Continued……

Unit -2

Threads and Processes

Objectives

- To introduce the notion of a thread — a smallest dispatchable unit of CPU utilization that forms the basis of multithreaded computer systems
- To examine issues related to multithreaded programming

I. Process

- Resource ownership - process is allocated a virtual address space to hold the process image
- Scheduling/execution - follows an execution path that may be interleaved with other processes
- These two characteristics are treated independently by the operating system

Process

- Dispatching is referred to as a thread
- Resource of ownership is referred to as a process or task
- Protected access to processors, other processes, files, and I/O resources

2. Multithreading

- Operating system supports multiple threads of execution within a single process
- MS-DOS supports a single thread
- UNIX supports multiple user processes but only supports one thread per process
- Windows 2000, Solaris, Linux, Mach, and OS/2 support multiple threads

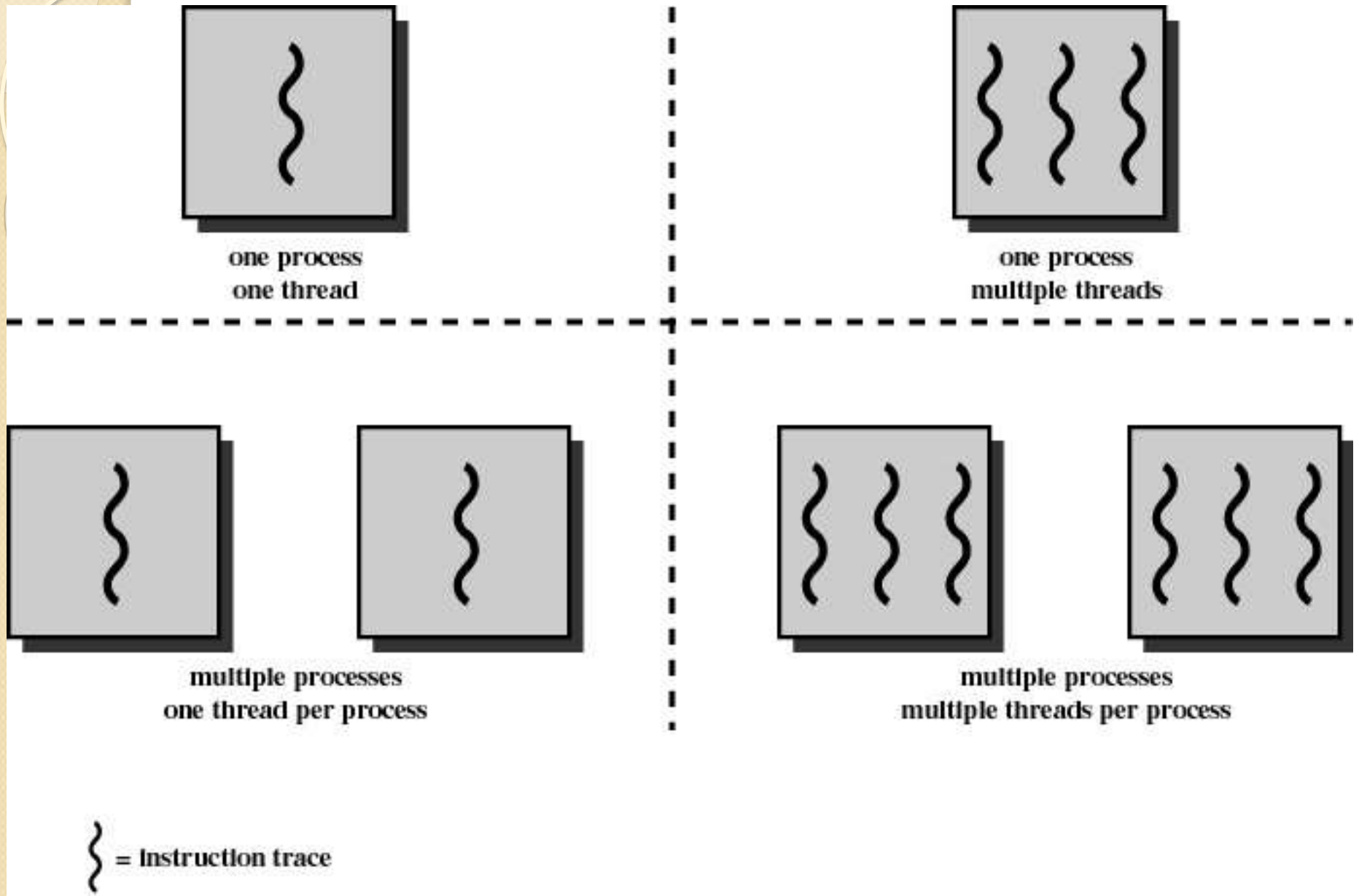
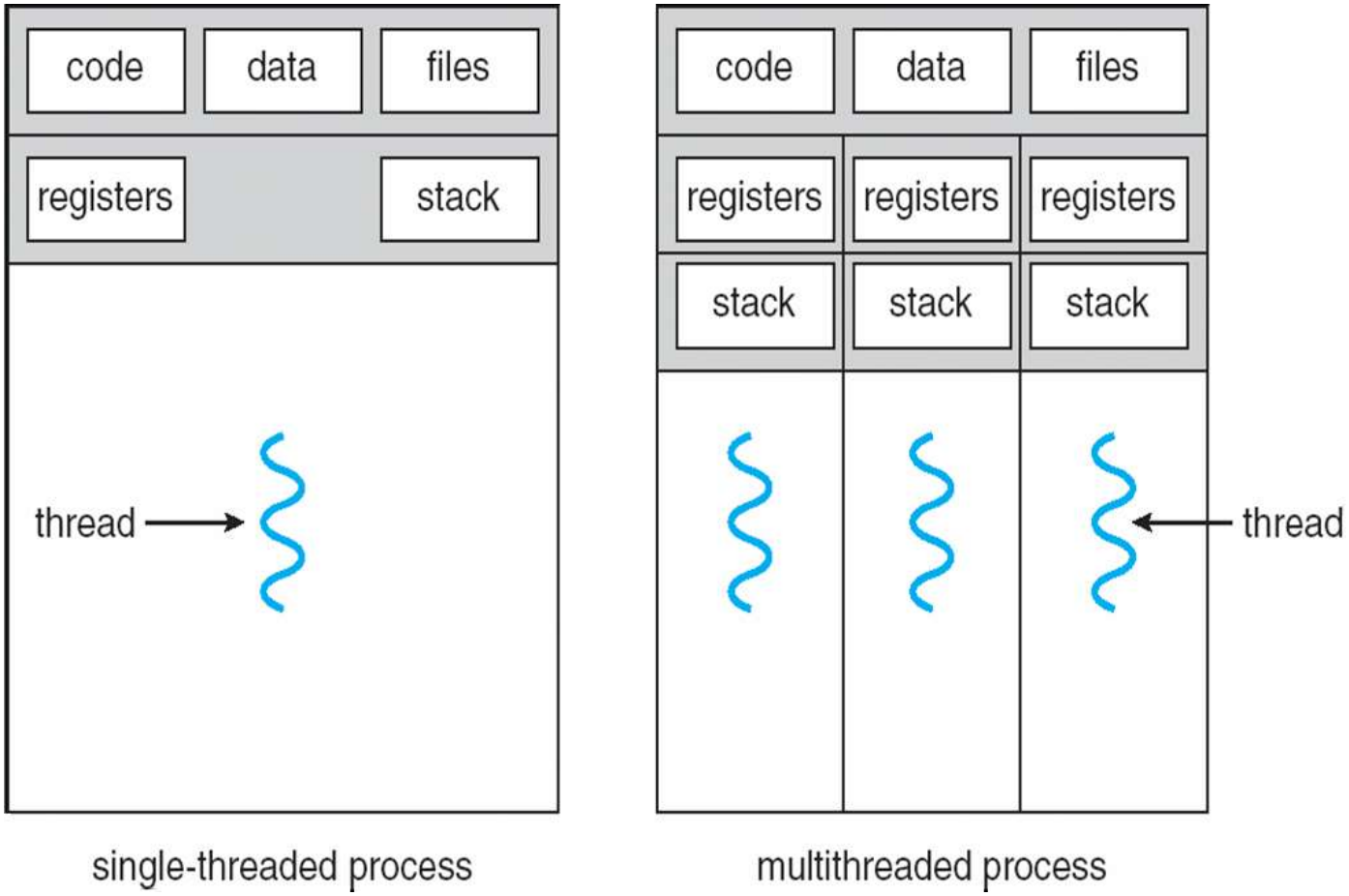


Figure 4.1 Threads and Processes [ANDE97]

Single and Multithreaded Processes



single-threaded process

multithreaded process

3. Thread

- An execution state (running, ready, etc.)
- Saved thread context when not running
- Has an execution stack
- Some per-thread static storage for local variables
- Access to the memory and resources of its process
  - all threads of a process share this

Threads

- Suspending a process involves suspending all threads of the process since all threads share the same address space
- Termination of a process, terminates all threads within the process

Benefits of Threads

- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Less time to switch between two threads within the same process
- Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel

Uses of Threads in a Single-User Multiprocessing System

- Foreground to background work
- Asynchronous processing
- Speed execution
- Modular program structure

4. Thread States

- States associated with a change in thread state
  - Spawn
    - Issue another thread
  - Block
  - Unblock
  - Finish
    - Deallocate register context and stacks

5. Levels of Threads

5.1 User-Level Threads

- All thread management is done by the application
- The kernel is not aware of the existence of threads

5.2 Kernel-Level Threads

- W2K, Linux, and OS/2 are examples of this approach
- Kernel maintains context information for the process and the threads
- Scheduling is done on a thread basis

5.3 Combined Approaches

- Example is Solaris
- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads done in the user space

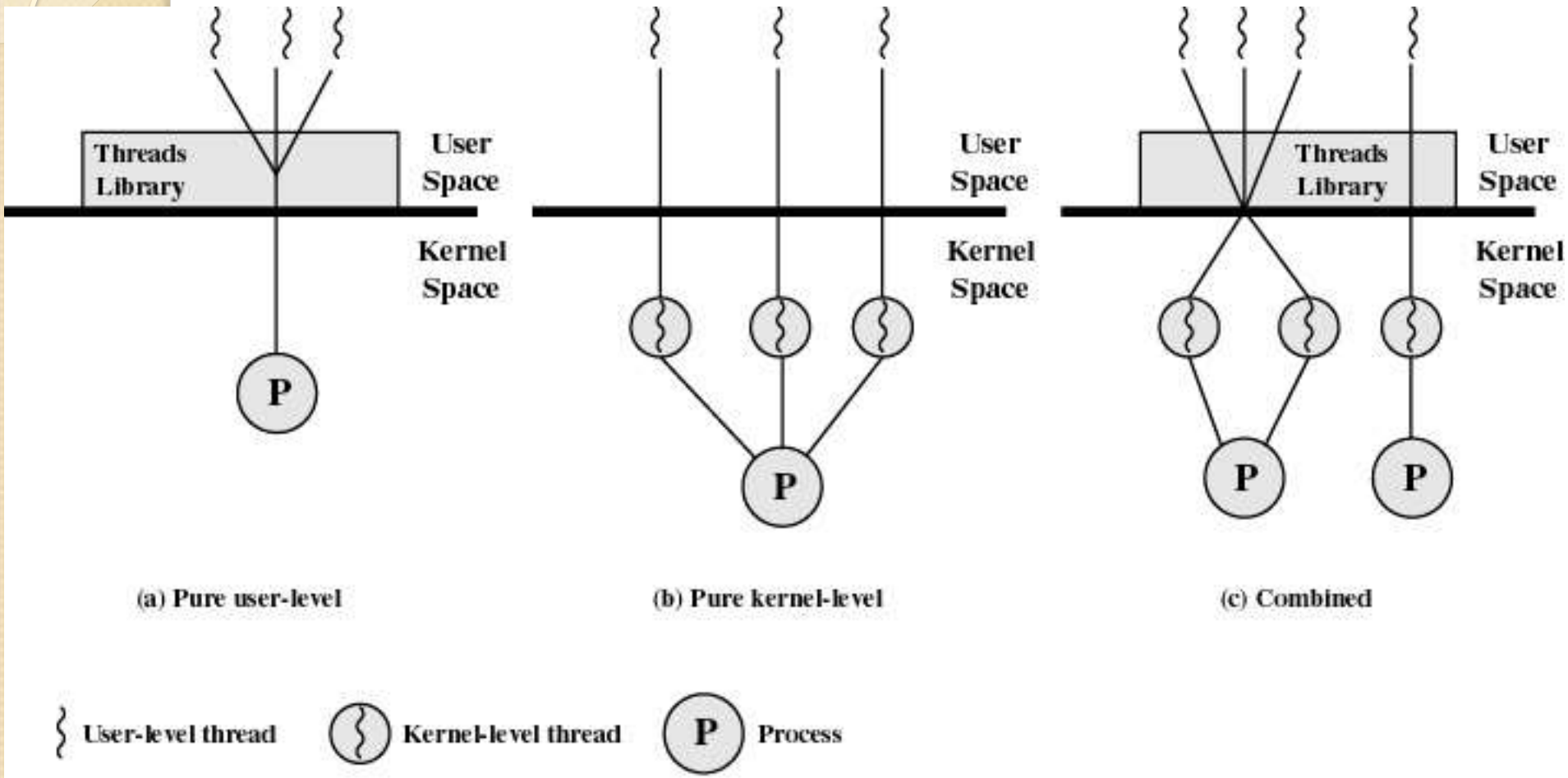


Figure 4.6 User-Level and Kernel-Level Threads

Relationship Between Threads and Processes

Threads:Process	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, OS/2, OS/390, MACH

Relationship Between Threads and Processes

Threads:Process	Description	Example Systems
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:M	Combines attributes of M:1 and 1:M cases	TRIX

Unit 2

Process and Process Scheduling

1. Major Requirements of an Operating System

- Interleave the execution of several processes to maximize processor utilization while providing reasonable response time
- Allocate resources to processes
- Support inter-process communication and
- Creation of user processes

2. Process

- Also called a task
- Execution of an individual program
- Can be traced
  - list the sequence of instructions that execute

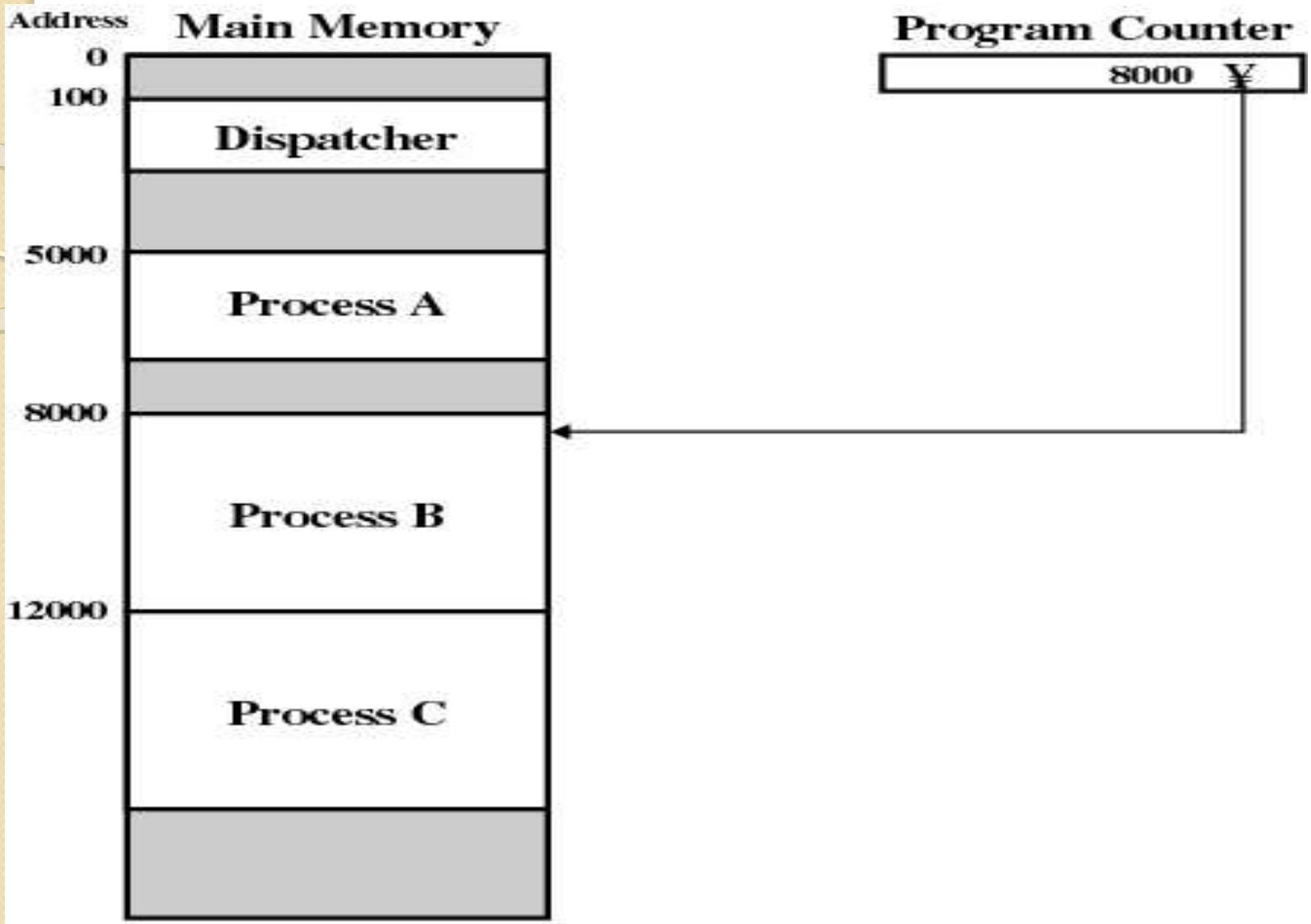


Figure 3.1 Snapshot of Example Execution (Figure 3.3) at Instruction Cycle 13

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

(a) Trace of Process A **(b) Trace of Process B** **(c) Trace of Process C**

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Figure 3.2 Traces of Processes of Figure 3.1

1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
----------	Time out
7	100
8	101
9	102
10	103
11	104
12	105
13	8000
14	8001
15	8002
16	8003
----------	I/O request
17	100
18	101
19	102
20	103
21	104
22	105
23	12000
24	12001
25	12002
26	12003
27	12004
28	12005
----------	Time out
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
----------	Time out
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
----------	Time out

100 = Starting address of dispatcher program

shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figure 3.3 Combined Trace of Processes of Figure 3.1

3. Process Creation (how?)

- Submission of a batch job
- User logs on
- Created to provide a service such as printing
- Process creates another process
- Batch job issues *Halt* instruction
- User logs off
- Quit an application
- Error and fault conditions

Reasons for Process Termination

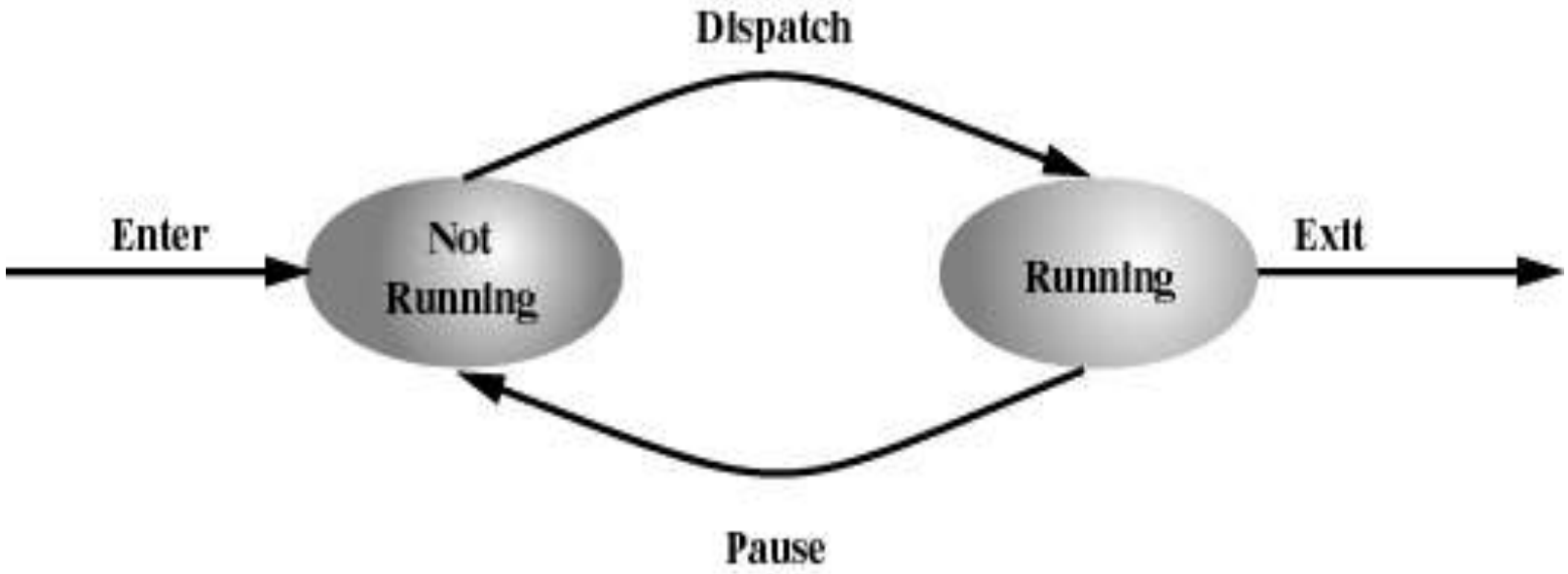
- Normal completion
- Time limit exceeded
- Memory unavailable
- Bounds violation
- Protection error
  - example write to read-only file
- Arithmetic error
- Time overrun
  - process waited longer than a specified maximum for an event

Reasons for Process Termination

- I/O failure
- Invalid instruction
  - happens when try to execute data
- Privileged instruction
- Data misuse
- Operating system intervention
  - such as when deadlock occurs
- Parent terminates so child processes terminate
- Parent request

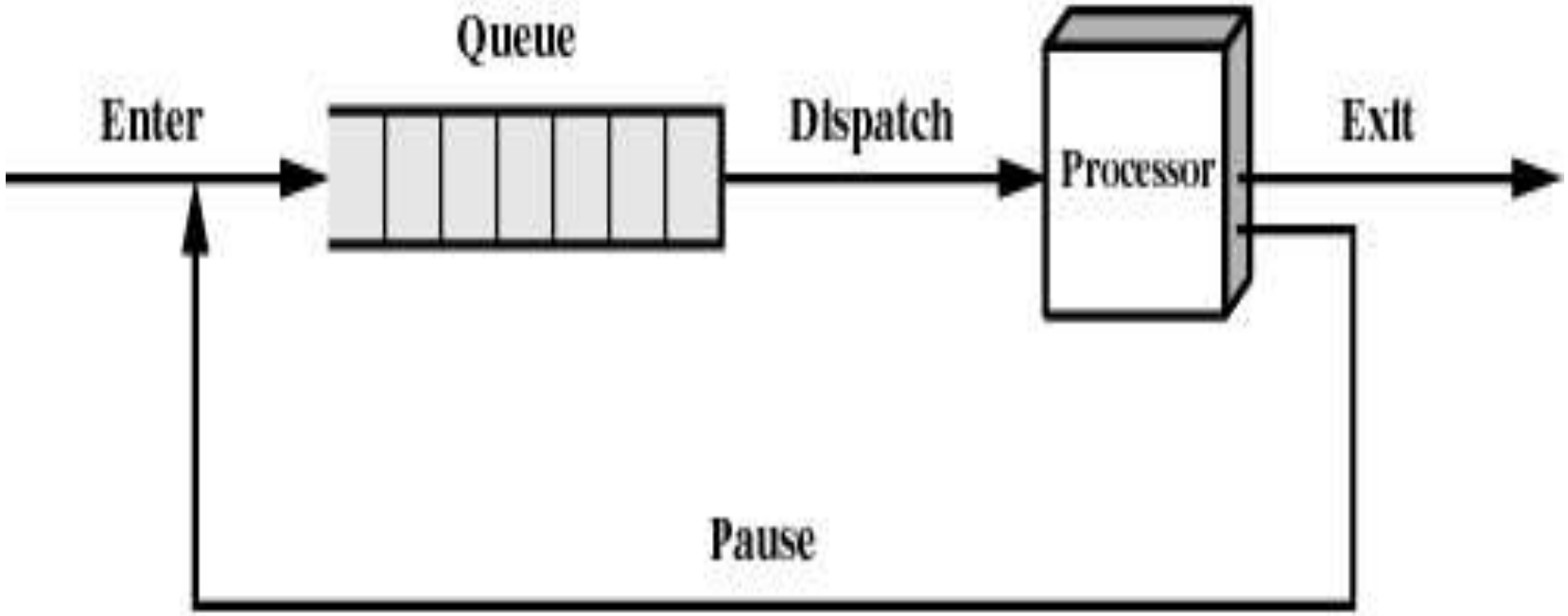
4. Two-State Process Model

- Process may be in one of two states
  - Running
  - Not-running



(a) State transition diagram

Not-Running Process in a Queue



(b) Queuing diagram

Processes

- Not-running
  - ready to execute
- Blocked
  - waiting for I/O
- Dispatcher cannot just select the process that has been in the queue the longest because it may be blocked

5. A Five-State Model

- Running
- Ready
- Blocked
- New
- Exit

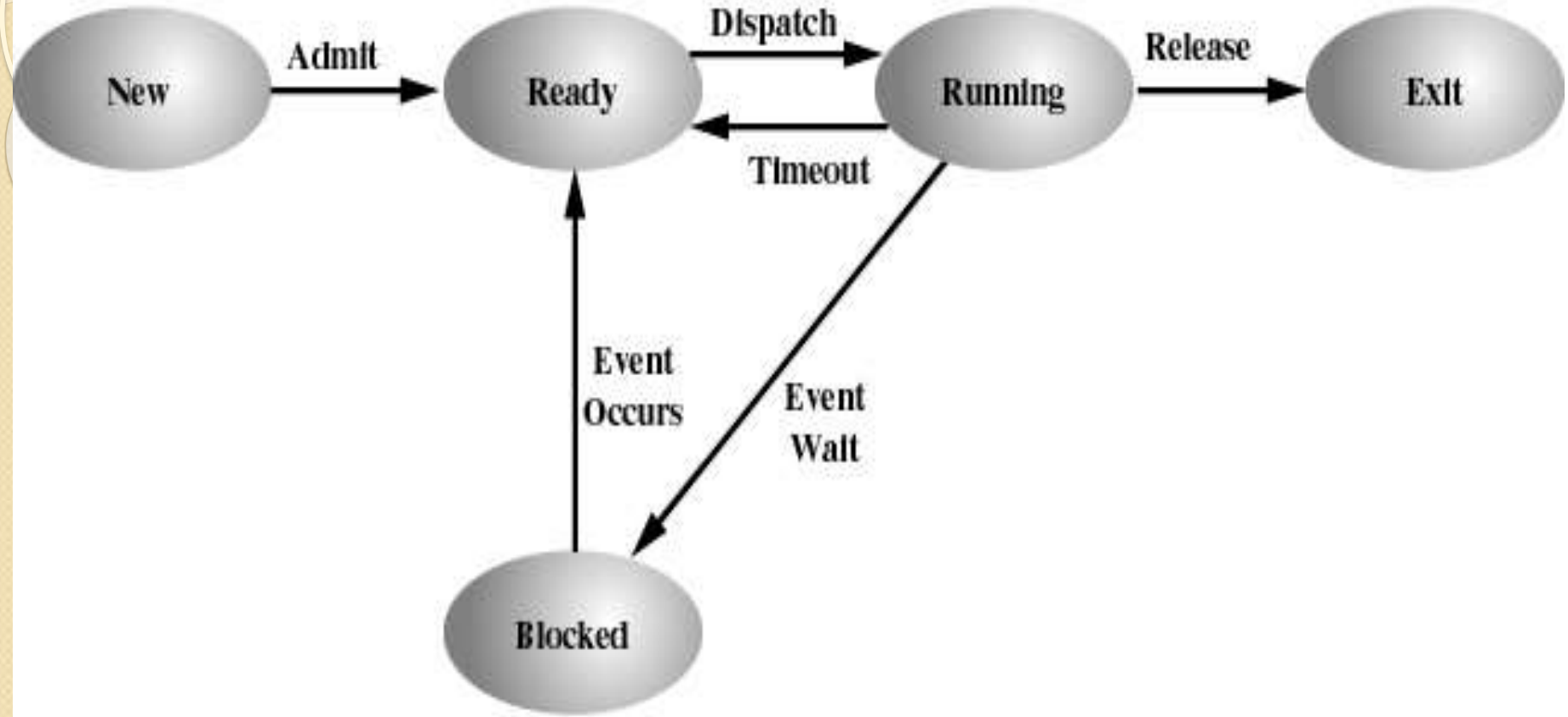
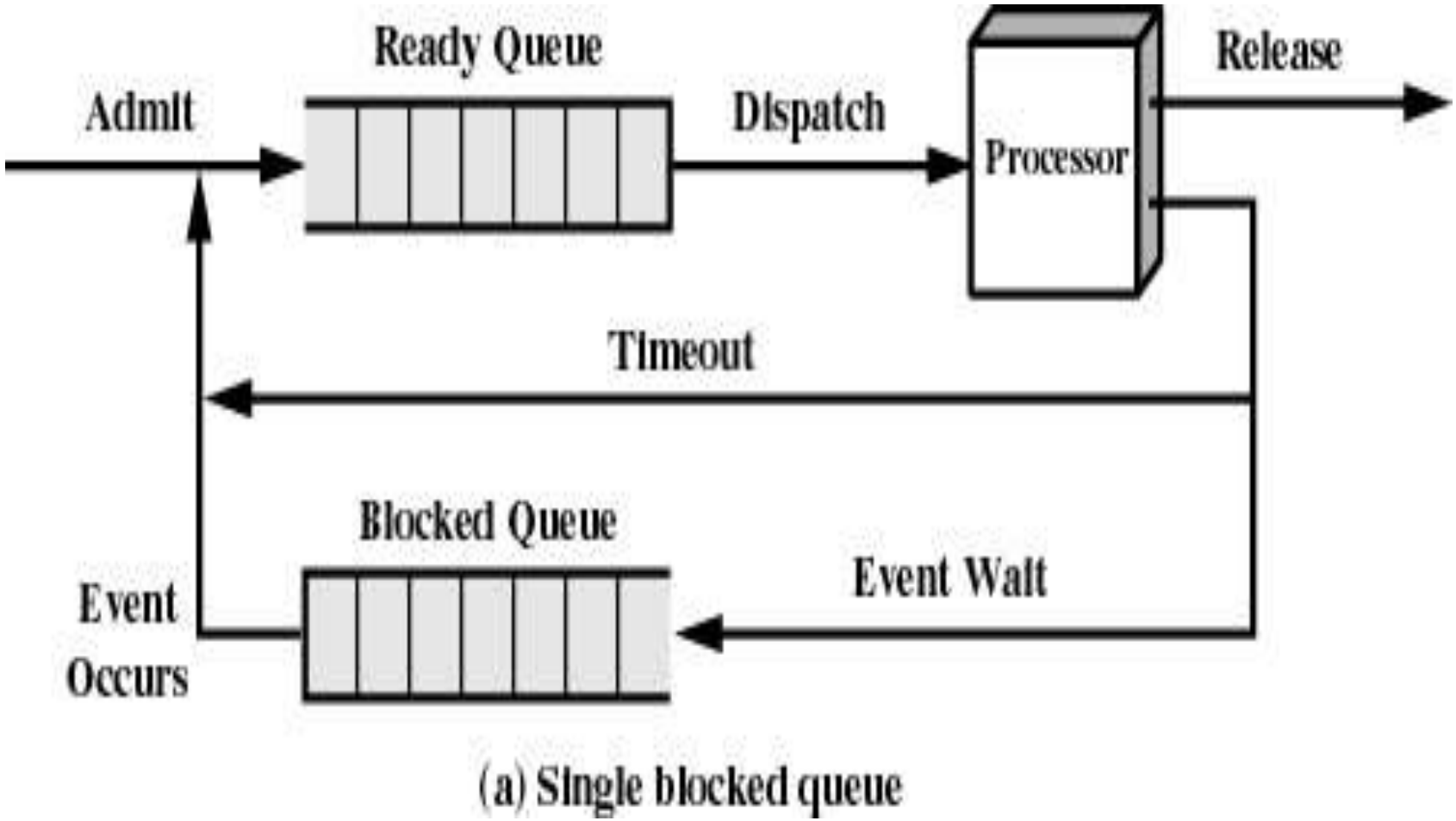
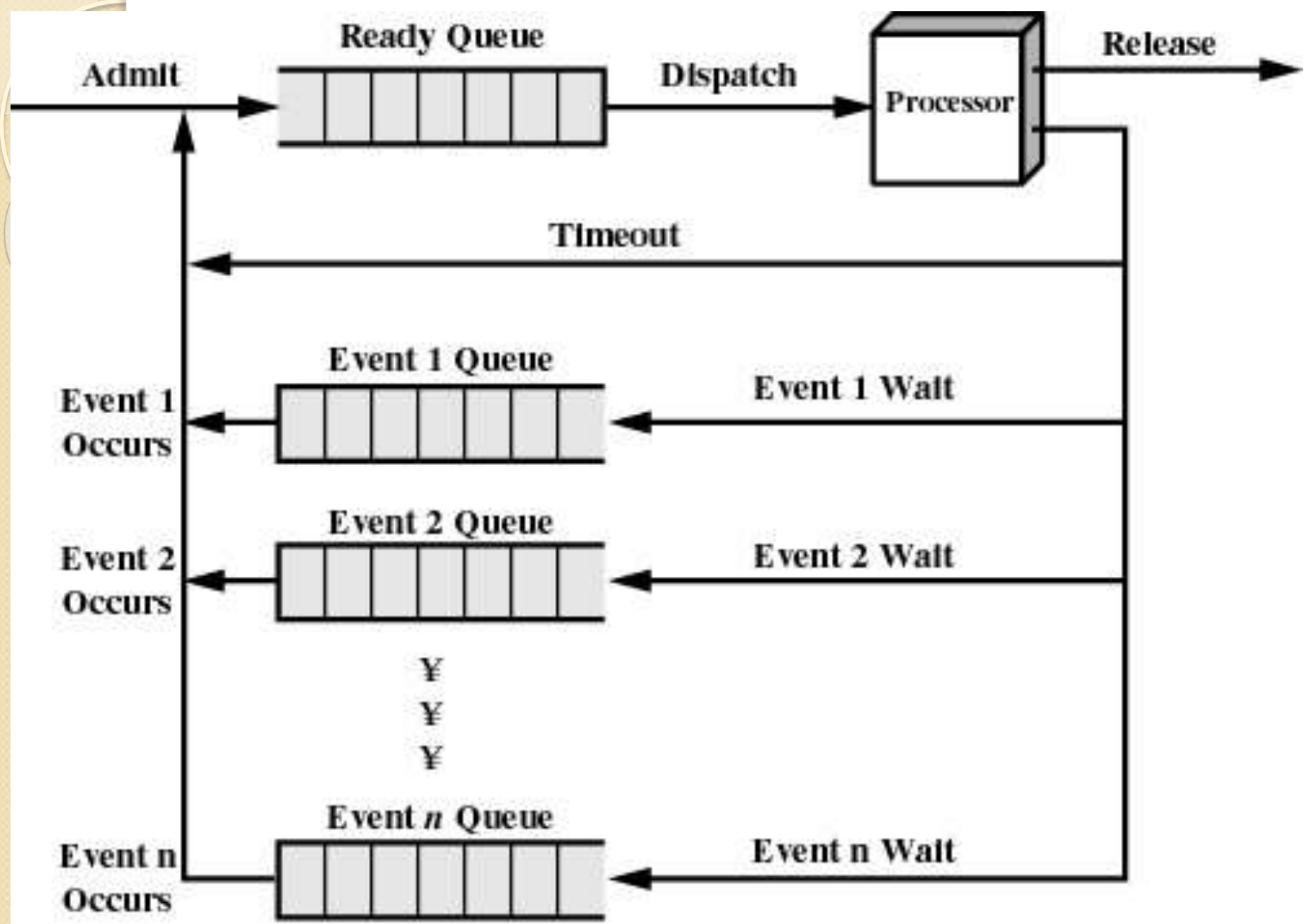


Figure 3.5 Five-State Process Model

Using Two Queues

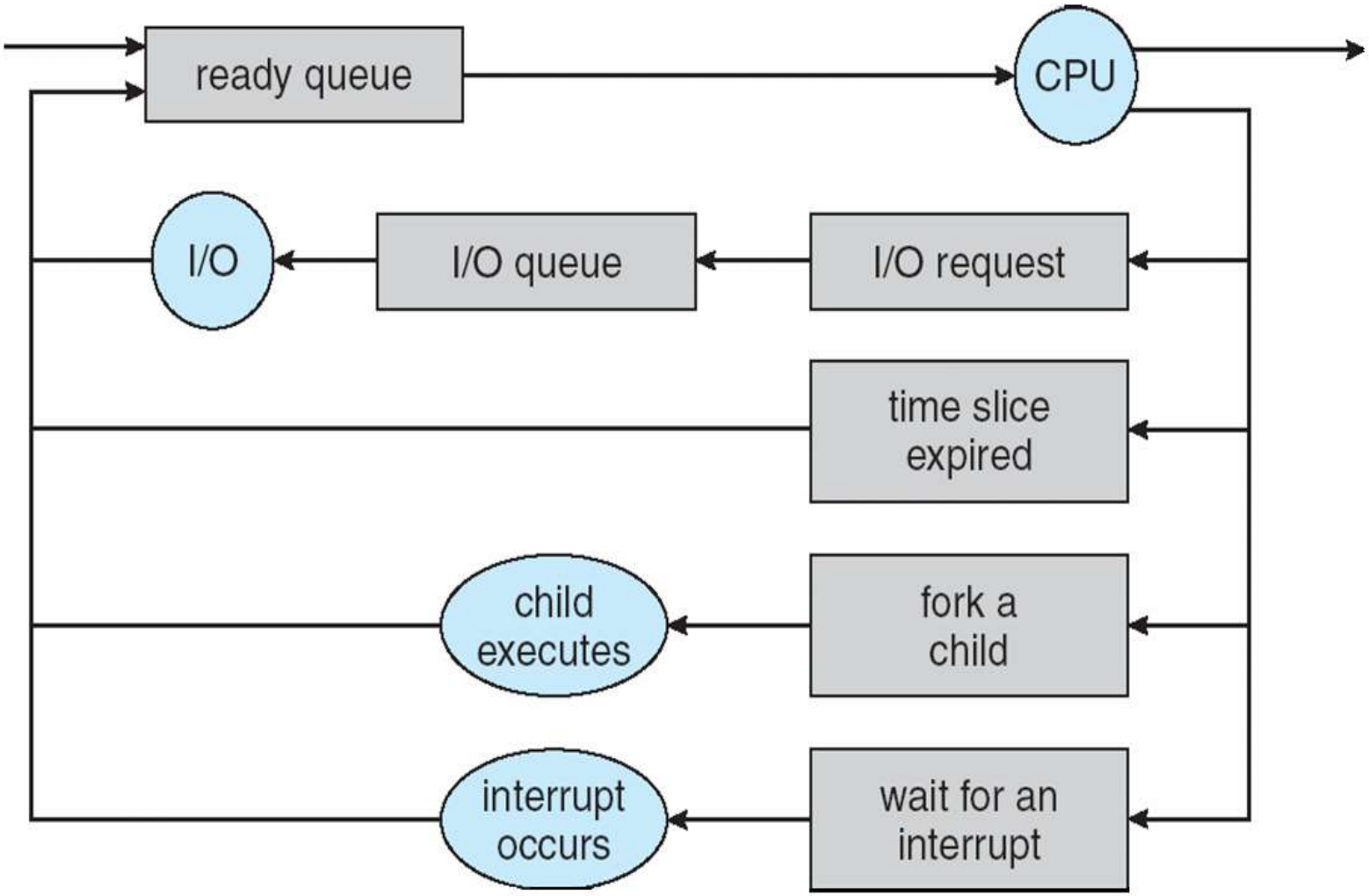


(a) Single blocked queue



(b) Multiple blocked queues

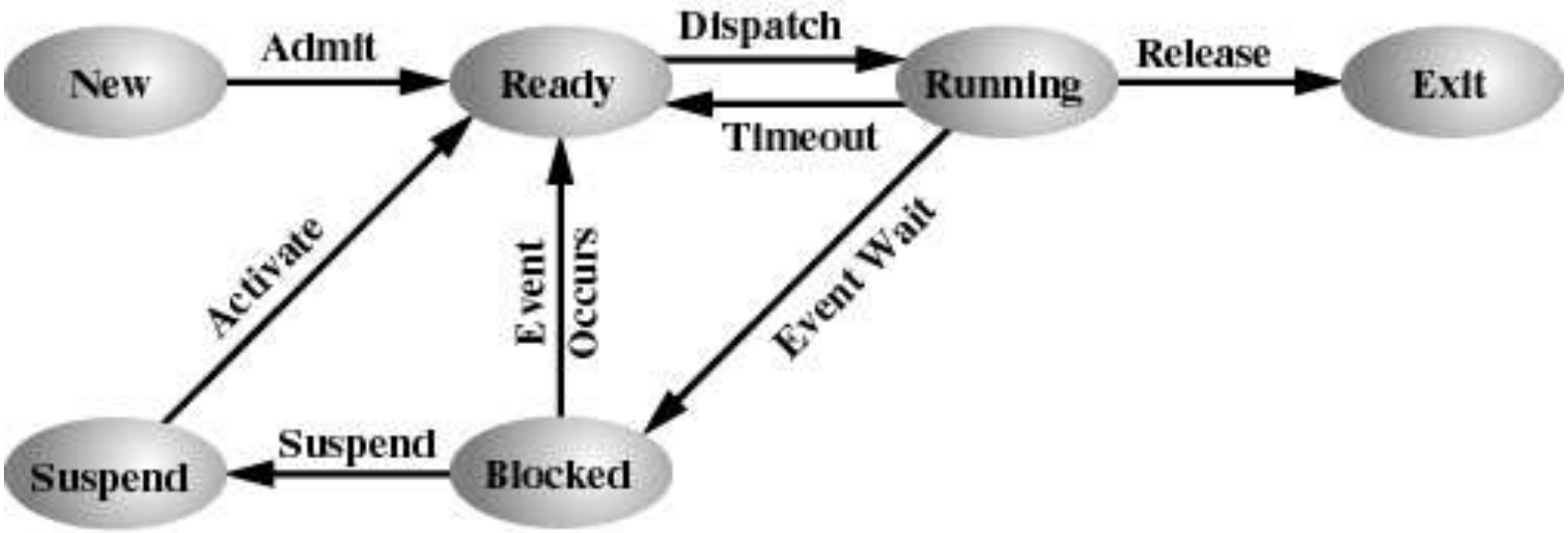
Example


ready queue
CPU
I/O
I/O queue
I/O request
time slice expired
child executes
fork a child
interrupt occurs
wait for an interrupt

6. Suspended Processes

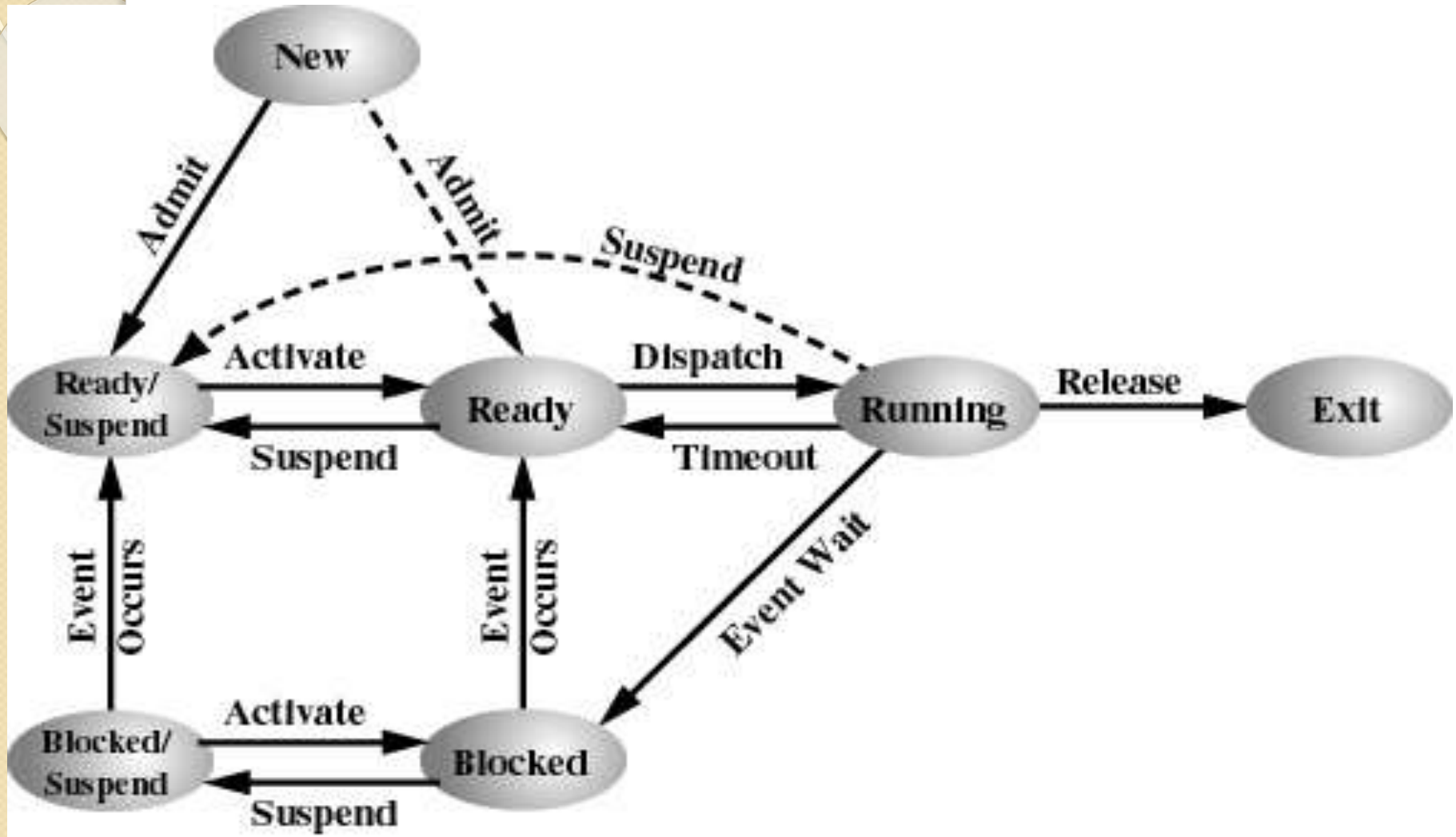
- Processor is faster than I/O so all processes could be waiting for I/O
- Swap these processes to disk to free up more memory
- Blocked state becomes suspend state when swapped to disk
- Two new states
  - Blocked, suspend
  - Ready, suspend

One Suspend State



(a) With One Suspend State

Two Suspend States



(b) With Two Suspend States

7. Operating System Control Structures

- Information about the current status of each process and resource
- Tables are constructed for each entity the operating system manages

7.a Memory Tables

- Allocation of main memory to processes
- Allocation of secondary memory to processes
- Protection attributes for access to shared memory regions
- Information needed to manage virtual memory

7.b I/O Tables

- I/O device is available or assigned
- Status of I/O operation
- Location in main memory being used as the source or destination of the I/O transfer

7.c File Tables

- Existence of files
- Location on secondary memory
- Current Status
- Attributes
- Sometimes this information is maintained by a file-management system

7.d Process Table

- Where process is located
- Attributes necessary for its management
  - Process ID
  - Process state
  - Location in memory

Process Location

- Process includes set of programs to be executed
  - Data locations for local and global variables
  - Any defined constants
  - Stack
- Process control block
  - Collection of attributes
- Process image
  - Collection of program, data, stack, and attributes

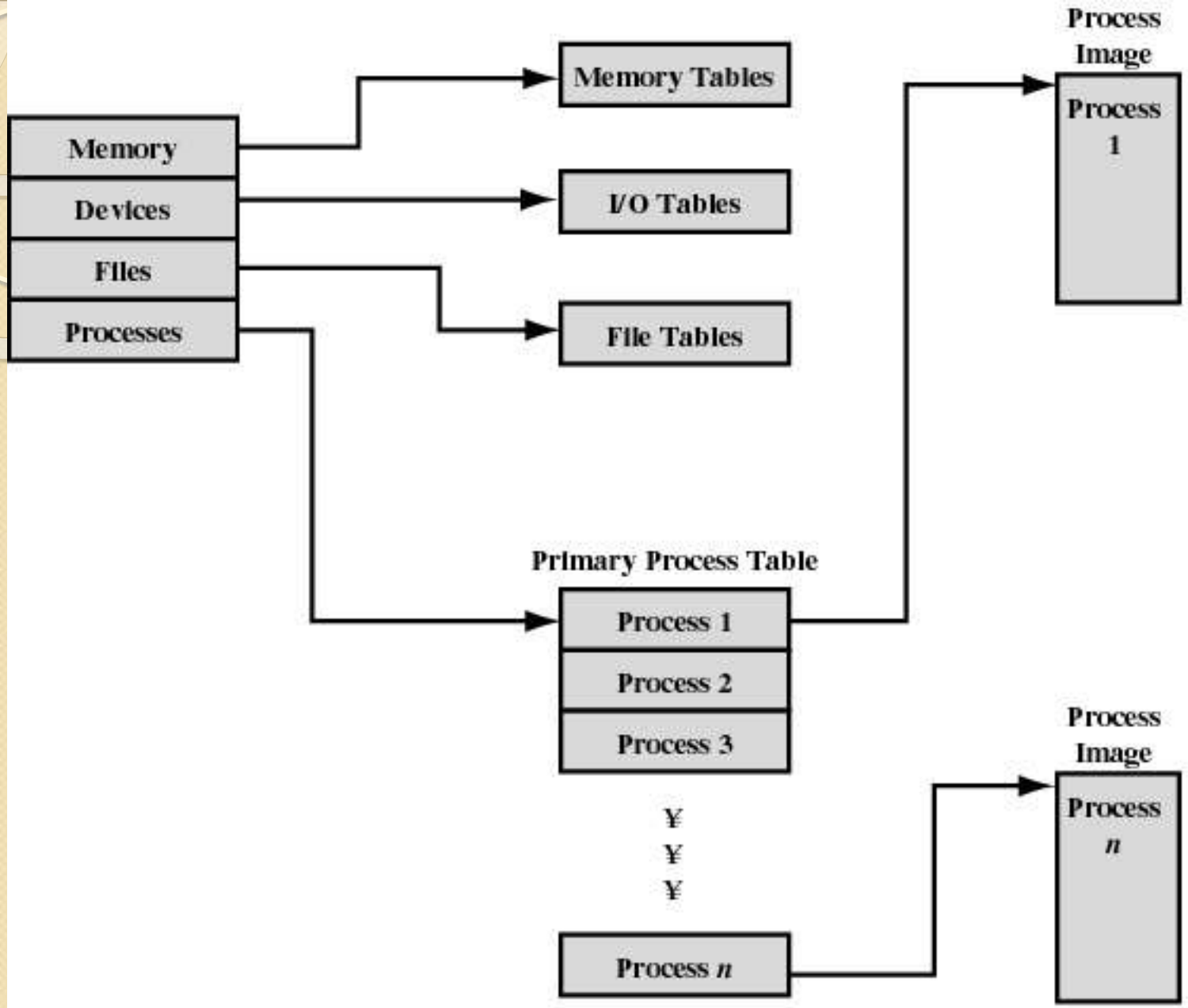


Figure 3.10 General Structure of Operating System Control Tables

8. Process Control Block

1. Process identification (Identifiers)

- Numeric identifiers that may be stored with the process control block include
- Identifier of this process
- Identifier of the process that created this process (parent process)
- User identifier

Process Control Block

2. Processor State Information

2.1 User-Visible Registers

- A user-visible register is one that may be referenced by means of the machine language that the processor executes.
- Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.

Process Control Block

Processor State Information (contd..)

2.2 Control and Status Registers

These are a variety of processor registers that are employed to control the operation of the processor. These include

- ***Program counter:*** Contains the address of the next instruction to be fetched
- ***Condition codes:*** Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
- ***Status information:*** Includes interrupt enabled/disabled flags, execution mode

Process Control Block

Processor State Information (contd…..)

- **Stack Pointers**
  - Each process has one or more last-in-first-out (LIFO) system stacks associated with it.
  - A stack is used to store parameters and calling addresses for procedure and system calls.
  - The stack pointer points to the top of the stack.

Process Control Block

3. Process Control Information

3.1 Scheduling and State Information

This is information that is needed by the operating system to perform its scheduling function.

Typical items of information:

- ***Process state:*** defines the readiness of the process to be scheduled for execution (e.g., running, ready, blocked, halted, exit).
- ***Priority:*** One or more fields may be used to describe the scheduling priority of the process.

Process Control Block

Scheduling and State Information (contd…)

Scheduling-related information: This will depend on the scheduling algorithm used.

Examples:- the amount of time that the process has been waiting and

the amount of time that the process executed the last time it was running.

Event: Identity of event the process is awaiting before it can be resumed

Process Control Block

Process Control Information (contd…)

- **Data Structuring**
  - A process may be linked to other process in a queue, ring, or some other structure.
  - For example, all processes in a waiting state for a particular priority level may be linked in a queue.
  - A process may exhibit a parent-child (creator-created) relationship with another process.
  - The process control block may contain pointers to other processes to support these structures.

Process Control Block

Process Control Information (contd…..)

- **Inter-process Communication**
  - Various flags, signals, and messages may be associated with communication between two independent processes.
  - Some or all of this information may be maintained in the process control block.
- **Process Privileges**
  - Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed.
  - In addition, privileges may apply to the use of system utilities and services.

Process Control Block

Process Control Information (contd….)

- **Memory Management**
  - This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.
- **Resource Ownership and Utilization**
  - Resources controlled by the process may be indicated, such as opened files.
  - A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.

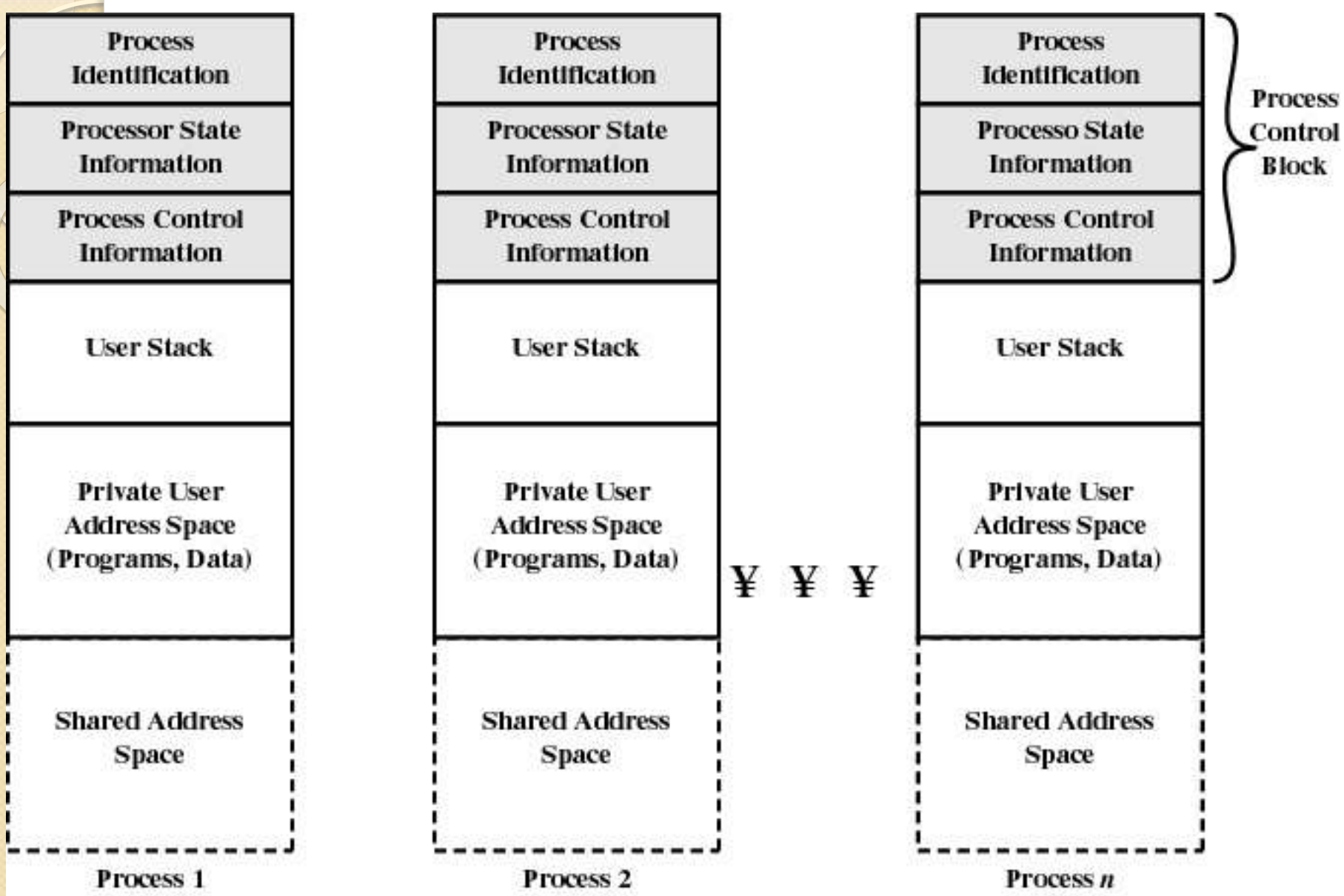


Figure 3.12 User Processes in Virtual Memory

9. Processor State Information

- Contents of processor registers
  - User-visible registers
  - Control and status registers
  - Stack pointers
- Program status word (PSW)
  - contains status information
  - Example: the EFLAGS register on Pentium machines

Modes of Execution

- User mode
  - Less-privileged mode
  - User programs typically execute in this mode
- System mode, control mode, or kernel mode
  - More-privileged mode
  - Kernel of the operating system

Process Creation

- Assign a unique process identifier
- Allocate space for the process
- Initialize process control block
- Set up appropriate linkages
  - Ex: add new process to linked list used for scheduling queue
- Create or expand other data structures
  - Ex: maintain an accounting file

When to Switch a Process ?

- Clock interrupt
  - process has executed for the maximum allowable time slice
- I/O interrupt
- Memory fault
  - memory address is in virtual memory so it must be brought into main memory

When to Switch a Process

- Trap
  - error occurred
  - may cause process to be moved to Exit state
- Supervisor call
  - such as file open

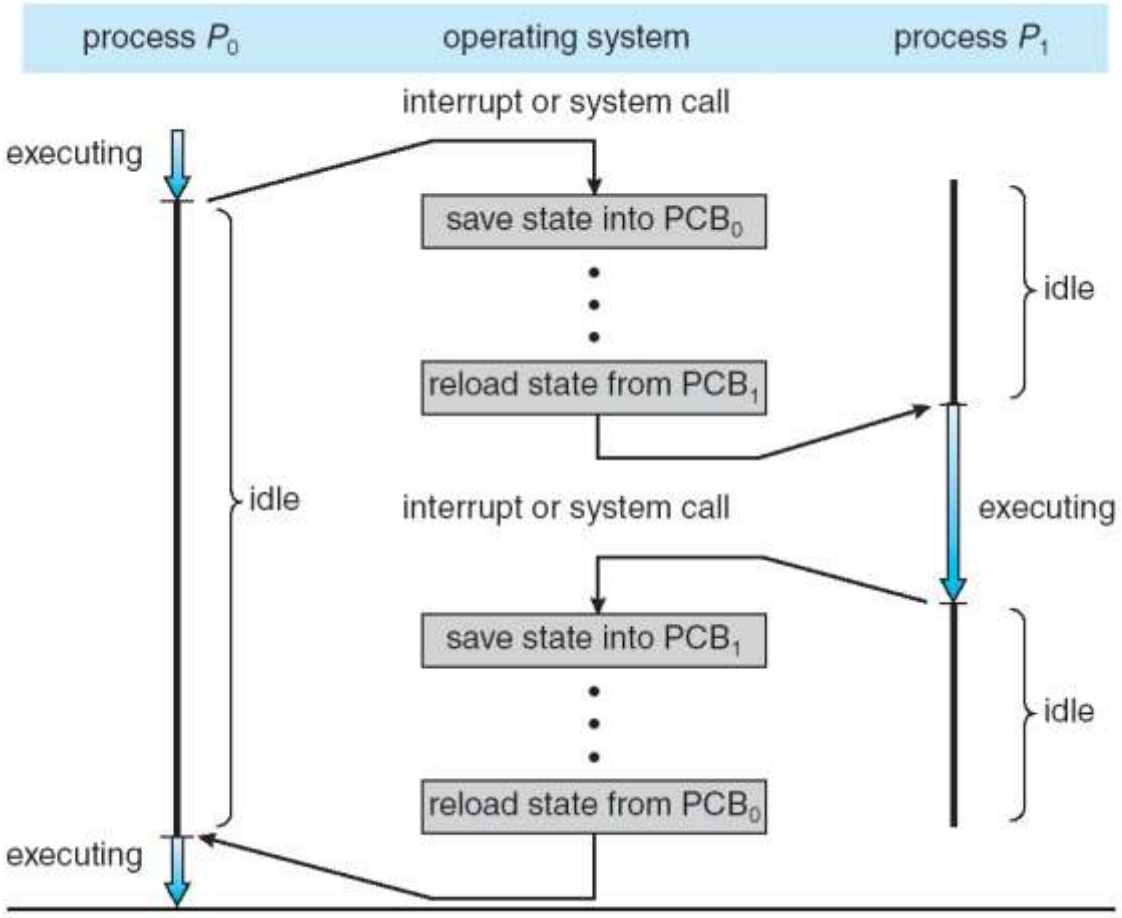
Change of Process State

- Save context of processor including program counter and other registers
- Update the process control block of the process that is currently running
- Move process control block to appropriate queue - ready, blocked
- Select another process for execution

Change of Process State

- Update the process control block of the process selected
- Update memory-management data structures
- Restore context of the selected process

Switching between the Processes (context switch)



10. Execution of the Operating System

- Non-process Kernel
  - execute kernel outside of any process
  - operating system code is executed as a separate entity that operates in privileged mode
- Execution Within User Processes
  - operating system software within context of a user process
  - process executes in privileged mode when executing operating system code

Execution of the Operating System

- Process-Based Operating System
  - major kernel functions are separate processes
  - Useful in multi-processor or multi-computer environment

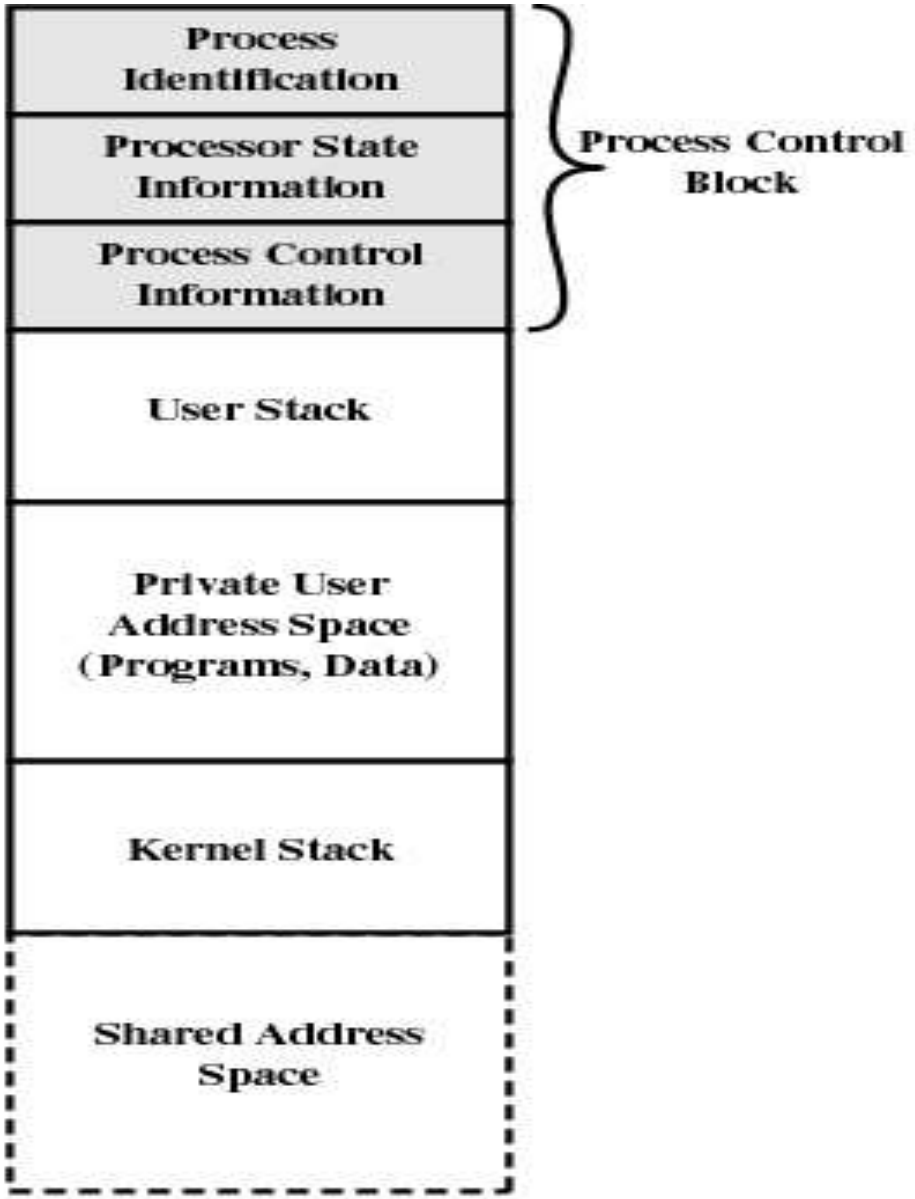


Figure 3.15 Process Image: Operating System Executes Within User Space

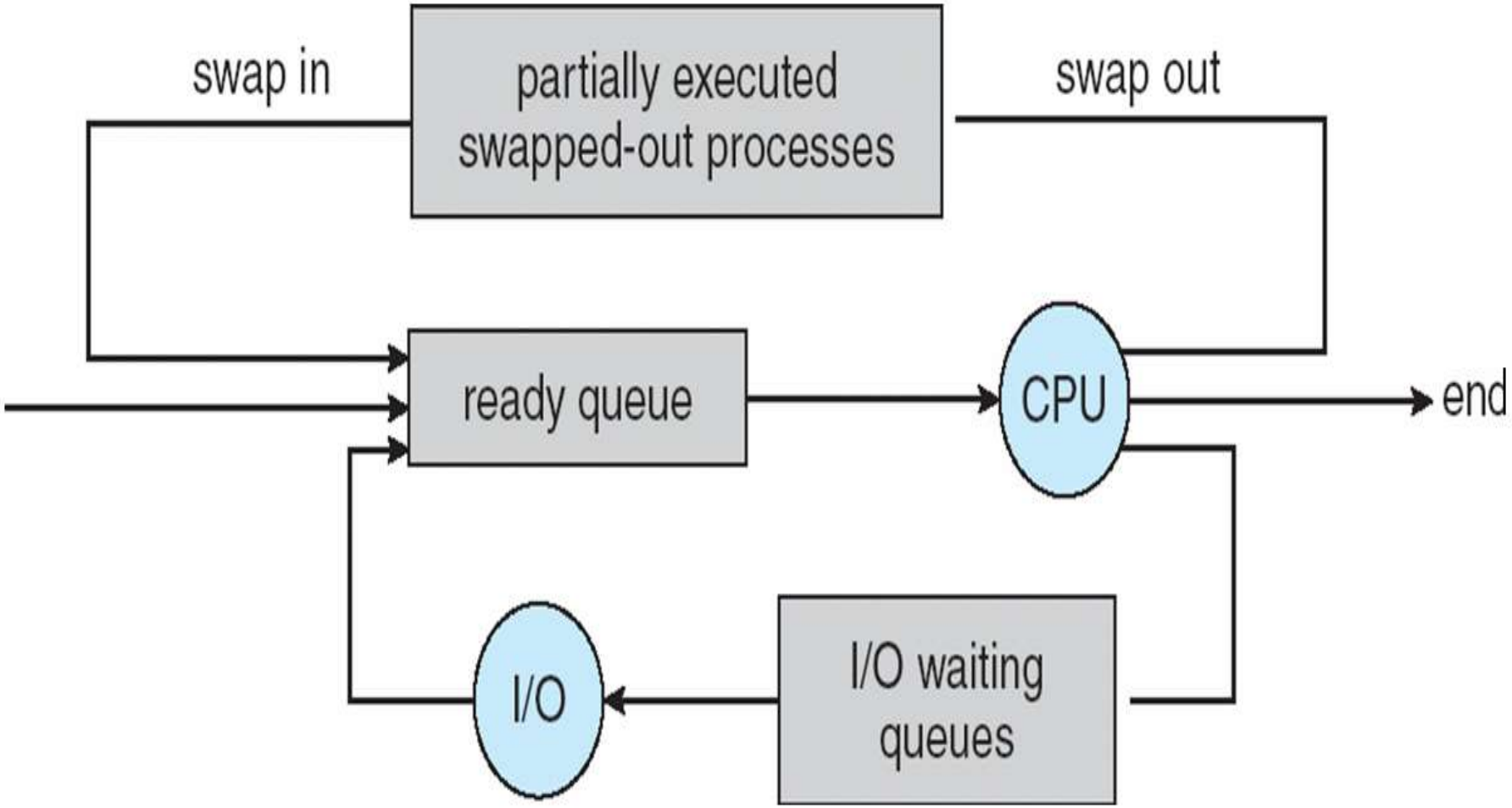
II. Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

Schedulers (Cont)

- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (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

Addition of Medium Term Scheduling



12. Process Scheduling

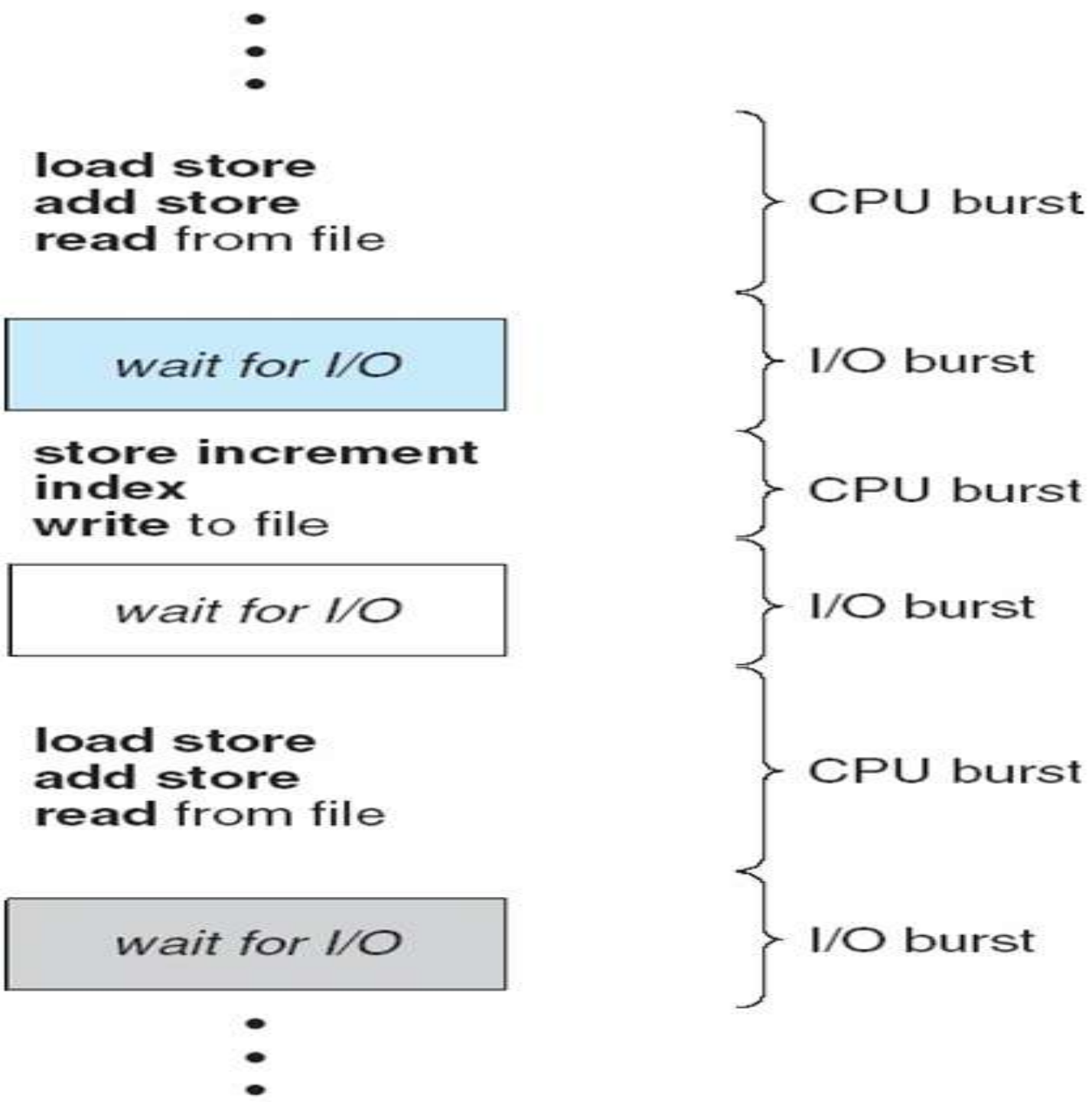
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Operating Systems Examples
- Algorithm Evaluation

A. Scheduling Objectives

- To introduce process scheduling, which is the basis for multi-programmed operating systems
- To describe various process-scheduling algorithms
- To discuss evaluation criteria for selecting a process-scheduling algorithm for a particular system

B. Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- Process execution time consists of CPU execution time and I/O wait time
  - **CPU burst** distribution
  - CPU – I/O Burst Cycle


load store
add store
read from file
wait for I/O
store increment
index
write to file
wait for I/O
load store
add store
read from file
wait for I/O
CPU burst
I/O burst
CPU burst
I/O burst
CPU burst
I/O burst

C. CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **non-preemptive**
- All other scheduling is **preemptive**

Dispatching

- Dispatcher module gives control of the CPU to the process selected
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start execution of another

D. Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # 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)

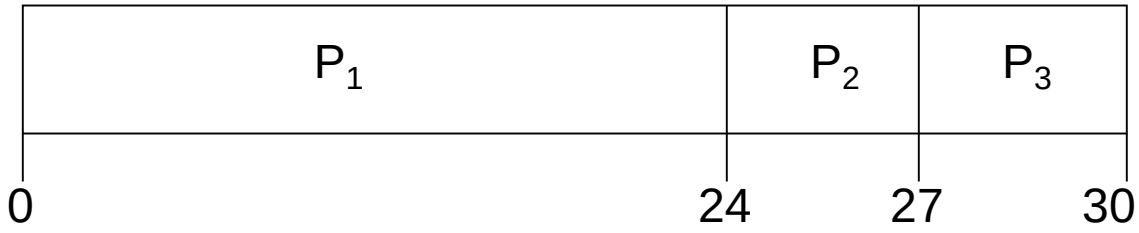
E. Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

1. First-Come, First-Served (FCFS) Scheduling

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:



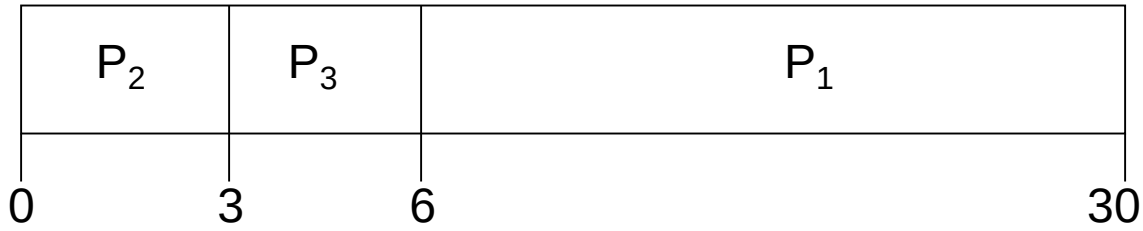
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1$$

- 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
- *Convoy effect* short process behind long process

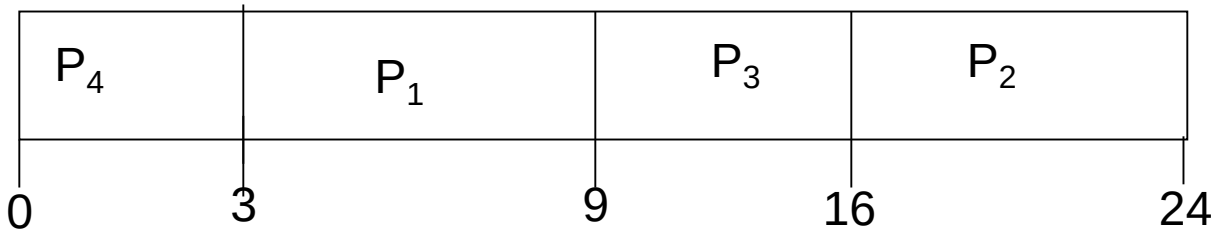
2. Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use 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

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

3. Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

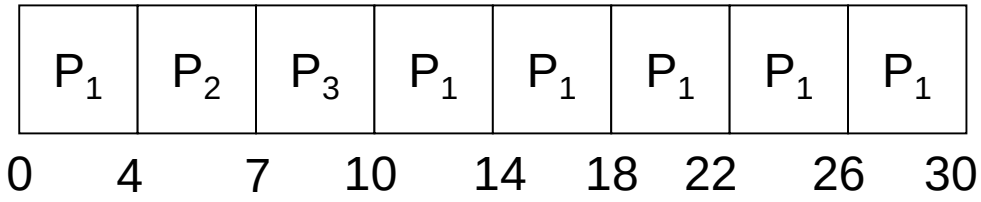
4. Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

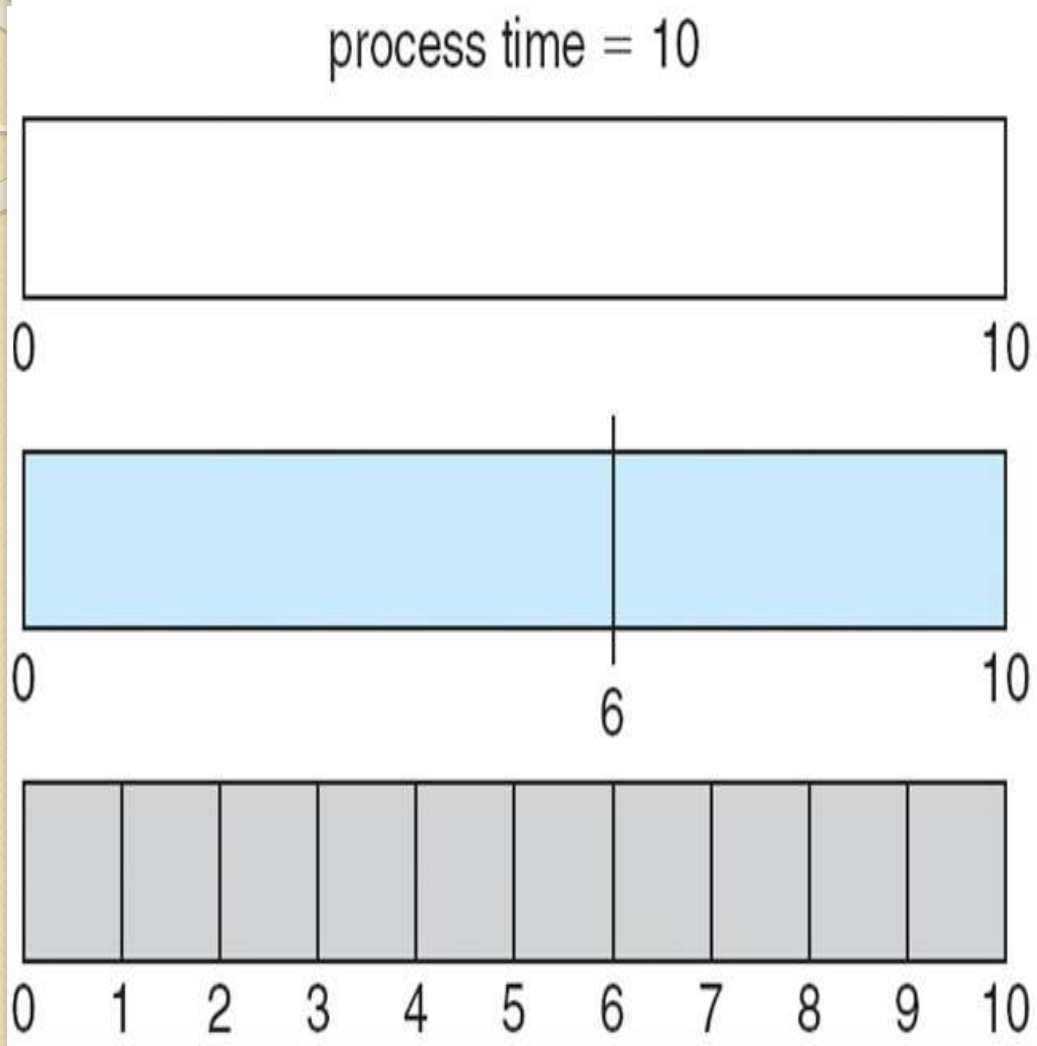
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

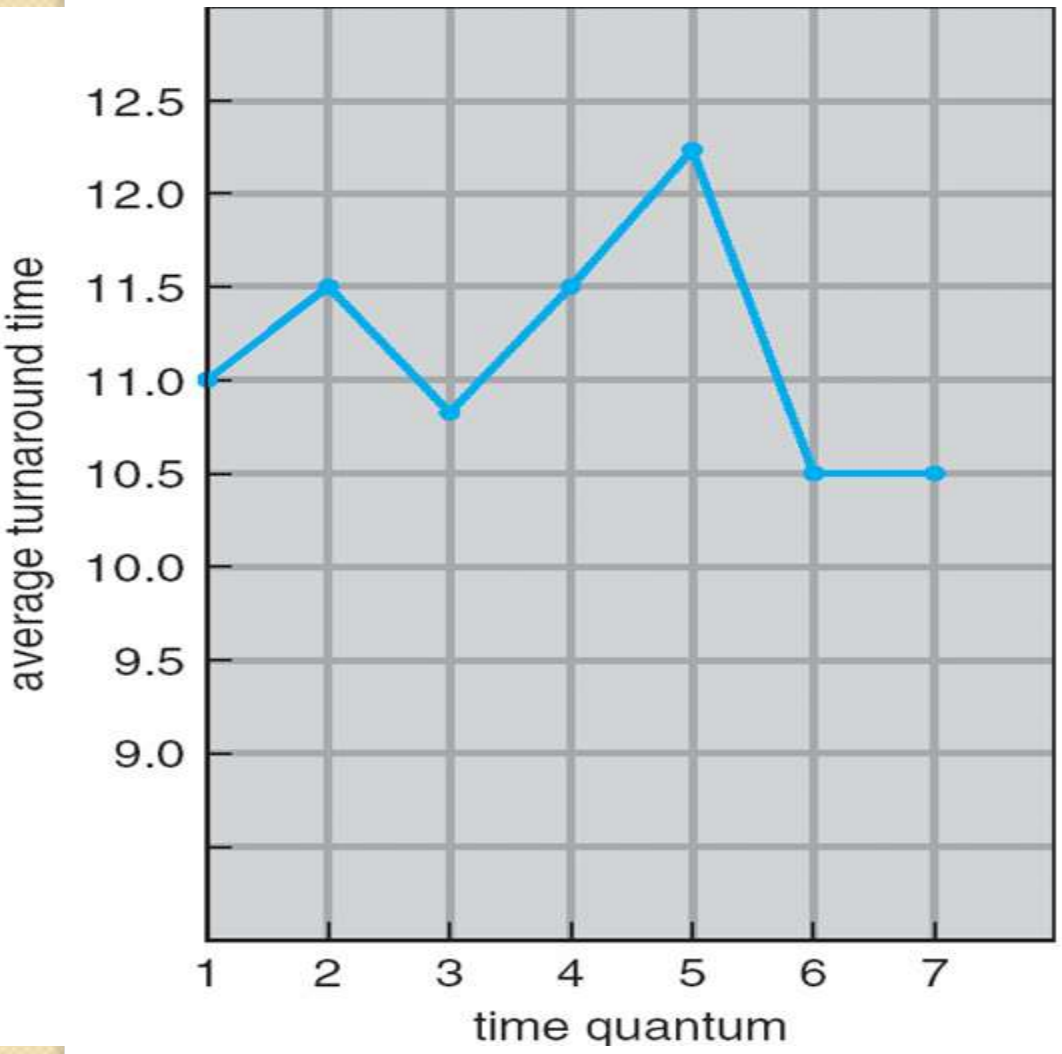


- Typically, higher average turnaround than SJF, but better *response*

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum

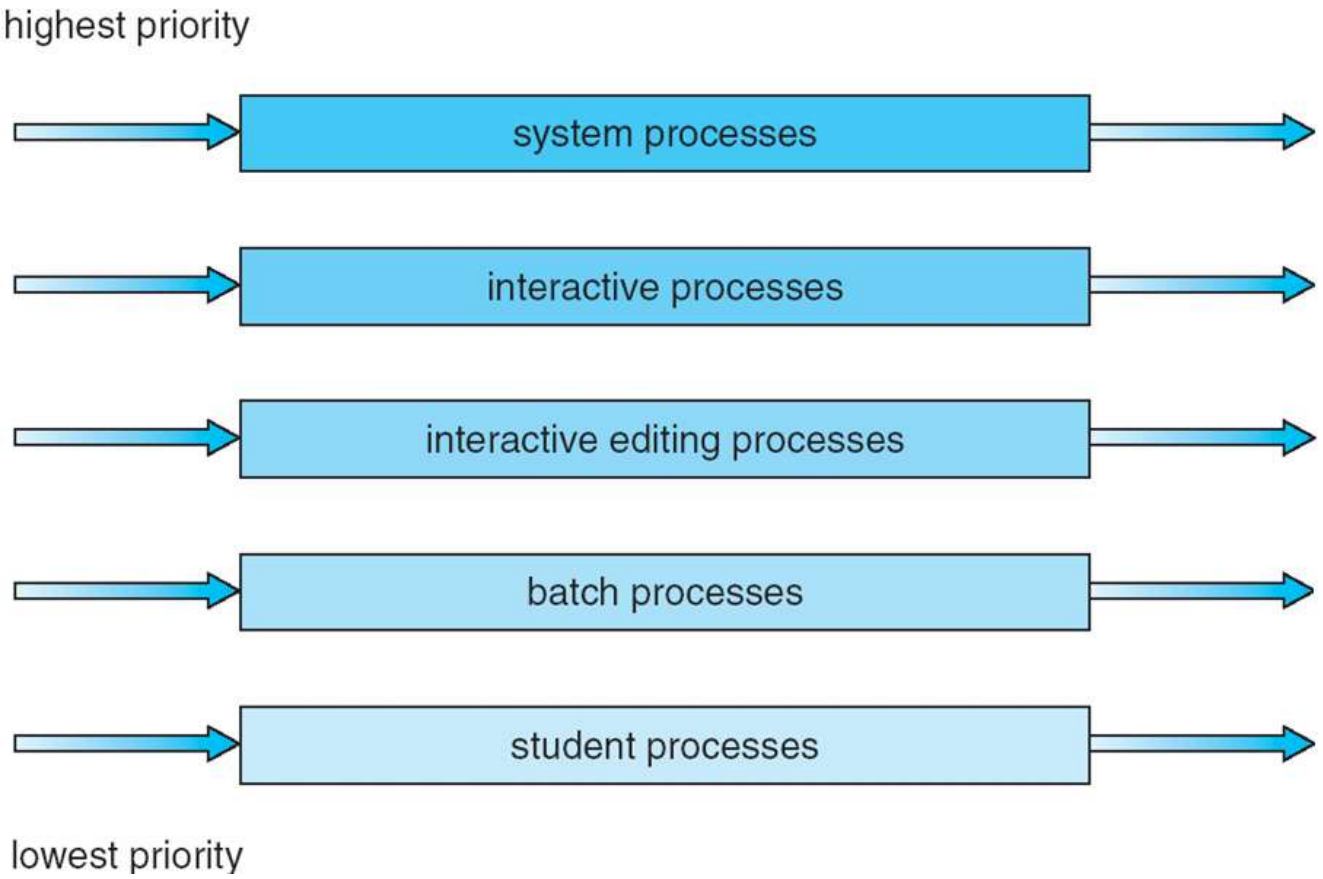


process	time
P_1	6
P_2	3
P_3	1
P_4	7

5. Multilevel Queue

- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Queue Scheduling



6. Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example:

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Multilevel Feedback Queues