Process

Description and Control

Prof. Siddharth Shah

Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behavior of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.

Requirements of an OS

- Fundamental Task: Process Management
- The Operating System must
  - Interleave the execution of multiple processes
  - Allocate resources to processes, and protect the resources of each process from other processes
  - Enable processes to share and exchange information
  - Enable synchronization among processes.

What is a “Process” ?

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system resources

Process Elements

A process is comprised of:

- Program code (possibly shared)
- A set of data
- A number of attributes describing the state of the process

Process Elements

While the process is running it has a number of elements including

- **Identifier:** A unique identifier associated with this process, to distinguish it from all other processes.
- **State:** If the process is currently executing, it is in the running state.
- **Priority:** Priority level relative to other processes.
- **Program counter:** The address of the next instruction in the program to be executed.

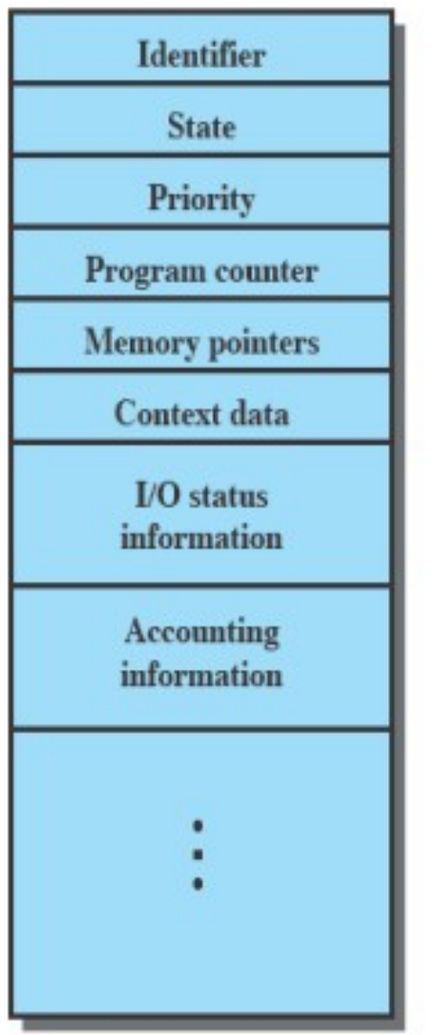
Process Elements

While the process is running it has a number of elements including

- **Memory pointers:** Includes pointers to the program code and data associated with this process, plus any memory blocks shared with other processes.
- **Context data:** These are data that are present in registers in the processor while the process is executing.
- **I/O status information:** Includes outstanding I/O requests, I/O devices (e.g., tape drives) assigned to this process, a list of files in use by the process, and so on.
- **Accounting information:** May include the amount of processor time and clock time used, time limits, account numbers, and so on.

Process Control Block (PCB)

- Contains the process elements
- Created and manage by the operating system
- Allows support for multiple processes
- Thus, we can say that a process consists of **program code** and **associated data** plus a **PCB**.



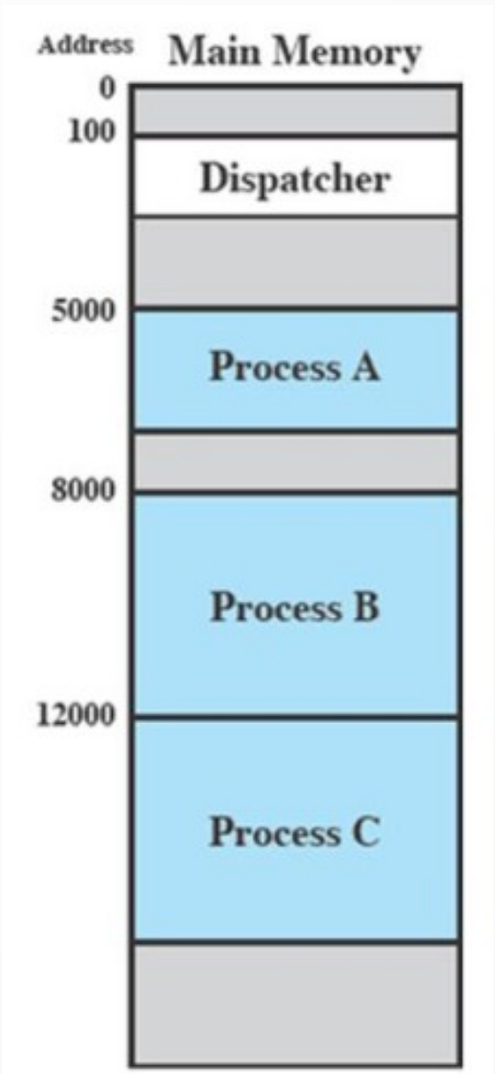
Simplified Process Control Block

Trace of a Process

- The behavior of an individual process is shown by listing the sequence of instructions that are executed
- This list is called a **Trace**
- **Dispatcher** is a small program which switches the processor from one process to another

Process Execution

- Consider three processes being executed
- All are in memory (plus the dispatcher)
- Lets ignore virtual memory for this.



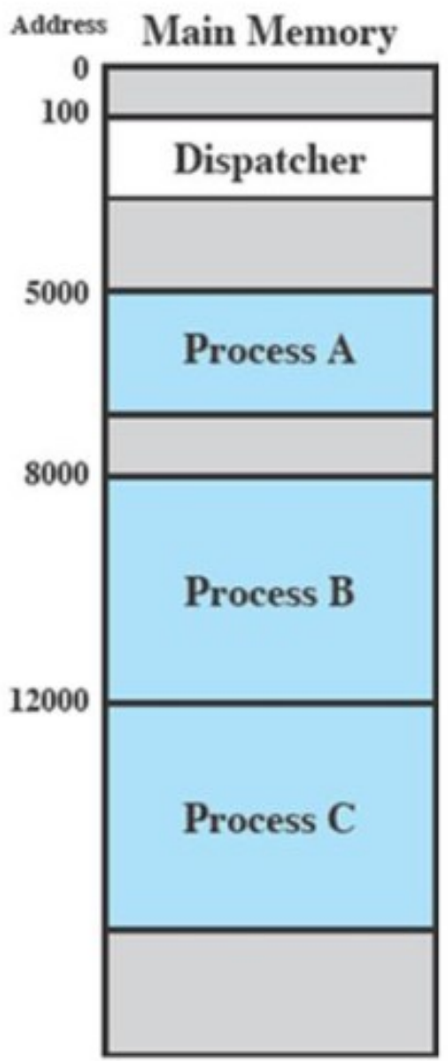
Trace from the Processes Point of View

Each Process runs to completion

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

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

Trace from the Processor Point of View



1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
----------	Timeout
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
----------	Timeout
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
----------	Timeout
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
----------	Timeout

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

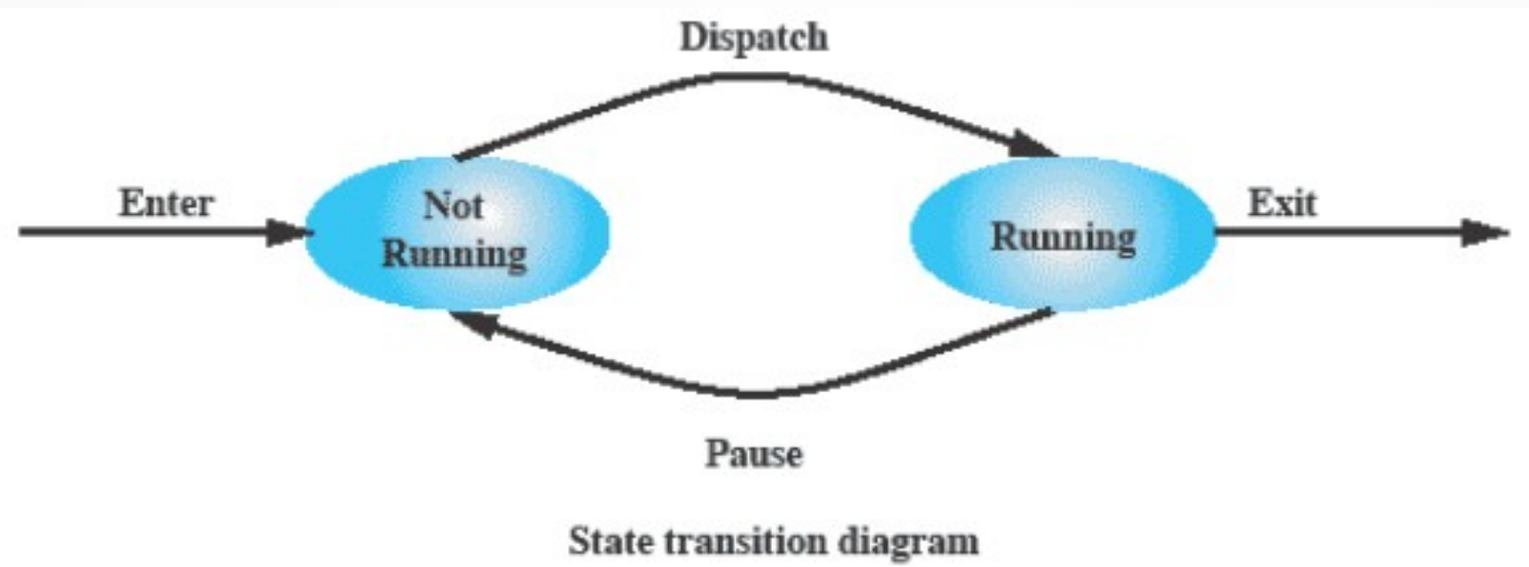
Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.

2 – State Process Model

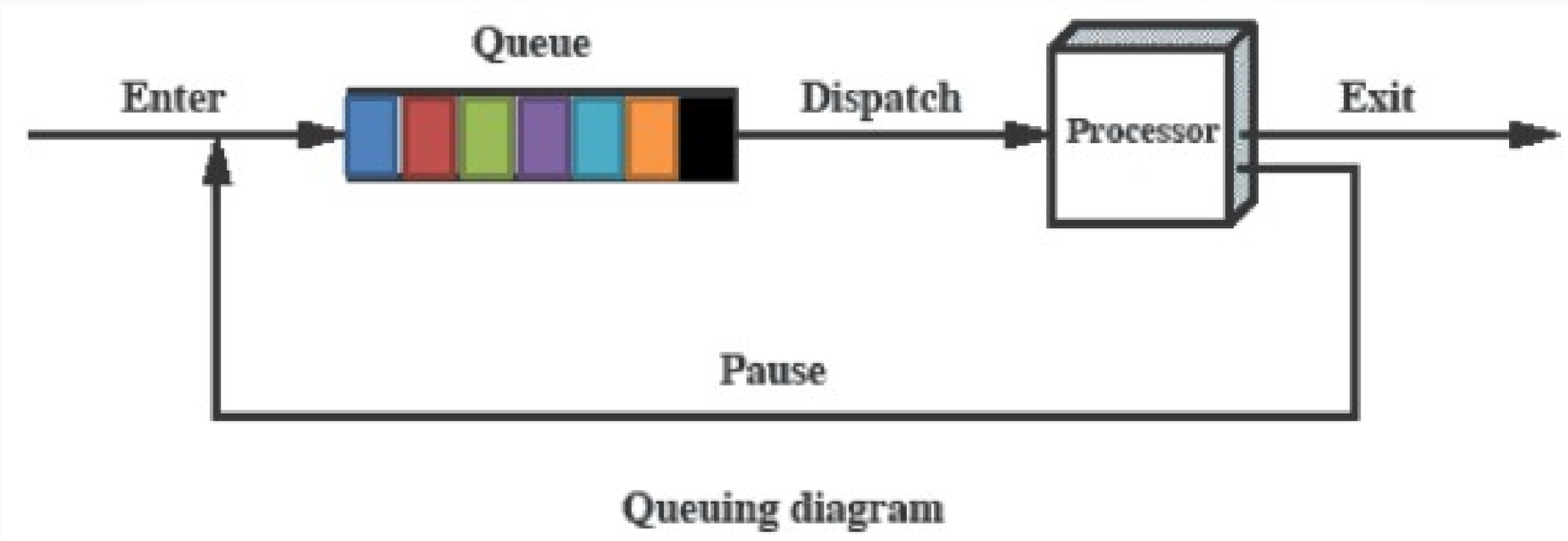
Process may be in one of two states

- Running
- Not-running



State transition diagram

Queuing Diagram



Queuing diagram

Processes moved by the dispatcher of the OS to the CPU then back to the queue until the task is competed

Process Creation

- The OS builds a data structure to manage the process
- Traditionally, the OS created all processes
- But it can be useful to let a running process create another
- This action is called process spawning
- Parent Process is the original, creating, process
- Child Process is the new process

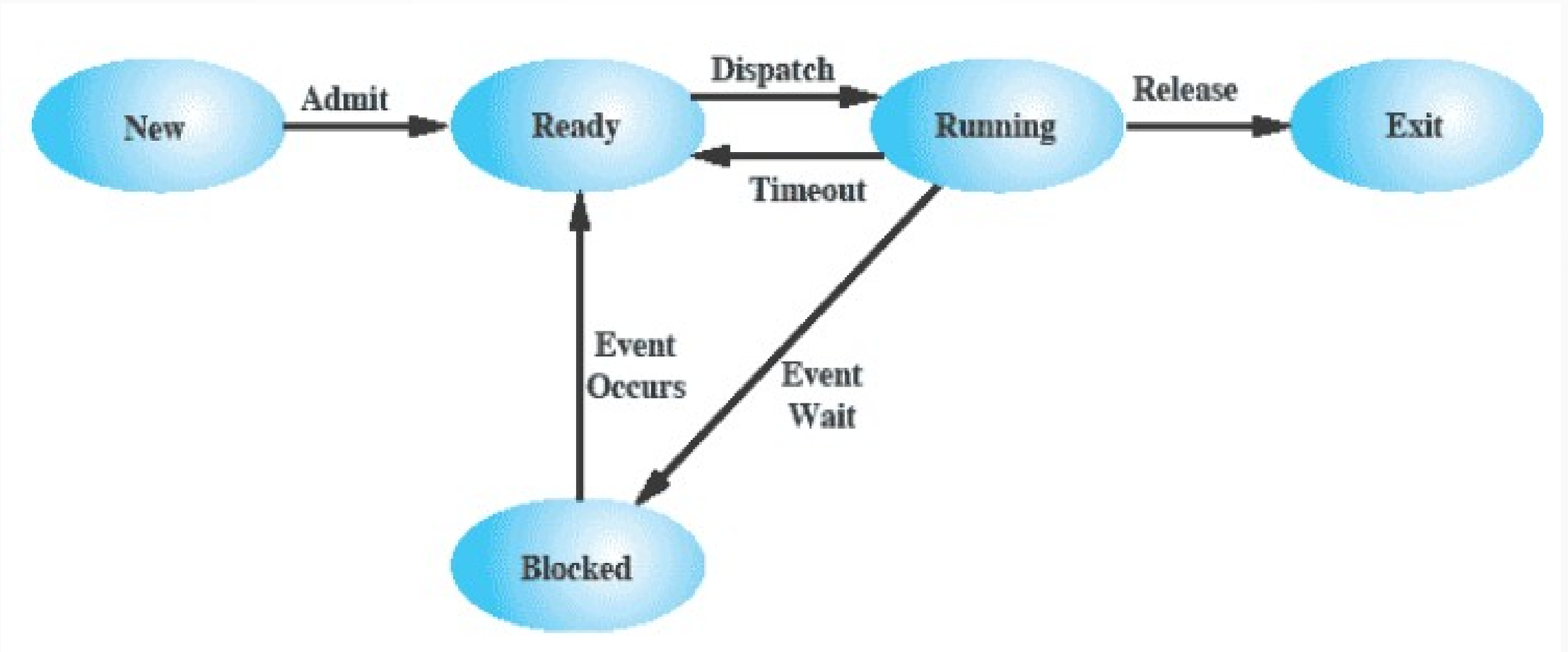
Process Termination

- There must be some way that a process can indicate completion.
- This indication may be:
  - A HALT instruction generating an interrupt alert to the OS.
  - A user action (e.g. log off, quitting an application)
  - A fault or error
  - Parent process terminating

Process Birth and Death

Creation	Termination
New batch job	Normal Completion
Interactive Login	Memory unavailable
Created by OS to provide a service	Protection error
Spawned by existing process	Operator or OS Intervention

5 – State Process Model



5 – State Process Model

1. **Running:** The process that is currently being executed.
2. **Ready:** A process that is prepared to execute when given the opportunity.
3. **Blocked/Waiting:** A process that cannot execute until some event occurs, such as the completion of an I/O operation.
4. **New:** A process that has just been created but has not yet been admitted to the pool of executable processes by the OS. Typically, a new process has not yet been loaded into main memory, although its process control block has been created.
5. **Exit:** A process that has been released from the pool of executable processes by the OS, either because it halted or because it aborted for some reason.

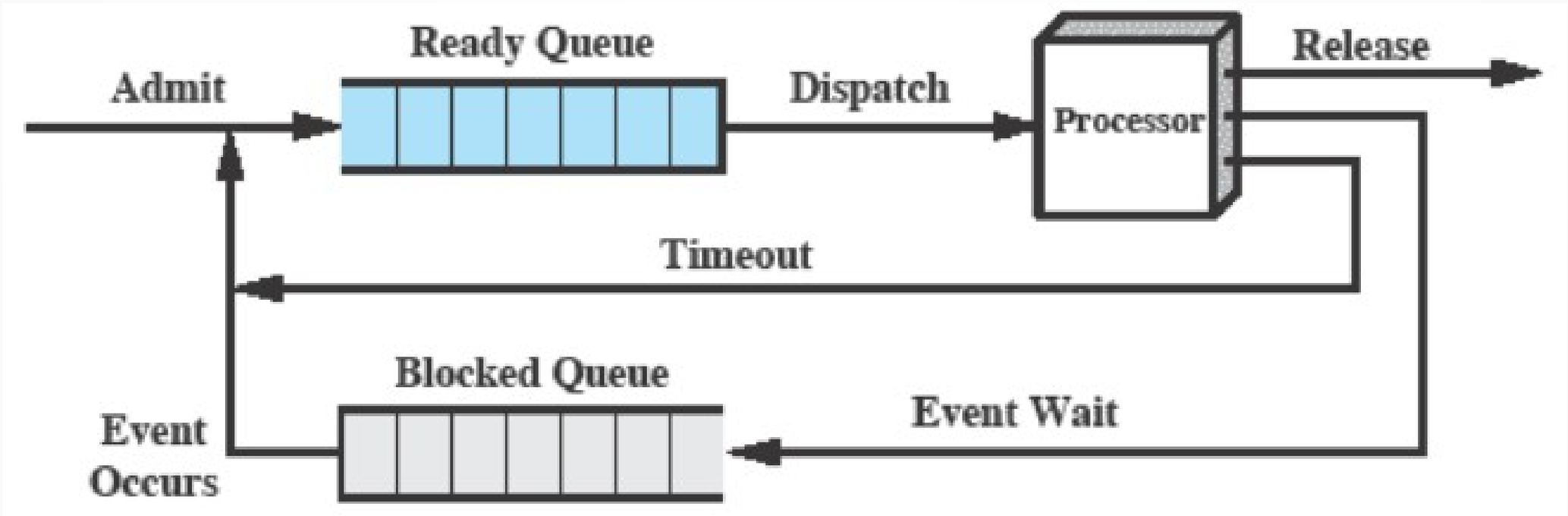
Process State Transitions (1)

1. Null → New : Process is created
2. New → Ready : O.S. is ready to handle another process (Memory, CPU)
3. Ready → Running : Select another process to run
4. Running → Exit : Process has terminated
5. Running → Ready : End of time slice or higher-priority process is ready

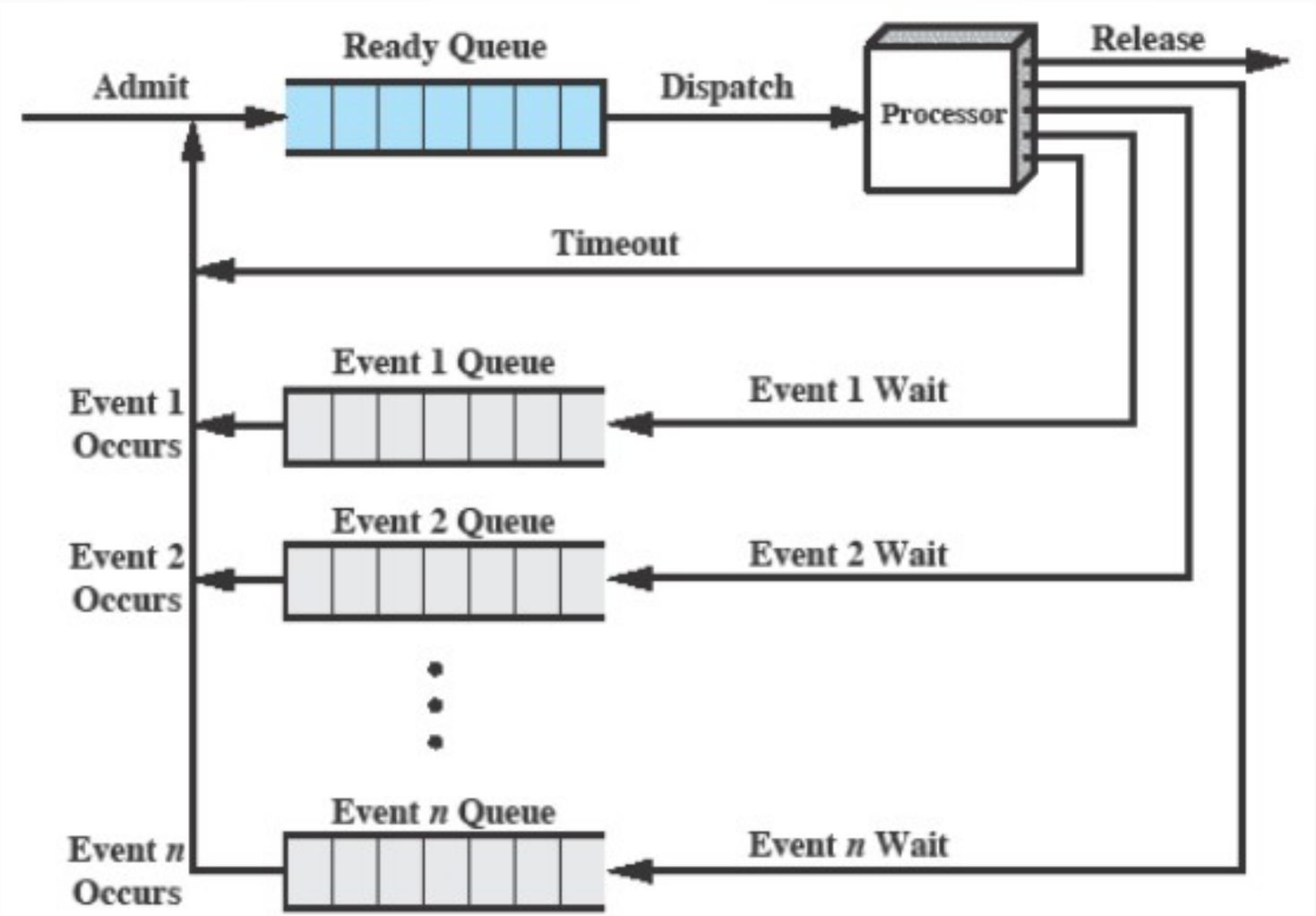
Process State Transitions (2)

6. Running → Blocked : Process is waiting for an event (I/O, Synchronization)
7. Blocked → Ready : The event a process is waiting for has occurred, can continue
8. Ready → Exit : Process terminated by O.S. or parent
9. Blocked → Exit : Same reasons

Single Blocked Queue



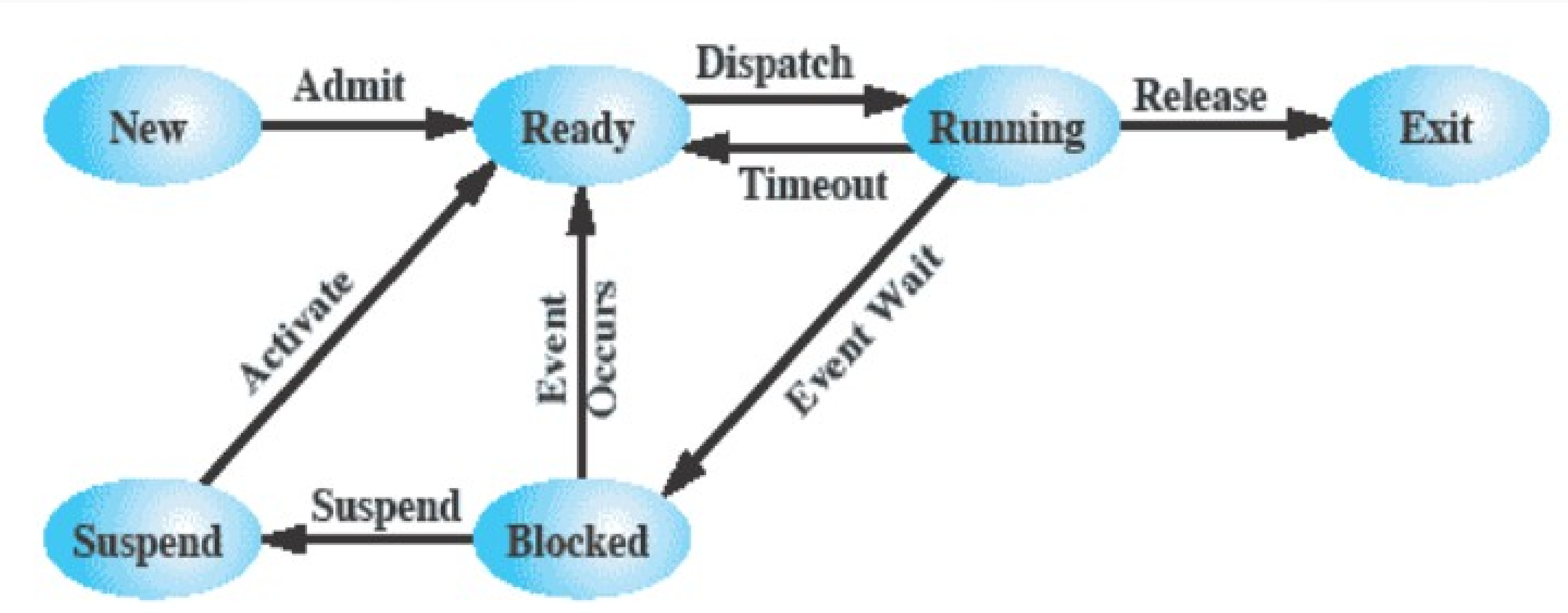
Multiple Blocked Queue



Suspended Processes

- Processor is faster than I/O so all processes could be waiting for I/O
  - A processor could be idle most of the time.
  - Swap these processes to disk to free up more memory and use processor on more processes
- Blocked state becomes suspend state when swapped to disk
- When all of the processes in main memory are in the Blocked state, the OS can suspend one process by putting it in the Suspend state and transferring it to disk.
- The space that is freed in main memory can then be used to bring in another process.

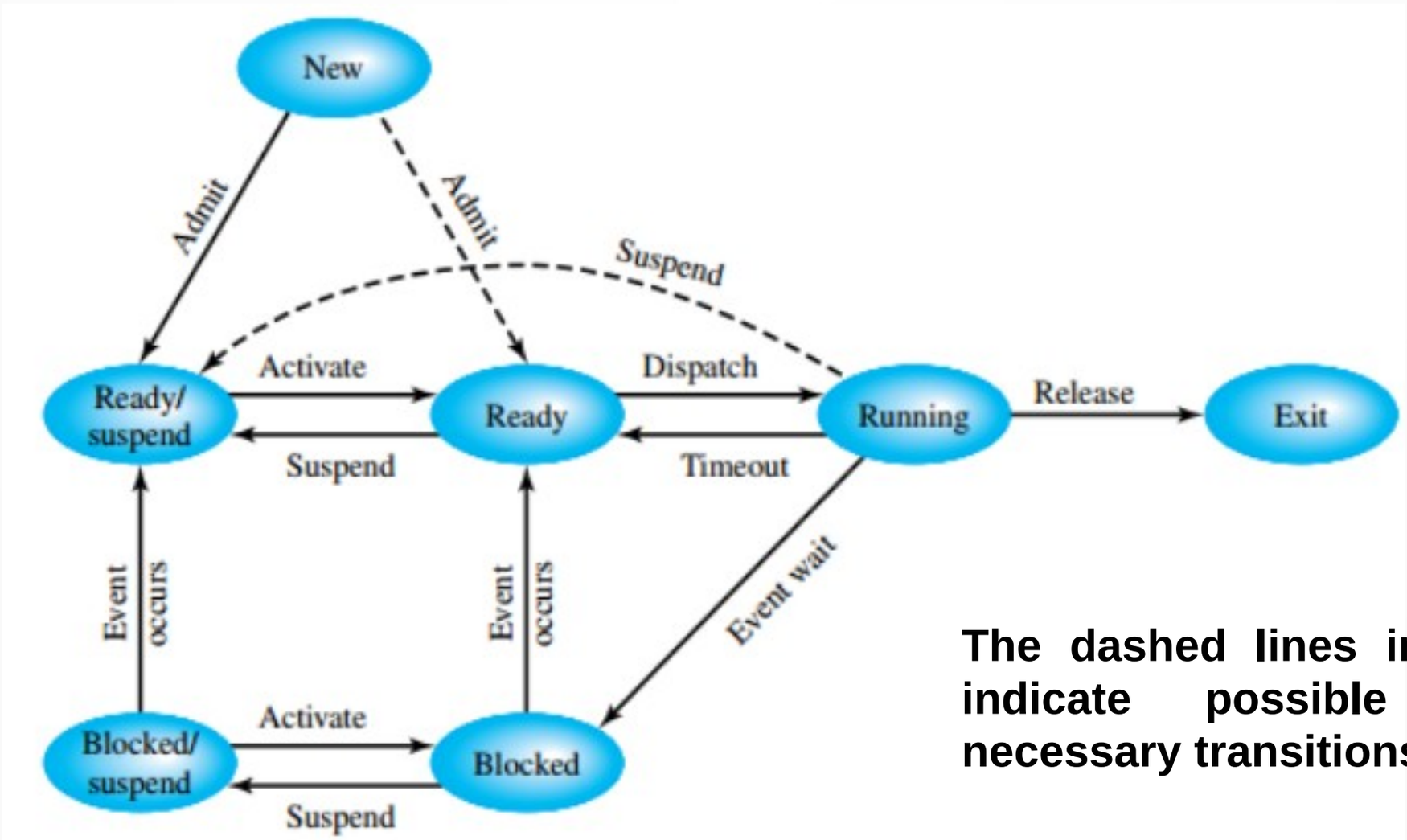
6 – State Process Model



Suspended Processes

- Two choices to bringing the process in main memory on place of suspended process.
  - Admit new process
  - Bring in a previously suspended process.
- No point in bringing the suspended process as it is a blocked process and not ready for execution
- Expand suspend state into two states
  - Blocked/Suspend : The process is in secondary memory and awaiting an event
  - Ready/Suspend : The process is in secondary memory but is available for execution as soon as it is loaded into main memory

7 – State Process Model



The dashed lines in the figure indicate possible but not necessary transitions

New Transitions (1)

1. Blocked → Blocked Suspend: Swap out a blocked process to free memory
2. Blocked Suspend → Ready Suspend: Event occurs (O.S. must track expected event)
3. Ready Suspend → Ready: Activate a process (higher priority, memory available)
4. Ready → Ready Suspend: Need to free memory for higher-priority processes

New Transitions

5. New → Ready or Ready Suspend: Can choose where to put new processes
6. Blocked Suspend → Blocked: Reload process expecting event soon
7. Running → Ready Suspend: Preempt for higher-priority process
8. Any → Exit: When parent terminates a child process or parent get terminated and hence all children exit

Characteristics of Suspended Process

1. The process is not immediately available for execution
2. The process may or may not be waiting on an event. If it is, this blocked condition is independent of the suspend condition, and occurrence of the blocking event does not enable the process to be executed immediately.
3. The process was placed in a suspended state by an agent: either itself, a parent process, or the OS, for the purpose of preventing its execution.
4. The process may not be removed from this state until the agent explicitly orders the removal.

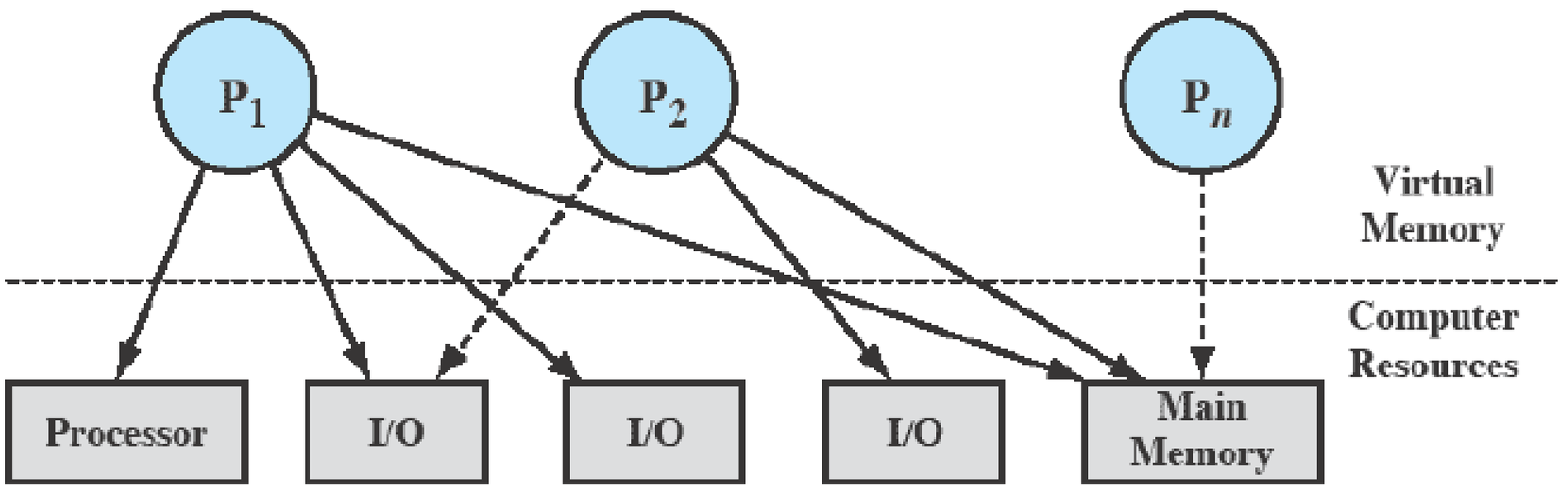
Reason for Process Suspension

Reason	Comment
Swapping	The OS needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS Reason	OS suspects process of causing a problem.
Interactive User Request	e.g. 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.
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 descendants.

Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.

Processes and Resources

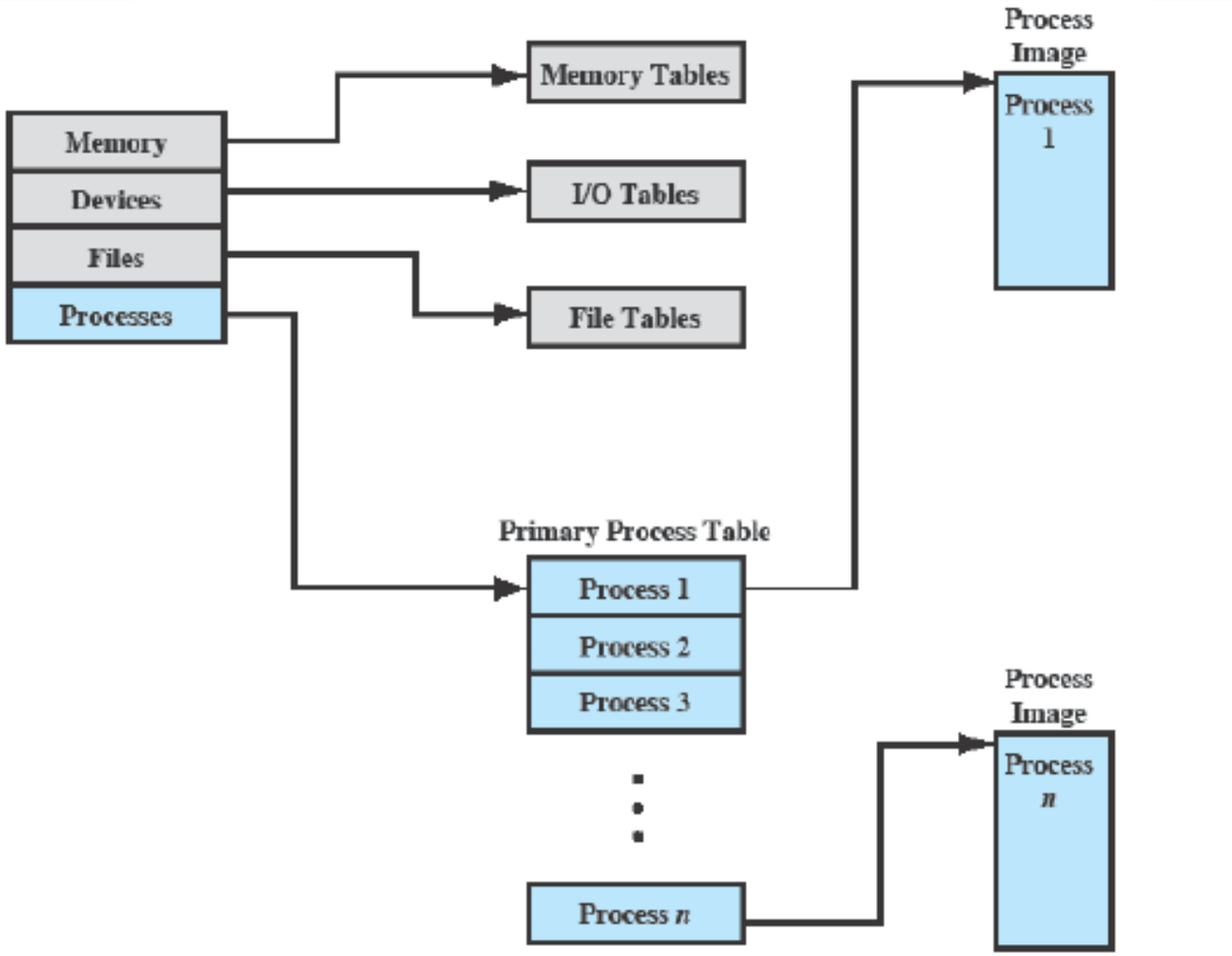


Processes and Resources (resource allocation at one snapshot in time)

OS Control Structure

- For the OS to manage processes and resources, it must have information about the current status of each process and resource.
- Tables are constructed for each entity the operating system manages

OS Control Tables



General Structure of Operating System Control Tables

Memory Tables

- Memory tables are used to keep track of both main and secondary memory.
- Must include this information:
  - 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

- Used by the OS to manage the I/O devices and channels of the computer.
- The OS needs to know
  - Whether the I/O device is available or assigned
  - The status of I/O operation
  - The location in main memory being used as the source or destination of the I/O transfer

File Tables

- These tables provide information about:
  - Existence of files
  - Location on secondary memory
  - Current Status
  - Other attributes
- Sometimes this information is maintained by a file management system

Process Tables

- To manage processes the OS needs to know details of the processes
  - Location in memory
  - Process attributes
- Memory, I/O, and files are managed on behalf of processes, so there must be some reference to these resources, directly or indirectly, in the process tables.

Physical manifestation of a Process

- A process will consist of at least sufficient memory to hold the programs and data of that process.
- In addition, the execution of a program typically involves a stack that is used to keep track of procedure calls and parameter passing between procedures.
- Finally, each process has associated with it a number of attributes that are used by the OS for process control. Typically, the collection of attributes is referred to as a process control block.
- This collection of program, data, stack and attributes (PCB) is referred as the Process Image.

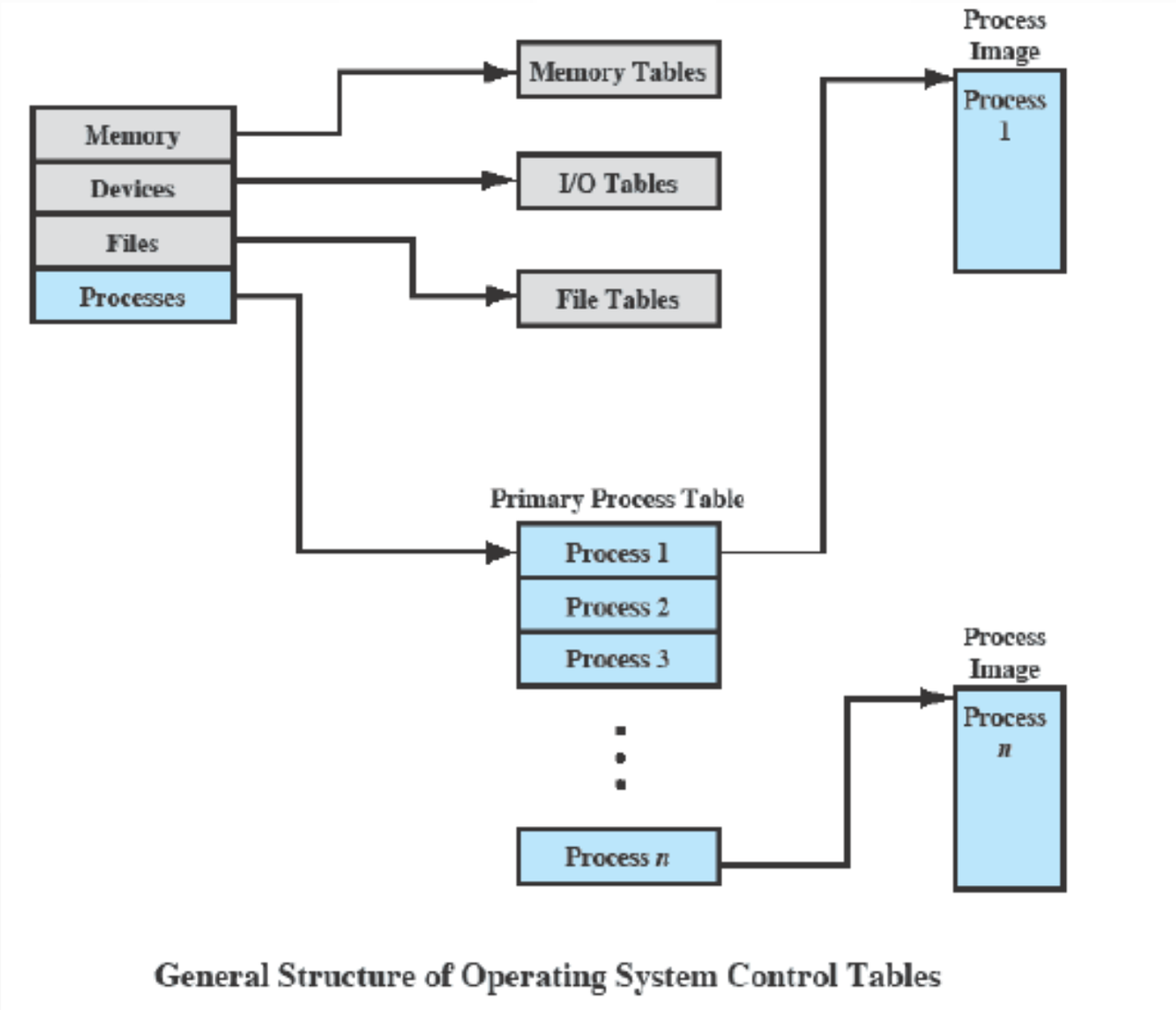
Elements of a Process Image

- **User Data:** The modifiable part of the user space. It mainly contains program data.
- **User Program:** The program to be executed.
- **User Stack:** Each process has one or more last-in-first-out (LIFO) 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 OS to control the process.

Process Location

- Depends on memory management scheme.
- Process image is maintained as a contiguous or continuous, block of memory.
- Normally this block is maintained in secondary memory (usually Disk).
- But to manage the process, a small portion must be loaded in main memory.
- To execute the process, it need to be loaded in main memory or virtual memory.
- In case where process image is loaded in virtual memory, it is required for OS to know the location of each process on disk and location of a process in main memory.

Process Location (Cont..)



General Structure of Operating System Control Tables

Process Attributes - PCB

The process control block information can be grouped into three general categories:

- Process identification
- Processor state information
- Process control information

Process Identification

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

Processor State Information

User Visible Registers

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

Processor State Information

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

Processor State Information

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 Information

This is the additional information needed by the OS to control and coordinate the various active processes.

- Scheduling and State Information
- Data Structuring
- IPC
- Process Privileges
- Memory Management
- Resource Ownership and Utilization

Scheduling & State Information (1)

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

- Process state: Defines the readiness of the process to be scheduled for execution (e.g., running, ready, waiting, halted).
- Priority: One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest-allowable).

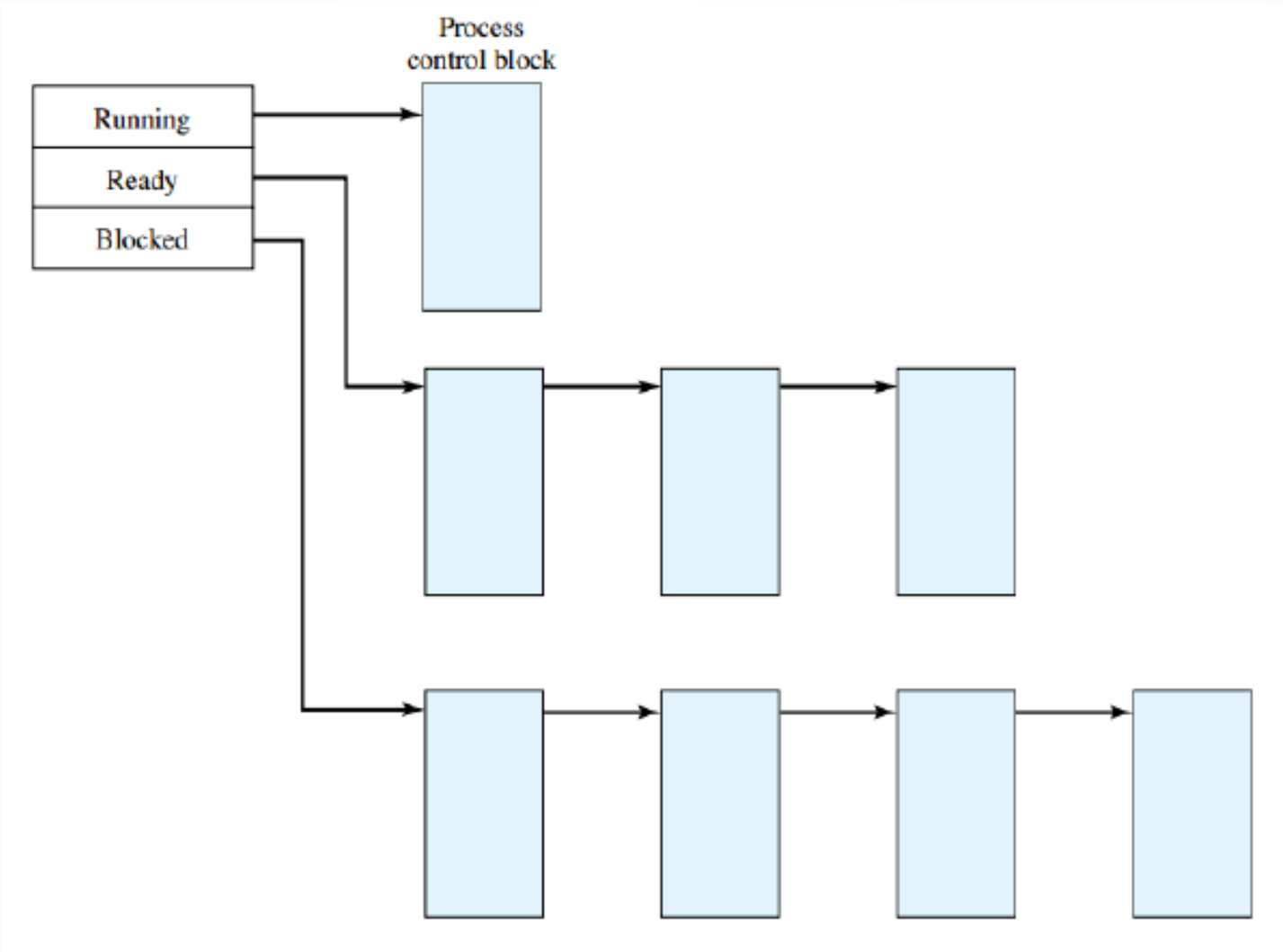
Scheduling & State Information

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

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 List Structure



Interprocess Comunication

- 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.

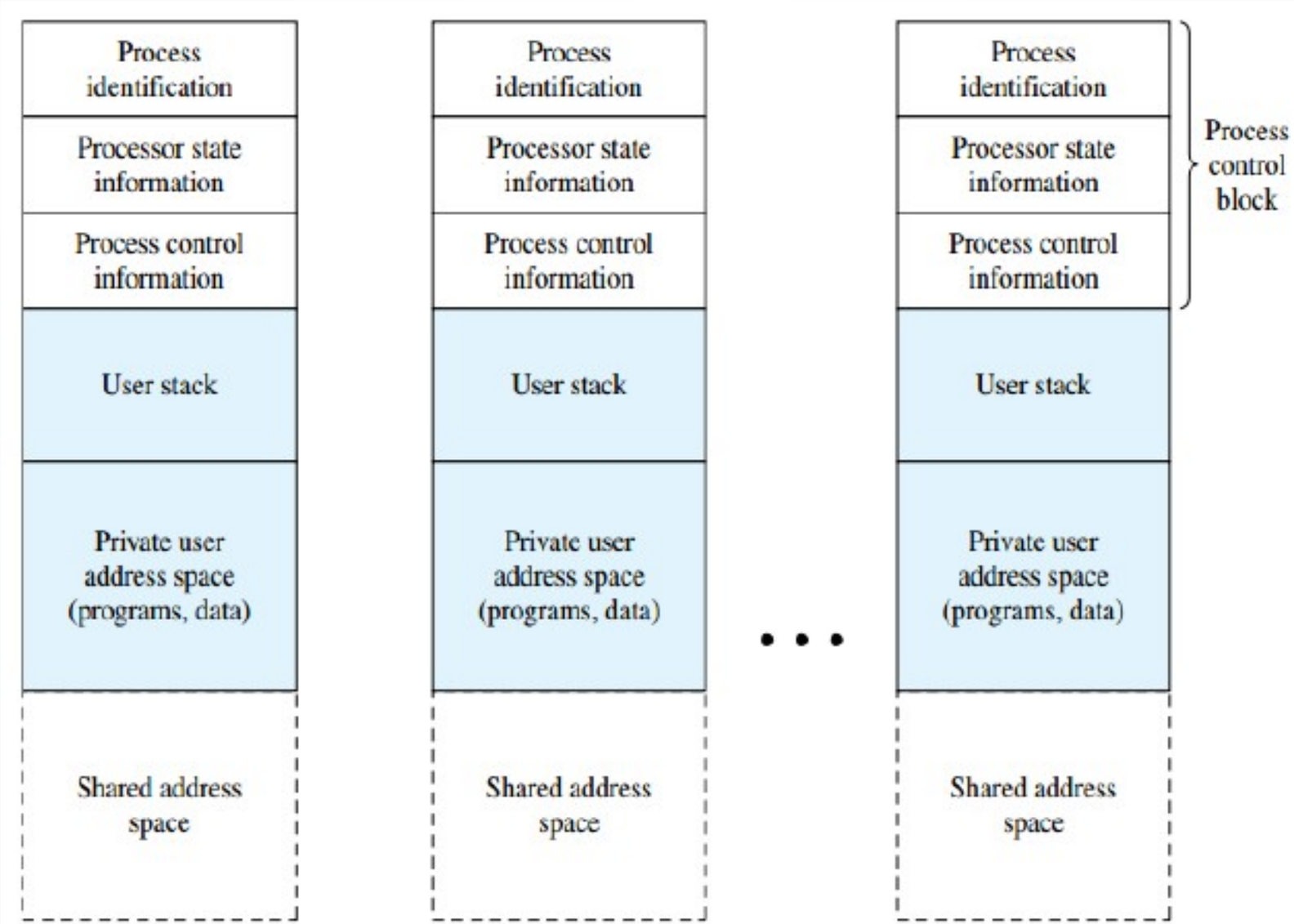
Memory Management

This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.

Resource Ownership & 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.

User Processes in Virtual Memory



Role of the PCB

- The most important data structure in an OS
  - It defines the state of the OS
- Process Control Block requires protection
  - A faulty routine could cause damage to the block destroying the OS’s ability to manage the process
  - Any design change to the block could affect many modules of the OS

Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.

Modes of Execution

Most processors support at least two modes of execution

- **User mode**
  - Less-privileged mode
  - User programs typically execute in this mode
- **System mode**
  - More-privileged mode
  - Kernel of the operating system

Functions of OS Kernel

Process Management

- Process creation and termination
- Process scheduling and dispatching
- Process switching
- Process synchronization and support for interprocess communication
- Management of process control blocks

Functions of OS Kernel

Memory Management

- Allocation of address space to processes
- Swapping
- Page and segment management

I/O Management

- Buffer management
- Allocation of I/O channels and devices to processes

Functions of OS Kernel

Support Functions

- Interrupt handling
- Accounting
- Monitoring

Process Creation

Once the OS decides to create a new process it:

- Assigns a unique process identifier
- Allocates space for the process
- Initializes process control block
- Sets up appropriate linkages
- Creates or expand other data structures

Switching Processes

Several design issues are raised regarding process switching

- What events trigger a process switch?
- We must distinguish between mode switching and process switching.
- What must the OS do to the various data structures under its control to achieve a process switch?

When to switch processes

A process switch may occur any time that the OS has gained control from the currently running process. Possible events giving OS control are:

Mechanism	Cause	Use
Interrupt	External to the execution of the current instruction	Reaction to an asynchronous external event
Trap	Associated with the execution of the current instruction	Handling of an error or an exception condition
Supervisor call	Explicit request	Call to an operating system function

Mechanisms for Interrupting the Execution of a Process

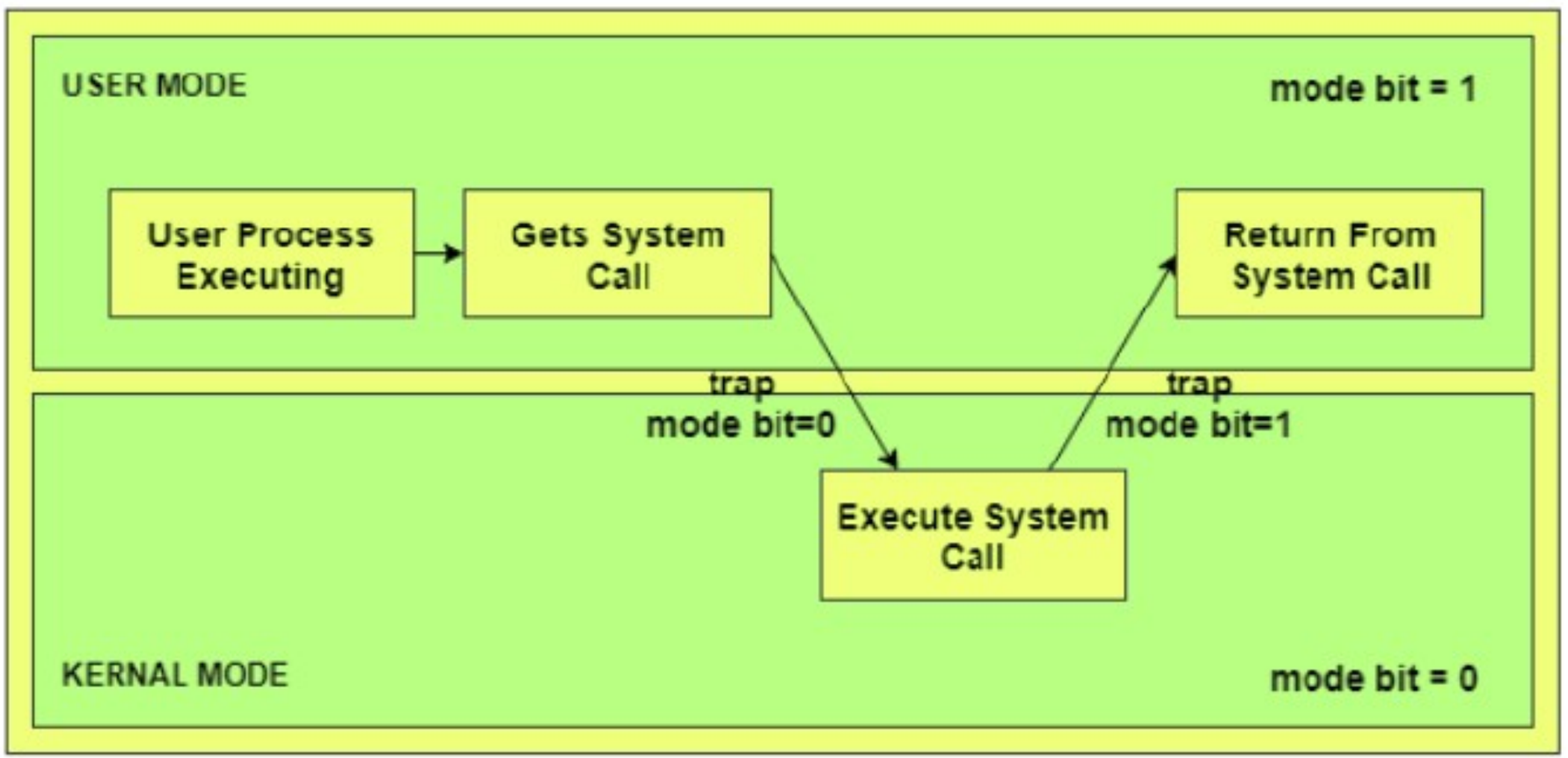
Reasons for Switching

- **Interrupt (external to running process)**
  - Clock interrupt (time slice expired)
  - I/O interrupt (an event occurred)
- **Trap (internal to running process)**
  - Memory Fault
  - Invalid operation (i.e. divide by zero)
- **System / Supervisor Call**

Mode Switching

- When a running process changes its mode from one to other it is called mode switching.
- While switching the mode, processor state information of PCB is saved (user– kernel) and restored back (kernel - user)

Mode Switching



Source: tutorialspoint

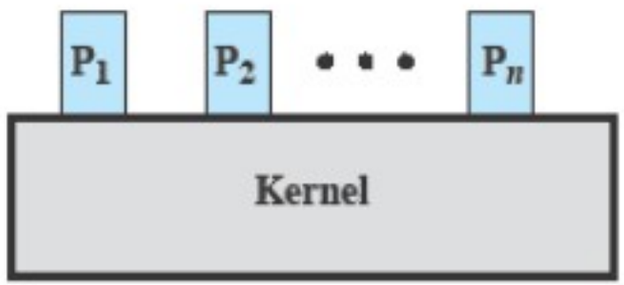
Steps in a process switch

1. Save context of processor including program counter and other registers
2. Update the process control block of the process that is currently in the Running state
3. Move process control block to appropriate queue – ready; blocked; ready/suspend
4. Select another process for execution
5. Update the process control block of the process selected
6. Update memory-management data structures
7. Restore context of the selected process

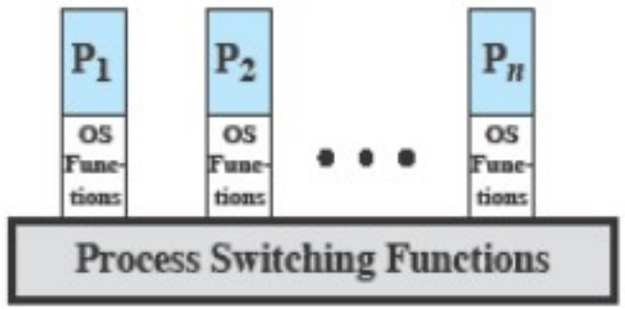
Is OS a process?

- If the OS is just a collection of programs and if it is executed by the processor just like any other program, is the OS a process?
- If so, how is it controlled?
- Who (what) controls it?

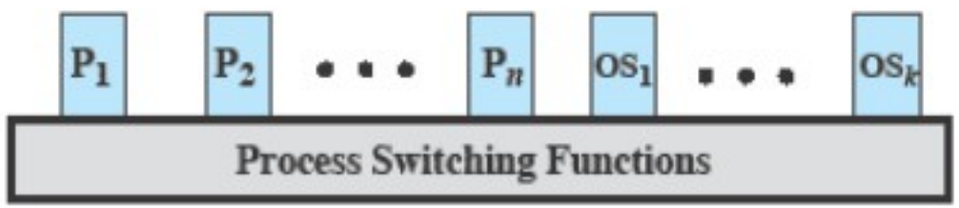
Execution of the OS



(a) Separate kernel



(b) OS functions execute within user processes

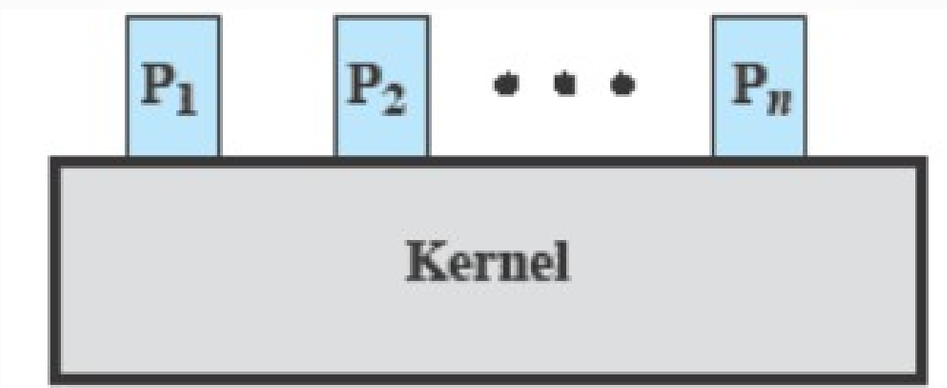


(c) OS functions execute as separate processes

Relationship Between Operating System and User Processes

Non-process Kernel

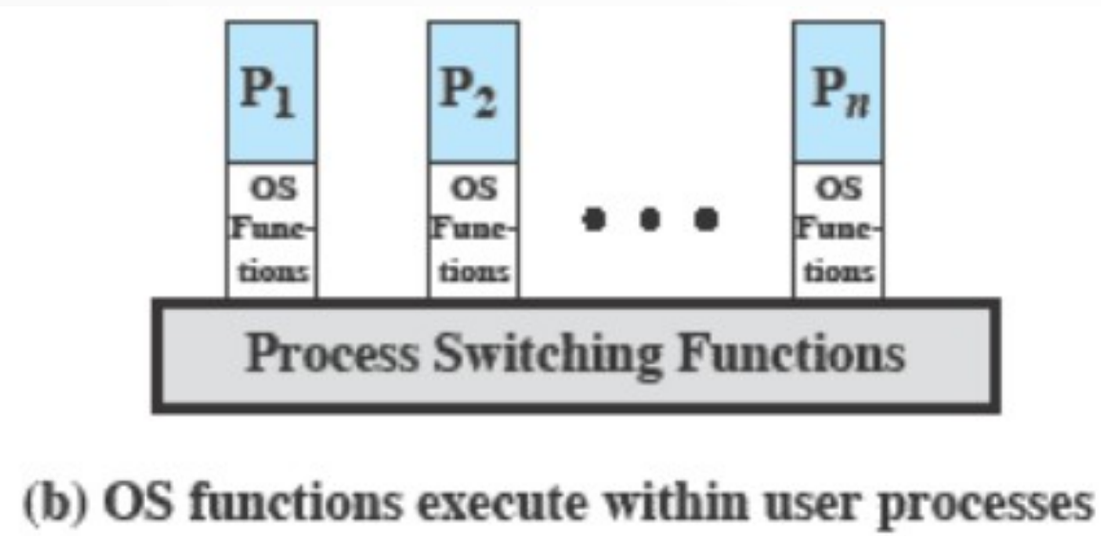
- Kernel runs outside the context of any running process
- Kernel has its own memory and stack
- The concept of process is considered to apply only to user programs
- Operating system code is executed as a separate entity that operates in privileged mode



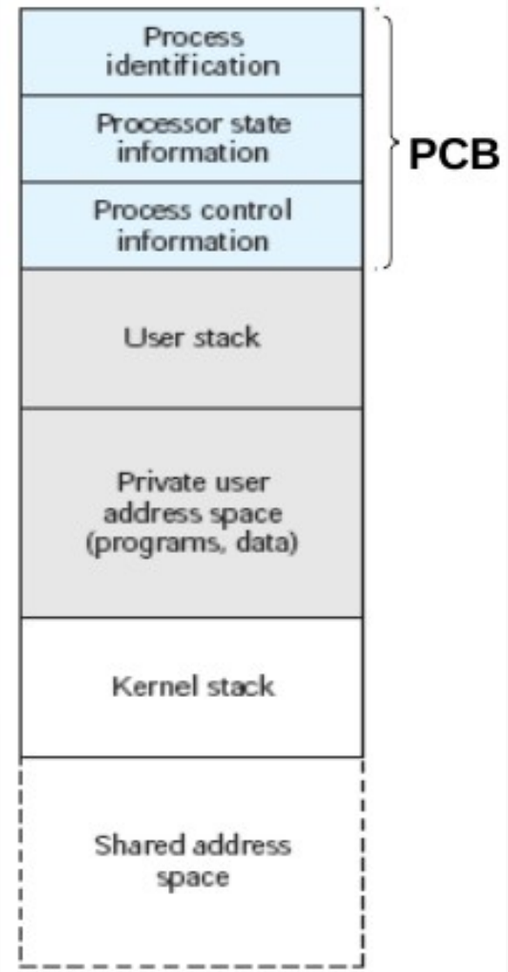
(a) Separate kernel

Execution within User Processes

- Operating system software within context of a user process
- No need for Process Switch to run OS routine



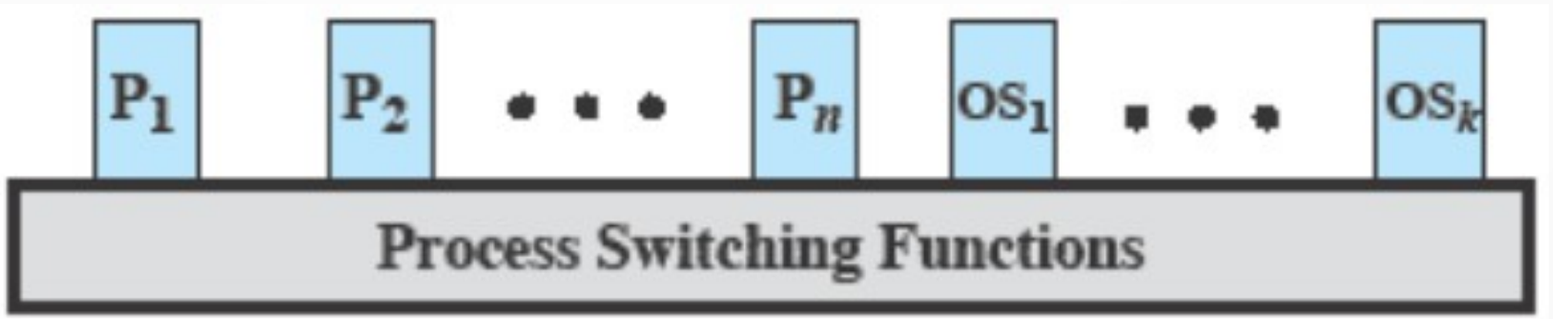
(b) OS functions execute within user processes



OS executes within user space

Process-based OS

- Implement the OS as a collection of system process
- Encourages a clean design
- Useful in a multiprocessor system



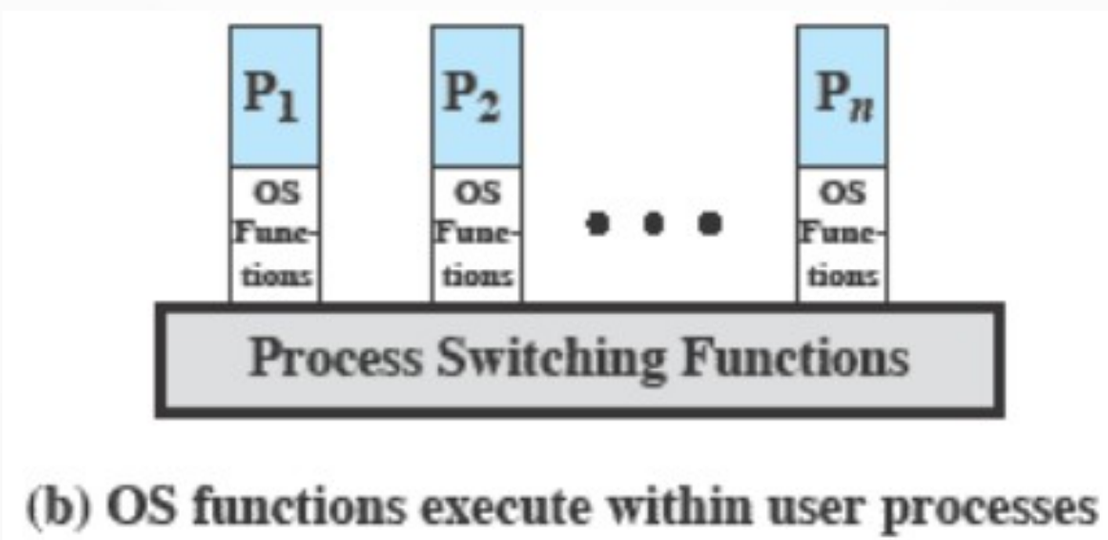
(c) OS functions execute as separate processes

Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.

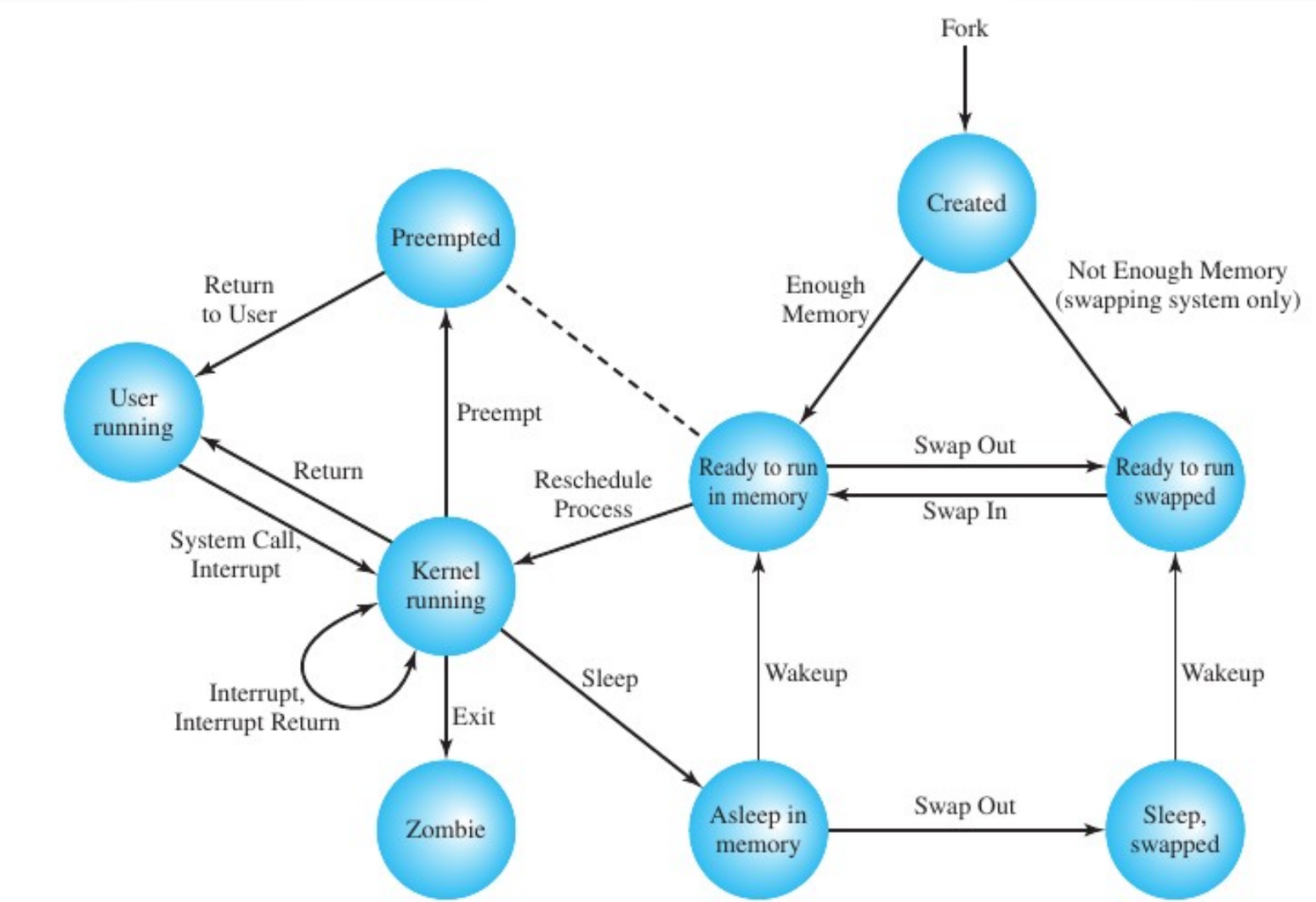
UNIX SVR4

- Uses the model (as shown in figure below) where most of the OS executes in the user process
- **System Processes** - Kernel mode only
- **User Processes**
  - User mode to execute user programs and utilities
  - Kernel mode to execute instructions that belong to the kernel.



(b) OS functions execute within user processes

UNIX Process State Transition



UNIX Process State Transition Diagram

UNIX Process States

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

Unix Process Image

- A process in UNIX is a set of data structures that provide the OS with all of the information necessary to manage and dispatch processes.
- The elements of the process image can be organized into three parts:
  - User-Level Context
  - Register Context
  - System-Level Context

Unix Process Image

User-Level Context

- Process text: Executable machine instructions of the program.
- Process Data: Data accessible by the program of this process.
- User Stack: Contains the arguments, local variables, and pointers for functions executing in user mode.
- Shared Memory: Memory shared with other processes, used for interprocess communication.

Unix Process Image

Register Context

- Program Counter: Address of next instruction to be executed; may be in kernel or user memory space of this process.
- Processor Status Register: Contains the hardware status at the time of preemption; contents and format are hardware dependent.
- Stack Pointer: Points to the top of the kernel or user stack, depending on the mode of operation at the time or preemption.
- General-purpose registers: Hardware dependent.

Unix Process Image

System-Level Context

- Process table entry: Defines state of a process; this information is always accessible to the operating system.
- U area: Process control information that needs to be accessed only in the context of the process
- Per process region table: Defines the mapping from virtual to physical addresses; also contains a permission field that indicates the type of access allowed the process: read-only, read-write, or read-execute
- Kernel Stack: Contains the stack frame of kernel procedures as the process executes in kernel mode

State of a Process

- **Process Table Entry**
  - Contains general fields of processes that must be always be accessible to the kernel
- **U area**
  - further characteristics of the process only need to be accessible to the running process itself

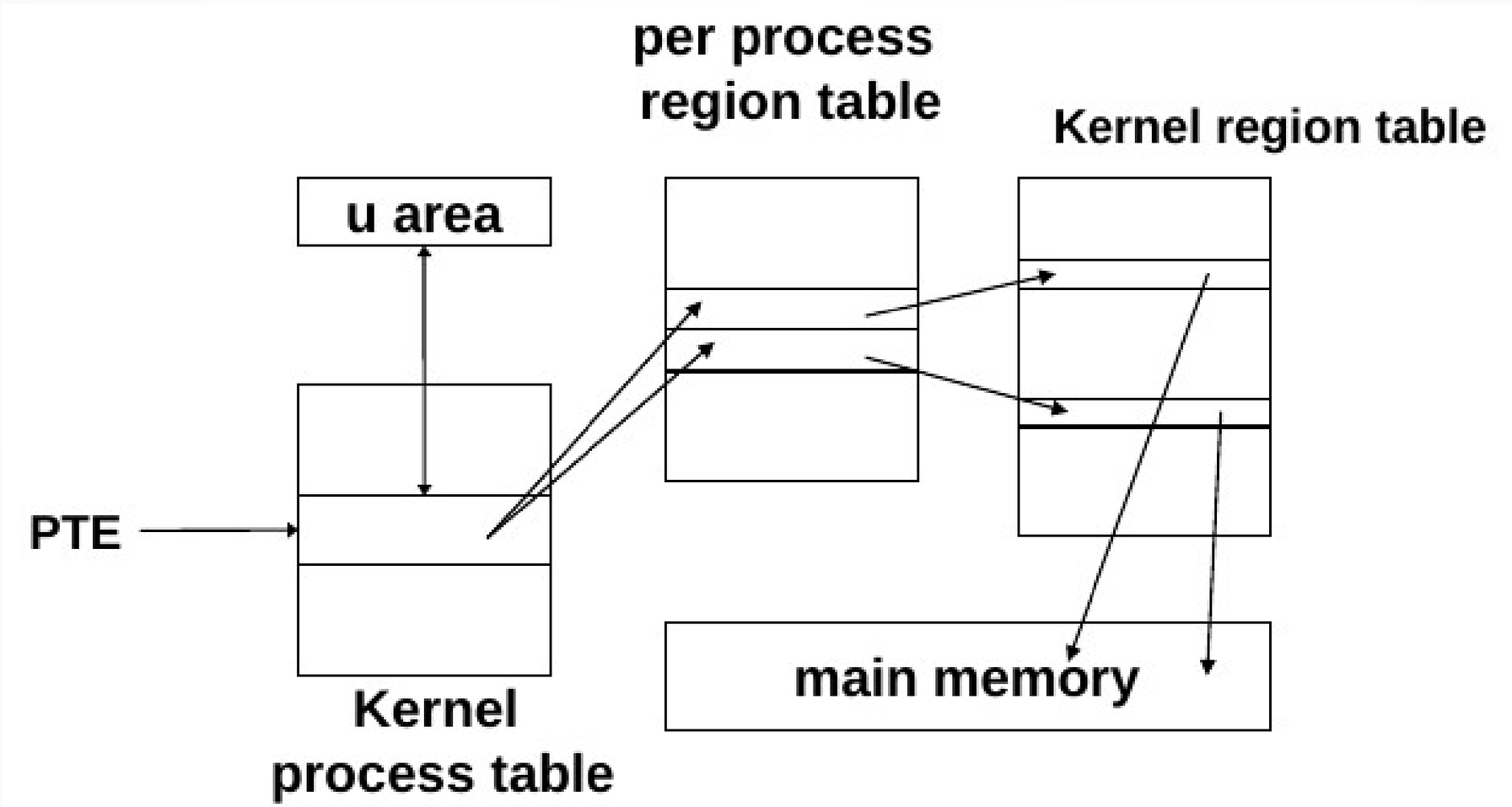
Process Table Entry

Process status	Current state of process.
Pointers	To U area and process memory area (text, data, stack).
Process size	Enables the operating system to know how much space to allocate the process.
User identifiers	The **real user ID** identifies the user who is responsible for the running process. The **effective user ID** may be used by a process to gain temporary privileges associated with a particular program; while that program is being executed as part of the process, the process operates with the effective user ID.
Process identifiers	ID of this process; ID of parent process. These are set up when the process enters the Created state during the fork system call.
Event descriptor	Valid when a process is in a sleeping state; when the event occurs, the process is transferred to a ready-to-run state.
Priority	Used for process scheduling.
Signal	Enumerates signals sent to a process but not yet handled.
Timers	Include process execution time, kernel resource utilization, and user-set timer used to send alarm signal to a process.
P_link	Pointer to the next link in the ready queue (valid if process is ready to execute).
Memory status	Indicates whether process image is in main memory or swapped out. If it is in memory, this field also indicates whether it may be swapped out or is temporarily locked into main memory.

U Area

Process table pointer	Indicates entry that corresponds to the U area.
User identifiers	Real and effective user IDs. Used to determine user privileges.
Timers	Record time that the process (and its descendants) spent executing in user mode and in kernel mode.
Signal-handler array	For each type of signal defined in the system, indicates how the process will react to receipt of that signal (exit, ignore, execute specified user function).
Control terminal	Indicates login terminal for this process, if one exists.
Error field	Records errors encountered during a system call.
Return value	Contains the result of system calls.
I/O parameters	Describe the amount of data to transfer, the address of the source (or target) data array in user space, and file offsets for I/O.
File parameters	Current directory and current root describe the file system environment of the process.
User file descrip-tor table	Records the files the process has open.
Limit fields	Restrict the size of the process and the size of a file it can write.
Permission modes fields	Mask mode settings on files the process creates.

Data Structures for Process



Process Creation

- Process creation is by means of the kernel system call, **fork( )**.
- This causes the OS, in Kernel Mode, to:
  1. Allocate a slot in the process table for the new process.
  2. Assign a unique process ID to the child process.
  3. Copy of process image of the parent, with the exception of any shared memory.

Process Creation (Cont..)

4. Increment the counters for any files owned by the parent, to reflect that an additional process now also owns those files.
5. Assign the child process to the Ready to Run state.
6. Returns the ID number of the child to the parent process, and a 0 value to the child process.

After Creation

After creating the process the Kernel can do one of the following, as part of the dispatcher routine:

- Stay in the parent process.
- Transfer control to the child process
- Transfer control to another process.

Introduction to Security

- OS assigns privileges to processes (e.g., access to files, memory, devices).
- Root access = full control (also known as admin/supervisor access).
- Security goal: Prevent unauthorized access and privilege escalation.

System Access Threats

Two main categories:

- Intruders (referred as hacker/cracker)
- Malicious Software (Malware)

Types of Intruders

Type	Description
Masquerader	External attacker using stolen credentials
Misfeasor	Insider misusing legitimate access
Clandestine User	Gains supervisory control to hide malicious activity

Malicious Software (Malware)

- Exploits OS or software vulnerabilities
- Affects applications, utilities, kernel-level components

Types of Malware

Criteria	Examples
Needs Host (Parasitic)	Virus, Logic Bomb, Backdoor
Independent	Worm, Bot program
Non-replicating	Logic Bomb, Backdoor
Replicating	Virus, Worm

Countermeasures – Intrusion Detection Systems (IDS)

Purpose: Monitor and warn of unauthorized access

Types:

- Host-based IDS: Monitors a single host
- Network-based IDS: Monitors network traffic

Components of IDS

1. Sensors – Collect data (e.g., packets, logs)
2. Analyzers – Detect signs of intrusion
3. User Interface – For alerts and configuration

Countermeasures – Authentication

Verifies identity of a user

Steps:

1. Identification (e.g., username)
2. Verification (e.g., password)

Authentication Methods

Method	Examples
Something you **know**	Password, PIN
Something you **have**	Smart card, token
Something you **are (**Static**)**	Fingerprint, Face ID
Something you **do (**Dynamic**)**	Voice, Typing pattern

Countermeasures – Access Control

- A policy governing access to resources
- Controls who can access what, and how
- Depends on:
  - Authentication
  - Authorization database
  - Auditing access logs

Countermeasures – Firewalls

- Protect internal systems from external threats
- Only authorized traffic allowed
- Must be:
  - Central point for traffic
  - Policy-driven
  - Secure and hardened

Types of Firewalls

- **Network Firewall:** Protects entire network
- **Personal Firewall:** Protects individual systems

References

1. Operating Systems: Internals and Design Principles by William Stallings.

2. A Book - “The Design of The UNIX Operating System” by Maurice J. Bach

Questions ?