# OPERATING SYSTEM
# SEM IV
# UNIT - II

By,
Dr. Himani Deshpande

# OPERATING SYSTEM

| Course Code | Course Name | Credit |
|---|---|---|
| CSC404 | Operating System | 03 |

| Course Code | Course Name | Examination Scheme | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Theory Marks | | | | Term Work | Pract. /Oral | Total |
| | | Internal assessment | | | End Sem. Exam | | | |
| | | Test1 | Test 2 | Avg. | | | | |
| ITC403 | Operating System | 20 | 20 | 20 | 80 | -- | -- | 100 |

# Course Outcome

1. Understand the objectives, functions and structure of OS
2. Analyze the concept of process management and evaluate performance of processscheduling algorithms.
3. Understand and apply the concepts of synchronization and deadlocks
4. Evaluate performance of Memory allocation and replacement policies
5. Understand the concepts of file management.
6. Apply concepts of I/O management and analyze techniques of disk schedulin

# SYLLABUS

| Module | Detailed Content | | Hours |
|---|---|---|---|
| **1** | **Operating system Overview** | | **4** |
| | 1.1 | Introduction, Objectives, Functions and Evolution of Operating System | |
| | 1.2 | Operating system structures: Layered, Monolithic and Microkernel | |
| | 1.3 | Linux Kernel, Shell and System Calls | |
| **2** | **Process and Process Scheduling** | | **9** |
| | 2.1 | Concept of a Process, Process States, Process Description, Process Control Block. | |
| | 2.2 | Uniprocessor Scheduling-Types: Preemptive and Non-preemptive scheduling algorithms (FCFS, SJF, SRTN, Priority, RR) | |
| | 2.3 | Threads: Definition and Types, Concept of Multithreading | |
| **3** | **Process Synchronization and Deadlocks** | | **9** |
| | 3.1 | Concurrency: Principles of Concurrency, Inter-Process Communication, Process Synchronization. | |
| | 3.2 | Mutual Exclusion: Requirements, Hardware Support (TSL), Operating System Support (Semaphores), Producer and Consumer problem. | |
| | 3.3 | Principles of Deadlock: Conditions and Resource, Allocation Graphs, Deadlock Prevention, Deadlock Avoidance: Banker‟s Algorithm, Deadlock Detection and Recovery, Dining Philosophers Problem. | |

# Syllabus

| | | | |
|---|---|---|---|
| | | Deadlock Detection and Recovery, Dining Philosophers Problem. | |
| **4** | **Memory Management** | | **9** |
| | 4.1 | Memory Management Requirements, Memory Partitioning: Fixed, Partitioning, Dynamic Partitioning, Memory Allocation Strategies: Best-Fit, First Fit, Worst Fit, Paging and Segmentation, TLB | |
| | 4.2 | Virtual Memory: Demand Paging, Page Replacement Strategies: FIFO, Optimal, LRU, Thrashing | |
| **5** | | **File Management** | **4** |
| | 5.1 | Overview, File Organization and Access, File Directories, File Sharing | |
| **6** | | **I/O management** | **4** |
| | 6.1 | I/O devices, Organization of the I/O Function, Disk Organization, I/O Management and Disk Scheduling: FCFS, SSTF, SCAN, CSCAN, LOOK, C-LOOK. | |

# Text Books

1. **William Stallings,** Operating System: Internals and Design Principles, Prentice Hall, 8thEdition, 2014, ISBN-10: 0133805913 • ISBN-13: 9780133805918.
2. Abraham Silberschatz, Peter Baer **Galvin** and Greg Gagne, Operating System Concepts, John Wiley &Sons, Inc., 9thEdition, 2016, ISBN 978-81-265-5427-0

# Reference Books


1. Achyut Godbole and Atul Kahate, Operating Systems, McGraw Hill Education, 3rdEdition
2. **Andrew Tannenbaum**, Operating System Design and Implementation, Pearson, 3rdEdition.
3. Maurice J. Bach, “Design of UNIX Operating System”, PHI
4. Sumitabha Das, “UNIX: Concepts and Applications”, McGraw Hill, 4thEdition

# UNIT- II (Process and Process Scheduling )

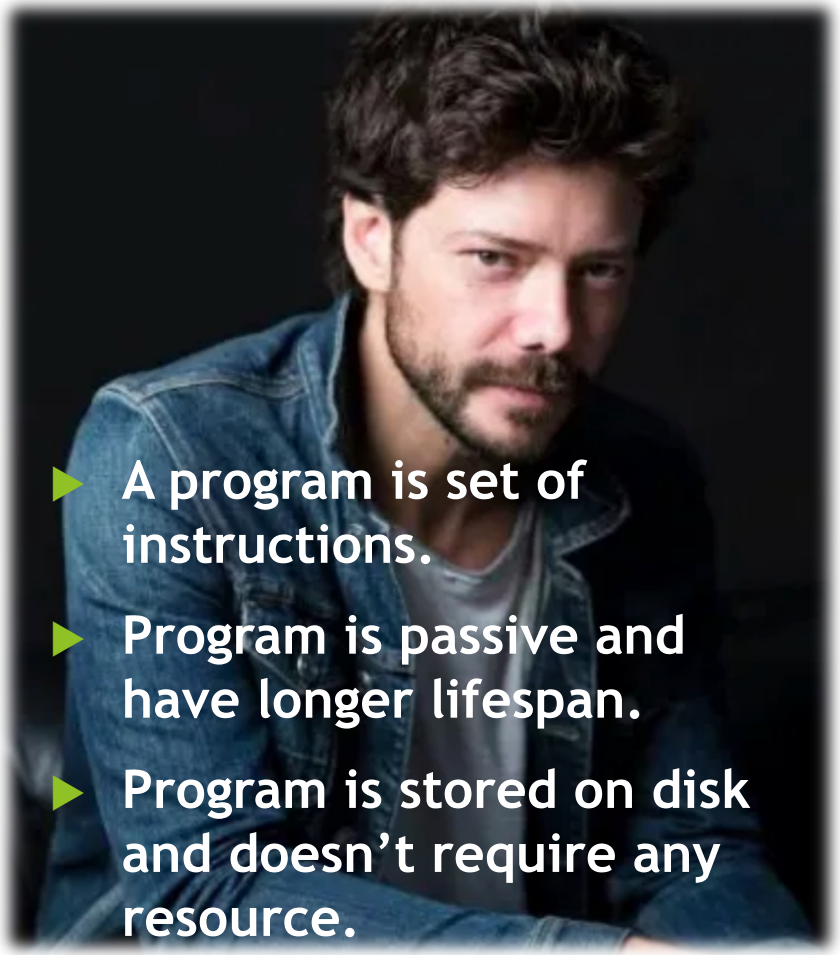
2.1 Concept of a Process, Process States, Process Description, Process Control Block.

2.2 Uniprocessor Scheduling-Types: Preemptive and Non-preemptive scheduling algorithms (FCFS, SJF, SRTN, Priority, RR)

2.3 Threads: Definition and Types, Concept of Multithreading

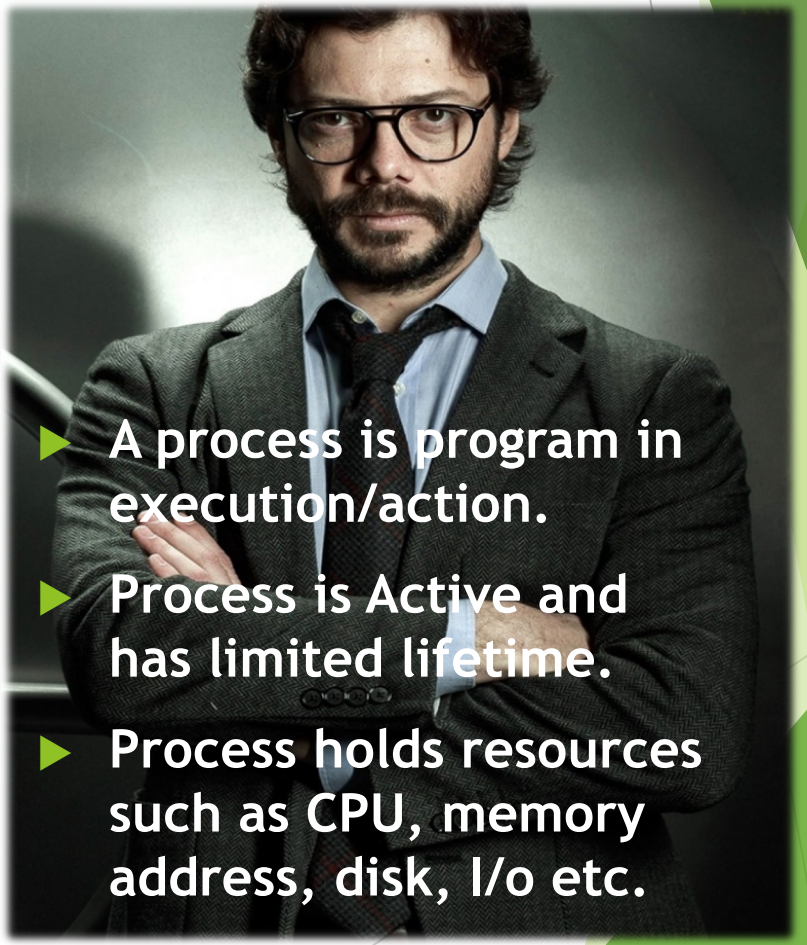
# Program Vs process

**Alvaro Morte**



- A program is set of instructions.
- Program is passive and have longer lifespan.
- Program is stored on disk and doesn't require any resource.

**Professor**



- A process is program in execution/action.
- Process is Active and has limited lifetime.
- Process holds resources such as CPU, memory address, disk, I/o etc.

# Program vs Process

**Chai Tea Recipe:**

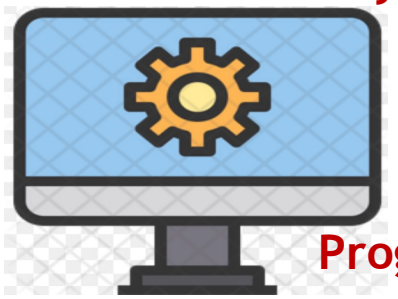
1. *Add water in pan.*
2. *Add sugar, tea leaves, spices.*
3. *Bring to boil and simmer.*
4. *Add milk.*
5. *Bring to boil and simmer.*
6. *Strain tea in teapot.*

# Process

- **Algorithm:**
  A part of computer program that performs a well-defined task.
- **Software:**
  A collection of computer programs, libraries, and related data are referred to as a software.

- A program becomes a process when an executable file is loaded into memory.
- Although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences.

When in memory



Program

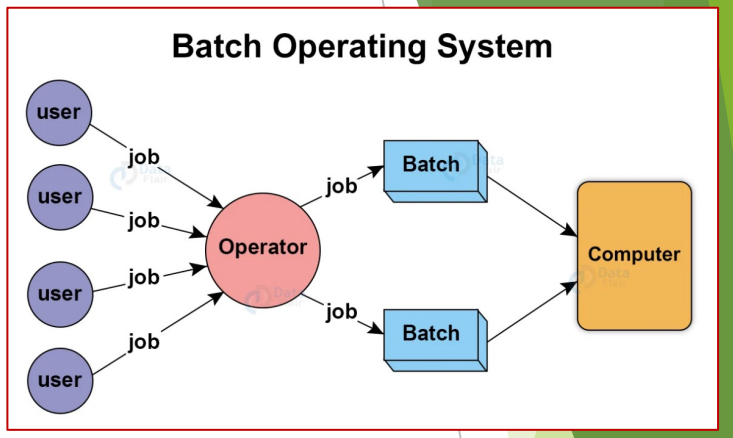
CPU and resources allocated

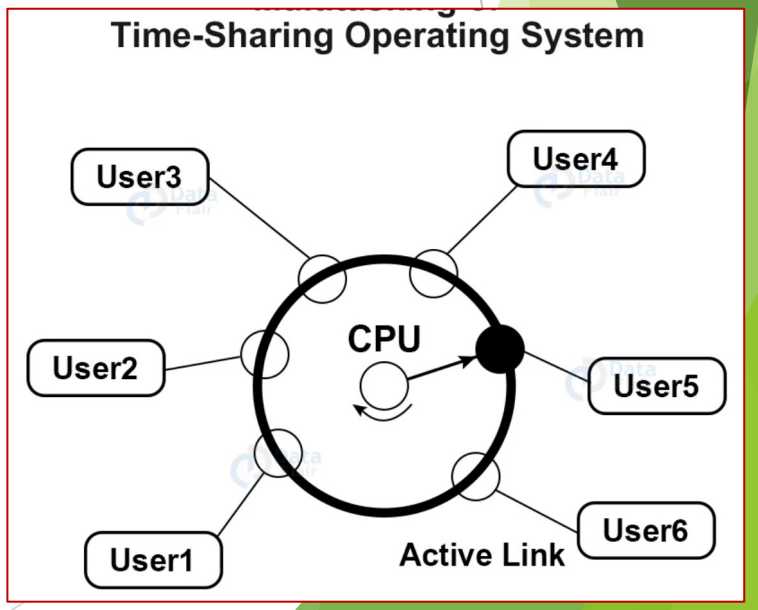
Process

# Process

- An operating system executes a variety of programs:
  - Batch system – jobs ( Program +I/O+ Control Inst.)
  - Time-shared systems – user programs or tasks

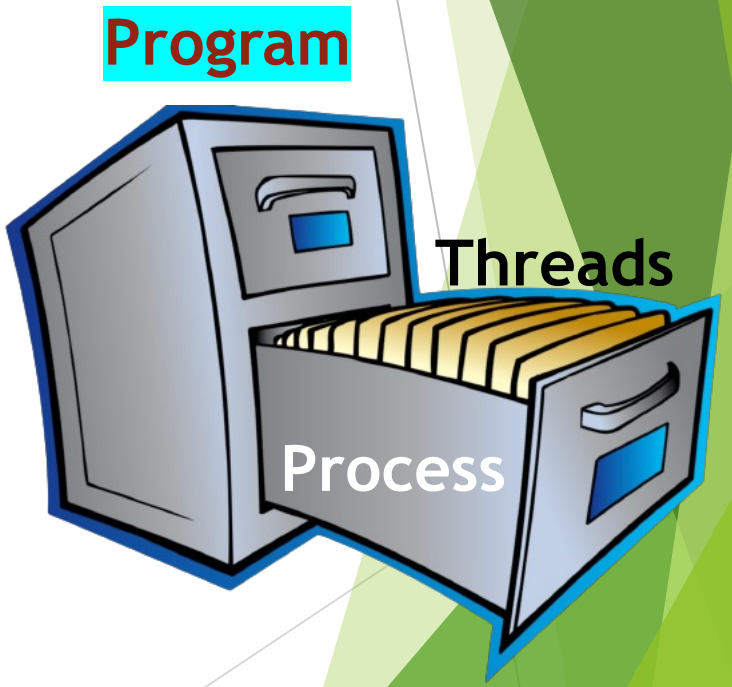
- Textbook uses the terms job and process almost interchangeably



# PROCESS

- In **computing**, a **process** is the instance of a **computer** program that is being executed by one or many threads.
- It contains the program code and its activity.
- Depending on the **operating system** (**OS**), a **process** may be made up of multiple threads of execution that execute instructions concurrently.



# Program to process

- A program becomes a process when an executable file is loaded into memory and becomes active.

Two common techniques for loading executable files are

- double-clicking an icon representing the executable file
- entering the name of the executable file on the command line

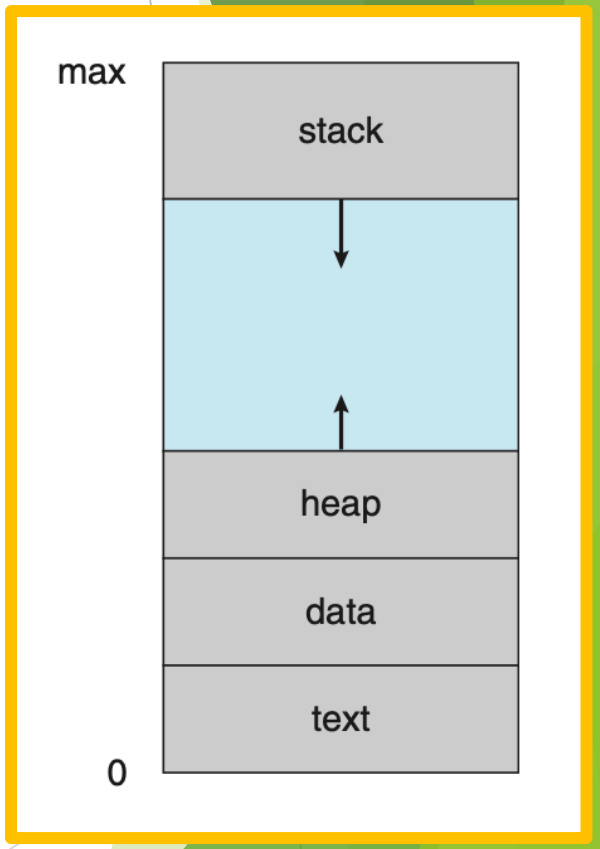
# A process includes:

- program counter
- stack
- data section
- heap

# Memory Layout of Process

The status of the current activity of a process is represented by the value of the **program counter** and the contents of the processor's registers.

- **Text Section. : the executable code**
  It includes the current activity which is represented by value of Program counter(PC) and the contents of the processor's register.
- **Data Section : global variables**
- **Heap Section :**
  memory that is dynamically allocated during program run time
- **Stack Section :**
  temporary data storage when invoking functions
  (such as function parameters, return addresses, and local variables)

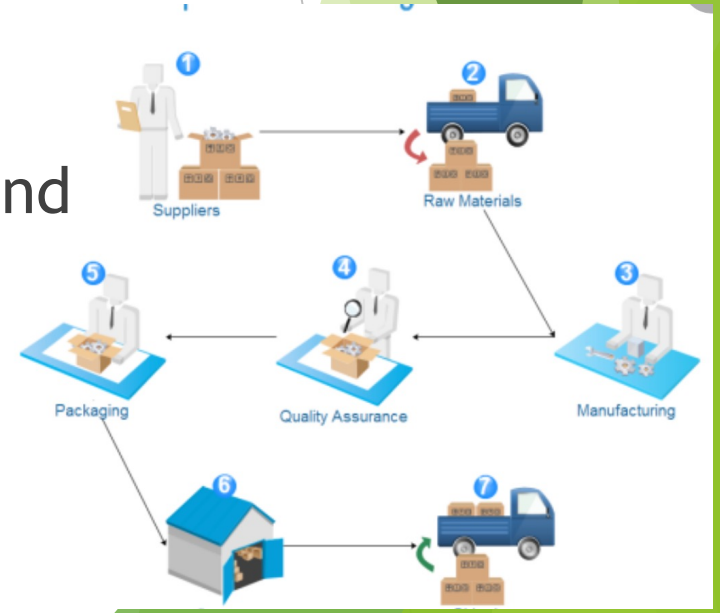



The sizes of the text and data sections are fixed, as their sizes do not change during program run time.

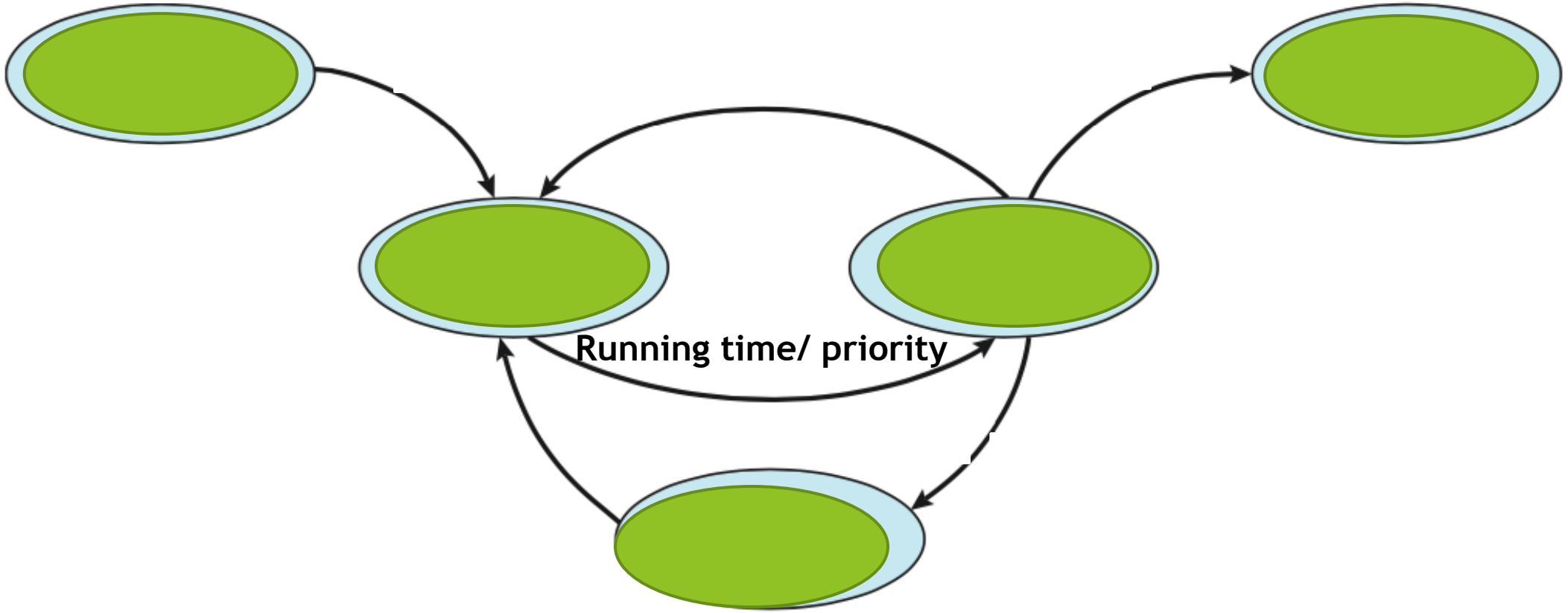
However, the stack and heap sections can shrink and grow dynamically during program execution.

# States of Process

- When a process executes, it passes through different states.
- State of a process defines the current activity of that process.
- These stages may differ in different operating systems, and the names of these states are also not standardized.

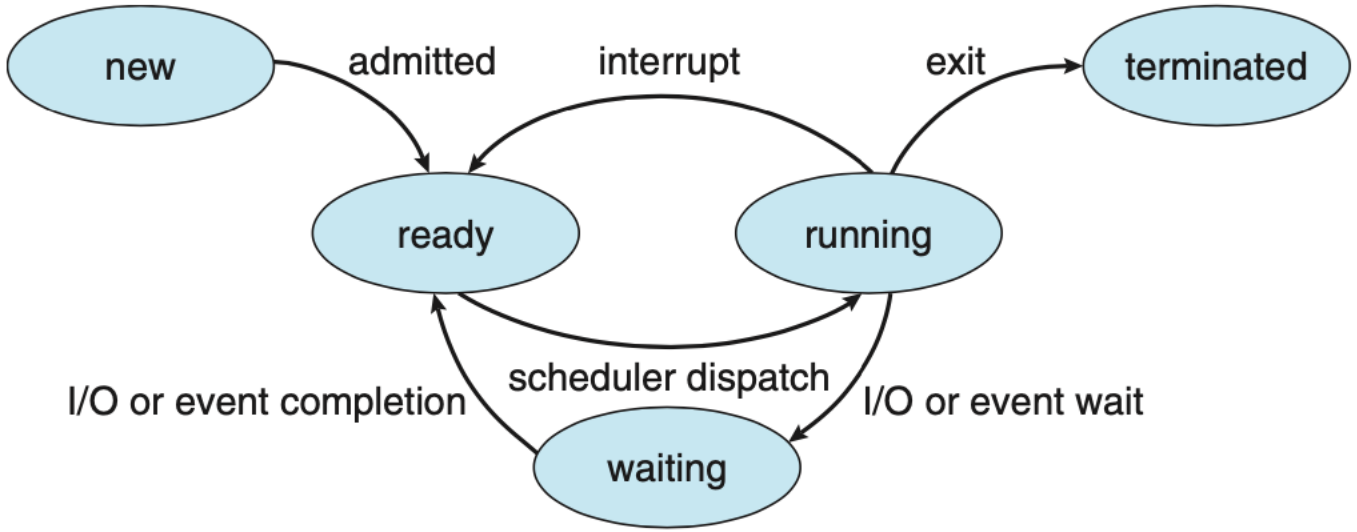


# States of Process



# States of Process

- **New -**
  The process is in the stage of **being created.**
- **Ready** –
  The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.
- **Running** –
  The CPU is working on this process's instructions.
- **Waiting** –
  The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur.
- **Terminated** –
  The process has completed.



# Process Control Block (PCB)

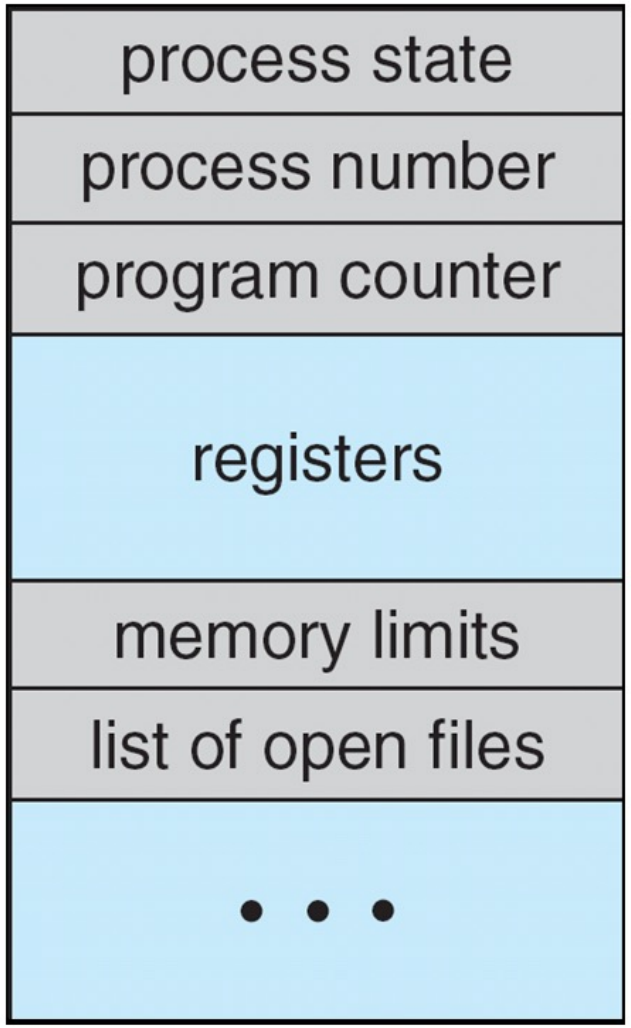
- As the operating system supports multi-programming, it needs to keep track of all the processes.
- Each process is represented in the operating system by a process control block (PCB)—also called a task control block.
- It contains many pieces of information associated with a specific process.
- All these information is required and must be saved when the process is switched from one state to another.

# Process Control Block (PCB)

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

# Process Control Block (PCB)




**Process state:** The state may be new, ready, running & so on

**Program counter:** It indicates the address of the next instruction to be executed for this program.

**CPU registers:** These vary in number and type based on architecture. They include accumulators, stack pointers, general purpose registers etc.

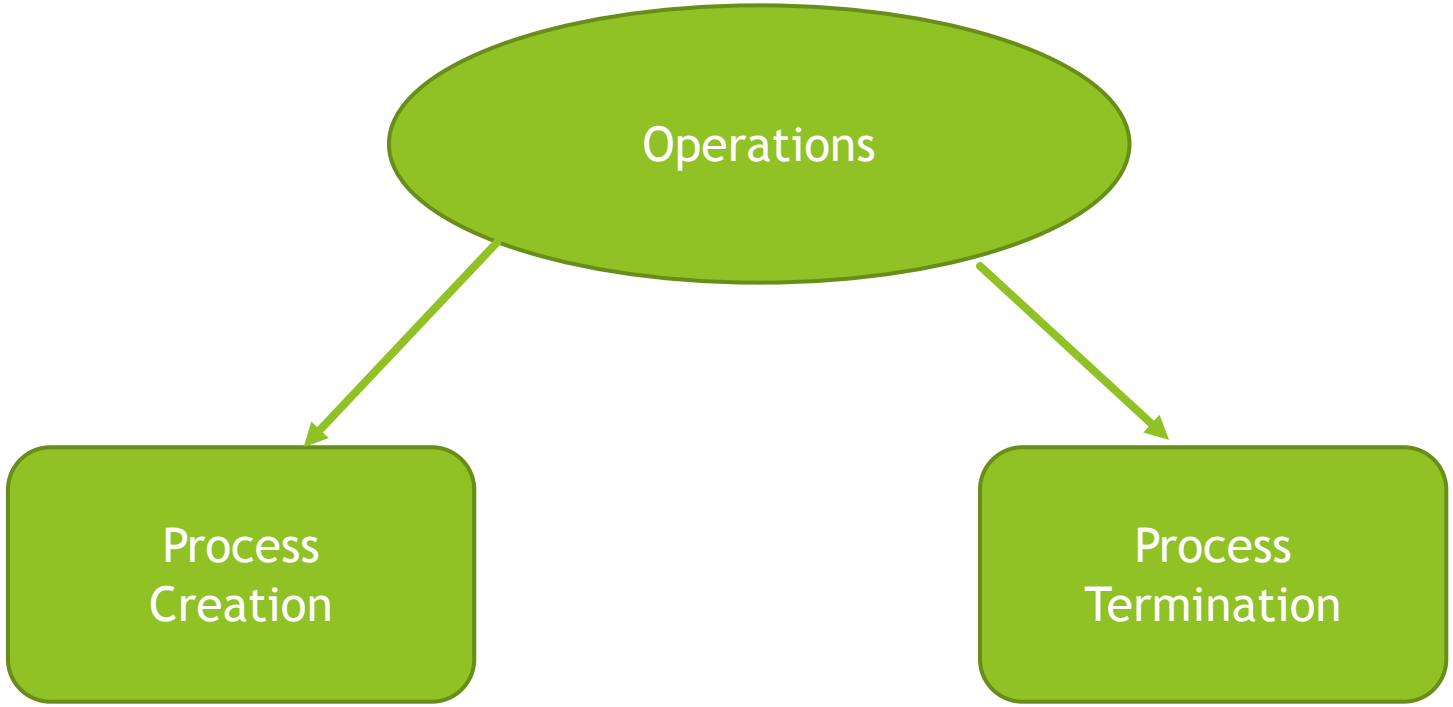
**CPU scheduling information:** This includes process priority, pointers to scheduling queues and any scheduling parameters.

**Memory-management information:** This includes the value of base and limit registers (protection) and page tables, segment tables depending on memory.

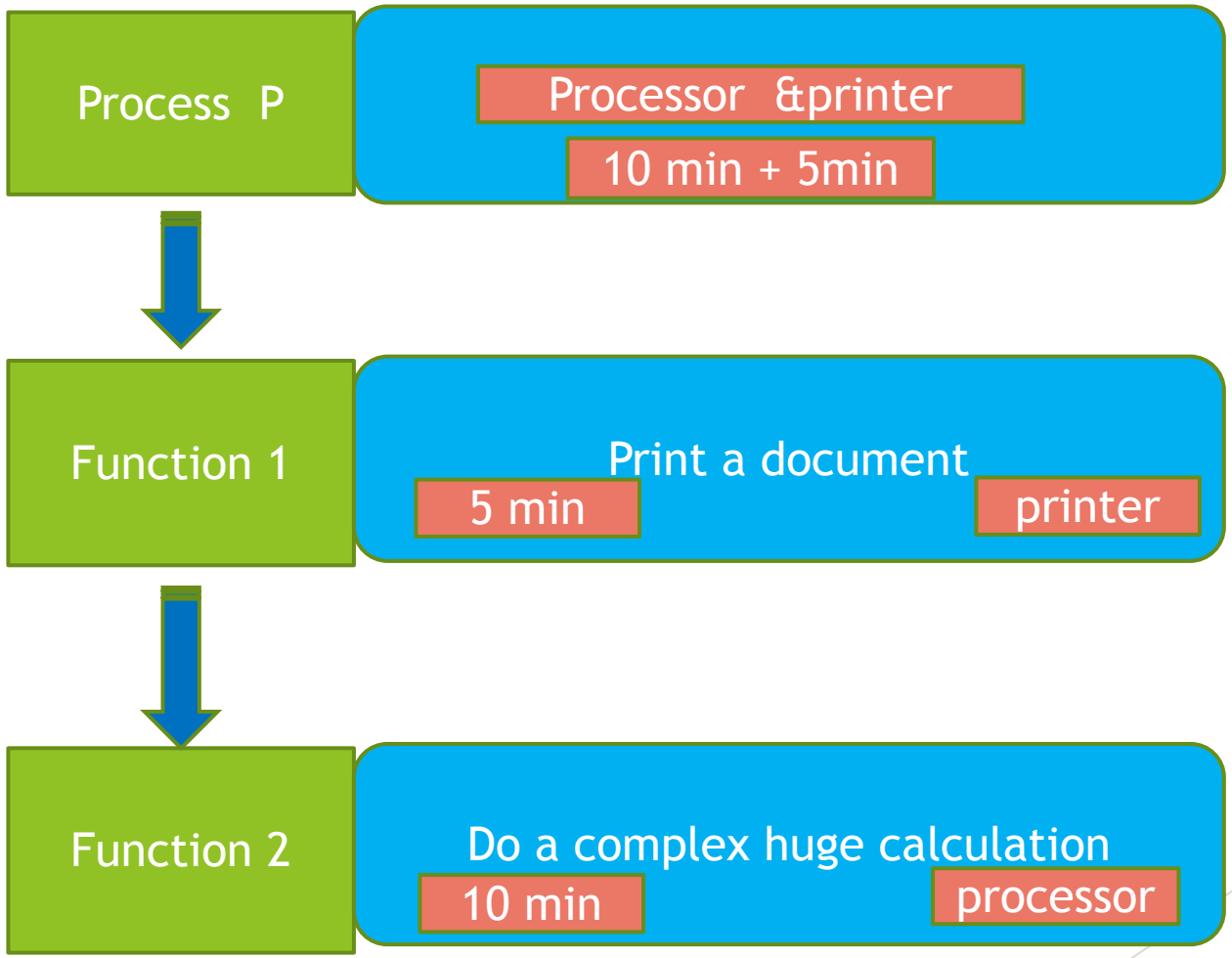
**Accounting information:** It includes amount of CPU and real time used, account numbers, process numbers etc

**I/O status information:** It includes list of I/O devices allocated to this process, a list of open files etc

# Operations On Process

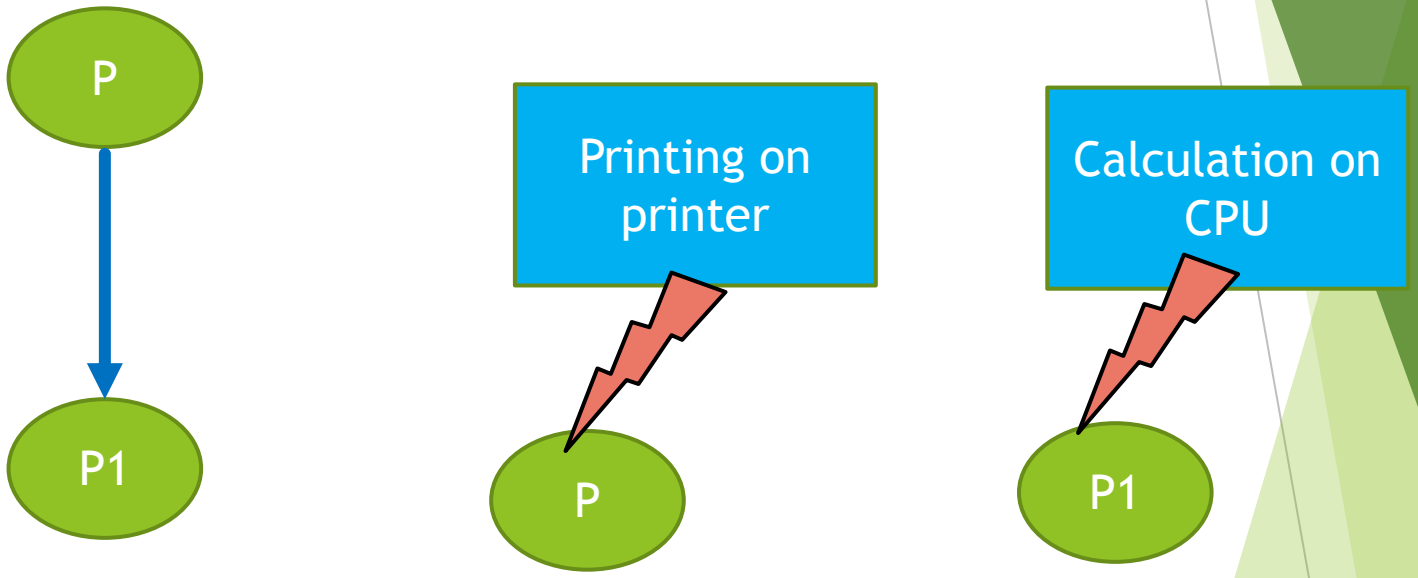


# Single process execution




Execution with Single process

# Process creation




Total time taken to complete **both the tasks** = 10 min
Total time taken will be the time taken by slowest process.

# Process creation

pid
process identifier

Process Creation

- Parent – creating process
- Child- new process

How to create child process

- fork() – unix
- createprocess()- Windows

How a child process gets it resources.

- OS
- Subset of parents resources

# Process creation

**Parent**

For execution parent process can either

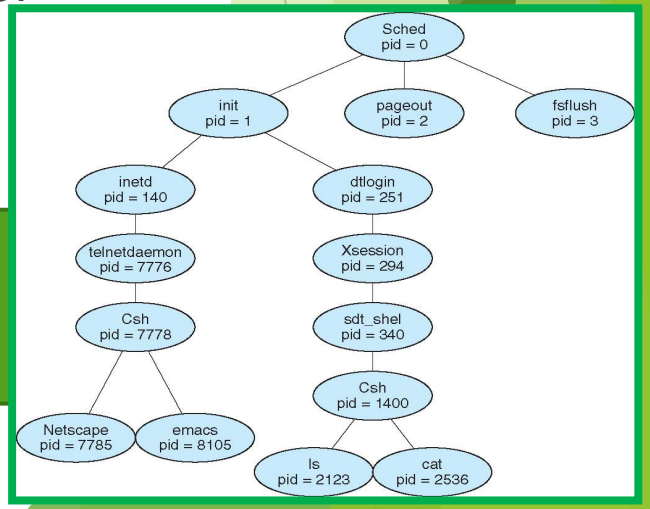
- Continues to execute concurrently with its children. Or
- Wait until some or all the children terminates.

**Child**

For address-space child process

- Can be a duplicate of the parent process.
  (it has the same program and data as the parent). Or
- Has a new program loaded into it.

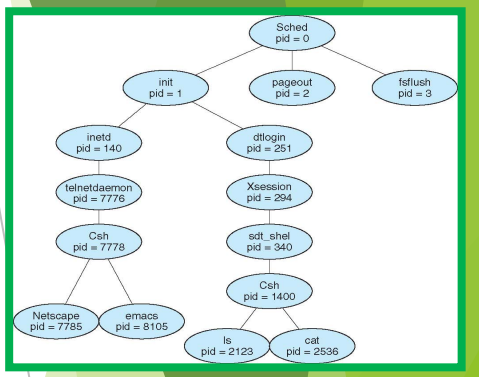
Child process can further create process forming a tree of processes.



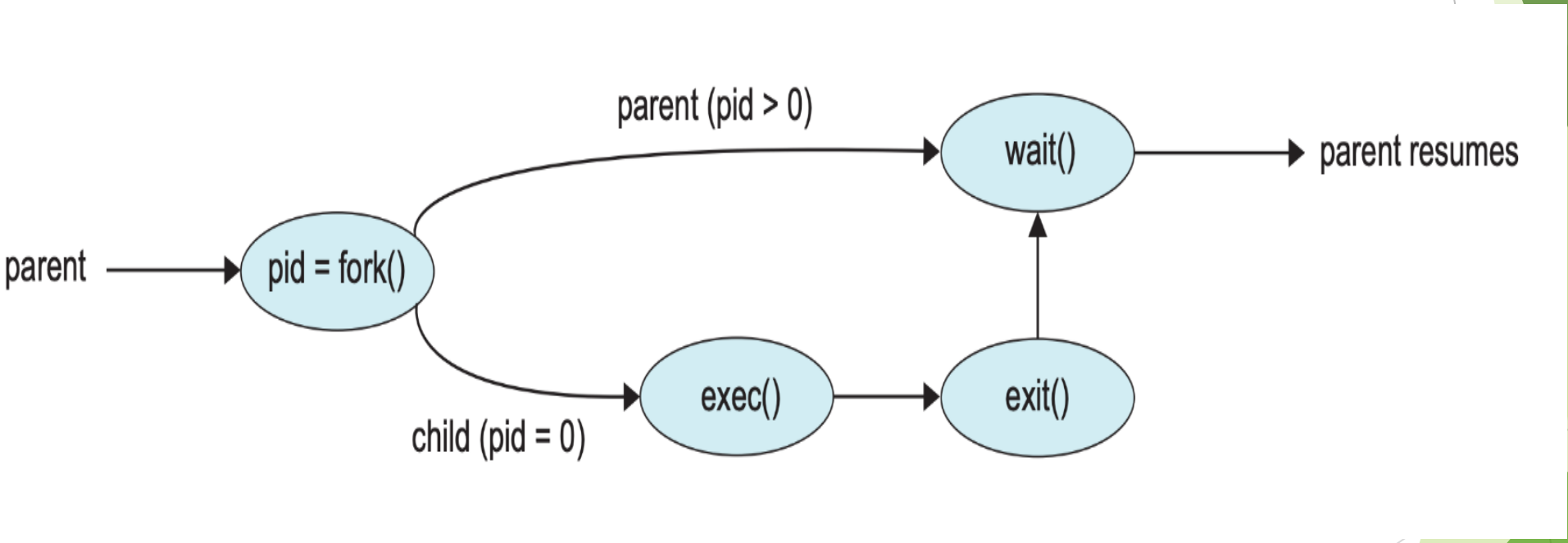
# Process Termination

- Process executes last statement and asks the OS to delete it with **exit()** system call.
  - Output data and status from child to parent process(via **wait() system call**)
  - Process' resources are deallocated by operating system

- **Only Parent** may terminate execution of children processes **(abort)** in following conditions:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - **cascading termination**



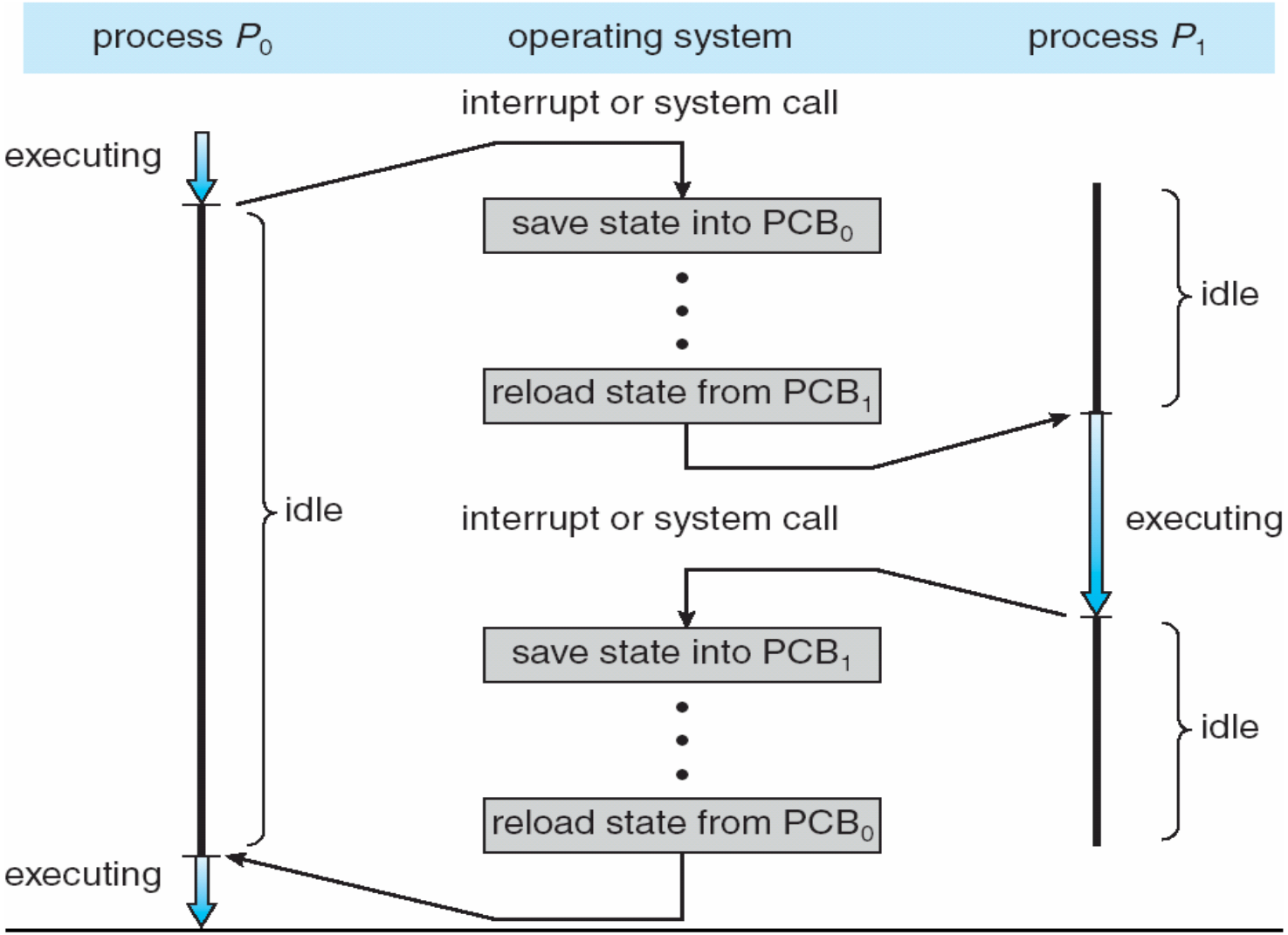

# Process Creation



# Context Switch

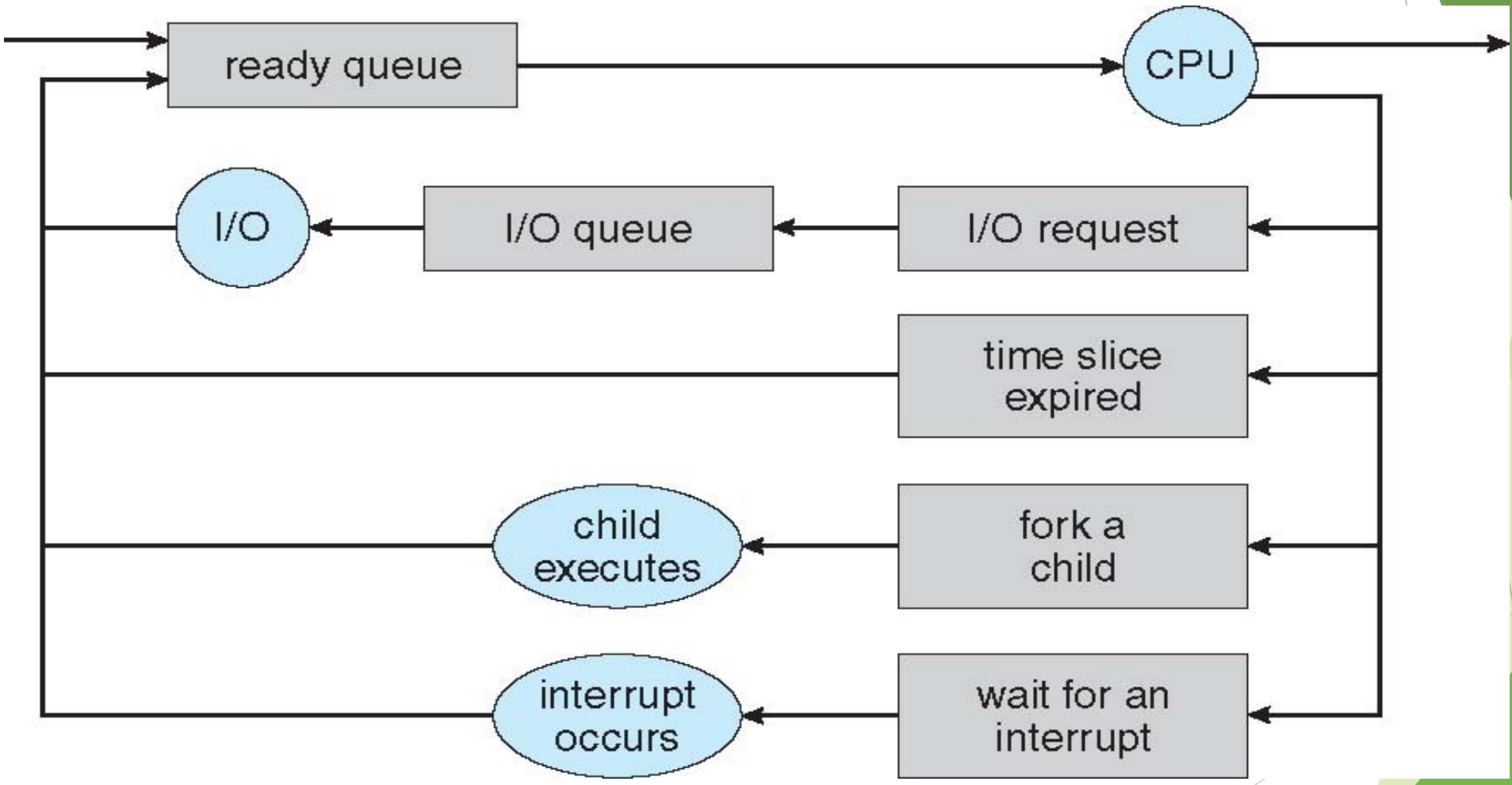
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**.
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

# Context Switching

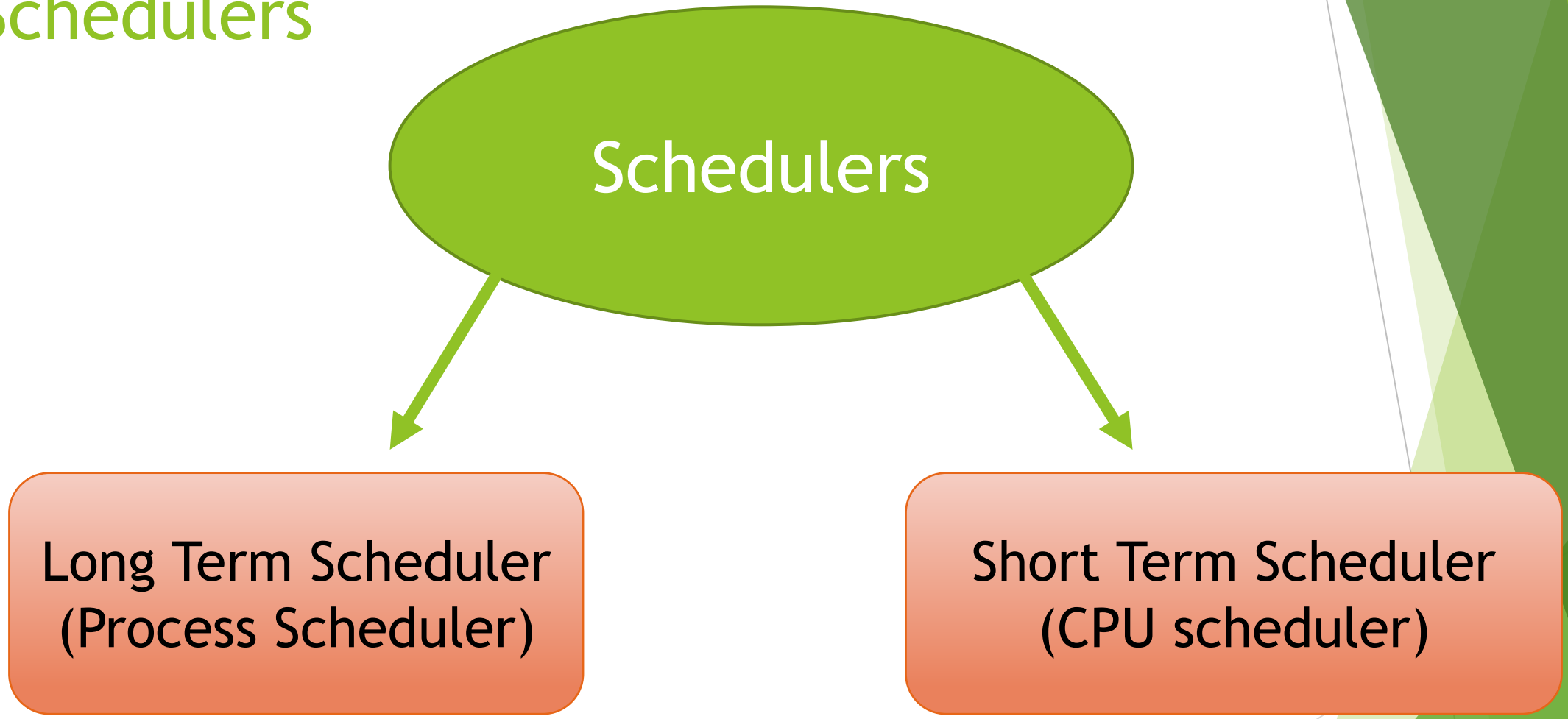


# SCHEDULING

# Representation of Process Scheduling



# Schedulers

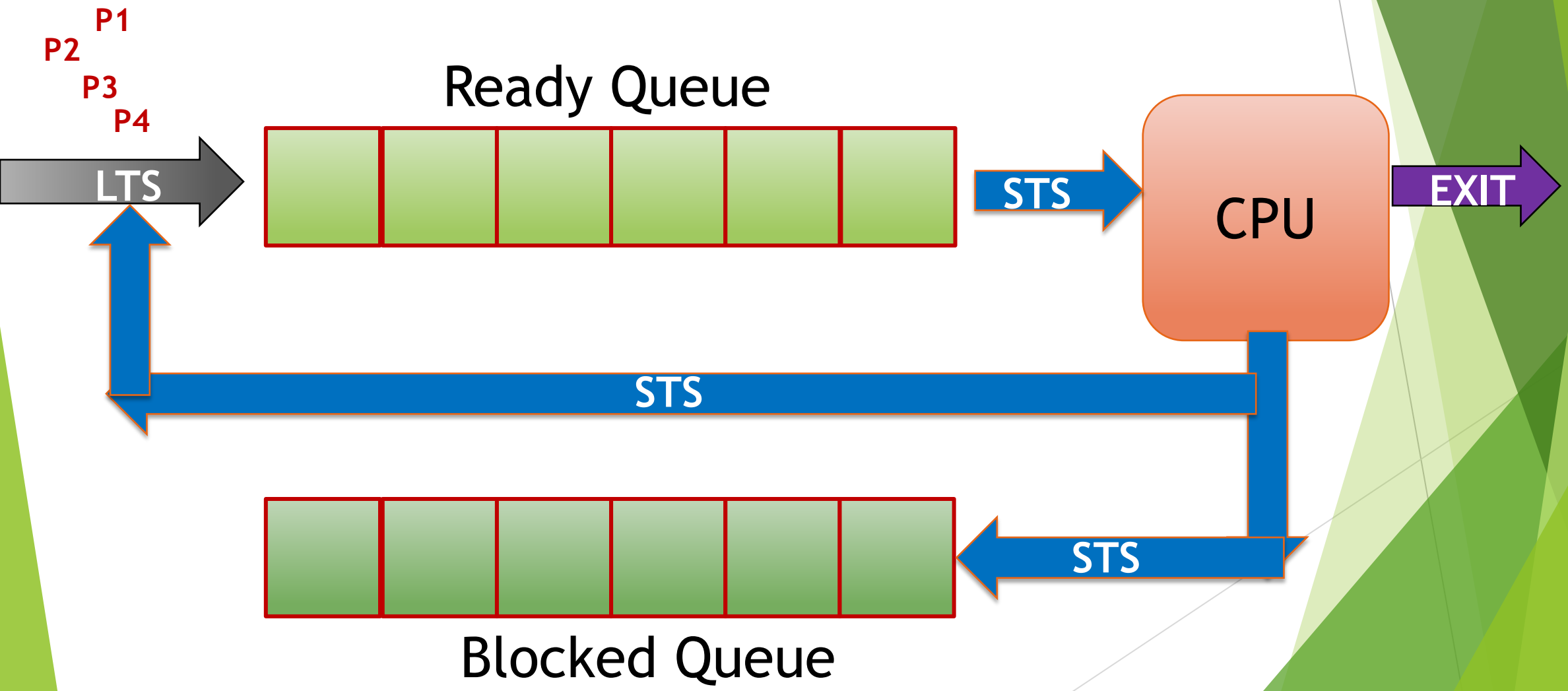

Schedulers
Long Term Scheduler
(Process Scheduler)
Short Term Scheduler
(CPU scheduler)

# Schedulers

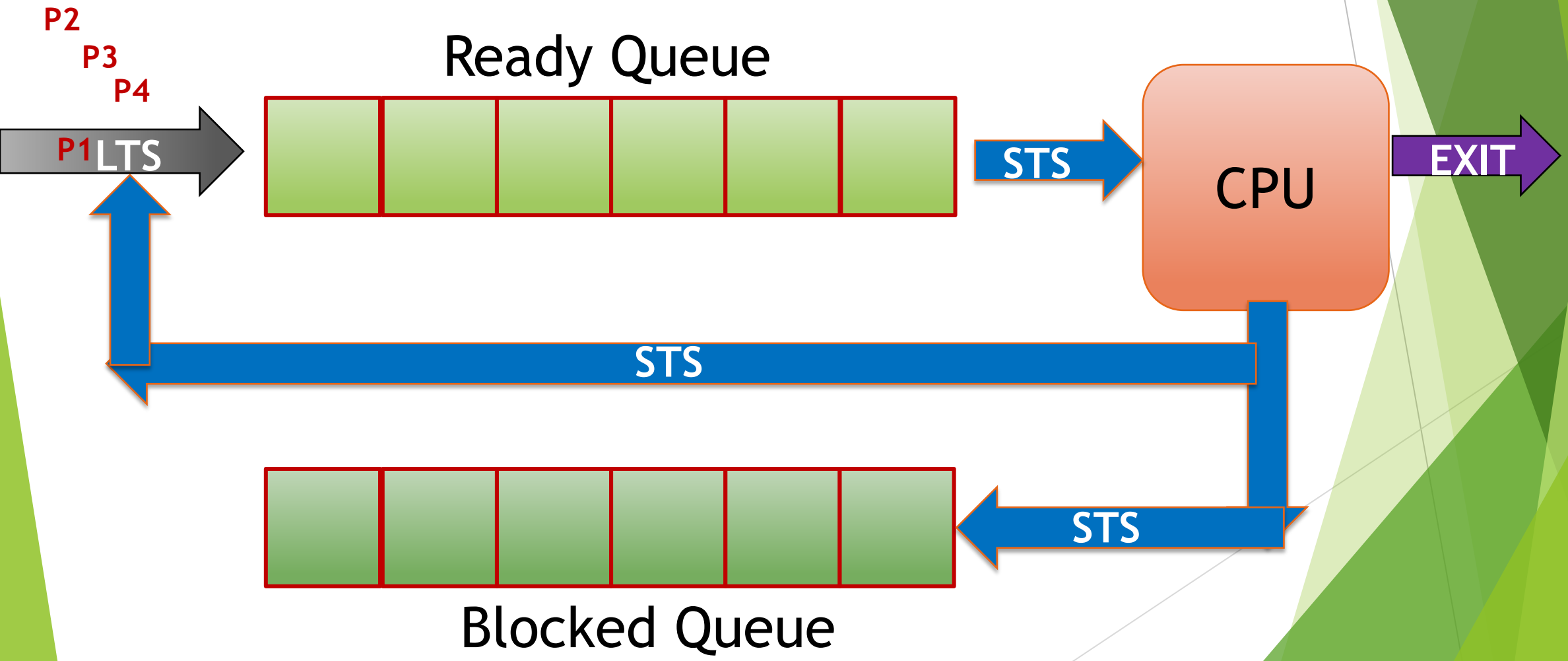
- **Long-term scheduler** (or job/process scheduler) –
  - selects which processes should be brought into the "ready queue"

- **Short-term scheduler** (or CPU scheduler) –
  - selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system

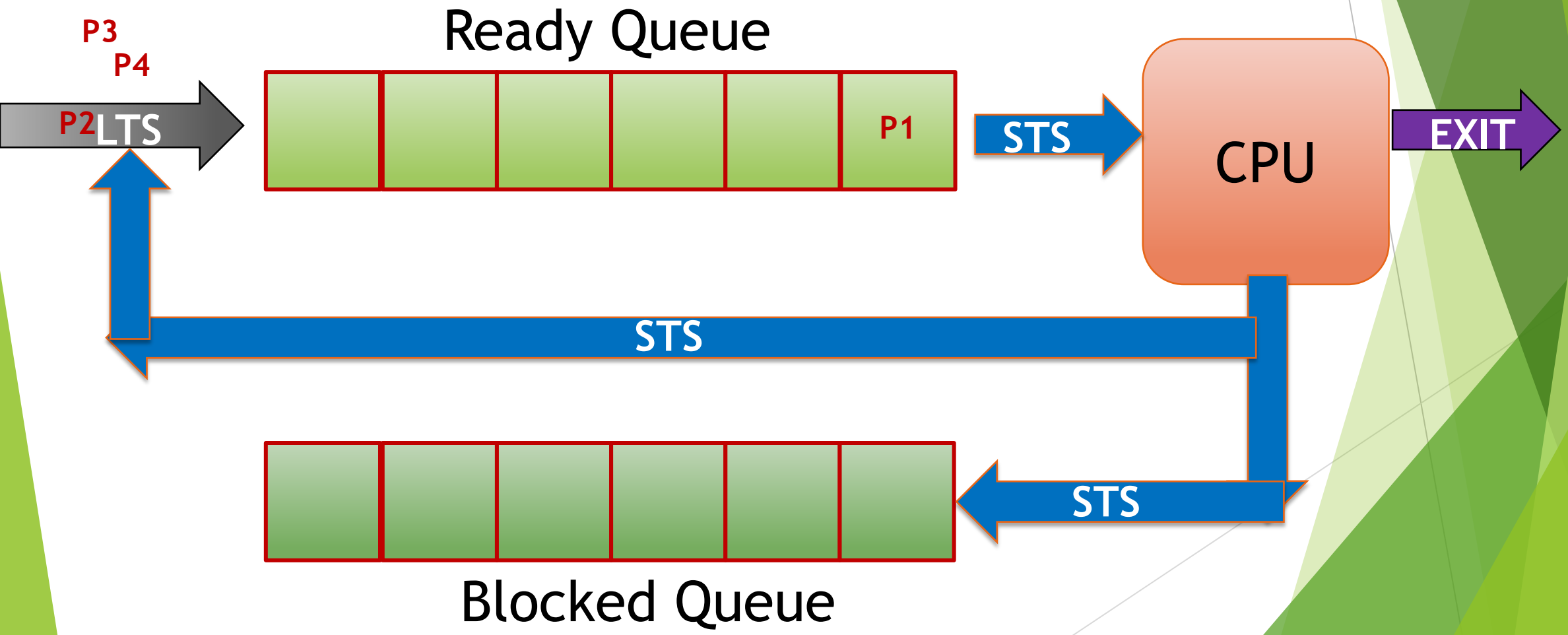
# Scheduling



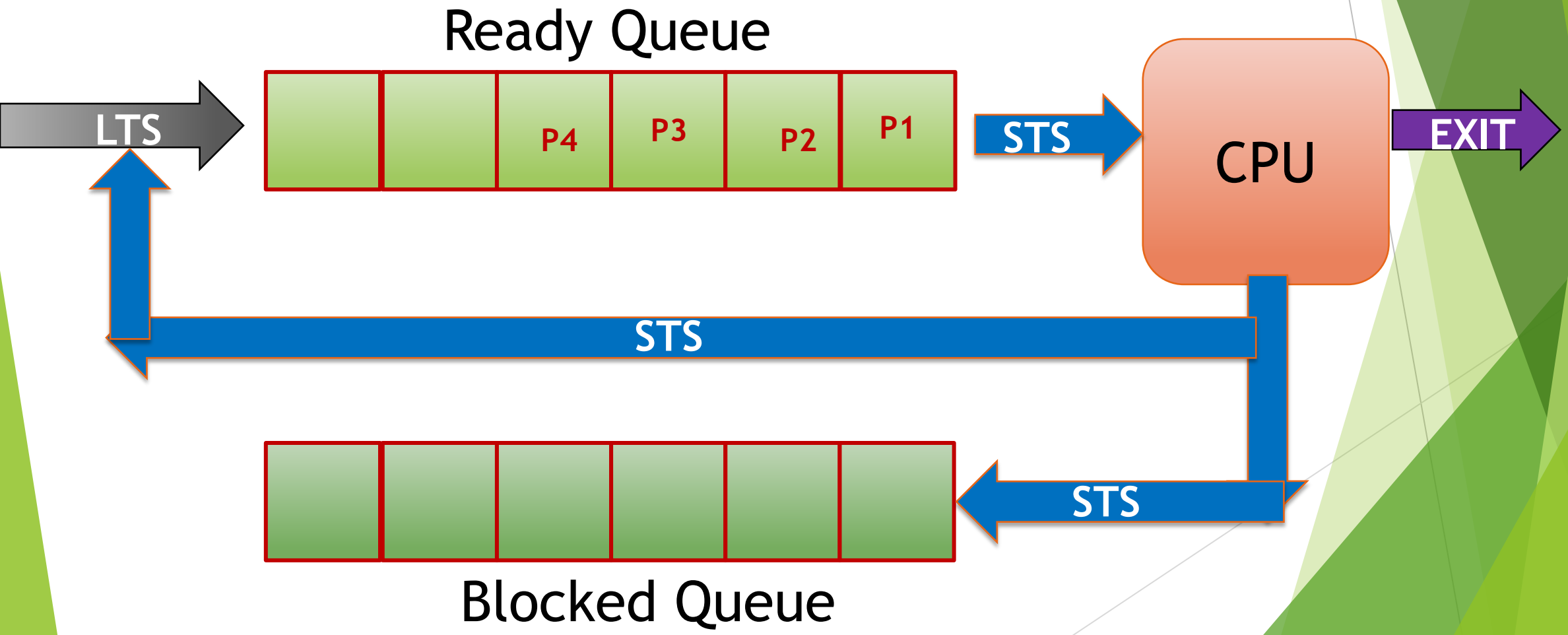
# Scheduling



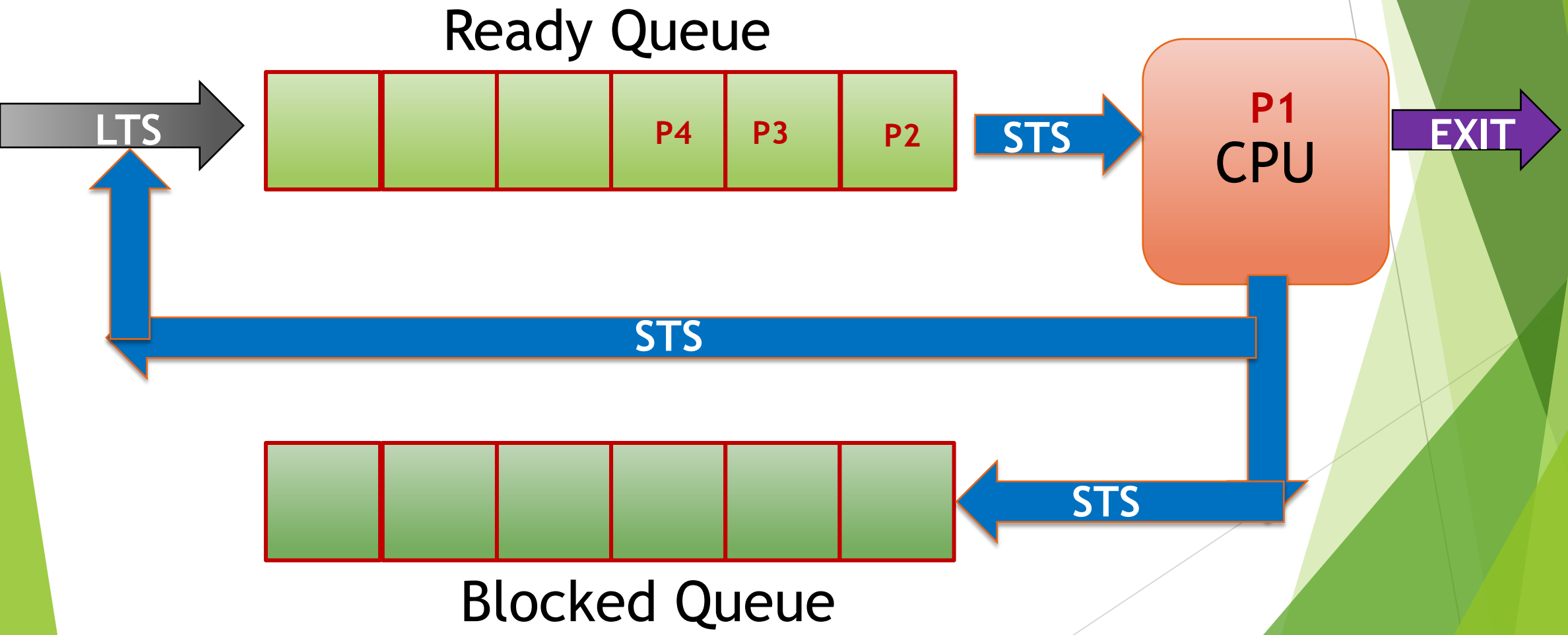
# Scheduling



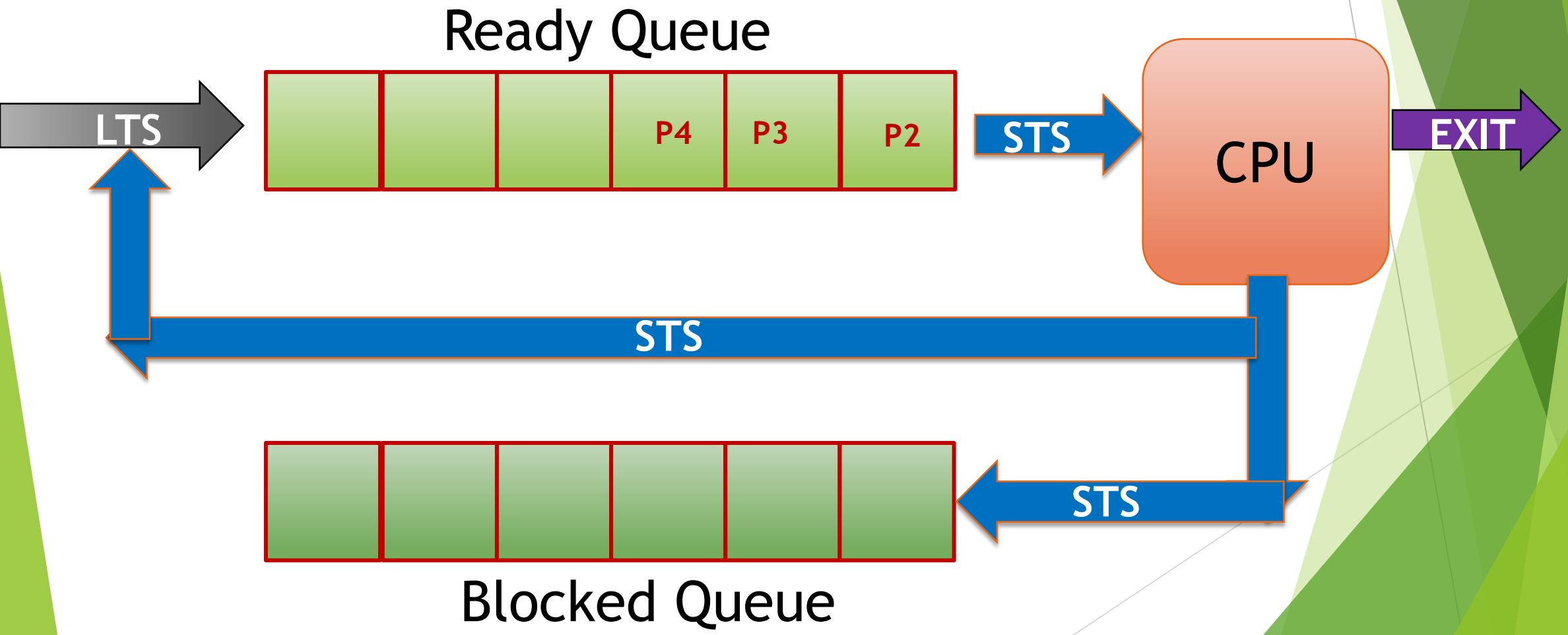
# Scheduling



# Scheduling



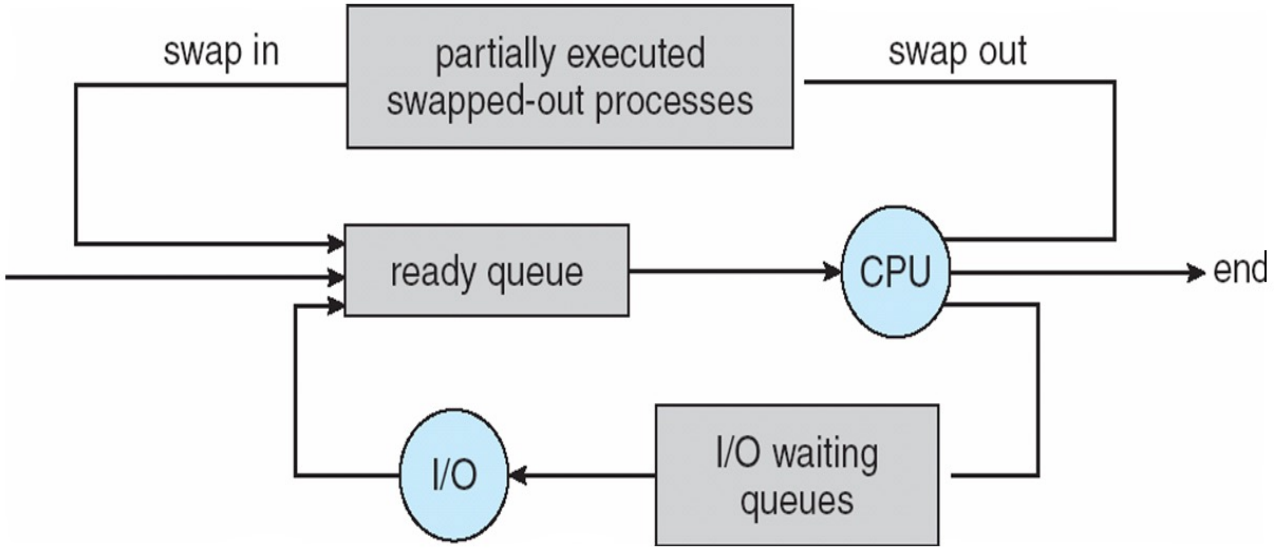
# Scheduling



# Schedulers

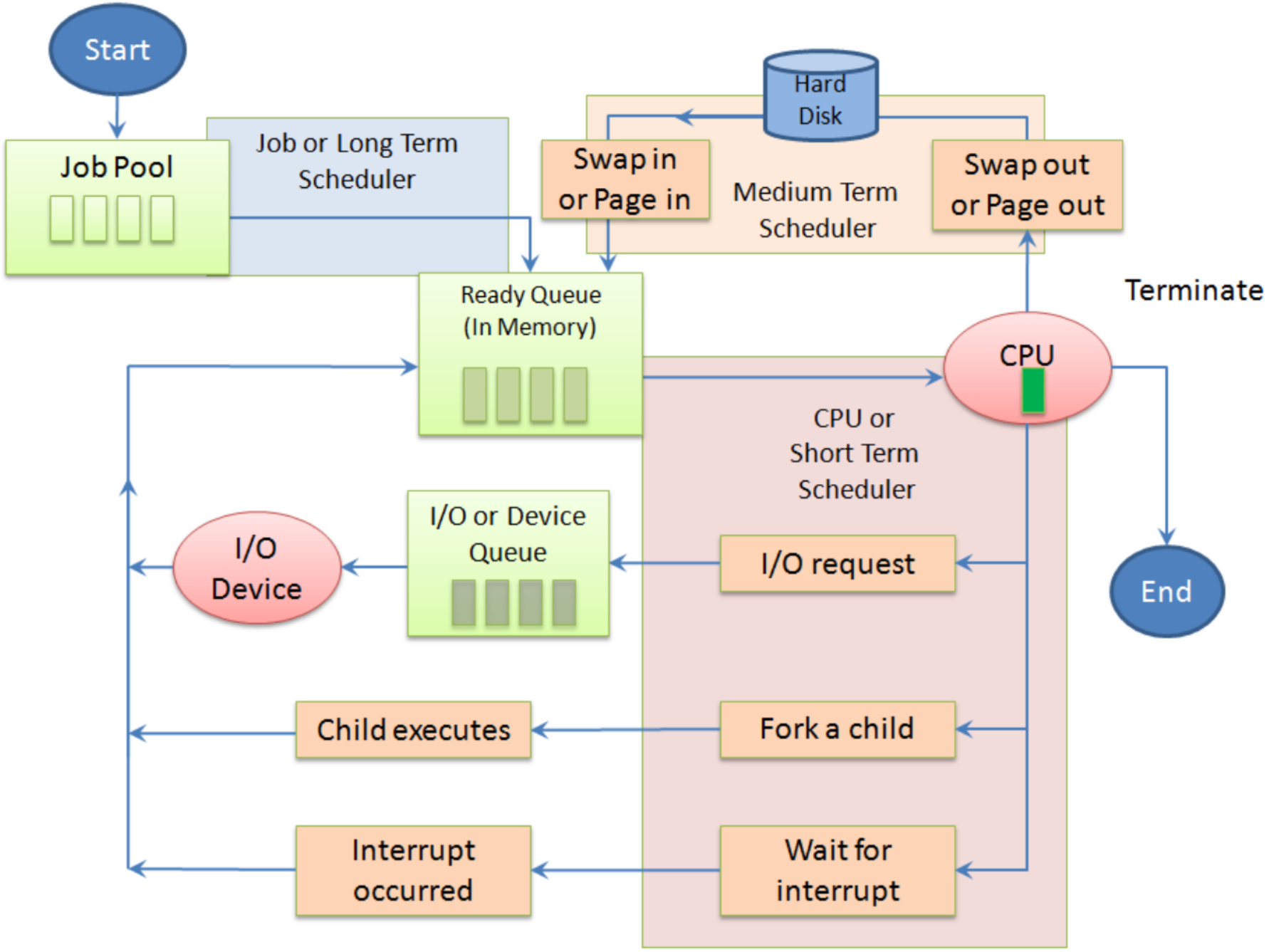
- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is not invoked frequently (seconds, minutes) $\Rightarrow$ (may be slow)

# Medium Term Scheduling




- Medium-term is also called swapping scheduler.
- It helps you to send process back to memory.
- Reduce the level of multiprogramming.

# Schedulers in OS



# CPU Scheduling

When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process and this pattern continues.

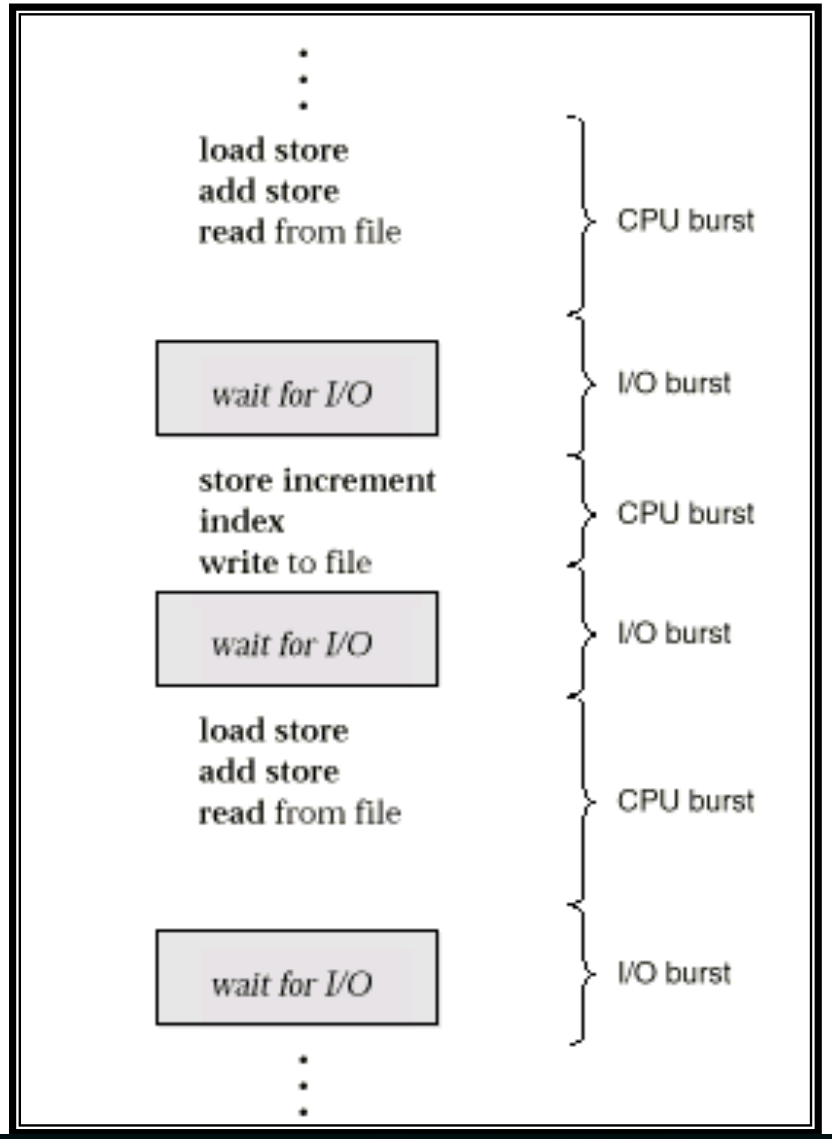

# CPU Scheduling

- CPU scheduling is the basis of multiprogrammed OS.
- Switching CPU among processes, the OS can make the computer more productive.
- In single processor system only one process can run at a time.

**Objective of multi programming :**

To have some process running at all times , to maximize CPU utilization.

# Alternating Sequence of CPU And I/O Bursts




Process execution consists of a cycle of **CPU execution** and **I/O wait**

CPU burst is when process is being executed in the CPU

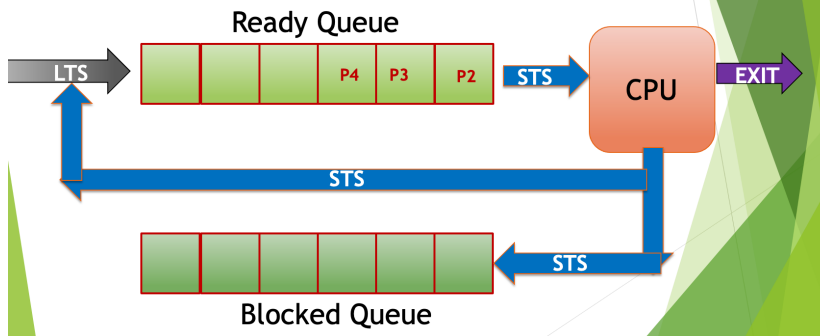
I/O burst is when CPU is waiting for I/O for further execution

Eventually, the final **CPU burst** ends with a system request to terminate execution.

# Process

- Processes can be described as either:
  - **I/O-bound process** –
    spends more time doing I/O than computations,
    many short CPU bursts
  - **CPU-bound process** –
    spends more time doing computations;
    few very long CPU bursts

# CPU scheduling




- **CPU Scheduler(STS) :**
  Selects a process from the processes in the memory that are ready to execute and allocate CPU to it.
- **Dispatcher:**
  Dispatcher is the module that gives control of CPU to the process selected by short-term scheduler.

- **MAIN OBJECTIVE**
  - → Maximum CPU utilization obtained with multiprogramming
  - → CPU–I/O Burst Cycle – CPU burst distribution

# CPU Scheduler

- CPU scheduling decisions may take place when a process:
    1. Switches from running to waiting state.
    2. Switches from running to ready state.
    3. Switches from waiting to ready.
    4. Terminates.

- **Preemptive:** allows a process to be interrupted in the midst of its CPU execution, taking the CPU away to another process
- **Non- Preemptive:** the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state.
- Scheduling under 1 and 4 is *non-preemptive*.

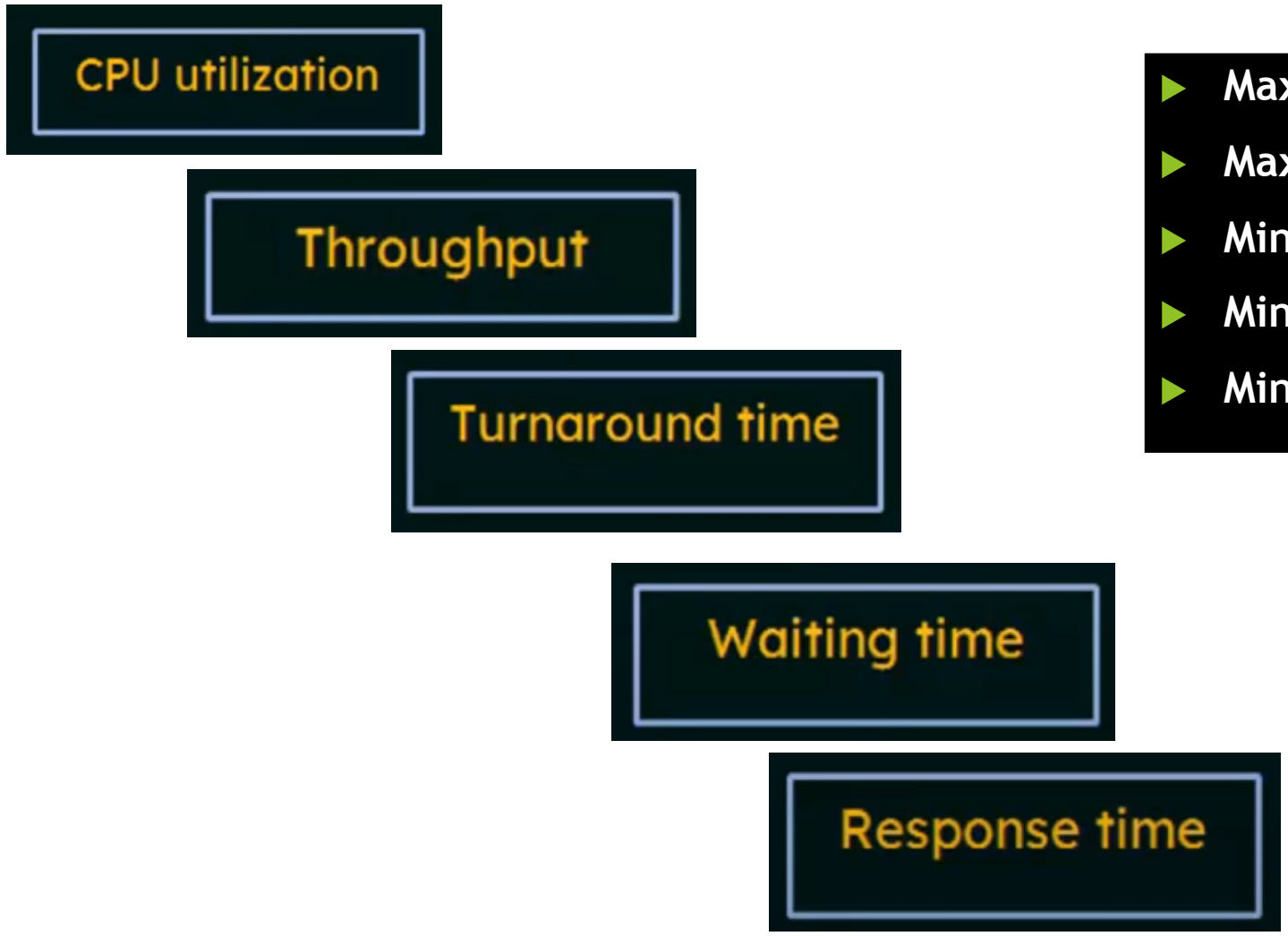
# CPU scheduling

- **Arrival time :**
  - is the time when a process enters into the ready state and is ready for its execution.
- **Exit time**
  - is the time when a process completes its execution and exit from the system.
- **Response time**
  - is the time spent when the process is in the ready state and gets the CPU for the first time.
  - Response time =
    Time at which the process gets the CPU for the first time - Arrival time

# CPU scheduling

- **Waiting time**
  - is the total time spent by the process in the ready state waiting for CPU.
  - **Waiting time = Turnaround time - Burst time**
- **Burst time**
  - total **time** taken by the process for its execution on the CPU.
- **Turnaround time**
  - is the total amount of time spent by the process from coming in the ready state for the first time to its completion.
  - **Turnaround time = Burst time + Waiting time or**
    **Turnaround time = Exit time - Arrival time**
- **Throughput**
  - is a way to find the efficiency of a CPU.
  - It can be defined as the number of processes executed by the CPU in a given amount of time.

# Scheduling Criteria



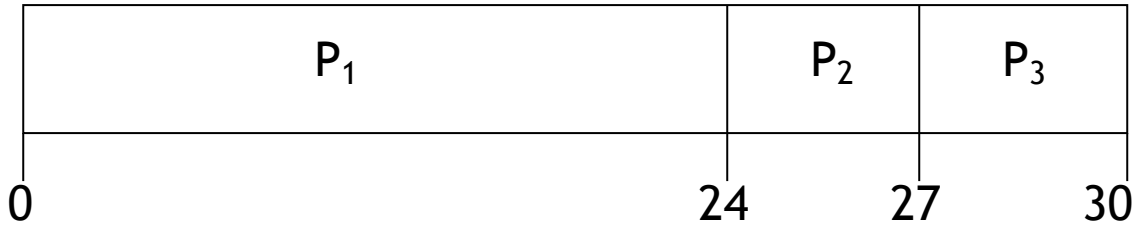

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

# CPU Scheduling Algorithms

# First-Come, First-Served (FCFS) Scheduling Non-preemptive

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

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




- Waiting time for $P_1$ = 0;

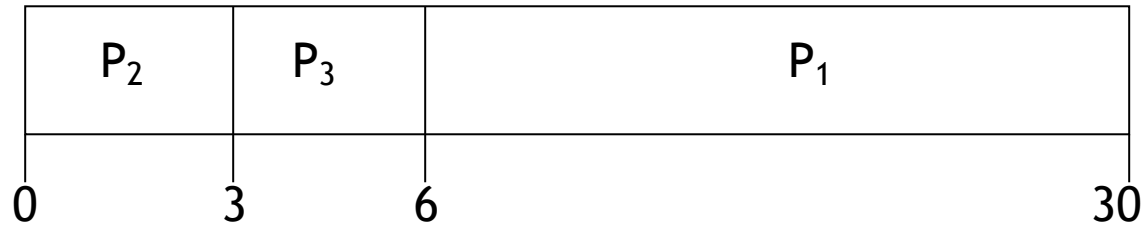
  $P_2$ = 24; $P_3$ = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

# FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1 .$$

- The Gantt chart for the schedule is:




- Waiting time for $P_1 = 6;$
- $P_2 = 0; P_3 = 3$
- Average waiting time: **(6 + 0 + 3)/3 = 3**
- Much better than previous case. With shortest process first

# FCFS

| Process | Duration | Oder | Arrival Time |
|---|---|---|---|
| P1 | 24 | 1 | 0 |
| P2 | 3 | 2 | 0 |
| P3 | 4 | 3 | 0 |

GANTT CHART


P1 waiting time : 0
P2 waiting time : 24
P3 waiting time : 27

The Average waiting time :

(0+24+27)/3 = 17

# FCFS

| Process | Arrival Time | Burst Time(BT) | Completion time | Turn Around time (CT-AT) | Waiting Time (TAT-BT) | Response Time |
|---|---|---|---|---|---|---|
| P1 | 2 | 2 | | | | |
| P2 | 0 | 1 | | | | |
| P3 | 2 | 3 | | | | |
| P4 | 3 | 5 | | | | |
| P5 | 4 | 4 | | | | |

# FCFS

| Process | Arrival Time | Burst Time(BT) | Completion time | Turn Around time (CT-AT) | Waiting Time (TAT-BT) | Response Time |
|---|---|---|---|---|---|---|
| P1 | 2 | 2 | 4 | 2 | 0 | 0 |
| P2 | 0 | 1 | 1 | 1 | 0 | 0 |
| P3 | 2 | 3 | 7 | 5 | 2 | 2 |
| P4 | 3 | 5 | 12 | 9 | 4 | 4 |
| P5 | 4 | 4 | 16 | 12 | 8 | 8 |


Response time = Waiting time
(non preemptive)

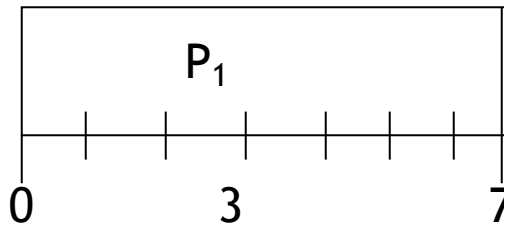
# Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its CPU burst.
- Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - **Non-preemptive** - once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt.
    This scheme is know as the **Shortest-Remaining-Time-First (SRTF).**
- SJF is optimal – gives minimum average waiting time for a given set of processes.

# Example of 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 |

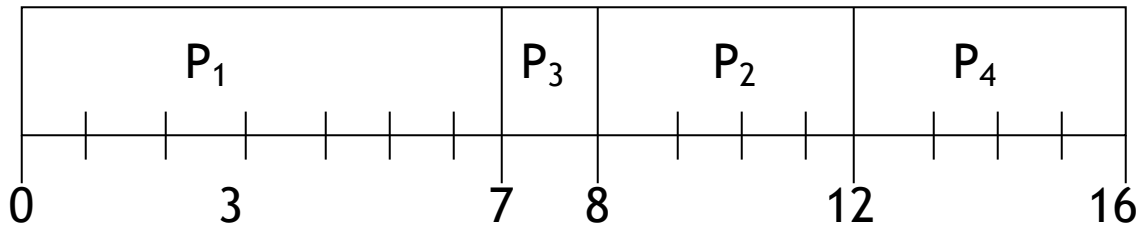
- SJF (non-preemptive)



# Example of 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

# Example of Preemptive SJF → SRTF

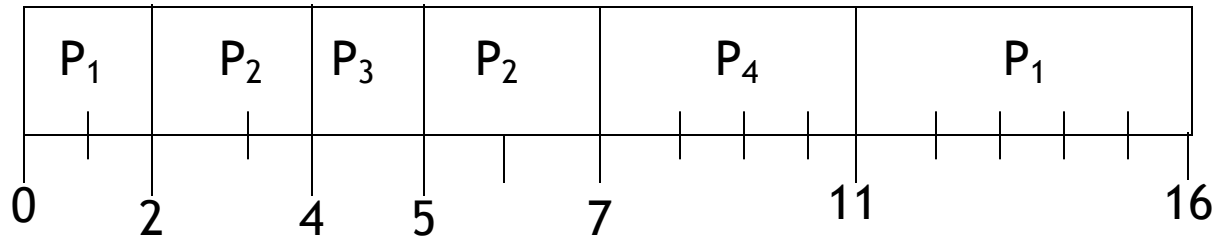
| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |

- SJF (preemptive)

# Example of Preemptive SJF → SRTF

| 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 (preemptive)




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

# Round Robin (RR)

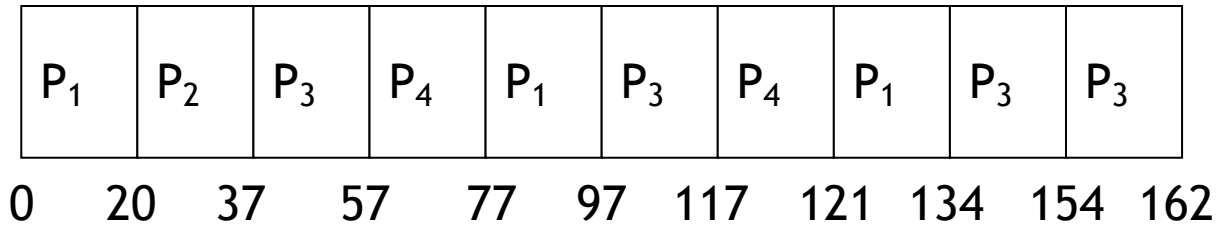
- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds.
- After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Performance
  - *q* large ⇒ FIFO
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high.

| n | q |
|---|---|
| 5 | 10 |

# Example of RR with Time Quantum = 20

| Process | Burst Time |
|---|---|
| $P_1$ | 53 |
| $P_2$ | 17 |
| $P_3$ | 68 |
| $P_4$ | 24 |

- The Gantt chart is:




- Typically, higher average turnaround than SJF, but better *response*.

# Priority Scheduling

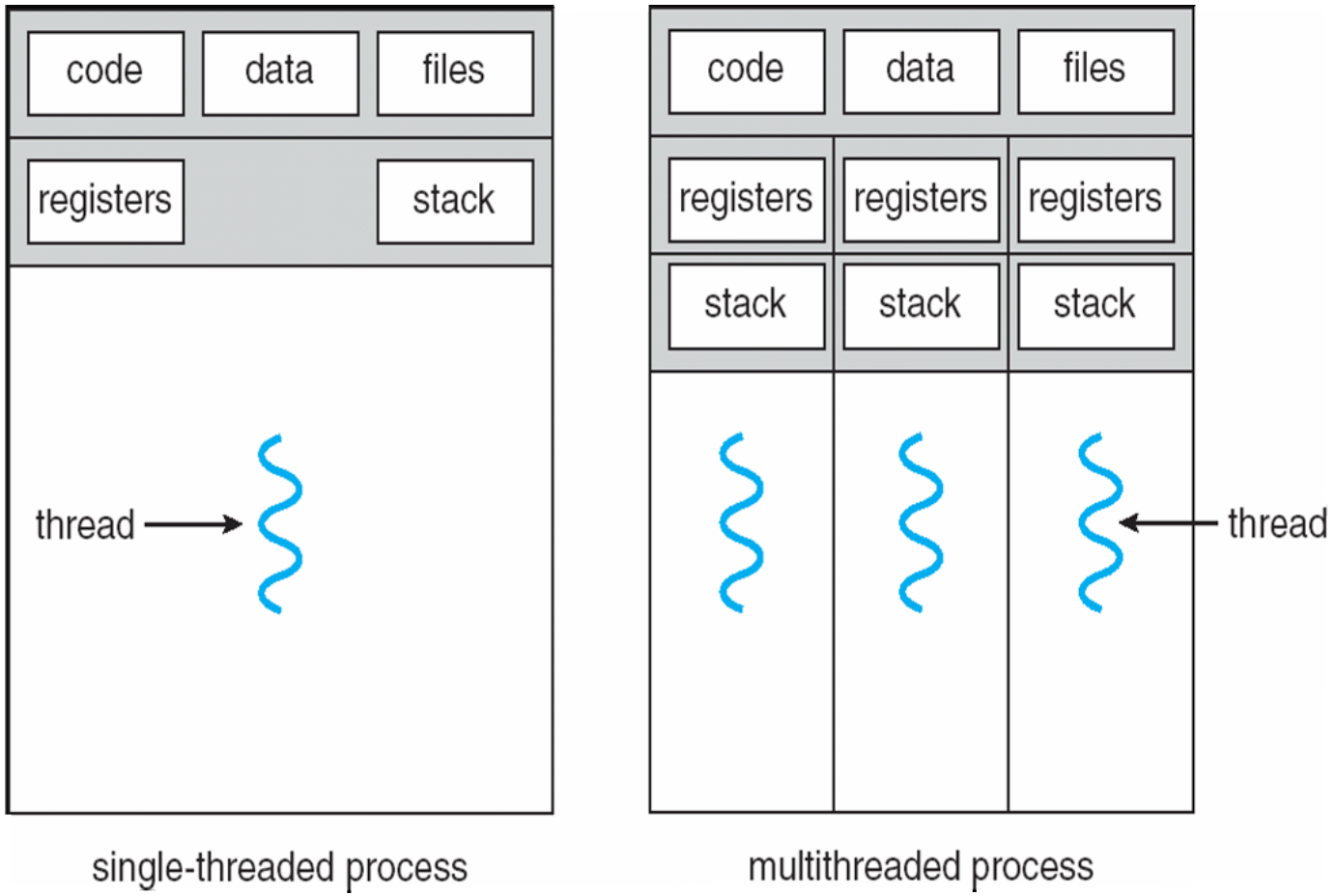
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority ( smallest integer ≡ highest priority).
  - Preemptive
  - nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem ≡ Starvation – low priority processes may never execute.
- Solution ≡ Aging – as time progresses increase the priority of the process.

# THREADS

# Thread Overview

- Threads are mechanisms that permit an application to perform multiple tasks concurrently.
- Thread is a basic unit of CPU utilization
  - Thread ID
  - Program counter
  - Register set
  - Stack
- A single program can contain multiple threads
  - Threads share with other threads belonging to the same process
    - Code, data, open files…….

# Single and Multithreaded Processes




single-threaded process

multithreaded process

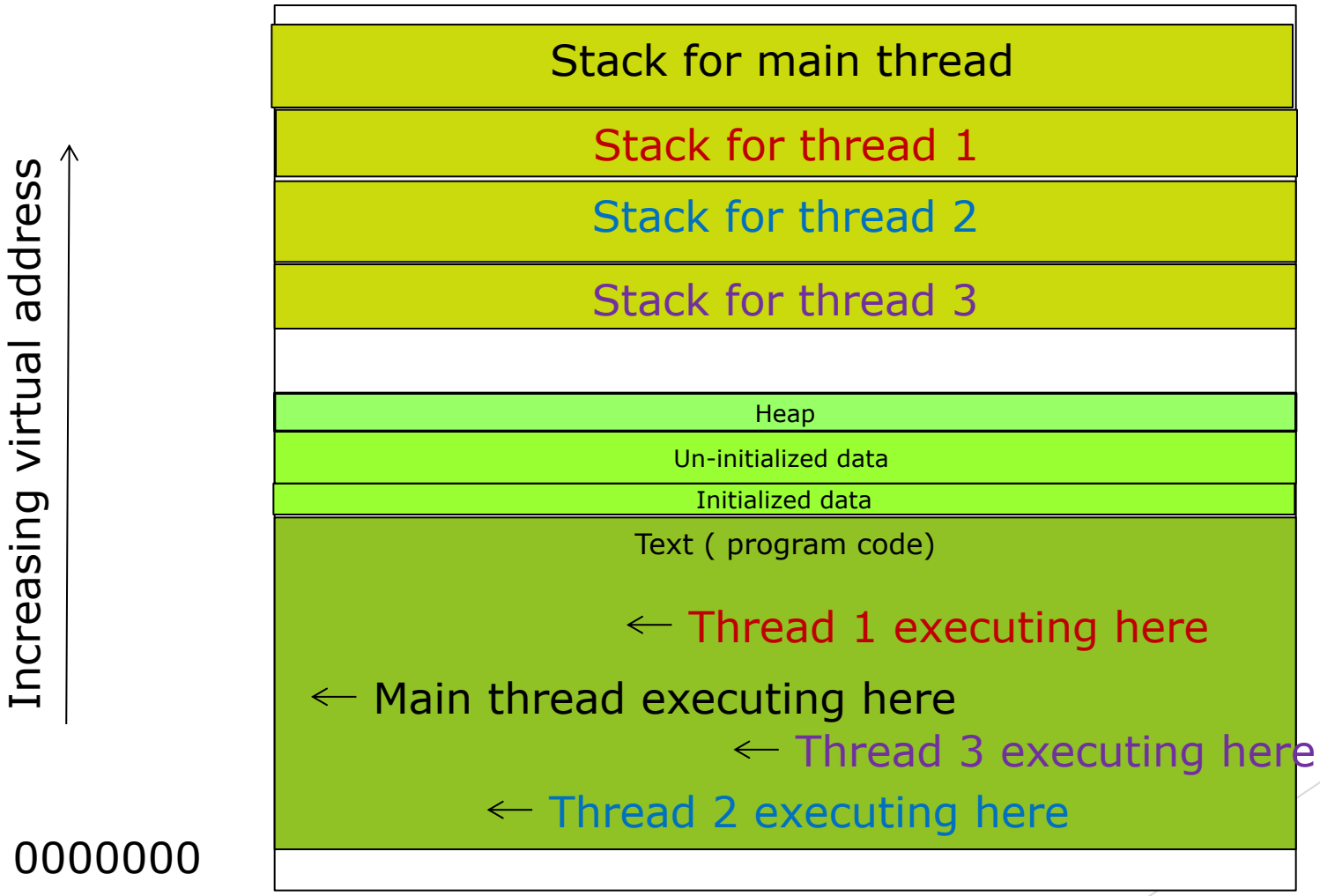
**heavyweight** process

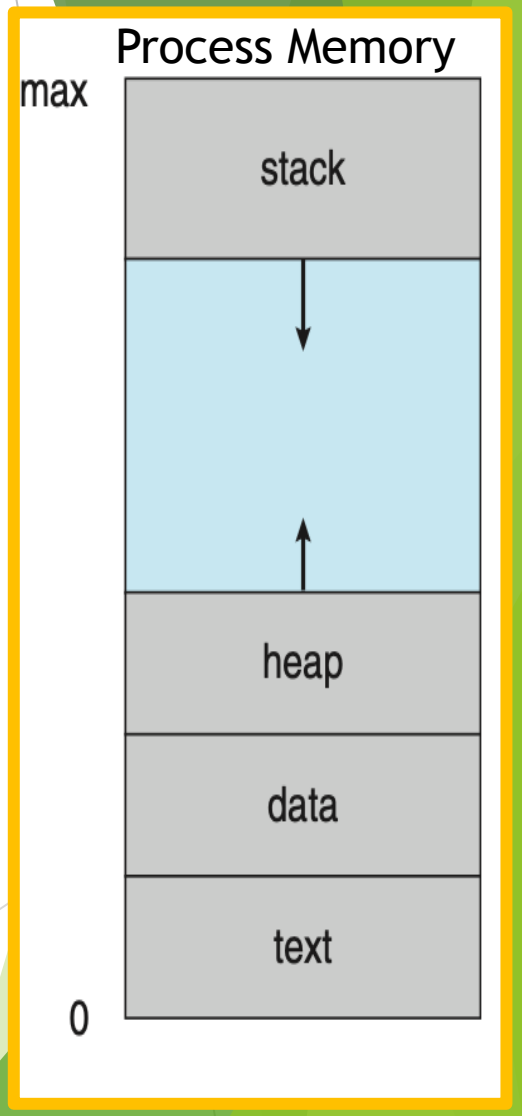
**lightweight** process

Traditional **(heavyweight)** process has a single thread of control

# Threads in Memory

Memory is allocated for a process in segments or parts:



# Threads

## Threads share….

- Global memory
- Process ID and parent process ID
- Controlling terminal
- Process credentials (user )
- Open file information
- Timers
- ……..

## Threads specific Attributes….

- Thread ID
- Thread specific data
- CPU affinity
- Stack (local variables and function call linkage information)
- ……

# Benefits

- **Responsiveness**
  Interactive application can delegate background functions to a thread and keep running

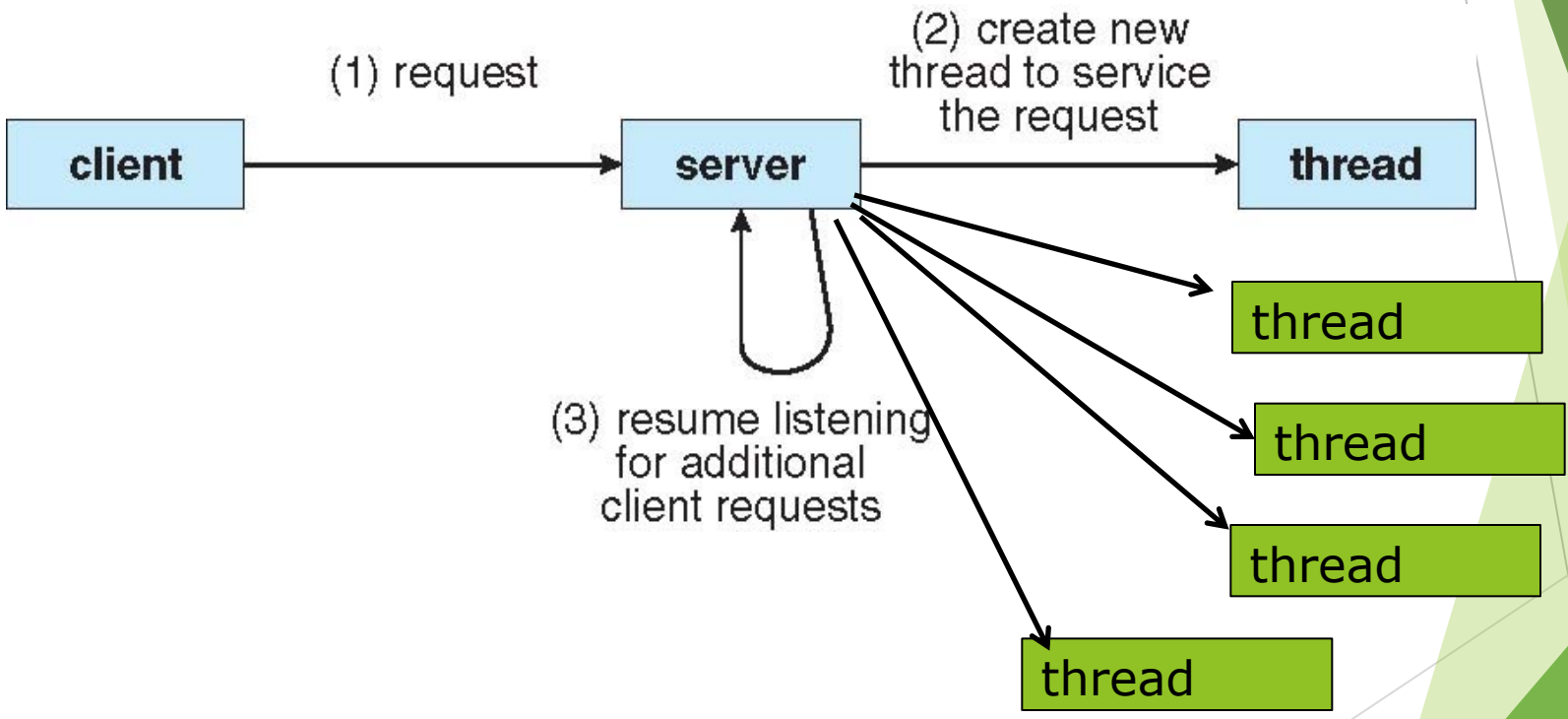
- **Resource Sharing**
  Several different threads can access the same address space

- **Economy**
  Allocating memory and new processes is costly. Threads are much ‘cheaper’ to initiate.

- **Scalability**

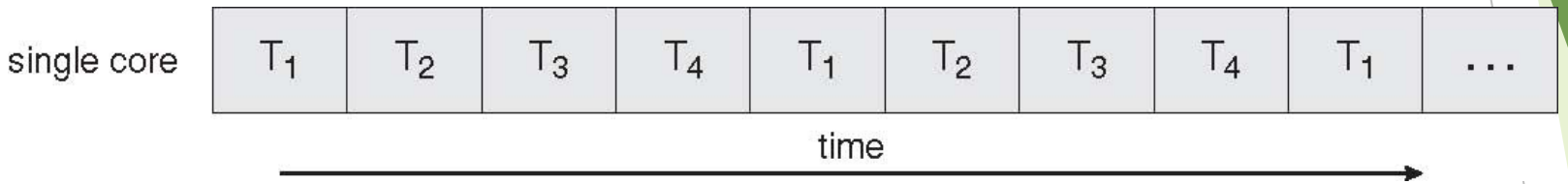
  Use threads to take advantage of multiprocessor architecture

# Multithreaded Server Architecture

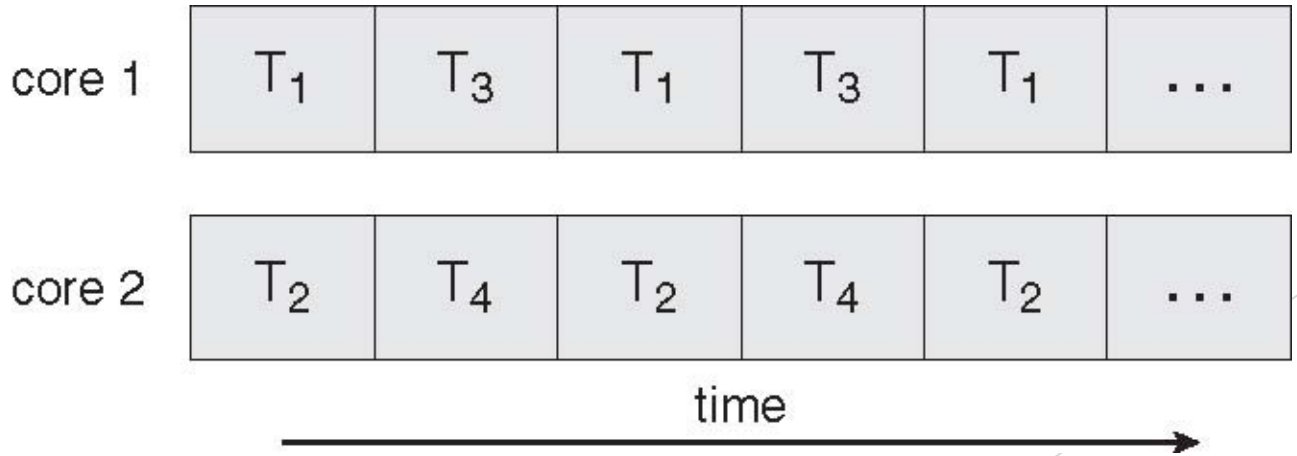


# Multicore Programming

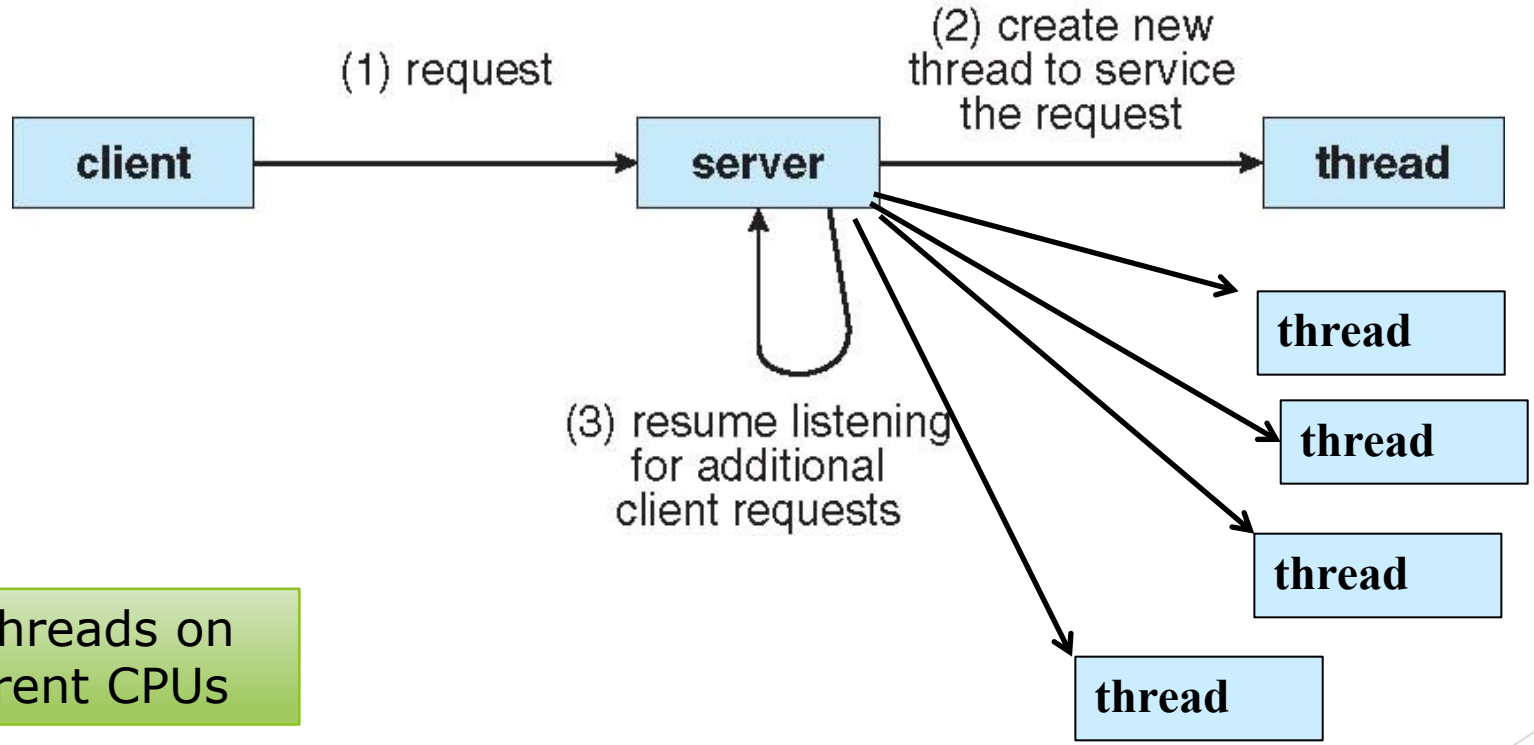
## Concurrent Execution on a Single-core System




## Parallel Execution on a Multicore System



# Threads Assist Multicore Programming




Let threads on different CPUs

# Multithreading Models

- Support provided at either
  - **User level -> user threads**

    Supported above the kernel  and managed without kernel support
  - **Kernel level -> kernel threads**

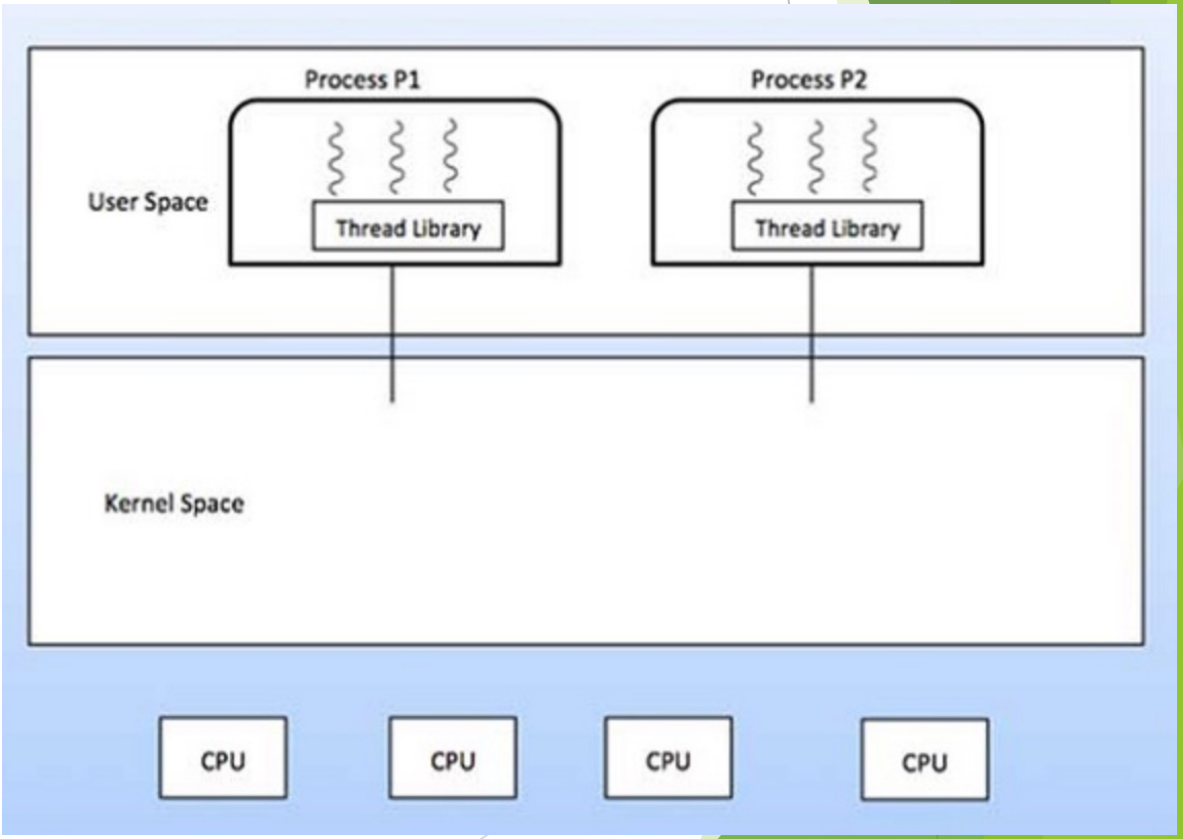
    Supported and managed directly by the operating system

# User Threads

- Thread management done by user-level threads library
- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads

Thread switching does not need to call OS and to cause interrupt to Kernel.

Kernel doesn't know about the user level thread and manages them as if they were single-threaded processes.

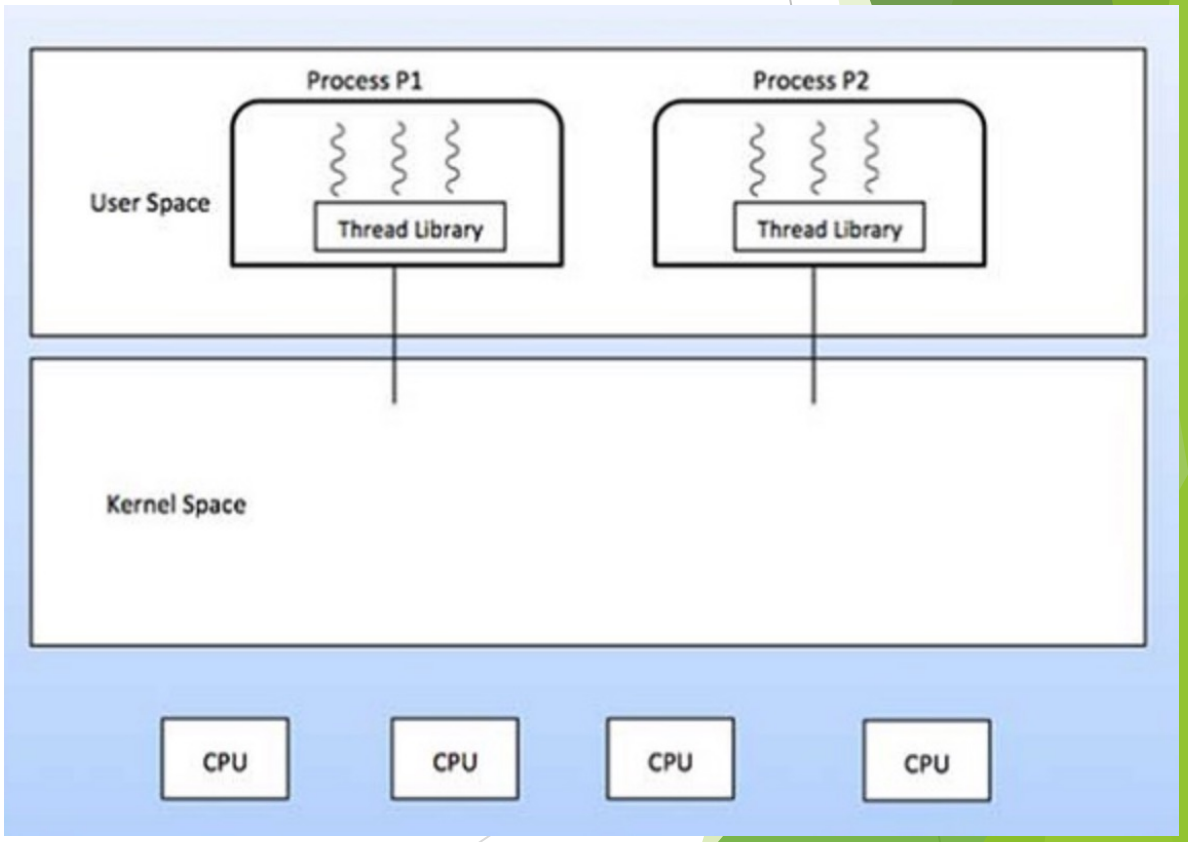


# Kernel Threads

- Supported by the Kernel

  Kernel threads are supported directly by the operating system.

- Kernel-level threads are slower than user-level threads



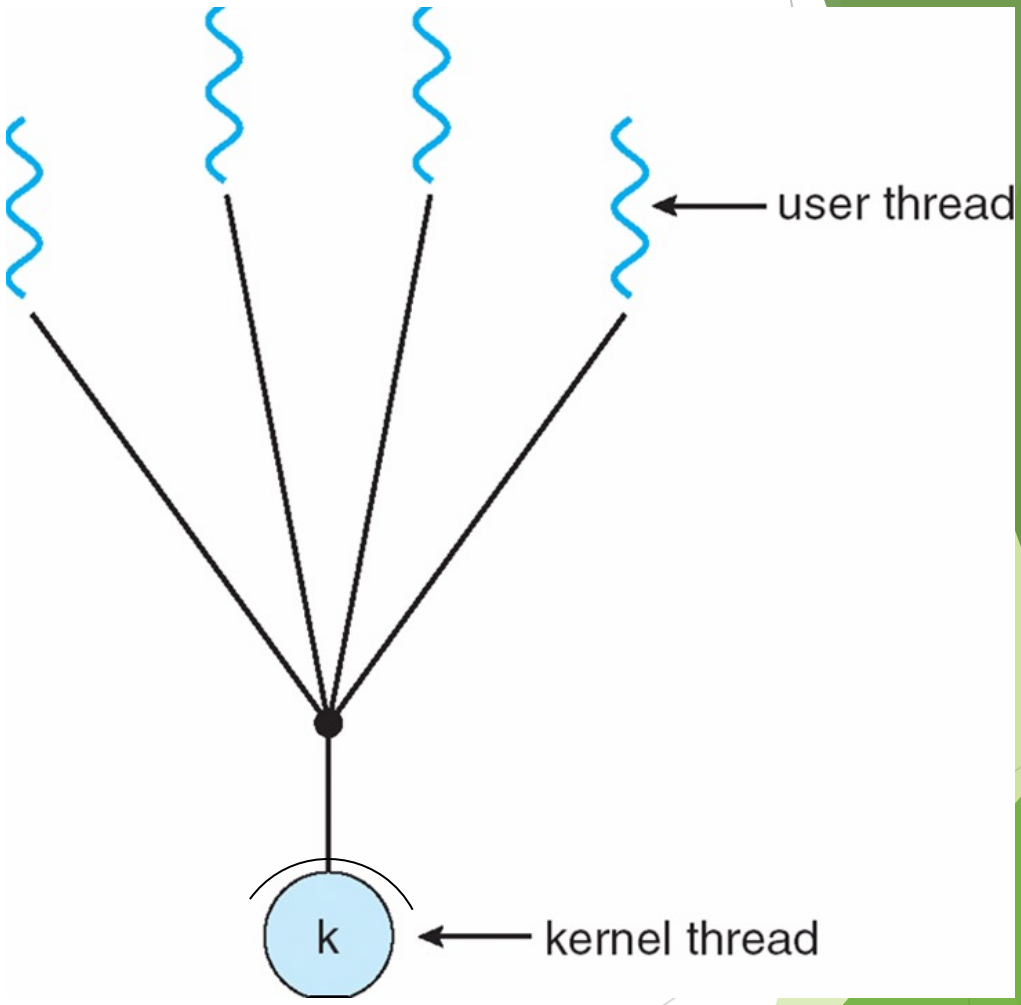
# Multithreading Models

User Thread – to - Kernel Thread

- Many-to-One
- One-to-One
- Many-to-Many

# Many-to-One

Many user-level threads mapped to single kernel thread

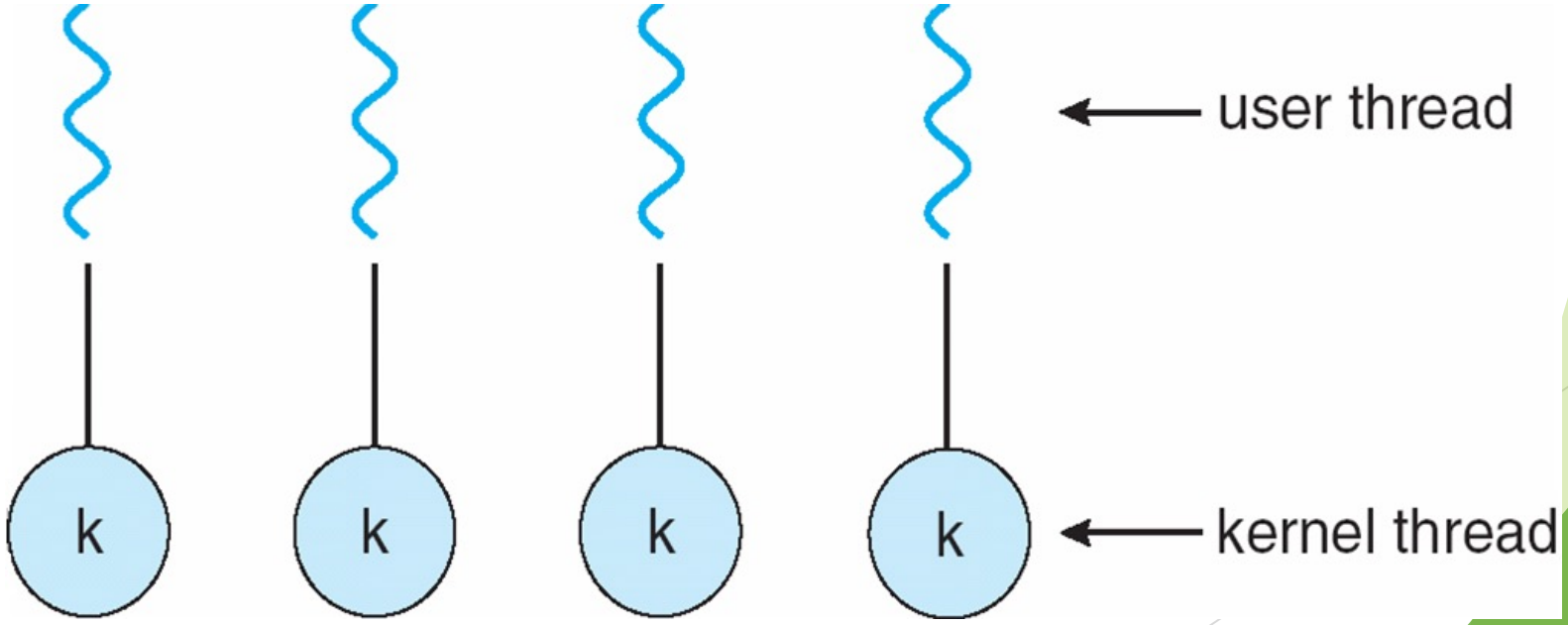


# One-to-One

Each user-level thread maps to kernel thread

Examples

- Windows NT/XP/2000
- Linux

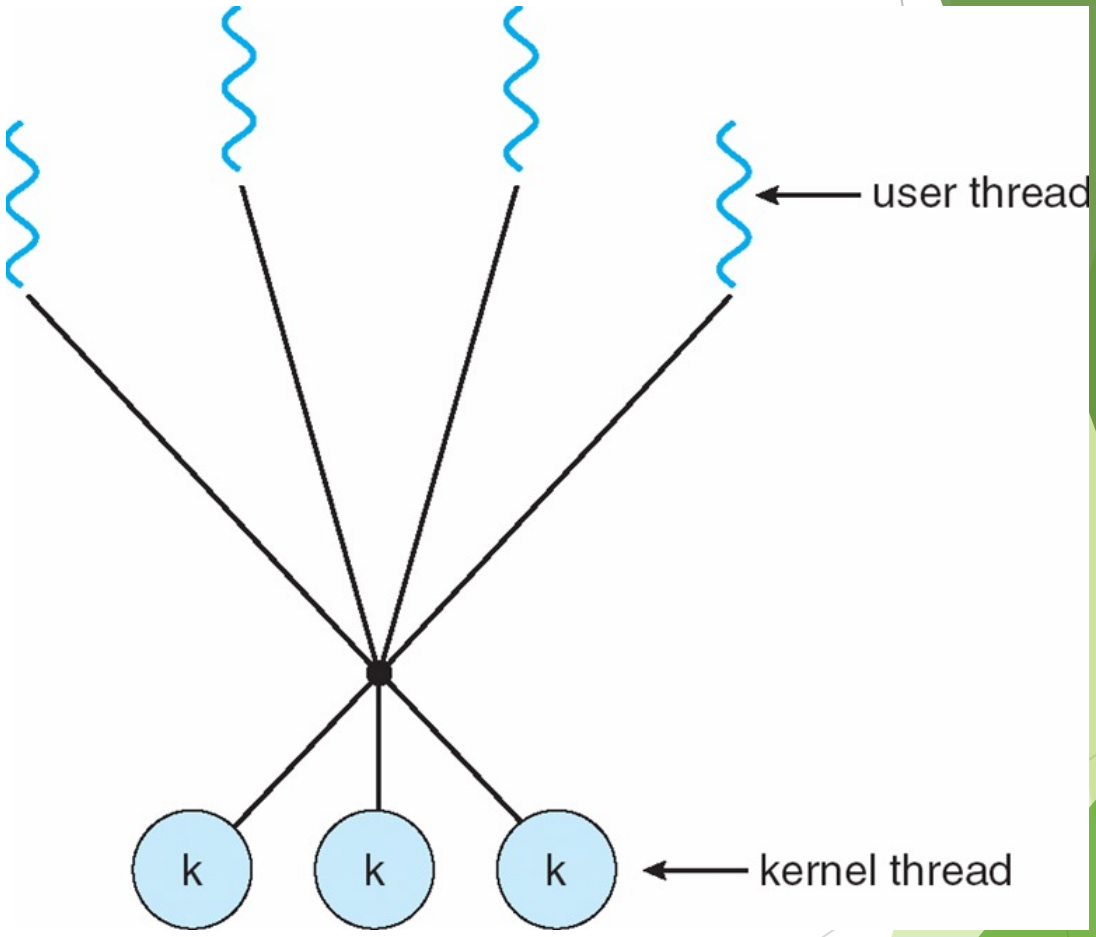


# Many-to-Many Model

Allows many user level threads to be mapped to many kernel threads

Allows the operating system to create a sufficient number of kernel threads

- Example
  - Windows NT/2000 with the *ThreadFiber* package

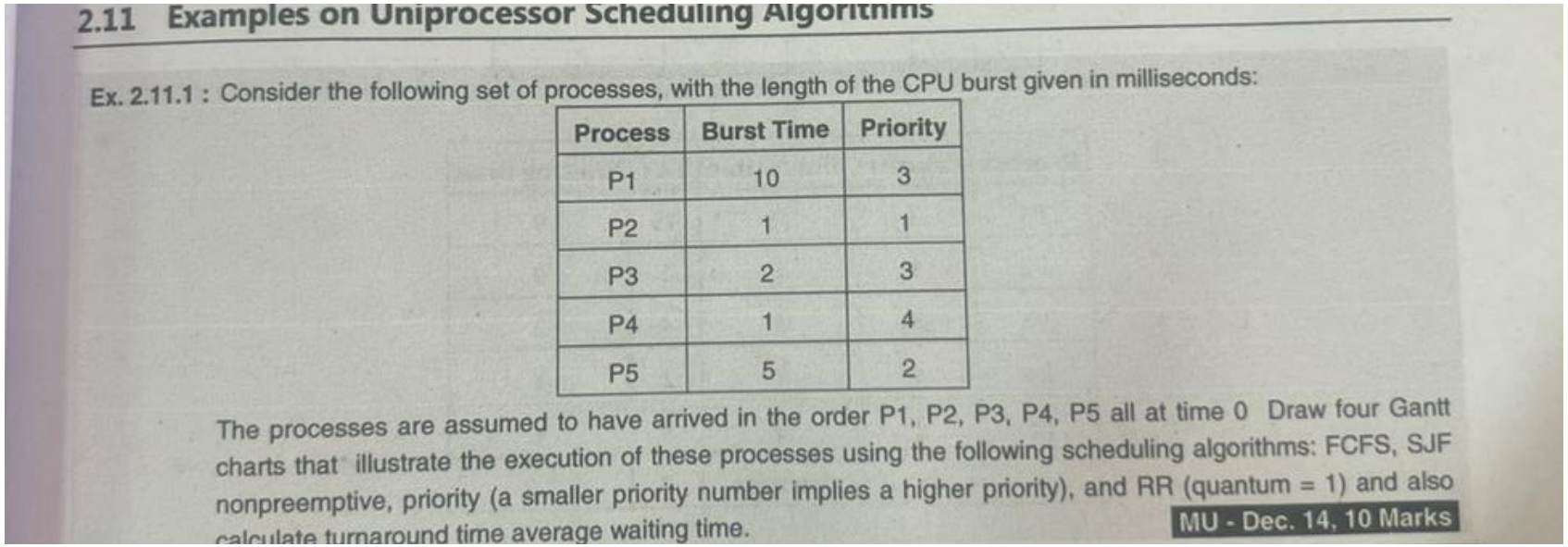


# University Questions

- What do you mean by process? 4 Marks
- **Explain different states of process with diagram.** 5,6,10 marks
- Explain role of PCB. 5 Marks
- Explain LTS, STS and MTS in detail. Differentiate between them. 10 Marks
- Differentiate between thread and process. 5 Marks
- Discuss the importance of multithreading. 2 Marks

# Scheduling Algorithm

**University Questions**



## 2.11 Examples on Uniprocessor Scheduling Algorithms

**Ex. 2.11.1 :** Consider the following set of processes, with the length of the CPU burst given in milliseconds:

| Process | Burst Time | Priority |
|---|---|---|
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0 Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF nonpreemptive, priority (a smaller priority number implies a higher priority), and RR (quantum = 1) and also calculate turnaround time average waiting time.

**MU - Dec. 14, 10 Marks**

# Scheduling Algorithm

## University Questions

**Ex. 2.11.3 :** Consider the four processes P1, P2, P3 and P4 with length of CPU burst time. Find out average waiting time and average turnaround time for the following algorithm.

1. FCFS
2. RR (slice = 4 ms)
3. SJF

| Process | Arrival time | Burst time |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |

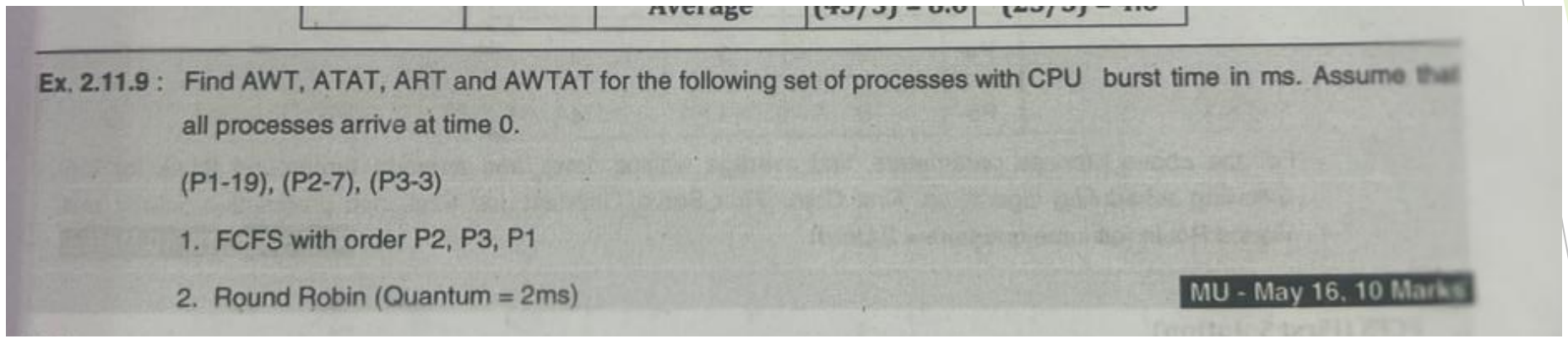
**Ex. 2.11.10 :** Assume the following processes arrive for execution at the time indicated and the length of cpu burst time

| Job | Burst time | Priority | Arrival time |
|---|---|---|---|
| P1 | 8 | 3 | 3 |
| P2 | 1 | 1 | 1 |
| P3 | 3 | 2 | 2 |
| P4 | 2 | 3 | 3 |
| P5 | 6 | 4 | 4 |

For the above process parameters, find average waiting times and average turnaround times for the following scheduling algorithms. First Come First Serve, Shortest Job First, non preemptive priority and Round Robin (assume quantum = 2 Units) **MU - Dec. 16, 10 Marks**

# Scheduling Algorithm

**University Questions**

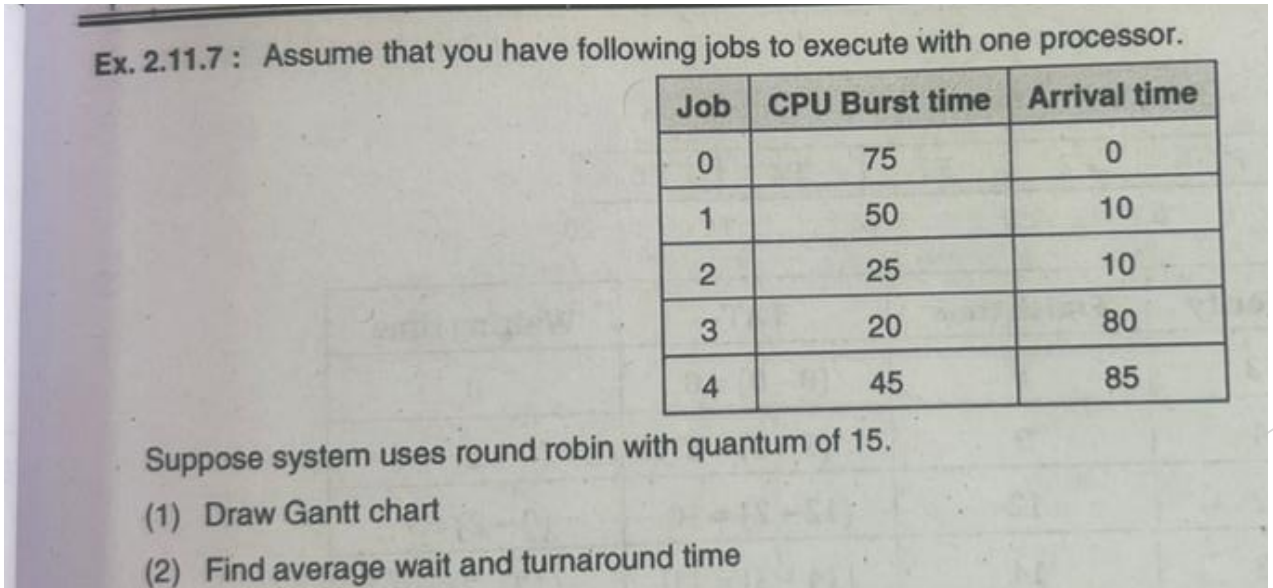


Ex. 2.11.9 : Find AWT, ATAT, ART and AWTAT for the following set of processes with CPU burst time in ms. Assume that all processes arrive at time 0.

(P1-19), (P2-7), (P3-3)

1. FCFS with order P2, P3, P1
2. Round Robin (Quantum = 2ms)

MU - May 16, 10 Marks



Ex. 2.11.7 : Assume that you have following jobs to execute with one processor.

| Job | CPU Burst time | Arrival time |
|---|---|---|
| 0 | 75 | 0 |
| 1 | 50 | 10 |
| 2 | 25 | 10 |
| 3 | 20 | 80 |
| 4 | 45 | 85 |

Suppose system uses round robin with quantum of 15.

(1) Draw Gantt chart

(2) Find average wait and turnaround time

# Scheduling Algorithm

## 2.11 Examples on Uniprocessor Scheduling Algorithms

**Ex. 2.11.1 :** Consider the following set of processes, with the length of the CPU burst given in milliseconds:

| Process | Burst Time | Priority |
|---|---|---|
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0 Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF nonpreemptive, priority (a smaller priority number implies a higher priority), and RR (quantum = 1) and also calculate turnaround time average waiting time.

**MU - Dec. 14, 10 Marks**

# END of UNIT-II