

CS 423  
Operating System Design  
CPU Scheduling Policies

Jan 27

Ram Kesavan

# Logistics

Next lecture: TAs will walkthrough MP0  
Everyone to use EngrIT VMs for MPs

4Cr: 1st paper has been added to course web-page  
Review due before start of Feb3rd class (2pm CST)

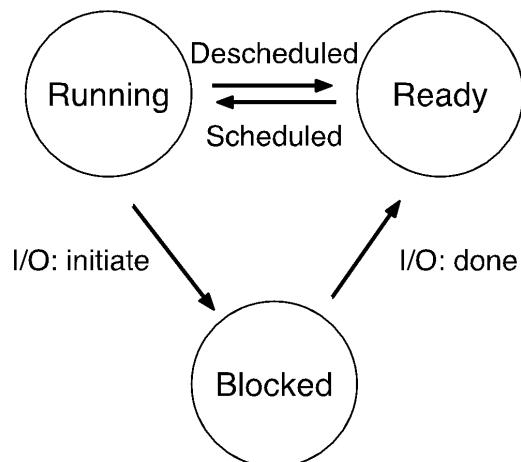
This lecture:

1. Recap scheduling mechanism
2. Scheduling policies

# RECAP: SCHEDULING MECHANISM

Process: abstraction to virtualize the CPU

Use time sharing to switch between processes



When and to which process to switch – policy

How to switch processes – mechanism

How many processes can be in each state simultaneously?

# RECAP: SCHEDULING MECHANISM

## Limited Direct Execution (LDE)

- Natively run the program on CPU

- Use system calls for restricted ops, such as accessing devices

- Use timer interrupt to grab control from user process

# RECAP: CPU STATE OF A PROCESS

# RECAP: SYSCALL

# LEARNING OUTCOMES

## Scheduling policies

How does the OS decide ***which*** process to run?

What are some of the metrics to optimize for?

What are some of the policies: FCFS, SJF, STCF, RR, MLFQ

What to do when OS doesn't have complete information?

How to handle mix of interactive and batch processes?

# Terminology

**Workload:** set of **jobs** (arrival time, run\_time)

**Job:** process that runs for some time period

Processes move between READY & BLOCKED

**Scheduler:** Decide which READY job to run

**Metric:** measure of scheduling quality (e.g., turnaround time, response time, fairness)

# APPROACH

Assumptions  
about the  
workload

Pick one or  
more  
Metric(s)

Scheduling  
policy

# METRIC 1: TURNAROUND TIME

Turnaround time = *completion\_time - arrival\_time*

Example:

Process A arrives at time t = 10, finishes t = 30

Process B arrives at time t = 10, finishes t = 50

Turnaround time

A = 20, B = 40

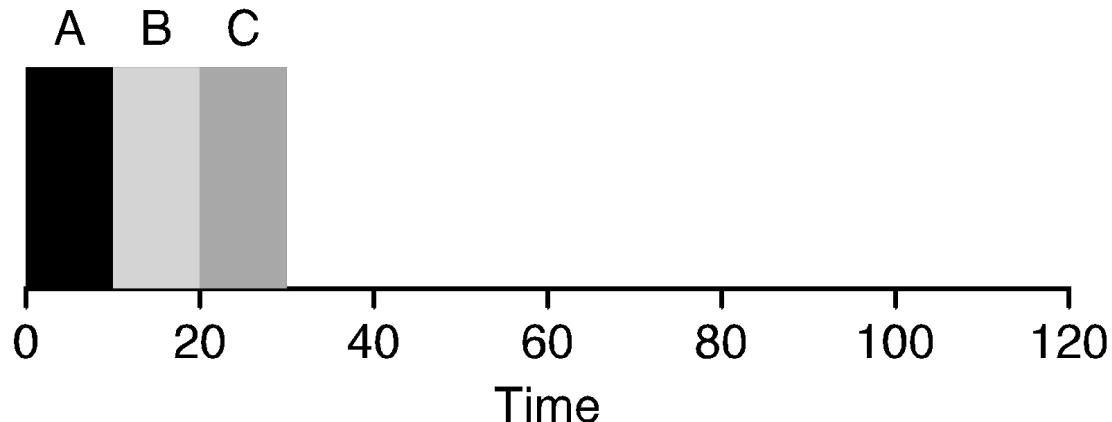
Average = 30

# ASSUMPTIONS

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

# FIFO / FCFS

Job	arrival(s)	run time (s)	turnaround (s)
A	~0	10	
B	~0	10	
C	~0	10	



Average  
Turnaround  
Time =

# ASSUMPTIONS

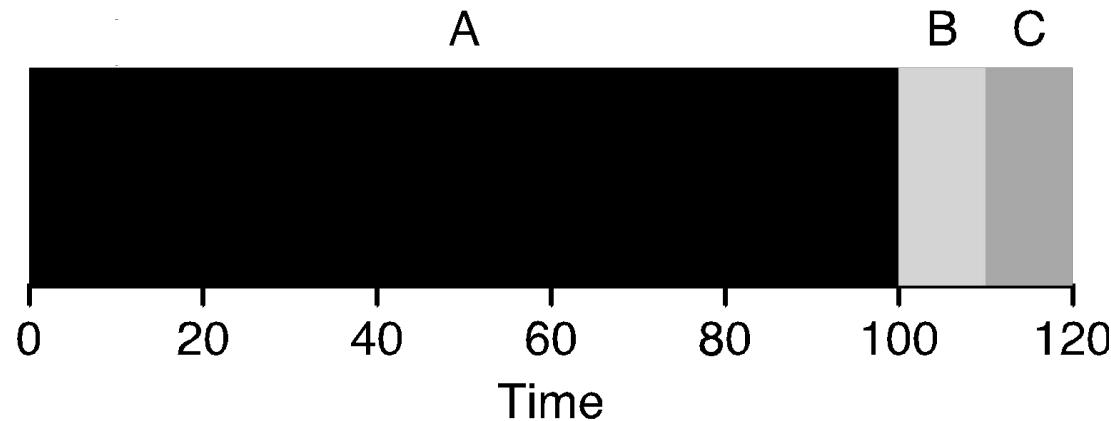
1. ~~Each job runs for the same amount of time~~
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use the CPU (no I/O)
5. Run-time of each job is known (perfect knowledge)

How will FIFO perform without this assumption?

# When FIFO is Bad

Job	Arrival(s)	run time (s)
A	~0	100
B	~0	10
C	~0	10

Average Turnaround Time?



What is one schedule that could be better?

# CHALLENGE

Turnaround time suffers when short jobs must wait for long jobs

Convoy effect: e.g., everyone slowed down by a slow vehicle in a single-lane road

New scheduler:

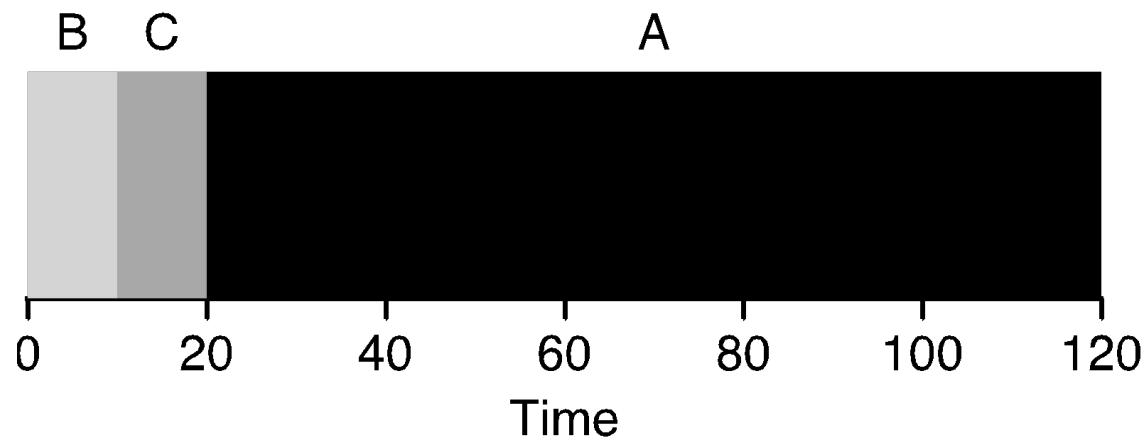
SJF (Shortest Job First)

Choose job with shortest run-time!

(note: we assume the OS knows the time for each job)

# SHORTEST JOB FIRST (SJF)

Job	Arrival(s)	run time (s)	Turnaround (s)
A	~0	100	
B	~0	10	
C	~0	10	



Average  
Turnaround  
Time

# SJF Theory

SJF is provably optimal for minimizing turnaround time (assuming no preemption)

Intuition:

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

# ASSUMPTIONS

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

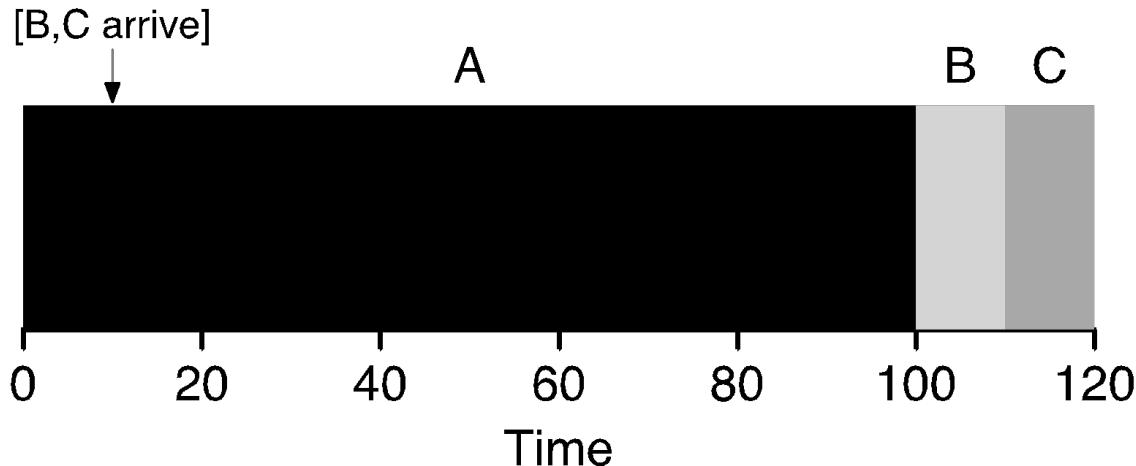
Job	Arrival(s)	run time (s)
A	~0	100
B	10	10
C	10	10

What will be the schedule with SJF?

Take a min, chat with neighbors, and sketch the schedule and calculate the turnaround time

Average  
Turnaround  
Time ?

Job	Arrival(s)	run time (s)
A	~0	100
B	10	10
C	10	10



TA time(A):  
TA time(B):  
TA time(C):

Avg = ?

# PREEMPTIVE SCHEDULING

Previous schedulers:

FIFO and SJF are non-preemptive

Only schedule new job when previous job voluntarily relinquishes CPU  
(after it exits)

Preemptive: Schedule different job by taking CPU away from running job

STCF (Shortest Time-to-Completion First) - preemptive version of SJF

Always run job that will complete the quickest

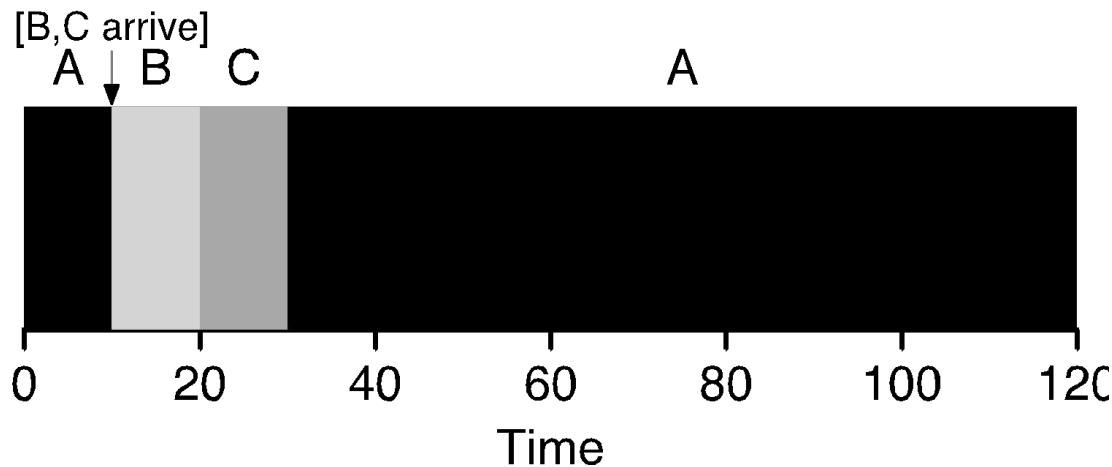
# ASSUMPTIONS

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

# PREEMPTIVE STCF

Job	Arrival(s)	run time (s)
A	~0	100
B	10	10
C	10	10

Average Turnaround Time



$$(10 + 20 + 120) / 3 = 50\text{s}$$

# PREEMPTIVE STCF

Job	Arrival(s)	run time (s)
A	~0	15
B	10	10
C	10	10

What is the schedule and turnaround time here?

# ASSUMPTIONS

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

# NOT IO AWARE

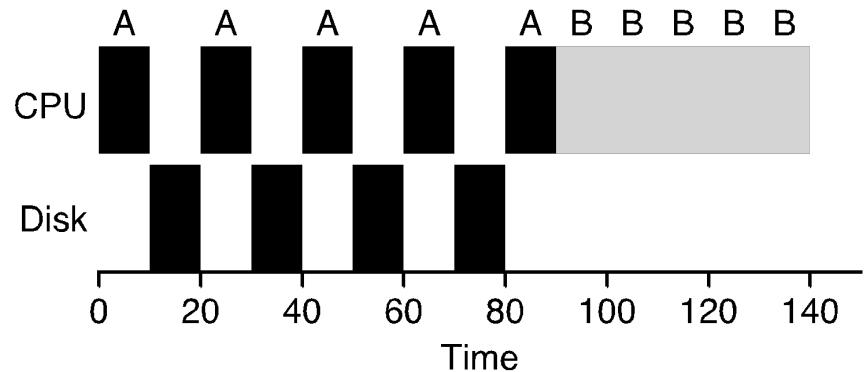
Assume A and B need 50ms compute each

But...A issues an IO (that takes 10ms) after running for 10ms

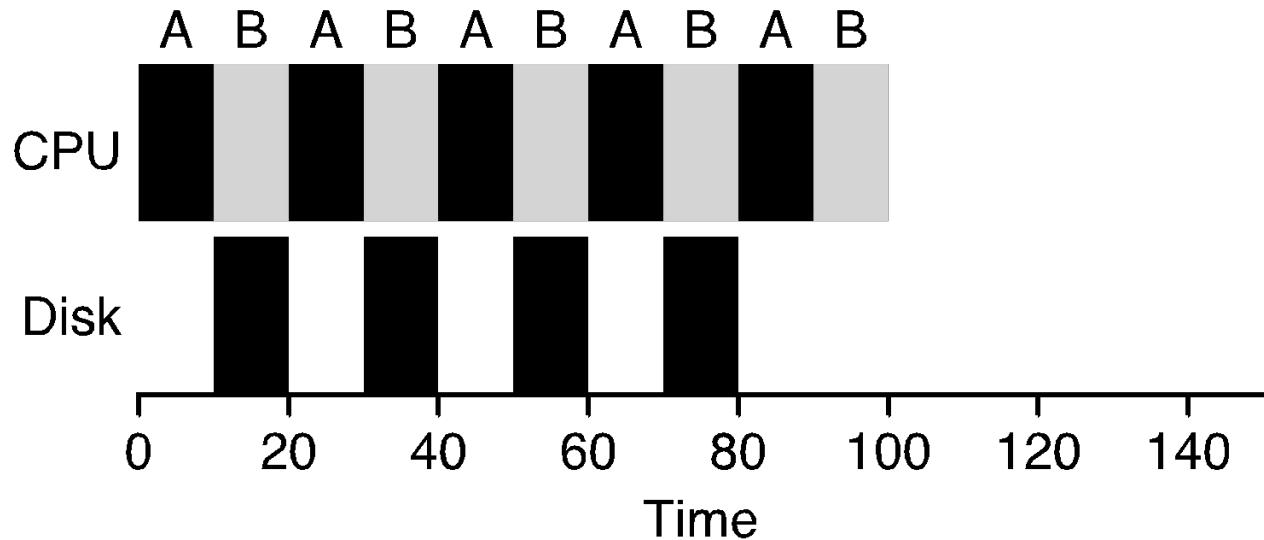
B issues no IOs

STCF: if not IO-aware, can pick A initially

If you consider the entire process A  
as a single job



# I/O AWARE SCHEDULING



Each 10ms  
sub-job of A is  
shorter than Job B

With STCF,  
Job A preempts Job B

Treat Job A as 5 separate 10ms sub-jobs

When one sub-job completes I/O, another sub-job is ready

# ASSUMPTIONS

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. ~~Once started, each job runs to completion~~
4. ~~All jobs only use the CPU (no I/O)~~
5. ~~Run time of each job is known (perfect knowledge)~~

# What if you don't know job runtime?

For metric of turnaround time:

If jobs have same length:

Then FIFO works well

If jobs have different lengths:

SJF is better than FIFO

Key question: How can OS get short jobs to complete first if OS doesn't know which are short?

# Solution: Round Robin (RR)

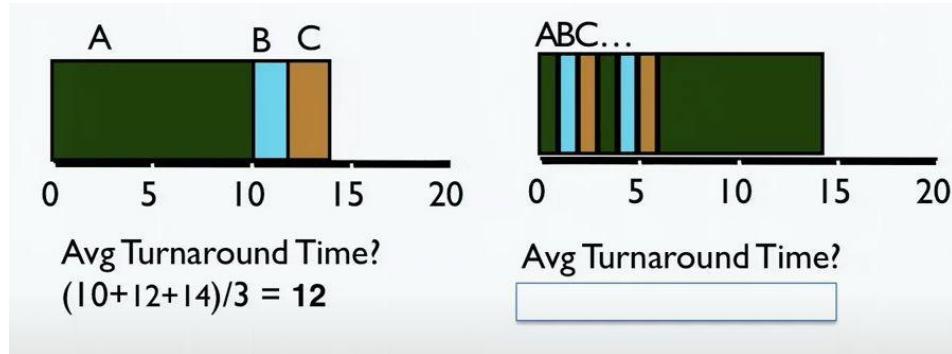
Idea: alternate ready processes for a fixed-length time (called slice or quantum)

Preemptive

Short jobs will finish after fewer time slices

Short jobs will finish sooner than long ones

# Different Job Lengths and Turnaround Time

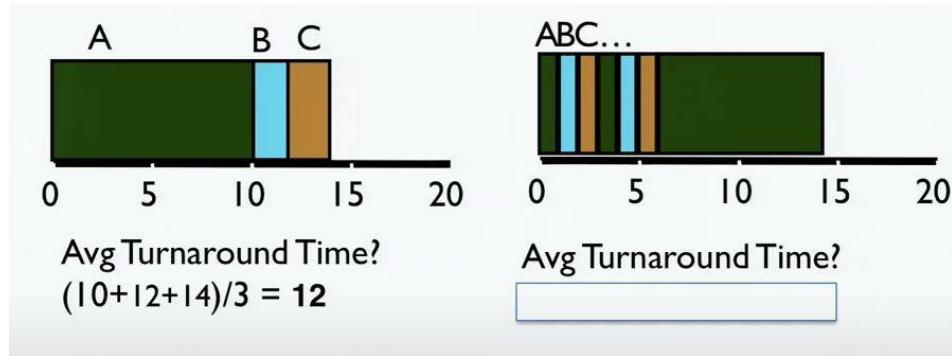


All arrive at t=0 but A arrives a little earlier (so gets scheduled 1st in FIFO)  
Turnaround time in RR: ?

Turnaround time in SJF: ?

Say RR uses 1ms timeslice

# Different Job Lengths and Turnaround Time



All arrive at t=0 but A arrives a little earlier (so gets scheduled 1st in FIFO)

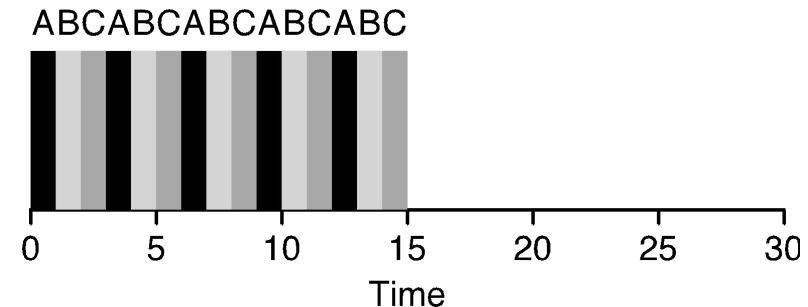
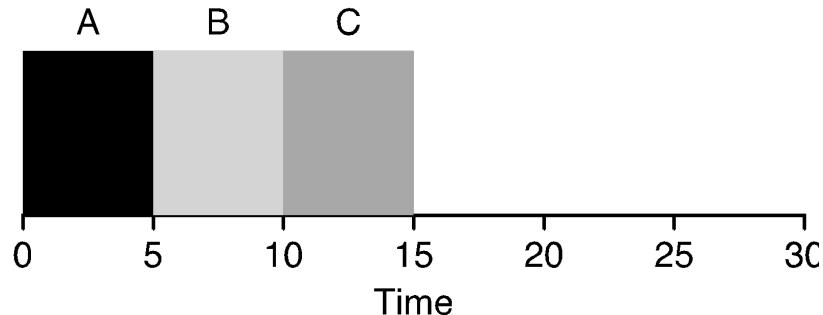
Turnaround time in RR: B=>5, C=>6, A=>6+8=14;  $(5+6+14)/3 = 8.333$

Turnaround time in SJF:  $(2+4+14)/3 = 6.667$

For variable length jobs, RR gets close to SJF in turnaround time and is better than FIFO

When you don't know job lengths, RR does reasonably well when job lengths are different

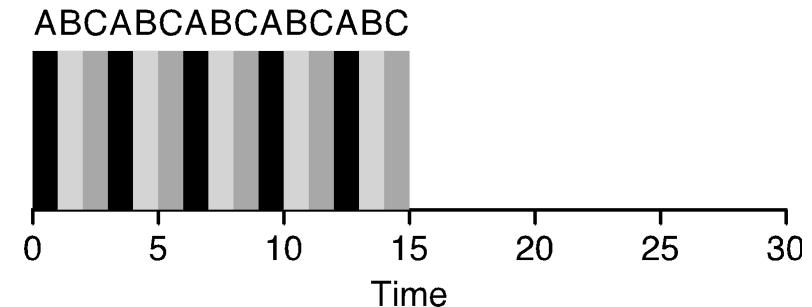
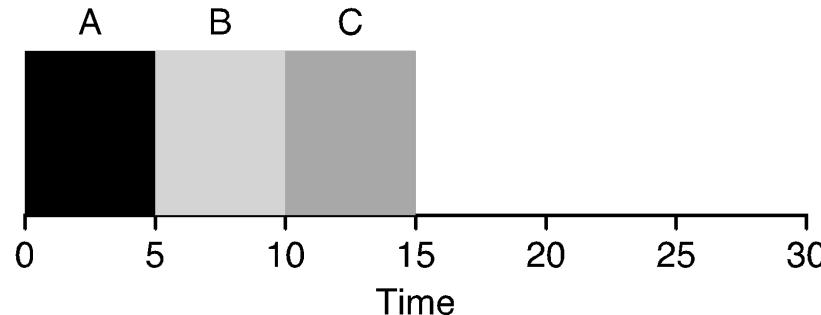
# ROUND ROBIN SCHEDULER



When is RR bad in terms of turnaround time?

When jobs are equal length...

# ROUND ROBIN



Average Turnaround Time

$$(5 + 10 + 15)/3 = 10\text{s}$$

$$(13 + 14 + 15)/3 = 14\text{s}$$

# Batch vs Interactive

Turnaround time is a good metric for CPU-bound programs

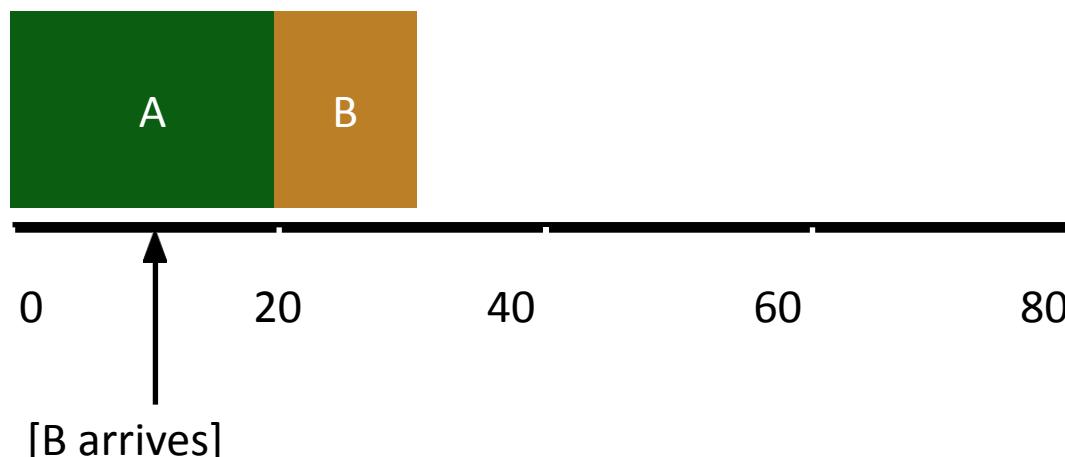
It is not a good metric for interactive programs

# METRIC 2: RESPONSE TIME

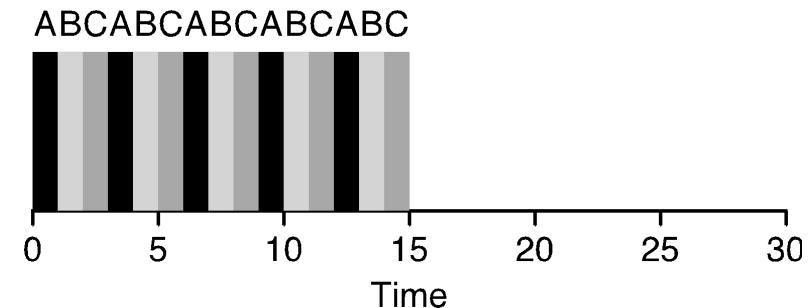
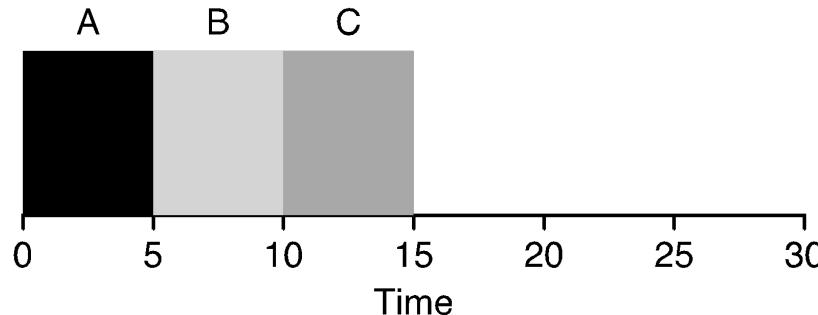
Response time = *first\_run\_time - arrival\_time*

B's turnaround: 20s

B's response: 10s



# ROUND ROBIN



Average Response Time

$$(0 + 5 + 10)/3 = 5\text{s}$$

$$(0 + 1 + 2)/3 = 1\text{s}$$

Average Turnaround Time

$$(5 + 10 + 15)/3 = 10\text{s}$$

$$(13 + 14 + 15)/3 = 14\text{s}$$

# TRADE-OFFS

Round robin increases turnaround time, decreases response time  
Trade-off: **fairness vs turnaround time**

Tuning challenges:

- What is a good time slice?
- What is the overhead in context switching? (hint: not just saving and restoring state)

Round robin: doesn't have to know the job length

Short jobs will complete before long jobs (like SJF)

# ASSUMPTIONS

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. ~~Once started, each job runs to completion~~
4. ~~All jobs only use the CPU (no I/O)~~
5. ~~Run time of each job is known (perfect knowledge)~~

# MULTI-LEVEL FEEDBACK QUEUE

# MLFQ: GENERAL PURPOSE SCHEDULER

Must support two job types with distinct goals

- “interactive” programs care about response time
- “batch” programs care about turnaround time

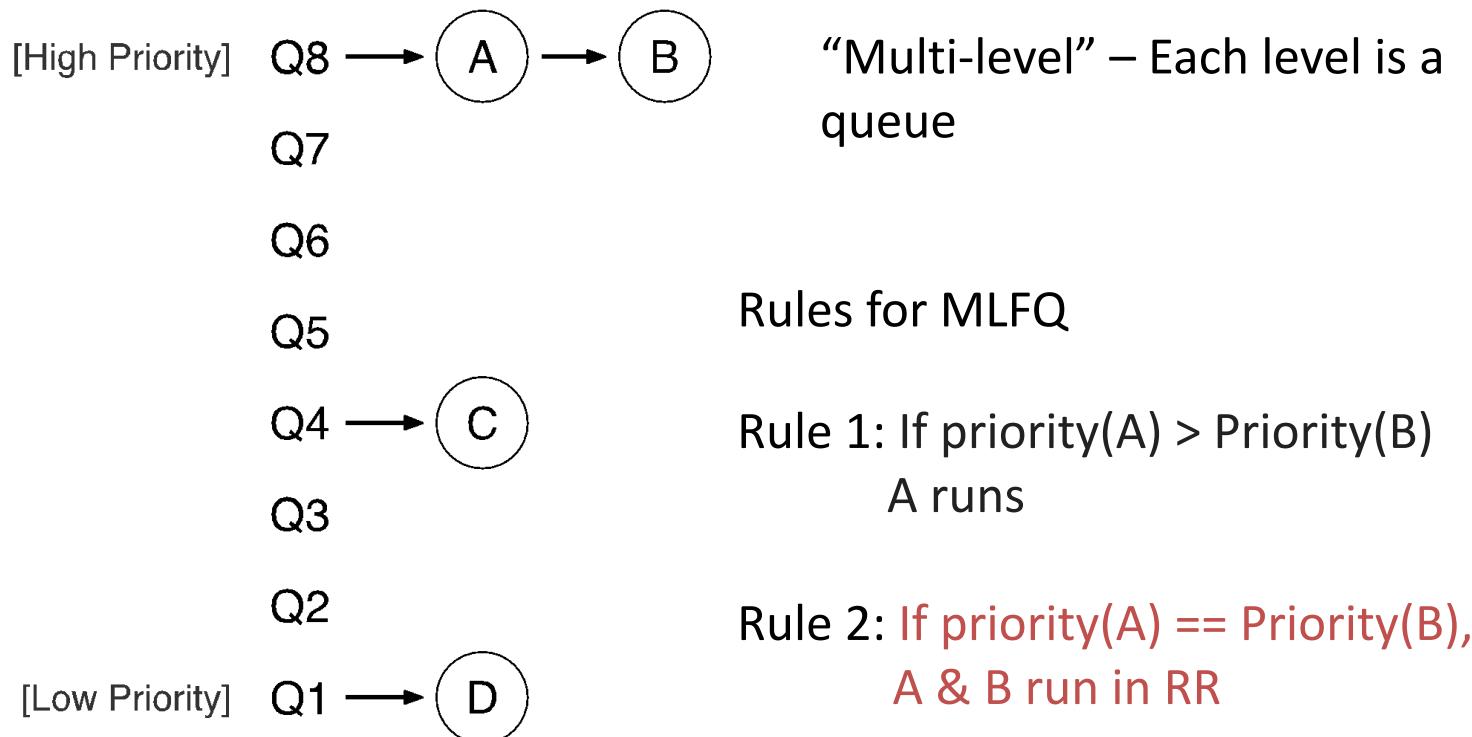
Approach:

Multiple levels of round-robin

Higher levels have higher priority

Processes are preemptable

# MLFQ EXAMPLE



# CHALLENGES

How to set priority?

Does a process stay on one queue or move across queues?

Approach:

Use recent past behavior of process to predict future

Common approach in systems when don't have perfect knowledge

If interactive recently, will likely be interactive in the near future

# MORE MLFQ RULES

Rule 1: If  $\text{priority}(A) > \text{Priority}(B)$ , A runs

Rule 2: If  $\text{priority}(A) == \text{Priority}(B)$ , A & B run in RR

Rule 3: Processes start at top priority

Rule 4: If job uses whole slice, demote process  
(longer time slices at lower priorities)

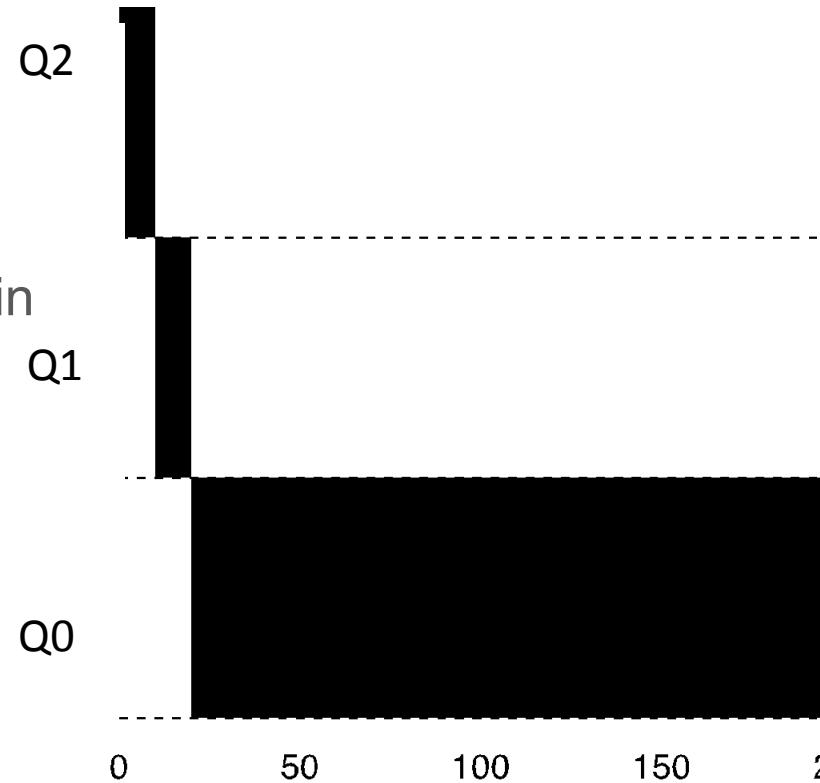
# ONE LONG JOB

Starts at top

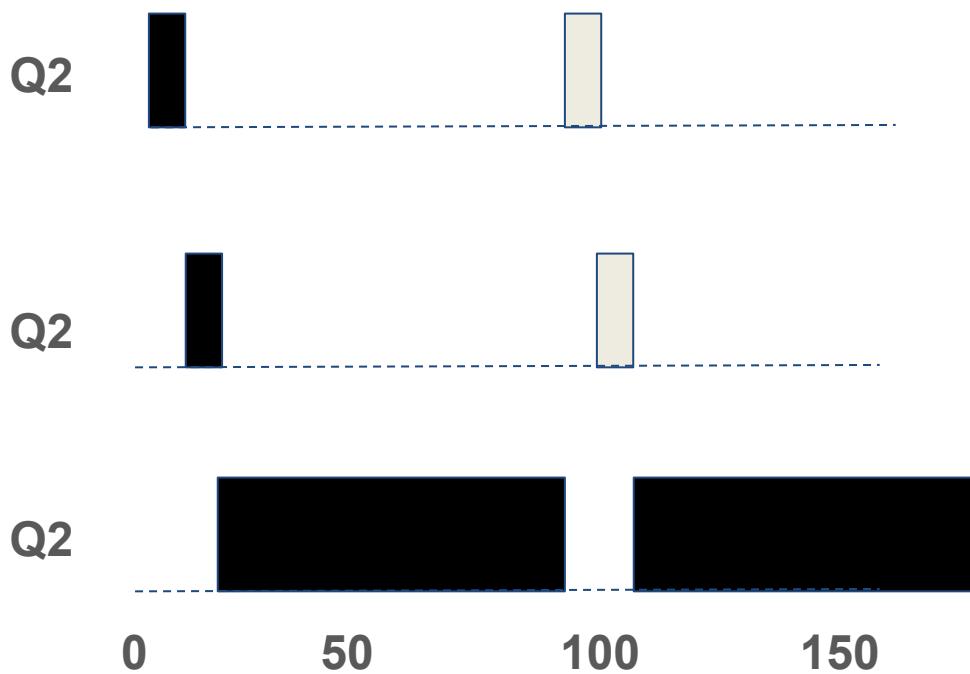
Uses whole slice

Moves down

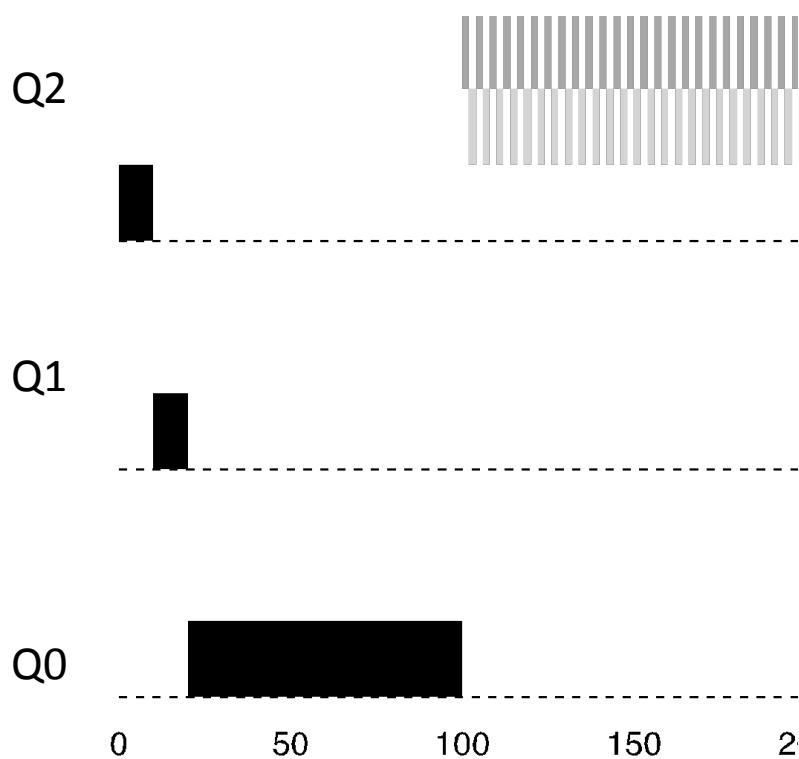
Moves down again



# INTERACTIVE SHORT PROCESS JOINS

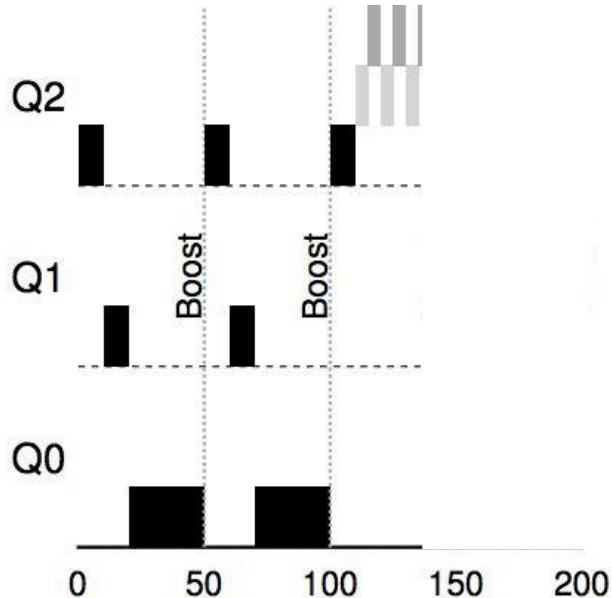
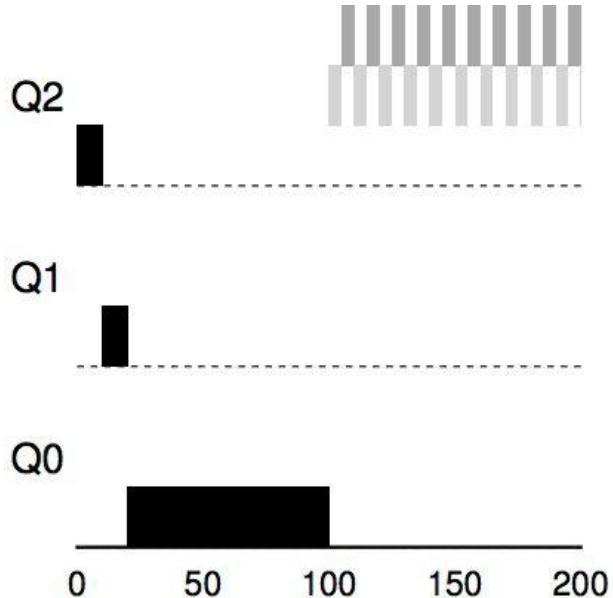


# MLFQ PROBLEMS?



What is the problem  
with this schedule ?

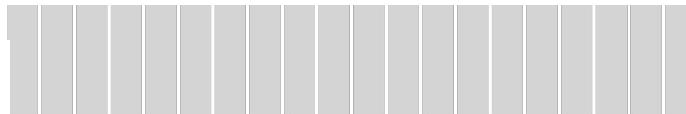
# AVOIDING STARVATION



Rule 5: After some time period  $S$ , move all the jobs in the system to the topmost queue.

# GAMING THE SCHEDULER ?

Q2

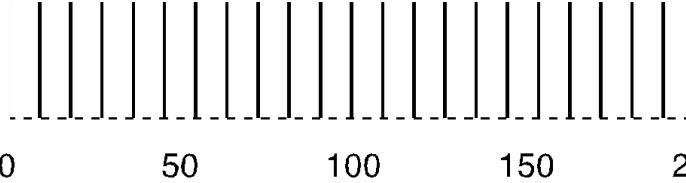


Job could trick scheduler by doing I/O just before time-slice end

Q1

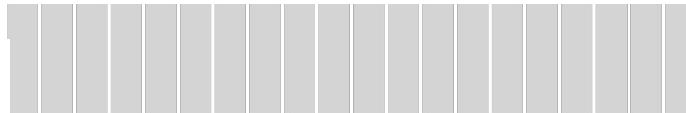
When high-pri job is blocked, low pri job gets to run but when ready, it preempts the low-pri job

Q0



# GAMING THE SCHEDULER ?

Q2



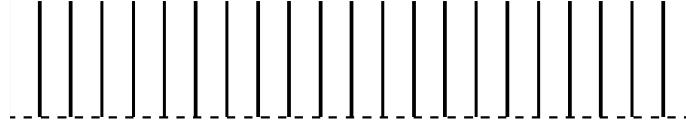
Job could trick scheduler by doing I/O just before time-slice end

Q1



Rule 4\*: Account for total run time at priority level  
Downgrade when exceed threshold

Q0



0 50 100 150 200

# MLFQ Summary

Works well with a mix of compute-intensive and interactive jobs  
Can finish shorter jobs earlier

5 rules to implement MLFQ

Versions of it are used in BSD, Sun Solaris, Windows NT, and Linux.

But multi-core versions!

# FAIRNESS: Quick Detour

How to define? 4 kids share a cake, each gets 25%  
If one kid can eat only 10%, is it fair to force her to eat 25%?

**Min-max Fairness:** Least demanding gets fair share first  
Kid1 takes 10%, And her 15% is divvied up: others kids  
entitled to 30% fair share

Topic useful for any practical system design:  
Queueing Theory

# SUMMARY

## Scheduling Policies

Understand **workload characteristics** like arrival, CPU, I/O

Scope out goals, **metrics** (turnaround time, response time)

Metrics: Latency vs throughput

Real-life schedulers are complicated beasts

Lots of trade-offs based

Past behavior is good predictor of future behavior