RUTGERS UNIVERSITY
Computer Sciences Department

CS 416 Operating Systems Design

Sudarsun Kannan

CPU Virtualization: Scheduling

Questions answered in this lecture:

What are different scheduling policies, such as:
FCFS, SJF, STCF, RR and MLFQ?

What type of workload performs well with each scheduler?

What scheduler does Linux currently use?

Disclaimer: Materials derived, reused, and modified from OSTEP book and lectures of Prof. Andrea and Remzi Arpaci-Dusseau and Prof. Yojip Won

Announcements

- Reading:
  - Today cover Chapters 7-9
- Project 1: Warm-up with using C
  - Finish Part A, Part B, Part C by Feb 13th
  - Goal is for everyone to learn material
  - Do not copy code from others!
  - Indicate your group members in the code file (project1.c) as a comment and do not change

Midterm – March 5th – In-class

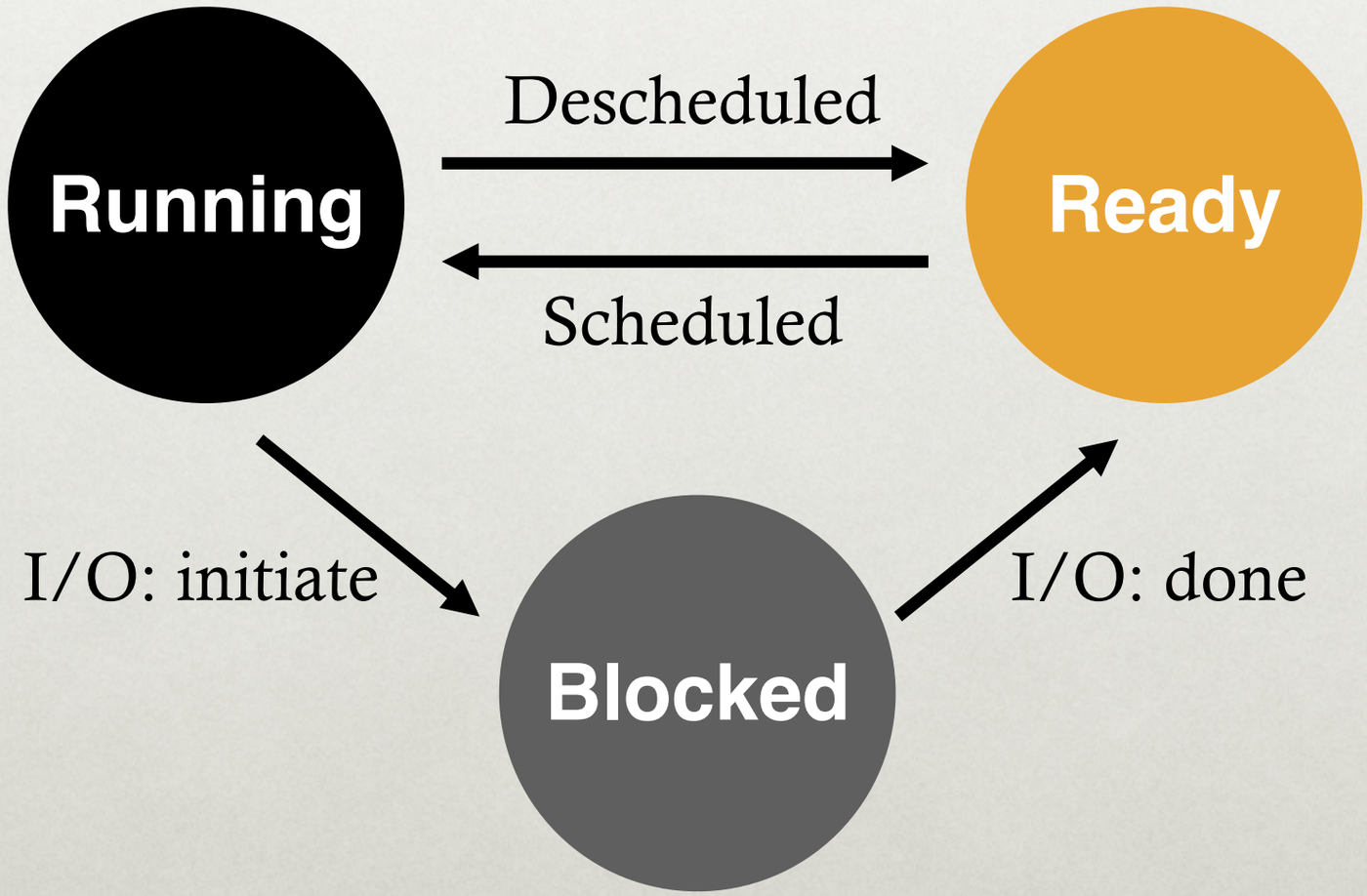
CPU Virtualization: Two Components

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)
  - Policy to determine which process gets CPU when

Review:
State Transitions



How to transition? ("mechanism")
When to transition? ("policy")

Vocabulary

Workload: set of **job** descriptions (arrival time, run_time)

- Job: View as current CPU burst of a process
- Process alternates between CPU and I/O
  process moves 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

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

Scheduling Basics

Workloads:

arrival_time
run_time

Schedulers:

FIFO
SJF
STCF
RR

Metrics:

turnaround_time
response_time

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*

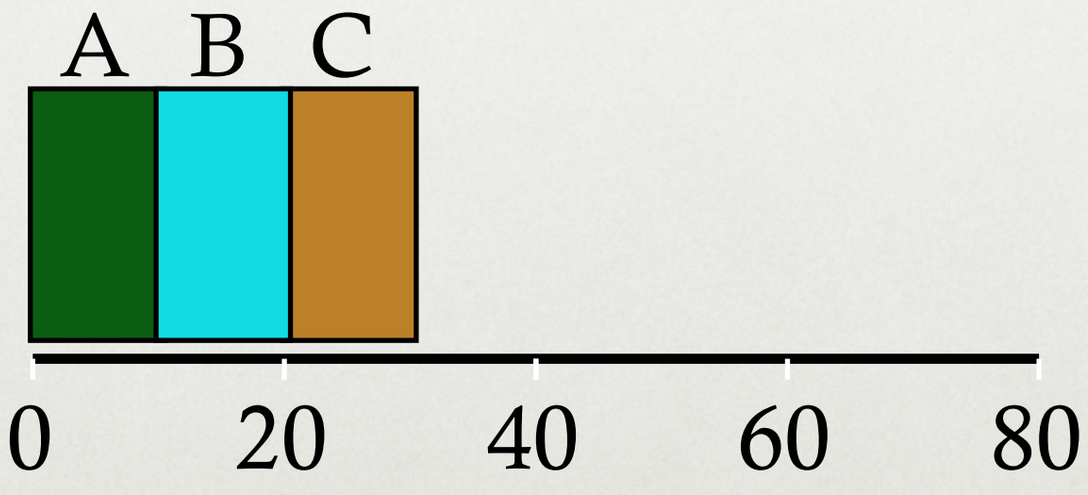
FIFO: Event Trace

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

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

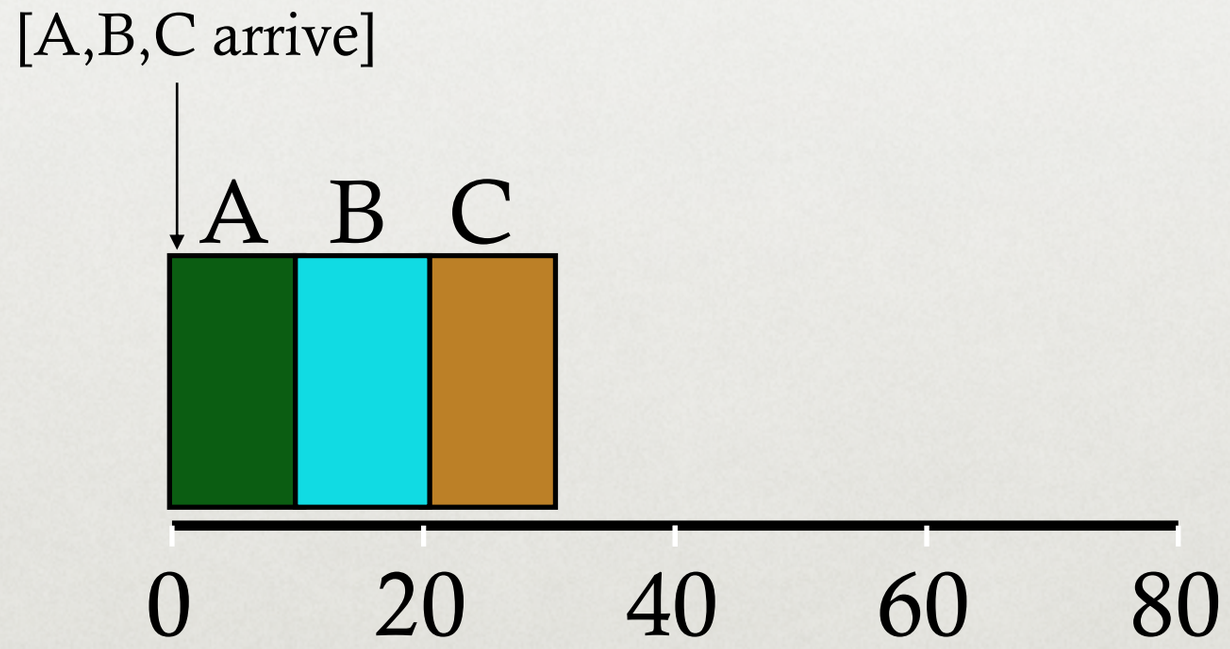
FIFO (Identical JOBS)

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

FIFO (IDENTICAL JOBS)



What is the average turnaround time?

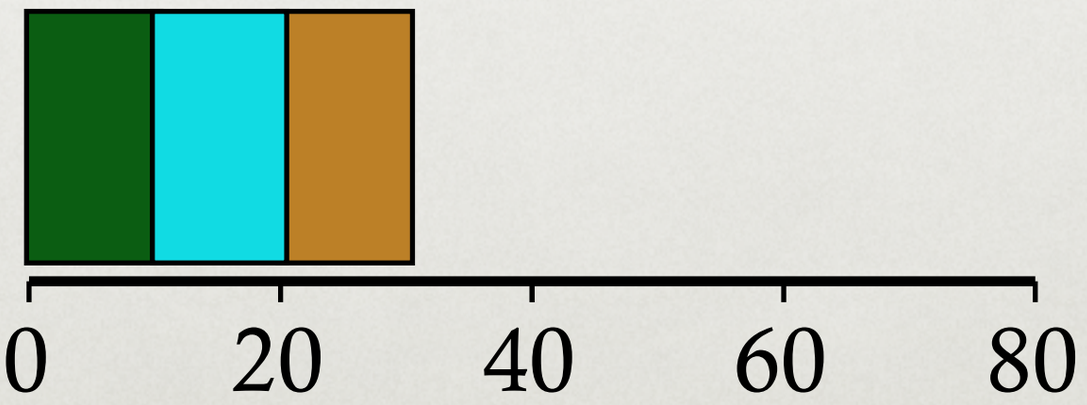
Def: *turnaround_time* = *completion_time - arrival_time*

FIFO (IDENTICAL Jobs)

A: 10s

B: 20s

C: 30s



What is the average turnaround time?

Def: *turnaround_time* = *completion_time - arrival_time*

(10 + 20 + 30) / 3 = **20s**

Scheduling Basics

Workloads:

arrival_time

run_time

Schedulers:

FIFO

SJF

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. The run-time of each job is known

Any Problematic Workloads for FIFO?

Workload: ?

Scheduler: FIFO

Metric: turnaround is high

Example: Big First Job

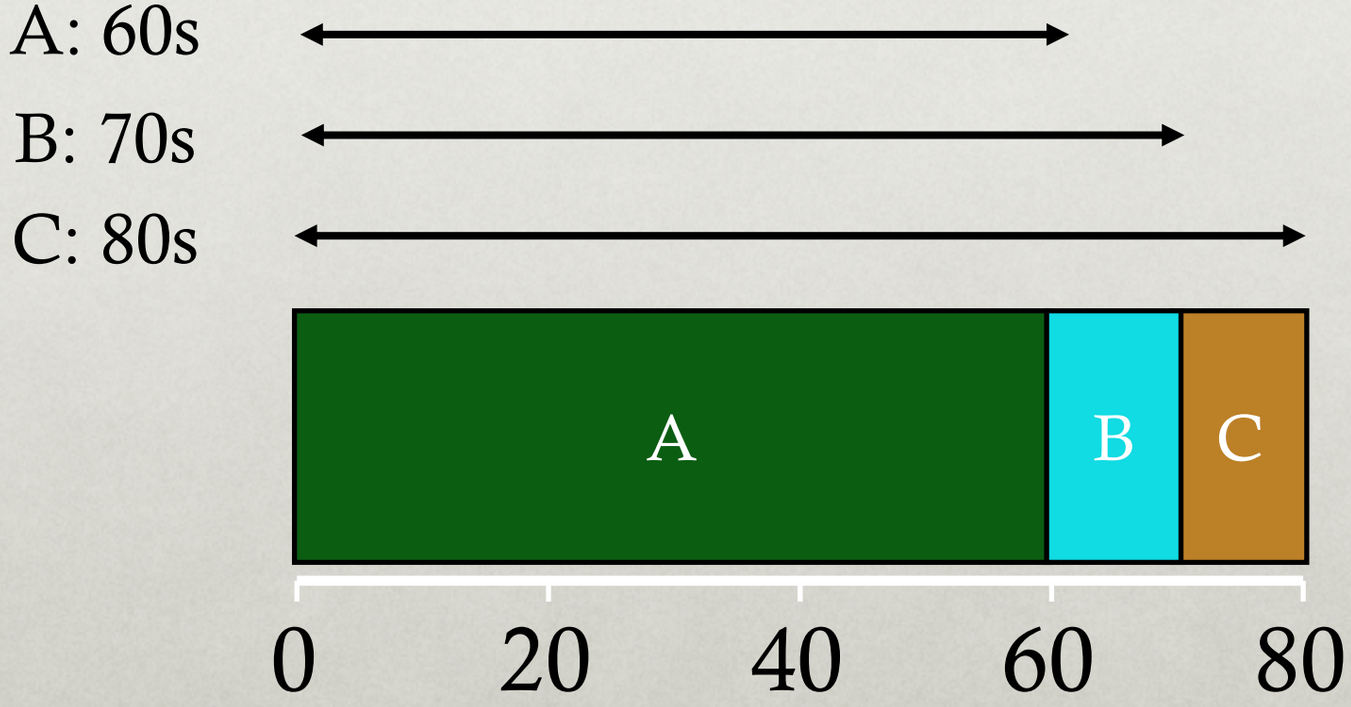
JOB	arrival_time (s)	run_time (s)
A	~0	**60**
B	~0	10
C	~0	10

Draw Gantt chart for this workload and policy…

What is the average turnaround time?

Example: Big First Job

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



Average turnaround time: **70s**

Convoy Effect

Passing the Tractor

Problem with Previous Scheduler:

FIFO: Turnaround time can suffer when short jobs must wait for long jobs

New scheduler:

SJF (Shortest Job First)

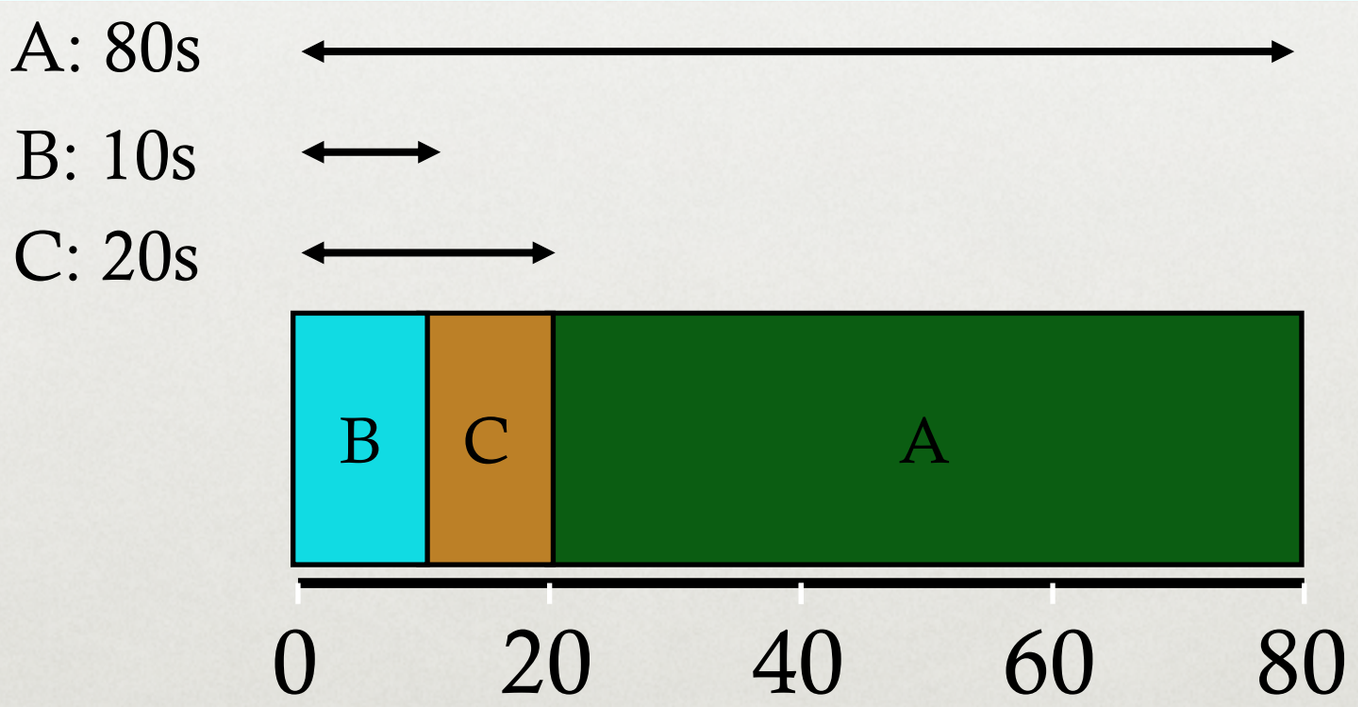
Choose job with smallest *run_time*

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**

Average turnaround with FIFO: 70s

For minimizing average turnaround time (with no preemption):
SJF is provably optimal

Moving shorter job before longer job improves turnaround time of short job more than it harms turnaround time of long job

Scheduling Basics

Workloads:

arrival_time

run_time

Schedulers:

FIFO

SJF

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)

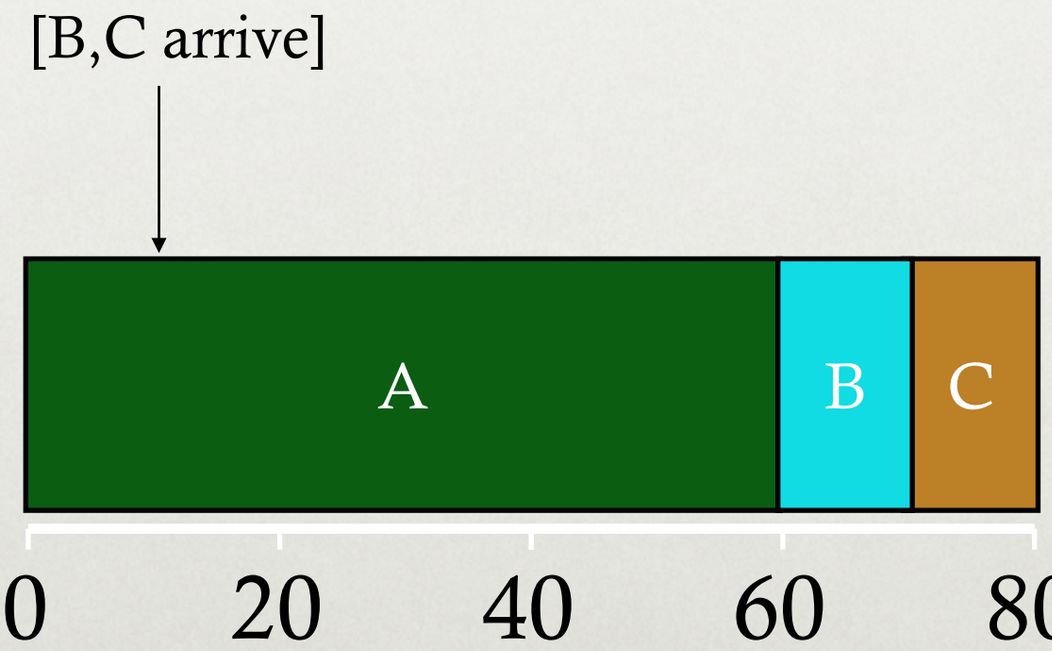
4. The run-time of each job is known

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 is the average turnaround time?

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

Preemptive Scheduling

Prev schedulers:

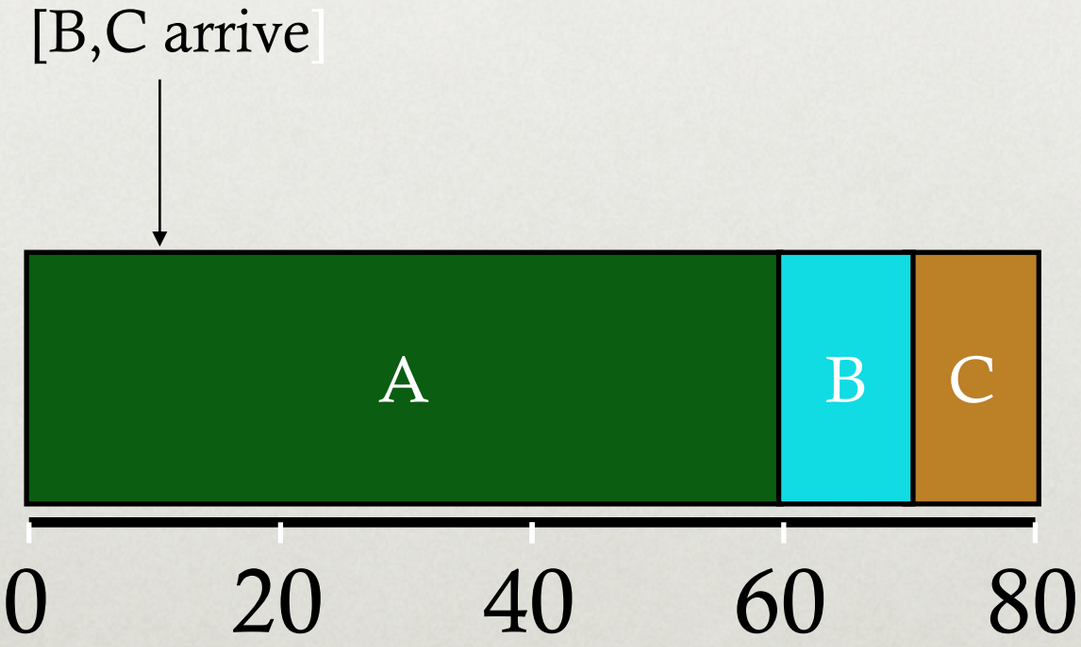
- FIFO and SJF are non-preemptive
- Only schedule new job when previous job voluntarily relinquishes CPU (performs I/O or exits)

New scheduler:

- 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**

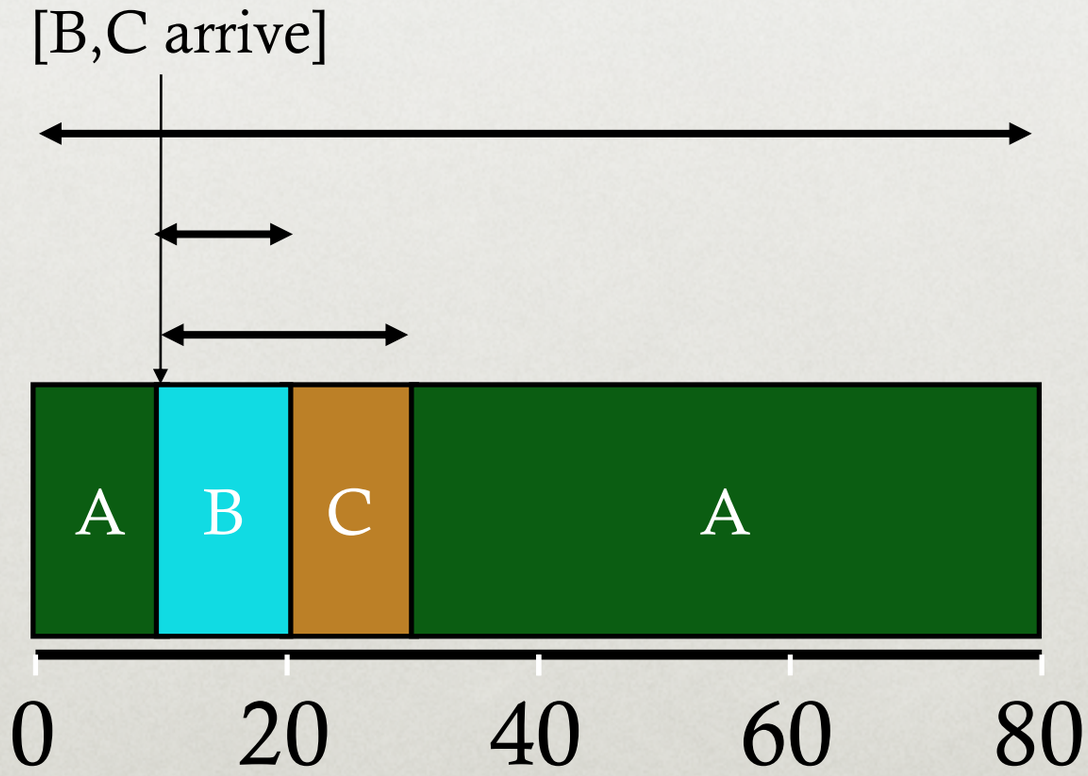
PREEMPTIVE: STCF

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

A: 80s

B: 10s

C: 20s



Average turnaround time with STCF?

36.6

Average turnaround time with SJF: **63.3s**

Scheduling Basics

Workloads:

arrival_time

run_time

Schedulers:

FIFO

SJF

STCF

RR

Metrics:

turnaround_time

response_time

Response Time

Sometimes care about when job starts instead of when it finishes

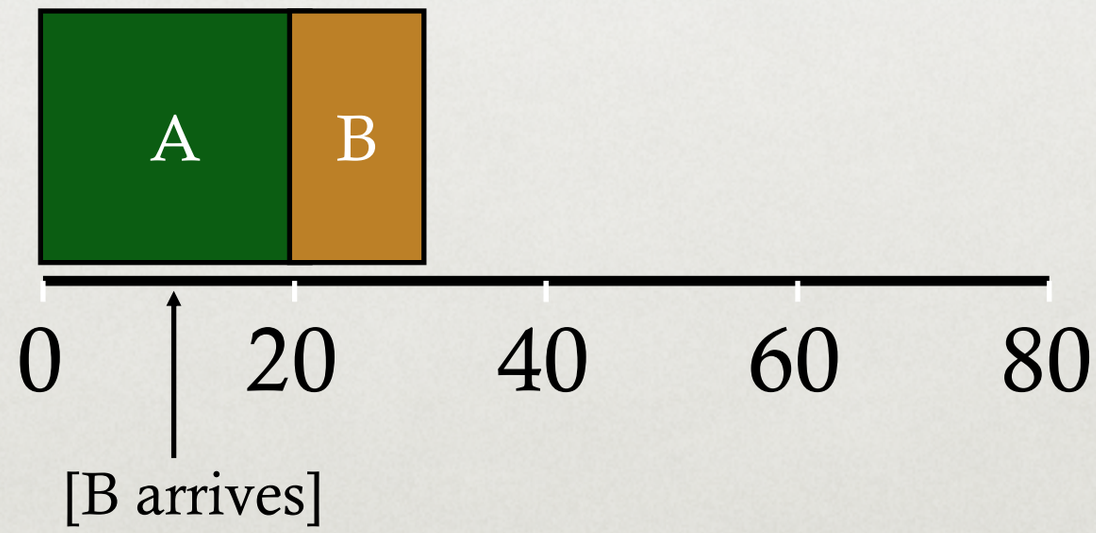
New metric:

response_time = *first_run_time* - *arrival_time*

Response vs. Turnaround

B's turnaround: 20s

B's response: 10s



Round-Robin Scheduler

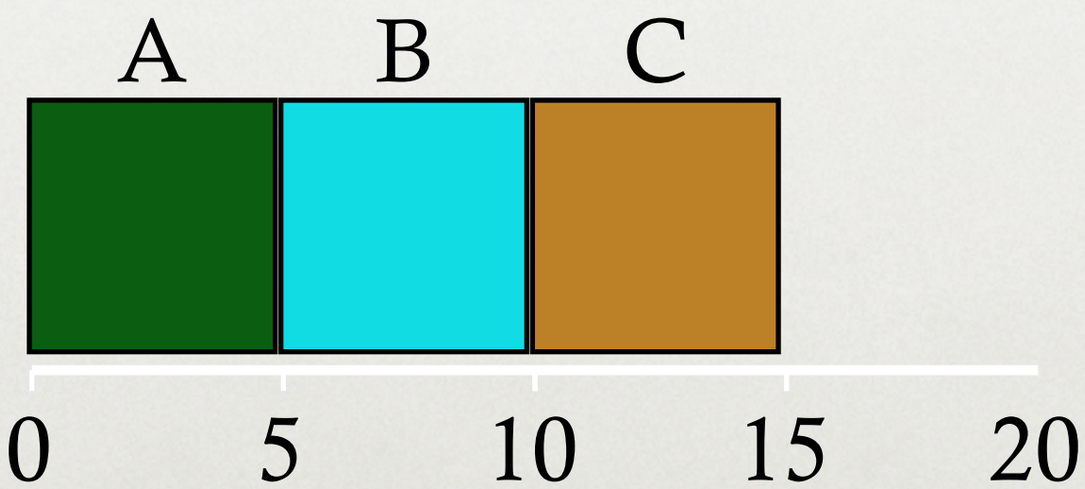
Prev schedulers:

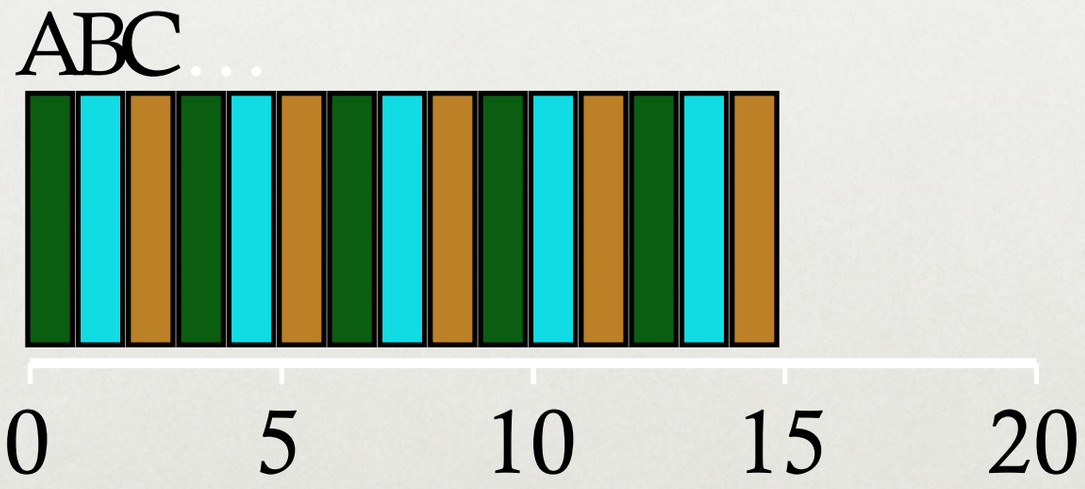
FIFO, SJF, and STCF can have poor response time

New scheduler: RR (Round Robin)

Alternate ready processes every fixed-length time-slice

FIFO vs RR





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

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

In what way is RR worse?
Ave. turn-around time with equal job lengths 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

Scheduling Basics

Workloads:

arrival_time

run_time

Schedulers:

FIFO

SJF

STCF

RR

Metrics:

turnaround_time

response_time

Summary

Understand goals (metrics) and workload, then design scheduler around that

General purpose schedulers need to support processes with different goals

Past behavior is good predictor of future behavior

Random algorithms (lottery scheduling) can be simple to implement, and avoid corner cases.