Operating Systems

Autumn 2024

CPU scheduling

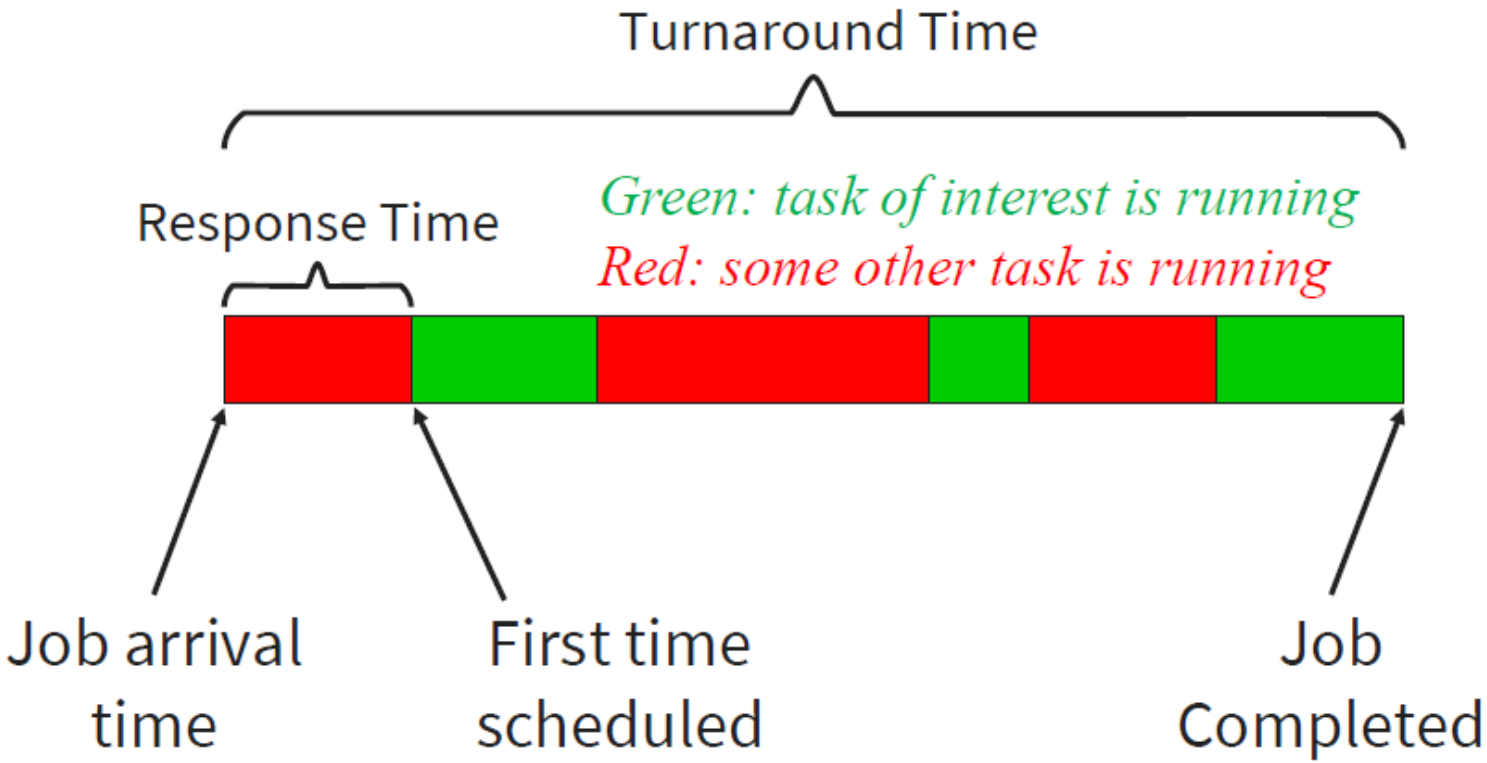
The scheduling problem

- Which process should the OS run?
  - if no runnable process (i.e. no process in the ready state), run the idle task
  - if a single process runable, run this one
  - if more than one runnable process, a scheduling decision must be taken
- Scheduler: code (logic) that decides which process to run as per some policy
- Today: What are the basic scheduling policies? When do they work well?

Standard usage

- Refer to schedulable entities as jobs — could be processes, threads, etc.
- Job: a task that needs a period of CPU time
- Job arrival time
  - when the job was first submitted
- Job run time
  - Time needed to run the task without contention

Scheduling metrics



- Execution time: sum of green periods
- Waiting time: sum of red periods
- Turnaround time: sum of both

Performance terminology

- Turnaround time: How long?
  - User-perceived time to complete some job (completion_time - arrival_time)
- Response time:
  - User-perceived time before first output (initial_schedule_time - arrival_time)
- Waiting time: How much thumb-twiddling?
  - Time on the ready queue (not running)

Performance terminology

- Throughput: How many jobs over time?
  - The rate at which jobs are completed
- Overhead: How much useless work?
  - Time lost due to switching between jobs
- Fairness: How equally are jobs treated?
  - Jobs get same amount of CPU time over some interval

Workload assumptions (to be relaxed later)

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

Scheduling basics

Workloads:

- arrival_time
- run_time

Scheduling Policies:

- FIFO
- SJF (SJN, SPN)
- STCF
- RR

Metrics:

- turnaround_time
- response_time

FIFO

- Run jobs to completion in arrival_time order
  - aka FCFS (first come first served)
- simple, minimal context switch overhead

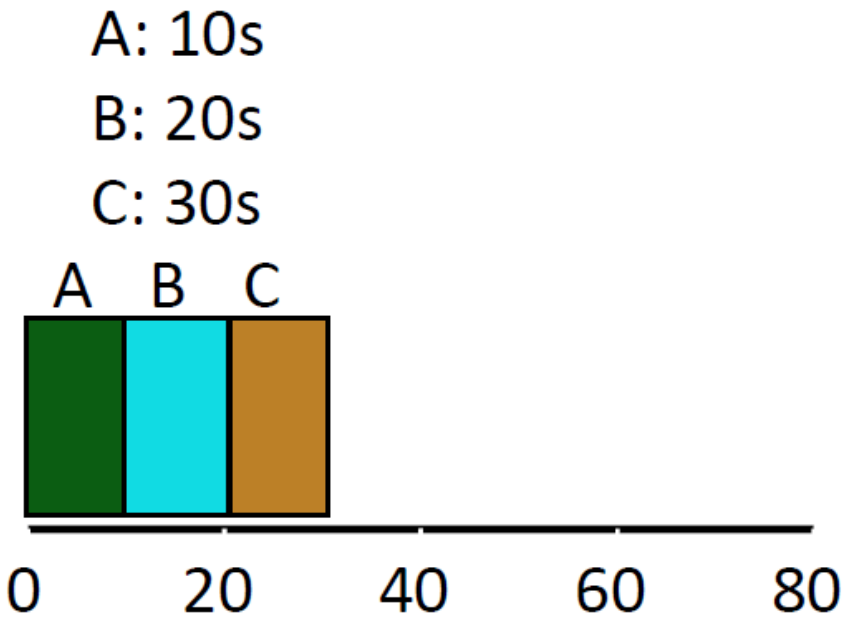
JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10

Time	
0	A arrives
0	B arrives
0	C arrives
0	run A
10	complete A
10	run B
20	complete B
20	run C
30	complete C

FIFO: Identical jobs

- Gantt chart: illustrates how jobs are scheduled over time on a CPU

JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10



- What is the average turnaround time?
  - $(10+20+30)/3 = 20$s

Scheduling basics

Workloads:
- arrival_time
- run_time

Scheduling Policies:
- FIFO
- SJF (SJN, SPN)
- STCF
- RR

Metrics:
- turnaround_time
- response_time

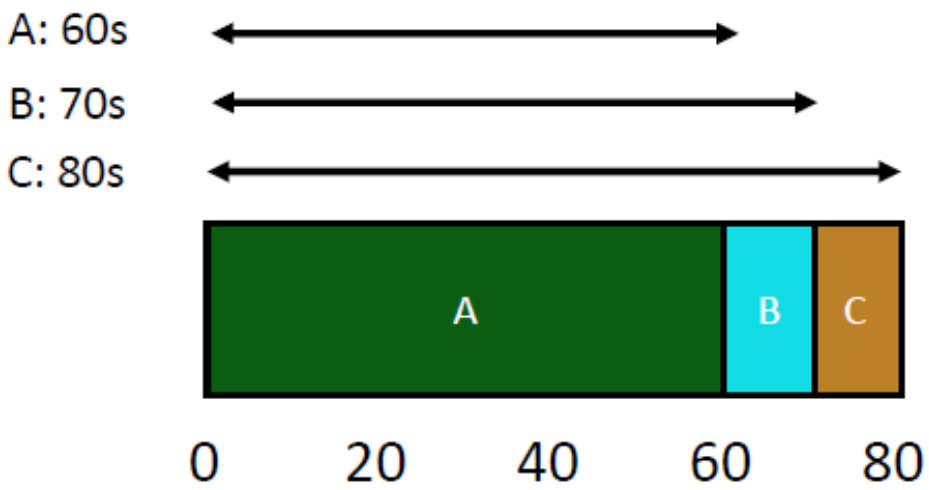
Workload assumptions

1. ~~Each job runs for the same amount of time~~
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

FIFO: Convoy effect

- Problem: turnaround time can suffer when short jobs must wait for long jobs

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	~0	10
C	~0	10



Average turnaround time: 70s

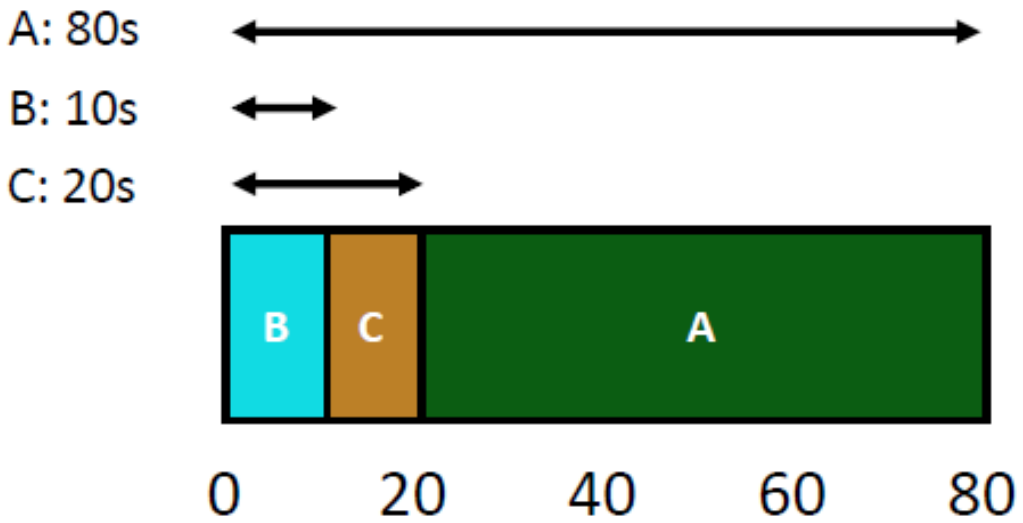
Shortest Job First (SJF)

- Idea: choose job with the smallest run_time
  - aka shortest job next (SJN), shortest process next (SPN)
- Consider our previous example

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	~0	10
C	~0	10

What is the average turnaround time with SJF?

SJF turnaround time



- What is the average turnaround time?
  - (80+10+20)/3 = 36.7s
- Average turnaround with FIFO: 70s

Scheduling basics

Workloads:
- arrival_time
- run_time

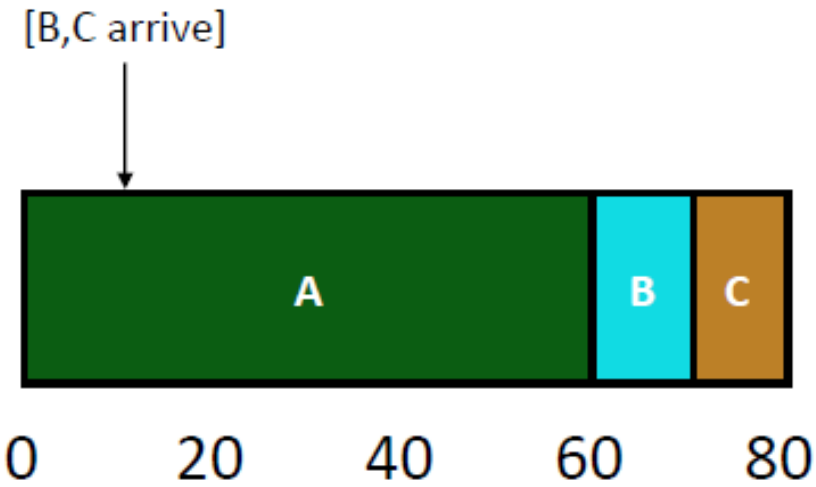
Scheduling Policies:
- FIFO
- SJF (SJN, SPN)
- STCF
- RR

Metrics:
- turnaround_time
- response_time

Workload assumptions

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

SJF with late arrivals



JOB	arrival_time (s)	run_time (s)
A	~0	60
B	~10	10
C	~10	10

- What is the average turnaround time?
  - (60 + (70-10) + (80-10)) / 3 = 63.3s

Preemptive scheduling

- FIFO and SJF are non-preemptive
  - Only schedule new job when previous job voluntarily relinquishes CPU
- Preemptive: potentially schedule different job at any point by taking CPU away from running job

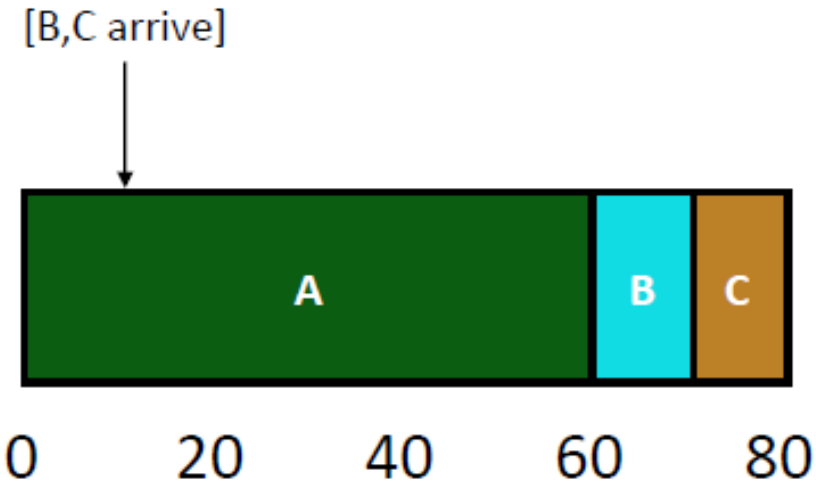
Shortest Time-to-Completion First (STCF)

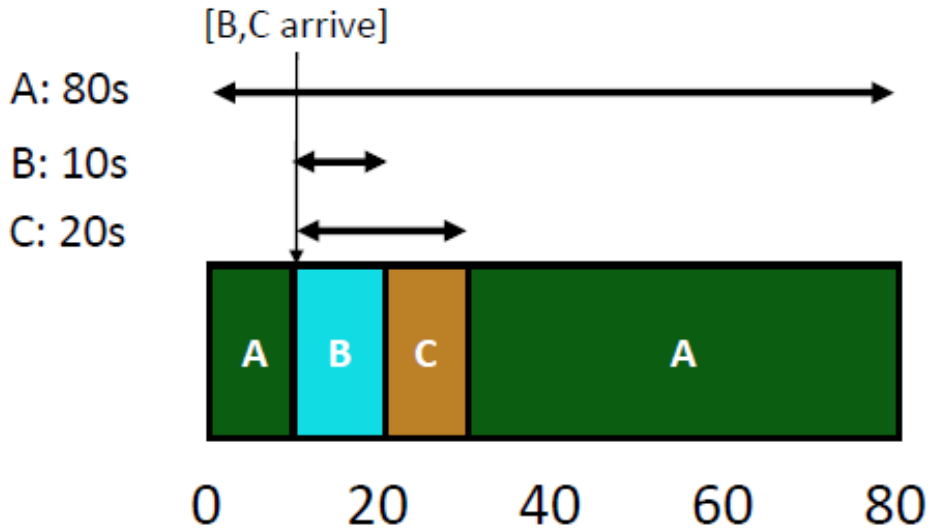
- SJF + preemption
- Select job with the least remaining time to run next
- Consider our previous example

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	~10	10
C	~10	10

- Average turnaround time with STCF?

SJF vs. STCF





- Average turnaround time with STCF
  - ((80-0) + (20-10) + (30-10))/3 = 36.7s
- Average turnaround time with SJF
  - 63.3s

Q

A preemptive scheduler is best characterized by its ability to carry out this state transition:

1. READY $\rightarrow$ RUNNING
2. RUNNING $\rightarrow$ READY

Q

Which of the following state transitions will never take place in a FIFO scheduler?

1. READY $\rightarrow$ RUNNING
2. RUNNING $\rightarrow$ READY

Scheduling basics

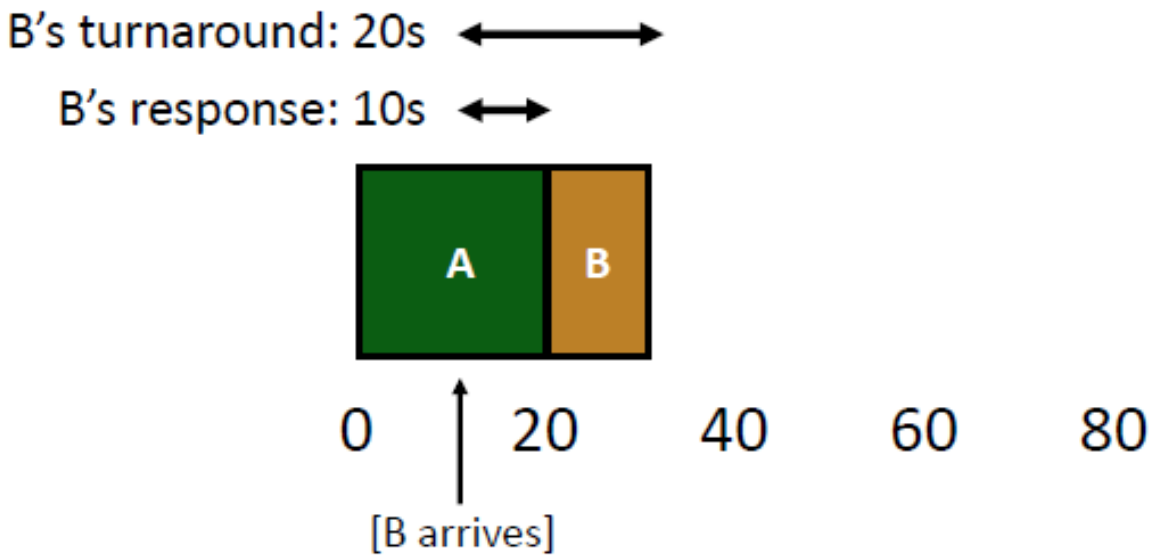
Workloads:
- arrival_time
- run_time

Scheduling Policies:
- FIFO
- SJF (SJN, SPN)
- STCF
- RR

Metrics:
- turnaround_time
- response_time

Response vs. turnaround

- Response time: first run time - arrival time

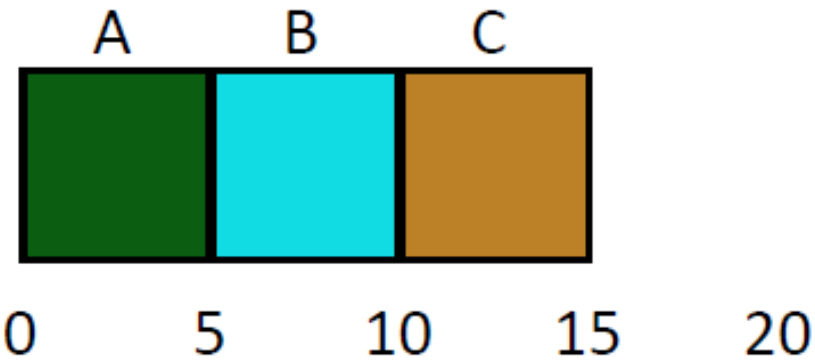


- FIFO, SJF, and STCF can have poor response time

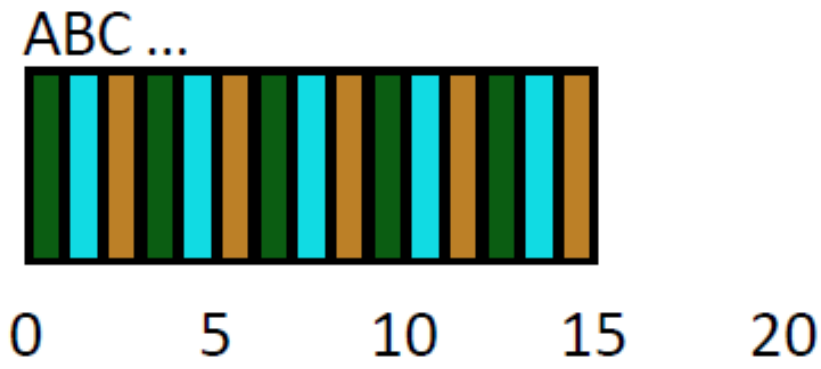
Round-Robin (RR)

- Each job allowed to run for a quantum
  - quantum = some configured period of time
- Context is switched (at the latest) at the end of the quantum – premption!
- Next job is the one on the ready queue that hasn't run for the longest amount of time

FIFO vs RR



Avg Response Time?
(0+5+10)/3 = 5



Avg Response Time?
(0+1+2)/3 = 1

RR Pros and Cons

- In what way is RR worse?
  - Avg. turnaround time with equal job length is horrible
- Other reasons why RR could be better?
  - If don't know run-time of each job, gives short jobs a chance to run and finish fast

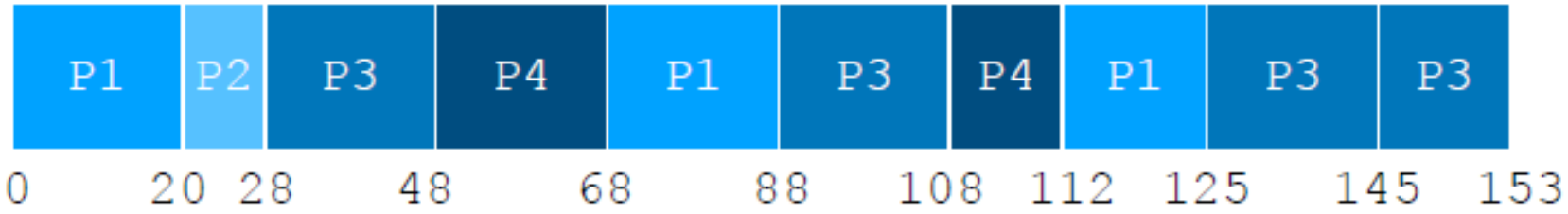
Time quantum

- What is a good quantum size?
- Too long, and it morphs into FIFO
- Too short, and time is wasted on context switching
- Typical quantum: about 100X cost of context switch ($\sim$ 100 ms vs. $<<$ 1 ms)

RR Example

- time slice = 20
- workload (with runtimes)
  - P1: 53
  - P2: 8
  - P3: 68
  - P4: 24

RR Example



Avg turnaround time = (125+28+153+112)/4 = 104.25

Scheduling basics

Workloads:
- arrival_time
- run_time

Scheduling Policies:
- FIFO
- SJF (SJN, SPN)
- STCF
- RR

Metrics:
- turnaround_time
- response_time

Workload assumptions

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. ~~All jobs only use the CPU (no I/O)~~
4. Run-time of each job is known

Incorporating I/O

- When a job initiates an I/O request
  - The job is blocked waiting for I/O completion
  - The scheduler should schedule another job on the CPU
- When the I/O completes
  - An interrupt is raised
  - The OS moves the process from blocked back to the ready state

I/O bound and CPU bound jobs

- A job is CPU bound if it needs lost of CPU time to progress
  - These jobs have a lot of logic and math

I/O bound and CPU bound jobs

- A job is CPU bound if it needs lost of CPU time to progress
  - These jobs have a lot of logic and math
  - Usually in running or ready state

I/O bound and CPU bound jobs

- A job is CPU bound if it needs lost of CPU time to progress
  - These jobs have a lot of logic and math
  - Usually in running or ready state
- Job is I/O bound if it needs to do lots of I/O to progress
  - These jobs access disk, network, etc

I/O bound and CPU bound jobs

- A job is CPU bound if it needs lost of CPU time to progress
  - These jobs have a lot of logic and math
  - Usually in running or ready state
- Job is I/O bound if it needs to do lots of I/O to progress
  - These jobs access disk, network, etc
  - Usually in the waiting (or, blocked) state

RR

- Consider a mixture of I/O-bound and CPU-bound jobs
- Even though the I/O completes quickly, the I/O-bound task must wait to be reassigned the CPU until the CPU-bound tasks both complete their time slice