MEMORY MANAGEMENT (UNIT 3)

By: Mrs. Nidhi Divecha (ME CMPN)

MEMORY MANAGEMENT & ITS FUNCTIONS

- 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.

FUNCTIONS:

- Memory management keeps track of each and every memory location.
- 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(deallocates memory).

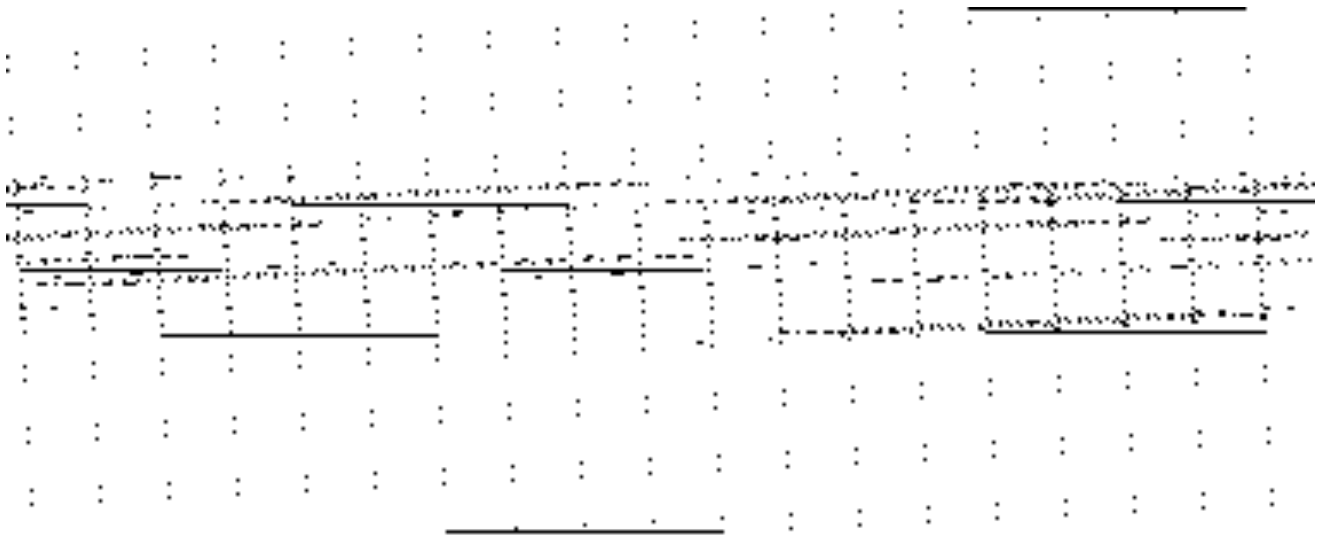
WHY USE MEMORY MANAGEMENT?

Here, are reasons for using memory management:

- It allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.
- Tracks whenever inventory gets freed or unallocated. According to it will update the status.
- It allocates the space to application routines.
- It also make sure that these applications do not interfere with each other.
- Helps protect different processes from each other
- It places the programs in memory so that memory is utilized to its full extent.

ADDRESS SPACE AND MEMORY SPACE

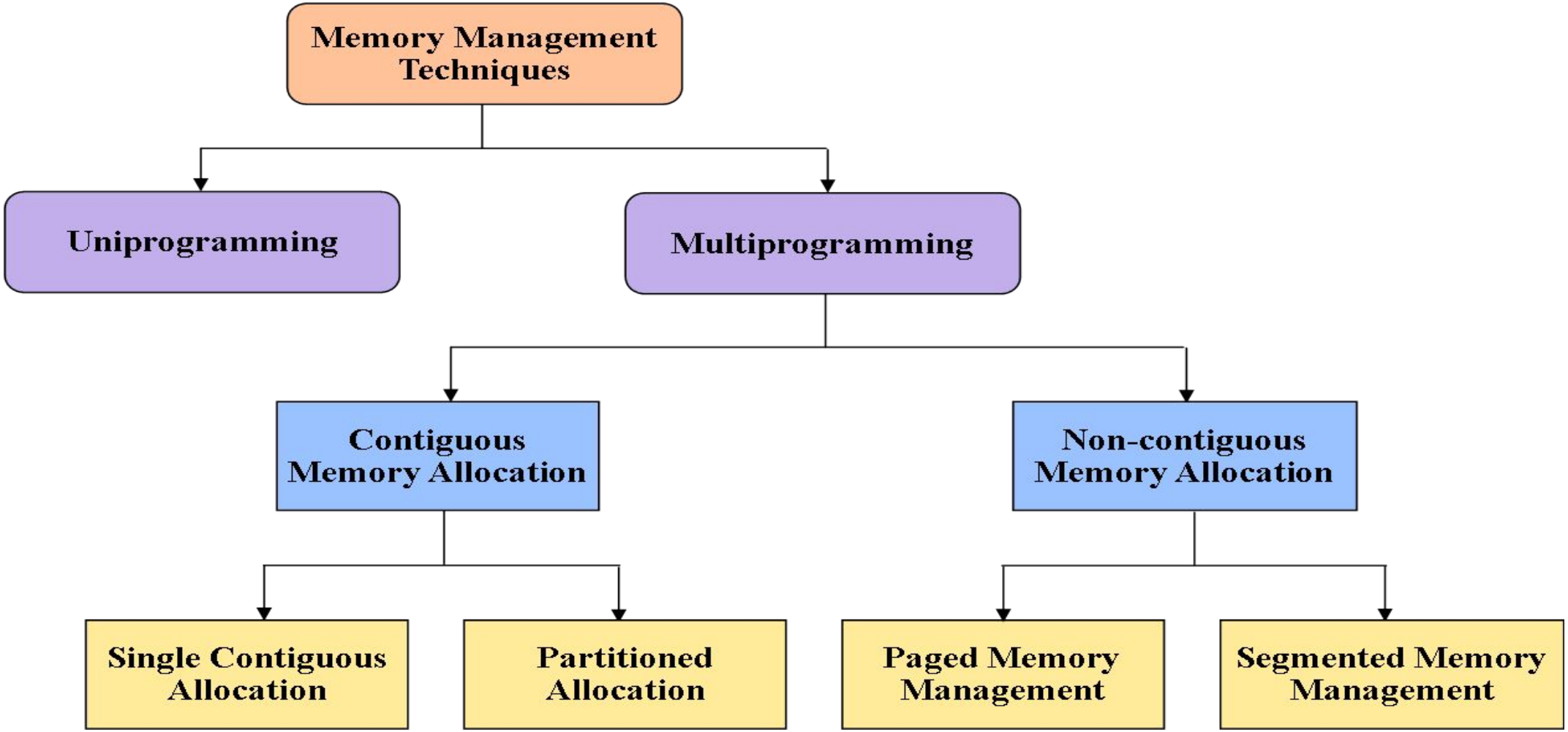
- Virtual memory is the address used by the programmer and the set of such addresses is called **address space.**
- An address in main memory is called a **physical address.**
- The set of such locations in main memory is called the **memory space.**
- Thus, the memory space consists of the actual memory locations directly addressable for processing.



LOGICAL VS PHYSICAL ADDRESS SPACE

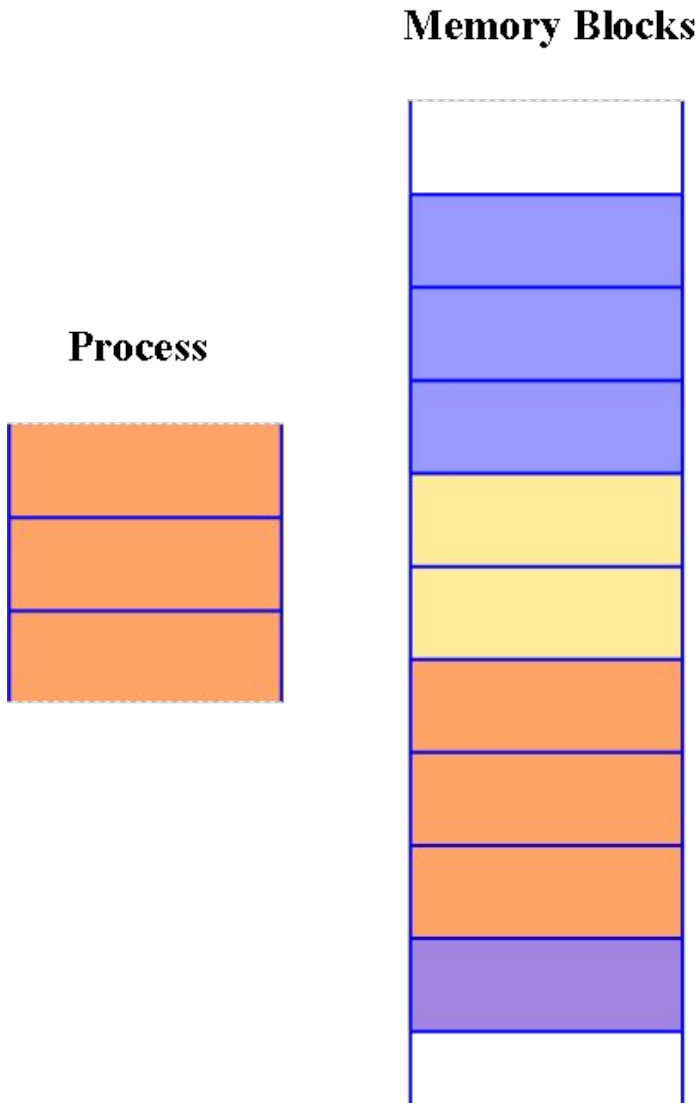
Logical	Physical
Generated by the cpu; also referred to as virtual address space	Address seen by the memory management unit
Logical address space is the set of all logical addresses generated by the program	Physical address space is the set of all physical addresses generated by the program

MEMORY MANAGEMENT TECHNIQUES

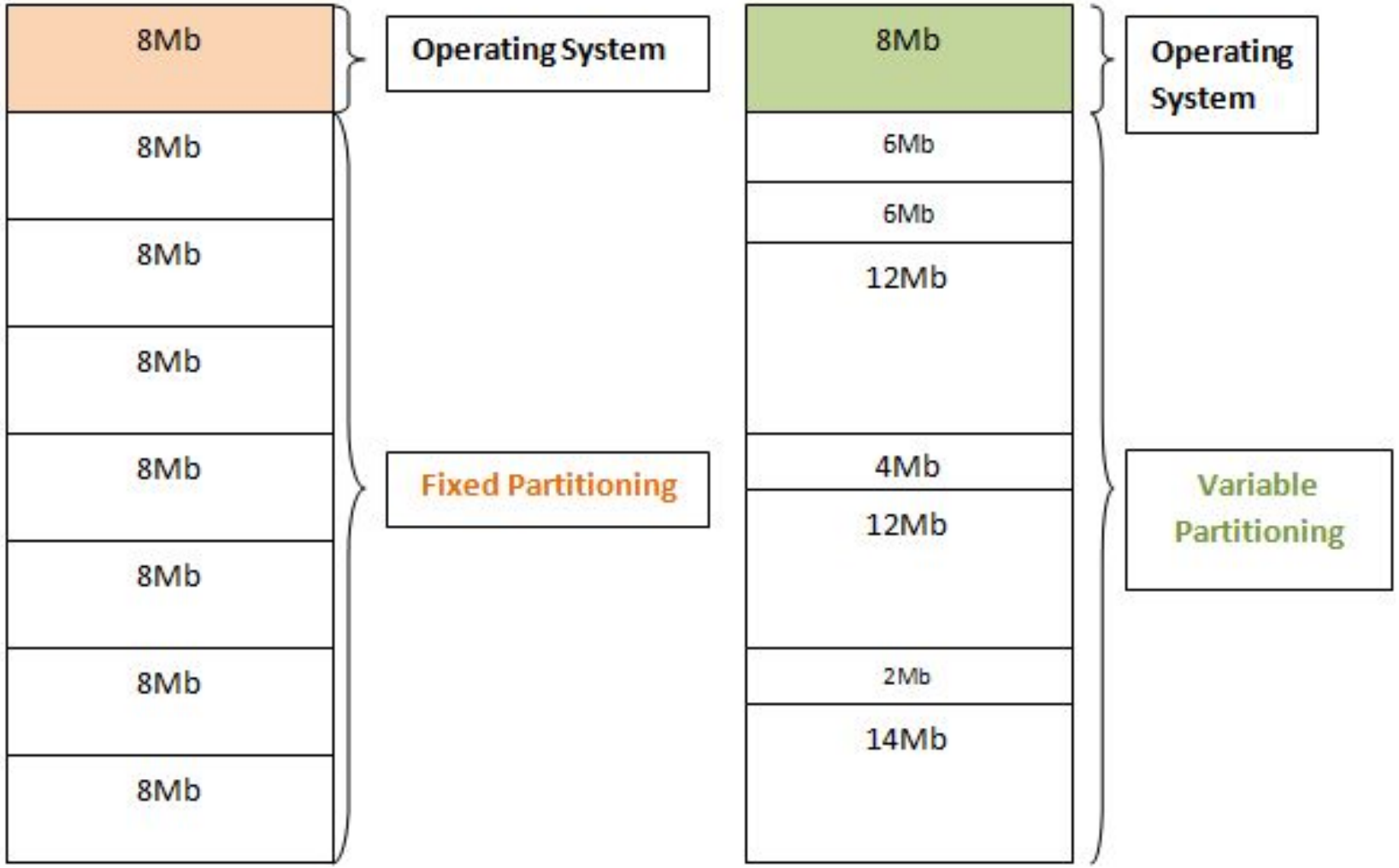


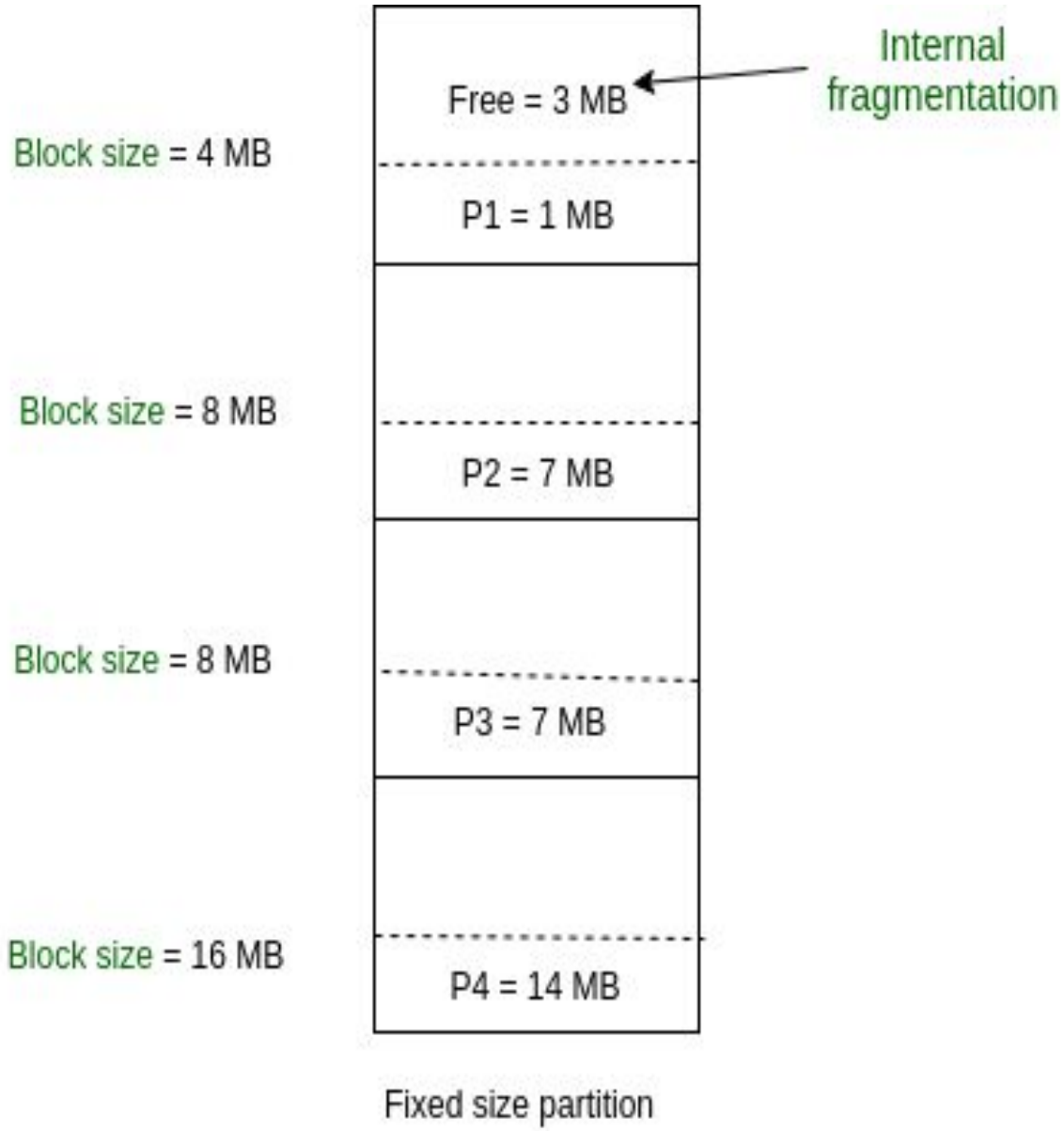
CONTIGUOUS MEMORY ALLOCATION

- Simplest storage space allocation technique where contiguous memory locations are assigned to each program.
- In contiguous memory allocation, all the available memory space remain together in **one place.**
- It means freely available memory partitions are not scattered here and there across the whole memory space.
- In the contiguous memory allocation, both the operating system and the user must reside in the main memory. The main memory is divided into two portions one portion is for the operating and other is for the user program.
- In the contiguous memory allocation when any user process request for the memory a single section of the contiguous memory block is given to that process according to its need.
- OS has to estimate the amount of memory required for the complete process before allocation.
- We can accomplish the contiguous memory allocation by separating memory into the **fixed-sized partition.**



Fixed sized partition & variable sized partition:





FIXED-SIZED PARTITION (STATIC)

- This is the oldest and simplest technique used to put more than one process in the main memory(RAM).
- In this partitioning, **number of partitions in RAM are fixed** but **size** of each partition may or **may not be same.**
- Also known as **MFT** –Multiprogramming with fixed no of tasks.
- As it is contiguous memory allocation, hence no spanning(cannot break the process) is allowed.
- Here partition are made before execution or during system configure.

Advantage:

- Management or accounting is simple.

Disadvantage:

- Occurrence of Internal fragmentation.

Fragmentation:

- Wastage of memory

It is of two types:

Internal Fragmentation: that fragmentation which is internal part of allocated partition.

External Fragmentation: fragmentation due to unallocated partition.

As illustrated in above figure,

- first process is only consuming 1MB out of 4MB in the main memory.
  Hence, Internal Fragmentation in first block is (4-1) = 3MB.
  Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.
- Suppose process P5 of size 7MB comes.
- But this process cannot be accommodated inspite of available free space because of contiguous allocation (as spanning is not allowed).
- Hence, 7MB becomes part of External Fragmentation.

VARIABLE PARTITION ALLOCATION (DYNAMIC)

- It is a part of contiguous allocation technique.
- It is used to alleviate the problem faced by fixed partitioning.
- In contrast with fixed partitioning, partition cannot be made before the execution or during system configure.

Various **features** associated with variable Partitioning-

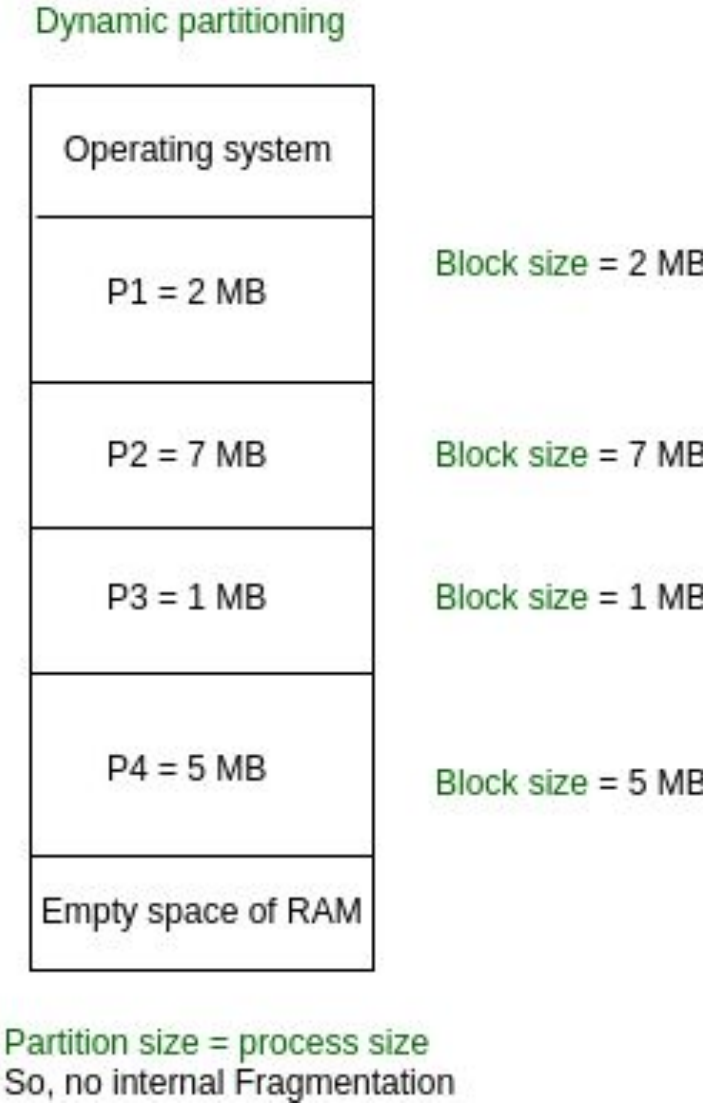
- Initially RAM is empty, and partitions are made during the run-time according to process's need instead of partitioning during system configure.
- The size of partition will be equal to incoming process.
- The partition size varies according to the need of the process so that the internal fragmentation can be avoided to ensure efficient utilization of RAM.
- Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory's size.

Advantage:

- This technique is free from internal fragmentation.

Disadvantage:

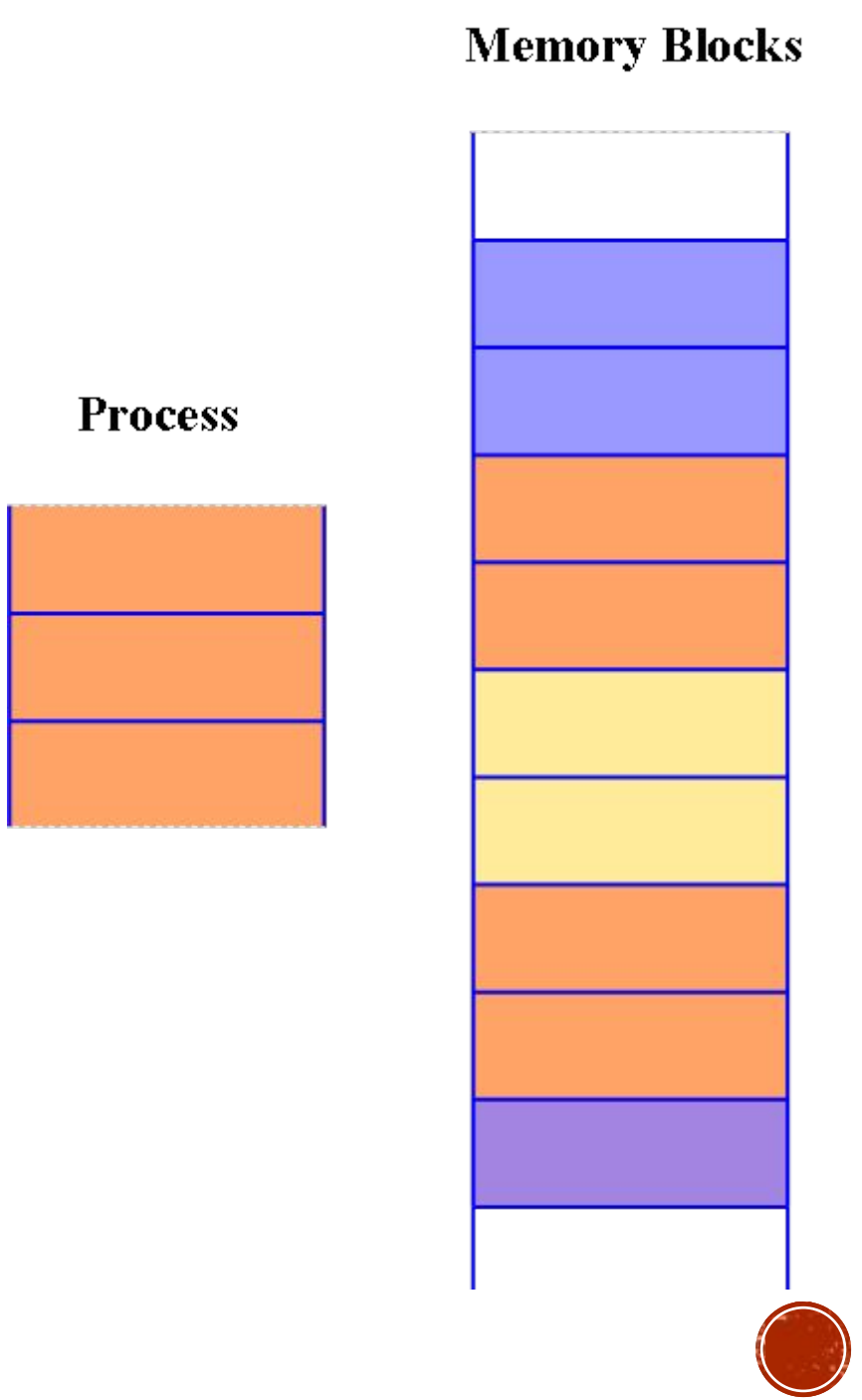
- Management is troublesome as memory is getting absolutely divided after some time.



- There will be external fragmentation in spite of absence of internal fragmentation.
- For example, suppose in above example- process P1(2MB) and process P3(1MB) completed their execution.
- Hence two spaces are left i.e. 2MB and 1MB.
- Let’s suppose process P5 of size 3MB comes.
- The empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation.
- The rule says that process must be contiguously present in main memory to get executed.
- Hence it results in External Fragmentation.

NON-CONTIGUOUS MEMORY ALLOCATION

- Program is divided into chunks called **segments.**
- Each segment can be placed in different parts of memory.
- This technique is time-consuming.
- Non-Contiguous memory allocation allocates the memory space present in different locations to the process as per its requirements.
- As all the available memory space is in a distributed pattern so the freely available memory space is also scattered here and there.
- In non-contiguous allocation, Operating system needs to maintain the table which is called **Page Table** for each process which contains the base address of each block which is acquired by the process in memory space
- This technique of memory allocation helps to reduce the wastage of memory, which eventually gives rise to Internal and external fragmentation.
- Spanning is allowed which is not possible in other techniques like Dynamic or Static Contiguous memory allocation. That's why paging is needed to ensure effective memory allocation. Paging is done to remove External Fragmentation.
- OS takes the help of virtual storage technique when it runs out of primary memory.



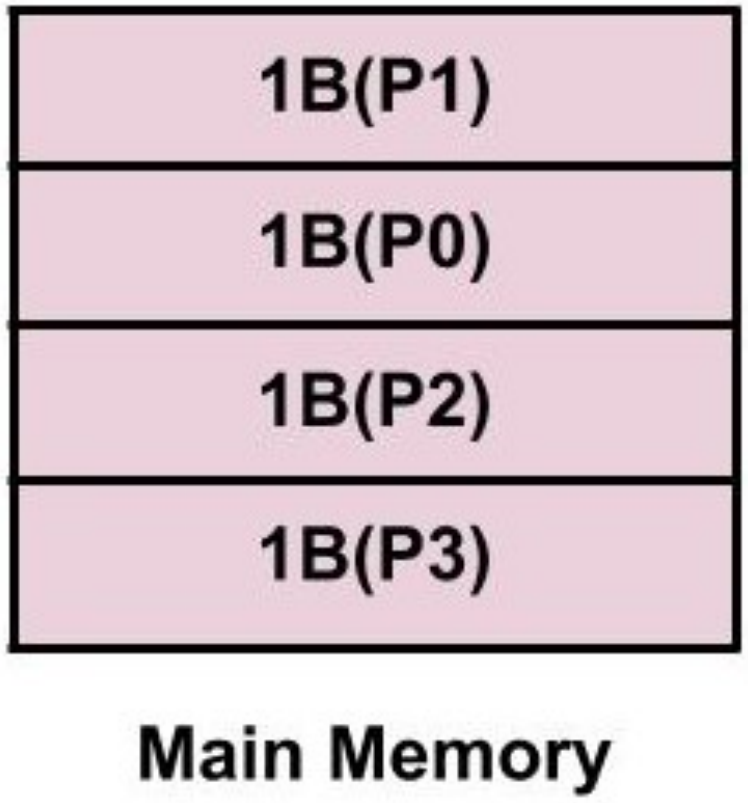
Two approaches to virtual storage:

- **Program paging:** A program is broken down into **fixed size page.**
- **Program Segmentation:** A program is broken down into **logical units** called segments.
- OS uses a combination of page and program segmentation to optimize memory usage.

Basis for comparison	Paging	Segmentation
Basic	In paging, program is divided into fixed or mounted size pages.	In segmentation, program is divided into variable size sections.
Size	Page size is determined by hardware.	Here, the segment size is given by the user.
Performance	It is faster in the comparison of segmentation.	Segmentation is slow.
Fragmentation	Paging could result in internal fragmentation.	Segmentation could result in external fragmentation.
Logical address	In paging, logical address is split into page number and page offset.	Here, logical address is split into segment number and section offset.
Address	Paging comprises a page table which encloses the base address of every page.	While segmentation also comprises the segment table which encloses segment number and segment offset.
Table	Page table is employed to keep up the page data.	Segment Table maintains the section data.
Visibility	Paging is invisible to the user.	Segmentation is visible to the user.

PAGING

- In paging, main memory is divided into a number of blocks of fixed size called **frames.**
- Paging is a non-contiguous memory allocation technique in which secondary memory and the main memory is divided into **equal size** partitions.
- Each process is divided into a number of blocks of fixed size called **pages.**
- The partitions of the **secondary memory** are called **pages** while the partitions of the **main memory** are called **frames**.
- Here **size of page = size of frame**
- They are divided into equal size partitions to have maximum utilization of the main memory and avoid external fragmentation.
- **Example:** We have a process P having process size as 4B, page size as 1B.
- Therefore, there will be we four pages(say, P0, P1, P2, P3) each of size 1B. Also, when this process goes into the main memory for execution then depending upon the availability, it may be stored in non-contiguous fashion in the main memory frame as shown below:



Translation of logical Address into physical Address

- As a CPU always generates a logical address and we need a physical address for accessing the main memory. This mapping is done by the MMU(memory management Unit) with the help of the **page table**.

Logical Address: The logical address consists of two-parts page **number** and **page offset.**

- **Page Number:** It tells the exact page of the process which the CPU wants to access.
- **Page Offset:** It tells the exact word on that page which the CPU wants to read.

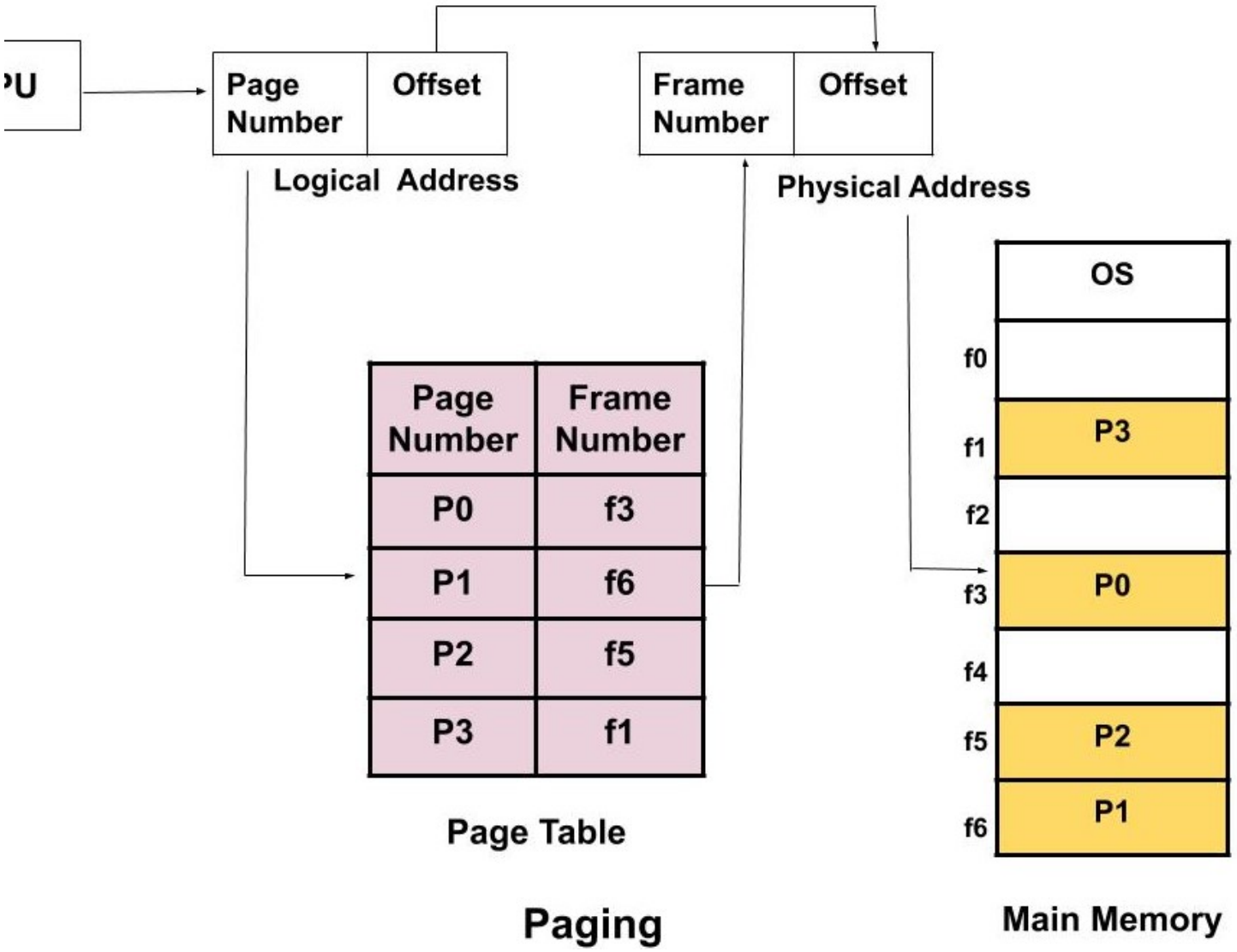
Logical Address = Page Number + Page Offset

Physical Address: The physical address consists of two parts **frame number** and **page offset.**

- **Frame Number:** It tells the exact frame where the page is stored in physical memory.
- **Page Offset:** It tells the exact word on that page which the CPU wants to read. It requires no translation as the page size is the same as the frame size so the place of the word which CPU wants access will not change.

Physical Address = Frame Number + Page Offset

- **Page table:** A page table contains the frame number corresponding to the page number of some specific process. So, each process will have its own page table. A register called Page Table Base Register(PTBR) which holds the base value of the page table.



HOW IS THE TRANSLATION DONE?

- The CPU generates the logical address which contains the **page number** and the **page offset**. The PTBR register contains the address of the page table.
- Now, the page table helps in determining the **frame number** corresponding to the page number.
- Now, with the help of frame number and the page offset the physical address is determined and the page is accessed in the main memory.

SEGMENTATION

- In paging, we were blindly diving the process into pages of fixed sizes but in segmentation, we divide the process into modules for better visualization of the process.
- Here each segment or module consists of the same type of functions.
- **For example,** the main function is included in one segment, library function is kept in other segments, and so on.
- As the size of segments may vary, so memory is divided into variable size parts.

Advantages of Segmentation

- The size of the segment table is less compared to the size of the page table.
- There is no internal fragmentation.

Disadvantages of Segmentation

- When the processes are loaded and removed (during swapping) from the main memory then free memory spaces are broken into smaller pieces and this causes external fragmentation.
- Here also the time to access the data increases as due to segmentation the main memory has to be now accessed two times. First, we need to access the segment table which is also stored in the main memory and second, combine the base address of the segment with the segment offset and then get the physical address which is again stored in the main memory.

Translation of logical Address into physical Address

- As a CPU always generates a logical address and we need a physical address for accessing the main memory. This mapping is done by the MMU(memory management Unit) with the help of the **segment table**.

Logical Address: The logical address consists of two parts **segment number** and **segment offset.**

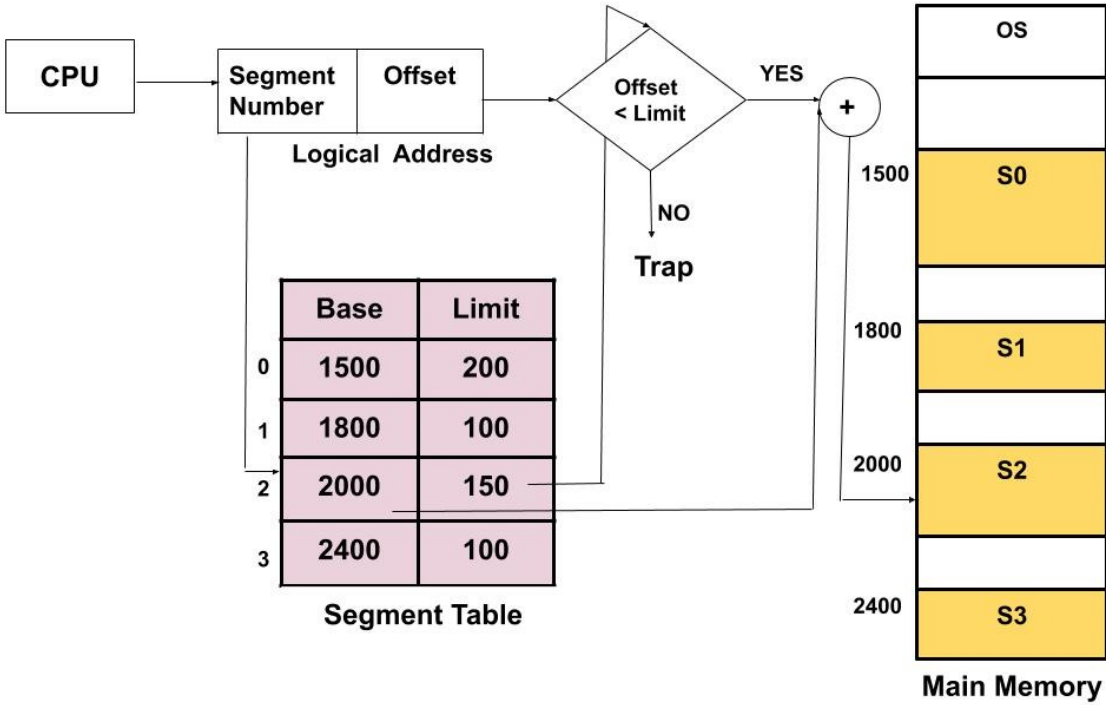
- **Segment Number:** It tells the specific segment of the process from which the CPU wants to read the data.
- **Segment Offset:** It tells the exact word in that segment which the CPU wants to read.

Logical Address = Segment Number + Segment Offset

- **Physical Address:** The physical address is obtained by adding the **base address** of the segment to the segment offset.
- **Segment table:** A segment table stores the base address of each segment in the main memory. It has two parts i.e. **Base and Limit**. Here, **base** indicates the base address or starting address of the segment in the main memory. **Limit** tells the size of that segment. A register called Segment Table Base Register(STBR) which holds the base value of the segment table. The segment table is also stored in the main memory itself.

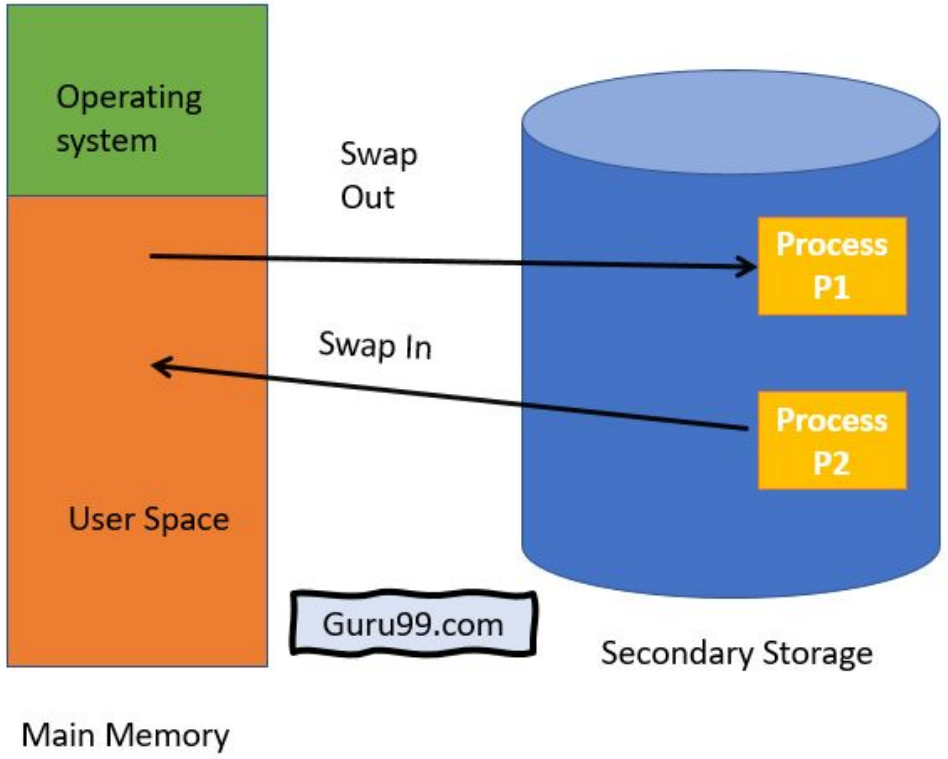
HOW IS THE TRANSLATION DONE?

- The CPU generates the logical address which contains the segment **number** and the segment **offset**. **STBR** register contains the address of the segment table. Now, the segment table helps in determining the **base address of the segment** corresponding to the page number.
- Now, the segment offset is compared with the limit corresponding to the Base. If the **segment offset** is greater than the **limit** then it is an invalid address. This is because the CPU is trying to access a word in the segment and this value is greater than the size of the segment itself which is not possible. If the segment offset is less than or equal to the limit then only the request is accepted. The physical address is generated by adding the base address of the segment to the segment offset.



Segmentation

WHAT IS SWAPPING?



- Swapping is the technique used by an operating system for **efficient management of memory space** of a computer system.
- Swapping involves performing two tasks called **swapping in** and **swapping out.**
- The task of placing the pages or blocks of data from the **hard disk to the main memory** is called **swapping in.**
- On the other hand, the **task of removing page**s or blocks of data from **main memory to the hard disk** is called **swapping out.**
- The swapping technique is useful when larger program is to be executed or some operations have to performed on a large file.
- Swapping is a method in which the process should be swapped **temporarily** from the main memory to the **backing store.**
- It will be later brought back into the main memory for continue execution.
- Backing store is a hard disk or some other secondary storage device that should be big enough in order to accommodate copies of all memory images for all users.

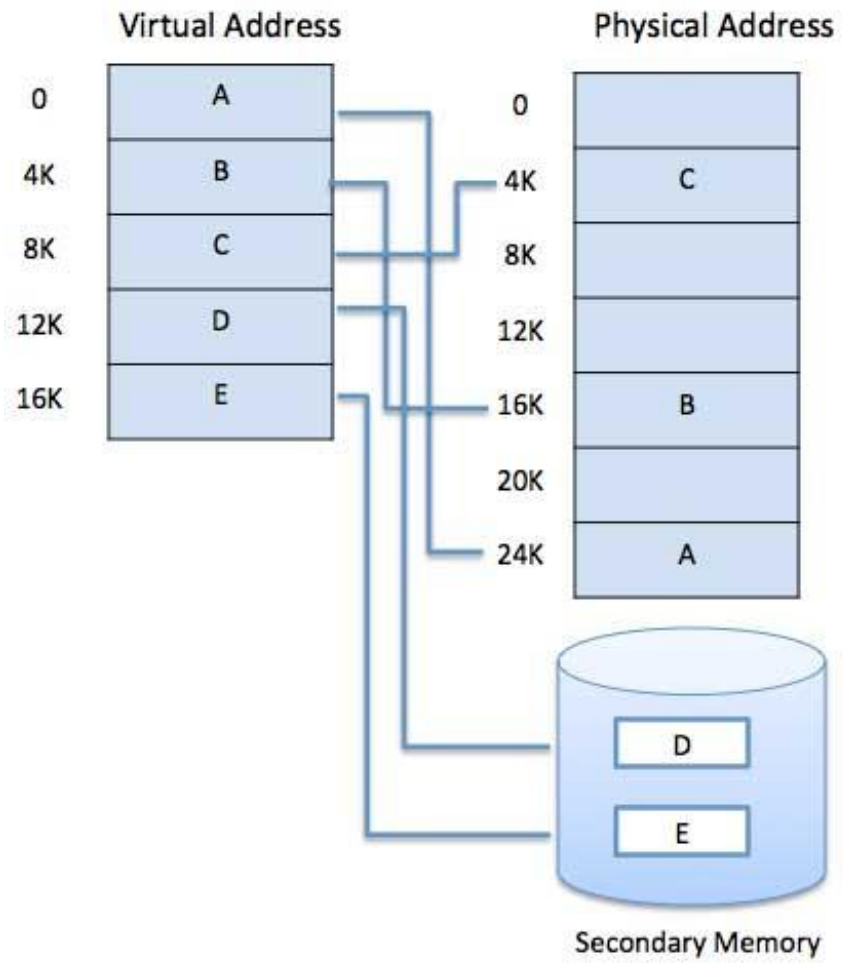
Benefits of Swapping

- It offers a **higher degree of multiprogramming.**
- It helps to get **better utilization of memory**.
- **Minimum wastage of CPU time** on completion so it can easily be applied to a priority-based scheduling method to improve its performance.

VIRTUAL MEMORY IN OPERATING SYSTEM

- Virtual memory is **temporary memory** which is used along with the RAM of the system.
- Virtual Memory is a **storage scheme** that provides user an illusion of **having a very big main memory.**
- This is done by **treating a part of secondary memory as the main memory.**
- In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.
- Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.
- By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.
- **Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk.**
- **Second, it allows us to have memory protection, because each virtual address is translated to a physical address.**
- It is a technique that is implemented using both **hardware and software**.
- It **maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.**

- Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The **MMU's** job is to translate **virtual addresses into physical addresses**. A basic example is given below −



- **Virtual memory** is implemented using **Demand Paging or Demand Segmentation**.

Advantages of Virtual Memory

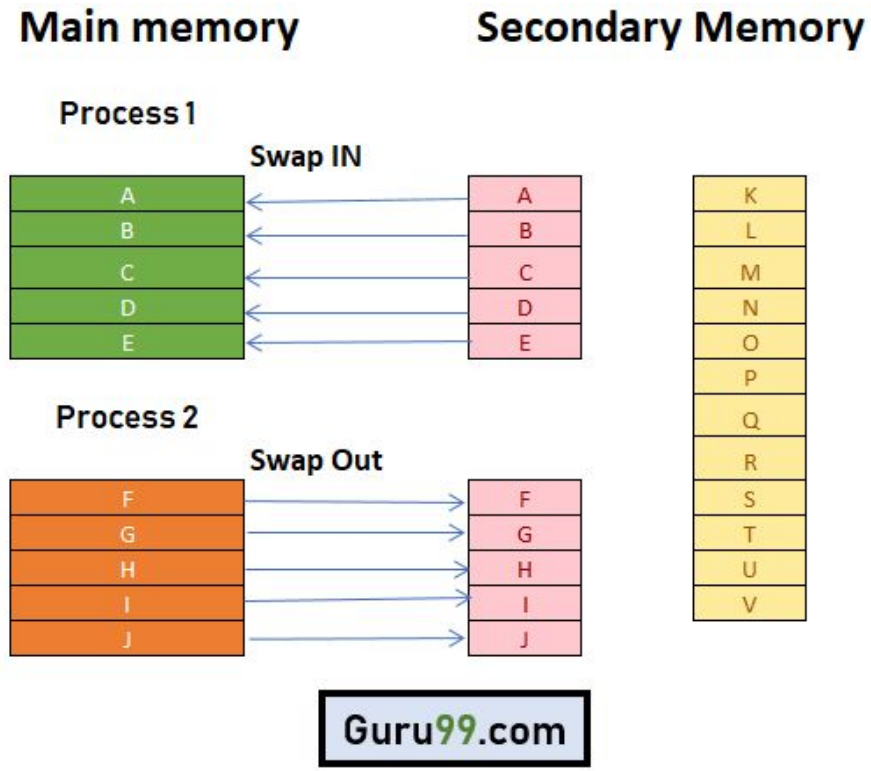
- The degree of Multiprogramming will be increased.
- User can run large application with less real RAM.
- There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

- The system becomes slower since swapping takes time.
- It takes more time in switching between applications.
- The user will have the lesser hard disk space for its use.

DEMAND PAGING

- Demand Paging is a popular method of virtual memory management.
- In demand paging, the **pages of a process which are least used, get stored in the secondary memory.**
- A page is copied to the main memory when its demand is made, or page fault occurs. There are various page replacement algorithms which are used to determine the pages which will be replaced.



- A demand paging mechanism is very much similar to a paging system with swapping where processes stored in the secondary memory and pages are loaded only on demand, not in advance.
- So, when a context switch occurs, the OS never copy any of the old program's pages from the disk or any of the new program's pages into the main memory. Instead, it will start executing the new program after loading the first page and fetches the program's pages, which are referenced.
- During the program execution, if the program references a page that may not be available in the main memory because it was swapped, then the processor considers it as an invalid memory reference.
- That's because the page fault and transfers send control back from the program to the OS, which demands to store page back into the memory.

PAGE REPLACEMENT ALGORITHMS

- In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.
- **Page Fault –** A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Here, are some important Page replacement methods

- FIFO
- Optimal Algorithm
- LRU Page Replacement

FIFO Page Replacement

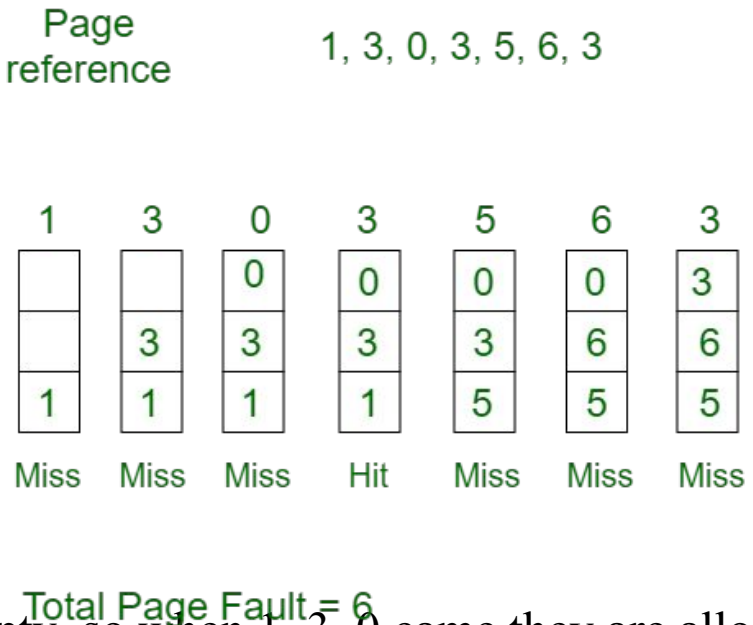
- FIFO (First-in-first-out) is a simple implementation method. In this method, memory selects the page for a replacement that has been in the virtual address of the memory for the longest time.

Features:

- Whenever a new page loaded, the page recently comes in the memory is removed. So, it is easy to decide which page requires to be removed as its identification number is always at the FIFO stack.
- The oldest page in the main memory is one that should be selected for replacement first.

Example-1

- Consider page reference string 1, 3, 0, 3, 5, 6 with 3-page frames. Find number of page faults.



- Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**
  when 3 comes, it is already in memory so —> **0 Page Faults.**
  Then 5 comes, it is not available in memory, so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**
  6 comes, it is also not available in memory, so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**
  Finally, when 3 come it is not available, so it replaces 0 **1-page fault**

LRU Page Replacement

- The full form of LRU is the Least Recently Used page. This method helps OS to find page usage over a short period of time. This algorithm should be implemented by associating a counter with an even- page.

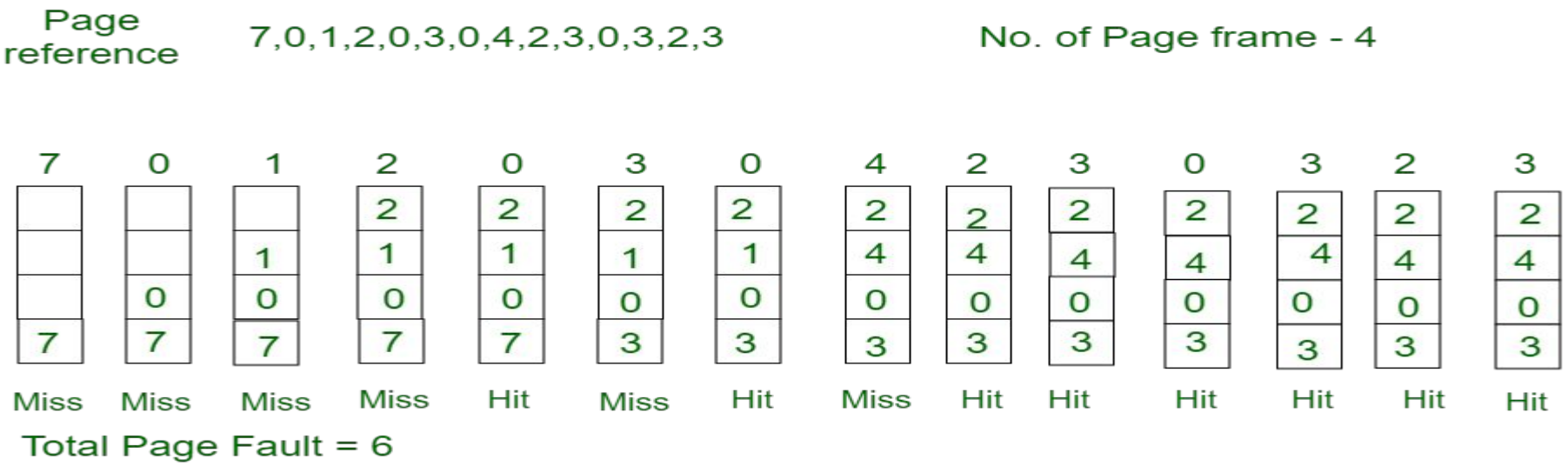
How does it work?

- Page, which has not been used for the longest time in the main memory, is the one that will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Features:

- The LRU replacement method has the highest count. This counter is also called aging registers, which specify their age and how much their associated pages should also be referenced.
- The page which hasn't been used for the longest time in the main memory is the one that should be selected for replacement.
- It also keeps a list and replaces pages by looking back into time.

- Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4-page frames. Find number of page faults.



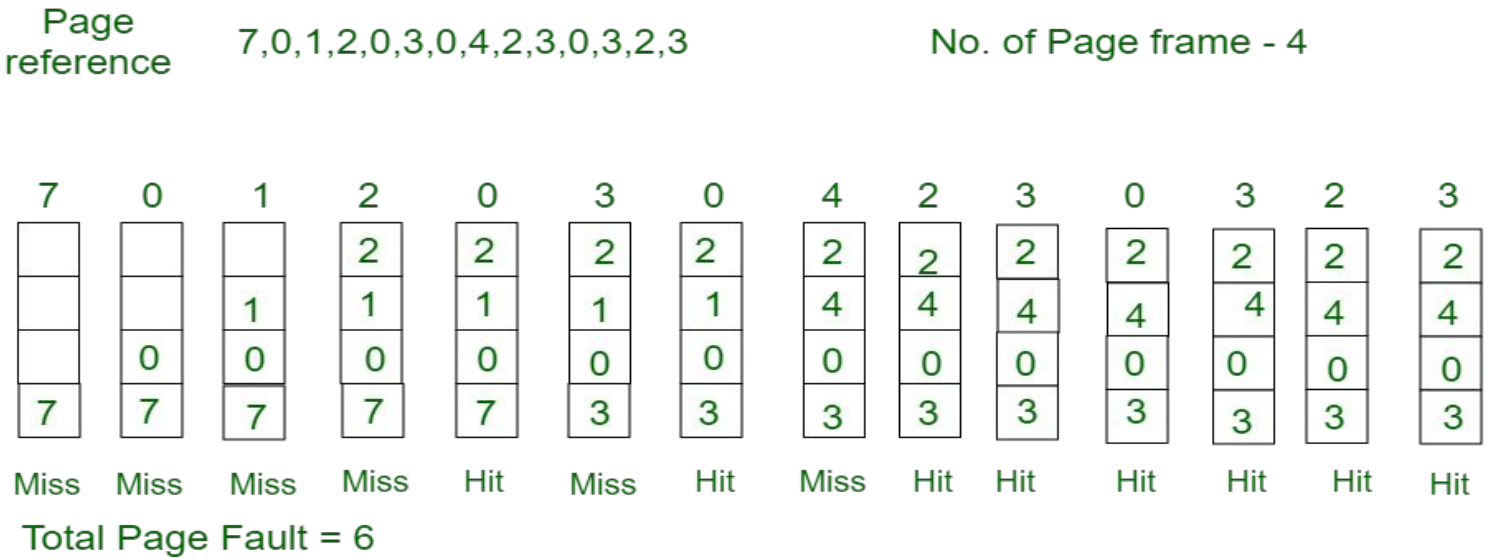
Here LRU has same number of page fault as optimal but it may differ according to question.

- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
  0 is already their so —> **0 Page fault.**
  when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
  0 is already in memory so —> **0 Page fault**.
  4 will takes place of 1 —> **1 Page Fault**
  Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Optimal Algorithm

- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.



- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
  0 is already there so —> **0 Page fault.**
  when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
  0 is already there so —> **0 Page fault.**.
  4 will takes place of 1 —> **1 Page Fault.**

SHELL INTRODUCTION & SHELL SCRIPTING

- **SHELL** is a program which provides the interface between the user and an operating system.
- When the user logs in OS starts a shell for user.
- **Kernel** controls all essential computer operations, and provides the restriction to hardware access, coordinates all executing utilities, and manages Resources between process.
- Using kernel only user can access utilities provided by operating system.

Types of Shell:

The C Shell –

- Denoted as csh
- Bill Joy created it at the University of California at Berkeley. It incorporated features such as aliases and command history. It includes helpful programming features like built-in arithmetic and C-like expression syntax.

In C shell:

- Command full-path name is /bin/csh,
- Non-root user default prompt is hostname %,
- Root user default prompt is hostname #.

The Bourne Shell –

- Denoted as sh
- It was written by Steve Bourne at AT&T Bell Labs. It is the original UNIX shell. It is faster and more preferred. It lacks features for interactive use like the ability to recall previous commands. It also lacks built-in arithmetic and logical expression handling. It is default shell for Solaris OS.
- For the Bourne shell the:
- Command full-path name is /bin/sh and /sbin/sh,
- Non-root user default prompt is $,
- Root user default prompt is #.

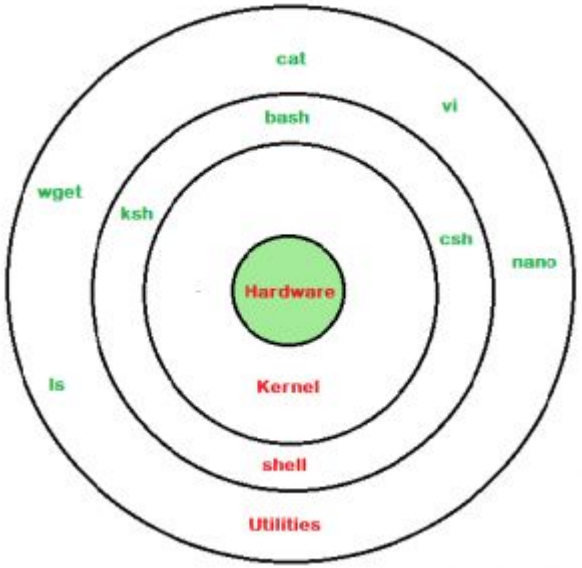
GNU Bourne-Again Shell –

- Denoted as bash
- It is compatible to the Bourne shell. It includes features from Korn and Bourne shell.

- For the GNU Bourne-Again shell the:
- Command full-path name is /bin/bash,
- Default prompt for a non-root user is bash-g.gg$
- (g.ggindicates the shell version number like bash-3.50$),
- Root user default prompt is bash-g.gg#.

INTRODUCTION TO LINUX SHELL AND SHELL SCRIPTING

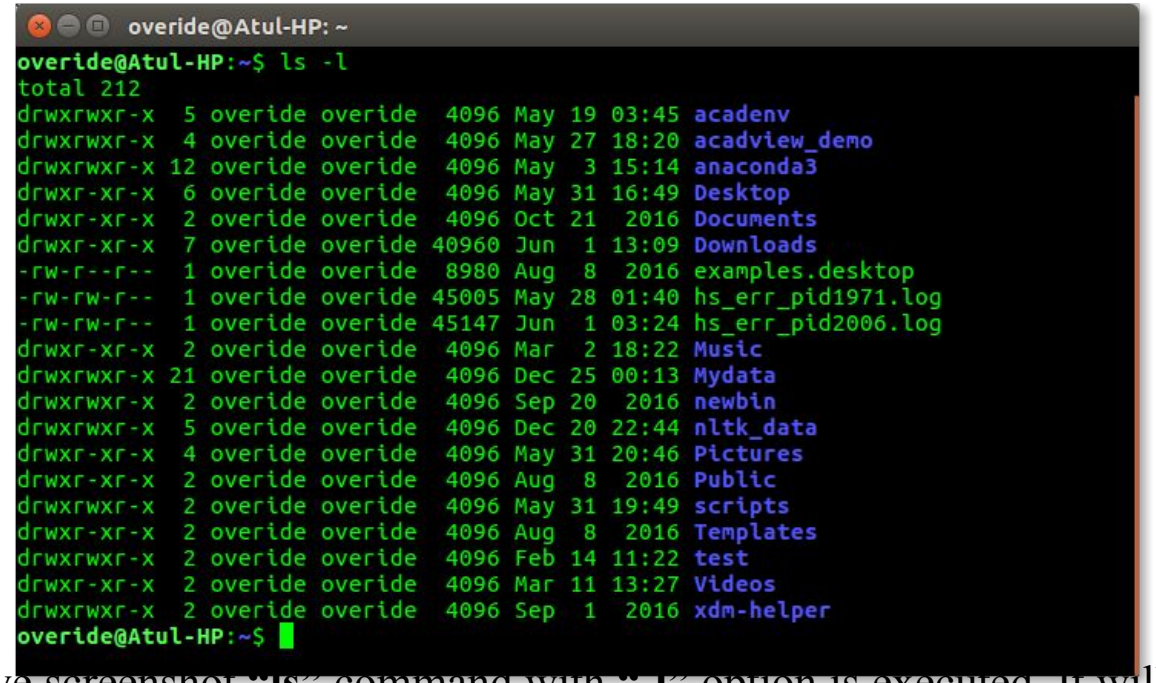
- A shell is special user program which provide an interface to user to use operating system services. Shell accept human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.



- Shell is broadly classified into two categories –
  - Command Line Shell
  - Graphical shell

Command Line Shell

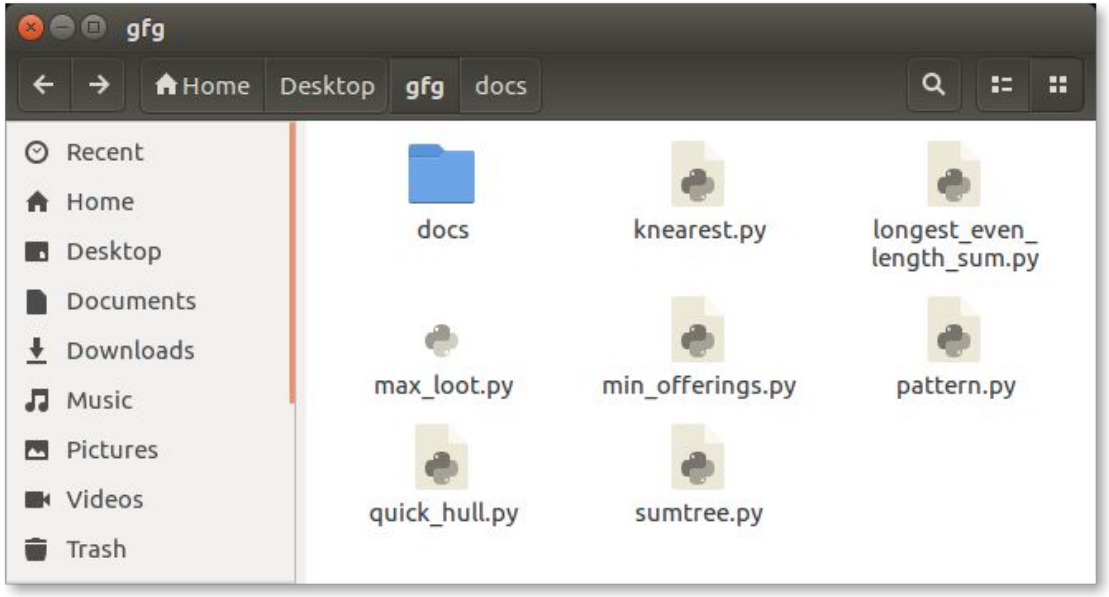
- Shell can be accessed by user using a command line interface. A special program called **Terminal** in linux/macOS or **Command Prompt** in Windows OS is provided to type in the human readable commands such as “cat”, “ls” etc. and then it is being execute. The result is then displayed on the terminal to the user.



- In above screenshot **“ls”** command with **“-l”** option is executed. It will list all the files in current working directory in long listing format.
  Working with command line shell is bit difficult for the beginners because it’s hard to memorize so many commands. It is very powerful; it allows user to store commands in a file and execute them together. This way any repetitive task can be easily automated. These files are usually called **batch files** in Windows and **Shell** Scripts in Linux/macOS systems.

Graphical Shells

- Graphical shells provide means for manipulating programs based on graphical user interface (GUI), by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows. Window OS or Ubuntu OS can be considered as good example which provide GUI to user for interacting with program. User do not need to type in command for every actions.A typical GUI in Ubuntu system –



- There are several shells are available for Linux systems like –
  - BASH (Bourne Again SHell) – It is most widely used shell in Linux systems. It is used as default login shell in Linux systems and in macOS. It can also be installed on Windows OS.
  - CSH (C SHell) – The C shell's syntax and usage are very similar to the C programming language.
  - KSH (Korn SHell) – The Korn Shell also was the base for the POSIX Shell standard specifications etc.

SHELL SCRIPTING

- Shell Scripting is an open-source computer program designed to be run by the **Unix/Linux shell.**
- **Shell** is a UNIX term for an interface between a user and an operating system service.
- Shell provides users with an interface and accepts human-readable commands into the system and executes those commands which can run automatically and give the program's output in a shell script.
- As shell can also take commands as input from file, we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called **Shell Scripts** or **Shell Programs**.
- Shell scripts are similar to the batch file in MS-DOS. Each shell script is saved with **.sh** file extension. eg. **myscript.sh**
- A shell script have syntax just like any other programming language. If you have any prior experience with any programming language like Python, C/C++ etc. it would be very easy to get started with it.

- A shell script comprises following elements –
  - Shell Keywords – if, else, break etc.
  - Shell commands – cd, ls, echo, pwd, touch etc.
  - Functions
  - Control flow – if..then..else, case and shell loops etc.

Why do we need shell scripts

- There are many reasons to write shell scripts –
  - To avoid repetitive work and automation
  - System admins use shell scripting for routine backups
  - System monitoring
  - Adding new functionality to the shell etc.

CONDITIONAL STATEMENTS | SHELL SCRIPT

- **Conditional Statements:** There are total 5 conditional statements which can be used in bash programming
- if statement
- if-else statement
- if..elif..else..fi statement (Else If ladder)
- if..then..else..if..then..fi..fi..(Nested if)
- switch statement

if-else statement

- If specified condition is not true in if part then else part will be execute.

Syntax

- if [expression]
- then
- statement1
- else
- statement2
- fi

switch statement

- case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern
- When a match is found all of the associated statements until the double semicolon (;;) is executed.
- A case will be terminated when the last command is executed.
- If there is no match, the exit status of the case is zero.

Syntax:

- case in
- Pattern 1) Statement 1;;
- Pattern n) Statement n;;
- esac

UTILITY PROGRAMS

Cut

- The cut command in UNIX takes the section from a file and outputs it to standard out.
- However, it does not delete any part of the file it extracts text from.
- cut is powerful in that it may accept multiple files for its standard input.
- -b: Select only the bytes specified. May be a single, set or range of bytes, separated by a comma.
- -c: Specify the number of characters from each line.
- -f: Extract a set of specified fields.
- -d: Used with the -f option. Use a specified delimiter rather than default tab.

$ cat test.txt

- doh re me fa so
- 1 2 3 4 5
- $ cut -f 3-5 test.txt
- me fa so
- 3 4 5
- $ cut -f 1,3-4 test.txt
- doh me fa
- 1 3 4

Paste

- The paste command is used to merge lines of files together. With this command, you can add one or more columns (or fields) of text to a file. There are two options you should be aware of:
- -d: Specify the delimiter to be used instead of tabs.
- -s: Append in serial instead of parallel. (Horizontal pasting instead of vertical.)
- $ cat names.txt
- Billy
- Bob
- Chase
- Jon
- Jonathan
- $ cat birthdates.txt
- 09/21/1992
- 08/12/1982
- 05/24/1999
- 04/23/1974
- 08/09/2001

$ paste -d ',' names.txt birthdates.txt
Billy,09/21/1992
Bob,08/12/1982
Chase,05/24/1999
Jon,04/23/1974
Jonathan,08/09/2001

- $ paste -d ',' 5names.txt 3birthdates.txt
- Billy,05/24/1999
- Bob,04/23/1974
- Chase,08/09/2001
- Jon,
- Jonathan,

Join

- In short, join takes a common column between two tables, and joins them together based on that attribute.

- -t: Specify a delimiter
- -1 n: Use the nth column as the join key for the first column.
- -2 n: Use the nth column as the join key for the second column.
- -a n: Also print the unprintable lines from n, where n is 1 or 2 (first or second file).

GREP COMMAND IN UNIX/LINUX

- The grep filter searches a file for a particular pattern of characters and displays all lines that contain that pattern.
- The pattern that is searched in the file is referred to as the regular expression (grep stands for globally search for regular expression and print out).

Syntax:

- grep [options] pattern [files]

Options Description

- -c : This prints only a count of the lines that match a pattern
- -h : Display the matched lines, but do not display the filenames.
- -i : Ignores, case for matching
- -l : Displays list of a filenames only.
- -n : Display the matched lines and their line numbers.
- -v : This prints out all the lines that do not matches the pattern
- -e exp : Specifies expression with this option. Can use multiple times.
- -f file : Takes patterns from file, one per line.
- -E : Treats pattern as an extended regular expression (ERE)
- -w : Match whole word
- -o : Print only the matched parts of a matching line, with each such part on a separate output line.

Sample Commands

Consider the below file as an input.

$cat > geekfile.txt

- unix is great os. unix is opensource. unix is free os.
- learn operating system.
- Unix linux which one you choose.
- uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

1. Case insensitive search : The -i option enables to search for a string case insensitively in the give file. It matches the words like “UNIX”, “Unix”, “unix”.

$grep -i "UNix" geekfile.txt

Output:

unix is great os. unix is opensource. unix is free os.

Unix linux which one you choose.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

2. Displaying the count of number of matches : We can find the number of lines that matches the given string/pattern

$grep -c "unix" geekfile.txt

Output:

2

3. Display the file names that matches the pattern : We can just display the files that contains the given string/pattern.

$grep -l "unix" *

or

$grep -l "unix" f1.txt f2.txt f3.xt f4.txt

Output:

geekfile.txt

4. Checking for the whole words in a file : By default, grep matches the given string/pattern even if it found as a substring in a file. The -w option to grep makes it match only the whole words.

$ grep -w "unix" geekfile.txt

Output:

unix is great os. unix is opensource. unix is free os.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

5. Displaying only the matched pattern : By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

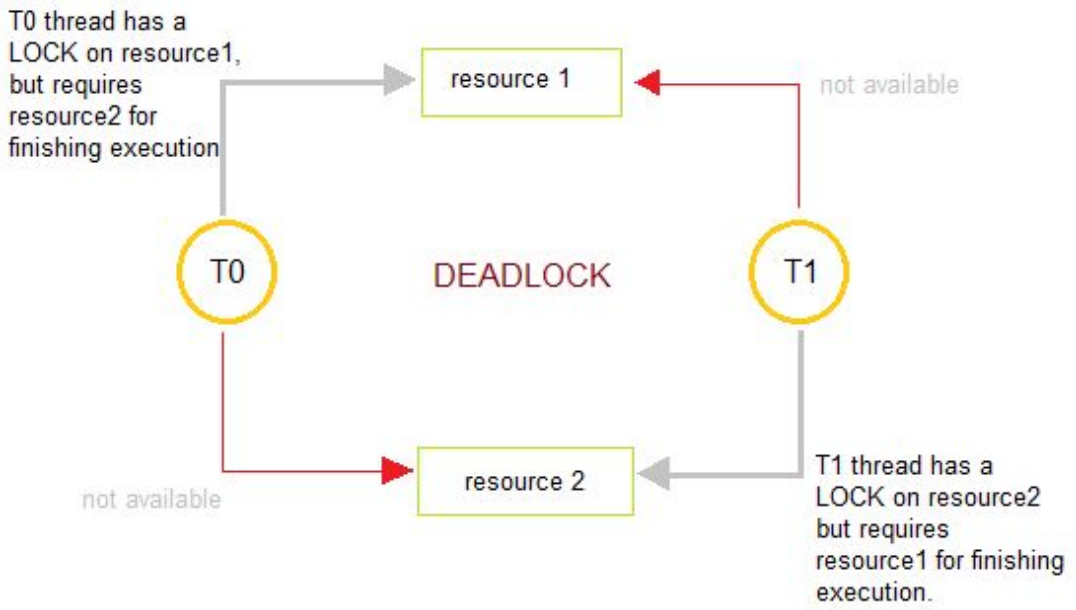
$ grep -o "unix" geekfile.txt

Output:

```
unix unix unix unix unix unix
```

DEADLOCKS

- It is a situation where 2 processes sharing the same resource are effectively preventing each other from accessing the resources.



- In the above figure, process T0 has resource1, it requires resource2 in order to finish its execution.
- Similarly, process T1 has resource2 and it also needs to acquire resource1 to finish its execution. In this way, T0 and T1 are in a deadlock because each of them needs the resource of others to complete their execution but neither of them is willing to give up their resources.

Basically, in the Normal mode of Operation utilization of resources by a process is in the following sequence:

- **Request:**
  Firstly, the process requests the resource.
- In a case, if the request cannot be granted immediately(e.g: resource is being used by any other process), then the requesting process must wait until it can acquire the resource.
- **Use:**
  The Process can operate on the resource (e.g: if the resource is a printer then in that case process can print on the printer).
- **Release:**
  The Process releases the resource.

STARVATION VS DEADLOCK

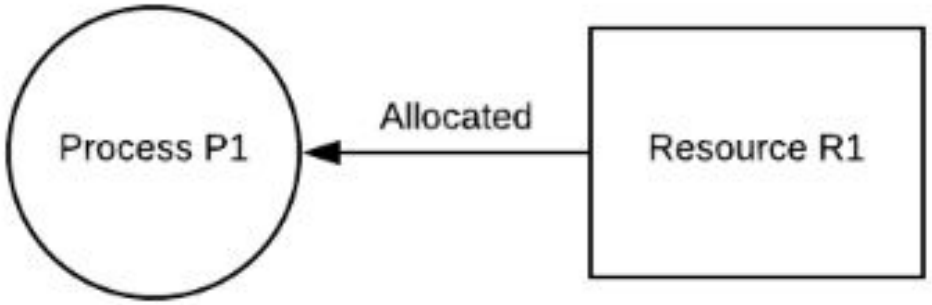
Starvation	Deadlock
When all the low priority processes got blocked, while the high priority processes execute then this situation is termed as Starvation.	Deadlock is a situation that occurs when one of the processes got blocked and no process proceeds.
Starvation is a **long waiting**, but it is not an infinite process.	Deadlock is an **infinite waiting**.
It is not necessary that every starvation is a deadlock.	There is starvation in every deadlock.
It occurs due to the uncontrolled priority and resource management.	Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.

NECESSARY CONDITIONS FOR DEADLOCK(CHARACTERIZATION)

- The deadlock situation can only arise if all the following four conditions hold simultaneously:

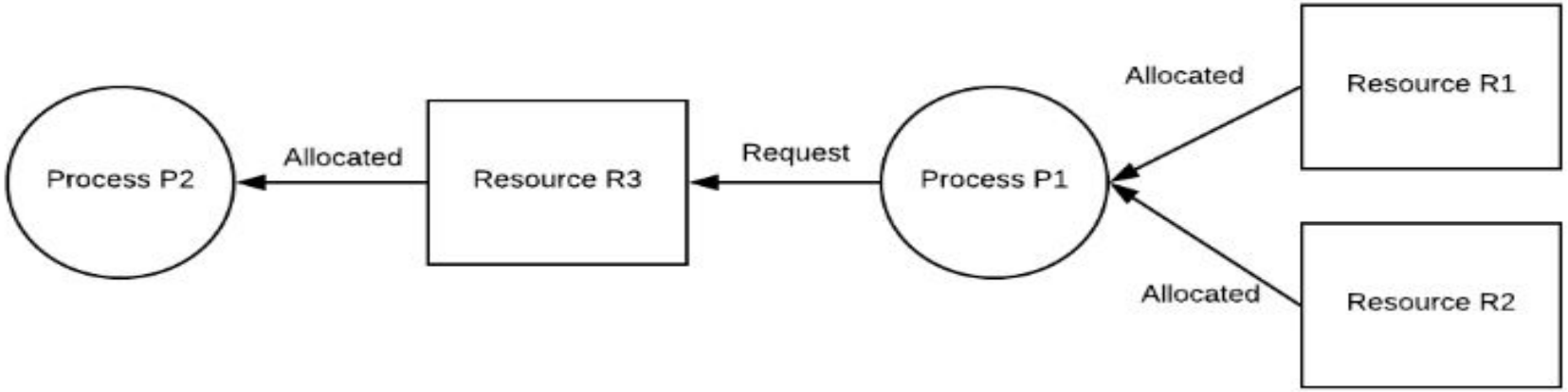
Mutual Exclusion

- A resource can only be shared in mutually exclusive manner.
- It implies, if two process cannot use the same resource at the same time.
- There should be a resource that can only be held by one process at a time.
- In the diagram below, there is a single instance of resource R1, and it is held by process P1 only. It means that if the resource(like a printer) is being used by any process and some other process comes then it must wait for the resources to be released.



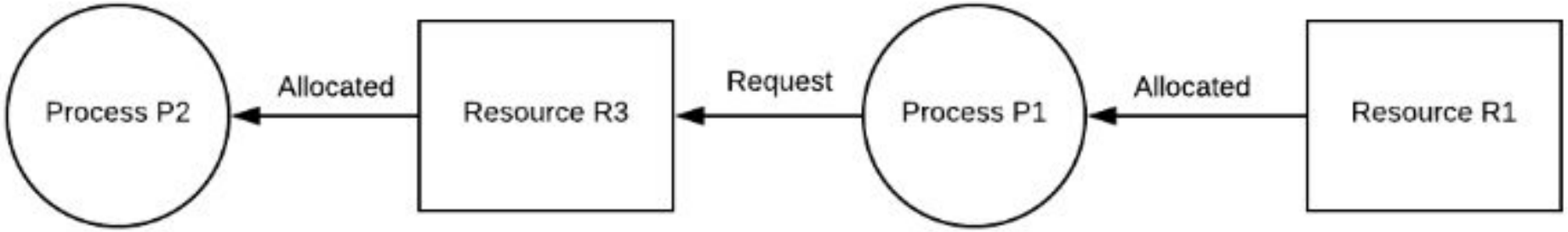
Hold and wait

- The process is holding some resources and is waiting for the other resources.



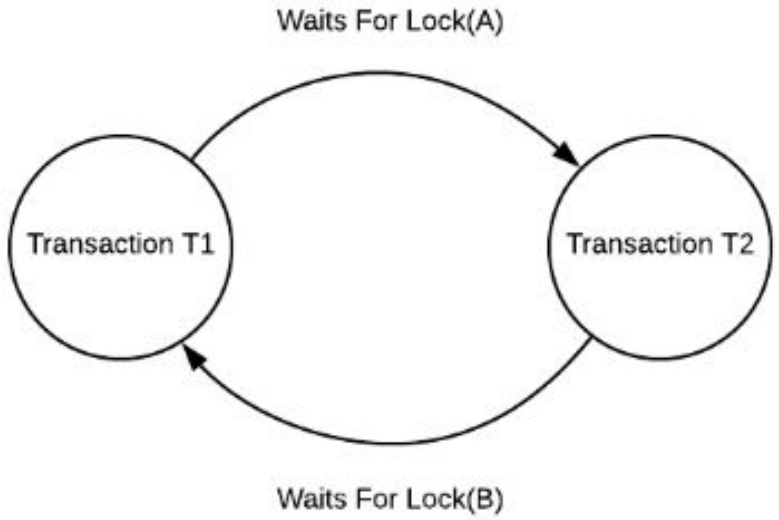
No preemption

- If a process is holding a resource, then its resources can not be forcibly taken from it. In the diagram below, process P1 cannot preempt resource R3 from Process P2. It will only be released when P2 relinquishes it voluntarily after its execution is complete.



Circular wait

- The process is waiting for the resource which is being held by the next process and this forms a chain of waiting.
- Process P1, P2, P3 are forming a chain and waiting for the resource being held by each other.



If the above four conditions hold simultaneously then a Deadlock may occur.

DEADLOCK HANDLING METHODS

There are three classical approaches for deadlock handling, namely −

- Deadlock prevention
- Deadlock avoidance
- Deadlock detection

Deadlock prevention

- There is a very famous proverb. **“Prevention is better than cure”**.
- So, we try to prevent the deadlock from happening.
- As we know here are four necessary conditions for deadlock to occur. We will prevent any or all of the conditions from holding true.

Removing Mutual Exclusion:

- The resources which we use should be sharable.
- Like CPU, it can be used by more than one process at a time. But this not always the case. Some resources are non-sharable like printers.
- So, if the resources are non-sharable then this condition cannot be removed and hence the deadlock can't be prevented.

Removing No Preemption:

- Here, we will forcibly preempt or stop a process and thus, the resources held by this process will be released.
- Now, these resources can be used by other process and deadlock can be removed. In the above example, we can stop the process P1 and thus, P1 will release the resource R1. Now, this R1 resource can be used by P2 to complete its execution and hence, the deadlock is removed.

Removing Hold & Wait Condition:

- Before a process starts, we will give all the resources required by it.
- Thus, a deadlock situation will never arise as it will never have to wait.

Removing Circular Wait:

- To avoid the circular wait, the resource may be numbered, and each process can request the resource in increasing order of these numbers.
- Example: Assume that R5 resource is allocated to P1, if next time P1 asks for R4, R3 that are lesser than R5; then such request will not be granted. Only the request for resources that are more than R5 will be granted.
- This algorithm if implemented increases the complexity and leads to poor resource utilization.

DEADLOCK AVOIDANCE

Basic facts:

- If a system is in safe state, then no deadlock.
- If a system is in unsafe state, then there is a possibility of deadlock.

- Deadlock Avoidance ensures that a system will never enter an unsafe state.
- **Bankers Algorithms** is also a deadlock avoidance strategy.

BANKER'S ALGORITHM

- It is a banker algorithm used to **avoid deadlock** and **allocate resources** safely to each process in the computer system.
- The '**S-State**' examines all possible tests or activities before deciding whether the allocation should be allowed to each process.
- It also helps the operating system to successfully share the resources between all the processes.
- The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources.

Following are the important data structures terms applied in the banker's algorithm as follows:

- Suppose n is the number of processes, and m is the number of each type of resource used in a computer system.
- **Available**: It is an array of length 'm' that defines each type of resource available in the system. When Available[j] = K, means that 'K' instances of Resources type R[j] are available in the system.
- **Max:** It is a [n x m] matrix that indicates each process P[i] can store the maximum number of resources R[j] (each type) in a system.
- **Allocation:** It is a matrix of m x n orders that indicates the type of resources currently allocated to each process in the system. When Allocation [i, j] = K, it means that process P[i] is currently allocated K instances of Resources type R[j] in the system.
- **Need:** It is an M x N matrix sequence representing the number of remaining resources for each process. When the Need[i] [j] = k, then process P[i] may require K more instances of resources type Rj to complete the assigned work.
  Nedd[i][j] = Max[i][j] - Allocation[i][j].
- **Finish**: It is the vector of the order **m**. It includes a Boolean value (true/false) indicating whether the process has been allocated to the requested resources, and all resources have been released after finishing its task.

- The Banker's Algorithm is the combination of the safety algorithm and the resource request algorithm to control the processes and avoid deadlock in a system:

Safety Algorithm

- It is a safety algorithm used to check whether or not a system is in a safe state or follows the safe sequence in a banker's algorithm:

1. There are two vectors **Wok** and **Finish** of length m and n in a safety algorithm.

- Initialize: Work = Available
  Finish[i] = false; for I = 0, 1, 2, 3, 4… n - 1.

2. Check the availability status for each type of resources [i], such as:

- Need[i] <= Work
  Finish[i] == false
  If the i does not exist, go to step 4.

3. Work = Work +Allocation(i) // to get new resource allocation

- Finish[i] = true
- Go to step 2 to check the status of resource availability for the next process.

4. If Finish[i] == true; it means that the system is safe for all processes.

Resource Request Algorithm

- A resource request algorithm checks how a system will behave when a process makes each type of resource request in a system as a request matrix.
- Let create a resource request array R[i] for each process P[i]. If the Resource Request$_i$ [j] equal to 'K', which means the process P[i] requires 'k' instances of Resources type R[j] in the system.

1. When the number of **requested resources** of each type is less than the **Need** resources, go to step 2 and if the condition fails, which means that the process P[i] exceeds its maximum claim for the resource. As the expression suggests:

- If Request(i) <= Need
  Go to step 2;

2. And when the number of requested resources of each type is less than the available resource for each process, go to step (3). As the expression suggests:

- If Request(i) <= Available
  Else Process P[i] must wait for the resource since it is not available for use.

3. When the requested resource is allocated to the process by changing state:

- Available = Available - Request
  Allocation(i) = Allocation(i) + Request (i)
  Need$_i$ = Need$_i$ - Request$_i$
- When the resource allocation state is safe, its resources are allocated to the process P(i). And if the new state is unsafe, the Process P (i) has to wait for each type of Request R(i) and restore the old resource-allocation state

- **Example:** Consider a system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	3			

1. Determine if the system is safe or not.
2. What will happen if the resource request (1, 0, 0) for process P1 can the system accept this request immediately?

Context of the need matrix is as follows:

- Need [i] = Max [i] - Allocation [i]
  Need for P1: (7, 5, 3) - (0, 1, 0) = 7, 4, 3
  Need for P2: (3, 2, 2) - (2, 0, 0) = 1, 2, 2
  Need for P3: (9, 0, 2) - (3, 0, 2) = 6, 0, 0
  Need for P4: (2, 2, 2) - (2, 1, 1) = 0, 1, 1
  Need for P5: (4, 3, 3) - (0, 0, 2) = 4, 3, 1

Process	Need A	B	C
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

Apply the Banker's Algorithm:

- Available Resources of A, B and C are 3, 3, and 2.
- Now we check if each type of resource request is available for each process.

Step 1: For Process P1:

- Need <= Available
- 7, 4, 3 <= 3, 3, 2 condition is **false**.
- **So, we examine another process, P2.**

Step 2: For Process P2:

- Need <= Available
- 1, 2, 2 <= 3, 3, 2 condition **true**
- New available = available + Allocation
- (3, 3, 2) + (2, 0, 0) => 5, 3, 2
- **Similarly, we examine another process P3.**

Step 3: For Process P3:

- P3 Need <= Available
- 6, 0, 0 < = 5, 3, 2 condition is **false**.
- **Similarly, we examine another process, P4.**

Step 4: For Process P4:

- P4 Need <= Available
- 0, 1, 1 <= 5, 3, 2 condition is **true**
- New Available resource = Available + Allocation
- 5, 3, 2 + 2, 1, 1 => 7, 4, 3
- **Similarly, we examine another process P5.**

Step 5: For Process P5:

- P5 Need <= Available
- 4, 3, 1 <= 7, 4, 3 condition is **true**
- New available resource = Available + Allocation
- 7, 4, 3 + 0, 0, 2 => 7, 4, 5
- Now, we again examine each type of resource request for processes P1 and P3.

Step 6: For Process P1:

- P1 Need <= Available
- 7, 4, 3 <= 7, 4, 5 condition is **true**
- New Available Resource = Available + Allocation
- 7, 4, 5 + 0, 1, 0 => 7, 5, 5
- **So, we examine another process P2.**

Step 7: For Process P3:

- P3 Need <= Available
- 6, 0, 0 <= 7, 5, 5 condition is true
- New Available Resource = Available + Allocation
- 7, 5, 5 + 3, 0, 2 => 10, 5, 7
- **Hence, we execute the banker's algorithm to find the safe state and the safe sequence like P2, P4, P5, P1 and P3.**

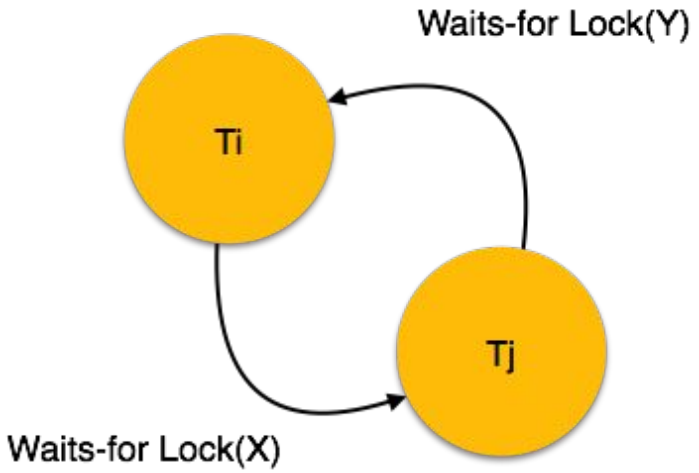
For granting the Request (1, 0, 2), first we have to check that **Request <= Available**, that is (1, 0, 2) <= (3, 3, 2), since the condition is true. So the process P1 gets the request immediately.

DEADLOCK DETECTION

- If deadlock prevention and avoidance are not done properly then the system may enter a deadlock state. So, we need to detect the deadlock and recover it.
- In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.



RECOVERY FROM DEADLOCK

There are three basic approaches to recovery from deadlock:

- Inform the system operator and allow him/her to take manual intervention.
- Terminate one or more processes involved in the deadlock
- Preempt resources.

Process Termination

- Two basic approaches, both of which recover resources allocated to terminated processes:
  - Terminate all processes involved in the deadlock. This solves the deadlock problem.
  - Terminate processes one by one until the deadlock is broken. This is more conservative but requires doing deadlock detection after each step.

Resource Preemption

- When preempting resources to relieve deadlock, there are three important issues to be addressed:
  - **Selecting a victim** - Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.
  - **Rollback** - Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately, it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning. (I.e., abort the process and make it start over.)
  - **Starvation** - How do you guarantee that a process won't starve because its resources are constantly being preempted? One option would be to use a priority system and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.