
Process Management

Process Scheduling

Lecture 3

Overview

- Concurrent Execution
- Process Scheduling
 - Definition
 - Process behavior
 - Processing environment
 - Criteria for good scheduling
 - Procedure of process scheduling
- Scheduling Algorithms
 - For batch processing systems
 - For interactive systems

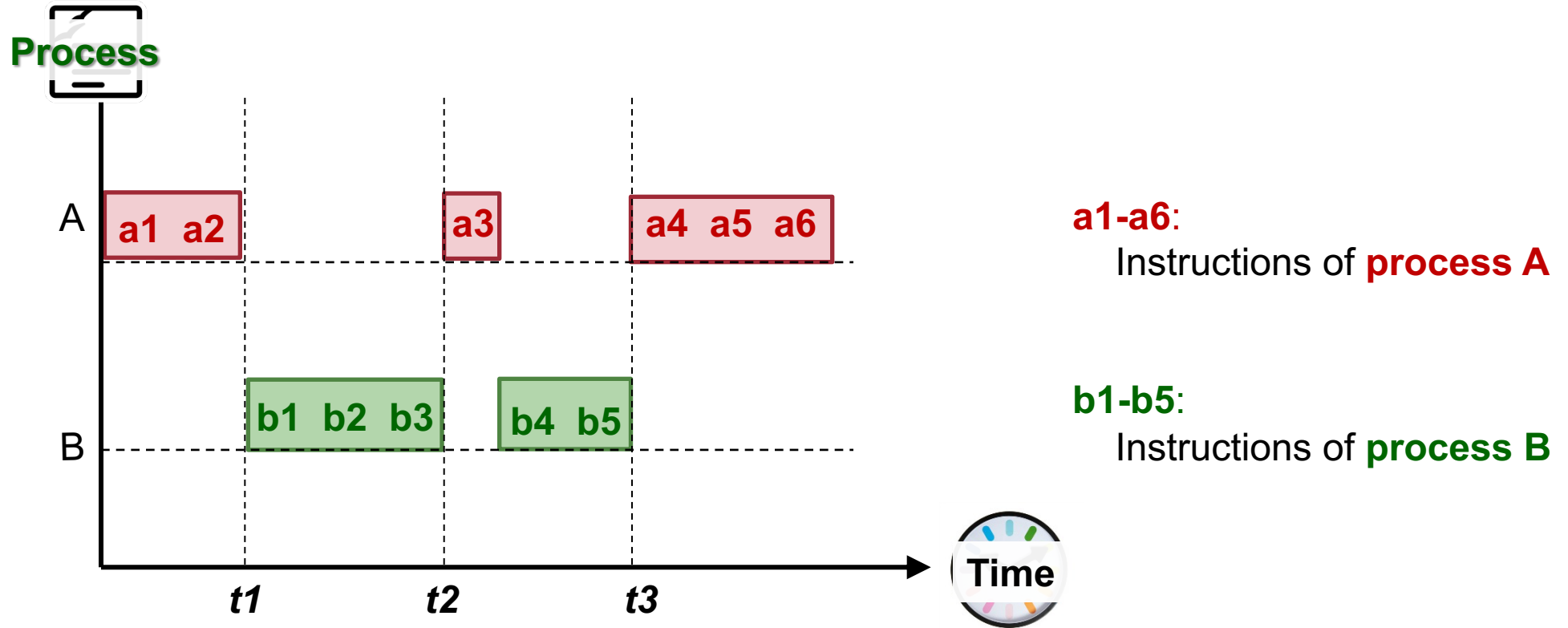
Concurrent Execution

■ Concurrent processes:

- ❑ Logical concept meaning that multiple processes **progress** in execution (at the same time)
- ❑ Could be virtual parallelism:
 - Illusion of parallelism (*pseudo-parallelism*)
- ❑ Could be physical parallelism
 - E.g., Multiple CPUs / Multi-Core CPU to allow parallel execution of multiple processes

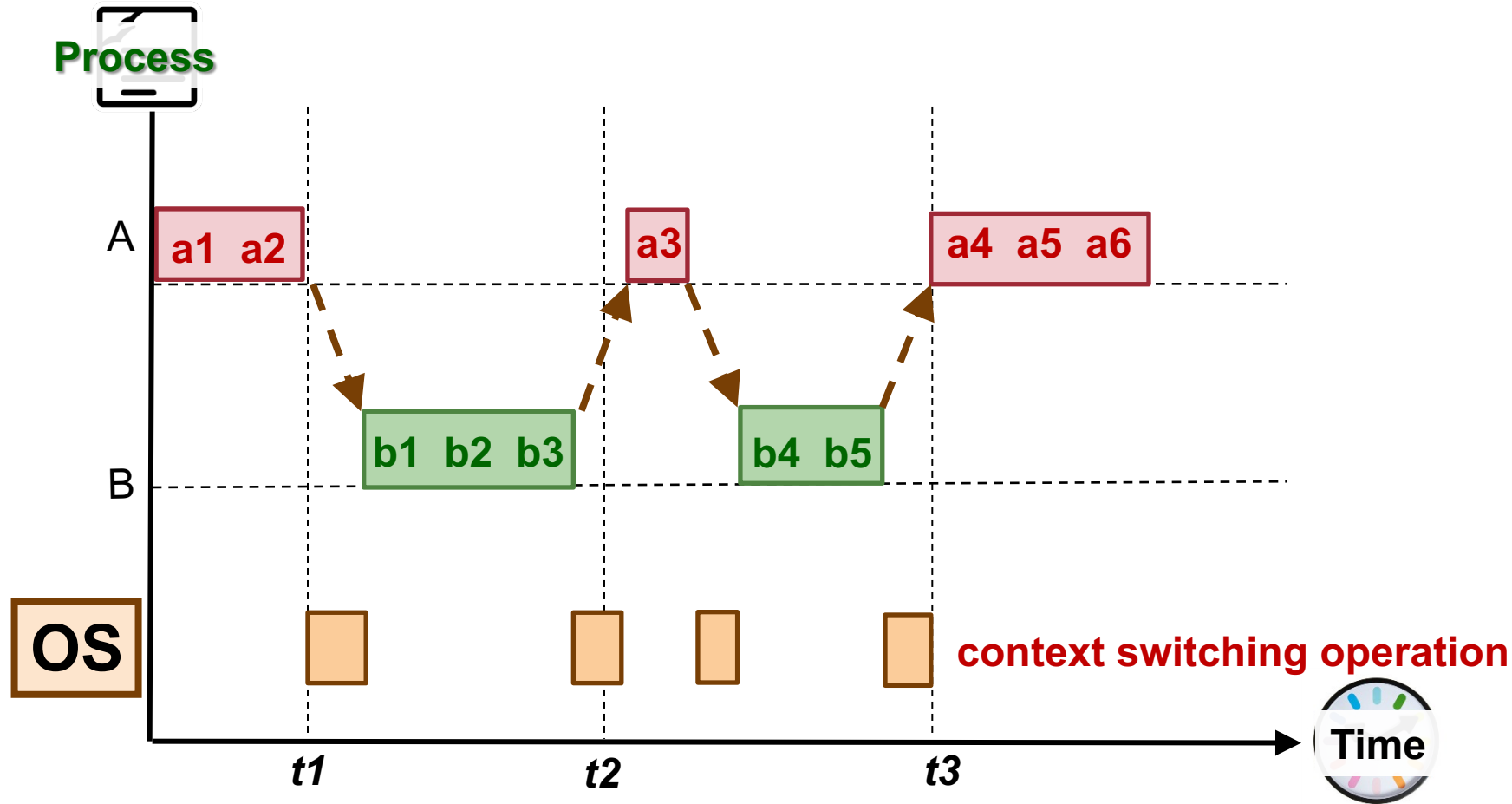
■ You can assume the two forms of parallelisms are not distinguished in the following discussion

Concurrency Example (Simplistic)



Concurrent execution on 1 CPU (core):
Interleave instructions from both processes
Also called **timeslicing**

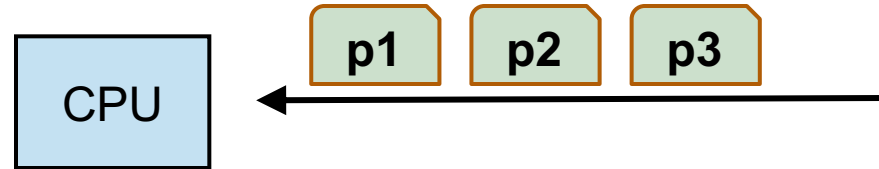
Interleaved Execution (context switch)



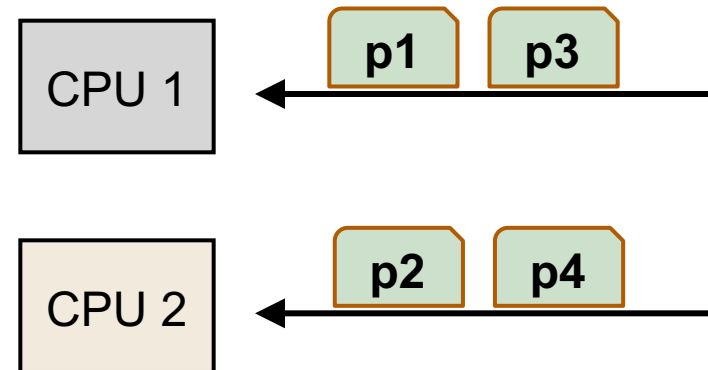
- Multitasking needs to change context between A and B:
 - OS incurs overhead in switching processes

Multitasking OS

- 1 core (CPU): timesliced execution of tasks



- Multiprocessor: timeslicing on n CPUs



Scheduling in OS: A definition

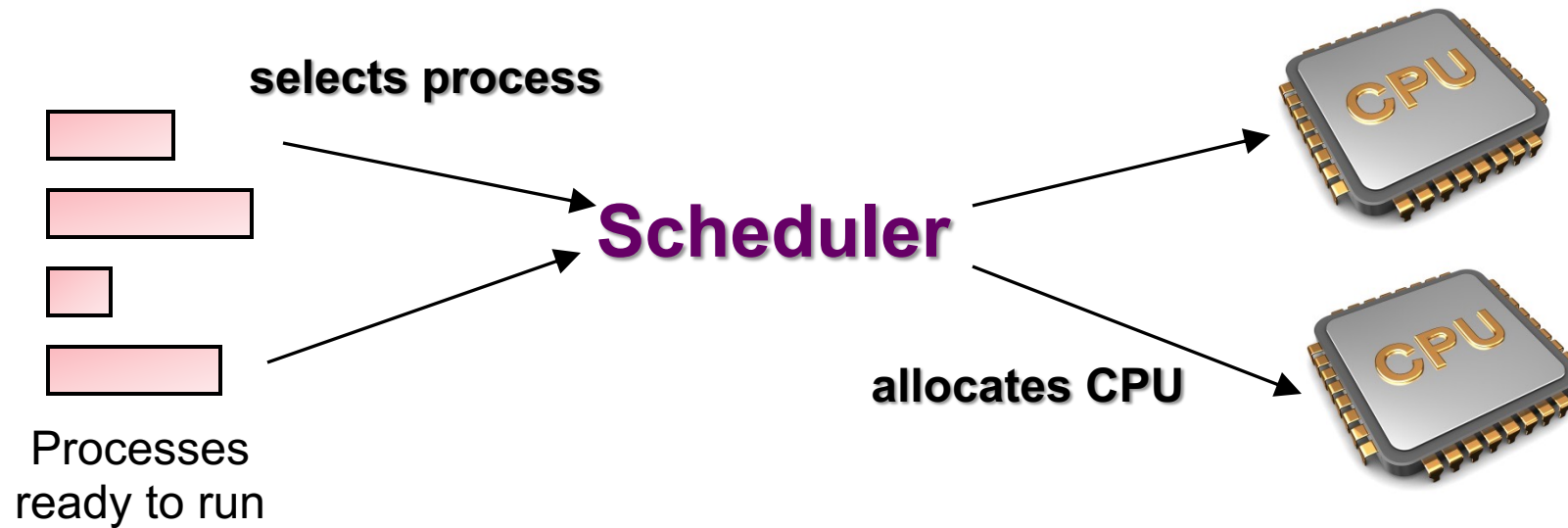
■ Problems with having multiple processes:

- ❑ If ready-to-run process is more than available CPUs, which should be chosen to run?
 - Similar idea in thread-level scheduling
- ❑ Known as the **scheduling problem**

■ Terminology:

- ❑ Scheduler
 - Part of the OS that makes scheduling decision
- ❑ Scheduling algorithm
 - The algorithm used by scheduler

Scheduling: Illustration



- Each process has different requirement of CPU time
 - ❑ **Process behavior**
- Many ways to allocate
 - ❑ Influenced by the **processing environment**
 - ❑ Known as **scheduling algorithms**
- A number of **criteria to evaluate the scheduler**

Process Behavior

- A typical process goes through phases of:

CPU-Activity:

- Computation
- E.g., number crunching
- **Compute-Bound Process** spends majority of its time here

IO-Activity:

- Requesting and receiving service from I/O devices
- E.g., print to screen, read from file, etc.
- **IO-Bound Process** spends majority of its time here

Processing Environment

■ Three categories:

1. **Batch Processing:**

- No user interaction required, no need to be responsive

2. **Interactive** (or Multiprogramming):

- With active user interacting with system
- Should be responsive: low and consistent in response time

3. **Real time processing:**

- Have deadline to meet
- Usually periodic process

Criteria for Scheduling Algorithms

- Many criteria to evaluate scheduling algorithms:
 - ❑ Largely influenced by the processing environment
 - ❑ May be conflicting

Criteria for **all processing environments**:

- **Fairness:**
 - ❑ Should get a fair share of CPU time
 - On a per process basis OR
 - On a per user basis
 - ❑ Also means **no starvation**
- **Utilization:**
 - ❑ All parts of the computing system should be utilized

When to perform scheduling?

- Two types of scheduling policies

- Defined by **when** scheduling is triggered

- **Non-preemptive (Cooperative)**

- A process stayed scheduled (in running state) until it blocks or gives up the CPU voluntarily

- **Preemptive**

- A process is given a fixed time quota to run
 - Possible to block or give up early
 - At the end of the time quota, the running process is suspended
 - Another ready process gets picked if available

Scheduling a Process: Step-by-Step

1

- Scheduler is triggered (OS takes over)

2

- If context switch is needed:
 - Context of current running process is saved and placed on blocked queue / ready queue

3

- Pick a suitable process **P** to run base on scheduling algorithm

4

- Setup the context for **P**

5

- Let process **P** run

SCHEDULING FOR BATCH PROCESSING

Overview

- On batch processing system:
 - ❑ No user interaction
 - ❑ Non-preemptive scheduling is predominant
- Scheduling algorithms are generally easier to understand and implement
 - ❑ Commonly resulted in variants/improvements that can be used for other type of systems
- Three algorithms covered:
 - ❑ **F**irst-**C**ome **F**irst **S**erved (**FCFS**)
 - ❑ **S**hortest **J**ob **F**irst (**SJF**)
 - ❑ **S**hortest **R**emaining **T**ime (**SRT**)

Criteria for batch processing

■ Turnaround time:

- ❑ Total time taken, i.e., finish time - arrival time
- ❑ Related to **waiting time**: time spent waiting for CPU

■ Throughput:

- ❑ Number of tasks finished per unit time
- ❑ i.e., Rate of task completion

■ CPU utilization:

- ❑ Percentage of time when CPU is working on a task

First-Come First-Served: **FCFS**

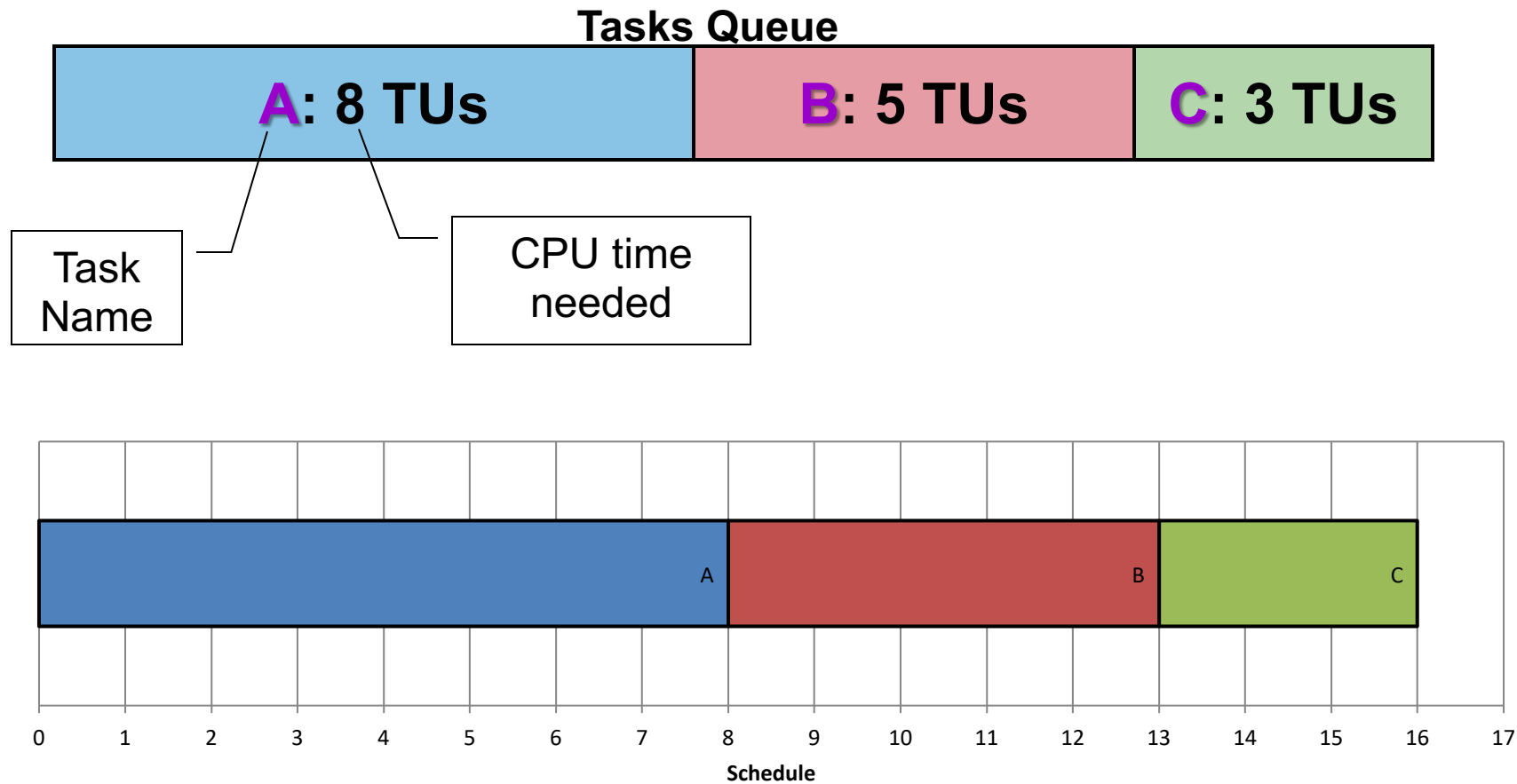
■ General Idea:

- ❑ Tasks are stored on a First-In-First-Out (FIFO) queue based on arrival time
- ❑ Pick the first task in the queue to run until:
 - Task is done OR task is blocked
- ❑ Blocked task is removed from the FIFO queue
 - When it is ready again, it is placed at the back of queue
 - i.e., just like a newly arrive task

■ Guaranteed to have no **starvation**:

- ❑ The number of tasks in front of task X in FIFO is always decreasing
➔ task X will get its chance eventually

First-Come First-Served: Illustration



- The average total waiting time for 3 tasks
 - $(0 + 8 + 13)/3 = 7$ Time Units

First-Come First-Served: **Shortcomings**

- Simple reordering can reduce the average waiting time!
- Also, consider this scenario:
 - ❑ First task (task **A**) is CPU-Bound and followed by a number of IO-Bound tasks **X**
 - ❑ Task **A** running
 - All tasks **X** waiting in ready queue (I/O device idling)
 - ❑ Task **A** blocked on I/O
 - All tasks **X** execute quickly and blocked on I/O (CPU idling)
 - ❑ known as **Convoy Effect**

Shortest Job First: SJF

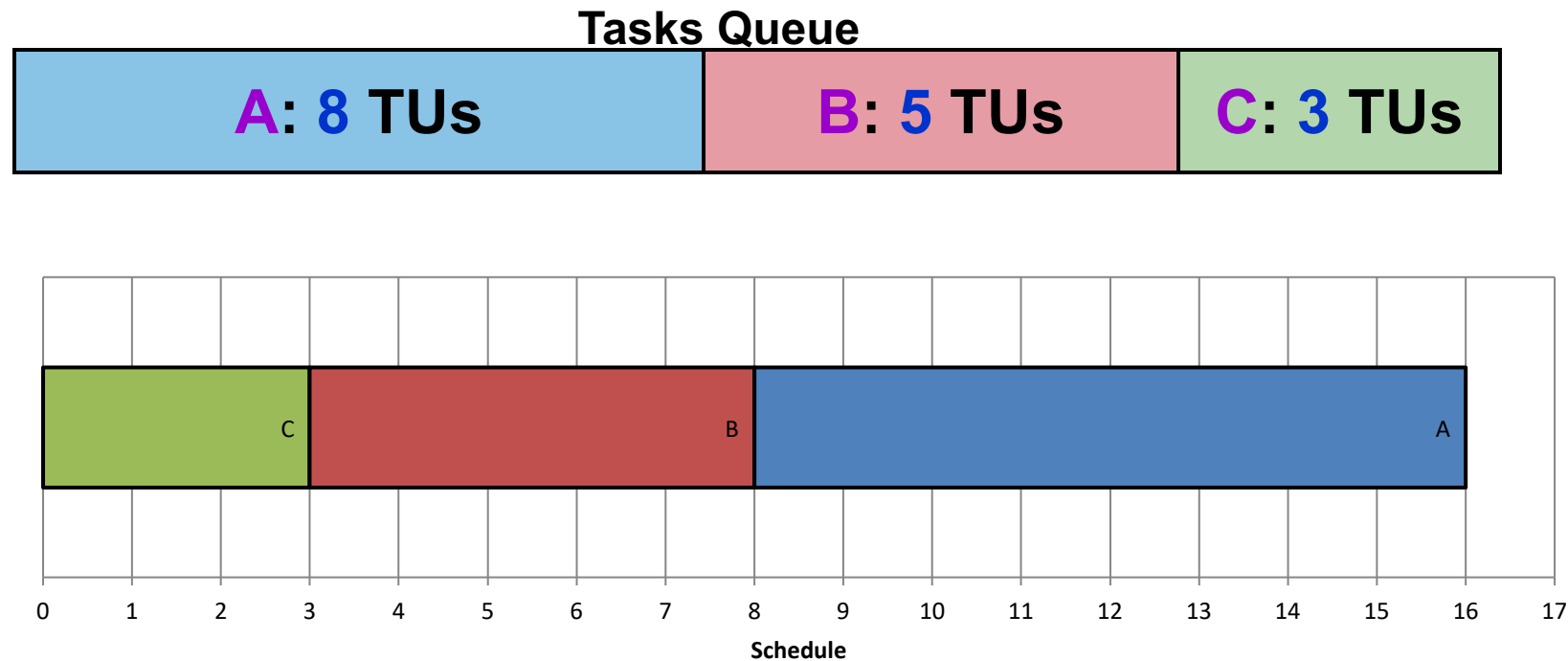
■ General Idea:

- ❑ Select task with the smallest total CPU time

■ Notes:

- ❑ Need to know **total CPU time** for a task in advance
 - Have to "guess" if this info is not available
- ❑ Given a fixed set of tasks:
 - Minimizes average waiting time
- ❑ Starvation is possible:
 - Biased towards short jobs
 - Long jobs may never get a chance!

Shortest Job First: Illustration



- The average total waiting time for 3 tasks
 - $(0 + 3 + 8)/3 = 3.66$ Time Units
- Can be shown that SJF **guarantees** smallest average waiting time

Shortest Job First: Predicting CPU Time

- A task usually goes through several phases of CPU-Activity:
 - Possible to guess the future CPU time requirement by the previous CPU-Bound phases

- Common approach (Exponential Average):

$$\text{Predicted}_{n+1} = \alpha \text{Actual}_n + (1-\alpha) \text{Predicted}_n$$

- Actual_n = The most recent CPU time consumed
- Predicted_n = The past history of CPU Time consumed
- α = Weight placed on recent event or past history
- Predicted_{n+1} = Latest prediction

Shortest Remaining Time: **SRT**

■ General Idea:

□ Variation of SJF:

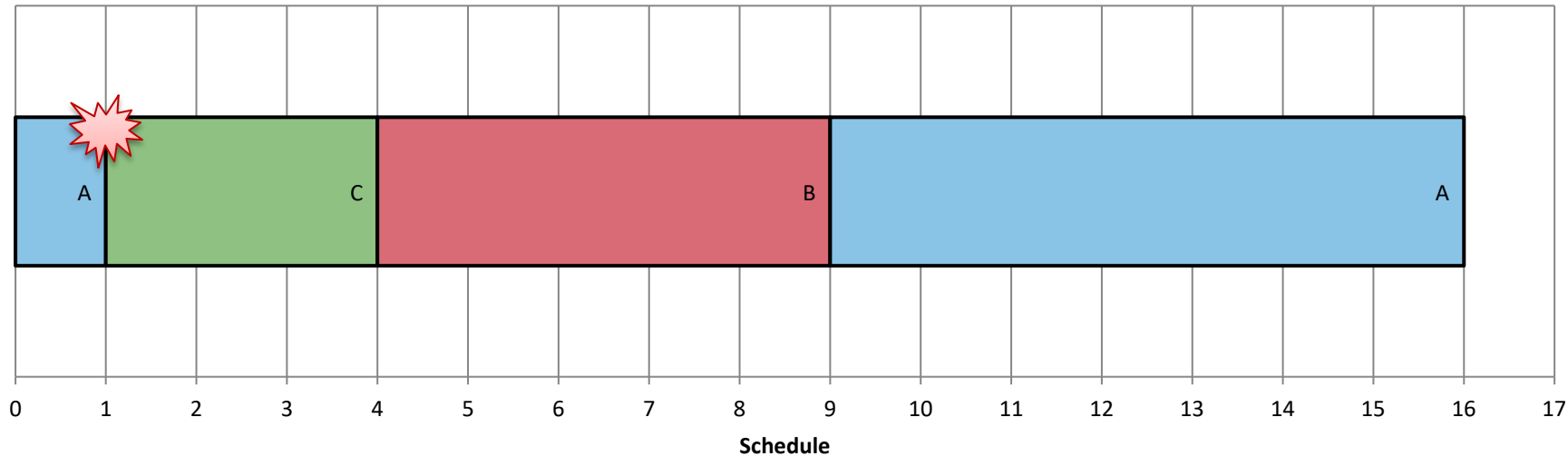
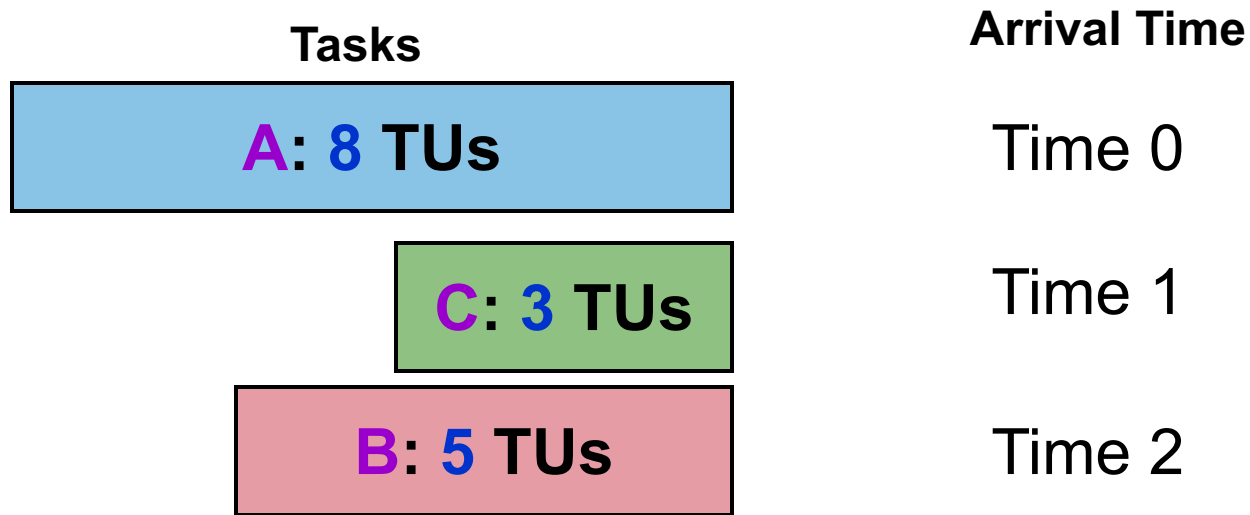
- Use remaining time
- Preemptive

□ Select job with shortest remaining (or expected) time

■ Notes:

- New job with shorter remaining time can preempt currently running job
- Provide good service for short job even when it arrives late

Shortest Remaining Time First: Illustration



SCHEDULING FOR INTERACTIVE SYSTEMS

Criteria for **interactive environment**

■ **Response time:**

- Time between request and response by system

■ **Predictability:**

- Variation in response time, lesser variation == more predictable

Preemptive scheduling algorithms are used to ensure good response time

→ Scheduler needs to run **periodically**

Ensuring Periodic Scheduler

■ Questions:

- ❑ How can the scheduler "take over" the CPU periodically?
- ❑ How can we ensure the user program can never stop the scheduler from executing?

■ Ingredients for answer:

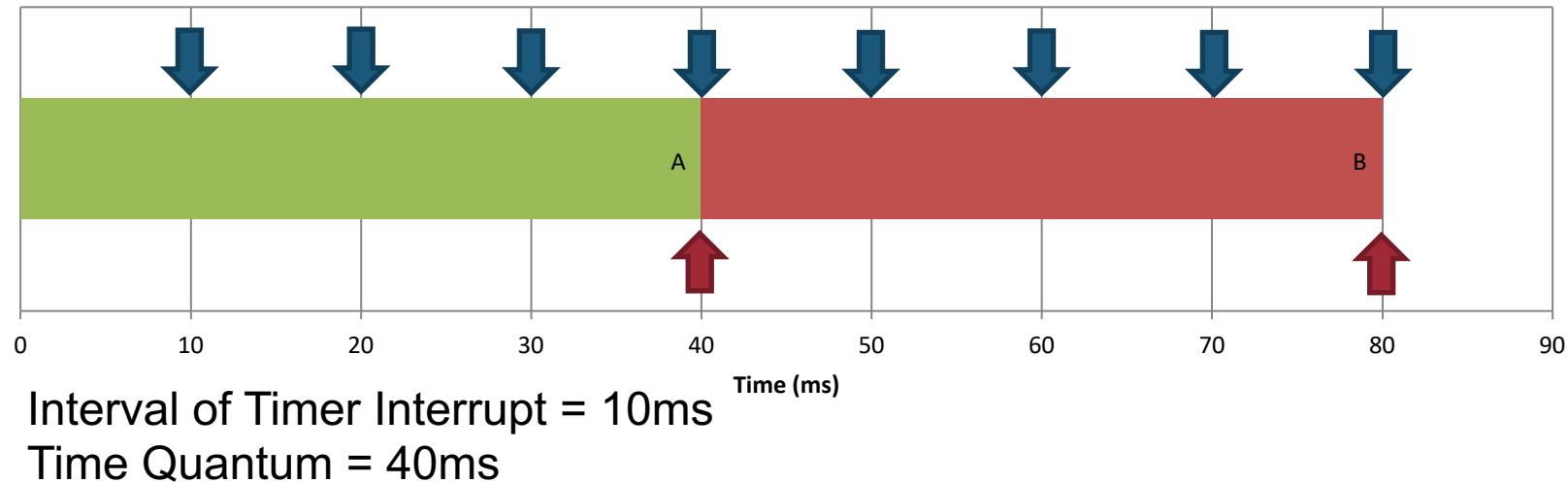
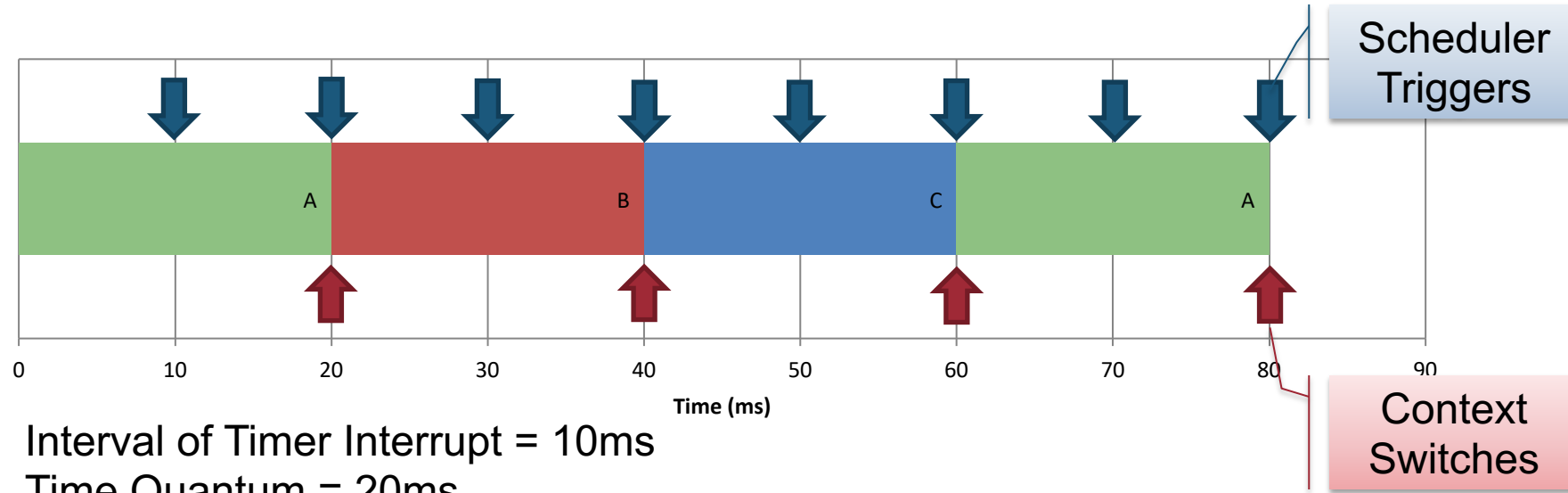
- ❑ Timer interrupt = Interrupt that goes off periodically (based on hardware clock)
- ❑ OS ensures timer interrupt cannot be intercepted by any other program

➔ Timer interrupt handler **invokes scheduler**

Terminology: Timer & Time Quantum

- Interval of Timer Interrupt (ITI):
 - ❑ OS scheduler is invoked on every timer interrupt
 - ❑ Typical values (1ms to 10ms)
- Time Quantum:
 - ❑ Execution duration given to a process
 - ❑ Could be constant or variable among the processes
 - ❑ Must be multiples of interval of timer interrupt
 - ❑ Large range of values (commonly 5ms to 100ms)

Illustration: ITI vs Time Quantum



Scheduling Algorithms:

- Algorithms covered:

1. Round Robin (RR)
2. Priority Based
3. Multi-Level Feedback Queue (MLFQ)
4. Lottery Scheduling

Round Robin: **RR**

■ General Idea:

- ❑ Tasks are stored in a FIFO queue
- ❑ Pick the first task from queue front to run until:
 - A fixed **time slice** (**quantum**) elapsed, or
 - The task gives up the CPU voluntarily, or
 - The task blocks
- ❑ The task is then placed at the end of queue to wait for another turn
 - Blocked task will be moved to other queue to wait for its requested resource
- ❑ When blocked task is ready again, it is placed at the end of queue

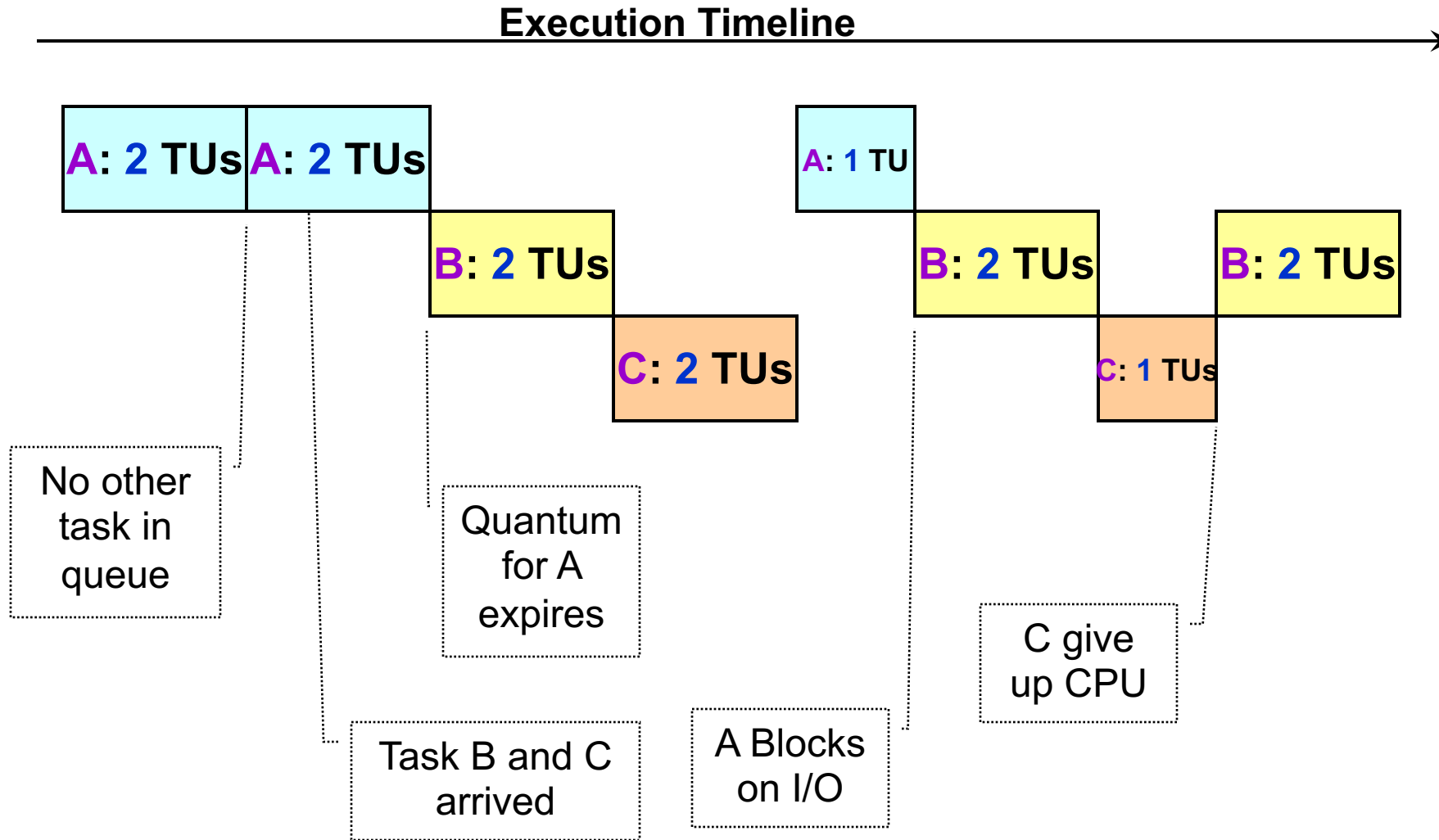
Round Robin: **RR**

(cont)

■ Notes:

- ❑ Basically a preemptive version of FCFS
- ❑ **Response time guarantee:**
 - Given n tasks and quantum q
 - Time before a task get CPU is bounded by $(n-1)q$
- ❑ **Timer interrupt needed:**
 - For scheduler to check on quantum expiry
- ❑ The **choice of time quantum** duration is important:
 - Big quantum: Better CPU utilization but longer waiting time
 - Small quantum: Bigger overhead (worse CPU utilization) but shorter waiting time

Round Robin: Illustration



Priority Scheduling

■ General Idea:

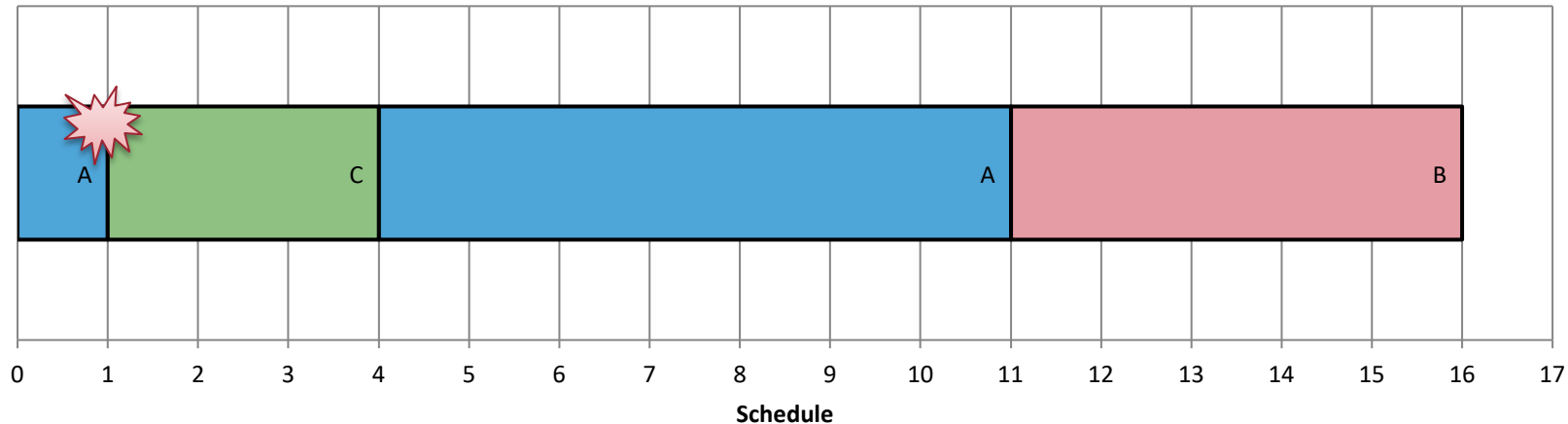
- ❑ Some processes are more important than others
 - Cannot treat all process as equal
- ❑ Assign a priority value to all tasks
- ❑ Select task with the highest priority value

■ Variants:

- ❑ Preemptive version:
 - Higher priority process can preempt running process with lower priority
- ❑ Non-preemptive version:
 - Late coming high priority process has to wait for next round of scheduling

Priority Scheduling: Illustration

Tasks	Arrival Time	Priority (1=highest)
A: 8 TUs	Time 0	3
C: 3 TUs	Time 1	1
B: 5 TUs	Time 1	5



Priority Scheduling: Shortcomings

- Low priority process can starve:
 - ❑ High priority process keep hogging the CPU
 - ❑ Even worse in preemptive variant
- Possible solutions:
 - ❑ Decrease the priority of currently running process after every time quantum
 - Eventually dropped below the next highest priority
 - ❑ Give the current running process a time quantum
 - This process is not considered in the next round of scheduling
- Generally, it is hard to guarantee or control the exact amount of CPU time given to a process using priority

Priority Scheduling: Priority Inversion

- Consider the scenario:
 - ❑ Priority: {A = 1, B = 3, C = 5} (1 is highest)
 - ❑ Task **C** starts and locks a resource (e.g., file)
 - ❑ Task **B** preempts **C**
 - **C** is unable to unlock the resource
 - ❑ Task **A** arrives and needs the same resource as **C**
 - but the resource is locked!
 - ➔ Task **B** continues execution even if Task **A** has higher priority
- Known as **Priority Inversion**:
 - ❑ Lower priority task preempts higher priority task

Multi-level Feedback Queue (MLFQ)

- Designed to solve one BIG + HARD issue:
 - ❑ How do we schedule without perfect knowledge?
 - ❑ Most algorithms require certain information (process behavior, running time, etc.)
- MLFQ is:
 - ❑ Adaptive: "Learn the process behavior automatically"
 - ❑ Minimizes both:
 - Response time for IO bound processes
 - Turnaround time for CPU bound processes

MLFQ: Rules

■ Basic rules:

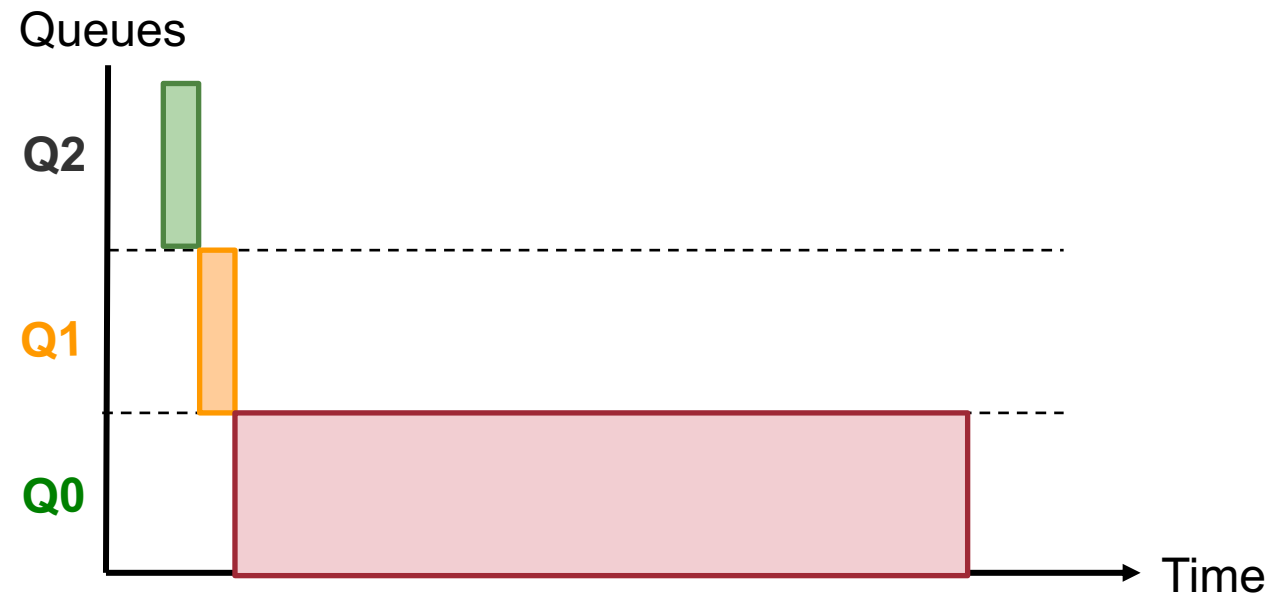
1. If $\text{Priority}(A) > \text{Priority}(B) \rightarrow A$ runs
2. If $\text{Priority}(A) == \text{Priority}(B) \rightarrow A$ and B runs in RR

■ Priority Setting/Changing rules:

1. New job \rightarrow Highest priority
2. If a job fully utilized its time quantum \rightarrow priority reduced
3. If a job gives up / blocks before finishes its time quantum \rightarrow priority retained

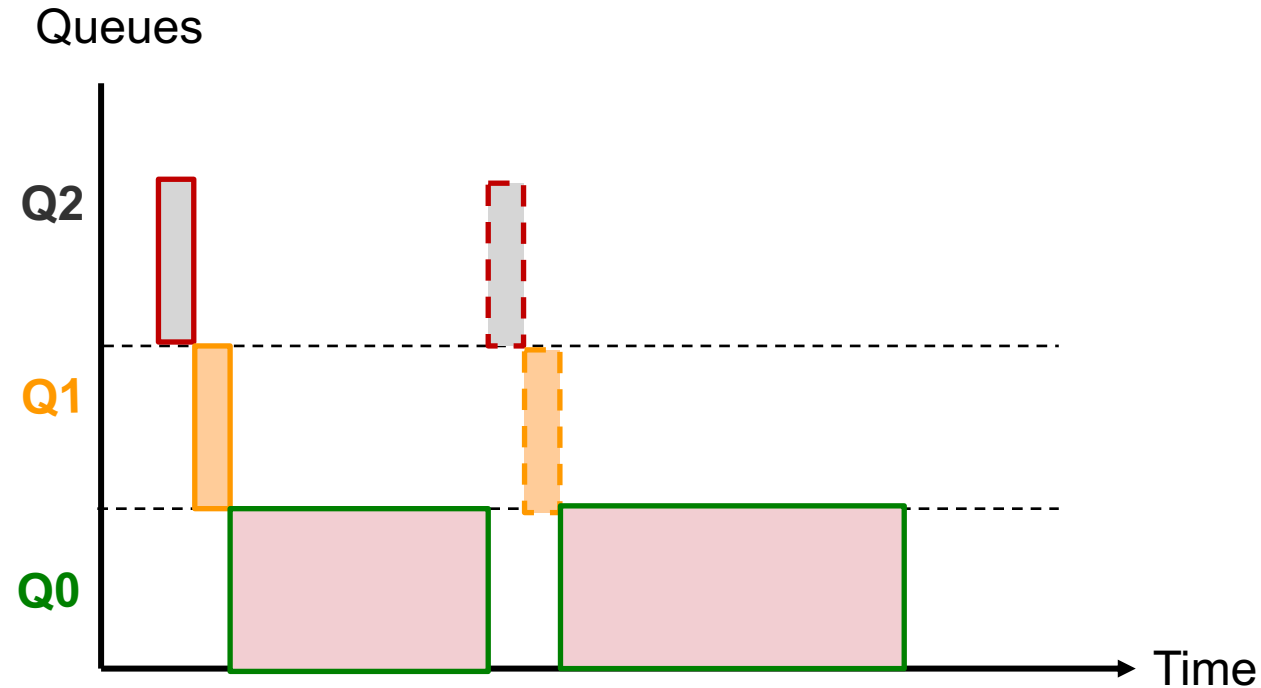
MLFQ: Example 1

- 3 Queues: Q2 (highest priority), Q1, Q0
- A single long running job
 - Try to apply the rules and check your understanding



MLFQ: Example 2

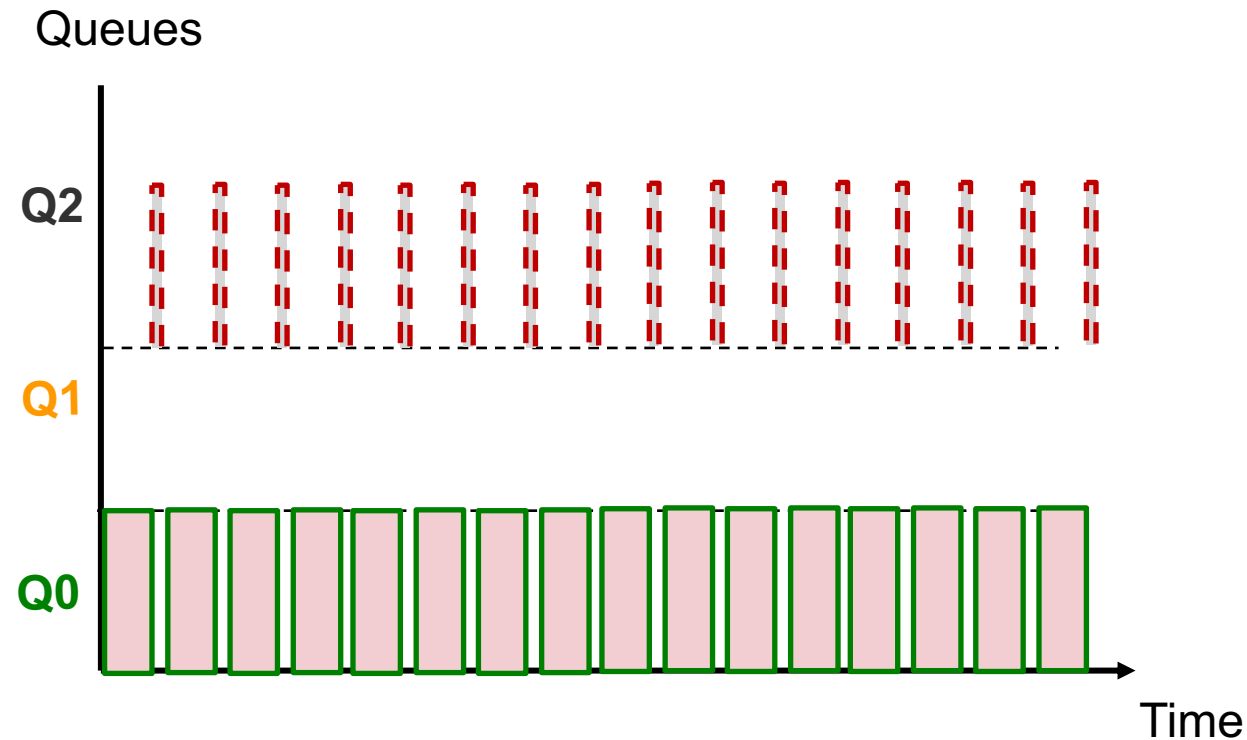
- Example 1 + a short job in the middle
 - A short job appears sometime in the middle



MLFQ: Example 3

■ Two jobs:

- **A** = CPU bound (already in the system for quite some time)
- **B** = I/O bound



MLFQ: Questions to ponder

- Can you think of a way to abuse the algorithm? 😊
 - Equivalent question: MLFQ does not work well for what kind combination of jobs?
- What are the ways to rectify the above?

Lottery Scheduling

■ General Idea:

- ❑ Give out “lottery tickets” to processes for various system resources
 - E.g., CPU time, I/O devices, etc.
- ❑ When a scheduling decision is needed:
 - A lottery ticket is chosen randomly among eligible tickets
 - The winner is granted the resource
- ❑ In the long run, a process holding **X%** of tickets
 - Can win **X%** of the lottery held
 - Use the resource **X%** of the time

Lottery Scheduling: Properties

- **Responsive:**
 - ❑ A newly created process can participate in the next lottery
- **Provides good level of control:**
 - ❑ A process can be given Y lottery tickets
 - It can then distribute to its child process
 - ❑ An important process can be given more lottery tickets
 - Can control the proportion of usage
 - ❑ Each resource can have its own set of tickets
 - Different proportion of usage per resource per task
- **Simple Implementation**

Summary

■ Scheduling in OS:

- ❑ Basic definition
- ❑ Factors that affect scheduling
 - Process, Environment
- ❑ Criteria of good scheduling

■ Scheduling Algorithms:

- ❑ FCFS, SJF, SRT for batch processing systems
- ❑ RR, Priority base, Multi-Level Queues, MLFQ and Lottery scheduling for interactive systems

Reference

- Modern Operating System (3rd Edition)
 - ❑ By Andrew S. Tanenbaum
 - ❑ Published by Pearson
 - ❑ Chapter 2.4
- Operating System Concepts (7th Edition)
 - ❑ By Silberschatz, Galvin, and Gagne
 - ❑ Published by Wiley Brothers
 - ❑ Chapter 5
- Operating Systems: Three Easy Pieces
 - ❑ By Arpaci-Dusseau and Arpaci-Dusseau
 - ❑ <http://pages.cs.wisc.edu/~remzi/OSTEP/>