CPU Virtualization: Scheduling

Sridhar Alagar

Sharing the CPU

- Mechanism - Dispatcher
  - How to switch to another process?
- Policy - Scheduler
  - Which process to switch to?

Workload assumptions

1. Each job runs for the **same amount of time**

2. All jobs **arrive** at the same time

3. All jobs only use the **CPU** (i.e., they perform no I/O)

4. The **run-time** of each job is known

Performance metric

- Turnaround time

$$T_{turnaround} = T_{completion} - T_{arrival}$$

- Other metrics?
  - Fairness

- Metrics can be conflicting with each other

FIFO (or FCFS)

- Run jobs the order in which they arrived
  - Easy to implement
- Example

jobs	arrival time (s)	run time (s)
A	~0	10
B	~0	10
C	~0	10

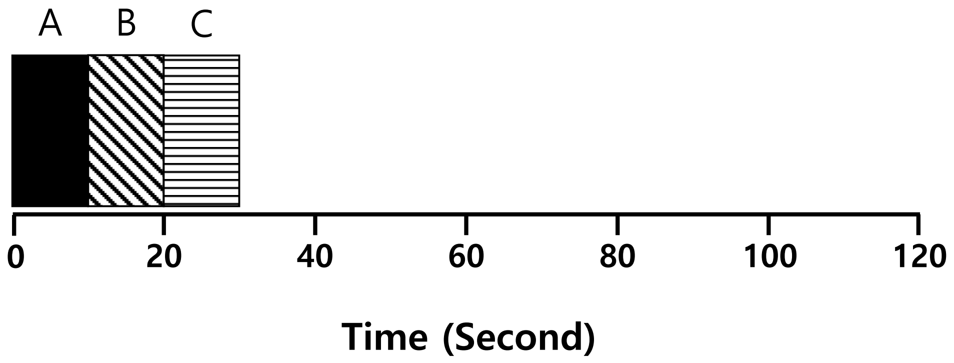
FIFO - Event trace

jobs	arrival time (s)	run time (s)
A	~0	10
B	~0	10
C	~0	10

Time	Event
0	A arrives
0	B arrives
0	C arrives
0	run A
10	complete A
10	run B
20	complete B
20	run C
30	complete C

FIFO - Gantt Chart

jobs	arrival time (s)	run time (s)
A	~0	10
B	~0	10
C	~0	10



$$\boldsymbol{Average\ turnaround\ time = \frac{10 + 20 + 30}{3} = 20\ secs}$$

Workload assumptions

1. ~~Each job runs for the **same amount of time**~~

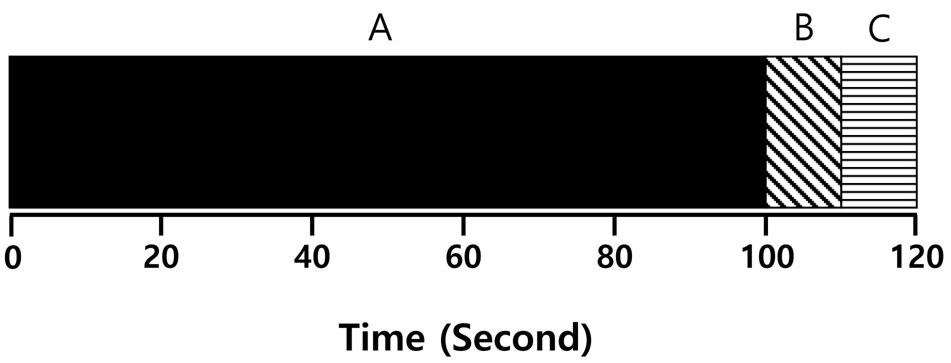
2. All jobs **arrive** at the same time

3. All jobs only use the **CPU** (i.e., they perform no I/O)

4. The **run-time** of each job is known

FIFO - Big job first

jobs	arrival time (s)	run time (s)
A	~0	100
B	~0	10
C	~0	10



$$\boldsymbol{Average\ turnaround\ time} = \frac{\mathbf{100 + 110 + 120}}{\mathbf{3}} = \mathbf{110}\ \boldsymbol{sec}$$

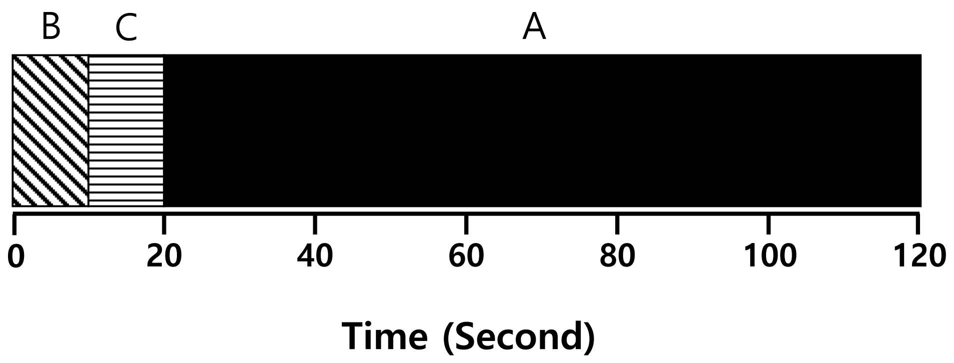
Convoy effect

Passing the tractor

- Problems with FIFO
  - Short jobs have to wait for long jobs to finish
- New Scheduler – Shortest job first
  - choose the job with the smallest run time as the next job

Shortest job first (SJF)

jobs	arrival time (s)	run time (s)
A	~0	100
B	~0	10
C	~0	10



$$Average\ turnaround\ time = \frac{120 + 10 + 20}{3} = 50\ sec$$

Shortest job first (SJF)

- Moving shorter jobs before longer jobs improves the turnaround time for shorter jobs
- Moving longer jobs later does not affect the overall completion time
- SJF is optimal (provable)

Workload assumptions

1. ~~Each job runs for the **same amount of time**~~

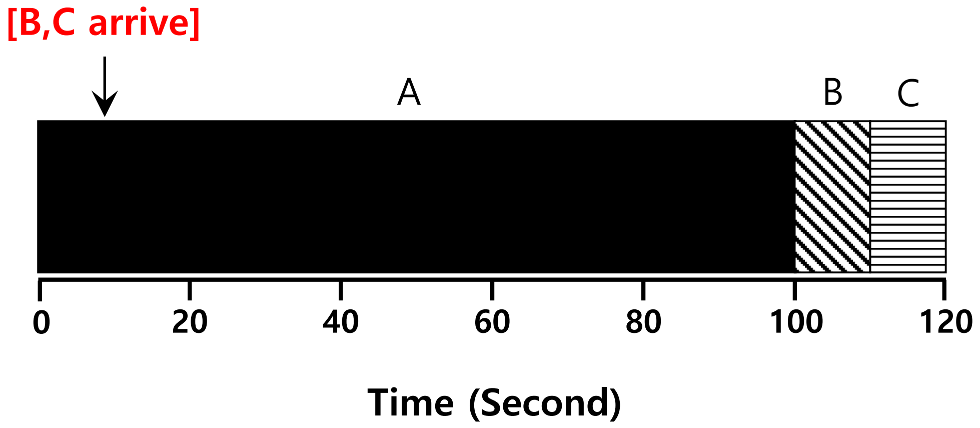
2. ~~All jobs arrive at the same time~~

3. All jobs only use the **CPU** (i.e., they perform no I/O)

4. The **run-time** of each job is known

SJF - different arrival times

jobs	arrival time (s)	run time (s)
A	~0	100
B	~10	10
C	~10	10



$$\textit{Average turnaround time} = \frac{\mathbf{100 + (110 - 10) + (120 - 10)}}{\mathbf{3}} = \mathbf{103.3}\ \textit{secs}$$

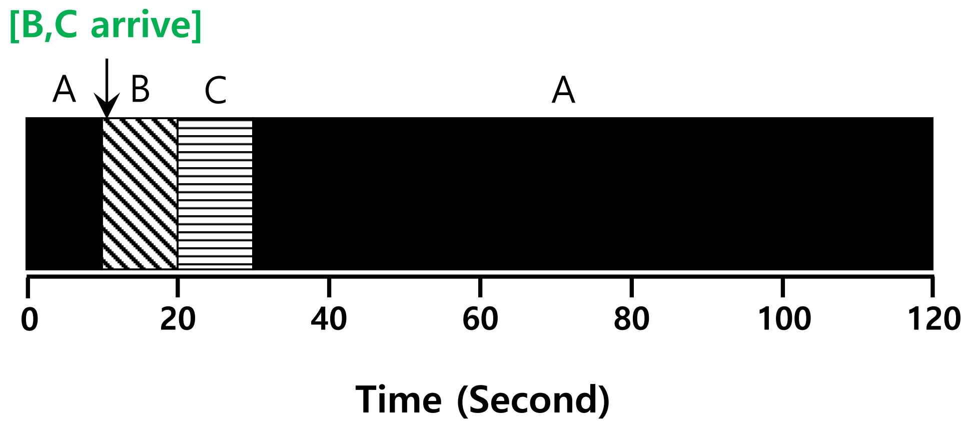
Stuck behind the tractor again!!

Shortest time to completion first (STCF)

- When a new job arrives, check if it will complete sooner than the job running in the CPU
  - if so, switch to the new job
- Preemption is required
  - we already know how to switch context
- After completing a job, execute the job with shortest time

STCF - different arrival times

jobs	arrival time (s)	run time (s)
A	~0	100
B	~10	10
C	~10	10

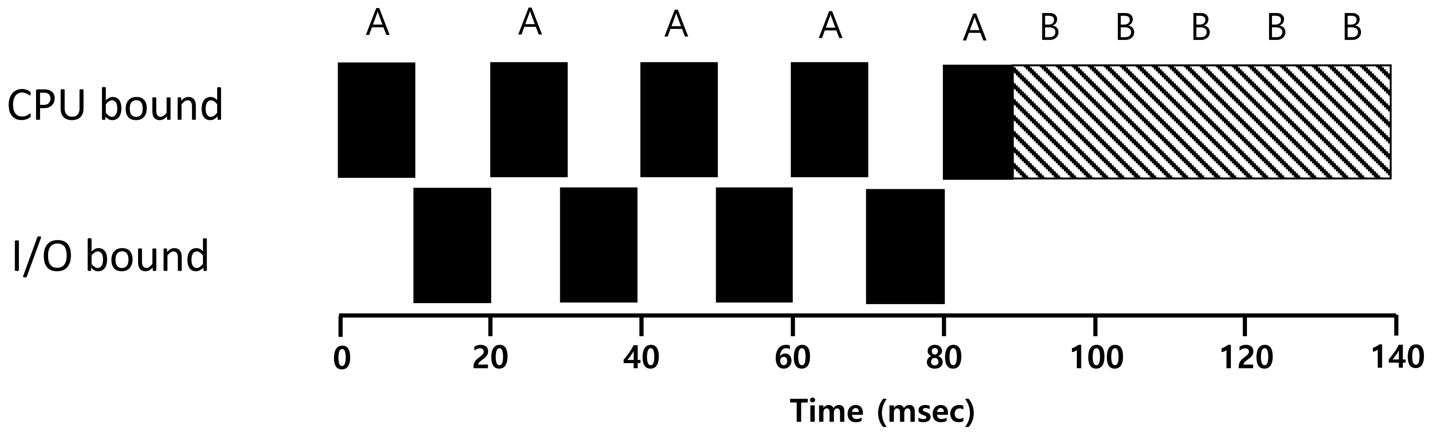


$$\boldsymbol{Average\ turnaround\ time} = \frac{\mathbf{120 + (20 - 10) + (30 - 10)}}{\mathbf{3}} = \mathbf{50}\ \boldsymbol{secs}$$

Workload assumptions

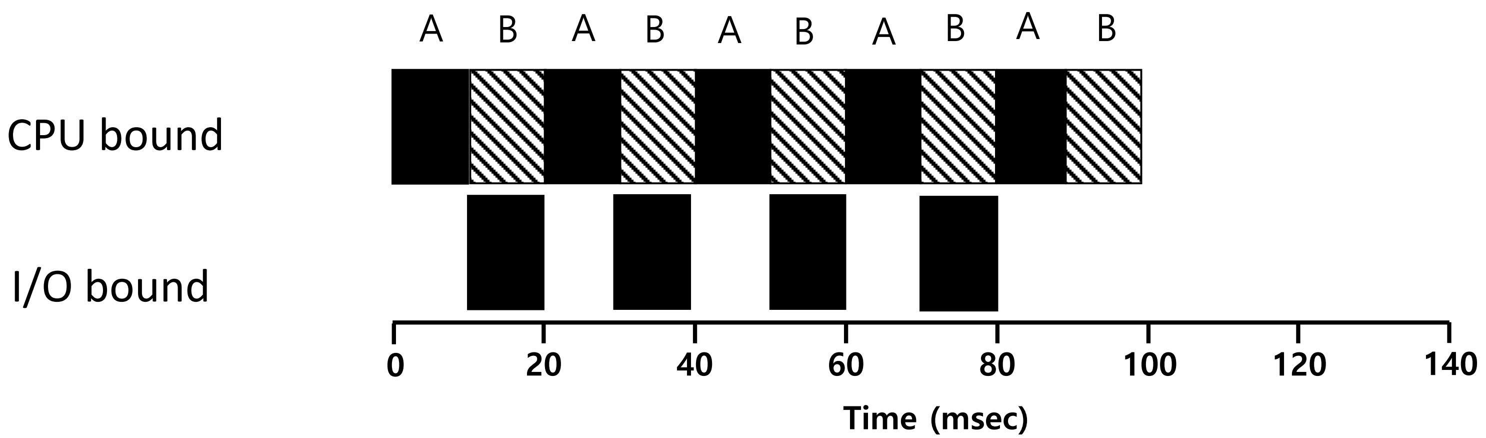
1. ~~Each job runs for the **same amount of time**~~
2. ~~All jobs arrive at the same time~~
3. ~~All jobs only use the CPU (i.e., they perform no I/O)~~
4. The **run-time** of each job is known

Not I/O aware



Poor use of resources

I/O aware



- Treat job A as several separate CPU bursts
- When job A completes I/O, another job A is ready
- Each CPU burst is shorter than job B, so with SCTF, job A preempts job B

New metric: Response time

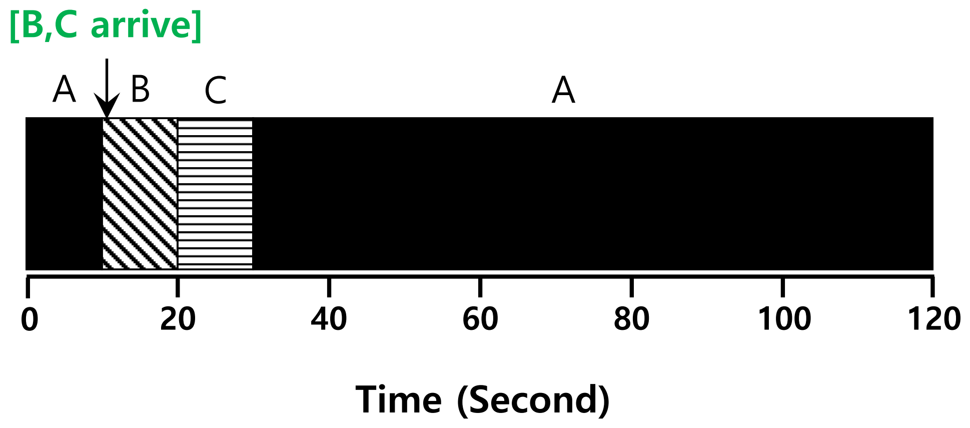
- Important when a job is started than when it is finished
  - interactive jobs

$$T_{response} = T_{firstrun} - T_{arrival}$$

- FCFS, SJF, STCF are not good at minimizing response time

Response time vs turnaround time

jobs	arrival time (s)	run time (s)
A	~0	100
B	~10	10
C	~10	10



$$\boldsymbol{response\ time\ for\ C = (20\ - 10) = 10\ secs}$$

How to minimize the response time?

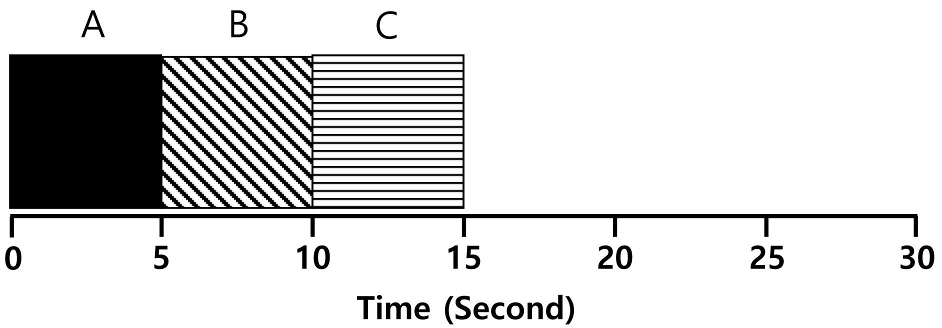
Round robin scheduler

- Alternate ready processes every fixed-length time-slice

- Time-slice also called as time-quantum

- Length of quantum could be a multiple of timer interrupt period

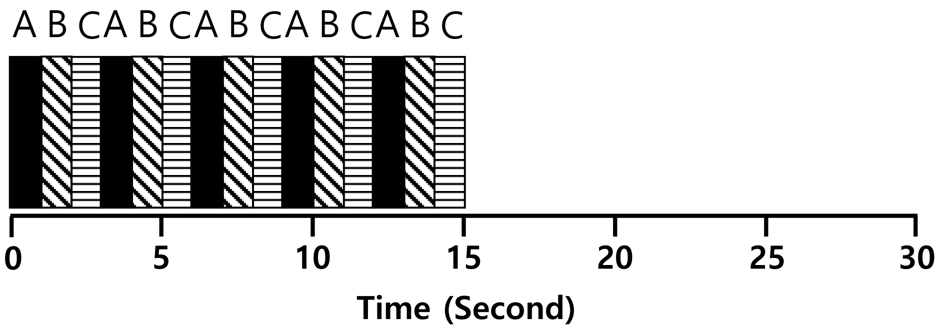
RR vs SJF scheduling



SJF (Bad for Response Time)

$$T_{average\ response} = \frac{0+5+10}{3} = 5\ secs$$

$$T_{average\ turnaround} = \frac{5+10+15}{3} = 10\ secs$$



RR with a time-slice of 1sec (Good for Response Time)

$$T_{average\ response} = \frac{0+1+2}{3} = 1\ secs$$

$$T_{average\ turnaround} = \frac{13+14+15}{3} = 14\ secs$$

Length of time-slice is critical

- Shorter time-slice
  - better response time
  - too many context switching. Overhead becomes high
- Longer time-slice
  - fewer context switching
  - poorer response time
- A trade-off

Another benefit of round robin

- Run time need not be known
- In fact, RR does not care whether a process terminates or not

Workload assumptions

1. ~~Each job runs for the **same amount of time**~~

2. ~~All jobs arrive at the same time~~

3. ~~All jobs only use the CPU (i.e., they perform no I/O)~~

4. ~~The run-time of each job is known~~

General Purpose Scheduling

- Need to support two types of jobs with different goals
    - interactive jobs care about response time
    - batch jobs care about turn around time

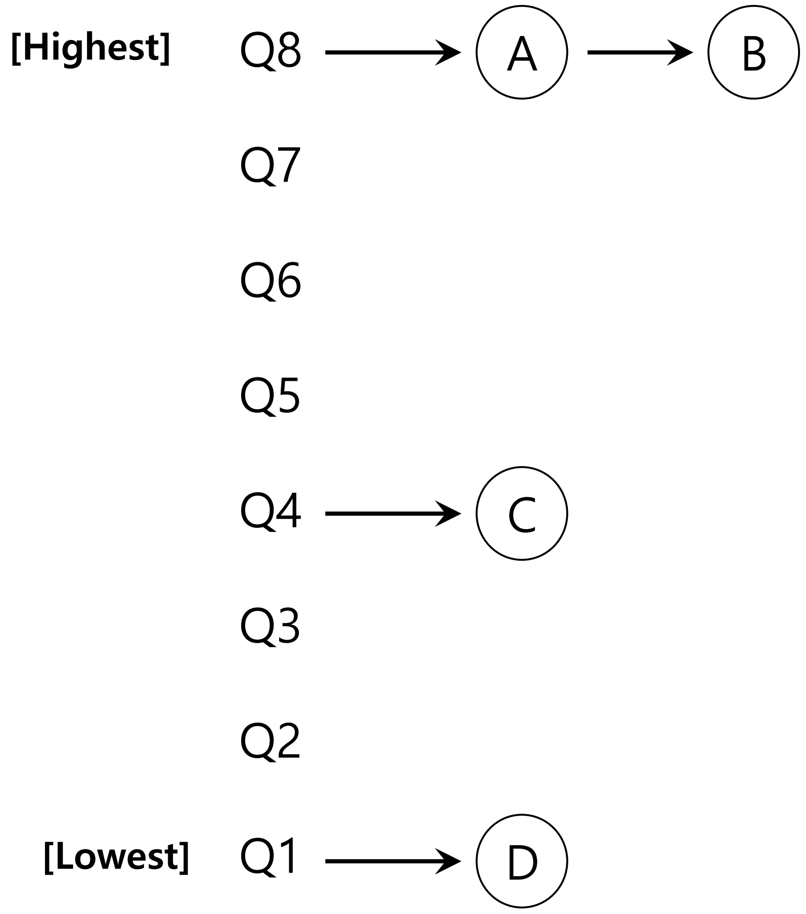
- Run interactive jobs first
- Run shorter jobs before longer ones

- Need to prioritize

Multi level feedback queue (MLFQ)

- Every level has a priority;
  - top level has the highest

- Rule 1: If Priority(A) > Priority(B), A runs (B doesn't).
- Rule 2: If Priority(A) = Priority(B), A & B run in RR.



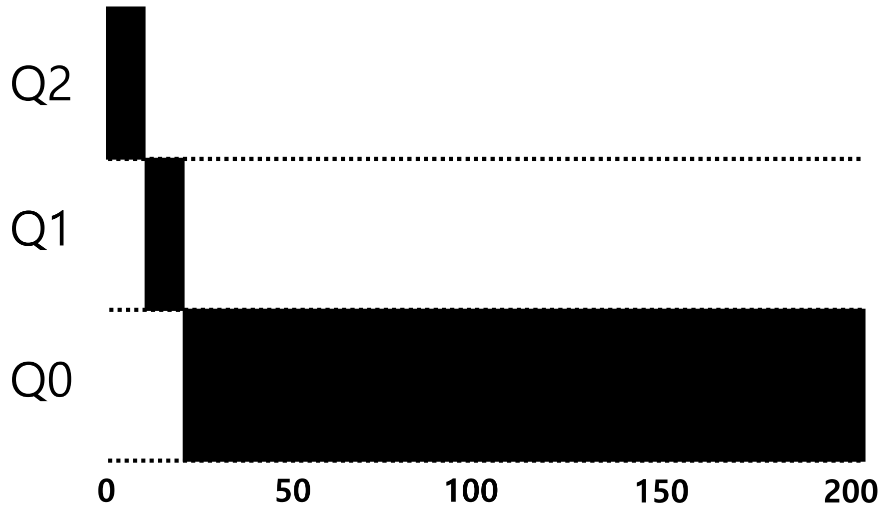
How to set priority?

- Typically, jobs alternate between CPU and I/O bursts
- Use past history to predict the future CPU burst time
- A new job should be high priority if
  - it is interactive – quick response
  - it is a short job – faster turn around time

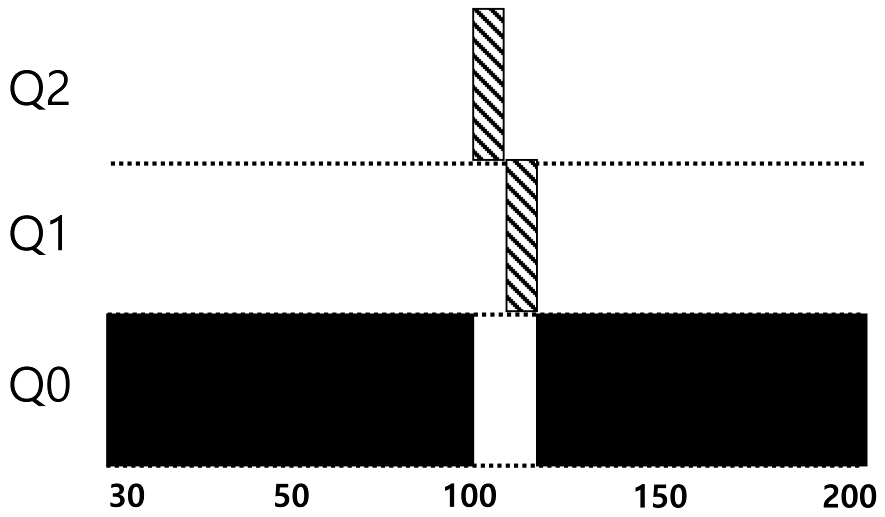
More MLFQ rules

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

A single long job

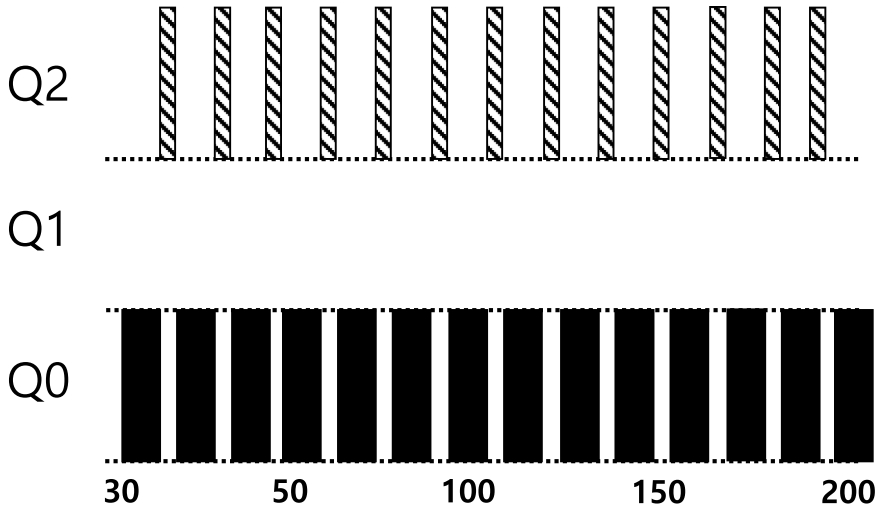


Along came a short job



Mimics SJF

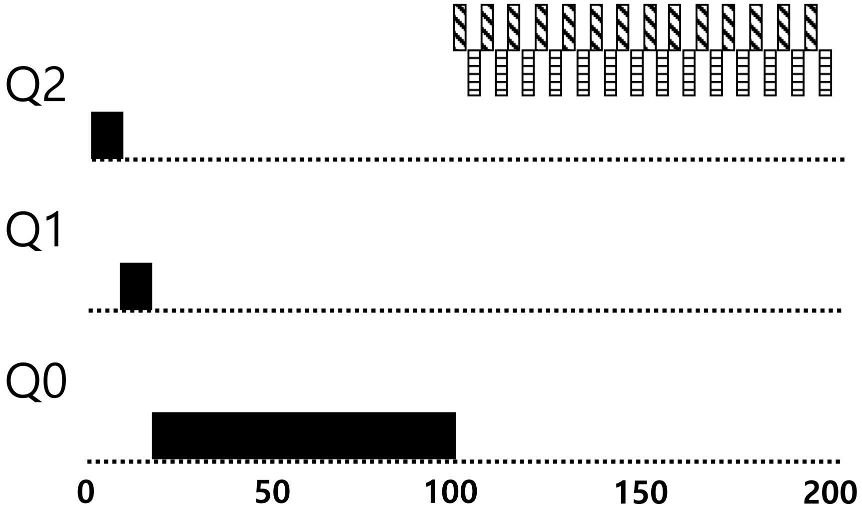
An interactive process joins



- An interactive process never uses the entire time slice. So never demoted. Quick response time

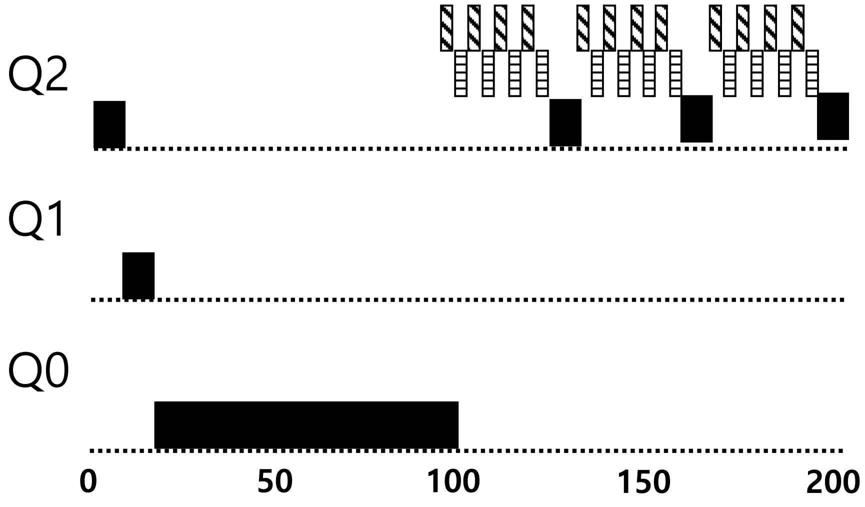
Problems with MLFQ

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



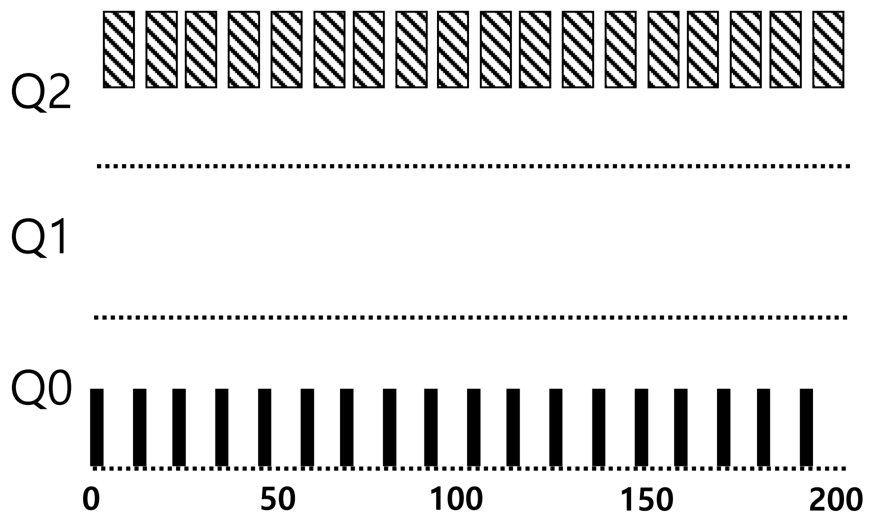
Prevent starvation: Priority Boost

- **Rule 5**: After some time period S, move all the jobs in the system to the topmost queue



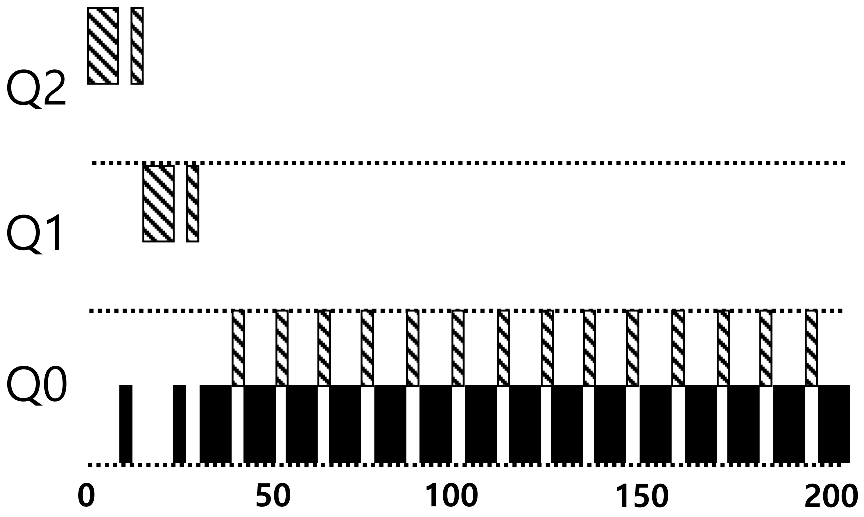
Problems with MLFQ

- Game the scheduler
  - After running 99% of a time slice, issue an I/O operation
  - The job gain a higher percentage of CPU time



Prevent gaming: Better accounting

- **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)



Tuning MLFQ

- High priority queue – short time slice

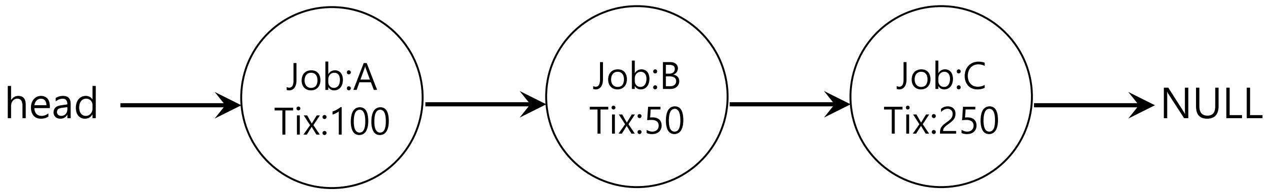
- Low priority queue – large time slice

Lottery Scheduling

- Proportional (fair) share
  - each job is given a certain percentage of time
- Outline
  - allocate each job its share of lottery tickets
  - high priority job gets more tickets
  - whoever wins the lottery runs
- Very simple to implement

Implementation

- Who wins?
  - winning ticket # 300
  - winning ticket # 75
  - winning ticket # 120



Implementation

```
int counter = 0;
int winner = getrandom(0, totaltickets);
node_t *current = head;
while(current) {
    counter += current->tickets;
    if (counter > winner) break;
    current = current->next;
}
// current gets to run
```

Disclaimer

- Some of the materials in this lecture slides are from the materials prepared by Prof. Arpaci, and Prof. Youjip. Thanks to all of them.