Operating System:
Multi-level Feedback Queue

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

발전된 스케줄링 알고리즘 설명.

Towards a General CPU Scheduler

- Goals
  - Optimize turnaround time
    - ➢ SJF, STCF: No *a priori* knowledge on the workloads (Assumption 5)
  - Minimize response time for interactive jobs
    - ➢ RR: Bad turnaround time

- Challenge: No *a priori* knowledge on the workloads
  - The run time of each job is known (Assumption 5) — 이건 슬라이드에선 가정을 해서 설명한것.

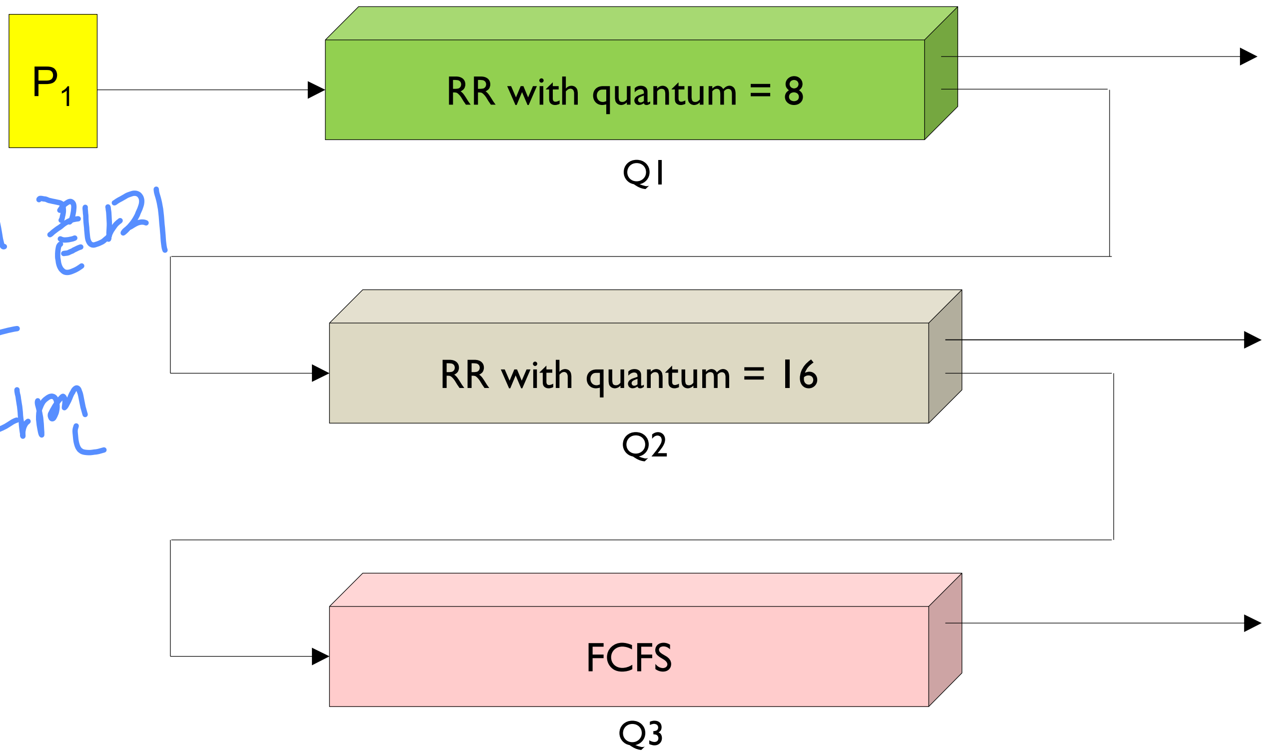
workload가 얼마나 CPU를 사용할지에 대한 정보가 없다는 게 문제

- How can the scheduler learn the characteristics of the jobs and make better decisions?
  - Learn from the past to predict the future (as in branch predictors or cache algorithms)

이전 결과에서 예측.

Multilevel Feedback Queue

- A process can move between the various queues

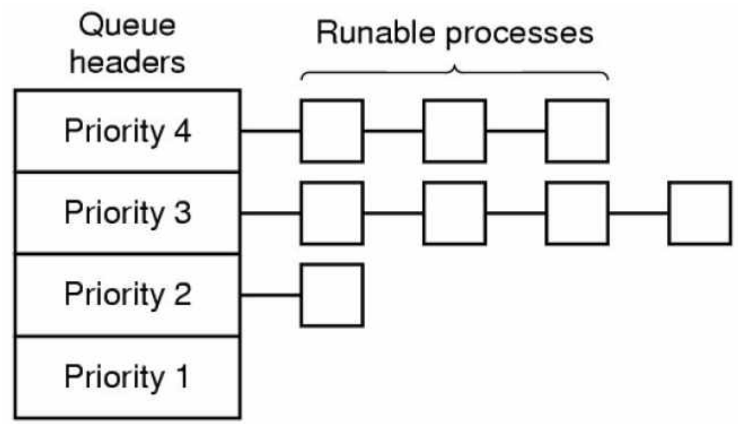


Q1 에서 실행이 끝나지
않으면 Q2로
그래도 안끝나면
Q3로 이동.

MLFQ: Basic Rules

큐별로 우선순위가 다름.

- MLFQ has a number of distinct queues
  - Each queues 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 Priority(A) > Priority(B), A runs (B doesn't)
Rule 2: If Priority(A) = Priority(B), A & B run in RR

MLFQ: Basic Rules (Cont.)

- MLFQ varies the priority of a job based on its observed behavior

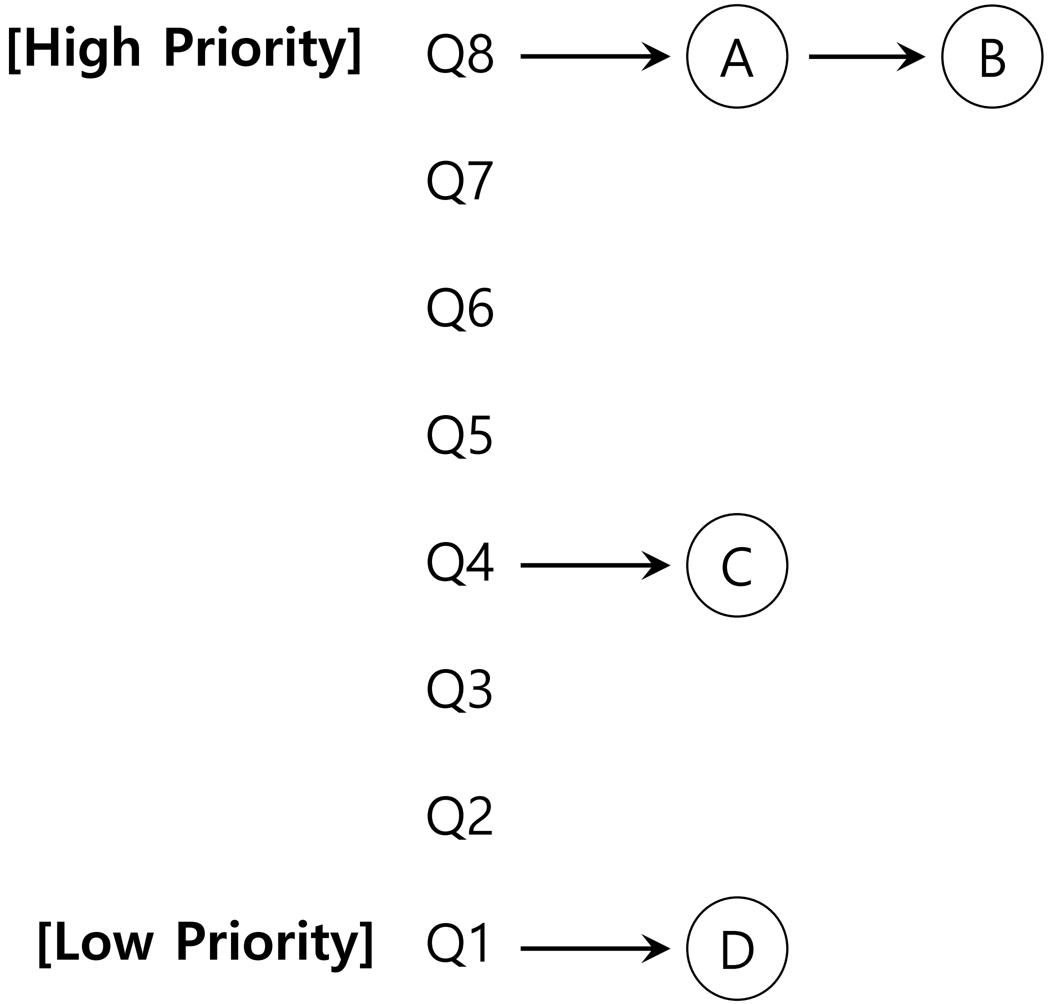
- Typical workload: a mix of
  - Interactive jobs (I/O-intensive jobs)
    - ➢ short-running, require fast response time
    - ➢ A job repeatedly relinquishes the CPU while waiting IOs
    - ➢ Keep its priority high

I/O 관련 job은 CPU를 별로 사용하지 않고 응답시간이 중요하니까 우선순위↑

  - CPU-intensive jobs
    - ➢ A job uses the CPU intensively for long periods of time
    - ➢ don't care about response time
    - ➢ Reduce its priority

CPU를 오래 쓰는 job은 응답시간 보단 더 길게 CPU를 쓰는게 중요 우선순위↓

MLFQ Example



MLFQ: How to Change Priority

CPU를 많이 쓰는지 I/O를 많이 하는지를 처음엔 알 수가 X.

- MLFQ priority adjustment algorithm:
  - **Rule 3:** When a job enters the system, it is placed at the highest priority 따라서 일단 우선순위는 높게 잡고.
  - **Rule 4a:** If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down on queue) time slice를 다 쓰면 우선순위를 낮춤.
  - **Rule 4b:** If a job gives up the CPU before the time slice is up, it stays at the same priority level

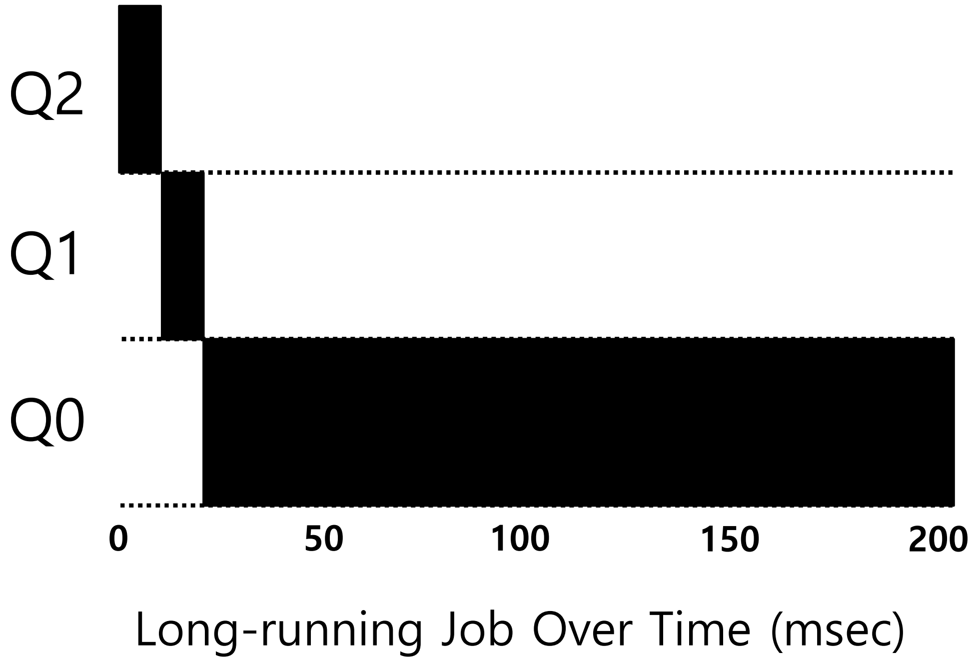
time slice가 끝나기 전에 CPU를 반납하면 우선순위가 유지되는 효과

In this manner, MLFQ approximates SJF

구조적으로 러닝타임이 짧은 I/O 작업은 높은 우선순위로 실행되기 좋고 긴 러닝타임은 주어진 time slicing을 초과하는 실행시간으로 우선순위가 낮아짐.

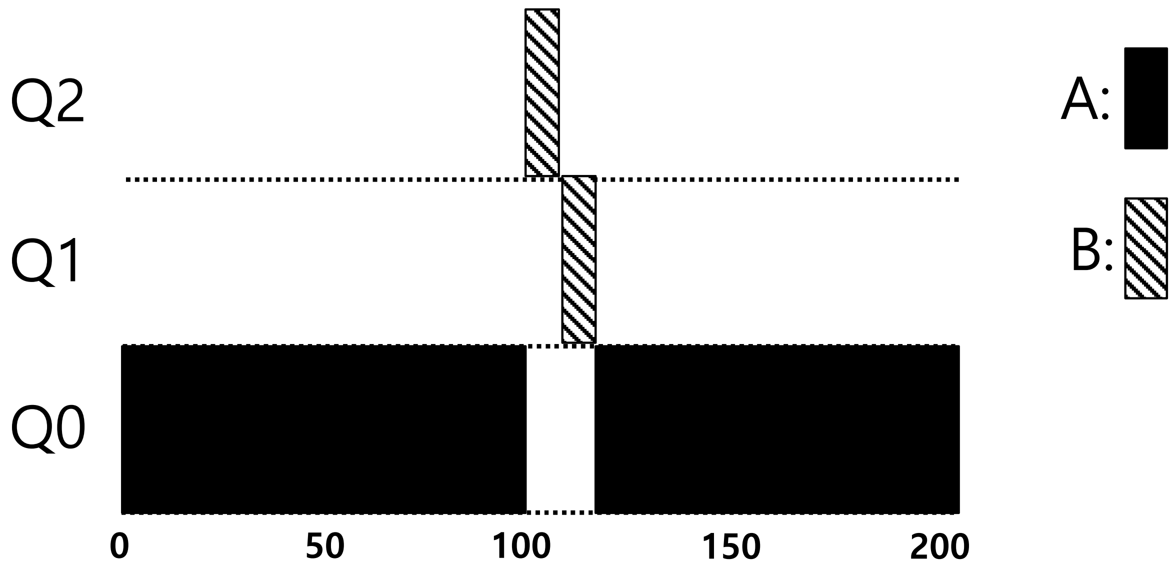
Example 1: A Single Long-Running Job

- 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

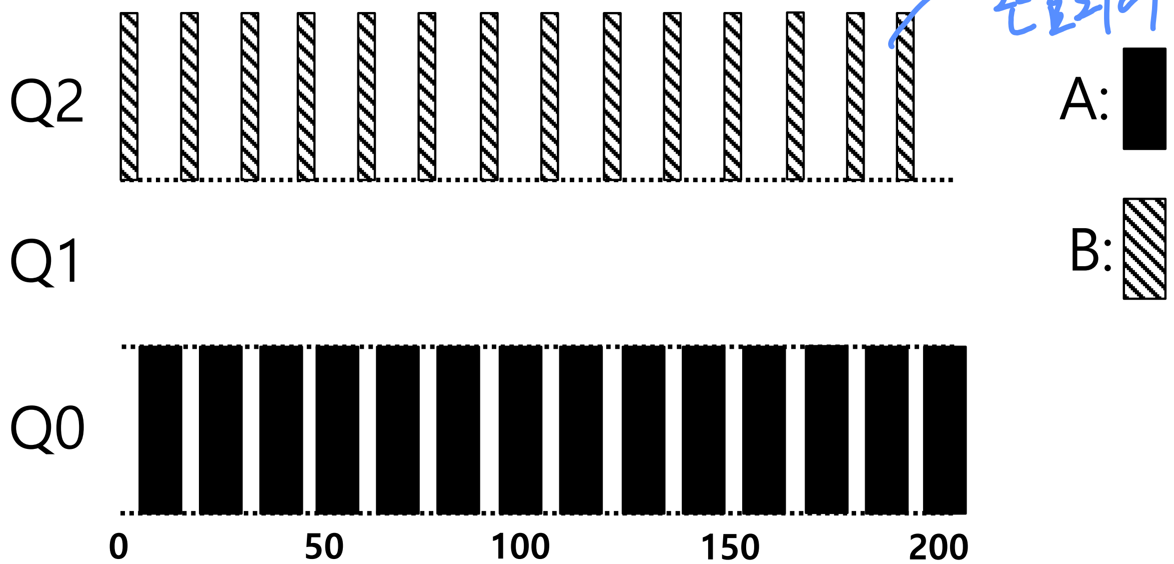


Along Came An Interactive Job (msec)

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

time slice가 끝나기전에 완료되어 자발적 반납을 진행하므로 우선순위유지.



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 우선순위가 높은 → 이런 일들이 너무 많으면 CPU-intensive job이
  - Long-running jobs will never receive any CPU time CPU를 못받음.
- Game the scheduler
  - After running 99% of a time slice, issue an I/O operation time slice 끝나기전에 입출력요청해서 우선순위를 의도적으로 유지.
  - The job gain a higher percentage of CPU time

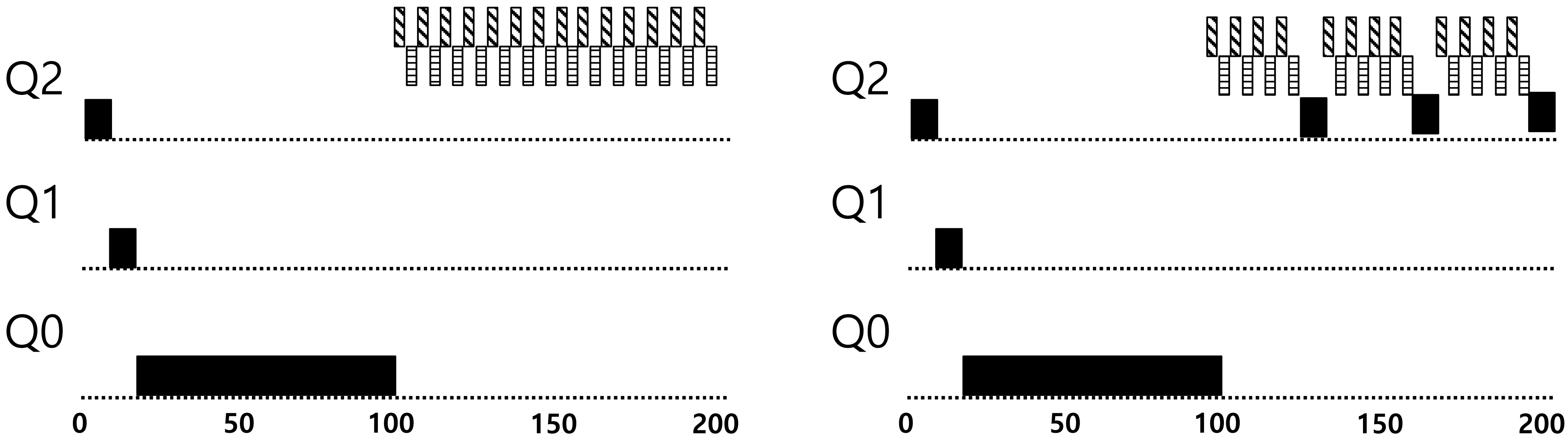
녹음 듣기.

- A program may change its behavior over time
  - CPU bound process → I/O bound process

The Priority Boost

특정시간 S가 지나면 모든 job의 priority를 상승시킴

- 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)



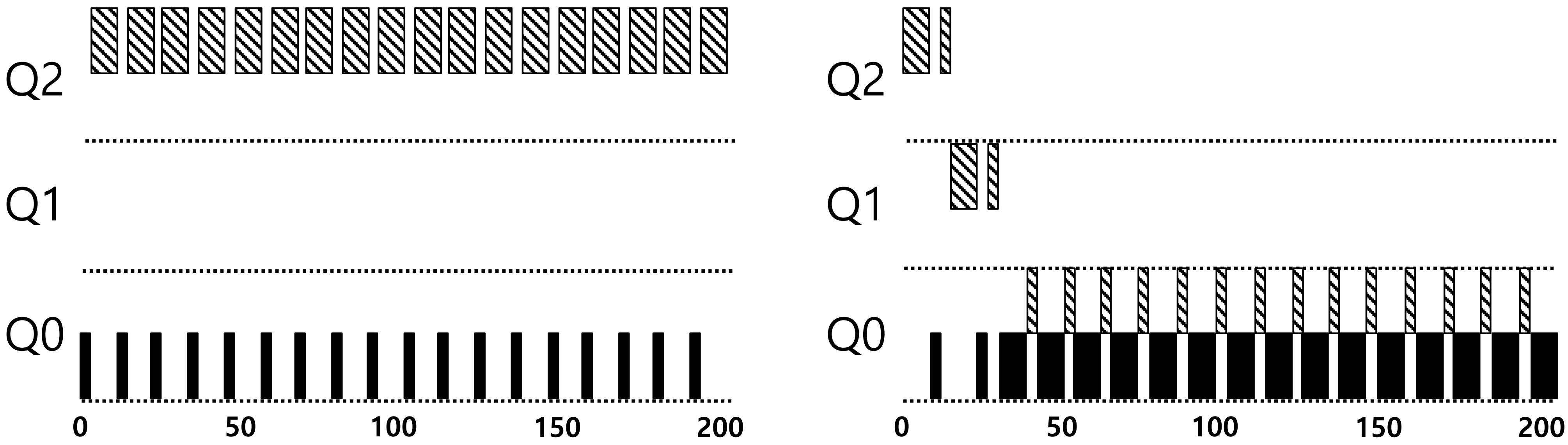
Without(Left) and With(Right) Priority Boost A: B: C:

Better Accounting

- How to prevent gaming of our scheduler?
- Solution:

time slice와 다른개념으로 추가적으로 할당받은시간 이상을 사용하면 우선순위하락

  – **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)

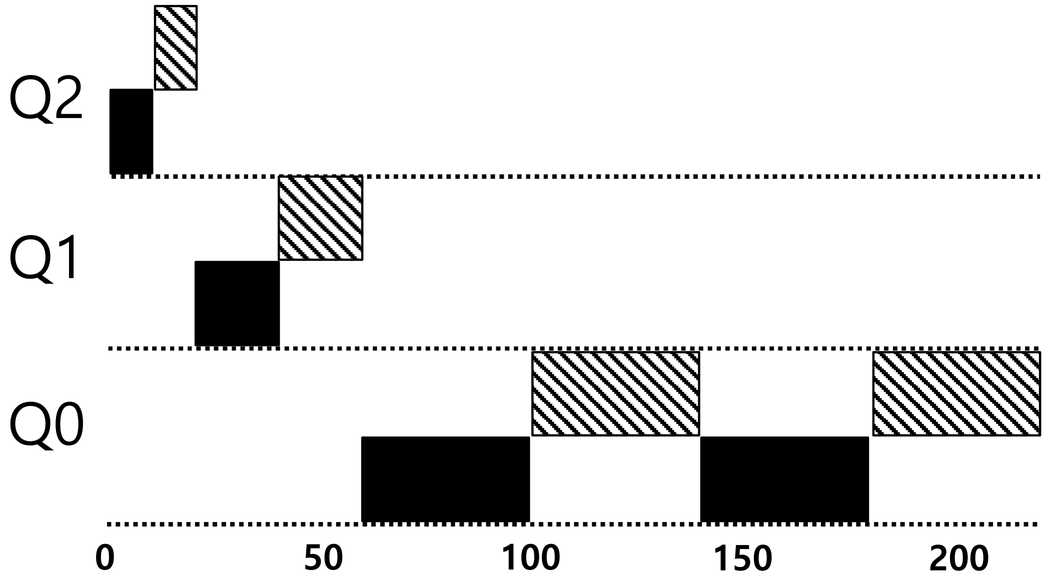


Without(Left) and With(Right) Gaming Tolerance

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

The Solaris MLFQ implementation 예시.

- 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

MLFQ: Summary

- The refined set of MLFQ rules:
  - **Rule 1:** If Priority(A) > Priority(B), A runs (B doesn't)
  - **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
  - **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