Module - 4

By Dr. Renuka S. Gound

Module-4

- Memory Management Swapping,
- Logical versus Physical Address Space, Contiguous Allocation,
- Paging - Basic Method, Hardware Support, Protection; Structure of Page Table Hierarchical Paging,
- Segmentation Basic Method, Segmentation Hardware.
- Virtual Memory: Demand Paging; Page Replacement Basics,
- Algorithms - FIFO, Optimal, LRU, Thrashing Causes of Thrashing.

Partition Allocation Methods in Memory Management

Memory management techniques

- Single contiguous allocation: Simplest allocation method used by MS-DOS. All memory (except some reserved for OS) is available to a process.
- Partitioned allocation: Memory is divided into different blocks or partitions. Each process is allocated according to the requirement.
- Paged memory management: Memory is divided into fixed-sized units called page frames, used in a virtual memory environment.

Memory Allocation

- One of the simplest methods of allocating memory is to assign processes to variably sized partitions in memory, where each partition may contain exactly one process.
- In this variable partition scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied.
- Initially, all memory is available for user processes and is considered one large block of available memory, a hole.
- Eventually, as you will see, memory contains a set of holes of various sizes

Partitioned allocation:

- In Partition Allocation, when there is more than one partition freely available to accommodate a process's request, a partition must be selected.
- To choose a particular partition, a partition allocation method is needed.
- A partition allocation method is considered better if it avoids internal fragmentation.
- When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.

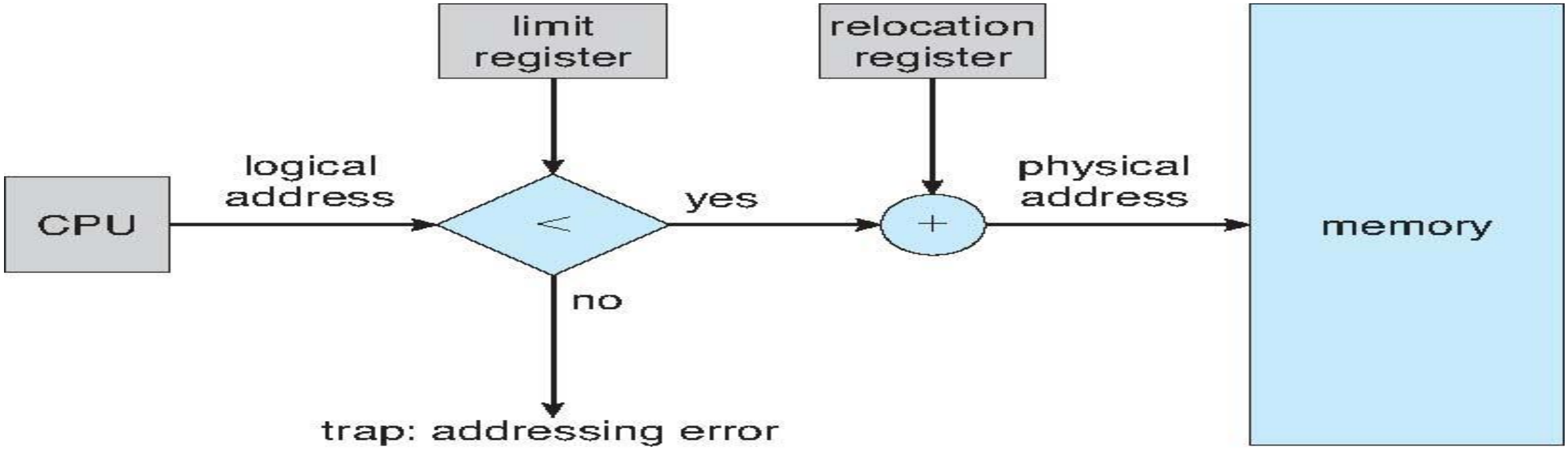
Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory

Contiguous Allocation (Cont.)

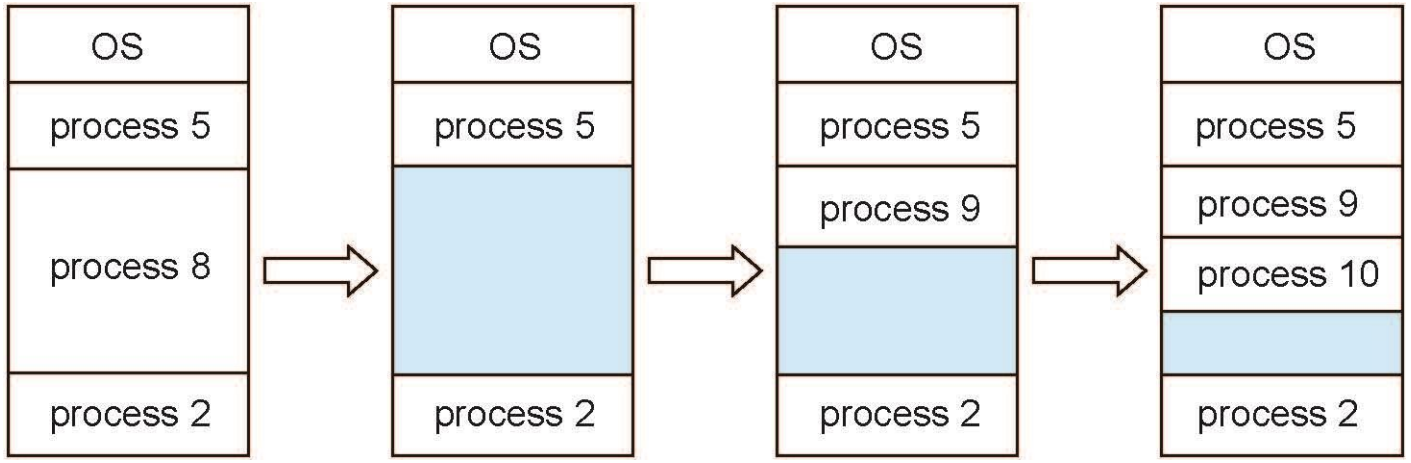
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*
  - Can then allow actions such as kernel code being **transient** and kernel changing size

Hardware Support for Relocation and Limit Registers



Multiple-partition allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **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
  - Process exiting frees its partition, adjacent free partitions combined
  - 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

Placement Algorithms

There are different Placement Algorithm:

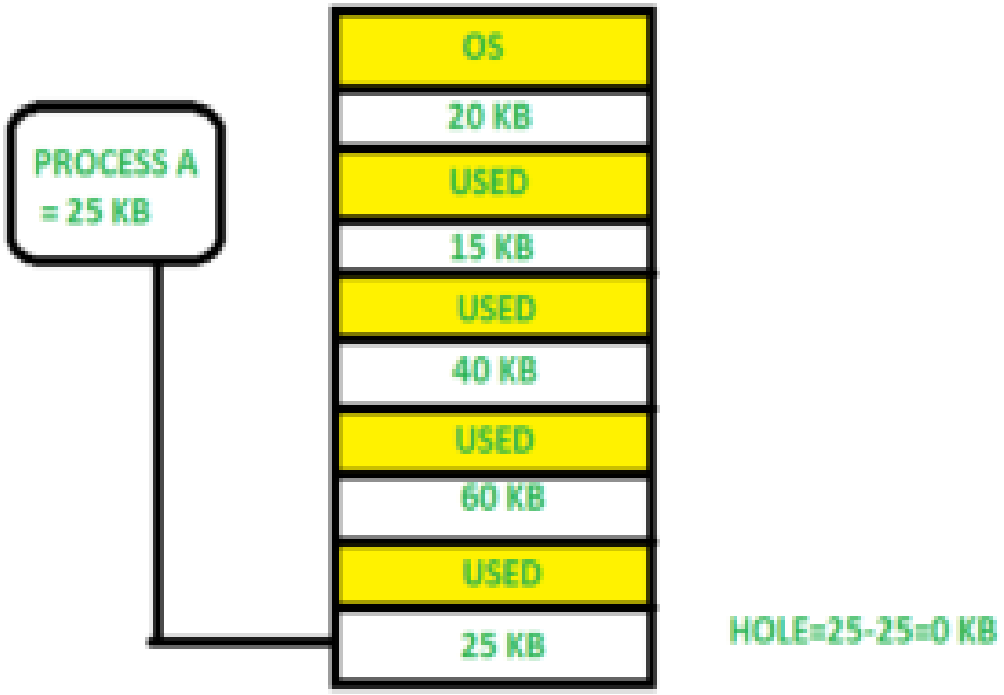
- A. First Fit
- B. Best Fit
- C. Worst Fit

First Fit

Allocate the first hole that is big enough.
Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended.
We can stop searching as soon as we find a free hole that is large enough.

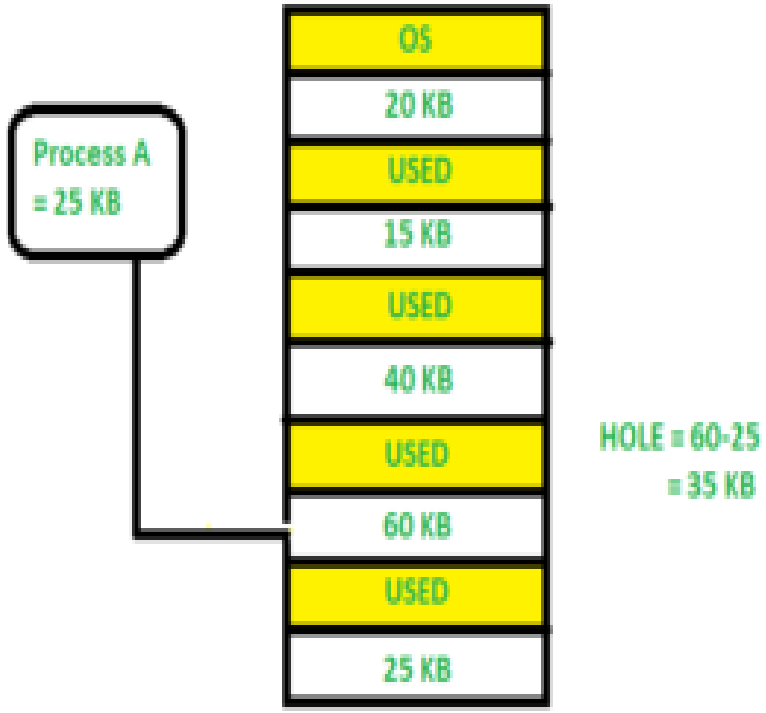
Best Fit

- Allocate the process to the partition which is the first smallest sufficient partition among the free available partition.
- It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



Worst Fit

- Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory.
- It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.



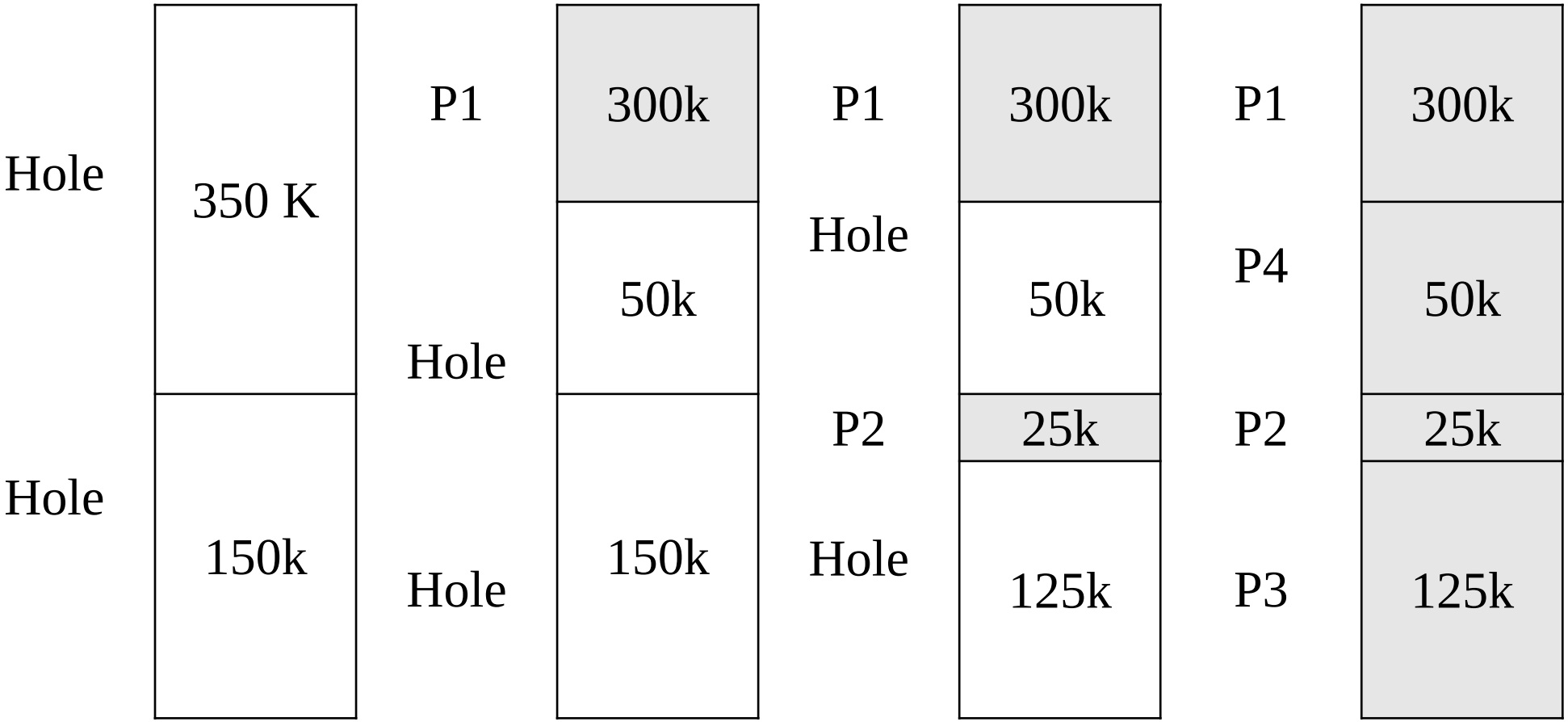
Example

Exercise: Consider the requests from processes in given order 300K, 25K, 125K, and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K.

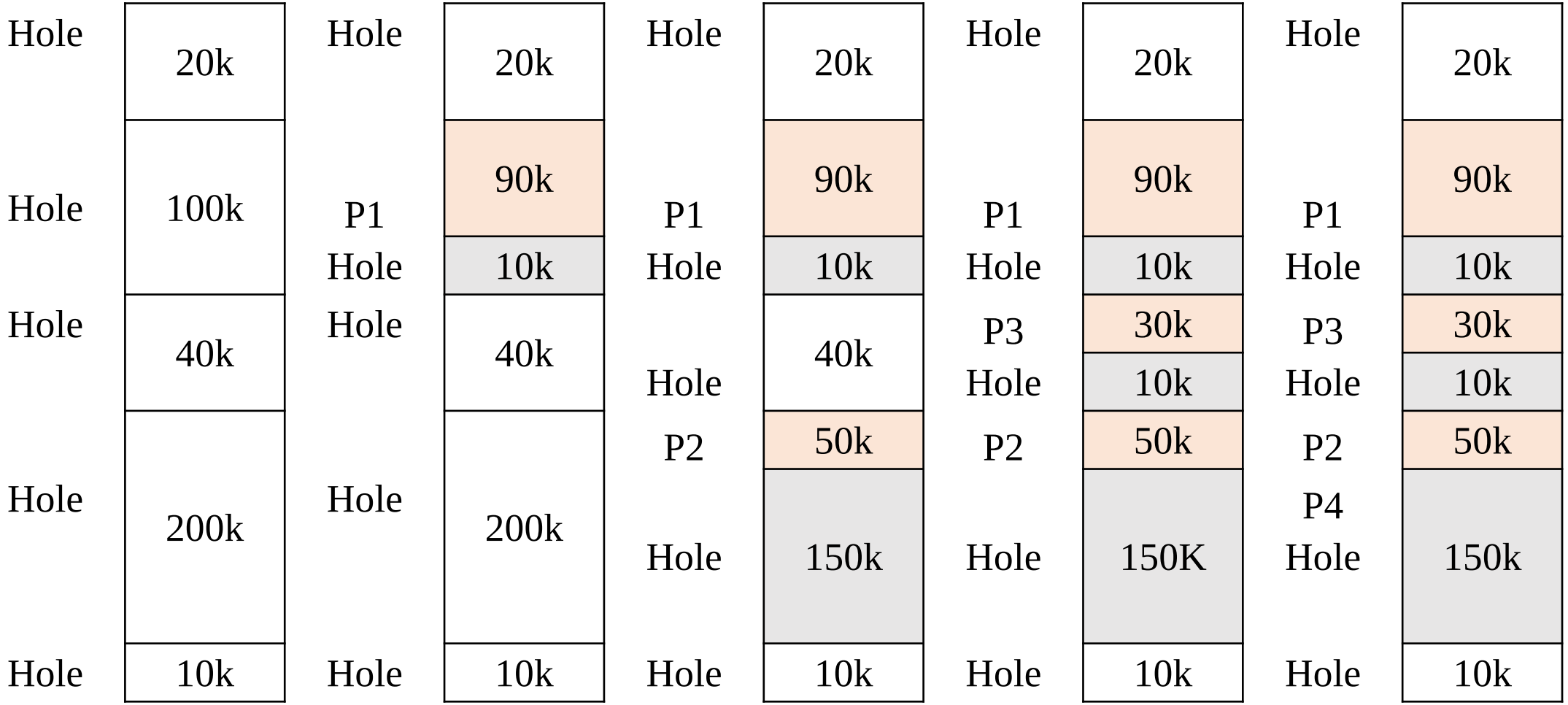
First Fit:

- 300K request is allocated from 350K block, 50K is left out.
- 25K is allocated from the 150K block, 125K is left out.
- Then 125K and 50K are allocated to the remaining left out partitions.
- So, the first fit can handle requests.

First Fit


Hole
350 K
Hole
150k
P1
300k
Hole
50k
Hole
150k
P1
300k
Hole
50k
P2
25k
Hole
125k
P1
300k
P4
50k
P2
25k
P3
125k

Process={90K,50K,30K,40k}
Memory Blocks={20k, 100k, 40k, 200k,10k}
The first fit can handle requests or not…..



40k is not allocated as free blocks are not Available

Best Fit

- The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.
- Advantage
- Memory utilization is much better than first fit as it searches the smallest free partition first available.
- Disadvantage
- It is slower and may even tend to fill up memory with tiny useless holes.

External fragmentation

- The first-fit for memory allocation suffer from external fragmentation. As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous: storage is fragmented into a large number of small holes.

Fragmentation

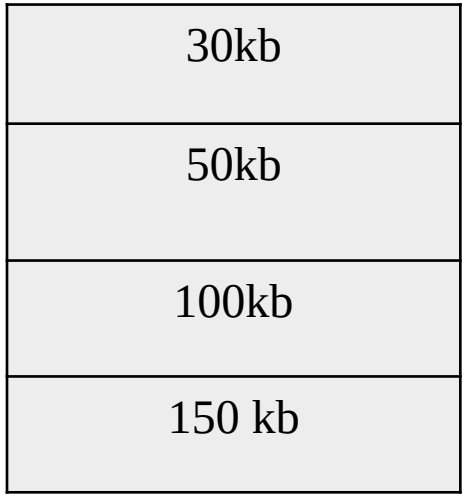
- External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous
- Internal Fragmentation – allocated memory may be slightly larger than requested memory, unused memory that is internal to a partition

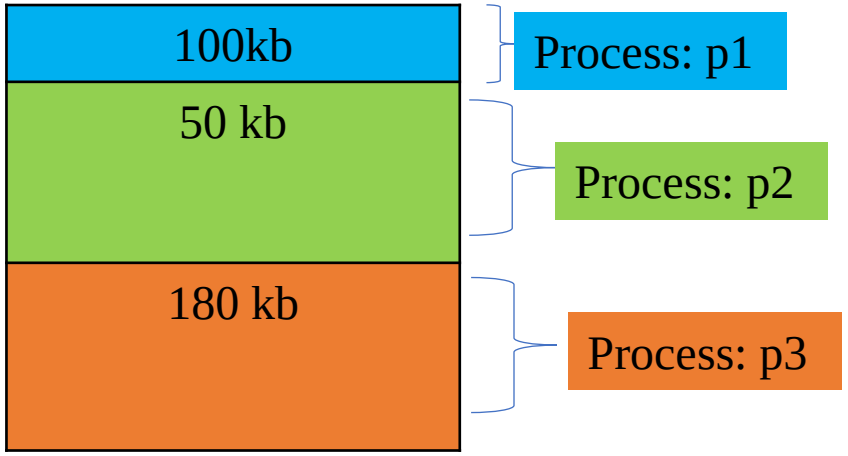
One solution to the problem of external fragmentation is compaction. The goal is to shuffle the memory contents so as to place all free memory together in one large block.

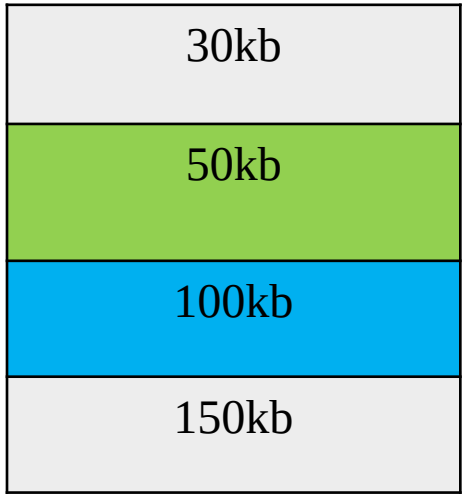
Physical Memory is divided into Variable sized blocks called partitions

Logical memory is divided into variable sized processes

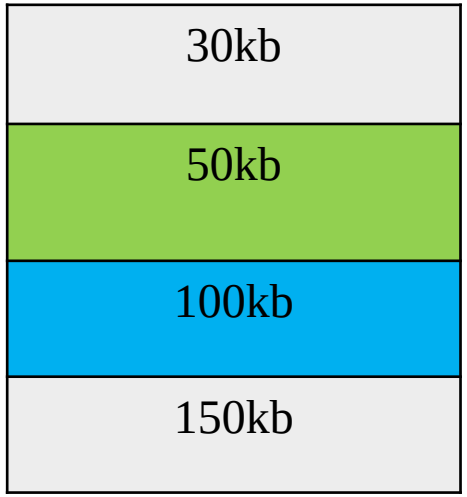
Partitions



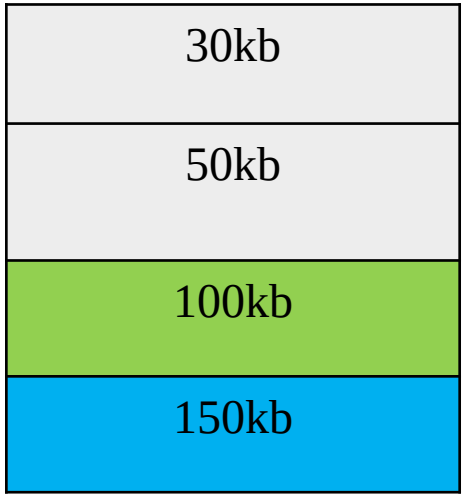




First Fit



Best Fit

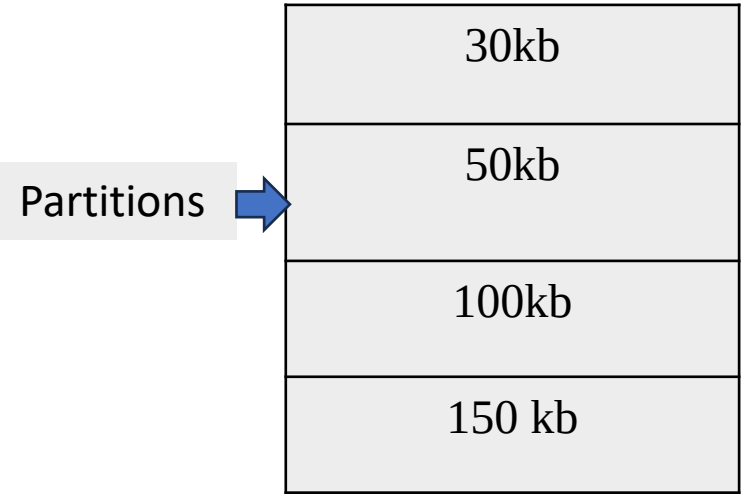


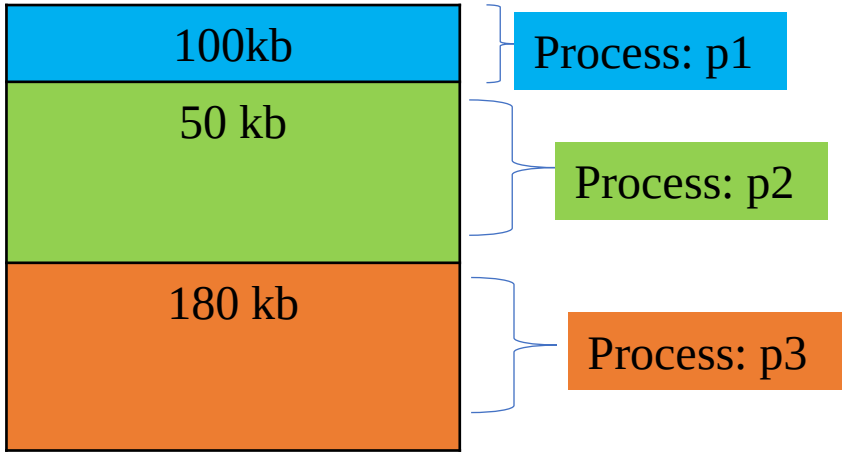
Worst Fit

Contiguous Allocation

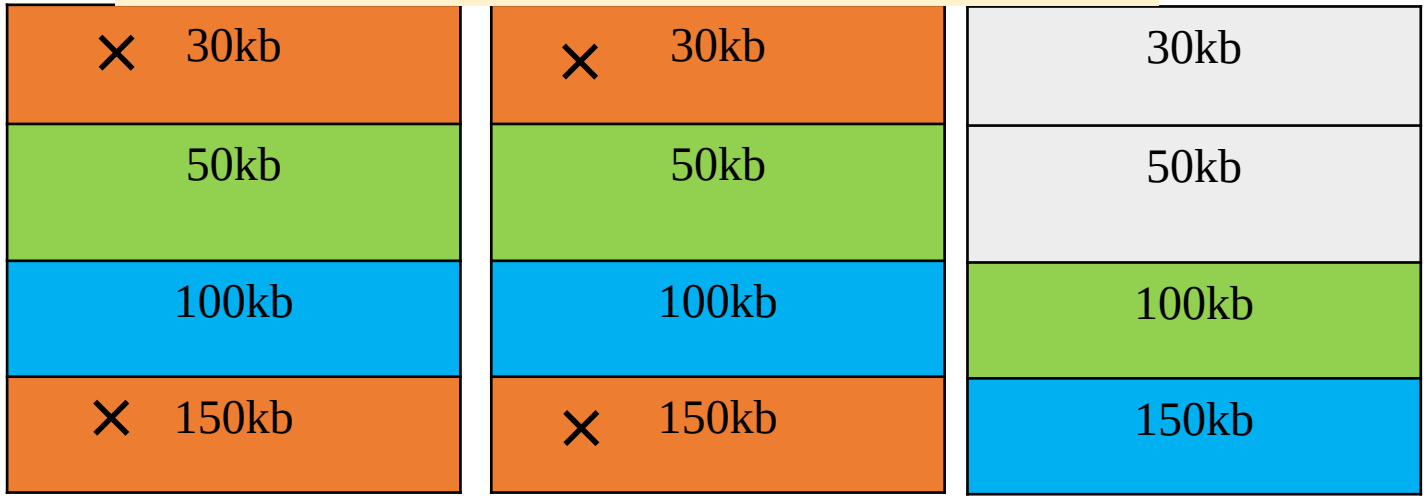
Physical Memory is divided into Variable sized blocks called partitions

Logical memory is divided into variable sized processes





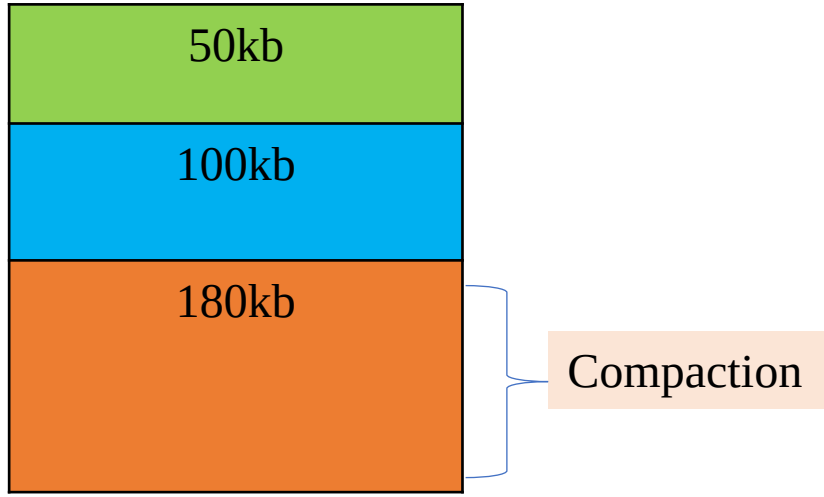
Contiguous Allocation



First Fit

Best Fit

Worst Fit



First Fit/Best Fit

Paging in Operating System

- Memory management discussed thus far has required the physical address space of a process to be contiguous.
- Paging, is a memory management scheme that permits a process's physical address space to be non-contiguous.
- Paging avoids external fragmentation and the associated need for compaction, two problems that plague contiguous memory allocation. \
- Because it offers numerous advantages, paging in its various forms is used in most operating systems, from those for large servers through those for mobile devices.
- Paging is implemented through cooperation between the operating system and the computer hardware.

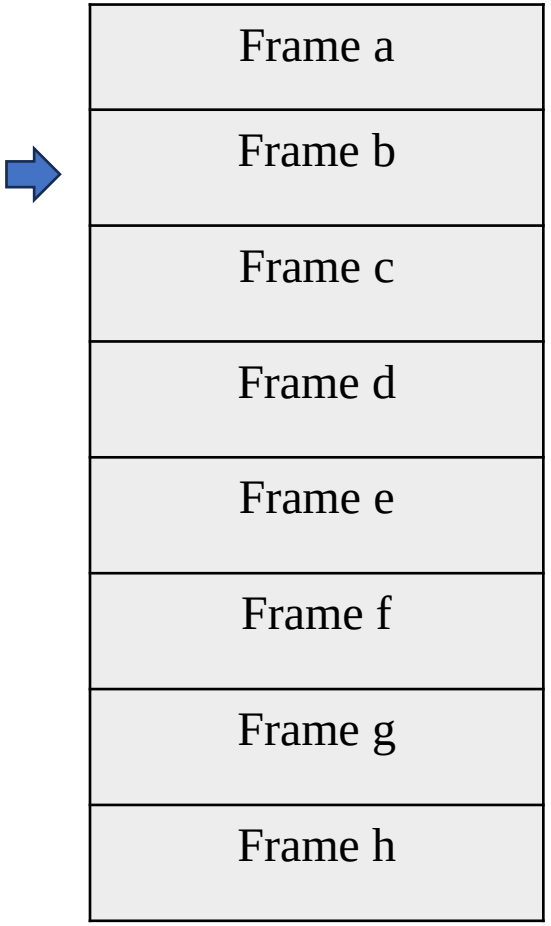
Basic Method

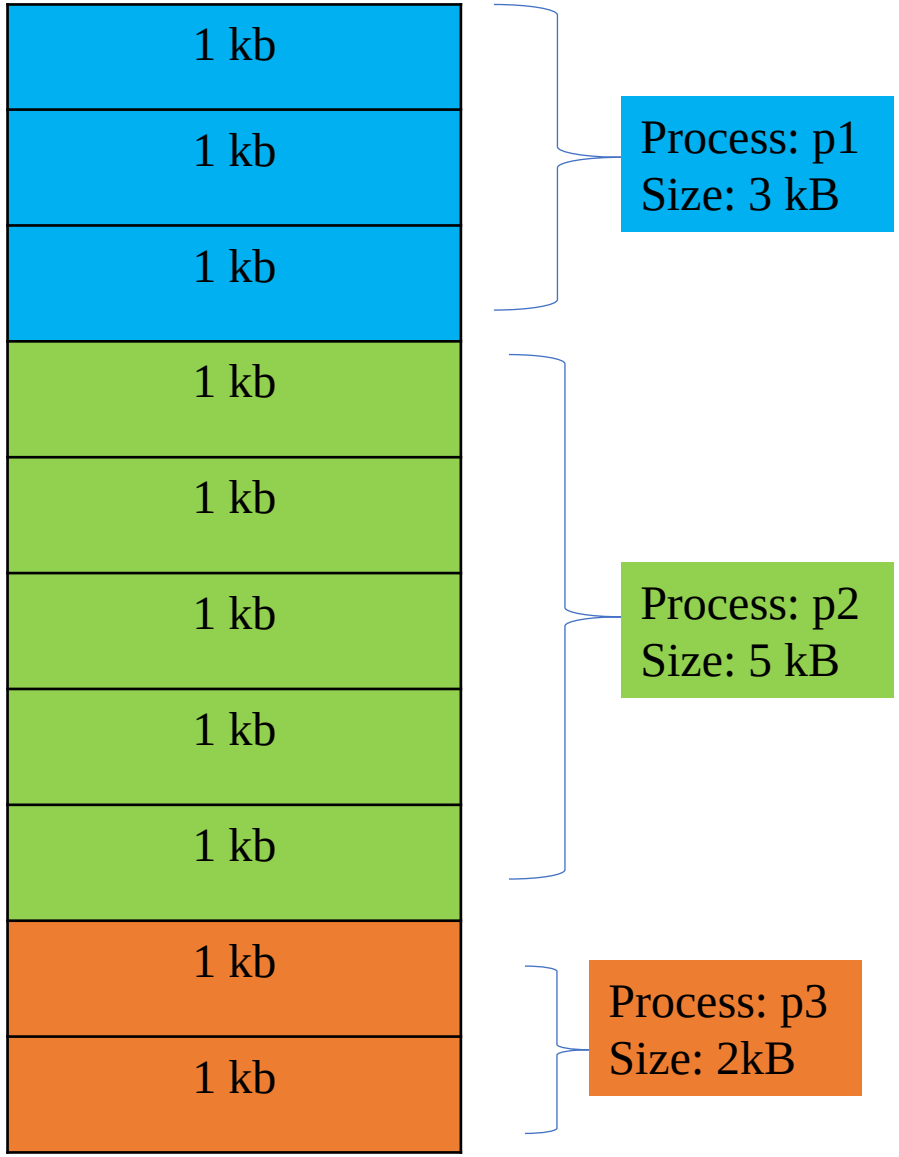
- The basic method for implementing paging involves
  1. **Breaking physical memory into fixed-sized blocks called frames**
  2. **Breaking logical memory into blocks of the same size called pages.**
- When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or the backing store).
- The backing store is divided into fixed-sized blocks that are the same size as the memory frames or clusters of multiple frames.
- For example, the logical address space is now totally separate from the physical address space, so a process can have a logical 64-bit address space even though the system has less than 264 bytes of physical memory.

Physical Memory is divided into fixed-sized blocks called frames

Logical memory is divided into blocks of the same size called pages.

Frames of 1kb each

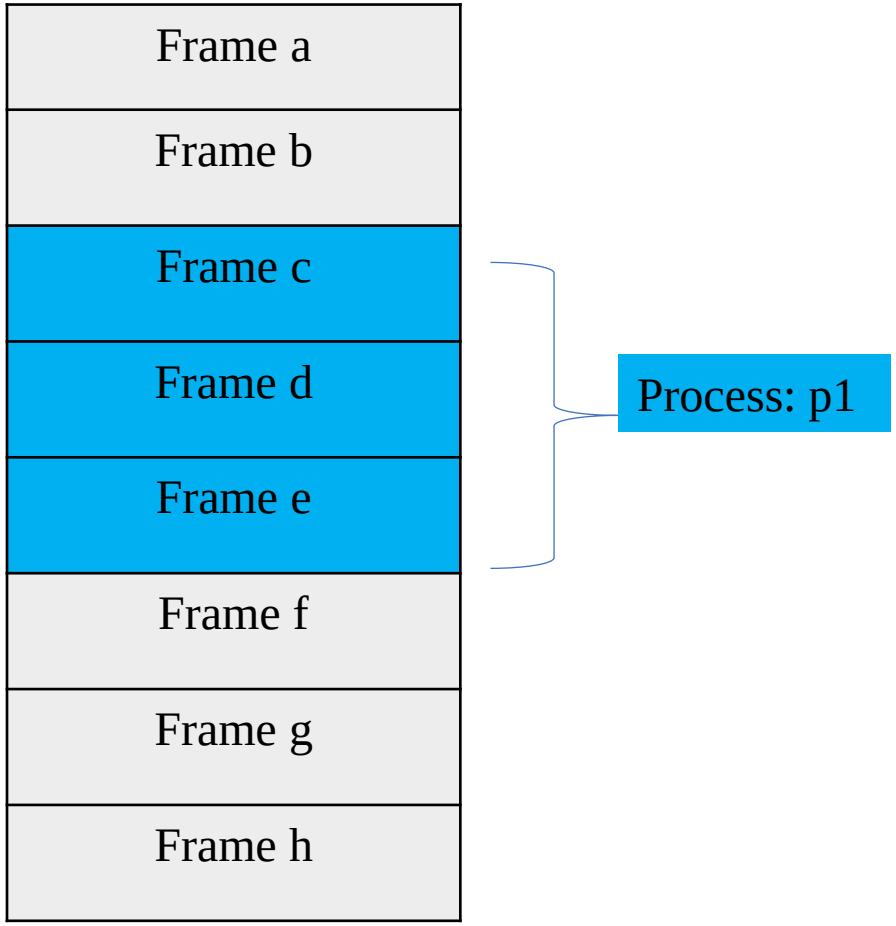


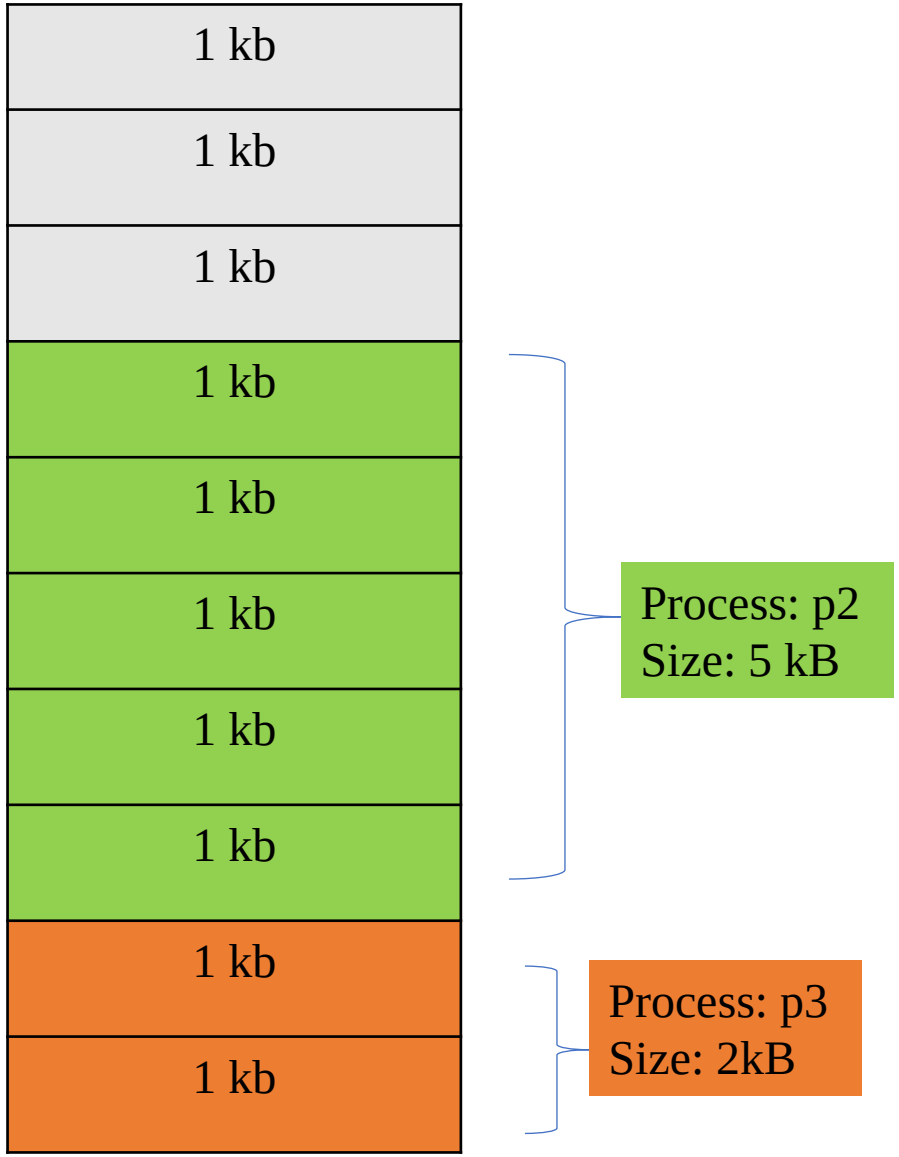


Physical Memory is divided into fixed-sized blocks called frames

Logical memory is divided into blocks of the same size called pages.

Frames of 1kb each

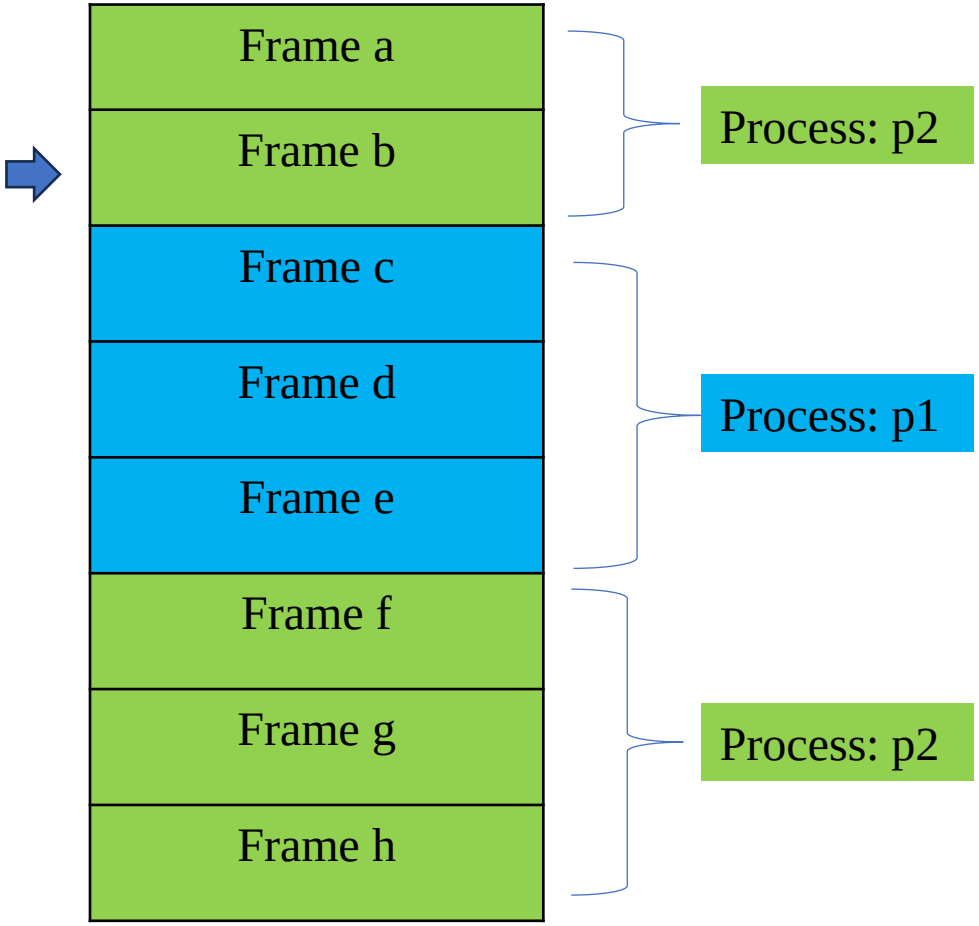


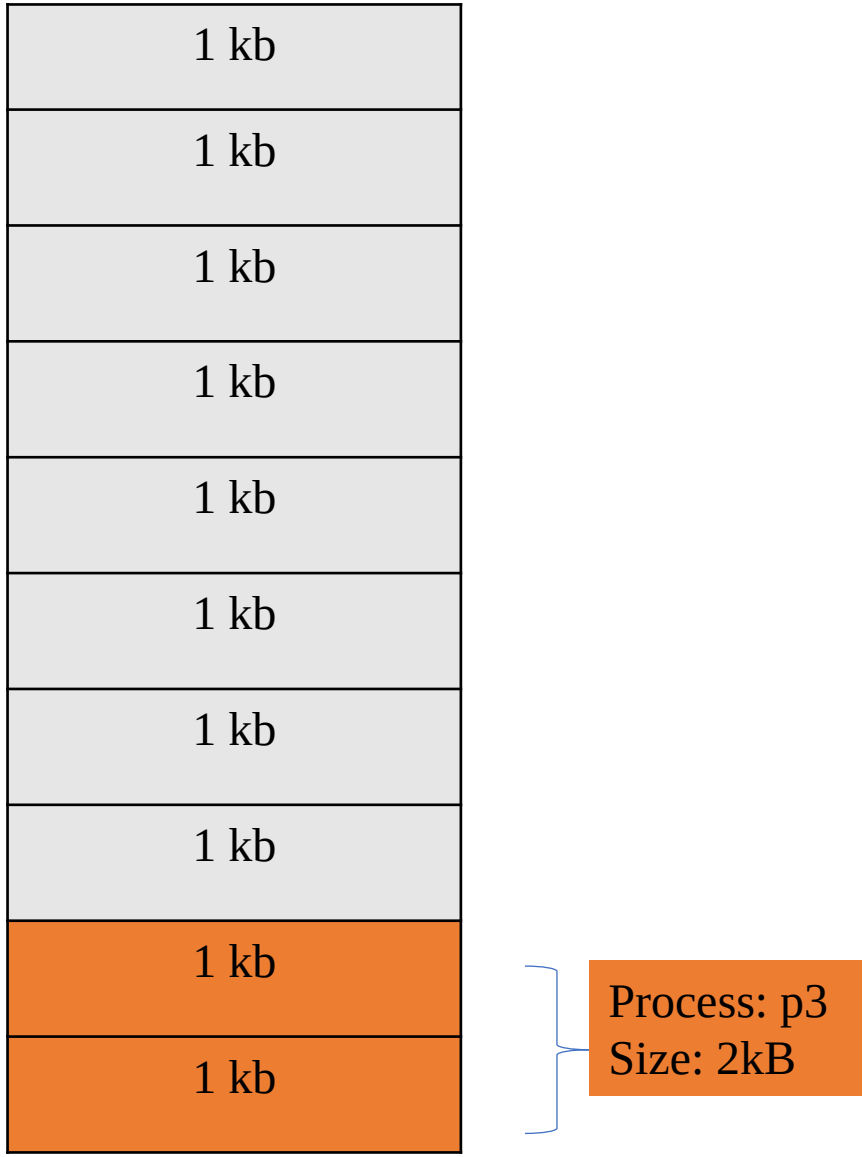


Physical Memory is divided into fixed-sized blocks called frames

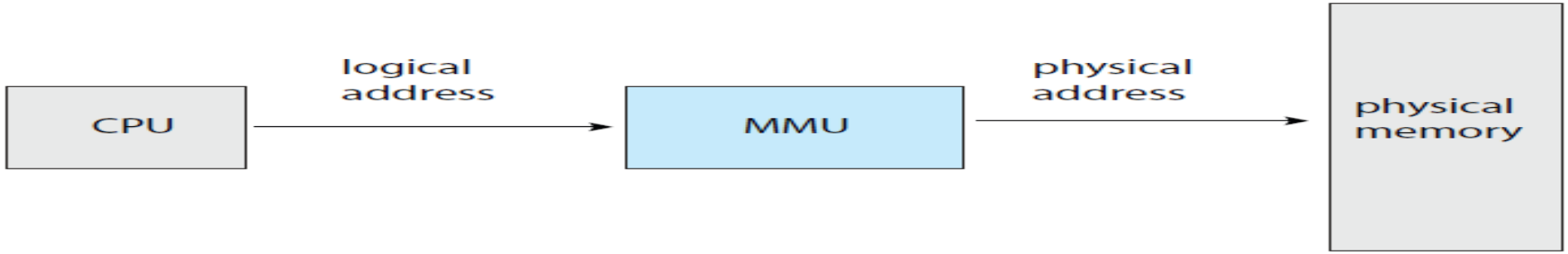
Logical memory is divided into blocks of the same size called pages.

Frames of 1kb each





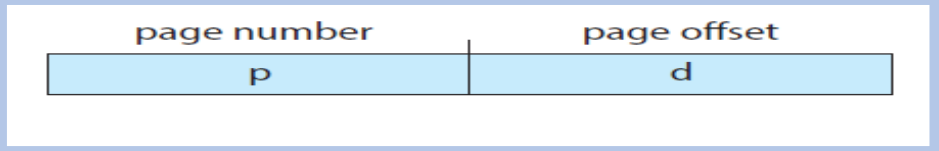
Logical Versus Physical Address Space



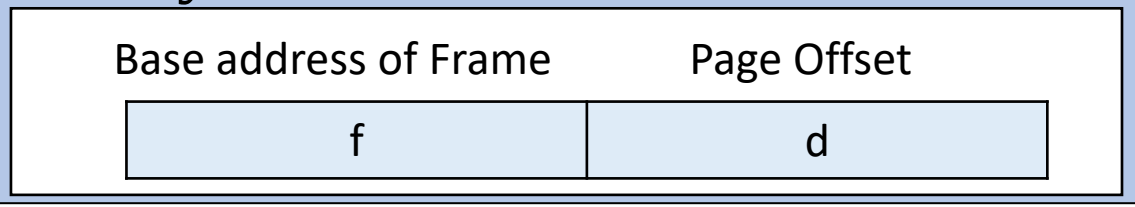
- An address generated by the CPU is commonly referred to as a **logical address**, whereas an address seen by the memory unit—that is, the one loaded into a physical address.
- Binding addresses at either compile or load time generates identical logical and physical addresses.
- The set of all logical addresses generated by a program is a logical address space.
- The set of all physical addresses corresponding to these logical addresses is a physical address space.
- The run-time mapping from virtual to physical addresses is done by a hardware device called the memory-management unit (MMU)

Logical Versus Physical Address Space

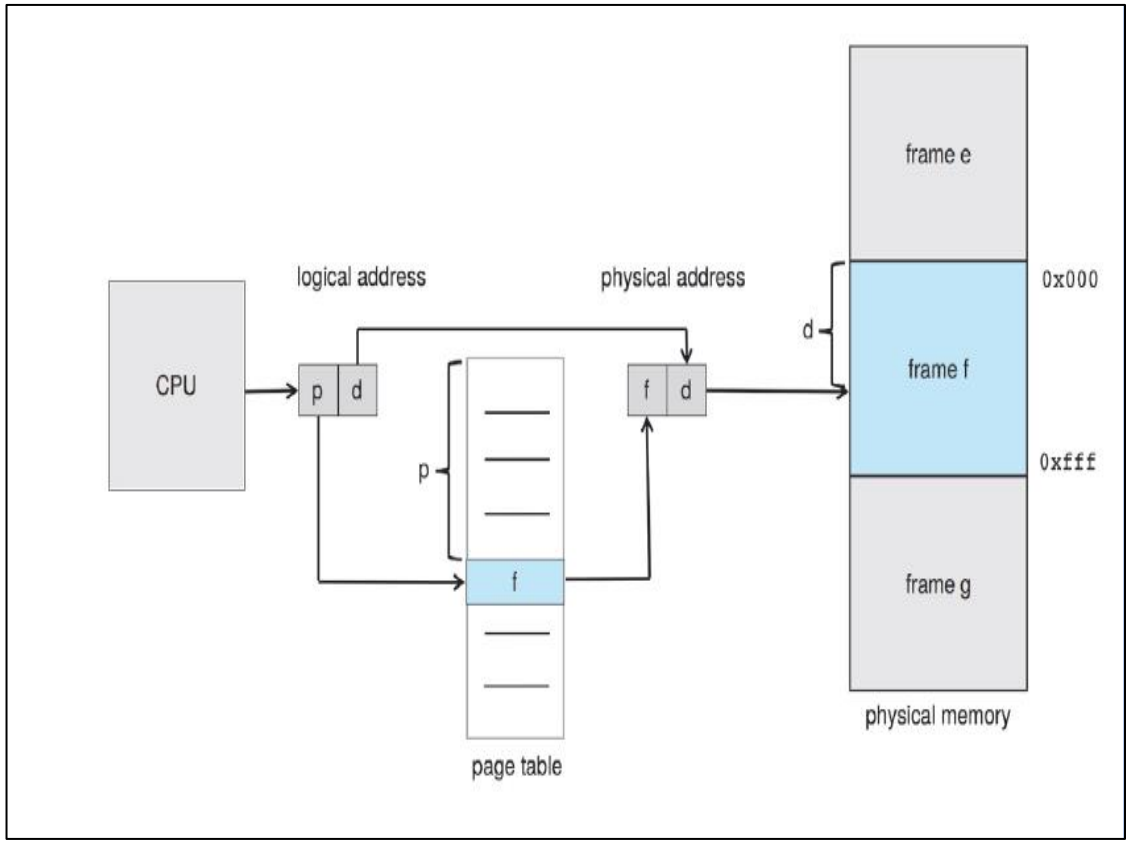
- Every address generated by the CPU is divided into two parts: a page
- number (p) and a page offset (d):



- The page number is used as an index into a per-process page table.
- The page table contains the base address of each frame in physical memory, and the offset is the location in the frame being referenced.
- The base address of the frame is combined with the page offset to define the physical memory address.



The paging model of memory



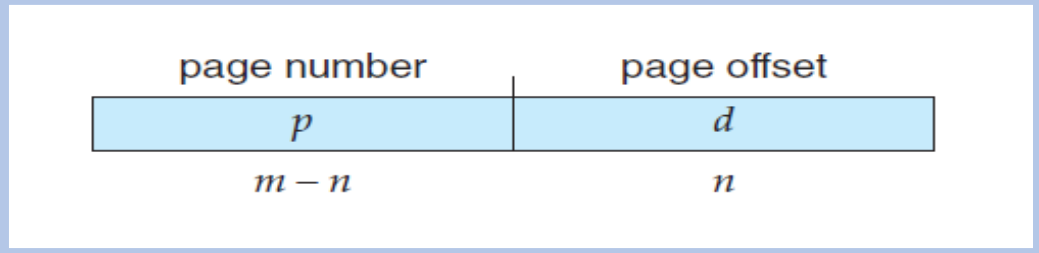
1. Extract the page number p and use it as an index in the page table.
2. Extract the corresponding frame number f from the page table.
3. Replace the page number p in the logical address with the frame number f.

As the offset d does not change, it is not replaced, and the frame number and offset now comprise the physical address

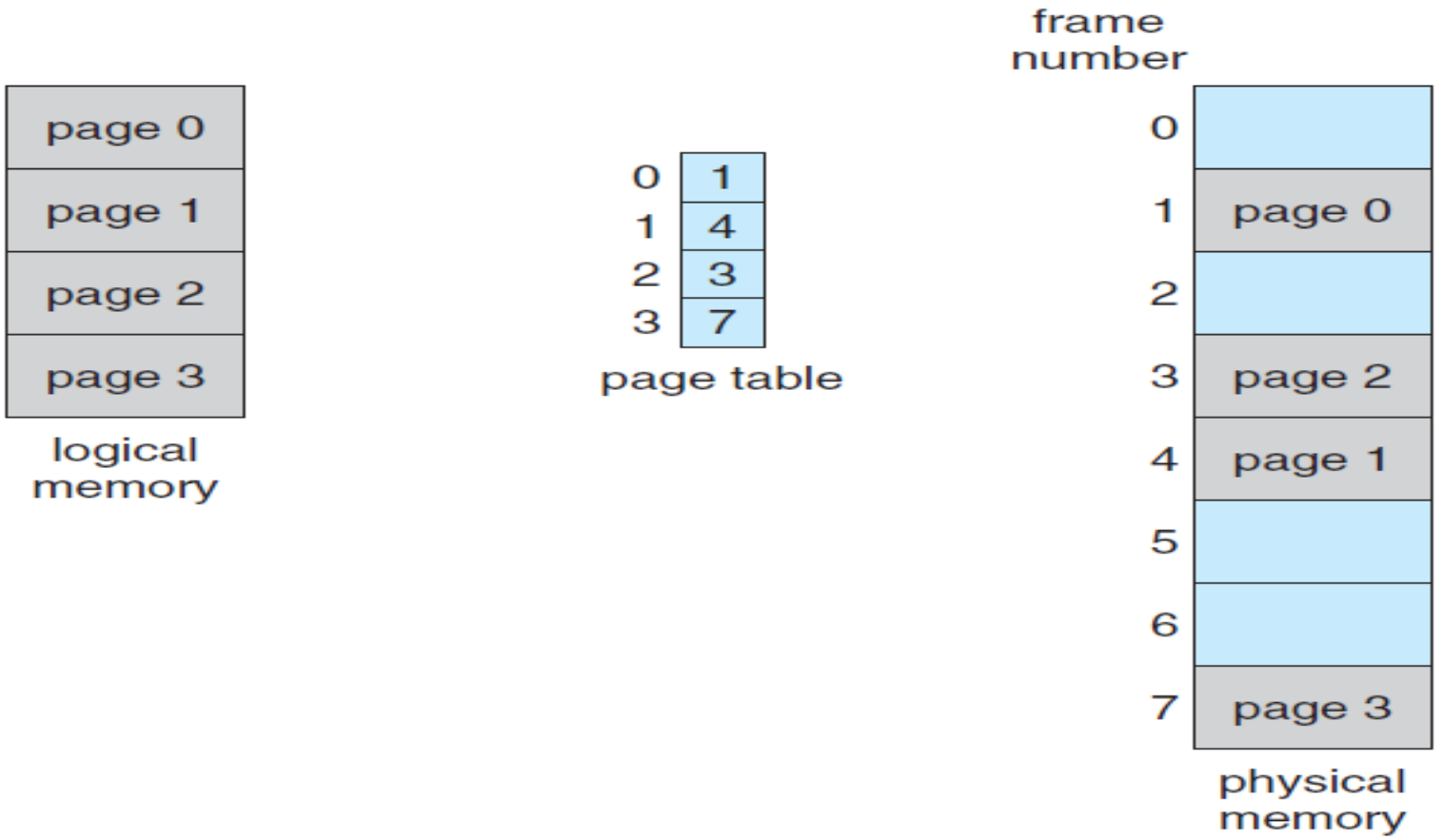
The MMU translates a logical address generated by the CPU to a physical address:

Page Size

- The page size (like the frame size) is defined by the hardware.
- The size of a page is a power of 2, typically varying between 4 KB and 1 GB 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 2^m, and a page size is 2^n bytes, 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:



Paging model of logical and physical memory.

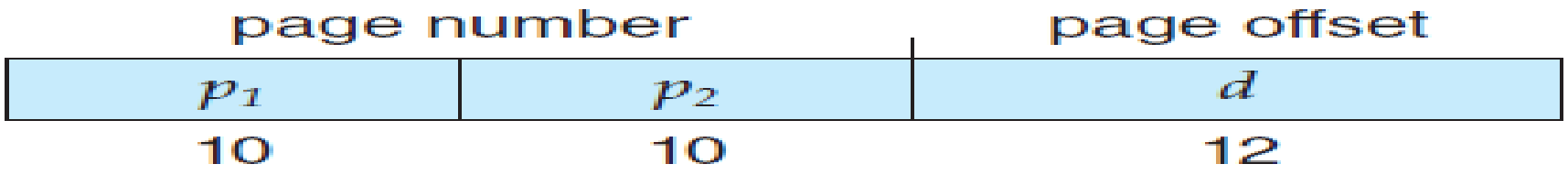


Structure of the Page Table

- Most modern computer systems support a large logical address space (2^{32} to 2^{64}). In such an environment, the page table itself becomes excessively large.
- For example, consider a system with a 32-bit logical address space.
- If the page size in such a system is 4 KB (2^{12}),
  - then a page table may consist of over 1 million entries ($2^{20} = 2^{32}/2^{12}$). Assuming that each entry consists of 4 bytes,
  - Each process may need up to 4 MB of physical address space for the page table alone

two-level paging algorithm

- One way is to use a two-level paging algorithm, in which the page table itself is also paged
- For example, consider again the system with a 32-bit logical address space and a page size of 4 KB.
  - A logical address is divided into a page number consisting of 20 bits and a page offset consisting of 12 bits. Because we page the page table, the page number is further divided into a 10-bit page number and a 10-bit page offset. Thus, a logical address is as follows:



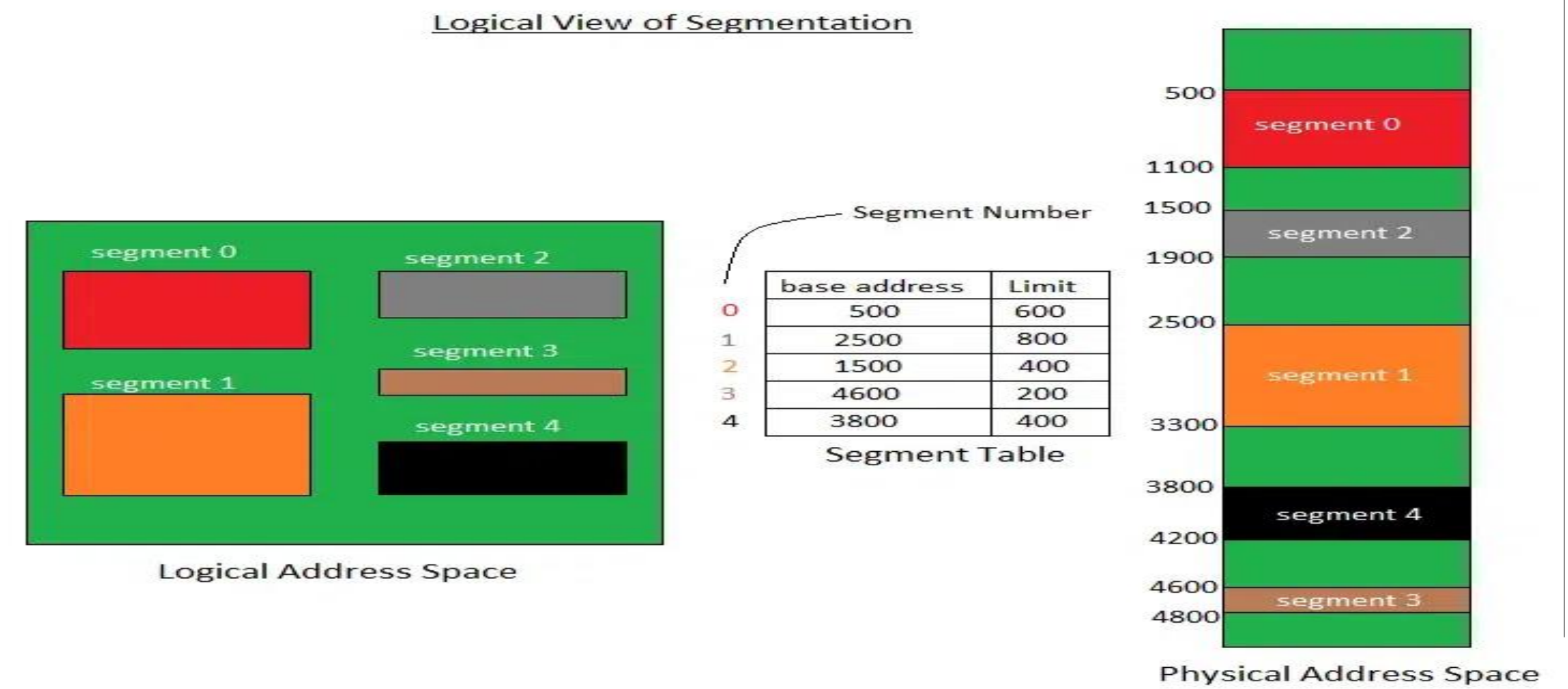
Segmentation Basic Method

- A process is divided into Segments.
- The chunks that a program is divided into which are not necessarily all of the exact sizes are called segments.
- Segmentation gives the user’s view of the process which paging does not provide.
- Here the user’s view is mapped to physical memory.

What is Segment Table?

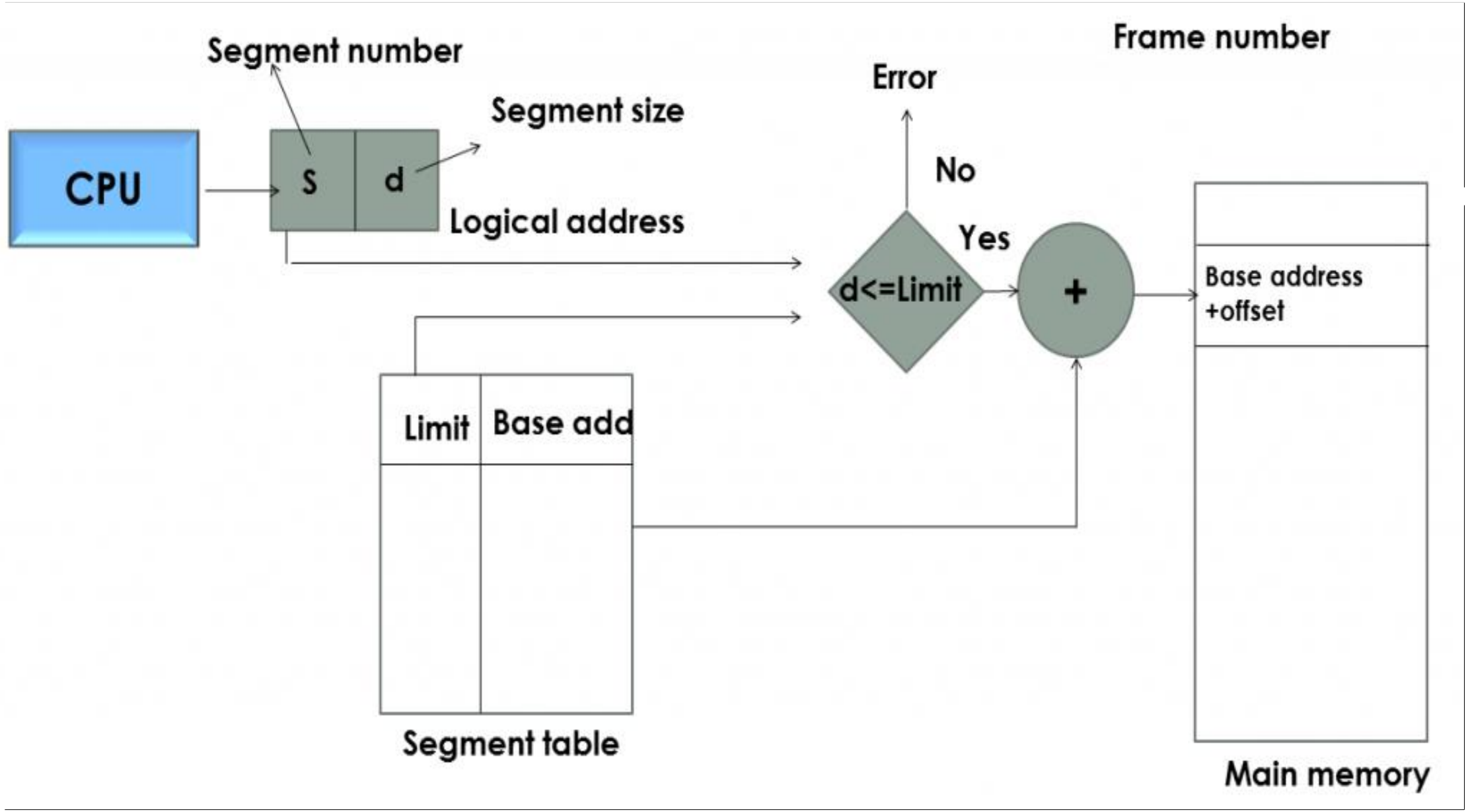
- It maps a two-dimensional Logical address into a one-dimensional Physical address. It’s each table entry has:
- **Base Address:** It contains the starting physical address where the segments reside in memory.
- **Segment Limit:** Also known as segment offset. It specifies the length of the segment.

Logical View of Segmentation



Segmentation

- The process is divided into large chunks called segments or modules.
- The CPU generates this logical address.
- This logical address is to access secondary memory. But CPU has to access the main memory. In this case, address translation is required to convert a logical address into a physical one.
- A physical address is an address for finding data in the main memory. So now the CPU will take the help of the segment table.
- A segment table is a data structure for storing the information of all process segments.


Frame number
Segment number
Error
Segment size
CPU
S
d
No
Logical address
Yes
d<=Limit
+
Base address
+offset
Limit
Base add
Segment table
Main memory

Segmentation

- CPU uses a segment table to map the logical address to a physical address. In the segment table, there are two types of information
  - Limit: Actual size of a segment.
  - Base Address: the address of the segment in the main memory.
- Then if the value of offset(d)<=Limit.
  - Then only the CPU can read that segment;
- else, the error will be there.
- Offset(d) depicts the size of that segment CPU wants to read.

SEGMENTATION PROBLEMS

Problem-

- Consider the following segment table-

Segment No.	Base	Length
0	1219	700
1	2300	14
2	90	100
3	1327	580
4	1952	96

Which of the following logical address will produce trap addressing error?

1. 0, 430
2. 1, 11
3. 2, 100
4. 3, 425
5. 4, 95

Solution

- In a segmentation scheme,
- the generated logical address consists of two parts-
- 1. Segment Number
- 2. Segment Offset

Segment Offset must always lie in the range [0, limit-1].

If segment offset becomes greater than or equal to the limit of segment, then trap addressing error is produced.

- **Option-A: 0, 430**
- Segment Number = 0
- Segment Offset = 430
- In the segment table,
- limit of segment-0 is 700.
- Thus, segment offset must always lie in the range

= [0, 700-1] =[0, 699]

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1219 + 430 = 1649

- **Option-B: 1, 11**
- Segment Number = 1
- Segment Offset = 11
- In the segment table,
- limit of segment-1 is 13.
- Thus, segment offset must always lie in the range

= [1, 13-1] =[1, 12]

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 2300 + 11 = 2311

- **Option-C: 2, 100**
- Segment Number = 2
- Segment Offset = 100
- In the segment table,
- limit of segment-2 is 100.
- Thus, segment offset must always lie in the range = [2, 100-1] =[0, 99]
- In the segment table, limit of segment-2 is 100.
- Thus, segment offset must always lie in the range = [0, 100-1] = [0, 99] , Since generated segment offset does not lie in the above range, so request generated is invalid. Therefore, trap will be produced.

- Option-D: 3, 425-
- Here,
- Segment Number = 3
- Segment Offset = 425
- In the segment table, limit of segment-3 is 580.
- Thus, segment offset must always lie in the range = [0, 580-1] = [0, 579]
- Now,
- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1327 + 425 = 1752

- Option-E: 4, 95-
- Here,
- Segment Number = 4
- Segment Offset = 95
- In the segment table, limit of segment-4 is 96.
- Thus, segment offset must always lie in the range = [0, 96-1] = [0, 95] Now,Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced. Physical Address = 1952 + 95 = 2047

Virtual Memory

- Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.

2. A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and the use of a page or segment table permits this.

Demand paging

- Demand paging in os is a memory management technique used by modern operating systems to efficiently manage memory usage.
- The basic idea behind demand paging is to allow the operating system to load only the parts of a program that are currently needed into memory, rather than loading the entire program all at once.
- When a program is launched, the operating system does not load the entire program into memory at once. Instead, it loads only the parts of the program that are immediately required to start executing. These parts are typically the program's code and any data that is immediately needed. As the program runs and requires more memory, the operating system loads additional parts of the program into memory as needed.

Demand paging

- The operating system also keeps track of which parts of the program are currently being used and which parts are not. If a part of the program has not been used for a long time, the operating system can remove it from memory to free up space for other programs. This process is called “swapping.”
- Demand paging in os is an effective technique because it allows the operating system to use memory more efficiently. Rather than allocating a large block of memory for a program, the operating system can allocate only the memory that is actually needed. This can help reduce the amount of memory that is wasted, which can be especially important in systems with limited memory resources.
- Overall, demand paging is a key technique used by modern operating systems to manage memory usage and improve system performance.

Thrashing

- Thrashing is a condition or a situation when the system is spending a major portion of its time servicing the page faults, but the actual processing done is very negligible.

- **Causes of thrashing:**
- High degree of multiprogramming.
- Lack of frames.
- Page replacement policy.

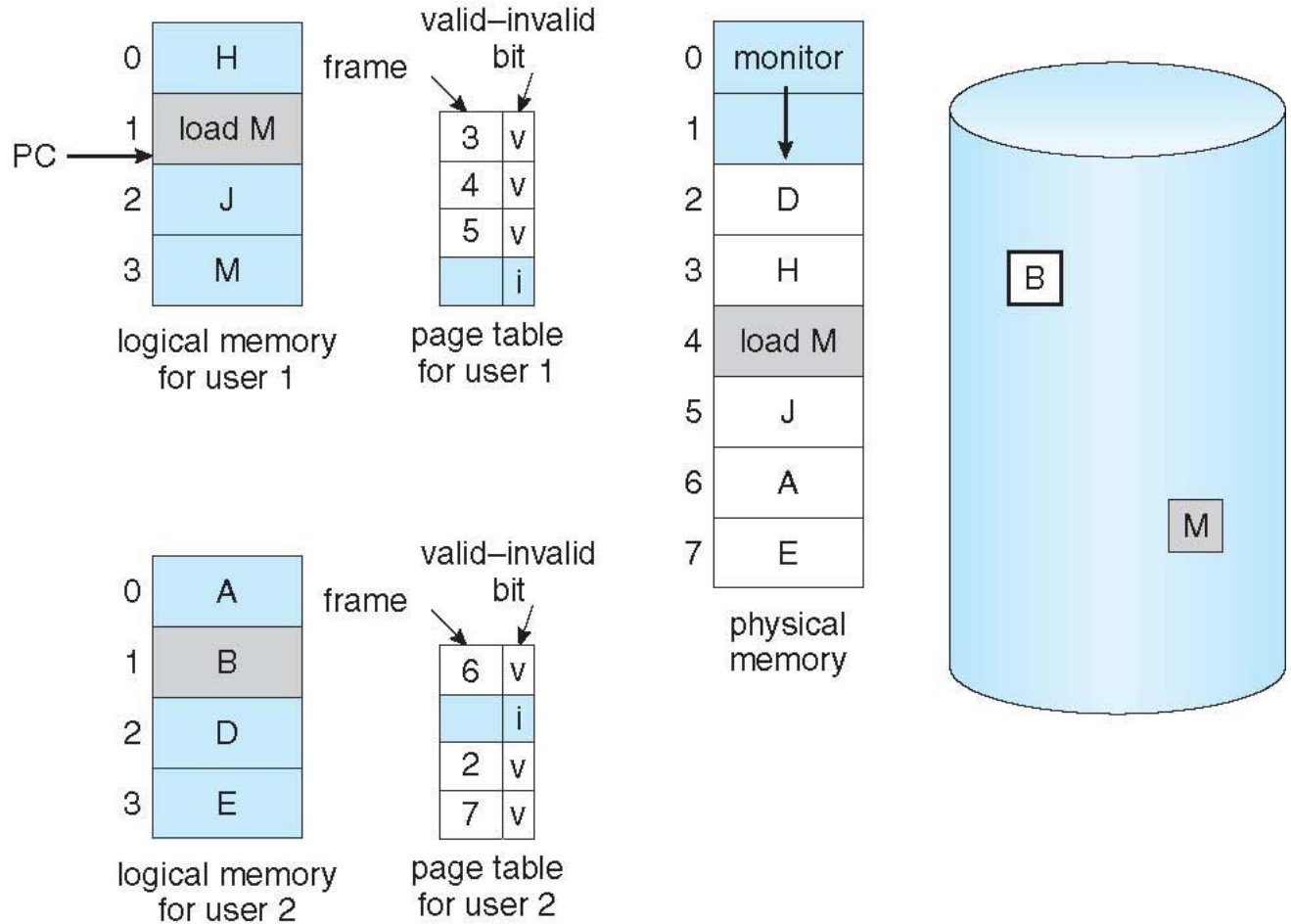
Causes of Thrashing

- If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new system. A global page replacement algorithm is used. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.
- CPU utilization is plotted against the degree of multiprogramming.
- As the degree of multiprogramming increases, CPU utilization also increases.
- If the degree of multiprogramming is increased further, thrashing sets in, and CPU utilization drops sharply.
- So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multiprogramming.

Page Replacement

- Prevent **over-allocation** of memory by modifying page-fault service routine to include page replacement
- Use **modify** (**dirty**) **bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

Need For Page Replacement



Basic Page Replacement

1. Find the location of the desired page on disk

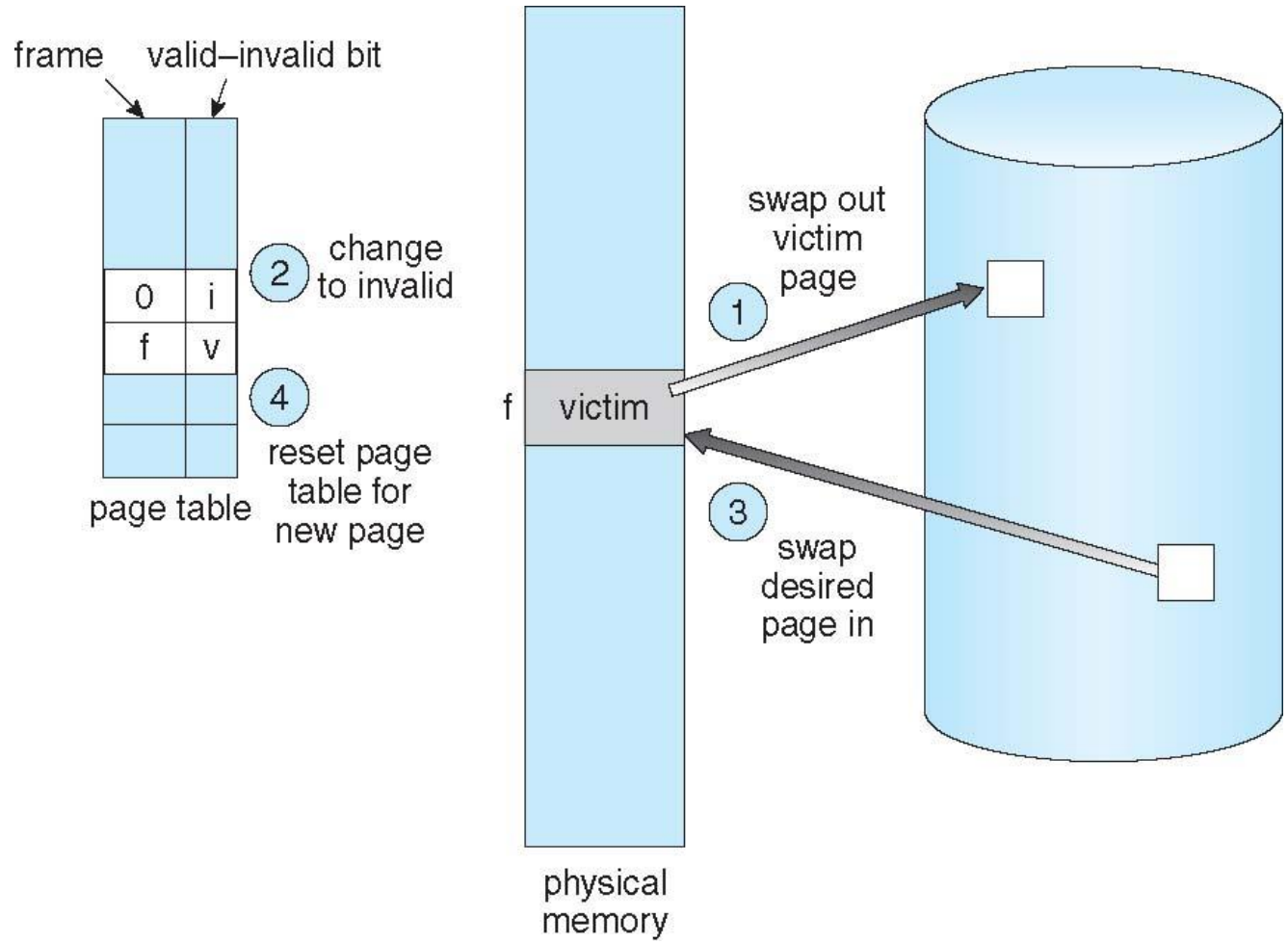
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim frame**
     - Write victim frame to disk if dirty

3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

Page Replacement



Basic Page Replacement

1. Find the location of the desired page on disk

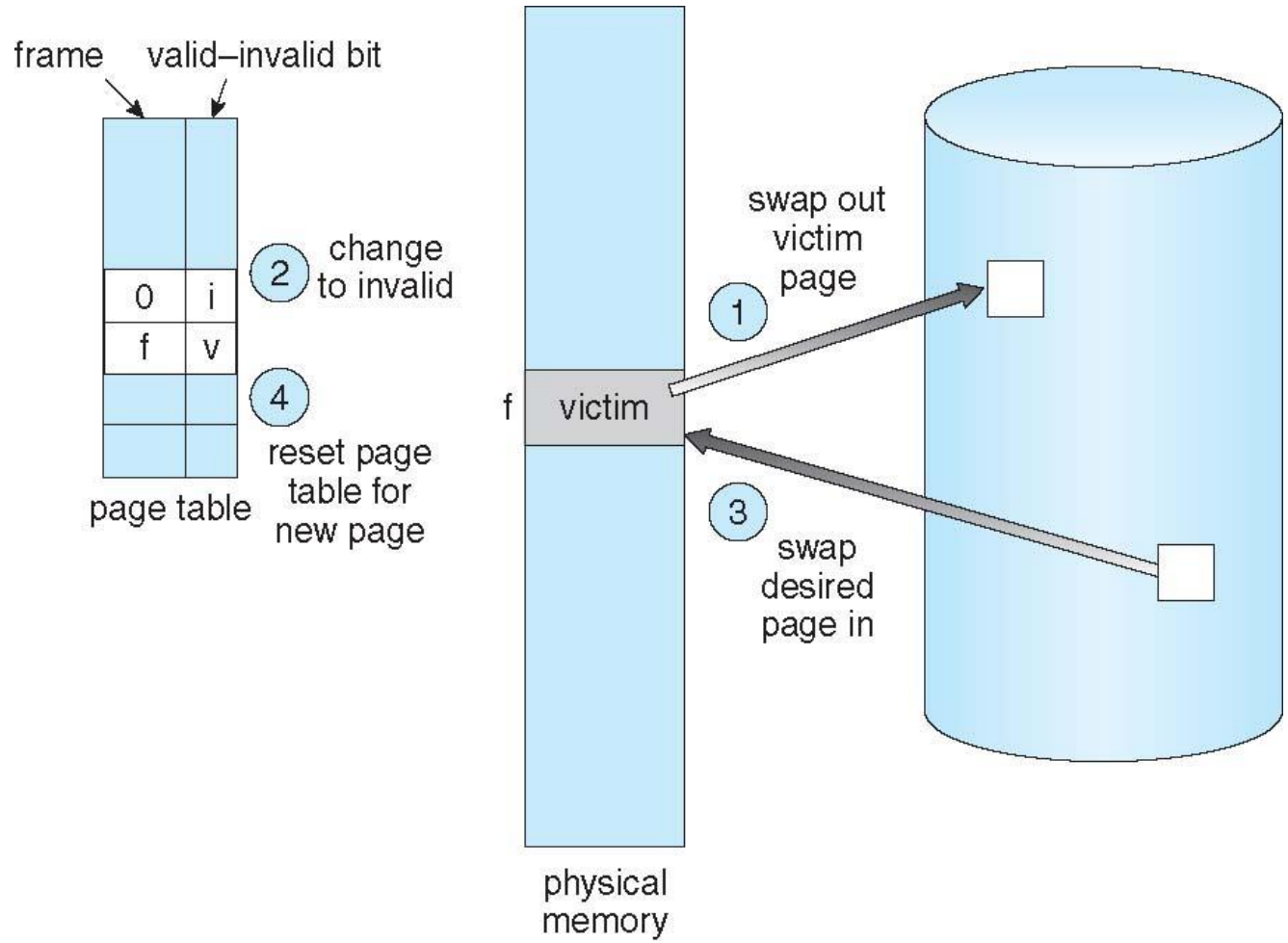
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim frame**
     - Write victim frame to disk if dirty

3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

Page Replacement



Page and Frame Replacement Algorithms

- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is

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

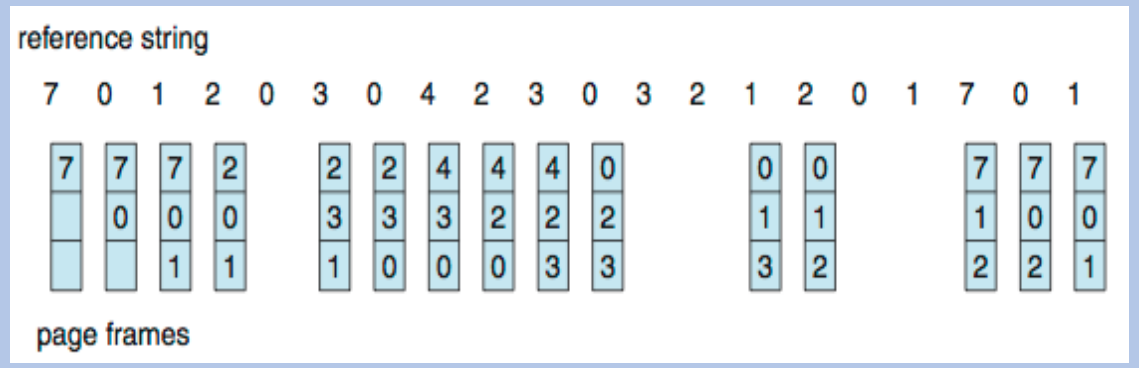
Page and Frame Replacement Algorithms

- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is

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

First-In-First-Out (FIFO) Algorithm

- Reference string:
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)

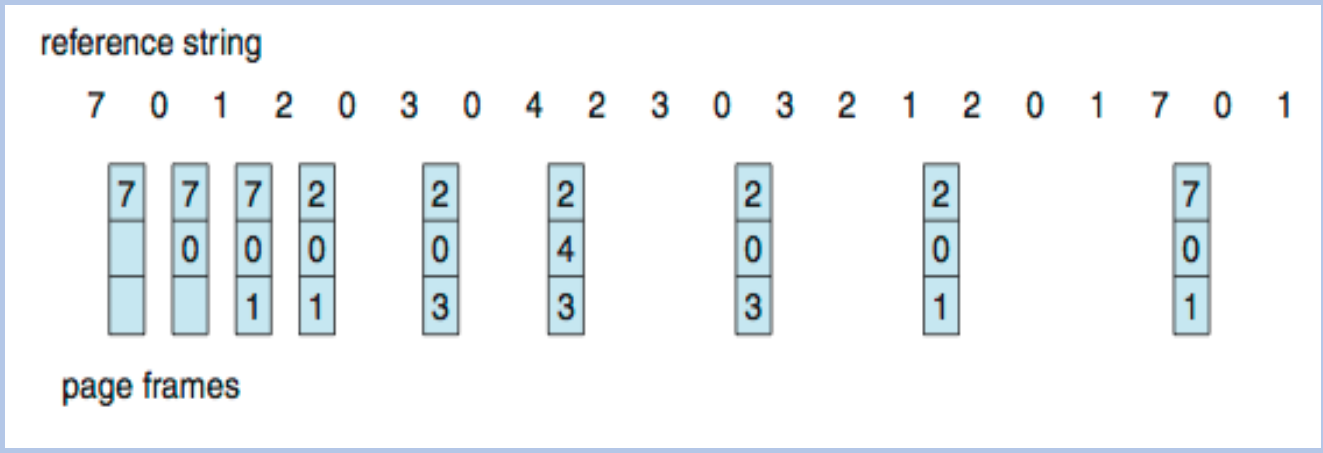


15 page faults

- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults!
    - **Belady's Anomaly**
- How to track ages of pages?
  - Just use a FIFO queue

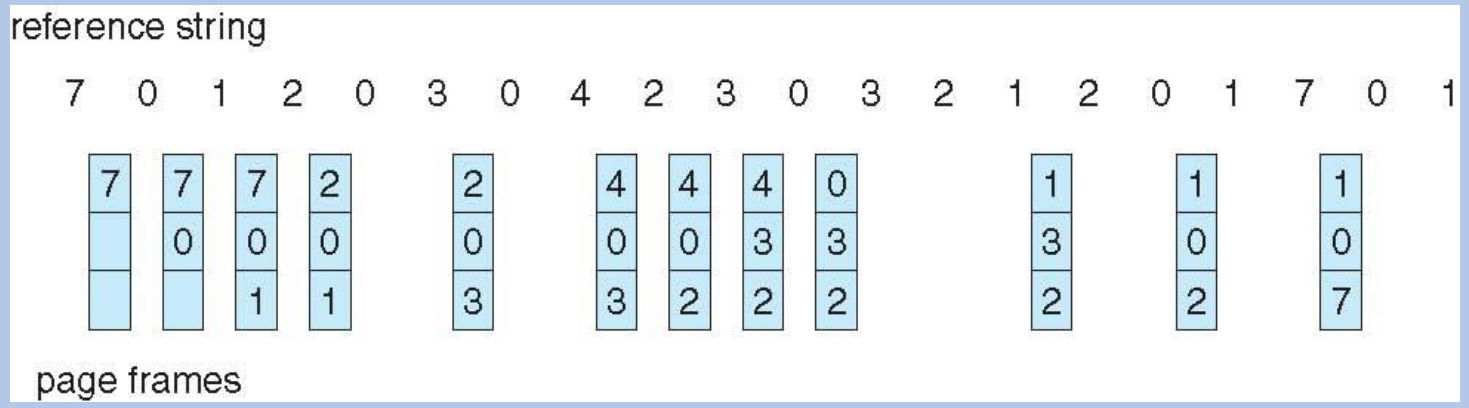
Optimal Algorithm

- Replace page that will not be used for longest period of time
  - 9 is optimal for the example
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs



Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

LRU Algorithm (Cont.)

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to find smallest value
    - Search through table needed
- Stack implementation
  - Keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - But each update more expensive
  - No search for replacement
- LRU and OPT are cases of **stack algorithms** that don't have Belady's Anomaly