OPERATING SYSTEM
SEM IV
UNIT - II

By,
Himani Deshpande

OPERATING SYSTEM

Course Code	Course Name	Teaching Scheme (Contact Hours)			Credits Assigned			
		Theory	Practical	Tutorial	Theory	Practical /Oral	Tutorial	Total
ITC403	Operating System	03	--	--	03	--	--	03

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

SYLLABUS

Sr. No.	Module	Detailed Content	Hours	CO Mapping
0	Prerequisite	Programming Language C; Basic of Hardware i.e. ALU, RAM, ROM, HDD, etc.; Computer-System Organization.	02	-
I	Fundamentals of Operating System	Introduction to Operating Systems; Operating System Structure and Operations; Functions of Operating Systems; Operating System Services and Interface; System Calls and its Types; System Programs; Operating System Structure; System Boot. Self-learning Topics: Study of any three different OS.	03	CO1
II	Process Management	Basic Concepts of Process; Operation on Process; Process State Model and Transition; Process Control Block; Context Switching; Introduction to Threads; Types of Threads, Thread Models; Basic Concepts of Scheduling; Types of Schedulers; Scheduling Criteria; Scheduling Algorithms. Self-learning Topics: Study the comparison between Scheduling Algorithms.	06	CO2
III	Process Coordination	Basic Concepts of Inter-process Communication and Synchronization; Race Condition; Critical Region and Problem; Peterson's Solution; Synchronization Hardware and Semaphores; Classic Problems of Synchronization; Message Passing; Introduction to Deadlocks; System Model, Deadlock Characterization; Deadlock Detection and Recovery; Deadlock Prevention; Deadlock Avoidance. Self-learning Topics: Study a real time case study for Deadlock detection and recovery.	09	CO3
IV	Memory Management	Basic Concepts of Memory Management; Swapping; Contiguous Memory Allocation; Paging; Structure of Page Table; Segmentation; Basic Concepts of Virtual Memory; Demand Paging, Copy-on Write; Page Replacement Algorithms; Thrashing. Self-learning Topics: Implement Page Replacement Algorithm.	09	CO4
V	Storage Management	Basic Concepts of File System; File Access Methods; Directory Structure; File-System Implementation; Allocation Methods; Free Space Management; Overview of Mass-Storage Structure; Disk Structure; Disk Scheduling; RAID Structure; Introduction to I/O Systems. Self-learning Topics: Study the advantages and disadvantages of RAID.	06	CO5
VI	**Special-purpose Operating Systems**	**Open-source and Proprietary Operating System; Fundamentals of Distributed Operating System; Network Operating System; Embedded Operating Systems; Cloud and IoT Operating Systems; Real-Time Operating System; Mobile Operating System; Multimedia Operating System; Comparison between Functions of various Special-purpose Operating Systems.** **Self-learning Topics: Study any one case study on Module VI.**	04	CO6

SUGGESTED BOOKS

- A. Silberschatz, P. Galvin, G. Gagne, Operating System Concepts, 10th ed., Wiley, 2018.
- W. Stallings, Operating Systems: Internal and Design Principles, 9th ed., Pearson, 2018.
- A. Tanenbaum, Modern Operating Systems, Pearson, 4th ed., 2015.

*(pdf on teams)

UNIT- II (PROCESS MANAGEMENT)

- Basic Concepts of **Process;**
  Operation on Process;
  Process State Model and Transition;
  Process Control Block;
  Context Switching;

- Introduction to **Threads;**
  Types of Threads, Thread Models;
  Basic Concepts of Scheduling;
  Types of Schedulers;
  Scheduling Criteria;
  Scheduling Algorithms.

GALVIN CHAPTER -4

- 4.1 Overview of threads
  - 4.1.1 Motivation
  - 4.1.2 Benefits
- 4.2 Multicore Programming
  - 4.2.1 Programming Challenges
  - 4.2.2 Types of parallelism
- 4.3 Multithreading Models
  - 4.3.1 Many to one model
  - 4.3.2 One to One Model
  - 4.3.3 Many to Many Model
- 4.4 Thread Libraries
  - 4.4.1 Pthreads
  - 4.4.2 Windows Threads
  - 4.4.3 Java Threads
- 4.5 Implicit Threading
  - 4.5.1 Implicit Threading
  - 4.5.2 Fork Join
  - 4.5.3 OpenMP
  - 4.5.4 Grand Central Dispatch
  - 4.5.5 Intel Thread Building Blocks
- 4.6 Threading Issues
  - 4.6.1 The fork( ) and exec( ) System Calls
  - 4.6.2 Signal Handling
  - 4.6.3 Thread Cancellation
  - 4.6.4 Thread-Local Storage
  - 4.6.5 Scheduler Activation
- 4.7 Operating-System Examples

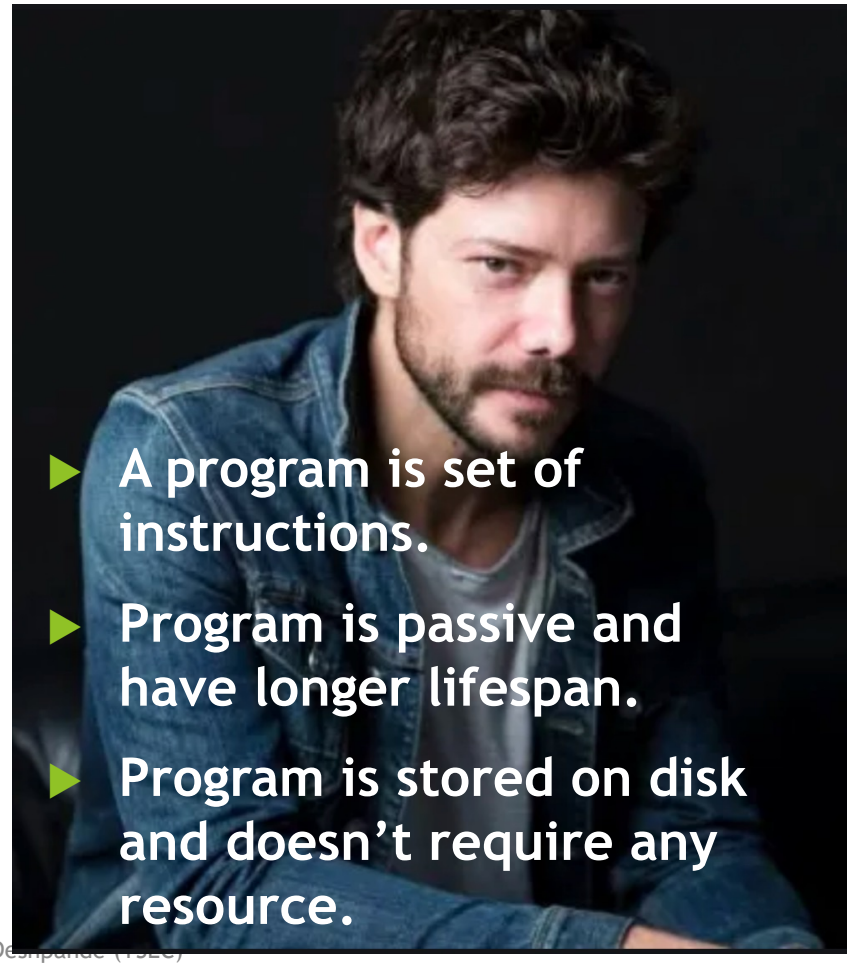
Introduction to Threads;
Types of Threads,
Thread Models;
Basic Concepts of Scheduling;
Types of Schedulers;
Scheduling Criteria;
Scheduling Algorithms.

UNIT- III (PROCESS COORDINATION)

- Basic Concepts of Inter-process Communication and Synchronization
  Race Condition;
  Critical Region and Problem;
  Peterson's Solution;
  Synchronization Hardware and Semaphores;
  Classic Problems of Synchronization;
  Message Passing;
  Introduction to Deadlocks;
  System Model, Deadlock Characterization;
  Deadlock Detection and Recovery;
  Deadlock Prevention;
  Deadlock Avoidance.

Program Vs process

Alvaro Morte



Professor

Program vs Process

Chai Tea Recipe:

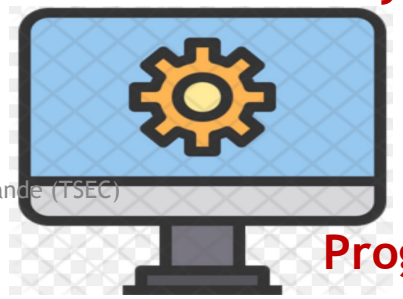
1. *Add water in pan.*
2. *Add sugar, tea leaves, spices.*
3. *Bring to boil and simmer.*
4. *Add milk.*
5. *Bring to boil and simmer.*
6. *Strain tea in teapot.*

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
- A process includes:
  - program counter
  - stack
  - data section
  - heap

In Linux, how do you see process info and what can you see will be taken in Unix lab

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.

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

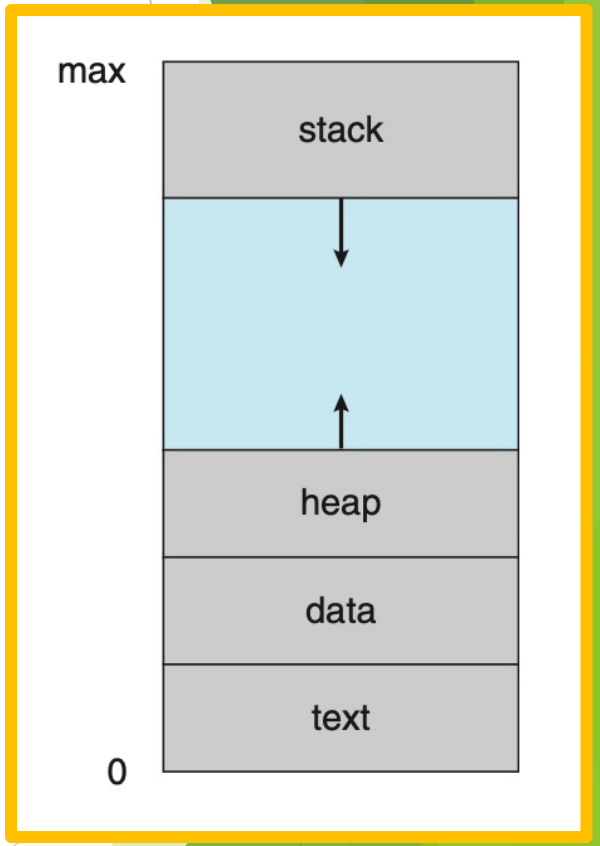
  Two common techniques for loading executable files are

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

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)

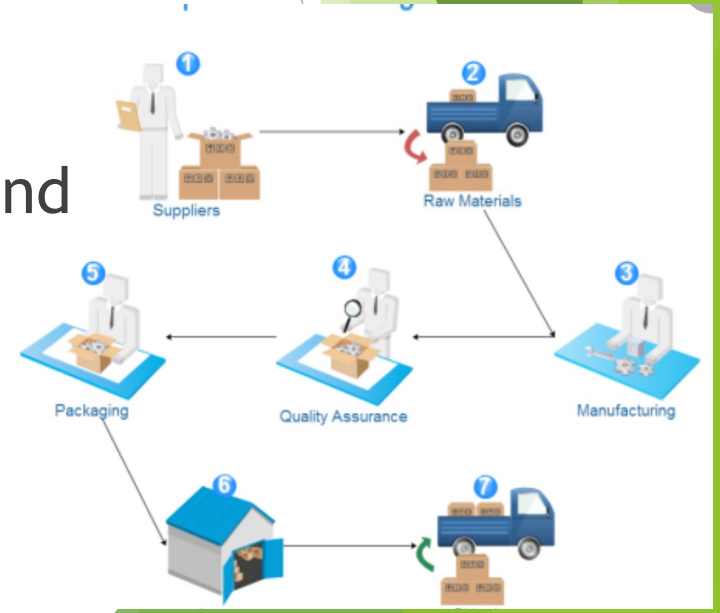


The sizes of the text and data sections are fixed, as their sizes do not change during program run time.

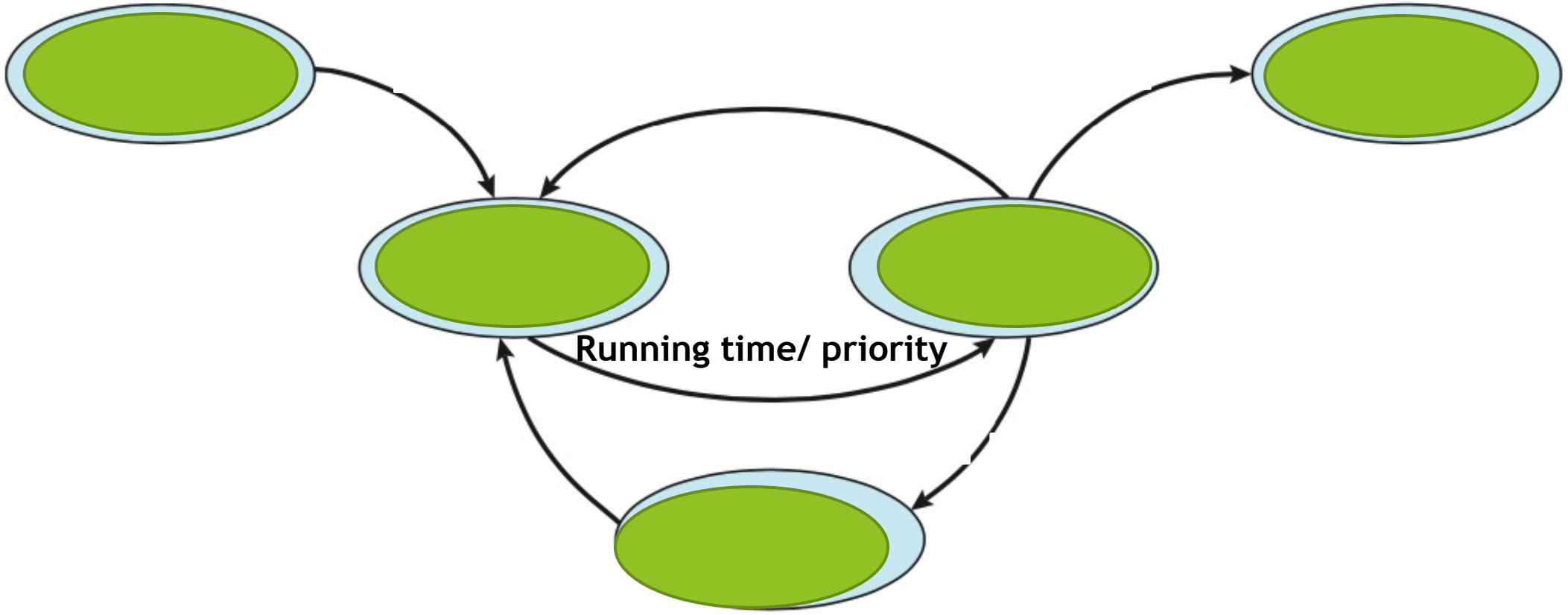
However, the stack and heap sections can shrink and grow dynamically during program execution.

States of Process

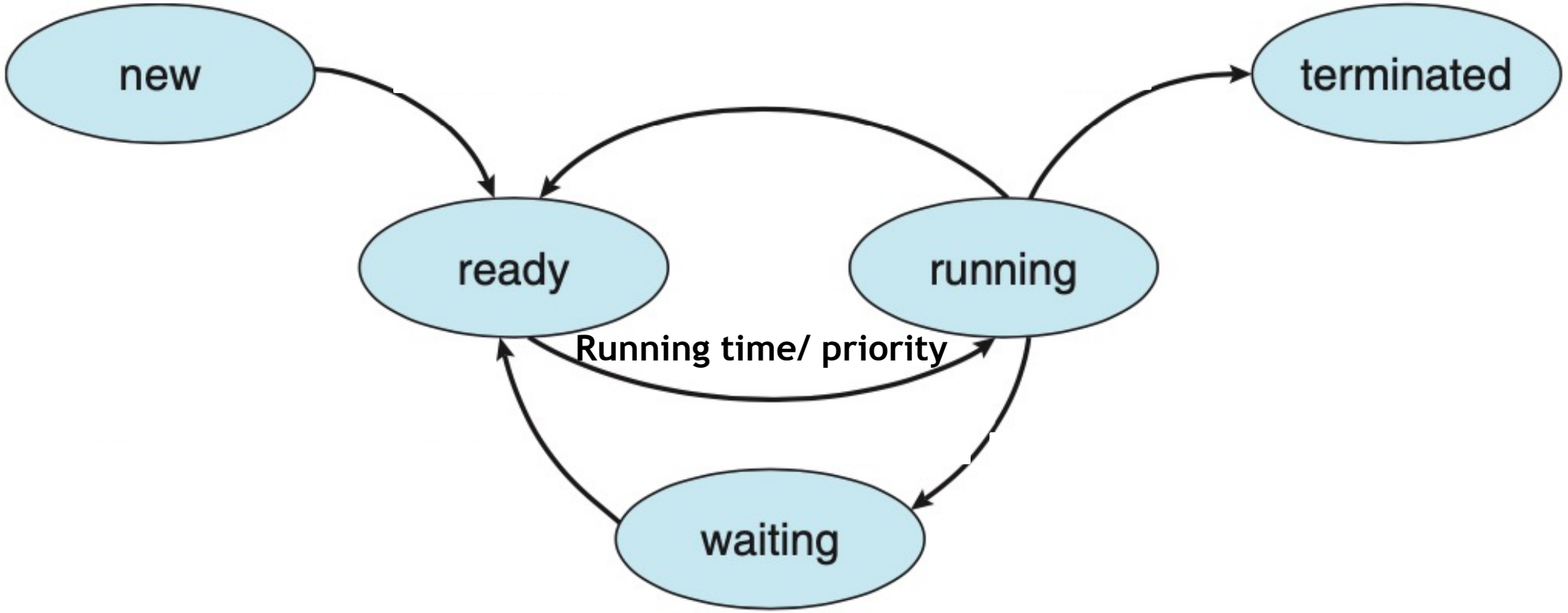
- When a process executes, it passes through different states.
- State of a process defines the current activity of that process.
- These stages may differ in different operating systems, and the names of these states are also not standardized.



States of Process

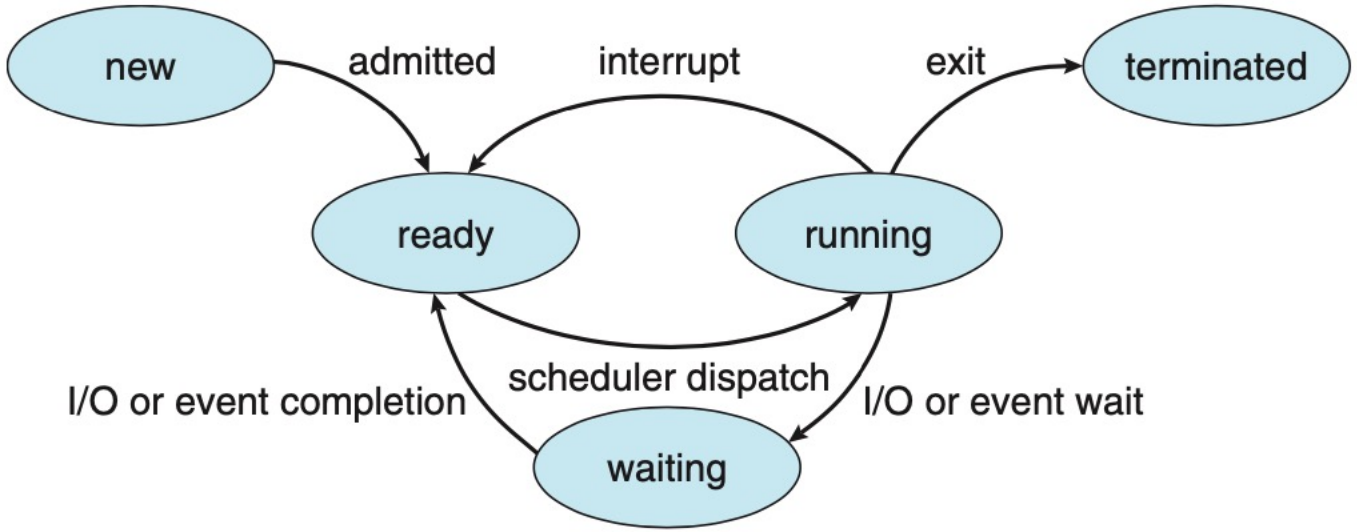


States of Process



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)

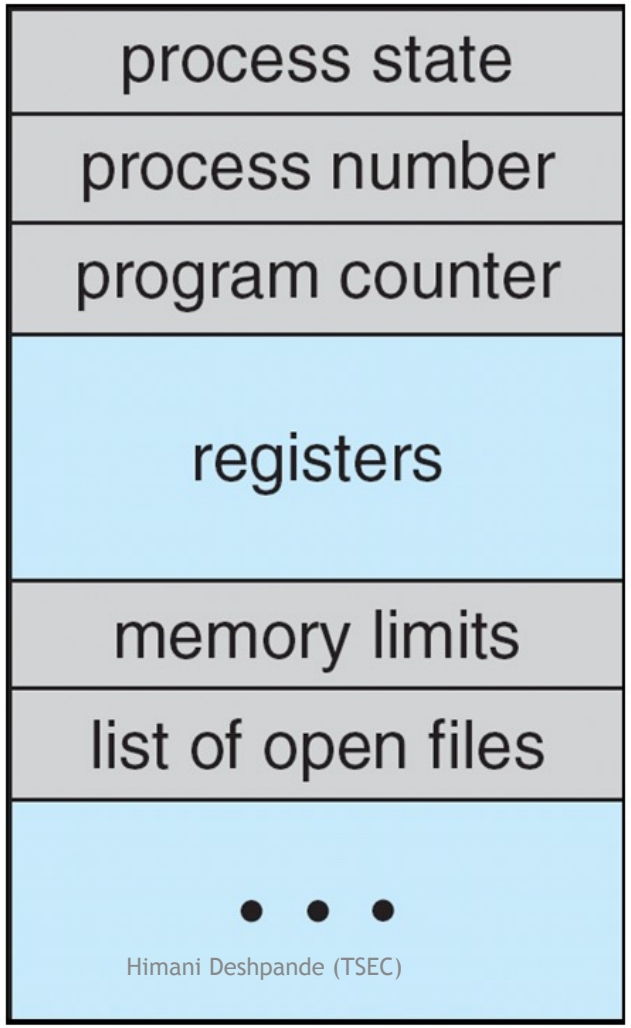
- As the operating system supports multi-programming, it needs to keep track of all the processes.
- Each process is represented in the operating system by a process control block (PCB)—also called a task control block.
- It contains many pieces of information associated with a specific process.
- All 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.

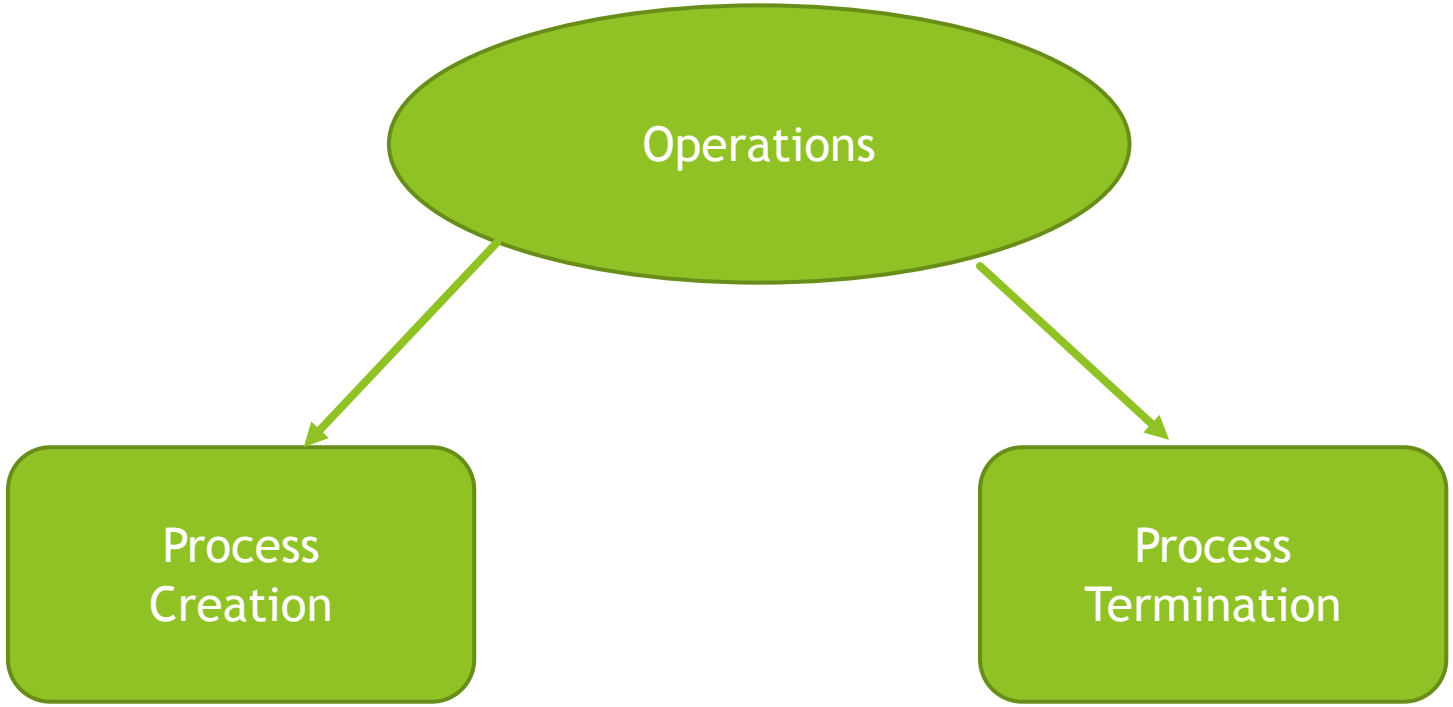
CPU scheduling information: This includes process priority, pointers to scheduling queues and any scheduling parameters.

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

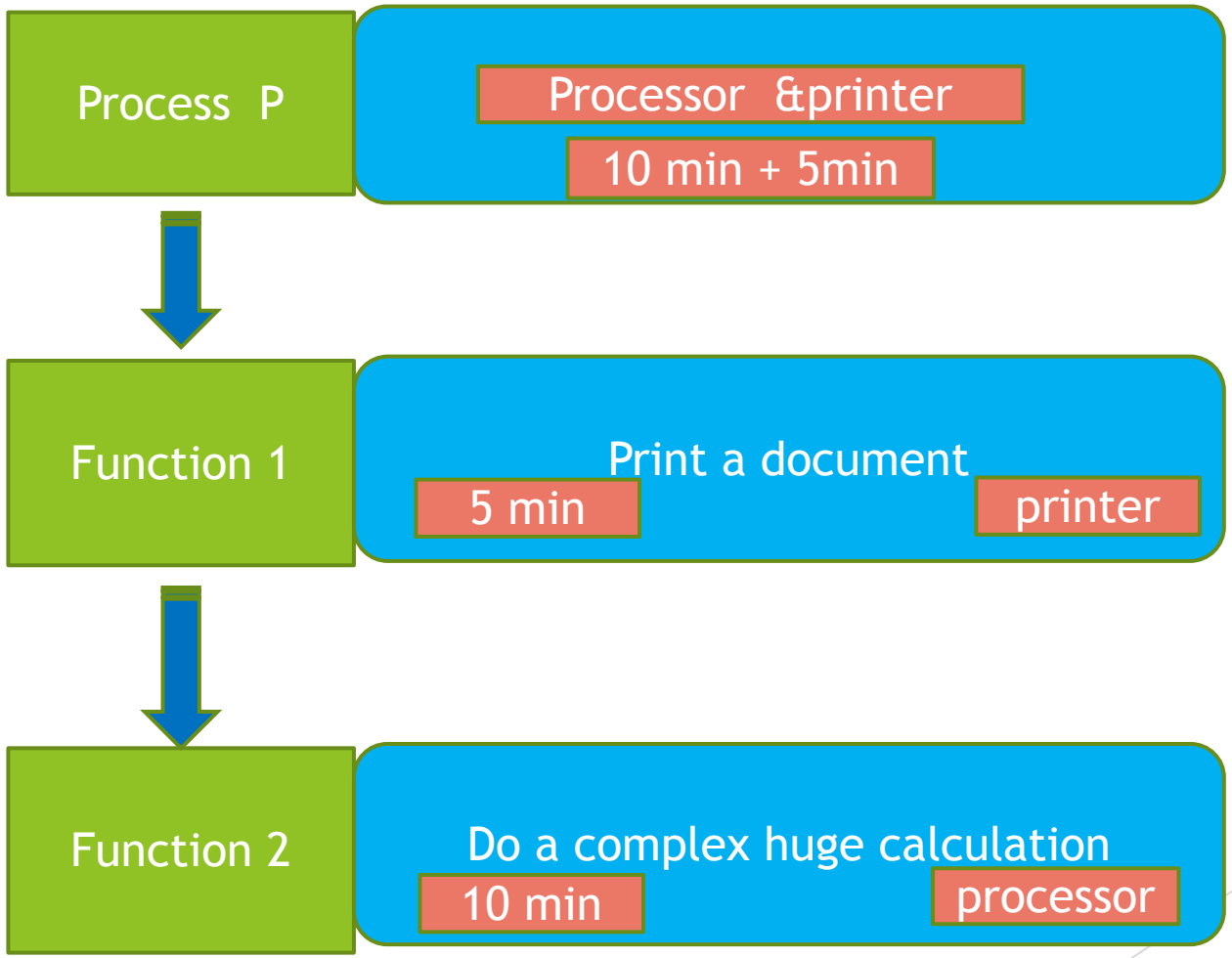
Accounting information: It includes amount of CPU and real time used, account numbers, process numbers etc

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

Operations On Process

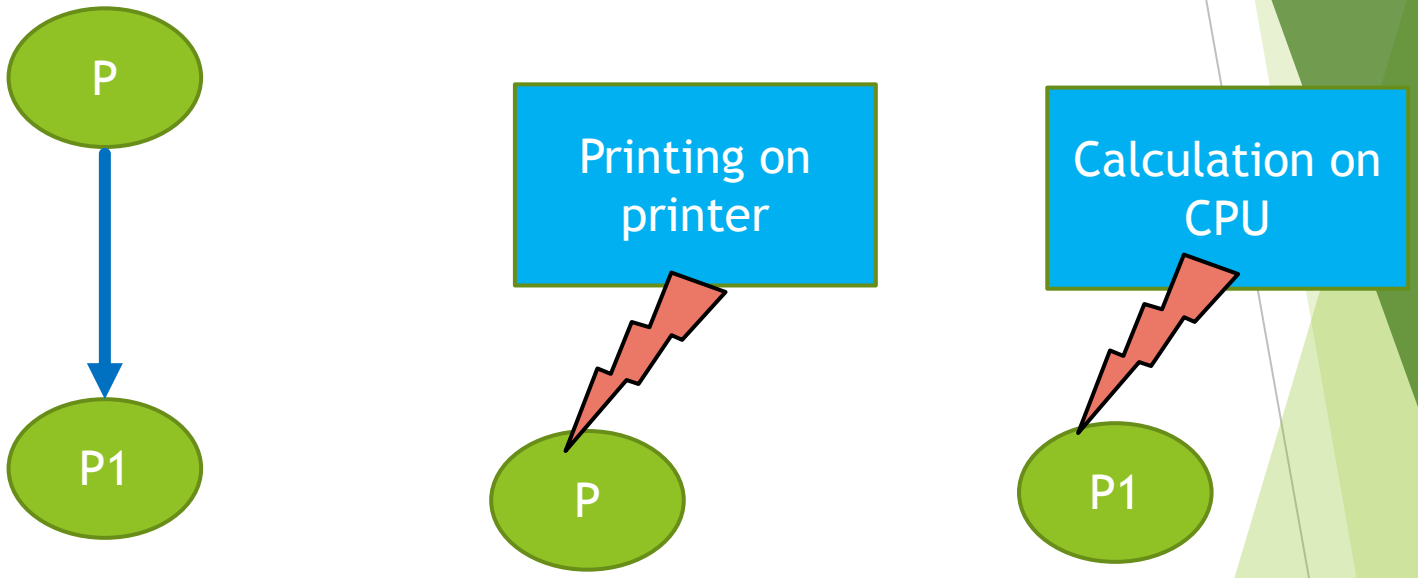


Single process execution



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

Resource sharing options

- Parent and children share all resources
- Children share subset of parent's resources
- Parent and child share no resources

Process creation

Parent

For execution parent process can either

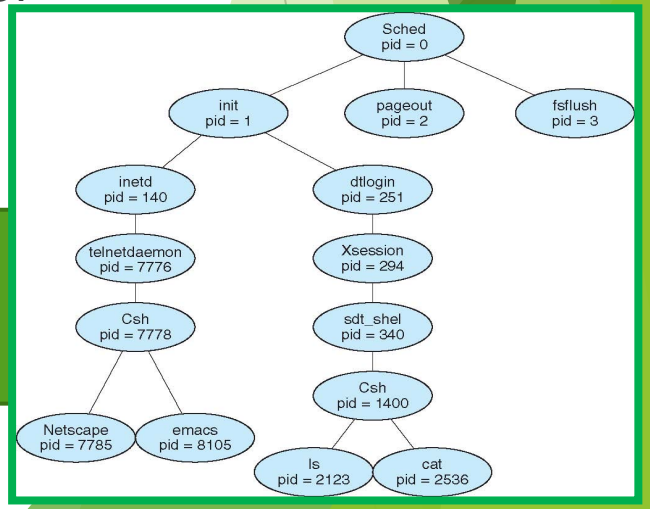
- Continues to execute concurrently with its children. Or
- Wait until some or all the children terminates.

Child

For address-space child process

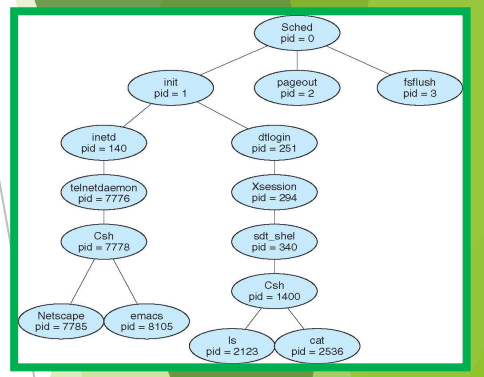
- Can be a duplicate of the parent process.
  (it has the same program and data as the parent). Or
- Has a new program loaded into it.

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



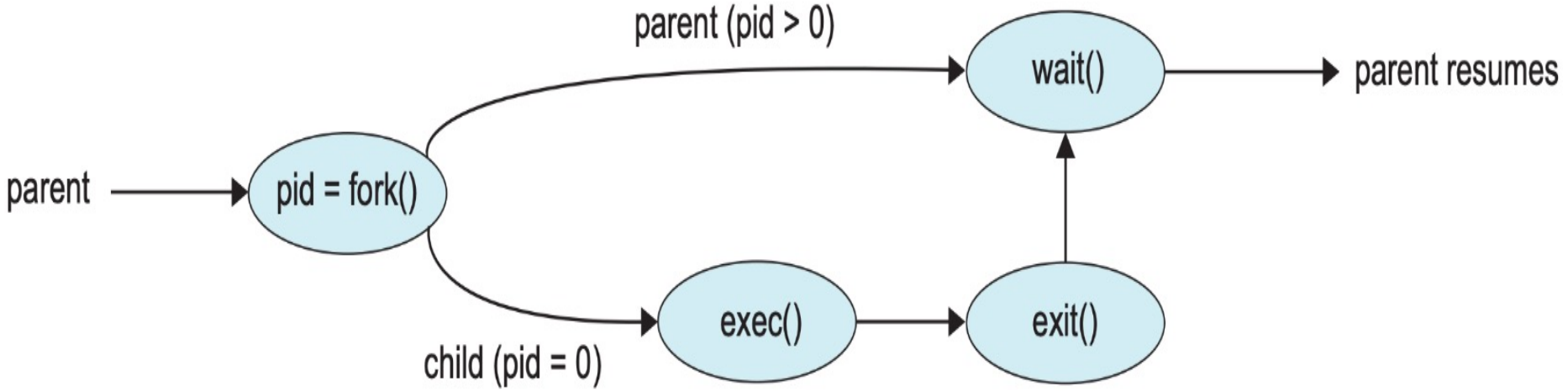
Process Termination

- 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**

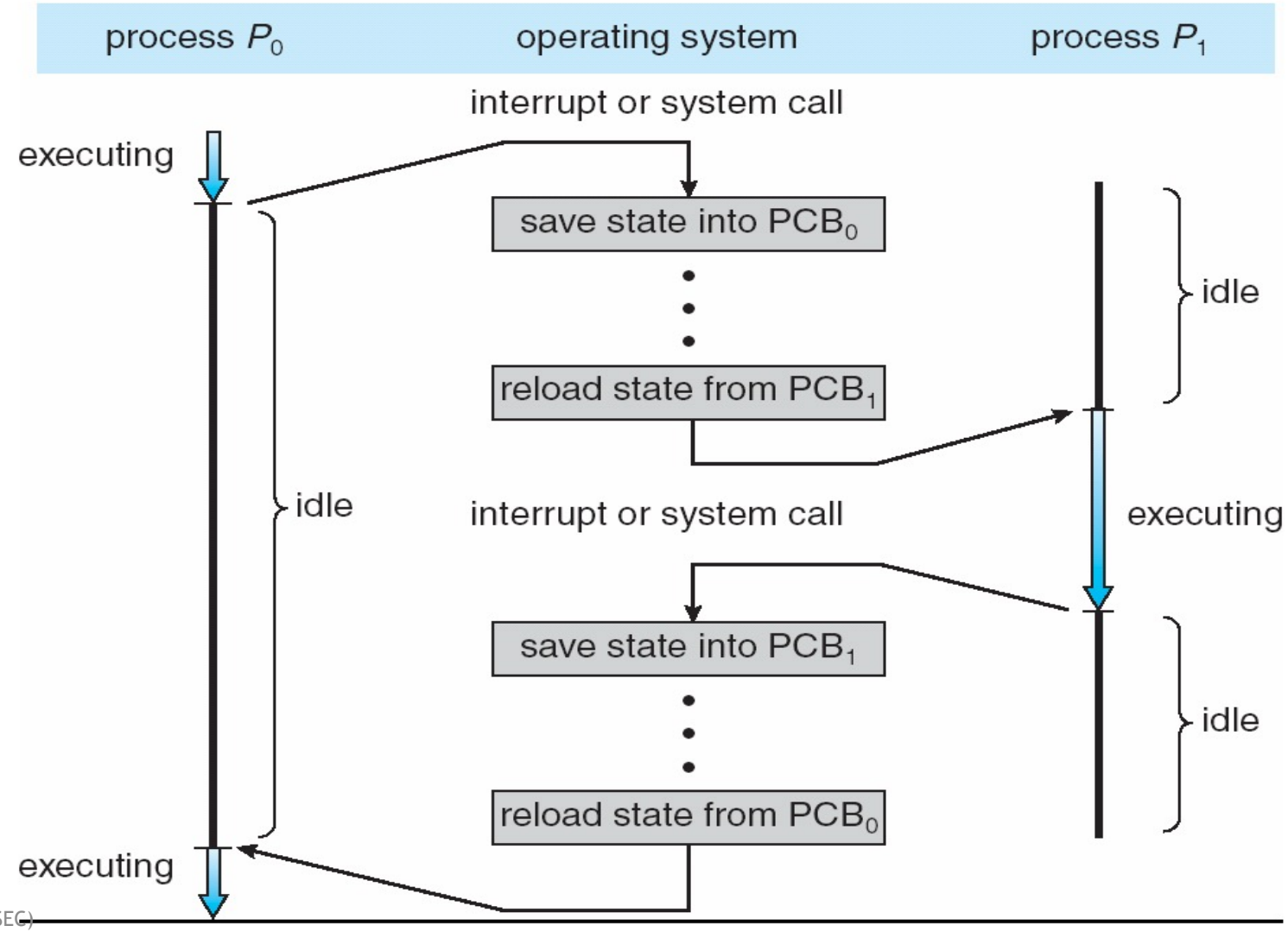
Process Creation



Context Switch

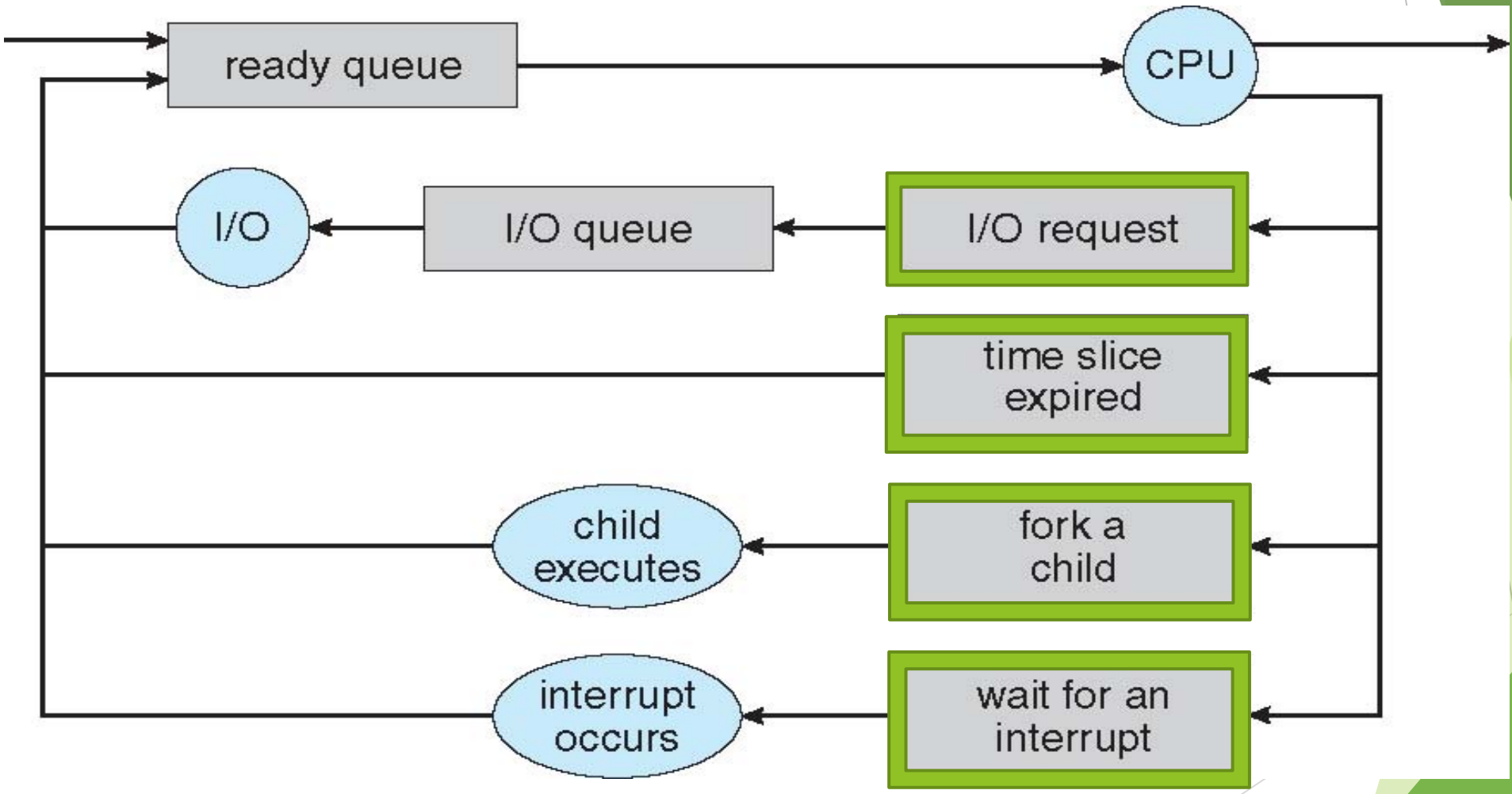
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**.
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The 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

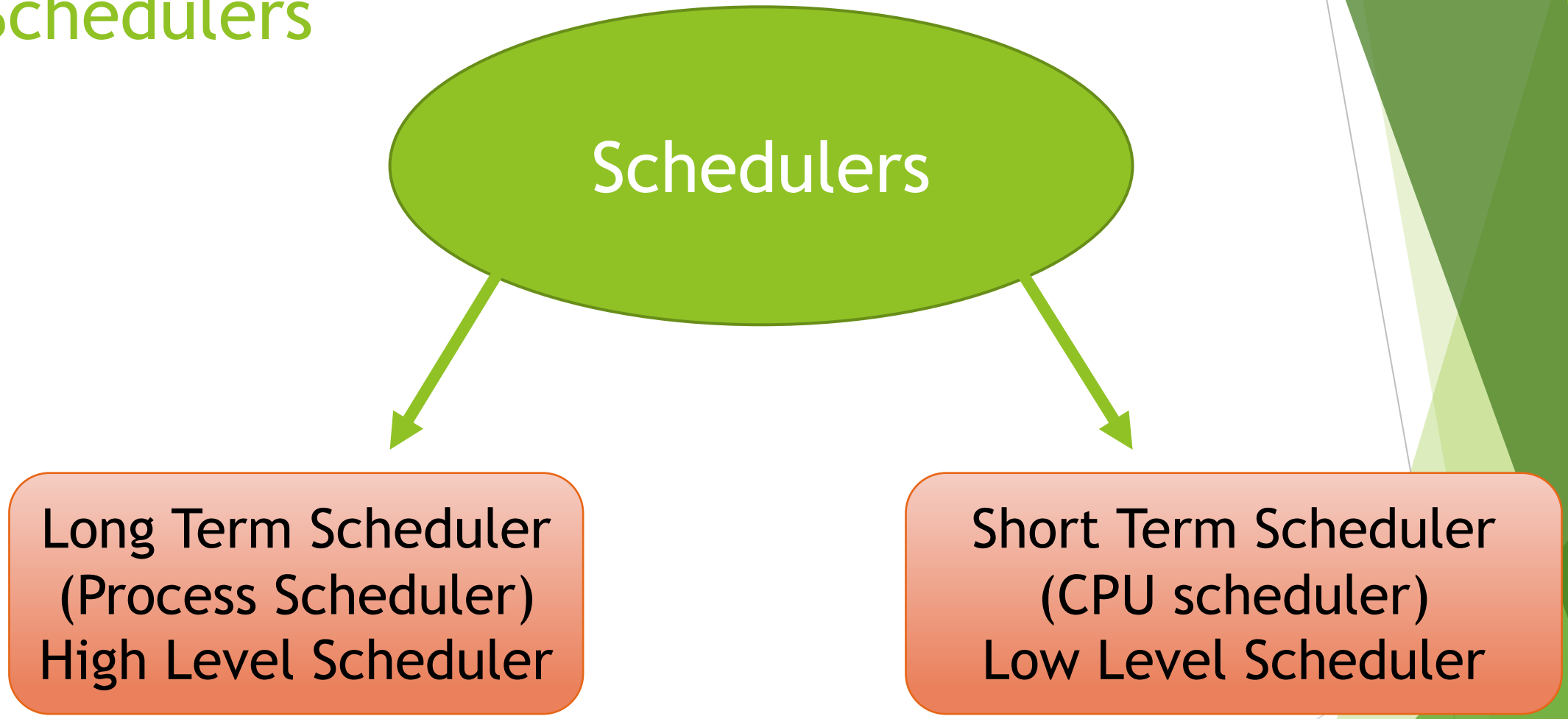


SCHEDULING

Representation of Process Scheduling



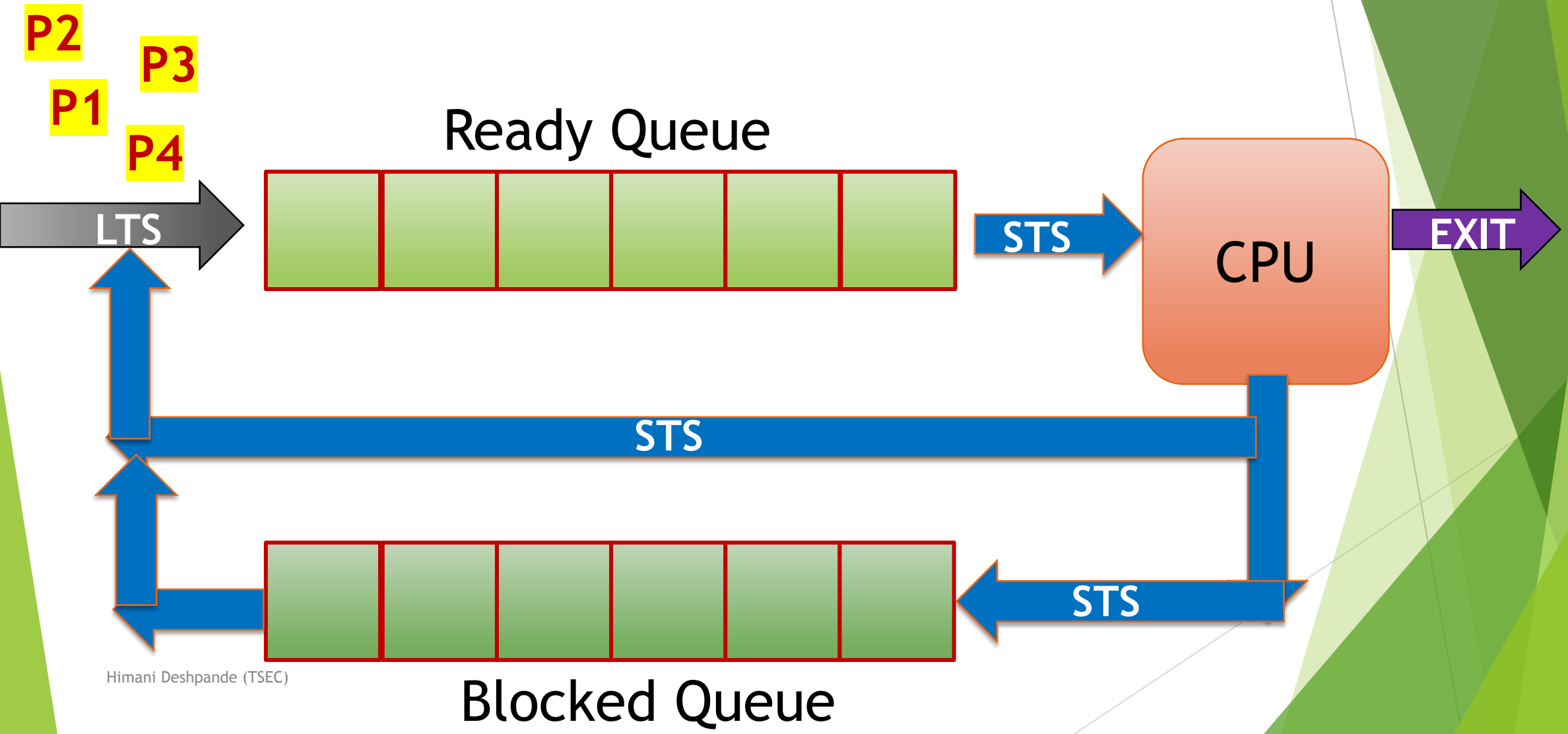
Schedulers


Schedulers
Long Term Scheduler
(Process Scheduler)
High Level Scheduler
Short Term Scheduler
(CPU scheduler)
Low Level Scheduler

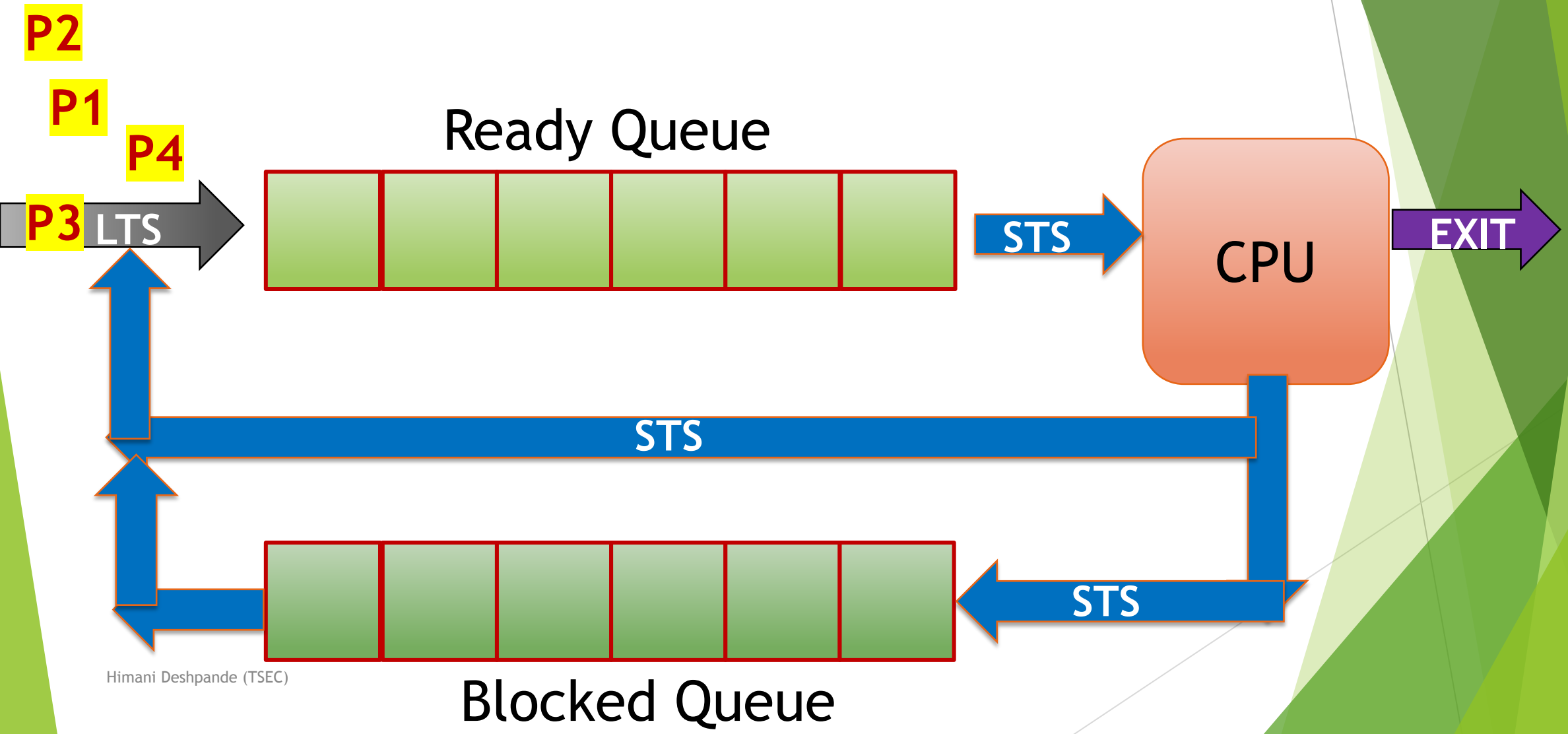
Schedulers

- **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

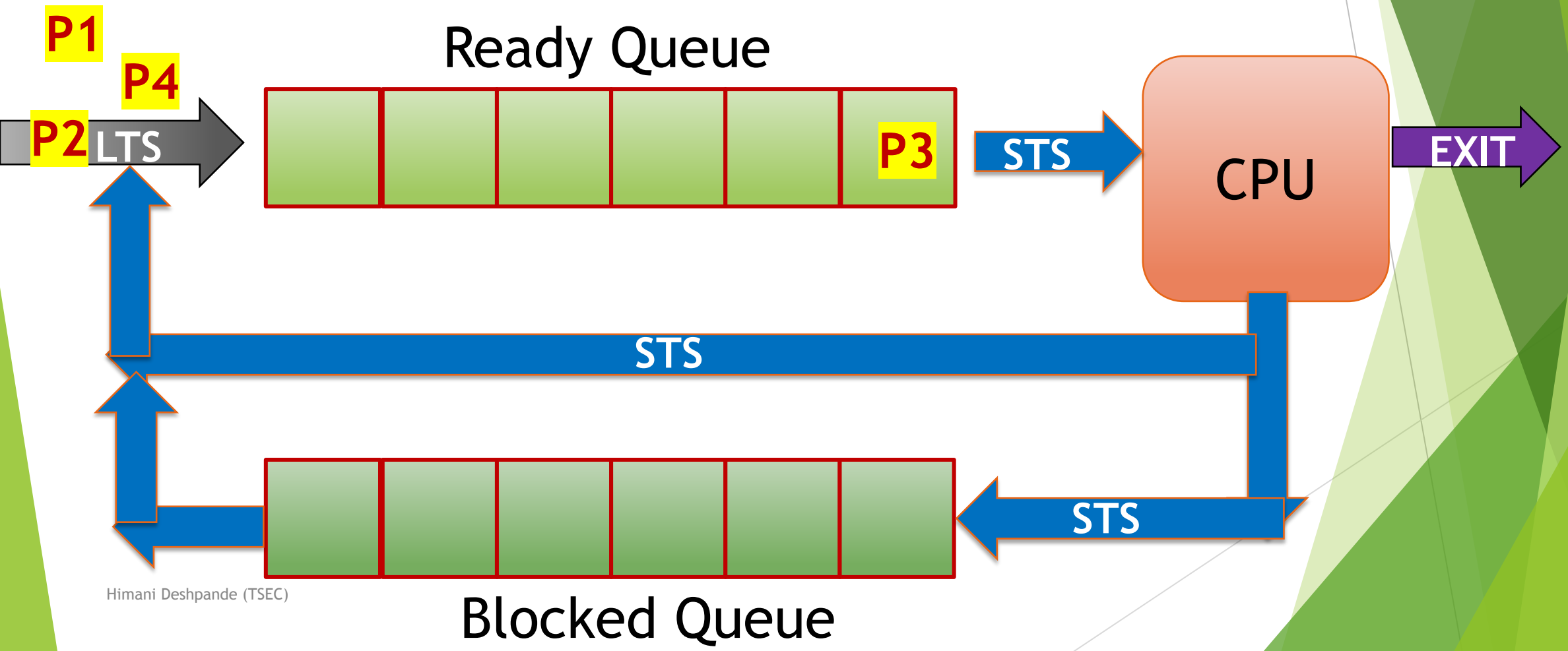
Scheduling



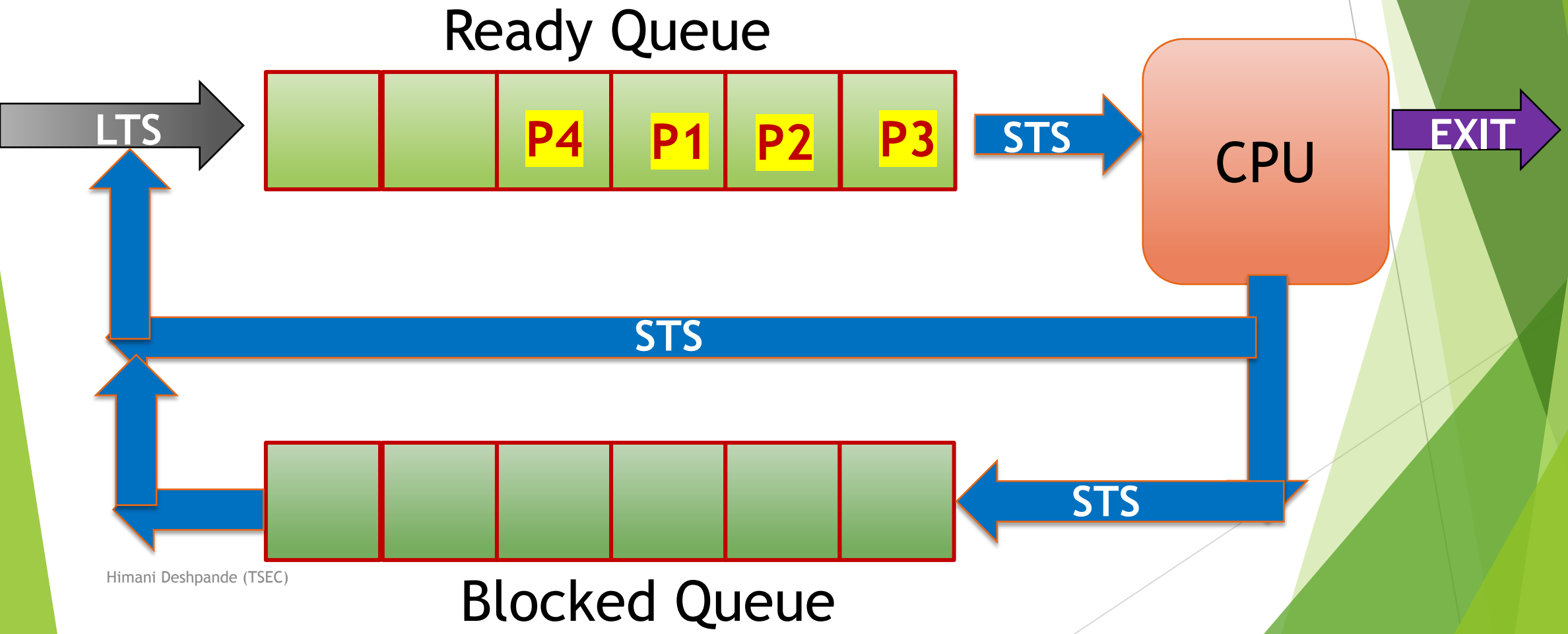
Scheduling



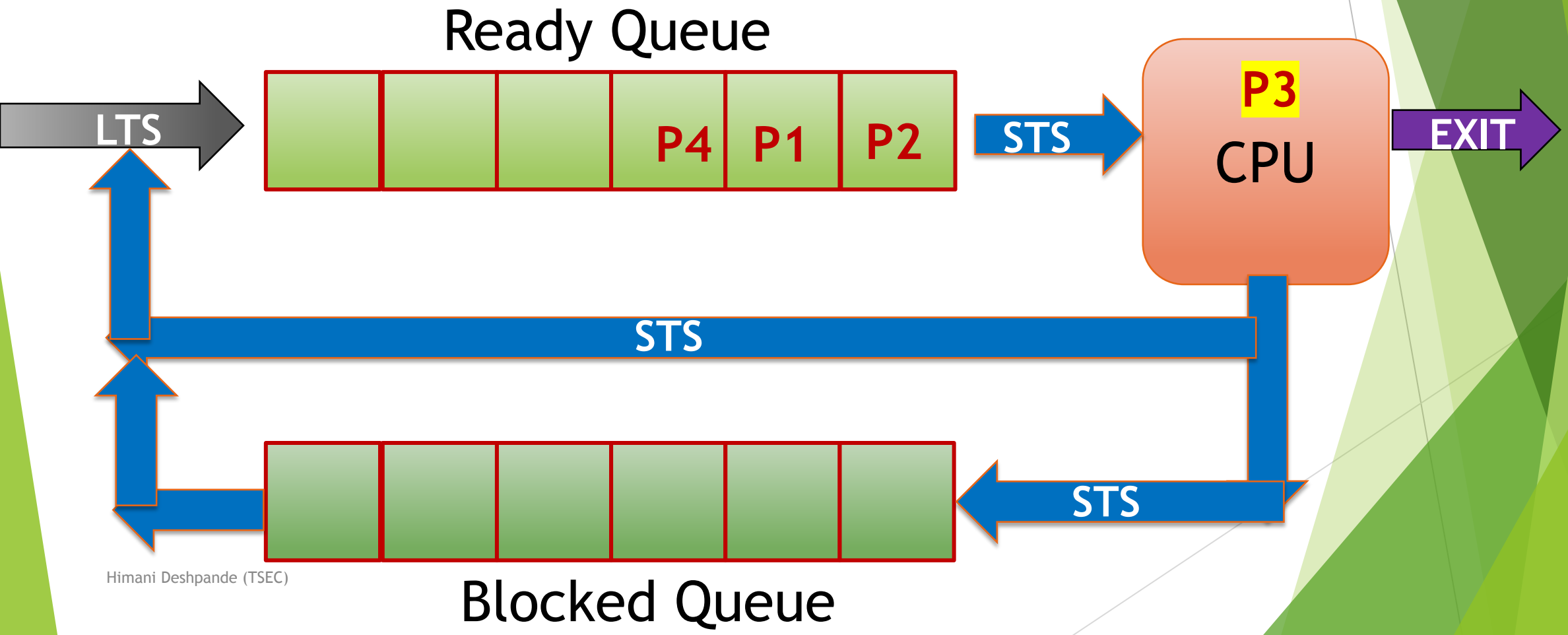
Scheduling



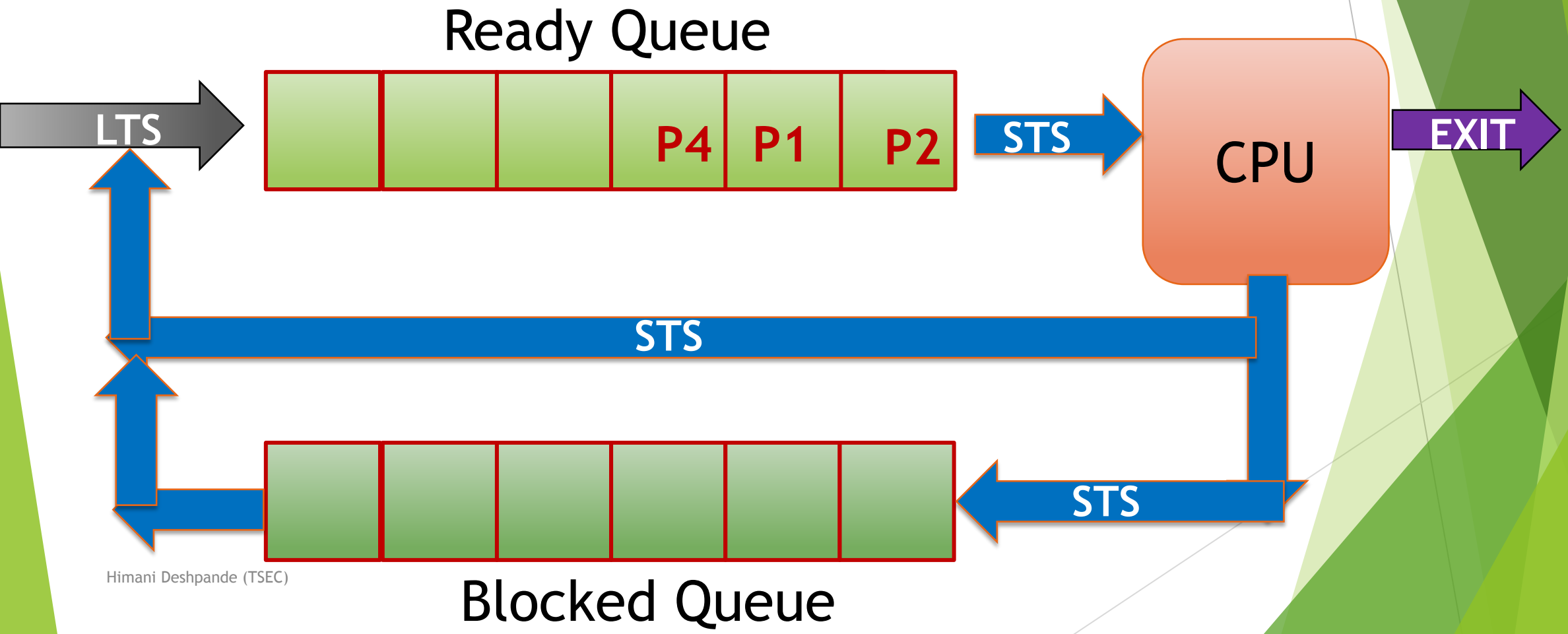
Scheduling



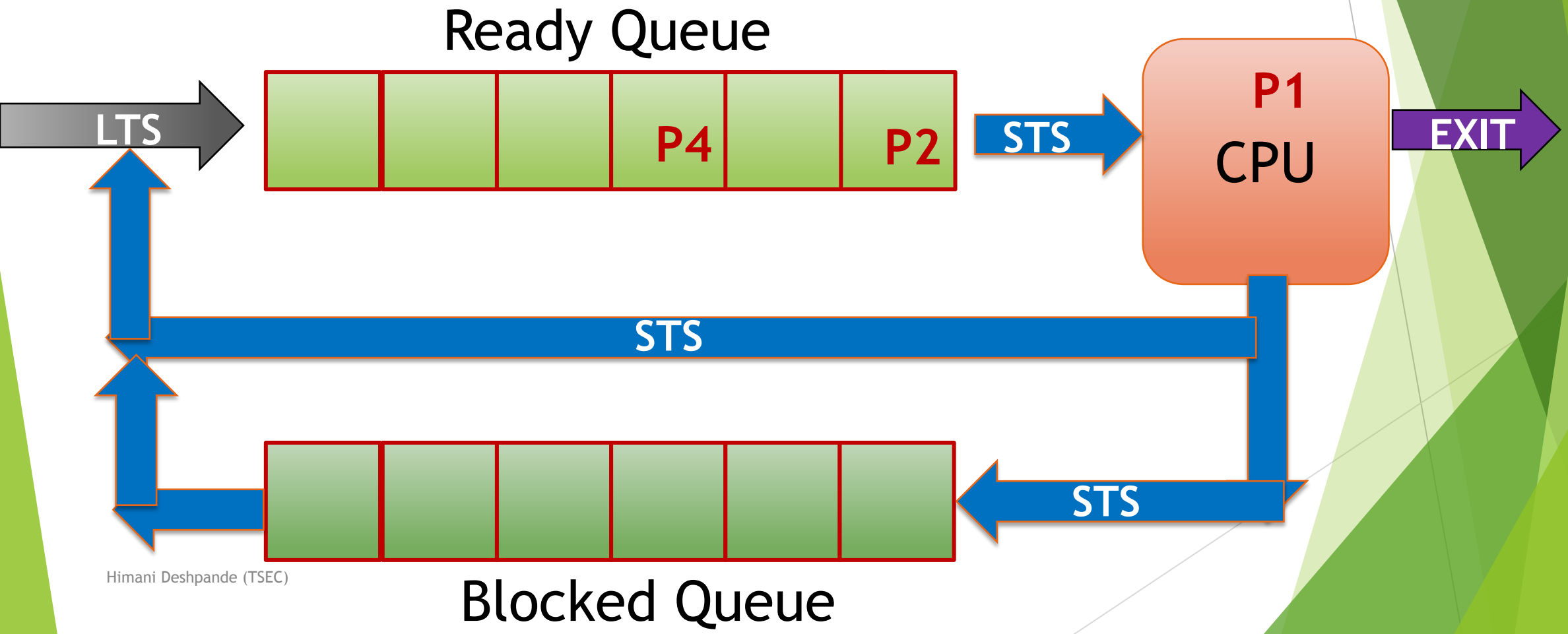
Scheduling



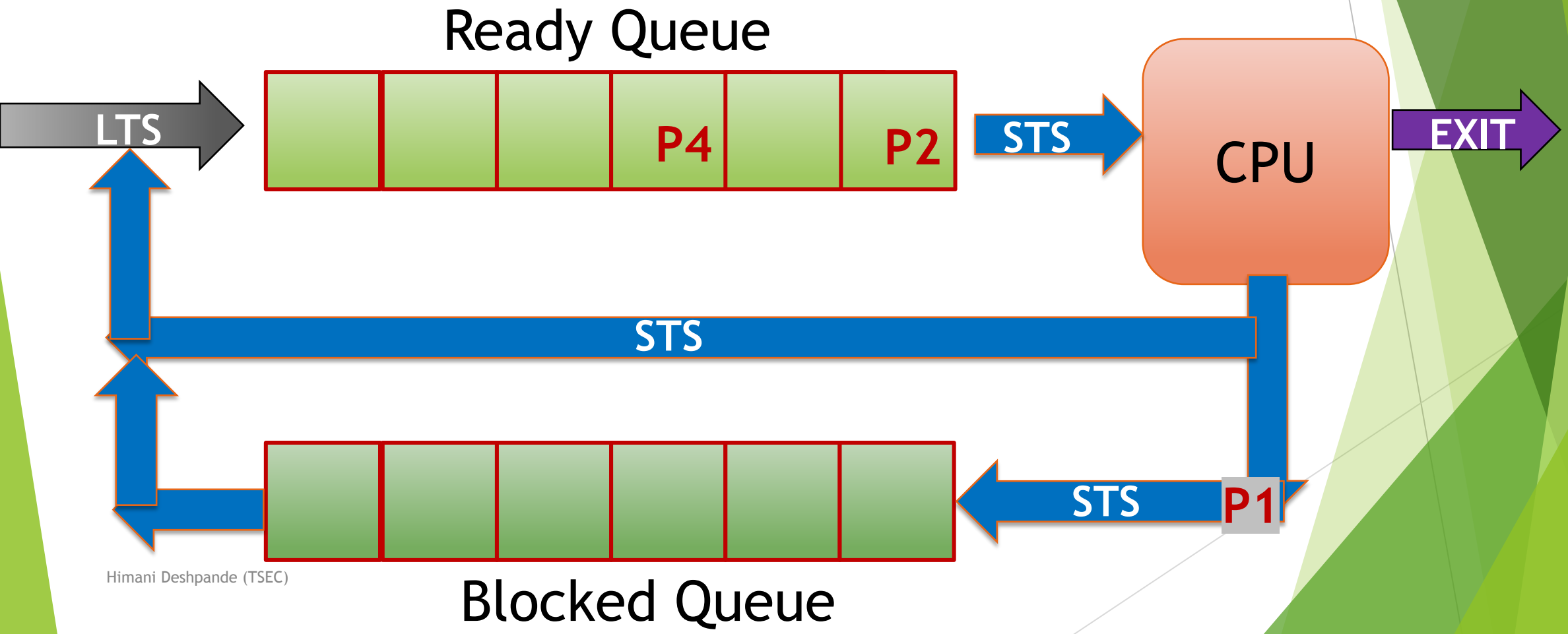
Scheduling



Scheduling



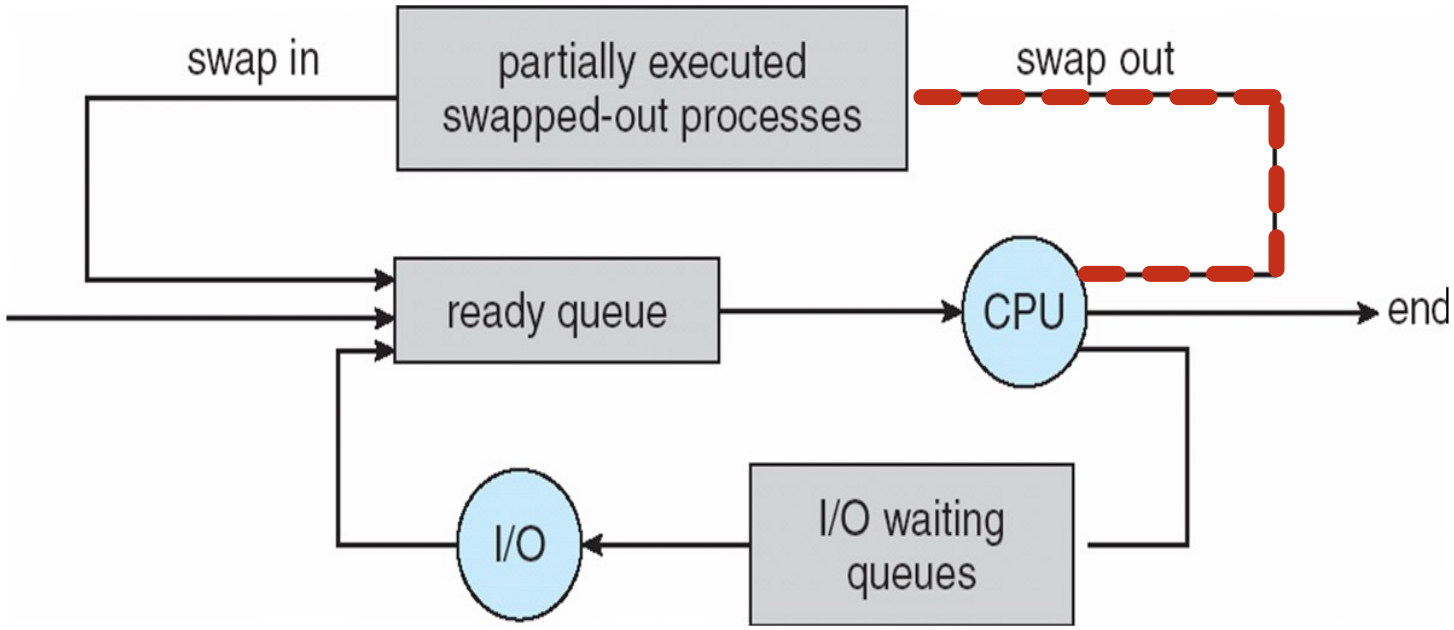
Scheduling



Schedulers

- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)

Medium Term Scheduling



Medium Level Scheduler may move a blocked process temporarily to secondary storage

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

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.

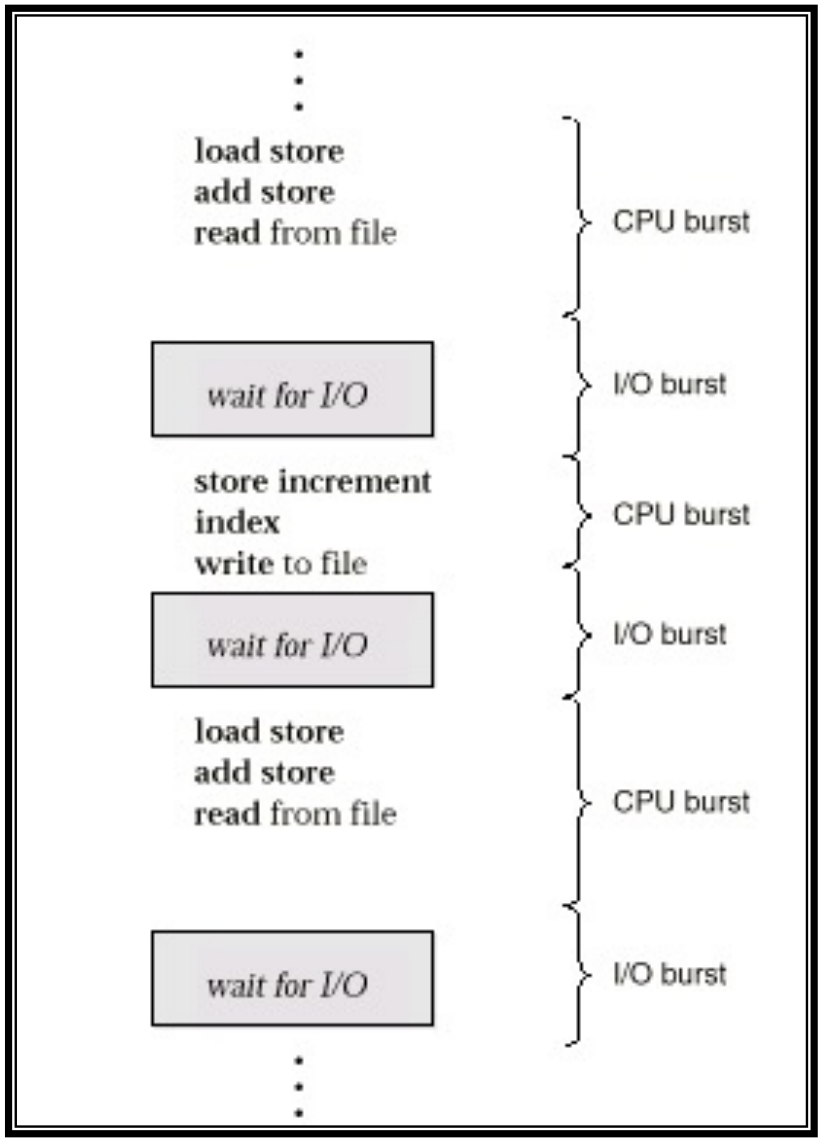
CPU Scheduling

- 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

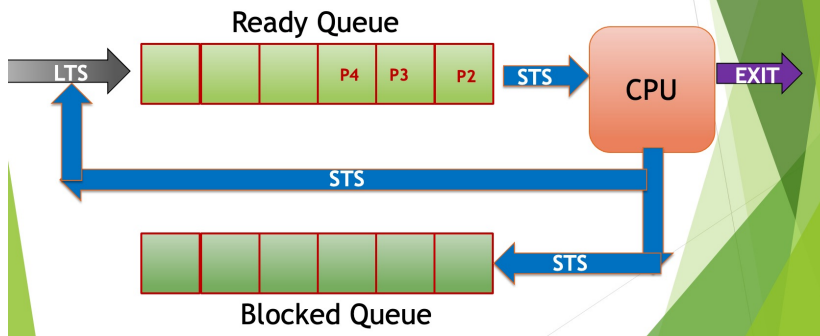
I/O burst is when CPU is **waiting** for I/O for further execution

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

Dispatcher

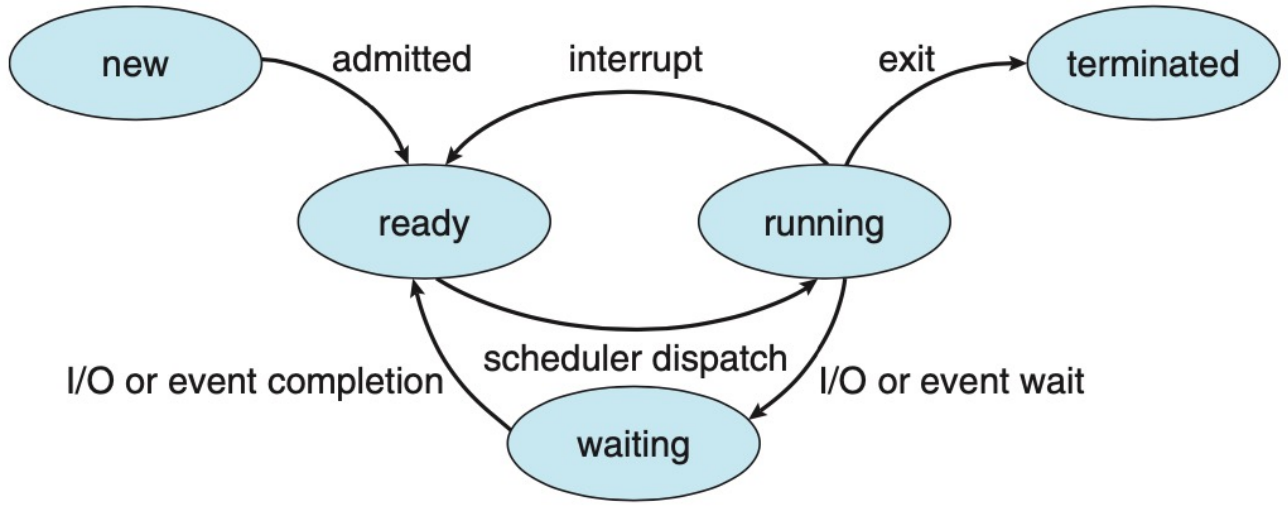
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - jumping to the proper location in the user program to restart that program

- ***Dispatch latency*** –
  - time it takes for the dispatcher to stop one process and start another running.

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 & Non-preemptive

- **Preemptive:**
  - allows a process to be interrupted in the midst of its CPU execution, taking the CPU away to another process

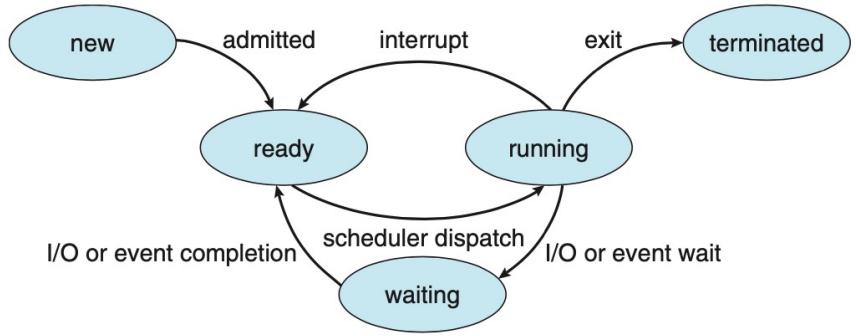
- **Non- Preemptive:**
  - the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state.

CPU Scheduler

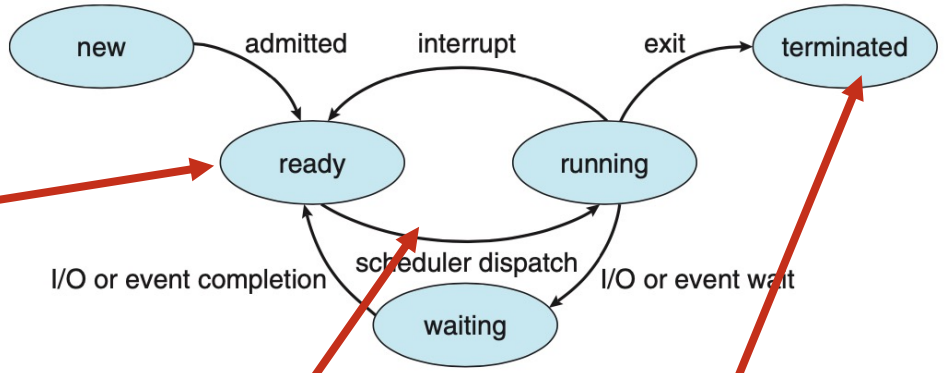
- CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state.
2. Switches from running to ready state.
3. Switches from waiting to ready.
4. Terminates.

Scheduling under 1 and 4 is *non-preemptive*.

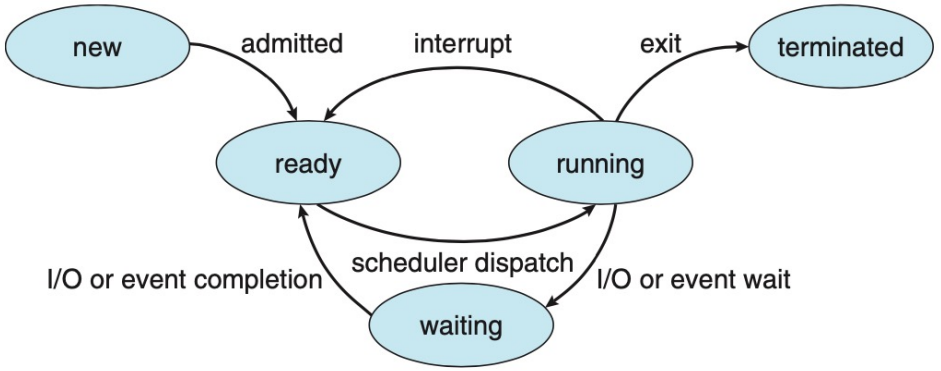


CPU scheduling



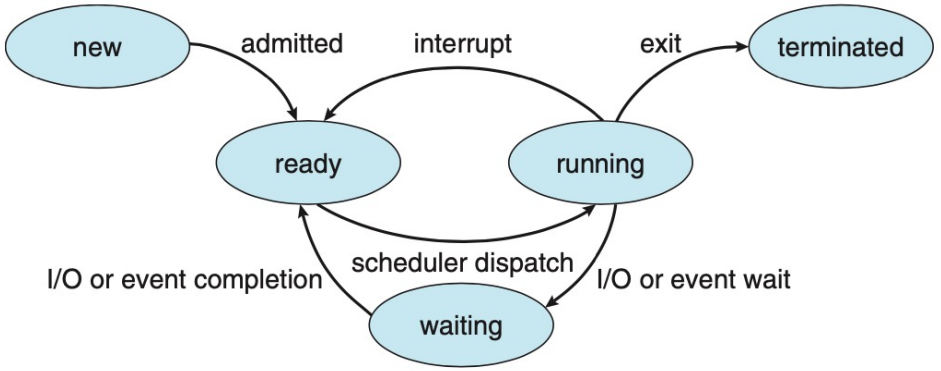
- **Arrival time :**
  - is the time when a process enters into the ready state and is ready for its execution.
- **Exit time**
  - is the time when a process completes its execution and exit from the system.
- **Response time**
  - is the time spent when the process is in the ready state and gets the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

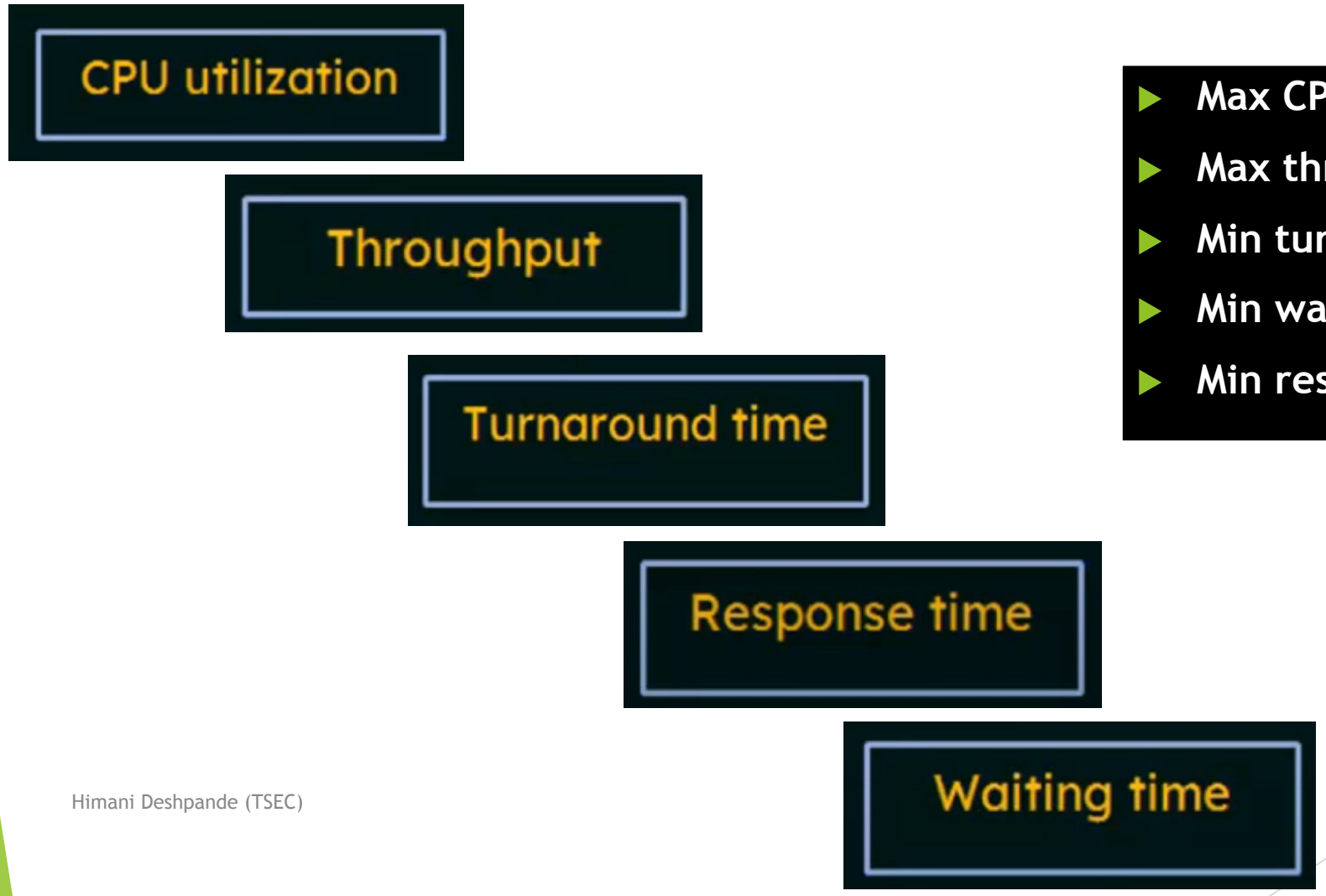
CPU scheduling



- **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



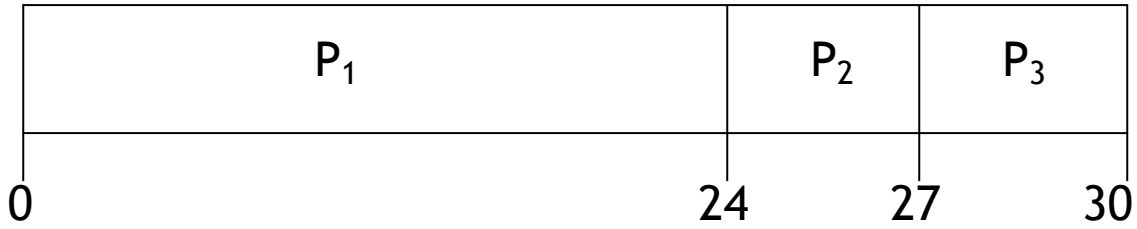
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

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

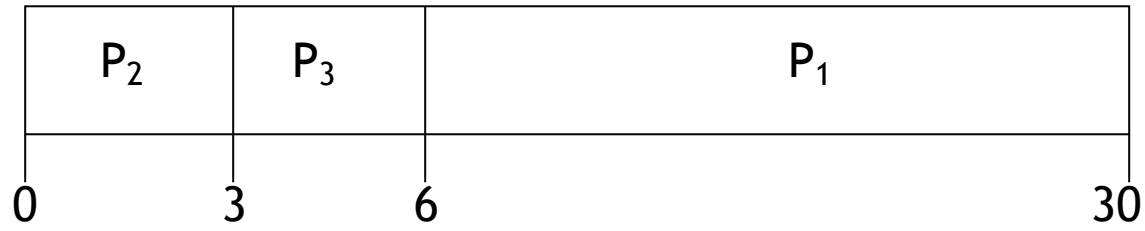
- Average waiting time: (0 + 24 + 27)/3 = 17

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)

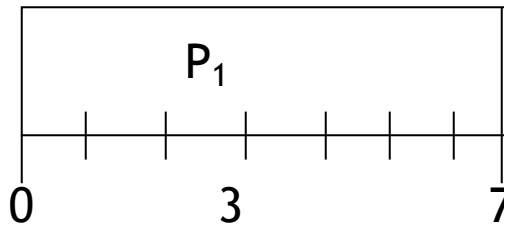
Shortest-Job-First (SJF) Scheduling

- 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

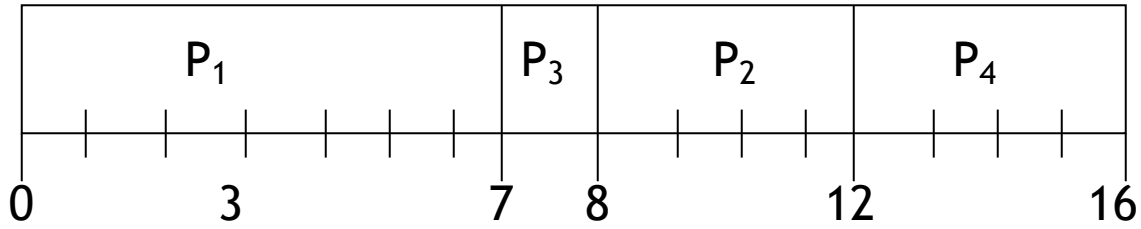
- SJF (non-preemptive)



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

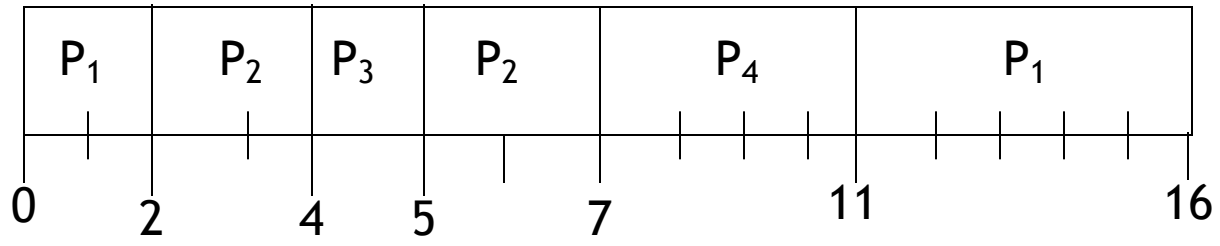
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)

Example of Preemptive SJF

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)

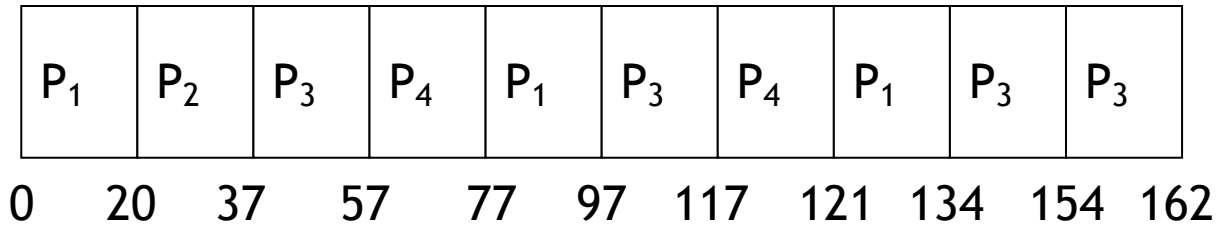
- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds.
- After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the 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.