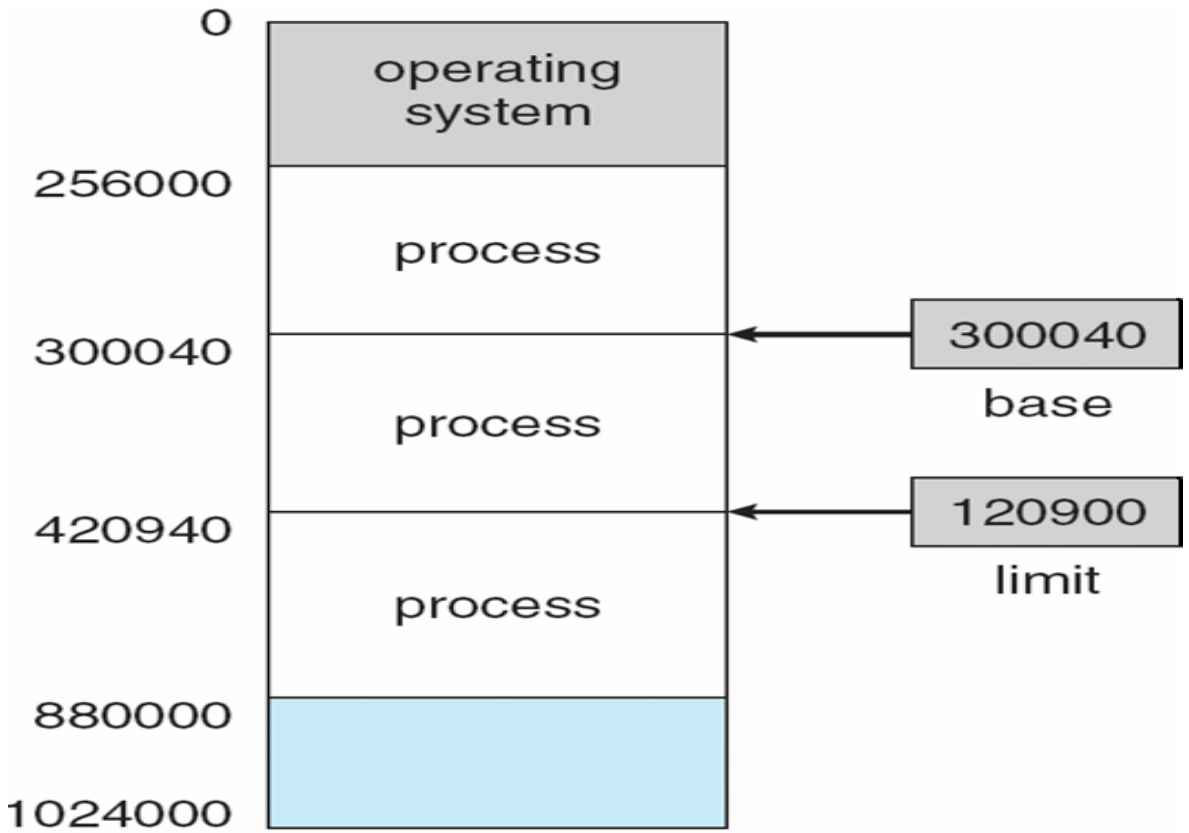


-Memory: collection of so
represented in the binary format.
-Bit: 1bit
-Nibble: 4bit
-byte: 8bit
-word:16



Base and Limit Registers

- A pair of **base** and **limit** registers define the logical address space



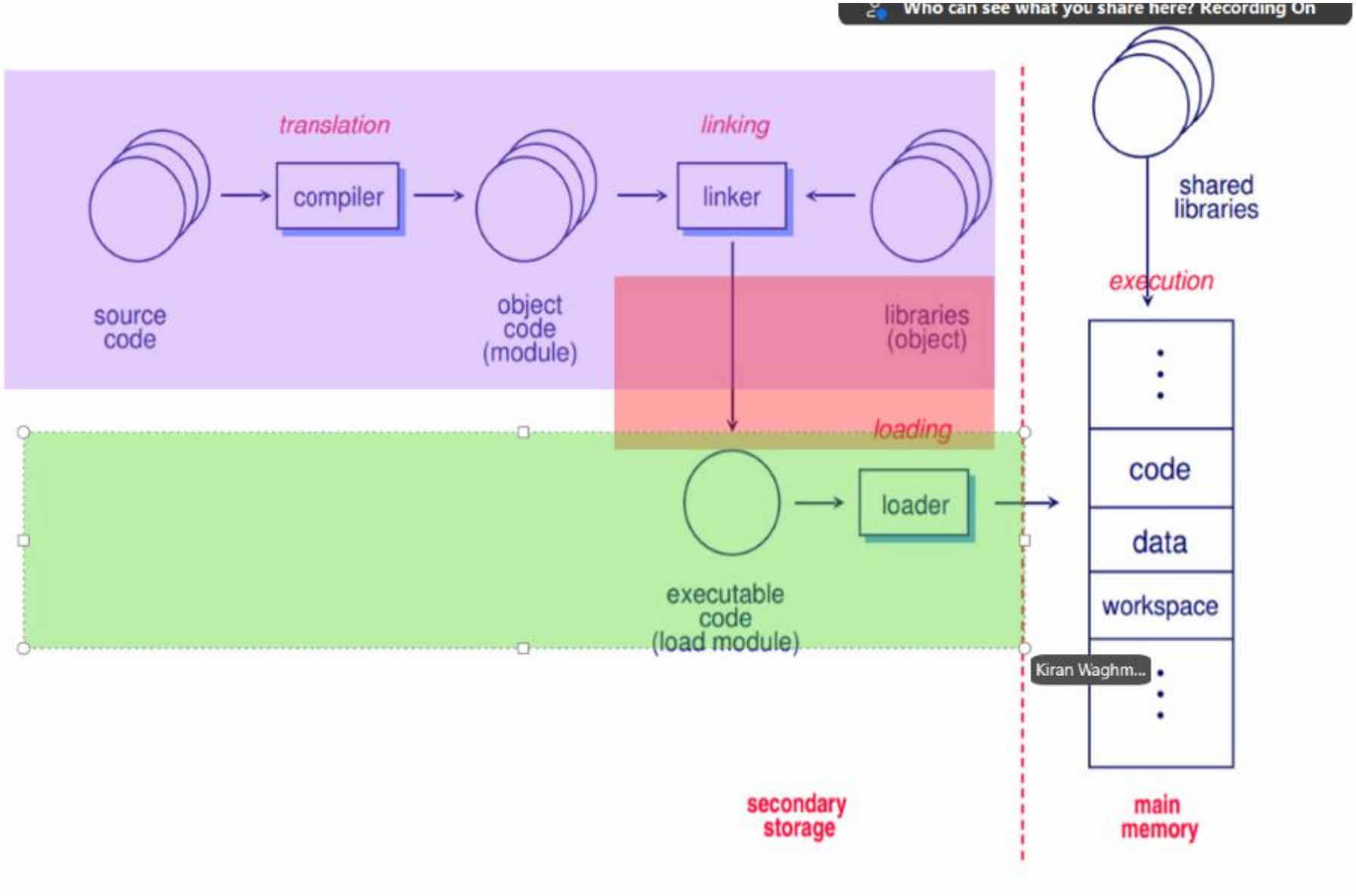
Need for Memory Management in OS

- 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:

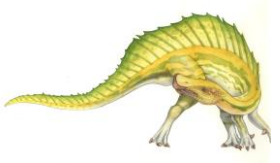
- **Allocate and de-allocate memory** before and after process execution.
- To **keep track of used memory** space by processes.
- To **minimize fragmentation** issues.
- To **proper utilization** of main memory.
- To **maintain data integrity** while executing of process.

From *source* to *executable* code

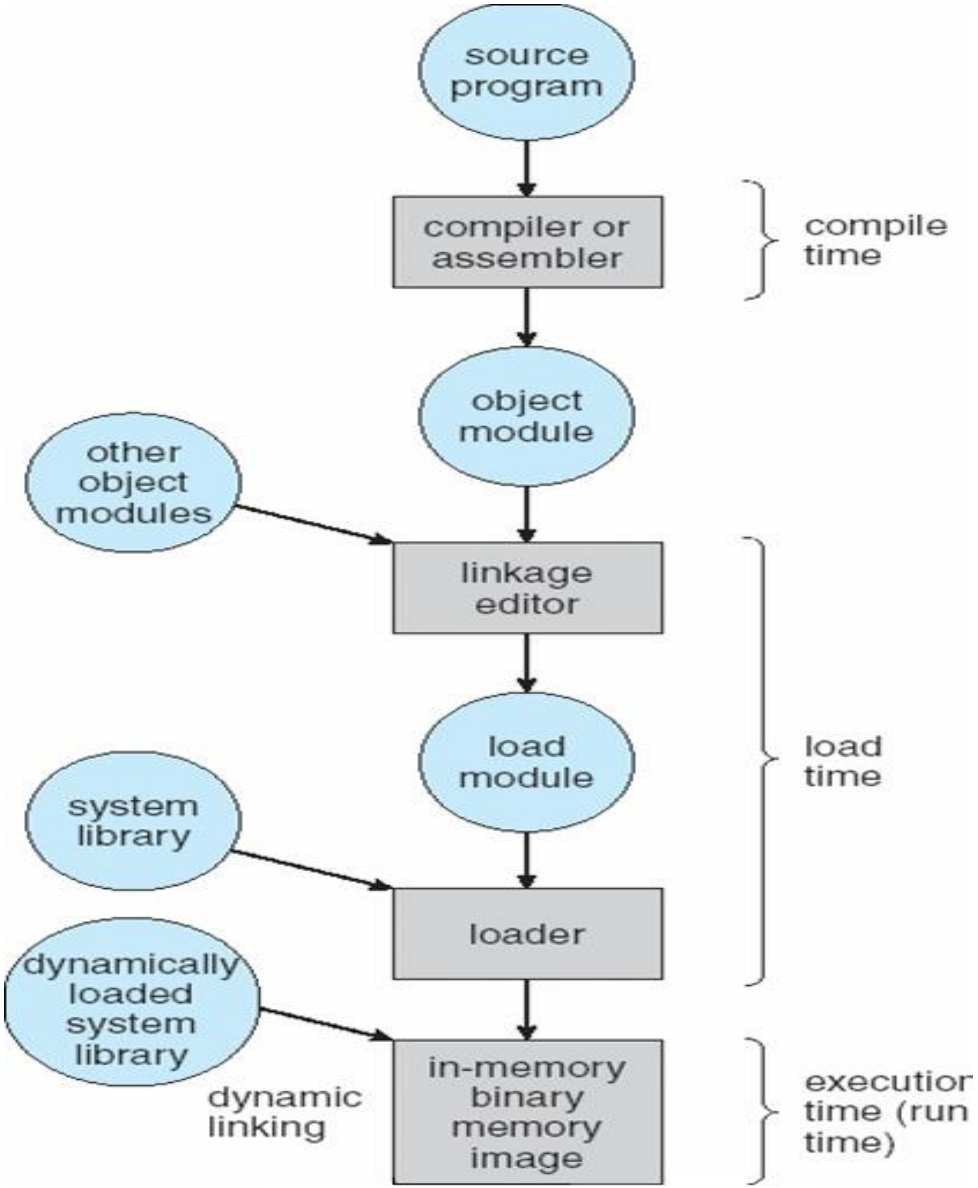


Binding of Instructions and Data to Memory

- 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)



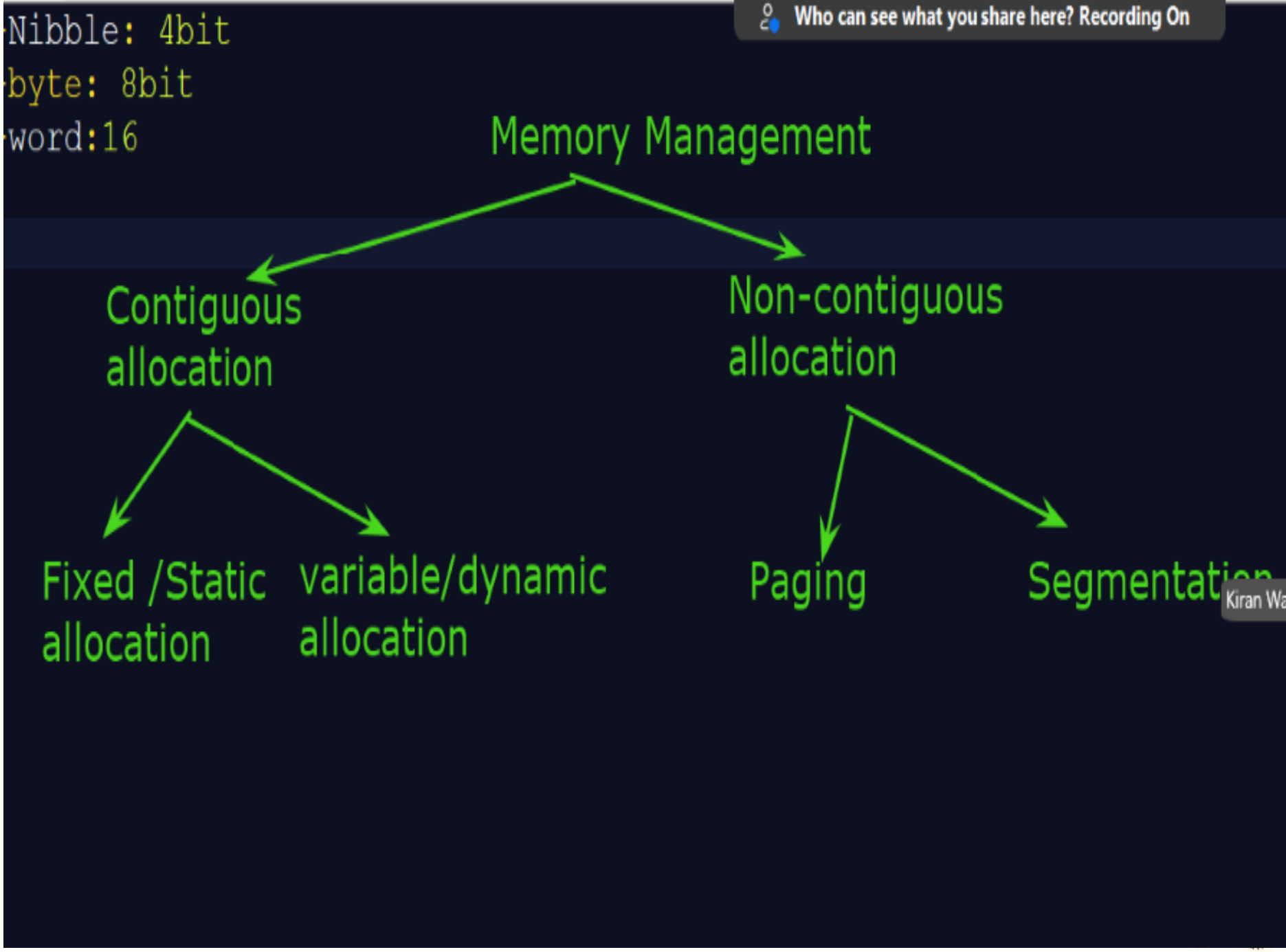
Multistep Processing of a User Program



Nibble: 4bit

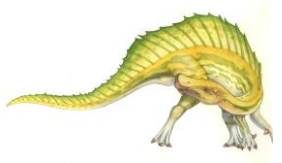
byte: 8bit

word:16



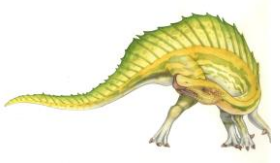
Logical and Physical Address Space:

- **Logical Address space:** An address generated by the CPU is known as “Logical Address”.
- It is also known as a **Virtual address**.
- Logical address space can be **defined as the size of the process**.
- A logical address can be changed.
- **Physical Address space:** An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a “Physical Address”.
- A Physical address is also known as a **Real address.**
- The set of all physical addresses corresponding to these logical addresses is known as **Physical address space**.
- A physical address is **computed by MMU.**
- The run-time mapping from virtual to physical addresses is done by a **hardware device Memory Management Unit(MMU).**
- The physical address **always remains constant**.



Logical vs. Physical Address Space

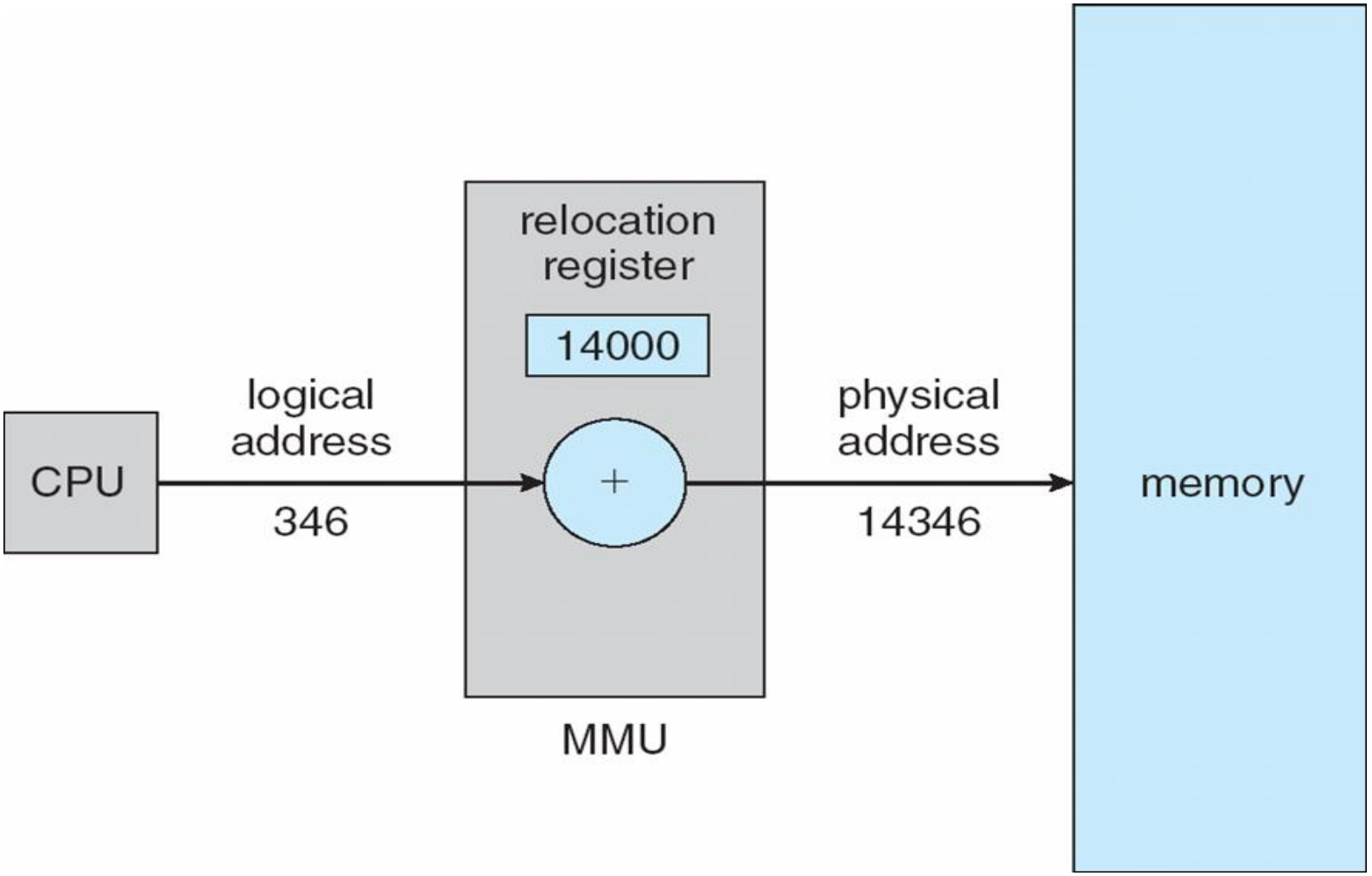
- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical 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



Mute Start Video Security Participants Chat

You are screen

Mouse Select Text Draw Stamp Spotlight Eraser

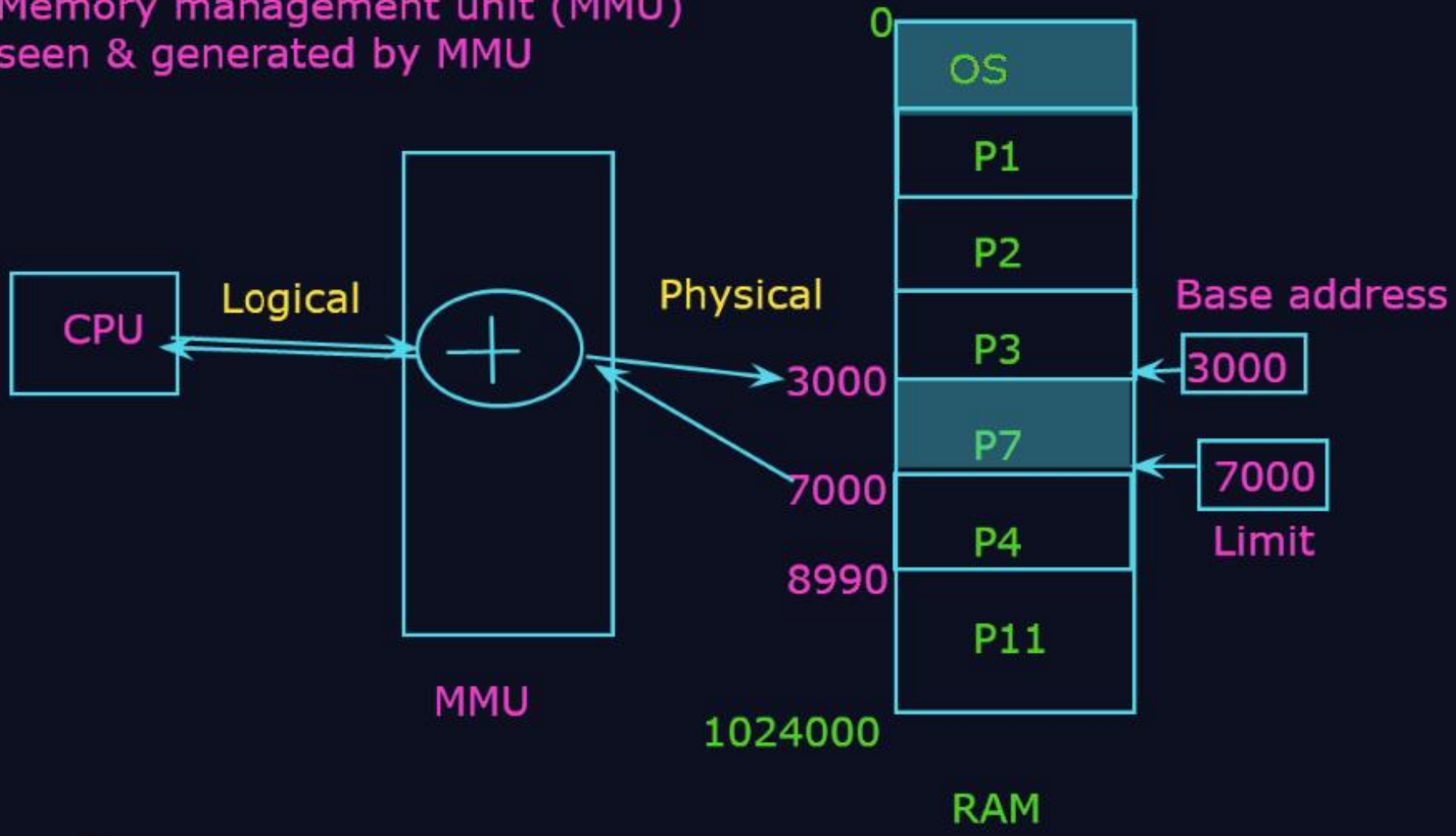
Who can see what you share here?

Sep22.java new 2 Shellprogramming.sh GMM model.txt SimpleKmeans

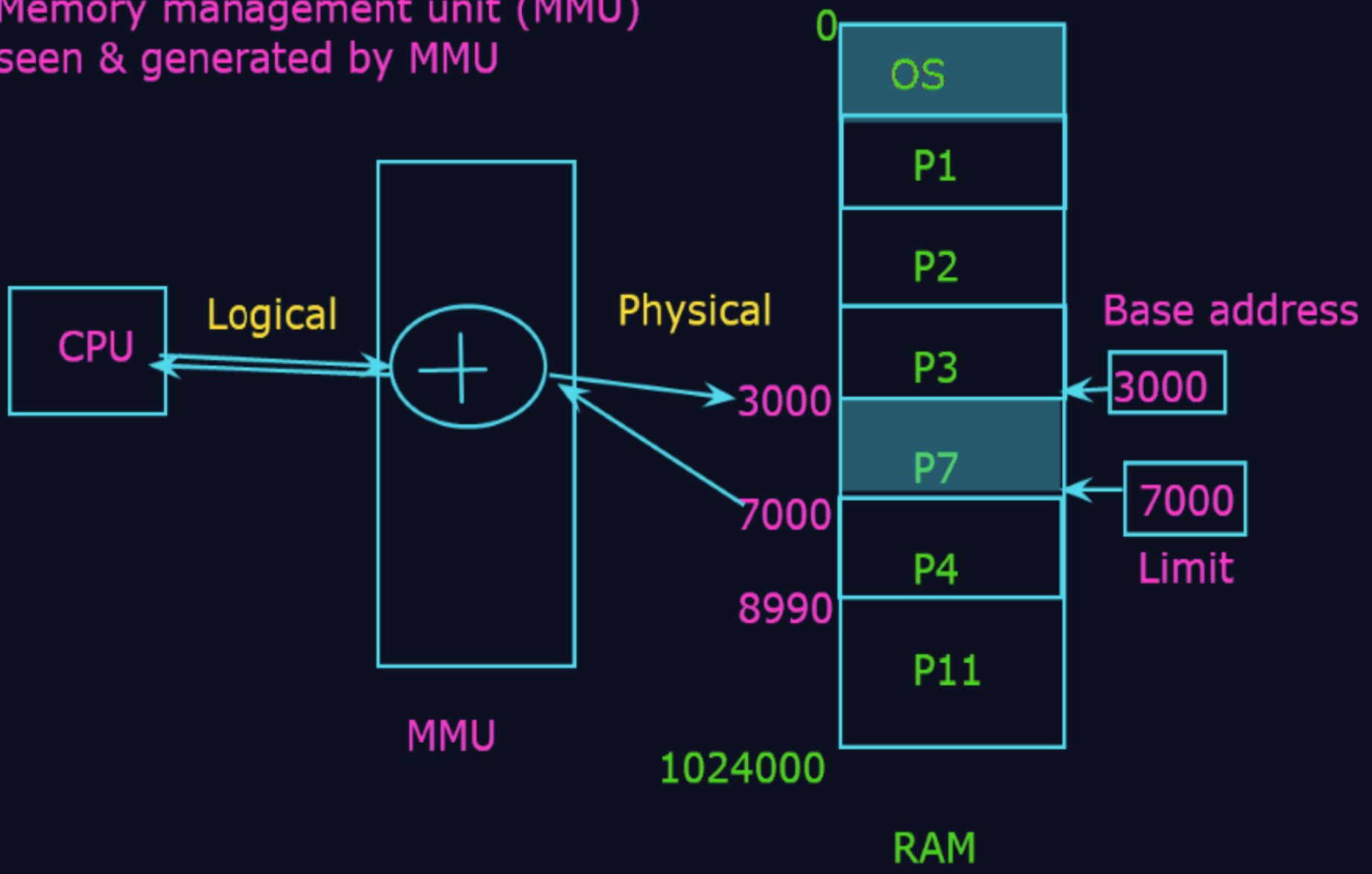
Logical Address:
-generated by CPU

Non-contiguous memory allocation:
-virtual address

Physical Address:
-Memory management unit (MMU)
-seen & generated by MMU



Physical Address:
-Memory management unit (MMU)
-seen & generated by MMU



Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required implemented through program design

Static and Dynamic Loading:

- To load a process into the main memory is done by a loader. There are two different types of loading :
- **Static loading**:- In static loading load the entire program into a fixed address. It requires more memory space.
- **Dynamic loading**:- The entire program and all data of a process must be in physical memory for the process to execute.
- So, the size of a process **is limited to the size of physical memory.**
- To gain proper memory utilization, dynamic loading is used.
- In dynamic loading, a **routine is not loaded until it is called**.
- All routines **are residing on disk** in a relocatable load format.
- One of the advantages of dynamic loading is that **unused routine is never loaded.**
- This loading is **useful when a large amount of code is needed to handle** it efficiently.

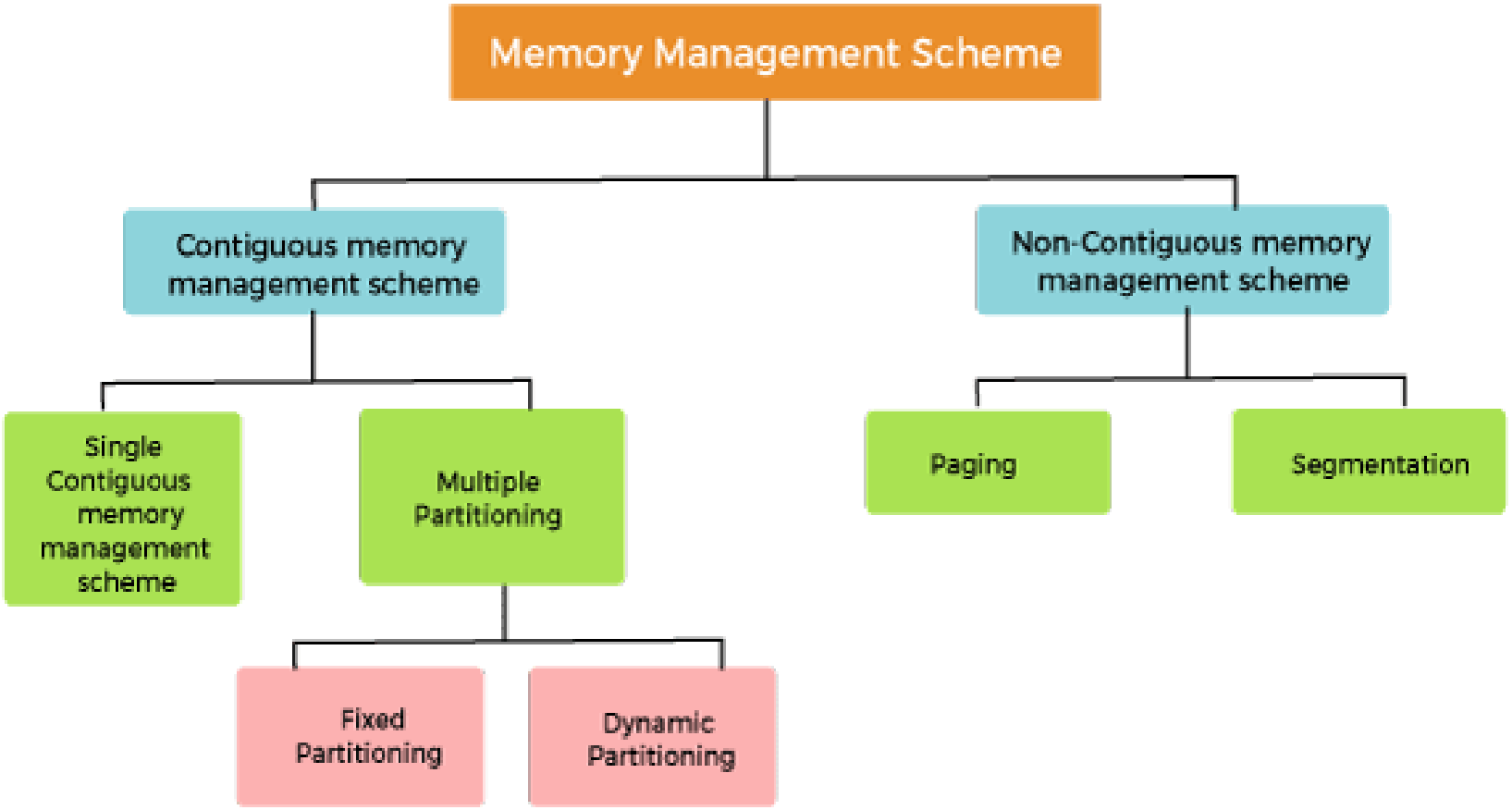
Dynamic Linking

- Linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system needed to check if routine is in processes' memory address
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**

Static and Dynamic linking:

- To perform a linking task a **linker i**s used.
- A linker is a program that takes **one or more object files generated by a compiler and combines them into a single executable** file.

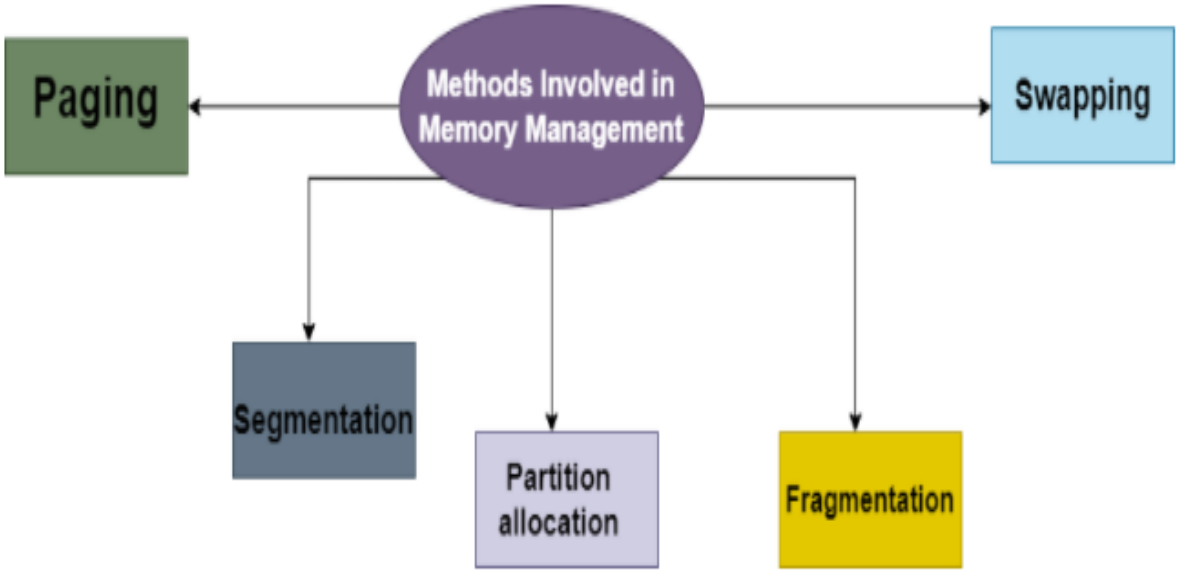
- Static linking: In static linking, the linker combines all necessary program modules into a single executable program. So there **is no runtime dependency.** Some operating systems support only static linking, in which system language libraries are treated like any other object module.
- Dynamic linking: The basic concept of dynamic linking is similar to dynamic loading.
- In dynamic linking, “Stub” is included for each appropriate library routine reference. **A stub is a small piece of code**. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.

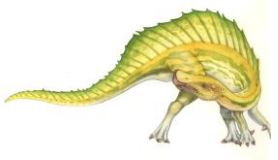


Classification of memory management schemes

Methods Involved in Memory Management

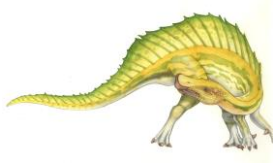
There are various methods and with their help Memory Management can be done intelligently by the Operating System:



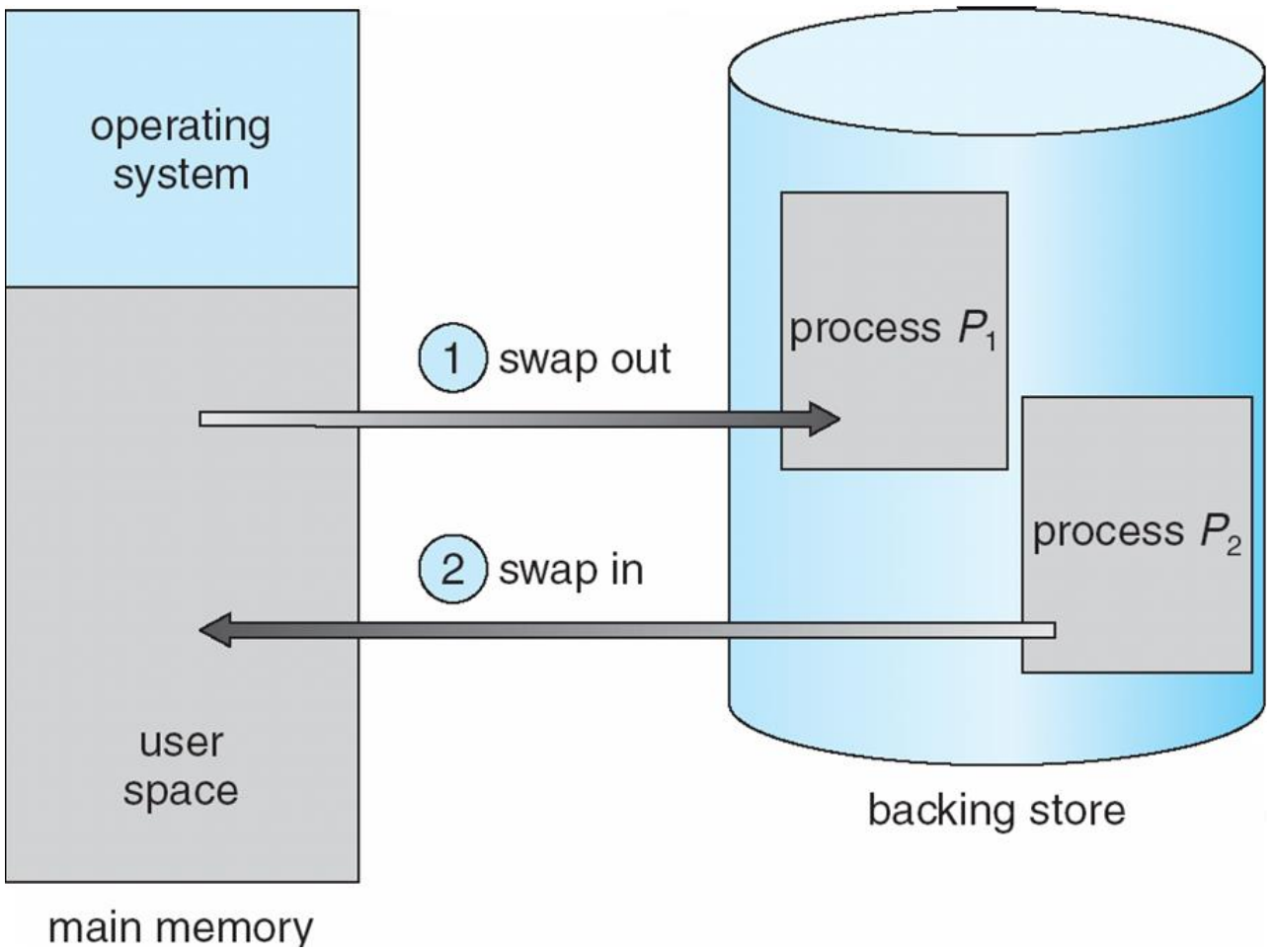


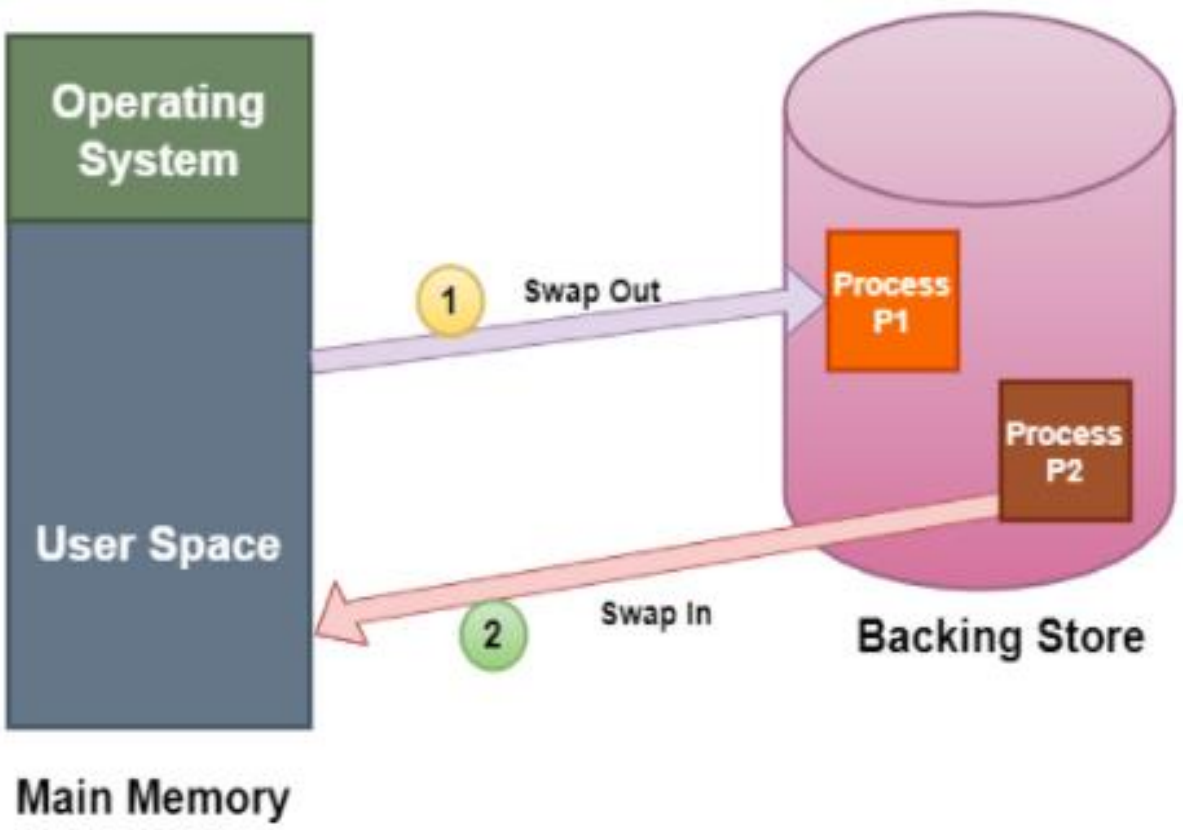
Swapping

- A process **can be swapped temporarily** out of memory to a backing store, and then brought back into memory for continued execution
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk



Schematic View of Swapping



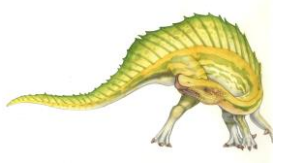

Operating System
User Space
Main Memory
1
Swap Out
2
Swap In
Process P1
Process P2
Backing Store

Swapping :

- When a process is executed it must have resided in memory.
- Swapping is a process of swap a process temporarily into a secondary memory from the main memory, which is fast as compared to secondary memory.
- A swapping allows more processes to be run and can be fit into memory at one time.
- The main part of swapping is transferred time and the total time directly proportional to the amount of memory swapped.
- Swapping is also known as **roll-out, roll in**, because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process.
- After **finishing higher priority work**, the lower priority process swapped back in memory and continued to the execution process.

Swap In and Swap Out in OS

- The procedure by which any process gets removed from the hard disk and placed in the main memory or RAM commonly known as Swap In.
- On the other hand, Swap Out is the method of removing a process from the main memory or RAM and then adding it to the Hard Disk.
- **Advantages of Swapping**
- The swapping technique **mainly helps the CPU to manage multiple processes within a single main memory**.
- This technique **helps to create and use virtual memory**.
- With the help of this technique**, the CPU can perform several tasks** simultaneously. Thus, processes need not wait too long before their execution.
- This technique is **economical.**
- This technique can be easily applied **to priority-based scheduling** in order to improve its performance.



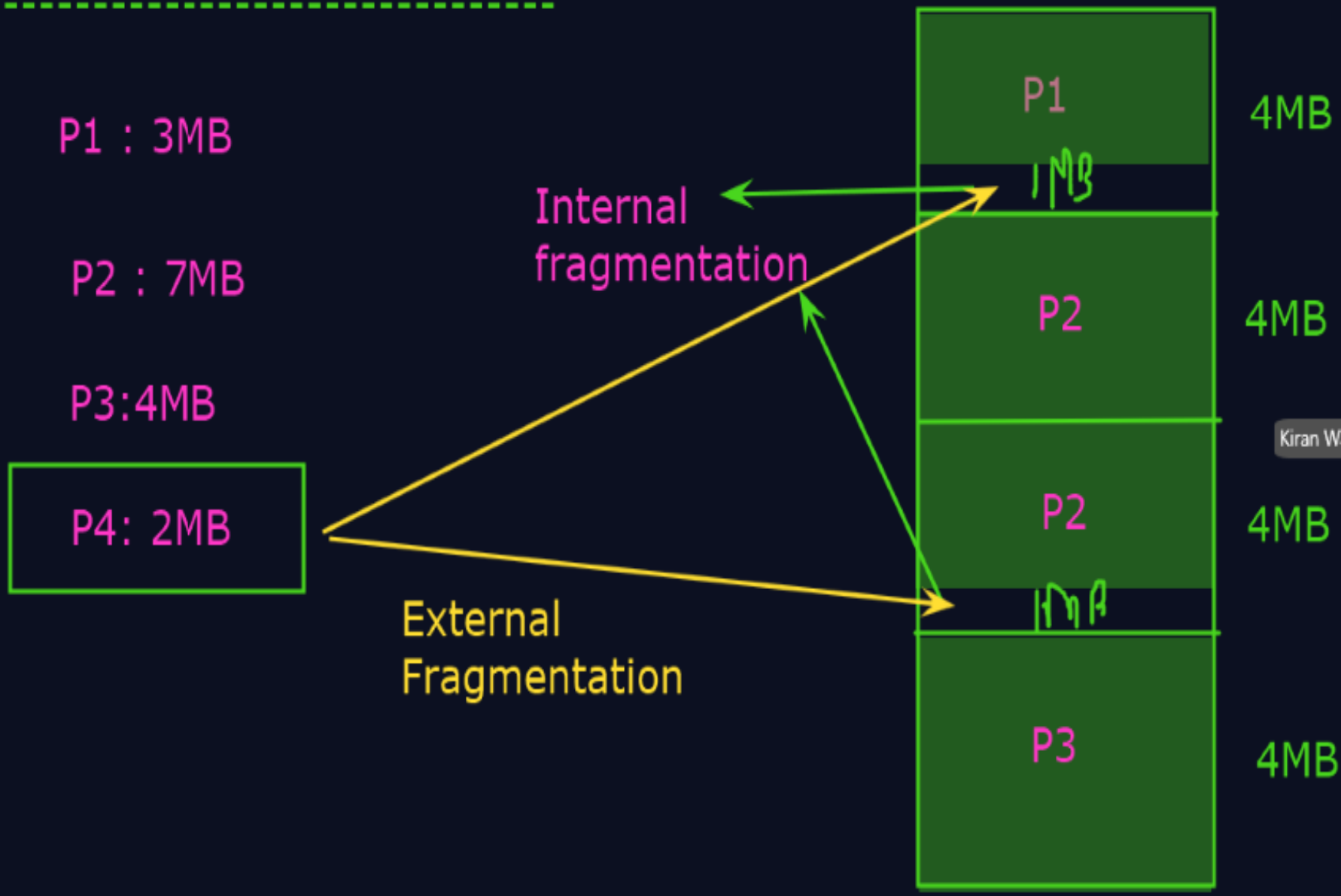
Disadvantages of Swapping

- There may occur inefficiency in the case if a resource or a variable is commonly used by those processes that are participating in the swapping process.
- If the algorithm used for swapping is not good then the overall method can increase the number of page faults and thus decline the overall performance of processing.
- If the computer system loses power at the time of high swapping activity then the user might lose all the information related to the program.

Contiguous Allocation

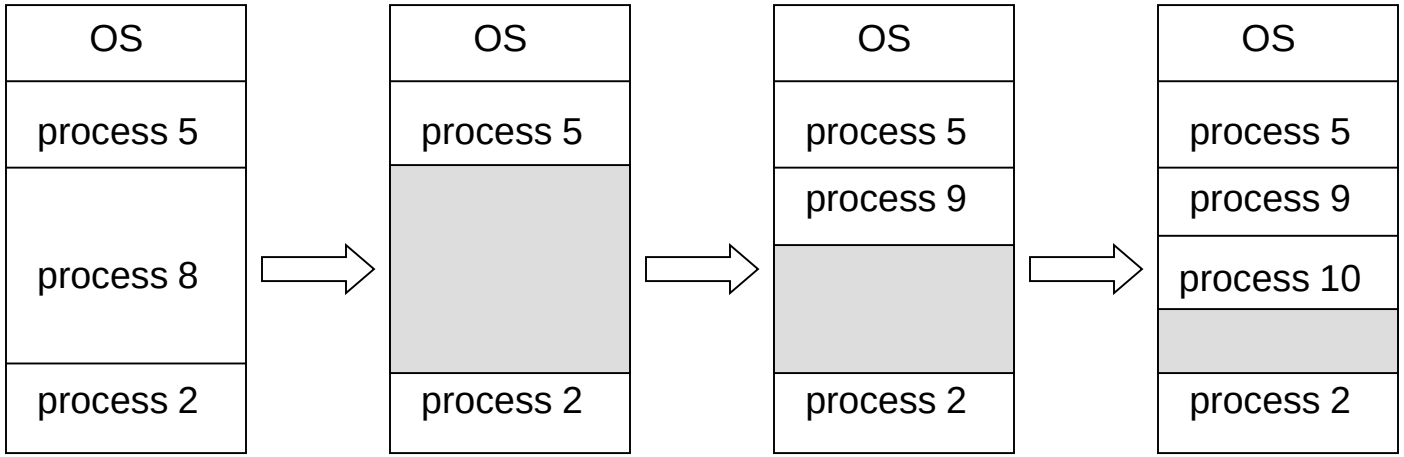
- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register 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*

Contiguous Memory allocation:



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)



Contiguous Memory allocation:

P1 : 3MB

P2 : 7MB

P3:4MB

P4: 2MB

Dynamic allocation

Storage Allocation system:

First-fit:
Best-fit:
worst-fit:
next-fit:

I/O :k :20%
CPU:(1-k) : 80%

CPU : (1-k^2)

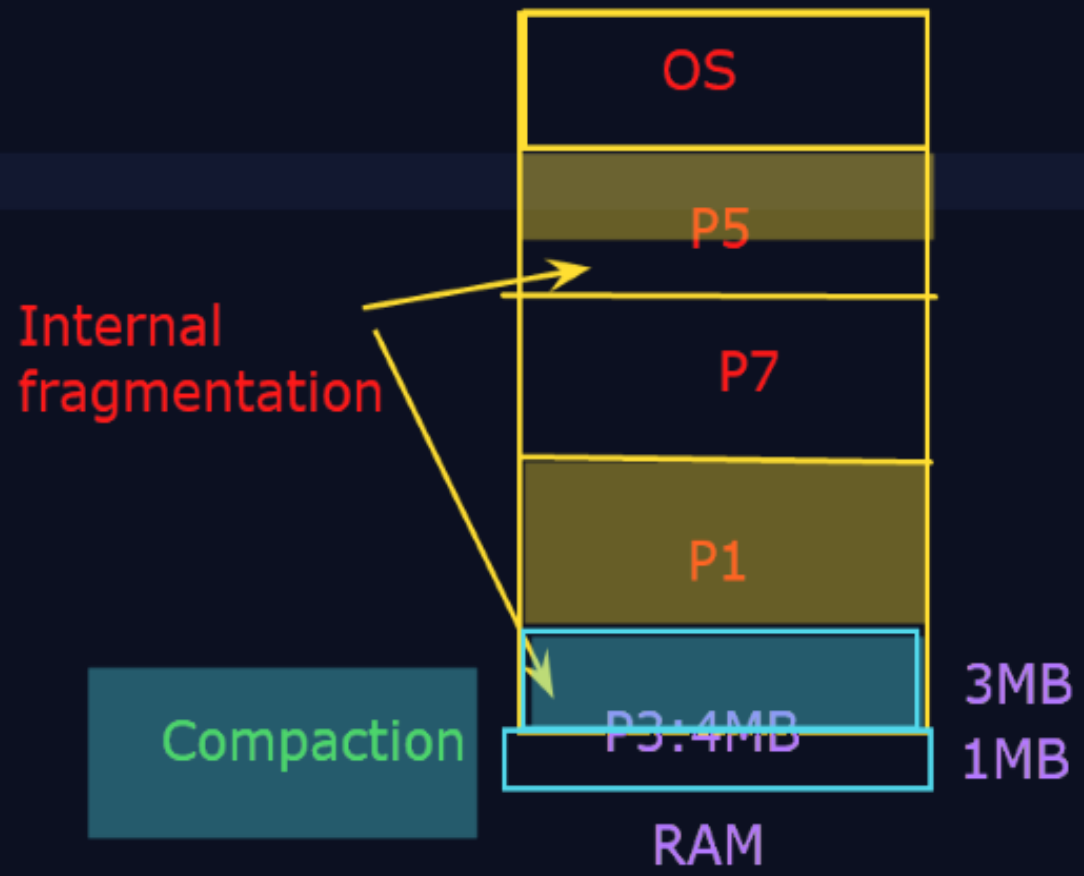
1-0.2=0.8

80%

1-(0.2*0.2)=0.96

Static Partitioning:
-Internal & Extranal fragmentation

Dynamic Partitioning:
Memory Allocation Technique:
-First-fit
-Best-fit
-Worst-fit
-Next-fit

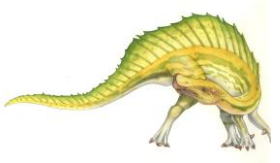


Dynamic Storage-Allocation Problem

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

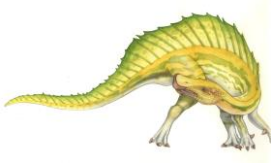
- **First-fit**: Allocate the *first* hole that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the *largest* hole; must also search entire list
  - Produces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



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.

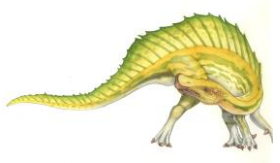
In this technique, the main memory is divided into partitions of equal or different sizes. The operating system always resides in the first partition while the other partitions can be used to store user processes. The memory is assigned to the processes in contiguous way.

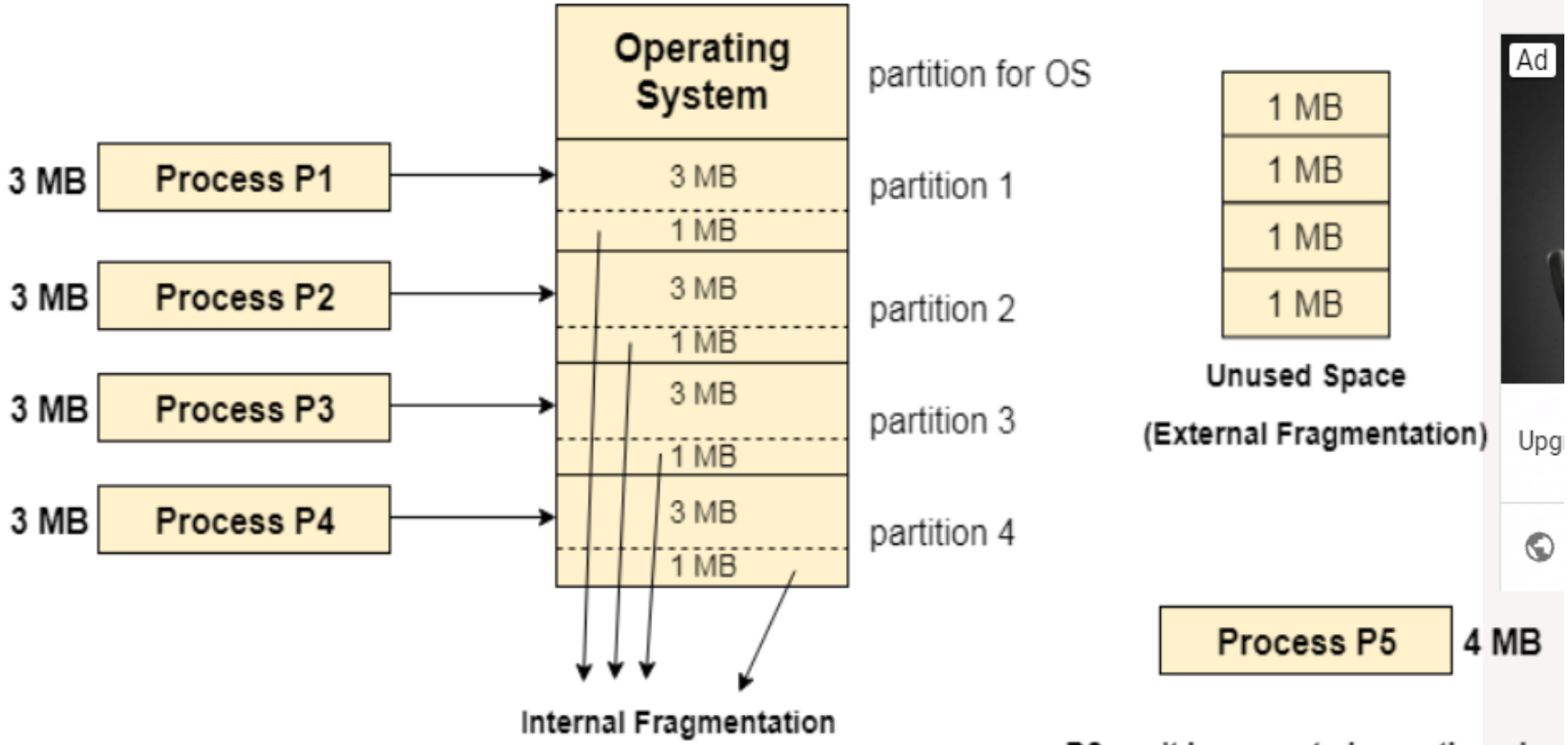
In fixed partitioning,

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.

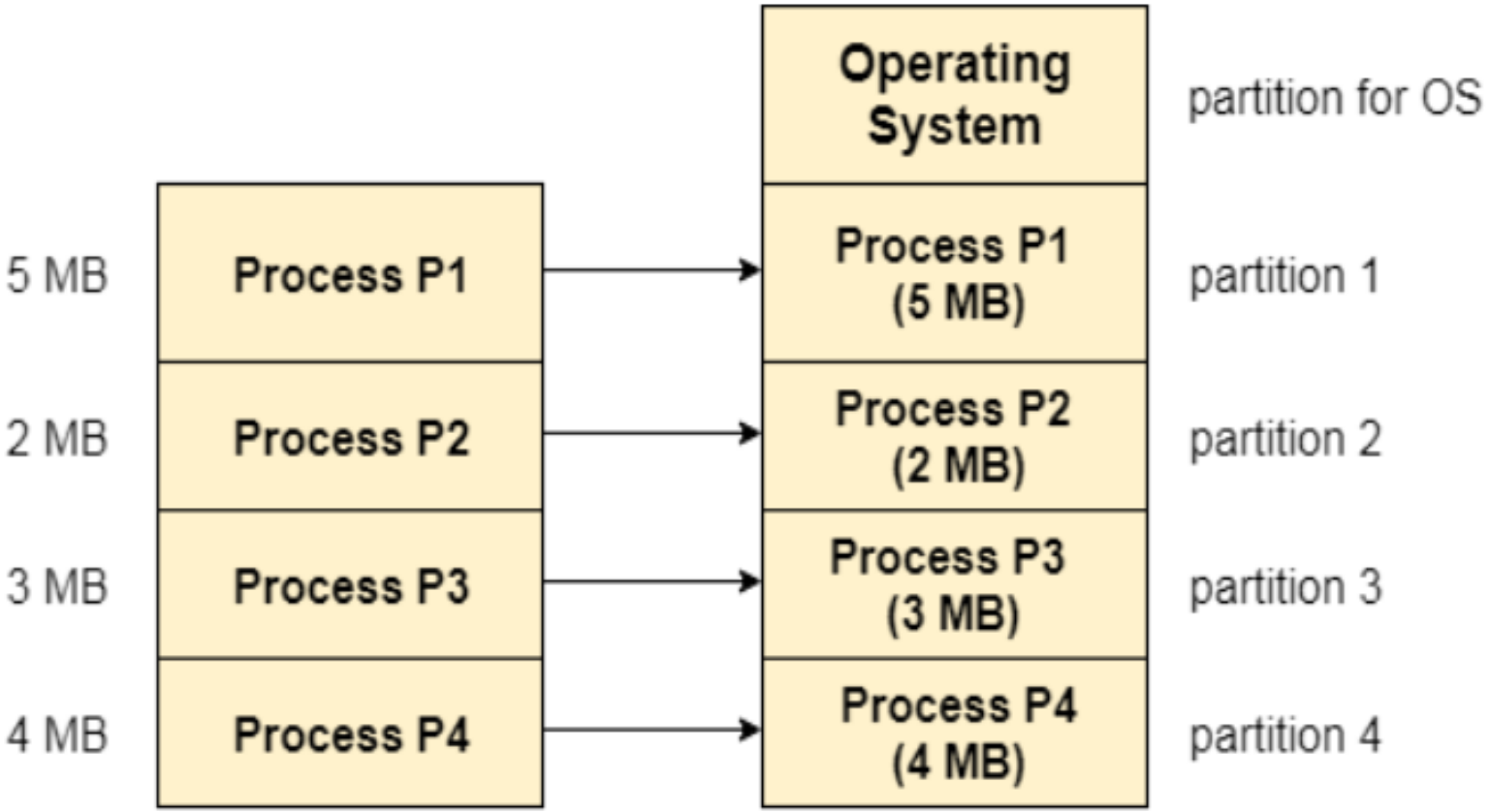
In this technique, the partition size is not declared initially.

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.

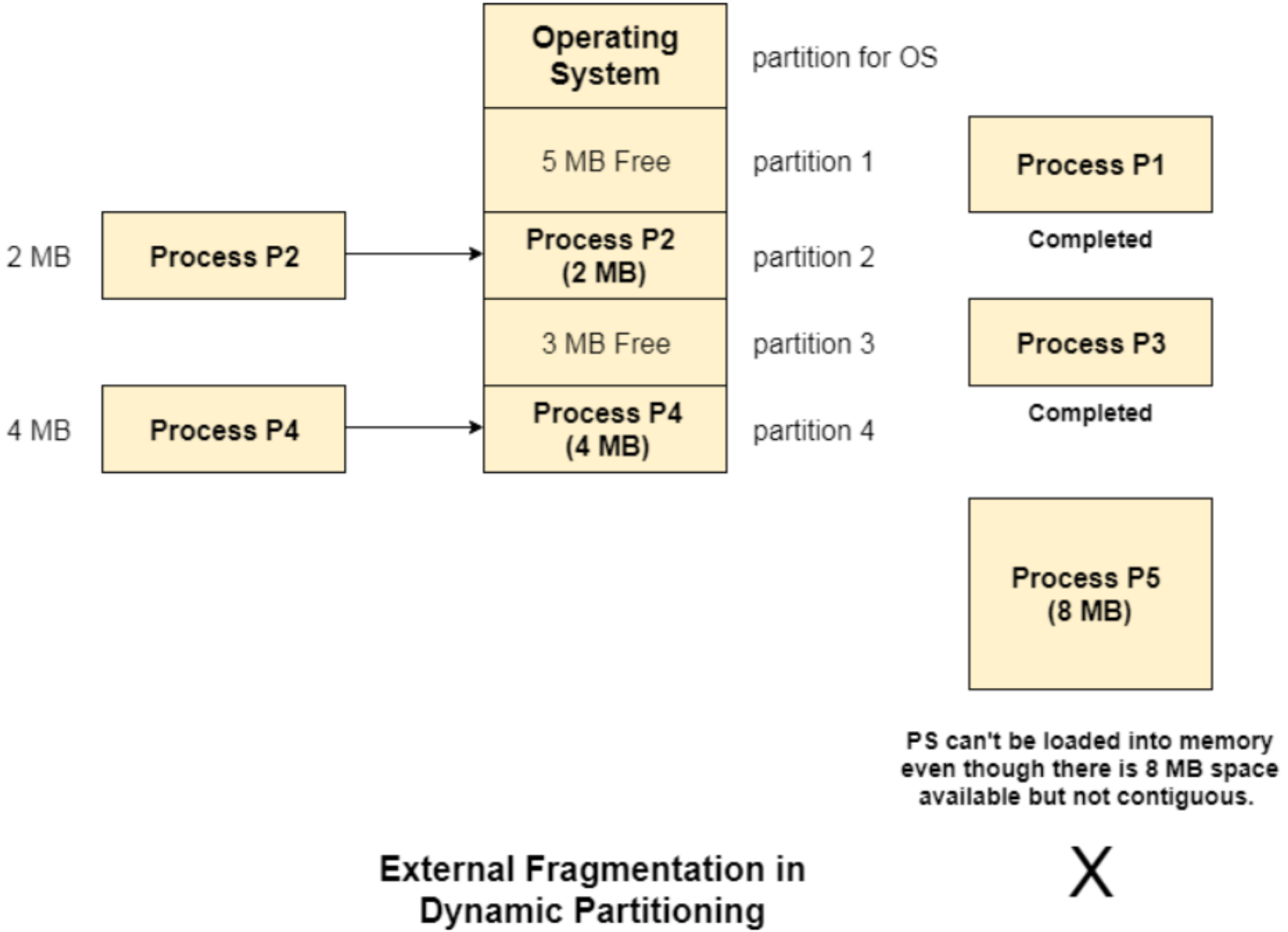


Dynamic Partitioning

(Process Size = Partition Size)

Fragmentation

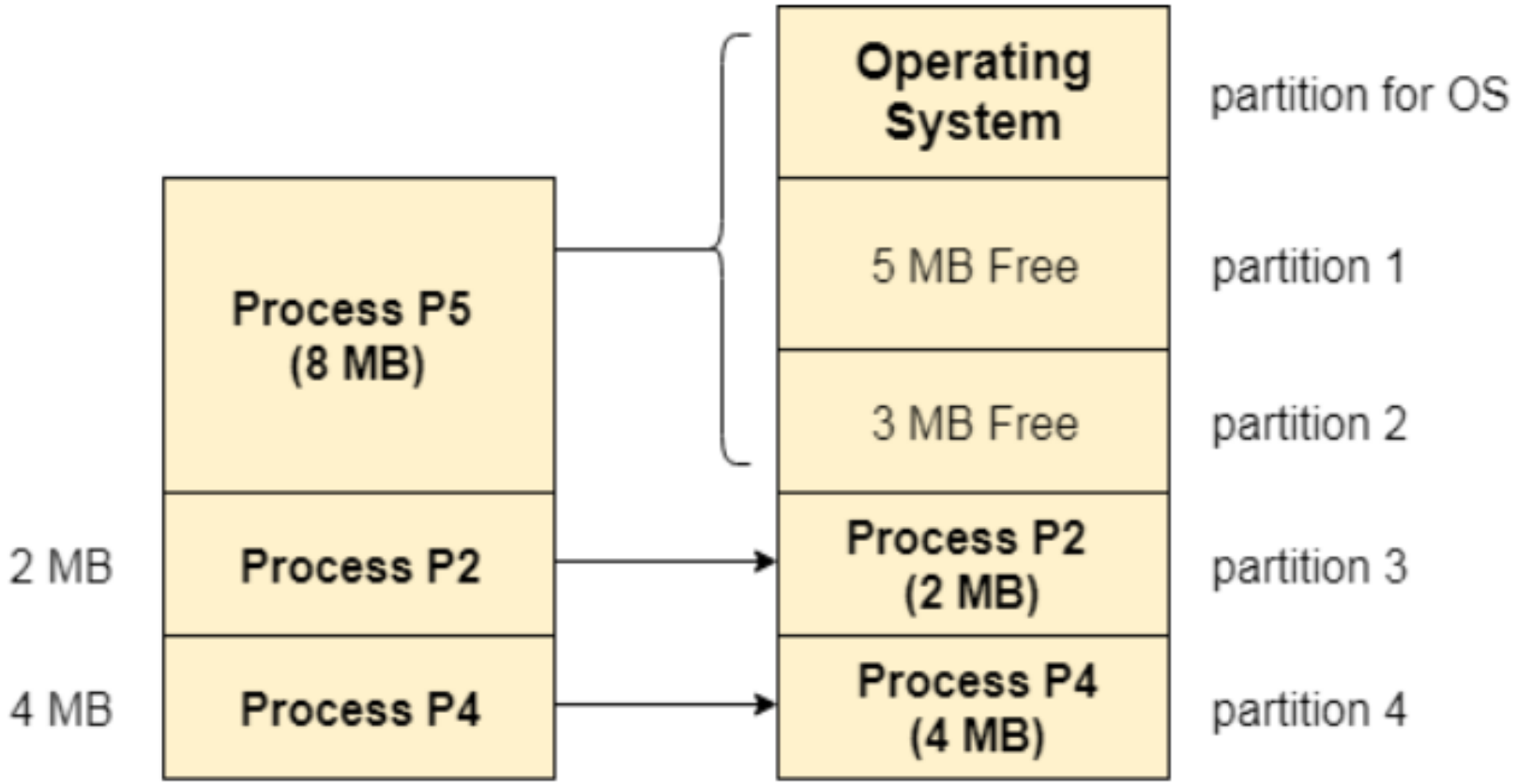
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation 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



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.

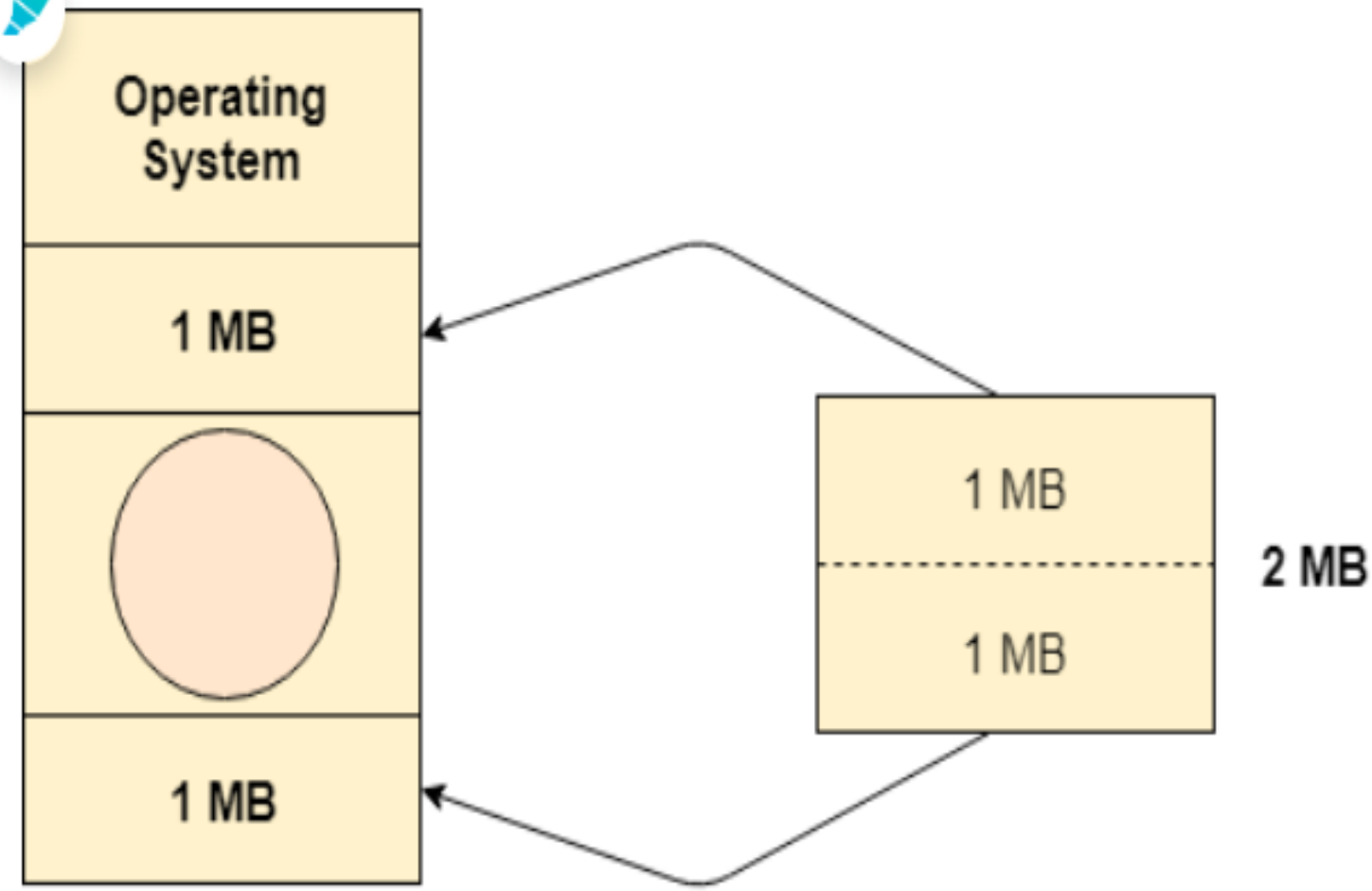


Now PS can be loaded into memory because the free space is now made contiguous by compaction

Compaction

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

Non-contiguous memory allocation:

Paging:

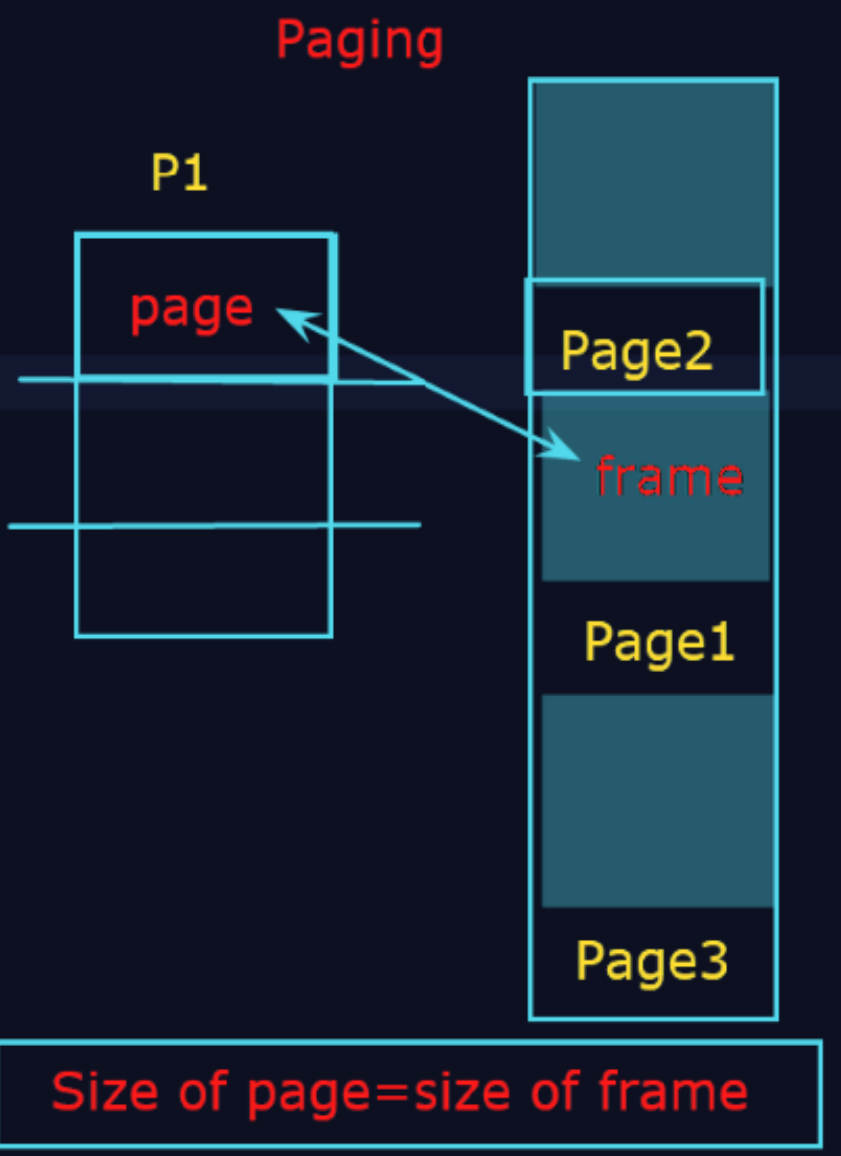
-Non-contiguous memory allocation

-OS

Segmentation:

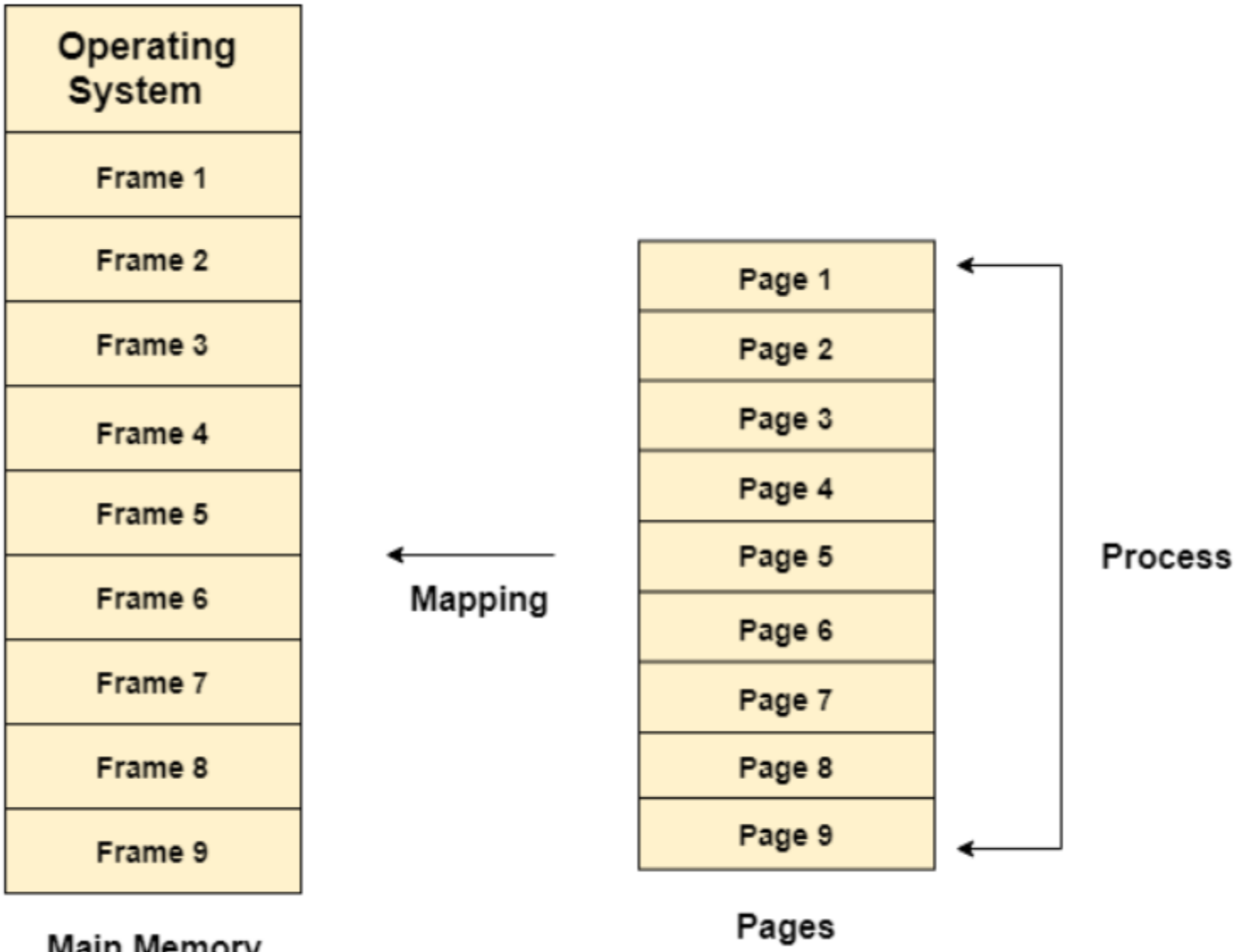
-Non-contiguous memory allocation

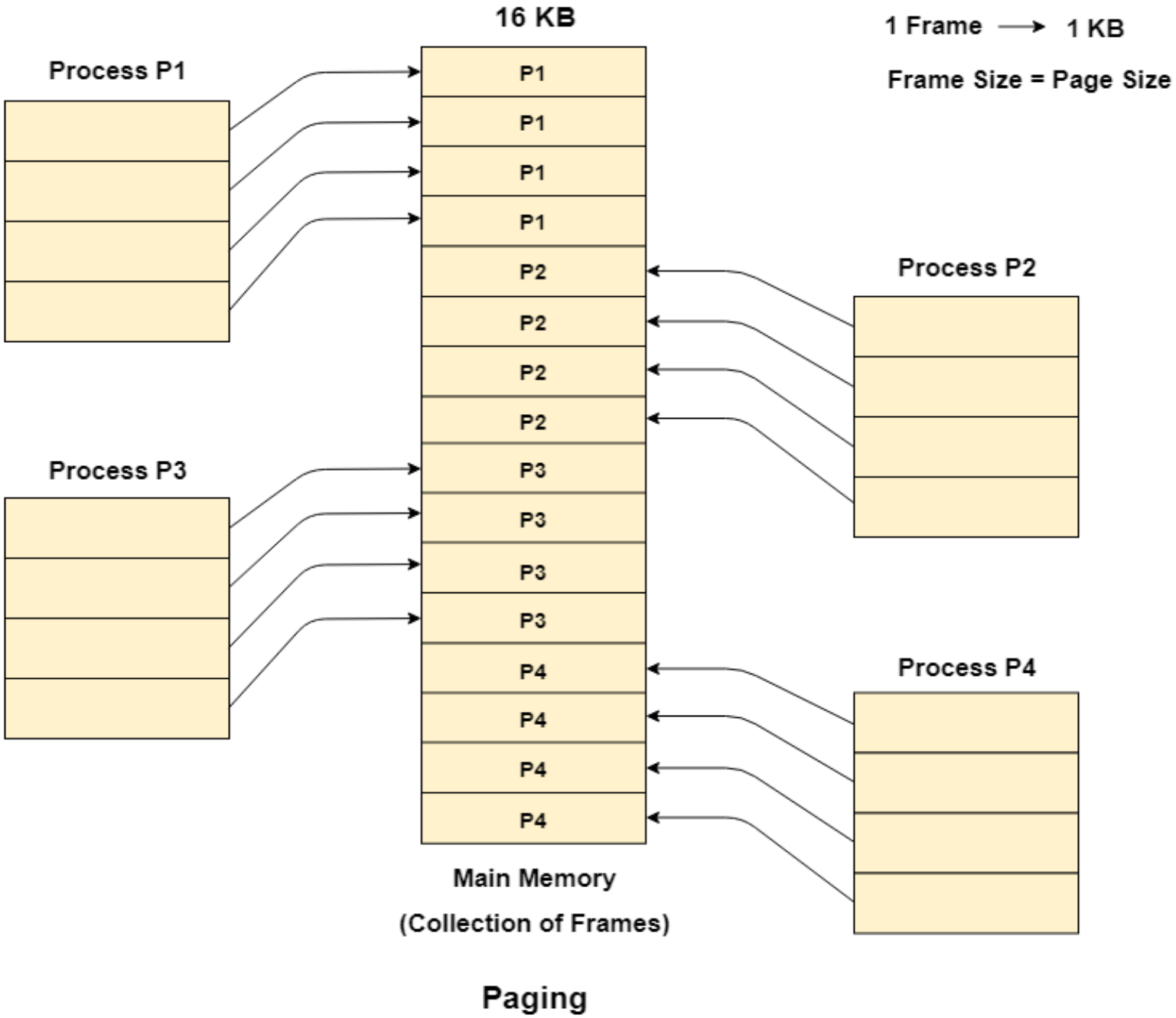
-Compiler



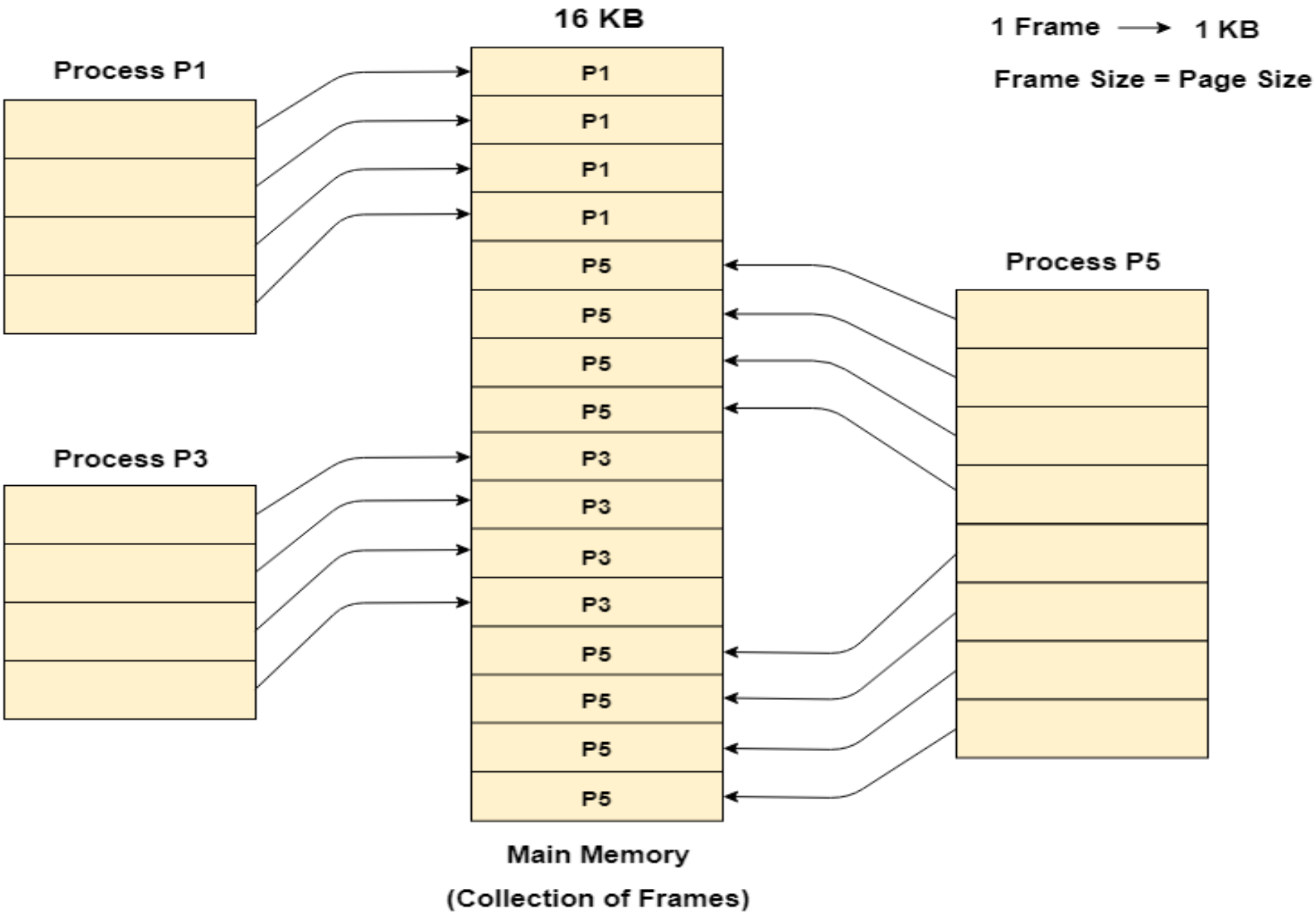
Paging

- 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





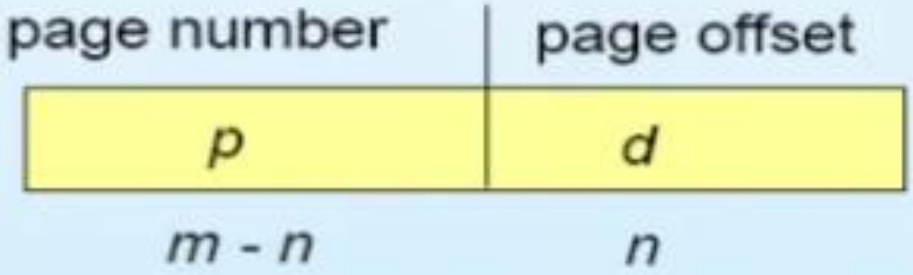
Paging


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)

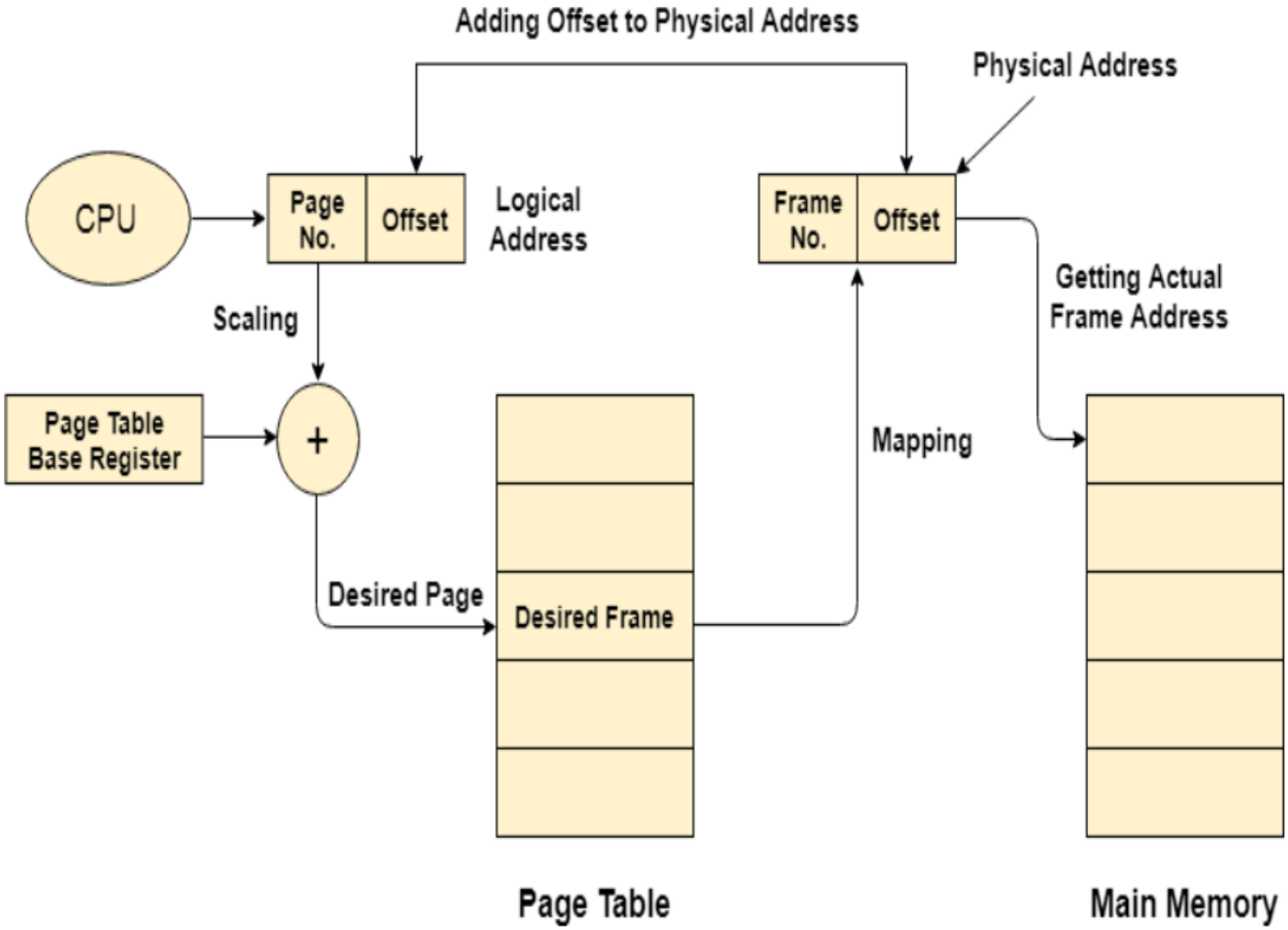
Paging

Paging Hardware

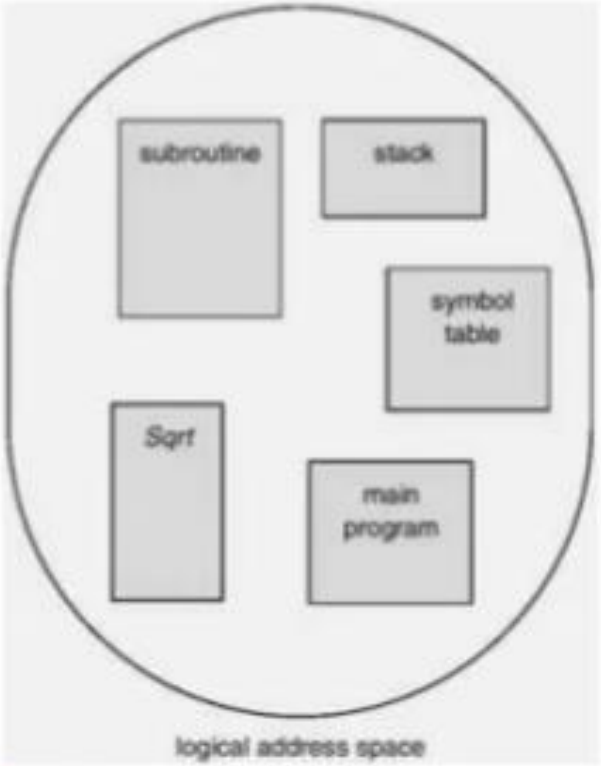
- Every address generated by the CPU is divided into two parts: **Page number (p)** and **Page offset (d)**
  - **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory
  - **Page offset (d)** – combined with base address to define the physical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



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

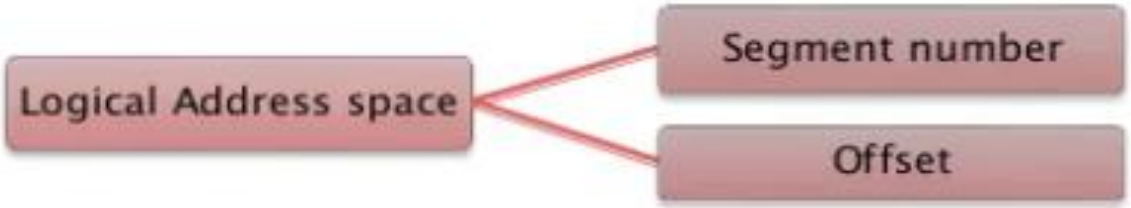


Segmentation



- 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.

Logical addressing in Segmentation



The mapping of the logical address to the physical address is done with the help of the segment table.

the length of the segment

SEGMENT TABLE

Segment Limit	Segment Base	Other bits

starting address of the corresponding segment in main memory

A bit is needed to determine if the segment is already in main memory (P)
Another bit is needed to determine if the segment has been modified since it was loaded in main memory (M)

Segmentation

- **Segments are variable-sized**
  - Dynamic memory allocation required (first fit, best fit, worst fit).
- **External fragmentation**
  - In the worst case the largest hole may not be large enough to fit in a new segment. Note that paging has no external fragmentation problem.
- **Each process has its own segment table**
  - like with paging where each process has its own page table. The size of the segment table is determined by the number of segments, whereas the size of the page table depends on the total amount of memory occupied.
- **Segment table located in main memory**
  - as is the page table with paging
- **Segment table base register (STBR)**
  - points to current segment table in memory
- **Segment table length register (STLR)**
  - indicates number of segments

Non-contiguous memory allocation:

Paging:

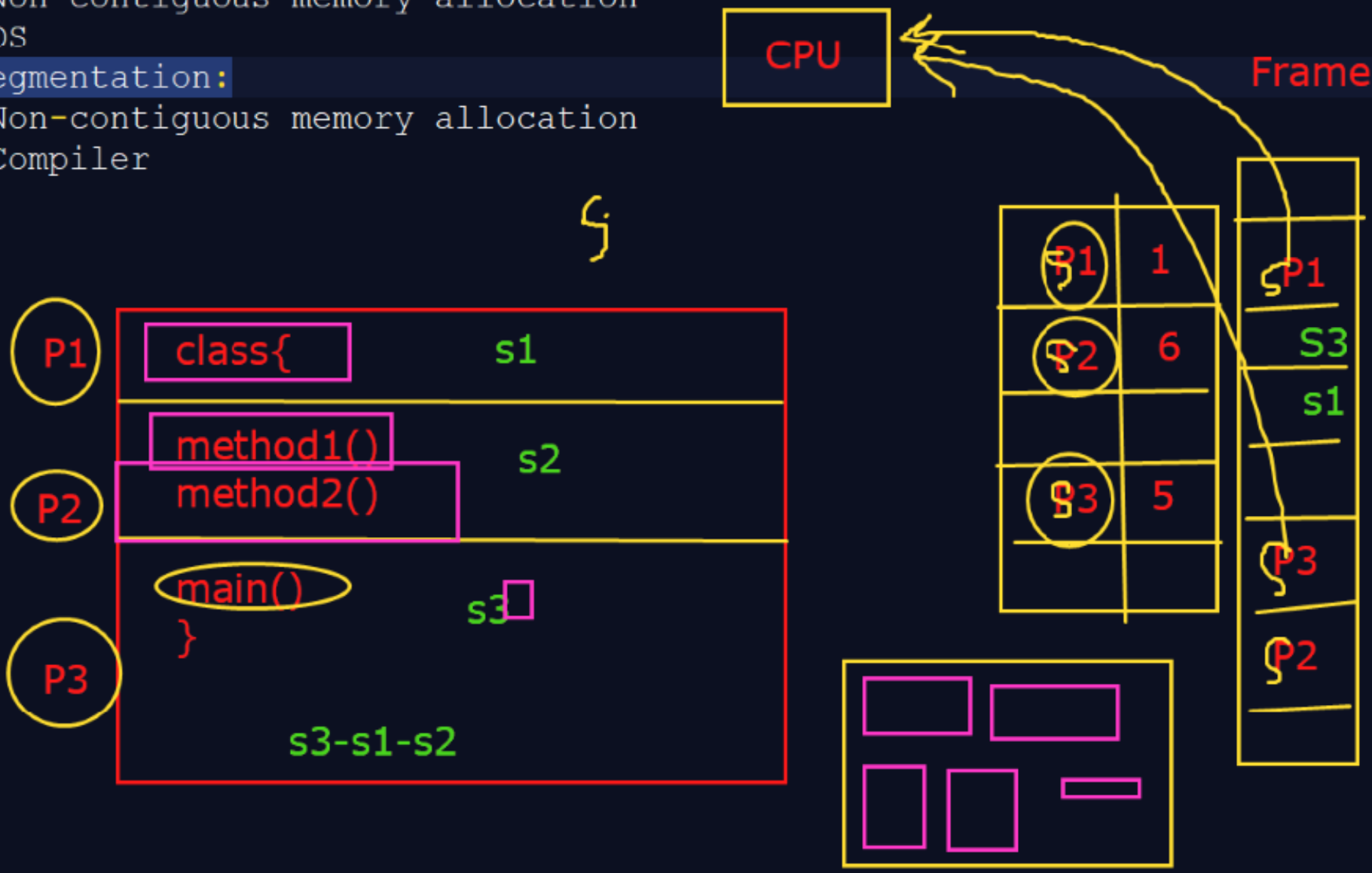
-Non-contiguous memory allocation

-OS

Segmentation:

-Non-contiguous memory allocation

-Compiler



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

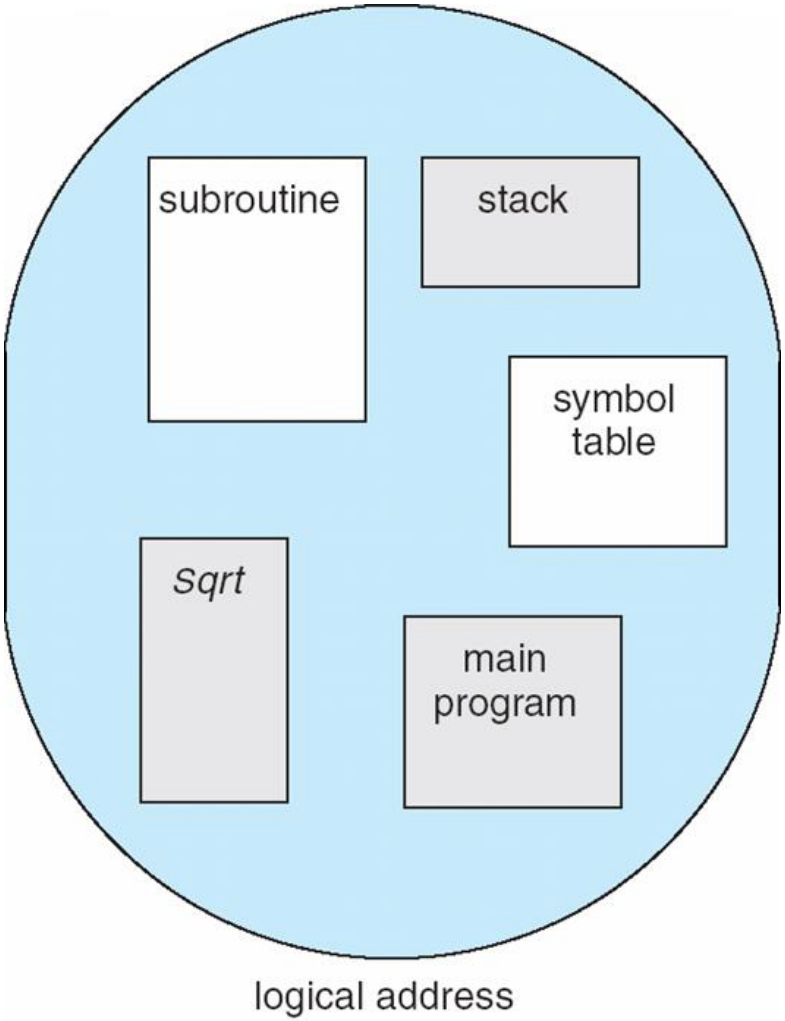
    common block

    stack

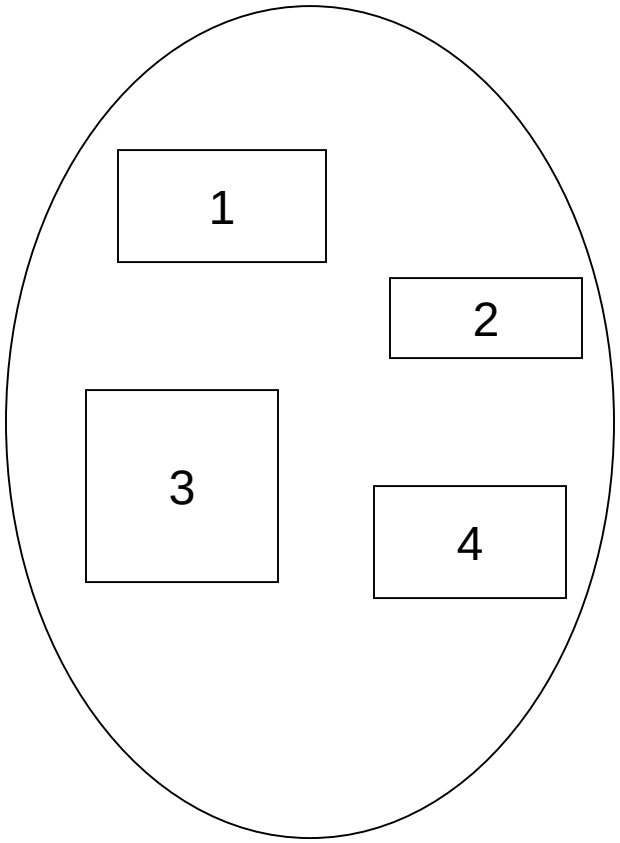
    symbol table

    arrays

User's View of a Program



Logical View of Segmentation



user space

physical memory space

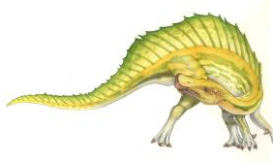
Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

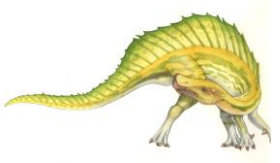
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location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

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

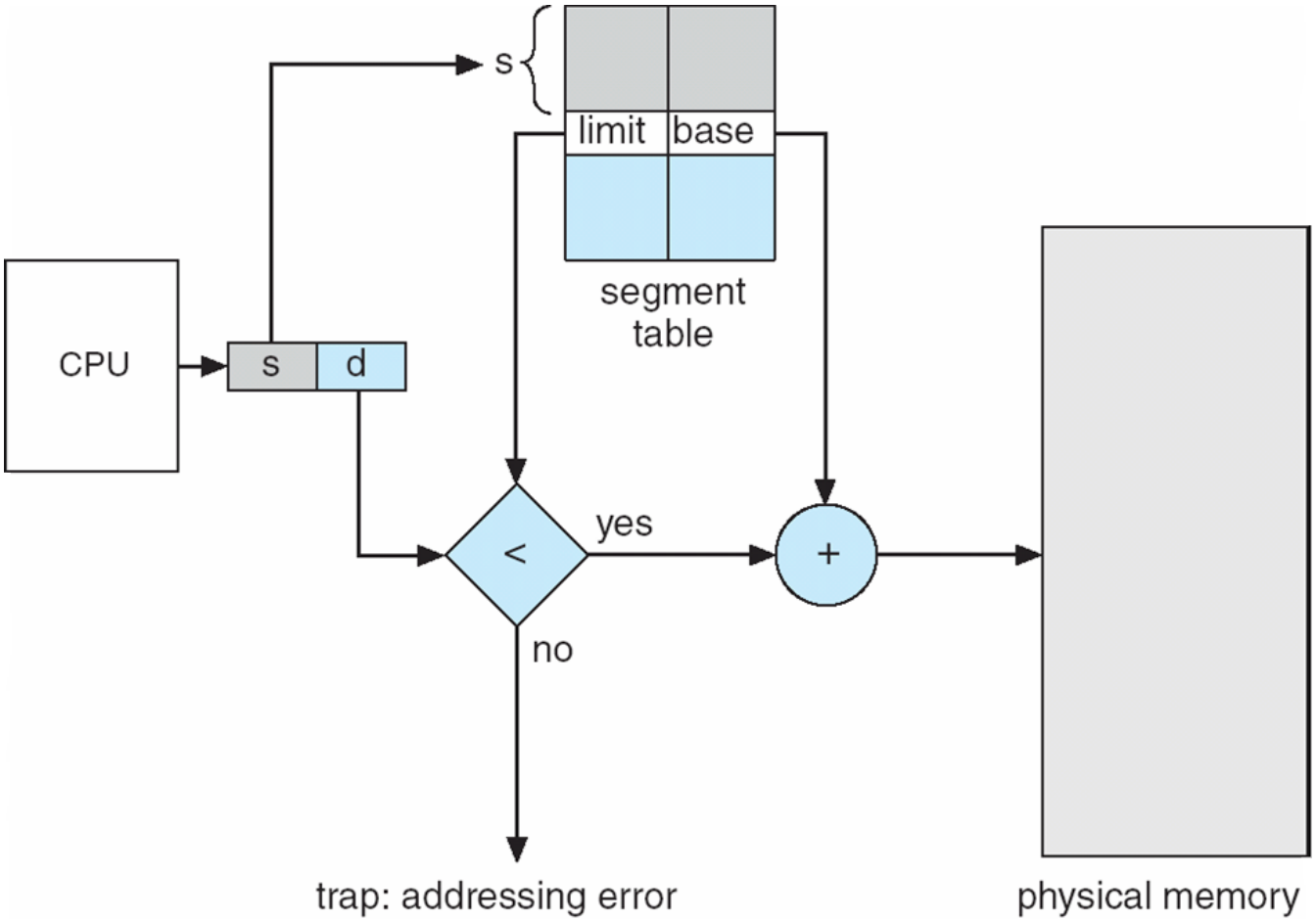


Segmentation Architecture (Cont.)

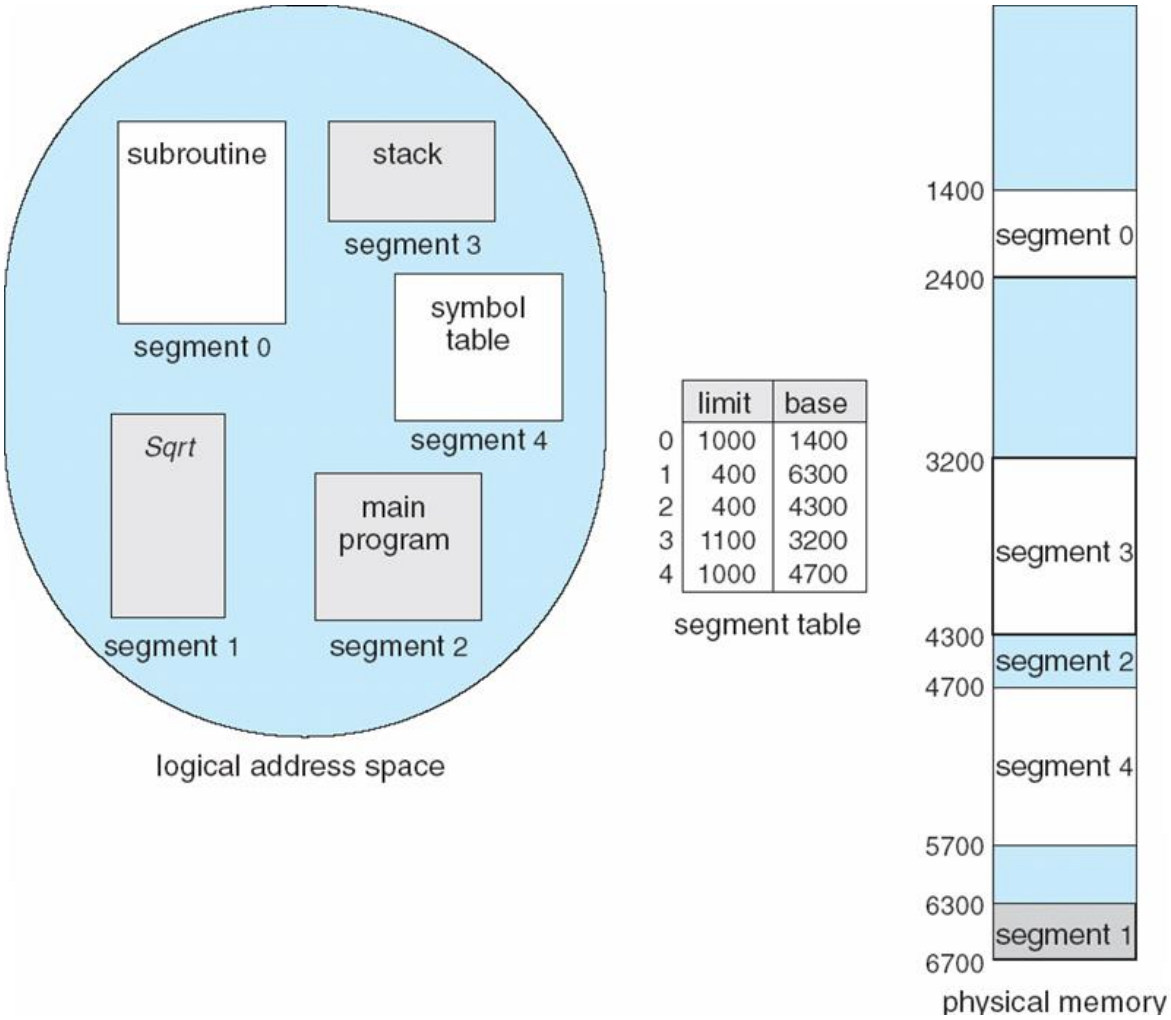
- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

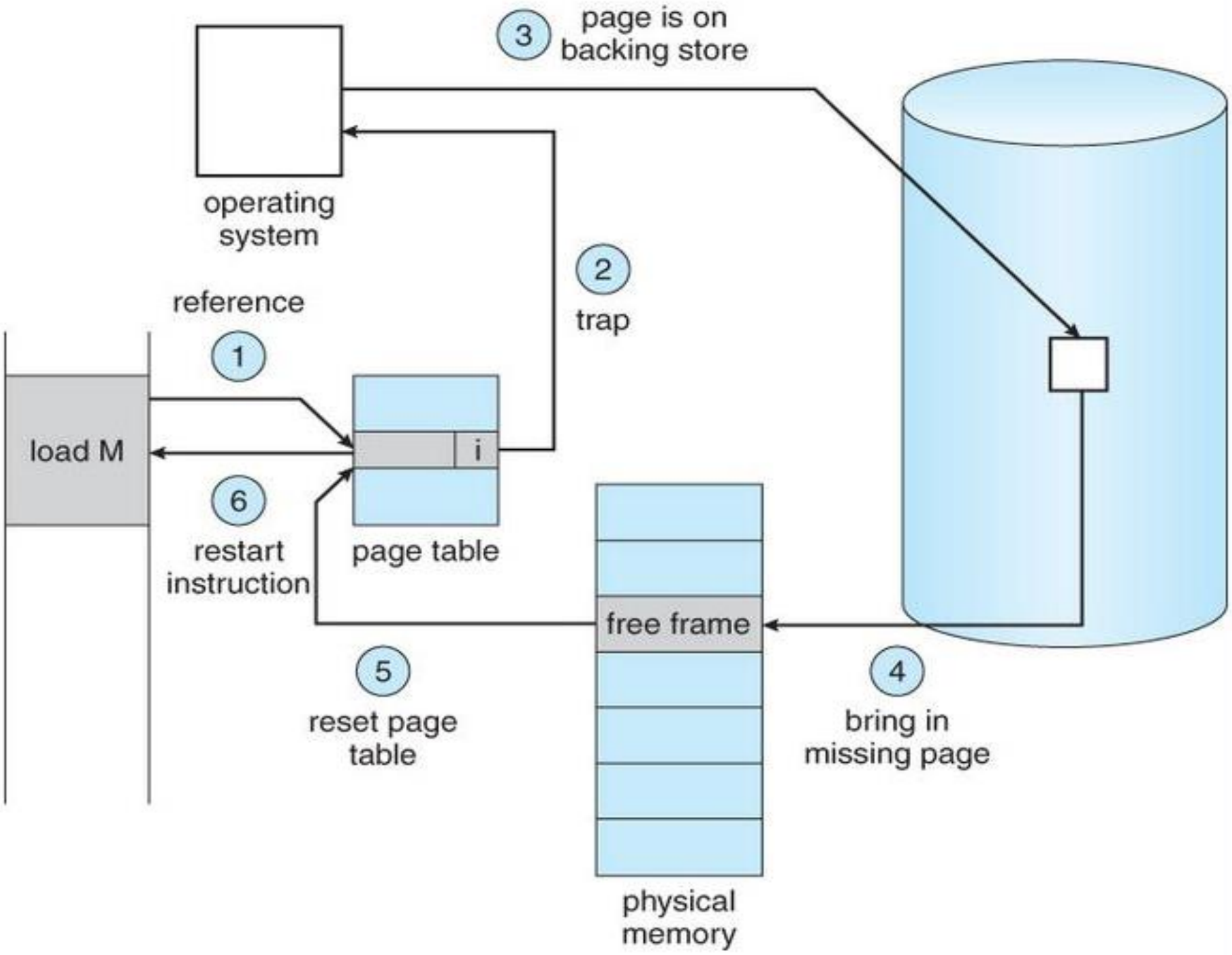


Segmentation Hardware



Example of Segmentation

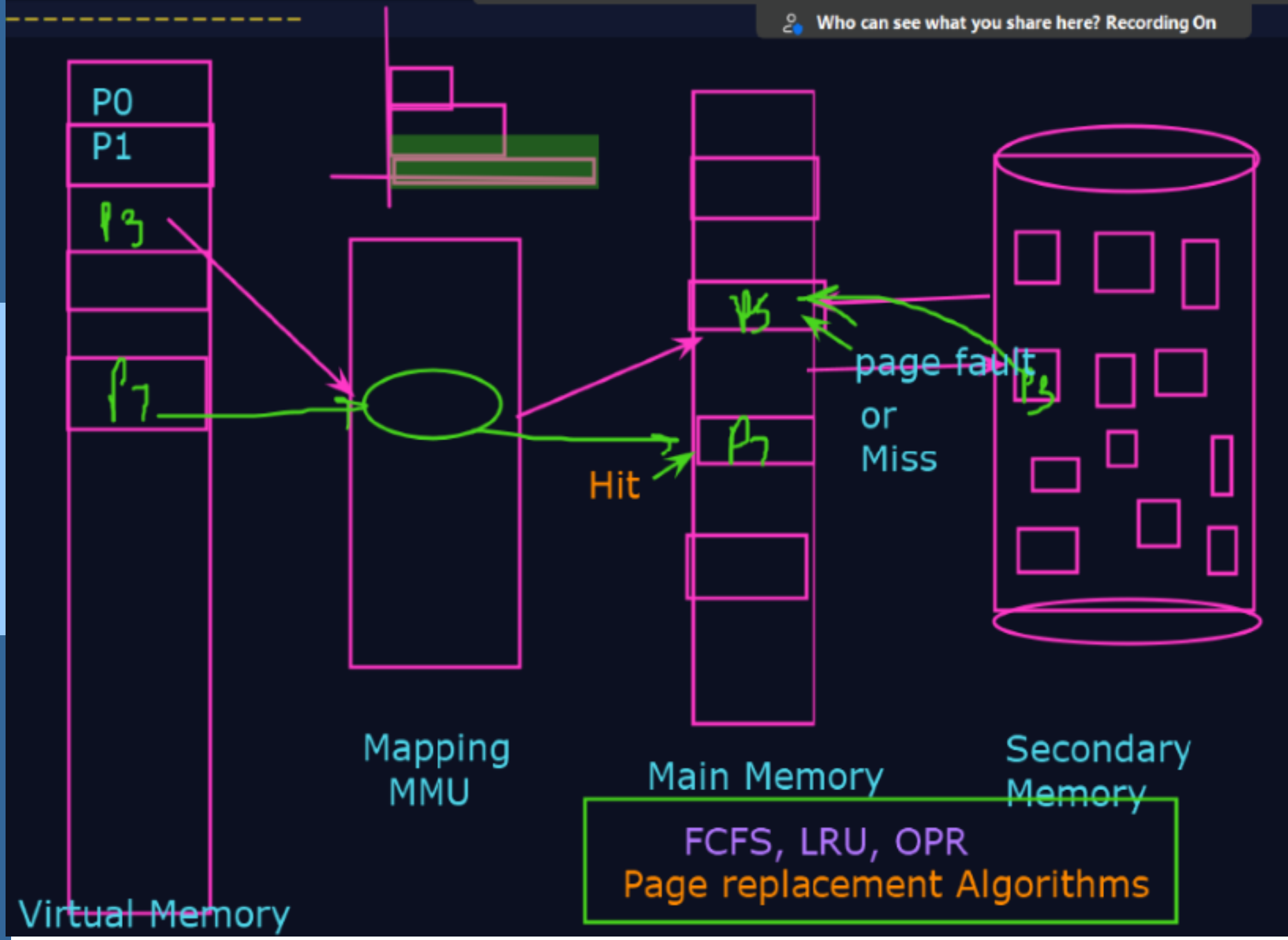



3
page is on
backing store
operating
system
2
trap
reference
1
load M
i
6
restart
instruction
page table
free frame
5
reset page
table
4
bring in
missing page
physical
memory

Virtual Memory:

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo

Who can see what you share here? Recording On



Page Fault Algorithms: FIFO,OPR,LRU

Sequence: 7,0,1,2,0,3,0,4,2,3,0,3,1,2,0 frame:3

Calculate : Hit, Miss (page fault)

FIFO

f3			1	1	1	1	0	0	0	3	3	3	2	2
f2		0	0	0	0	3	3	3	2	2	2	1	1	1
f1	7	7	7	2	2	2	2	4	4	4	0	0	0	0
	★	★	★	★	✓	★	★	★	★	★	★	★	★	✓

Hit=3

Miss=12

Hit Ratio = 3/15*100=

Miss Ratio = 13/15*100=

Who can see what you share here? Recording On

Virtual Memory:

FCFS:
LRU:
OPR:

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

f1			2	2	2	2	2	2	2	2	2	2	2	2
f2		1	1	1	1	4	4	4	4	4	4	4	4	4
f3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f4	7	7	7	7	7	3	3	3	3	3	3	3	1	1

Belady's Anomaly:

Pagefault increase-->
no of frames are increasing

Inteprocess Communication :

1. Independent Process

2. Cooperative Process :sharing

-information sharing

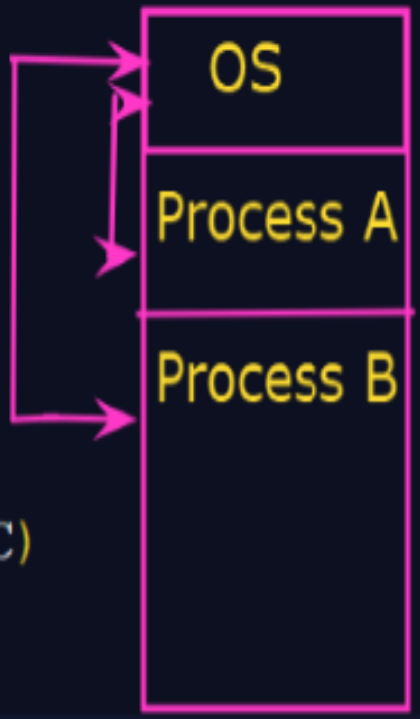
-computation speed

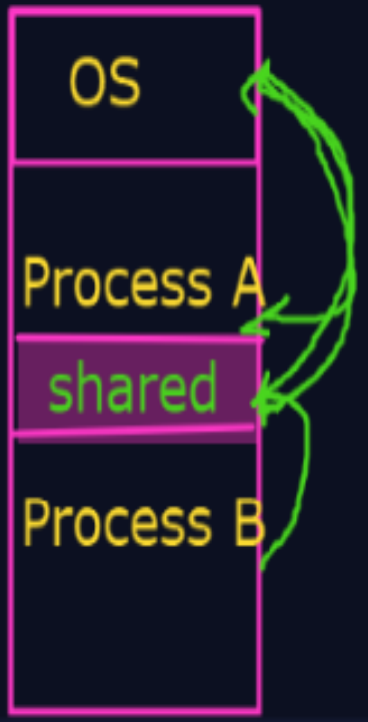
-Interprocess communication (IPC)

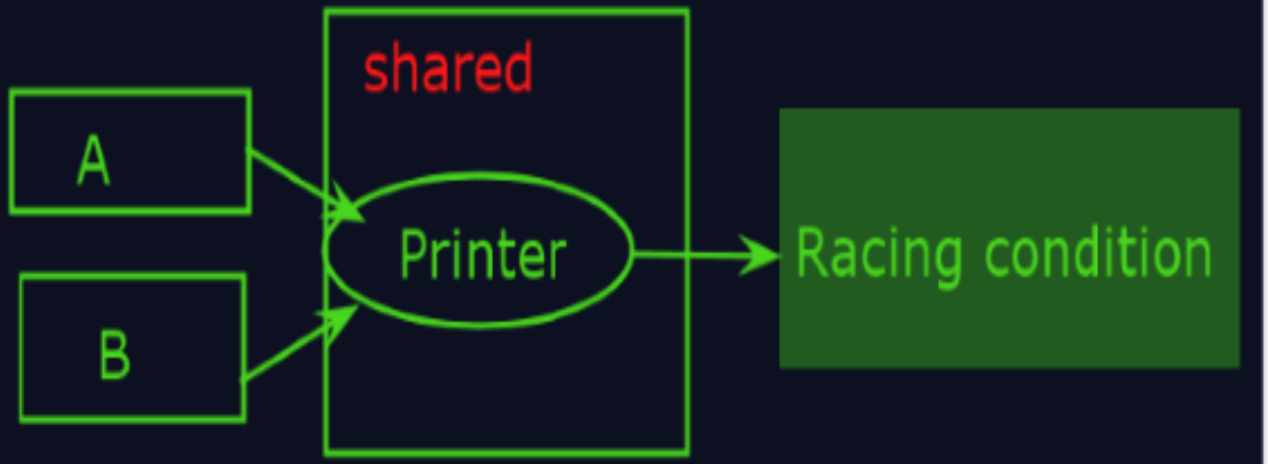
-2 models:

-shared memory

-Message passing





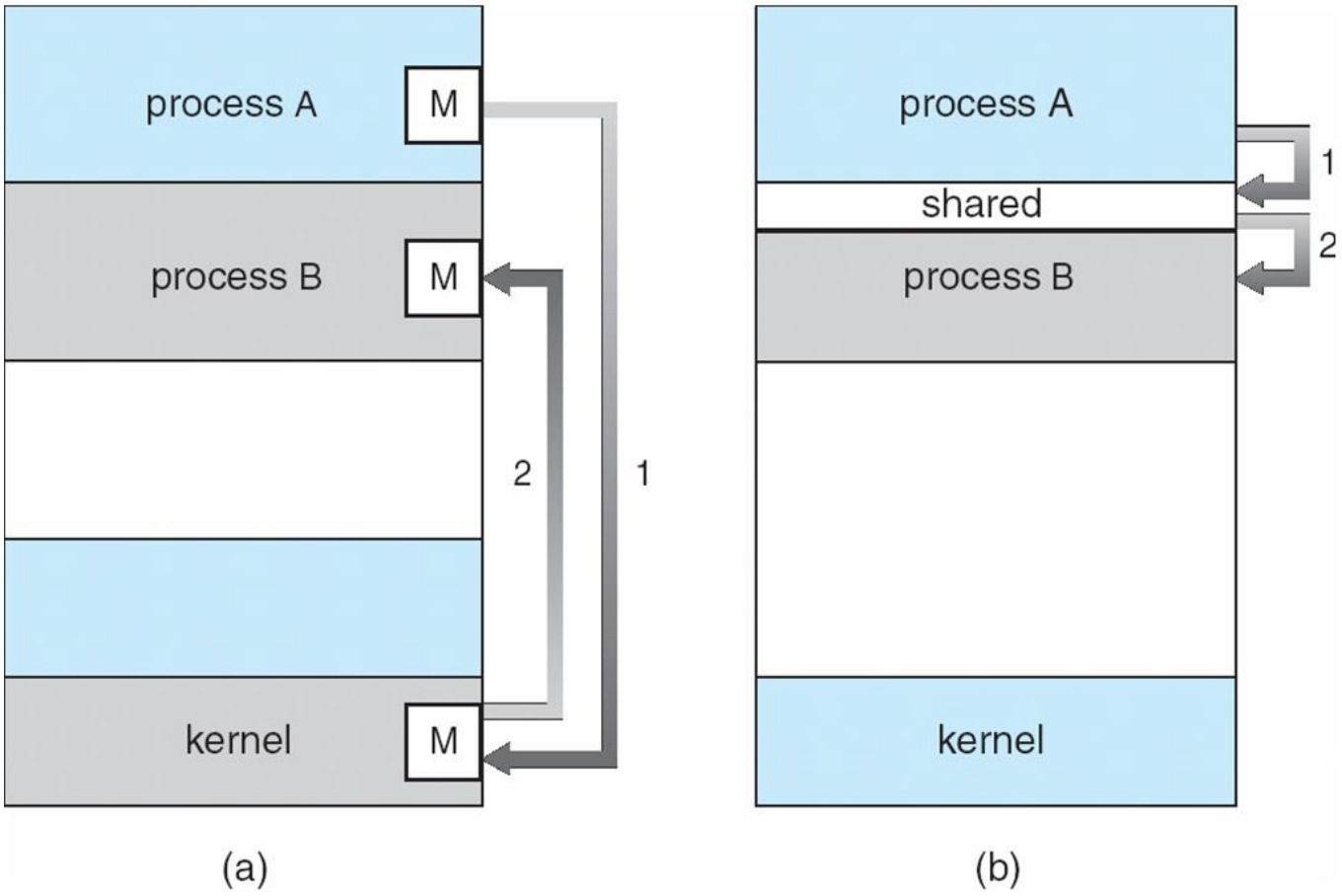


Soln: Synchronization

Interprocess Communication

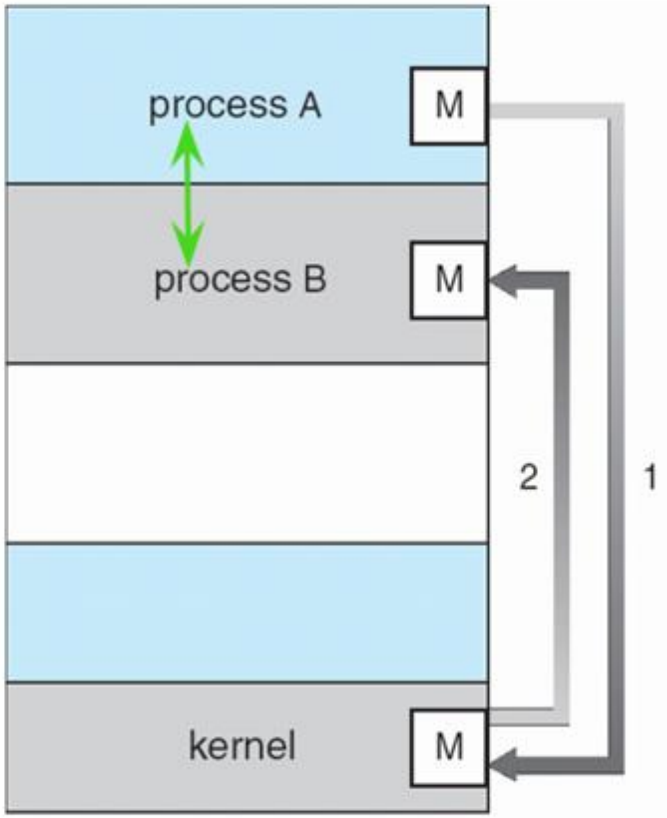
- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - Shared memory
  - Message passing

Communications Models



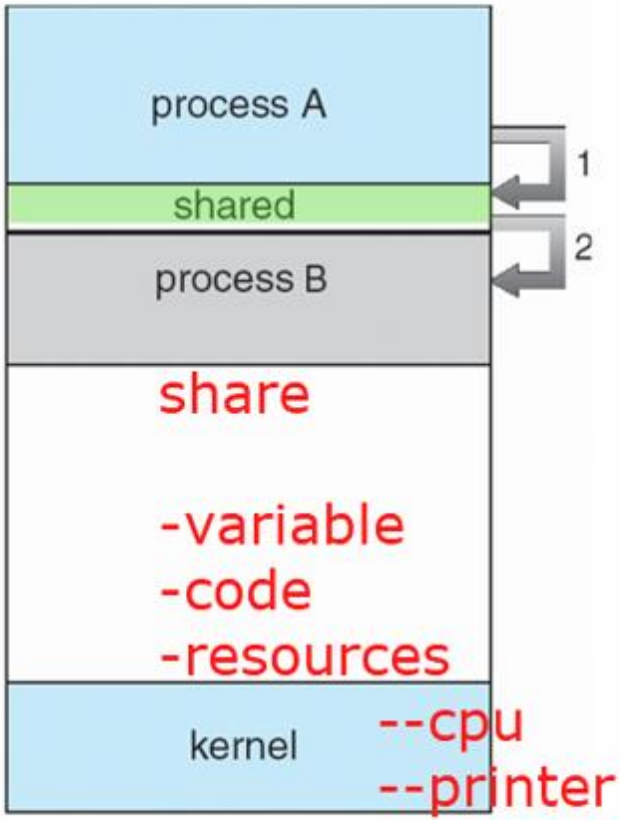
Communications Models

Independent process



(a)

Cooperative process



(b)

Soln : Synchronization

Process A

Process B

Ask for same resource-->Racing cond

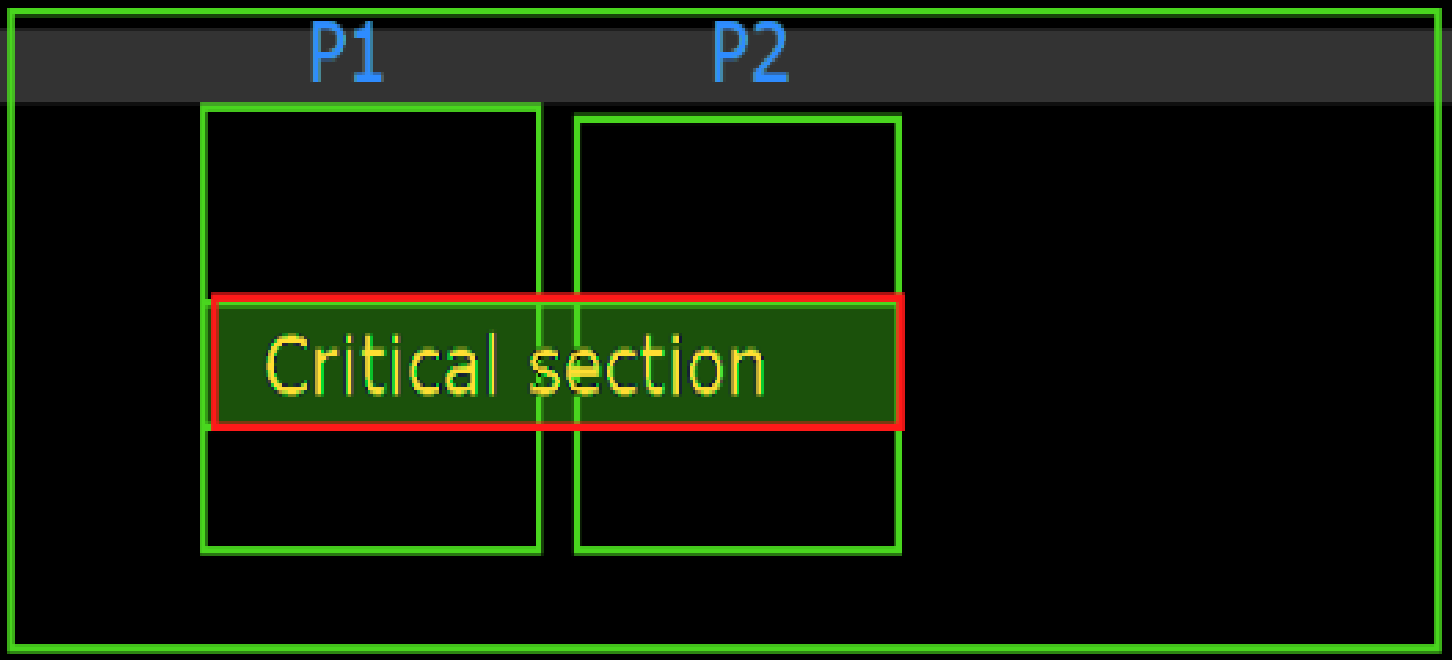
Cooperating Processes

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

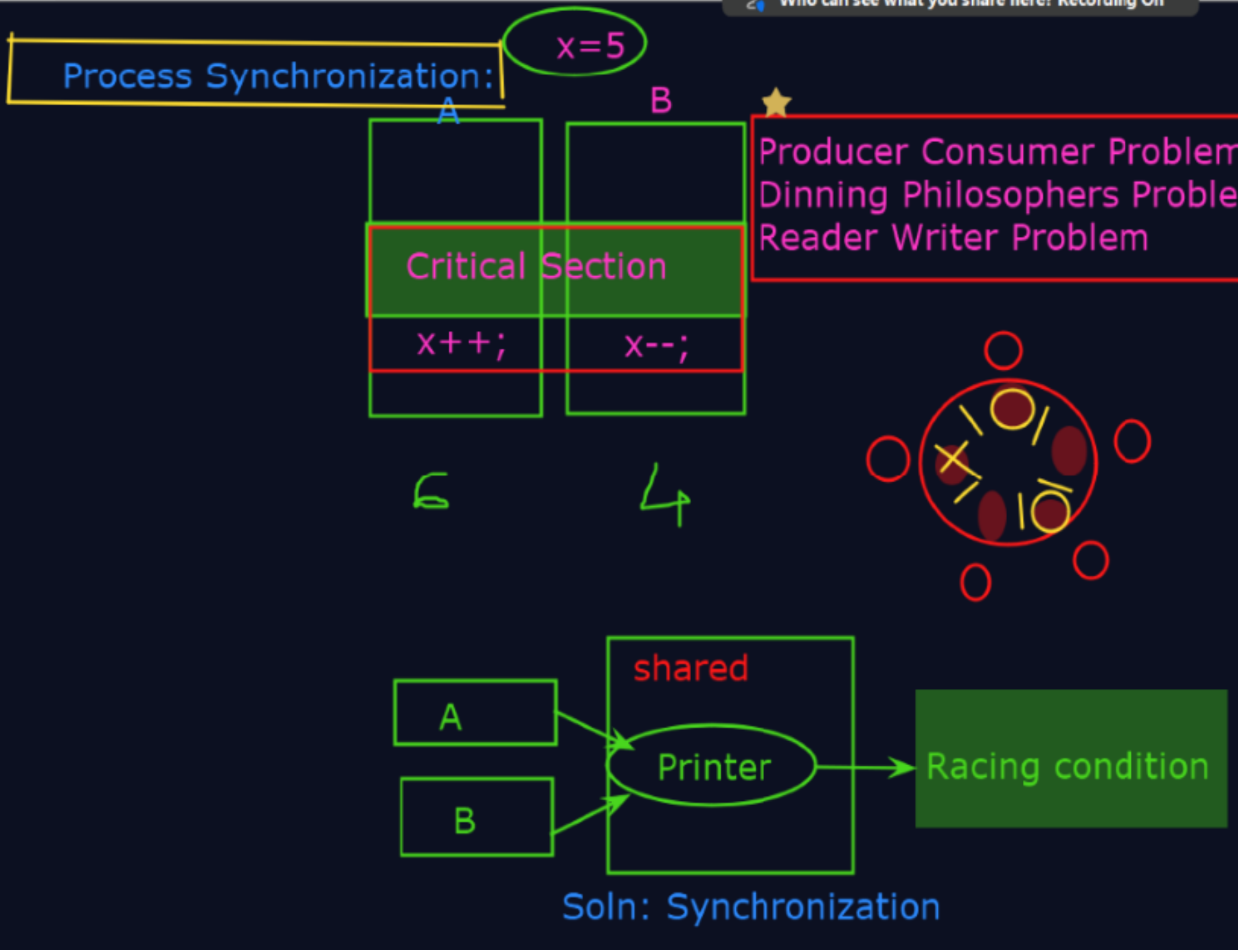
- Independent Processes
- Two processes are said to be independent if the execution of one process does not affect the execution of another process.

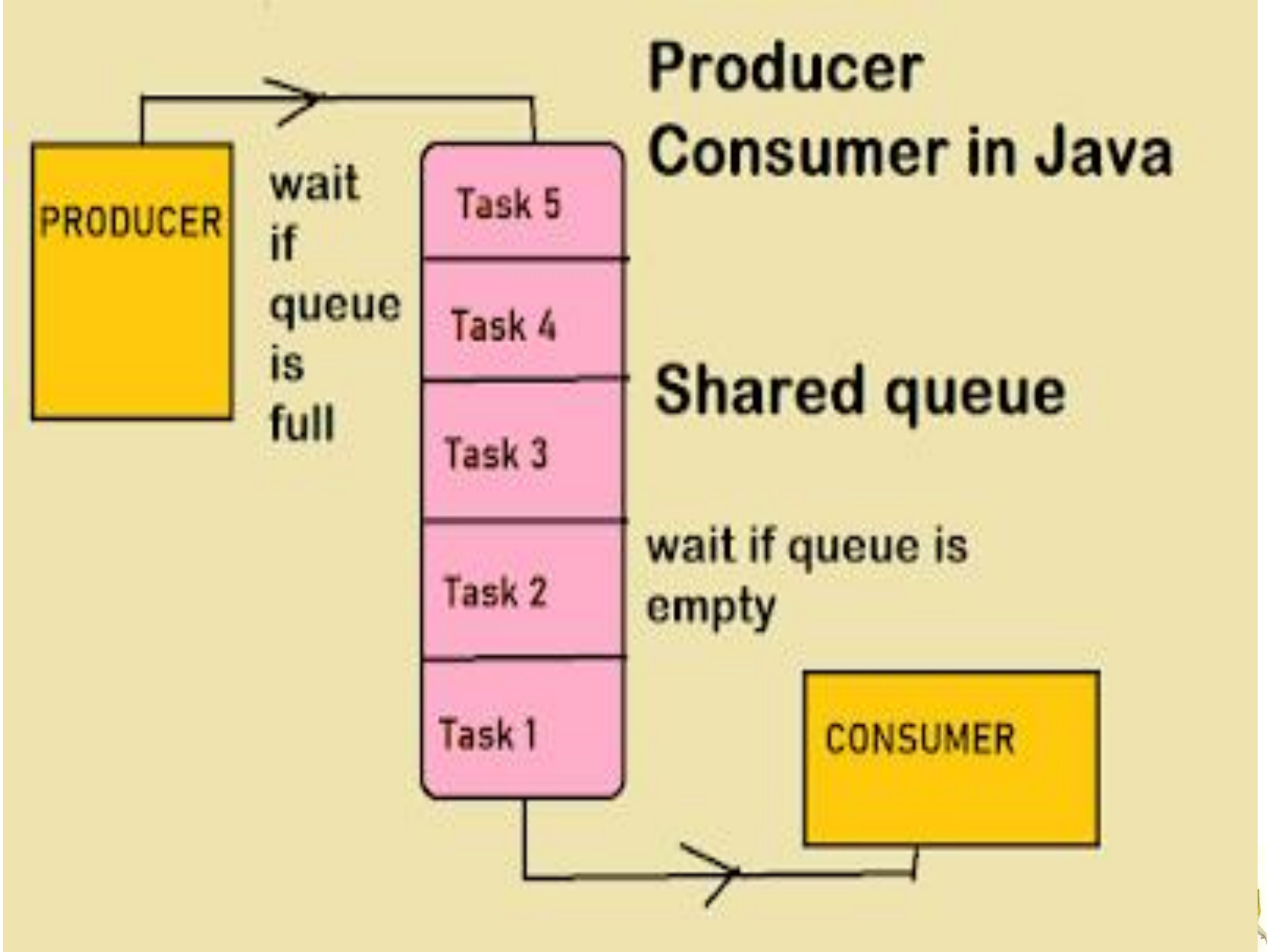
- Cooperative Processes
- Two processes are said to be cooperative if the execution of one process affects the execution of another process. These processes need to be synchronized so that the order of execution can be guaranteed.

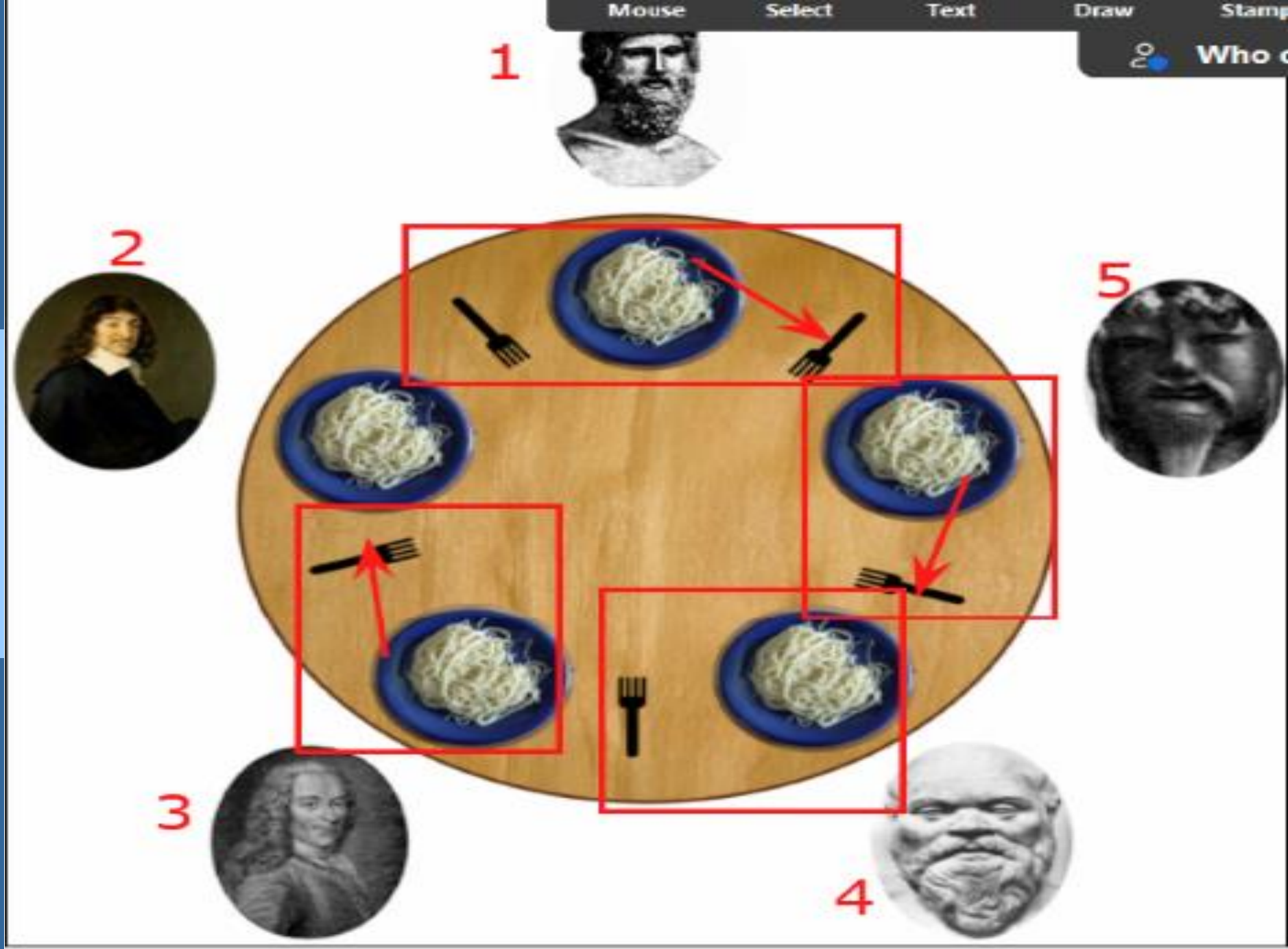
Process Synchronization:



Producer - Consumer Problem
Dinning Philosophers
Reader Writer Problem




Producer
Consumer in Java
PRODUCER
wait
if
queue
is
full
Task 5
Task 4
Task 3
Task 2
Task 1
Shared queue
wait if queue is
empty
CONSUMER


Mouse
Select
Text
Draw
Stamp
Who
1
2
3
4
5