Chapter : Processes

Chapter : Processes

- Process Concept
- Process Scheduling
- Operations on Processes

Process

- A program is a passive entity, such as a file containing a list of instructions stored on disk.

- Process is an instance of a program.
- A process is an active entity with a program counter specifying the next instruction to execute and a set of associated resources.

Process

- Process – a program in execution
- Process execution must progress in sequential manner
- A process includes:
  - program counter
  - stack
  - data section

Process

- **Process memory** is divided into four sections for efficient working :
- The **Text section** is made up of the compiled program code
- The **Data section** is made up of the global and static variables
- The **Heap** is used for the dynamic memory allocation
- The **Stack** is used for local variables.

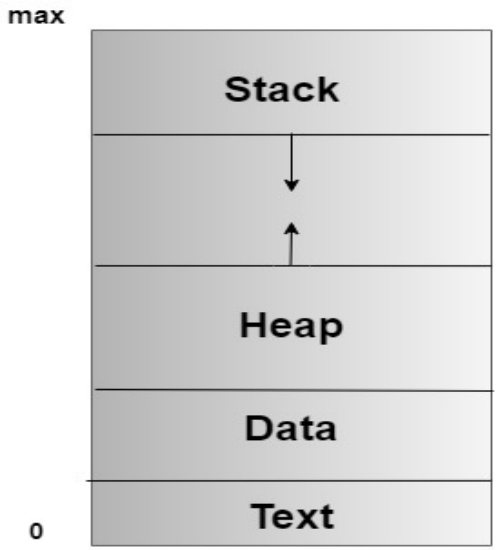
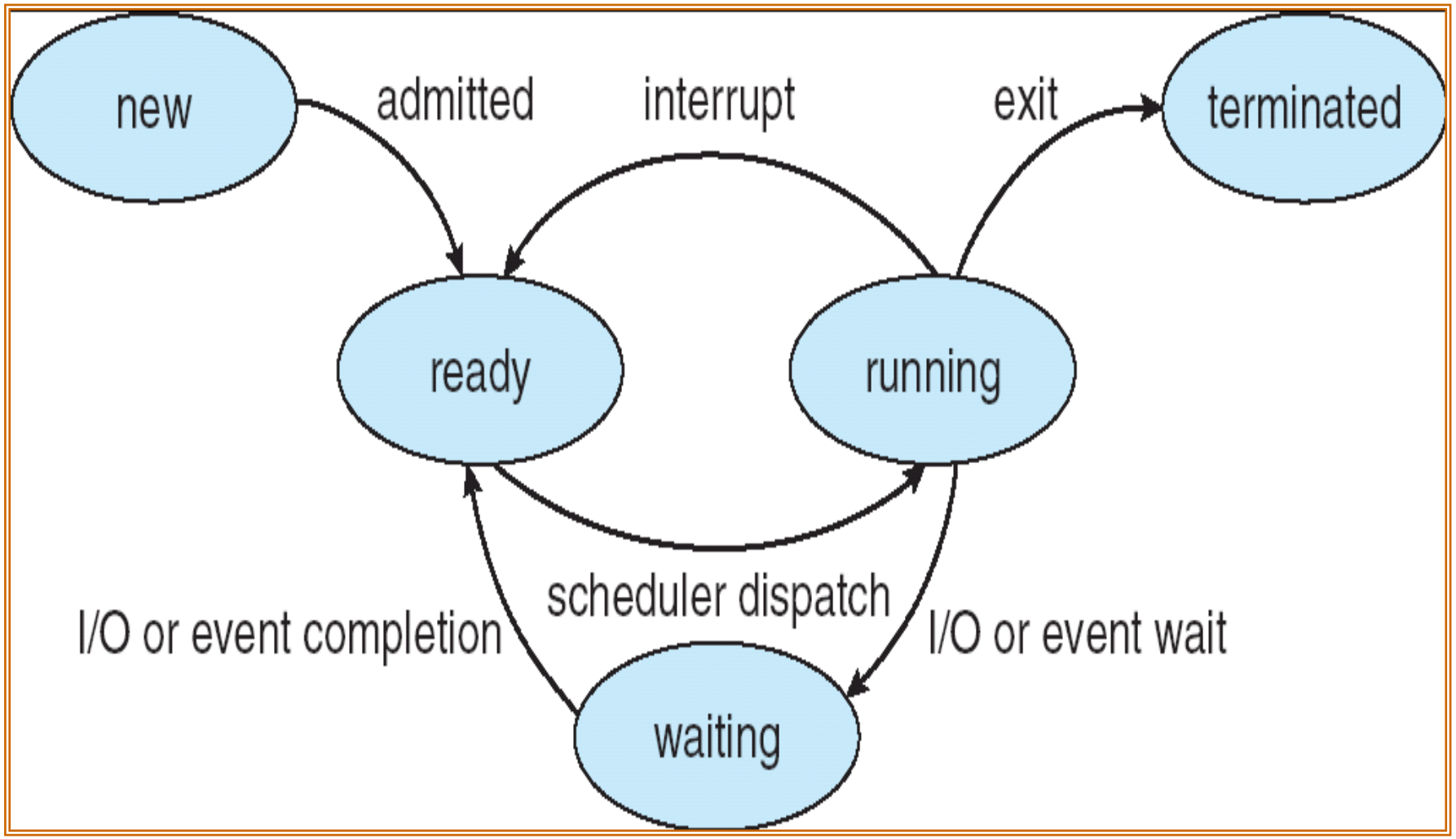


Figure: Process in the Memory

Process States

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **ready**: The process is waiting to be assigned to a processor and is ready to get executed
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur/ waiting for resources
  - **terminated**: The process has finished execution

Process States



Process Control Block (PCB)

Process Control Block (PCB, also called Task Controlling Block) is a data structure in the operating system kernel containing the information needed to manage a particular **process**.

The PCB is "the manifestation of a **process** in an operating system".

Process Control Block (PCB)

Information associated with each process:

1. **Process State-** new, ready….etc.
2. **Pointer-** to the parent process
3. **Program Counter-** next instruction to be executed.
4. **Process Number- Unique identification** number in OS
5. **CPU Registers-** Various CPU registers **where process** need to be **stored for execution** for running state.
6. **CPU Scheduling Information-** Process Priority and other info. required for scheduling

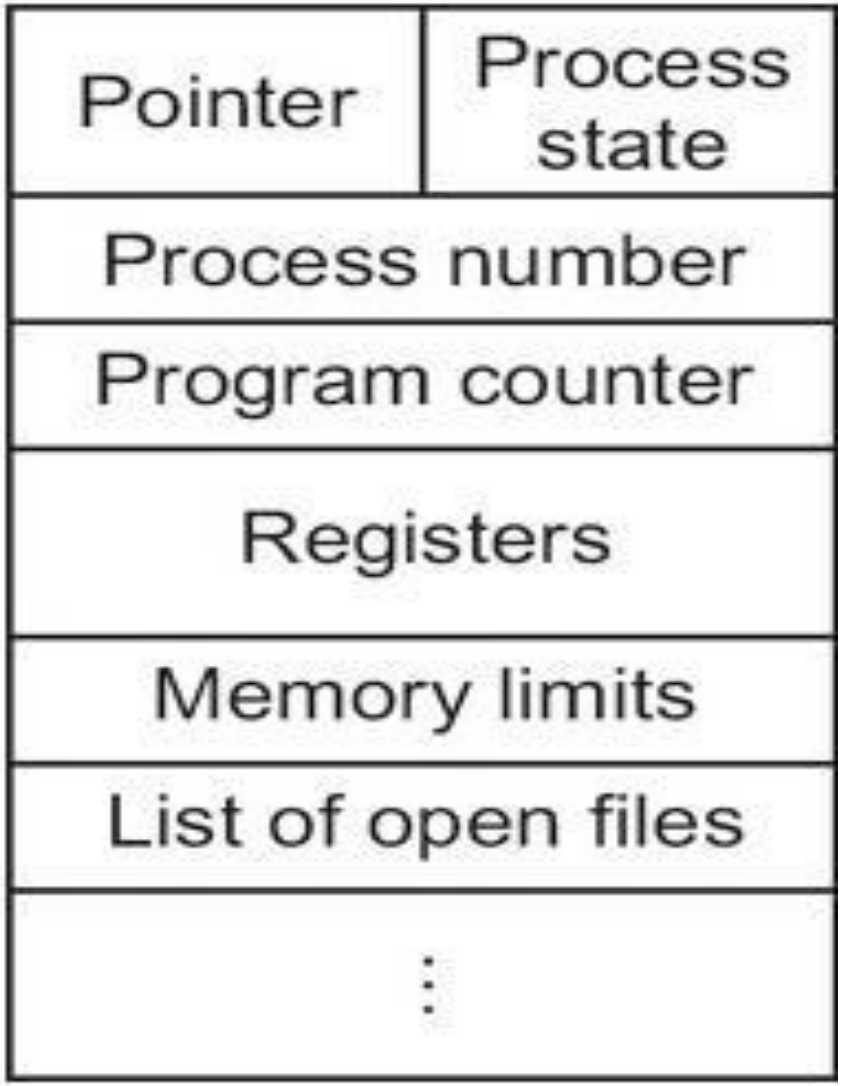
Process Control Block (PCB)

7. **Memory-Management Information-** Registers, Page tables (where process is saved)used by OS

8. **Accounting Information-** For how long CPU and other resources are allocated to process

9. **I/O Status Information-** List of devices allocated to the process

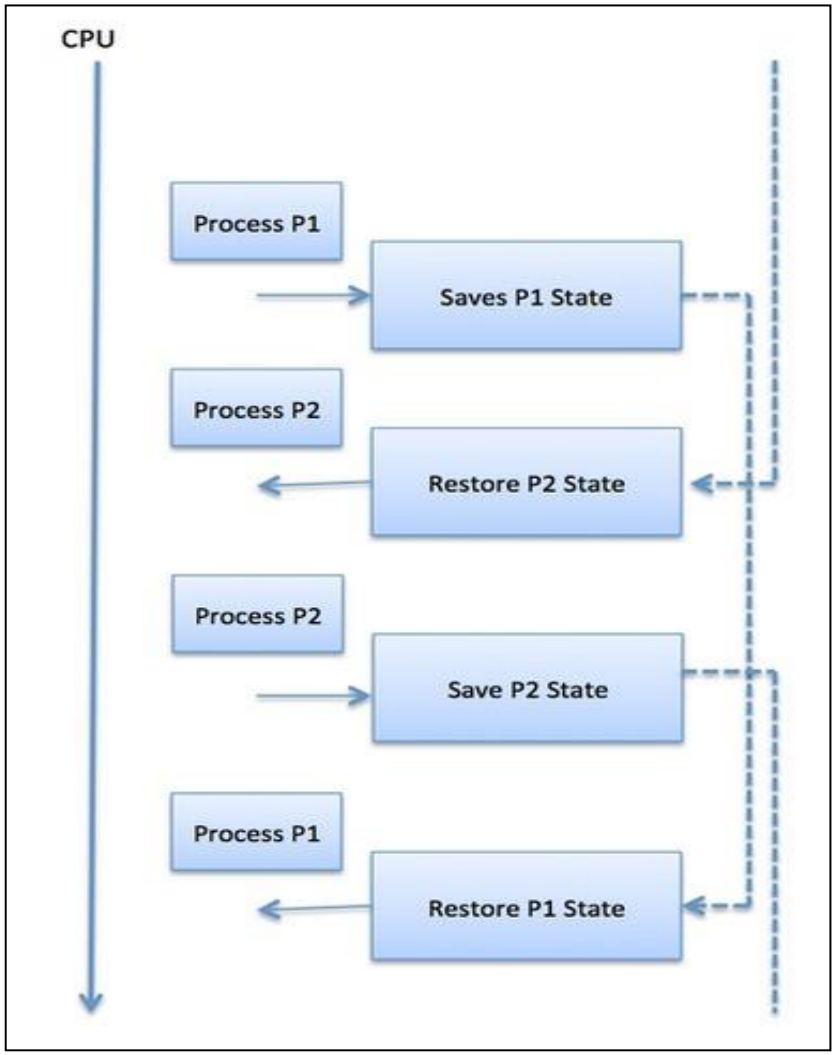
Process Control Block (PCB)



Process Scheduling

Context Switch

When CPU switches to another process, the system must save the state of the old process in the stack and load the saved state for the new process or reload the state of current process from stack.



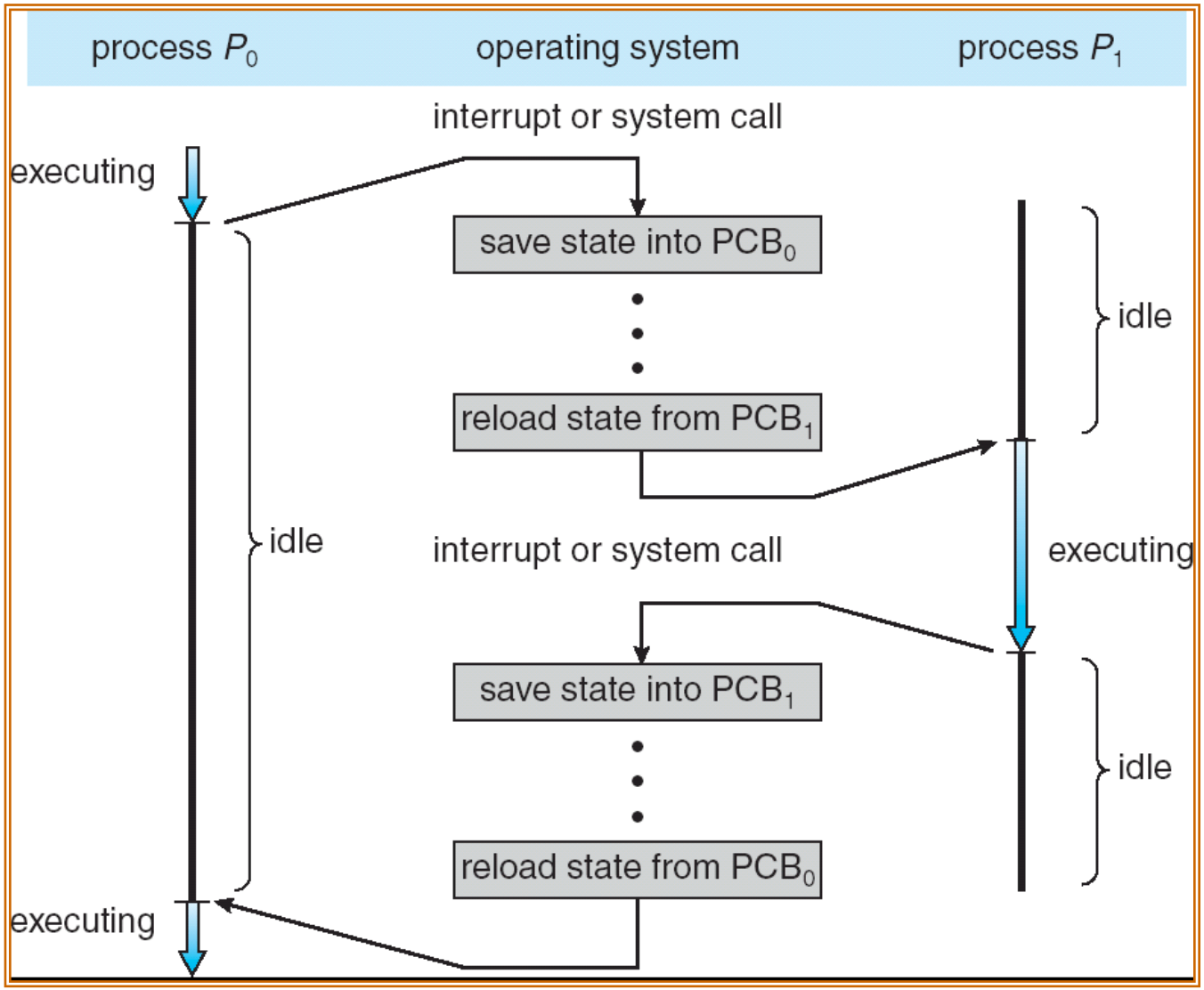
Context Switch

- Switching the CPU to another process requires **saving** the state of the old process and **loading** the saved state for the new process. This task is known as a **context switch**.
- The **context** of a process is represented in the **Process Control Block(PCB)** of a process;
- it includes the value of the CPU registers, the process state and memory-management information.
- When a context switch occurs, the Kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

Context Switch

- Context switch time is **pure overhead**, because the **system does no useful work while switching**.
- Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied
- Typical speeds range from 1 to 1000 microseconds.
- Context switching is expensive:
  - Direct Cost: No. of cycles for load and store instructions
  - Indirect Cost: When process run a lot of its data is stored on processor cache.
- Must limit the frequency of context switching.

CPU Switch From Process to Process



Process Scheduling

- The activity of determining which process in the ready state should be moved to the running state is known as Process Scheduling.

- The prime aim of the process scheduling system is:
  - To keep the CPU busy all the time and to deliver minimum response time for all programs.
  - For achieving this, the scheduler must apply appropriate rules for swapping processes.

Process Scheduling

Schedulers fall into one of the two general categories:

- **Non pre-emptive scheduling:** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive scheduling:** When the operating system decides to favor another process, pre-empting the currently executing process.

Process Scheduling

- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Scheduling Queues

- The OS maintains all PCBs in Process Scheduling Queues.
- The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue.
- When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

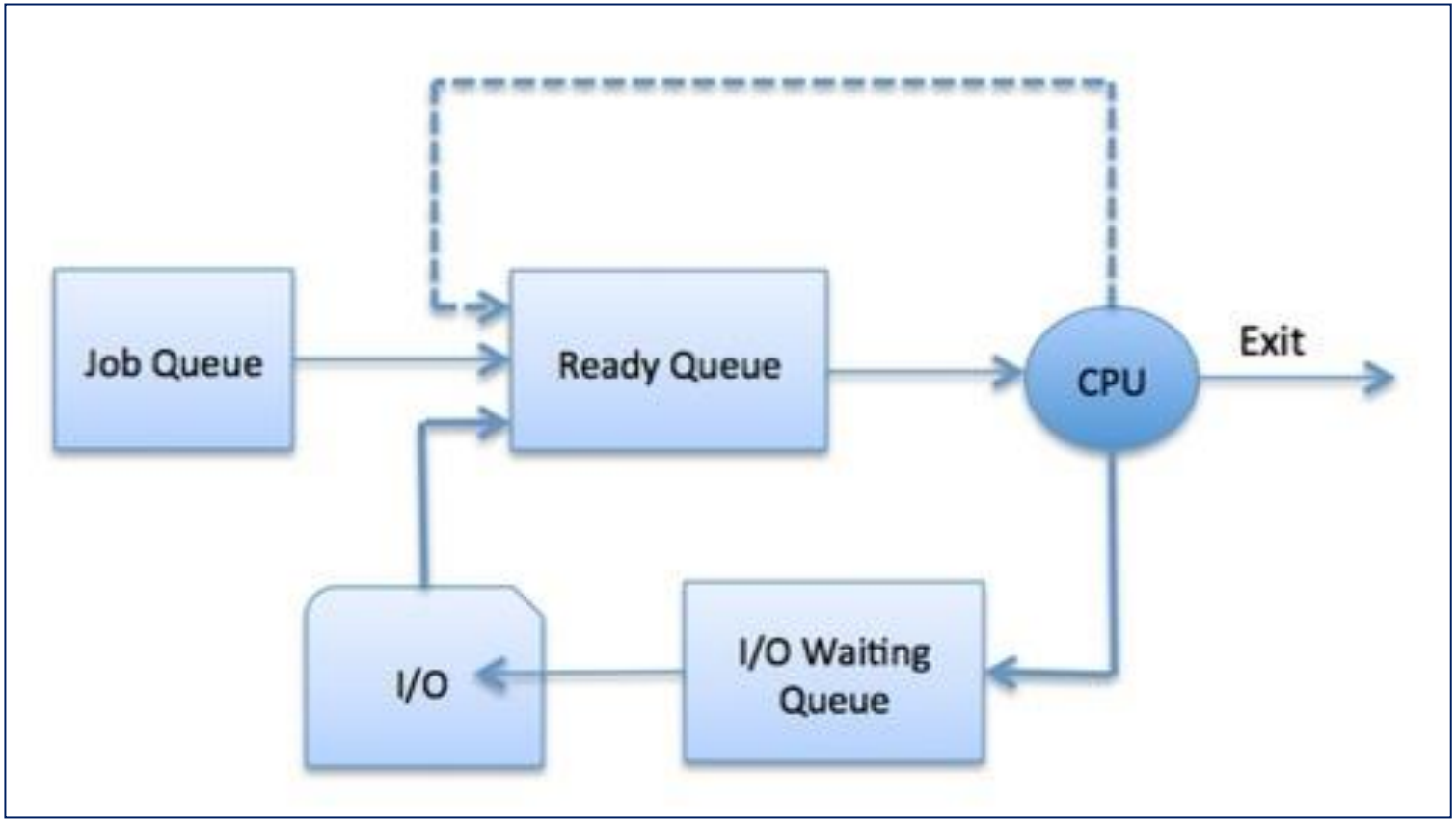
Process Scheduling Queues

- **Job queue** − This queue keeps all the processes in the system.

- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute.
  - A new process is always put in this queue.

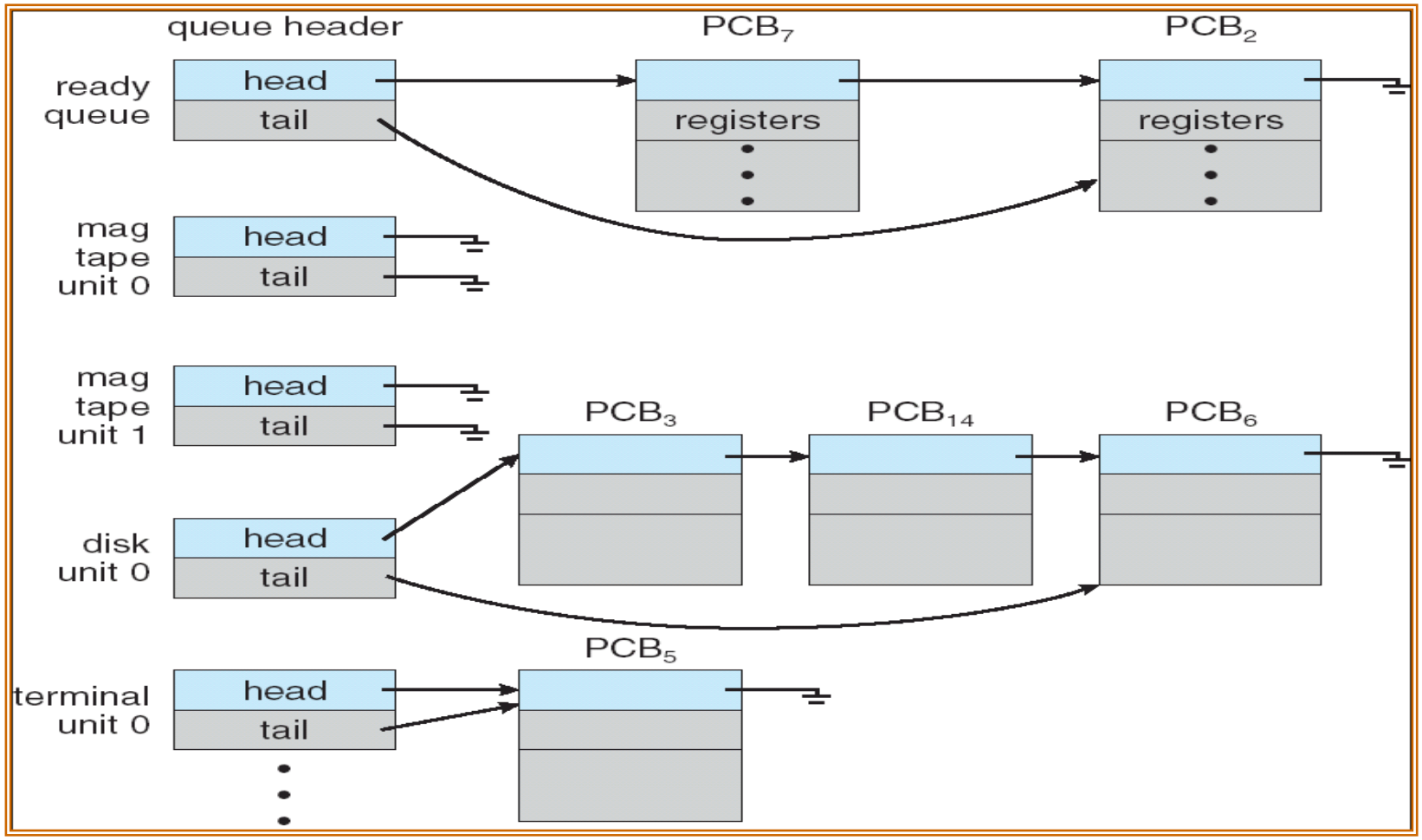
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.

Process Scheduling Queues

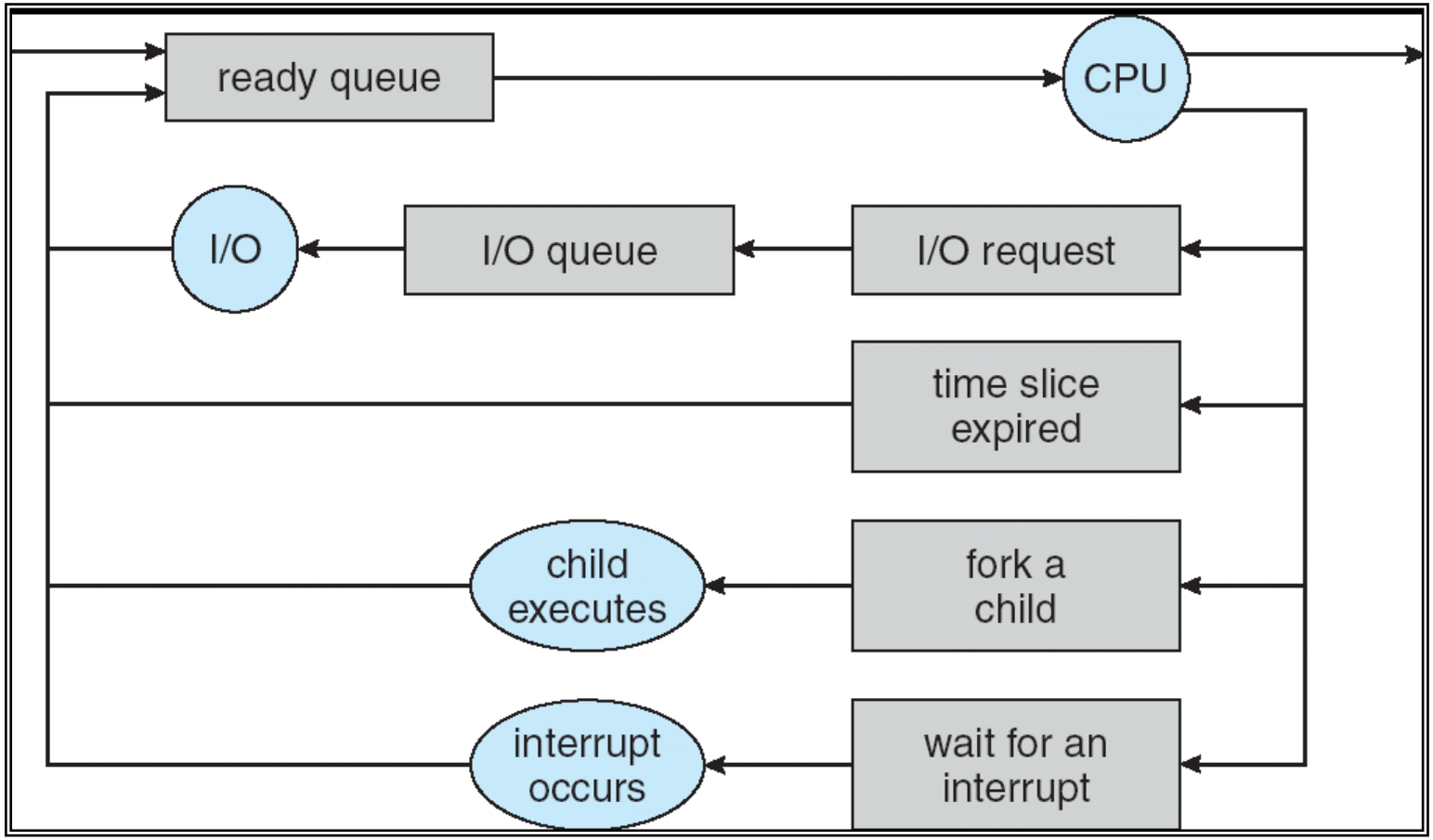


Ready Queue And Various I/O Device Queues

Ready Queue is implemented using Linked List



Queuing Diagram



Schedulers

- Schedulers are special system software which handle process scheduling in various ways.
- Their main task is to select the jobs to be submitted into the system and to decide which process to run.

Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler / Job Scheduler

- It **determines which processes are admitted** to the system for processing.
- Selects processes from pool (disk) and bring them into the ready queue
- It **selects processes from the queue and loads them into memory** for execution.
- When a process changes the state from new to ready, then there is use of long-term scheduler.
- It controls **Degree of Multiprogramming (DoM)**
- **DoM:** The degree of multiprogramming describes the maximum number of processes that a single-processor system can accommodate efficiently.

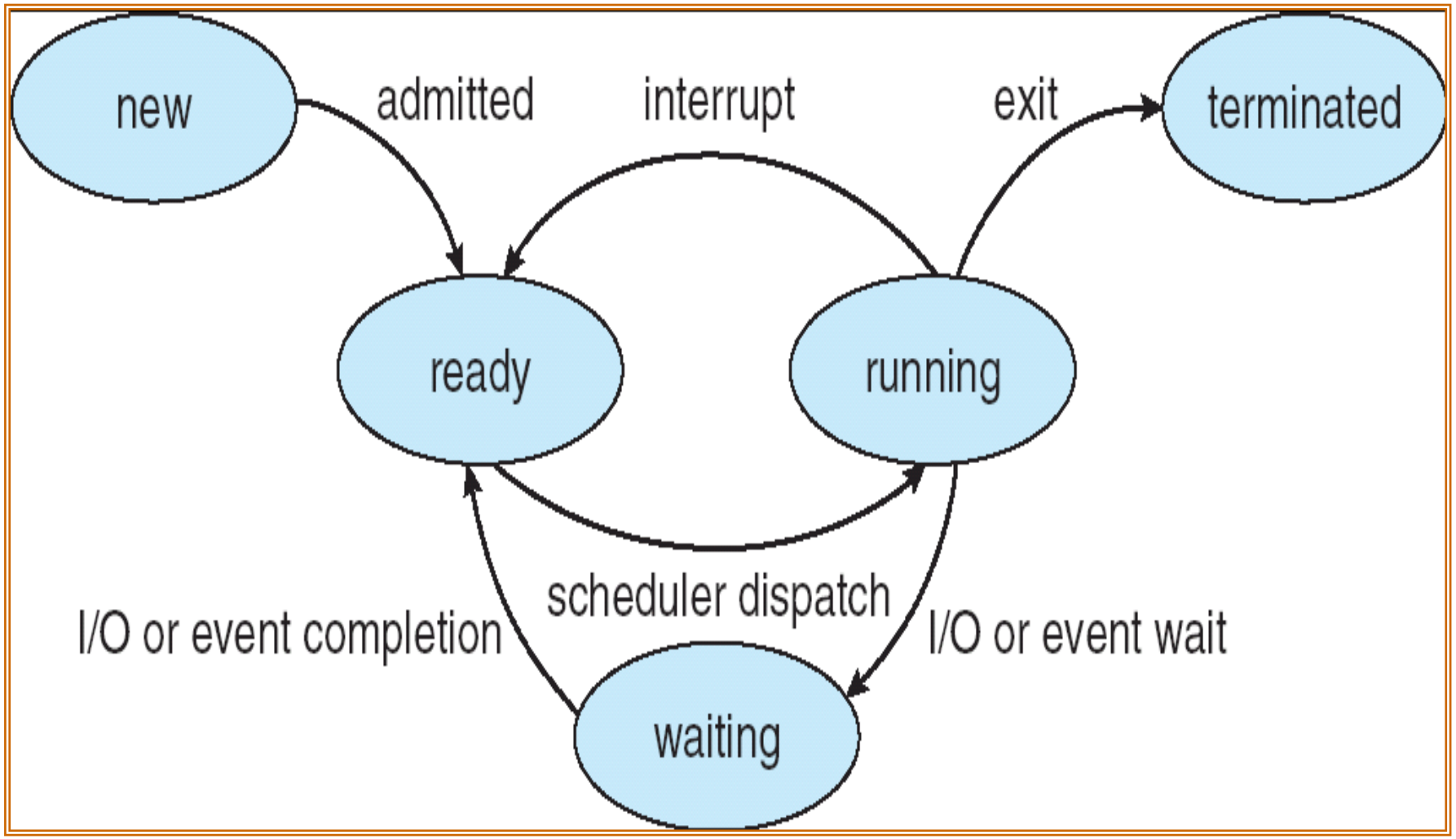
Long Term Scheduler / Job Scheduler

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor (CPU) bound.

- **I/O BOUND:** that spends its more time in doing I/O than computations.
- **CPU BOUND:** that spends its more time doing computations.
- LTS selects a good process.

- **Good Process: (I/O Bound + CPU Bound)**

Process States



Short-Term Scheduler (CPU Scheduler)

- Short-term schedulers, also known as dispatchers.
- Selects which process should be executed next among the processes and allocates CPU to one of them.

(From Ready Queue Selects one process for execution and Allocate CPU **(Running Queue))**

- It must select process for CPU execution frequently.
- Short-term schedulers are faster than long-term schedulers.

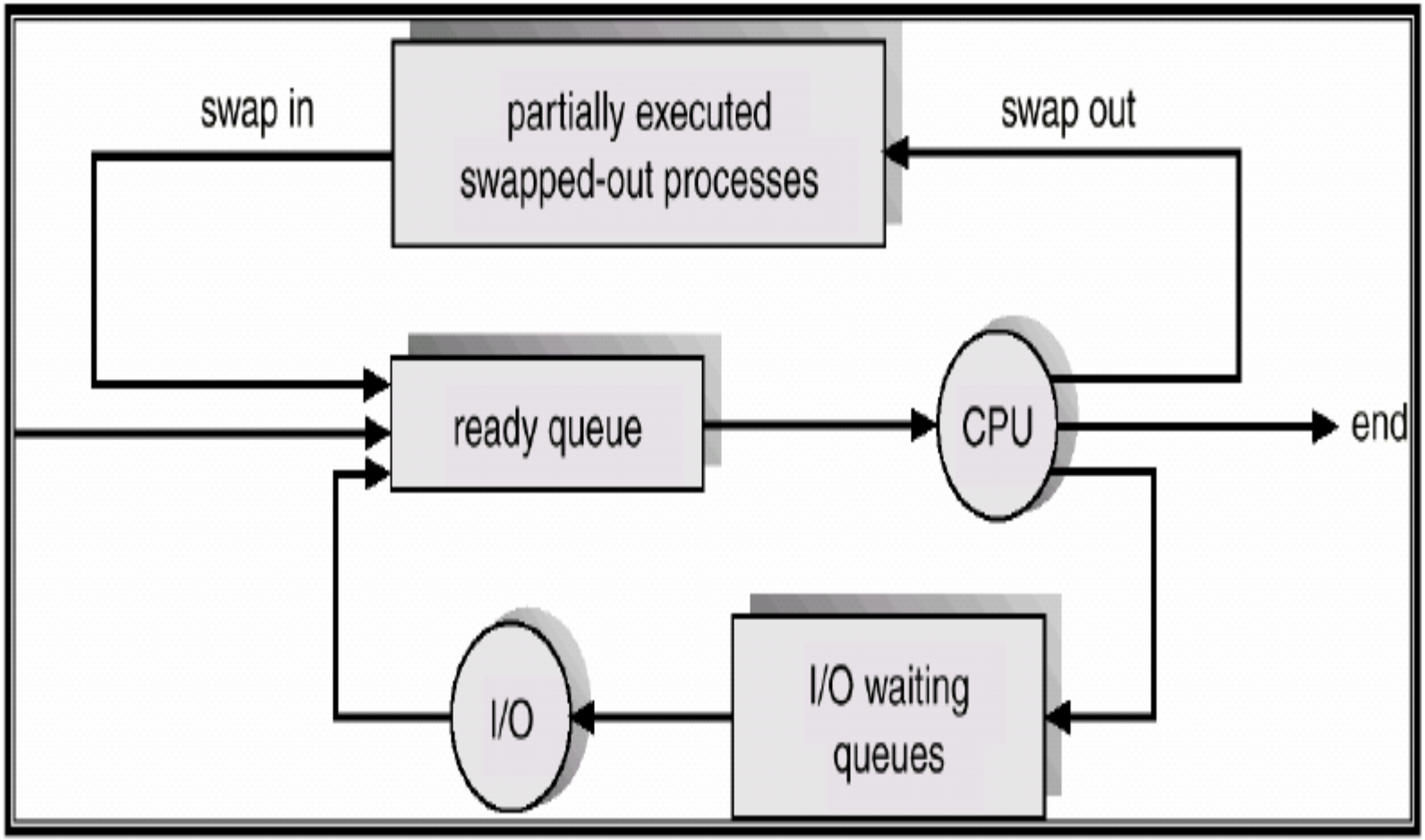
Medium Term Scheduling

- Medium-term scheduling is a part of **swapping**.
- Reduces DOM(Degree of Multiprogramming)
- Processes are created and stored in Main Memory by Long Term Scheduler, But these processes has to wait for their turn to get execute.
- In Main Memory only those processes are kept that require execution.
- When ready queue is empty,

MTS Swap-In and Swap-Out the processes from Main Memory and Secondary Memory.

- Ready + Running + Waiting Queues are in Main Memory

Medium Term Scheduling



Process Operations

Operations on Process

- 1. Process Creation
- 2. Process Termination

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes

- The process which creates other process, is termed the **parent** of the other process, while the created sub-process is termed its **child**.

- Each process is given an integer identifier, termed as process identifier, or PID. The parent PID (PPID) is also stored for each process.

Process Creation (Cont.)

- **Resource sharing**
  - Parent and children **share all resources**
  - **Children share subset** of parent's resources
  - Parent and child **share no resources**
- **Execution Sequence**
  - Parent and children **execute concurrently**
  - Parent waits until children terminate

Process Creation (Cont.)

- **Address Space**

Address space may refer to a range of either physical or virtual **addresses** accessible to a processor or reserved for a process.

  - Child process is duplicate of parent process (same program and data
  - Child process has new program loaded into it

- Each process is identified by its process identifier

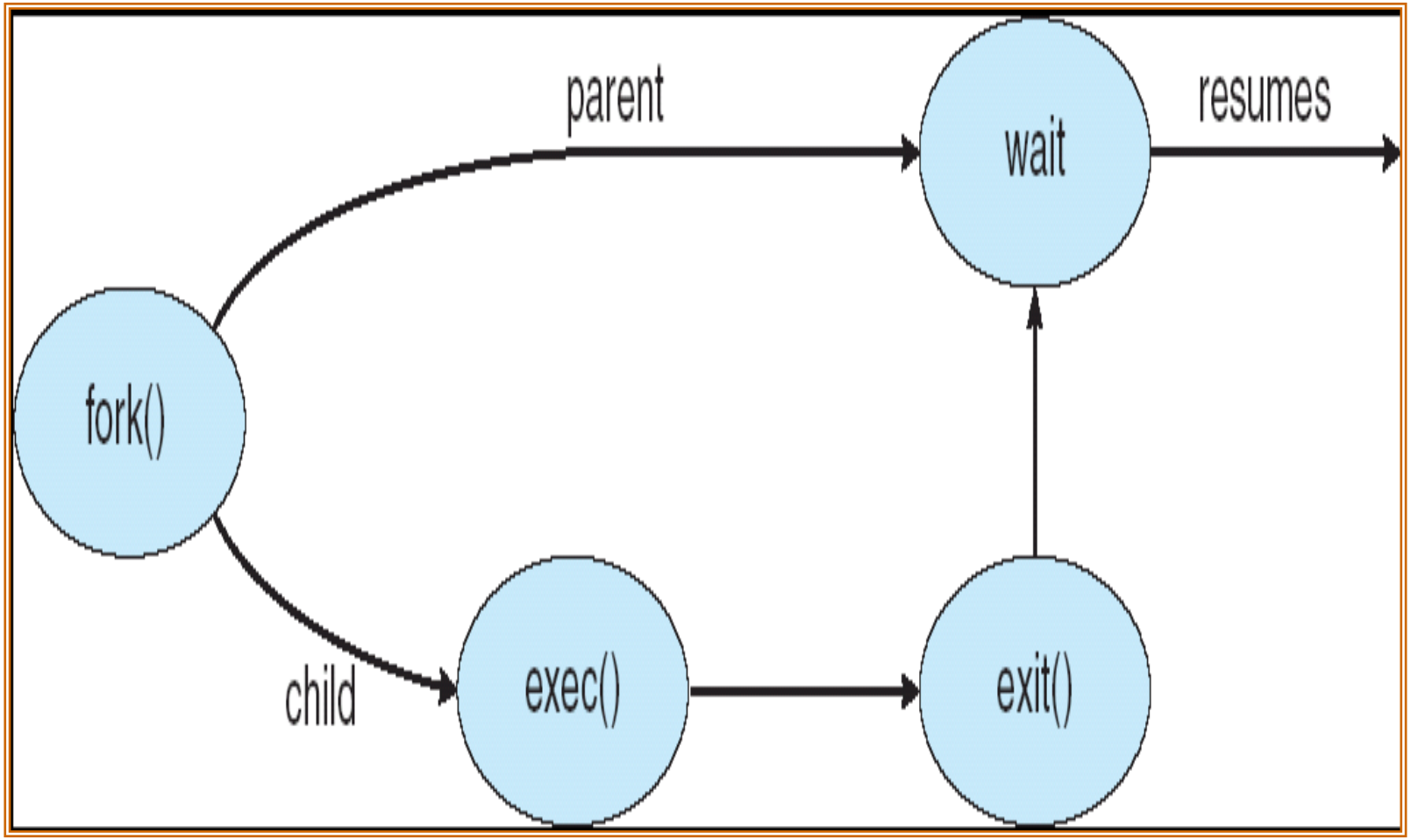
- UNIX examples
  - **fork ()** system call creates new process
  - **exec()** system call used after a **fork** to replace the process' memory space with a new program

fork()

- System call **fork()** is used to create processes.
- It takes **no arguments** and returns a process ID.
- The purpose of **fork()** is to create a ***new*** process, which becomes the *child* process of the caller.
- After a new child process is created, ***both*** processes will execute the next instruction following the ***fork()*** system call.

This can be done by testing the returned value of **fork()**:

- If **fork()** returns a **negative value**, the creation of a child process was **unsuccessful**.
- **fork()** returns a zero to the newly created child process.
- **fork()** returns a positive value, the ***process ID*** of the child process, to the parent.


parent
wait
resumes
fork()
child
exec()
exit()

Process Creation

- There are two options for the parent process after creating the child :

1. **Wait for the child process to terminate before proceeding.**
   - Parent process makes a wait() system call, for either a specific child process or for any particular child process, which causes the parent process to block until the wait() returns.
2. **Run concurrently with the child, continuing to process without waiting.**

There are also two possibilities **in terms of the address space** of the new process:

- The child process is a duplicate of the parent process.
- The child process has a program loaded into it.

Process Termination

- Process executes last statement and asks the operating system to terminate it (by using **exit() system call**)
  - Process' resources are de-allocated by operating system

- **Parent may terminate execution of children processes** (**abort**)

Why?

  - Child has **exceeded allocated resources**
  - **Task** assigned to child is **no longer required**
  - If parent is exiting/Terminated
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*

Process Termination

- **Processes may also be terminated** by the system for a variety of **reasons**,
  - The inability of the system to deliver the necessary system resources.
  - In response to a KILL command or other unhandled process interrupts.
  - A parent may kill its children if the task assigned to them is no longer needed.
  - If the parent does not exit, the system may or may not allow the child to continue without a parent (**orphaned processes** are generally inherited by init, which then proceeds to kill them.)

Process Termination

- **When a process ends, all of its system resources are freed up**. The process **termination status** and execution times are **returned to the parent** if the parent is waiting for the child to terminate,

or eventually returned to init if the process already became an orphan.

- The processes which are trying to terminate but cannot do so because their parent is not waiting for them are termed **zombies**. These are eventually **inherited by init process as orphans** and killed off.