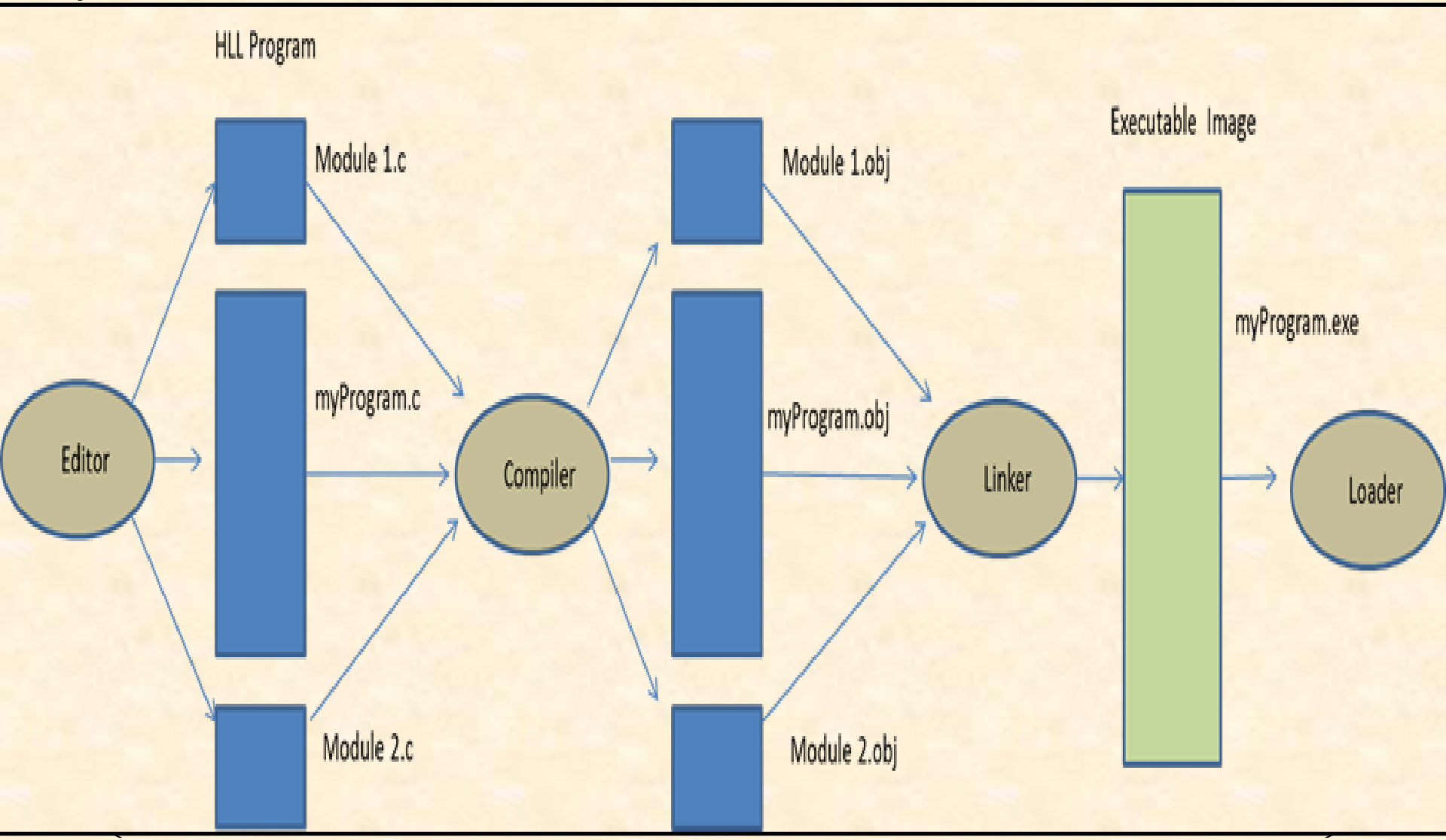
CS-3103 : Operating Systems : Sec-A (NB) : Process

Let's learn about process….

- Process Concept – what is a process, Relationships between Processes and Programs
- Process Scheduling
- Operation on Processes
- Cooperating Processes
- Concurrency and Parallelism
- Inter process Communication

- A program written in a high-level language (say C) passes through the following stages before it is loaded in the computer system for execution.



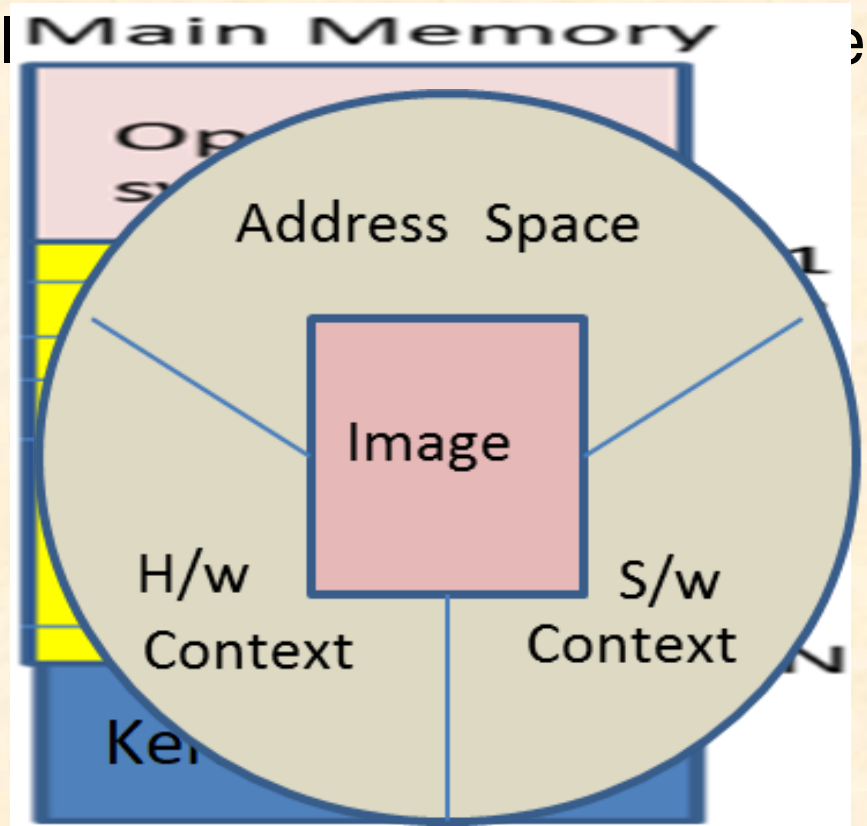
Stages of program development and execution

Program compilation, linking, loading, CPU allocation and a Process is running….

- *Create* and *edit* the program's source code (we call *myProgram.c*) in an editor, notepad/gedit/vi/codeblock….
- *Compile* the program and its associated modules into intermediate files (*myProgram.obj*)
- *Link* the program with its associated modules and library functions to obtain executable code (*myProgram.exe*)
- *Load* '*myProgram.exe*' into the main memory.
- *Allocate* the CPU to run '*myProgram.exe*' .
- *Process*: Execution of '*myProgram.exe*' using resources allocated to it.

Process Concept

- In a multiprogramming environment there are multiple programs that reside in the main memory (MM). Since, each program belongs to a different user, it becomes the responsibility of the operating system to protect one program from another. This is the reason, the OS opens a separate new environment called ***process*** for an incoming executable program.
- The process consists of the foll… program:
  - Hardware context
  - Software context
  - Program's address space

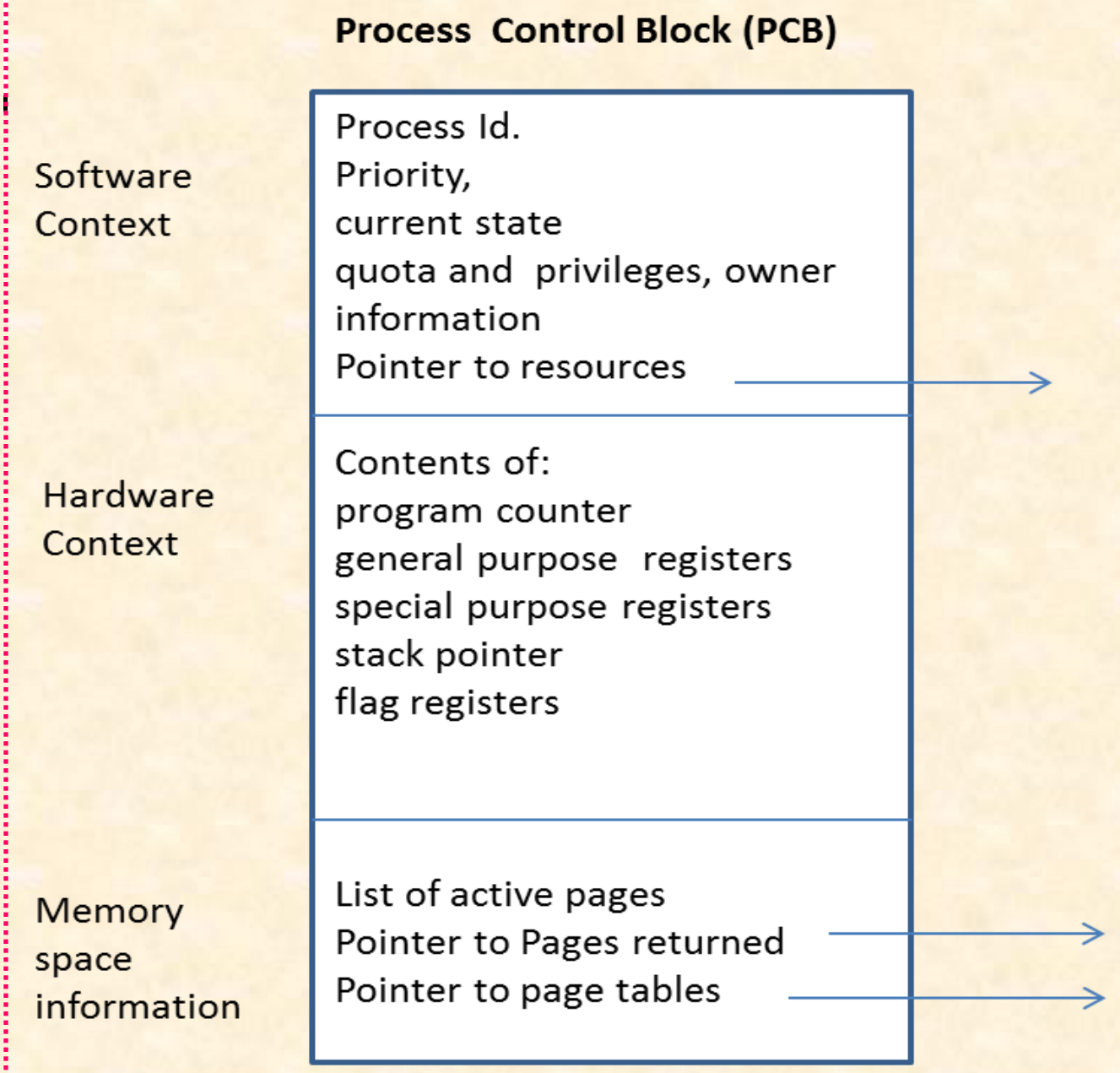


Process Concept

- Hardware context : This is the information about the hardware status of the process. It consists of details on the current contents of the program counter (PC), general purpose registers (GPR), stack pointer (SP), flag registers etc.
- Software context : In S/w context, we have details such as process id (pid ➔ getpid(), getppid()), process priority (priority scheduling), current state of the process (new, ready, running, wait, sleep, defunct, terminated), quota of memory pages, privileges, owner information (user id, uid can be obtained in unix, bash shell…) etc.
- Program's address space : This information is about the active pages of the program image, pages returned by the program, page tables etc.

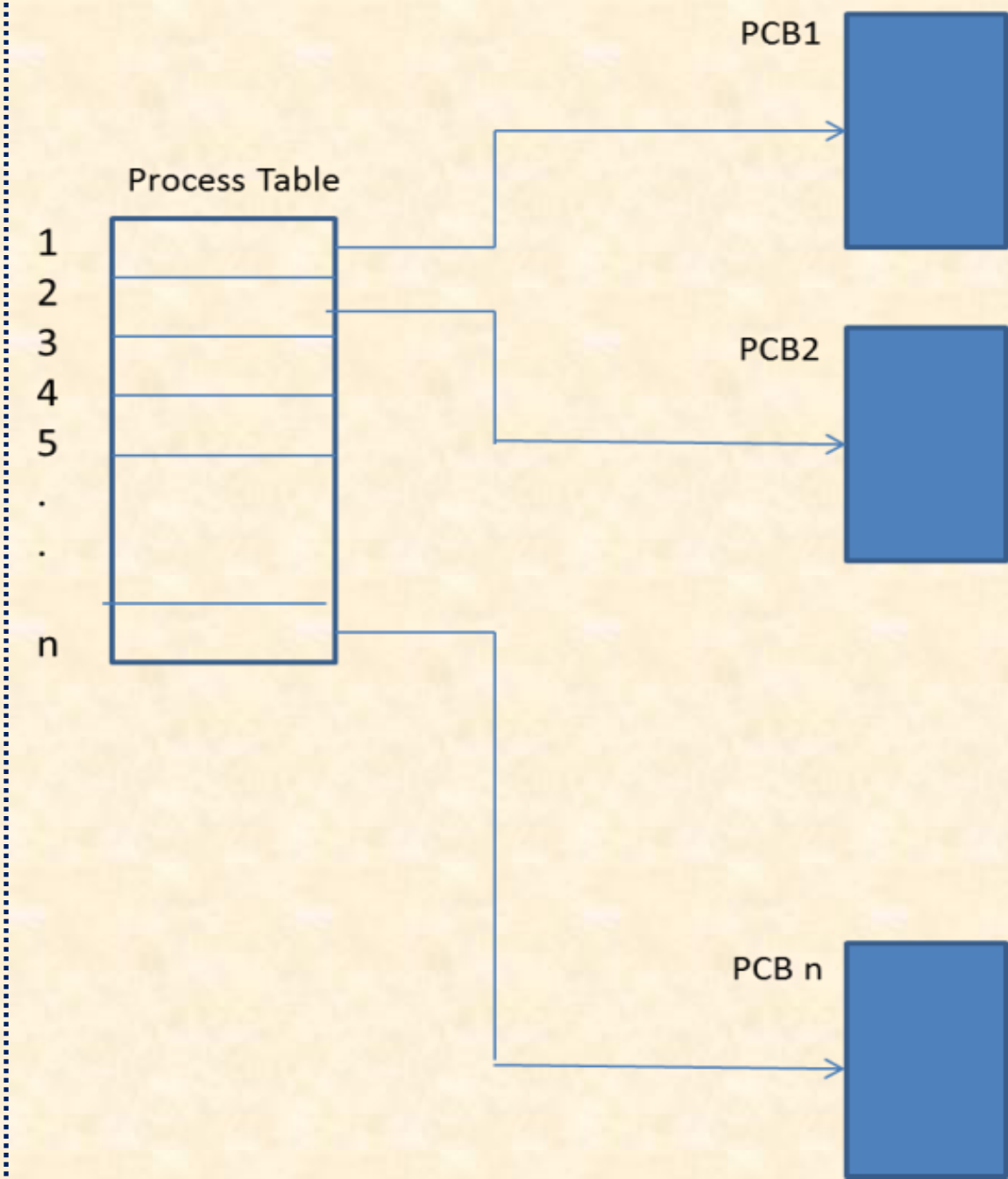
Process Control Block

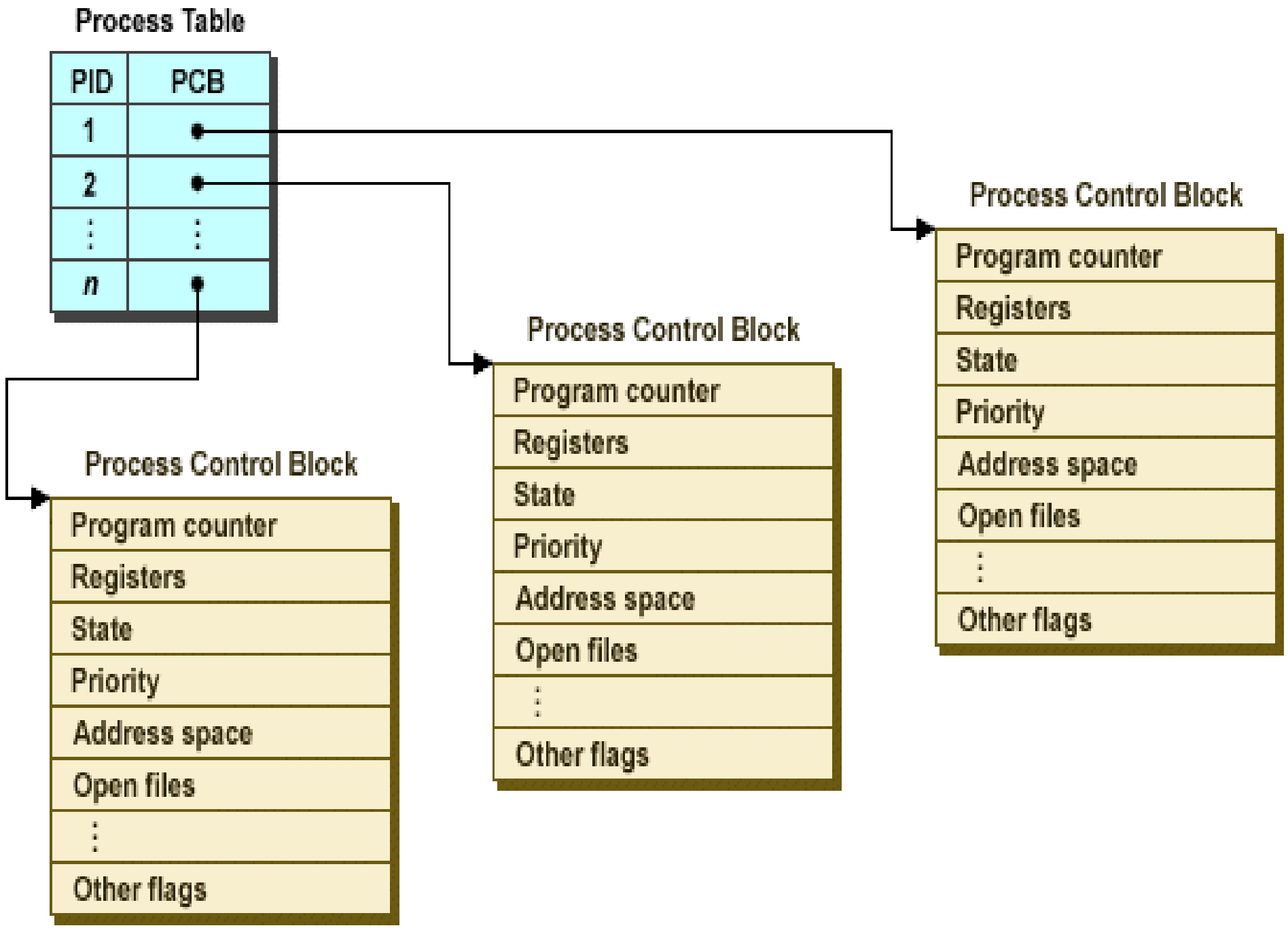
- All the information about the process is stored by the operating system in a data structure called ***Process Control Block (PCB).*** A separate PCB is maintained by the OS for every process opened by OS. ***The process behaves like a unit*** and ***the process that executes, not the program***….



Process Table

- As soon as a program is loaded into MM, OS not only opens a process for this program, but also makes an entry about the process into a **process table**.
- This process table is a global data structure that contains pointers to all the resident processes present in the system.
- Each process entered in this table has a pointer to its PCB.




Process Table
PID
PCB
1
2
n
Process Control Block
Program counter
Registers
State
Priority
Address space
Open files
Other flags
Process Control Block
Program counter
Registers
State
Priority
Address space
Open files
Other flags
Process Control Block
Program counter
Registers
State
Priority
Address space
Open files
Other flags

S.N.	Information & Description
1	**Pointer** Pointer points to another process control block. Pointer is used for maintaining the scheduling list.
2	**Process State** Process state may be new, ready, running, waiting and so on.
3	**Program Counter** Program Counter indicates the address of the next instruction to be executed for this process.
4	**CPU registers** CPU registers include general purpose register, stack pointers, index registers and accumulators etc. number of register and type of register totally depends upon the computer architecture.
5	**Memory management information** This information may include the value of base and limit registers, the page tables, or the segment tables depending on the memory system used by the operating system. This information is useful for deallocating the memory when the process terminates.
6	**Accounting information** This information includes the amount of CPU and real time used, time limits, job or process numbers, account numbers etc.

Process States and State Transitions

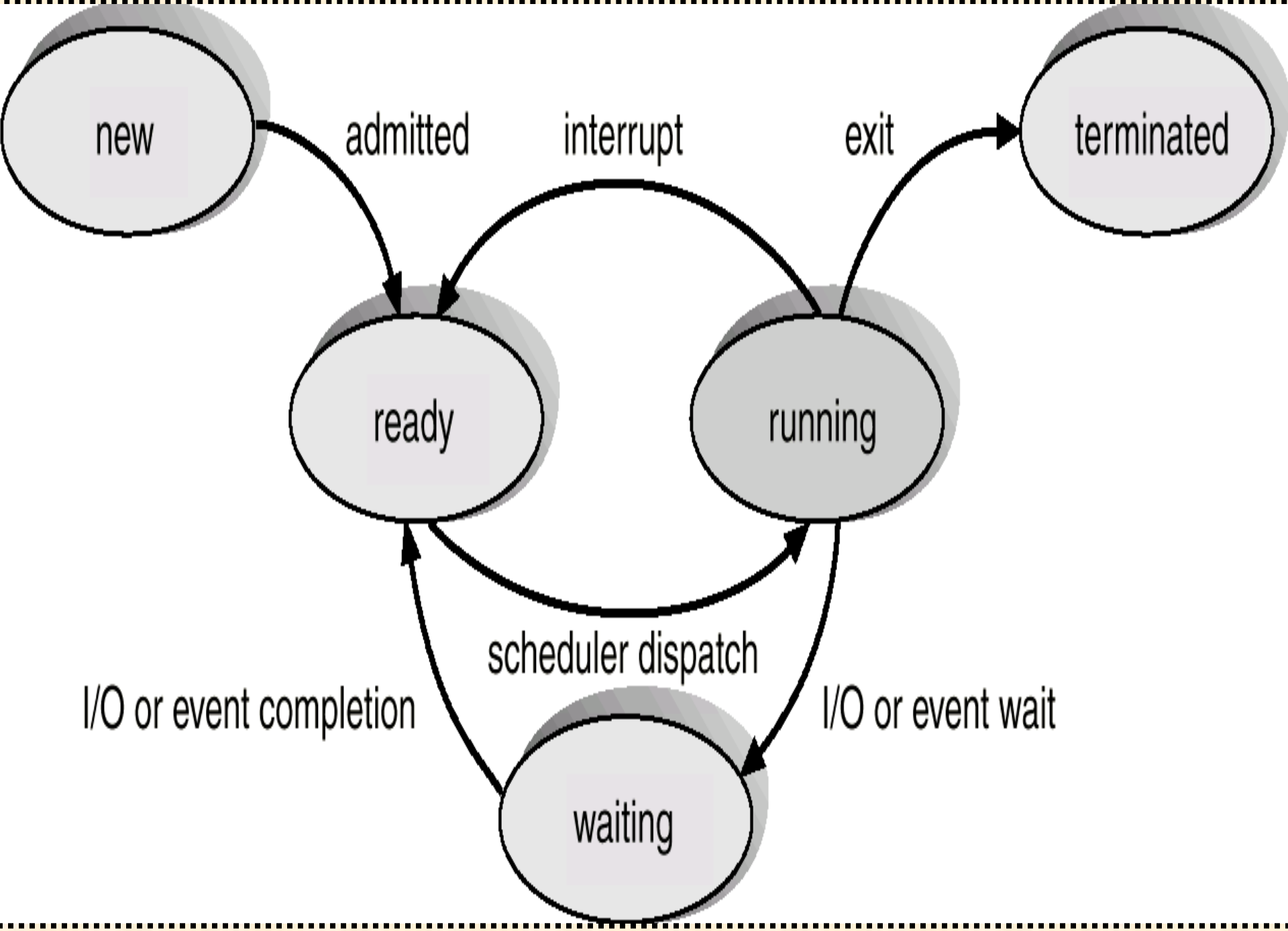
- The loader, scheduler, dispatcher etc. are important components of an OS.
- The loader loads the executable programs into the MM.
- ***For every executable program resident in the memory, the OS opens a process***.
- These ***ready to run*** processes are kept in a queue.

- ***Out of the resident processes, one process is selected by the scheduler for execution*** (***run***).

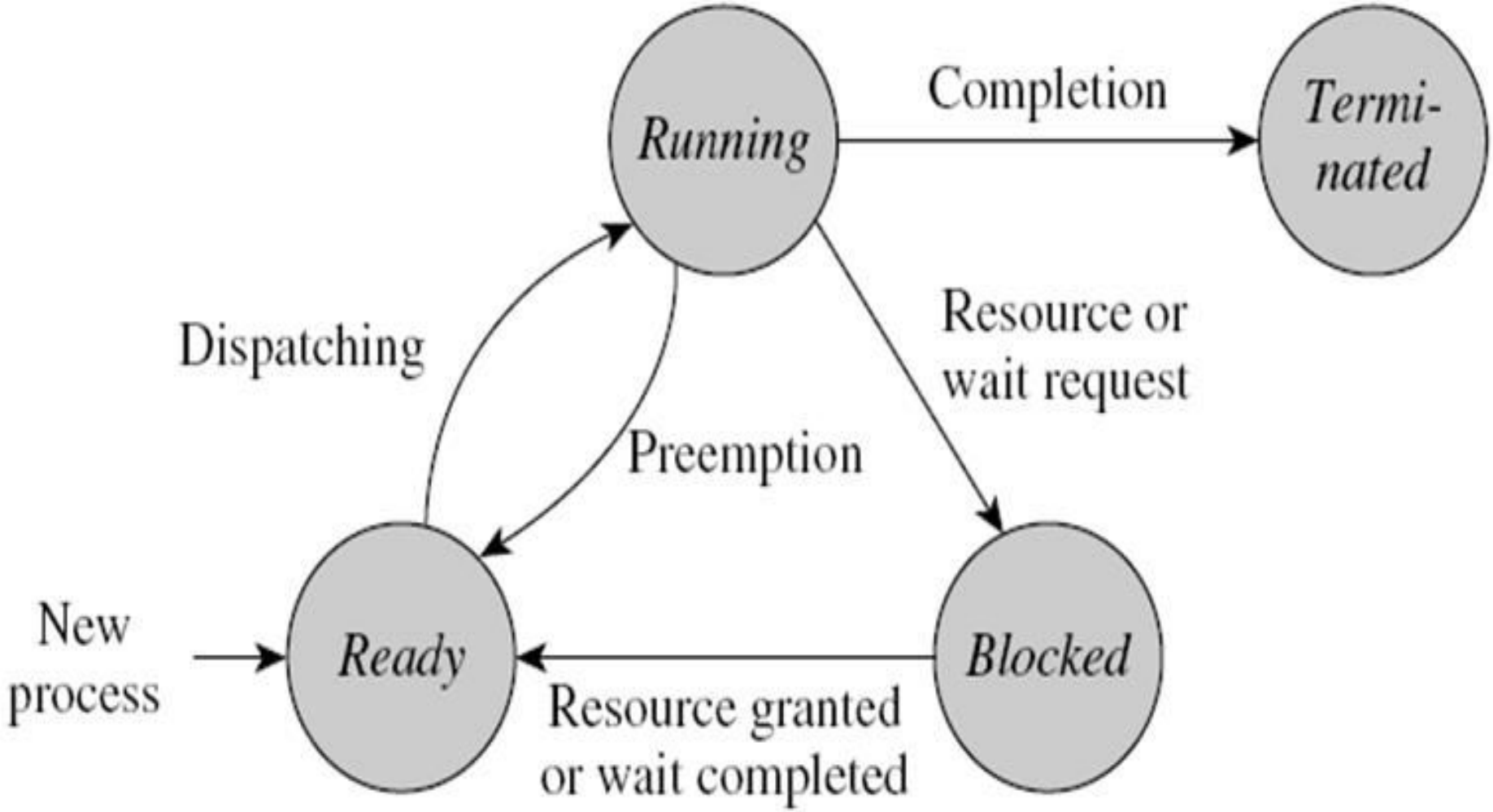
- The selection is based on a scheduling policy.
- The dispatcher allocates the CPU to the selected process.
- If the executing process requests some input/output (I/O) operation, then the process is made to ***wait*** till its requested (I/O) operation is completed by the I/O controller.
- In the meantime, the CPU is allocated to next scheduled process.

Process States and State Transitions

- As a process executes, it changes *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 process.
  - **terminated**: The process has finished execution.


new
admitted
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

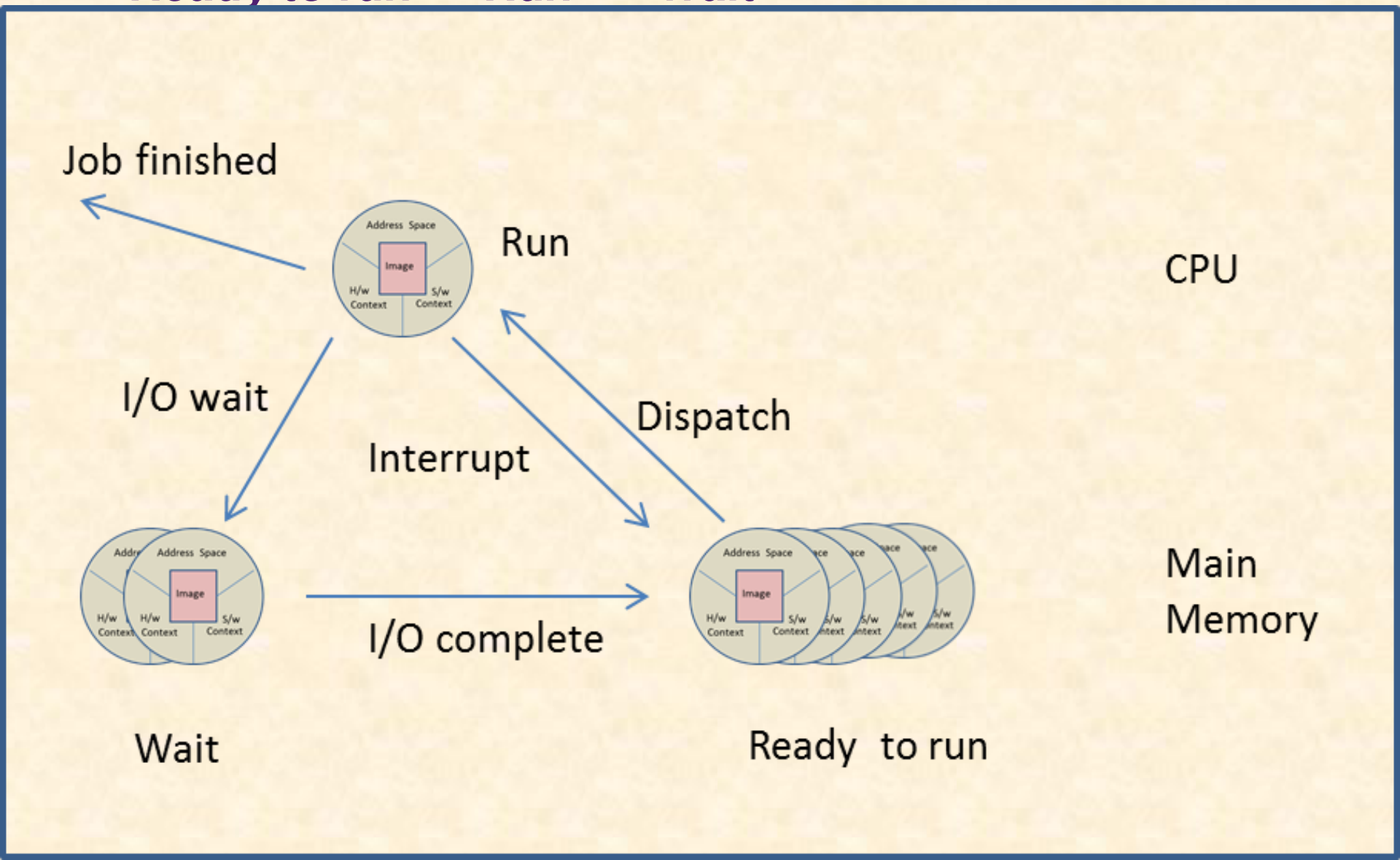
- **A *state transition* for a process is a change in its state**
  - **Caused by the occurrence of some event such as the start or end of an I/O operation**



Fundamental state transitions for a process.

Process States and State Transitions

- A process can be in any one of the following states:
  - ***Ready to run -- Run -- Wait***



A couple of important points to note:

1. Only one process is in the '***run***' state because the CPU can be allocated to only one process at a time.
2. If the above '***running***' process requests an I/O operation, then it joins a '***wait***' queue. A waiting process is also called '***blocked***' process.
3. The waiting process remains in the '***wait***' queue until its required I/O operation is performed by the I/O controller.
4. Example: A process may wait for one of the following reasons:
   a) Wait for the user to enter data from the keyboard. Any program that need a range to calculate fabonacci series.
   b) Wait for a server response. Consider the case when a user clicks a link in a browser (like Google Chrome, Mozilla Firefox), an HTTP request is sent to the concerned server. While this HTTP request is being served, the process (Chrome) has to wait.

A couple of important points to note:

5. As soon as the I/O operation is complete, the process joins the '***Ready to run***' queue.
6. From '***Ready to run***' state, a process (P_i) can be dispatched back to '***run***' state only when it is scheduled so by the CPU scheduler.
7. We have a different case of P_i 's state transition from '***run***' to '***ready to run***' when its allotted time-quanta is over (round-robin scheduling).

A couple of important points to note:

8. P_i leaves CPU for any of the following reasons:

a) **The job is over**
b) **A runtime error, like an arithmetic error or a stack overflow, has occurred.**
c) **Abnormal termination because of the occurrence of an exception.**
d) **Insufficient memory is available in the system.**
e) **An invalid instruction execution has occurred.**
f) **Error in the input/output operation.**

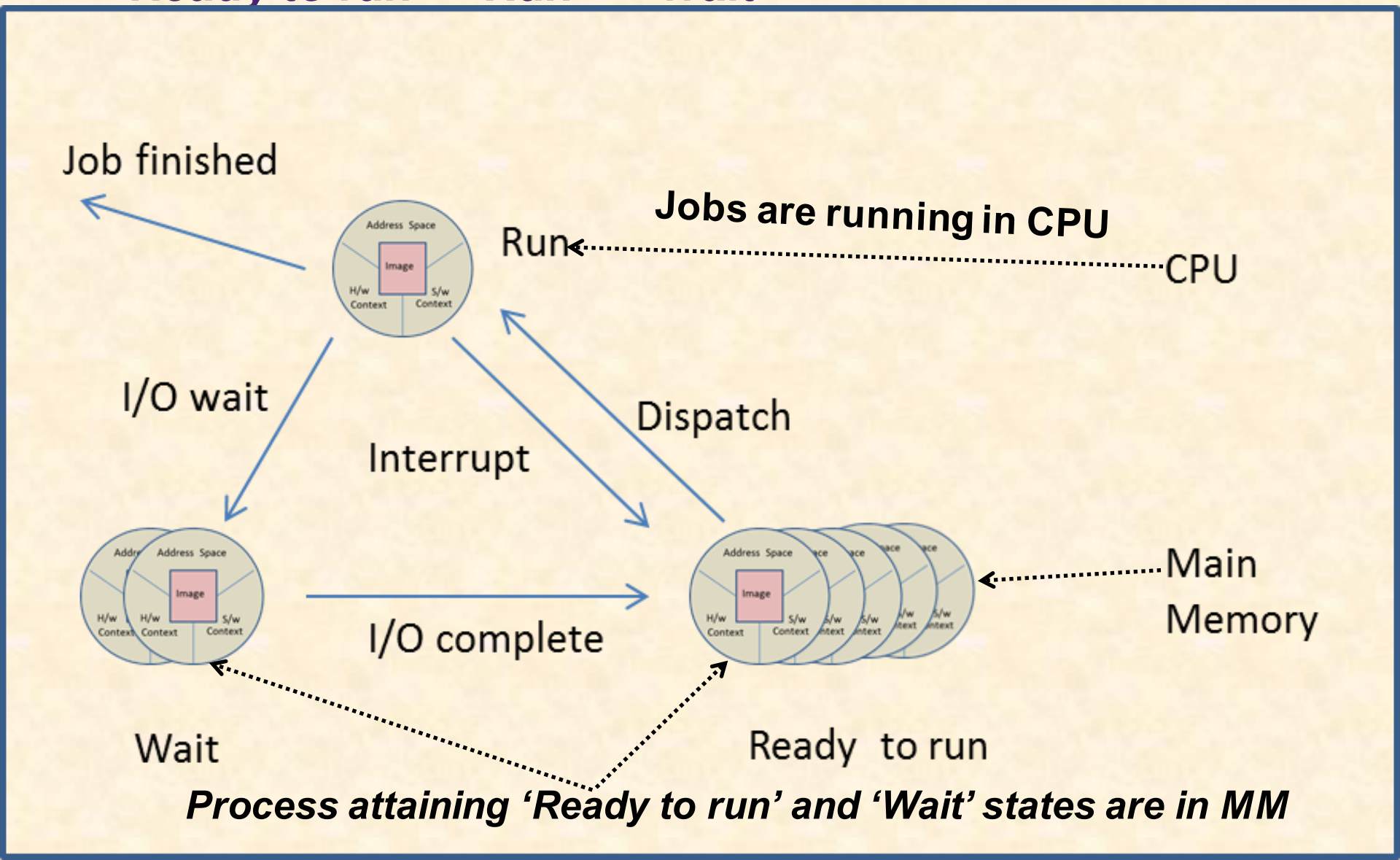
Degree of multiprogramming

Degree of multiprogramming is defined *as the number of active '**ready to run**' processes present in the main memory (MM) of the computer system*.

- Initially, the degree of multiprogramming is ***high*** because almost all the processes present in the main memory are '***ready to run***'.
- However, after some time the degree of multiprogramming decreases, because some of the processes join the '***wait***' queue where they wait for the completion of some I/O.
- When the number of processes in the '***wait***' queue increases significantly, then, in order to increase the degree of multiprogramming, some of the waiting processes are moved to the secondary storage. This is called '***Process swap out***'.
- The swapped out processes are said to be '***blocked suspended***' (Fig. A)
- In addition, to accommodate a higher priority incoming process, a 'ready to run' process may also be swapped out and the '***swapped out***' process is called a '***ready suspend***'.

Process States and State Transitions

- A process can be in any one of the following states:
  - ***Ready to run -- Run -- Wait***



State Diagram of Process States

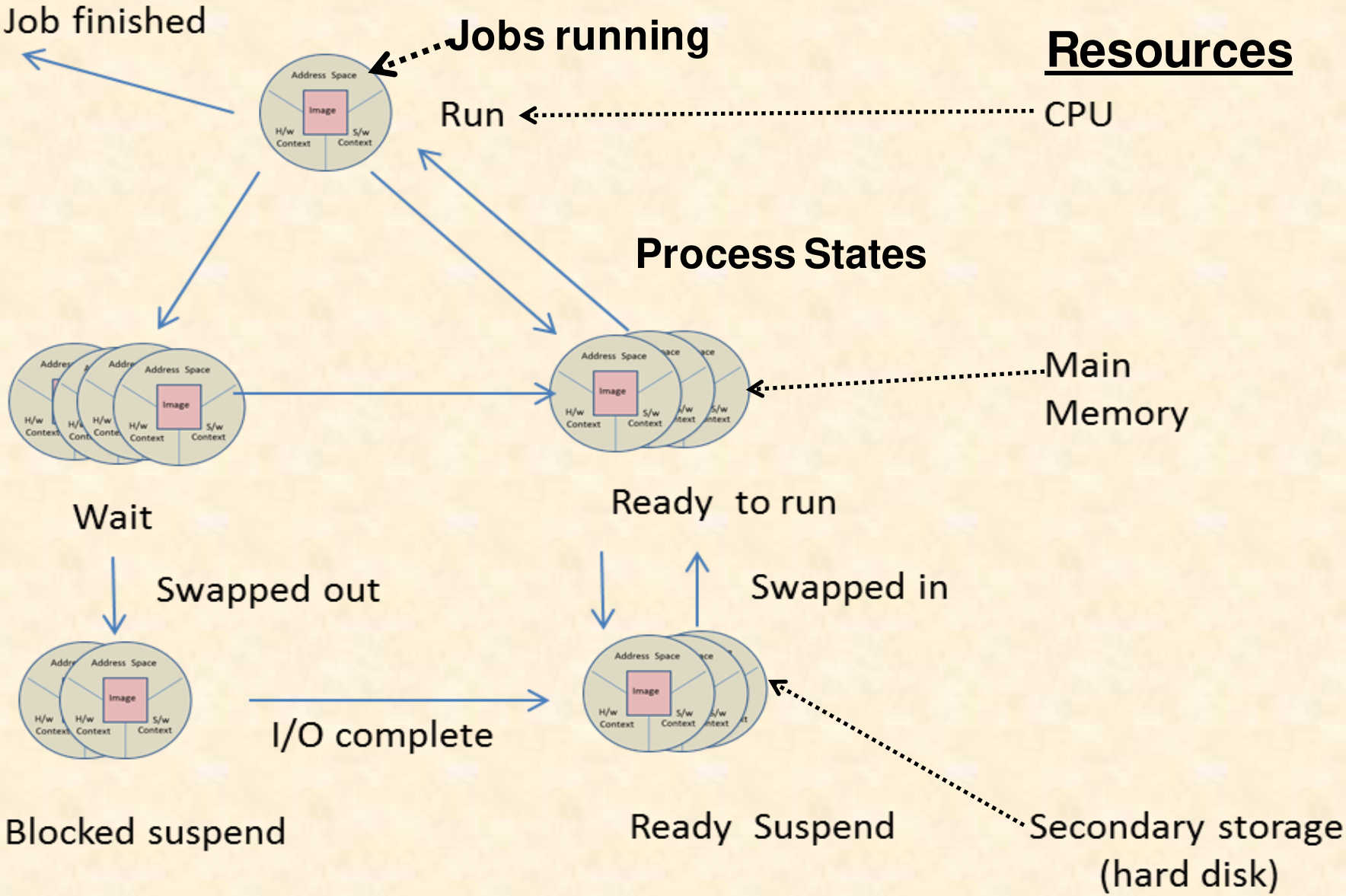


Fig. A

Degree of multiprogramming

- ***Ready→run***: Out of the 'ready to run' processes, one process is scheduled to occupy the CPU and move into the '***run***' state.
- ***Run→Ready***: When the time allotted to a running process is completed, P1 leaves the CPU and joins the '***ready to run***' queue.
- ***Run → Wait***: When a running process asks for an I/O operation, it is made to leave the CPU and the process joins the '***wait***' queue.
- ***Wait → Ready***: As soon as the I/O of a waiting process is completed, it joins the '***ready to run***' state.
- ***Wait → Blocked suspend***: When a waiting or blocked process remains in the main memory for long time, it is sometimes swapped out to a swap area on the secondary storage.
- ***Ready → Ready suspend***: A '***ready to run'*** process is swapped out to secondary storage to make room for higher priority process.
- ***Blocked suspend → Ready suspend***: As and when the I/O operation of a blocked suspend process is completed, P1 joins the '***ready suspend***' queue in the secondary storage.
- ***Ready suspend → Ready***: As soon as some other job terminates, a '***ready suspend***' process is swapped into the '***ready***' queue.


START

Process State Transitions

- A system contains two processes P_1 and P_2

Process State Transitions in a Time-Sharing System

Time	Event	Remarks	New states	
			P_1	P_2
0		P_1 is scheduled	*running*	*ready*
10	P_1 is preempted	P_2 is scheduled	*ready*	*running*
20	P_2 is preempted	P_1 is scheduled	*running*	*ready*
25	P_1 starts I/O	P_2 is scheduled	*blocked*	*running*
35	P_2 is preempted	—	*blocked*	*ready*
		P_2 is scheduled	*blocked*	*running*
45	P_2 starts I/O	—	*blocked*	*blocked*

Concurrent Execution of Programs

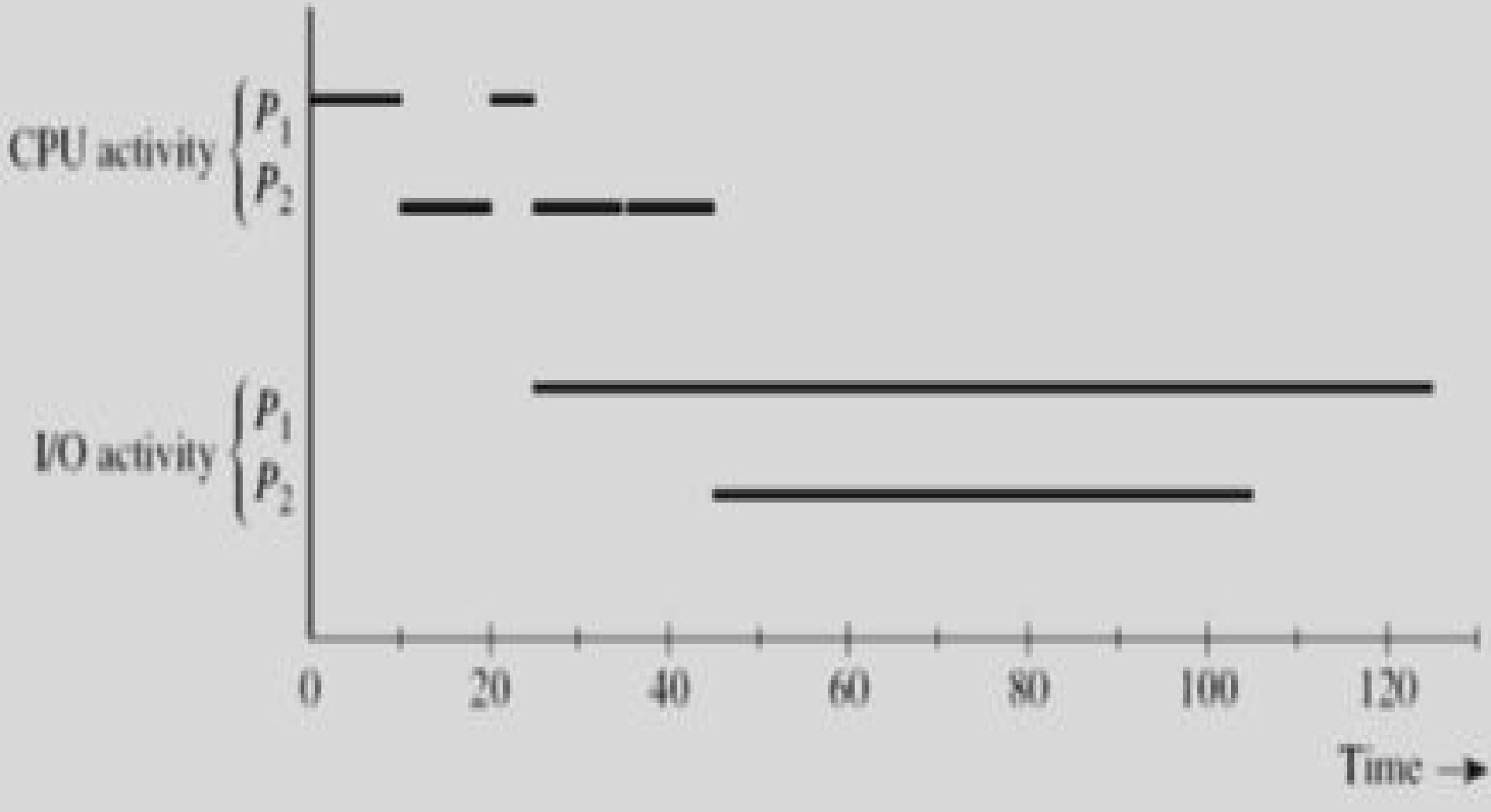
- ***How does concurrency provide any benefit?***
- Answer: **Throughput increases by simultaneous operation of processes on a CPU.**
- Two events or tasks are concu
  are performed at the same tim
  from a parent process creating
  space in MM and running co
  child's termination, control come

Throughput: number of processes completed per unit time. **Turnaround Time**: mean time from submission to completion of process. **Waiting Time**: Amount of time spent ready to run but not running. **Response Time**: Time between submission of requests and first response to the request

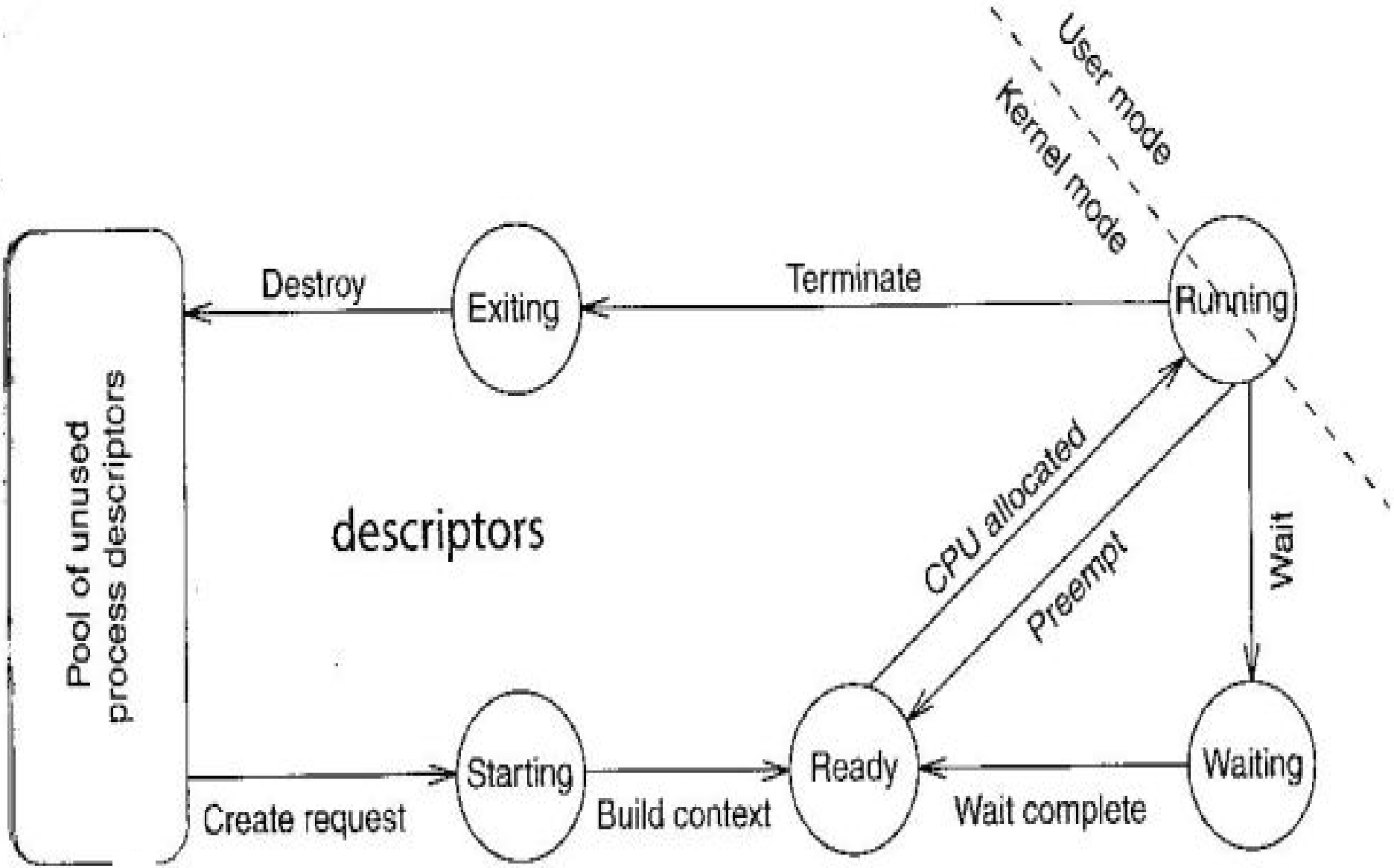
- In ***uniprocessor*** systems, at the most, one process may be running on the CPU while other processes (if any) are in the ready or waiting states.
- In ***multiprocessor*** systems, many processes can simultaneously run on different CPUs.

Concurrent Execution of Programs…

- The objective of ***time-sharing*** is to switch the CPU among processes so frequently that users can interact with each program while it is running.



State transition in uniprocessor system

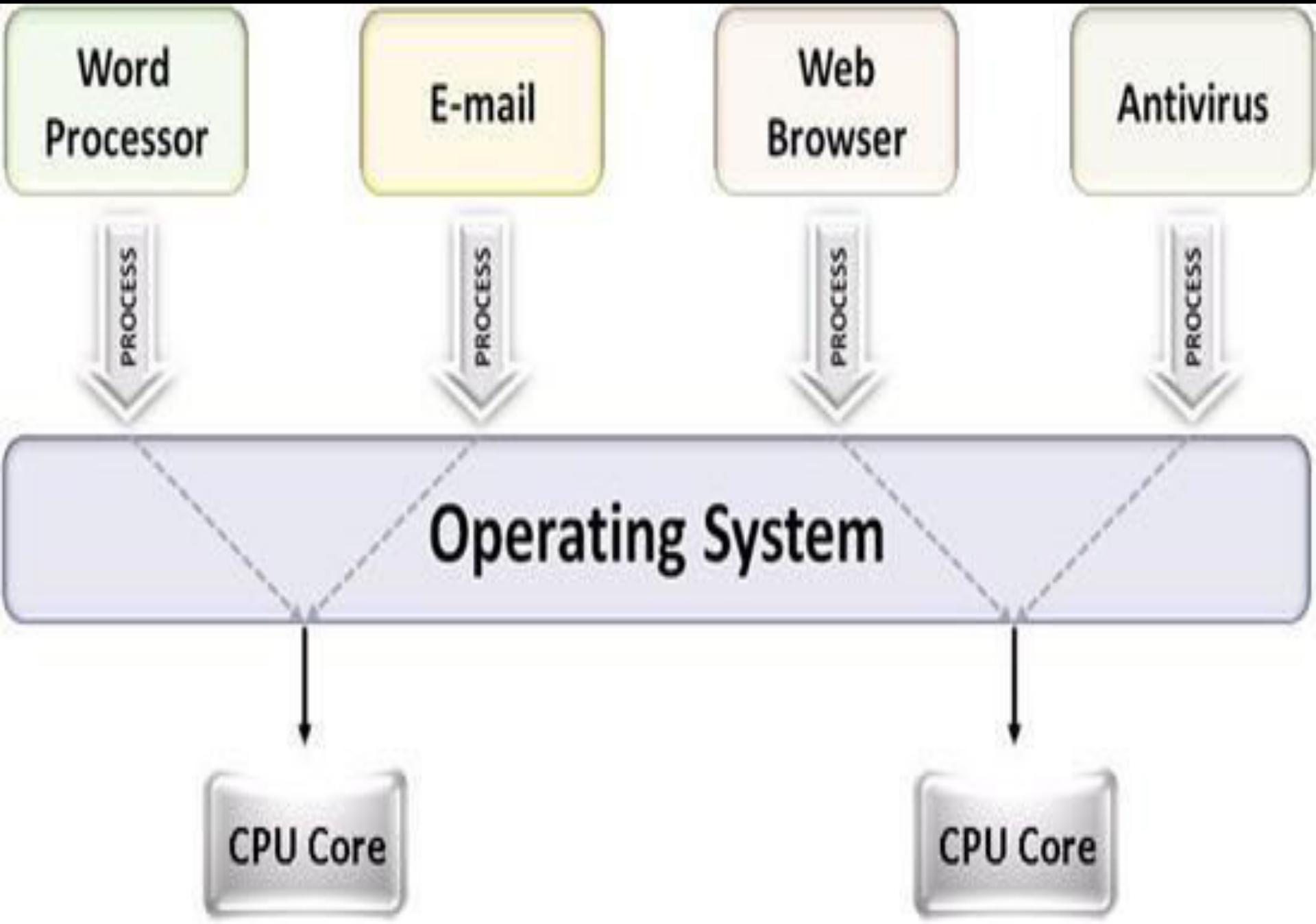


The process-state transition diagram.

Concurrent Execution in Multi-process Systems

- A multi-process system (or multitasking system) is one that executes many processes concurrently.
- The objective of multiprocessing is to have a ***process running on the processor at all times***, doing purposeful work.
- ***Many processes are executed concurrently to improve the performance of the system, and to improve the utilization of system resources such that as the processor, the main memory, disks, printers, network interface cards, etc.***
- The operating system executes processes by switching the processor among them. This switching is called ***context switching***, ***process switching***, or ***task switching***.

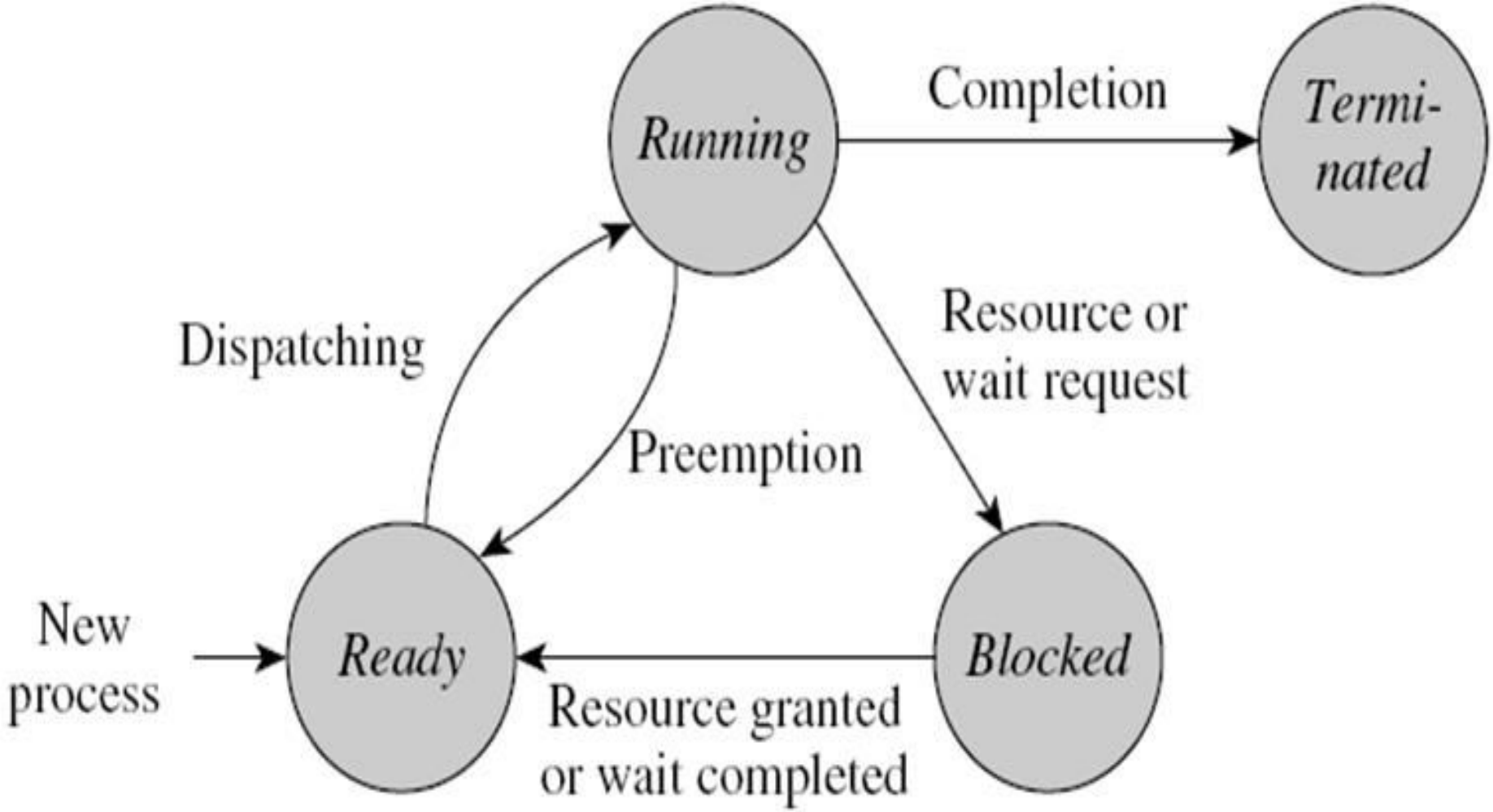
Multi-process Systems…



Context Save, Scheduling, and Dispatching

- **Context save function**:
  - Saves CPU state in PCB, and saves information concerning context
  - Changes process state from *running* to *ready*
- **Scheduling function**:
  - Uses process state information from PCBs to select a *ready* process for execution and passes its id to dispatching function
- **Dispatching function**:
  - Sets up context of process, changes its state to *running*, and loads saved CPU state from PCB into CPU
  - Flushes address translation buffers used by MMU

- **A *state transition* for a process is a change in its state**
  - **Caused by the occurrence of some event such as the start or end of an I/O operation**



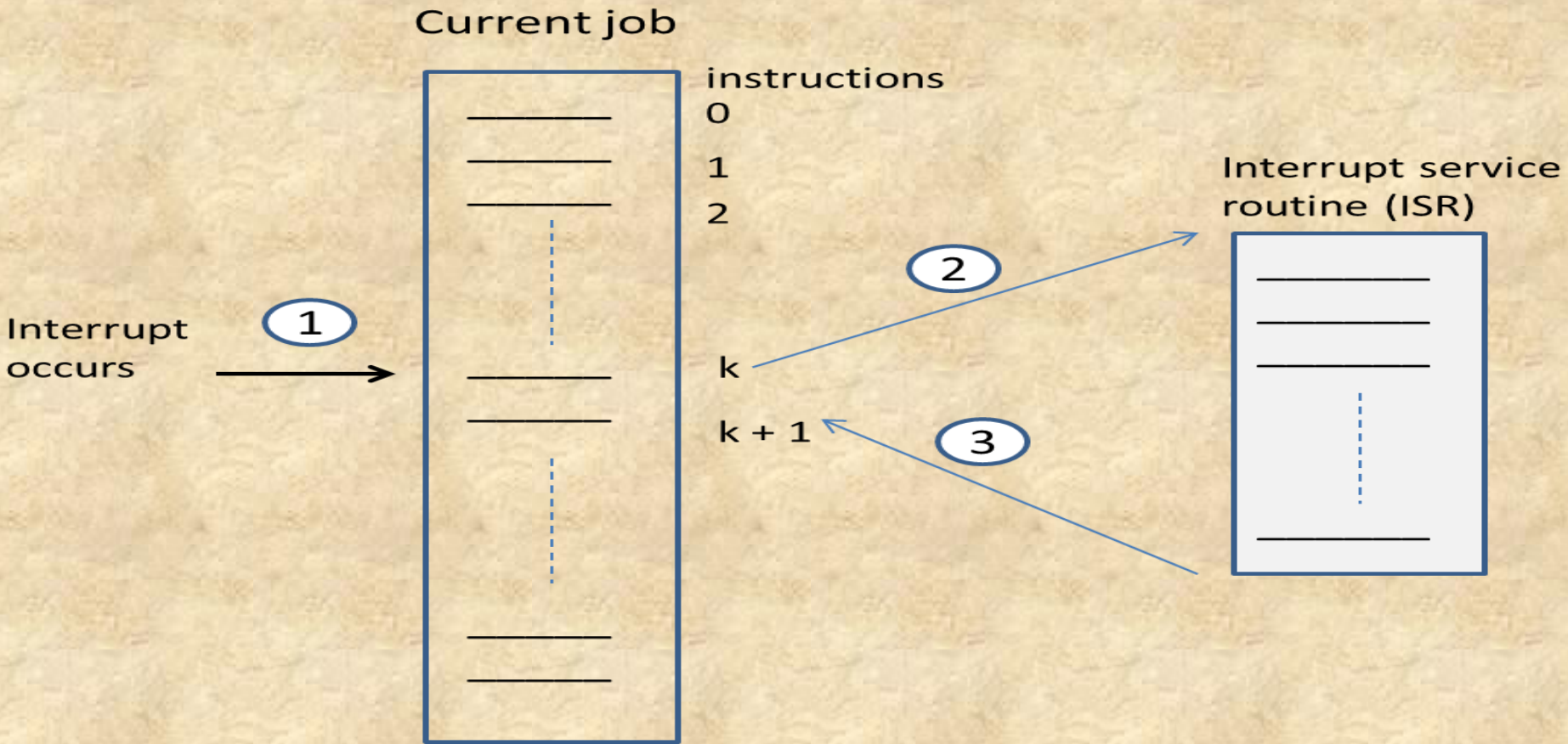
Fundamental state transitions for a process.

Context Switch

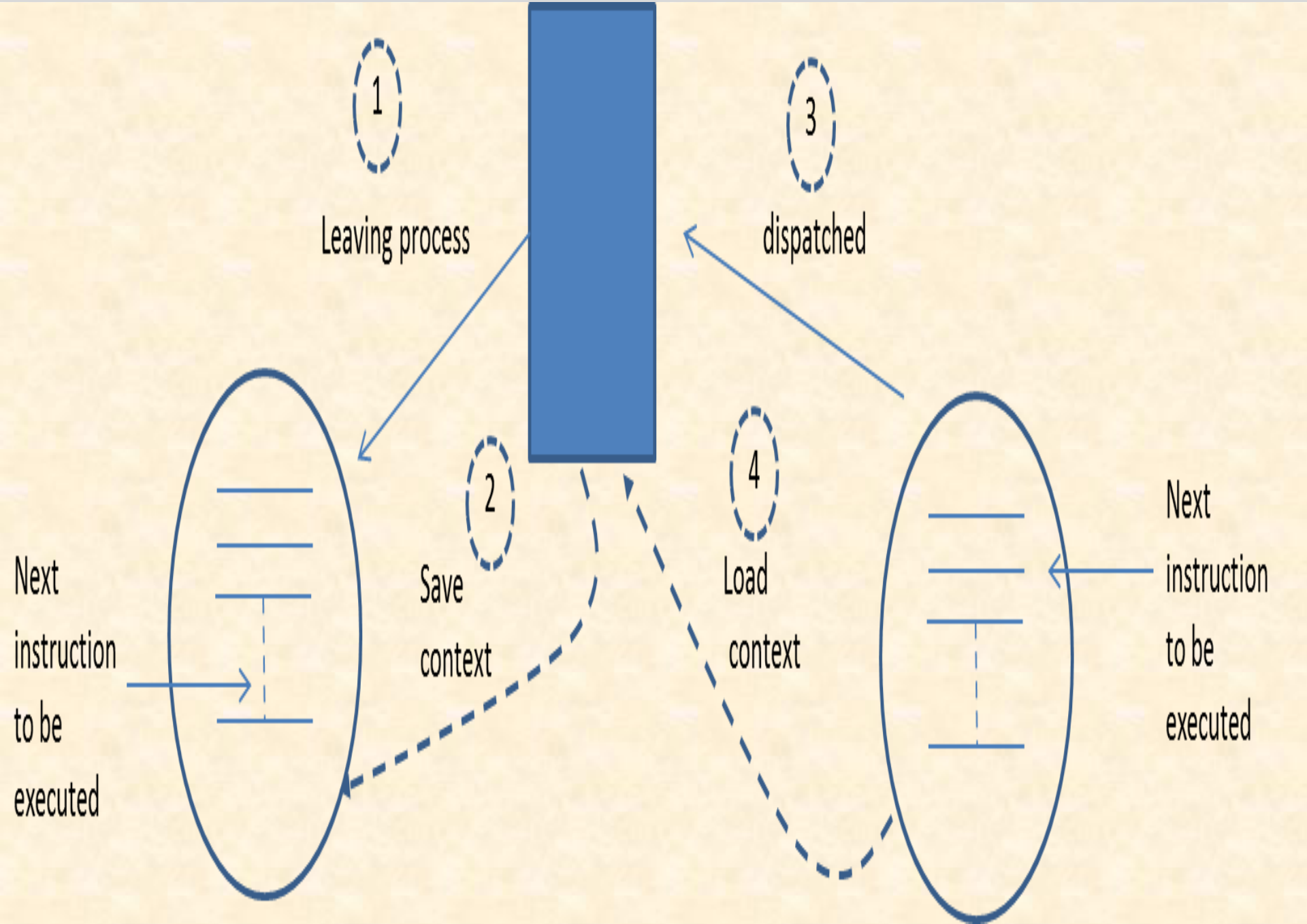
- **The process of changing *Context* (state) from an executing program (process → Pi) to an *interrupt handler* is called the *Context Switch* and its control requires a combination of hardware and software. [What is interrupt?]**
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- ***Context-switch time is overhead; the system does no useful work while switching. Time dependent on hardware support.***
- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.
- Using this technique a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

Example of Interrupt-driven I/O

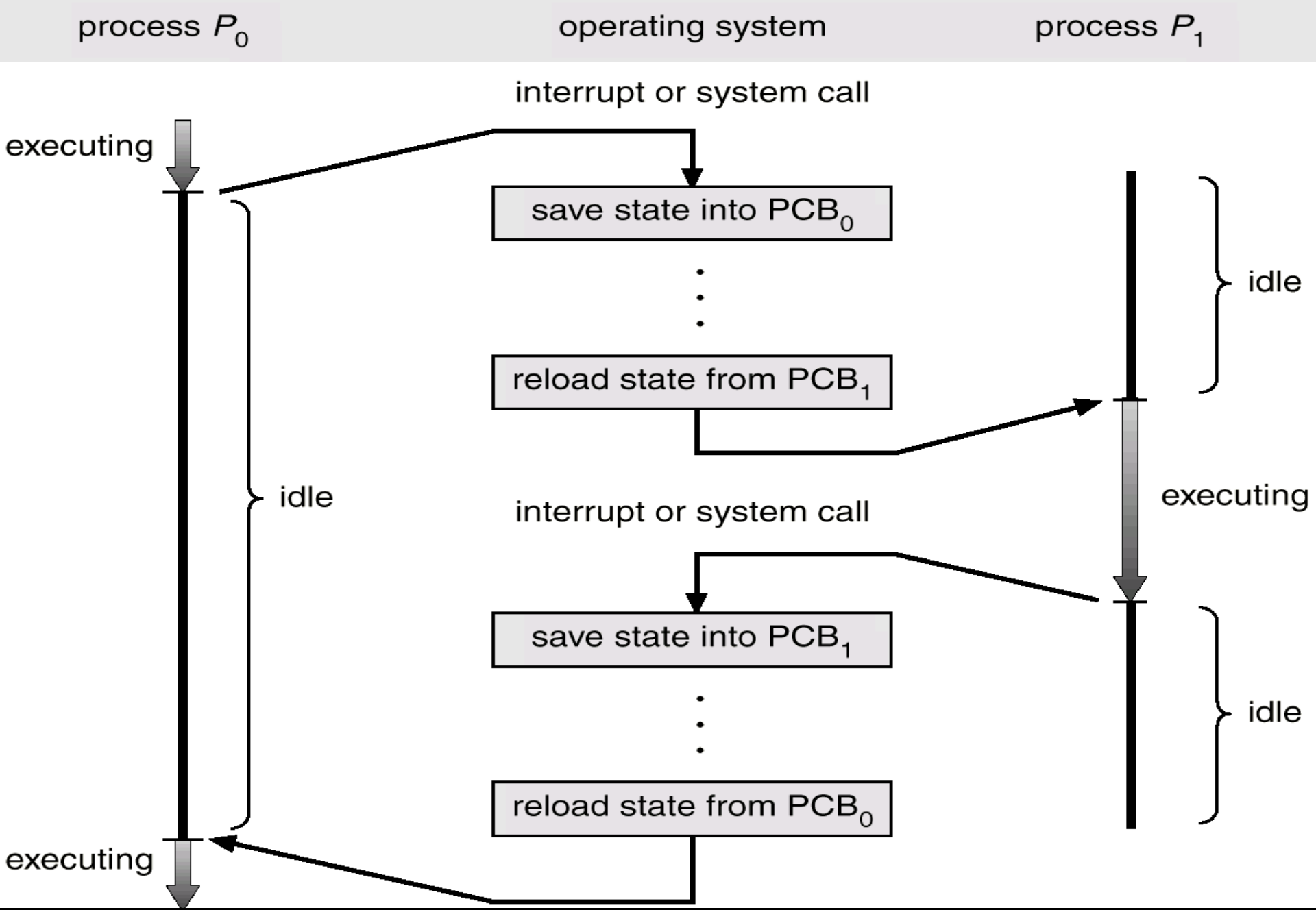
- When an I/O command is encountered by the CPU, it hands over the command to the I/O handler.
- The I/O handler initiates the I/O operation on the target device.
- When the I/O handler is finished with the I/O operation on the device, the I/O handler informs the CPU by issuing an 'interrupt'.



Context Switch



CPU Context Switch From Process to Process



Event Handling

- **Events that occur during the operation in OS:**
  1. **Process creation event**
  2. **Process termination event**
  3. **Timer event**
  4. **Resource request event**
  5. **Resource release event**
  6. **I/O initiation request event**

Event Handling (continued)

- Events that occur during the operation in OS (continued):
  7. I/O completion event
  8. Message send event
  9. Message receive event
  10. Signal send event
  11. Signal receive event
  12. A program interrupt
  13. A hardware malfunction event

Event Handling (continued)

- When an event occurs, the kernel must find the process whose state is affected by it
  - OSs use various schemes to speed this up
    - E.g., *event control blocks* (ECBs)

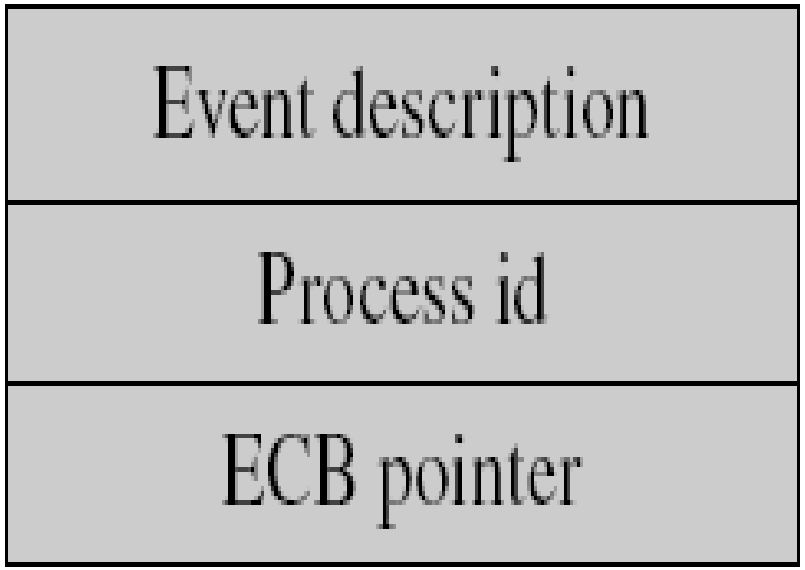


Figure Event control block (ECB).

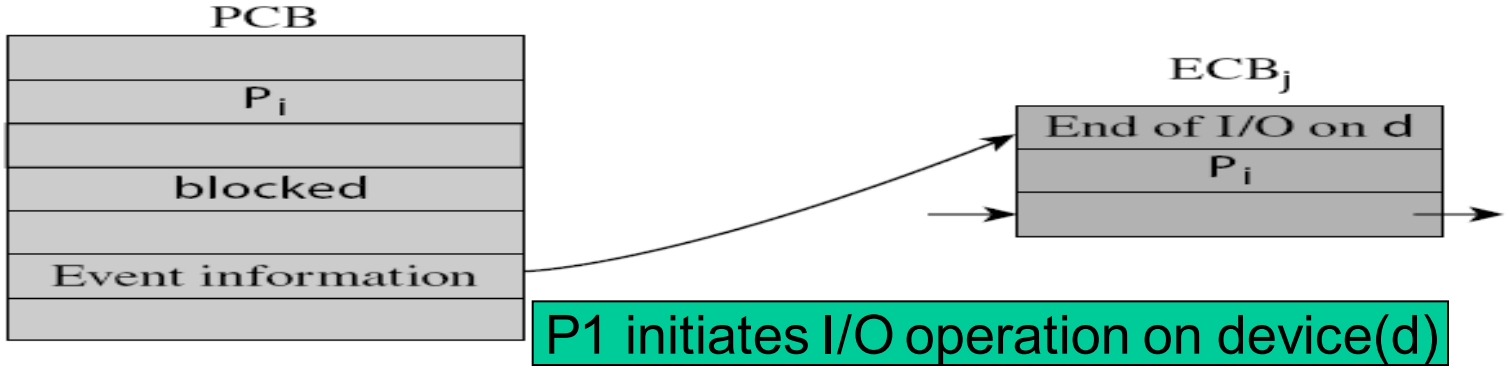


Figure PCB-ECB interrelationship.

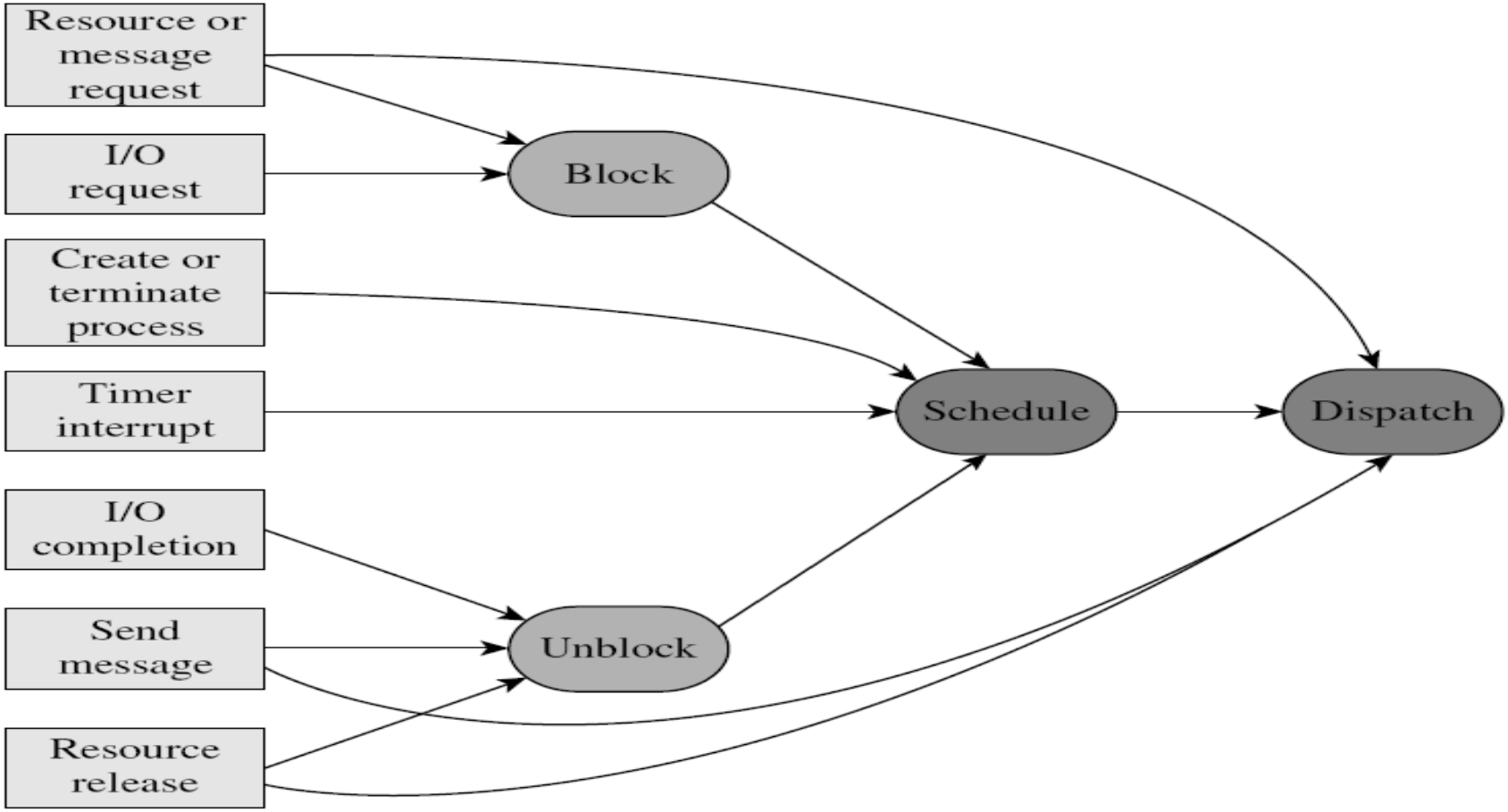


Figure Event handling actions of the kernel.

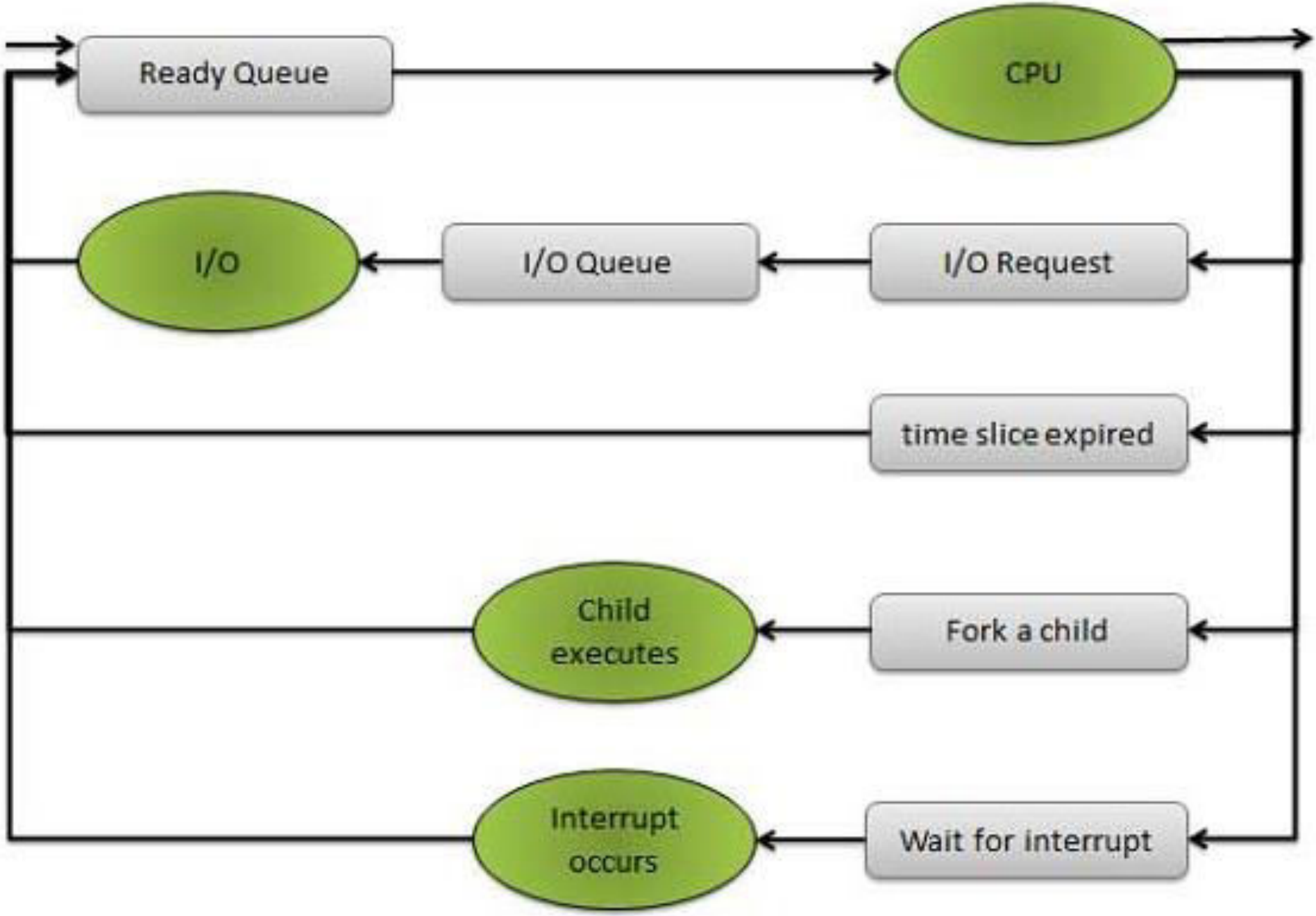
Process Scheduling (continued…)

- As processes enter the system, they are put into a ***job queue***.
- This queue consists of all processes in the system.
- The processes that are residing in main memory and are ready and waiting to execute are kept in the ***ready queue***.
- This queue is stored as a linked list.
- A ready-queue header contains pointers to the first and final PCBs in the list.
- Each PCB is extended to include a pointer that points to the next PCB in the ready queue.

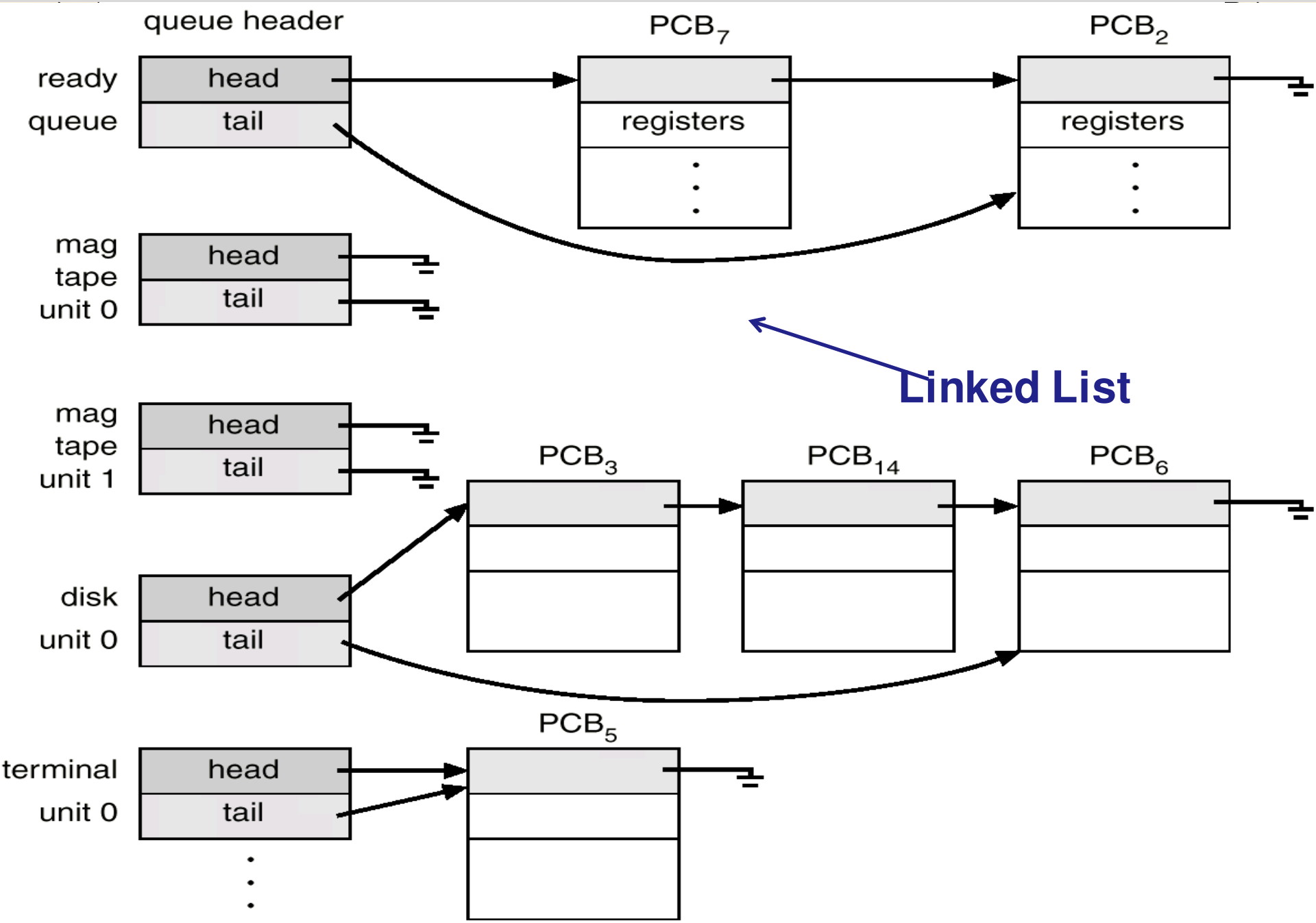
Process Scheduling Queues

- Scheduling queues refers to queues of processes or devices. When the process enters into the system, then this process is put into a job queue.
- This queue consists of all processes in the system. The operating system also maintains other queues such as device queue.
- Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has its own device queue.
- The following queuing diagram of process scheduling shows 1) Queue is represented by rectangular box; 2) The circles represent the resources that serve the queues; 3) The arrows indicate the process flow in the system;

Representation of Process Scheduling



Ready Queue And Various I/O Device Queues



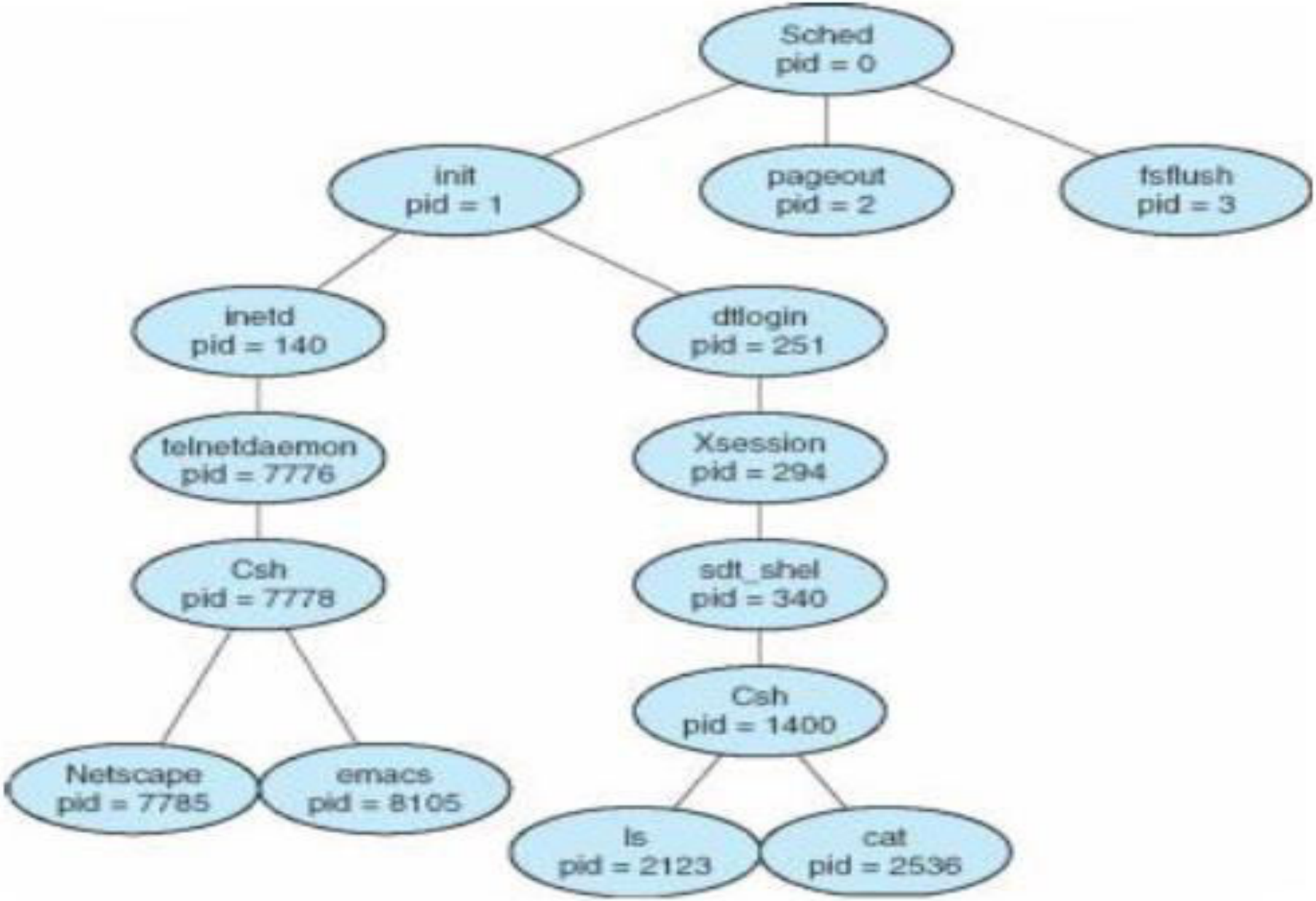
Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

Process Creation (Cont.)

- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.
- UNIX examples
  - **fork** system call creates new process
  - **execve** system call used after a **fork** to replace the process' memory space with a new program.

Operating System Concepts – A tree of processes on a typical Solaris



What process has Process ID 0?

- There are two tasks with specially distinguished process IDs: ***swapper or sched has process ID 0 and is responsible for paging***.

- This process is actually part of the kernel rather than a normal user-mode process. ***Process ID 1 is usually the init process primarily responsible for starting and shutting down the system.***

- Originally, process ID 1 was not specifically reserved for init by any technical measures: it simply had this ID as a natural consequence of being the first process invoked by the kernel.

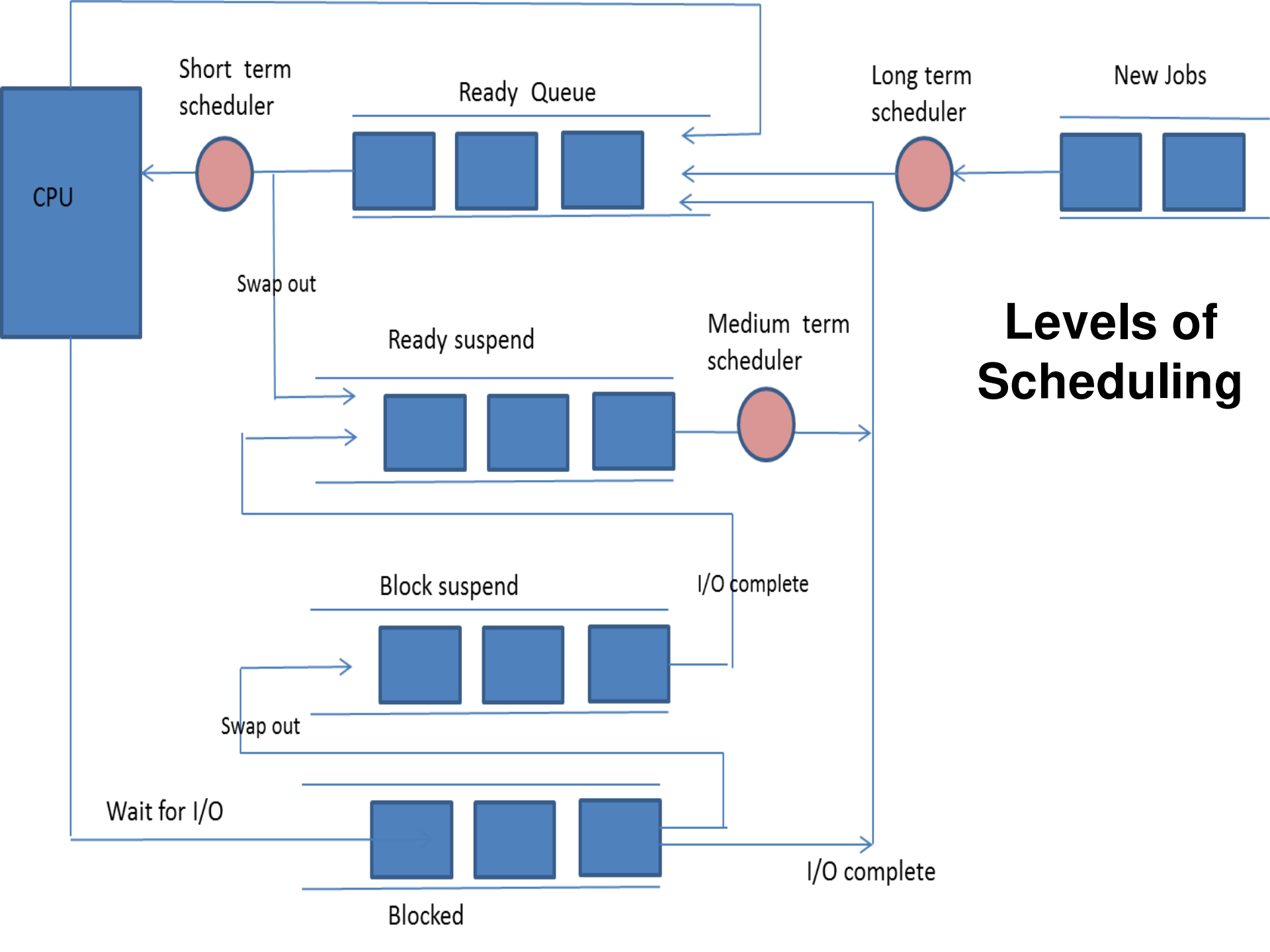
- More recent Unix systems typically have additional kernel components visible as 'processes', in which case PID 1 is actively reserved for the init process to maintain consistency with older systems.

Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
  - Output data from child to parent (via **wait**).
  - Process' resources are deallocated by operating system.
- Parent may terminate execution of children processes (**abort**).
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - Parent is exiting.
    - Operating system does not allow child to continue if its parent terminates.
    - Cascading termination.

Levels of Scheduling

- **Ready Queue** – These resident processes are waiting for the CPU.
- **New Jobs** – These are waiting to get main memory.
- **Blocked** – These resident processes are waiting for I/O completion.
- **Ready Suspend** – The processes are ready to run but have been swapped out to the hard disk.
- **Block Suspend** – These processes need I/O completion and have been swapped out to the hard disk.



Levels of Scheduling


START

Schedulers

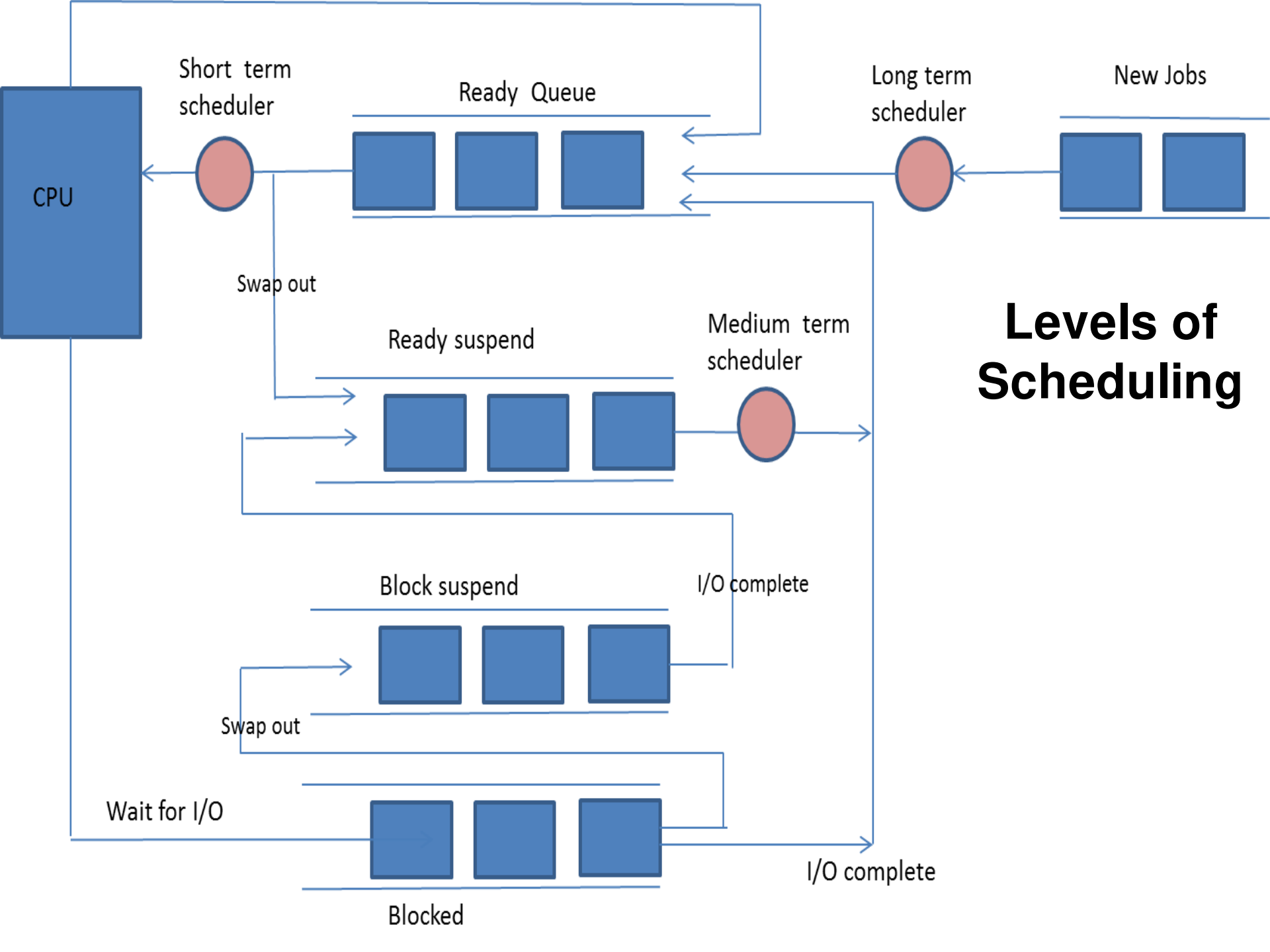
- **Short-term scheduler** ***manages the fair allocation of CPU time to processes. It makes the schedule of 'ready to run' processes. [NEXT SLIDE….]***
- It is also responsible for efficient process switching. The process switch time should be very small so that the system is able to provide better response time.
- The size of the context is also kept to the minimum so that the copying of the context into the PCB consumes the least time.

Short Term Scheduler

- ***It is also called CPU scheduler.***
- Main objective is increasing system performance in accordance with the chosen set of criteria.
- ***It is the change of ready state to running state of the process.***
- ***CPU scheduler selects processes among a set of processes that are ready to execute and allocates CPU to one of them.***
- ***Short term scheduler also known as dispatcher, execute most frequently*** and makes the fine grained decision of which process to execute next.
- *Short term scheduler is faster than long term scheduler*.

Schedulers (Cont.)

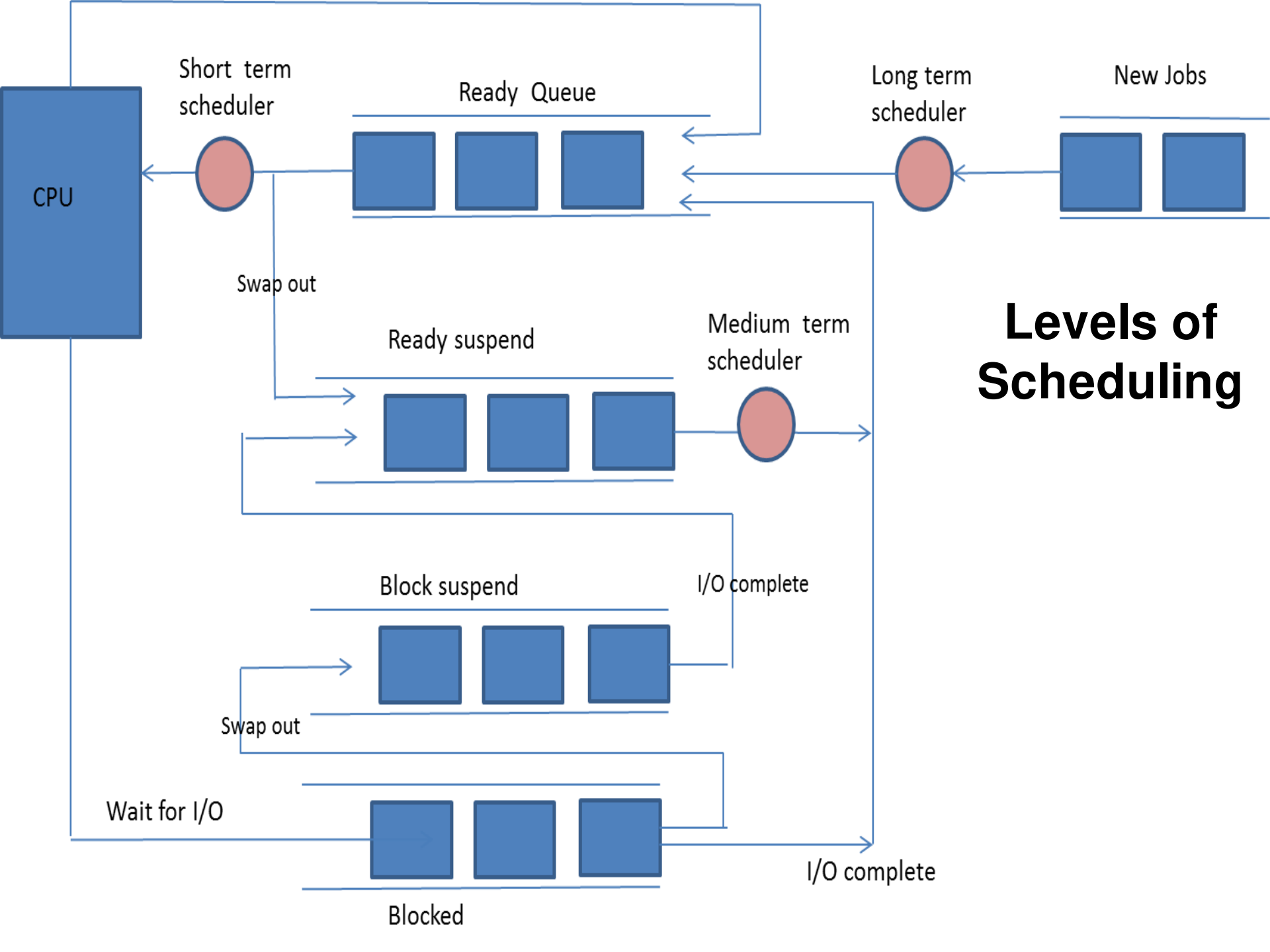
- **Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast).**
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow).
- The long-term scheduler controls the *degree of multiprogramming.*
- Processes can be described as either:
  - **I/O-*bound process*** – 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.



Levels of Scheduling

Long Term Schedulers (Cont.)

- *It is also called job scheduler. Long term scheduler determines which programs are admitted to the system for processing.*
- *Job scheduler selects processes from the queue and loads them into memory for execution.*
- *Process loads into the memory for CPU scheduling.*
- *The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound.*
- **It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.**
- **Time-sharing operating systems have no long term scheduler. When process changes the state from new to ready, then there is use of long term scheduler.**



Levels of Scheduling

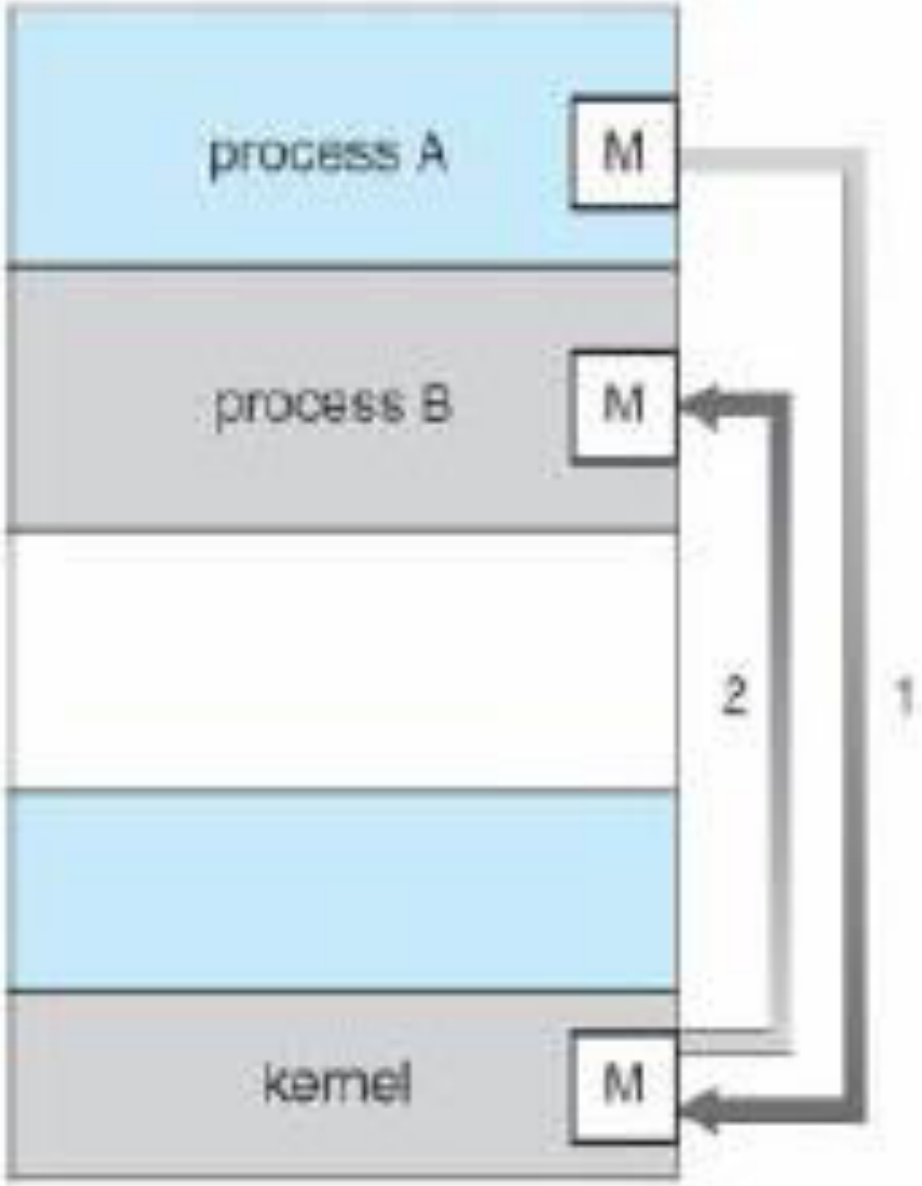
Medium Term Scheduling

- *Medium term scheduling is part of the swapping*.
- *It removes the processes from the memory*. It reduces the degree of multiprogramming.
- *The medium term scheduler is in-charge of handling the swapped out-processes*.
- *Running process may become suspended if it makes an I/O request*.
- *Suspended processes cannot make any progress towards completion*.
- *In this condition, to remove the process from memory and make space for other process, the suspended process is moved to the secondary storage*.
- *This process is called swapping, and the process is said to be swapped out or rolled out.* Swapping may be necessary to improve the process mix.

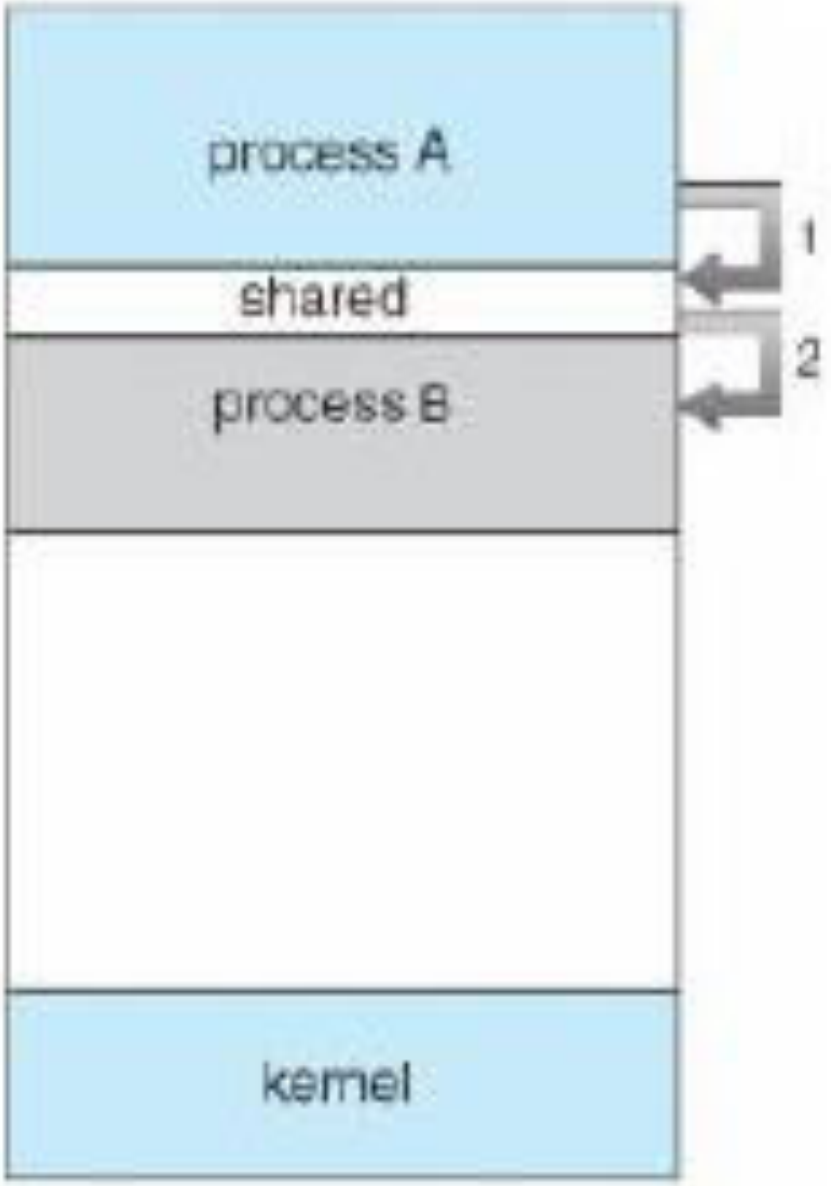
Interprocess Communication (IPC) [*more in threads' slide*]

- Processes within a system may be independent or cooperating
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need interprocess communication (IPC)
- Two models of IPC
  - Shared memory
  - Message passing

Communications Models



(a)



(b)

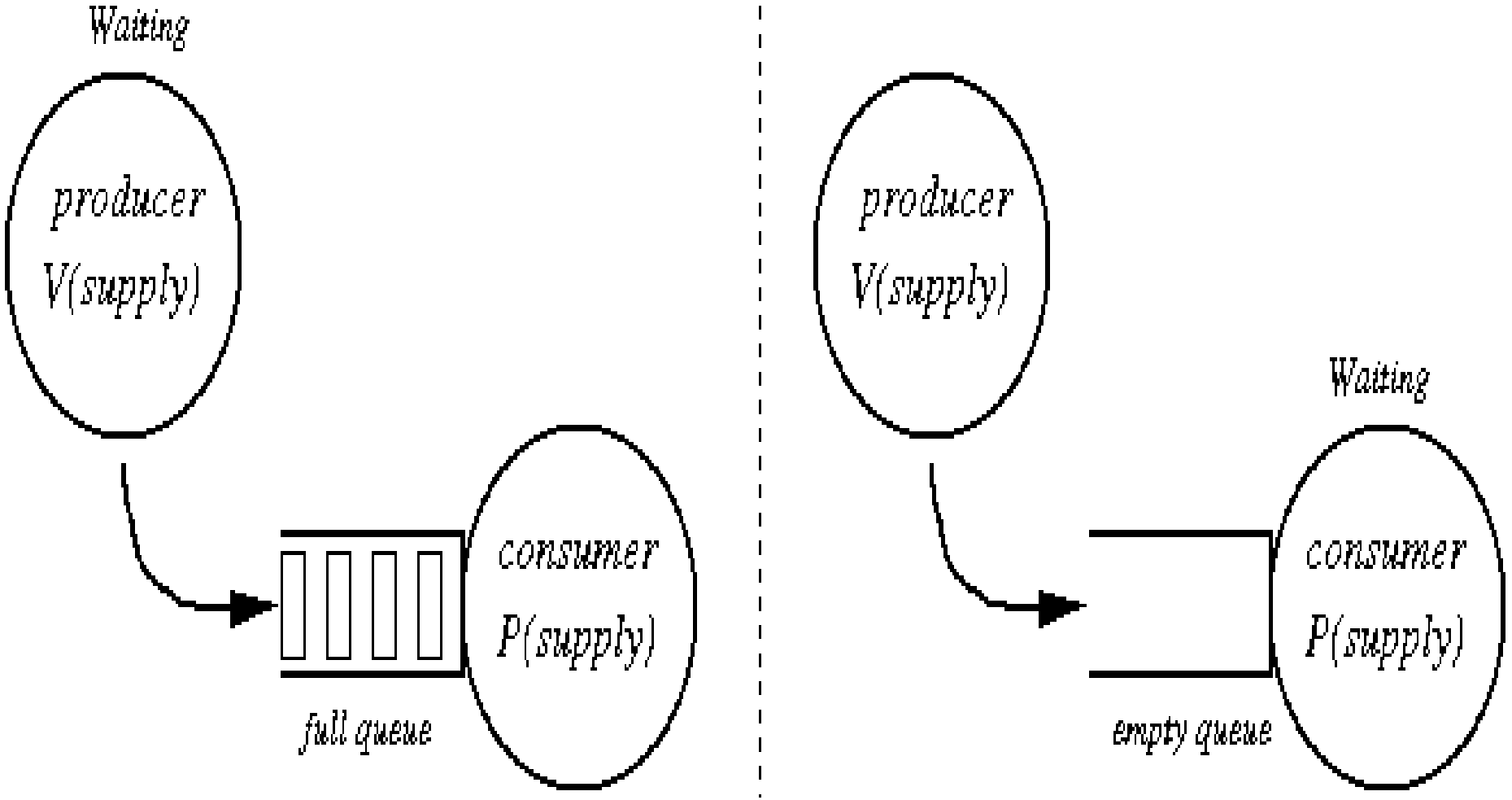
Cooperating Processes

- ***Independent*** process cannot affect or be affected by the execution of another process.
- ***Cooperating*** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Let's Understand Cooperating processes using producer-consumer problem

- In Producer – Consumer problem a producer process produces information that is consumed by a consumer process. **For example, a print program produces characters that are consumed by the printer driver.**
- **A compiler may produce assembly code, which is consumed by an assembler. The assembler, in turn, may produce object modules, which are consumed by the loader.**
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
- A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced. In this situation, the consumer must wait until an item is produced.

producer-consumer problem



Producer and Consumer


START