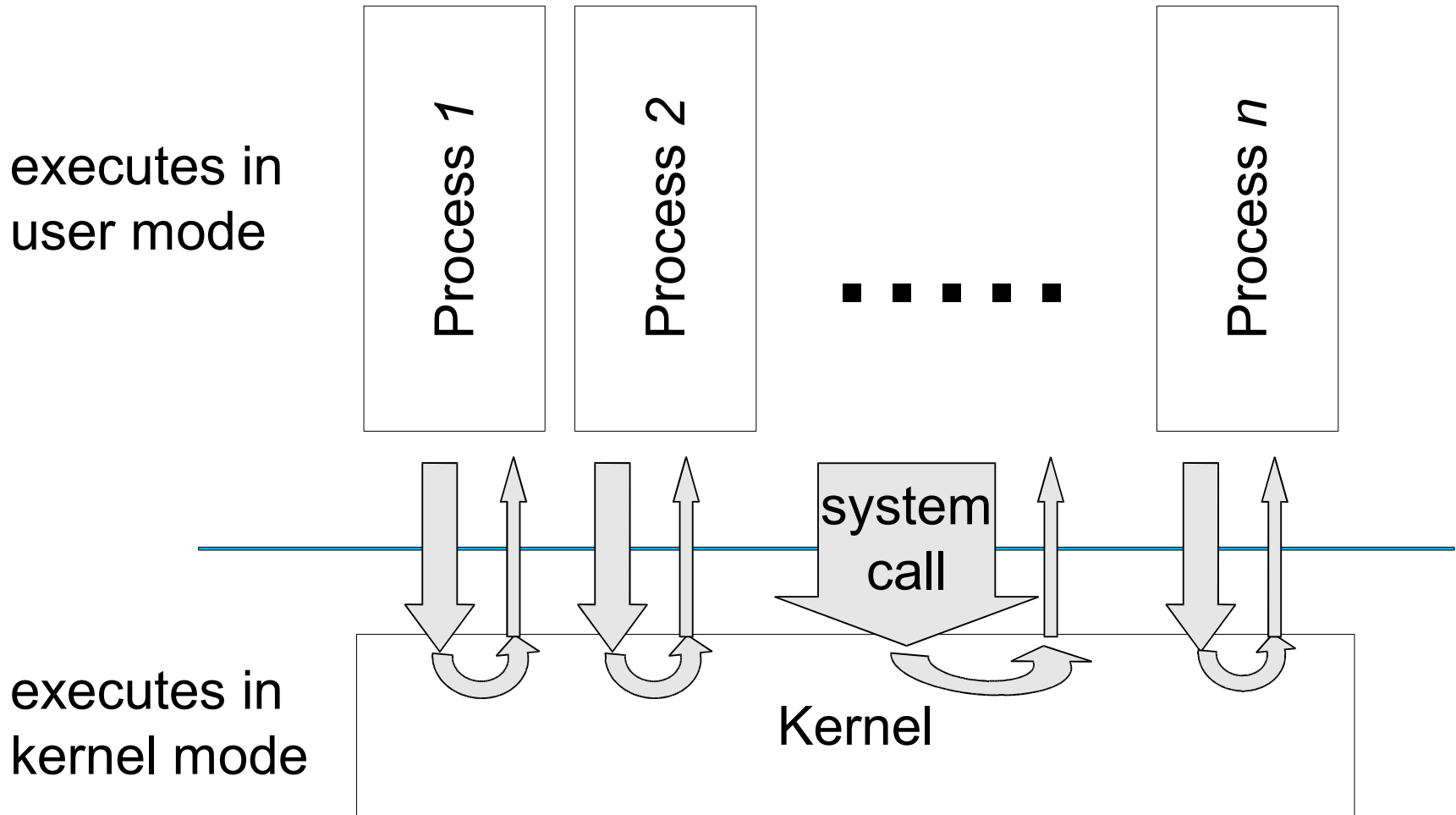# Process Description and Control

Process

- A program in execution (running) on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by
  - a at least one sequential thread
  - an associated set of system resources
  - a current state of CPU (and other resources)

Operating System and Processes

- Interleave the execution of multiple processes
  - maximize processor utilization
  - providing reasonable response time
- Allocate resources to processes
- Support interprocess communication and user creation of processes

Operating System and Processes

executes in user mode

executes in kernel mode

Process 1

Process 2

. . . . .

Process *n*

system call

Kernel

Process Creation

Table 3.1   Reasons for Process Creation

New batch job	The operating system is provided with a batch job control stream, usually on tape or disk. When the operating system is prepared to take on new work, it will read the next sequence of job control commands.
Interactive logon	A user at a terminal logs on to the system.
Created by OS to provide a service	The operating system can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).
Spawned by existing process	For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes.

Process Termination

Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded	The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time ("wall clock time"), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.
Memory unavailable	The process requires more memory than the system can provide.
Bounds violation	The process tries to access a memory location that it is not allowed to access.
Protection error	The process attempts to use a resource such as a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.
Arithmetic error	The process tries a prohibited computation, such as division by zero, or tries to store numbers larger than the hardware can accommodate.

Process Termination

Time overrun	The process has waited longer than a specified maximum for a certain event to occur.
I/O failure	An error occurs during input or output, such as inability to find a file, failure to read or write after a specified maximum number of tries (when, for example, a defective area is encountered on a tape), or invalid operation (such as reading from the line printer).
Invalid instruction	The process attempts to execute a nonexistent instruction (often a result of branching into a data area and attempting to execute the data).
Privileged instruction	The process attempts to use an instruction reserved for the operating system.
Data misuse	A piece of data is of the wrong type or is not initialized.
Operator or OS intervention	For some reason, the operator or the operating system has terminated the process (for example, if a deadlock exists).
Parent termination	When a parent terminates, the operating system may automatically terminate all of the offspring of that parent.
Parent request	A parent process typically has the authority to terminate any of its offspring.

Trace of Process

- Sequence of instruction (addresses) for each process

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

scheduler and dispatcher

- scheduler: part of the kernel that decides which is the next process executed by the CPU

- dispatcher: part of the kernel that set up the CPU registers to execute the process
  - restore the context for the process

- in moder OS usually scheduling and dispatching are performed together by the same routine
  - we use "scheduler" or "dispatcher" depending on the aspect we need to emphasize
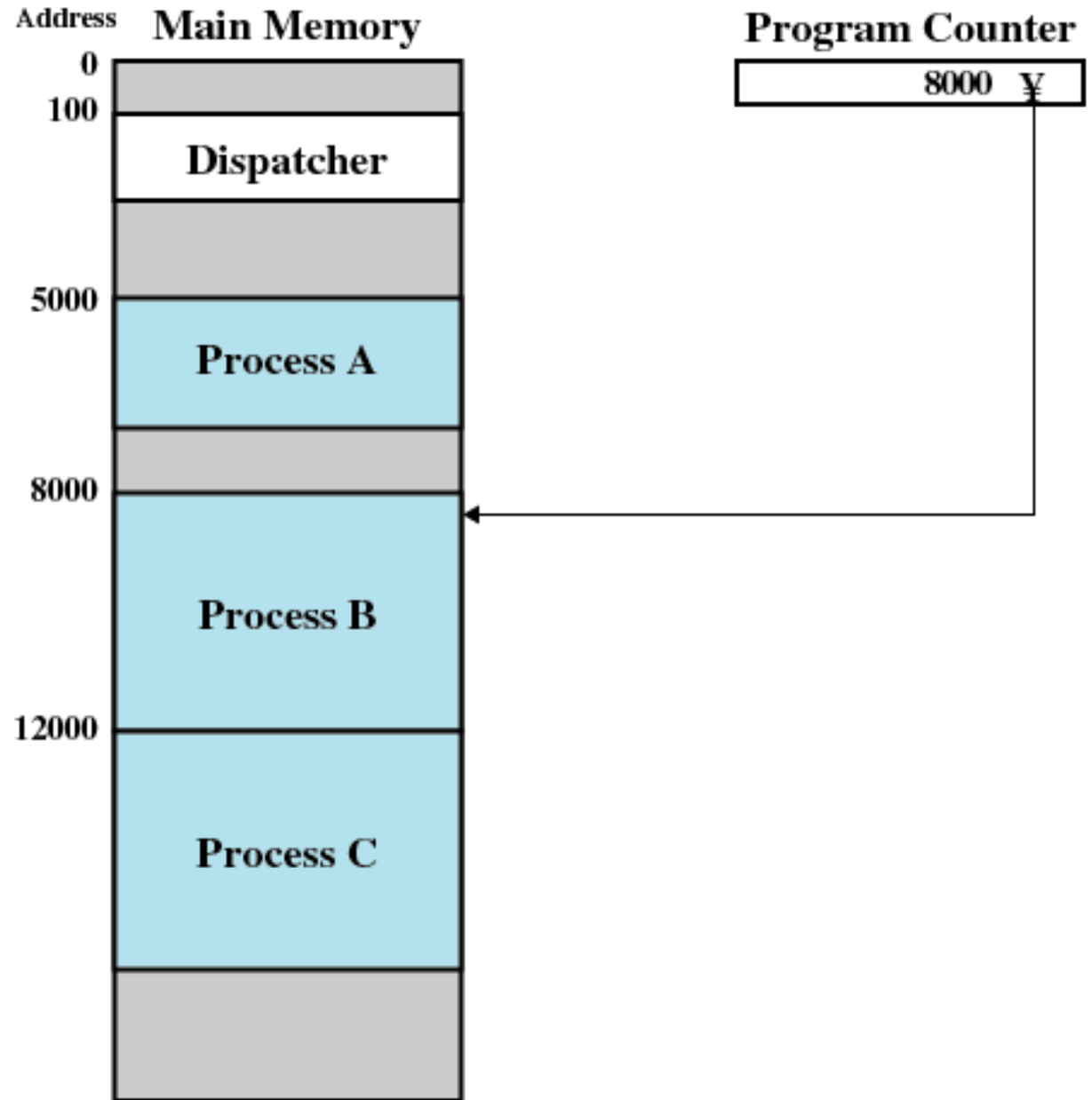
Dispatcher

- The *dispatcher* switches the processor from one process to another (**process switch**)

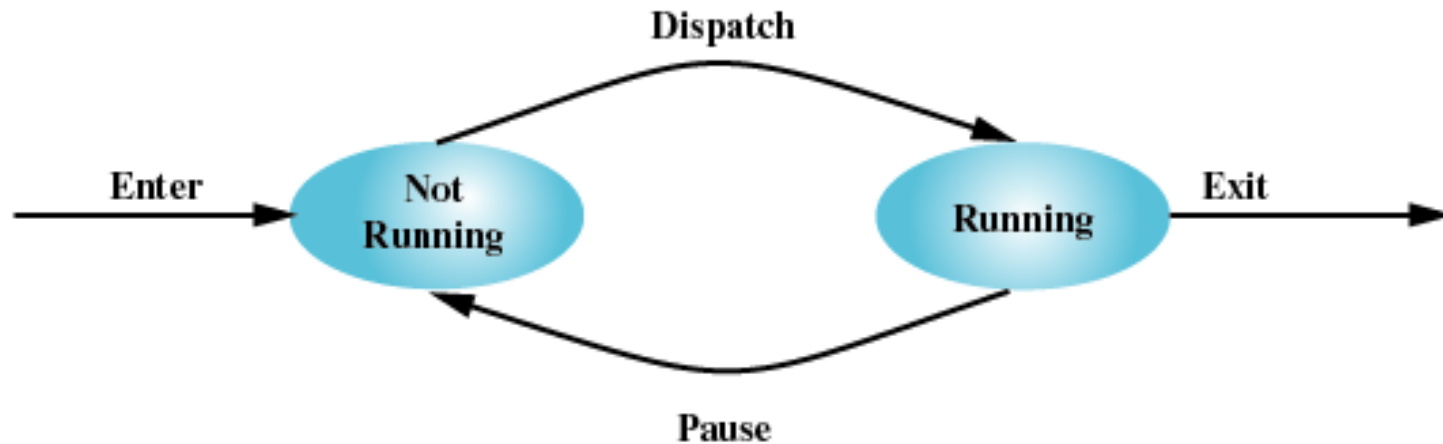|			
1	5000	27	12004
2	5001	28	12005
3	5002	----------------- Time out	
4	5003	29	100
5	5004	30	101
6	5005	31	102
----------------- Time out		32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	103	36	5007
11	104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	----------------- Time out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

100 = Starting address of dispatcher program

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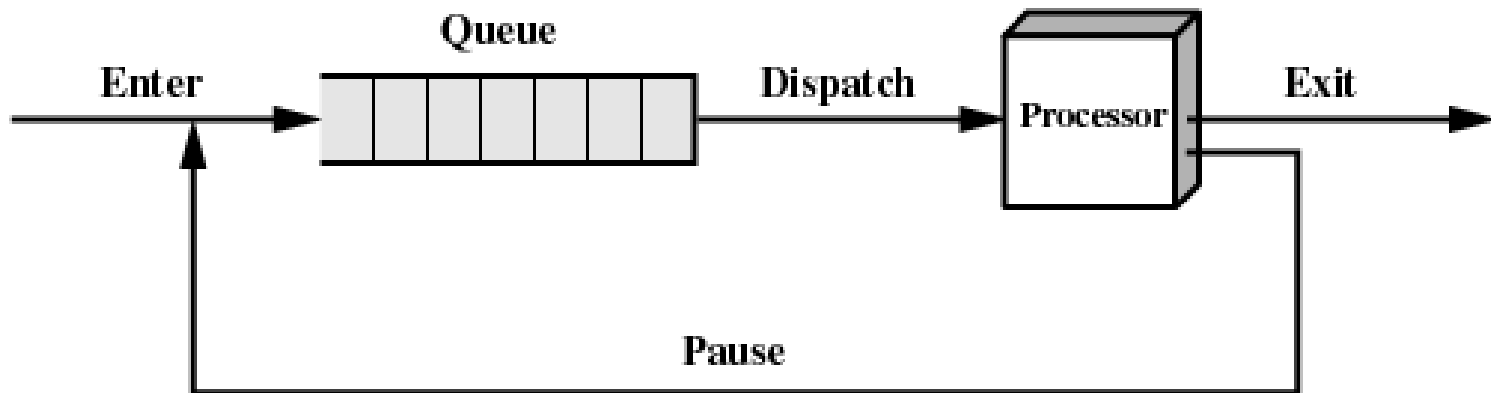second and fourth columns show address of instruction being executed

Processes and Memory

Address	Main Memory
0	
100	Dispatcher
5000	
	Process A
8000	
	Process B
12000	
	Process C

Program Counter
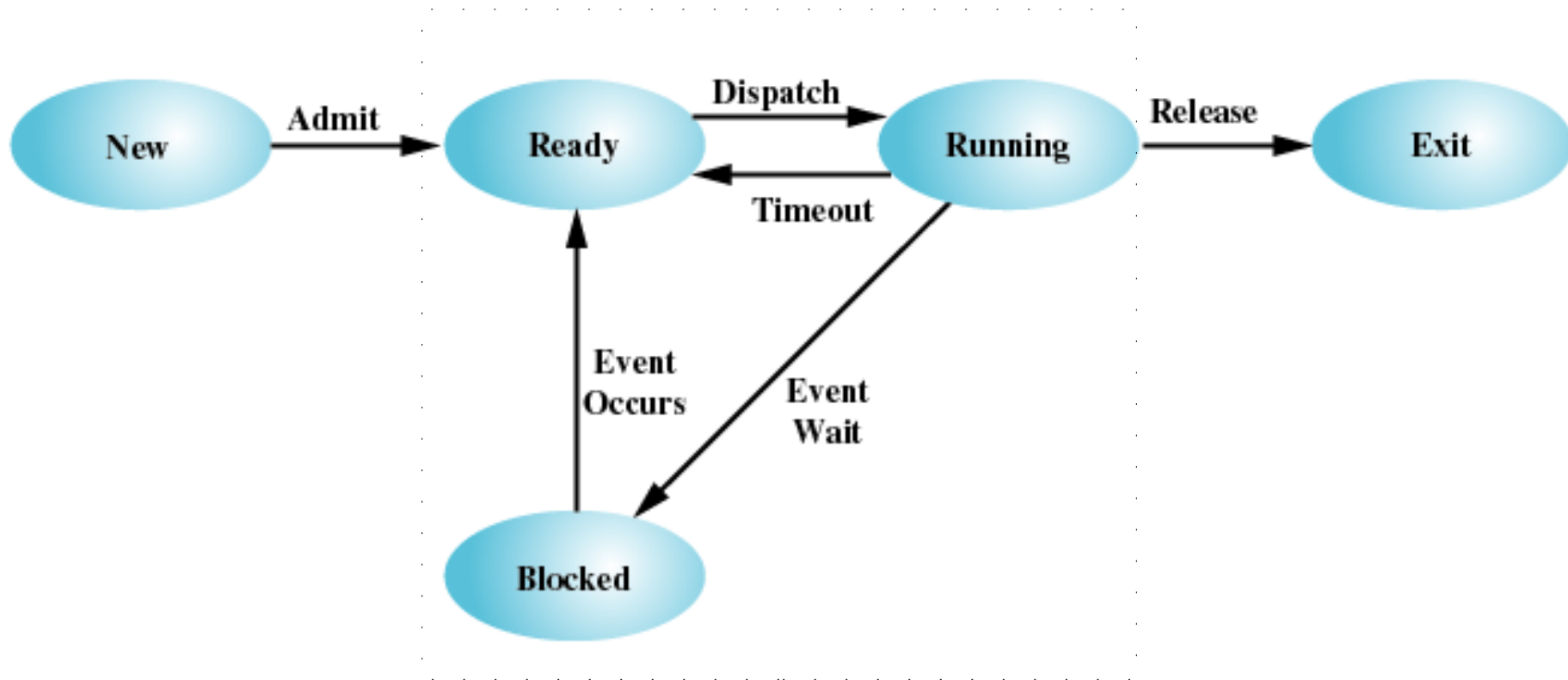
8000

Two-State Process Model

(a) State transition diagram

(b) Queuing diagram

Five-State Process Model

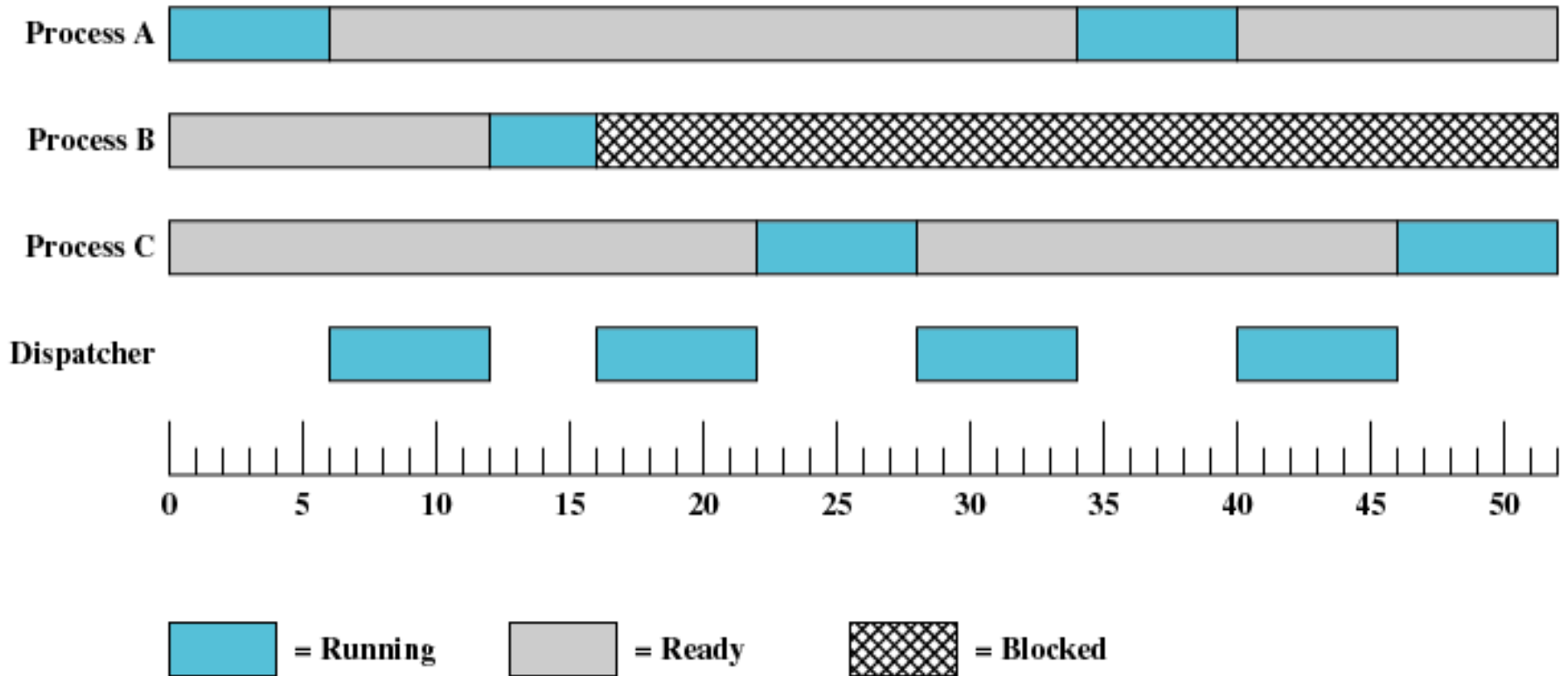
Figure 3.6   Five-State Process Model
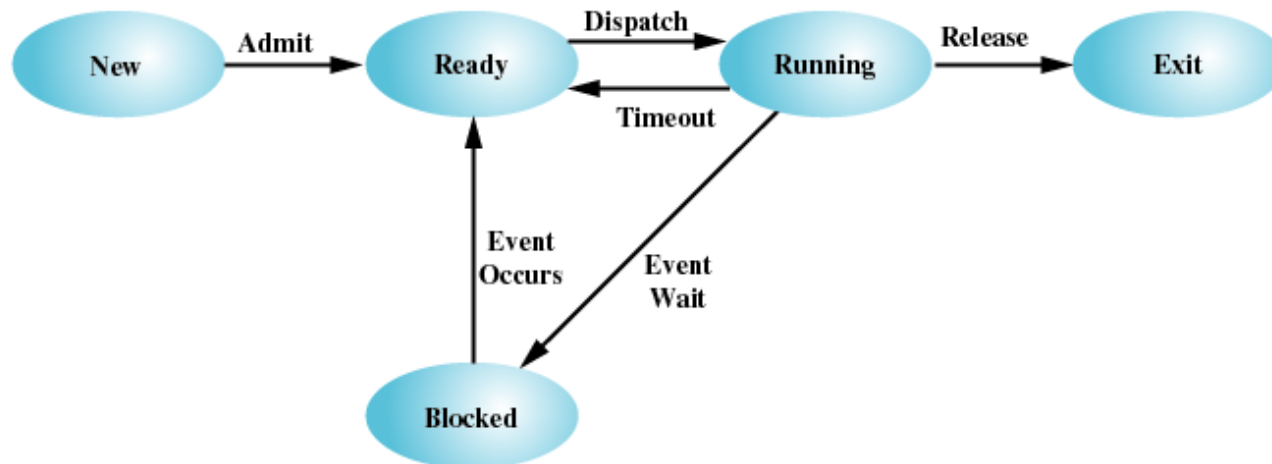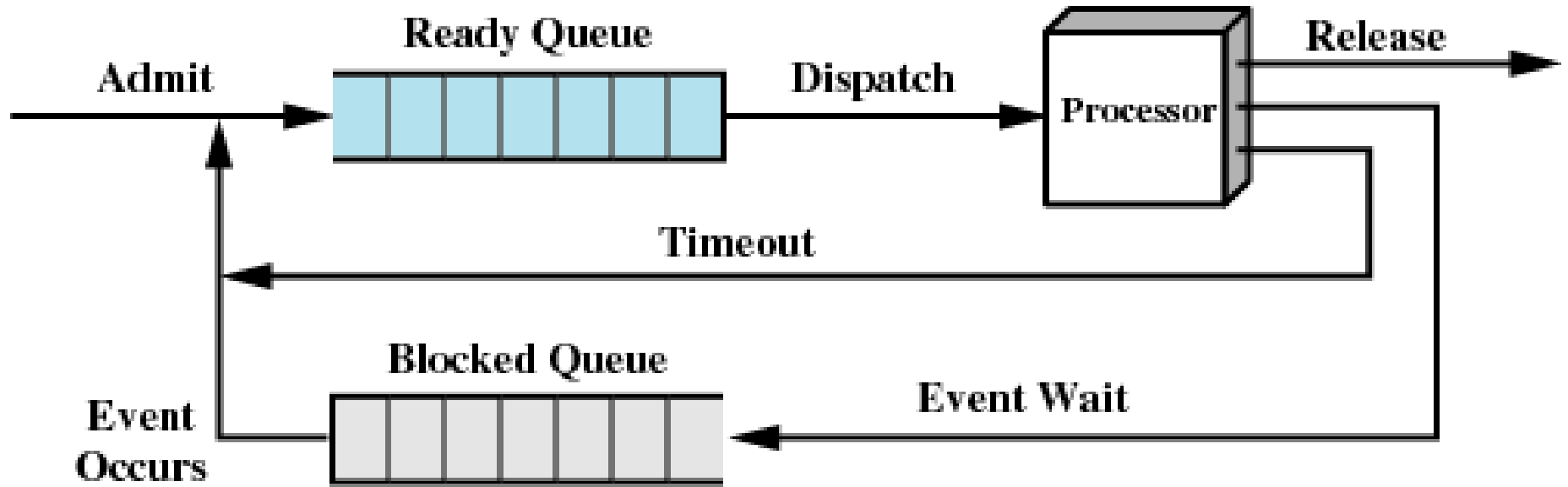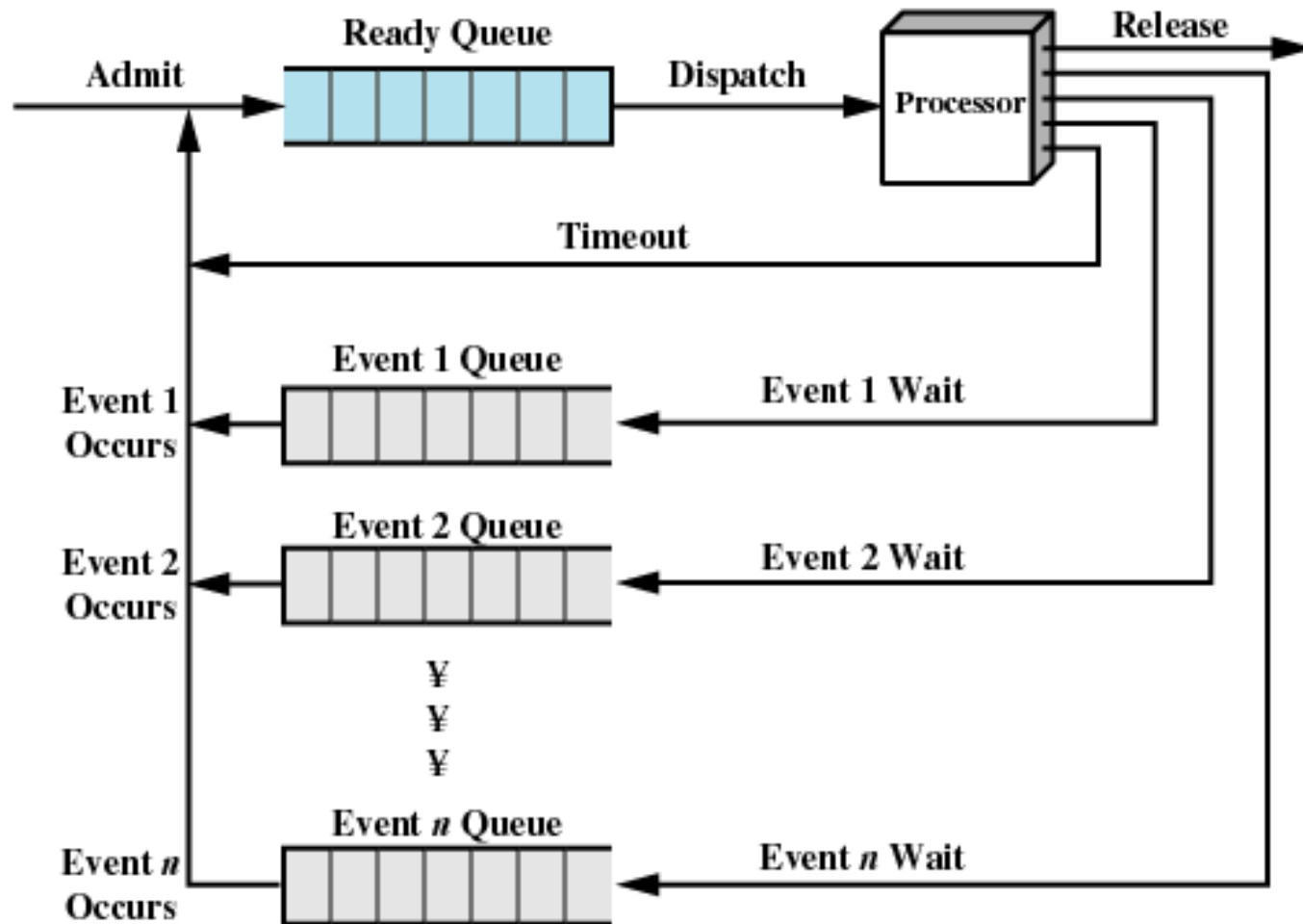
Process States

Figure 3.7   Process States for Trace of Figure 3.4

One sequential I/O device

Many sequential I/O devices

Admit → **Ready Queue** → **Dispatch** → **Processor** → **Release**

Timeout

Event 1 Occurs ← **Event 1 Queue** ← **Event 1 Wait**

Event 2 Occurs ← **Event 2 Queue** ← **Event 2 Wait**

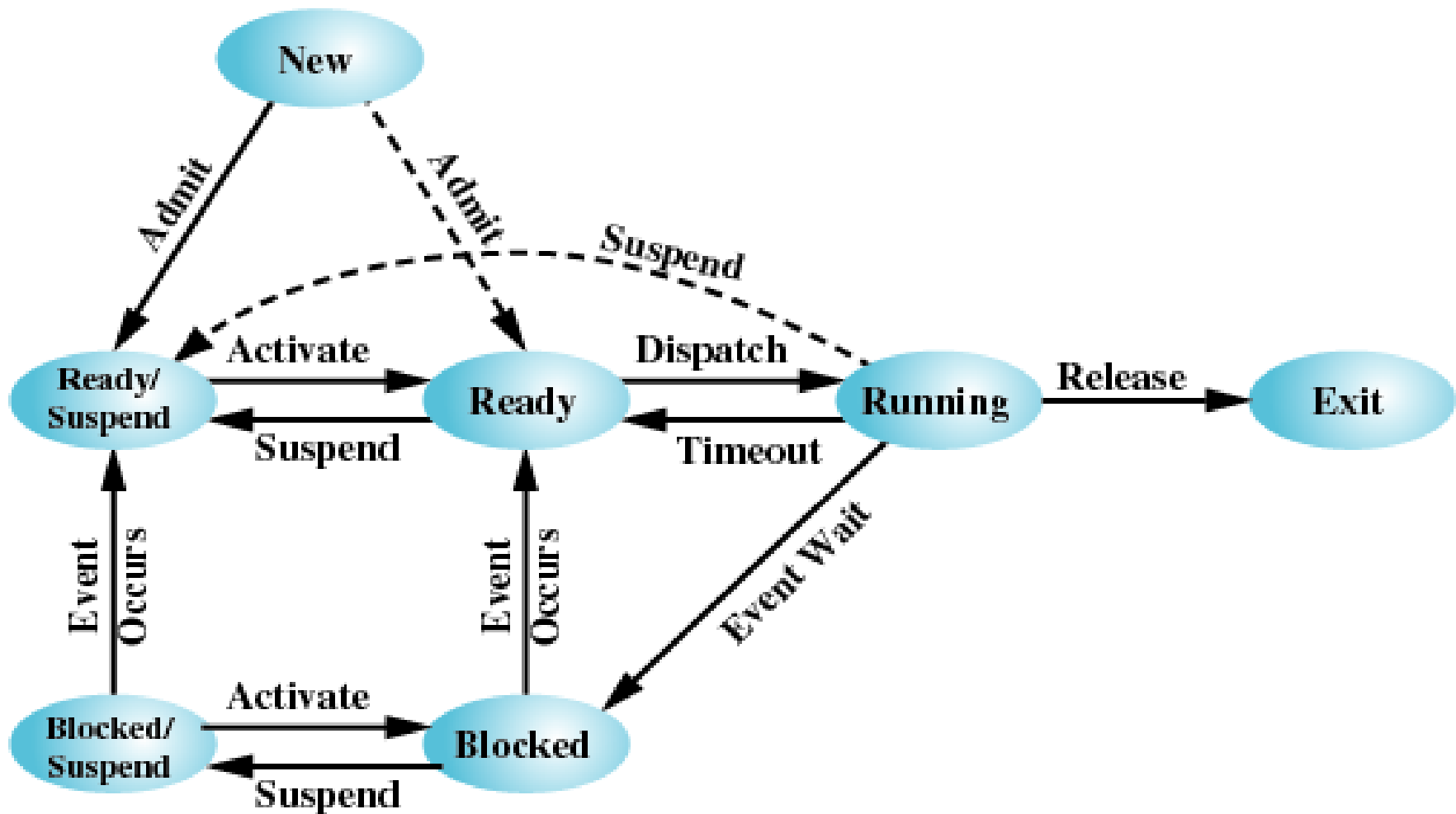**Event n Occurs** ← **Event n Queue** ← **Event n Wait**

(b) Multiple blocked queues

Suspended Processes

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

Two New States

(b) With Two Suspend States

Several Reasons for Process Suspension

Swapping	The operating system needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The operating system may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.
Parent process request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendents.

process description

Process Image

Table 3.4   Typical Elements of a Process Image

User Data
> The modifiable part of the user space. May include program data, a user stack area, and programs that may be modified.

User Program
> The program to be executed.

System Stack
> 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.

Process Control Block
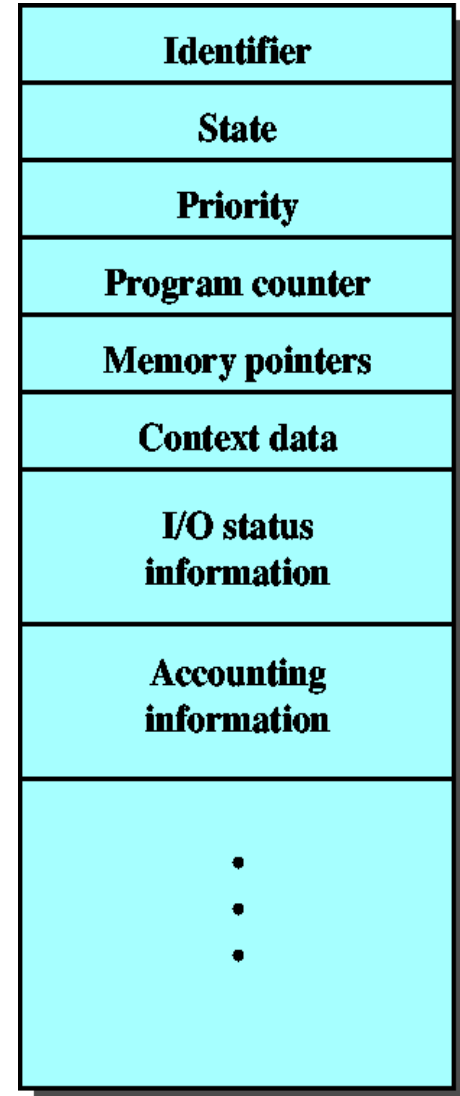> Data needed by the operating system to control the process (see Table 3.5).

OS controls assignment of resources to processes
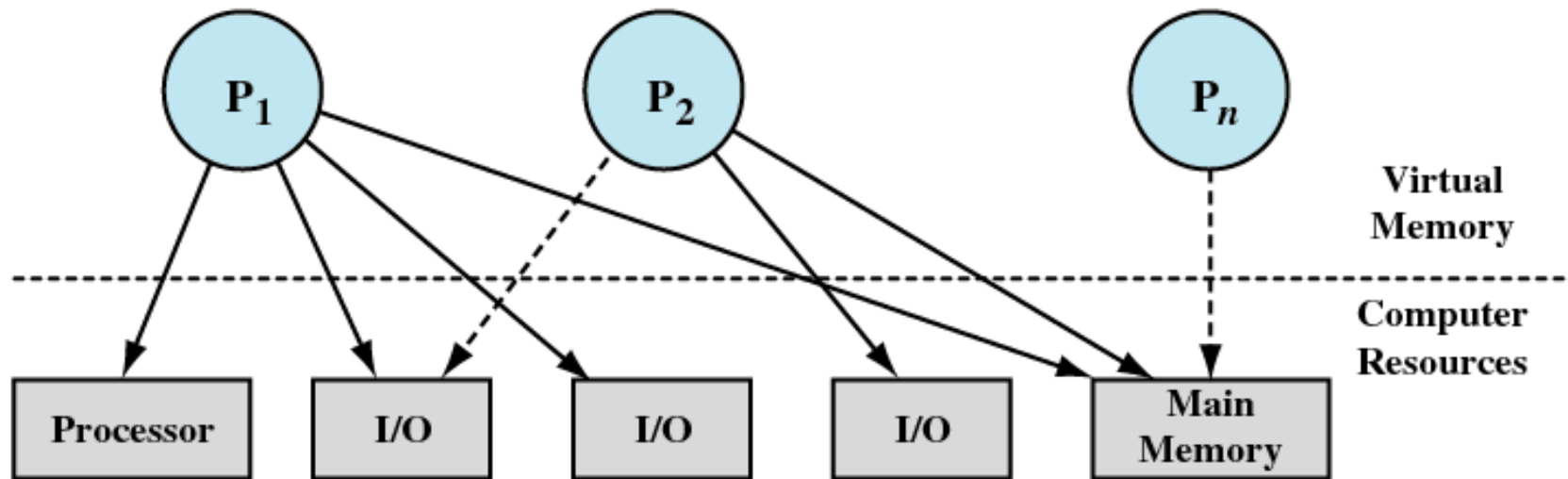
- the OS maintains many data structures for this purpose

Process Control Block (PCB)

- contains data about **one** process
  - one instance for each process
- contains all the information we need to...
  - ...interrupt a running process
  - ...resume execution
- created and managed by the operating system
- allows support for multiple processes

Identifier
State
Priority
Program counter
Memory pointers
Context data
I/O status information
Accounting information
⋮

Process Elements in PCB

they largely depend on the OS

- Process Identifier (PID)
- State (ready, blocked, etc.)

    – if blocked, events the process is waiting for

- Priority (for the scheduler)
- saved CPU registers and PC (a.k.a. context)
- Memory pointers (program, data, stack, tables, etc.)
- I/O status information (open files, outstanding I/O requests, inter-processes comunication, etc)
- Accounting information (CPU time used, limits, etc.)
- user that owns the process, and/or privileges
- process that created the process

Data Structuring

- ## PCB – PCB pointers
  - parent-child (creator-created) relationship with another process

- ## queues
  - all processes in a waiting state for a particular priority level may be linked in a queue.

Process Creation

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

PCB synonyms

- process descriptor
- task control block
- task descriptor

linux

- task_struct

PCB related data structures

- process table
- memory tables
- I/O tables
- file tables

Process Table

- one entry for each process
- contains a minimal amount of information needed to activate the process
  - usually a "pointer" to the PCB
  - it may be a complex data structure (tree, hash table, ecc.)

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

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

File Tables

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

process control

mode switch

- two cases
  - user-mode → kernel-mode
    - triggered by an interrupt or a system call
    - set cpu in priviledged mode
    - may save the cpu state
  - kernel-mode → user-mode
    - triggered by the kernel when it "decides" to resume process execution
    - set cpu in unpriviledged mode
    - may restore all or part of the cpu state

mode switch

- it is not a process switch
- not necessarily implies a process switch
- many system calls require a mode switch

process switch (dispatching)

- a process switch assigns the cpu to a different process
  - before: P_1 running, P_2 ready
  - after: P_1 not running, P_2 running

- it is performed in kernel-mode
  - it requires two mode switches
    1 user-mode $\rightarrow$ kernel-mode before the process switch
      - triggered by interrupt, trap or system call
      - kernel possibly fulfill a request (e.g. I/O)
    2 kernel-mode $\rightarrow$ user-mode after the process switch
      - into the process chosen by the kernel (scheduler)

process switch

- it modifies OS data structures
  - set proper state in PCB of P_1 and P_2
  - update queues
    - move P_1 into the appropriate queue
    - move P_2 out of the ready queue
  - update CPU memory tables for the image of P_2
- the next mode switch (kernel-mode $\rightarrow$ user-mode) will restore the cpu state of P_2

typical situations for switching mode and/or process

- Clock interrupt
  - process has executed for the maximum allowable time slice
  - always switch process

- system call
  - process switch when it is a blocking I/O request
  - OS may check if other processes have greater priority and possibly switch process

typical situations for switching mode and/or process

- I/O interrupt
  - a blocked process may become ready
  - process switch depends on OS policies and priorities

- Trap
  - error or exception
    - current process usually die and process is switched
  - memory fault (virtual memory)
    - current process becomes blocked (waiting for the page) and process is switched
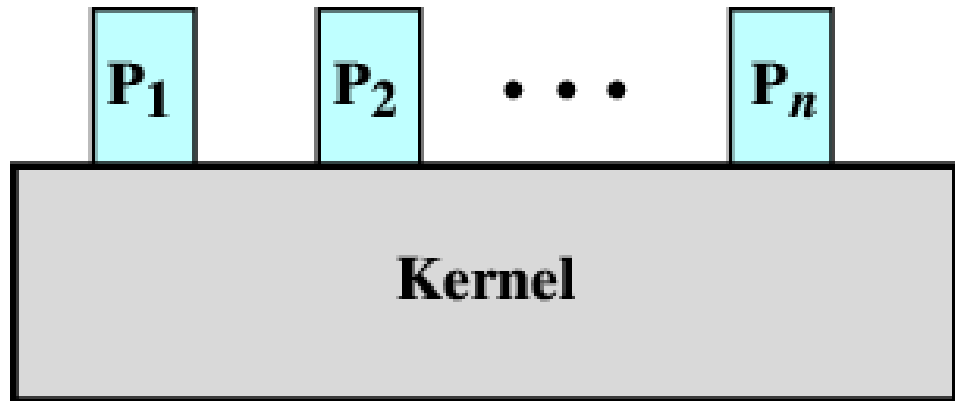
execution of the OS

- the OS is executed by the cpu

- is the OS a process?

- several architectures are possible
  - non-process kernel
  - kernel execution within user processes
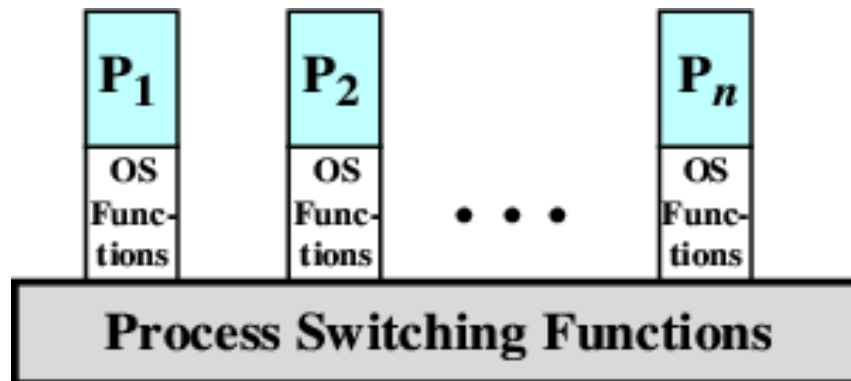 - process-based operating system

non-process kernel

- kernel is executed outside of any process
  - when it is executed cpu is in privileged mode
  - kernel has its own "memory space"
    - there is not a OS process anyway!
    - inefficient
  - kernel implements tricks to access the images of processes
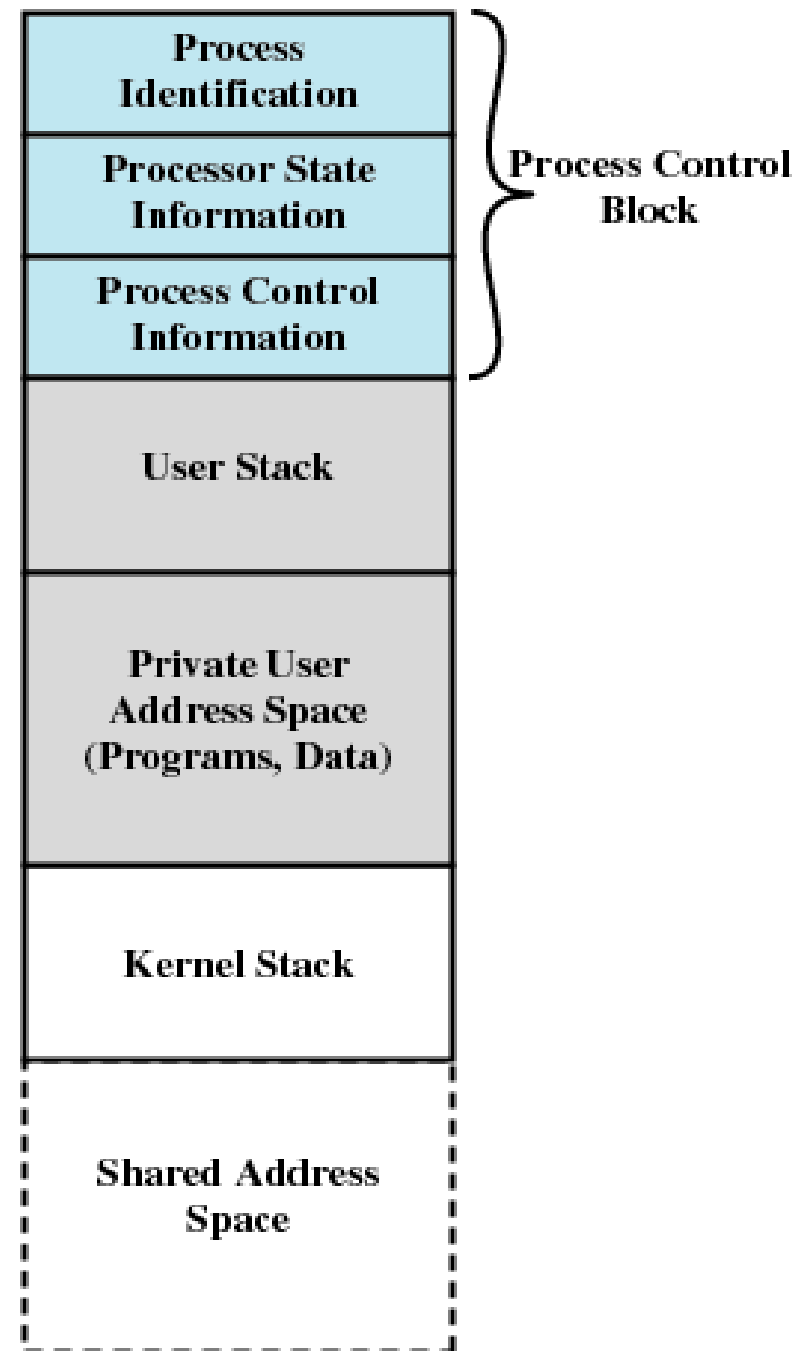 - obsolete

Execution Within User Processes

- large part of the kernel executes within the context of a user process

  - no reconfiguration of CPU memory table is needed (efficient)

  - kernel execution need only a mode switch

  - Process executes in privileged mode when executing operating system code

P₁	P₂		Pₙ
P₁	**P₂**	• • •	**P_n**
OS Func- tions	OS Func- tions		OS Func- tions

Process Switching Functions

Execution Within User Processes

- each process has its own **image**

- image contains also
  - kernel stack
  - kernel program
  - kernel data

- kernel program and data are shared by all images
  - kernel mode is needed to read and write them

Process Control Block
Process Identification
Processor State Information
Process Control Information

User Stack

Private User Address Space (Programs, Data)

Kernel Stack

Shared Address Space

Execution Within User Processes

- to fulfill a system call, interrupt or trap
  - mode is switched
  - process is not switched
  - current usable image remain the same
  - both kernel data and current process data can be accessed

- a process switch occours if and only if a new process is scheduled and dispatched
  - process switch is the only activity that can be considerd outside of any process

Execution Within User Processes

- All UNIX systems adopt this approach

process-based OS (microkernel)

- Implement the os as a collection of system processes
- each system call pays a process switch penalty and an "inter-process message" penalty
- process switch and inter-process comunication are the only activities that can be considerd outside of any process

| P_1 | P_2 | • • • | P_n | OS_1 | • • • | OS_k |

Process Switching Functions
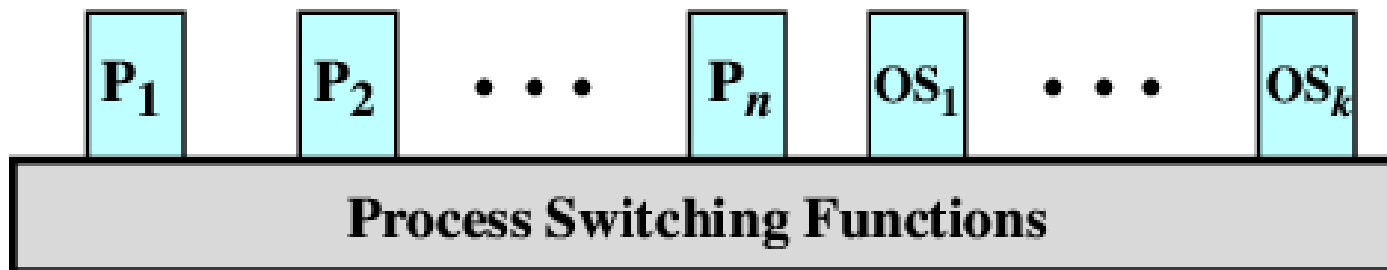
process-based OS: design choice

- may processes run in kernel mode to access hardware?
- drivers are implemented in the kernel or as processes?

- consider the efficiency of the alternatives of an I/O operation
  - how many inter-processes messages?
  - how many mode switches?
  - how many process switches?
  - how many times dispatcher run?

process-based OS (microkernel)

- modular and robust

- flexible
  - services may be added, removed or distributed

- quite inefficient

- portable
  - only the microkernel depends on hardware

- Windows adopts this approach