



Process Concepts

BY:

ROSHAN R. ROHIT

LECTURER,

IT DEPT,

GPG,SURAT

What is a program?

- ❑ A program is an organized collection of instruction.
- ❑ A program is a static/passive entity. Once user writes a program, it will remain as it is until user modifies it.
- ❑ A program has a longer life span. Once a program is written in some file and stored on disk, it remains there till file is not deleted.
- ❑ A program requires memory space on disk to store all the instruction only.

What is a process?

- ❑ A process is a program in execution . It is a unit of work within the system.
- ❑ Program is a passive entity; Process is an active entity.
- ❑ Process needs resources to accomplish its task
 1. CPU, Memory, I/O, File
 2. Initialization of data
- ❑ Process termination requires release of any reusable resources

What is a process?

- ❑ Single-threaded process has one program counter specifying location of next instruction to execute
- ❑ The process executes instruction sequentially, one at a time, until completion.

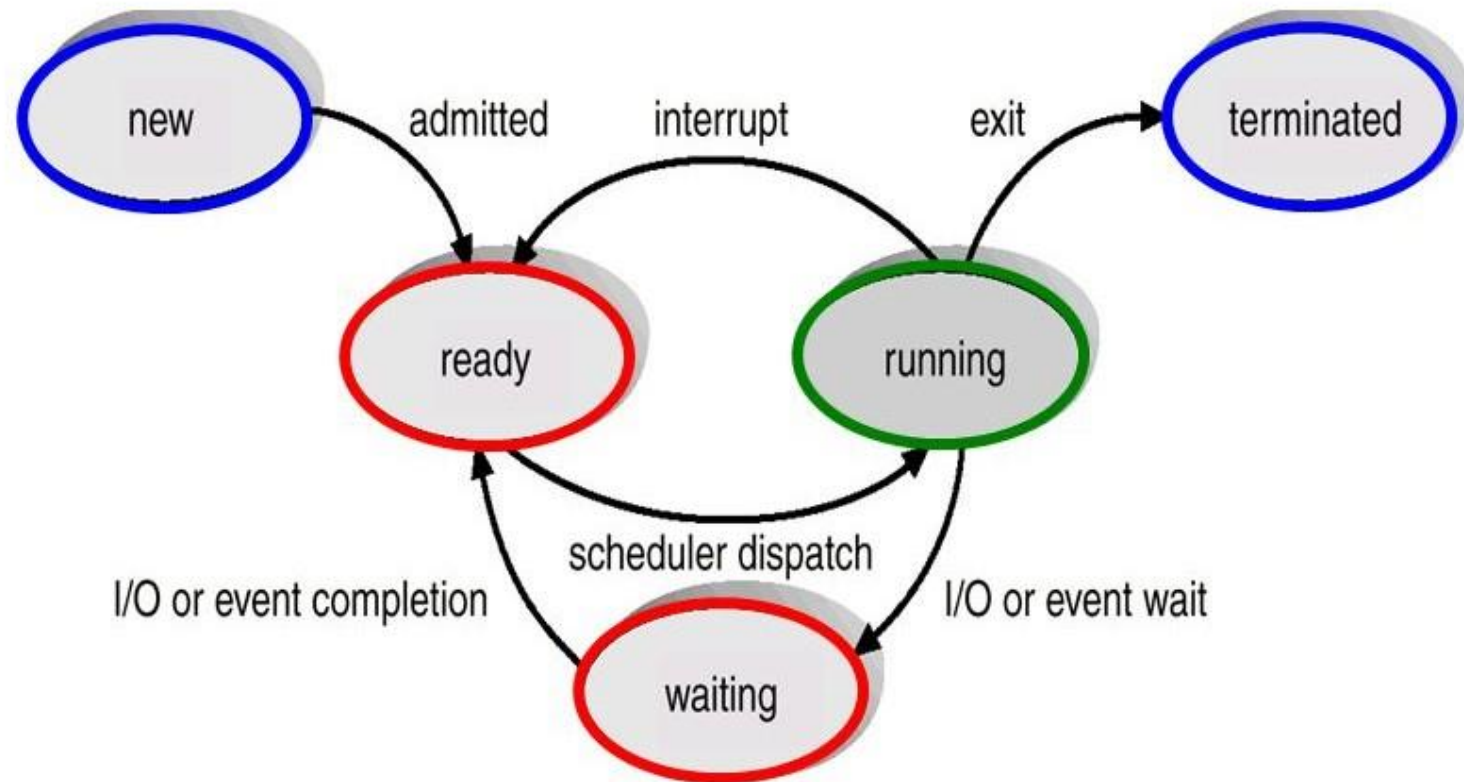
What is a process?

- ❑ Multi-threaded process has one program counter per thread
- ❑ A Process has limited life span.
- ❑ A process contains various resources like as memory space, disk, printers, scanners, etc.
- ❑ A process contains memory space which is called its address space.

Process life cycle

- ❑ A process is a dynamic/active entity.
- ❑ It changes its states during time duration of its execution.
- ❑ A process state indicates current activity of a process.

Process Life Cycle



Process States

- ❑ **New** : The process is being created.
- ❑ **Ready** : The process is waiting to be assigned to a processor
- ❑ **Running** : Instruction are being executed.
- ❑ **Waiting** : The process is waiting for some I/O completion or some event to occur.
- ❑ **Terminated** : The process has finished execution.

Process Control Block



Operating System is responsible for various activities related to process management, such as process creation, execution, termination, etc.



As modern OS are multi-tasking, there may be more than one process in execution at a time.



OS maintains a table, called Process Table, to store all the information about each process.



Process table contains various entries – one entry per process, called Process Control Block(PCB) or Task Control Block(TCB),

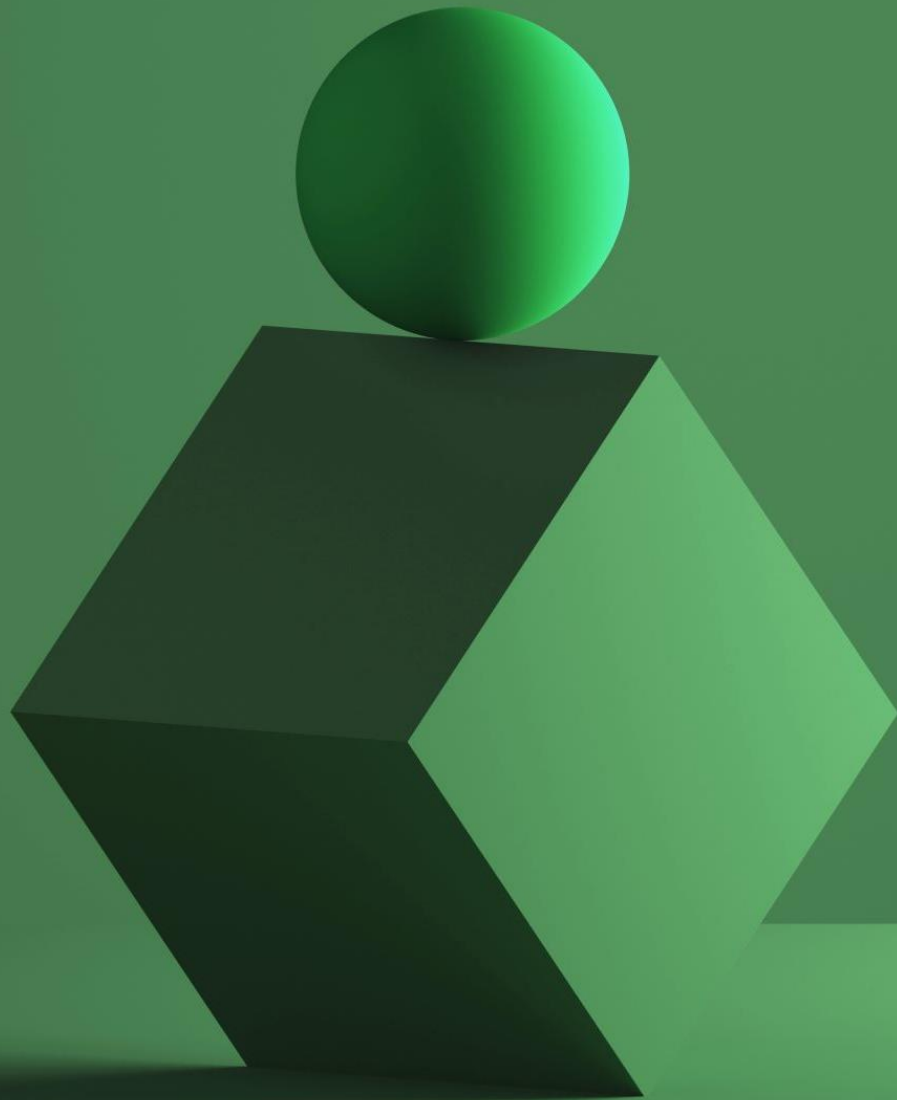
Process Control Block

- ❑ **Process Identifiers** - Process identifier(PID), parent process identifier(PPID) & User Identifier(UID)
- ❑ **Process state** – running, waiting, ready etc.
- ❑ **Program counter** – location of instruction to be executed next.
- ❑ **CPU registers** – various CPU registers are used during execution of a process such as accumulators, index registers, stack pointer, general purpose registers, etc.
- ❑ **CPU scheduling information** –Includes process priority, pointers to various scheduling queues, information about events on which process is waiting & other scheduling parameters

Process Control Block

- ❑ **Memory-management information** – It includes values of the base and limit registers, information about page table.
- ❑ **Accounting Information** – it includes the amount of CPU and real time used, time limits.
- ❑ **I/O status information** – I/O devices allocated to process, list of open files.
- ❑ **Pointers** – it contains information about where the code, data and stack regions are stored in main memory.





Process & Processor Scheduling

BY:

ROSHAN R. ROHIT

LECTURER,

IT DEPT,

GPG,SURAT

Basic Terminologies



Scheduling: It is the process of making a choice of which process to run next whenever more than one processes are simultaneously in the "Ready" state.



Scheduler: It is an OS module which makes a choice of which process to run next whenever more than one processes are simultaneously in the "Ready" state.



Scheduling Algorithm: It is an algorithm, which is used by the scheduler in making a choice of which process to run next whenever more than one processes are simultaneously in the "Ready" state.

Goals of Scheduling Algorithm



Optimize the utilization of system resources



Be fair to all pending jobs/processes



Ensure that more critical processes get priority over others

Process Behavior

Process execution consists of a cycle of CPU execution and I/O wait. It alternate between these two states.

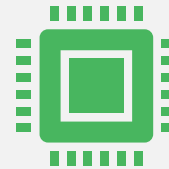
CPU burst is the time duration for which process executes continuously before waiting for an I/O operation.

I/O operation involves operations either related to reading an input from some device or a file or writing an output to some device or a file.

CPU bound process



It contains long CPU burst and thus infrequent I/O wait.

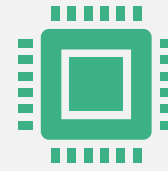


It performs lots of computation and does little I/O.



It keeps CPU busy most of the time.

I/O bound process



It contains short CPU burst and thus frequent I/O wait.



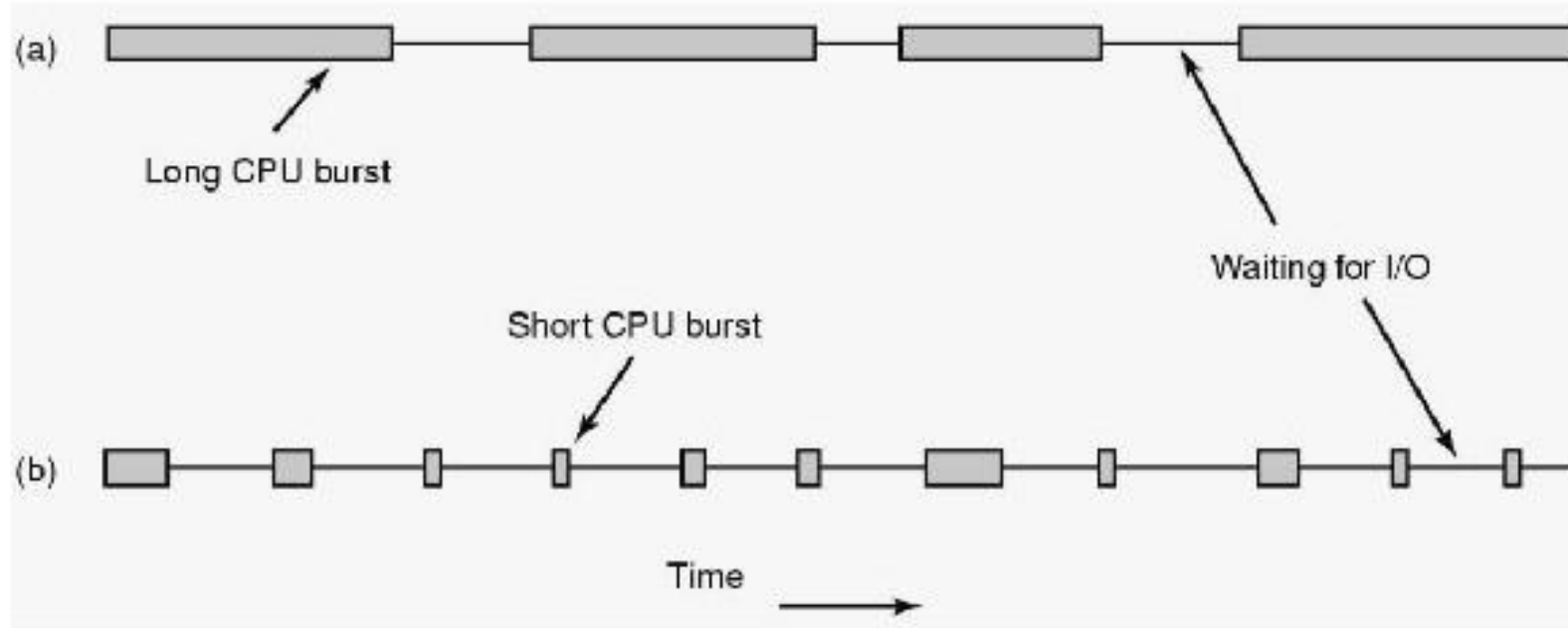
It performs little computation and does lots of I/O operations.



It keeps waiting for I/O operation to complete most of the time

CPU vs I/O bound process

CPU-bound and I/O-bound Processes



(a) A CPU-bound process

(b) An I/O-bound process

When to schedule?



When process switches from running state to waiting state.



When process switches from running state to terminated state.



When process switches from running state to ready state.



When process switches from ready state to running state.

Non-Preemptive Scheduling

One process selected to execute is allowed to run until it voluntarily enters wait state or gets terminated.

It is simple but it's not efficient.

Doesn't require timer or clock interrupt.

Preemptive Scheduling

One process selected to execute is allowed to run only for some maximum time duration.

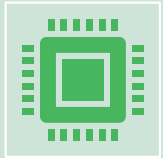
It is very efficient but complex.

Requires timer and clock interrupt

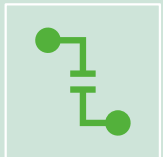
Scheduling Queues



Job queue: It consists all the processes in the system. As processes enter the system, they are put into this queue.

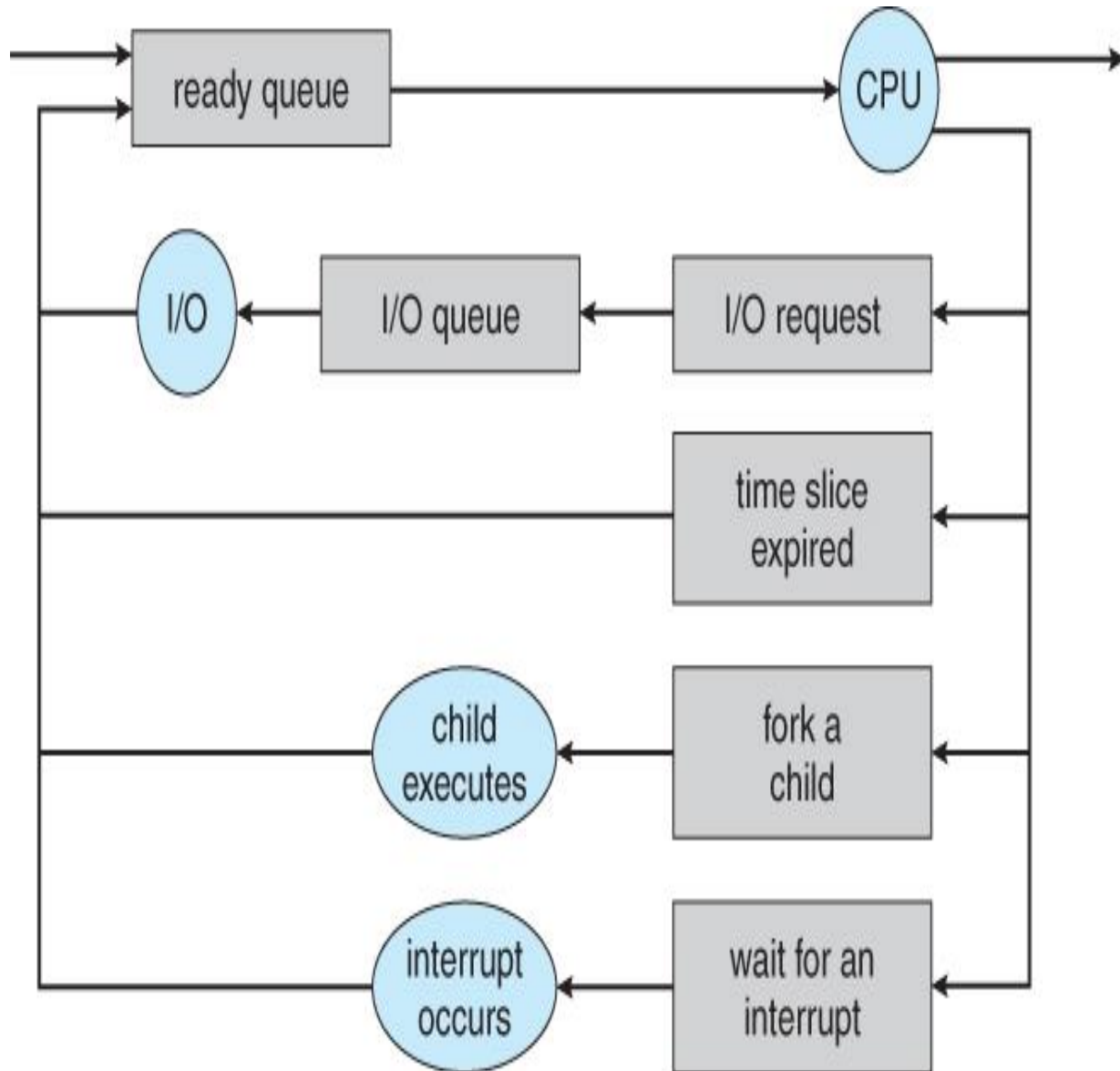


Ready queue: It consists of processes which are residing in main memory, ready to run and waiting for CPU.



Device queue: It consists of processes which are waiting for a particular device. Each device has its own queue.

Queueing Diagram

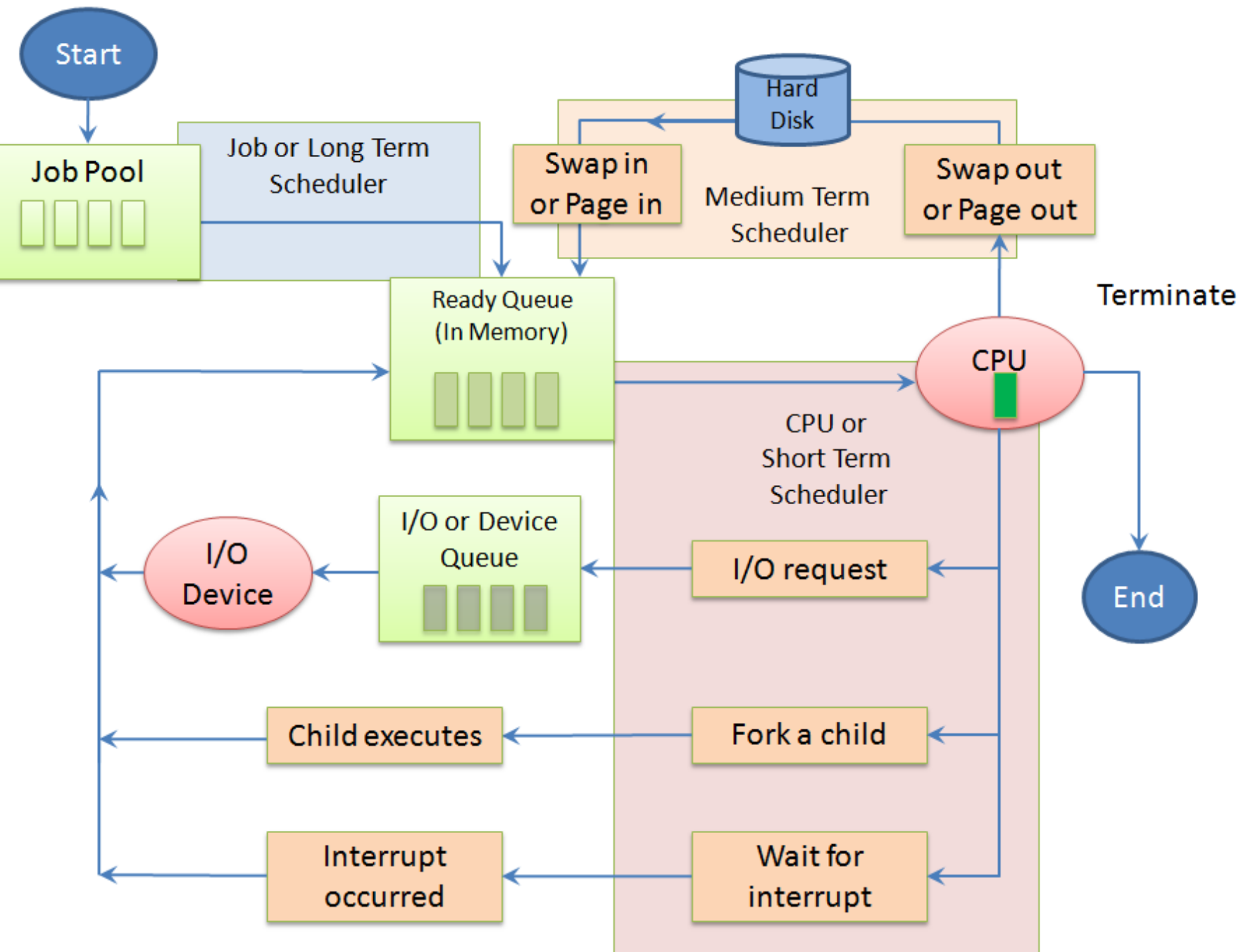


- ❖ A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution.
- ❖ Once the process is assigned CPU and is executing, one of the several events could occur:
- ❖ Process can issue an I/O request and then be placed in an I/O queue.
- ❖ Process can create a new sub-process and wait for its termination.
- ❖ Process can be removed forcibly from the CPU and put back in 'Ready' queue
- ❖ A process continues such cycle until it terminates



Types of Scheduler

- ❖ There are three different types of schedulers depending upon the scheduling queue and operating system.
- ❖ Long-term Scheduler
- ❖ Medium-term Scheduler
- ❖ Short-term Scheduler



Schedulers on Queuing Diagram

- ❖ The figure describes the positions of the schedulers in the queuing diagram.

Long-term Scheduler

It is a first level scheduler, found in OS where there is support for 'batch', like batch operating systems.

It works like a 'batch queue' where it selects the next job/process to be executed. The process is then loaded in main memory and waits in 'Ready' state for CPU to become free.

It executes much less frequently compared to other schedulers.

It controls the degree of multi-programming.

Medium-term Scheduler



It is a second level scheduler, found in OS where there is support of 'Swapping'.



It swaps-in and swaps-out processes between main memory and disk.



It executes when memory is filled up by processes which are in 'Waiting' state or when memory space is vacated by a terminating process.



It controls the degree of multi-programming.

Short-term Scheduler

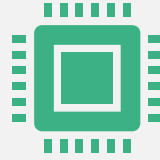
It is a third level scheduler, found in all modern Operating systems.

It works with the 'ready' queue. It selects the next process to be executed by CPU whenever more than one process is simultaneously in the 'Ready' state.

It controls the 'Ready-to-Running' state transitions in process life cycle.

It executes much frequently as compared to other schedulers and hence it must be fast to keep scheduling overhead low.

Objectives of Scheduler



Long-term Scheduler: To provide balanced mix of jobs, such as CPU-bound and I/O bound, to the short-term scheduler.



Medium-term Scheduler: To make efficient use of main memory.



Short-term Scheduler: To keep CPU always busy to get better utilization.

Context Switch

Suppose there are two processes, process-0 & process-1, executing simultaneously.

Process-0 is in 'Running' state and Process-1 is in 'Ready' state.

Now Process-0 needs some I/O operation, it goes in 'Waiting' state.

As CPU becomes free, scheduler selects Process-1 and it goes into 'Running' state.

Meanwhile, Process-0 completes I/O and enters in 'Ready' state.

Now when Process-1 releases CPU, Process-0 will be selected by scheduler to execute on CPU.

At this time, it must continue its execution from the point at which it has suspended execution.

It is the task of the OS to make such type of execution possible.

Process Context



Process context is the information about a process which is stored in a Process control block(PCB).

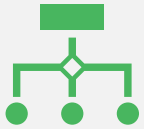


It includes process state, program counter, values of CPU registers, memory/file management etc.

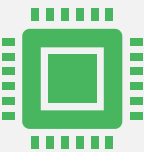


Process context is basically its current state (what is in its registers).

What is Context Switch??



The process of saving the context of one process and loading the context of another process is known as **Context Switching**.



It is an overhead to the system, as there is no useful work done during context switch, hence it should take minimum possible time.

When does Context Switch Occur?



When running process requests for some I/O operation.



When running process requests wait for some event.



When running process terminates.



When running process is to be preempted, i.e. when the process has run for its maximum allowed time duration or when some higher priority process enters in 'ready' state.

Dispatcher

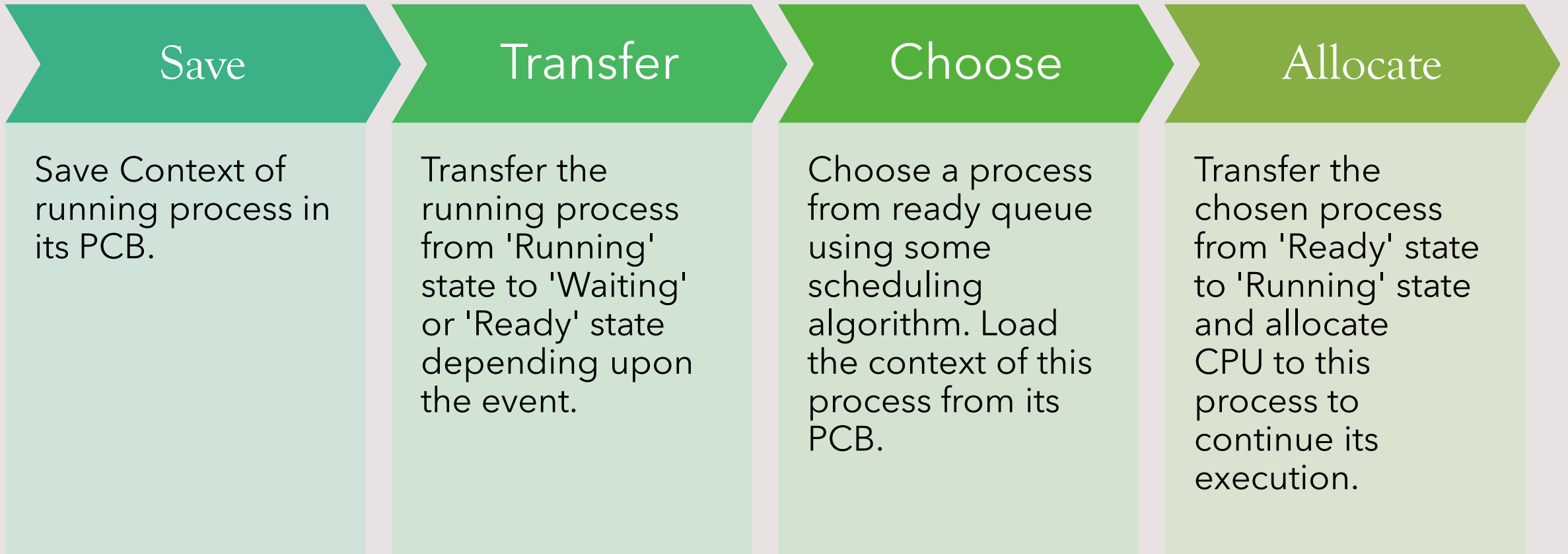
It is an OS module which transfers control of CPU from one process to another.

It is last step in scheduling.

It stores the context of current running process in its PCB and loads context of another process selected by the scheduler.

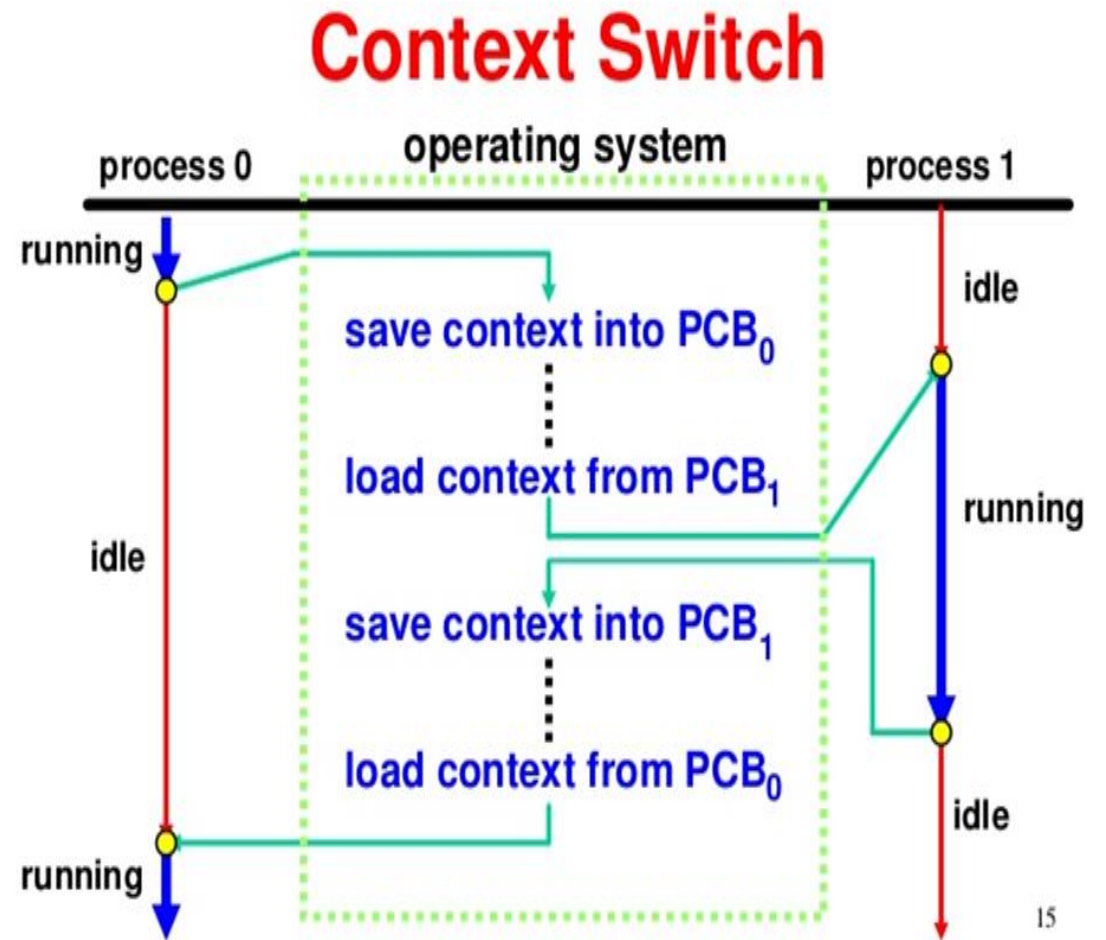
Dispatcher makes it possible for a process to continue its execution from the point where it has left previously.

Steps in process of Context Switch



Context Switch (example)

- The figure explains the context switch between two processes, process-0 and process-1.



ANY
QUESTIONS?



Process Scheduling Algorithms

BY:

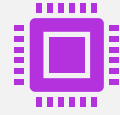
ROSHAN R. ROHIT

LECTURER,

IT DEPT,

GPG,SURAT

Performance Criteria of Scheduling Algorithms



CPU Utilization:



It is an average fraction of time during which CPU is busy.



It ranges from 0 to 100 percent.



CPU should remain as busy as possible

Performance Criteria of Scheduling Algorithms



Throughput



Number of processes completed per time unit is called throughput.



For long processes, this rate may be 1 process per hour; for short transactions, it might be 10 processes per second.

Performance Criteria of Scheduling Algorithms

Turn-Around Time

Time required to complete execution of a process is called turn-around time.

It specifies that how long it takes to complete a process execution.

Turn-around time = Process finish time - Process arrival time

Performance Criteria of Scheduling Algorithms

Waiting Time

It is the total time duration spent by a process waiting in 'Ready' queue.

Waiting time = Turn-around time – Actual execution time

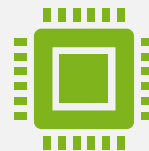
Performance Criteria of Scheduling Algorithms



Response Time



It is a time between issuing a command/request and getting output/result.



It is more important in interactive systems. User requests should be served as quickly as possible.

Scheduling Algorithms

Scheduling algorithm is also known as *scheduling policy*.

There are various scheduling algorithms available which works based on different criteria to make a choice.

Scheduling algorithm can be either non-preemptive or preemptive.

FIRST COME, FIRST SERVED (FCFS)



Here, the process that arrives first is served first.



The process which stands first in a queue is selected first.



This strategy is easily implemented by FIFO queue(First In First Out).



It works in a non-preemptive manner.

Example of FCFS

- Gantt Chart is as shown below:



Process	Arrival time(T0)	Time required for completion(ΔT)
P0	0	15
P1	1	2
P2	3	9
P3	5	6

Example of FCFS

Turn-around time:
 $81/4=20.25$ ms

Waiting time:
 $49/4=12.25$ ms

Process	Arrival time(T_0)	Completion time (ΔT)	Finish time(T_1)	Turn-around Time($TAT=T_1-T_0$)	Waiting time ($TAT- \Delta T$)
P0	0	15	15	15	0
P1	1	2	17	16	14
P2	3	9	26	23	14
P3	5	6	32	27	21

Advantages of FCFS

Simple & Fair

No starvation

Easy to understand

Easy to implement

Disadvantages of FCFS

Not efficient

Convoy effect is possible.

CPU utilization may be too low.

SHORTEST JOB FIRST(SJF)

Here, the process that requires shortest time to complete the execution, is served first.

So, the process whose execution time is less is first in the queue.

It also works in non- preemptive manner.

Example of SJF

- Gantt Chart is as shown below:



Process	Arrival time(T_0)	Time required for completion(ΔT)
P0	0	15
P1	1	2
P2	3	9
P3	5	6

Example of SJF

Turn-around time:
 $78/4=19.5$ ms

Waiting time:
 $46/4=11.5$ ms

Process	Arrival time(T_0)	Completion time (ΔT)	Finish time(T_1)	Turn-around Time($TAT=T_1-T_0$)	Waiting time ($TAT- \Delta T$)
P0	0	15	15	15	0
P1	1	2	17	16	14
P2	3	9	32	29	20
P3	5	6	23	18	12

Advantages of SJF

Less waiting time

Good response for short processes.

Disadvantages of SJF

It is difficult to estimate time required to complete execution.

Starvation is possible for long process.

Long processes may wait for ever.

SHORTEST REMAINING TIME NEXT(SRTN)



Here, process whose remaining run time is shortest ,is served first.



Its decision mode is preemptive .



If the process having shortest running time occurs in the queue, CPU is forcibly taken away from previous process having comparatively large remaining time.



This is implemented by using FIFO queue.

Example of SRTN

- Gantt Chart is as shown below:



Process	Arrival time(T_0)	Time required for completion(ΔT)
P0	0	10
P1	1	6
P2	3	2
P3	5	4

Example of SRTN

Turn-around time:
 $78/4=19.5$ ms

Waiting time:
 $46/4=11.5$ ms

Process	Arrival time(T_0)	Completion time (ΔT)	Finish time(T_1)	Turn-around Time($TAT=T_1-T_0$)	Waiting time ($TAT- \Delta T$)
P0	0	10	22	22	12
P1	1	6	9	8	2
P2	3	2	5	2	0
P3	5	4	13	8	4

Advantages of SRTN

Less waiting time

Quite good response for short processes.

Disadvantages of SRTN

It is difficult to estimate remaining time necessary to complete execution.

Starvation is possible for long process.

Long processes may wait for ever.

Context switch overhead is there.

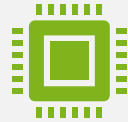
ROUND ROBIN SCHEDULING (RR)



This scheduling algorithm is quite similar to the FCFS.



Its decision mode is preemptive.



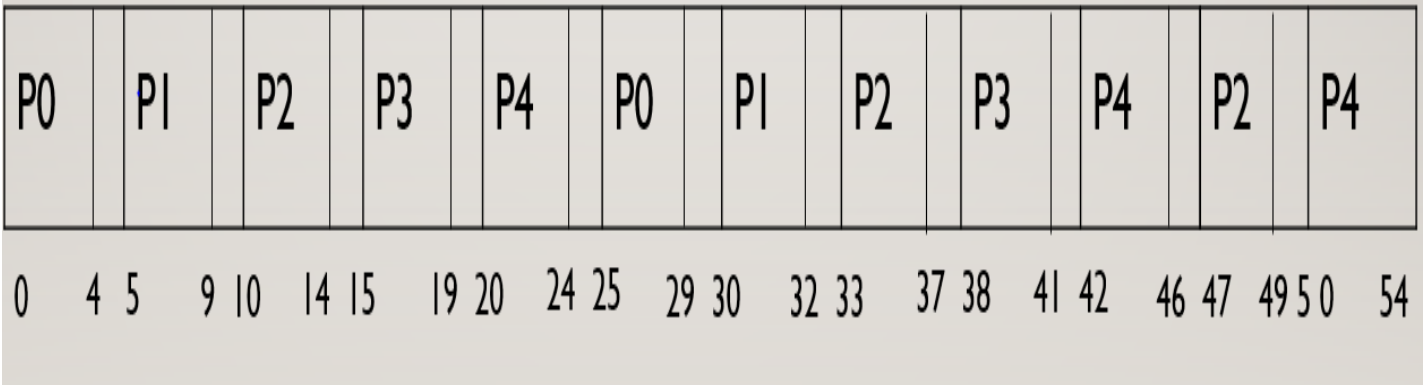
This algorithm is totally based on time quantum and context switch given.



This is implemented by circular queue.

Example of RR

- The time quantum is 4 ms and context switch overhead is 1 ms.
- Gantt Chart is as shown below:



Process	Arrival time(T_0)	Time required for completion(ΔT)
P0	0	8
P1	3	6
P2	5	10
P3	8	7
P4	11	12

Example of RR

Turn-around time:
 $178/5=35.6$ ms

Waiting time:
 $135/5=27$ ms

Process	Arrival time(T0)	Completion time (ΔT)	Finish time(T1)	Turn-around Time(TAT=T1-T0)	Waiting time (TAT- ΔT)
P0	0	8	29	29	21
P1	3	6	32	29	23
P2	5	10	49	44	34
P3	8	7	41	33	26
P4	11	12	54	43	31

Advantages of RR

One of the oldest and most widely used algorithms

Simplest and fairest algorithm

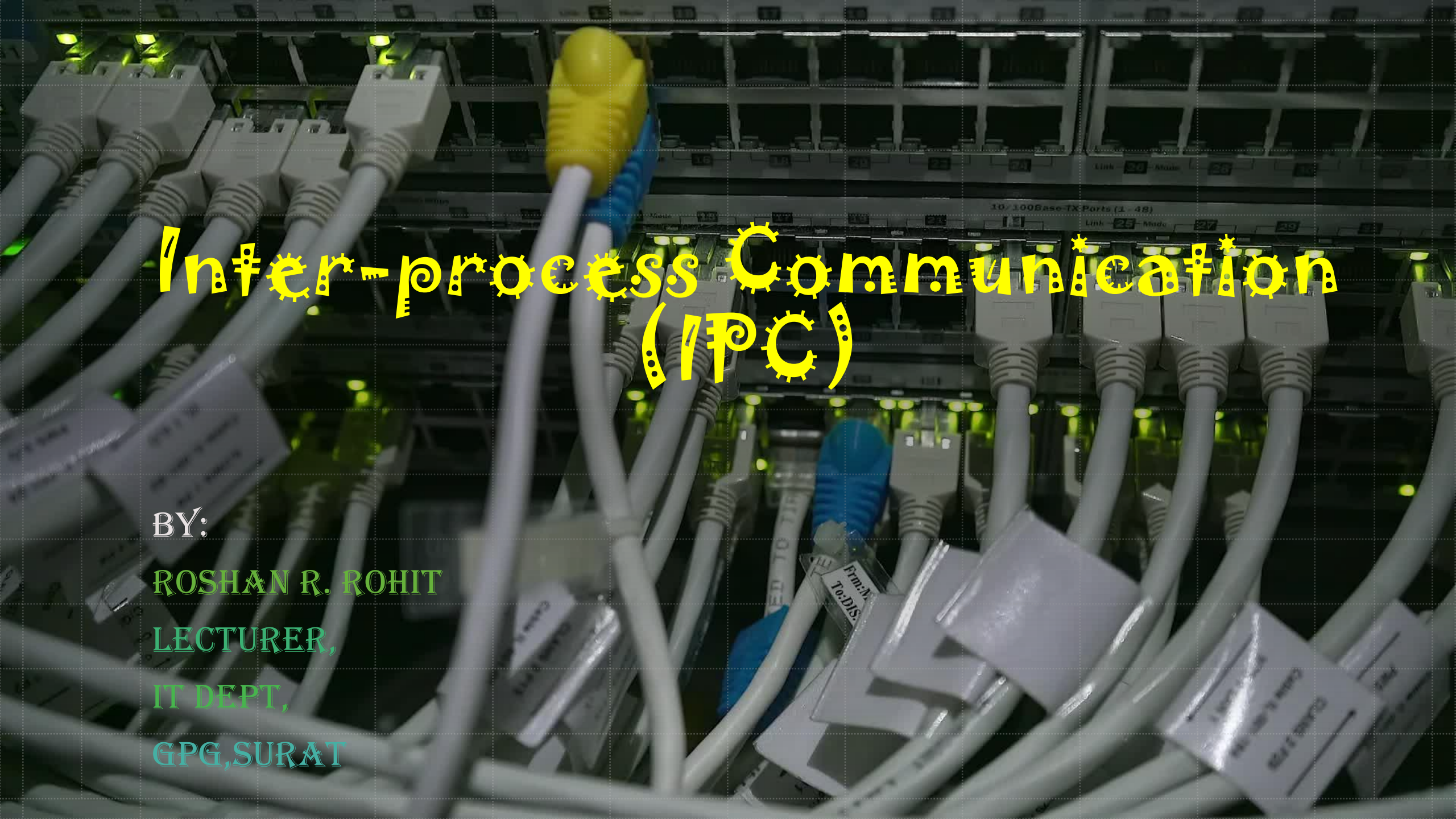
Disadvantages of RR

Context switch overhead is there.

Determination of time quantum is too critical.

ANY
QUESTIONS?



A close-up photograph of a network switch or patch panel. Numerous white Ethernet cables are plugged into the ports. Some cables have labels attached to them, such as 'From: M...' and 'To: DIS...'. A few ports have yellow and blue Ethernet connectors. The background is dark, and the overall lighting is dim, with some green indicator lights visible on the switch.

Inter-process Communication (IPC)

BY:

ROSHAN R. ROHIT

LECTURER,

IT DEPT,

GPG, SURAT

INTRODUCTION



Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.



Processes can communicate with each other through:



Sharing data.



Message passing

EXAMPLES OF IPC



Shell pipeline in Unix



Printing on printer in Network



Chat or Mail server

Issues related to IPC

How can one process pass information to another ?

Two or more processes should not get into each other's way

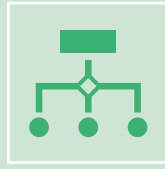
Proper sequencing should be maintained in execution of processes, when dependencies are present



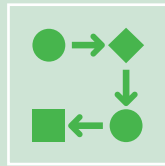
Independent v/s Co-operating processes

- The concurrent processes executing in the operating system may be either independent processes or co-operating processes.

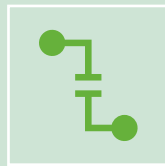
Independent Processes



This process does not communicate with any other process in the system.



Such process do not share any data with any other process.

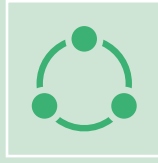


It cannot affect or be affected by other processes executing in the system.



It does not participate in inter process communication.

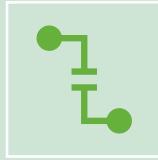
Co- operating processes



This process communicate with other processes in the system.



Such process shares data with other processes.



It can affect or be affected by the other processes executing in the system.



It participates in inter process communication

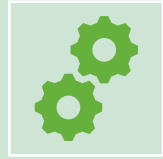
Advantages of Co- operating Processes



Information Sharing



Computation Speedup



Modularity



Convenience



Disadvantages of Co-operating Processes

Race condition

Process synchronization

Process Synchronization

It is a mechanism to ensure a systematic sharing of resources among concurrent processes.

Two or more processes should not come into each other's way.

Also, proper sequencing should be maintained while executing cooperating processes.



Race conditions / Racing Problems

- “Situations where two or more processes are reading or writing some shared data and the final result depends on the relative order of their execution, is called race condition or racing problem.”

Example of Racing Problem

- Suppose two processes p0 and p1 are accessing a common integer variable 'y' as follows. Suppose the initial value of y=5

- process p0:

- `read(y);`

- `y:=y+1;`

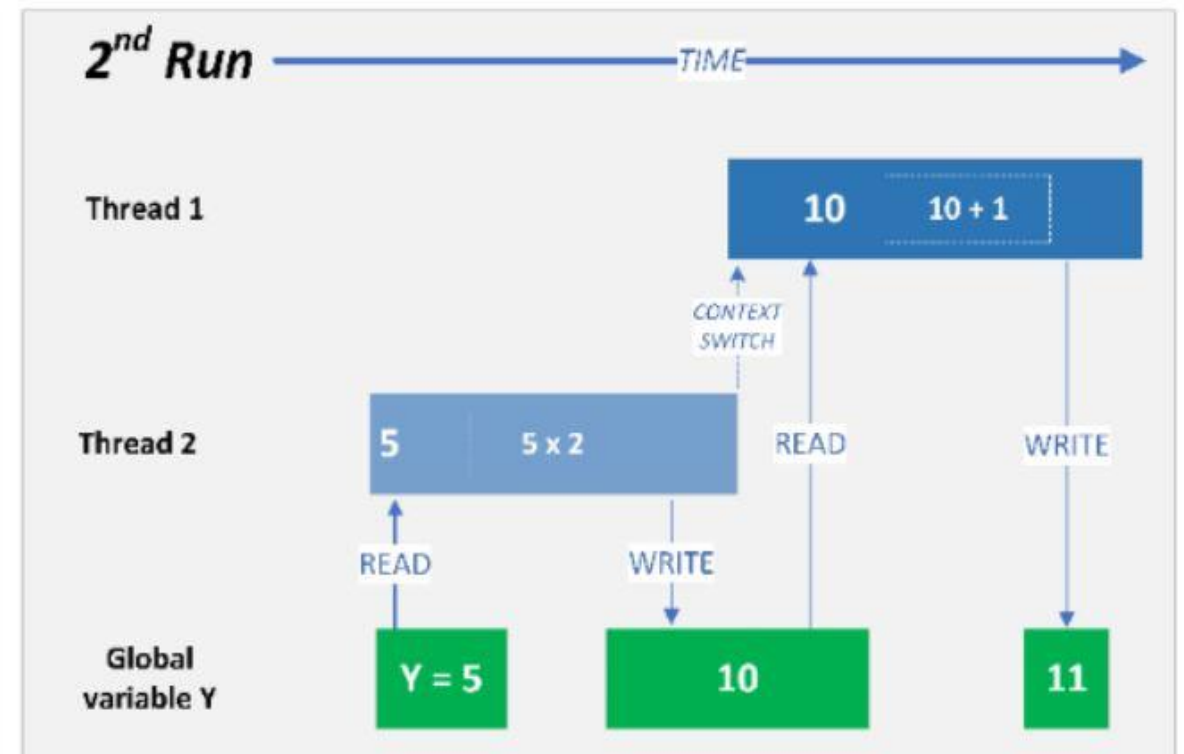
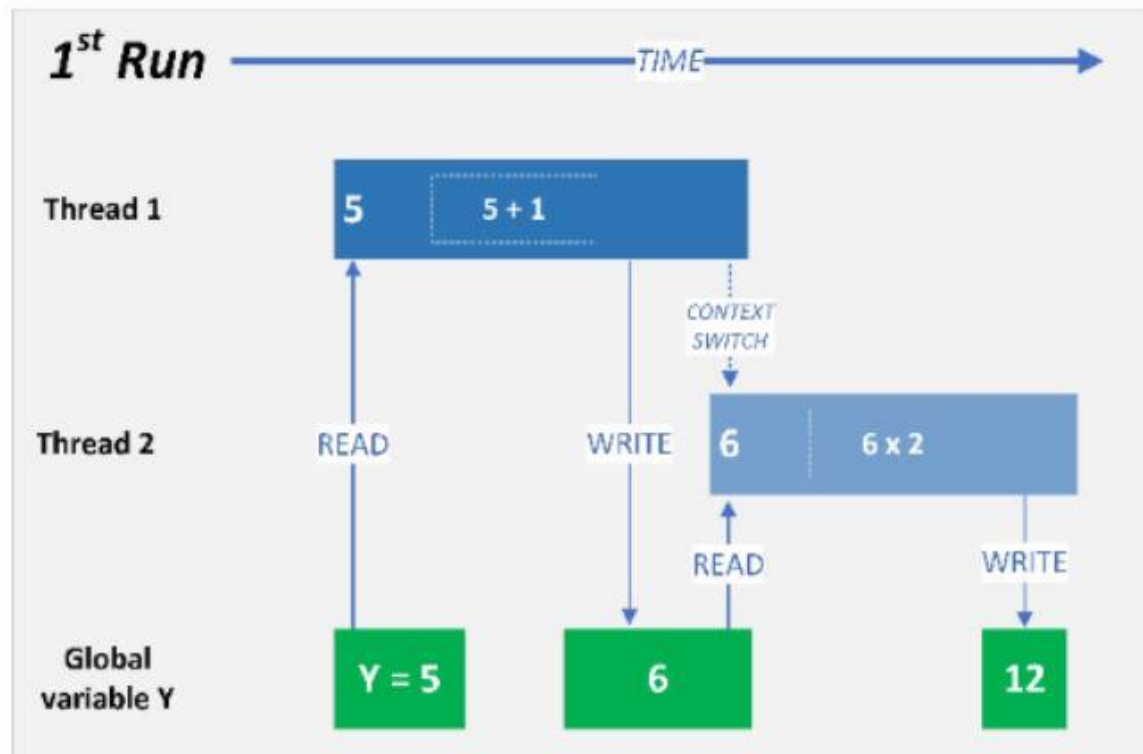
- `write (y);`

- process p1:

- `read(y);`

- `y:=y*2;`

- `write(y);`

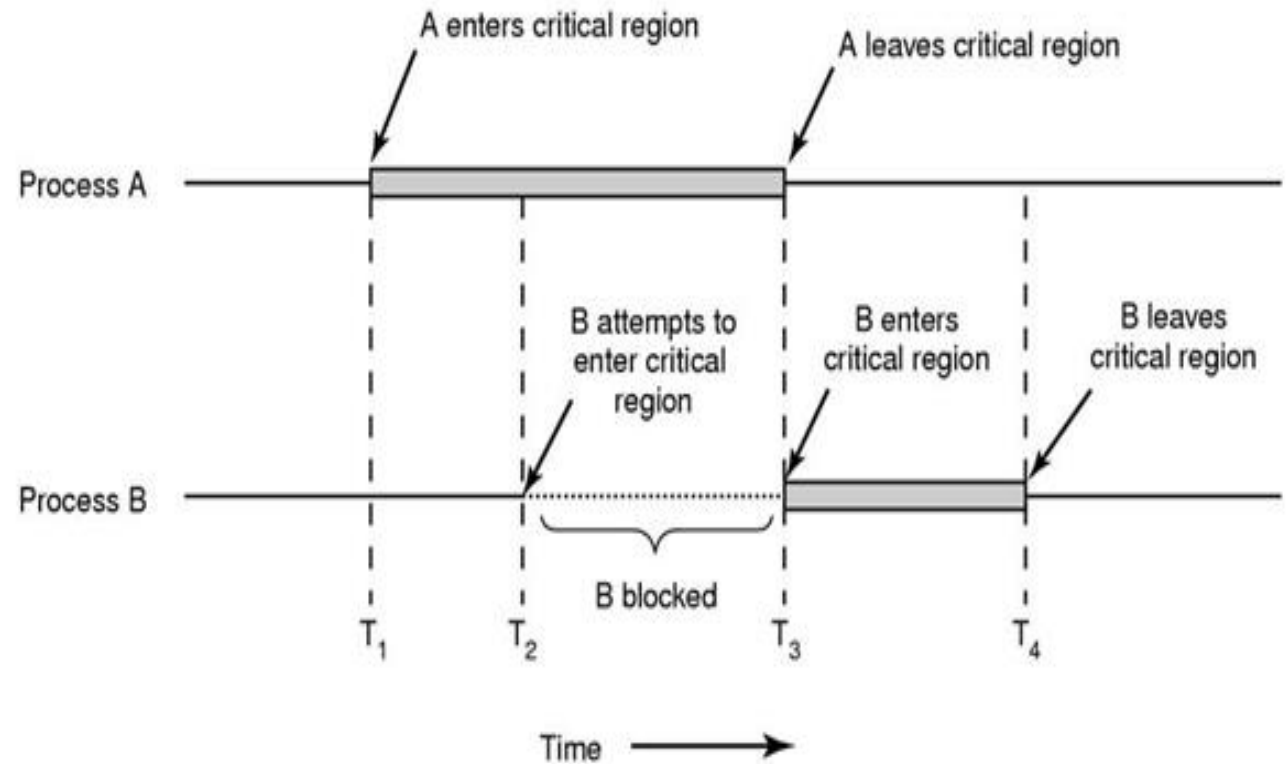


Mutual exclusion

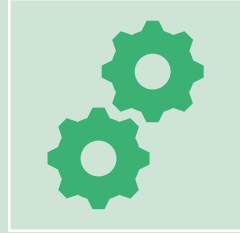
- It is a property of process synchronization which states that “no two processes can exist in the critical section at any given point of time”

Mutual Exclusion

Region of code that two or more related processes *must not execute at the same time* in order to avoid race-conditions.



Critical section



A very popular solution to **process synchronization** is the implementation of the **critical section**, which is a segment of code that can be accessed by only one single **process** at a certain instance in time.



The **critical section** is a portion of code where **processes** sharing data are controlled using semaphores.

General Structure of a Critical Section Solution

Entry Section:

- This code segment is executed when a process wants to enter its critical section.
- Eligibility of a process to enter critical section is checked here.
- This section assures that at most one process is in the critical section as per the requirement of a mutual exclusion.

```
do {
```

```
    entry section
```

```
        critical section
```

```
    exit section
```

```
        remainder section
```

```
} while (TRUE);
```


General Structure of a Critical Section Solution

Critical Section:

- This is a code segment where a process accesses a shared resource.
- At a time, only one of the co-operating processes will be in its critical section.
- When a process is executing its critical section, it is enjoying an exclusive access to the shared resource.

do {

entry section

critical section

exit section

remainder section

} while (TRUE);

General Structure of a Critical Section Solution

Exit Section:

- This code segment is executed when a process exits its critical section.
- A process performs certain operations, indicating its exit from the critical section.
- This section enables one of the waiting processes to enter its critical section.

```
do {
```

```
    entry section
```

```
    critical section
```

```
    exit section
```

```
    remainder section
```

```
} while (TRUE);
```

General Structure of a Critical Section Solution

Remainder Section:

- This is the remaining part of a process's code.
- A process performs tasks which are independent from other processes.

```
do {
```

```
    entry section
```

```
        critical section
```

```
    exit section
```

```
        remainder section
```

```
} while (TRUE);
```

Monitor

It is a programming language concept used to provide process synchronization.

It is collection of:

Variables: representing shared data

Procedures: representing a set of atomic operations on shared data

Condition variables: controlling progress of a process.

General Syntax of a Monitor

- Different languages have their own syntax to implement monitors.
- The general syntax of a monitor can be considered as given in the figure.

Monitor Demo // Name of the Monitor

```
{  
variables;  
  
condition variables;  
  
procedure p1 {.....}  
  
procedure p2 {.....}  
  
}
```




Q₁₀

U₁

E₁

S₁

T₁

I₁

O₁

N₁

S₁

H₄

R₁



Deadlocks

BY:

ROSHAN R. ROHIT

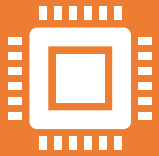
LECTURER,

IT DEPT,

GPG, SURAT



What are Resources ?



Resource is anything that can be used by only a single process at any instance of time.



It can be physical resources such as CPU, memory, printer, scanner, etc, or it can be logical resources such as files, tables, variables, etc.

Resource Types

Resources can be categorized in one of two categories:

1) Preemptable

2) Non-preemptable

Deadlocks can occur when processes have been granted exclusive access to such resources.

Non-preemptable resource

Resource that cannot be taken away from the process currently using it, without causing execution to fail.

Process gets affected when resource is taken away from it during its execution.

Such resource may cause deadlocks

Example: Printer, Scanner

Preemptable Resource

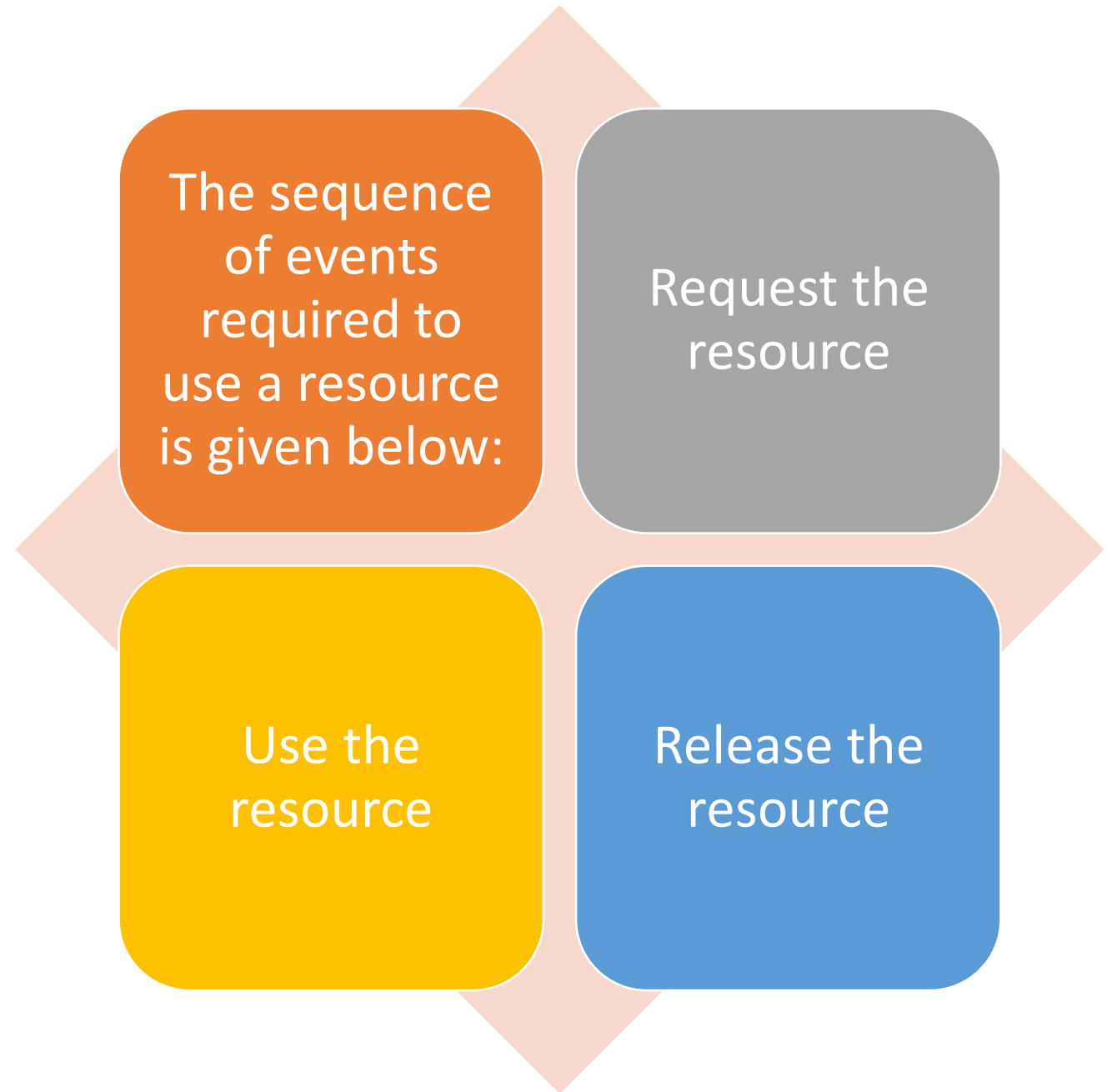
Resource that can be taken away from the process currently using it, with no ill effects.

Process doesn't get affected when resource is taken away from it during its execution.

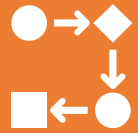
Such resources don't cause deadlocks

Example: Memory, CPU.

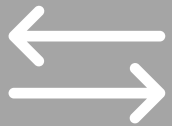
Events Required to use a Resource



What is a Deadlock??



A set of processes is deadlocked, if each process in the set is waiting for an event that only another process in the set can cause.



No process can come out from waiting state. All the processes will continue to wait forever..!!!

Conditions for DEADLOCK

MUTUAL EXCLUSION

Each resource can be assigned to exactly one process.

If any process request resource which is not free, then that process will wait until resource becomes free.

Conditions for DEADLOCK

HOLD AND WAIT

A process is holding at least one resource and waiting for additional resources.

Also such resources are currently being held by other processes.

Conditions for DEADLOCK

NO PREEMPTION

Resource cannot be preempted . Once resource are granted , they cannot be forcibly taken away from a process .

They must be explicitly released by the process holding them.

Conditions for DEADLOCK

CIRCULAR WAIT

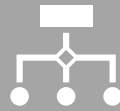
There must be a circular chain of 2 or more processes .

Each process is waiting for a resource which is held by the next member process of the chain.

Resource Allocation Graph



The four conditions for deadlock to occur can be modeled by using a directed graph.



This graph contains two types of nodes:



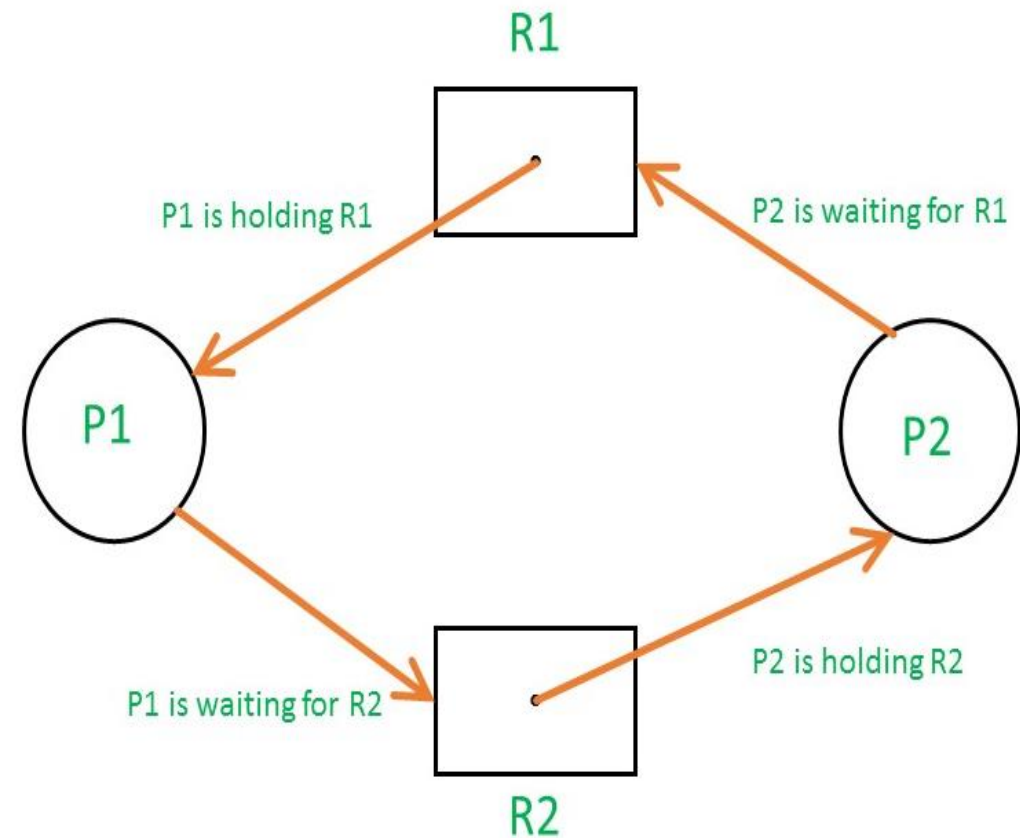
Processes, represented by circles



Resources, represented by squares

Resource Allocation Graph

- ❑ Such type of graph is called a resource allocation graph.
- ❑ It can be used to represent a deadlock.
- ❑ If a resource allocation graph contains a cycle, it means there is a deadlock.



SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK

Dealing with Deadlocks



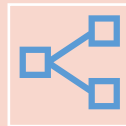
There are 4 strategies from prevention for DEADLOCK...



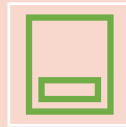
1. Just ignore the problem.



2. Detection and recovery.



3. Dynamic avoidance.



4. Prevention.

Just ignore
the
problem
altogether

According to this strategy, deadlock are ignored at all . It is assumed that deadlock never occur in the system.

The reason behind this is that the deadlock prevention price is high.

This is also known as 'Ostrich' algorithm: stick your head in the sand and pretend there is no problem at all..!!

UNIX operating system follows this strategy.

Detection and recovery.

This strategy detects the occurred deadlock and take some actions to recover from the deadlock.

To detect deadlock, a resource allocation graph is constructed including all processes and resources. Such type of checking is done periodically.

To recover form deadlock there are three way as discussed in the next slides.

Recovery through Preemption



In some cases, it may be possible to temporarily take a resource away from its current owner and give it to another.

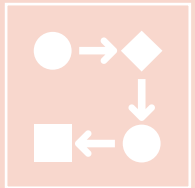


If so, do it.

Recovery through Rollback



Some processes are rolled back to a previous moment when there was no deadlock.

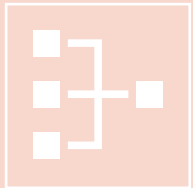


This requires periodic check pointing of processes which means its state is written to a file so that it can be restarted later.

Recovery through Termination



The easiest way to break a deadlock is to kill one or more processes.



One possibility is to kill a process in the cycle.

Dynamic avoidance



This strategy suggest us to avoid deadlock.



This can be done by carefully allocating resources.

Deadlock Prevention

There are four conditions for a deadlock to occur. Deadlocks can be prevented by attacking one of these conditions.

1. Mutual exclusion

2. Hold and wait

3. No preemption

4. Circular wait

Mutual exclusion

Allow sharable use of resources whenever possible.

If possible, resources can be spooled.

Not possible for all resources.

Hold and Wait

It requires that all processes should request all the resource at the beginning of execution.

If everything is available, process will be allocated whatever it ends. Then, it can run to completion.

If one or more resources are busy, nothing will be allocated, and the process would just wait.

No preemption

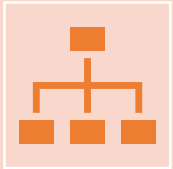


Normally, not possible.

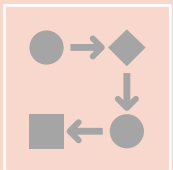


Because non-preemptable resources like printer, scanner cannot be preempted.

Circular Wait



Resources are allocated in some fixed order.



It is assured that there is no cycle in allocating resources.

