Operating System
Session 3

Hakim Sabzevari University
Dr.Malekzadeh

Process Management

Process

- A process/task is a program in execution (an instance of a running program). A program by itself is not a process. A program becomes a process when the executable file is loaded into memory and run.

- Program is a static list of instructions and data stored on hard disk (a program can also refer to the raw, un-compiled source code) made up of program statements while process is a dynamic entity. Program contains the instructions to be executed by processor.

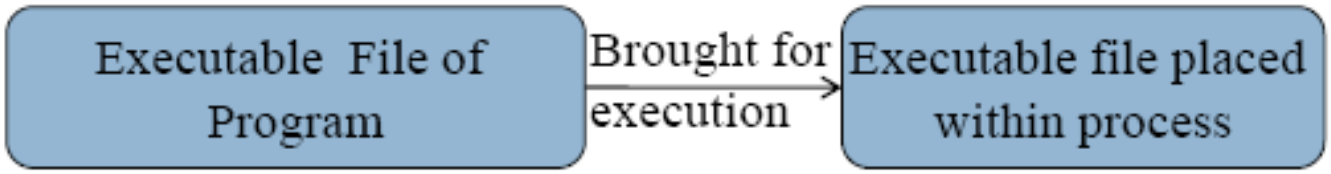
- For example when you select Start -> Program -> Word in Windows, a new process is created to run the executable file with the name Winword.exe.

An active process is a process that is running

Atomic= indivisible

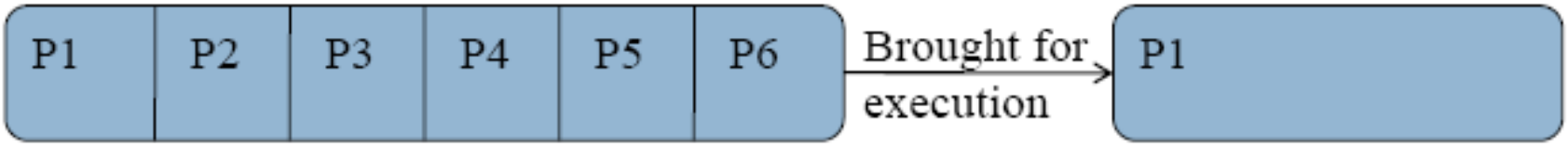
Process cont…

- Process moves between memory and disk during execution.
- Input queue is the collection of programs on the disk that are waiting to be brought into memory to run the program.



Hard Disk

Main Memory



Input Queue

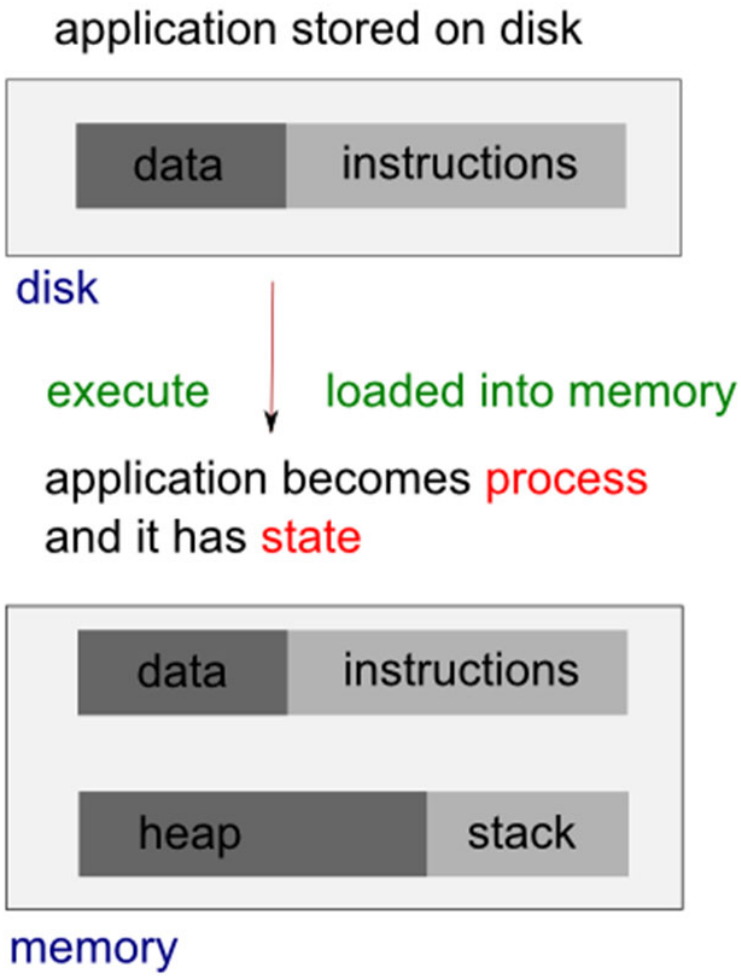
Degree of multiprogramming

- An important concept in multiprogramming is the degree of multiprogramming: number of programs in memory at a time. The degree of multiprogramming describes the maximum number of processes that a system can accommodate efficiently.

- The primary factor affecting the degree of multiprogramming is the amount of memory available to be allocated to executing processes. If the amount of memory is too limited, the degree of multiprogramming will be limited because fewer processes will fit in memory.

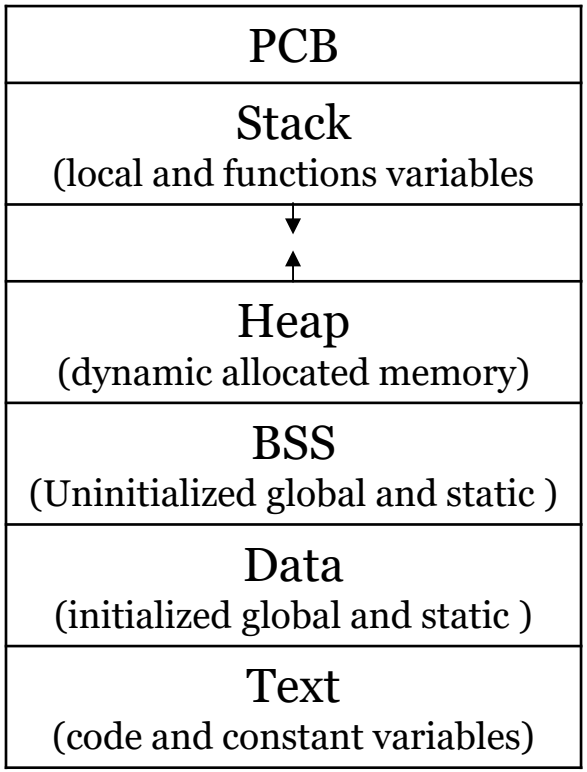
Process contents

- Program: code + data
- Process: code + data + more data + PCB



Process contents cont…

- A process generally includes six parts:



When you open some EXE file in Notepad, you can see that it includes a lot of "Gibberish" language, something that is not readable to human. It is the machine code, the computer representation of the program instructions. This includes all user defined as well as system functions.

Process contents cont…

- Text/code: contains constant variables and also the machine code of the compiled program initially stored on disk in an executable file.

- Data: it is also called the initialized data segment or simply the data segment and often denoted as .data. A data segment is a portion of virtual address space of a program, which contains the initialized global and static variables.

Process contents cont…

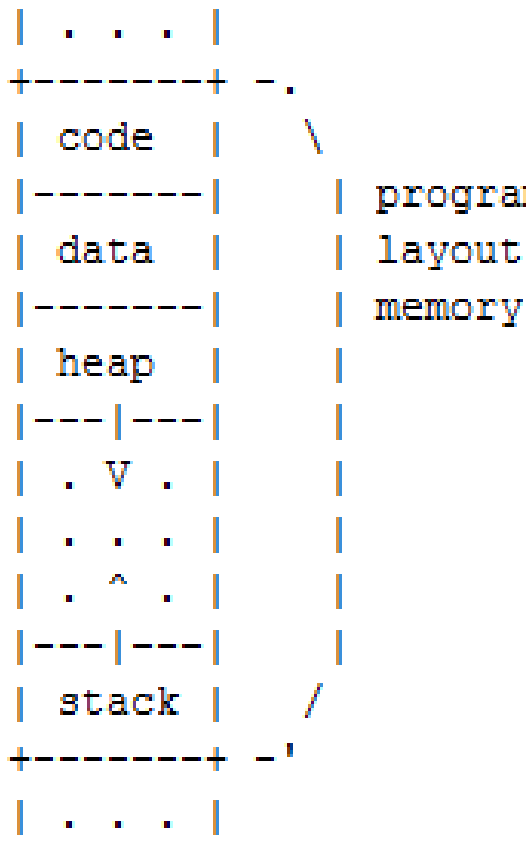
- Block Storage Start/Block Started by Symbol (BSS) segment starts at the end of the data segment and contains all uninitialized global and static variables.

- Example1: a variable declared **static int i;** would be contained in the BSS segment.
- Example2: a global variable declared **int j;** would be contained in the BSS segment.

Process contents cont…

- Heap: is memory that is dynamically allocated during process run time. This area is managed by malloc, realloc, and free.
- Stack: is used for local variables when they are declared, function parameter, function return values. The size of this region is contained within the program.
- Process Control Block (PCB)

Example:

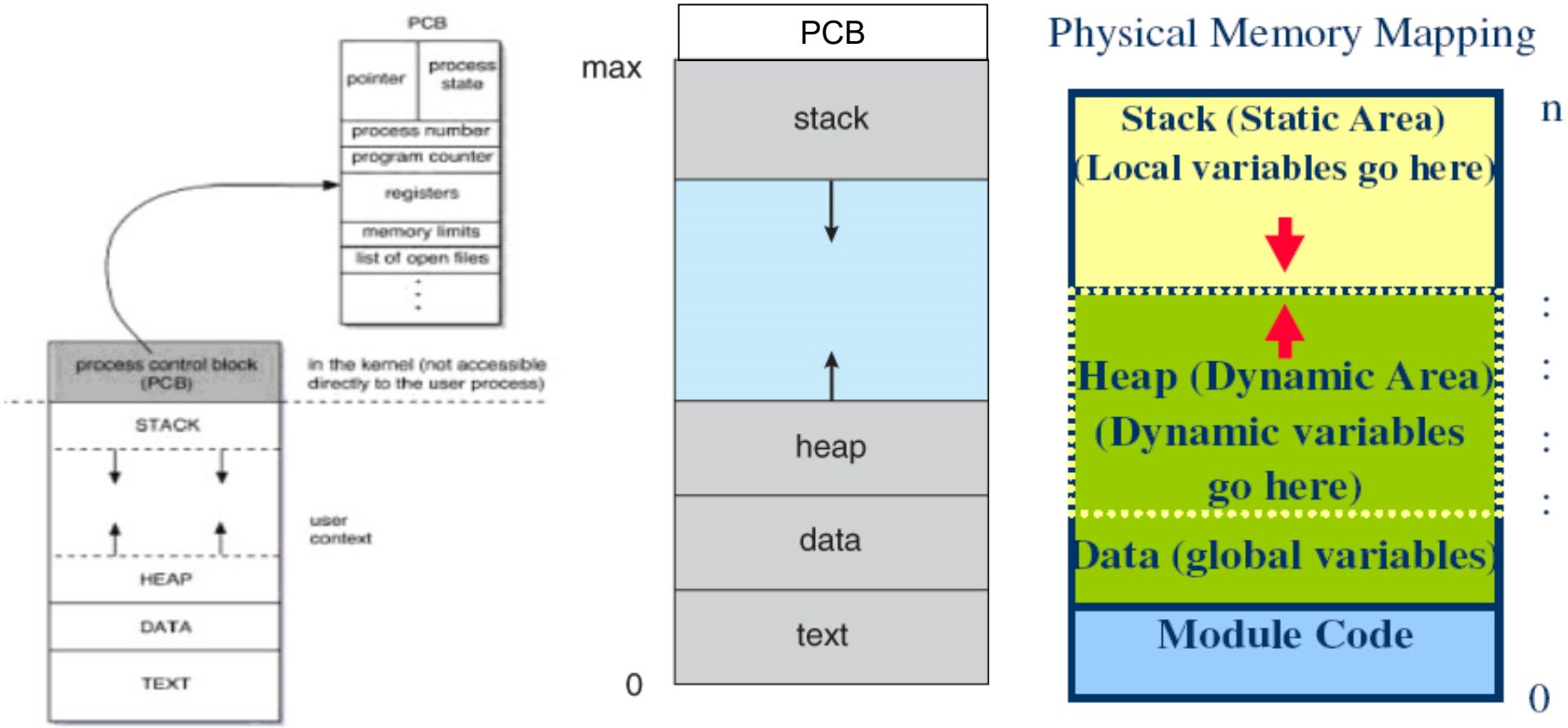



```
int g = 1;          /* g placed in data */

int main(){
   int a;                    /* a on stack */
    ...
   int *b = malloc(...);  /* b in heap */
    ...
   a = ...;                        /* code */
}
```

Process contents cont…

- The stack and the heap start at opposite ends of the process's free space and grow towards each other. If they meet, then either a stack overflow error will occur, or else a call to new or malloc will fail due to insufficient memory available.



Stack vs. Heap

Stack vs. Heap cont…

- Stack is a special region of memory that stores temporary local variables created by each function (including the main() function).
- Every time a function declares a new variable, it is "pushed" onto the stack. Then every time a function exits, all of the variables of that function, are "popped" (free) and that region of memory becomes available for other stack variables.
- The advantage of using the stack to store variables:
  - The memory is managed for you. You don't have to allocate memory by hand, or free it once you don't need it any more.
  - Because the CPU organizes stack memory so efficiently, reading from and writing to stack variables is very fast.

Stack vs. Heap cont…

- The heap is a region of your computer's memory that is not managed automatically for you by the CPU. To allocate memory on the heap, you must use malloc() or calloc() or new(), which are built-in C functions.

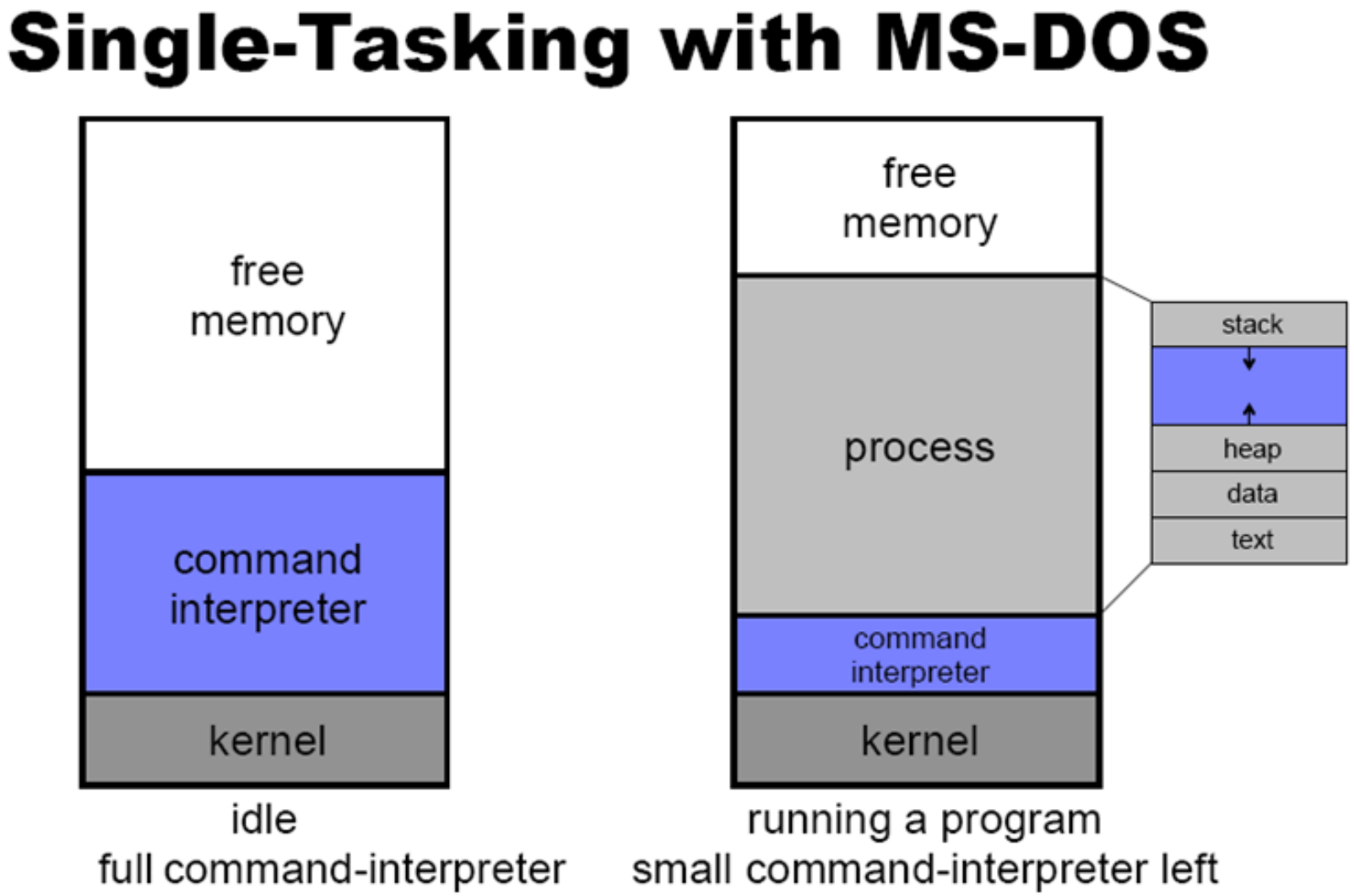
- Once you have allocated memory on the heap, you are responsible for using free() to **deallocate** that memory once you don't need it any more. If you fail to do this, your program will have what is known as a memory leak. That is, memory on the heap will still be set aside (even after the program terminated) and won't be available to other processes.

- The heap segment provides more stable storage of data for a program; memory allocated in the heap remains in existence until you free it up (so for example you can keep it for the entire duration of a program).

Process in single-tasking OS (e.g. MS-DOS)

- In single-tasking OS, only one process can be in memory at a time, MS-DOS is the best known example.
- A command interpreter is loaded upon boot. When a process is launched in DOS, the command interpreter first unloads to hard disk as much of itself as it can to free up memory, then loads the process and transfers control to it.
- The reason to this is memory back then was very limited and expensive.

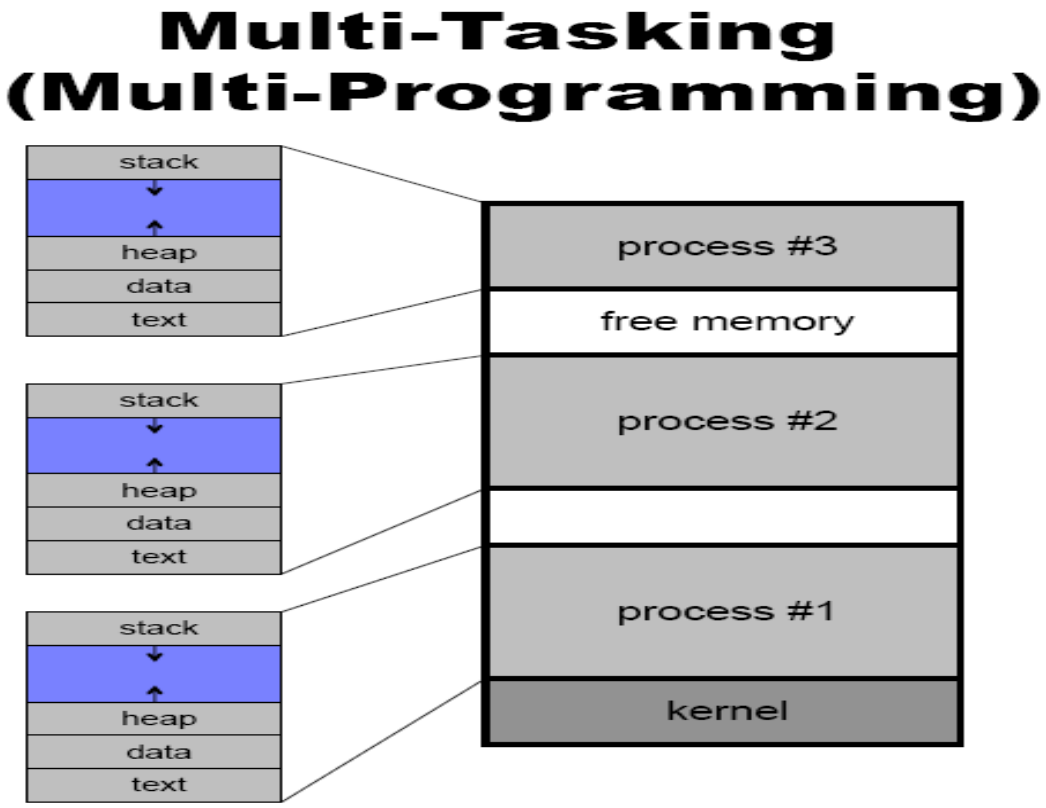
Process in single-tasking OS cont…



Process in multi-tasking OS (Linux)

- In a multi-tasking OS, multiple processes can co-exist in memory. Therefore, each process has a name (an integer) which is called a process ID (PID). The PID namespace is global to the system. Only one process at a time has a particular PID. A PPID refers to the parent PID.

In this system, the command interpreter remains completely resident when executing a process.



Process in multi-tasking OS cont…

- Process control commands include:
  - create (fork)
  - execute (exec)
  - wait
  - end (exit)
  - abort (kill)

Process in multi-tasking OS cont…

- Two ways to create a process:
  - Build one from scratch: like Windows (CreateProcess(…))
    - Load specified code and data into memory
    - Create empty call stack
    - Create and initialize PCB (make it look like a context switch)
    - Add process to ready list
  - Clone an existing one: like UNIX (fork())
    - Stop current process and save its state
    - Copy code, data, stack and PCB
    - Add new Process PCB to ready list

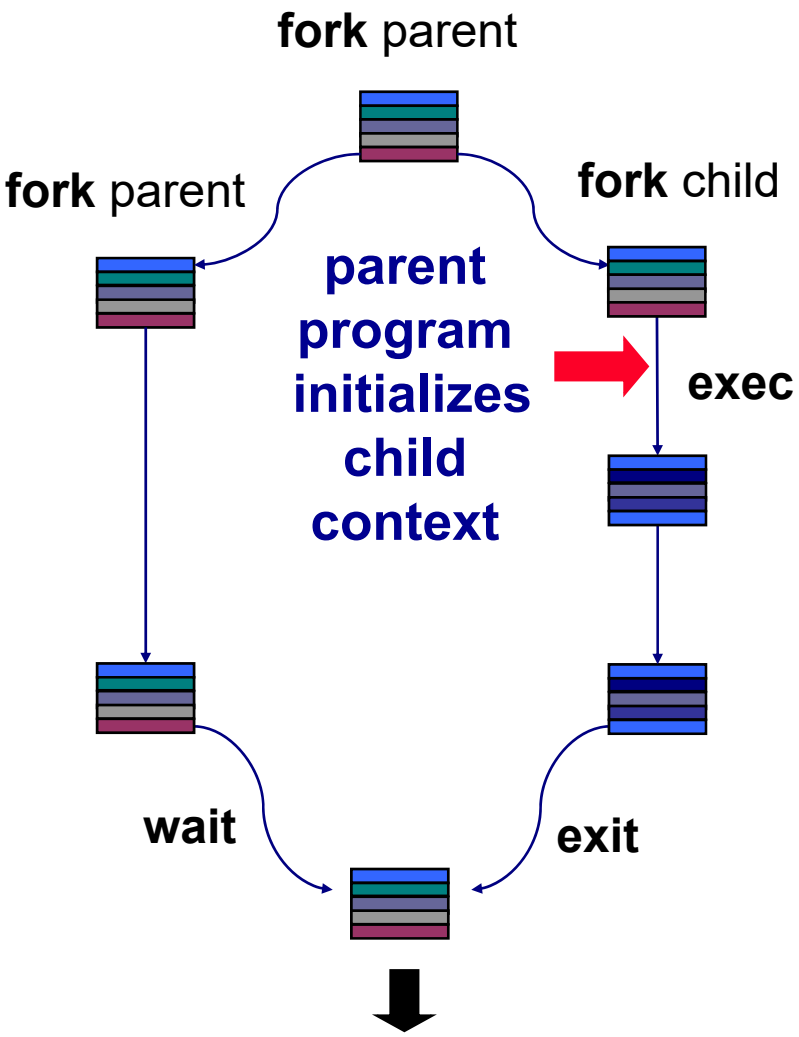
Process in multi-tasking OS cont…

- To start a new process, UNIX uses the fork command to create the process and exec command to execute it.

- A "fork" system call creates a second process which is an exact duplicate (clone) of the original process (command line interpreter). The original creator process is called the parent, and the created cloned process is called the child, with its own unique process PID and parent PPID.

- **After the *fork*, the original process and the copy go their separate ways**. The child process then executes an "exec" system call, which replaces its code with that of the desired process and transfer control to it.

Process in multi-tasking OS cont…



int pid = fork();

Create a new process that is a clone of its parent.

exec*("program" *[, argvp, envp]*);

Overlay the calling process virtual memory with a new program, and transfer control to it.

exit(status);

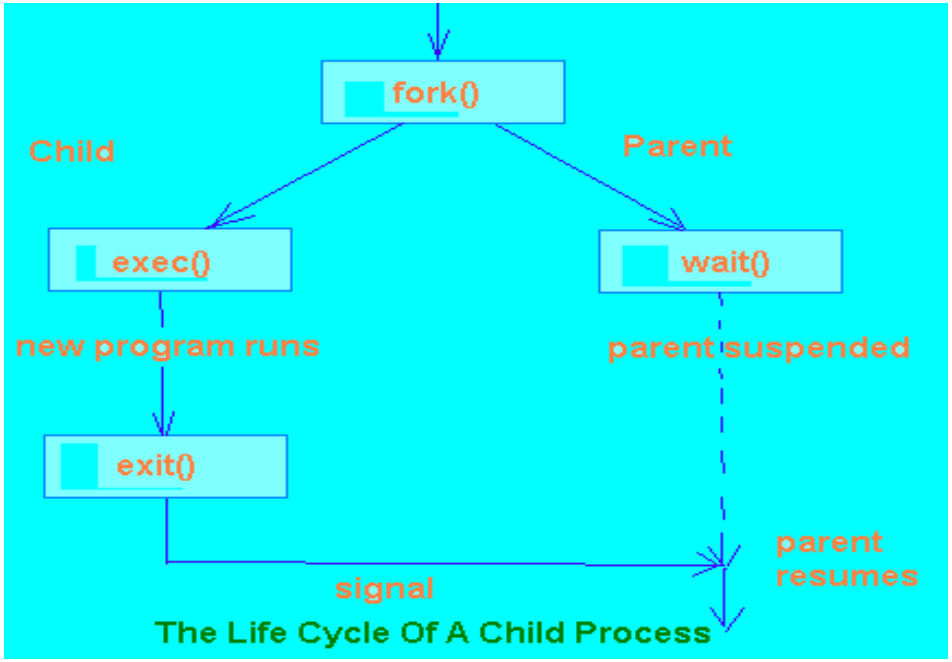
Exit with status, destroying the process.

int pid = wait*(&status);

Wait for exit (or other status change) of a child.

Process in multi-tasking OS cont…

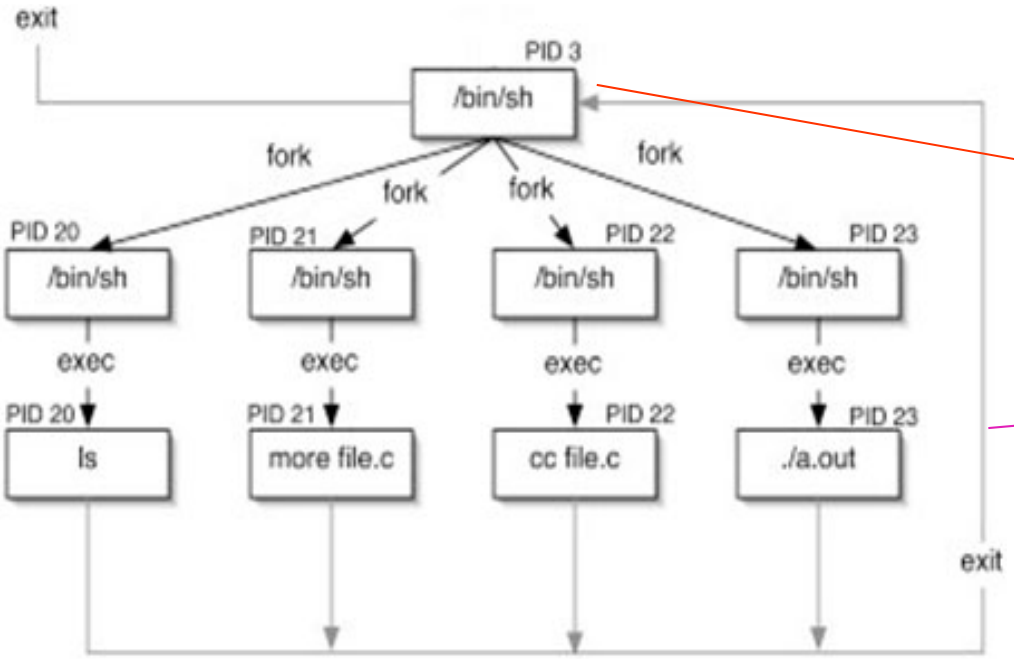
- What the parent process does while the child process runs?
- It either:
  - continues the job that it was doing
  - Or waits for the child process to terminate (exit). In this case, a parent process will just execute a **wait()** system call. This call will suspend the parent process until any of its child processes terminates, at which time the **wait()** call returns and the parent process can continue.



Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

But how is the first process made?

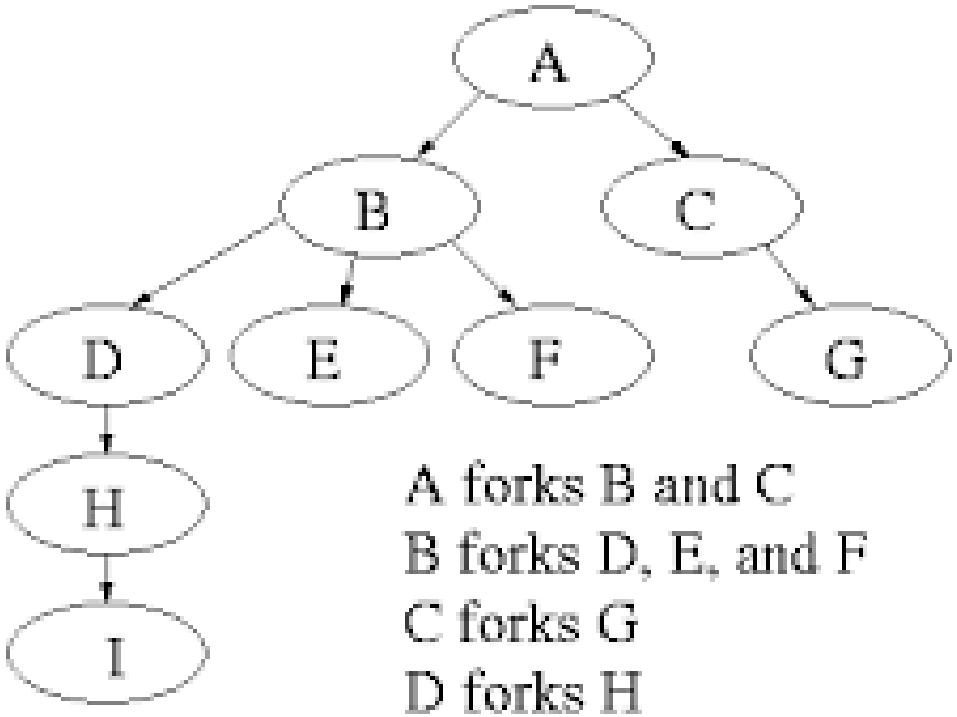


Login runs **shell** after user authenticates.

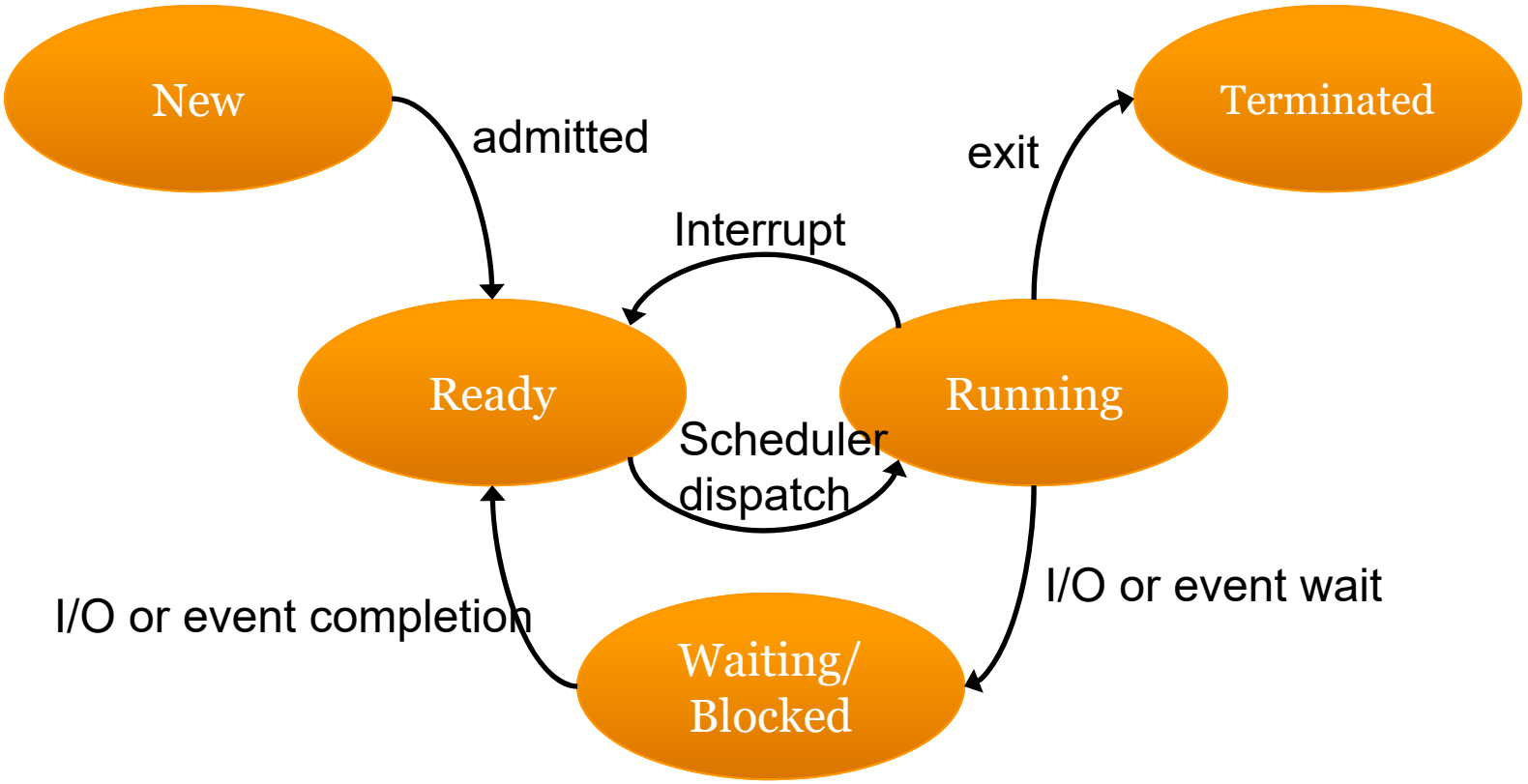
Shell runs user commands as child processes.

Process hierarchy

- Operating system needs some way to create and kill processes. When a process is created, it creates another process which in turn crates some more processes and so on, thus forming process hierarchy or process tree.



Process life cycle



Process states

- The lifetime of a process can be divided into several states, each indicates what the process is currently doing. At any point in time, a process can be in one of the following five states:
  - New/Start: the process is in the stage of being created. New processes are created by existing processes:
    - creator is called the parent
    - created process is called the child
  - Ready: the process is in a queue called ready queue waiting for the CPU to be assigned to it.

Process states cont…

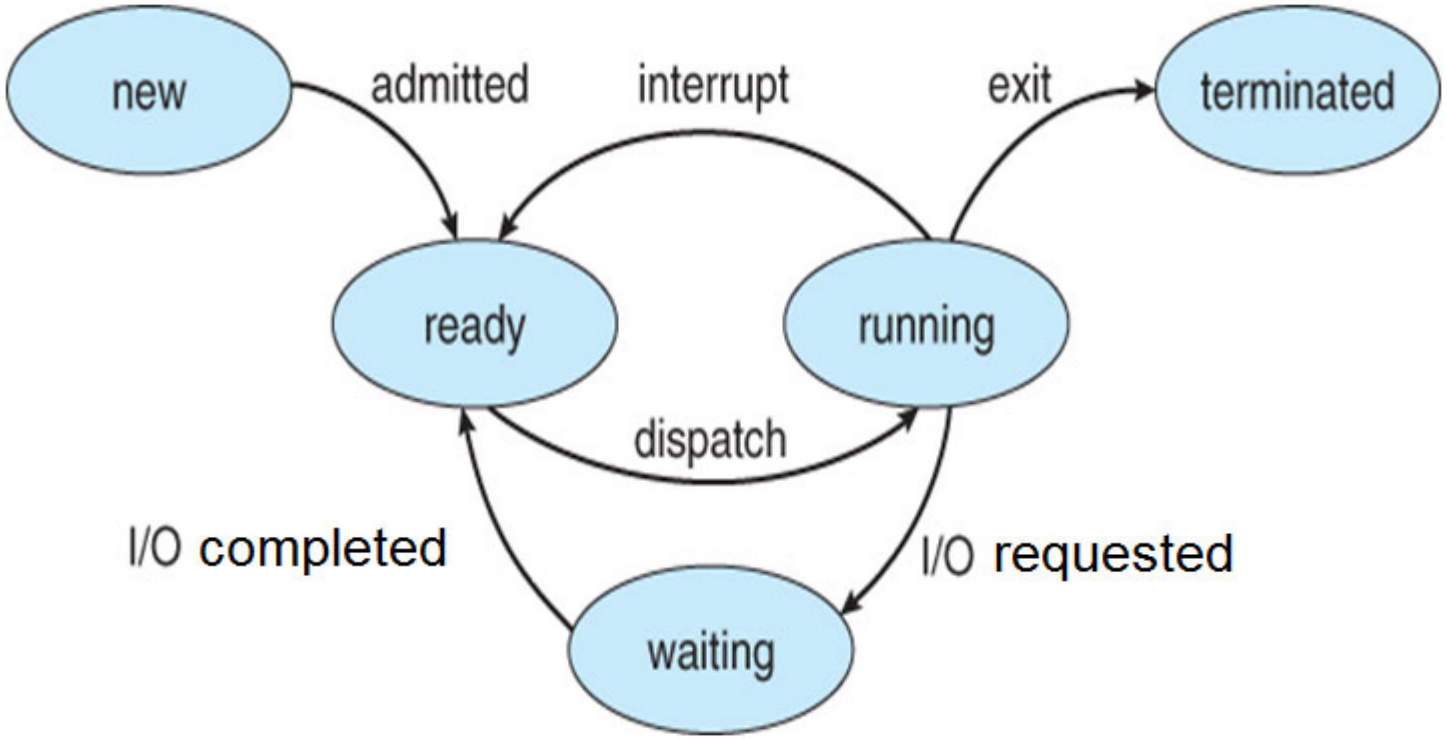
- Running: the **CPU has been assigned** to the Process and it is being executed. A program called scheduler which is a part of operating system, pick-ups one ready process for execution by passing a control to it.
- Terminated/Halt: after a process completed, the process will be automatically terminated by the CPU and it de-allocates the memory of the process.

Process states cont…

- Waiting/Blocked: in this state, the process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example a process is waiting for some input and output operations or a message from (or the completion of) another process. In this state, since the process cannot make progress until the event happens, the process is kept in a queue called wait queue. When the event happened then the process will again be on ready state.

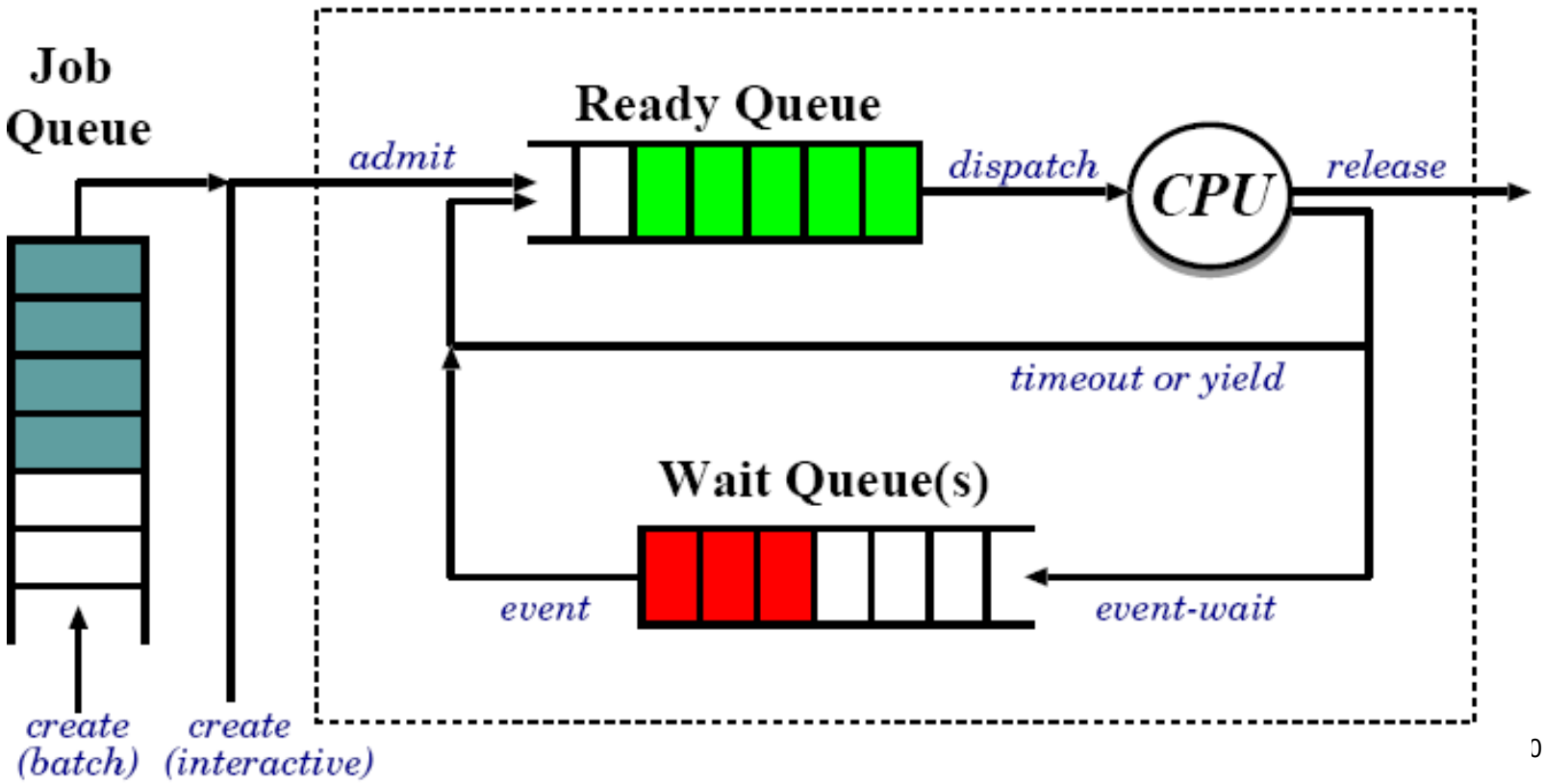
Process states cont…

- As a process executes, it moves from state to state:
  - Dispatching (توزيع): Move from ready state to running state (context switching)
  - Time-out: Move from running state to ready state
  - Wake-up: Move from waiting state to ready state



Queuing diagram of process scheduling

- job/ input queue: a set of all programs on disk
- Ready Queue: on memory
- Wait queue/ Blocked Queue/ Device queue/ I/O queue



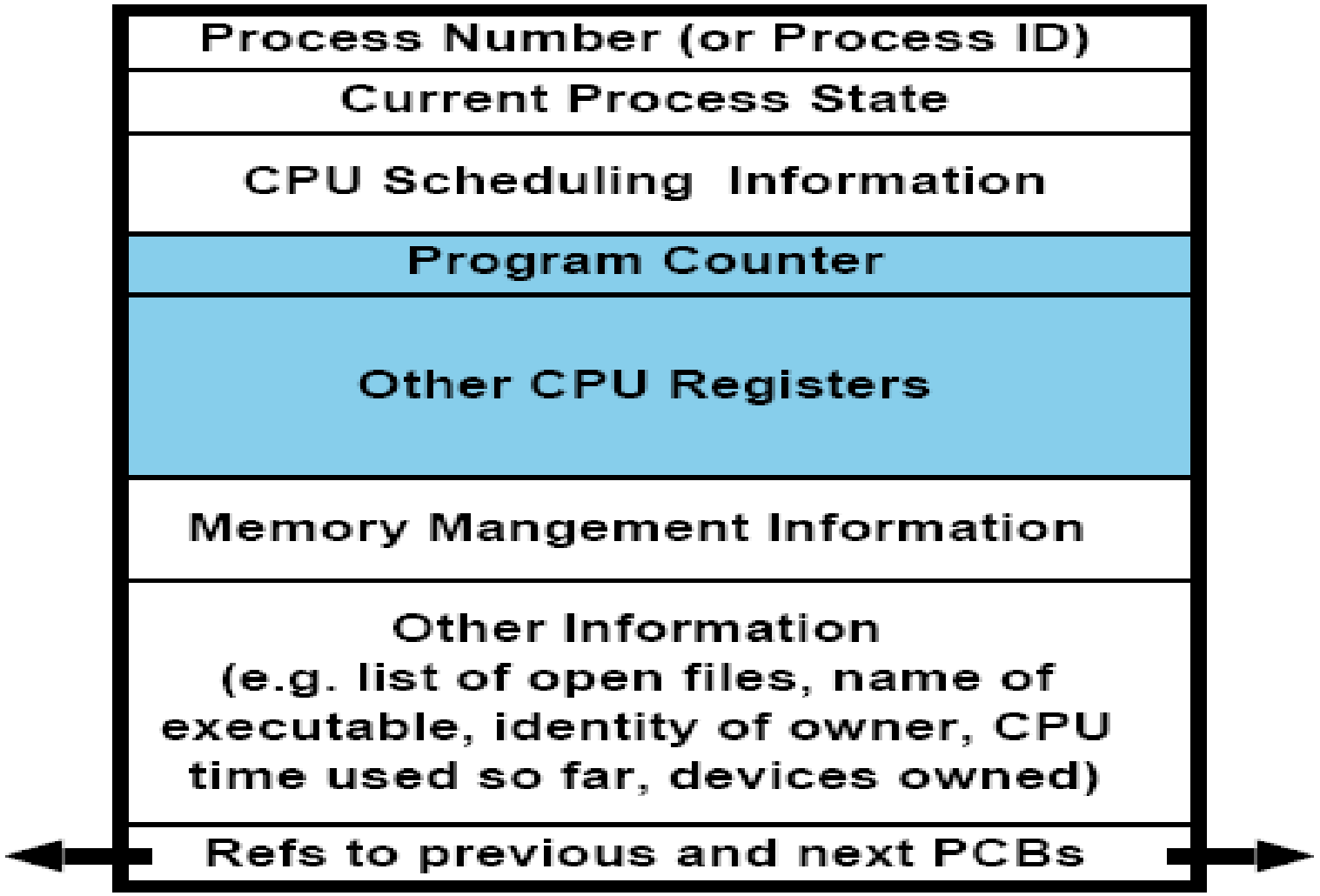
PCB

- The OS must manage a large amount of data for each process. Usually that data is stored in a data structure called a process control block (PCB). The information in PCB include:
  - process ID (PID)
  - Process state
  - CPU-Scheduling information: such as the priority given to the process by the OS.
  - Memory-Management information: this information may include the value of base and limit register. This information is useful for de-allocating the memory when the process terminates. E.g. page tables or segment tables.
  - Accounting information: amount of resources used so far like actual CPU time used in executing a process.
  - I/O Status information: list of I/O devices allocated to the process, open files, etc.

PCB cont…

- CPU registers and Program Counter: these need to be saved and restored when moving processes in and out of the CPU. Registers include accumulator, general purpose registers, index registers, PC, etc. Keep in mind that there is only one CPU and therefore only one set of CPU registers. These registers contain the values for the currently executing process (the one in the running state). Each time a process is moved to the running state, the register values for the currently running process are stored into its PCB (so that the process be allowed to continue correctly afterwards), and the register values of the new running state are loaded into the CPU.

PCB cont…


Process Number (or Process ID)
Current Process State
CPU Scheduling Information
Program Counter
Other CPU Registers
Memory Mangement Information
Other Information
(e.g. list of open files, name of executable, identity of owner, CPU time used so far, devices owned)
Refs to previous and next PCBs

Process management

- To accomplish its tasks, each process needs certain resources including CPU time, memory, files, and I/O devices.

- Process Management include:
  - Inter-process communication (IPC)
  - Management of PCBs
  - Process scheduling
    - Process dispatching
    - Process switching/ Context switching

Management of PCBs

- The OS has a PCB table. For each process, there is one entry in the table. A new PCB is created when a process is first created (the new state) and is kept around until the process terminates.

- When a process is created:
  - OS allocates a PCB for it
  - OS initializes PCB
  - OS puts PCB on the correct queue

- When a process is terminated:
  - PCB may be kept for a while (to receive signals, etc.)
  - eventually, OS de-allocates the PCB

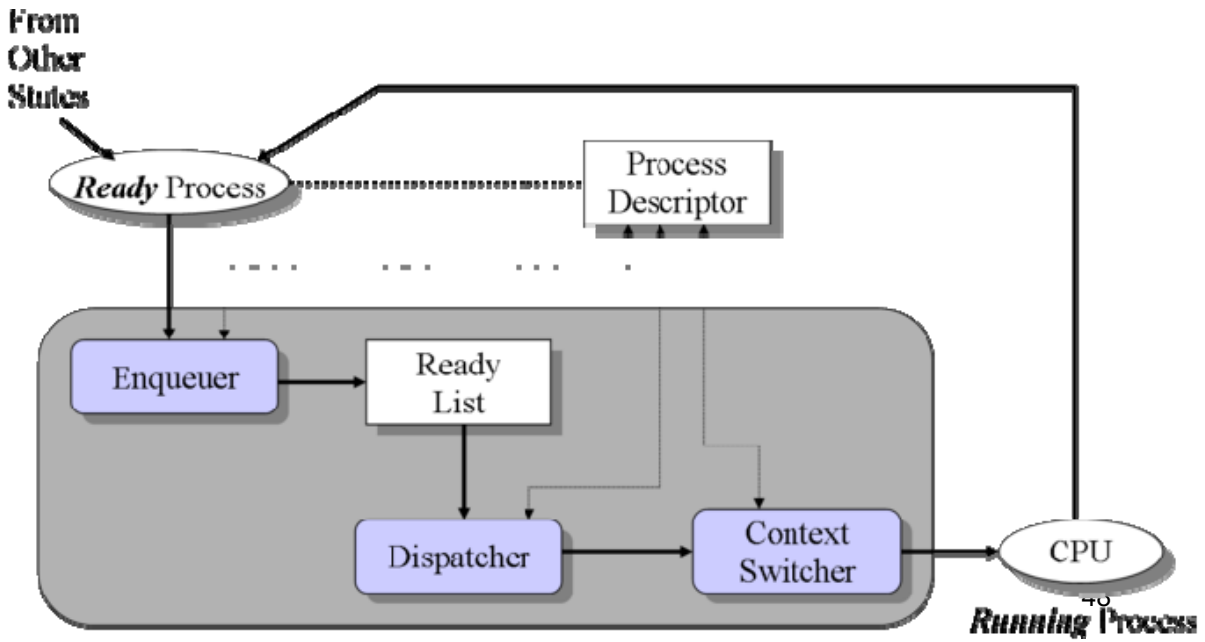
Process scheduling

- The process scheduler is a part of OS that performs three functions:
  - Selects the next process to which the CPU will be allocated
  - Deallocates the CPU from the process currently executing
  - Allocate the CPU to the newly selected process

- To perform these three functions, there are three components:

Process scheduling cont…

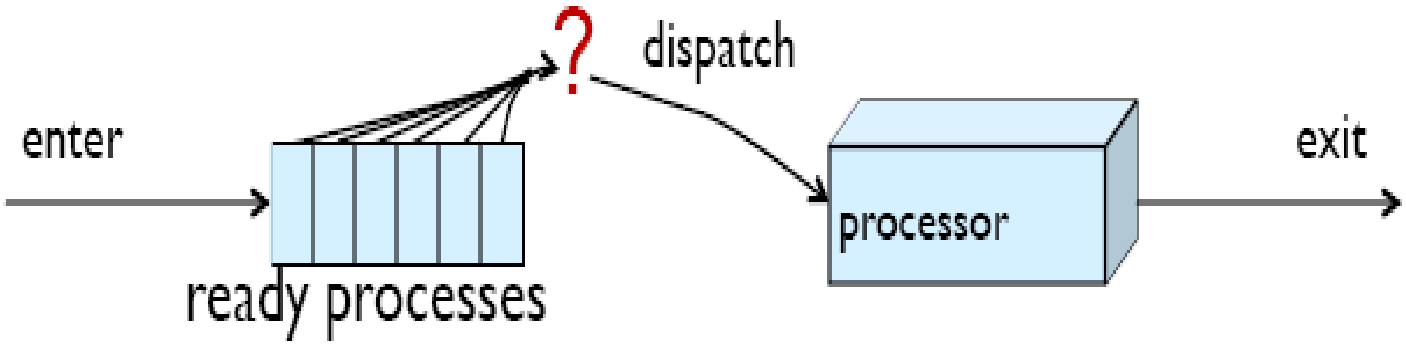
- A scheduler has three components to perform Sequence of Scheduling Mechanism:
  1. Enqueuer: is a component of scheduler that inserts the process that requests CPU service into the ready queue.
  2. Context-switcher: saves the content of the current process in its process descriptor (PCB), deallocates the CPU from that process, and invokes dispatcher.
  3. Dispatcher: is invoked after the current process has been removed from the CPU. It selects the next process from the ready queue and performs another context switch. This time, context switcher loads its context, and then allocates the CPU to this newly selected process.
- Data Structures:
  - Process Descriptor
  - Ready List



Process dispatching

- The dispatcher module is an OS program that gives control of the CPU to the next process.

- The term **dispatch latency** is defined as the time it takes for the dispatcher to stop one process and run another process. Dispatch latency is optimal when it is minimized.



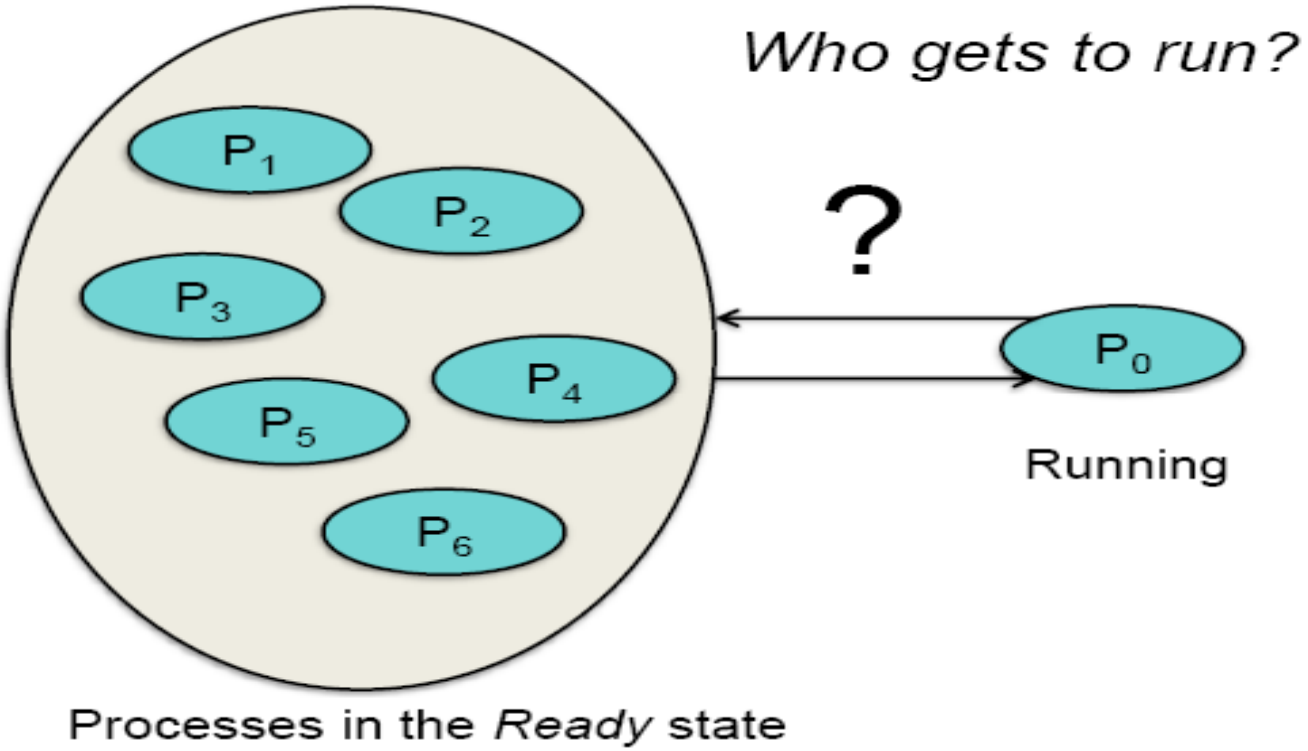
Process switching/ Context switching

- Since for a single-CPU system, only one process is executed by the CPU at a time, multiple processes may be loaded in memory but only one is in the Running state and all others are, e.g., in the Ready state. Therefore, the CPU performs process switching between the processes to run all them.

- (remember that the act of assigning a CPU from one task to another one is called context switch).

Process scheduler

- The process scheduler is responsible for deciding:
  - whether the currently running process should continue running
  - if not, which process should run next



Process scheduler cont…

Policies: what to do
Mechanisms: how to do

- Process Scheduler (زمانبند) is a part of OS kernel that decides which process to run. The process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.

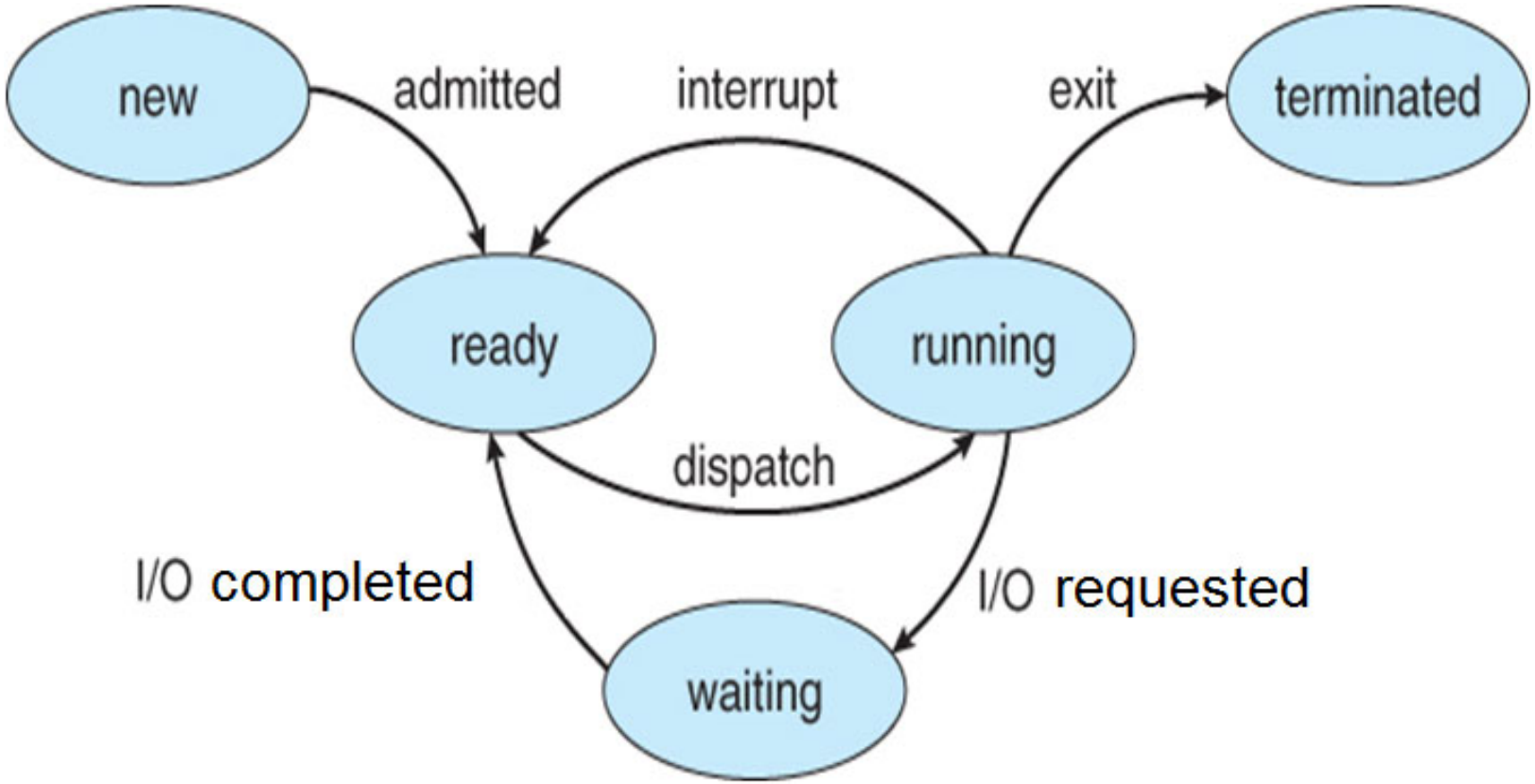
- Scheduling policy:
  - When to remove a process from CPU
  - What is the next process to run.

- Scheduling mechanisms/algorithms:
  - How to remove the process from CPU
  - How to select the next process

- Scheduling criteria: metrics that are commonly used to evaluate the performance of a scheduling algorithm.

A Five-state process model

- The five basic states of a process are:



Suspended process

- So far, all processes had to be in main memory. Even with virtual memory, keeping too many processes in main memory can decrease systems' performance.

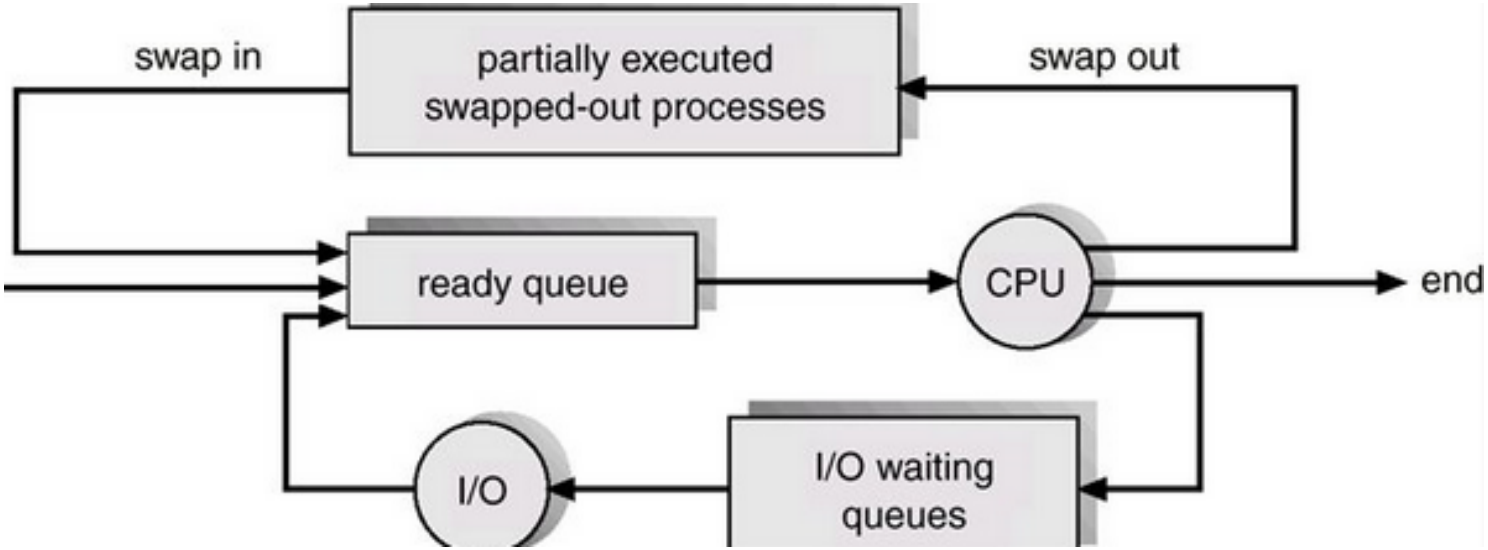
- For example, since CPU is faster than I/O, all processes can be waiting for I/O together. Thus even with multiprogramming, CPU can be idle most of the time.

- Solutions for this problem are:
  - Expanding memory; which is expensive
  - Swapping

Suspended process cont…

- In some systems with virtual memory techniques it may be sometimes necessary and advantageous to reduce the degree of concurrent processing. That is, to remove some of the processes from the main memory, later can be brought from secondary memory and continued where it left off for better memory management.

- A process can be suspended by the OS. A process is said to be suspended when it is brought from main memory to the secondary memory (disk).



Suspended process cont…

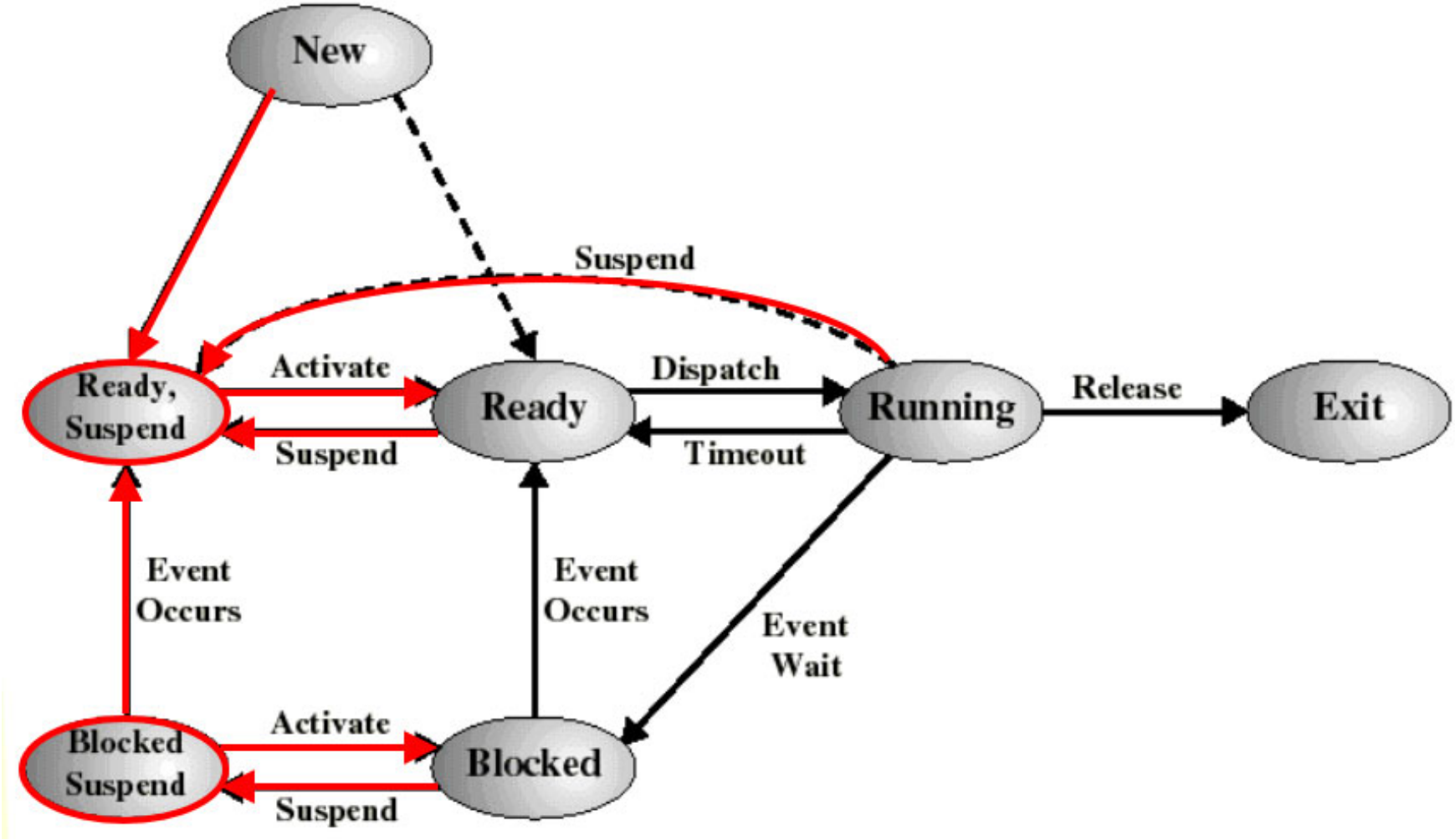
- Scheduler may decide to 'swap out' process when there are reasons for process suspension such as:
  - A process is waiting (need I/O). Since a CPU is faster than I/O, so it can swap all the processes that are waiting for I/O to disk to free up more memory and brings in a process that is ready to execute.
  - A process which has not been active for some time
  - A process which has a low priority
  - A process which is taking up a large amount of memory
  - In timesharing OS

Suspended process cont…

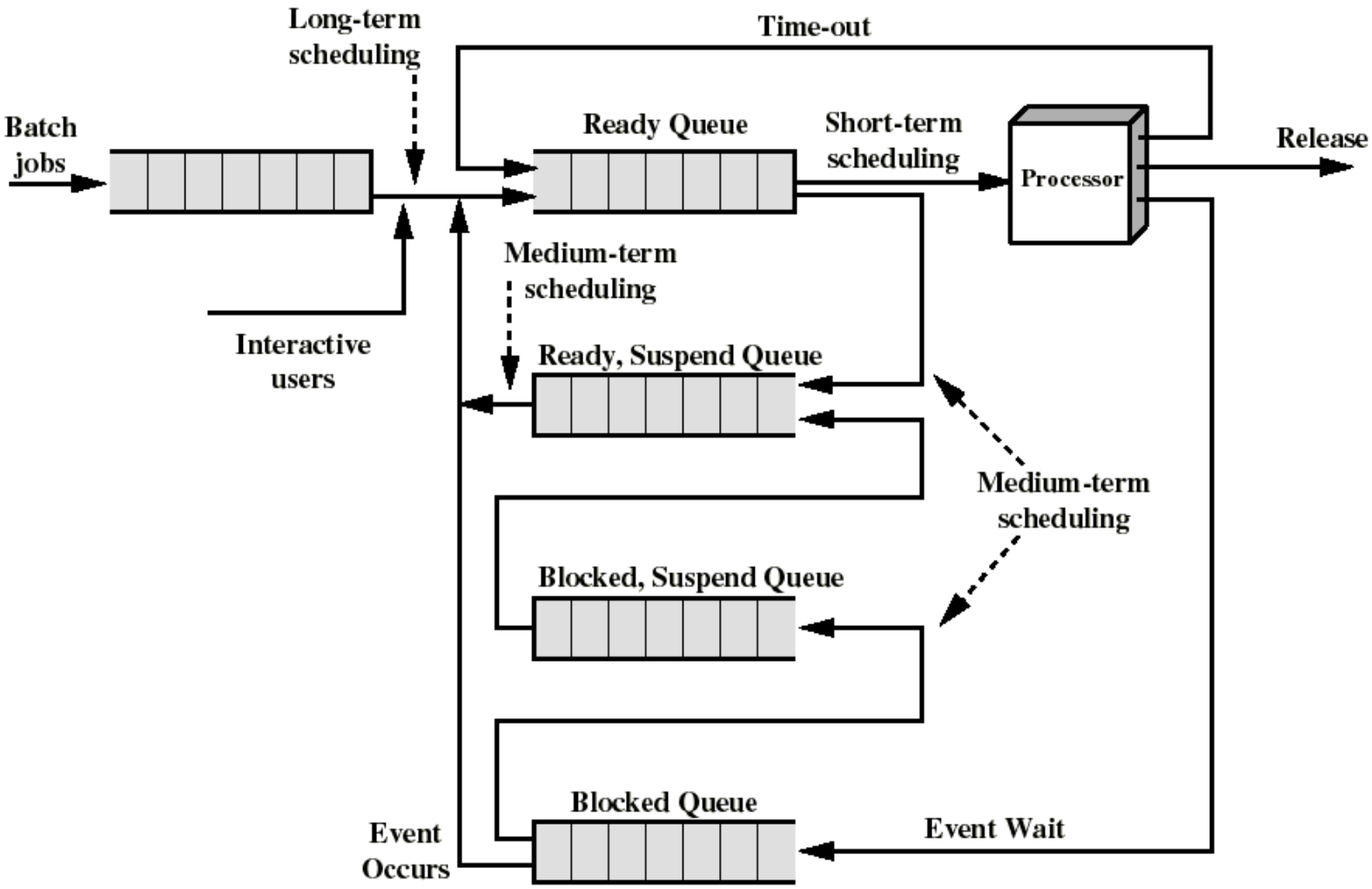
- This will provide two new suspend states:
  - Ready suspend: the ready state process becomes ready suspend state when swapped out to disk
  - Blocked suspend: the blocked state process becomes blocked suspend state when swapped to disk

- In this case there is a Seven-state process model:

A Seven-state process model

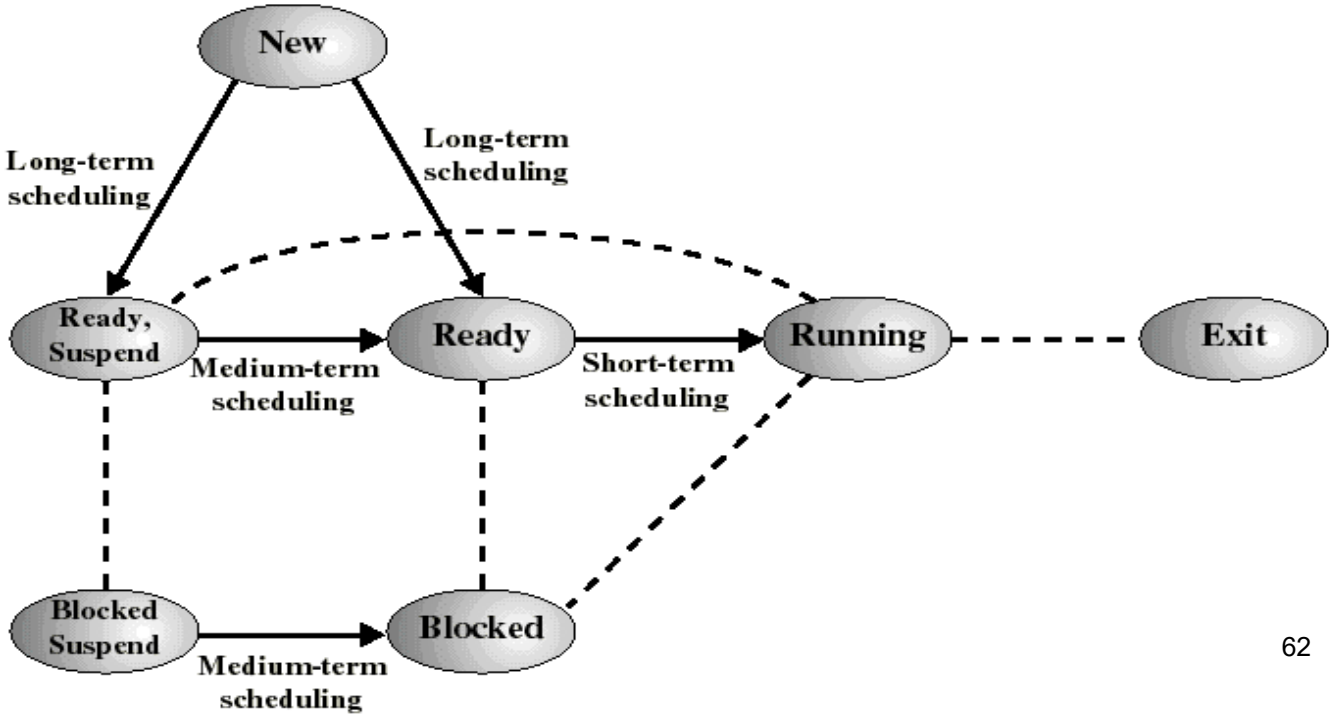


Queuing Diagram for Scheduling



Process schedulers cont…

- Three common schedulers are:
  - Long-term: which process to add to the ready queue (job scheduler). It controls the degree of multiprogramming in multitasking systems.
  - Short-term: which ready process to execute next (CPU scheduler/ Dispatcher).



Process schedulers cont…

- Medium-term/swapping scheduler (میان مدت) decides which process to swap out/in memory. This also controls degree of multiprogramming.

  The medium term scheduler improves multiprogramming by allowing multiple processes to reside in memory and swapping out processes to **make space for other process**.

  Thus, the medium term scheduler is required when we have **limited memory**. If we are running multiple programs and we have very large memory (larger than the size of all processes plus addition space for other requirements) then medium term scheduler is not needed.

Process schedulers cont…

- When the OS needs to select another process to execute, the scheduling policy decides which of the processes to assign to the CPU. Scheduling policy effectively determines the order of the ready queue so that the next process to run is at the **head of the queue**.

- In general, scheduling policies can be:
  - Preemptive
  - non-preemptive/ Cooperative

Preemptive scheduler

- If another process with a higher priority suddenly becomes able to use the CPU, the kernel will immediately context-switch to the higher priority process. We call this preemption which means the higher-priority process preempted the lower-priority process.

- Pros:
  - No limitation on the choice of scheduling algorithm
  - Makes Time sharing possible

- Cons:
  - Additional overheads e.g. more frequent context switching.

non-preemptive scheduler

- It is a run-to-completion schedulers. Once the CPU has been given to a process, it cannot be preempted. The scheduler cannot take the CPU away from a process and a process holds the CPU until it is **willing** to give it up.

- Pros:
  - Decreases turnaround time
  - Does not require special HW (e.g., timer)

- Cons:
  - Limited choice of scheduling algorithm

Question

- Which type of scheduling (preemptive / non-preemptive) occurs in the following settings?
  - Restaurant
  - Hospital emergency room
  - Professor's office hours

Scheduling Criteria (معیار های زمانبندی):

- Different types of CPU-scheduling algorithms have different properties. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms. There are a number of metrics that are commonly used to evaluate the performance of a scheduling algorithm:

  - CPU utilization; How busy is the CPU: is percentage of time that the CPU spends executing programs. We want to keep the CPU as busy as possible. The CPU utilization can range from 0 percent (for a lightly loaded system) to 100 percent (for a heavily loaded system).

Scheduling Criteria cont…

- Throughput: is number of processes completed per time unit. E.g. 10 jobs/minute. For long processes, this rate may be one process per hour; for short processes, it may be ten processes per second.
- Turnaround time; How long from job submission to job termination. It is the total time until the process leaves the running state.

  Turnaround time= time spent waiting to get into memory+ time waiting in the ready queue+ time executing on the CPU+ time doing I/O.

Scheduling Criteria cont…

- Waiting time: amount of time a process has been waiting in the ready queue before it goes to the running state.
- Response time: The turnaround time is generally limited by the speed of the output device. In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response.

Scheduling Criteria cont…

- Fairness: giving each process a fair percentage of CPU
- Optimization:
  - Maximize throughput
  - Maximize CPU utilization
  - Minimize average turnaround time
  - Minimize waiting time
  - Minimize response time.

CPU Scheduling algorithms cont…

- Common CPU scheduling algorithms include:
  - First-Come First-Served Scheduling (FCFS)
  - Shortest job first Scheduling (SJF)
  - Shortest remaining time first (SRTF)/ preemptive SJF
  - Round Robin Scheduling (نوبت گردشی)/ preemptive FCFS