Systems I – CS 6013

Computer Architecture and Operating Systems

Lecture 11: Scheduling

MASTER OF SOFTWARE DEVELOPMENT (MSD) PROGRAM

J. DAVISON DE ST. GERMAIN

SPRING 2024

*Adapted previous MSD slides

Lecture 11 – Topics

- Scheduling
  - FIFO, SJF, STCF, RR

Quiz (21 Points Total)

1. (3 pts) In high-level languages, what 3 things are necessary to declare a function?
   - What is defining a function vs declaring?
2. In assembly, a function (at a very high level) consists of 3 parts… what are they?
   - (3 pts) 1-2 words for each section
   - (5 pts) What do each of those sections do? (1-3 lines to answer each section)
3. (2 pts) List at least two things that a regular user's program cannot do without OS support?
4. (2 pts) How can a user's program do those things – write down the assembly command that allows this (and what information it needs to do its job)
5. (6 pts) What happens with respect to the OS/hardware when a user's program does the above?

- Slack'ers… send answers when we are done

Miscellaneous

- Reminder - Week 4 Vocabulary Assignment
  - Due Mon, Feb 5
- Start on your `dshell`...
  - What does the "d" stand for, you ask?
  - Davison

Other Shells?

- ashell – Aiden
- bshell – Brian, Ben
- cshell – Chen, Corinne
- eshell - Elisabeth
- jshell – Jessica, Jake*, Josh
- kshell – Koffi
- lshell – Lindsay
- mshell – Melanie, Mina
- pshell – Phillip
- rshell – Rylie, Ray
- sshell – Sami, Sohan, Sarah, Shane
- tshell – Tailang, Tina
- xshell – Xuan, Xiyao, Xiaohan
- yshell – Yuyao, Yijun
- zshell – Zach

Review

- We discussed fork(), exec(), wait().
- We talked about LDE and the challenges therein.
  - First challenge: restricting access to sensitive state of the CPU.
    - Solution: Privilege Rings.
  - Second challenge: preventing denial of service without sacrificing efficiency.
    - Solution: Context switch and scheduling (using traps/interrupts).
- Multiprocessing
  - Separate mechanism (how to context switch)
  - From policy (when to context switch)
- Today: Dive into process scheduling (policy).

Context Switching

- Process A is running.
  - Save process A registers to Process Control Block (PCB)
  - Load process B's PCB into registers
- Process B is running
- 1000-1500 cycles, ~1-2 `uS`
  - What is a `uS`?
    - Microsecond… which is how long?
    - 1 millionth of a second

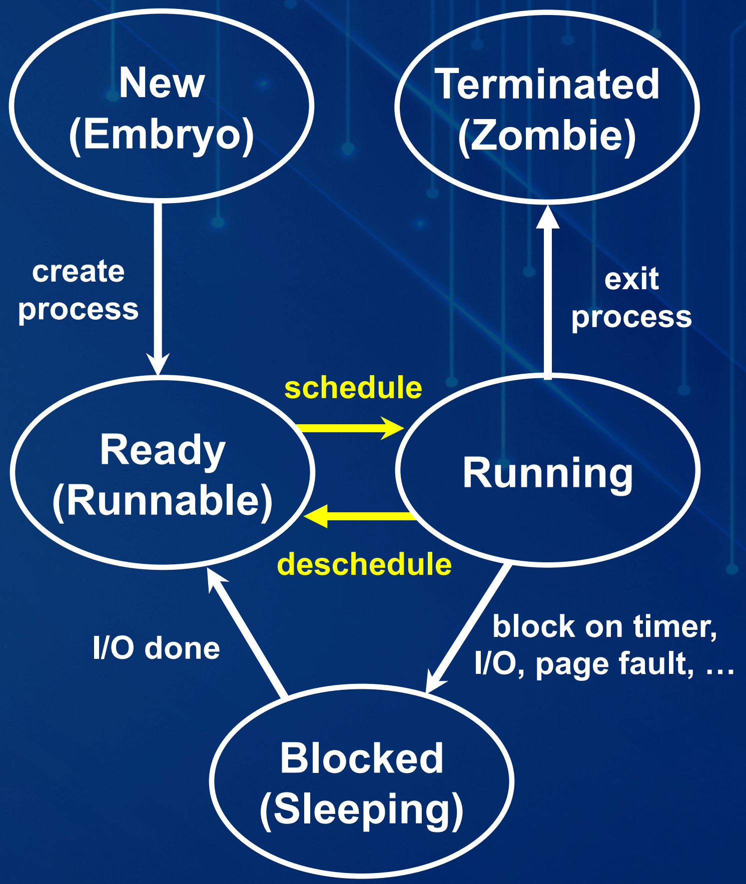
- What is a cycle?
  - In general, the amount of time it takes to execute one instruction*
- How many cycles on a modern CPU?
  - Running at ~3 GHz
  - 3 billion

CPU Virtualization: Multiprocessing

- Dispatcher – so far, we've covered:
  - Low-level mechanism (The *how*)
  - 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
  - Policy to determine which process gets CPU when

Review: Process State

- Recall the FSA for processes…
  - What is a FSA?
    - Finite State Automata
  - Which means?
    - A process (or anything that is represented by an FSA) can only be in one state at a time.
- Scheduling is moving a *Ready* process from the Ready Queue to start running on the CPU.
  - What happens to the process that was running?
    - Put back on the Ready Queue
  - Other ways for a process to be removed from the running state (and thus a different process getting to run)?
    - Running process blocks due to I/O, etc



Vocabulary

- Workload: set of job descriptions (arrival time, run time)
  - Burst – number of cycles a process stays on the CPU before needing I/O
  - Processes alternates between CPU and I/O
  - Processes move between ready and blocked queues
- Scheduler: logic that decides which ready job to run
- Metric: measurement of scheduling quality

Scheduling Performance Metrics

- Minimize turnaround time
  - Do not want to wait long for job to complete
  - Completion time – arrival time
- Minimize response time
  - Schedule interactive jobs promptly so users see output quickly
  - Initial schedule time – arrival time
- Minimize waiting time
  - Do not want to 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
  - If we were always doing context switches, no real work would happen - Thrashing
- Maximize fairness
  - All jobs get same amount of CPU over some time interval

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 in advance

Scheduling Basics

Workloads:
- Arrival Time
- Run Time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround time
- response time

* First In, First Out; Shortest Job First; Shortest Time to Completion First; Round Robin

Example: Workload, Scheduler, Metric

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

- FIFO: First In, First Out
  - also called FCFS (first come first served)
  - run jobs in *arrival_time* order

What is our turnaround? *completion_time - arrival_time*

FIFO: Event Trace

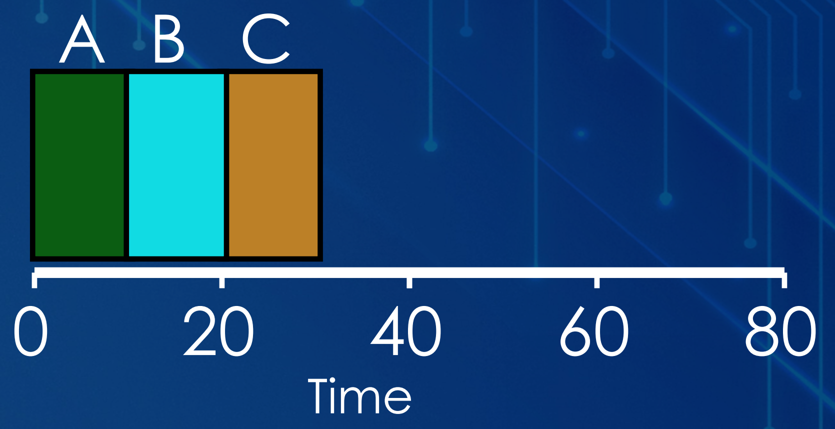
JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10

- What is the ***completion*** time of A, B, and C?
  - A – 10 seconds
  - B – 20 seconds
  - C – 30 seconds
- Run time is different from completion time…
  - Note: "turnaround time" is the same thing as completion time.

Time	Event
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

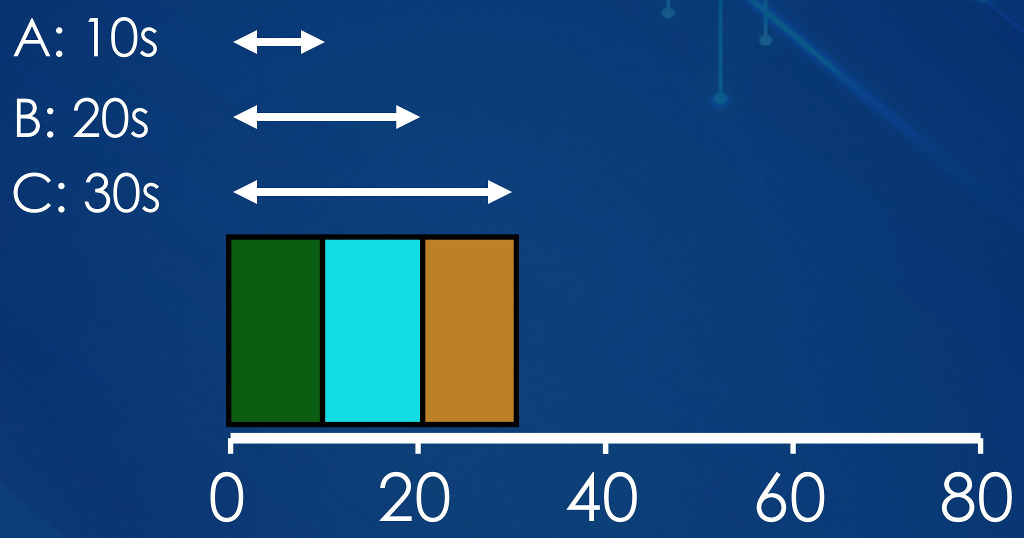
FIFO - Visualization

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



Gantt chart: Illustrates how jobs are scheduled over time on a CPU (this is a general type of chart used to illustrate how things are laid out over some time period, used across engineering/management/operations/finance)

FIFO – Turnaround Time?



- What is the *average* turnaround time?

  turnaround_time = completion_time - arrival_time

  (10 + 20 + 30) / 3 = 20s
- Each job has a run time of 10s, but the average turnaround is 20s…
- If you arrive late, you may wait a long time…

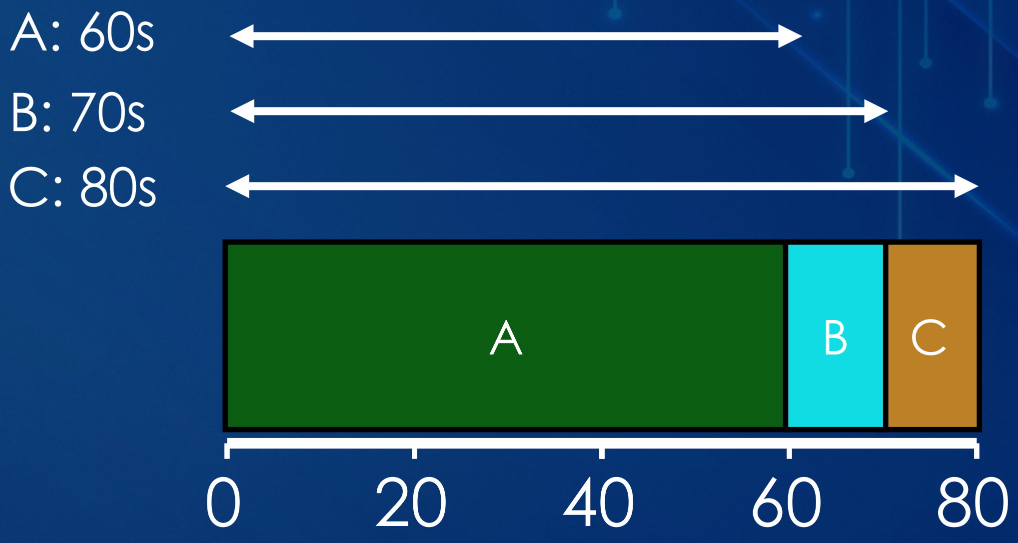
Workload Assumptions

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. The run-time of each job is known in advance

Counterexample: Big First Job

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

- FIFO Disadvantage
  - Can be unfair to short jobs!



Average turnaround time: 70 s

Convoy Effect

Passing the Tractor

- Problem with Previous Scheduler:
  - FIFO: Turnaround time suffers when short jobs wait behind long jobs

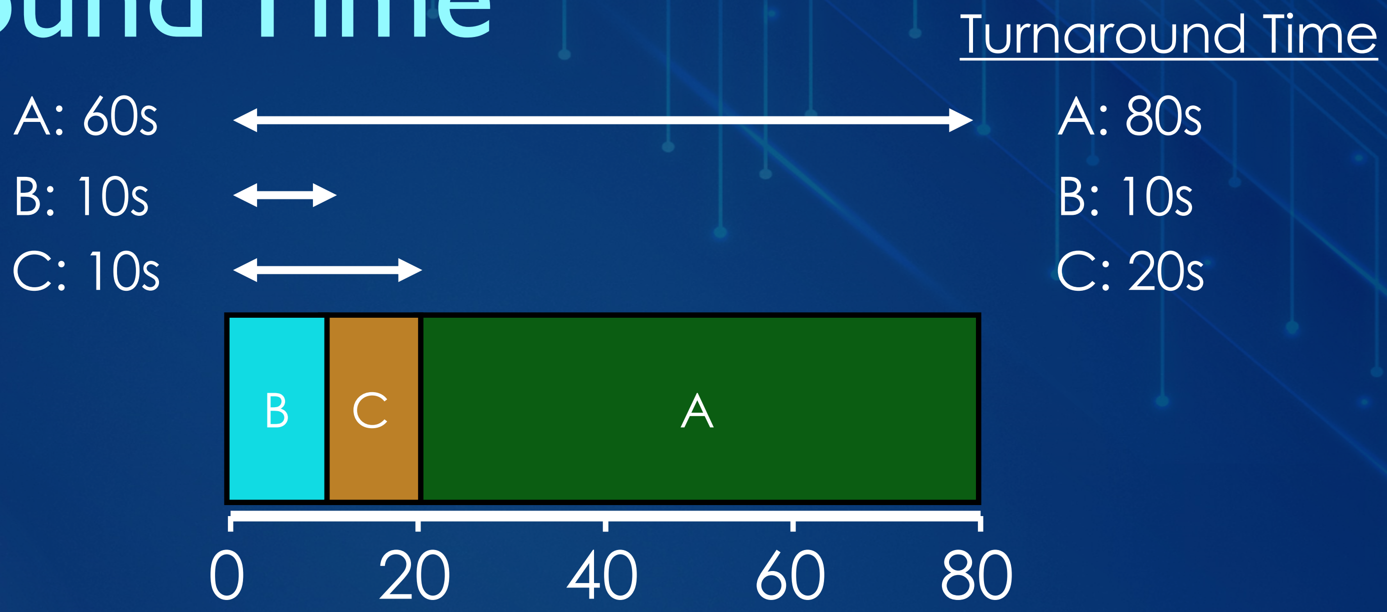
- New scheduler:
  - SJF (Shortest Job First)
  - Choose job with shortest *run_time**

Shortest Job First

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

What is the average turnaround time with SJF?

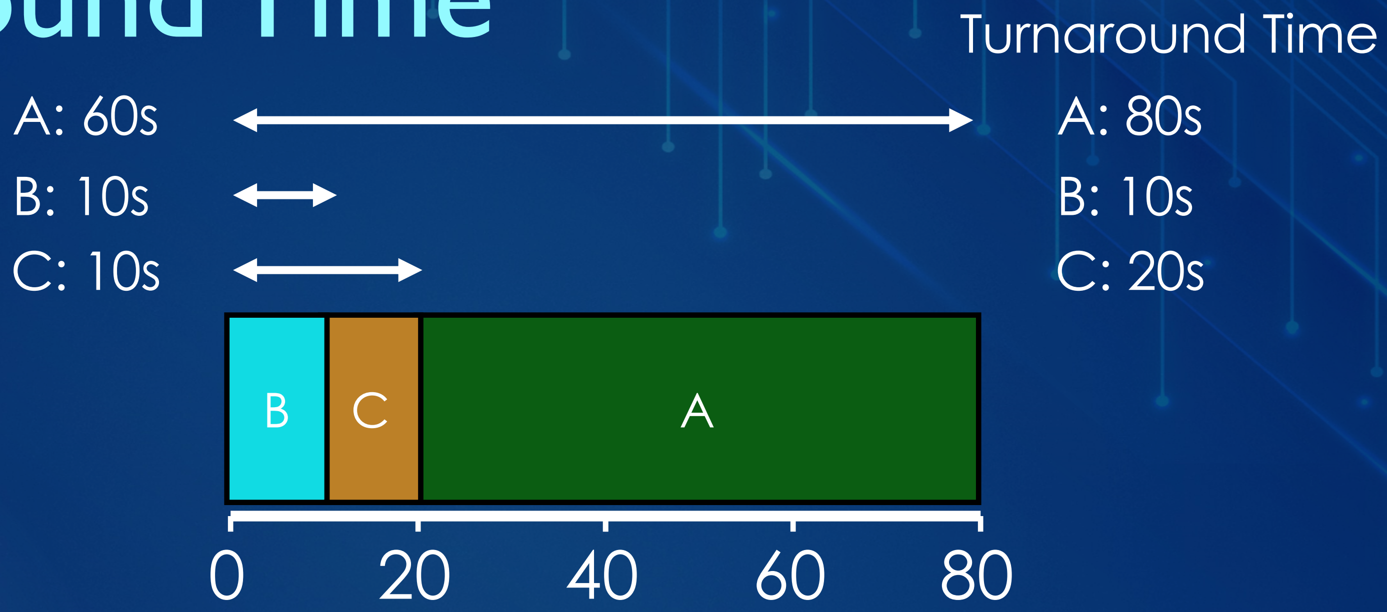
SJF Turnaround Time



- What is the average turnaround time with SJF?

  (80 + 10 + 20) / 3 = ~36.7s
- Note: Average turnaround with FIFO: 70s
- SJF optimal in minimizing turnaround time (with no preemption)
- Shorter job before longer job improves turnaround time of short, more than it harms turnaround time of long

SJF Turnaround Time



- Disadvantages of SJF?
  - What happens if short jobs keep showing up?
    - Very long turnaround times for some jobs, or even starvation!
  - Can't practically use in a real system because?
    - Don't know the run times of jobs in advance.

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. The run-time of each job is known

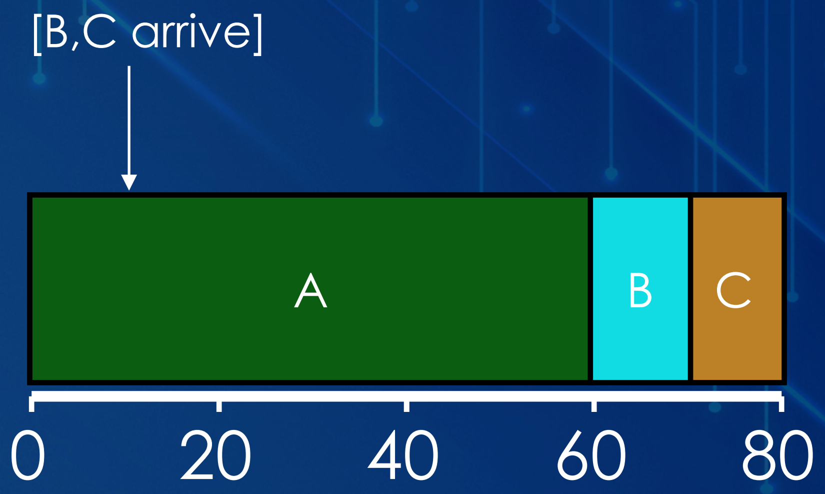
Shortest Job First (Arrival Time)

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

What is the average turnaround time with SJF?

Stuck Behind a Tractor Again

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



What job runs first?
What is the average turnaround time?

(60 + (70 – 10) + (80 – 10)) / 3 = 63.3s

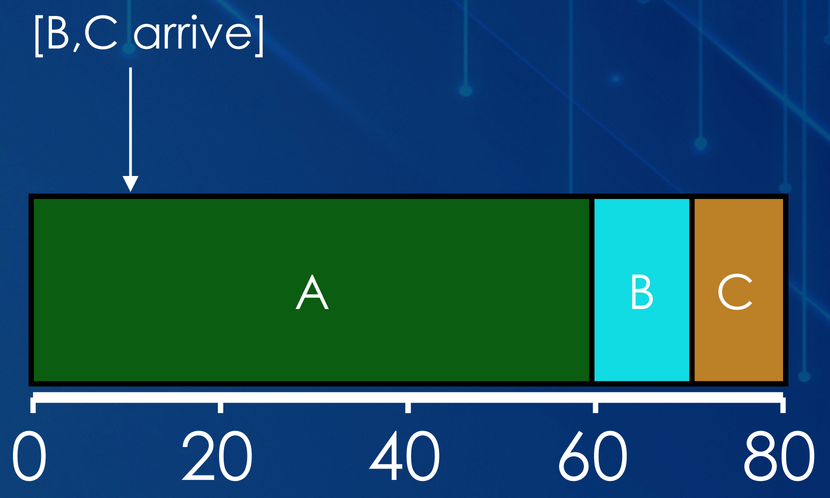
Assumes no preemption…

Preemptive Scheduling

- **Previous schedulers:**
  - FIFO and SJF are non-preemptive
  - Only schedule new job when previous job voluntarily relinquishes CPU (performs I/O or exits)
- **New scheduler schemes:**
  - Preemptive: Potentially schedule different job at any point by taking CPU away from running job
  - STCF (Shortest Time-to-Completion First)
  - Always run job that will complete the quickest

Non-Preemptive: SJF

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

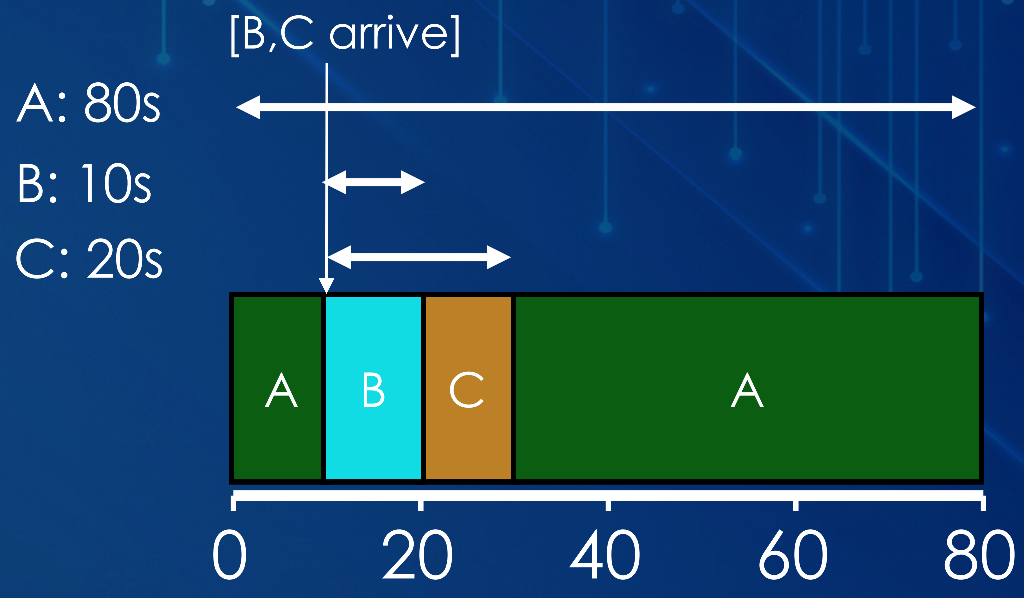


Average turnaround time:

(60 + (70 – 10) + (80 – 10)) / 3 = 63.3s

Preemptive: STCF (shortest time to completion)

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



Average turnaround time with STCF? 36.7s

Average turnaround time with SJF: 63.3s

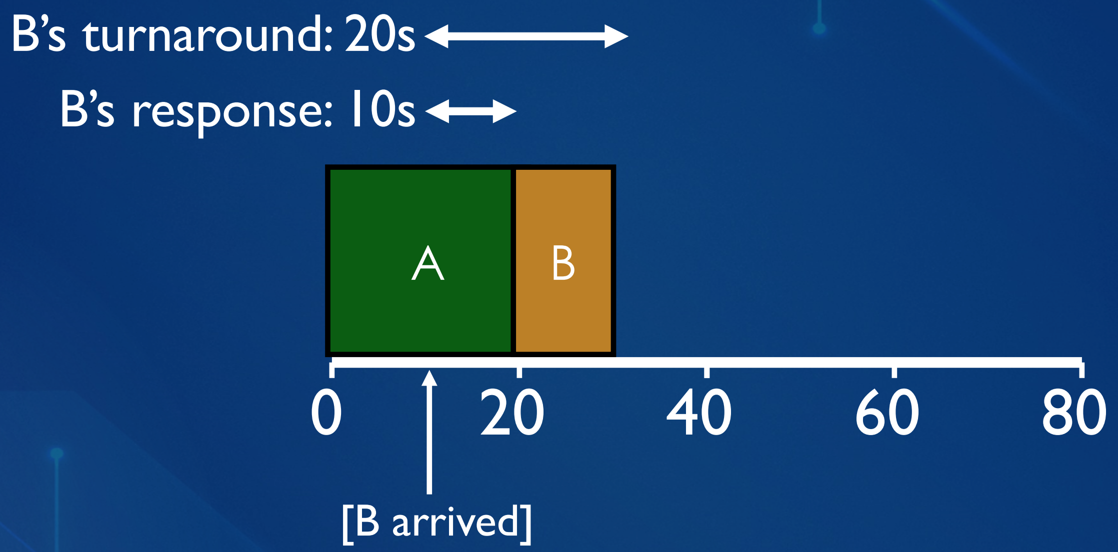
Response Time

- SCTF okay for batch systems, but
  - For time sharing systems, ***when*** a job completes is less important

- Sometimes we care about when a job ***starts*** instead of when it finishes.

- New metric:
  - response_time = first_run_time - arrival_time

Response vs. Turnaround Time

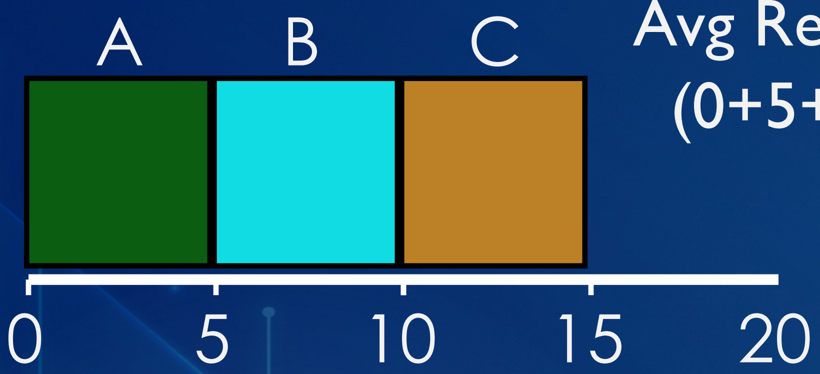


Round-Robin Scheduler

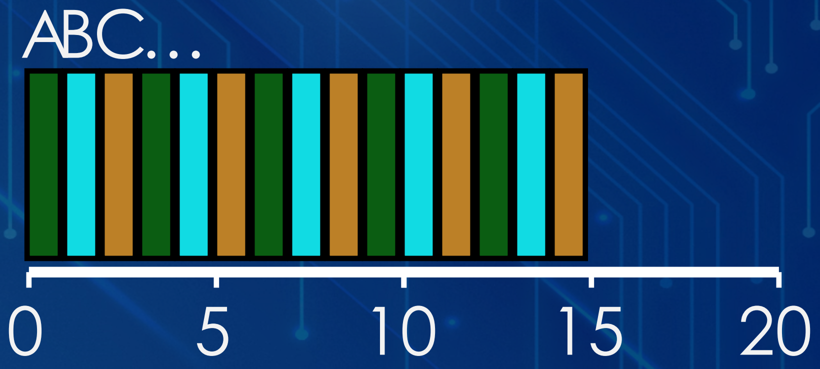
- **Previous schedulers:**
  - FIFO, SJF, and STCF can have poor response time

- **Next scheduler idea**: RR (Round Robin)
  - Alternate ready processes
  - Switch after fixed-length time-slice (or quantum)

FIFO vs Round-Robin



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



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

- Which scheduling algorithm is this?
- In what way is RR worse?
  - Avg turn-around time with equal job lengths is horrible
- Other reasons why RR could be better?
  - If run-times unknown, short jobs get chance to run, finish fast
  - Fair
- Note, preemptive multi-tasking has the major advantage that as a programmer, we can pretend the entire CPU is ours alone, and do not have to worry about trying to coordinate with other processes (assuming a CPU could actually run them concurrently)…

Scheduler

- Decides what process to run and when… So:
- Where does the scheduler itself run?
  - It is "just another program"
  - Runs on the same CPU as a user program.
    - Note: though in kernel mode. Why?
    - Needs to access kernel memory, and the memory of all other processes.

Summary

- Understand goals (metrics) and workload, then design scheduler around that
- General purpose schedulers need to support processes with different goals

~ Fin ~