SUBJECT: BASICS OF OPERATING SYSTEM
[4330703] SEMESTER: 3RD

Unit : 2

Process Management

A. K. Panchasara

Sr. Lecturer in Computer Engineering

AVPTI-Rajkot

Process Concepts

A process is program in execution. A process defines the fundamental unit of computation for the computer

- **Program v/s Process**

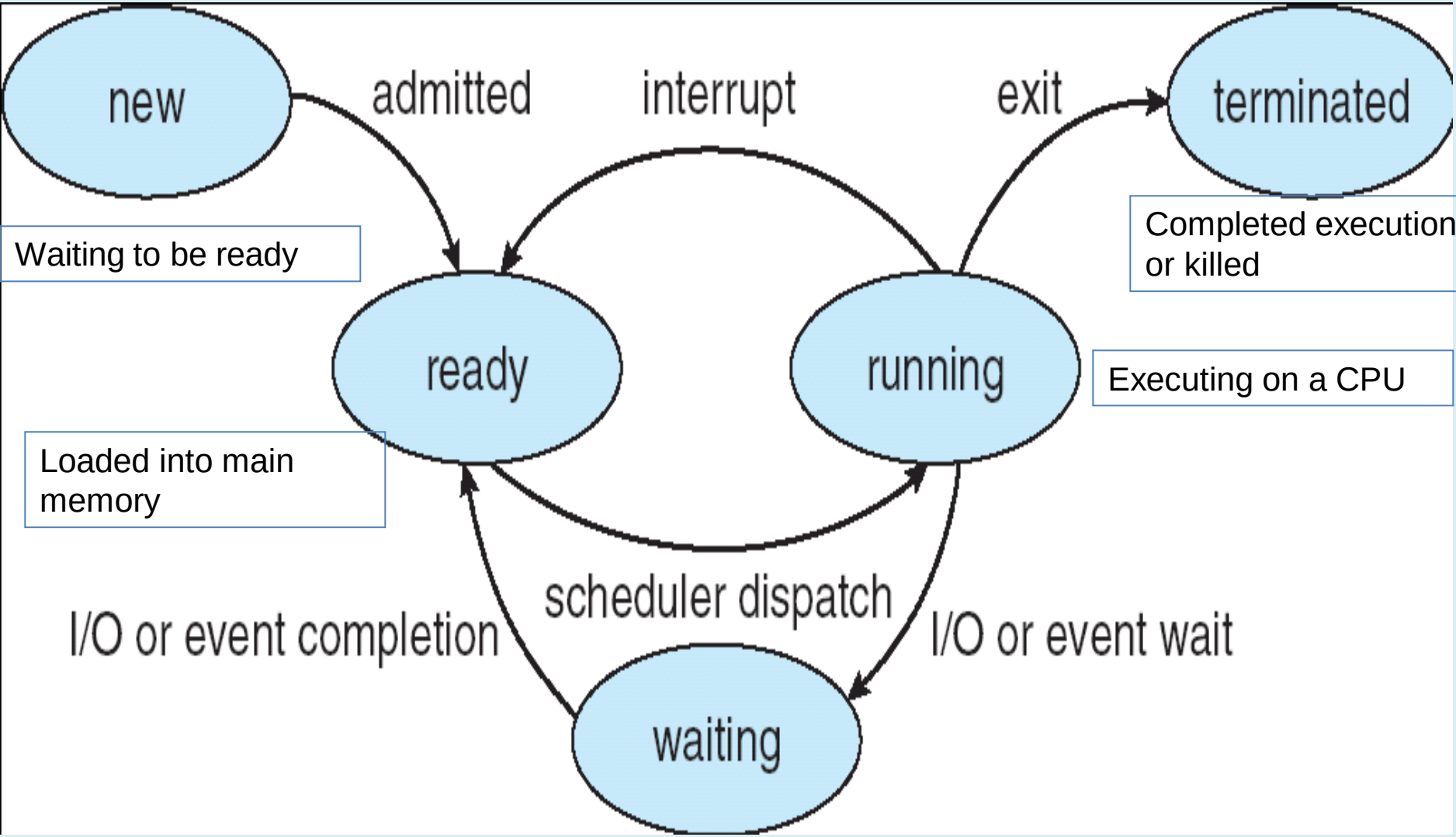
Program	Process
• a program is an organized collection of instructions	• it is an instance of computer program that is being executed • Process is a program in execution
• its like a recipe of cooking. (contain variables and statements)	• its referred as a task or job.
• it's a **static / passive** entity	• process is **active / dynamic** entity
• it has a **longer life span**	• it has **limited life span**
• a program is stored on disk • it requires memory space on disk	• it contain various resources like- - memory, space, disk, printer etc • it contain memory space (add space)

•Several processes can be associated with the same program

Process Concepts

- Process creation and Process termination
- Process creation (4 principal events)
  1. System initialization → when system is booted, they do not need user interaction and execute in background. Its called **daemon** process.
  2. Execution of a process creation system call →running process can create a new process using system call
     - in UNIX → 'fork' and in windows → 'create process'
  3. User request to create new process → typing a command or double click
  4. Initiation of a batch job → apply to only batch systems.
- Process termination (4 principal events)
  1. Normal exit (voluntarily)
  2. Error exit (voluntarily) → when process cause error
  3. Fatal error (in voluntarily) → ex – delete a file which is not exist
  4. Killed by another process (in voluntarily) →in UNIX – 'kill' and in windows – 'terminate process'

Process Life Cycle / Process States



Process Life Cycle / Process States

- It's a dynamic activity, it changes state in time duration in its execution
- Each process may be one of the following states.
- **NEW:**
  - When a process is first created. It occupies NEW state. It awaits to enter in 'ready' state.
- **READY:**
  - In which process has been loaded into main memory. Now process awaits for execution on CPU.
  - It is possible that more than one process in this state.
- **RUNNING:**
  - In which a process is currently executing on CPU
  - In a one processor system, one and only one process can be in this state.

Process Life Cycle / Process States

- **WAITING:**
  - In which process is waiting for some I/O completion, or some event to occur.
  - Ex → printing a document
- **TERMINATED:**
  - Process is in this state when it terminates.
  - Process can be terminated by completion its execution or it can be killed explicitly.

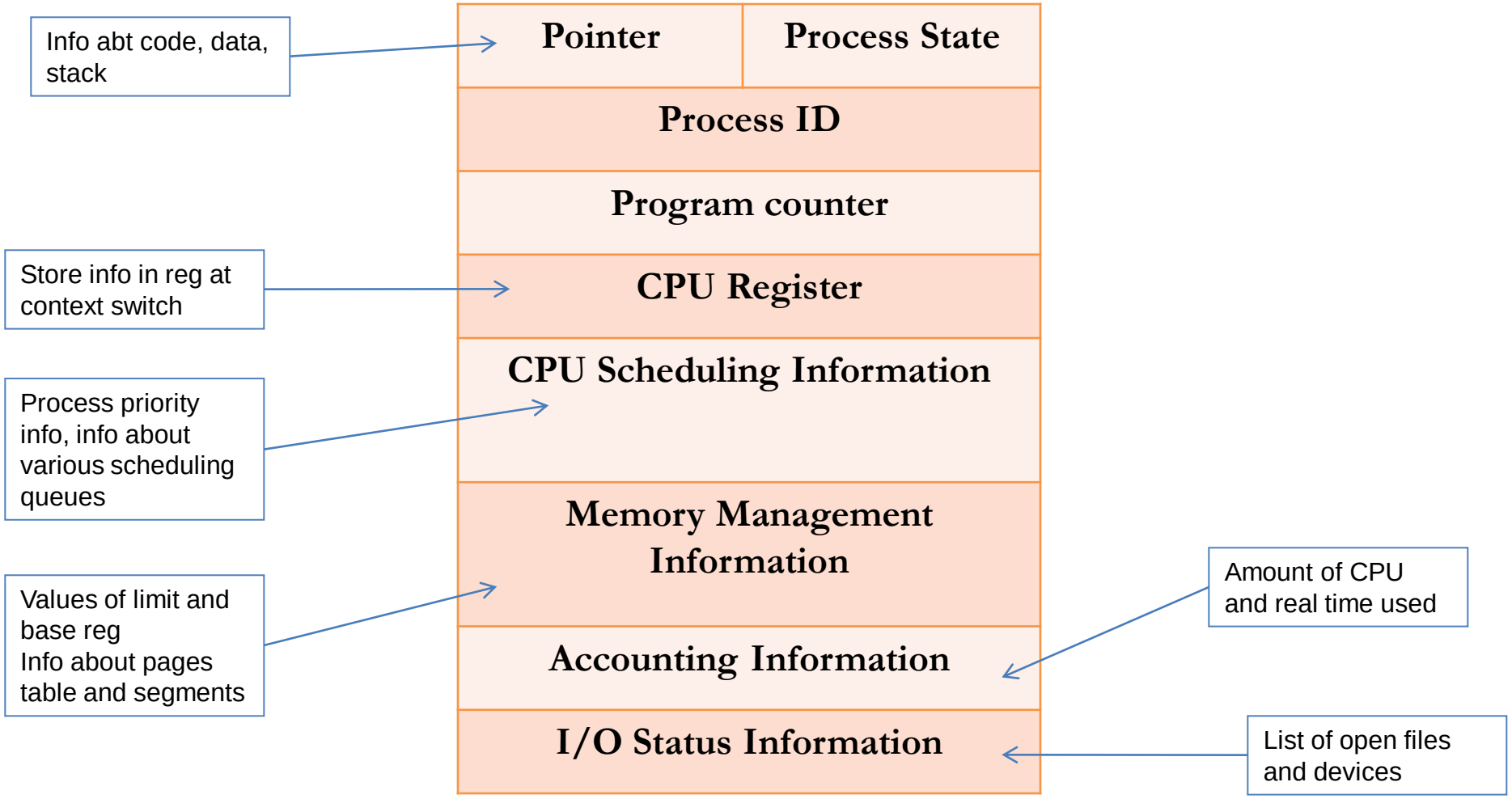
From state	Event	To state
New	Accepted	Ready
Ready	Scheduled/dispatch	Running
Running	Need I/O	Waiting
Running	Scheduler time out	Ready
Running	Completion/Error/Killed	Terminated
Waiting	I/O completed or wake up event	Ready

Process Control Block (PCB)

- OS is responsible for various activities related to process management.
- OS manages all concurrent processes.
- OS maintains all such processes by storing all information about each process in ***PROCESS TABLE***.
- PROCESS TABLE contains various entries.
- ONE entry per PROCESS called Process Control Block (PCB) or Task Control Block (TCB)

Process Control Block (PCB)

- Definition



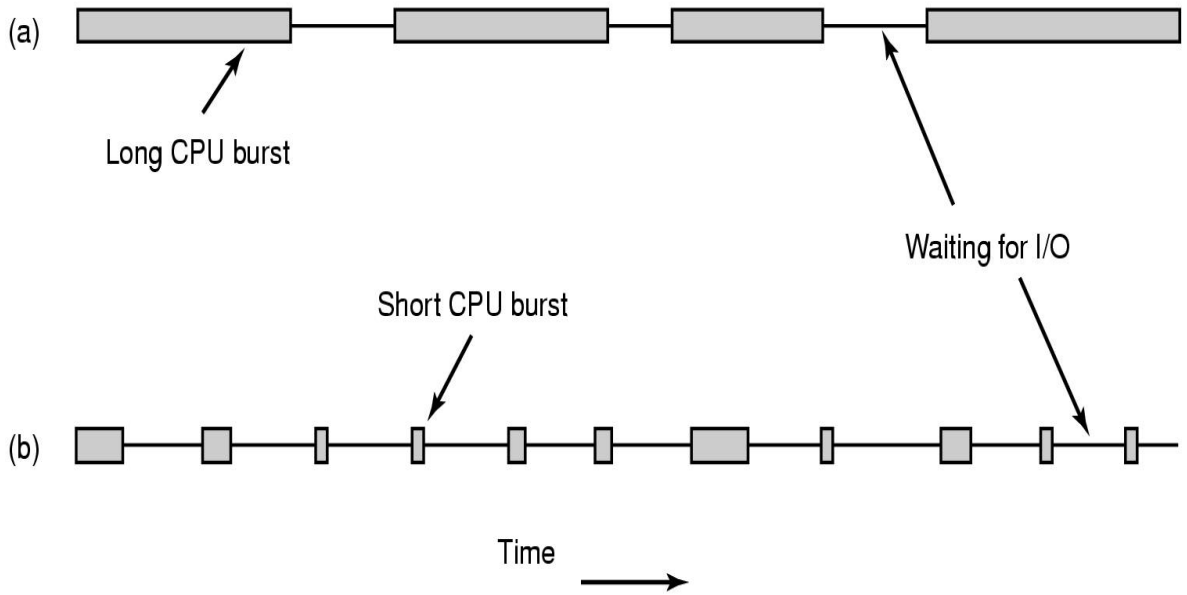
Process and Processor Scheduling (1)

- Concept of Scheduling (with few definitions)
- ***(1) Scheduling-*** is the process of making a choice of which process to run next whenever more than processes are simultaneously in the READY state
- Scheduling affects the resource utilization and overall system performance.
- ***(2) Scheduler-*** is an OS module (part of an OS) which actually makes a choice of which process to run next whenever more than one processes are in the READY state.
- Scheduling refers to the entire process of selecting a process while, scheduler actually does this job.
- ***(3) Scheduling algorithm-*** is an algorithm, which is used by the scheduler to do the process of scheduling

Process and Processor Scheduling (2)

- Goals →
  - Optimum utilization of resources (efficiency)
  - Fair scheduling to all process/jobs (no-corruption)
  - More critical processes get priority over others.(priority)

- Process Behavior →
  - CPU Burst and I/O Burst
  - CPU bound processes and I/O bond processes



Process and Processor Scheduling (3)

- When to schedule ?
  - Running to waiting, running to terminate, running to ready, waiting to ready
  - Non pre-emptive scheduling and pre-emptive scheduling
- Scheduling Queues
  - To keep watch in which state the process is

1. Job queue
   1. It consists of all processes in the system
2. Ready queue
   1. It consists of processes which are residing in main memory, ready to run and waiting for CPU
3. Device queue
   1. It consists of processes which are waiting for a particular device.
   2. Each devices has its own device queue

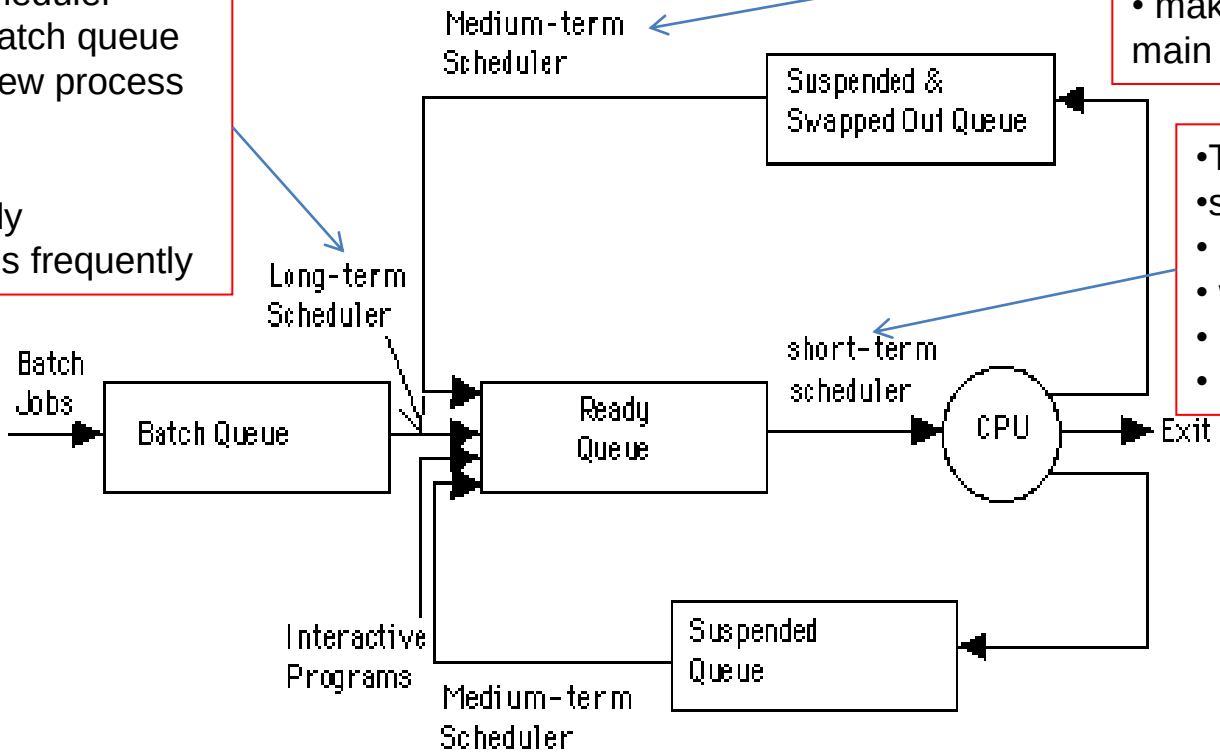
Process and Processor Scheduling (4)

- Types of Schedulers

•First level scheduler
•Works with batch queue
•need when new process enters
• slow speed
•New → Ready
• executes less frequently

•Second level scheduler
•used for swapping
• executes in two occasions
• speed is intermediate
• make efficient use of main memory

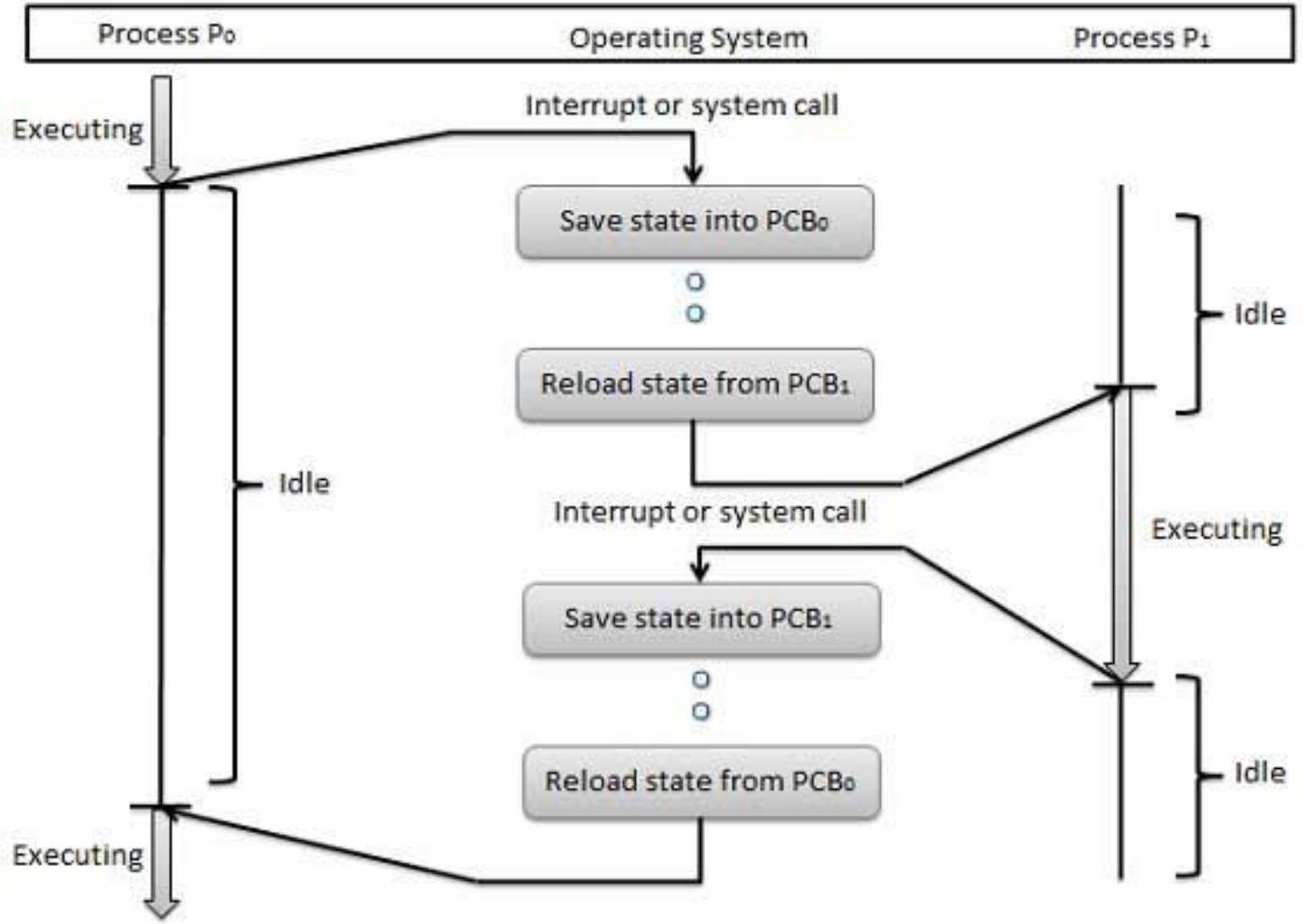
•Third level scheduler
•speed is fast
• it is CPU scheduler
• works with ready queue
• State :Ready → Running
• keep CPU always busy



Context Switch

- Process context is the information about the process, which is stored in PCB
- Context switch is the process of transferring control of the CPU from running process to another process selected from a ready queue
- Four events when context switch requires:
  1. Process request for I/O operation
  2. Process request wait for some event (Eg. Lock on database table to be released)
  3. Process terminates
  4. Process pre-empted
- Dispatcher → is an OS module which transfers control of CPU from one process to another
- Steps:
- Save the context of running process in PCB
- Transfer the running process from running to waiting or ready (depending on event)
- Choose process from the ready queue.
- Load the context of this process from its PCB.
- Transfer the chosen process from the ready to the running and allocate CPU to this process.

Example: Context Switch



Performance Criteria

1. CPU Utilization
   - It is the time during which the CPU is busy.
   - Ranges from 0 to 100 % (optimum is 40 to 90)
   - CPU should remain as busy as possible.
2. Throughput
   1. No of processes completed per time unit
3. Turn around Time: it is the time required to complete the execution of a process is called T.A.T

   Turn around time (TAT) = process finish time (F.T) – Process arrival time (A.T)

Performance Criteria

4. Waiting time = turn around time (T.A.T) – actual execution time (A.E.T)
   - It is total time duration spent by a process waiting in ready queue/state.
   - AET is also called Burst Time(BT)
5. Response time
   - It is a time between issuing a command/request and getting output/result.
   - User request should be as quickly as possible.
   - User requests are given more priority compared to background processes.

Scheduling Algorithms

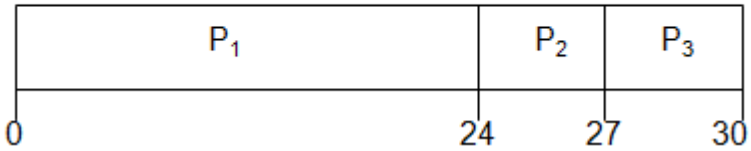
1. FCFS (First Come First Served)

- Selection criteria : process request first, served first.
- Decision mode : Non preemptive
- Implementation :
- Example :

Process	Arrival Time	Execution Time	Finish Time	TAT	Waiting Time
P0	0	24	24	24	0
P1	0	3	27	27	24
P2	0	3	30	30	27

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3
The Gantt Chart for the schedule is:



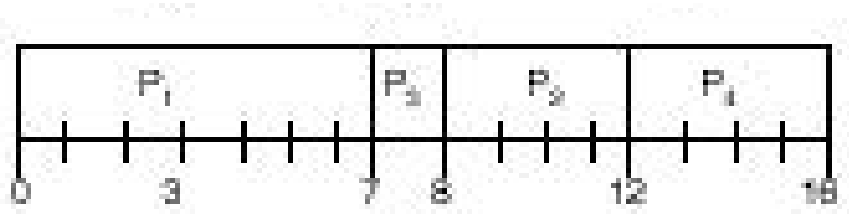
- Avg TAT = (24+27+30)/3 = **27**
- Avg Waiting time
  = (0 + 24 + 27)/3 = **17**

Scheduling Algorithms

1. SJF (Shortest Job First)

- Selection criteria : shortest process is served first.
- Decision mode : Non preemptive
- Implementation :
- Example :

Process	Arrival Time	Execution Time	Finish Time	TAT	Waiting Time
P1	0	7	7	7	0
P2	2	4	12	10	6
P3	4	1	8	4	3
P4	5	4	16	11	7



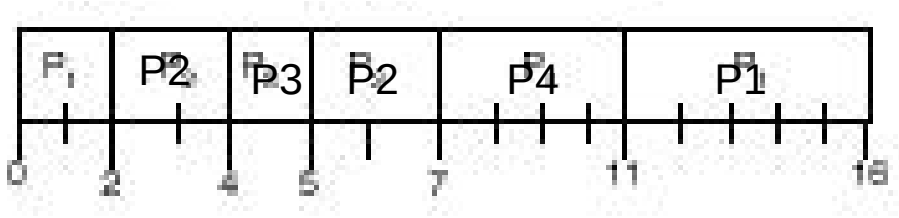
- Avg TAT = **16.25**
- Avg Waiting time = **4**

Scheduling Algorithms

1. SRTN (Shortest Remaining Time Next) / SJF with Pre-emption

- Selection criteria : process having remaining time shortest is served first
- Decision mode : **PREEMPTIVE**
- Implementation : Processes are sorted based on remaining running time
- Example :

Process	Arrival Time	Execution Time	Finish Time	TAT	Waiting Time
P1	0	7	16	16	9
P2	2	4	7	5	1
P3	4	1	5	1	0
P4	5	4	11	6	2



- Avg TAT = **7**
- Avg Waiting time = **3**

Scheduling Algorithms

1. Round Robin (RR)

- Selection criteria : same as FCFS
- Decision mode :preemptive with time slice/time quantum/time interval
- Implementation : (1) CPU burst < Time quantum then voluntarily release CPU
- (2) CPU burst > TQ then preempted and moved to the end of queue.
- Example :

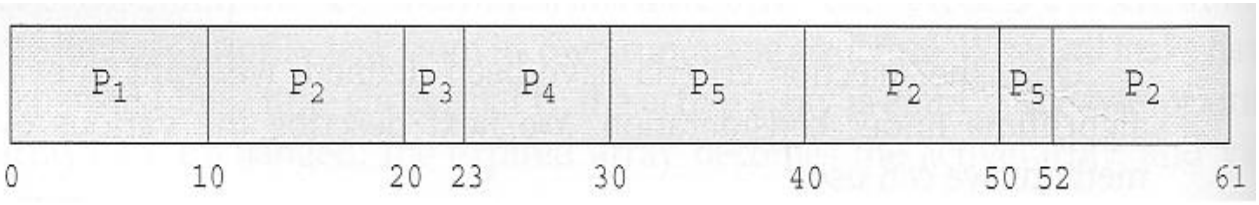
Process	Arrival Time	Execution Time	Finish Time	TAT	Waiting Time
P1	0	10	10	10	0
P2	0	29	61	61	32
P3	0	3	23	23	20
P4	0	7	30	30	23
P5	0	12	52	52	40

Scheduling Algorithms

1. Round Robin (RR)

- Selection criteria : same as FCFS
- Decision mode :preemptive with time slice/time quantum/time interval
- Implementation : (1) CPU burst < Time quantum then voluntarily release CPU
- (2) CPU burst > TQ then preempted and moved to the end of queue.
- Example :

Process	Arrival Time	Execution Time	Finish Time	TAT	Waiting Time
P1	0	10	10	10	0
P2	0	29	61	61	32
P3	0	3	23	23	20
P4	0	7	30	30	23
P5	0	12	52	52	40



- Avg TAT = **23**
- Avg Waiting time = **35.2**

Inter Process Communication (IPC)

- Simultaneously running processes communicate with each other → is called IPC
- Examples are:
  - A Shell pipeline in UNIX
  - Printing on printer in network
  - Chat or Mail server
- IPC is useful in distributed environment.
- Issues in IPC → three main issues are:

1. How one can pass info to another
   1. Using shared memory, shared file, message passing
2. Two or more processes should not get in to each others way
   2. Second process has to wait until first process finishes
3. Proper sequencing should be maintained

Inter Process Communication (IPC)

Independent Process	Co-operating Process
- Don't communicate with each other	- Can communicate with each other
- Process doesn't share any data	- Process share any data with other processes
- It can't affect or been affected by other processes	-It can affect or been affected by other processes
- It doesn't participate in IPC	- It participate in IPC

Inter Process Communication (IPC)

- **Advantages of co-operating processes**

1. Information sharing
2. Computation speed
3. Modularity
    1. Each system function should be divided into separate process or threads
4. Convenience
    1. Used when parallel tasks need to execute

- **Disadvantages of co-operating processes**

1. Race condition
2. Need proper synchronization

Process Synchronization

- ***Definition:*** Is a mechanism to ensure a **systematic sharing** of resources among concurrent processes.
- Race condition (Racing Problem):
- ***Definition:*** Situations, where two or more processes are reading or writing some ***shared*** data and final result depends on the *relative order of their execution is called…*
- Ex: Possibility 1 (Initial value of A = 1000)

Process P0	Process P1
Read(A); --- -- A = A – 100; Write(A); -- --	-- -- Read(A) -- -- A = A + 200 Write(A);

Process Synchronization

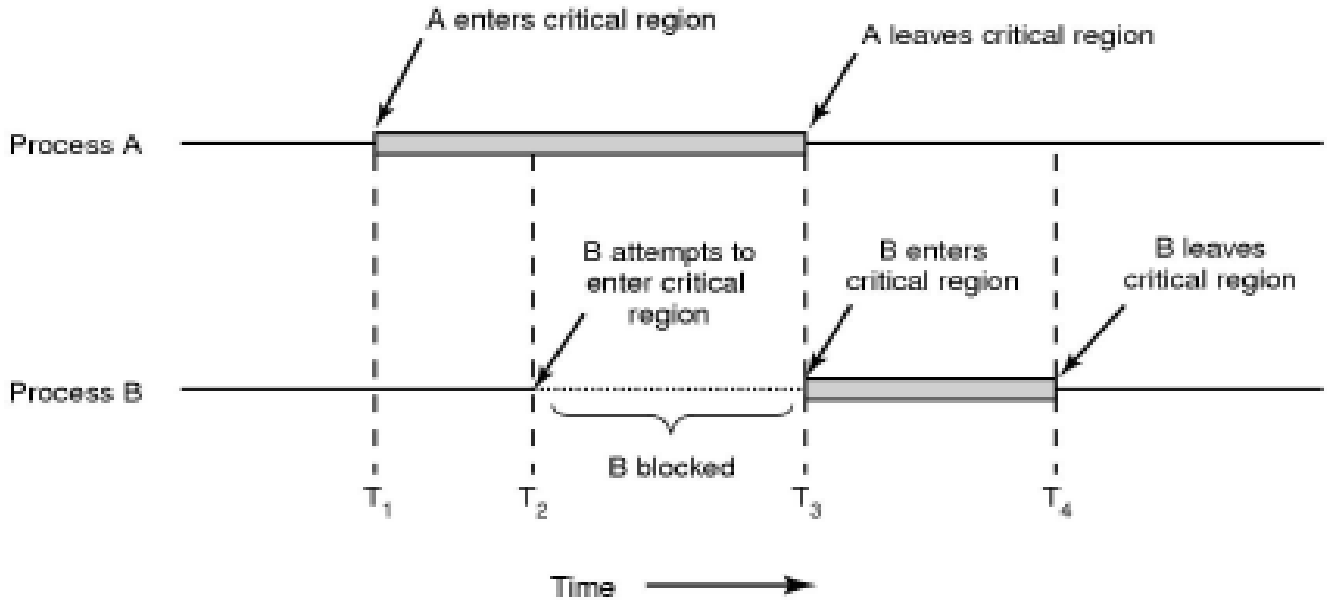
- Ex: Possibility 2

Process P0	Process P1
Read(A); --- -- -- -- A = A – 100; Write(A);	-- Read(A) A = A + 200 Write(A); -- -- --

- Shared resources should be some variables, some file or even some physical device.
- Race condition is a problem with synchronization and it should be avoided.

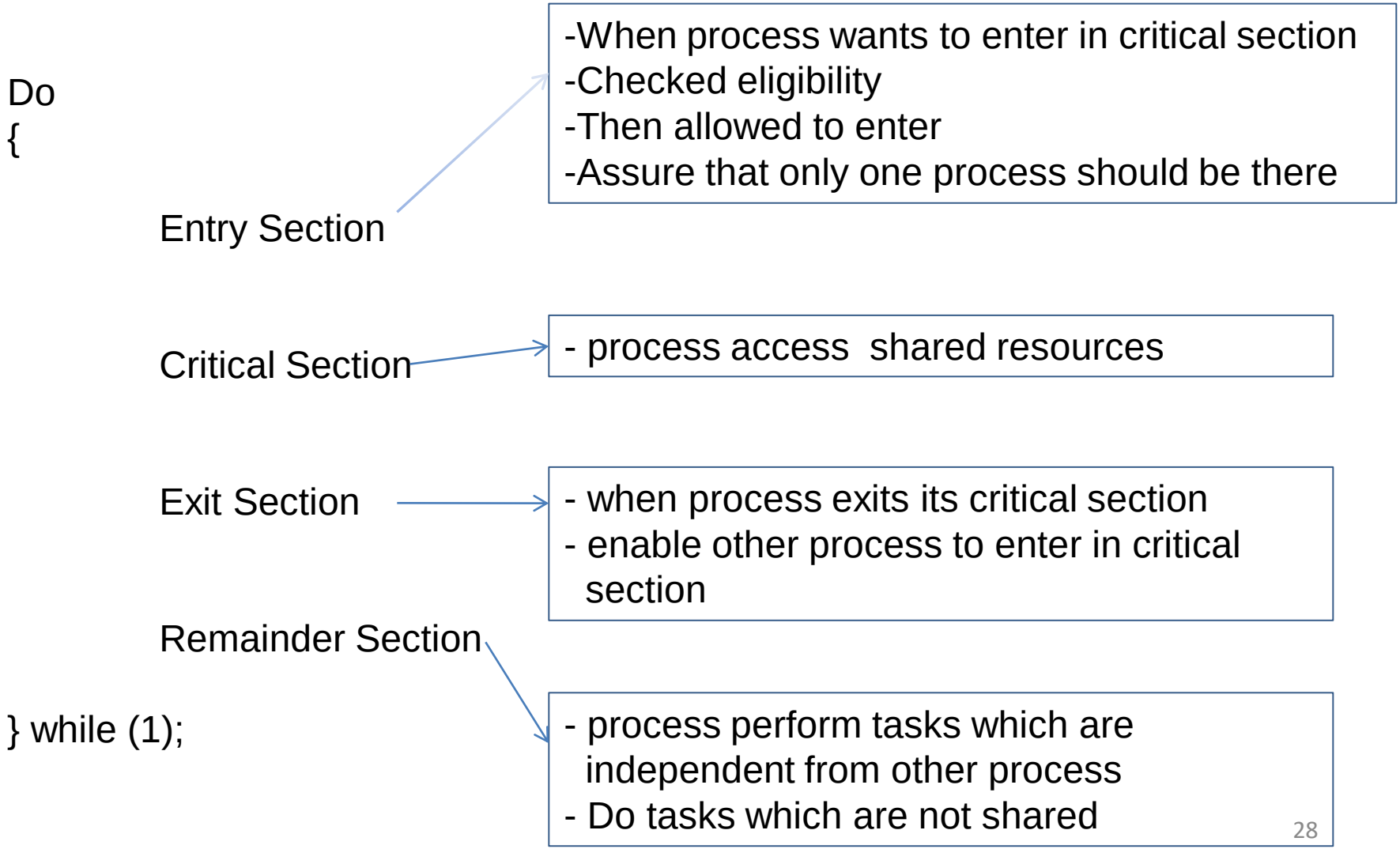
Process Synchronization: Mutual Exclusion and Critical Section

- A code segment of a process where the shared resources are accessed is referred as a **critical section. (critical region)**
- **Mutual exclusion** is the way of making sure that if one process is in critical section then other excluded from it.
- **Only one process should be allowed to execute in critical section**.
- It is the responsibility of the OS to keep track.



Process Synchronization: Mutual Exclusion and Critical Section

- General structure of Critical Section solution



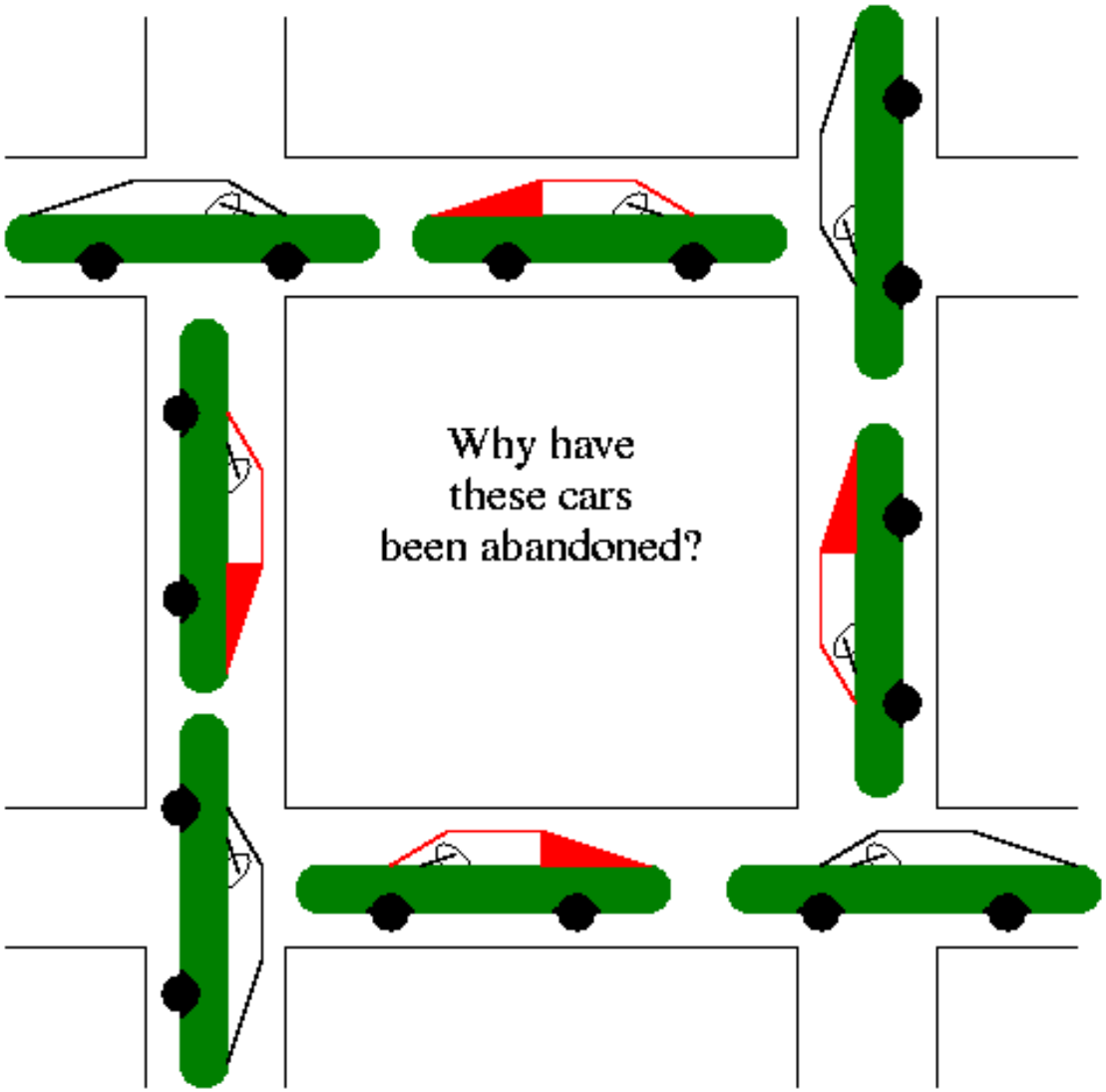
Monitor

- Is a programming lang. concept Used to provide process synchronization
- Monitor is a collection of →
  - Variables : representing shared data
  - Procedures : set of atomic operations on shared data
  - Condition variables : to control progress of a process
- General syntax of Monitor is →

```
Monitor monitor_name
{
        // shared variable
        // condition variable
        procedure P1(…)
        {
        }
        procedure P2(…)
        {
        }
}
```

- Process can't access monitor's data directly
- to achieve mutual exclusion →Only one process can be active in a monitor
- condition var allow a process to get blocked
- '***signal'*** and ***'wait'*** can be performed on condition var
- all languages do not support monitor
- lang like JAVA, C# provide support to monitor

Deadlocks



Deadlocks

- Resources
  - **is anything that can be used by only a single process at any instance of time.**
  - It can be physical resources or can be logical
  - Deadlocks can occur when processes have been granted exclusive access to such resources.
  - **Pre-emptable resources:**
    - Resources can be taken away from the process currently using it
    - Ex → Memory, CPU
    - Don't cause deadlock
  - **Non Pre-emptable resources:**
    - Resources can't be taken away from the process
    - May cause deadlock Ex→ Printer, Scanner

Deadlocks

- A set of processes is deadlocked if each process in the set is waiting for an event that only other process in the set can cause.
- None of the processes can come out of 'waiting' state and all processes will continue to wait forever.
- **Conditions for Deadlocks: (four main conditions)**

1. **Mutual Exclusion :** each resource can be assigned to exactly one process which is not free.
2. **Hold and Wait :** a process must be holding at least one resource and waiting for additional resource, and that is currently held by other process
3. **No preemption :** resources can't be preempted
4. **Circular Wait :** there must be a circular chain of two or more processes.

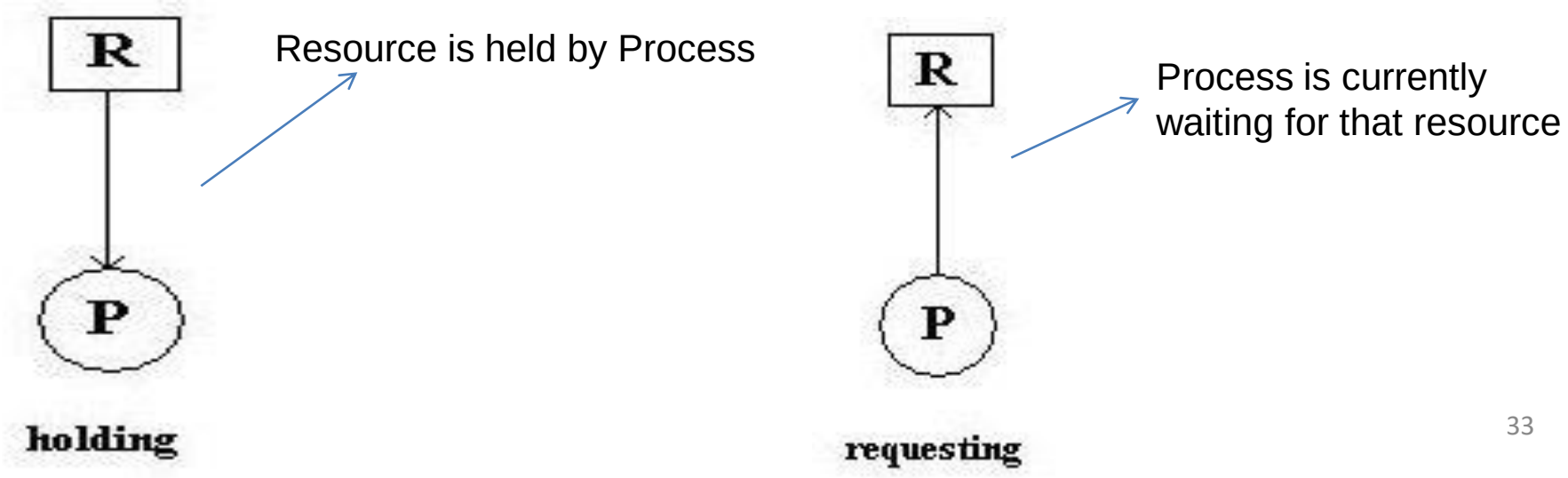
(All of these four conditions must be present to occur a deadlock.)

Deadlocks

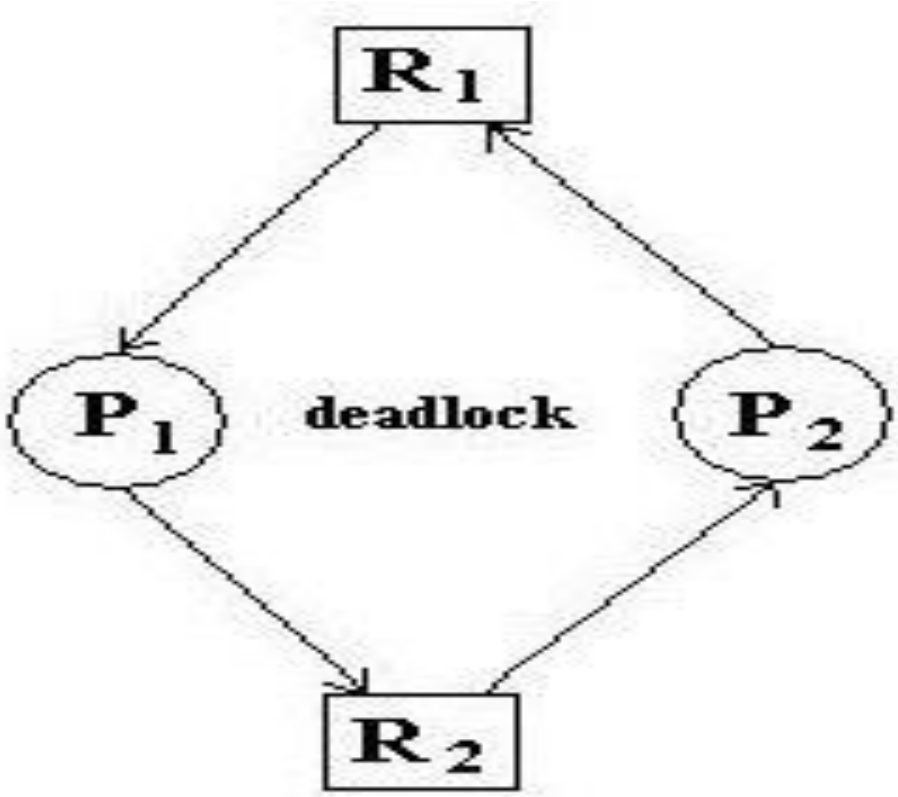
- **Resource Allocation Graph**
- is used to represent Deadlock
- All four conditions that must be needed to occur Deadlock, can be modeled by using this graph.
- A graph contains two types of nodes:

1. Processes → represented by **circles**
2. Resources → represented by **squares**

2



Deadlocks



A cycle in a graph means that there is a deadlock

Deadlocks

Dealing with Deadlocks

- Four strategies are used :

1. **Ignore the problem** → it is assumed that deadlock will never occur in the system.'

   - This is also known as '***Ostrich Algorithm***', Ex : UNIX OS

   ("To stick one's head in the sand and pretend there is no problem")

2. **Detection and Recovery** → allow deadlocks to occur

   - don't prevent for deadlock, but periodically checks for deadlocks
   - For that resource allocation graph is constructed
   - To recover from deadlock:

   I. Recovery through preemption (take away the resource forcefully)

   II. Recovery through rollback (rolled back to a moment where no deadlock)

   III. Recovery through termination (kill process)

Deadlocks

3. **Dynamic Avoidance** → avoid deadlock by carefully allocating the resources.

- Resources will be allocated only if it is safe (no deadlock).

4. **Prevention** → deadlock can be prevented by attacking one of four possibilities that needed to occur in deadlock.

I. Mutual Exclusion :

- Allow sharable resources to use OR
- Resources can be spooled (for ex. One process will perform the printing operation called printer daemon. Other process will request for printing to printer daemon process. This daemon process will do task on behalf of requesting process. Printer daemon process doesn't require any resource rather than printer.)
- Not possible for all resources

II.Hold and Wait :

- All the processes should request all the resources at the beginning of execution.
- If everything is available, resources will be allocated.
- If one or more resources are busy, nothing will be allocated
- Problem
  - It is difficult to determine in advance which resources are required in execution

Deadlocks

III.No Pre-emption :

- If a process holding a resource, that must be released before acquiring any other resource.
- If a process request a resource and if that is held by any other process, OS may preempt the second process and force it to release the resource.
- Sequential I/O devices can't be pre-empted like – printer and scanner
- CPU and main memory can be pre-empted.

IV.Circular Wait :

- Resources are allocated in some fixed order. And, it is assured that there is no cycle in allocating resources.

Threads ..

Thank you...