Scheduling: RR, Priority, MLFQ, and Lottery

CS 571: Operating Systems (Spring 2020)
Lecture 5

Yue Cheng

- CPU scheduling worksheet posted on BB

Review: FIFO, SJF

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 (no I/O)
4. The run-time of each job is known

FIFO

- First-In, First-Out: Run jobs in arrival (time) order

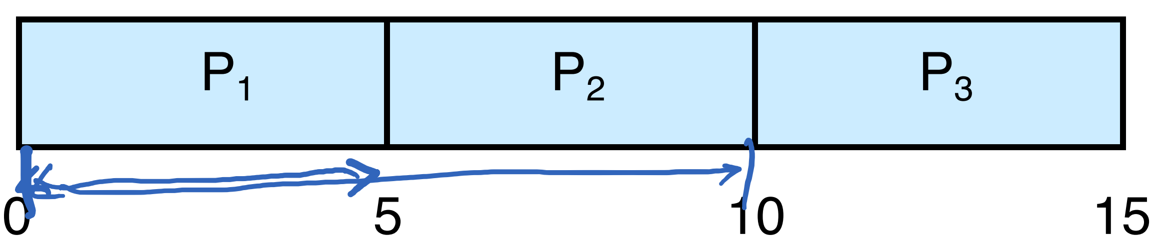
FIFO

First-In, First-Out: Run jobs in arrival (time) order

Def: waiting_time = start_time – arrival_time

Process	Burst Time
P_1	5
P_2	5
P_3	5

- Suppose that the processes arrive in order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule:



- Waiting time for P_1 = 0; P_2 = 5; P_3 = 10
- Average waiting time: 5

15/3 = 5

FIFO

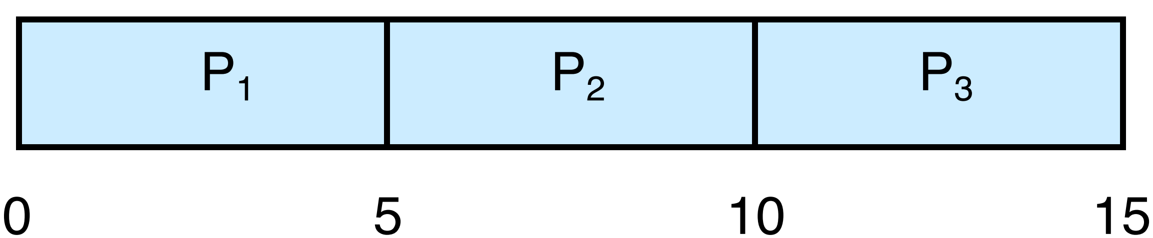
First-In, First-Out: Run jobs in arrival (time) order

What is the average turnaround time? (Q2)?

Def: turnaround_time = completion_time – arrival_time

Process	Burst Time
P_1	5
P_2	5
P_3	5

- Suppose that the processes arrive in order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule:



- Waiting time for P_1 = 0; P_2 = 5; P_3 = 10
- Average waiting time: 5

FIFO

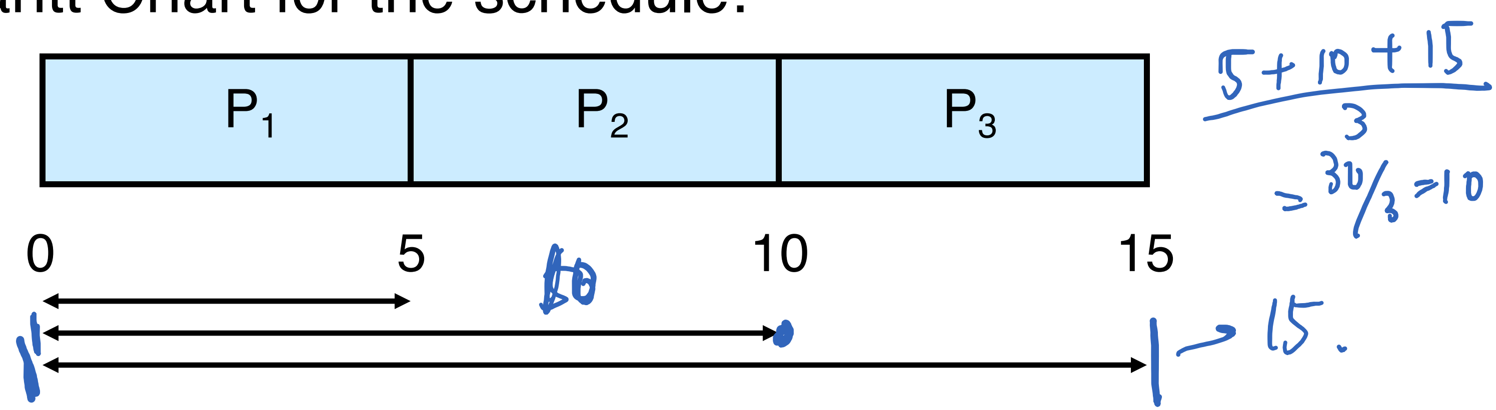
First-In, First-Out: Run jobs in arrival (time) order

What is the average turnaround time? (Q2)?

Def: turnaround_time = completion_time – arrival_time

Process	Burst Time
P_1	5
P_2	5
P_3	5

- Suppose that the processes arrive in order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule:



FIFO

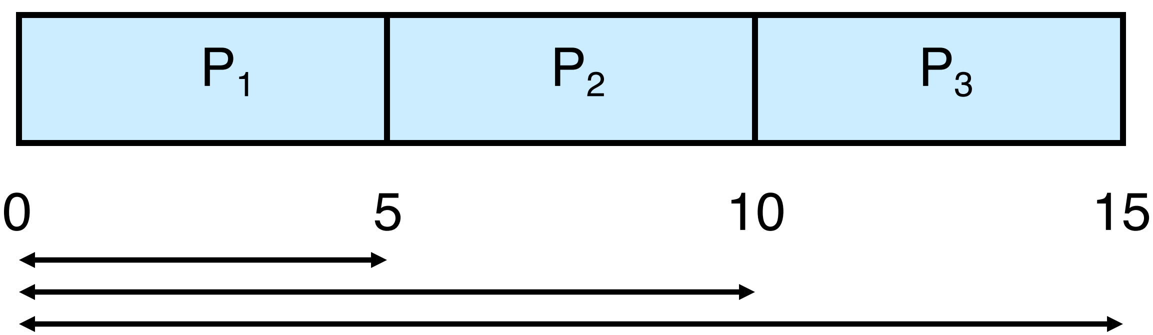
First-In, First-Out: Run jobs in arrival (time) order

What is the average turnaround time? (Q2)?

Def: turnaround_time = completion_time – arrival_time

Process	Burst Time
P_1	5
P_2	5
P_3	5

- Suppose that the processes arrive in order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule:



Average turnaround time: (5+10+15)/3 = 10

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 (no I/O)

4. The run-time of each job is known

Workload Assumptions

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

2. All jobs arrive at the same time

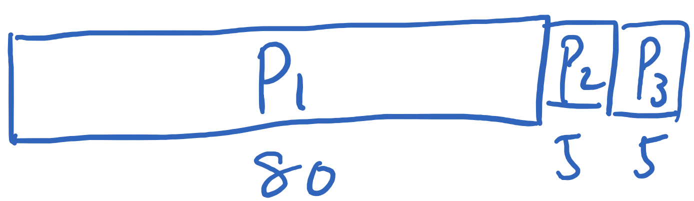
3. All jobs only use the CPU (no I/O)

4. The run-time of each job is known

Example: Big First Job

JOB	arrival_time	run_time
P1	~0	80
P2	~0	5
P3	~0	5

What is the average turnaround time? (Q3)



$$\frac{80+85+90}{3} = \frac{165+90}{3} = \frac{255}{3}$$

Example: Big First Job

JOB	arrival_time	run_time
P1	~0	80
P2	~0	5
P3	~0	5

Example: Big First Job

JOB	arrival_time	run_time
P1	~0	80
P2	~0	5
P3	~0	5

Average turnaround time: (80+85+90) / 3 = 85

Convoy Effect

Better Schedule?

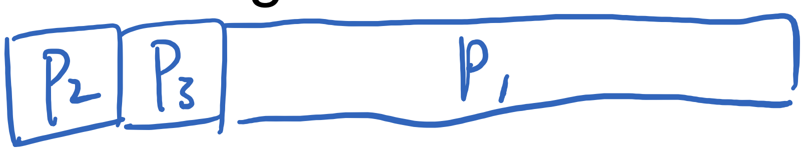
Passing the Tractor

- New scheduler: SJF (Shortest Job First)
- Policy: When deciding which job to run, choose the one with the smallest run_time

Example: SJF

JOB	arrival_time	run_time
P1	~0	80
P2	~0	5
P3	~0	5

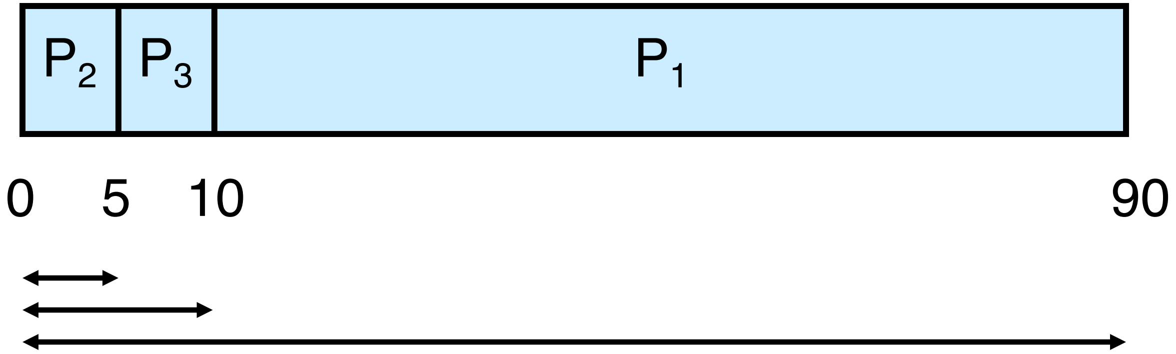
What is the average turnaround time with SJF? (Q4)



$$\frac{5 + 10 + 90}{3} = \frac{105}{3}$$

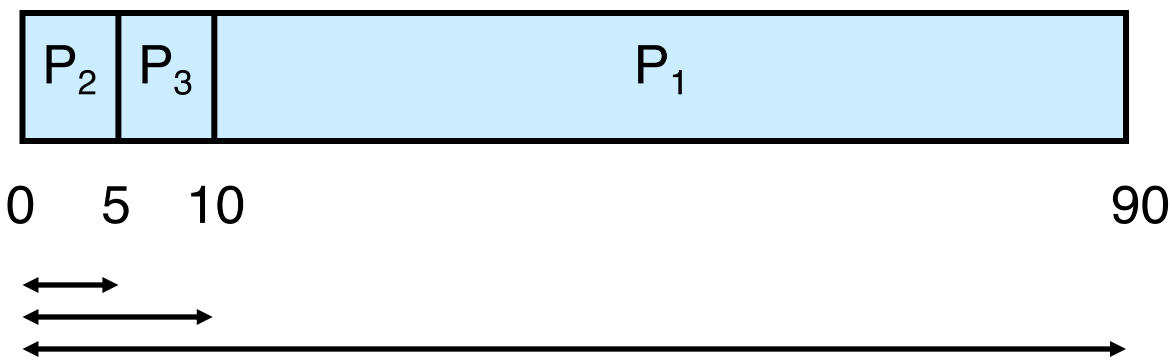
Example: SJF

JOB	arrival_time	run_time
P1	~0	80
P2	~0	5
P3	~0	5



Example: SJF

JOB	arrival_time	run_time
P1	~0	80
P2	~0	5
P3	~0	5



35<<85

Average turnaround time: (5+10+90) / 3 = 35

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 (no I/O)
4. The run-time of each job is known

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 (no I/O)
4. The run-time of each job is known

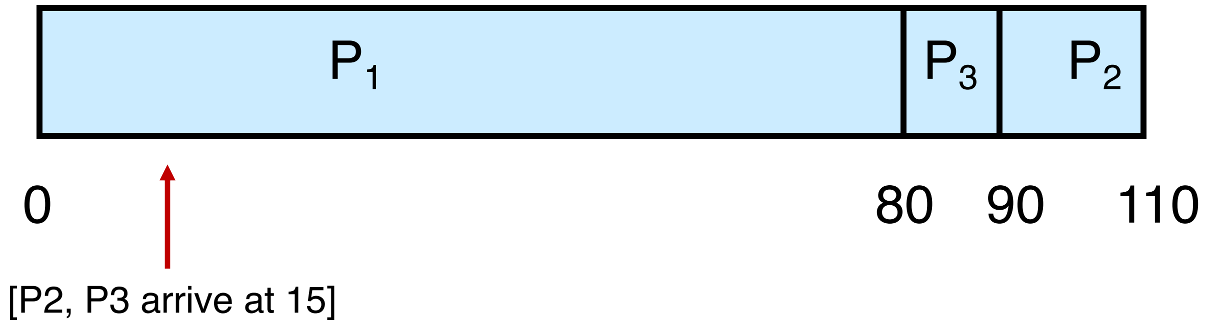
Shortest Job First (Arrival Time)

JOB	arrival_time	run_time
P1	~0	80
P2	**~15**	20
P3	**~15**	10

What is the average turnaround time with SJF? (Q5)

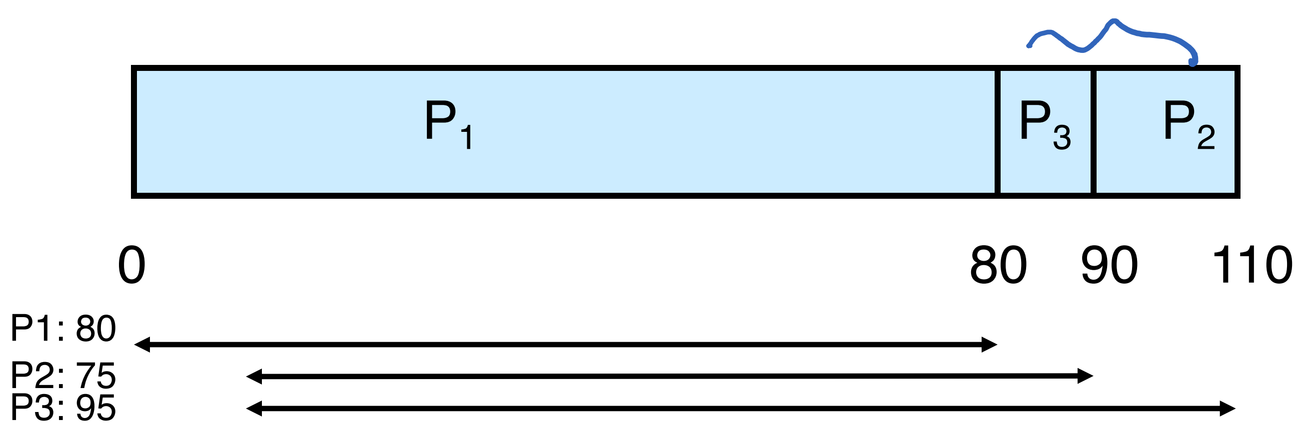
Shortest Job First (Arrival Time)

JOB	arrival_time	run_time
P1	~0	80
P2	**~15**	20
P3	**~15**	10



Shortest Job First (Arrival Time)

JOB	arrival_time	run_time
P1	~0	80
P2	**~15**	20
P3	**~15**	10



FIFO: 85

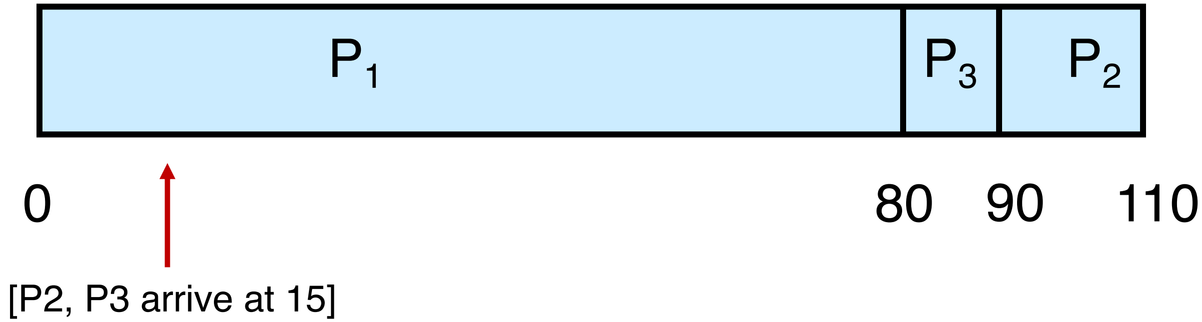
Average turnaround time: (80+75+95) / 3 = ~83.3

A Preemptive Scheduler

- Previous schedulers: FIFO and SJF are non-preemptive
- New scheduler: STCF (Shortest Time-to-Completion First)
- Policy: Switch jobs so we always run the one that will complete the quickest

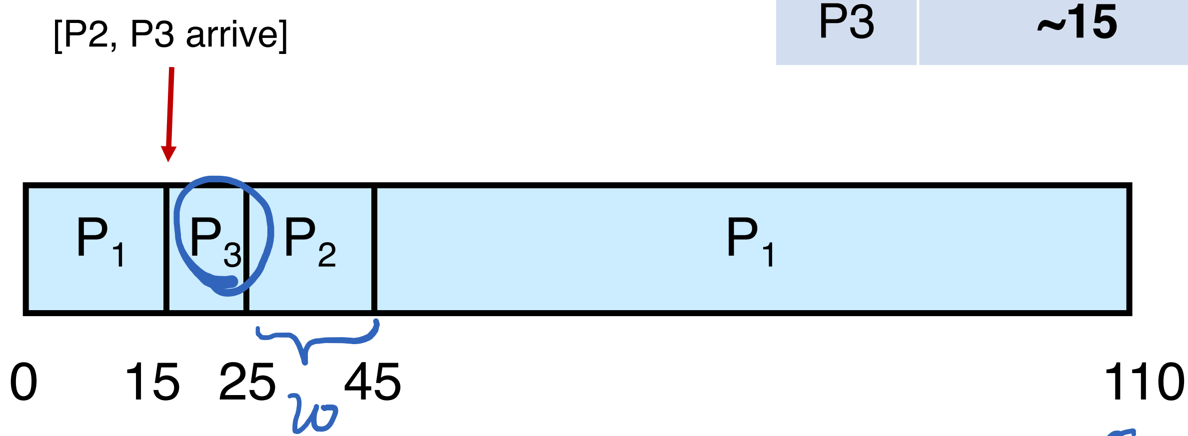
SJF

JOB	arrival_time	run_time
P1	~0	80
P2	**~15**	20
P3	**~15**	10



STCF

JOB	arrival_time	run_time
P1	~0	80
P2	**~15**	20
P3	**~15**	10



What is the average turnaround time with SRTF? (Q6)

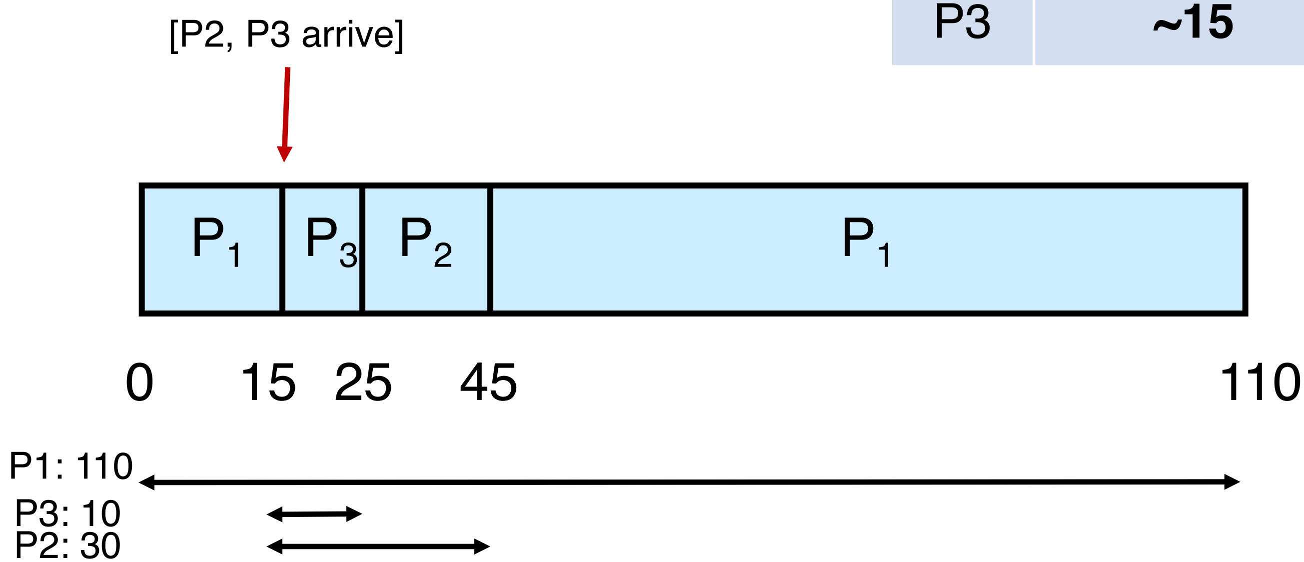
P1 : 110

P2 : 45 − 15 = 30

P3 : 25 − 15 = 10

STCF

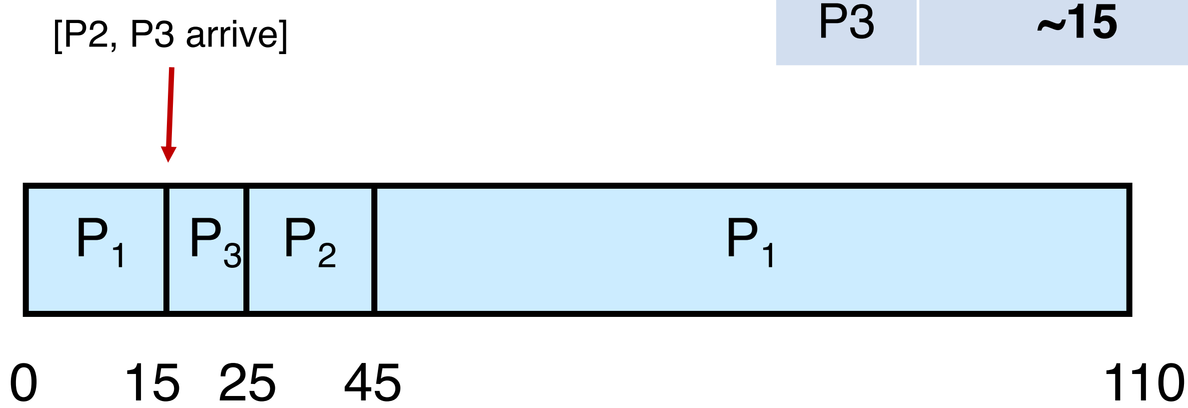
JOB	arrival_time	run_time
P1	~0	80
P2	**~15**	20
P3	**~15**	10



Average turnaround time: (110+30+10) / 3 = 50

STCF

JOB	arrival_time	run_time
P1	~0	80
P2	**~15**	20
P3	**~15**	10



What is the average waiting time with STCF? (Q7)

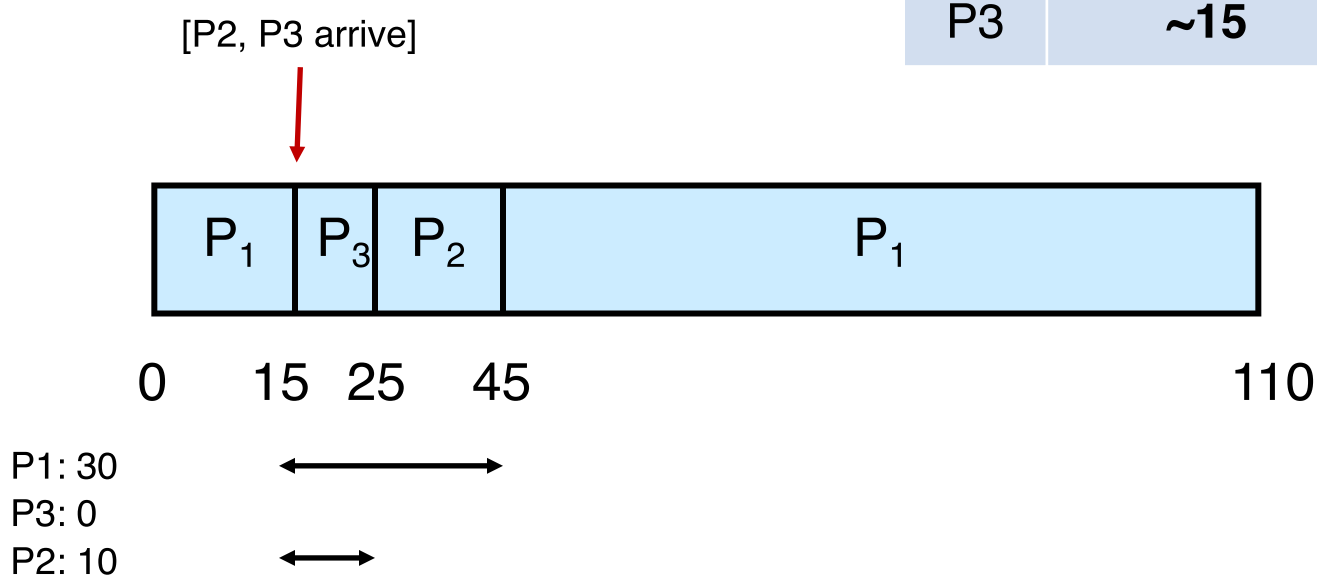
$P_1 : 0 + (45 - 15) = 30$

$P_2 : 25 - 15 = 10$

$P_3 : 0$

STCF

JOB	arrival_time	run_time
P1	~0	80
P2	**~15**	20
P3	**~15**	10



Average waiting time: (30+10+0) / 3 = ~13.3

Outline

- Scheduling algorithms
  - First In, First Out (FIFO)
  - Shortest Job First (SJF)
  - Shortest Time-to-Completion First (STCF)
    - Optimality discussion
  - Round Robin (RR)
  - Priority
  - Multi-Level Feedback Queue (MLFQ)
  - Lottery Scheduling

Optimality of SJF and STCF

- Non-preemptive SJF is optimal if all the processes are ready simultaneously
    - Gives minimum average waiting time for a given set of processes

Optimality of SJF and STCF

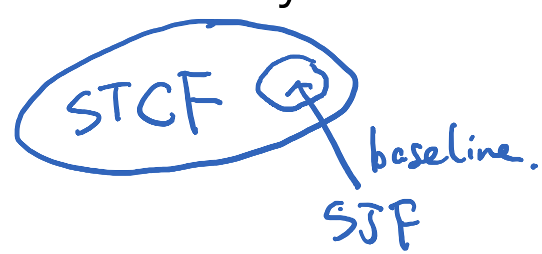
- Non-preemptive SJF is optimal if all the processes are ready simultaneously
  - Gives minimum average waiting time for a given set of processes

- What is the intuition behind the optimality of STCF?

Optimality of SJF and STCF

- Non-preemptive SJF is optimal if all the processes are ready simultaneously
  - Gives minimum average waiting time for a given set of processes

- What is the intuition behind the optimality of STCF?
  - A: STCF is optimal, considering a more realistic scenario where all the processes may be arriving at different times



Optimality of SJF and STCF

- Non-preemptive SJF is optimal if all the processes are ready simultaneously
  - Gives minimum average waiting time for a given set of processes

Q: What's the problem?

We don't exactly know how long a job would run!

- What is the intuition behind the optimality of SRTF?
  - A: SRTF is optimal, considering a more realistic scenario where all the processes may be arriving at different times

Estimating the Length of Next CPU Burst

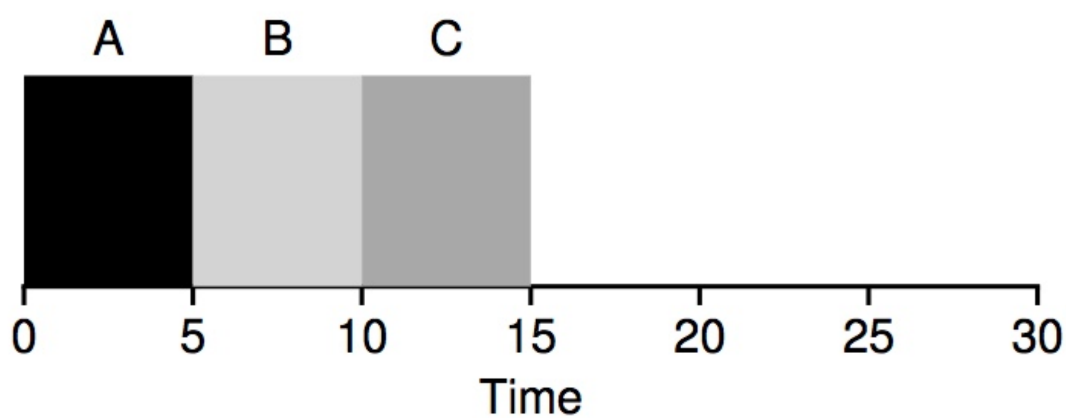
- Idea: Based on the observations in the recent past, we can try to predict
- Techniques such as exponential averaging are based on combining the observations in the past and our predictions using different weights
- Exponential averaging
  - t_n: actual length of the n^{th} CPU burst
  - z_{n+1}: predicted value for the next CPU burst
  - $z_{n+1} = k.t_n + (1-k).z_n$
  - Commonly, k is set to ½

Response Time

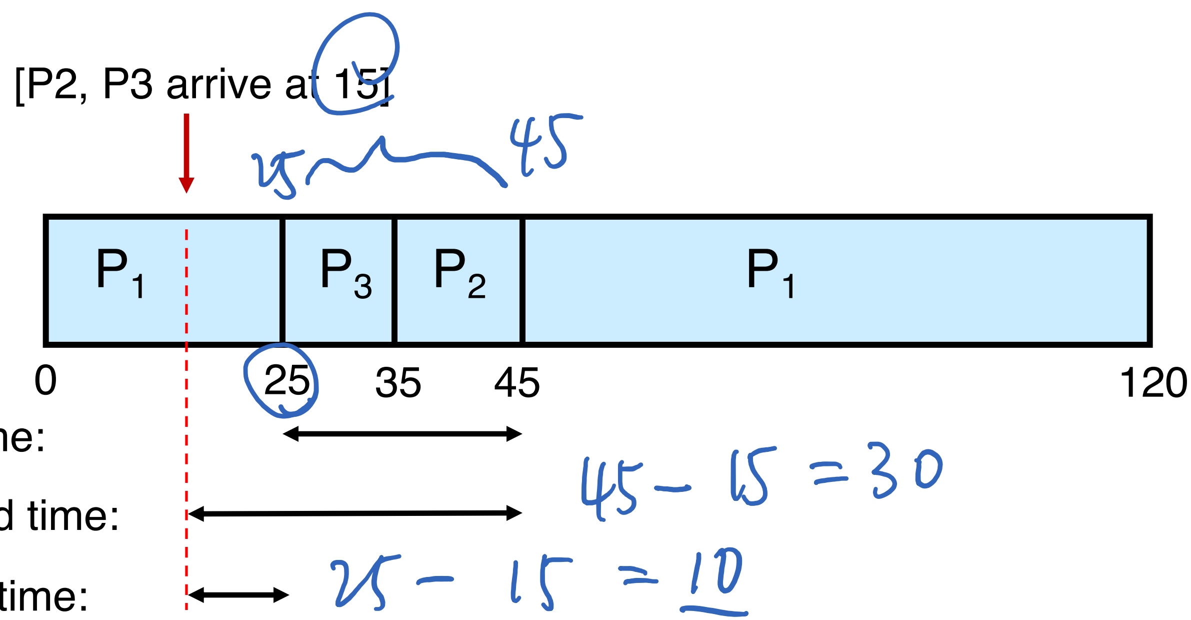
- Response time definition

$$T_{response} = T_{first\_run} - T_{arrival}$$

- SJF's average response time (all 3 jobs arrive at same time)
  - (0 + 5 + 10)/3 = 5



Waiting, Turnaround, Response

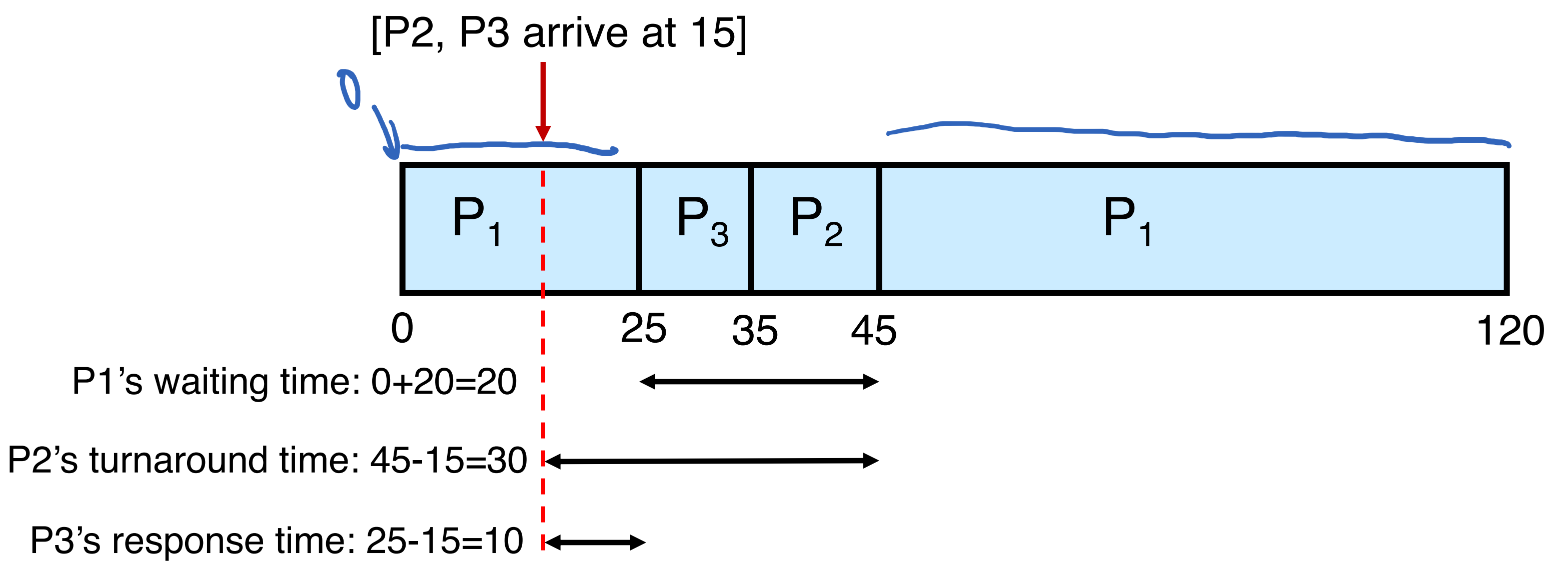


P1's waiting time:

P2's turnaround time:

P3's response time:

Waiting, Turnaround, Response



Q: What is P1's response time?

Round Robin (RR)

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 (no I/O)

4. The run-time of each job is known

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 (no I/O)~~
4. The run-time of each job is known

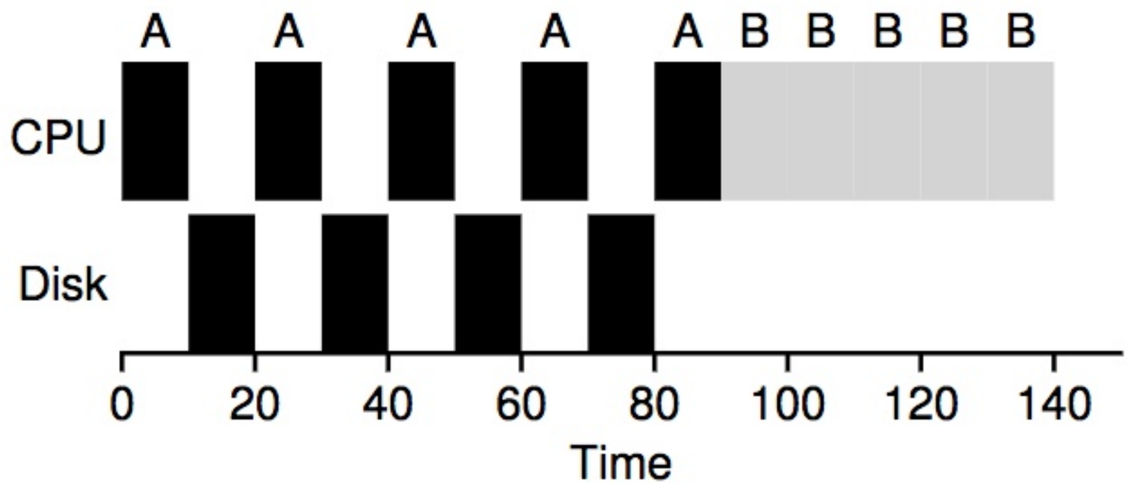
Extension to Multiple CPU & I/O Bursts

- When the process arrives, it will try to execute its first CPU burst
  - It will join the ready queue
  - The priority will be determined according to the underlying scheduling algorithm and considering only that specific (i.e. first) burst
- When it completes its first CPU burst, it will try to perform its first I/O operation (burst)
  - It will join the device queue
  - When that device is available, it will use the device for a time period indicated by the length of the first I/O burst.
- Then, it will re-join the ready queue and try to execute its second CPU burst
  - Its new priority may now change (as defined by its second CPU burst)!

Round Robin (RR)

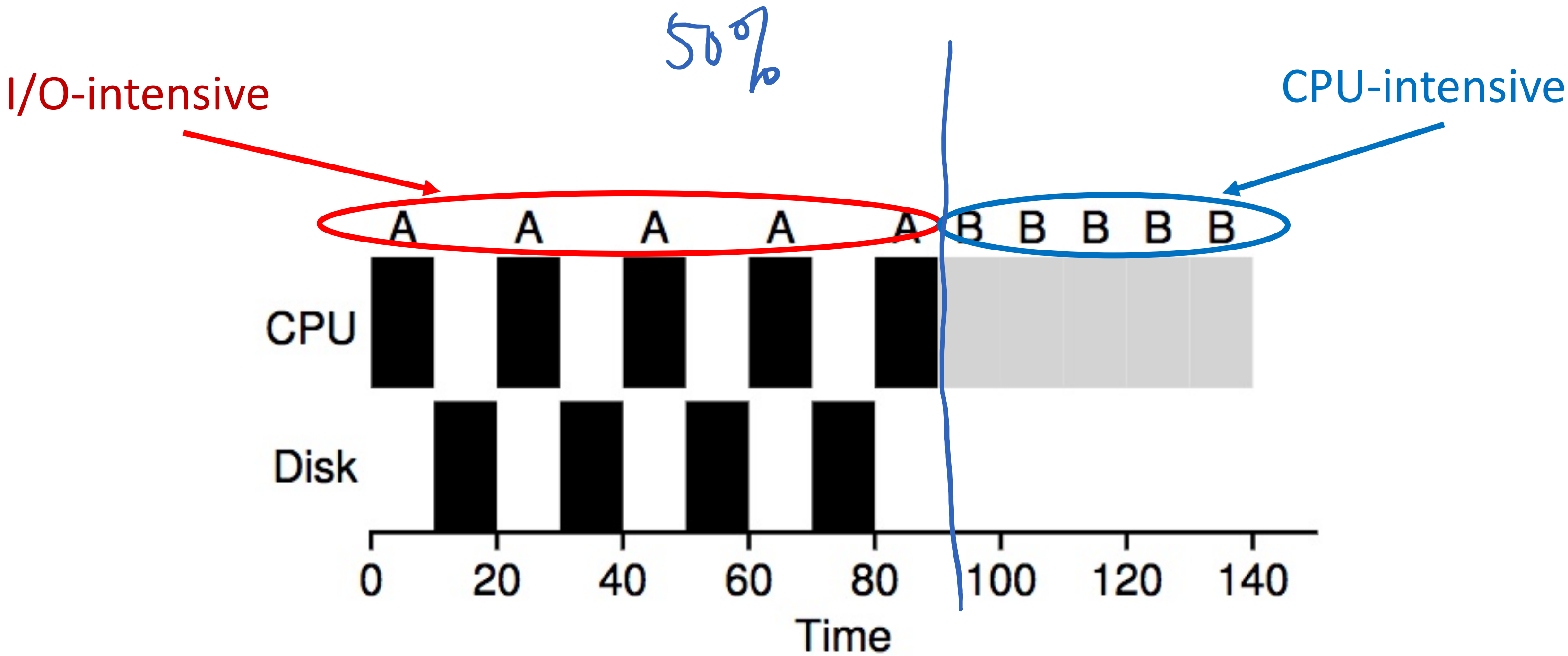
- Each process gets a small unit of CPU time (time quantum). After this time has elapsed, the process is preempted and added to the end of the ready queue
- Newly-arriving processes (and processes that complete their I/O bursts) are added to the end of the ready queue
- If there are n processes in the ready queue and the time quantum is q, then no process waits more than (n-1)q time units
- Performance
  - q large ⇒ FIFO
  - q small ⇒ Processor Sharing (The system appears to the users as though each of the n processes has its own processor running at the $(1/n)^{th}$ of the speed of the real processor)

Not I/O Aware



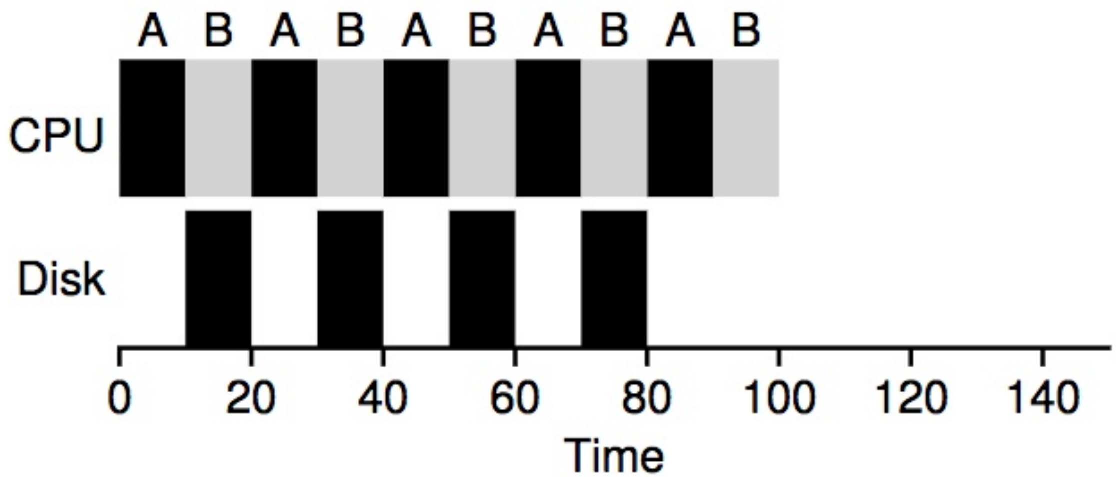
Poor use of resources

Not I/O Aware



Poor use of resources

I/O Aware (Overlap)



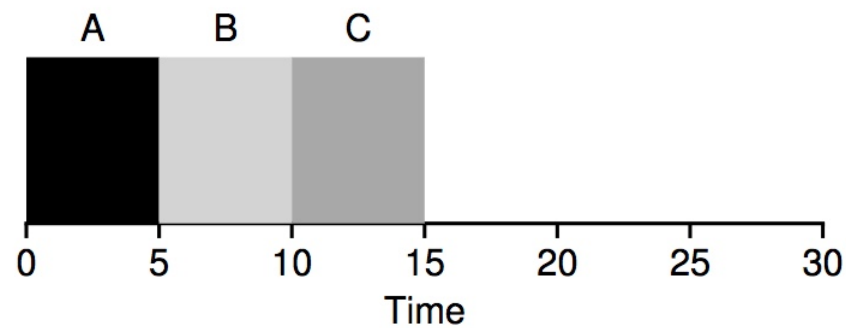
Overlap allows better use of resources!

RR

Process	Burst Time
A	5
B	5
C	5

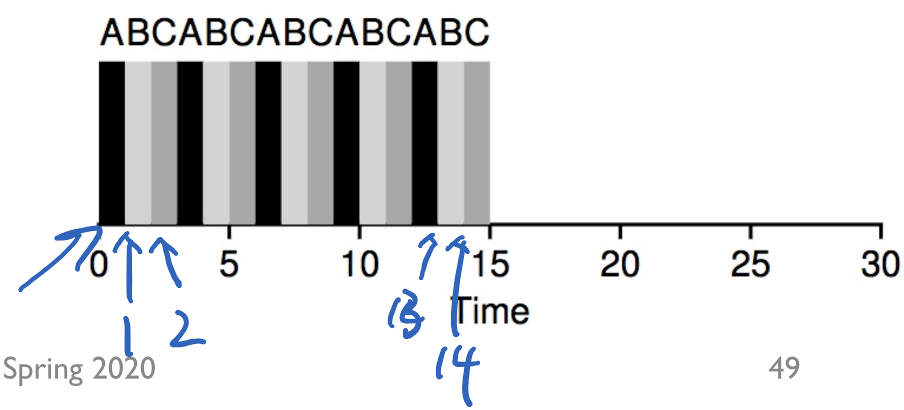
- SJF's average response time
  - (0 + 5 + 10) / 3 = 5

turnaround time: (5 + 10 + 15)/3 = 30/3 = 10



- RR's average response time (time quantum = 1)
  - (0 + 1 + 2) / 3 = 1

turnaround time = (13 + 14 + 15)/3 = 42/3 = 14



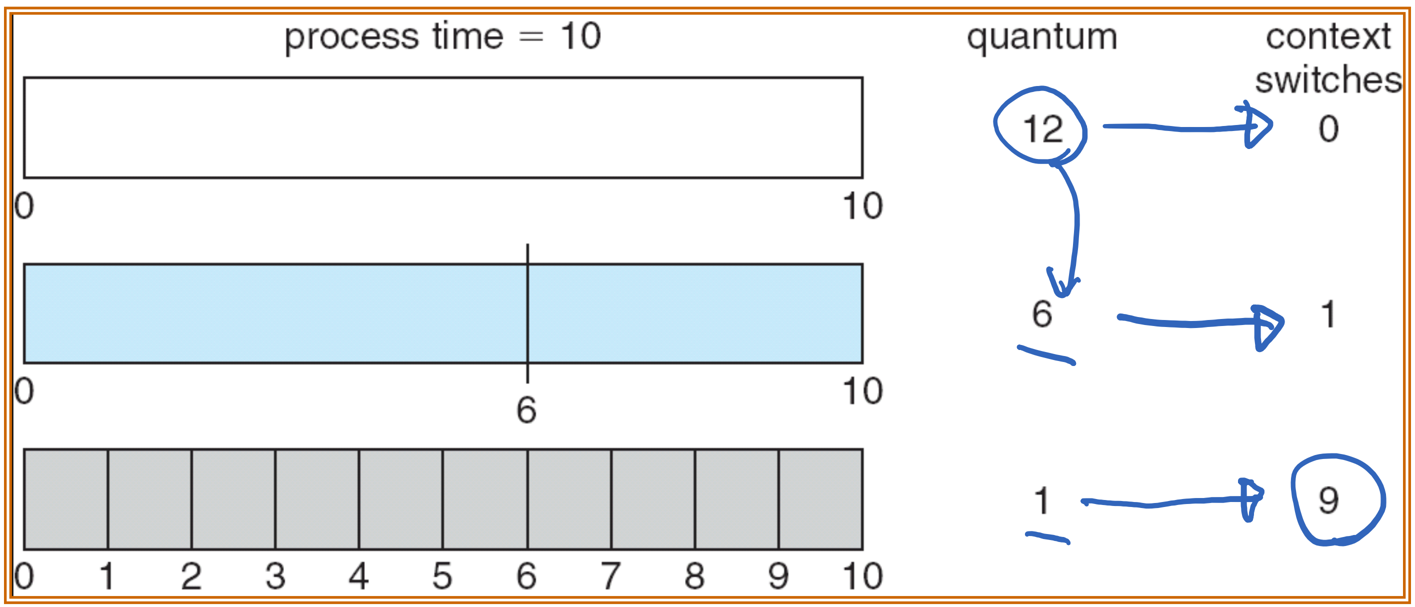
Tradeoff Consideration

- Typically, RR achieves higher average turnaround time than SJF, but better response time
  - Turnaround time only cares about when processes finish

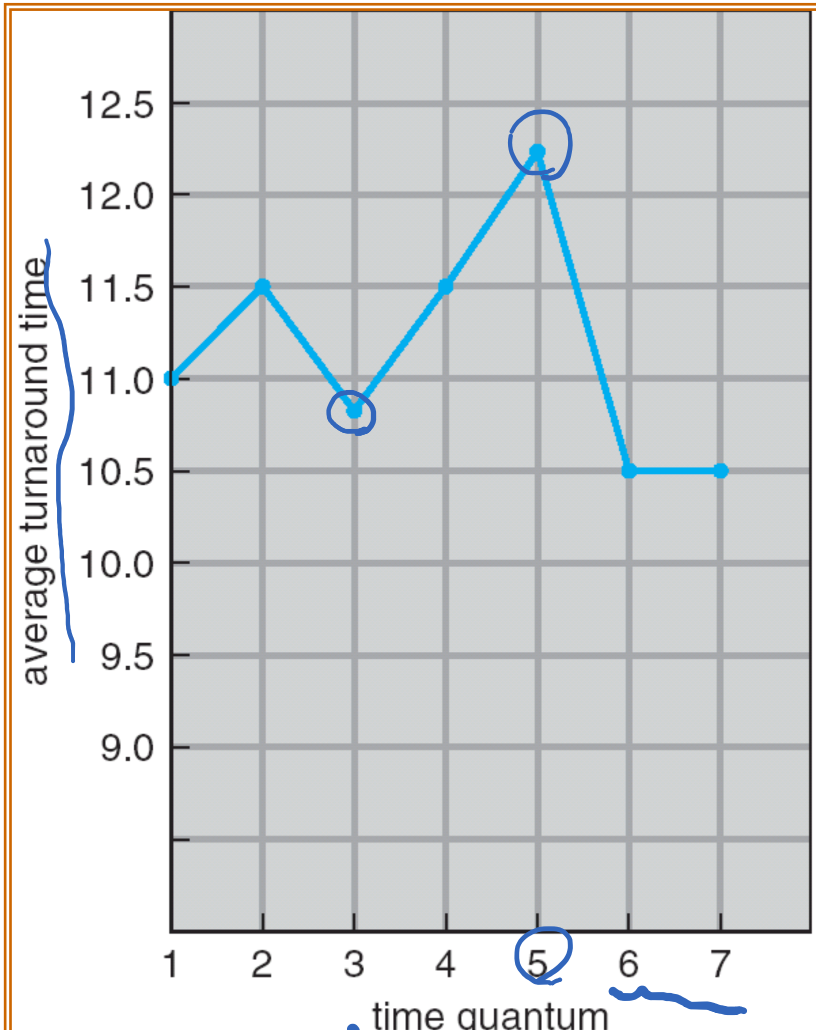
- RR is one of the worst policies
  - -IF- turnaround time is the metric

Choosing a Time Quantum

- The effect of quantum size on context-switching time must be carefully considered
- The time quantum must be large with respect to the context-switch time
- Turnaround time also depends on the size of the time quantum

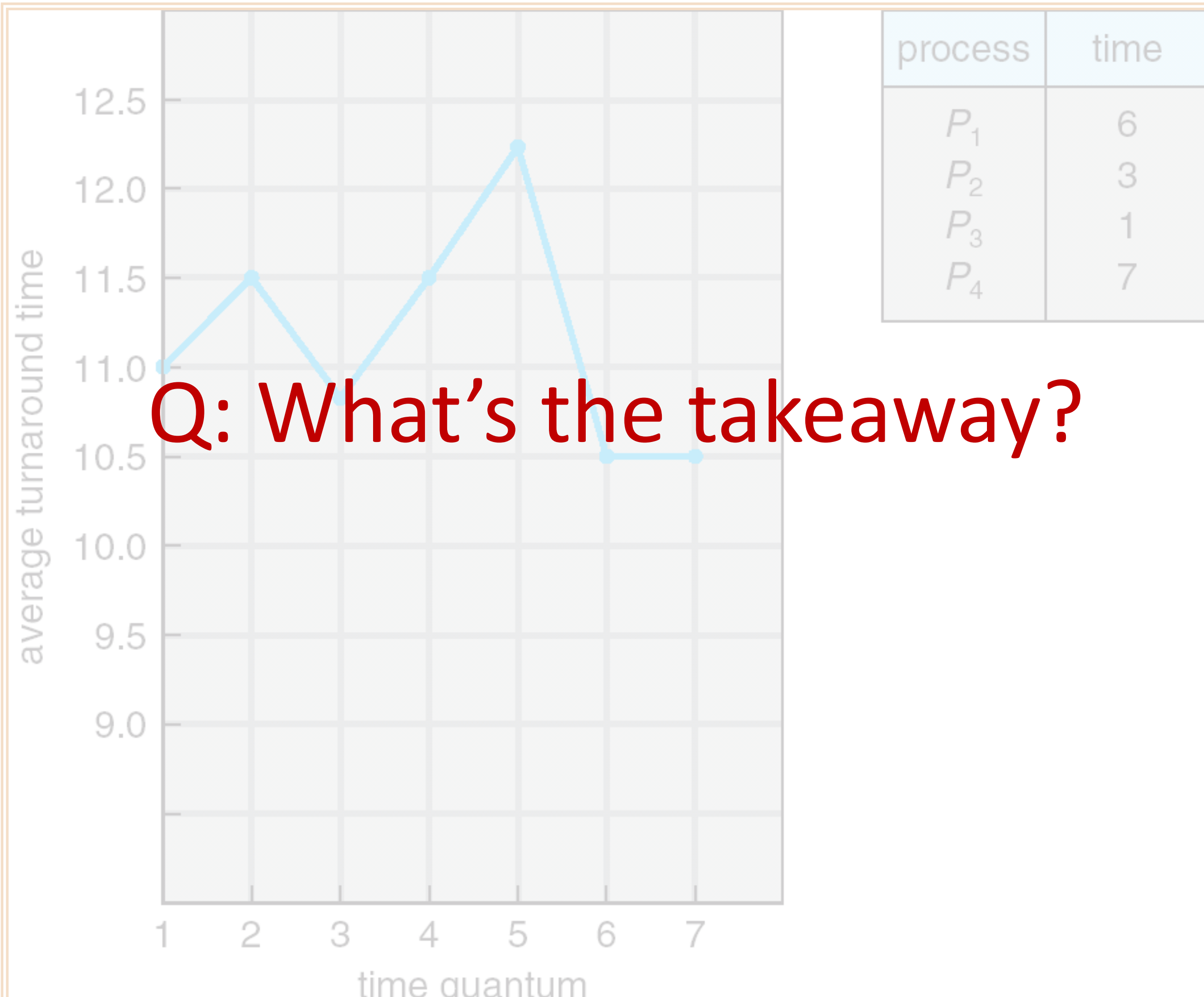


Time Quantum vs. Turnaround Time



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Time Quantum vs. Turnaround Time



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Q: What's the takeaway?

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 (no I/O)
4. The run-time of each job is known

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 (no I/O)~~

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

Priority-Based Scheduling

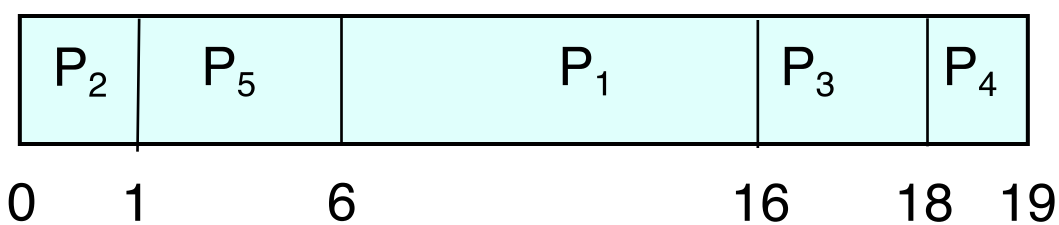
Priority-Based Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority
  - (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive

Example for Priority-Based Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2

Priority-Based Scheduling (cont.)

- Priority Assignment
  - Internal factors: timing constraints, memory requirements, the ratio of average I/O burst to average CPU burst …
  - External factors: Importance of the process, financial considerations, hierarchy among users …

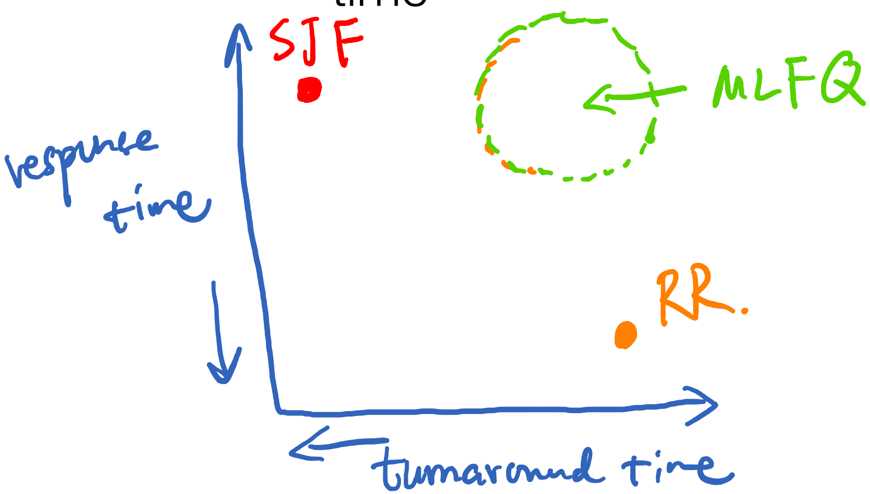
- Problem: Indefinite blocking (or starvation) – low priority processes may never execute

- One solution: Aging
  - As time progresses increase the priority of the processes that wait in the system for a long time

Multi-Level Feedback Queue (MLFQ)

Multi-Level Feedback Queue (MLFQ)

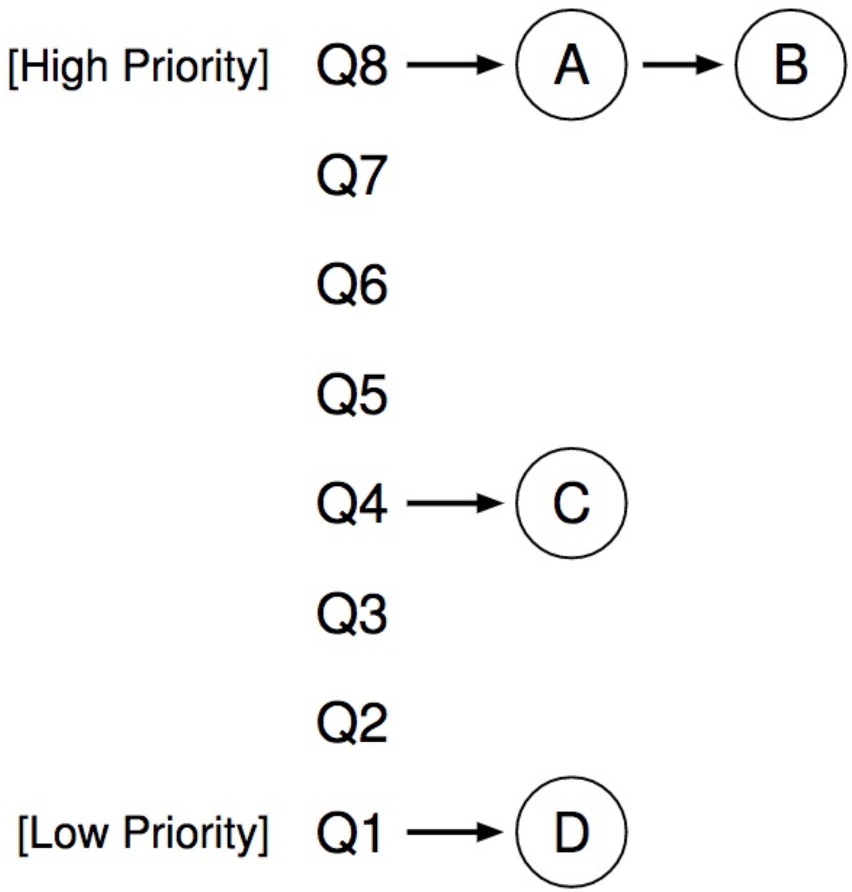
- Goals of MLFQ
  - Optimize turnaround time
    - In reality, SJF does not work since OS does not know how long a process will run
  - Minimize response time
    - Unfortunately, RR is really bad on optimizing turnaround time



MLFQ: Basics

- MLFQ maintains a number of queues (multi-level queue)
  - Each assigned a different priority level
  - Priority decides which process should run at a given time

MLFQ Example



How to know process type to set priority?

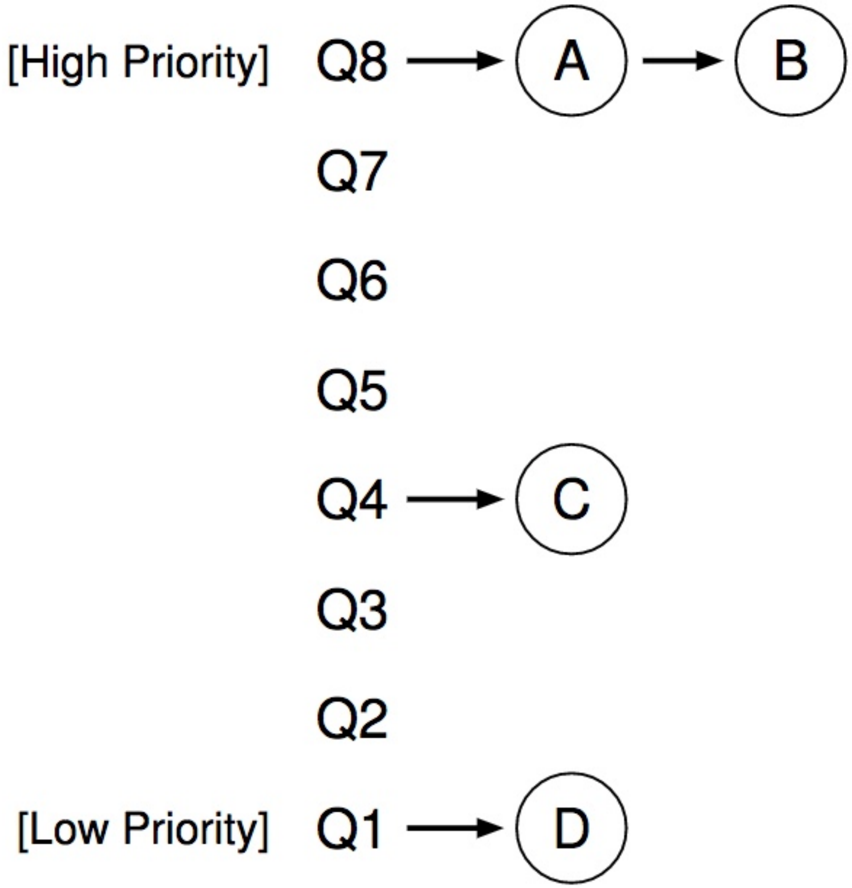
1. nice
2. history

How to Check Nice Values in Linux?

nice value

- `% ps ax -o pid,ni,cmd`

MLFQ Example



How to know process type to set priority?

1. nice
2. history

In this example, A and B are given high priority to run, while C and D may starve

MLFQ: Basic Rules

- MLFQ maintains a number of queues (multi-level queue)
  - Each assigned a different priority level
  - Priority decides which process should run at a given time

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

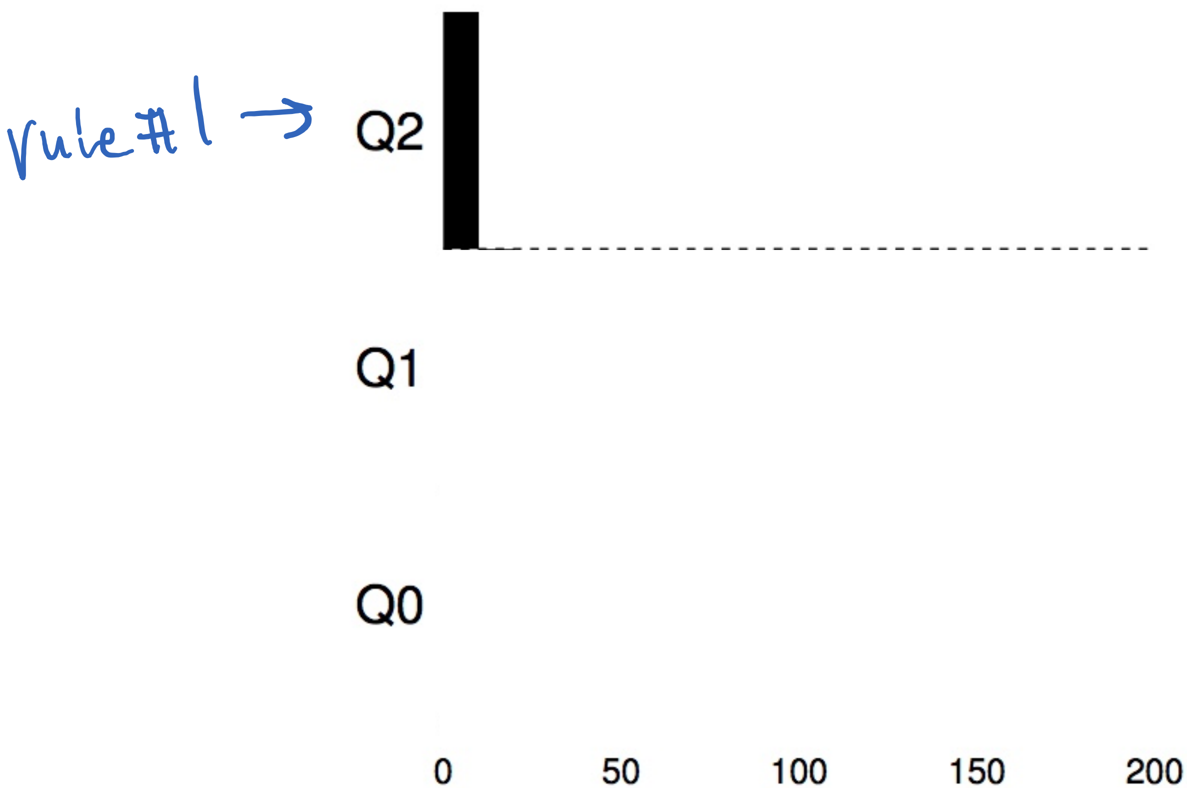
Attempt #1: Change Priority

- Workload
  1. • Interactive processes (many short-run CPU bursts)
  2. • Long-running processes (CPU-bound)
- Each time quantum = 10ms

- **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue).
- **Rule 4a:** If a job uses up an entire time slice while running, its priority is *reduced* (i.e., it moves down one queue).
- **Rule 4b:** If a job gives up the CPU before the time slice is up, it stays at the *same* priority level.

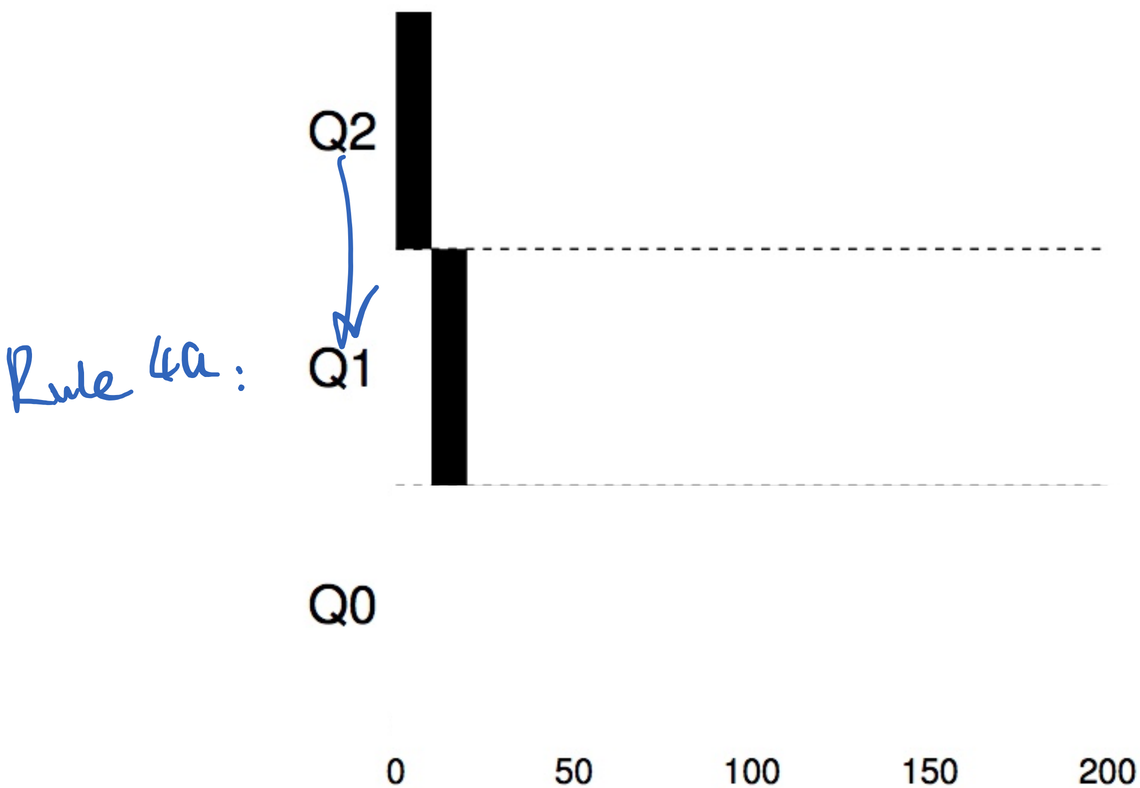
Example 1: One Single Long-Running Process

- A process enters at highest priority (time quantum = 10ms)



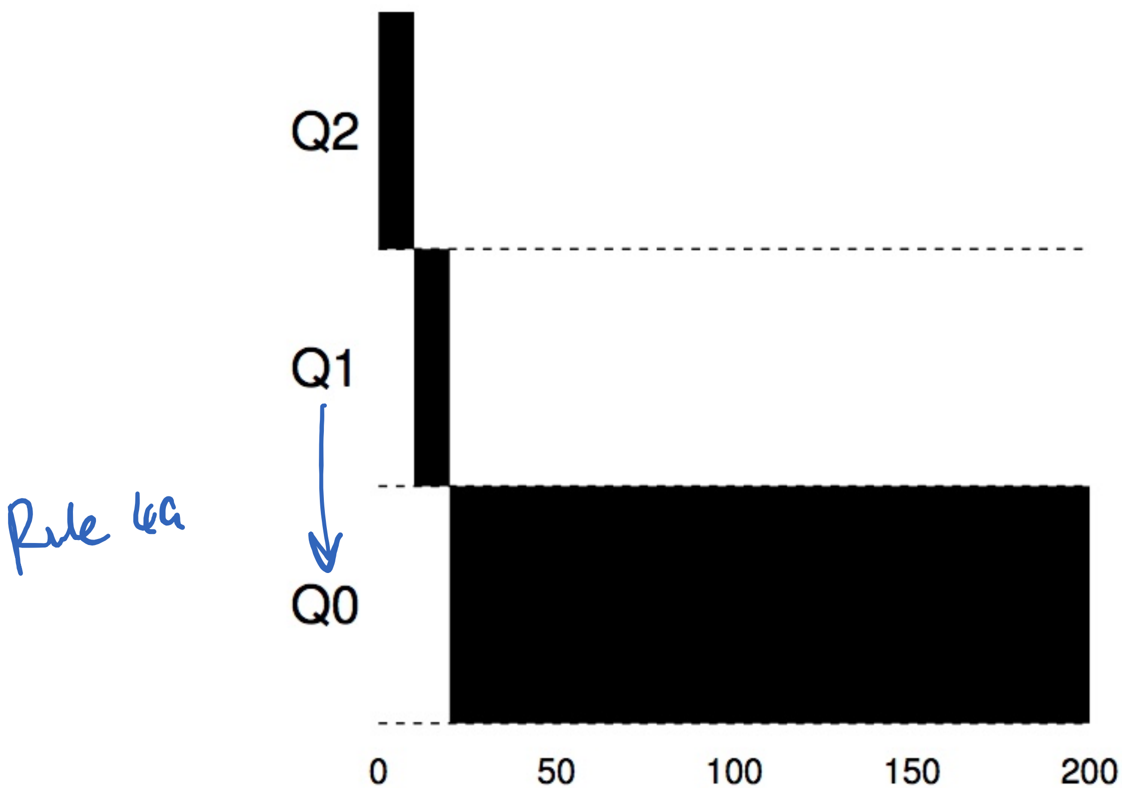
Example 1: One Single Long-Running Process

- A process enters at highest priority (time quantum = 10ms)



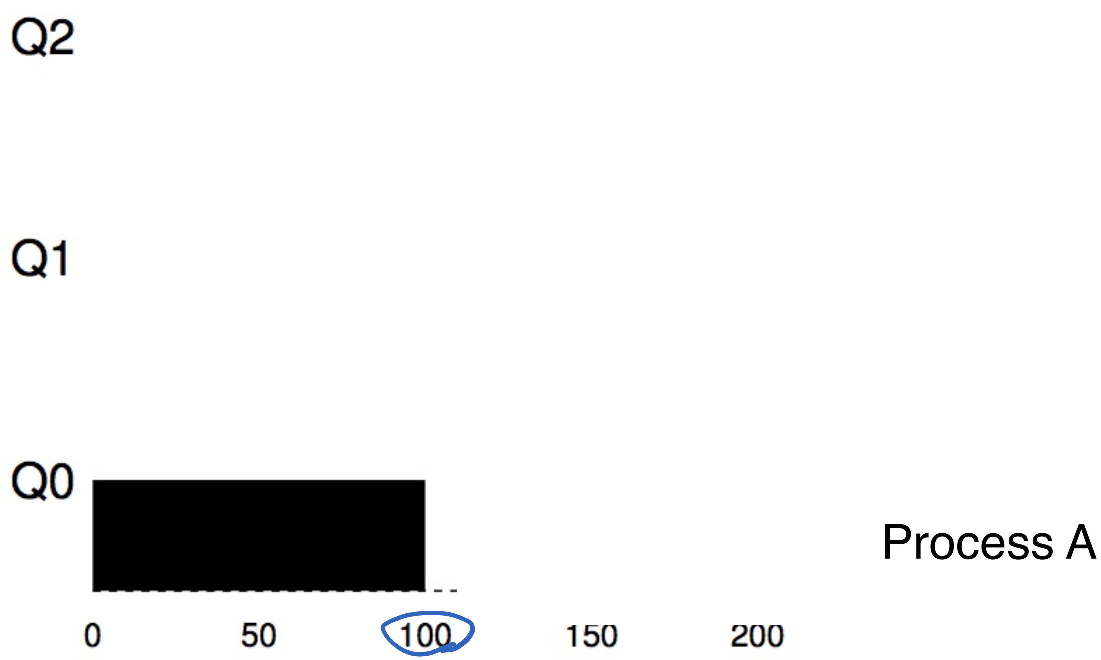
Example 1: One Single Long-Running Process

- A process enters at highest priority (time quantum = 10ms)



Example 2: Along Came a Short-Running Process

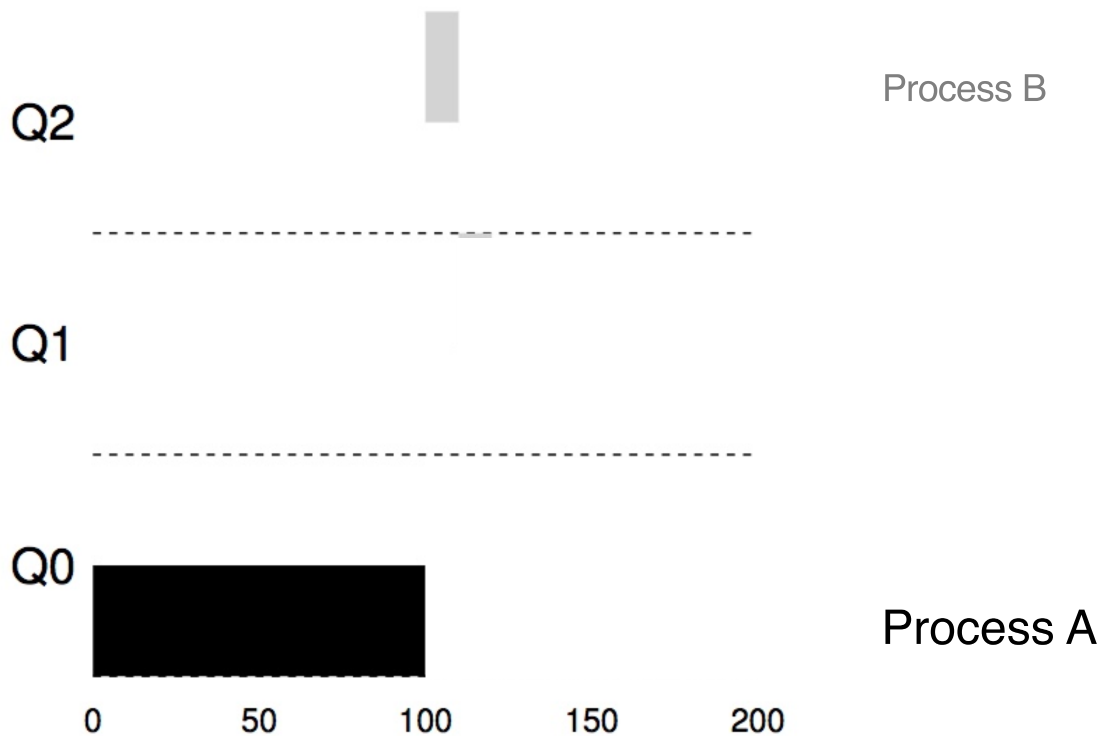
- Process A: long-running process (start at 0)



Example 2: Along Came a Short-Running Process

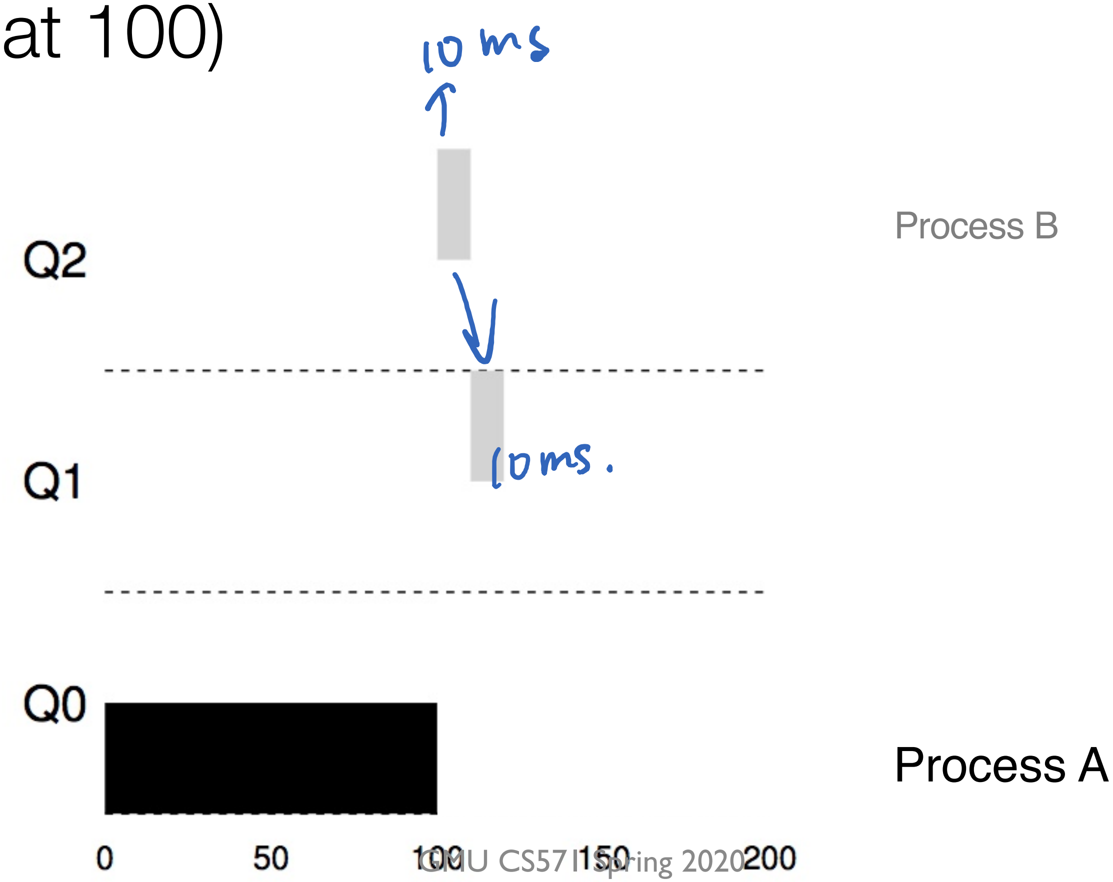
- Process A: long-running process (start at 0)
- Process B: short-running interactive process (start at 100)

Rule #1



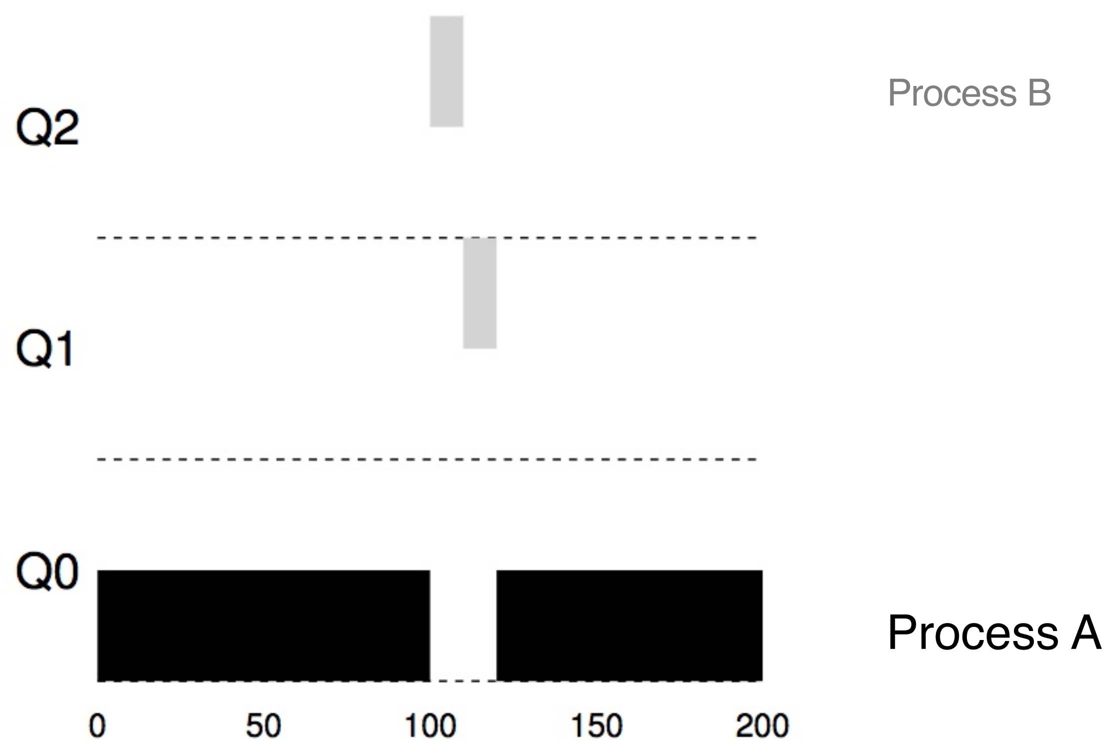
Example 2: Along Came a Short-Running Process

- Process A: long-running process (start at 0)
- Process B: short-running interactive process (start at 100)



Example 2: Along Came a Short-Running Process

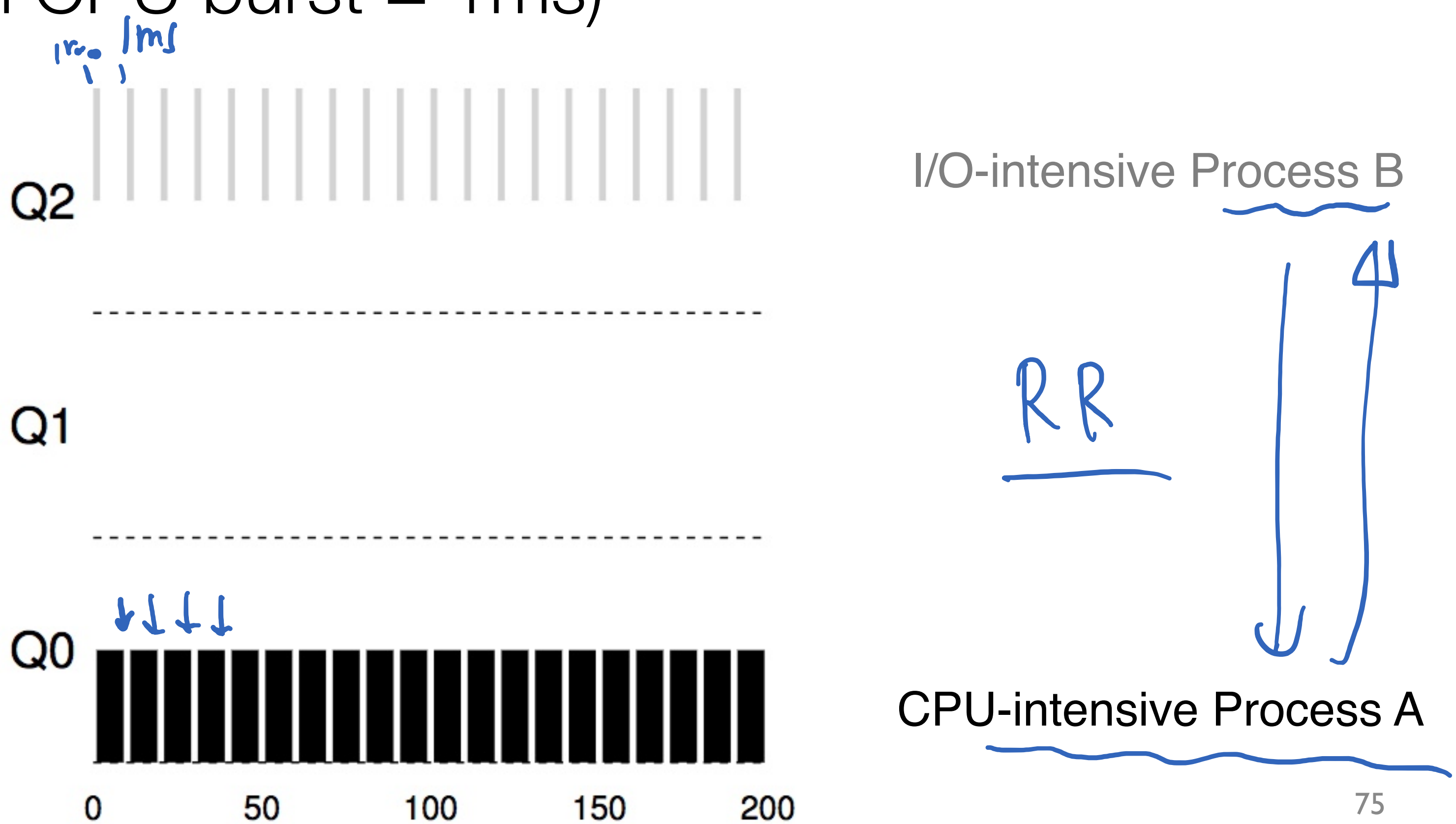
- Process A: long-running process (start at 0)
- Process B: short-running interactive process (start at 100)



Example 3: What about I/O?

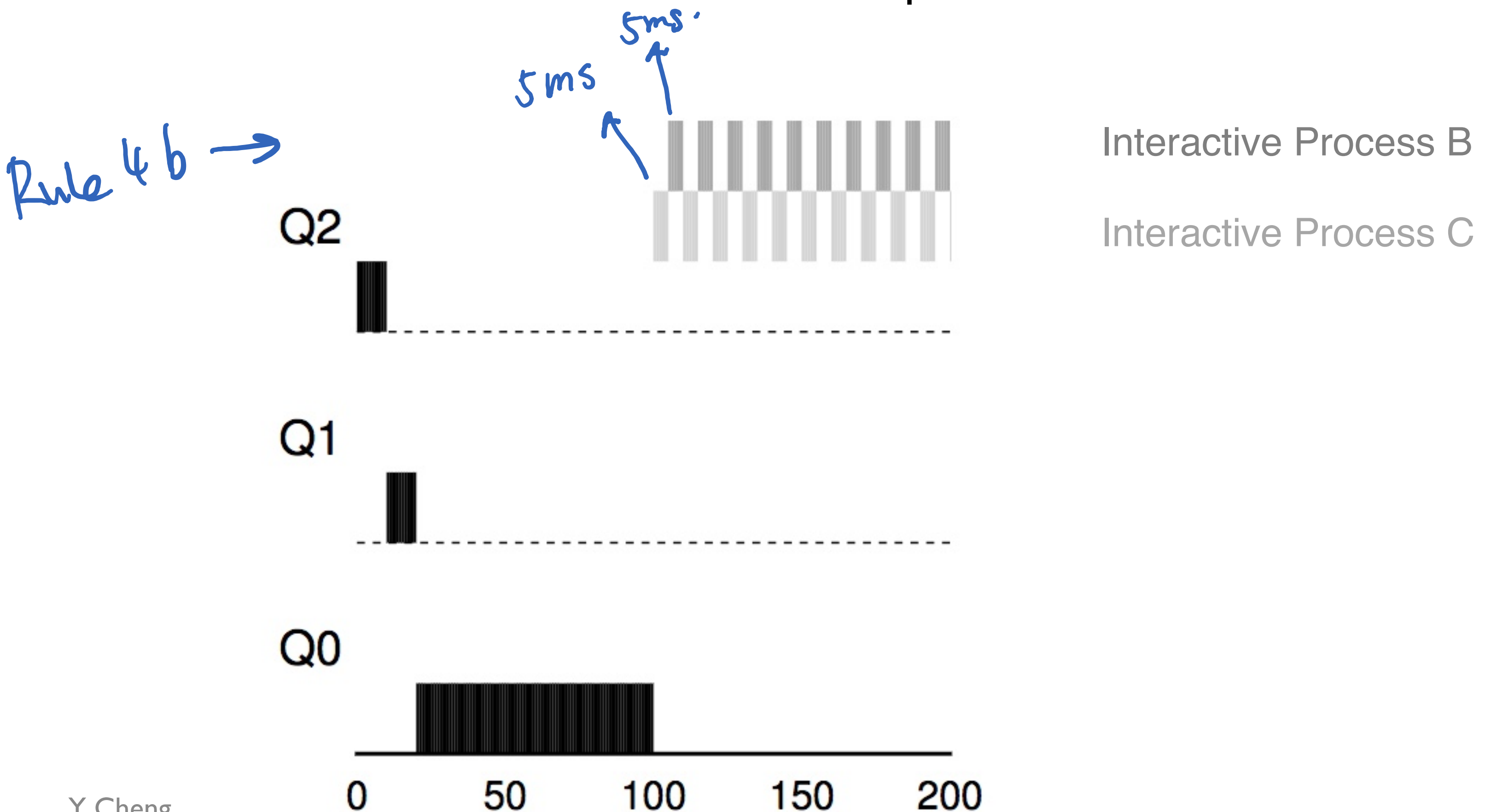
- Process A: long-running process
- Process B: I/O-intensive interactive process (each CPU burst = 1ms)

Rule 4b



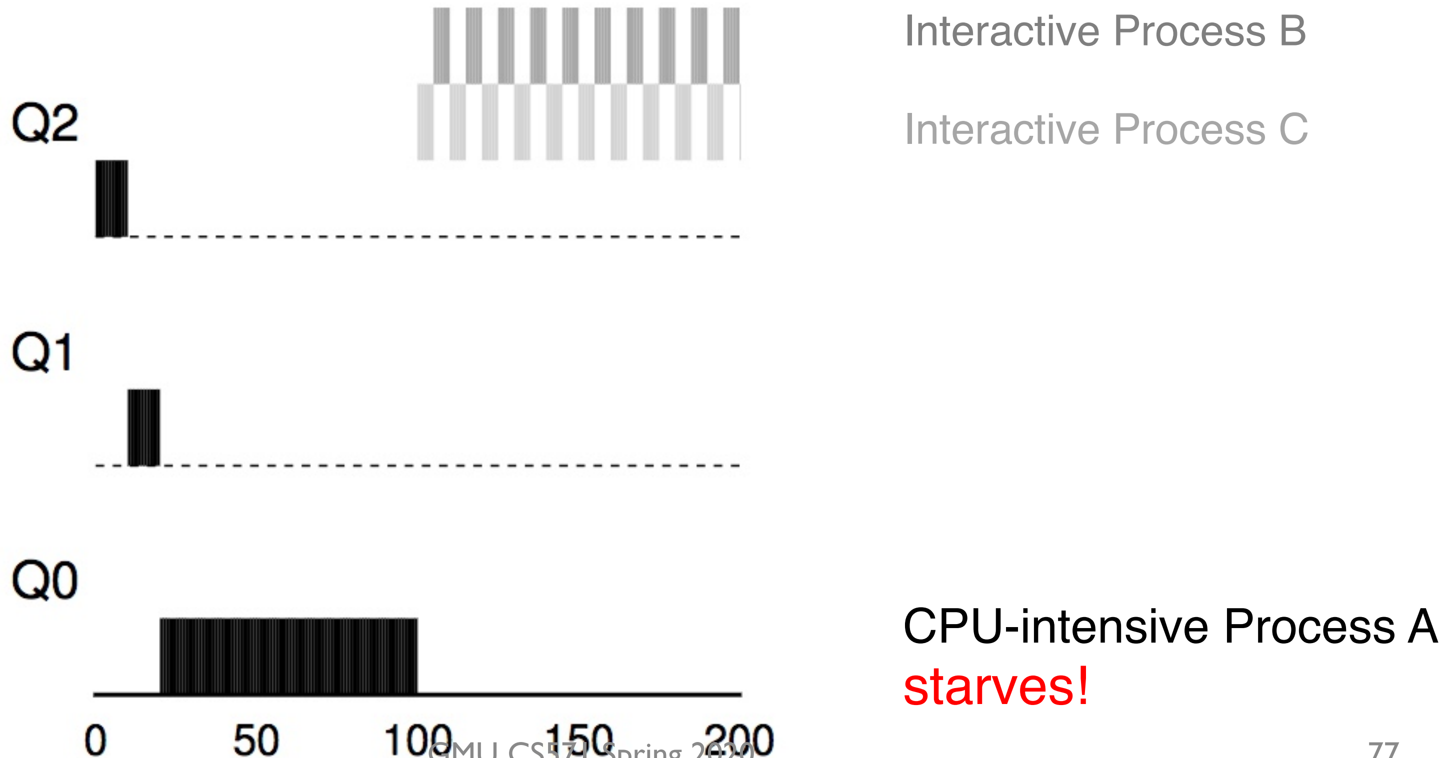
Example 4: What's the Problem?

- Process A: long-running process
- Process B + C: Interactive process



Example 4: What's the Problem?

- Process A: long-running process
- Process B + C: Interactive process

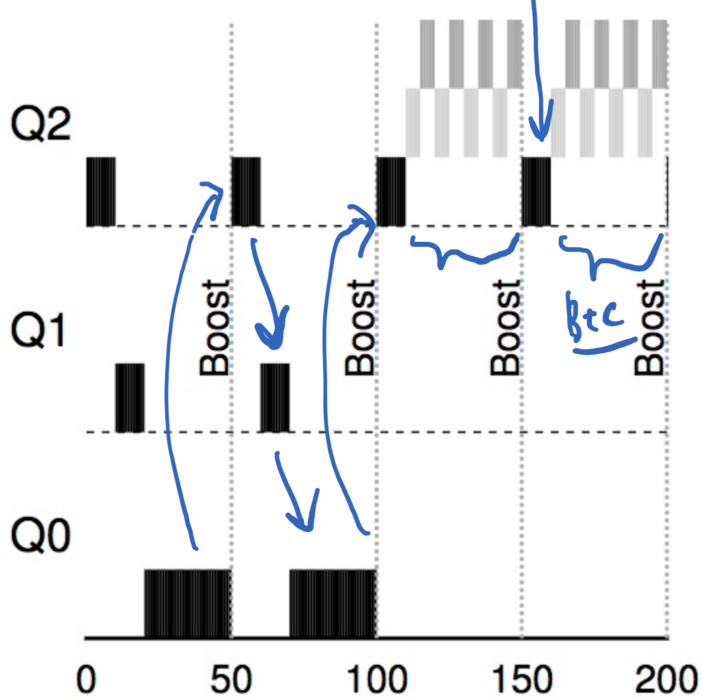


Attempt #2: Priority Boost

- Simple idea: Periodically boost the priority of all processes

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

configurable



Interactive Process B

Interactive Process C

S == 50 ms.

CPU-intensive Process A proceeds!

Tuning MLFQ

- MLFQ scheduler is defined by many parameters:
  - Number of queues
  - Time quantum of each queue
  - How often should priority be boosted?
  - A lot more…
- The scheduler can be configured to match the requirements of a specific system
  - Challenging and requires experience

Lottery Scheduling

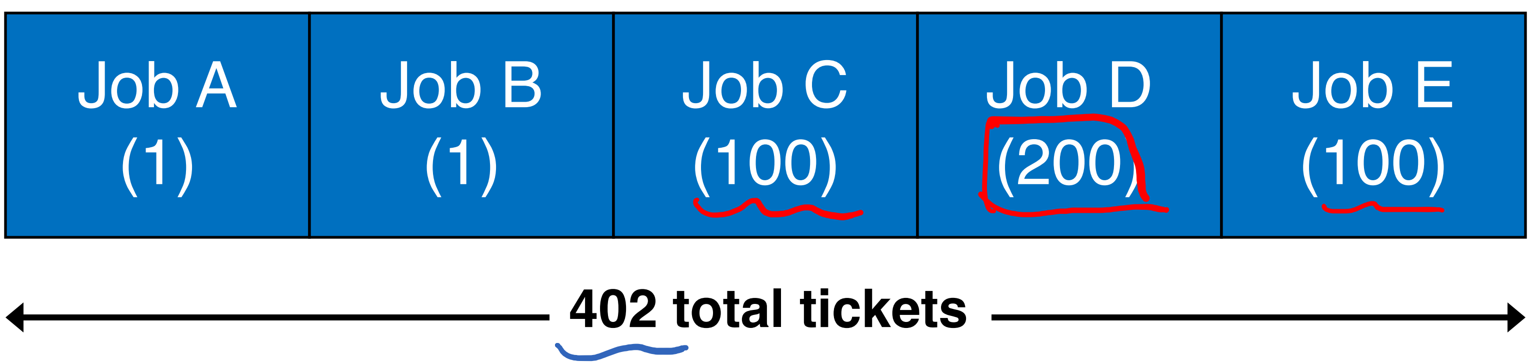
Lottery Scheduling

- Goal: Proportional share
  - One of the fair-share schedulers
- Approach
  - Gives processes lottery tickets
  - Whoever wins runs
  - Higher priority → more tickets

Lottery Code

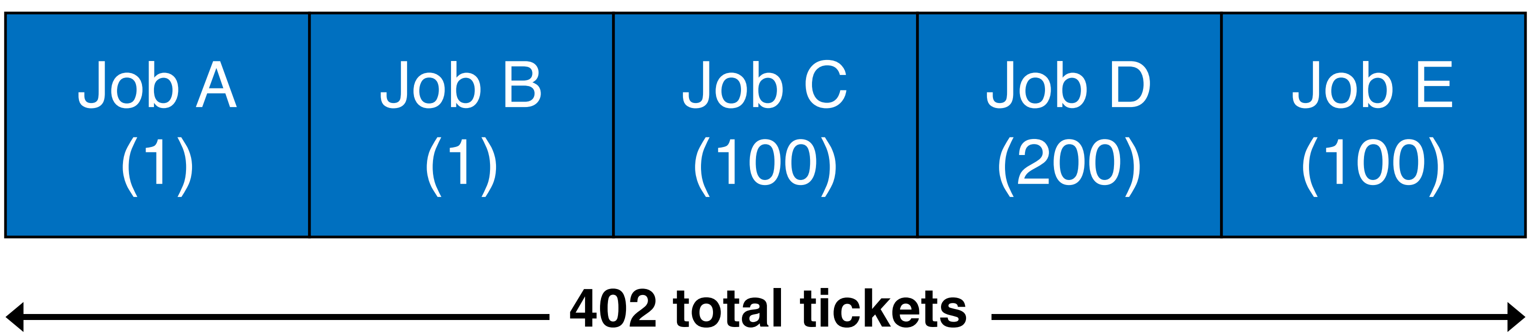
```
// counter: used to track if we've found the winner yet
int counter = 0;

// winner: use some call to a random number generator to
//         get a value, between 0 and the total # of tickets
int winner = getrandom(0, totaltickets);

// current: use this to walk through the list of jobs
node_t *current = head;

// loop until the sum of ticket values is > the winner
while (current) {
    counter = counter + current->tickets;
    if (counter > winner)
        break; // found the winner
    current = current->next;
}
// 'current' is the winner: schedule it...
```

Lottery Scheduling Example

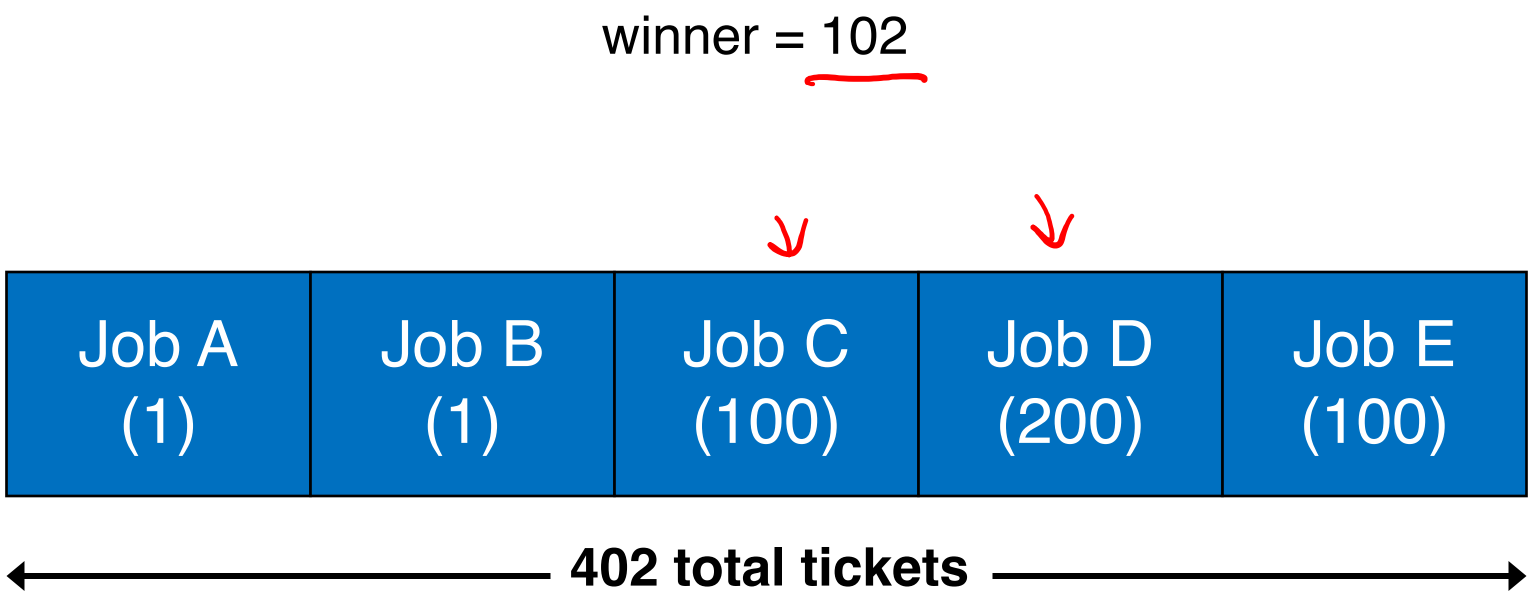


Lottery Scheduling Example

winner = random(402)



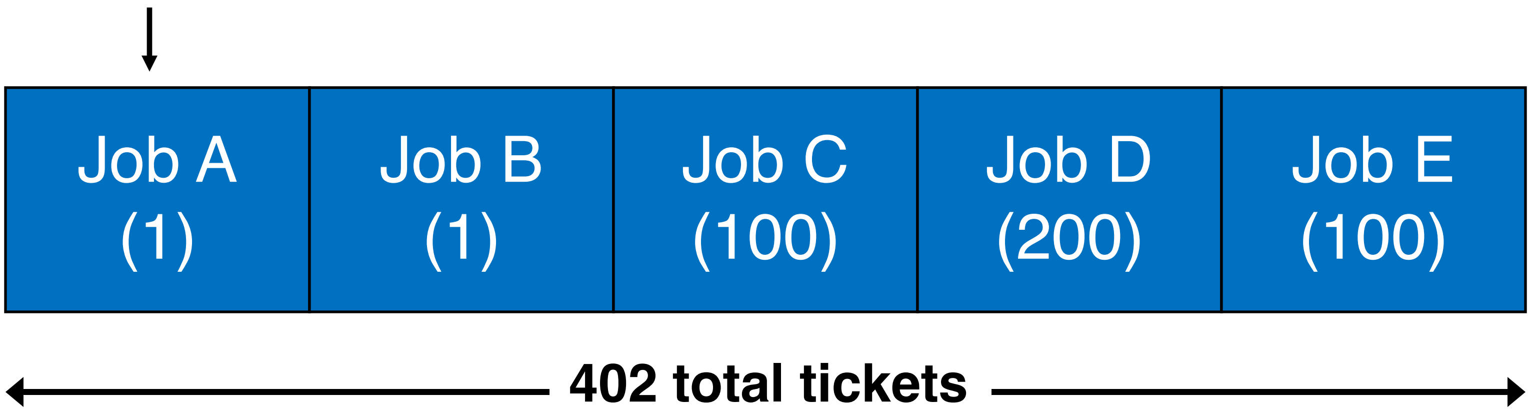
Lottery Scheduling Example



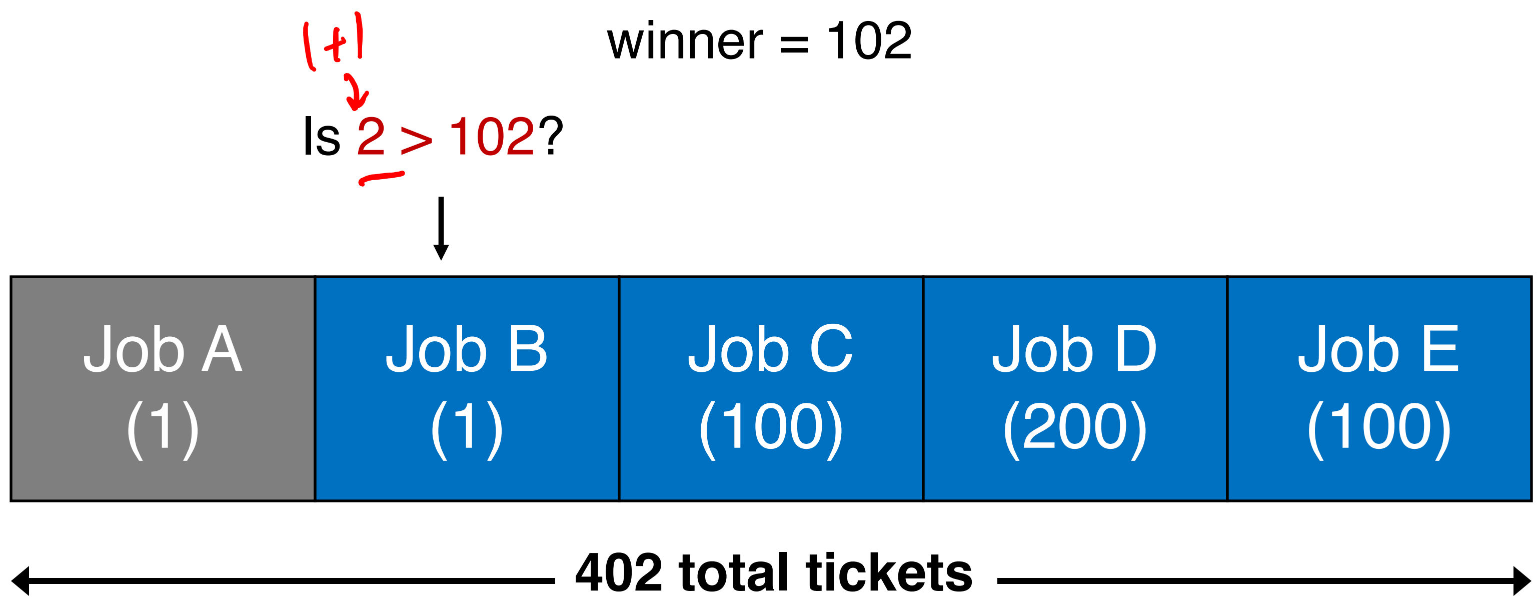
Lottery Scheduling Example

winner = 102

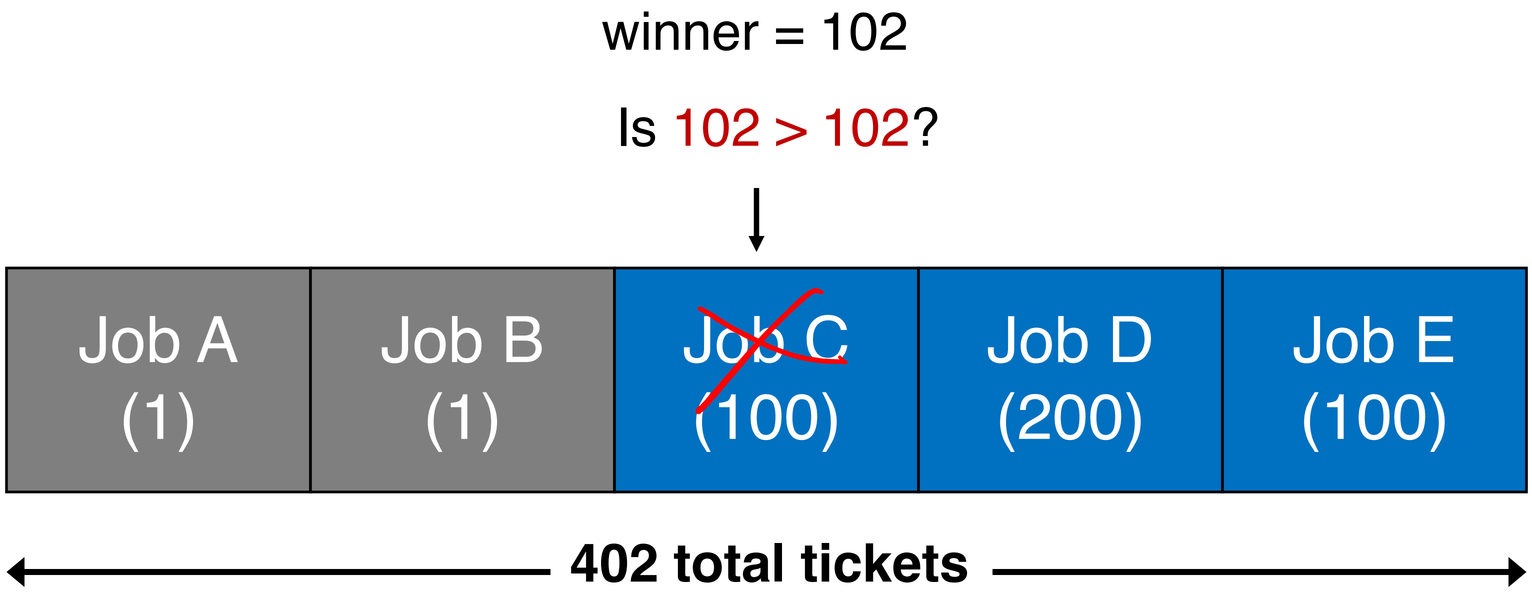
Is 1 > 102?



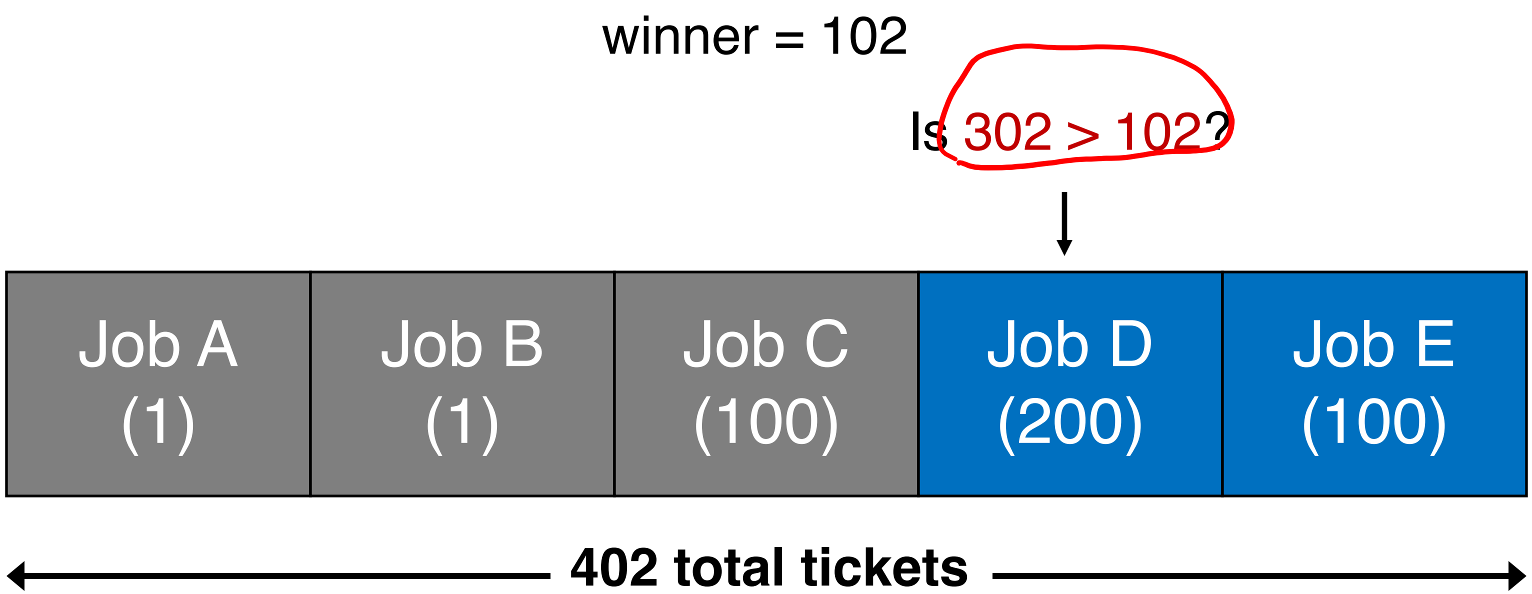
Lottery Scheduling Example



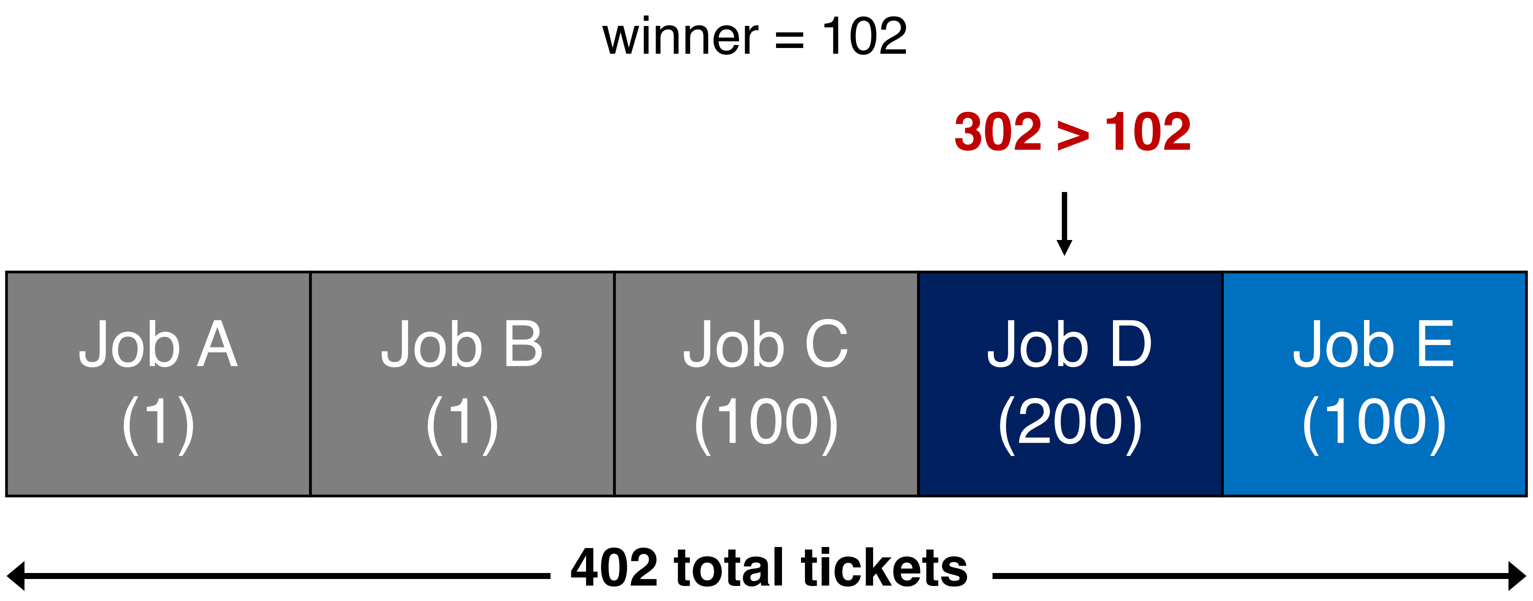
Lottery Scheduling Example



Lottery Scheduling Example



Lottery Scheduling Example



OS picks Job D to run!