

Operating Systems 2024/2025

T Class 06 – CPU Scheduling (1/2)

Vasco Pereira (vasco@dei.uc.pt)

Dep. Eng. Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

operating system

noun

the collection of software that directs a computer's operations, controlling and scheduling the execution of other programs, and managing storage, input/output, and communication resources.

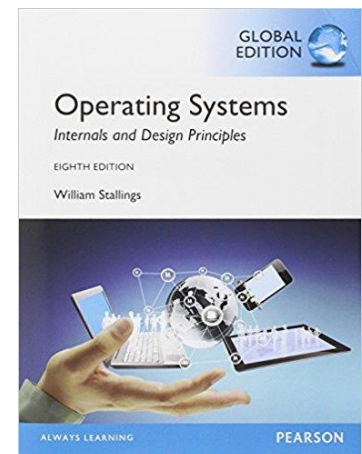
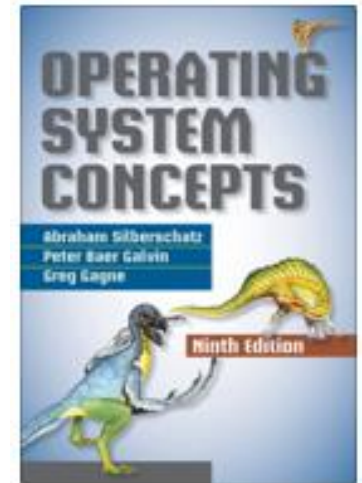
Abbreviation: OS

Source: Dictionary.com

Disclaimer

- This slides and notes are based on the companion material [Silberschatz13]. The original material can be found at:
 - <http://codex.cs.yale.edu/avi/os-book/OS9/slide-dir/>
- In some cases, material from [Stallings15] may also be used. The original material can be found at:
 - <http://williamstallings.com/OS/OS5e.html>
 - <http://williamstallings.com/OperatingSystems/>
- The respective copyrights belong to their owners.

Note: Some slides are also based on previous versions from Bruno Cabral, Paulo Marques and Luis Silva (Operating Systems classes of DEI-FCTUC).



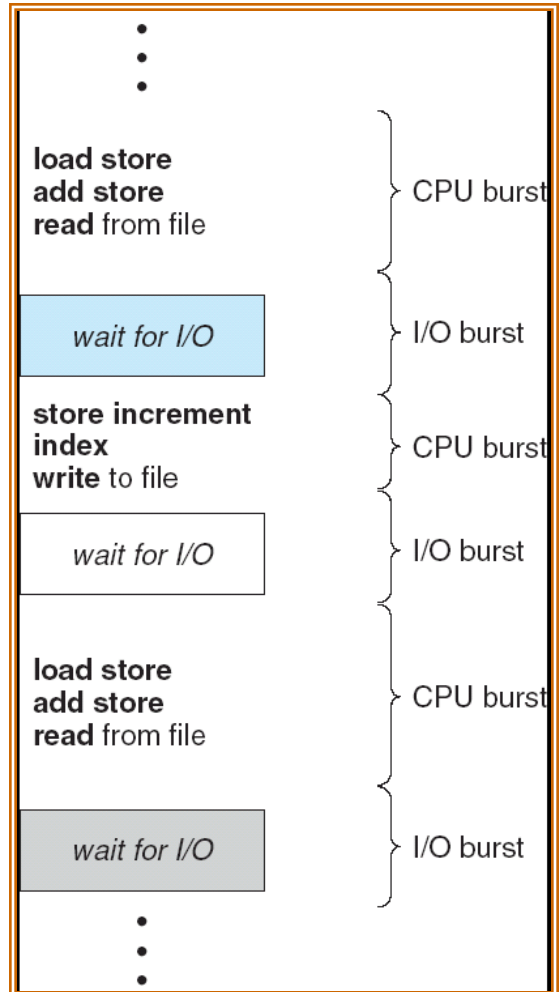
- Today's outlines:
 - CPU scheduling of processes in a single processor
 - Scheduling algorithms

Why CPU scheduling?

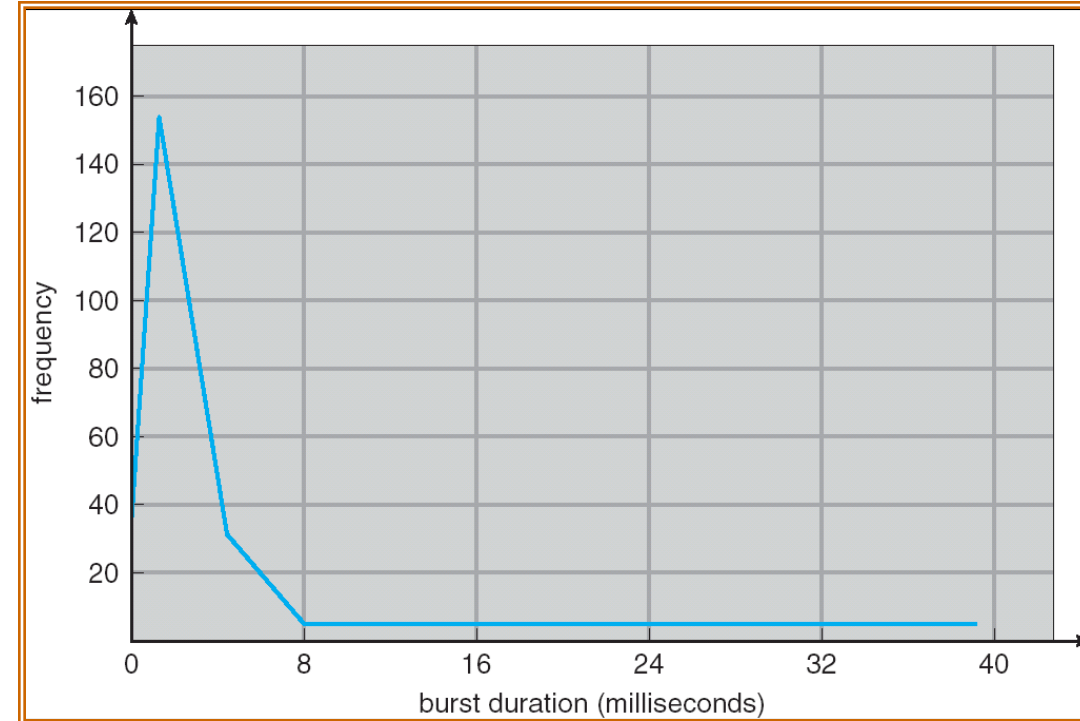
- It is the basis of multiprogrammed OSs
- It allows one process to use the processor while all others are kept on hold
- It maximizes the CPU utilization by putting on hold all processes that are waiting for some resource (e.g. I/O)
- CPU scheduling aims to make the system efficient, fast and fair
- On OSs that support kernel-level threads, it is threads that are being scheduled, not processes

Why can we optimize?

CPU-I/O Cycle



CPU-burst Times

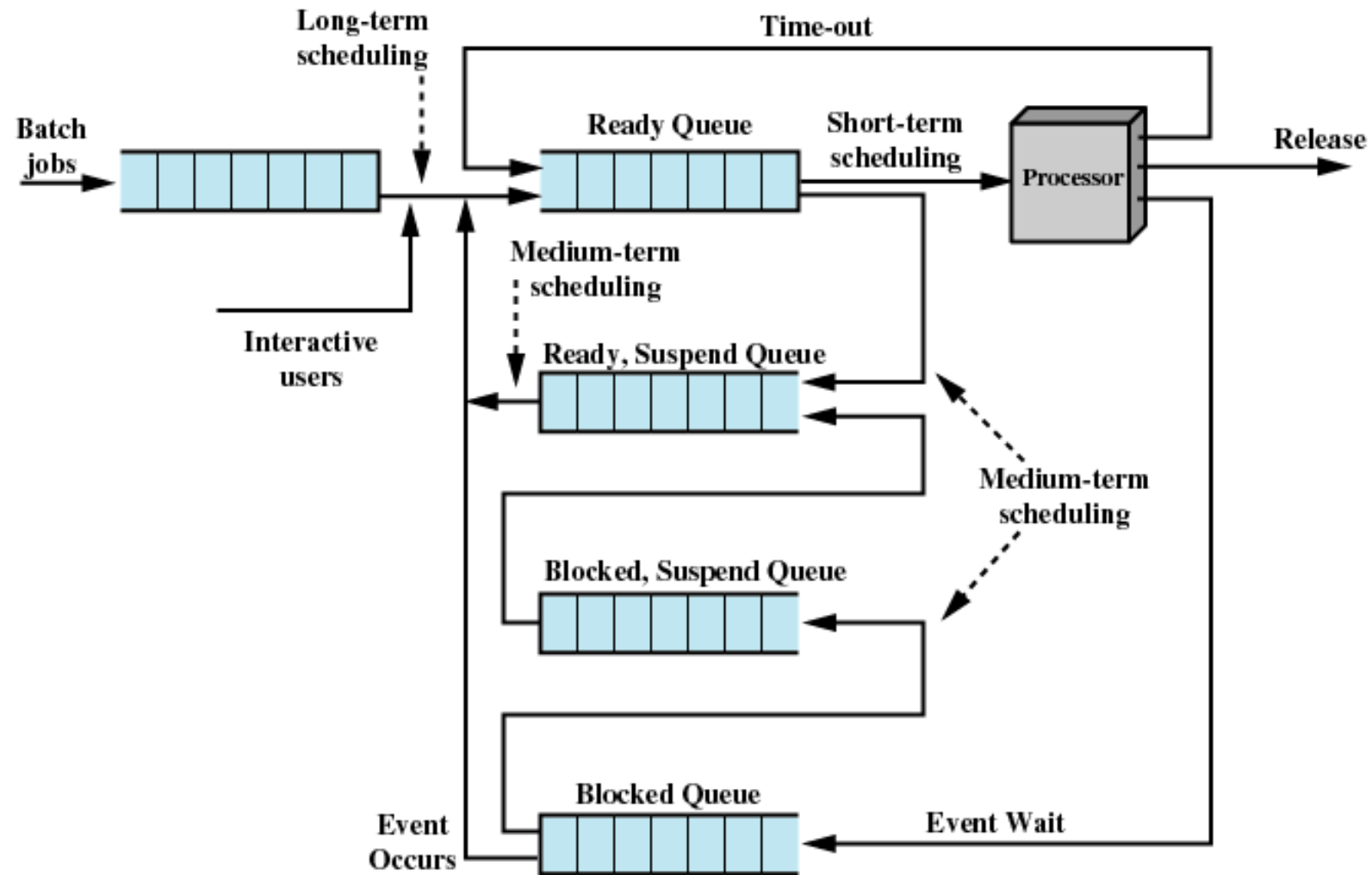


Statistically, processes in a computer tend to have a large number of short CPU bursts and a small number of long CPU bursts.

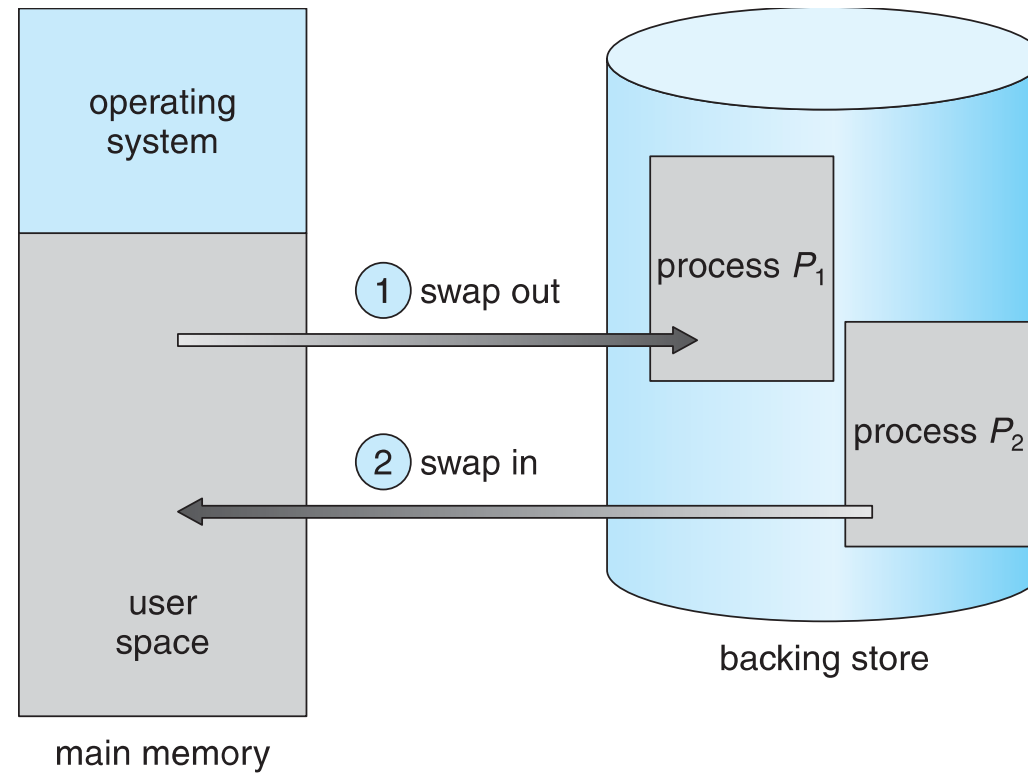
CPU scheduler

- When a CPU becomes idle the OS must select one process from the **ready queue** to be executed
 - The selection is done by **the short-term scheduler**
 - It selects a process from the ones in memory that are ready to execute
- When more processes are submitted than can be executed immediately (e.g., in a batch system) they are spooled to a mass-storage device (e.g., disk).
 - The **long-term scheduler/job scheduler** selects processes from the waiting list and loads them in memory for execution
- Swapping enables the removal of a process from memory to reduce the level of multiprogramming. The process can be later reintroduced and continue execution
 - The swapping is done by the **medium-term scheduler**

System Queues



Swapping



Decision Mode

Preemptive vs. Non-Preemptive

■ **Non-Preemptive Scheduling**

Once the CPU is allocated to a process, the process keeps the CPU until it terminates or blocks. A process blocks by switching to a waiting state, such as in I/O operations or OS service requests.

- (e.g., in Windows 3.x).

■ **Preemptive Scheduling**

The operating system may decide and is able to take the CPU away from a running process

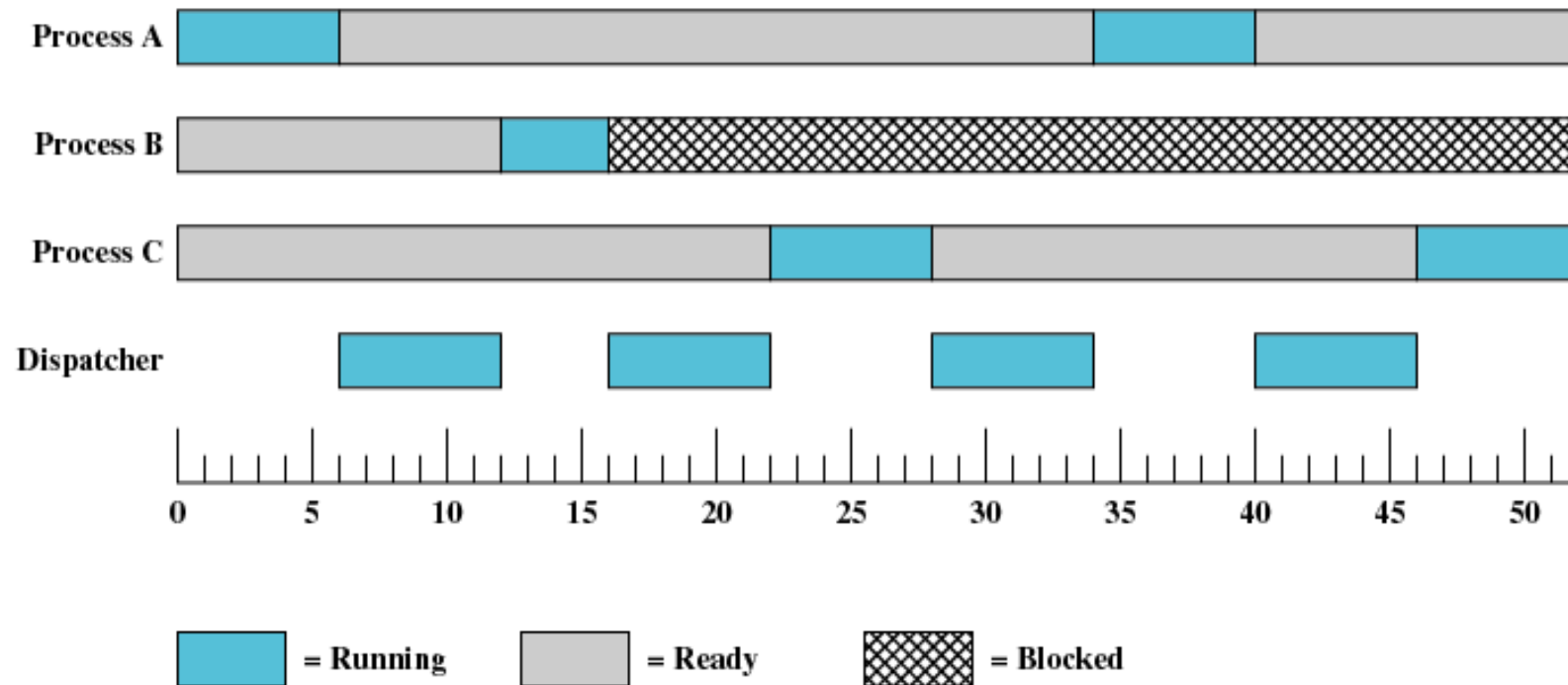
- (e.g., in WindowsXP, Linux, Mac OS X)

Dispatcher

- The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler
 - Switches context
 - Switches to user mode
 - Restarts the program in the proper location
- The dispatcher is invoked during every process switch
 - The time it takes for the dispatcher to stop one process and start another is known as the **dispatch latency**

Dispatcher (2)

■ Dispatcher operation



Scheduling Criteria

- **CPU utilization**

- Keep the CPU as busy as possible

- **Throughput**

- Number of processes that complete their execution per time unit

- **Turnaround time**

- Amount of time to complete a particular process (from submission to completion)

- **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

Optimization Criteria...

- As seen before, it all depends on what type of system we are aiming at...
- But we would like:
 - Maximum CPU utilization
 - Maximum throughput
 - Minimum turnaround time
 - Minimum waiting time
 - Minimum response time

That's a multi-objective function, not really possible...

Scheduling Algorithms


- FCFS (First-Come, First-Served)
- SJF - Shortest Job First (Shortest Process Next) – SPN or SRT
- PRIORITY
- ROUND-ROBIN
- HRRN (Highest Response Rate Next)
- MULTILEVEL
- MULTILEVEL FEEDBACK

Algorithm 1:

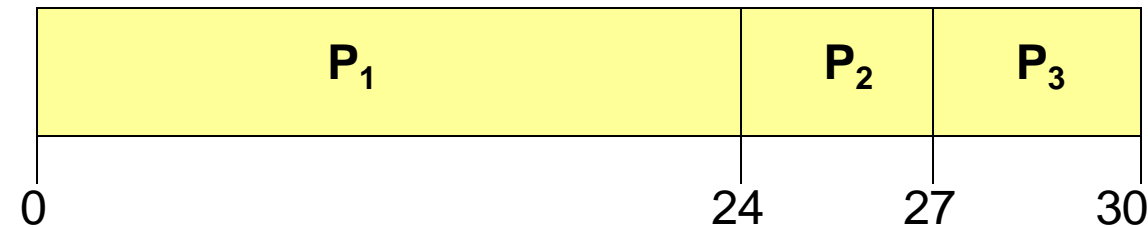
First-Come, First-Served (FCFS)

- The process that requests the CPU first gets it first
- Simple to manage with a FIFO queue
- Average waiting time is often long
- This is a **non-preemptive** algorithm!
 - A process keeps the CPU until it terminates or requests I/O.

First-Come, First-Served (2)

<u>Process</u>	<u>Burst Time</u>	 In time units
P_1	24	
P_2	3	
P_3	3	

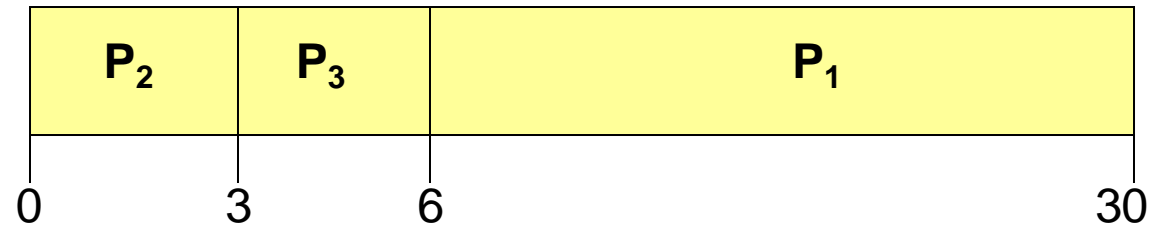
- Suppose that the processes arrive in the order: P_1, P_2, P_3
The Gantt Chart for the schedule is:



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

First-Come, First-Served (3)

- Suppose that the processes arrive in the order: P_2 , P_3 , P_1
- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case.

First-Come, First-Served (4)

- FCFS performs better for long processes, than for short ones
 - *Convoy effect*
 - Short processes arriving after long processes are **penalized**, since they must wait until the long process leaves the CPU
- I/O bound vs. CPU bound problem
 - FCFS tends to favor processor-bound over I/O-bound processes

Algorithm 2 and 3:

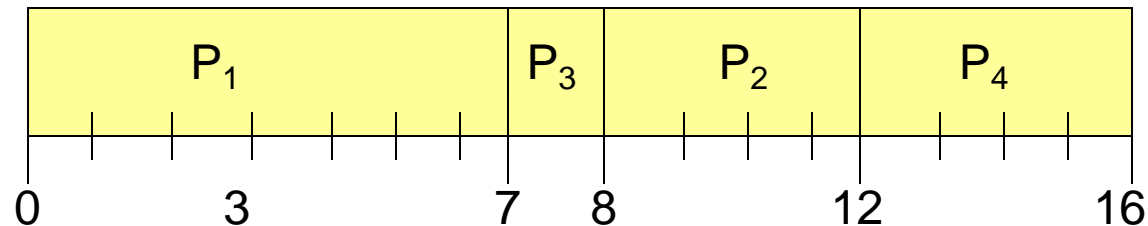
Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the **shortest time**.
 - How do you determine that? Can you guess the future?
 - Also known as **Shortest-Job-Next (SJN)**.
- Two schemes:
 - **Non-preemptive**: once CPU given to the process it cannot be preempted until completes its CPU burst. This scheme is known as **Shortest-Process-Next (SPN)**.
 - **Preemptive**: if a new process **arrives** with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the **Shortest-Remaining-Time First (SRTF)** or **SRT**.

SJF (Non-Preemptive) - SPN

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SPN (non-preemptive)

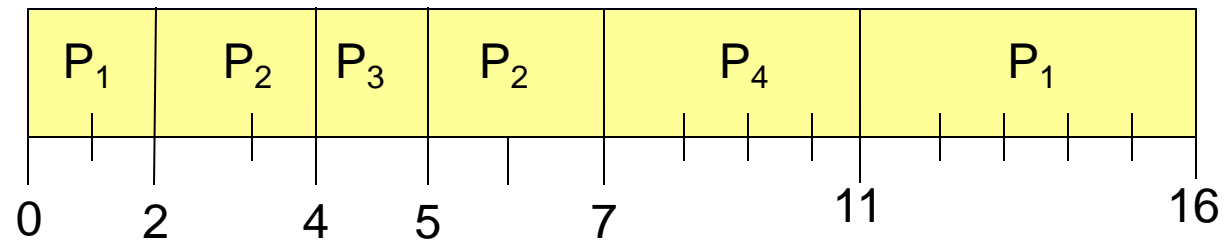


- Average waiting time = $(0 + 6 + 3 + 7)/4 = 4$
- Used frequently in long-term scheduling; in short-term is more difficult to assess the CPU time burst

SJF (Preemptive) - SRT

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SRT (preemptive)

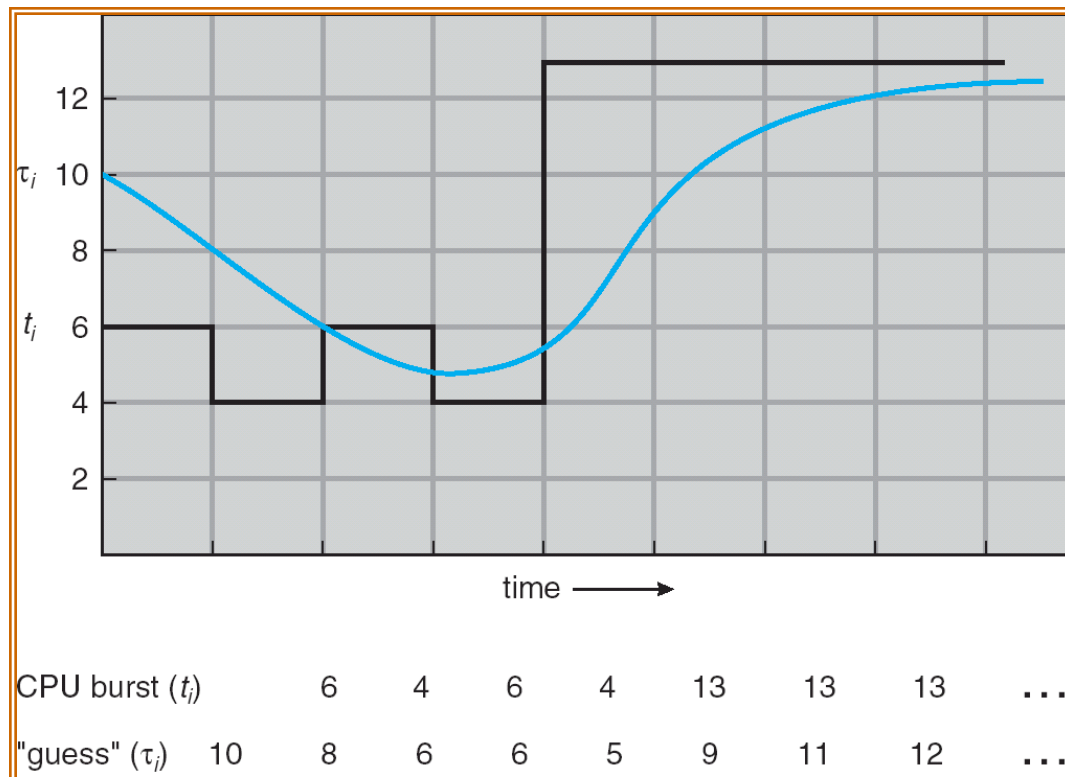


- Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$

How to predict the next CPU burst time?

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

- Based on an exponential moving average based on the past.



Algorithm 4: Priority Scheduling

- A priority number is associated with each process (can be anything measurable)
- The CPU is allocated to the process with the highest priority:
 - Preemptive
 - Non-preemptive
- Equal priority is scheduled in FCFS order
 - If two processes have the same priority, the one closer to the head of the ready-queue is chosen.
- SJF can be seen as priority scheduling where priority is the *predicted next CPU burst time*

Priority Scheduling (2)

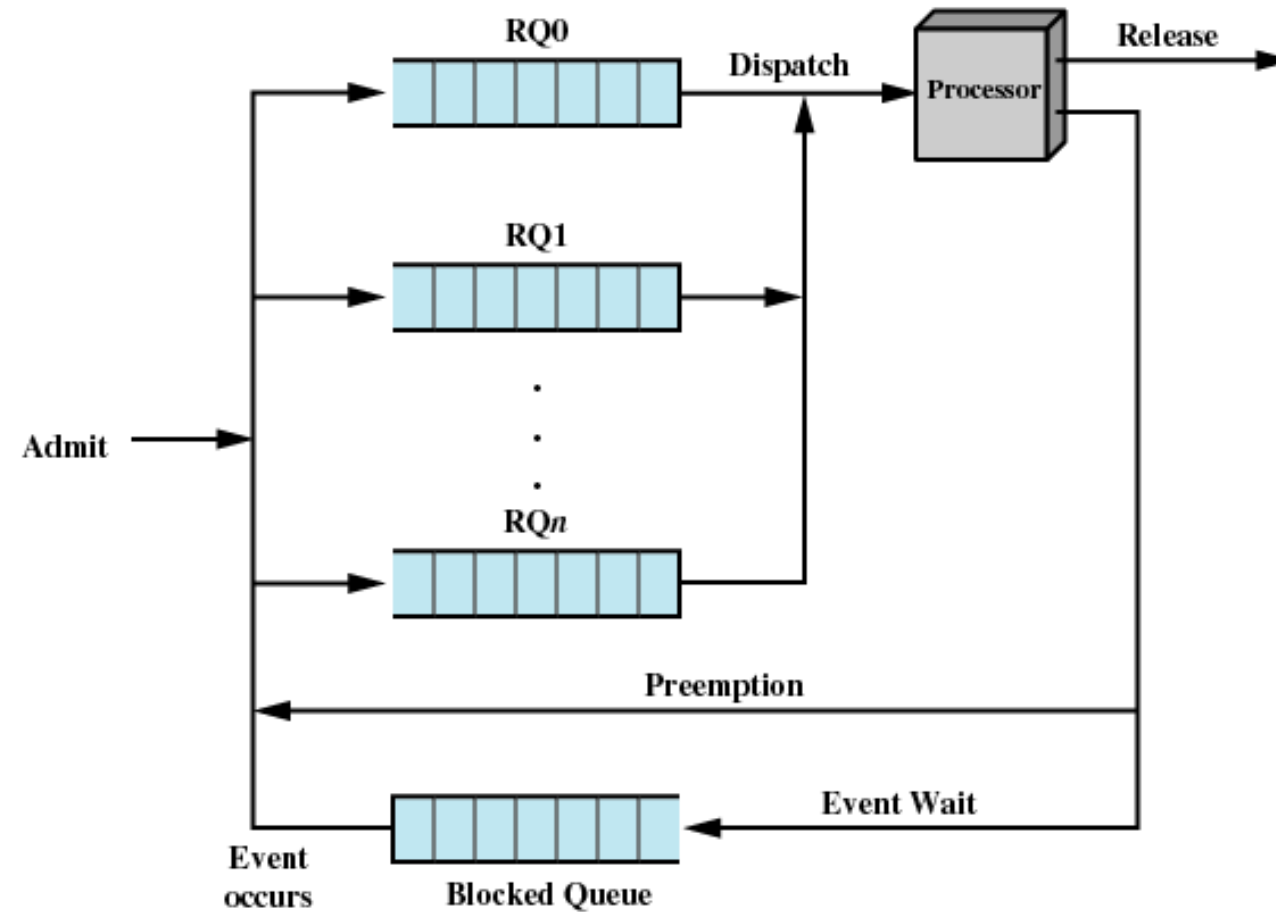
- Problem: Starvation
 - Low priority processes may never execute
- Solution -> Aging
 - As time progresses, the priority of the process increases

Starvation @MIT

When the system managers shutdown the IBM 7094 at MIT in 1973, they found a low-priority process that had been submitted in 1967 and had not yet run...



Priority-Based Scheduler



Algorithm 5: Round-Robin (RR)

- Is designed especially for time-sharing systems
- Similar to FCFS with added preemption
 - Reduces the penalty that short jobs suffer with FCFS by using preemption based on a clock.
- 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.
 - New processes are added to the **tail** of the ready queue
 - If, at a given instant, a process depletes its time quantum (and still has service time) and if at that instant a new process arrives to the system, the new process is placed in the queue in front of the process that just executed (but behind other processes that are already in the queue)

Round-Robin (2)

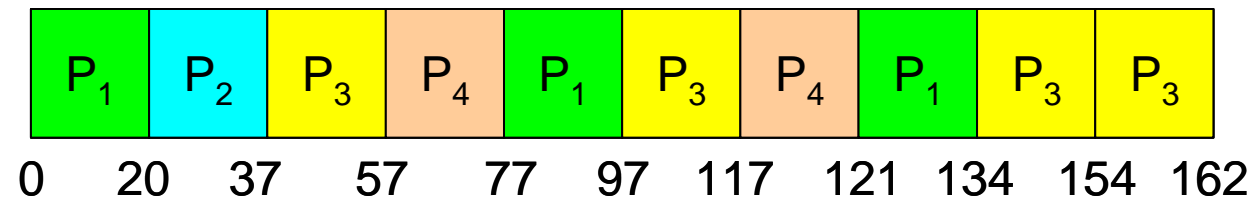
- If there are n processes in the ready queue and the time quantum is q , then each process gets $1/n$ of the CPU time in chunks of at most q time units at once.
 - No process waits more than $(n-1)q$ time units.
 - If the process CPU burst is less than q , the process will by itself release the CPU voluntarily (e.g., I/O request, process end)
- **Performance**
 - q large \Rightarrow FCFS
 - q small $\Rightarrow q$ must be large with respect to context switch, otherwise overhead is too high. If so, having N processes is like having a machine running at $1/N$ of the speed.

RR with a quantum of 20

<u>Process</u>	<u>Burst Time</u>
P_1	53
P_2	17
P_3	68
P_4	24

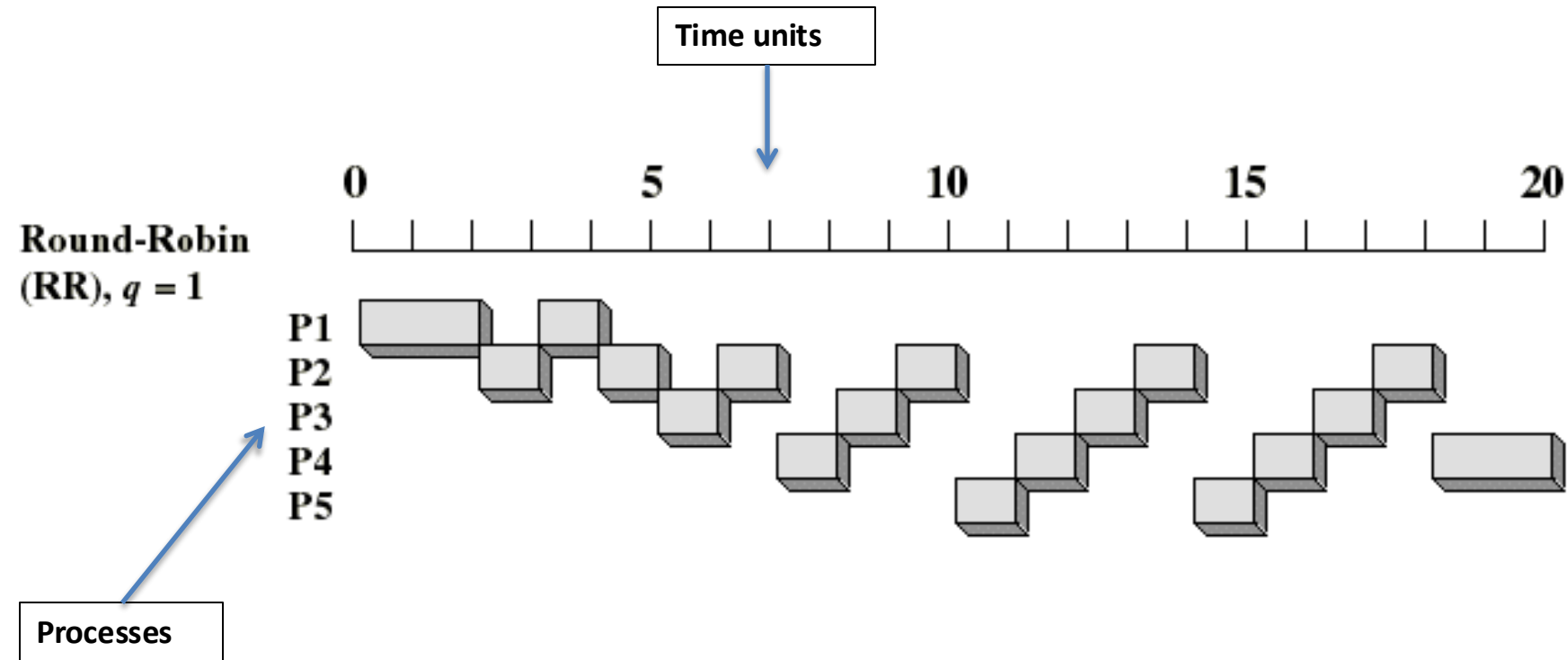
- Note: all processes are ready to execute by the order of their number

- The Gantt chart is:

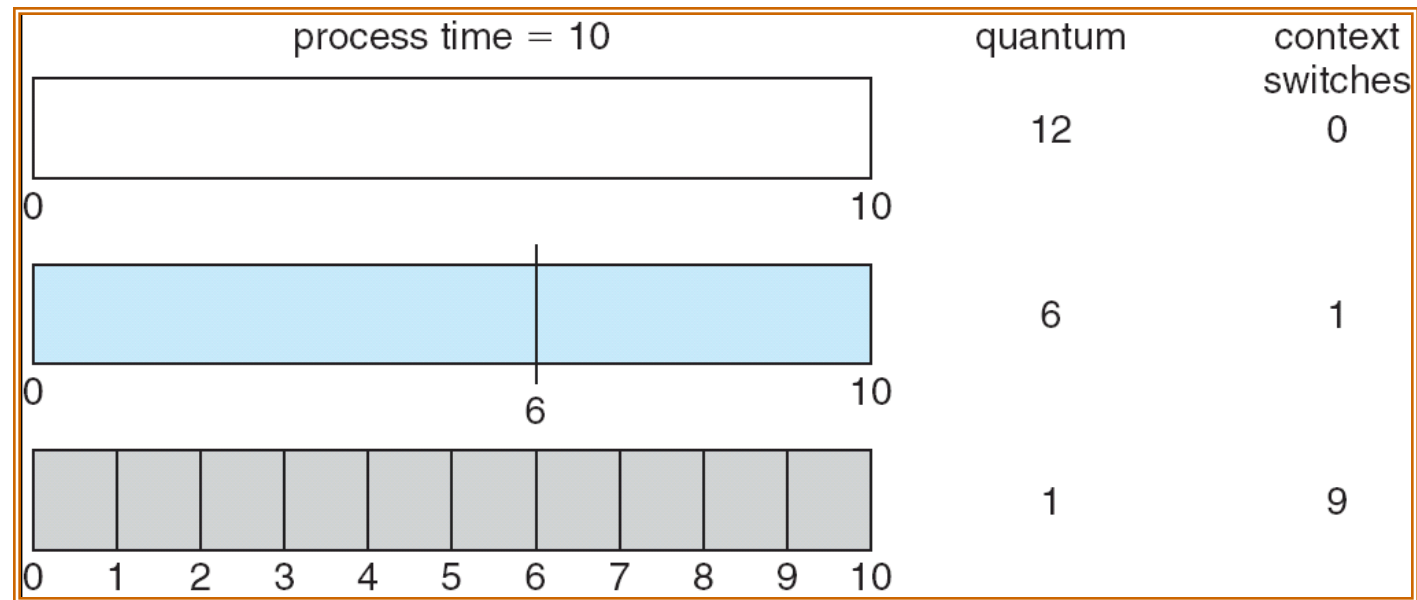


- Typically, higher average turnaround than SJF, but better *response time*

Timeline charts are relevant!

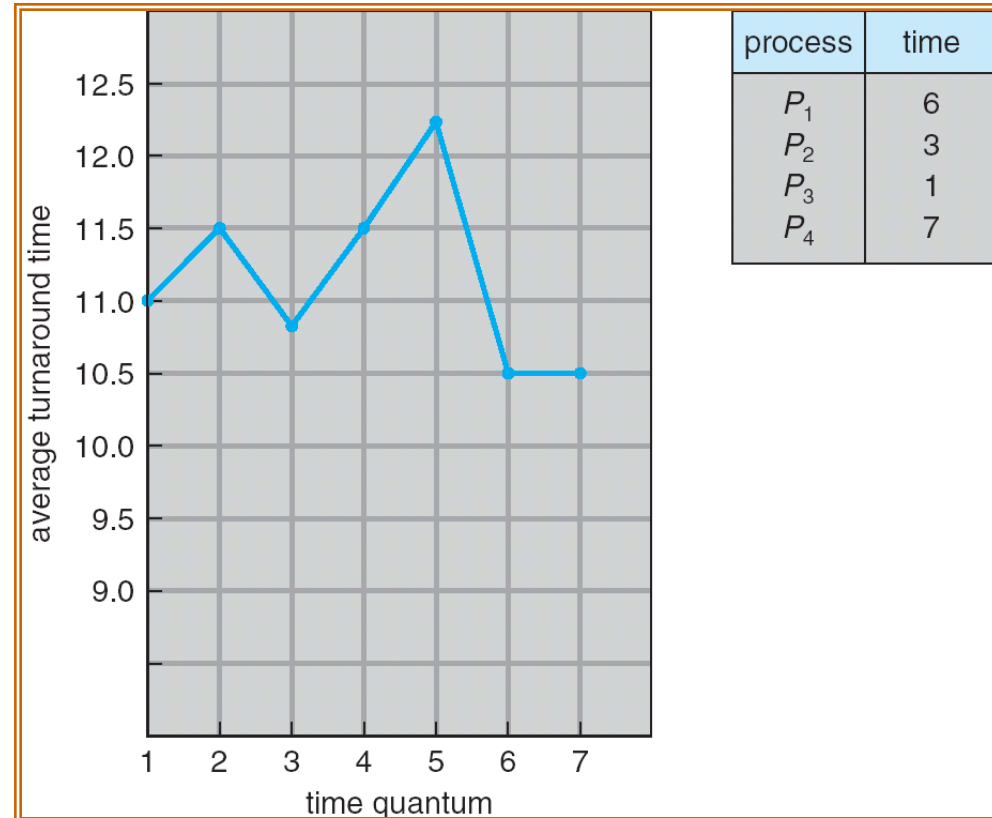


Time Quantum and Context Switches



Turnaround Time vs. Time Quantum

- Turnaround time depends on time quantum, but the relation is not direct.
- In general, turnaround time can be improved if most processes finish next CPU burst in a single time quantum.



Algorithm 6: Highest Response Ratio Next (HRRN)

- Choose next process with the highest ratio (R):

$$R = \frac{\text{time spent waiting} + \text{expected service time}}{\text{expected service time}}$$

- This algorithm accounts for the age of the process.
- Short jobs are favored (smaller denominator)
 - ex: for a waiting of 1 $\Rightarrow \frac{1+3}{3} > \frac{1+4}{4} \Rightarrow$ short job is favored
- Aging without service (waiting time) increases R.
- If Rs are equal, then uses FCFS
- Non-preemptive!**

Algorithm 7: Multilevel Queue Scheduling

- Used when processes are easily classified into different groups (e.g., foreground vs. background)
- Each process is assigned to one specific queue based on its specificities, and each queue has its own scheduling algorithm
- There is an additional scheduling among queues.

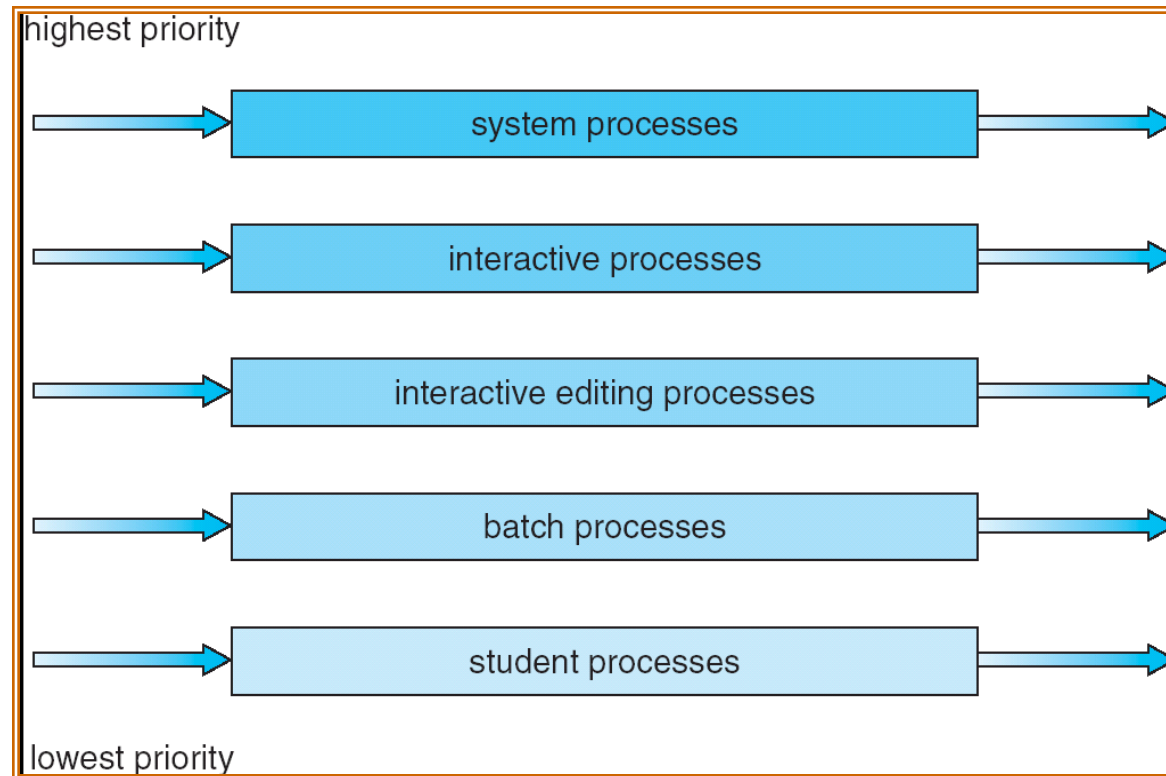
Multilevel Queue Scheduling (2)

Example

- Ready queue is partitioned into separate queues. E.g. :
 - Foreground (interactive)
 - Background (batch)
- Each queue has its own scheduling algorithm:
 - Foreground – RR
 - Background – FCFS
- Scheduling must be done between the queues.
 - **Fixed priority scheduling:** high priority queues must be empty for the lower to execute its processes (i.e., serve all processes from foreground; then from background). 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 Scheduling (3)

- Other example:



Algorithm 8: Multilevel Feedback Queue Scheduling

- Normally, when using Multilevel Queue Scheduling, processes remain in the queue they were assigned when entering the system.
 - Low scheduling overhead
- With **Multilevel Feedback Queue Scheduling**, **processes can move between queues!**
 - **Aging** can be implemented this way

Multilevel Feedback Queue Scheduling (2)

- Only when the high priority queues are empty will the lower have their processes executed
- Normally (not always!), a process that arrives for a higher priority queue will preempt a process from a lower priority queue

Multilevel Feedback Queue Scheduling (3)

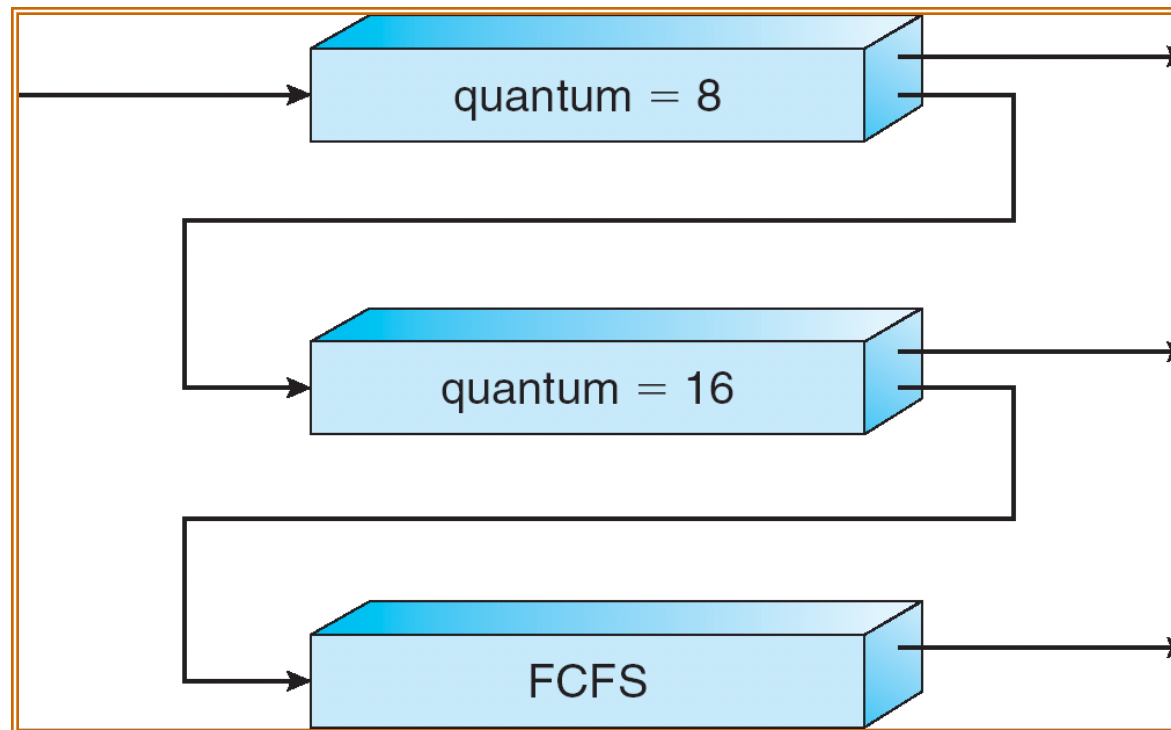
- The Multilevel-feedback-queue scheduler is defined by the following parameters:
 - number of queues
 - scheduling algorithms for each queue
 - method used to determine when to upgrade a process
 - method used to determine when to demote a process
 - method used to determine which queue a process will enter when that process needs service (when it first enters)

- Many configurations are possible! It can be configured to address specific needs.

Multilevel Feedback Queue Scheduling (4)

- Simple version of a Multilevel Feedback approach
 - Sometimes there is no indication of the length of the processes and then SPN, SRT and HRRN cannot be used.
 - A way to establish preferences is to penalize jobs that have been running longer
 - It focus on the time spent until now (that is known) instead of focusing on the time remaining to execute (most of the times unknown).
 - A process is demoted to a lower priority queue whenever it is preempted.

Example of a scheduler with three queues

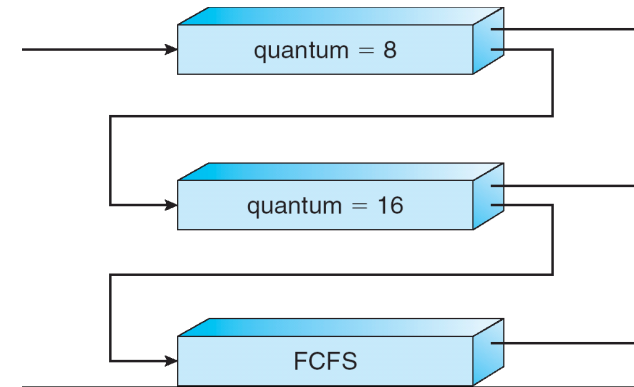


Note: queue serving is priority-based!

Example of a scheduler with three queues (2)

- Three queues:

- Q_0 – time quantum 8 milliseconds
- Q_1 – time quantum 16 milliseconds
- Q_2 – FCFS



- Scheduling

- A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1 .
- At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .
- Gives high-priority to any process with CPU bursts of 8ms or less.
 - Grabs CPU, do processing, move along to next IO cycle...
- Process with bursts of >8ms and <16ms are served quickly but with less priority
- Long running processes move to Q_2 and are served on any CPU cycles remaining.

Characteristics of Scheduling Algorithms

	Selection Function	Decision Mode	Throughput	Response Time	Overhead	Effect on Processes	Starvation
FCFS	$\max[w]$	Nonpreemptive	Not emphasized	May be high, especially if there is a large variance in process execution times	Minimum	Penalizes short processes; penalizes I/O bound processes	No
Round Robin	constant	Preemptive (at time quantum)	May be low if quantum is too small	Provides good response time for short processes	Minimum	Fair treatment	No
SPN	$\min[s]$	Nonpreemptive	High	Provides good response time for short processes	Can be high	Penalizes long processes	Possible
SRT	$\min[s - e]$	Preemptive (at arrival)	High	Provides good response time	Can be high	Penalizes long processes	Possible
HRRN	$\max\left(\frac{w + s}{s}\right)$	Nonpreemptive	High	Provides good response time	Can be high	Good balance	No
Feedback	(see text)	Preemptive (at time quantum)	Not emphasized	Not emphasized	Can be high	May favor I/O bound processes	Possible

w = time spent waiting
 e = time spent in execution so far
 s = total service time required by the process, including e

[Stallings04]

From Stallings

SCHEDULING EXAMPLES

Example

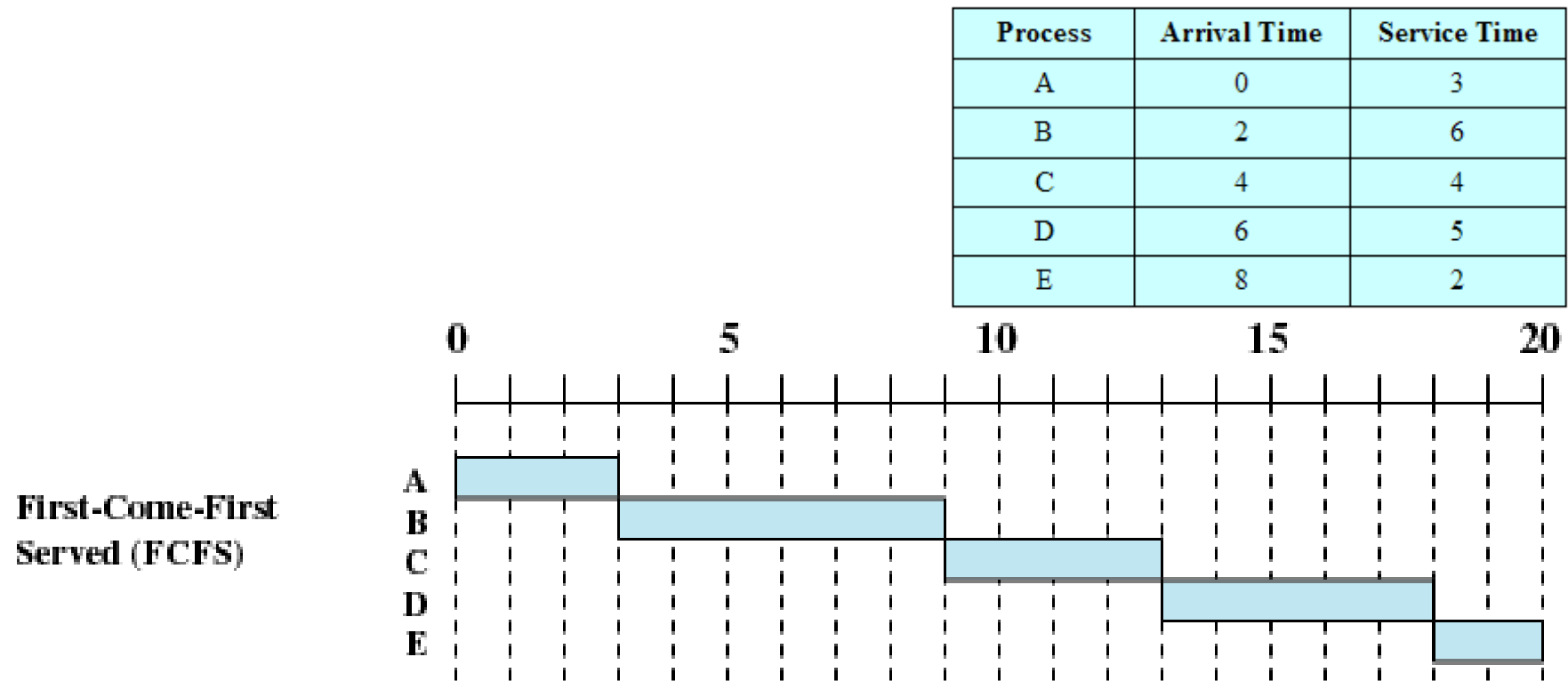
- This example is taken from [Stallings04]

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

- **Tasks**

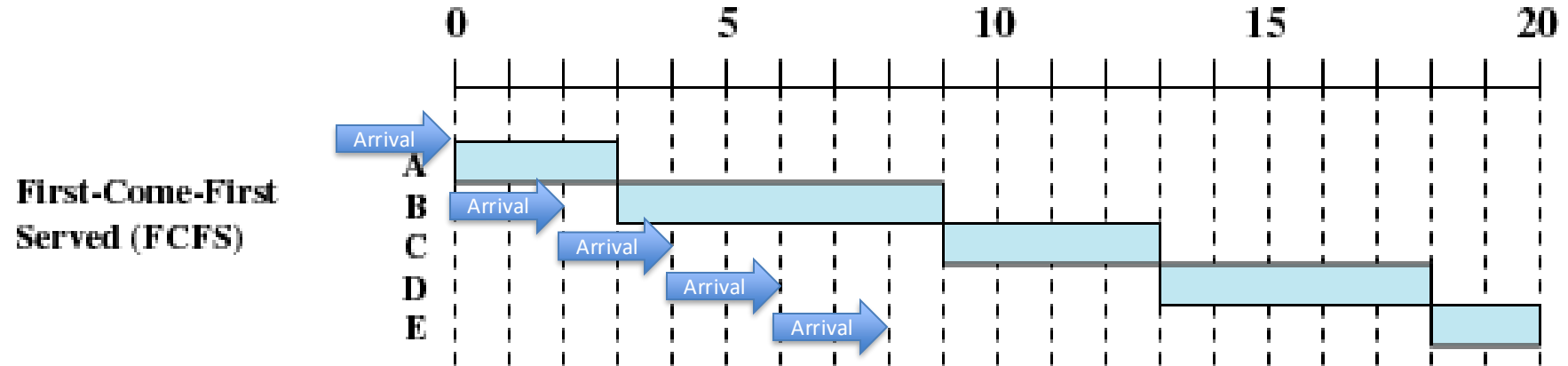
- Calculate **Finish-Time**, **Waiting-Time**, **Turnaround Time** for each process
- Calculate **Average Waiting Time** and **Average Turnaround Time**

Example – FCFS



- Short process may have to wait a very long time before it can execute
- Favors CPU-bound processes
- I/O processes must wait until CPU-bound process completes

Example – FCFS

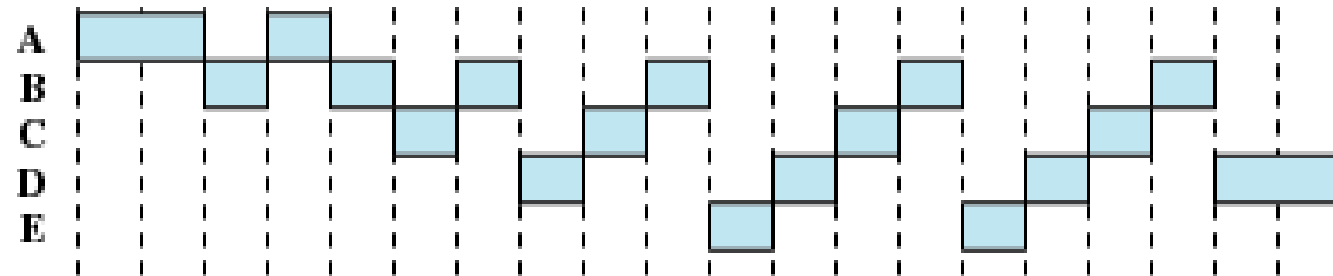


FCFS	A	B	C	D	E	Avg.
Finish Time	3	9	13	18	20	-
Waiting time	0	1	5	7	10	4.60
Turnaround Time	3	7	9	12	12	8.60

Example – Round-Robin ($q=1$)

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

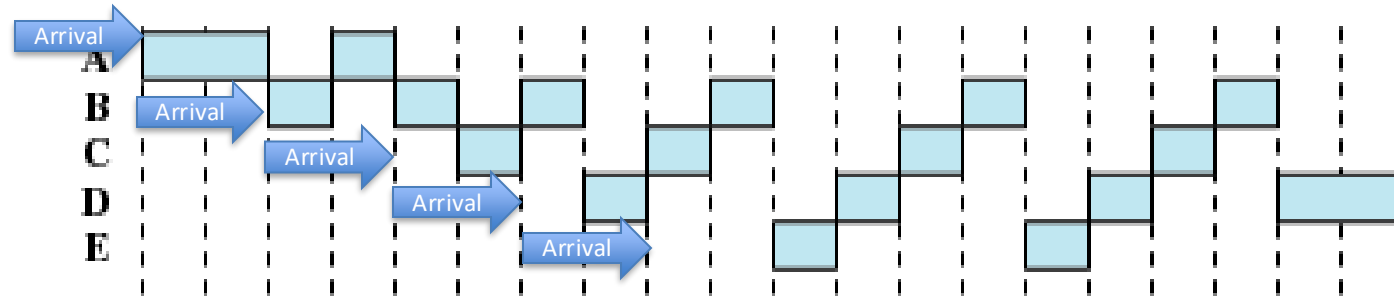
Round-Robin
(RR), $q = 1$



- Uses preemption based on a clock
- When an interrupt occurs, the running process is placed in the ready queue
- Next ready job is selected
- Known as time slicing
- Assume that when a new process arrives and, at the same time, the running process finishes his quantum, it is the previously running process that will be placed at the end of the ready queue

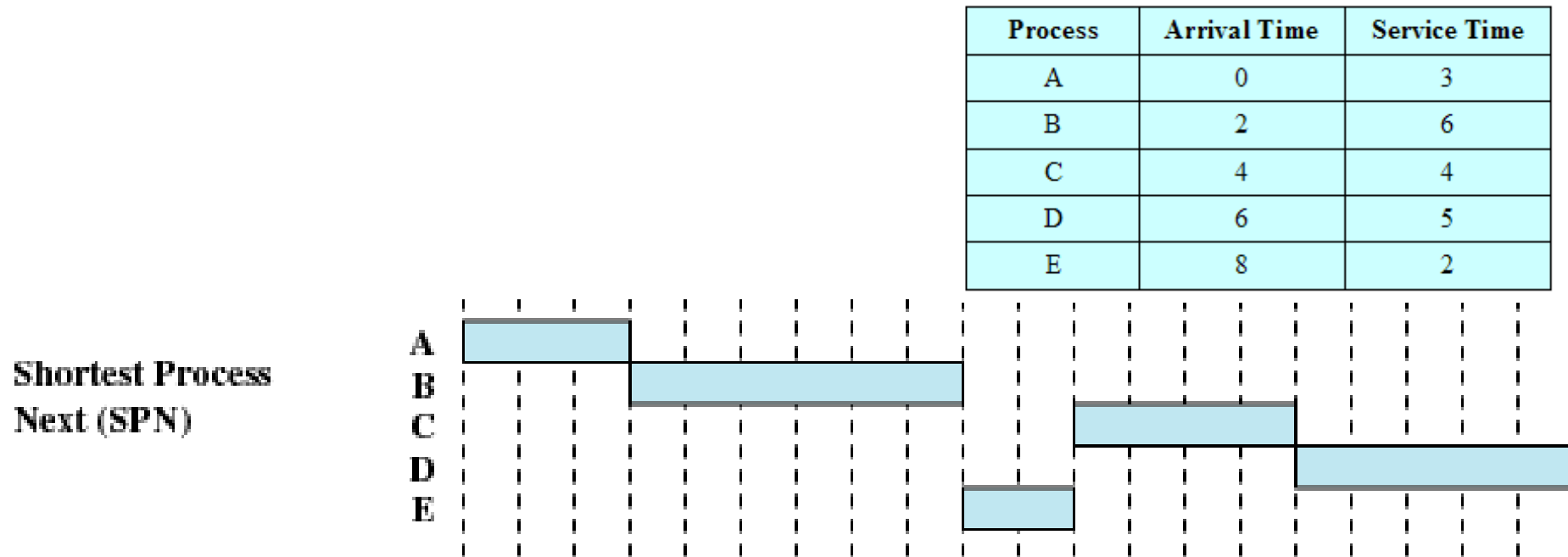
Example – Round-Robin ($q=1$)

Round-Robin (RR), $q = 1$



RR (q=1)	A	B	C	D	E	Avg.
Finish Time	4	18	17	20	15	-
Waiting time	1	10	9	9	5	6.80
Turnaround Time	4	16	13	14	7	10.80

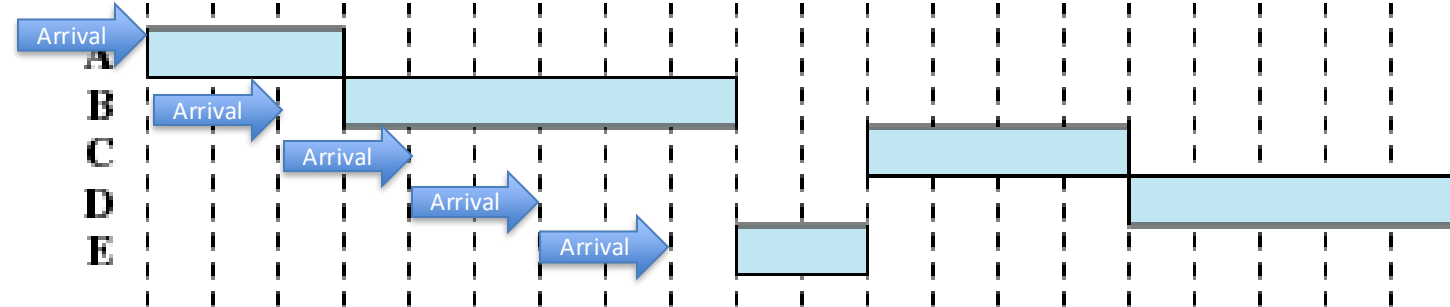
Example – SPN / SJF non-preemptive



- Non-preemptive policy
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes

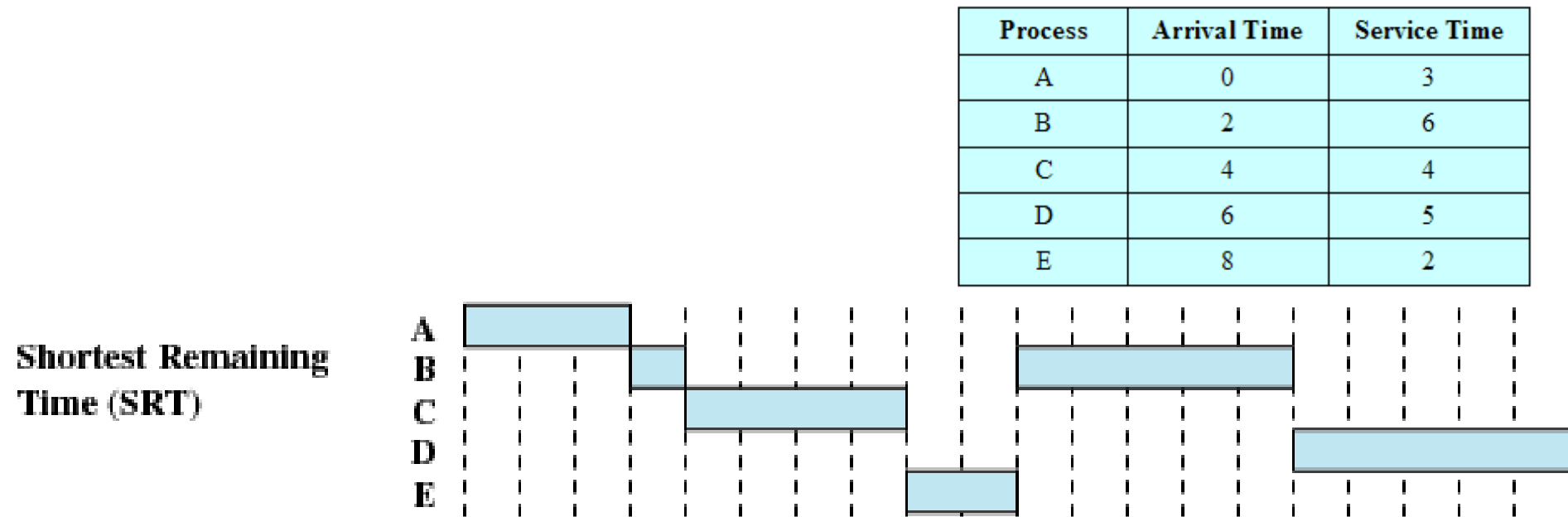
Example – SPN / SJF non-preemptive

**Shortest Process
Next (SPN)**



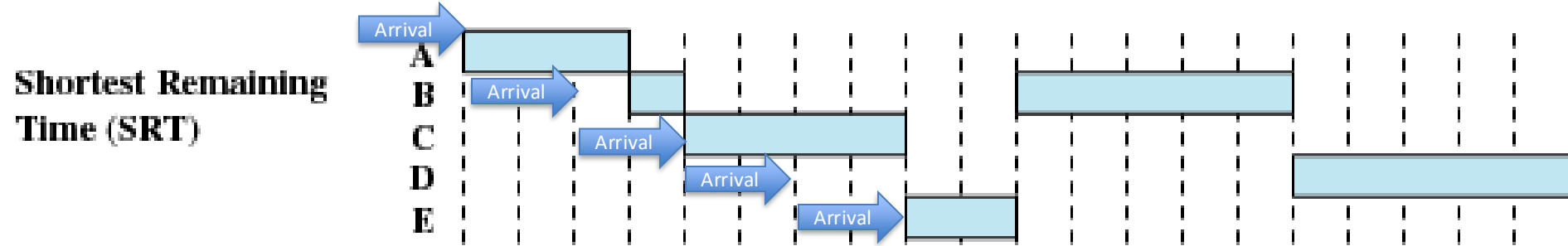
SPN	A	B	C	D	E	Avg.
Finish Time	3	9	15	20	11	-
Waiting time	0	1	7	9	1	3.60
Turnaround Time	3	7	11	14	3	7.60

Example – SRTF (or SRT) / SJF preemptive



- Preemptive version
- Must estimate processing time

Example – SRTF (or SRT) / SJF preemptive



SPN	A	B	C	D	E	Avg.
Finish Time	3	15	8	20	10	-
Waiting time	0	7	0	9	0	3.20
Turnaround Time	3	13	4	14	2	7.20

Example – Multilevel Feedback

■ Multilevel Feedback simplified

■ Two examples

■ Common procedure in the 2 examples:

- N queues
- A process enters RQ0 (Ready Queue 0)
- After its first preemption it is placed on RQ1 (next low priority queue); it repeats the same procedure for next queues
- Only when a process is preempted does it move to the next low priority queue
- Lower priority processes are only preempted by the clock, not by the arrival of a new high priority process

■ Example 1: FB(1) - parameters used:

- RR(1) in each queue

■ Example 2: FB(2ⁱ) – parameters used:

- Quantum in each queue is calculated by 2ⁱ
 - queue 0=2⁰=1 time unit;
 - queue 1=2¹=2 time units;
 - ...

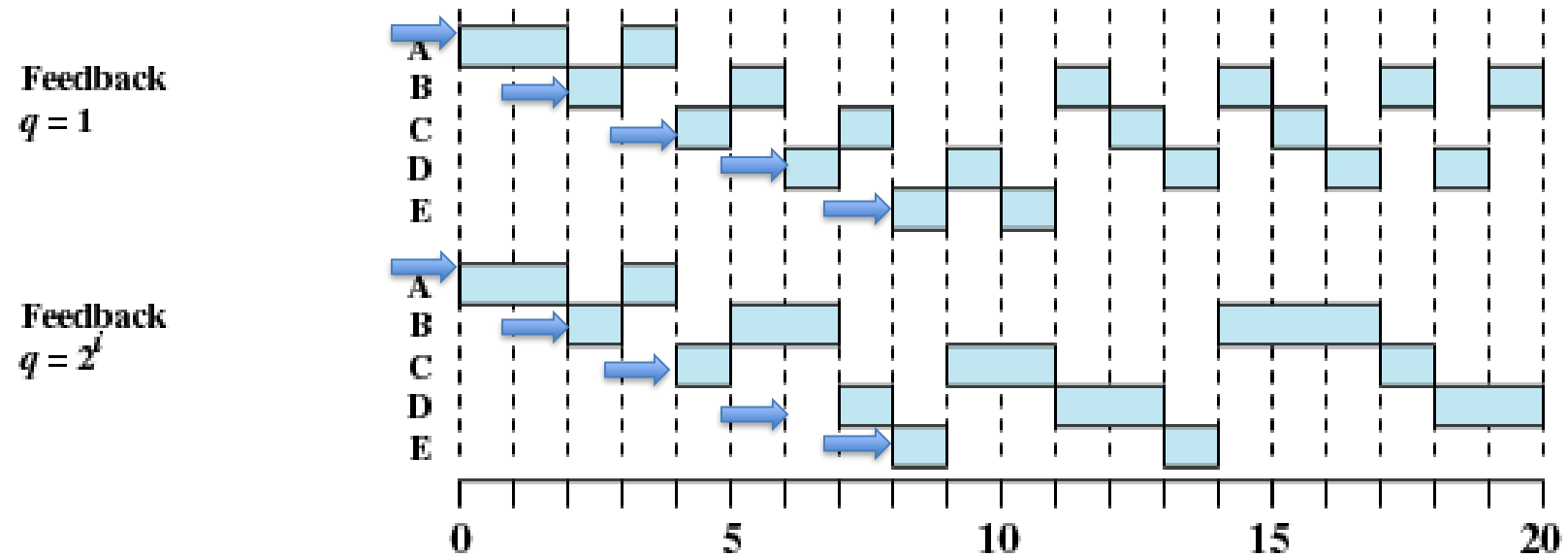
■ Penalizes jobs that are longer

■ May cause starvation if new processes keep appearing

- Solution: promote jobs to higher priority queues after some time

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

Example – Multilevel Feedback



Feedback ($q=1$)	A	B	C	D	E	Avg.
Finish Time	4	20	16	19	11	-
Waiting time	1	12	8	8	1	6.00
Turnaround Time	4	18	12	13	3	10.00

Feedback ($q=2^i$)	A	B	C	D	E	Avg.
Finish Time	4	17	18	20	14	-
Waiting time	1	9	10	8	4	6.40
Turnaround Time	4	15	14	14	6	10.00

Comparison of Scheduling Algorithms

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

First-Come-First Served (FCFS)

Round-Robin (RR), $q = 1$

Round-Robin (RR), $q = 4$

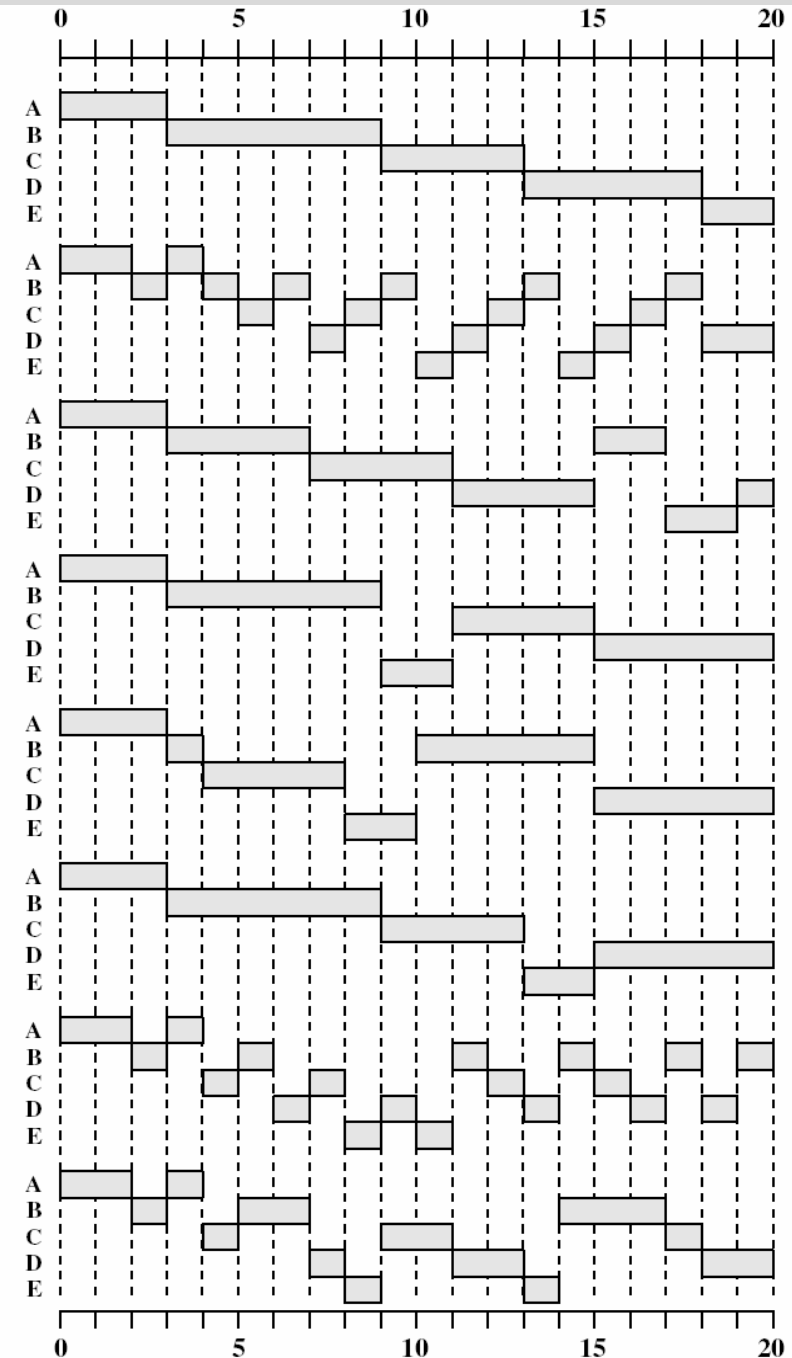
Shortest Process Next (SPN)

Shortest Remaining Time (SRT)

Highest Response Ratio Next (HRRN)

Feedback $q = 1$

Feedback $q = 2^i$



Comparison of Scheduling Algorithms (2)

	Process	A	B	C	D	E	
	Arrival Time	0	2	4	6	8	
	Service Time (T_S)	3	6	4	5	2	Mean
FCFS	Finish Time	3	9	13	18	20	
	Turnaround Time (T_T)	3	7	9	12	12	8.60
	T_T/T_S	1.00	1.17	2.25	2.40	6.00	2.56
RR $q = 1$	Finish Time	4	18	17	20	15	
	Turnaround Time (T_T)	4	16	13	14	7	10.80
	T_T/T_S	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$	Finish Time	3	17	11	20	19	
	Turnaround Time (T_T)	3	15	7	14	11	10.00
	T_T/T_S	1.00	2.5	1.75	2.80	5.50	2.71
SPN	Finish Time	3	9	15	20	11	
	Turnaround Time (T_T)	3	7	11	14	3	7.60
	T_T/T_S	1.00	1.17	2.75	2.80	1.50	1.84
SRT	Finish Time	3	15	8	20	10	
	Turnaround Time (T_T)	3	13	4	14	2	7.20
	T_T/T_S	1.00	2.17	1.00	2.80	1.00	1.59
HRRN	Finish Time	3	9	13	20	15	
	Turnaround Time (T_T)	3	7	9	14	7	8.00
	T_T/T_S	1.00	1.17	2.25	2.80	3.5	2.14
FB $q = 1$	Finish Time	4	20	16	19	11	
	Turnaround Time (T_T)	4	18	12	13	3	10.00
	T_T/T_S	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$	Finish Time	4	17	18	20	14	
	Turnaround Time (T_T)	4	15	14	14	6	10.60
	T_T/T_S	1.33	2.50	3.50	2.80	3.00	2.63

Comparison of Scheduling Algorithms (3)

■ Conclusions:

■ FCFS:

- Not fair, and average waiting time is poor, favors long processes.

■ Round Robin:

- Fair, but average waiting time is poor.

■ SPN:

- Not fair, but average waiting time is minimized assuming we can accurately predict the length of the next CPU burst. Starvation is possible. Favors short processes.

■ SRT:

- Better than SPN. Less interrupts than FCFS. Service time must be known.

■ HRRT:

- Service time must be known. While shorter processes are favored aging enables all processes to reach execution.

■ Multilevel Queuing:

- Most general scheduling algorithm, able to match specific needs. More complex.

- (We assumed that context switches took no time, which is unrealistic.)

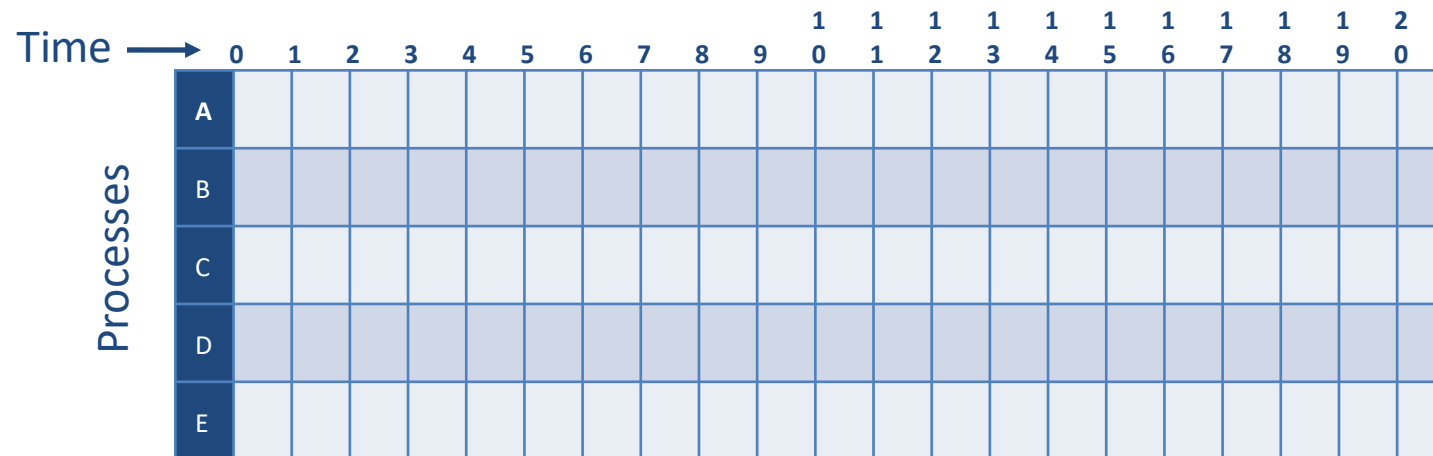
Scheduling using Timeline Charts (1)

Resolution example

- Example of RR(1) seen before:

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

- Step 1: Draw timeline chart



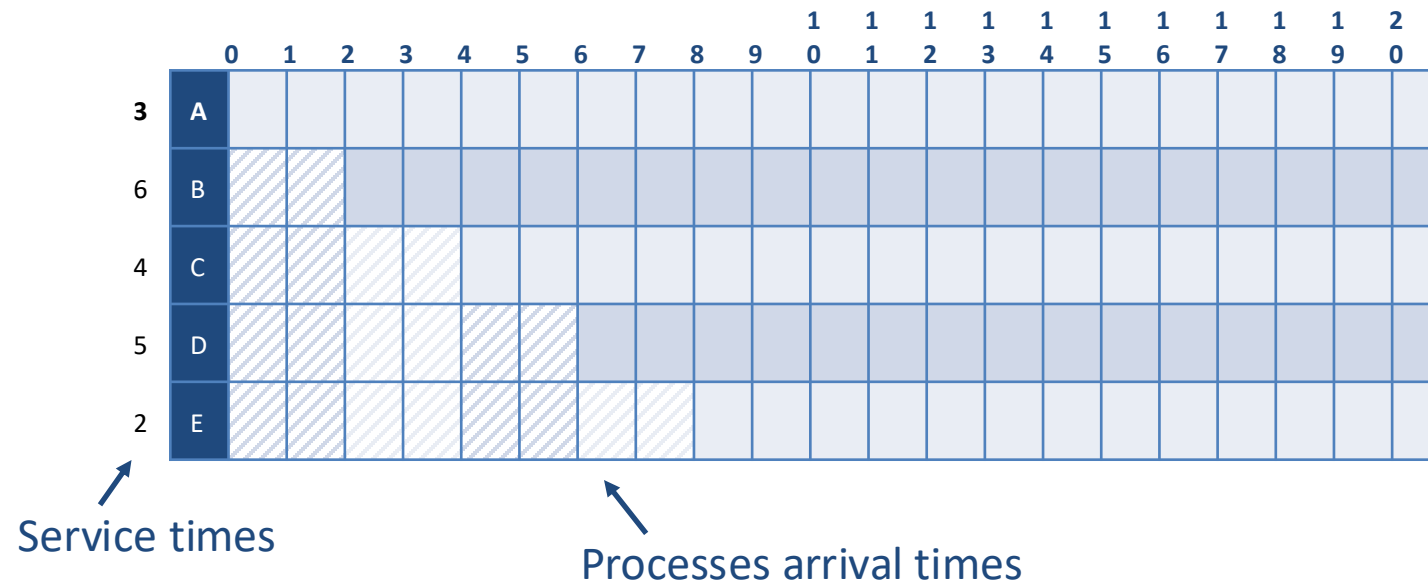
Scheduling using Timeline Charts (2)

Resolution example

- Example of RR(1) seen before:

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

- Step 2: Show arrival times and service times



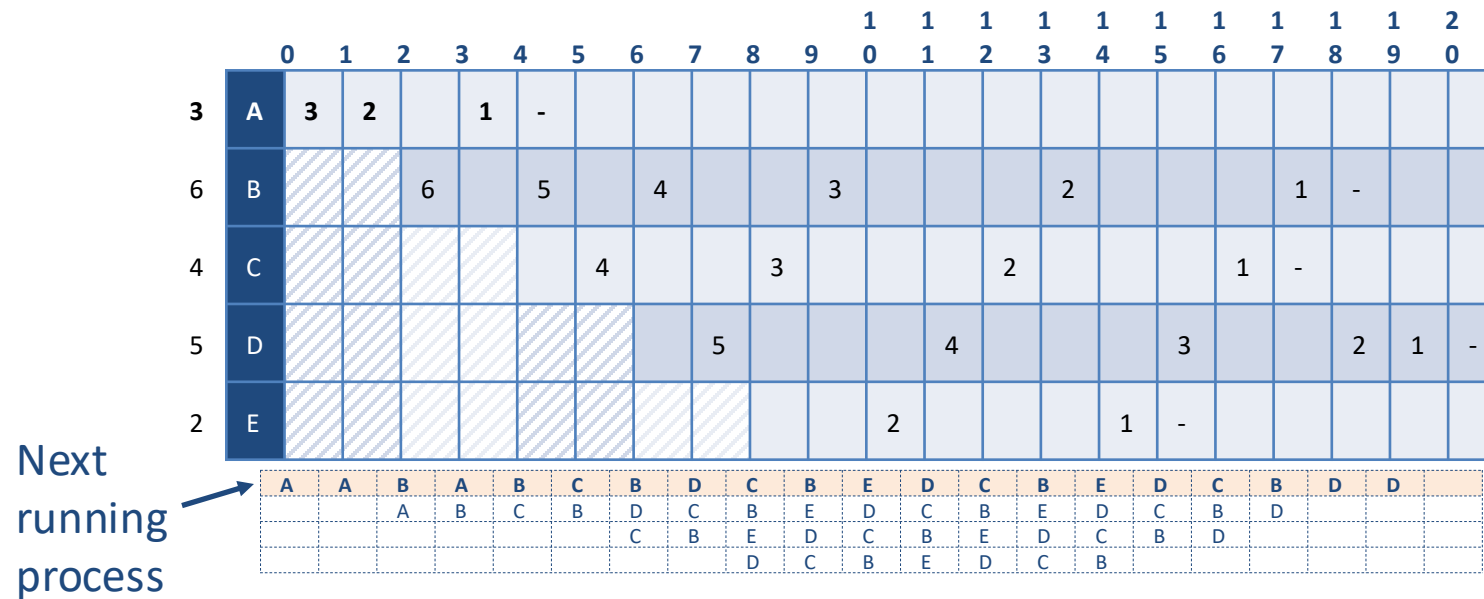
Scheduling using Timeline Charts (2)

Resolution example

- Example of RR(1) seen before:

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

- Step 3: Start scheduling, decreasing service time and depicting the queue



Scheduling

EXERCISES

Scheduling

Exercises – notes

- Notes to exercises
 - If two processes arrive at the same time, the one that first enters the ready-queue is the one that is in the higher line in the table where the processes characteristics are defined;
 - When a new process arrives and, at the same time, the process that is running leaves the processor, it is the process that was running that will go to the end of the ready-queue (i.e., the process that just arrived gets first into the ready-queue);
 - If two processes have the same characteristics, the one closer to the head of the ready-queue is chosen.

Exercise A

- Consider 5 processes (P1...P5) with the following behaviour:

Process	Arrival time	Processing time (ms) (CPU burst)	Priority
P1	0	8	4
P2	0	6	1
P3	0	1	2
P4	0	9	2
P5	0	3	3

- A.1** – Assume that the highest priority is **1** and use a Gantt to depict the scheduling of the processes using the following algorithms:
 - FCFS
 - SPN
 - Non-preemptive Priority Scheduling
 - Round-Robin (quantum=1ms)
- A.2** - For each algorithm create a table with the following values: (a) *Turnaround Time*; (b) *Waiting Time*.

Exercise B

- Consider 5 processes (P1...P5) with the following behaviour:

Process	Arrival time	Processing time (ms) (CPU burst)	Priority
P1	0	8	4
P2	2	6	1
P3	2	1	2
P4	1	9	2
P5	3	3	3

- B.1** – Assume that the highest priority is **1** and use a Gantt to depict the scheduling of the processes using the following algorithms:
 - FCFS
 - SPN
 - Non-preemptive Priority Scheduling
 - Preemptive Priority scheduling
 - Round-Robin (quantum=1ms)
- B.2** - For each algorithm create a table with the following values:
 - (a) *Turnaround Time*; (b) *Waiting Time*.

Exercise C

- Consider 5 processes (P1...P5) with the following behaviour:

Process	Arrival time (ms)	Processing time (ms)
P1	0.0	6
P2	1.5	4
P3	3.5	4
P4	4.0	1
P5	6.5	5

- C.1** - Use a Gantt to depict the scheduling of the processes using the following algorithms:
 - Round-Robin** (quantum = 3 ms)
 - SRT (Shortest-Remaining-Time-First)**
 - HRRN**
 - Multilevel Feedback** (3 queues: Q0,Q1,Q2 using algorithms RR(1ms), RR(2ms), FCFS)
 - (i) a process that arrives for a higher priority queue will preempt a process from a lower priority queue;
 - (ii) a process is demoted to a lower priority queue whenever it is preempted.
- C.2** - For each algorithm create a table with the following values: (a) *Finish Time*; (b) *Turnaround Time*; (c) *Waiting Time*.

Exercise D

- Consider 5 processes (P1...P5) with the following behaviour:

Process	Arrival time	Processing time
P1	0.0	7
P2	1.0	5
P3	3.0	2
P4	3.0	4
P5	8.0	1

- D.1** - Use a Gantt to depict the scheduling of the processes using the following algorithms:
 - Round-Robin** (quantum = 3)
 - SRT**
 - HRRN**
 - Multilevel Feedback** (3 queues: Q0,Q1,Q2 using algorithms RR(1), RR(1), FCFS)
- D.2** - For each algorithm create a table with the following values: (a) *Finish Time*; (b) *Turnaround Time*; (c) *Waiting Time*.

Exercise E

- Consider 5 processes (P1...P5) with the following behaviour:

Process	Arrival time	Processing time
P1	0.0	5
P2	1.0	3
P3	2.0	1
P4	6.0	5
P5	8.0	1

- E.1** - Use a Gantt to depict the scheduling of the processes using the following algorithms:
 - Round-Robin** (quantum = 2)
 - SPN**
 - SRT**
 - HRRN**
 - Multilevel Feedback with aging**
 - (i) 3 queues: Q0, Q1, Q2 using algorithms RR(1), RR(1), FCFS
 - (ii) Aging: when a process in queue Q1 or Q2 is inactive for 2 time units it is moved to the next high priority queue
 - (iii) Equal priority processes are scheduled based on the arrival time (older processes are executed first)
- E.2** - For each algorithm create a table with the following values: (a) *Finish Time*; (b) *Turnaround Time*; (c) *Waiting Time*.

Exercise F

6- Considere que tem 5 processos: P1..P5. A execução destes processos alterna entre **bursts** de execução (CPU) e chamadas I/O onde os processos ficam bloqueados. Na Figura seguinte estão representados os **bursts** de execução e I/O, prioridades e tempos de chegada ao sistema. Os processos têm prioridades diferentes (sendo 5 a prioridade mais alta).

	CPU	I/O	CPU	I/O	CPU	PRIORITY	ARRIVAL TIME
P1	1	1	2			1	0
P2	2	1	2	2	1	4	2
P3	2	2	2	4	1	5	3
P4	2	3	2			2	0
P5	1	1	1	3	1	3	1

Use as seguintes tabelas para fazer a simulação dos seguintes algoritmos:

6.1- **PRIORITY (Preemptive)**

6.2- **ROUND-ROBIN, with Time-Quantum=2**

6.3- **Multilevel-Feedback (Q0: RR(1); Q1: RR(2); Q2: RR(2))**

Em caso de empate directo entre dois processos (X e Y), deve respeitar as seguintes regras:

- (a) se o processo X está a executar deixe-o executar;
- (b) se nenhum dos processos (X,Y) está a executar então escolha o processo com o menor ID.

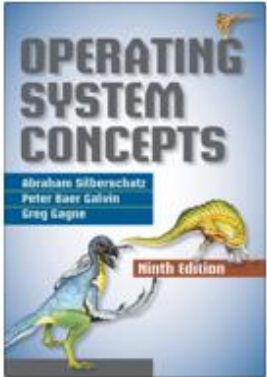
Exercise G

- Consider 5 processes (P1...P5) with the following behaviour:

Process	Arrival time (ms)	Processing time (ms)
P1	0	6
P2	3	4
P3	7	4
P4	8	1
P5	13	5

- **G.1** - Use a Gantt to depict the scheduling of the processes using the following algorithms:
 - Round-Robin (quantum = 3 ms)
 - SRT (Shortest-Remaining-Time-First)
 - HRRN
 - Multilevel Feedback
 - (i) 3 queues: Q0, Q1, Q2 using algorithms RR(1 ms), RR(2 ms), FCFS;
 - (ii) a process that arrives for a higher priority queue will preempt a process from a lower priority queue;
 - (iii) a process is demoted to a lower priority queue whenever it is preempted.
- **G.2** - For each algorithm create a table with the following values: (a) *Finish Time*; (b) *Turnaround Time*; (c) *Waiting Time*.

References



- [Silberschatz13]
 - Chapter 6 – CPU Scheduling

Thank you! Questions?



I keep six honest serving men. They taught me all I knew. Their names are What and Why and When and How and Where and Who.
—Rudyard Kipling