Lecture 5: Multi-Level Feedback Queue

Operating Systems

Content taken from: https://www.cse.iitb.ac.in/~mythili/os/

https://pages.cs.wisc.edu/~remzi/OSTEP/

Last Class

- **Workload assumptions which we relaxed one at a time:**
  - Each job runs for the same amount of time.
  - All jobs arrive at the same time.
  - Once started, each job runs to completion.
  - All jobs only use the CPU (i.e., they perform no I/O)
  - The run-time of each job is known.
- **Scheduling Policies:**
  - FIFO, SJF, STCF, RR
- **Scheduling metrics:**
  - Turnaround time and Response time

Multi-Level Feedback Queue (MLFQ)

- Optimize both turnaround time and response time
- 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

- General-purpose scheduling
- Support two types of jobs with distinct goals
  - **Interactive** jobs care about response time
  - **Batch** jobs care about turnaround time
- **Approach:**
  - Multiple levels of queues.
  - A higher-level queue has higher priority than lower level queues.
  - Round-Robin within each queue.
  - Preemption is allowed for a higher priority job to run in place of lower priority job.

Basic Rules

- **Rule 1:** If priority(A) > Priority(B), A runs
- **Rule 2:** If priority(A) == Priority(B), A & B run in RR

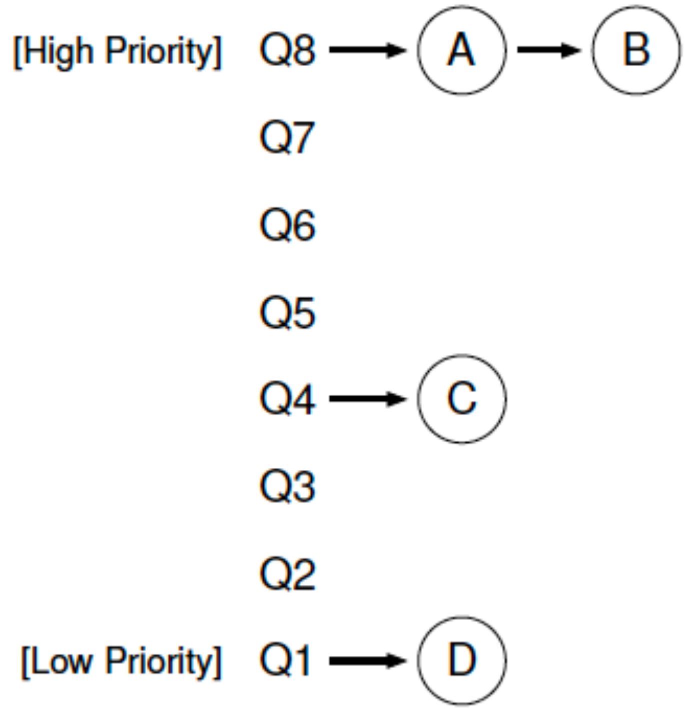


Figure 8.1: **MLFQ Example**

How to set priority for each job?

- Use past behavior of process to predict future behavior
- Processes alternate between I/O and CPU work
- Guess how CPU burst (job) will behave based on past CPU bursts (jobs) of this process

More Rules

- **Rule 1:** If priority(A) > Priority(B), A runs
- **Rule 2:** If priority(A) == Priority(B), A & B run in RR
- **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue)
- **Rule 4:** If a job uses up an entire time slice while running, its priority is **reduced** (i.e. it moves down one queue)
- **Rule 5:** If a job gives up the CPU before the time slice is up, it stays at the **same** priority level.

Example: A Single Long-Running Job

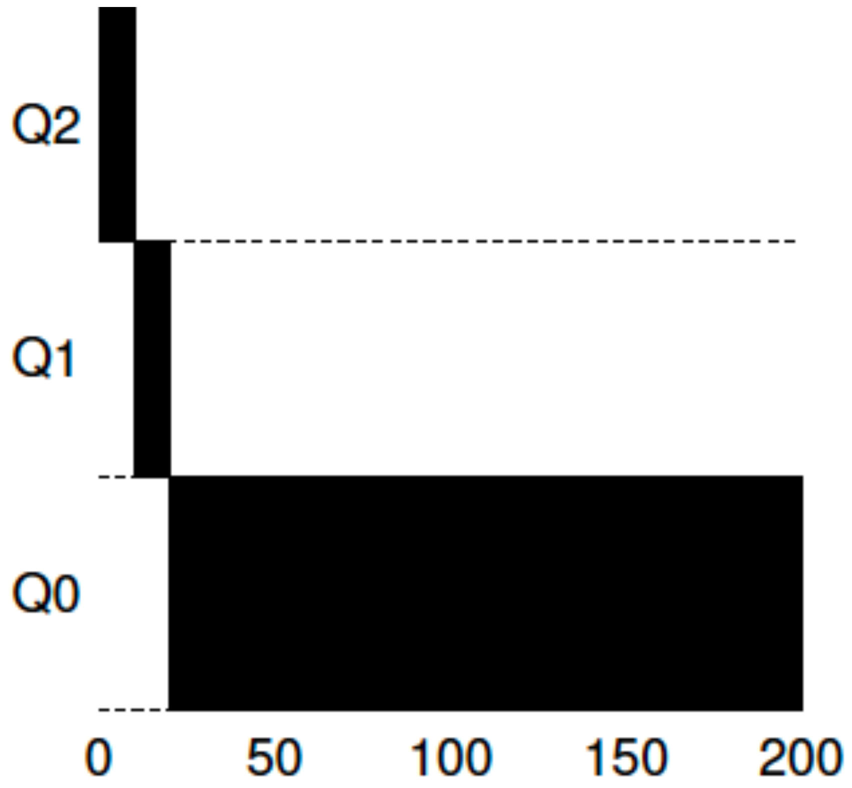


Figure 8.2: **Long-running Job Over Time**

Along Came a Short-Running Interactive Job

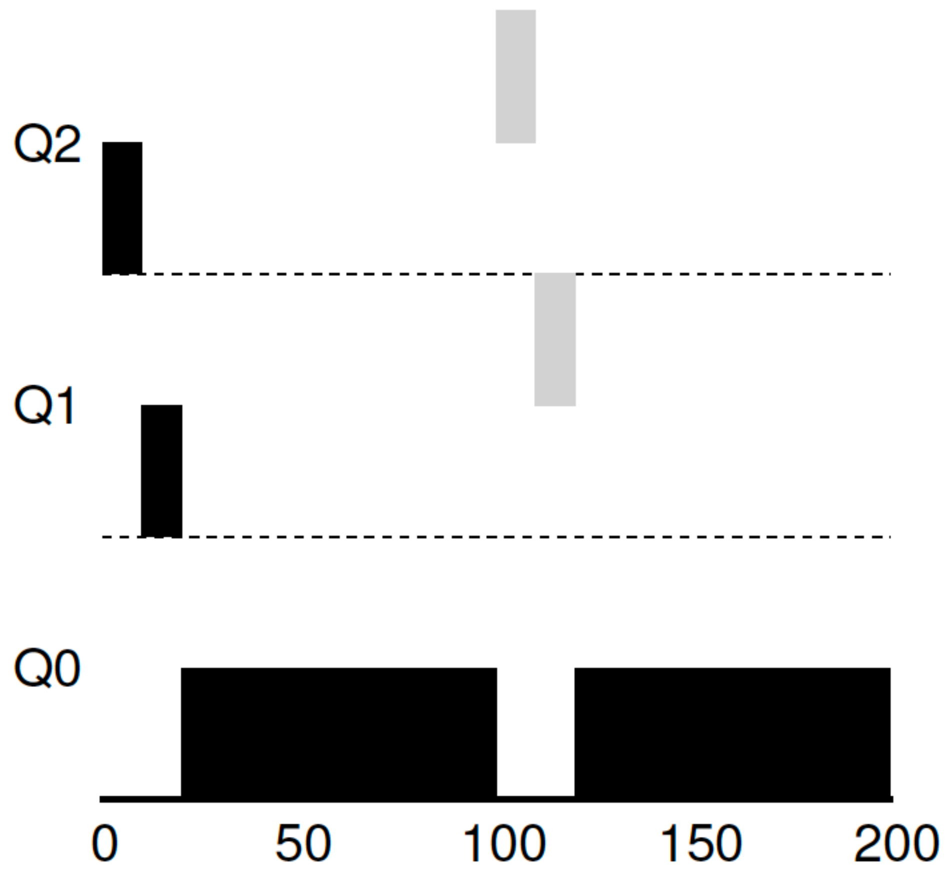


Figure 8.3: **Along Came An Interactive Job**

What about I/O?

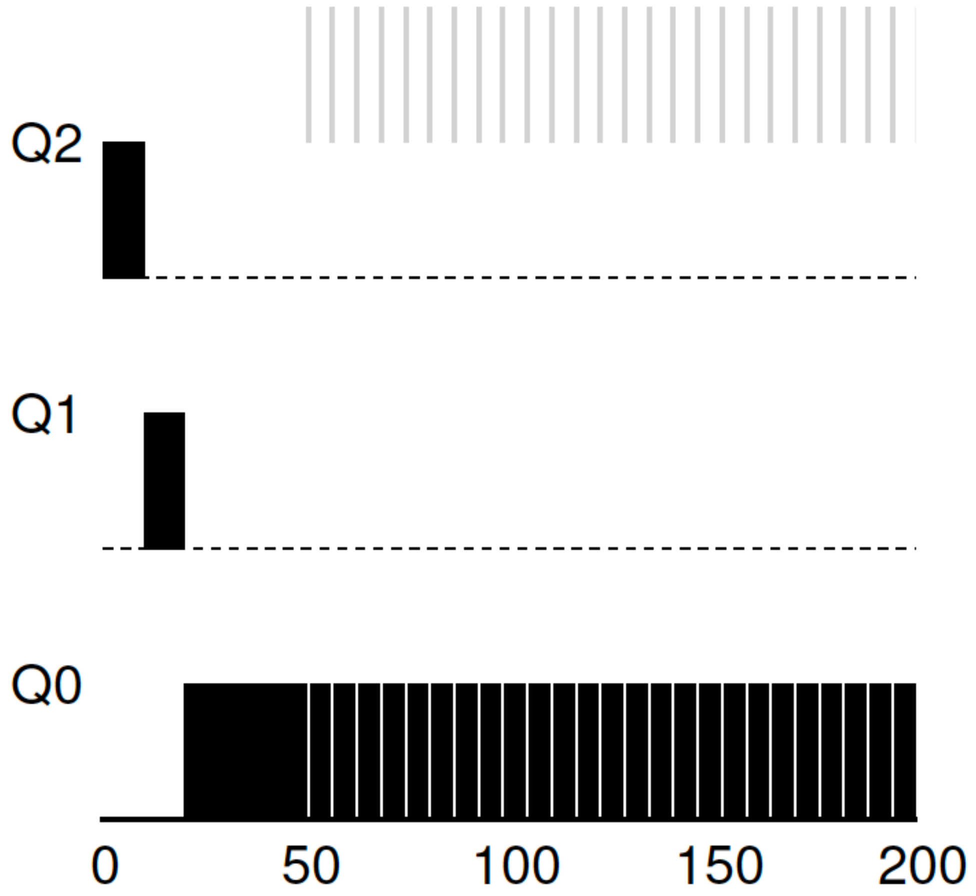


Figure 8.4: **A Mixed I/O-intensive and CPU-intensive Workload**

Any problems with the current MLFQ?

Any problems with the current MLFQ?

- Starvation
- Gaming the Scheduler
- A job may change its behavior over time
  - From being CPU-bound to a phase of interactivity

Prevent Starvation: The Priority Boost

- Periodically boost the priority of all the jobs in system
- **Rule 5:** After some time-period S, move all the jobs in the system to the topmost queue.
- Solves the problem of starvation and jobs changing behavior over time.

Prevent Starvation: The Priority Boost

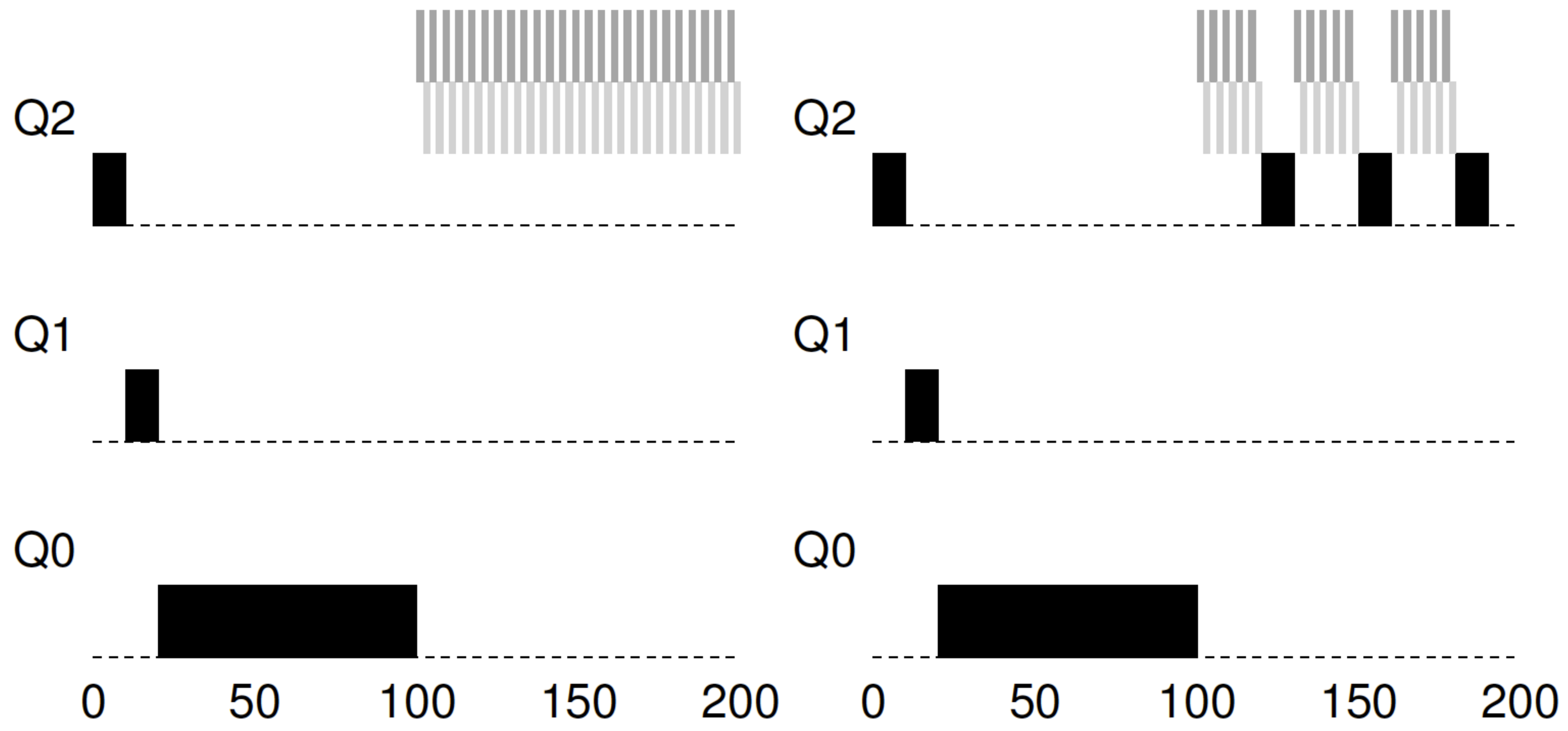


Figure 8.5: **Without (Left) and With (Right) Priority Boost**

Prevent Gaming: Better Accounting

- Instead of forgetting how much of a time slice a process used at a given level, the scheduler should keep track; once a process has used its allotment, it is demoted to the next priority queue
- **Rule 4 rewritten:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).

Prevent Gaming: Better Accounting

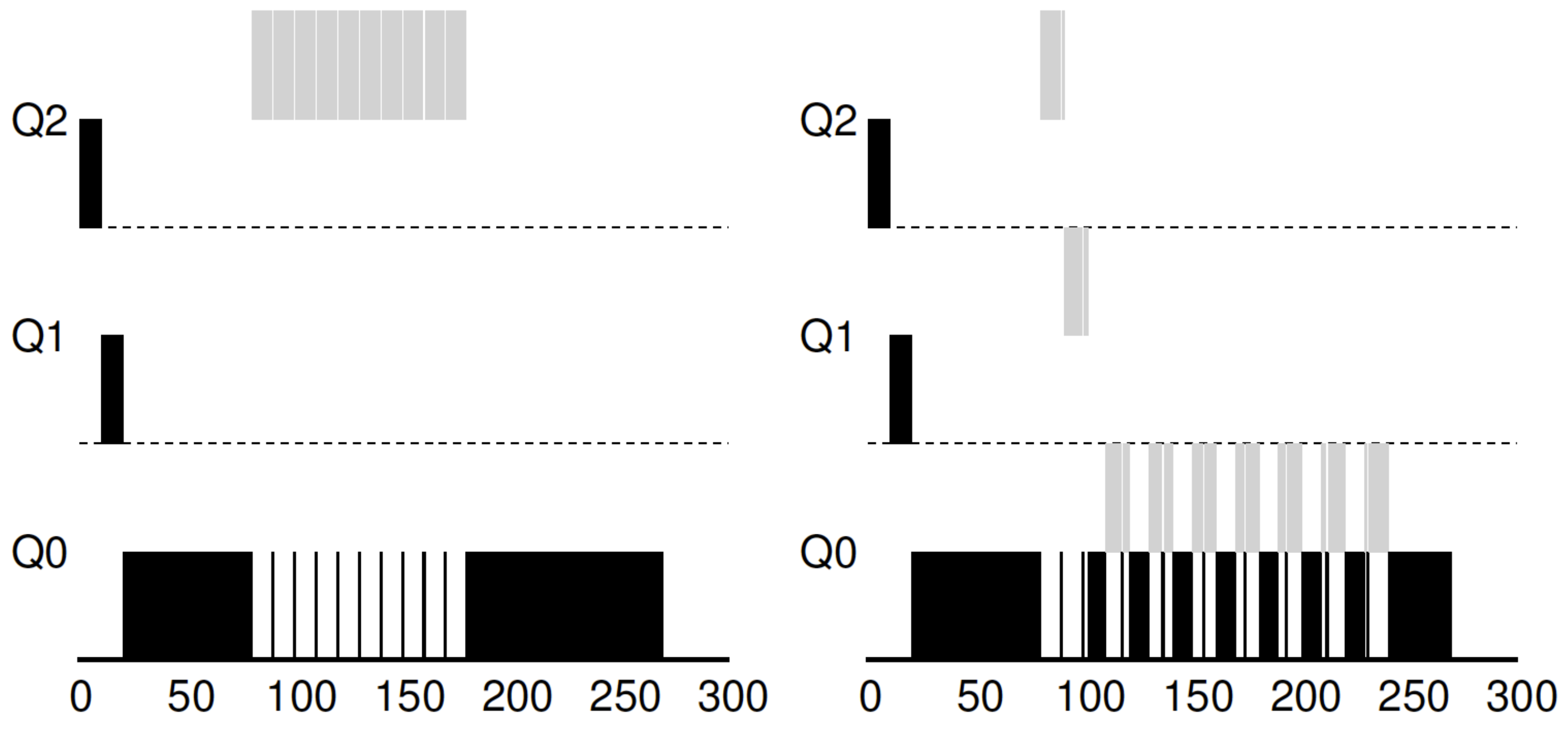


Figure 8.6: **Without (Left) and With (Right) Gaming Tolerance**

Varying time slice per queue

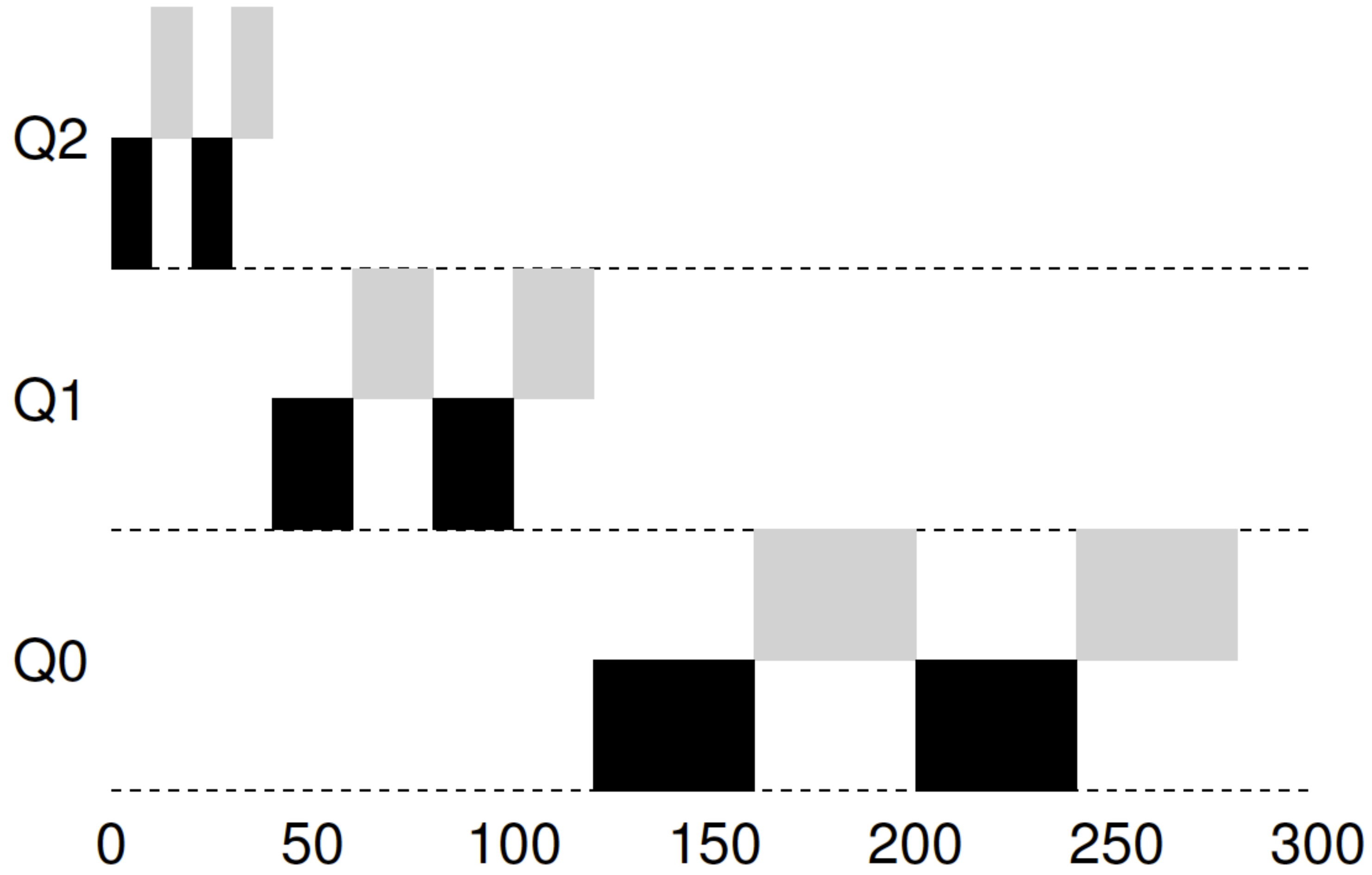
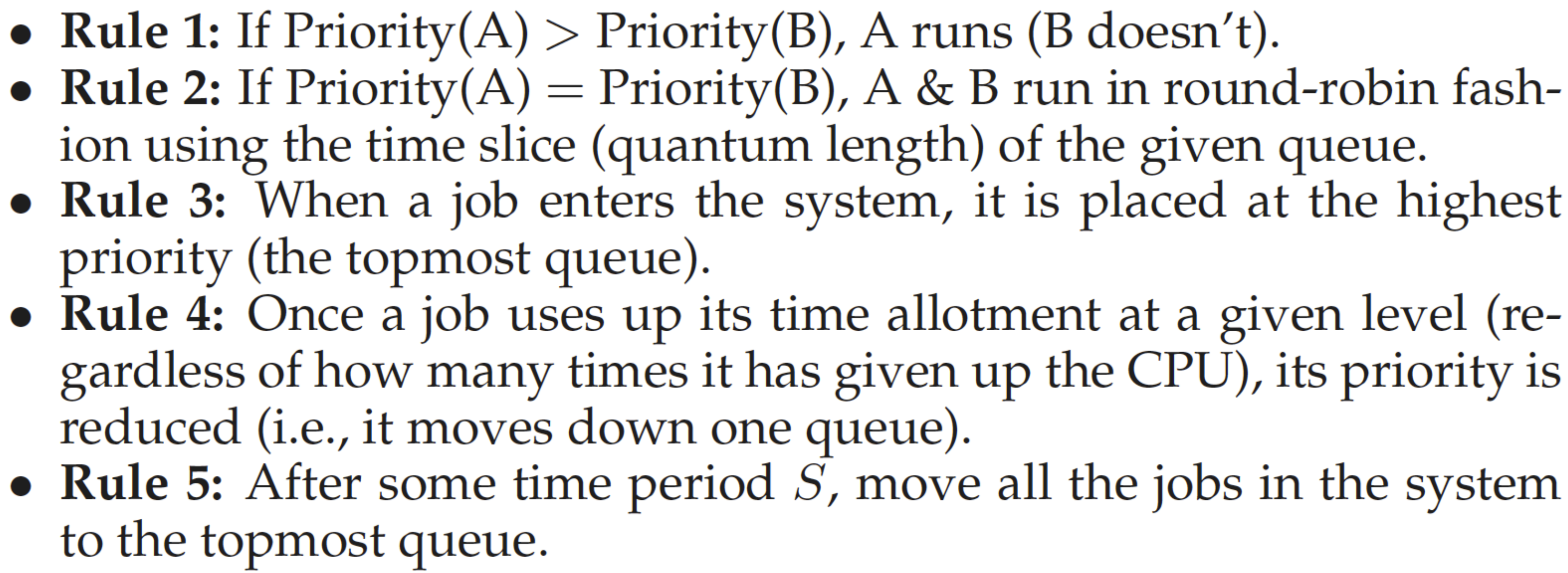


Figure 8.7: **Lower Priority, Longer Quanta**

Tuning MLFQ

- How to set the parameters associated with MLFQ?
    - Number of queues
    - Size of time slice per queue
    - Frequency of priority boost
- No easy answers
- Set these parameters based on
    - Understanding of workloads
    - Trial and Error results

MLFQ: Final Set of Rules



- **Rule 1:** If Priority(A) $>$ Priority(B), A runs (B doesn't).
- **Rule 2:** If Priority(A) $=$ Priority(B), A & B run in round-robin fashion using the time slice (quantum length) of the given queue.
- **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue).
- **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- **Rule 5:** After some time period S, move all the jobs in the system to the topmost queue.

MLFQ: Summary

- Multiple levels of queues for prioritizing different types of jobs
- Use feedback to determine the priority of a given job
- Delivers excellent performance for short-running interactive jobs
- Makes progress for long-running CPU-intensive workloads

Lottery Scheduling

- **Goal:** Guarantee each job to obtain a certain percentage of CPU time based on its priority
- **Approach:**
  - Give lottery tickets to all the processes
  - Higher priority process has more lottery tickets
  - Hold a lottery for every time slice

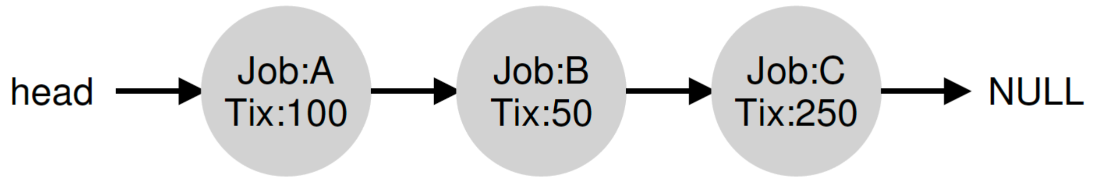
Lottery Example

```
// counter: used to track if we've found the winner yet
int counter = 0;

// winner: use some call to a random number generator to
//         get a value, between 0 and the total # of tickets
int winner = getrandom(0, totaltickets);

// current: use this to walk through the list of jobs
node_t *current = head;
while (current) {
    counter = counter + current->tickets;
    if (counter > winner)
        break; // found the winner
    current = current->next;
}
// 'current' is the winner: schedule it...
```

Figure 9.1: **Lottery Scheduling Decision Code**


head
Job:A
Tix:100
Job:B
Tix:50
Job:C
Tix:250
NULL