

8: Scheduling: The Multi-Level Feedback Queue

Operating System: Three Easy Pieces

Multi-Level Feedback Queue (MLFQ)

- ▣ A Scheduler that learns from the past to predict the future.
- ▣ Objective:
 - ◆ Optimize **turnaround time** → Run shorter jobs first
 - ◆ Minimize **response time** without *a priori knowledge of job length*.

MLFQ: Basic Rules

- MLFQ has a number of distinct **queues**.
 - ◆ Each queue is assigned a different priority level.
- A job that is ready to run is on a single queue.
 - ◆ A job **on a higher queue** is chosen to run.
 - ◆ Use round-robin scheduling among jobs in the same queue

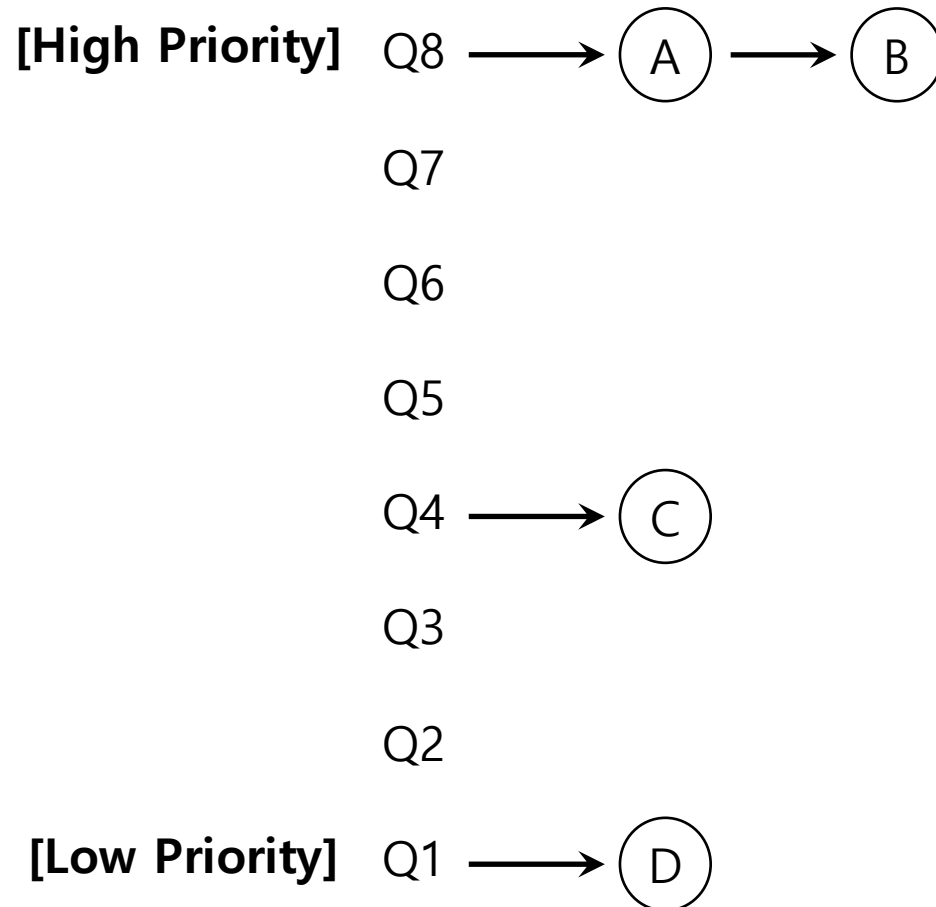
Rule 1: If $\text{Priority}(A) > \text{Priority}(B)$, A runs (B doesn't).

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

MLFQ: Basic Rules (Cont.)

- ▣ MLFQ varies the priority of a job based on **its observed behavior**.
- ▣ Example:
 - ◆ A job repeatedly relinquishes the CPU while waiting IOs → Keep its priority high
 - ◆ A job uses the CPU intensively for long periods of time → Reduce its priority.

MLFQ Example



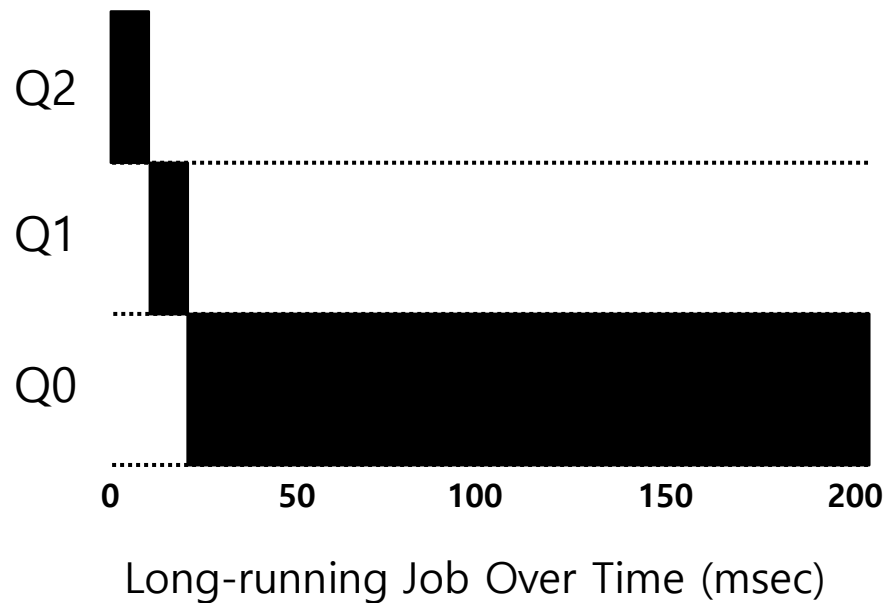
MLFQ: How to Change Priority

- MLFQ priority adjustment algorithm:
 - ◆ **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 (i.e., it moves down on queue).
 - ◆ **Rule 4b:** If a job gives up the CPU before the time slice is up, it stays at the same priority level

In this manner, MLFQ approximates SJF

Example 1: A Single Long-Running Job

- A three-queue scheduler with time slice 10ms



Example 2: Along Came a Short Job

□ Assumption:

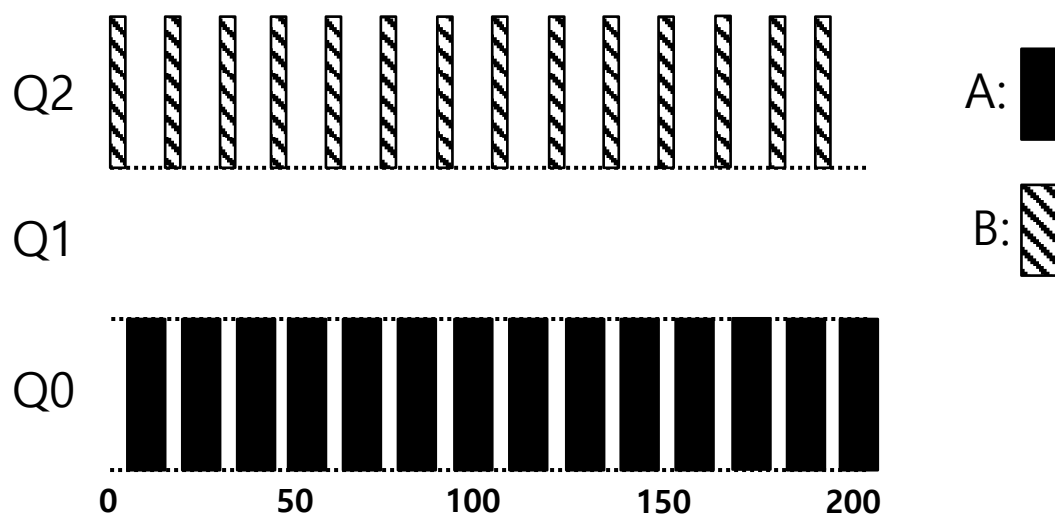
- ◆ **Job A:** A long-running CPU-intensive job
- ◆ **Job B:** A short-running interactive job (20ms runtime)
- ◆ A has been running for some time, and then B arrives at time $T=100$.



Example 3: What About I/O?

□ Assumption:

- ◆ **Job A:** A long-running CPU-intensive job
- ◆ **Job B:** An interactive job that need the CPU only for 1ms before performing an I/O



A Mixed I/O-intensive and CPU-intensive Workload (msec)

The MLFQ approach keeps an interactive job at the highest priority

Problems with the Basic MLFQ

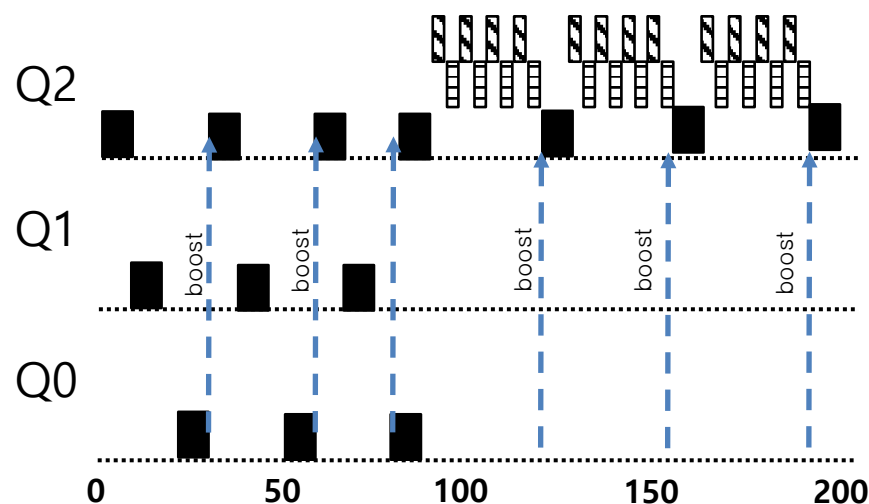
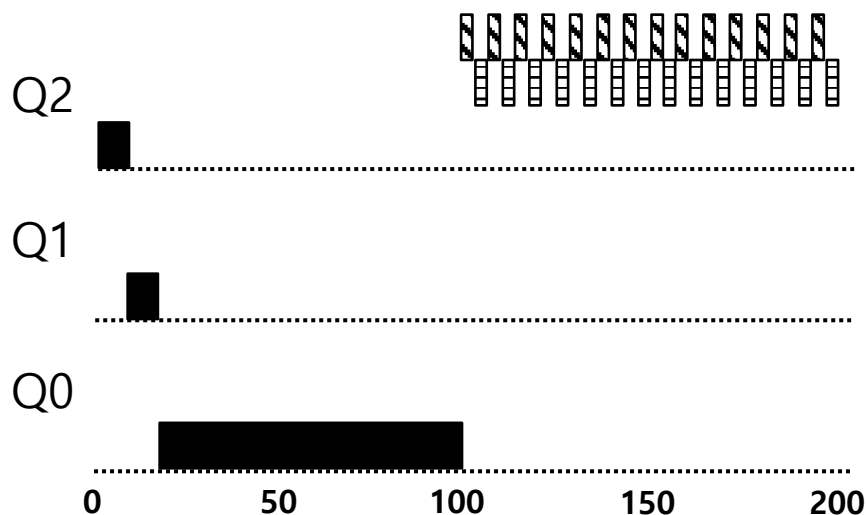
- ❑ Starvation
 - ◆ If there are “too many” interactive jobs in the system.
 - ◆ Lon-running jobs will never receive any CPU time.
- ❑ Game the scheduler
 - ◆ After running 99% of a time slice, issue an I/O operation.
 - ◆ The job gain a higher percentage of CPU time.
- ❑ A program may change its behavior over time.
 - ◆ CPU bound process → I/O bound process

Attempt #3: The Priority Boost

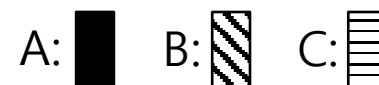
- ▣ **Rule 5:** After some time period S , move all the jobs in the system to the topmost queue.

- ◆ Example:

- A long-running job(A) with two short-running interactive job(B, C)

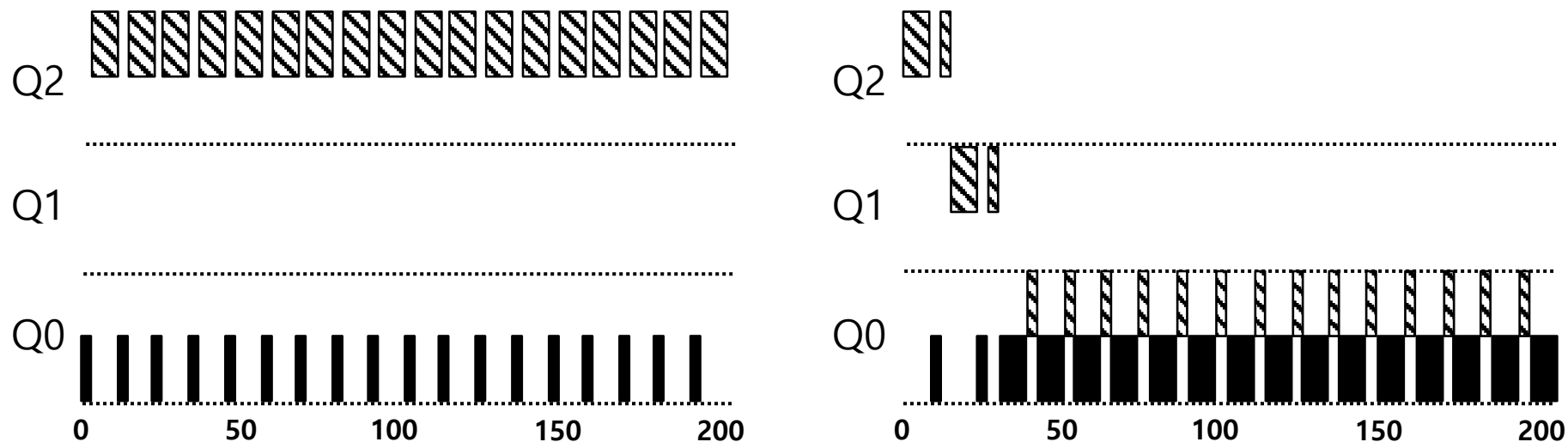


Without(Left) and With(Right) Priority Boost



Attempt #3: Better Accounting

- How to prevent gaming of our scheduler?
- Solution:
 - ◆ **Rule 4** (Rewrite Rules 4a and 4b): 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 on queue).

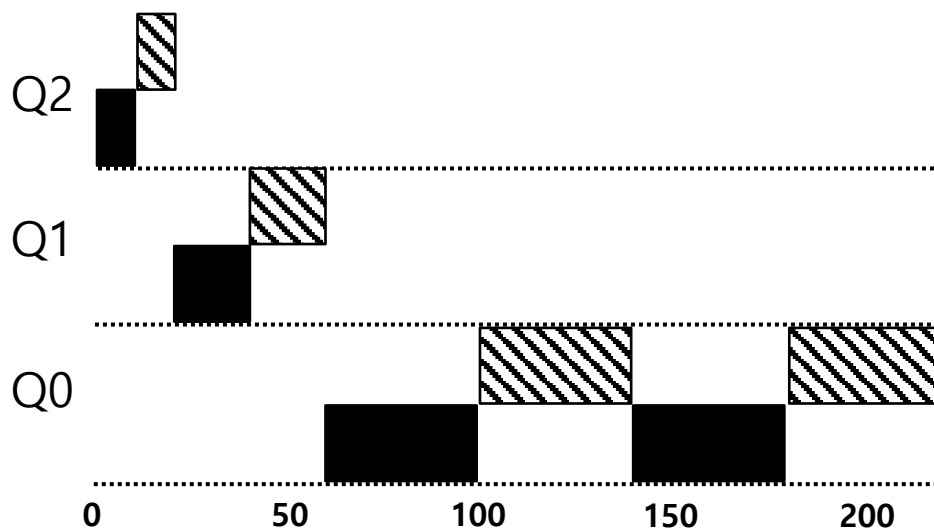


Without(Left) and With(Right) Gaming Tolerance

Tuning MLFQ And Other Issues

Lower Priority, Longer Quanta

- ◆ The high-priority queues → Short time slices
 - E.g., 10 or fewer milliseconds
- ◆ The Low-priority queue → Longer time slices
 - E.g., 100 milliseconds



Example) 10ms for the highest queue, 20ms for the middle,
40ms for the lowest

How to paramtrize the scheduler?

- ❑ Not an easy but critical task (it's workload dependent)
- ❑ The Solaris MLFQ implementation (exposed to sys adm.)
 - ◆ For the Time-Sharing scheduling class (TS)
 - 60 Queues
 - Slowly increasing time-slice length
 - The highest priority: 20msec
 - The lowest priority: A few hundred milliseconds
 - Priorities boosted around every 1 second or so
- ❑ Some hints from the user/sysadm
 - ◆ `nice`, `renice`

MLFQ: Summary

- The refined set of MLFQ rules:
 - ◆ **Rule 1:** If $\text{Priority}(A) > \text{Priority}(B)$, A runs (B doesn't).
 - ◆ **Rule 2:** If $\text{Priority}(A) = \text{Priority}(B)$, A & B run in RR.
 - ◆ **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 (regardless of how many times it has given up the CPU), its priority is reduced(i.e., it moves down on queue).
 - ◆ **Rule 5:** After some time period S , move all the jobs in the system to the topmost queue.

- Disclaimer: This lecture slide set is used in AOS course at University of Cantabria. Was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book written by Remzi and Andrea Arpaci-Dusseau (at University of Wisconsin)