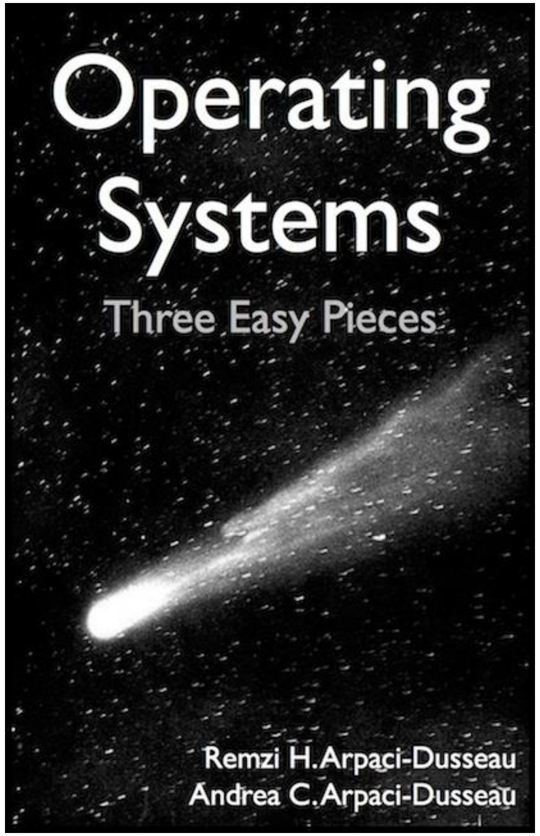
CMSC 125: Operating Systems

- Instructor: **Joseph Anthony C. Hermocilla**
- Email: jchermocilla@up.edu.ph
- Web: https://jachermocilla.org

Resources

Book: https://pages.cs.wisc.edu/~remzi/OSTEP/

Slides Template:
https://pages.cs.wisc.edu/~remzi/OSTEP/Educators-Slides/Youjip/



Acknowledgement

- This lecture slide set 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 at University of Wisconsin.

8: Scheduling:
The Multi-Level Feedback Queue

Operating System: Three Easy Pieces

Multi-Level Feedback Queue (MLFQ)

- First described in the **Compatible Time-Sharing System (CTSS)** by Corbato
- Objectives
  1. Optimize **turnaround time** by <u>running shorter processes first</u>
  2. Minimize **response time** <u>without a priori knowledge of a process' run-time</u>
- A scheduling policy that <u>learns from the past to predict the future</u>

MLFQ: Basic Rules

- MLFQ has a number of distinct **queues**
  - Queues are assigned different **priority** levels.

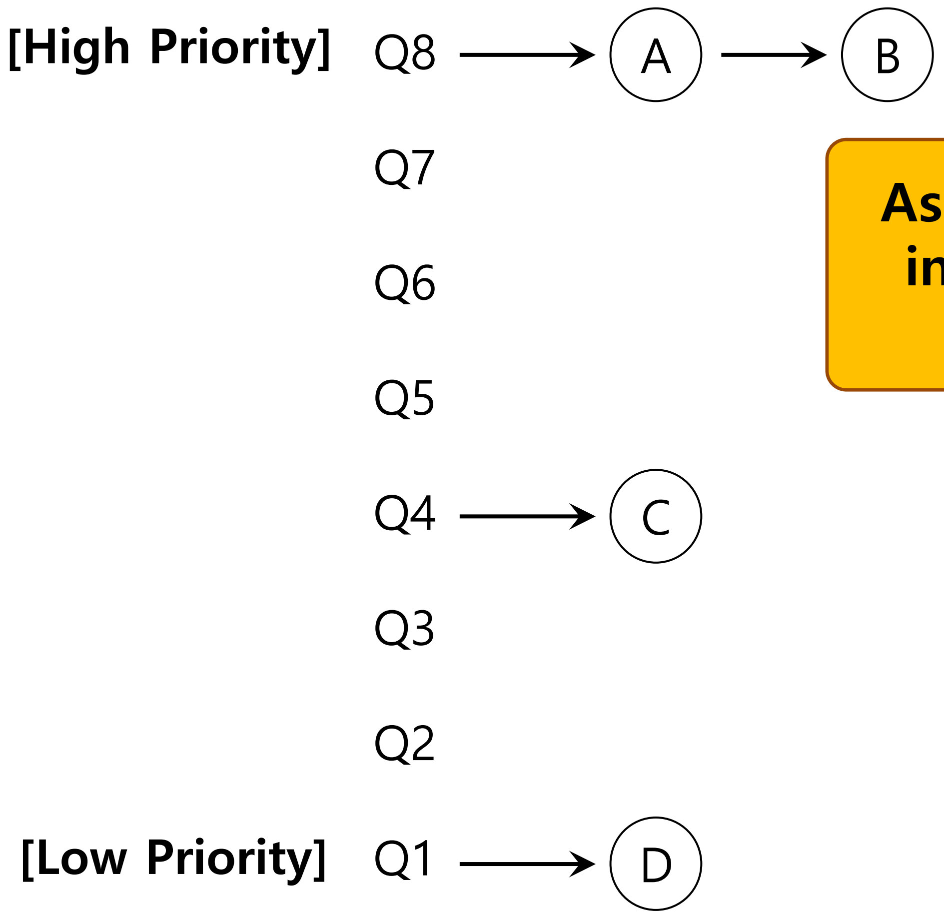
- A process that is ready to run is assigned to a single queue
  1. Scheduler chooses a process **on a higher priority queue** to run on the CPU
  2. Use round-robin scheduling among processes on the same queue (since they have the same priority level)

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: Basic Rules (Cont.)

- Key is how the scheduler sets the priority levels of the queues
- MLFQ varies the priority of a process based on its observed behavior (rather than on a fixed priority value)
- Example:
  - A process that repeatedly relinquishes the CPU while waiting IOs (an interactive process) → Keep its priority high
  - A process that uses the CPU intensively for long periods of time (a CPU-intensive process) → Reduce its priority

MLFQ Example



As long as there are processes in high-priority queues, they will be scheduled first.

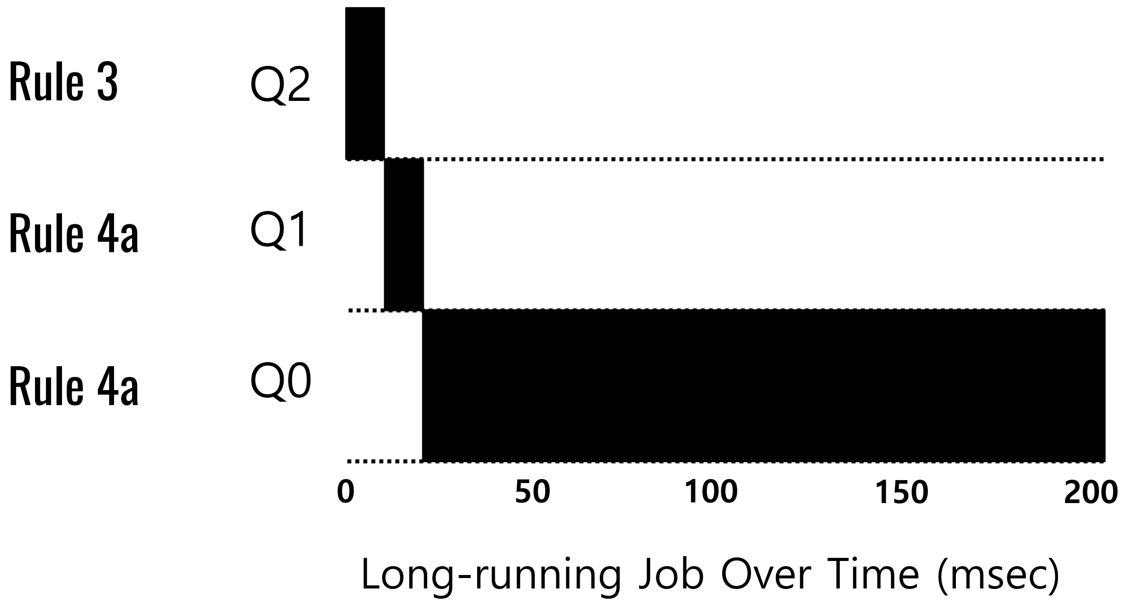
How to Change Priority: Approach #1

- **Rule 3**: When a process enters the system, it is placed at the highest priority queue
- **Rule 4a**: If a process uses up an entire time slice while running, its priority is reduced (i.e., it moves down on queue).
- **Rule 4b**: If a process 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 Process

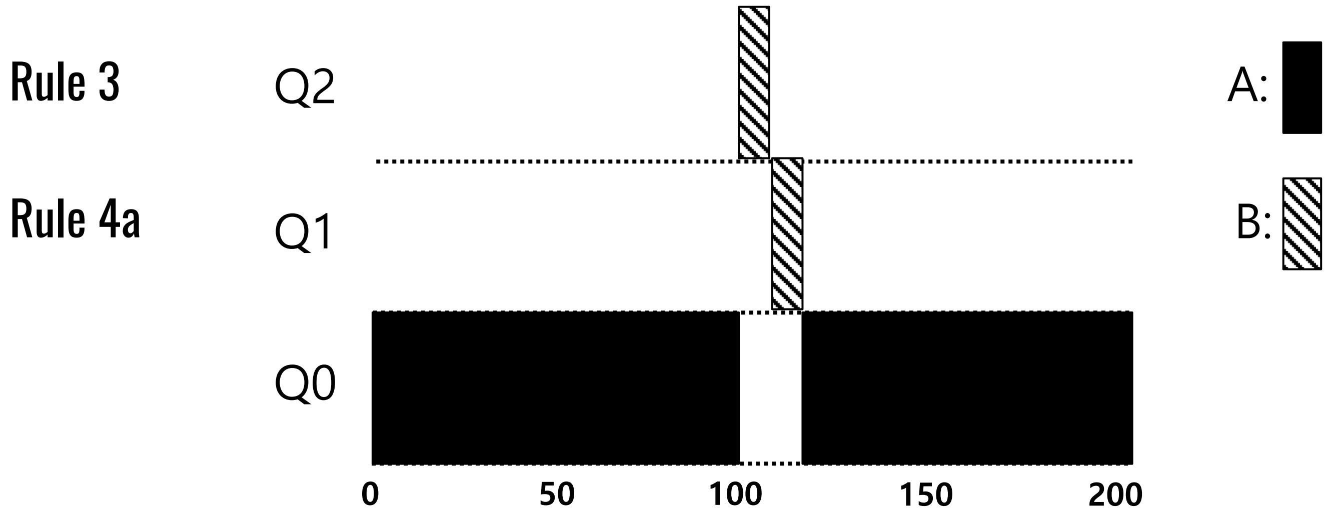
- A three-queue scheduler with time slice 10ms



Example 2: Along Came a Short Process

- Assumptions:
  - **Process A**: A long-running <u>CPU-intensive</u> process
  - **Process B**: A short-running <u>interactive</u> process (20ms runtime)
  - A has been running for some time, and then B arrives at time T=100.

Scheduler does not know the run-time of new process. Assumes it is short. If it is, then it will finish early. Thus approximates SJF.

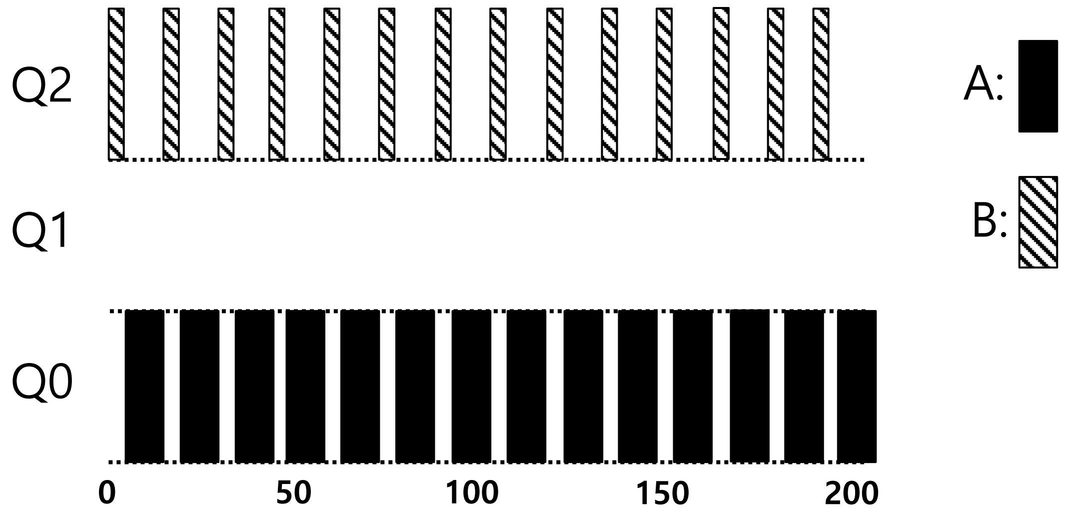


Along Came An Interactive Job (msec)

Example 3: What About I/O?

- Assumptions:
  - **Process A**: A long-running CPU-intensive process
  - **Process B**: An interactive process that needs the CPU only for 1ms before performing an I/O

Rule 4b



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

Don’t penalize a process (by reducing priority) if CPU is given up before time slice completes.
The MLFQ approach keeps an interactive job at the highest priority

Problems with the Basic MLFQ and Approach #1

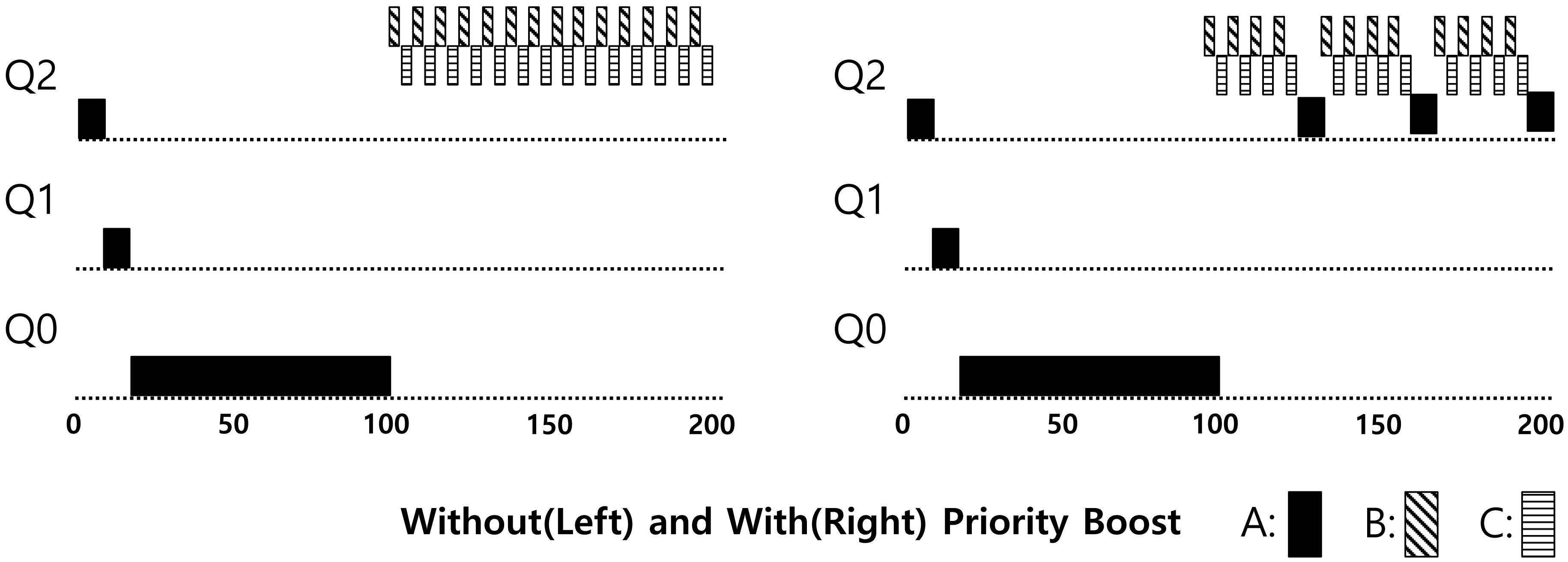
1. **Starvation**
   - If there are “too many” interactive processes in the system, long-running processes will never receive any CPU time

2. **A process may change its behavior over time**
   - CPU bound process becomes an I/O bound process

3. **Gaming the scheduler**
   - After running 99% of a time slice, issue an I/O operation – does not consume entire time slice thus remains in current queue
   - The process gains a higher percentage of CPU time

Approach #2: The Priority Boost

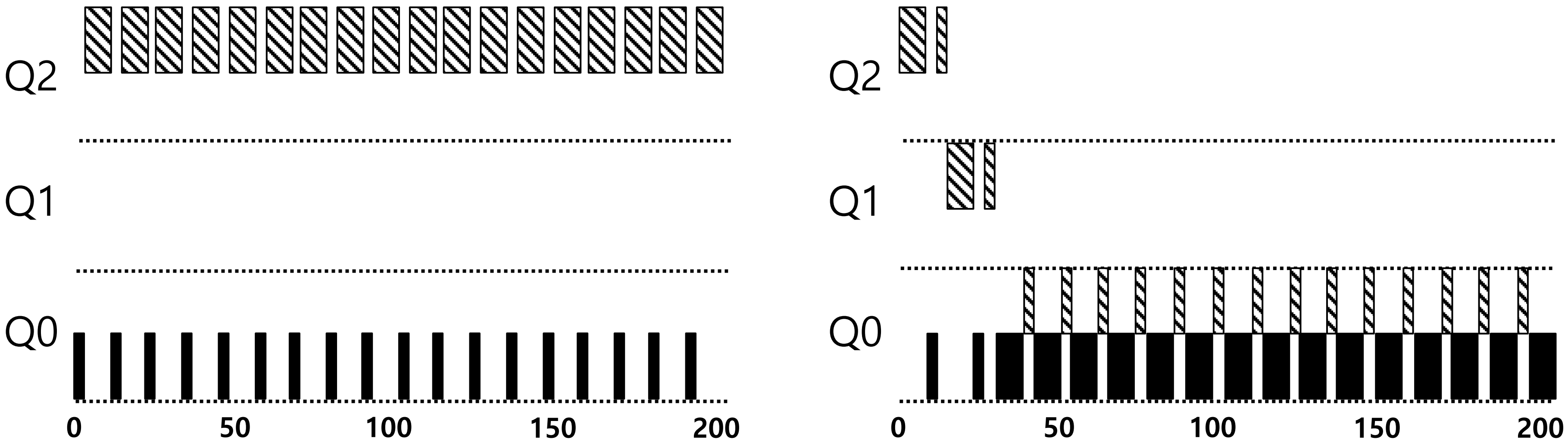
- Solves starvation and a CPU-bound process becoming interactive
- **Rule 5:** After some time period **S**, move all the processes in the system to the topmost queue
- What is a good value for **S**? – an interesting question
  - Example:
    - A long-running process(A) with two short-running interactive process(B, C), priority boost every S=50ms



Without(Left) and With(Right) Priority Boost

Approach #3: Better accounting of CPU time consumed by processes in each queue

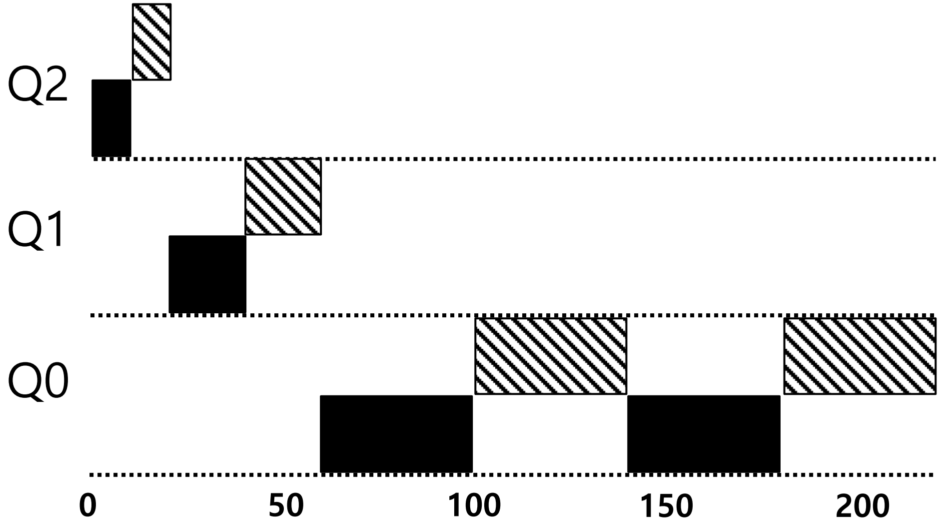
- Prevents a process from gaming the scheduler
- Solution:
  - **Rule 4** (Rewrite of Rules 4a and 4b): Once a process has consumed its time slice in a given queue (regardless of how many times it has given up the CPU), **its priority is reduced** (i.e., it moves down on queue).



Without(Left) and With(Right) Gaming Tolerance

Tuning MLFQ And Other Issues

- How do we parameterize the MLFQ scheduler?
  - # of queues, time slice per queue, best value for S, etc.
  - Typically:
    - High-priority queues are given short time slices
      - E.g., 10 or fewer milliseconds
    - Low-priority queues are given longer time slices
      - E.g., 100 milliseconds



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

Lower Priority, Longer Quanta

Tuning MLFQ And Other Issues (Cont.)

- For Solaris - Time-Sharing scheduling class (TS)
  - Uses a set of tables
  - 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
- For FreeBSD, a mathematical formula is used to determine priority, usage is **decayed** over time
- For other OSes, higher priority levels are given to the OS/kernel services
  - Allows users to give an **advice** to help set the priority levels, example: the `nice` command

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.