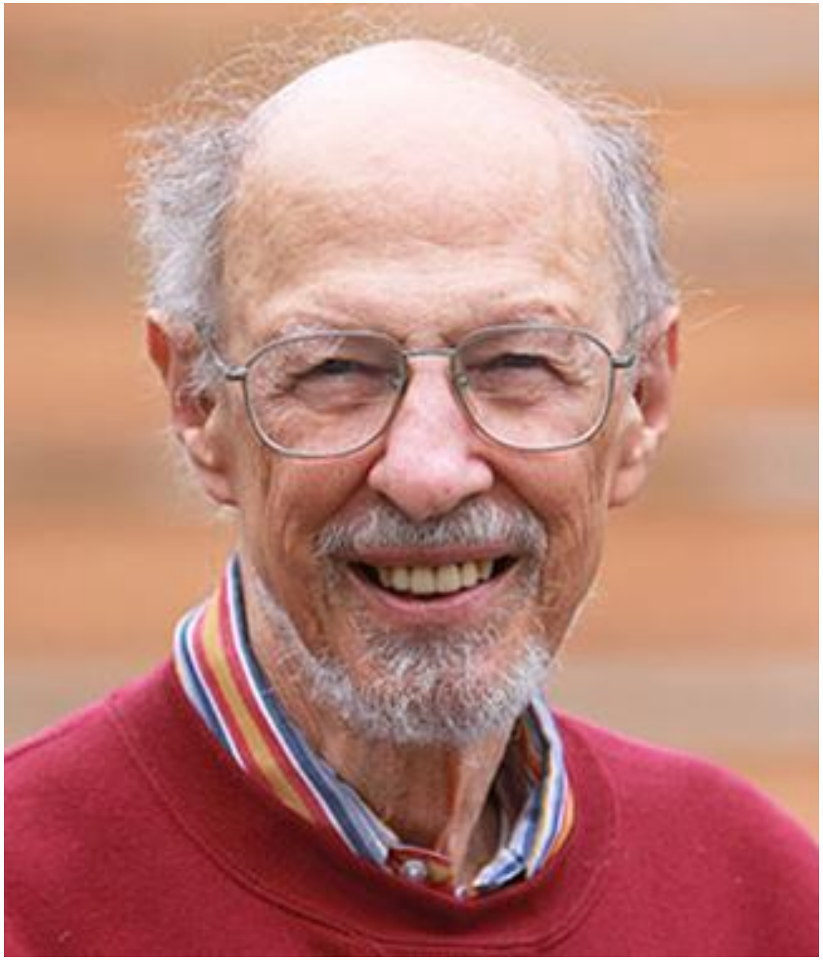
COS40003 Concurrent Programming

Lecture 3 (b): scheduling II

Multi-Level Feedback Queue

- **Fernando José Corbató**
  - **Turing Award Winner 1990**
  - "for his pioneering work in organizing the concepts and leading the development of the general-purpose, large-scale, time-sharing and resource-sharing computer systems"



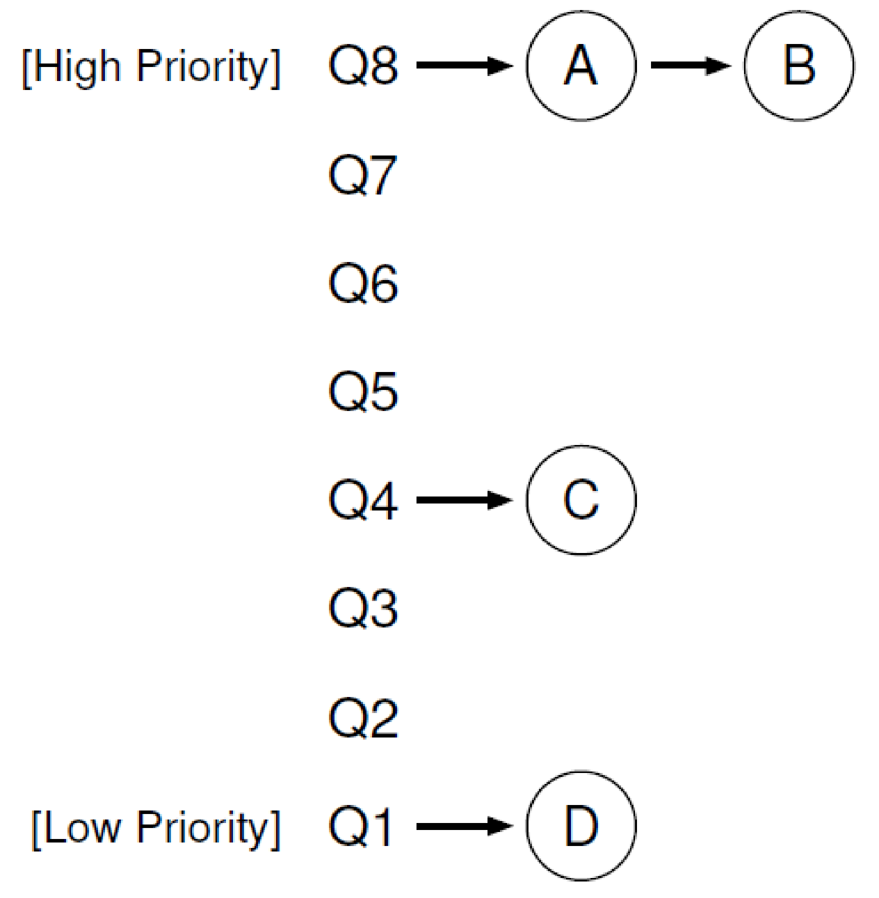
Multi-Level Feedback Queue

- Aim (achieve both)

- Optimize turnaround time
  - Dilemma: knowing SJF/PSJF is good, but how to know how long a job will run;

- Minimize response time
  - Dilemma: RR is good in response time, but is terrible for turnaround time

MLFQ Basics



- Tasks put in **different queues**, with **different priority** level.

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

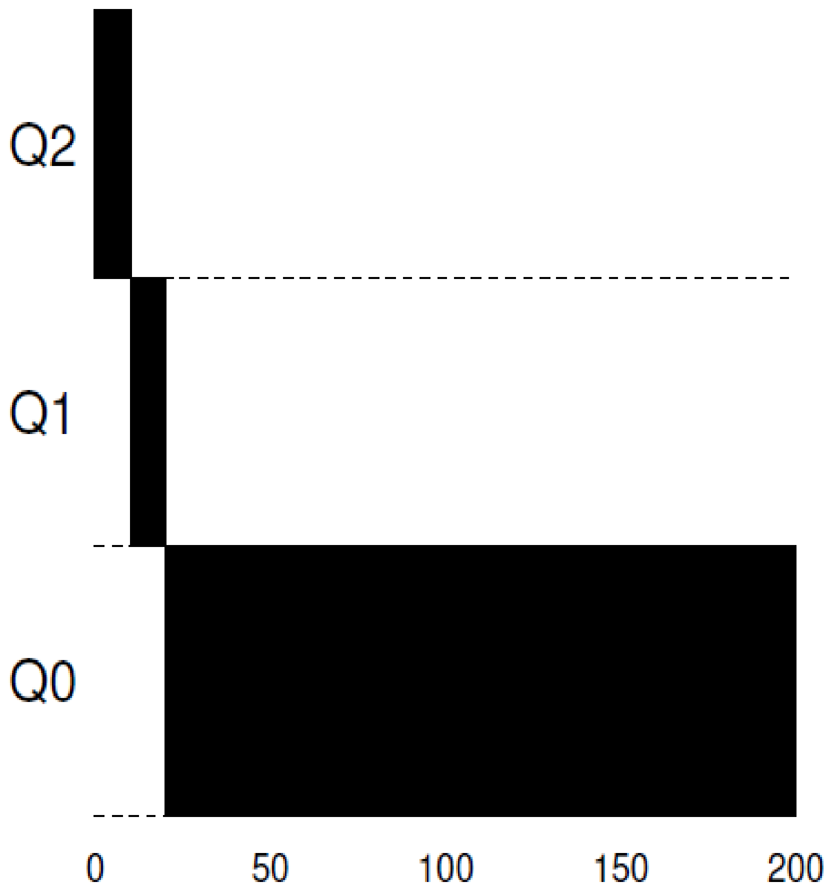
MLFQ: How To Change Priority?

- Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue).

- Rule 4a: If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down to the next (lower) queue).

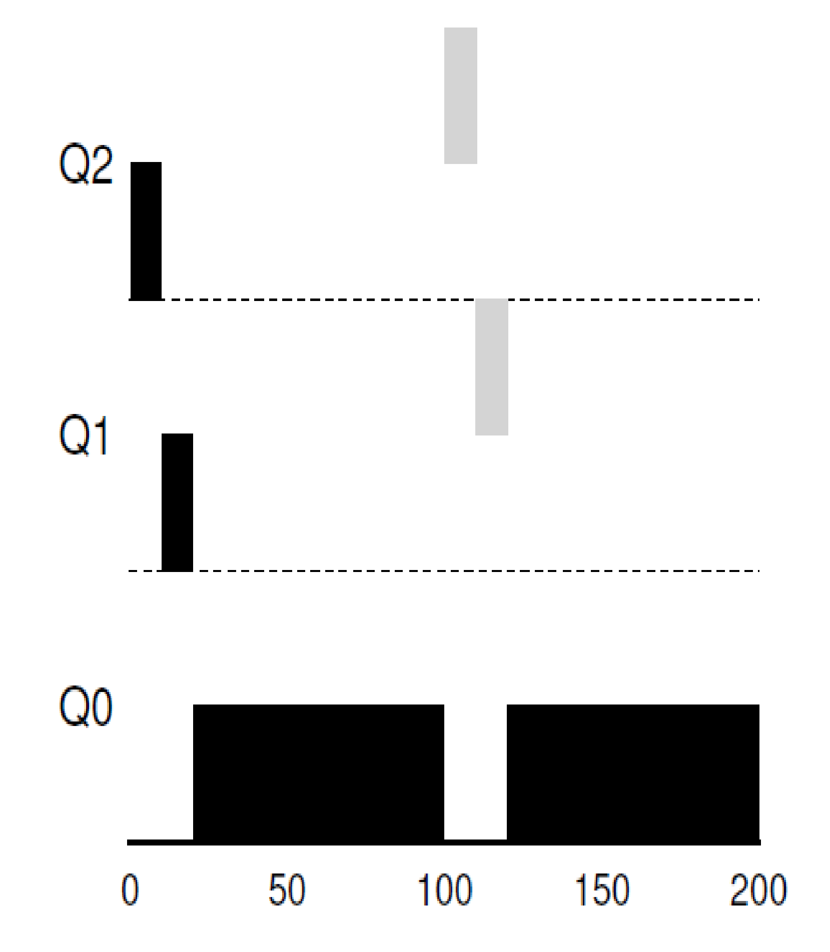
- Rule 4b: If a job gives up the CPU before the time slice is up, it stays at the same priority level.

Example 1: A Single Long-Running Job



- The job enters at the highest priority (Q2) (Rule 3)
- 10ms later, the job moves to Q1 (Rule 4a)
- 10ms later, the job moves to Q0 (Rule 4a)

Example 2: Along came a short job



- B (which will take 20ms) arrives at time T = 100
- B is firstly inserted into the top queue (Rule 3)
- B completes before reaching the bottom queue, in two time slices (Rule 4a)

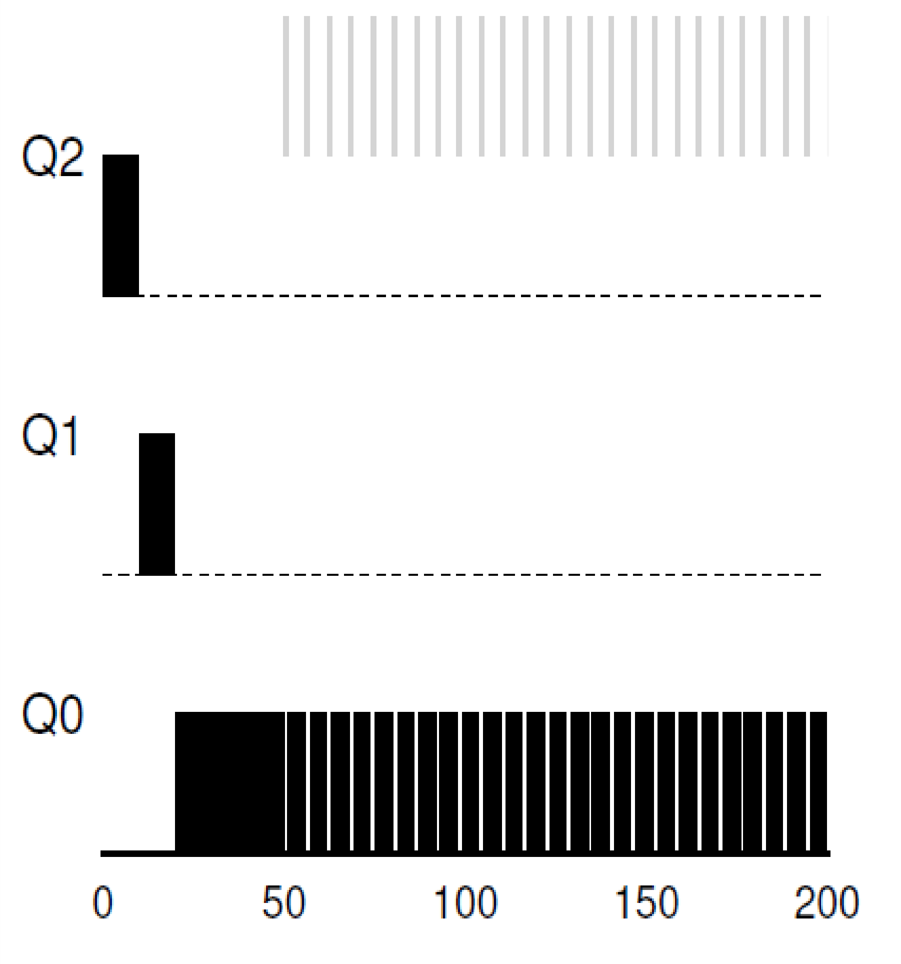
How MLFQ approximates SJF?

- A job is firstly assumed to be a short job, thus giving the job high priority.

- If it actually is a short job, it will run quickly and complete;

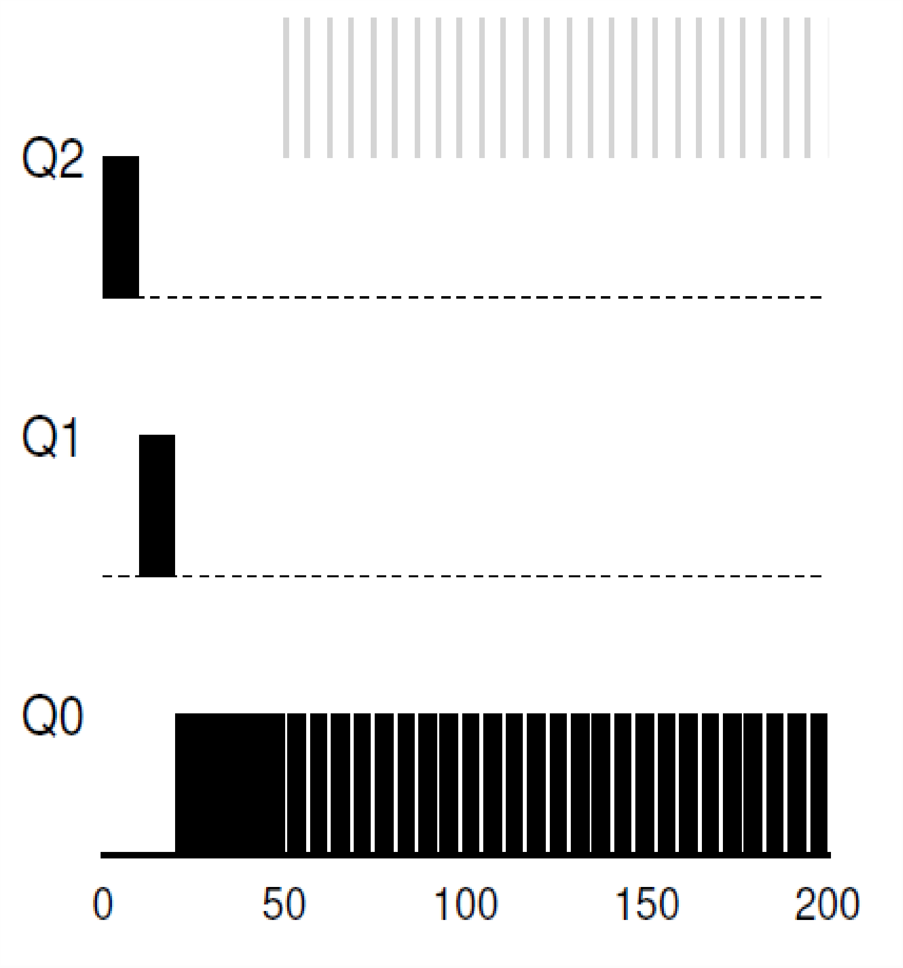
- If it is not a short job, it will slowly move down the queues, and thus soon prove itself to be a long-running job.

Example 3: What About I/O?



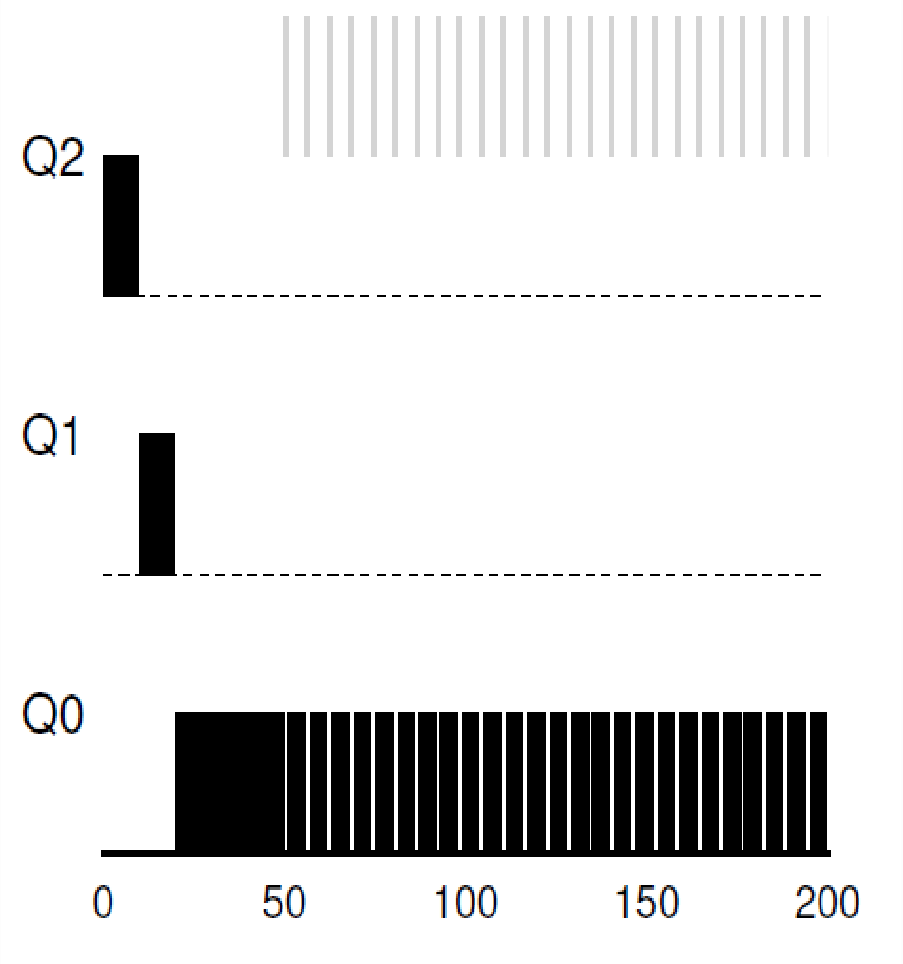
- B (an interactive job), for example, is doing a lot of I/O (say by waiting for user input from the keyboard or mouse)

Example 3: What About I/O?



- B (an interactive job) that needs the CPU only for 1 ms before performing an I/O competing for the CPU with a long-running batch job A

Example 3: What About I/O?



- Do not want to penalize such Task B.
- Rule 4b: if a process gives up the processor before using up its time slice, we keep it at the same priority level.

Problems with our current MLFQ?

- It seems to do a fairly good job:
  - sharing the CPU fairly between long-running jobs
  - letting short or I/O-intensive interactive jobs run quickly. (better turnaround time)
  - Letting all jobs starting with high priority (better response time)

- **Any problem?**

Problems with our current MLFQ?

- Starvation:
  - if there are “too many” interactive jobs in the system, they will combine to consume all CPU time, and thus long-running jobs will never receive any CPU time.
- Game the scheduler
  - before the time slice is over, issue an I/O operation (to some file you don’t care about) and thus give up the CPU; doing so allows you to remain in the same queue, and thus gain a higher percentage of CPU time.

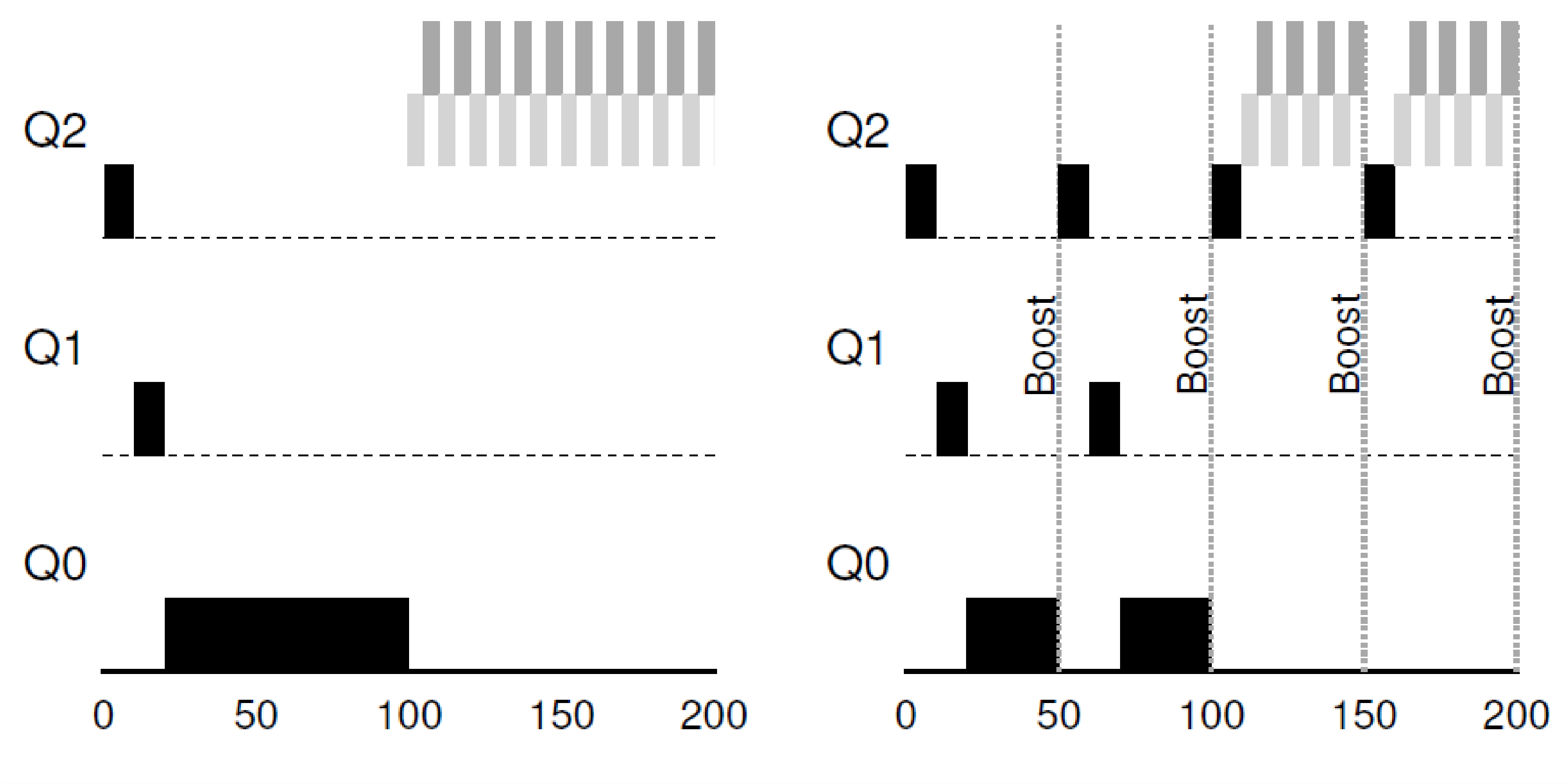
MLFQ: How to better manage priority: Priority Boost

- To resolve starvation:

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

MLFQ: How to better manage priority: Priority Boost

- a long-running job competing for the CPU with two short-running interactive jobs



MLFQ: How to better manage priority: better accounting

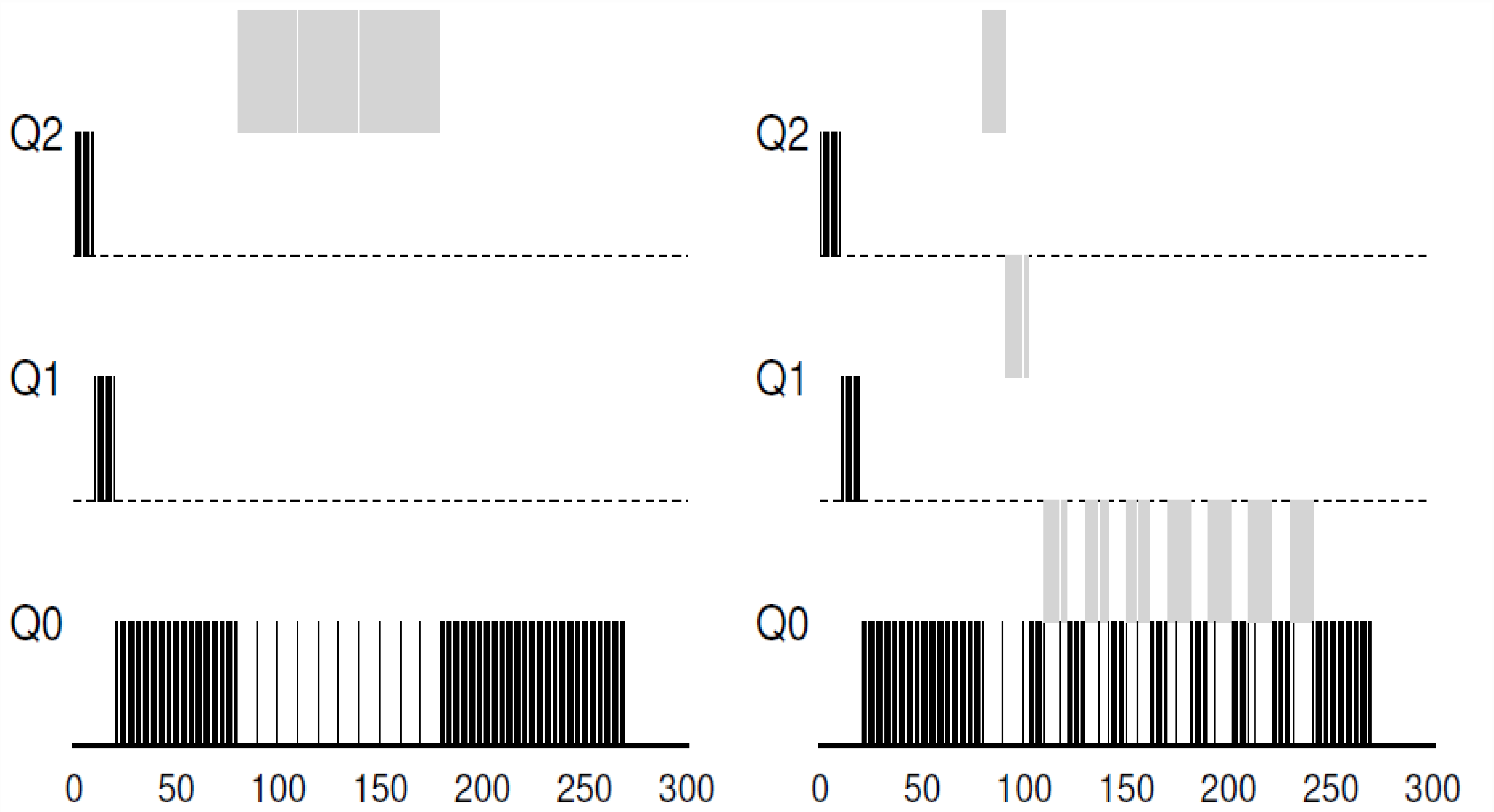
- To resolve gaming the scheduler:

- Better accounting: it does not matter CPU occupation is in one long burst or many small ones.

- **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).

(To replace Rule 4(a) and Rule 4(b))

Old Rules 4a and 4b VS the new anti-gaming Rule 4



Multi-Level Feedback Queue

Summary :

- **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 (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.

Tuning MLFQ and Other Issues

- One big question: how to **parameterize** such a MLFQ?
  - How many queues should there be?
  - How big should the time slice be per queue?
  - How often should priority be boosted?

- There are no easy answers to these questions. ☹

Tuning MLFQ and Other Issues

- One widely accepted rule:
  - most MLFQ variants allow for varying time-slice length across different queues
  - The high-priority queues are usually given short time slices;
  - The low-priority queues, in contrast, given longer time slices.

Why MLfQ is good?

- Instead of demanding a priori knowledge of a job, it observes the execution of a job and prioritizes it accordingly
  - Has excellent overall performance (similar to SJF/PSJF) for short-running interactive jobs
  - Fair to long-running CPU-intensive workloads.

- Applied in operating systems: BSD Unix, Solaris, Windows Series

Acknowledgement

- Chapter 7-9
  - Operating Systems: Three Easy Pieces

- 8.ppt
  - Intro to Operating System at Portland State University
  - by Jonathan Walpole

Questions?