University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

Operating Systems Scheduling

Dr. Vivek Nallur (vivek.nallur@ucd.ie)

Process Scheduling

Burst Cycles

- Process execution typically consists of a cycle of two states
  - CPU burst
    - A period of CPU execution with no I/O usage
  - I/O burst
    - wait for I/O signal

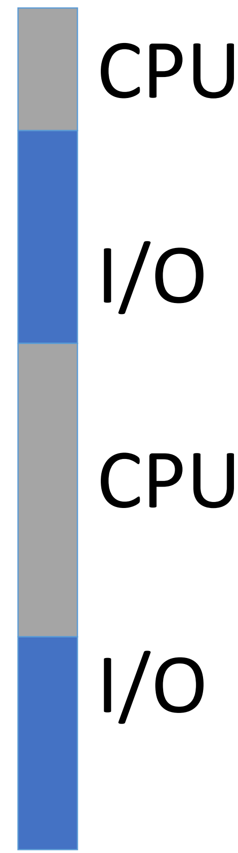
Burst Cycle Example

```
fscanf(file1,"%i,%i",&a,&b)



c=a+b;
d=(a*b)-c;
fprintf(file2,”%i,%i\n”,c,d);

...
```



Principle of multiprogramming

- Mix the **CPU** and **I/O** bursts of different processes to increase utilisation

- When one process is waiting for I/O we execute a process that is not

Scheduling

- When a process finishes or waits for I/O, what process should we execute next?

- Difficult choice, there will always be more processes than CPUs

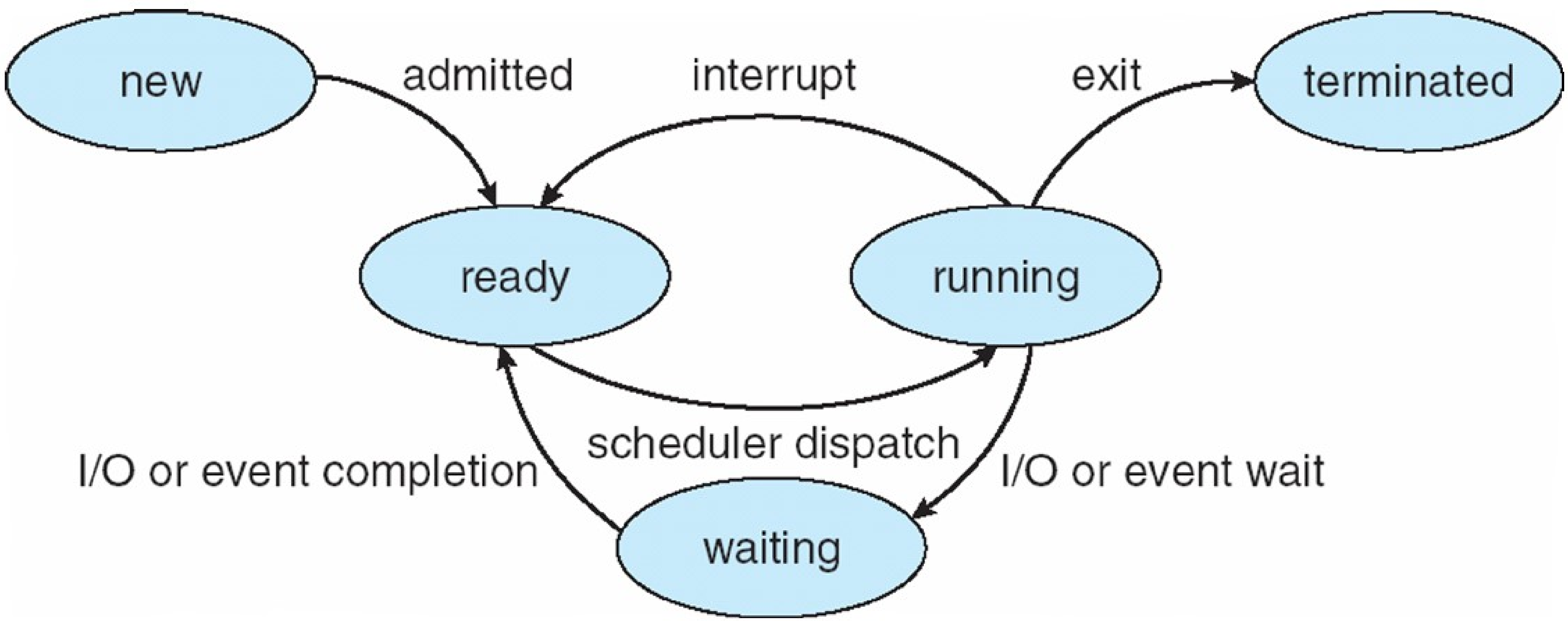
- The part of the operating system that makes this decision is called the **scheduler**

- The **scheduler's** decision is based on a scheduling algorithm

Threads or Processes

- In an OS that supports kernel threads, these are the **smallest unit**s of scheduling
- Most issues that apply to process scheduling also apply to thread scheduling

Process States



Levels of Scheduling

- Long-term scheduler
  - High level
- Medium-term scheduler
  - Mid level
- Short-term scheduler
  - Low level

Long-Term Scheduler

- This controls the pool of processes admitted to compete for system resources

- A program becomes a process once selected by the long-term scheduler, and it is added to the ready queue

Long-Term Scheduler

- The long-term scheduler controls the **degree** of multiprogramming
- The more processes admitted, the smaller the **percentage** of time that each process can be executed

Medium-Term Scheduler

- Selects what processes are kept in memory, actively competing for **CPU acquisition**
- The medium-term scheduler acts as a buffer, suspending and resuming processes to fine tune the **system load**

Short-term Scheduler

- Selects what process is assigned next to the CPU

- Only selects from processes in the ready queue

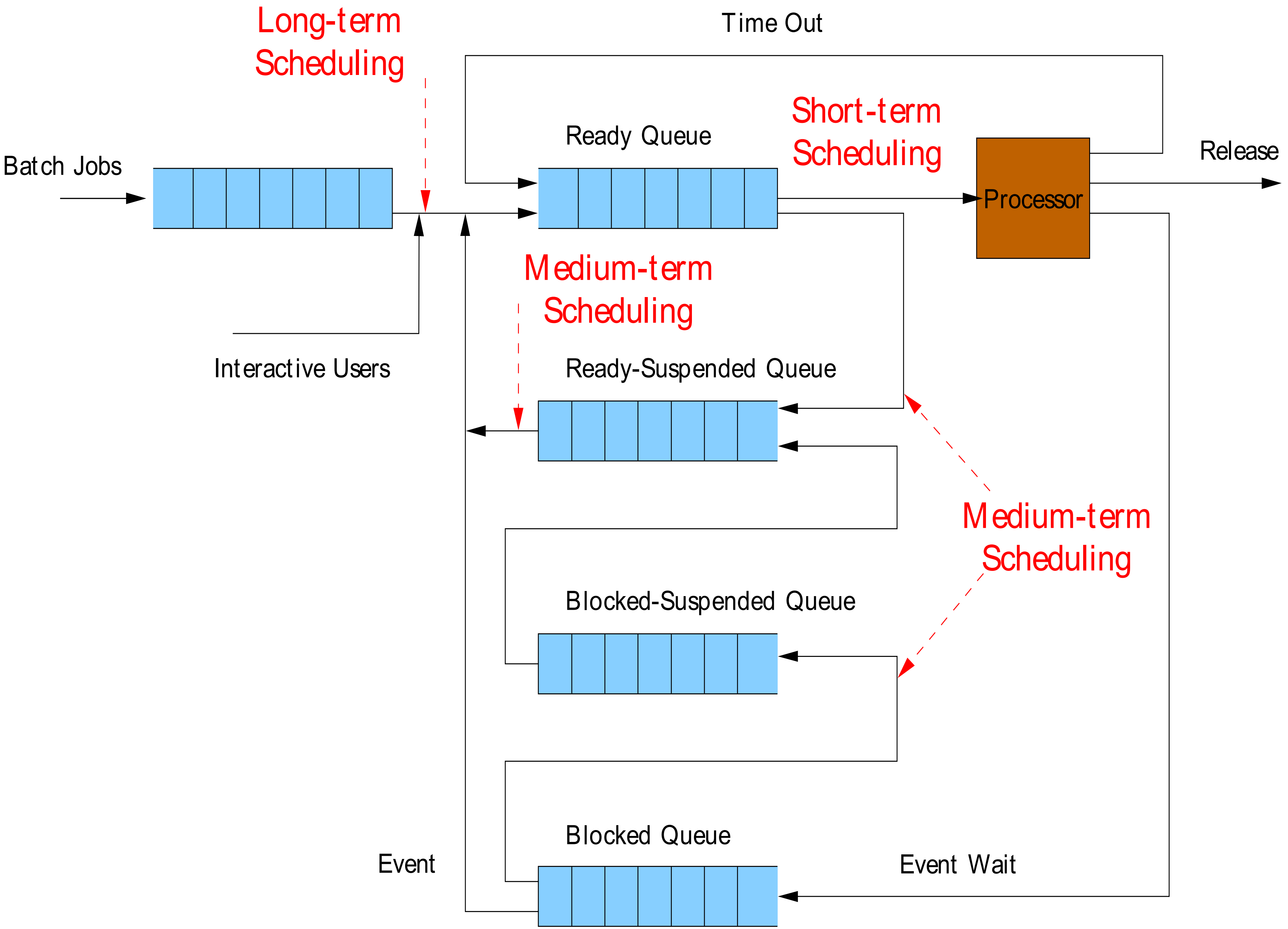
Scheduling Queues

- Ready queue:
  - Queue of all processes that are **eligible** to be scheduled
- Ready-Suspended queue:
  - Queue of all processes that are **ready** to be execute but have been **excluded** by the medium-term scheduler

Scheduling Queues

- Blocked queue:
  - Queue of all processes that are **waiting** for an event before going to ready queue

- Blocked-Suspended queue:
  - Queue of all processes that are **waiting** for an event and have been **excluded** by the medium-term scheduler


Long-term
Scheduling
Time Out
Short-term
Scheduling
Ready Queue
Batch Jobs
Release
Processor
Medium-term
Scheduling
Interactive Users
Ready-Suspended Queue
Medium-term
Scheduling
Blocked-Suspended Queue
Blocked Queue
Event
Event Wait

Dispatcher

- When the short-term scheduler selects the next process, the dispatcher routine gives it control of the CPU

- It must be as fast as possible (low dispatch latency), since it is invoked during every process switch

Dispatcher Functions

- Switch context
  - Store relevant data in the PCB of running process
  - Swap in PCB contents of process to be run
- Jump to the right location in the program to (re)start it, switching to user/kernel mode if required

Dispatch Latency

- Before the process can actually be dispatched, it must go through a **conflicts** phase;
  - Acquisition of resources needed by new process to execute
  - Preemption of running process resources if they should only be held while running

Non-preemptive Scheduling

- Some scheduling decisions take place when a process leaves the processor **voluntarily**

- If scheduling takes place under these circumstances: non-preemptive scheduling

Nonpreemptive Scheduling Features

- Errant processes can **block** the system
- Short processes can experience long **delays**, if long processes are running non-preemptively
- Time from process submission to process completion is quite predictable without preemption

Preemptive Scheduling

- Preemptive scheduling is when a process is **forced** to leave the processor it is running in, in order to give it to another process

- Malicious or errant processes can be removed from the CPU

- Improved response times are possible:
  - Important for interactive systems, time-sharing
  - Necessary for soft real-time systems (but hard real-time systems do not use preemption)

Scheduling Objectives and Criteria

Scheduling Objectives

- What does the scheduler consider in its decisions?
- Different schedulers will have different **goals** and policies

Scheduling Objectives Examples

- Maximise processor **utilisation**
- Maximise **throughput**
  - Processes completed per unit of time
- Minimise waiting time in the ready queue

Scheduling Objectives Examples

- Minimise response time (latency)
  - Waiting time in the ready queue for the **first** acquisition of the CPU
- Minimise turnaround
  - Total waiting time + total execution time
- Complete processes by given deadlines

Scheduling Objectives

- All of these goals cannot be met at the same time

- Operating scheduling algorithms generally focus on a subset of the list

Scheduling Objectives

- All scheduling strategies should achieve the following:
  - **Policy enforcement**
    - Guarantee that the system scheduling policy is actually carried out
  - **Fairness and Balance**
    - No process is starved
    - Similar processes are treated the same

Scheduling Objectives

- All scheduling strategies should achieve the following:
  - **Predictability**
    - Under a similar load, the same process should run in the approximately same amount of time
  - **Scalability**
    - Graceful performance degradation under heavy loads

Process Behaviour

- To realise its scheduling objectives, the scheduler should also consider process **behaviour**

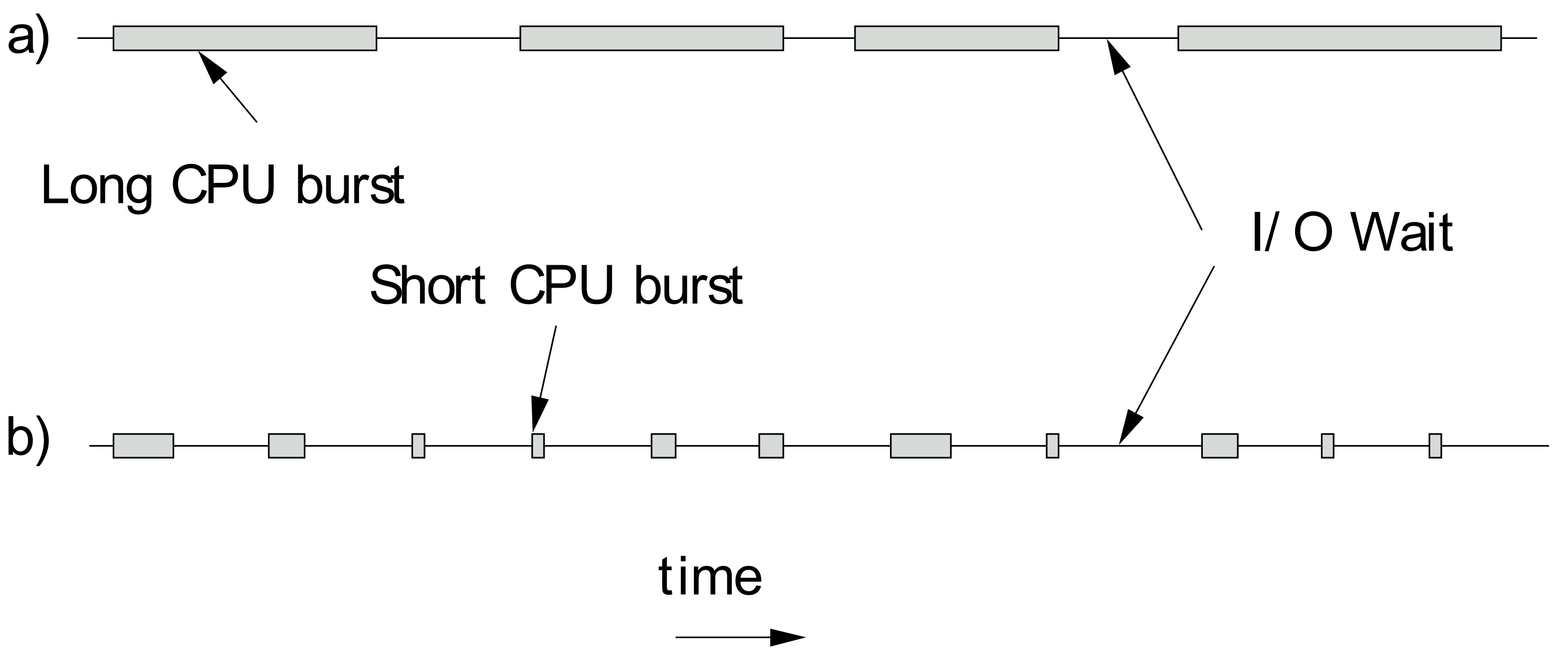
- Processes can classified according to their **burst pattern**

Process Behaviour

- Processor-bound (CPU-bound)
  - This is a process that tends to use **all available** CPU time

- I/O-bound
  - This are processes that tend to generate **I/O requests frequently** and release the processor

Process Behaviour

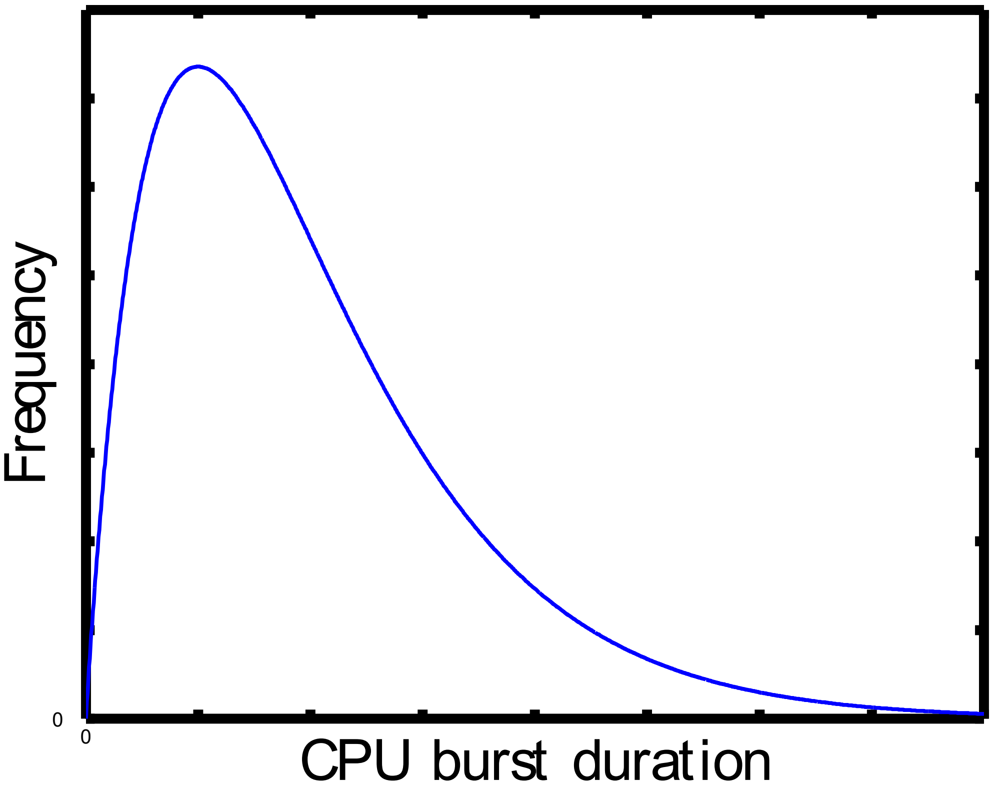


Typical Processes

- As processor technology improves faster than disk technology, most processes tend to be I/O-bound nowadays

Typical Processes

Considering all processes, the probability distribution of CPU burst times in any system typically looks like this



Process Classification

- Processes are classified according to their activity

- A batch process performs work with no user interaction
  - These are usually CPU-bound

Process Classification

- Processes are classified according to their activity

- An interactive process requires frequent user input
  - These are typically I/O-bound

Scheduling Objectives

- Typically scheduling objectives depend on process behaviour
- For batch systems **throughput** and **turnaround time** are important
- For interactive systems **response time** is important

Scheduling Algorithms

First Come First Served

Scheduling Algorithms

- The scheduling algorithm implements the system **policies** in order to achieve the scheduling **objectives**
- Its decisions may take into account
  - Preemptibility
  - Accountancy
  - Priorities

First Come First Served (FCFS)

- Simplest scheduling algorithm
- Processes are dispatched in the order they arrive in the ready queue
- Implements the First In First Out (FIFO) principle

First Come First Served (FCFS)

- If we have three processes in the queue, P_1, P_2 and P_3
- Their burst times are 24, 3 and 3 respectively

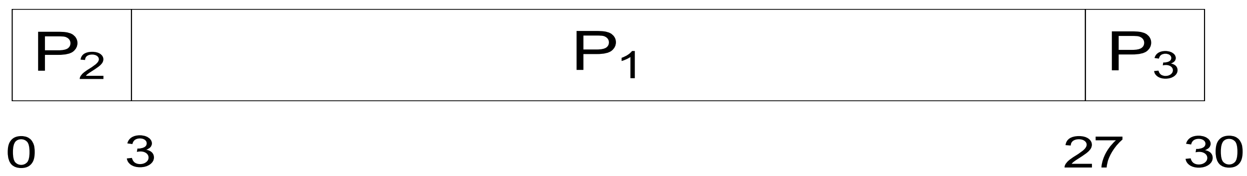
First Come First Served (FCFS)

- Arrival and finish times

- Waiting times:
  - P_1: 0
  - P_2: 24
  - P_3: 27
  - Average: 17

First Come First Served (FCFS)

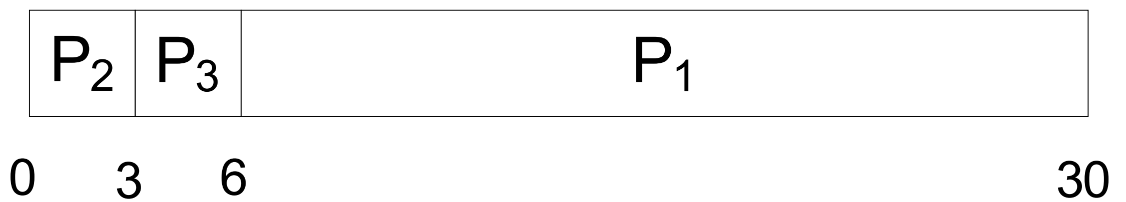
- Gantt chart: Arrival and finish times



- Waiting times:
  - P_1: 3
  - P_2: 0
  - P_3: 27
  - Average: 10

First Come First Served (FCFS)

- Arrival and finish times



- Waiting times:
  - P_1: 6
  - P_2: 0
  - P_3: 3
  - Average: 3

First Come First Served (FCFS)

- FCFS is good for CPU-bound processes

- But the waiting time is bad if long processes come first

- Better to have short ones first

Scheduling Algorithms

Shortest Job First

Shortest Job First

- Estimate the length of the **next** CPU burst for each process
- Always schedule the process with the **shortest** upcoming burst time next

Shortest Job First

- There are two versions of the algorithm:
  - Nonpreemptive – Shortest Job First (SJF)
  - Preemptive – Shortest Time Remaining First (STRF)

Shortest Job First

- Shortest Job First
  - When a process is chosen it must **complete** its CPU burst

- Shortest Time Remaining First
  - If a new process arrives, and its next burst time is shorter than the **remaining** time on the current process we preempt it

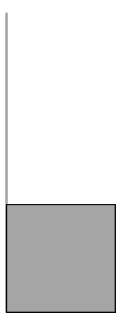
Example: Shortest Job First

	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4

Time: 0

Example: Shortest Job First

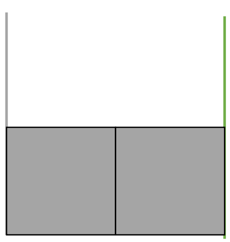
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 1

Example: Shortest Job First

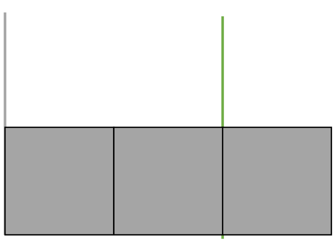
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 2

Example: Shortest Job First

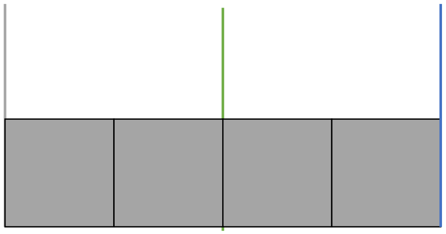
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 3

Example: Shortest Job First

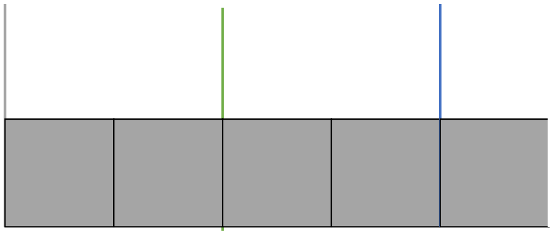
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 4

Example: Shortest Job First

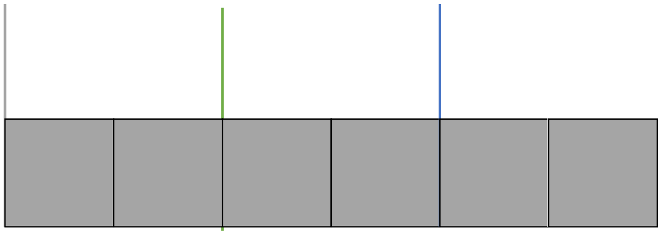
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 5

Example: Shortest Job First

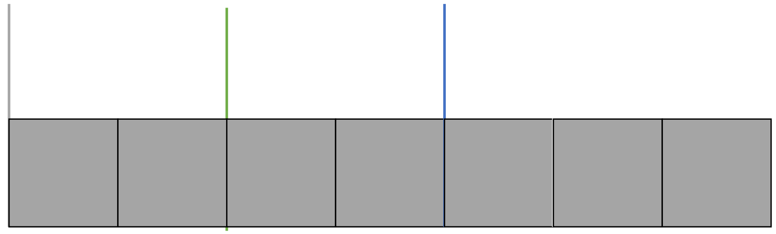
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 6

Example: Shortest Job First

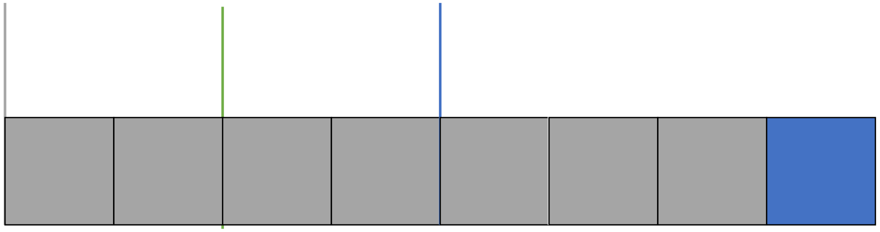
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 7

Example: Shortest Job First

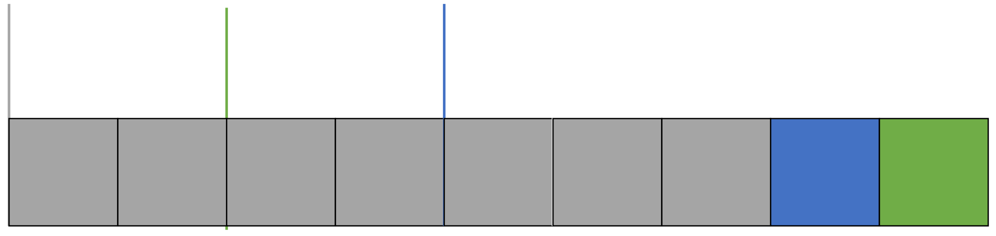
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 8

Example: Shortest Job First

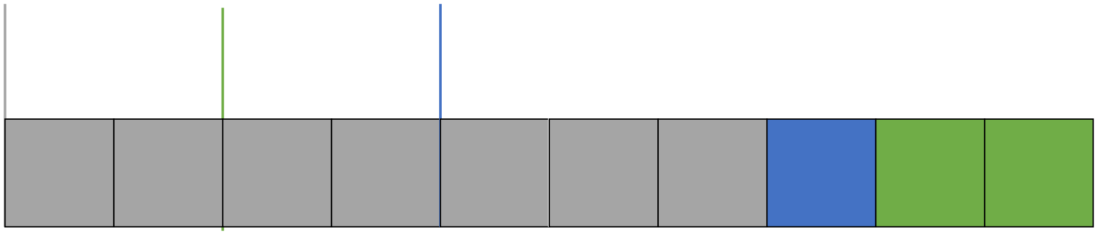
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 9

Example: Shortest Job First

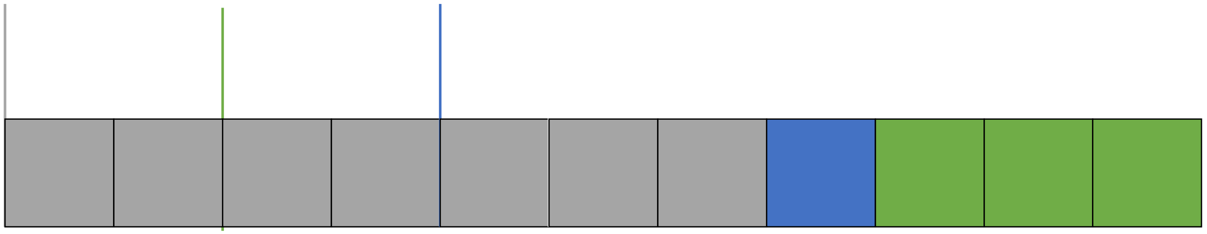
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 10

Example: Shortest Job First

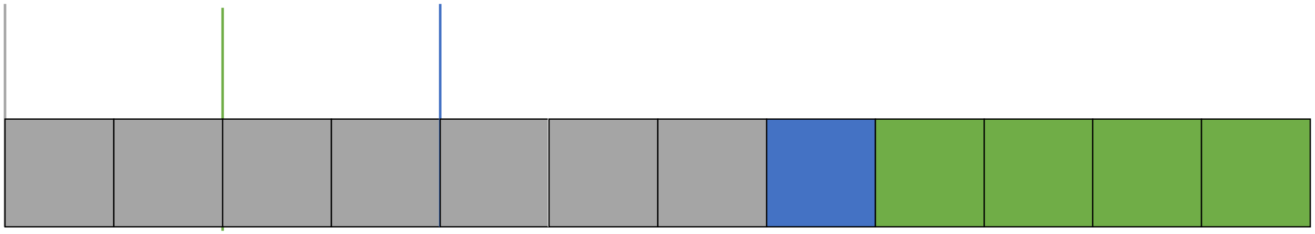
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 11

Example: Shortest Job First

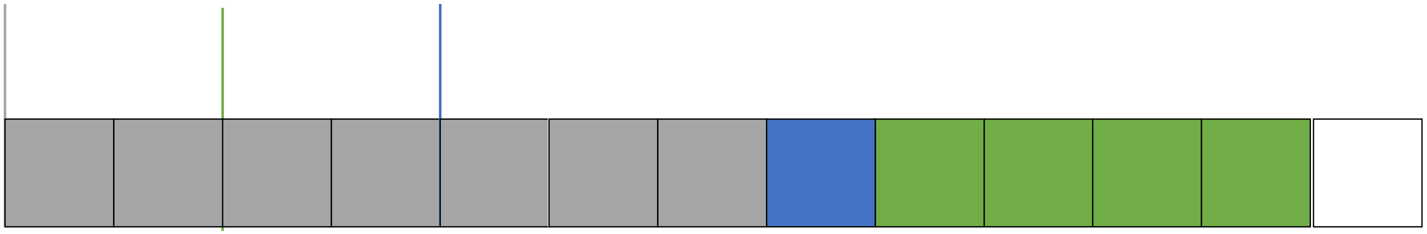
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 12

Example: Shortest Job First

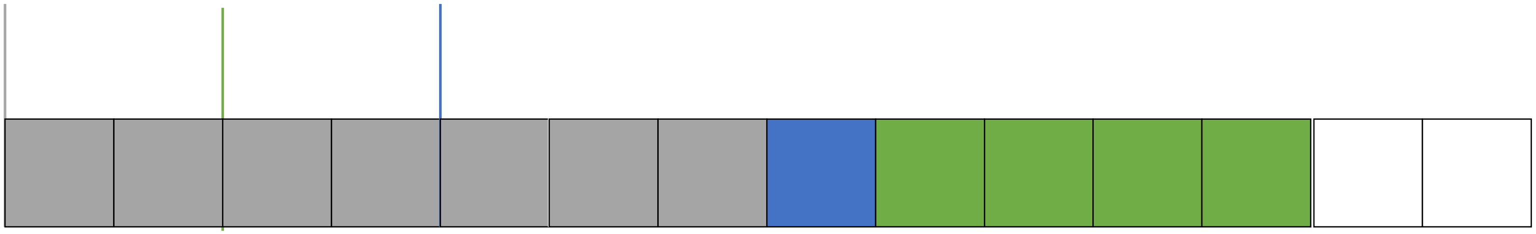
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 13

Example: Shortest Job First

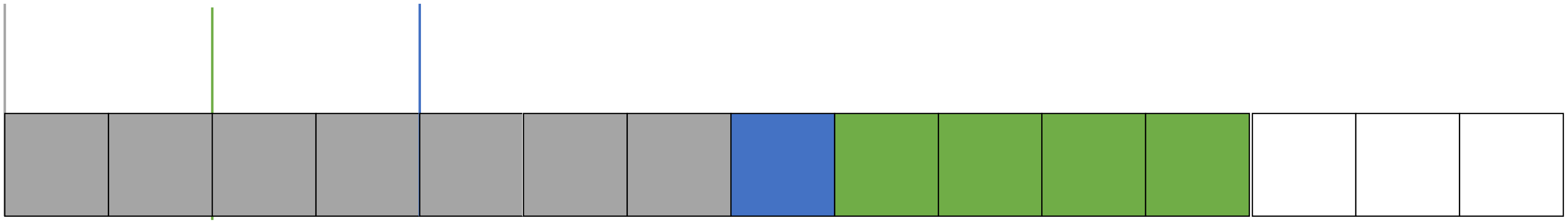
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 14

Example: Shortest Job First

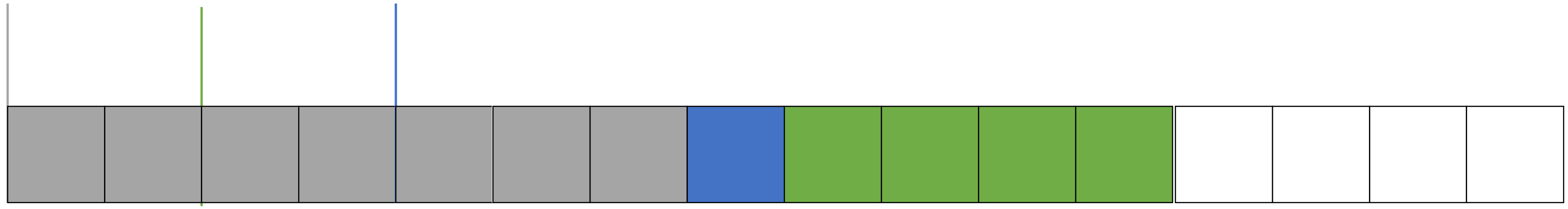
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 15

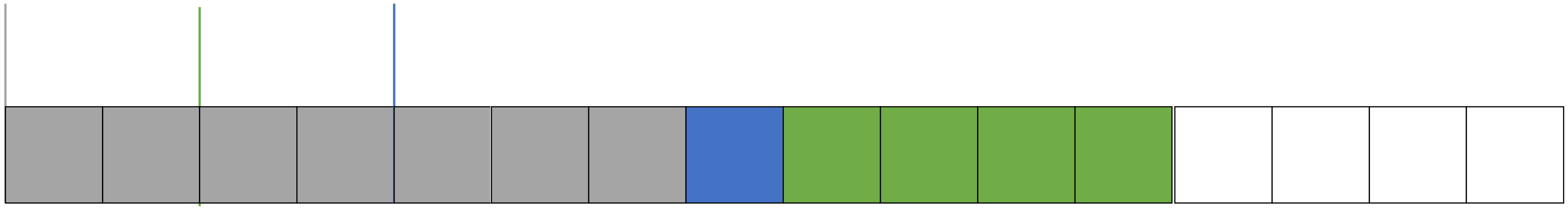
Example: Shortest Job First

	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 16

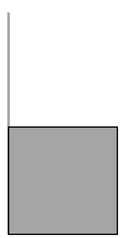
Average Waiting Time



- Waiting Time:
  - P_1: 0
  - P_2: 6
  - P_3: 3
  - P_4: 7
  - Average: 4

Ex: Shortest Time Remaining First

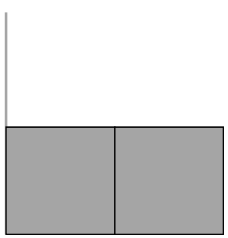
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 1

Ex: Shortest Time Remaining First

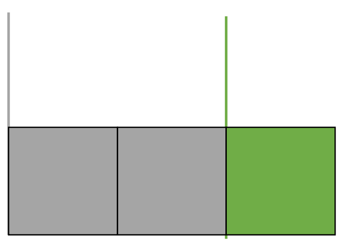
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 2

Ex: Shortest Time Remaining First

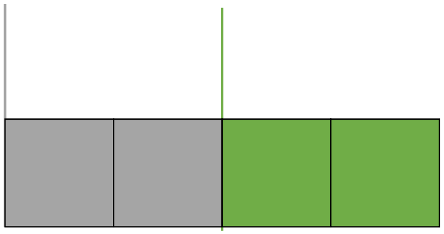
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 3

Ex: Shortest Time Remaining First

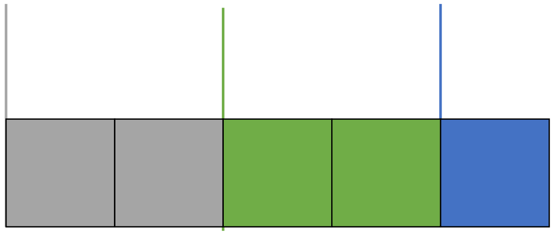
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 4

Ex: Shortest Time Remaining First

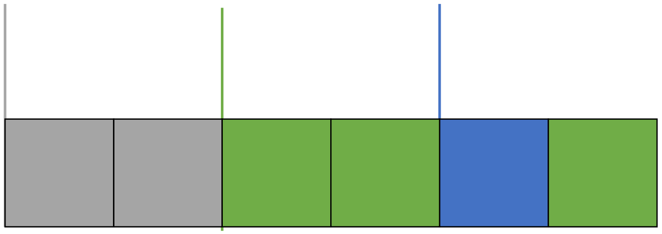
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 5

Ex: Shortest Time Remaining First

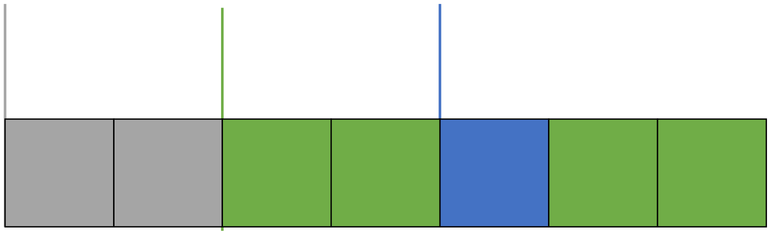
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 6

Ex: Shortest Time Remaining First

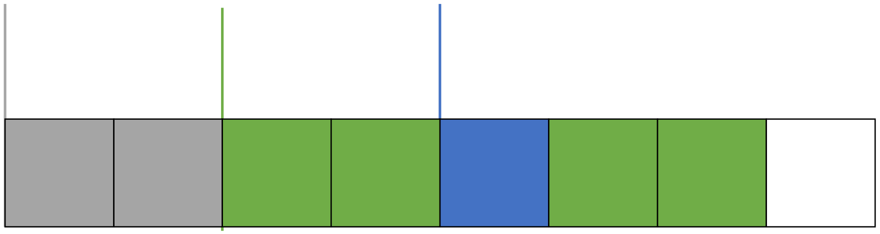
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 7

Ex: Shortest Time Remaining First

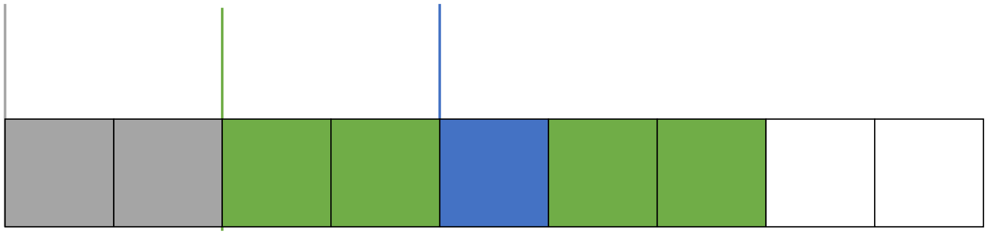
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 8

Ex: Shortest Time Remaining First

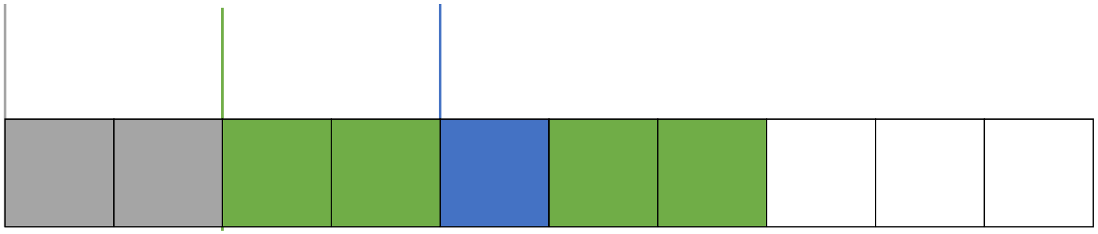
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 9

Ex: Shortest Time Remaining First

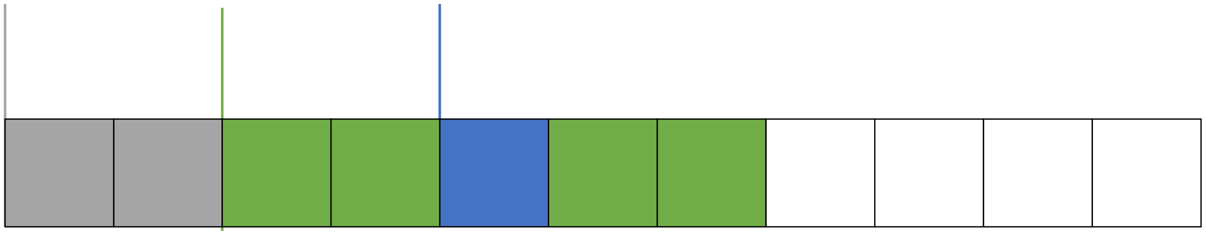
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 10

Ex: Shortest Time Remaining First

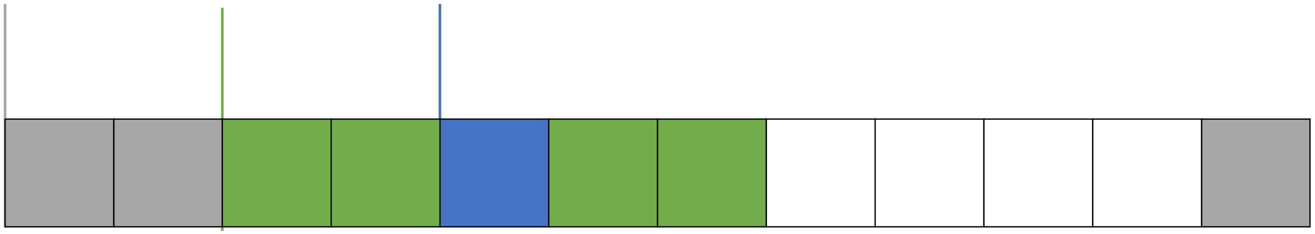
	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 11

Ex: Shortest Time Remaining First

	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 12

Ex: Shortest Time Remaining First

	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4

Time: 13

Ex: Shortest Time Remaining First

	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4

Time: 14

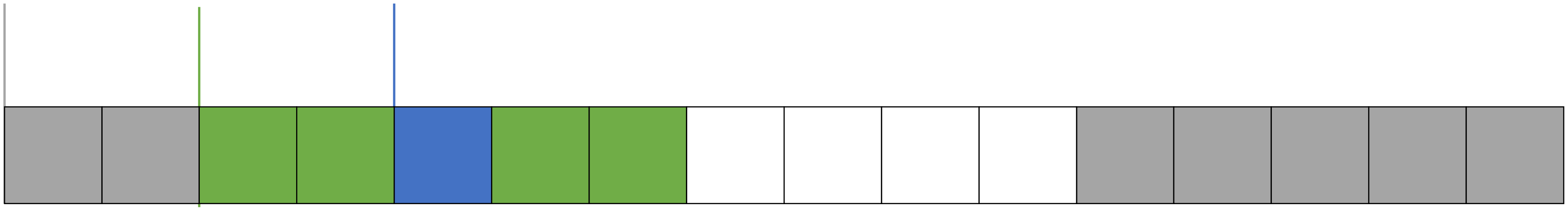
Ex: Shortest Time Remaining First

	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4

Time: 15

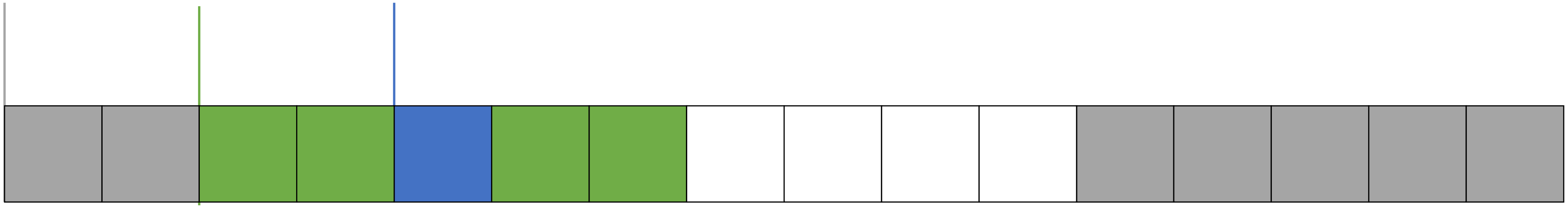
Ex: Shortest Time Remaining First

	P_1	P_2	P_3	P_4
Arrival Time	0	2	4	5
Burst Time	7	4	1	4



Time: 16

Average Waiting Time



- Waiting Time:
  - P_1: 9
  - P_2: 1
  - P_3: 0
  - P_4: 2
  - Average: 3

Shortest Job First

- Assuming all processes arrive at the same time, we can **prove** that non-preemptive shortest job first is optimal (the best possible)

Proof

- If there are n processes with CPU burst times $t_1, \ldots, t_n$ and they are scheduled $P_1, P_2, \ldots, P_n$,
- Waiting times are:
  - W_1: 0
  - W_2: t_1
  - W_3: $t^1 + t_2$
  - W_k: $t_1 + t_2 + \ldots + t_{k-1} + t_k$

Proof

- Waiting times are:
  - W_k: $t_1 + t_2 + \ldots + t_{k-1} + t_k$
- Average waiting time is:

$$w = \frac{\sum_{k=1}^{n} w_k}{n}$$

Proof

- w is minimum if all w_k are minimum

- Proof by Induction
  - w_1 is always minimum
  - w_2 is minimum if $t_1 \le t_{i,}$ where i = 2,...,n
  - w_k is minimum if $t_{k-1} \le t_{i,}$ where i = k,...,n

- Therefore minimum is when

$$t1 \le t2 \le \ldots \le tn$$

Burst Time Estimation

- Shortest job first is optimal, but we need to know the burst times before
- This is usually not possible
- These times must be estimated
  - If it is not accurate, we can preempt

Burst Time Estimation

- Next CPU burst is usually predicted as an **exponential average** of the measured lengths of previous CPU bursts

Exponential Average

- let t_n be the measured length of the n-th CPU burst
- let τ_n be the estimated length of the n-th CPU burst
- The estimated value for the next CPU burst is:

$$\tau_{n+1} = \alpha * t_n + (1-\alpha)\tau_n, \quad 0 \le \alpha \le 1$$

Exponential Average

- The value of α determines how the prediction works
  - If $\alpha = 0$ then $\tau_{n+1} = \tau_n$
    - Recent times have no effect
  - If $\alpha = 1$ then $\tau_{n+1} = t_n$
    - Only the most recent CPU burst is considered
- Usually α is 0.5

Scheduling Algorithms

Round Robin

Round Robin

- Processes are dispatched FIFO but only given a limited amount of CPU time
  - Called a **quantum** or **time slice**

- After this time has elapsed, the process is preempted and added to the end of the ready queue

Round Robin

- RR guarantees a **maximum waiting time** in the ready queue
  - if there are n processes in the queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once
  - No process waits more than $(n - 1)q$ time units
    - good for interactivity

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

$q = 20$

Time: 0

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

q = 20

Time: 20

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

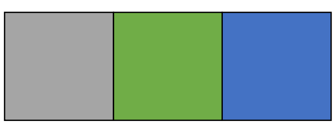
q = 20

Time: 37

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

q = 20

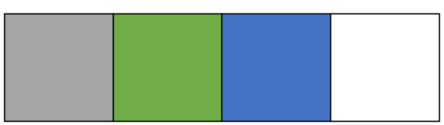


Time: 57

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

q = 20

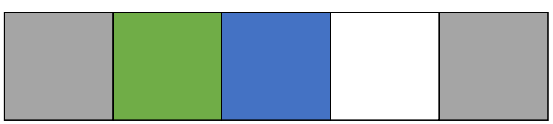


Time: 77

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

q = 20

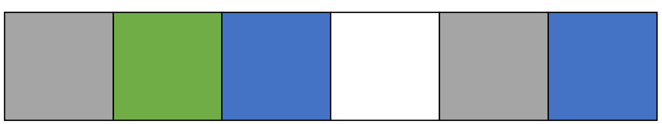


Time: 97

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

$q = 20$

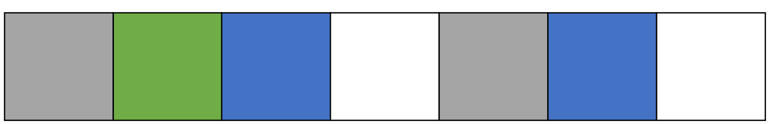


Time: 117

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

q = 20



Time: 121

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

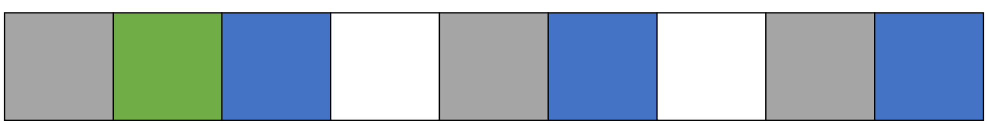
q = 20

Time: 134

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

q = 20

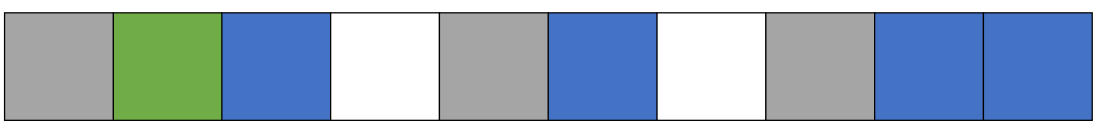


Time: 154

Ex: Round Robin

	P_1	P_2	P_3	P_4
Burst Time	53	17	68	24

q = 20



Time: 162

Performance

- Round robin performance heavily depends on the quantum size
- If q is large then it is just FCFS
- If q is small performance is better
  - But it cannot be too much smaller

Context Switch

- q must still be large when compared with context switch time, otherwise overhead is too high
- If the context switch time is `c`: we lose `100 × c/(c+q)` % of the CPU time doing switching (i.e., throughput decreases)
- In real systems the quantum is usually 10–100 ms, while the context switch is typically less than 10 µs (0.1% – 0.01%)

Turnaround Time in RR

- The average turnaround time also depends on q but it does not steadily improve as q increases

- In general, it improves if most processes finish their next CPU burst in one single quantum

Turnaround Time in RR Example

- If we have three processes each with bursts of 10 time units
  - `q` = 1: average turnaround is 29
  - `q` = 10: average turnaround is 20
- Usually RR has higher average turnaround than SJF but better response time

Scheduling Algorithms

Priority Scheduling

Priority Scheduling

- RR makes the implicit assumption that all processes are equally important, which is not reasonable in general

- In priority scheduling we schedule the most important processes first

Priority Scheduling

- A priority number (integer) is associated with each process
- Priorities quantify the relative importance of processes
  - Small numbers mean high priority
- The CPU is allocated to the process with the highest priority

Priority Scheduling Starvation

- Low priority processes might never execute
- Solution: increase priority as time progresses (**aging**)

Priority Scheduling

- Priority scheduling can be either preemptive or nonpreemptive
- Priorities can be either static or dynamic

Static Priorities

- Do not respond to environment changes, which could be exploited to increase throughput and reduce latency

- Easier to implement

Dynamic Priorities

- Do respond to change;
  - e.g.: the OS may want to temporarily decrease the priority of a process holding a key resource needed by a higher-priority process

- More complex to implement, increased overheads

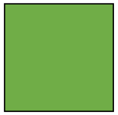
Ex: Priority Scheduling

	P_1	P_2	P_3	P_4	P_5
Burst Time	10	1	2	1	5
Priority	3	1	4	5	2

Time: 0

Ex: Priority Scheduling

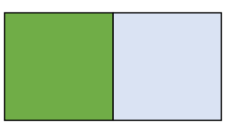
	P_1	P_2	P_3	P_4	P_5
Burst Time	10	1	2	1	5
Priority	3	1	4	5	2



Time: 1

Ex: Priority Scheduling

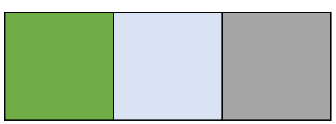
	P_1	P_2	P_3	P_4	P_5
Burst Time	10	1	2	1	5
Priority	3	1	4	5	2



Time: 6

Ex: Priority Scheduling

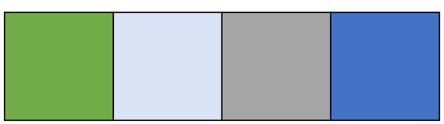
	P_1	P_2	P_3	P_4	P_5
Burst Time	10	1	2	1	5
Priority	3	1	4	5	2



Time: 16

Ex: Priority Scheduling

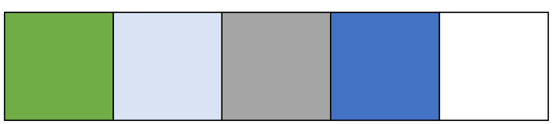
	P_1	P_2	P_3	P_4	P_5
Burst Time	10	1	2	1	5
Priority	3	1	4	5	2



Time: 18

Ex: Priority Scheduling

	P_1	P_2	P_3	P_4	P_5
Burst Time	10	1	2	1	5
Priority	3	1	4	5	2



Time: 19

Waiting Time

- Waiting Time
  - P_1: 6
  - P_2: 0
  - P_3: 16
  - P_4: 18
  - P_5: 1
  - Average: 8.2

Scheduling Algorithms

Multilevel Queue

Multilevel Queue Scheduling

- The ready queue is partitioned into separate queues with their own scheduling algorithm

- E.g. foreground queue (interactive) with RR, background queue (batch) with FCFS

Multilevel Queue Scheduling

- In multilevel queues there must be **scheduling among the queues** too

- Common strategies are:
  - Fixed priority scheduling
  - Time slice

Fixed priority scheduling

- Serve first all processes from foreground queue, then all from background queue
- Absolute precedence of higher-priority queues: possibility of starvation

Time Slice

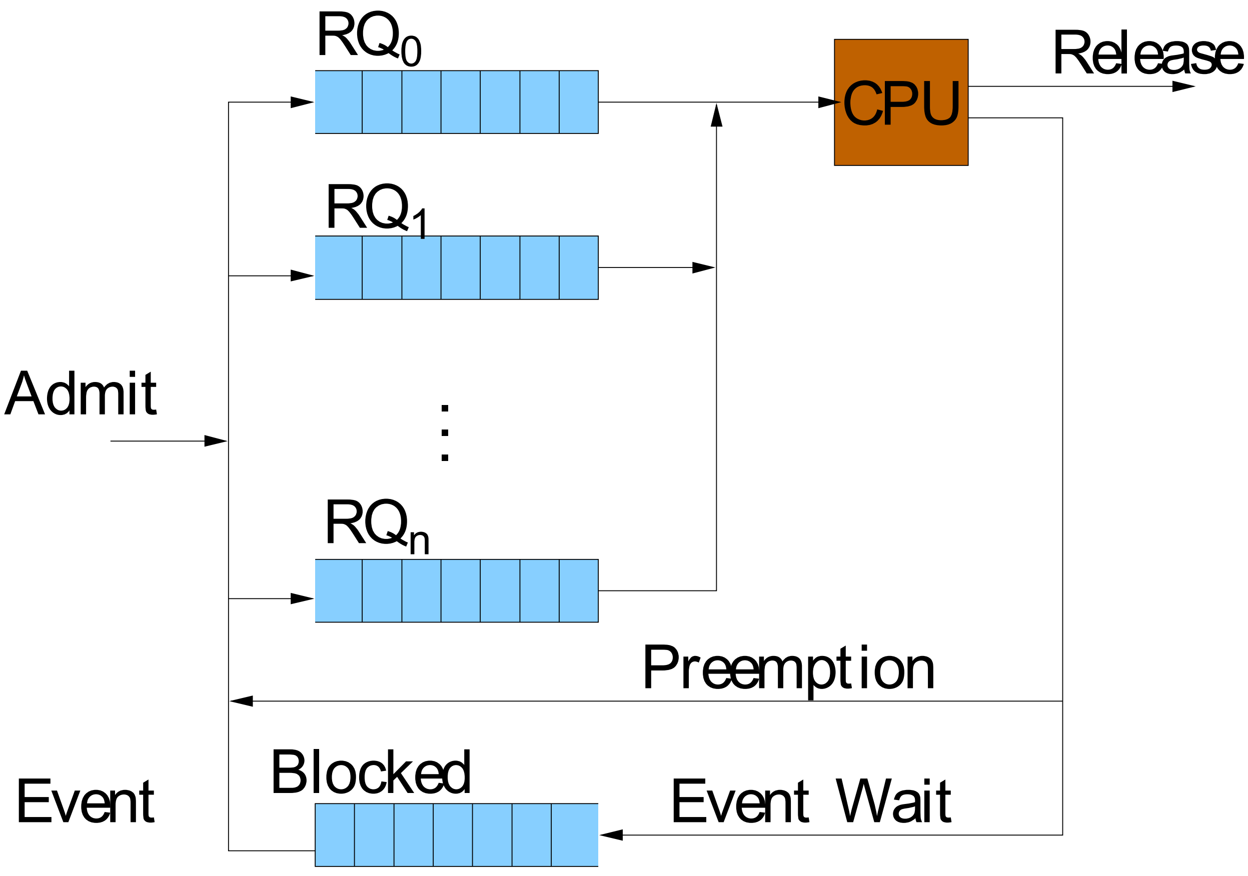
- Each queue gets a certain amount of CPU time which it can schedule among its processes

- E.g.: 80% to foreground in RR, 20% to background in FCFS

Multilevel Queue

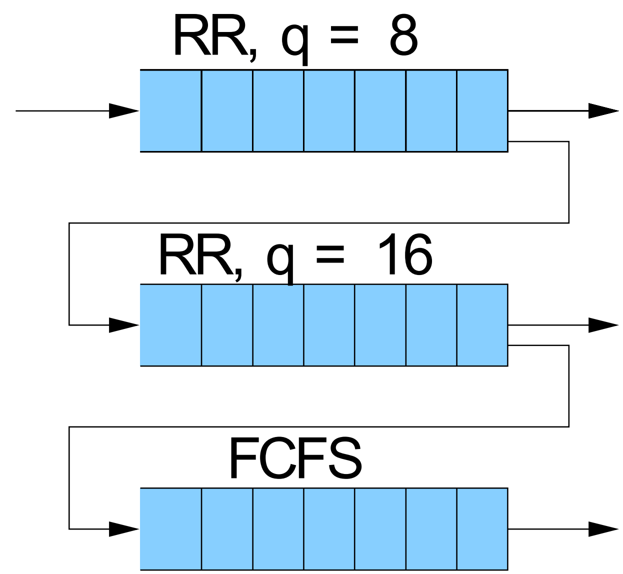
- RQ_0 system processes (highest priority)
- RQ_1 interactive processes
- …
- RQ_n batch processes (lowest priority)

Multilevel Queue



Multilevel Queue with Feedback

- Allows processes to move between queues
- Longer processes are penalised and eventually moved to lower priority queues



Multilevel Queue with Feedback

- System is adaptable
  - When a new process acquires the processor, the system does not know its burst pattern

Traditional Unix Scheduling (BSD 4.3)

- Priority-based
- Multilevel queue feedback using RR within each queue
- 128 priorities in 32 queues (4 adjacent priorities)

Traditional Unix Scheduling (BSD 4.3)

- If a running process does not block or complete within one second, it is preempted
  - Kernel processes (priorities 0–49) cannot be preempted
  - Priorities are recomputed once per second, based on recent CPU usage

Scheduling in Real-Time Systems

- Must meet the needs of processes that must produce correct output by a certain time (**deadlines** or **timing constraints**)
- SJF is optimal for the average waiting time, but it **does not guarantee** a fixed waiting time for any process

Scheduling in Real-Time Systems

- SJF cannot be used in real-time scenarios
  - CPU-bound processes with deadlines will wait too long

- If all processes can meet their deadlines regardless of their execution order, shortest **deadlines** first would be optimal

Soft Real-Time Scheduling

- Missing an occasional deadline is undesirable but tolerable
  - e.g.: multimedia playback

- Priority scheduling required; real-time has highest priority

- Small dispatch latency required (system calls should be preemptible)

Hard Real-Time Scheduling

- Absolute deadlines that always have to be met
  - e.g.: air traffic control

- Special purpose software running on dedicated hardware

Next Week

- Study Time
  - Review Chapter 6