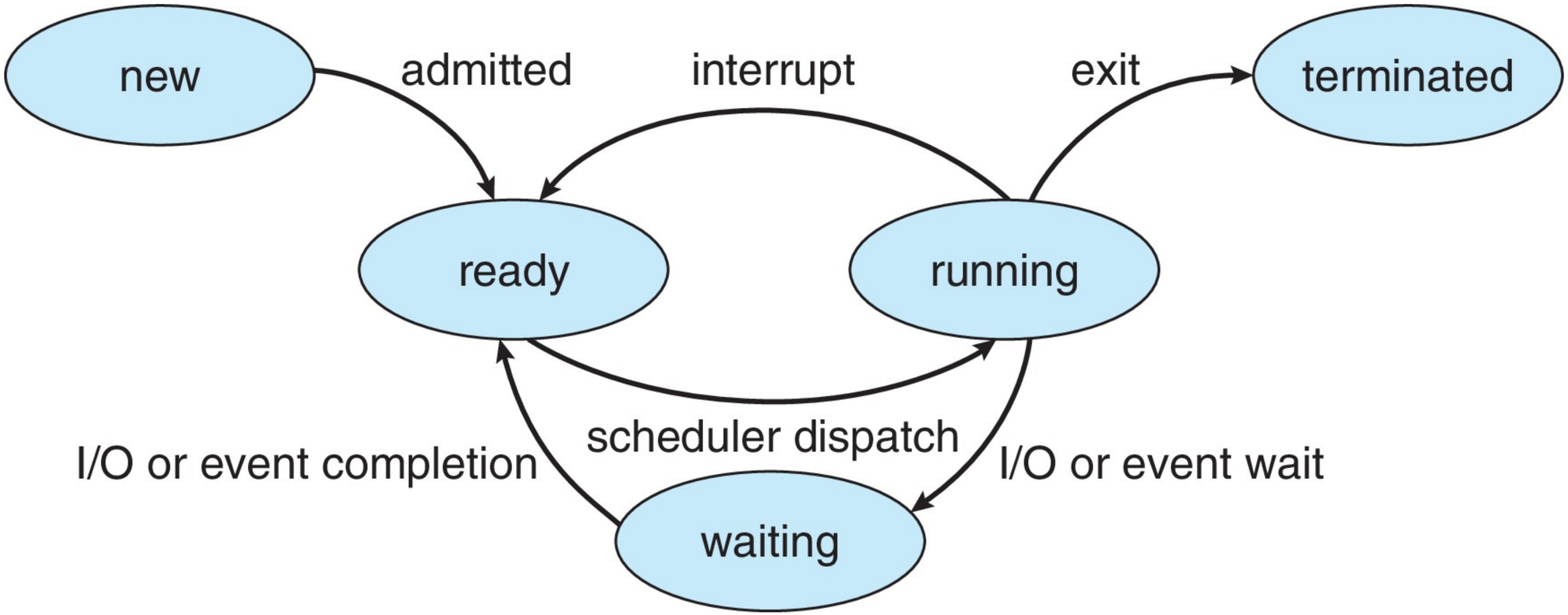
CPU Scheduling

01/24/2023

Professor Amanda Bienz

Review : Life of a Process



Scheduling : Introduction

- **Workload assumptions :**
  - Each job runs for the **same amount of time**
  - All jobs **arrive** at the same time
  - All jobs only use the **CPU** (i.e. no I/O)
  - The **run-time** of each job is known

Scheduling Metrics

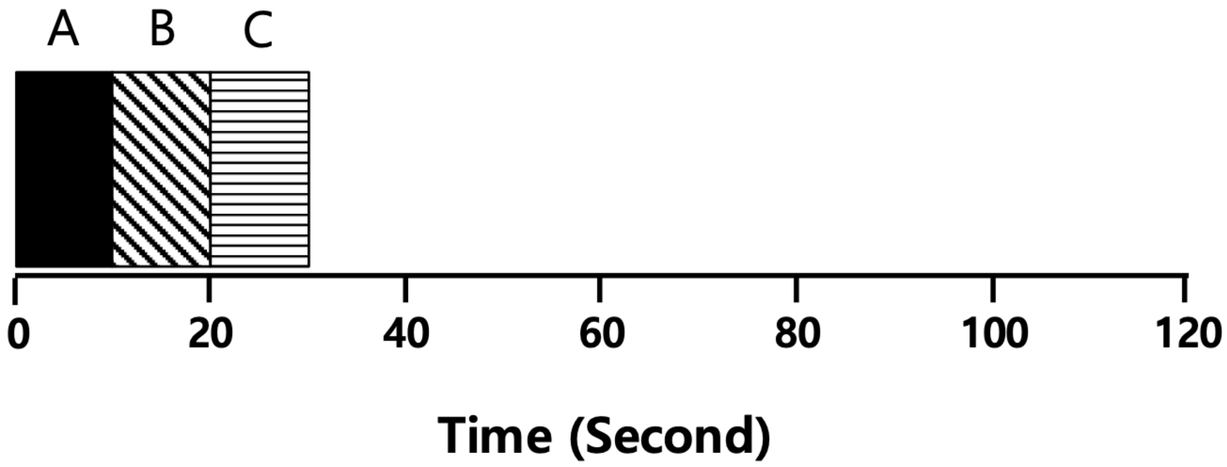
- **Turnaround time :** the time at which the job completes minus the time at which the job arrived in the system

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

- **Fairness :** performance and fairness are often at odds in scheduling
- **Role of scheduler metrics :**
  - Simple metrics are good way to understand basic scheduler tradeoffs
  - Optimizing explicit metrics can make sense in dedicated systems
  - Schedulers in general purpose systems frequently have hard-to-quantify (and thus optimize) optimization criteria

First In, First Out (FIFO)

- **First Come, First Served (FCFS)**
  - Very simple and easy to implement (it's just a queue)
- **Example:**
  - A arrives just before B, which arrives just before C
  - Each job runs for 10 seconds
- Turnaround times?

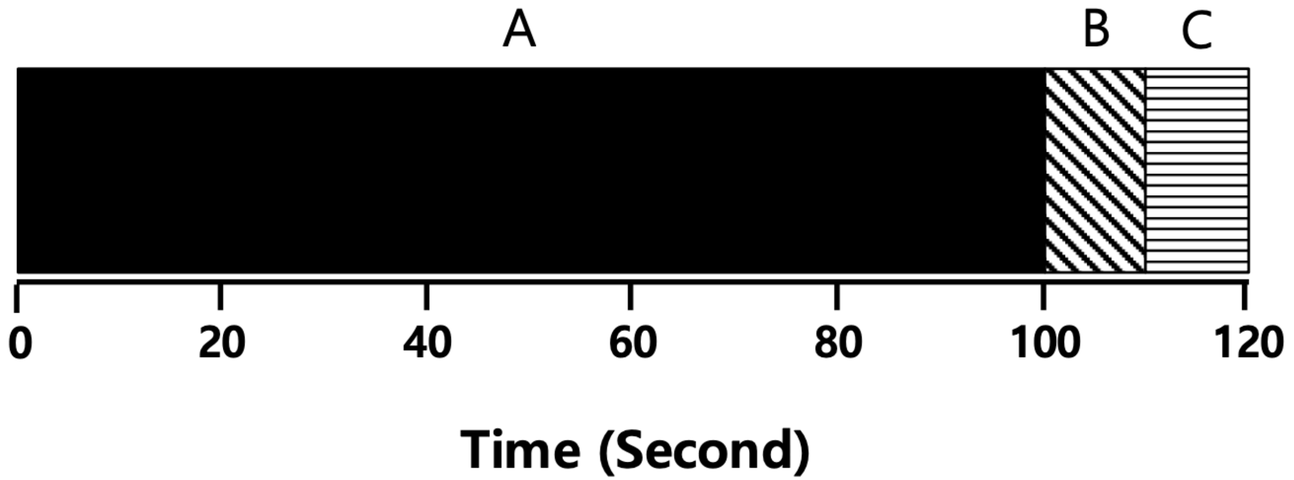


Problem with FIFO - Convoy Effect

- What if we relax assumption 1 :
  - Each job no longer runs for the same amount of time
- Example :
  - A arrives just before B, which arrives just before C
  - A runs for 100 seconds, B and C each run for 10

Problem with FIFO - Convoy Effect

- What if we relax assumption 1 :
  - Each job no longer runs for the same amount of time
- Example :
  - A arrives just before B, which arrives just before C
  - A runs for 100 seconds, B and C each run for 10

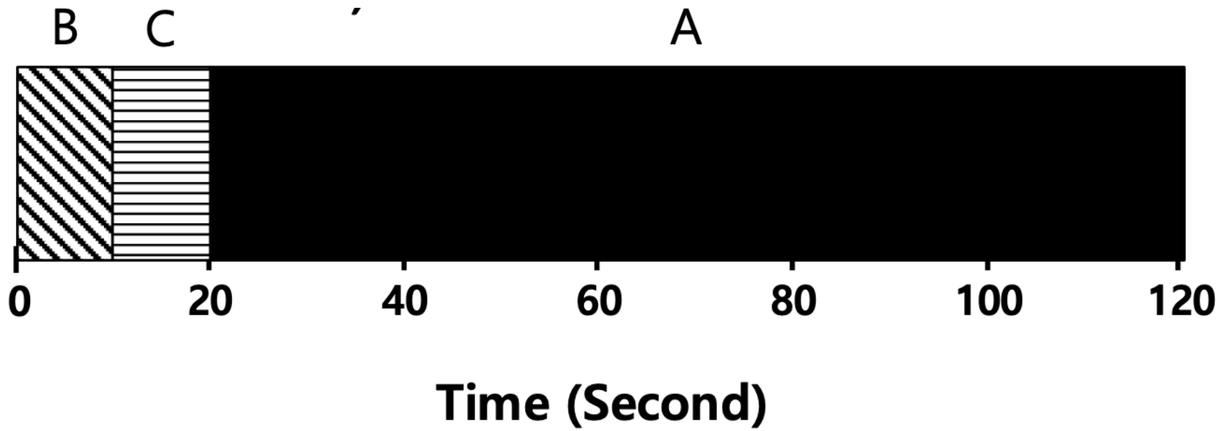


Shortest Job First (SJF)

- **Run the shortest job first, then the next shortest, and so on**
  - Non-preemptive scheduler
- **Example :**
  - A arrives just before B, which arrives just before C
  - A runs for 100 seconds, B and C run for 10

Shortest Job First (SJF)

- **Run the shortest job first, then the next shortest, and so on**
  - Non-preemptive scheduler
- **Example :**
  - A arrives just before B, which arrives just before C
  - A runs for 100 seconds, B and C run for 10

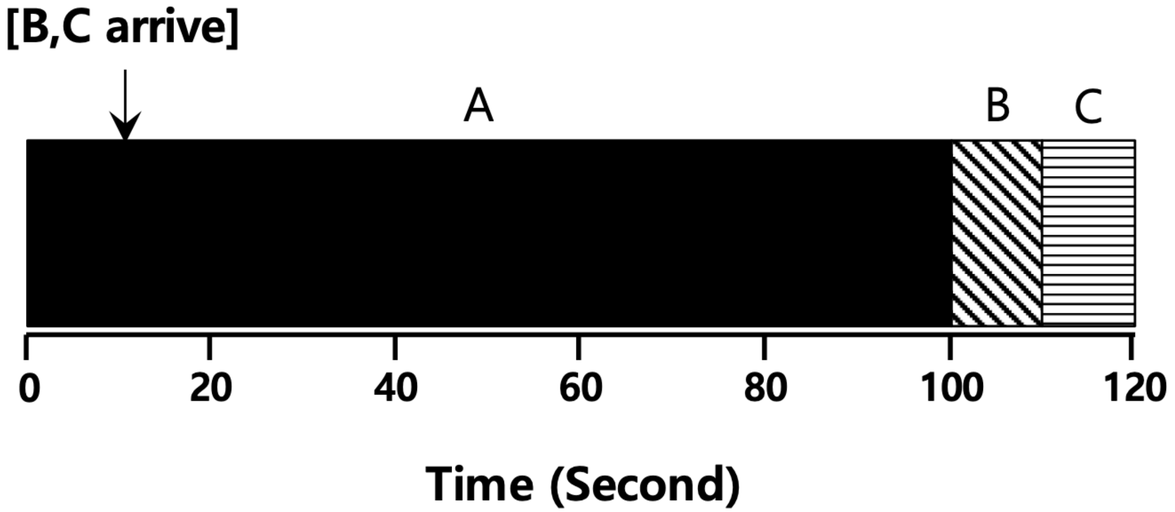


SJF with Late Arrivals from B and C

- Let’s relax assumption 2 : Jobs can arrive at any time
- **Example :**
  - A arrives at t=0 and needs to run for 100 seconds
  - B and C arrive at t=10 and each need to run for 10 seconds

SJF with Late Arrivals from B and C

- Let's relax assumption 2 : Jobs can arrive at any time
- **Example :**
  - A arrives at t=0 and needs to run for 100 seconds
  - B and C arrive at t=10 and each need to run for 10 seconds



Shortest Time-to-Completion First (STCF)

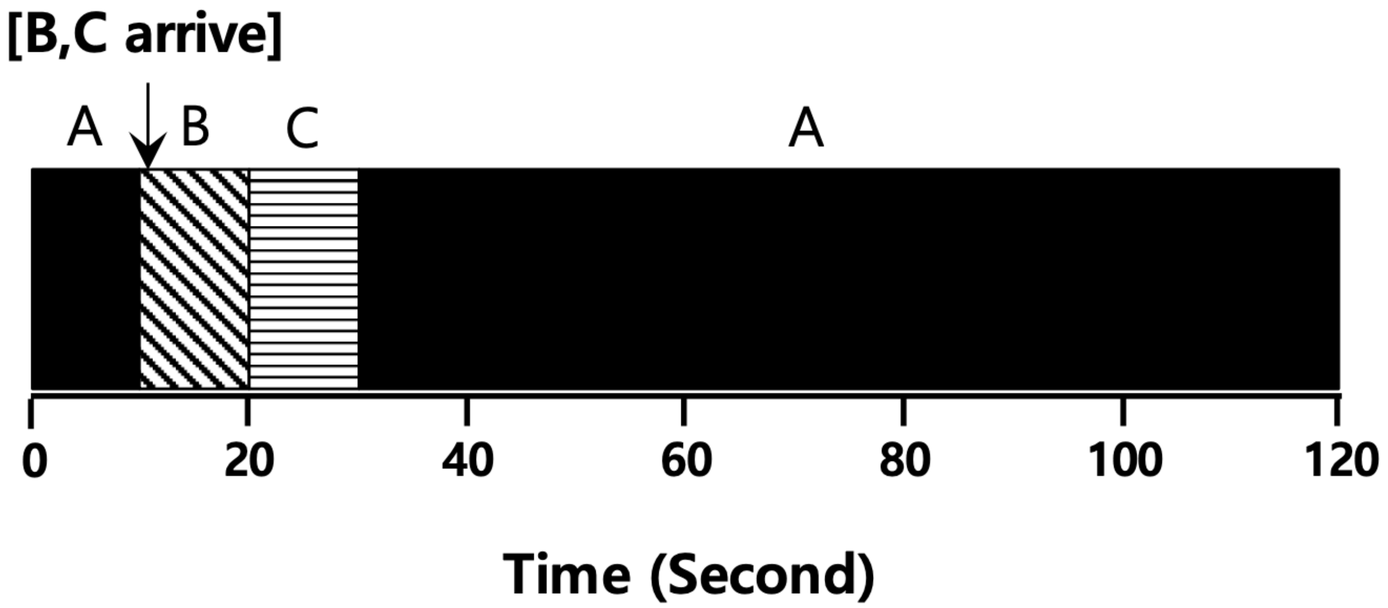
- Add preemption to SJF
  - Also know as Preemptive Shortest Job First (PSJF)
- A new job enters the system:
  - Look at the remaining jobs and new job
  - Schedule the job which has the least time left

Shortest Time-to-Completion First (STCF)

- **Example:**
  - A arrives at t=0 and needs to run for 100 seconds
  - B and C arrive at t = 10 and each need to run for 10 seconds

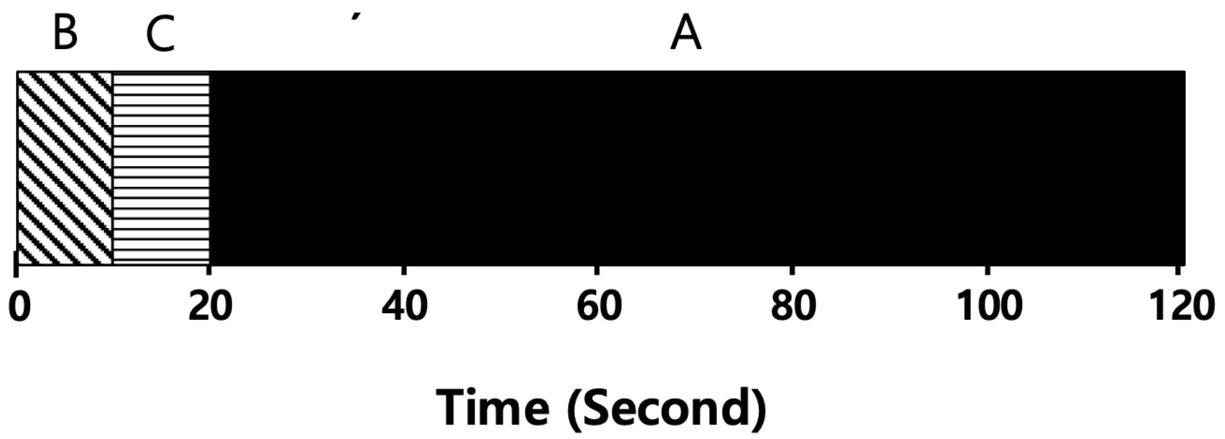
Shortest Time-to-Completion First (STCF)

- **Example:**
  - A arrives at t=0 and needs to run for 100 seconds
  - B and C arrive at t = 10 and each need to run for 10 seconds



Priority Scheduling

- Scheduling where jobs with highest priority go first
- Shortest Job First : example of this type of scheduling, priority is inverse length of job
- Priority could be based on any criteria



Priority Scheduling Example

- A arrives just before B, which arrives just before C
- A : takes 10 seconds, priority 3 (low)
- B : takes 20 seconds, priority 2
- C : takes 10 seconds, priority 1 (high)

Problem with Priority Scheduling

- Indefinite blocking (starvation): Process that is ready to run but waiting for CPU can be considered blocked.
  - What happens if you have a few low priority processes, and a steady stream of high priority processes?
  - Low priority processes could potentially remain blocked forever

Solution to Indefinite Blocking

- Aging : gradually increasing the priority of processes as they sit in the system, blocked

Priority Scheduling Example

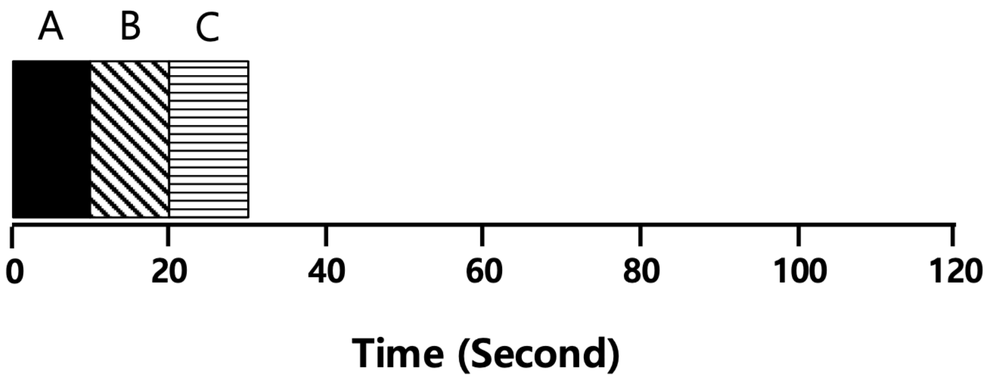
- A arrives just before B, which arrives just before C
- A : takes 10 seconds, priority 3 (low)
- B : takes 20 seconds, priority 2
- C : takes 10 seconds, priority 1 (high)
- Aging : priority increases every 10 seconds blocked

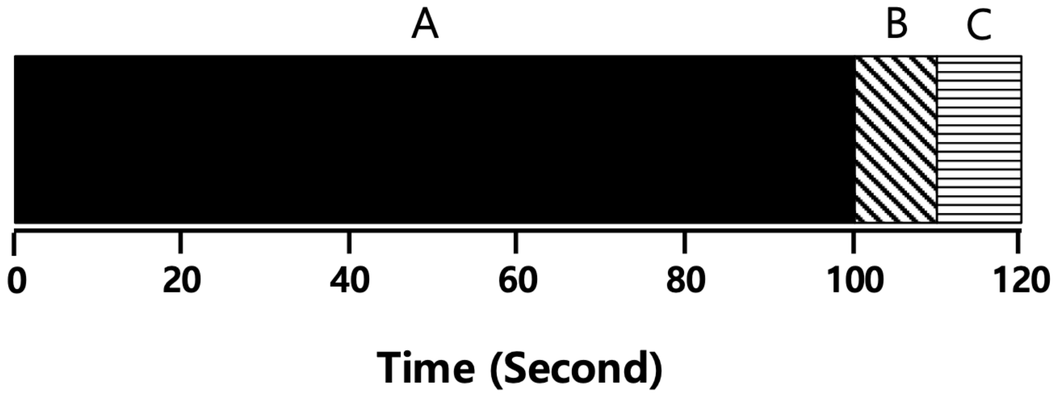
New scheduling metric : Response time

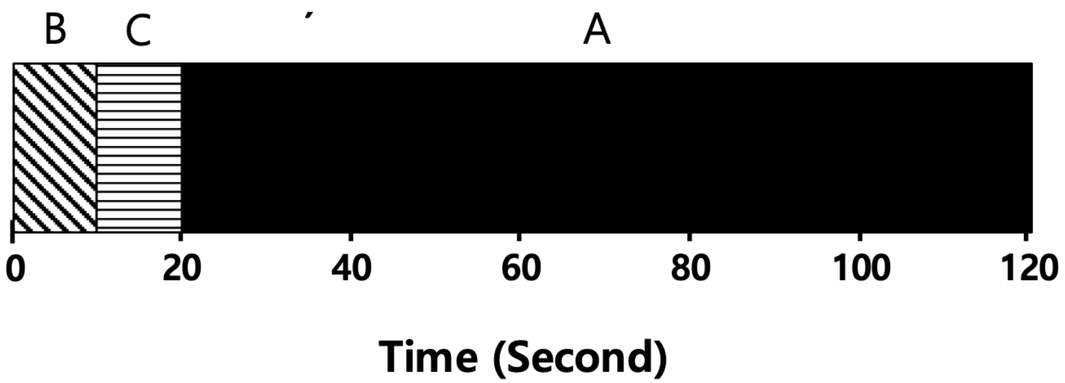
- **The time from when the job arrives to the first time it is scheduled**

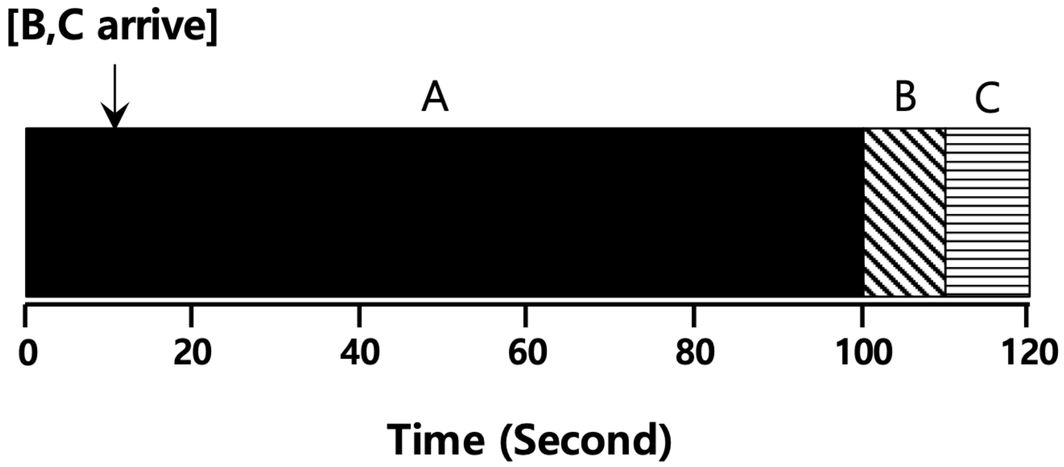
$$T_{response} = T_{firstrun} - T_{arrival}$$

New scheduling metric : Response time









New scheduling metric : Response time

- **The time from when the job arrives to the first time it is scheduled**

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

- **STCF and related disciples are not particularly good for response time**
- How can we build a scheduler that is sensitive to response time?

Round Robin (RR) Scheduling

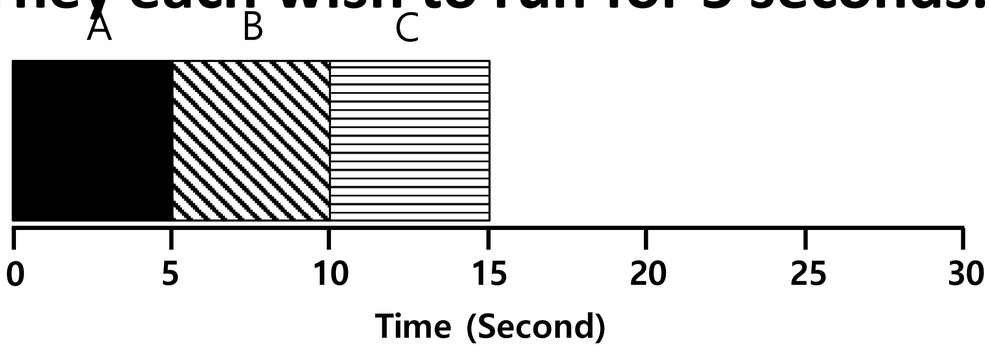
- Time slice scheduling
  - Run a job for a time slice and then switch to the next job in the run queue until the jobs are all finished
  - Sometimes called a scheduling quantum
  - Repeatedly runs slices of jobs until all jobs are finished
  - Length of a time slice must be **a multiple of** the timer-interrupt period
- **Is RR fair? Is turnaround time optimal? What about response time?**

RR Scheduling Example

- A, B and C arrive at the same time.
- They each wish to run for 5 seconds.

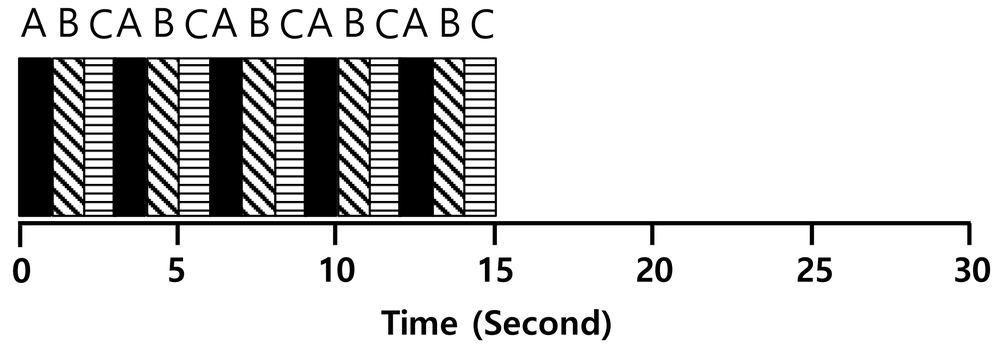
RR Scheduling Example

- A, B and C arrive at the same time.
- They each wish to run for 5 seconds.



SJF (Bad for Response Time)

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



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

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

Length of time slice

- Shorter time slice :
  - Better response time
  - Cost of context switching will dominate overall performance
- Longer time slice
  - Amortize the cost of switching
  - Worse response time
- **System designer will weigh this trade-off**

Priority Scheduling + RR

- Job with highest priority executed first
- If multiple jobs with same priority, they are executed with RR

Priority + RR Example

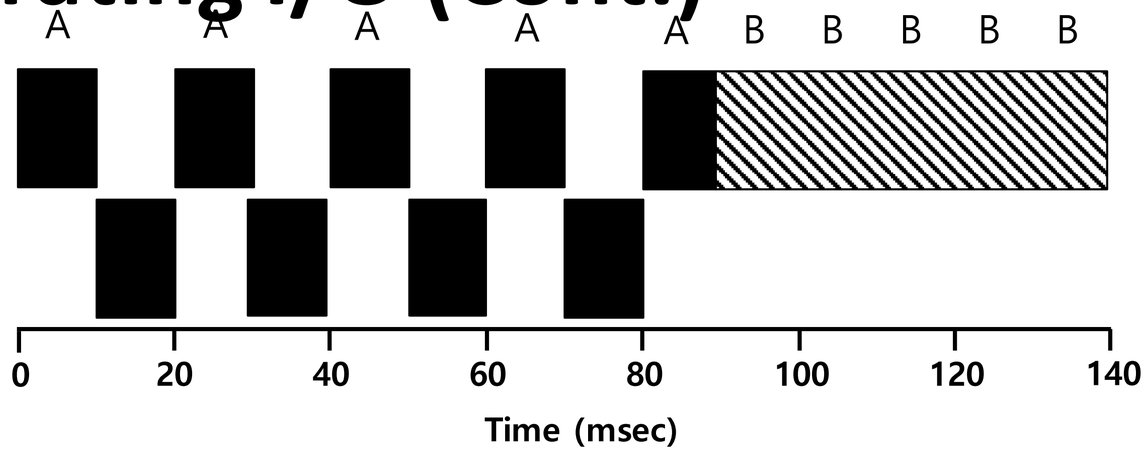
- A arrives just before B, which arrives just before C
- A : takes 10 seconds, priority 2
- B : takes 20 seconds, priority 2
- C : takes 10 seconds, priority 1 (high)

Incorporating I/O

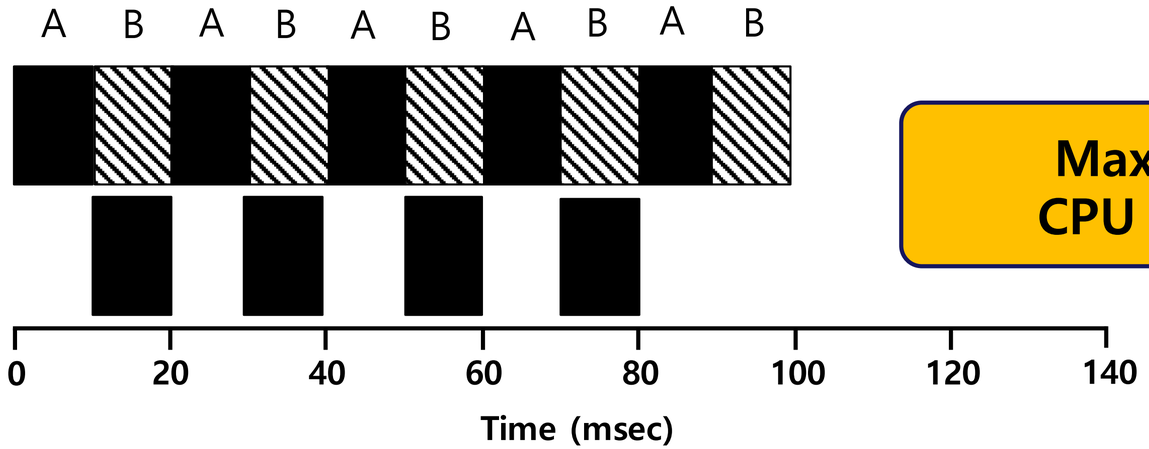
- Let’s relax assumption 3 : All programs perform I/O
- **Example :**
  - A and B need 50 ms of CPU time each
  - A runs for 10s and then issues I/O request
    - I/O each take 10ms
  - B simply uses the CPU for 50ms and performs no I/O
  - The scheduler runs A first, then B after

Incorporating I/O Example

Incorporating I/O (Cont.)



Poor Use of Resources



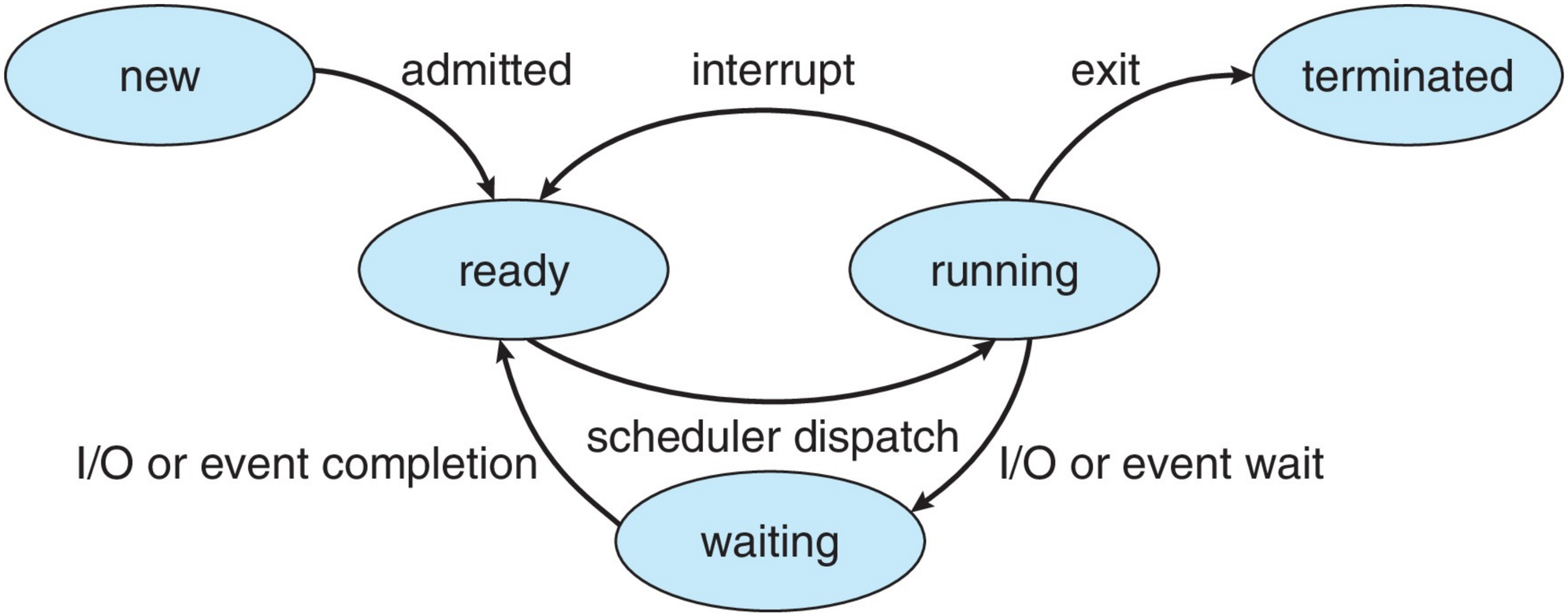
Overlap Allows Better Use of Resources

Maximize the CPU utilization

Incorporating I/O (Cont.)

- When a job initiates an I/O request:
  - The job is blocked waiting for I/O completion
  - The scheduler should schedule another job on the CPU
- When I/O completes:
  - An interrupt is raised
  - The OS moves the process from blocked back to ready state

Review : Life of a Process



For Tuesday

- 5.3.5 Multilevel Queue Scheduling : Pg 214 - 217
- Also covering fairness in scheduling, **no reading**