UNIT - 2

Process and Process Management

What is a Process?

- A process is a program in execution.
- A process in execution needs resources like processing resource, memory and IO resource.
- Imagine a program written in C – my_prog.c.
- After compilation we get an executable.
- If we now give a command to run the program, it becomes a process.

What is a Process?

- A computer can have several process running or active at any given time.
- In case of multiple users on a system, all users share a common processing resource.

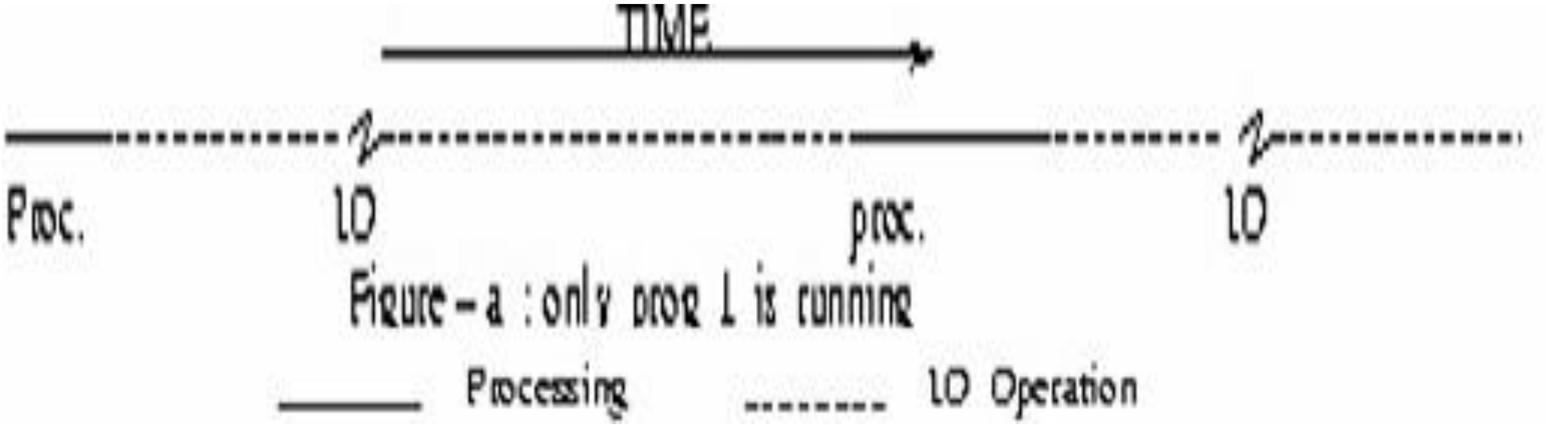
Multi-programming and Time Sharing

- Let us consider a system with only one processor and one user running one program: prog_1.

- IO and processing will happen alternately.

- When IO is required, say keyboard input, the processor idles. This is because we are nearly a million times slower than the processor !!!

One Program - One User - Uni-processor Operation



*Note: Most of the time the processor is idling !!

PROCESSOR UTILIZATION

- What percentage of time are we engaging the processor?
- Recall that for computing requires a program to reside in main memory to run.
- Clearly, having just one program would result in gross under utilization of the processor.
- To enhance utilization, we should try to have more than one ready-to-run program resident in main memory.

Processor Utilization (cont..)

- A processor is the central element in a computer's operation.
- A computer's throughput depends upon the extent of utilization of its processor.
- Previous figure shows processor idling for very long periods of time when only one program is executed.
- Now let us consider two ready-to-run memory resident programs and their execution sequence.

MULTI-PROGRAMMING SUPPORT IN OS

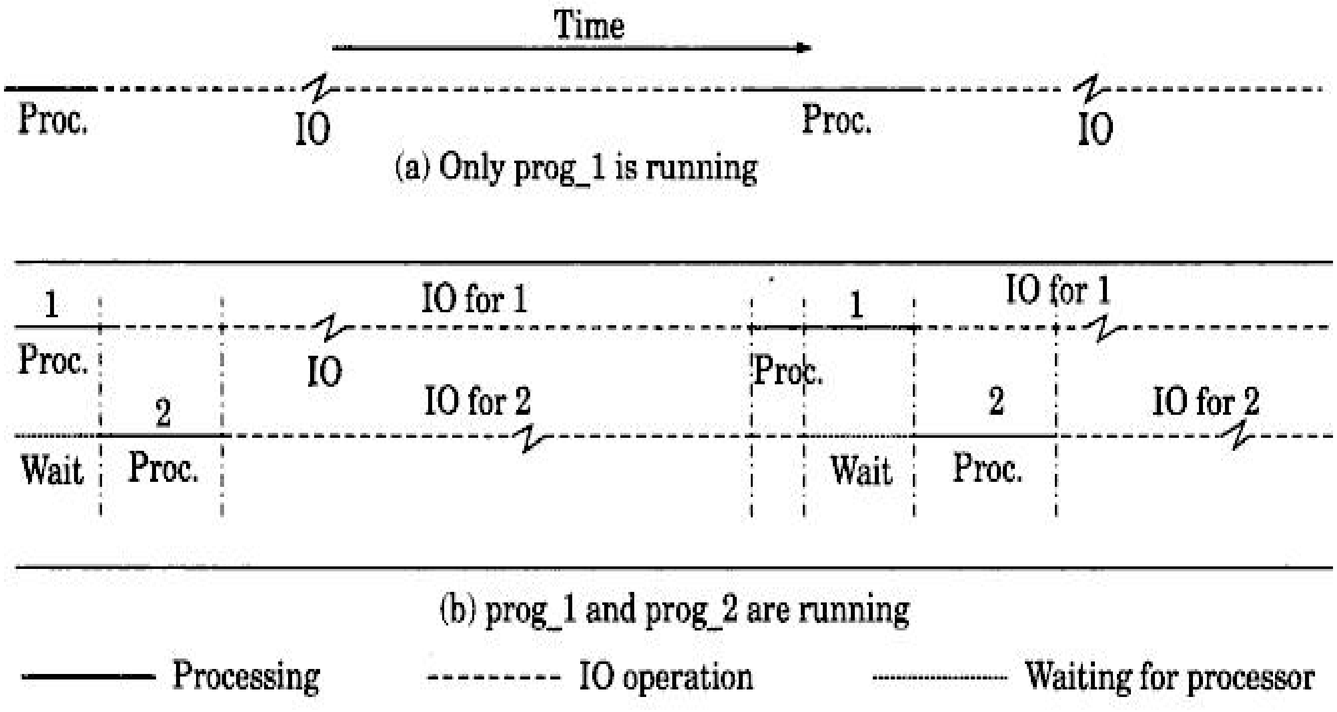


Figure 3.1 Multiple-program processing.

Consider two programs : *prog_1 and prog_2* resident in main memory.

MULTI-PROGRAMMING SUPPORT IN OS (CONT..)

IN THE FIGURE,

- WHEN PROG_1 IS NOT ENGAGING THE PROCESSOR MAY BE UTILIZED TO RUN ANOTHER READY-TO-RUN PROGRAM.

- CLEARLY, THESE TWO PROGRAMS CAN BE PROCESSED
  WITHOUT SIGNIFICANTLY SACRIFICING THE TIME REQUIRED TO PROCESS EITHER OF THEM.

MULTI-PROGRAMMING SUPPORT IN OS (CONT..)

- Disadvantage - overhead faced while switching the context of use of the processor.
- Advantage - computer resource utilization is improved.
- Advantage – memory utilization is improved with multiple processes residing in main memory.
- A system would give maximum throughput when all its components are busy all the time.

Response Time

Consider the following scenario :

- The number of ready-to-run programs must be maximized to maximize throughput of processor.
- These programs could belong to different users.
- A system with its resources being used by multiple users is called time sharing system.
- For example, a system with multiple terminals. Also a web server serving multiple clients
- However, such a usage has overheads. Lets see some of the overheads

Response Time

- In case of a switch in the context of the use of the processor, we must know where in the program sequence the program was suspended.

- In addition, intermediate results stored in registers have to be safely stored in a location before suspension.

Response Time

- When a large number of resident user programs compete for the processor resource, the *frequency of storage, reloads and wait periods also increase.*

- If overheads are high, users will have to wait longer for their programs to execute **=>*Response*** Time of the system becomes longer.

- Response Time *is the time interval which spans the time from when the last character has been input to the time when the first character of the output appears.*

Response Time

- In a time sharing system, it is important to achieve an acceptable *response time.*

- In a plant with an *on-line system, system devices* are continuously monitored : to determine the *criticality of a plant condition. Come to think of it* even a library system is an on-line system.

- If an online system produces a response time within acceptable limits, we say it is a *real-time system.*

IMPORTANT TERMS

- Multiprogramming : A mode of operation that provides for the interleaved execution of two or more computer programs by a single processor.

- Time Sharing : The concurrent use of device by a number of users.

Con..

- Time Slice : The maximum amount of time that a process can execute before being interrupted

- PCB : The manifestation of a process in an operating system. It is a data structure containing information about the characteristics and state of process.

IMPORTANT TERMS

- Swapping : Lifting the program from the memory to the disk & from the disk to the memory is known as Swapping.

- CPU Utilization : We want to keep CPU as busy as possible means to maximize the usage of processor among the users.

CON..

- Response Time : Response Time is the time interval which spans the time from when the last character has been input to the time when the first character of the output appears.

- Throughput : It is the total volume of work performed by the system over a given period of time.

Important Terms

- Turnaround Time : The interval from the time of submission of a process to the time of completion is measured as Turn-around time.

- Waiting Time : The period spent waiting in the ready queue is measured as waiting time.

PROCESS STATES

In all the previous examples, we said

- A process is in "*RUN*" *state if is engaging the* processor,
- A process is in "*WAIT*" *state if it is waiting for IO to* be completed
- In our *simplistic model we may think of 5 states:*

  - *New-process*

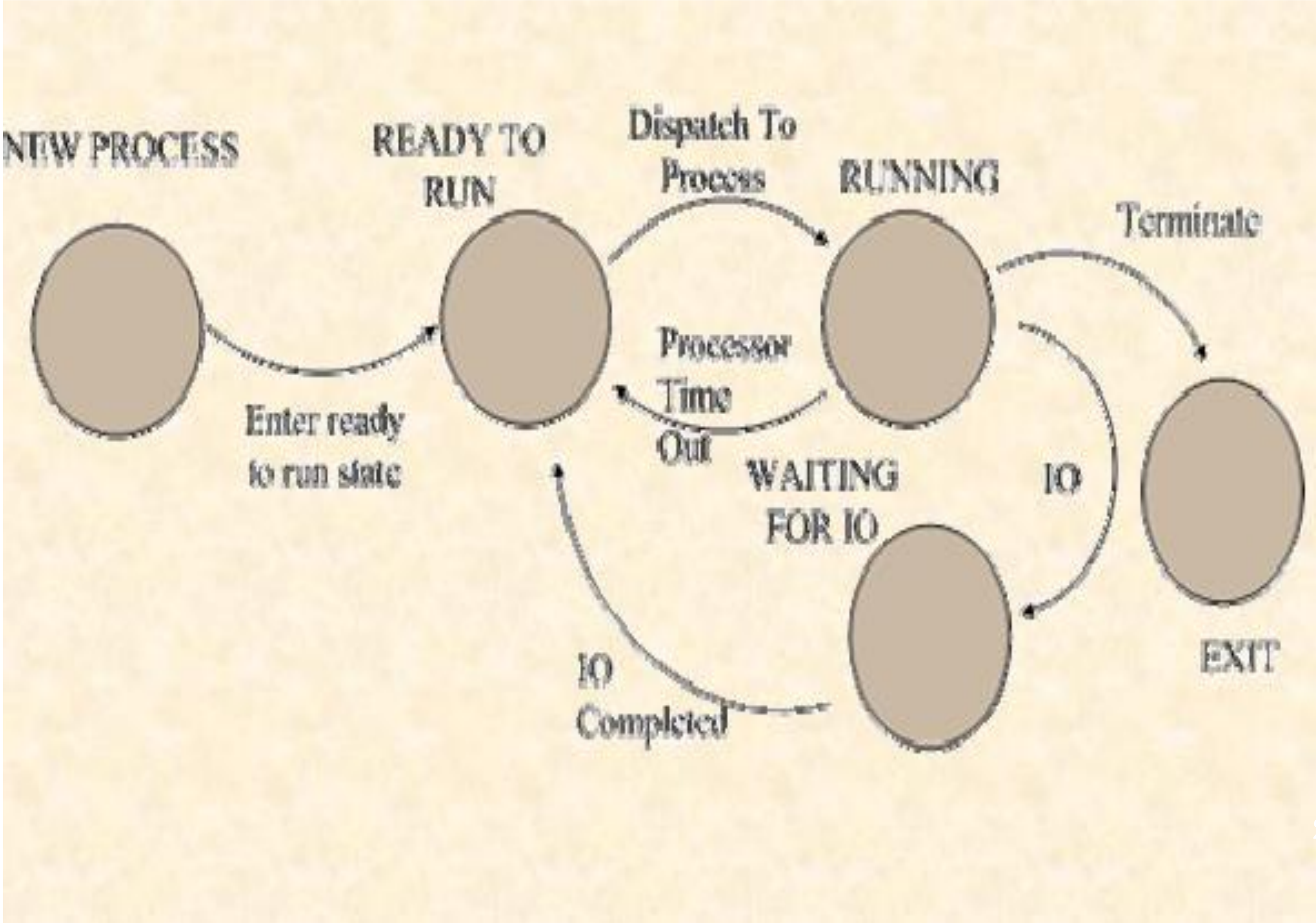
  - *Ready-to-run*

  - *Running*

  - *Waiting-for-IO*

  - *Exit*

Modelling Process States



PROCESS STATE : MANAGEMENT ISSUES

- When a process is created, OS assigns it an ID - pid and creates a data structure to record its progress. The state of the process is now ready-to-run.

- OS has a dispatcher that selects a ready-to-run process and assigns to the processor.

- OS allocates a time slot to run this process.

- OS monitors the progress of every process during its life time.

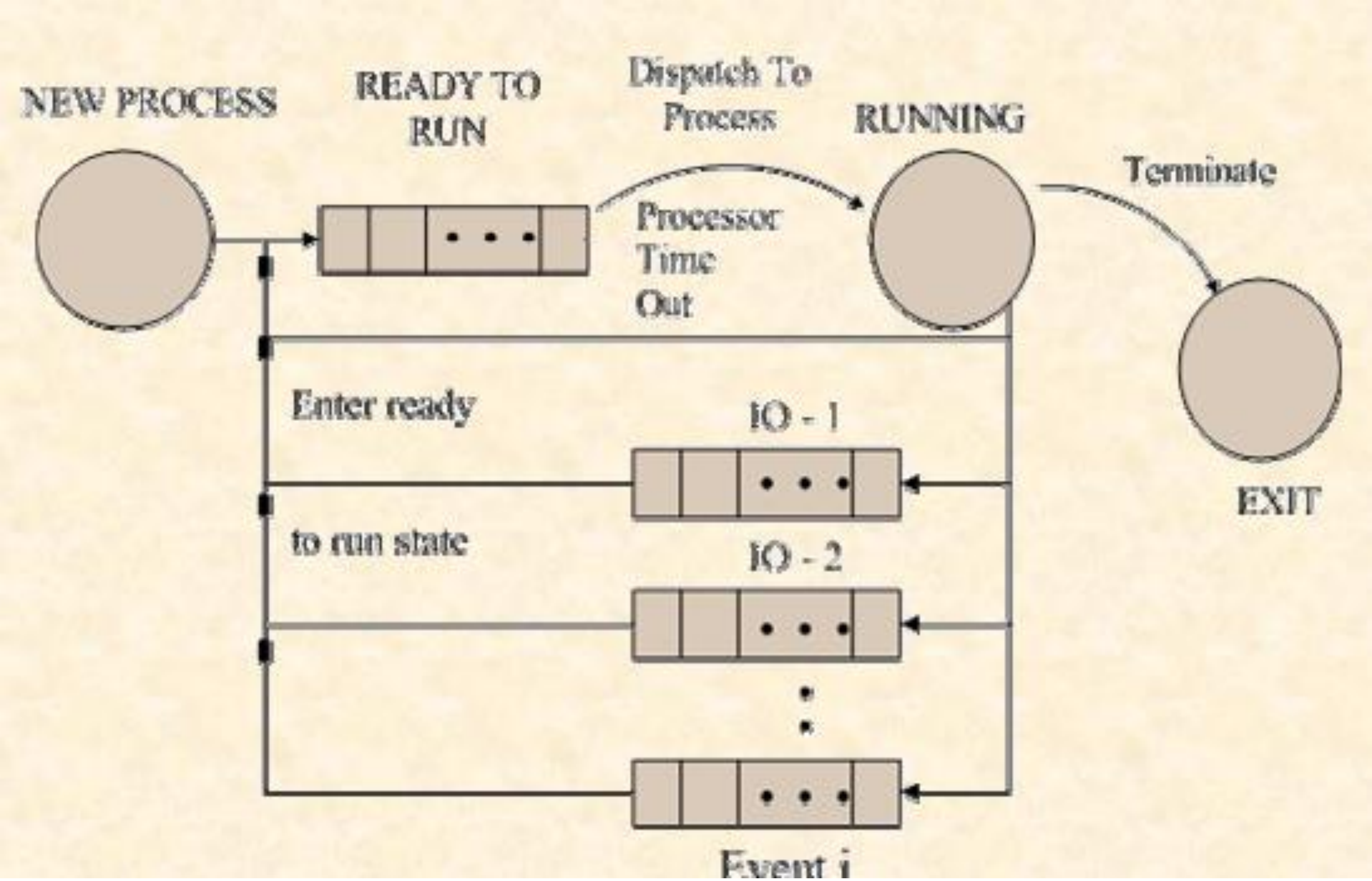
PROCESS STATE : MANAGEMENT ISSUES

- A process may not progress till a certain event occurs - synchronizing signal.
- A process waiting for IO is said to be blocked for IO.
- OS manages all these process migrations between process states.

A Queuing Model

- Data structures are used for process management.
- OS maintains a queue for all ready-to-run processes.
- OS may have separate queue for each of the likely events (including completion of IO).

Queue Based Model



Queue Based Model

- This model helps in the study and analysis of chosen OS policies.
- As an example, Consider the First Come First Served policy for ready-to-run queue.
- To compare this policy with one that prioritizes processes we can study : The average and maximum delays experienced by the lowest priority process.

Queue Based Model

- Comparison of the response times and throughputs in the two cases.

- Processor utilization in the two cases and so on.

Such studies offer new insights - for instance, what level of prioritization leads to starvation.

SCHEDULING CONSIDERATIONS

- OS maintains process data in various queues.

- These queues advance based on scheduling policies :-

  - First Come First Served.
  - Shortest Job First
  - Priority Based Scheduling
  - Batch Processing.

- each policy affects the performance of the overall system.

TYPES OF SCHEDULING

- Pre-emptive Scheduling

  A pre-emptive scheduling allows a higher priority process to replace a currently running process even if it’s time slice is not over or it has not requested for any IO.

- Non pre emptive Scheduling

  A non-pre-emptive scheduling means that a running process retains the control of the CPU & allocated resources, until it releases the CPU either by terminating or by switching to the waiting state.

TYPES OF SCHEDULER

- Short term scheduler(CPU scheduler)

  Selects from among the processes that are ready to execute, & allocates the CPU to one of them.

- Long term scheduler(job scheduler)

  Selects processes from pool which are spooled to mass storage device & loads them in memory for execution.

- Medium term scheduler

  Reduces process from memory & then re-introduce memory with continuing its execution where it was left by technique of swapping.

Choosing a scheduling policy

- Scheduling policy depends on the nature of operations.

- An OS policy may be chosen to suit situations with specific requirements.

- Within a computer system, we need policies to schedule access to processor, memory, disc, IO and shared resource (e.g. printers).

Policy Selection

- A scheduling policy is often determined by a machine’s configuration and its pattern of usage.

- Scheduling is considered in the following context:

  We have only one processor in the system

  We have a multi-programming system – more than one ready-to-run program in memory.

Policy Selection

- We shall study the *effect on the following quality parameters:*

  Response time to users

  Turn around time

  Processor utilization

  Throughput of the system

  Fairness of allocation

  Effect on other resources.

  We see that measures for response time and turn around are *user centered parameters.*

Policy Selection

- Process utilization and throughput are *system centered considerations.*
- *Fairness of allocation and effect on other resources affect both the system and users.*
- An OS performance can be *tuned by choosing an* appropriate scheduling policy.

Comparison of Policies

- Let us consider 5 processes *P1 through P5.*
- We shall make the following assumptions:

  - The jobs have to *run to completion.*

  - *No new jobs arrive till these jobs are processed.*

  - *Time required for each job is known appropriately.*

  - During the *run of jobs there is no suspension for IO*

FCFS(First Come First Served) algorithm

- This is a *Non-Pre-emptive* scheduling algorithm. FCFS strategy assigns to processes in the order in which they request the processor. The process that requests the CPU first is allocated the CPU first. When a process comes in, add its PCB to the tail of ready queue. When running process terminates, next process (PCB) at head of ready queue is selected and run.
- While the FCFS algorithm is easy to implement, it ignores the service time request and all other criteria that may influence the performance with respect to turnaround or waiting time.
- Problem - One process can monopolize CPU
- Solution - Limit the amount of time a process can run without a context switch. This time is called a time slice.

SJF(SHORTEST JOB FIRST) WITH NON PREEMPTION

- Give CPU to the process with the shortest next burst
- If equal, use FCFS
- Can be preemptive or not
- Assumption : Know the length of the next CPU burst of each process in Ready Queue
- Optimal with respect to waiting time!
- Problem - how to know the next burst?

  User specifies (e.g. for batch system)

  Guess/predict based on earlier bursts, using exponential average:

$$\tau_{n+1} = \alpha * t_n + (1 - \alpha) * \tau_n$$

- Solution - Multi-Level Feedback Queques

PRIORITY

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).
- Problem - Starvation (Indefinite Blocking) – low priority processes may never execute.
- Solution - Aging – as time progresses increase the priority of the process.

Comparison of Three Non-Preemptive Scheduling Policies

11/15/2017

	PROCESS NUMBER				
	P1	P2	P3	P4	P5
TIME	20	10	25	15	5

(A)

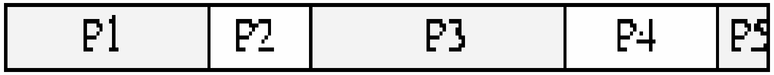
THE PROCESSES FOR PROCESSING

(B)

	TIME TO RESPOND				
	P1	P2	P3	P4	P5
TIME	20	30	55	70	75

FCFS (QUEUE) ORDER : P1, P2, P3, P4. P5

AVERAGE TIME TO COMPLETE : 50



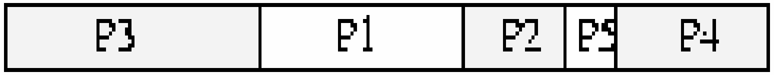
GANTT CHART

	TIME TO RESPOND				
	P3	P1	P2	P5	P4
TIME	25	45	55	60	75

(C)

PRIORITY QUEUE : P3, P1, P2, P5, P4

AVERAGE TIME TO COMPLETE : 52



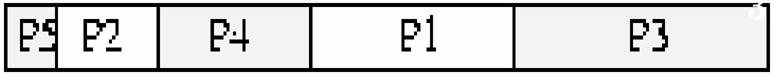
GANTT CHART

(D)

	TIME TO RESPOND				
	P5	P2	P4	P1	P3
TIME	5	15	30	50	75

SHORTEST JOB FIRST : P1, P2, P3, P4. P5

AVERAGE TIME TO COMPLETE : 35



GANTT CHART

Summary of Results

- Case B Average time to complete - 50 (case of FCFS)
- Case C Average time to complete - 52 (case of prioritized)
- Case D Average time to complete – 35 (case of Shortest Job First)
- Clearly it would seem that shortest job first is the best policy. Infact theoretically also this can be proved

ROUND ROBIN(RR)

- FCFS with Preemption

- Each process gets a small unit of CPU time (*time quantum/time slice*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.

- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.

- Ready Queue treated as circular queue

ROUND ROBIN(RR)

- Turnaround time typically larger than SRTF but better response time

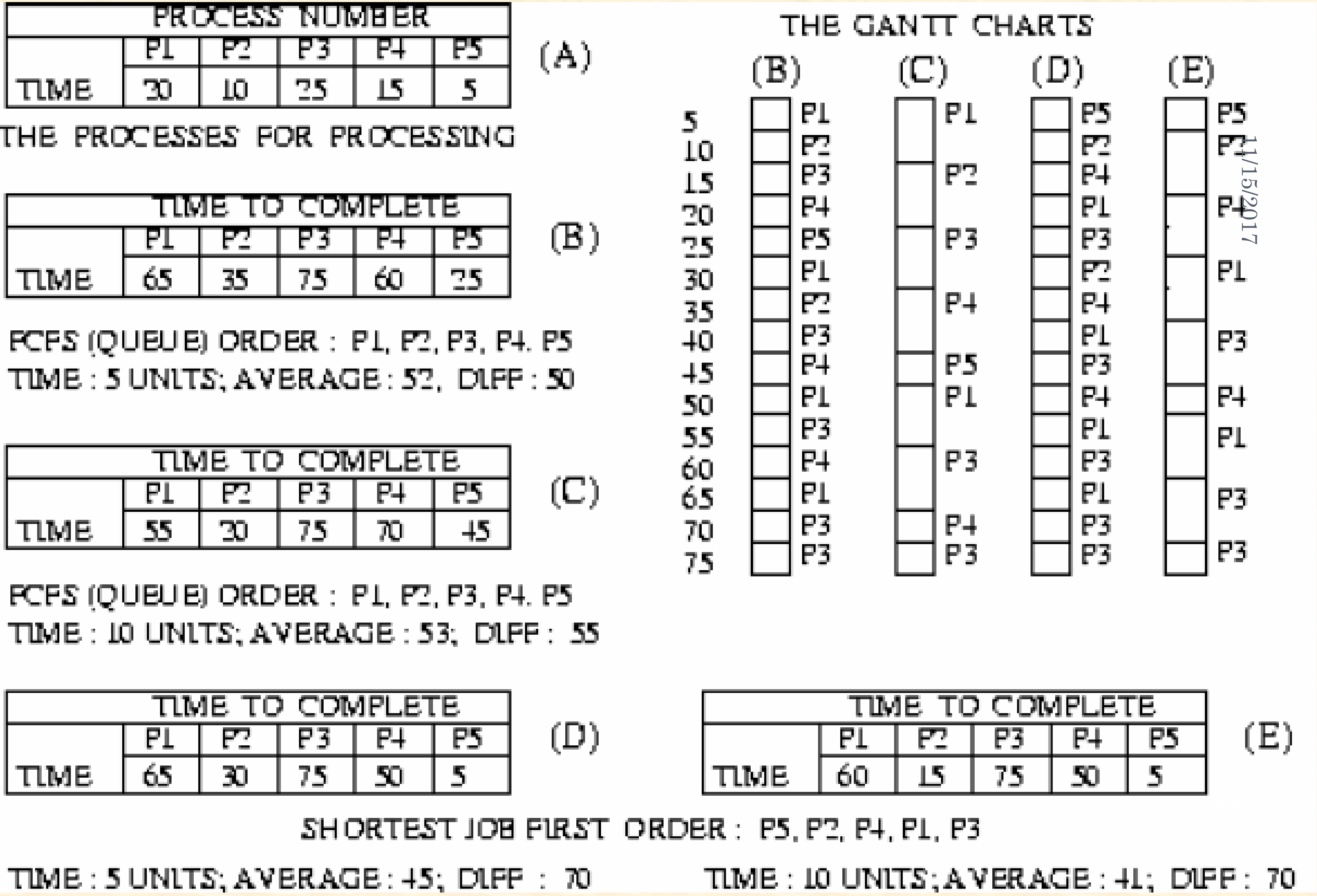
- Problem : Performance depends on quantum *q*

  - Small *q: Overhead due to context switches (& scheduling) so* q should be large with respect to context-switching time

  - Large *q: Behaves like FCFS [*rule of thumb: 80% of bursts should be shorter than *q (also improves turnaround time)]*

- *Solution :* Introduce priority based scheduling(SRTF).

Preemptive Policies



PROCESS NUMBER					
	P1	P2	P3	P4	P5
TIME	30	10	25	15	5

(A)

THE PROCESSES FOR PROCESSING

TIME TO COMPLETE					
	P1	P2	P3	P4	P5
TIME	65	35	75	60	25

(B)

FCFS (QUEUE) ORDER : P1, P2, P3, P4. P5

TIME : 5 UNITS; AVERAGE : 52, DIFF : 50

TIME TO COMPLETE					
	P1	P2	P3	P4	P5
TIME	55	30	75	70	45

(C)

FCFS (QUEUE) ORDER : P1, P2, P3, P4. P5

TIME : 10 UNITS; AVERAGE : 53; DIFF : 55

TIME TO COMPLETE					
	P1	P2	P3	P4	P5
TIME	65	30	75	50	5

(D)

TIME TO COMPLETE					
	P1	P2	P3	P4	P5
TIME	60	15	75	50	5

(E)

THE GANTT CHARTS

	(B)	(C)	(D)	(E)
5	P1	P1	P5	P5
10	P2		P2	P2
15	P3	P2	P4	
20	P4		P1	P4
25	P5	P3	P3	
30	P1		P2	P1
35	P2	P4	P4	
40	P3		P1	P3
45	P4	P5	P3	
50	P1	P1	P4	P4
55	P3		P1	P1
60	P4	P3	P3	
65	P1		P1	P3
70	P3	P4	P3	
75	P3	P3	P3	P3

SHORTEST JOB FIRST ORDER : P5, P2, P4, P1, P3

TIME : 5 UNITS; AVERAGE : 45; DIFF : 70

TIME : 10 UNITS; AVERAGE : 41; DIFF : 70

Preemptive Policies

In the figure, we *compare four cases:*

- *Round-robin allocation with time slice = 5 units*(CASE B)
- *Round-robin allocation with time slice = 10 units* (CASE C)
- *Shortest Job First within the Round-robin; time slice = 5 units (CASE D)*
- *Shortest Job First within the Round-robin; time slice = 10 units. (CASE E)*

Summary of Results

- Case B Average time to complete - 52
- Case C Average time to complete - 53
- Case D Average time to complete – 45
- Case E Average time to complete - 41
- Clearly it would seem that shortest job first is the best policy. Infact theoretically also this can be proved

SJF WITH PREEMPTION (SRTF)

- It was assumed before that all jobs were present initially.

- A more realistic situation is when processes arrive at different times.

- Each job is assumed to arrive with an estimate of time required to complete.

- Considering the estimated remaining time, variation of SJF is designed.

- **Shortest Remaining Time First** : When a process arrives to RQ, sort it in and select the SJF including the running process, possibly interrupting it.

Shortest Remaining Time Schedule

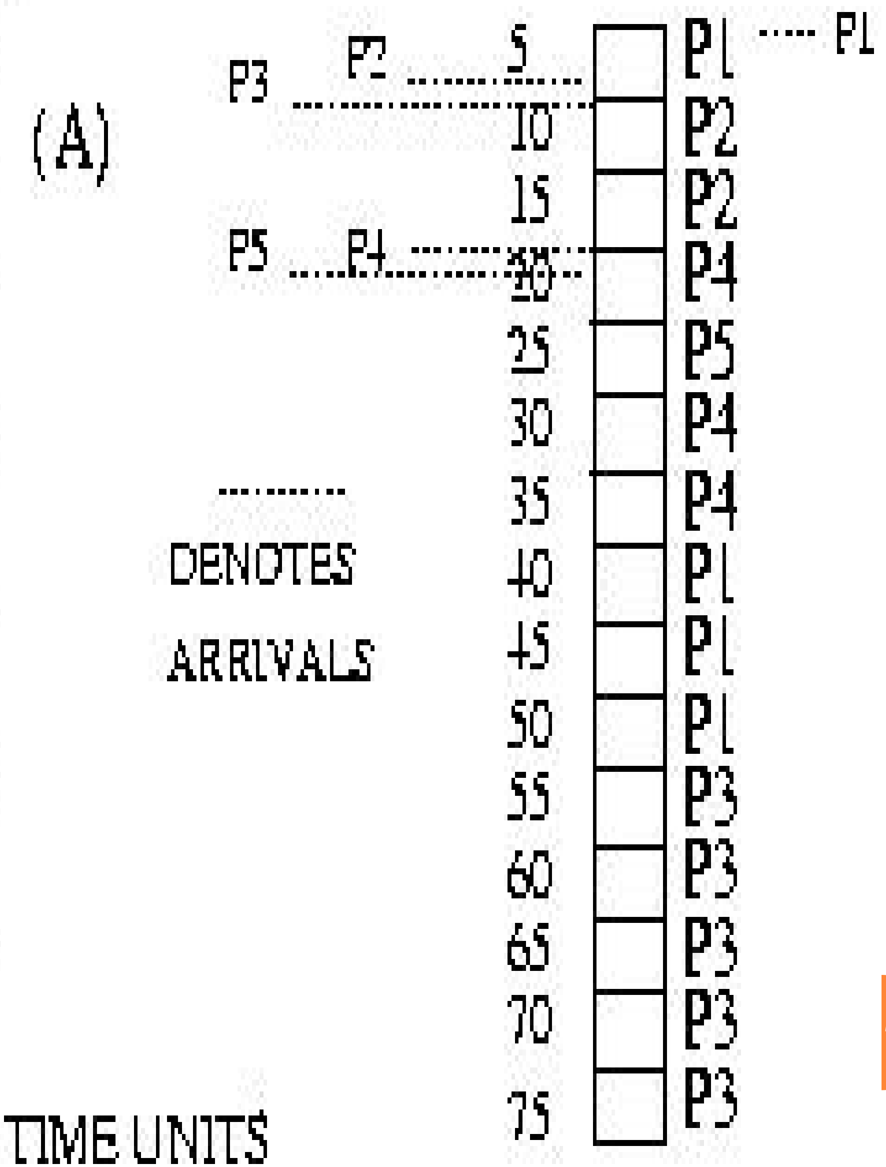
THE PROCESSES FOR PROCESSING

	PROCESS NUMBER				
	P1	P2	P3	P4	P5
TIME	20	10	25	15	5
ARRIVAL	0	3	5	15	17
SERVICE START	0	5	50	15	20
COMPLETED	50	15	75	35	25
DELAY	50	12	70	20	8

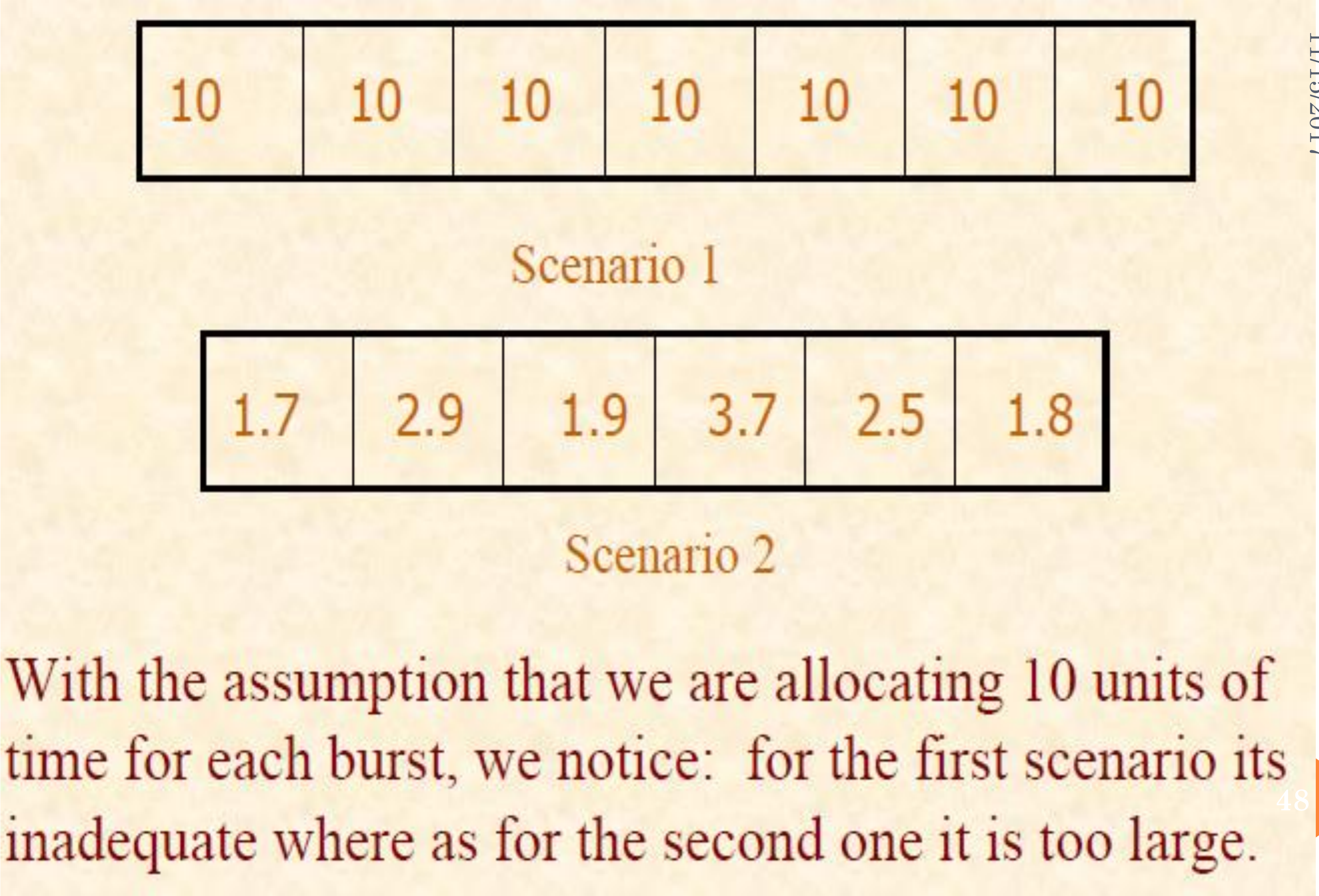
TIME SLICE : 5 UNITS;

AVERAGE TIME TO COMPLETE : 32 TIME UNITS

THE GANTT CHART



How to Estimate Completion Time?



10	10	10	10	10	10	10

Scenario 1

1.7	2.9	1.9	3.7	2.5	1.8

Scenario 2

With the assumption that we are allocating 10 units of time for each burst, we notice: for the first scenario its inadequate where as for the second one it is too large.

How to Estimate Completion Time?

The following *strategies can be observed:*

- Allocate the next larger time slice to the time actually used.

- Allocate the average over the last several time slice utilizations. It gives all previous utilizations equal weightages to find the next time slice allocation.

- Use the entire history but give lower weightages to the utilization in past (Exponential Averaging technique).

Exponential Averaging Technique

- We denote our current, nth, CPU usage burst by tn. Also, we denote the average of all past usage bursts up to now by.

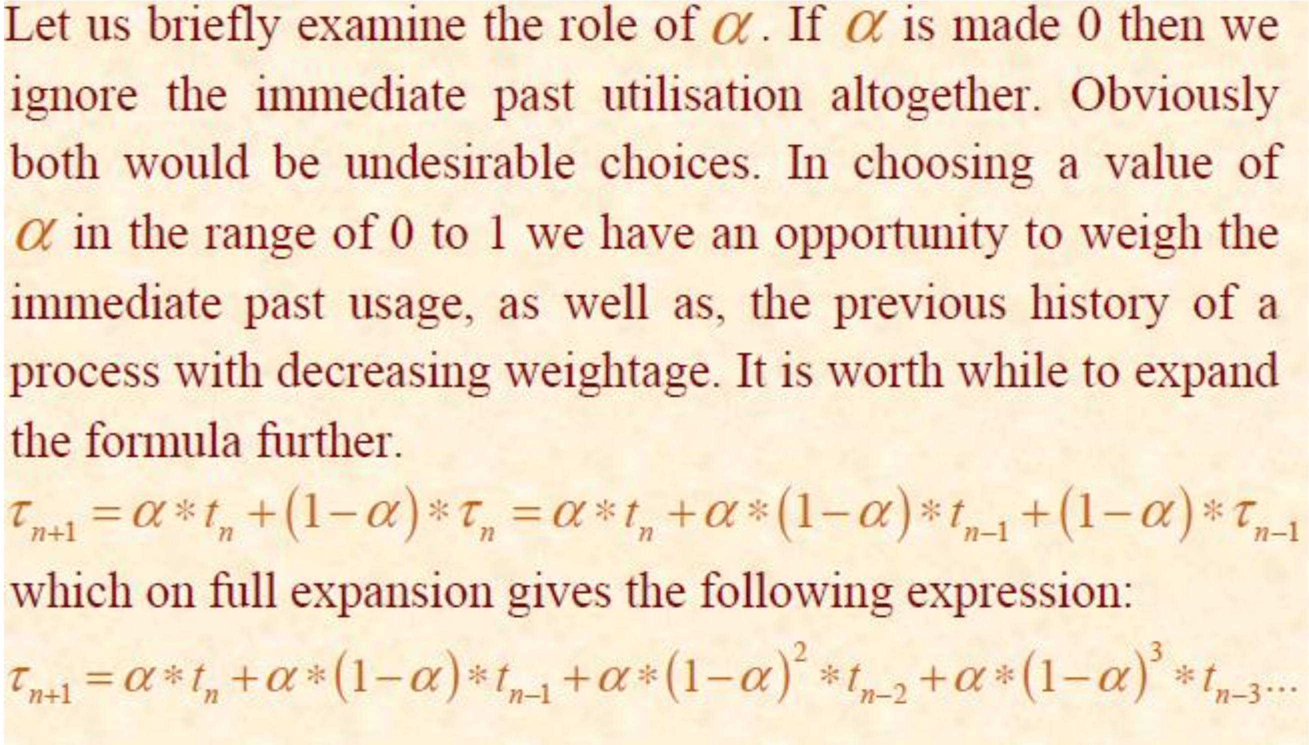
- Using a weighting factor 0 ≤α ≤ n with tn and 1- A and with , we estimate the next CPU usage burst.

- The predicted value of is computed as :

$$\tau_{n+1} = \alpha * t_n + (1 - \alpha) * \tau_n$$

This formula is called an exponential averaging formula.

EXPONENTIAL AVERAGING TECHNIQUE



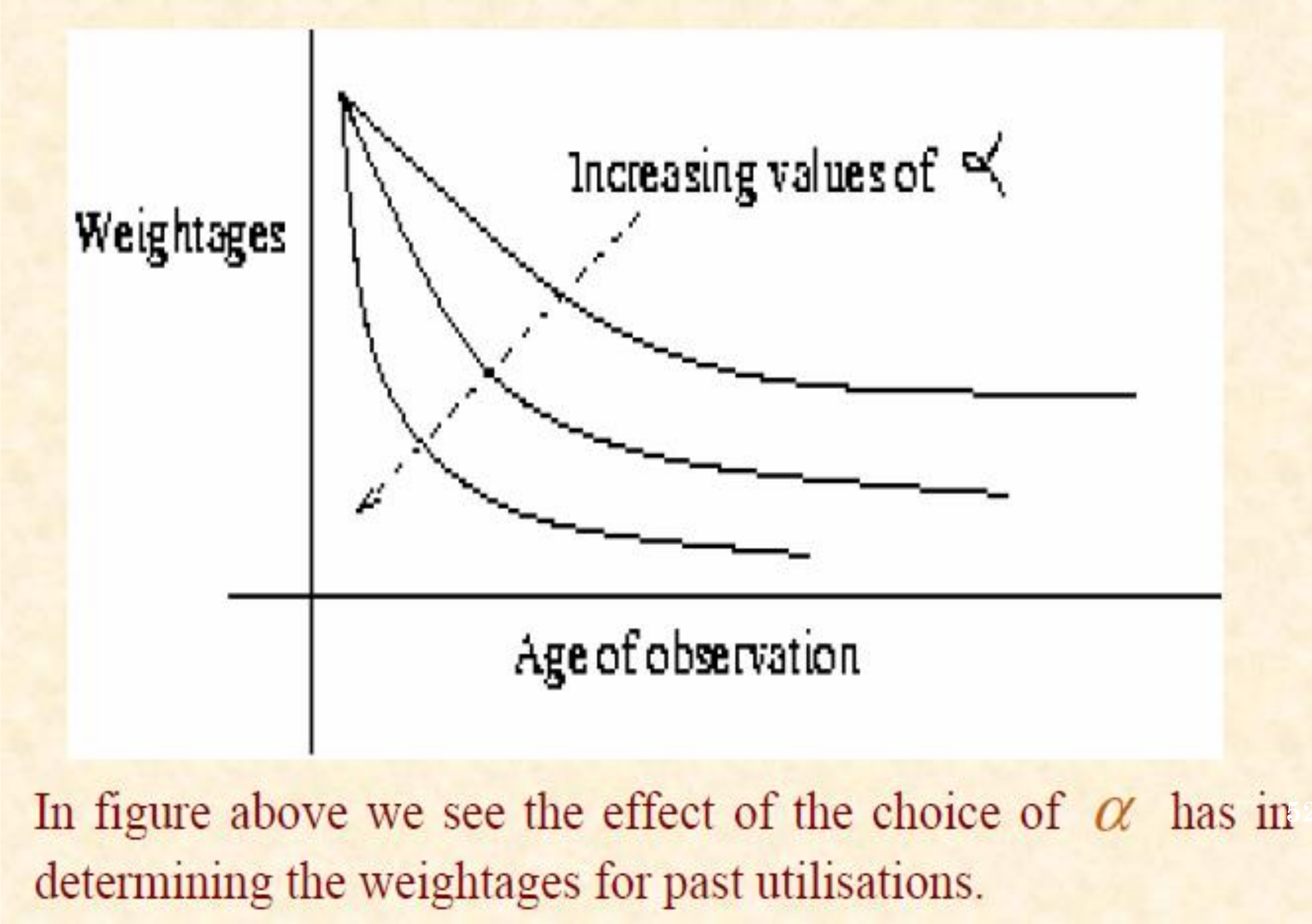
Let us briefly examine the role of α. If α is made 0 then we ignore the immediate past utilisation altogether. Obviously both would be undesirable choices. In choosing a value of α in the range of 0 to 1 we have an opportunity to weigh the immediate past usage, as well as, the previous history of a process with decreasing weightage. It is worth while to expand the formula further.

$$\tau_{n+1} = \alpha * t_n + (1-\alpha) * \tau_n = \alpha * t_n + \alpha * (1-\alpha) * t_{n-1} + (1-\alpha) * \tau_{n-1}$$

which on full expansion gives the following expression:

$$\tau_{n+1} = \alpha * t_n + \alpha * (1-\alpha) * t_{n-1} + \alpha * (1-\alpha)^2 * t_{n-2} + \alpha * (1-\alpha)^3 * t_{n-3} \ldots$$

Exponential Averaging Technique



In figure above we see the effect of the choice of α has in determining the weightages for past utilisations.

PROCESS CONTEXT SWITCHING

- OS maintains a lot of information about the resources used by a running process.

- The information stored establishes the context for the process. Usually the following is stored.

  - The program computed.
  - The values in various registers.
  - The process states etc.

PROCESS CONTEXT SWITCHING

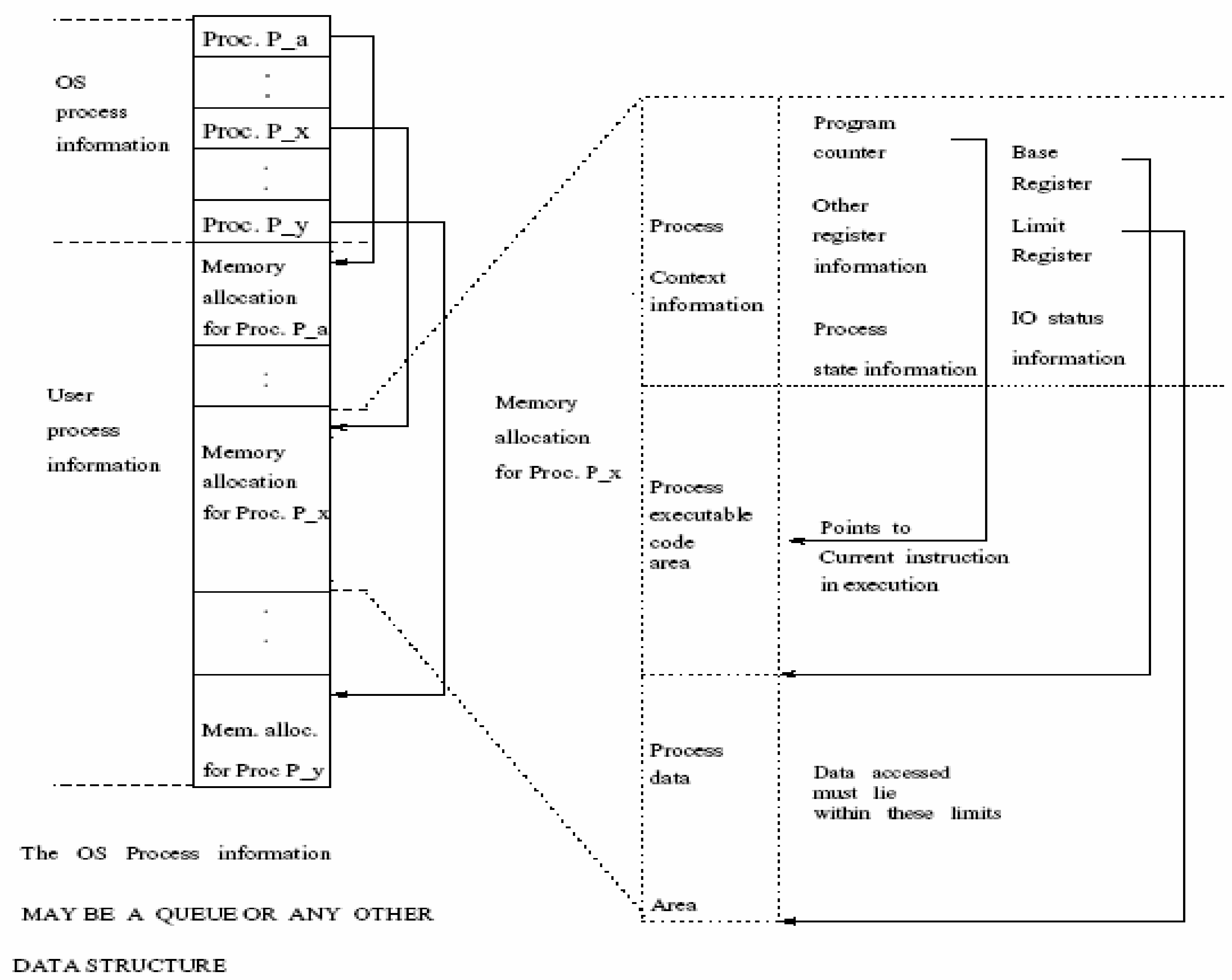
- When a process is switched the context information needs to be changed as follows :--

  - For the outgoing process: Store the information of the current process in the some area of memory

  - For incoming process: Copy the previously stored information from memory.

- The information context that is switched is illustrated in the figure in the following slide.

Process Context Switching



11/15/2017

Process Context Switching

An OS maintains, and keeps updating, a lot of information about the resources in use for a running process. For instance, each process in execution uses the program counter, registersand other resources within the CPU. So, whenever a process is switched, the OS moves out, and brings in, considerable amount of context switching information as shown in the previous figure. We see that process Px is currently executing (note that the program counter is pointing in executable code area of Px).

Process Context Switching

- Let us now switch the context in favor of running process Py.

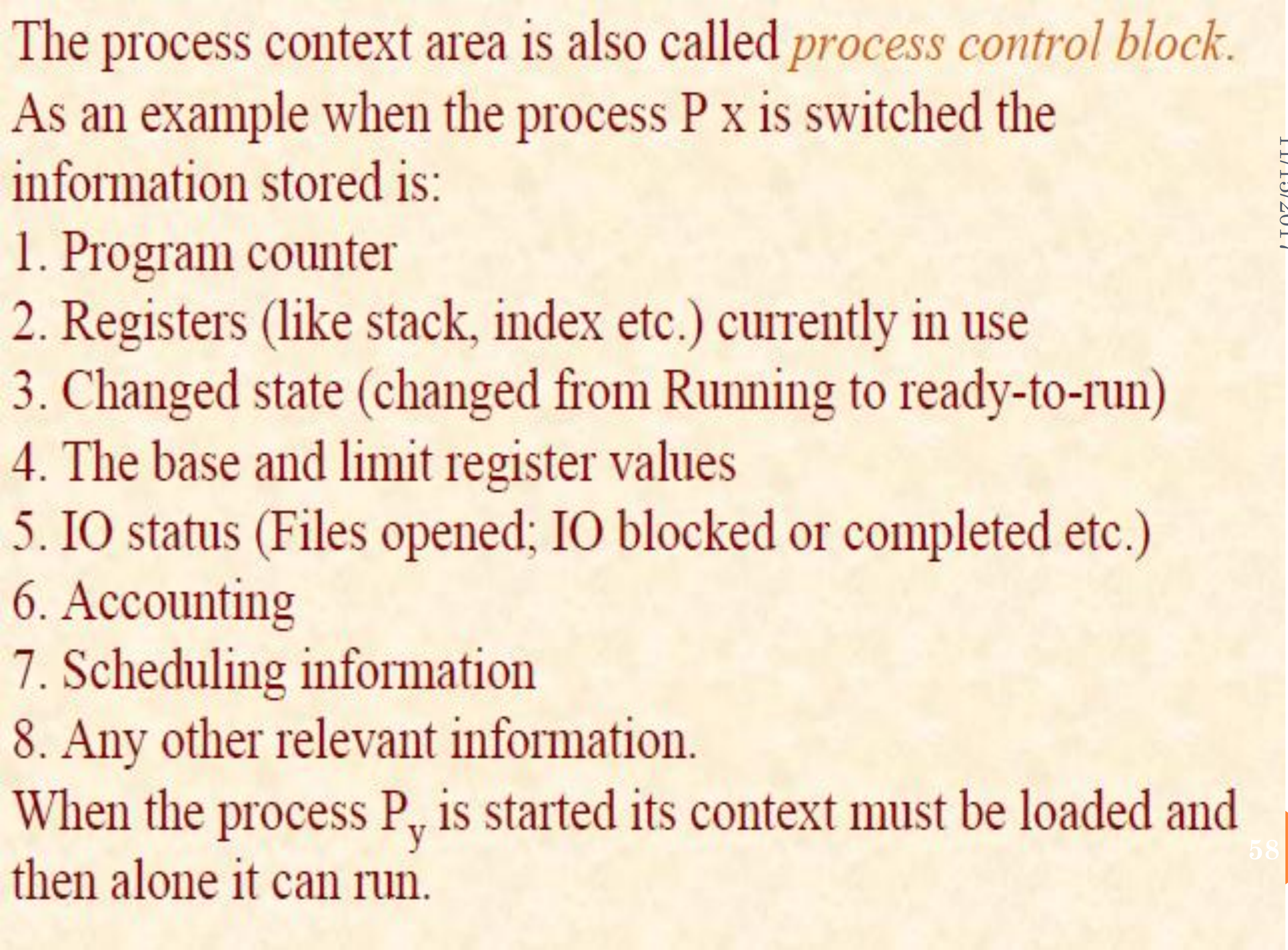
- The following must happen:--

  - All the current context information about process Px must be updated in its own context area.

  - All context information about process Py must be downloaded in its own context area.

  - The program counter should have an address value to an instruction of process Py. and process Py must be now marked as running.

Process Context Switching



The process context area is also called *process control block.*

As an example when the process P x is switched the information stored is:

1. Program counter
2. Registers (like stack, index etc.) currently in use
3. Changed state (changed from Running to ready-to-run)
4. The base and limit register values
5. IO status (Files opened; IO blocked or completed etc.)
6. Accounting
7. Scheduling information
8. Any other relevant information.

When the process P_y is started its context must be loaded and then alone it can run.

UNIX PROCESS STATE INFORMATION

- Command used to obtain information about process is ps
- Options

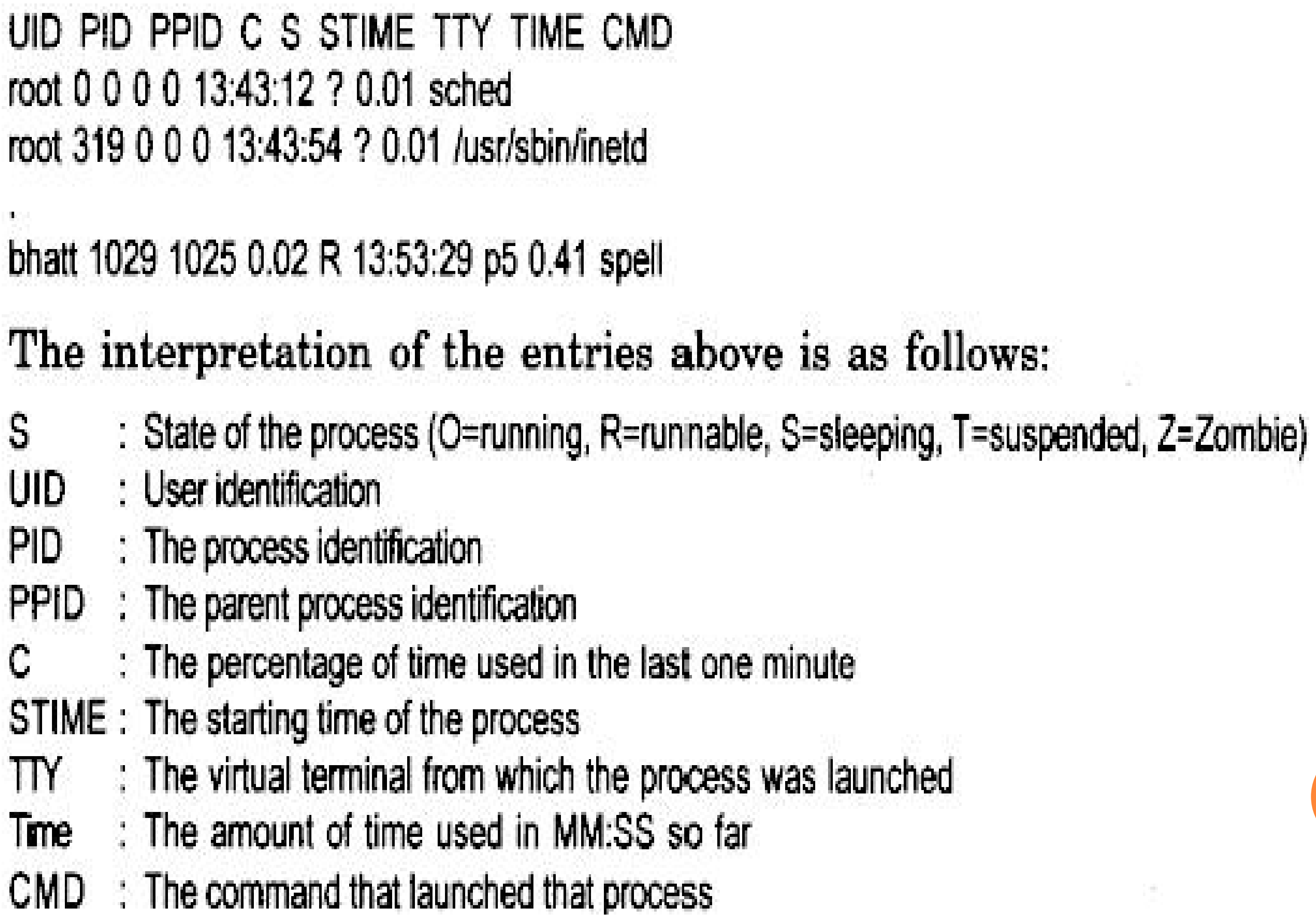
  -e gets information on all running processes

  -f

  -l long listing

UNIX Process state information

$ ps –ef



```
UID PID PPID C S STIME TTY TIME CMD
root 0 0 0 0 13:43:12 ? 0.01 sched
root 319 0 0 0 13:43:54 ? 0.01 /usr/sbin/inetd

.
bhatt 1029 1025 0.02 R 13:53:29 p5 0.41 spell
```

The interpretation of the entries above is as follows:

S : State of the process (O=running, R=runnable, S=sleeping, T=suspended, Z=Zombie)
UID : User identification
PID : The process identification
PPID : The parent process identification
C : The percentage of time used in the last one minute
STIME : The starting time of the process
TTY : The virtual terminal from which the process was launched
Time : The amount of time used in MM:SS so far
CMD : The command that launched that process