Textbook:
Operating System Concepts

9th Edition

Abraham Silberschatz
Peter Baer Galvin
Greg Gagne

Process Management

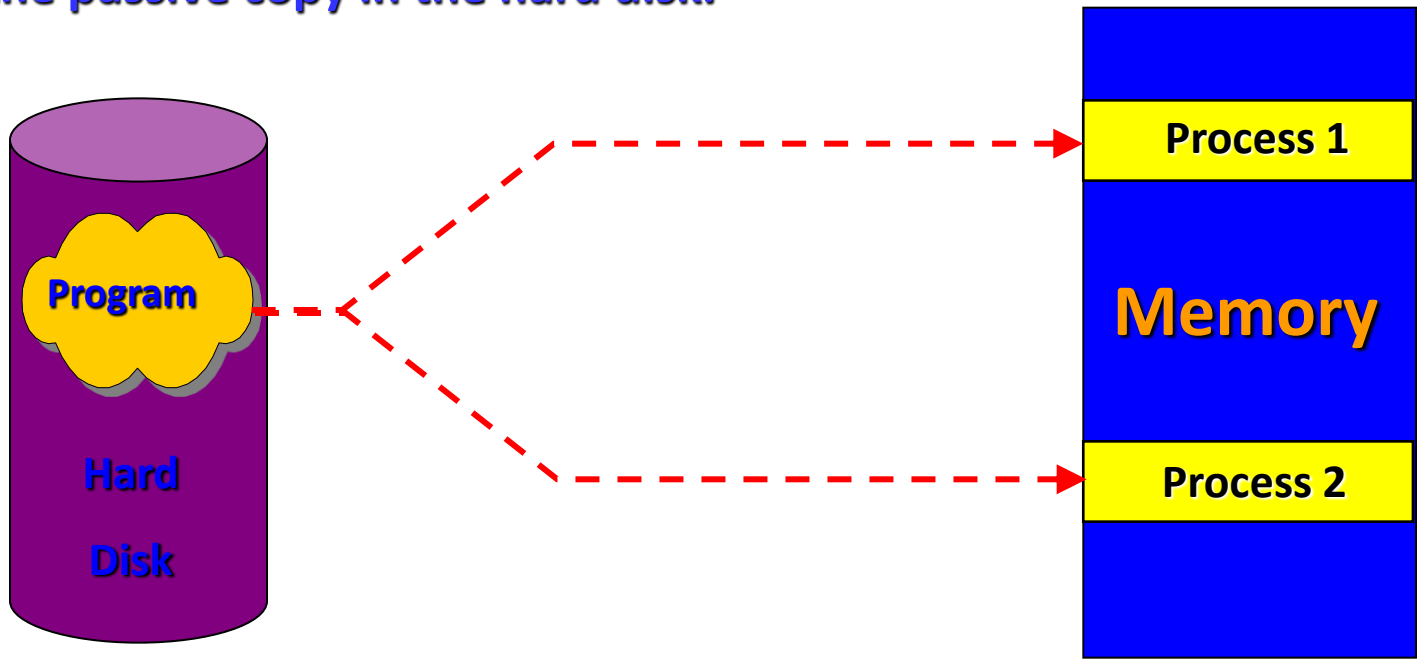
Chapter 3

Q: Define the following:

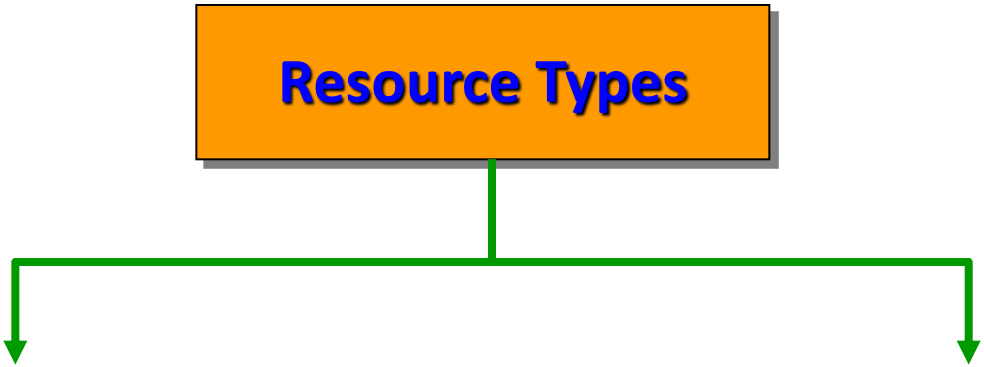
Process – Resource

Sol:

Process :is a program in execution; it is an active entity in the memory while the program is the passive copy in the hard disk.



Resource: any things in the system that may be used by active processes. They may be hardware (printers, tape drives, memory), or software (files, databases).



Preemptive Resources

A resource that stops the current process when a higher priority process comes.

Ex: Memory.

Non Preemptive Resources

A resource that can not stop the current process when a higher priority process comes.

Ex: CD recorder

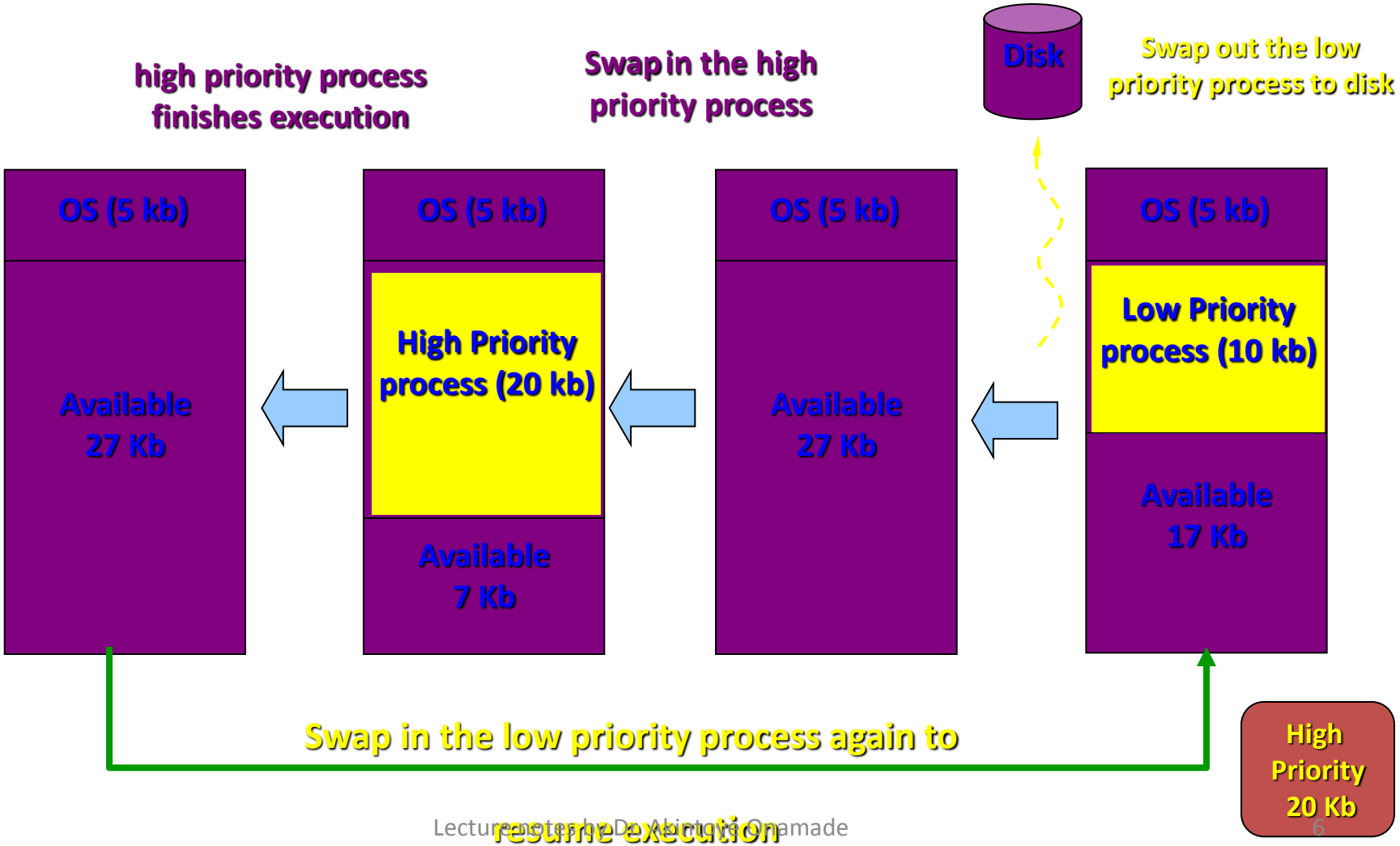
Q: Explain why? Memory is a preemptive resource.

Sol:

Because

- It allows a low priority process to swapped out from it to the disk when a higher priority process arrives and needs a larger amount of memory than the available. The low priority process can resume execution later (swapped in again).

- Assume a system with 32 kb memory size.
- 5kb are used for OS, 10 kb for the low priority process.
- Hence, the available space is 17 kb.
- A higher priority process arrives and needs 20 kb.



Q: Explain why? CD recorder is nonpreemptive resource.

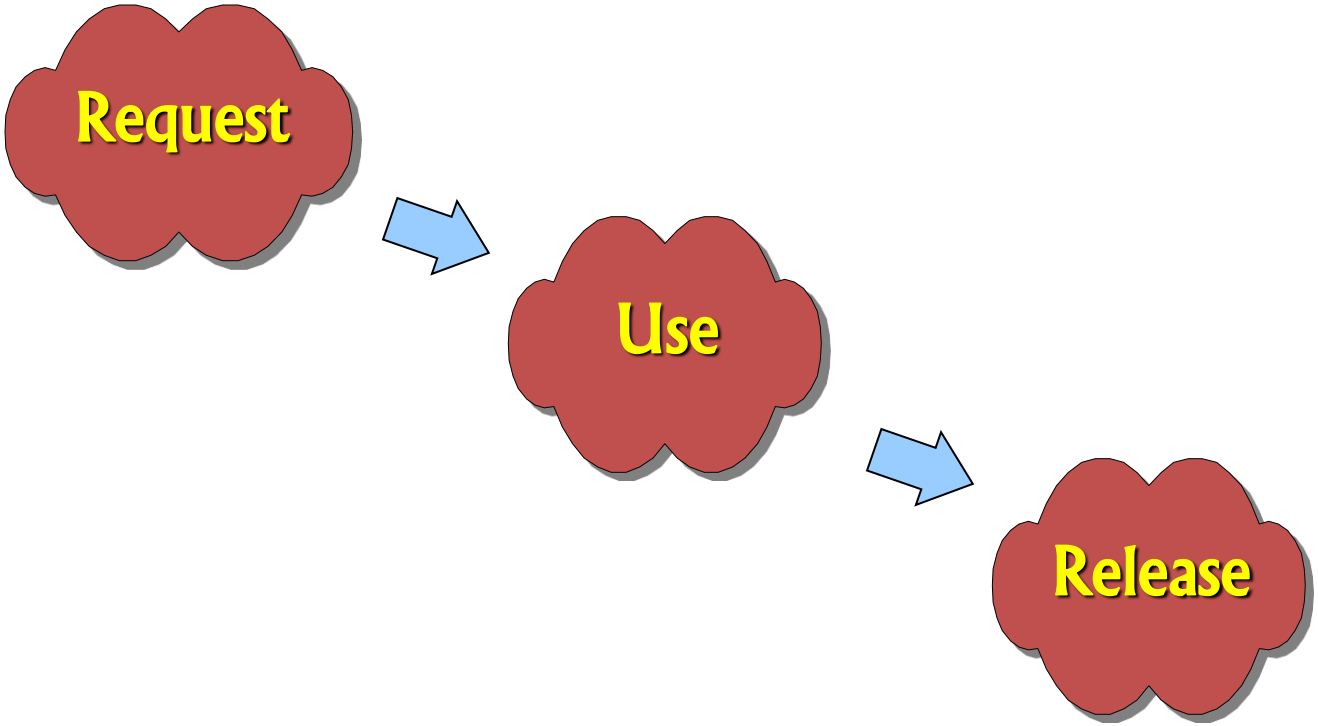
Sol:

Because: If a process has begun to burn a CD-ROM, suddenly taking the CD recorder away from it and giving it to another process will result in a bad CD.

Q: What are the different steps to utilize a resource by a process?

Sol:

A process may utilize a resource in only the following sequence as shown below:

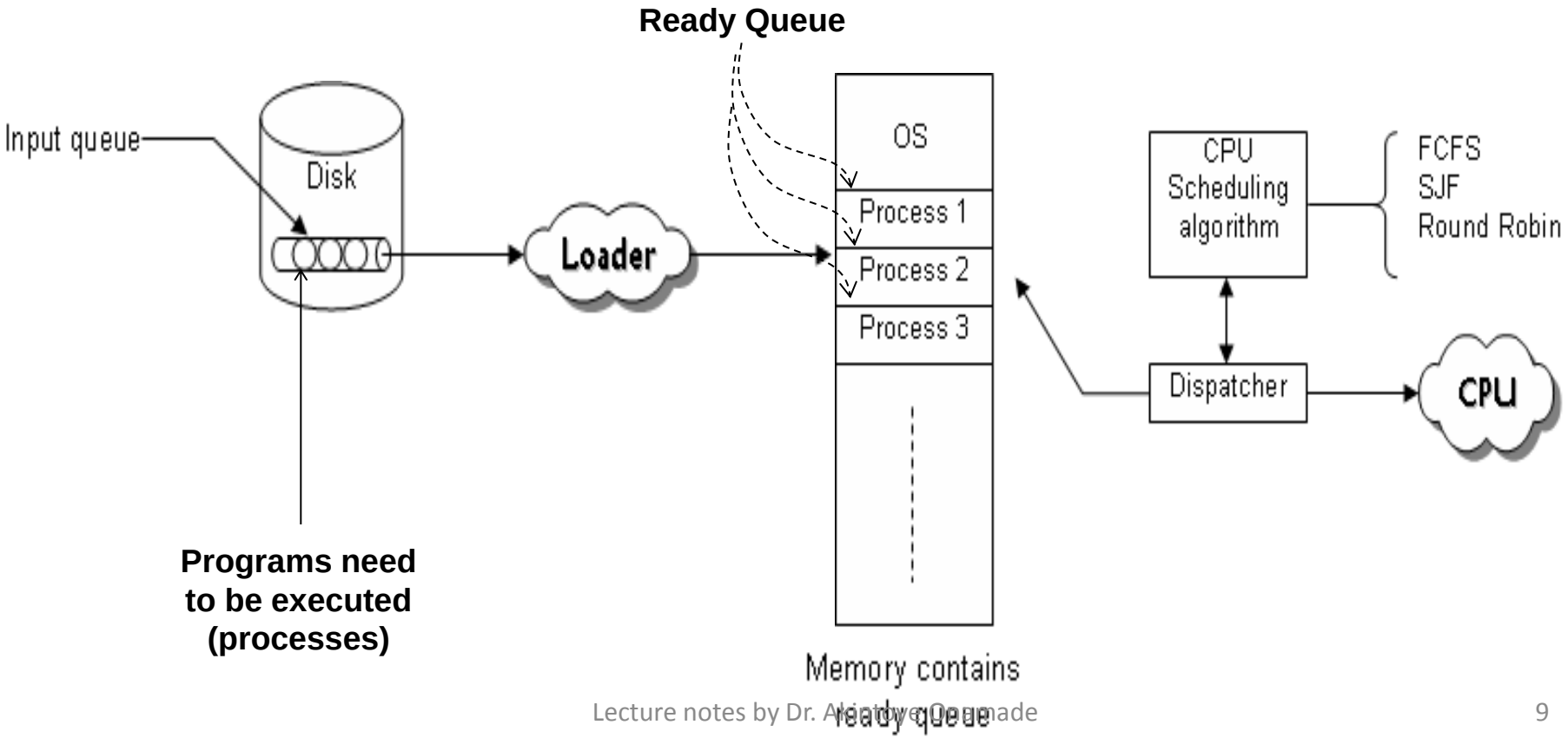


Q: Explain the main differences between a input (job) and ready queues?

Sol:

-input queue: stores the programs that will be opened soon (located at the disk)

- ready queue: contains the active processes that are ready to be executed (located in the memory).



Q: Explain process states and state transitions and then draw the process state diagram.

Sol:

As process executes, it changes its state. The process state may be:

- New: **The process is being created.**
- Ready: **The process is waiting for the processor.**
- Running: **The process instructions are being executed.**
- Waiting/Blocked: **The process is waiting for an event to happen (such as I/O completion).**
- Terminated: **The process has finished execution.**

Process States and State Transitions 1

- When a user runs a program, processes are created and inserted into the ready list.
- A process moves toward the head of the list as other processes terminate or complete their turns using a processor.
- When a processor becomes available, that process is given a processor and is said to make a ***state transition*** from the ***ready state*** to ***running state***.
- The act of assigning a processor to the first process on the ready list is called ***dispatching*** and is performed by a system entity called the ***dispatcher***.
- Processes that are in the ready or running states are said to be **awake**.

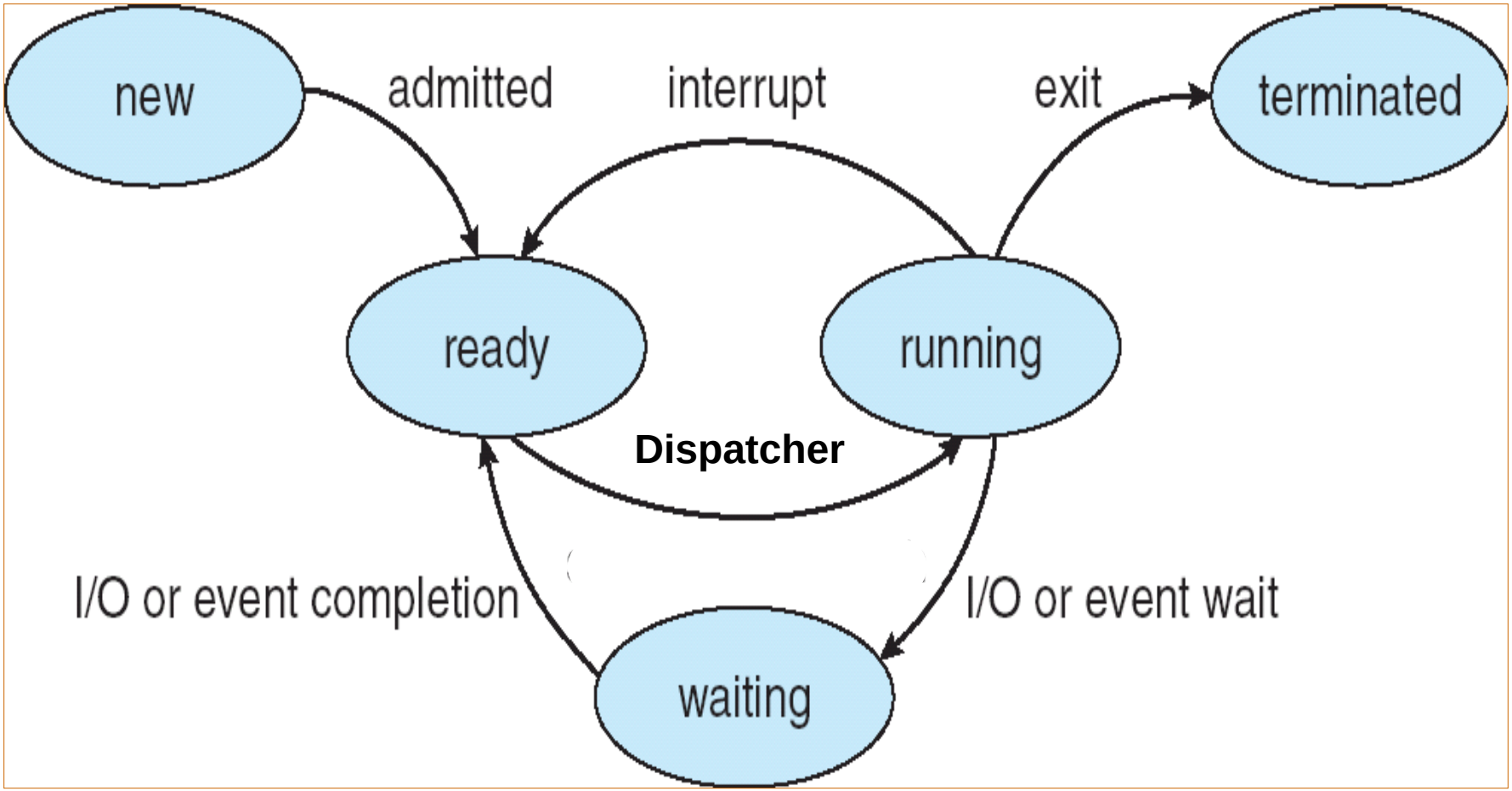
Process States and State Transitions 2

- To prevent any one process from monopolizing the system, the OS sets a hardware ***interrupting clock/interval timer,*** to allow a process to run for a specific time interval or ***quantum***.
- If the process does not voluntarily yield the processor before the time interval expires, the interrupting clock generates an interrupt, causing the OS to gain control of the processor.
- The OS then changes the state of the previously running process to ready and dispatches the first process on the ready list, changing its state from ready to running.
- If a running process initiates an input/output operation before its quantum expires, and therefore must wait for the I/O operation to complete before it can use a processor again, then the running process voluntarily relinquishes the processor. In this case, the process is said to ***block*** itself, pending the completion of the I/O operation.

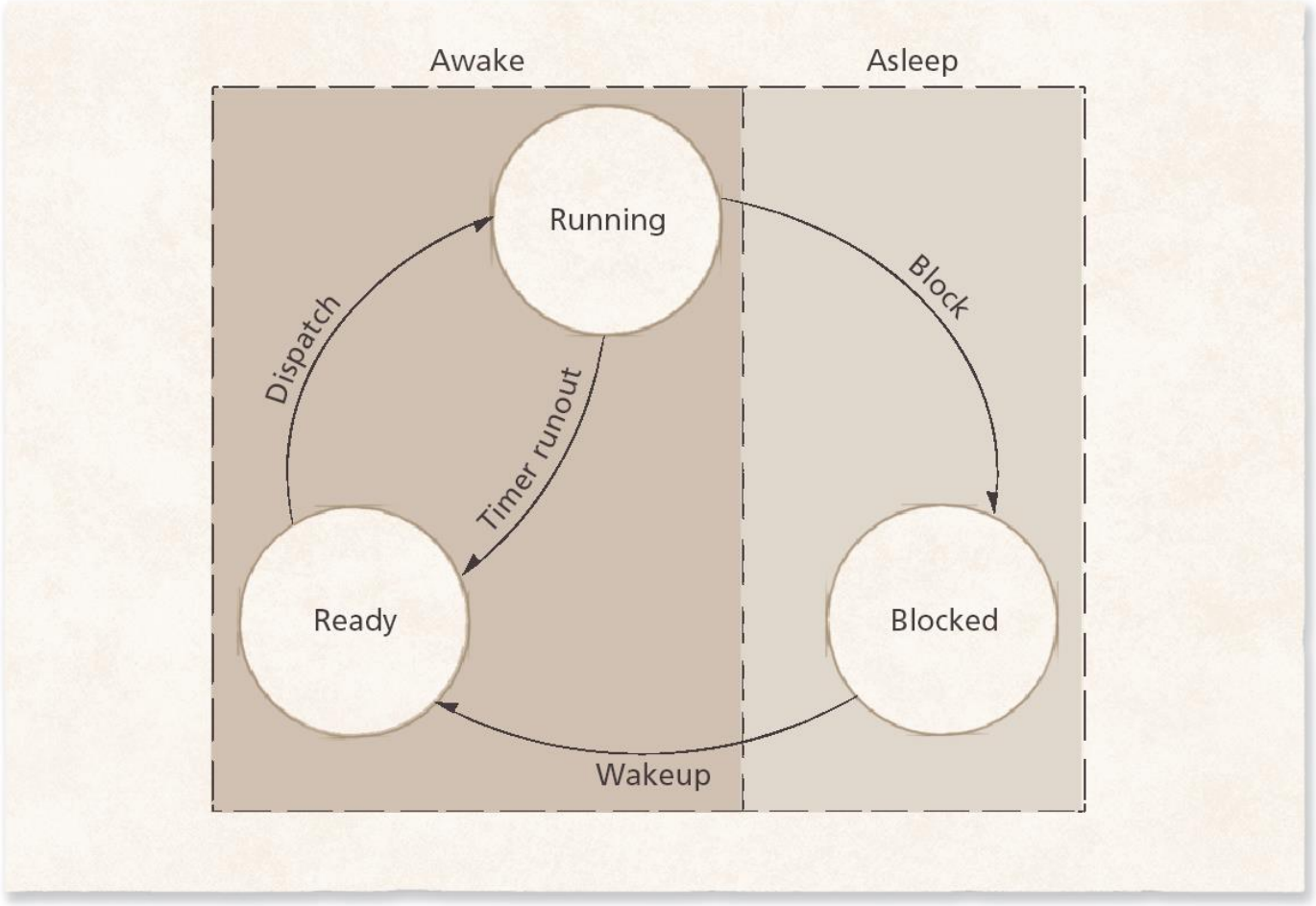
Process States and State Transitions 3

- Processes in the blocked state are said to be **asleep**, because they cannot execute even if a processor becomes available.
- If I/O operation or some other event the process is waiting for completes, the OS changes the process from the blocked to the ready state.

The process state diagram is:



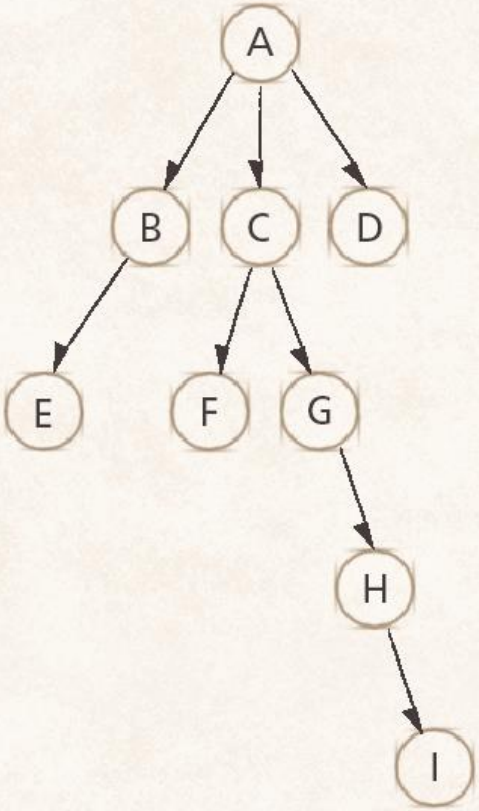
The process state diagram with Awake and Asleep transitions



Process Operations

- A process may spawn a new process
  - The creating process is called the parent process
  - The created process is called the child process
  - Exactly one parent process creates a child
  - When a parent process is destroyed, operating systems typically respond in one of two ways:
    - Destroy all child processes of that parent
    - Allow child processes to proceed independently of their parents

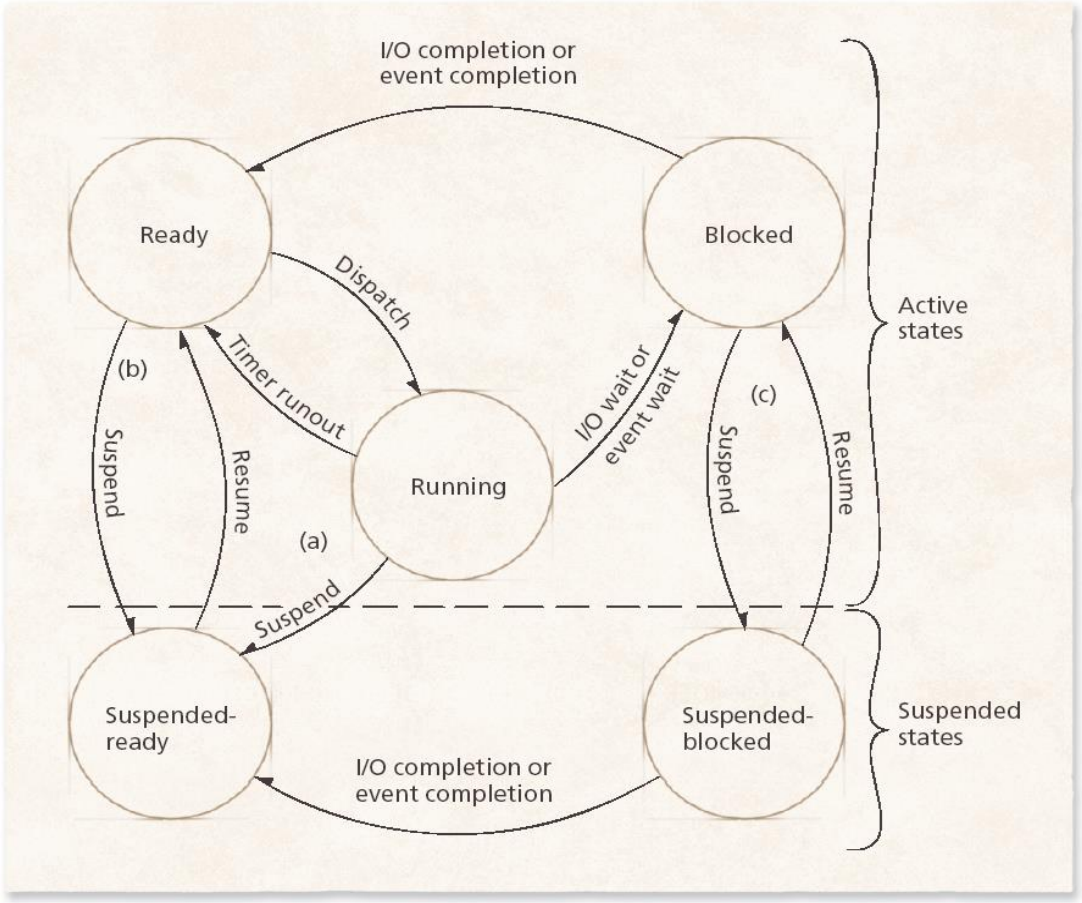
Process Operations



Suspend and Resume

- Suspending a process
  - Indefinitely removes it from contention for time on a processor without being destroyed
  - Useful for detecting security threats and for software debugging purposes
  - A suspension may be initiated by the process being suspended or by another process
  - A suspended process must be resumed by another process
  - Two suspended states:
    - *suspendedready*
    - *suspendedblocked*

Suspend and Resume



What are concurrent processes?

- Concurrent processing is a computing model in which multiple processors execute instructions simultaneously for better performance.

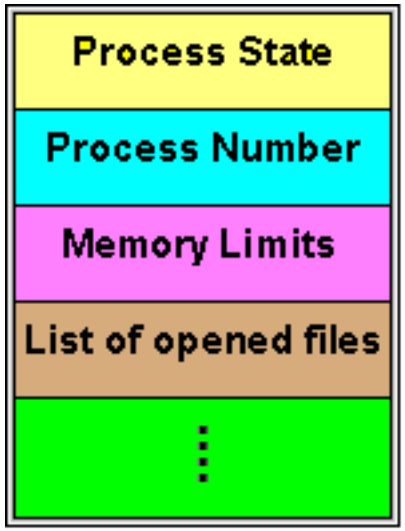
What is asynchronous concurrent execution?

- Processes that operate independently of one another but must occasionally communicate and synchronize to perform cooperative tasks are said to execute asynchronously.

What is Process Control Block (PCB) /Process Descriptors?

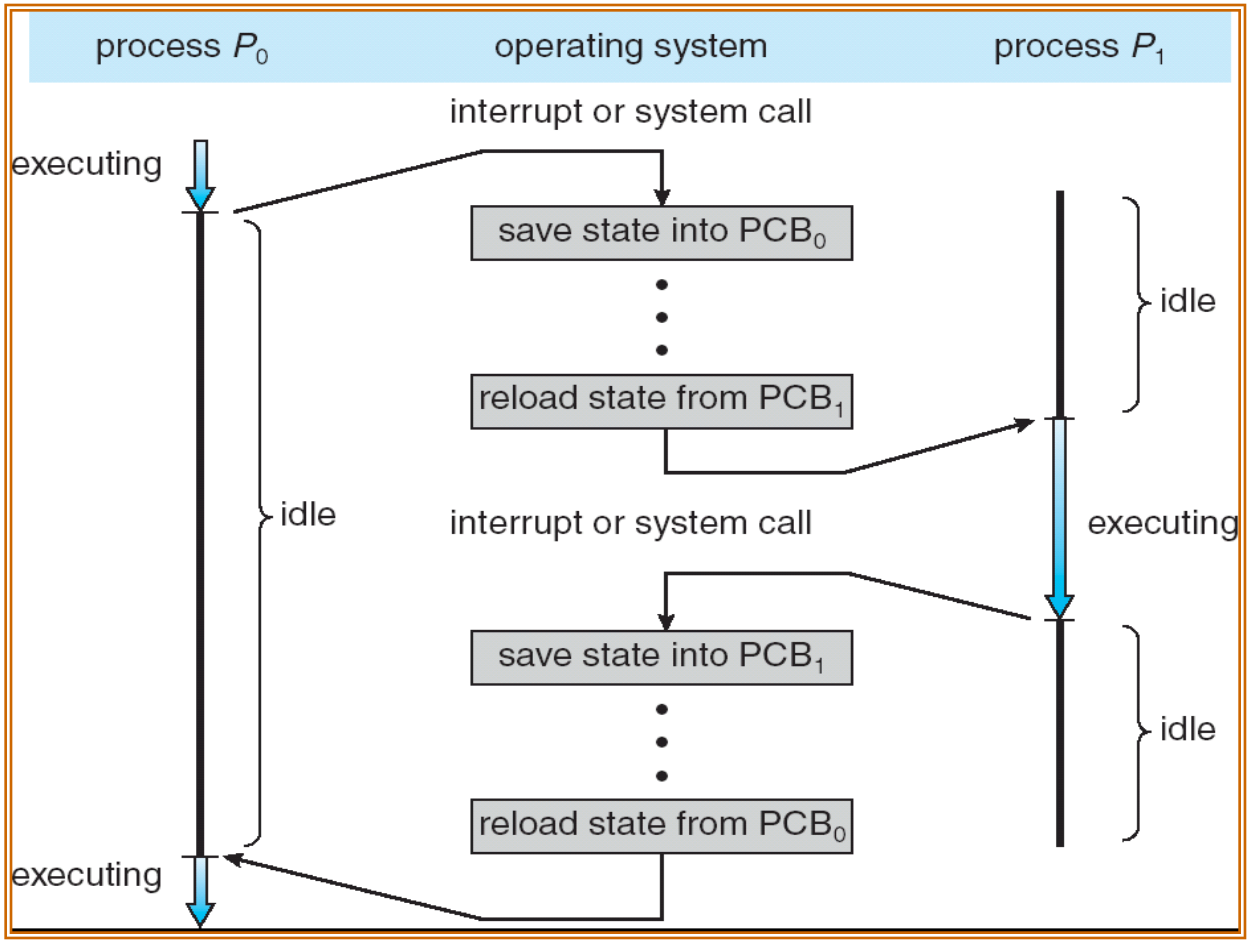
- Process control block (PCB) is the way used to represent the process in the operating system.
- PCB maintains information that the OS needs to manage the process.

- **It contains information about the process such as:**
  - **Process state** (new, ready, running or blocked)
  - **Process Identification number (PID)**
  - **Address of the next instruction** to be executed in the process.
  - **Process priority.**
  - **Process assigned memory.**
  - **I/O information** (such as I/O devices and opened files).



Q: Show graphically how OS uses the PCB to switch between active processes

Sol:



- **When interrupt happened, OS saves the state of the current active process to its PCB so it can continue correctly when it resumes its execution again.**

- **To resume execution, OS reloads the process state from its PCB and continue execution.**

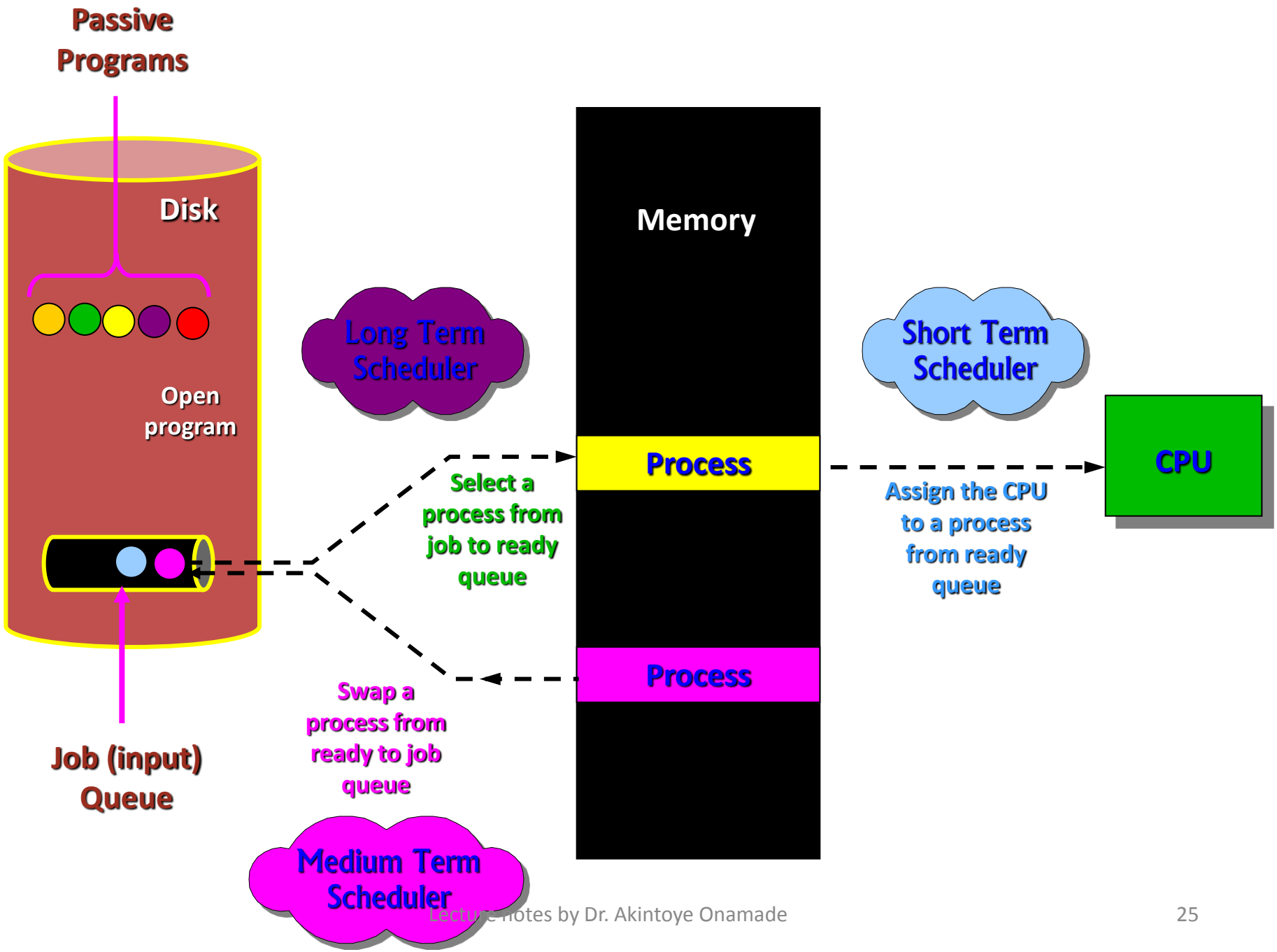
Q: what is meant by Schedulers?, then discuss different types of them?

Sol:

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue from the job queue (determine the degree of multi-programming)

- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU

- Medium-term scheduler: swap out the process from memory (ready queue) and swapped in again later (it decrease the degree of multiprogramming).

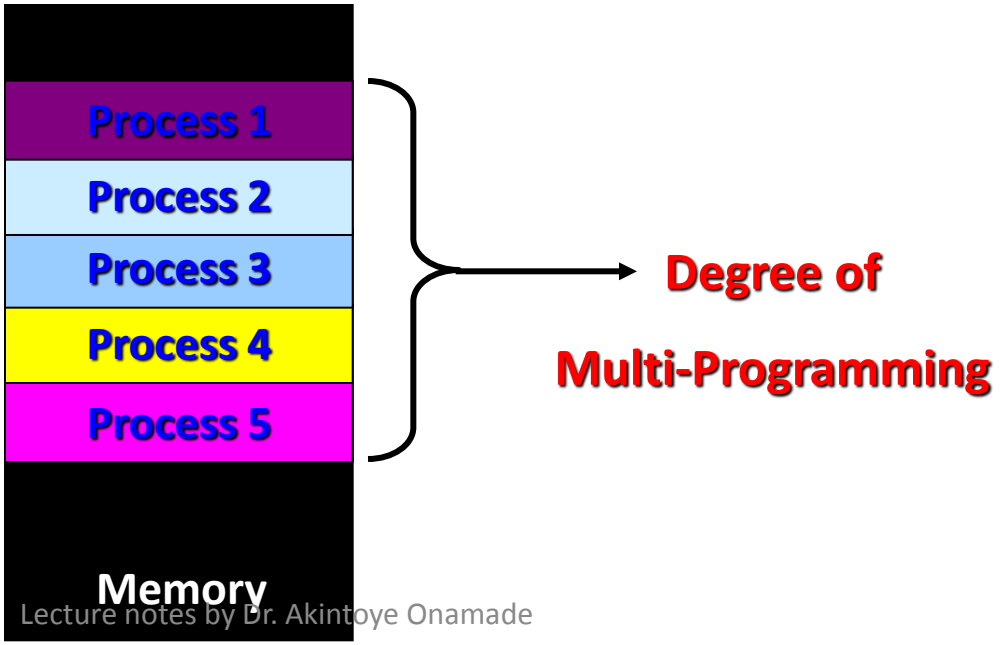

Passive Programs
Disk
Open program
Job (input) Queue
Long Term Scheduler
Memory
Process
Process
Select a process from job to ready queue
Swap a process from ready to job queue
Short Term Scheduler
Assign the CPU to a process from ready queue
CPU
Medium Term Scheduler

Q: Explain why?

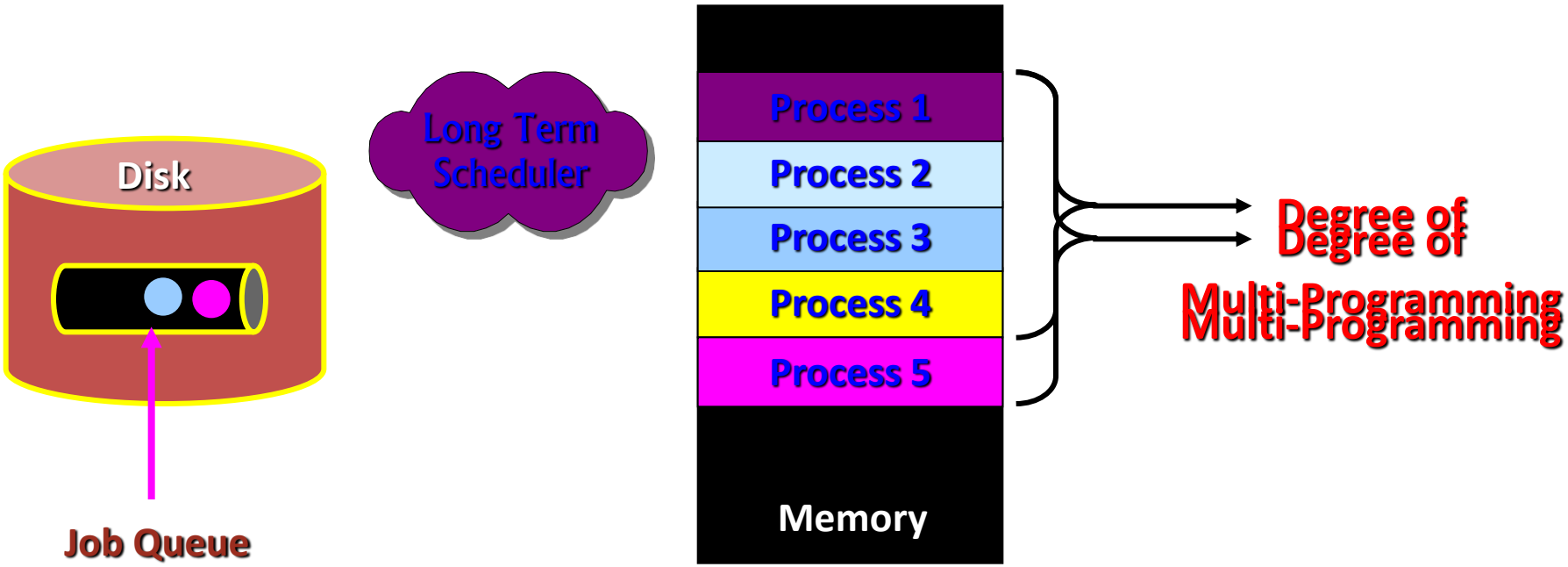
- Long term scheduler increases the degree of multiprogramming.
- Medium term scheduler decreases the degree of multiprogramming.

Sol:

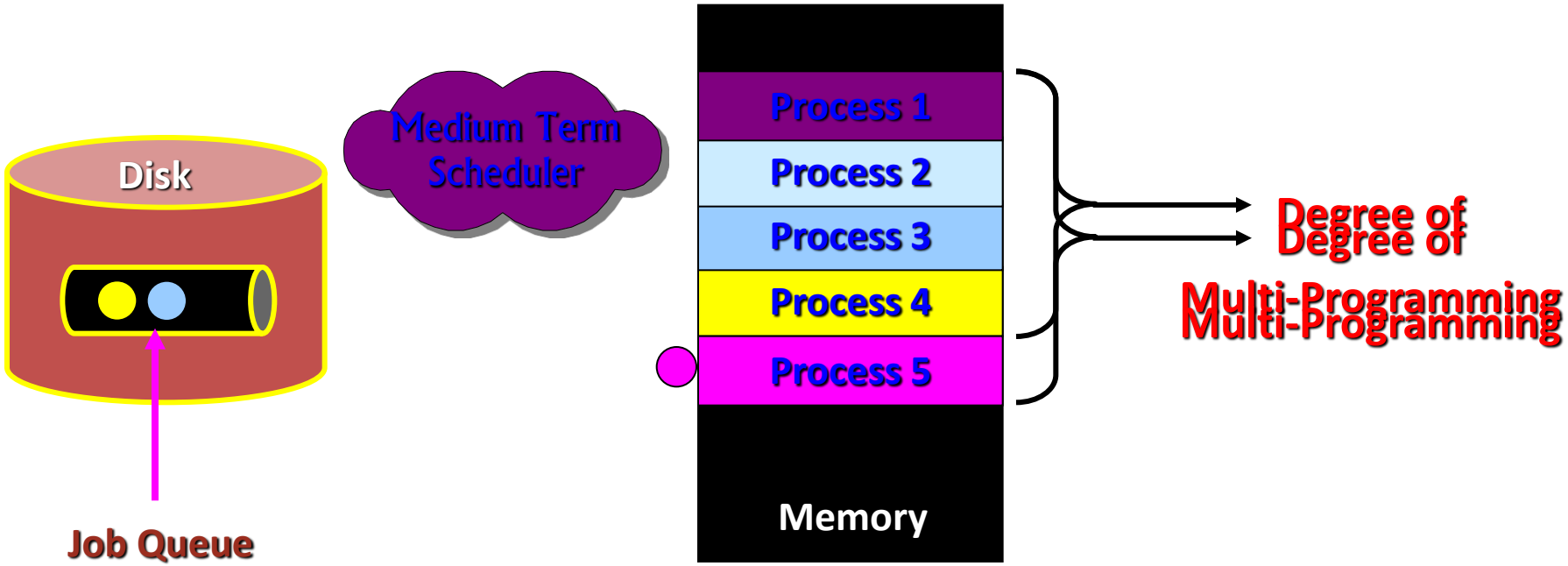
- **Degree of multi-programming is the number of processes that are placed in the ready queue waiting for execution by the CPU.**



- Since Long term scheduler selects which processes to be brought to the ready queue, hence, it increases the degree of multiprogramming.



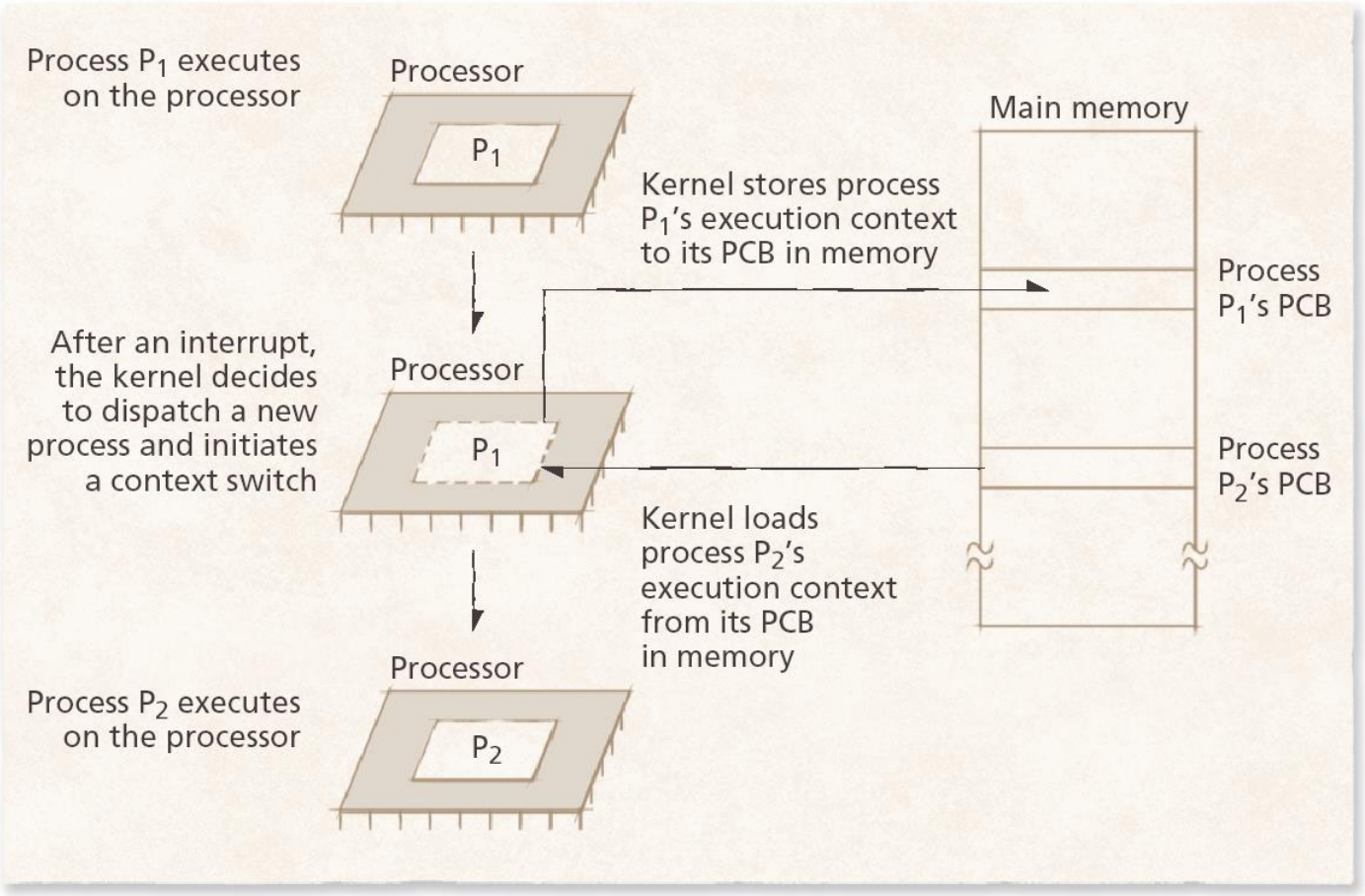
Since Medium term scheduler picks some processes from the ready queue and swap them out of memory, hence, it decreases the degree of multiprogramming.



Context Switching

- Context switches
  - Performed by the OS to stop executing a *running* process and begin executing a previously *ready* process
  - Save the execution context of the *running* process to its PCB
  - Load the *ready* process's execution context from its PCB
  - Must be transparent to processes
  - Require the processor to not perform any "useful" computation
    - OS must therefore minimize context-switching time
  - Performed in hardware by some architectures

Context Switching



Q: Give a suitable definition for the "Context Switch", and then explain why context switch is a pure overhead?

Sol:

- **Context switch** is the time needed by OS to switch the processor from a process to another.

- **Context switch($P_x \rightarrow P_y$)** = $T_x + T_y$, where

  T_x : time needed to save the state of P_x in PCB_x

  T_y : time needed to load the state of P_y in PCB_y

- **Context switch** is a pure overhead because the system does no useful work while switching.

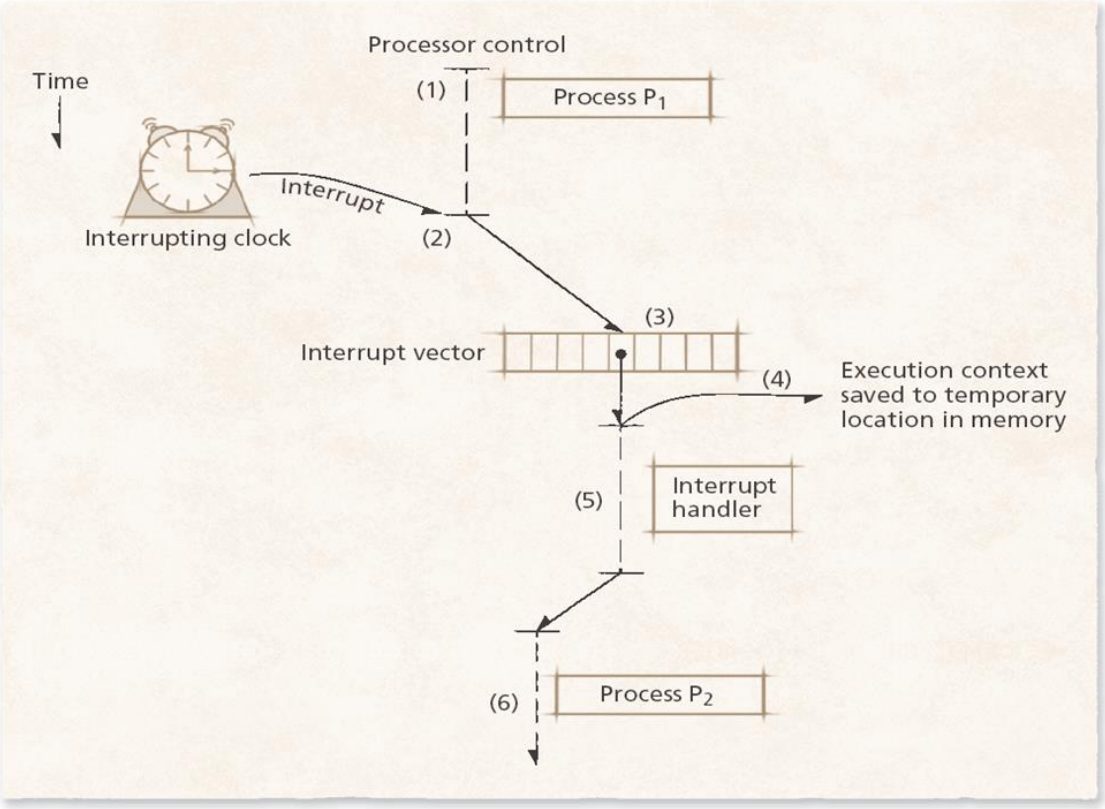
Interrupts

- Interrupts enable software to respond to signals from hardware
  - May be initiated by a running process
    - Interrupt is called a trap
    - Synchronous with the operation of the process
    - For example, dividing by zero or referencing protected memory
  - May be initiated by some event that may or may not be related to the running process
    - Asynchronous with the operation of the process
    - For example, a key is pressed on a keyboard or a mouse is moved
  - Low overhead
- Polling is an alternative approach
  - Processor repeatedly requests the status of each device
  - Increases in overhead as the complexity of the system increases

Interrupt Processing

- Handling interrupts
  - After receiving an interrupt, the processor completes execution of the current instruction, then pauses the current process
  - The processor will then execute one of the kernel's interrupt-handling functions
  - The interrupt handler determines how the system should respond
  - Interrupt handlers are stored in an array of pointers called the interrupt vector
  - After the interrupt handler completes, the interrupted process is restored and executed or the next process is executed

Interrupt Processing



Interrupt Classes

- Supported interrupts depend on a system's architecture
  - The IA-32 specification distinguishes between two types of signals a processor may receive:
    - Interrupts
      - Notify the processor that an event has occurred or that an external device's status has changed
      - Generated by devices external to a processor
    - Exceptions
      - Indicate that an error has occurred, either in hardware or as a result of a software instruction
      - Classified as faults, traps or aborts

Interrupt Classes

Common interrupt types recognized in the Intel IA-32 architecture.

Interrupt Type	Description of Interrupts in Each Type
I/O	These are initiated by the input/output hardware. They notify a processor that the status of a channel or device has changed. I/O interrupts are caused when an I/O operation completes, for example.
Timer	A system may contain devices that generate interrupts periodically. These interrupts can be used for tasks such as timekeeping and performance monitoring. Timers also enable the operating system to determine if a process's quantum has expired.
Interprocessor interrupts	These interrupts allow one processor to send a message to another in a multiprocessor system.

Interrupt Classes

Intel IA-32 exception classes.

Exception Class	Description of Exceptions in Each Class
Fault	These are caused by a wide range of problems that may occur as a program's machine-language instructions are executed. These problems include division by zero, data (being operated upon) in the wrong format, attempt to execute an invalid operation code, attempt to reference a memory location beyond the limits of real memory, attempt by a user process to execute a privileged instruction and attempt to reference a protected resource.
Trap	These are generated by exceptions such as overflow (when the value stored by a register exceeds the capacity of the register) and when program control reaches a breakpoint in code.
Abort	This occurs when the processor detects an error from which a process cannot recover. For example, when an exception-handling routine itself causes an exception, the processor may not be able to handle both errors sequentially. This is called a double-fault exception, which terminates the process that initiated it.

Signals

- Software interrupts that notify a process that an event has occurred
  - Do not allow processes to specify data to exchange with other processes
  - Processes may catch, ignore or mask a signal
    - Catching a signal involves specifying a routine that the OS calls when it delivers the signal
    - Ignoring a signal relies on the operating system's default action to handle the signal
    - Masking a signal instructs the OS to not deliver signals of that type until the process clears the signal mask

Message Passing

- Message-based interprocess communication
  - Messages can be passed in one direction at a time
    - One process is the sender and the other is the receiver
  - Message passing can be bidirectional
    - Each process can act as either a sender or a receiver
  - Messages can be blocking or nonblocking
    - Blocking requires the receiver to notify the sender when the message is received
    - Nonblocking enables the sender to continue with other processing
  - Popular implementation is a pipe
    - A region of memory protected by the OS that serves as a buffer, allowing two or more processes to exchange data

Message Passing

- IPC in distributed systems
  - Transmitted messages can be flawed or lost
    - Acknowledgement protocols confirm that transmissions have been properly received
    - Timeout mechanisms retransmit messages if acknowledgements are not received
  - Ambiguously named processes lead to incorrect message referencing
    - Messages are passed between computers using numbered ports on which processes listen, avoiding this problem
  - Security is a significant problem
    - Ensuring authentication

Reference

- Operating System Concepts 9th Edition by Abraham Silberschatz, Peter Baer Galvin and Greg Gagne
- Eng. Mohamed Fathy : Slides on “Introduction to Operating System” (which formed the major components of this lecture notes)
- Slides on “Process Concepts” from Department of Computer Science, Babcock University