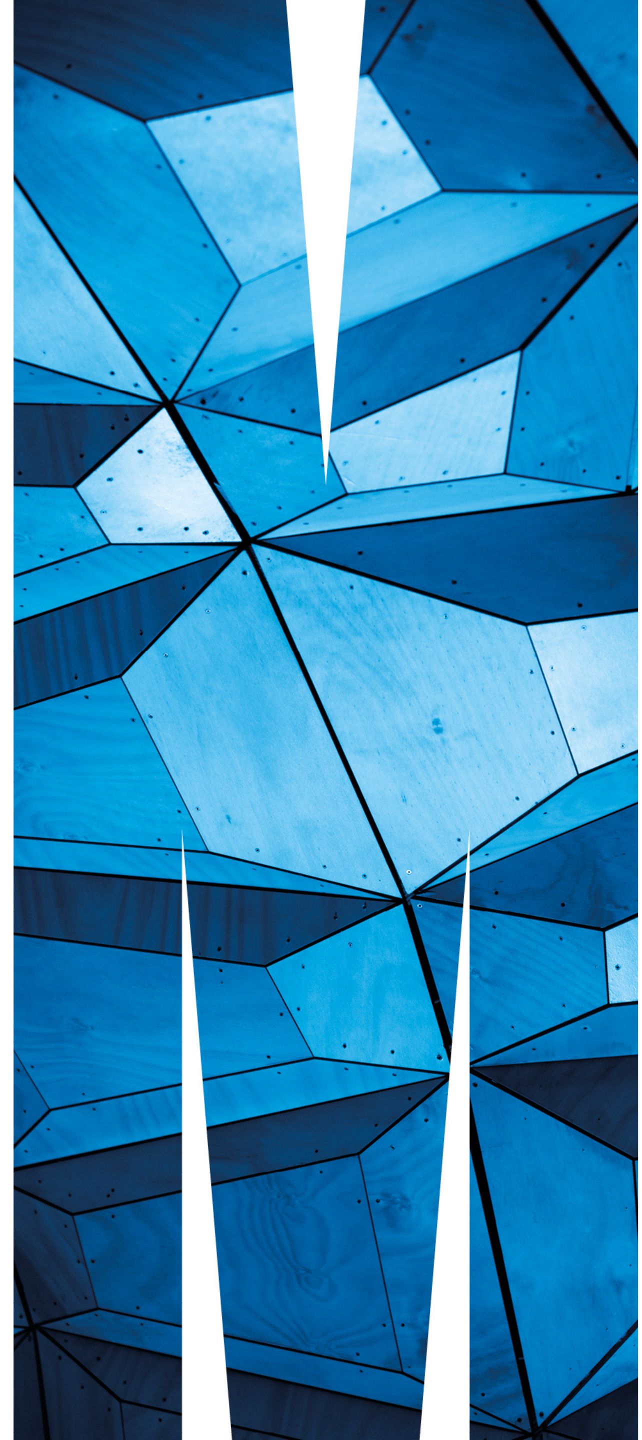
MONASH
INFORMATION
TECHNOLOGY

FIT2100 Semester 2 2017

Lecture 3: Processes

(Reading: Stallings, Chapter 3)

Jojo Wong



Lecture 3: Learning Outcomes

- Upon the completion of this lecture, you should be able to:
  - Define the concept of process
  - Understand the relationship between processes and process control blocks (PCB)
  - Discuss the concepts of process state and state transition
  - Describe the process states in Unix systems

What do we understand about OS?

OS: Management of Application Execution

- System resources are made available to multiple applications

Multiprogramming and Multitasking

- The processor is switched among multiple applications such that all will appear to be progressing

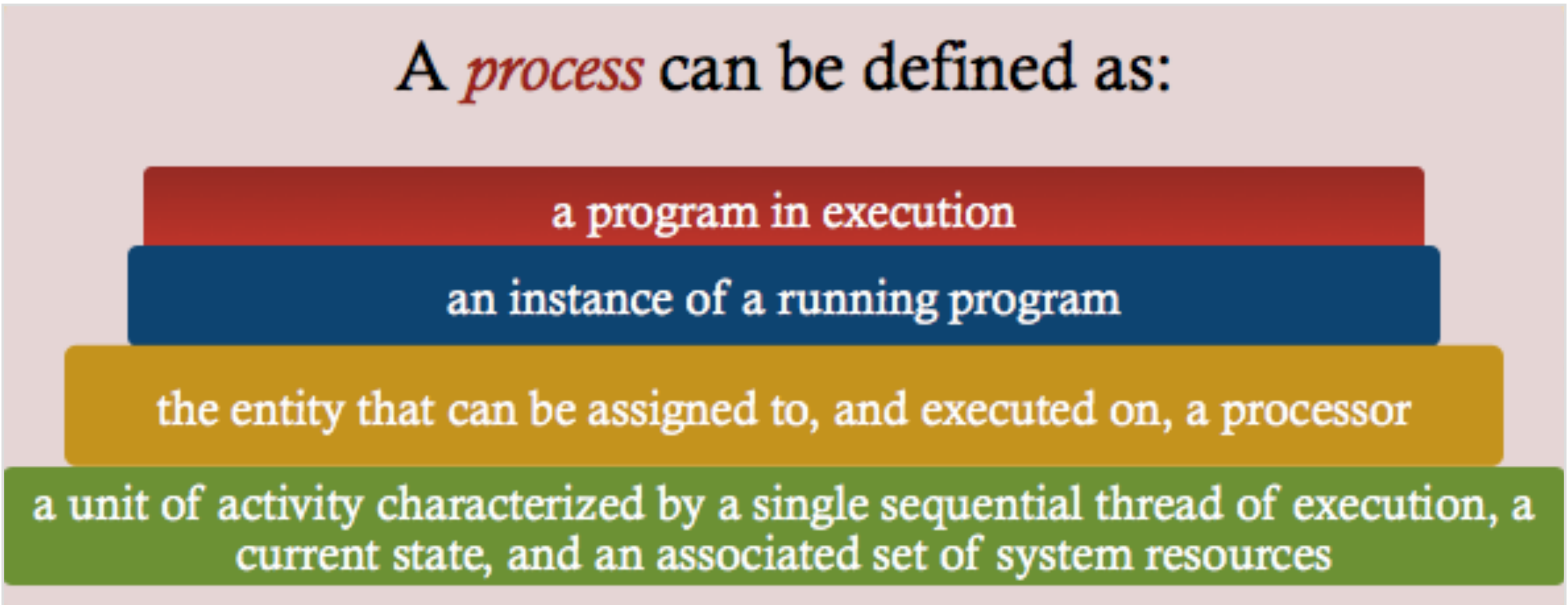
Time slicing of the processor time

- The processor and I/O devices can be used efficiently

What is a process?

The Concept of Process

- Fundamental to the *structure* of operating system



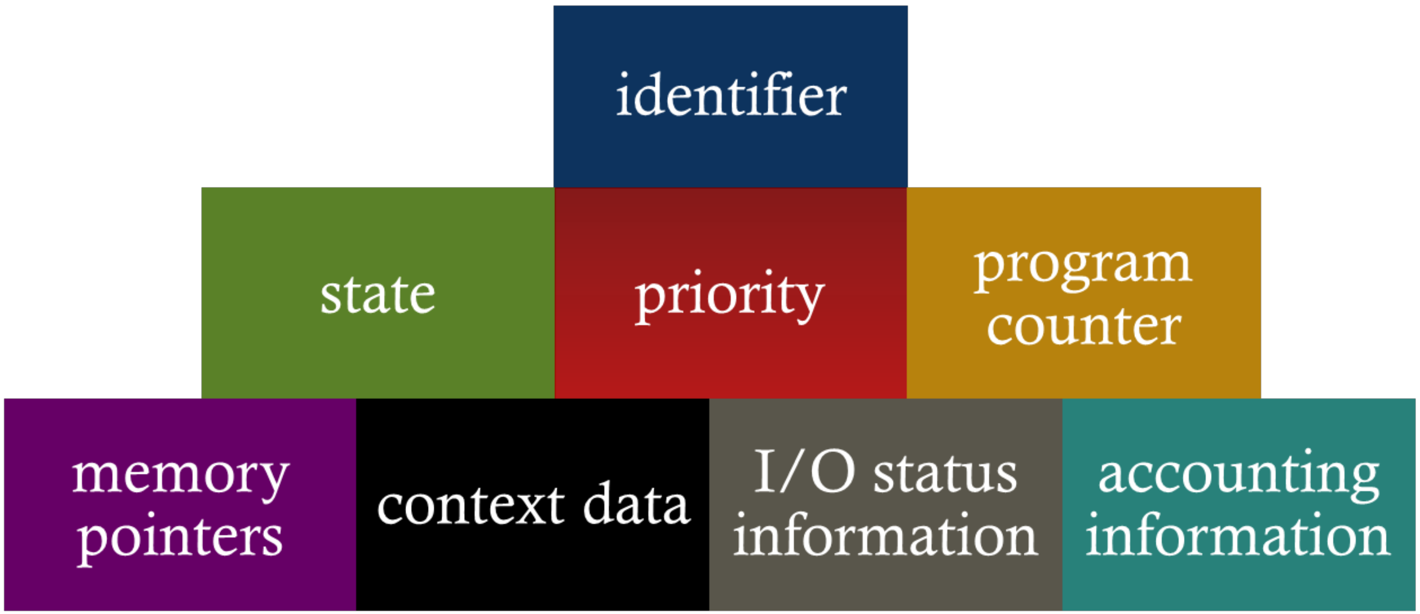
Process Elements

- Two essential elements of a process:
  - Program code — may be shared with other processes that are executing the same program
  - A set of data associated with the program code

When the processor begins to execute the program code, this executing entity is referred as a ***process***

Process Elements (Attributes)

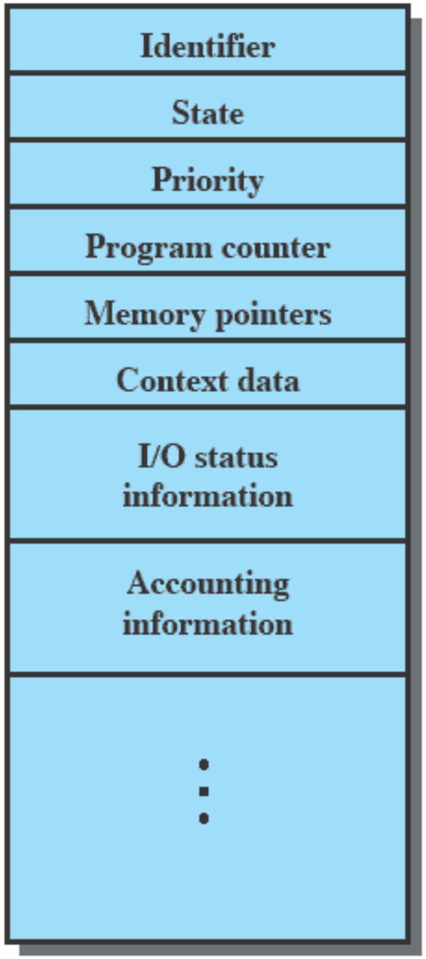
- While the program is executing, the process can be uniquely characterised by a number of elements:



Process Control Block (PCB)

- Data structure — contains the process elements
- Interrupt a running process is possible — later resume execution as if the interruption had not occurred
- Created and managed by OS
- The key tool to support *multiprocessing*

Process = program code + data + PCB



How does a program get executed?

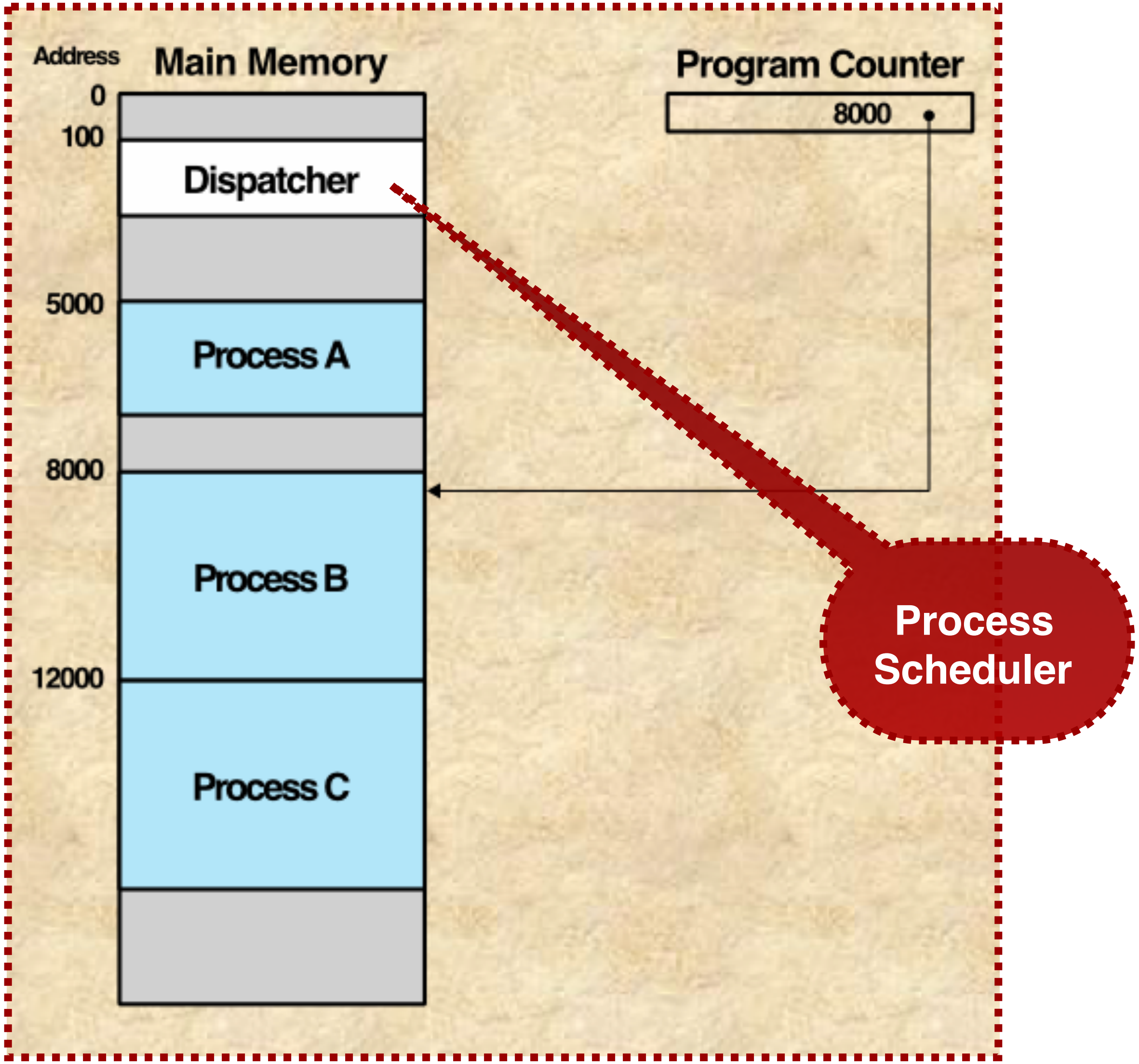
Program Execution

- For a program to be executed, a process (task) is created for that program

- Processor's perspective:
  - Processor executes instructions based on the changing values on the PC register

- Program's perspective:
  - Its execution involves a sequence of instructions within that program

Process Execution: Example



Memory Locations for the Code of the Processes

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

I/O operation is invoked

Traces of Processes

Process A can execute 6 instructions

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

Traces of Processes

Dispatcher decides the next process to be scheduled

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

Traces of Processes

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	

Dispatcher decides the next process to be scheduled and decided on Process B

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

Traces of Processes

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	

Dispatcher decides the next process to be scheduled

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

Traces of Processes

Dispatcher decides the next process to be scheduled and decided on Process C

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

When are processes created?

Process Creation: Reasons

New batch job	The OS is provided with a batch job control stream, usually on tape or disk. When the OS 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 OS 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 Creation: Parent and Child Processes

Process spawning

- when the OS creates a process at the explicit request of another process

Process Creation: Parent and Child Processes

Process spawning	*Parent process*
• when the OS creates a process at the explicit request of another process	• is the original, creating, process

Process Creation: Parent and Child Processes

Process spawning	*Parent process*	*Child process*
• when the OS creates a process at the explicit request of another process	• is the original, creating, process	• is the new process

More on Process Creation

- Once OS decides to create a new process:

Assigns a unique process identifier to the new process

Allocates memory space for the process

Initialises the process control block

Sets the appropriate linkages

Creates or expands other data structures

Process Termination

- There must be a means for a process to indicate its completion

exit system call in Unix/Linux

- A batch job should include a **HALT** instruction or an explicit OS service call for termination

- For an interactive application, the action of the user will indicate when the process is completed
  - E.g. log off, quitting an application, interrupting an interactive process by Control-C)

sending signals to Unix/Linux

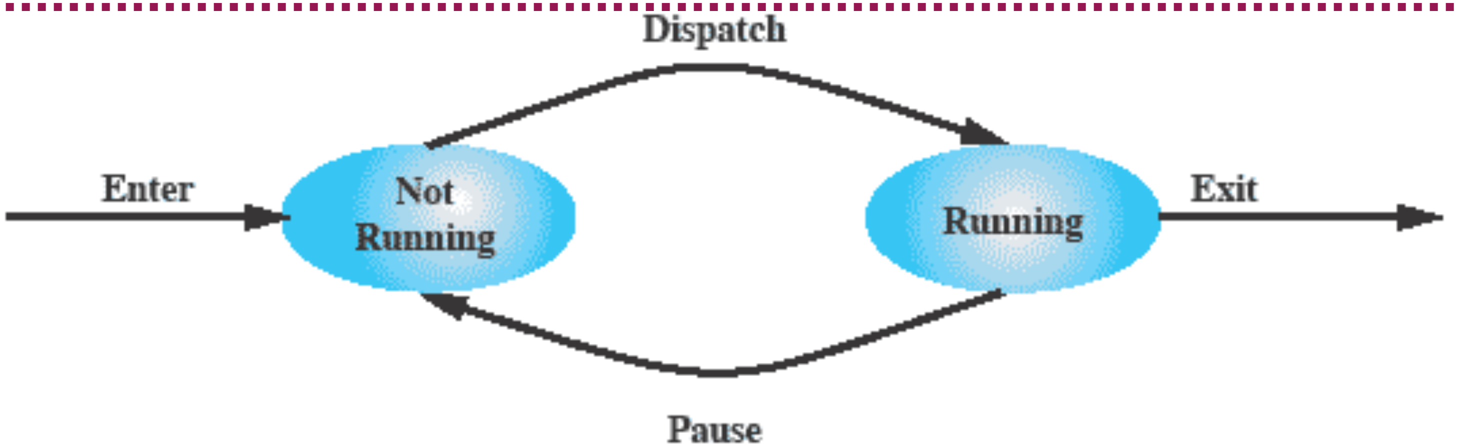
Process Termination: Reasons

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.
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 (e.g., 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.

What is a process state?

Two-State Process Model

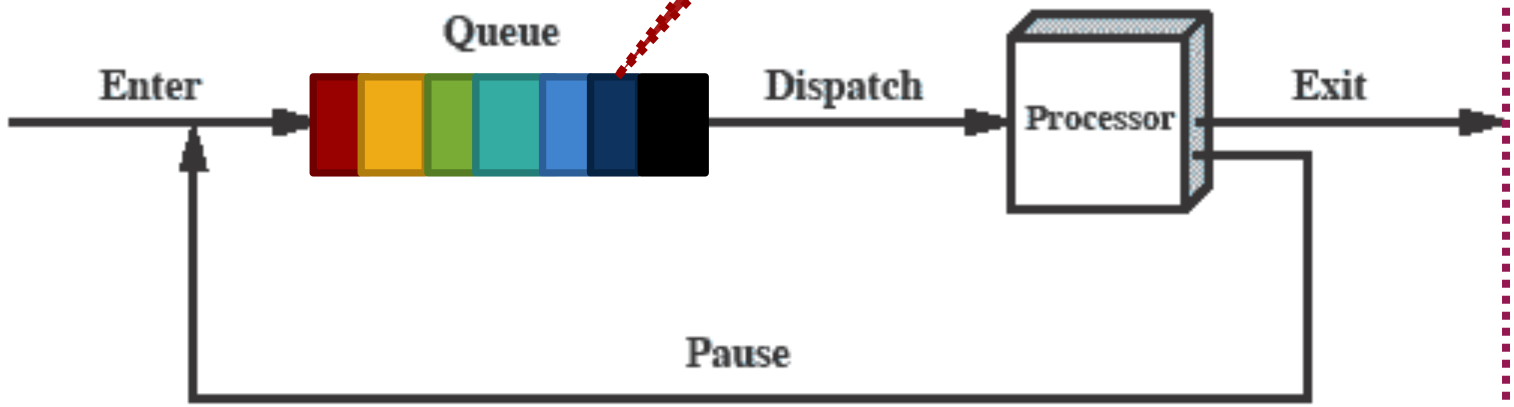
- A process may be in one of the two states:
  - running or not-running



(a) State transition diagram

Processes in a Queue

Pointers to PCB

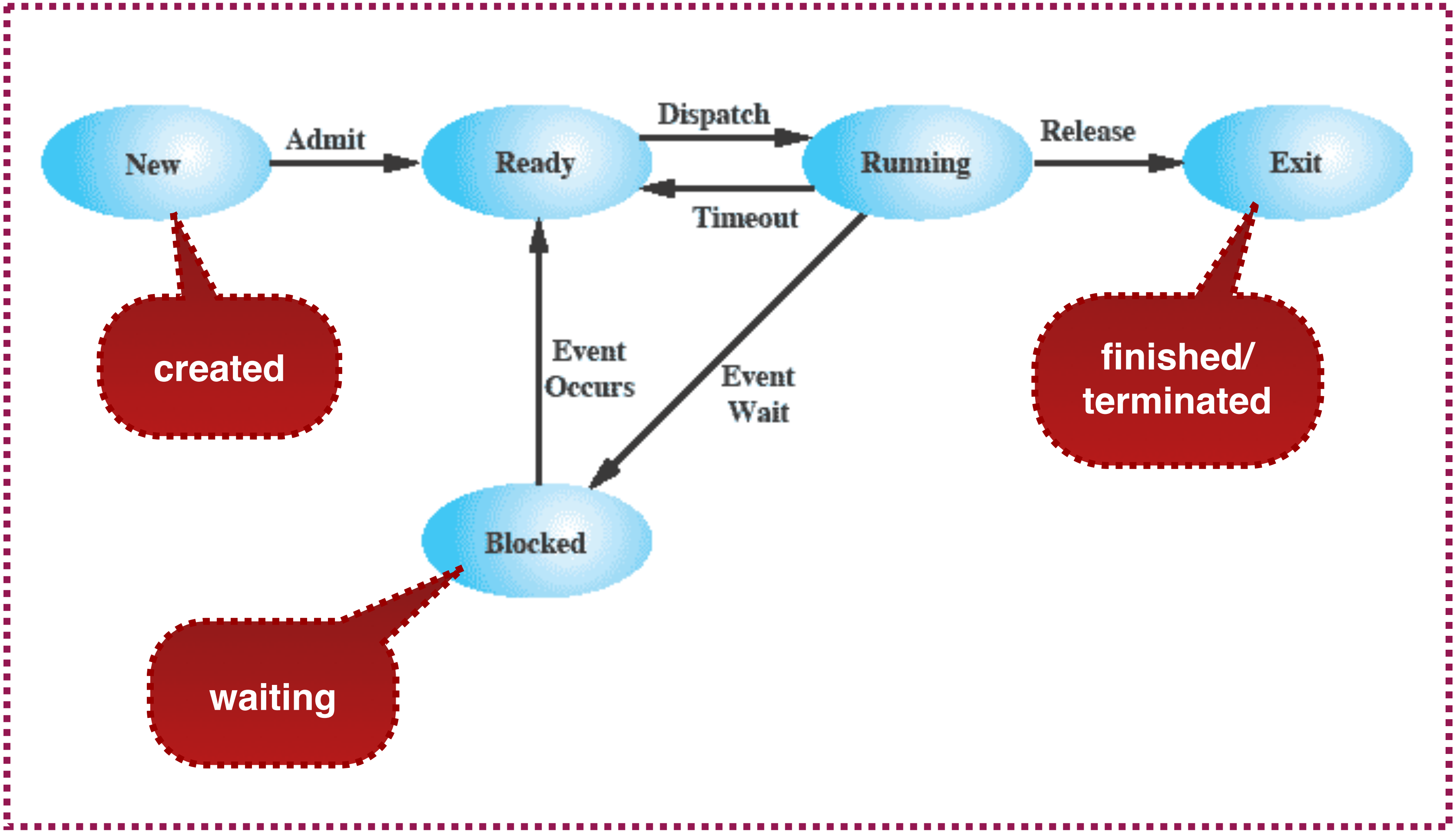


(b) Queuing diagram

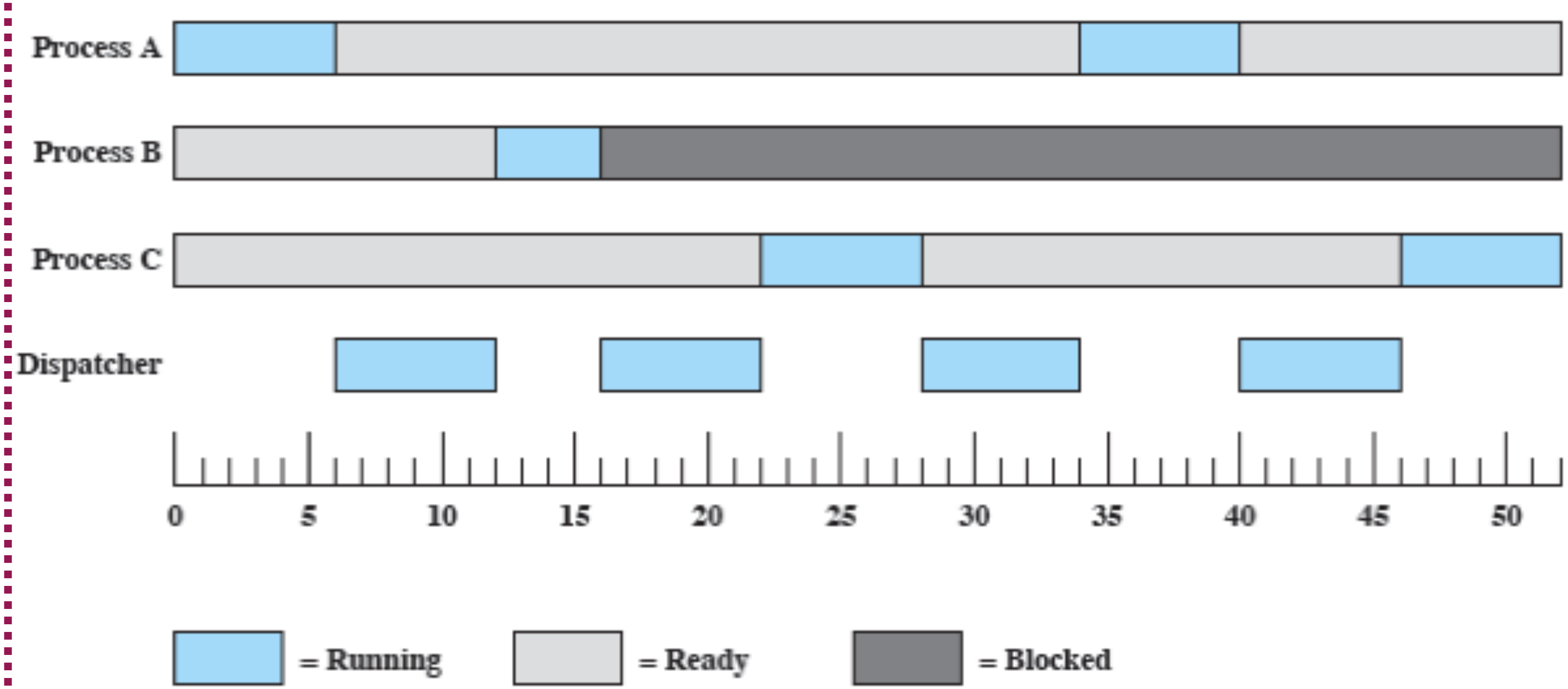
Is one queue sufficient?

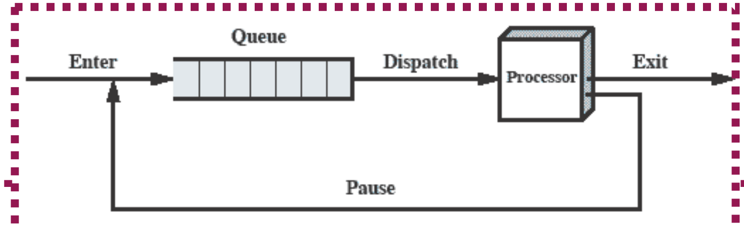
Round Robin Strategy

Five-State Process Model



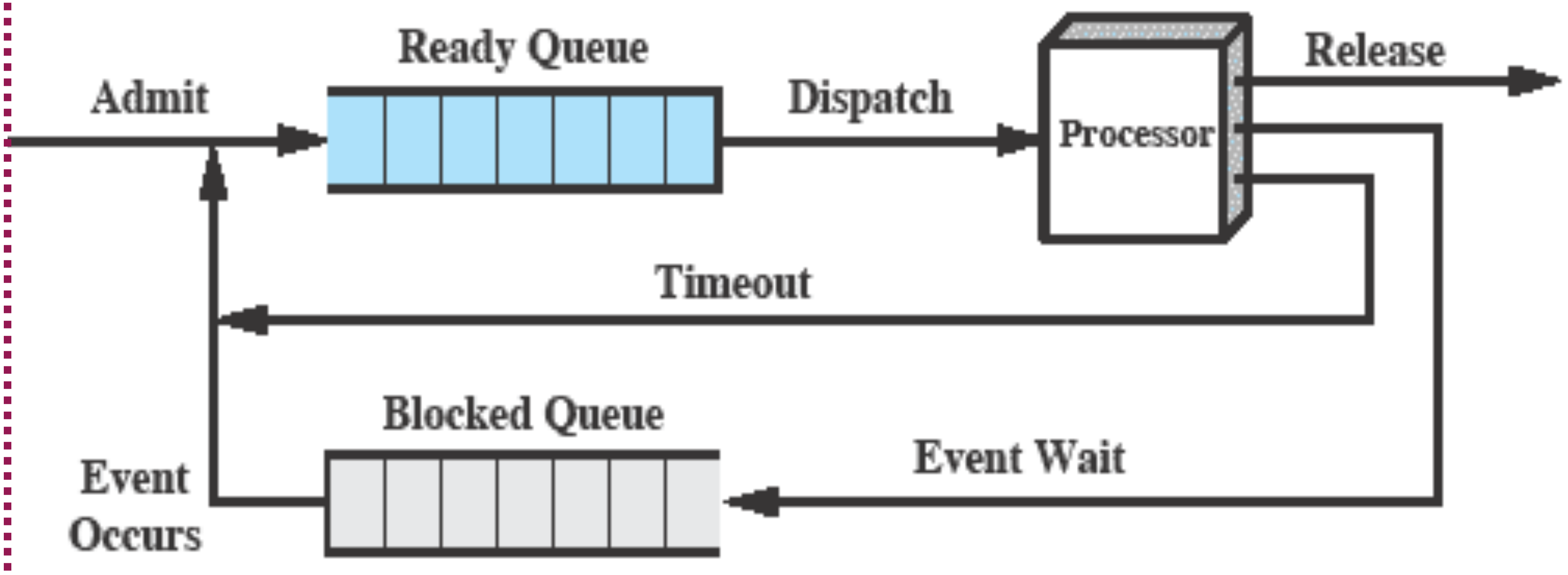
Process States for Tracing Processes in Slide 29





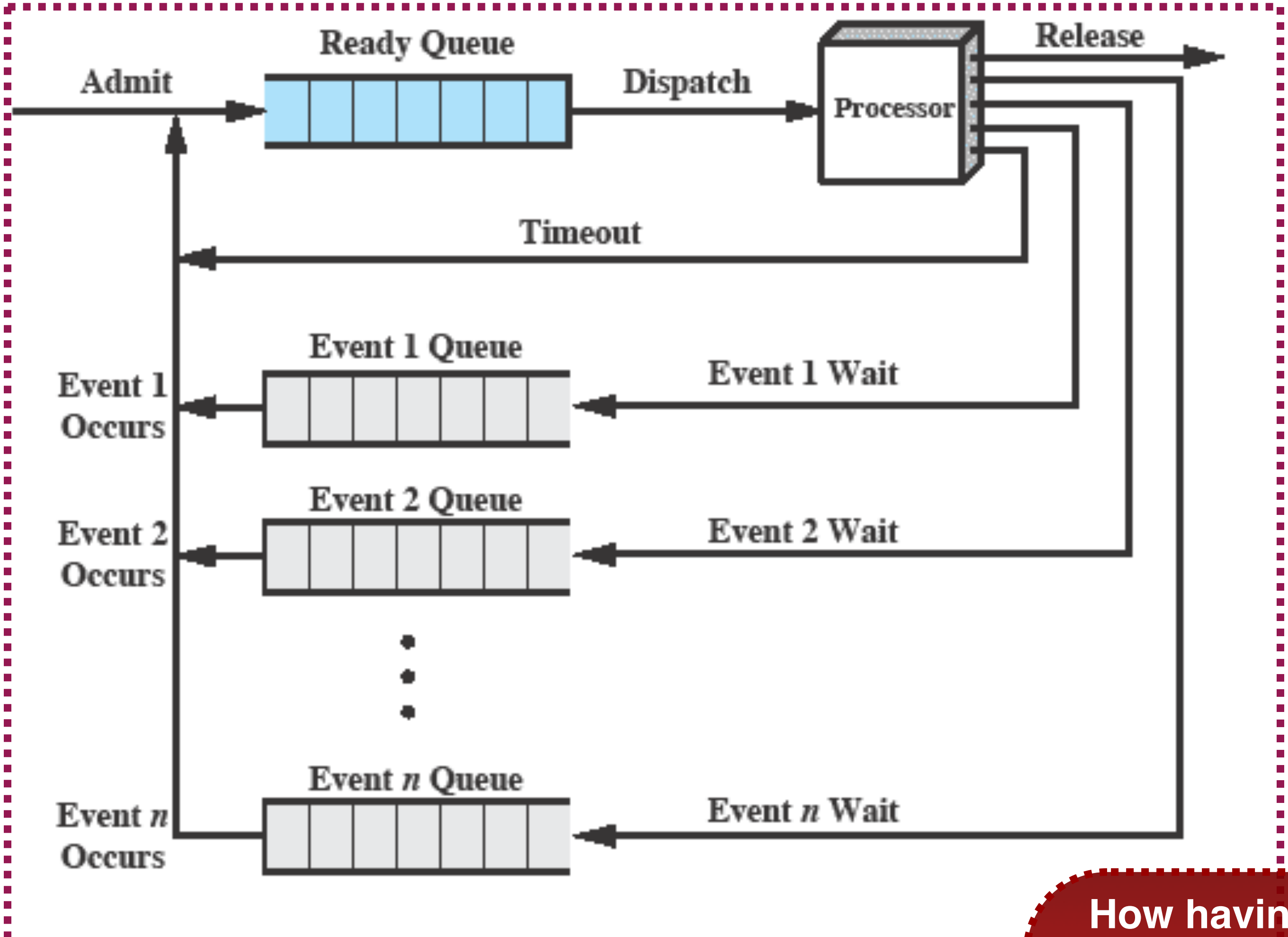
(b) Queuing diagram

Two Process Queues



(a) Single blocked queue

Multiple 'BLOCKED' Queues



(b) Multiple blocked queues

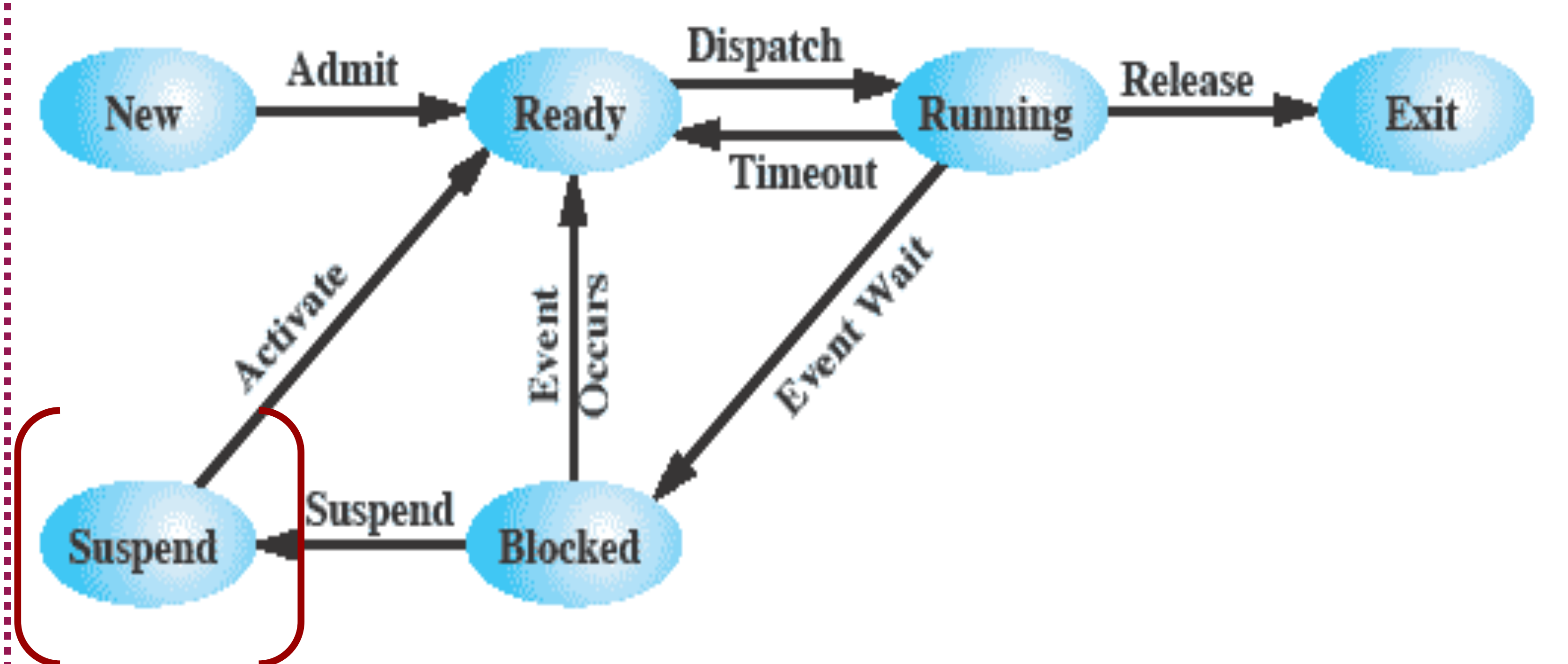
How having about multiple READY queues?

When will a process be suspended?

Suspended Processes

- Swapping:
  - Moving part or all of a process from main memory to disk
  - When none of the processes in main memory is in the READY state, the OS swaps one of the BLOCKED processes out on to disk into a suspend queue
  - OS then brings in another process from the suspend queue or honours a new process request

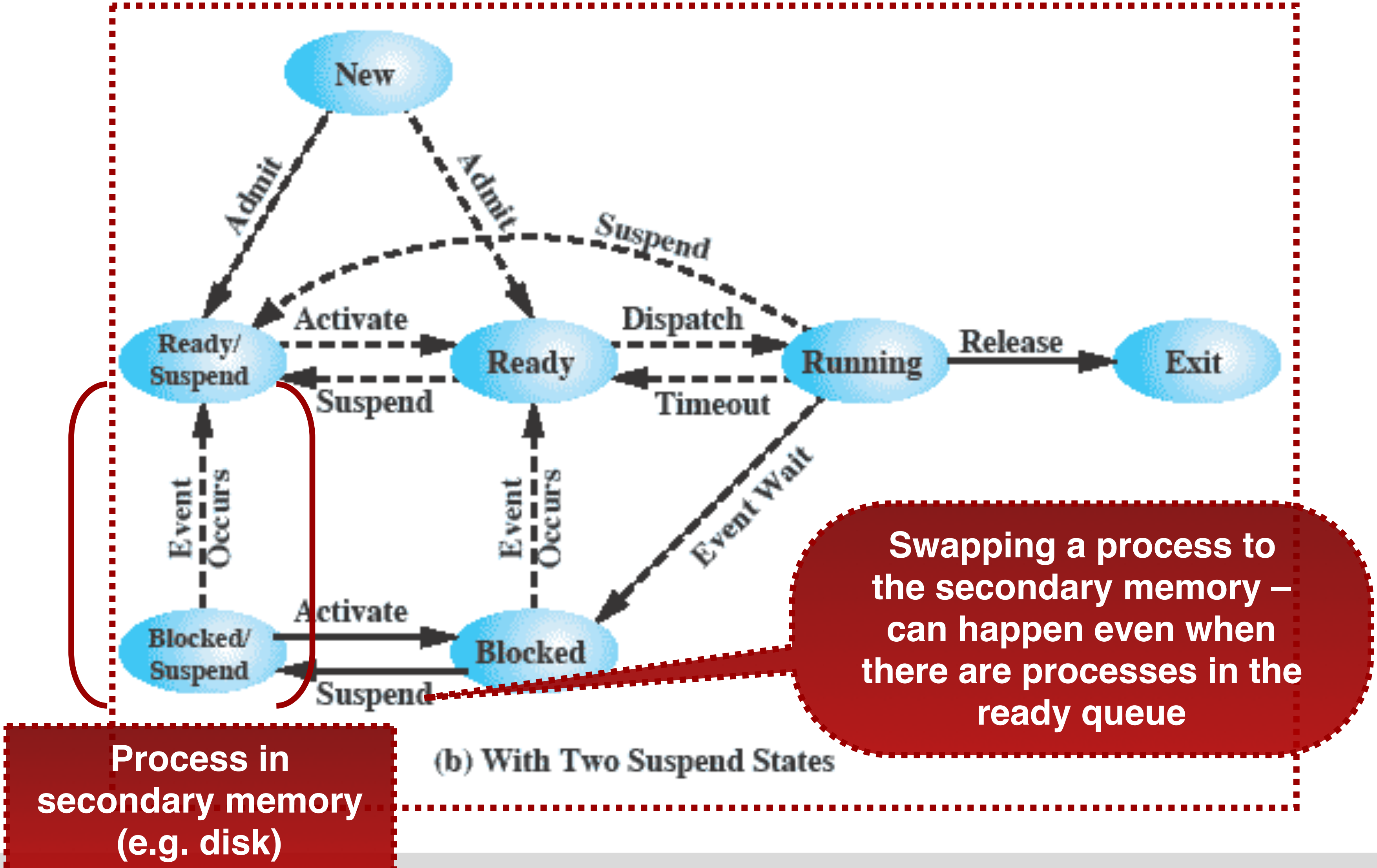
Process Model: One Suspend State



(a) With One Suspend State

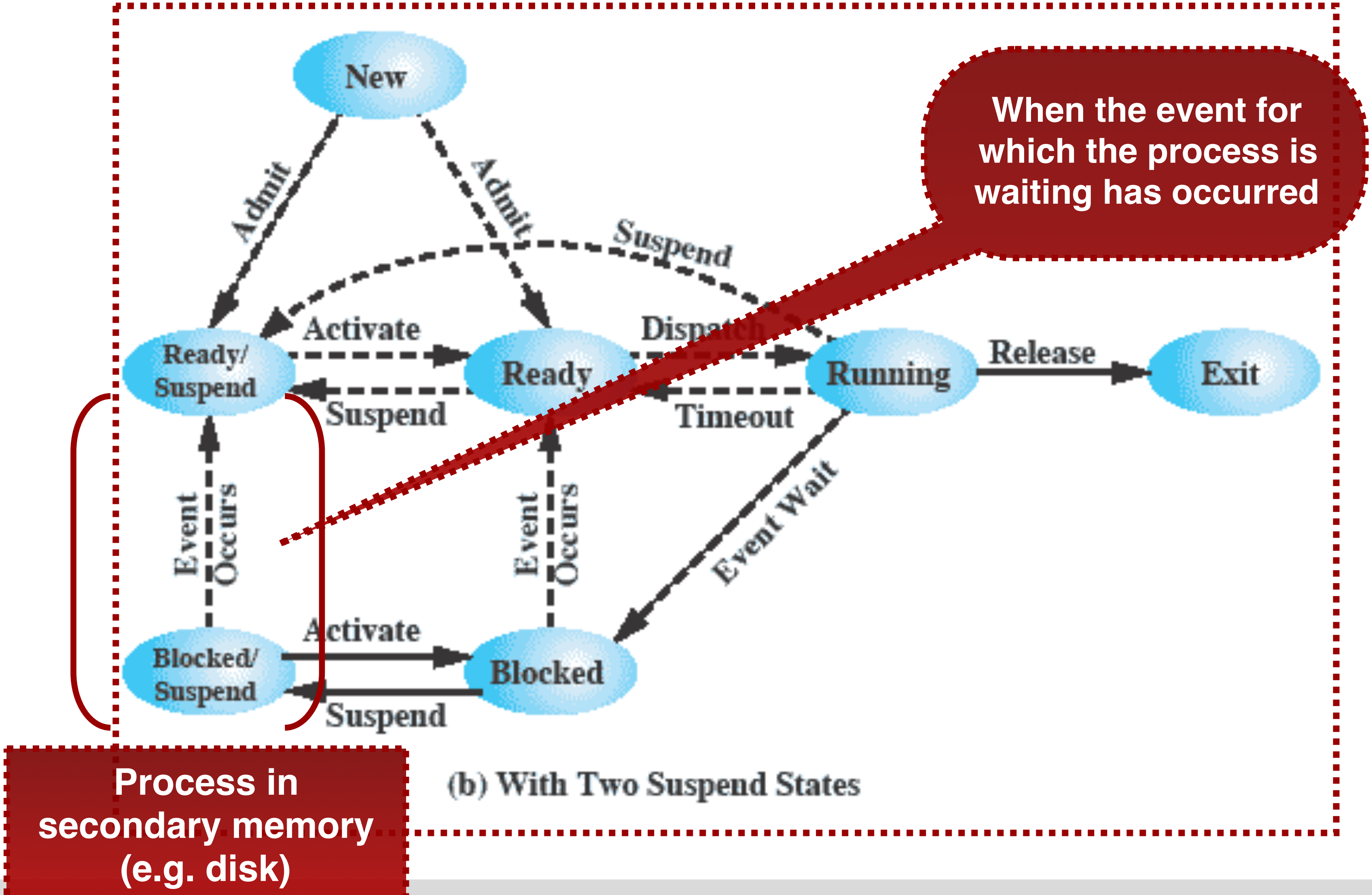
Process in secondary memory (e.g.

Process Model: Two Suspend States



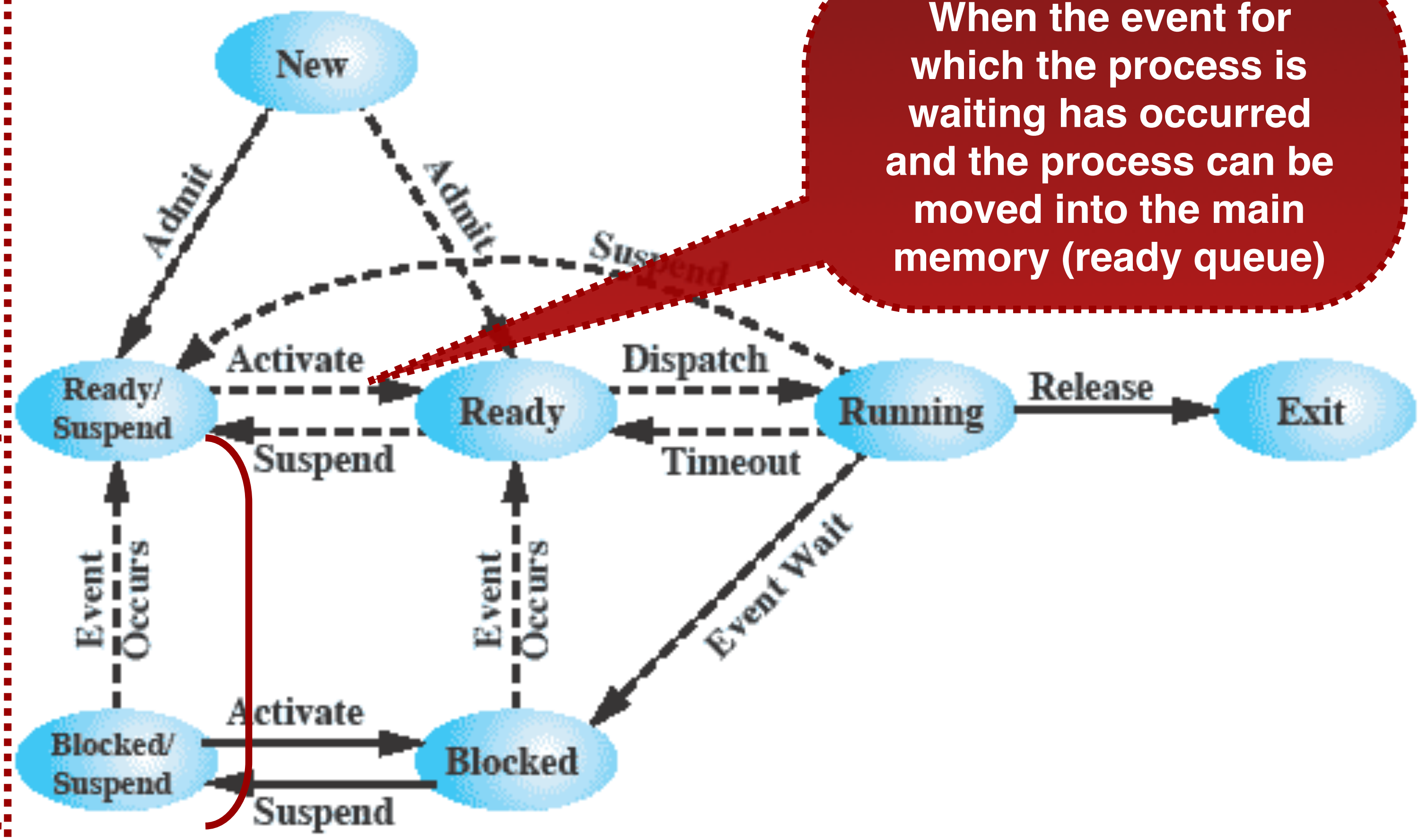
(b) With Two Suspend States

Process Model: Two Suspend States



(b) With Two Suspend States

Process Model: Two Suspend States



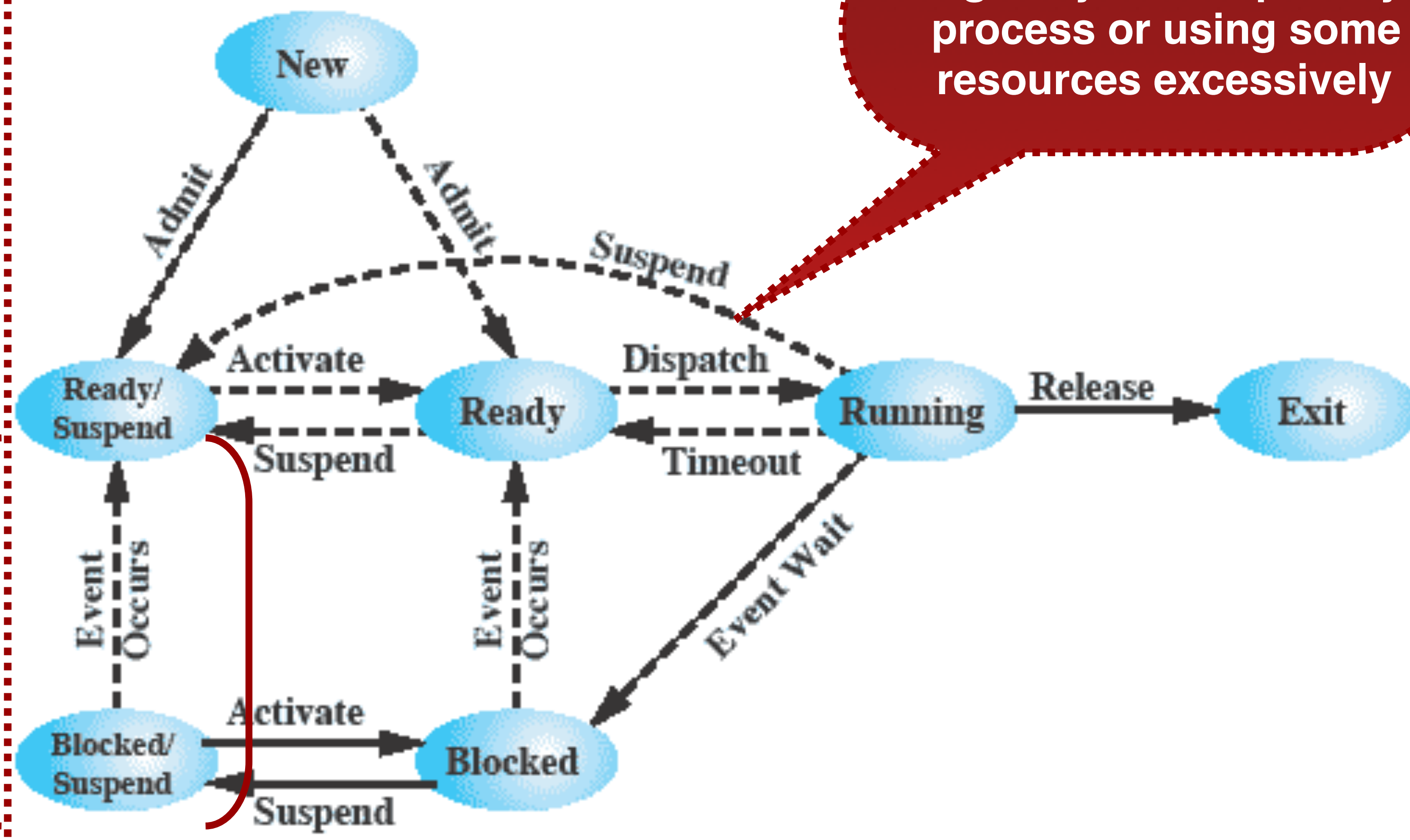
(b) With Two Suspend States

When the event for which the process is waiting has occurred and the process can be moved into the main memory (ready queue)

Process in secondary memory (e.g. disk)

Process Model: Two Suspend States

e.g. may be low priority process or using some resources excessively



(b) With Two Suspend States

Process in secondary memory (e.g. disk)

Suspended Process: Characteristics

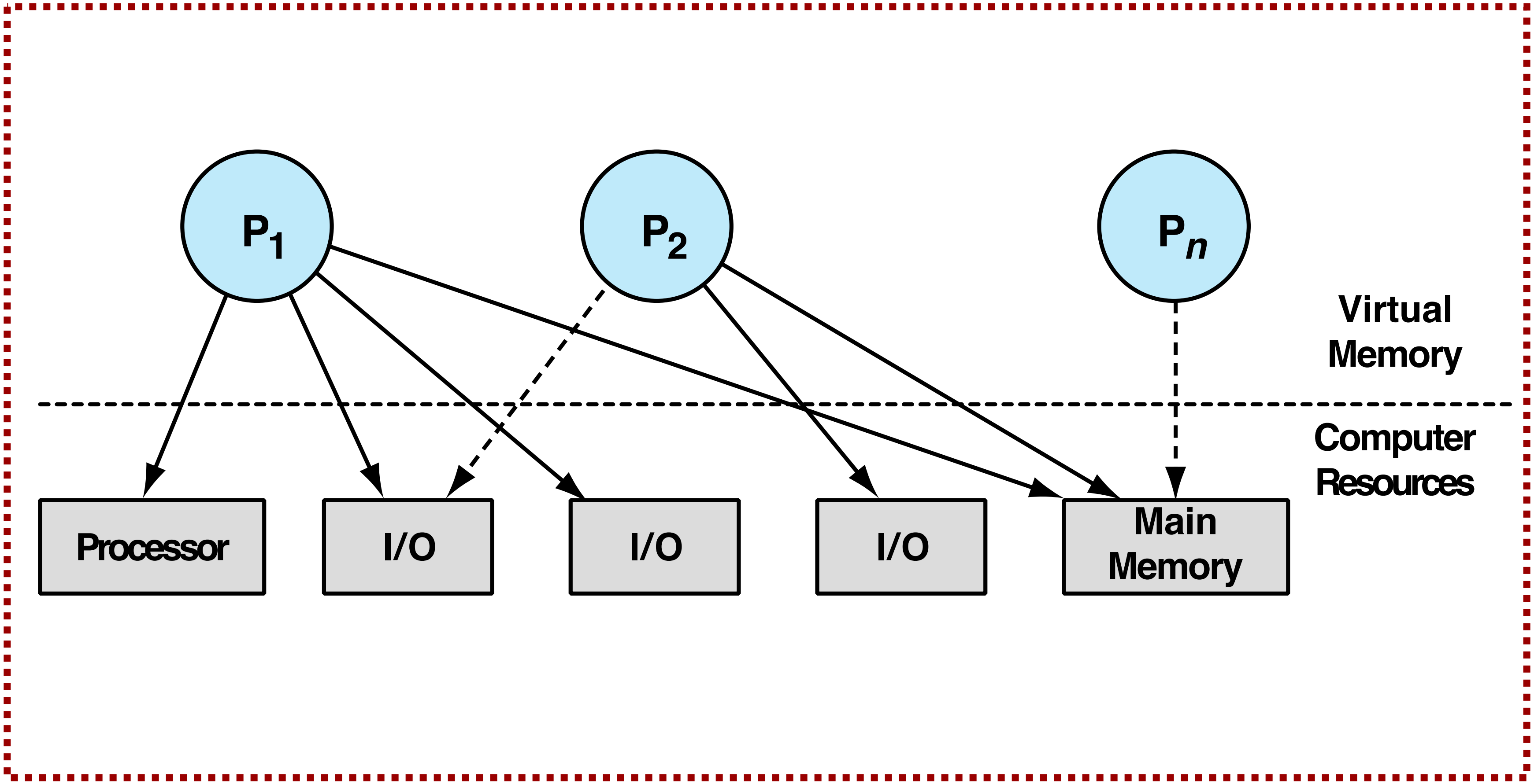
- The process is not immediately available for execution
- The process was placed in a suspended state by an agent — either itself, a parent process, or the OS — for the purpose of preventing its execution
- The process may or may not be waiting on an event
- The process may not be removed from this state until the agent explicitly orders the removal

Process Suspension: Reasons

e.g. Deadlock

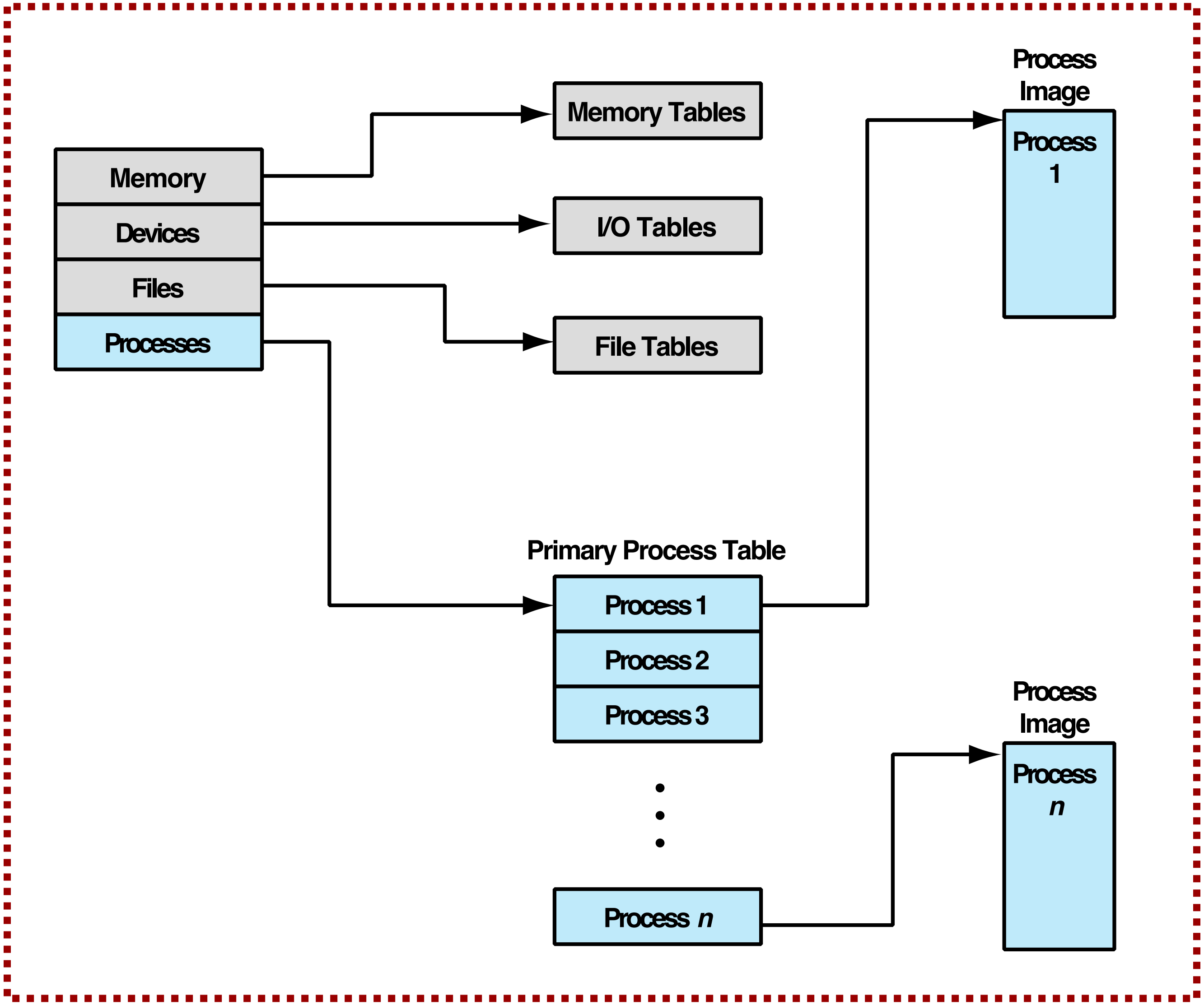
Swapping	The OS needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The OS may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for 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 descendants.

What is the state of each of the processes?



How does OS control and manage resources for processes?

OS Control Tables: General Structure



Process Tables

- Must be maintained to manage processes
- Must be some reference to other system resources — memory, I/O, and files — directly or indirectly
- Must be accessible by the OS — subject to memory management

What does OS need to know?

- To manage and control a process, the OS must know:

Where the process is located

Which attributes of the process that are necessary for its management

What is the physical representation of a process?

Process Location

- A process must include a program or set of programs to be executed
- A process will consist of at least sufficient memory — to hold the programs and data of that process
- Execution of a program typically involves a stack — to keep track of procedure calls and parameter passing between procedures

Process Attributes

PCB

- Each process has associated with it a number of attributes — used by the OS for process control
- Collection of program, data, stack, and attributes is referred as — ***process image***
- Process image location depends on the memory management scheme being used

Process Image: Typical Elements

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.

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 (see Table 3.5).

Process Control Block: Typical Elements (Part I)

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

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.

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

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 Identification

- Each process is assigned a unique numeric **identifier**
  - Index in the primary process table
  - Mapping for OS to locate the appropriate tables based on the process identifier
- Many of the tables controlled by the OS may use process identifiers to *cross-reference* process tables
- When processes communicate with one another, the process identifier informs the OS of the destination of a particular communication
- When processes are allowed to create other processes, identifiers indicate the parent and descendants of each process

Processor State Information

- Consists of the contents of processor registers:
  - User-visible registers
  - Control and status registers
  - Stack pointers

program counter, program status word

- Program status word (PSW):
  - A register or a set of registers
  - Contains condition codes plus other status information

Interrupt enabled, execution mode

Arithmetic overflow

Process Control Information

- Additional information needed by the OS to control and coordinate various active processes

Process Control Block: Typical Elements (Part II)

Process Control Information

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, 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-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.

Interprocess 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.

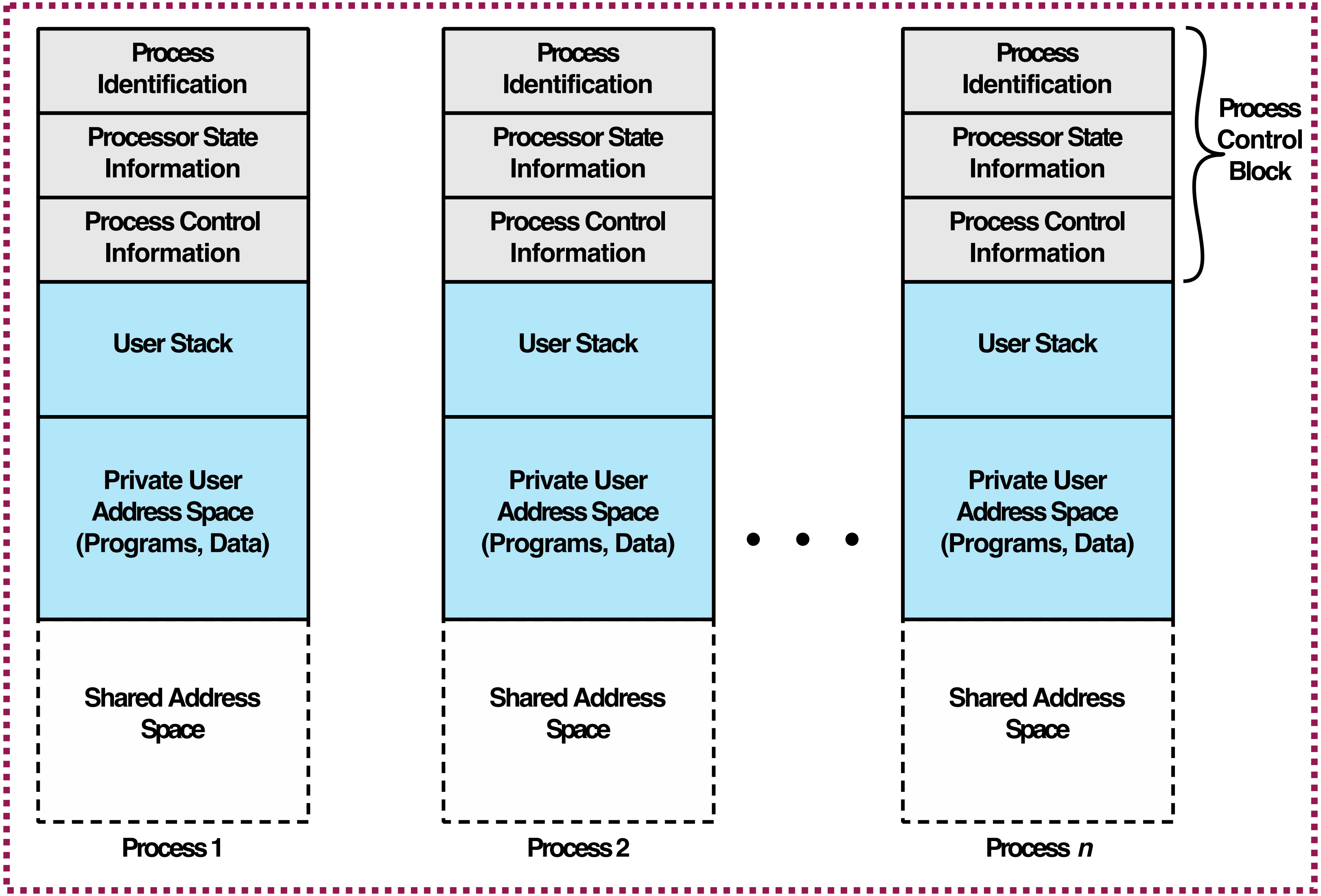
Memory Management

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

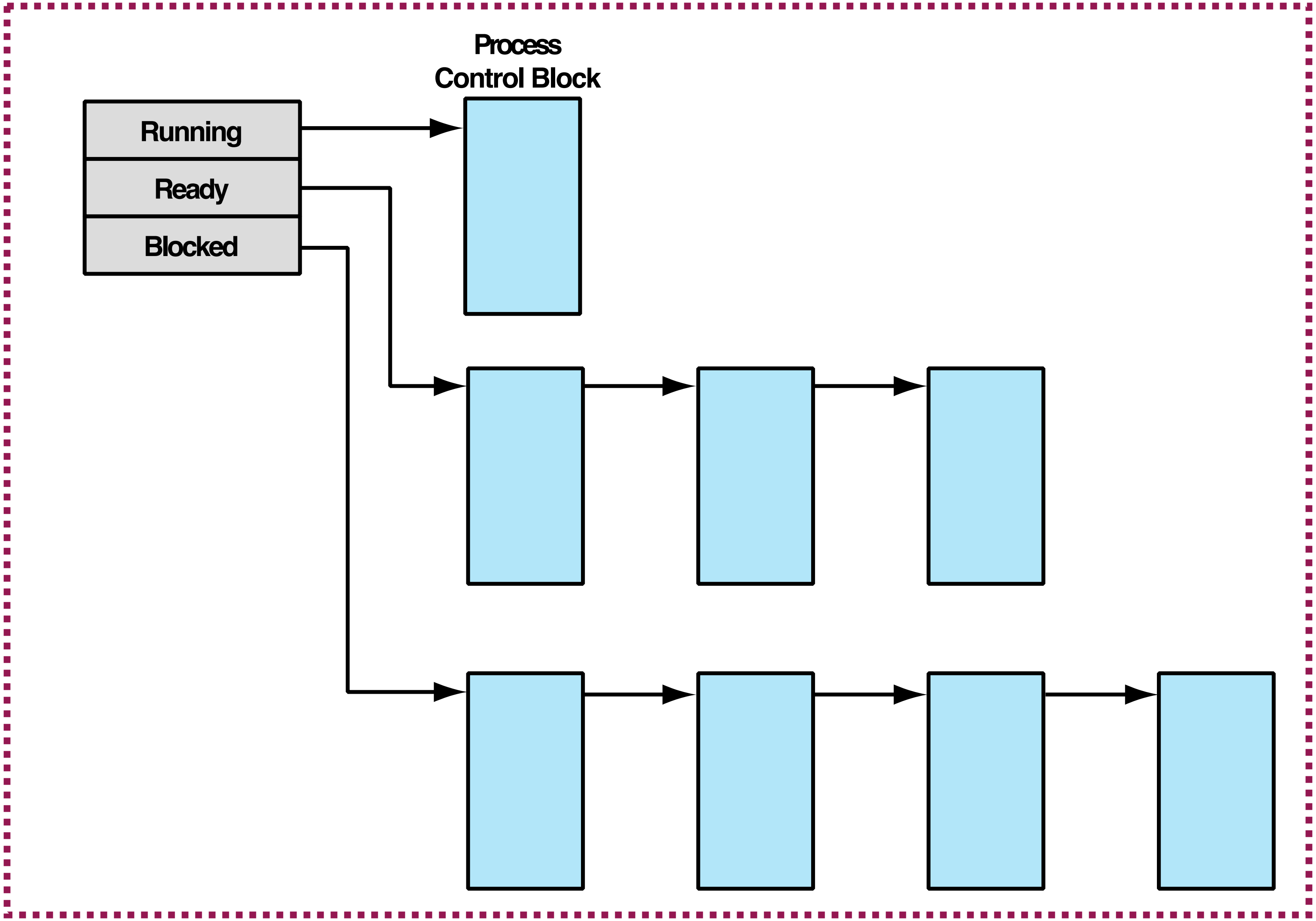
Resource Ownership and Utilization

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.

Process Image: Structure



Process List: Structure



Role of Process Control Block (recap)

- Most important data structure in an OS
- Contains all of the information about a process that is needed by the OS
- Blocks are read and/or modified by virtually every module in the OS

scheduling, resource allocation, interrupt processing

A set of process control blocks defines the state of OS

When a process switch occur?

Process Switching

- Process switch may occur any time that the OS has gained control from the currently running process

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

e.g. when a process has used up its CPU time slice – CPU timer interrupt

MONASH University

Process Switching

- Process switch may occur any time that the OS has gained control from the currently running process

e.g. an arithmetic overflow

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

Mode Switching

If no interrupts are pending the processor:

proceeds to the fetch stage and fetches the next instruction of the current program in the current process

MONASH University

Mode Switching

If no interrupts are pending the processor:

proceeds to the fetch stage and fetches the next instruction of the current program in the current process

If an interrupt is pending the processor:

sets the program counter to the starting address of an interrupt handler program

switches from user mode to kernel mode so that the interrupt processing code may include privileged instructions

Full Process Switch: Steps

1. Save the context of the processor
2. Update the PCB of the process currently in the RUNNING state
3. Move the PCB of this process to the appropriate queue
4. Select another process for execution
5. Update the PCB of the process
6. Update memory management
7. Restore the context of the processor to that which existed at the time the selected process was last

If the currently running process is to be moved to another state (READY, BLOKED, etc.), then the OS must make substantial changes in its environment

MONASH University

OS Kernel: Typical Functions

Process Management

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

Memory Management

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

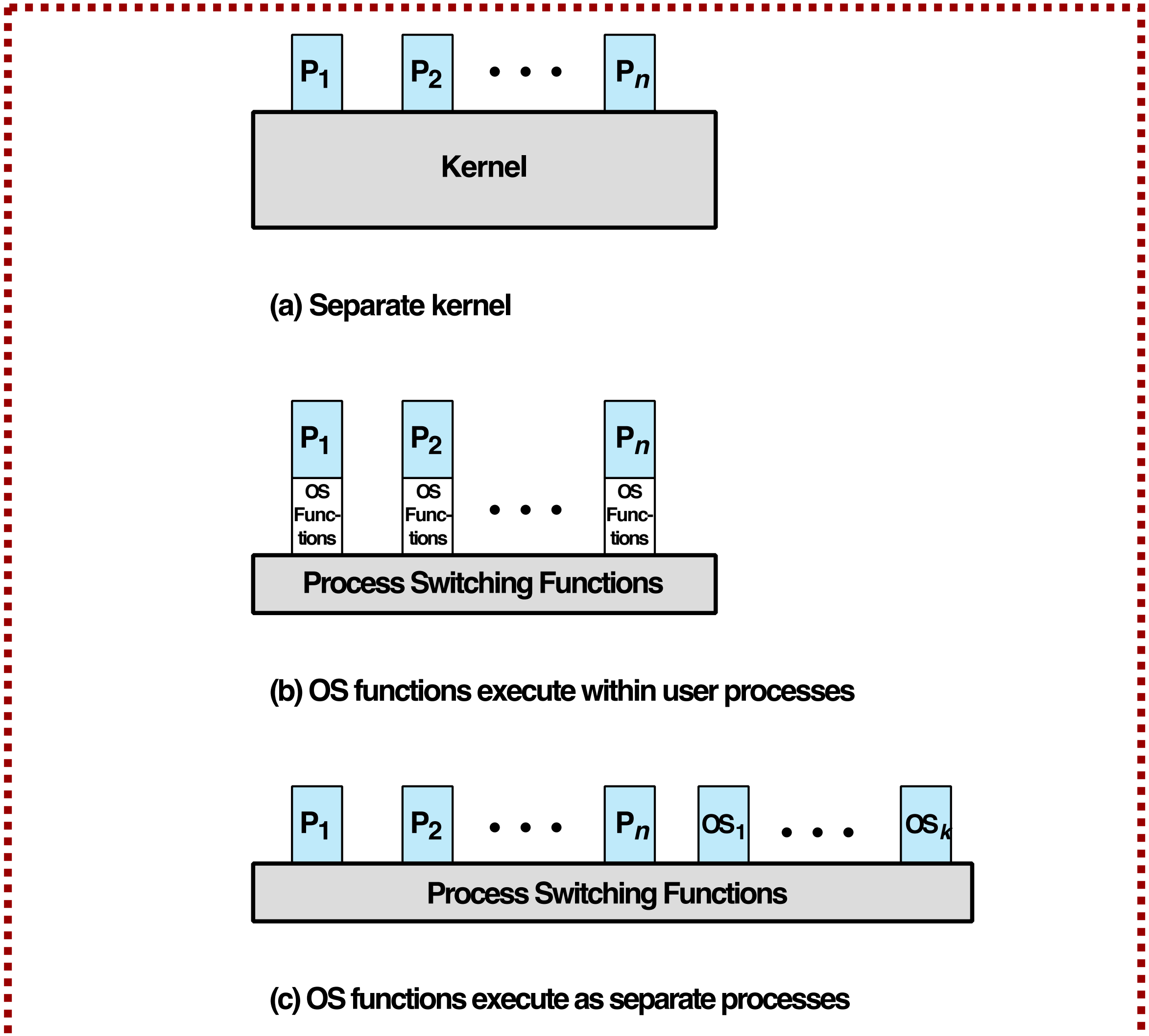
I/O Management

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

Support Functions

- Interrupt handling
- Accounting
- Monitoring

Execution of OS Functions



What are the process states defined in Unix OS?

Unix: Processes

- Adopts the model where most of the OS executes within the environment of a user process
- **System Processes** — run in *kernel* mode
  - Executes operating system code to perform administrative and housekeeping functions

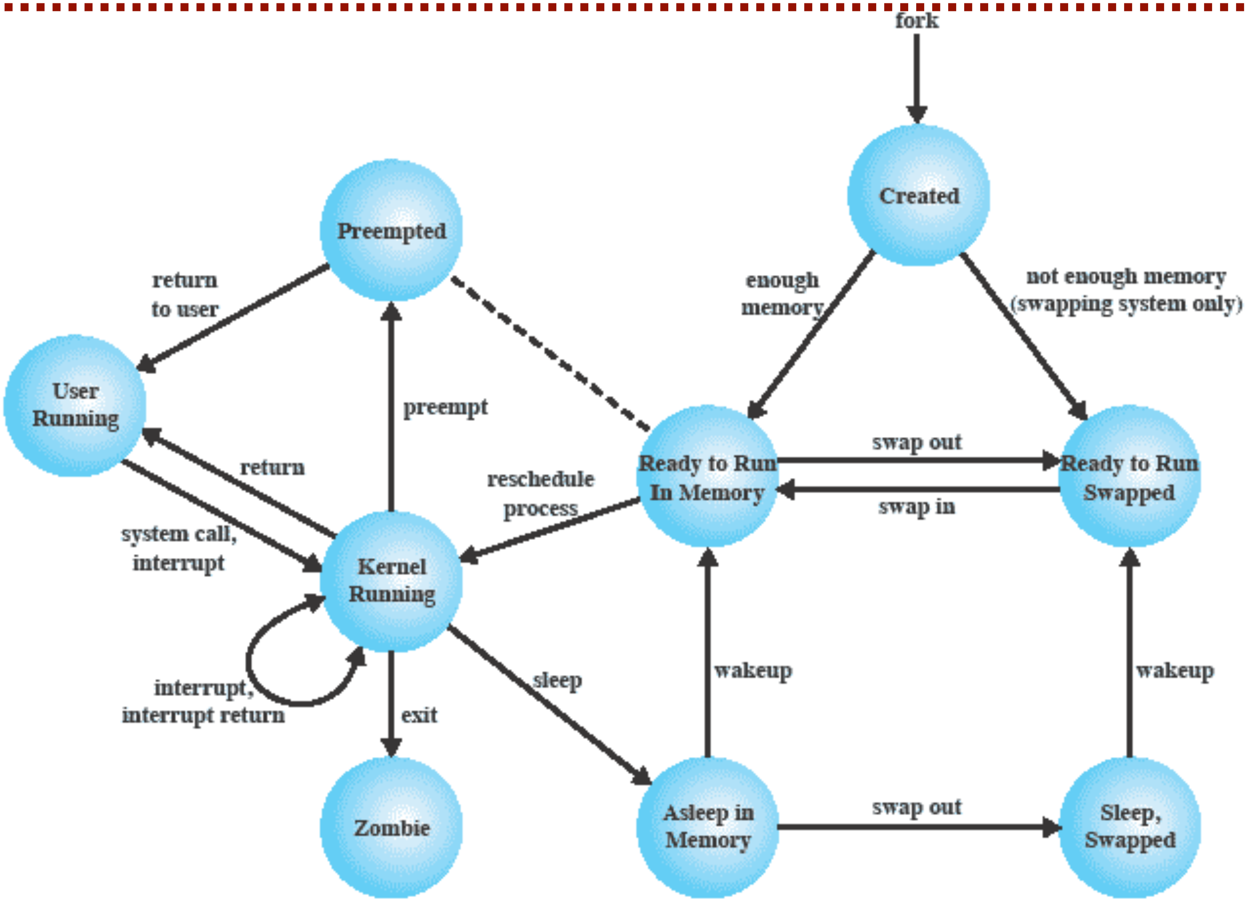
memory allocation, process swapping

- **User Processes**
  - Operate in *user* mode to execute user programs and utilities
  - Operate in *kernel* mode to execute instructions that belong to the kernel — by issuing a system call, when an exception is generated, or when an interrupt occurs

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 State Transition



UNIX Process State Transition Diagram

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
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
System-Level Context	
Process table entry	Defines state of a process; this information is always accessible to the operating system
U (user) 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

Unix: Process Creation

- Process creation is through the kernel system call — **fork()** — which 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. Make a copy of the process image of the parent, with the exception of any shared memory
4. Increments counters for any files owned by the parent, to reflect that an additional process now also owns those files
5. Assigns 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

Unix: After Process Creation

- OS 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

Parent process continues to execute

Child process begins to execute

Both parent and child are turned into “Ready to Run” state

Summary

- So far, we have discussed:
  - Fundamental concepts of process
  - Process control block (PCB)
  - Process states and state transitions
  - Unix process states

- Next week:
  - Concepts of threads
  - Principles of concurrency