COMPS267F Chapter 3

Process Management

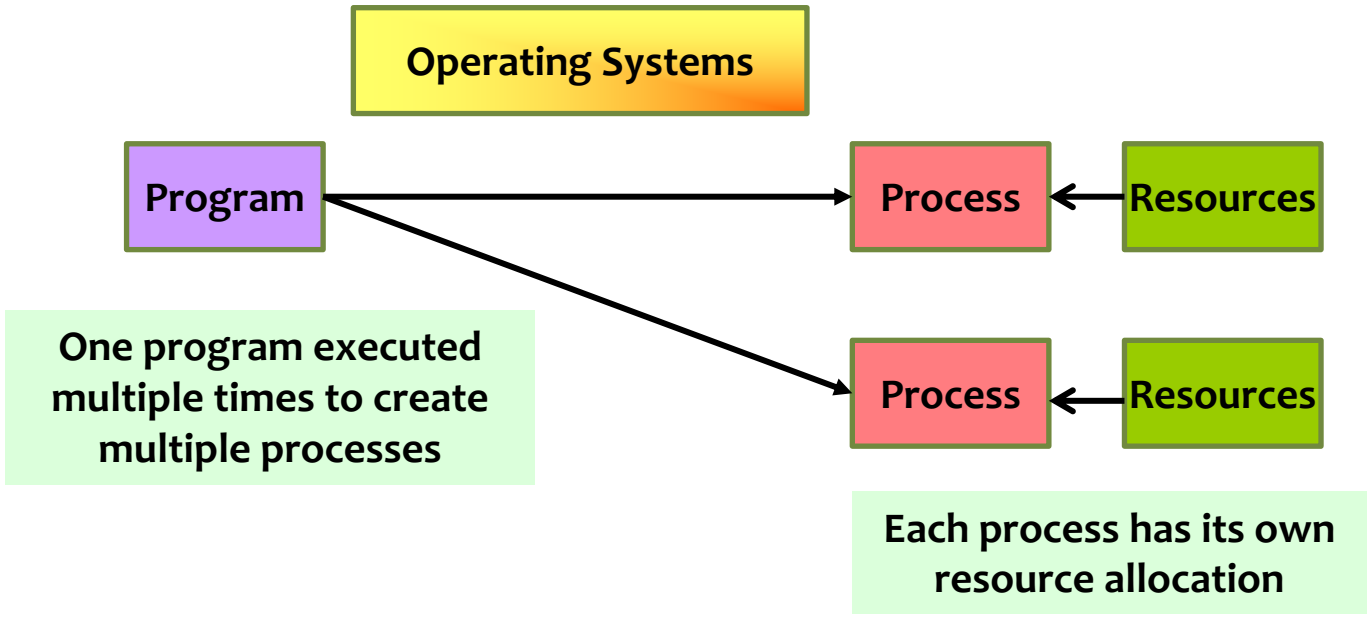
Dr. Andrew Kwok-Fai LUI

Definitions

- Avoid mixing up these terms
  - Multi-processing
  - Single-programming (uni-programming)
  - Multi-programming

executing multiple programs on a computer

From Programs to Processes



Resource Need of a Program

```
int sumOfArray(int array[], int len) {
  int i, sum = 0;
  for (i=0; i<len; i++) {
    sum += array[i];
  }
  return sum;
}
int main() {
  int array[256];
  int i;
  FILE* fp;
  printt("Example Program\n");
  fp = fopen("input.txt", "r");
  for (i=0; i<256; i++) {
    array[i] = fscanf("%d");
  }
  fclose(fp);
  printf("%d\n", sumOfArray(array, len));
}
```

CPU

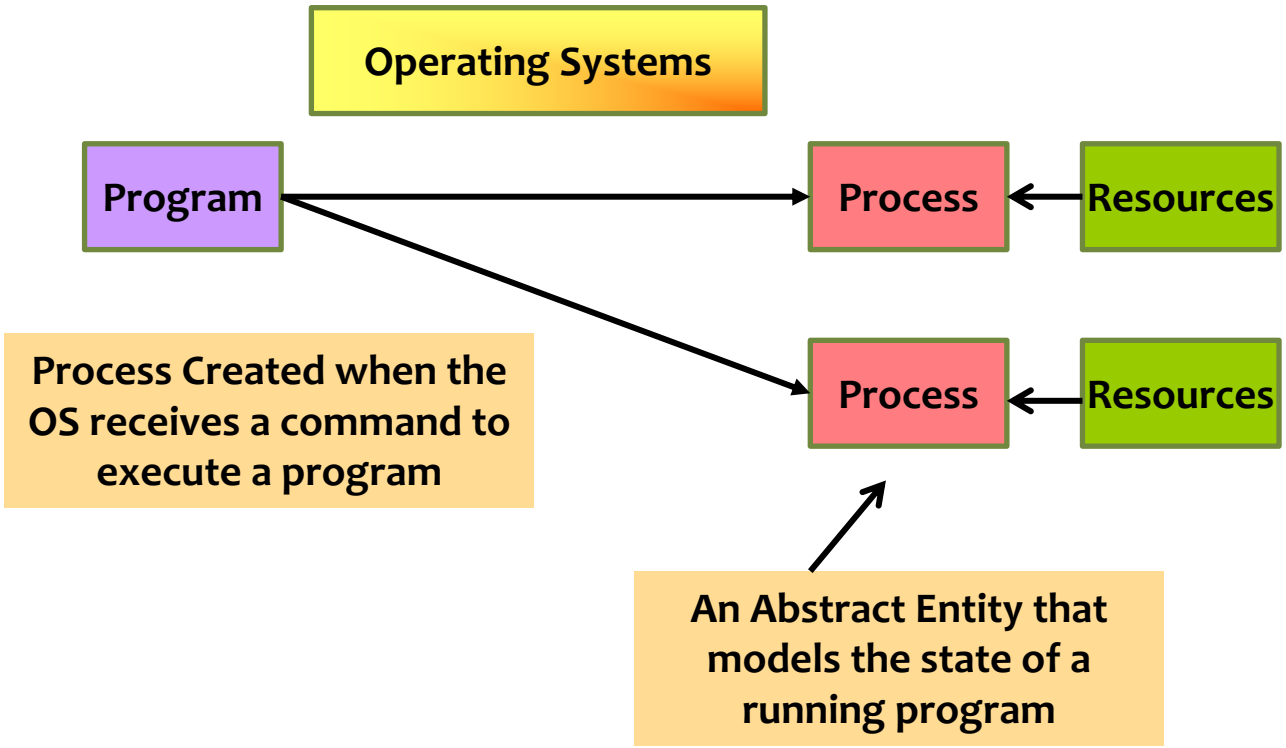
Memory

Memory

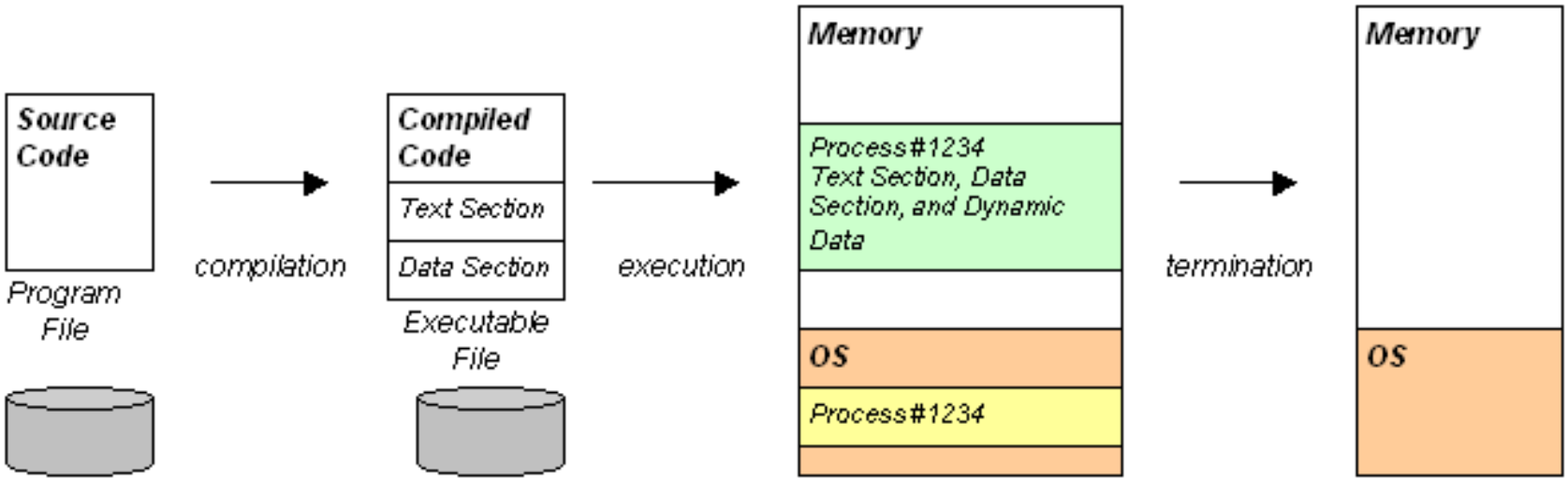
Hard-disk

Memory

From Programs to Processes



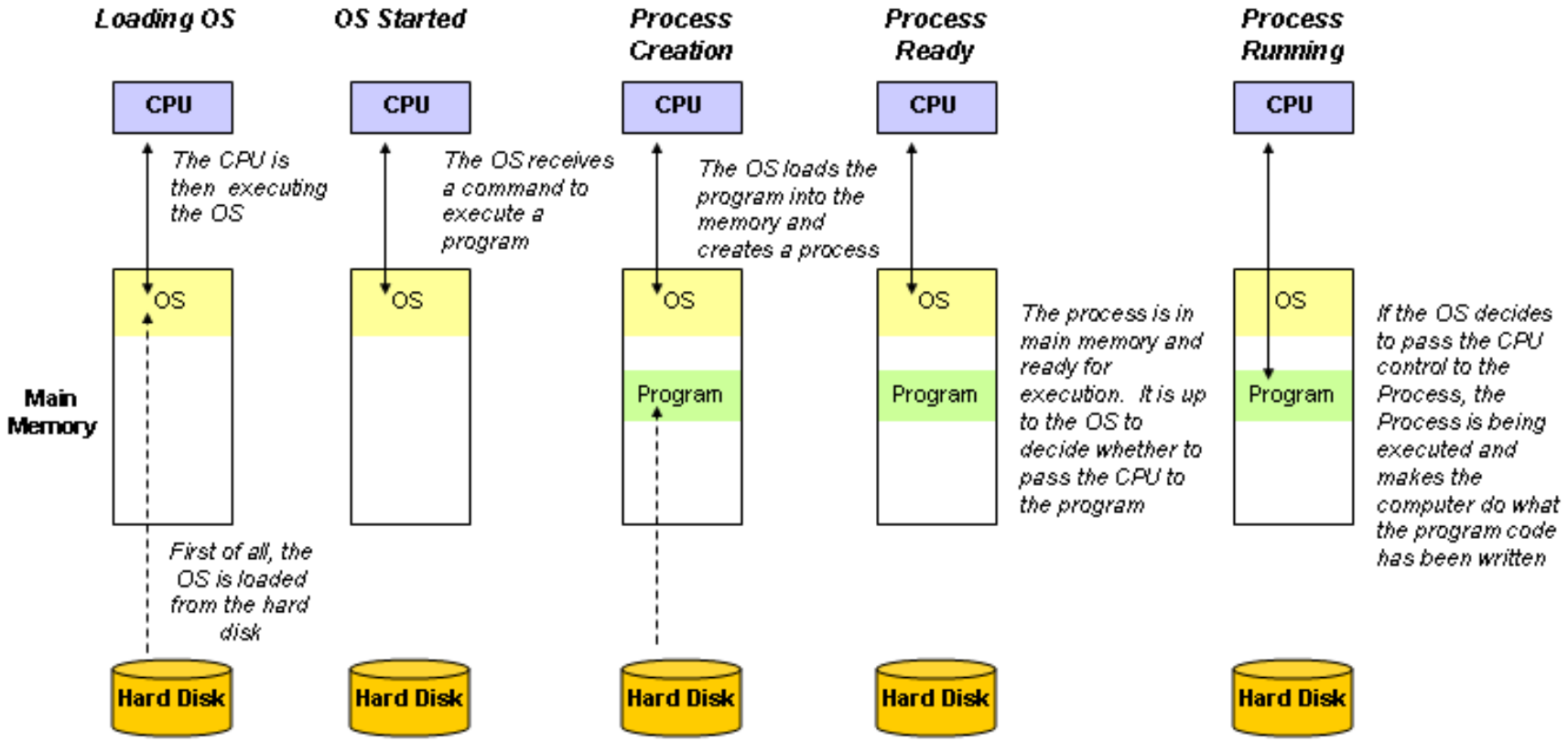
Birth and Death of Process



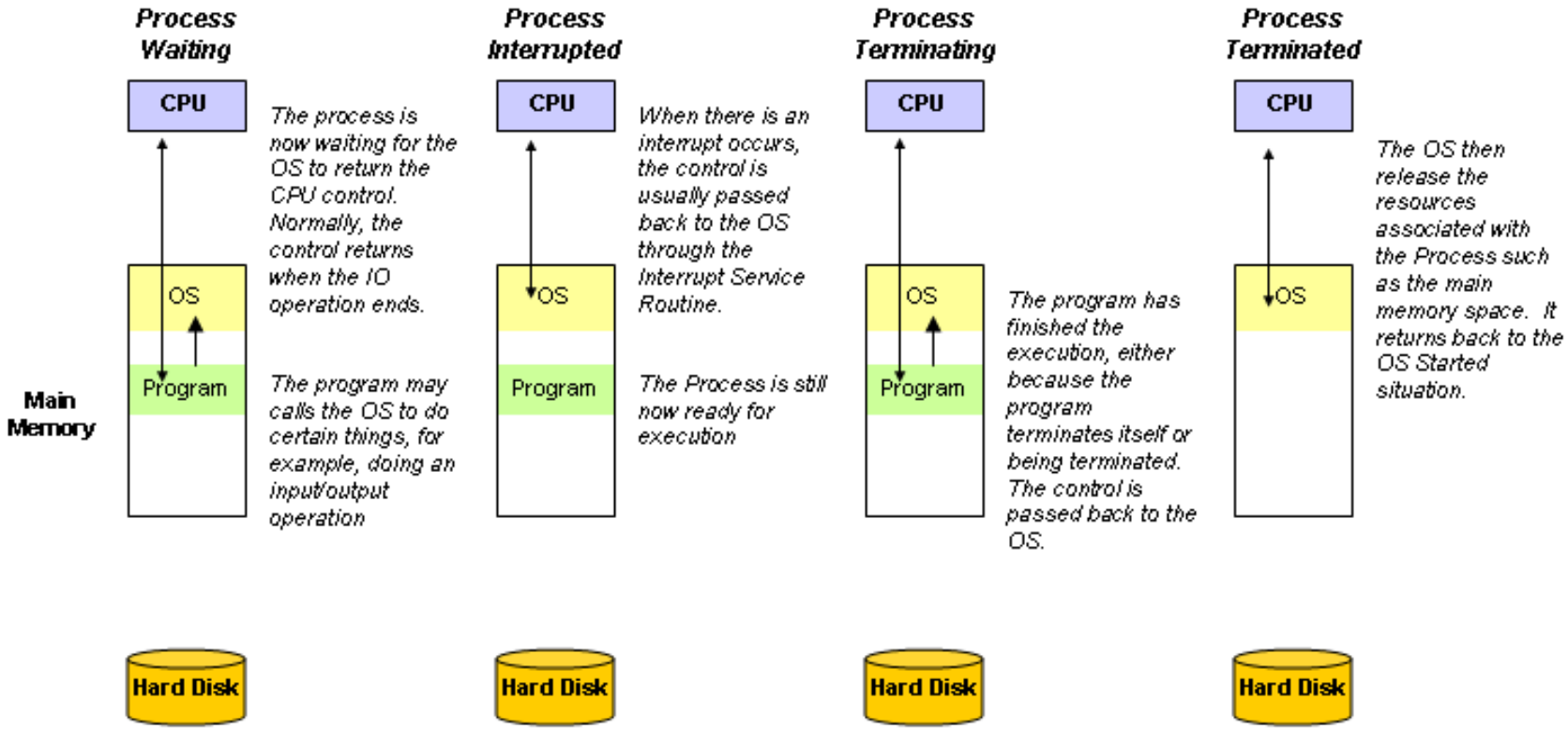
What is a Program?

- Ask the same question again?
  - An executable file
    - Compiled or machine code
    - Marked executable
  - Contains data associated with programs (such as constants)
- Executed interactively (double-click) or a command shell

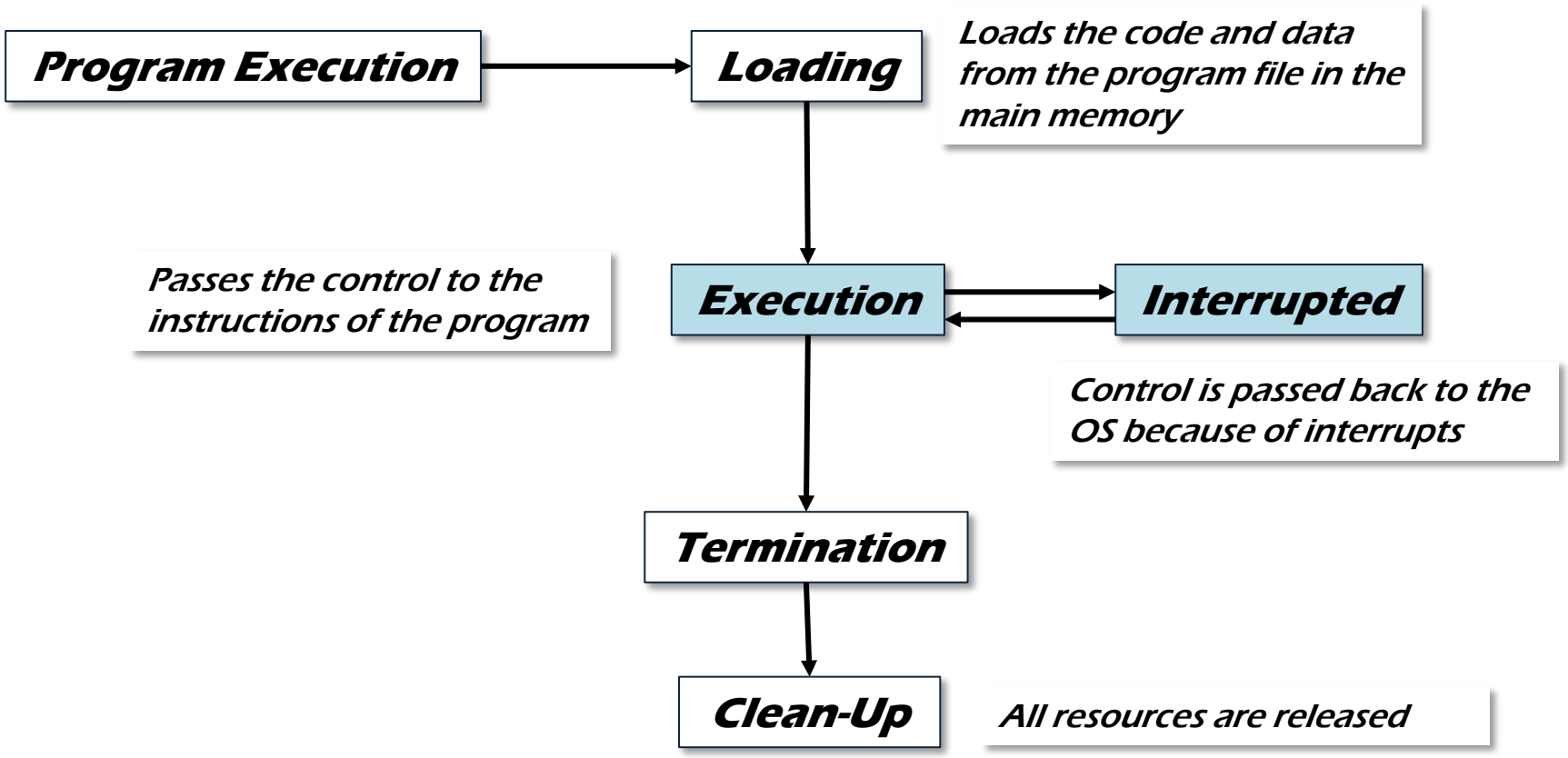
Life Cycle of a Process



Life Cycle of a Process



Life Cycle of a Process



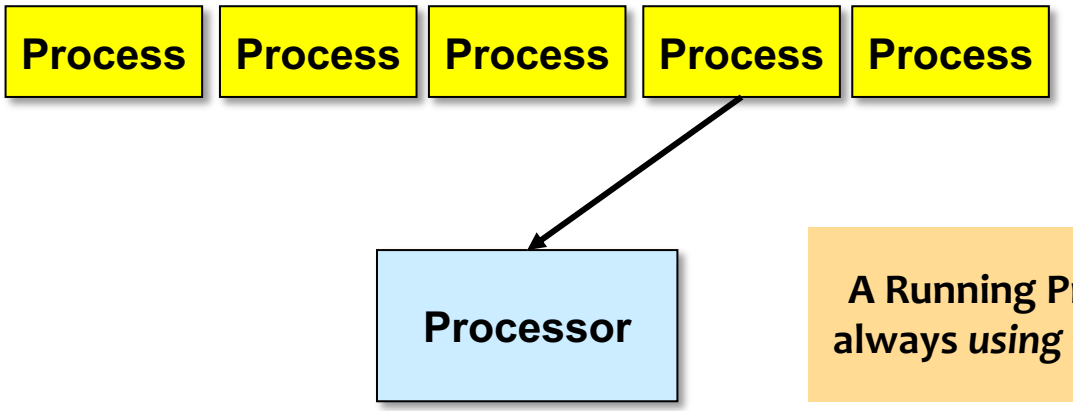
a secret of multiprogramming

A Secret of Multi-programming

- More than one process exists on a computer at a time
  - How can it happen?
  - How can two programs running on one processor?
    - It can happen on computers with one single-core processor

A Secret of Multi-programming

A Running Process is not always *running*



A Running Process is not always *using the Processor*

A Secret of Multi-programming

Its Instructions are being executed by the Processor

It is doing I/O or waiting for I/O to finish

It is simply waiting

On interactive systems, users are slow and GUI is often idle

Processor

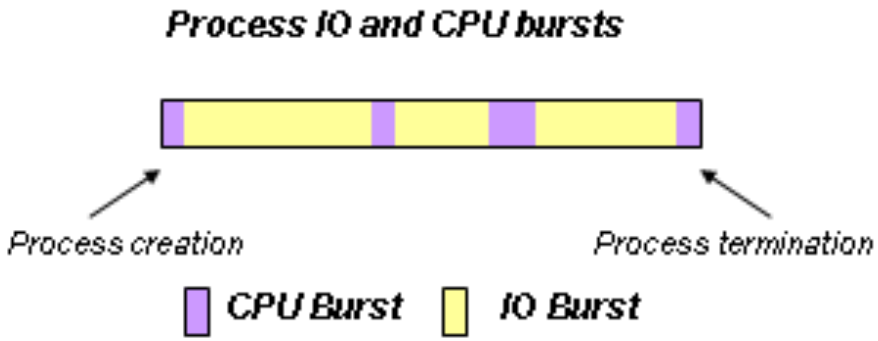
The Internet is also slow and process needs to wait

A Secret of Multi-programming

- Only few programs uses a lot of processor
  - Training an artificial intelligence (AI) machine learner
  - Doing scientific calculation (i.e. simulation of weather)
- Most other programs are IO-bound programs

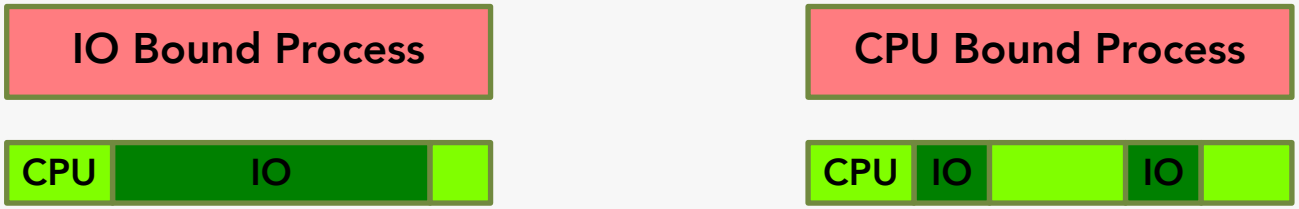
IO Bursts and CPU Bursts

- Programs execution involves alternative periods of CPU bursts and IO bursts
  - IO Burst: Process doing or waiting for IO
    - Requiring IO Devices
  - CPU Burst: Process instructions are being executed
    - Requiring CPU



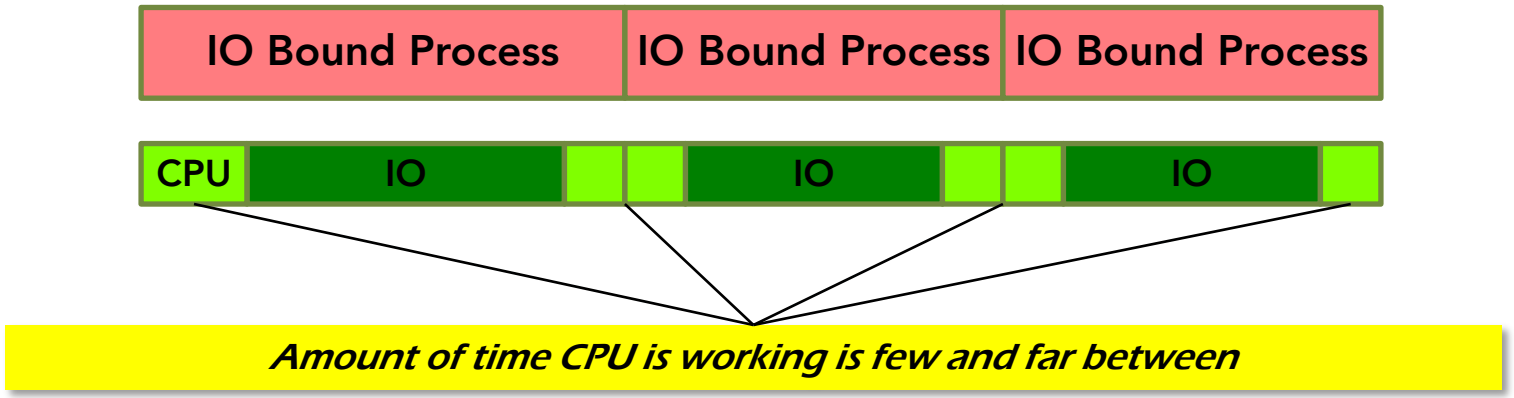
IO Bound and CPU Bound Processes

- According to the ratio between IO Burst and CPU Burst
  - IO bound processes
    - Processes waiting or doing a lot of IO
    - Influenced by IO performance
  - CPU bound processes
    - Intensive execution of code such as scientific calculation
    - Fewer IO operations
    - Influenced mainly by CPU performance



Consequence of IO Bound

- Result in low CPU utilization
  - In waiting for IO, CPU becomes idle
  - Even the most CPU bound process will perform IO

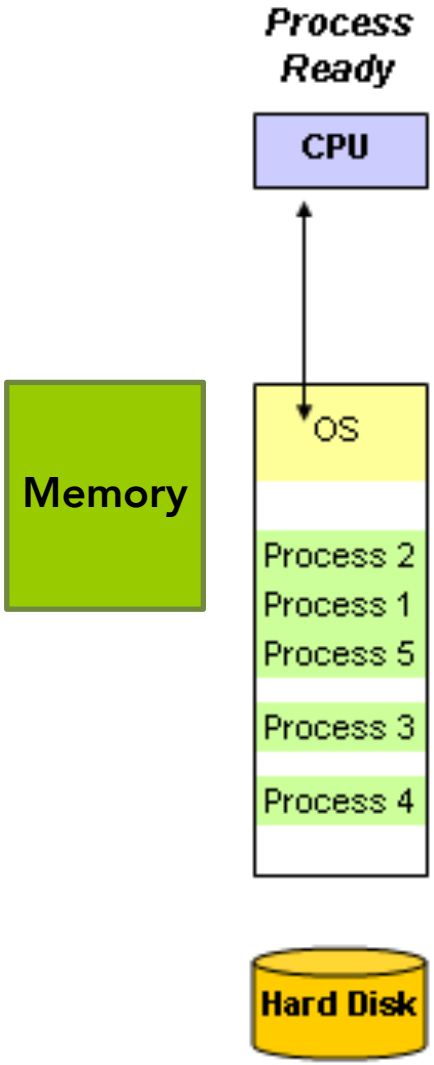


benefits of multi-programming os

Single-Programming OS

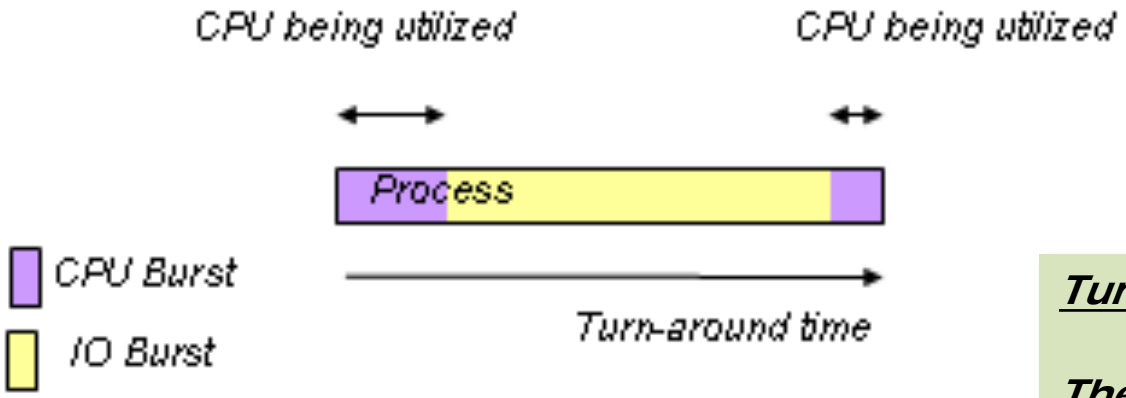
- Single-programming OS means waste of money
  - Execution of only one program is allowed
    - Cannot fully utilize the CPU
    - For example, IO-bound program spending 90% I/O, only 10% CPU
    - CPU is utilized by programs 10%

Multi-Programming OS



Multi-Programming OS

- Multi-programming should improve CPU utilization but it might impact on the turn-around time



CPU Utilization Rate

The proportion of time that the CPU is doing the work of executing user programs.

Turn-around Time

The time between the creation of a process and the termination of a process.

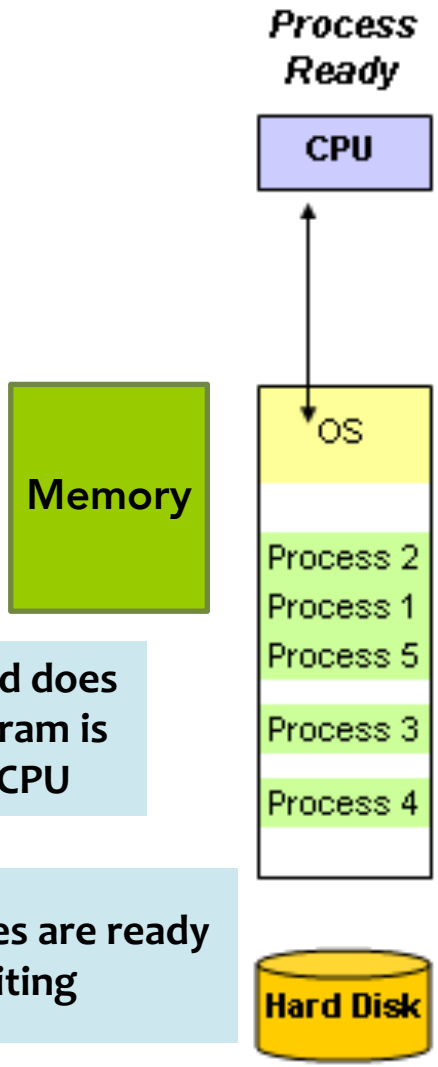
Multi-Programming OS

- Consider a set of processes that spend 90% time on IO bursts, running on a single processor
  - Run 10 such processes at the same time
  - The chance of all 10 processes doing the IO is low, at least one of them should want to do CPU burst.
- If there is at least one process ready to use the processor, the processor is utilized
  - The more processes there are ready, the more likely one of them is in CPU burst.

High CPU Utilization

- The strategy to make money spent worth more
  - Increase the number of ready processes
  - Reduce setup time of processes
- The method to achieve the above
  - Load multiple programs into the main memory
    - They are at least ready for execution in one aspect
    - Loading take time and loading is done first even if the process cannot use the CPU now
  - Setup time can be quite slow
    - Running the next process needs context switching
    - More on that later

Multi-Programming OS



At most one program can be executed by the CPU at each moment

A program executed does not mean the program is executed by the CPU

The memory size has restricted the number of programs

Many processes are ready but waiting

Example

Example 1: CPU Utilization and Turn-Around Time

A set of five processes is to be executed on a single processor computer system. Each process has the following characteristics: the turn-around time is 10 ms, and it spends 20% of the time in CPU bursts and 80% in IO bursts.

Calculate the CPU Utilization and the overall turnaround time of the five processes if only the execution a single process is allowed (non multi-programming).

The processes must be executed in the following pattern. The execution of one process first. Only after the process has completed execution, the execution of the second process will begin.

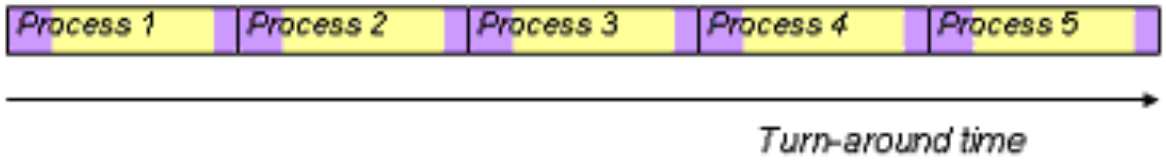
Total turnaround time is 10 ms (each process) x 5 = 50 ms

Each process spends 10 ms x 20% = 2 ms using the CPU and 10 ms x 80 % = 8 ms doing IO operations.

Totally the time the CPU is being used is 2 ms (each process) x 5 = 10 ms.

CPU Utilization = 10 ms / 50 ms = 20%

This is a low utilization rate.



Example: Multi-Programming

Example 2: CPU Utilization and Turn-Around Time

Consider the above example again.

Calculate the CPU Utilization and the overall turnaround time of the five processes if multi-programming is supported.

Assumption #1: Ignore the time taken by the OS to do work in executing the processes.

Assumption #2: Many IO operations cannot happen in parallel.

The OS will execute all five processes at the same time. The OS first loads the program of these five processes into the main memory.

There is only one CPU and so the OS will select one of the processes in CPU bursts to use the CPU. When the process selected has finished one CPU burst and doing IO operations, the OS will select another process in CPU burst to use the CPU.

The OS tries to arrange the five processes so that the CPU is not idling.

Total CPU burst time of 5 processes = 2 ms (each process) x 5 = 10 ms

Total IO burst time of 5 processes = 8 ms (each process) x 5 = 40 ms

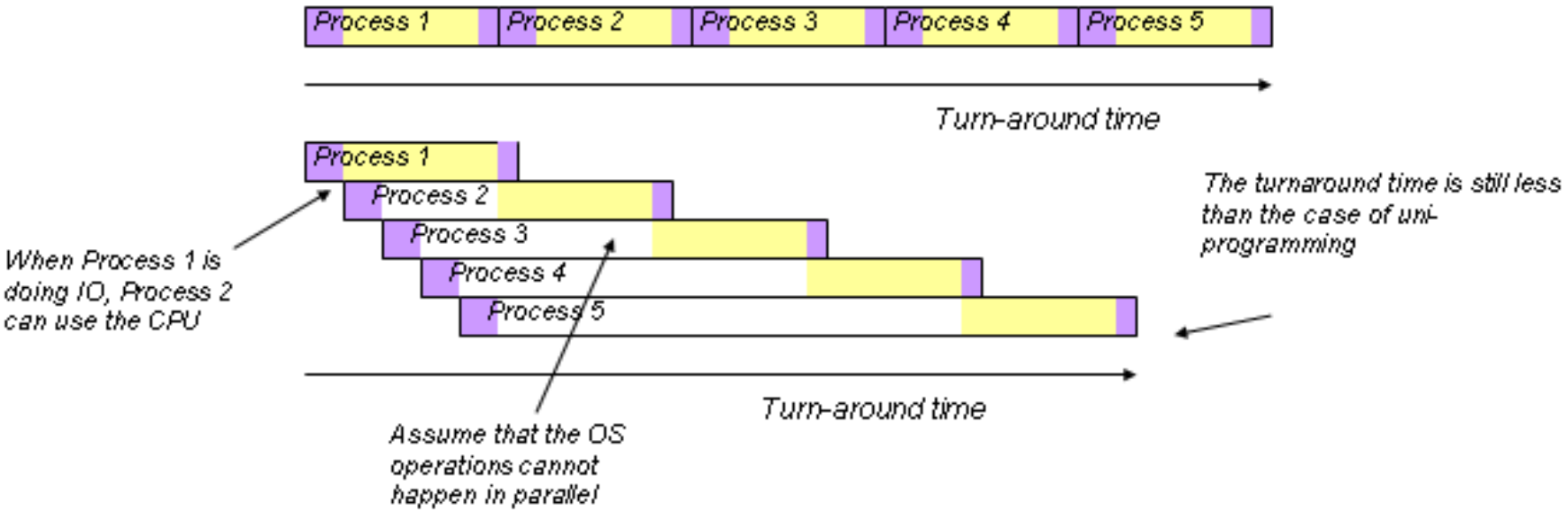
The CPU burst time can happen in parallel with the IO burst time, except for the first process.

Turnaround time = 40 ms (IO burst time) + 2 ms (CPU burst time of first process = 42 ms

CPU Utilization = 10 ms / 42 ms = 24%

The turnaround time is shortened and the CPU utilization improved.

Example: Multi-Programming



Example: Multi-Programming

Example 3: OS Process Management Overhead

Consider Example 2.

In a simplified model, the multi-programming OS spends on average 1 ms per process in overhead operations such as loading programs. Assume that these overhead operations of the processes cannot happen in parallel.

Calculate the turn-around time and CPU utilization for the situation of Example 2.

The total time for the OS to manage the five processes = 1 ms x 5 = 5 ms

Turnaround time becomes = 40 ms (IO burst time) + 2 ms (CPU burst time of first process) + 5 ms (overhead) = 47 ms

CPU Utilization = 10 ms / 47 ms = 21%

The turnaround time is longer and the CPU utilization reduced.

process management

Process Management in Multi-Programming OS

- Keep records of state of processes
  - States are defined by different resource requirement
    - It is being loaded and created
    - It is ready for execution and waiting for the CPU
    - It is controlling and using the CPU (in actual execution)
    - It is doing or waiting for IO
    - It is being terminated
  - State transitions according to defined conditions
  - Data structures for efficient processing
  - Algorithms for achieving high system performance

Process Management in Multi-Programming OS

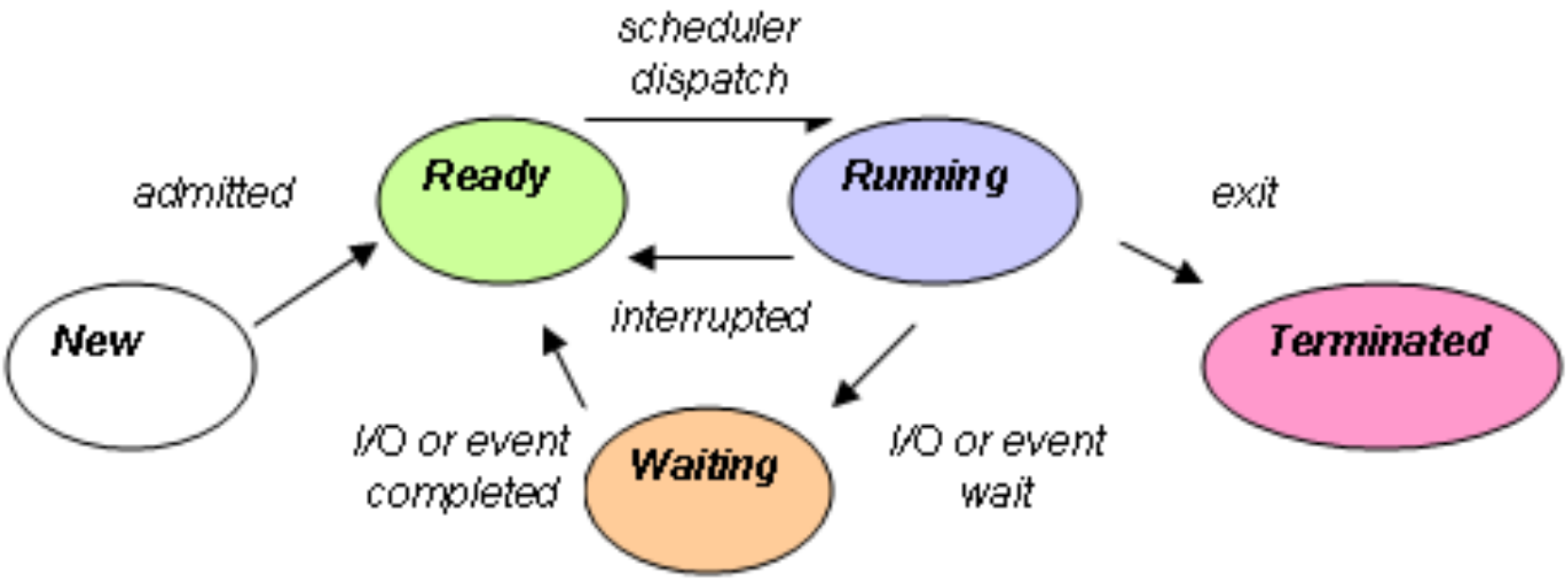
- Keep records of state of computing resources
  - The amount of spare main memory for process creation.
  - The availability of the CPU for process execution.
  - The availability of IO devices for assigning processes in IO bursts to wait or to use the device.

Process Management in Multi-Programming OS

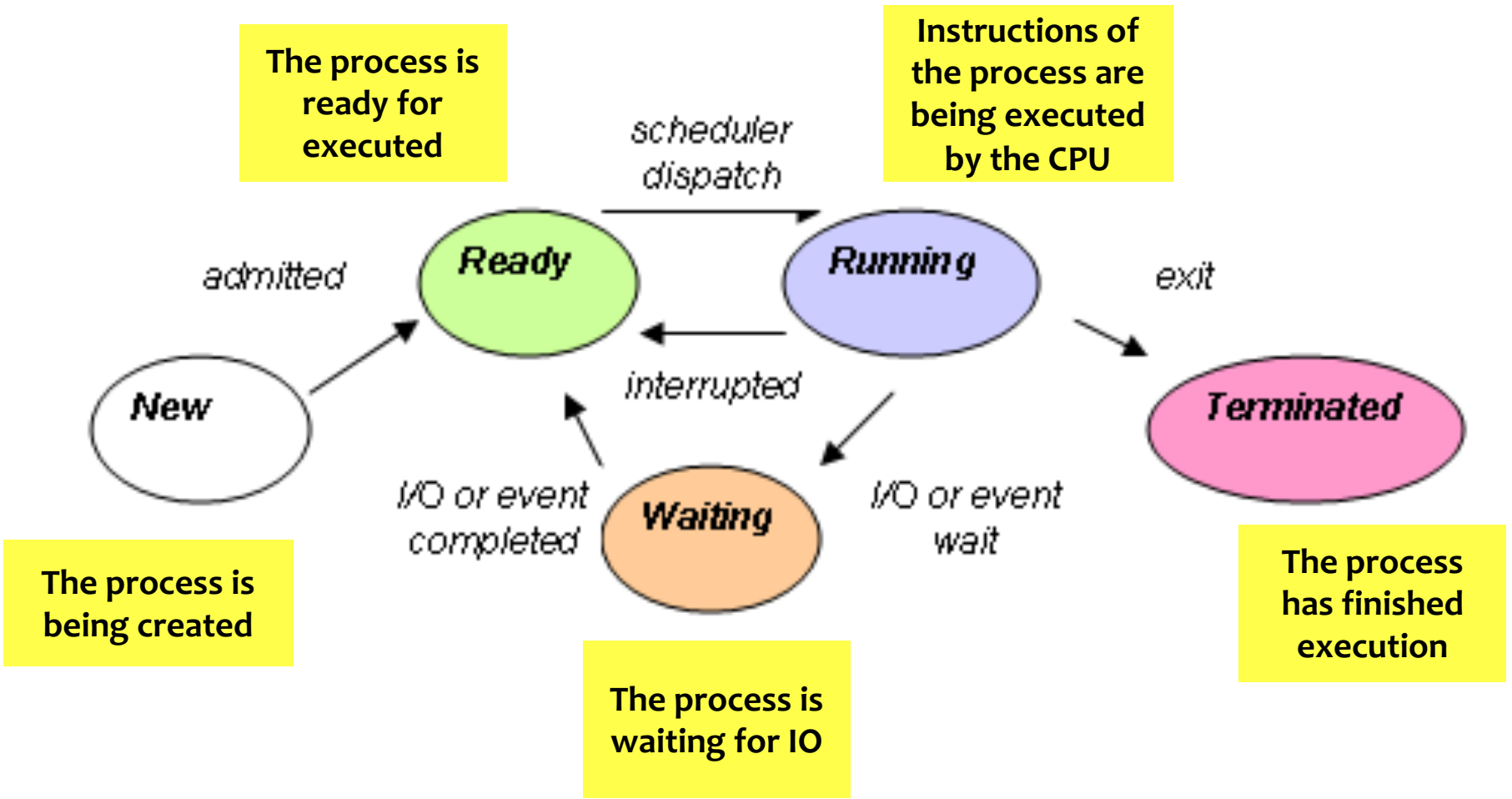
- The work done by the OS is process management is considered to be overheads
  - These are not useful work
  - Running user programs are considered useful
- Consequences of management overheads
  - Reduced CPU utilization
  - Increased turn-around time

the five-state model

The Five-State Model Process Management



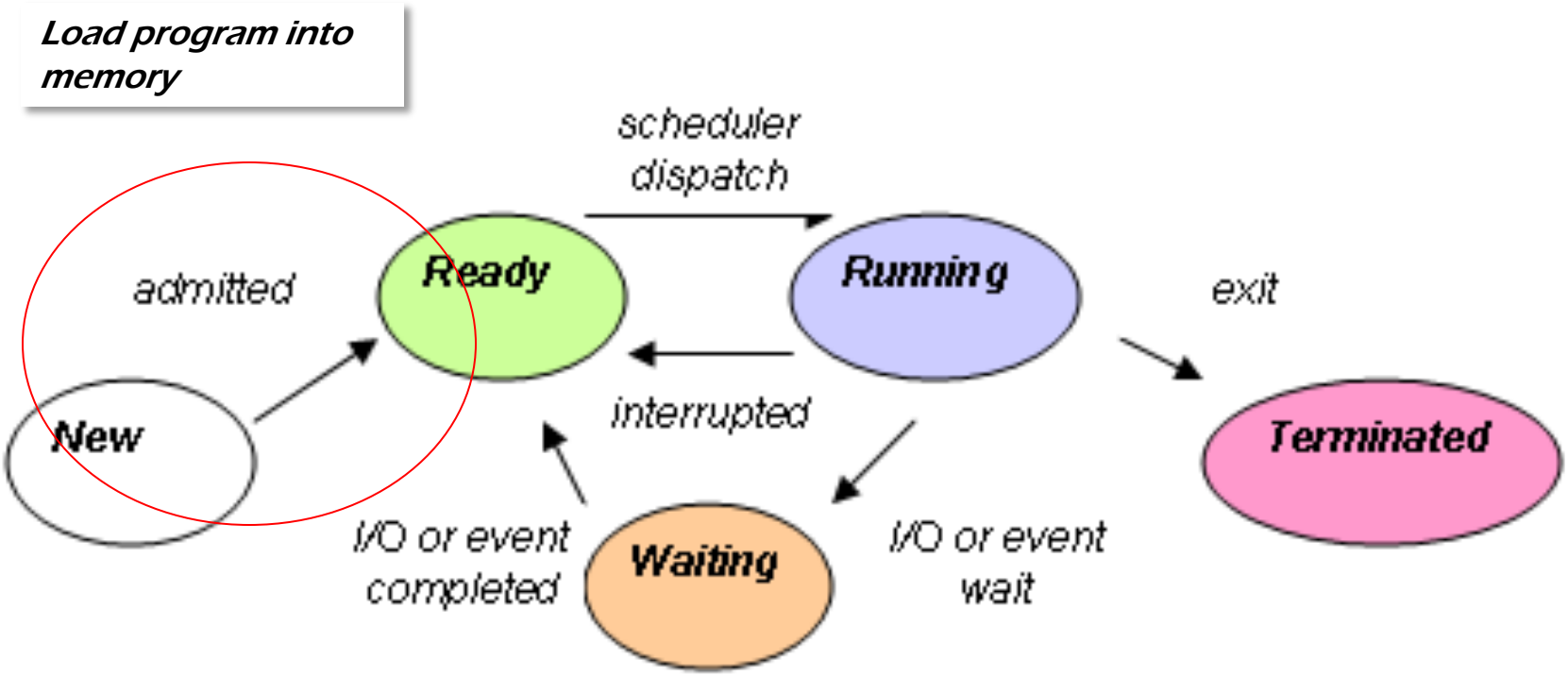
The Five-State Model Process Management



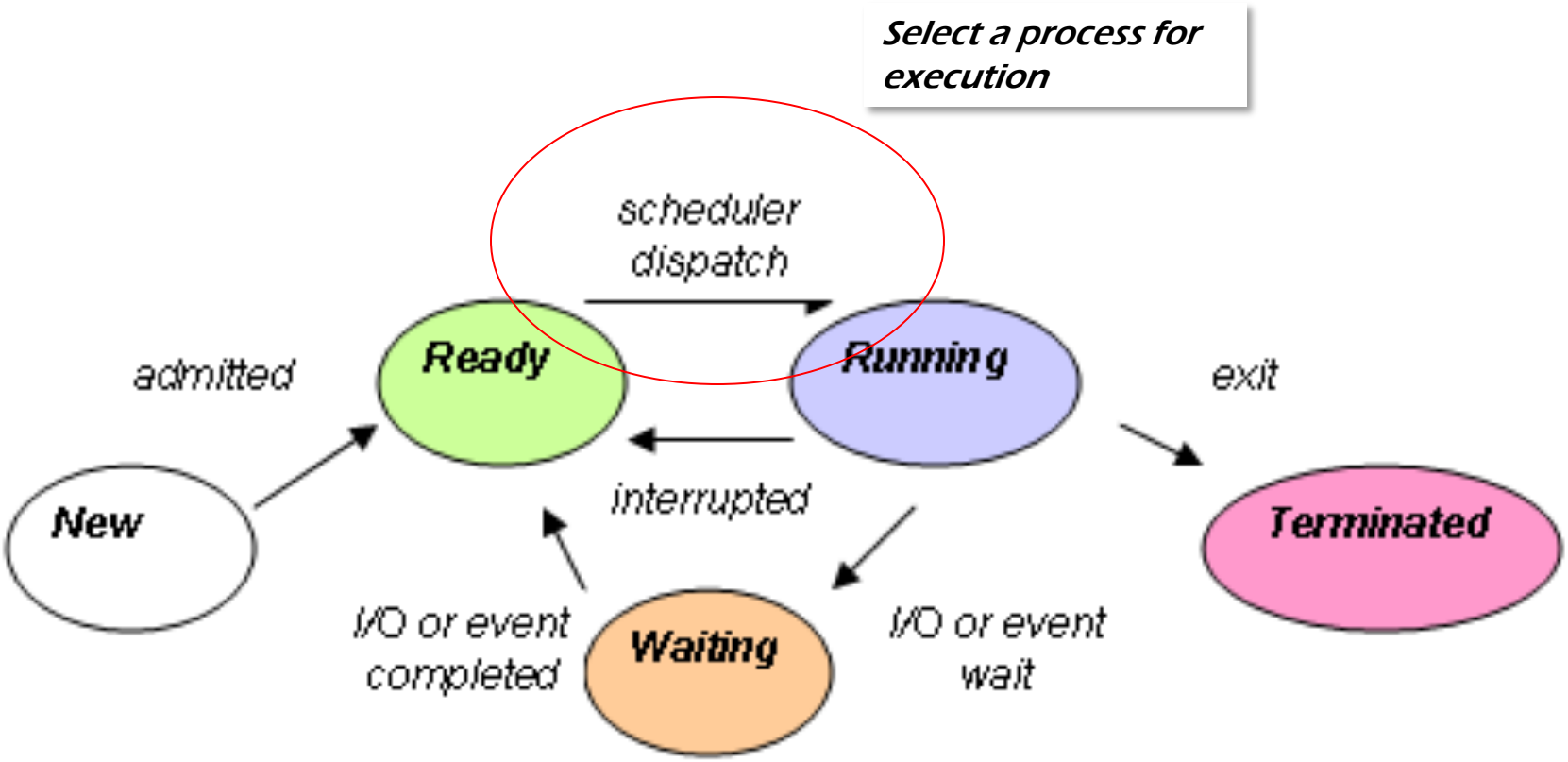
The Five-State Model Process Management

- The five states represent the typical operations of a process in multi-programming OS.
- This model is only the most common model used in OS literature.
  - There are 2, 3, 4, and 7-state models.

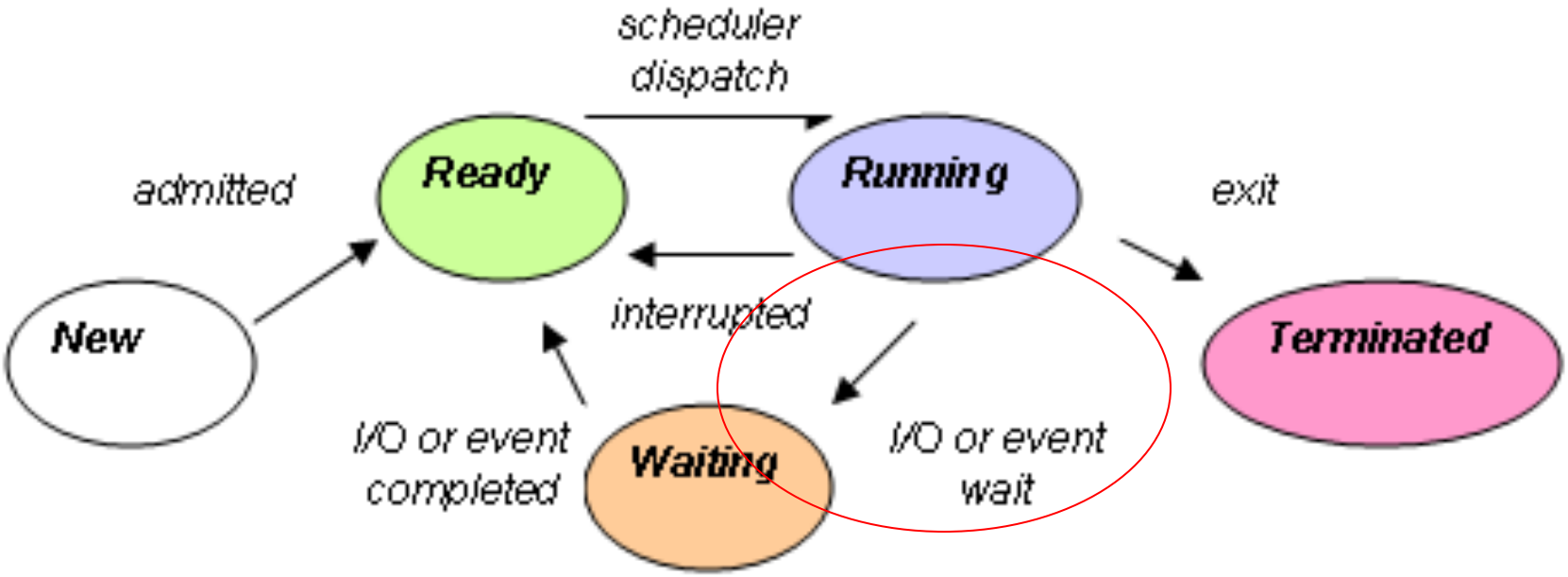
Transition from One State to Another



Transition from One State to Another

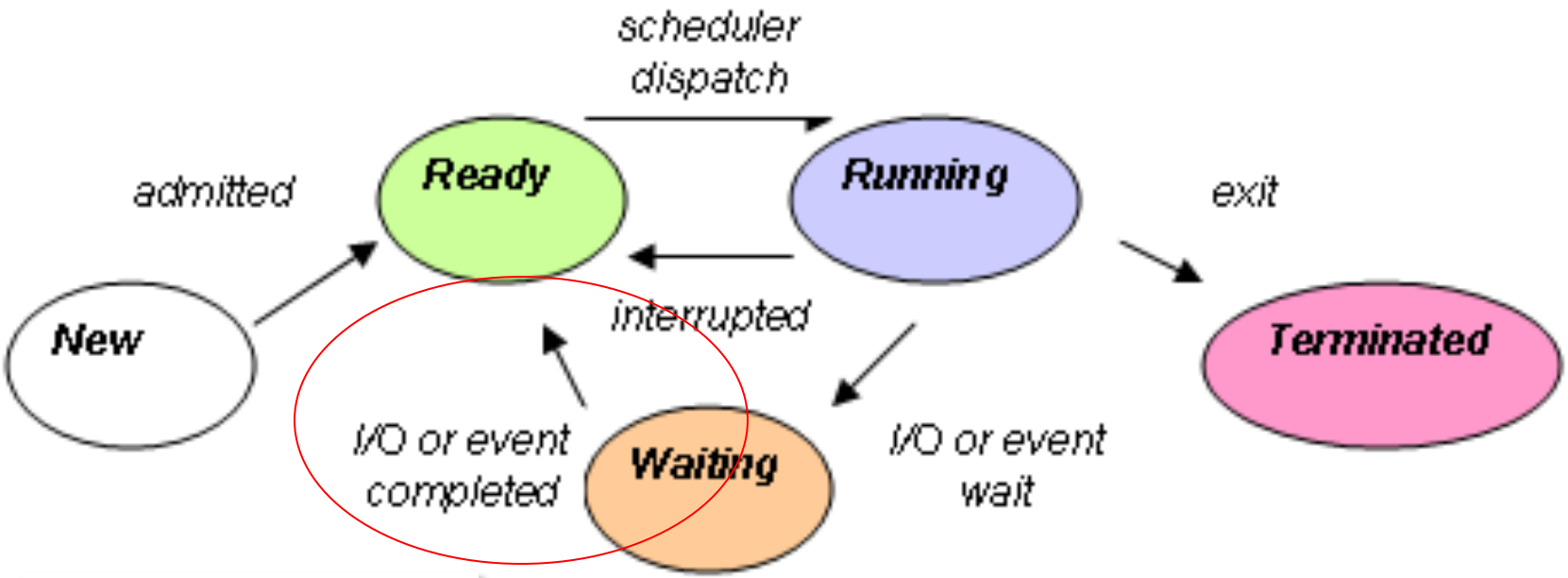


Transition from One State to Another



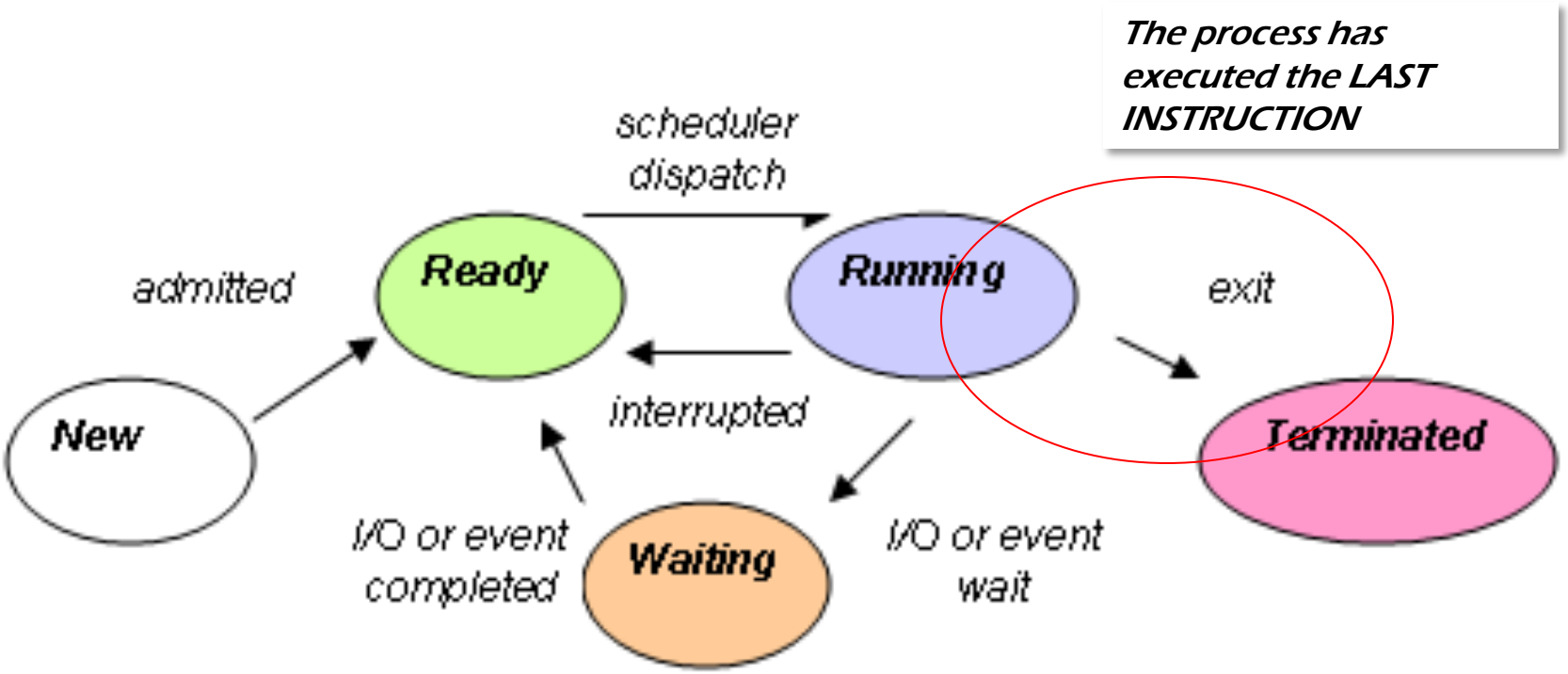
A process cannot continue execution because it has to perform an IO operation

Transition from One State to Another

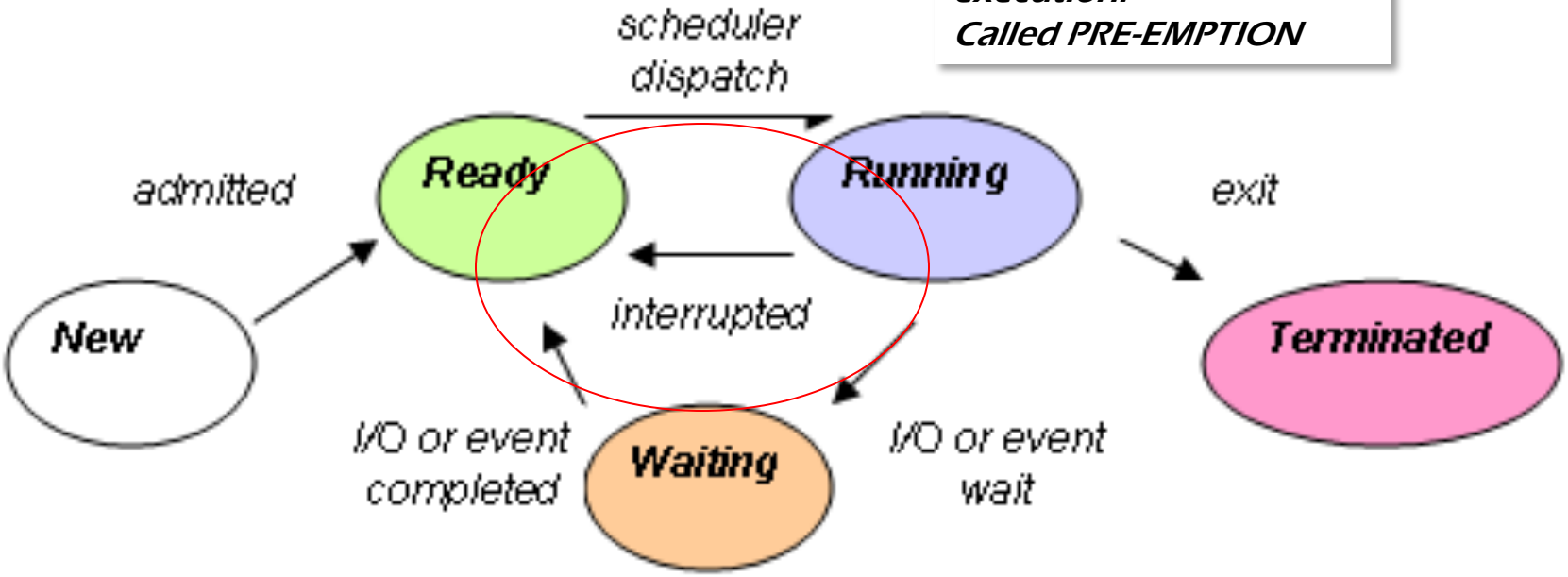


The process finished IO operation and ready for execution again

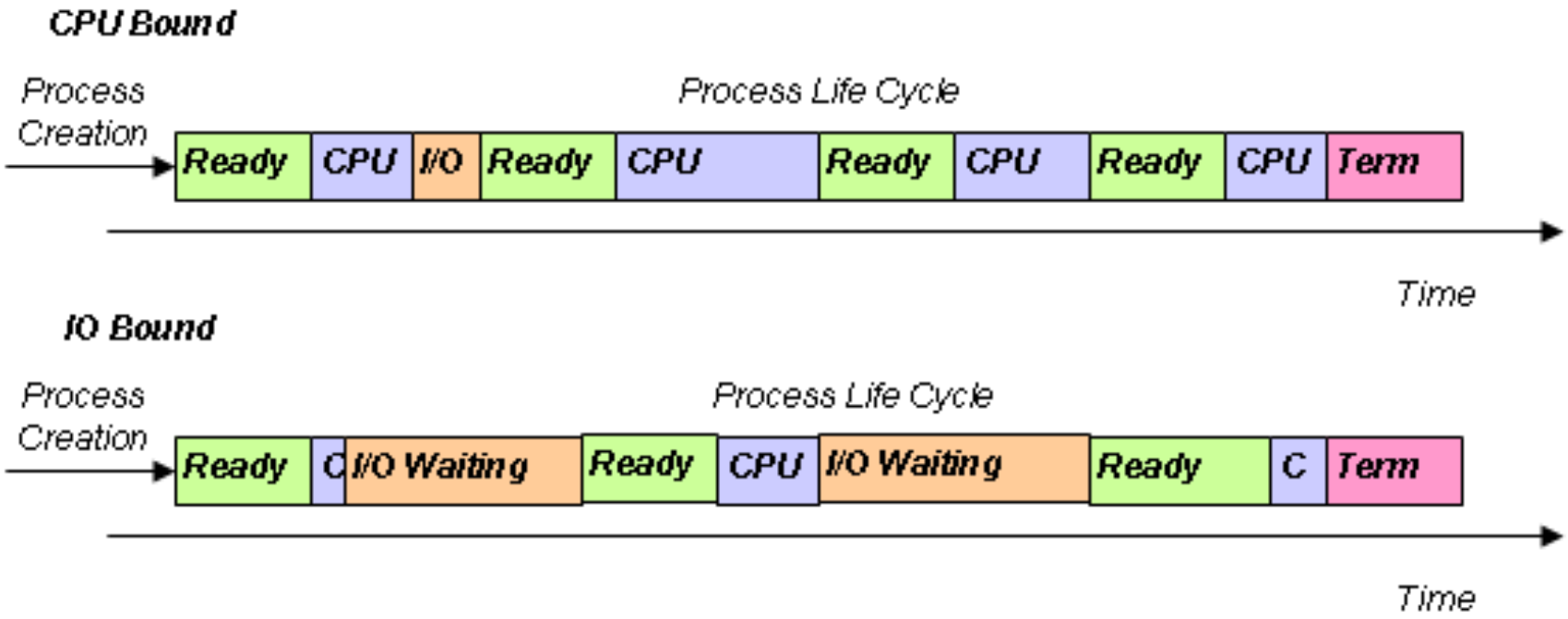
Transition from One State to Another



Transition from One State to Another



Transition from One State to Another



key concepts in process management

Process Control Block (PCB)

- A table-shape data structure contains important dynamic information about the state of a process
  - One PCB one process
    - Which state is the process?
    - Where is the process in memory?
    - Doing I/O?
  - PCB also stores CPU registers if the process is suspended from CPU execution
    - The stored values will be copied back to the CPU when resume
  - Usually stored in main memory

Process Control Block (PCB)

Data	*Descriptions*
Process State	New, Ready, Running, etc.
Program Counter	The position of the program where the CPU is executing.
CPU Registers	The data involved in CPU calculations and data manipulations are saved here: accumulators, index registers, stack pointers, condition code (overflow, carry, etc).
CPU Scheduling Information	Process priority, queues, and other scheduling information.
Memory Management	Base and limit registers, and other information for memory management (to be covered later).
Accounting Information	Amount of CPU time used, quotas, and other book-keeping information.
I/O Status	Keep track of the I/O status: lists of open files.

Program Counter and Context

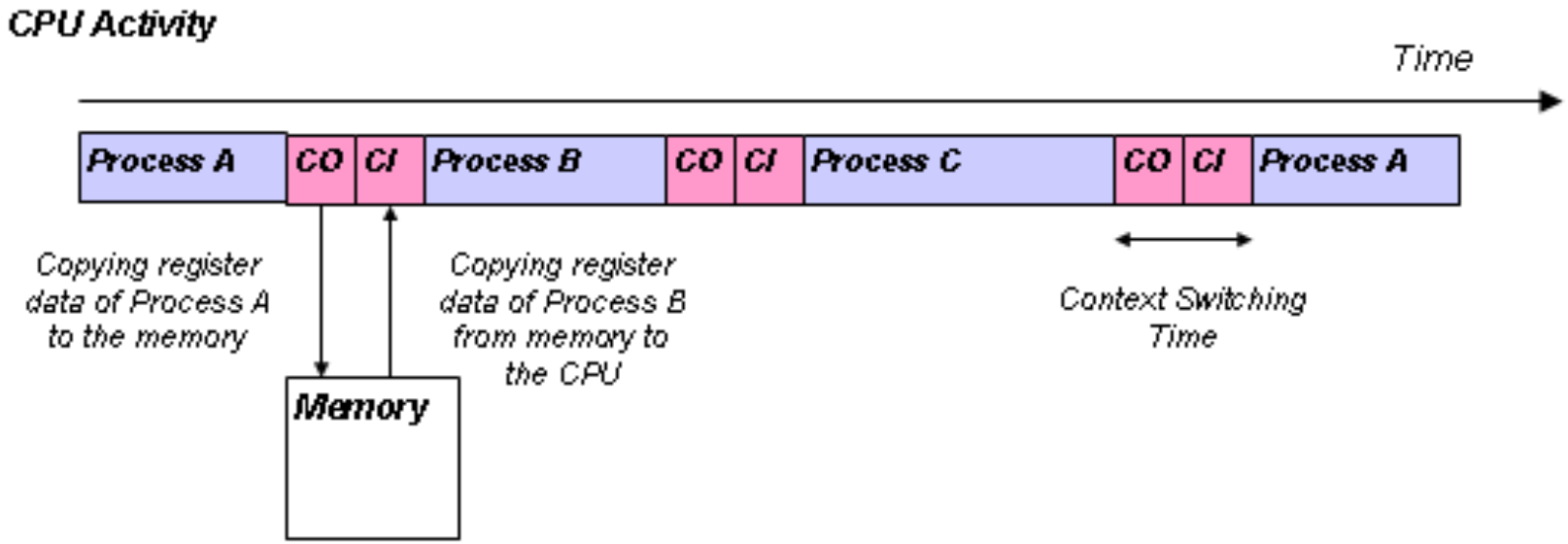
- The program counter is stored in the PCB when the process is suspended
  - It indicates which instruction should be executed next
  - The process should resume at the address of the last instruction executed

Program Counter and Context

- The values in other registers are also essential to keep
  - The process should resume the instruction execution in the correct context
    - The registers are changed by the earlier process using the CPU.
    - The registers and associated data are known as the context
- Example:
  - A process will perform X = A + B
  - The value of A and B must be loaded into the CPU registers first
  - Later X will also be available in a CPU register
    - Further calculation is often happening
  - The set of registers are known as the context

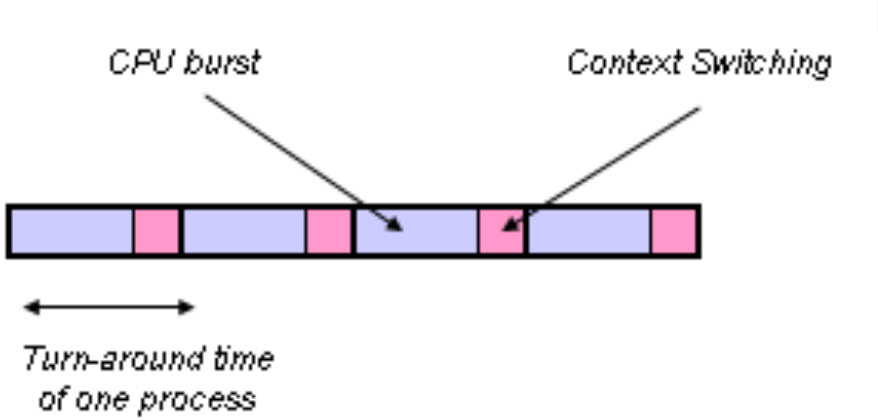
The Context

- The context is the data being used for the continued execution of a process
  - The context of an out-going process should be copied-out (CO)
  - The context of an in-coming process should be copied-in (CI)



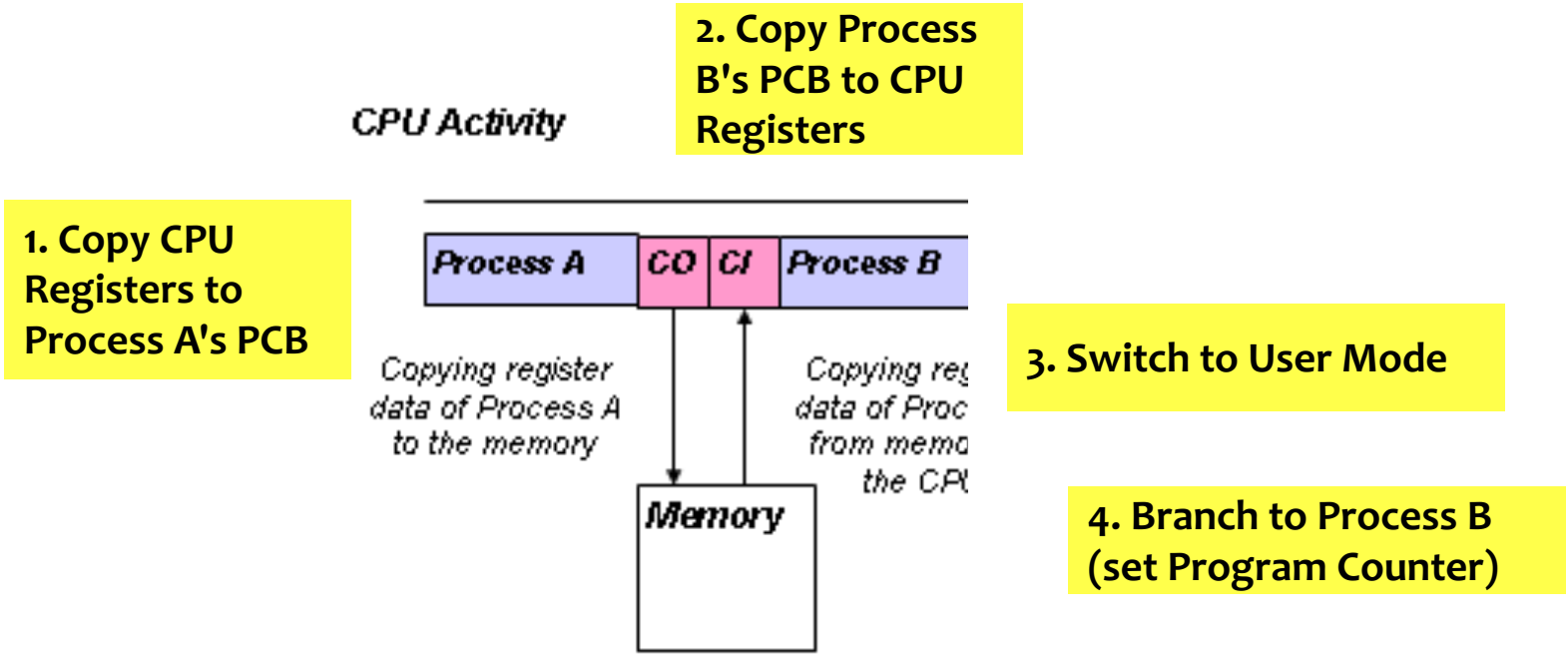
Context Switching

- The operation to load the CPU registers with correct data before a process resume or begin execution
  - CPU registers are stored away when leaving the Running state
  - CPU registers are loaded in before entering into Running state
- Context switching uses CPU time
  - Considered a management overhead
    - The shorter it is the better
    - Efficient dispatcher is important



The Dispatcher

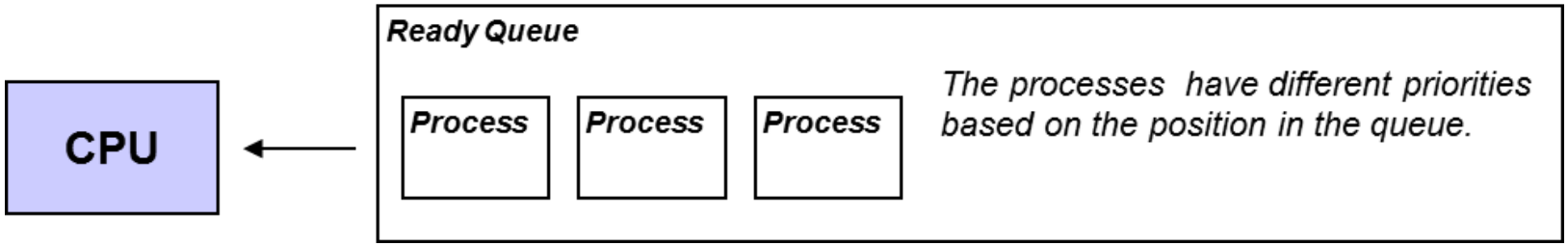
- The module executes context switching between processes
  - Make sure everything is done quickly



The Ready Queue

- A number of processes are found in the Ready state
  - They are placed inside a data structure called the Ready Queue
  - OS can adopt a form of first-come-first-serve manner (or other priorities) of selecting a process for CPU execution

The first in the queue will be given the green light to use the CPU first



Example: Context Switching Efficiency

Example 4: CPU Utilization with Context Switching

A set of four CPU bound processes (named P1, P2, P3, and P4) share the CPU. Assume that the CPU burst time of each of these processes is 10 ms long. The context switching time is 1 ms. If these CPU bound processes take turn to control the CPU that the next process comes on when the last process has terminated. Estimate the CPU utilization.

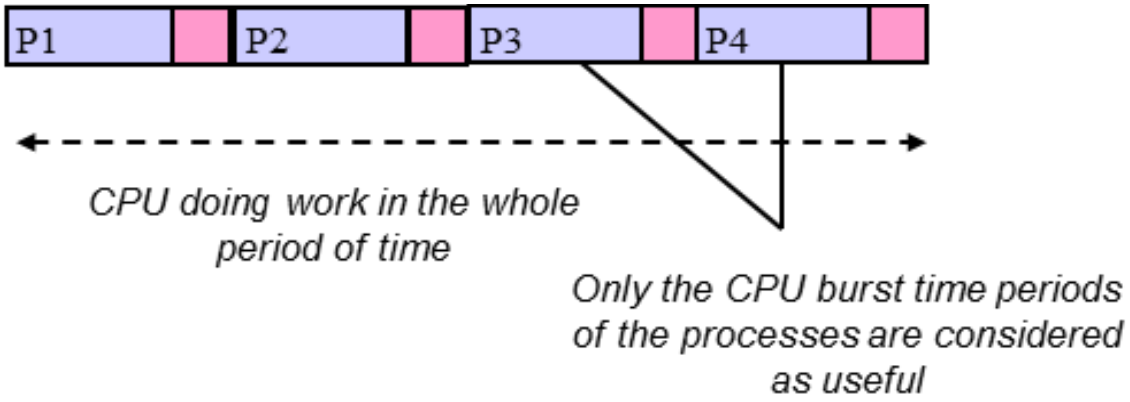
Answer:

For each process, the CPU burst time is 10 ms and context switching is on average 1 ms.

The total CPU work time is 10 ms + 1 ms = 11 ms.

The CPU burst time is the time CPU doing useful work.

CPU utilization = 10 ms / 11 ms = 91%



Example: Context Switching Efficiency

Example 5: CPU Utilization with More Frequent Context Switching

Consider Example 4. If the OS decides that each of the four processes will not complete its execution in one go. Instead, each process will execute its CPU burst time in two steps. So the sequence of executing processes becomes P1 – P2 – P3 – P4 – P1 – P2 – P3 – P4. Estimate the CPU utilization.

Answer:

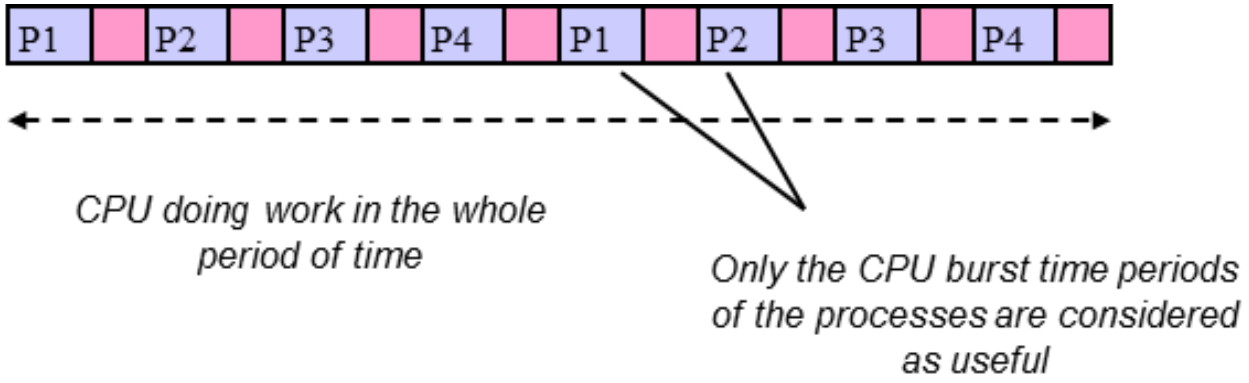
For each process, there are now two context-switching on average.

Sum of turnaround time of all four processes = 10 ms x 4 + 1 ms x 2 x 4 = 48 ms.

The CPU burst time is the time CPU doing useful work.

CPU utilization = 10 ms x 4 / 48 ms = 83%

The CPU utilization reduces if there is more switching between processes.



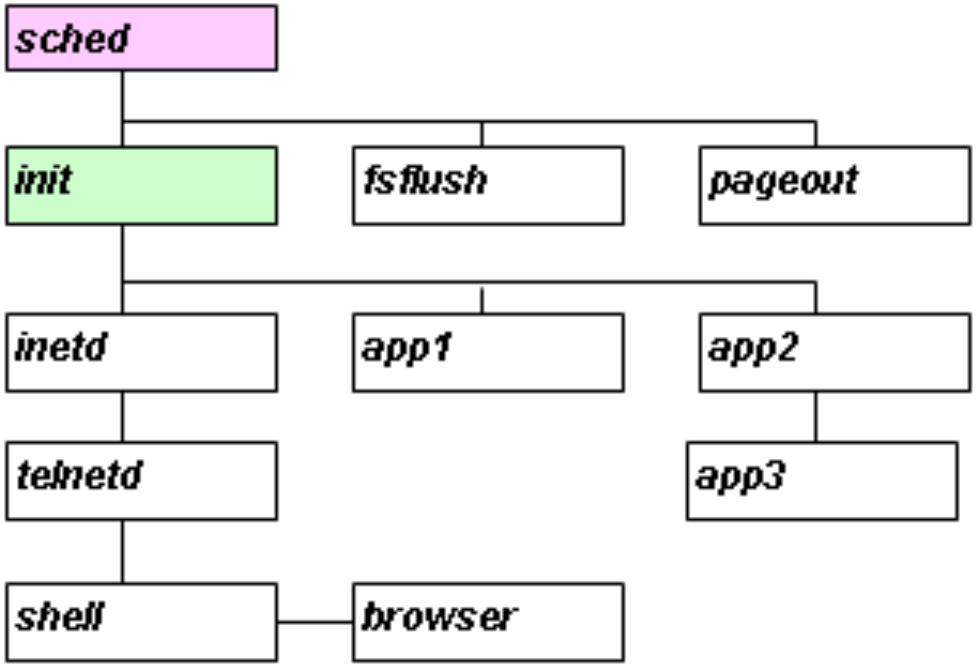
process creation, hierarchy & terminate

Process Hierarchy

- OS is a program
  - Exists as a process during operation
  - In fact, OS consists of a set of processes doing different tasks

Process Hierarchy

- New processes are created from an existing process
  - The first process is created after system boot-up sequence
  - New process is called child process, original the parent process



Process Hierarchy

- A C program example of process creation
  - Process creation with system function `fork`

```
#include <stdio.h>

int main() {
  int pid;
  pid = fork();
  if (pid < 0) {
    fprintf(stderr, "Fork Failed");
    exit(1);
  } else if (pid == 0) {
    /* section executed by the child process */
    printf("Child is sleepingn");
    sleep(10);
    printf("Child waiting for I/On");
    getchar();
  } else {
    /* section executed by the parent process */
    printf("Parent is waitingn");
    wait(NULL);
    printf("Parent received that Child Completen");
    exit(0);
  }
}
```

The pid is different in the original process and the child process

The same program can contain different code for the two possible pids

Process Termination

- Important for resources to be released and cleaned-up
- Different ways of process termination
  - Terminate normally (program ends after last statement)
  - Signaled termination (calling exit system call)
  - Serious errors happened (terminated by OS)
  - Terminated by another process (child process)

introduction to process scheduling

Process Scheduling

- Process scheduling is the algorithm or method of selecting a process for using resources
  - Two types of resources: Memory and CPU
    - Main Memory: Storing program code and data
    - CPU: Executing the program code
  - Selecting a process for CPU and Memory are two different issues
    - Scheduling for Memory : Long-term scheduling
    - Scheduling for CPU: Short-term or CPU scheduling

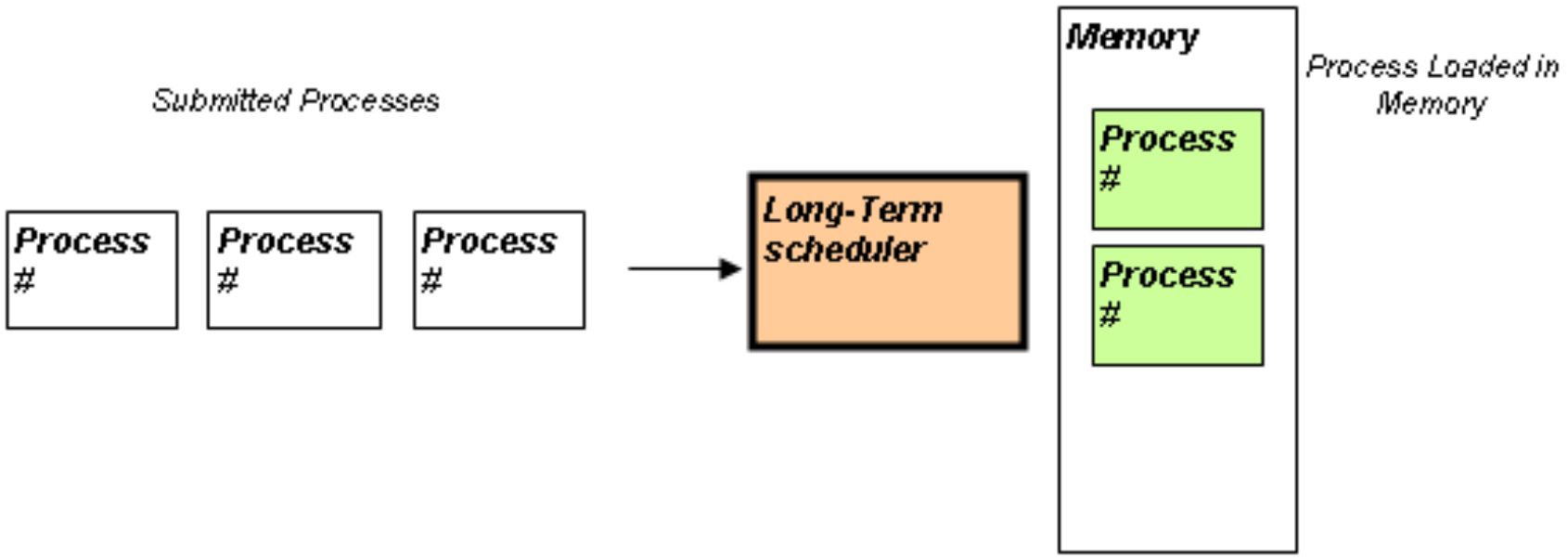
Two Types of Process Schedulers

- Long-term scheduling controls the number of processes admitted into the ready queue
  - Size of the main memory
  - Desired level of multi-programming
  - Desired system resource utilization
- Short-term scheduling
  - Also called CPU scheduling
  - Allocate CPU to one of the ready processes

Long Term Scheduler

- Two operations
  - Select processes from the submitted processes
  - Load them into the main memory
- Considerations
  - Available space of the main memory
  - User experience with desired level of multi-programming
  - Utilization of limited resources

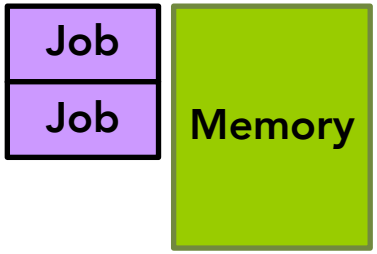
Long-Term Scheduler



Approaches of Long-Term Scheduler

Admit fewer processes – maybe due to smaller main memory

Long-Term Scheduler



Jobs in the Ready Queue

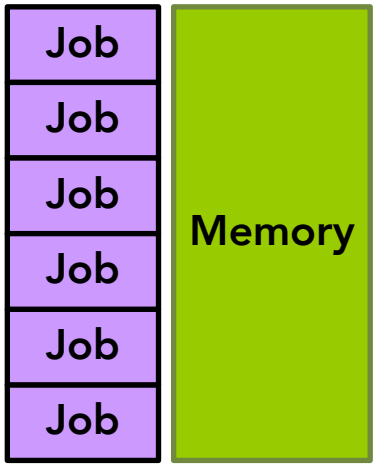
CPU

Short-Term Scheduler

Lower CPU utilization

Admit many processes – the main memory can take them

Long-Term Scheduler



Higher CPU utilization

CPU

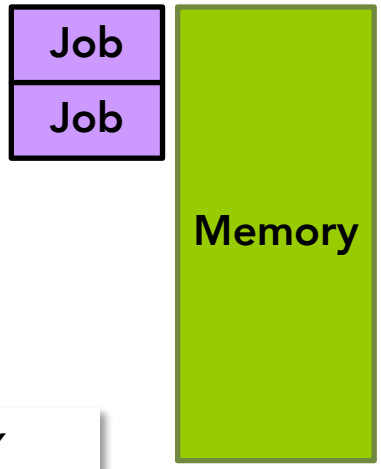
Short-Term Scheduler

Approaches of Long-Term Scheduler

Admit too few processes

Long-Term Scheduler

Few Jobs in the Ready Queue



CPU

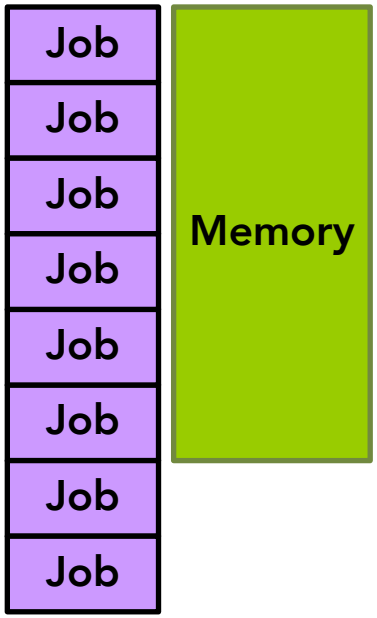
Short-Term Scheduler

CPU is often idle

Approaches of Long-Term Scheduler

Admit too many processes

Long-Term Scheduler



Many Jobs in the Ready Queue

CPU

Short-Term Scheduler

Either

Jobs have to wait for a long time

OR

CPU has to context switch frequently – lower CPU utilization

Suitable Level of Multi-Programming

- The level of multi-programming is the number of processes that are ready for execution in a computer system
- Finding the right level of processes is important
- Memory size is an important factor
  - The number of processes that can be loaded
  - The number of processes in the ready queue

Example: Level of Multi-Programming

Example 6: Number of Processes in the Ready Queue

Given that the size of the main memory of a computer system is 1M. The OS occupied 350 K of the main memory. Now, there are a number of processes to be created and each process requires 100 K main memory. Estimate the maximum number of processes that can be admitted to the ready queue.

Answer:

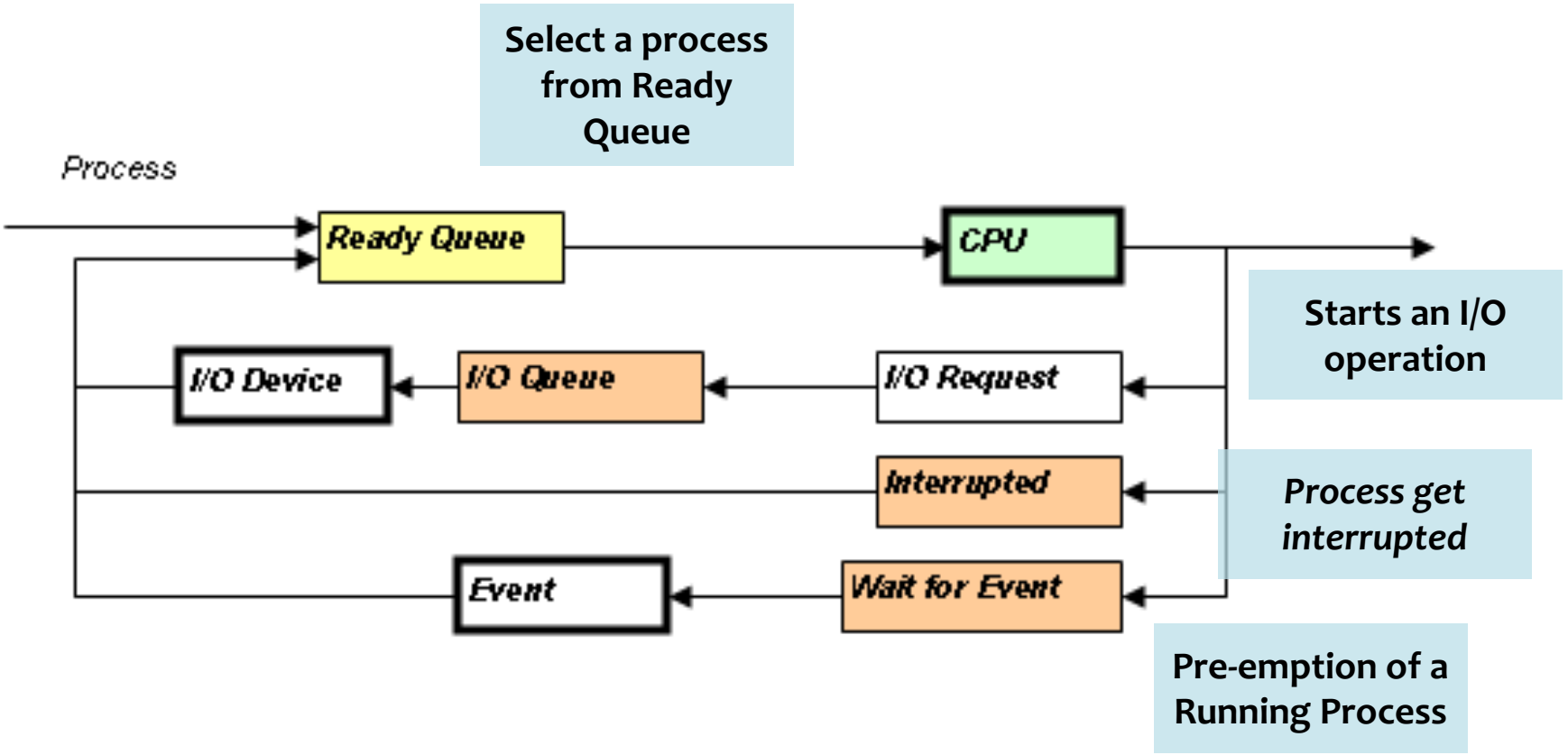
Remaining free memory space = 1M = 1000K – 350K = 650 K

Maximum number of process that can be loaded into the main memory = 650 K / 100K = 6

Short-Term Scheduling

- The Short-term scheduler is built-in with an algorithm
  - Allocate CPU to a process selected from the ready queue
  - Whether to pre-empt a process using the CPU
  - Determines the frequency of process switching
- Factors
  - The OS allows all processes to complete the execution
    - Minimizing context switching
    - Reducing response time
  - The OS switches the processes very frequency
    - Increasing context switching and reducing CPU utilization
    - Improving response time

Short Term Scheduler



multi-threading

Thread of Execution

- An abstract concept describes progress of execution in a program

```
period = int(input("Enter the number of months to analyse: "))
count = 1
total = 0

while count <= period:
    sales = float(input("Enter sales in month #{}: ".format(count)))
    total = total + sales
    count = count + 1

print("Total sales = $", total)
print("Average monthly sales = $", total / period)
```

Thread of Execution

The thread starts running from beginning

```
period = int(input("Enter the number of months to analyse: "))
count = 1
total = 0

while count <= period:
    sales = float(input("Enter sales in month #{}: ".format(count)))
    total = total + sales
    count = count + 1

print("Total sales = $", total)
print("Average monthly sales = $", total / period)
```

The thread finishes after the last statement

Problem with A Single Thread

- Assumed one execution thread per process
- Cause poor user experience
  - Each thread is a worker
  - A worker does one thing at a time
  - While doing one thing, cannot respond to other orders
- Multi-threading is needed for better user experience

Problem with A Single Thread

- The UI has one thread running and waiting for user interactions

The UI thread is running and waiting

The UI thread detected button press

The UI thread carries out spell checking

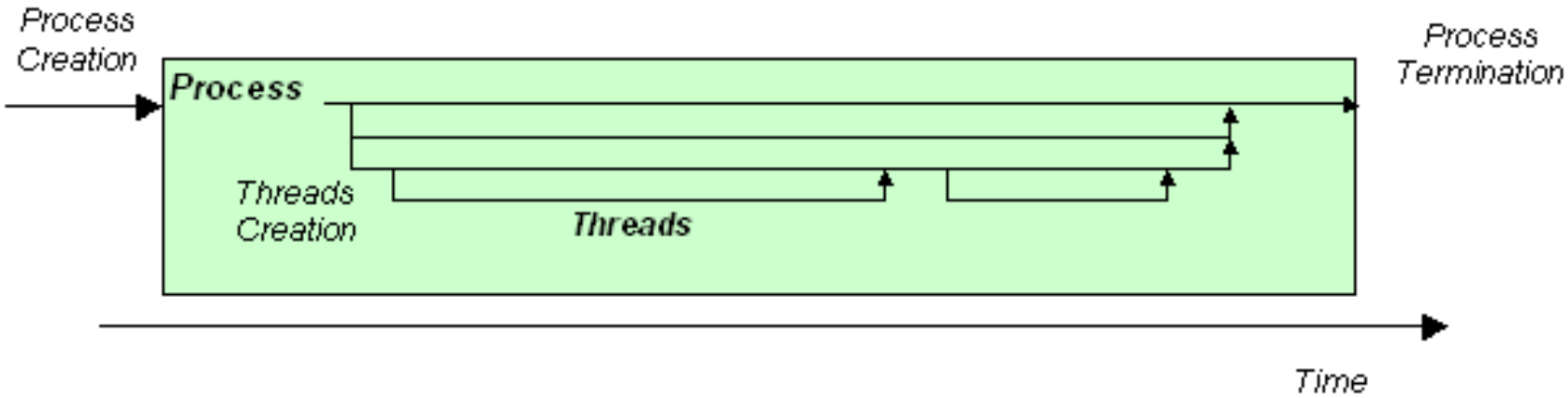
No thread is available for detecting other button press during this period

Threads

- A thread is a light-weight process
  - It belongs to a process
  - Several threads are sharing resources of a process
  - It requires less amount of resources
  - It has its own Program Counter
  - It keeps its own register set, run-time stack, and state information

Threads

- A single process has multiple threads
  - Sharing the same address space
  - Sharing the same global variables
  - Sharing system resources



Threads

- What is implication threads belong to same process?
  - Threads can access data of other threads
  - Threads are assumed to be friendly to each other
    - They belong to the same program

Thread Implementation

- Which thread should run in a multithreading program?
  - Same as the situation of multiple processes
  - Scheduling is required
- How threading is implemented?
  - User level threads
  - Kernel level threads

Types of Threads

- User-level threads
  - The OS kernels know only the processes, not the threads.
  - A thread can execute only when its parent process is selected for execution.
  - Context switching does not involve the OS kernel.
- Kernel-level threads
  - Directly managed by the OS
  - The OS kernels know all the threads, and threads, instead of processes, are the unit of selection

Comparison of Thread and Process

- Switching between threads is more efficiency than switching between processes
  - There are less resource types to deal with in context switching

Each Process has	Each Thread has
Program counter	Program counter
Stack	Stack
Register set	Register set
Child thread	Child thread
State	State
Address space	
Global variables	
Open Files	
Timers	
Semaphores	
Accounting Information	

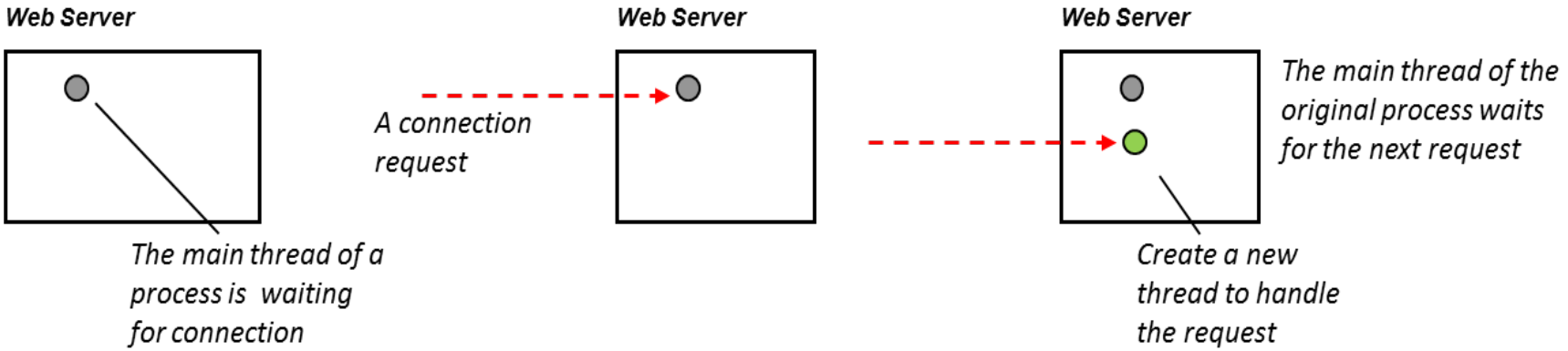
A Summary of All Resource Types

- A process is associated with many resource types
  - Creation takes significant management overhead
  - It is considered heavy-weight

Resources	***Descriptions***
Stack	For keeping local variables, parameters, and return values in function calls
Register set	The data involved in CPU calculations and data manipulations are saved here: accumulators, index registers, stack pointers, condition code (overflow, carry, etc).
Child thread	
State	
Address space	The memory space for the text section, and dynamic memory allocation.
Global variables	The data section of the process
Open Files	Including network connections.
Timers	
Semaphores	For synchronization between threads

Thread Pooling

- Consider a web server
  - Waits and receives thousands of incoming requests
  - Some requests take long time
  - Web servers are multi-threading

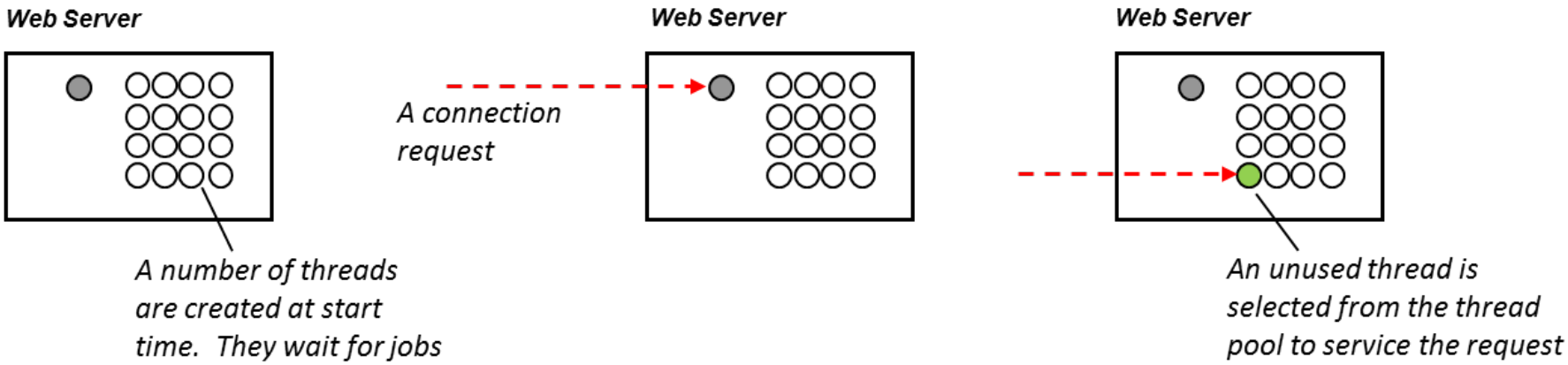


Thread Pooling

- A new thread for each incoming request
  - The new thread can handle the request for a long time
  - The new thread can be destroyed when the request is completed
  - The original thread can listen to new requests
- Creation and destruction of thread incur overhead
  - Should be avoided

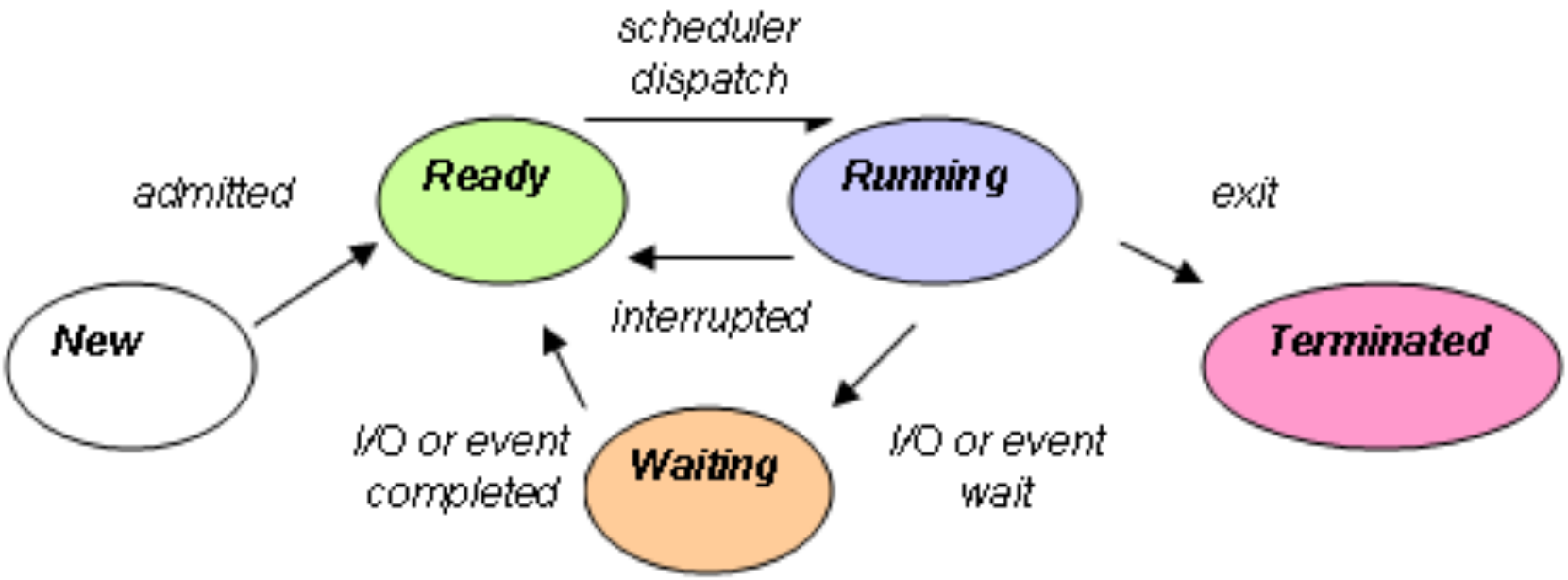
Thread Pooling

- Pooling is a common technique
  - A number of threads are created at the start-up of the web server
  - Gathered in a structure called a thread pool
  - Used threads are called upon to handle new incoming request
  - Finished request the thread returns to the pool



case study: unix process management

The Five-State Model Process Management



UNIX

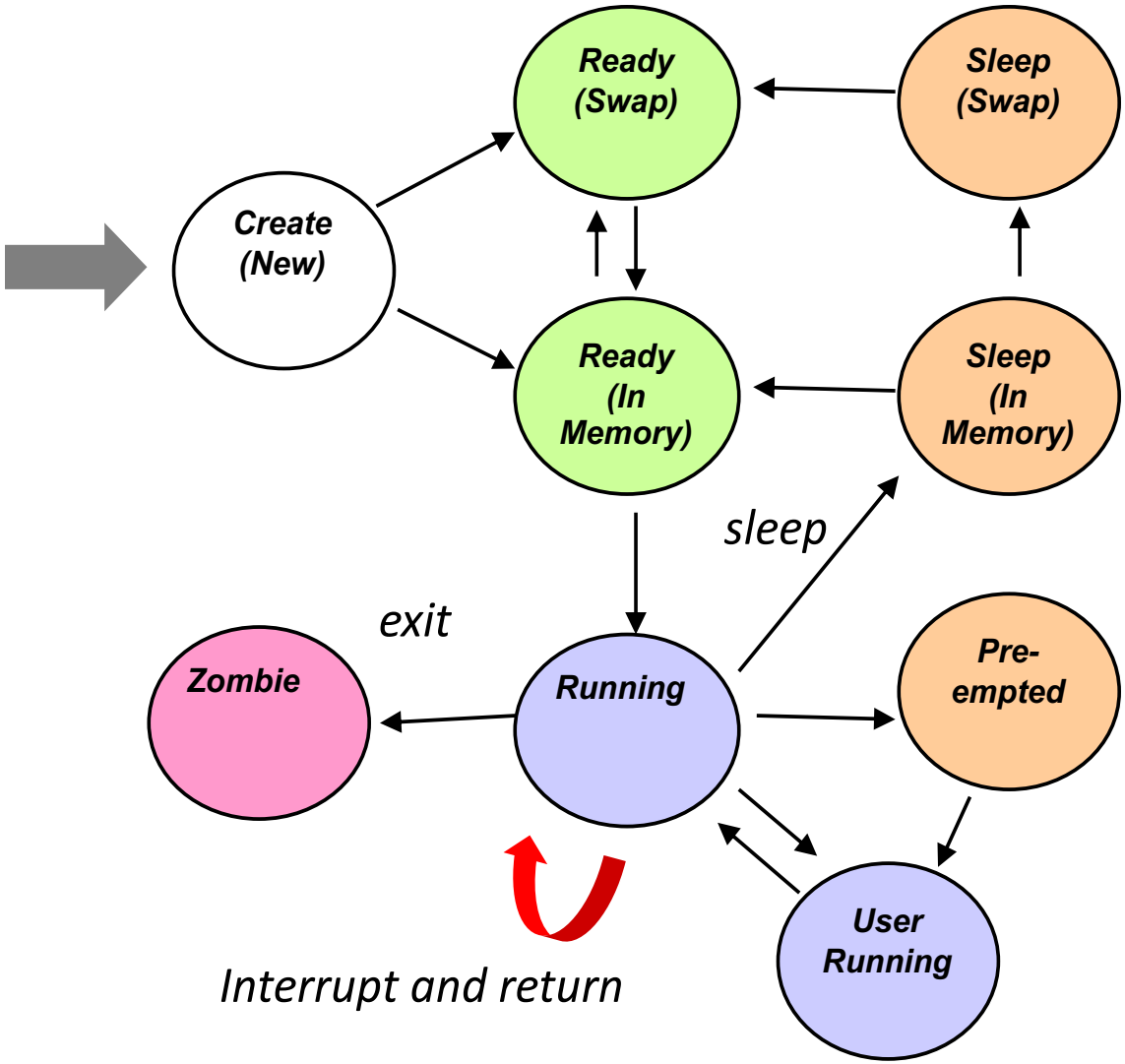
- UNIX is the most important OS family that has influenced modern computing
  - Supported multi-tasking and multi-user
  - The parents of Linux, BSD, Apple MacOS

UNIX SVR4 OS Nine State Model

- UNIX has no long-term scheduler
- The process model has nine states

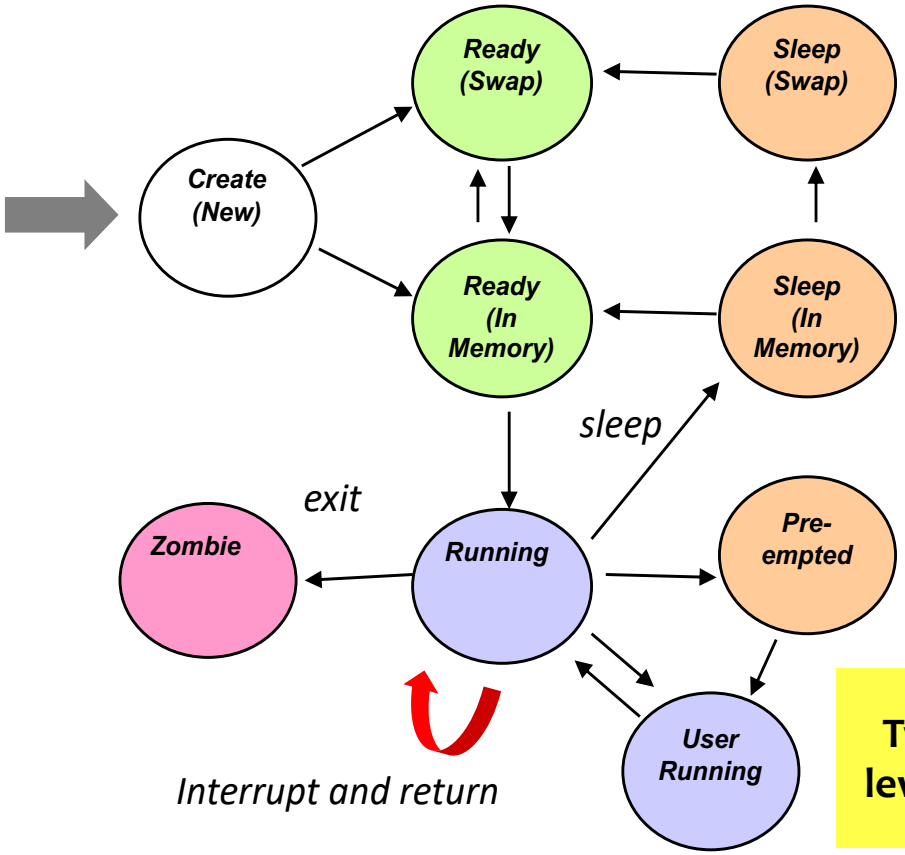
Create	Equivalent to the New state
Ready (In Memory)	Equivalent to the Ready state
Ready (Swap)	Equivalent to the Ready state but the program memory is swapped out to the secondary memory
User Running	Running state in user mode
Kernel Running	Running state in kernel (monitor) mode
Sleep	Equivalent to the Waiting state
Pre-empted	Equivalent to the Waiting state
Zombie	Equivalent to the Terminate state. The process does not exist, but the record remains for the parent process to clean up

UNIX SVR4 OS Nine State Model



UNIX SVR4 OS Nine State Model

Process may be swapped to the hard disk, unloaded from main memory



Two Running states for user level or kernel level execution

Acknowledgement

- Images and Graphics used in this set of slides
  - Designed by Photoroyalty - Freepik.com