MLFQ

Multi level feedback queue.

In situations where you can adaptively learn from history, do so.

Goal of MLFQ

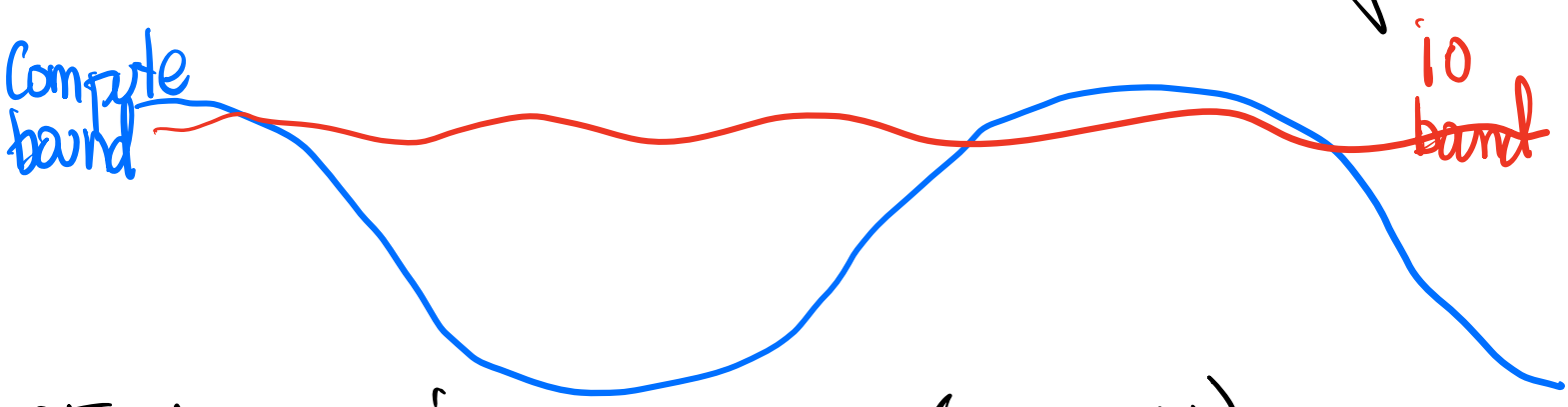
keep interactive processes interactive

Original Unix scheduler - i.e. the one in the original V6

The more you run, the lower your priority

The lower your priority, the less you run.

The less you run, the higher your priority



Interactive processes (e.g. vi) spends most of its time waiting for input... not running... so priority stays high!

MLFQ assumptions:

*Multiple levels of priority.
*Each has a queue of runnables
*Running tasks are not on any queue
*Waiting/blocked tasks are not on any queue.

Rule 1 if Pri(A) > Pri(B) run A... ie A is on queue above B.

Rule 2 RR* for tasks on same queue

* or other algorithm.

Rule 3 Jobs arrive on top queue

Rule 4a When job uses up quantum it is demoted.

4b If job gives up CPU, it returns to same queue (eg I/O)

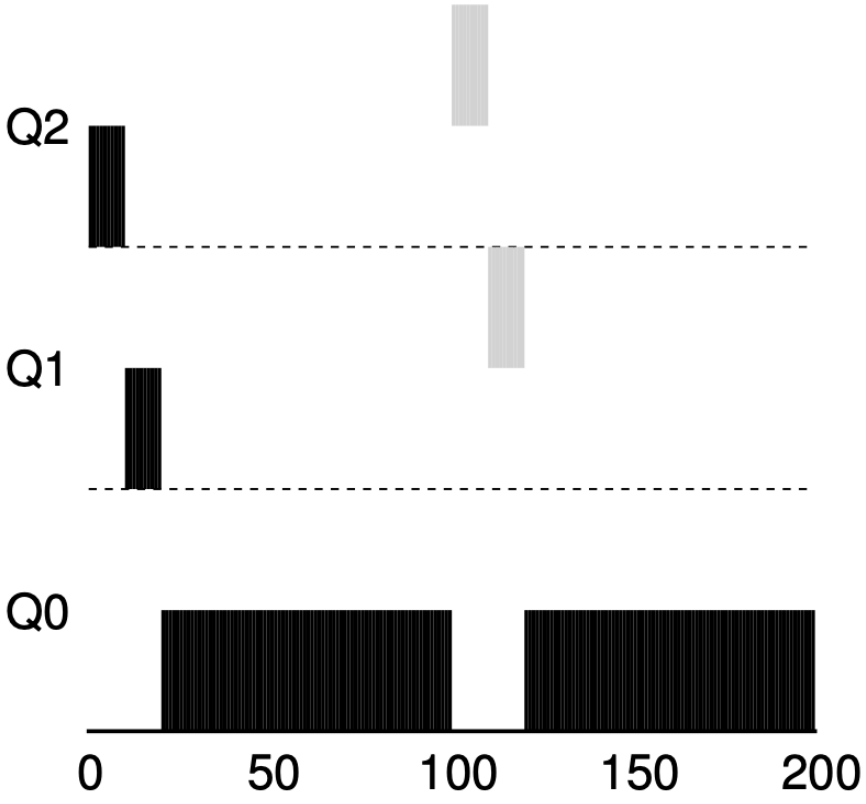
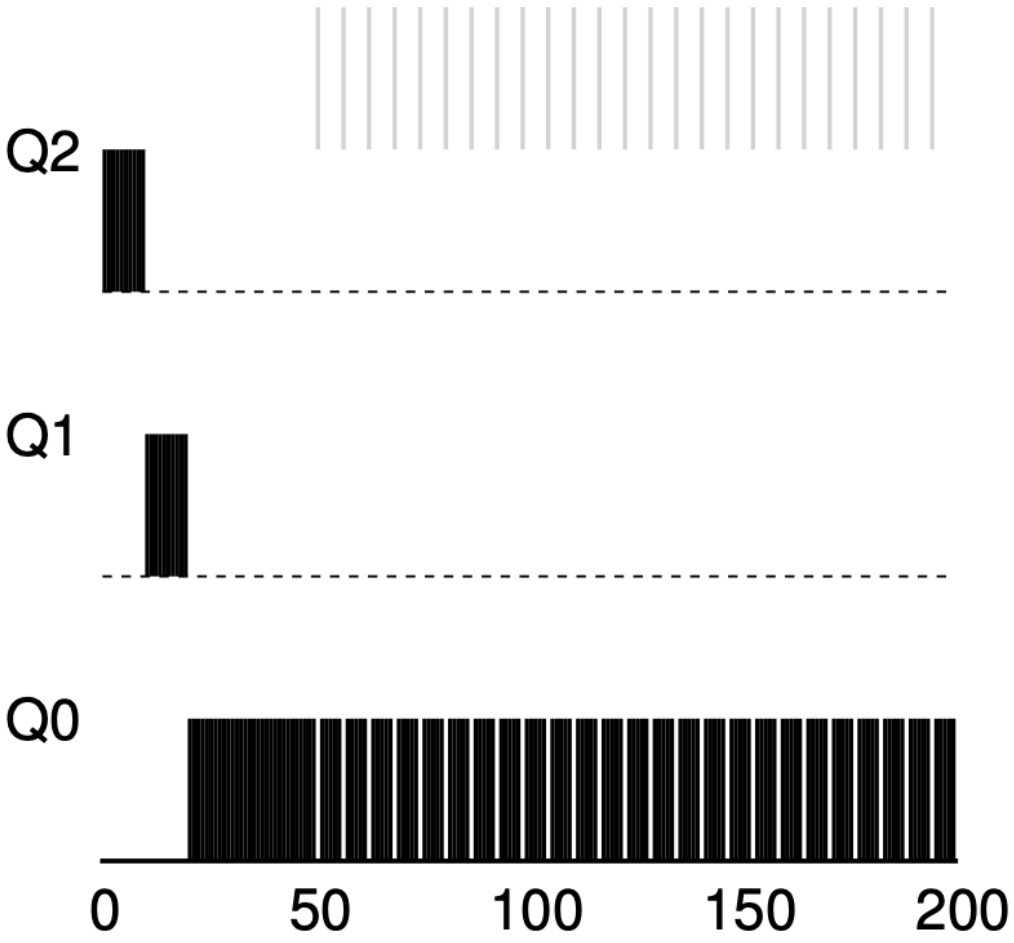


Figure 8.3: **Along Came An Interactive Job**

See how short jobs finished quickly, even w/o knowing ahead of time?



Doing I/O kept job in gray at high priority because it ran only briefly.

Could be an editor for example

Notice so far jobs only sink in priority.

What if compute bound job changes to interactive? tough

Rule 5

After a while (called EPOCH) put every job back on highest priority queue and start over.

Revise Rule 4 – no more oppty to cheat. Time on cpu measured cumulatively – can't cheat by doing an I/O

MLFQ is basis scheduler for

BSD derivatives

Solaris

Windows NT ⟶ Win 11