Scheduling: Multi-level feedback queue

Glenn Bruns
CSUMB

Getting realistic about scheduling

We want low turnaround time.

Can we realistically use SJF?

We don't know when jobs will arrive.

More importantly, we don't know how long jobs will run.

How to run short jobs first when we don't know which jobs will be short?

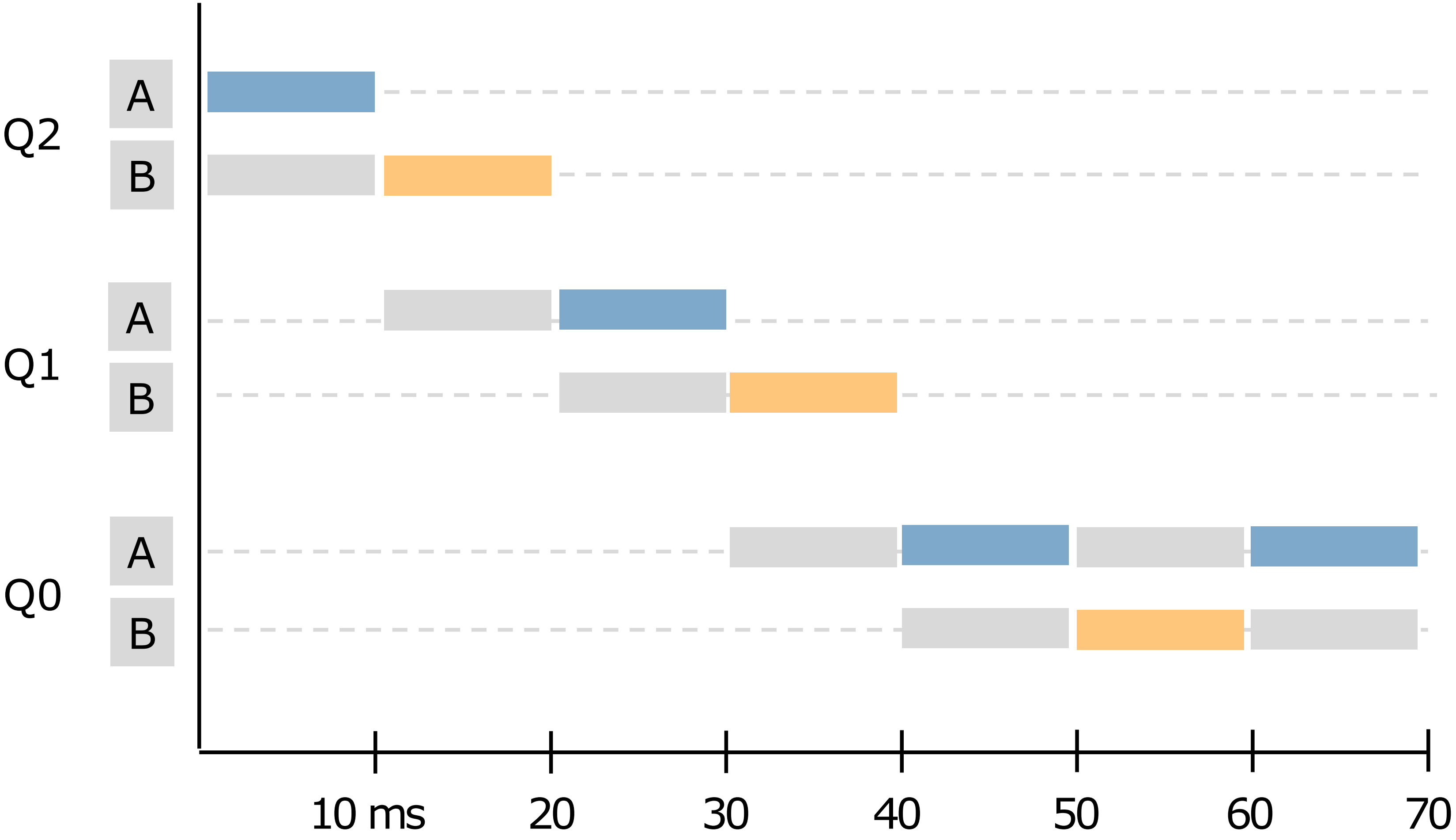
Lecture Objectives

At the end of this lecture, you should be able to:

- □ Define "compute bound" and "I/O bound" jobs
- □ Simulate the behavior of a Multi Level Feedback Queue (MLFQ) scheduler
- □ Give examples showing how MLFQ helps with scheduling problems

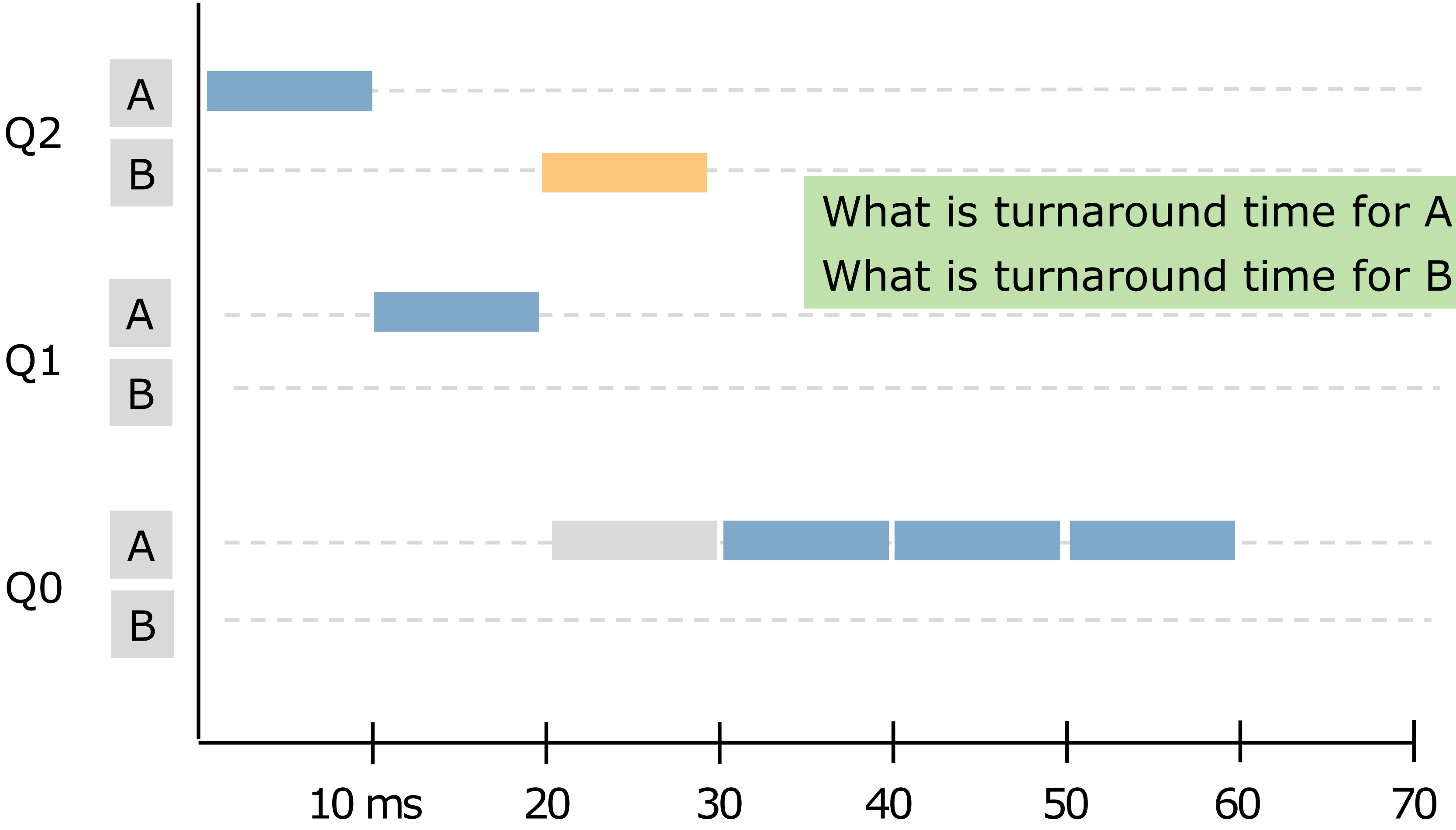
Play the MLFQ Game

3 queues, 2 processes (A,B) of 50 ms, 10 ms per time slice



Suppose one job is short

process A runs for 50 ms; shows up at time 0.
process B runs for 10 ms; shows up at time 20. 10 ms per time slice



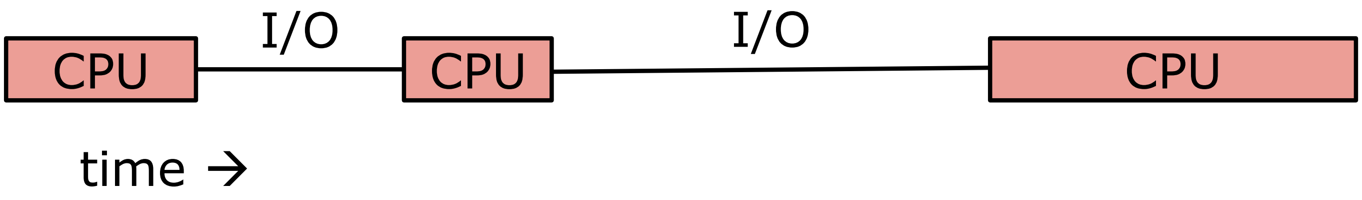
What is turnaround time for A?

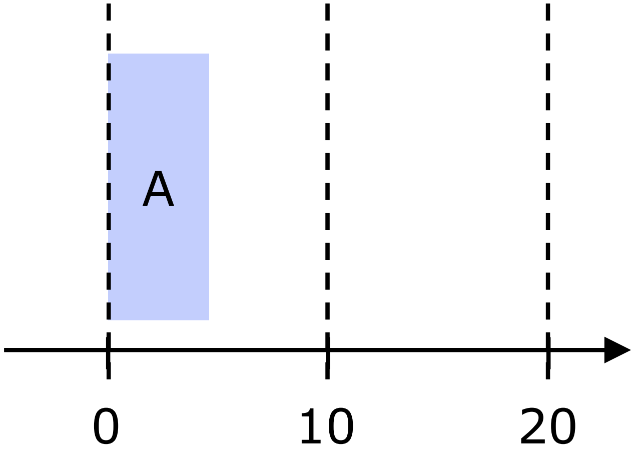
What is turnaround time for B?

Processes do both CPU and I/O

I/O could be mouse or keyboard input, printer output, etc., but especially disk reads and writes

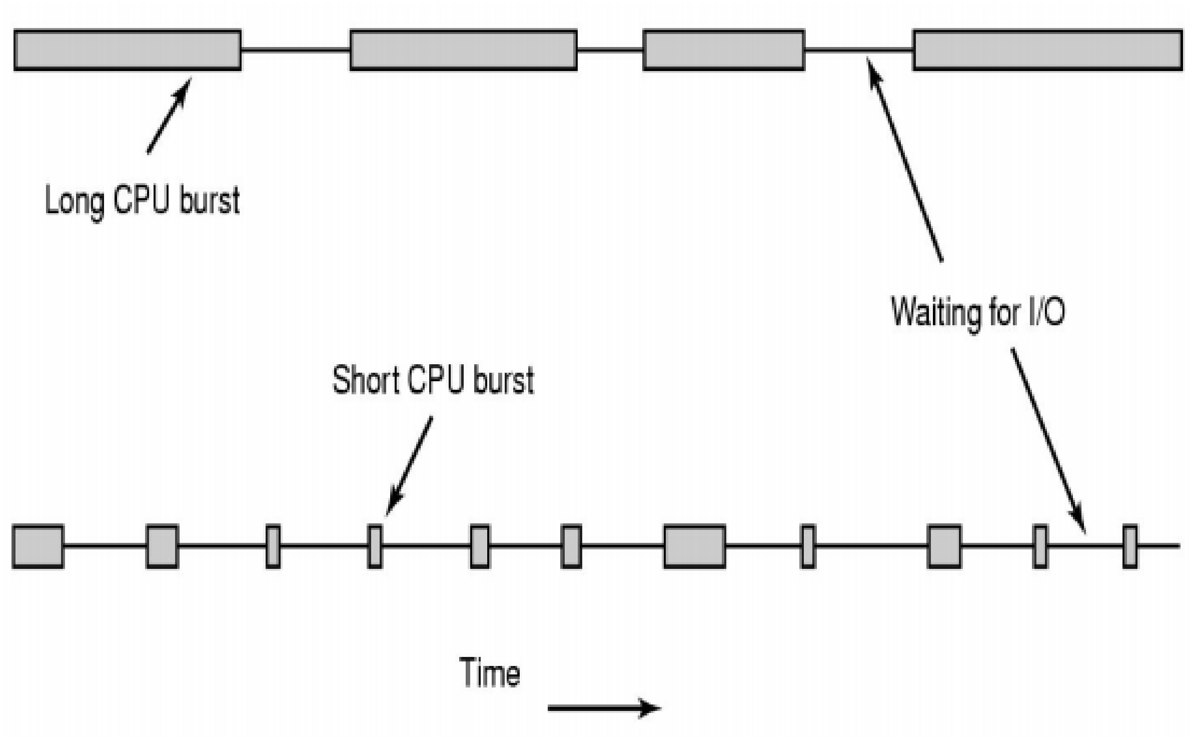
```
x = x * 5;                                         // CPU
char *status = fgets(s, MAX_BUF-1, infile);        // I/O
```





Process A "gave up the CPU" before the end of its time slice, because it needed to do I/O

CPU-bound jobs vs. I/O-bound jobs



CPU bound

I/O bound

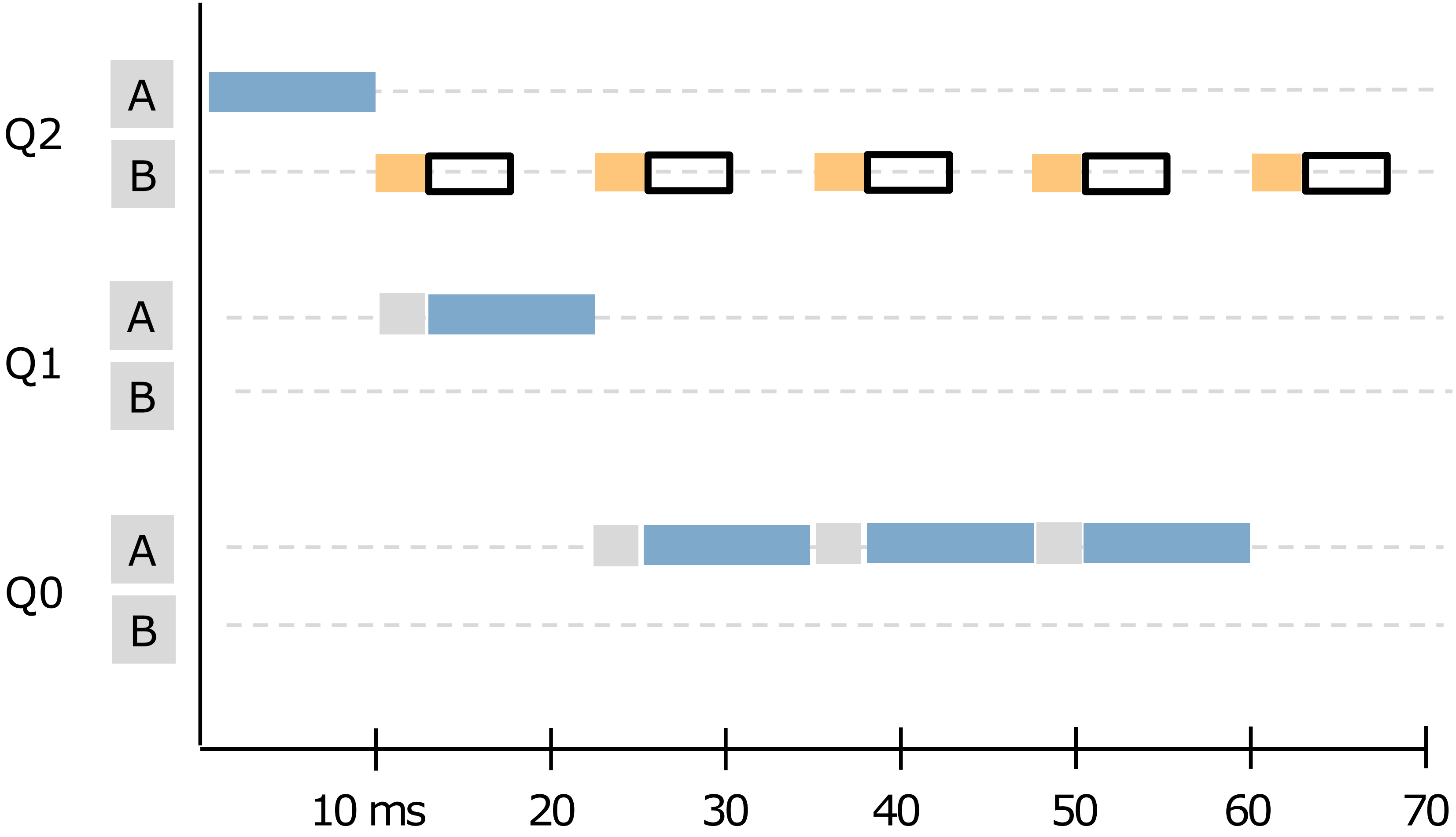
We want to:

- make sure short and interactive jobs get handled quickly
- adapt to jobs as they run

(diagram from Modern Operating Systems, Andrew Tanenbaum)

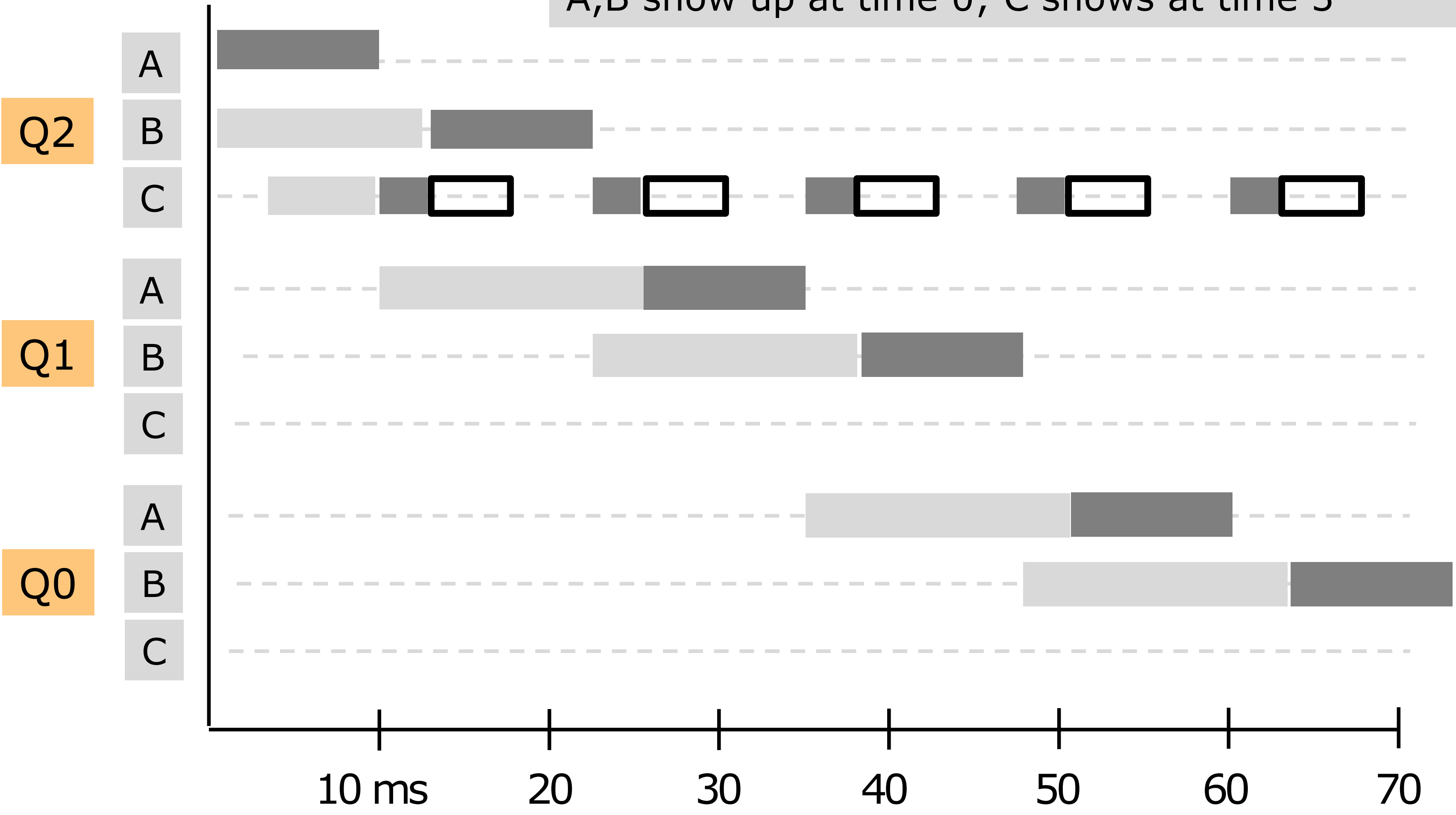
New rule for the game

3 queues, process A is CPU bound, 50 ms
process B is I/O bound, 10 ms, I/O every 2 ms

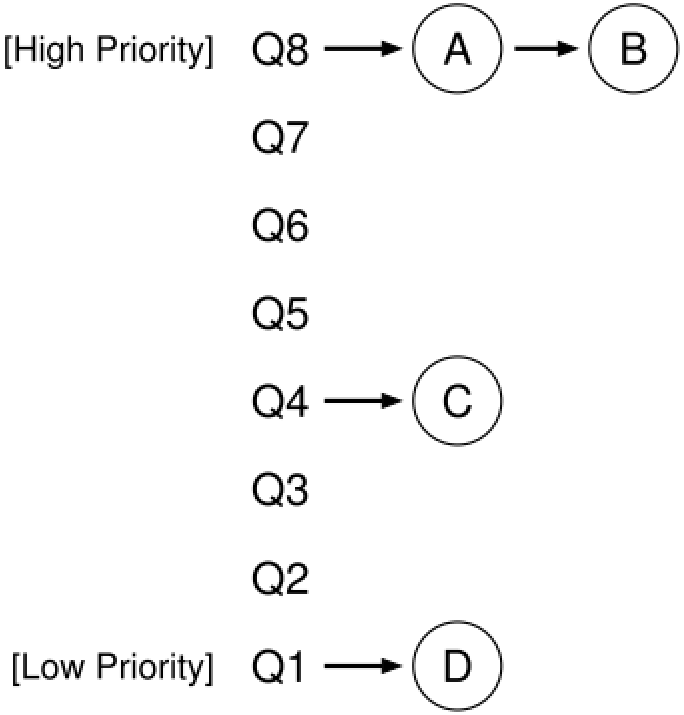


Play again, with a third process

3 queues, processes A,B are CPU bound, 50 ms
process C is I/O bound, 10 ms, I/O every 2 ms
A,B show up at time 0; C shows at time 3



Rules so far



situation	change to priority
job starts	highest priority
job gives up CPU before end of time slice	keep at current priority
job uses full time slice	reduce priority level by 1

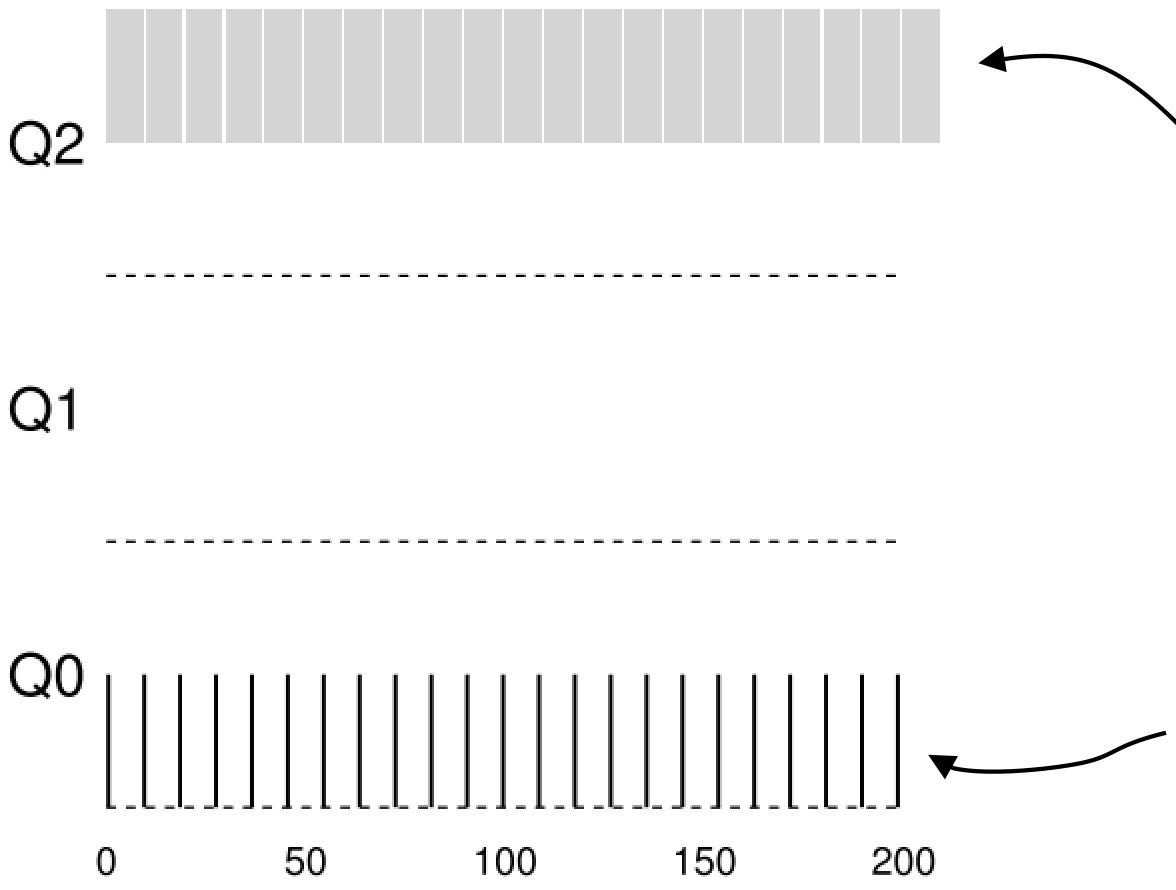
(diagram from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

Exercise

What are some potential problems with these rules?

- **Starvation**: if there are always higher-priority jobs, lower priority jobs will never run
- If a job is *sometimes* long-running, and sometimes interactive, it will sink to lowest priority
- A non-interactive job can “game the system” by doing a little I/O just before the end of its time slice – it will stay at highest priority

Example: Gaming the system

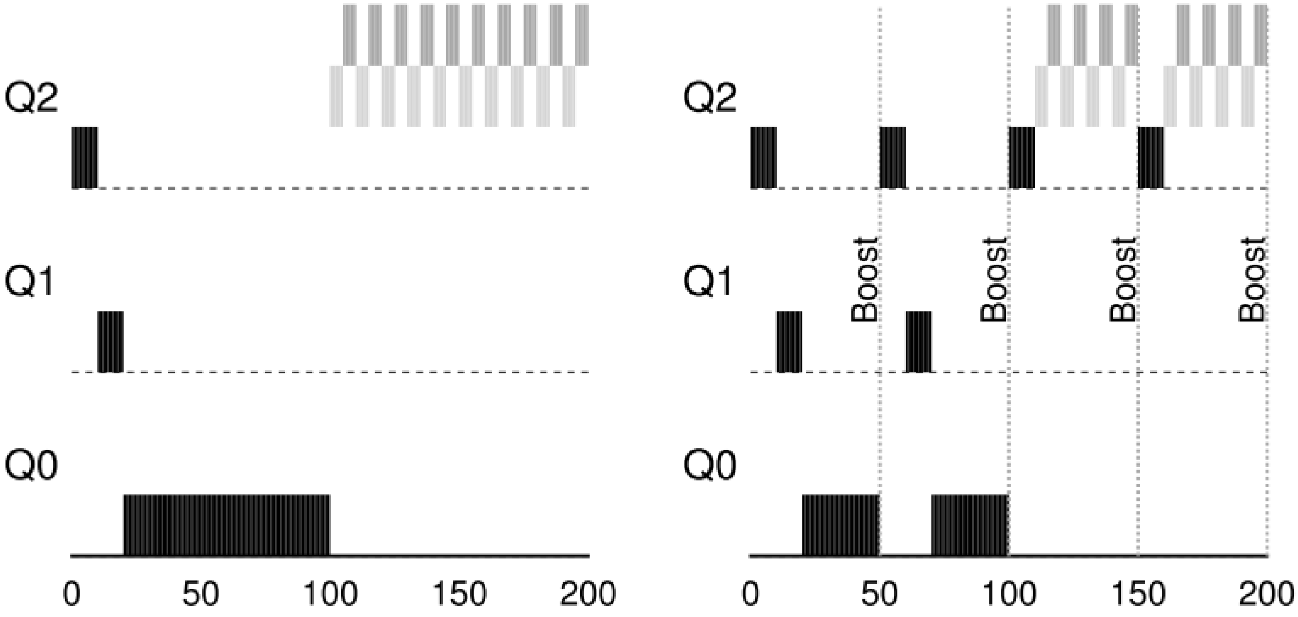


this job runs I/O just before end of time slice; retains high priority

this poor sucker didn’t cheat and loses out

(diagram from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

New rule: Boosting



without boosting

with boosting

New rule: occasionally move every job to highest priority

"Boosting"

(diagram from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

Exercise

Which of these problems are solved by boosting?

1. Starvation: if there are always higher-priority jobs, lower priority jobs will never run

2. If a job is *sometimes* long-running, and sometimes interactive, it will sink to lowest priority

3. A non-interactive job can “game the system” by doing a little I/O just before the end of its time slice – it will stay at highest priority

Problems with new solution

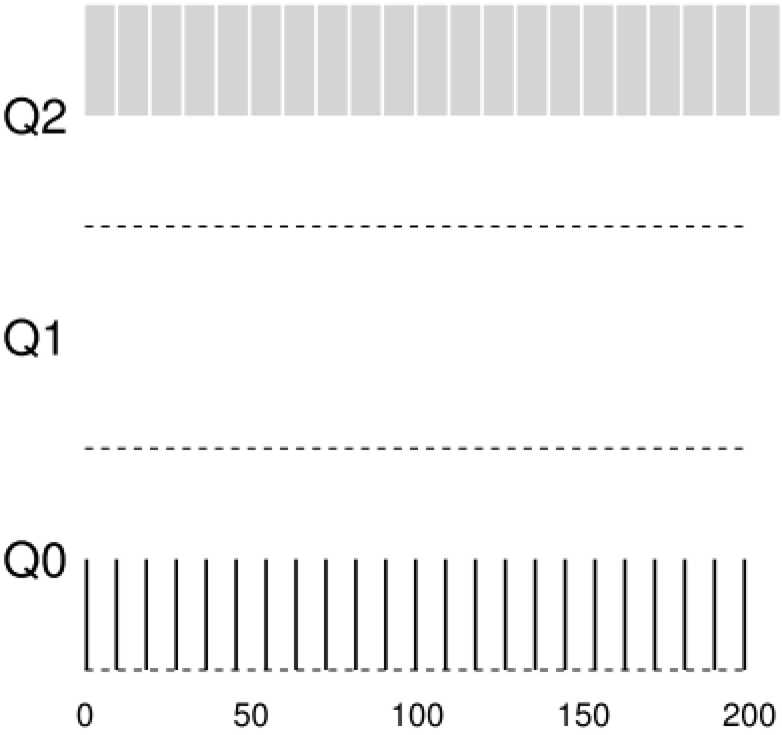
- The time interval S for boosting is an unfortunate system “tuning” parameter
- Some gaming still possible

Exercise:

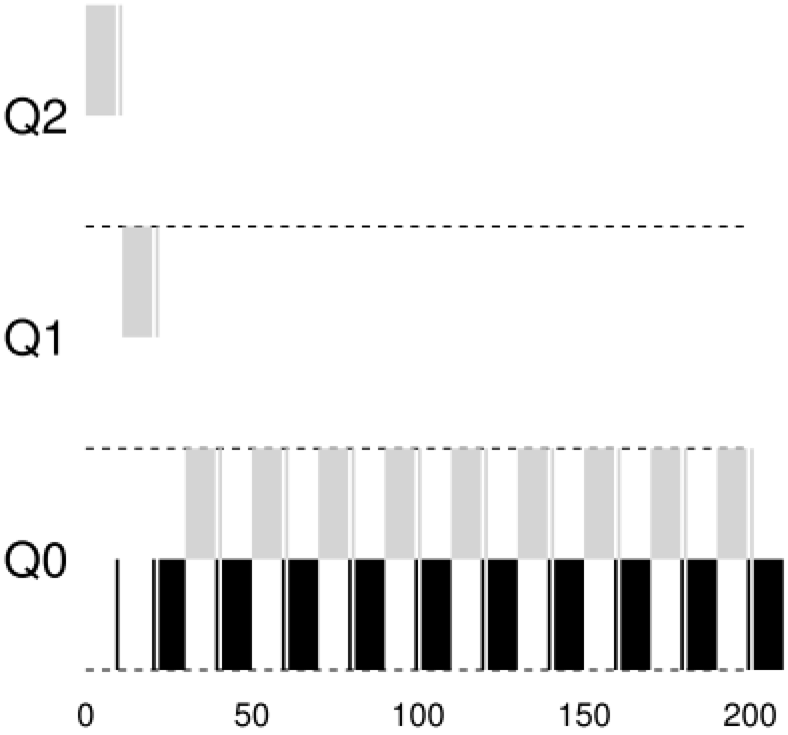
- what happens when S is too small?
- what happens when S is too large?

Improving MLFQ again

Revised rule: lower a job's priority when it has consumed its CPU allotment



before



after – using revised rule

(diagram from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

MLFQ rules for updating priorities

condition	change to priority
job starts	highest priority
job uses up time allotment at a given level	reduce priority level by 1
end of time period S	set all jobs to highest priority

Summary

The Multi-level Feedback Queue (MLFQ) scheduling policy maintains multiple job queues.

Jobs in higher-priority queues are processed first.

Longer, non-interactive jobs tend to move to lower-priority queues.

Many operating systems (Solaris, FreeBSD, Windows NT) use a form of MLFQ.