

Solutions to other MLQ problems

- ➔ **To prevent starvation of low priority thread**
change the priority over time by either
 - increase priority as a function of waiting time
 - or decrease priority as a function of CPU consumption
- ➔ **To decide on the priority**
by observing and keeping track of the thread CPU usage

MLFQ - Multilevel **Feedback** Queue Scheduling (preemptive)

➔ Same as MLQ but change the priority of the process based on *observations*

Rule 1	If $\text{Priority}(A) > \text{Priority}(B)$, A runs
Rule 2	If $\text{Priority}(A) = \text{Priority}(B)$, A & B run in round-robin fashion using the time slice (quantum length) of the given queue
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

✓ **Good** : Turing-award winner algorithm