CPU Virtualization: Scheduling with Multi-level Feedback Queues

Prof. Patrick G. Bridges

Reminder

- **Schedulers seeks choose which job to run when to run given to optimize some scheduling metric**
  - Turn-around time
  - Response time
  - Lots of others…
- **For systems with mixed workloads, there's not generally an easy single metric to optimize**
- **General-purpose systems rely on heuristic schedulers that try to balance the qualitative performance of the system**
- **Question: What's wrong with round robin?**
- **Aside: How hard is "optimal" scheduling for an arbitrary performance metric?**

MLFQ (Multi-Level Feedback Queue)

Goal: general-purpose scheduling

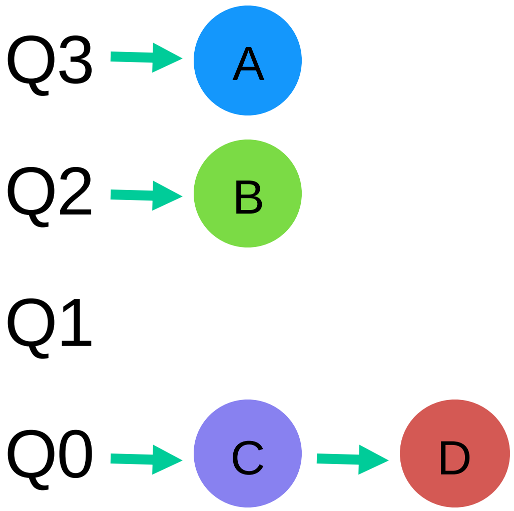
Must support two job types with distinct goals

- **"interactive" programs care about response time**
- **"batch" programs care about turnaround time**

Approach: multiple levels of round-robin;
each level has higher priority than lower levels and preempts them

Basic Mechanism: Multiple Prioritized RR Queues

- **Rule 1: If priority(A) > Priority(B), A runs**
- **Rule 2: If priority(A) == Priority(B), A & B run in RR**



“Multi-level”

Policy: how to set priority?

Approach 1: "nice” command
Approach 2: history “feedback”

MLFQ: Basic Rules (Cont.)

- **MLFQ varies the priority of a job based on its observed behavior.**
- **Example:**
  - A job repeatedly relinquishes the CPU while waiting IOs → Keep its priority high
  - A job uses the CPU intensively for long periods of time → Reduce its priority.

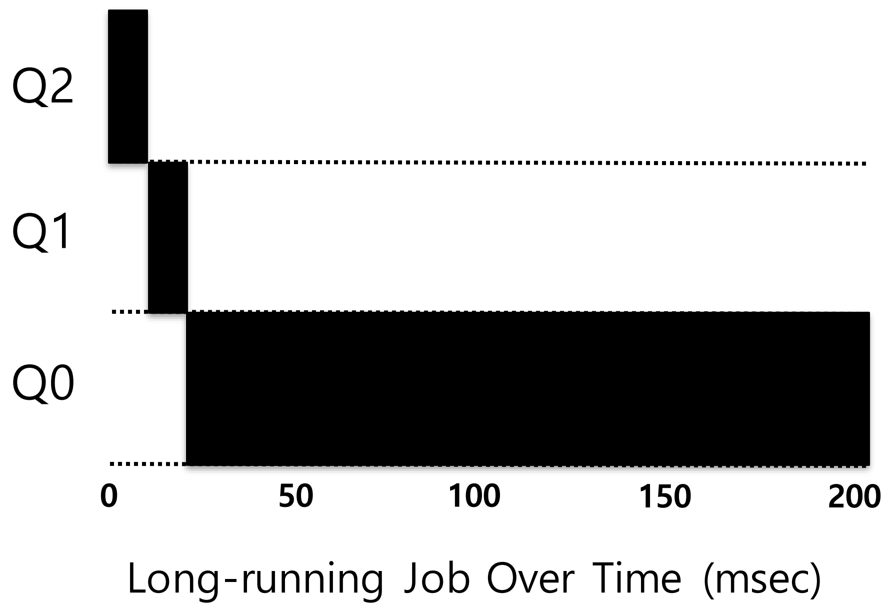
MLFQ: How to Change Priority

- **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

Example 1: A Single Long-Running Job

- A three-queue scheduler with time slice 10ms

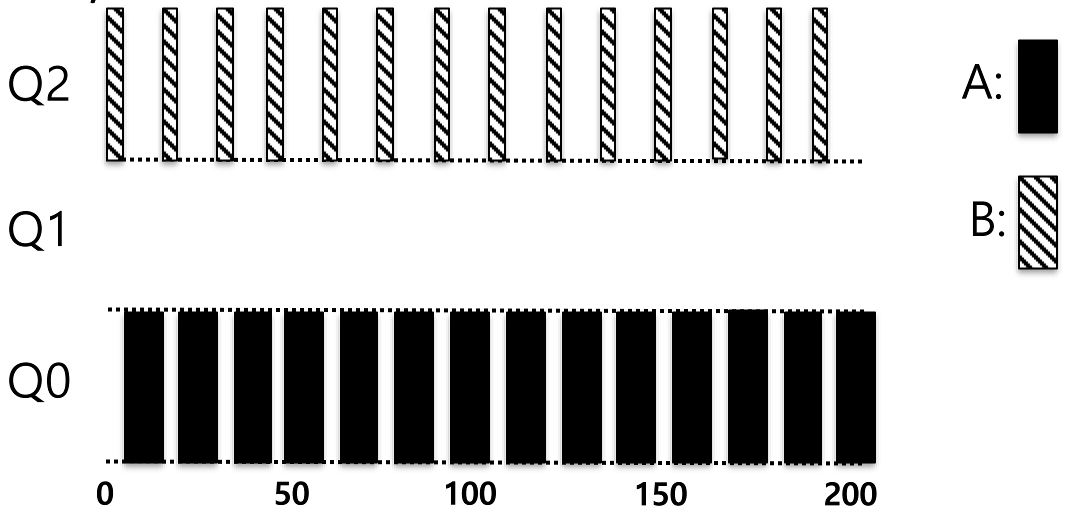


Example 2: Along Came a Short Job

- **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.

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



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

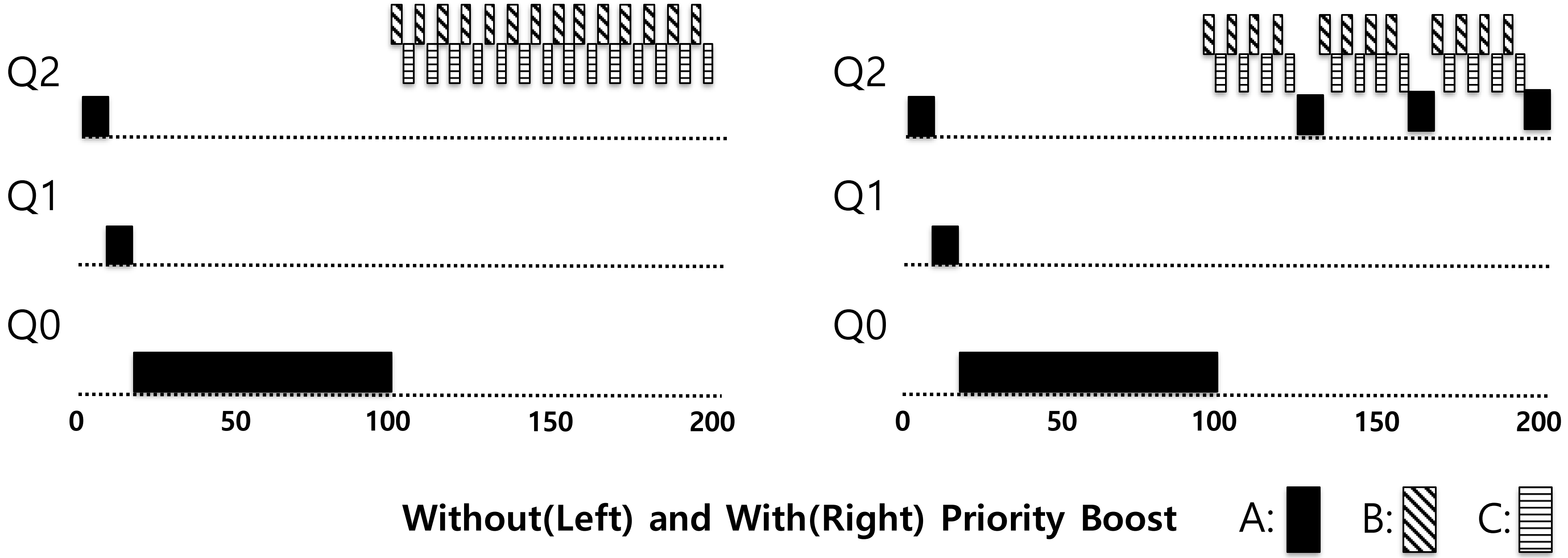
- **Starvation**
  - If there are “too many” interactive jobs in the system.
  - Lon-running jobs will never receive any CPU time.

- **Game 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 → I/O bound process

The Priority Boost

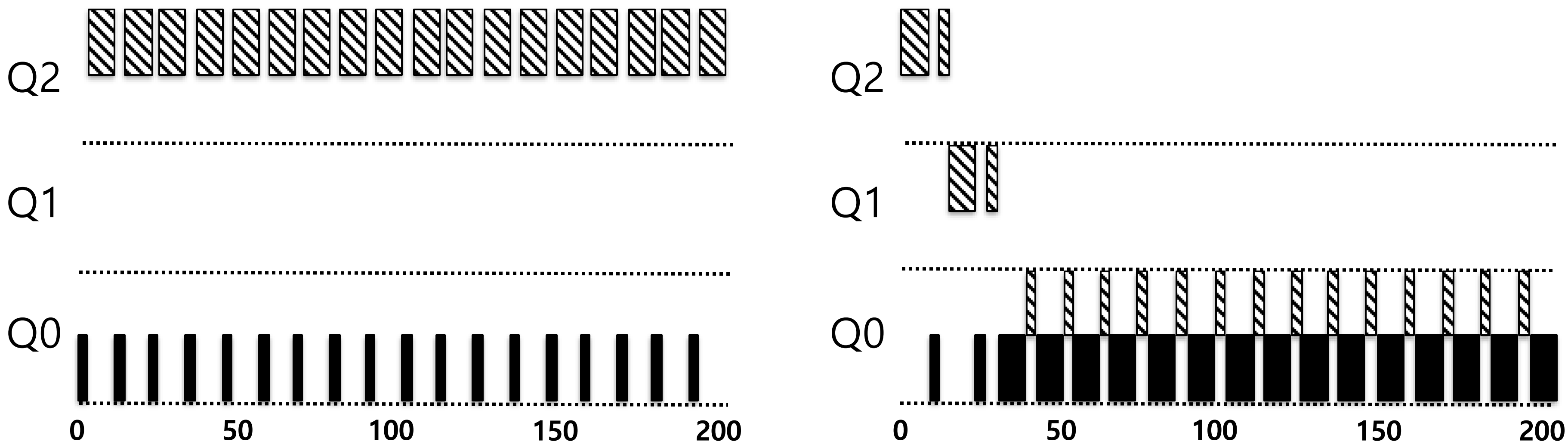
- **Rule 5: After some time period S, move all the jobs in the system to the topmost queue.**
  - Example:
    - A long-running job(A) with two short-running interactive job(B, C)



Without(Left) and With(Right) Priority Boost

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 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

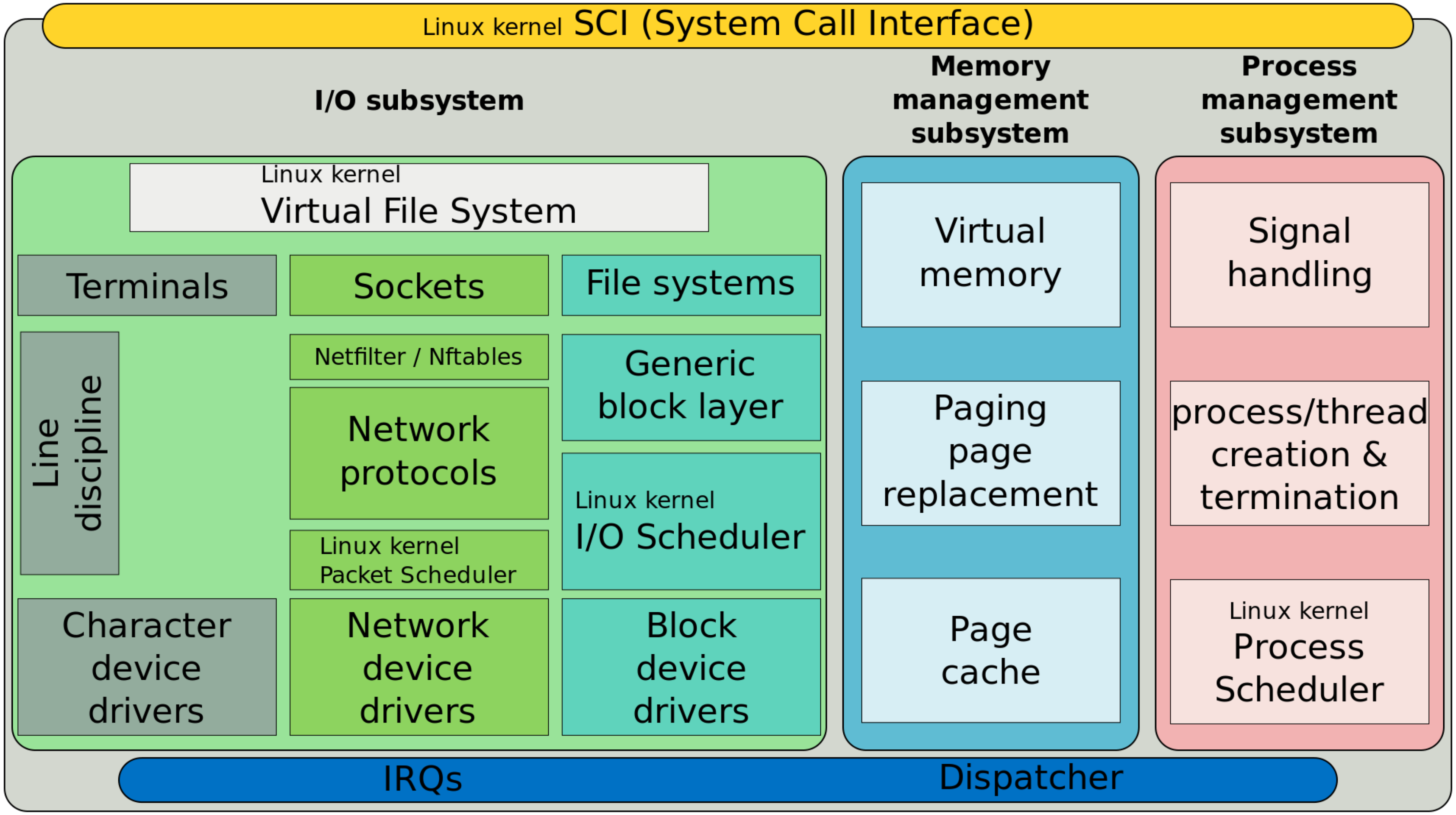
The Solaris MLFQ implementation

- **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.

Some slides added by Jed...



https://commons.wikimedia.org/wiki/File:Simplified_Structure_of_the_Linux_Kernel.svg

O(1) scheduler (older)

- **Two arrays, switching between them is just changing a pointer**
- **Uses heuristics to try to know which processes are interactive**
  - Average sleep time
- **https://en.wikipedia.org/wiki/O(1)_scheduler**

CFS scheduler (currently in Linux)

- **Completely Fair Scheduler**
- **Red-black tree of execution to the nanosecond**
  - niffies
- **Like weighted fair queuing for packet networks**
- **An ideal processor would share equally**
- **maximum execution time = time the process has been waiting to run / total number of processes**
- **https://en.wikipedia.org/wiki/Completely_Fair_Scheduler**

BFS (now MuQQS)

- **Brain “Hug” Scheduler**
- **Specifically for desktops**
- **Weighted round-robin where the weights are based on some very complex formulae (see Wikipedia for details)**
  - **No priority modification for sleep behavior**
  - **Time slice = 6ms (human perception of jitter ≈ 7ms)**
- **Performs slightly better than CFS for <16 cores**

- **https://en.wikipedia.org/wiki/Brain_Fuck_Scheduler**
- **https://lwn.net/Articles/720227/**