Recap

Basic Scheduling Principles

- FIFO: non-preemptive
- SJF (Shortest Job First): non-preemptive
- STCF (Shortest Time-To-Completion First): preemptive
- RR (Round Robin): preemptive

Open question: scheduler does not know how long it will take for a job to complete!

L5: Multi-Level Feedback Queue

(Based on Chapter 8)

Slides revised from
Matthew Tancreti
Iowa State University

Multi-Level Feedback Queue

SJF and STCF have good features, but

they require **oracle** vision and

they have serious flaws for some workloads

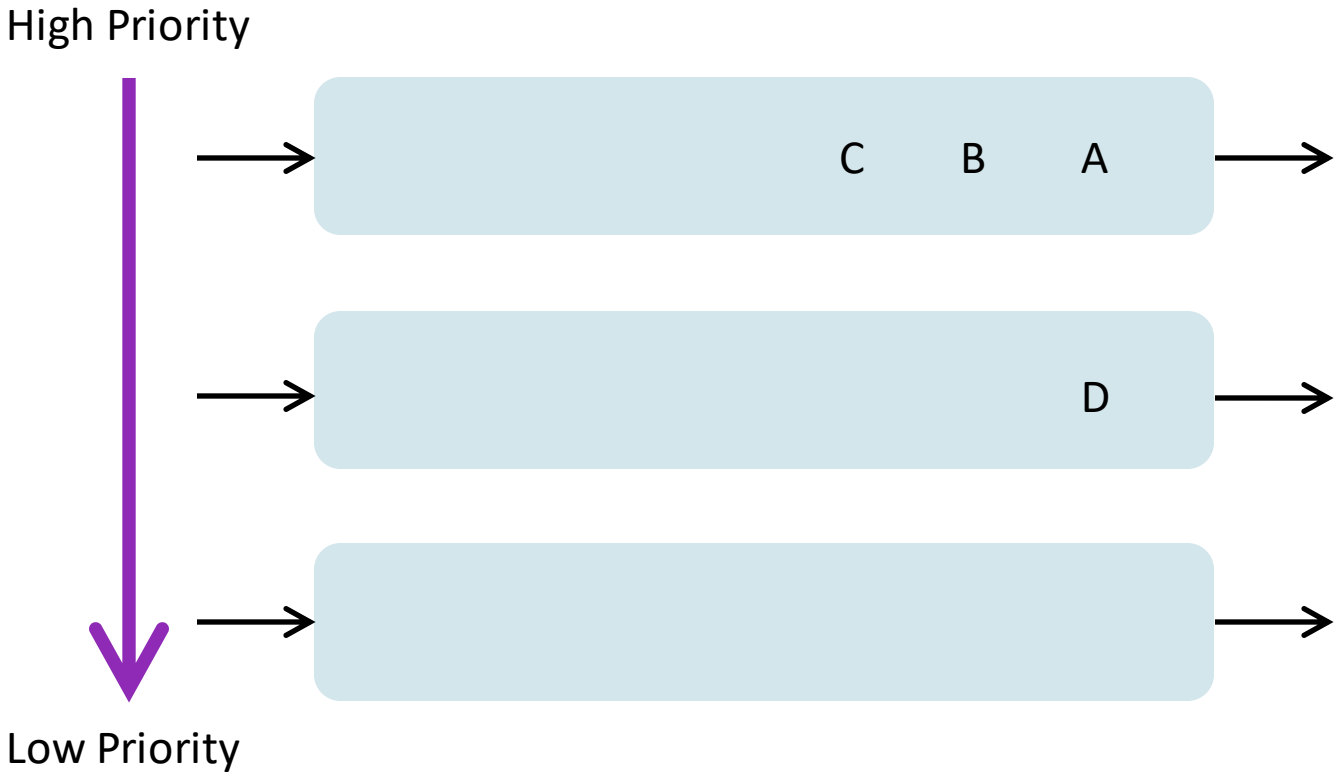
RR has low response time, but it treats all jobs equally which can result in poor turnaround time and frequent context switches

How can we combine the ideas from SJF, STCF and RR into a single scheduler?

Multi-Level Queue

Rule 1: If Priority(A) > Priority(B), A runs (B doesn't)

Rule 2: If Priority(A) = Priority(B), A and B run in RR order



Examples of Common Processes

Process 1: waits for user to press key, performs short task such as adding character to display buffer and then waits for next key press

Process 2: needs several minutes of CPU time to render video

Which process is more important?

User wants video to finish rendering quickly,

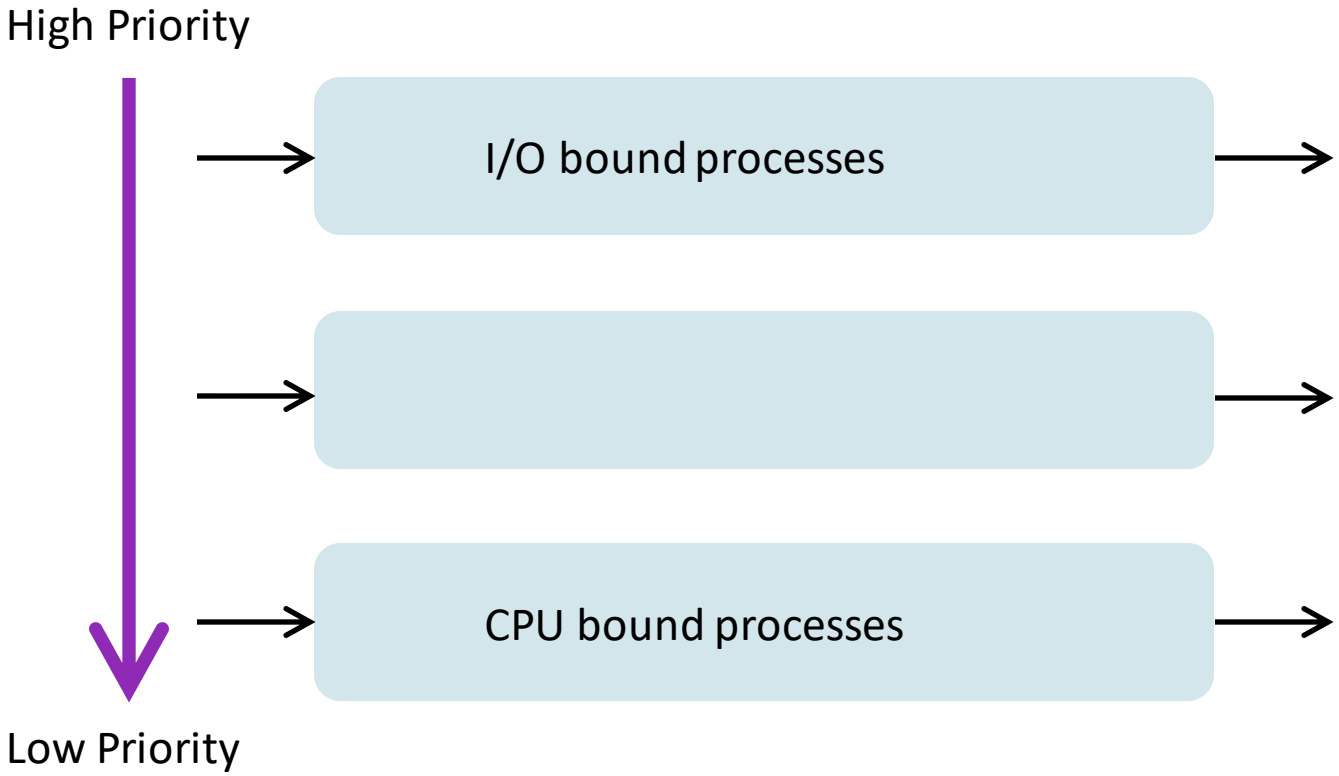
but also expects “instant” response to keyboard presses

Which processes should get highest priority?

CPU bound vs I/O bound processes

Processes that need fast response but little CPU time should be highest priority

Processes that need long CPU time and little I/O should be lowest priority



Problems

But we can't just ask a process: "are you CPU or I/O bound?"

Also, the behavior of processes change over time, for example,

Time 1: process is taking input from user and disk (I/O bound)

Time 2: process is performing long computation (CPU bound)

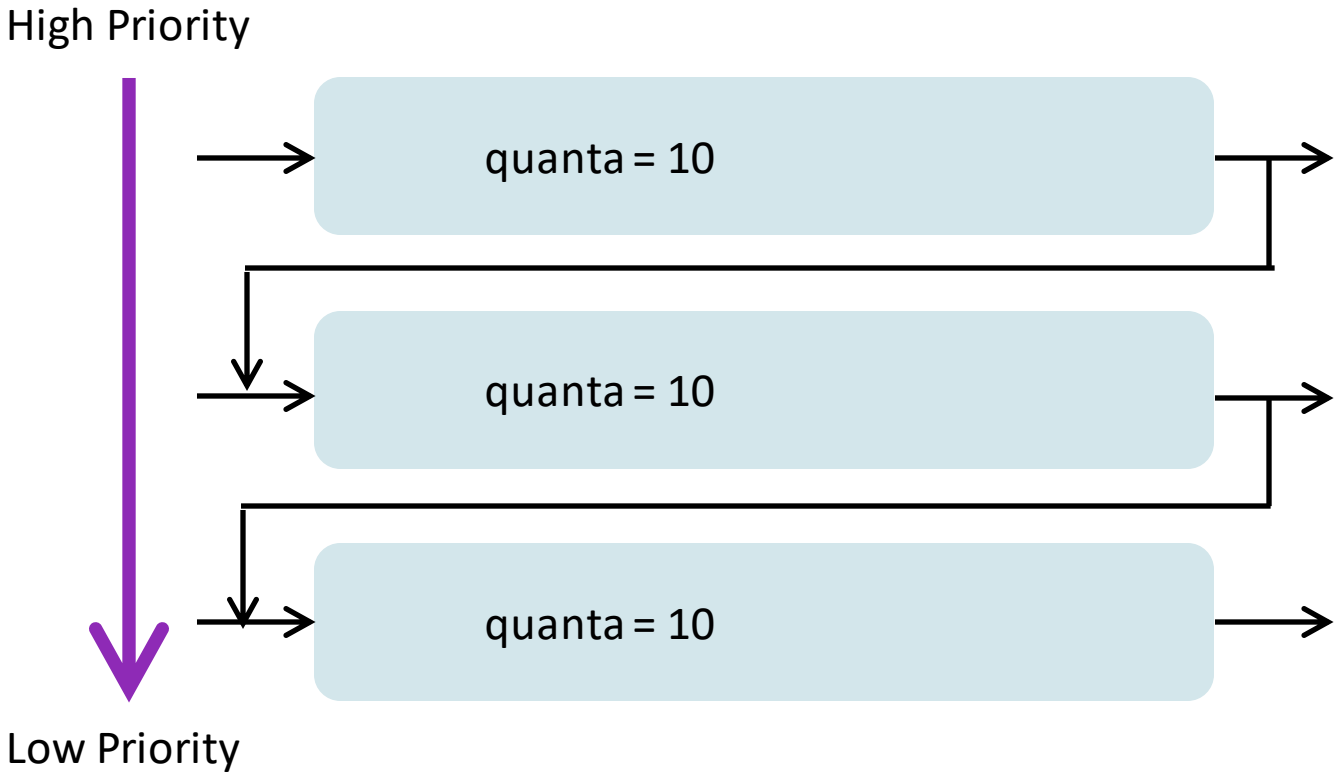
Can we devise policies that account for different types of processes?

Adding Feedback to Multi-Level Queue

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

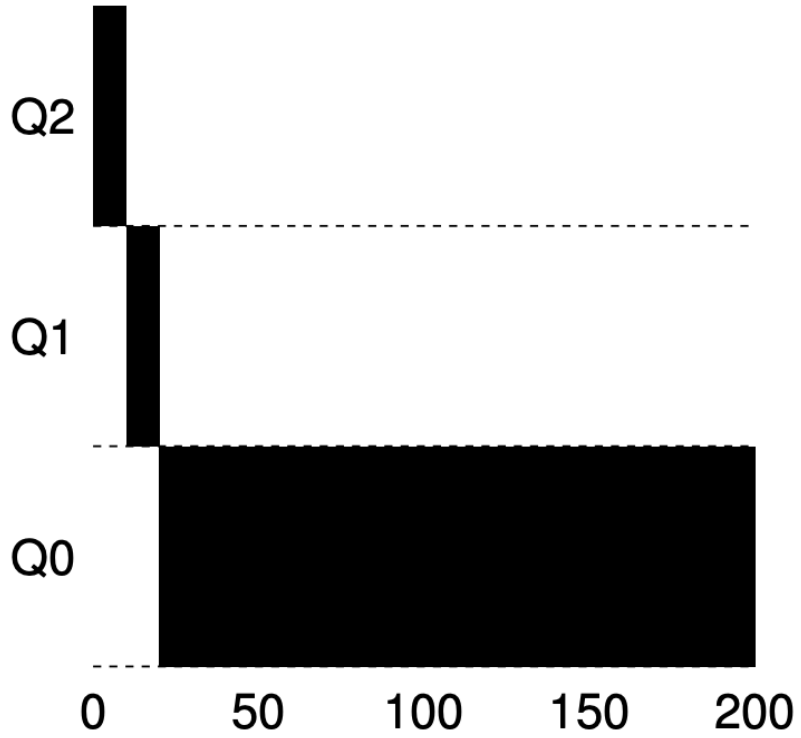
Rule 4a: If a process uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue).

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



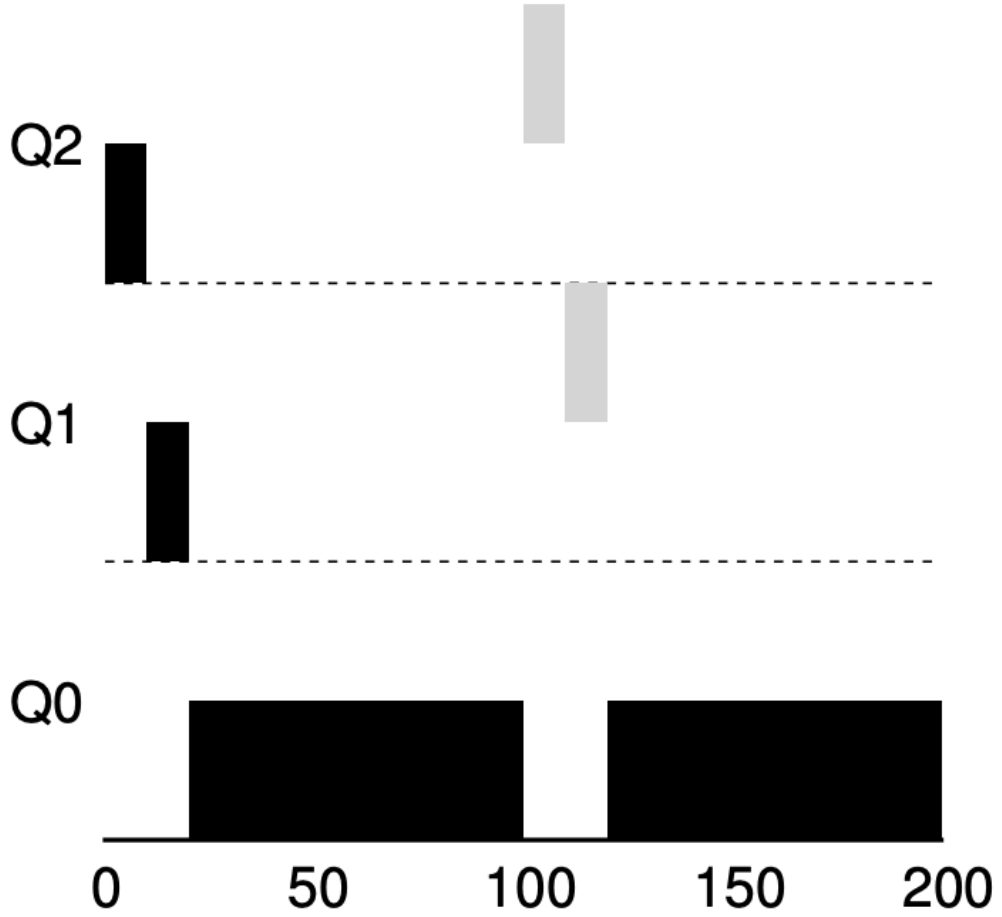
Example of MLFQ with Single Job

Example: A long job arrives at time 0



Example of MLFQ with Two Jobs

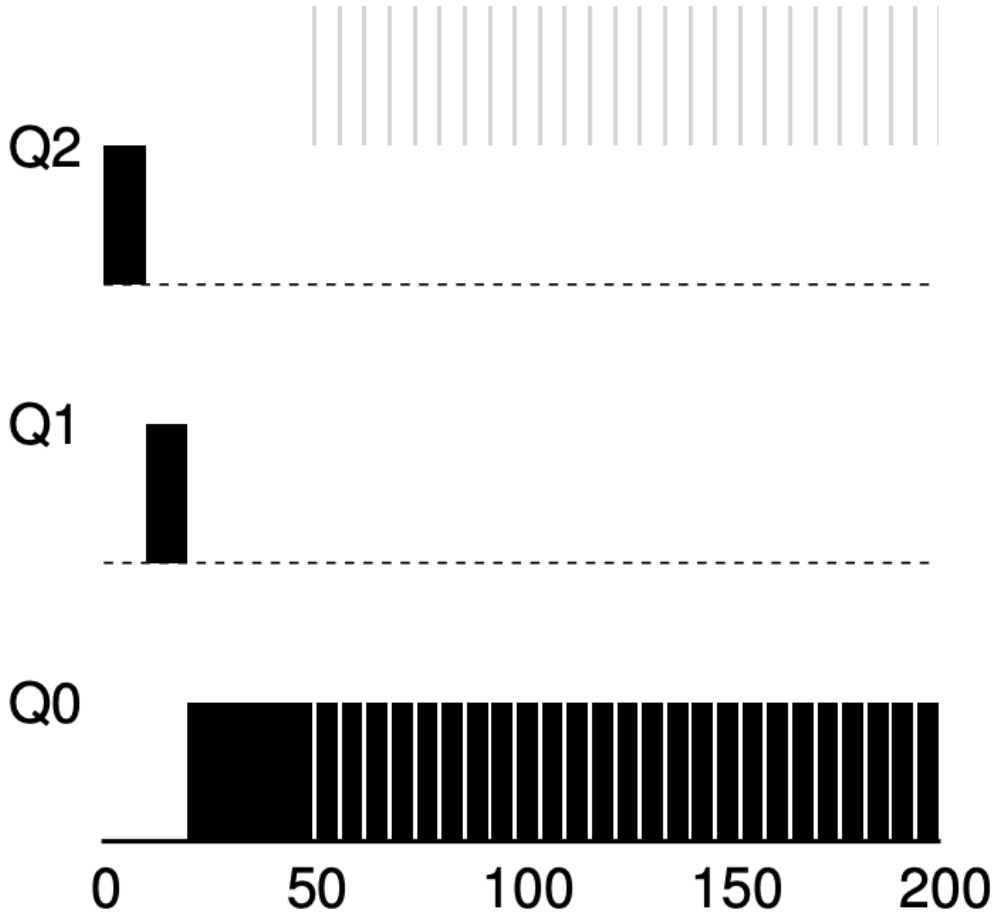
Example: A medium job arrives at time 100, it starts in Q2 giving it priority



Example of MLFQ with Frequent I/O Bound Processes

Example: Several short jobs arrive and are completed quickly while long job keeps making progress

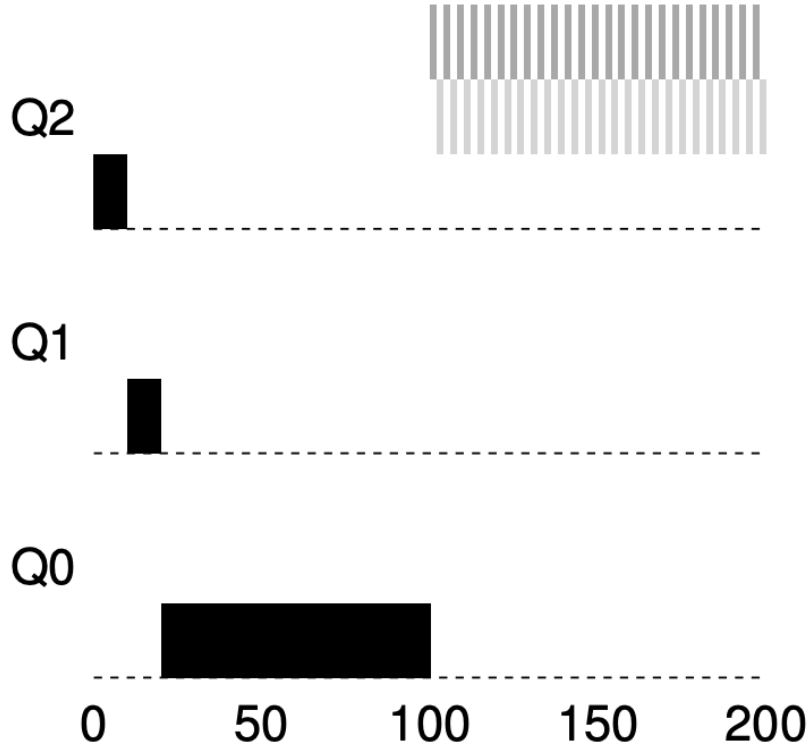
We are getting the same result as STCF, but no oracle required!



Problem: Starvation

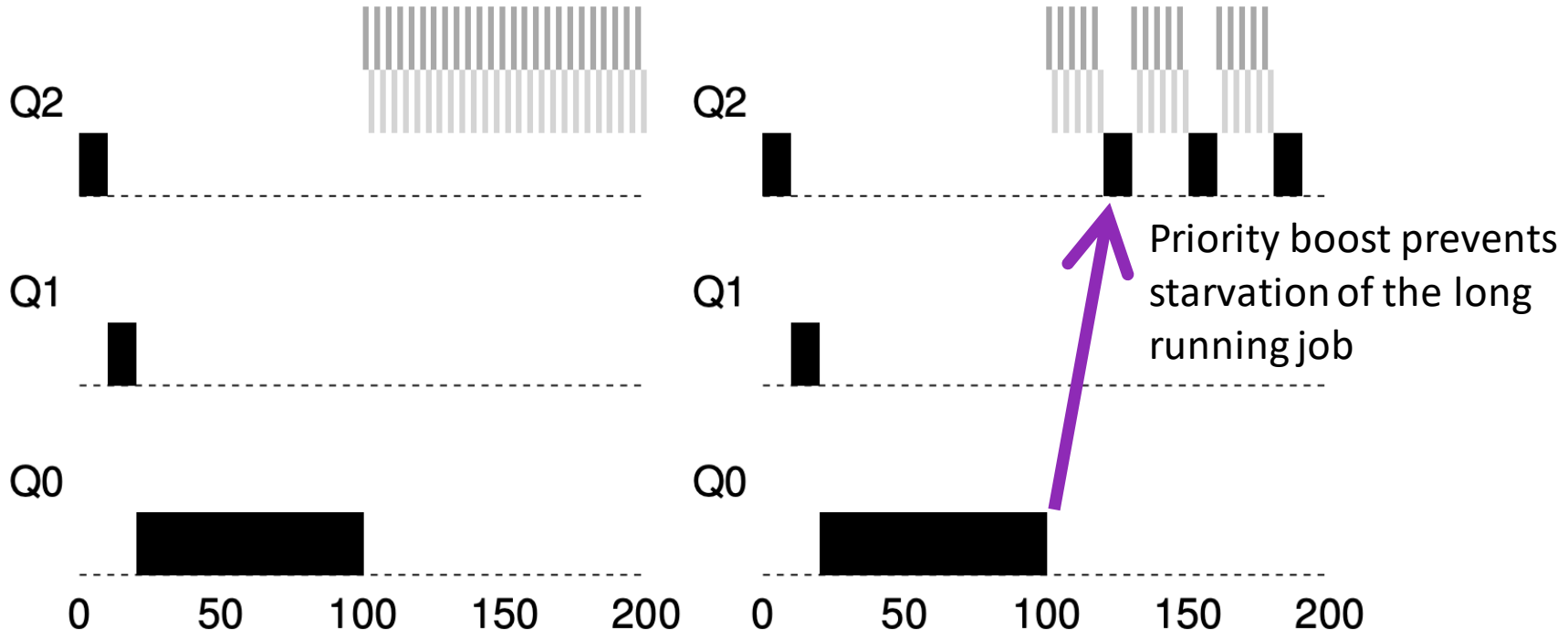
What happens if we turn up the rate of I/O bound jobs?

Now the long job can't make progress, it faces **starvation**



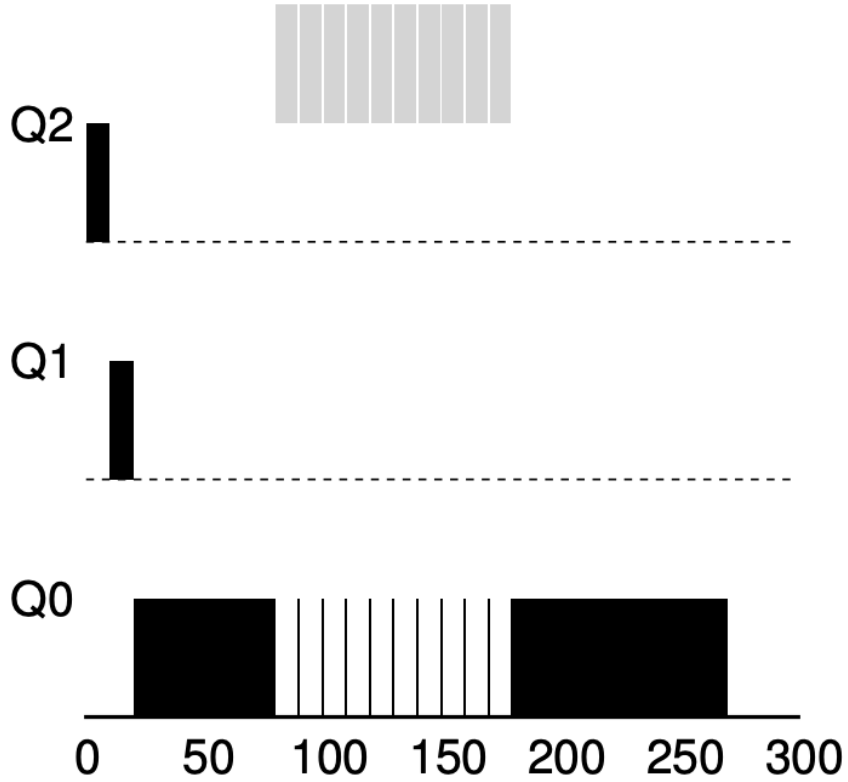
Priority Boost

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



Problem: Gaming the System

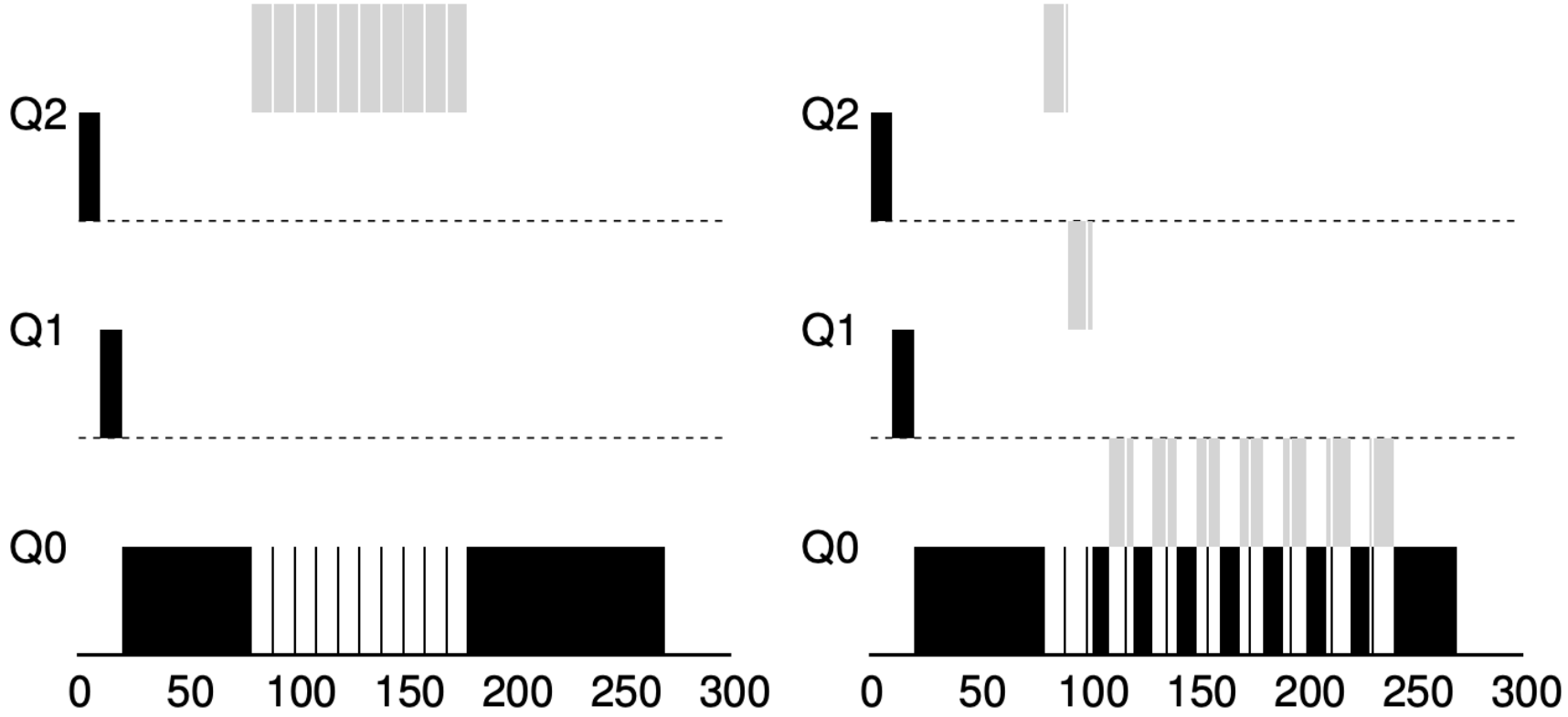
Smart programmers will catch on: “I can make my process run faster if I never let a complete quanta expire”



Rule 4a: If a process uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue).
Rule 4b: If a process gives up the CPU before the time slice is up, it stays at the same priority level.

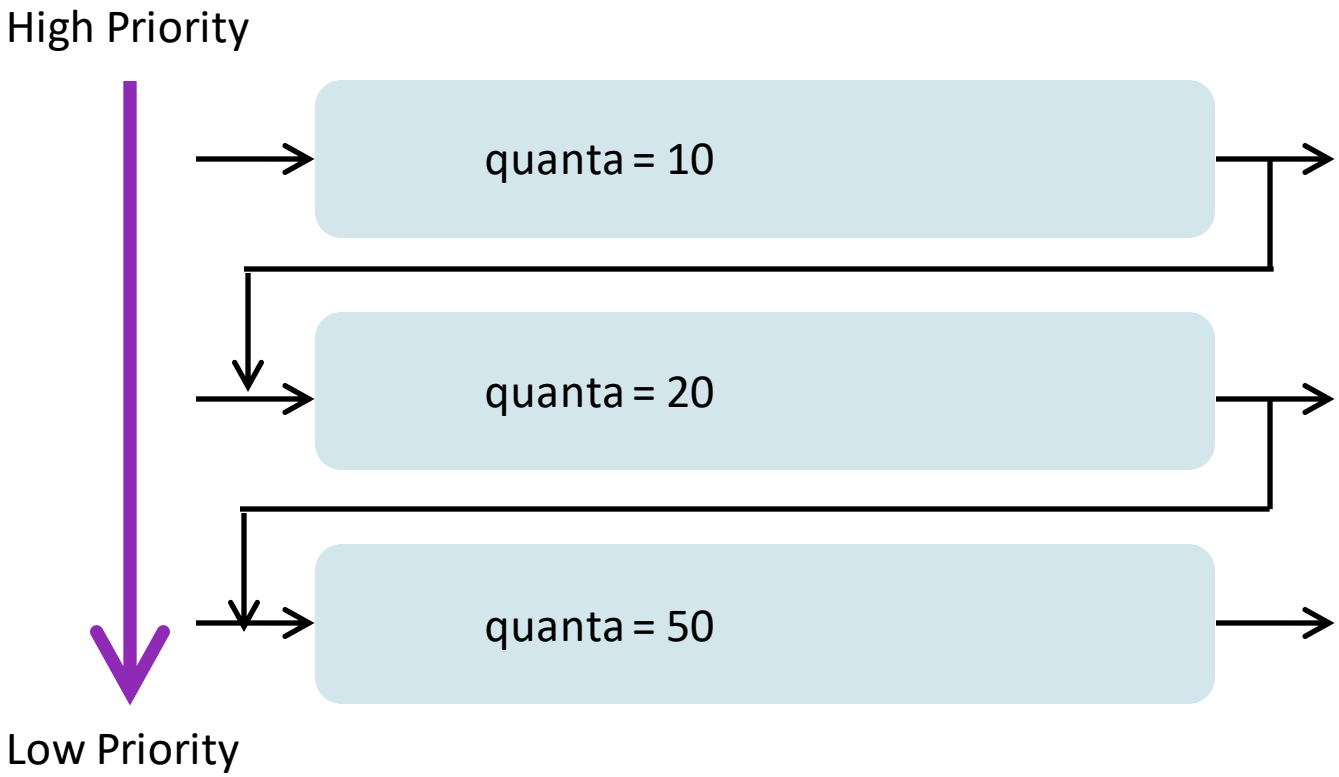
Problem: Gaming the System

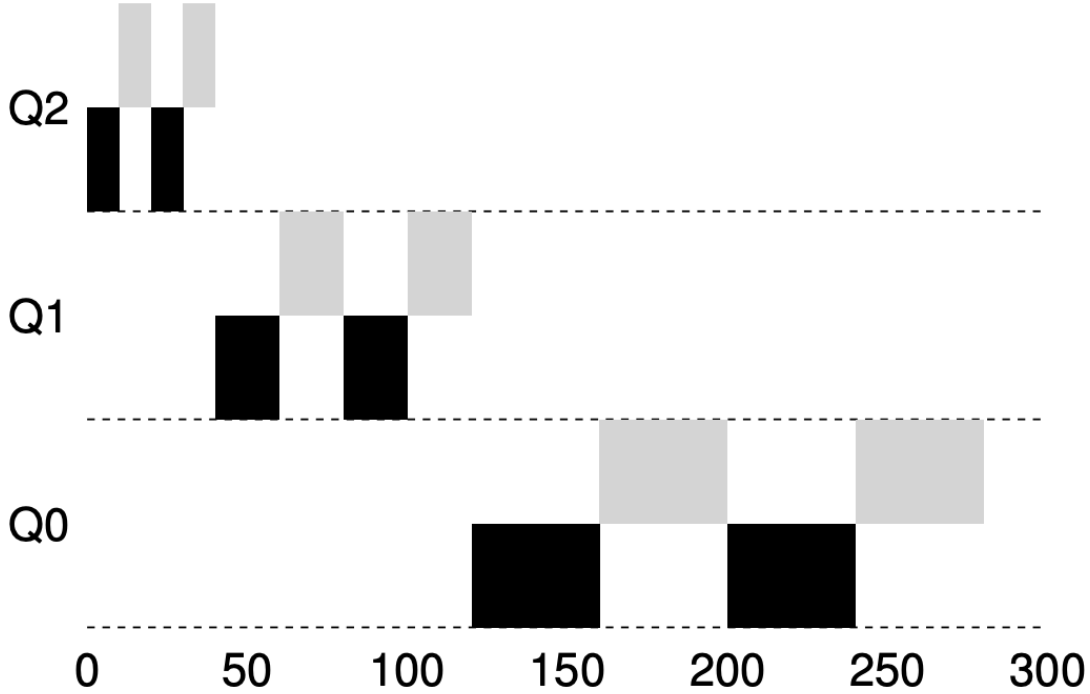
New **Rule 4**: Once a process uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).



Optimization: Lower Priority, Longer Quanta

Longer quanta in the lower queues can reduce frequency of context switches





The Complete MLFQ

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 process enters the system, it is placed at the highest priority (the topmost queue).

Rule 4: Once a process 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 processes in the system to the topmost queue.