

# Operating System

## 8. Scheduling: The Multi-Level Feedback Queue

## 8. Scheduling: The 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



# 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.~~

# History

- Use past behavior of process to predict future behavior
  - Common technique in systems
- Processes alternate between **I/O** and **CPU** work
- Guess how CPU burst (job) will behave based on past CPU bursts (jobs) of this process

# Multi-Level Feedback Queue (MLFQ)

- A Scheduler that learns from the past to predict the future.
- Objective:
  - Optimize **turnaround time** → Run shorter jobs first
  - Minimize **response time** without *a priori knowledge of job length*.

# MLFQ: Basic Rules

- MLFQ has a number of distinct **queues**.
  - Each queue is assigned a *different priority level*.
- A job that is ready to run is on a single queue.
  - A job **on a higher queue** is chosen to run.
  - Use round-robin scheduling among jobs in the same queue

**Rule 1:** If  $\text{Priority}(A) > \text{Priority}(B)$ , A runs (B doesn't).

**Rule 2:** If  $\text{Priority}(A) = \text{Priority}(B)$ , A & B run in RR.

## MLFQ: How to Change Priority

- MLFQ varies the priority of a job based on **its observed behavior**.

**Rule 3:** When a job enters the system, it is placed at the highest priority

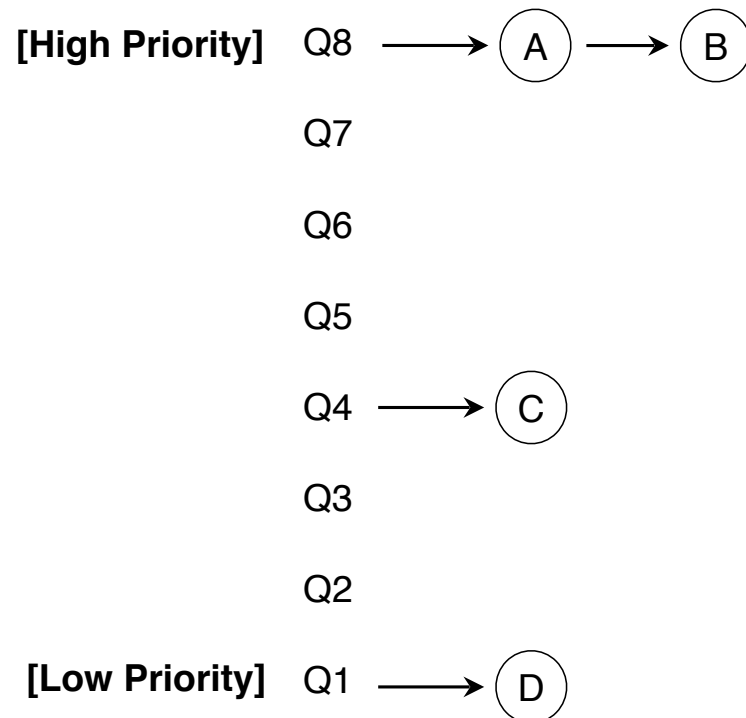
**Rule 4a:** If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down on queue).

**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

# MLFQ: Overview

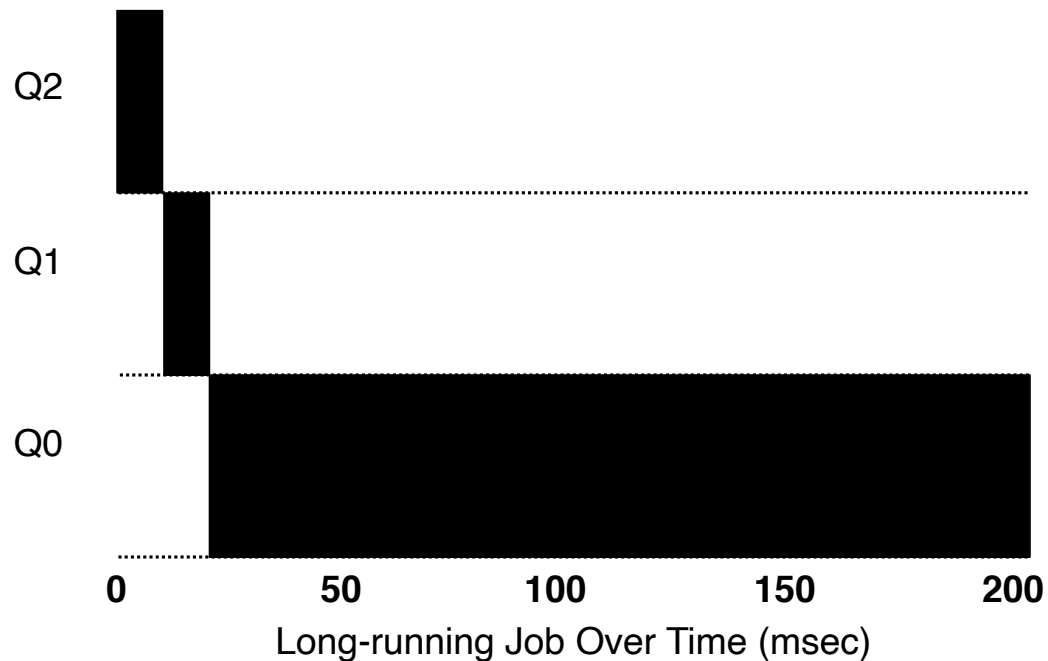
- A job repeatedly relinquishes the **CPU** while waiting **IOs**  
→ Keep its priority *high*
- A job uses the **CPU** intensively for long periods of time  
→ *Reduce* its priority.





### Example 1: A Single Long-Running Job

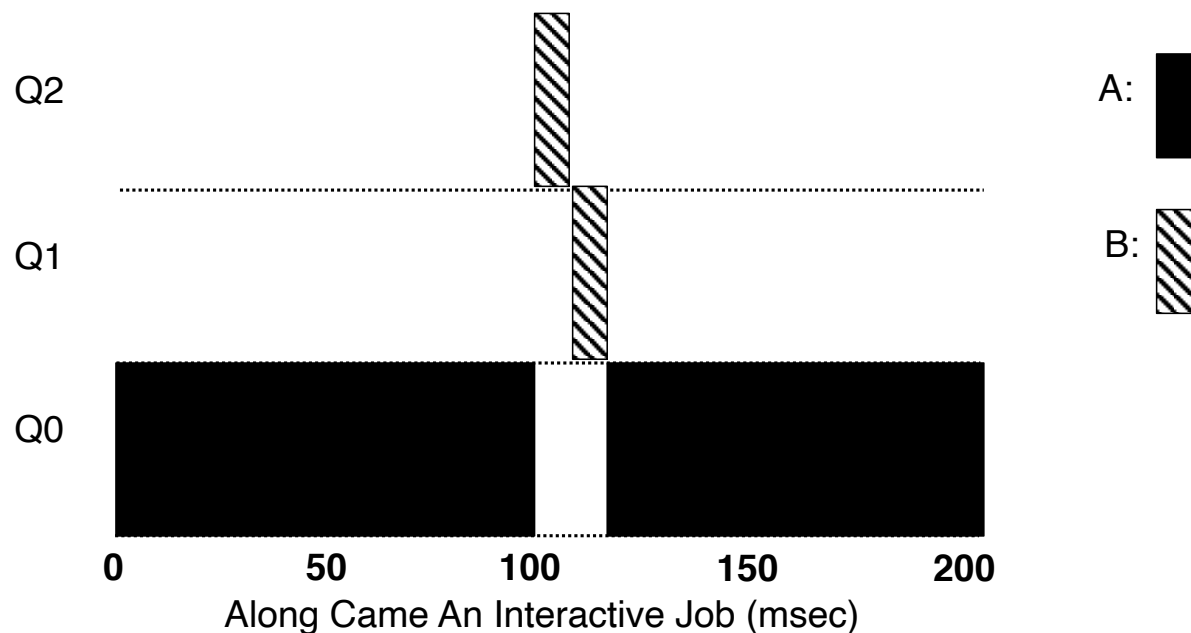
- A three-queue scheduler with time slice 10ms



## Example 2: Along Came a Short Job

- Assumption:

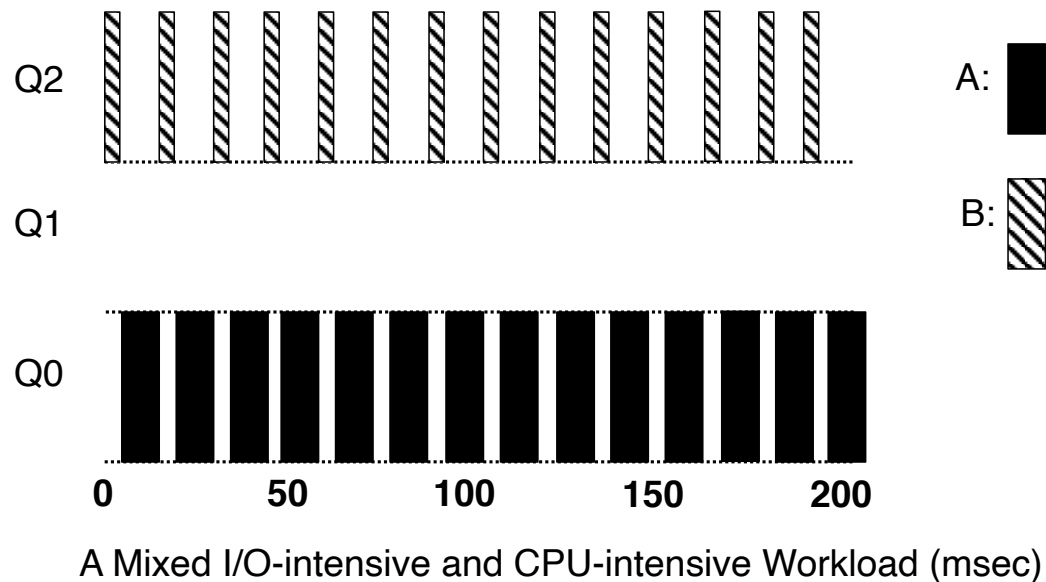
- **Job A:** A long-running CPU-intensive job
- **Job B:** A short-running interactive job (20ms runtime)
- A has been running for some time, and then B arrives at time  $T=100$ .



### 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**



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

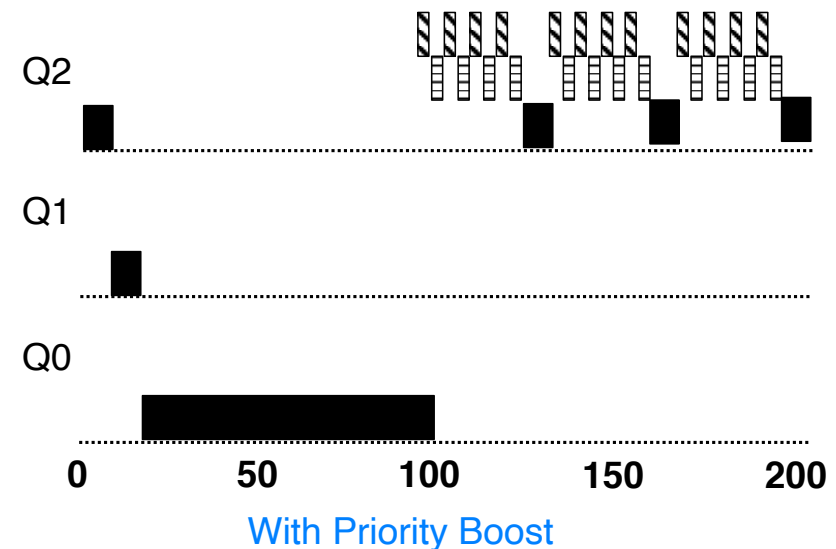
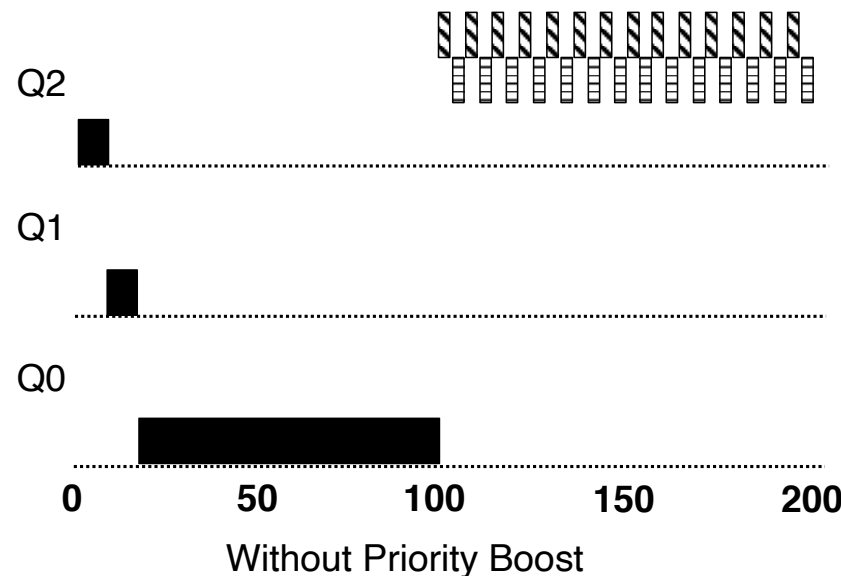
- After running 99% of a time slice, issue an I/O operation.
- The job gain a higher percentage of CPU time.
- A program may **change its behavior** over time.
  - **CPU** bound process → **I/O** bound process

# The Priority Boost

## ■ Example:

- A long-running job(A) with two short-running interactive job(B, C)

A:  B:  C: 

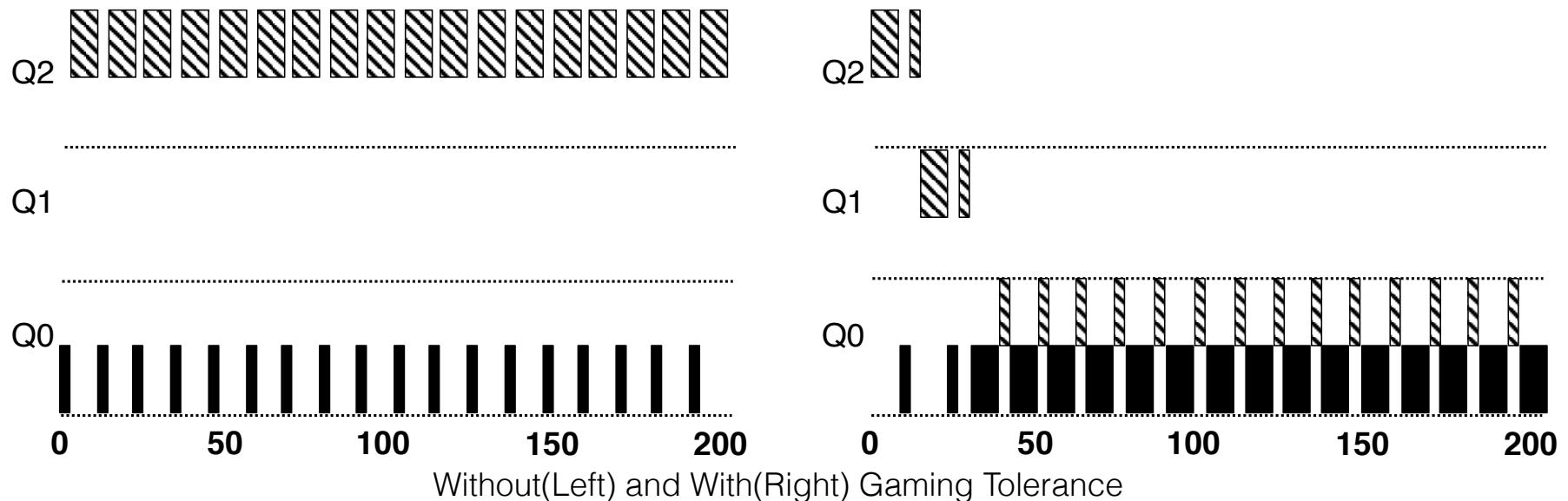


**Rule 5:** After some time period  $S$ , move all the jobs in the system to the topmost queue.

# Better Accounting

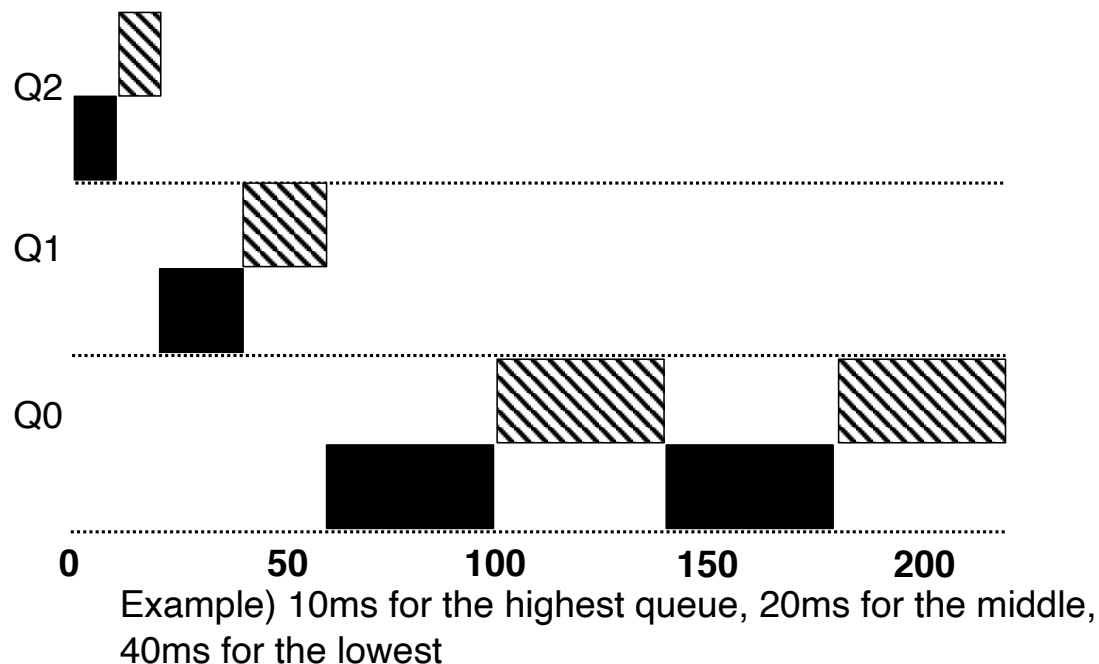
- How to prevent gaming of our scheduler?

**Rule 4 (Rewrite 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).



# Tuning MLFQ And Other Issues

- The high-priority queues → Short time slices
  - E.g., 10 or fewer milliseconds
- The Low-priority queue → Longer time slices
  - E.g., 100 milliseconds



Lower Priority,  
Longer Quanta

# MLFQ: Summary

**Rule 1:** If  $\text{Priority}(A) > \text{Priority}(B)$ , A runs (B doesn't).

**Rule 2:** If  $\text{Priority}(A) = \text{Priority}(B)$ , 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.



# Slightly Different Implementation

### ■ **A new variant of MLFQ**

- Only move a task to the next lower priority queue when it is preempted
- If a waiting (I/O) task becomes ready, sort it back at the top of the queue it was before.

### ■ **One or more events might happen at time $t$**

- New tasks arrive in system
- Time-Slot of running task is over
  - Waiting (E/A) task becomes ready again

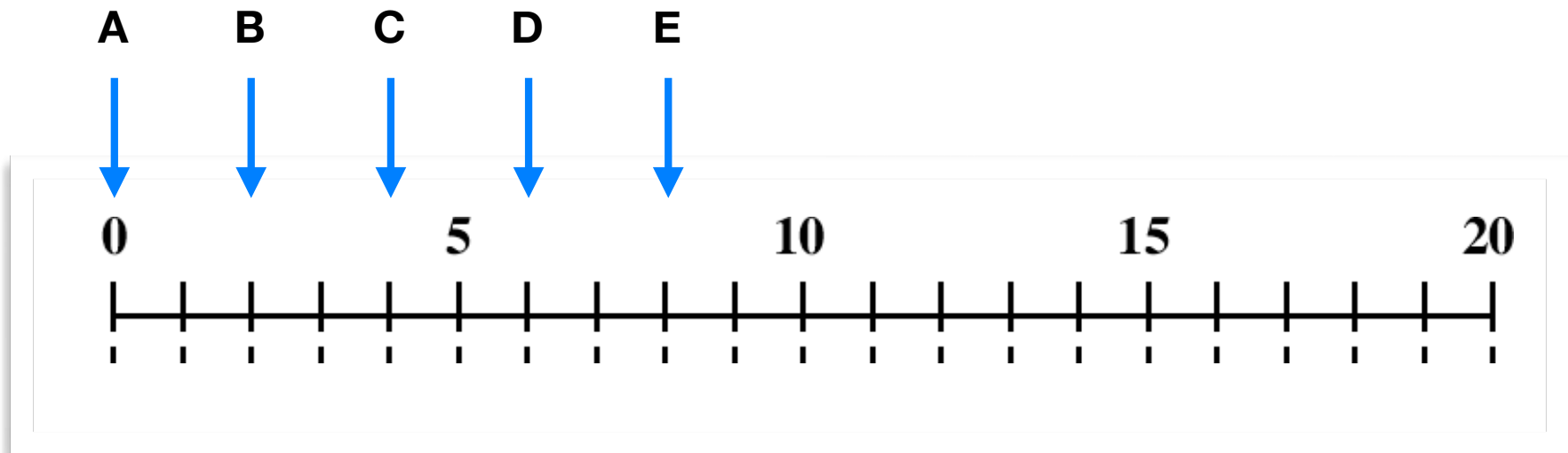
### ■ **Benefits?**

## Order of Scheduler Decisions

- **Sort all new arrived tasks into highest priority queue.**
- **Is time slot over?**
  - Check if other tasks in system are ready, then move task to lower priority queue
- **Does E/A Task come back from waiting?**
  - Sort it into the queue it was before at first position.
- **Choose first task from the highest priority queue to run next.**

# Taskset

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

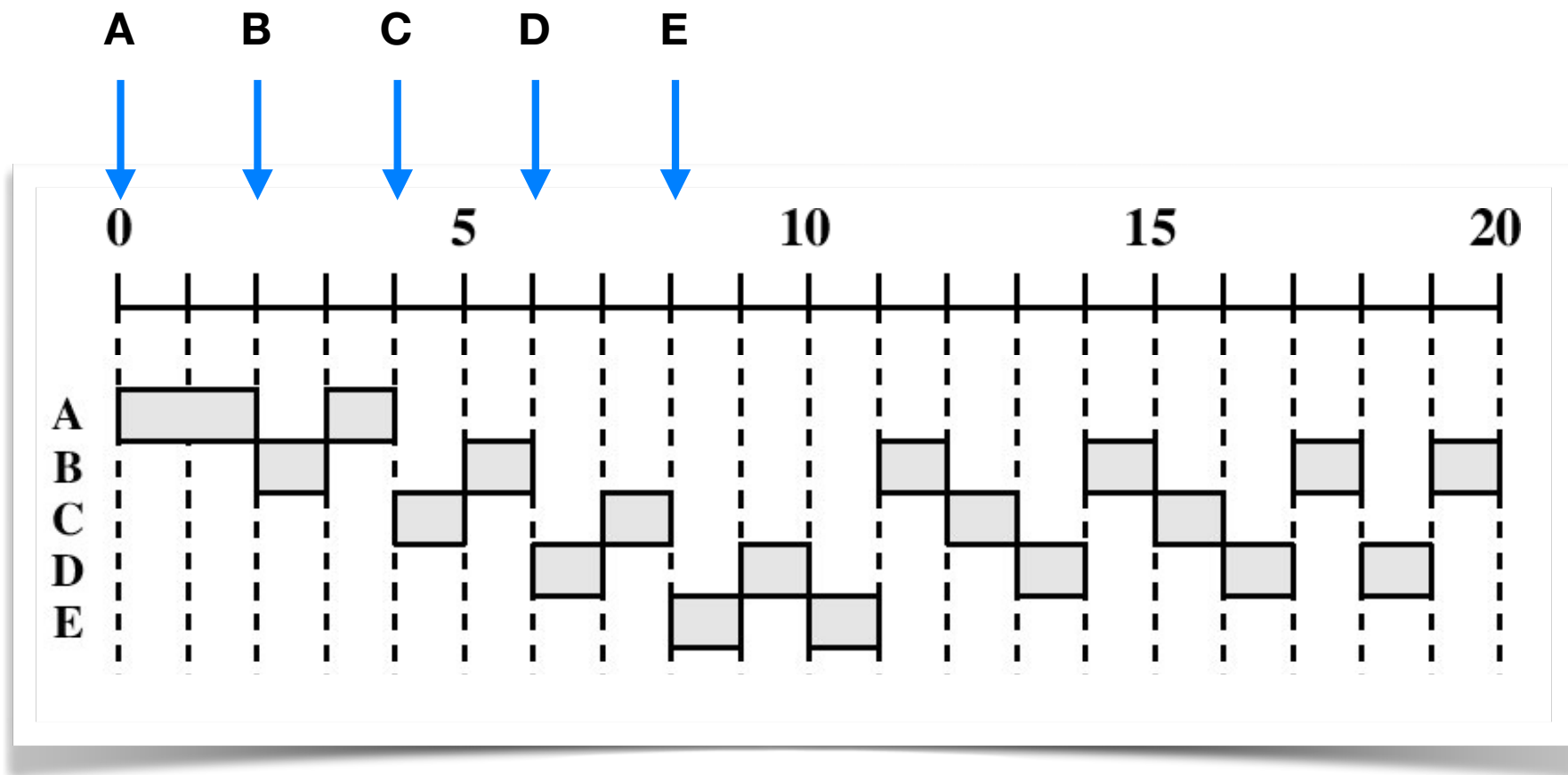


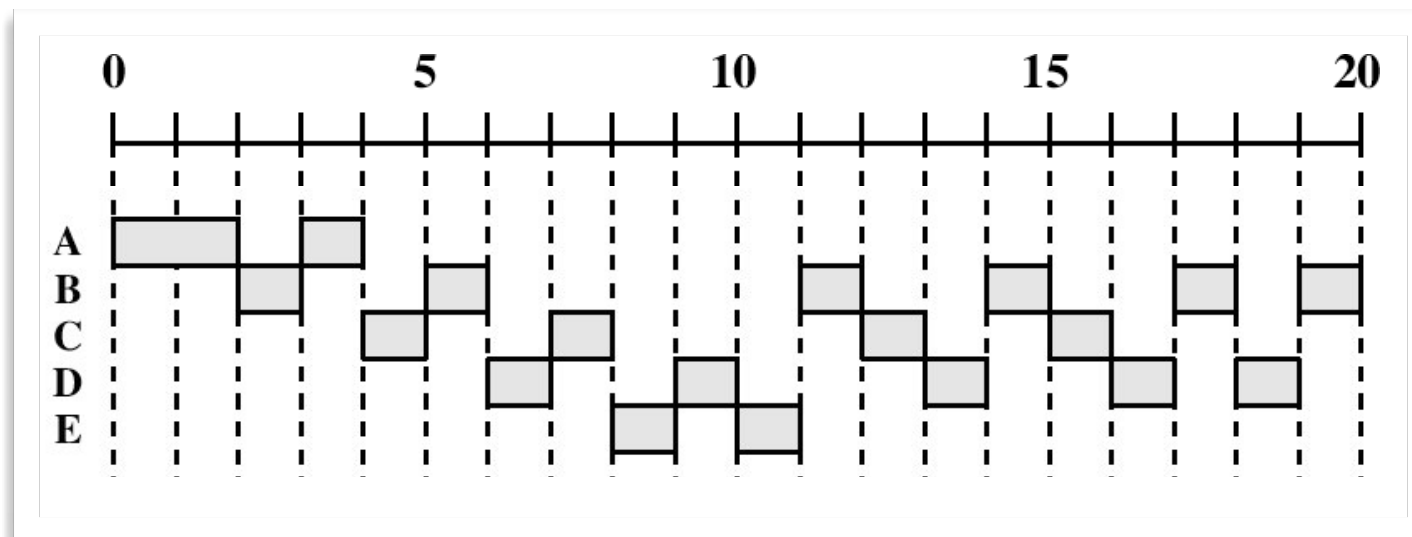
# MLFQ Analysis (q=1)

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

t	run	q1	q2	q3
0	A			
1				
2				
3				
4				
5				
6				
7				
8				
9				

# MLFQ Analysis ( $q=1$ )



MLFQ Analysis ( $q=1$ )

	A	B	C	D	E	Mean
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time ( $T_s$ )	3	6	4	5	2	
Finish Time	4	20	16	19	11	
Turnaround Time ( $T_r$ )	4	18	12	13	3	10.00
$T_r/T_s$	1.33	3.00	3.00	2.60	1.5	2.29

The background of the slide features a close-up, slightly blurred image of a clock face. The clock has a white face with black numbers and hands. The numbers 4, 10, 11, 12, 17, 18, 19, 25, 26, and 27 are visible. The clock hands are black, and the overall image has a soft, out-of-focus quality. In the top left corner, there is a small portion of a calendar grid showing dates from 10 to 27.

Thanks

Questions?