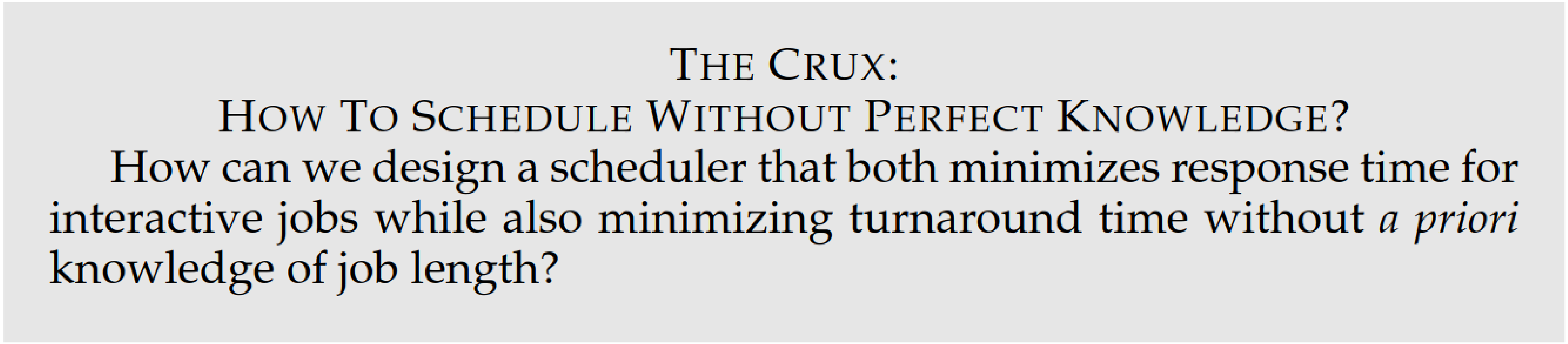
Chapter 8: Multi-Level Feedback Queue

Adam Disney

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Multi-Level Feedback Queue (MLFQ) Goals

- We want to optimize **turnaround time**, but we don't know how long jobs will run.
- We also want the system feel responsive to interactive users, so we want to optimize **response time** as well.



THE CRUX:
HOW TO SCHEDULE WITHOUT PERFECT KNOWLEDGE?

How can we design a scheduler that both minimizes response time for interactive jobs while also minimizing turnaround time without *a priori* knowledge of job length?

MLFQ: Basic Rules

- We're going to have multiple **queues**.
- Each is assigned a different **priority level**.
- When we need to make a scheduling decision, we pick the job with the highest priority.
- For jobs in the same priority queue, use Round Robin.

MLFQ: Basic Rules

- Rule 1: if Priority(A) > Priority(B), A runs.
- Rule 2: if Priority(A) = Priority(B), A & B run in RR.

MLFQ: Basic Rules

- The key to MLFQ scheduling lies in how we select priorities for jobs.
- For example, a job that often yields waiting on keyboard input might have a high priority because it is obviously interactive.
- MLFQ will learn based on history of observed behavior to make decisions.
  - For example, if a job keeps releasing control waiting on keyboard input, MLFQ might give it a high priority so that it is interactive.
  - In contrast, if a job uses the CPU for a long time, reduce its priority.

MLFQ: Basic Rules

- With our current rules, only A & B will run and never give C or D any CPU time.
- Job priority must change over time!

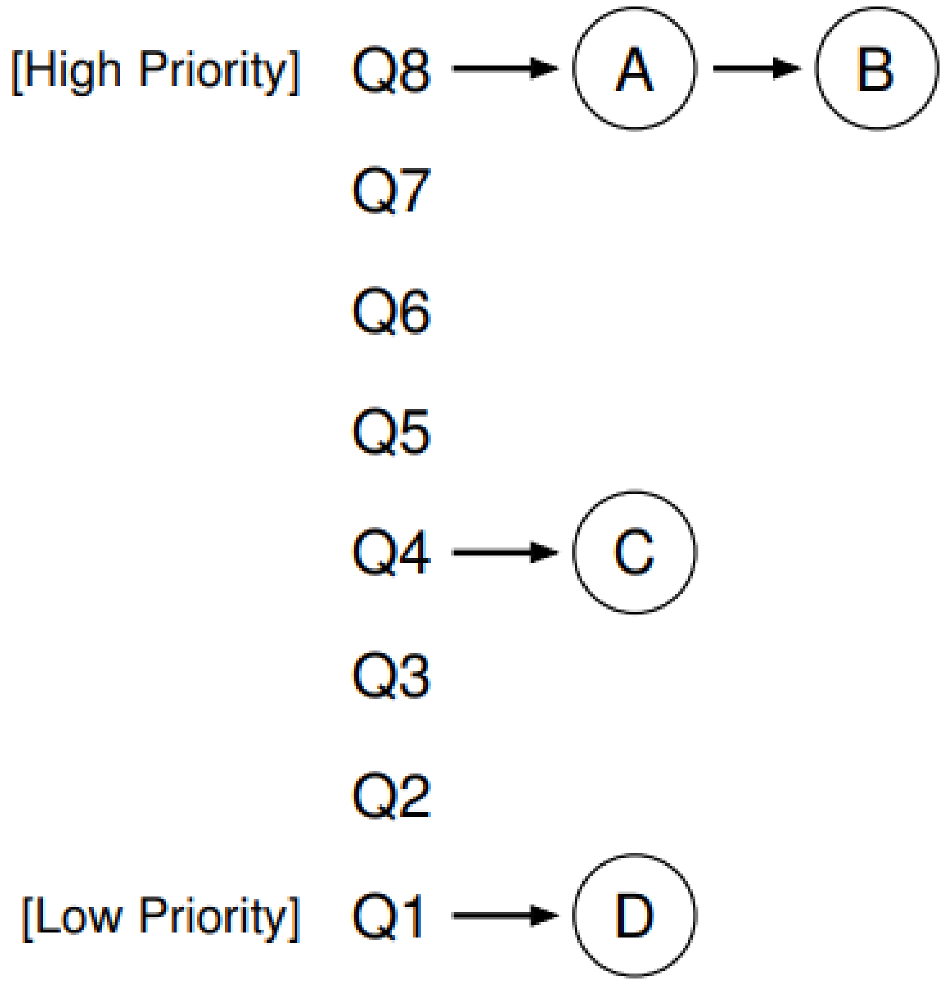


Figure 8.1: **MLFQ Example**

Attempt #1: How to Change Priority

- Each job is given an **allotment**. This is the amount of time the job can be in a given queue before its priority is reduced. For simplicity, we'll assume it's equal to a single time slice.
- Rule 3: When a job enters the system, it is placed at the highest priority.
- Rule 4a: If a job uses up an entire time slice while running, its priority is reduced.
- Rule 4b: If a job gives up the CPU before the time slice is up, it stays at the same priority level.

Example 1: A Single Long-Running Job

- Let's look at a long running job in a 3-queue scheduler.
- Starts at the top and quickly gets knocked down to the bottom.

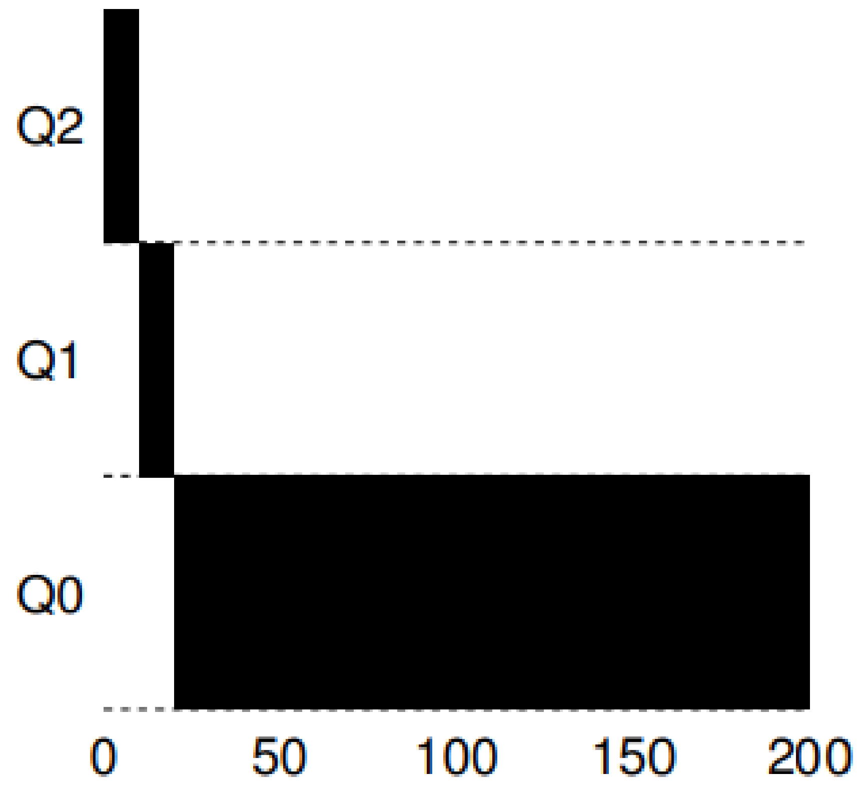


Figure 8.2: **Long-running Job Over Time**

Example 2: Along Came A Short Job

- We can see how MLFQ tries to approximate SJF.
- Assume jobs are short until they prove otherwise.

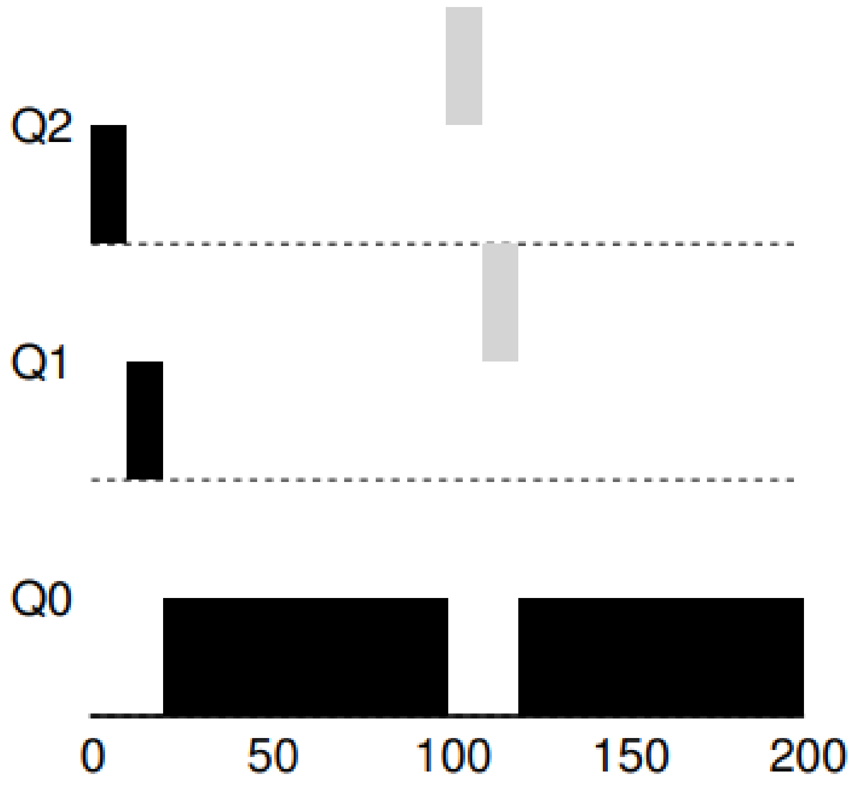


Figure 8.3: **Along Came An Interactive Job**

Example 3: What About I/O?

- Interactive jobs will stay at highest priority because it is likely that it will not use the entire time slice before blocking on I/O and remain at the top priority.

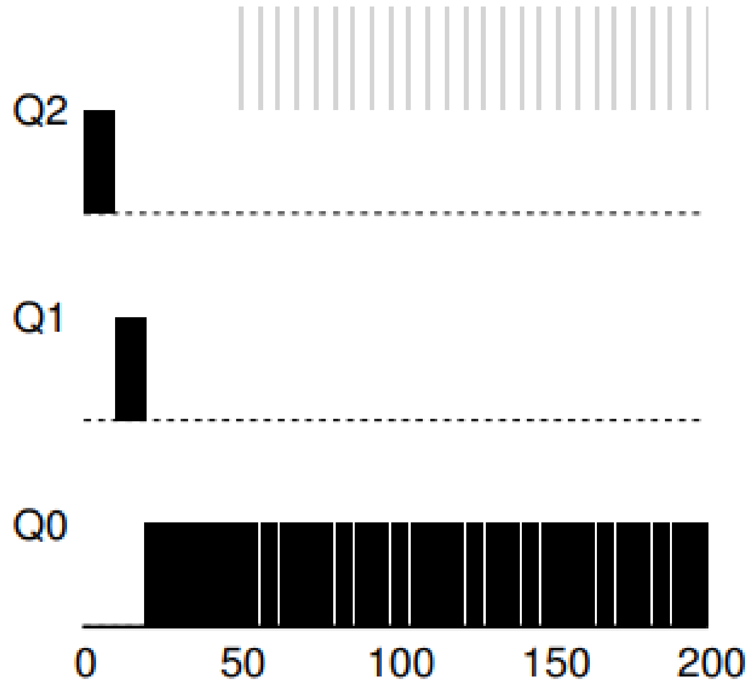


Figure 8.4: **A Mixed I/O-intensive and CPU-intensive Workload**

Problems With Our Current MLFQ

- Seems to work well but what can go wrong?

Issue #1: Starvation

- Too many interactive jobs will **starve** long running jobs.
- We need a way for jobs to increase in priority to avoid this.
- (Ignore the right-side figure for now)

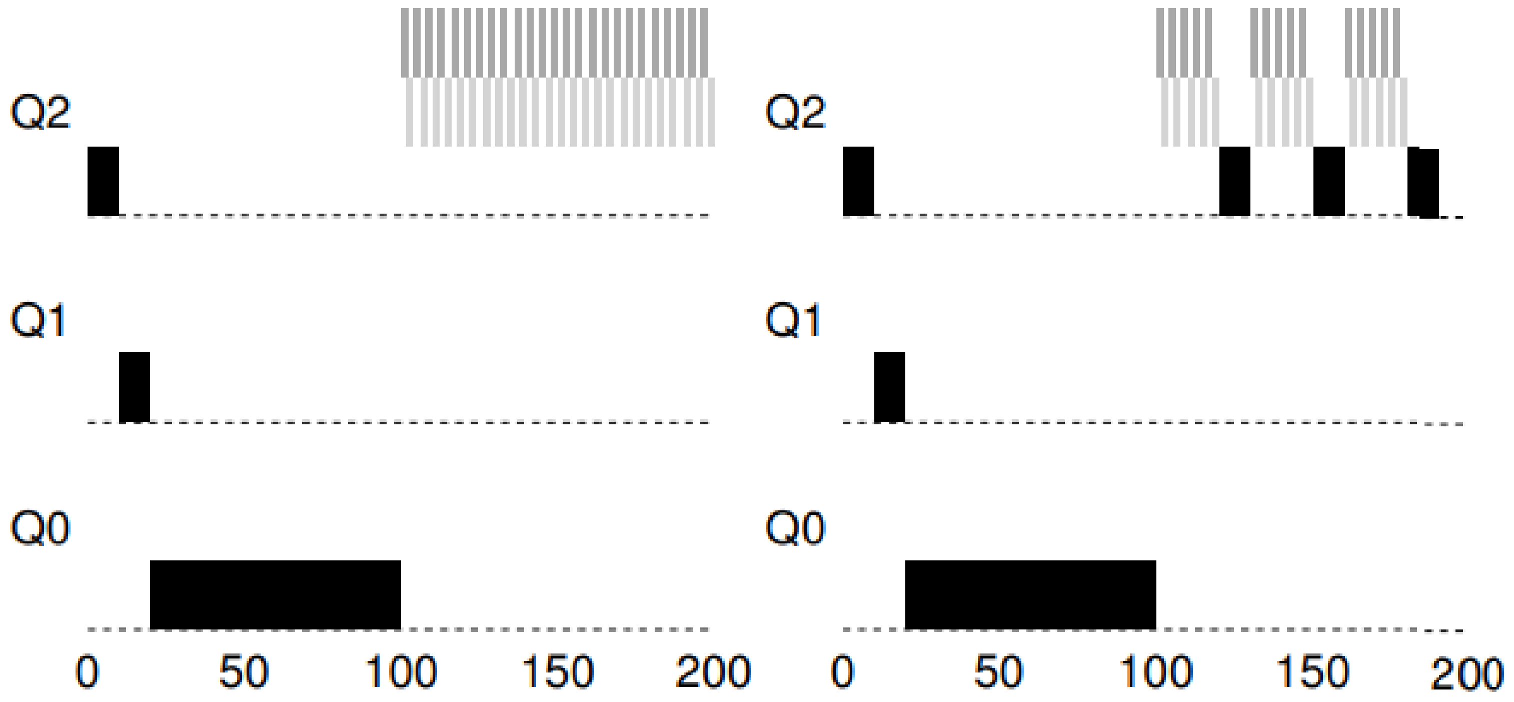


Figure 8.5: **Without (Left) and With (Right) Priority Boost**

Issue #2: Gaming the Scheduler

- A program could issue I/O or yield after 99% of the time slice to stay in the top priority queue.

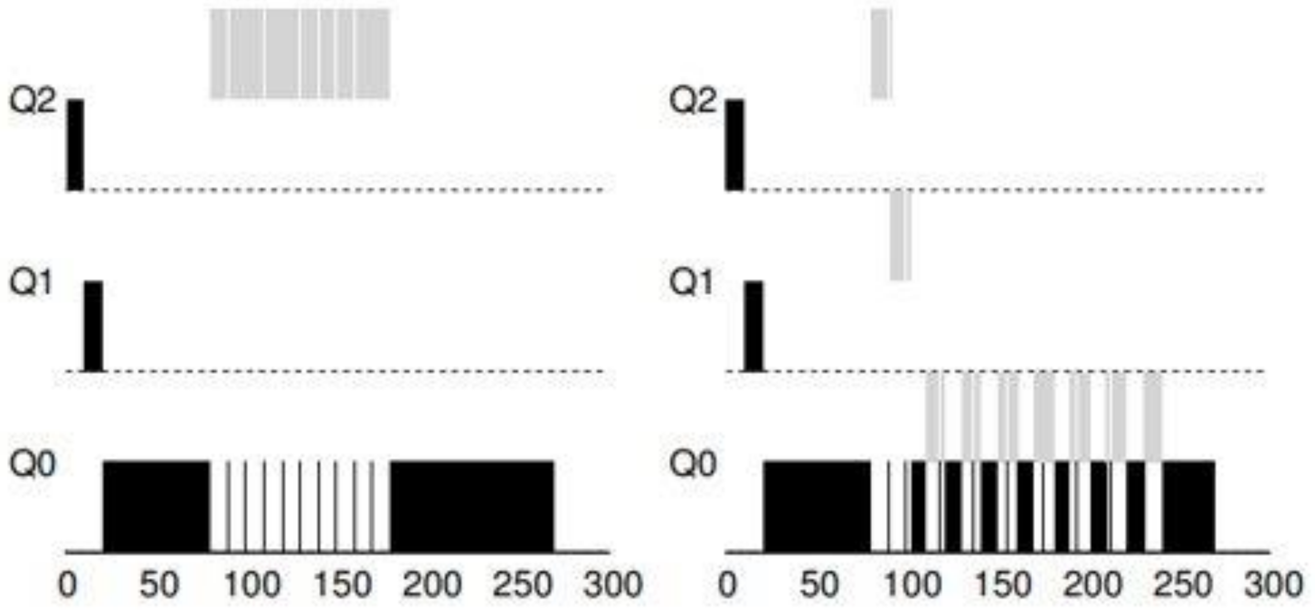


Figure 8.6: **Without (Left) and With (Right) Gaming Tolerance**

Issue #3: Changing Behavior

- A job's behavior might change over time
  - (CPU intensive <-> I/O intensive)
- A job that started non-interactive would not be prioritized well when it later became interactive.

Attempt #2: The Priority Boost

- Let's attempt to avoid starvation by boosting priority of jobs.
- Rule 5: After some time period S, move all the jobs in the system to the topmost queue.
- This guarantees no starvation.
- Low priority jobs that have become interactive will be treated properly.

Attempt #2: The Priority Boost

- The first job was CPU intensive at first but became interactive.
- With the priority boost, we account for this.
- That's two of the three issues solved...

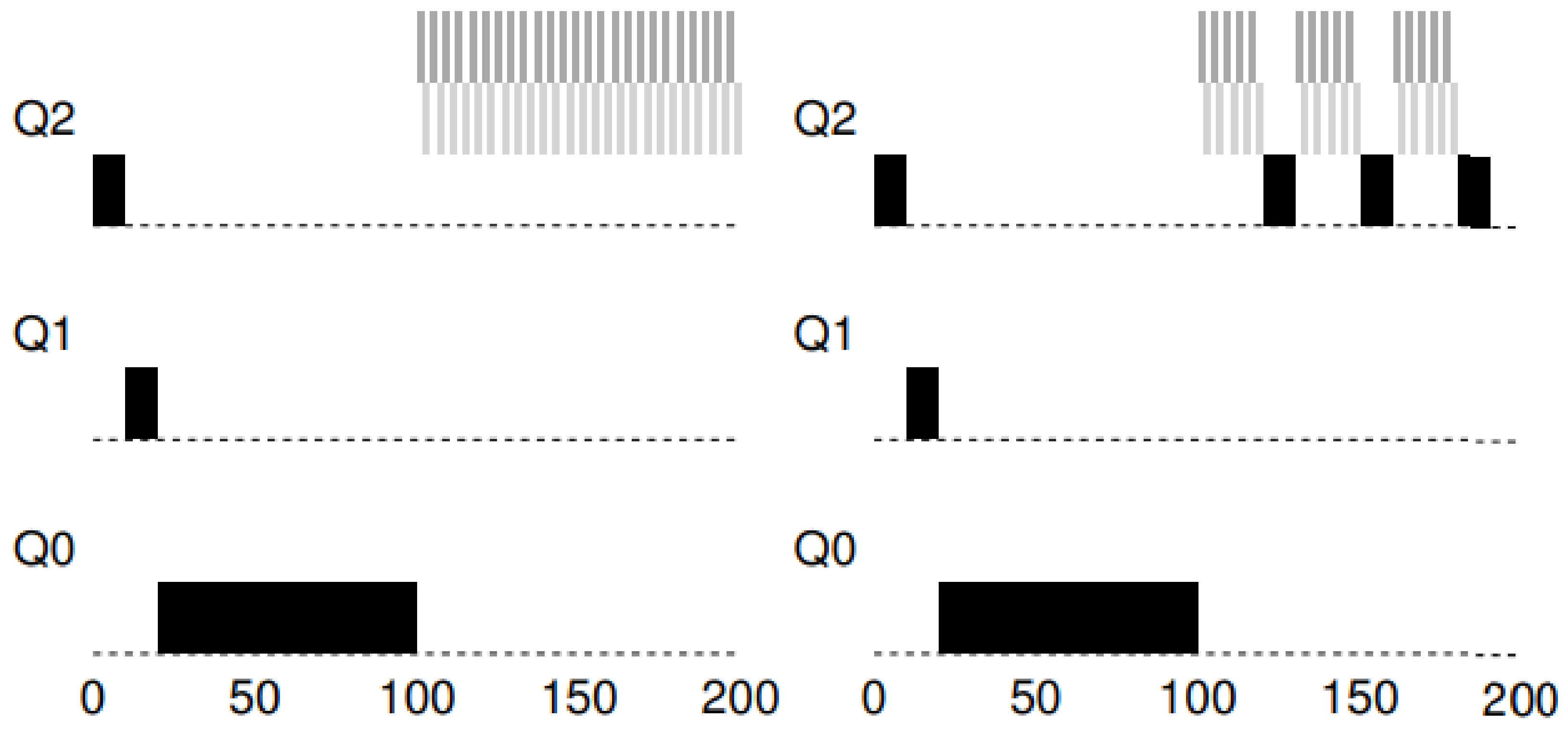


Figure 8.5: **Without (Left) and With (Right) Priority Boost**

Attempt #3: Better Accounting

- Let's try to prevent gaming of the scheduler.
- Let's change rule 4
- Rule 4: Once a job uses up its time allotment at a give level, its priority is reduced.

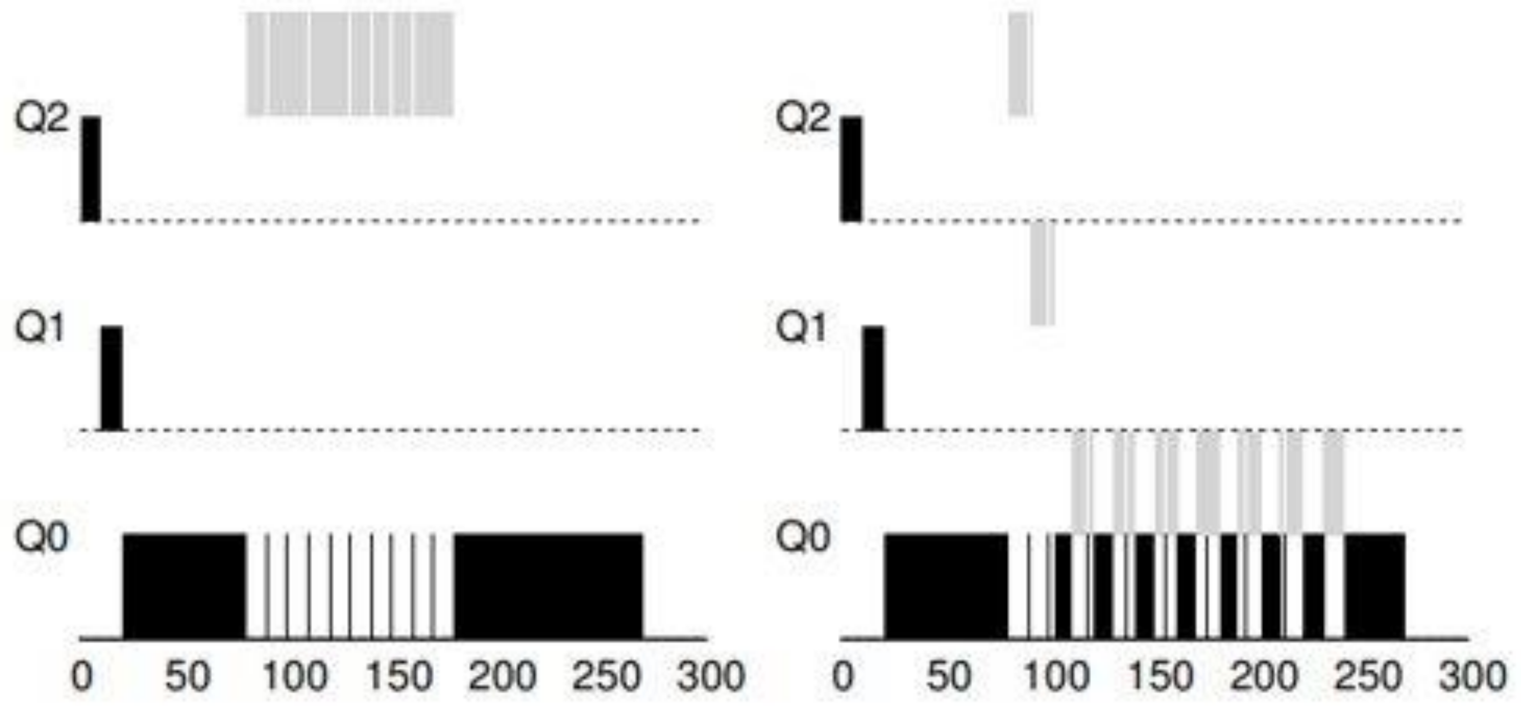


Figure 8.6: **Without (Left) and With (Right) Gaming Tolerance**

Tuning MLFQ and Other Issues

- **Parameterization**
  - How many queues?
  - How big are time slices at each level?
  - How often to boost priority?
- This is where understanding your workload is important.
- Scheduler implementations often have extra features like "nice"

MLFQ: Summary

- Rule 1: If Priority(A) > Priority(B), A runs
- Rule 2: If Priority(A) = Priority(B), A & B run in RR fashion using the time slice of the given queue
- Rule 3: When a job enters the system, it is placed at the highest priority
- Rule 4: Once a job uses up its time allotment at a given level, its priority is reduced
- Rule 5: After some time period S, move all the jobs in the system to the topmost queue