Embedded Operating System

❖Module Contents

- **OS Concepts (in depth)**
  - Introduction
  - Process management
  - CPU scheduling
  - Memory management
  - File & IO management
- **Linux Programming**
  - Commands
  - Shell scripts
  - System call programming
- **UNIX/LINUX Internals**
- **Embedded Linux**

❖Pre-requisites

- C Programming
- Computer Fundamentals
- CPU Architecture -- ARM Cortex-A
- Linux basic commands
- OS basic concepts*

❖Books

- OS Concepts Galvin (Chapter 1 & 2)
- OS Design -- Crowley
- Design of UNIX OS -- Bach (File IO, IPC)
- Professional Linux Kernel Architecture (Linux internals -- VFS, Task management, ...)
- Linux kernel development -- Love (Linux internals -- Memory management)
- Beginning Linux Programming -- Neil (Commands and Shell Scripts, SysCall Basics)
- Linux Programming Interface -- Kerrisk (Linux SysCall Programming)

Evaluations

- **Theory**
  - CCEE (MCQ) - 40 marks
  - Lab - 40 marks
    - Linux programming

- **Internals (20 marks)**
  - 10 marks - Technical interviews
  - 10 marks - Assignment

Todays Agenda

- Learning OS
- What is need of OS?
- What is OS?
- OS Functions
- Processes Management
- Memory Management

Steps for Learning OS

- **Step 1 :End user**
  - Linux commands
- **Step 2 : Administrator**
  - Install OS (Linux)
  - Configuration - Users, Networking, Storage, ...
  - Shell scripts
- **Step 3 : Programmer**
  - Linux System call programming
- **Step 4 : Designer/Internals**
  - UNIX & Linux internals

Q. Why there is a need of an OS?

- Computer is a machine/hardware does different tasks efficiently & accurately.
- Basic functions of computer :
    1. Data Storage : Memory Devices
    2. Data Processing : CPU/Processor
    3. Data Movement : I/O Devices
    4. Control

- As any user cannot communicates/interacts directly with computer hardware to do different tasks, and hence there is need of some interface between user and hardware.

Q. What is an Operating System?

- An OS is a **system softwar**e (i.e. collection of system programs) which acts as an **interface between End user and hardware.**
- An OS also acts as an **interface between computer programs and computer hardware**.
- An OS controls an execution of all programs and it also controls hardware devices which are connected to the computer system and hence it is also called as **a control program.**
- An OS manages limited available resources among all running programs, hence it is also called as a **resource manager.**
- An OS allocates resources like main memory, CPU time, i/o devices access etc... to all running programs, hence it is also called as **a resource allocator.**

Operating Systems Concepts

❖**From End User:** An OS is a software (i.e. collection of programs) comes either in CD/DVD, has following main components:

- **Bootable CD/DVD = Core OS + Applications + Utilities**
- **Core OS = Kernel (Performs all basic functions of OS.**

1. **Kernel**: It is a core program/part of an OS which runs continuously into the main memory does basic minimal functionalities of it. e.g. Linux: vmlinuz, Windows: ntoskrnl.exe
2. **Utility Software's:** e.g. disk manager, windows firewall, anti-virus software etc...
3. **Application Software's**: e.g. Google chrome, shell, notepad, MS office etc...

➢Functions of an OS:

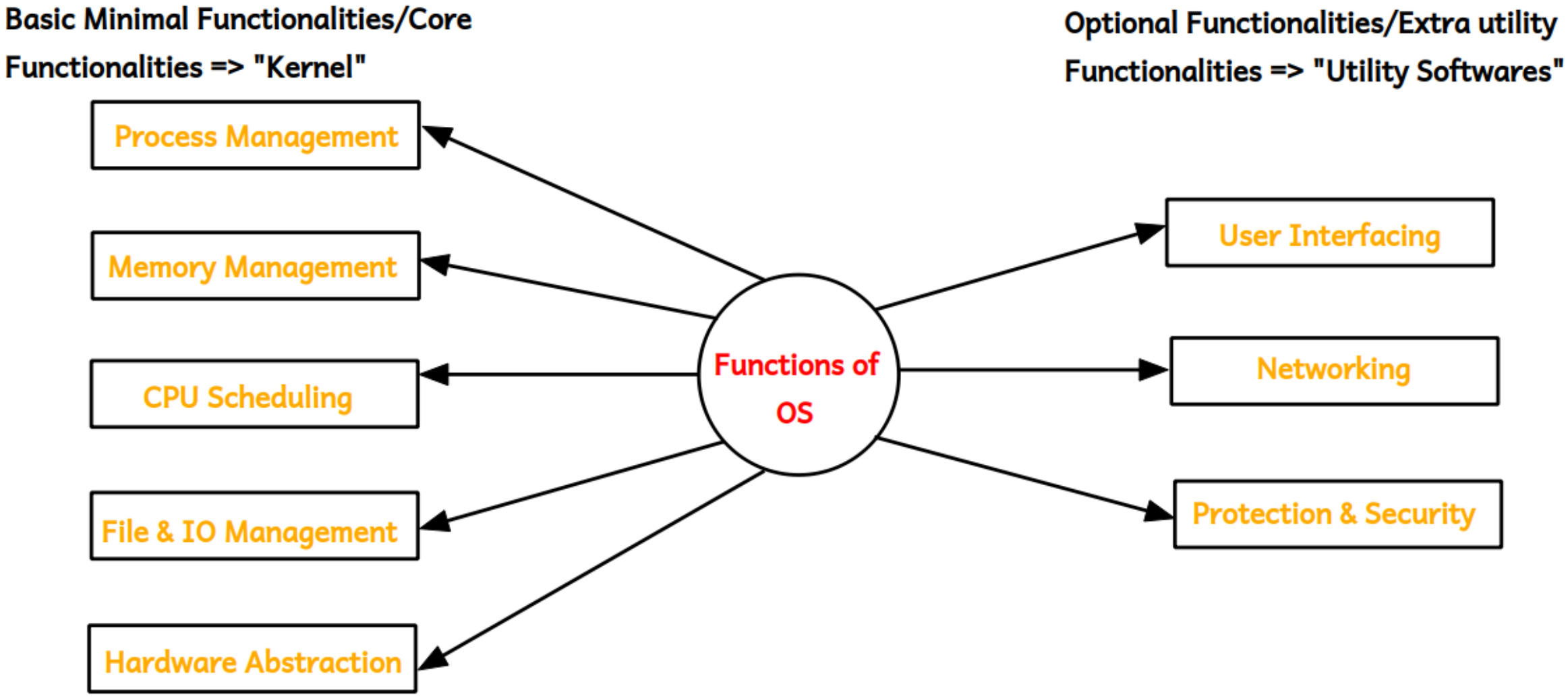
Basic minimal functionalities/Kernel functionalities:

1. CPU Scheduling
2. Process Management
3. Memory Management
4. Hardware Abstraction
5. File & IO Management

Extra utility functionalities/optional:

6. Protection & Security
7. User Interfacing
8. Networking

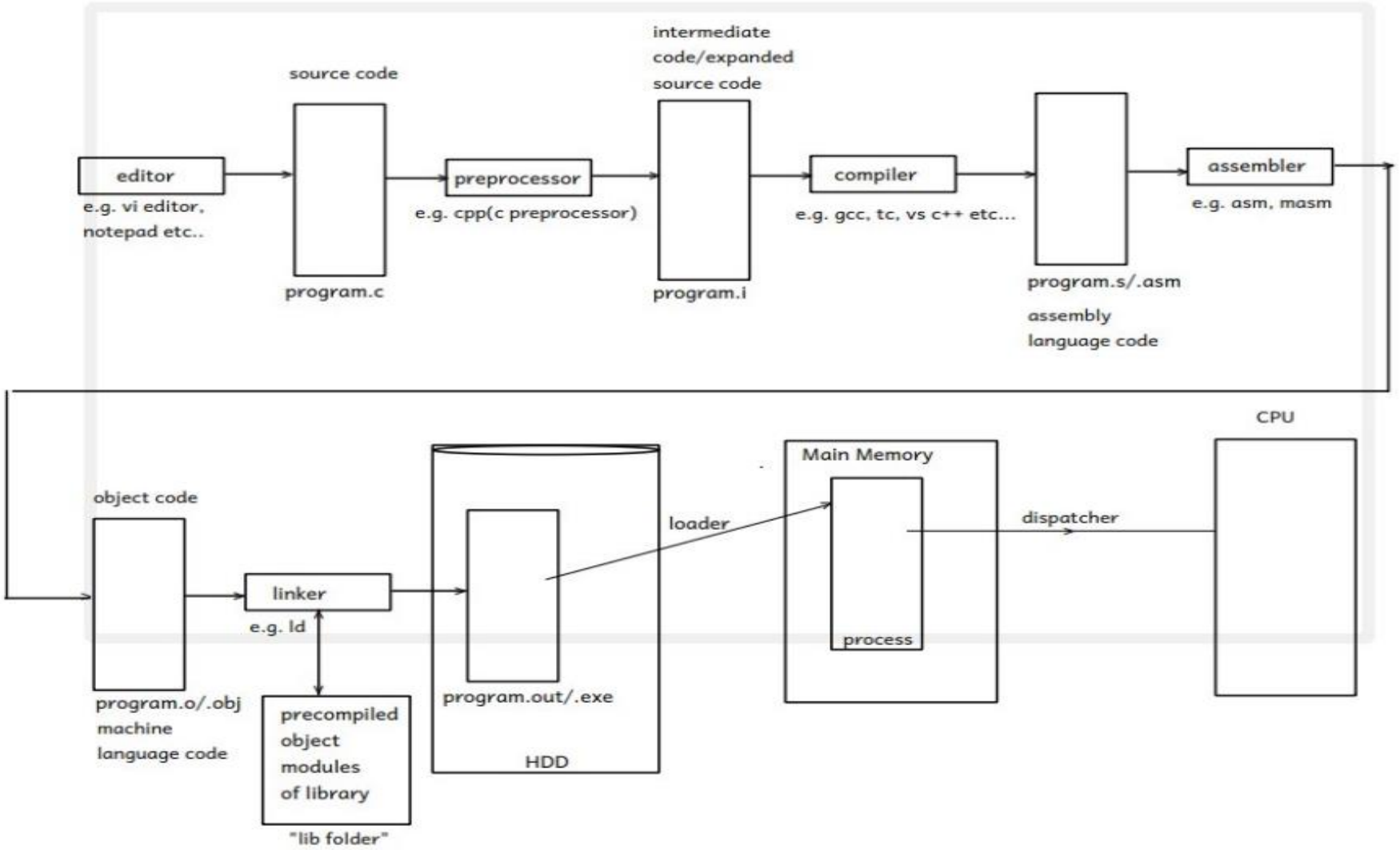
Operating Systems Concepts



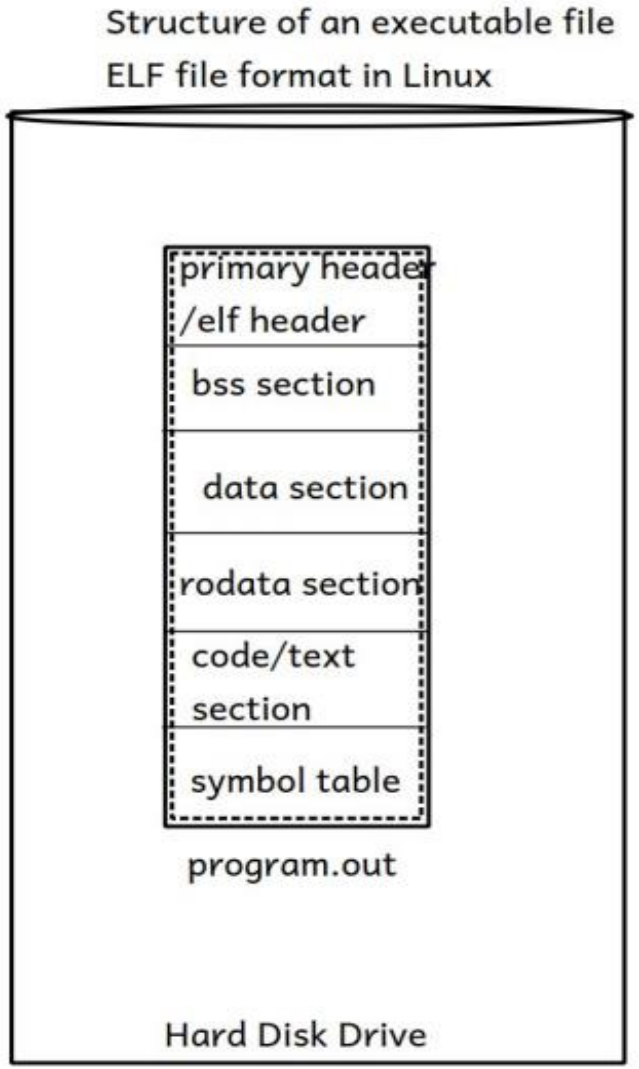
Process Management

- When we say an **OS does process management** it means an OS is responsible for process creation, to provide environment for an execution of a process, resource allocation, scheduling, resources management, inter process communication, process coordination, and terminate the process.
- **Includes:** Process & Thread concept, Process/Thread creation, Process life cycle, Inter-process communication, Race condition, Synchronization, Deadlock.
- **What is a Program?**
  - **User view:** - Program is a finite set of instructions written in any programming language given to the machine to do specific task.(Source Code)
  - **System view:** - Program is an executable file in HDD which divided logically into sections like exe header, bss section, data section, RO data section, code section, symbol table.(Executable File).

Operating Systems Concepts



Embedded Operating Systems



1. **primary header/exe header:** it contains information which is required to starts an execution of the program.
   e.g. - addr of an entry point function --> main() function
   - **magic number:** it is constant number generated by the compiler which is file format specific.
     - magic number in Linux starts with ELF in its eq **hexadecimal format**.
     - info about remaining sections.
2. **bss(block started by symbol) section:** it contains uninitialized global & static vars
3. **data section:** it contains initialized global & static vars
4. **rodata (readonly data) section:** it contains string literals and constants.
5. **code/text section:** it contains an executable instructions
6. **symbol table:** it contains info about functions and its vars in a tabular format.

Program

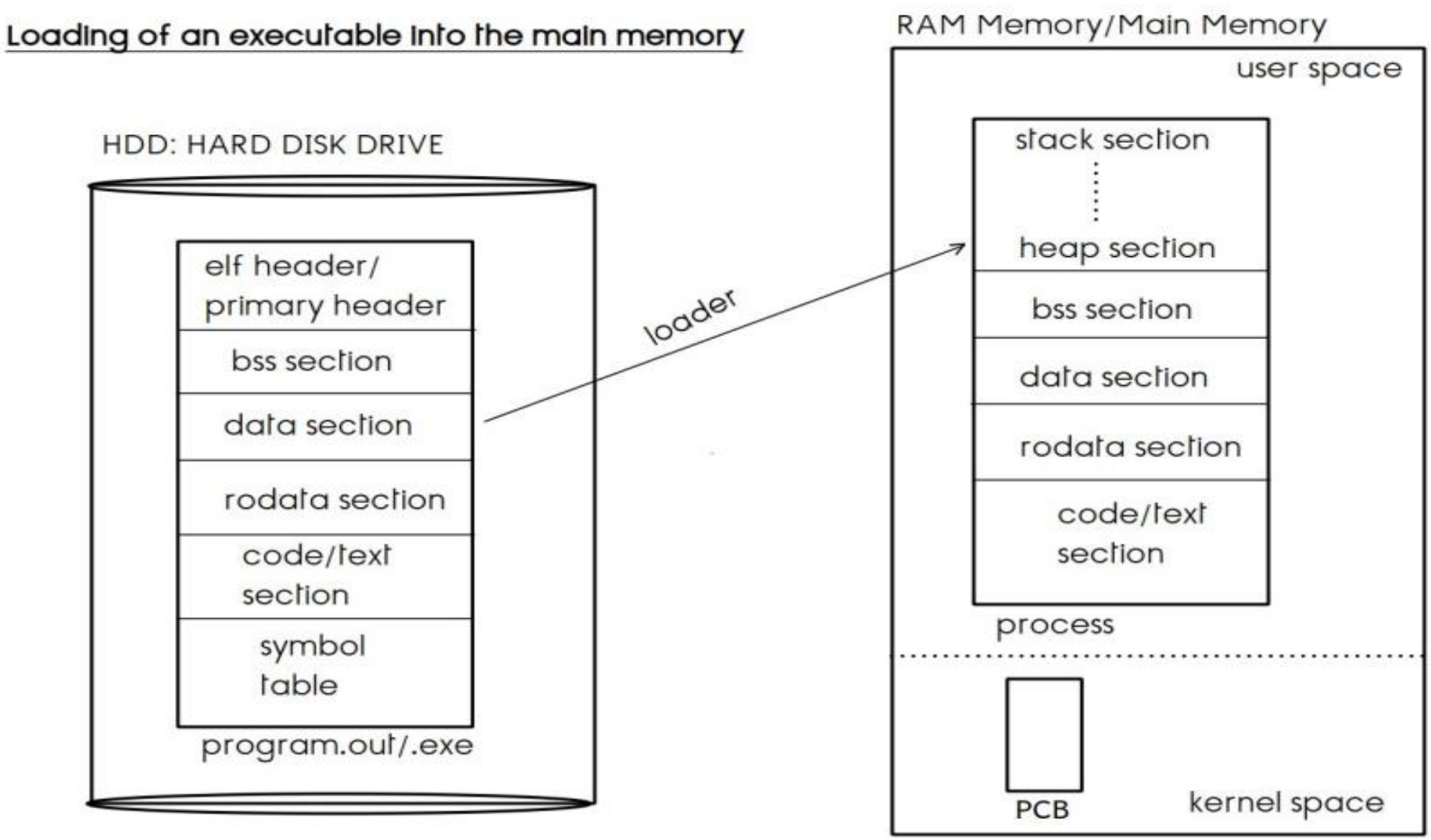
- Program Set of instructions given to the computer --> Executable file.
- Program --> Sectioned binary --> "objdump" & "readelf".
  - Exe header --> Magic number, Address of entry-point function, Information about all sections. (objdump -h program.out)
  - Text --> Machine level code (objdump -S program.out)
  - Data --> Global and Static variables (Initialized)
  - BSS --> Global and Static variables (Uninitialized)
  - RoData --> String constants
  - Symbol Table --> Information about the symbols (Name, Size, section, Flags, Address) (objdump -t program.out)
- Program (Executable File) Format
  - Windows -- PE
  - Linux -- ELF
- Program are stored on disk (storage).

What is a Process?

- **User view:**
  - Program in execution is called as a process.
  - Running program is called as a process.
  - When a program gets loaded into the main memory it is referred as a process.
  - Running instance of a program is referred as a process.

- **System view:**
  - Process is a program loaded into the main memory which has got PCB into the main memory inside kernel space and program itself into the main memory inside user
  - space has got bss section, rodata section, code section, and two new sections gets added for the process .
    - stack section: contains function activation records of called functions.
    - heap section: dynamically allocated memory

Embedded Operating Systems



Embedded Operating Systems

- Process control block(PCB) contains information about the process (required for the execution of process).
  1. Process id
  2. Exit status
  3. Scheduling information (State, Priority, Sched algorithm, Time, ...)
  4. Memory information (Base & Limit, Segment table, or Page table)
  5. File information (Open files, Current directory, ...)
  6. IPC information (Signals, ...)
  7. Execution context
  8. Kernel stack

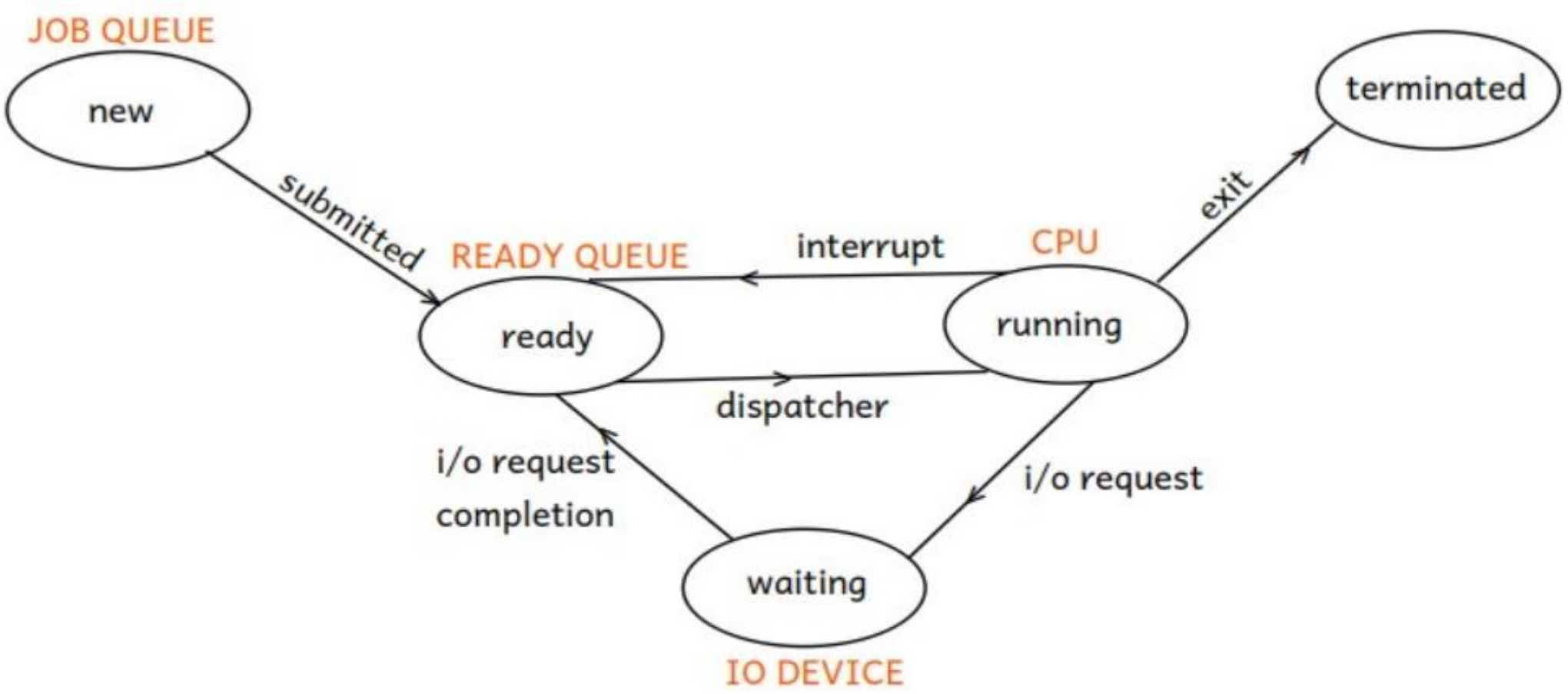
- PCB is also called as process descriptor (PD), uarea (UNIX), or task_struct (Linux).
- In Linux size of task_struct is approx 5KB.

➢Process States:

- Throughout execution, process goes through different states out of which at a time it can be only in a one state.

- States of the process:

1. **New state:** upon submission or when **a PCB** for a process gets created into the main memory process is in **a new state.**
2. **Ready state:** after submission, if process is in the main memory and waiting for the CPU time, it is in **a ready state**.
3. **Running state:** if currently the CPU is executing any process then state of that process is considered **as a running state**.
4. **Waiting state:** if a process is requesting for any i/o device then state of that process is considered as **a waiting state**.
5. **Terminated state:** upon exit, process goes into terminated state and its PCB gets destroyed from the main memory.



PROCESS STATE DIAGRAM

➢Process States:

- To keep track on all running programs, an OS maintains few data structures referred **as kernel data structures:**

1. **Job queue:** it contains list of PCB's of all submitted processes.
2. **Ready queue:** it contains list of PCB's of processes which are in the main memory and waiting for the CPU time.
3. **3. Waiting queue:** it contains list of PCB's of processes which are requesting for that particular device.

1. **Job Scheduler/Long Term Scheduler:** it is a system program which selects/schedules jobs/processes from job queue to **load them onto the ready queue**.
2. **CPU Scheduler/Short Term Scheduler:** it is a system program which selects/schedules job/process **from ready queue to load it onto the CPU.**
3. **Dispatcher:** it is a system program which loads a process onto the CPU which is scheduled by the CPU scheduler, and **the time required for the dispatcher to stops an execution of one process and to starts an execution of another process is referred as dispatcher latency.**

➢Context Switch:

- As during context-switch, the CPU gets switched from an execution context of one process onto an execution context of another process, and hence it is referred as **"context-switch".**
- **context-switch = state-save + state-restore**
- **state-save** of suspended process can be done i.e. an execution context of suspended process gets saved into its PCB.
- **state-restore** of a process which is scheduled by the CPU scheduler can be done by the dispatcher, dispatcher copies an execution context of process scheduled by the cpu scheduler from its PCB and restore it onto the CPU registers.

➢Memory Management

❖. Why there is a need of memory (main memory)management ?

- As main memory is must for an execution of any program and it is a limited memory, hence an OS manages main memory.
- To achieve maximum CPU utilization, an OS must support **multitasking, and to support multi-tasking multiple processes** must be submitted into the system at a time i.e. it must support multiprogramming, but **as main memory is limited** to support multiprogramming an **OS has to do memory management to complete an execution of all submitted processes.**
- **Memory space of one process should gets protected from another process**.

➢Memory Allocation:

- When a process is requesting for the main memory, there are two methods by which memory gets allocated for any process

1. Contiguous Memory Allocation

2. Non - contiguous Memory Allocation

1. Contiguous Memory Allocation:

- Under this method, process can complete its execution only if memory gets allocated for it in a contiguous manner.
- There are two methods by which memory gets allocated for process under contiguous memory allocation method.

1. Fixed Size Partitioning

2. Variable Size Partitioning

1. Fixed Size Partitioning:

- In this method, physical memory i.e. main memory (user space) is divided into fixed number of partitions and size of each partition is remains fixed.
- If any process is requesting for the memory it can be loaded into main memory in any free partition in which it can be fit.

❖**Advantages:**

- This method is simple to implement.

❖Disadvantages:

- **Internal fragmentation**: if memory remains unused which is internal to the partition.
- **Degree of multi-programming** is limited to the number of partitions in the main memory.
- **Maximum size of a process is limited** to max size partition in the main memory.
- To overcome limitations/disadvantages of fixed size partitioning method, variable/dynamic size partitioning method has been designed.

2. Variable/Dynamic Size Partitioning:

- In this method, initially whole user space i.e. physical memory is considered as a single free partition, and processes gets loaded into the main memory as they request for it.
- Size of partition and number of partitions are not fixed in advanced, it gets decided dynamically.

❖Advantages:

- There are very less chances of **internal fragmentation** - Degree of multi-programming is not limited/fixed
- Size of the process is not also limited, any size process may get load into the main memory.

❖ Disadvantages:

- **External fragmentation:** due to loading and removing of processes into and from the main memory, main memory is fragmented i.e. gets divided into used partitions and free partitions.

2. Variable/Dynamic Size Partitioning:

- In this method, initially whole user space i.e. physical memory is considered as a single free partition, and processes gets loaded into the main memory as they request for it.
- Size of partition and number of partitions are not fixed in advanced, it gets decided dynamically.

❖Advantages:

- There are very less chances of **internal fragmentation** - Degree of multi-programming is not limited/fixed
- Size of the process is not also limited, any size process may get load into the main memory.

❖ Disadvantages:

- **External fragmentation:** due to loading and removing of processes into and from the main memory, main memory is fragmented i.e. gets divided into used partitions and free partitions.

- In such case, if any new process is requesting for the memory and even if the requested size of memory is available, but due to unavailability of the memory in a contiguous manner request of that process cannot be accepted, this problem is referred as an **external fragmentation.**
- External fragmentation is the biggest problem under contiguous memory allocation, and hence there are two solutions on this problem:
- **1. Compaction**
  - shuffling of main memory can be done in such a way that all used partitions can be shifted to one side and all free partitions can be shifted to other side and contiguous large free partition will be made available for the new processes.
  - Compaction is practically not feasible as there is need to do recalculations of addresses every time.
- **2. Non- contiguous Memory Allocation**

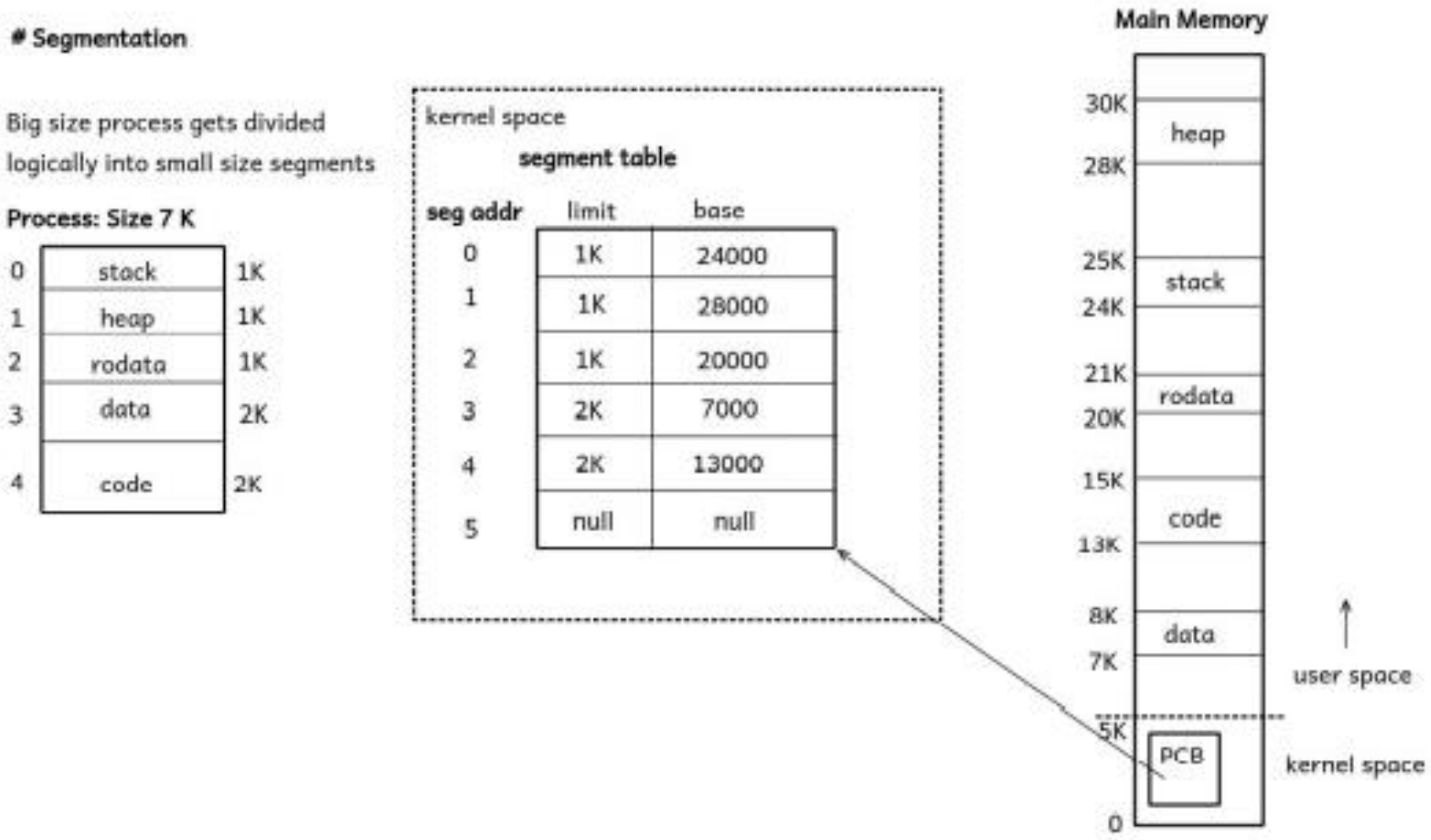
2. Non- contiguous Memory Allocation:

- Under this method, process can complete its execution even if memory gets allocated for it in a non - contiguous manner, and it can be achieved by two memory management techniques:
  1. **Segmentation**
  2. **Paging**
- So by using segmentation & paging techniques, process can complete its execution even after memory gets allocated for it in a **non- contiguous manner.**

- **Addresses generated by compiler (i.e. compiler + linker) are referred as logical addresses.**
- Addresses which can be see by the process when it is in the main memory referred as **physical addresses**.
- **MMU (Memory Management Unit):** which is a hardware unit converts logical address into physical address.
- **MMU is a hardware** contains adder circuit, comparator circuit, base register and limit register. Values of base register and limit registers gets change during context-switch, and memory space **of one process gets protected from another process**.
- CPU always executes program in its **logical memory space**.

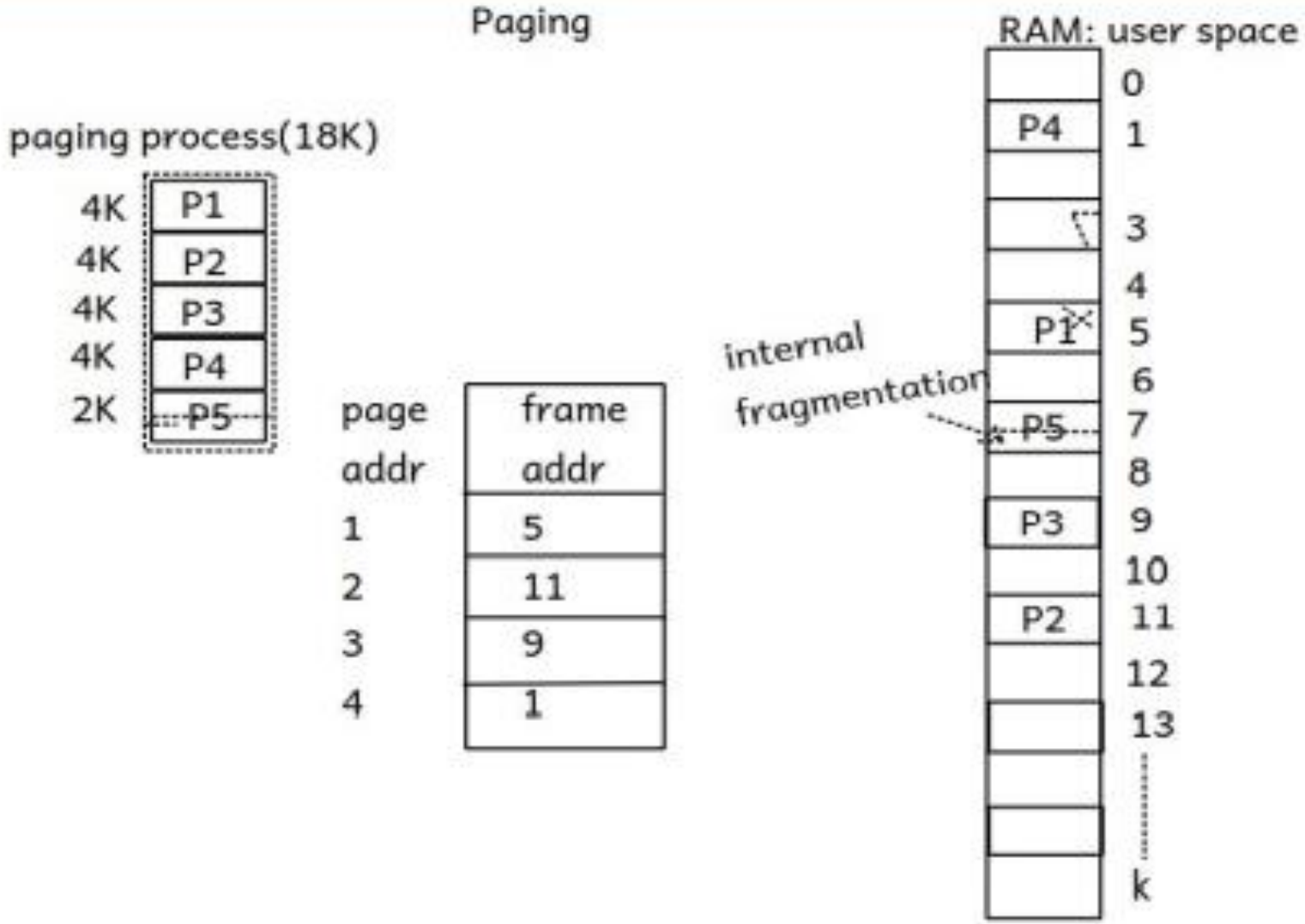
1. Segmentation

- In this technique, process in its logical memory is divided into small size segments **like stack segment, heap segment, data segment, bss segment , rodata segment, code segment etc...,** and when process is requesting for memory it is not requesting memory contiguously for whole process, memory gets allocated contiguously only for small size segments, and segments of one process may get load into the memory randomly at any locations, i.e. for a process memory gets allocated **in a non - contiguous manner**, and then only an execution of a process can be completed.
- As segments of a one process gets loaded randomly into the main memory, and in a system thousands processes are running at a time, hence to keep track on all the segments of each process, an OS maintains one table per process referred as **a segmentation table in which information about all the segments of that process can be kept**.
- Using segmentation an **external fragmentation can be reduced but cannot be completely avoided => to overcome this limitation paging technique has been designed**.


Segmentation
Big size process gets divided logically into small size segments
Process: Size 7 K
0 stack 1K
1 heap 1K
2 rodata 1K
3 data 2K
4 code 2K
kernel space
segment table
seg addr limit base
0 1K 24000
1 1K 28000
2 1K 20000
3 2K 7000
4 2K 13000
5 null null
Main Memory
30K
heap
28K
25K
stack
24K
21K
rodata
20K
15K
code
13K
8K
data
7K
5K
PCB
0
user space
kernel space

2. Paging :

- In this technique, physical memory (i.e. user space of a main memory) is divided into fixed size of blocks referred as **frames**, and process's logical memory space is divided into same size of blocks referred as **pages,** whereas maximum size of page must be equal to size of frame, i.e.
- **if e.g. size of frame = 4K, then maximum size of each page must be 4K, size of page may be less than 4K.**
- As process is divided into pages, so when it is requesting for memory, pages of one process may gets loaded into the main memory at any free frames, and for a process memory gets allocated in a non - contiguous manner.
- As pages of a one process gets loaded randomly into the main memory, and in a system thousands processes are running at a time, so to keep track on all the pages of each process, an OS maintains one table per process referred as **a page table** in which information about all the pages of that process can be kept.
- **There is no external fragmentation in paging.**
- **Internal fragmentation** may **exists in paging** when the size of page is less than size of frame.

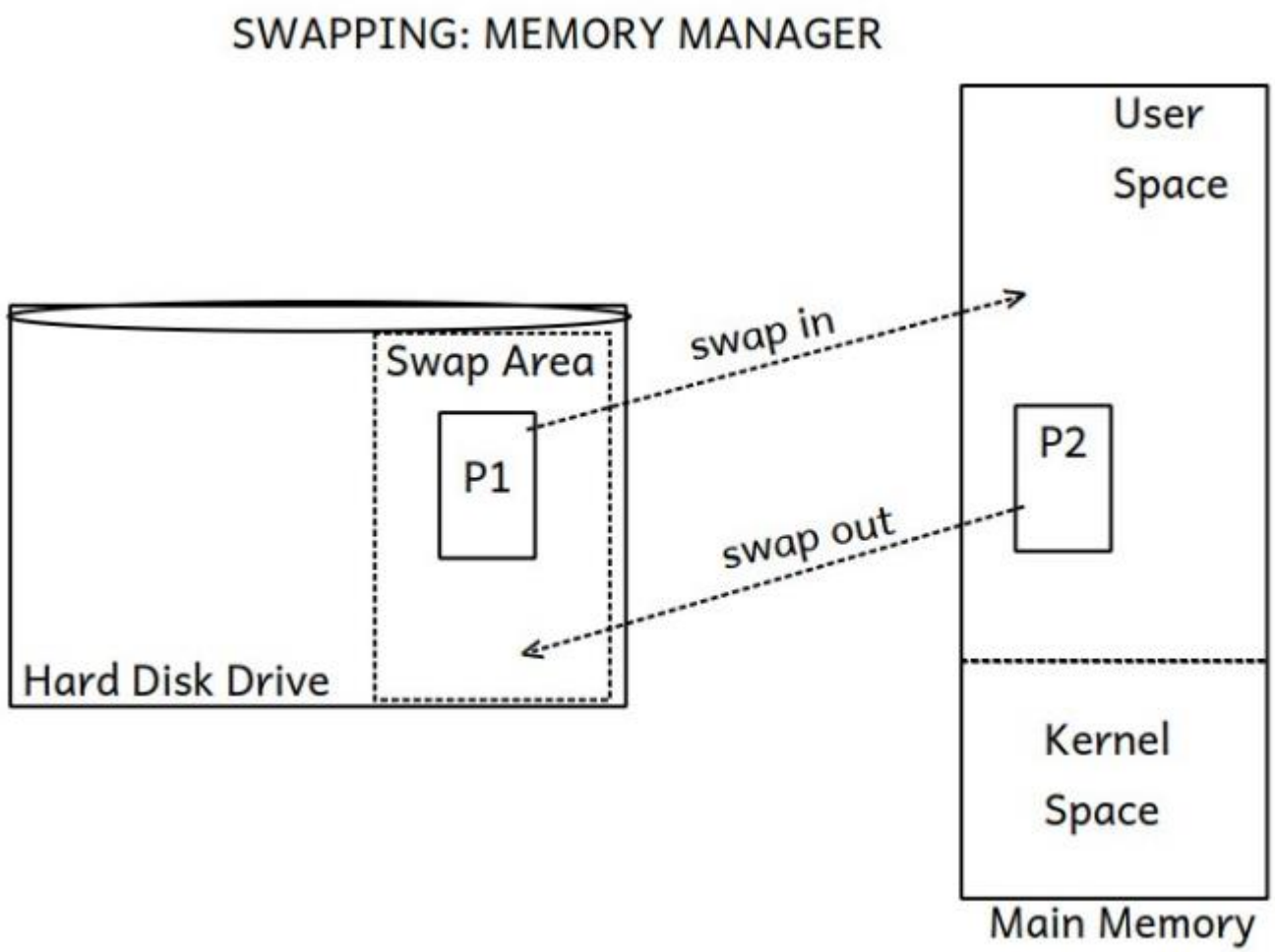

Paging
paging process(18K)
4K P1
4K P2
4K P3
4K P4
2K P5
page addr
frame addr
1 5
2 11
3 9
4 1
RAM: user space
internal fragmentation
0
1
3
4
5
6
7
8
9
10
11
12
13
k
P4
P1
P5
P3
P2

➢Swapping:

- **Swap area**: it is a portion of the hard disk drive (keep reserved while installation of an OS) can be used by an OS as an extension of the main memory in which inactive running programs can be kept temporarily and as per request processes can be swapped in and swapped out between swap area and the main memory by system program named as memory manager.
- In Linux swap area can be maintained in the form of swap partition, whereas in Windows swap area can be maintained in the form of swap files.
- **Conventionally size of the swap area should be doubles the size of the main memory,** i.e. if the size of main memory is 2 GB then size of swap area should be 4 GB, if the size of main memory is 4 GB then size of swap area should be 8 GB and so on.

➢Swapping:

- **Swap area**: it is a portion of the hard disk drive (keep reserved while installation of an OS) can be used by an OS as an extension of the main memory in which inactive running programs can be kept temporarily and as per request processes can be swapped in and swapped out between swap area and the main memory by system program named as memory manager.
- In Linux swap area can be maintained in the form of swap partition, whereas in Windows swap area can be maintained in the form of swap files.
- **Conventionally size of the swap area should be doubles the size of the main memory,** i.e. if the size of main memory is 2 GB then size of swap area should be 4 GB, if the size of main memory is 4 GB then size of swap area should be 8 GB and so on.


SWAPPING: MEMORY MANAGER
User
Space
swap in
Swap Area
P1
P2
swap out
Hard Disk Drive
Kernel
Space
Main Memory

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com