Process Management

Unit-2

Operating System/01CE1401

Department of Computer Engineering

Prof. Shilpa Singhal

Topics Covered

- Program
- Process
- Address Space
- Process Model
- Process States / Process Life Cycle
    - Two State
    - Three State
    - Five State
    - Seven State
- Process Control Block (PCB)
- Process Scheduling
    - Preemptive Scheduling
    - Non Preemptive Scheduling
- Process Schedulers
- Context Switching / Process Switching

Program

- Program contains a set of instructions designed to complete a specific task.
- Program exists at a single place and continues to exist until it is deleted.
- A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.
- A part of a computer program that performs a well-defined task is known as an algorithm. A collection of computer programs, libraries and related data are referred to as a software.

Program

A program is a piece of code which may be a single line o millions of lines. A computer program is usually written by a computer programmer in a programming language. For example here is a simple program written in C programming language –

Program:

```
#include <stdio.h> int main()

{

   printf("Hello, World! \n");

   return 0;

}
```

Process

- ❑ A process is a program in execution.
- ❑ A process generally also includes the process **stack**, which contains temporary data (such as function parameters, return addresses, and local variables), and a **data section**, which contains global variables.
- ❑ Or an entity that can be assigned and execute on a processor.
- ❑ It can be also defined as instance of the program running in the computer.

Process

- ❑ Program is passive entity stored on disk (executable file), process is active
- ❑ Program becomes process when executable file loaded into memory.
- ❑ Processes that are running in the background mode (e.g.checking email) are known as Daemons.
- ❑ Daemons are processes that run unattended. They are constantly in the background and are available at al times. Daemons are usual y started when the system starts, and they run until the system stops. A daemon process typical y performs system services and is available at al times to more than one task or user.

Eg: Crond, syslogd

Difference B/W zombie, orphan and daemon: https://www.tutorialspoint.com/zombie-vs-orphan-vs-daemon-processes

How is a program converted into the process?

The program can be converted into the process by following three actions

- Compiler
- Linker
- Loader

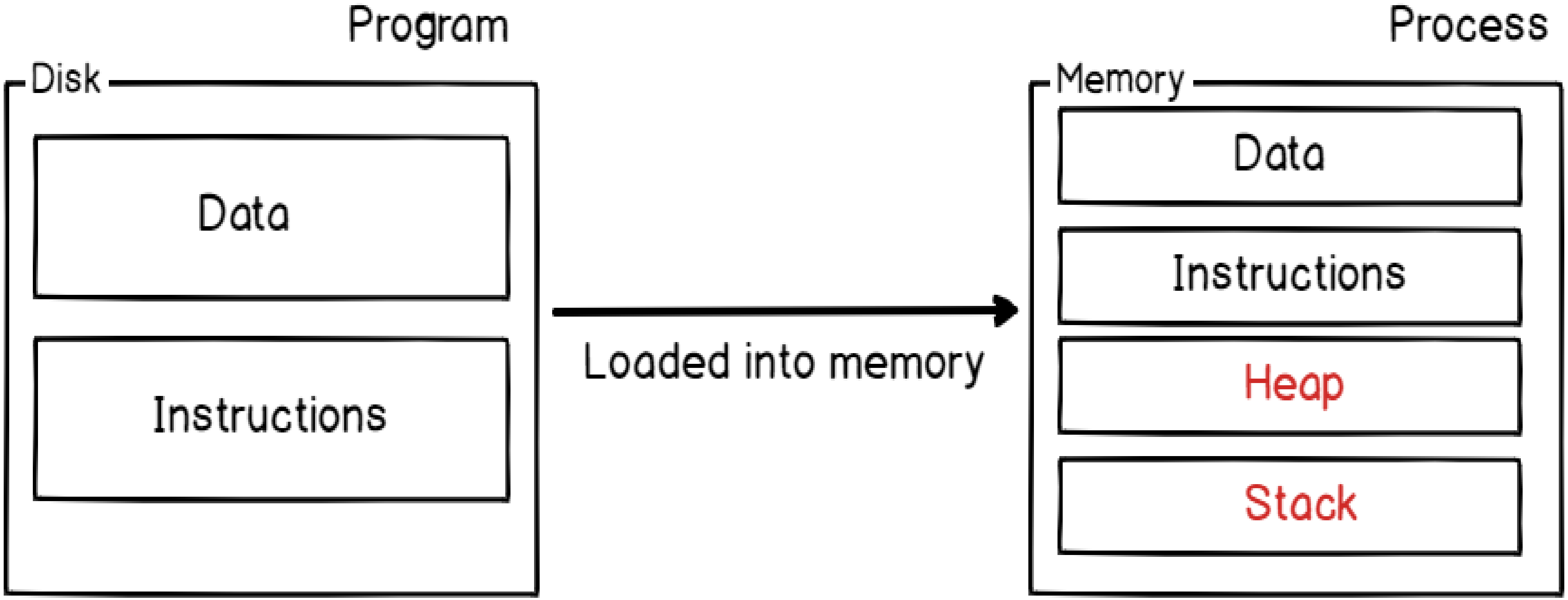
How is a program converted into the process?

Compiler
The compiler converts the high-level language code into the low-level language code or object code.
The compiler creates the .o file.

Linker
Linker converts the object code or low-level language code into the executable code.
The linker creates the .exe file.

Loader
The loader loads the executable code into the main memory and allocates the address space for the process.
A program contains code and data sections.

Program vs Process



Program vs Process

S. No.	Program	Process
1.	Program contains a set of instructions designed to complete a specific task.	Process is an instance of an executing program.
2.	Program is a passive entity as it resides in the secondary memory.	Process is a active entity as it is created during execution and loaded into the main memory.
3.	Program exists at a single place and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.
4.	Program is a static entity.	Process is a dynamic entity.
5.	Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.
6.	Program does not have any control block.	Process has its own control block called Process Control Block.

Address Space

What does a process look like in memory?

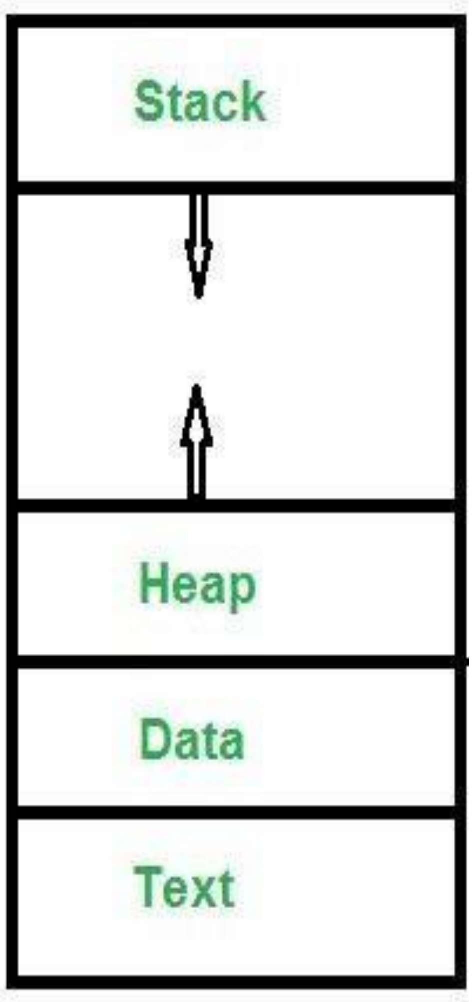
The address space is divided into 4 sections:

1.Stack: The stack contains temporary data, such as method/function parameters, returns addresses, and local variables.

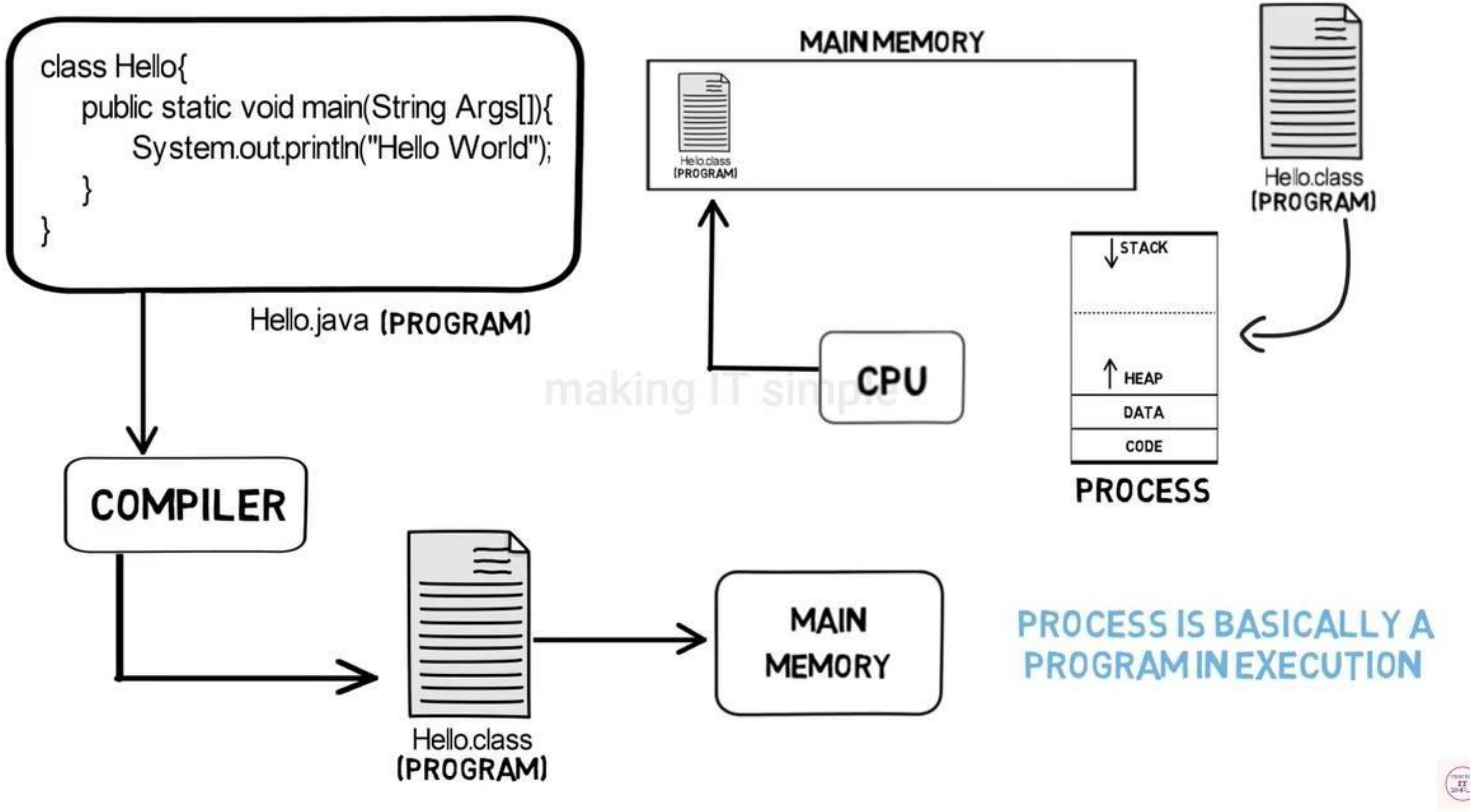
2.Heap Section: Dynamically allocated memory to process during its run time.

3.Data Section: Contains the global variable.

4.Text / Code Section: A Process, sometimes known as the Text Section, also includes the current activity represented by the value of the Program Counter.



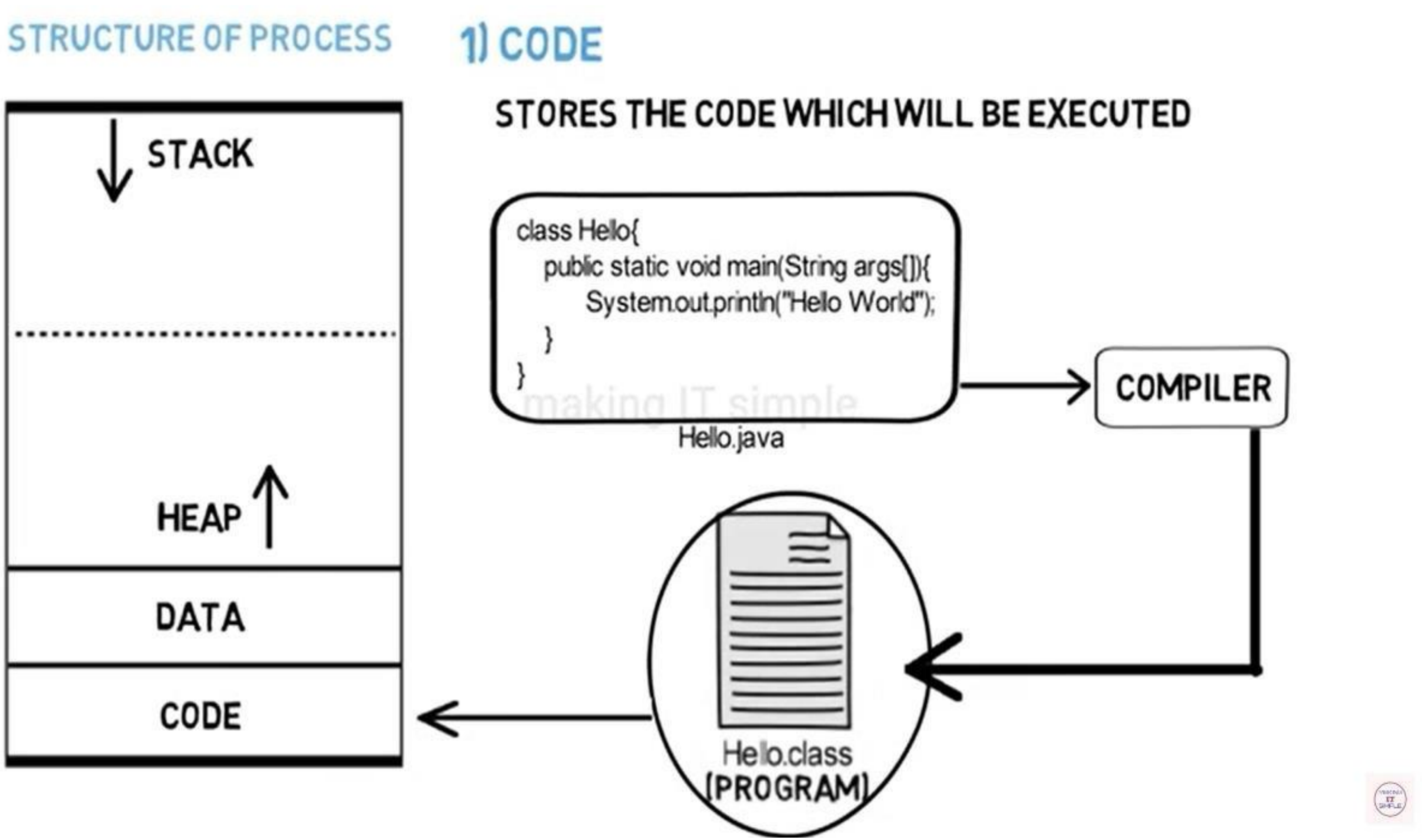
Address Space



```
class Hello{
    public static void main(String Args[]){
        System.out.println("Hello World");
    }
}
```

PROCESS IS BASICALLY A PROGRAM IN EXECUTION

Address Space 1.Code



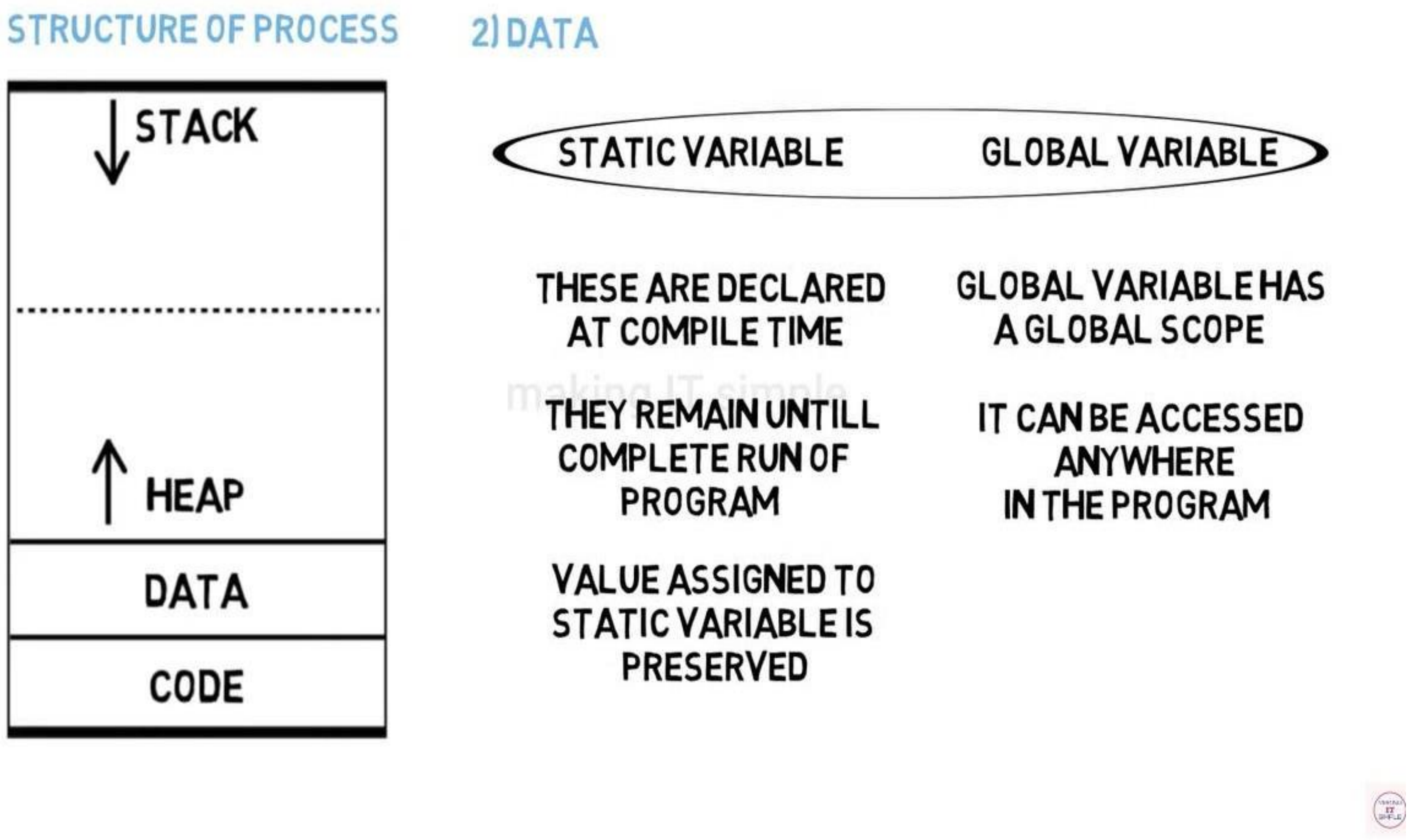
STRUCTURE OF PROCESS

1) CODE

STORES THE CODE WHICH WILL BE EXECUTED

```
class Hello{
   public static void main(String args[]){
       System.out.println("Hello World");
   }
}
```

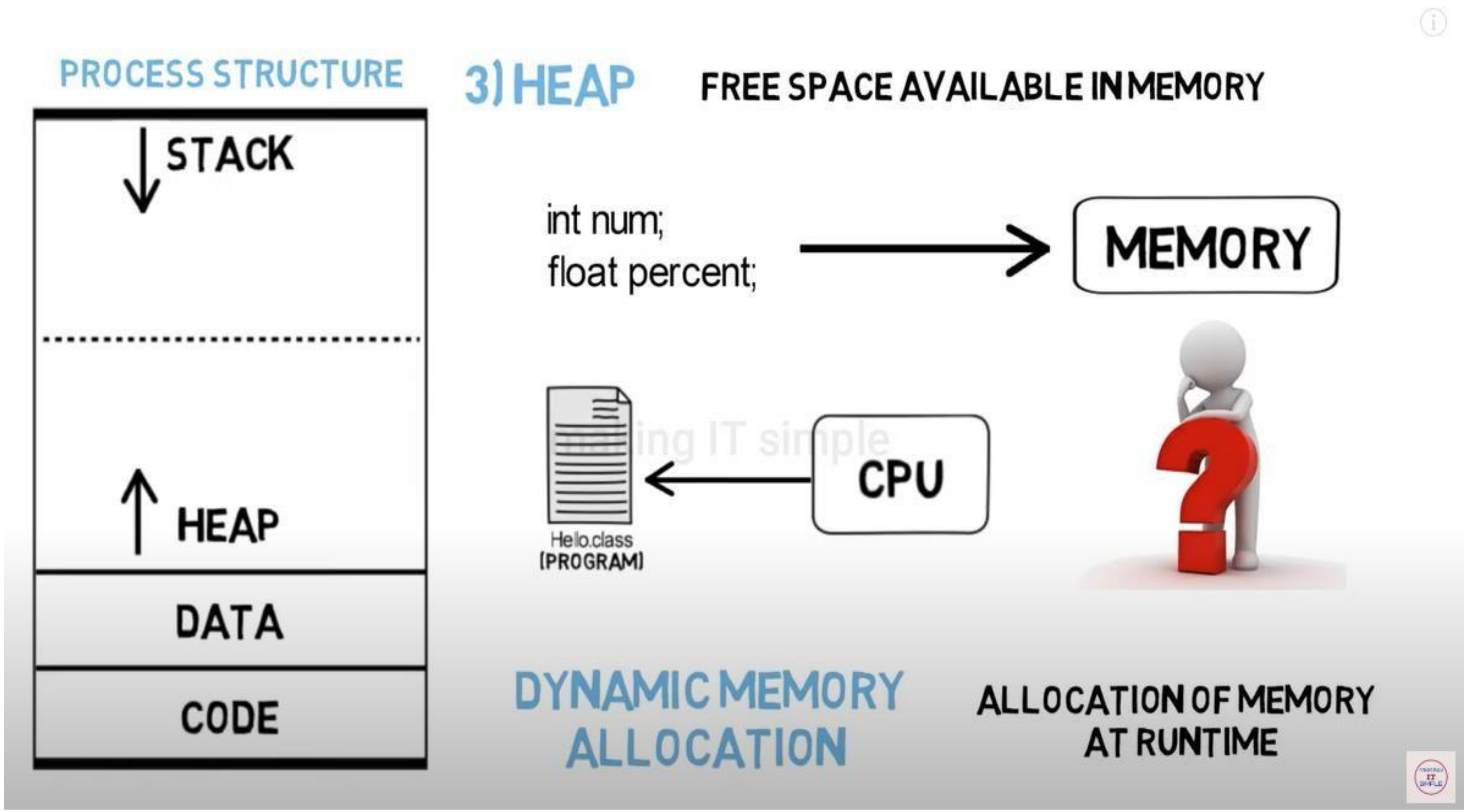
Address Space 2.Data



2) DATA

STATIC VARIABLE	GLOBAL VARIABLE
THESE ARE DECLARED AT COMPILE TIME	GLOBAL VARIABLE HAS A GLOBAL SCOPE
THEY REMAIN UNTILL COMPLETE RUN OF PROGRAM	IT CAN BE ACCESSED ANYWHERE IN THE PROGRAM
VALUE ASSIGNED TO STATIC VARIABLE IS PRESERVED	

Address Space 3.Heap



PROCESS STRUCTURE

STACK

HEAP

DATA

CODE

3) HEAP FREE SPACE AVAILABLE IN MEMORY

```
int num;
float percent;
```

MEMORY

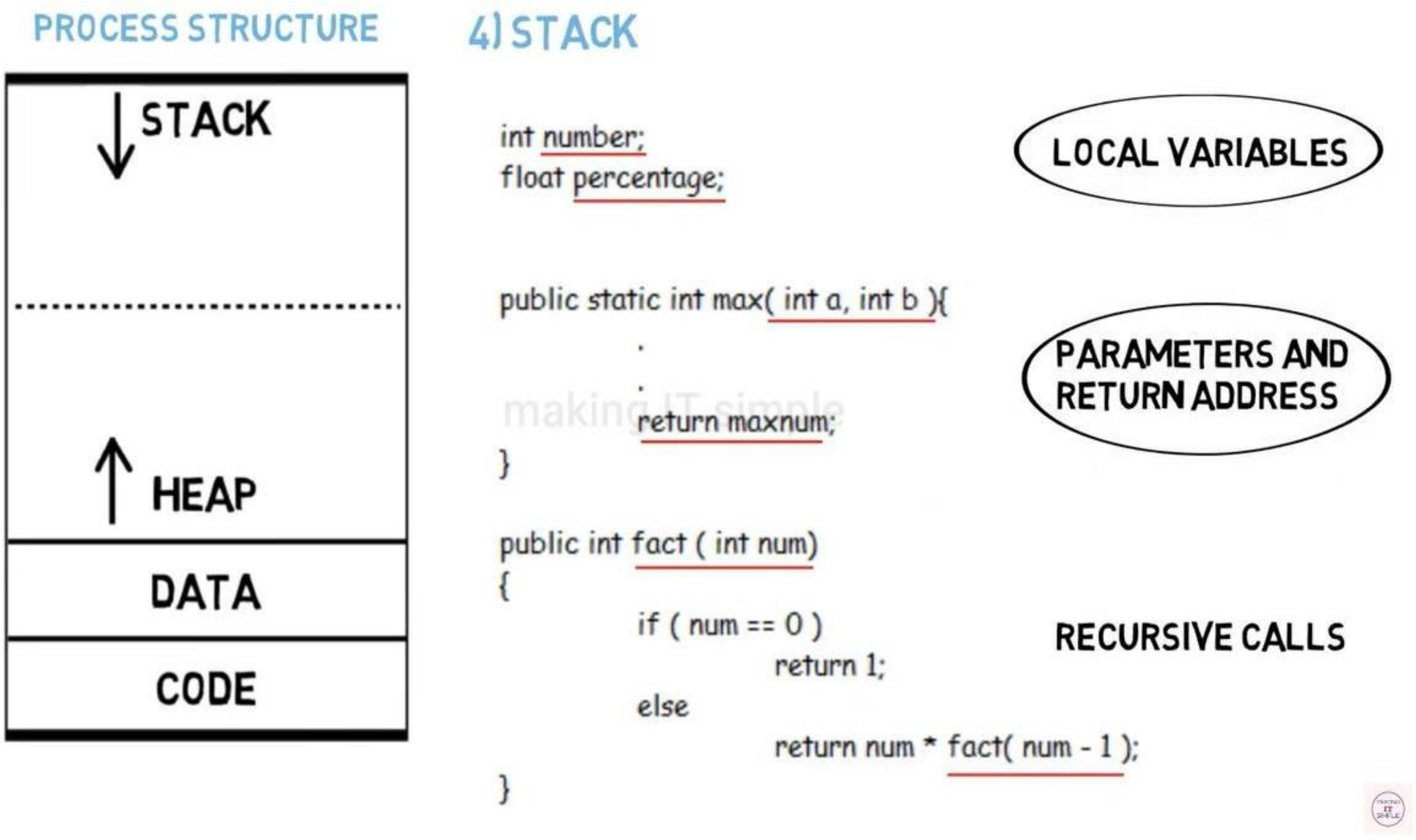
CPU

Hello.class
(PROGRAM)

DYNAMIC MEMORY ALLOCATION

ALLOCATION OF MEMORY AT RUNTIME

Address Space 4.Stack



4) STACK

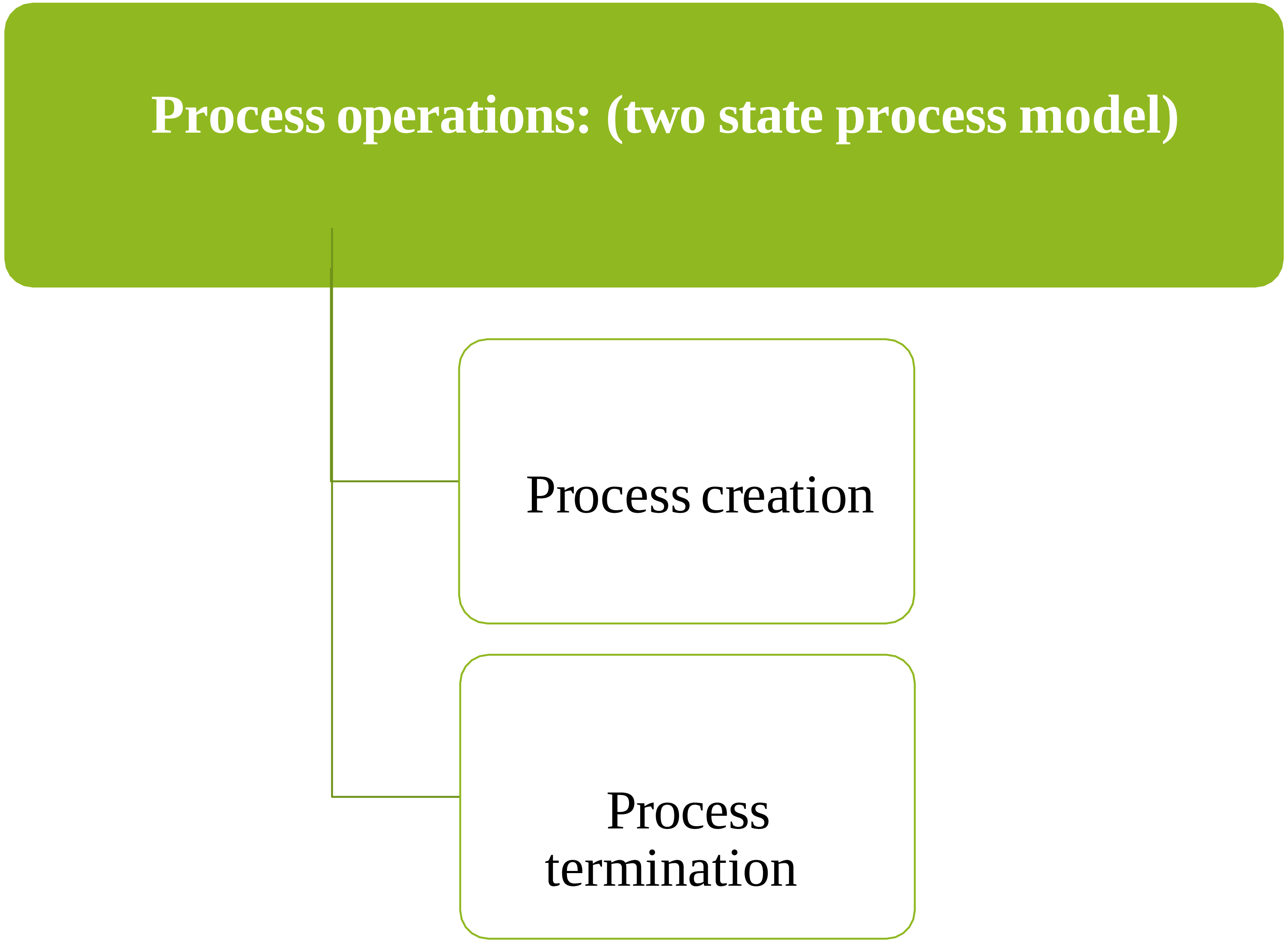
```
int number;
float percentage;


public static int max( int a, int b ){
            .
            .
            return maxnum;
}

public int fact ( int num)
{
            if ( num == 0 )
                        return 1;
            else
                        return num * fact( num - 1 );
}
```

LOCAL VARIABLES

PARAMETERS AND RETURN ADDRESS

RECURSIVE CALLS

Process Model



Process operations: (two state process model)

Process creation

Process termination

1.Process Creation

1.System initialization:

- When OS is booted, several system process are created. They provide system services. They do not need user interaction it just execute in background.

2.Execution of a process creation system call:

- A running process can issue system call to create new process. i.e in unix system call name “fork” is used to create new process.

1.Process Creation

3.A user request to create a new process

- User can start new process by typing command or double click on some application icon.

4.Initiation of a batch job:

- This applies only to the batch system. Here user submit batch jobs to the system. When there are enough resources are free to execute a next job a new job is selected from batch queue.

2.Process Termination

1.Normal exit (voluntary)

- Terminate when done their job. i.e. when all the instructions from program get executed.
- Even user can select option to terminate the process.

2.Error exit (voluntary):

- Process even terminated when some error occurred. Example divide by zero error.

2.Process Termination

3.Fatal error (involuntary):

- Fatal error are generated due to user mistake in executing program. Such as file name not found error.

4.Killed by another process (involuntary):

- If some other process request for OS to kill the process. Then UNIX it can be done by “kill” system call while in windows “TerminateProcess”

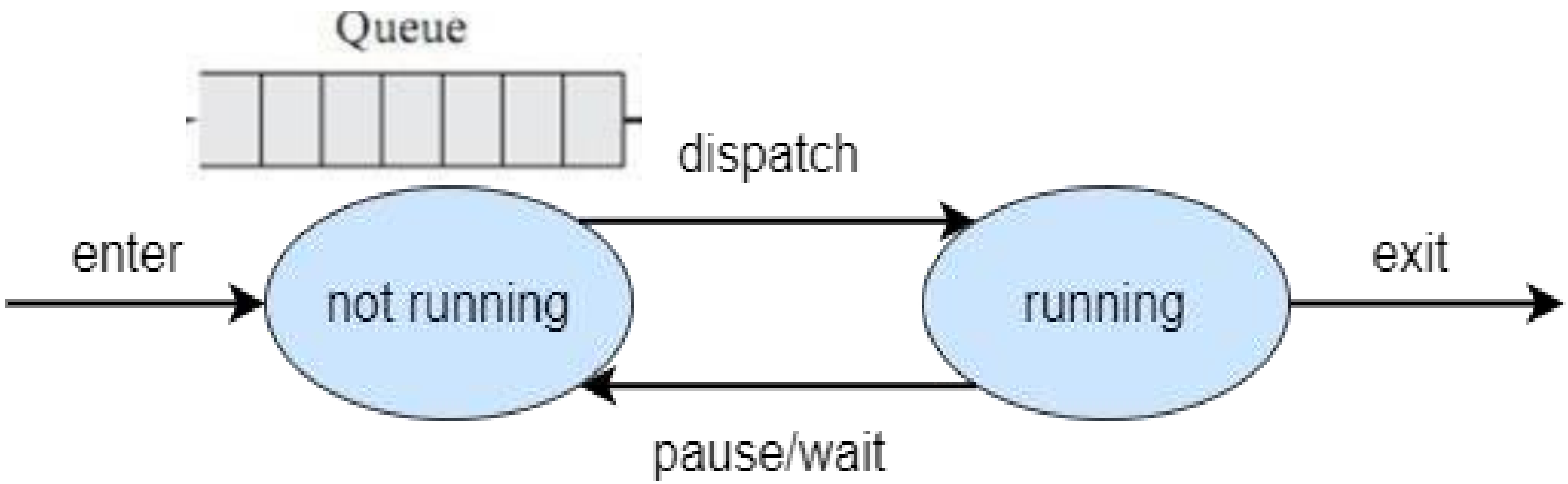
Process States / Process Life Cycle

- When a process executes, it passes through different states.
- These stages may differ in different operating systems, and the names of these states are also not standardized.
- When process executes, it changes state.
- Process state is defined as the current activity of the process.

1.Two State Process Model

Two State Process Model consists of two states:

- Not-running State: Process waiting for execution.
- Running State: Process currently executing.



1.Two State Process Model

- ❑ **NOT RUNNING**

  When O.S. is create the process and enter into the main memory for execution this time its state will be NOT RUNNING and these processes are waiting for the execution

- ❑ **RUNNING**

  When process is going for execution its state is called RUNNING.

1.Two State Process Model

- When a process is first created by the OS, it initializes the program control block for the process and the new process enters the system in Not-running state.
- After some time, the currently running process will be interrupted by some events, and the OS will move the currently running process from Running state to Not-running state.
- The dispatcher then selects one process from Not-running processes and moves the process to the Running state for execution.

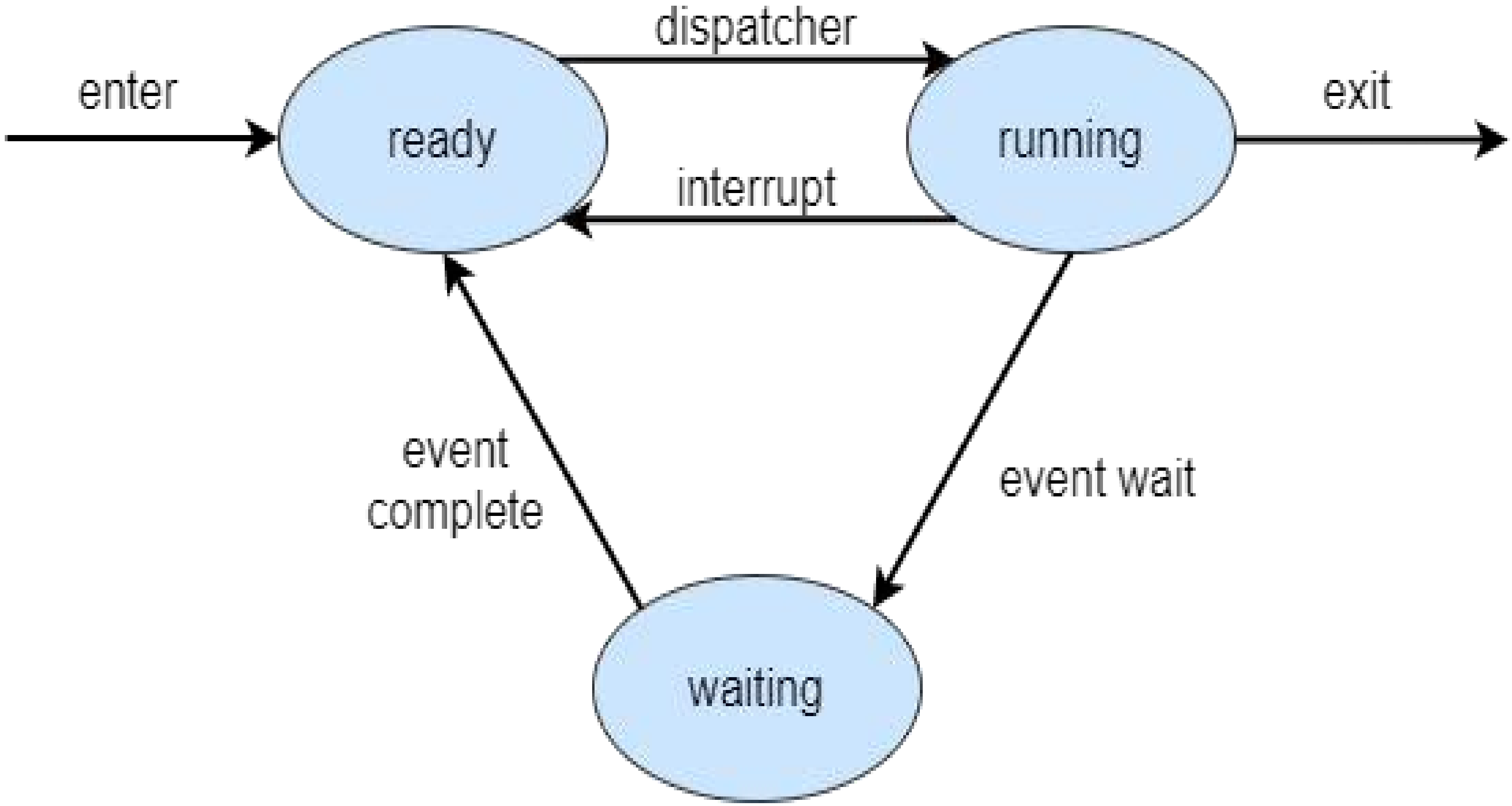
1.Two State Process Model

- For example, if we implement round-robin scheduling then the running process moves to not-running state after the time quantum.
- When a process finishes the execution, the process exits the system and the dispatcher again selects a new process and moves it to Running state.
- All the processes in the Not-running state are maintained in a queue.

2.Three State Process Model

- There is one major drawback of two state process model. When dispatcher brings a new process from not-running state to running state, the process might still be waiting for some event or I/O request.
- So, the dispatcher must traverse the queue and find a not-running process that is ready for execution. It can degrade performance.

2.Three State Process Model



Three State Process Model Transition Diagram

2.Three State Process Model

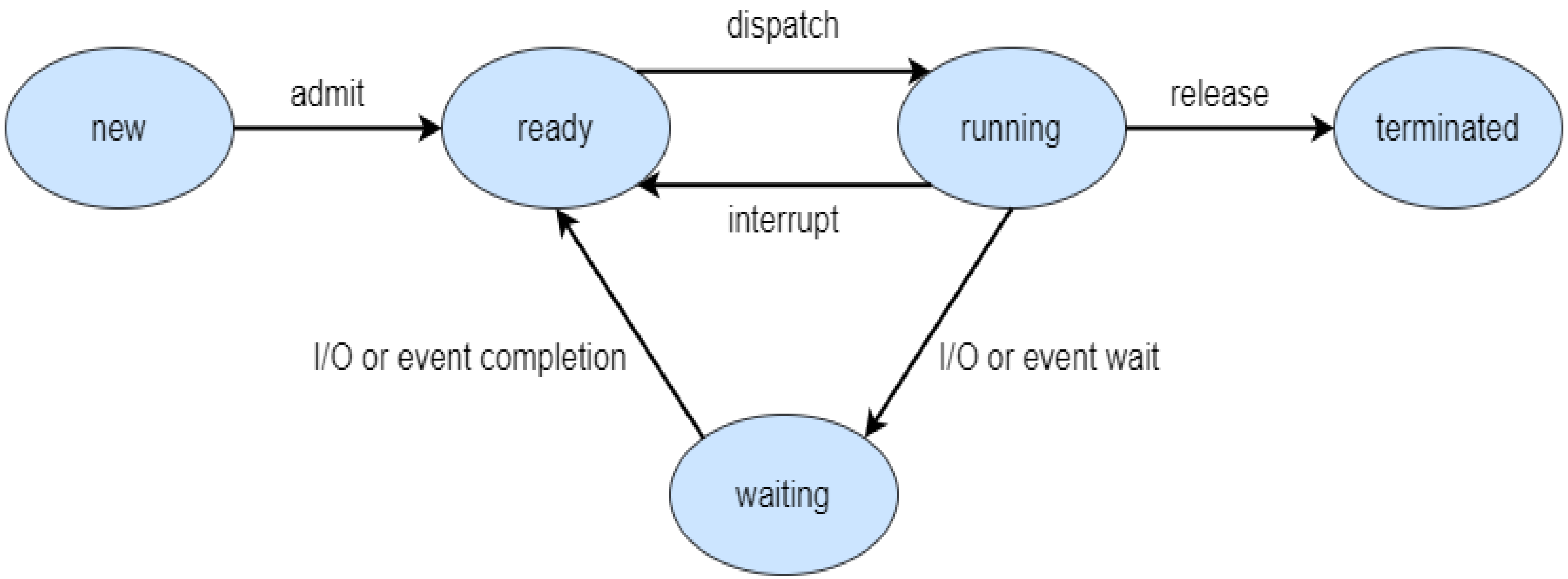
- To overcome this problem, we split the not-running state into two states: Ready State and Waiting (Blocked) State.
  - Ready State: The process in the main memory that is ready for execution.
  - Waiting or Blocked State: The process in the main memory that is waiting for some event.
- The OS maintains a separate queue for both Ready State and Waiting State.
- A process moves from Waiting State to Ready State once the event it’s been waiting for completes.

2.Three State Process Model

1. **Ready State**– A state in which a process is ready and waiting for its execution.
2. **Blocked State**– A state in which a process doesn’t execute until and unless a process event occurs, like completion of an Input/Output operation.
3. **Running State**– A state in which the process is currently executing.

3.Five State Process Model

In the five state model, we introduce two new states: new state and terminated state.



Five-State Process Model State Transition Diagram

3.Five State Process Model

1. **Running:** The currently executing process.
2. **Waiting/Blocked:** Process waiting for some event such as completion of I/O operation, waiting for other processes, synchronization signal, etc.
3. **Ready:** A process that is waiting to be executed.
4. **New:** The process that is just being created. The Program Control Block is already being made but the program is not yet loaded in the main memory. The program remains in the new state until the long term scheduler moves the process to the ready state (main memory).
5. **Terminated/Exit:** A process that is finished or aborted due to some reason.

3.Five State Process Model

Reason for New State

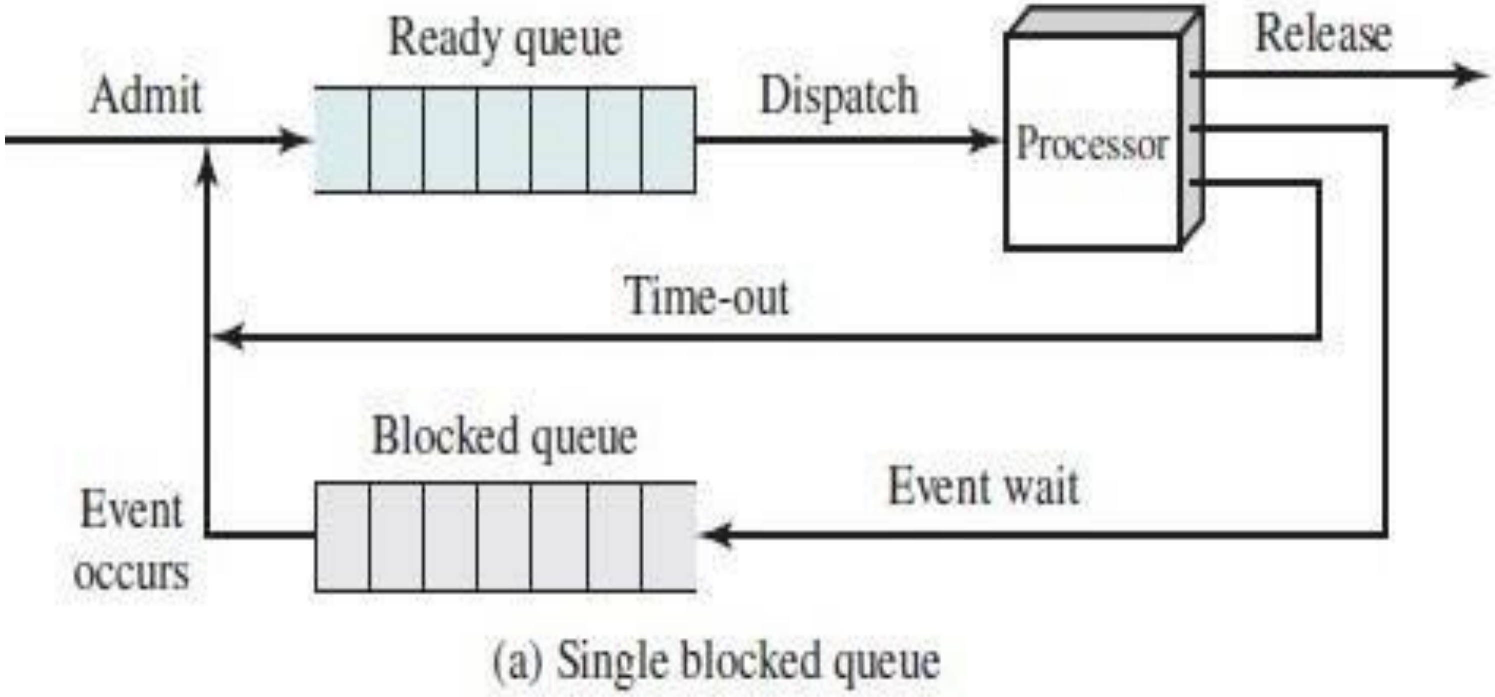
- In the previous models, we assumed that the main memory is large enough to accommodate all programs but this is not true. Modern programs are very large. Loading all processes in the main memory is not possible.
- When a new process is made, its program is not loaded in the main memory. The OS only stores some information about the process in the main memory. The long term scheduler moves the program to the main memory when sufficient space is available. Such a process is said to be in new state.

3.Five State Process Model

Reason for Terminated State

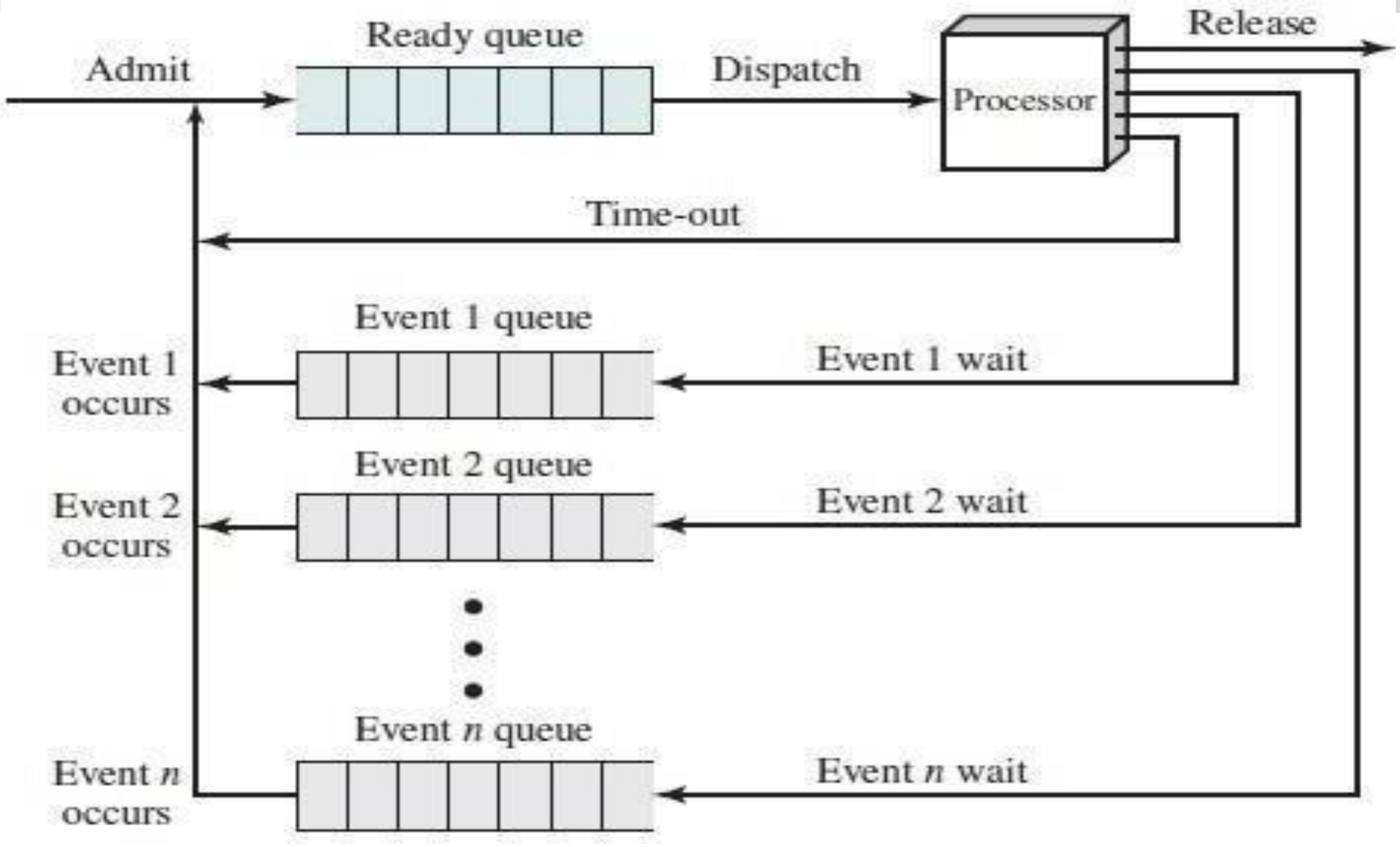
- In the previous models, when a process finishes execution, its resources are immediately freed. But there might be some other process that may need its data in the future.
- For example, when a child process finishes execution, the OS preserves its data until the parent call wait(). The child process is still in memory but not available for execution. The child process is said to be in terminated state.

3.Five State Process Model



Five State Process Model Queuing Diagram (With single blocked queue)

3.Five State Process Model



Five State Process Model Queuing Diagram (With multiple blocked queue)

3.Five State Process Model

State Transitions

- **New -> Ready:**
  - The long term scheduler picks up a new process from second memory and loads it into the main memory when there are sufficient resources available. The process is now in ready state, waiting for its execution.
- **Ready -> Running:**
  - The short term scheduler or the dispatcher moves one process from ready state to running state for execution.

3.Five State Process Model

terminated

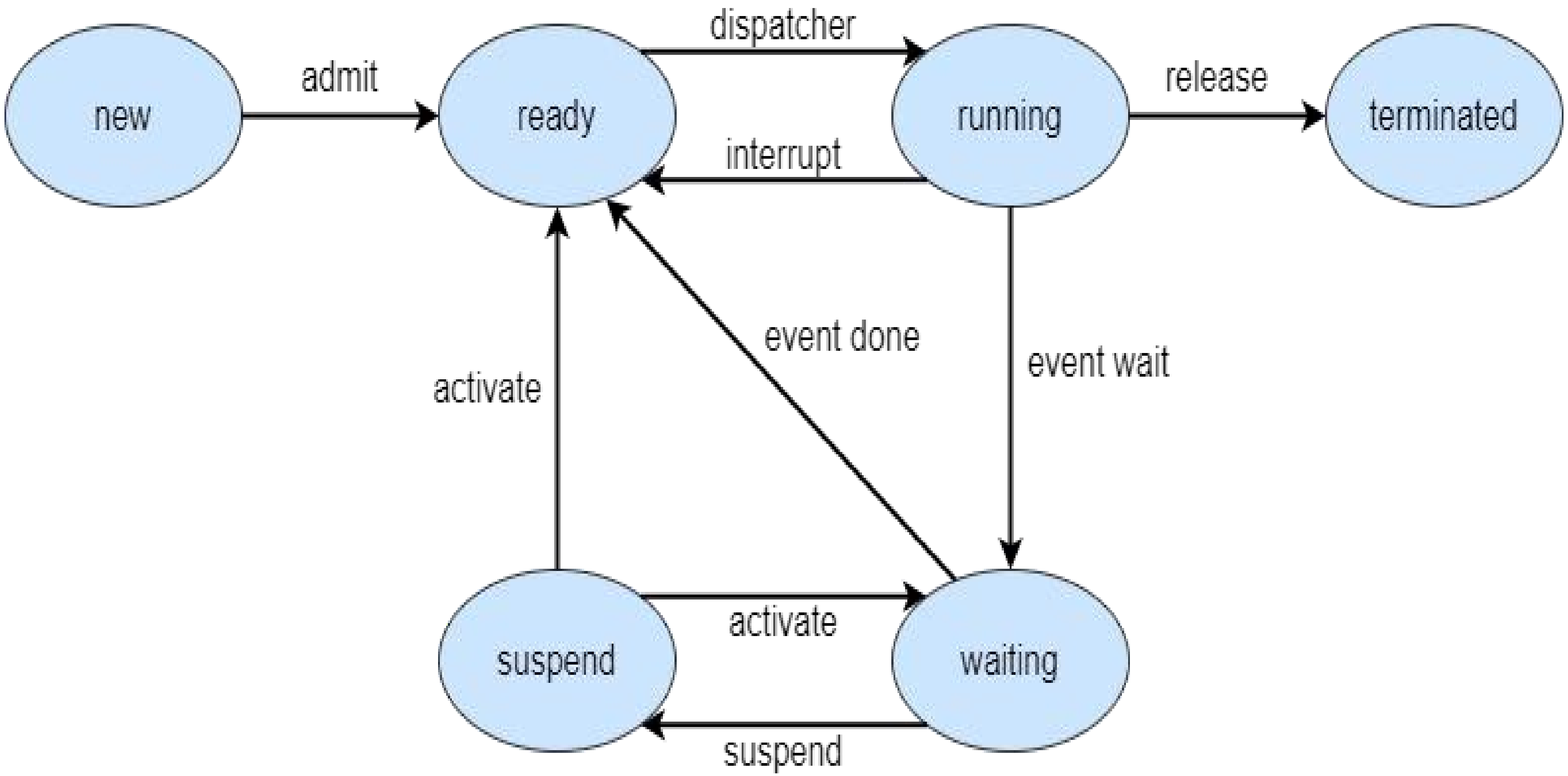
- **Running -> Terminated:**
  - The OS moves a process from running state to state if the process finishes execution or if it aborts.
- **Running -> Ready:**
  - This transition can occur when the process runs for a certain amount of time running without any interruption. For example, if we use round-robin to schedule processes, then the running process will move to the ready state after time quantum. Another example is if the priority of a process in the ready state is more than the priority of the currently running process, then OS may preempt the running process and move it to ready state.

3.Five State Process Model

- **Running -> Waiting:**
  - A process is put in the waiting state if it must wait for some event. For example, the process may request some resources or memory which might not be available. The process may be waiting for an I/O operation or it may be waiting for some other process to finish before it can continue execution.
- **Waiting -> Ready:**
  - A process moves from waiting state to ready state if the event the process has been waiting for, occurs. The process is now ready for execution.

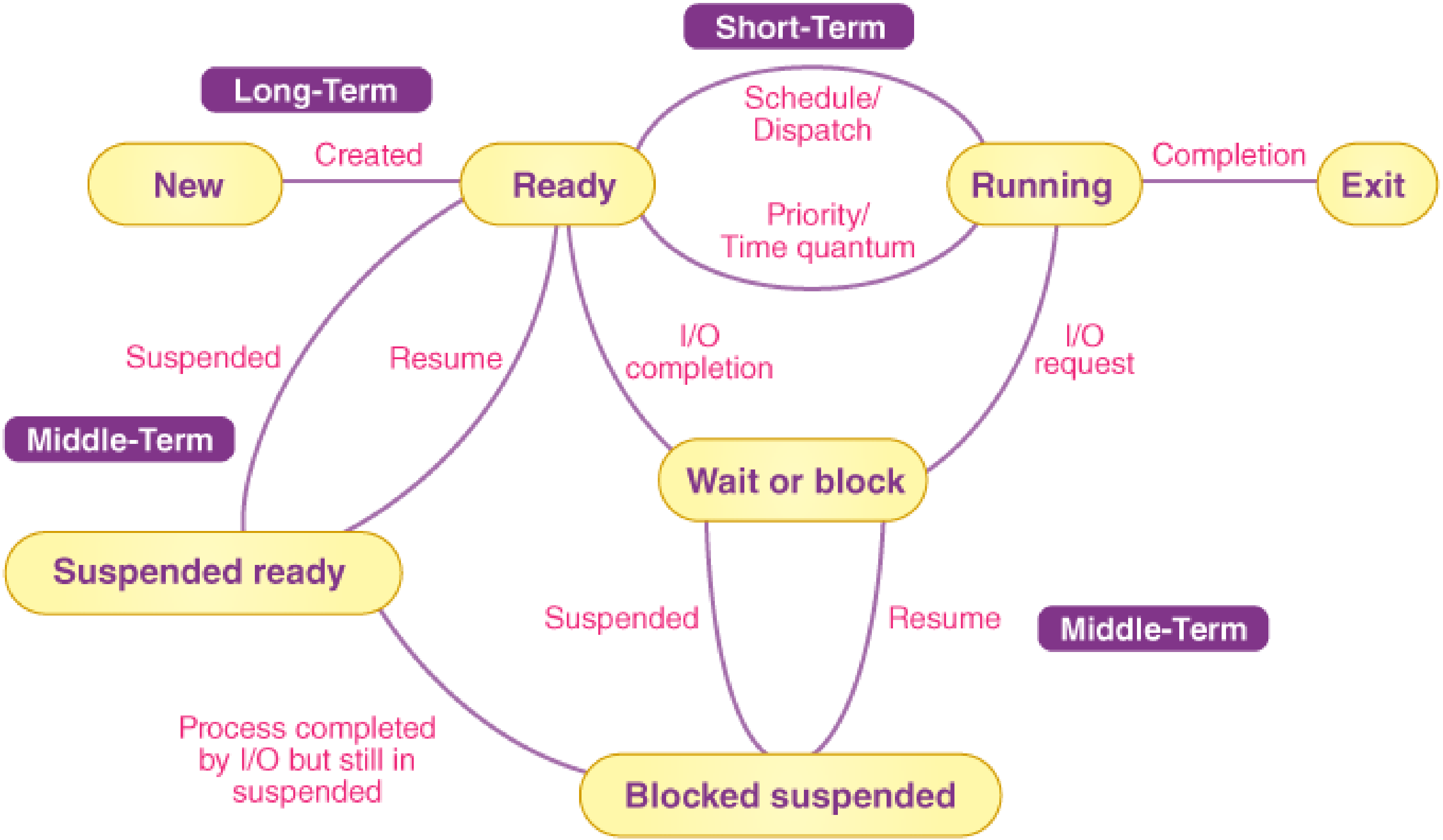
4.Six State Process Model

It is commonly as Five state process model with suspend state.



Six State Process Model Transition Diagram

5.Seven State Process Model



Animated video of 7state process model : https://www.youtube.com/watch?v=OP4WG5dRIXA

5.Seven State Process Model

1. **New (Create)** – In this step, the process is about to be created but not yet created, it is the program which is present in secondary memory that will be picked up by OS to create the process.
2. **Ready** – New -> Ready to run. After the creation of a process, the process enters the ready state i.e. the process is loaded into the main memory. The process here is ready to run and is waiting to get the CPU time for its execution. Processes that are ready for execution by the CPU are maintained in a queue for ready processes.
3. **Run** – The process is chosen by CPU for execution and the instructions within the process are executed by any one of the available CPU cores.
4. **Blocked or wait** – Whenever the process requests access to I/O or needs input from the user or needs access to a critical region(the lock for which is already acquired) it enters the blocked or wait state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the ready state.
5. **Terminated or completed** – Process is killed as well as PCB is deleted.

5.Seven State Process Model

6. Suspend ready – Process that was initially in the ready state but was swapped out of main memory(refer Virtual Memory topic) and placed onto external storage by scheduler is said to be in suspend ready state. The process will transition back to ready state whenever the process is again brought onto the main memory.

7. Suspend wait or suspend blocked – Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished it may go to suspend ready.

5.Seven State Process Model

- It is commonly known as Five state process model with two suspended states.
- There is one major drawback in the previous process state model. That is, the CPU doesn’t know which process in the suspend queue is ready for execution. CPU may swap a process that is still waiting for event completion from secondary memory back to the main memory. There is no point in moving a blocked process back to the main memory. The performance suffers.

5.Seven State Process Model

- To avoid this, we divide the suspend state into 2 states:
  - Blocked/Suspend: The process is in secondary memory but not yet ready for execution.
  - Ready/Suspend: The process is in secondary memory and ready for execution.

5.Seven State Process Model

State Transitions:

Blocked-> Blocked/Suspend :

- ○ If all the processes in the main memory are in the waiting state, the processor swaps out at least one waiting process back to secondary memory to free memory to bring another process.

- **Blocked/Suspend -> Blocked:**
  - ○ This transition might look unreasonable but if the priority of a process in Blocked/Suspend state is greater than processes in Ready/Suspend state then CPU may prefer process with higher priority.

5.Seven State Process Model

- **Blocked/Suspend -> Ready/Suspend:**
  - The process moves from Blocked/Suspend to Ready/Suspend state if the event, the process has been waiting for occurs.
- **Ready/Suspend -> Ready:**
  - The OS moves a process from secondary memory to the main memory when there is sufficient space available. Also, if there is a high priority process in Ready/Suspend state, then OS may swap it with a lower priority process in the main memory.

5.Seven State Process Model

- **Ready -> Ready/Suspend:**
  - The OS moves a process from the ready state to ready/suspended to free main memory for a higher priority process.

In all the process state models, a process can directly move from any state to terminated state. This is because the parent process can terminate the child process at any moment.

Q - 01

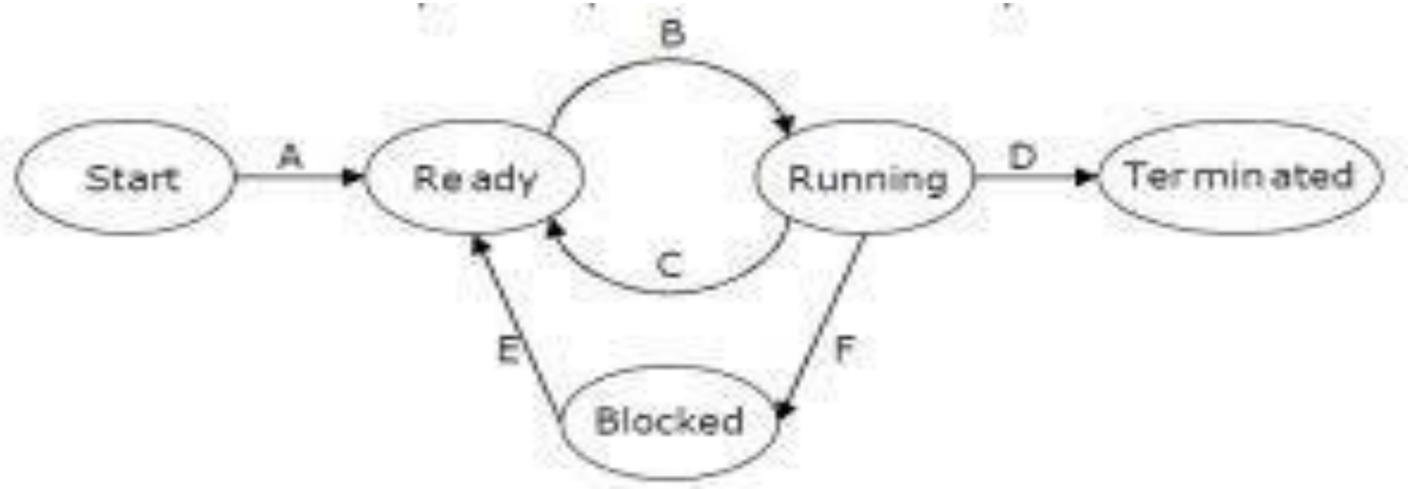
I. If a process makes a transition D, it would result in another process making transition A immediately.

II. A process P2 in blocked state can make transition E while another process P1 is in running state.

III. The OS uses preemptive scheduling.

IV. The OS uses non-preemptive scheduling.

Which of the above statements are TRUE?



Q- 1
Solution

- If a process makes a transition DD, it would result in another process making transition A immediately. - This is false. It is not said anywhere that one process terminates, another process immediately come into Ready state. It depends on availability of process to run & Long term Scheduler.
- A process P2 in blocked state can make transition E while another process P2 is in running state. - This is correct. There is no dependency between running process & Process getting out of blocked state.
- The OS uses preemptive scheduling. :- This is true because we got transition CC from Running to Ready.
- The OS uses non-preemptive scheduling. Well as previous statement is true, this becomes false.

So answer is II and III .

Q - 02

Process can get blocked from which of the following?

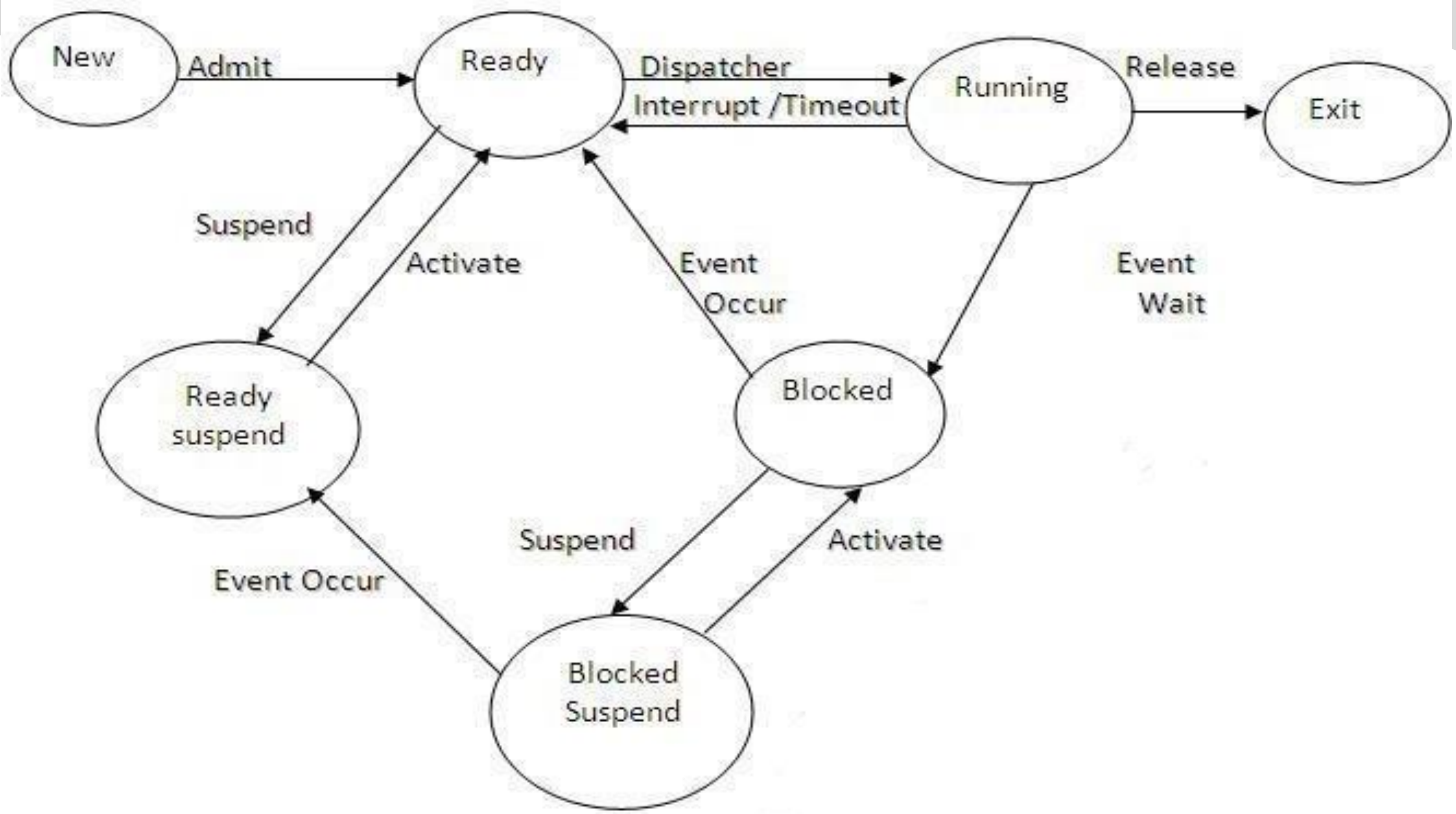
(A) Running or Suffered

(B) Ready or New

(C) Ready or Running

(D) terminate or New

Q – 02
Solution



here is a state called suspend ready, when system needs to free up resources, some processes are moved to that state from ready state

Answer-C

Process Control Block (PCB)

- PCB is a data structure in the OS and it contains all the information about a process.
- While creating a process the operating system performs several operations. To identify the processes, it assigns a process identification number (PID) to each process.
- As the operating system supports multiprogramming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process execution status.

Process Control Block (PCB)

Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc.

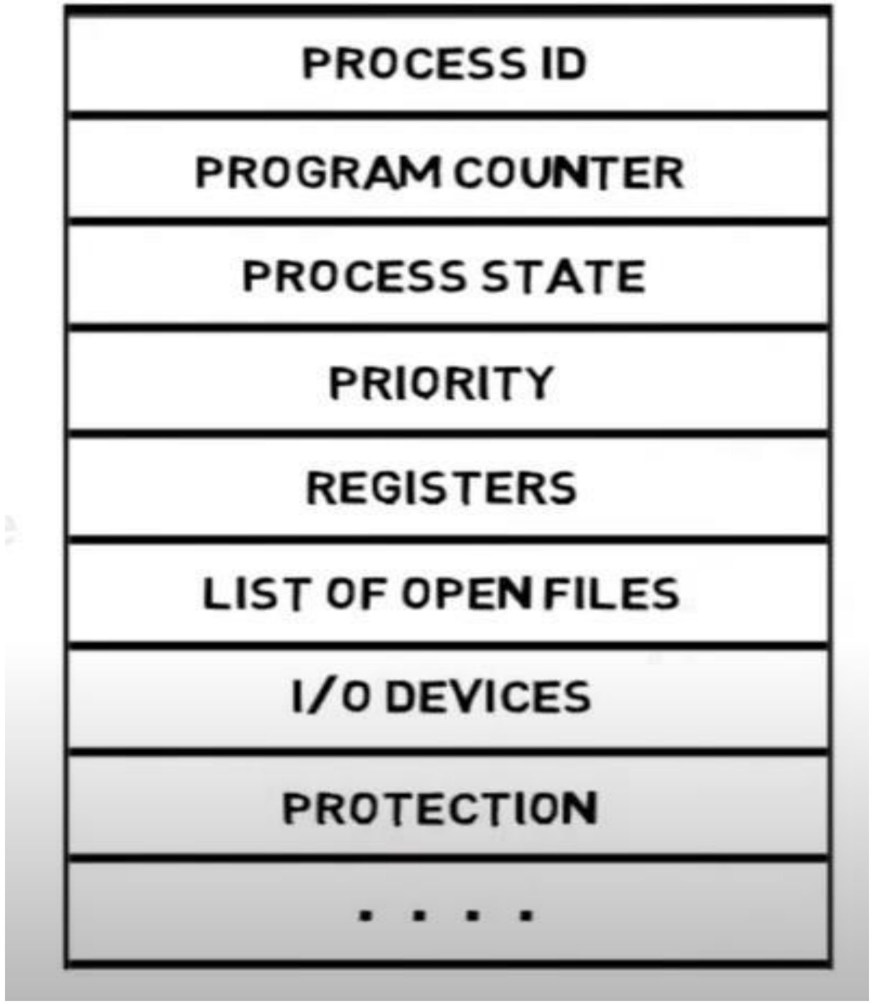
All these information is required and must be saved when the process is switched from one state to another. When the process makes a transitionfromone stateto another, the operating system must updateinformation in the process's PCB.

Process Control Block (PCB)

How PCB of a process is created in OS?

When a process is created, the operating system creates a corresponding process control block for storing the information of that process.

Process Control Block (PCB)



PROCESS ID
PROGRAM COUNTER
PROCESS STATE
PRIORITY
REGISTERS
LIST OF OPEN FILES
I/O DEVICES
PROTECTION
. . . .

Process Control Block (PCB)

- **Pointer** – It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.
- **Process state** – It stores the respective state of the process.
- **Process ID /number -**Every process is assigned with a unique id known as process ID or PID which stores the process identifier.
- **Program counter** – It stores the counter which contains the address of the next instruction that is to be executed for the process.

Process Control Block (PCB)

- **Register** – These are the CPU registers which includes: accumulator, base, registers and general purpose registers.
- **Memory limits** – This field contains the information about memory management system used by operating system. This may include the page tables, segment tables etc.
- **Open files list** – This information includes the list of files opened for a process.
- **Miscellaneous accounting and status data** – This field includes information about the amount of CPU used, time constraints, jobs or process number, etc.

Process Control Block (PCB)

- The process Control block stores the register content also known as execution content of the processor when it was blocked from running.
- This execution content architecture enables the operating system to restore a process’s execution context when the process returns to the running state.
- When the process makes a transition from one state to another, the operating system updates its information in the process’s PCB.

Process Control Block (PCB)

Process ID

- it process identifier (PID), parent process identifier and user identifier.

Priority

- its typically numeric value. A process is assigned a priority at its creation.

Process state

- the current state of the process.

Program counter

- indicate the address of the next instruction to be executed for process.

CPU register

- content of the register when the CPU was last released by the process.

Process Control Block (PCB)

CPU scheduling information

- it includes process priority, pointer to various scheduling queue, information about events on which process is waiting and other parameters.

Program Status Word (PSW)

- this is snapshot i.e. the image of the PSW when the CPU was last voluntarily leave by the process. Loading this snapshot back into the PSW would resume execution of the program.

Memory management information

- it includes values of *base and limit registers*, information about page table or segment table.

Process Control Block (PCB)

PCB pointer

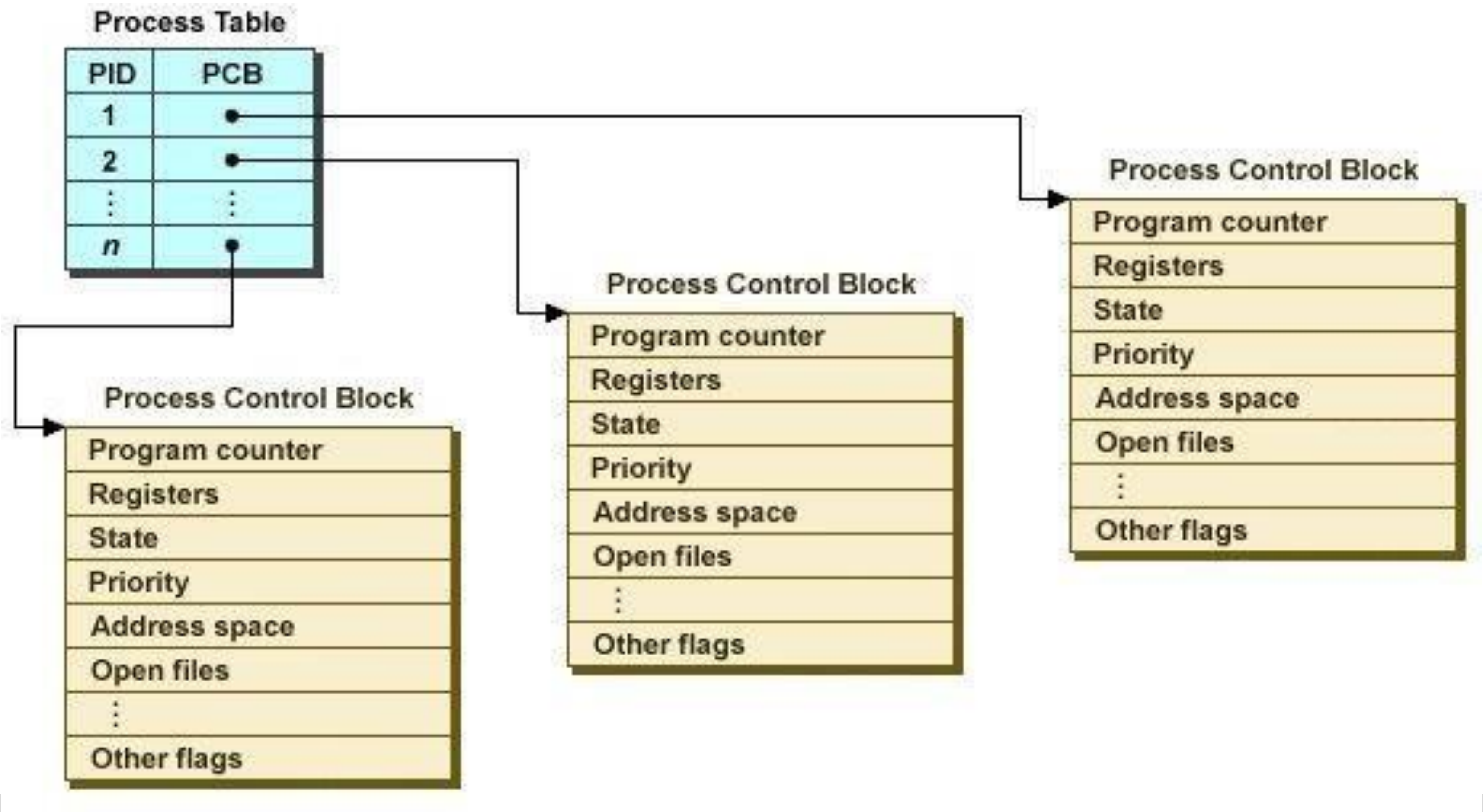
- this field is used to form a list of PCBs. The kernel maintains several lists of PCBs.

Event information

- for a process in the *blocked state*, this field contains information about *event for which the process is waiting*, when an event occurs kernel uses this information.

Process Table

The operating system maintains pointers to each process’s PCB in a process table so that it can access the PCB quickly.



Process Table

PID	PCB
1	•
2	•
⋮	⋮
n	•

Process Control Block

- Program counter
- Registers
- State
- Priority
- Address space
- Open files
- ⋮
- Other flags

Process Control Block

- Program counter
- Registers
- State
- Priority
- Address space
- Open files
- ⋮
- Other flags

Process Control Block

- Program counter
- Registers
- State
- Priority
- Address space
- Open files
- ⋮
- Other flags

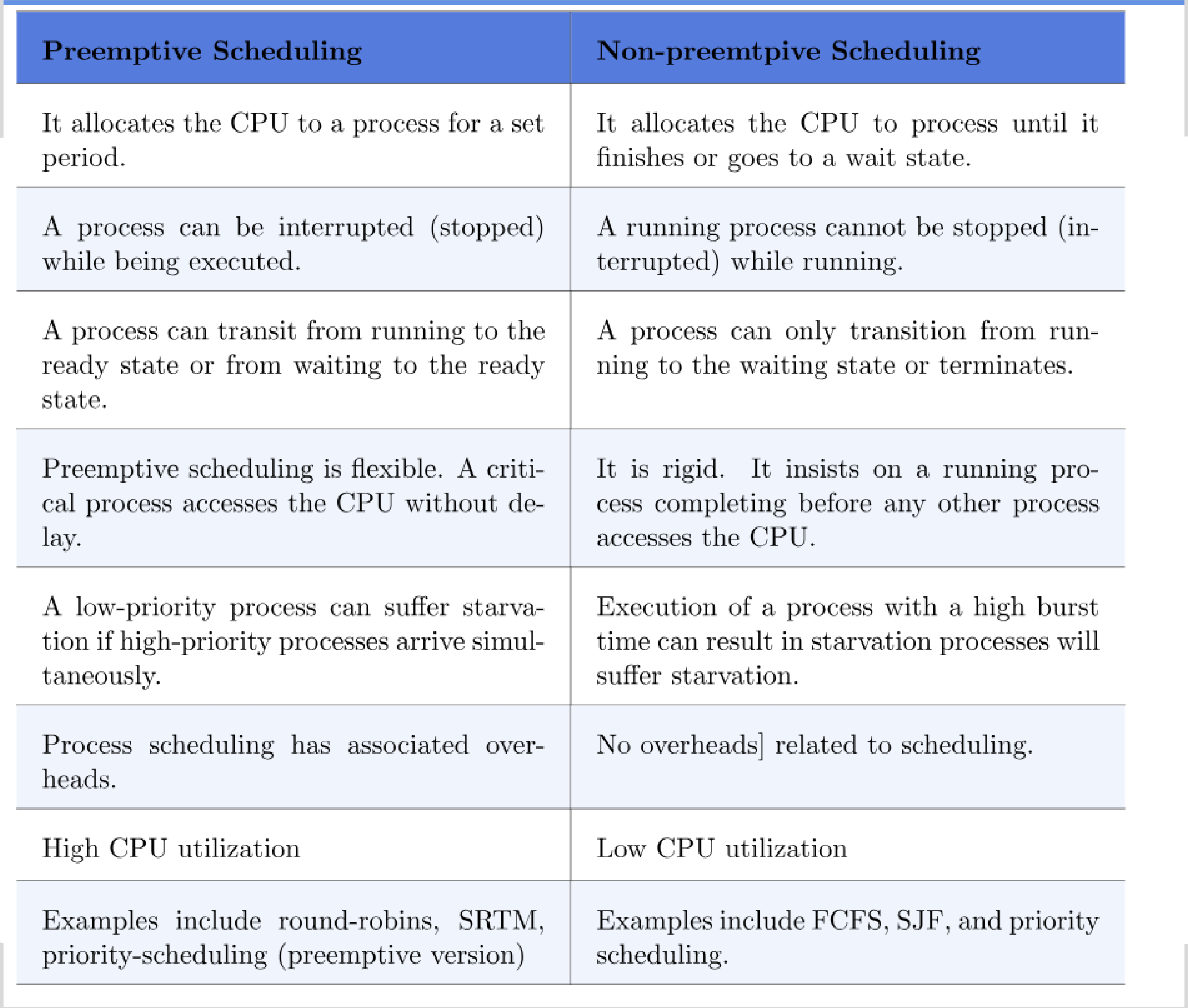
Process Scheduling

- The act of determining which process is in the ready state, and should be moved to the running state is known as Process Scheduling.
- The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs.
- For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

Process Scheduling

Scheduling fell into one of the two general categories:

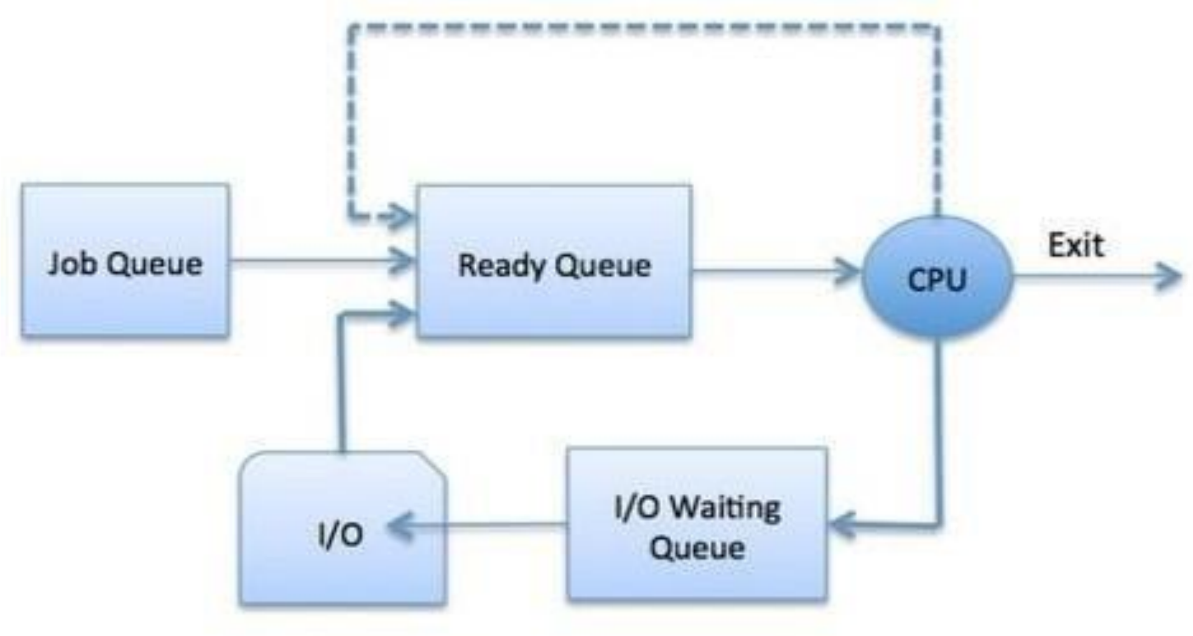
- **Non Pre-emptive Scheduling:** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive Scheduling:** When the operating system decides to favour another process, pre-empting the currently executing process.



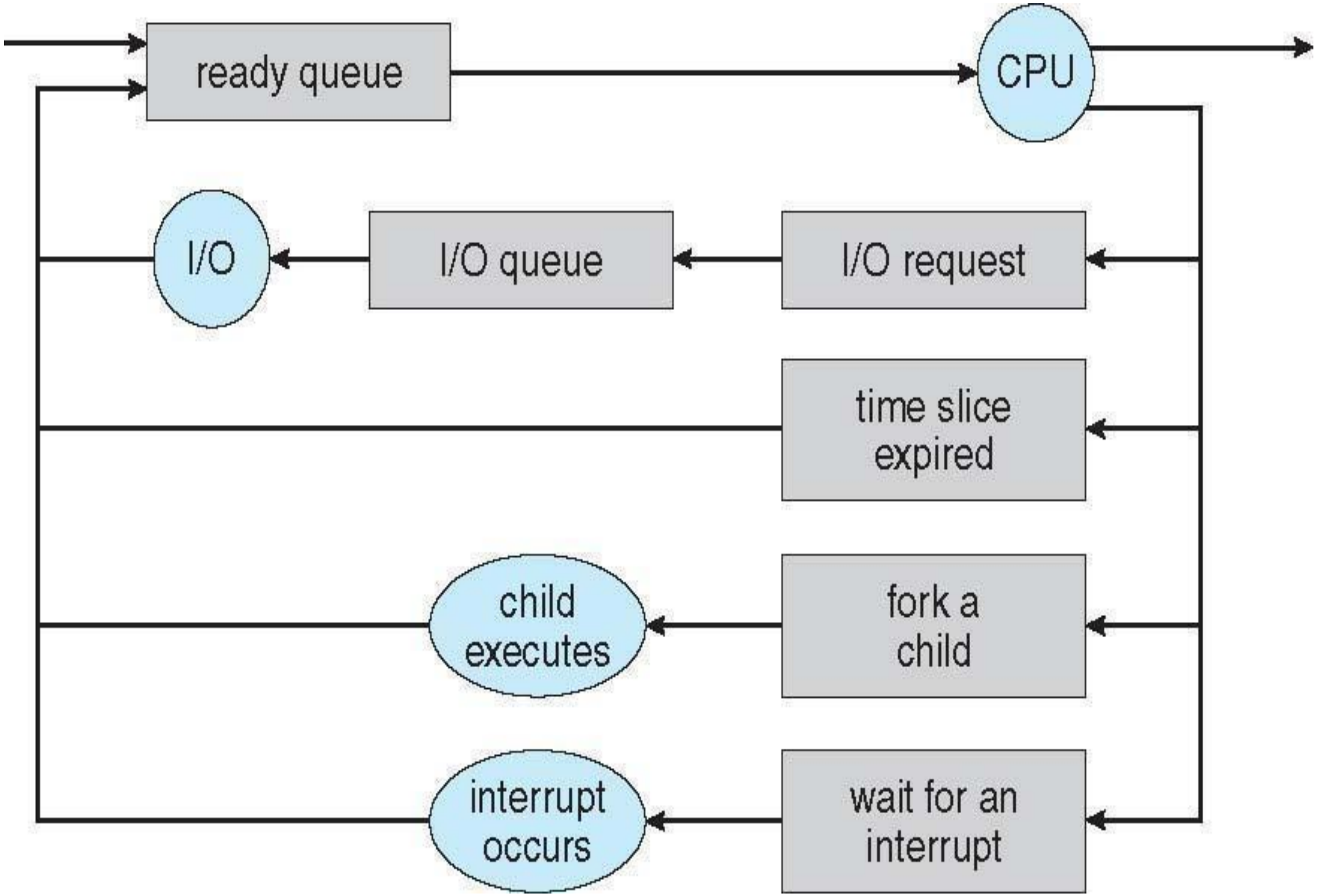
Preemptive Scheduling	Non-preemtpive Scheduling
It allocates the CPU to a process for a set period.	It allocates the CPU to process until it finishes or goes to a wait state.
A process can be interrupted (stopped) while being executed.	A running process cannot be stopped (interrupted) while running.
A process can transit from running to the ready state or from waiting to the ready state.	A process can only transition from running to the waiting state or terminates.
Preemptive scheduling is flexible. A critical process accesses the CPU without delay.	It is rigid. It insists on a running process completing before any other process accesses the CPU.
A low-priority process can suffer starvation if high-priority processes arrive simultaneously.	Execution of a process with a high burst time can result in starvation processes will suffer starvation.
Process scheduling has associated overheads.	No overheads] related to scheduling.
High CPU utilization	Low CPU utilization
Examples include round-robins, SRTM, priority-scheduling (preemptive version)	Examples include FCFS, SJF, and priority scheduling.

Scheduling Queues / Process Queues

- Maintains scheduling queues of processes
  - Job queue
  - Ready queue
  - Device queues - I/O Queue
- Processes migrate among the various queues



Scheduling Queues / Process Queues



Queuing Diagram

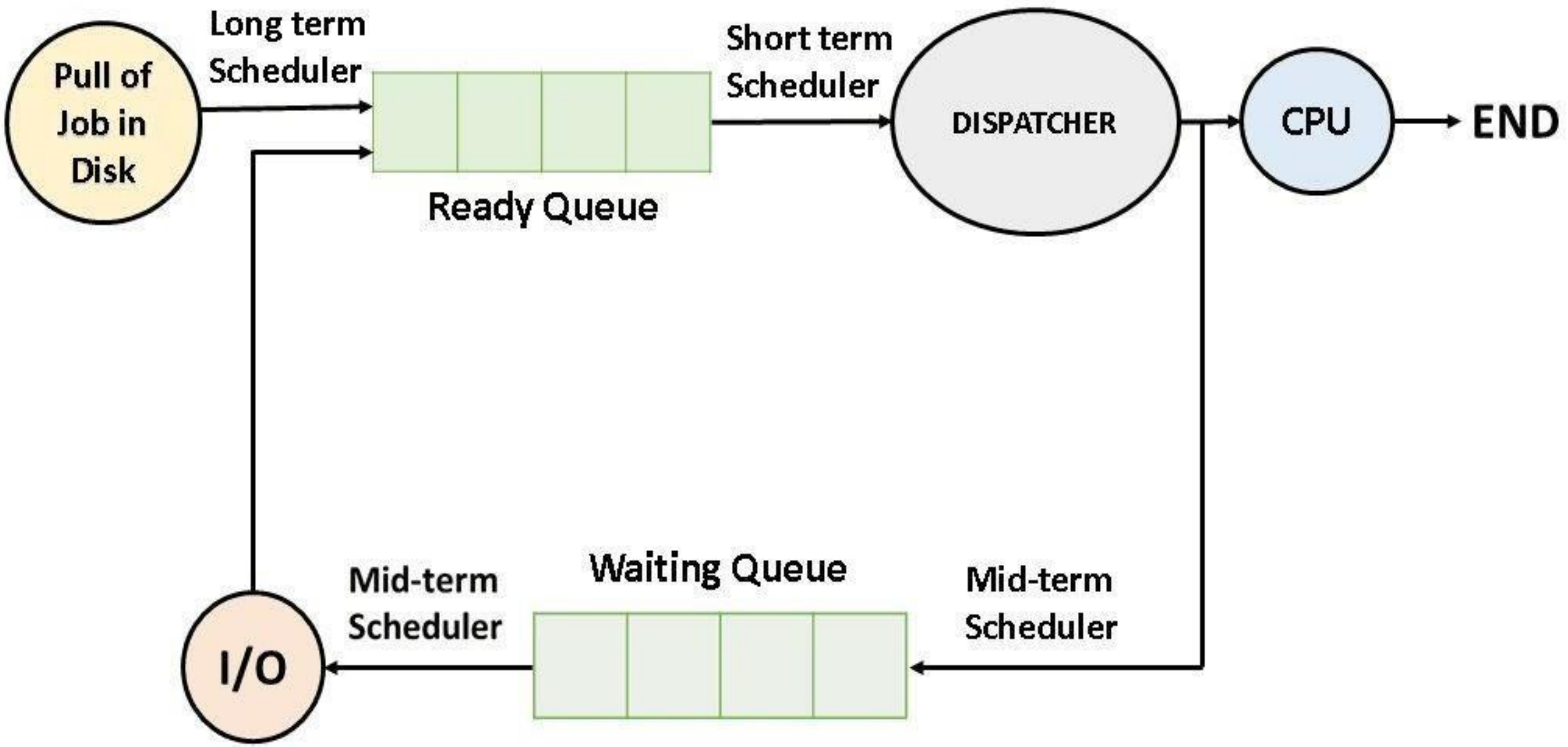
Scheduling Queues / Process Queues

- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.
- Processes waiting for a device to become available are placed in **Device Queues.** There are unique device queues available for each I/O device.

Process Scheduler

- Schedulers are special system software which handle process scheduling in various ways.
- Their main task is to select the jobs to be submitted into the system and to decide which process to run.
- Schedulers are of three types
    - Long-Term Scheduler / job scheduler
    - Short-Term Scheduler / CPU scheduler
    - Medium-Term Scheduler

Process Scheduler



Long Term Scheduler

- Due to the smaller size of main memory initially all the programs are stored in secondary memory. When they are loaded in main memory they are called process. This decision is made by Long Term Scheduler.
- Long Term Scheduler selects processes from the pool (or the secondary memory) and then maintains them in the primary memory's ready queue.
- It is also known as **Job Scheduler.**
- Long term scheduler runs less frequently.
- Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming.

Short Term Scheduler

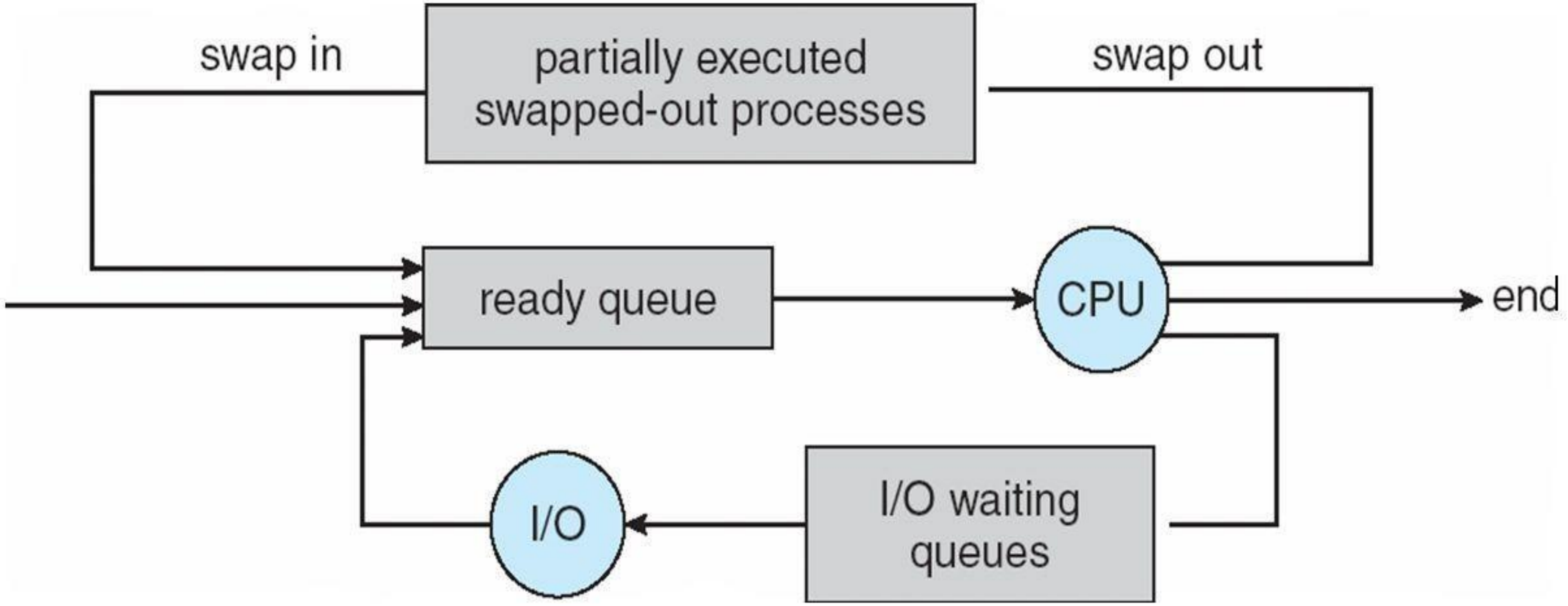
- There are many processes in the main memory initially all are present in the ready queue. Among all the processes a single process is to be selected for execution. This decision is handled by short term scheduler.
- It chooses one job from the ready queue and then sends it to the CPU for processing.
- This is also known as **CPU Scheduler**.
- Short term scheduler runs frequently.
- The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.

Medium Term Scheduler

- If the running state processes require some IO time to complete, the state must be changed from running to waiting. This is accomplished using a **Medium-Term scheduler**.
- The switched-out processes are handled by the Medium-Term scheduler.
- Swapped out processes are examples of this, and the operation is known as swapping.
- The Medium-Term scheduler here is in charge of stopping and starting processes.

Medium Term Scheduler

Swapping may be necessary to improve the process mix, or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

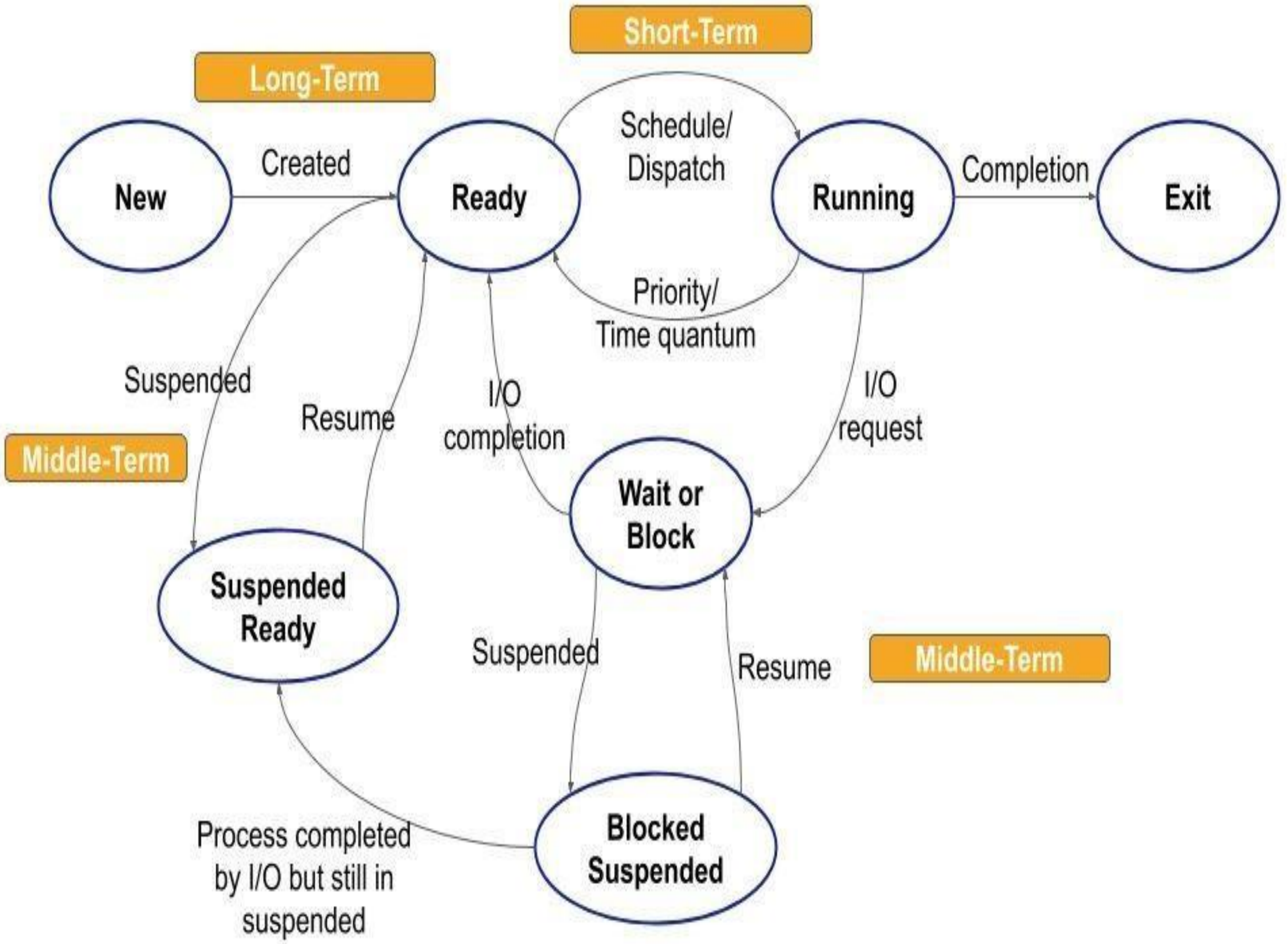


Addition of Medium-term scheduling to the queueing diagram.

Difference Scheduler

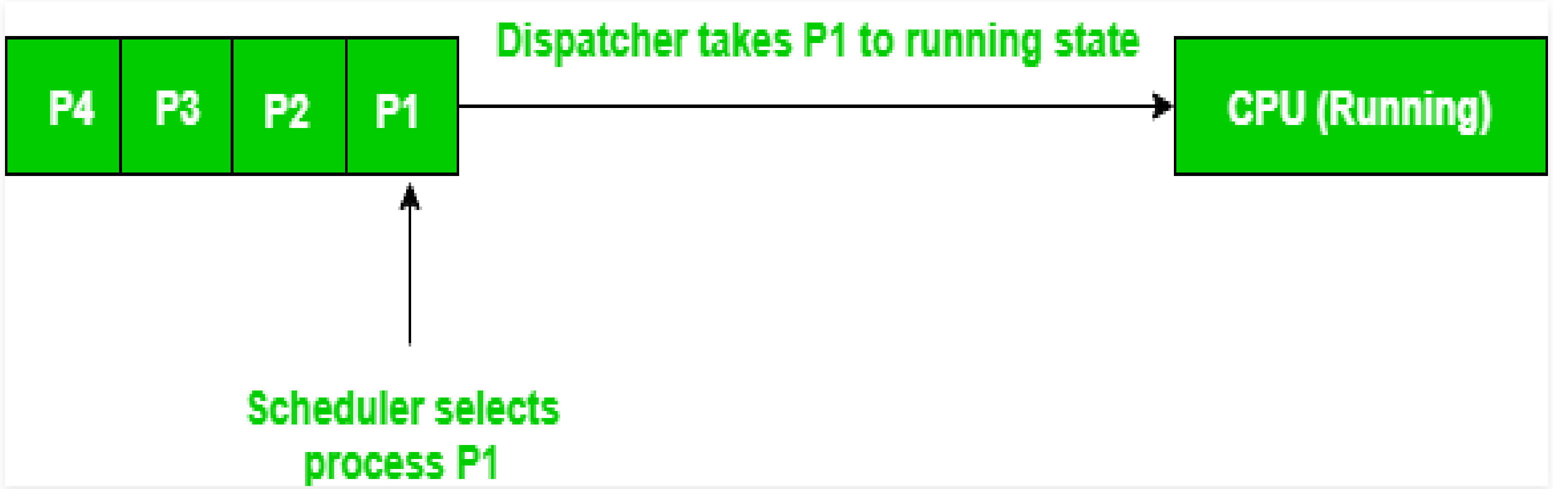
S.N	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Scheduler


Short-Term
Long-Term
New
Created
Ready
Schedule/ Dispatch
Running
Completion
Exit
Priority/ Time quantum
Suspended
Resume
I/O completion
I/O request
Middle-Term
Wait or Block
Suspended Ready
Suspended
Resume
Middle-Term
Process completed by I/O but still in suspended
Blocked Suspended

Dispatcher

- A dispatcher is a special program which comes into play after the scheduler.
- When the scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/queue.

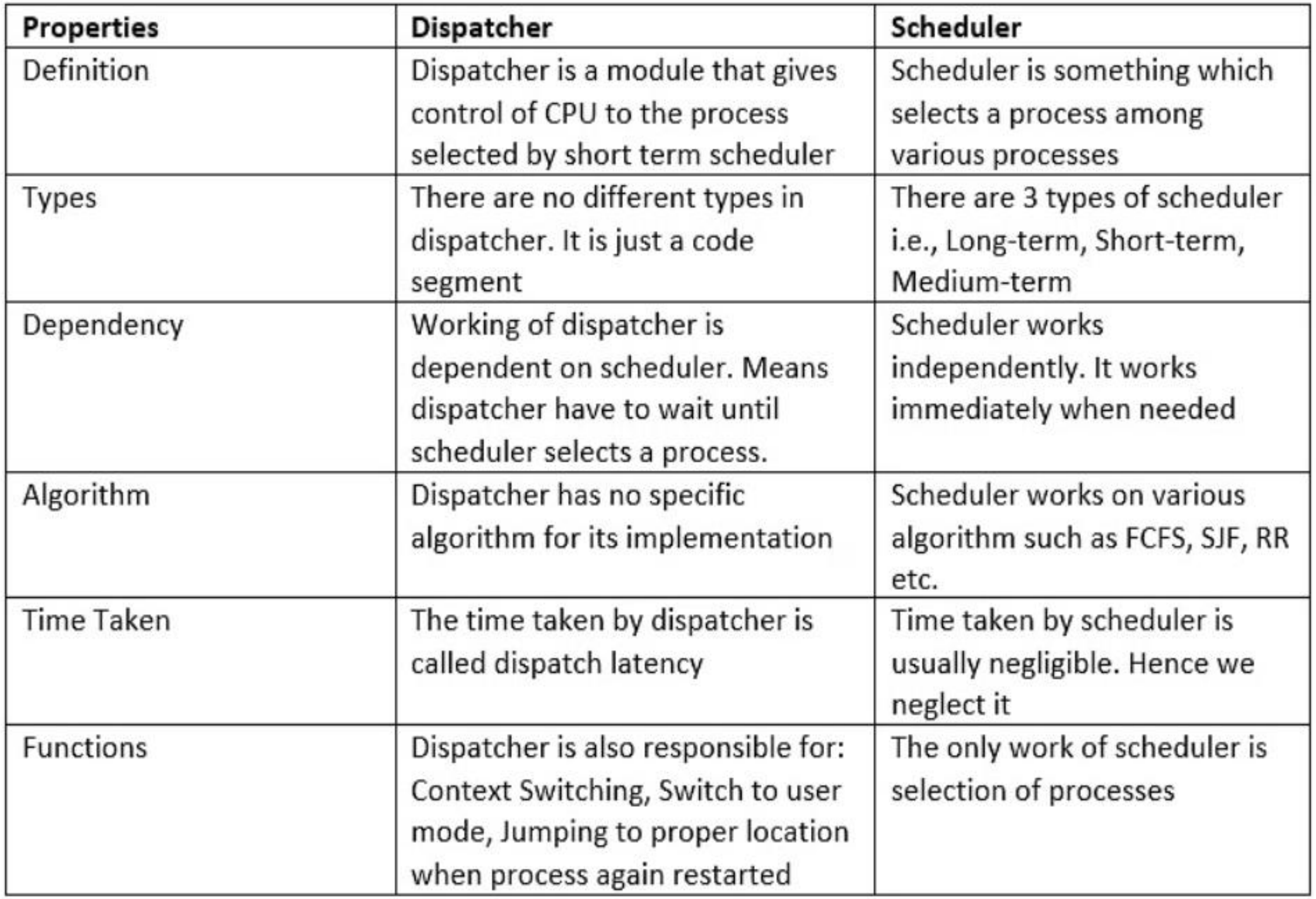

P4
P3
P2
P1
Dispatcher takes P1 to running state
CPU (Running)
Scheduler selects process P1

Dispatcher

The dispatcher is the module that gives a process control over the CPU after it has been selected by the short-term scheduler. This function involves the following:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

Dispatcher vs Scheduler



Properties	Dispatcher	Scheduler
Definition	Dispatcher is a module that gives control of CPU to the process selected by short term scheduler	Scheduler is something which selects a process among various processes
Types	There are no different types in dispatcher. It is just a code segment	There are 3 types of scheduler i.e., Long-term, Short-term, Medium-term
Dependency	Working of dispatcher is dependent on scheduler. Means dispatcher have to wait until scheduler selects a process.	Scheduler works independently. It works immediately when needed
Algorithm	Dispatcher has no specific algorithm for its implementation	Scheduler works on various algorithm such as FCFS, SJF, RR etc.
Time Taken	The time taken by dispatcher is called dispatch latency	Time taken by scheduler is usually negligible. Hence we neglect it
Functions	Dispatcher is also responsible for: Context Switching, Switch to user mode, Jumping to proper location when process again restarted	The only work of scheduler is selection of processes

Context Switching

- The Context switching is a technique or method used by the operating system to switch a process from one state to another to execute its function using CPUs in the system.
- When switching is performed in the system, the old running process's status is stored as registers, and the CPU is assigned to a new process for the execution of its tasks.
- While new processes are running in a system, the previous ones must wait in the ready queue.
- In simple terms, it is like loading and unloading the process from running state to ready state.

Context Switching

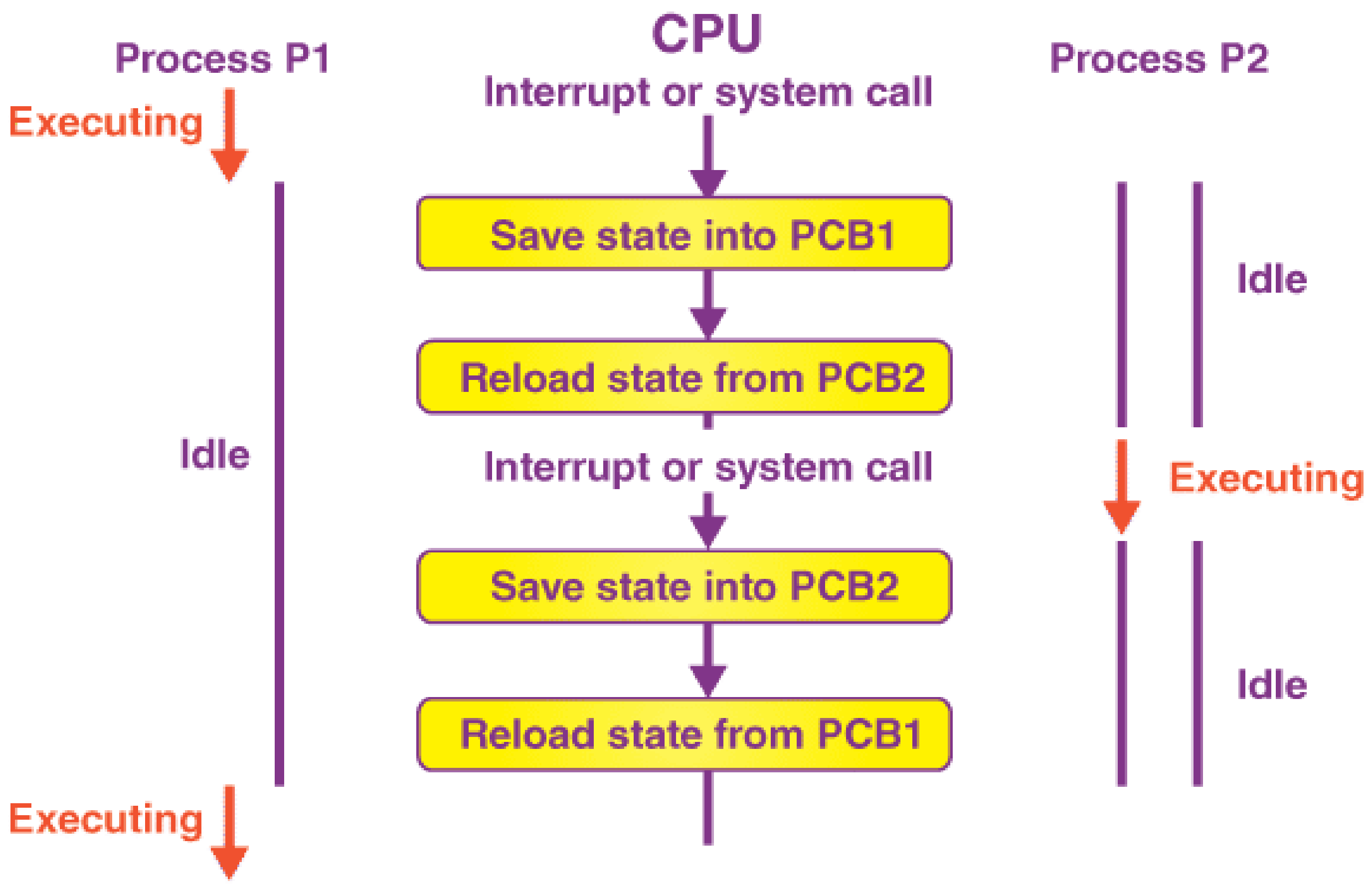
- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a **Context Switch**.
- When a context switch occurs, the Kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- It defines the characteristics of a multitasking operating system in which multiple processes shared the same CPU to perform multiple tasks without the need for additional processors in the system.

Context Switching

When does context switching happen?

- When a high-priority process comes to ready state (i.e. with higher priority than the running process)
- An Interrupt occurs
- User and kernel mode switch (It is not necessary though)
- Pre-emptive CPU scheduling used.

Context Switching



CPU Switch From Process to Process

Context Switching

Working

- The process P1 is initially running on the CPU for the execution of its task. At the very same time, P2, another process, is in its ready state. If an interruption or error has occurred or if the process needs I/O, the P1 process would switch the state from running to waiting.
- Before the change of the state of the P1 process, context switching helps in saving the context of the P1 process as registers along with the program counter (to PCB1). Then it loads the P2 process state from its ready state (of PCB2) to its running state.

Context Switching

- **Interrupts:** A CPU requests for the data to read from a disk, and if there are any interrupts, the context switching automatic switches a part of the hardware that requires less time to handle the interrupts.

- **Multitasking:** A context switching is the characteristic of multitasking that allows the process to be switched from the CPU so that another process can be run. When switching the process, the old state is saved to resume the process's execution at the same point in the system.

- **Kernel/User Switch:** It is used in the operating systems when switching between the user mode, and the kernel/user mode is performed.

THANK YOU