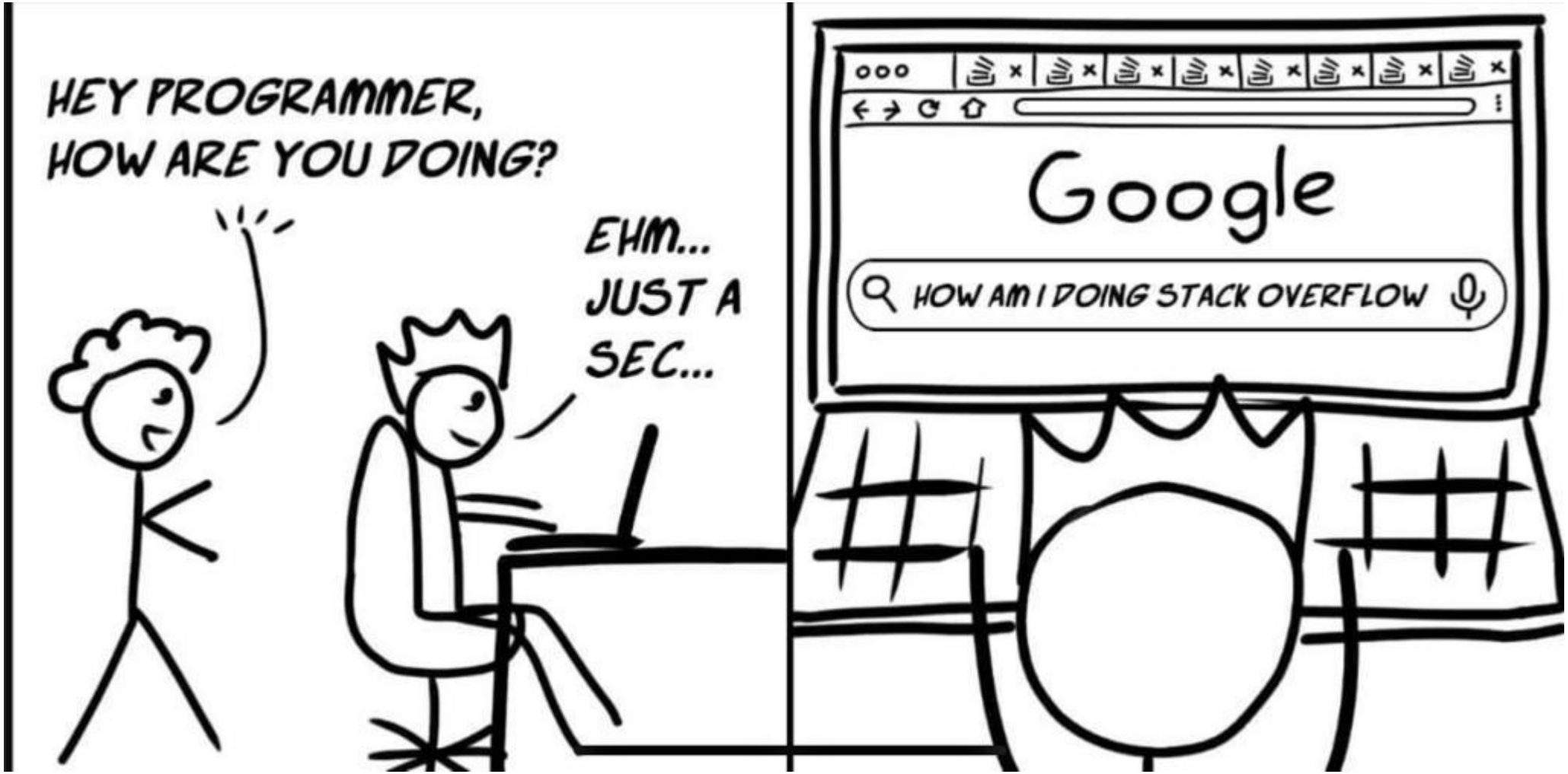
Introduction to Operating Systems and SQL for Data Science

Lecture 2 – Scheduling

Previous lecture

- Introduction to OS
- History of OS
- Processes
- Process creation (fork)
- Bash overview

Don’t hesitate to search….



Scheduling

- Which process will get CPU time?

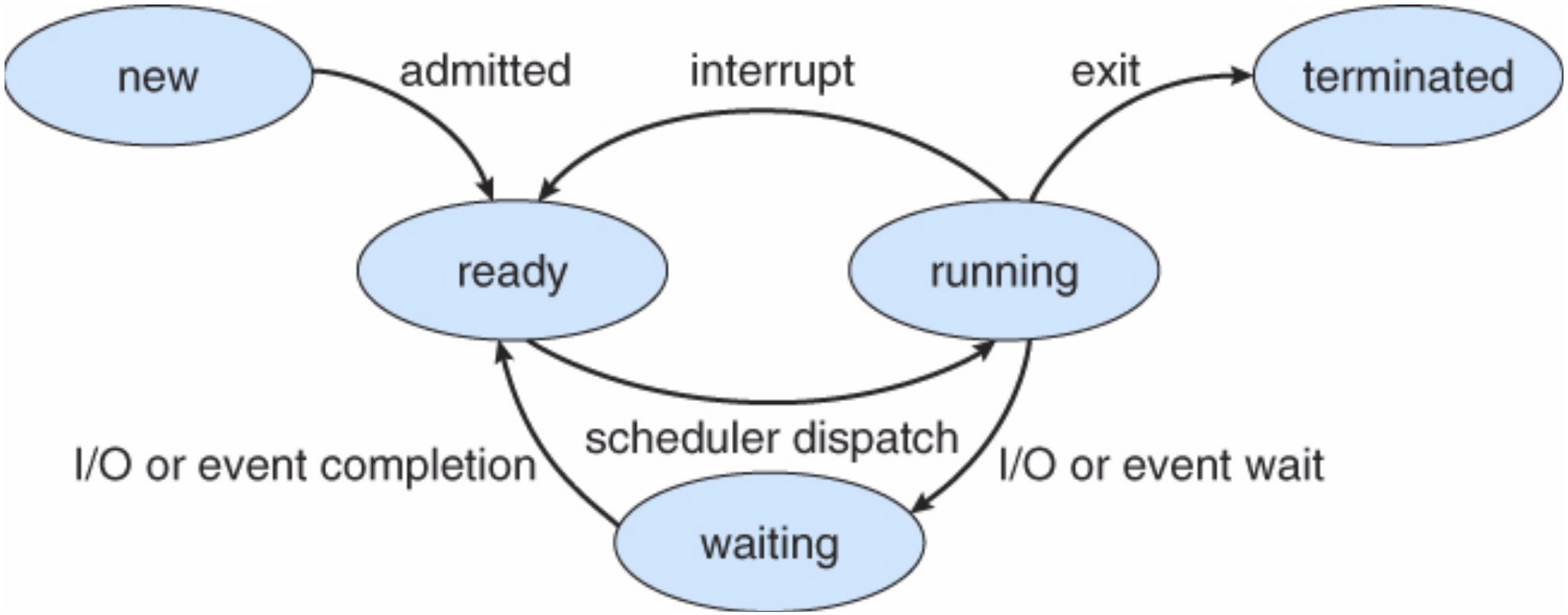
- The scheduler is part of the OS which get the decision which of the process will get to run on the CPU.

Why we need Scheduling?

- There are many processes waiting to run; More than CPUs that we can run processes on.
- We should decide which process can run on the CPU and which should be blocked. We can do it using different scheduling algorithms.
- Side effect – Starvation of processes if we don’t let processes any CPU time.

Reichman University

Process states



Reichman University

Process state

New - The process is in the stage of being created.

Ready - The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.

Running - The CPU is working on this process's instructions.

Waiting - The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.

Terminated - The process has completed.

Process communication

Process can’t access a memory of another process.

There is option for collaboration through I/O. For example, process 1 is output a record and process 2 is reading this record and doing some filtering on this record (aka piping).

The standard way to do so is by signals – which we not covered in this course.

Reichman University

CPU utilization

How much the CPU is busy?

Another definition – what is the probability that the CPU is busy in a given moment.

Example: If there is an eight processes in the system and each process is 30% of the time on IO => $1 - 0.3^8$ if the processes are Independent.

Reichman University

Context Switch Procedure

Change a process on a CPU:

1. Save the current state of the running process (all the information is stored in the processes table), program counter, registers.

2. Restore the register of the incoming process, we restore the values from the processes table.

If we switch too much the CPU will be busy in switching process instead of running them…

Reichman University

Context Switch - examples

1. If a process is end (exit/fatal error)

2. When a process is block (needs a user input/ read from a disk)

3. When a process get an interrupt. (the interrupt could come from a timer of from an OS request)

Reichman University

Scheduling algorithm types

1. Preemptive – make a running process into ready state (such as timer interrupt)

2. Non-preemptive – a process is running until its blocked or done.

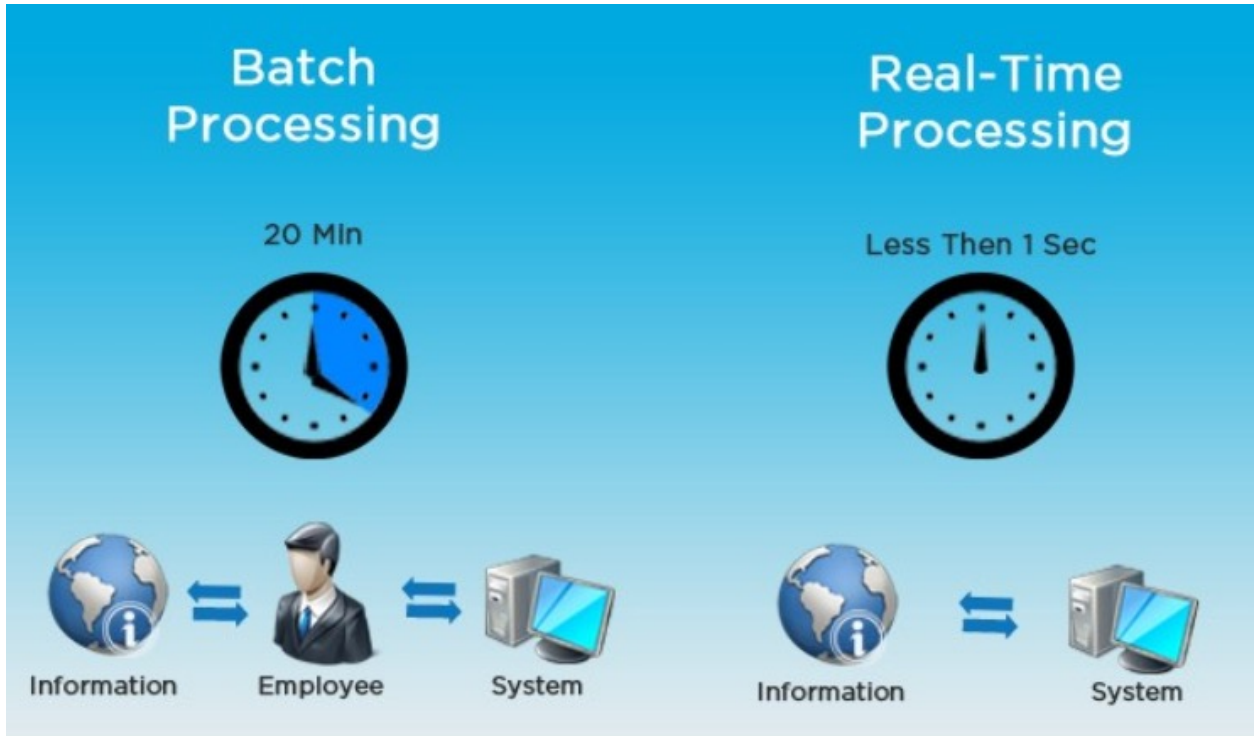
Schedulers system types

- Batch – on systems with no heavy computation, like on large enterprise (compute monthly salary, Stock, etc.). Usually, non-preemptive or preemptive with generous time.

- Interactive – systems that uses many interactive activities in the computer, or a server that respond to many users. Usually, preemptive. We want to avoid a process that takes most of the CPU time.

Reichman University

Batch vs Interactive



https://medium.com/@marudhu.pandiyan/is-real-time-processing-better-than-batch-processing-181adf5d6da2

Reichman University

Scheduling goals

1. Fairness – fair CPU time for each process (define fair…)

2. Policy enforcement – different priorities. for example, OS processes will get preference over other processes.

3. Balance – we want that all recourses will operate in parallel (CPU + IO)

Reichman University

Scheduling goals

4. Throughput – number of process that ends in a given time frame (we want to maximize)

5. Turnaround – the time of each process takes to end (we want to minimize)

6. Respond time – The time from the moment a user creates a request (for example, presses a button) until the request is executed.

Reichman University

Process-CPU Scale

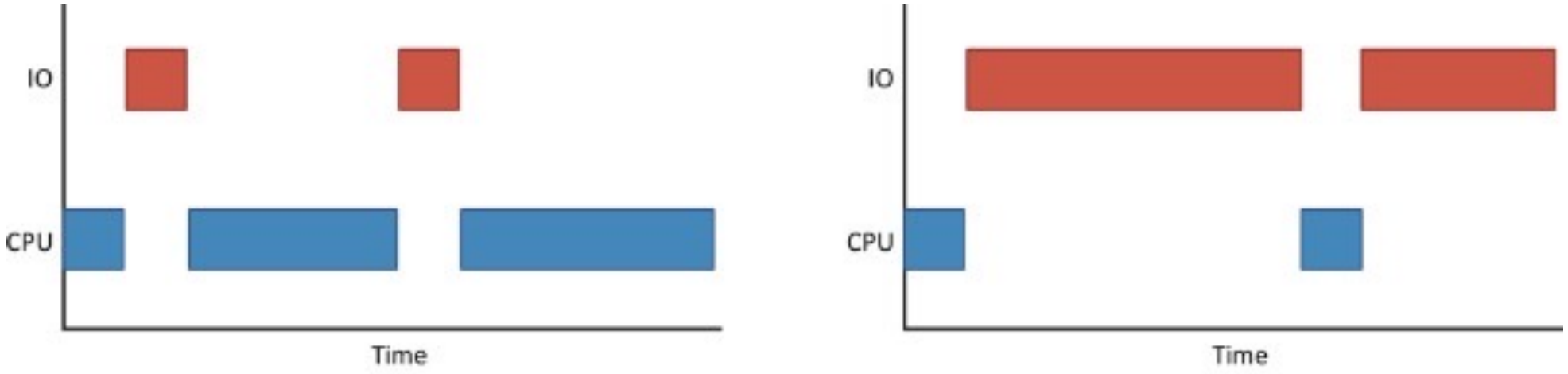
IO bound

CPU bound

- IO bound – Processes which needs many IO operation. For example, searching in a database…
- CPU bound – Processes which needs many CPU time. For example, compute median for many records.

Reichman University

IO vs CPU bound



(a) CPU-bound application

(b) IO-bound application

https://www.oreilly.com/library/view/hands-on-microservices-with/9781789342758/82531d54-039c-4c96-8fcb-58a53cee28e6.xhtml

https://stackoverflow.com/questions/868568/what-do-the-terms-cpu-bound-and-i-o-bound-mean

Reichman University

Batch System

Batch System

Run processes until they get blocked, how we let processes run on this system?

First Come First Served (FCFS)

Fair, simple but not always efficient.

Example (why the algorithm is not efficient):
Prcoess_1: 1000sec
Process_2: 1sec, wait 1000sec and then get 1sec CPU
Process_3: 1sec, wait 1001sec and then get 1sec CPU
.
.
.
Process_n: 1 sec, wait 1000+n sec and then get 1sec CPU

We can see that the turn around of the algorithm is not good…

Reichman University

Shortest Job First (SJF)

We can run the processes in an ascending order of the requested running time.

Suppose we know all the waiting processes and how much time each process needs. We will choose the processes with the minimum time first.

Shortest Job First (SJF) - Example

Four processes with the following times: 4,4,4,8 (seconds)

1. SJF (4,4,4,8): 4+8+12+20/4 = 11 sec average turn around.
2. LJF (8,4,4,4): 8+12+16+20/4 = 14 sec average turn around.

We can prove that SJF will set the optimal (minimum) turn around in the system.

Reichman University

Shortest Job First (SJF) - Prove

Suppose times: t_1,t_2,t_3,t_4 and arrange: $t_1t_2t_3t_4$
The total time is: $4t_1+3t_2+2t_3+t_4$.

t_1 factor is the highest, thus needs to be minimal.
(We can also prove it on N processes)

SJF is optimal only when all jobs come together. Otherwise, won't be optimal. There are some variants that can handle this uncertainty. Like Shortest remaining time next, which is SJF preemptive variation.

Reichman University

Shortest Job First (SJF) - Prove

We can also prove this in induction on N processes.
Basic and trivial is where n=1.

Where n=2: Suppose $t_2 > t_1$: (WLOG)

$TA(t_1 t_2) = t_1 + (t_1 + t_2) = 2t_1 + t_2$

$TA(t_2 t_1) = t_2 + (t_2 + t_1) = t_1 + 2t_2$

$TA(t_2 t_1) - TA(t_1 t_2) = t_2 - t_1 > 0$

Reichman University

Shortest Job First (SJF) - Prove

Let's look on N processes, suppose t_i is the shortest process.
Let’s look on two arrangement: (not the full process)

$TA(t_1, t_2, \ldots, t_n) = nt_1 + (n-1)t_2 + \ldots + t_n$
$TA(t_i, t_1, \ldots, t_n) = nt_i + (n-1)t_1 + \ldots + t_n$

$TA(t_1, t_2, \ldots, t_n) - TA(t_i, t_1, \ldots, t_n)$
$= nt_1 + (n-1)t_2 + \ldots + t_n - nt_i + (n-1)t_1 + \ldots + t_n$

t_i is the smallest, so we can change every t in t_i : $nt_1 > nt_i$

Reichman University

Interactive System

Interactive System

On interactive system we want the system to be highly responsive. Slow respond is **highly not** tolerated.

We can’t let a process run until it stop (in a non preemptive manner). Thus, we have to use a preemptive algorithm.

Most important term – respond time…

Round Robin

A cyclic queue of processes.

Each process have a limited time to run.

At the end of the turn, we switch processes and the last process that runs go to the back of the queue.

The only parameter is the quantum length (i.e., the running time)

Round Robin

The queue could also be a priority queue.

As such, some processes will get better positions in the queue like OS processes, rather than app processes.

We can also set dynamic priorities, so if the process is IO-bound we will let him run first. (why..)

Policies – (Simple) Example

Three process
P_1 -> 2,3,1 t=0
P_2 -> 6 t=0 (this is a CPU-bound process)
P_3 -> 2,2,2 t=2

Between each CPU request, there is a 3 seconds disk request

Terms: W-waiting, R-running, D-disk
B_D-writing/reading to disk.

Reichman University

SJF – (Simple) Example

P_1 -> 2,3,1 t=0
P_2 -> 6 t=0 (this is a CPU-bound process)
P_3 -> 2,2,2 t=2

	0	1	2	3	4	5	6	7	8	9
P_1	R	R	B_D	B_D	B_D	R	R	R	B_D	B_D
P_2	W	W	W	W	R	W	W	W	W	W
P_3			R	R	B	B_D	B_D	B_D	R	R

In t=0 P_1 is running first because he shorter than P_2

P_3 is running because P_1 writing to the disk and is shorter than P_2

P_3 is blocked because P_1 is writing to the disk, and only one process can write to the same disk

P_2 was stopped because it has more time to run than P_1 (aka pre-emption)

Reichman University

RR – (Simple) Example

P_1 -> 2,3,1 t=0
P_2 -> 6 t=0 (this is a CPU-bound process)
P_3 -> 2,2,2 t=2

Assume quantum = 2

	0	1	2	3	4	5	6	7	8	9
P_1	R	R	B_D	B_D	B_D	W	W	W	R	R
P_2	W	W	R	R	W	W	R	R	W	W
P_3			W	W	R	R	B_D	B_D	B_D	W

There is no importance who is shorter, the queue is FIFO. Let's assume P_1 was chosen randomly

The queue is for only ready processes! Blocked processes won't be included

When there is no processes that needs to run on the CPU, the CPU runs an idle process, and where there will be an interrupt (i.e., the quantum is reached) the idle will stop.
You can think of it as priority queue where idle has zero or small priority.

Reichman University

Policies - conclusion

1. FCFS – First Come First Serve (FIFO) – the first process gets CPU time.

2. Shortest job first – minimum turn around, assumption of all processes come together (hard assumption in real case scenarios)

3. Round Robin – each process has a running time (predefined).

Reichman University