SCHEDULLING

Scheduling

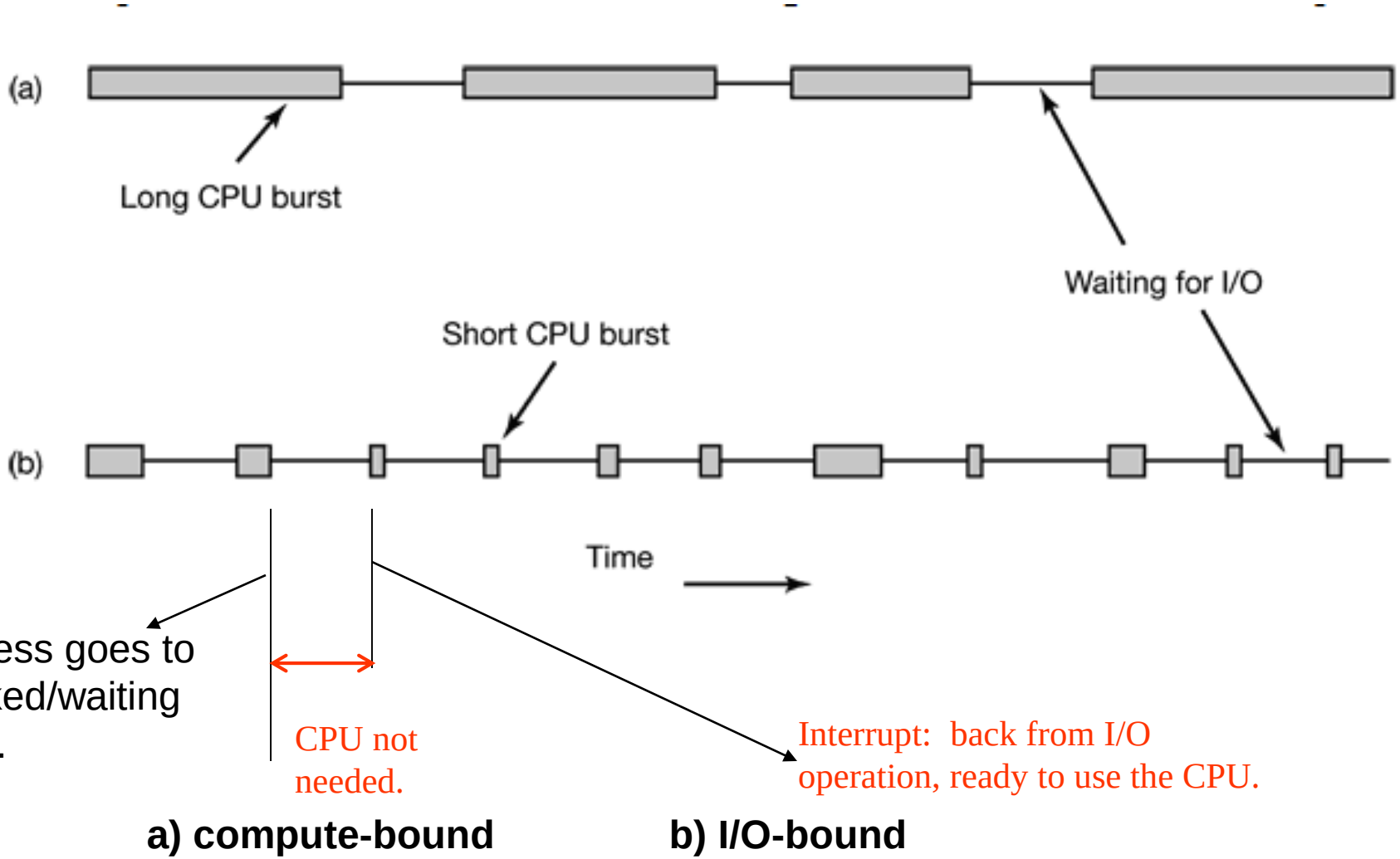
- Multiprogrammed computer
  - Multiple processes running concurrently
  - Processes compete for the CPU
  - If there is a single CPU, a choice has to be made which process to run next
- The part of the operating system that makes the choice is called the **scheduler.**
- The algorithm used by the scheduler to decide which process next is called the **scheduling algorithm**

Scheduling

Process behavior:

- compute-bound (CPU-bound)
  - spend most of their time computing
  - have long CPU bursts
  - infrequent I/O waits
- I/O-bound
  - have short CPU bursts
  - frequent I/O waits
- The key factor in identifying CPU-bound and I/O-bound processes is the length of the CPU burst, not the length of the I/O burst
- As CPUs get faster, processes tend to get more I/O-bound

Scheduling


(a)
Long CPU burst
Waiting for I/O
Short CPU burst
(b)
Time
Process goes to blocked/waiting state.
CPU not needed.
Interrupt: back from I/O operation, ready to use the CPU.
a) compute-bound
b) I/O-bound

Scheduling

When to Schedule?

- A new is created
  - Since the parent and child processes are in ready state, decision needs to be made whether to run the parent process or the child process.
- A process exits
  - That process can no longer run (since it no longer exists), so some other process must be chosen from the set of ready processes
  - If no process is ready, a system-supplied idle process is normally run
- A process blocks
  - when a process blocks on I/O, on a semaphore, or for some other reason, another process has to be selected to run
  - The reason for blocking may play a role in the selection of the next process, but the scheduler doesn't have enough info
- I/O interrupt
  - Scheduler decides to run the newly ready process, continue the interrupted process or run another process in the ready queue

Scheduling

Types of scheduling:

- Non-preemptive
  - scheduling algorithm picks a process to run and then just lets it run until it blocks (either on I/O or waiting for another process) or until it voluntarily releases the CPU
  - Even if it runs for hours, it will not be forcibly suspended
- Preemptive
  - Picks a process and lets it run for a maximum of some fixed time
  - Requires availability of clock

Scheduling

Types of Scheduling

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

Scheduling

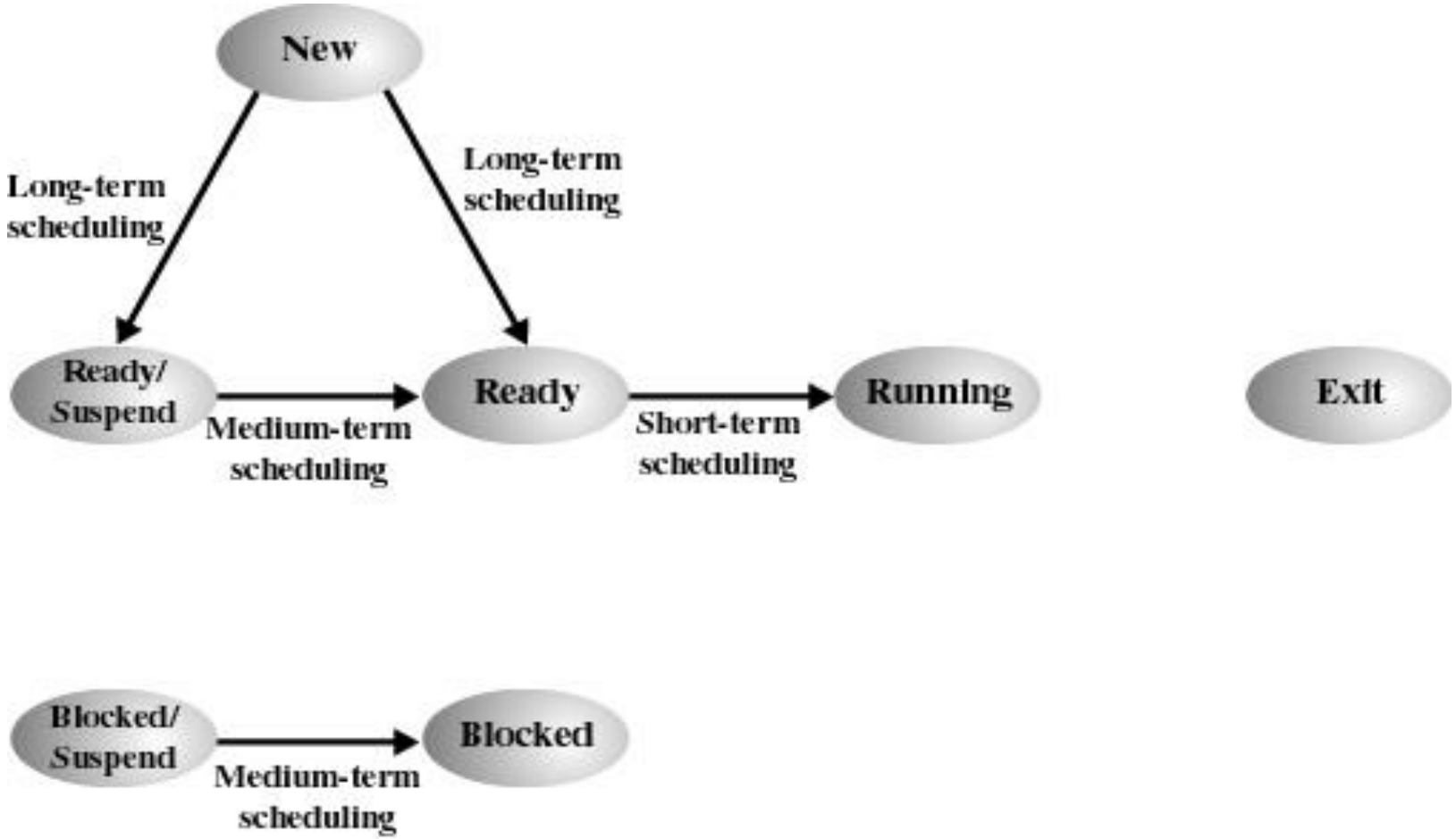
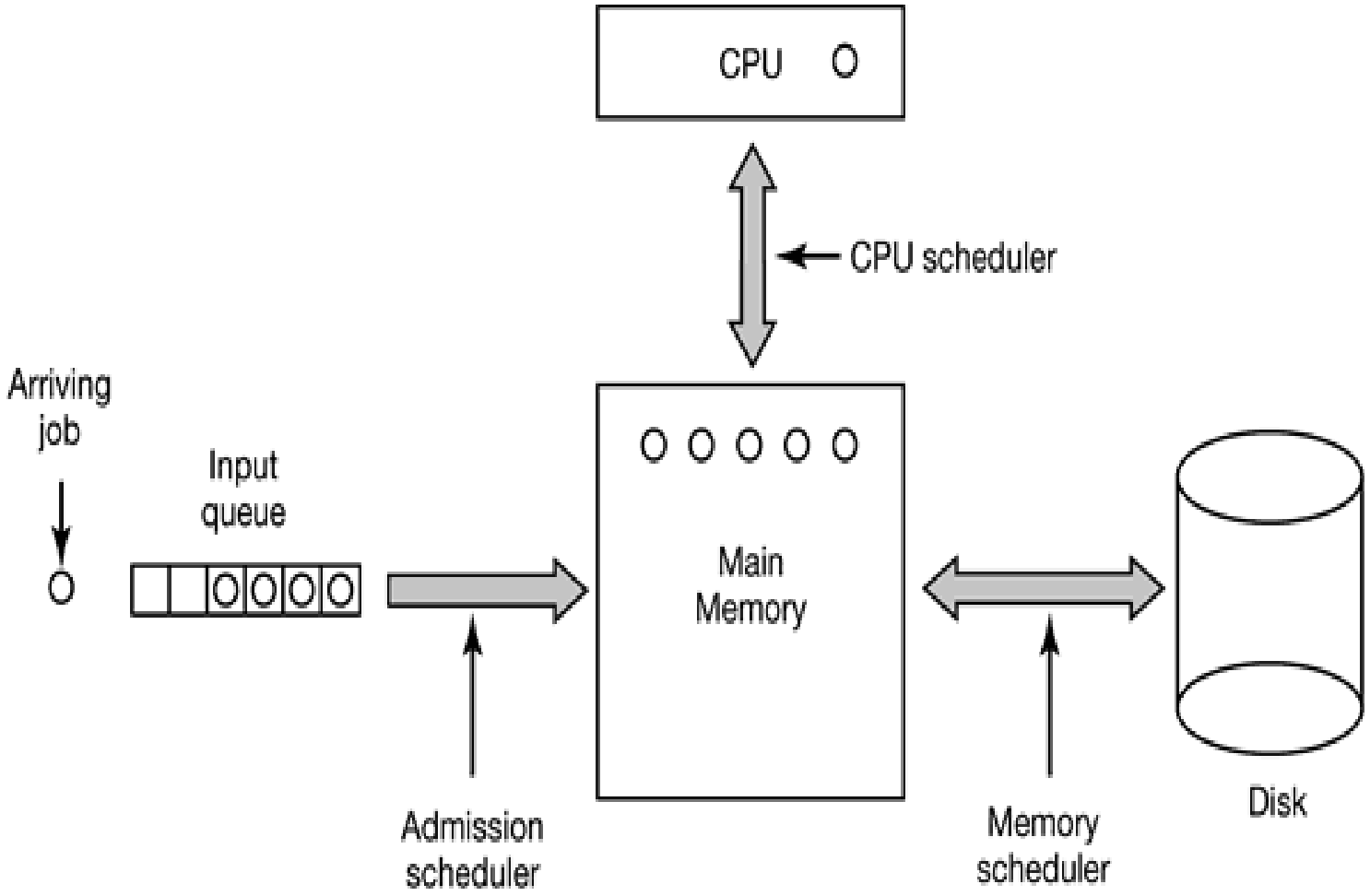


Figure 9.1 Scheduling and Process State Transitions

Scheduling


CPU
CPU scheduler
Arriving job
Input queue
Main Memory
Admission scheduler
Memory scheduler
Disk

Scheduling

Scheduling policies in different environments

- Batch systems:
  - there are no users impatiently waiting at their terminals for a quick response
  - nonpreemptive algorithms, or preemptive algorithms with long time periods for each process are often acceptable
  - This approach reduces process switches and thus improves performance
    - First Come First Served
    - Shorted Job First
    - Shortest Remaining Time Next

Scheduling

Scheduling policies in different environments

- Interactive systems:
  - preemption is essential to keep one process from hogging the CPU and denying service to the others
    - Round Robin
    - Priority Scheduling
    - Multiple Queues
    - Shortest Process Next
    - Guaranteed Scheduling
    - Lottery Scheduling
    - Fair-share Scheduling
- Real-time systems:
  - Static vs. dynamic

Criteria & Objectives

- CPU utilization – keep the CPU as busy as possible (40 – lightly loaded, 90 – heavily loaded)
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process (total time spent on the system)
- 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)

Scheduling Algorithm Goals

- All systems
  - Fairness - giving each process a fair share of the CPU
  - Policy enforcement - seeing that stated policy is carried out
  - Balance - keeping all parts of the system busy
- Batch systems
  - Throughput - maximize jobs per hour
  - Turnaround time - minimize time between submission and termination
  - CPU utilization - keep the CPU busy all the time
- Interactive systems
  - Response time - respond to requests quickly
  - Proportionality - meet users' expectations
- Real-time systems
  - Meeting deadlines - avoid losing data
  - Predictability - avoid quality degradation in multimedia systems

First-Come First-Served

- Simplest of all scheduling algorithms is nonpreemptive FCFS or FIFO
- As each process becomes ready, it joins the ready queue
- When the running process blocks or exits, the first process on the queue (i.e with the longest waiting time in the queue) is run next
- When a blocked process becomes ready, like a newly arrived job, it is put on the end of the queue
- Advantages:
  - Easy to understand and program
  - It is fair
- Limitations
  - Favors CPU-bound processes over I/O bound processes
  - May result in inefficient use of both the processor & I/O devices

First-Come First-Served

- FCFS performs (less normalized turnaround time) much better for long processes than short ones. Consider the following example:

Process	Arrival Time	Service Time (T_s)	Start Time	Finish Time	Turnaround Time (T_r)	T_r/T_s
W	0	1	0	1	1	1
X	1	100	1	101	100	1
Y	2	1	101	102	100	100
Z	3	100	102	202	199	1.99
Mean					100	26

Throughput = No jobs/total time

= 4/202

=0.02

Turn around time:

Tr_W = 1-0 = 1

Tr_X = 101-1=100

Tr_Y = 102-2=100

Tr_Z = *202-3=199*

Average Turn Around Time:

AvgTr = (1+100+100+199) / 4 = 100

First-Come First-Served

Process	Arrival Time	Service Time (T_s)	Start Time	Finish Time	Turnaround Time (T_r)	T_r/T_s
W	0	1	0	1	1	1
X	1	100	1	101	100	1
Y	2	1	101	102	100	100
Z	3	100	102	202	199	1.99
Mean					100	26

Waiting time:

Wt_W = 0-0=0

Wt_X = 1-1=0

Wt_Y = 101-2=99

Wt_Z = 102-3=99

Average Waiting time:

Avg Wt =

{Wt(W) + Wt(X) + Wt(Y) + Wt(Z)} / 4

=(0+0+99+99)/4

=49.5

Response Time:

Rt_W = 0-0=0

Rt_X = 1-1=0

Rt_Y = 101-2=99

Rt_Z = 102-3=99

AvgRt = (99+99)/4 = 49.5

Note: Response time is different from waiting time if the process is interrupted or blocks for I/O operations

Shortest Job First

- This another nonpreemptive batch algorithm that assumes the run times are known in advance.
- The scheduler picks the process with the shortest run time
- Shortest job first is only optimal when all the jobs are available simultaneously
- Shortest job first is provably optimal.
- Consider the case of four jobs, with run times of *a*, *b*, *c*, and *d*, respectively.
  - The first job finishes at time *a*, the second finishes at time *a* + *b*, and so on.
  - The mean turnaround time is $(4a + 3b + 2c + d)/4$
  - It is clear that *a* contributes more to the average than the other times, so it should be the shortest job, with *b* next, then *c*, and finally *d* as the longest as it affects only its own turnaround time

Shortest Job First

Process	Arrival Time	Service Time	Start Time	Finish Time	Turnaround Time	Tr/Ts
W	0	10	5	15	15	1.5
X	0	100	65	165	165	1.65
Y	0	5	0	5	5	1
Z	0	50	15	65	65	1.35
Mean					62.5	1.38

- **Throughput** = 4/165 = <u>0.024</u>
- **Avg. Turnaround time** = 15+165+5+65= <u>62.5</u>

Waiting time:

Wt_W = 5-0 = 5
Wt_X = 65-0 = 65
Wt_Y = 0-0 = 0
Wt_Z = 15-0 = 15
Avg Wt = (5+65+0+15)/4
= <u>21.25</u>

Response Time:

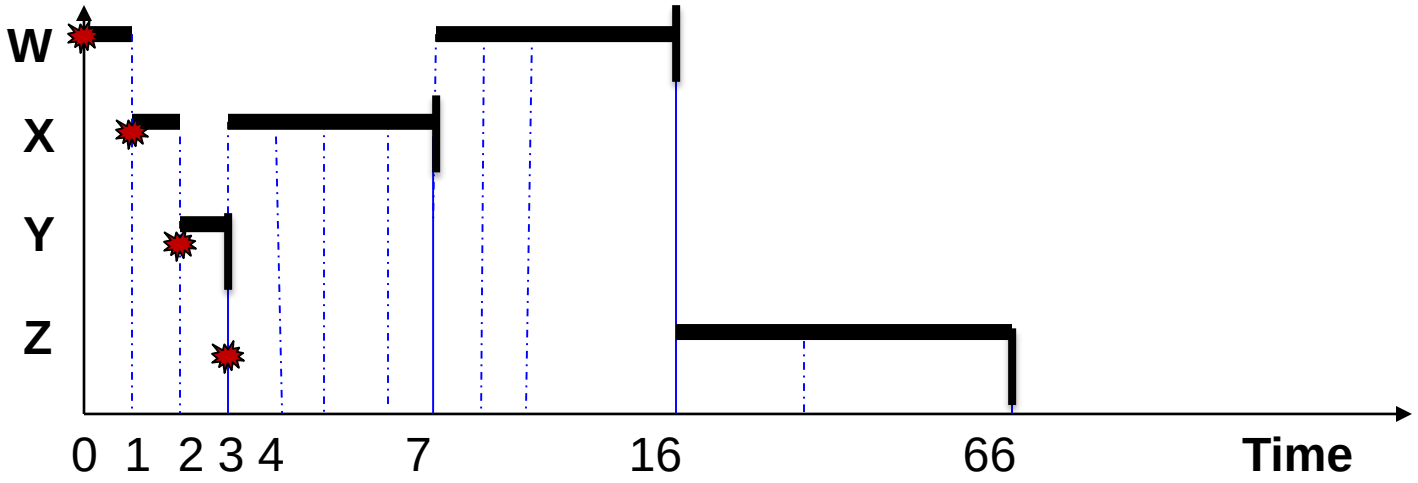
Rt_W = 5-0 = 5
Rt_X = 65-0 = 65
Rt_Y = 0-0 = 0
Rt_Z = 15-0 = 15
Avg Rt = (5+65+0+15)/4
= <u>21.25</u>

Shortest Remaining Time Next

- It is a preemptive version of shortest job first
- The scheduler always chooses the process whose remaining run time is the shortest
- When a new job arrives, its total time is compared to the current process' remaining time
  - If the new job needs less time to finish than the current process, the current process is suspended and the new job started
- This scheme allows new short jobs to get good service

Shortest Remaining Time Next

Process	Arrival Time	Service Time	Start Time	Finish Time	Turnaround Time	Tr/Ts
W	0	10	0	16	16	1.6
X	1	5	1	7	6	1.2
Y	2	1	2	3	1	1
Z	3	50	16	66	63	1.26



Shortest Remaining Time Next

Process	Arrival Time	Service Time	Start Time	Finish Time	Turnaround Time	Tr/Ts
W	0	10	0	16	16	1.6
X	1	5	1	7	6	1.2
Y	2	1	2	3	1	1
Z	3	50	16	66	63	1.26

Turnaround Time:

$Tr_W = 16-0 = 16$

$Tr_X = 7-1 = 6$

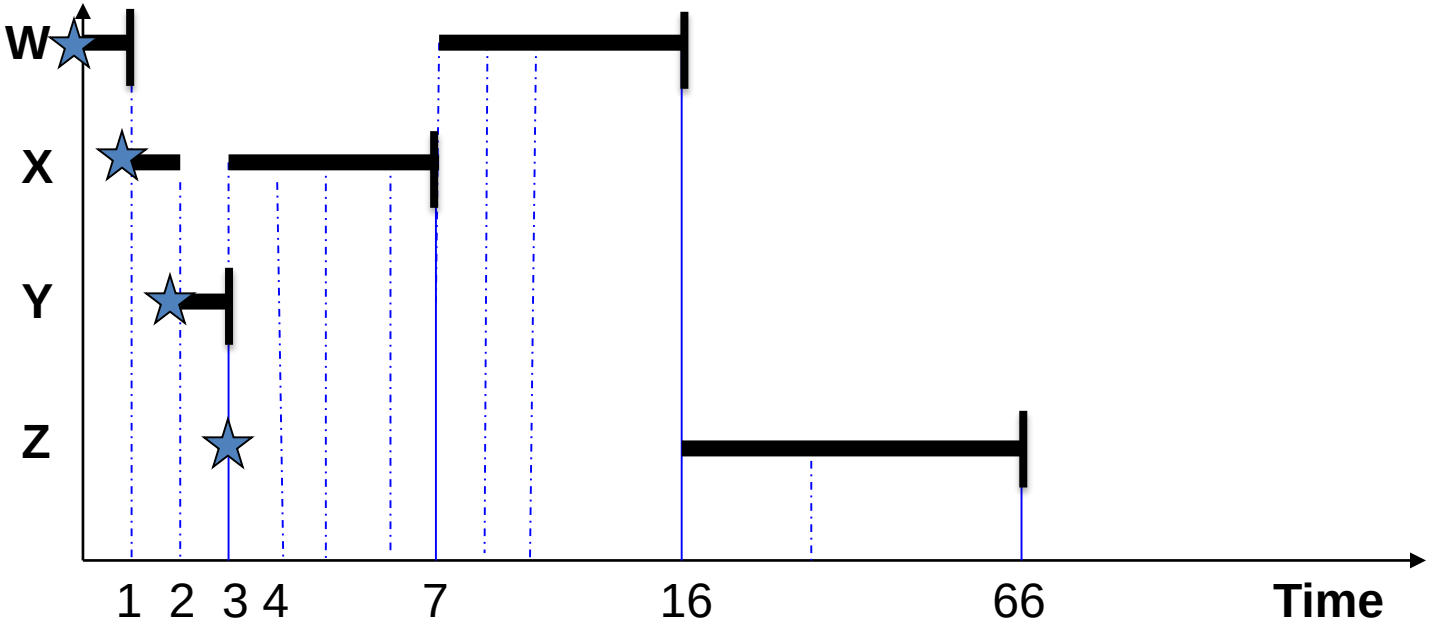
$Tr_Y = 3-2 = 1$

$Tr_Z = 66-3 = 63$

Avg Tr = (16+6+1+63)/4 = 21.5

Throughput = 4/66 = 0.063

Shortest Remaining Time Next



Waiting Time:

$Wt_W = 7-1 = 6$

$Wt_X = 3-2 = 1$

$Wt_Y = 0$

$Wt_Z = 16 – 3 = 13$

Avg Wt = (6+1+0+13)/4 = 5

Response Time:

$Rt_W = 0-0 = 0$

$Rt_X = 1-1 = 0$

$Rt_Y = 2-2 = 0$

$Rt_Z = 16-3 = 13$

Avg Rt = 13/4 = 3.25

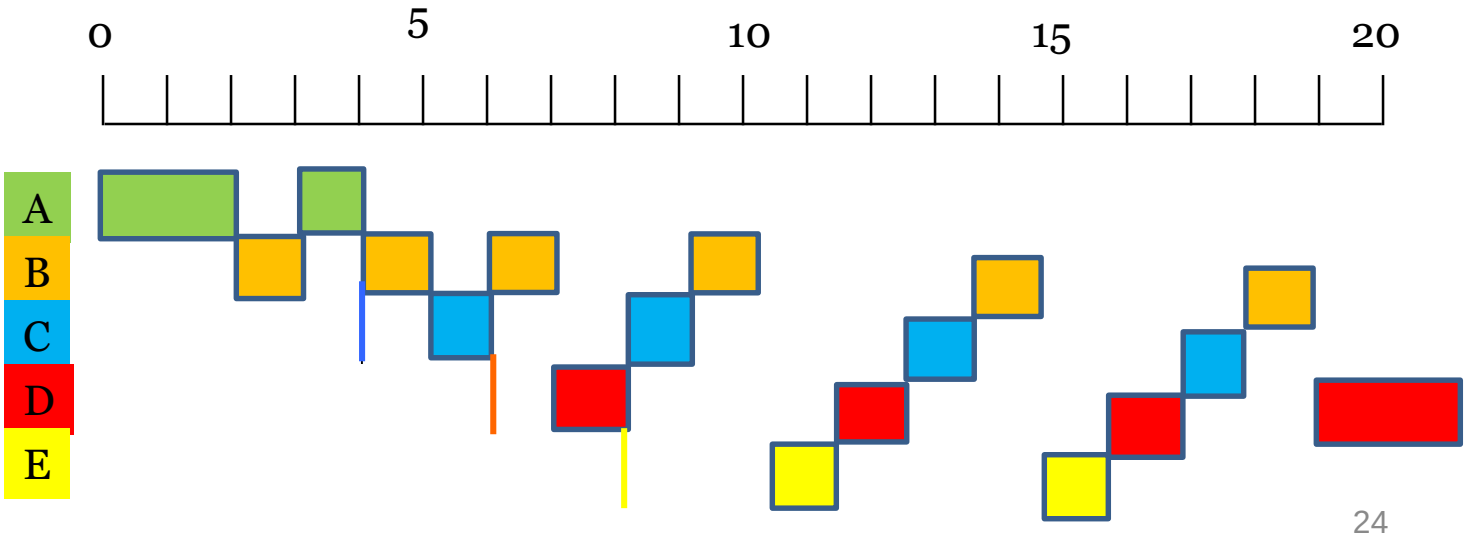
Round-Robin Scheduling

- One of the oldest, simplest, fairest, and most widely used algorithm.
- Each process is assigned a time interval, called its **quantum**, which it is allowed to run.
- At the end of the quantum, the CPU is preempted and given to another process
- Round robin is easy to implement
  - scheduler needs to maintain a list of runnable processes
  - When the process uses up its quantum, it is put on the end of the list
- Limitation
  - Overhead involved in handling the clock interrupt and performing the scheduling and dispatching function
  - Relative treatment of processor-bound and I/O-bound processes

Round-Robin Scheduling

Process	Arrival Time	Service Time	Start Time	Finish Time	Turnaround Time	Tr/Ts
A	0	3	0	4	4	1.33
B	2	6	2	19	17	2.83
C	4	4	5	18	14	3.50
D	6	5	7	21	15	3
E	8	2	10	16	9	4.5

q=1



Round-Robin Scheduling

Process	Arrival Time	Service Time	Start Time	Finish Time	Turnaround Time	Tr/Ts
A	0	3	0	4	4	1.33
B	2	6	2	19	17	2.83
C	4	4	5	18	14	3.50
D	6	5	7	21	15	3
E	7	2	10	16	9	4.5

Turnaround time:

TrA = 4-0 = 4

TrB = 19-2 = 17

TrC = 18-4 = 14

TrD = 21-6 = 15

TrE = 16-7 = 9

Avg Tr = (4+17+14+15+9)/5 = 11.8

Response Time:

RtA = 0-0 = 0

RtB = 2-2 = 0

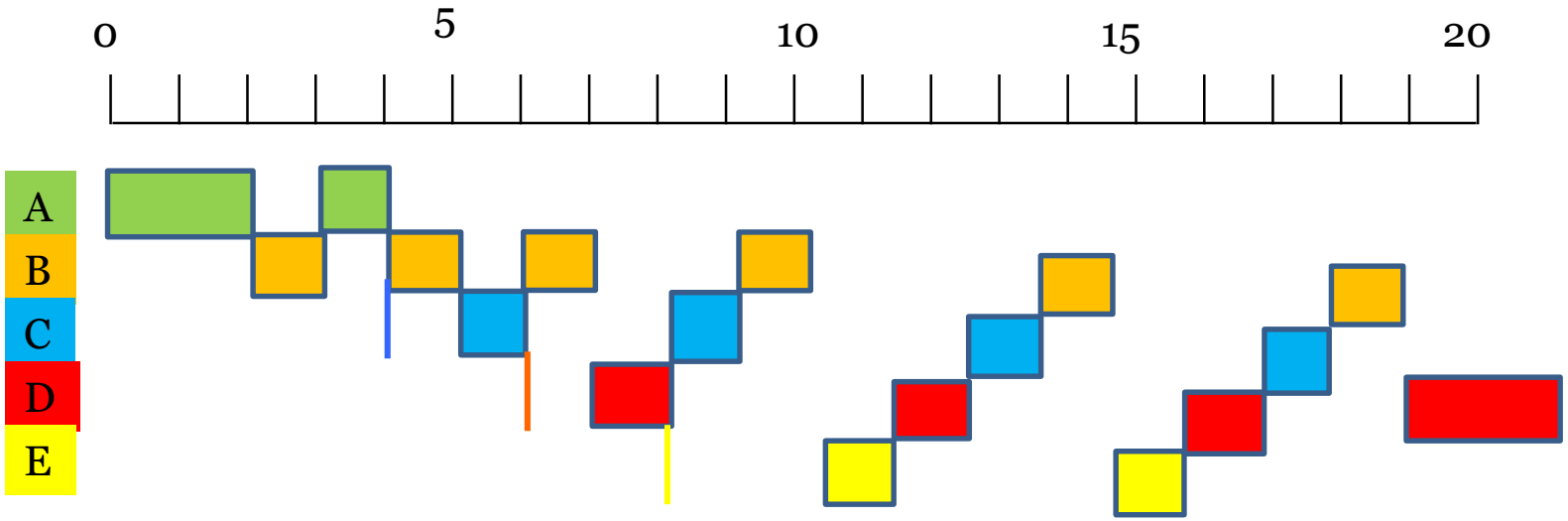
RtC = 5-4 = 1

RtD = 7-6 = 1

RtE = 10-7 = 3

Avg Rt = (0+0+1+1+3)/5 =1

Round-Robin Scheduling



Waiting time:

WtA = 1

WtB = 1+1+2+3+3 = 10

WtC = 1+2+3+3 = 9

WtD = 1+3+3+2 = 9

WtE = 2+3 = 5

Avg Wt = (9+9+5)/5 =4.6

Priority Scheduling

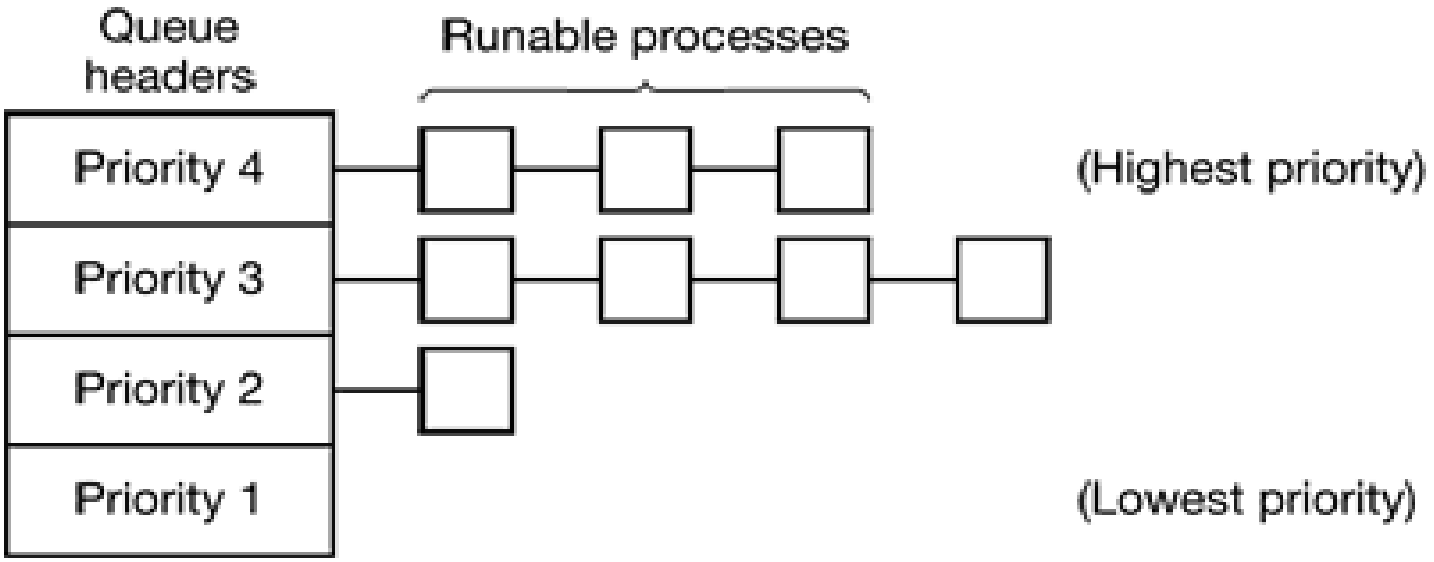
- Each process is assigned a priority (int), and the runnable process with the highest priority is allowed to run
- High priority process may run indefinitely (starvation of low priority processes)
  - Aging - the scheduler may decrease the priority of the currently running process at each clock tick
  - Alternatively, each process may be assigned a maximum time quantum that it is allowed to run
- Priorities can be assigned to processes
  - Statically
    - Predefined before the system starts
  - Dynamically
    - I/O-bound processes might be given high priority by the system

Priority Scheduling

Process	Arrival Time	Processing Time	Priority
A	0	8	3
B	1	3	1 (lowest)
C	3	5	2
D	5	2	10 (highest)

Multiple Queue Scheduling

- It is often convenient to group processes into priority classes:
  - use priority scheduling among the classes
  - round-robin scheduling within each class



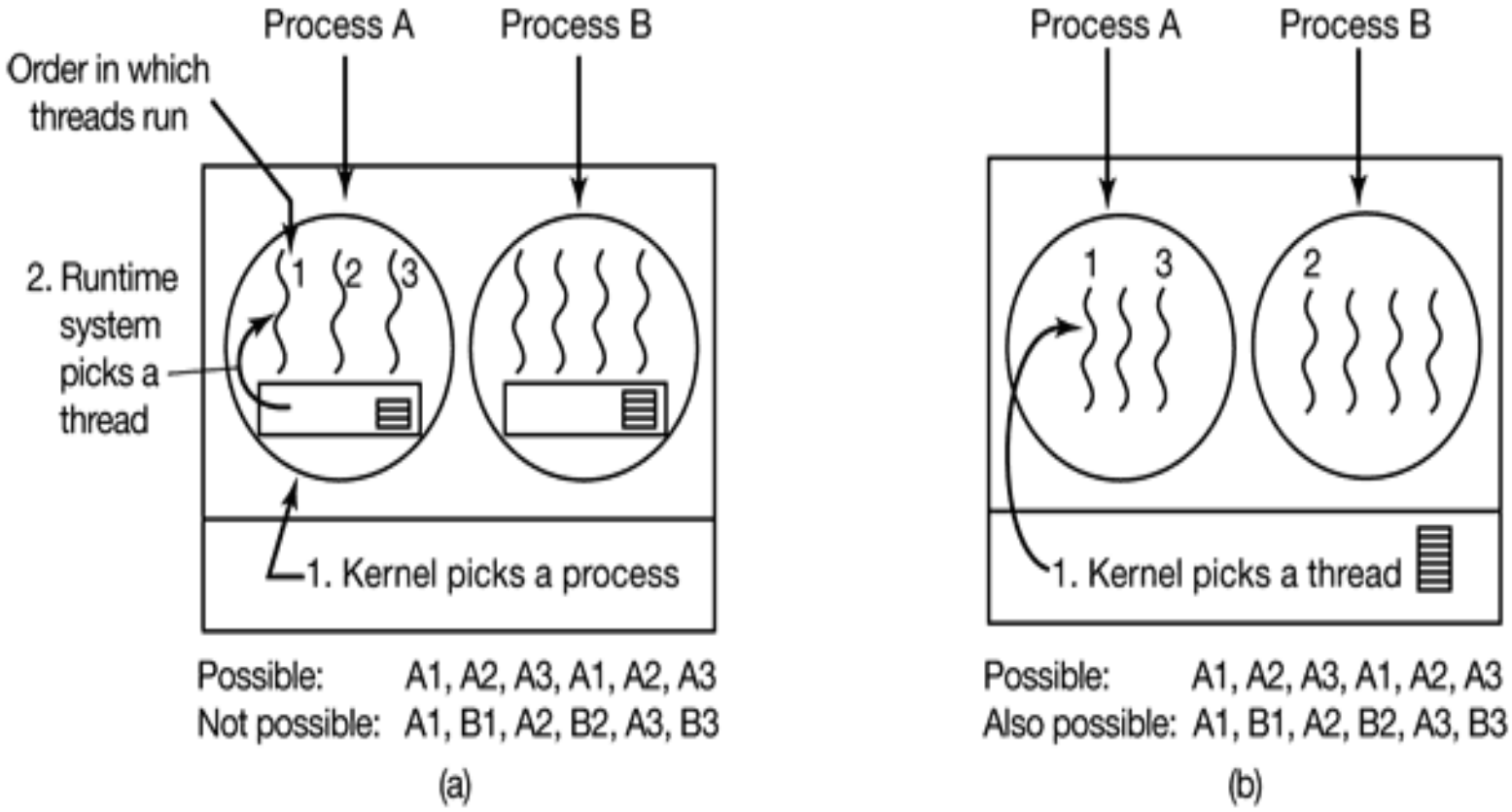
Thread Scheduling

- **User-level threads:**
  - Since the kernel is not aware of the existence of threads, it operates as it always does, picking a process
  - The runtime system of the process decides which thread to run next
  - Since there are no clock interrupts to threads, this thread may continue running as long as it wants to
  - If it uses up the process' entire quantum, the kernel will select another process to run
  - Round-robin scheduling and priority scheduling are most common
  - The only constraint is the absence of a clock to interrupt a thread that has run too long

Thread Scheduling

- Kernel-level threads
  - Here the kernel picks a particular thread to run
  - The thread is given a quantum and is forceably suspended if it exceeds the quantum
- With kernel-level threads it requires a full context switch, changing the memory map, and invalidating the cache, which is several orders of magnitude slower➔ **Low performance (system)**
- On the other hand, with kernel-level threads, having a thread block on I/O does not suspend the entire process as it does with user-level threads ➔ **Higher Performance (process)**

Thread Scheduling



(a) Possible scheduling of user-level threads with a 50-msec process quantum and threads that run 5 msec per CPU burst.

(b) Possible scheduling of kernel-level threads with the same characteristics as (a)

Multiple-Processor Scheduling

We can classify multiprocessor systems as follows:

- **Loosely coupled or distributed multiprocessor, or cluster:**
  - Consists of a collection of relatively autonomous systems, each processor having its own main memory and I/O channels
- **Functionally specialized processors:**
  - there is a master, general-purpose processor; specialized processors are controlled by the master processor and provide services to it.
- **Tightly coupled multiprocessing:**
  - Consists of a set of processors that share a common main memory and are under the integrated control of an operating system
- In multiprocessor systems threads can be used to exploit true parallelism in an application
- Scheduling algorithms in multiprocessing focuses on thread scheduling

Multiple-Processor Scheduling

- Four general approaches:

1. **Load sharing:**
   - Processes are not assigned to a particular processor.
   - A global queue of ready threads is maintained, and each processor, when idle, selects a thread from the queue
2. **Gang scheduling:**
   - A set of related threads is scheduled to run on a set of processors at the same time, on a one-to-one basis
3. **Dedicated processor assignment:**
   - Each program is allocated a number of processors equal to the number of threads in the program, for the duration of the program execution.
   - When the program terminates, the processors return to the general pool for possible allocation to another program
4. **Dynamic scheduling:**
   - The number of threads in a process can be altered during the course of execution

Real-Time Scheduling

- A **real-time** system is one in which time plays an essential role
- *Hard real-time* systems
  - required to complete a critical task within a guaranteed amount of time
- *Soft real-time* computing
  - requires that critical processes receive priority over less fortunate ones
- Aperiodic events
  - Events which occur unpredictably that a real-time system may have to respond
- **Periodic** events
  - occurring at regular intervals
- Real-time systems which responds to periodic events are schedulable