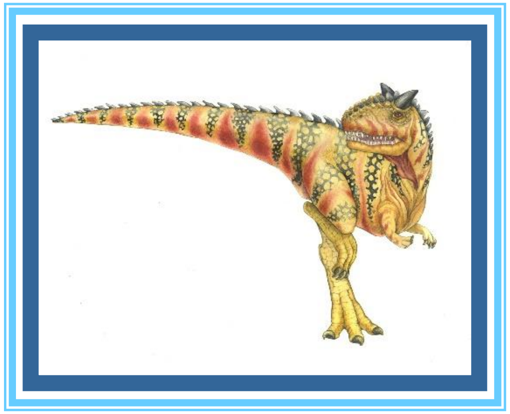
Operating Systems



Lecture 4:

Scheduling

Contact Information

Instructor: Engr. Shamila Nasreen

Assistant Professor

Department of Software Engineering

MUST

Email: shamila.se@must.edu.com

Office hours: Wednesday: 8:30–10:00 AM

Thursday: 8:30–10:00 AM

Objectives

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems.
- To describe various CPU-scheduling algorithms.
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system.
- To examine the scheduling algorithms of several operating systems.

Today's Lecture Agenda

- Types of Processor Scheduling
  - Long Term Scheduling
  - Medium term Scheduling
  - Short Term Scheduling
- Scheduling Algorithms
- Short term scheduling Criteria

Basic Concepts

- Maximum CPU utilization is obtained with multiprogramming
  - Several processes are kept in memory at one time
  - Every time a running process has to wait, another process can take over use of the CPU
- Scheduling of the CPU is fundamental to operating system design
- The CPU scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them

Aims Of Scheduling

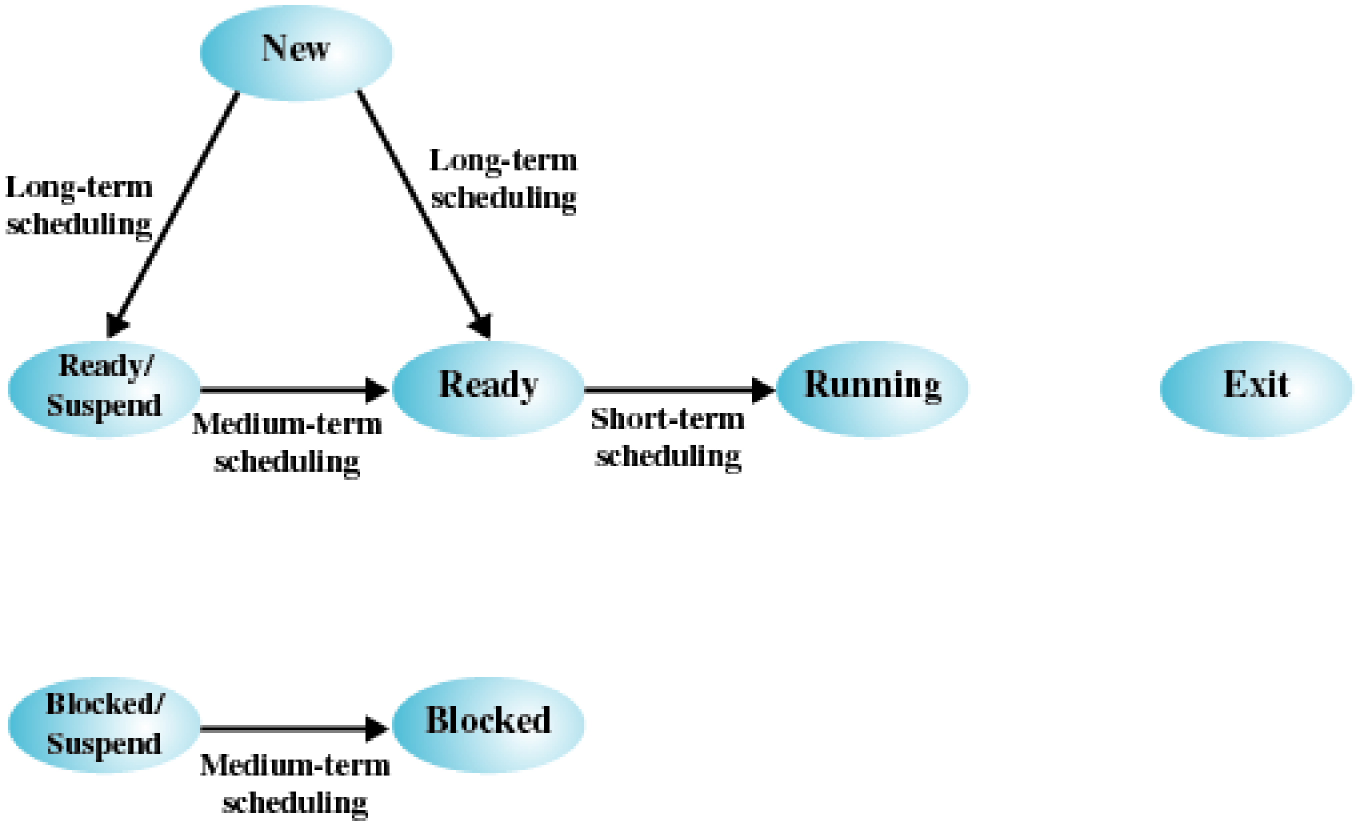
- The key to multiprogramming is scheduling.
- Aims:
  - Assign processes to be executed by the processor(s)
    -
  - Response time
  - Throughput
  - Processor efficiency
  - Fairness

Types of Scheduling

Table 9.1 Types of Scheduling

Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

Scheduling and Process State Transitions



Medium term scheduling

- Part of the swapping function
- Based on the need to manage the degree of multiprogramming

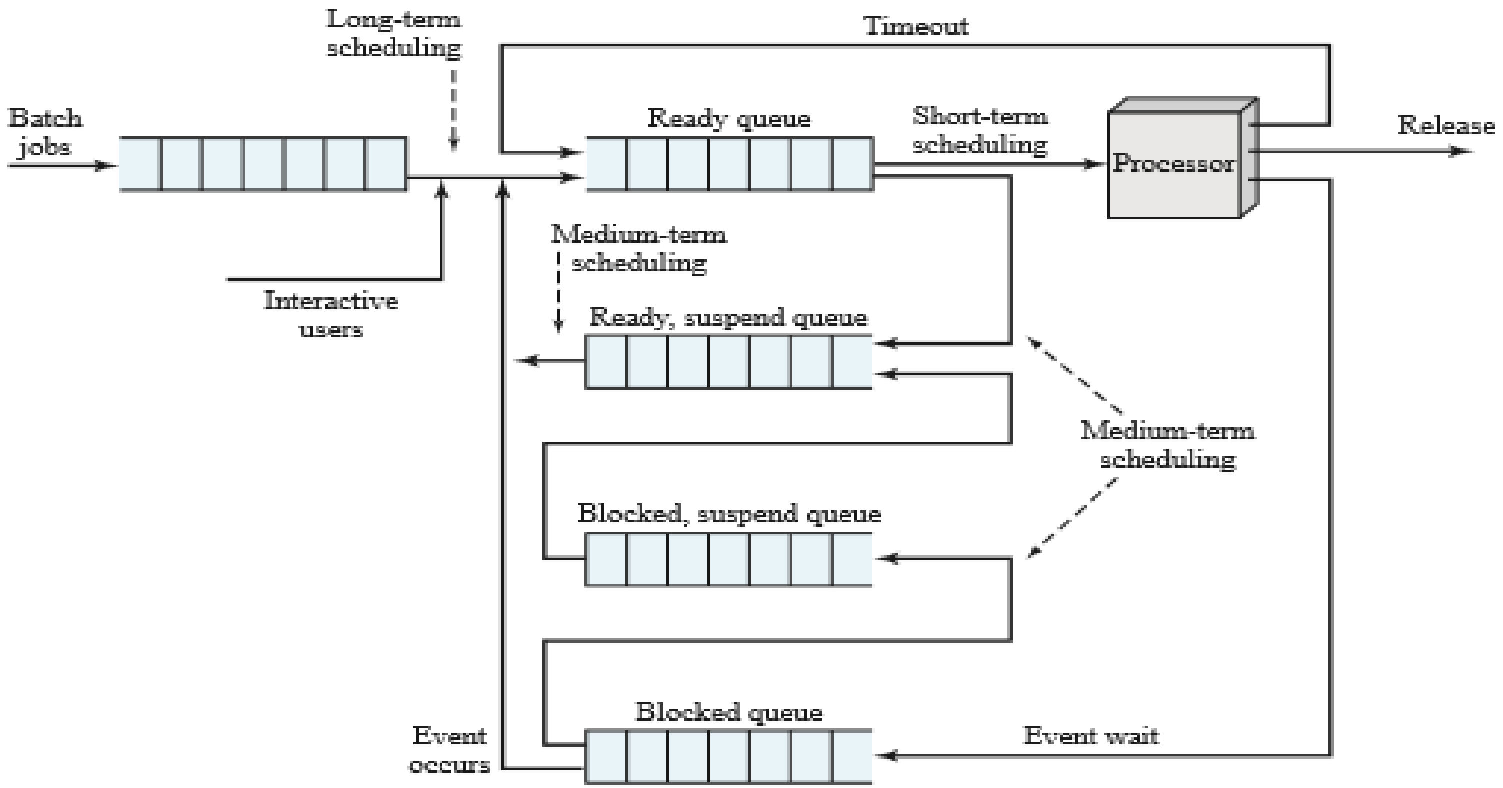


Figure 9.3 **Queuing Diagram for Scheduling**

Short Term Scheduling

- In terms of Frequency of execution:
  - the **long-term scheduler** executes relatively infrequently
  - and makes the coarse-grained decision of whether or not to take on a new process and which one to take.
  - The **medium-term scheduler** is executed somewhat more frequently to make a swapping decision.
  - The **short-term scheduler** executes most frequently
  - and makes the fine-grained decision of which process to execute next.
- known as the dispatcher
- Dispatch latency – time it takes for the dispatcher to stop one process and start another running

Short Term Scheduling

Points in time when OS may perform process scheduling:

- When a process blocks waiting for an event.
  - Perform a system call.
  - I/O request
  - Wait() invocation
- When an interrupt happens.
  - Clock interrupt.
  - I/O interrupt.
- When a process switches from waiting state to ready state
  - I/O completion
- When a process ends.

Short Term Scheduling Criteria

- Generally, a set of criteria is established against which various scheduling policies may be evaluated.
- **User Oriented Criteria:**
  - User oriented criteria relate to the behavior of the system as perceived by the individual user or process.
  - An example is **response time** in an interactive system.
  - Response time is the elapsed time between the submission of a request until the response begins to appear as output.
  - This quantity is visible to the user and is naturally of interest to the user.
  - We would like a scheduling policy that provides "**good**" service to various users.
  - In the case of response time, a threshold may be defined, say 2 seconds.
  - Then a goal of the scheduling mechanism should be to maximize the number of users who experience an average response time of 2 seconds or less.

Short Term Scheduling Criteria

- **System Oriented Criteria:**
  - focus is on effective and efficient utilization of the processor.
  - An example is **throughput**, which is the rate at which processes are completed.
  - This is certainly a worthwhile measure of system performance and one that we would like to maximize.
  - However, it focuses on system performance rather than service provided to the user.

Decision Mode of Scheduling

- Nonpreemptive
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O
- Preemptive
  - Currently running process may be interrupted and moved to the Ready state by the operating system
  - Allows for better service since any one process cannot monopolize the processor for very long

Scheduling Criteria

- Different CPU scheduling algorithms have different properties
- The choice of a particular algorithm may favor one class of processes over another
- In choosing which algorithm to use, the properties of the various algorithms should be considered
- Criteria for comparing CPU scheduling algorithms may include the following
  - **CPU utilization** – percent of time that the CPU is busy executing a process
  - **Throughput** – number of processes that are completed per time unit
  - **Response time** – amount of time it takes from when a request was submitted until the first response occurs (but not the time it takes to output the entire response)
  - **Waiting time** – the amount of time before a process starts after first entering the ready queue (or the sum of the amount of time a process has spent waiting in the ready queue)
  - **Turnaround time(Tq)** – amount of time to execute a particular process from the time of submission through the time of completion
    - Overall time a process is in system.
    - Tq = Tf – Ti
    - Tf: Finalization time.
    - Ti: Initiation time.

Optimization Criteria

- It is desirable to
  - Maximize CPU utilization
  - Maximize throughput
  - Minimize turnaround time
  - Minimize start time
  - Minimize waiting time
  - Minimize response time
- In most cases, we strive to optimize the <u>average</u> measure of each metric

First come, First serve

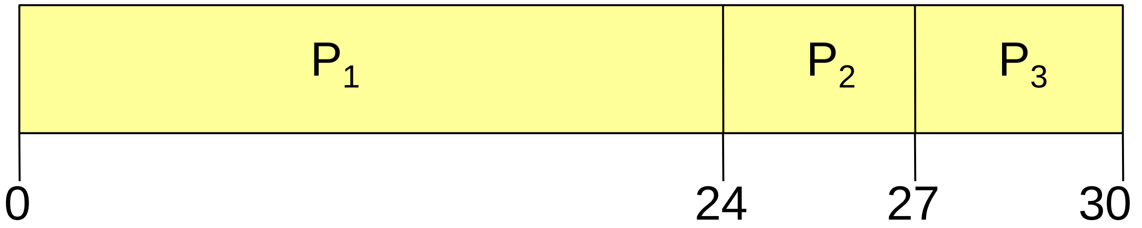
- Simplest scheduling algorithm:
  - Run jobs in order that they arrive
- Non-preemptive
  - A Process keeps CPU until done(terminate) or requesting I/O
- Advantage:
  - Simplicity

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

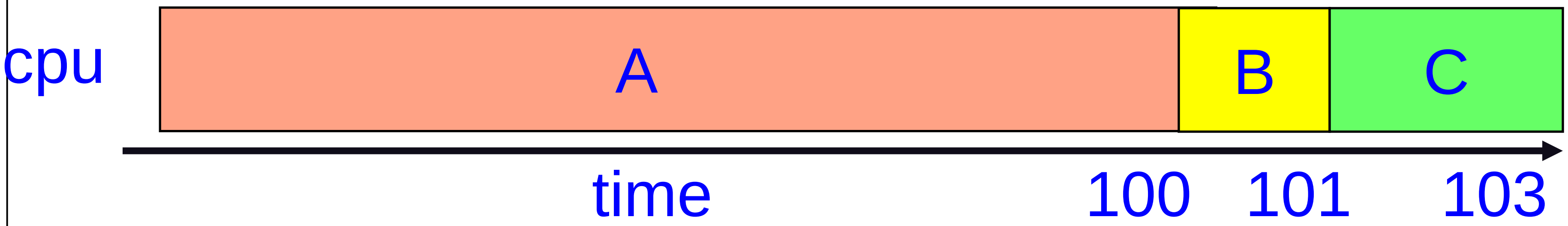
  The Gantt Chart for the schedule is:



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

First come, First serve

- Disadvantage
  - Wait time depends on arrival order
  - Unfair to later jobs
  - (worst case: long job arrives first)
  - It is a troublesome algorithm for time-sharing systems
- Three jobs (times: A=100, B=1, C=2) arrive in the order A, B, C

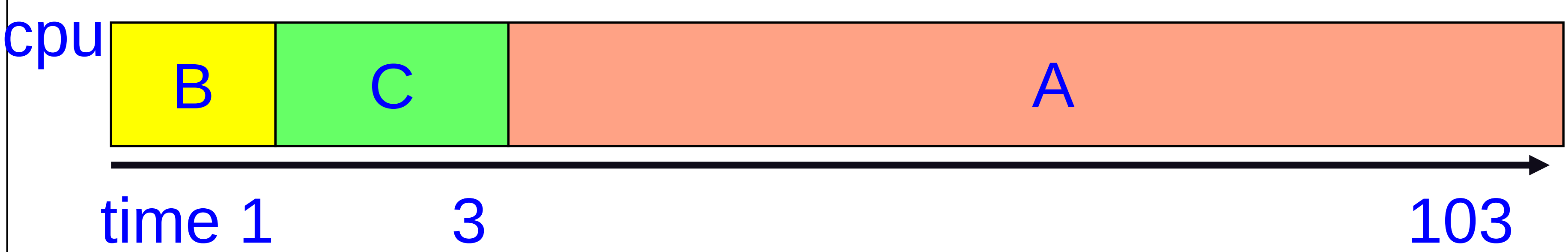


Average Waiting Time $= (0 + 100 + 101) / 3$

$= 67$

First come, First serve

- Now if they arrive in the order B, C, A



Average Waiting Time $= (0 + 1 + 3) / 3$

$= 1.33$

FCFS Convoy effect

- A CPU bound job will hold CPU until
  - Terminates
  - Or it causes an I/O burst
    - Rare occurrence, since the process is CPU-bound
- Long periods where no I/O requests issued, and CPU held
- Result:
  - Poor I/O device utilization
- Penalizes short processes: *Convoy effect* short process behind long process

Activity #1

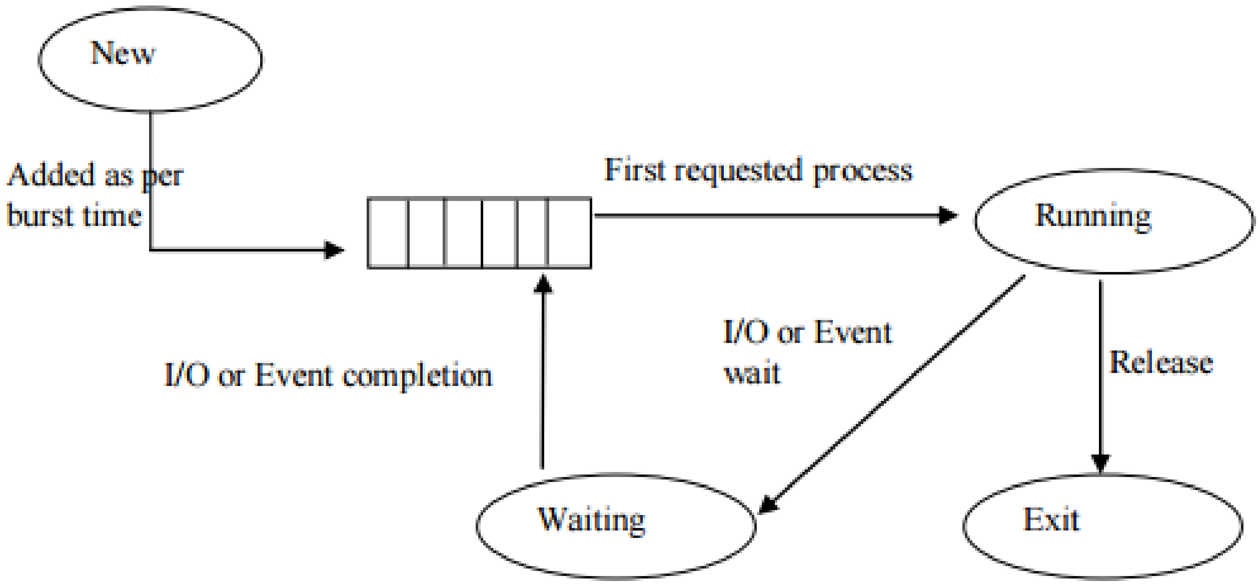
Process	Arrival	Service	Initialize	End	Wait
A	0	3			
B	2	6			
C	4	4			
D	6	5			
E	8	2			

Average waiting time?
Average turn around time?

Shortest-Job-First (SJF) Scheduling

- The SJF algorithm associates with each process the length of its next CPU burst
- When the CPU becomes available, it is assigned to the process that has the smallest next CPU burst (in the case of matching bursts, FCFS is used)
- Two schemes:
  - **Nonpreemptive** – once the CPU is given to the process, it cannot be preempted until it completes its CPU burst
  - **Preemptive** – if a new process arrives with a CPU burst length less than the remaining time of the current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF)

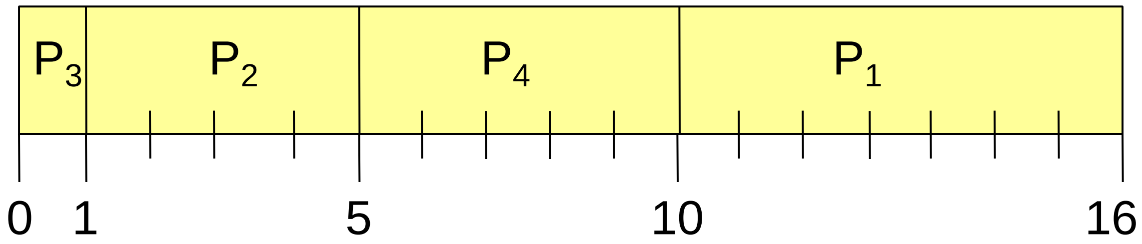
Shortest-Job-First (SJF) Scheduling



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)

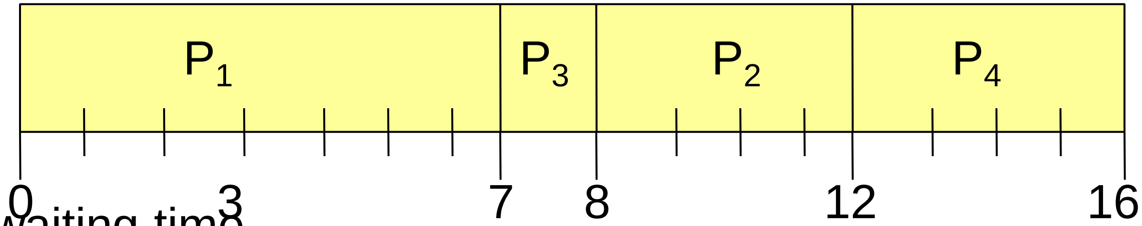


- Average waiting time = (0 + 1 + 5 + 10)/4 = 4
- Average turn-around time = (1 + 5 + 10 + 16)/4 = 8

Example #2: Non- Pre-emptive 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

Shortest Job First(SJF)

- Non-preemptive algorithm.
  - Selects shortest job.
- It can only be applied if duration of each job is known beforehand.
- •SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
- Starvation possibility:
  - If short jobs are continuously arriving, longer jobs never are in position to be executed.

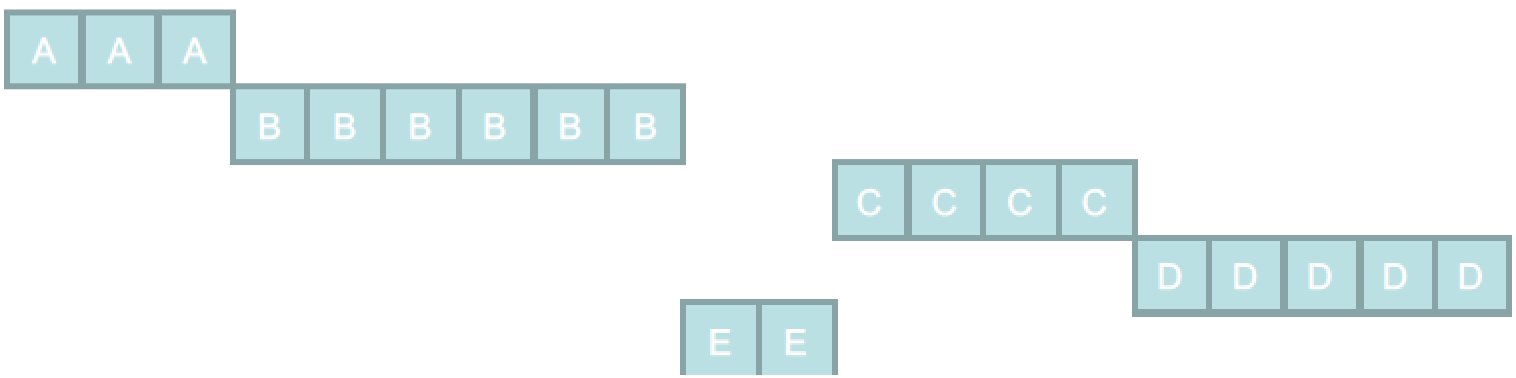
Activity #2

Process	Arrival	Service	Initialize	End	Wait
A	0	3			
B	2	6			
C	4	4			
D	6	5			
E	8	2			

Average waiting time?
Average turn around time?

Activity #2

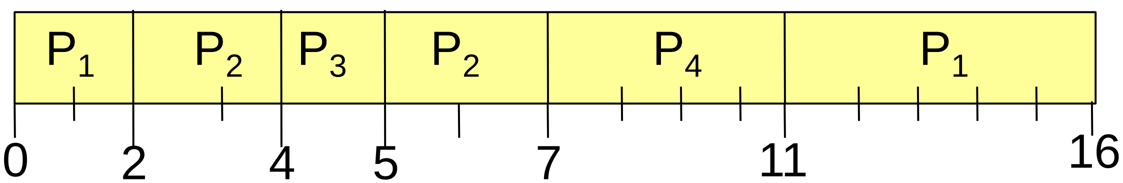
Process	Arrival	Service	Initialize	End	Wait
A	0	3			
B	2	6			
C	4	4			
D	6	5			
E	8	2			



Shortest-remaining-time-first(SRTF)

- Preemptive version of shortest Job next policy

Process	Arrival Time	Burst Time	Remaining Time		
P_1	0.0	7	5	5	5
P_2	2.0	4	2	0	0
P_3	4.0	1	0	0	0
P_4	5.0	4	4	4	0



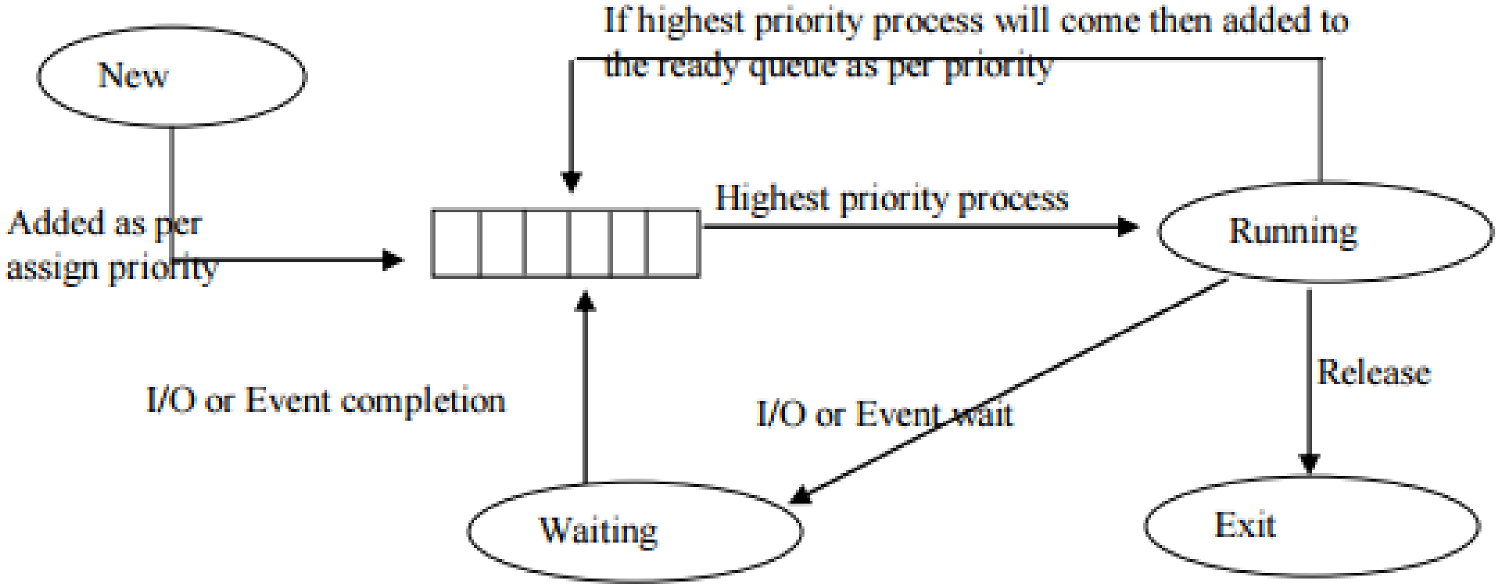
- Average waiting time
  = ([(0 – 0) + (11 - 2)] + [(2 – 2) + (5 – 4)] + (4 - 4) + (7 – 5))/4
  = 9 + 1 + 0 + 2)/4
  = 3

- Average turn-around time = ??

Priority Scheduling

- The OS assigns a fixed priority rank to every process.
- The CPU is allocated to the process with the highest priority (smallest integer = highest priority)
- The scheduler arranges the processes in the ready queue in order of their priority.
- Lower priority processes get interrupted by incoming higher priority processes.
- Characteristics
  - Starvation can happen to the low priority process.
  - The waiting time gradually increases for the equal priority processes .
  - Higher priority processes have smaller waiting time and response time.

Priority Scheduling



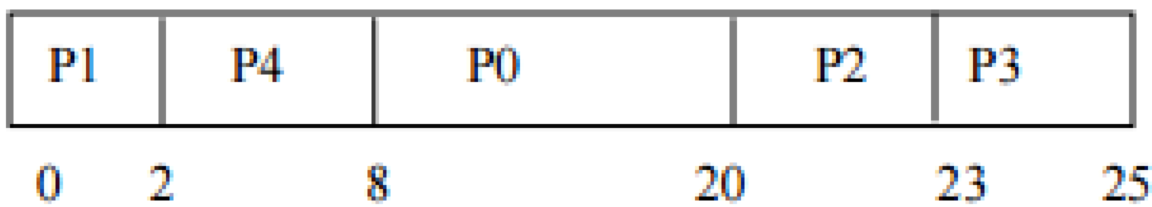
Priority Scheduling

- The SJF algorithm is a special case of the general priority scheduling algorithm
  - priority is the predicted next CPU burst time
- Priorities can be static or dynamic or both
- Among the Processes of equal priority
  - Round robin
  - FCFS

Priority Scheduling

Priority is assigned for each process as follows.

Process ID	Burst Time(ms)	Priority
P0	12	3
P1	2	1
P2	3	3
P3	2	4
P4	6	2



Gantt Chart for priority Scheduling

Waiting Time:

P0=8, P1=0, P2=20,P3=23,P4=2

Avg Waiting Time:8+0+20+23+2/5= 10.6

Avg Turnaround Time:20+2+23+25+8/5=15.6

Priority scheduling

- Priority scheduling can be Preemptive or Non-Preemptive
- When a process arrives and enters the Ready Queue
- Its priority is compared with the currently Running Process
- If Higher
  - Preemptive Scheduling
    - Run the New Thread
  - Non-Preemptive Scheduling
    - Continue running the Current process.

Priority scheduling

- High priority always runs over low priority.
- Starvation
  - A low Priority process may indefinitely wait for the CPU
- Solution: Aging
  - Gradually increase the Priority of processes that wait in the system for a long time.

Round robin (RR)

- Solution to job monopolizing CPU? Interrupt it.
  - Run job for some “time slice,”
  - When time is up, or it blocks
  - It moves to back of a FIFO queue
- Advantage:
  - Fair allocation of CPU across jobs
  - Low average waiting time when job lengths vary

Round robin (RR)

- In the round robin algorithm, each process gets a small unit of CPU time (a *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, 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 $(n\text{-}1)q$ time units.
- Performance of the round robin algorithm
  - q large $\Rightarrow$ FCFS
  - q small $\Rightarrow$ q must be greater than the <u>context switch</u> time; otherwise, the overhead is too high
- One rule of thumb is that 80% of the CPU bursts should be shorter than the time quantum

Round Robin's Disadvantage

- Good for Varying sized jobs
- But what about same-sized jobs?
- Assume 2 jobs of time =100 each:

- Avg completion time?
- (200 + 200) / 2 = 200
- How does this compare with FCFS for same two jobs?
- (100 + 200) / 2 = 150

RR Time slice tradeoffs

- Performance depends on length of the timeslice
- Context switching isn’t a free operation.
- If timeslice time is set too high (attempting to amortize context switch cost)
  - You get FCFS.
  - i.e. Processes will finish or block before their slice is up anyway
- If it’s set too low you’re spending all of your time context switching between threads.

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

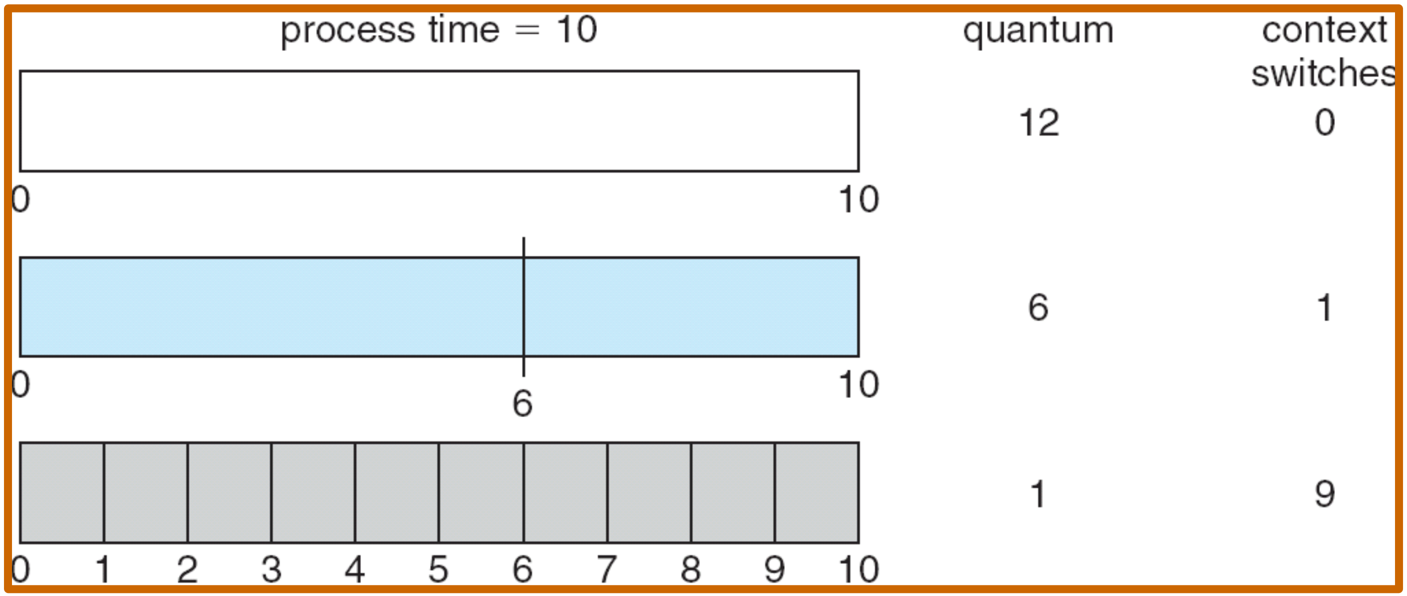
- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

- Typically, higher average turnaround than SJF, but better *response time*
- Average waiting time
  = ([(0 – 0) + (77 - 20) + (121 – 97)] + (20 – 0) + [(37 – 0) + (97 - 57) + (134 – 117)] + [(57 – 0) + (117 – 77)]) / 4
  = (0 + 57 + 24) + 20 + (37 + 40 + 17) + (57 + 40)) / 4
  = (81 + 20 + 94 + 97)/4
  = 292 / 4 = 73
- Average turn-around time = 134 + 37 + 162 + 121) / 4 = 113.5

Time Quantum and Context Switches



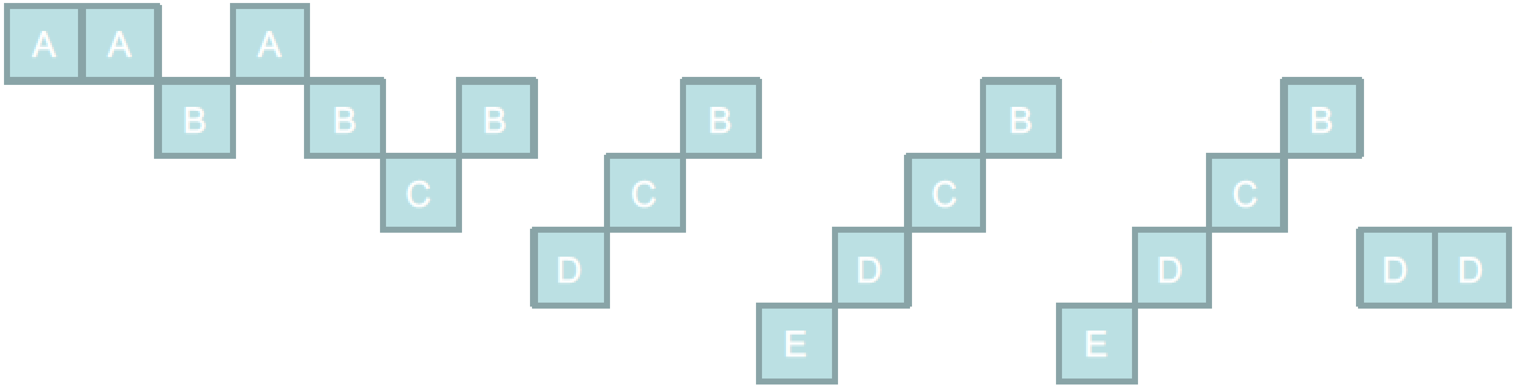
Activity #3 (q=1)

Process	Arrival	Service	Initialize	End	Wait
A	0	3			
B	2	6			
C	4	4			
D	6	5			
E	8	2			

Average waiting time?
Average turn around time?

Activity #3 (q=1)

Process	Arrival	Service	Initialize	End	Wait
A	0	3			
B	2	6			
C	4	4			
D	6	5			
E	8	2			



44

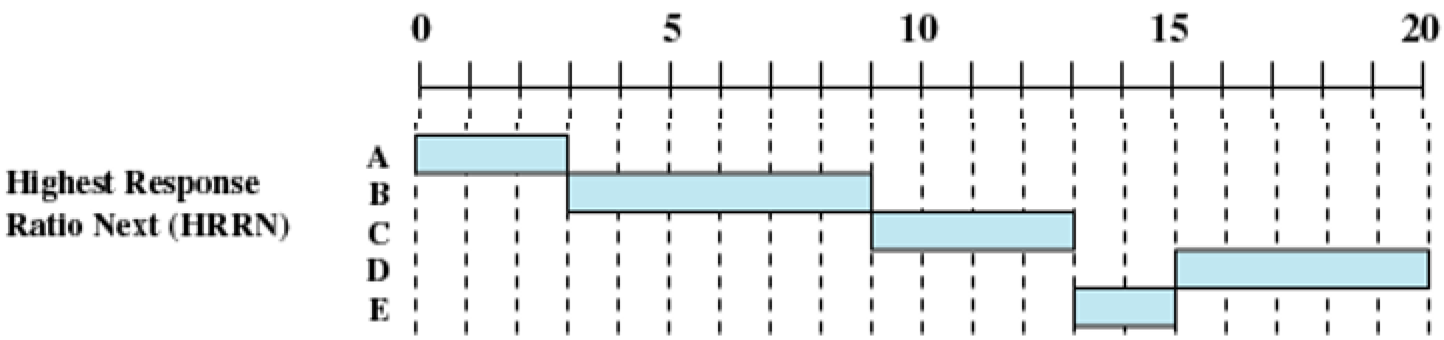
Highest Response Ratio Next (HRRN)

- Choose next process with the greatest ratio
- time spent waiting + expected service time
- R=w+s/s
- R=response ratio
- W=time spent waiting for processor
- S=expected service time
- (HRRN) scheduling is a non-preemptive discipline
- Jobs gain higher priority the longer they wait, which prevents indefinite postponement (process starvation).

Highest Response Ratio Next (HRRN)

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



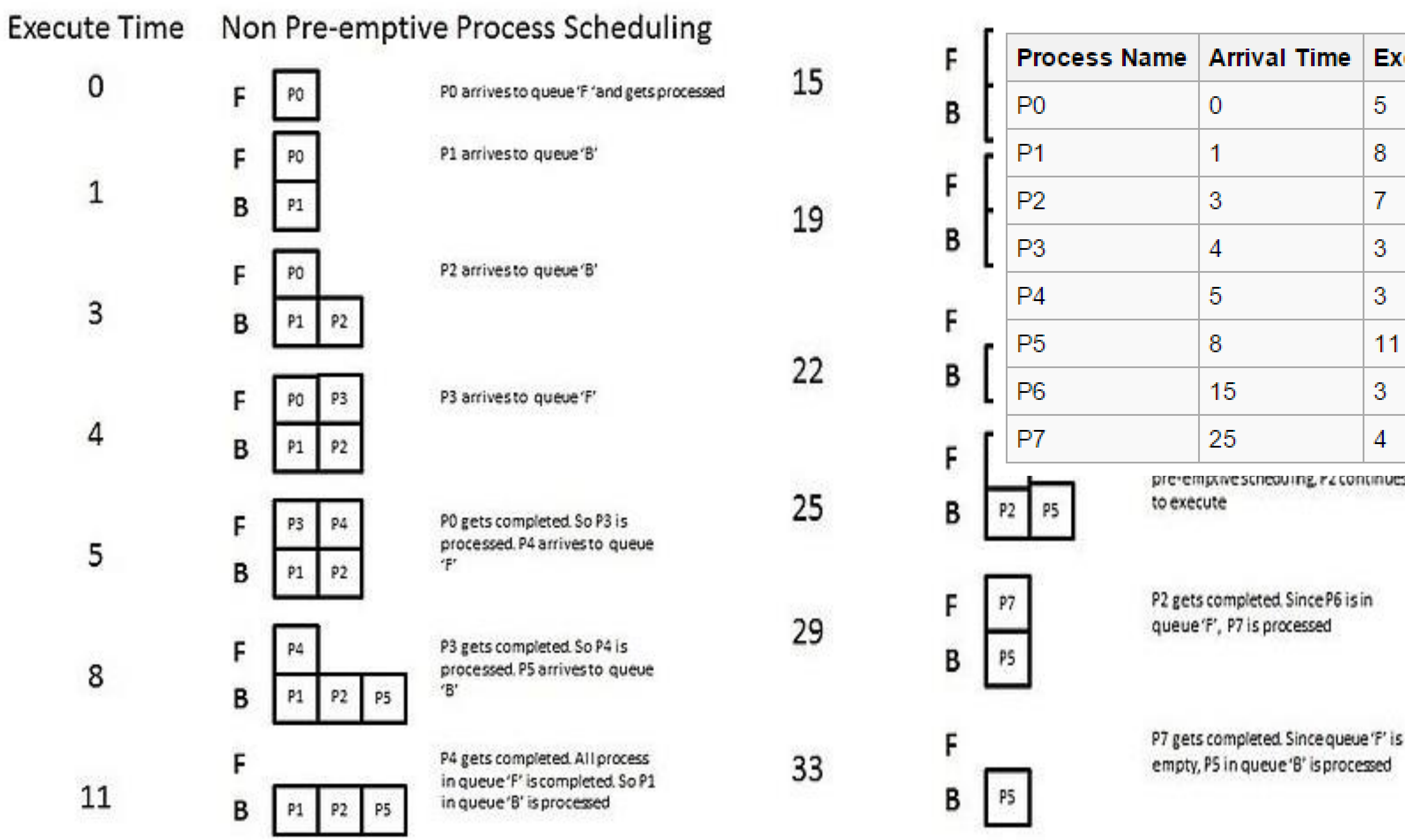
Multi-level Queue Scheduling

- Multi-level queue scheduling is used when processes can be classified into groups
- For example, **foregruound** (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
- 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

Multi-level Queue Scheduling

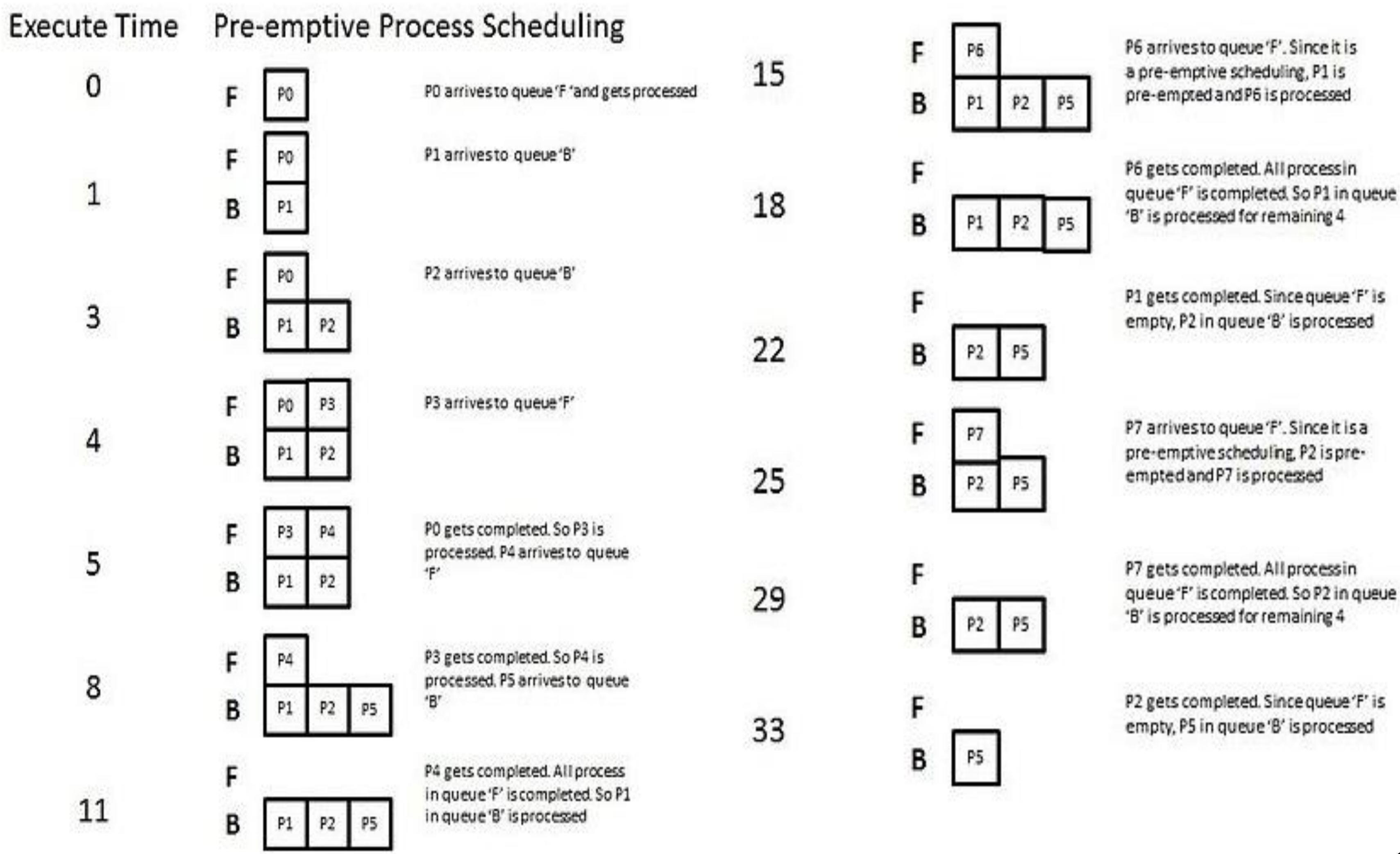
Process Name	Arrival Time	Execute Time	Type
P0	0	5	Foreground
P1	1	8	Background
P2	3	7	Background
P3	4	3	Foreground
P4	5	3	Foreground
P5	8	11	Background
P6	15	3	Foreground
P7	25	4	Foreground

Multi-level Queue Scheduling(non Preemptive)



Process Name	Arrival Time	Execute Time	T
P0	0	5	Fore
P1	1	8	Bac
P2	3	7	Bac
P3	4	3	Fore
P4	5	3	Fore
P5	8	11	Bac
P6	15	3	Fore
P7	25	4	Fore

Multi-level Queue Scheduling

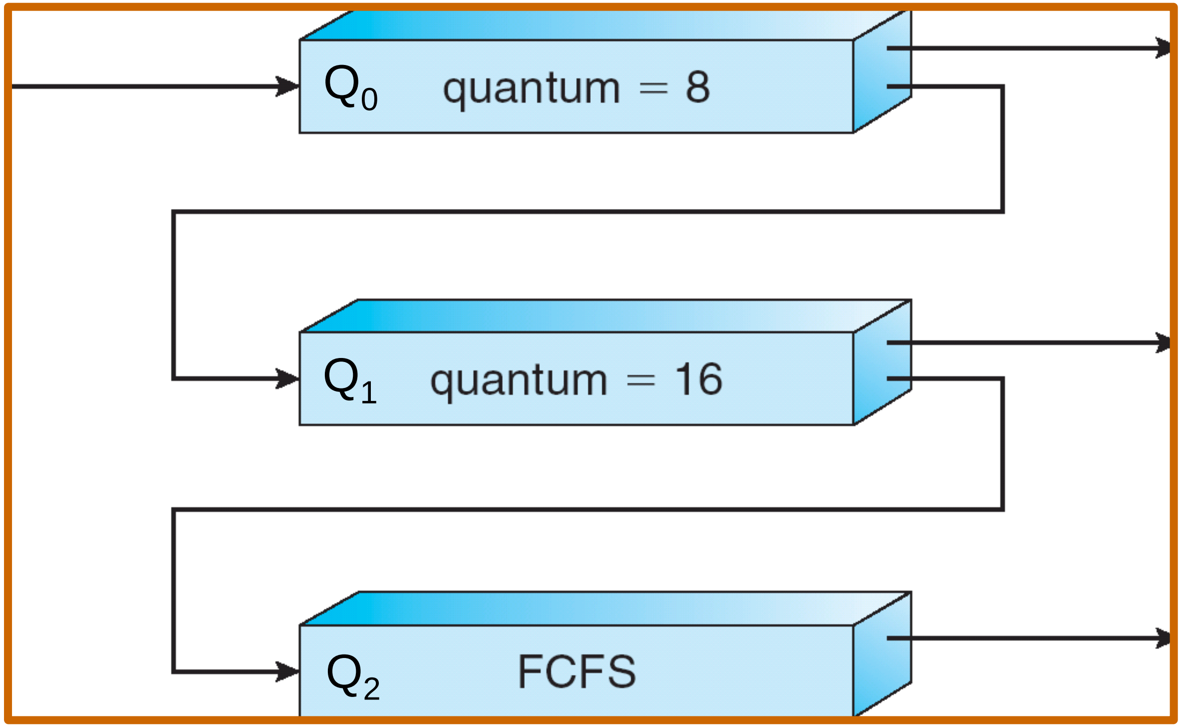


Multilevel Feedback Queue Scheduling

- 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 <u>promote</u> a process
  - Method used to determine when to <u>demote</u> 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 pre-empted and moved to queue Q_2 (FCFS).



Example of Multilevel Feedback Queue

- At the base level queue the processes circulate in round robin fashion until they complete and leave the system.
  - Processes in the base level queue can also be scheduled on a FCFS basis.
- Optionally, if a process blocks for I/O, it is 'promoted' one level, and placed at the end of the next-higher queue. This allows I/O bound processes to be favored by the scheduler and allows processes to 'escape' the base level queue.
- Only if the highest level queue has become empty will the scheduler take up a process from the next lower level queue.
  - The same policy is implemented for picking up in the subsequent lower level queues.
    - Meanwhile, if a process comes into any of the higher level queues, it will preempt a process in the lower level queue.

MLBQ

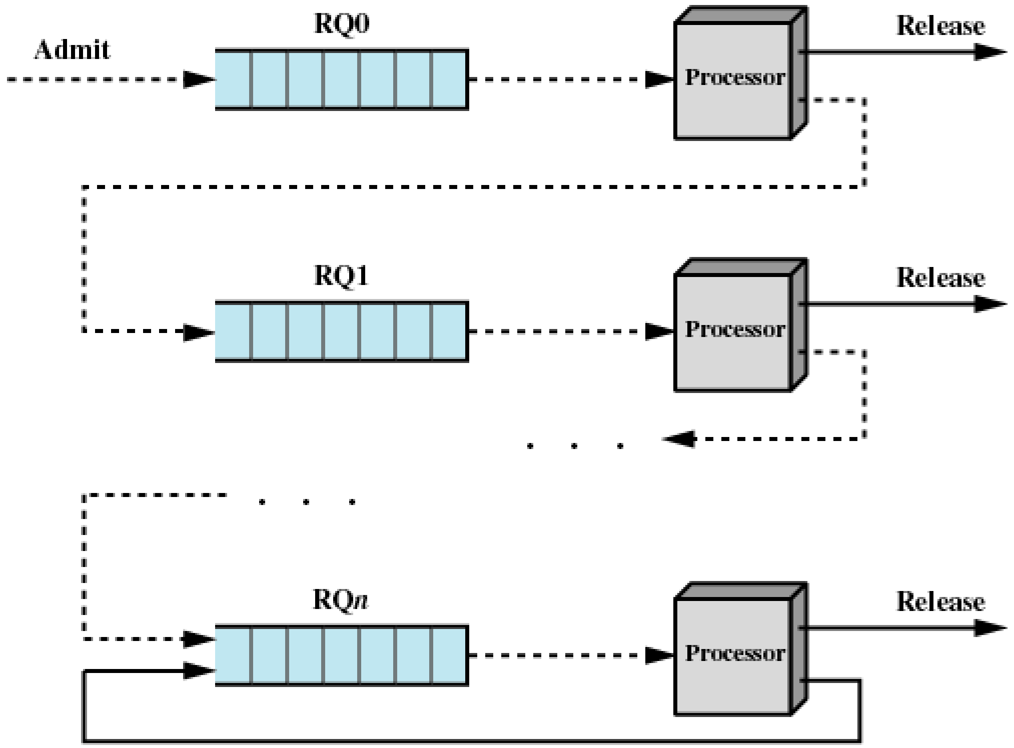


Figure 9.10 Feedback Scheduling

Example of Multilevel Feedback Queue

Process	Burst Time
P1	30
P2	20
P3	10

The system has three RR queues with the following time slices:
1 for queue1, 2 for queue 2, and 4 for queue3.
Draw the Gantt Chart

Lottery scheduling: random simplicity

- **Lottery scheduling! Very simple idea:**
  – give each process some number of lottery tickets
  – On each scheduling event, randomly pick ticket
  – run winning process
- **How to use?**
  – Approximate priority: low-priority, give few tickets, high-priority give many
  – Approximate SJF: give short jobs more tickets, long jobs fewer. Key: If job has at least 1, will not starve
  – Degrades gracefully as load changes. Adding or deleting a job affects all jobs proportionately, independent of the number of tickets a job has.

Lottery scheduling

- Example: give all jobs 1/n of cpu?
  - 4 jobs, 1 ticket each

  - each gets (on average) 25% of CPU.
  - Delete one job:

  - automatically adjusts to 33% of CPU!

Lottery scheduling

- Short jobs get 10 tickets, long jobs get 1 ticket each.

# short jobs/ # long jobs	% of CPU each short job gets	% of CPU each long job gets
1/1	91%	9%
0/2		
2/0		
10/1		
1/10		

Lottery scheduling

- Short jobs get 10 tickets, long jobs get 1 ticket each.

# short jobs/ # long jobs	% of CPU each short job gets	% of CPU each long job gets
1/1	91% (10/11)	9% (1/11)
0/2		50% (1/2)
2/0	50% (10/20)	
10/1	10% (10/101)	< 1% (1/101)
1/10	50% (10/20)	5% (1/20)

Example

- Three threads
  - A has 5 tickets
  - B has 3 tickets
  - C has 2 tickets
- If all compete for the resource
  - B has 30% chance of being selected
- If only B and C compete
  - B has 60% chance of being selected

- Reading Assignment: Is Lottery scheduling is ***starvation-free?***
-

5.4 Multiple-Processor Scheduling

Multiple-Processor Scheduling

- If multiple CPUs are available, load sharing among them becomes possible; the scheduling problem becomes more complex
- We concentrate in this discussion on systems in which the processors are identical (homogeneous) in terms of their functionality
  - We can use any available processor to run any process in the queue
- Two approaches: **Asymmetric** processing and **symmetric** processing (see next slide)

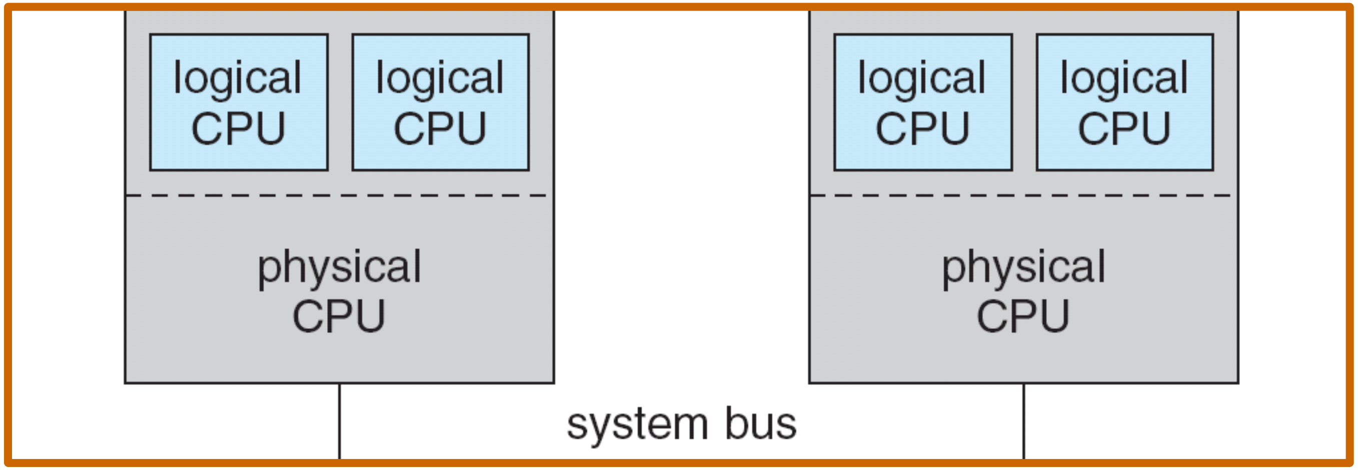
Multiple-Processor Scheduling

- *Asymmetric multiprocessing (ASMP)*
  - One processor handles all scheduling decisions, I/O processing, and other system activities
  - The other processors execute only user code
  - Because only one processor accesses the system data structures, the need for data sharing is reduced
- Symmetric multiprocessing (SMP)
  - Each processor schedules itself
  - All processes may be in a common ready queue or each processor may have its own ready queue
  - Either way, each processor examines the ready queue and selects a process to execute
  - Efficient use of the CPUs requires load balancing to keep the workload evenly distributed
    - In a **Push** migration approach, a specific task regularly checks the processor loads and redistributes the waiting processes as needed
    - In a **Pull** migration approach, an idle processor pulls a waiting job from the queue of a busy processor
  - Virtually all modern operating systems support SMP, including Windows XP, Solaris, Linux, and Mac OS X

Symmetric Multithreading

- Symmetric multiprocessing systems allow several threads to run concurrently by providing multiple physical processors
- An alternative approach is to provide multiple **logical** rather than **physical** processors
- Such a strategy is known as symmetric multithreading (SMT)
  - This is also known as hyperthreading technology
- The idea behind SMT is to create multiple logical processors on the same physical processor
  - This presents a view of several logical processors to the operating system, even on a system with a single physical processor
  - Each logical processor has its own **architecture state**, which includes general-purpose and machine-state registers
  - Each logical processor is responsible for its own interrupt handling
  - However, each logical processor shares the resources of its physical processor, such as cache memory and buses
- SMT is a feature provided in the hardware, not the software
  - The hardware must provide the representation of the architecture state for each logical processor, as well as interrupt handling (see next slide)

A typical SMT architecture



SMT = Symmetric Multi-threading