Process Scheduling

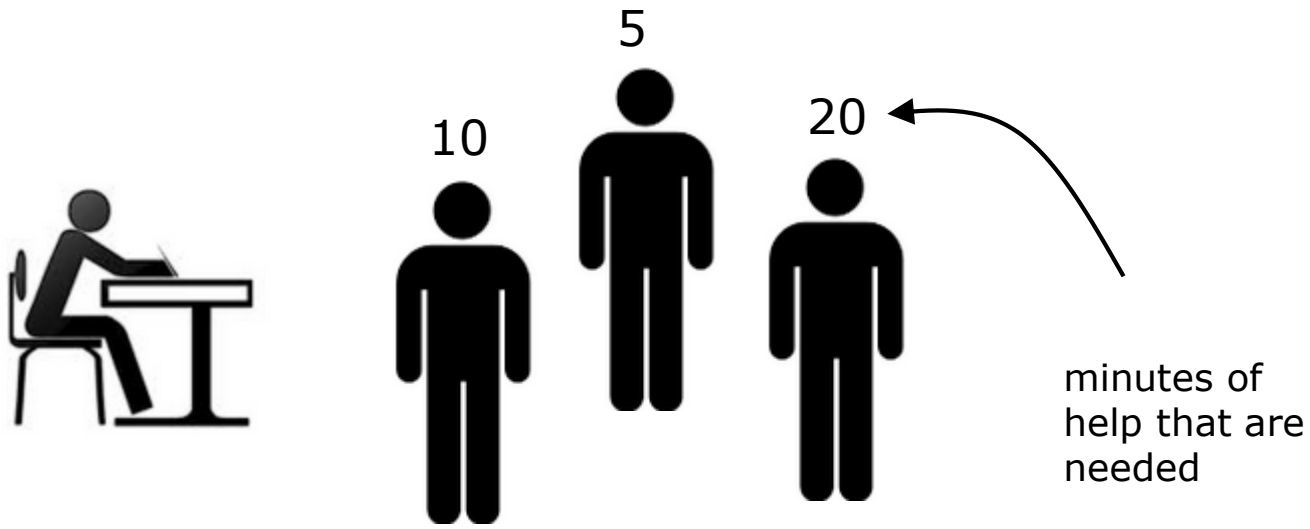
Glenn Bruns
CSUMB

Lecture Objectives

At the end of this lecture, you should be able to:

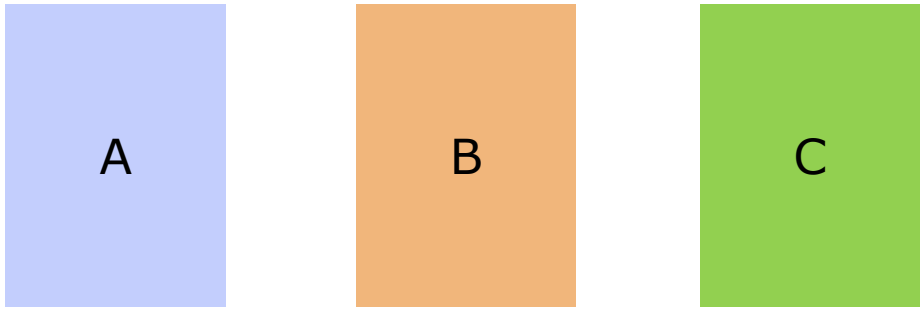
- Describe some **process scheduling algorithms** and be able to apply them by hand
- Describe some **process scheduling metrics** and be able to compute them by hand

Think of people at a help desk



- The help desk person has 3 people to serve
- To the help desk, each person is a "**job**"
- Which job should be processed first?

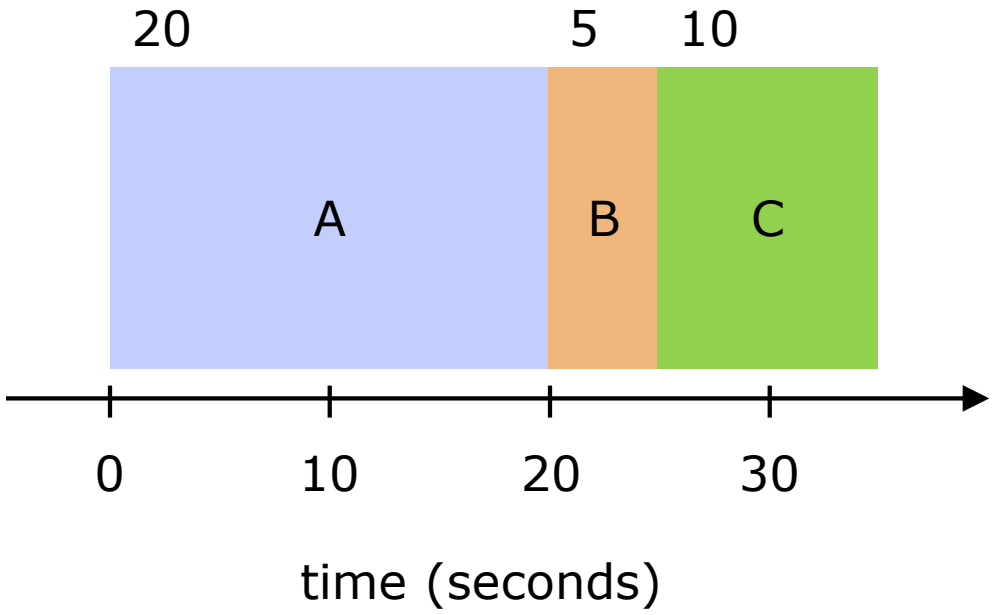
How should OS run a few programs?



Assumptions:

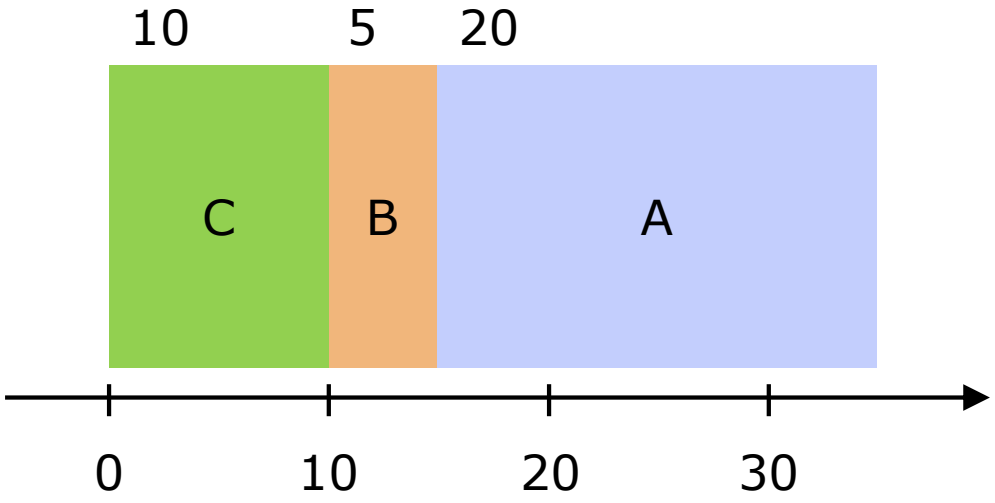
1. the programs all arrive at the same time
2. each program runs uninterrupted until it stops
3. the programs use CPU only
4. we know how long each program will run

What is the best way to run these?



If program A takes 20 seconds to run, B takes 5, and C takes 10, is this the best way to schedule them?

Exercise



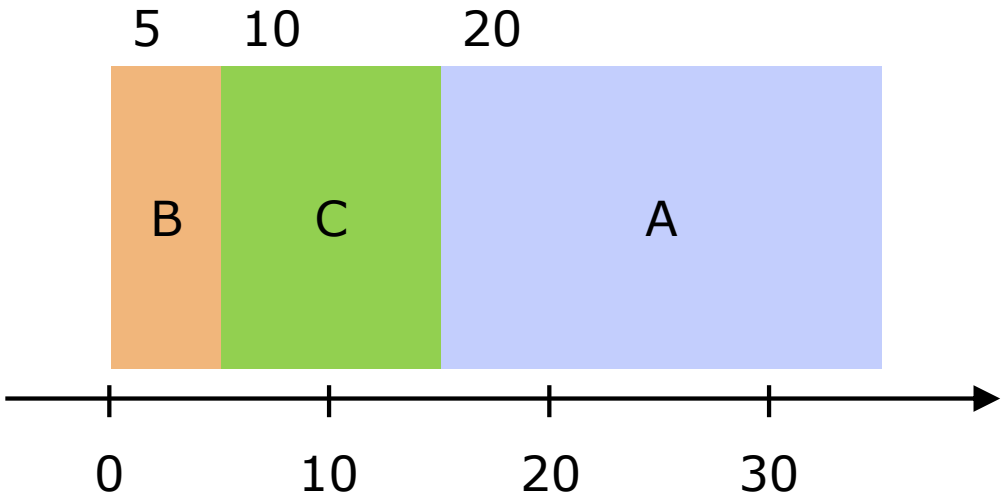
turnaround time
= amount of time between job arriving and job finishing

What's the average "*turnaround time*" with this schedule?

Which schedule would minimize turnaround time?

What would the average turnaround time be with that schedule?

Shortest Job First (SJF) policy



This policy minimizes average turnaround time (if jobs arrive at same time)

Exercise

Let A, B, C be the running times of three jobs.

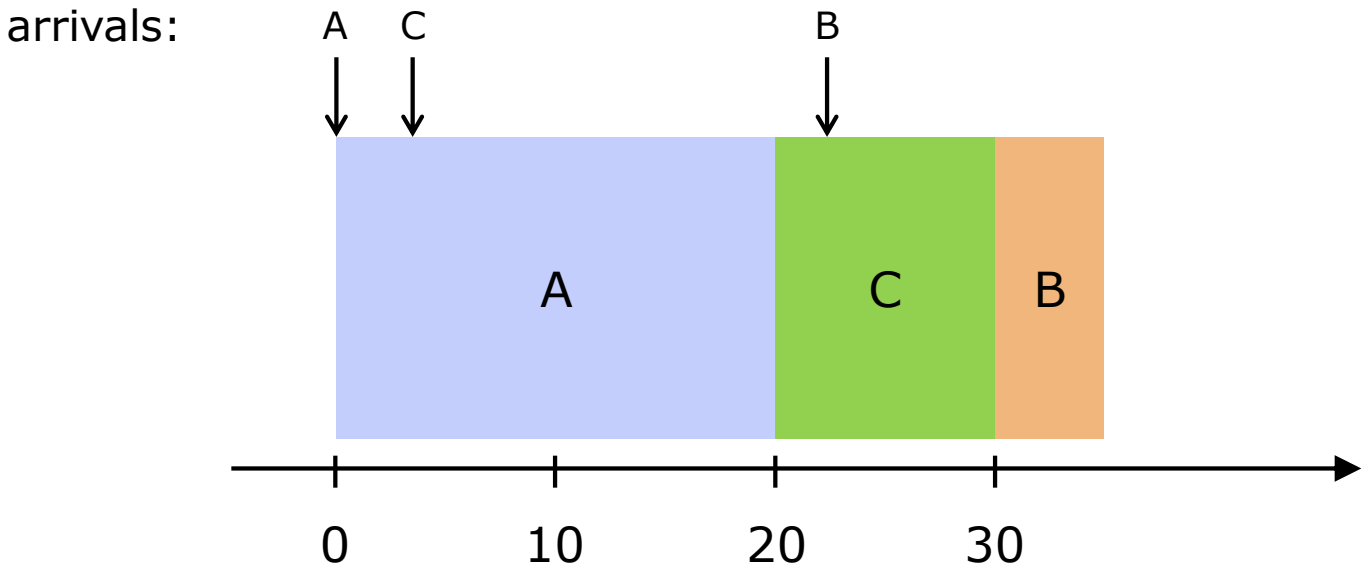
Write the average turnaround time as a formula if we run A, then B, then C.

Is it clear now why putting the shortest job first minimizes average turnaround time?

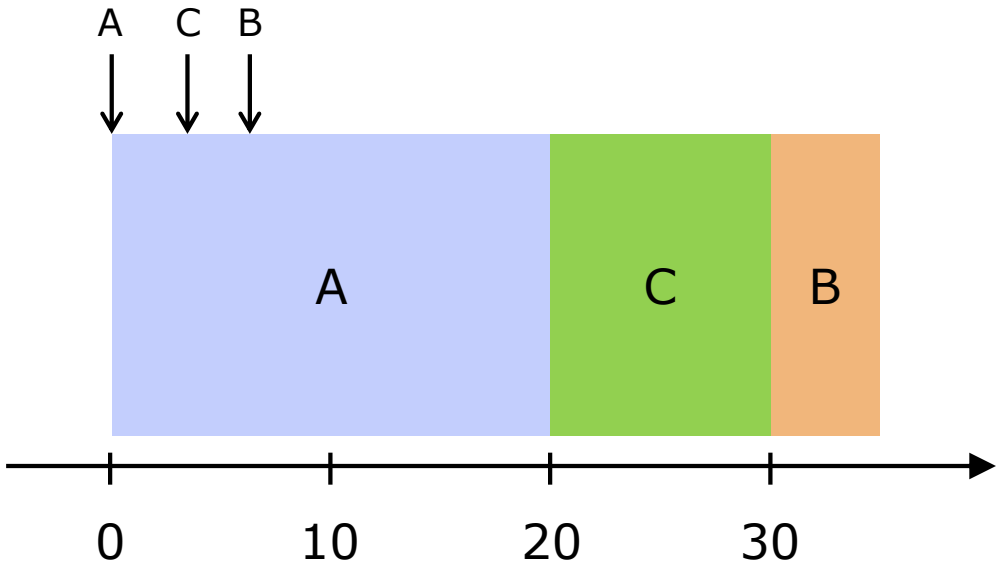
Exercise

Suppose jobs don't arrive at the same time.

What problem could arise?



First-come, first-served (FCFS) policy

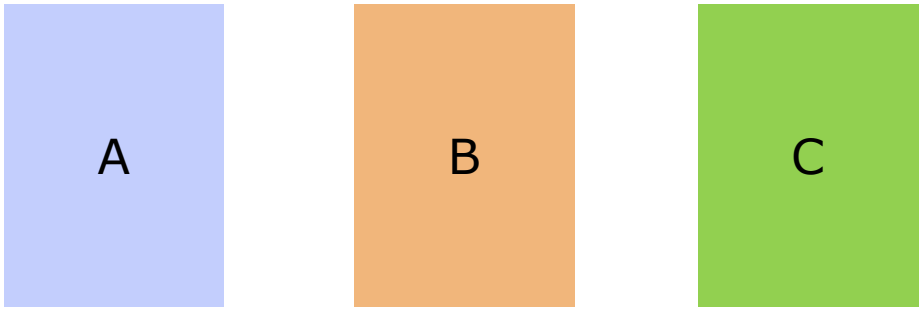


Process jobs in order they arrive, regardless of time they need.

This example is like waiting behind a person with a big cart at the market.

Avg. turnaround is about 26 seconds.

Allow programs to be interrupted

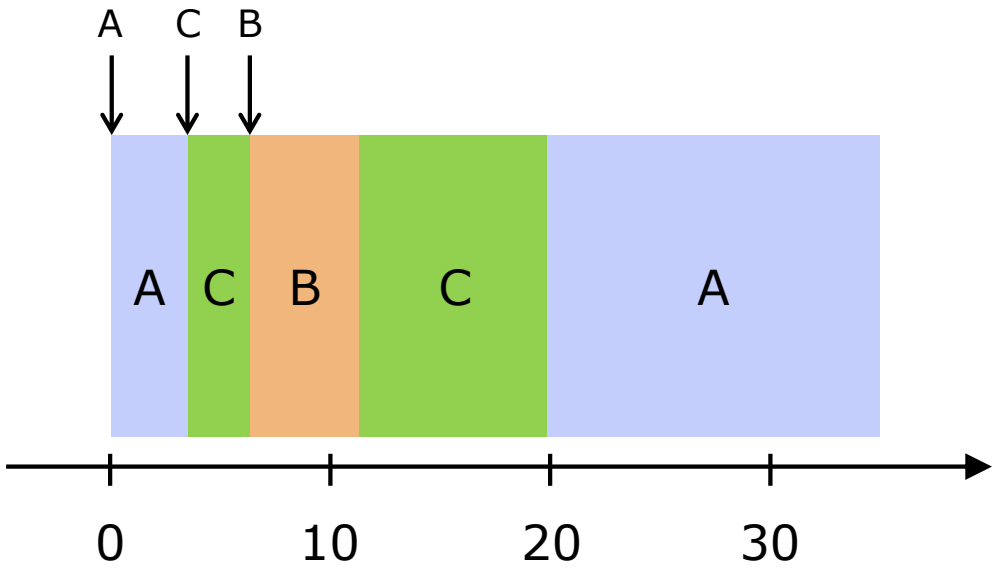


now lets say we can interrupt processes

Assumptions:

1. ~~the programs all arrive at the same time~~
2. each program runs uninterrupted until it stops
3. the programs use CPU only
4. we know how long each program runs

Shortest time to completion first (STCF)

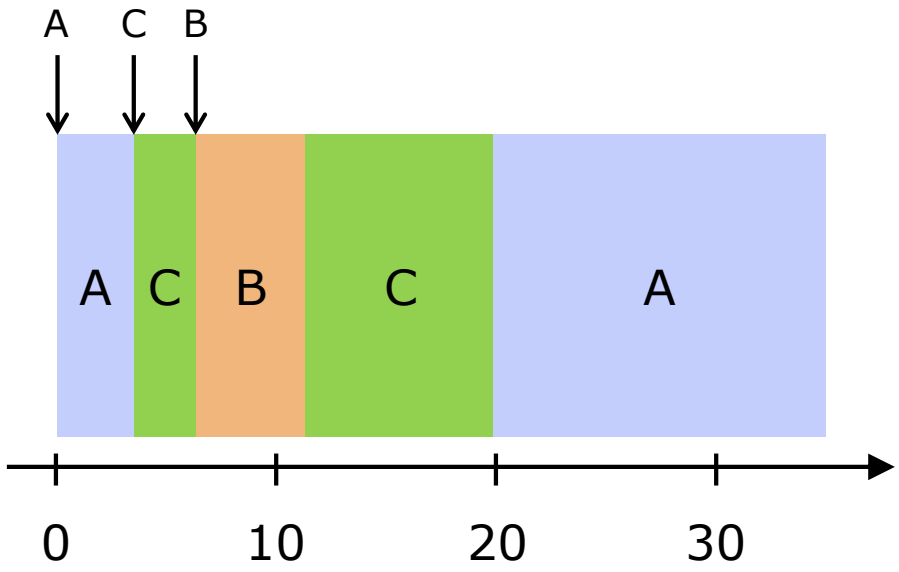


Whenever a job arrives, run the job that would finish first.

Here, C "preempts" A, and then B preempts C.

Avg. turnaround time is better than FCFS: about 19 seconds

Average response time

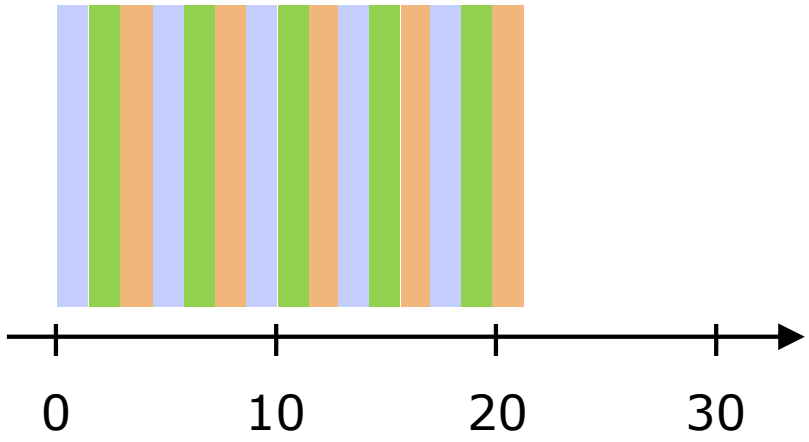


response time = amount of time before job arriving and system first responding to the job

Do we really want to wait for 10 seconds or more for our program to run?

We want an *interactive* system.

Round Robin scheduling

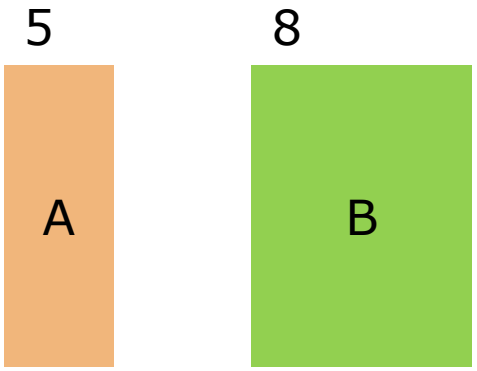


Exercise: why not make time slices very short?

Take turns, giving each job a time slice.

Good for response time, bad for turnaround time

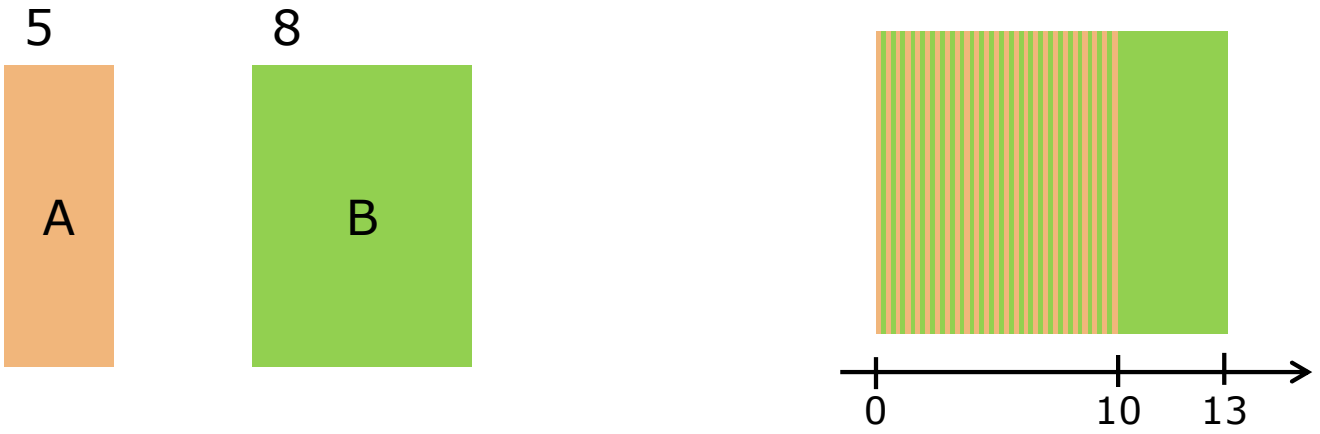
Round robin example



What is the average turnaround when A and B are run with round robin scheduling? Job times above are in seconds.

Assume the 'time slice' (or 'quantum') that a job gets when it runs is very short.

Solution to example



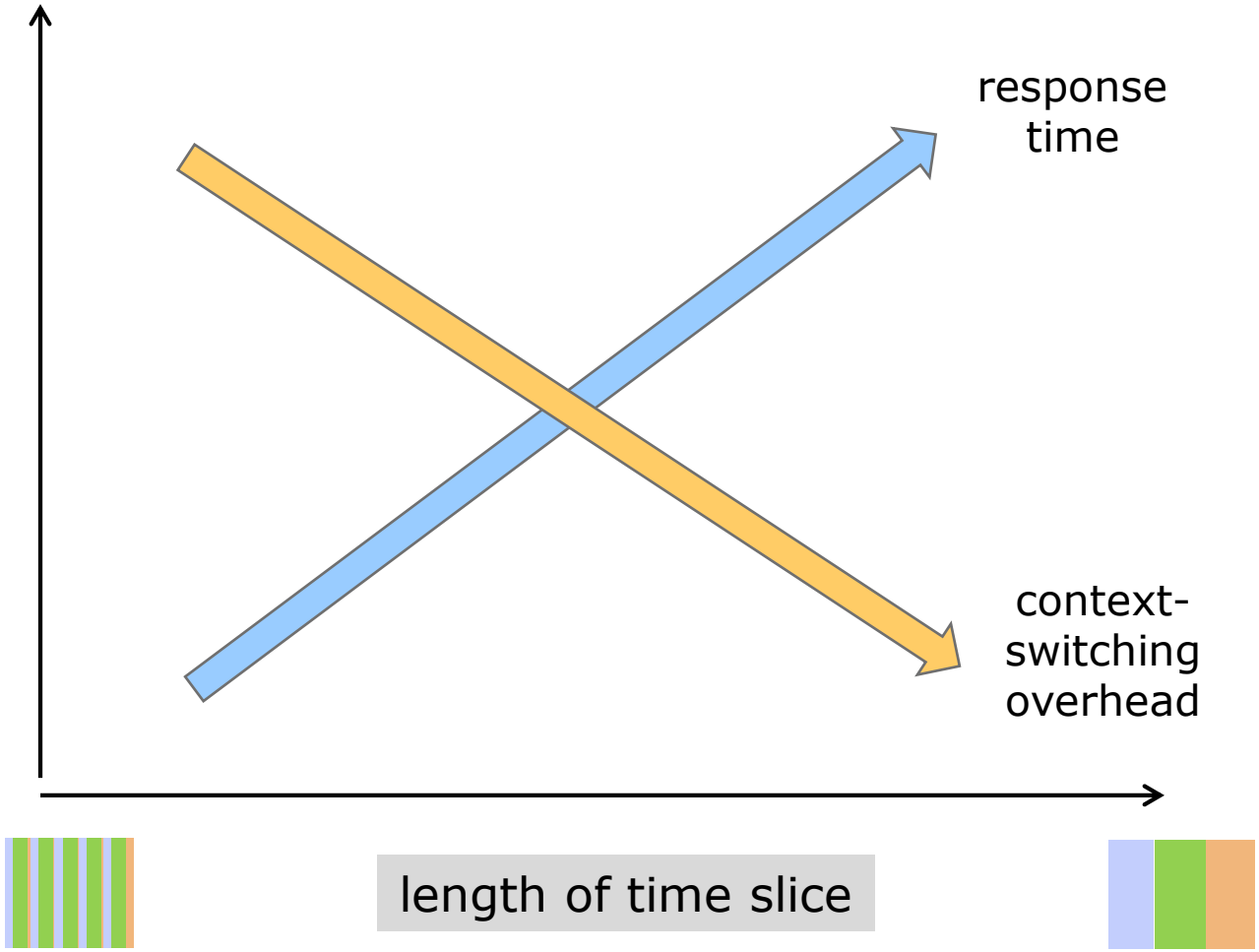
A is the shortest job. It will take 10 seconds to complete, because A and B will share the CPU until A finishes.

During those 10 seconds A and B both receive 5 seconds of CPU time. So when A is finished, B still needs 3 more seconds of CPU time.

So the turnaround time for A is 10, and the turnaround time for B is 13.

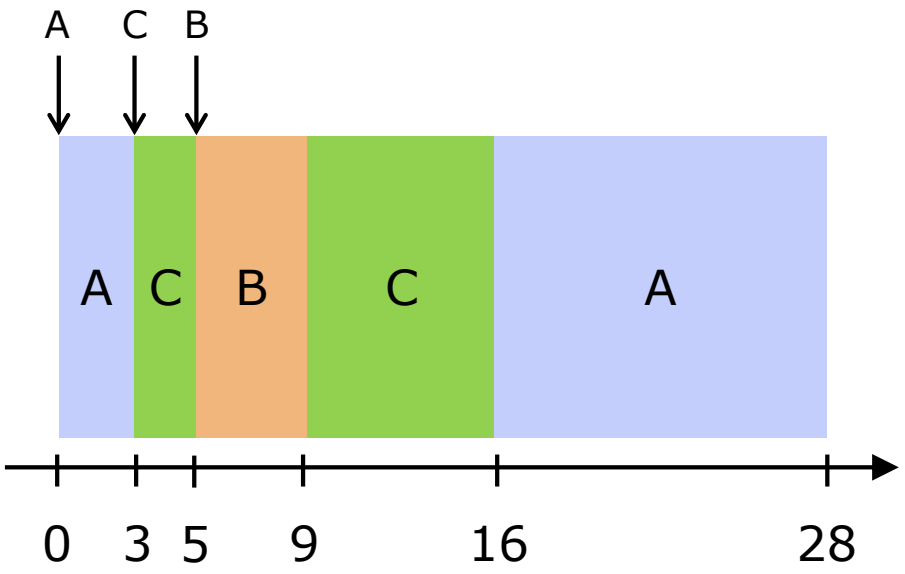
Avg. turnaround time for A and B is (10 + 13)/2 = 11.5

Round Robin and context switching



Waiting time

- Waiting time is also sometimes used as a metric
- It means: total time a process spends waiting for service.



Waiting time =
turnaround time
- job length

process	waiting time
A	28 - 15 = 13
B	4 - 4 = 0
C	13 - 9 = 4

Summary

- Some **policies** for job scheduling:
  - First-come, First-served (FCFS)
  - Shortest Job First (SJF)
  - Shortest Time-to-Completion First (STCF)
  - Round Robin
- Some **metrics** for measuring how well a policy works:
  - Avg. Turnaround Time
  - Avg. Response Time
- Need to make tradeoffs between turnaround time and response time