

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

```
#define MAXPAROLA 30
#define MAXRIGA 80
```

```
int main(int argc, char *argv[])
```

```
{
    int freq[MAXPAROLA]; /* vettore di contatori
    delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;
```

```
for(i=0; i<MAXPAROLA; i++)
    freq[i]=0;
```

```
if(argc != 2)
{
    fprintf(stderr, "ERRORE, serve un parametro con il nome del file\n");
    exit(1);
}
```

```
f = fopen(argv[1], "r");
if(f == NULL)
{
    fprintf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
    exit(1);
}
```

```
while( fgets( riga, MAXRIGA, f ) != NULL )
```



The CPU Scheduling

CPU Scheduling

Stefano Quer and Stefano Scanzio

Dipartimento di Automatica e Informatica

Politecnico di Torino

skenz.it/os

stefano.scanzio@polito.it

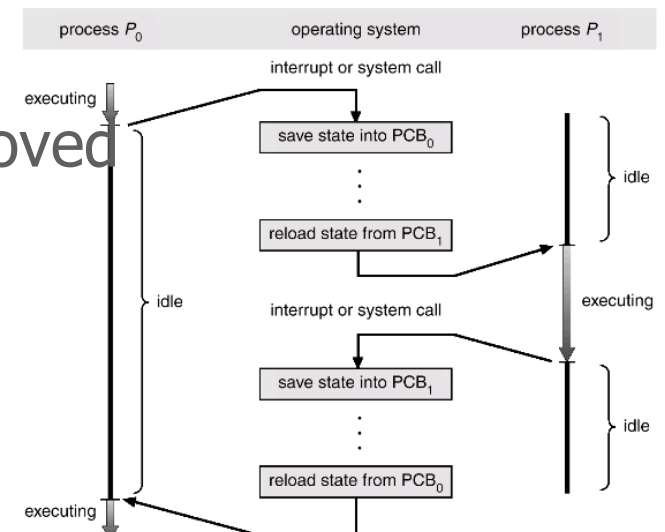
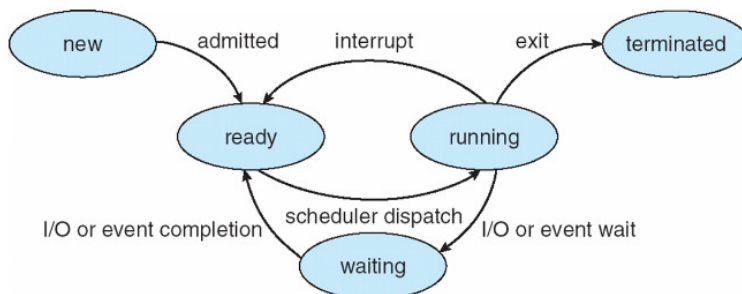
Fundamental concepts

- ❖ One of the main targets of multiprogramming is to maximize the use of the CPU resource
- ❖ To reach this target, more than one task (i.e., process or thread) is assigned to each
 - The scheduler must implement the better scheduling **algorithm** for the assign of the CPU to a task
 - Scheduler performance is evaluated through **cost functions**
 - Different applications require different algorithms and cost functions

From section
u04s02

Algorithm

- ❖ General scheduling procedure
 - CPU is assigned to a task
 - Each time a process enters a waiting state, terminates, an interrupt is received, etc., it is necessary to perform a context switching operation
 - For each context switching
 - The task in the running state is moved in the ready queue
 - A task in the ready queue is moved in the running state



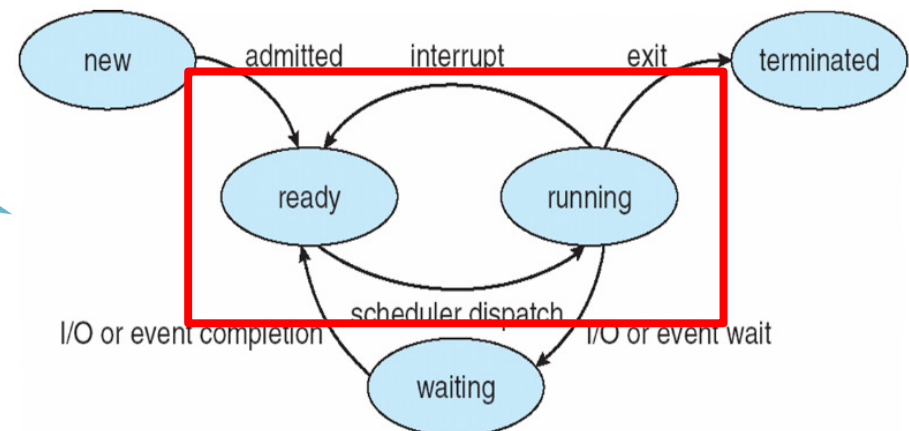
CPU scheduler

❖ Several types of scheduler exist

➤ Short-term scheduler

- Selects the process to which assign the CPU within the set of processes ready for execution in the main memory
- Is executed very frequently
- Is executed with timings in the order of milliseconds after an interrupt due to a timer or an I/O operation
- Must be very fast

It schedules processes in RAM

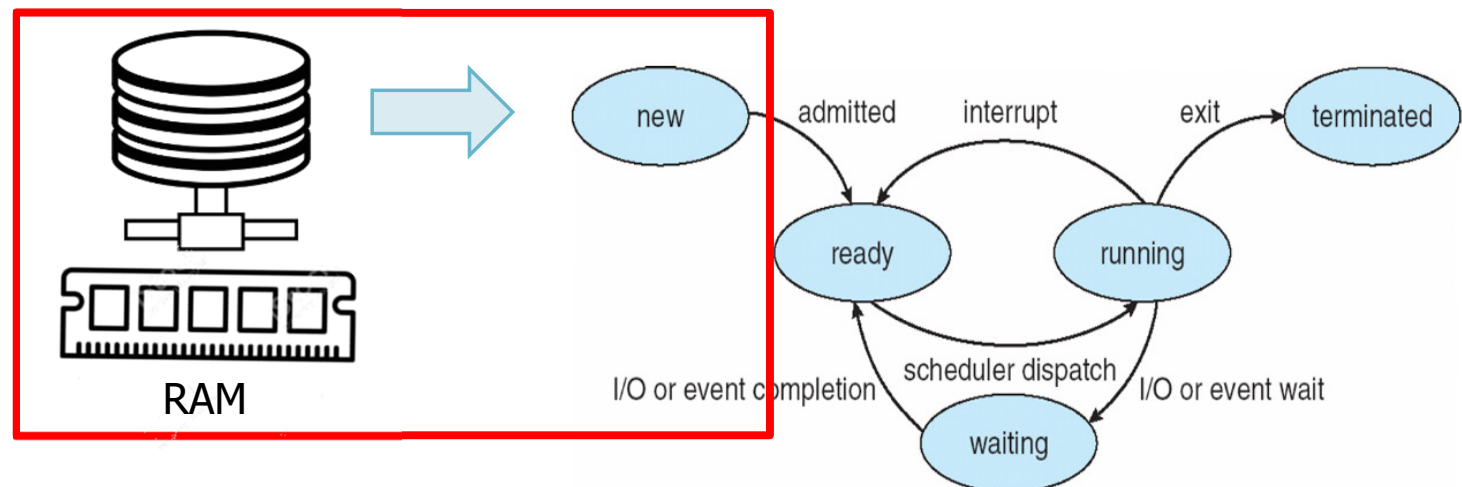


CPU scheduler

➤ Medium-term scheduler

- Moves processes from the main memory to the second memory and vice versa
 - Selects which process to insert in the ready list
- Is executed less frequently
 - Is executed with timings in the order of seconds
- In practice, it controls the number of processes in RAM

It schedules processes in disk

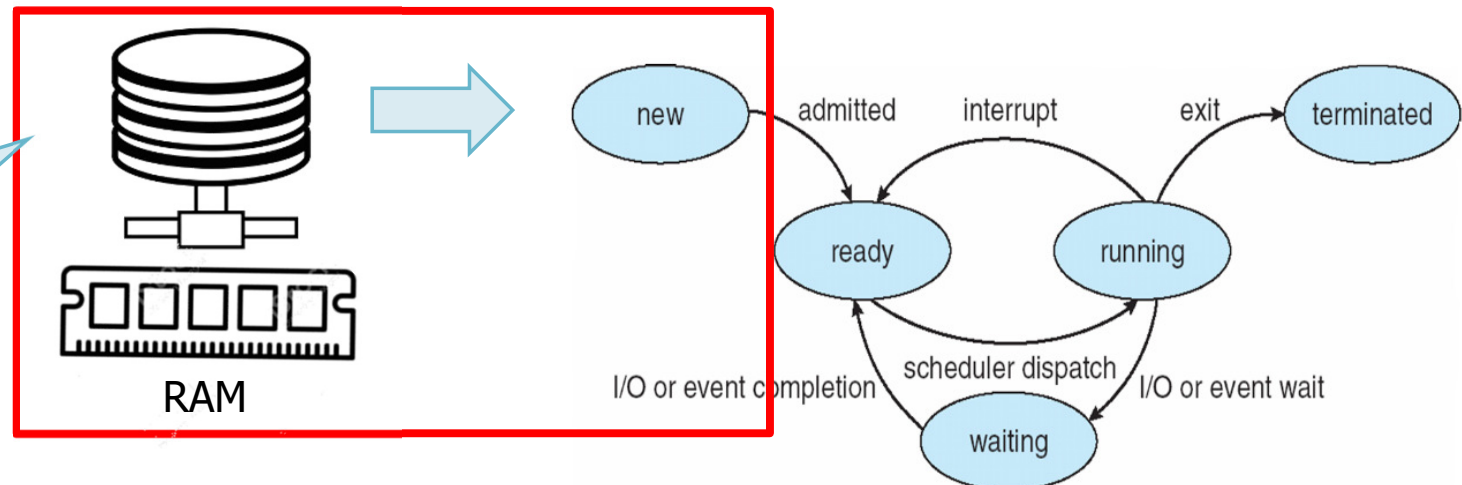


CPU scheduler

➤ Long-term scheduler

- Also called job scheduler or admission scheduler selects the programs the system wishes to process based on its resources
- Is executed much less frequently
 - Is executed with timings in the order of minutes
- It basically controls the degree of multiprogramming

It selects processes on disk for the medium-term scheduler



CPU scheduler

Static analysis of processes

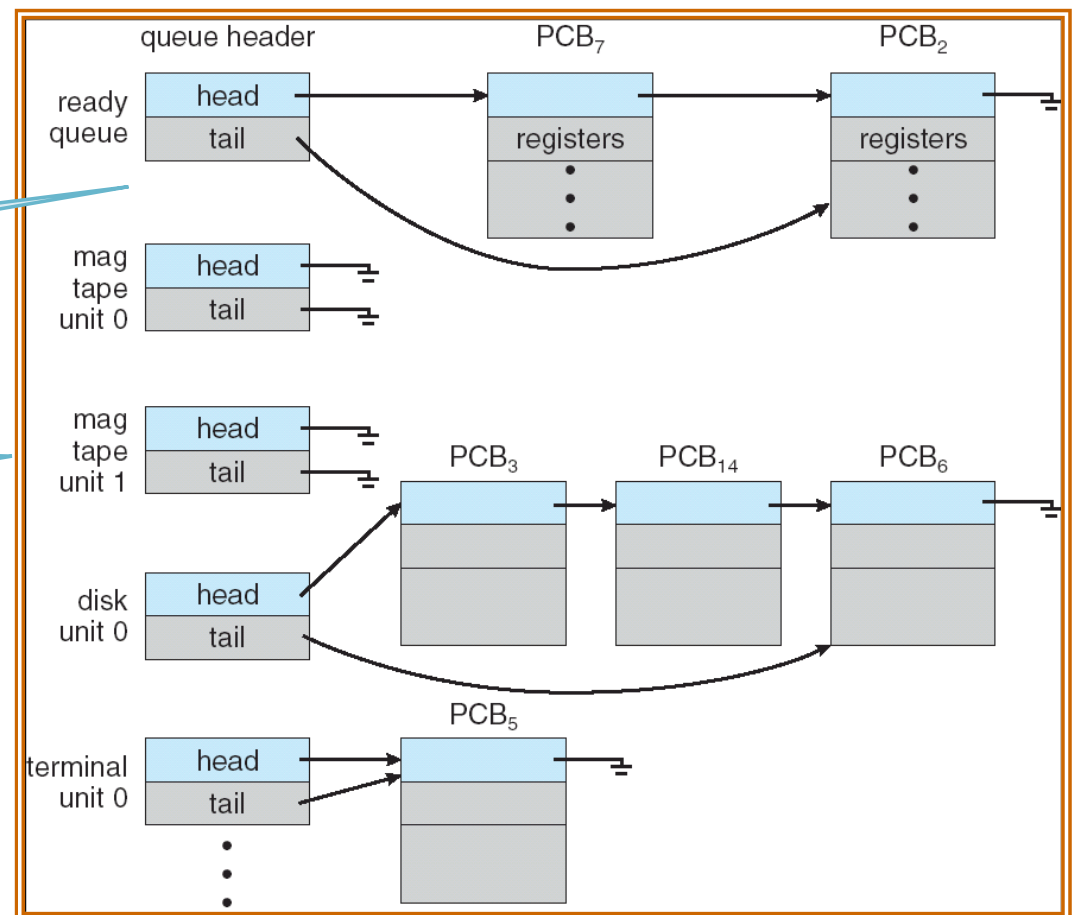
❖ The scheduler manages the processes waiting for a device through (**processes**) **queue**

- There are several queues, one per device
- Each queue is a linked list

Queue of ready processes

Queue of processes waiting I/O

To maximize efficiency, each device has its own tail



Queuing diagram

Dynamic analysis
of processes

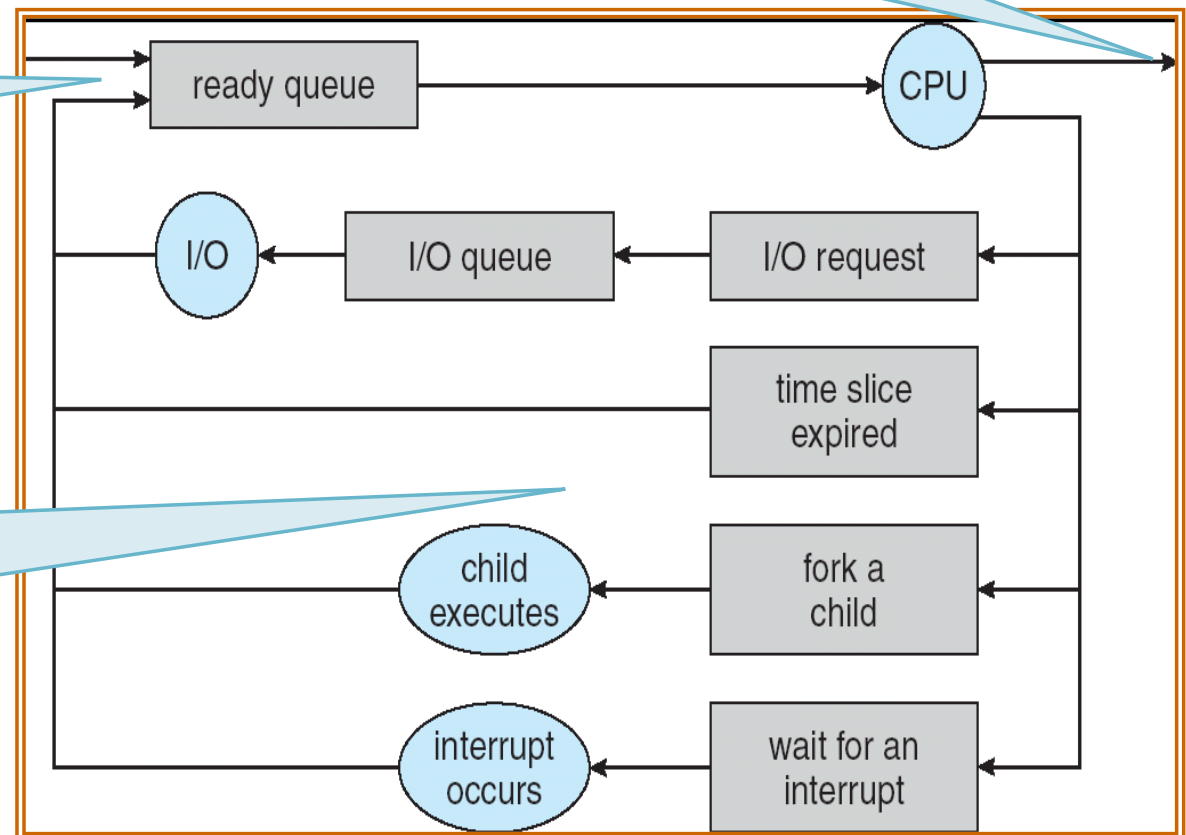
❖ The queuing diagram shows the possible process transitions from one queue to another one

➤ Each rectangle represents
a queue

Process initially goes
on the ready queue

A running process
releases the CPU and
goes on the ready state
due to (I/O completion,
interrupt, fork, etc.)

Process terminates



Algorithms

Extension

Algorithms without preemption

FCFS (First Come First Served)
Scheduling in order of arrival

SJF (Shortest Job First)
Scheduling in order of length

PS (Priority Scheduling)
Scheduling in order of priority

MQS (Multilevel Queue Scheduling)

Algorithms with preemption

RR (Round Robin)
Circular scheduling

SRTF (Shortest Remaining Time
First)
Scheduling for minimum remaining
time

Multi-level queues scheduling

Non preemptive

The CPU is **not** subtracted to another task, i.e., the task must release the CPU voluntarily

Preemptive

The CPU can be subtracted to another task, i.e., CPU burst are defined (e.g., maximum execution times) at the end of which the CPU is reassigned to another task

Cost functions

Cost function	Description	Optimum
CPU utilization	Percentage of CPU utilization	[0-100%] Maximum
Throughput	Number of processes completed in a time unit	Maximum
Turnaround time	Time that passes from the submission to the termination of a process	Minimum
Waiting time	Total time spent in the ready queue (sum of the times spent in the queue)	Minimum
Response time	Time elapsed between the submission and the production of the first response	Minimum

FCFS (First Come First Served)


❖ Algorithm

- The CPU is assigned to the tasks following the order in which they requested it
 - Tasks are managed through a FIFO queue
 - A new task is inserted in the queue tail
 - A task to serve is extracted from the queue head
- Scheduling can be sketched by means of a **Gantt diagram** (1917)
 - Bar chart showing the planning (start and end times) of the activities

Remember: No task is interrupted, i.e., the CPU can **only** be released voluntarily

FCFS (First Come First Served)

Example 1



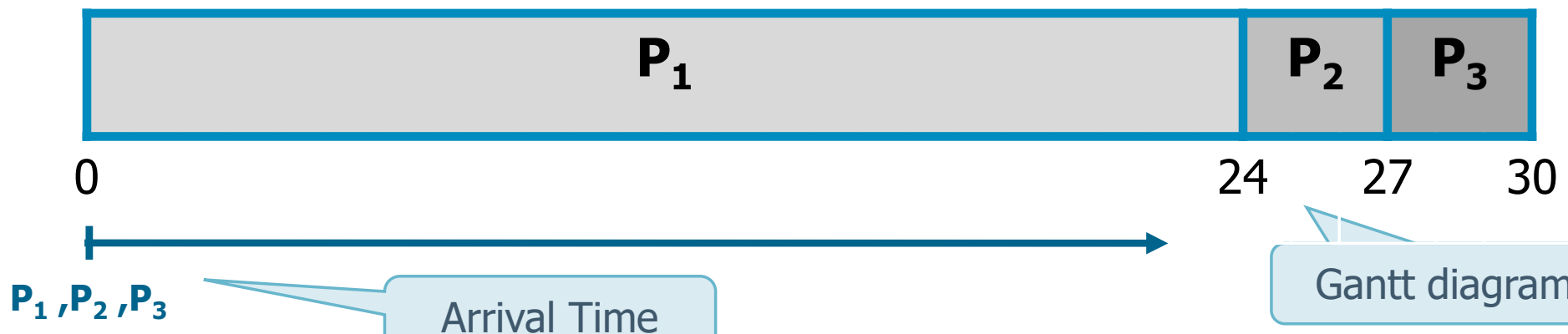
P	Arrival Time	Burst Time
P ₁	0	24
P ₂	0	3
P ₃	0	3

Task arrival order

Expected duration
(unit of time)


P	Waiting Time
P ₁	$(0-0) = 0$
P ₂	$(24-0) = 24$
P ₃	$(27-0) = 27$

Average waiting time:
 $(0+24+27)/3=17$



FCFS (First Come First Served)

Example 2



P	Arrival Time	Burst Time
P ₂	0	3
P ₃	0	3
P ₁	0	24

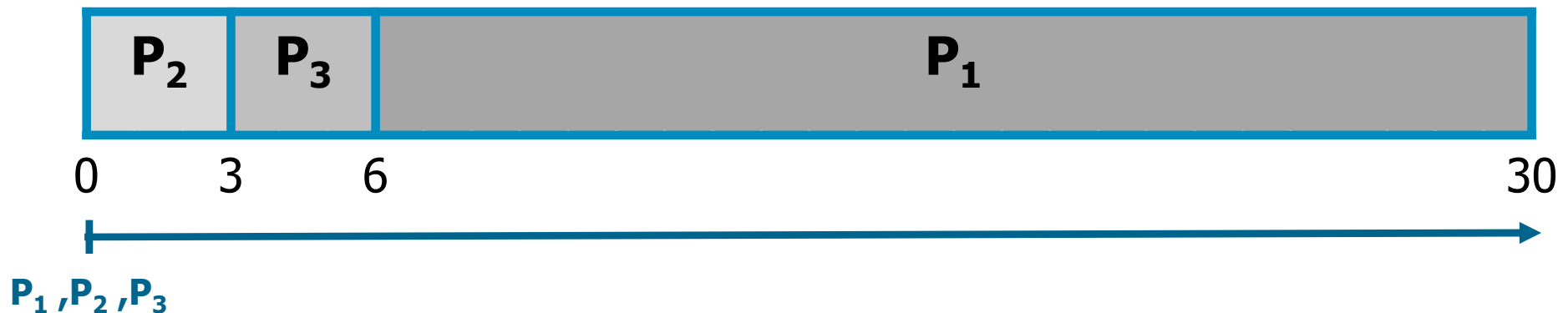
Task arrival order

Expected duration (unit of time)

P	Waiting Time
P ₁	$(6-0)=6$
P ₂	$(0-0)=0$
P ₃	$(3-0)=3$

Average waiting time:
 $(6+0+3)/3=3$

Much **better** than the previous one:
long processes delay short ones



FCFS (First Come First Served)

❖ Advantages

- Easy to understand
- Easy to implement

❖ Disadvantages


- Waiting times
 - Relatively long
 - Variables and not optimal
- Unsuitable for real-time systems (no preemption)
- Queue effect
 - Short tasks queued after long tasks, wait for a long time uselessly

SJF (Shortest-Job-First)

❖ Algorithm

- To each task is associated the duration of the next CPU request (**next CPU burst**)
- The tasks are scheduled in order of duration of their next request
 - Scheduling in order of length
 - In case of ex-aequo (i.e., tasks with the same length) the FCFS scheduling is applied

SJF (Shortest-Job-First)

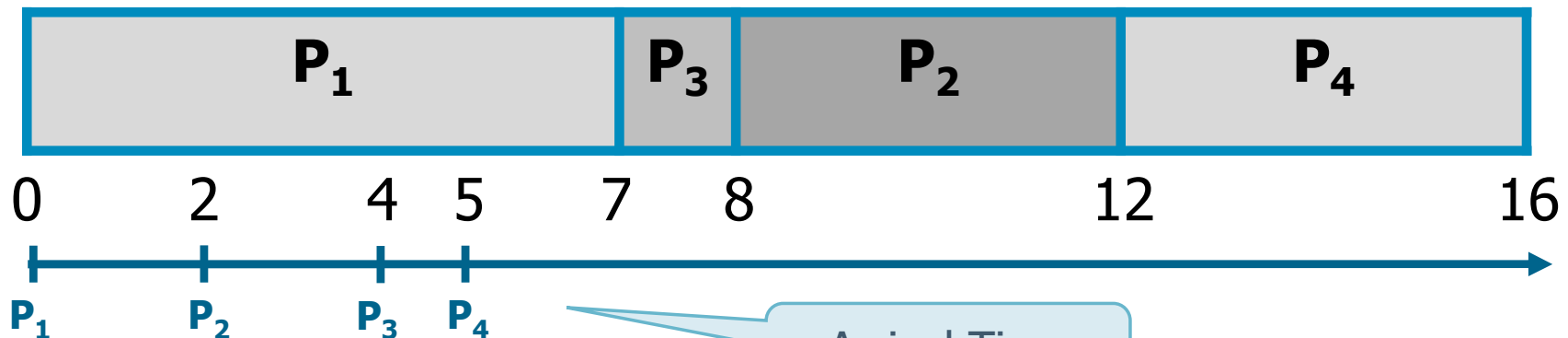


P	Arrival Time	Burst Time
P ₁	0	7
P ₂	2	4
P ₃	4	1
P ₄	5	4

Task arrival order

Expected duration

P	Waiting Time
P ₁	$(0-0) = 0$
P ₂	$(8-2) = 6$
P ₃	$(7-4) = 3$
P ₄	$(12-5) = 7$
Average waiting time: $(0+6+3+7)/4=4$	



Arrival Time

SJF (Shortest-Job-First)

❖ Advantages

- It can be demonstrated that SJF is an optimal algorithm, using the waiting time as a criterion
 - By moving the short processes before the long ones, the waiting time of the first decreases more than the increase of the waiting time of the seconds

❖ Disadvantages

- Possible starvation
- Difficult of application, due to the impossibility to know a priori the future behavior of the task
 - Next burst time is **unknown**
 - It is possible to **estimate** this time using different methods (e.g., the exponential average)

SJF (Shortest-Job-First)

❖ Exponential average

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

Expected value for the next burst

(Real) duration of the n-th burst

Estimated n-th burst

$\alpha = [0, 1]$
control the relative weight recent vs. past history

$$\alpha = 0 \rightarrow \tau_{n+1} = \tau_n$$

$$\alpha = 1 \rightarrow \tau_{n+1} = t_n$$

❖ Proceeding by substitution

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

- Since both α and $1 - \alpha$ are minor than 1, older terms weight less

PS (Priority Scheduling)

❖ Algorithm


➤ A priority associated to each task

- Priority is typically represented with integer number
- The higher the priority the smaller the integer number
- Priorities can be determined based on
 - Internal criteria: used memory, number of used files, etc.
 - External criteria: owner of the task, etc.

➤ CPU is allocated to the task with higher priority

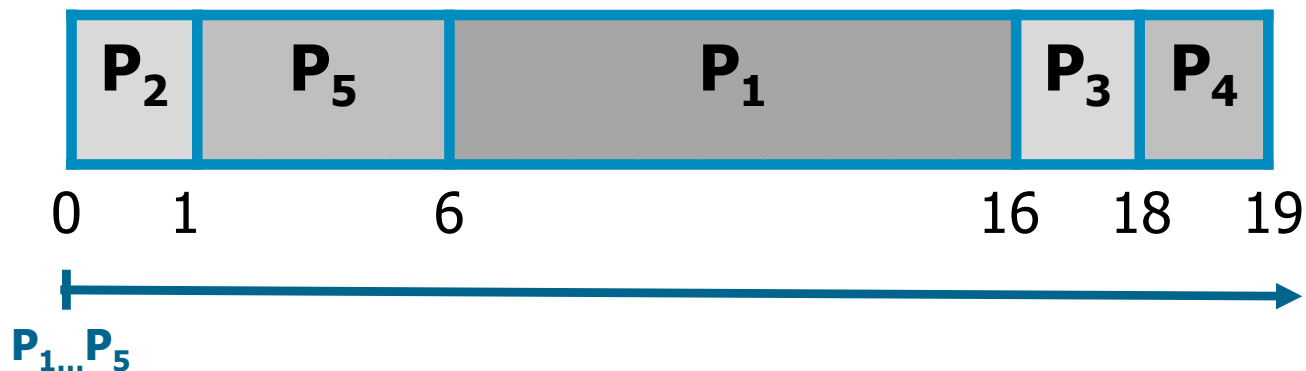
- PS = SJF with the duration of the CPU burst substituted with the priority

PS (Priority Scheduling)



P	Arrival Time	Priority	Burst Time
P ₁	0	3	10
P ₂	0	1	1
P ₃	0	4	2
P ₄	0	5	1
P ₅	0	2	5

P	Waiting Time
P ₁	$(6-0) = 6$
P ₂	$(0-0) = 0$
P ₃	$(16-0) = 16$
P ₄	$(18-0) = 18$
P ₅	$(1-0) = 1$
Average waiting time: $(6+0+16+18+1)/5=8.2$	



PS (Priority Scheduling)

❖ Drawbacks

➤ Possible starvation

- In highly loaded systems, tasks with low priority can wait forever
 - MIT: IBM stopped in 1973 with a process queued since 1967
- A possible solution to starvation is aging of tasks
 - The tasks priority is gradually increase over time

RR (Round Robin)

- ❖ Round Robin or circular scheduling
- ❖ Version of FCFS with **preemption**
- ❖ Algorithm
 - The CPU usage is divided into "time quantum" (i.e., discrete temporal intervals)
 - Each task can use the CPU for a maximum time equal to the quantum, and then it is inserted again in the ready queue
 - The ready queue is managed using a FIFO policy
 - New processes are inserted in the ready queue
- ❖ Designed specifically for time sharing (and some basic real-time systems)

RR (Round Robin)

P	Arrival Time	Burst Time
P ₁	0	53
P ₂	0	17
P ₃	0	68
P ₄	0	24

Quantum:
20 units

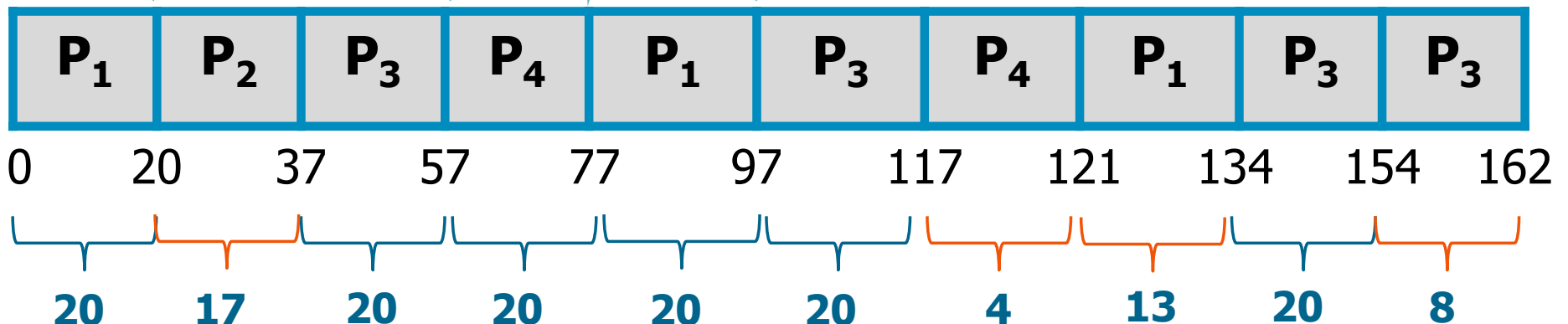
Remaining:
P₄:4

Remaining:
P₁:33

Remaining:
P₃:48

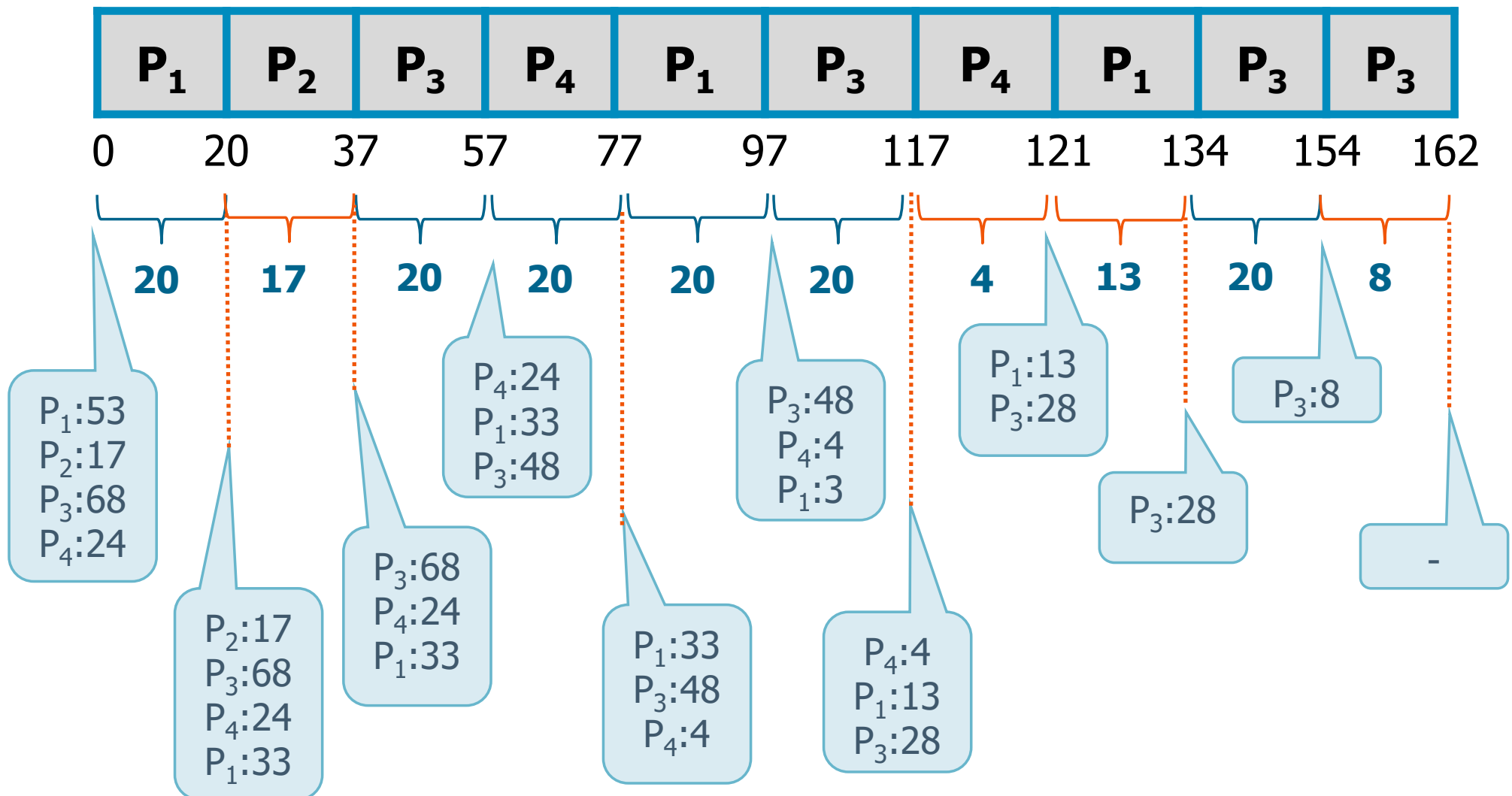
Remaining:
P₁:13

P	Waiting time
P ₁	$(0-0)+(77-20)+(121-97)=81$
P ₂	$(20-0)=20$
P ₃	$(37-0)+(97-57)+(134-117)+(154-154)=94$
P ₄	$(57-0)+(117-77)=97$
Average waiting time: $(81+20+94+97)/4=73.00$	



RR (Round Robin)

❖ More in details



RR (Round Robin)


❖ Drawbacks

- The average waiting time is relatively long
- Substantial dependence of performance on the length of the quantum
 - Quantum long: RR degenerates into FCFS
 - Quantum short: too much context switching is performed, and switching/management times are very high (if compared with useful work)

SRTF (Shortest-Remaining-Time-First)

- ❖ Version of SJF with **preemption**
- ❖ Algorithm
 - It proceeds with a scheduling of type SJF, but
 - if a task with smaller burst time (than the running one) is submitted, the CPU is preempted in favor of the new task
- ❖ Similar characteristics of the SJF scheduler

SRTF (Shortest-Remaining-Time-First)



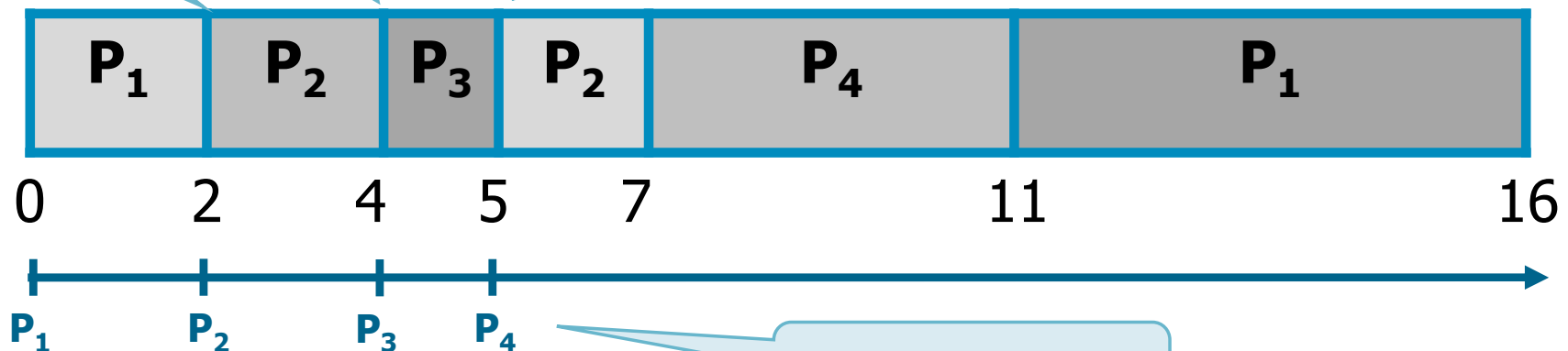
P	Arrival Time	Burst Time
P ₁	0	7
P ₂	2	4
P ₃	4	1
P ₄	5	4

P	Waiting time
P ₁	$(0-0)+(11-2) = 9$
P ₂	$(2-2)+(5-4) = 1$
P ₃	$(4-4) = 0$
P ₄	$(7-5) = 2$
Average waiting time: $(9+1+0+2)/4=3$	

Remaining:
P₁:5; P₂:4

Remaining:
P₁:5; P₂:2; P₃:1

Remaining:
P₁:5; P₂:2; P₄:4



Arrival Time

SRTF (Shortest-Remaining-Time First)

❖ Advantages

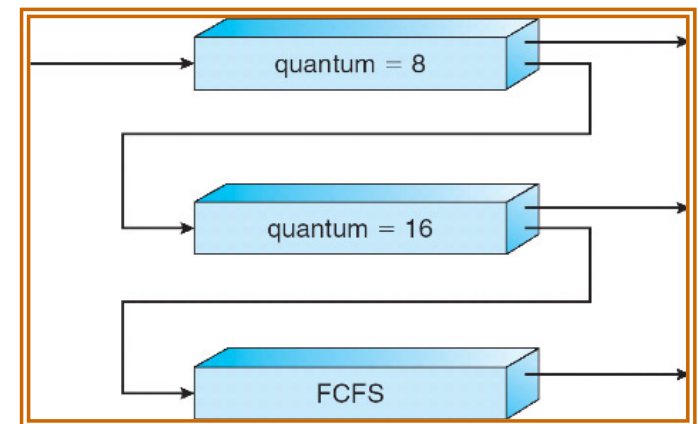
- Short tasks are handled very quickly
- Because the task with the least remaining time is executed and its remaining time can only decrease, context switching occurs only when a new processes arrive
- The overhead required by the algorithm is minimal

❖ Drawbacks

- Like SJF, it requires an accurate estimates of the execution time
- Like SJF suffers of starvation

MQS (Multilevel Queue Scheduling)

- ❖ Applied to situations where tasks can be classified into different groups
 - Foreground, background, system, etc.
- ❖ Algorithm
 - The ready queue is divided into different queues
 - Each queue can be managed with its own scheduling algorithm
 - It can be modified to allow the transfer of tasks between the various queues
 - MQS with feedback



Considerazioni aggiuntive

- ❖ Lo scheduler è un task che deve essere schedulato in maniera simile agli altri task
 - Nello scheduling senza prelazione
 - Lo scheduler è invocato ogni volta che un programma termina o lascia il controllo
 - Nello scheduling con prelazione
 - Lo scheduler è invocato periodicamente da un interrupt periodico della CPU
 - Gli altri task non possono prevenire questo procedimento

Additional considerations

- ❖ Scheduling can be performed at the process or thread level
 - If the OS allows the use of threads, the scheduling is normally performed at the threads level (processes are not taken into account)
- ❖ Threads scheduling
 - The SO takes into account only T at kernel level, and it ignores T at user level (which are managed through a library)
 - As a consequence, the scheduling can be performed only for T at kernel level (if they exist)

Additional considerations

❖ Scheduling for multiprocessors systems

- All previous examples have been made assuming the existence of a single CPU
- In the case of more than one CPU, load can be shared
 - The **load balance** is automatic for OS with waiting queues common to all processors
- There are several schemes
 - **Asymmetric** multi-processing: a master processor distribute the load among slave processors
 - **Symmetric** multi-processing: each processor provides for its own scheduling

Additional considerations


❖ Scheduling for real-time systems

- They try to respond in real-time and within predefined deadline to events
 - Events (e.g., raise of a signal and subsequent interrupt) guide the scheduling
 - **Latency** is defined as the time elapsing between the occurrence of an event and its management
- There are two types of real-time systems
 - Soft real-time
 - They give priority to critical processes, but do not guarantee response times (only probabilistic guarantees)
 - Hard real-time
 - The execution of the tasks is guaranteed within a maximum time limit (deadline)

Exam Italian Course: 2017/02/17

Exercise

❖ Considering the following set of processes



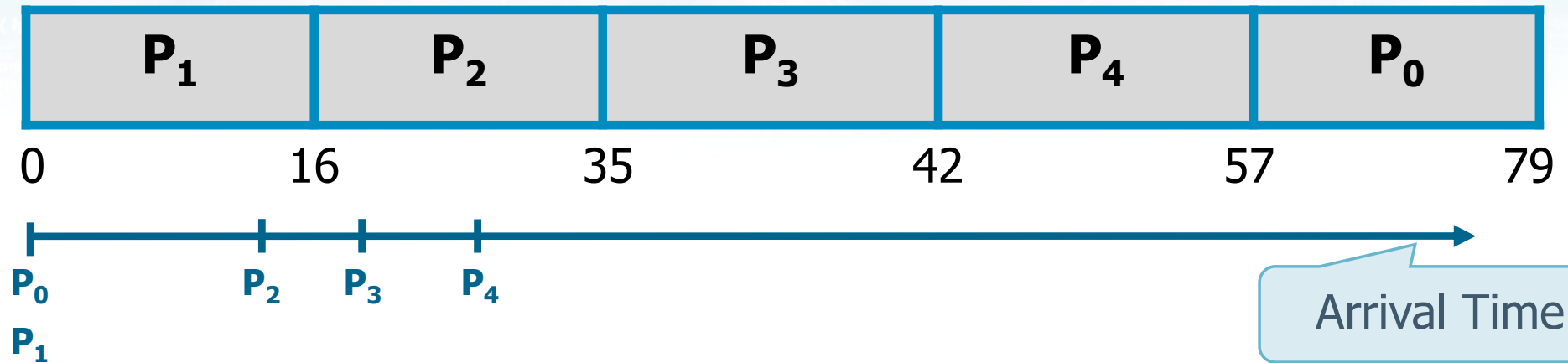
P	Arrival Time	Burst Time	Priority
P ₀	0	22	5
P ₁	0	16	2
P ₂	15	19	4
P ₃	17	7	1
P ₄	25	15	1

- Draw the Gantt diagram for the PS (Priority Scheduling), RR (Round Robin), and SRTF (Shortest Remaining Time First) algorithms
- Compute the average waiting time

Arrival order of the tasks

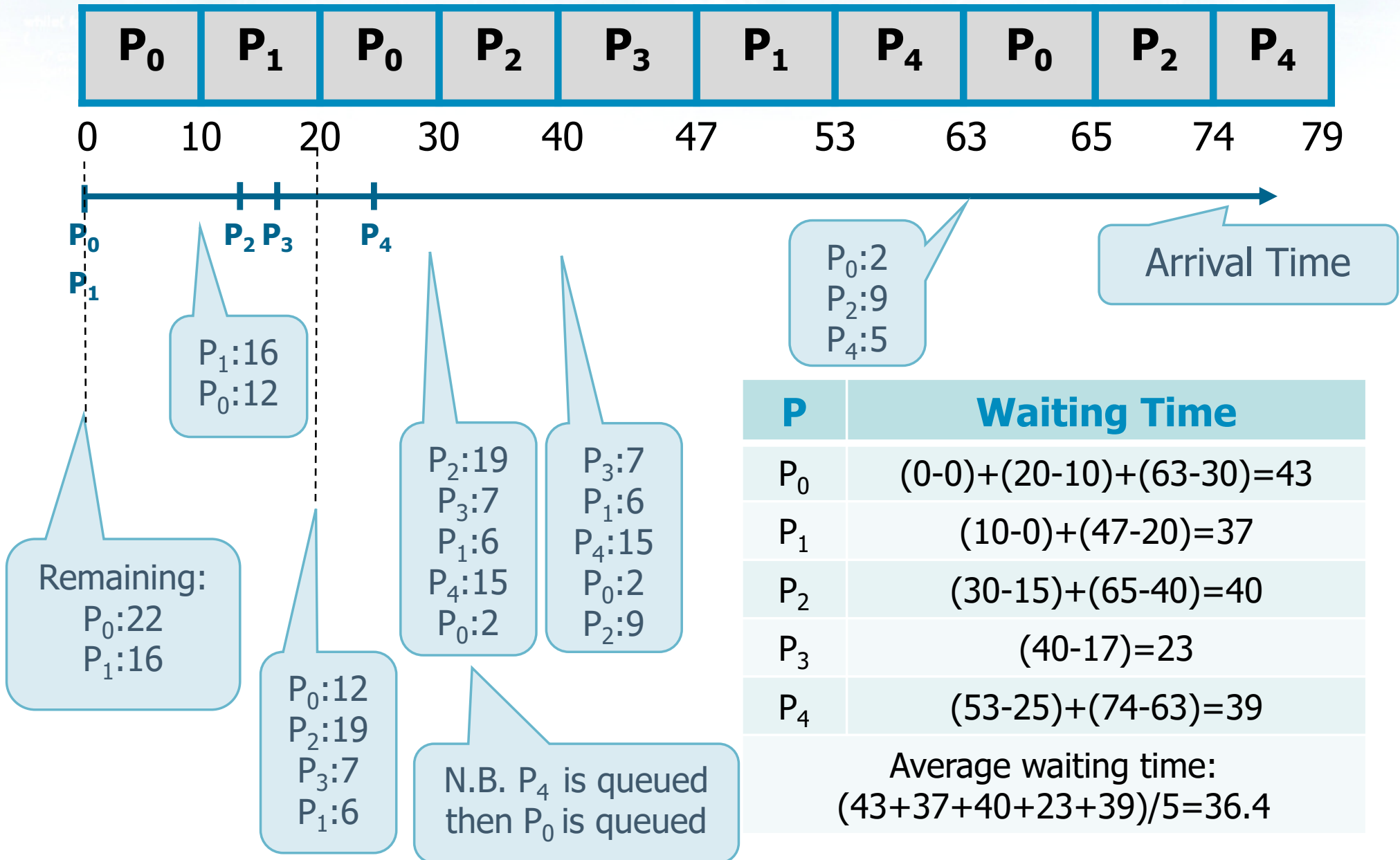
Maximum priority= smaller value
Temporal Quantum = 10

Exercise: PS



P	Waiting time
P_0	$57-0=57$
P_1	$0-0=0$
P_2	$16-15=1$
P_3	$35-17=18$
P_4	$42-25=17$
Average waiting time: $(57+0+1+18+17)/5=18.6$	

Exercise: RR



Exercise: SRTF

