

CPU SCHEDULING

Andrea Arpaci-Dusseau
CS 537, Spring 2019

ANNOUNCEMENTS

- Project 1 Due this Monday midnight
 - Piazza: TAs and Peer Mentors will NOT answer questions after noon on Monday; No lab hours after 10pm
- Project 2 available: Due Monday Sept 23rd
 - Two videos are available
 - Start BEFORE discussion section
- Simple Homework in Canvas available: Process
 - Practice Exam Questions; Gives solutions; Can retake
 - Due 1 week
- Midterm 1: Oct 10 (Thu) instead of Oct 9 (Wed, Yom Kippur)

CPU SCHEDULING: LEARNING OUTCOMES

- How does the OS decide which process to run?
- What are some metrics to optimize?
- What are different scheduling policies, such as:
FCFS, SJF, STCF, RR and MLFQ?
- How to handle mix of interactive and batch processes?
- What to do when OS doesn't have complete information?

RECAP

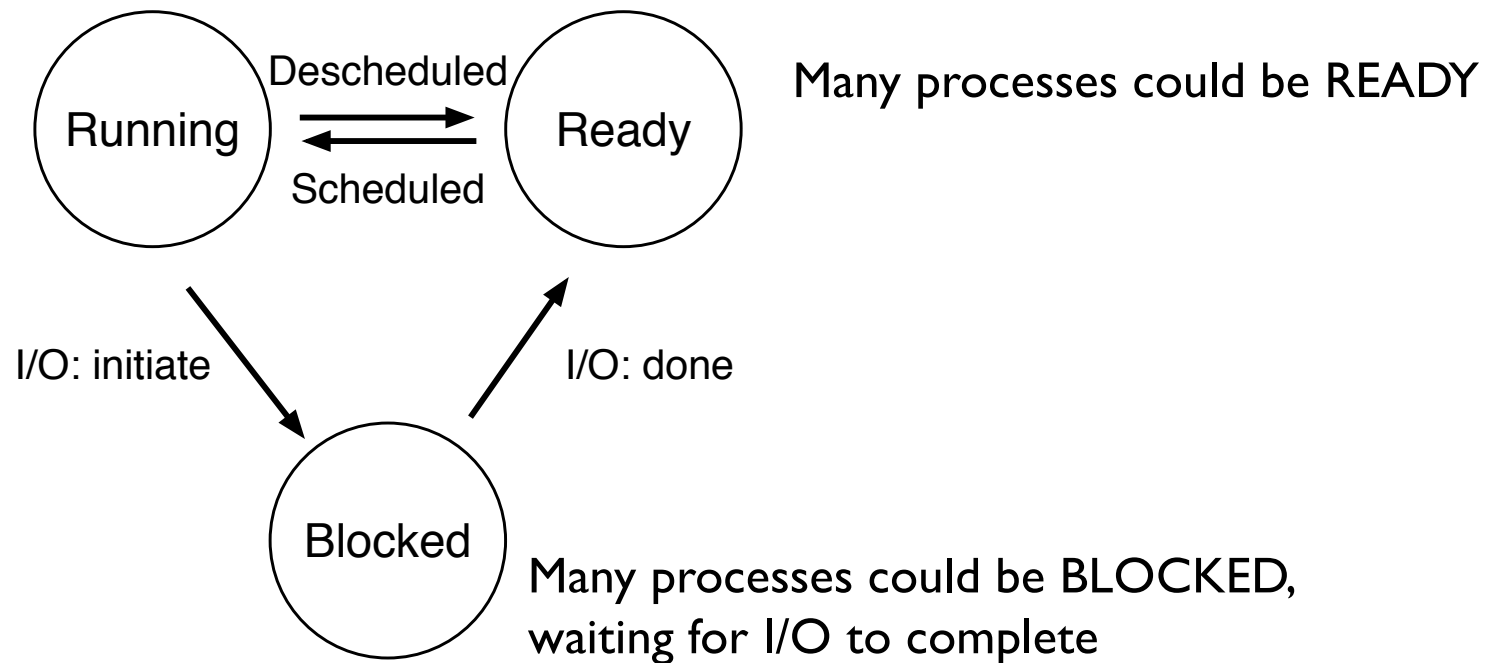
RECAP: SCHEDULING MECHANISM

Process: Abstraction to virtualize CPU

Use time-sharing in OS to switch between processes

PROCESS STATE TRANSITIONS

At most 1 process RUNNING



How to transition? ("mechanism")

When to transition? ("policy")

How many processes can be in each state simultaneously?

RECAP: SCHEDULING MECHANISM

Limited Direct Execution

Use system calls to run access devices from user mode

Use timer interrupts to context switch for multi-tasking

SCHEDULING TERMINOLOGY

Workload: set of **jobs** (arrival time, run_time)

Job: Current scheduling burst of a process
Alternates between CPU and I/O
Moves between ready and blocked queues

Scheduler: 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
- $\text{Completion_time} - \text{arrival_time}$

Minimize response time

- Can't control how long job needs to run; minimize time before scheduled
- $\text{Initial_schedule_time} - \text{arrival_time}$

Maximize throughput (jobs completed / second)

- Want many jobs to complete per unit of time

Maximize resource utilization (% time CPU busy)

- Keep expensive devices busy

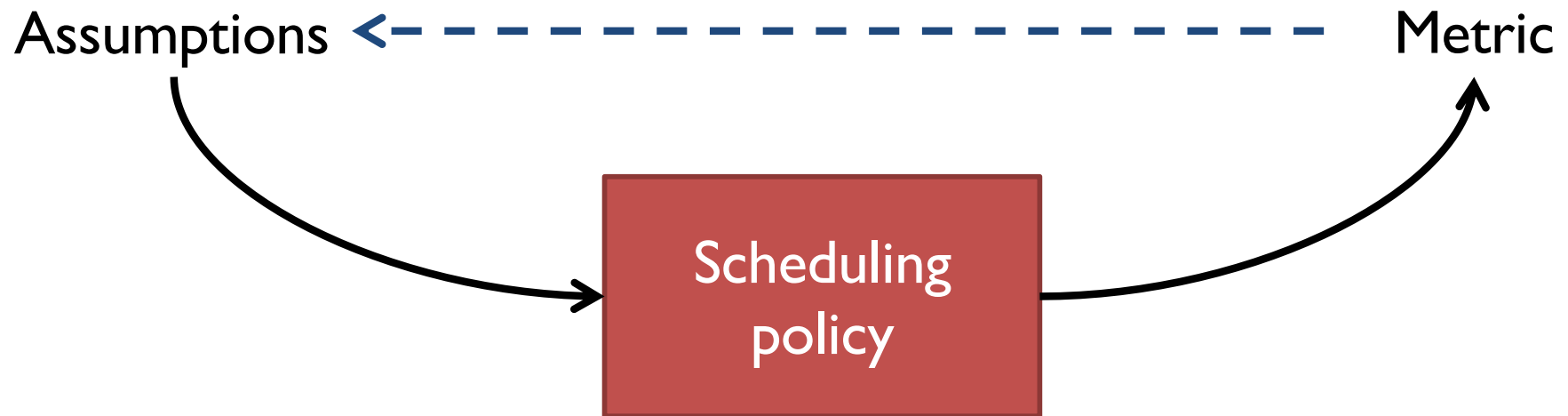
Minimize overhead (# of context switches and cache misses)

- Reduce number of context switches

Maximize fairness (variation of CPU time across jobs)

- All jobs get same amount of CPU over some time interval

LECTURE FORMAT



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
(Oracle, perfect knowledge)

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 =

B =

Average =

FIFO / FCFS



FIFO / FCFS

FIFO: First In, First Out

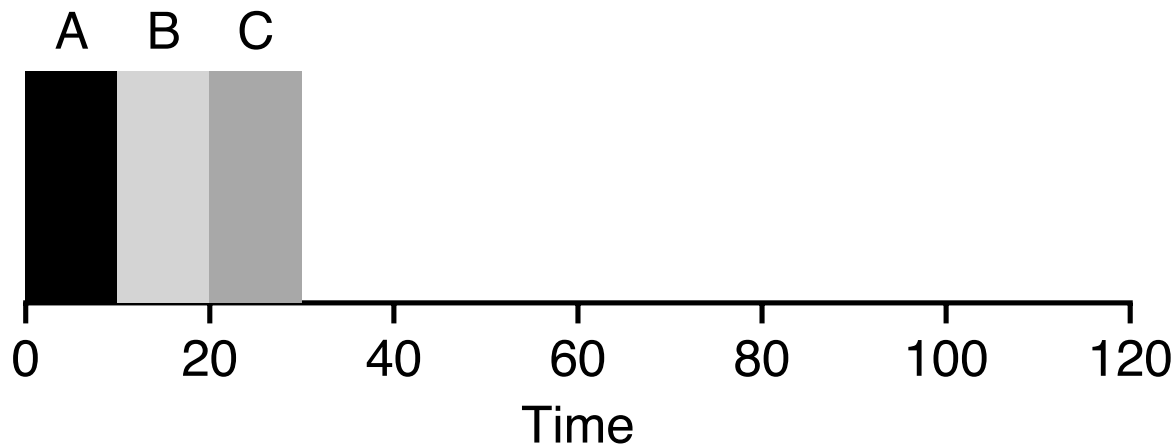
FCFS: First Come, First Served

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

Run jobs in *arrival_time* order (ties go to first job in list)

FIFO / FCFS

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



Gantt chart: Illustrate how jobs are scheduled over time

Average
Turnaround
Time ?

2-MINUTE NEIGHBOR CHAT

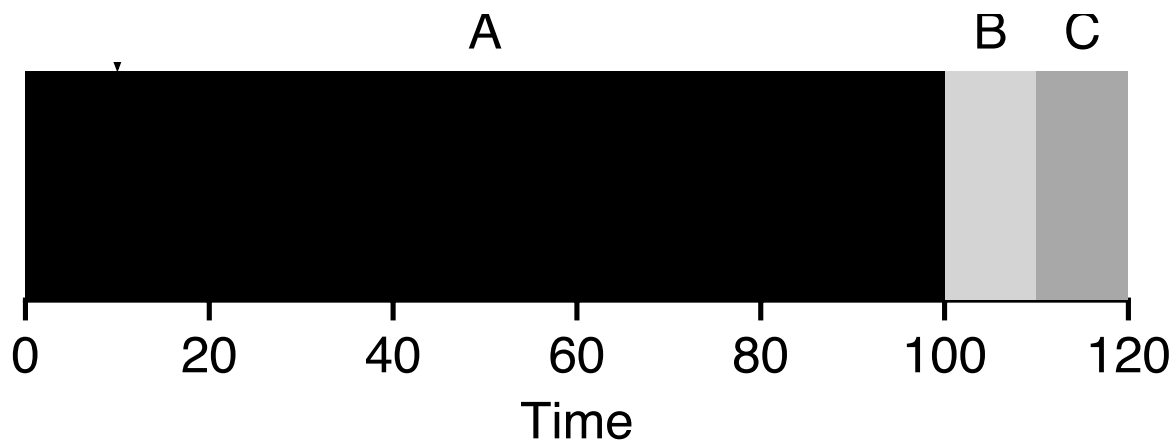
- ~~1. Each job runs for the same amount of time~~
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

How will FIFO perform without this assumption ?

What scenarios can lead to bad performance?

LONG-RUNNING FIRST JOB

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



Average
Turnaround
Time?

SCHEDULING PROBLEM: CONVOY EFFECT



CHALLENGE

Turnaround time suffers when short jobs must wait for long jobs

New scheduler:

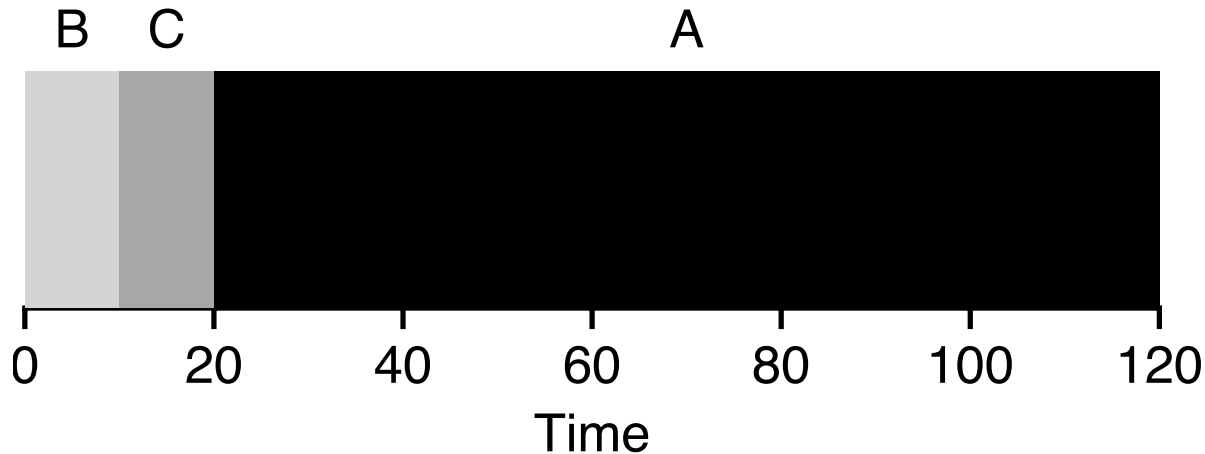
SJF (Shortest Job First)

Choose job with smallest run_time!

(Assume OS has perfect information...)

SHORTEST JOB FIRST (SJF)

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



Average
Turnaround
Time?

FIFO: 110s ?!

SJF THEORY

- SJF is provably optimal for minimizing average turnaround time (assuming no preemption)
- Intuition:
Moving shorter job before longer job **improves** turnaround time of short job more than it **harms** turnaround time of long

ASSUMPTIONS

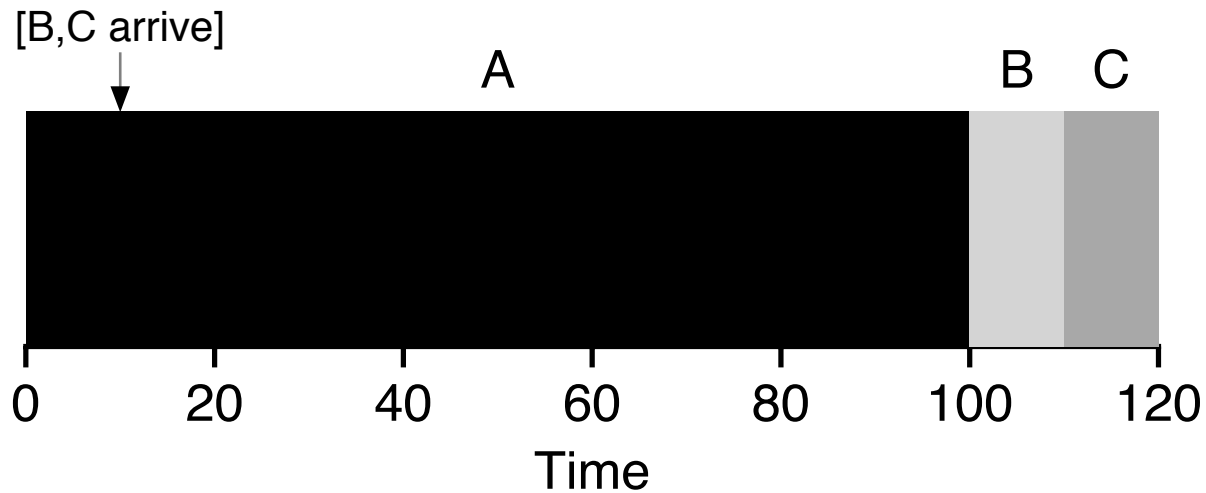
- ~~1. Each job runs for the same amount of time~~
- ~~2. All jobs arrive at the same time~~
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

2-MINUTE NEIGHBOR CHAT

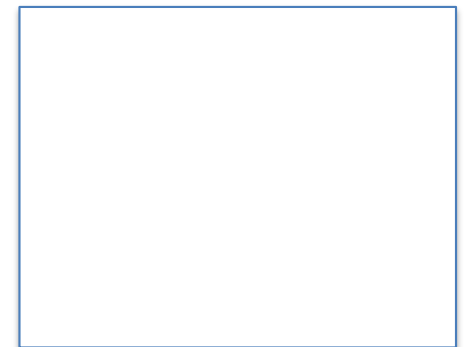
Job	Arrival(s)	run time (s)
A	0	100
B	10	10
C	~10	10

Gantt Chart and Average Turnaround Time with SJF?

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



Average
Turnaround
Time ?



PREEMPTIVE SCHEDULING

Previous schedulers:

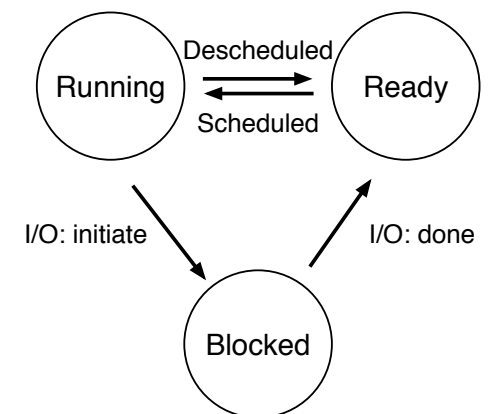
FIFO and SJF are non-preemptive (never deschedule a running process)

Only schedule new job when previous job voluntarily relinquishes CPU
(e.g., performs I/O or exits)

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

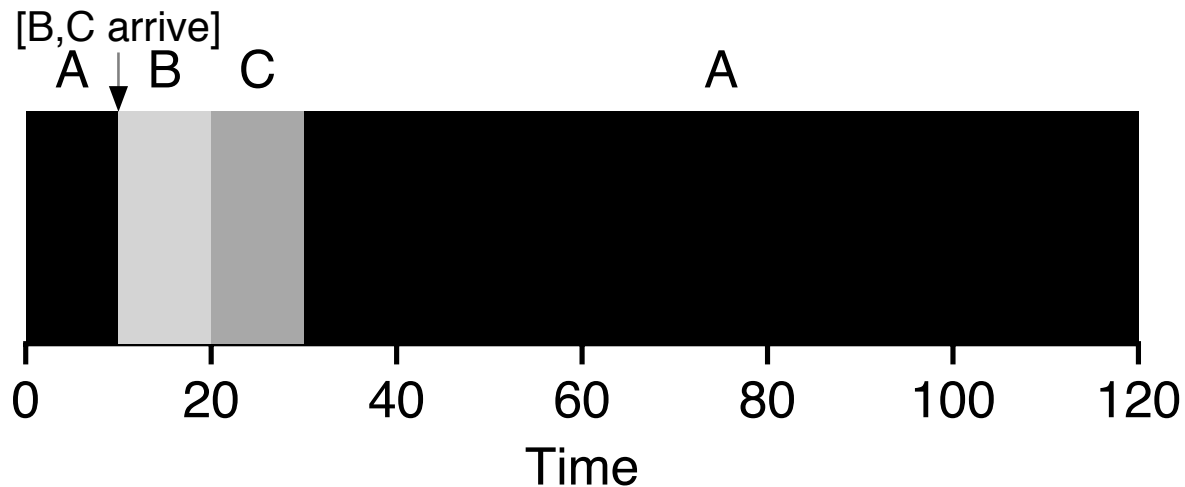
STCF (Shortest Time-to-Completion First)

Always run job that will complete the quickest



PREMPTIVE STCF (OR SCTF)

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

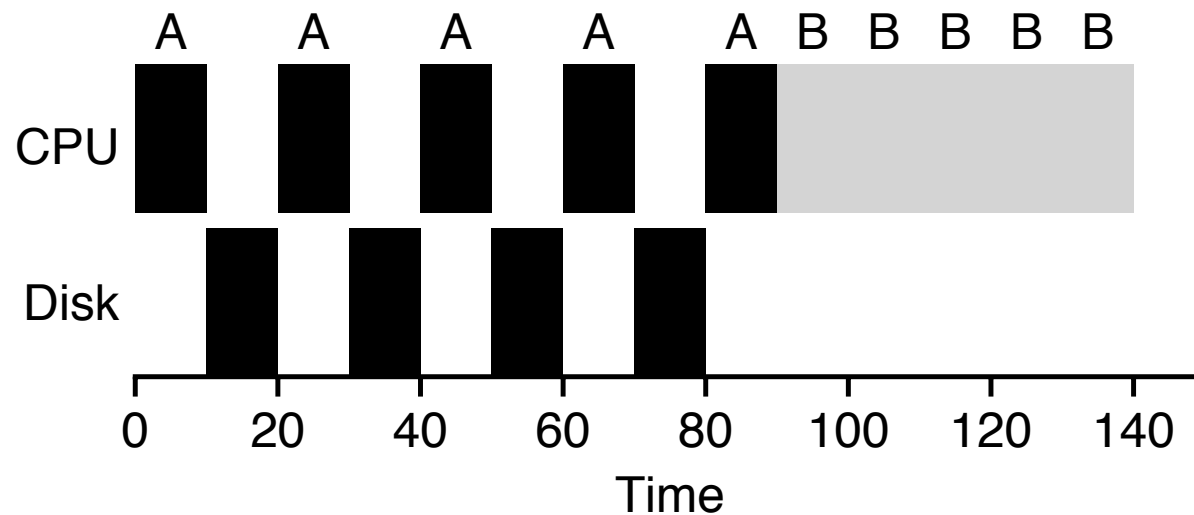


Average
Turnaround
Time

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

NOT IO AWARE

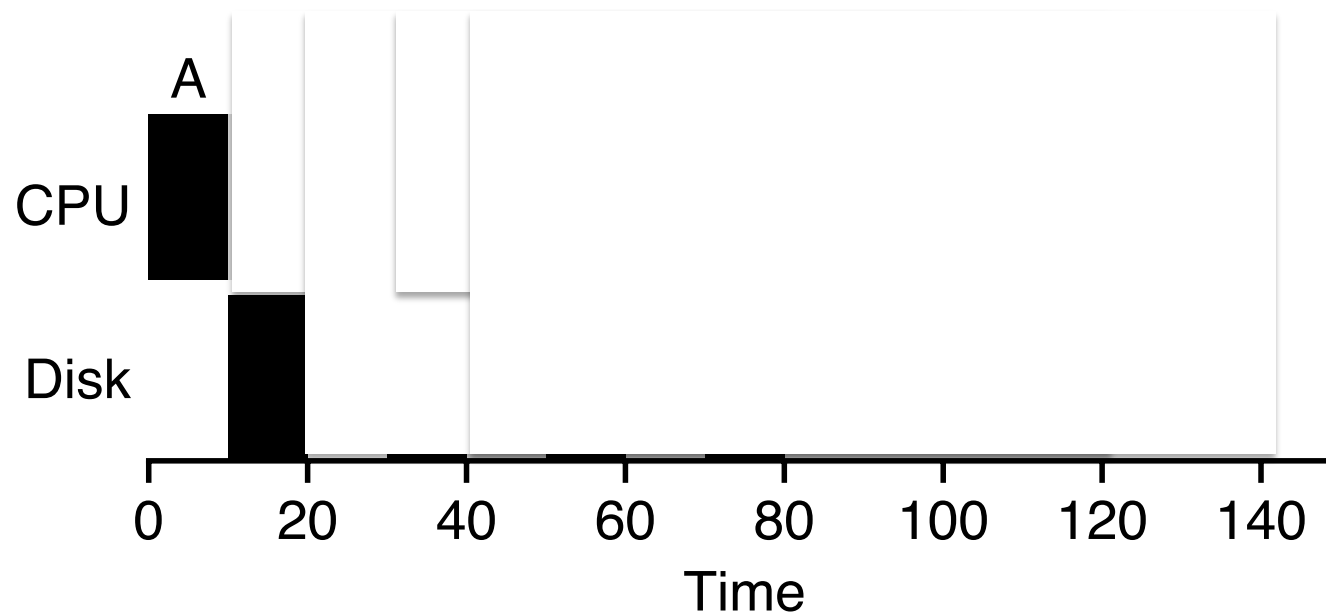


Job holds on to CPU while blocked on disk!

Instead, treat Job A as multiple separate CPU bursts

I/O AWARE SCHEDULING

B is a long CPU-bound job



When Job A completes I/O, another Job A is ready

Each CPU burst of A is shorter than Job B; With SCTF, Job A preempts Job B

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

WHAT IF DO NOT KNOW JOB RUNTIME?

- For metric of average turnaround:
- If jobs have same length
 - FIFO is fine
- If jobs have much different lengths
 - SJF is much better
- How can OS get short jobs to complete first if OS doesn't know which are short?

ROUND-ROBIN SCHEDULER

New scheduler: RR (Round Robin)

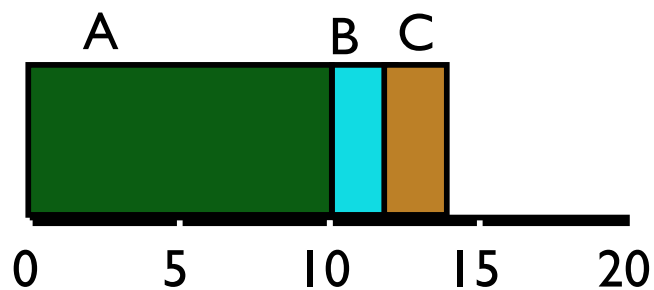
Alternate ready processes for a fixed-length time-slice

Preemptive

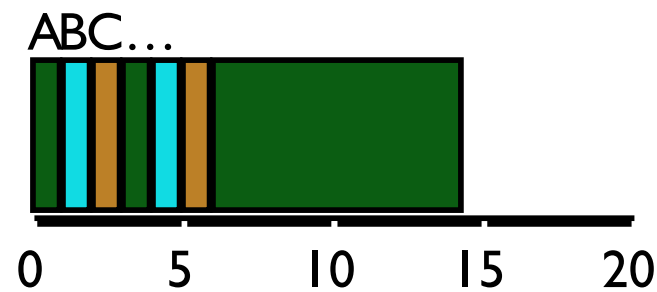
Short jobs will finish after fewer time-slices

Short jobs will finish sooner than long jobs

FIFO VS RR: JOBS DIFFERENT LENGTHS



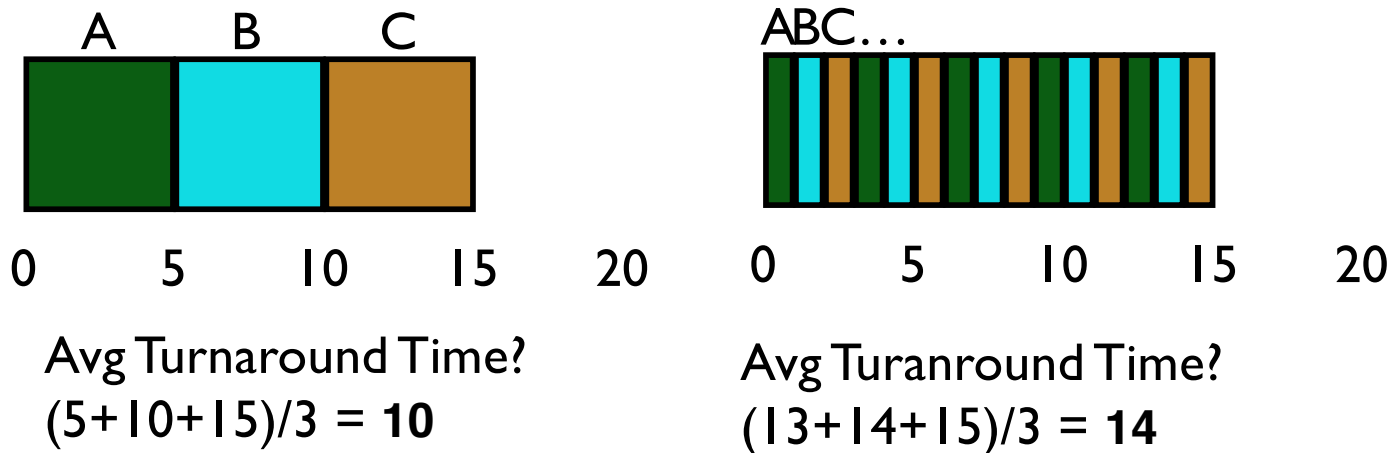
Avg Turnaround Time?
 $(10+12+14)/3 = 12$



Avg Turnaround Time?

If don't know run-time of each job, RR gives short jobs a chance to run and finish fast

FIFO VS RR: JOBS SAME LENGTHS



When is RR worse than FIFO?

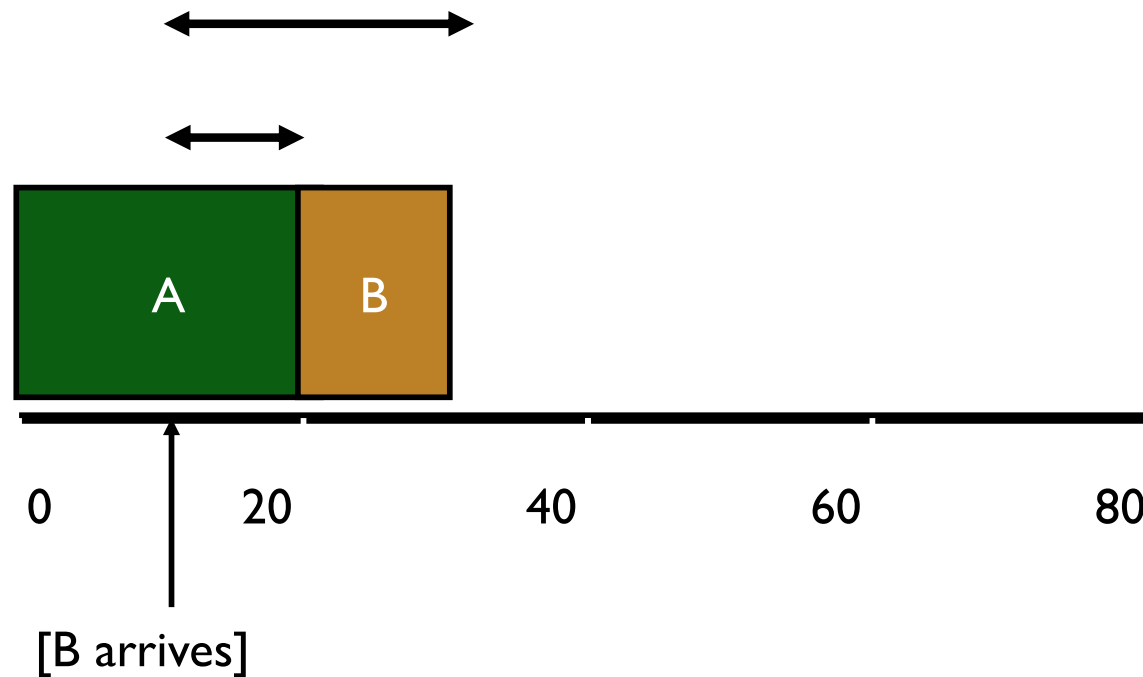
Average turn-around time with equal job lengths

METRIC 2: RESPONSE TIME

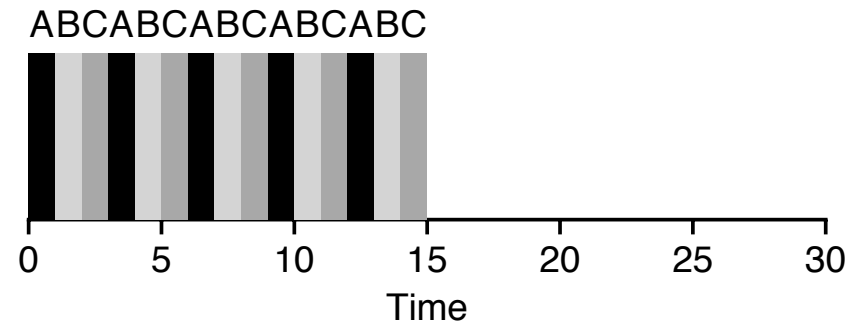
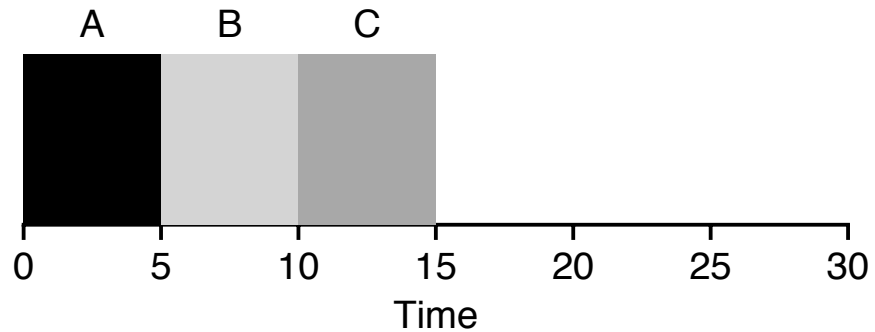
Response time = *first_run_time* - *arrival_time*

B's turnaround: 20s

B's response: 10s



ROUND ROBIN SCHEDULER



Average Response Time

TRADE-OFFS

Round robin:

MAY increase turnaround time, decreases response time

Tuning challenges:

What is a good time slice for round robin?

What is the overhead of context switching?

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

MULTI-LEVEL FEEDBACK QUEUE

MLFQ: GENERAL PURPOSE SCHEDULER

Used in practice

Must support two job types with distinct goals

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

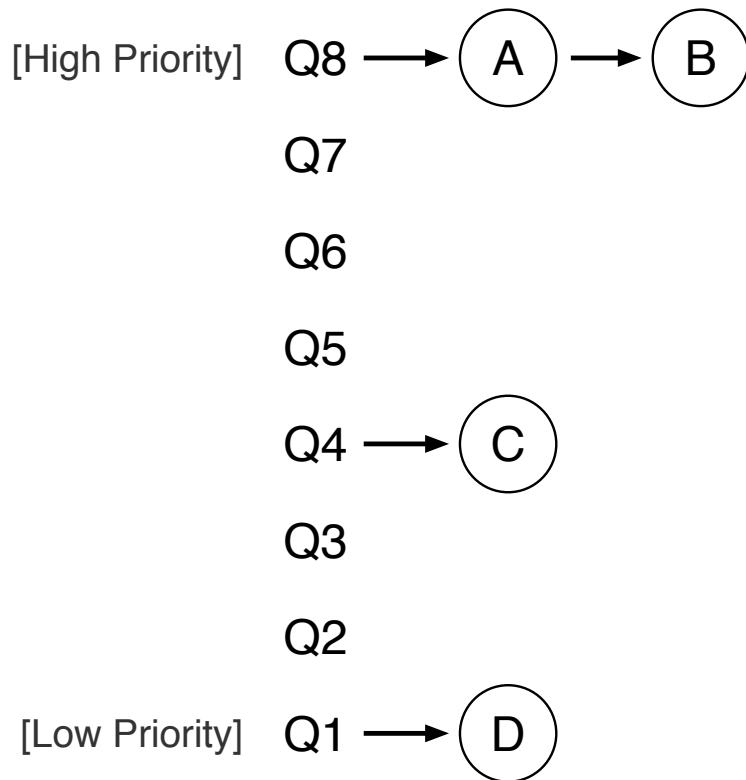
Approach:

Multiple levels of round-robin

Each level has higher priority than lower level

Can preempt them

MULTI-LEVEL PRIORITIES



“Multi-level” – Each level is a queue!

Rules for MLFQ

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

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

How to to set priority?

Approach 1: Static (no changes): nice command

Approach 2: Dynamic: Use history for feedback

FEEDBACK: HISTORY

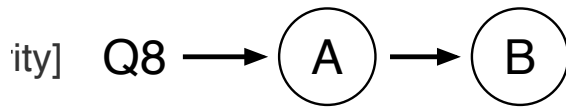
Approach:

Use past behavior of process to predict future!

Common approach in OS when don't have perfect knowledge

Guess how CPU burst (job) will behave based on past CPU bursts

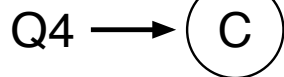
MORE MLFQ RULES



Q7

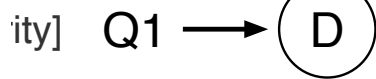
Q6

Q5



Q3

Q2



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

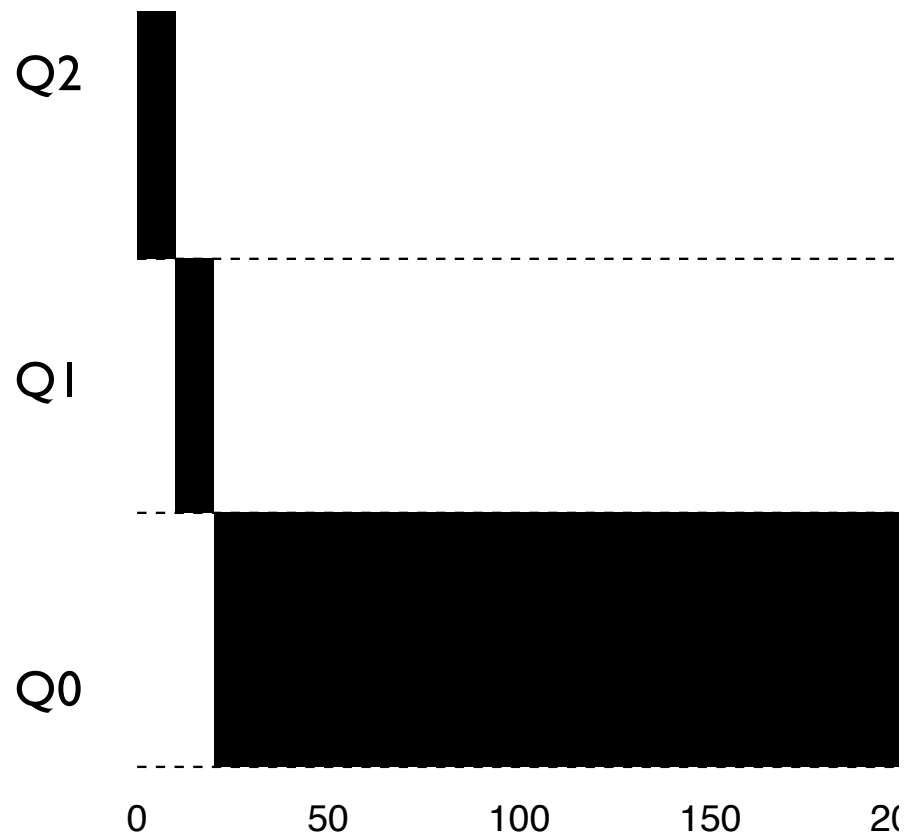
Rule 2: If $\text{priority}(A) == \text{Priority}(B)$, A & B run in RR

Rule 3: Processes start at top priority

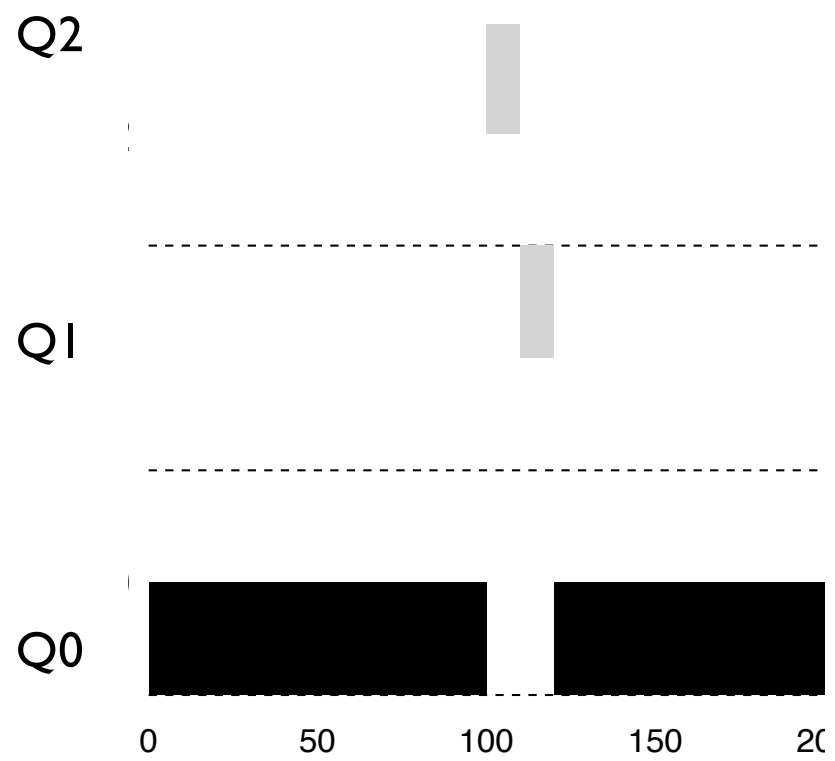
Rule 4: If job uses whole slice, demote process

(longer time slices at lower priorities)

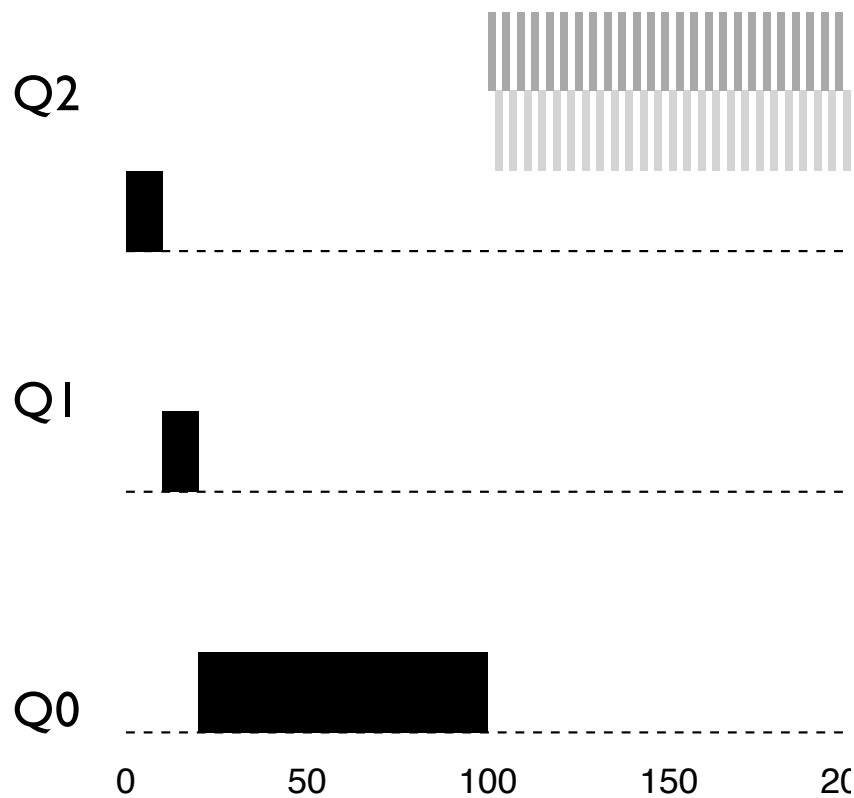
EXAMPLE: ONE LONG JOB



INTERACTIVE PROCESS JOINS



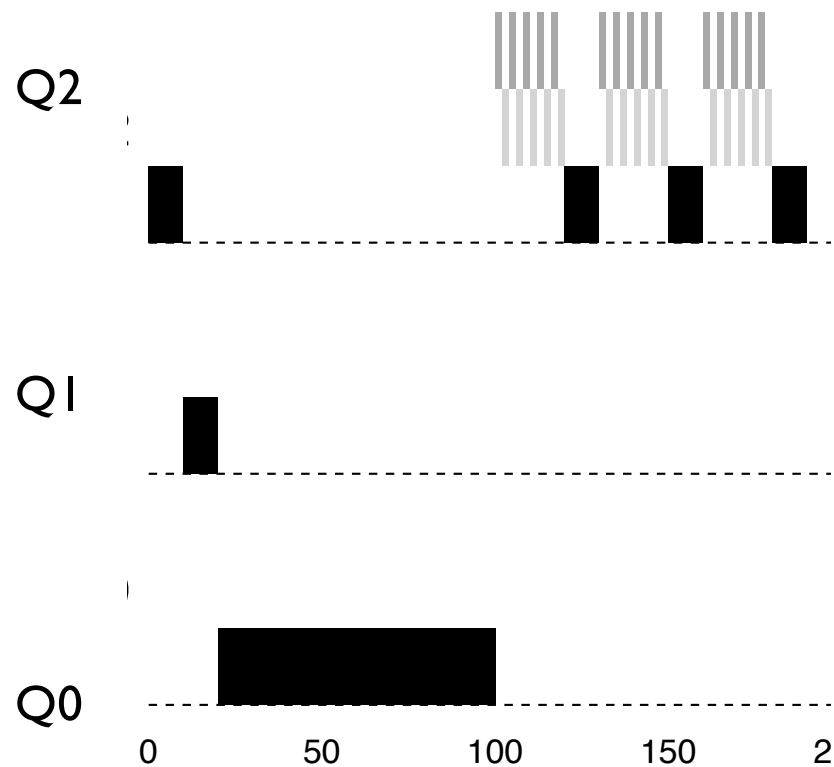
MLFQ PROBLEMS?



Two (or more) short jobs arrive
Each uses CPU for short burst
Each stays at high priority

What is the problem
with this schedule ?

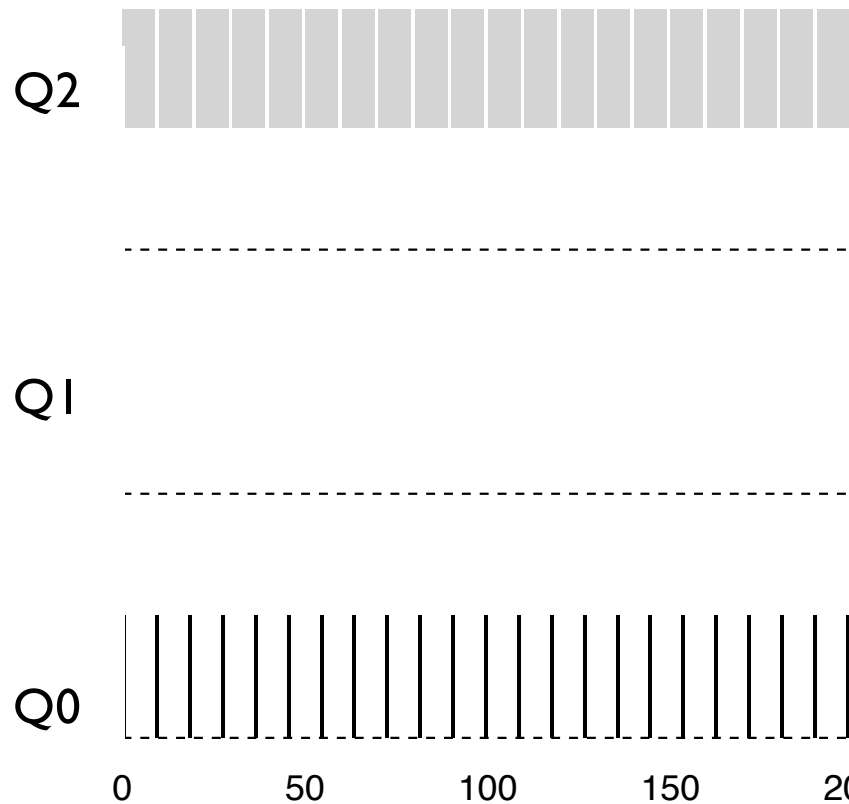
AVOIDING STARVATION



Periodically **boost** priority of all jobs (or all jobs that haven't been scheduled)

What is wrong with this diagram?

GAMING THE SCHEDULER ?



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

Account for **total run time** at priority
Downgrade when exceed threshold

OTHER SCHEDULERS: LOTTERY SCHEDULING

Other metrics and schedulers appropriate for other environments

Purchasing fixed amount of cloud resources

Goal: proportional (fair) share

Approach:

- give processes lottery tickets
- whoever wins runs
- higher priority => more tickets

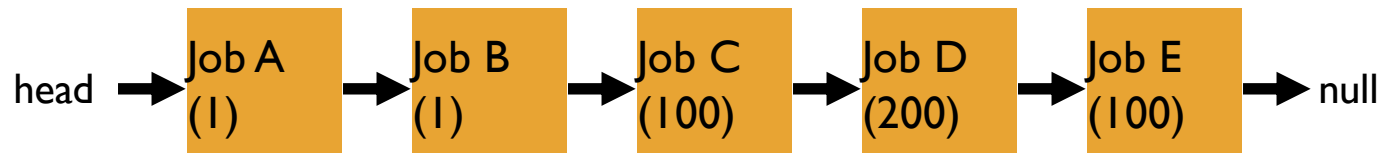
Amazingly simple to implement

LOTTERY EXAMPLE

```
int counter = 0;
int winner = getRandom(0, totaltickets);
node_t *current = head;
while(current) {
    counter += current->tickets;
    if (counter > winner) break;
    current = current->next;
}
// current gets to run
```

Who runs if winner is:

50
350
0



SUMMARY

- No ideal scheduler for every workload and metric
 - Understand goals (metrics) and workload, design scheduler around that
- General purpose schedulers need to support processes with different goals
- Past behavior is good predictor of future behavior?

ANNOUNCEMENTS

- Project 1 Due this Monday midnight
- Project 2 available: Due Monday Sept 23rd
- Simple Homework in Canvas available: Process
- Midterm 1: Oct 10 (Thu) instead of Oct 9 (Wed, Yom Kippur)