CS 471 Operating Systems

Yue Cheng

George Mason University

Fall 2019

Announcement

- OS/161 PA1 posed on BB
  - Due 11:59PM 10/18

Review: CV vs. Semaphores

CV != Semaphores

- Condition variables != semaphores
  - Although their operations have similar names, they have entirely different semantics
  - However, they each can be used to implement the other
- Access to the CV is controlled by a lock
  - `wait()` blocks the caller, who gives up the lock
    - `Semaphore::wait()` just blocks the thread on the queue
  - `signal()` causes a waiting thread to wake up
    - If no waiting thread, the signal is lost
    - `Semaphore::post()` increases the semaphore count, allowing future entry even if no thread is waiting
    - CV has no history

CV vs. Semaphores

- CV rule of thumb:
  - Keep state in addition to CV
    - Empty, full in PCP
  - Always do wait and signal while holding a lock
  - Whenever you acquire a lock, re-check state
    - by using `while()` instead of `if()`

CV vs. Semaphores

- CV rule of thumb:
  - Keep state in addition to CV
    - Empty, full in PCP
  - Always do wait and signal while holding a lock
  - Whenever you acquire a lock, re-check state
    - by using `while()` instead of `if()`

- How do semaphores eliminate these needs?

CV vs. Semaphores

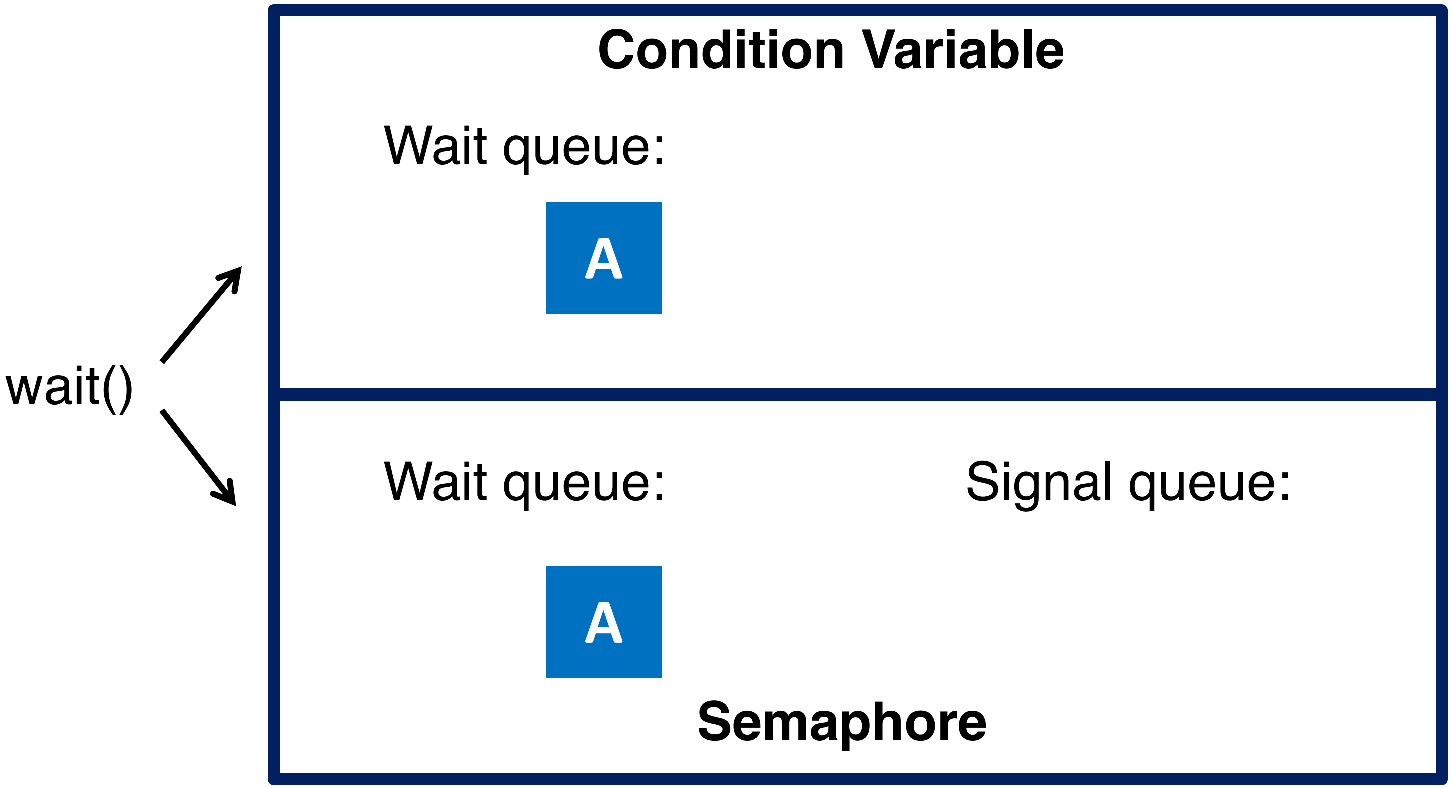
Condition Variable

Wait queue:

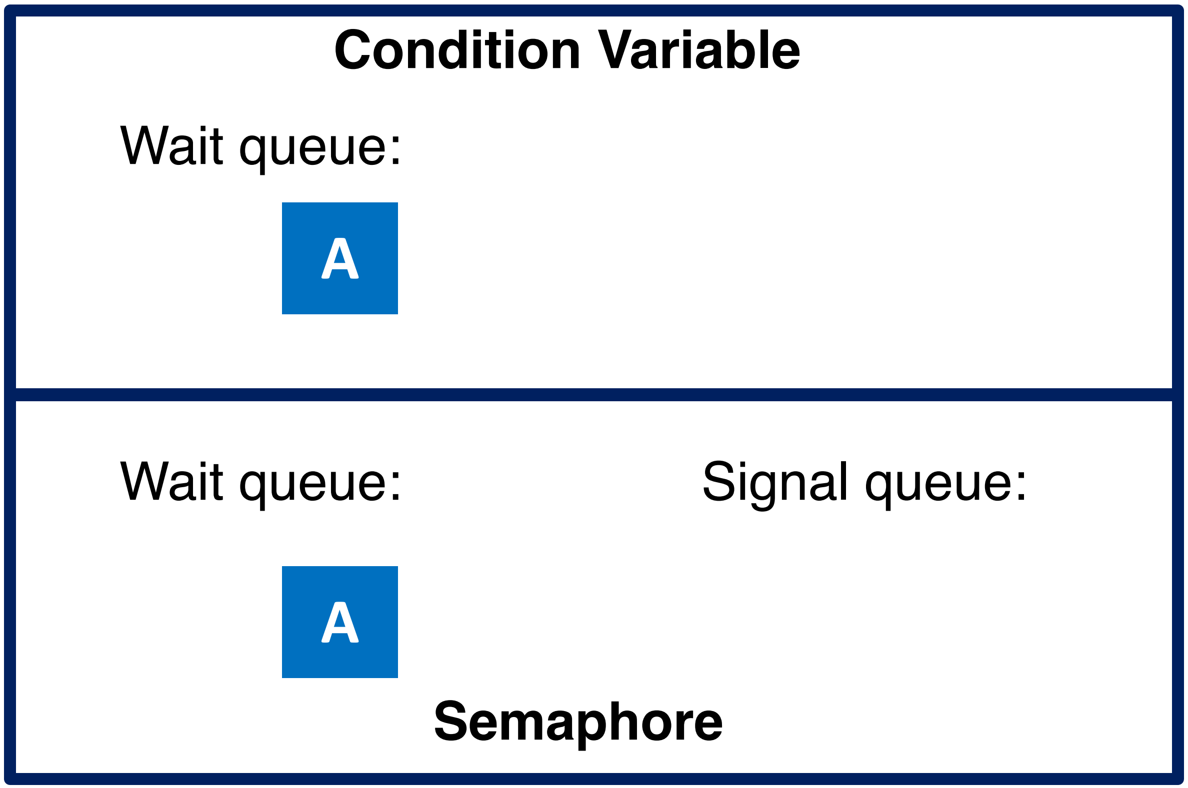
Wait queue: Signal queue:

Semaphore

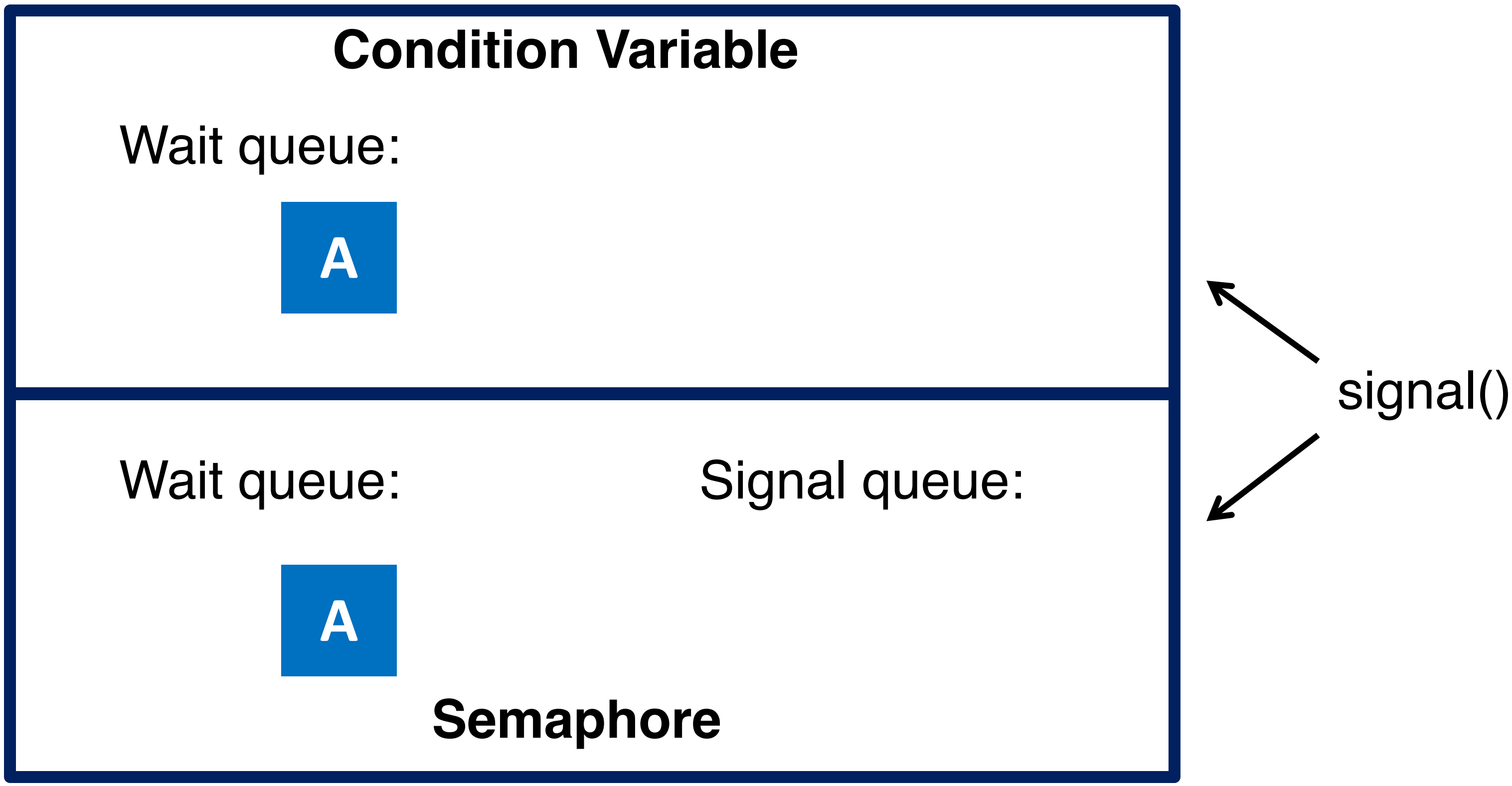
CV vs. Semaphores



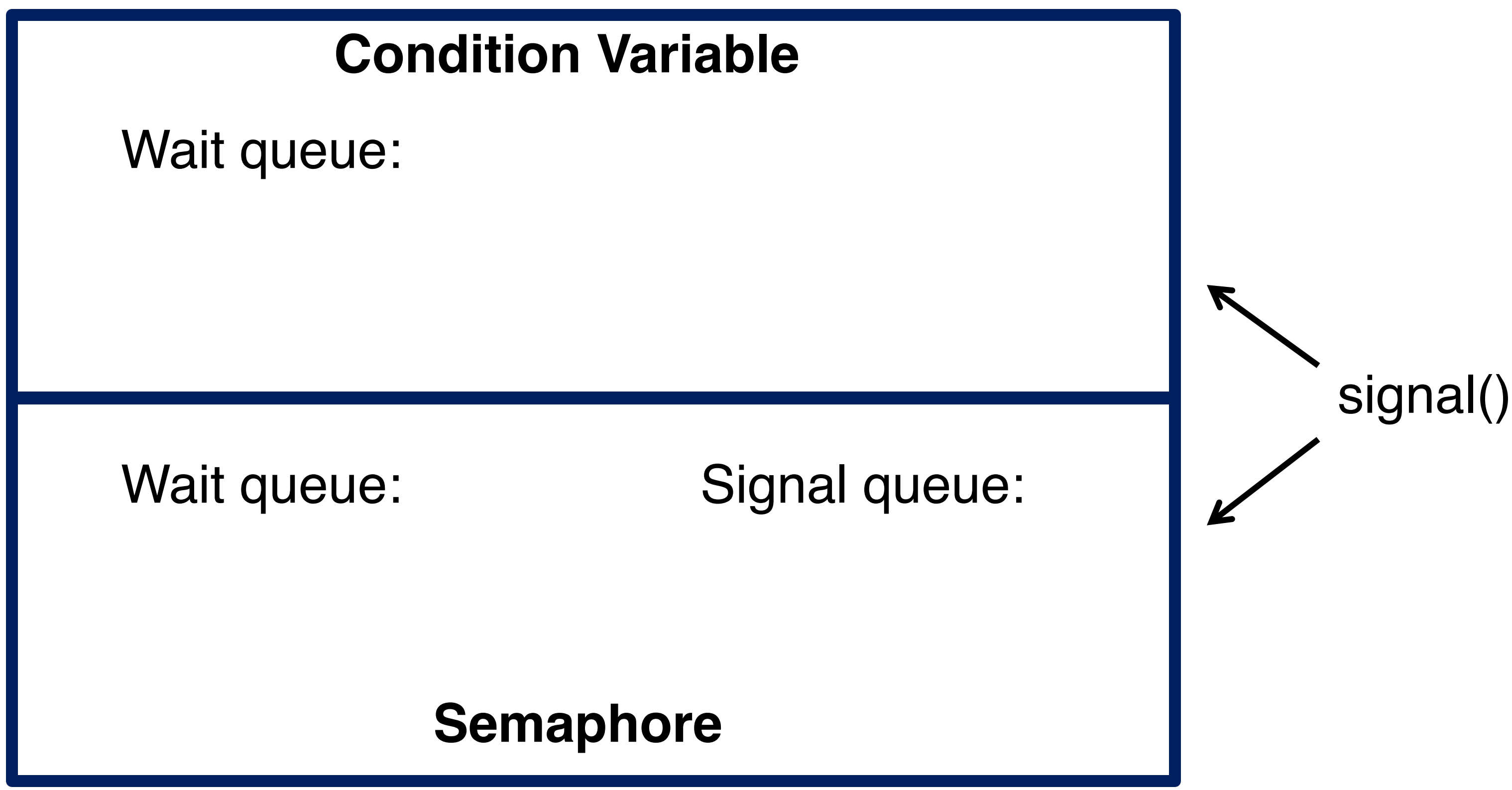
CV vs. Semaphores



CV vs. Semaphores



CV vs. Semaphores



CV vs. Semaphores

Condition Variable

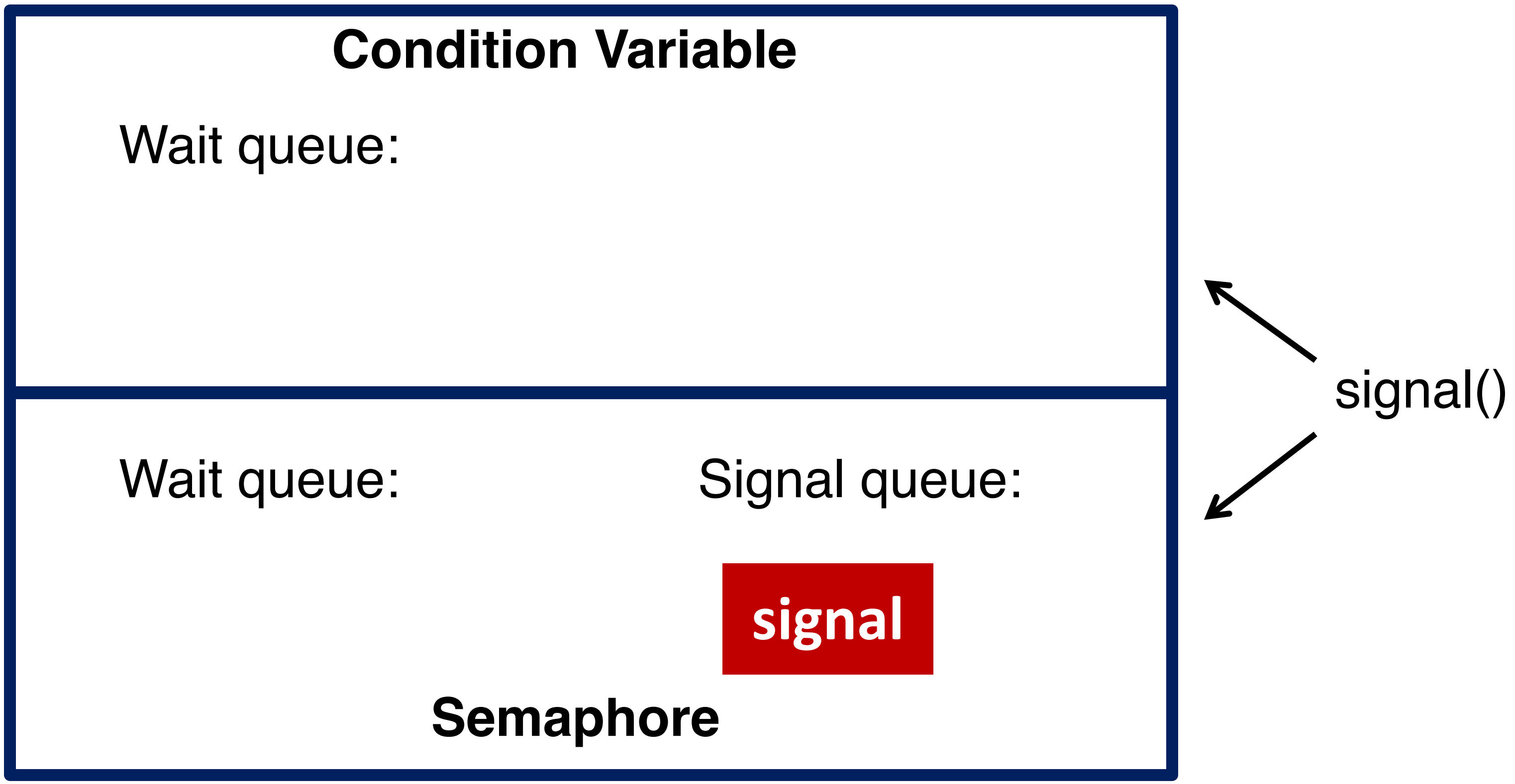
Wait queue:

Wait queue:

Signal queue:

Semaphore

CV vs. Semaphores



CV vs. Semaphores

Condition Variable

Wait queue:

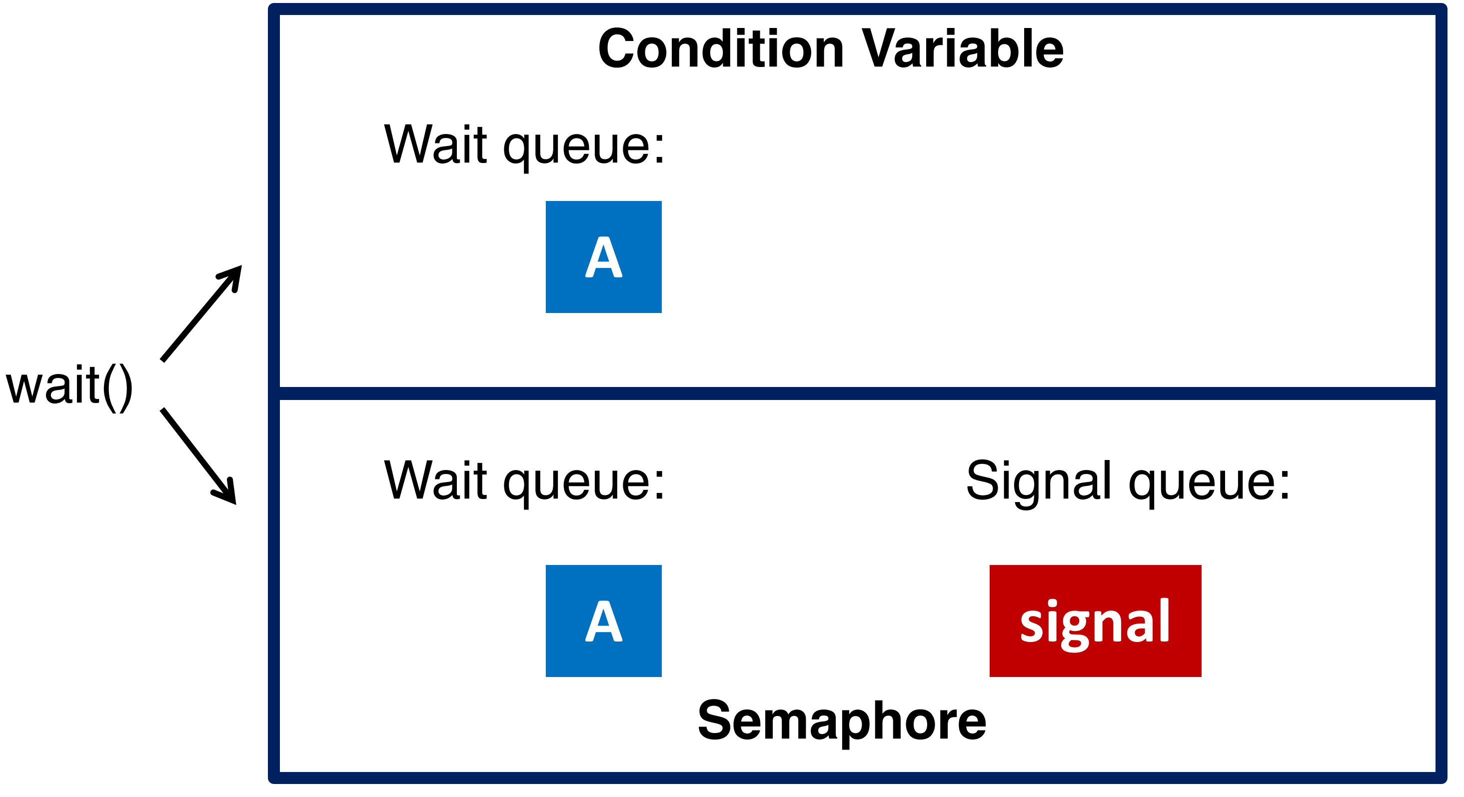
Wait queue:

Signal queue:

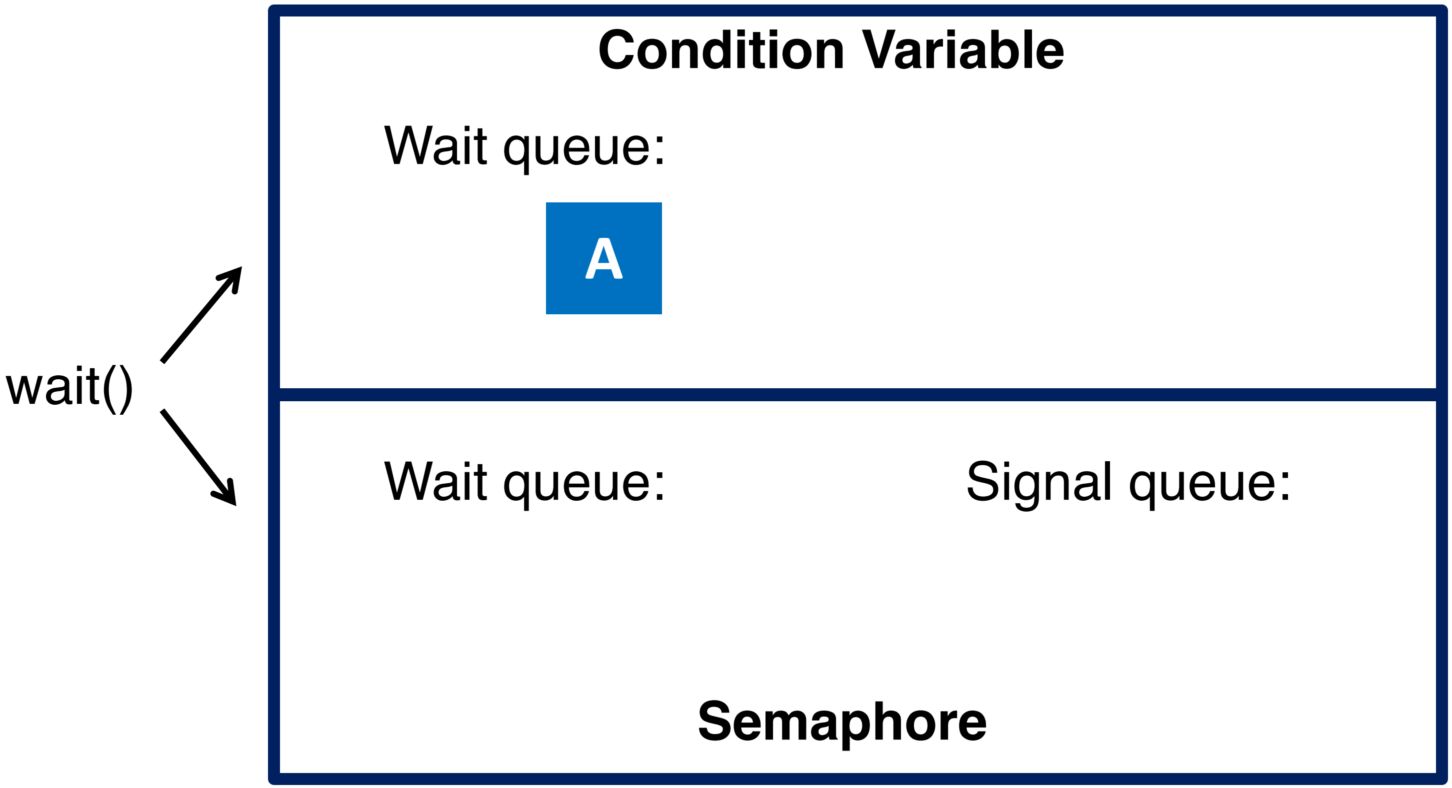
signal

Semaphore

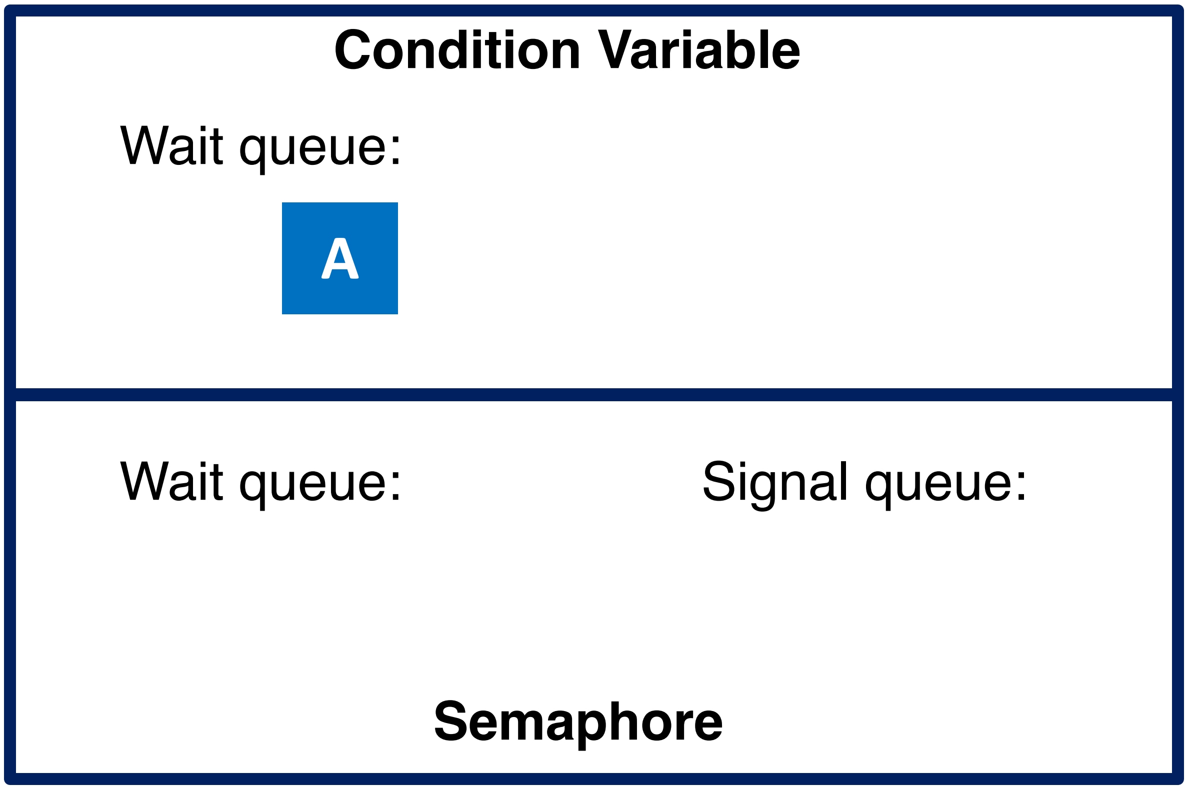
CV vs. Semaphores



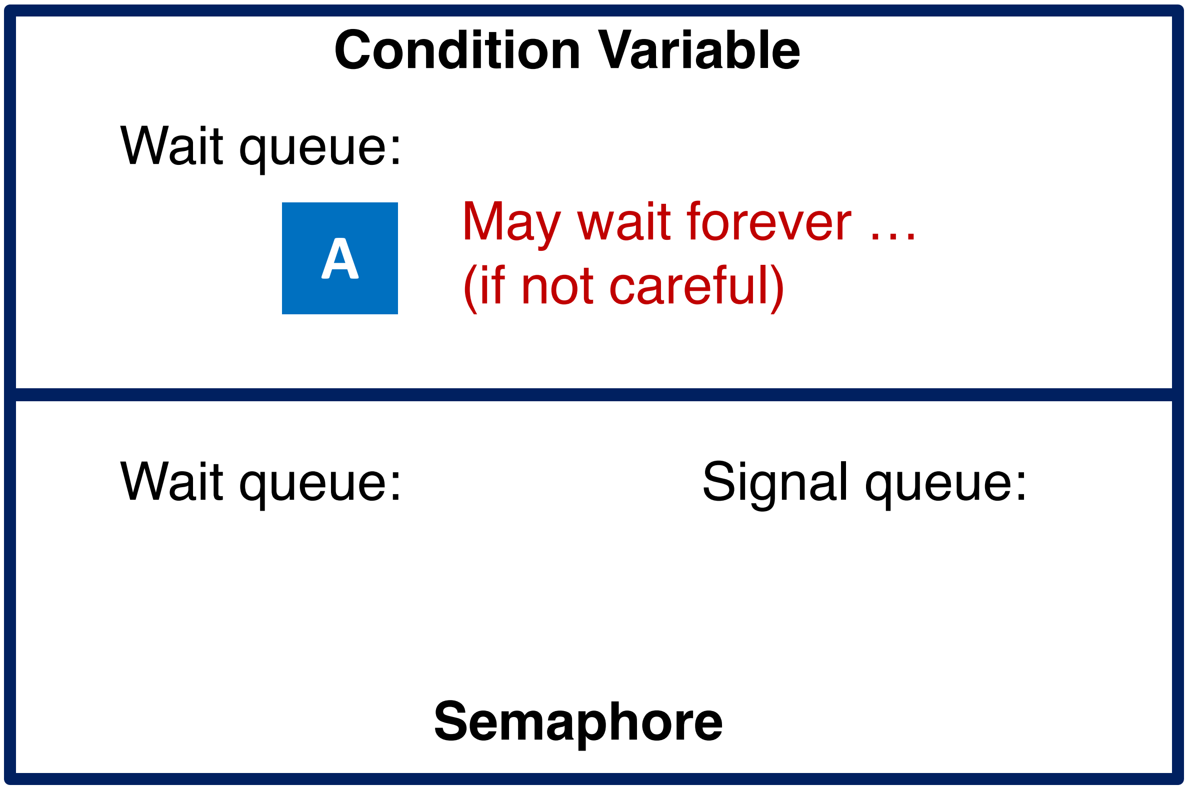
CV vs. Semaphores



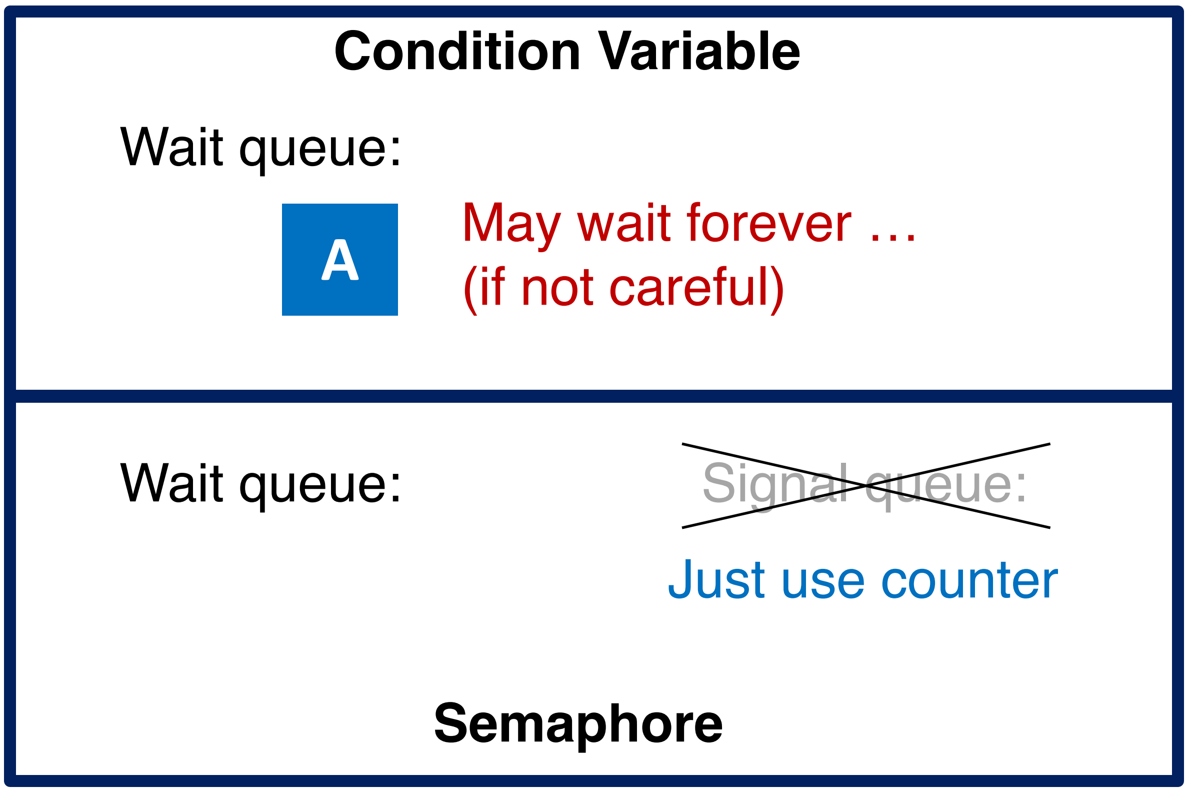
CV vs. Semaphores



CV vs. Semaphores



CV vs. Semaphores



CPU Scheduling

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
  - First-In-First-Out
  - Shortest-Job-First, Shortest-Remaining-Time-First
  - Priority Scheduling
  - Round Robin
  - Multi-level Queue
  - Multi-level Feedback Queue

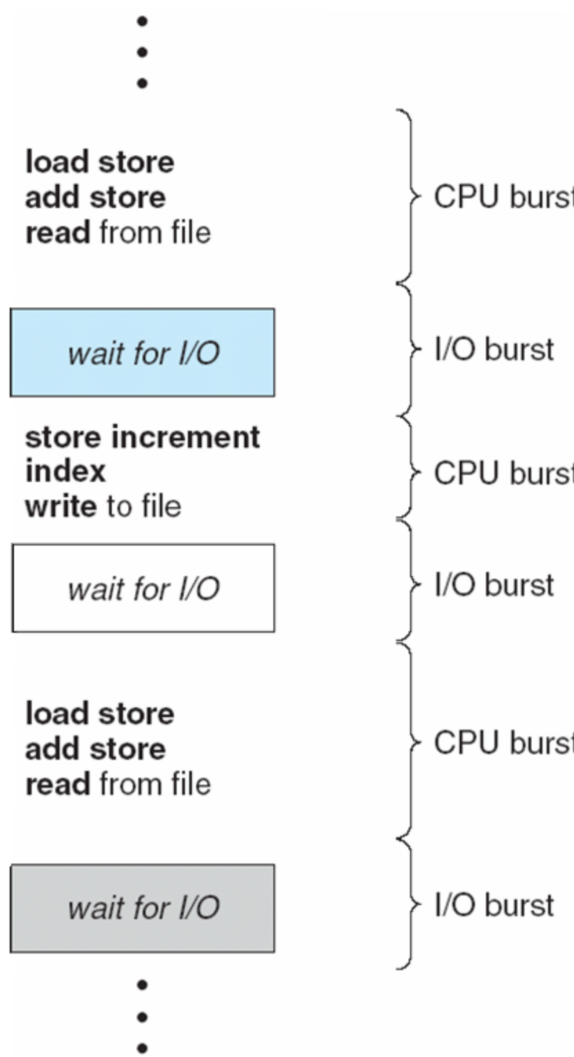
Basic Concepts

- During its lifetime, a process goes through a sequence of CPU and I/O bursts

- The CPU scheduler (a.k.a. short-term scheduler) will select one of the processes in the ready queue for execution

- The CPU scheduler algorithm may have tremendous effects on the system performance
  - Interactive systems: Responsiveness
  - Real-time systems: Not missing the deadlines

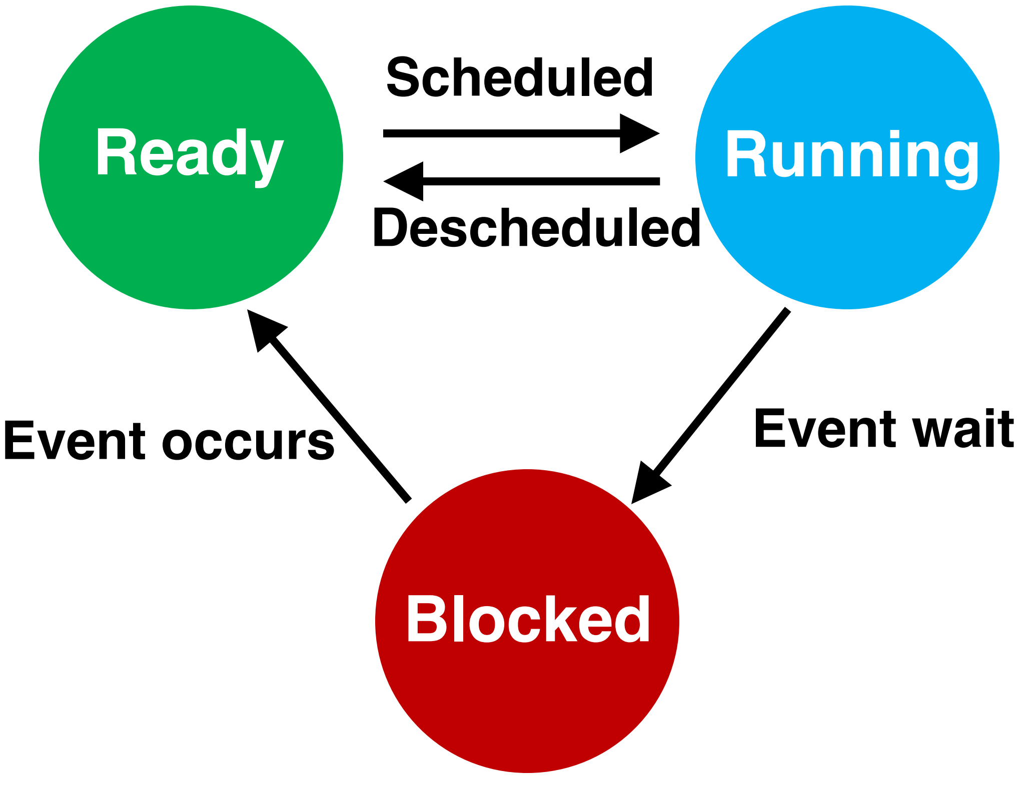
Alternating Sequence of CPU and I/O Bursts



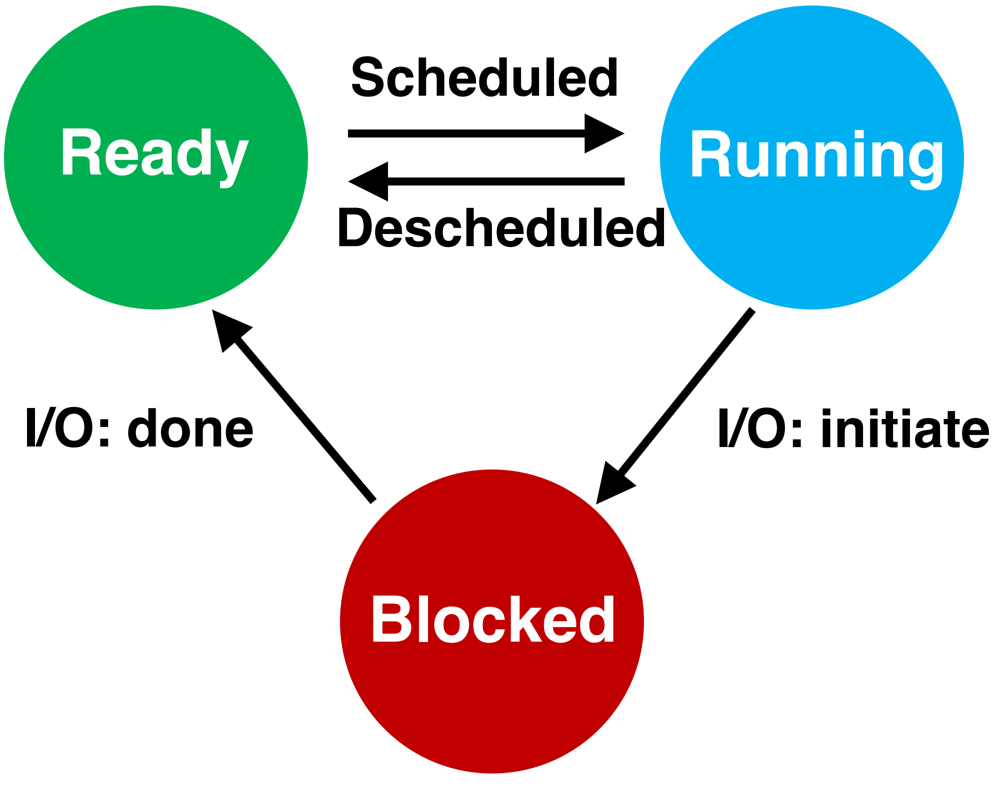
When to Schedule?

- Under the simple process state transition model, CPU scheduler can be potentially invoked at five different points:
  1. When a process switches from the new state to the ready state
  2. When a process switches from the running state to the waiting (or blocked) state
  3. When a process switches from the running state to the ready state
  4. When a process switches from the waiting state to the ready state
  5. When a process terminates

Process State Transitions



Process State Transitions



Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper (previously saved) location in the user program to restart that program

- Scheduler → Policy: When and how to schedule
- Dispatcher → Mechanism: Actuator following the commands of the scheduler

Scheduling Metrics

- To compare the performance of scheduling algorithms
  - CPU utilization – percentage of time CPU is busy executing jobs
  - Throughput – # of processes that complete their execution per time unit
  - Turnaround time – amount of time to execute a particular process
  - Waiting time – amount of time a process has been waiting in the ready queue
  - Response time – amount of time it takes from when a request was submitted until the first response is produced, not the complete output
  - Meeting the deadlines (real-time systems)

Optimization Goals

- To **maximize**:
  - Maximize the CPU utilization
  - Maximize the throughput

- To **minimize**:
  - Minimize the (average) turnaround time
  - Minimize the (average) waiting time
  - Minimize the (average) response time

Waiting Time

- Waiting time definition

$$T_{waiting} = T_{start} - T_{arrival}$$

- Average waiting time = Sum($T_{waiting}$)/ #processes

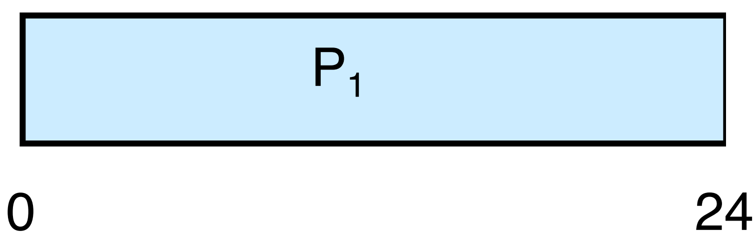
- **For now, we assume**
  - –**Average waiting time** is the performance measure
  - –Only one CPU burst (e.g., in milliseconds or ms) per process
  - –Only CPU, No I/O
  - –All processes arrive at the same time
  - –Once started, each process runs to completion

First-In-First-Out (FIFO)

First-In-First-Out (FIFO)

Process	Burst Time
P_1	24

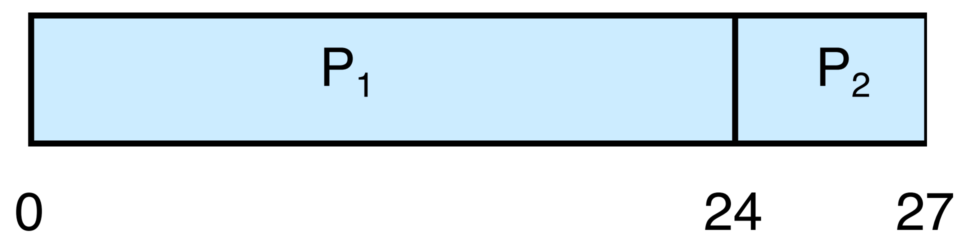
- Suppose that the processes arrive in order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule:



First-In-First-Out (FIFO)

Process	Burst Time
P_1	24
P_2	3

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



First-In-First-Out (FIFO)

Process	Burst Time
P_1	24
P_2	3
P_3	3

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

First-In-First-Out (FIFO)

Process	Burst Time
P_1	24
P_2	3
P_3	3

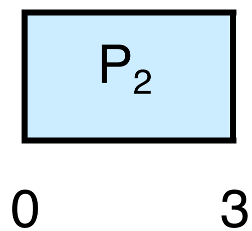
- Suppose that the processes arrive in order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: 17

FIFO (cont.)

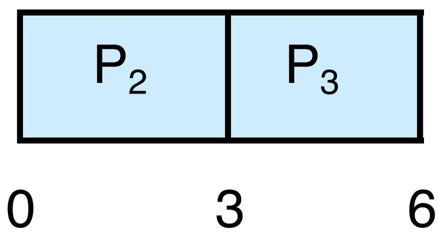
- Suppose that the processes arrive in order P_2 , P_3 , P_1

- The Gantt chart for the schedule:



FIFO (cont.)

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



FIFO (cont.)

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

FIFO (cont.)

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

- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3

FIFO (cont.)

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

- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Problems:
  - **Convoy effect** (short processes behind long processes)
  - Non-preemptive: Not suitable for time-sharing systems

Shortest-Job-First (SJF)

Shortest-Job-First (SJF)

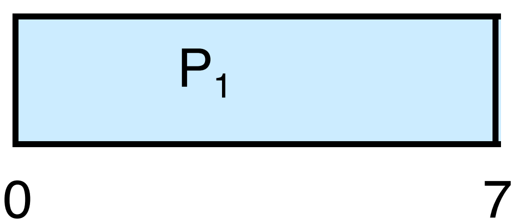
- Associate with each process the length of its next CPU burst
- The CPU is assigned to the process with the smallest (next) CPU burst (run_time)

- Two schemes (modes):
  - Non-preemptive
  - Preemptive: Also known as the **Shortest-Remaining-Time-First (SRTF)**

Example for Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



Example for Non-Preemptive SJF

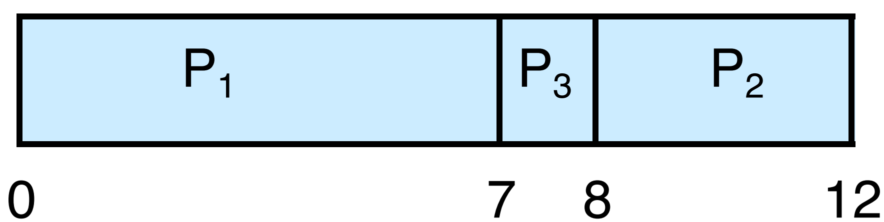
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

Example for Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

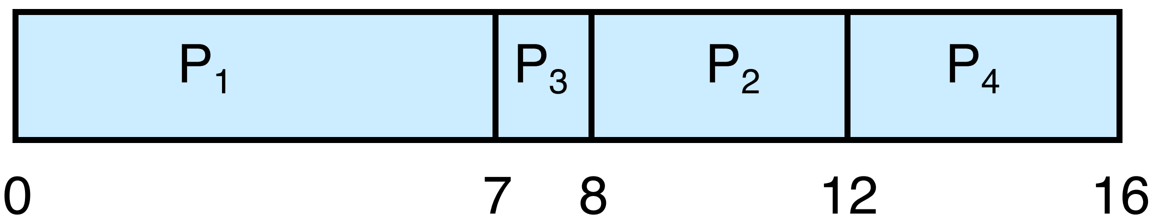
- SJF (non-preemptive)



Example for Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

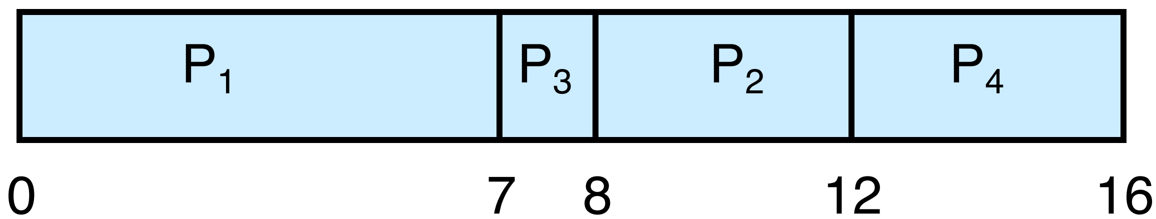
- SJF (non-preemptive)



Example for Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

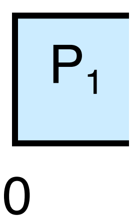
Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	

Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	5
P_2	2.0	4	

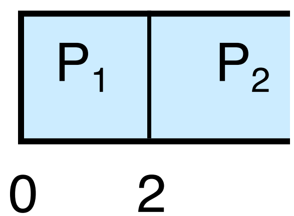
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	5
P_2	2.0	4	4

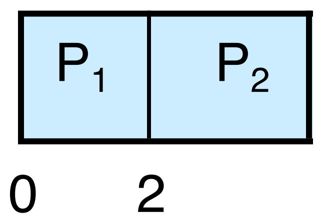
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	5
P_2	2.0	4	4
P_3	4.0	1	1

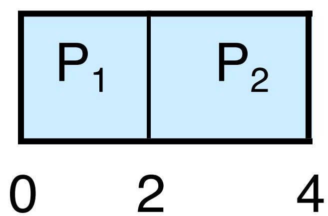
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	5
P_2	2.0	4	**2**
P_3	4.0	1	1

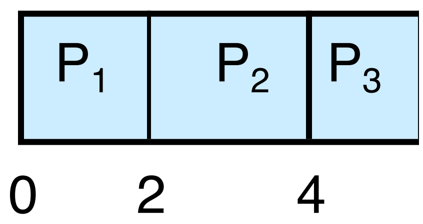
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	5
P_2	2.0	4	2
P_3	4.0	1	1

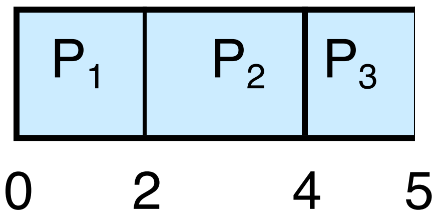
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	5
P_2	2.0	4	2
P_3	4.0	1	0
P_4	5.0	4	4

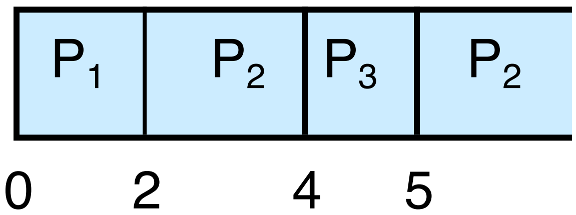
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	5
P_2	2.0	4	2
P_3	4.0	1	0
P_4	5.0	4	4

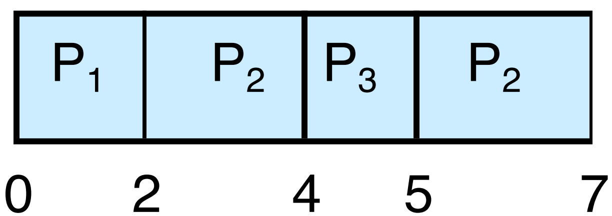
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	5
P_2	2.0	4	0
P_3	4.0	1	0
P_4	5.0	4	4

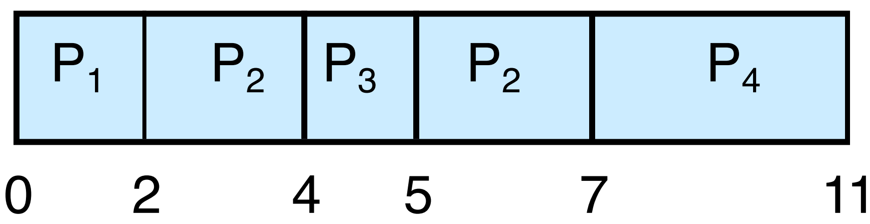
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	Left Time
P_1	0.0	7	5
P_2	2.0	4	0
P_3	4.0	1	0
P_4	5.0	4	0

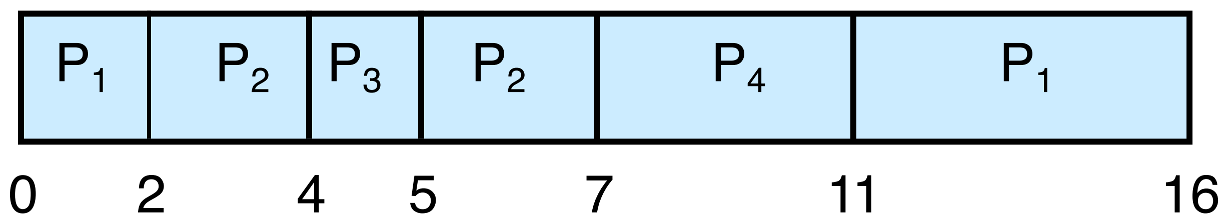
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	0
P_2	2.0	4	0
P_3	4.0	1	0
P_4	5.0	4	0

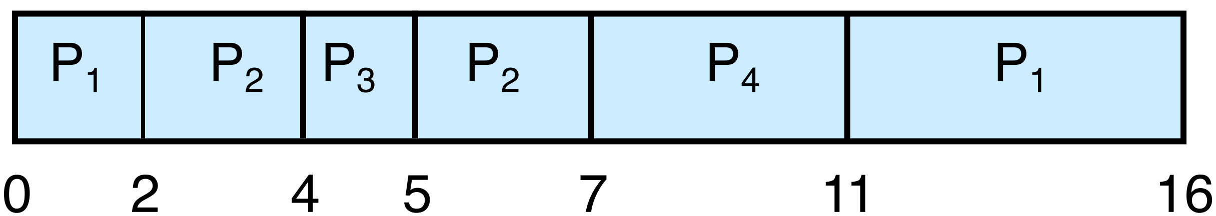
- SJF (preemptive)



Example for Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time	**Left Time**
P_1	0.0	7	0
P_2	2.0	4	0
P_3	4.0	1	0
P_4	5.0	4	0

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 + 2)/4 = 3