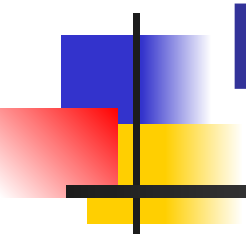
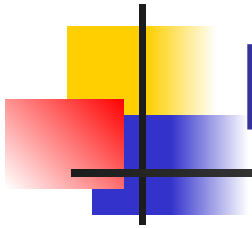


Lecture 3

CPU Scheduling



Lecture Highlights

- Introduction to CPU scheduling
  - What is CPU scheduling
  - Related Concepts of Starvation, Context Switching and Preemption
- Scheduling Algorithms
- Parameters Involved
- Parameter-Performance Relationships
- Some Sample Results

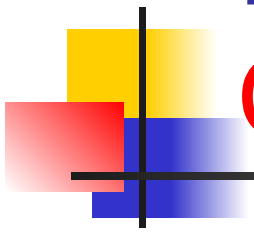
Introduction
What is CPU scheduling

- **An operating system must select processes for execution in some fashion.**
- **CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.**
- **The selection process is carried out by an appropriate scheduling algorithm.**

Related Concepts
Starvation

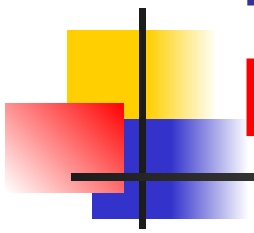
- This is the situation where a process waits endlessly for CPU attention.
- As a result of starvation, the starved process may never complete its designated task.

Related Concepts
Context Switch

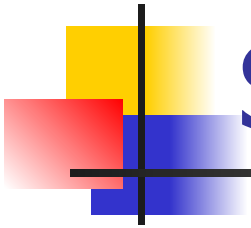


- Switching the CPU to another process requires saving the state of the old process and loading the state of the new process. This task is known as a context switch.
- Increased context switching affects performance adversely because the CPU spends more time switching between tasks than it does with the tasks itself.

Related Concepts
Pre-emption



- Preemption involves the CPU purging a process being served in favor of another process.
- The new process may be favored due to a variety of reasons like higher priority, smaller execution time, etc as compared to the currently executing process.



Scheduling Algorithms

- Scheduling algorithms deal with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.
- Some commonly used scheduling algorithms:
  - First In First Out (FIFO)
  - Shortest Job First (SJF) without preemption
  - Preemptive SJF
  - Priority based without preemption
  - Preemptive Priority base
  - Round robin
  - Multilevel Feedback Queue

Scheduling algorithms
A process-mix

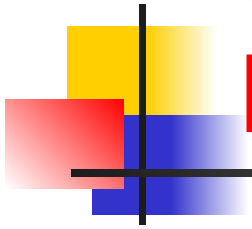
- When studying scheduling algorithms, we take a set of processes and their parameters and calculate certain performance measures
- This set of processes, also termed a process-mix, is comprised of some of the PCB parameters.

Scheduling algorithms
A sample process-mix

- Following is a sample process-mix which we'll use to study the various scheduling algorithms.

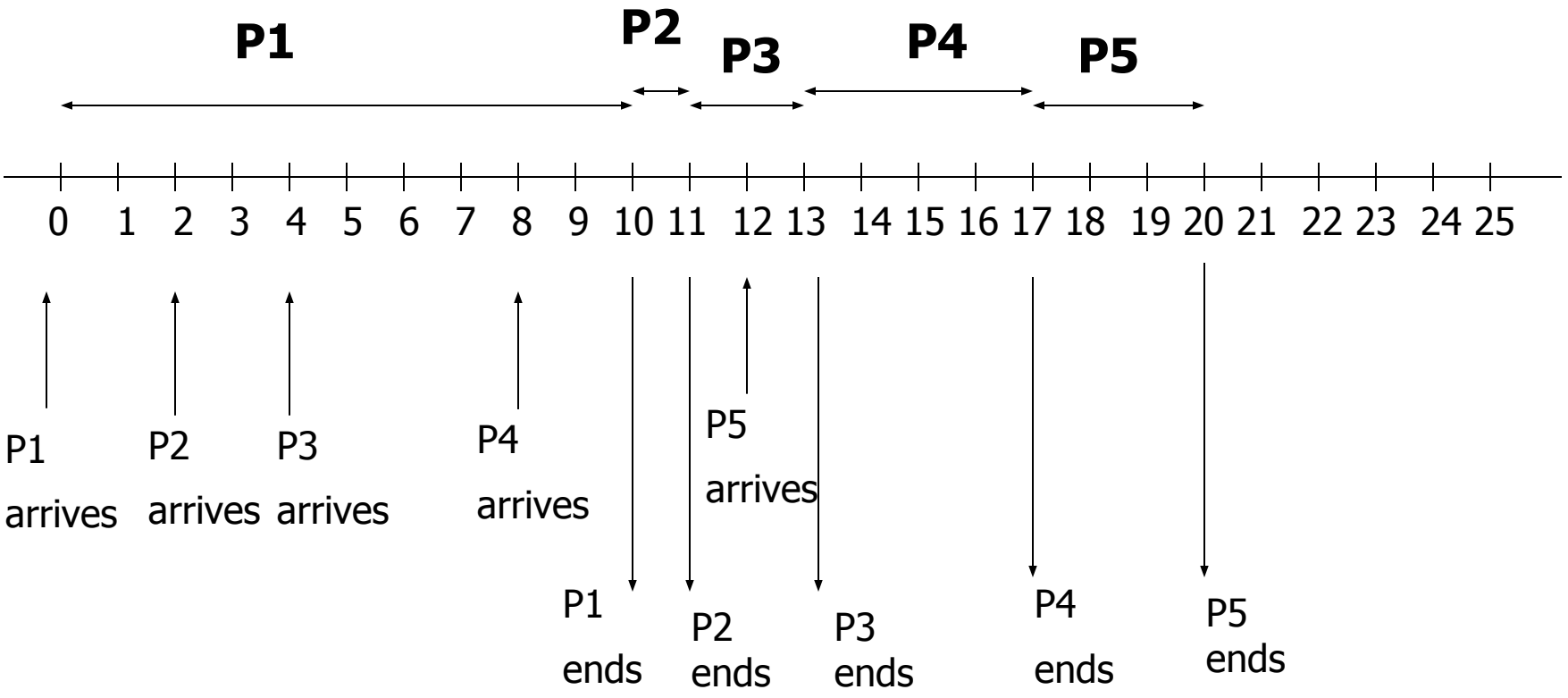
Process ID	Time of Arrival	Priority	Execution Time
1	0	20	10
2	2	10	1
3	4	58	2
4	8	40	4
5	12	30	3

Scheduling Algorithms
First In First Out (FIFO)

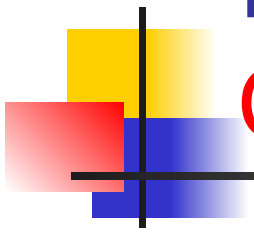


- This is the simplest algorithm and processes are served in order in which they arrive.
- The timeline on the following slide should give you a better idea of how the FIFO algorithm works.

Scheduling Algorithms
First In First Out (FIFO)



FIFO Scheduling Algorithm
Calculating Performance Measures



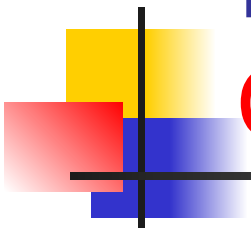
Turnaround Time = End Time – Time of Arrival

- Turnaround Time for P1 = 10 – 0 = 10
- Turnaround Time for P2 = 11 – 2 = 9
- Turnaround Time for P3 = 13 – 4 = 9
- Turnaround Time for P4 = 17 – 8 = 9
- Turnaround Time for P5 = 20 – 12 = 8

Total Turnaround Time = 10+9+9+9+8 = 45

Average Turnaround Time = 45/5 = 9

FIFO Scheduling Algorithm
Calculating Performance Measures

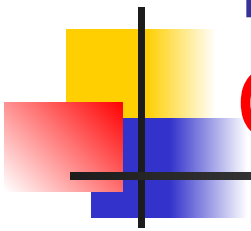


Waiting Time = Turnaround Time – Execution Time

- Waiting Time for P1 = 10 – 10 = 0
- Waiting Time for P2 = 9 – 1 = 8
- Waiting Time for P3 = 9 – 2 = 7
- Waiting Time for P4 = 9 – 4 = 5
- Waiting Time for P5 = 8 – 3 = 5

Total Waiting Time = 0+8+7+5+5 = 25

Average Waiting Time = 25/5 = 5



FIFO Scheduling Algorithm

Calculating Performance Measures

Throughput

Total time for completion of 5 processes = 20

Therefore, Throughput = 5/20

= 0.25 processes per unit time

FIFO Scheduling Algorithm

Pros and cons

Pros

- FIFO is an easy algorithm to understand and implement.

Cons

- The average waiting time under the FIFO scheme is not minimal and may vary substantially if the process execution times vary greatly.
- The CPU and device utilization is low.
- It is troublesome for time-sharing systems.

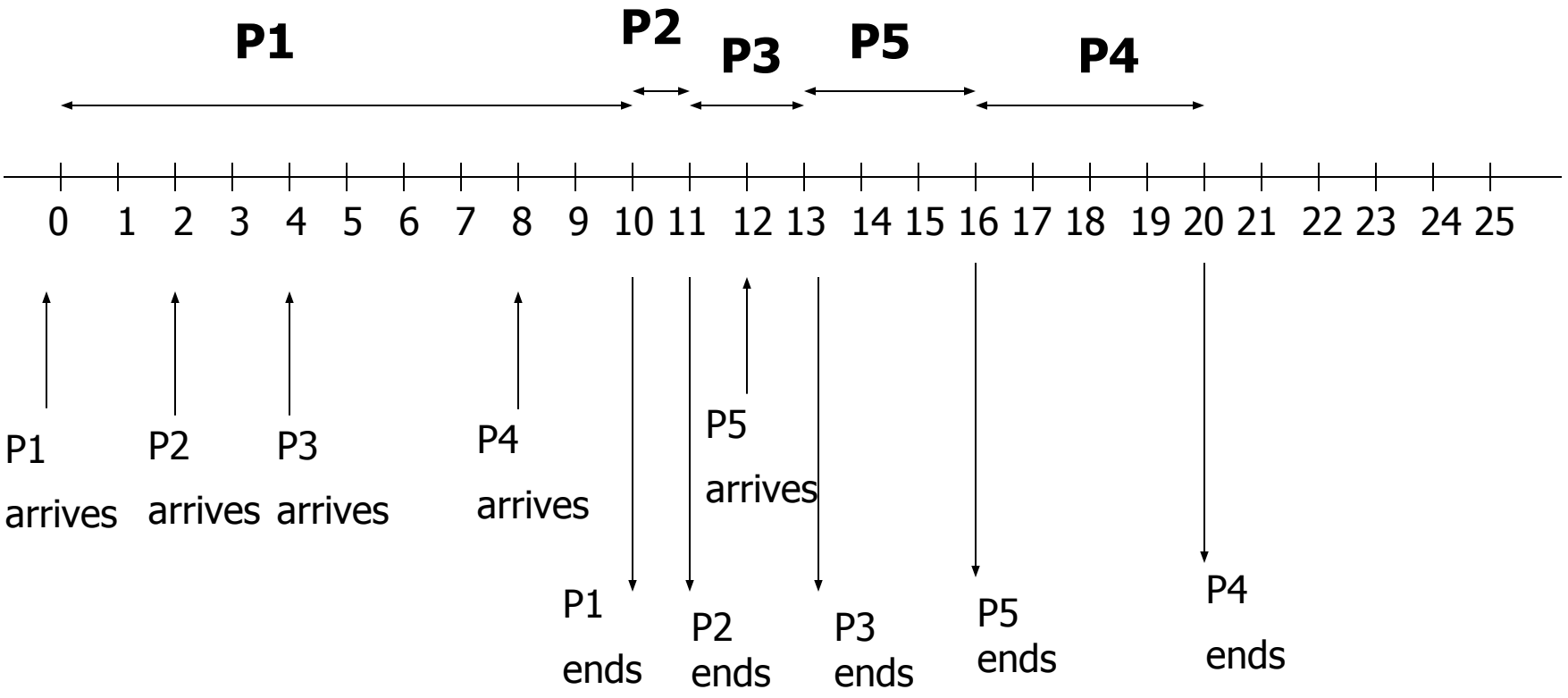
Scheduling Algorithms

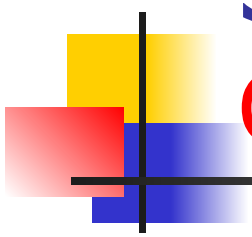
Shortest Job First (SJF) without preemption

- The CPU chooses the shortest job available in its queue and processes it to completion. If a shorter job arrives during processing, the CPU does not preempt the current process in favor of the new process.
- The timeline on the following slide should give you a better idea of how the SJF algorithm without preemption works.

Scheduling Algorithms

Shortest Job First (SJF) without preemption





SJF without preemption
Calculating Performance Measures

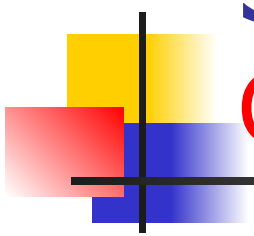
Turnaround Time = End Time – Time of Arrival

- Turnaround Time for P1 = 10 – 0 = 10
- Turnaround Time for P2 = 11 – 2 = 9
- Turnaround Time for P3 = 13 – 4 = 9
- Turnaround Time for P4 = 20 – 8 = 12
- Turnaround Time for P5 = 16 – 12 = 4

Total Turnaround Time = 10+9+9+12+4 = 44

Average Turnaround Time = 44/5 = 8.8

SJF without preemption
Calculating Performance Measures



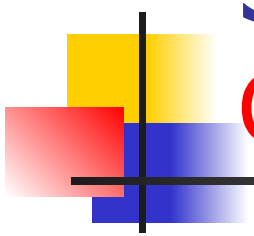
Waiting Time = Turnaround Time – Execution Time

- Waiting Time for P1 = 10 – 10 = 0
- Waiting Time for P2 = 9 – 1 = 8
- Waiting Time for P3 = 9 – 2 = 7
- Waiting Time for P4 = 12 – 4 = 8
- Waiting Time for P5 = 4 – 3 = 1

Total Waiting Time = 0+8+7+8+1 = 24

Average Waiting Time = 24/5 = 4.8

SJF without preemption
Calculating Performance Measures



Throughput

Total time for completion of 5 processes = 20

Therefore, Throughput = 5/20

= 0.25 processes per unit time

SJF without preemption
Pros and cons

Pros

- The SJF scheduling is provably optimal, in that it gives the minimum average waiting time for a given set of processes.

Cons

- Since there is no way of knowing execution times, the same need to be estimated. This makes the implementation more complicated.
- The scheme suffers from possible starvation.

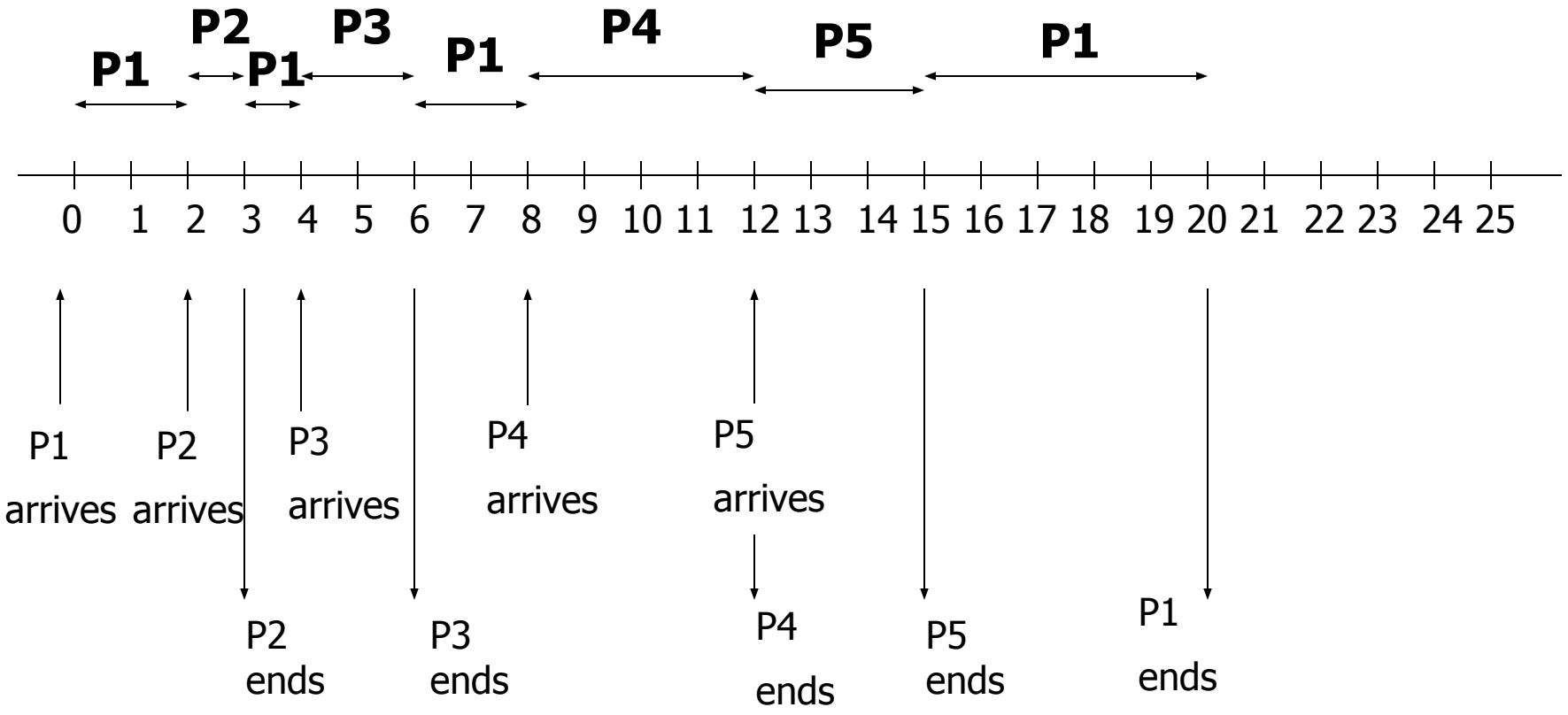
Scheduling Algorithms

Shortest Job First (SJF) with preemption

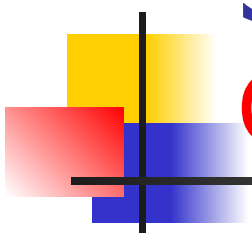
- The CPU chooses the shortest job available in its queue and begins processing it. If a shorter job arrives during processing, the CPU preempts the current process in favor of the new process.Execution times are compared by time remaining and not total execution time.
- The timeline on the following slide should give you a better idea of how the SJF algorithm with preemption works.

Scheduling Algorithms

Shortest Job First (SJF) with preemption



SJF with preemption
Calculating Performance Measures

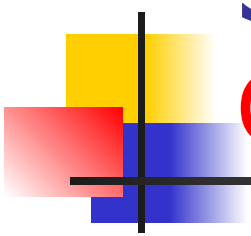


Turnaround Time = End Time – Time of Arrival

- Turnaround Time for P1 = 20 – 0 = 20
- Turnaround Time for P2 = 3 – 2 = 1
- Turnaround Time for P3 = 6 – 4 = 2
- Turnaround Time for P4 = 12 – 8 = 4
- Turnaround Time for P5 = 15 – 12 = 3

Total Turnaround Time = 20+1+2+4+3 = 30

Average Turnaround Time = 30/5 = 6



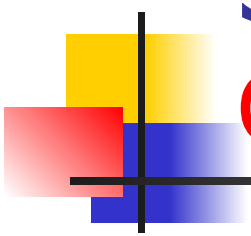
SJF with preemption
Calculating Performance Measures

Waiting Time = Turnaround Time – Execution Time

- Waiting Time for P1 = 20 – 10 = 10
- Waiting Time for P2 = 1 – 1 = 0
- Waiting Time for P3 = 2 – 2 = 0
- Waiting Time for P4 = 4 – 4 = 0
- Waiting Time for P5 = 3 – 3 = 0

Total Waiting Time = 10+0+0+0+0 = 10

Average Waiting Time = 10/5 = 2



SJF with preemption

Calculating Performance Measures

Throughput

Total time for completion of 5 processes = 20

Therefore, Throughput = 5/20

= 0.25 processes per unit time

SJF with preemption
Pros and cons

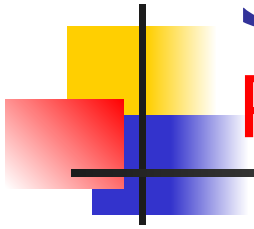
Pros

- Preemptive SJF scheduling further reduces the minimum average waiting time for a given set of processes.

Cons

- Since there is no way of knowing execution times, the same need to be estimated. This makes the implementation more complicated.
- Like the non-preemptive counterpart, the scheme suffers from possible starvation.
- Too much preemption may result in higher context switching times adversely affecting system performance.

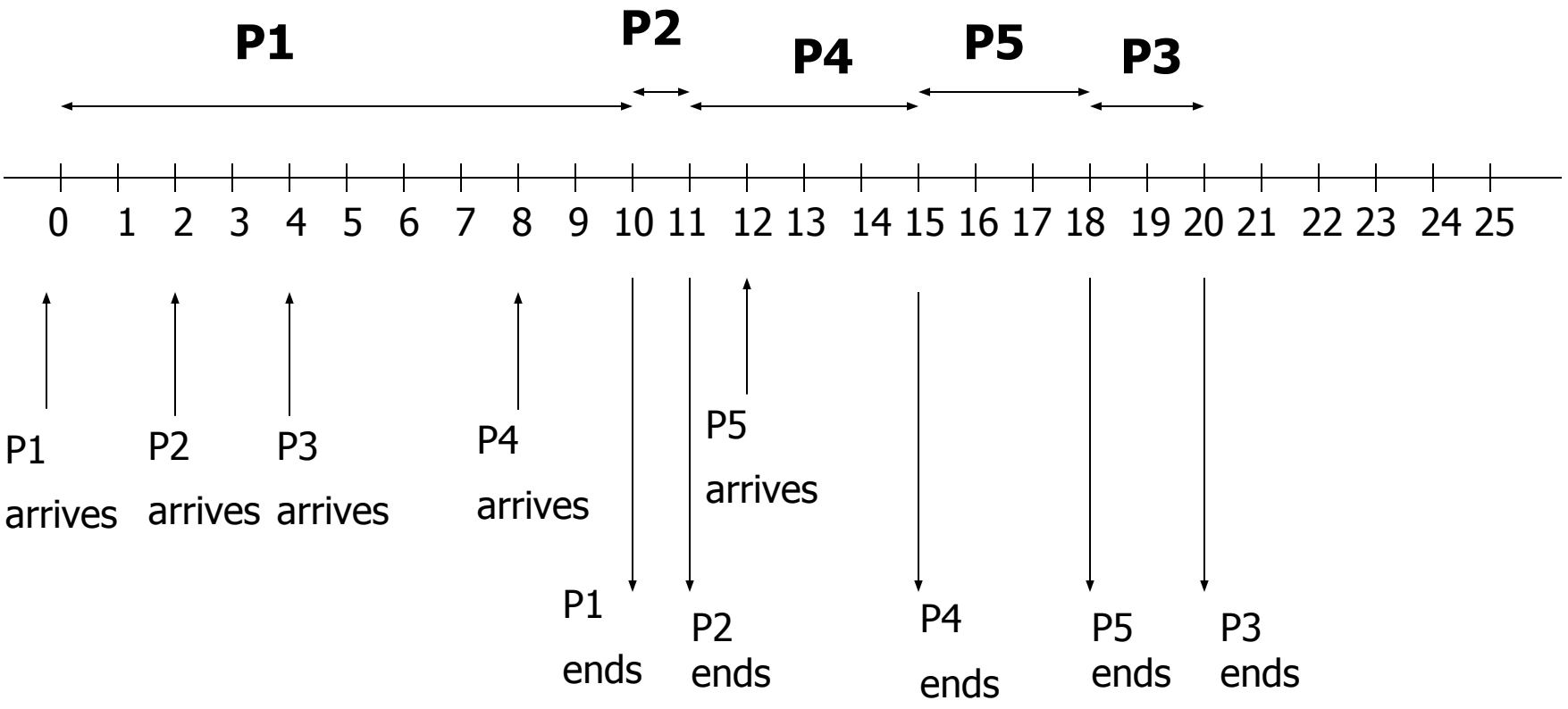
Scheduling Algorithms
Priority based without preemption



- The CPU chooses the highest priority job available in its queue and processes it to completion. If a higher priority job arrives during processing, the CPU does not preempt the current process in favor of the new process.
- The timeline on the following slide should give you a better idea of how the priority based algorithm without preemption works.

Scheduling Algorithms

Priority based without preemption



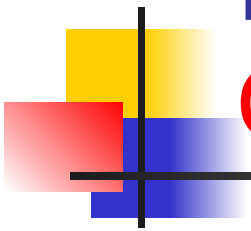
Priority based without preemption
Calculating Performance Measures

Turnaround Time = End Time – Time of Arrival

- Turnaround Time for P1 = 10 – 0 = 10
- Turnaround Time for P2 = 11 – 2 = 9
- Turnaround Time for P3 = 20 – 4 = 16
- Turnaround Time for P4 = 15 – 8 = 7
- Turnaround Time for P5 = 18 – 12 = 6

Total Turnaround Time = 10+9+16+7+6 = 48

Average Turnaround Time = 48/5 = 9.6



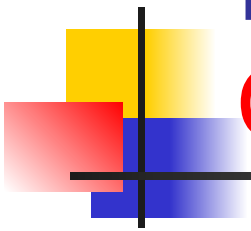
Priority based without preemption
Calculating Performance Measures

Waiting Time = Turnaround Time – Execution Time

- Waiting Time for P1 = 10 – 10 = 0
- Waiting Time for P2 = 9 – 1 = 8
- Waiting Time for P3 = 16 – 2 = 14
- Waiting Time for P4 = 7 – 4 = 3
- Waiting Time for P5 = 6 – 3 = 3

Total Waiting Time = 0+8+14+3+3 = 28

Average Waiting Time = 28/5 = 5.6



Priority based without preemption
Calculating Performance Measures

Throughput

Total time for completion of 5 processes = 20

Therefore, Throughput = 5/20

= 0.25 processes per unit time

Priority based without preemption
Pros and cons

Performance measures in a priority scheme have no meaning.

Cons

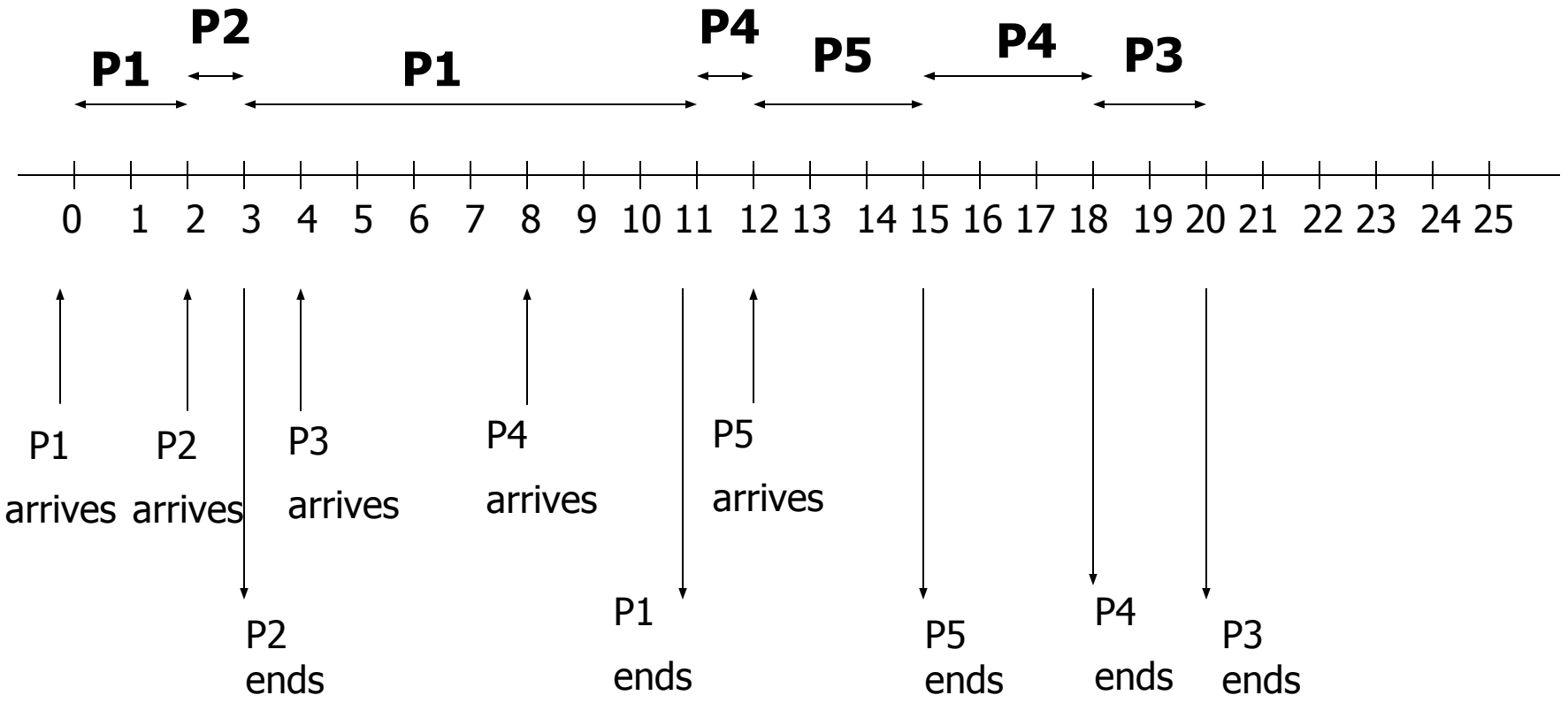
- A major problem with priority based algorithms is indefinite blocking or starvation.

Scheduling Algorithms
Priority based with preemption

- The CPU chooses the job with highest priority available in its queue and begins processing it. If a job with higher priority arrives during processing, the CPU preempts the current process in favor of the new process.
- The timeline on the following slide should give you a better idea of how the priority based algorithm with preemption works.

Scheduling Algorithms

Priority based with preemption



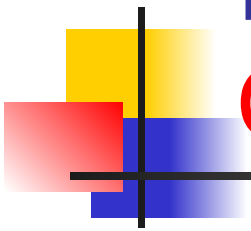
Priority based with preemption
Calculating Performance Measures

Turnaround Time = End Time – Time of Arrival

- Turnaround Time for P1 = 11 – 0 = 11
- Turnaround Time for P2 = 3 – 2 = 1
- Turnaround Time for P3 = 20 – 4 = 16
- Turnaround Time for P4 = 18 – 8 = 10
- Turnaround Time for P5 = 15 – 12 = 3

Total Turnaround Time = 11+1+16+10+3 = 41

Average Turnaround Time = 41/5 = 8.2



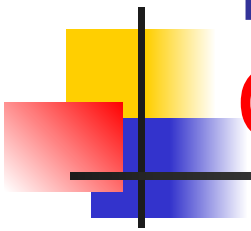
Priority based with preemption
Calculating Performance Measures

Waiting Time = Turnaround Time – Execution Time

- Waiting Time for P1 = 11 – 10 = 1
- Waiting Time for P2 = 1 – 1 = 0
- Waiting Time for P3 = 16 – 2 = 14
- Waiting Time for P4 = 10 – 4 = 6
- Waiting Time for P5 = 3 – 3 = 0

Total Waiting Time = 1+0+14+6+0 = 21

Average Waiting Time = 21/5 = 4.2



Priority based with preemption
Calculating Performance Measures

Throughput

Total time for completion of 5 processes = 20

Therefore, Throughput = 5/20

= 0.25 processes per unit time

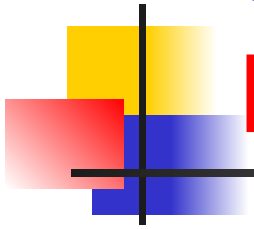
Priority based with preemption
Pros and cons

Performance measures in a priority scheme have no meaning.

Cons

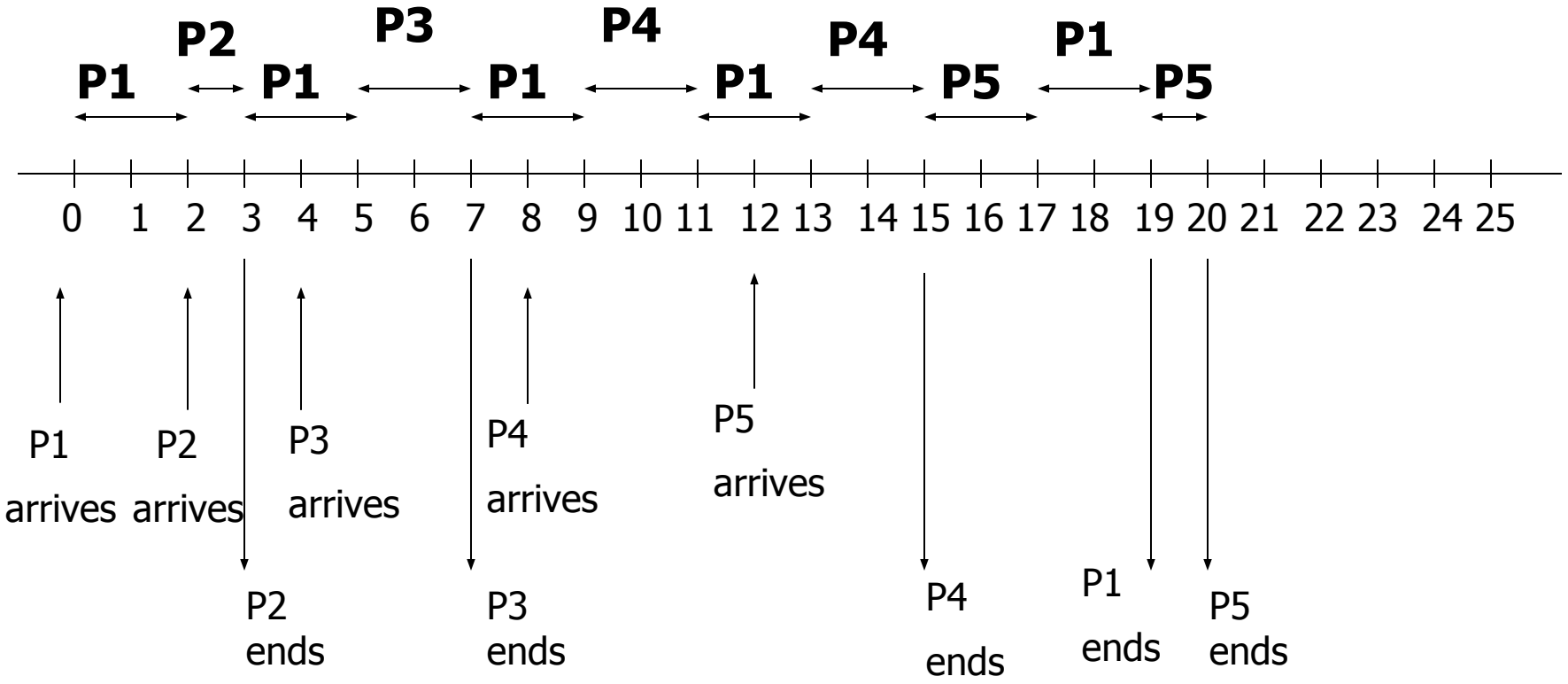
- A major problem with priority based algorithms is indefinite blocking or starvation.
- Too much preemption may result in higher context switching times adversely affecting system performance.

Scheduling Algorithms
Round Robin

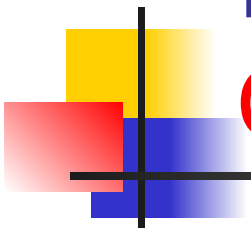


- The CPU cycles through it's process queue and in succession gives its attention to every process for a predetermined time slot.
- A very large time slot will result in a FIFO behavior and a very small time slot results in high context switching and a tremendous decrease in performance.
- The timeline on the following slide should give you a better idea of how the round robin algorithm works.

Scheduling Algorithms
Round Robin



Round Robin Algorithm
Calculating Performance Measures

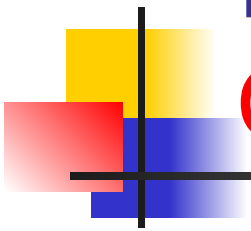


Turnaround Time = End Time – Time of Arrival

- Turnaround Time for P1 = 19 – 0 = 19
- Turnaround Time for P2 = 3 – 2 = 1
- Turnaround Time for P3 = 7 – 4 = 3
- Turnaround Time for P4 = 15 – 8 = 7
- Turnaround Time for P5 = 20 – 12 = 8

Total Turnaround Time = 19+1+3+7+8 = 38

Average Turnaround Time = 38/5 = 7.6



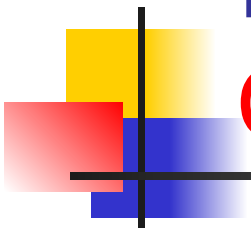
Round Robin Algorithm
Calculating Performance Measures

Waiting Time = Turnaround Time – Execution Time

- Waiting Time for P1 = 19 – 10 = 9
- Waiting Time for P2 = 1 – 1 = 0
- Waiting Time for P3 = 3 – 2 = 1
- Waiting Time for P4 = 7 – 4 = 3
- Waiting Time for P5 = 8 – 3 = 5

Total Waiting Time = 9+0+1+3+5 = 18

Average Waiting Time = 18/5 = 3.6



Round Robin Algorithm
Calculating Performance Measures

Throughput

Total time for completion of 5 processes = 20

Therefore, Throughput = 5/20

= 0.25 processes per unit time

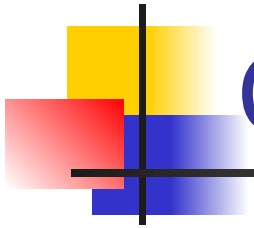
Round Robin Algorithm
Pros and cons

Pros

- It is a fair algorithm.

Cons

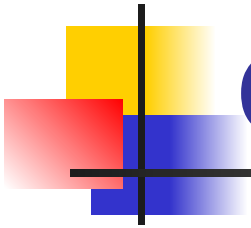
- The performance of the scheme depends heavily on the size of the time quantum.
- If context-switch time is about 10% of the time quantum, then about 10% of the CPU time will be spent in context switch (the next example will give further insight into this).



Comparing Scheduling Algorithms

The following table summarizes the performance measures of the different algorithms:

Algorithm	**Av. Waiting Time**	**Av. Turnaround Time**	**Throughput**
FIFO	5.0	9.0	0.25
SJF(no preemption)	4.8	8.8	0.25
SJF(preemptive)	2.0	6.0	0.25
Priority(no preemption)	5.6	9.6	0.25
Priority(preemptive)	4.2	8.2	0.25
Round Robin	3.6	7.6	0.25

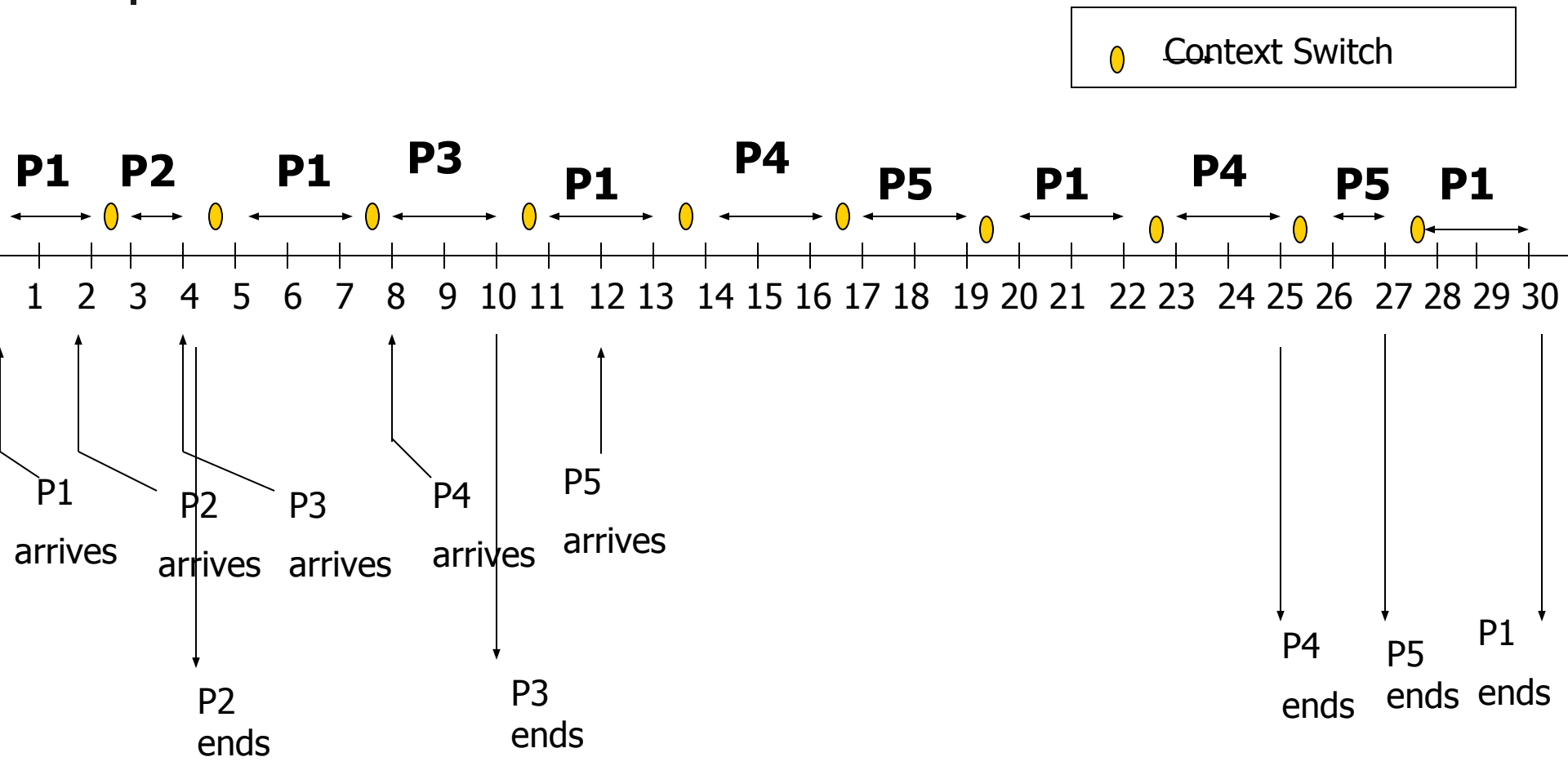


Comparing Scheduling Algorithms

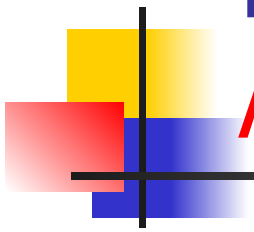
- The preemptive counterparts perform better in terms of waiting time and turnaround time. However, they suffer from the disadvantage of possible starvation
- FIFO and round robin ensure no starvation. FIFO, however, suffers from poor performance measures
- The performance of round robin is dependent on the time slot value. It would also be the most affected once context switching time is taken into account as you'll see in the next example (in the above cases context switching time = 0).

Round Robin Algorithm

An example with context switching



Round Robin Algorithm
An example with context switching



Turnaround Time = End Time – Time of Arrival

- Turnaround Time for P1 = 30 – 0 = 30
- Turnaround Time for P2 = 4 – 2 = 2
- Turnaround Time for P3 = 10 – 4 = 6
- Turnaround Time for P4 = 25 – 8 = 17
- Turnaround Time for P5 = 27 – 12 = 15

Total Turnaround Time = 30+2+6+17+15 = 70

Average Turnaround Time = 70/5 = 14

Round Robin Algorithm
An example with context switching

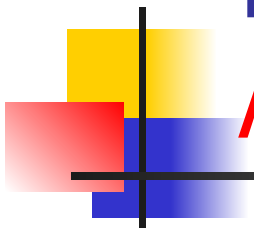
Waiting Time = Turnaround Time – Execution Time

- Waiting Time for P1 = 30 – 10 = 20
- Waiting Time for P2 = 2 – 1 = 1
- Waiting Time for P3 = 6 – 2 = 4
- Waiting Time for P4 = 17 – 4 = 13
- Waiting Time for P5 = 15 – 3 = 12

Total Waiting Time = 20+1+4+13+12 = 50

Average Waiting Time = 50/5 = 10

Round Robin Algorithm
An example with context switching



Throughput

Total time for completion of 5 processes = 30

Therefore, Throughput = 5/30

= 0.17 processes per unit time

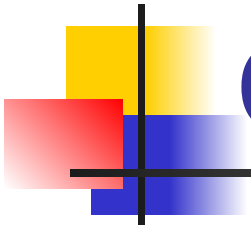
CPU Utilization

% CPU Utilization = 20/30 * 100 = 66.67%

Comparing Performance Measures of Round Robin Algorithms

The following table summarizes the performance measures of the round robin algorithm with and without context switching:

Performance Measures	Round Robin Scheme without Context Switching	Round Robin Scheme with Context Switching
Av. Waiting Time	3.6	10
Av. Turnaround Time	7.6	14
Throughput	0.25	0.17
% CPU Utilization	100%	66.67%

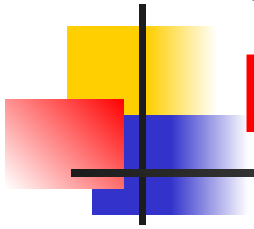


Context Switching Remarks

The comparison of performance measures for the round robin algorithm with and without context switching clearly indicates –

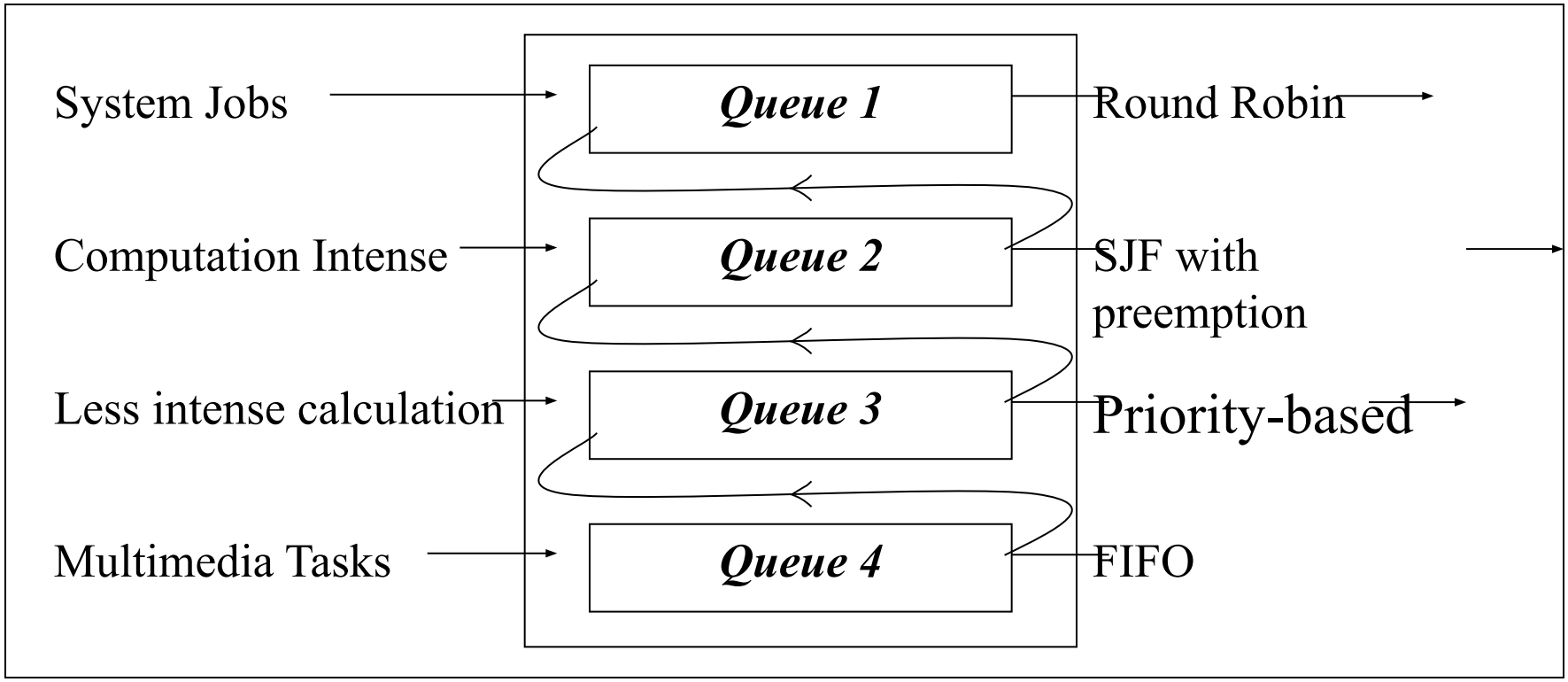
- Context switching time is pure overload, because the system does no useful work while switching.
- Thus, the time quantum should be large with respect to context switching time. However, if it is too big, round robin scheduling degenerates to a FIFO policy. Consequently, finding the optimal value of time quantum for a system is crucial.

Scheduling Algorithms
Multi-level Feedback Queue



- In the previous algorithms, we had a single process queue and applied the same scheduling algorithm to every process.
- A multi-level feedback queue involves having multiple queues with different levels of priority and different scheduling algorithms for each.
- The figure on the following slide depicts a multi-level feedback queue.

Scheduling Algorithms
Multi-level Feedback Queue

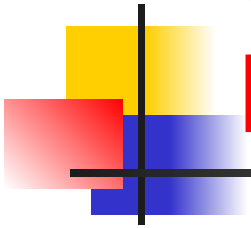


Scheduling Algorithms
Multi-level Feedback Queue

As depicted in the diagram, different kinds of processes are assigned to queues with different scheduling algorithms. The idea is to separate processes with different CPU-burst characteristics and their relative priorities.

- Thus, we see that system jobs are assigned to the highest priority queue with round robin scheduling.
- On the other hand, multimedia tasks are assigned to the lowest priority queue. Moreover, the queue uses FIFO scheduling which is well-suited for processes with short CPU-burst characteristics.

Scheduling Algorithms
Multi-level Feedback Queue



- Processes are assigned to a queue depending on their importance.
- Each queue has a different scheduling algorithm that schedules processes for the queue.
- To prevent starvation, we utilize the concept of aging. Aging means that processes are upgraded to the next queue after they spend a predetermined amount of time in their original queue.
- This algorithm, though optimal, is most complicated and high context switching is involved.

Process & Thread Management

- Concept of process and threads
- Process states
- Process management
- Context switching
- Interaction between processes and OS
- Multithreading
- Example OS : Linux

Process

Process – a program in execution;

- Program is a passive entity, whereas process is a active entity.
- process execution must progress in sequential fashion

A process includes:

- program counter
- Stack (contain local variables, parameters, return addresses)
- data section (contain global variables)

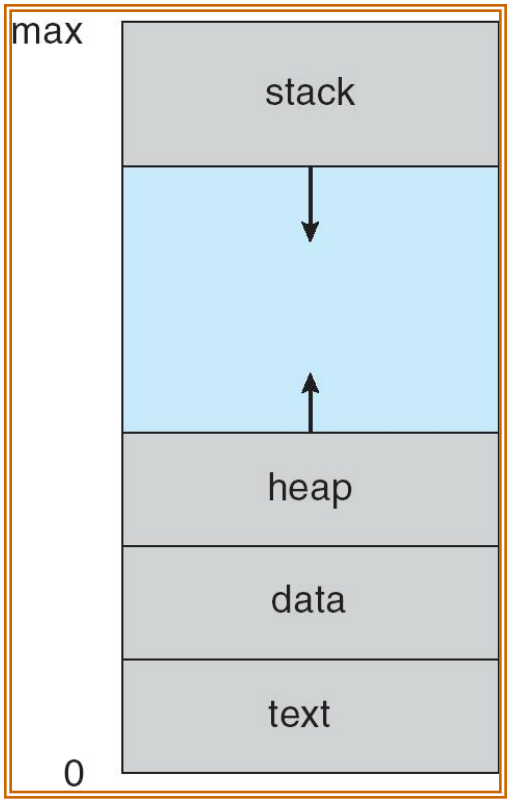
Multiprogramming

- Computers (at least uniprocessors) don't really run multiple programs simultaneously, it just looks that way
- Each process runs to completion, but is interleaved with other processes
- As a process runs, it may have to wait for things like user input or disk I/O
- While one process waits, another can run
- This is multiprogramming

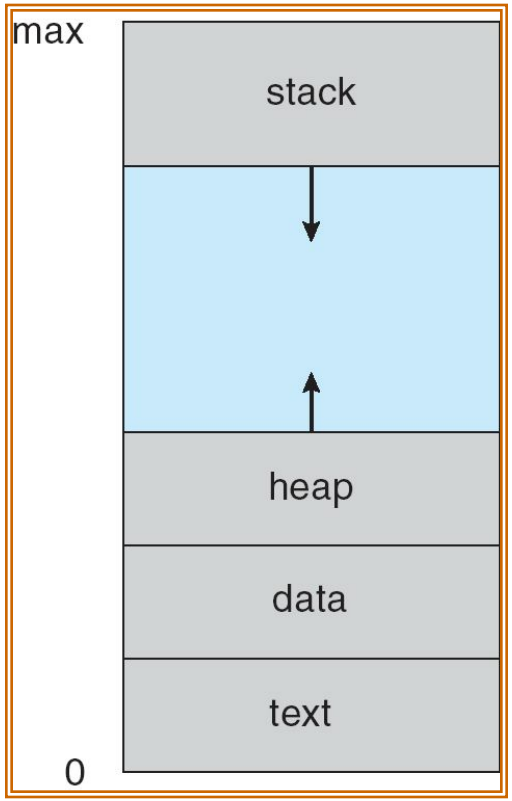
Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms ***job*** and ***process*** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter
  - stack
  - data section
  - code
  - heap
  - allocated memory

Processes in Memory



Process A



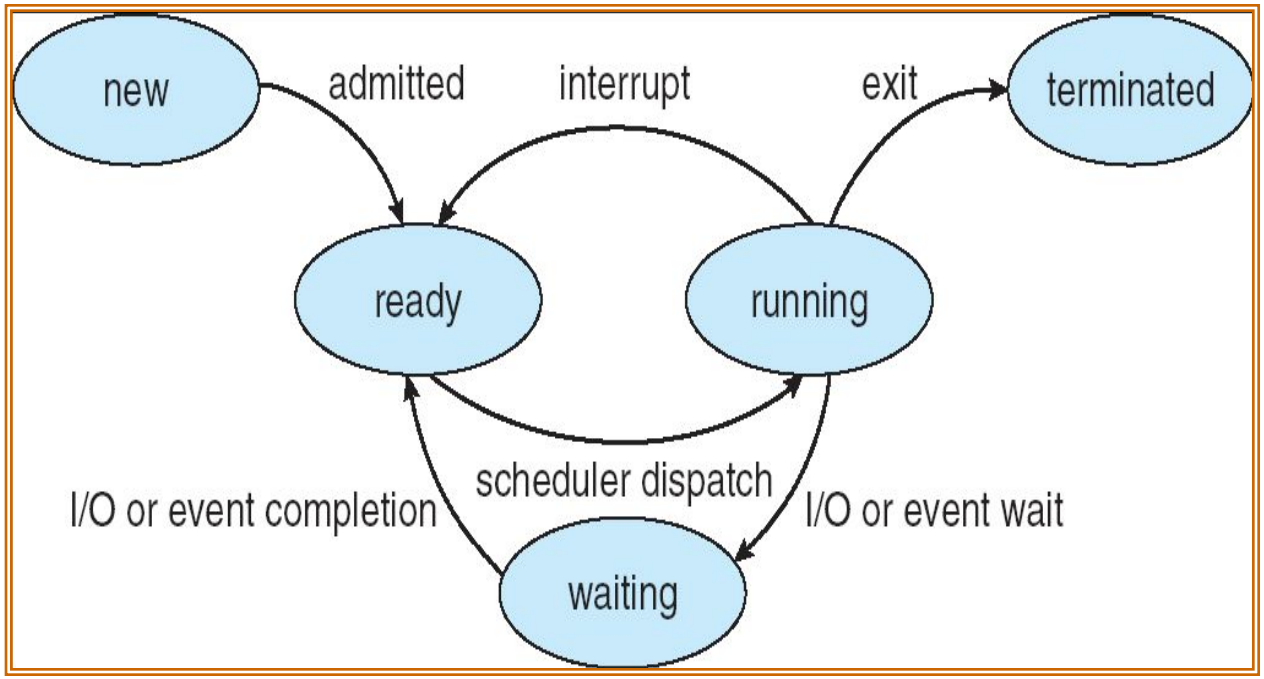
Process B

Processes and Scheduling

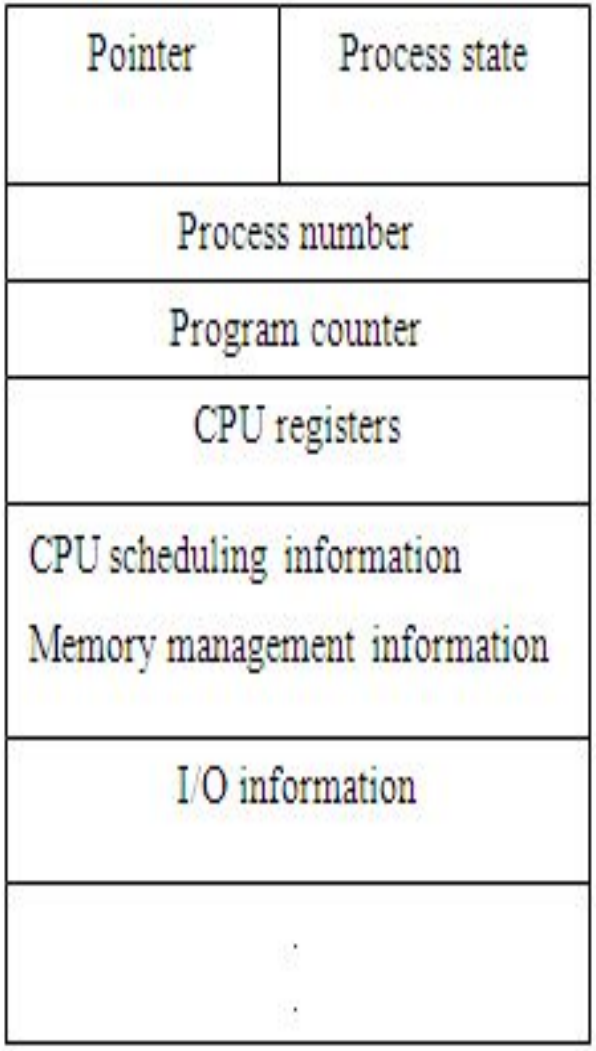
- Processes have *isolation* from each other
  - Address space
  - Security context
  - Termination protection
- Processes are scheduled separately from each other
- One process blocking or being pre-empted allows another to run
- On some systems, a process can be composed of several *threads* which share the process
- On a multiprocessor, processes can and do run simultaneously

Diagram of Process State

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a process
  - **terminated**: The process has finished execution



Process Control Block (PCB)



- **Program counter** contains the address of the next instruction to be executed.
- **CPU registers** which include the accumulator, general purpose registers, index register, stack pointer.
- **CPU scheduling information** consists of parameters for scheduling various processes by the CPU,process priority, pointers to scheduling queue.
- **Memory management information** includes information pertaining to base registers, limit registers, page tables, segment tables.
- **I/O status information** includes list of I/O devices allocated,list of open files

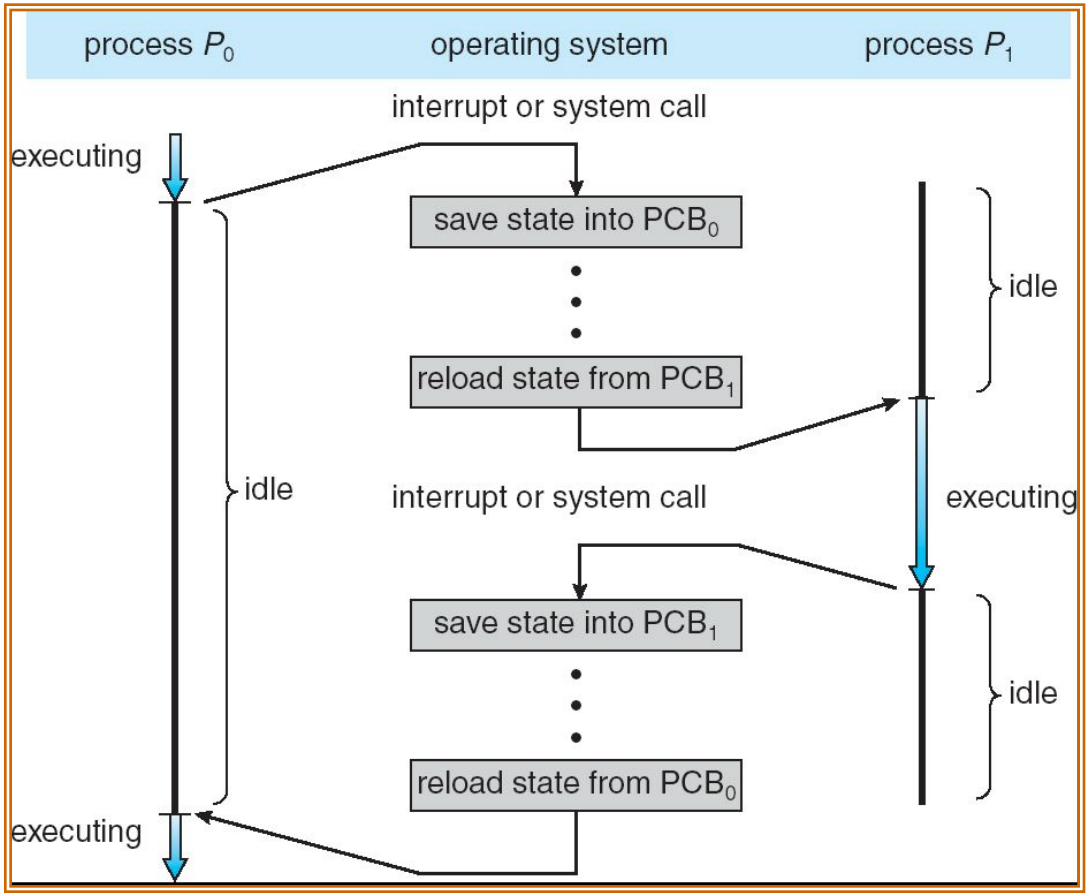
Process Control Block (PCB)

- **I/O** **Status information.** This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

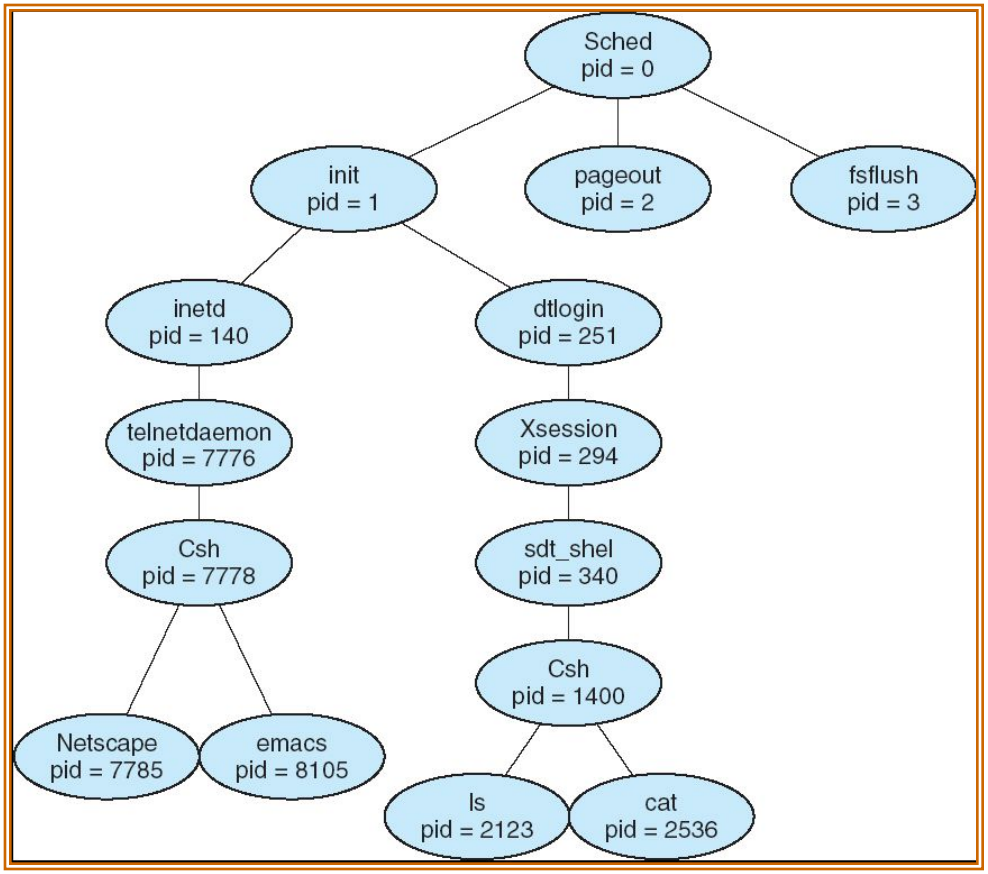
Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

CPU Switch From Process to Process



A tree of processes on a typical Solaris



Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

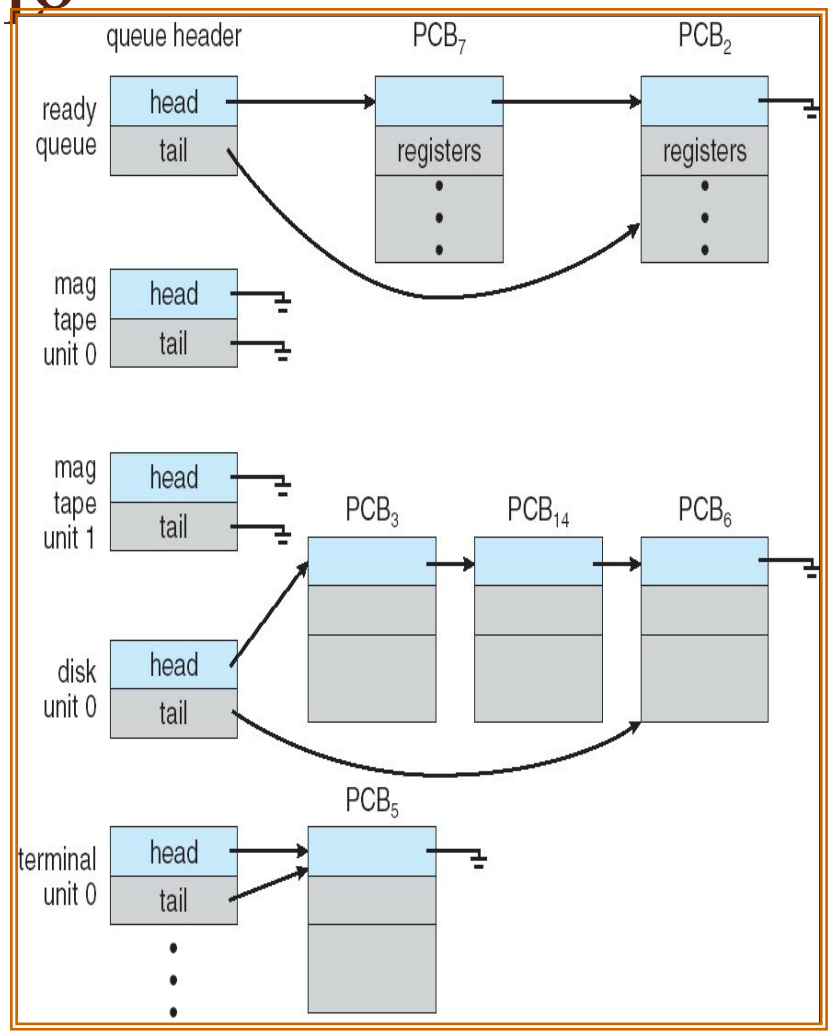
Process Scheduling

- The objective of multiprogramming is to have some process running at all times.
- To maximize the CPU utilization scheduling is done among various processes.
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute.
    - A ready queue header contains pointer to first & last PCBs in the linked list.
    - Each PCB has pointer, points to next process in queue.
  - **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues

Process Scheduling

A new Process is put in ready queue. It waits in queue until it is selected for execution(dispatched)& given to CPU.

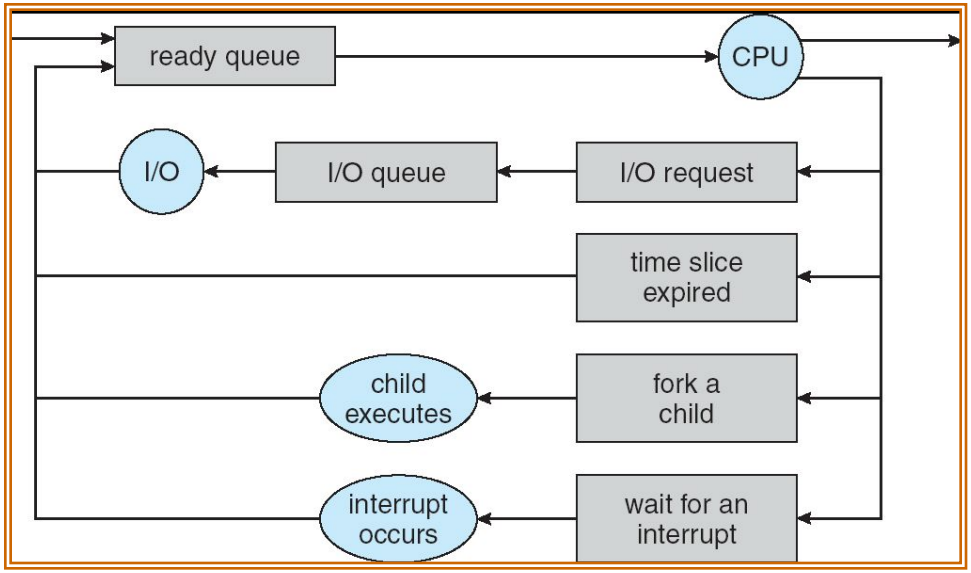
- Process issue I/O request, enters I/O queue.
- Process creates new process & waits for its termination.
- Process removed from CPU due to interrupt & again placed in ready queue.



Ready Queue And Various I/O Device Queues

Process Scheduling

Queuing representation of process scheduling



- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

CPU Schedulers

- **Long-term scheduler (or job scheduler)**

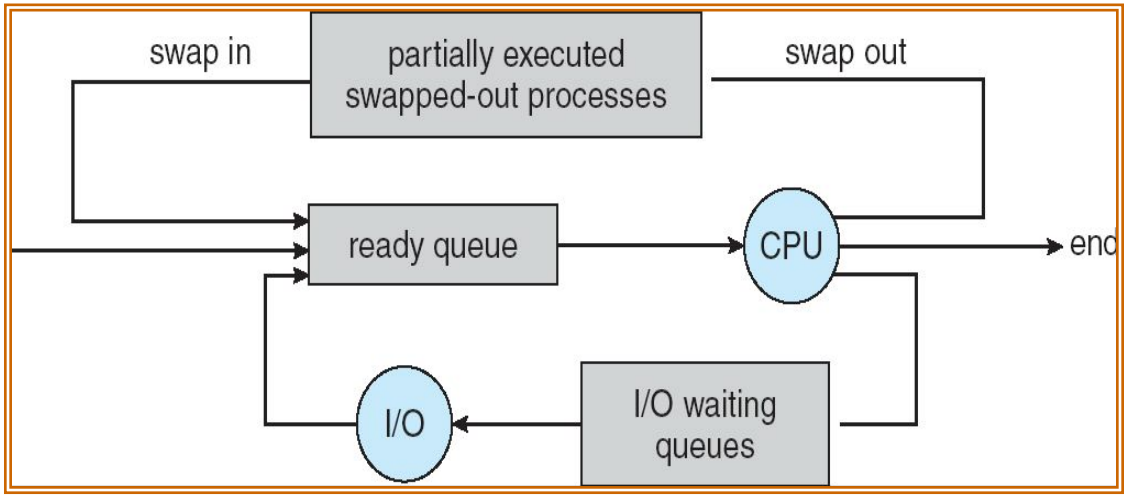
  ✔ selects which processes from secondary storage should be brought into the ready queue.

- **Short-term scheduler (or CPU scheduler)**

  ✔ selects which process should be executed next(from ready queue) and allocates CPU.

Medium term scheduler

- It removes processes from memory and swaps to disk to reduce the degree of multiprogramming to increase throughput.



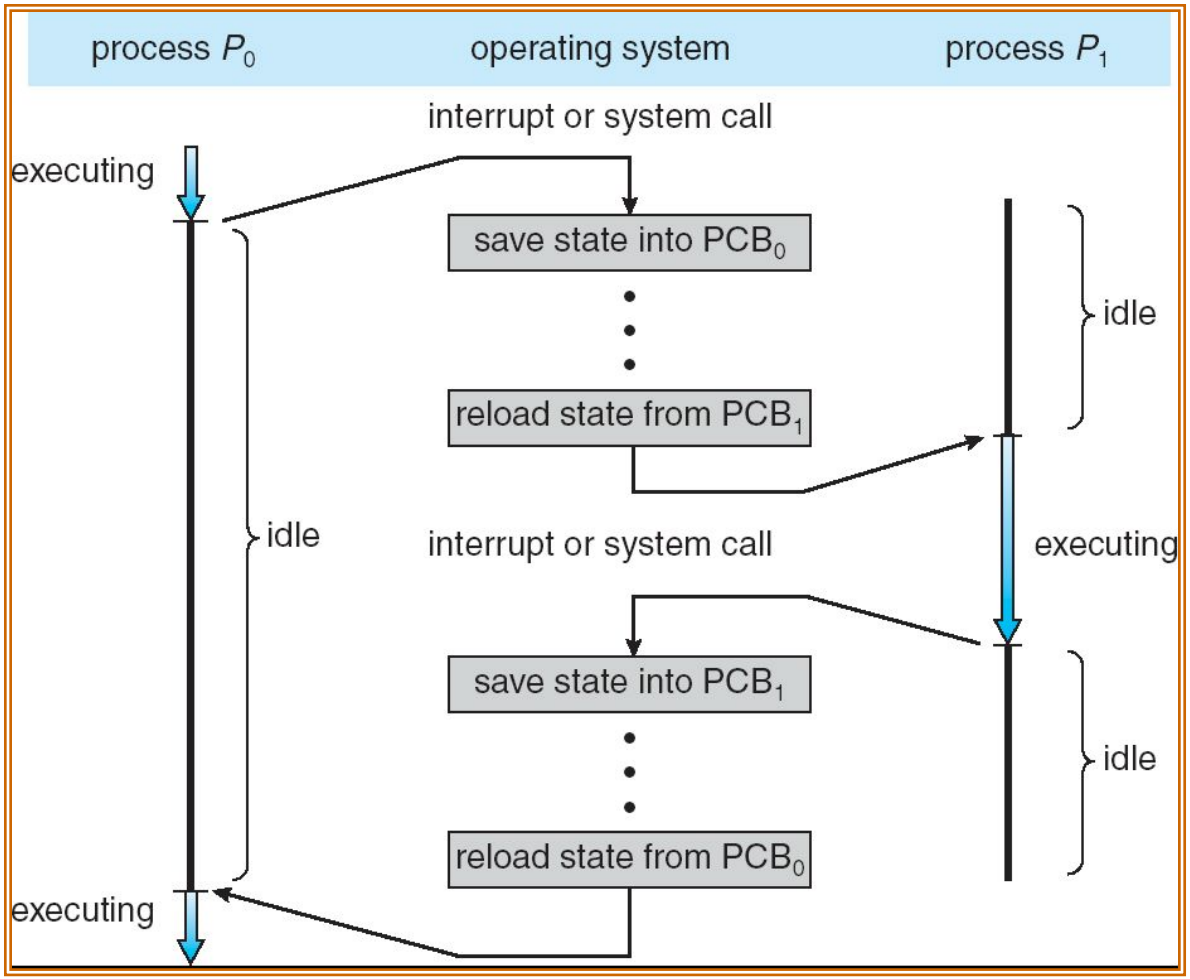
Long term scheduler	Short term scheduler	Medium term scheduler
Selects the process from the disk and loads them into main memory for execution, puts in ready queue	Chooses the process from ready queue and assigns it to CPU	Swaps in and out the process from memory
Speed is less	Speed is fast	Speed is moderate
Transition of process from New to Ready	Transition of process from Ready to executing	No process state transtition
Not present in time sharing system	Minimal in Time sharingsystem	Present in Time sharing system
Supply a reasonable mix of jobs, such as I/O bound and CPU bound	Select a new process to allocate to CPU frequently	Process are swapped in and out for balanced process mix

Long term scheduler	Short term scheduler	Medium term scheduler
Controls degree of multiprogramming through placing process in ready queue	It has a control over degree of multiprogramming as it allocates process to CPU	Reduce the degree of multiprogramming by swapping process in and out
It is called as job scheduler	It is called as CPU scheduler	

Context Switch

- The machine register contain the hardware context of currently running process. When a context switch occurs, these registers are saved in PCB of current process.
  - When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
  - Context-switch time is overhead; the system does no useful work while switching.
  - Context switch time dependent on hardware support. For example, if register contents do not have to be saved because of the availability of large number of registers, then context switch time will be low.

Context Switch



Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Types of Scheduling

- Non-Preemptive
- Preemptive

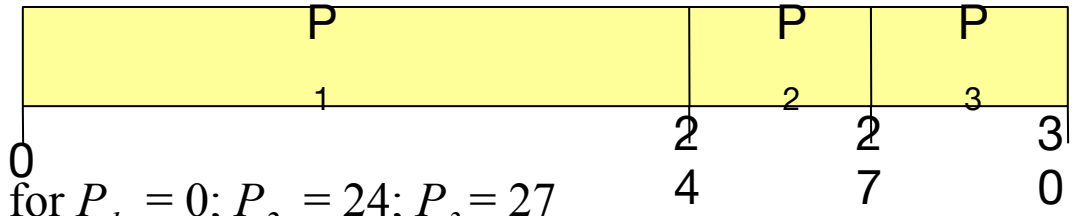
Types of Scheduling algorithm

- First in First out(FIFO)
- Shorest Job First (SJF)
- Priority Scheduling
- Round Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- With FCFS, the process that requests the CPU first is allocated the CPU first
- Case #1: Suppose that the processes arrive in the order: P_1 , P_2 , P_3
- Gantt Chart :

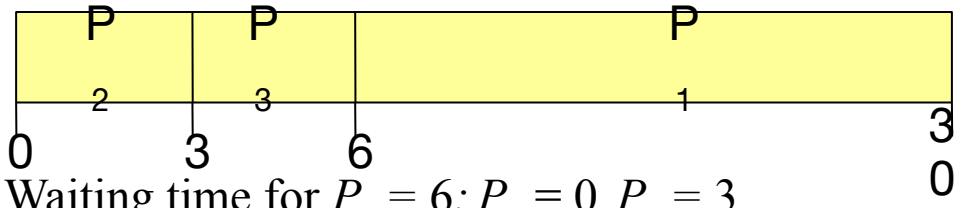


- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: (0 + 24 + 27)/3 = 17
- Average turn-around time: (24 + 27 + 30)/3 = 27

FCFS Scheduling (Cont.)

- **Case #2: Suppose that the processes arrive in the order: P_2 , P_3 , P_1**

Gantt chart:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3 (Much better than Case #1)
- Average turn-around time: (3 + 6 + 30)/3 = 13

FCFS Scheduling (Cont.)

- Case #1 is an example of the **convoy effect;** all the other processes wait for one long-running process to finish using the CPU
  - This problem results in lower CPU and device utilization; Case #2 shows that higher utilization might be possible if the short processes were allowed to run first
- The FCFS scheduling algorithm is **non-preemptive**
  - Once the CPU has been allocated to a process, that process keeps the CPU until it releases it either by terminating or by requesting I/O
  - It is a troublesome algorithm for time-sharing systems

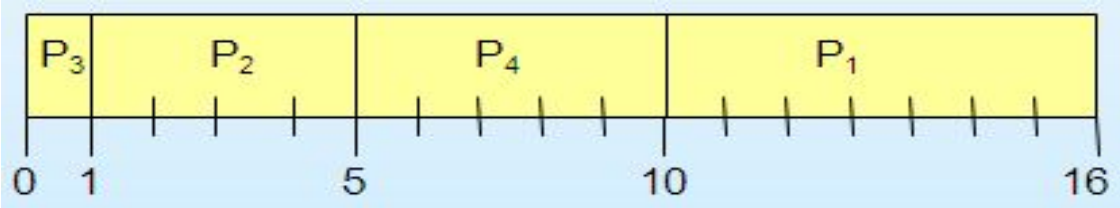
Shortest-Job-First (SJR) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - **nonpreemptive –** once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **preemptive –** if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.

Example #1: Non-Preemptive SJF (simultaneous arrival)

Process	Arrival Time	Burst Time
P_1	0.0	6
P_2	0.0	4
P_3	0.0	1
P_4	0.0	5

- SJF (non-preemptive, simultaneous arrival)

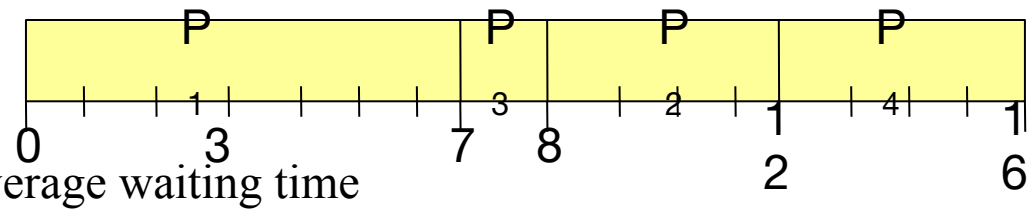


- Average waiting time = (0 + 1 + 5 + 10)/4 = 4
- Average turn-around time = (1 + 5 + 10 + 16)/4 = 8

Example #2: Non-Preemptive SJF (varied arrival times)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive, varied arrival times)



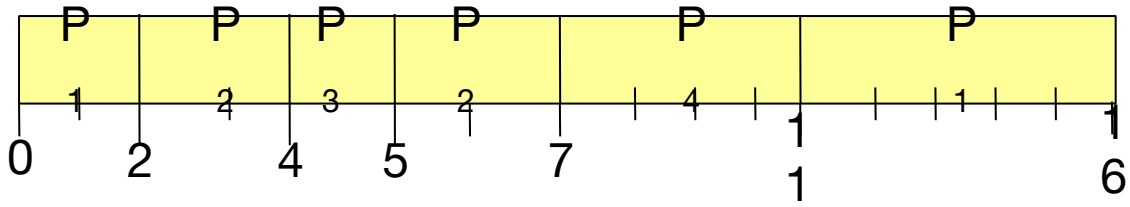
- Average waiting time
  $= ((0 – 0) + (8 – 2) + (7 – 4) + (12 – 5))/4$
  $= (0 + 6 + 3 + 7)/4 = 4$
- Average turn-around time:
  $= ((7 – 0) + (12 – 2) + (8 - 4) + (16 – 5))/4$
  $= (7 + 10 + 4 + 11)/4 = 8$

Waiting time : sum of time that a process has spent waiting in the ready queue

Example #3: Preemptive SJF (Shortest-remaining-time-first)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive, varied arrival times)



- Average waiting time
  = ([(0 – 0) + (11 - 2)] + [(2 – 2) + (5 – 4)] + (4 - 4) + (7 – 5))/4
  = 9 + 1 + 0 + 2)/4
  = 3
- Average turn-around time = ([16-0] + [7-2] + [5-4] +[11-5])/4 = 7

Waiting time : sum of time that a process has spent waiting in the ready queue

Priority Scheduling

- A priority number (integer) is associated with each process
- Priority scheduling is a form of preemptive scheduling where priority is the basis of preemption.
- It can be non-preemptive also.
- (smallest integer ≡ highest priority)
- **Problem ≡ Starvation** – low priority processes may never execute.
- **Solution ≡ Aging** – as time progresses increase the priority of the process.

Priority Scheduling

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	*2*	4
P4	1	5
P5	*5*	2

Process	Burst Time	Priority	Arrival time
P1	10	3	0
P2	5	2	1
P3	*2*	1	2

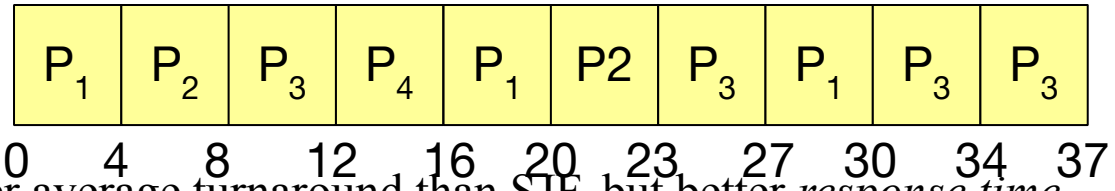
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), 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, No process waits more than $(n-1)q$ time units.
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large w.r.t. context switch, otherwise overhead is too high.
- One rule of thumb is that 80% of the CPU bursts should be shorter than the time quantum

Example of RR with Time Quantum = 4 ms

Process	Burst Time
P_1	11
P_2	7
P_3	15
P_4	4

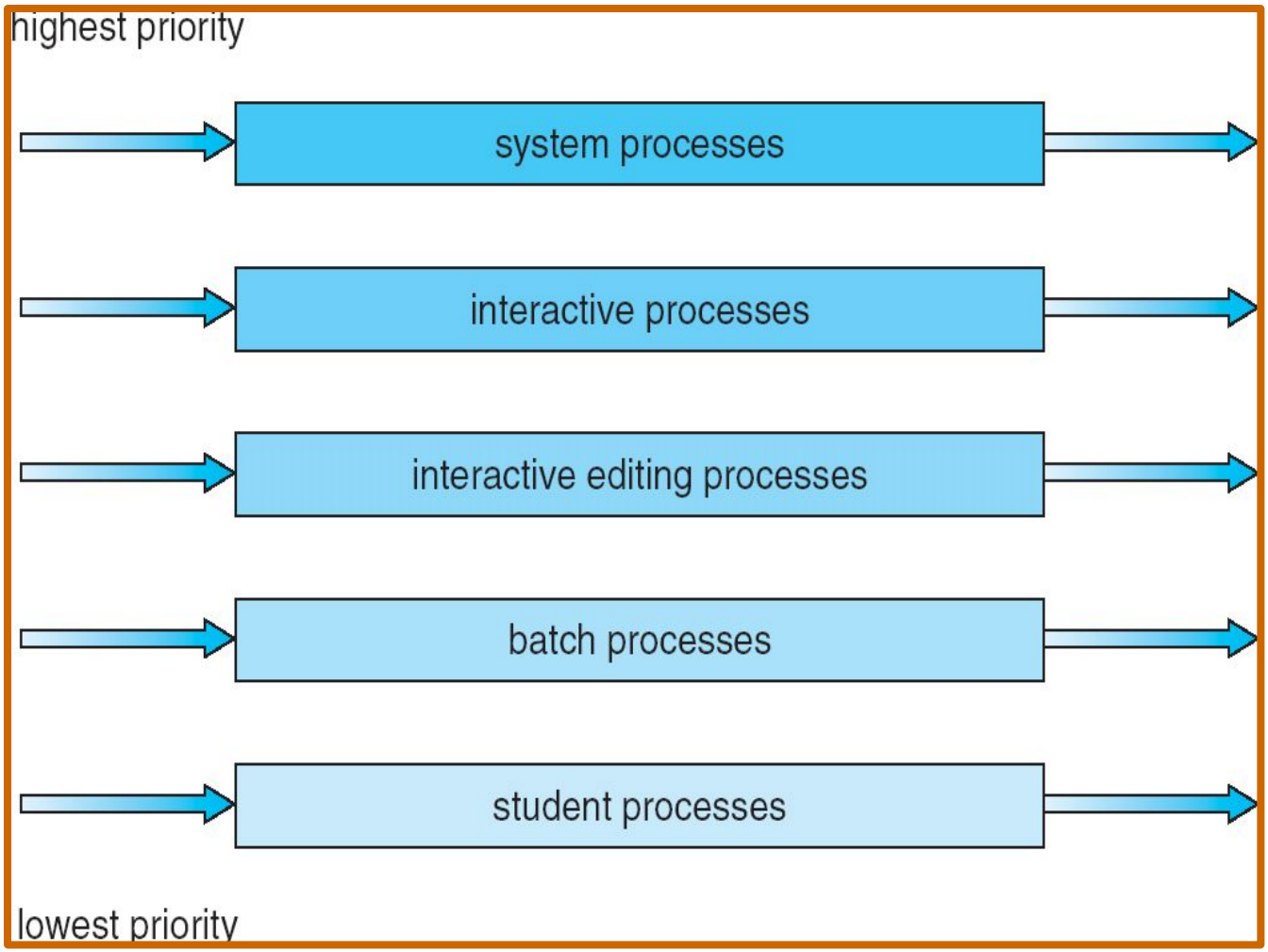
- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better *response time*
- Average turn-around time = (30-0)+(23-0)+(37-0)+(16-0)
- =30 + 23 + 37 + 16) / 4 = 26.5
- Average waiting time = [0+(16-4)+(27-20)]+[4+(20-8)]+[8+(23-12)+(30-27)]+[12]
- = (19+4+22+12)/4

  =14.25

Multi-level Queue Scheduling



Multi-level Queue Scheduling

- Multi-level queue scheduling is used when processes can be classified into groups
- For example, **foreground** (interactive) processes and **background** (batch) processes
  - The two types of processes have different response-time requirements and so may have different scheduling needs
  - Also, foreground processes may have priority (externally defined) over background processes
- A multi-level queue scheduling algorithm partitions the ready queue into several separate queues
- The processes are permanently assigned to one queue, generally based on some property of the process such as memory size, process priority, or process type

Multi-level Queue Scheduling

- Each queue has its own scheduling algorithm
  - The foreground queue might be scheduled using an RR algorithm
  - The background queue might be scheduled using an FCFS algorithm
- In addition, there needs to be scheduling among the queues, which is commonly implemented as fixed-priority pre-emptive scheduling
  - The foreground queue may have absolute priority over the background queue
- One example of a multi-level queue are the five queues shown below
- Each queue has absolute priority over lower priority queues
- For example, no process in the batch queue can run unless the queues above it are empty
- However, this can result in starvation for the processes in the lower priority queues

Multilevel Queue Scheduling

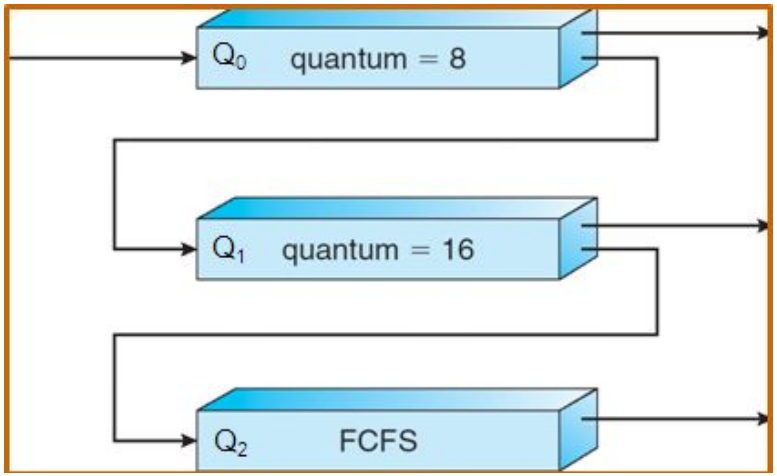
- Another possibility is to time slice among the queues
- Each queue gets a certain portion of the CPU time, which it can then schedule among its various processes
  - The foreground queue can be given 80% of the CPU time for RR scheduling
  - The background queue can be given 20% of the CPU time for FCFS scheduling

Multilevel Feedback Queue Scheduling

- In multi-level feedback queue scheduling, a process can move between the various queues; aging can be implemented this way
- A multilevel-feedback-queue scheduler is defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to promote a process
  - Method used to determine when to demote a process
  - Method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

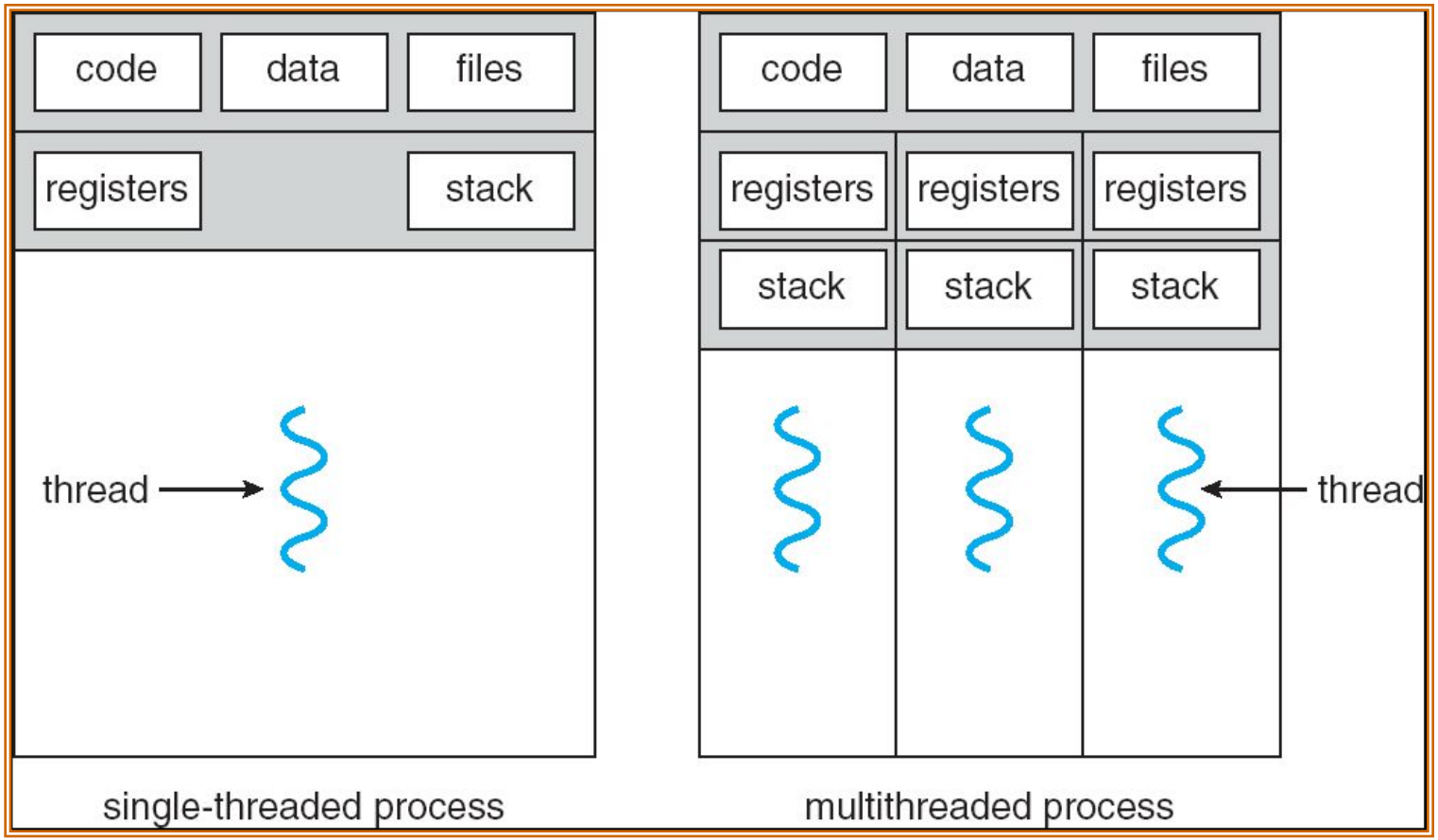
- Scheduling
  - A new job enters queue Q_0 *(RR)* and is placed at the end. When it gains the CPU, the job receives 8 milliseconds. If it does not finish in 8 milliseconds, the job is moved to the end of queue Q_1.
  - A Q_1 (RR) job receives 16 milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 (FCFS).



Multithreading

- Multithreading refers to the ability of an OS to support multiple , concurrent paths of execution within a single process.
- A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- A traditional (or **heavyweight) process** has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time.

Single and Multithreaded Processes



Multithreading (Benefits)

- **Responsiveness:** Multithreading an interactive application increases responsiveness to the user.
- **Resource Sharing:** Threads share memory and resources of the processes to which they belong.
- **Economy:** Process creation is expensive.
- **Utilization of Multiprocessor Architectures:** Benefits of multithreading increase in multiprocessor systems because different threads can be scheduled to different processors

Difference Process / Thread

Process	Thread
Program in execution	It is the part of process
It is heavy weight process	It is light weight process
Process context switch takes more time	Thread context switch takes less time
New process creation, termination takes more time	New Thread creation, termination takes lesse time
Each process executes the same code but has its own memory ad file resources	All thread can share same set of open files, child process
In process based implementation if one process is blocked , no other server process can execute until the first process unblocked	In multithreaded server implementation, if one thread blocked and waiting, second thread in the same process could execute
Multiple redundant process use more resources	Multiple threaded process use fewer resources

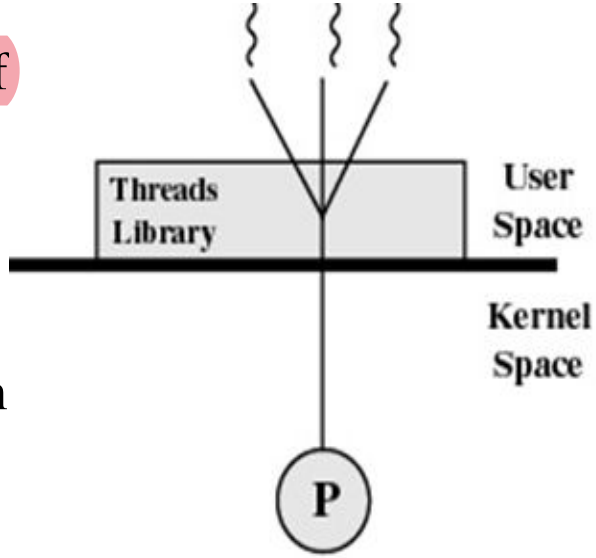
Multithreading: Types of threads

1. User level threads
2. Kernel level threads

Multithreading: Types of threads

User level threads

- Threads are implemented at user level by Thread Library: creates, scheduling and management of threads without kernel involvement.
- Hence fast to create & manage.
- Status of information table is maintained within Thread Library hence better scalability.



User level threads

- ADVANTAGES

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.
- Supported above the kernel, via a set of library calls at the user level
- Fast switching among threads: Threads do not need to call OS and cause interrupts to kernel – a user-level threads package can be implemented on an Operating System that does not support threads.
- User-level threads does not require modification to operating systems

User level threads

- Disadvantages

- When a user level thread executes a blocking system call, not only that thread is blocked, but also all of the threads within the process are blocked.
- There is a lack of coordination between threads and operating system kernel.
- Therefore, process as whole gets one time slice irrespective of whether process has one thread or 1000 threads within.
- It is up to each thread to relinquish control to other threads
- Multithreaded application cannot take advantage of multiprocessing

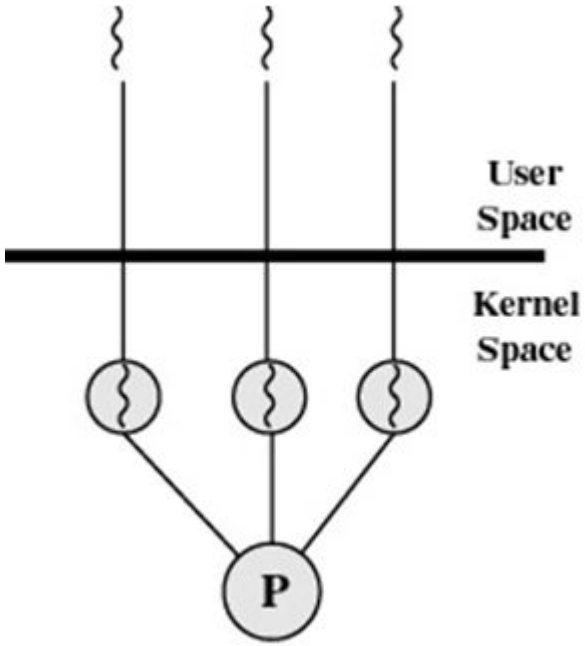
User level threads

- Example thread libraries: –
- POSIX Pthreads
- Win32 threads
- Java threads

Multithreading: Types of threads

- Kernel Threads

- Kernel maintains process table and keeps track of all processes.
- Kernel threads are slow & insufficient because it requires a full TCB(Thread Control Block) for each thread to manage and schedule.



Kernel Threads

- ADVANTAGES
- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can multithreaded.
- DISADVANTAGES
- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within same process requires a mode switch to the Kernel.

Kernel Threads

- Disadvantages – The kernel-level threads are slow and inefficient .
- For instance, threads operations are hundreds of times slower than that of user-level threads.
- Since kernel must manage and schedule threads as well as processes.
- It require a full thread control block (TCB) for each thread to maintain information about threads.
- As a result there is significant overhead and increased in kernel complexity

Kernel Threads

- Examples – Windows XP/2000, Solaris, Linux, Tru64 UNIX, Mac OS X

Difference between User Level & Kernel Level Thread

User Level thread	Kernel Level Thread
User level threads are faster to create and manage	Kernel level threads are slower to create and manage.
Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads
User level thread is generic and can run on any operating system	Kernel level thread is specific to the operating system
Multi-threaded application cannot take advantage of multiprocessing	Kernel routines themselves can be multithreaded.

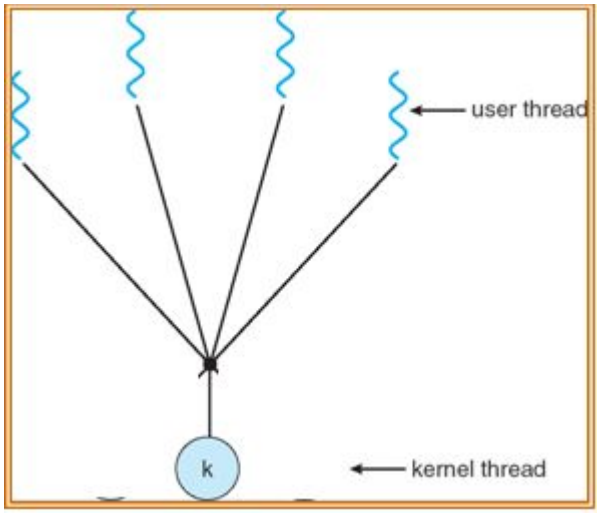
Multithreading: Thread Models

- Many-to-One Model
- One-to-one Model
- Many-to-Many Model

Multithreading: Many-to-One Model

Many-to-One: maps many user level threads into one kernel level thread.

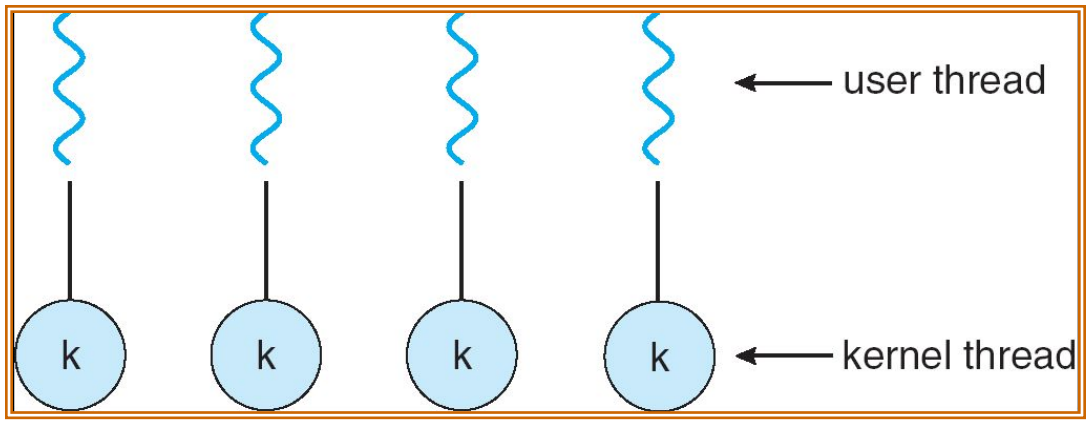
- The entire process will block if a thread makes a blocking system call
- Since only one thread can access kernel at a time, multiple threads cannot run concurrently and thus cannot make use of multiprocessors



Multithreading: One-to-One Model

One-to-One: maps each user level thread to a kernel level thread

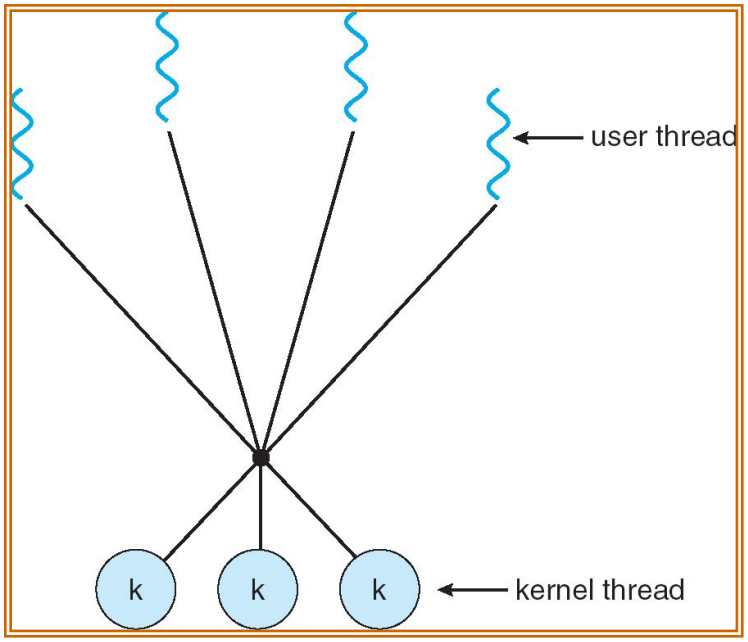
- Creating a user level thread results in creating a kernel thread
- More overhead and allows parallelism
- Because of the overhead most implementations limit the number of kernel threads created



Multithreading: Many-to-Many Model

Many-to-Many: Multiplexes many user level threads to a smaller or equal number of kernel threads

- Has the advantages of both the many-to-one and one-to-one model



i) FCFS

ii) Pre-emptive and non pre-emptive SJF

iii) Pre-emptive priority

Process	Arrival Time	Burst Time	Priority
P1	0	8	3
P2	1	1	1
P3	2	2	2
P4	3	3	3
P5	4	6	4

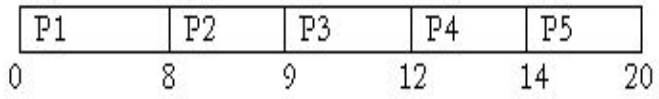
FCFS

Process	Arrival Time	Burst Time	Priority	Waiting Time	Turn Around Time
P1	0	8	3	0	8
P2	1	1	1	7	8
P3	2	3	2	7	10
P4	3	2	3	9	11
P5	4	6	4	10	16

Average WT: Total Waiting Time / No of processes: 0+7+7+9+10/5 = 6.6

Average TAT: Total Turn Around Time / No of processes: 8+8+10+11+16/5 = 10.6

Gantt Chart:



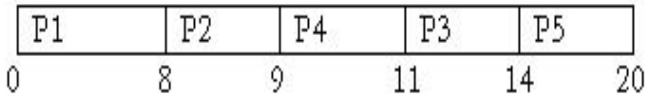
Non Pre-emptive SJF:

Process	Arrival Time	Burst Time	Waiting Time	Turn Around Time
P1	0	8	0	8
P2	1	1	7	8
P3	2	3	9	12
P4	3	2	6	8
P5	4	6	10	16

Average WT: Total Waiting Time / No of processes = 6.4

Average TAT: Total Turn Around Time / No of processes = 10.4

Gantt Chart:



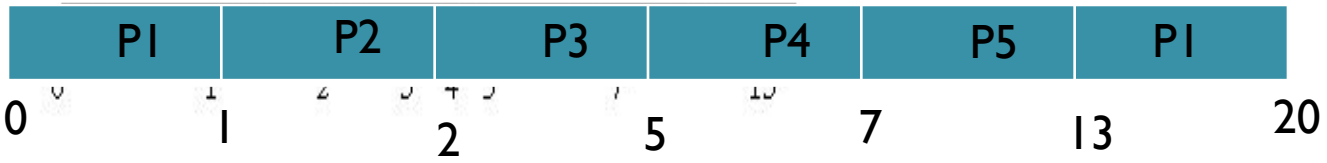
Pre-emptive SJF:

Process	Arrival Time	Burst Time	Waiting Time	Turn Around Time
P1	0	8	12	20
P2	1	1	0	1
P3	2	3	0	3
P4	3	2	2	4
P5	4	6	3	9

Average WT: Total Waiting Time / No of processes = 3.4

Average TAT: Total Turn Around Time / No of processes = 7.4

Gantt chart:



Pre-emptive Priority:

Process	Arrival Time	Burst Time	Waiting Time	Turn Around Time
P1	0	8	3	12
P2	1	1	1	1
P3	2	3	2	3
P4	3	2	3	11
P5	4	6	4	16

Average WT: Total Waiting Time/No of processes = 4.6

Average TAT: Total Turn Around Time/No of processes = 8.6

Gantt Chart :