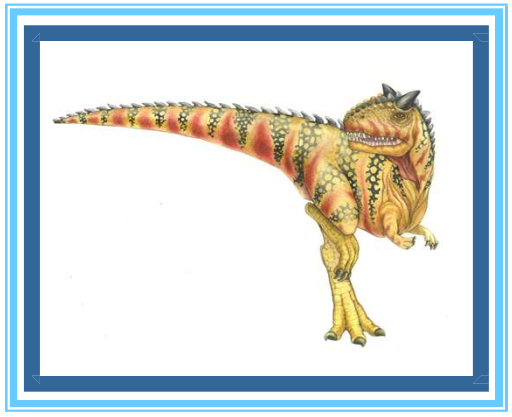
Chapter 6: CPU Scheduling



Outline

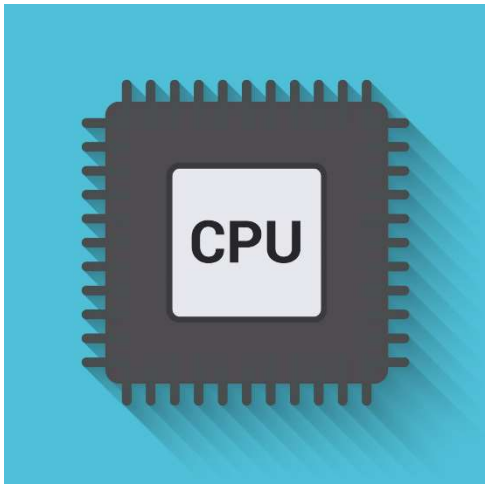
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Algorithm Evaluation

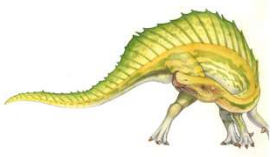
Objectives

- Describe various **CPU scheduling algorithms**
- **Assess** CPU scheduling algorithms based on scheduling criteria
- Apply modeling and simulations to **evaluate** CPU scheduling algorithms

Scheduling

- An OS must allocate resources amongst competing processes.
- The resource provided by a processor is execution time.
- The resource is allocated by means of scheduling
  - determines which processes will wait and which will progress.
- The aim of processor scheduling is to assign processes to be executed by the processor over time, in a way that meets system **objectives:**
  - response time, throughput, and processor efficiency.

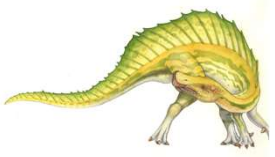




Scheduling Objectives

The scheduling function should

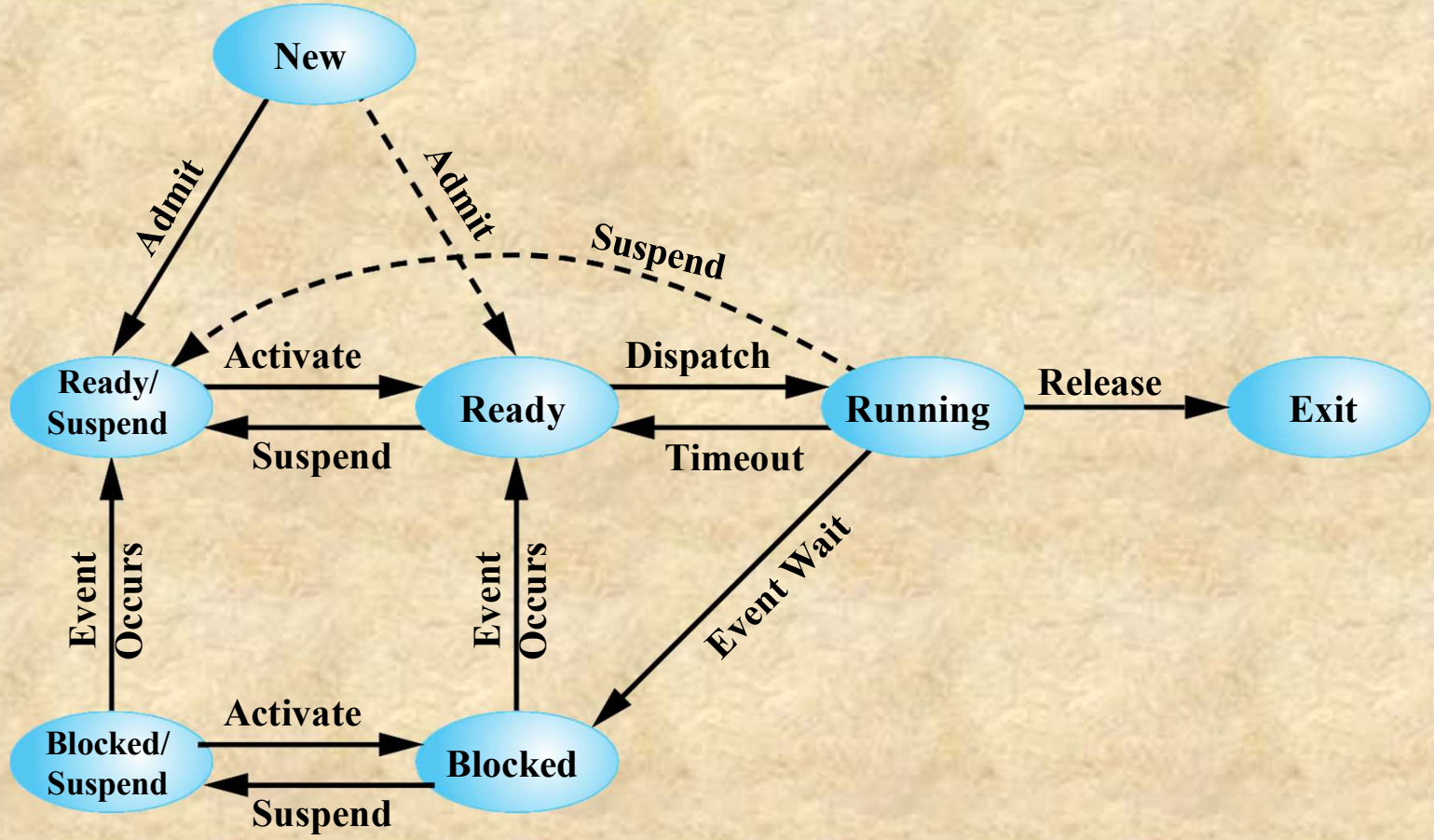
- Share time **fairly** among processes
- Prevent starvation of a process
- Use the processor efficiently
- Have low overhead
- Prioritise processes when necessary (e.g. real time deadlines)



Types of Scheduling

- **Long-term scheduling** is performed when a new process is created.
  - This is a decision whether to add a new process to the set of processes that are currently active.
- **Medium-term scheduling** is a part of the swapping function.
  - This is a decision whether to add a process to those that are at least partially in main memory and therefore are available for execution.
- **Short-term scheduling** is the actual decision of which ready process to execute next. Known as the dispatcher.

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



(b) With Two Suspend States

Figure 3.9 Process State Transition Diagram with Suspend States

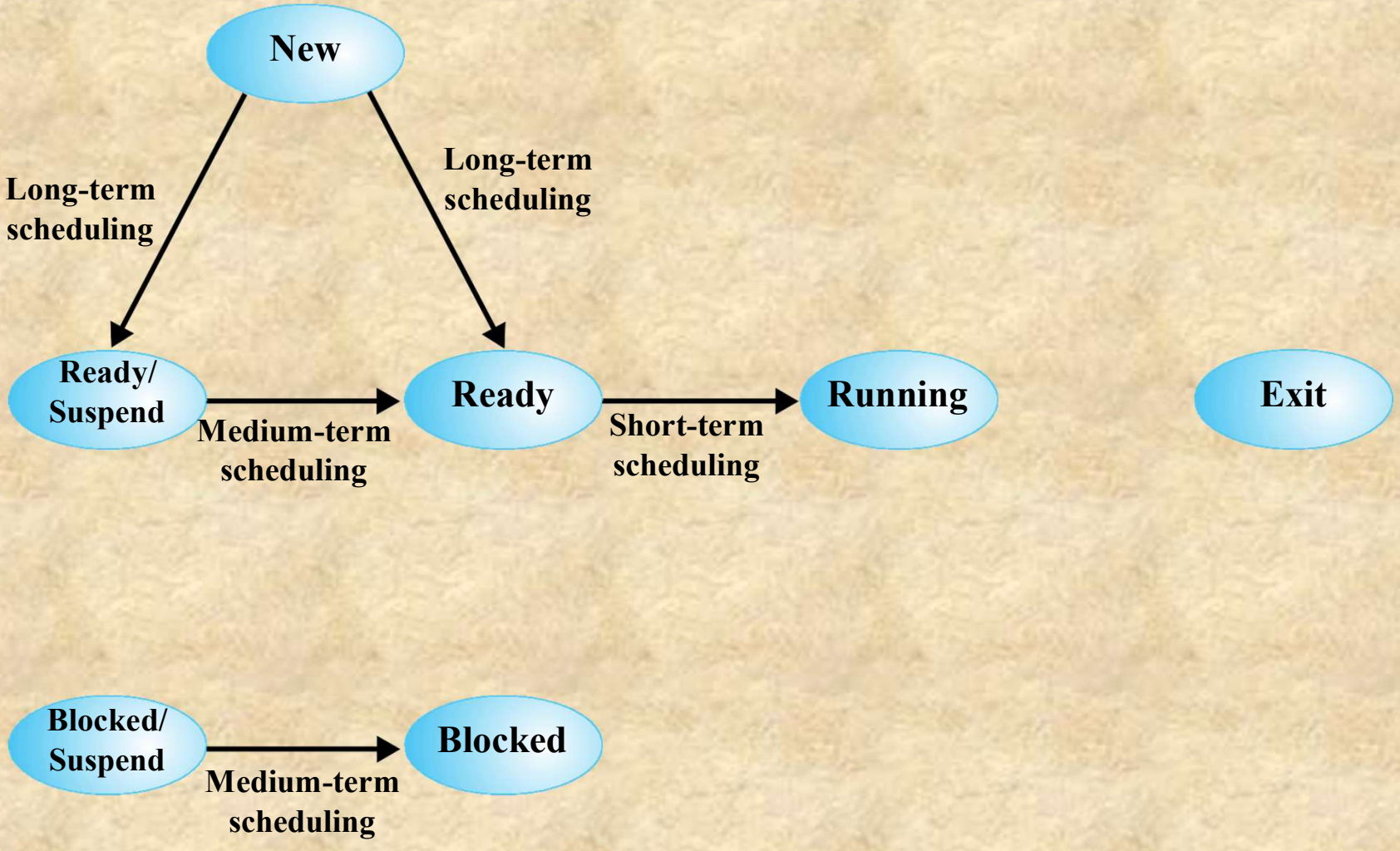


Figure 9.1 Scheduling and Process State Transitions

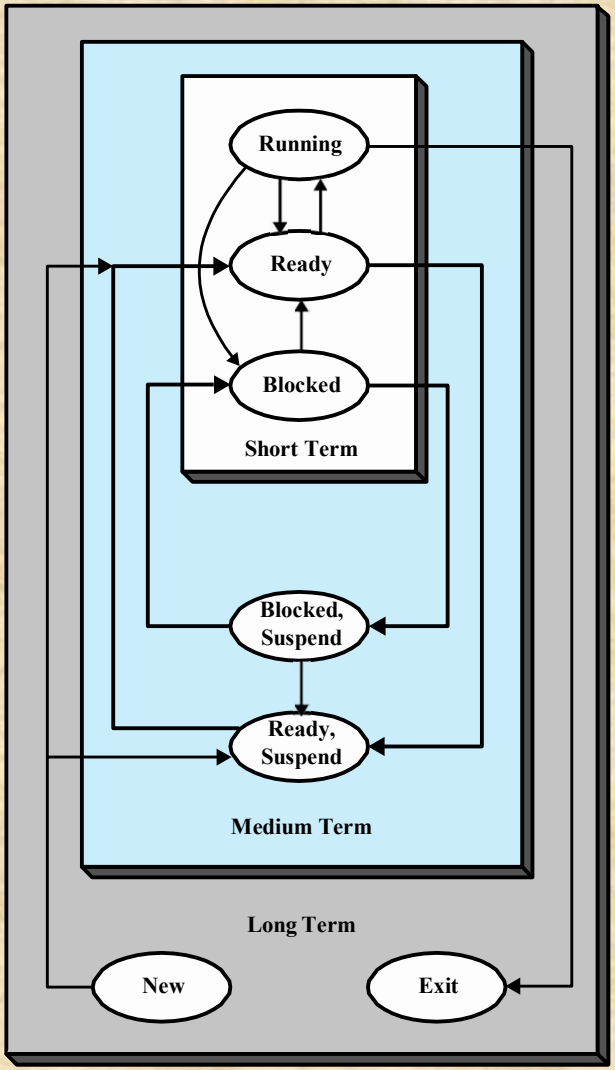
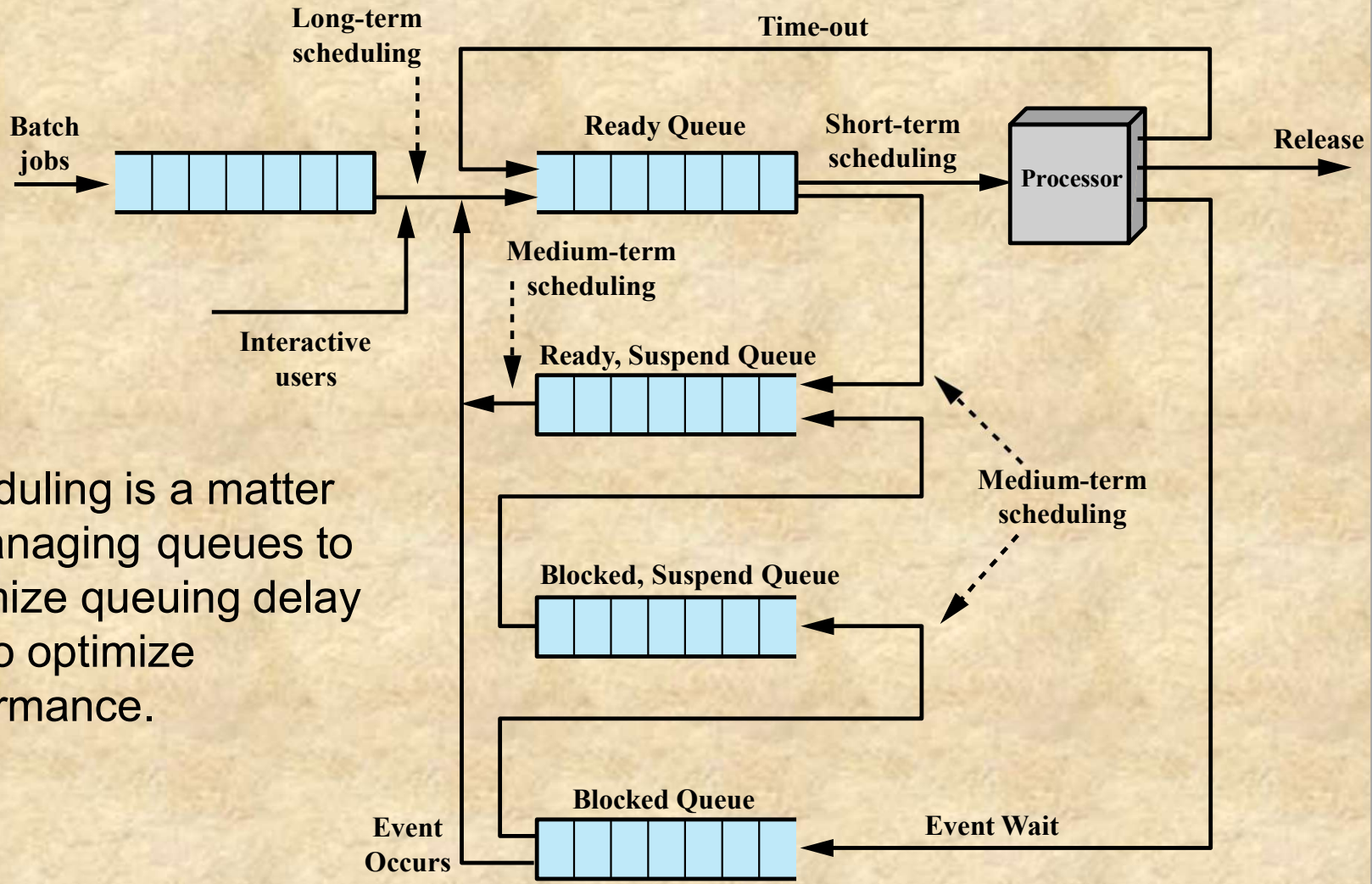


Figure 9.2 Levels of Scheduling

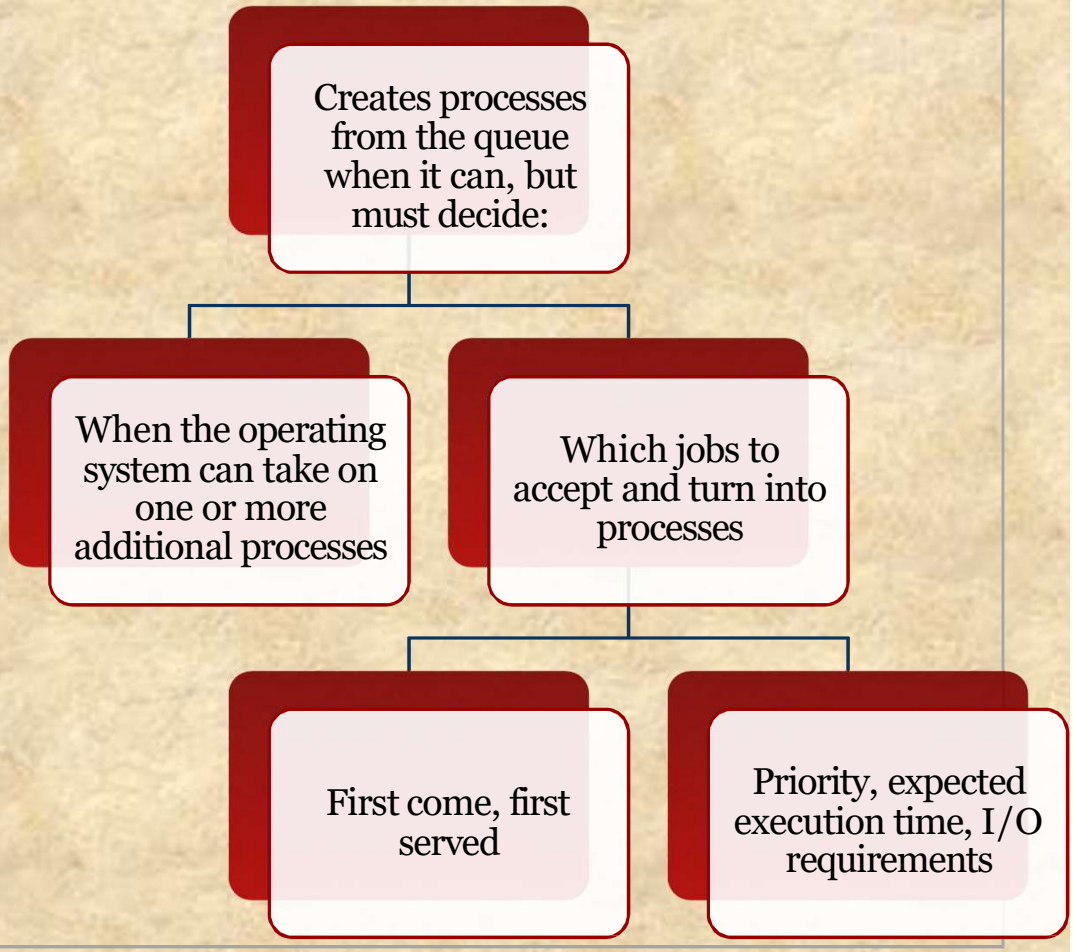


Scheduling is a matter of managing queues to minimize queuing delay and to optimize performance.

Figure 9.3 Queuing Diagram for Scheduling

Long-Term Scheduler

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
  - The more processes that are created, the smaller the percentage of time that each process can be executed
  - May limit to provide satisfactory service to the current set of processes



Medium-Term Scheduling

- Part of the swapping function
- Swapping-in decisions are based on the need to manage the degree of multiprogramming
  - Considers the memory requirements of the swapped-out processes

Short-Term(CPU) Scheduling

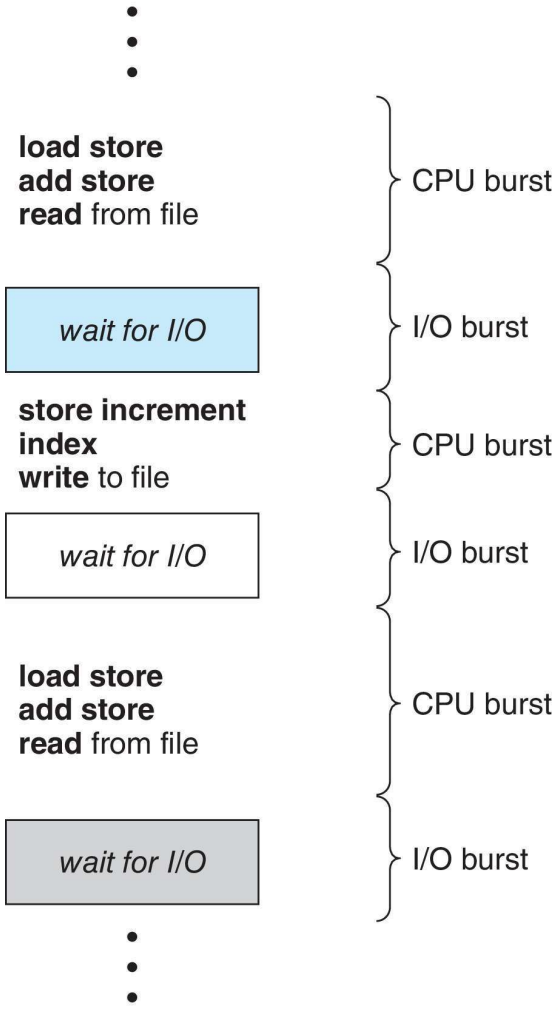
- Known as the dispatcher
- Executes most frequently (reason to call it short-term scheduling)
- Select from among the processes in ready queue, and allocate the CPU to one of them.
- Makes the fine-grained decision of which process to execute next
- Invoked when an event occurs that may lead to the blocking of the current process or that may provide an opportunity to preempt a currently running process in favor of another

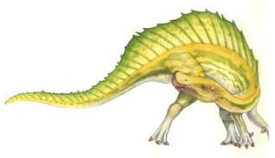
Examples:

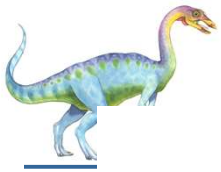
- Clock interrupts
- I/O interrupts
- Operating system calls
- Signals (e.g., semaphores)

Basic Concepts

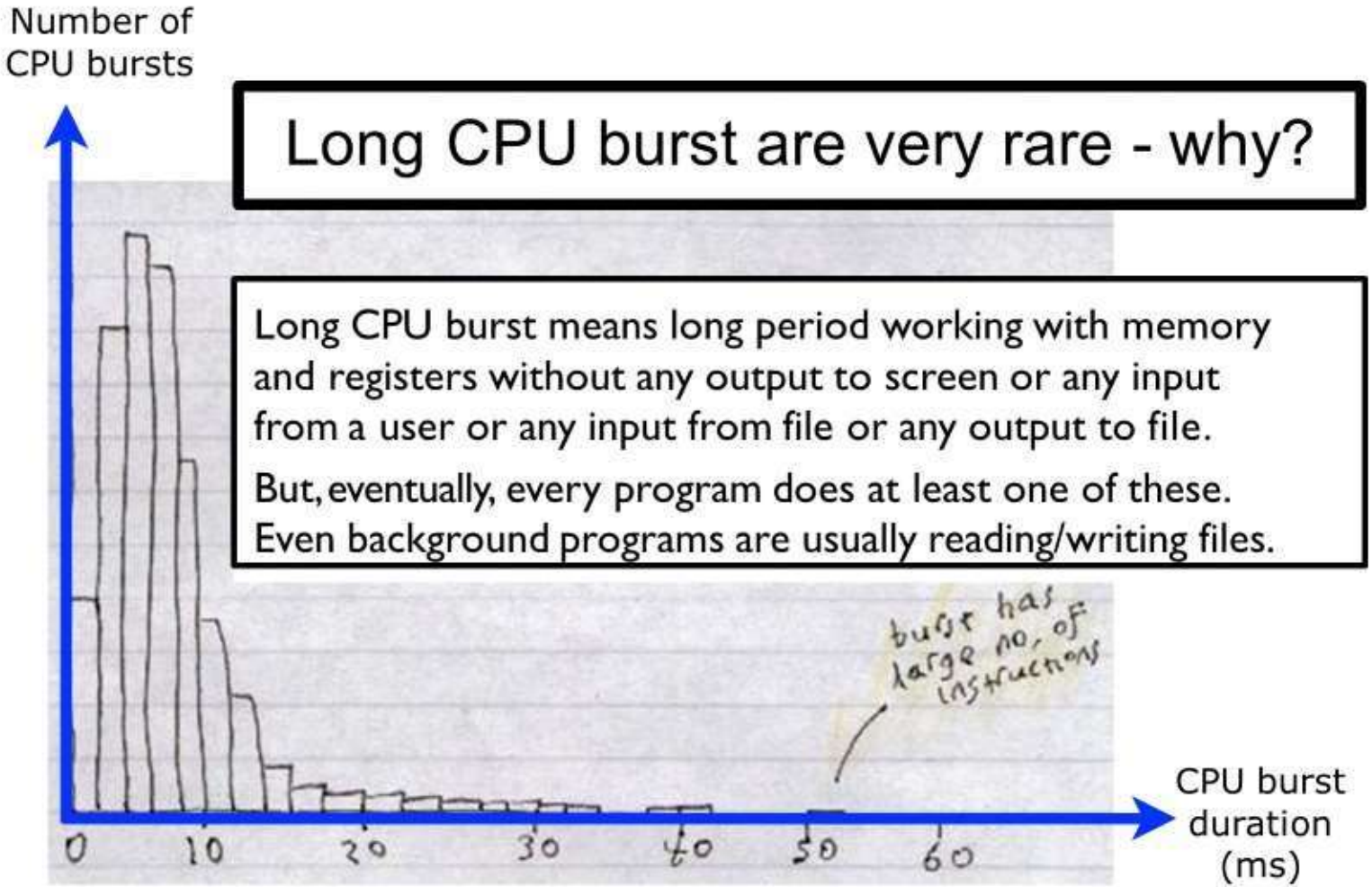
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern







Histogram of CPU-burst times



Nonpreemptive vs Preemptive

Nonpreemptive

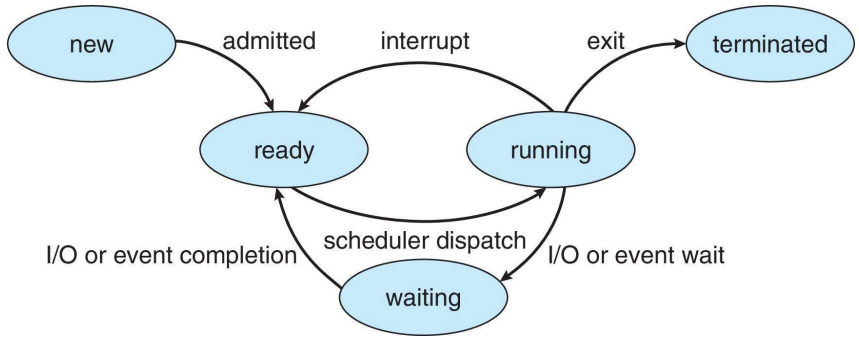
- Once a process is in the running state, it will continue until it terminates or blocks itself for I/O

Preemptive

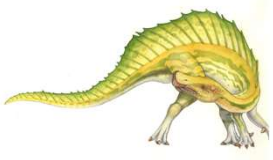
- Currently running process may be interrupted and moved to ready state by the OS
- Decision to preempt may be performed when a new process arrives, when an interrupt occurs that places a blocked process in the Ready state, or periodically, based on a clock interrupt

CPU Scheduler

- The **CPU scheduler** selects from among the processes in ready queue, and allocates a CPU core to one of them
  - Queue may be ordered in various ways
- When is CPU scheduler invoked?
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready (maybe the process has a higher priority compare to the one is currently running)
  4. Terminates
- For situations 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution.
- For situations 2 and 3, however, there is a choice.

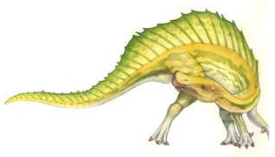


Scheduling under 1 and 4 is **non-preemptive**. It means the process decides to leave the CPU by **itself** .
All other scheduling is **preemptive**. It means the OS forces the process to leave the CPU.



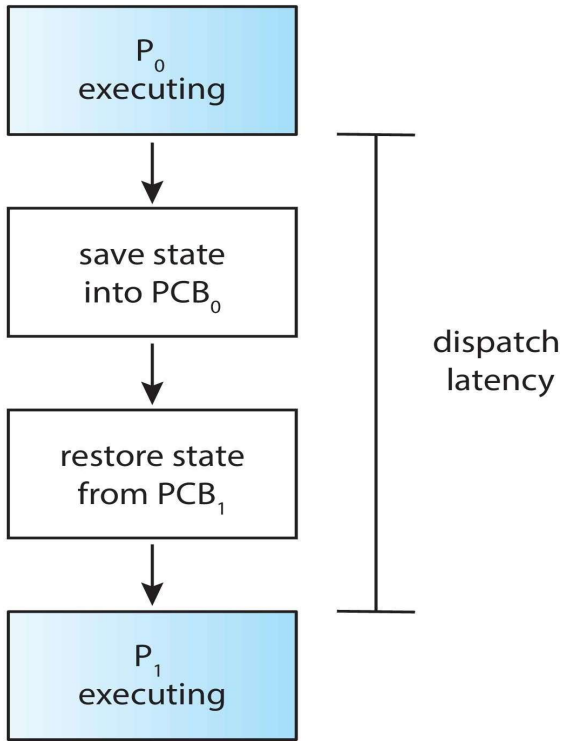
Preemptive and Nonpreemptive Scheduling

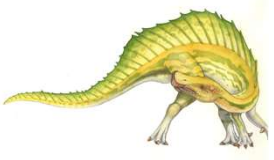
- When scheduling takes place only under circumstances 1 and 4, the scheduling scheme is **nonpreemptive**.
- Otherwise, it is **preemptive**.
- Under Nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by **terminating** or by **switching** to the waiting state.
- Virtually all modern operating systems including Windows, MacOS, Linux, and UNIX use preemptive scheduling algorithms.



Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the CPU scheduler; this involves:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running





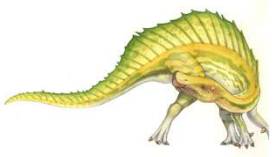
Scheduling Criteria

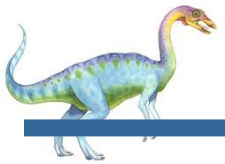
User oriented:

- **Turnaround time** – amount of time to execute a particular process
- **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).
- **Waiting time** – amount of time a process has been waiting in the ready queue

System oriented:

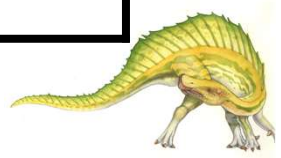
- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit





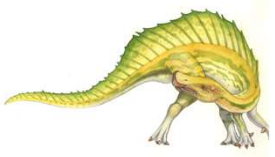
Scheduling criteria

Criteria	Definition	Goal
CPU utilization	The % of time the CPU is executing user level process code.	Maximize
Throughput	Number of processes that complete their execution per time unit.	Maximize
Turnaround time	Amount of time to execute a particular process.	Minimize
Waiting time	Amount of time a process has been waiting in the ready queue.	Minimize
Response time	Amount of time it takes from when a request was submitted until the first response is produced.	Minimize

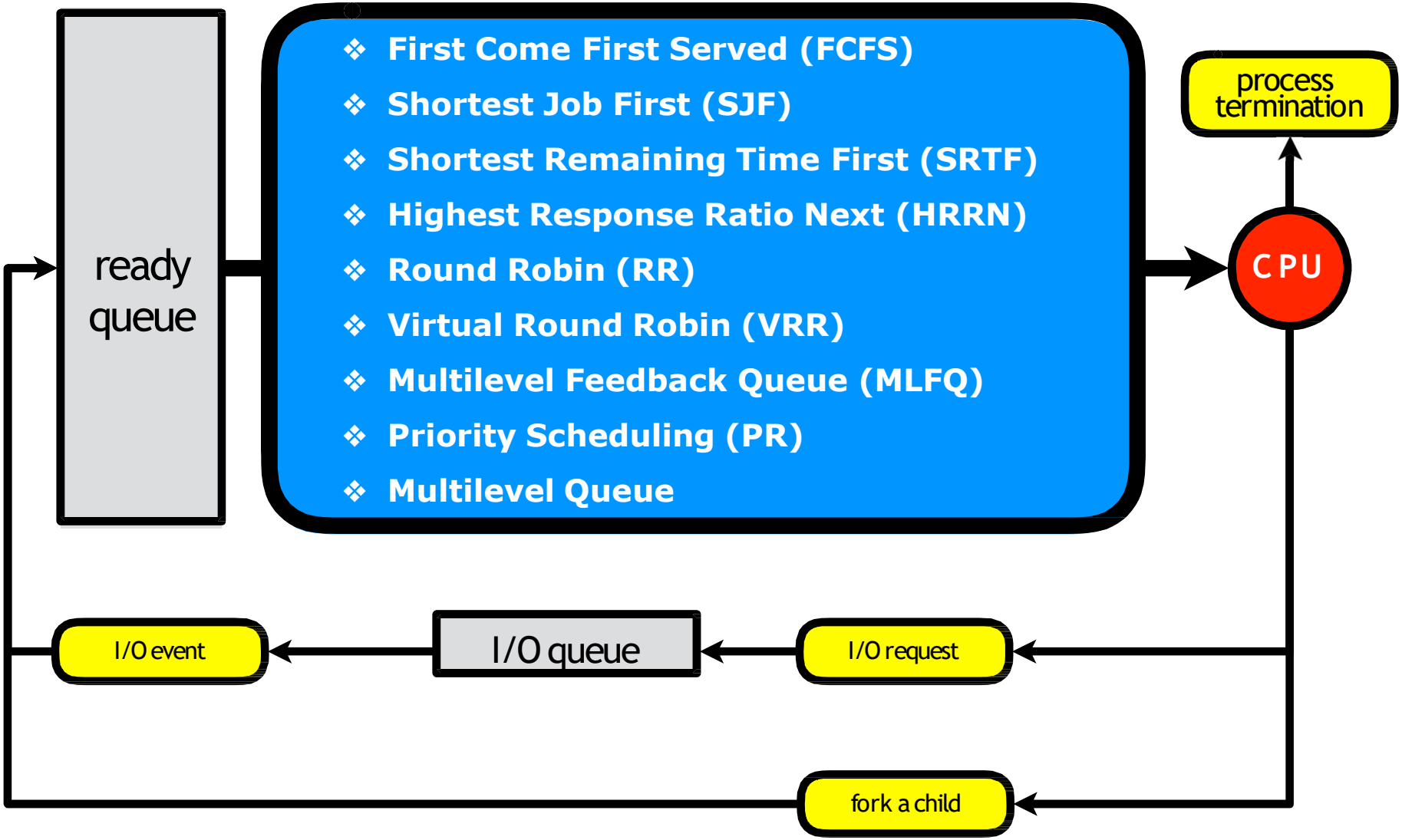


Scheduling Algorithms

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
- Highest Response Ratio Next (HRRN)
- Round Robin (RR)
- Virtual Round Robin (VRR)
- Multilevel Feedback Queue (MLFQ)
- Priority Scheduling (PR)
- Multilevel Queue

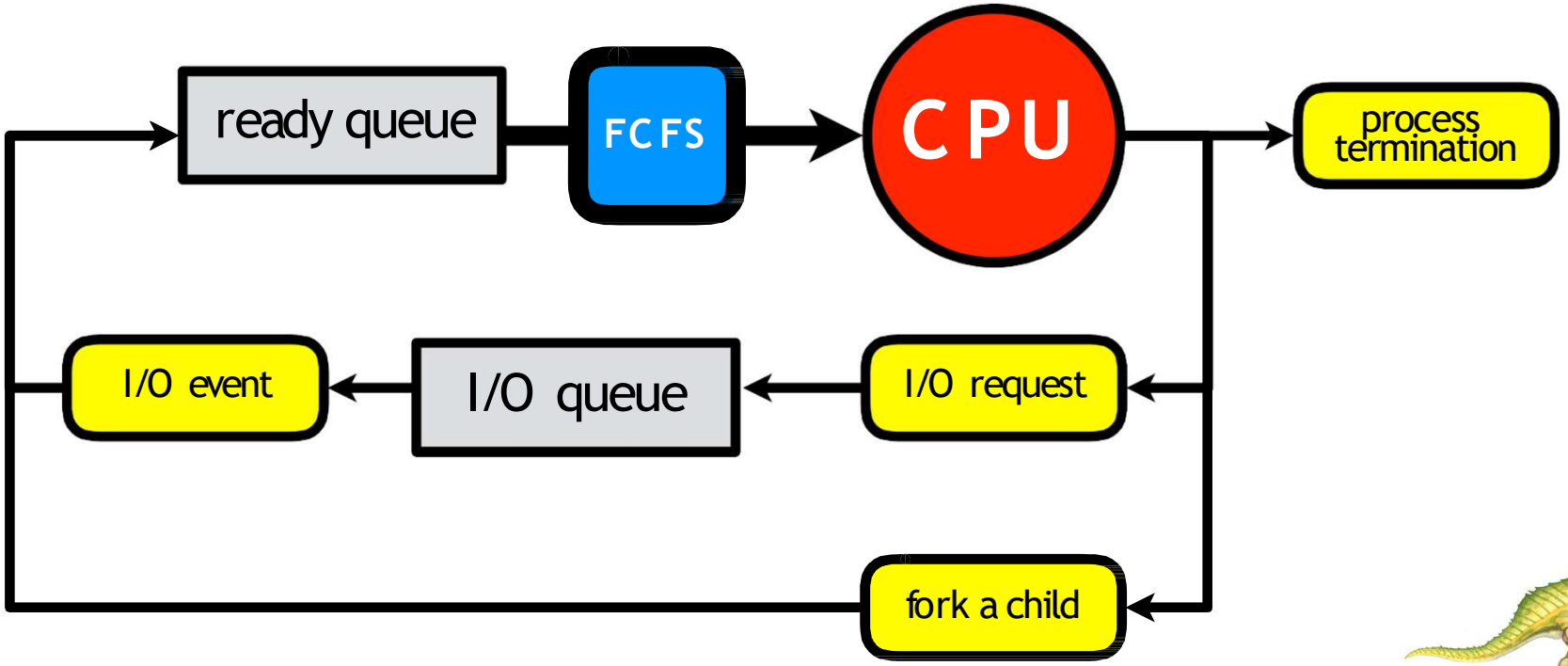


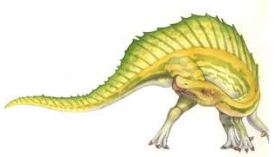
Scheduling algorithms



First-Come, First-Served (FCFS)

The first come, first served (commonly called FIFO – first in, first out) process scheduling algorithm is the simplest process scheduling algorithm. Processes are executed on the CPU in the same order they arrive to the ready queue.





Arrival Time: Time at which the process arrives in the ready queue.

Completion Time: Time at which process completes its execution.

Burst Time: Time required by a process for CPU execution.

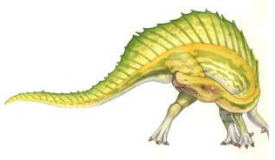
Turn Around Time: Time Difference between completion time and arrival time.

Turn Around Time = Completion Time – Arrival Time

Waiting Time (W.T): Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time – Burst Time

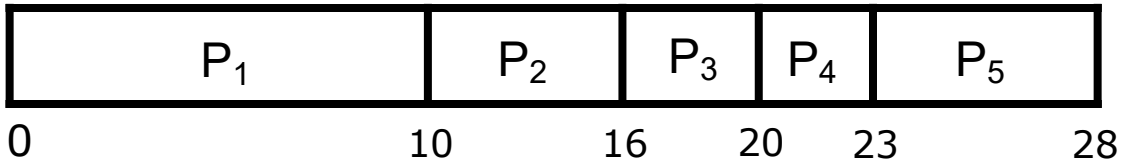
Response Time: Time at which the process received its first response.



First- Come, First-Served (FCFS) Scheduling

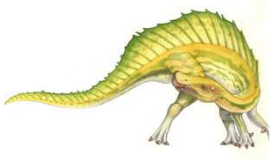
Process	Arrival Time	CPU Burst Time
P_1	0	10
P_2	3	6
P_3	3	4
P_4	8	3
P_5	13	5

Gantt Chart:



$$\text{Avg Response Time} = \frac{10+13+17+15+15}{5} = 14$$

$$\text{Avg Waiting Time} = \frac{0+7+13+12+10}{5} = 8.4$$



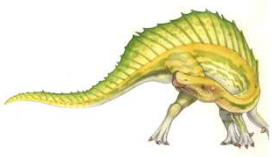
FCFS Scheduling (Cont.)

Advantages:

- FCFS is simple and easy

Disadvantages:

- FCFS is **non-preemptive**
- FCFS performs much better for long processes(CPU bound) than short ones(I/O bound).
- Favors CPU bound processes.
  - I/O-bound processes have to wait until CPU-bound process completes
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes



PID	P3	P2	P1
CPU burst time	3	3	24

The processes arrive to the ready queue in the order: P1, P2, P3.

Gantt Chart for the FCFS schedule

PID	Waiting time
P1	0
P2	24
P3	27

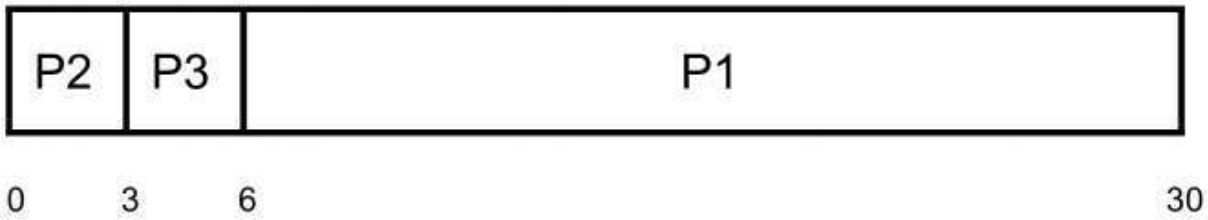
Average waiting time

$(0 + 24 + 27)/3 = 17$

PID	P1	P3	P2
CPU burst time	24	3	3

What if the same processes arrive to the ready queue in the order: P2, P3, P1.

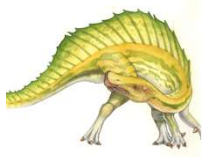
Gantt Chart for the FCFS schedule



PID	Waiting time
P1	6
P2	0
P3	3

Average waiting time

$(6 + 0 + 3)/3 = 3$



The convoy effect

When using FCFS scheduling, if I/O bound (short CPU burst) processes are scheduled after CPU bound (long CPU burst) processes, the average waiting time increases.

Short CPU burst

Short CPU burst

Long CPU burst

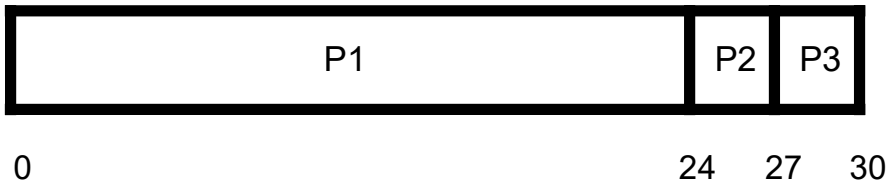
The convoy effect

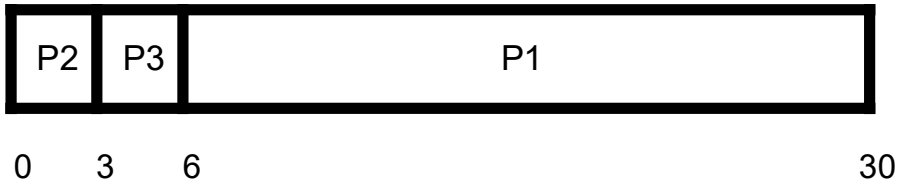
When using FCFS scheduling, if I/O bound (short CPU burst) processes are scheduled after CPU bound (long CPU burst) processes, the average waiting time increases.

PID	P3	P2	P1
CPU burst time	3	3	24

PID	P1	P3	P2
CPU burst time	24	3	3

Gantt Charts for the FCFS schedules





PID	Waiting time
P1	0
P2	24
P3	27

Average waiting time

(0 + 24 + 27) / 3 = 17

PID	Waiting time
P1	6
P2	0
P3	3

Average waiting time

(6 + 0 + 3) / 3 = 3

What is the optimal schedule?

To answer this question we must first define what we mean with optimal.

A better question

What schedule minimizes the average waiting time?

In general, if we have CPU bursts $x_1, \ldots x_n$, calculate the total waiting time T_{wait}.

$$
\begin{aligned}
T_{wait} = {} & 0 + \\
& x_1 + \\
& x_1 + x_2 + \\
& x_1 + x_2 + x_3 + \\
& \quad \ldots \quad + \\
& x_1 + x_2 + x_3 + \ldots + x_{n-1} \\
= {} & (n-1)x_1 + (n-2)x_2 + \ldots + x_{n-1}
\end{aligned}
$$

Now, calculate the average waiting time.

$$\mathbf{Average}(T_{wait}) = [(n-1)x_1 + (n-2)x_2 + \ldots + x_{n-1}]/n$$

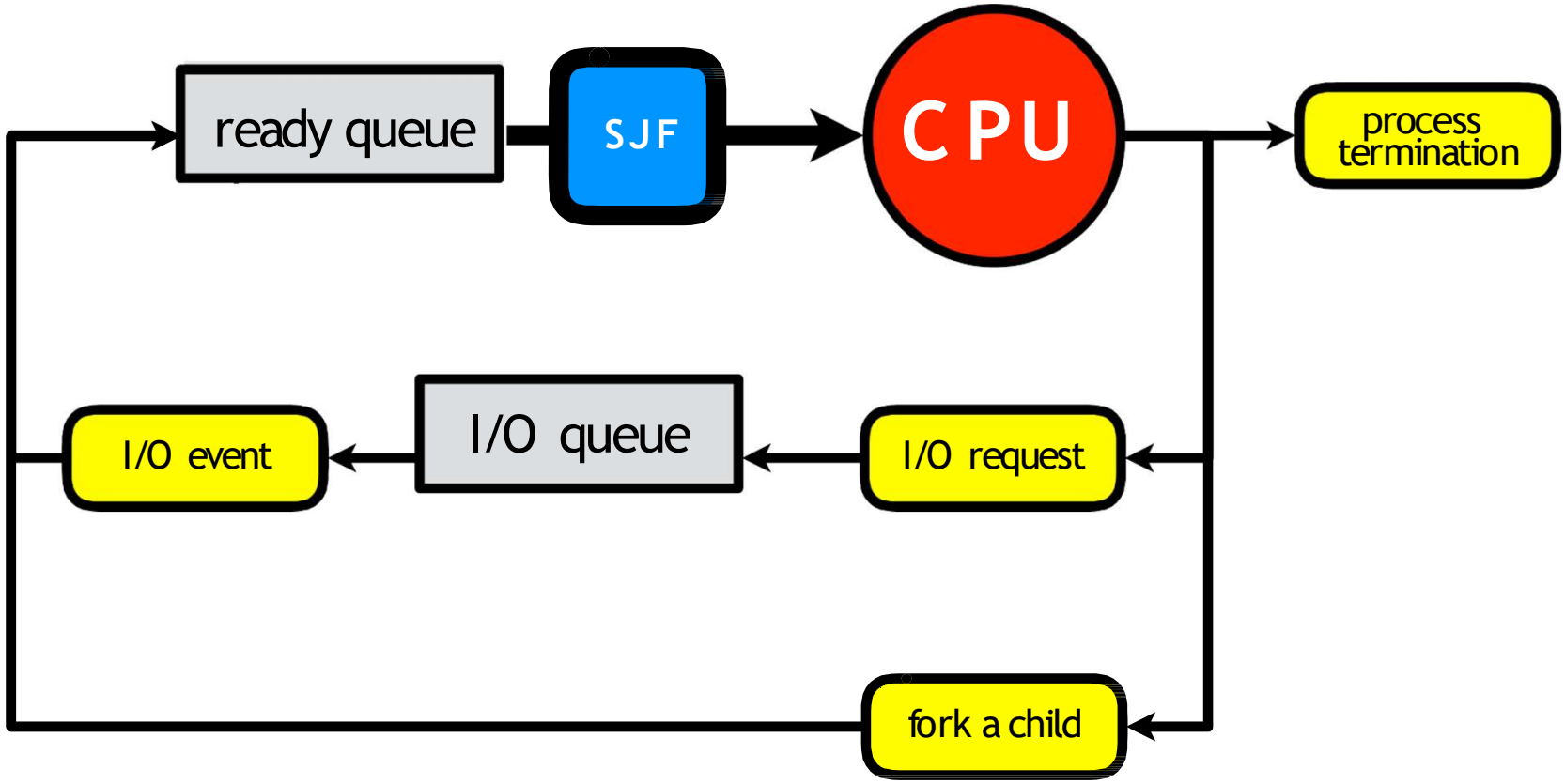
The average waiting time is reduced if the x_i's that are multiplied the most times are the smallest ones.

SJF

Shortest Job First

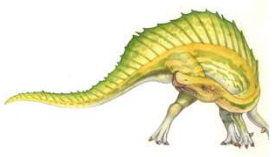
Shortest Job First (SJF)

Shortest Job First (SJF) scheduling assigns the process estimated to complete fastest, i.e, the process with shortest CPU burst, to the CPU as soon as CPU time is available.



Shortest-Job-First (SJF) Scheduling

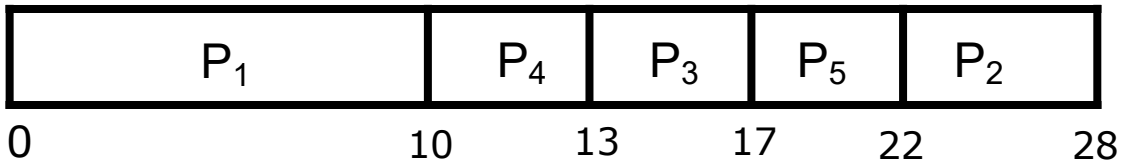
- Process with shortest **expected** processing time is selected next
- Short process jumps ahead of longer processes
- Non-preemptive policy
- Also known as Shortest Process Next (SPN)
- SJF is optimal
  - Gives minimum average waiting time for a given set of processes



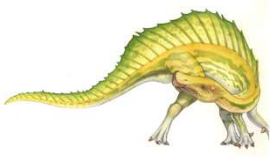
Shortest-Job-First (SJF) Scheduling

Process	Arrival Time	CPU Burst Time
P_1	0	10
P_2	3	6
P_3	3	4
P_4	8	3
P_5	13	5

Gantt Chart:



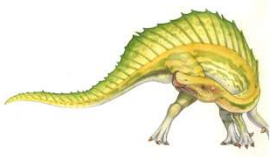
$$\text{Avg Waiting Time} = \frac{0 + 19 + 10 + 2 + 4}{5} = 7$$



Shortest-Job-First (SJF) Scheduling

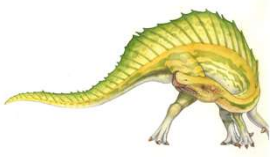
- Associate with each process the length of its **next CPU burst**
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request

Process	Arrival Time	CPU Burst Time
P_1	0	10
P_2	3	6
P_3	3	4
P_4	8	3
P_5	13	5



Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
- Preemptive version called **shortest-remaining-time-first**
- How do we determine the length of the next CPU burst?
  - **Could ask the user**
  - **Estimate**

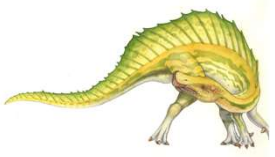


Determining Length of Next CPU Burst

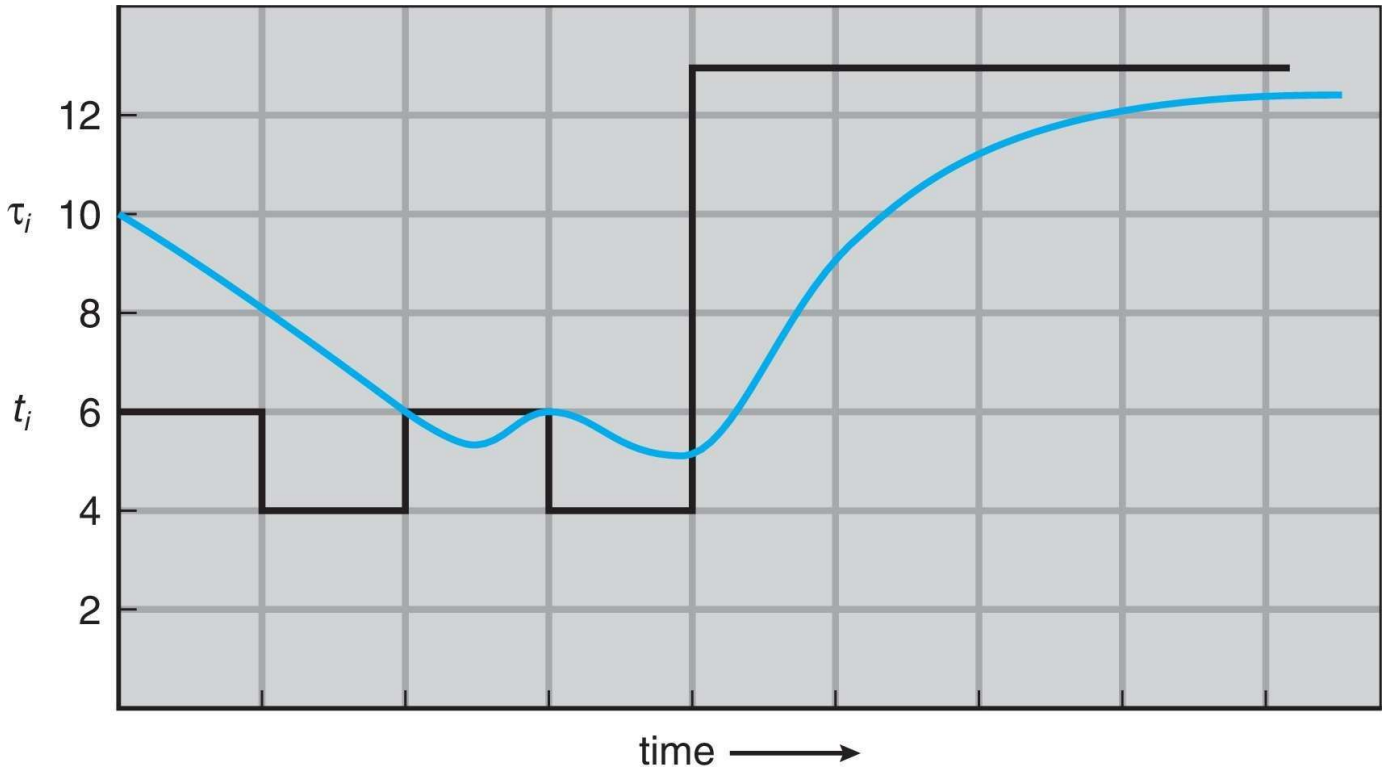
- We need to **estimate** the required processing time (next CPU burst) of each process.
- Estimation can be done by using the length of previous CPU bursts, using **exponential averaging**

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

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$ (Commonly set to ½)



Prediction of the Length of the Next CPU Burst



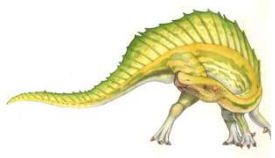
CPU burst (t_i)		6	4	6	4	13	13	13 ...
"guess" (τ_i)	10	8	6	6	5	9	11	12 ...

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

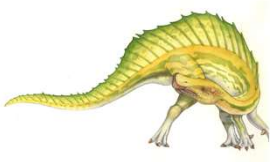
$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor



Shortest Remaining Time First (SRTF)

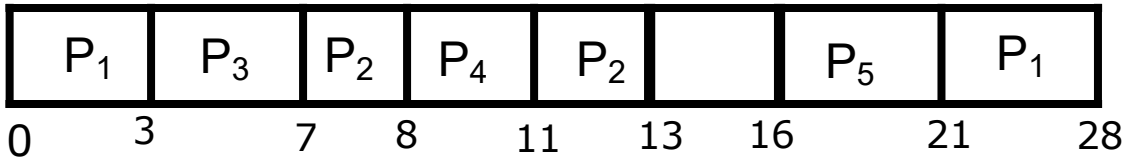
- SRTF is the **preemptive** version of SJF
  - If the newly arrived process has a shorter running time than what is left of the currently executing process, then the OS will preempt the current process.



Shortest Remaining Time First (SRTF)

Process	Arrival Time	CPU Burst Time
P_1	0	10
P_2	3	6
P_3	3	4
P_4	8	3
P_5	13	5

Gantt Chart:



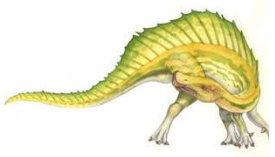
$$\text{Avg Waiting Time} = \frac{18 + 7 + 0 + 0 + 3}{5} = 5.6$$

Highest Response Ratio Next (HRRN)

- Important SJF and SRTF disadvantage
  - Possibility of **starvation** for longer processes
- HRRN choose next process with the greatest ratio

$$\uparrow Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time} \downarrow$$

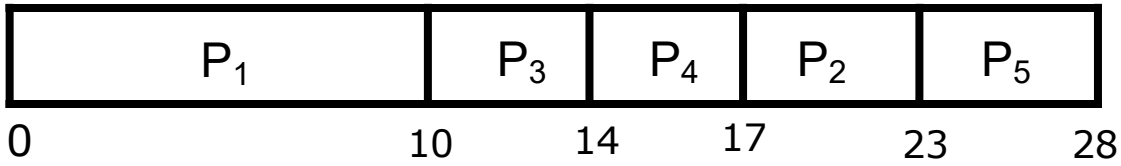
- A smaller denominator yields a larger ratio so that shorter jobs are favoured,
  - But aging without service increases the ratio so that a longer process will eventually get past competing shorter jobs.
- **Non-preemptive** policy



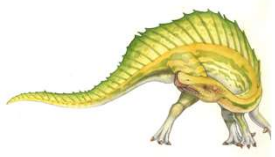
Highest Response Ratio Next (HRRN)

Process	Arrival Time	CPU Burst Time			
P_1	0	10			
P_2	3	6	13/6	17/6	20/6
P_3	3	4	11/4		
P_4	8	3	5/3	9/3	
P_5	13	5		6/5	9/5

Gantt Chart:

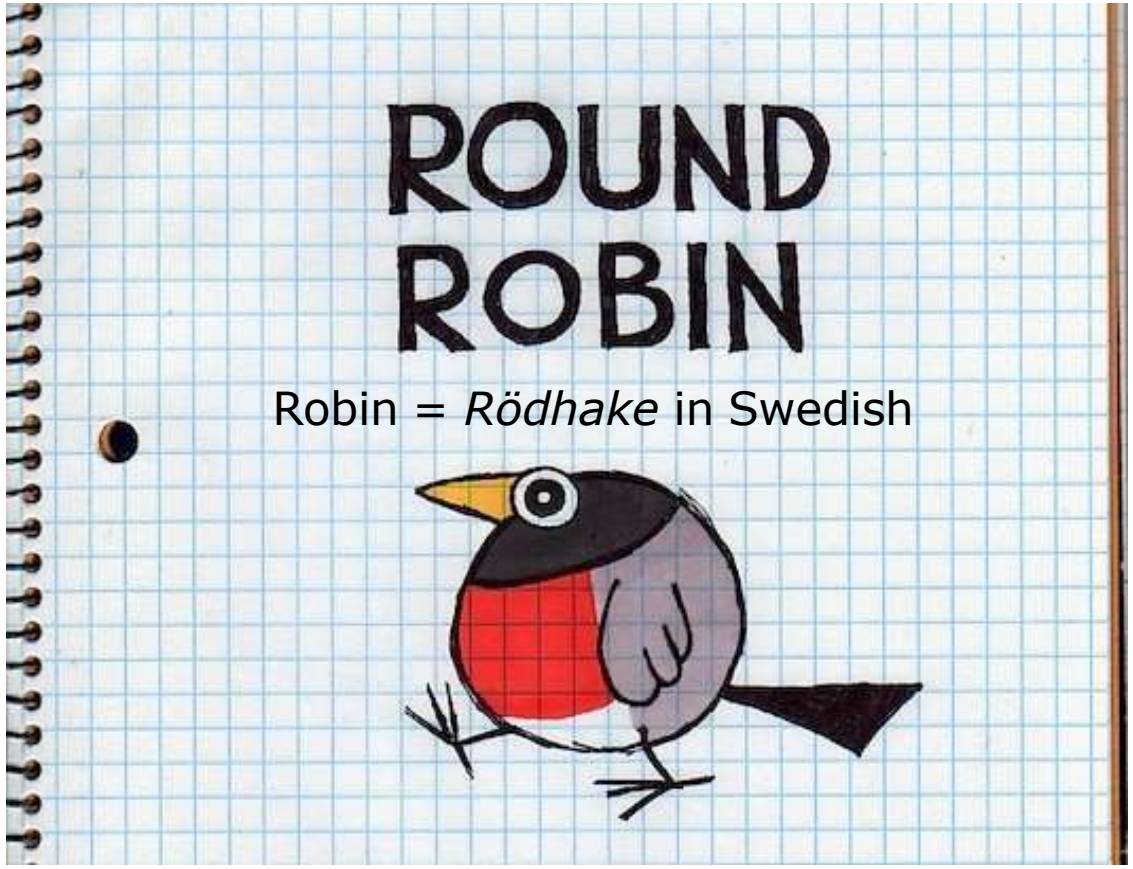


$$\text{Avg Waiting Time} = \frac{0+14+7+6+10}{5} = 7.4$$

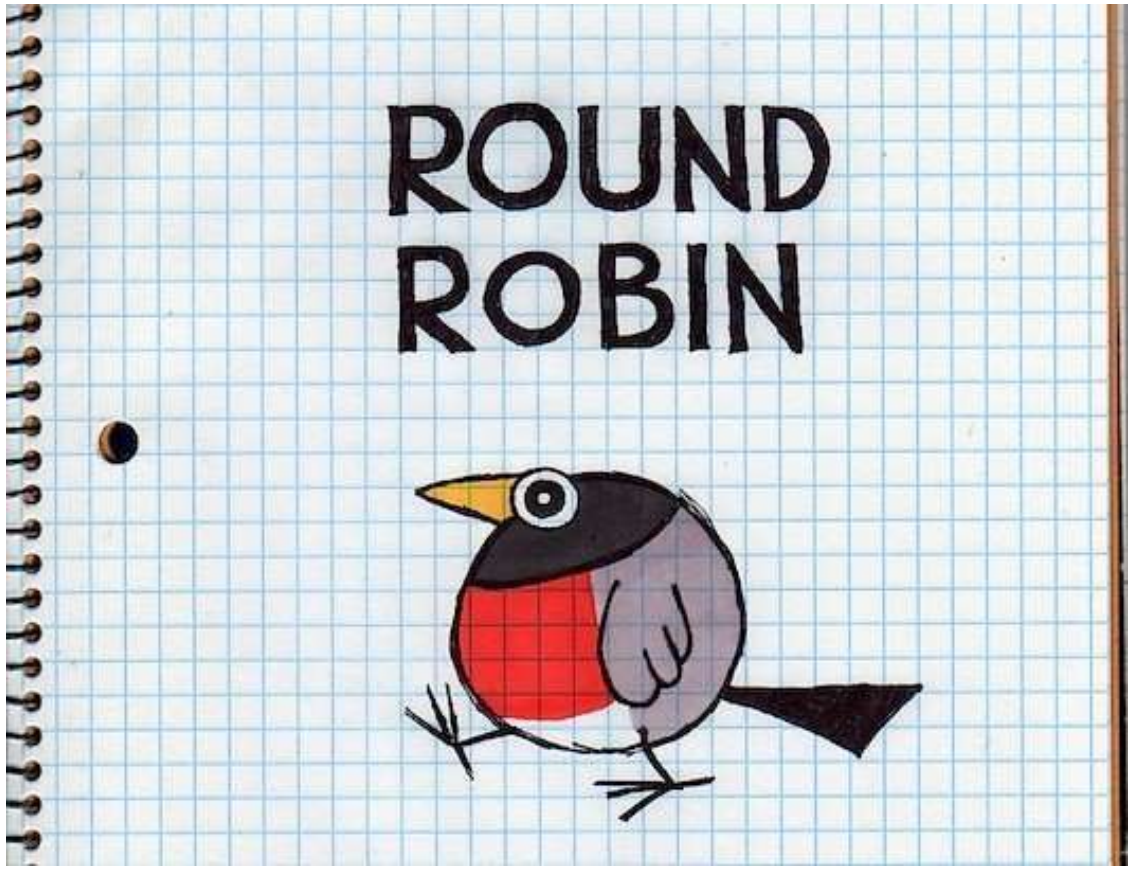


RR

Round Robin



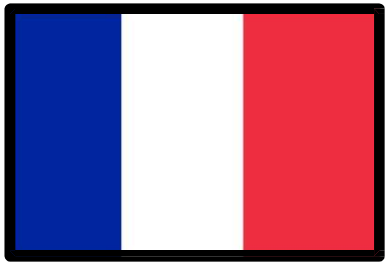
In general, round-robin refers to a pattern or ordering whereby items are encountered or processed sequentially, often beginning again at the start in a circular manner.

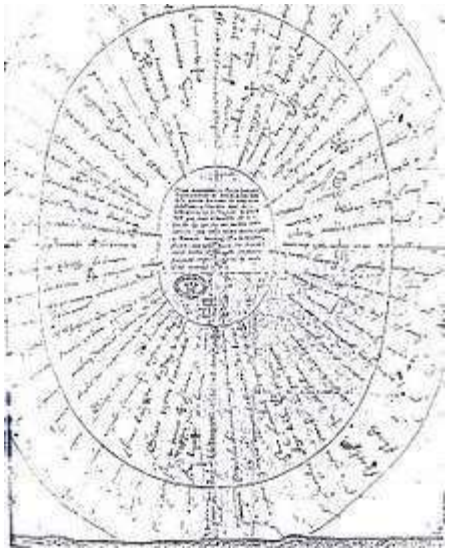


Round Robin is one of the simplest **CPU scheduling** algorithms that also **prevents starvation**.

Etymology

The phrase round-robin actually has nothing whatever to do with a bird, robin or any other kind.



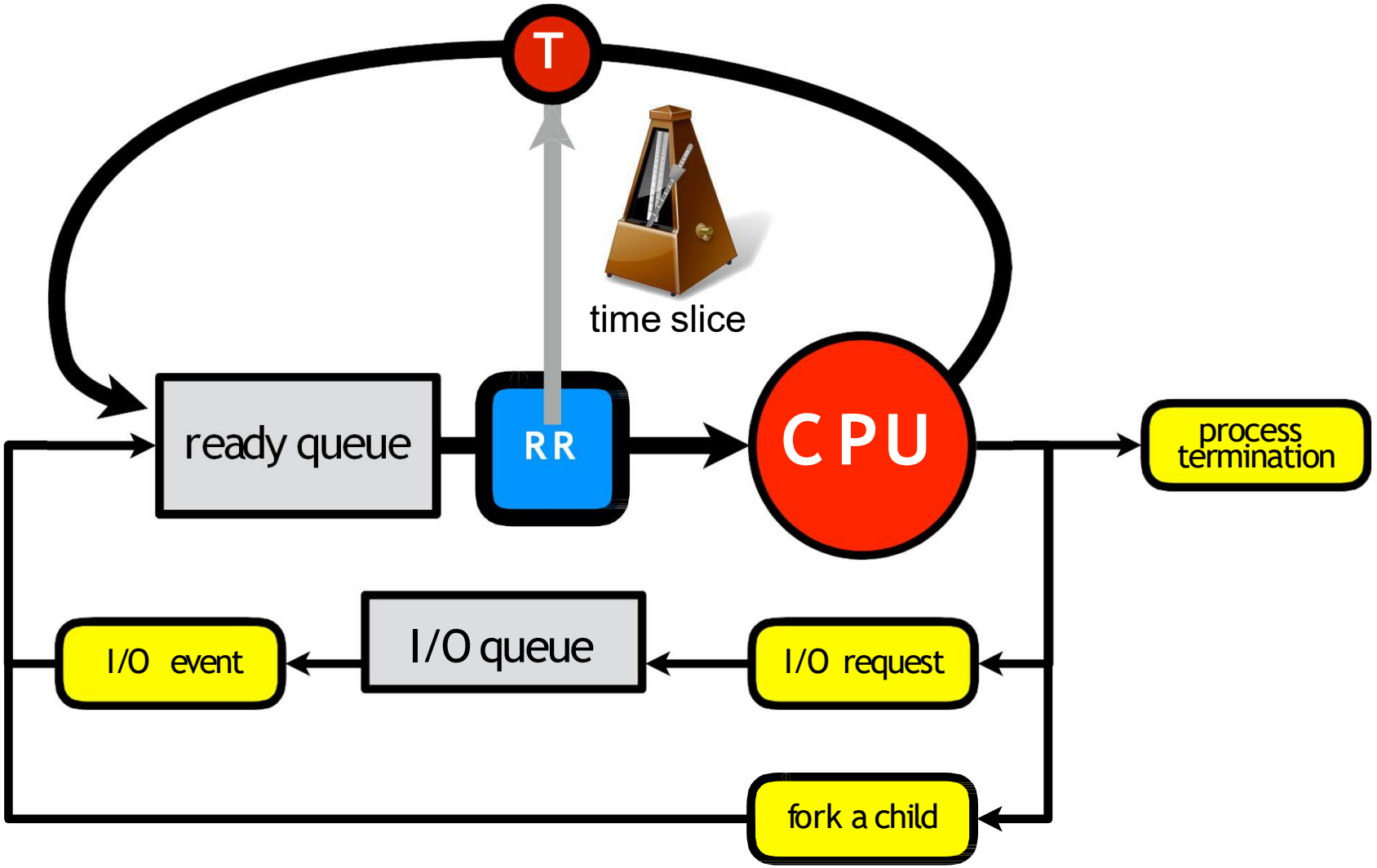


- The term round-robin dates from the 17th-century French **ruban rond** (round ribbon).
- Originally, round-robin is a **document signed by multiple parties in a circle**.
- Round-robin described the practice of signatories to petitions against authority (usually Government officials petitioning the Crown) appending their names on a document in a non-hierarchical circle or ribbon pattern (and so disguising the order in which they have signed) so that none may be identified as a ringleader.

Source: https://en.wikipedia.org/wiki/Round-robin_(document) 2016-02-02

Round Robin (RR)

Round Robin (RR) is a scheduling algorithm where time slices are assigned to each process in equal portions and in circular order.

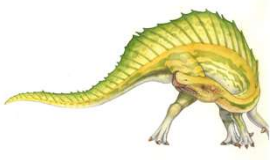


Round Robin (RR)

- RR is design specially for time sharing systems.
- It is similar to FCFS scheduling, but preemption is added to switch between processes.

How it works:

- Ready queue is organized as a FIFO queue of processes
- Each process gets a small unit of CPU time (**time quantum** *q*), usually 10-100 milliseconds.
- After this time has elapsed, the process is preempted and added to the end of the ready queue.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high

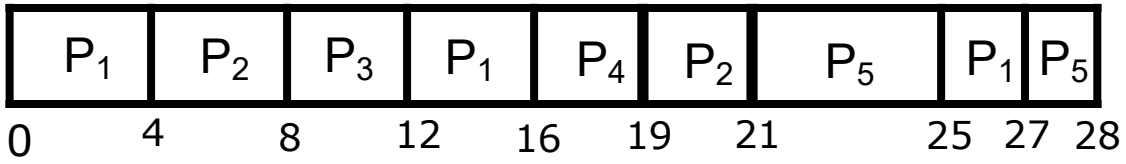


Round Robin (RR)

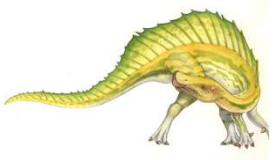
Process	Arrival Time	CPU Burst Time
P_1	0	10
P_2	3	6
P_3	3	4
P_4	8	3
P_5	13	5

$q = 4$

Gantt Chart:

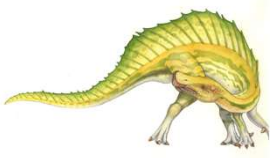


$$\text{Avg Waiting Time} = \frac{17 + 12 + 5 + 8 + 10}{5} = 10.14$$

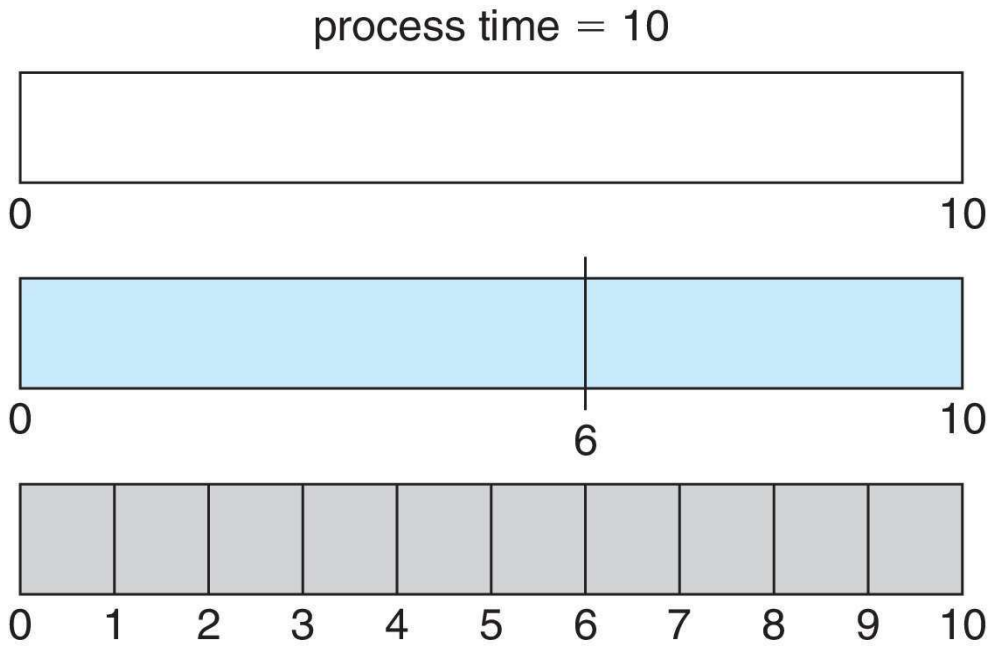


Round Robin (RR)

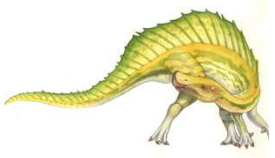
- The average waiting time under the RR policy is often long.
- Performance
  - *q* large ⇒ FIFO
  - *q* small ⇒ **processor sharing equally between all the processes (completely fair!)**
    - *q* must be large with respect to context switch, otherwise overhead is too high
    - (**in theory**) creates the appearance that each of n processes has its own processor running at 1/n the speed of the real processor.
    - But q must be large compare to context switch, otherwise overhead is too high (q usually 10ms to 100ms, context switch < 10 μsec)



Time Quantum and Context Switch Time



quantum	context switches
12	0
6	1
1	9



Virtual Round Robin (VRR)

- Similar to Round Robin
- There is an FCFS **auxiliary queue** to which processes are moved after being released from an I/O block.
- Processes in the auxiliary queue get **preference** over those in the main ready queue.
- When a process is dispatched from the auxiliary queue, it runs no longer than a time equal to the basic time quantum **minus** the total time spent running (of the last execution, CPU burst) since it was last selected from the main ready queue.

Virtual Round Robin (VRR)

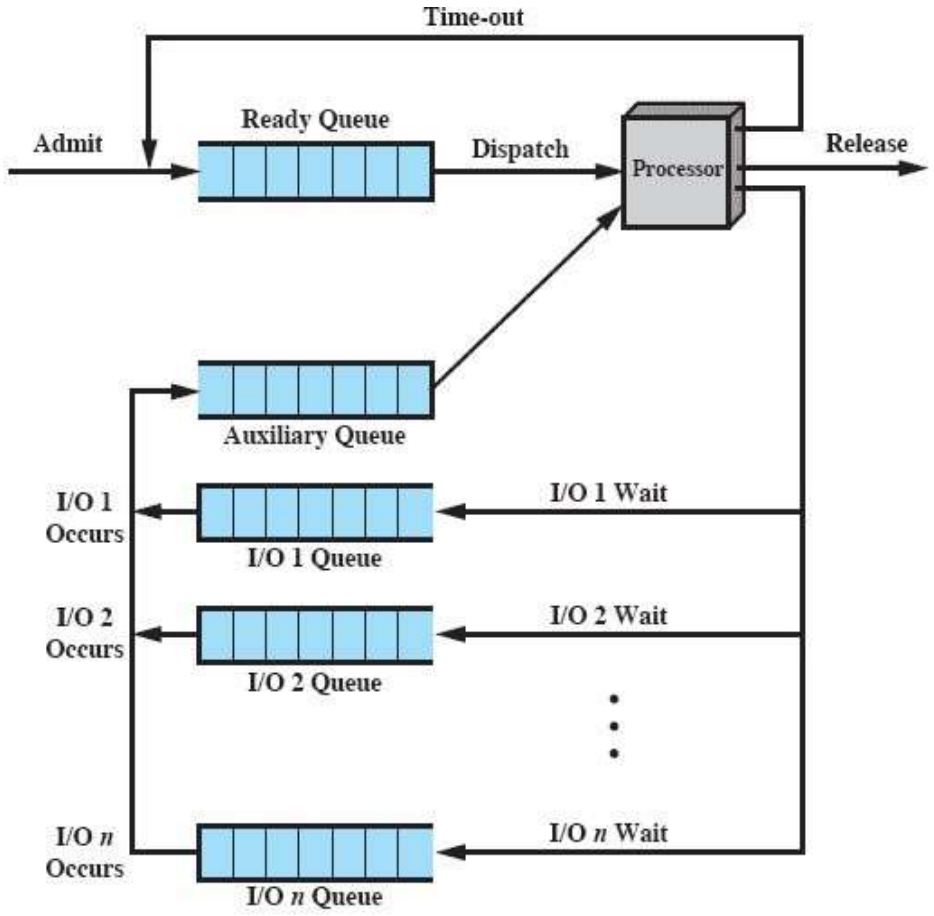


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

Multilevel Feedback Queue

- Penalize jobs that have been running longer
- Example
  - Three queues:
    - RQ_0 – RR with time quantum 8 milliseconds
    - RQ_1 – RR time quantum 16 milliseconds
    - RQ_2 – FCFS

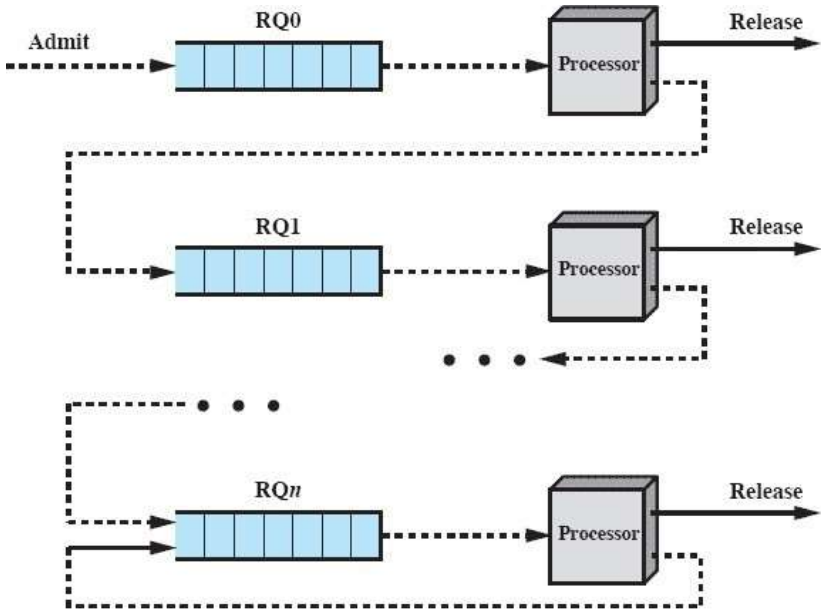
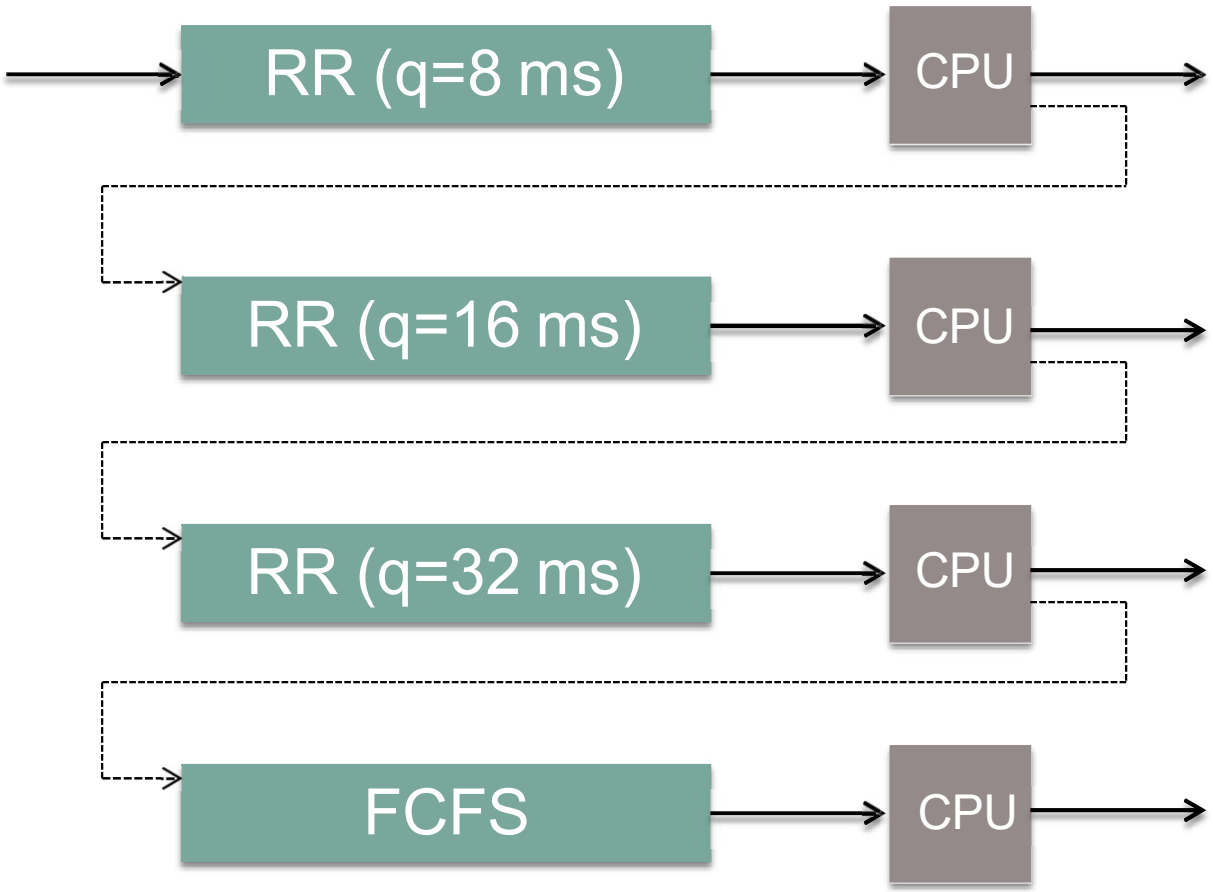


Figure 9.10 Feedback Scheduling

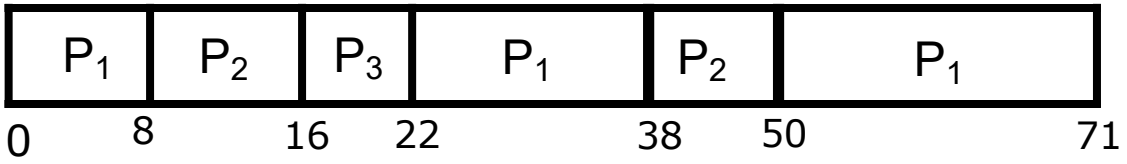
Multilevel Feedback Queue



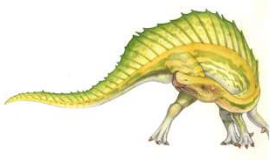
Multilevel Feedback Queue

Process	CPU Burst Time
P_1	45
P_2	20
P_3	6

Gantt Chart:

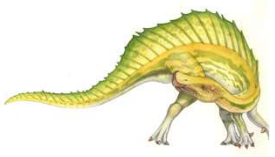


$$\text{Avg Waiting Time} = \frac{26 + 30 + 16}{3} = 24$$



Priority Scheduling

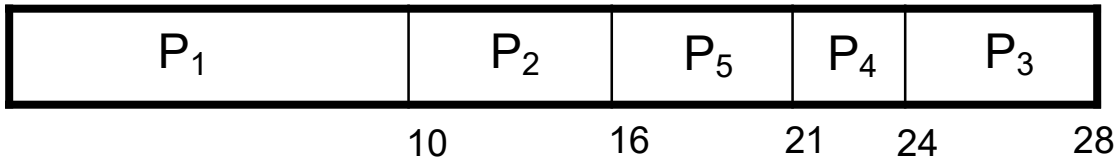
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation**
  - low priority processes may never execute
  - Solution ≡ **Aging** – as time progresses increase the priority of the process



Priority Scheduling (non-preemptive)

Process	Arrival Time	CPU Burst Time	Priority
P_1	0	10	4
P_2	3	6	2
P_3	3	4	5
P_4	8	3	3
P_5	13	5	1

Gantt Chart:

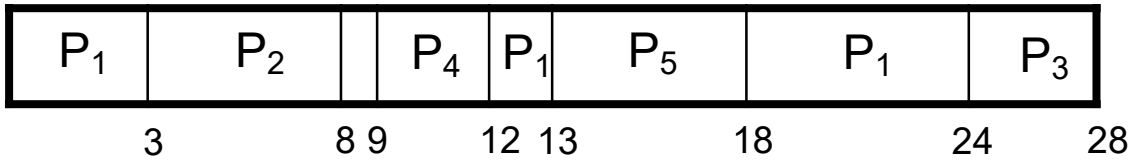


$$\text{Avg Waiting Time} = \frac{0 + 7 + 21 + 13 + 3}{5} = 8.8$$

Priority Scheduling (preemptive)

Process	Arrival Time	CPU Burst Time	Priority
P_1	0	10	4
P_2	3	6	2
P_3	3	4	5
P_4	8	3	3
P_5	13	5	1

Gantt Chart



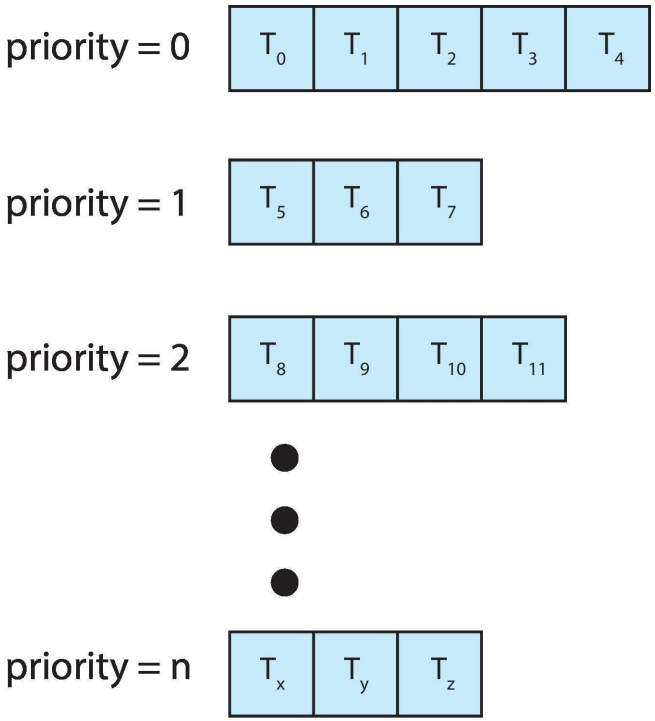
$$\text{Avg Waiting Time} = \frac{14 + 0 + 21 + 1 + 0}{5} = 7.2$$

Multilevel Queue

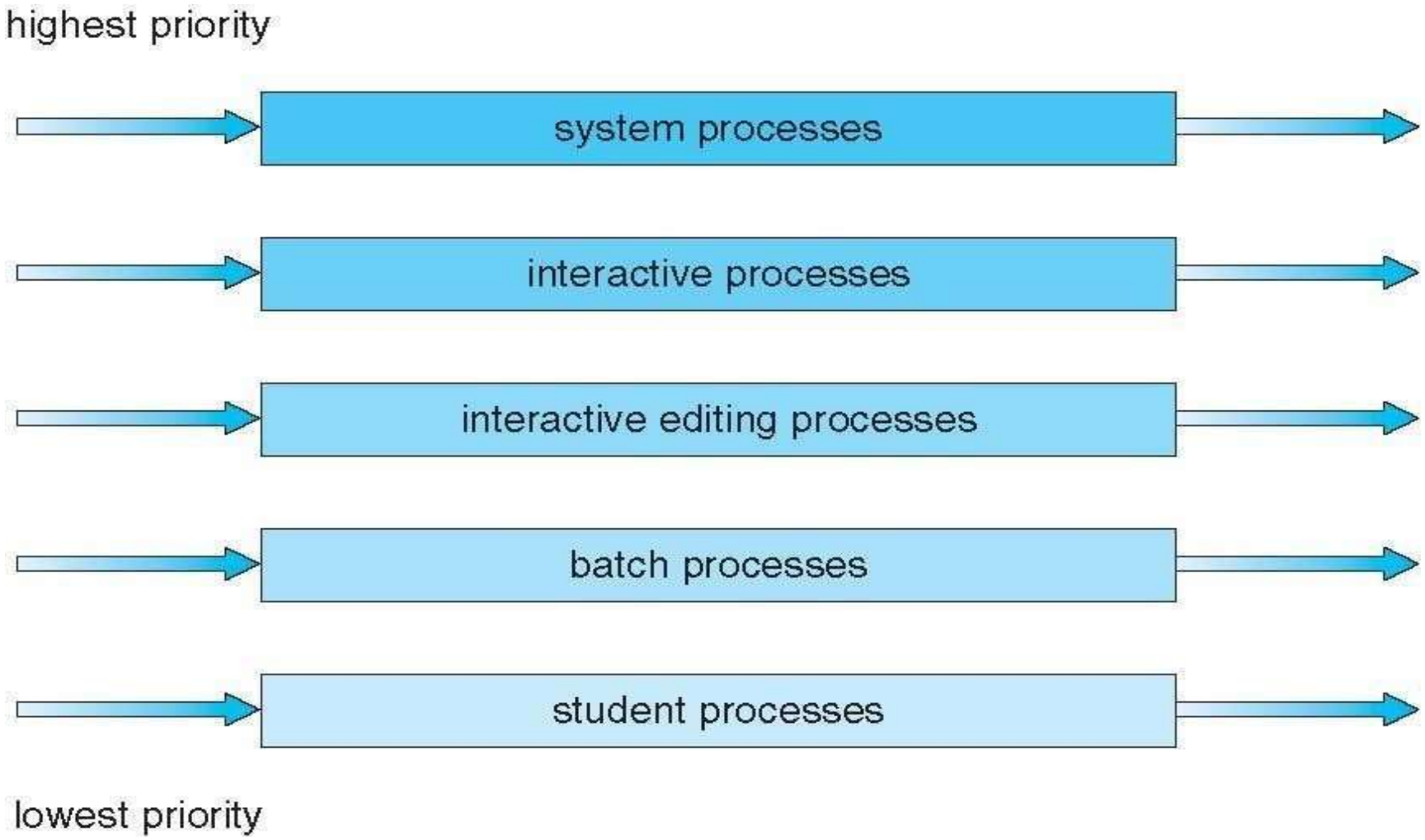
- Ready queue is partitioned into separate queues
  - foreground (interactive)
  - background (batch)
- The processes are permanently assigned to one queue, generally based on some property
- Each **queue** has its own **scheduling** algorithm:
  - Foreground: RR
  - Background: FCFS
- Scheduling must be done **between the queues**:
  - Fixed priority preemptive scheduling
  - Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR 20% to background in FCFS

Multilevel Queue

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!

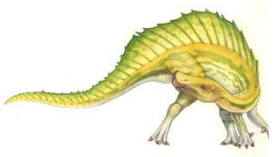


Multilevel Queue



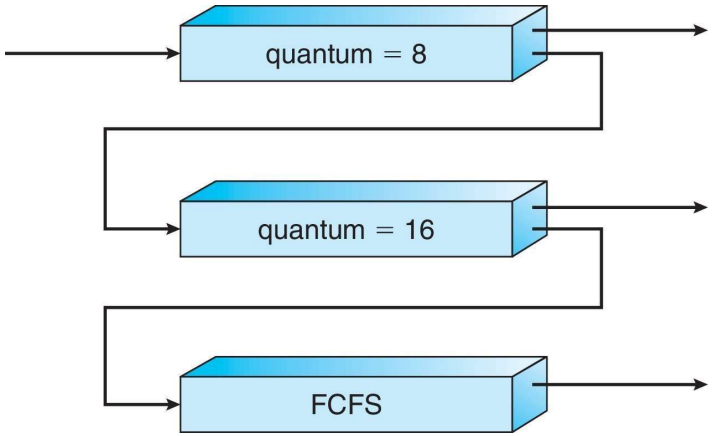
Multilevel Feedback Queue

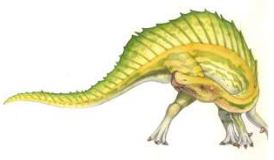
- A process can move between the various queues.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to upgrade a process
  - Method used to determine when to demote a process
  - Method used to determine which queue a process will enter when that process needs service
- Aging can be implemented using multilevel feedback queue



Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new process enters queue Q_0 which is served in RR
    - When it gains CPU, the process receives 8 milliseconds
    - If it does not finish in 8 milliseconds, the process is moved to queue Q_1
  - At Q_1 job is again served in RR and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2





Algorithm Evaluation

- How do we select a CPU scheduling algorithm for a particular system?
  - Deterministic Modeling
    - takes a particular predetermined workload and defines the performance of each algorithm for that workload (prev. examples)
  - Queueing Models
  - Simulations
  - Implementation

Queueing Models

- Using I/O and CPU burst distribution (exponential, Poisson, …) and also arrival-time distribution, we can compute the average throughput, utilization, waiting time, and so on for most algorithms.
- Example: Little's Formula
  - n = average queue length
  - W = average waiting time in queue
  - λ = average arrival rate into queue
  - Little's law – in steady state, processes leaving queue must equal processes arriving, thus $n = \lambda \times W$

Simulations

- Queueing models are limited and are often only approximations of real systems
- **Simulations** are more accurate
  - Programmed model of computer system
  - Clock is a variable
  - Gather statistics indicating algorithm performance
  - Data to drive simulation gathered via
    - Random number generator according to probabilities
    - Distributions defined mathematically or empirically
    - Trace tapes record sequences of real events in real systems

Implementation

- Even simulations have limited accuracy
- Just implement new scheduler and test in real systems
  - High cost, high risk
  - Environments vary

End of Chapter 6