Operating Systems

Processes - I

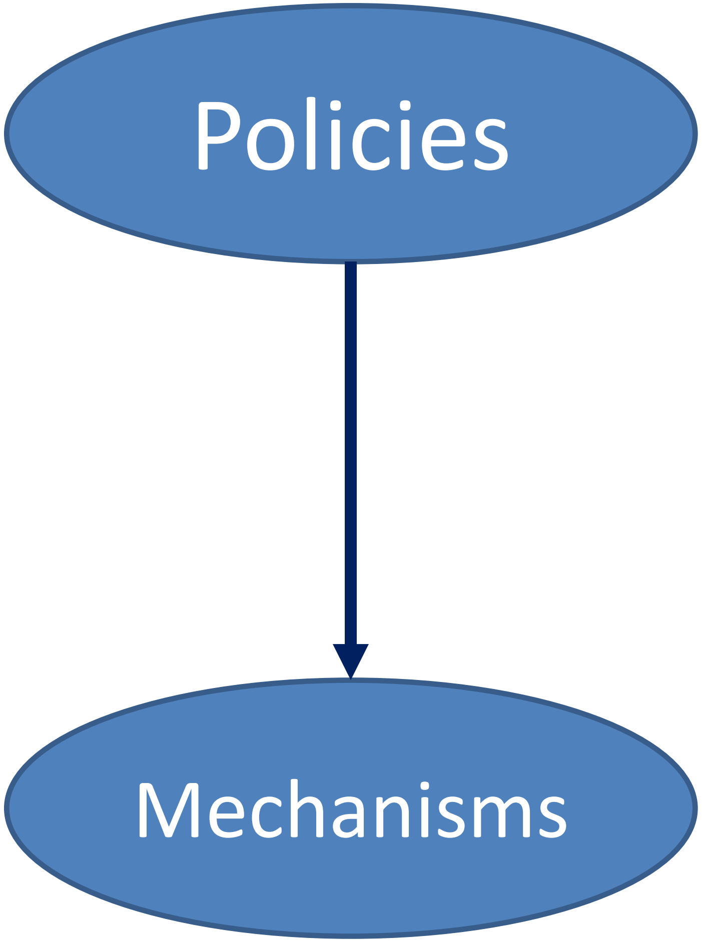
Mohamed Zahran (aka Z)

mzahran@cs.nyu.edu

http://www.mzahran.com

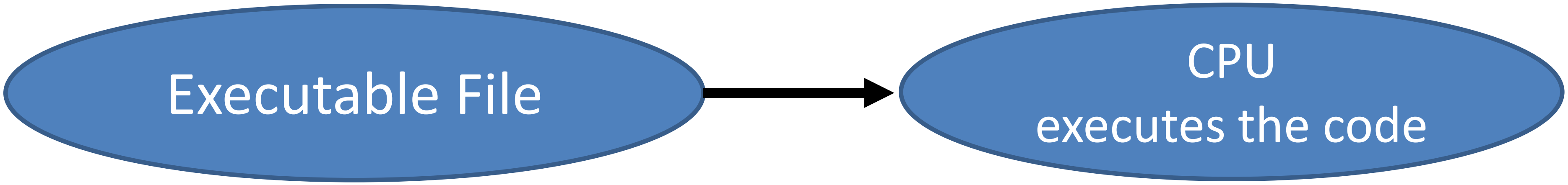
OS Management of Application Execution

- Resources are made available to multiple applications.
- The CPU(s) is/are switched among multiple processes so all will appear to be progressing.
  - This is called time-sharing.
- **Main Goal:** The processor, memory, and I/O devices can be used efficiently
- In the *processes lectures*, we will concentrate on how to use CPUs efficiently among processes.
- This requires some mechanisms and policies.



- A set of algorithms
- Make high level decisions:
    - Which process to run next?
    - When to stop a process?
    - …
- Decision is based on some criteria:
    - Performance
    - Power
    - Priority
    - ….

- Low-level protocol to implement a functionality
- Example:
    - How to switch one process for the other (context-switching)
    - Collecting process’s metrics



Sometime called:

- Object file
- Binary file

Initially, there is an executable file of a program residing on your disk.

A command is issued to the OS to start executing this program.

The loader, part of the OS, extracts needed info from the executable file, loads them in memory, the OS does some initialization, and the CPU is told to start executing.

When the CPU begins to execute the program code, we refer to this executing entity as a ***process***

Three Kinds of Object Files (sometimes called modules)

- **Relocatable object file** (`.o` file)
  - Contains code and data in a form that can be combined with other relocatable object files to form executable object file.
    - Each `.o` file is produced from exactly one source (`.c`) file

- **Executable object file** (`a.out` file)
  - Contains code and data in a form that can be copied directly into memory and then executed.

- **Shared object file** (`.so` file)
  - Special type of relocatable object file that can be loaded into memory and linked dynamically, at either load time or run-time.
  - Called *Dynamic Link Libraries* (DLLs) by Windows

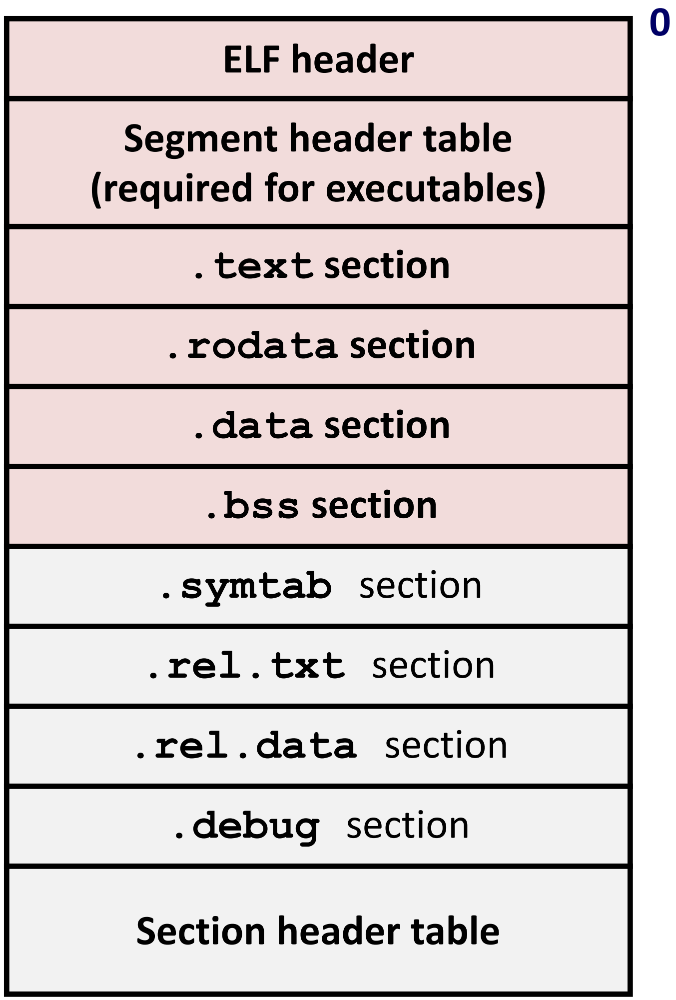
Executable and Linkable Format (ELF)

- Standard binary format for object files
  - Originally proposed by AT&T System V Unix, later adopted by BSD Unix variants and Linux
- One unified format for
  - Relocatable object files (`.o`),
  - Executable object files (`a.out`)
  - Shared object files (`.so`)

- Generic name: ELF binaries

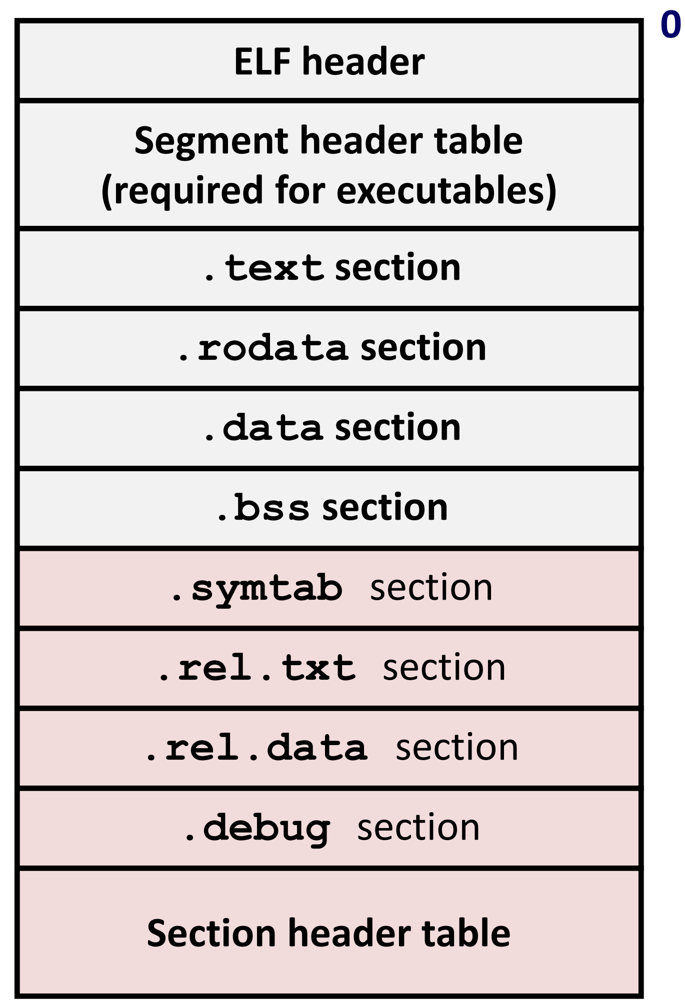
ELF Object File Format

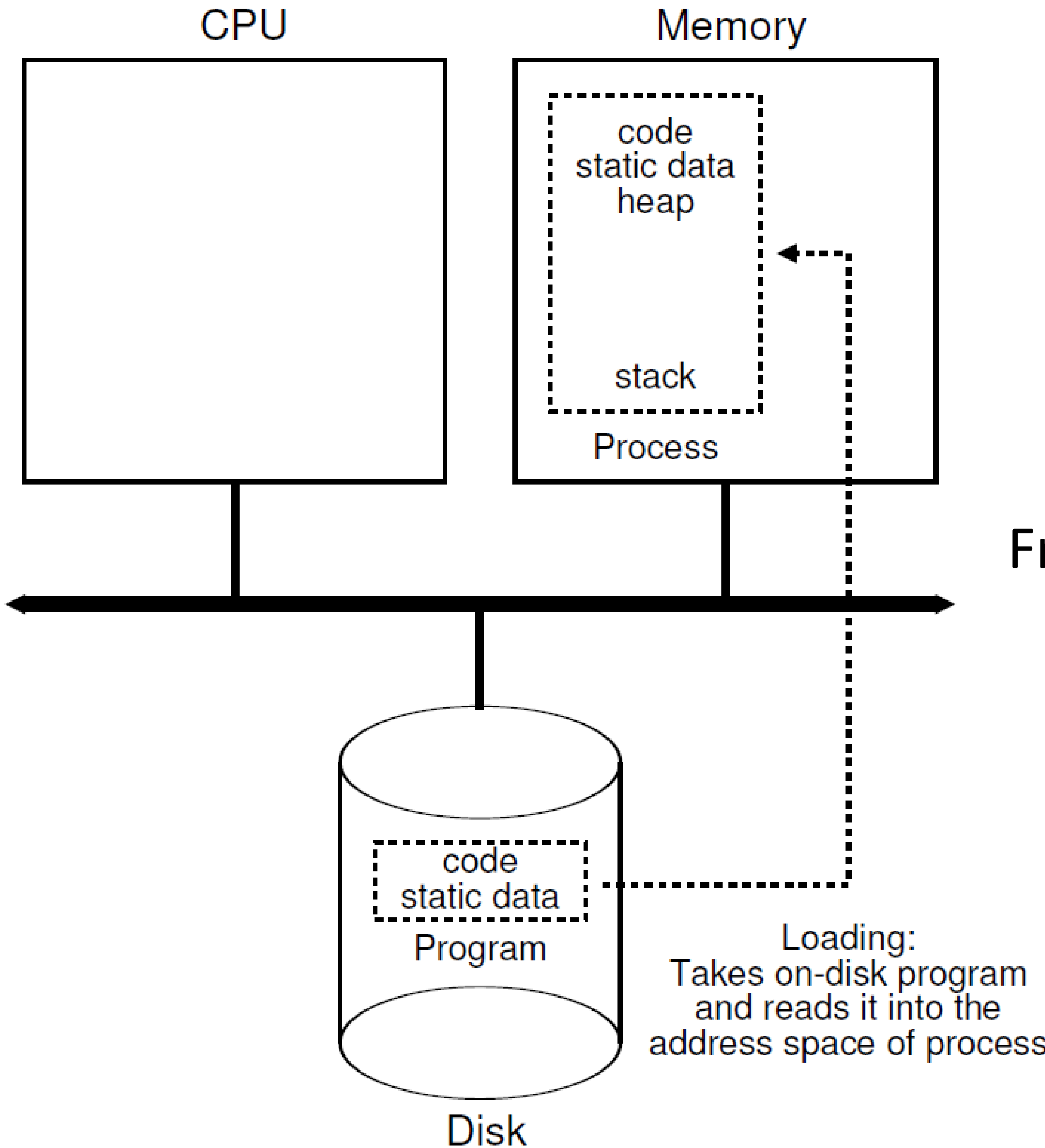
- Elf header
  - Word size, byte ordering, file type (.o, exec, .so), machine type, etc.
- Segment header table
  - Page size, virtual addresses memory segments (sections), segment sizes.
- `.text` section
  - Code
- `.rodata` section
  - Read only data: jump tables, ...
- `.data` section
  - Initialized global variables
- `.bss` section (Block Started by Symbol)
  - Uninitialized global variables
  - Only the length but no data
  - Later, the program loader will allocate memory for it.



ELF Object File Format (cont.)

- `.symtab` section
  - Symbol table
  - Procedure and global variable names
- `.rel.text` section
  - Relocation info for **`.text`** section
  - Addresses of instructions that will need to be modified in the executable
- `.rel.data` section
  - Relocation info for **`.data`** section
  - Addresses of pointer data that will need to be modified in the merged executable
- `.debug` section
  - Info for symbolic debugging (**`gcc -g`**)
- Section header table
  - Offsets and sizes of each section





From a program
to a process.

The Process Model

A process is an instance of an executing program and includes

- Program counter
- Registers
- Variables
- Code
- …

If a program is running twice, does it count as two processes? or one?

Process

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

Where do we store all this info?

identifier

state

priority

program counter

memory pointers

context data

I/O status information

accounting information

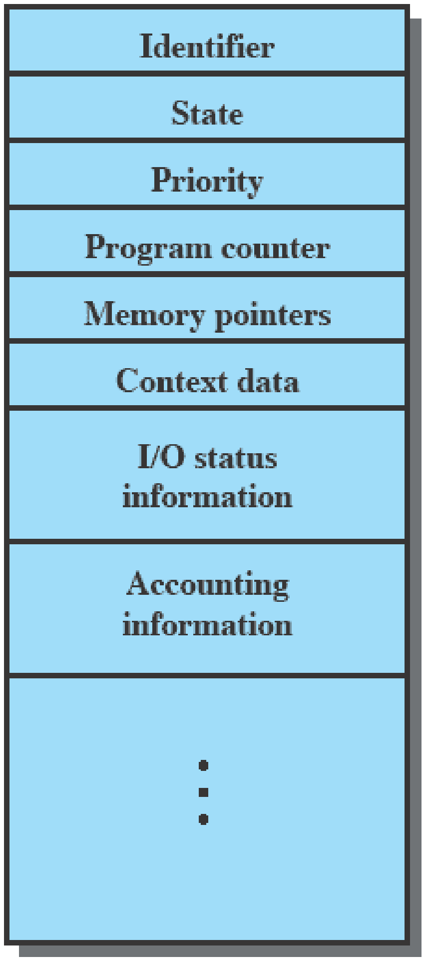
Data Structure

- The OS keeps a list of all processes in the system. → process list (also called process table or control table).

- Each entry of this list contains information about a process. → process control bock

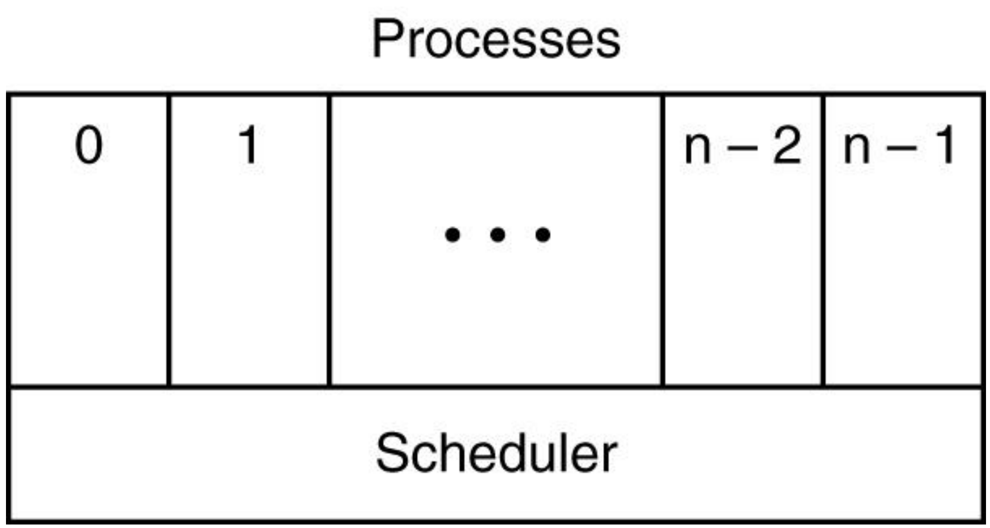
Process Control Block

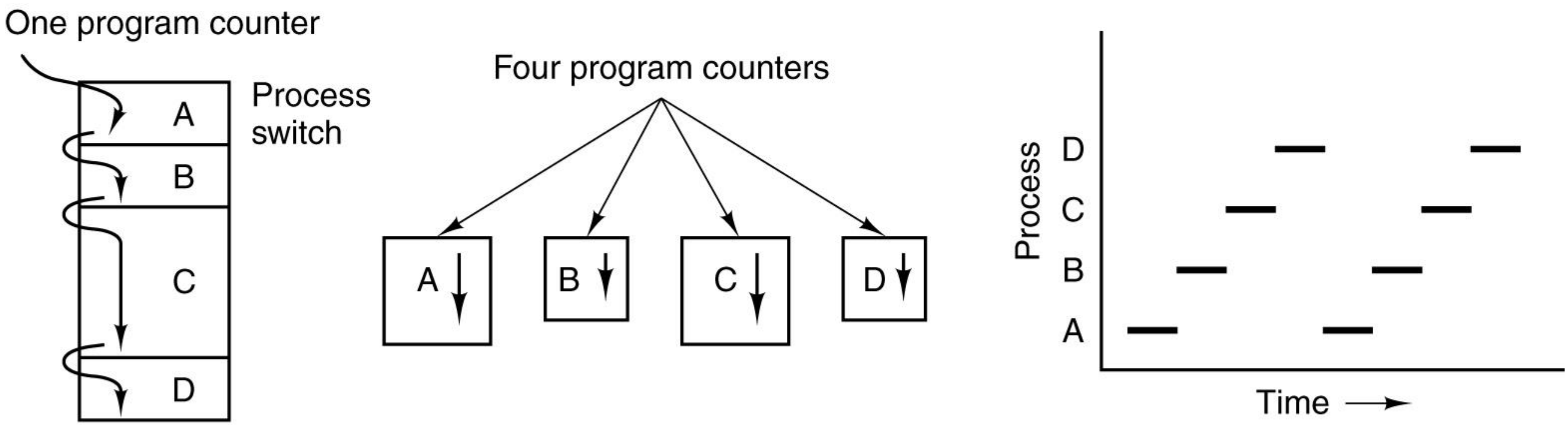
- Contains the process elements
- It is possible to interrupt a running process and later resume execution as if the interruption had not occurred
- Created and managed by the operating system
- Key tool that allows support for multiple processes



Multiprogramming

- One CPU and several processes
- CPU switches from process to process quickly

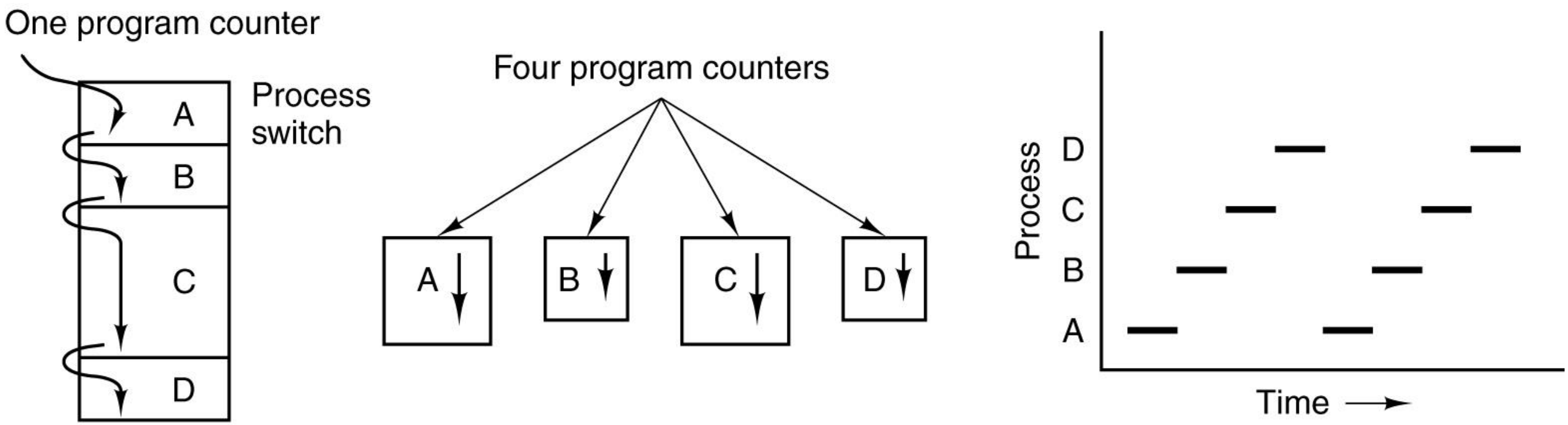




(a) (b)

What Really Happens **What We Think It Happens**

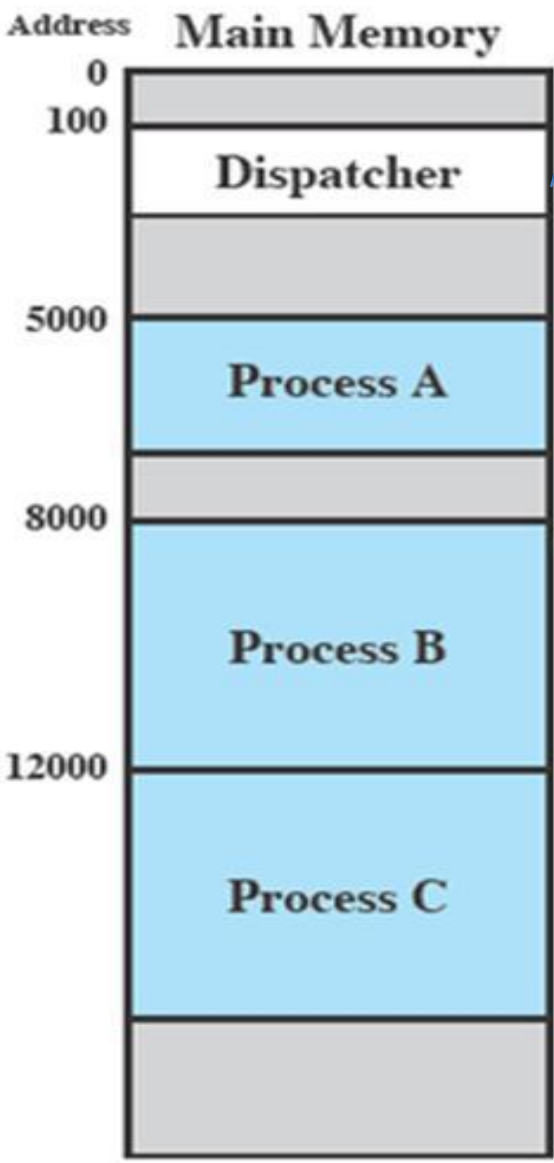
If we run the same program several times, will we get the same execution time?



(a) (b)

What Really Happens **What We Think It Happens**

Example



time	Instruction address
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	

Small program that switches the processor from one process to another (also called Scheduler)

Operations with Processes (i.e. APIs)

- Creation
- Termination or destroying
- Status
- Wait
- Misc: suspend, resume, inquire, …

Process Creation: When Does it Happen?

- System initialization
  - At boot time
  - Foreground
  - Background (daemons)
- Execution of a process creation system call by a running process
- A user request
- A batch job
- Created by OS to provide a service
- Interactive logon

Process Termination: When Does it Happen?

- Normal exit (voluntary)
- Error exit (voluntary)
- Fatal error (involuntary)
- Killed by another process (involuntary)

Process Termination: More Scenarios

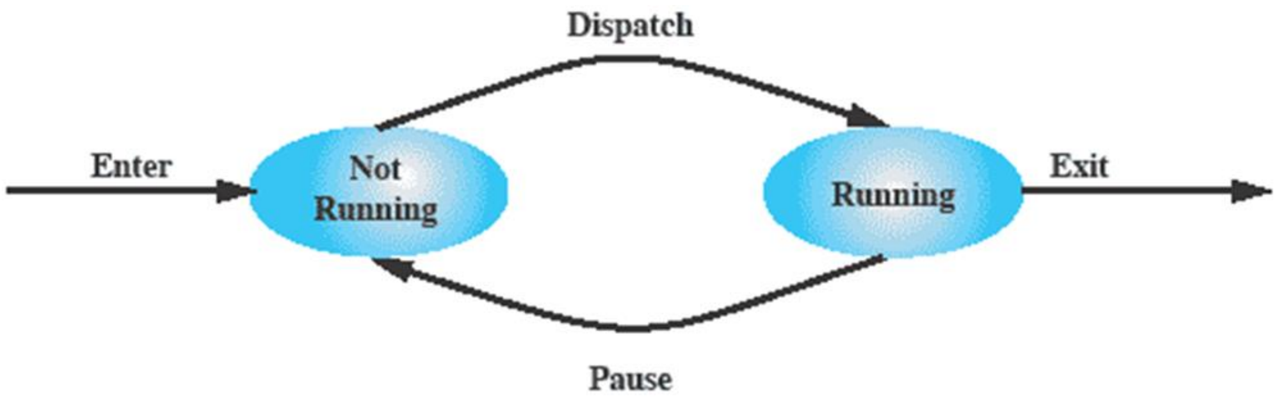
Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded	The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time ("wall clock 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.

Process State

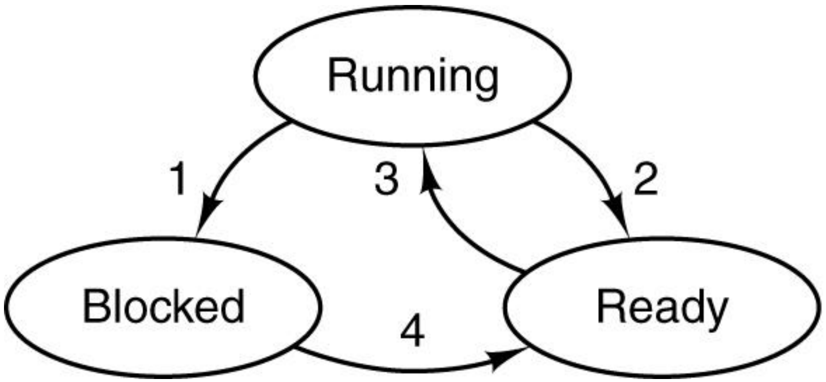
- Depending on the implementation, there can be several possible state models.
- Each state model shows:
  - number of states
  - type of each state
  - when to move from a state to another

Process State: Two-State Model

The simplest model

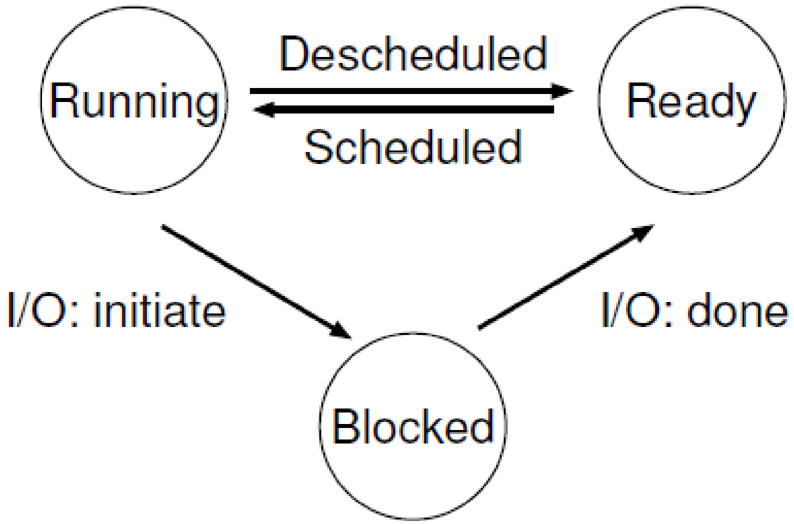


Process State: Three-State Model



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Process State: Three-State Model



Summarizing the previous slide

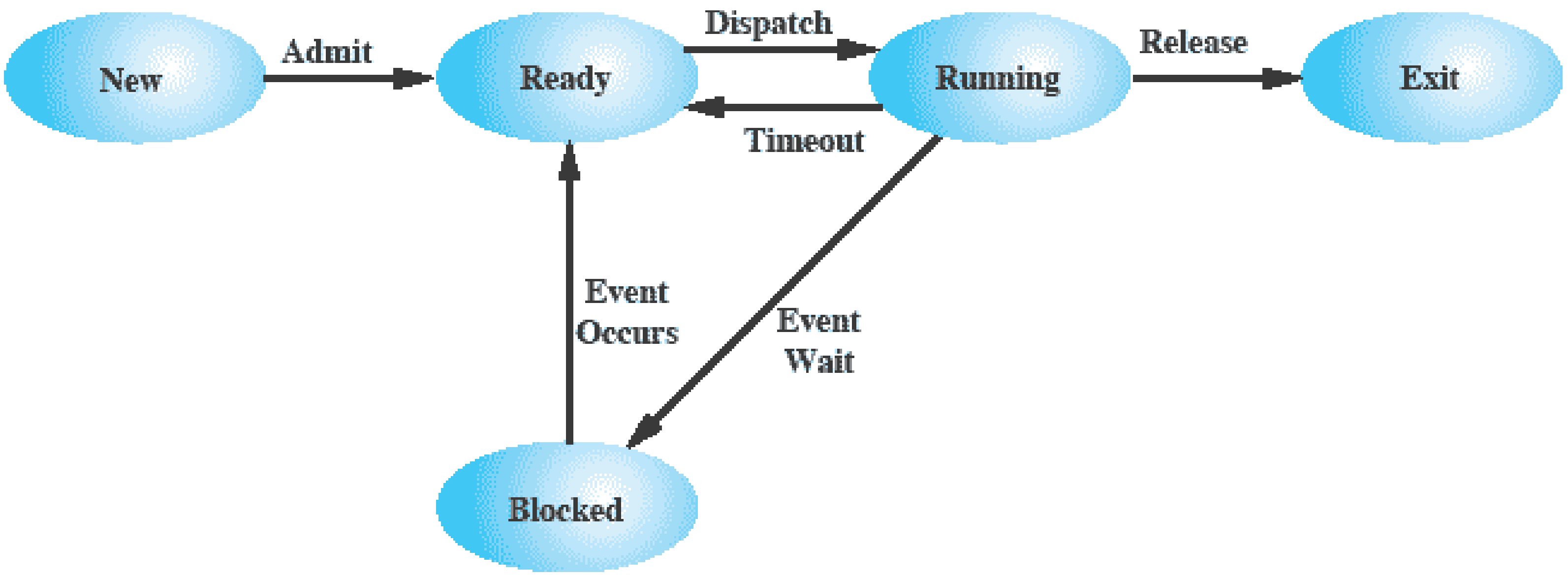
Example:

Time	Process_0	Process_1	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	Process_0 now done
5	–	Running	
6	–	Running	
7	–	Running	
8	–	Running	Process_1 now done

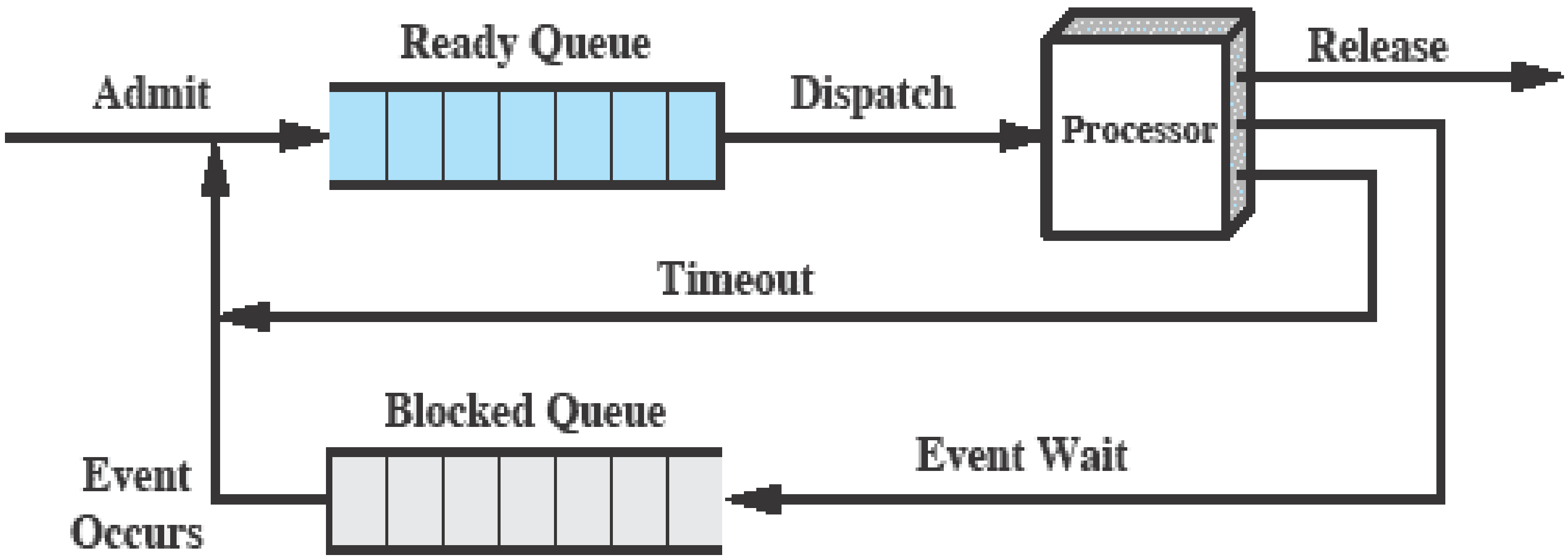
Example:

Time	Process_0	Process_1	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	Process_0 initiates I/O
4	Blocked	Running	Process_0 is blocked,
5	Blocked	Running	so Process_1 runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	Process_1 now done
9	Running	–	
10	Running	–	Process_0 now done

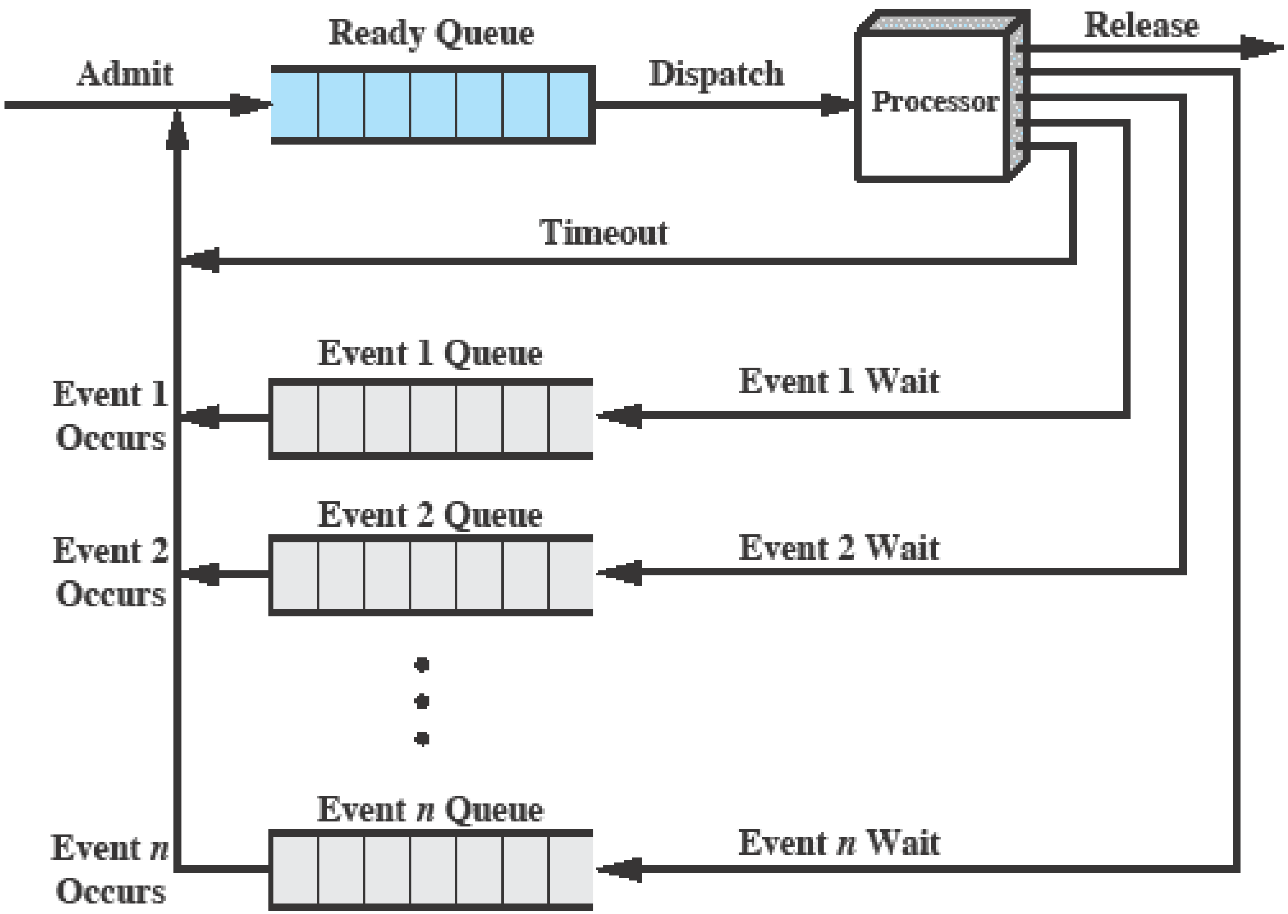
Process State
Five-State Model



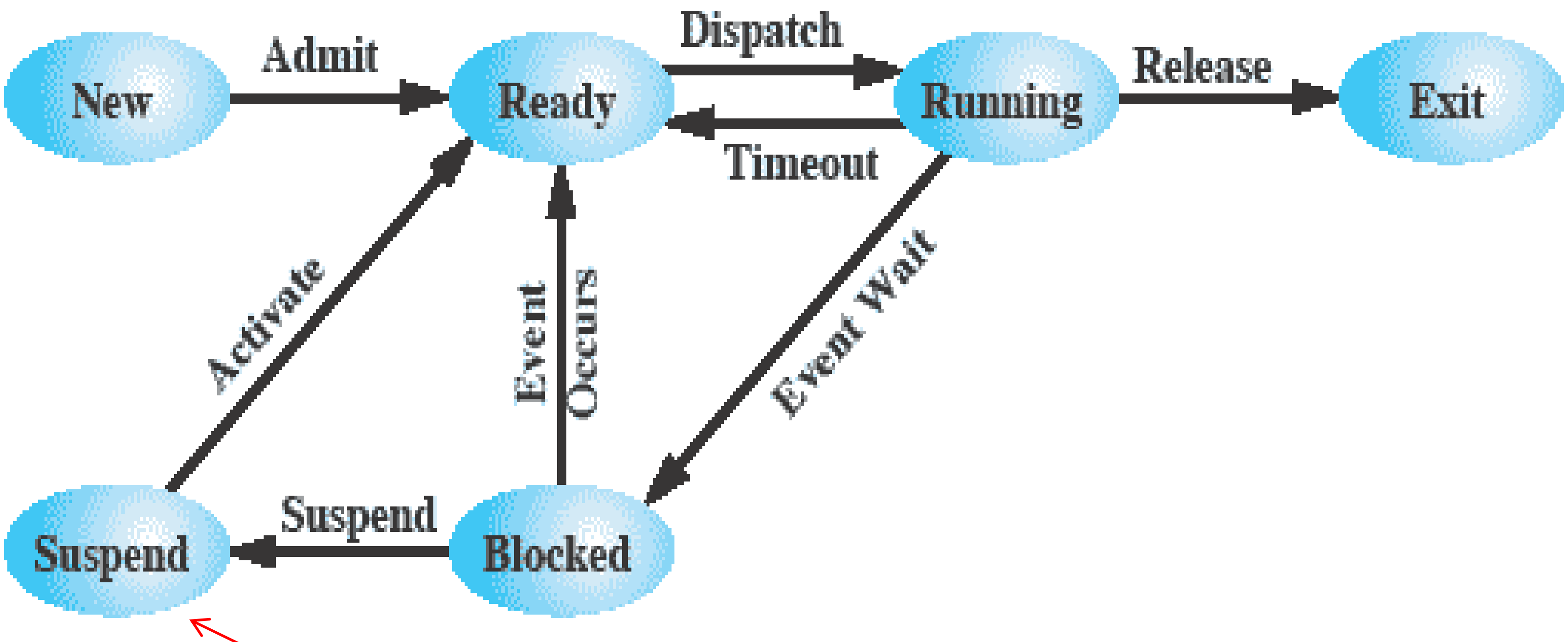
Using Queues to Manage Processes



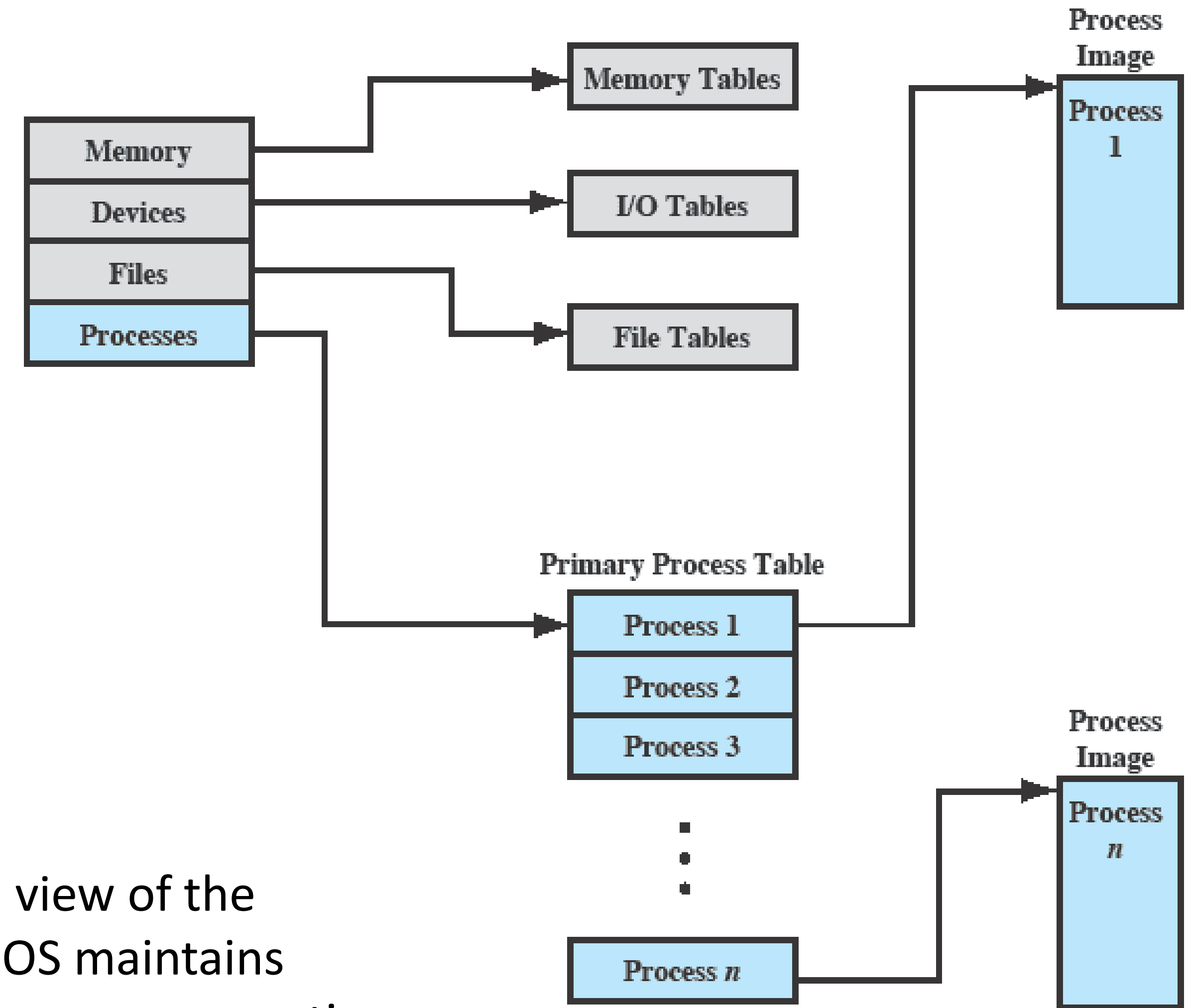
Using Queues to Manage Processes



One Extra State!



Swapped to disk



Conceptual view of the tables that OS maintains in order to manage execution of processes on resources.

A Bit About Interrupts

- **Interrupt means**:
  - Current running process is suspended
  - OS takes control (i.e. the machine moves form user mode to kernel mode)
- **Interrupt occurs** due to many scenarios, for example:
  - Time out of current running process
  - Hardware interrupt from an I/O device
  - Page fault

A Bit About Interrupts

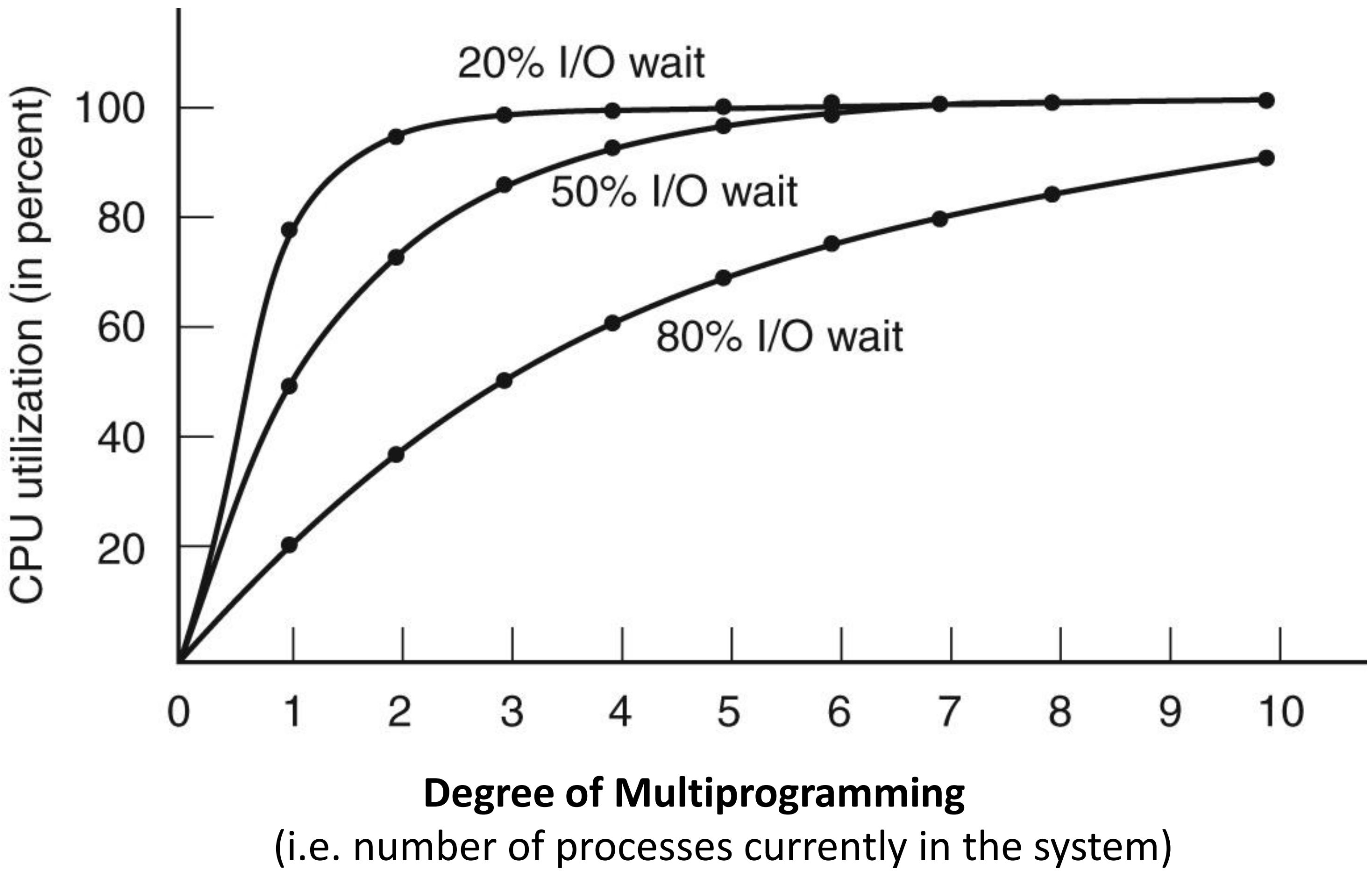
1. Hardware saves program counter, registers, etc of the current process.
2. Hardware loads program counter of the interrupt service routine (ISR).
3. ISR runs.
4. When done, the next process to run is picked.
5. Program counter, registers, etc of the picked process are loaded.
6. The picked process starts running,

Moving from User Mode to Kernel Mode

- Interrupts
- System call
- Exception (e.g. divide by zero, segmentation fault, etc)

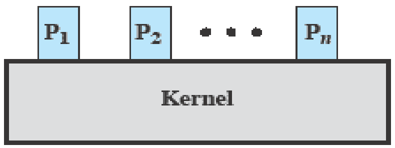
Simple Modeling of Multiprogramming

- A process spends fraction p waiting for I/O
- Assume n processes in memory at once
- The probability that all processes are waiting for I/O at once is p^n
- So -> CPU Utilization = $1 - p^n$

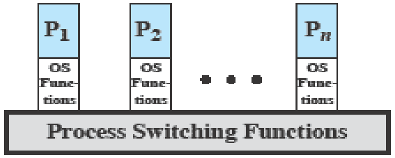


Multiprogramming lets processes use the CPU when it would otherwise become idle.

Executing the OS Itself



(a) Separate kernel



(b) OS functions execute within user processes

(c) OS functions execute as separate processes

Conclusions

- Process is the most central concept in OS
- Process is the main way by which OS virtualizes the CPU
- Even with many cores (i.e., many CPUs), we still need multiprogramming (i.e., time-sharing).