OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

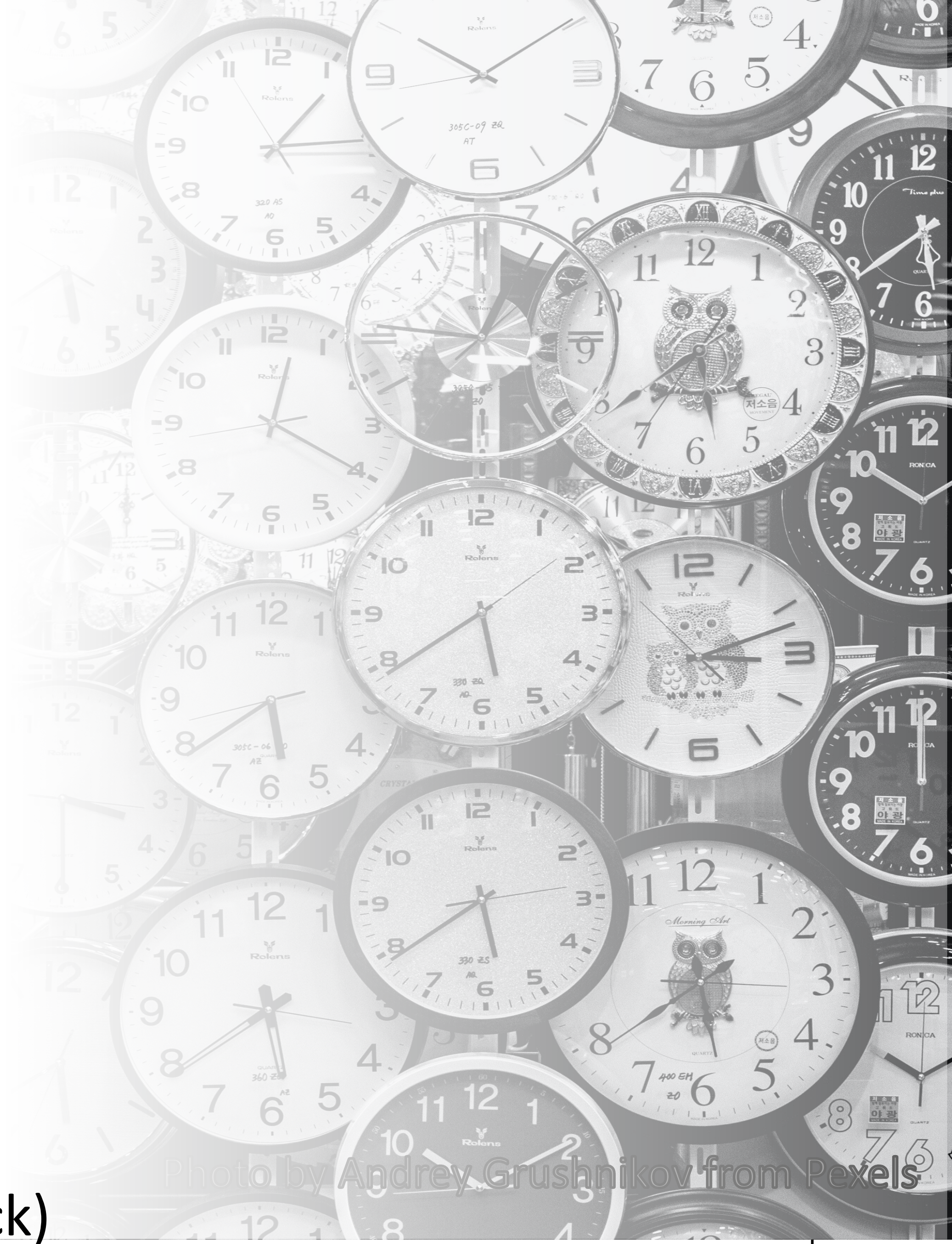

Dr Takfarinas Saber
takfarinas.saber@universityofgalway.ie


# CT213 Computing System & Organisation

## Lecture 5: CPU Management - Scheduling

# Content

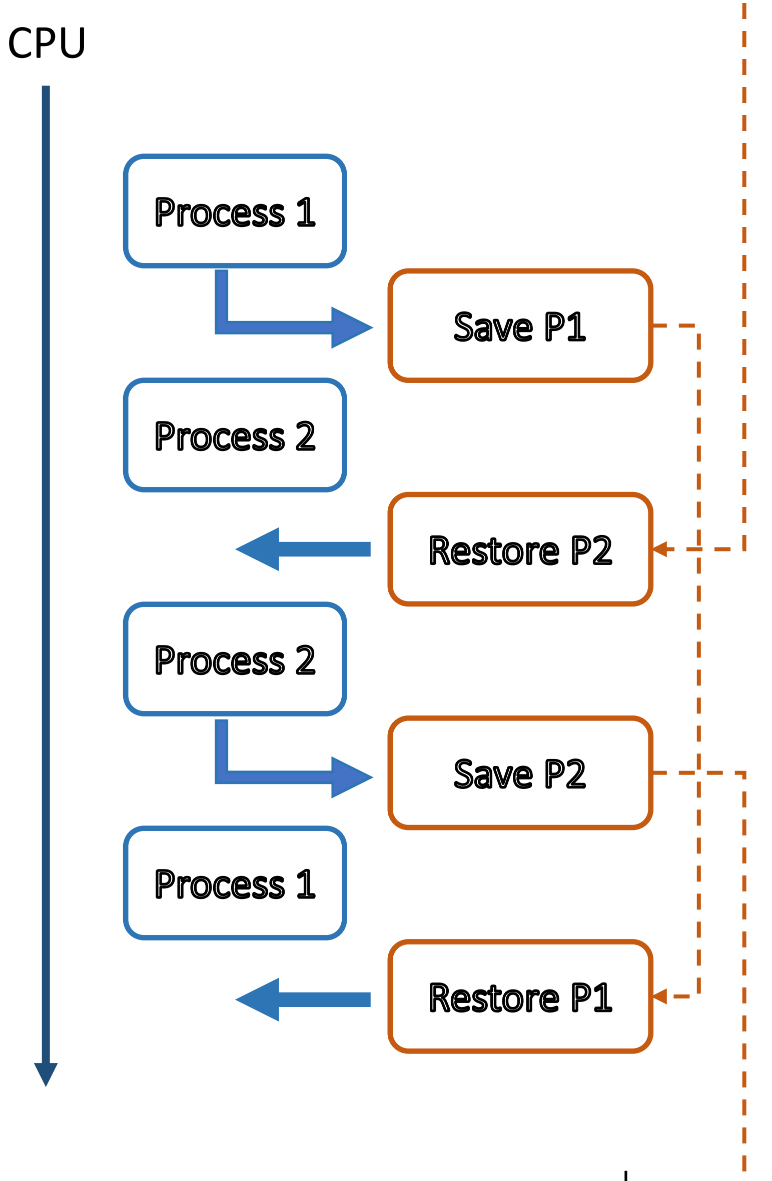
- Process scheduler organisation
- Scheduler types:
  - Non-preemptive
  - Preemptive
- Scheduling algorithms
  - FCFS (First Come First Served)
  - SRTN (Shortest Remaining Time Next)
  - SJF (Shortest Job First)
  - Time slice (Round Robin)
  - Priority based preemptive scheduling
  - MLQ (Multiple Level Queue)
  - MLQF (Multiple Level Queue with Feedback)



# Scheduling

- Scheduling allows one process to use the CPU while the execution of another process is on hold (i.e., in waiting state) due to unavailability of any resource like I/O etc
  - Aims to make the system efficient, fast and fair.

- Scheduling is part of the process manager



# Scheduling

- Scheduling is the mechanism that handles
  - **the removal of the running processes from the CPU**
  - **and the selection of another process**
- It is responsible for ***multiplexing*** processes on the CPU.

  -> when it is time for the ***running*** process to be removed from the CPU (in a *ready* or *suspended* state), a different process is selected from the set of processes in the ready state
- The selection of another process is based on a particular strategy.
  - The **scheduling algorithm** will determine the order in which the OS will execute the processes.

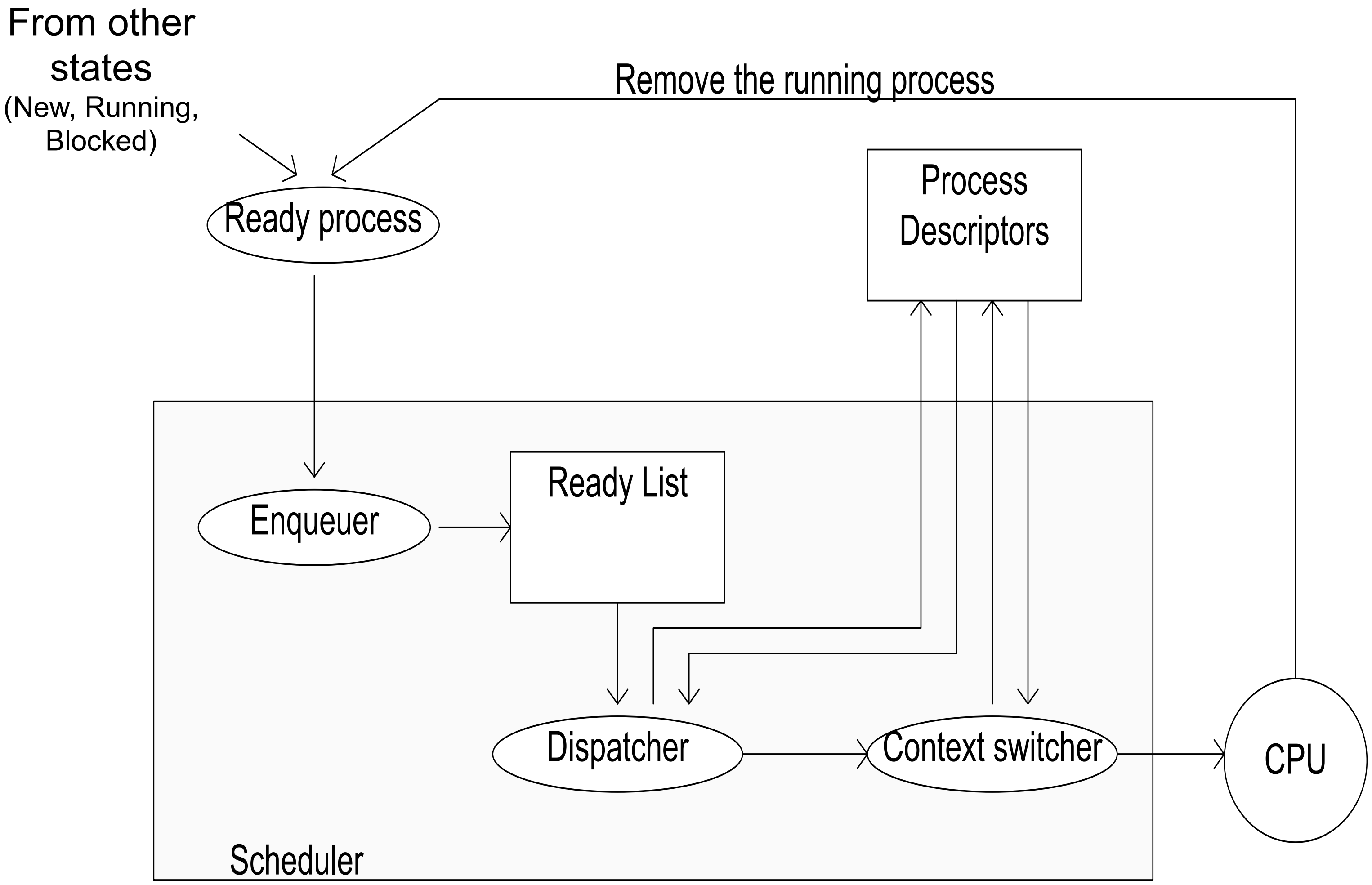
# Scheduler Organisation

When a process is changed in the ready state, the ***enqueuer*** places a pointer to the process descriptor into a ready list

***Context switcher*** saves the content of all processor registers of the process being removed into the process' descriptor, whenever the scheduler switches the CPU from executing a process to executing another

- Voluntary context switch
- Involuntary context switch

The ***dispatcher*** is invoked after the current process has been removed from the CPU; the dispatcher chooses one of the processes enqueued in the ready list and then allocates CPU to that process by performing another context switch from itself to the selected process



# Scheduler Types

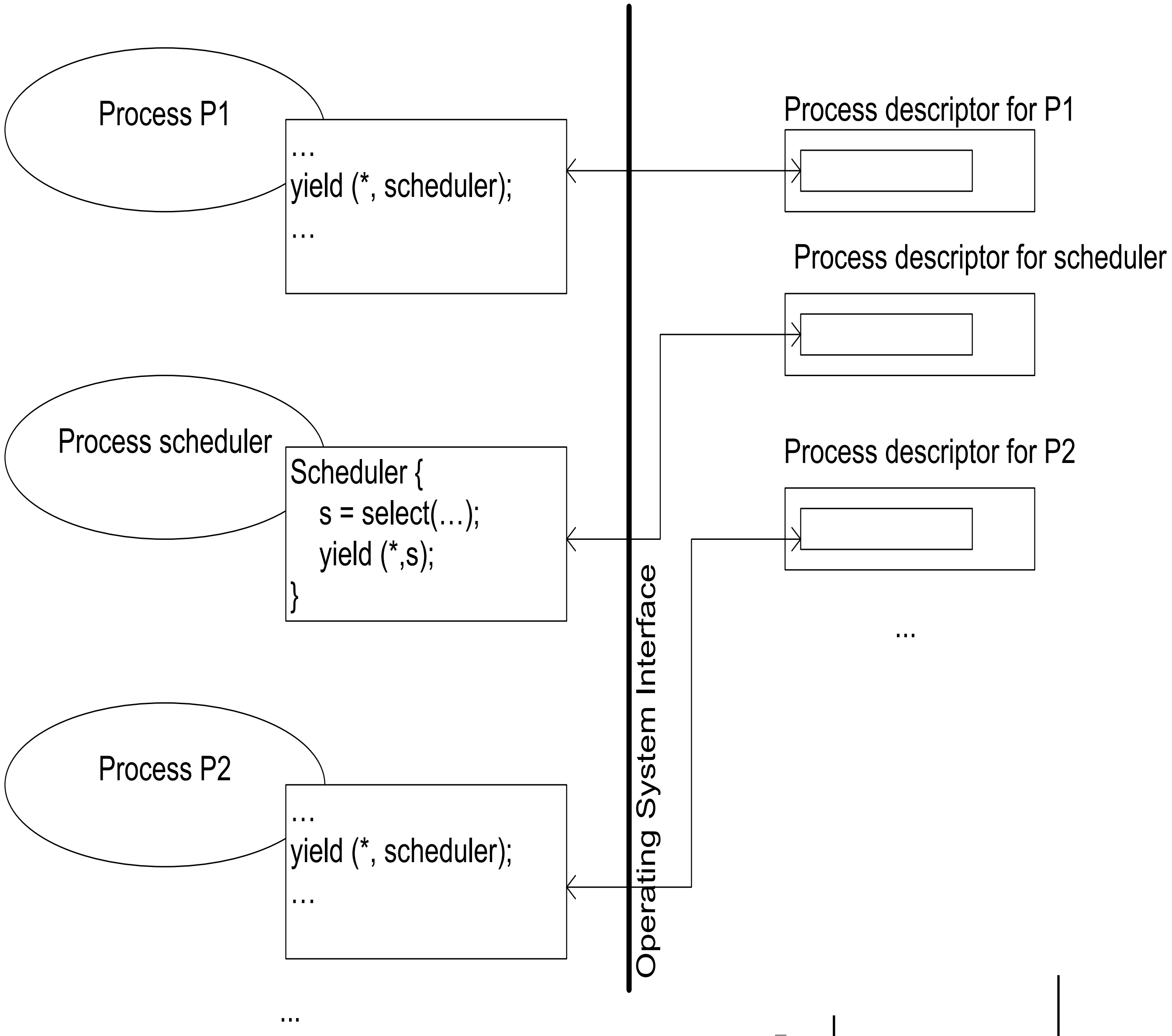
- *Cooperative* scheduler (**voluntary** CPU sharing)
  - Each process will ***periodically invoke*** the process scheduler, voluntarily sharing the CPU
  - Each process should call a function that will implement the process scheduling.
    - ***yield*** **(**$P_{current}$, $P_{next}$**)** (sometimes implemented as an instruction in hardware), where $P_{current}$ is an identifier of the current process and the $P_{next}$ is an identifier of the next process)

- *Preemptive* scheduler (**involuntary** CPU sharing)
  - The interrupt system ***enforces periodic involuntary interruption*** of any process's execution; it can force a process to involuntarily execute a yield type function (or instruction)
  - This is done by incorporating an ***interval timer*** device that produces an interrupt whenever the time expires

# Cooperative Scheduler

- Possible problems:
  - If the processes do not voluntarily cooperate with the others, one process could keep the CPU forever

- Cooperative multitasking allows much ***simpler implementation*** of applications
  - because their ***execution is never unexpectedly interrupted*** by the process scheduler



# Preemptive Scheduler

- A programmable interval timer will cause an ***interrupt*** to run every K clock ticks of an interval time
  - thus causing the hardware to execute the logical equivalent of a yield instruction to invoke the interrupt handler
- The interrupt handler for the timer interrupt will call the scheduler to reschedule the processor ***without*** any action on the part of the running process
- The scheduler decides which process is run next
- The scheduler is guaranteed to be invoked once every K clock ticks
  - Even if a given process will execute an infinite loop, it will ***not*** block the execution of the other processes

```
IntervalTimer{
  InterruptCount = InterrptCount -1;
  if (InterruptCount <=0){
    InterruptRequest = TRUE
    InterruptCount = K;
  }
}

SetInterval(<programableValue>{
  K = programmableValue;
  InterruptCount = K;
}
```

# Performance Elements

- Having a set of processes P={pi, 0<=i<n}

  - **Service time**, ***τ(pi)*** – the amount of time a process needs to be in active/running state before it completes

  - **Wait time**, ***W(pi)*** – the time the process waits in the ready state before its first transition in the active state

  - **Turn around time, $T_{TRnd}(pi)$** – the amount of time between the moment a process enters the ready state and the moment the process exits the running state for the last time

- Those elements are used to measure the ***performance*** of each scheduling algorithm

# Selection Strategies

- **Non-preemptive strategies**
  - Allow any process to run to completion once it has been allocated the control of the CPU
  - A process that gets the control of the CPU, releases the CPU whenever it ends or when it voluntarily gives up the control of the CPU
- **Preemptive strategies**
  - The highest priority process among all *ready* processes is allocated the CPU
  - All lower priority processes are made to yield to the highest priority process whenever it requests the CPU
    - The scheduler is called every time a process enters the ready queue as well as when an interval timer expires
  - It allows for equitable resource sharing among processes at the expense of overloading the system

# Scheduling Algorithms

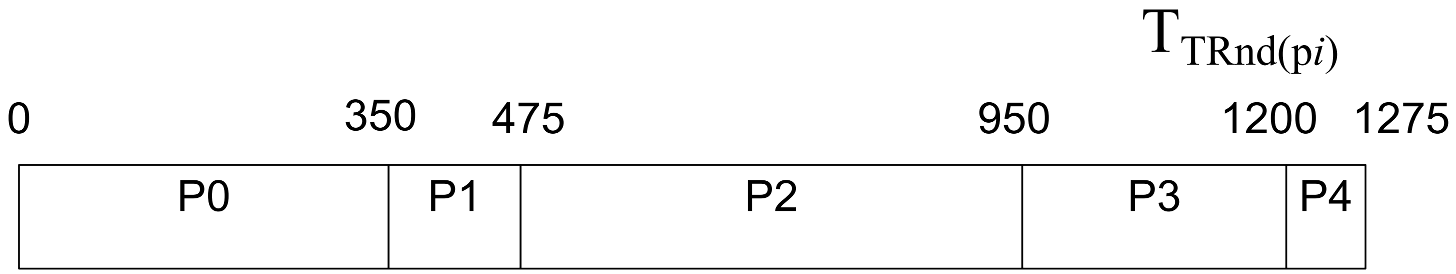
- FCFS (First Come First Served)
- SJF (Shortest Job First)
- SRTN (Shortest Remaining Time Next)
- Time slice (Round Robin)
- Priority based preemptive scheduling
- MLQ (Multiple Level Queue)
- MLQF (Multiple Level Queue with Feedback)

# First Come First Served


- ***Non-preemptive*** algorithm
- This scheduling strategy assigns ***priority*** to processes in the order in which they request the processor
  - The priority of a process is computed by the enqueuer by ***time stamping*** all incoming processes and then having the dispatcher select the process that has the ***oldest time stamp***
  - Possible implementation: using a FIFO data structure (where each entry points to a process descriptor)
    - the enqueuer adds processes to the tail of the queue and the dispatcher removes processes from the head of the queue
- Easy to implement
- It is not widely used because of processes unpredictable
  - *turn around time*
  - *waiting time*

# FCFS Example

| P$i$ | $\tau(Pi)$ |
|---|---|
| 0 | 350 |
| 1 | 125 |
| 2 | 475 |
| 3 | 250 |
| 4 | 75 |




- **Average turn around time:**
  - $T_{TRnd}$ = (350 +475 +950 + 1200 + 1275)/5 = 850

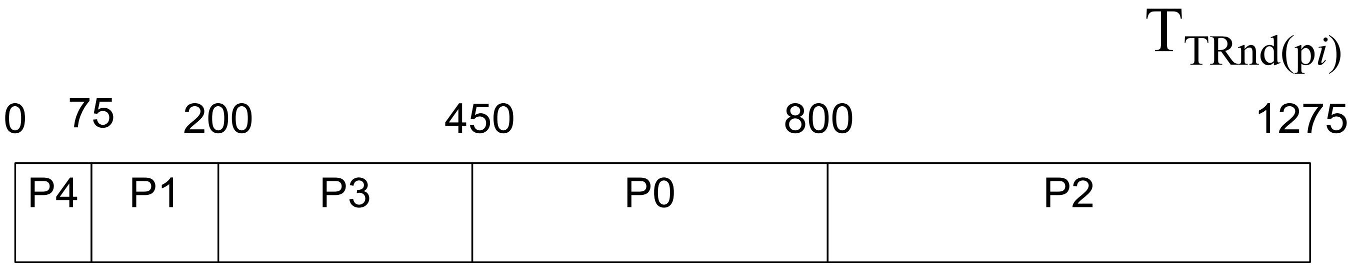
- **Average wait time:**
  - W = (0 + 350 +475 + 950 + 1200)/5 = 595

# Shortest Job First

- **N*on-preemptive***
- It is an optimal algorithm from the point of view of average turn around time
  - It minimises the average turn around time
- Preferential service of short jobs
- It requires the ***knowledge of the service time*** for each process
- In the extreme case, where the system has little idle time, the processes with large service time will never be served
- In the case where it is not possible to know the service time for each process, this is estimated using predictors.

# SJF Example

| P$i$ | $\tau$(P$i$) |
|---|---|
| 0 | 350 |
| 1 | 125 |
| 2 | 475 |
| 3 | 250 |
| 4 | 75 |




- ***Average turn around time:***
  - $T_{TRnd}$ = (800 + 200 +1275 + 450 + 75)/5 = 560
- ***Average wait time:***
  - W = (450 + 75 +800 + 200 + 0)/5 = 305

# Shortest Remaining Time Next (SRTN)

- Similar to SJF
  - But ***preemptive***

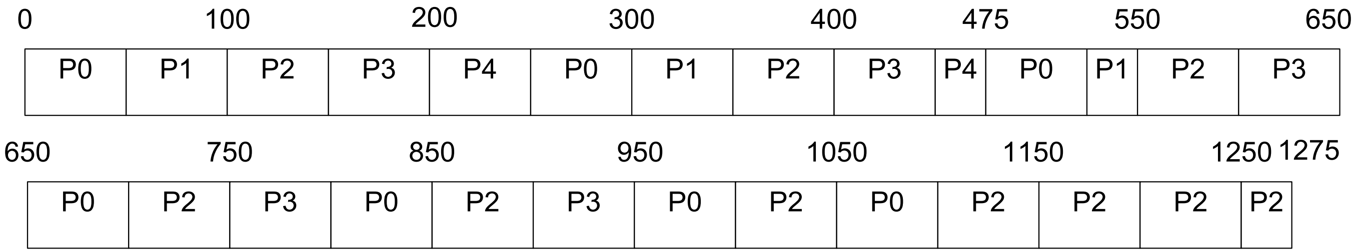
- a ***long job which is mostly complete*** might have a very short time remaining, and would therefore be prioritised

# Time Slice (Round Robin)

- **Preemptive algorithm**
- Each process gets a time slice of CPU time, distributing the ***processing time equitably*** among all processes requesting the processor
- Whenever the time slice expires, the control of the CPU is given to the next process in the ready list
  - the process being switched is placed back into the ready process list
- It implies the existence of a ***specialized timer*** that measures the processor time for each process
  - every time a process becomes active, the timer is initialized
- It is ***not well suited for long jobs***, since the scheduler will be called multiple times until the job is done
- It is very sensitive to the size of the time slice
  - Too big – large delays in response time for interactive processes
  - Too small – too much time spent running the scheduler
  - Very big – turns into FCFS
- The time slice size is determined by analyzing the number of the instructions that the processor can execute in the given time slice.

# Time Slice (Round Robin) Example

| $P_i$ | $\tau(P_i)$ |
|---|---|
| 0 | 350 |
| 1 | 125 |
| 2 | 475 |
| 3 | 250 |
| 4 | 75 |

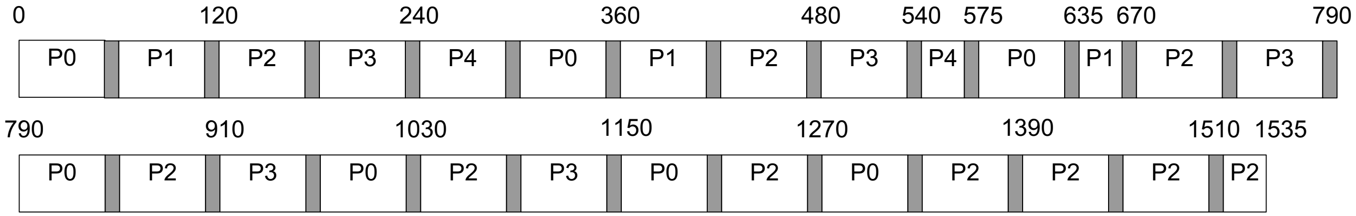



Time slice size is 50, negligible amount of time for context switching

- ***Average turn around time***:
  - $T_{TRnd}$ = (1100 + 550 + 1275 + 950 + 475)/5 = 870
- ***Average wait time***:
  - W = (750+425+800+700+400)/5 = 615
- The wait time shows the benefit of RR algorithm in the terms of how quickly a process receives service

# RR scheduling with overhead example

| i | τ(p*i*) |
|---|---|
| 0 | 350 |
| 1 | 125 |
| 2 | 475 |
| 3 | 250 |
| 4 | 75 |




Time slice size is 50, 10 units of time for context switching

- ***Average turn around time:***
  - $T_{TRnd}$ = (1320 + 660 + 1535 + 1140 + 565)/5 = 1044

- ***Average wait time:***
  - W = (620 + 535 + 1060 + 890 + 490)/5 = 719

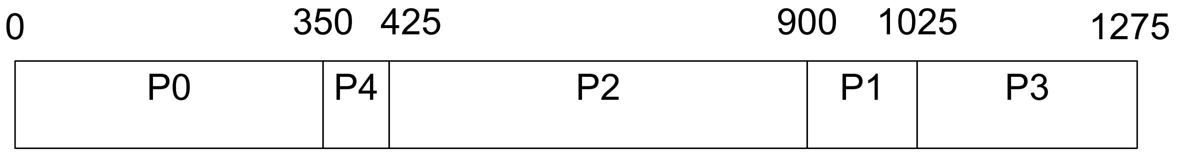
# Priority based scheduling (Event Driven)

- Both ***preemptive*** and ***non-preemptive*** variants
- Each process has an ***externally assigned priority***
- Every time an event occurs that generates a process switch, the ***process with the highest priority*** is chosen from the ready process list
- There is the possibility that processes with ***low priority will never gain CPU time***
- There are variants with static and dynamic priorities; the dynamic priority computation solves the problem with processes that may never gain CPU time (the longer the process waits, the higher its priority becomes)
- It is used for real time systems.

# Priority based schedule example

| P$i$ | $\tau$(P$i$) | Priority |
|---|---|---|
| 0 | 350 | 5 |
| 1 | 125 | 2 |
| 2 | 475 | 3 |
| 3 | 250 | 1 |
| 4 | 75 | 4 |

Highest priority corresponds to **highest** value



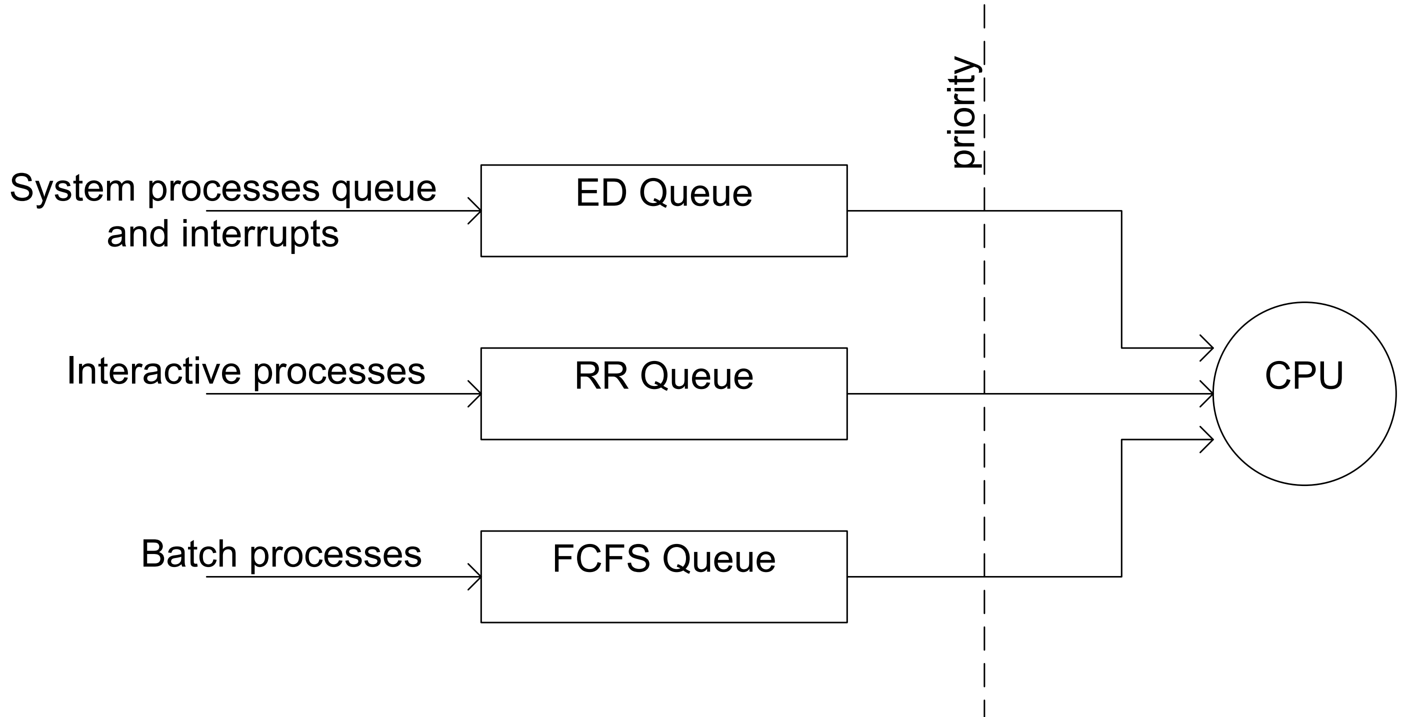

- ***Average turn around time:***
  - $T_{TRnd}$ = (350 + 425 + 900 + 1025 + 1275)/5 = 795
- ***Average wait time:***
  - W = (0 + 350 + 425 + 900 + 1025)/5 = 540

# Multiple Level Queue scheduling

- Complex systems have requirements for real time, interactive users and batch jobs
  - Therefore, a ***combined scheduling mechanism*** should be used
- The processes are divided in ***classes***
- Each class has a process queue, and it has assigned a specific scheduling algorithm
- Each process queue is treated according to a queue scheduling algorithm:
  - Each queue has assigned a priority
  - As long as there are processes in a higher priority queue, those will be serviced

# MLQ Example

- **2 queues**
  - Foreground processes (highest priority)
  - Background processes (lowest priority)
- **3 queues**
  - OS processes and interrupts (highest priority, serviced ED)
  - Interactive processes (medium priority, serviced RR)
  - Batch jobs (lowest priority, serviced FCFS)



# Multiple Level Queue with feedback

- Same with MLQ, but the processes could ***migrate from class to class*** in a dynamic fashion
- Different ***strategies*** to modify the priority:
  - Increase the priority for a given process (the user needs larger share of the CPU to sustain acceptable service)
  - Decrease the priority for a given process (the user process is trying to get more CPU share, which may impact on the other users)
  - If a process is giving up the CPU before its time slice expires, then the process is assigned to a higher priority queue
- During the evolution to completion, a process may go through a ***number of different classes***
- ***Any of the previous algorithms may be used for treating a specific process class.***

# Exercise

- Draw a Gantt Chart that illustrate the execution of these processes using the following scheduling algorithm:
  - ➢ FCFS (First Come First Served)
  - ➢ SJF (Shortest Job First) – nonpreemptive
  - ➢ SRTN (Shortest Remaining Time Next)
  - ➢ Time slice (Round Robin, assume a time slice of 1 second)
  - ➢ Priority based preemptive scheduling
- Calculate the average waiting time using each scheduling algorithm.

Larger Number = Higher Priority

| Process | Length (s) | Arrival time (s) | Priority |
|---|---|---|---|
| | 5:00 | 0:00 | 1 |
| P2 | 2:00 | 2:00 | 2 |
| P3 | 1:00 | 3:00 | 3 |

# References


- “Operating Systems – A modern perspective”, Garry Nutt, ISBN 0-8053-1295-1

- Process Scheduling: https://www.youtube.com/watch?v=THqcAa1bbFU