Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - Homework 6 is due **this Friday**
  - Quiz 1 and Lab 2 due on Tuesday 2/28 at 11:59 pm
  - Project 2 is due Friday 3/17 at 11:59 pm
- Midterm exam on Thursday 3/2
  - In-person, on paper, closed book
  - Study guide, old exam, and practice Midterm on Canvas
- Lost points because autograder or simple mistake?
  - please reach out to Grader TA over Piazza
- Navigating the Panopto Videos
  - Video contents
  - Search in captions

Previous lecture …

- Sleepy Barbers solution using Condition Variables

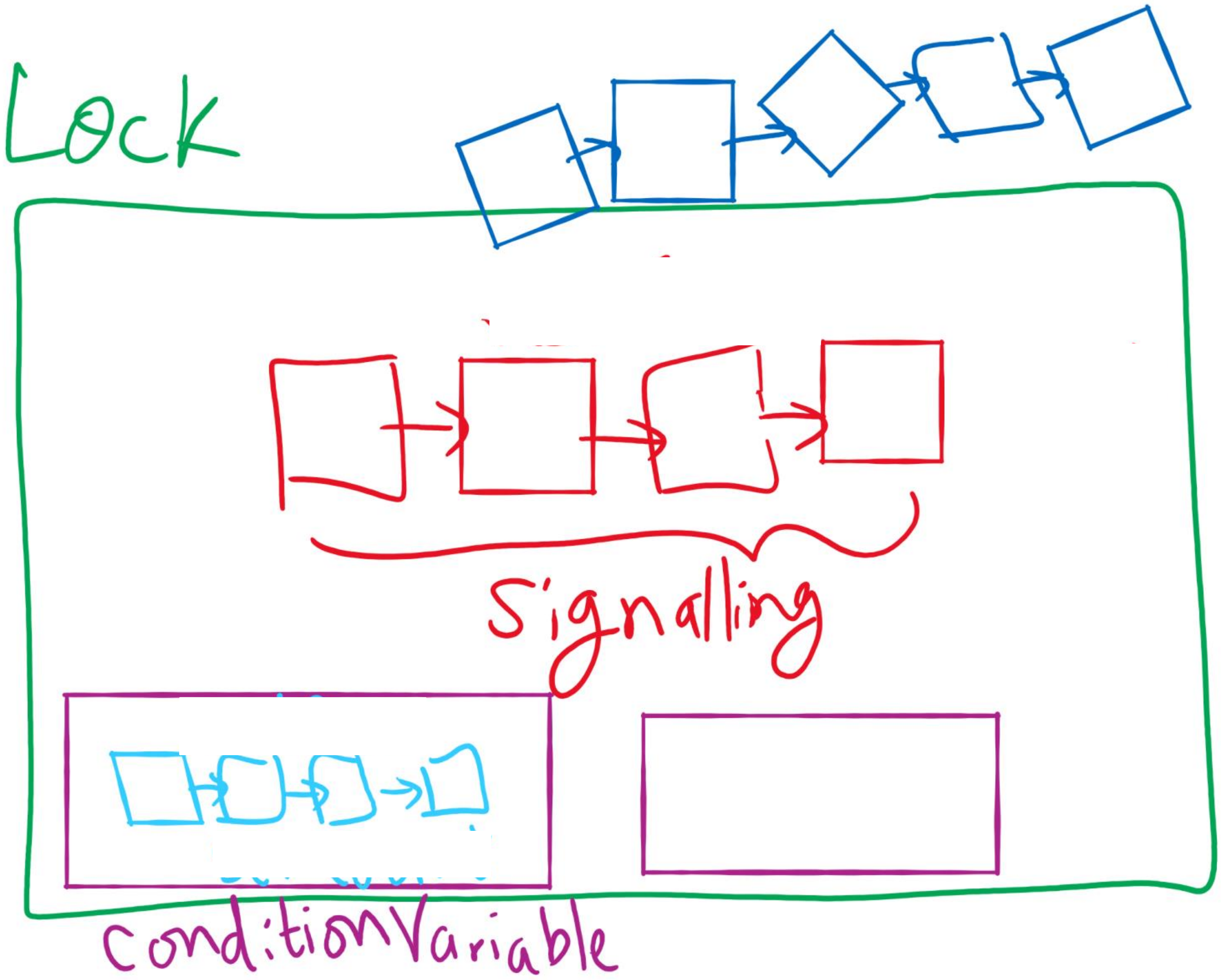
Today ...

- How to implement condition variables
- Reflections on using semaphores and condition variables
- CPU Scheduling

User-level implementation of Condition Variables

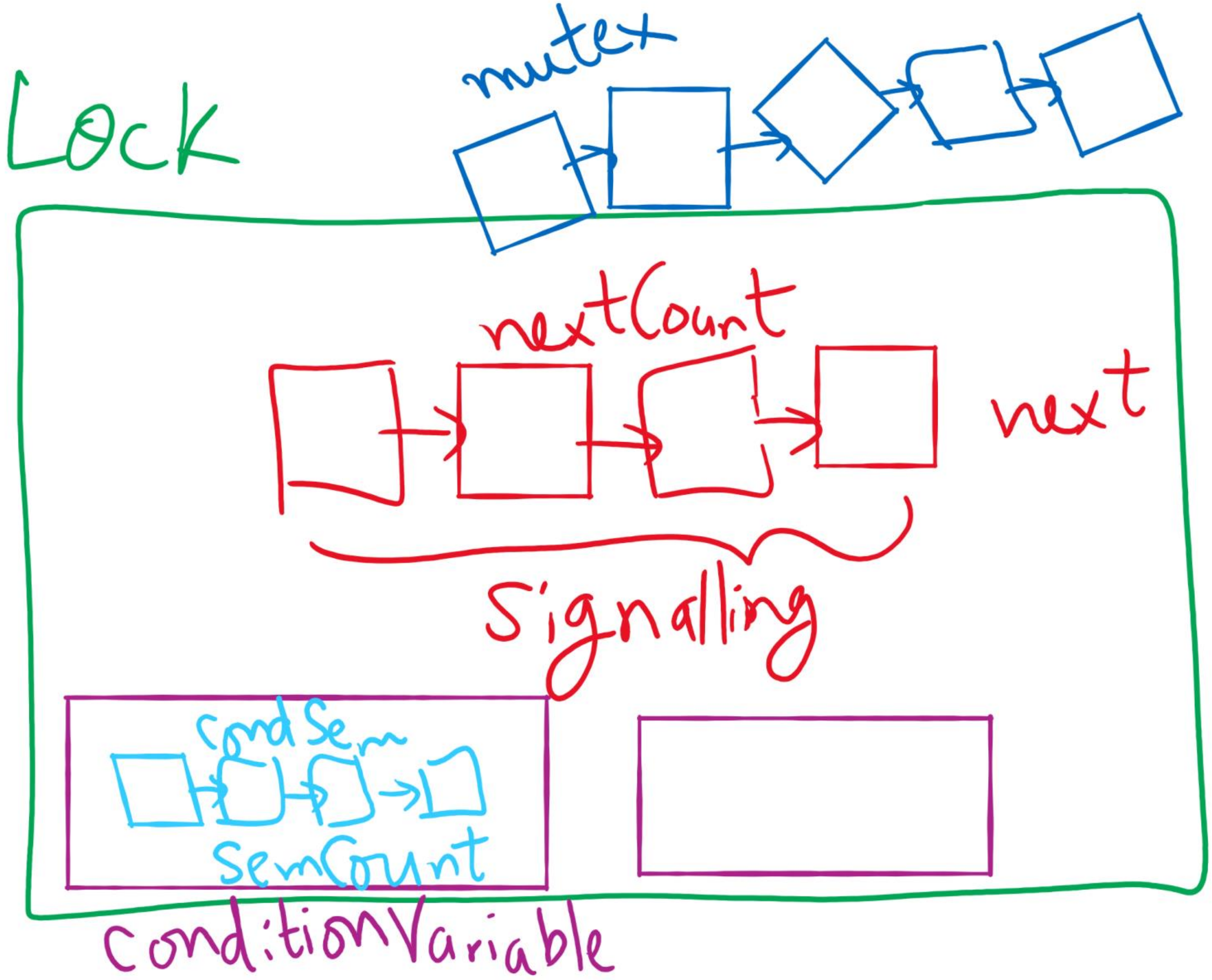
- Why?
  - Some operating systems don't have condition variables
  - Another exercise on solving synchronization problems with semaphores
- What are the waiting situations?
  - waiting to acquire the lock
  - waiting on the condition variable
  - waiting after signaling on a condition variable
    - This is Hoare semantics
      - signaling process waits
    - Compare to Mesa semantics
      - signaled process waits

Lock and Condition Variable Implementation



Lock and Condition Variable Implementation

- Let's have a semaphore for each waiting situation



User-level implementation of Condition Variables

A Lock with two waiting queues

```
struct Lock {
  Semaphore mutex(1);
   Semaphore next(0);
  int nextCount = 0;
}
```

Acquire(){

```
  mutex.down();
}
```

Release(){

```
   if(nextCount > 0){
      next.up();
      nextCount--;
  } else mutex.up();
}
```

Condition Variable

```
struct ConditionVariable {
   Semaphore condSem(0);
   int semCount = 0;
   Lock *lk;
}
```

```
Wait(){
  if(lk->nextCount > 0)
    lk->next.up();
    lk->nextCount--;
  else lk->mutex.up();
  semCount++;
  condSem.down();
  semCount--;
}
```

```
Signal(){
  if(semCount > 0){
    condSem.up()
    lk->nextCount++
    lk->next.down();
    lk->nextCount—
  }
}
```

Let’s trace our solution!

- Note: Monitor is another name for Lock

P

enter monitor

while(condition)
cv.wait()

exit monitor

Q

enter monitor

change variables to make condition false
cv.signal()

exit monitor

Reflections on semaphore usage

- Semaphores can be used as
  - Resource counters
  - Waiting spaces
  - For mutual exclusion

Reflections on Condition Variables

- Define a class and put all shared variables inside the class
- Include a mutex and a condition variable in the class
- For each public method of the class
  - Start by locking the mutex lock
  - If need to wait, use a while loop and wait on the condition variable
  - Before **<u>broadcasting</u>** on the condition variable, make sure to change the waiting condition

Final Remarks on Process Synchronization

- Many other synchronization mechanisms
  - Message passing
  - Barriers
  - Futex
  - Re-entrant locks
  - AtomicInteger, AtomicX

Problem of the Day: CPU Scheduling

How does the ***short-term scheduler*** select the next process to run?

CPU Scheduling

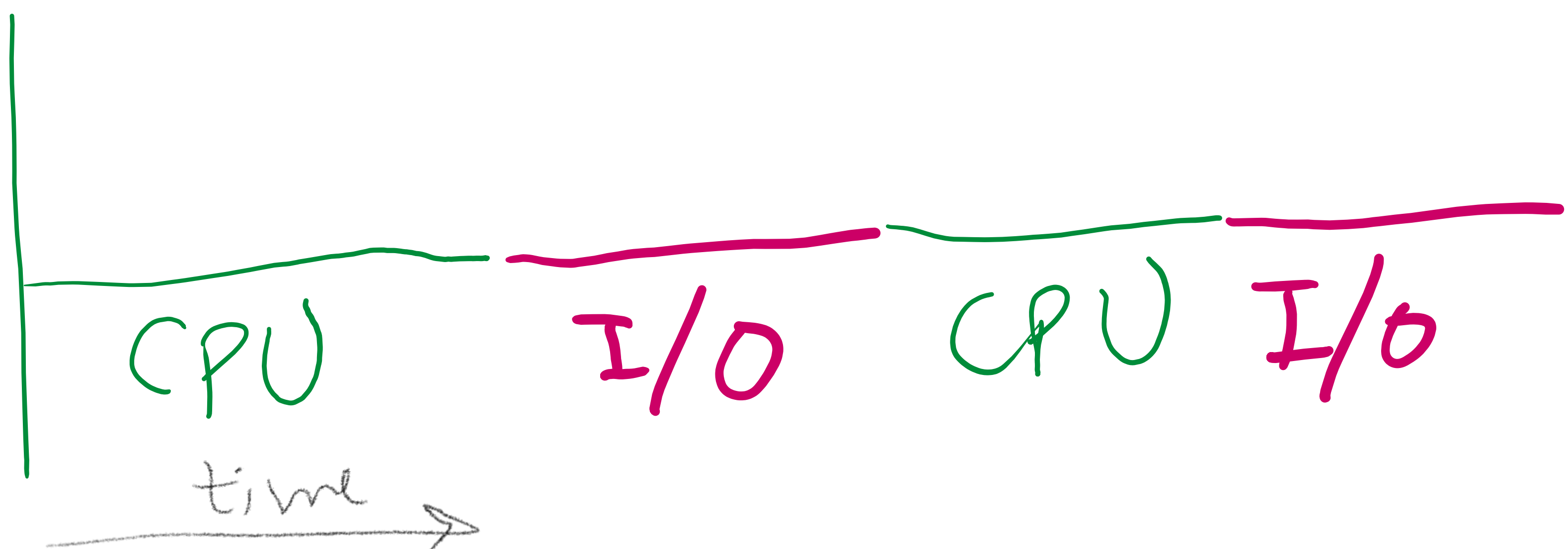
- Scheduling the processor among all ready processes
- User-oriented criteria
  - Response Time: Elapsed time between the submission of a request and the receipt of a response
  - Turnaround Time: Elapsed time between the submission of a process to its completion
- System-oriented criteria
  - Processor utilization
  - Throughput: number of process completed per unit time
  - Fairness

Short-Term Scheduler Dispatcher

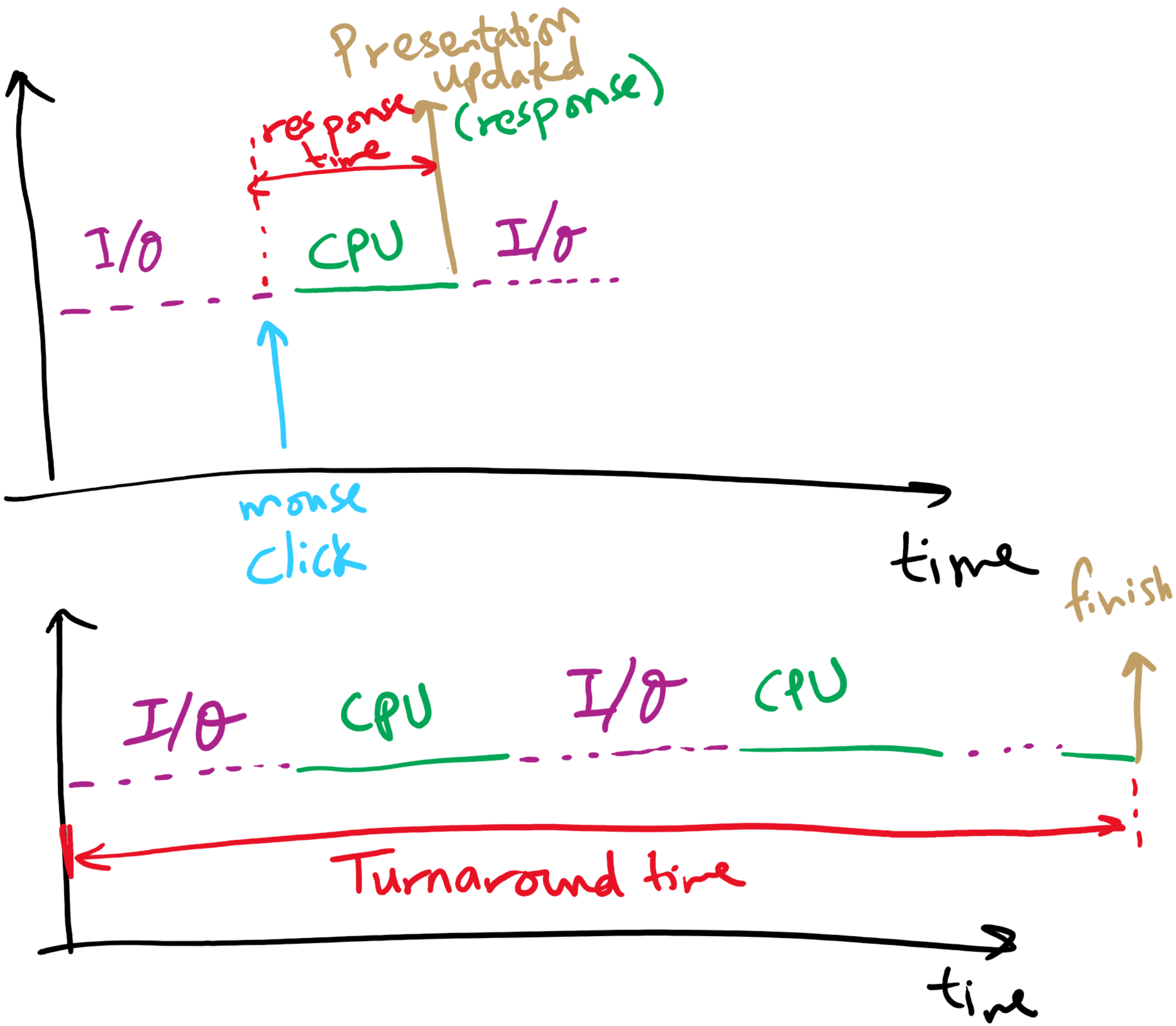
- The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler
- The functions of the dispatcher include:
  - Switching context
  - Switching to user mode
  - Jumping to the location in the user program to restart execution
- The dispatch latency must be minimal

The CPU-I/O Cycle

- Processes require alternate use of processor and I/O in a repetitive fashion
- Each cycle consist of a CPU burst followed by an I/O burst
  - A process terminates on a CPU burst
- CPU-bound processes have longer CPU bursts than I/O-bound processes



Response time vs. Turnaround time



Scheduling Algorithms

- First-Come, First-Served Scheduling
- Shortest-Job-First Scheduling
  - Also referred to as Shortest Process Next
- Priority Scheduling
- Round-Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

Characterization of Scheduling Policies

- The selection function determines which ready process is selected next for execution
- The decision mode specifies the instants in time the selection function is exercised
  - Nonpreemptive
    - Once a process is in the running state, it will continue until it terminates or blocks for an I/O
  - Preemptive
    - Currently running process may be interrupted and moved to the Ready state by the OS
    - Prevents one process from monopolizing the processor

Process Mix Example

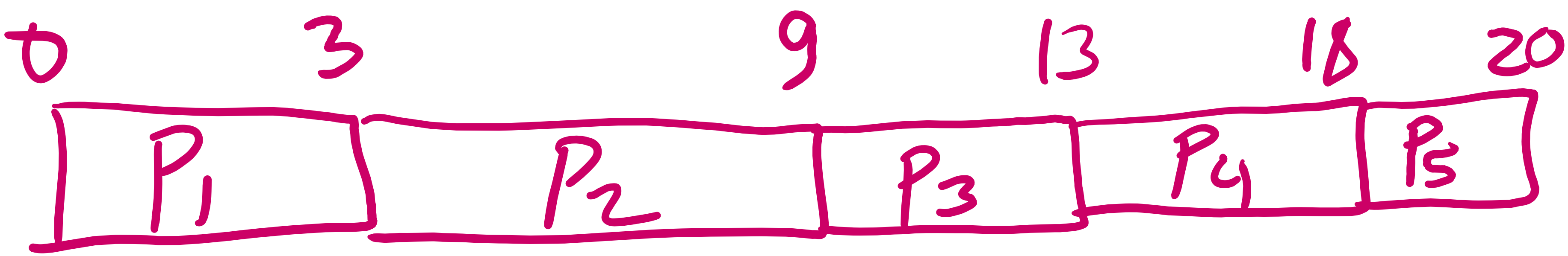
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

Service time = total processor time needed in one (CPU-I/O) cycle
Jobs with long service time are CPU-bound jobs and are referred to as "long jobs"

First Come First Served (FCFS)

- Selection function: the process that has been waiting the longest in the ready queue (hence, FCFS)
- Decision mode: non-preemptive
  - a process runs until it blocks for an I/O

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Average Response Time

$$\text{Average Response Time} = \frac{\overset{P_1}{(3-0)} + \overset{P_2}{(9-2)} + \overset{P_3}{(13-4)} + \overset{P_4}{(18-6)} + \overset{P_5}{(20-8)}}{5}$$

FCFS drawbacks

- Favours CPU-bound processes
  - CPU-bound processes monopolize the processor
  - I/O-bound processes have to wait until completion of CPU-bound process
    - I/O-bound processes may have to wait even after their I/Os are completed (poor device utilization)
    - Convoy effect
  - Better I/O device utilization could be achieved if I/O bound processes had higher priority

Convoy Effect