Scheduling: The Multi-Level Feedback Queue

Operating System (OS)
안인규

강의자료는 KAIST 원유집 교수님 OS 강의 자료를 참고했습니다.

Recap: No More Oracle

1. ~~Each job runs for the **same amount of time.**~~
2. ~~All jobs **arrive** at the same time.~~
3. ~~Once started, each job runs to completion~~
4. ~~All jobs only use the **CPU** (i.e., they perform no I/O).~~
5. ~~The **run-time** of each job is known.~~

- Let's relax assumption 5: The run-time of each job is known
  - The OS usually knows very little about the length of each job
  - How can we build an approach that behaves like SJF/STCF without such *a priori* knowledge?

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

- Multi-Level Feedback Queue (MLFQ) has a number of distinct queues
  - Each queues is assigned a different priority level

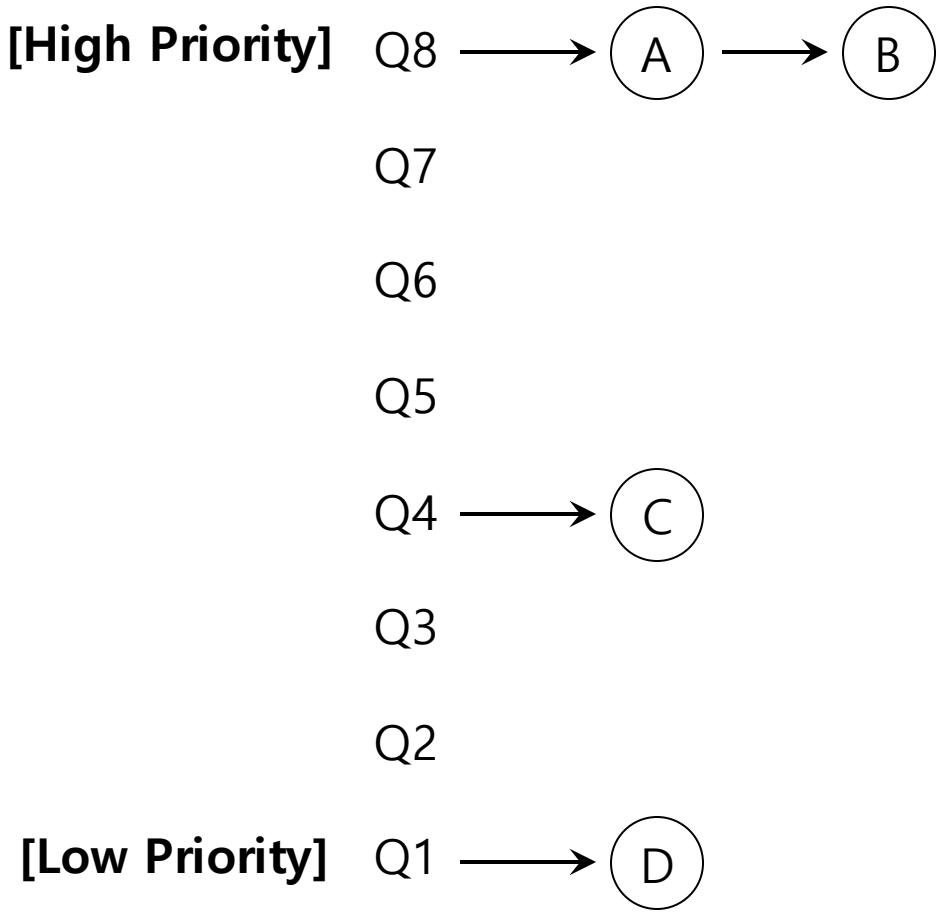
- A job that is ready to run in on a single queue
  - A job on a **higher queue** is chosen to run
  - Use **round-robin** scheduling amount jobs in the same queue

Rule 1: If Priority(A) > Priority(B), A runs (B doesn't).
Rule 2: If Priority(A) = Priority(B), A & B run in RR.

MLFQ: Basic Rules

- MLFQ varies the priority of a job based on its observed behavior (관찰된 동작)
- Example:
  - 작업이 I/O를 기다리면서 반복적으로 CPU를 포기하면 → 우선순위를 높게 유지
  - 작업이 오랜 시간 동안 CPU를 집중적으로 사용하면 → 우선순위를 낮춤

MLFQ: Example



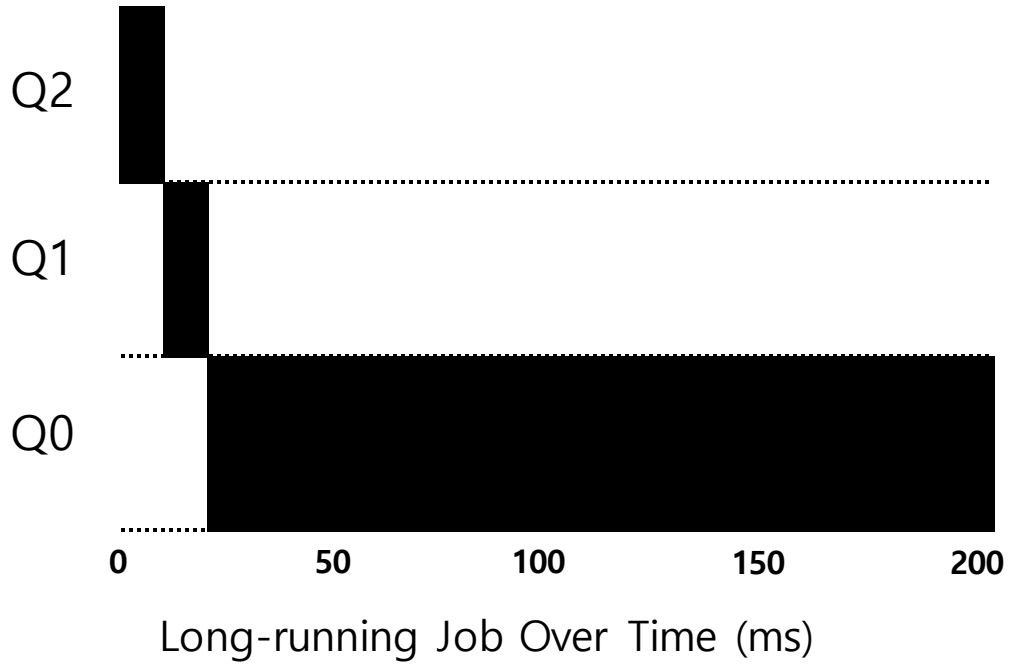
MLFQ: How to Change Priority

- 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

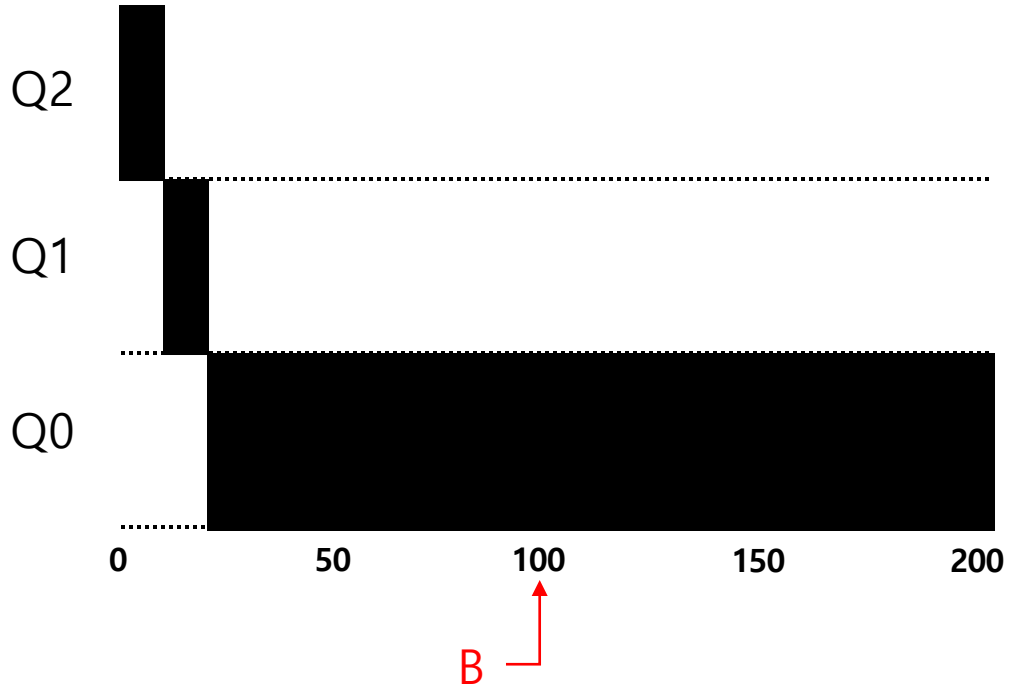
MLFQ: Example 1

- A single long-running job
  - A three-queue scheduler with time slice 10ms



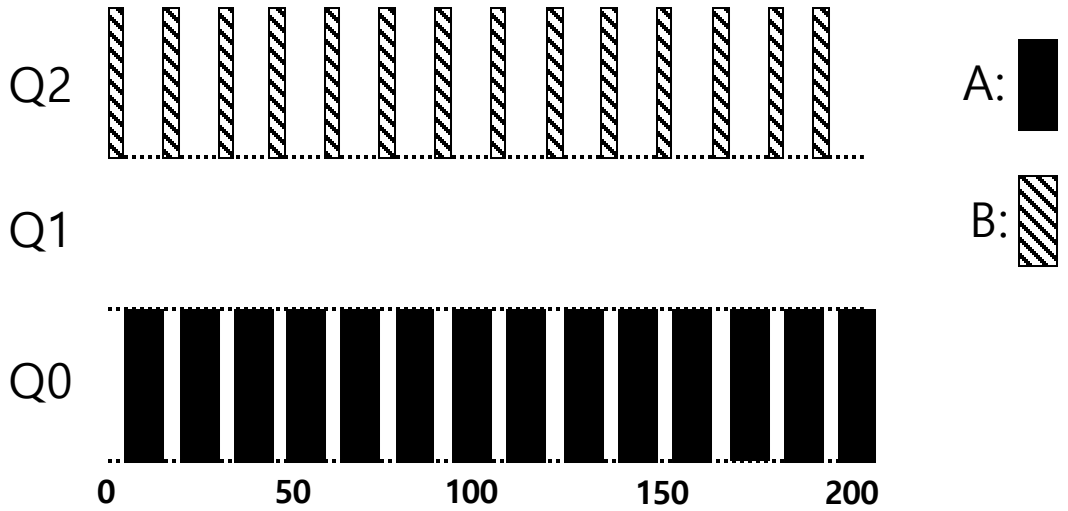
MLFQ: Example 2

- 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



MLFQ: Example 3

- 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 (ms)

The MLFQ approach keeps an interactive job at the highest priority

Problems with the Basic MLFQ

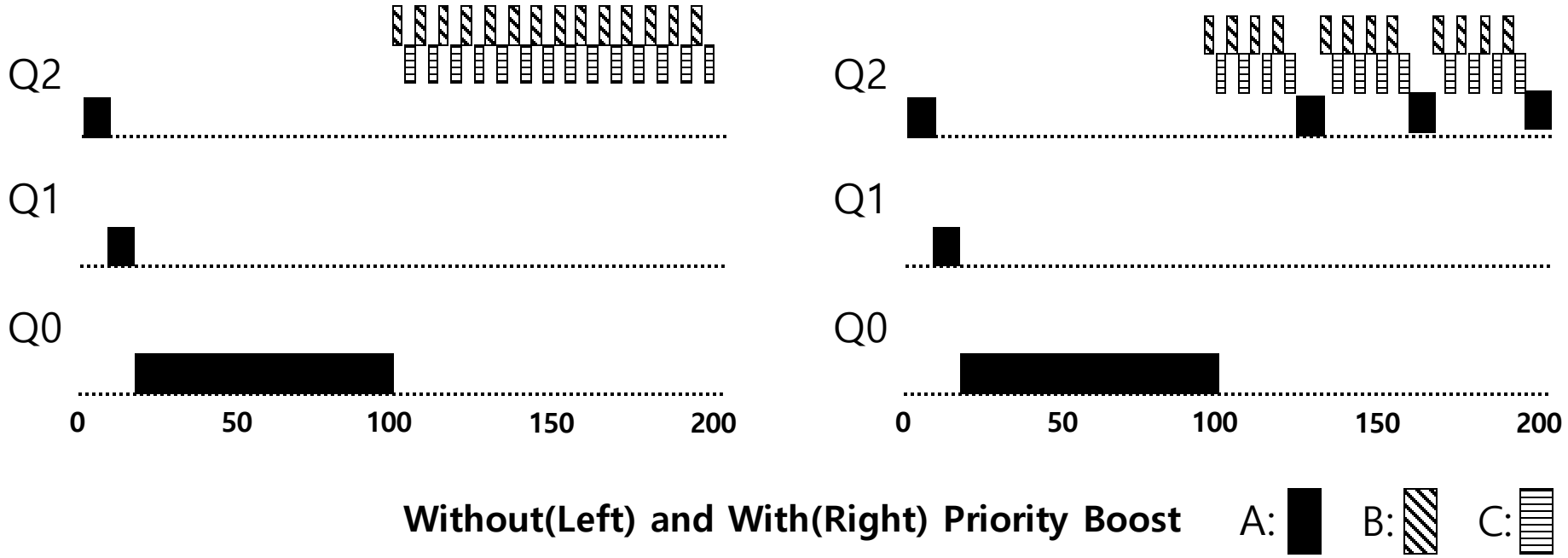
- Starvation
  - If there are “too many” interactive jobs in the system.
  - Non-running jobs will never receive any CPU time.

- Game (속이다) 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 (CPU 중심 작업) → I/O bound process (I/O 중심 작업)

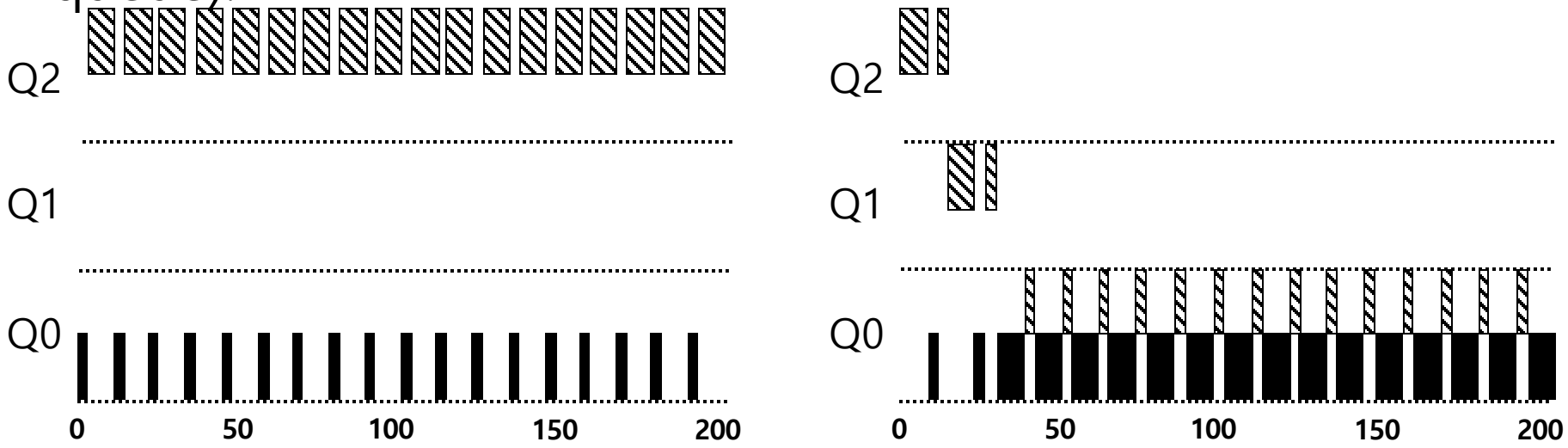
The Priority Boost

- **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)



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 (할당시간) 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).

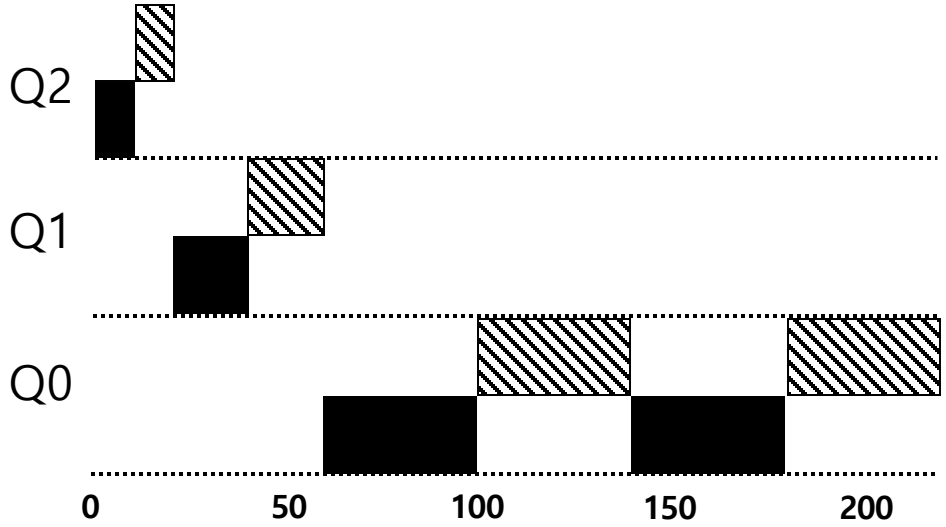


Without(Left) and With(Right) Gaming Tolerance

Tuning MLFQ and Other Issues

Lower Priority, Longer Quanta (Quantum의 복수)

- 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

The Solaris MLFQ implementation

- For the Time-Sharing scheduling class (TS)
  - 60 Queues
  - Slowly increasing time-slice length
    - The highest priority: 20ms
    - The lowest priority: A few hundred milliseconds
  - Priorities boosted around every 1 second or so.