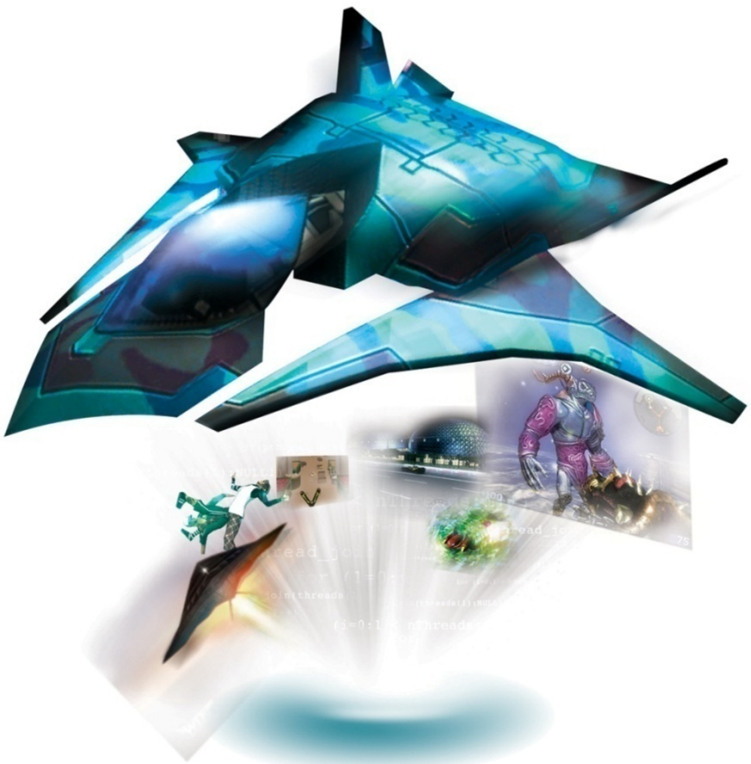
# DM2112
# DIGITAL ENTERTAINMENT SYSTEMS

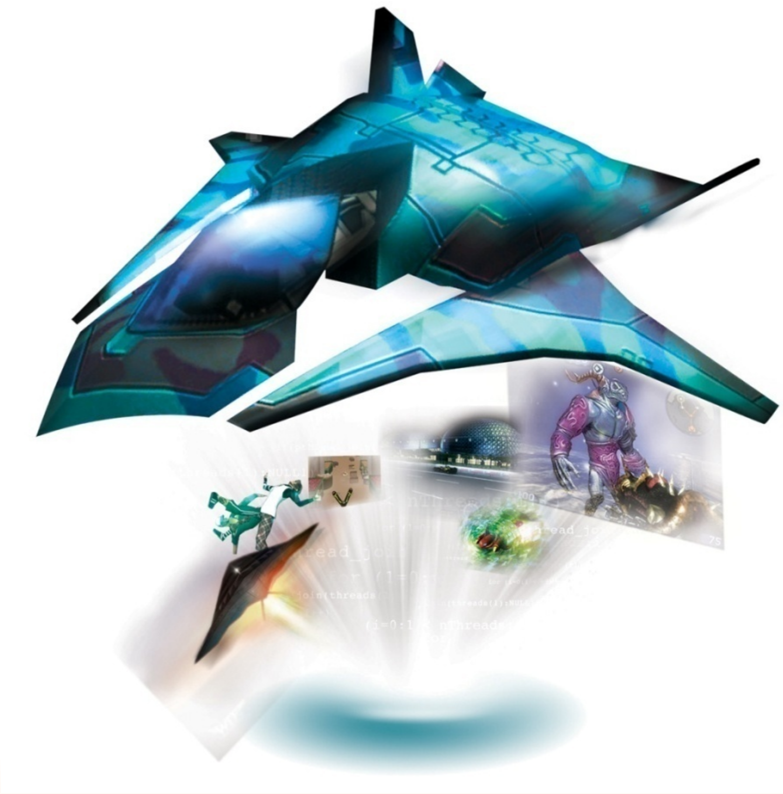


## Operating System Concepts Pt 1

# Today's Menu

- What is an Operating System?
- Interrupts
- Process Control Block
- Process Scheduling
- CPU Scheduling

# WHAT IS AN OPERATING SYSTEM?



# Operating Systems

- What's an Operating System (also called "OS")?

- It's a computer program that:
  - Manages all resources.
  - Decides between conflicting requests for efficient and fair resource use.
  - Controls execution of programs to prevent errors and improper use of the computer.

- A program that acts as an intermediary between a user of a computer and the computer hardware.

# Operating Systems

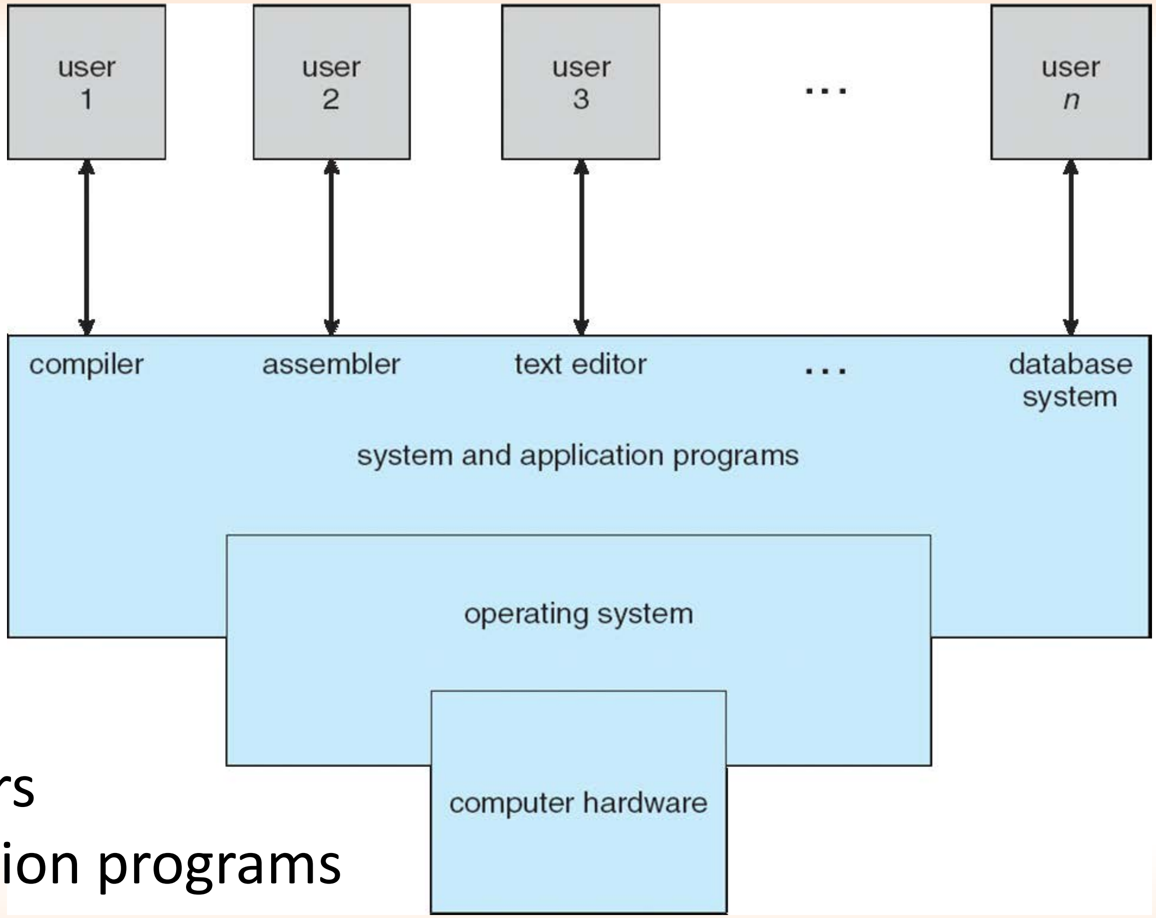
What it does for you:

- Executes your programs.
- Make a system more convenient for you to use.
- Makes solving problems easier.
- An operating system is a **resource allocator**.
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- An operating system is a **control program**.
  - Controls execution of programs to prevent errors and improper use of the computer

# Bootstrap / Computer Startup

- How is the operating system loaded?
  - **bootstrap program** is loaded at power-up or reboot (so that's why we say we "boot" up a computer).
    - Stored in ROM or EPROM
    - Loads basic part of OS (i.e. kernel) and starts execution

# The Components of a Computer System




1. The users
2. Application programs
3. Operating System
4. Actual hardware

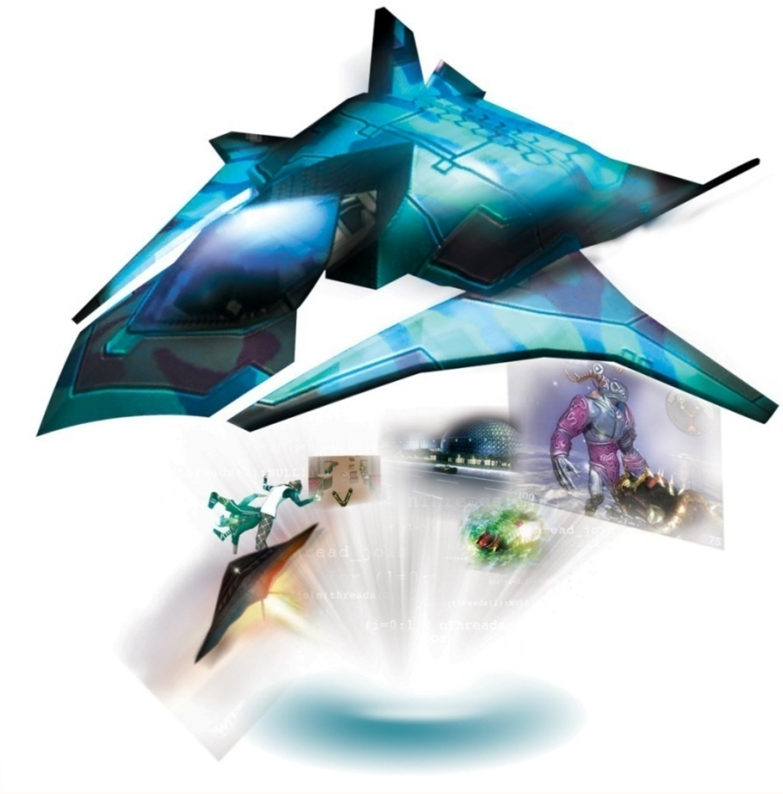
# The Components of a Computer System

A computer system can be divided into four components:

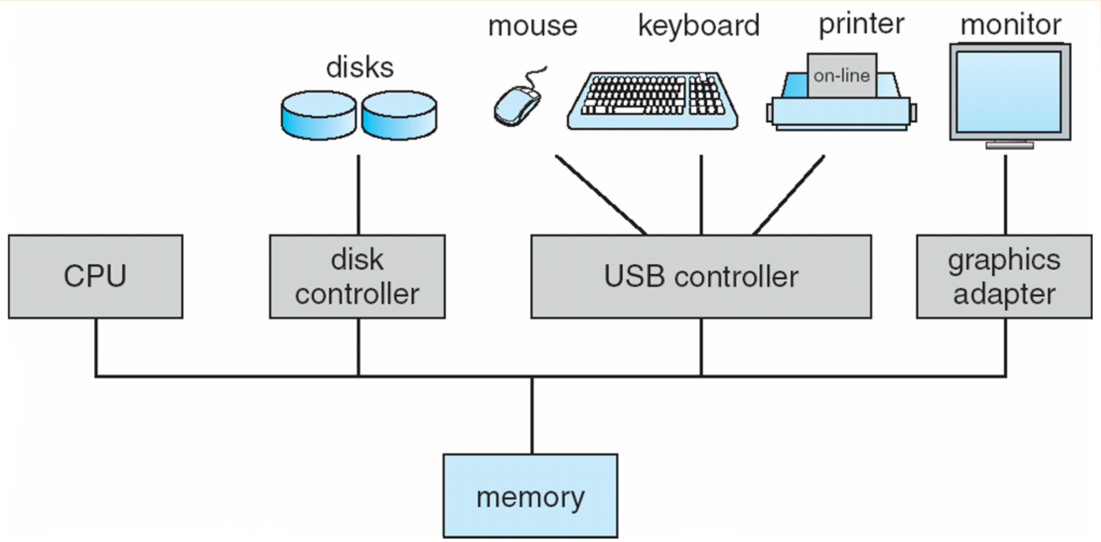
- **Hardware**
  - provides basic computing resources
  - CPU, memory, I/O devices
- **Operating system**
  - Controls and coordinates use of hardware among various applications and users
- **Application programs**
  - Define the ways in which the system resources are used to solve the computing problems of the users
  - Word processors, compilers, web browsers, database systems, video games
- **Users**
  - People, machines, other computers

- The operating system handles the interaction between the hardware, the application programs, and the users.

# INTERRUPTS



# Interrupts



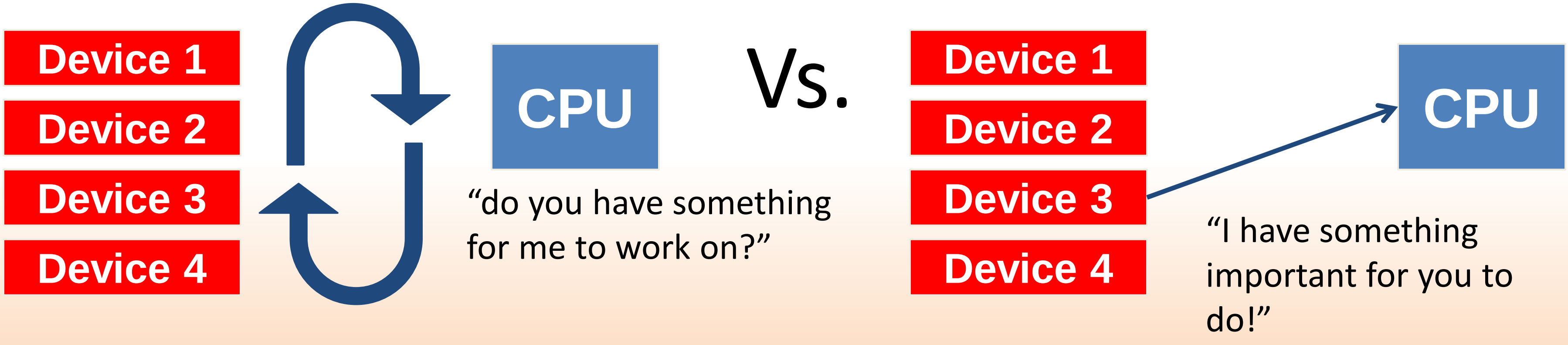

- A computer system has many devices competing for the CPU's attention.
  - Each kind of device has its own device controller
  - Each device controller has its own buffer
  - The CPU moves data to/from main memory to the device controller's buffers, and the controller transfers the data to/from the appropriate device
  - The device controller informs the CPU that it has finished its I/O operation by causing an interrupt

# What Happens When an Interrupt Occurs?

- The address of the interrupted instruction must be saved.
  - Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*
- The operating system preserves the state of the CPU by storing the register values (this CPU state is called the "context").
- The operating system determines which type of interrupt has occurred.
  - Polling (i.e. software interrupt)
  - Vectored interrupt (i.e. hardware interrupt)
- Separate segments of code determine what action should be taken for each type of interrupt.
- An operating system is **interrupt driven.**

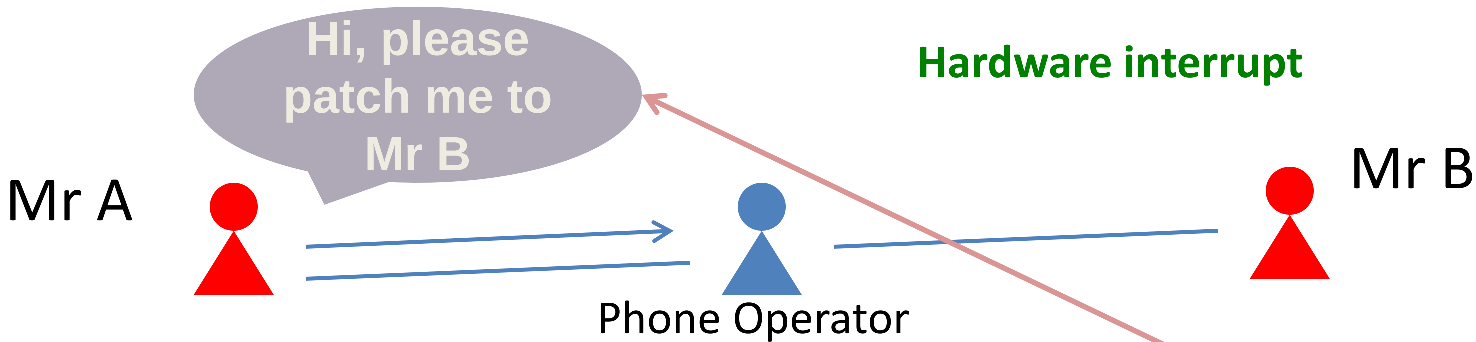
# Hardware Interrupts (IRQ)

- An interrupt request (IRQ) is a common way that a piece of hardware informs the OS that an important event has occurred.
  - Versus having CPU continuously poll ('ask') each device if there is an event
    - Polling is a costly process (Why?)
  - Fast response from CPU when triggered



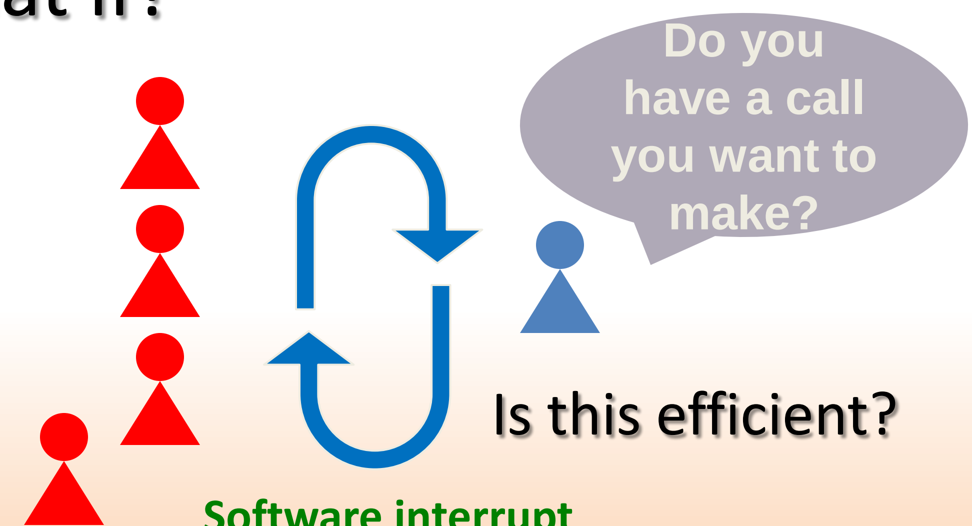
# Analogy – Phone Operator

- Long long ago (in a galaxy far far away?)




Hardware interrupt

- But what if?

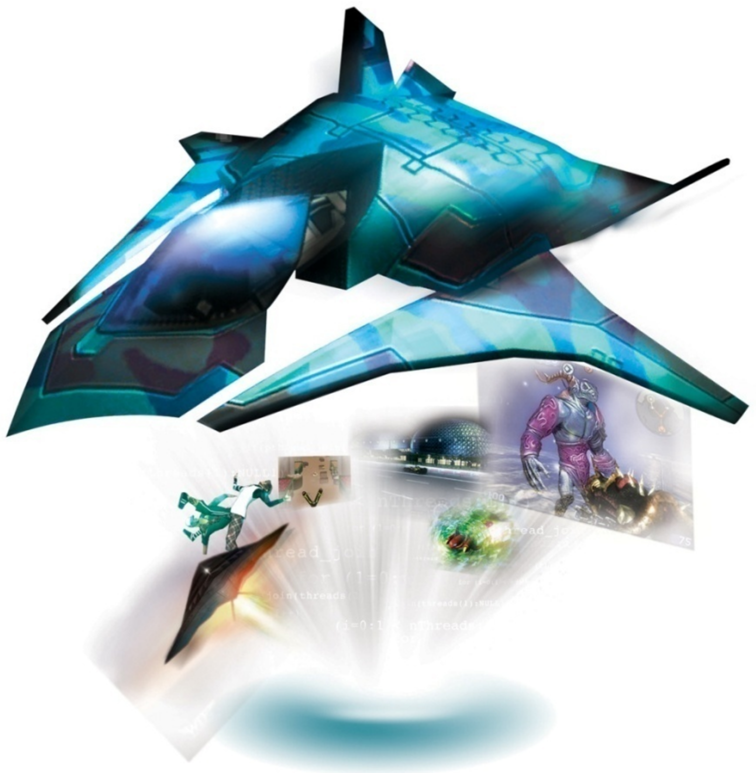



A good example of what an interrupt is like

Is this efficient? No!

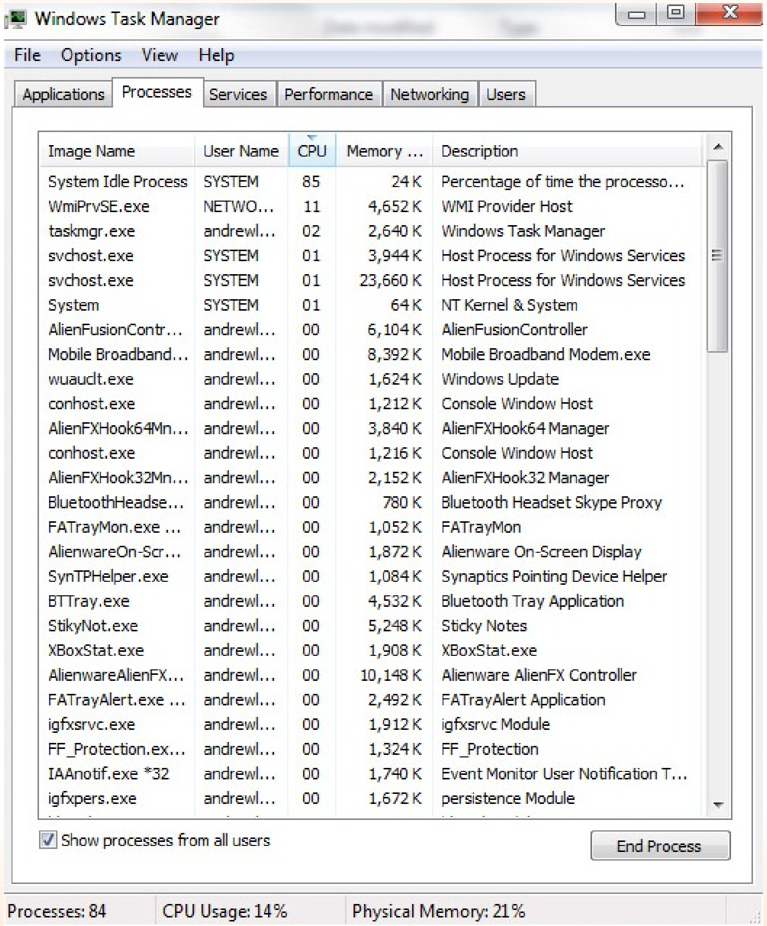
Software interrupt

# PROCESSES



# Concept of a Process

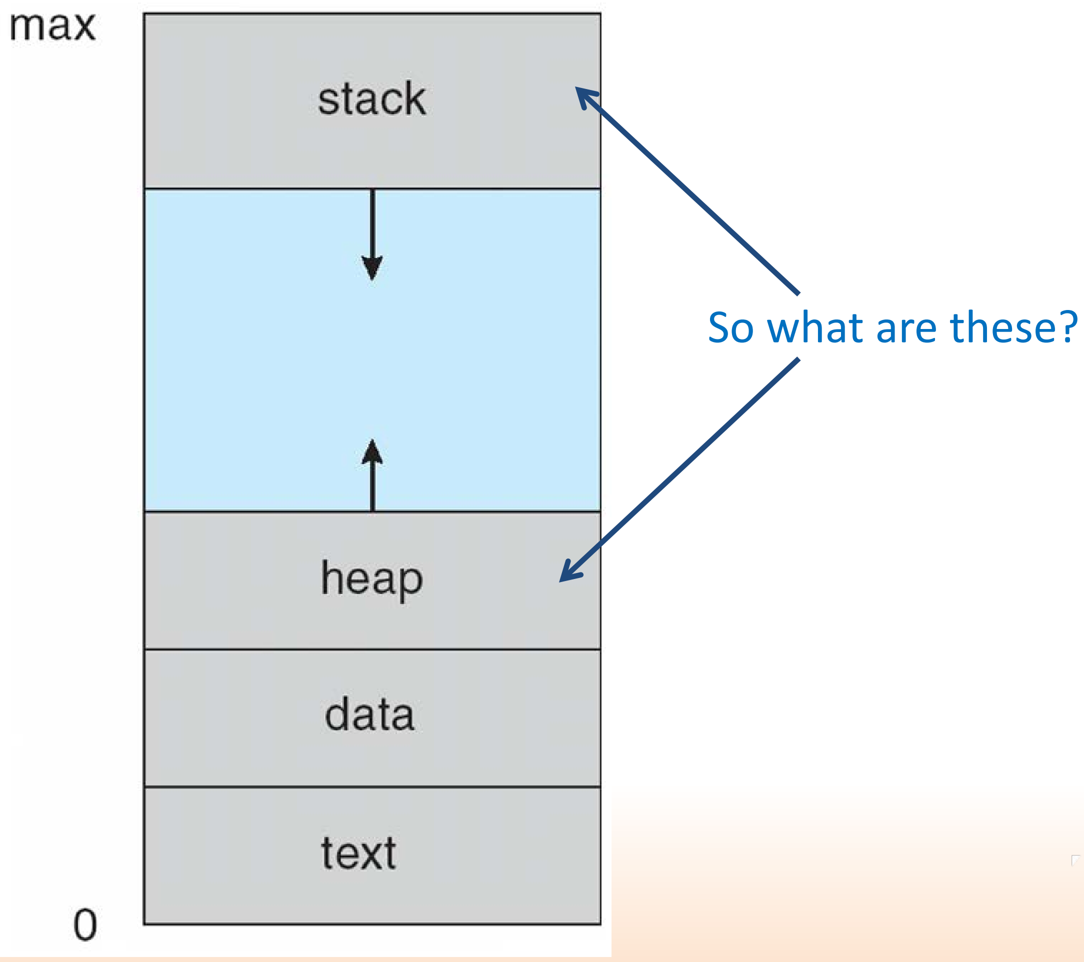
- An operating system executes a variety of programs:
  - ➢ Batch system – jobs
  - ➢ Time-shared systems – user programs or tasks
- We usually use the terms *job* and *process* almost interchangeably.



# Concept of a Process

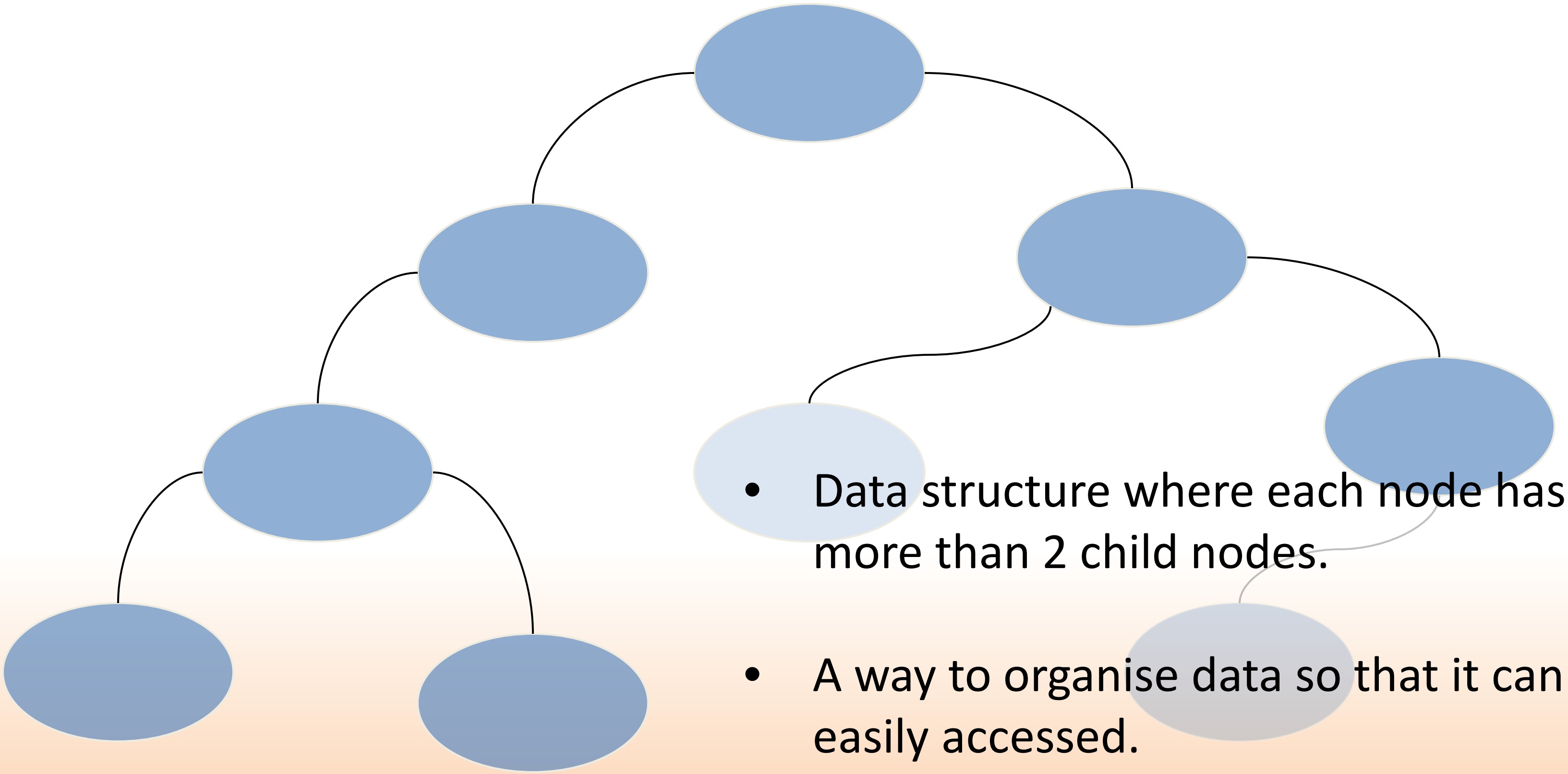
- Process – a program in execution; process execution must progress in sequential fashion.
- A program on its own is just a bunch of code. It is only called a process when a program is running (i.e. when it is executing).
- A process includes:
  - A program counter
  - A stack
  - A data section

# A Process in Memory



# Heap

- A form of binary tree.
- A heap is used to store long-term data.

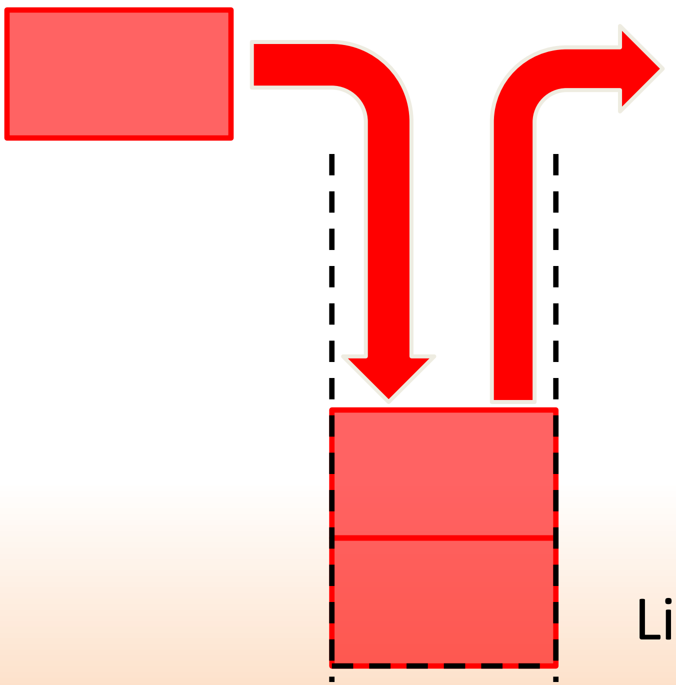


- Data structure where each node has no more than 2 child nodes.
- A way to organise data so that it can be easily accessed.

# Stack

- A reversed queue system.
  - Last in, First Out (LIFO or First In, Last Out - FILO)

- A stack is used to store short-term data.

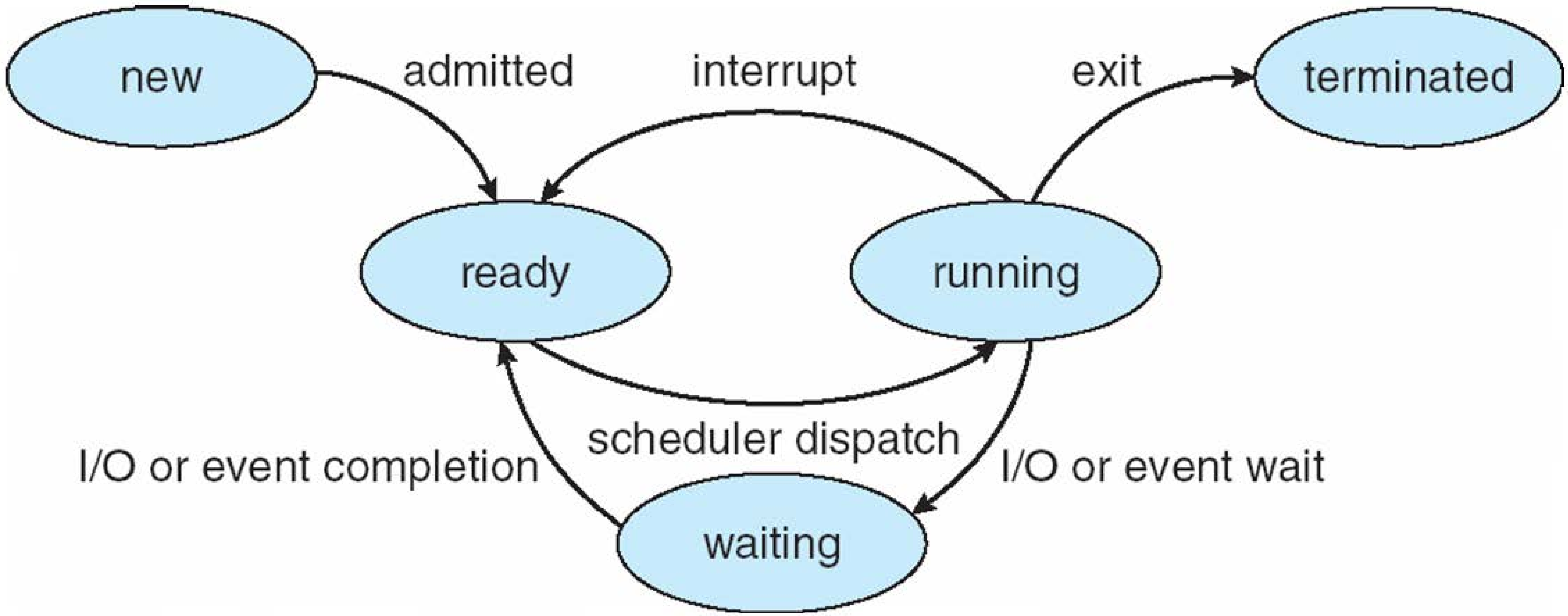


Like a stack of paper

# Process States

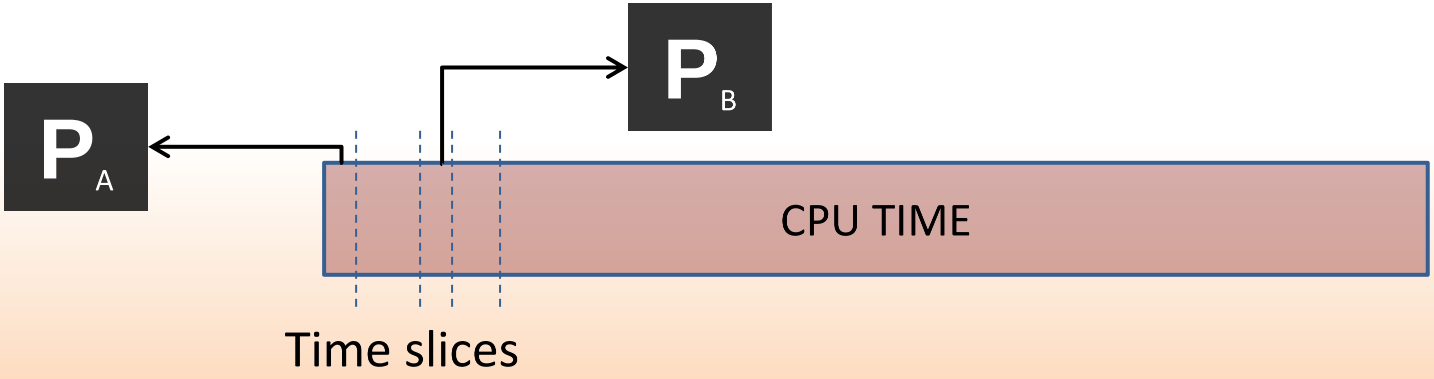
- As a process executes, it changes its *state.*
  - **New**: The process is being created
  - **Running**: Instructions are being executed
  - **Waiting**: The process is waiting for some event to occur
  - **Ready**: The process is waiting to be assigned to a processor
  - **Terminated**: The process has finished execution

# Diagram of Process State



# Process State

- Only 1 process can be running on a processor (i.e. CPU) at any given time.
- Multitasking is achieved by moving the current process out of the CPU, and then letting another process execute.
- CPU switches between processes every few milliseconds.
  - From user's point of view, all the programs are running at the same time
- The switching of processes every few seconds is called time slicing



# Process Creation

- Processes are arranged in a tree-like structure.
  - Can be parent or child of other processes
  - Root process has no parent (i.e. a root process is a kernel process)

- Process identified and managed via **process identifier (pid)**.

- Execution.
  - Parent and children execute concurrently
  - Parent waits until children terminate

# Example of Processes Running in a System

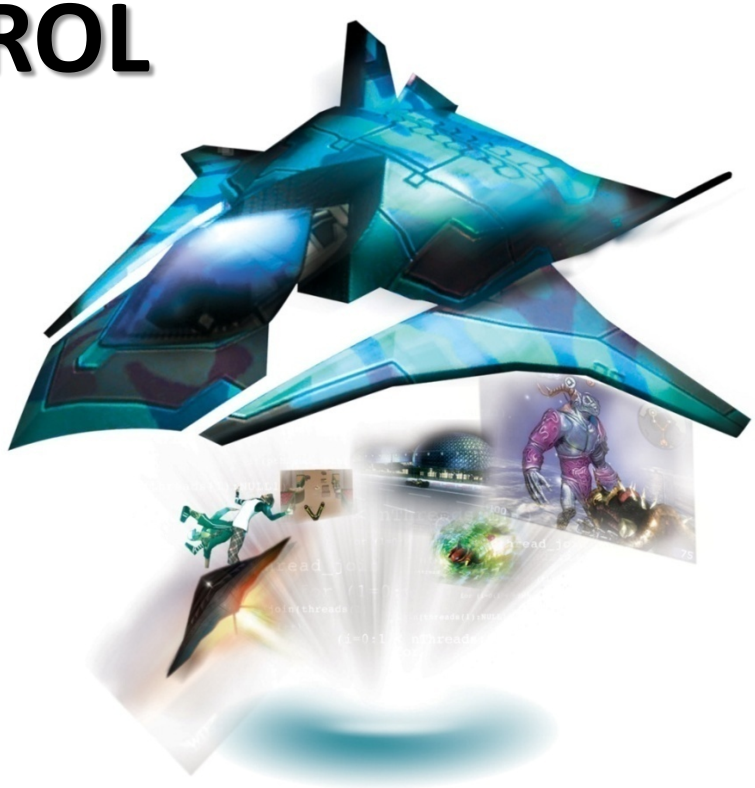
| PID | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+ | COMMAND |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 227 | root | 20 | 0 | 0 | 0 | 0 | S | 0 | 0.0 | 1:07.17 | usb-storage |
| 30904 | root | 20 | 0 | 19352 | 1344 | 960 | R | 0 | 0.0 | 0:00.05 | top |
| 1 | root | 20 | 0 | 24136 | 2300 | 1332 | S | 0 | 0.1 | 0:00.70 | init |
| 2 | root | 20 | 0 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.18 | kthreadd |
| 3 | root | 20 | 0 | 0 | 0 | 0 | S | 0 | 0.0 | 0:42.98 | ksoftirqd/0 |
| 6 | root | RT | 0 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.00 | migration/0 |
| 7 | root | RT | 0 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.00 | migration/1 |
| 9 | root | 20 | 0 | 0 | 0 | 0 | S | 0 | 0.0 | 0:15.45 | ksoftirqd/1 |
| 11 | root | 0 | -20 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.00 | cpuset |
| 12 | root | 0 | -20 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.00 | khelper |
| 13 | root | 0 | -20 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.00 | netns |
| 14 | root | 20 | 0 | 0 | 0 | 0 | S | 0 | 0.0 | 0:03.89 | kworker/u:1 |
| 15 | root | 20 | 0 | 0 | 0 | 0 | S | 0 | 0.0 | 0:01.13 | sync_supers |
| 16 | root | 20 | 0 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.07 | bdi-default |
| 17 | root | 0 | -20 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.00 | kintegrityd |
| 18 | root | 0 | -20 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.00 | kblockd |
| 19 | root | 0 | -20 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.00 | kacpid |
| 20 | root | 0 | -20 | 0 | 0 | 0 | S | 0 | 0.0 | 0:00.00 | kacpi_notify |

# Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Process’ resources are deallocated by operating system (the memory that it was using is cleared)

- Parent may terminate execution of children processes (**abort**) for the following reasons:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent process is exiting

# PROCESS CONTROL BLOCK



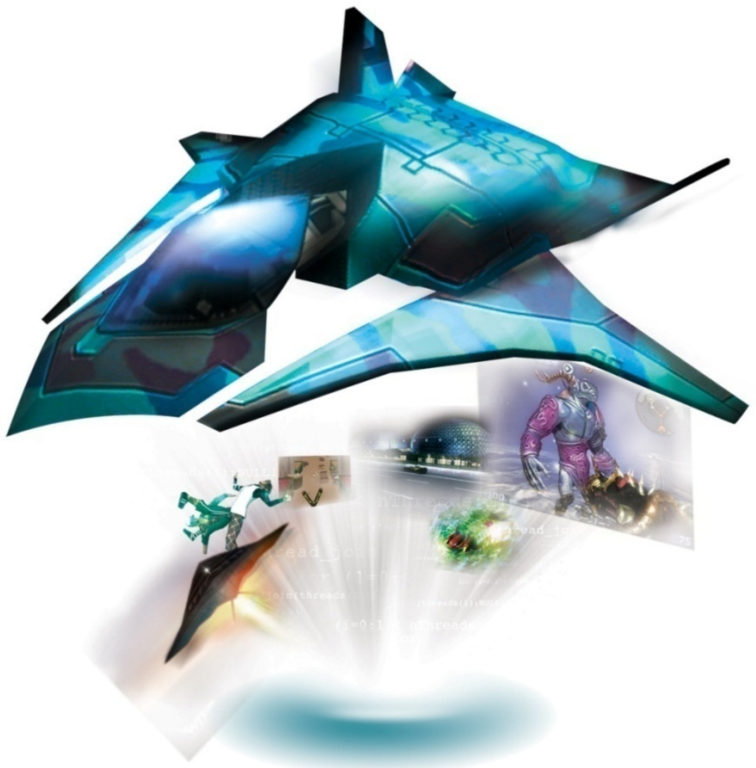
# Process Control Block (PCB)

Information associated with each process

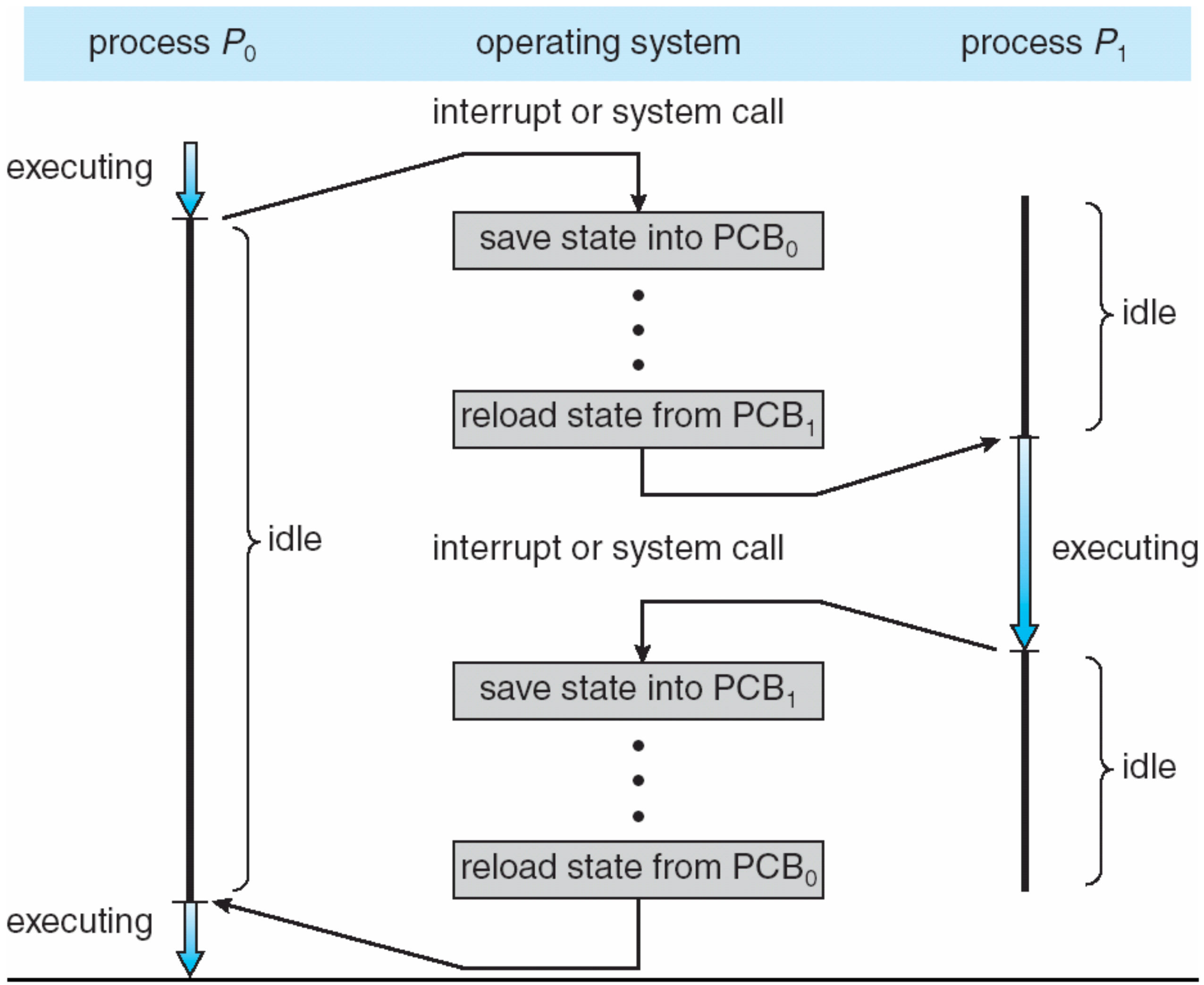
- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

# Process Control Block (PCB)

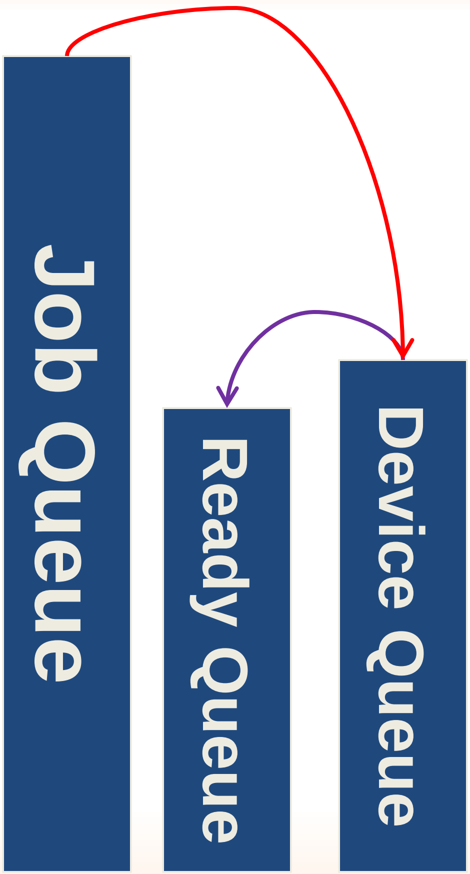
# PROCESS SCHEDULING



# CPU Switches From Process to Process



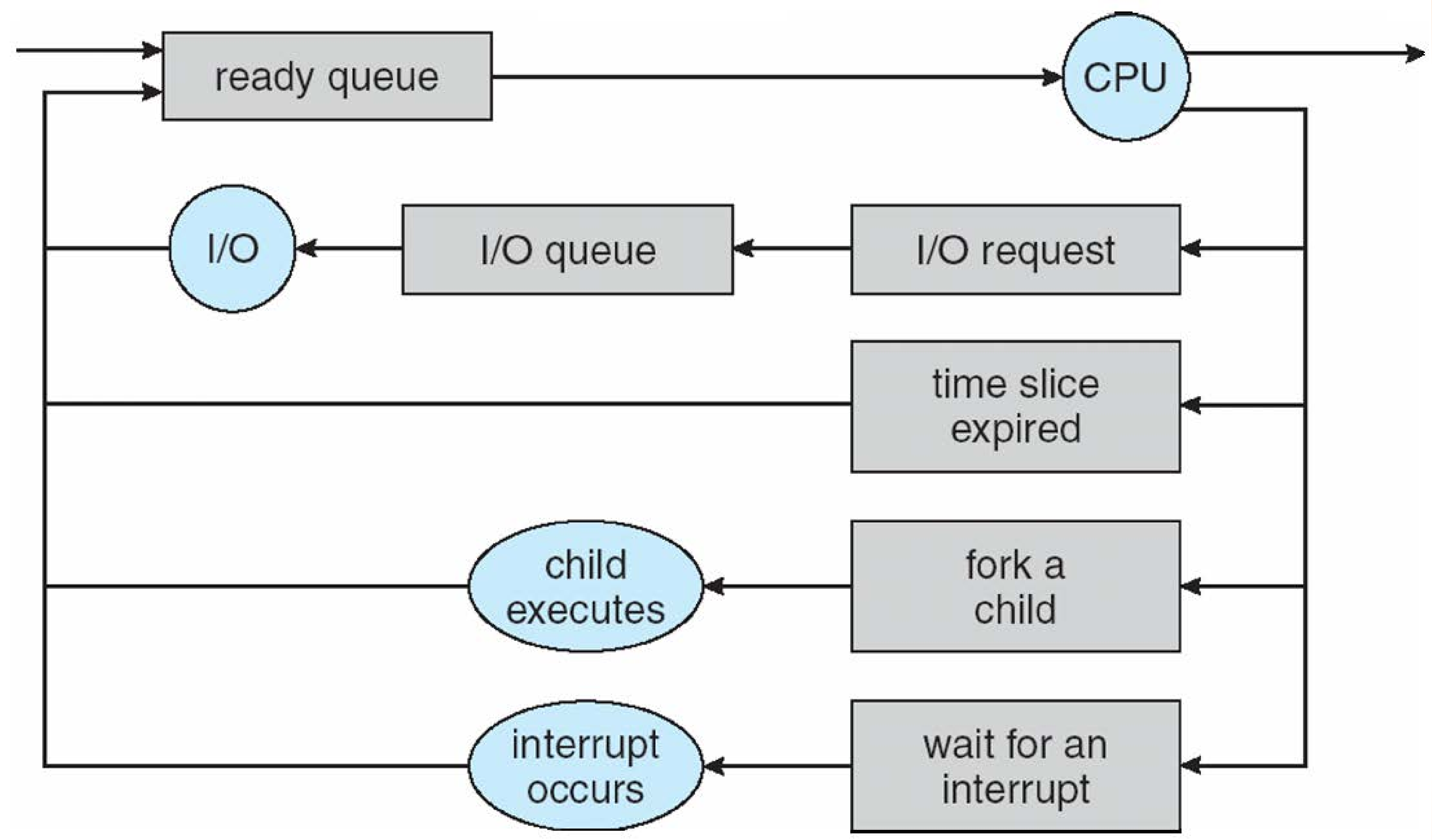
# Process Scheduling Queues




Processes move among the various queues

- **Job queue** – set of all processes in the system.
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute.
- **Device queues** – set of processes waiting for an I/O device.

# Representation of Process Scheduling



# Schedulers

- **Long-term scheduler**
  - Job scheduler
  - Selects which processes should be brought into the ready queue

- **Short-term scheduler**
  - CPU scheduler
  - Selects which process should be executed next and allocates CPU

# Schedulers

- **Short-term scheduler** is invoked very frequently (milliseconds) ⇒ (must be fast)

- **Long-term scheduler** is invoked very infrequently (seconds, minutes) ⇒ (may be slow)

# Processes & Scheduling

- 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

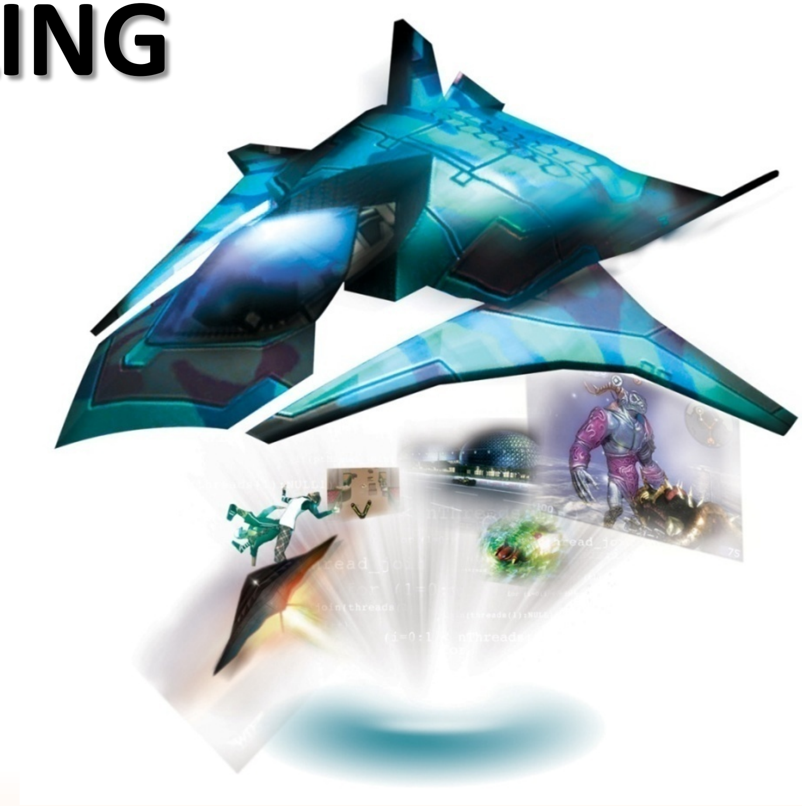
- CPU burst is the amount of time a process uses the processor before the process is no longer ready.
  - e.g. until the time when the process needs to wait for Input/Output such as waiting to write something to the harddisk, or waiting for user to type something on the keyboard

- I/O burst is the time a process is accessing the I/O devices until it has to wait for the CPU to be ready again.
  - e.g. while the process is accessing the I/O the CPU attends to other processes

# Context Switch

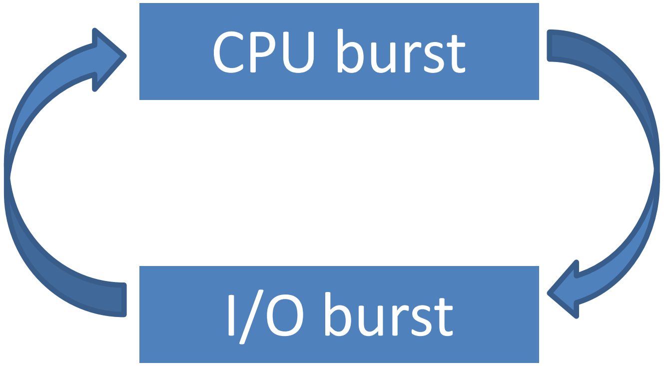
- When CPU switches to another process:
  - System must save the state of the old process
  - Load the saved state for the new process via a **context switch**
- Context of a process is represented in the PCB.
- Context switch time is overhead; the system does no useful work while switching.

# CPU SCHEDULING

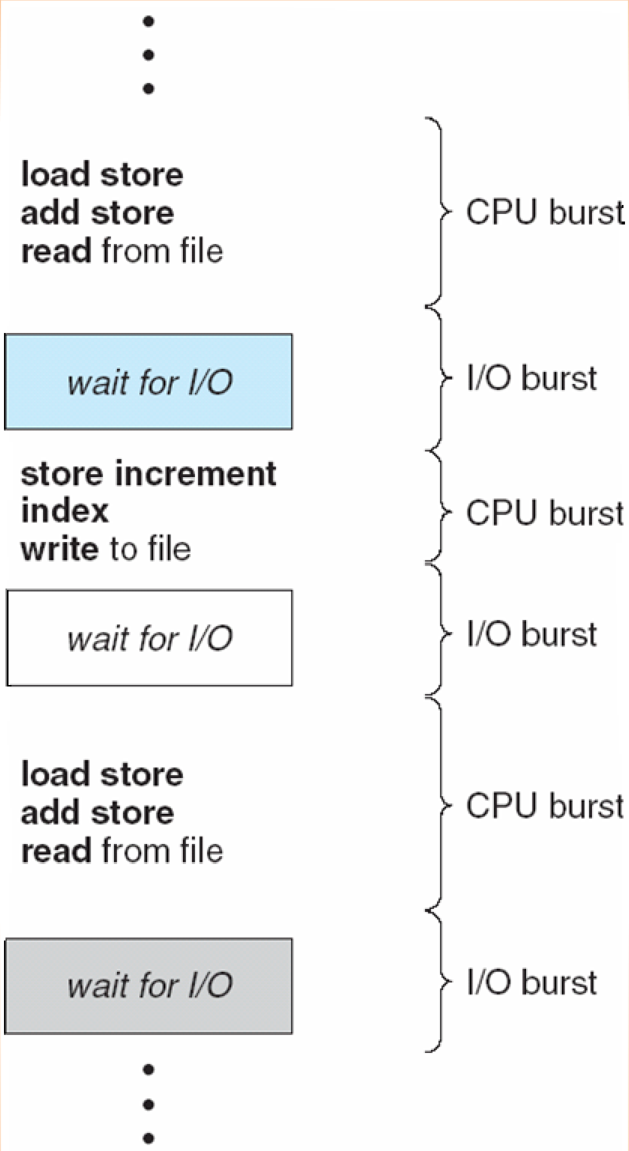


# Basic Concepts

- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.



# Alternating Sequence of CPU And I/O Bursts



# The Concept of Scheduling


A → B → C or

B → A → C or

B → C → A or

C → B → A

- Which job should we do first?
- Which job should come next?
- We need to consider things such as:
  - Which job came first?
  - Which job takes the shortest time to complete?

# Scheduling Criteria

- **CPU utilization** – aim is to keep the CPU as busy as possible.
- **Throughput** – Number of processes that complete their execution per time unit.
- **Turnaround time** – amount of time to execute a particular process.
- **Waiting time** – amount of time a process has been waiting in the ready queue.
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output  (for time-sharing environment).

# Scheduling Algorithm Optimization Criteria

- Maximum CPU utilization
- Maximum throughput
- Minimum turnaround time
- Minimum waiting time
- Minimum response time

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

| 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 (Continued)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1$$

- So now 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.

# Shortest-Job-First (SJF) Scheduling

- Associates with each process the length of its next CPU burst (i.e. the CPU burst time when the process is next using the CPU).

- Uses these lengths to schedule the process with the shortest time.

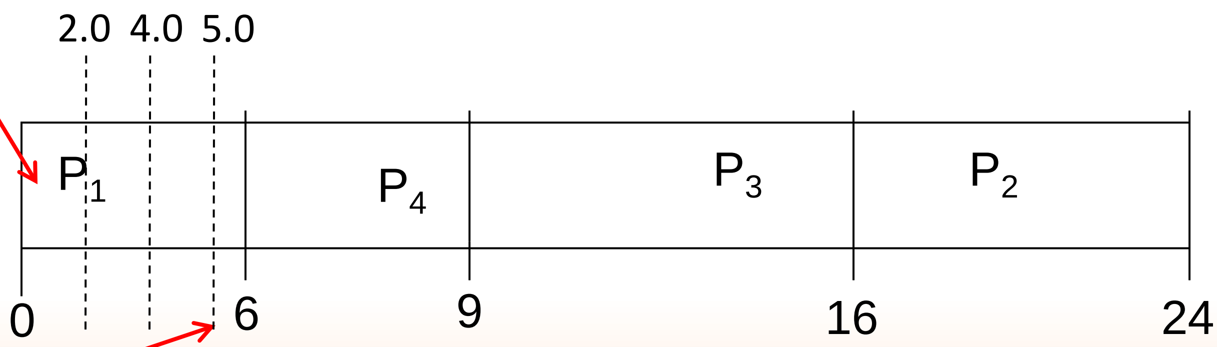
- SJF is optimal – gives minimum average waiting time for a given set of processes.
  - The difficulty is knowing the length of the next CPU request

# Example of SJF

| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 6 |
| $P_2$ | 2.0 | 8 |
| $P_3$ | 4.0 | 7 |
| $P_4$ | 5.0 | 3 |

Note that P1 runs first,
Because it arrived at time 0.0 where no other processes have arrived

SJF scheduling chart:




By time 5.0, all other jobs have arrived

Average waiting time = ((0-0) + (6-5) + (9-4) + (16-2)) / 4 = (0+1+5+14)/4 = 5

# FYI: Other Types of Scheduling

- Priority
- Round Robin
- Lottery
- Least Slack Time

- Not discussed in this module but do read up on them for your own enrichment.

# Acknowledgements

- Slides adapted from Operating System Concepts (8th Edition), by Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin. John Wiley & Sons Inc., ISBN 0-470-12872-0

# More Next Time

- Questions?