OPERATING SYSTEM

UNIT-3

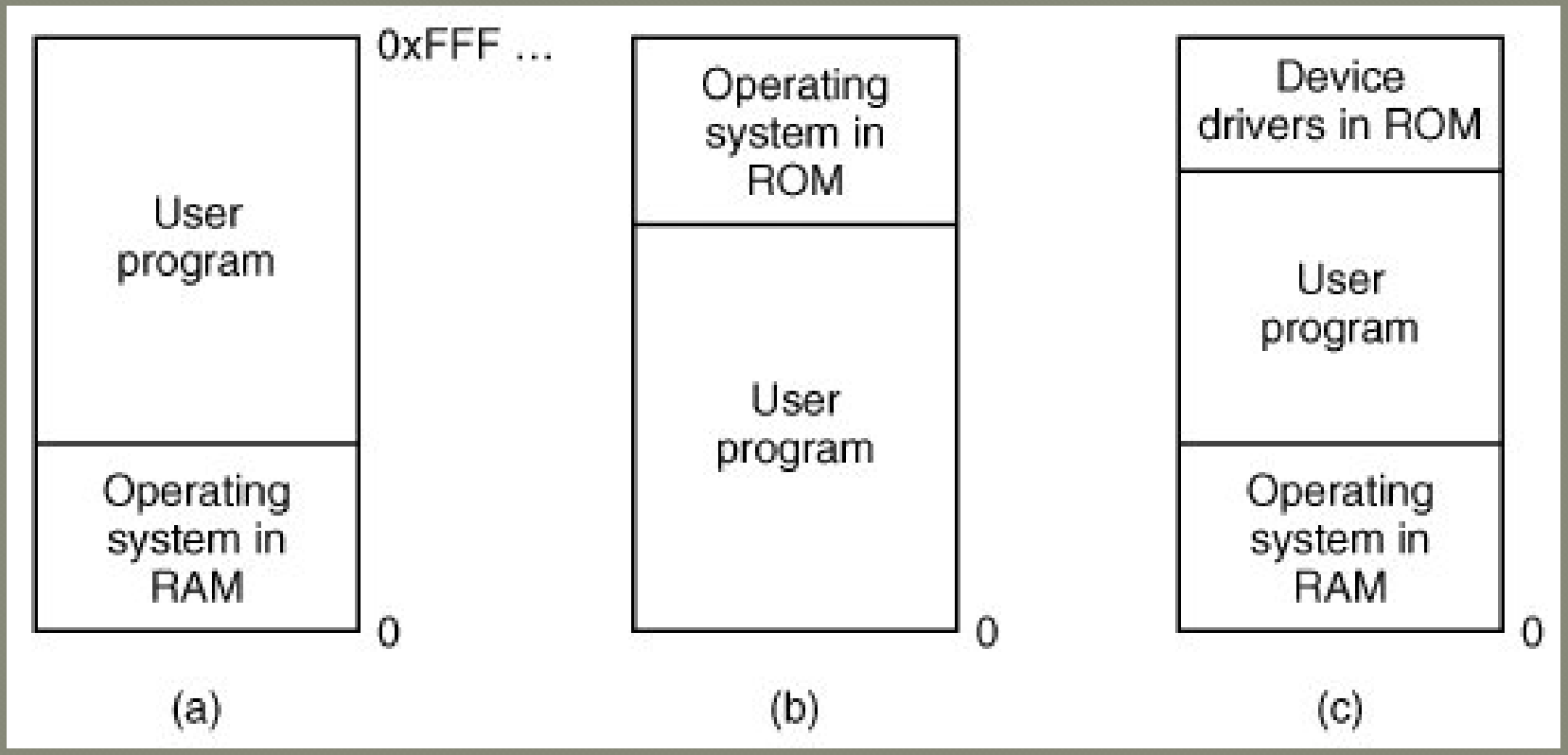
Memory Management

CSTutorialpoint.Com

Preliminaries of Memory Management

- Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Organizing memory with one user process

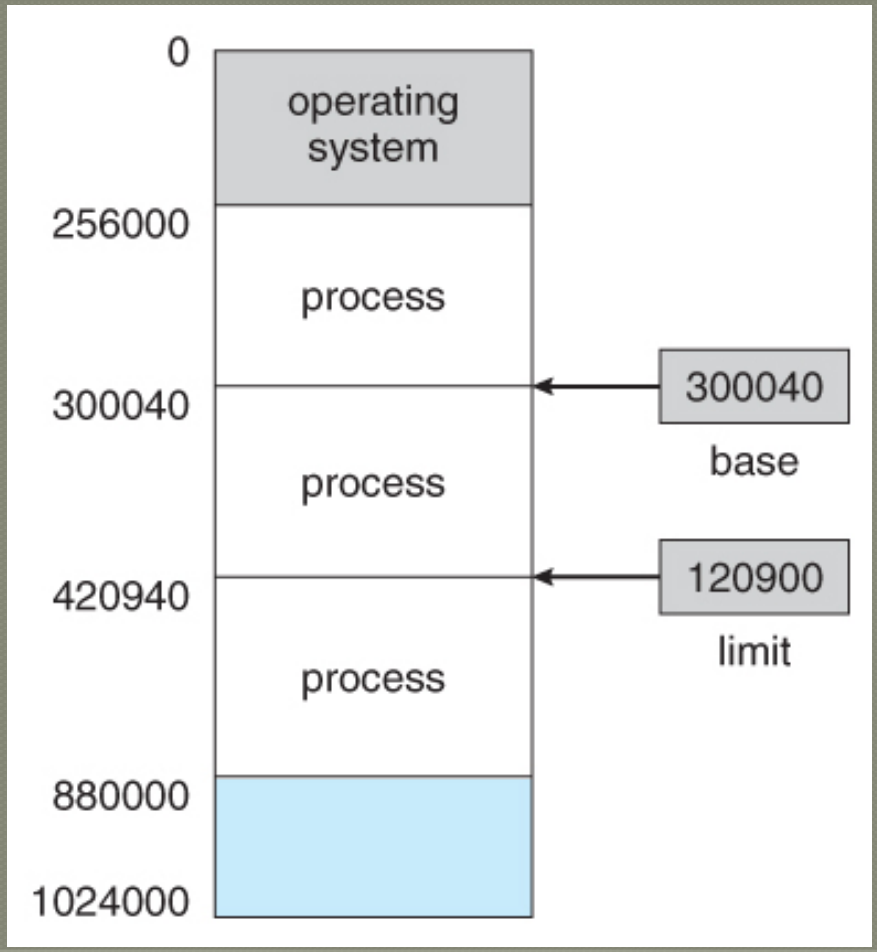


Basic Hardware

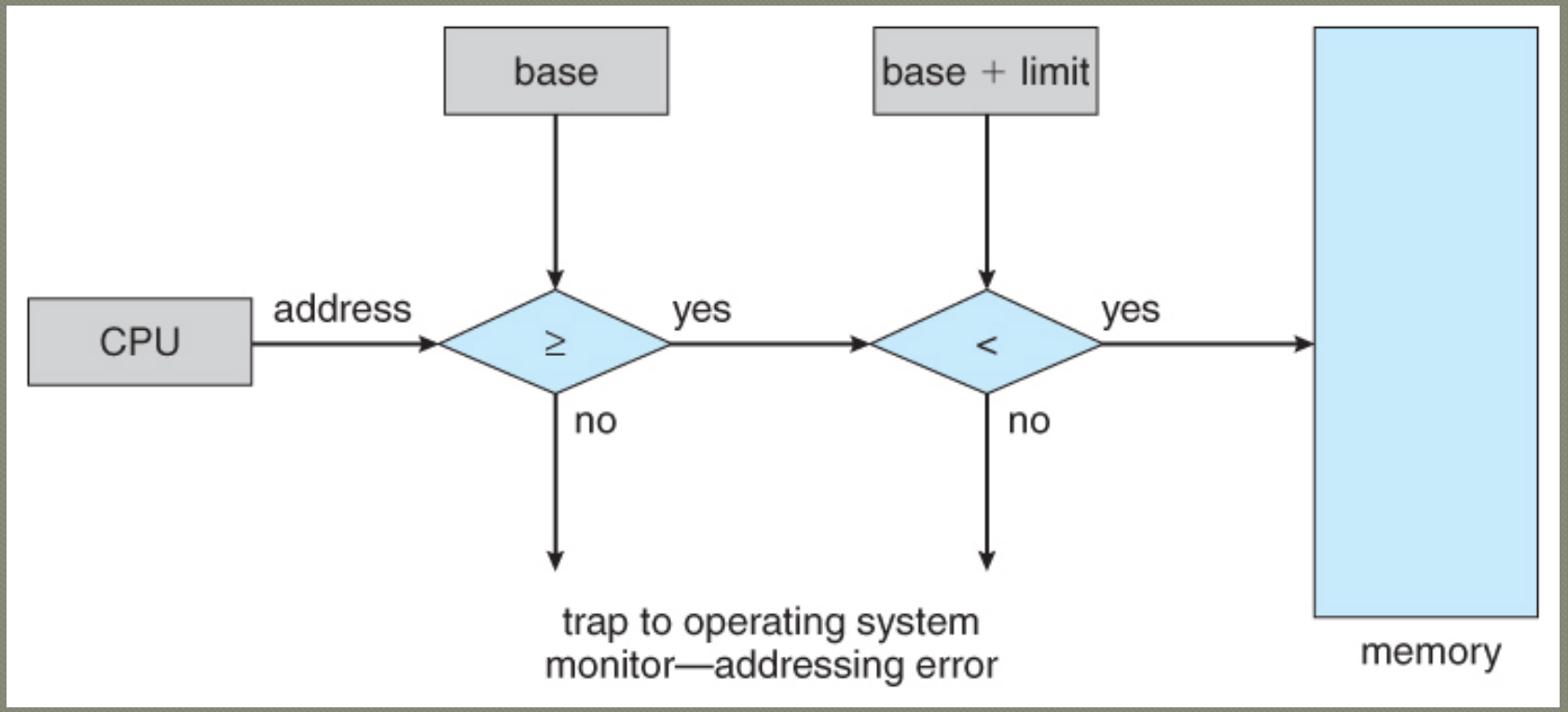
- Each process has a separate memory space. Separate per process memory space protects the process from each other and is fundamental to having multiple processes loaded in memory for concurrent execution.
- To separate memory space we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
- We can provide this protection by using two registers – base register and limit register.

- Base register- holds the smallest legal physical memory address.

- Limit register- specifies the size of the range.

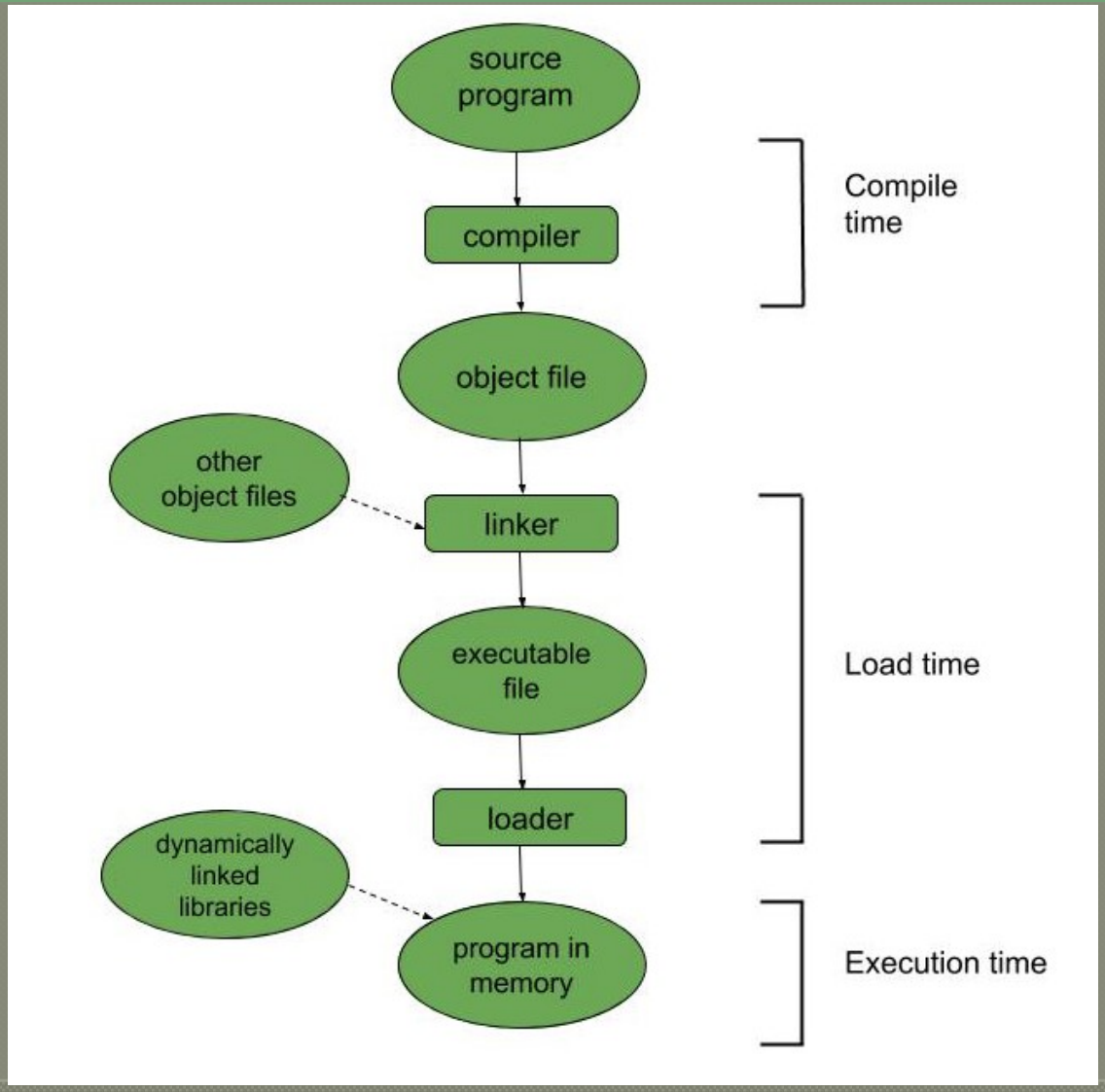


- The base and limit registers can be loaded only by the operating system.



Address binding

- A user program goes through several steps before being executed.



- Most systems allow a user process to reside in any part of the physical memory.
- Thus, address space of the computer starts at 00000
- first address of the user process need not be 00000.

Source Program

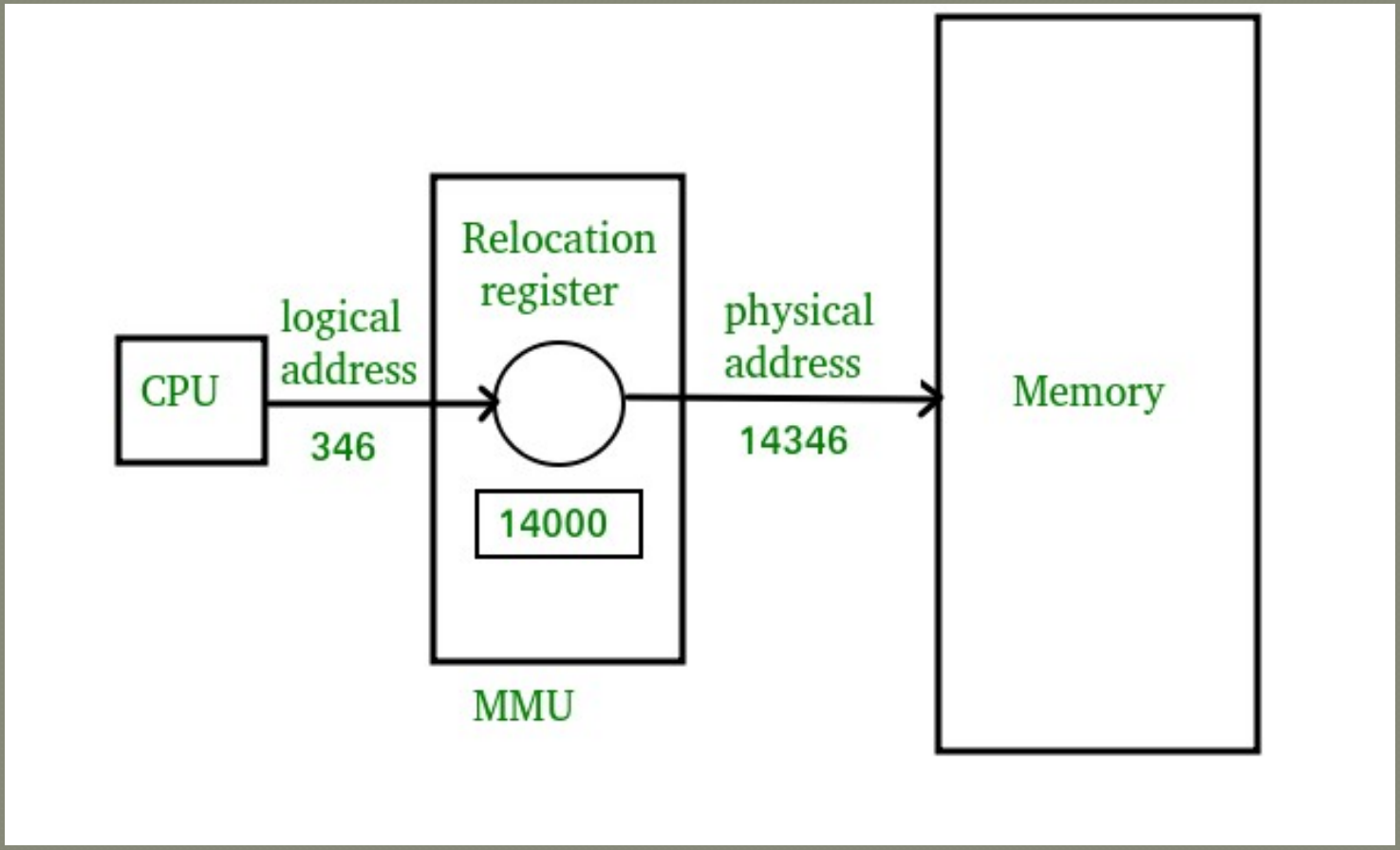
- Compile time:- if you know at compile time where the process will reside in memory then absolute code can be generated.
- Load time:- if it is not known at compile time where the process will reside in memory then compiler must generate relocatable code .
- Execution time:- if the process can be moved during its execution from one memory segment to another then binding must be delayed until run time.

Logical vs. Physical Address space

- An address generated by the CPU is referred to as logical address.
- An address seen by the memory unit i.e. the one loaded into the memory address register of the memory is referred to as a physical address.
- The compile time and load time address binding methods generate same logical and physical addresses. However the execution time address binding results in different logical and physical addresses . In this case the logical address is referred to as virtual address.

- The set of all logical addresses generated by a program is called “logical address space” .
- The set of all physical addresses corresponding to these logical addresses is called “Physical address space” .
- The run time mapping from virtual to physical address is done by a hardware device called memory management unit(MMU)

MMU


Relocation
register
logical
address
CPU
346
physical
address
14346
Memory
14000
MMU

- Simple MMU scheme is generalization of base register scheme.
- The base register is now called a relocation register.
- The value in relocation register is added to every address generated by user process at a time the address is sent to memory.
- The user program never sees the real physical address. The program creates a pointer to the location 346(in above example).
- The user program deals with logical addresses.

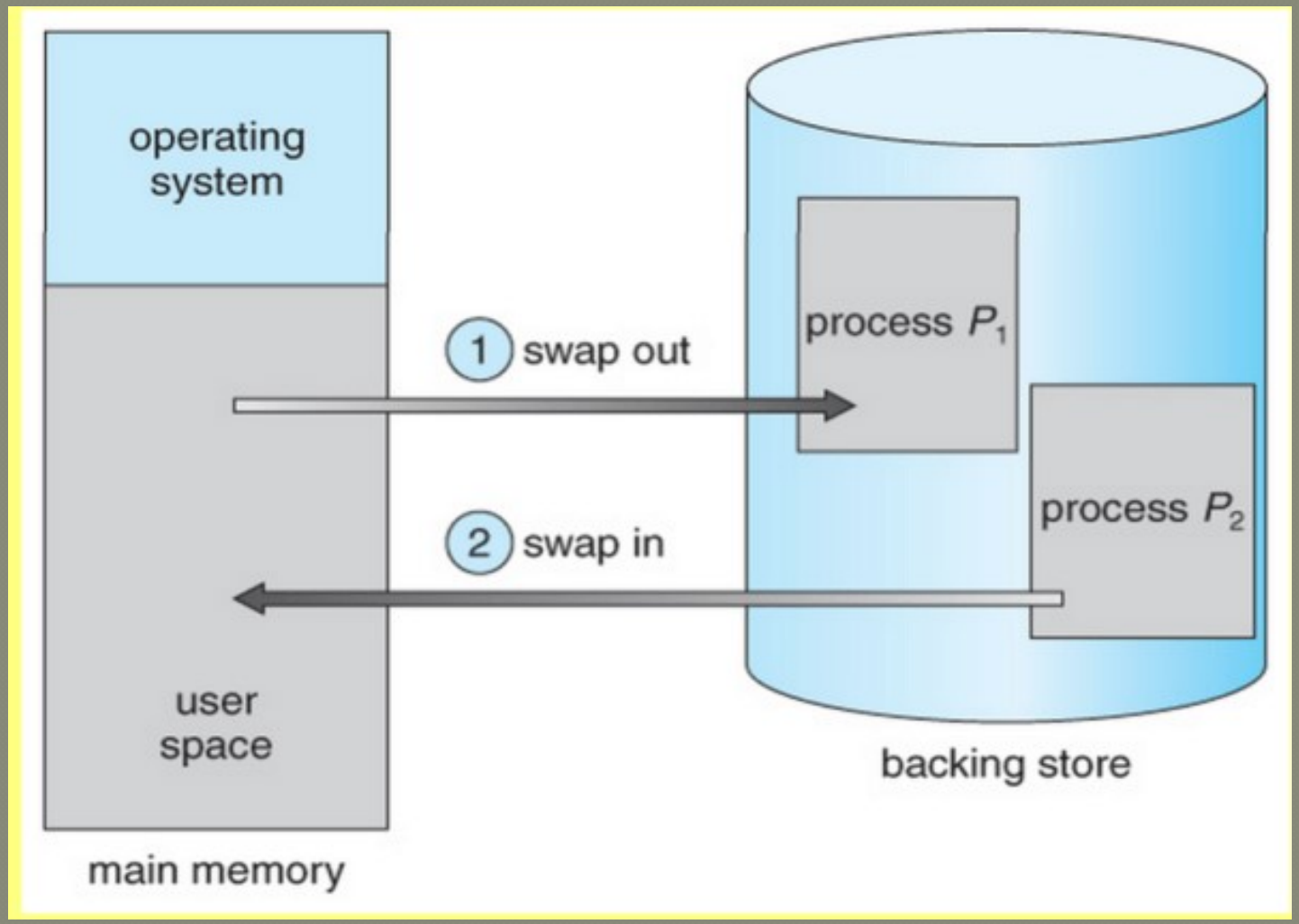
Swapping

- Moving processes from main memory to disk and back is called swapping.
- When a process is brought in memory it is called *swap in* and when a process is brought back .in disk called *swap out.*
- Swapping is usually employed in memory management system with contiguous allocation such as fixed and dynamically partitioned memory and segmentation.
- It is the work of swapper that which process will brought back in disk when there is no space in memory and a new process want to execute. Generally priority based scheduling is used by swapper to swap in and swap out processes.

- The priority given to each process may depend on process size or the time required for its execution.
- The process that has highest priority is put first by the swapper in memory and the process that has lowest priority is put out by the swapper from memory.
- The time required to swap out a process and then swap in another process is referred to as *swap time*.

- Swap time=time to swap out a process + time to swap in other process.
- Swap time highly depends on transfer rate and size of process. If process size is small and transfer rate is high then swap time will be very small.

swapping

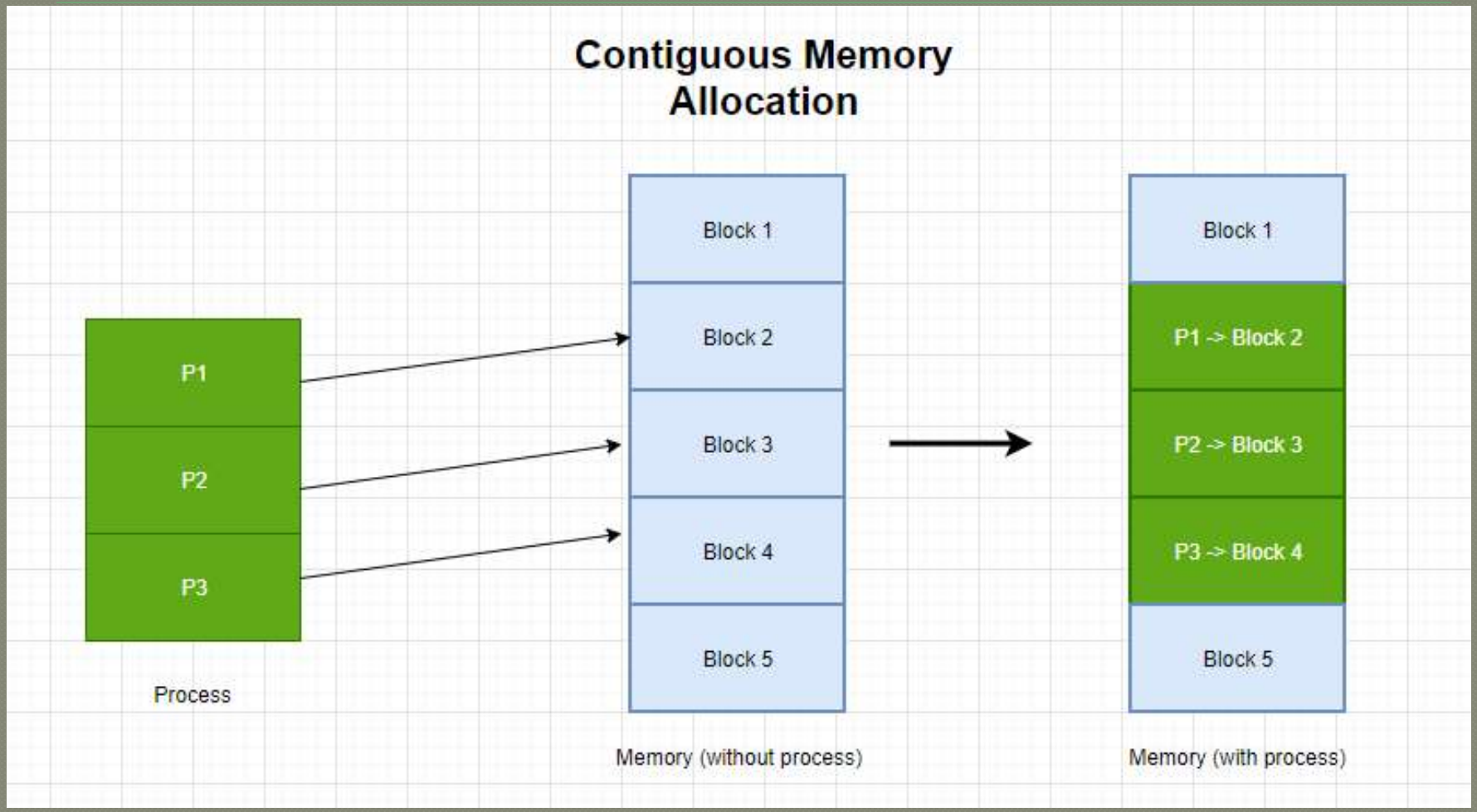

operating system
1 swap out
process P_1
process P_2
2 swap in
user space
backing store
main memory

Memory management techniques

- Contiguous memory Allocation
- Non-contiguous memory allocation

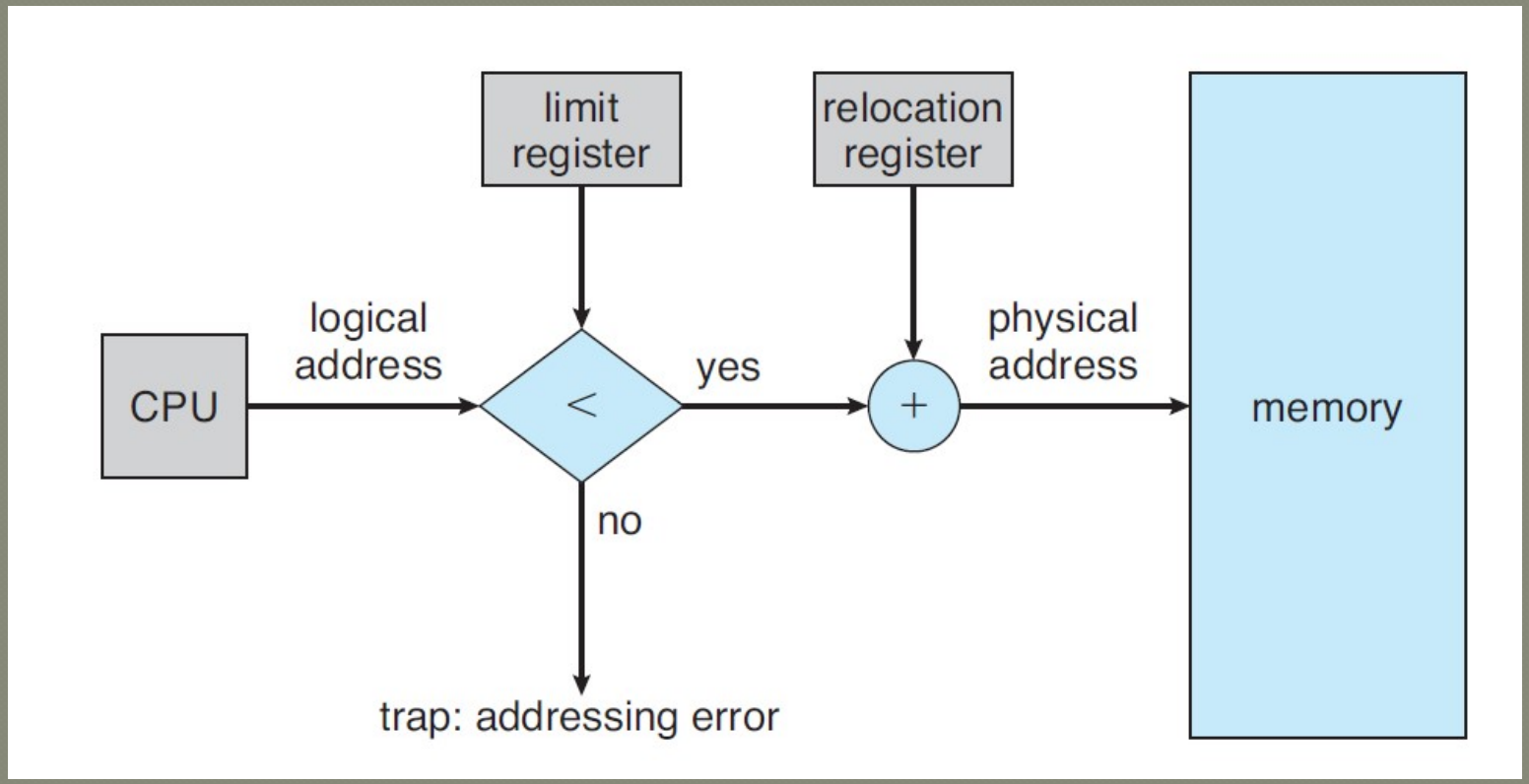
Contiguous Memory Allocation

- The memory is usually divided into two partition one for resident OS and one for the processes.
- In contiguous memory allocation each process is contained in a single section of memory that is contiguous to the section containing the next process.

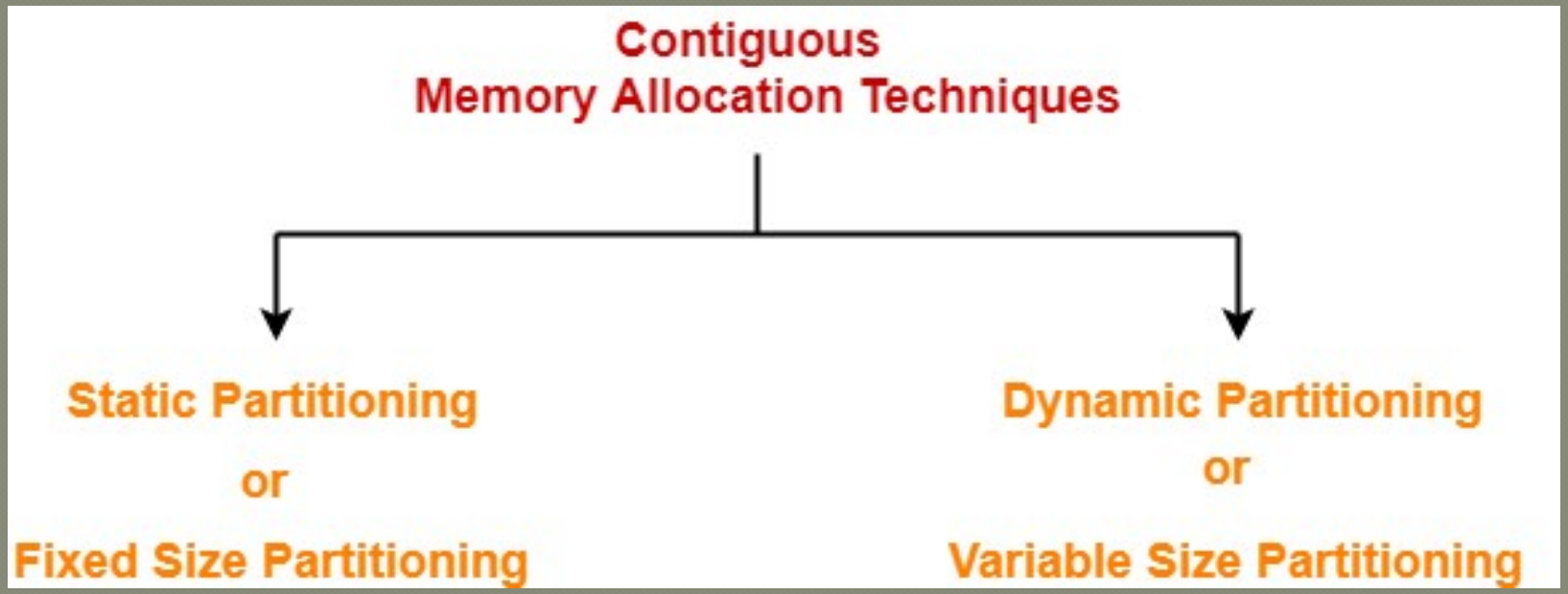

Contiguous Memory Allocation
P1
P2
P3
Process
Block 1
Block 2
Block 3
Block 4
Block 5
Memory (without process)
Block 1
P1 -> Block 2
P2 -> Block 3
P3 -> Block 4
Block 5
Memory (with process)

Memory Protection

The protection of operating system from user processes and protection of user processes from OS are the two main issues of memory protection.



Memory allocation



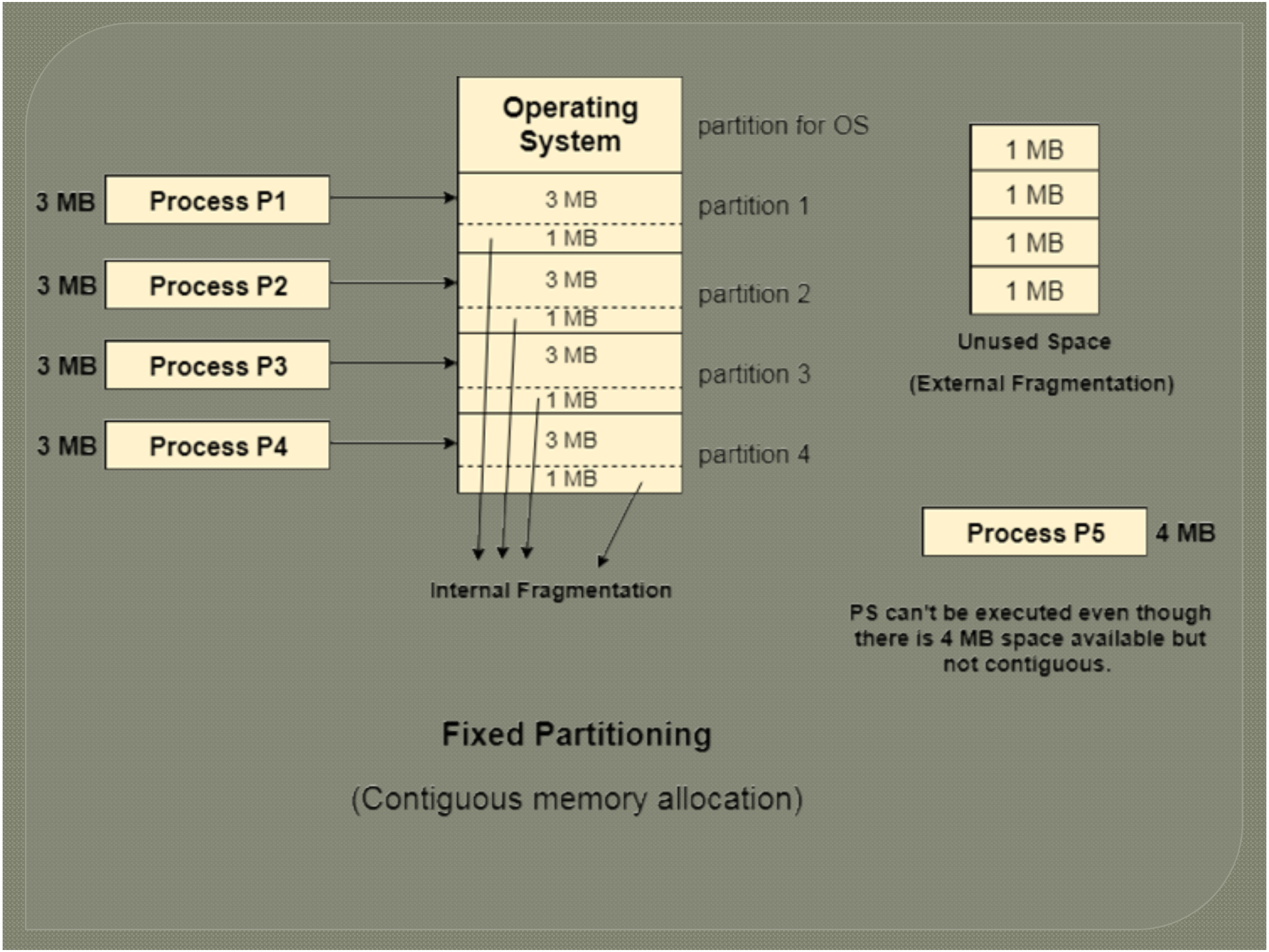
Fixed Partitioning Scheme

- The simplest method for allocating memory is to divide memory into several fixed sized partitions.
- Each partition may contain exactly one process.
- Thus the degree of multiprogramming is restricted by the number of partitions.
- In this multi-partition method , when partition is free, one process is selected from the input queue and is loaded into the free partition.
- When the process terminates the partition is available for another process.

- **There are various cons of using this technique.**

1. **Internal Fragmentation:** If the size of the process is lesser then the total size of the partition then some size of the partition get wasted and remain unused. This is wastage of the memory and called internal fragmentation.
   - As shown in the image below, the 4 MB partition is used to load only 3 MB process and the remaining 1 MB got wasted.

2. **External Fragmentation :** The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.
   - As shown in the image below, the remaining 1 MB space of each partition cannot be used as a unit to store a 4 MB process. Despite of the fact that the sufficient space is available to load the process, process will not be loaded.


Operating System
partition for OS
3 MB
Process P1
3 MB
partition 1
1 MB
3 MB
Process P2
3 MB
partition 2
1 MB
3 MB
Process P3
3 MB
partition 3
1 MB
3 MB
Process P4
3 MB
partition 4
1 MB
Internal Fragmentation
1 MB
1 MB
1 MB
1 MB
Unused Space
(External Fragmentation)
Process P5
4 MB
PS can't be executed even though there is 4 MB space available but not contiguous.
Fixed Partitioning
(Contiguous memory allocation)

3. Limitation on the size of the process

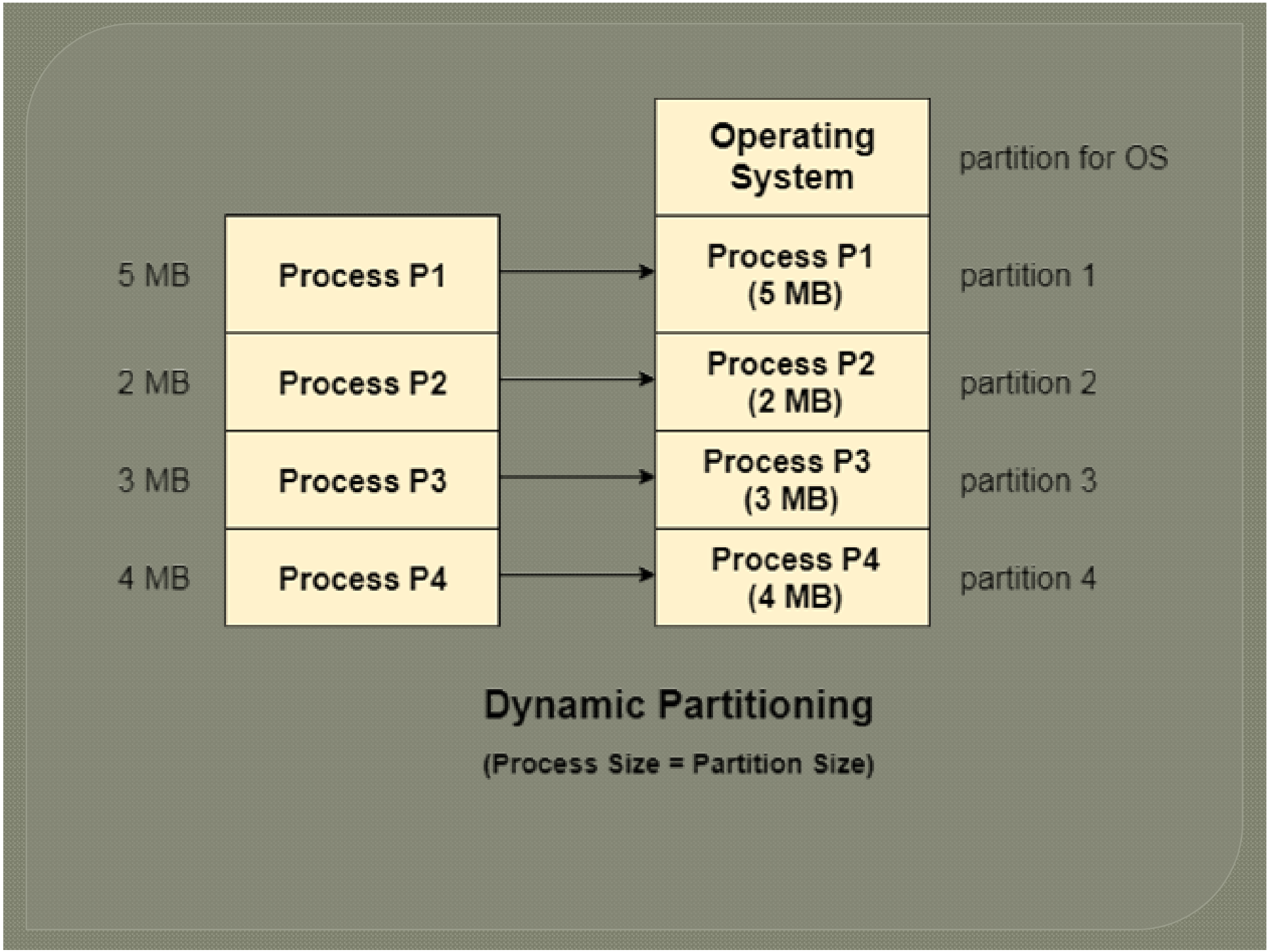
- If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

4. Degree of multiprogramming is less

- By Degree of multi programming, we simply mean the maximum number of processes that can be loaded into the memory at the same time. In fixed partitioning, the degree of multiprogramming is fixed and very less due to the fact that the size of the partition cannot be varied according to the size of processes.

Variable Partitioning scheme

- In this scheme the OS keeps a table indicating which part of memory is available and which is occupied.
- Initially , all memory is available for user processes and is considered one large block of available memory, a hole.
- Eventually , memory contains a set of holes of various sizes.
- 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.

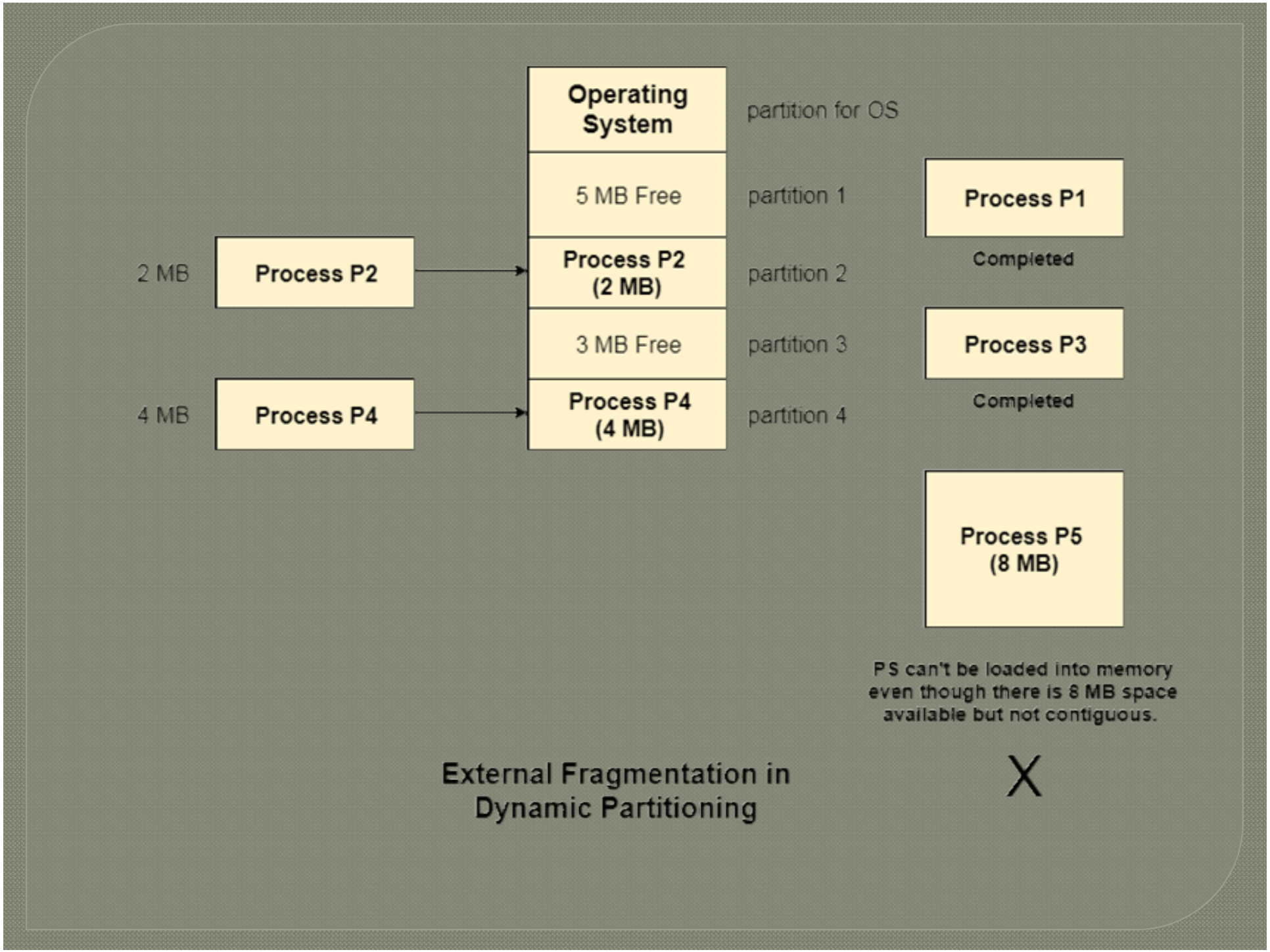

Operating System
partition for OS
5 MB
Process P1
Process P1 (5 MB)
partition 1
2 MB
Process P2
Process P2 (2 MB)
partition 2
3 MB
Process P3
Process P3 (3 MB)
partition 3
4 MB
Process P4
Process P4 (4 MB)
partition 4
Dynamic Partitioning
(Process Size = Partition Size)

Advantage

1. **No Internal Fragmentation**: Given the fact that the partitions in dynamic partitioning are created according to the need of the process, It is clear that there will not be any internal fragmentation because there will not be any unused remaining space in the partition.
2. **No Limitation on the size of the process**: In Fixed partitioning, the process with the size greater than the size of the largest partition could not be executed due to the lack of sufficient contiguous memory. Here, In Dynamic partitioning, the process size can't be restricted since the partition size is decided according to the process size.
3. **Degree of multiprogramming is dynamic**: Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time.

Disadvantage

- External Fragmentation: Absence of internal fragmentation doesn't mean that there will not be external fragmentation.
- Let's consider three processes P1 (1 MB) and P2 (3 MB) and P3 (1 MB) are being loaded in the respective partitions of the main memory.
- After some time P1 and P3 got completed and their assigned space is freed. Now there are two unused partitions (1 MB and 1 MB) available in the main memory but they cannot be used to load a 2 MB process in the memory since they are not contiguously located.
- The rule says that the process must be contiguously present in the main memory to get executed. We need to change this rule to avoid external fragmentation.


Operating System
partition for OS
5 MB Free
partition 1
Process P1
Completed
2 MB
Process P2
Process P2 (2 MB)
partition 2
3 MB Free
partition 3
Process P3
Completed
4 MB
Process P4
Process P4 (4 MB)
partition 4
Process P5 (8 MB)
PS can't be loaded into memory even though there is 8 MB space available but not contiguous.
External Fragmentation in Dynamic Partitioning
X

- Complex Memory Allocation: In Fixed partitioning, the list of partitions is made once and will never change but in dynamic partitioning, the allocation and deallocation is very complex since the partition size will be varied every time when it is assigned to a new process. OS has to keep track of all the partitions.
- Due to the fact that the allocation and deallocation are done very frequently in dynamic memory allocation and the partition size will be changed at each time, it is going to be very difficult for OS to manage everything.

Fregmentation

- As processes are loaded and removed from memory, the free memory space is broken into little pieces .
- It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

- There are two types of fragmentation
- Internal fragmentation
- External fragmentation

Allocation Strategies

1. First fit: Allocates the first hole, that is big enough.
2. Best fit: Allocates the smallest hole that is big enough.
3. Worst fit: Allocates the largest hole.

Request comes from the Process P4 (size 3kb)

OS
P1
<free> 10 KB
P2
<free> 16 KB
P3
<free> 4 KB

a. First fit algorithm allocates from the 10 KB block.
b. Best fit algorithm allocates from the 4 KB block.
c. Worst fit algorithm allocates from the 16 KB block.

New memory arrangements with respect to each algorithms will be as follows:

OS
P1
P4
<free> 7 KB
P2
<free> 16 KB
P3
<free> 4 KB

First Fit

OS
P1
<free> 10 KB
P2
<free> 16 KB
P3
P4
<free> 1 KB

Best Fit

OS
P1
<free> 10 KB
P2
P4
<free> 13 KB
P3
<free> 4 KB

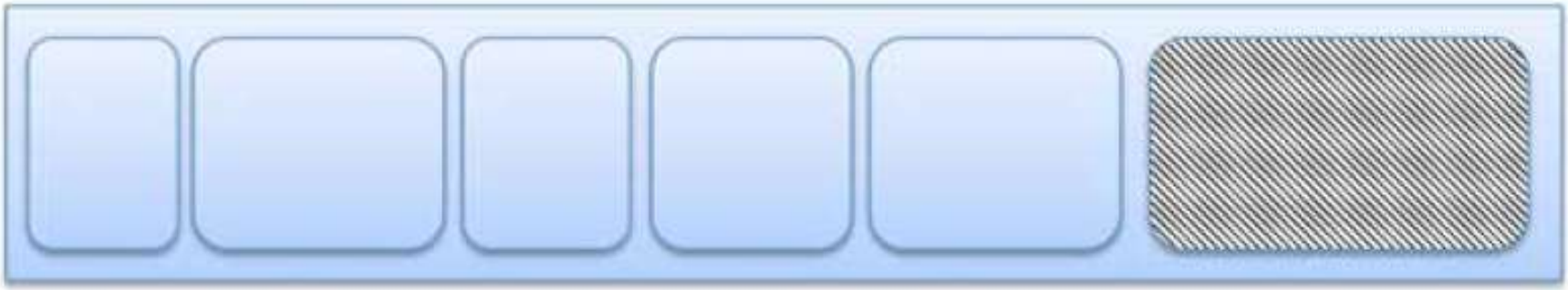
Worst Fit

Compaction

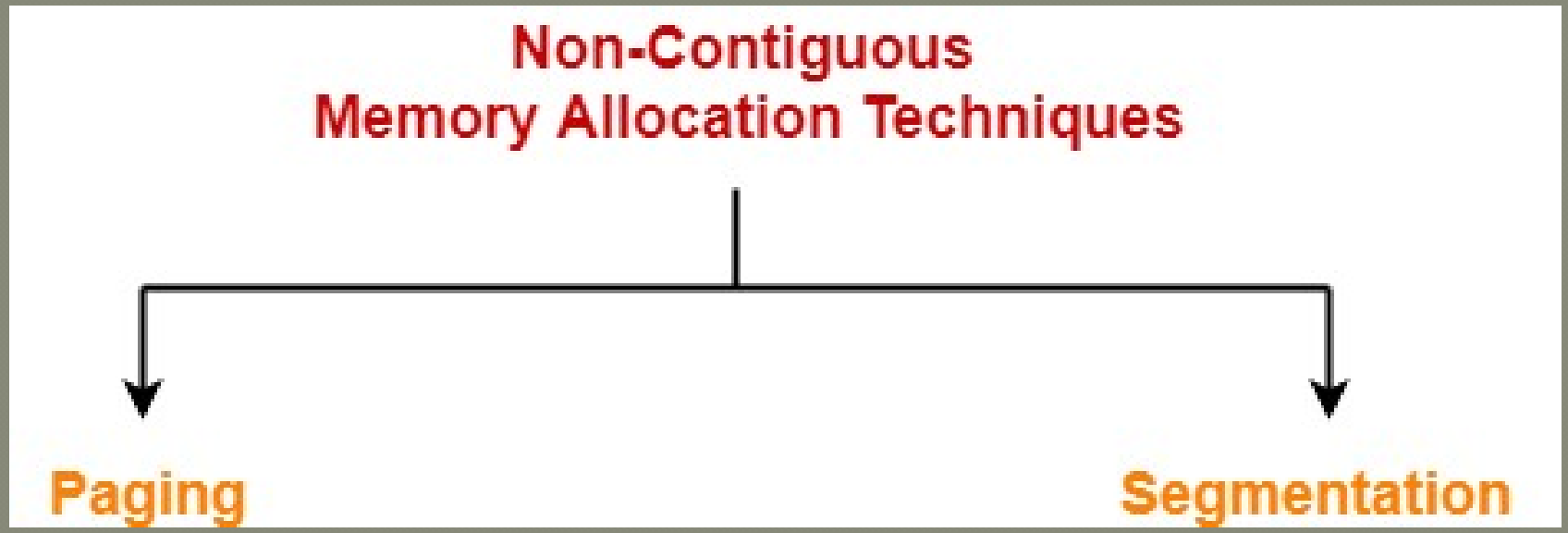
- One solution to the external fragmentation is compaction.
- The goal is to shuffle memory contents to place all the free memory memory together in one large block.
- Compaction is not always possible .If relocation is static and is done at load time then compaction is not possible.
- It is possible only if relocation is dynamic and is done at execution time.
- If addresses are relocated dynamically , relocation requires only moving the program and data then changing the base register to reflect the new base address.
- The simplest compaction algorithm is to move all the processes toward one end of memory ,producing one large block of available memory.
- This scheme can be expensive.

Fragmented memory before compaction

Memory after compaction



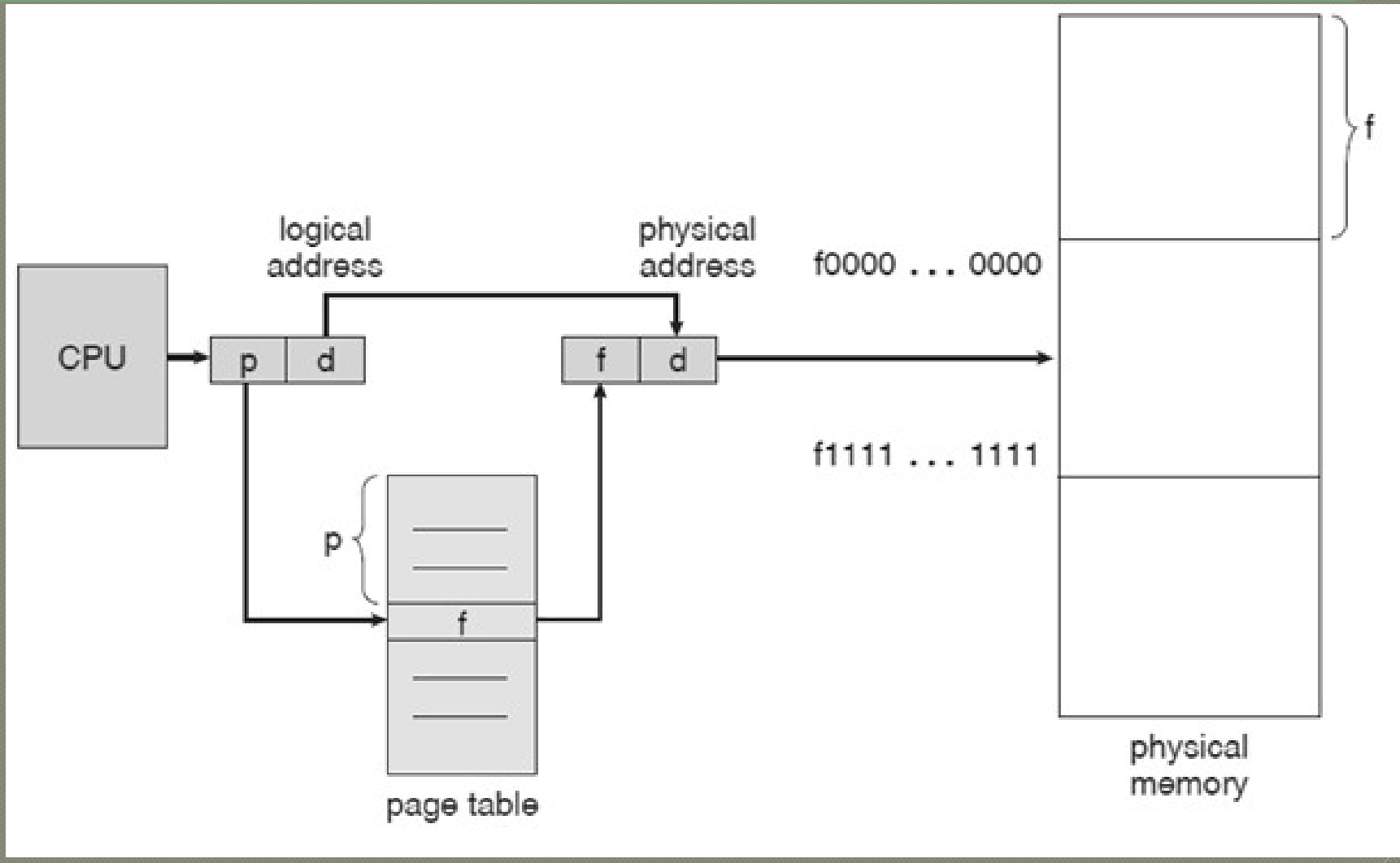
Non contiguous memory allocation



Paging

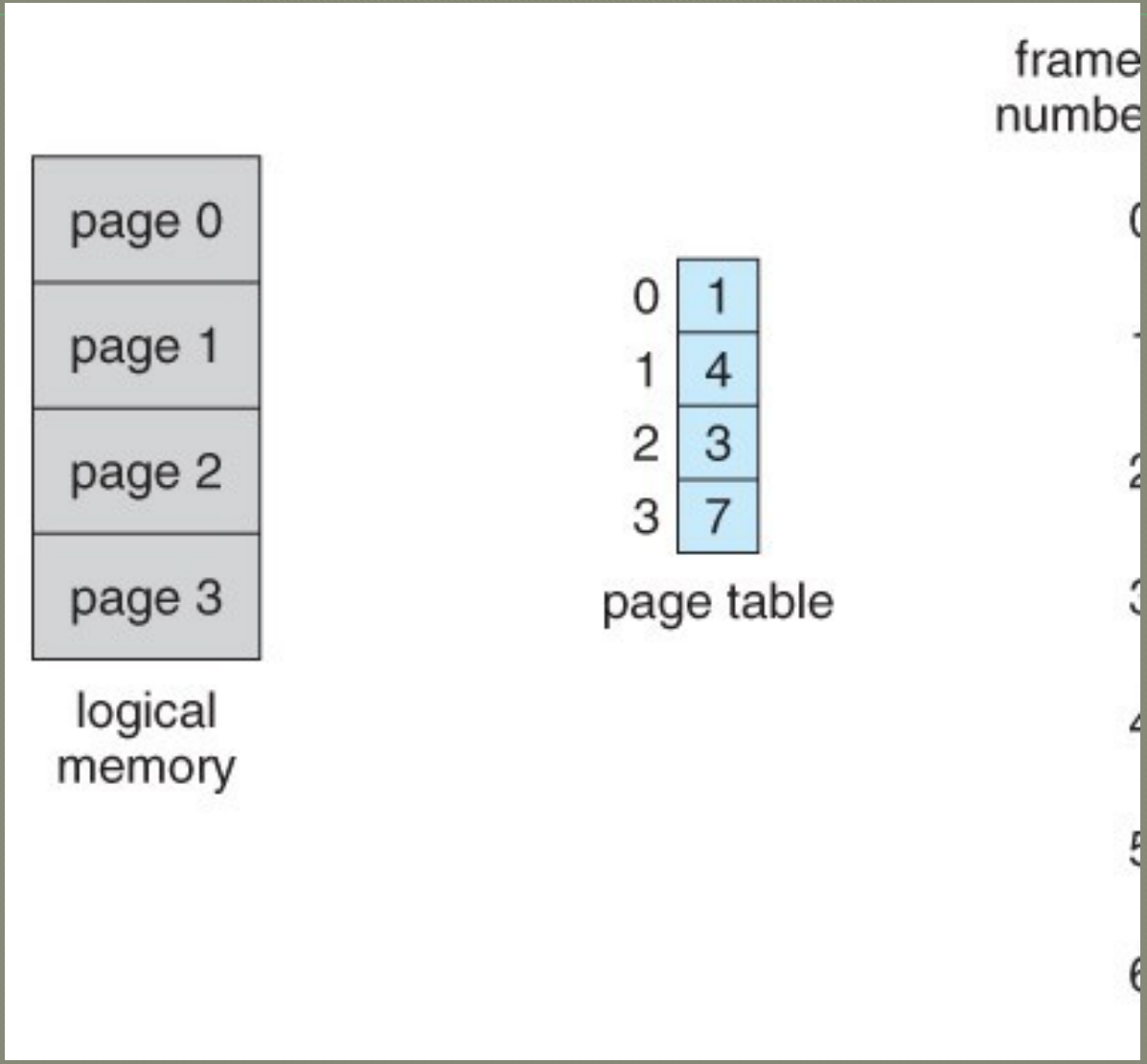
- Paging is a memory-management scheme that permits the physical address space of a process to be noncontiguous. Paging avoids external fragmentation and the need for compaction.
- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or the backing store).

Hardware support for paging



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

Page model

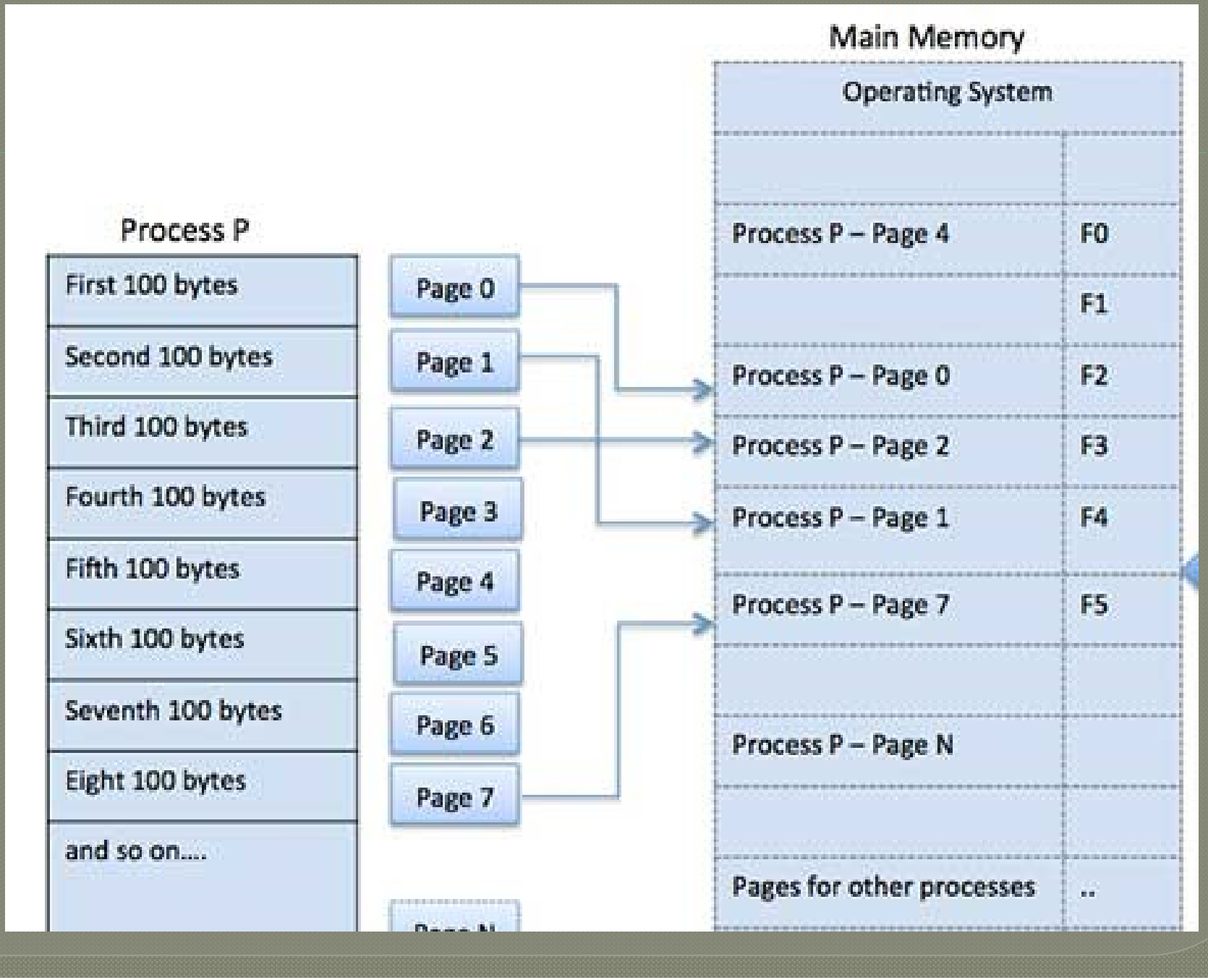

page 0
page 1
page 2
page 3
logical memory
0 1
1 4
2 3
3 7
page table
frame numbe

- The page size (like the frame size) is defined by the hardware.
- The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture.
- The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.
- If the size of the logical address space is 2m, and a page size is 271 addressing units (bytes or words then the high-order m- n bits of a logical address designate the page number, and the n low-order bits designate the page offset. Thus, the logical address is as follows:

Logical address

where p is an index into the page table and d is the displacement within the page.

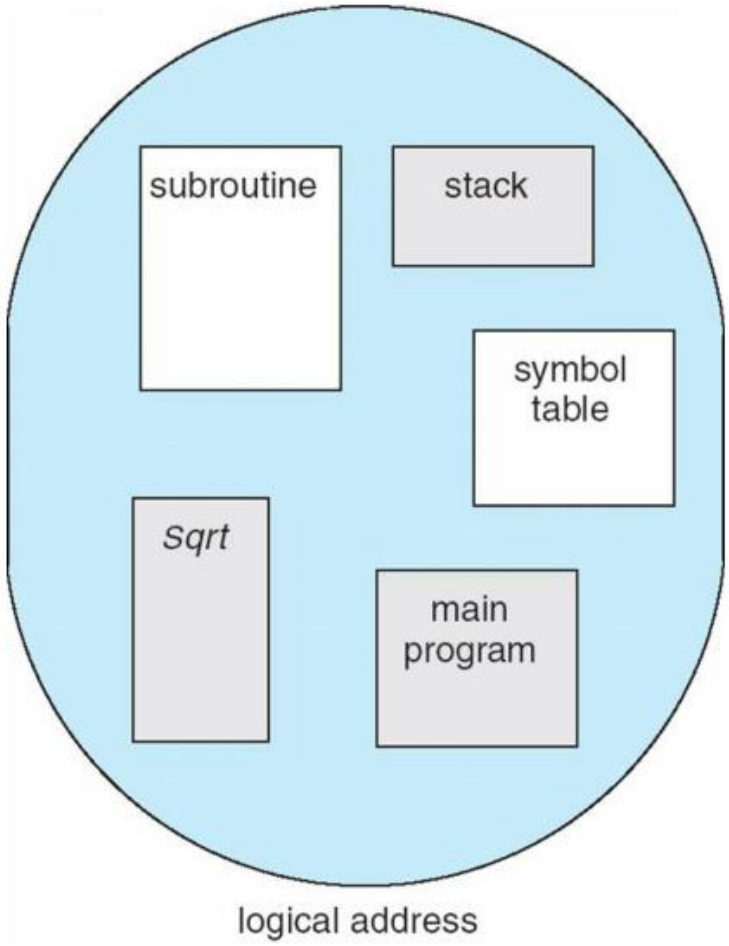
You may have noticed that paging itself is a form of dynamic relocation. Every logical address is bound by the paging hardware to some physical address. Using paging is similar to using a table of base (or relocation) registers, one for each frame of memory.


Main Memory
Operating System
Process P
First 100 bytes
Second 100 bytes
Third 100 bytes
Fourth 100 bytes
Fifth 100 bytes
Sixth 100 bytes
Seventh 100 bytes
Eight 100 bytes
and so on....
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7
Process P – Page 4
F0
F1
Process P – Page 0
F2
Process P – Page 2
F3
Process P – Page 1
F4
Process P – Page 7
F5
Process P – Page N
Pages for other processes
..

Advantages and Disadvantages of Paging

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

User's View of a Program



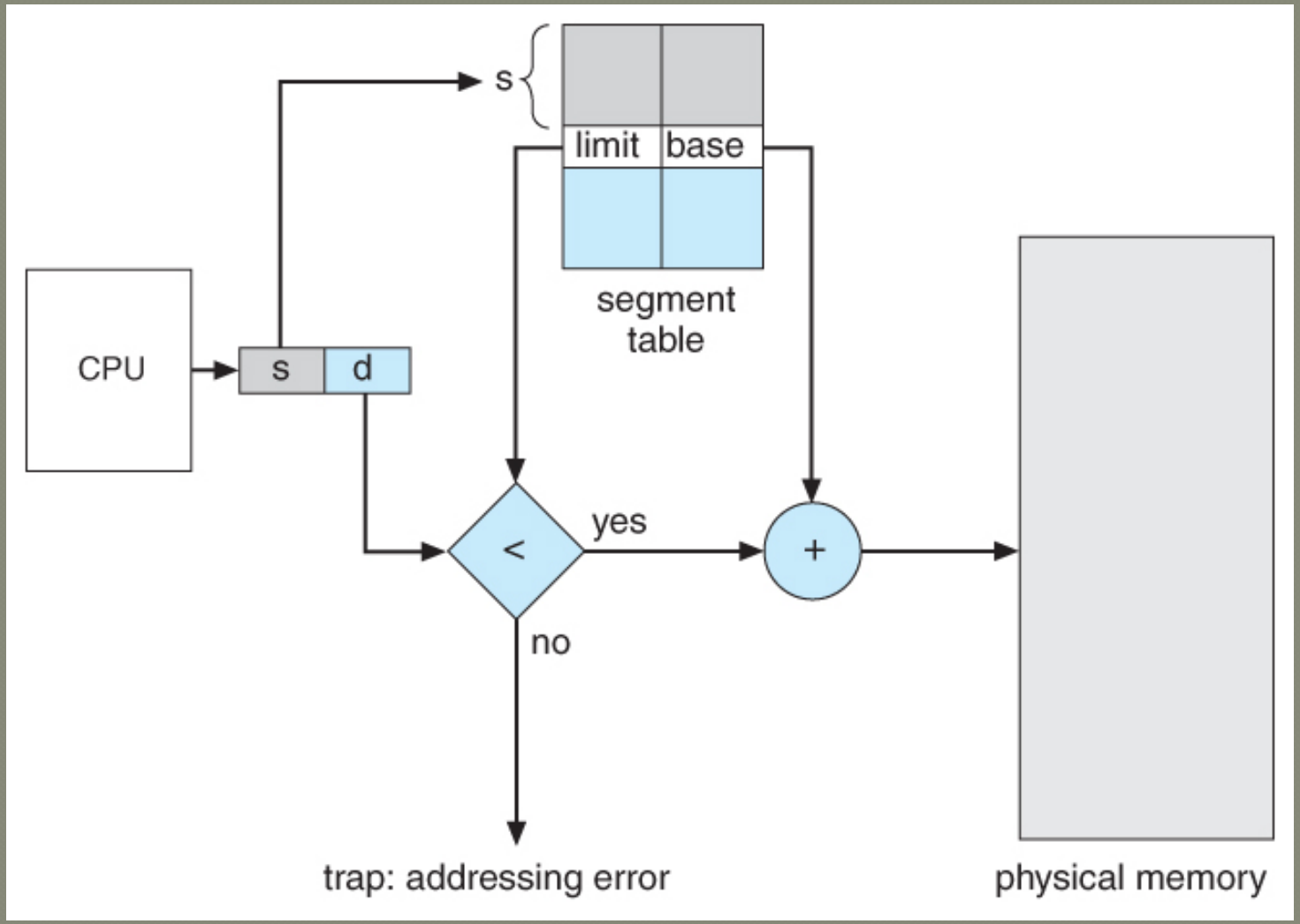
Segmentation

- Segmentation is a memory-management scheme that supports user view of memory.
- A logical address space is a collection of segments.
- Each segment has a name and a length.
- The addresses specify both the segment name and the offset within the segment.
- The user therefore specifies each address by two quantities: a segment name and an offset. (Contrast this scheme with the paging scheme, in which the user specifies only a single address, which is partitioned by the hardware into a page number and an offset, all invisible to the programmer.) For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name.

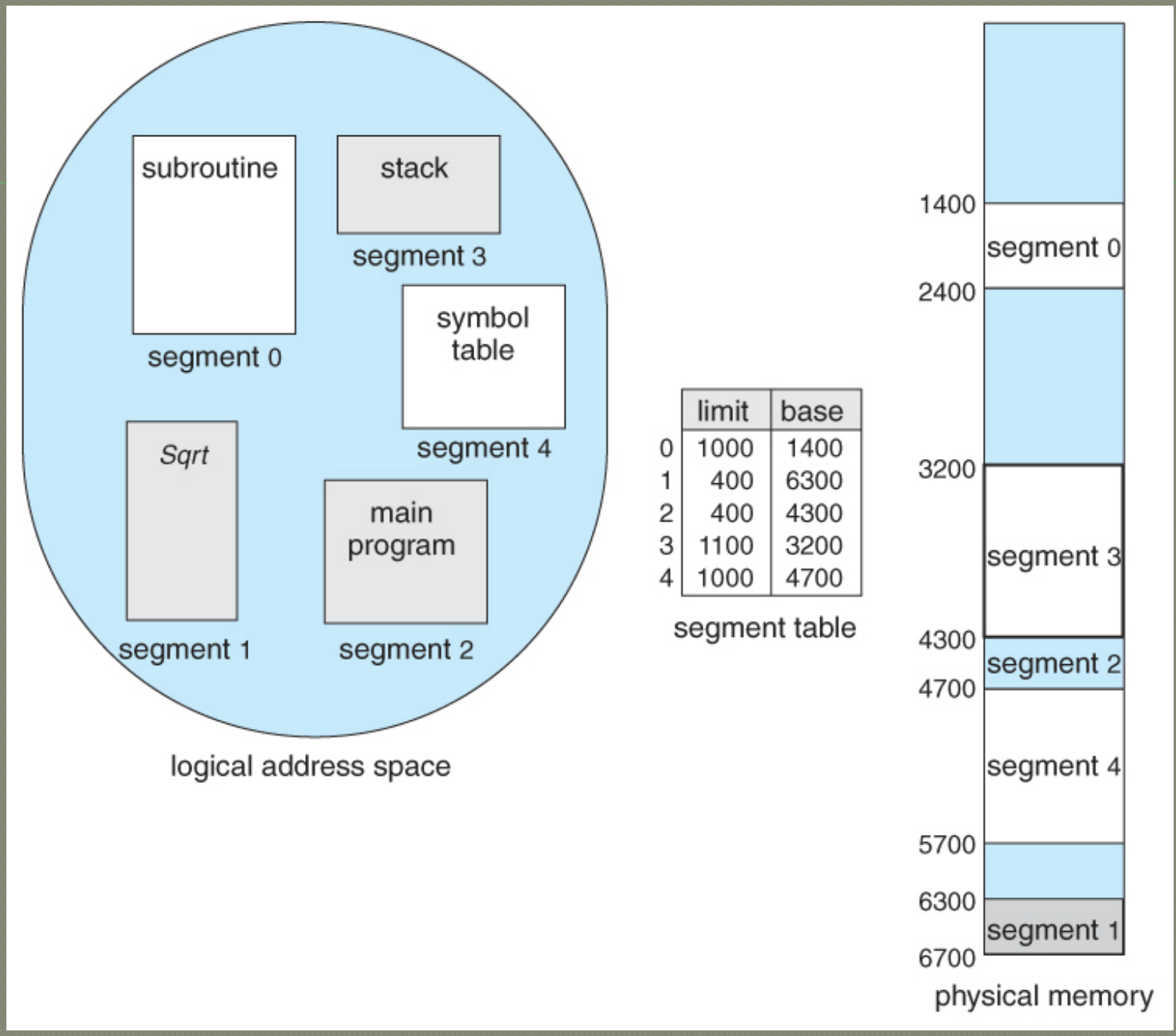
- Thus, a logical address consists of a two tuple :
- <segment number, offset>
- Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program. C compiler might create separate segments for the following:
- The code
- Global variables
- The heap, from which memory is allocated
- The stacks used by each thread
- The standard C library
- Libraries that are linked in during compile time might be assigned separate segments. The loader would take all these segments and assign them segment numbers.

- Although the user can now refer to objects in the program by a two-dimensional address, the actual physical memory is still, of course, a one-dimensional sequence of bytes.
- Thus, we must define an implementation to map two dimensional user-defined addresses into one-dimensional physical addresses. This mapping is effected by a segment table.
- Each entry in the segment table has a segment base and a segment limit.
- The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.

Segmentation hardware



- The use of a segment table is illustrated in Figure- A logical address consists of two parts: a segment number, s, and an offset into that segment, d.
- The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment).
- When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs.


subroutine
stack
segment 3
segment 0
symbol table
segment 4
Sqrt
main program
segment 1
segment 2
logical address space
limit
base
0 1000 1400
1 400 6300
2 400 4300
3 1100 3200
4 1000 4700
segment table
1400
segment 0
2400
3200
segment 3
4300
segment 2
4700
segment 4
5700
6300
segment 1
6700
physical memory

Virtual memory