Tutorial Week 9 - Scheduling

Michelle Nguyen

Contains slides from Dr. Pavol Federl (13b - scheduling)

Preemptive vs non-preemptive CPU scheduling

- **non-preemptive** — context switch happens only voluntarily
  - multitasking is possible, but only through cooperation
  - process runs until it does a blocking syscall (eg. I/O), terminates, or voluntarily yields CPU
  - example: FCFS
- **preemptive** — context switch can happen without thread's cooperation
  - usually as a direct or indirect result of some event, but not limited to clock interrupt

    eg. new job is added, existing process is unblocked
  - example: SRTN
- **preemptive time-sharing** — special case of preemptive
  - processes are context switched periodically to enforce time-slice policy
  - implemented through clock interrupts
  - without a clock, only cooperative multitasking (non-preemptive) is possible
  - example: RR
  - so common that 'preemptive' is often (mis)used to mean preemptive time-sharing

First-come-first-served (FCFS) scheduling

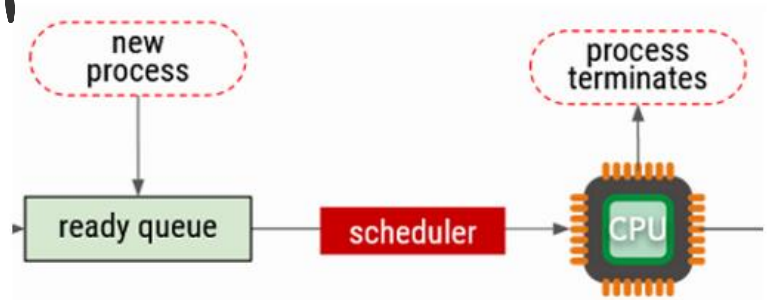
- one of the simplest scheduling algorithms
- FCFS is non-preemptive
- common in batch environments
- CPU assigned in the order the processes request it, using a FIFO ready queue
- a running job keeps the CPU until it is either finished, or it blocks
- *when running process blocks, next process from ready queue starts to execute*
- *when process is unblocked, it is appended at the end of the ready queue*
- requires **minimum number of context switches** — only N switches for N processes

Non-preemptive!

FCFS

P1

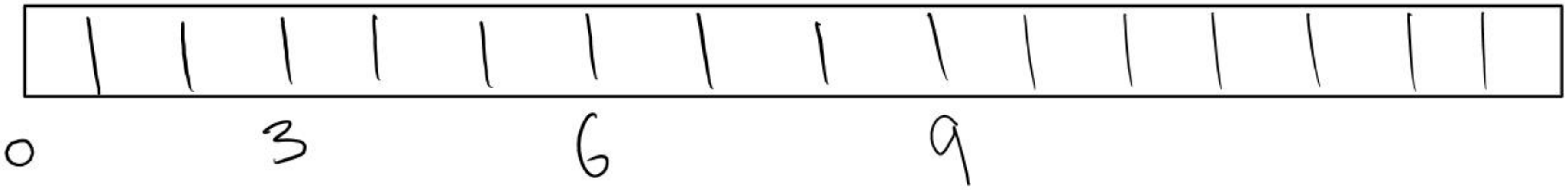
Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Above image cropped from Dr. Pavol Federl's CPSC 457 Slides (13b-scheduling)

Gantt chart

Current time: 0



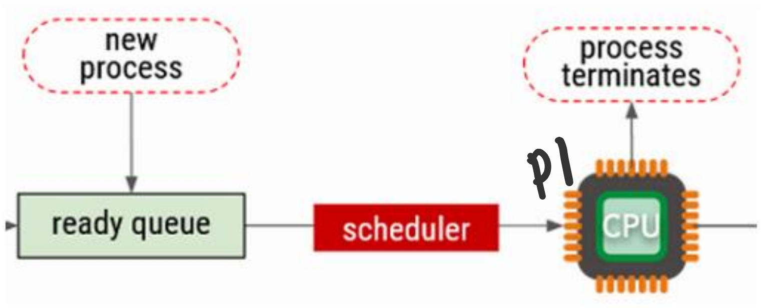
How many context switches?

Execution order?

Non-preemptive!

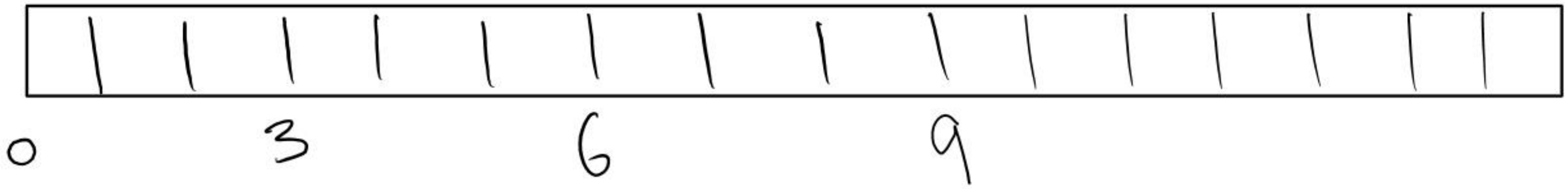
FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 0



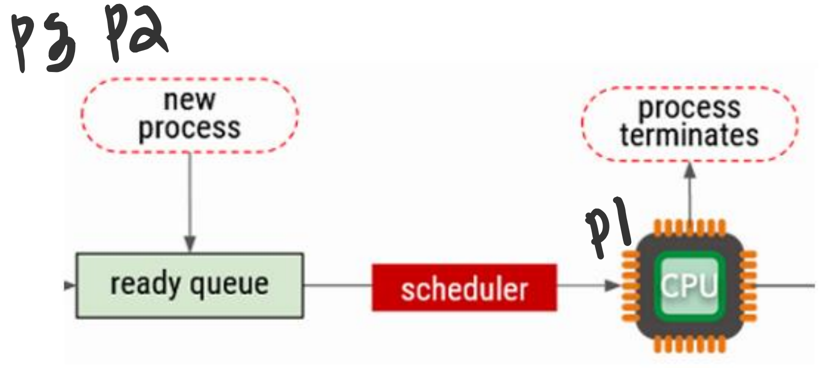
How many context switches?

Execution order?

Non-preemptive!

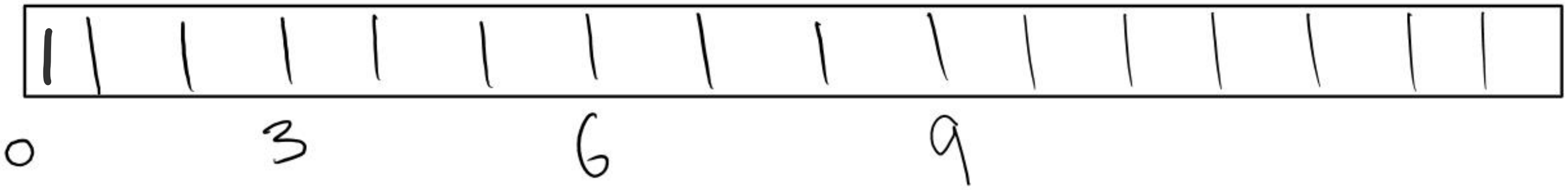
FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 1



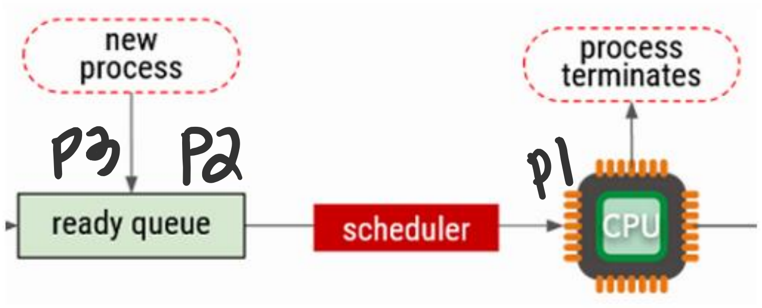
How many context switches?

Execution order?

Non-preemptive!

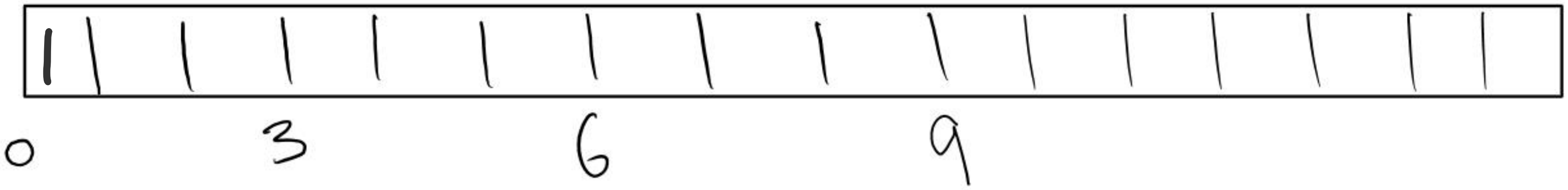
FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 1



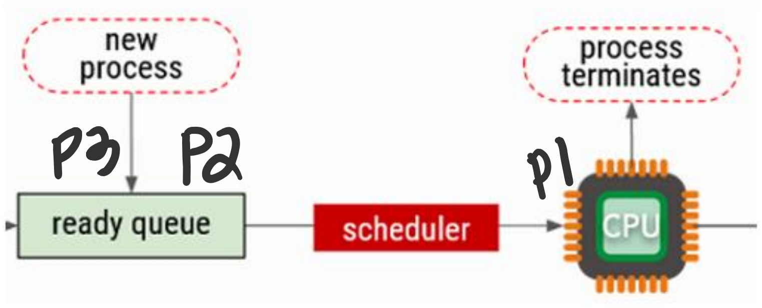
How many context switches?

Execution order?

Non-preemptive!

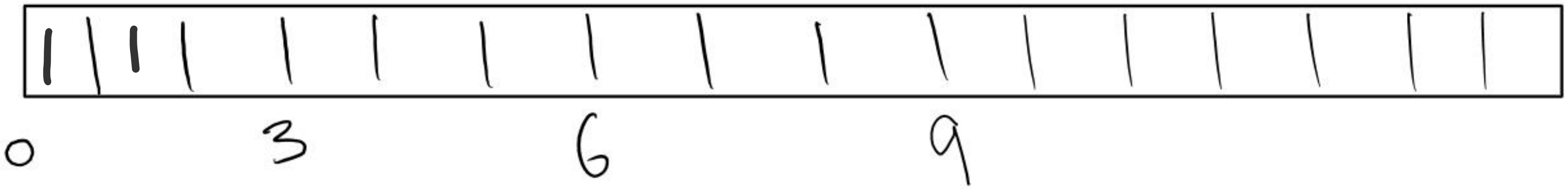
FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 2



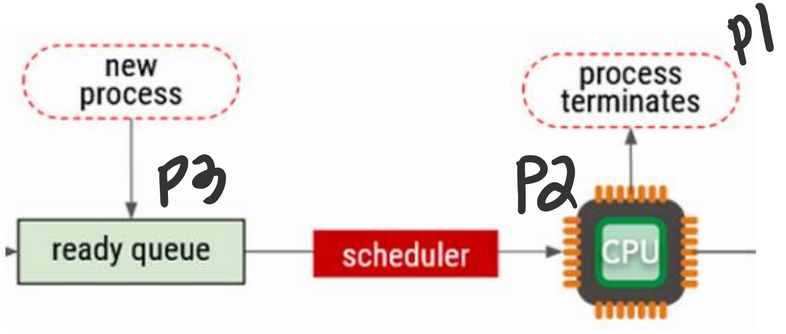
How many context switches?

Execution order?

Non-preemptive!

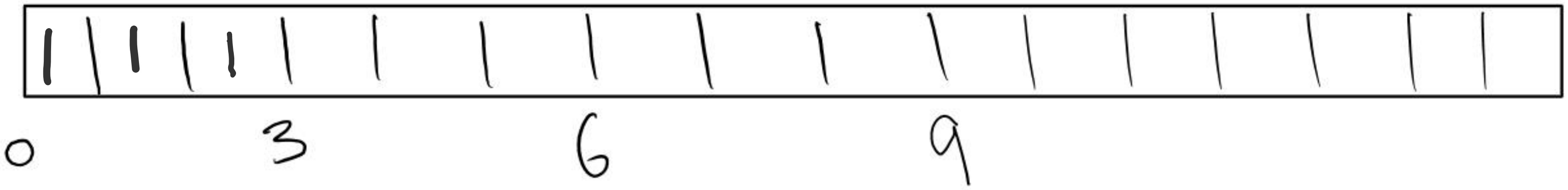
FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 3



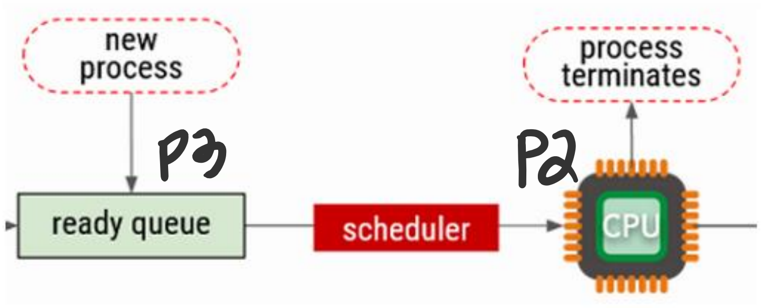
How many context switches?

Execution order?

Non-preemptive!

FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 4

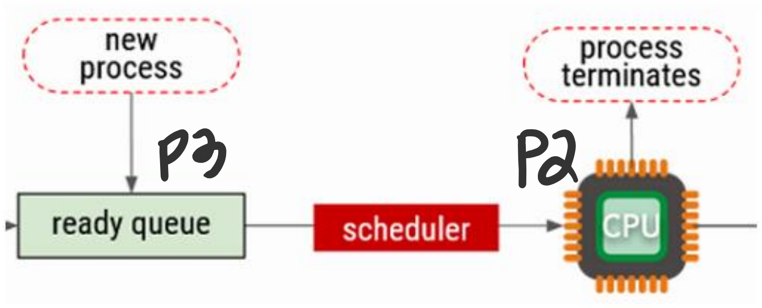
How many context switches?

Execution order?

Non-preemptive!

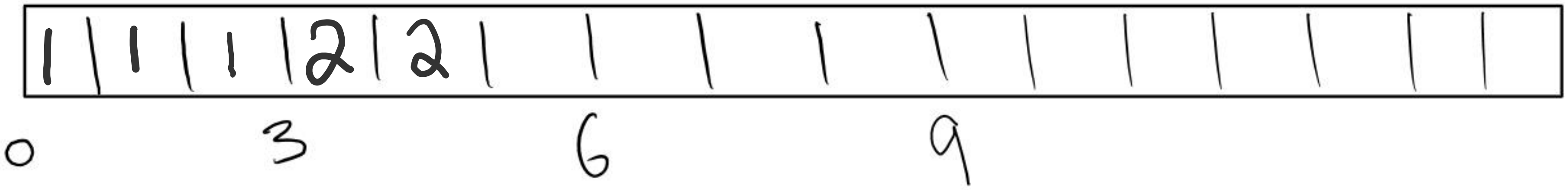
FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 5



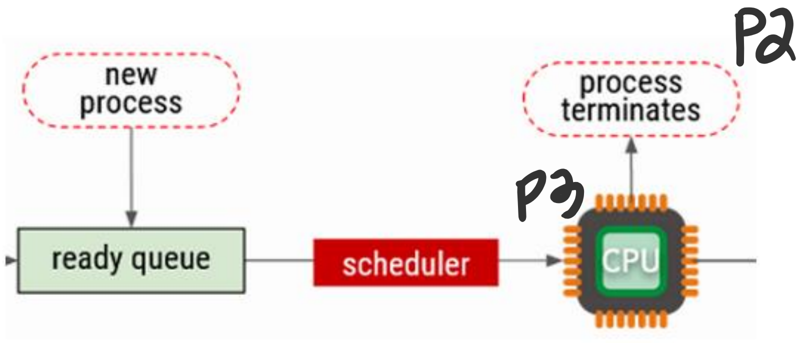
How many context switches?

Execution order?

Non-preemptive!

FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 6

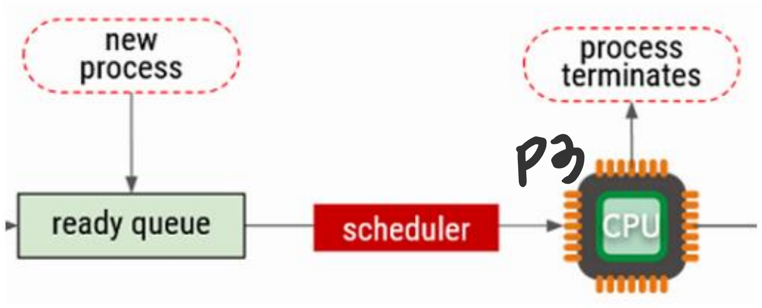
How many context switches?

Execution order?

Non-preemptive!

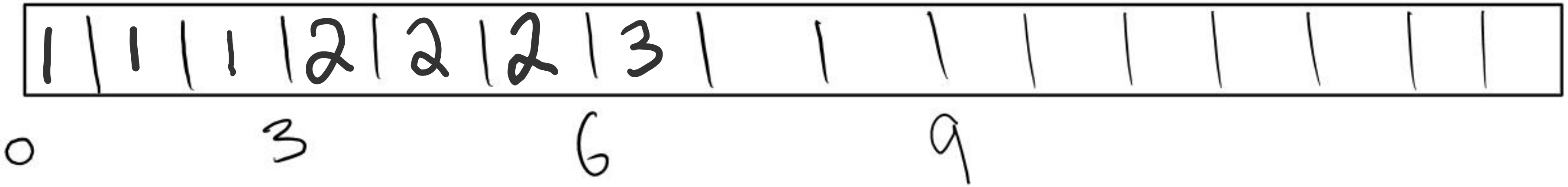
FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 7



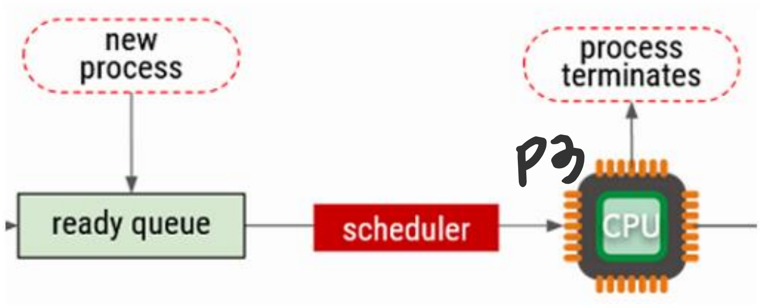
How many context switches?

Execution order?

Non-preemptive!

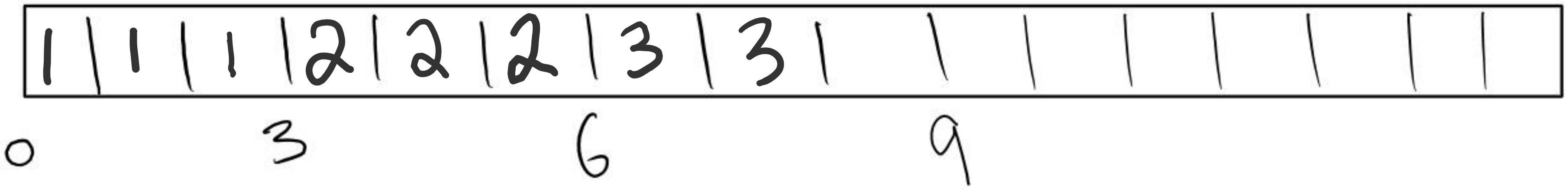
FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 8



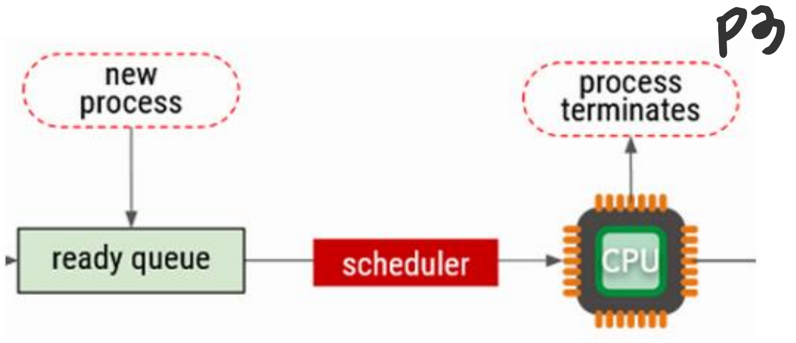
How many context switches?

Execution order?

Non-preemptive!

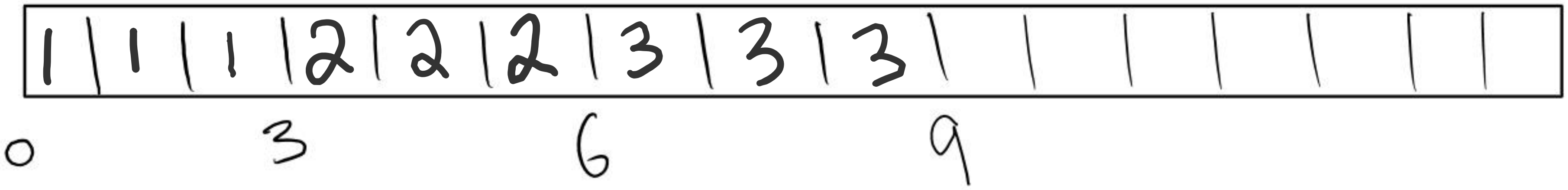
FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



Gantt chart

Current time: 9



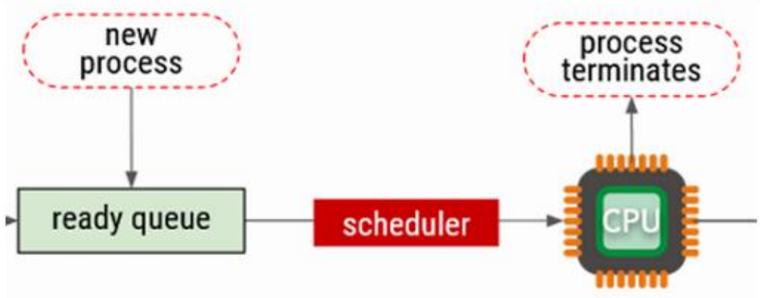
How many context switches?

Execution order?

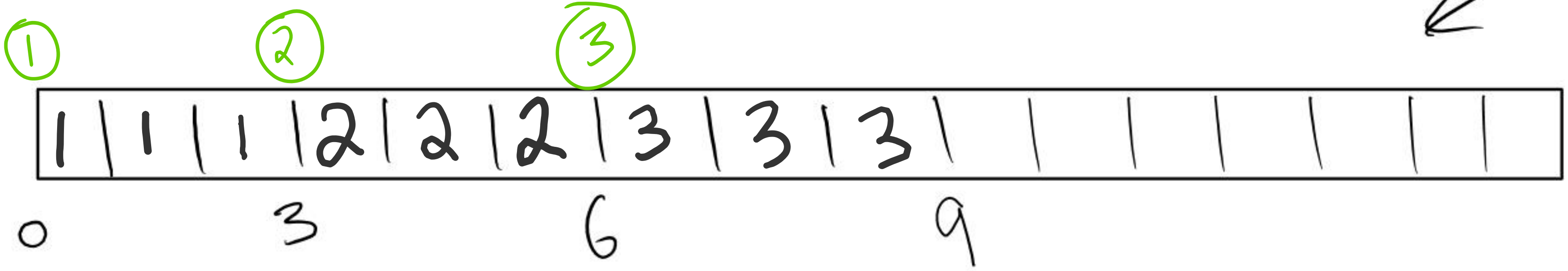
Non-preemptive!

FCFS

Process	Arrival	Burst
P1	0	3
P2	1	3
P3	1	3



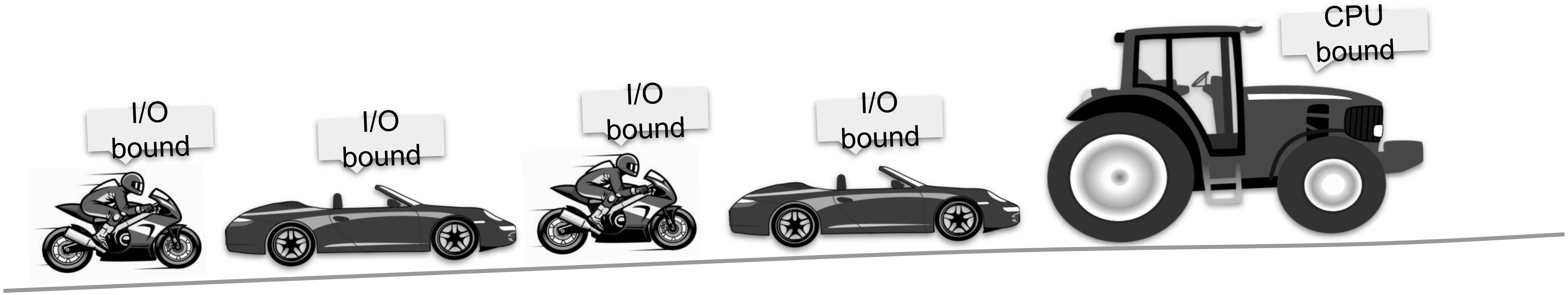
Gantt chart



How many context switches? 3

Execution order? P1, P2, P3

Convoy Effect



- big disadvantage of FCFS is the **convoy effect**
- convoy effect results in few CPU-bound process ruining the overall performance of a system with mostly IO-bound processes
- a CPU-bound process will tie up the CPU, making the IO-bound processes progress at a much slower rate
- leads to long periods of idle I/O devices

Round-robin scheduling (RR)

- RR scheduler is a preemptive version of the FCFS scheduler
- each process is assigned a time interval, called a **time slice** (aka **quantum**) e.g., 10 msec, during which it is allowed to run
- if the process exceeds the quantum, the process is preempted (context switch), and CPU is given to the next process in ready queue
- preempted process goes at the back of the ready queue
- what if the process calls blocking system call?

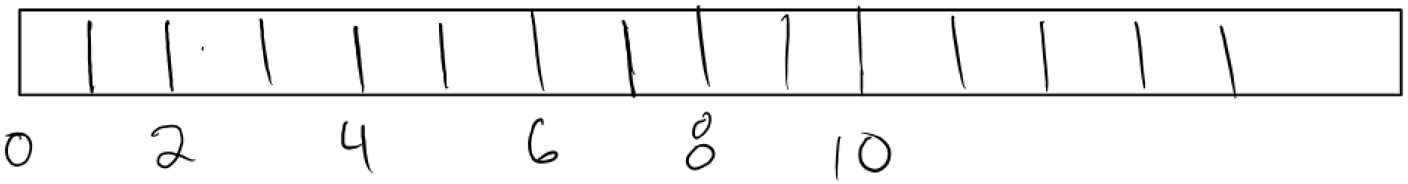
RR

Preemptive time-sharing

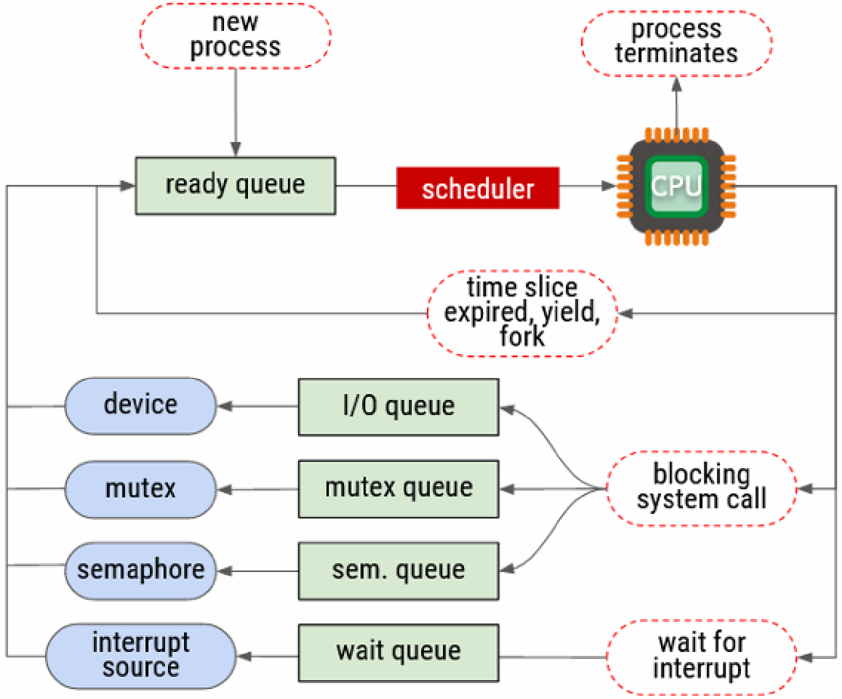
Time slice **(Quantum) : 2**

Process	Arrival	Burst
P1	0	4
P2	0	4
P3	0	4

Current time: 0



P3 P2 P1



Above image cropped from Dr. Pavol Federl's CPSC 457 Slides (13b-scheduling)

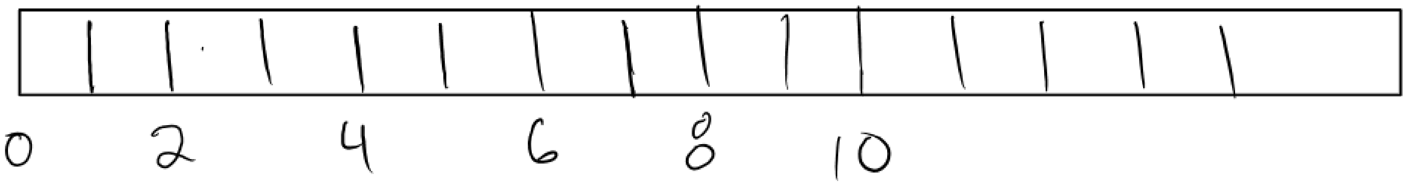
RR

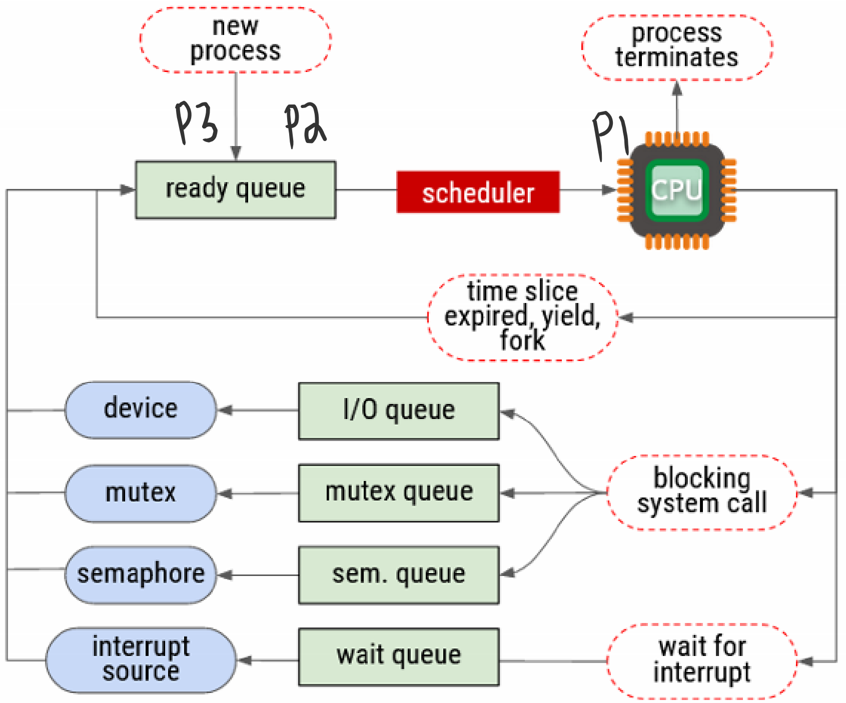
Preemptive time-sharing

$\boxed{\text{Time slice } \textbf{(Quantum) : 2}}$

Process	**Arrival**	**Burst**
P1	0	4
P2	0	4
P3	0	4

Current time: 0





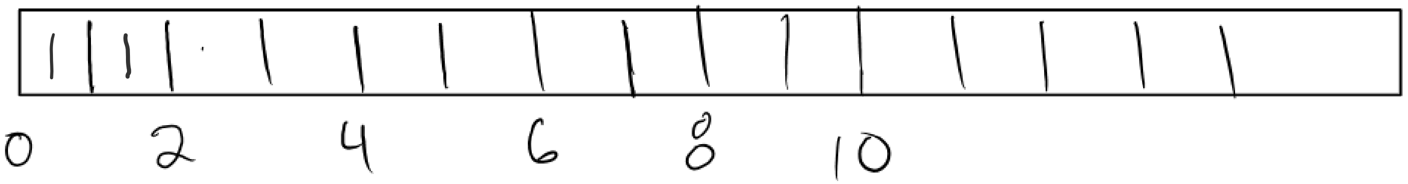
RR

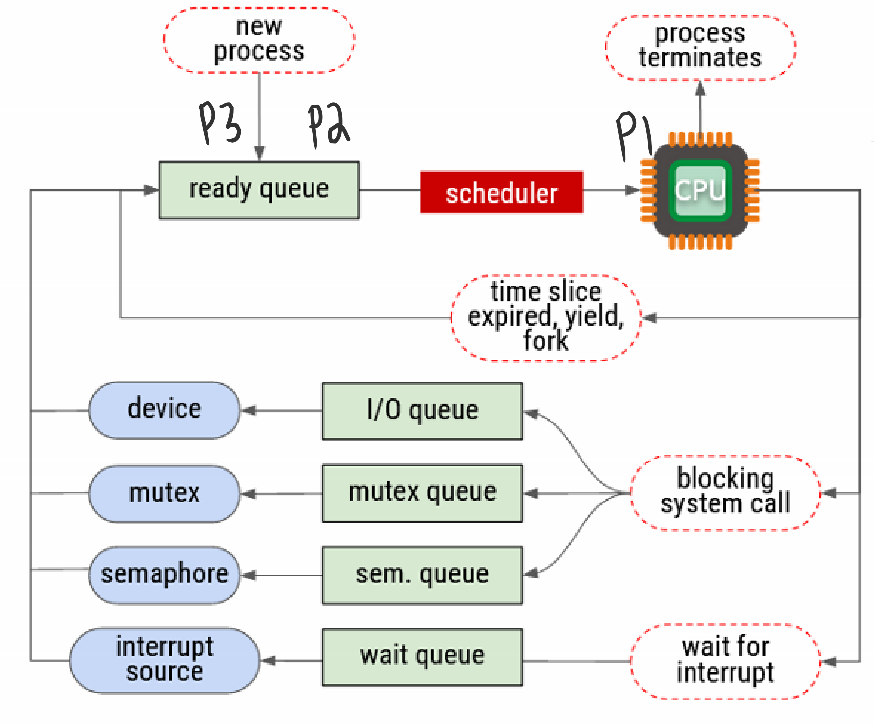
Preemptive time-sharing

$\boxed{\text{Time slice } \textbf{(Quantum)} : \textbf{2}}$

Process	**Arrival**	**Burst**
P1	0	~~4~~ 2
P2	0	4
P3	0	4

Current time: 2





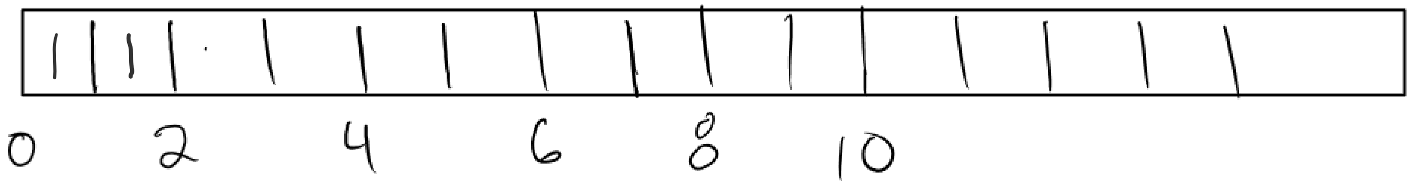
RR

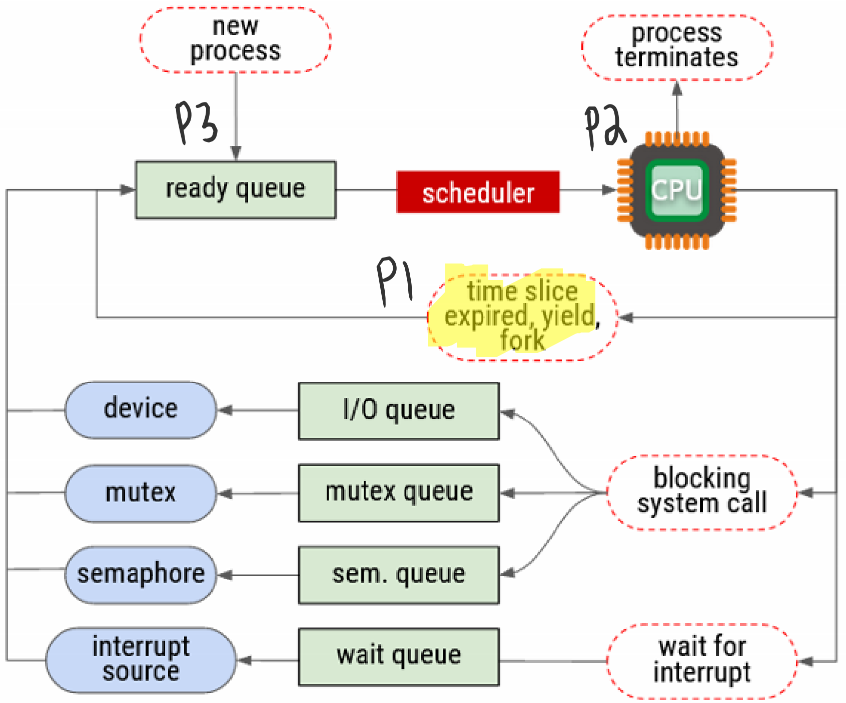
Preemptive time-sharing

Time slice **(Quantum) : 2**

Process	**Arrival**	**Burst**
P1	0	~~4~~ 2
P2	0	4
P3	0	4

Current time: 2



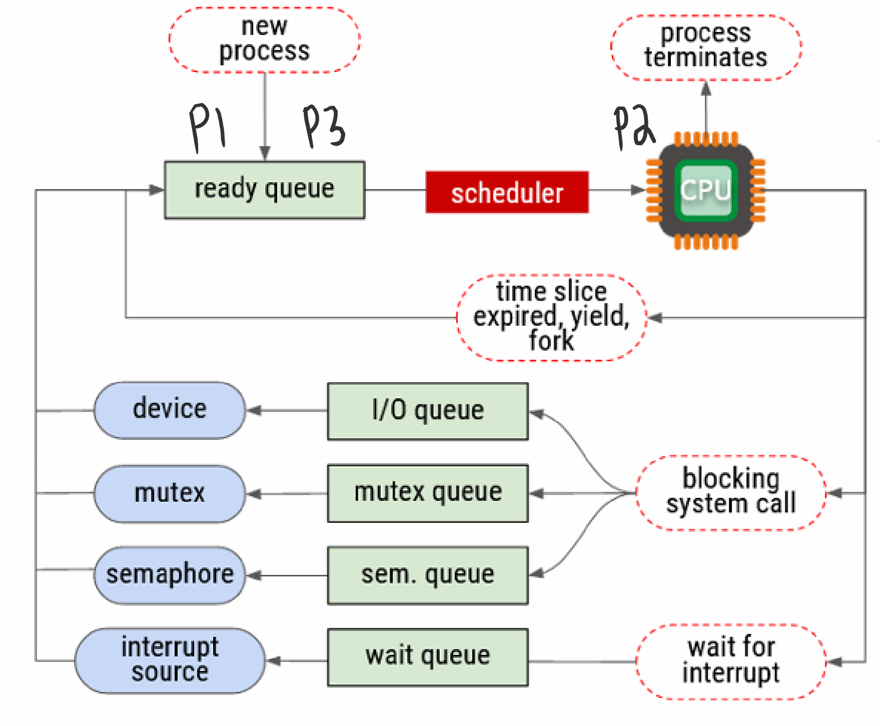


RR

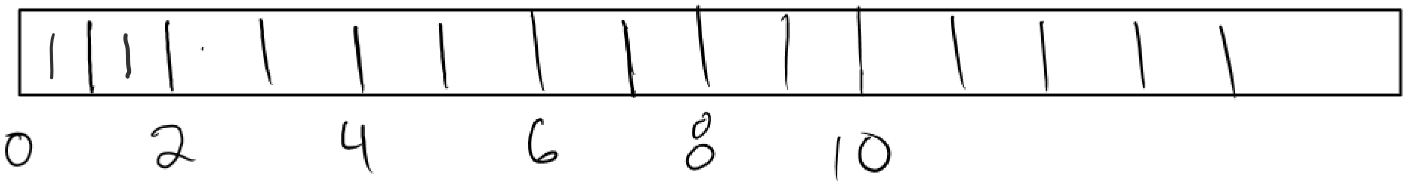
Preemptive time-sharing

Time slice **(Quantum) : 2**

Process	**Arrival**	**Burst**
P1	0	~~4~~ 2
P2	0	4
P3	0	4



Current time: 2



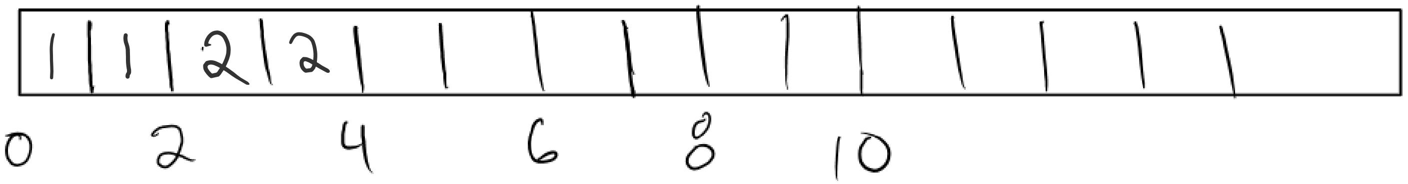
RR

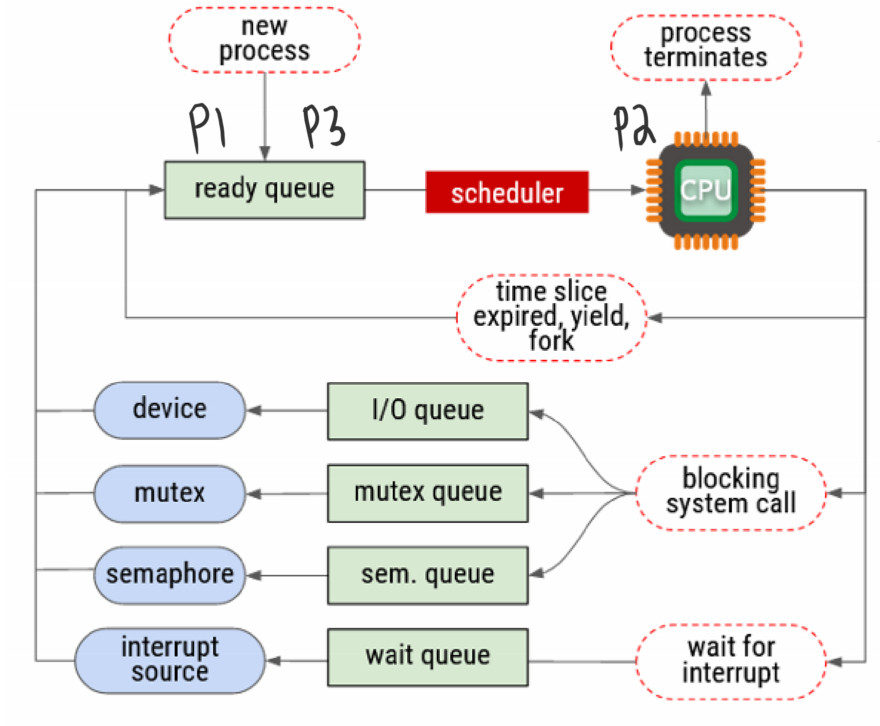
Preemptive time-sharing

Time slice (Quantum) : 2

Process	Arrival	Burst
P1	0	~~4~~ 2
P2	0	~~4~~ 2
P3	0	4

Current time: 4





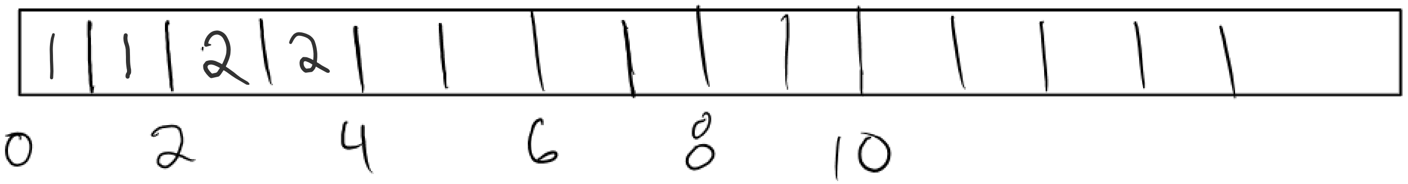
RR

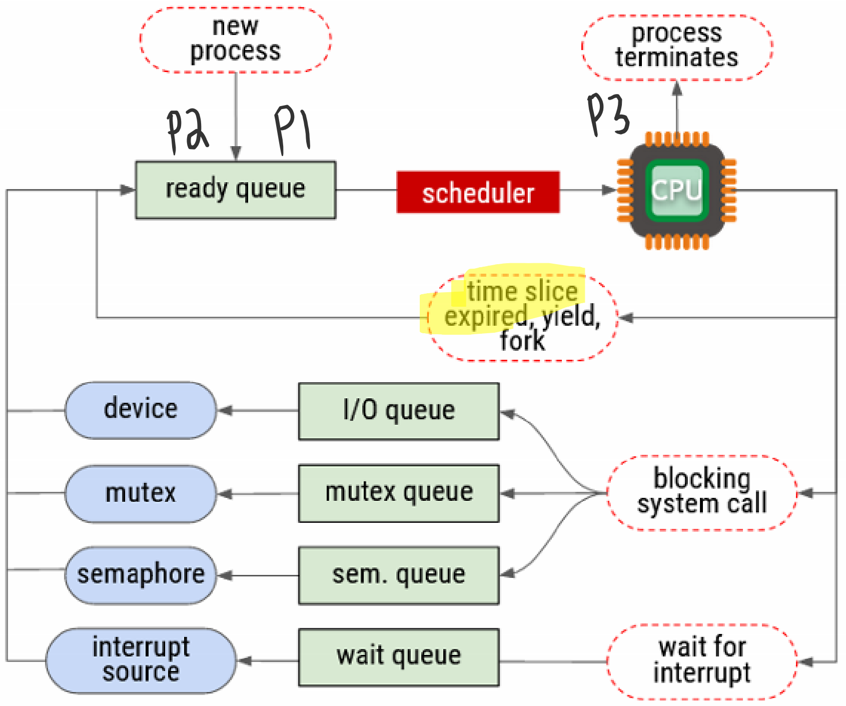
Preemptive time-sharing

Time slice **(Quantum) : 2**

Process	**Arrival**	**Burst**
P1	0	~~4~~ 2
P2	0	~~4~~ 2
P3	0	4

Current time: 4





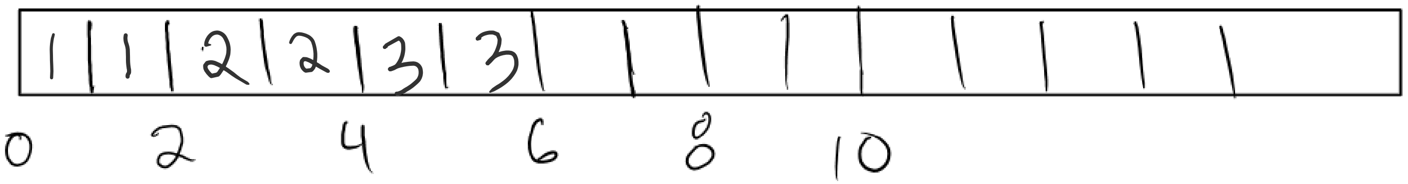
RR

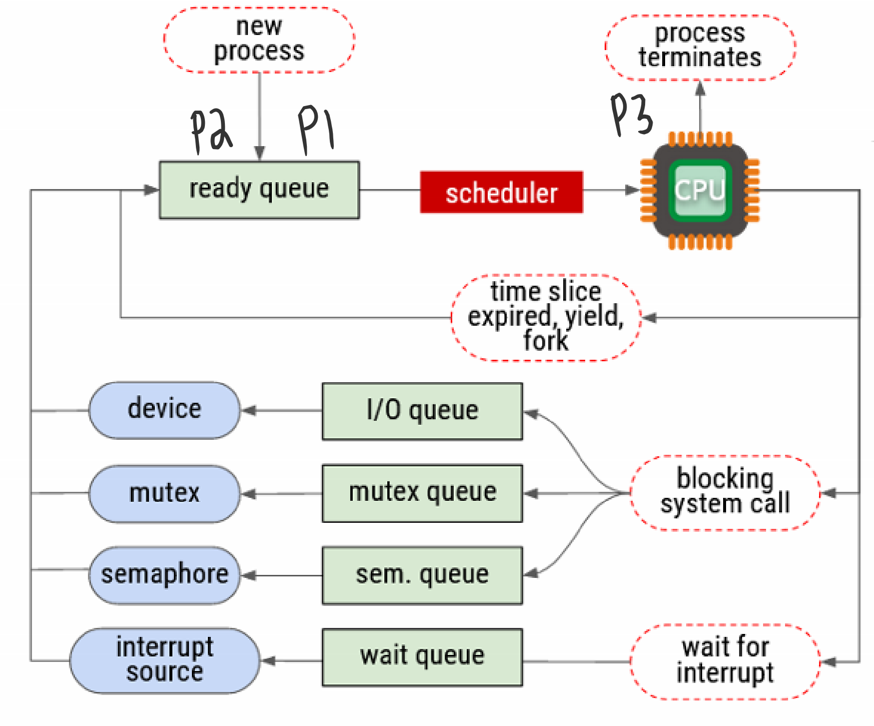
Preemptive time-sharing

Time slice **(Quantum) : 2**

Process	**Arrival**	**Burst**
P1	0	~~4~~ 2
P2	0	~~4~~ 2
P3	0	~~4~~ 2

Current time: 6





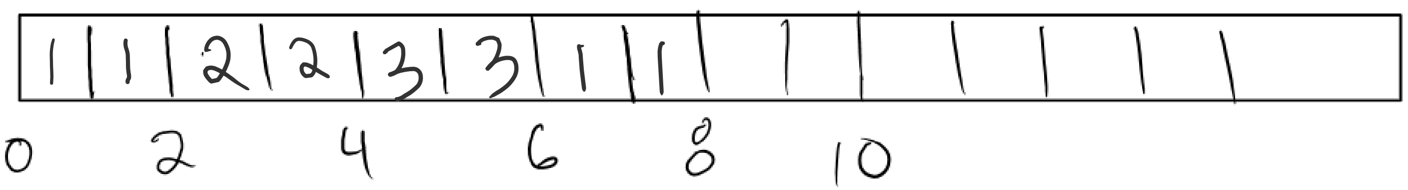
RR

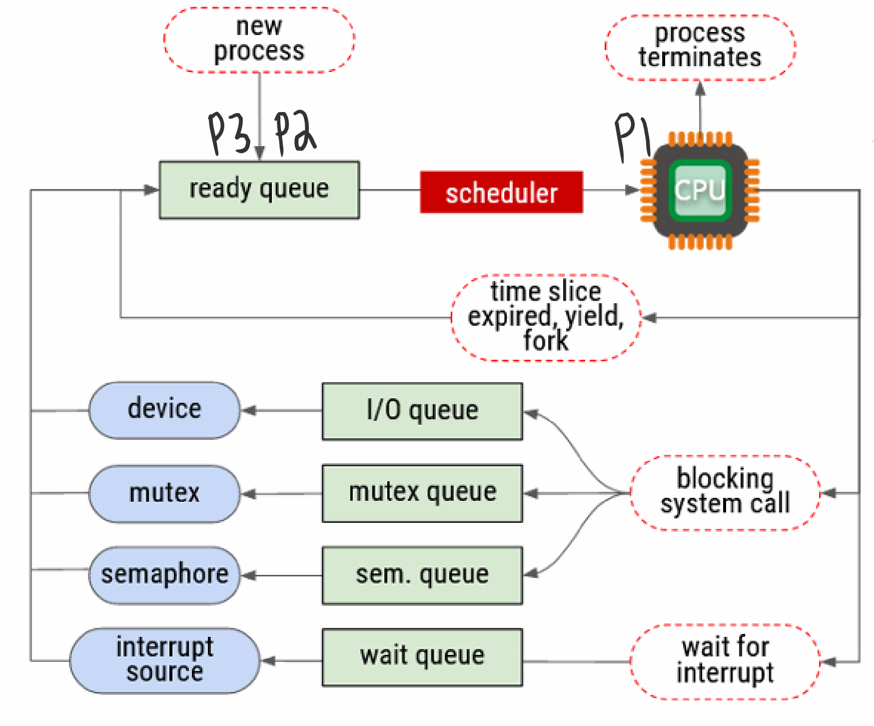
Preemptive time-sharing

Time slice **(Quantum) : 2**

Process	**Arrival**	**Burst**
P1	0	~~4~~ ~~2~~
P2	0	~~4~~ 2
P3	0	~~4~~ 2

Current time: 8





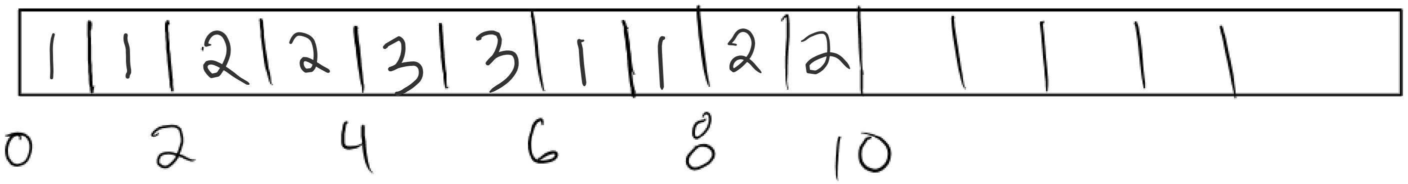
<u>RR</u>

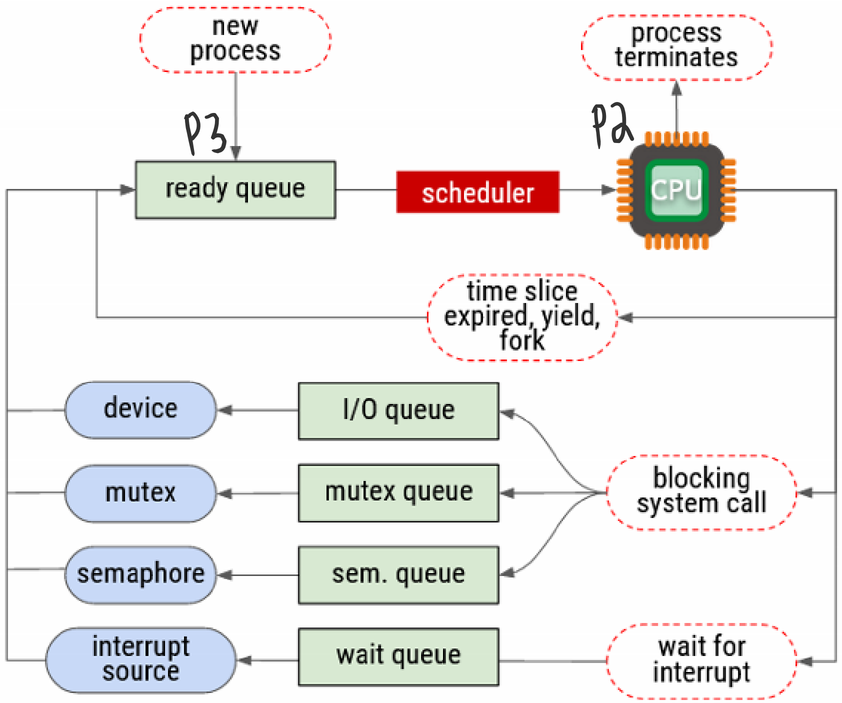
Preemptive time-sharing

Time slice **(Quantum) : 2**

Process	**Arrival**	**Burst**
P1	0	~~4~~ 2
P2	0	~~4~~ ~~2~~
P3	0	~~4~~ 2

Current time: 10





RR Preemptive time-sharing

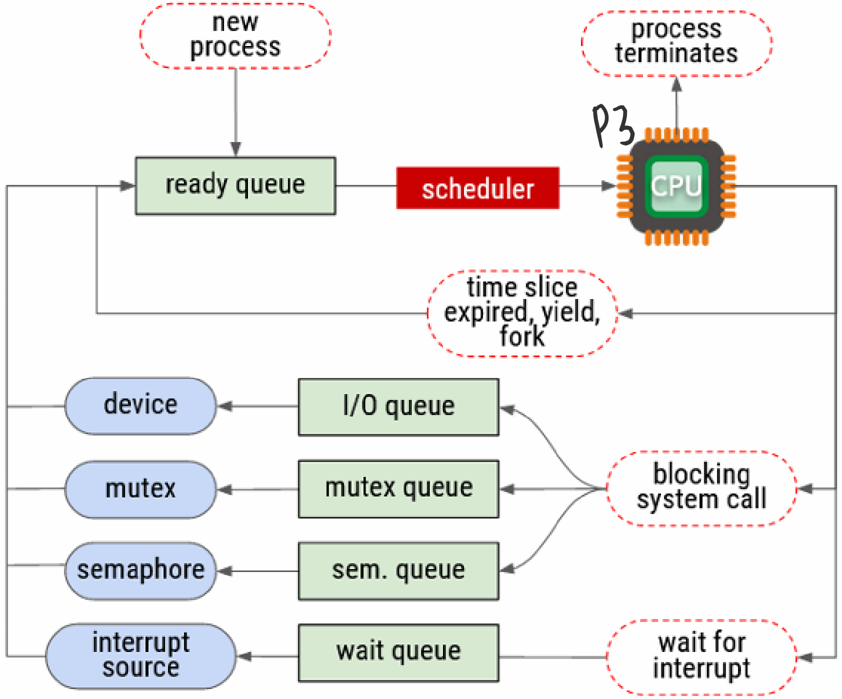
Time slice **(Quantum) : 2**

Process	**Arrival**	**Burst**
P1	0	~~4~~ ~~2~~
P2	0	~~4~~ ~~2~~
P3	0	~~4~~ ~~2~~

Current time: 10

1	1	2	2	3	3	1	1	2	2	3	3			

0 2 4 6 8 10



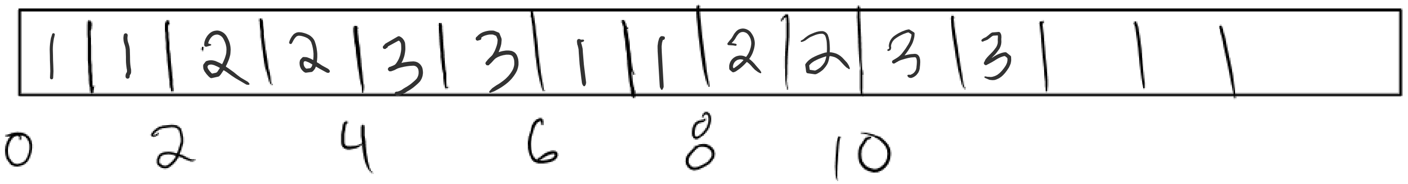
RR

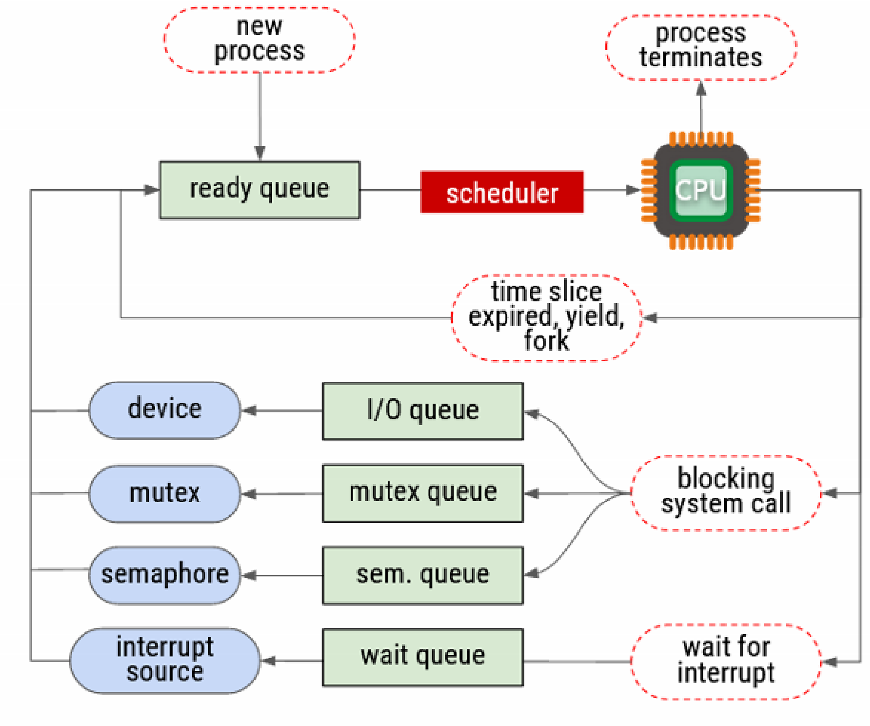
Preemptive time-sharing

Time slice **(Quantum) : 2**

Process	**Arrival**	**Burst**
P1	0	~~4~~ ~~2~~
P2	0	~~4~~ ~~2~~
P3	0	~~4~~ ~~2~~

Current time: 10





RR scheduling , Another Example

- construct a Gantt chart using **<u>quantum of 3 msec</u>**
- assume no I/O activity

Process	Remaining Burst
P1	6
P2	6
P3	3
P4	8
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Time: 0

Ready Queue:

0

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Process	Remaining Burst
P1	6
P2	6
P3	3
P4	8
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Time: 0

- P1 arrives
- P2 arrives

Ready Queue: P1, P2

0

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Process	Remaining Burst
P1	6
P2	6
P3	3
P4	8
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Time: 0

- P1 starts executing

Ready Queue: P2

1

0

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Time: 1

Process	Remaining Burst
P1	5
P2	6
P3	3
P4	8
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

- P3 arrives
- P1 continues executing

Ready Queue: P2, P3

1	1

0

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Time: 2

Process	Remaining Burst
P1	4
P2	6
P3	3
P4	8
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

- P4 arrives
- P1 continues executing

Ready Queue: P2, P3, P4

1	1	1

0

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Time: 3

Process	Remaining Burst
P1	3
P2	6
P3	3
P4	8
P5	2

- P1 is preempted by P2
- P5 arrives

!

Ready Queue: P3, P4

1	1	1	2

0 *3*

- **execution order**:

RR scheduling

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Time: 3

Process	Remaining Burst
P1	3
P2	6
P3	3
P4	8
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

- P1 is preempted by P2
- P5 arrives

!

Which enters the ready queue first?

Ready Queue: P3, P4

1	1	1	2

0 3

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Process	Remaining Burst
P1	3
P2	6
P3	3
P4	8
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Time: 3

- P1 is preempted by P2
- P5 arrives

Which enters the ready queue first? For this example, we will say pre-empted processes have priority

Ready Queue: P3, P4, P1, P5

1	1	1	2

0 *3*

- **execution order**:

RR scheduling

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Time: 6

- P2 is preempted by P3

Process	Remaining Burst
P1	3
P2	3
P3	3
P4	8
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Ready Queue: P4, P1, P5, P2

1	1	1	2	2	2	3

0 *3* *6*

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Time: 6

Process	Remaining Burst
P1	3
P2	3
P3	3
P4	8
P5	2

- P2 is preempted by P3

Ready Queue: P4, P1, P5, P2

1	1	1	2	2	2	3																		

0 *3* *6*

- **execution order**:

RR scheduling

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Time: 9

Process	Remaining Burst
P1	3
P2	3
P3	0
P4	8
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

- P3 finishes
- P4 starts executing

Ready Queue: P1, P5, P2

1	1	1	2	2	2	3	3	3	4

0 3 6 9

- **execution order**:

RR scheduling

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Process	Remaining Burst
P1	3
P2	3
P3	0
P4	5
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Time: 12

- P4 is preempted by P1

Ready Queue: P5, P2, P4

1	1	1	2	2	2	3	3	3	4	4	4	1												

0 *3* *6* *9* *12*

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Process	Remaining Burst
P1	0
P2	3
P3	0
P4	5
P5	2

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Time: 15

- P1 finishes
- P5 starts executing

Ready Queue: P2, P4

1	1	1	2	2	2	3	3	3	4	4	4	1	1	1	5									

0 *3* *6* *9* *12* *15*

- **execution order**:

RR scheduling

- construct a Gantt chart using **<u>quantum of 3 msec</u>**
- assume no I/O activity

<u>Time: 17</u>

- P5 finishes
- P2 starts executing

Process	Remaining Burst
P1	0
P2	3
P3	0
P4	5
P5	0

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Ready Queue: P4

1	1	1	2	2	2	3	3	3	4	4	4	1	1	1	5	5	2							
0			*3*			*6*			*9*			*12*			*15*		*17*							

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Time: 20

- P2 finishes
- P4 starts executing

Process	Remaining Burst
P1	0
P2	0
P3	0
P4	5
P5	0

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Ready Queue:

1	1	1	2	2	2	3	3	3	4	4	4	1	1	1	5	5	2	2	2	4				

0 3 6 9 12 15 17 20

- **execution order**:

RR scheduling

- construct a Gantt chart using **<u>quantum of 3 msec</u>**
- assume no I/O activity

<u>Time: 25</u>

- P4 finishes

Process	Remaining Burst
P1	0
P2	0
P3	0
P4	0
P5	0

Process	Arrival	Burst	Start	Finish
P1	0	6		
P2	0	6		
P3	1	3		
P4	2	8		
P5	3	2		

Ready Queue:

1	1	1	2	2	2	3	3	3	4	4	4	1	1	1	5	5	2	2	2	4	4	4	4	4

0 *3* *6* *9* *12* *15* *17* *20* *25*

- **execution order**:

- construct a Gantt chart using **quantum of 3 msec**
- assume no I/O activity

Time: 25

Process	Remaining Burst
P1	0
P2	0
P3	0
P4	0
P5	0

Process	Arrival	Burst	Start	Finish
P1	0	6	0	15
P2	0	6	3	20
P3	1	3	6	9
P4	2	8	9	25
P5	3	2	15	17

Ready Queue:

1	1	1	2	2	2	3	3	3	4	4	4	1	1	1	5	5	2	2	2	4	4	4	4	4

0 *3* *6* *9* *12* *15* *17* *20* *25*

- **execution order**: P1, P2, P3, P4, P1, P5, P2, P4

Shortest-job-first scheduling (SJF)

- another non-preemptive scheduling algorithm
  - applicable to batch systems, where job length (expected execution time) is known in advance
  - note: could be modified to be preemptive (eg. preemption when new job arrives, or existing one unblocks)
- when the CPU is available, it is assigned to the shortest job
  - shortest = shortest execution time
  - ties are resolved using FCFS
- SJF is similar to FCFS, but ready queue is sorted based on submitted estimate of execution time

SJF scheduling

- construct a Gantt chart
- assume no I/O activity

Process	Arrival	Burst	Start	Finish	Turnaround	Waiting
P1	0	6				
P2	0	6				
P3	1	3				
P4	2	8				
P5	3	2				

SJF scheduling

- construct a Gantt chart
- assume no I/O activity

Process	Arrival	Burst	Start	Finish	Turnaround	Waiting
P1	0	6	0	6	6	0
P2	0	6	11	17	17	11
P3	1	3	8	11	10	7
P4	2	8	17	25	23	15
P5	3	2	6	8	5	3

1	1	1	1	1	1	5	5	3	3	3	2	2	2	2	2	2	4	4	4	4	4	4	4	4

0 6 8 11 17 25

- execution order: P1, P5, P3, P2, P4
- average wait time: 7.2 units, context switches: 5

Shortest-remaining-time-next scheduling (SRTN)

- preemptive version of SJF
- next job is picked based on remaining time
  - remaining time = <expected execution time> – <time already spent on CPU>
- SRTN is similar to RR
  - but ready queue is a priority queue, 'sorted' based on remaining time
  - preemption happens as a result of adding a job

SRTN scheduling

- fill out the table & construct a Gantt chart
- assume no I/O activity

Process	Arrival	Burst	Start	Finish	Turnaround	Waiting
P1	0	6				
P2	0	6				
P3	1	3				
P4	2	8				
P5	3	2				

SRTN scheduling

- fill out the table & construct a Gantt chart
- assume no I/O activity

Process	Arrival	Burst	Start	Finish	Turnaround	Waiting
P1	0	6	0	11	11	5
P2	0	6	11	17	17	11
P3	1	3	1	4	3	0
P4	2	8	17	25	23	15
P5	3	2	4	6	3	1

1	3	3	3	5	5	1	1	1	1	1	2	2	2	2	2	2	4	4	4	4	4	4	4	4

0 1 4 6 11 17 25

- execution order: P1, P3, P5, P1, P2, P4
- average wait time: 6.4 units, context switches: 6

Simulation loop

Possible general structure:

```
curr_time = 0
while(1) {
    ... do whatever should happen
        at time curr_time
    if simulation done break
    curr_time ++
}
```

From Dr. Pavol Federl's CPSC 457 Slides (13b-scheduling)

Simulation loop

Possible FCFS scheduling simulation loop structure:

```
curr_time = 0
jobs_remaining = size of job queue
while(1) {
    if jobs_remaining == 0 break
    if process in cpu is done
        mark process done
        set CPU idle
        jobs_remaining --
        continue
    if a new process arriving
        add new process to RQ
        continue
    if cpu is idle and RQ not empty
        move process from RQ to CPU
        continue
    execute one burst of job on CPU
    curr_time ++
}
```

?

From Dr. Pavol Federl's CPSC 457 Slides (13b-scheduling)

Simulation loop

Possible FCFS scheduling simulation loop structure:

See fcfsSimulationLoop.cpp for code which follows this pseudocode

```
curr_time = 0
jobs_remaining = size of job queue
while(1) {
    if jobs_remaining == 0 break
    if process in cpu is done
        mark process done
        set CPU idle
        jobs_remaining --
        continue
    if a new process arriving
        add new process to RQ
        continue
    if cpu is idle and RQ not empty
        move process from RQ to CPU
        continue
    execute one burst of job on CPU
    curr_time ++
}
```

?

From Dr. Pavol Federl's CPSC 457 Slides (13b-scheduling)

FCFS Simulation

Process	Arrival	Burst
P1	1	10
P2	3	5
P3	5	3

Demos

Two possible ways to implement FCFS scheduling simulation:

- fcfsSimulationLoop.cpp
- fcfsEmmanuel.cpp