Uses of operating systems to manage the functionality of computers

- Why do we need an OS in a computer?

```
Not every computer has an OS. Operating systems are only used when its required to manage differnet tasks. If there is only one task to manage (fans, refrigerators) there is no need of an OS
```

- What is an Operating System?

```
An OS is a program that controls the execution of application programs and acts as an interface between applications and the computer hardware.
```

- Booting process of the OS

```
1. POST (Power on self start) - check for peripheral devices
2. BIOS in ROM reads Bootable instructions from CMOS (Complementry Metal-Oxide Semiconductor)
3. Finding the bootable disc and the sector to load the OS
4. Bootstrap loader loads the OS from the hard disk
```

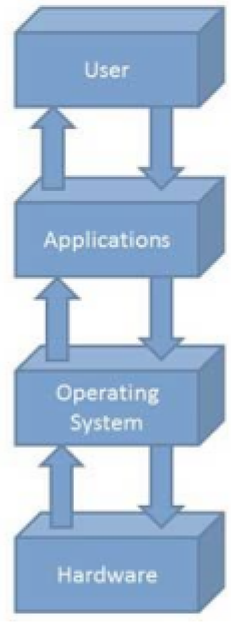
Difference between general purpose computers and embedded systems

General purpose	Embeded systems
Upgrades are done	No upgrades are done
Complicated user interactions	Simple user interactions
More resources are used	Less resources are used

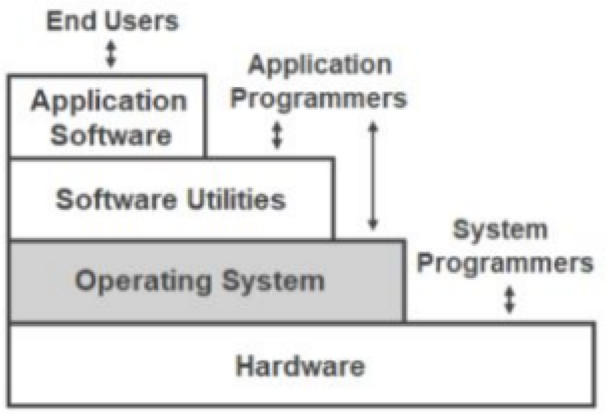
There are 3 main roles that an OS should provide.

1. **OS provides a virtual machine**
   - hides complex hardware details
   - provides a nice interface to application software and end users.
2. **OS manages computing resources (software and hardware)**
   - Keeps track of resource usage
   - Grants/ revokes permissions for resources (protection of program from other programs)
3. **OS allow executing applications**

Interactions between hardware components



How we interact with the OS

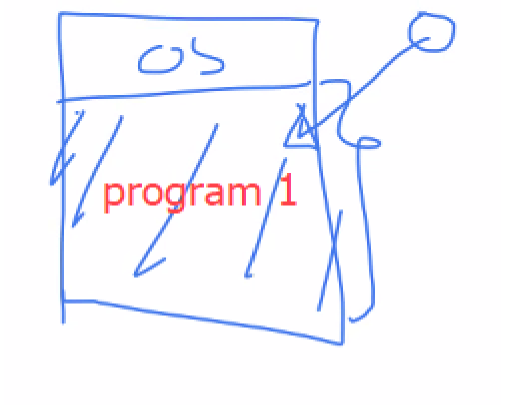


Evolution of OS

1. No OS (late 1940s – mid 1950s)
2. Simple batch system (mid 1950s)
3. Multi-programmed batch systems (1960s)
4. Time sharing systems (1960s)
5. Multi-programmed time sharing systems

No OS

- Used in the **1st generation computers**
- Programmers and users directly interacted with the hardware
- **Manual program scheduling**
  - We have to schedule manually which program execute first and next
- **Uniprogramming**
  - executing one program at a time



- **Serial Processing**
  - processing programs one after another (still only one program can be loaded to the RAM at one time)
  - due to this loading time is high (loaded bit by bit)

Problems

- One of the main problem is that the **processor is sat idle when loading programs and doing I/O**
- Having to interact with computer hardware directly.

Simple Batch System

- Used in the **2nd generation of computers.**
- These computers were built to maximize the processor utilization (By solving the problem had with No OS)
- First batch OS was **developed by General Motors** company
- OS loads and executes a new program from a batch of programs once the current program is ended.
- Batch of programs were recorded in a magnetic tape
- output of the current executed program is written to another tape.

Features of simple batch system

- No direct access to computer hardware
- Serial processing
- Uniprogramming
- High response time
- Processor sat idle during I/O

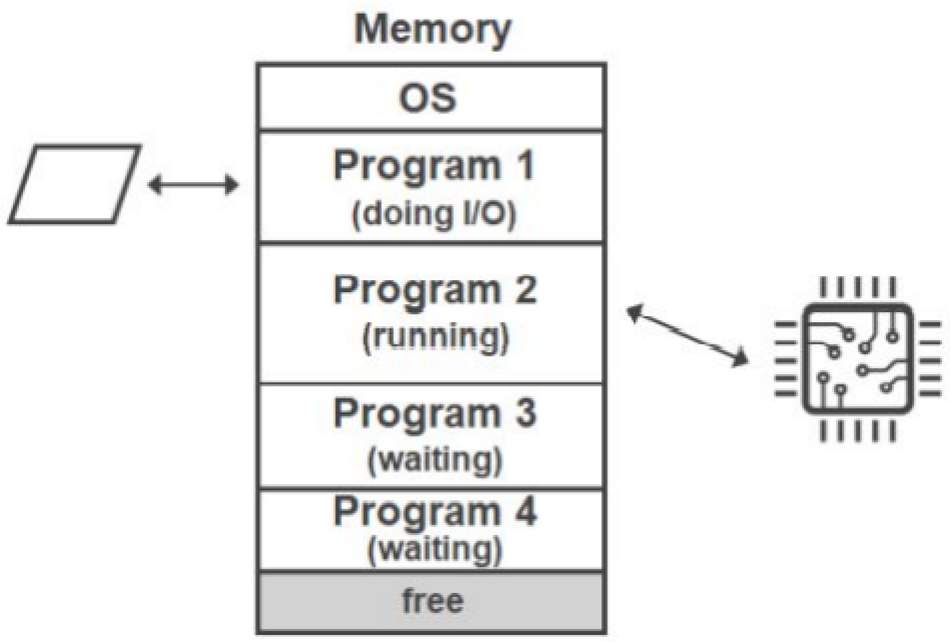
Problems

- Compared to the No OS, **loading time problem** and **direct access** problems are solved in this system. Still, the problem of processor sat idle when doing I/O remains. To resolve this next system was implemented.

Multi-Programmed batch Systems

- **Central theme of modern OS**

- **Introduced in 3rd generation** to minimize the processor idle time during I/O.
- **Memory is partitioned** to hold multiple programs
- When current program waiting for I/O, OS switches processor to execute another program in memory.



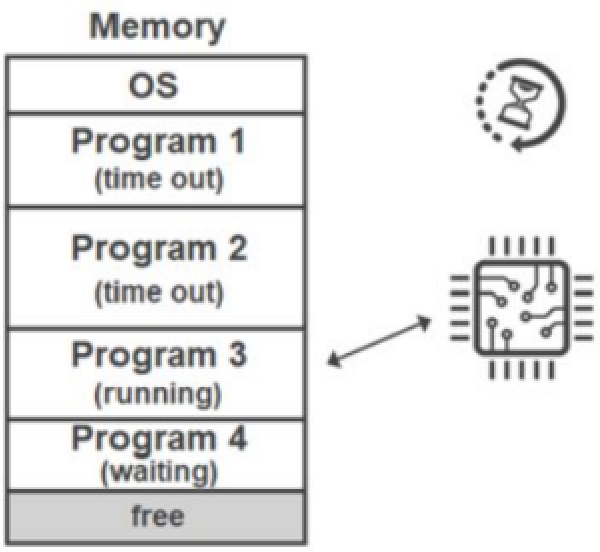
- If memory is large enough to hold more programs, processor could keep 100% busy and can maximize the processor utilization

Problems

- Still there is a problem with high response time(introduced time-sharing system as a solution)

Time Sharing System

- Introduced to **minimize the user response time** and maximize the user interaction during program execution **in 3rd generation of computers**.
- OS switches processor from one program to another after a certain time quantum (**context switching**).
- If the certain program doesn't finish execution in the time given, it will have to wait till the next time comes.
- Enables to share the processor time among multiple programs

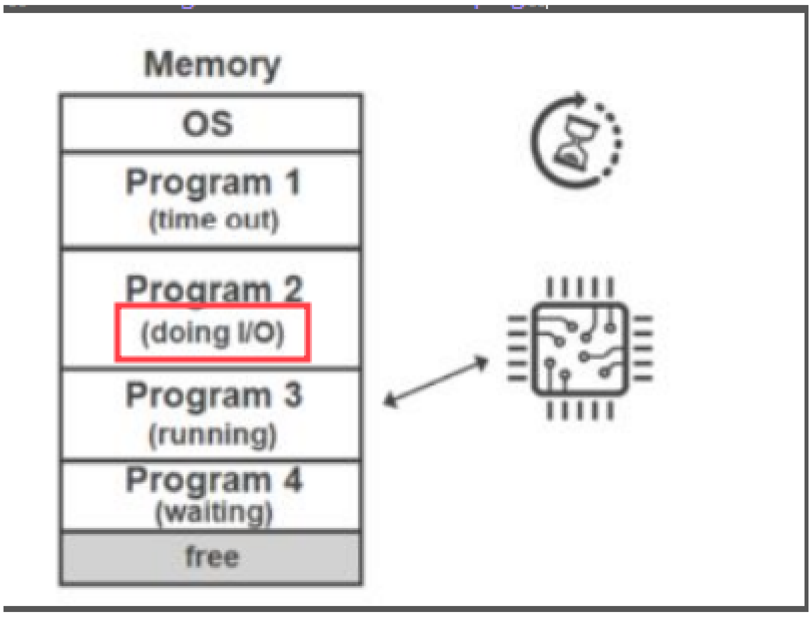


Advantages of time-sharing

- Each program gets equal opportunity
- Less chance of duplication of software
- CPU idle time can be reduced.

Multi Programming Time Sharing Systems

- This is kind of the addition of both concepts used in **muti-programmed batch system** and **time sharing system**
- **When current program wants to do I/O, OS switches processor to execute another program in memory.**



- Used in **Modern OS.**
- Context switching is used.

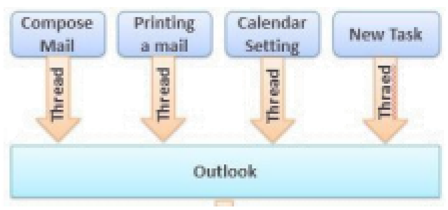
So in theory these systems were built to get the most usage out of the CPU one after the other.

Classification of Operating Systems

1. Different types of Operating Systems (Based on the users)
   - Single user - Facilitates single user to use the system at a time
     - Single-task - For Palm handheld computers, Ms-DOS
     - Multi-task - Microsoft's Windows, Apple MacOS
   - Multi User-Facilitates multiple users to use the system at a time
     - Multi-task - Unix server
2. Different types of Operating Systems (Based on Number of tasks)
   - Single Task – Executes only one program at a time
   - Multi Task - Executes multiple programs at a time

Multi-threading

> A thread is also called a sub process. A process can use threads to divide its processes to execute them simultaneously.



Real Time systems

- OS is designed to run applications with **very precise** timing and with a high **degree of reliability**.
- The main objective of real-time operating systems is **their quick and predictable response to events**.
- Examples:
  - Airbag control system in a car
  - Medical equipments
  - Nuclear plant control system

02 - Explores how an operating system manages directories & folders and files in computers

Explores how an operating system manages directories/folders and files in computers.

Files Management

Files

- File is a named collection of data stored in secondary storage
- There are 2 views of a file.
  1. Logical View (User's view) - As a collection of records

2. Physical View (OS's view) - Whether blocks of secondary storage space either allocated or not

Directories

- A directory is **a special type of file** that contains other files

File Management System

> A File Management System is a system that allows you to create, delete, move, rename and other actions to manage files properly.

File Systems

> A file system is used to control how data is stored and retrieved

- There are different types of file systems
  1. FAT (File Allocation Table)
  2. NTFS (New Technology File System)
  3. ReFS file system (Resilient File System)
     - developed by Microsoft as a **new generation file system** and it requires Windows 8 or Windows server 2012

FAT

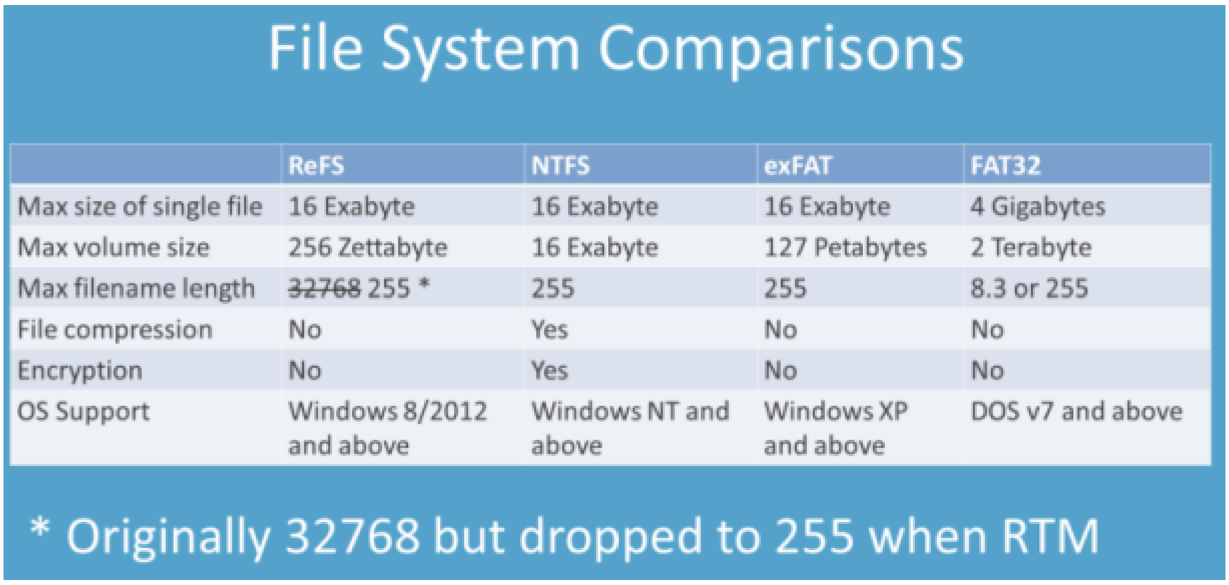
Features of FAT file system

- inherited from old DOS
- can only have a filename up to 8 characters
- can only have portions of size 4 GB
- Disk can get fragmented
- Two types of FAT file systems
  - FAT32
    - FAT32 can be read in just about any OS
  - exFAT
    - Microsoft file systems (**proprietary** and **patented**)

NTFS

Features of NTFS file system

- It provides security for both local and remote users.
- It supports 255 characters long file name.
- Partition size can be up to 16 Exabyte.
- Supports file compression.
- Lesser possibility of fragmentation.
- **proprietary file system** developed by Microsoft. This is improvement of FAT.



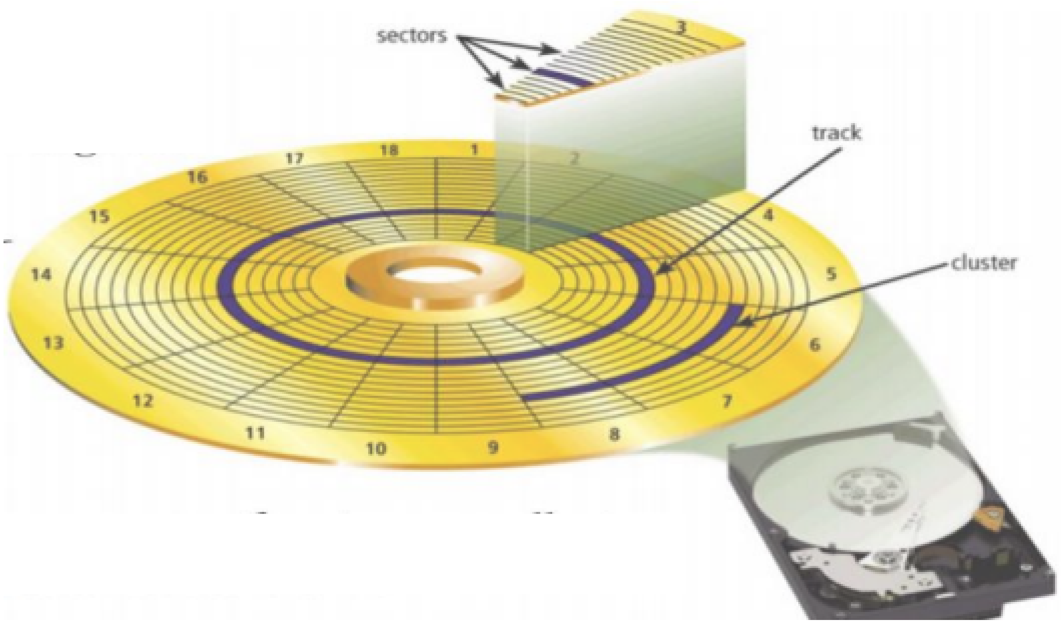
File System Comparisons

	ReFS	NTFS	exFAT	FAT32
Max size of single file	16 Exabyte	16 Exabyte	16 Exabyte	4 Gigabytes
Max volume size	256 Zettabyte	16 Exabyte	127 Petabytes	2 Terabyte
Max filename length	~~32768~~ 255 *	255	255	8.3 or 255
File compression	No	Yes	No	No
Encryption	No	Yes	No	No
OS Support	Windows 8/2012 and above	Windows NT and above	Windows XP and above	DOS v7 and above

* Originally 32768 but dropped to 255 when RTM

File Storage Management

- On secondary storage, stored data is organised using
  - Tracks - rings on platters
  - Sectors - tracks are devided into sectors
  - Cluster - a group of sectors, the smallest unit (512 bytes) allocated to store a file



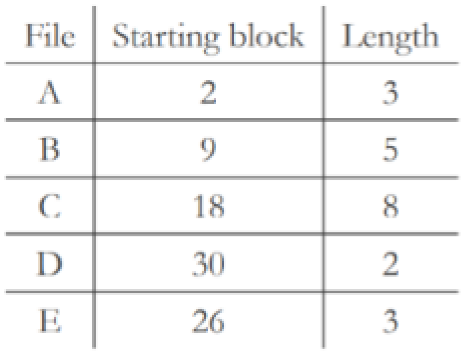
• OS uses a data structure called **File Allocation Table** (FAT) to keep the track of allocated and free blocks.

File Allocation Methods

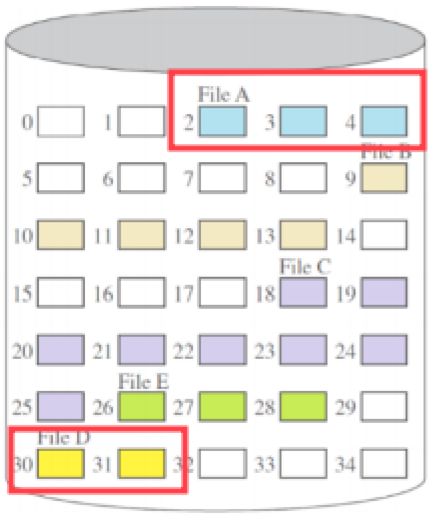
There are 3 main methods.
1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

Contiguous Allocation

- A single contiguous set of blocks is allocated to a file at the time of file creation. All the blocks are allocated closer to each other



File	Starting block	Length
A	2	3
B	9	5
C	18	8
D	30	2
E	26	3



Strengths

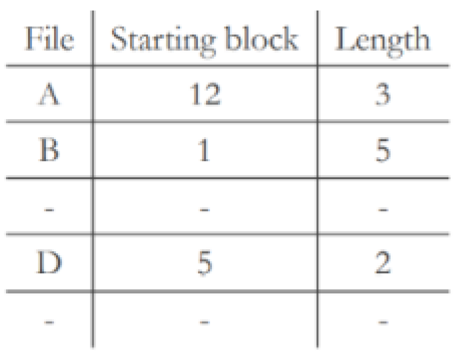
- Simple
- Easy Access

Weaknesses

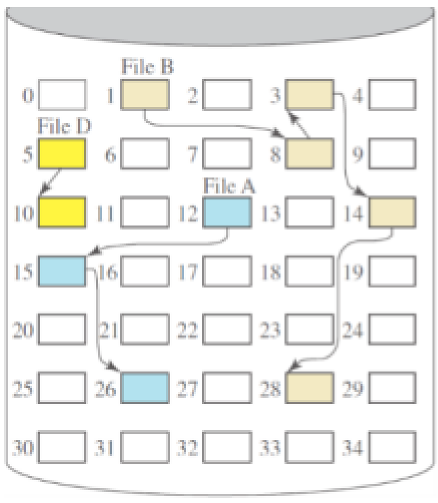
- File size should be known at the time of file creation.
- Extending file size is difficult
- External fragmentation(unusable free blocks between allocation)
  - This can be resolved by using defragmentation, but that results in system overhead (using the other resources unwantedly)

Linked Allocation

- This is kinda the opposite of the contiguous allocation.
- Inside each block,
  - the contents of the relevant file,
  - a link maintained to point to where the next block of the file is present.



File	Starting block	Length
A	12	3
B	1	5
-	-	-
D	5	2
-	-	-

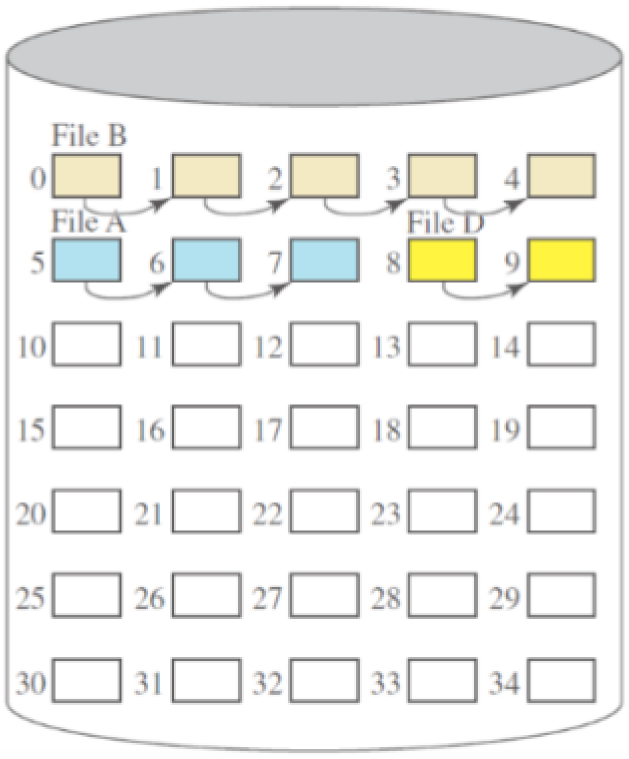


Strength

- No need to know the file size at the time of creation
- No external fragmentation to worry about because block by block is allocated at a time as needed.
- Files can grow dynamically and easily

Weaknesses

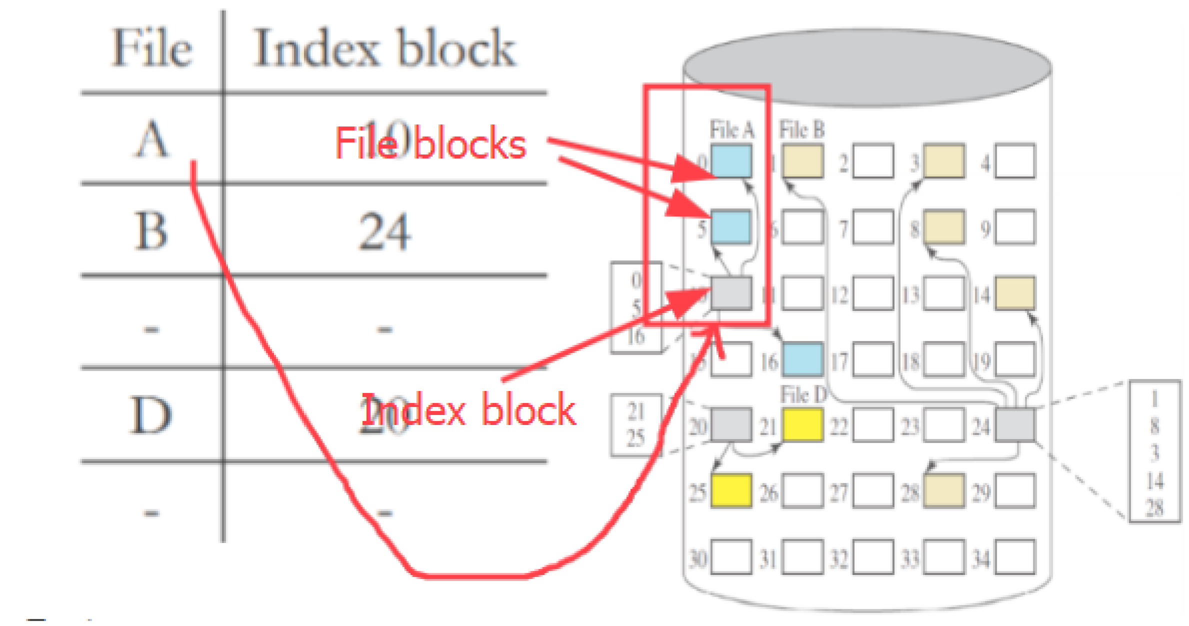
- Slow, many seeks to access the file
- Periodic consolidation can solve this but adds system overhead



By doing this, we can access the file fast. Even if its edited and the size gets large, that extended content can be saved in some place else and then again can be defragmented to make them closer in order.

Indexed Allocation

- FAT contains entry to index block of each file.
- The index block has entries for each block allocated to file
- The index table is also saved in a block/s
- Example:
  - UNIX file system - used indexed allocation with contiguous blocks
  - NTFS — used indexed allocation method



File	Index block
A	10
B	24
-	-
D	20
-	-

Features

- No need to know the file size at the time of creation
- Files can grow dynamically, and the index is updated accordingly
- File ends at nil pointer (empty index block)
- No external fragmentation
- Each index block contains 2 data
  - indexes of the blocks that contain the contents of the relevant file
  - pointer to next index block
- Using contiguous blocks in combination with this can **save the seek time**

The Secondary storage

This storage is typically used to store

- Source programs
- Executable programs
- Temporary data

Maintenance of Secondary storage

Defragmentation

```
This is a **process that reduces the amount of fragmentation.**
```

- Simply said, removing the spaces between the saved files.

Disk Cleanup

- Removing unnecessary files, including temporary internet files
- Allows you to empty the Recycle Bin, delete temporary files, and delete thumbnails

Disk formatting

```
Formatting is the process of preparing a data storage device for initial use,
which may also create one or more new file systems.
```

> As file deletion is done by the operating system, data on a disk are not fully erased during every high-level format. Instead, links to the files are deleted and the area on the disk containing the data is retains until it is overwritten

03 - Explores how an operating system manages processes in computers

Explores how an operating system manages processes in computers.

```
A Process is a program in execution or a program that can be assigned to and
executed by a processor.
```

Process and Program

Process is not a program. A program may have many processes.

A process consists of,

- Executable codes
- Data needed for execution
- Execution context

Similarities

- Both the program and process have Executable codes

Dis-similarities

Process	Program
Active	Passive
Exists for a limited time	Exists for longer
Have both data and instructions	Have only instructions

Interrupt Handling

```
Interrupt is an event that alters the sequence of execution of processes.
```

- **How to handle interrupts by OS**

```
Generally I/O devices are much slower than processor. Therefore, after an I/O
call, processor has to sit idle till the I/O device completes its event. Thus, OS
saves execution context of current process in PCB (Process Control Block) and
switches processor to execute some other process. When I/O event is completed, I/O
```

```
device interrupts OS. Operating System then retrieves the execution context of
suspended process and switches the processor to resume its execution from where it
was suspended.
```

Interrupts can occur either synchronously or asynchronously.

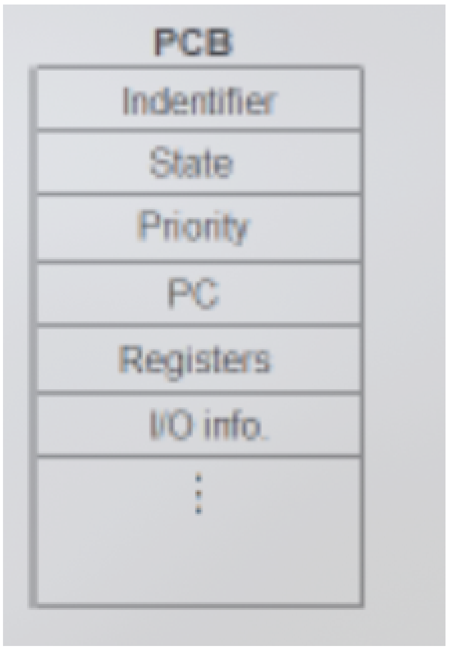
- Synchronous interrupts are triggered by the **system itself**, typically in response to some kind of error or exception. These are **predictable**
- Asynchronous interrupts, are triggered by **external events**, such as user input or a device requesting access to the system. These are **unpredictable**.

PCB

```
PCB is a data structure that is used to store the execution context of a process.
```

When a process is made a PCB is also made to store and restore the current state of the process. PCB also **stores the current information about the process**. When a process dies, the PCB also gets removed. PCB is a small space in the RAM that normal users don't have access to.

- **Information Stored in PCB**
  - Identifier
  - State
  - Priority
  - PC - address of the next instruction to be executed
  - Registers - CPU register values associated with the process
  - I/O info - I/O devices assigned to process



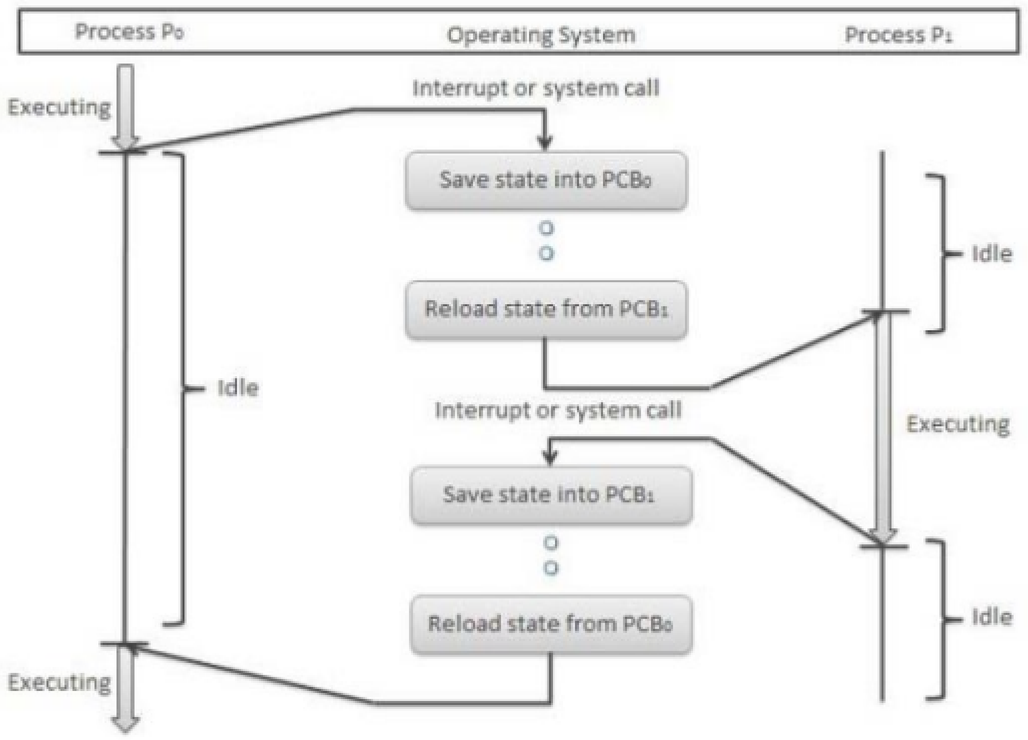
- What are 2 main techniques used to optimize processor utilization?

```
Multiprocgramming
Time sharing
```

> Both of these techniques use context switching

Context switching

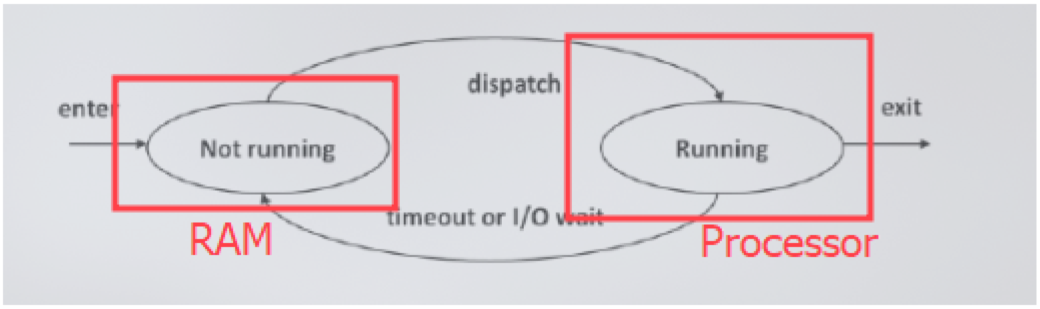
```
A context switch is the mechanism to store and restore the state or context of a
CPU in the Process Control Block so that a process execution can be resumed from
the same point at a later time.
```

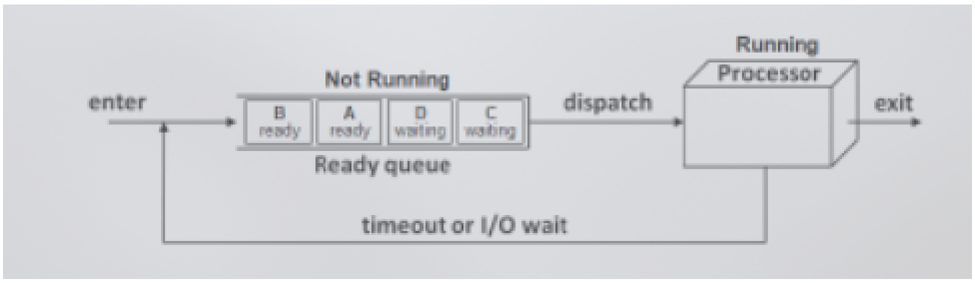


Execution of Processes

When processes are executed with interrupt handling and all that, OS uses a **Process Model** to manage the processes.

Two state process model





Only have 2 stages

1. `Running` state - Processes that are executing atm.
2. `Not Running` state - Processes that are in the queue to get executed.

> New processes enter into `Not Running` state. After the expiry of time quantum or process wants to do I/O, Operating System interrupts the process in `Running` state and transfers it to Not Running state. Operating System then dispatches another process in Not Running state to `Running` state for execution. Complete/aborted processes exit from `Running` state.

Since all the processes are in **one queue**, the processor can't dispatch other processes till the latest process is **done executing** or **its time of execution for that round is finished (timeout)**.

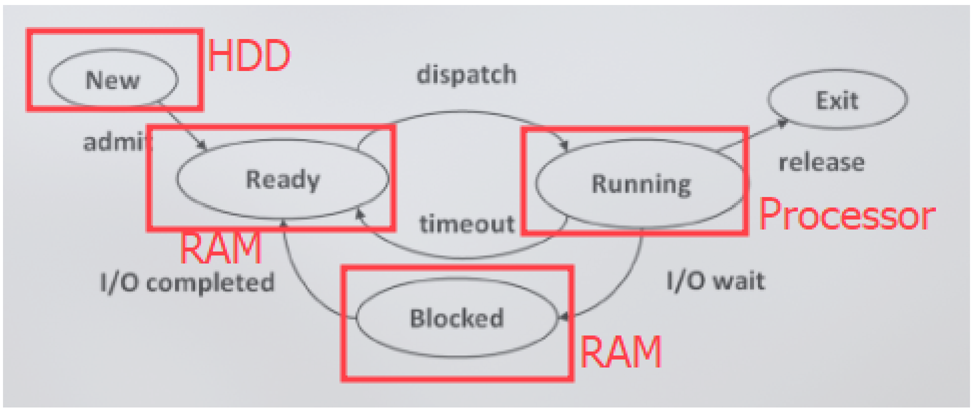
Problem

Some processes in `Not Running` state may be ready to run, but can be blocked by processes waiting for some I/O event completion.

Solution

We can divide the `Not Running` queue to 2 as `Ready` and `Blocked` so that the processor can dispatch them separately without waiting for one to finish.

Five State Process Model



There are 5 stages

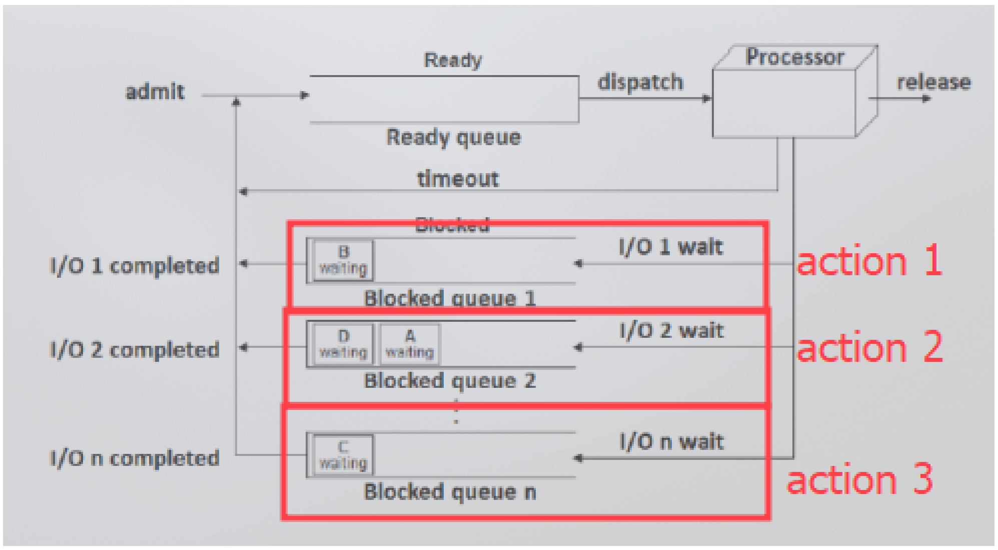
1. `New` - newly created process from the HDD
2. `Ready` - Processes that are in the queue to get dispatched
3. `Running` - Processes that are running atm
4. `Blocked` - Processes that are waiting for I/O
5. `Exit` - Processes that the execution is completed.

Problem

Processes in the `Blocked` state maybe **blocked even after their I/O is completed**

Solution

Blocked state should be split into groups according to the different I/O actions.



Now small awaited actions can be dispatched again to the `Ready` state without waiting for long awaited actions to be done.

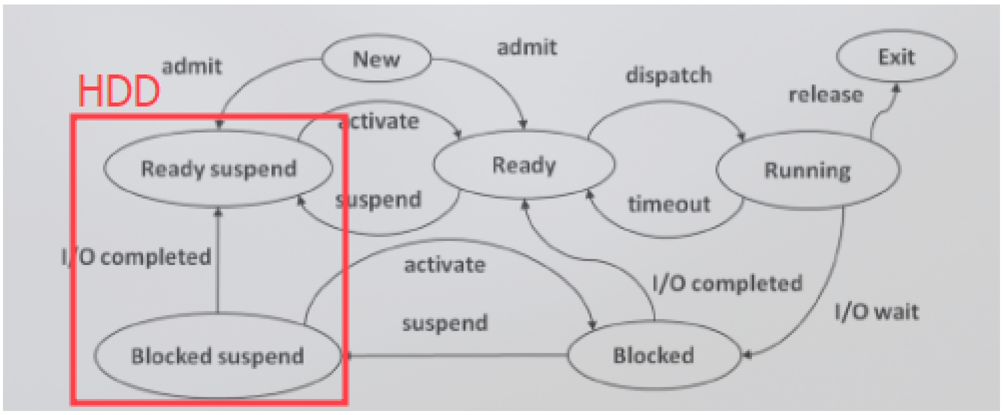
Problem

If all the processes are in blocked, RAM runs out of memory making the processor idle

Solution

Move some blocked processes to HDD (to virtual memory)

Seven State Process Model



`New` state is in the HDD as well. Meaning that **new processes are created in the HDD.** `activate` can also be named as `Swapped in`. `suspend` can also be named as `Swapped out`

There are 7 stages

1. `New` - newly created process from the HDD
2. `Ready` - Processes that are in the queue to get dispatched
3. `Running` - Processes that are running atm
4. `Blocked` - Processes that are waiting for I/O
5. `Blocked Suspended` - Processes that are blocked waiting for I/O which were moved to HDD to reserve RAM

6. `Ready Suspnded` - Processes that has completed I/O from Blocked Suspend, prioritization (Ready -> Ready suspend), newly admitted processes which aren't sent to RAM as there is no space (New -> Ready suspend)
7. `Exit` - Processes that the execution is completed.

Both these `Ready suspnded` and `Blocked suspended` are stored in HDD (virtual memory),

> Blocked -> Blocked suspend (Suspend / Swapped out)

```
- When memory in the RAM is not enough, so some of the blocked processes are moved to HDD
```

> Blocked suspend -> Blocked (Activate / Swapped in)

```
- Once RAM has enough space, process is again put into Blocked so that it can complete I/O (I/O can be completed while its in Blocked suspend as well)
```

> Blocked suspend -> Ready suspend

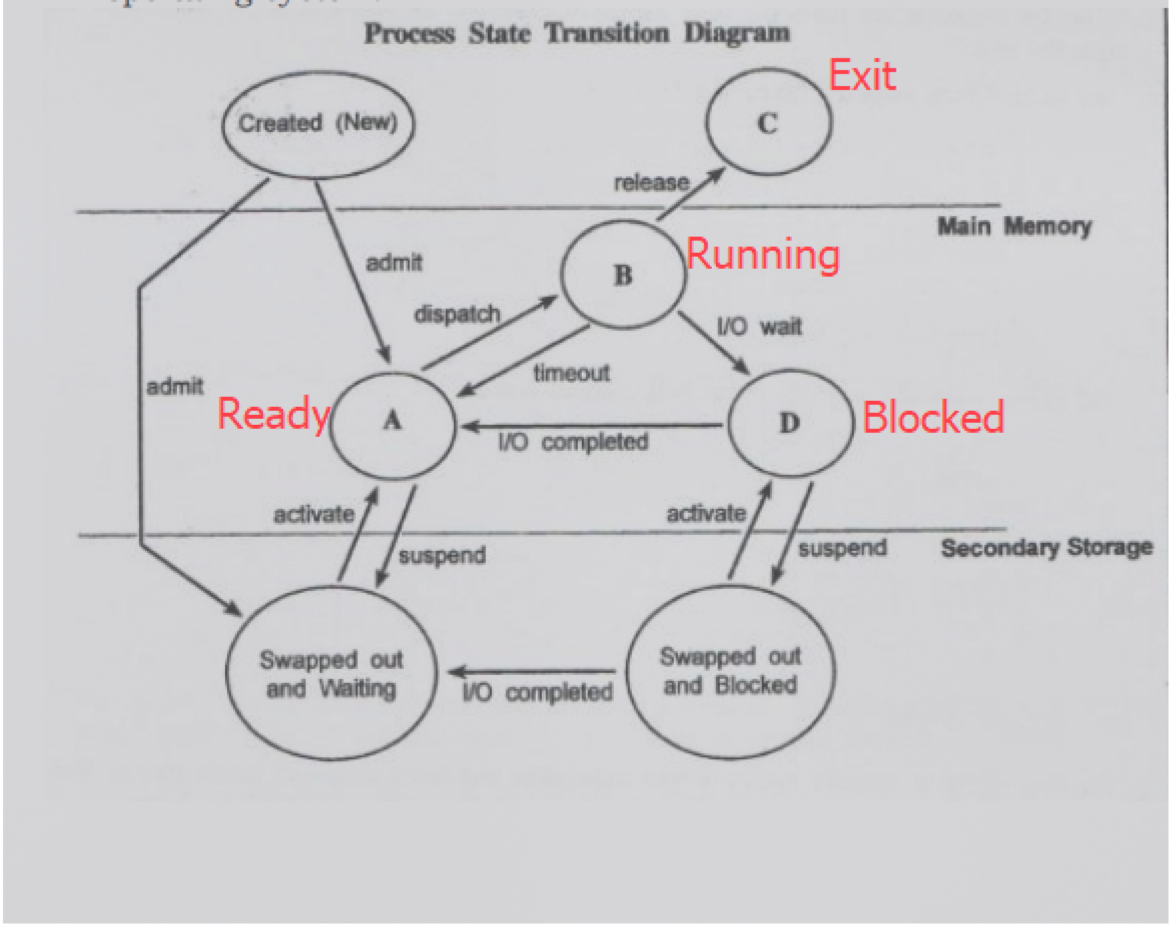
```
- If I/O event gets completed while it's in Blocked suspend, it's put into Ready suspend as it's ready to run
```

> Ready -> Ready suspend

```
 - When prioritization happens, processes that are ready to run are put into Ready suspend, giving place to high priority processes
```

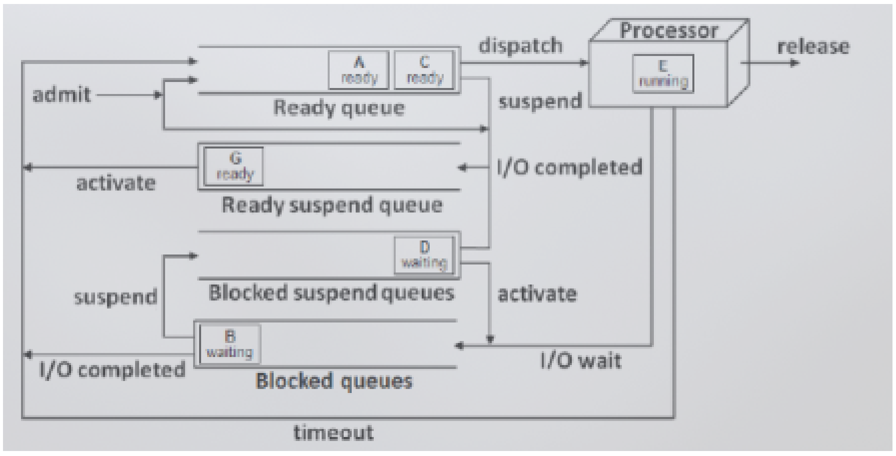
> New -> Ready suspend

```
- While creating the process, if a prioritized process gets created first, other low prioritized once are put in to Ready suspend.
- Newly created processes which aren't sent to RAM as there is no space are also sent here
```



When no process is ready to run and memory is full, one or more processes are **swapped out** and one or more new or suspended processes are **swapped in**.

```
Ready Suspend or Bloacked Suspend (HDD) --> Ready or Blocked(RAM) [swapped in]
(Getting processes to the RAM)
Ready or Blocked (RAM) --> Blocked suspend or Ready Suspend(HDD) [swapped out]
(Removes the process from RAM)
```



> Note that the Blocked and Blocked suspend states are implemented with multiple queues to handle processes waiting for different I/O events.

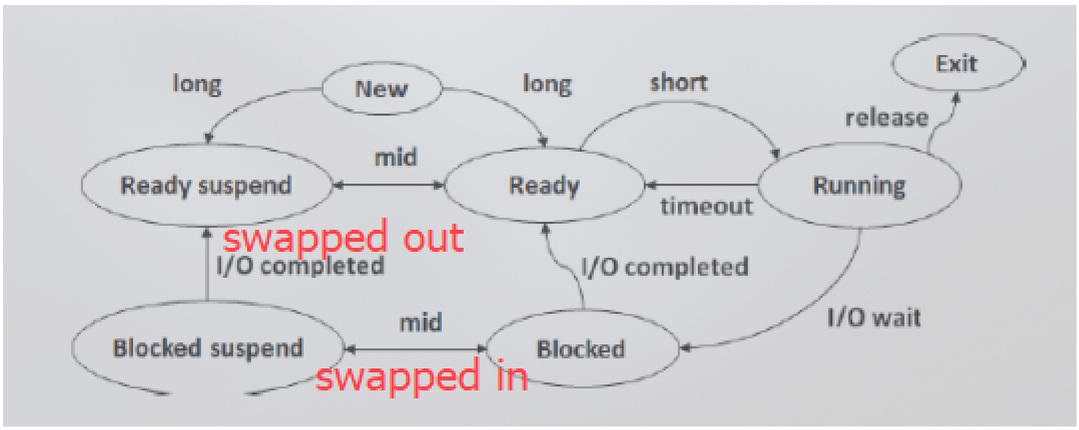
Types of Processes

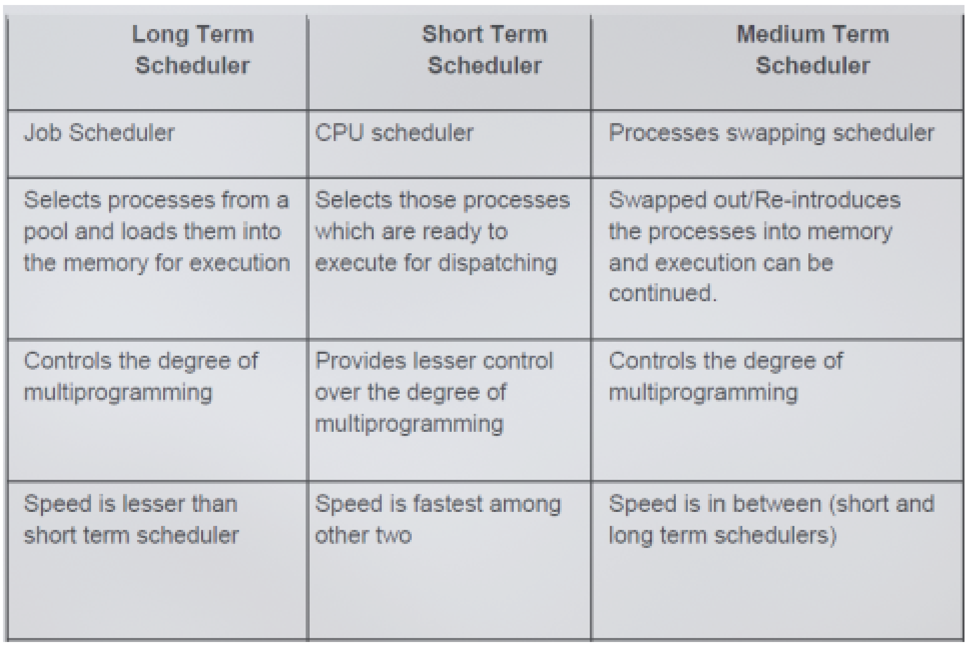
1. `Processor bound processes` - Spend more time for using processor than doing I/O
2. `I/O bound processes` – Spend more time for doing I/O than using processor

3. `Premptive processes` – can be interrupted, suspended the execution and resumed later
4. `Non-preemptive processes` – Cannot be interrupted and continue until process completes its execution or blocked by itself (Real Time processes)

Process Scheduling

1. `Long term scheduling` – schedules admission of processes, decides degree of multiprogramming
2. `Mid-term scheduling` - schedules swapping function
3. `Short term scheduling (aka dispatchers aka processor scheduling)` – schedules switching of processes between ready and running state (used in context switching, therefore very frequently happens)





Long Term Scheduler	Short Term Scheduler	Medium Term Scheduler
Job Scheduler	CPU scheduler	Processes swapping scheduler
Selects processes from a pool and loads them into the memory for execution	Selects those processes which are ready to execute for dispatching	Swapped out/Re-introduces the processes into memory and execution can be continued.
Controls the degree of multiprogramming	Provides lesser control over the degree of multiprogramming	Controls the degree of multiprogramming
Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between (short and long term schedulers)

```
Ready -> Running - Short term
From new -> any - Long term
Ready -> Ready suspend (vise versa) - Mid term
Blocked -> Blocked suspend (vise versa) - Mid term (or any swapping action)
```

Assignments to optimize the processor.

1. `Response Time` - time between submission of a request and receipt of response
2. `Thoughtput` - number of processes completed per unit time

3. Turnaround time – average time from submission of a process to its completion
4. Waiting time -time a process spends in ready state.

04 - Explores how an operating system manages the memory resources

Explores how an operating system manages the memory resources

- What is memory management?

```
Partitioning of memory is called memory management.
```

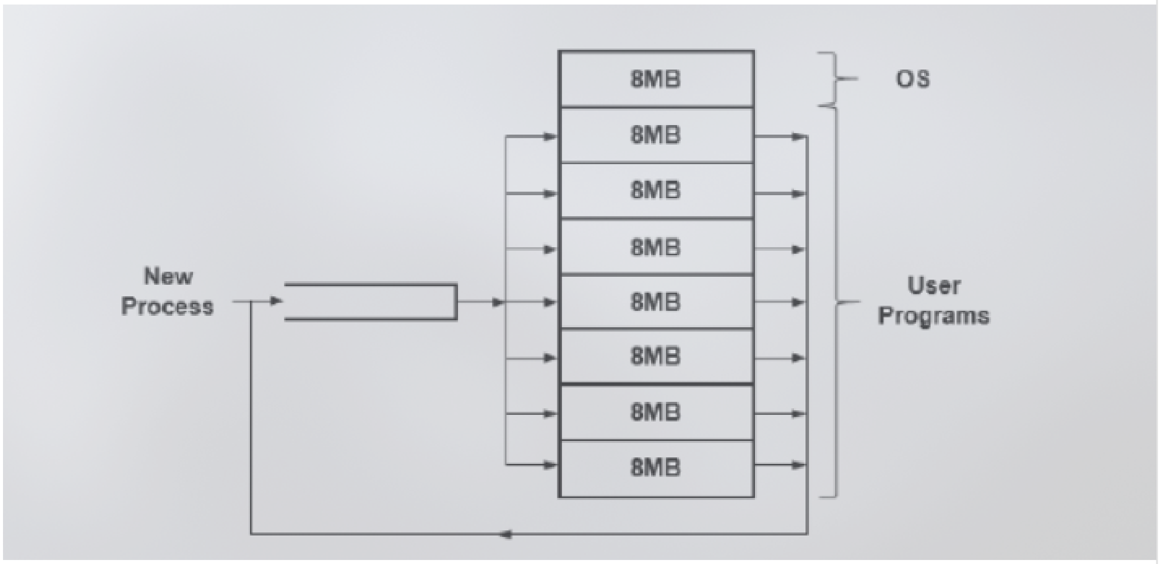
Types of partitioning

There are 2 types of partitions

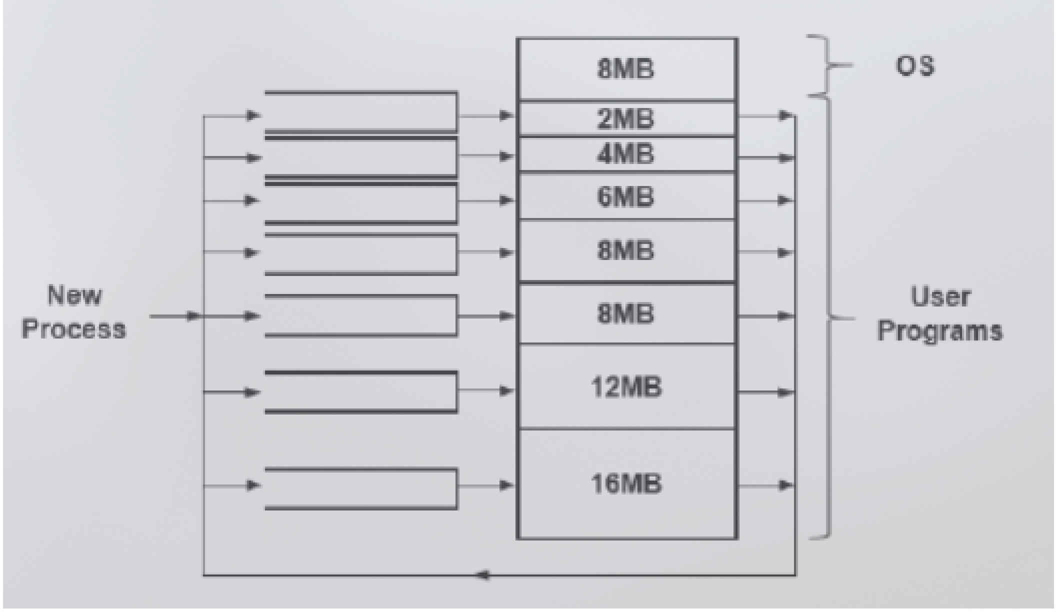
1. Fixed partitioning
2. Dynamic partitioning

Fixed Partitioning

- Partitions memory into fixed static portions during the design of the system
- Fixed number of processes can use memory at a time
- Two kinds of fixed partitioning
  - Equal size fixed partitioning



    - Strengths - Simple to implement, Minimum overhead (fewer activities for the OS)
    - Weaknesses - If process is too small it may use all the space (**internal fragmentation**), Large processes are required to redesign,
  - Unequal size fixed partitioning

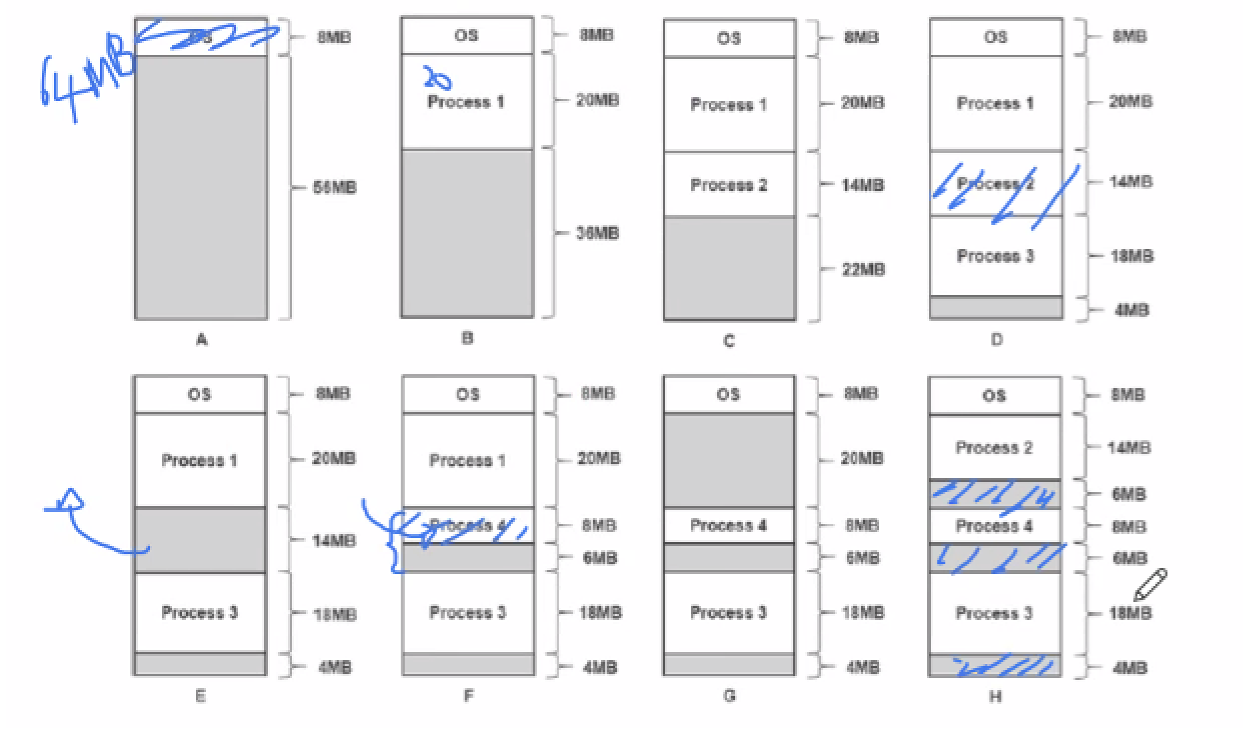


- Incoming process is directed to the smallest available partition that is big enough to load it
- Every partition has a separate queue to hold the swapped out processes suitable for that partition
- Strength - Reduces Internal fragmentation.
- Weaknesses - If process is smaller than the smallest available partition, it may not utilize the partition space efficiently, If process is larger than the largest partition in the system, **program is required to redesign**., There can be a process waiting in the queue to be processed

Dynamic Partitioning

- Partitions memory dynamically based on the size of incoming processes.
- No internal fragmentation happens because the exact amount of memory needed for the process is allocated
- Strengths
  - No internal fragmentation
  - No need to redesign programs
  - Variable number of processes can use memory at a time
- Weaknesses
  - Dynamic memory allocation overhead (OS uses the processor to partition the memory itself before loading the programs) (Overhead simply means using processor time by OS to do tasks that aren't exclusively related to OS)
  - External Fragmentation occurs due to swapping actions by the time passes.
- Solution
  - To solve this, all the processes in memory are moved to one end, leaves a large free space at the other end. (**periodical compaction**)
  - Processing overheads

External Fragmentation



> Say the OS swap in `Process 1` which is 20 MB, `Process 2` which is 14 MB and `Process 3` which is 18 MB at first. Then `Process 3` is swapped out after execution is done. Then another process of 4MB is swapped in while `Process 1` swaps out and another process of 14 MB fill that space. Now the memory has 6MB, 6MB and 4MB space left respectively. Now if there is a 12 MB process that needs to be loaded to the memory, it can't be done as there is no space block with 12 MB memory even though it has enough memory separately (6 + 6 + 4 = 16MB) This is called **External Fragmentation**

(When unusable space is left inside a group-sized container, its internal fragmentation, while if it happens outside its external fragmentation.)

Virtual Memory

When the memory in RAM isn't enough to store the currently executing processes, a part of the HDD is used to store those processes. That is called Virtual memory.
Even though the processes are stored in the HDD, **they are never executed** in there. Those programs are again loaded to RAM portion by portion and then gets executed.

There are 2 ways that the RAM loads such portions from Virtual Memory

1. Segmentation
2. Paging

Segmentation

- Similar to dynamic partition
- This method divides processes to segments based on thier subroutines (functions defined inside)
- Those segments are loaded to RAM in any place available.
- Features

    - No internal Fragmentation
    - External Fragmentation is possible but very less
    - System overheads

Paging

- Most popular virtual memory technology.
- Divides processes into small, equal size **pages** (default in most often - 4KB).
- Partitions memory into page size blocks (**page frames**).
- Transfers pages from disk to page frames in memory.
- When a frame is no longer needed, it's moved back to disk.
- Features
    - No external fragmentation

Address translation

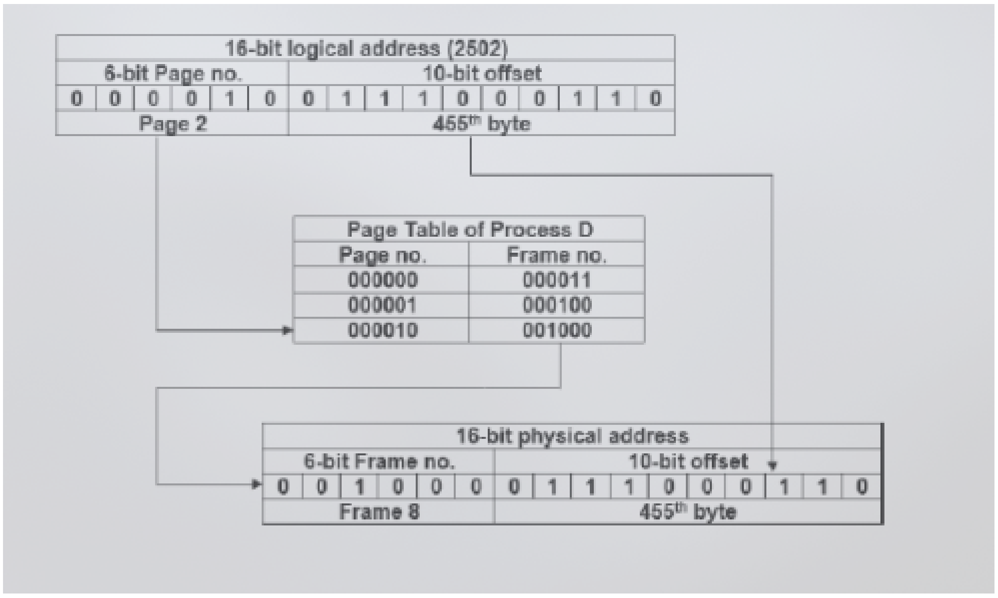
When paging happens, the process portions stored in the Virtual Memory (HDD) are moved to RAM. These portions stored in the HDD has a certain memory address. This address is called the `logical address` These logical addresses are moved to the page frames in the RAM. These page frames have a certain address as well. Those addresses are called `physical addresses`

```
pages (in HDD) -> logical addresses / virtual addresses
frames (in RAM) -> physical addresses
```

These addresses are mapped together so that the processor know which portion of the process was moved to which page frame in the RAM. This process of **translating logical addresses to physical addresses is known as Address Translation**

There are 2 things that helps with this process.

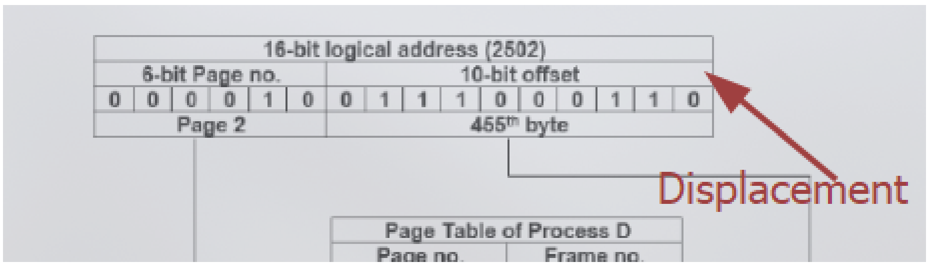
1. Hardware - Memory Management Unit (MMU) in the processor
2. Software - Page table that includes the mappings of the addresses

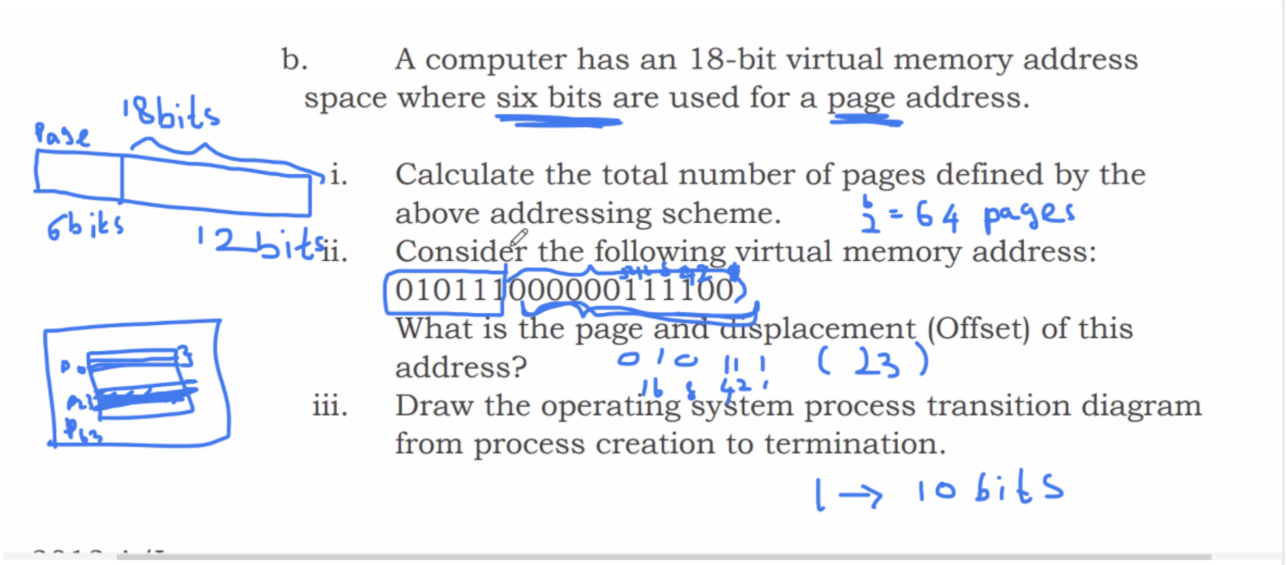


Page Table of Process D	
Page no.	Frame no.
000000	000011
000001	000100
000010	001000

> To process the 455th byte, the processor needs to look up the Page Table and go to `Page 2`. As the Page Table says, the 455th byte is stored in `Frame 8`

The offset is also called as `Displacement`





b. A computer has an 18-bit virtual memory address space where six bits are used for a page address.

i. Calculate the total number of pages defined by the above addressing scheme. $2^6 = 64$ pages

ii. Consider the following virtual memory address: 010111000000111100

What is the page and displacement (Offset) of this address? 010 111 (23)

iii. Draw the operating system process transition diagram from process creation to termination.

1 → 10 bits

- Find the no of pages by dividing the **memory of the program** from the page size
- Find the no of frames by dividing the **memory of the RAM** from the page size

Example case

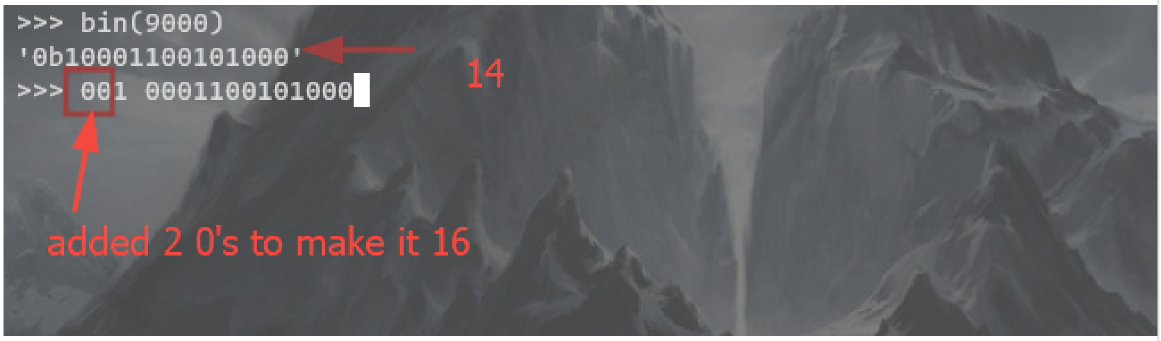
Say you have a program of 64 KB memory and your RAM is 128 KB and the system page size is 8 KB.

1. Now if you want to find the **no. of memory pages** that this program can have, you need to divide the memory of the program from the system page size `64 / 8 = 8`
2. Same as above, if you want to find the **no. of frames** that the RAM can have, need to divide the memory of the RAM from the system page size `128 / 8 = 16`
3. According to this, the first page has 8192 bytes, meaning that the 0th page is from bytes 0 to 8191 and the 1st page is from 8192 to 163833 and so on.
4. To find the **no. of bits used to represent the page number in the logical address**, you need to write the no of pages as power of 2 `8 = 2**3`. This means that you need 3 bits to represent the page no.
5. To find the **no. of bits used to represent the frame number in the logical address**, you need to write the no of frames as power of 2 `16 = 2**4`. This means that you need 4 bits to represent the frame no.

6. To find the **no. of bits needed to represent the offset (Displacement)**, you need to represent the page size in bytes (as a power of 2) `8 KB = 8 * 1024` -> `8 = 2**3 * 2**10 = 2**13`. This means 13 bits are used to represent the offset
7. The total bits used to represent the virtual address is `no. of bits in page no + no. of bits in offset` -> `3 + 13` which is `16 bits`.
8. Similarly. the total bits used to represent the physical address is `no. of bits in frame no + no. of bits in offset` -> `4 + 13` which is `17 bits`

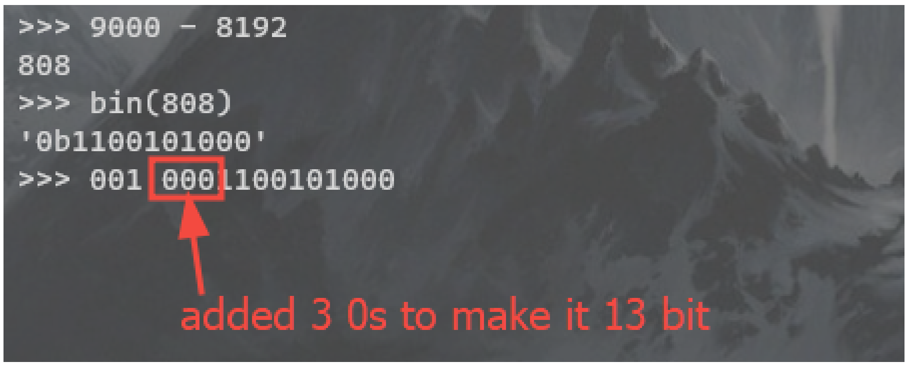
Now we can make the logical and physical addresses based on these.

1. Let's say you need to find the logical address of 9000, For that you have 2 ways, you either,
   - convert 9000 to binary, add 0's to front so that it becomes a 16-bit address ,get last 13 bits as the offset and the first 3 bits as the page number.



```
>>> bin(9000)
'0b10001100101000'
>>> 001 0001100101000
```

   - or subtract the no of bytes in a system page size (which in this case is 8192 (`8 * 1024`)) from 9000, get the binary of the difference as offset after filling with zeros so it becomes 16 and get 1 as 9000 lies under page 1



```
>>> 9000 - 8192
808
>>> bin(808)
'0b1100101000'
>>> 001 0001100101000
```

Input and output Device Management

Device driver

Device driver is a software. When a new device is introduced to the OS, the certain device driver must be installed in order for the OS to be able to communicate with it. Device drivers depends on both the hardware and the operating system loaded in to the computer

Spooling

Spooling is an acronym for `simultaneous peripheral operations on line`. Spooling refers to putting data of various I/O jobs in a buffer. This buffer is a special area in memory or hard disk which is accessible to I/O devices. There are certain things that the OS does relate to spooling.

1. Handles I/O device data spooling as devices have different data access rates.
2. Maintains the spooling buffer which provides a waiting station where data can rest while the slower device catches up.