COS3023
Operating Systems and Concurrency

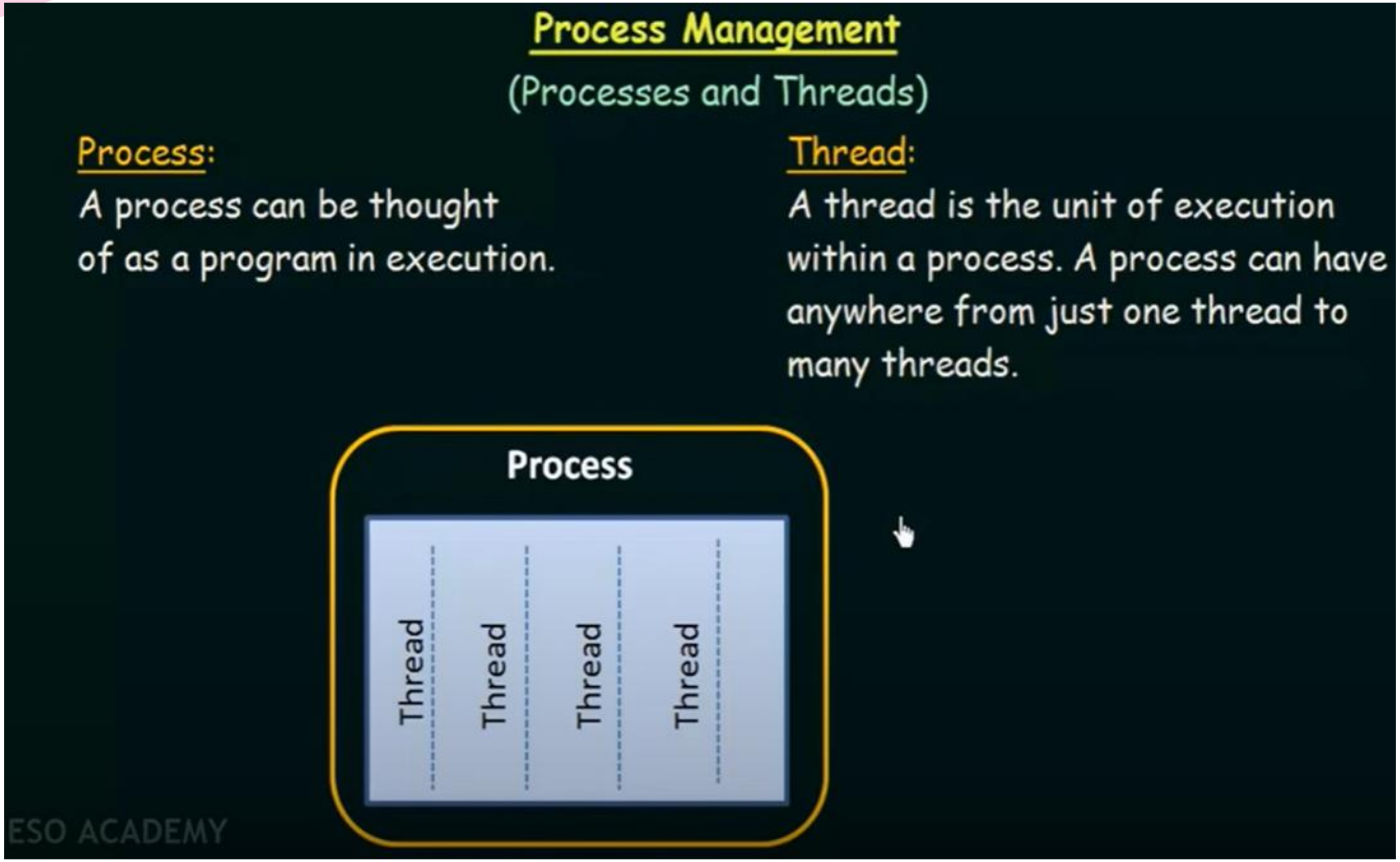
TOPIC 2- PROCESS MANAGEMENT (PART 1)

LECTURER : MS SHAFRAH

LEARNING OBJECTIVES

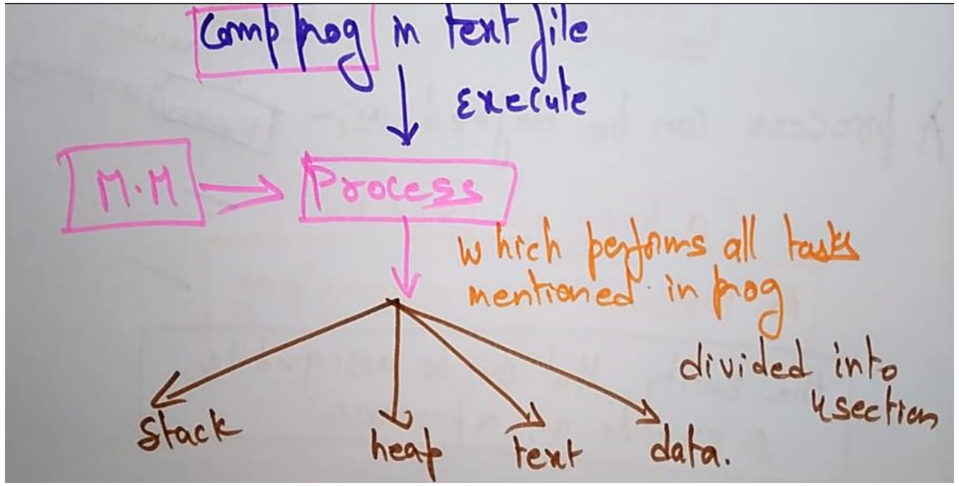
- To introduce the notion of a process—a program in execution, which forms the basis of all computation.
- To describe the various features of processes, including scheduling, creation, and termination.
- To explore interprocess communication using shared memory and message passing.
- To describe communication in client–server systems.

Process Management



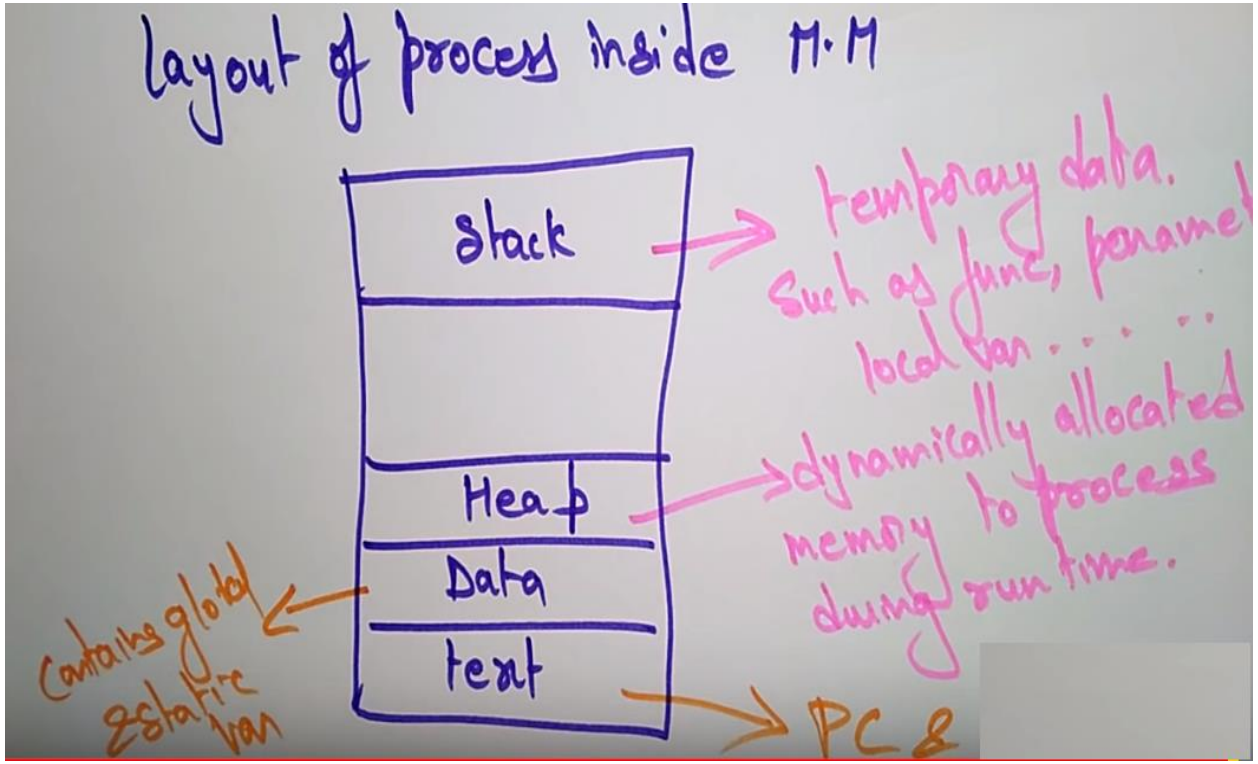
Process

- When a program entered the main memory, it is called a process.
- A process can be defined as:
    - A program in execution or
    - An instance of a running program or
    - An entity, that can be assigned to and execute on a processor.
- Example:

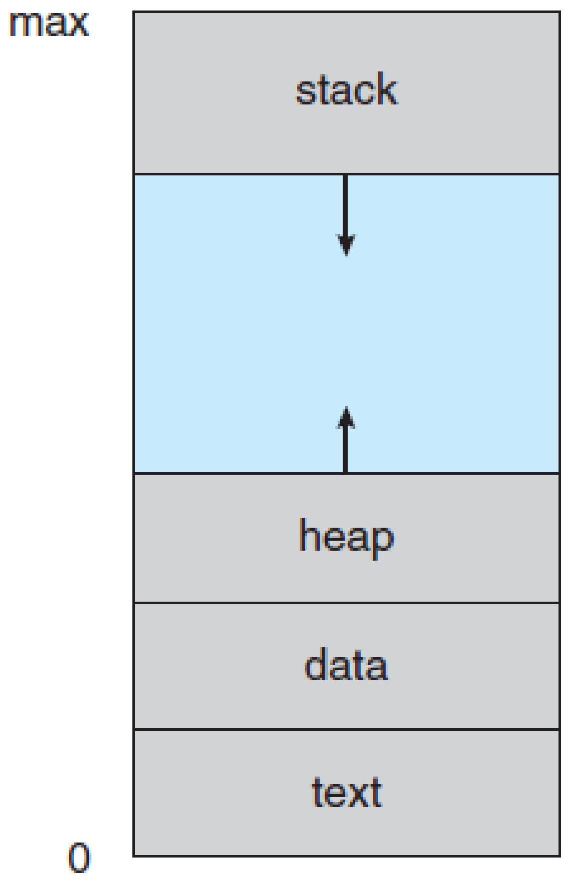


Layout of process inside main memory

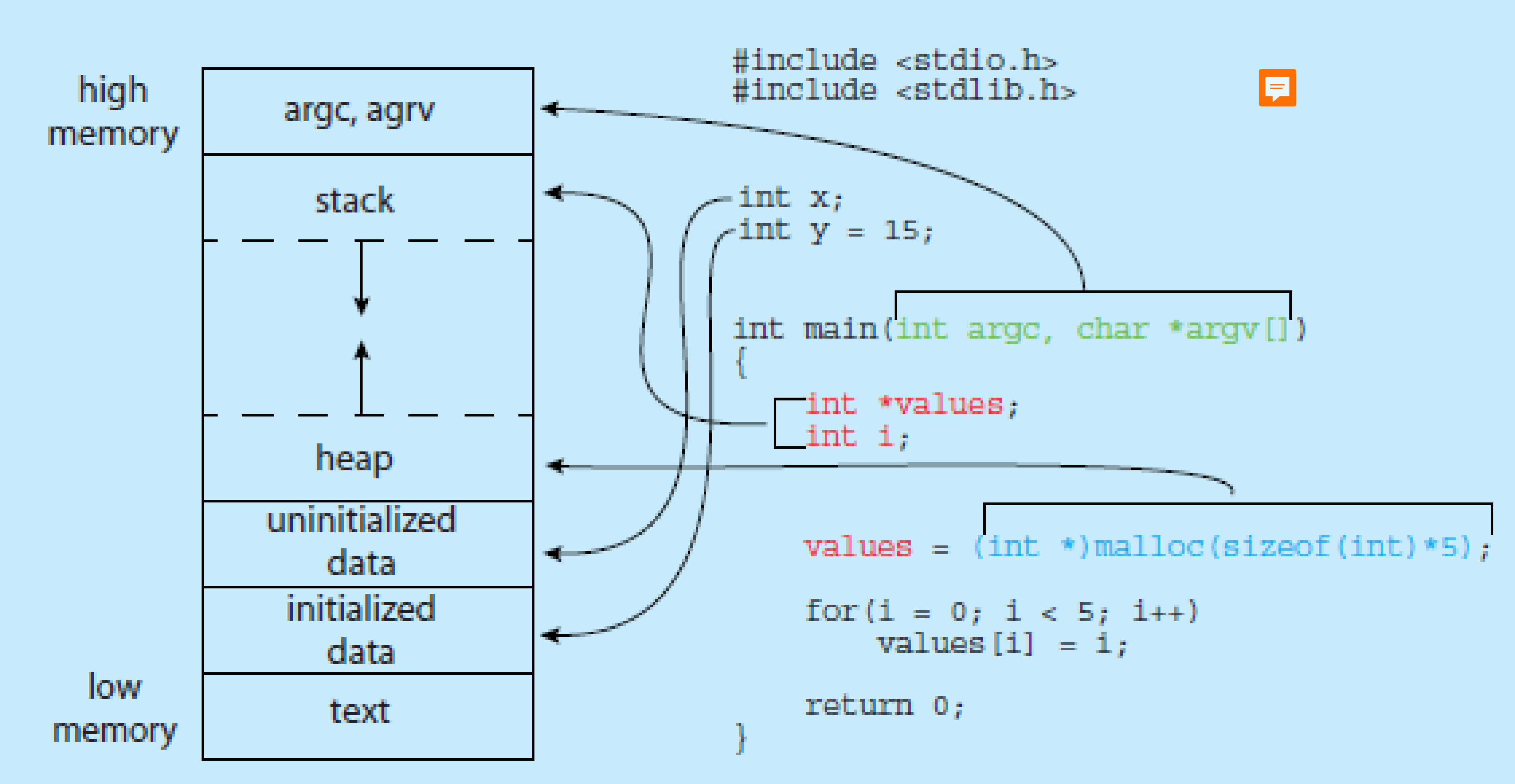
- A process can be divided into : Stack – Heap – Text - Data



Layout of a process in memory

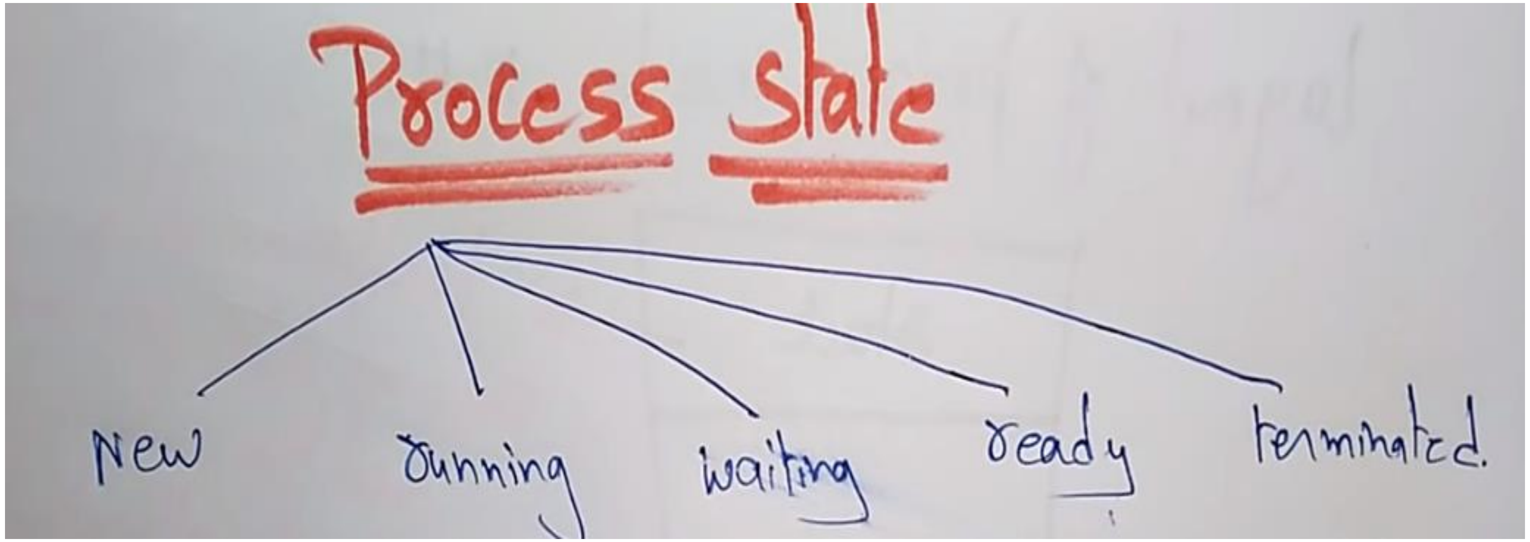


MEMORY LAYOUT OF A C PROGRAM

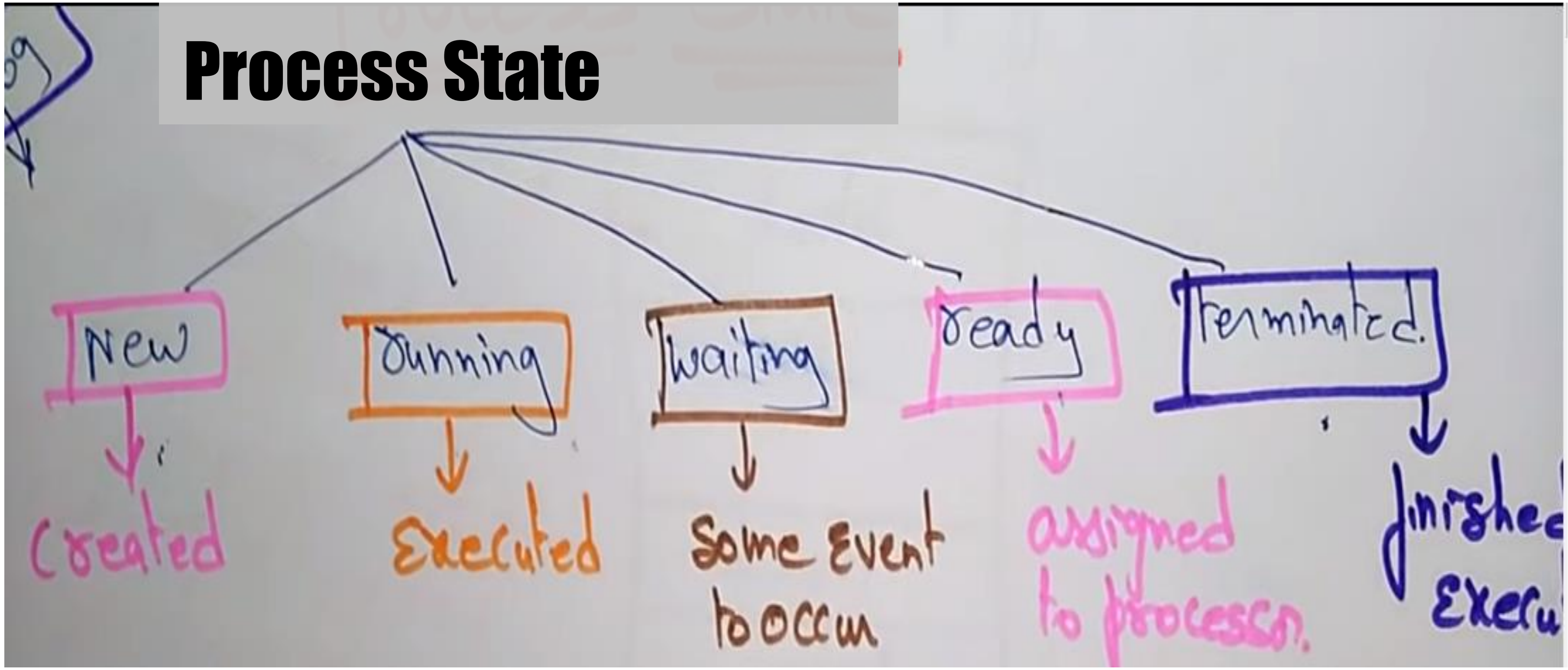


Process States

- When a process is executing, it has changes in states.



Process States

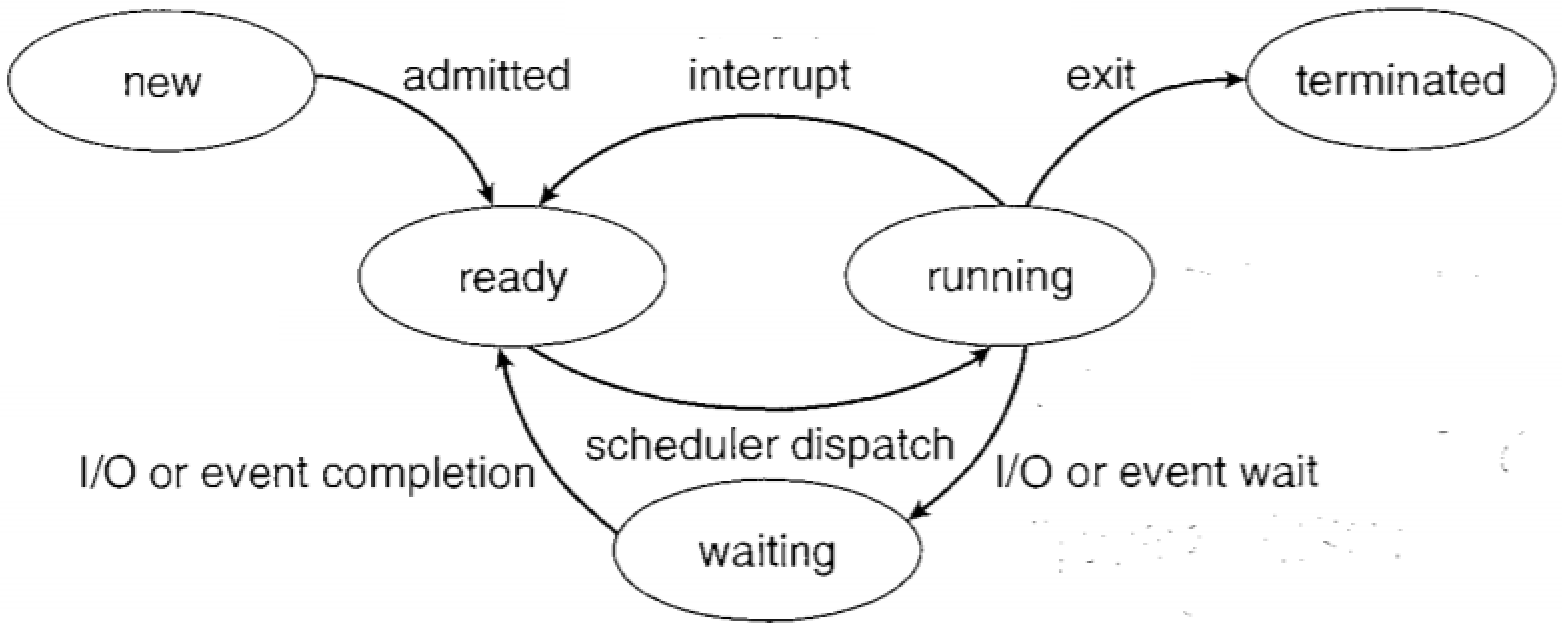


Process Execution State

- Execution state of a process indicates what it is doing
  - *new:* the OS is setting up the process state
  - *running:* executing instructions on the CPU
  - *ready:* ready to run, but waiting for the CPU
  - *waiting:* waiting for an event to complete
  - *terminated:* the OS is destroying this process
- As the program executes, it moves from state to state, as a result of the program actions (e.g., system calls), OS actions (scheduling), and external actions (interrupts).

Process State

State diagram



Process State - example

```
void main() {
printf('Hello
World');
}
```

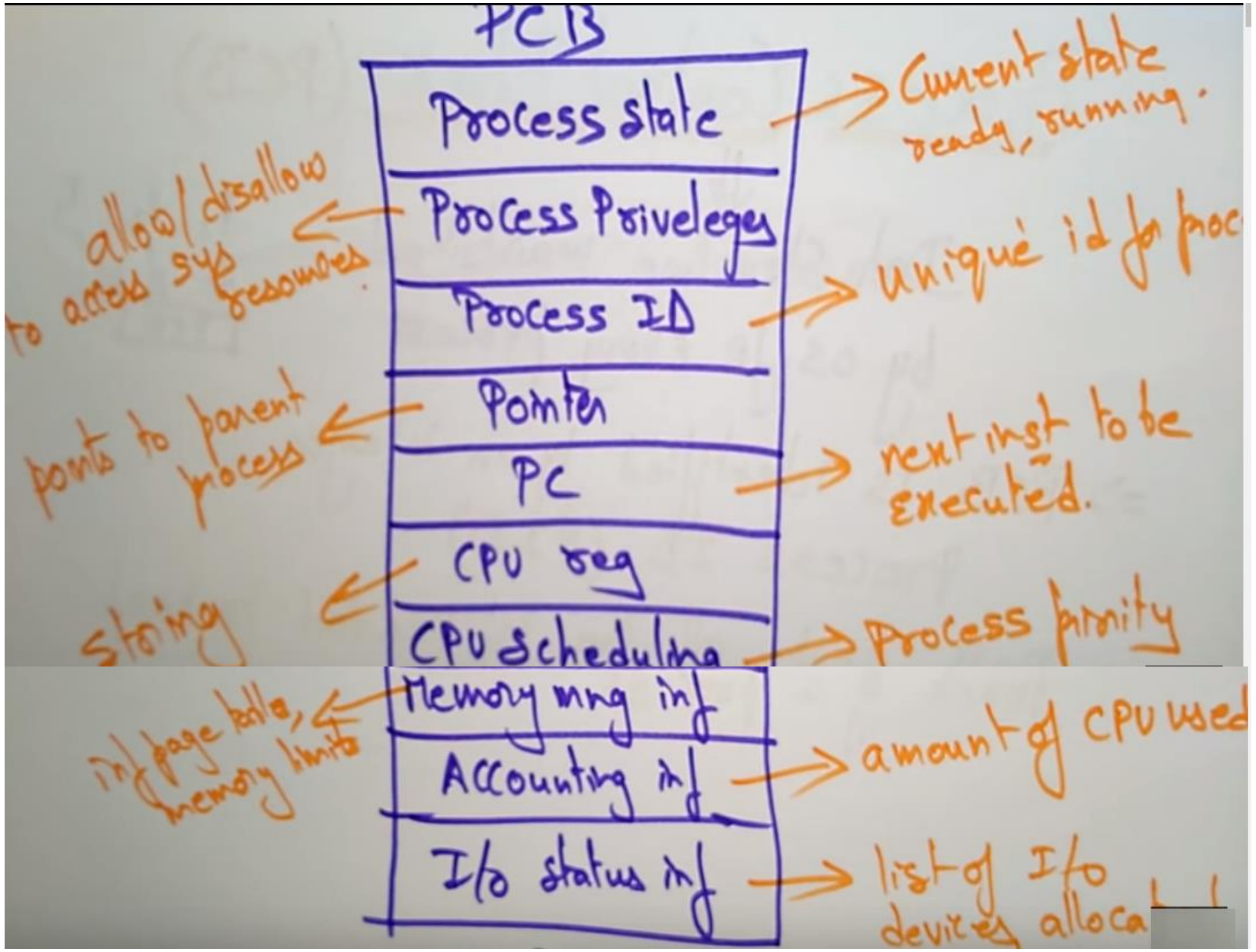
State sequence:

new
ready
running waiting for
I/O ready
running
terminated

Process Control Block (PCB)

- A data structure maintained by OS for every process to keep track of all processes.
- Each process's detail is stored in PCB.
- PCB is identified by an integer which is Process ID (PID). Eg : P1, P2, P3, ..
- PCB keeps all the information needed to keep track of a process.
- What type of information?

Program Control Block (PCB)



Threads

- The process model discussed so far has implied that a process is a program that performs a single thread of execution.
- For example, when a process is running a word-processor program, a single thread of instructions is being executed.
- This single thread of control allows the process to perform only one task at a time.

Threads

- Most modern operating systems have extended the process concept to allow a process to have multiple threads of execution
- Beneficial on multicore systems, where multiple threads can run in parallel.
- A multithreaded word processor could, for example, assign one thread to manage user input while another thread runs the spell checker.
- On systems that support threads, the PCB is expanded to include information for each thread.
- Other changes throughout the system are also needed to support threads.

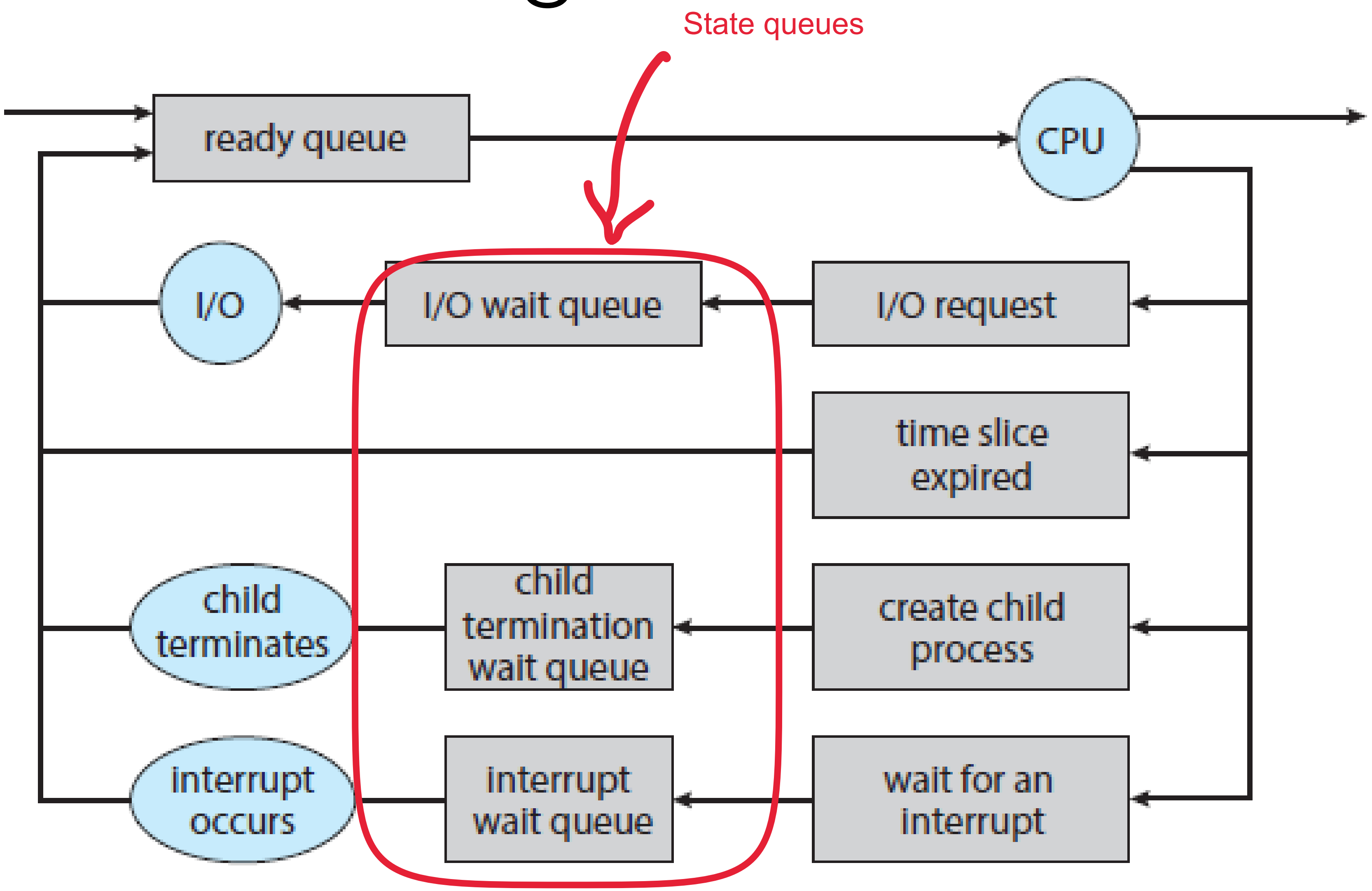
Process Scheduling

- Multiprogramming aims to have a process running at all times to maximize CPU utilization.
- Time sharing aims to switch a CPU core among processes frequently, allowing users to interact with running programs.
- The process scheduler selects an available process for program execution on a core.
- A single CPU core system can only run one process at a time, while a multicore system can run multiple processes simultaneously.
- In a multicore system, if there are more processes than cores, excess processes have to wait until a core becomes available for rescheduling.

Process State Queues

- The OS maintains the PCBs of all the processes in *state queues*.
- The OS places the PCBs of all the processes in the same *execution state* in the same queue.
- When the OS changes the state of a process, the PCB is unlinked from its current queue and moved to its new state queue.
- The OS can use different policies to manage each queue.
- Each I/O device has its own wait queue.

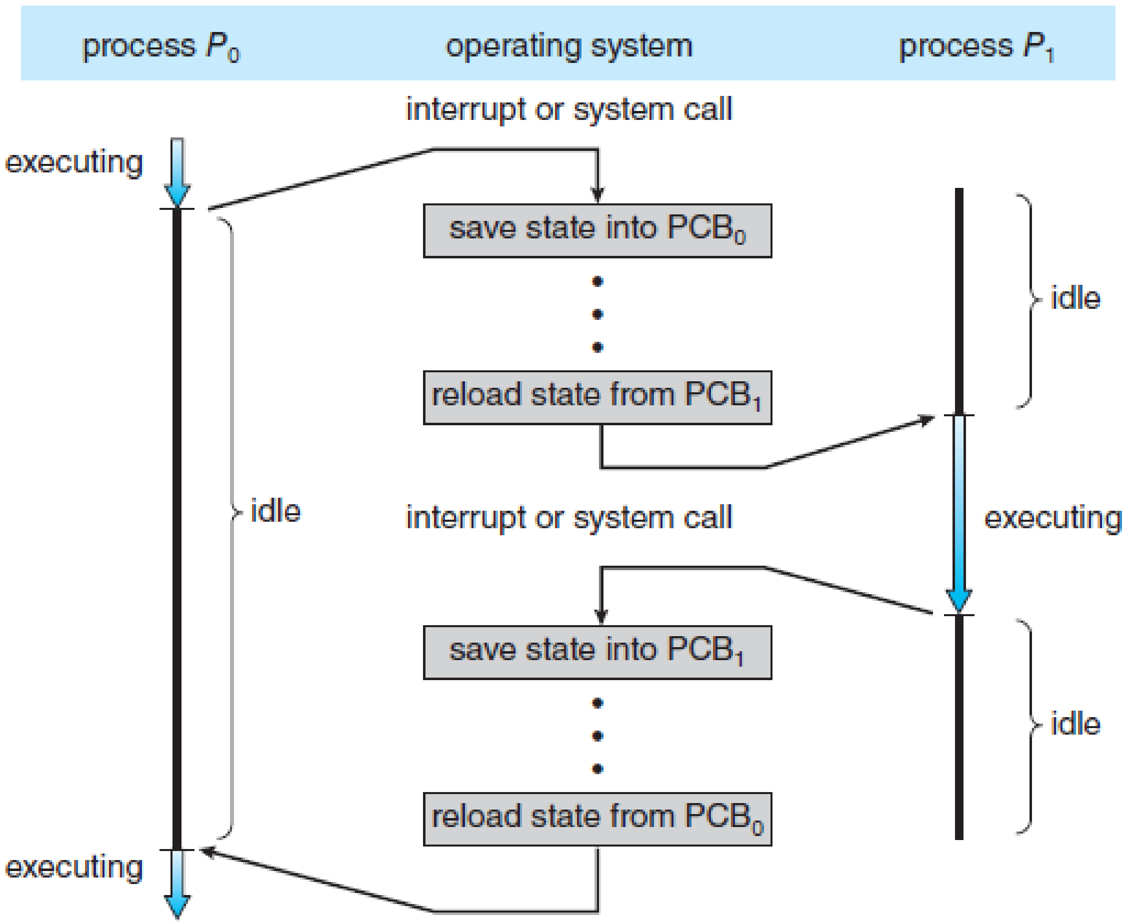
Queueing-diagram representation of process scheduling



Context Switch

- Starting and stopping processes is called a **context switch**, and is a relatively expensive operation.
- The OS starts executing a ready process by loading hardware registers (PC, SP, etc) from its PCB
- While a process is running, the CPU modifies the Program Counter (PC), Stack Pointer (SP), registers, etc.
- When the OS stops a process, it saves the current values of the registers, (PC, SP, etc.) into its PCB

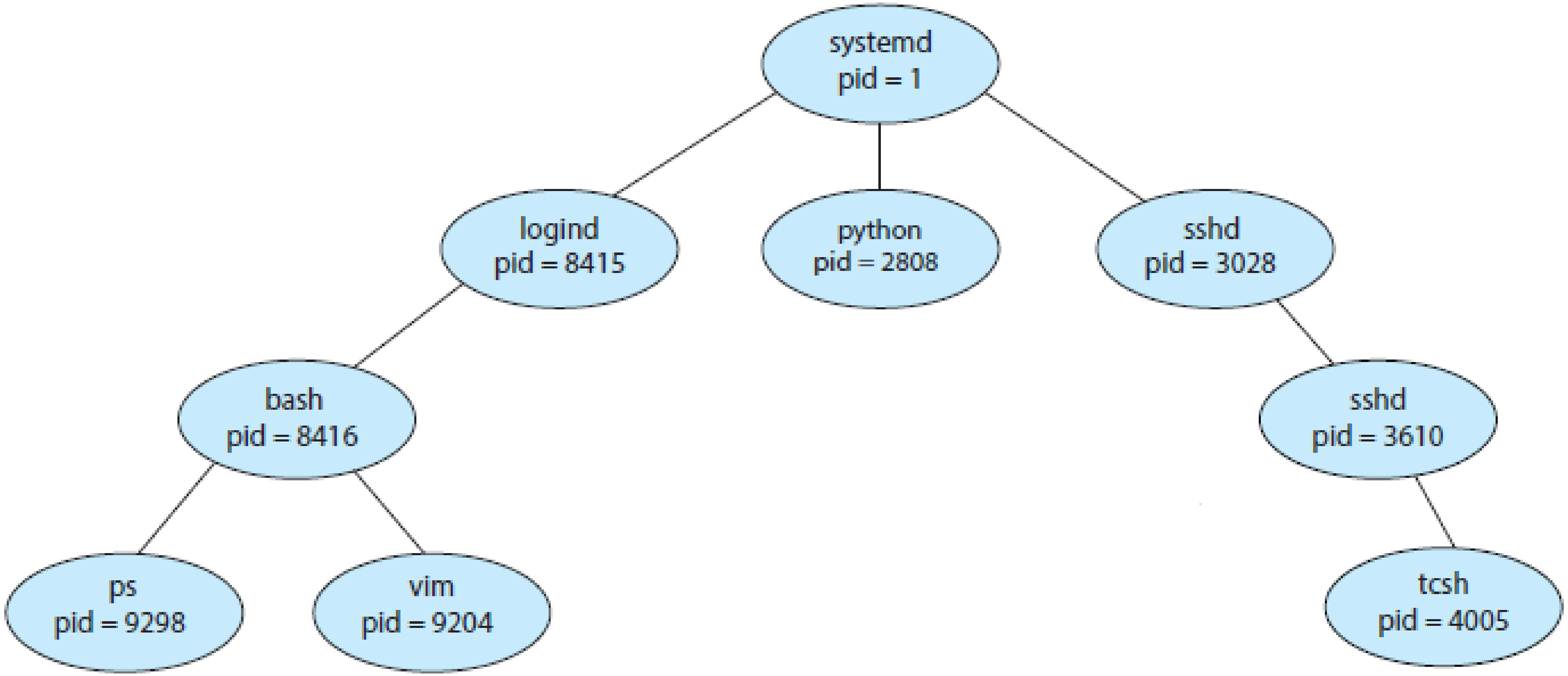
Diagram showing context switch from process to process.



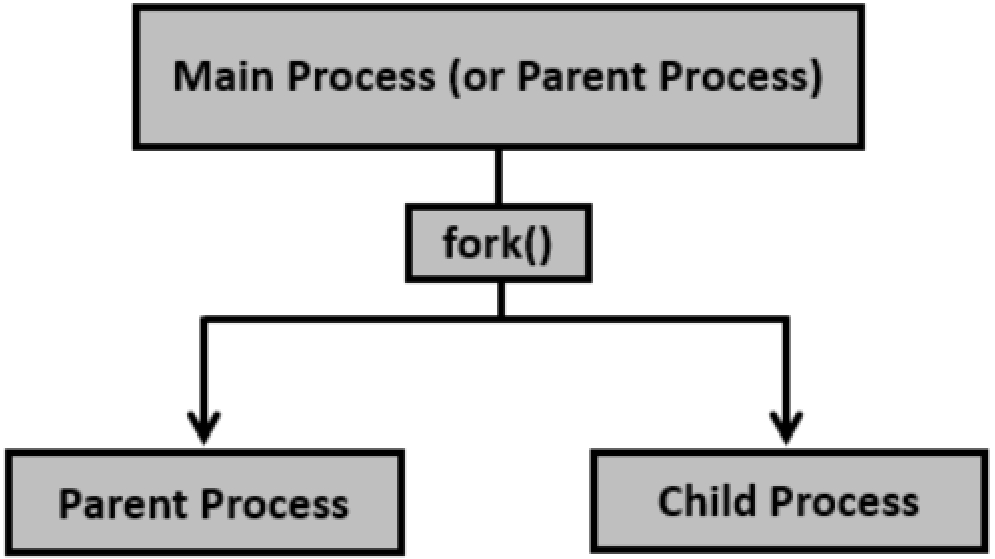
Creating a Process

- One process can create other processes to do work.
  - – The creator is called the *parent* and the new process is the *child*
  - – The parent defines (or donates) resources and privileges to its children
  - – A parent can either wait for the child to complete, or continue in parallel
- In Unix, the *fork* system call is used to create child processes
- Fork copies variables and registers from the parent to the child
- – The only difference between the child and the parent is the value returned by fork
  * In the parent process, fork returns the process id of the child
- * In the child process, the return value is 0

A tree of processes on a typical Linux system



Process Creation



Process Termination

- On process termination, the OS reclaims all resources assigned to the process.
- In Unix :
  - a process can terminate itself using the *exit* system call.
  - a process can terminate a child using the *kill* system

Cooperating Processes

- Any two process are either independent or cooperating
- Cooperating processes work with each other to accomplish a single task.
- Cooperating processes can
  - improve performance by overlapping activities or performing work in parallel,
  - enable an application to achieve a better program structure as a set of cooperating processes, where each is smaller than a single monolithic program, and
  - easily share information between tasks.

Explanation

- **New Process Admitted to Ready State:**
  - When a process has been created and initialized, it moves from the New state to the Ready state.
  - In the Ready state, the process is waiting to be assigned the CPU for execution.
  - The scheduler is responsible for selecting a process from the Ready state to move it to the Running state.
- **Ready State to Running State:**
  - When the scheduler selects a process from the Ready state, it moves to the Running state.
  - In the Running state, the process is being executed by the CPU.
- **Running State to Terminated State:**
  - When a process completes its execution, it transitions from the Running state to the Terminated state.
  - In the Terminated state, the process has finished its execution and will be removed from the system.

THANK YOU

Explanation

- **Running State Interrupt:**
  - If a process is interrupted while it is in the Running state (e.g., due to a hardware interrupt or a signal), it moves back to the Ready state.
  - The interruption could be caused by an external event or an I/O request.
- **Running State I/O or Event Wait:**
  - If a process in the Running state needs to wait for an I/O operation or an event to complete, it moves to the Wait state.
  - The process temporarily suspends its execution until the I/O operation or event is finished.
- **Wait State I/O or Event Completion:**
  - When the required I/O operation or event is completed, a process in the Wait state transitions back to the Ready state.
  - It is now ready to be scheduled and resume its execution.