Universidade Estadual de Campinas
Instituto de Computação

MC504 Sistemas Operacionais

T04

CPU Virtualization

An Introduction to Scheduling

Disclaimer: *This lecture slide set was developed to support the MC504 Operating System course at the University of Campinas and is mainly based on Chapter 07 of the Operating Systems: Three Easy Pieces book by Remzi and Andrea Arpaci-Dusseau of the University of Wisconsin, available at pages.cs.wisc.edu/~remzi/OSTEP/.*

Arthur João Catto, PhD

2º semestre de 2018

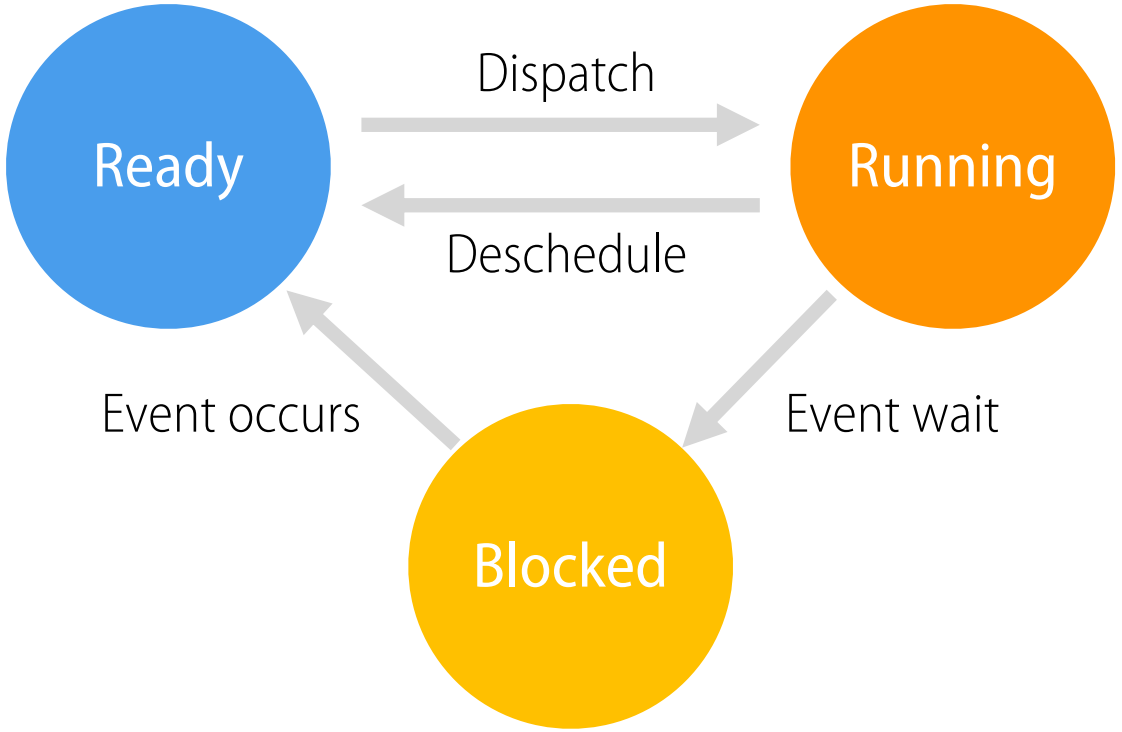
CPU Virtualization

Two Components

- Dispatcher (Previous lecture)
  - Low-level mechanism
  - Performs context-switch
    - Switch from user mode to kernel mode
    - Save execution state (registers) of old process in PCB
    - Insert PCB in ready queue
    - Load state of next process from PCB to registers
    - Switch from kernel to user mode
    - Jump to instruction in new user process
- Scheduler (Today)
  - Applies a policy to determine which process gets CPU when there is a process switch.

Review

State transitions



How to transition? ("mechanism")

When to transition? ("policy")

Usual Scheduling Performance Metrics

- Minimize turnaround time
  - Do not want to wait long for job to complete
  - $T_{turnaround} = T_{completion} - T_{arrival}$
- Minimize response time
  - Schedule interactive jobs promptly so users see output quickly
  - $T_{response} = T_{initial\_schedule} - T_{arrival}$
- Minimize waiting time
  - Do not spend much time in Ready queue
- Maximize throughput
  - Want many jobs to complete per unit of time
- Maximize resource utilization
  - Keep expensive devices busy
- Minimize overhead
  - Reduce number of context switches
- Maximize fairness
  - Give all jobs same amount of CPU over some time interval

Scheduling basics

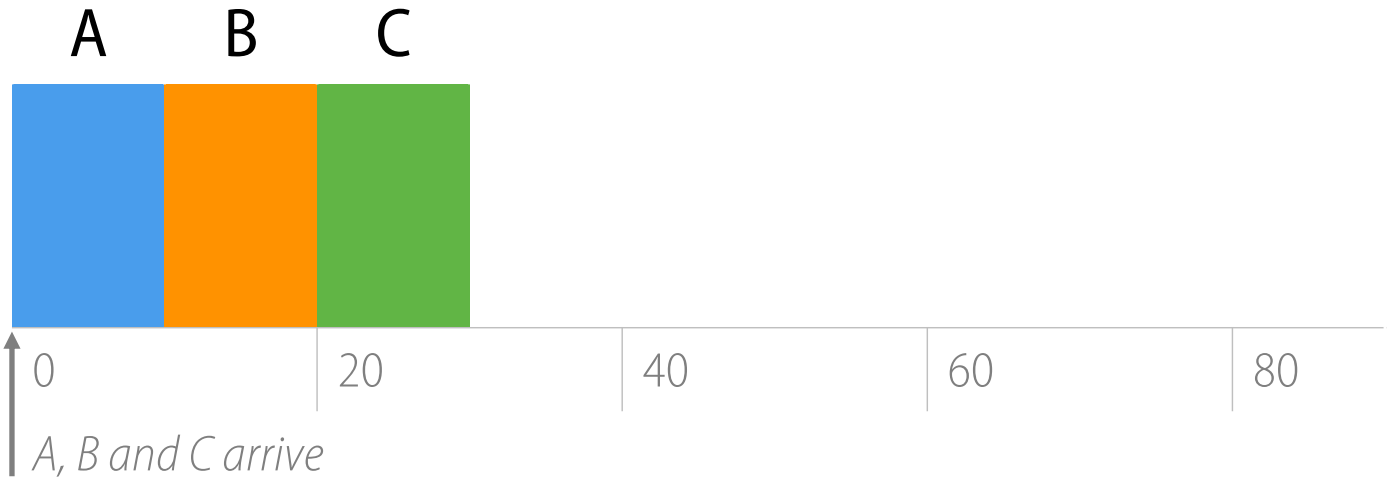
Workload	Scheduler	Metrics
$T_{arrival}$	FIFO	$T_{turnaround}$
T_{run}	SJF	$T_{response}$
	STCF	
	RR	

Workload Assumptions

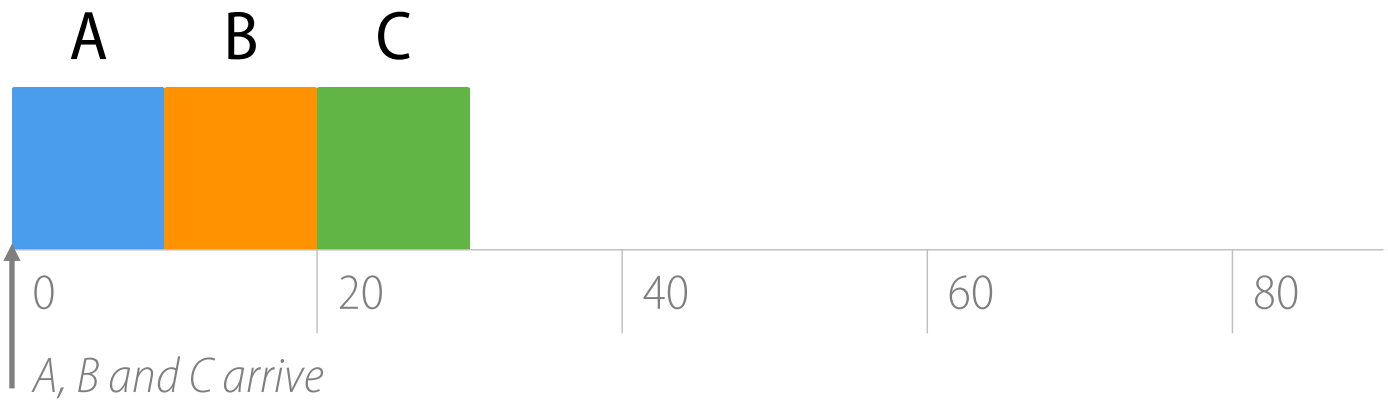
1. All jobs run for the same amount of time.
2. All jobs arrive at the same time.
3. All jobs only use the CPU (i.e. no I/O).
4. Run-time of each job is known.

First In, First Out (FIFO)

- Also called First Come, First Served (FCFS)
  - Very simple and easy to implement
  - Jobs are executed in $T_{arrival}$ order
- Example:
  - Job A arrives just before job B which arrives just before job C.
  - Each job runs for 10 seconds.



FIFO (identical jobs)



- What is the average turnaround time?
  - $T_{turnaround} = T_{completion} - T_{arrival}$
  - $aT_{turnaround} = 10 - 0$
  - $bT_{turnaround} = 20 - 0$
  - $cT_{turnaround} = 30 - 0$
  - $avgT_{turnaround} = \frac{10+20+30}{3} = 20$

Why wouldn't FIFO be our solution of choice?

Relaxed Workload Assumptions

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

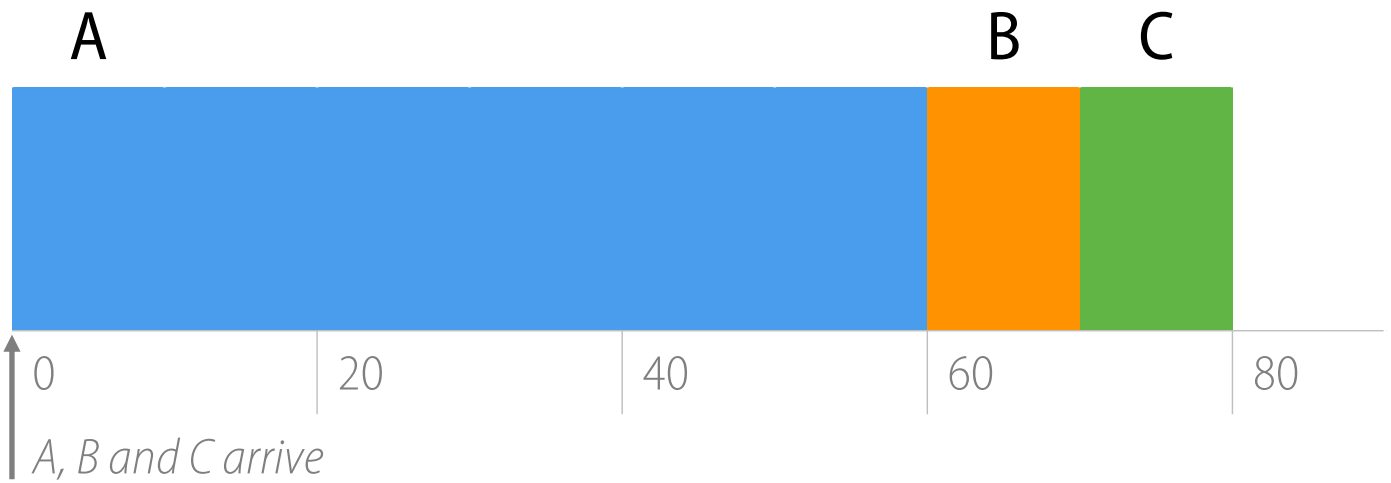
Any problematic workloads for FIFO?

Workload	Scheduler	Metrics
$T_{arrival}$	FIFO	$T_{turnaround}$
T_{run}	SJF	$T_{response}$
	STCF	
	RR	

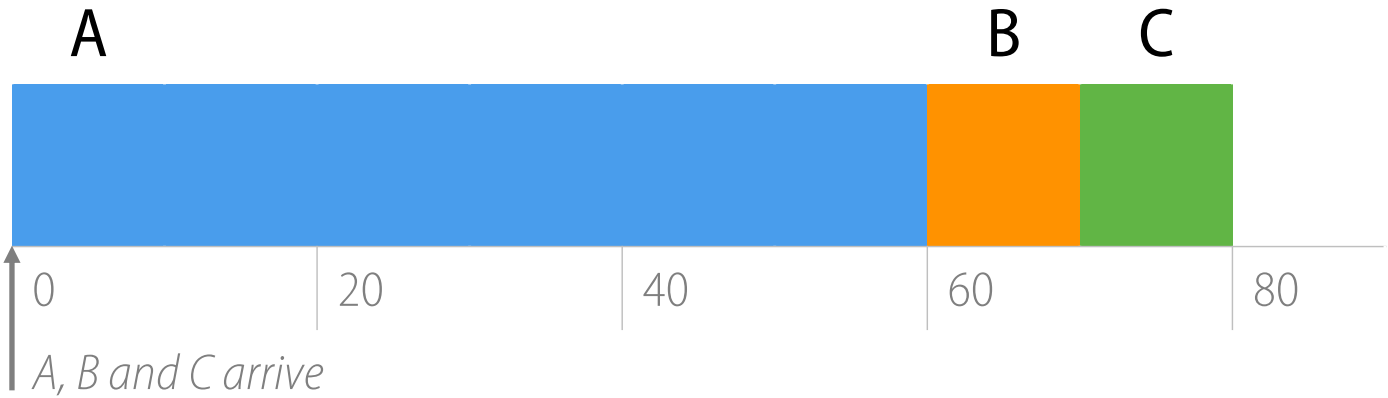
- Think of a really big first job…
  - What would happen to FIFO's turnaround time?

FIFO (big first job)

- Workload
  - Job A arrives just before job B which arrives just before job C.
  - Job A runs for 60 seconds, jobs B and C run for 10 seconds.



FIFO (identical jobs)



- What is the average turnaround time?
  - $T_{turnaround} = T_{completion} - T_{arrival}$
  - $aT_{turnaround} = 60 - 0$
  - $bT_{turnaround} = 70 - 0$
  - $cT_{turnaround} = 80 - 0$
  - $avgT_{turnaround} = \frac{60+70+80}{3} = 70$

Jake Fear, of Highbridge, Somerset, was fined £190 after driving at 25mph for more than three miles on the A39 near Glastonbury in Somerset (file photo)

Overtaking the tractor

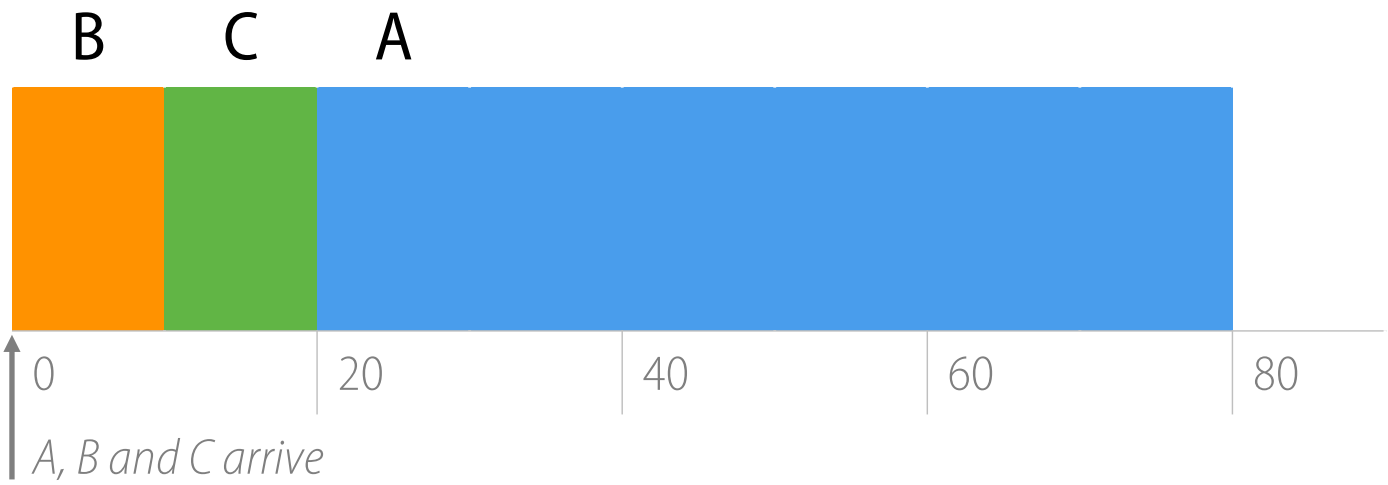
- What was the problem with FIFO?
  - Turnaround time can suffer when short jobs must wait for long jobs
- Let's try a new scheduler: **SJF (Shortest Job First)**
  - Choose job with smallest run time

Scheduling basics

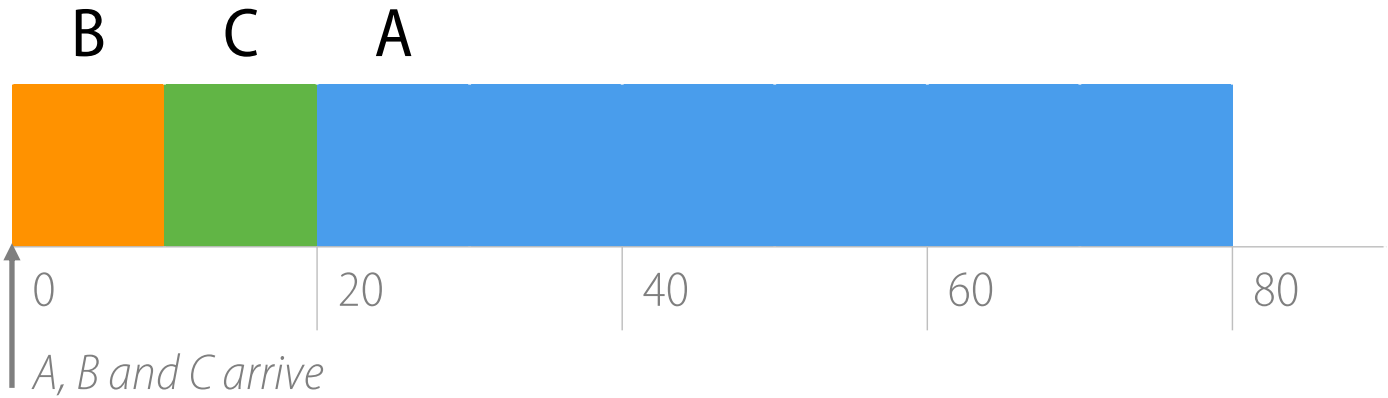
Workload	Scheduler	Metrics
$T_{arrival}$	FIFO	$T_{turnaround}$
T_{run}	SJF	$T_{response}$
	STCF	
	RR	

SJF (shortest job first)

- Workload
  - Jobs A, B and C arrive at the same time.
  - Job A runs for 60 seconds, jobs B and C run for 10 seconds.



SJF (shortest job first)



- What is the average turnaround time?
  - $T_{turnaround} = T_{completion} - T_{arrival}$
  - $aT_{turnaround} = 80 - 0$
  - $bT_{turnaround} = 10 - 0$
  - $cT_{turnaround} = 20 - 0$
  - $avgT_{turnaround} = \frac{80+10+20}{3} = 37$

SJF (shortest job first)

- **SJF is provably optimal**, for minimizing average turnaround time (with no preemption).
  - Scheduling a shorter job before a longer job improves the turnaround time of the shorter job more than it harms the turnaround time of the longer job.

Why wouldn't SJF be our solution of choice?

Relaxed Workload Assumptions

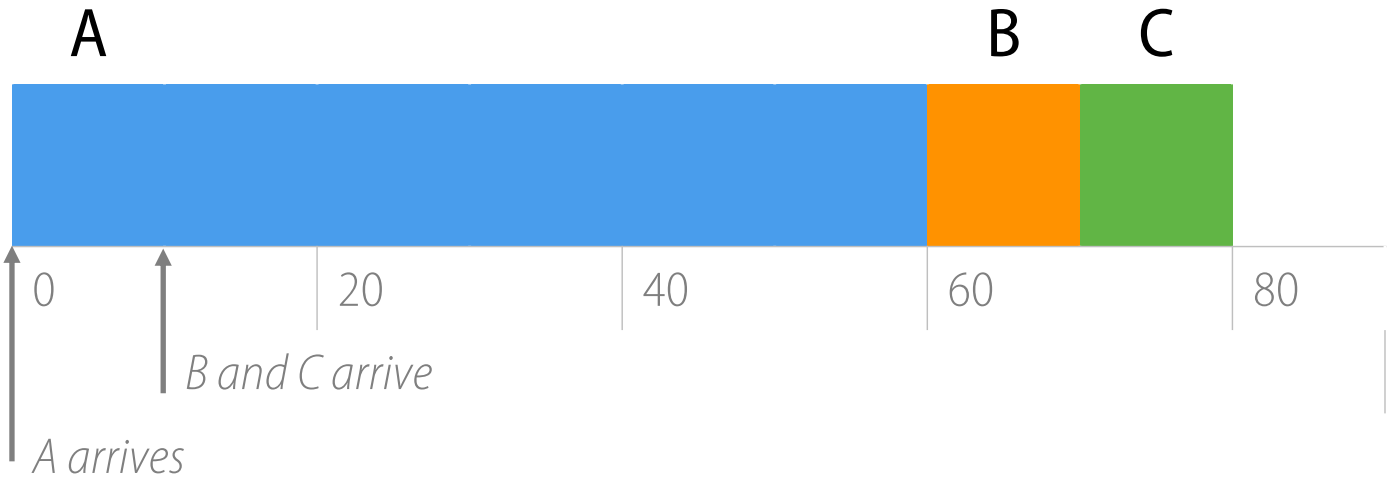
1. ~~All jobs run for the same amount of time.~~
2. ~~All jobs arrive at the same time.~~
3. All jobs only use the CPU (i.e. no I/O).
4. Run-time of each job is known.

Any problematic workloads for SJF?

Workload	Scheduler	Metrics
$T_{arrival}$	FIFO	$T_{turnaround}$
T_{run}	SJF	$T_{response}$
	STCF	
	RR	

- Imagine that the shorter jobs arrive after a longer one…
  - Let's make $aT_{arrival} = 0$ and $bT_{arrival} = cT_{arrival} = 10$.
  - What would happen to SJF's turnaround time?

Stuck behind a tractor again…

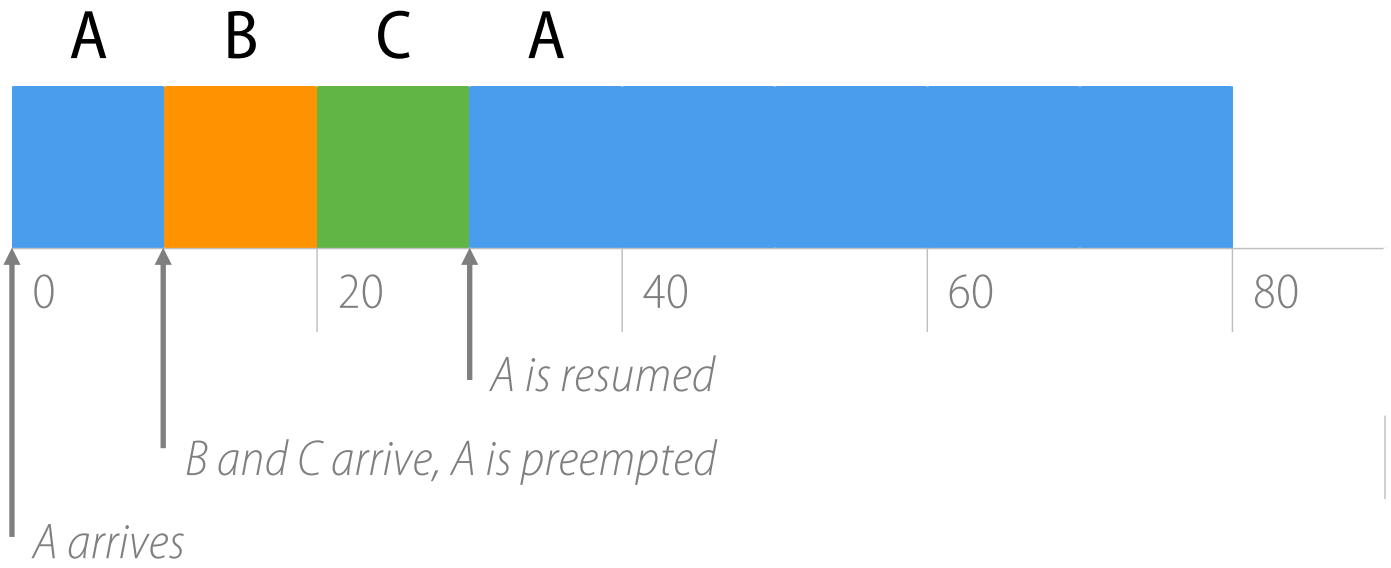


- What is the average turnaround time?
  - $T_{turnaround} = T_{completion} - T_{arrival}$
  - $aT_{turnaround} = 60 - 0$
  - $bT_{turnaround} = 70 - 10$
  - $cT_{turnaround} = 80 - 10$
  - $avgT_{turnaround} = \frac{60+60+70}{3} = 63$

A case for **preemptive scheduling**

- What was the problem with FIFO and SJF?
  - They are non-preemptive, i.e. they only schedule a new job when the previous job voluntarily relinquishes the CPU (performs I/O or exits)
- Let's try a new scheduler: **STCF (Shortest Time-to-Completion First)**
  - Also known as Preemptive Shortest Job First (PSJF) and Shortest Remaining-Time First (SRTF).
  - Potentially schedule a different job at any point by taking CPU away from running job.
  - Always run the quickest-to-complete job.

Preemptive STCF: overtaking the tractor again



- What is the average turnaround time?
  - $T_{turnaround} = T_{completion} - T_{arrival}$
  - $aT_{turnaround} = 80 - 0$
  - $bT_{turnaround} = 20 - 10$
  - $cT_{turnaround} = 30 - 10$
  - $avgT_{turnaround} = \frac{80+10+20}{3} = 37$

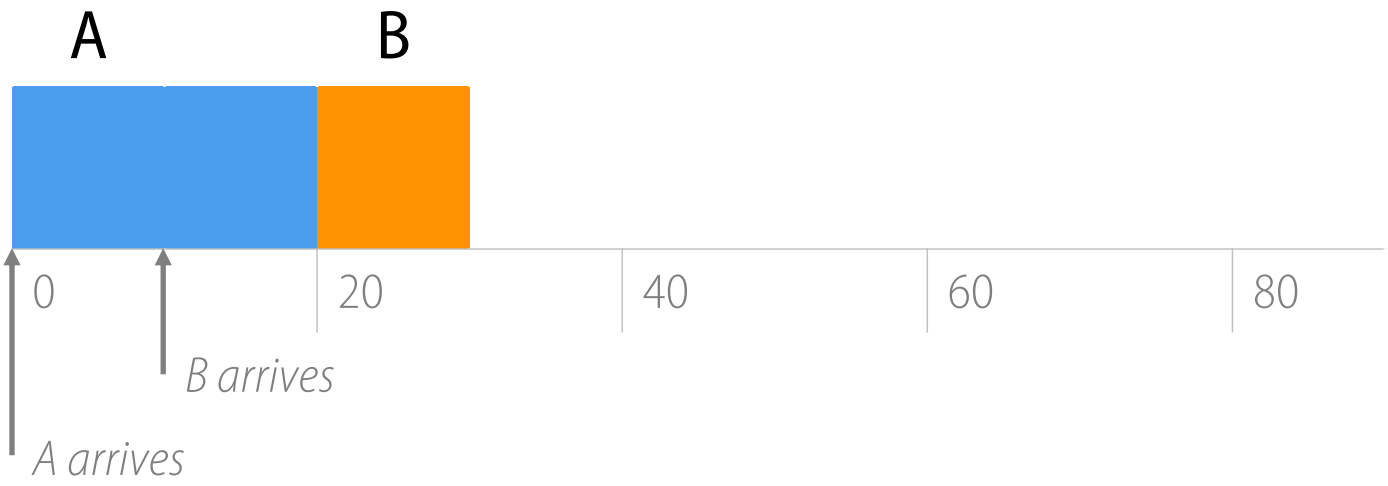
Why wouldn't STCF be our solution of choice?

Response time

- Sometimes care about when a job starts instead of when it finishes…
- Response time
  - Time until the job is scheduled for the first time
  - $T_{response} = T_{firstrun} - T_{arrival}$
- STCF and related techniques are not particularly good for response time.
- **How can we build a scheduler that is sensitive to response time?**

Response vs. Turnaround

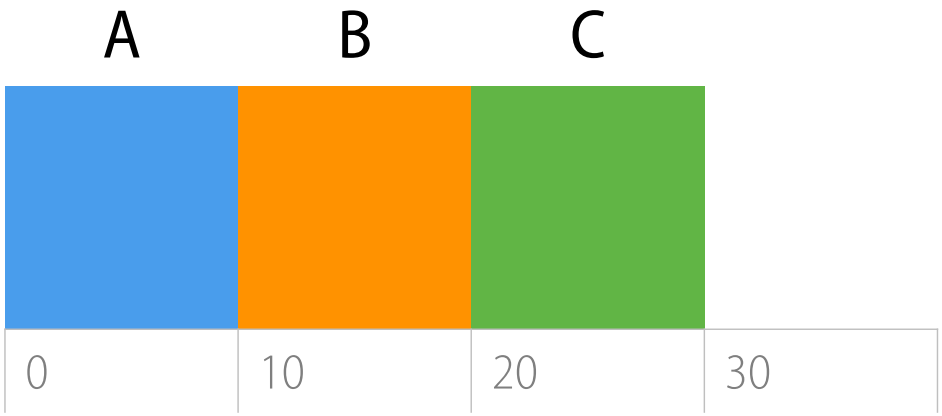
- In the non-preemptive example below…
  - B's turnaround time is…
    - $T_{turnaround} = T_{completion} - T_{arrival} = 30 - 10 = 20$
  - B's response time is…
    - $T_{response} = T_{firstrun} - T_{arrival} = 20 - 10 = 10$



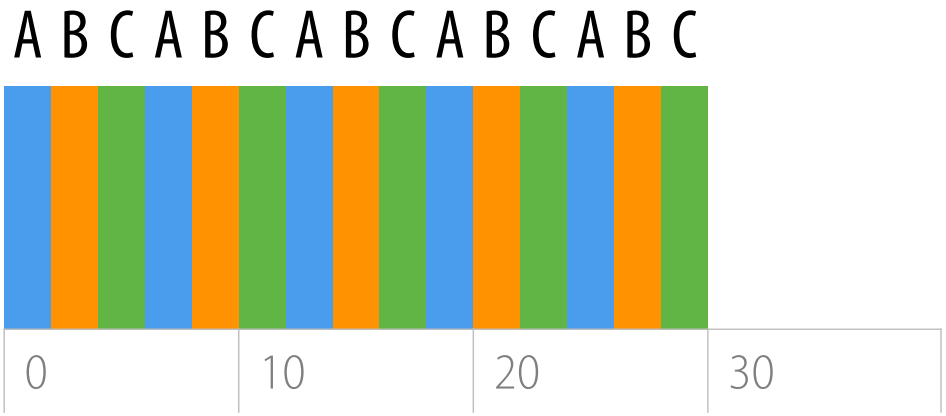
Caring about response time

- What was the problem with FIFO, SJF and STCF?
  - They may show poor response time.
- Let's try a new scheduler: **RR (Round Robin)**
  - Run a job for a time slice and then switch to the next job in the ready queue until all jobs are finished.
    - Time slice is sometimes called a *scheduling quantum*.
  - The length of the time slice must be *a multiple of* the timer-interrupt period.

FIFO vs. RR (simultaneous identical jobs)



$T_{response} = (0 + 10 + 20)/3 = 10$

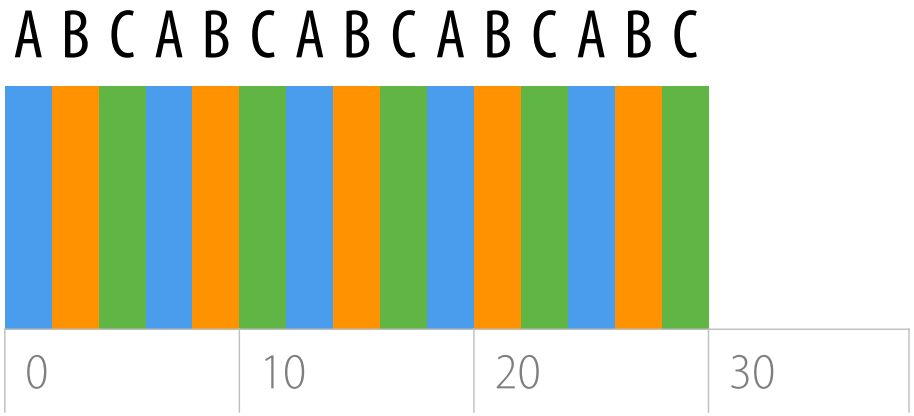


$T_{response} = (0 + 2 + 4)/3 = 2$

Why wouldn't RR be our solution of choice?

In what ways isn't RR ideal?

- RR is fair, but performs poorly on metrics such as turnaround time.
  - Let's calculate the average turnaround time for both options in our last example.



$$avgT_{response} = (0 + 10 + 20)/3 = 10$$
$$avgT_{turnaround} = \\ = \big((10 - 0) + (20 - 0) + (30 - 0)\big)/3 = 20$$

$$avgT_{response} = (0 + 2 + 4)/3 = 2$$
$$avgT_{turnaround} = \\ = \big((26 - 0) + (28 - 0) + (30 - 0)\big)/3 = 28$$

The length of the time slice is critical

- The shorter the time slice . . .
  - . . . the better the response time.
  - . . . the higher the relative cost of context switching.
- The longer time slice. . .
  - . . . the less noticeable the increase in the cost of switching.
  - . . . the worse the response time.

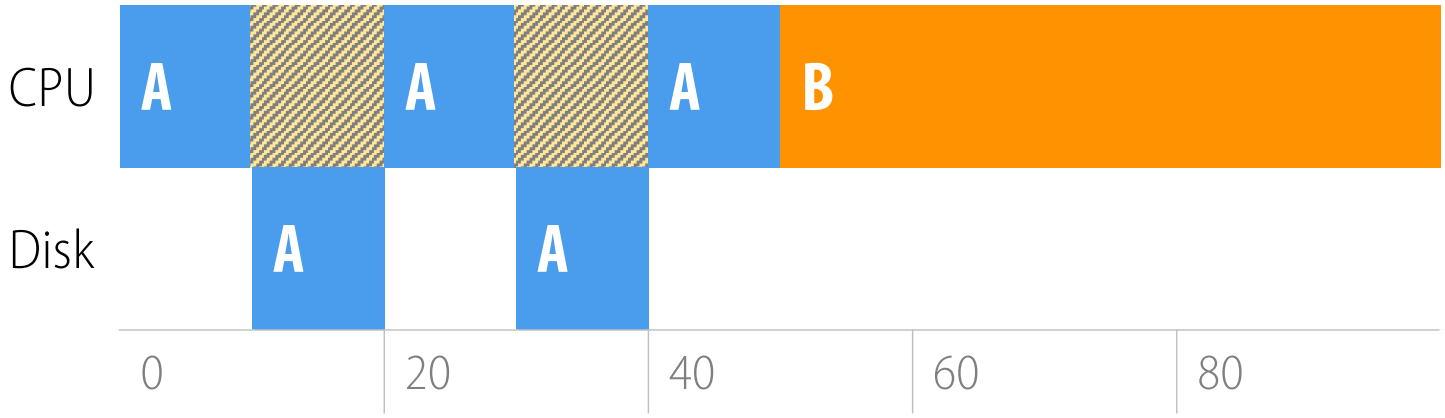
Relaxed Workload Assumptions

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

Incorporating I/O

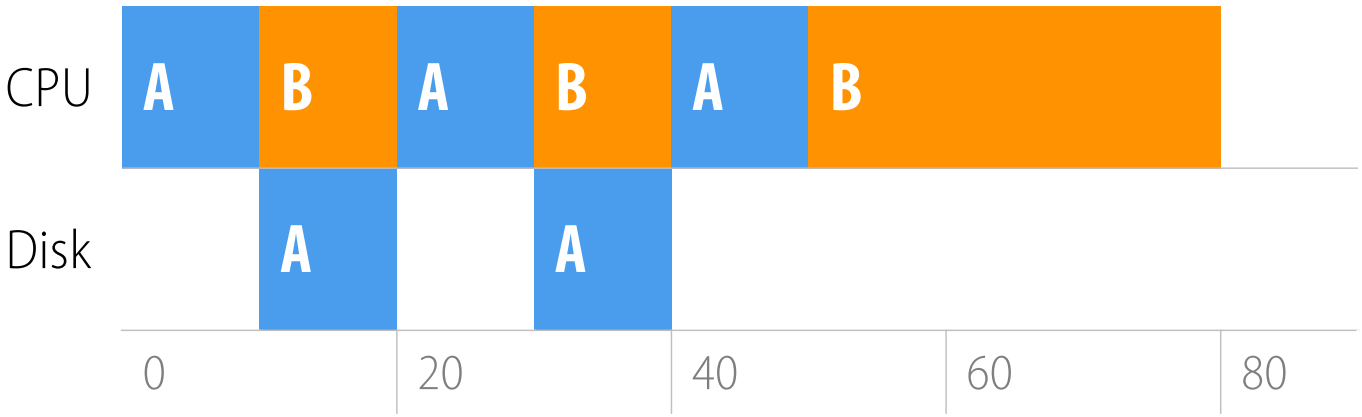
- Let's relax assumption 3: programs may perform I/O
- Example:
  - Jobs A and B need 50 units of CPU time each.
  - A runs for 10 units and then issues an I/O request
    - Each I/O operation takes 10 time units
  - B simply uses the CPU for 50 time units and performs no I/O
  - The scheduler runs A first, then B.

What happens when the scheduler isn't I/O aware?



- Resources may be poorly used.
- Job A holds on to CPU even while blocked waiting for disk.

Making the scheduler I/O aware



- Treat Job A as 3 separate CPU bursts.
- When Job A completes I/O, another Job A becomes ready.
- Each CPU burst is shorter than Job B, so even with SCTF, Job A preempts Job B.

Making the scheduler I/O aware

- 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 ready state.

Can we further relax our Workload Assumptions?

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

Well… that's a theme for next week.