

# Operating System Process and Resource Management

*Scheduling*



# *Course Objectives*

- ✓ Understand the objectives and the need for process scheduling
- ✓ Discover the different scheduling mechanisms
- ✓ Learn how to evaluate scheduling algorithms



# *Course Plan*

1. Definition
2. Mechanism
3. Planning
4. Algorithm Rating



# 1. Definition



# 1. Definition

## Problem

- When should we stop a process running on a processor (CPU)?
- Which process will run on the processor?
- Which thread will run on the processor?
- What happens if we have several processors for one computer?
- What happens when we have several computers and several processes using the same resources?

# 1. Definition

## Goals

- The first goal of an operating system is to maximize the utilization rate of the processor(s): to define the average number of processes executed in a certain time
- The second goal is to maximize the useful rate of the processor(s): to define the occupancy rate or the proportion of time used to execute user processes
- The operating system uses the scheduler located in the kernel
- Some operating systems use **load balancing** to run processes on multiple processors
  - This is an additional handler (other mechanisms exist for load balancing)

# 1. Definition

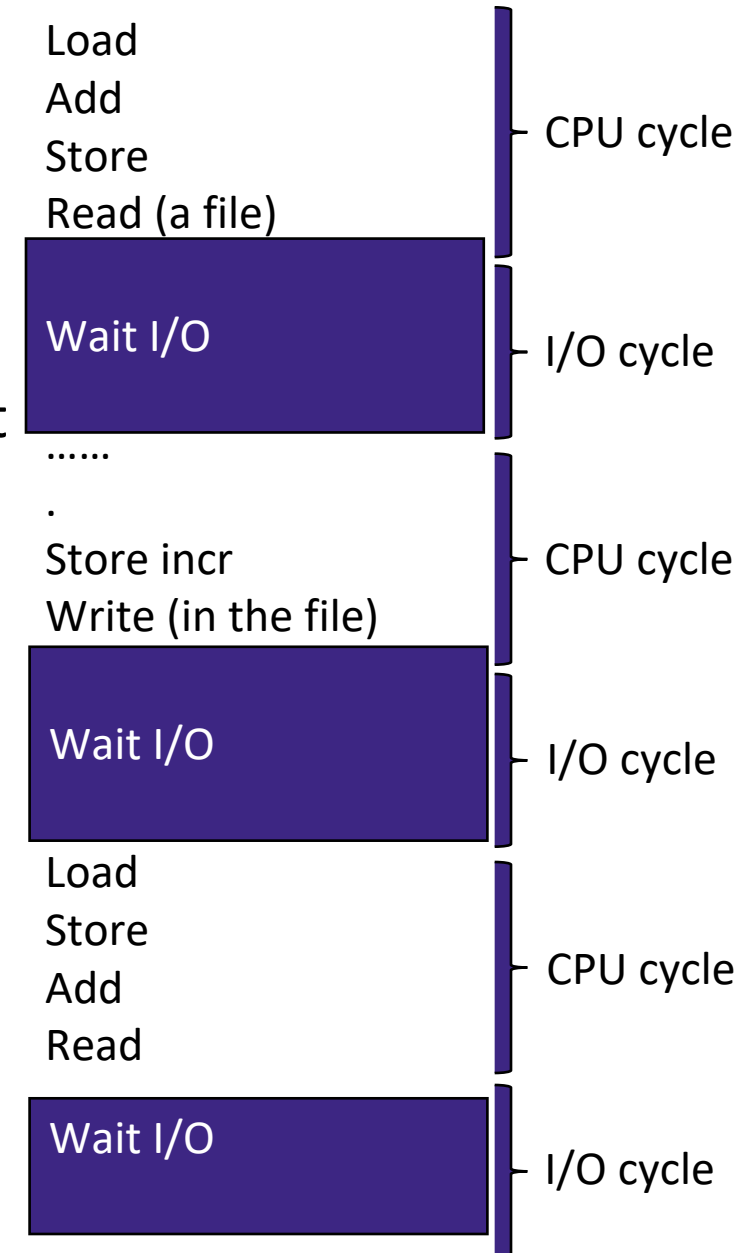
## Goals

- The scheduling mechanism must minimize:
  - **Waiting time** (amount of time a process waits in a ready state/queue)
  - **Response time** (time elapsed between the request submitting and the first response production, not necessarily the result); shared time environment
  - **Processing time** or **turn around time** (time required to execute a given process)
  - **Starvation mechanism** (a process is in a starvation situation when it is denied access to a resource for an indefinite time)
- A parallel program is a finite set of processes, each process executes one or more actions (tasks); the creation of a parallel program involves load balancing (the program must be broken down into several small actions so that they can be assigned to the different processors while respecting the integrity of the program)

# 1. Definition

## Process

- Processes constantly switch between input/output sequences and computation sequences on the CPU (alternating bursts or CPU/IO bursts)
- Processes that consume long periods of CPU time are uncommon





# 1. Definition

## Questions



## 2. Mechanism



## 2. Mechanism

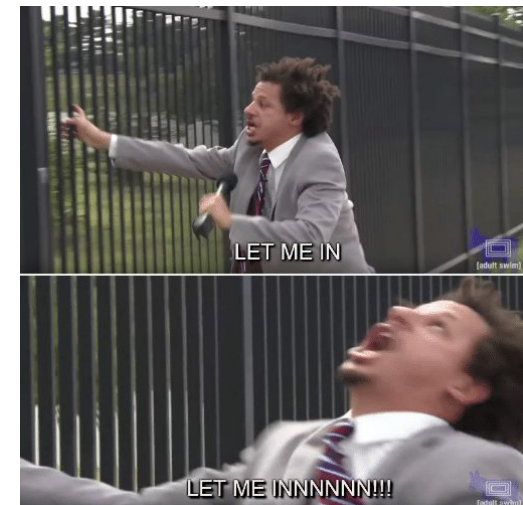
### Scheduling mechanism

- The scheduling mechanism is generally broken down into 2 parts applying 2 complementary scheduling policies:
  - The **short-term** scheduler (or CPU scheduler) that chooses which process to run (allocates the processor to it) during a millisecond period
  - The **long-term** scheduler (or admission/task scheduler) which chooses which process must be admitted to the **ready** queue and will be executed during a period ranging from one second to one minute (with a potential waiting phase during the period)
- The processor scheduler decides as soon as an executing process is stopped

## 2. Mechanism

### Scheduling mechanism

- This mechanism determines and can modify process states and supports:
  - Management of new processes by giving them the **ready** state or **waiting**
  - Management of completed processes that do not have any more requests
  - Management of operation between:
    - An executing process is **blocked**, and is in a **waiting** state
    - A process is **waiting**, and becomes **ready**



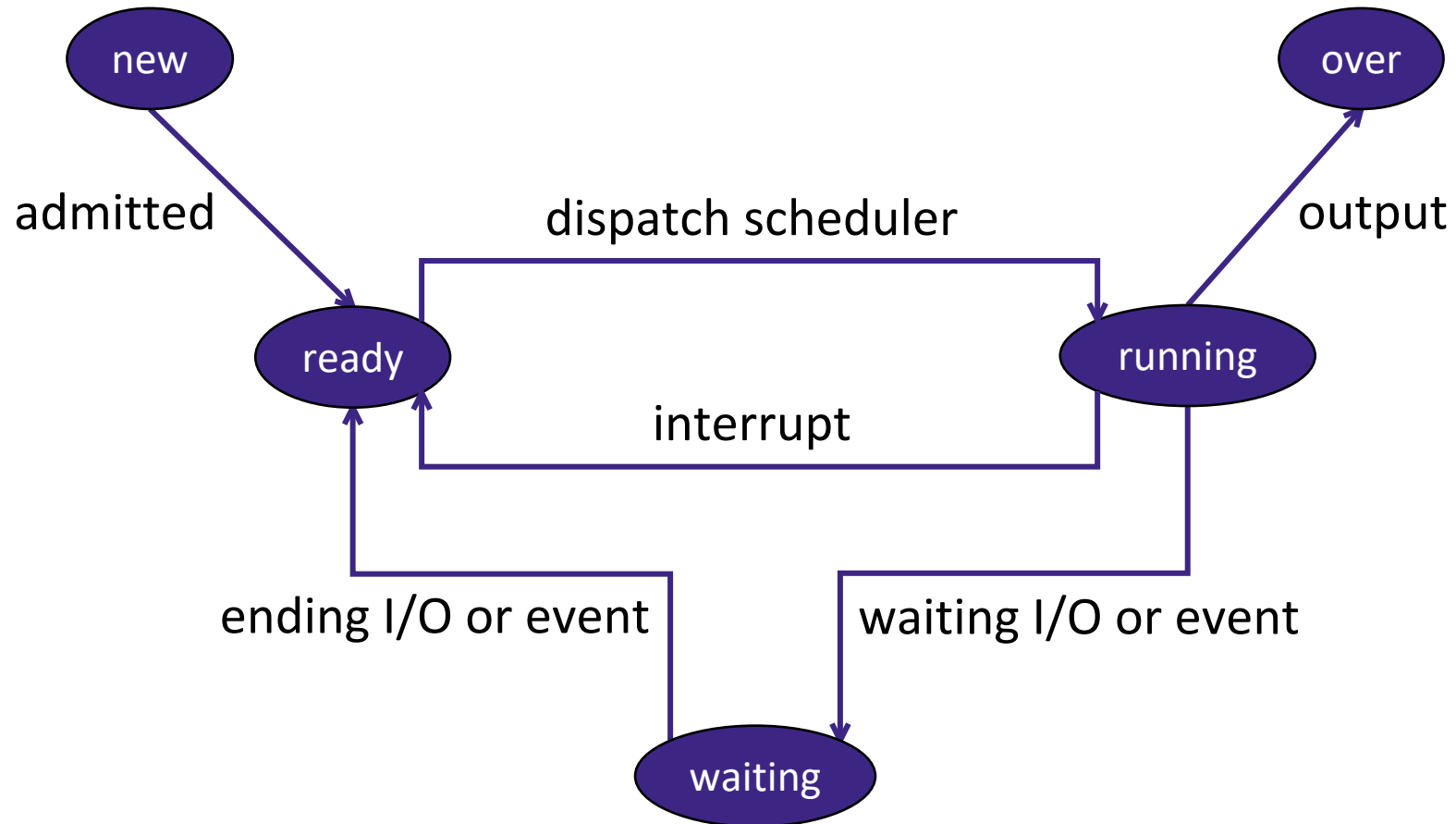
## 2. Mechanism

### Dispatcher mechanism

- To this first scheduling mechanism, we must add a **dispatcher** mechanism whose role is to pass the control of the processor to the selected process, which implies:
  - Context switching
  - Switching from **supervisor mode** to **user mode**
  - Jumping into the user program at the (re)start address
- This processing takes a certain amount of time (dispatch latency) which must be considered: the stopping and starting of a process is not immediate
- The context switching is calculated as soon as the processor is used, we assume initially that the processor was used by another process; this assumption is useful when calculating the context switching

## 2. Mechanism

### Dispatcher mechanism



## 2. Mechanism

### Scheduling requirements

- There will normally be several processes in the **ready** queue
- The scheduler must now answer the question: when the processor becomes available, which one to choose?
- The general requirements for a good scheduler are:
  - A good CPU use
  - Quick responses to user
- There are also specific scheduling requirements to be maximized:
  - **CPU utilization rate** (percentage)
  - **Throughput**: the number of processes running during all the time units allocated on the processor

## 2. Mechanism

### Scheduling requirements

- And specific scheduling requirements to be minimized:
  - **Turnaround time (TAT)**: the time taken by the process from submission to completion
  - **Waiting time (WT)**: the sum of all the time spent in the **ready** (waiting) queue
  - **Response time (RT)**: for interactive or remote systems, the time between a request and its response (first response, not necessarily the final result)
- **Completion time (CT)**: the time when the process completes its execution
- **Arrival time (AT)**: the time when the process has arrived in the ready state:  
 $TAT = CT - AT$
- **Burst time (BT)**: the time required by the process for its execution:  $WT = TAT - BT$



## 2. Mechanism

### Scheduling heuristics

- We notice 2 types of scheduling heuristics (concerns short-term scheduling):
  - **Non-preemptive** mechanisms that do not interrupt the operation of a process: only waiting on an input/output (which can stop the operation of a process) and the end of a process can cause the scheduler to be called
  - **Pre-emptive** mechanisms that can interrupt the operation of a process, which implies the implementation of a timer and more complex methods for managing context switching



## 2. Mechanism

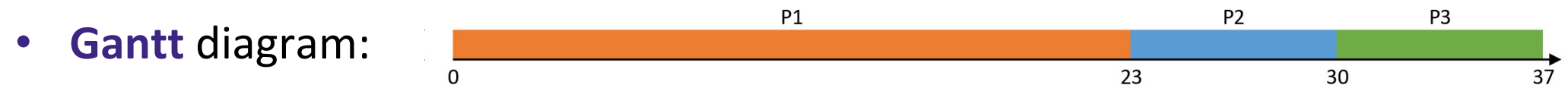
### FCFS

- With the previously mentioned requirements, there are several scheduling mechanisms
- The **FCFS** (First-Come First-Served, or First In First Out as **FIFO**) mechanism consists in using a queue where the processes waiting for the processor are stored in the order of arrival
- This scheduling favors long processes (in time units) or processes that are dependent on the CPU

## 2. Mechanism

### FCFS

- Consider processes P1 (23 TU), P2 (7 TU) and P3 (7 TU) coming in this order P1, P2, P3




- We notice that P1 does not wait, that P2 must wait for 23 TU and that P3 must wait for 30 TU
  - Average waiting time:  $(0 + 23 + 30) / 3 = 17,66\text{TU}$
  - Average turnaround time (for processes arrived at the same time):  
 $(23 + 30 + 37) / 3 = 30\text{TU}$
  - CPU utilization rate: 100%
  - Throughput:  $3 / (23 + 7 + 7) = 0,081 \text{ process/TU}$

## 2. Mechanism

### FCFS

- Consider same processes P1, P2, P3 coming in this order P2, P3, P1

- New **Gantt** diagram:

- We notice that P2 does not wait, that P3 must wait for 7 TU and that P1 must wait for 14 TU
  - Average waiting time:  $(0 + 7 + 14) / 3 = 7\text{TU}$
  - Average turnaround time (for processes arrived at the same time):  
 $(7 + 14 + 37) / 3 = 19,33\text{TU}$
  - CPU utilization rate: 100%
  - Throughput:  $3 / (7 + 7 + 23) = 0,081 \text{ process/TU}$

## 2. Mechanism

### FCFS

- In a way FCFS favors processor-dependent processes and can lead to a very bad use of both CPU and device resources
- One possibility is to interrupt processor-dependent processes from time to time to allow other processes to run (**preemption** mechanism)



## 2. Mechanism

### SJF

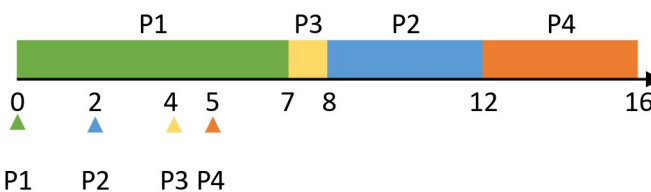
- The **SJF** (Shortest Job First) mechanism consists in selecting the process requiring the least execution time (the next shortest CPU burst)
- There are 2 approaches:
  - **Non-preemptive**: the process that has taken control of the processor does not leave it until the end of its request
  - **Preemptive**: if a new process arrives whose CPU usage time is less than the remaining execution time of the current process, then this new process obtains control of the processor (we speak then of **SRTF**, Shortest Remaining Time First)

## 2. Mechanism

### SJF

- Consider 4 processes P1, P2, P3 and P4

Process	Execution time (in TU)	Arrival time (in TU)
P1	7	0
P2	4	2
P3	1	4
P4	4	5

- Gantt** diagram:
 

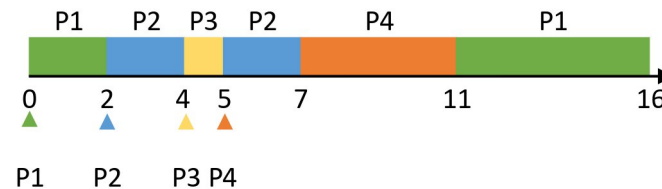
- We notice that P1 does not wait, that P2 must wait for 6 TU (8-2), that P3 must wait for 3 TU (7-4) and that P4 must wait for 7 TU (12-5)
  - Average waiting time:  $(0 + (8-2) + (7-4) + (12-5)) / 4 = 4\text{TU}$
  - Average turnaround time :  $(7 + (12-2) + (8-4) + (16-5)) / 4 = 8\text{ TU}$

## 2. Mechanism

### SJF

- Consider same processes P1, P2, P3 and P4 in a **preemptive SRTF** version

- New **Gantt** diagram:



- We notice that P3 does not wait, that P1 must wait for 9 TU (11-2), that P2 must wait for 1 TU (5-4) and that P4 must wait for 2 TU (7-5)
  - Average waiting time:  $(0 + 0 + 0 + (5-4) + (7-5) + (11-2)) / 4 = 3\text{TU}$
  - Average turnaround time :  $(16 + 5 + 1 + 6) / 4 = 7\text{ TU}$



## 2. Mechanism

### RR

- The **RR** (Round-Robin) mechanism is particularly used in time-sharing systems
- The scheduler allocates a quota (**quantum**) of processor time to each process in turn
- This mechanism is **preemptive**
  - The running process can be interrupted for internal reasons (*ex*: access to a device, end of process, *etc.*) or because the allocated time **quota** has run out
  - In the second case it is put back in the **ready** queue



## 2. Mechanism

### RR

- Consider 4 processes P1, P2, P3 and P4

Process	Execution time (in TU)	Arrival time (in TU)
P1	53	0
P2	17	0
P3	68	0
P4	24	0

- We set the quota (Q) to 20 TU

- Gantt** diagram:



## 2. Mechanism

### RR

- Average waiting time:  $((77-20)+(121-97)) + 20 + (37+(97-57)+(134-117)) + (57+(117-77)) / 4 = (81 + 20 + 94 + 97) / 4 = 73\text{TU}$
- Average turnaround time :  $(134 + 37 + 162 + 121) / 4 = 113,5\text{TU}$
- The times are much higher than with **SJF**, but no process is favored
- In order to apply one mechanism rather than another, analysis must be done to optimize performance

## 2. Mechanism

### Questions



# 3. Planning



## 3. Planning

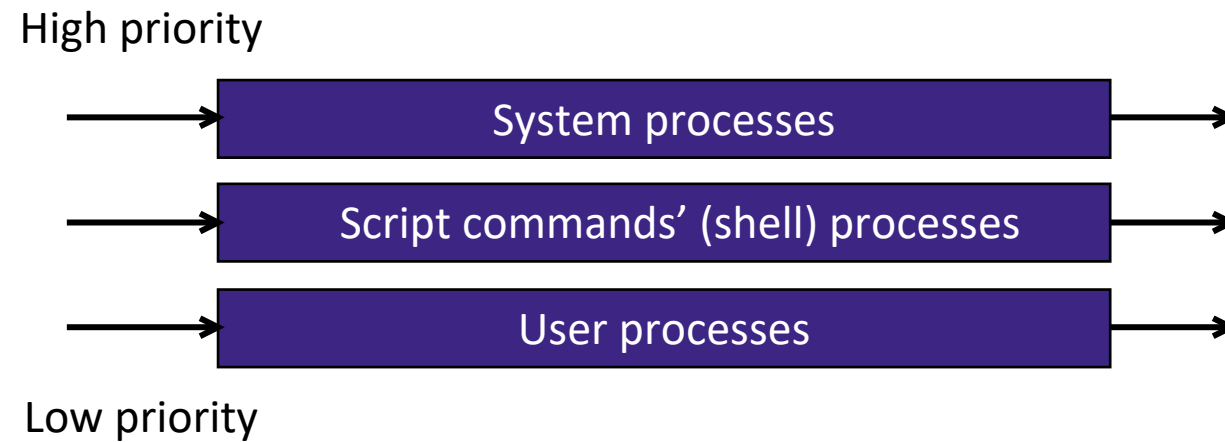
### Priority scheduling

- This mechanism consists in considering a priority coefficient associated with each process
- The processor is allocated to the highest priority process
  - **Non-preemptive** scheduling: at the end of the current process the system chooses the process with the highest priority
  - **Preemptive** scheduling: high priority processes cause the interruption of lower priority processes
- Low priority processes may never run (infinite wait), this is called **starvation**
  - Use “*aging*” to increase the priority of processes, we change the priority of a process according to its age and execution history

### 3. Planning

#### Adaptive multi-level queue scheduling

- In today's computers we can have systems that use several queues to determine priorities



- We have several queues, with different priorities and algorithms
- The process chooses to place itself in a queue according to its previous performance or its type

## 3. Planning

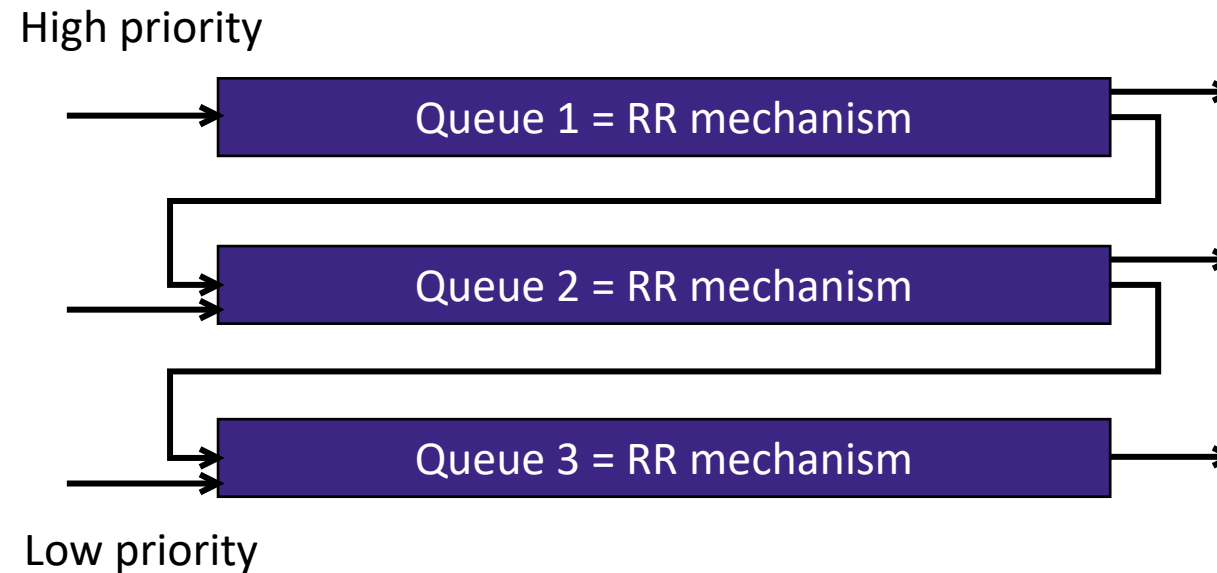
### Adaptive multi-level queue scheduling

- Parameters that describe this approach:
  - Number of queues
  - Scheduling mechanism in each queue
  - Method used to determine in which queue we introduce a process whose request can be served
- In today's computers we can have systems using dynamic modification of priorities (setting up of new queues called **retroactive** or **return** queues)
- A process can move from one queue to another, if it has spent too much time in one queue



### 3. Planning

#### Adaptive multi-level queue scheduling

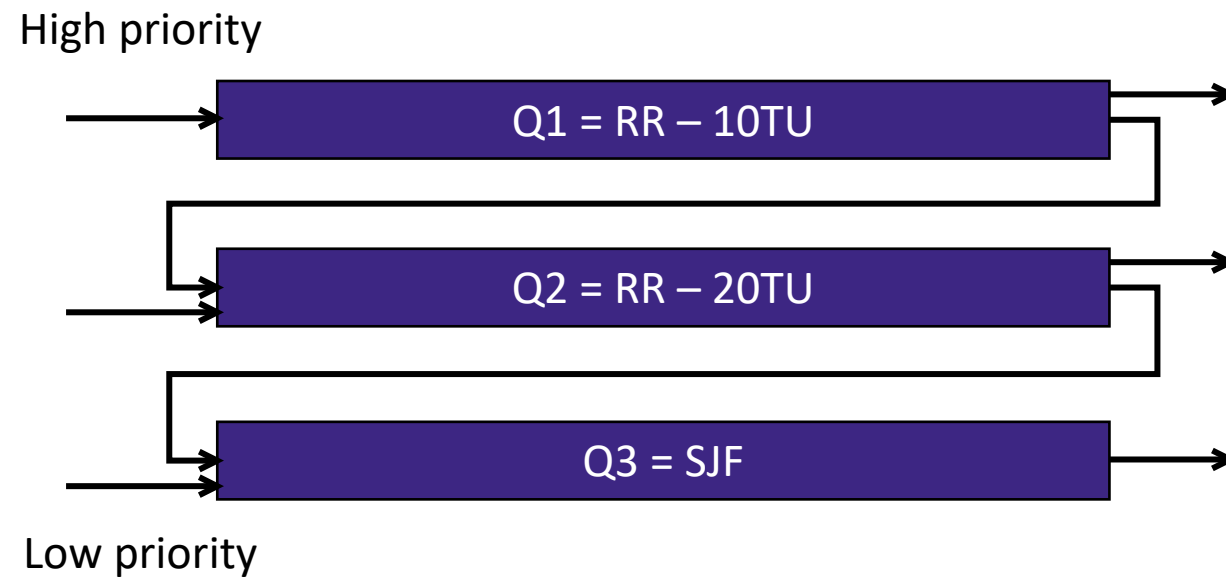


- In addition to the number of queues and the scheduling mechanism in each queue, we need to determine:
  - Algorithm for allocating queues to processes
  - Algorithm for allocating processor to processes
  - Method used to determine when a process changes queue (moves to a higher or lower priority queue)

### 3. Planning

#### Adaptive multi-level queue scheduling

- Consider queues Q1 (**RR** quantum 10 TU), Q2 (**RR** quantum 20 TU) and Q3 (**SJF**)



## 3. Planning

### Adaptive multi-level queue scheduling

- Possible scheduling according to an algorithm:
  - A new process enters Q1 where there are already running processes, it gets its 10 TU on the processor
  - This new process does not complete with 10 TU, it is placed in Q2, and gets 20 additional TU
  - It does not complete; it is stopped at 20 TU and placed in Q3
  - In Q3 if it is the shortest, it is completely executed otherwise it can be placed in Q1 again

## 3. Planning

### Real-time systems

- Real-time **hardware** systems, the scheduler must take into consideration:
  - Importance of knowing the critical actions and functions (*ex: controller*)
  - Importance of knowing the duration and deadlines of critical actions
  - Importance of guaranteeing that the actions are all executed without time overruns (resource allocations in advance, no waiting situation, no access, *etc.*)
- Real-time **software** systems:
  - The duration and deadlines remain important, but are not necessarily considered critical (*ex: systems in UDP exchanges*)
  - Assigning high **priority** for important or even critical processes
  - Need to allow **preemption** on system calls (**kernel mode**)

## 3. Planning

### Real-time systems

- In these real-time systems, there are then 2 types of scheduling:
  - Static scheduling:
    - Everything is designed in advance and therefore can be scheduled
    - Cyclic process
    - **RMS** (Rate-Monotonic Scheduling)
  - Dynamic scheduling:
    - The deadlines are known as they happen
    - Non-cyclic process
    - **EDF** (Earliest Deadline First)
    - **LLF** (Least Laxity First)



# 3. Planning

## Questions



# 4. Algorithm Rating



## 4. Algorithm Rating

### Analytical models

- We evaluate, for each algorithm, the parameters with predefined conditions (workload) that are as close as possible to the real conditions
- Example:

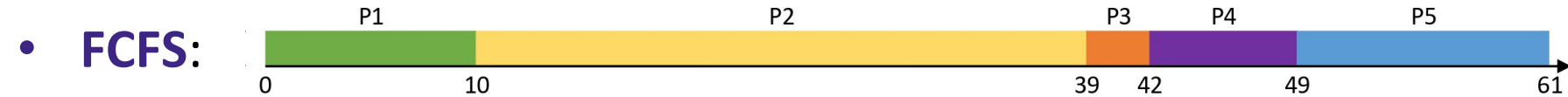
Process	Execution time (in TU)	Arrival time (in TU)
P1	10	0
P2	29	0
P3	3	0
P4	7	0
P5	12	0

- We want to establish the scheduling algorithm that minimizes the average waiting time

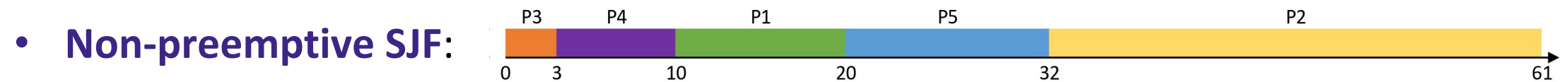


## 4. Algorithm Rating

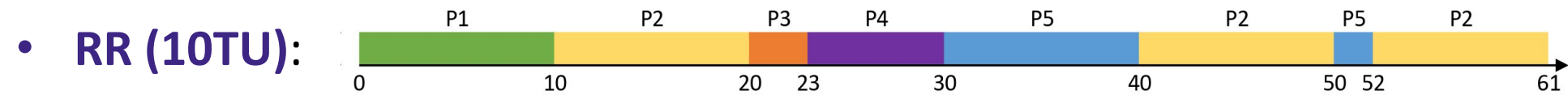
### Analytical models



- $WT = (0 + 10 + 39 + 42 + 49) / 5 = 28TU$



- $WT = (10 + 32 + 0 + 3 + 20) / 5 = 13TU$



- $WT = (0 + 32 + 20 + 23 + 40) / 5 = 23TU$

## 4. Algorithm Rating

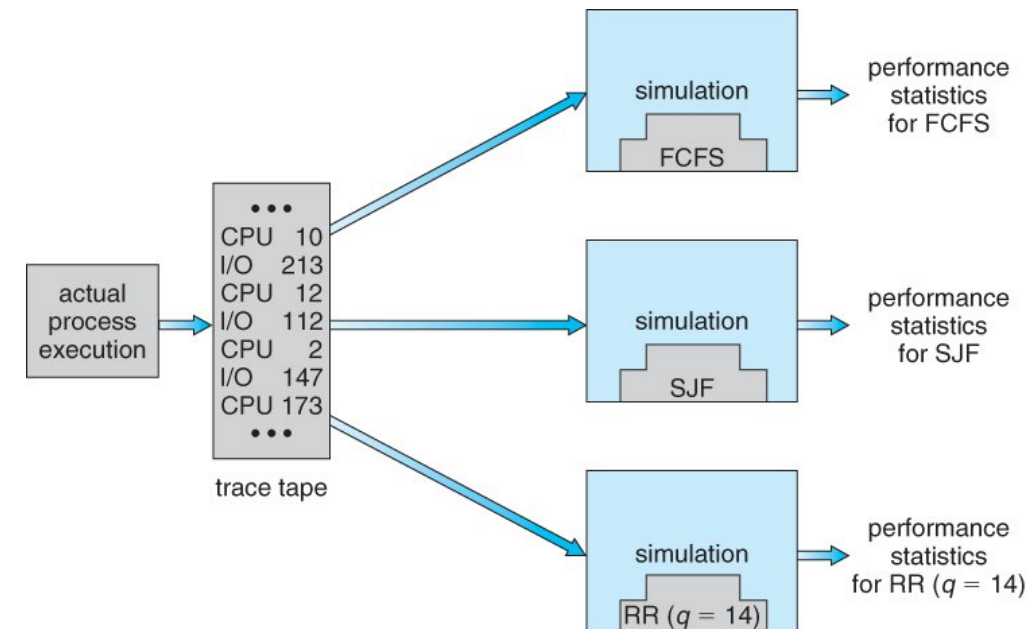
### Statistical models

- Analytical models use **cases** that simulate the reality when the system load is more or less always the same
- Since systems are rarely in this situation, we prefer to use statistical models, based on queue analysis:
  - The arrival in the different queues managed by the operating system is calculated based on a probability distribution
  - The burst time is also calculated based on a (different) probability distribution
- To establish statistical models that approximate reality, measurements must be taken over significant periods of time

## 4. Algorithm Rating

### Simulation

- The simulation method has the advantage of considering the **time** element which is introduced by the production order of the events
- To obtain realistic load samples, we systematically record real data obtained during normal system operation
- These data are then taken and introduced into simulators that reproduce the behaviors of the scheduling algorithms



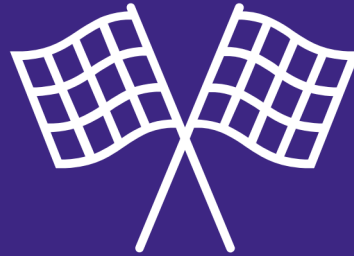
## 4. Algorithm Rating

### Questions



# Operating System Process and Resource Management

## Scheduling



Thank you for your attention