Tishk International University
Science Faculty
IT Department

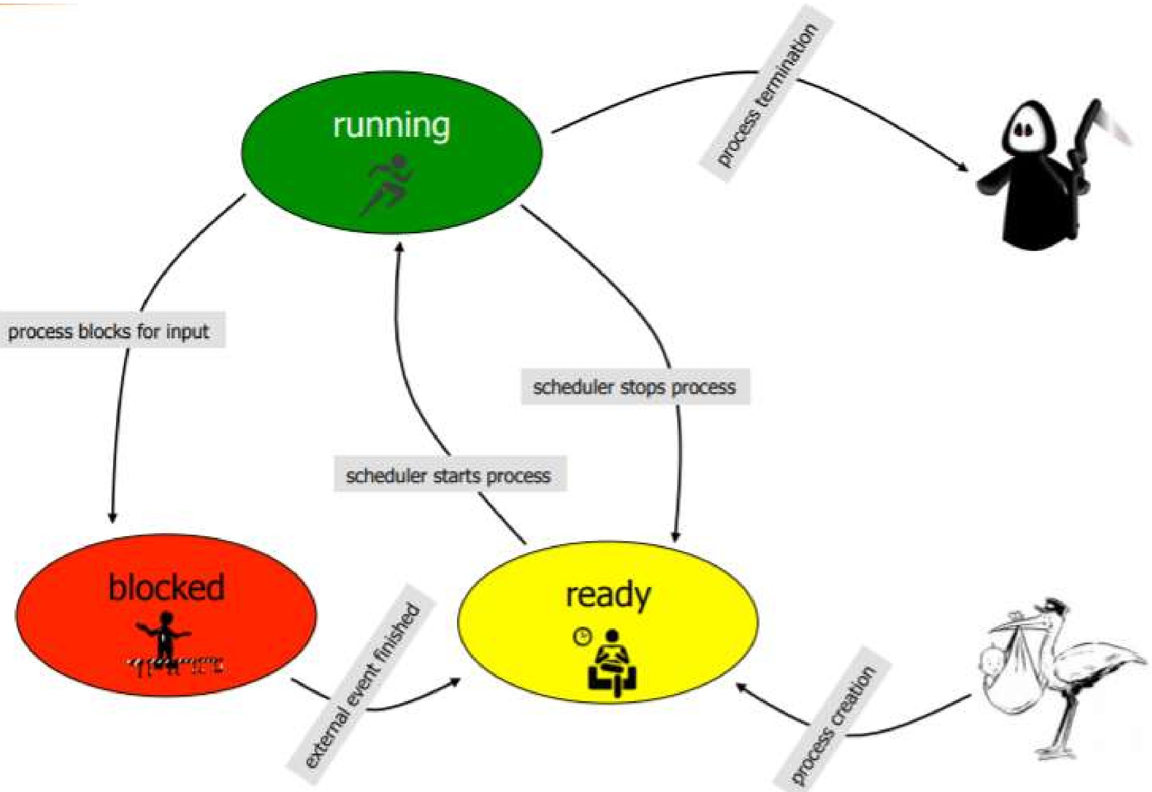
Operating Systems

Lecture 2: Processes and Threads

3rd Grade - Fall Semester

Instructor: Alaa Ghazi

Lecture 2: Processes & Threads

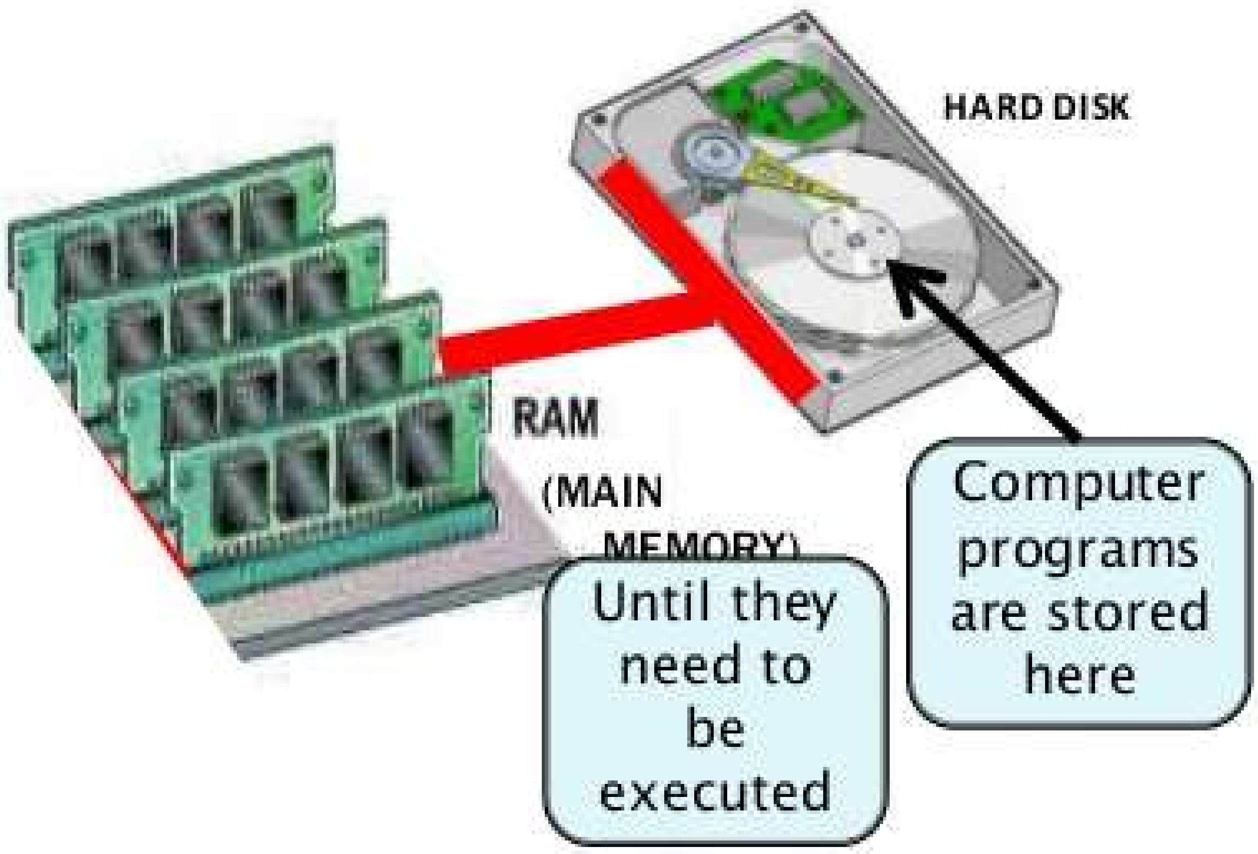


Agenda

General Definition of a Process?

A process is an activity, or series of activities, that converts an input, to an output, by doing work.

PART 1
Processes

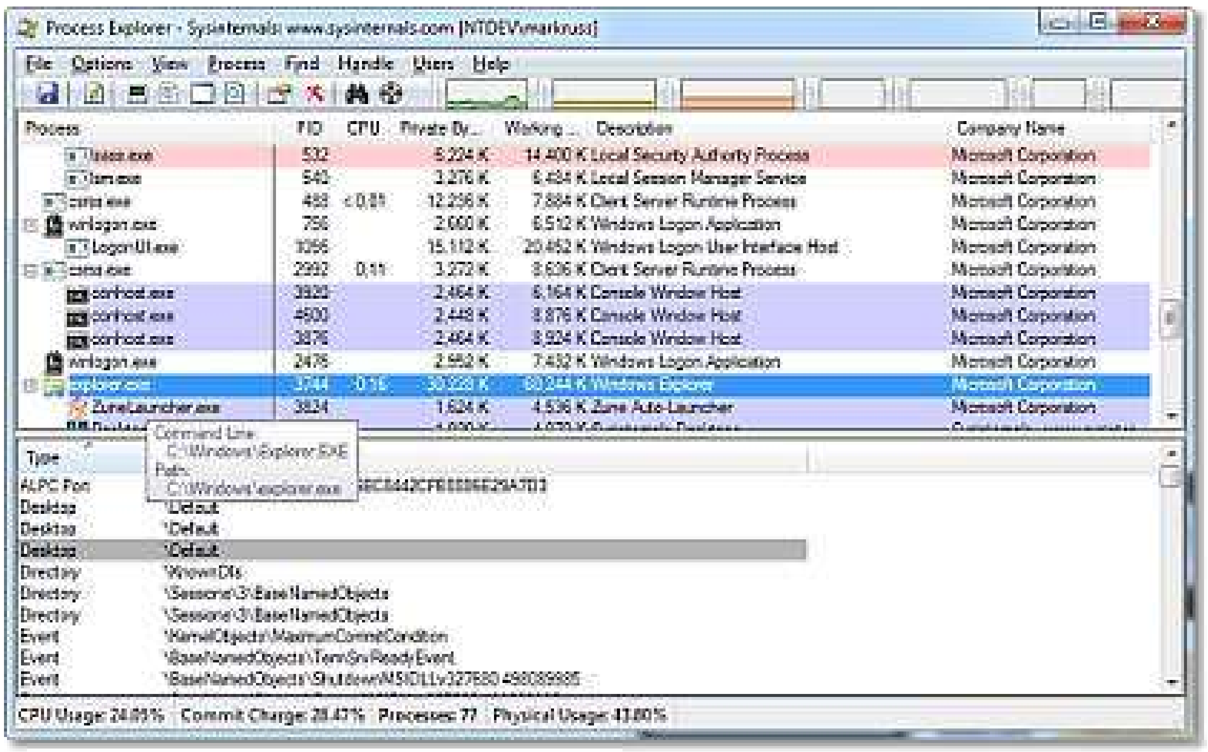


3.1 Process Concept in OS

- **Process** – is an active program in execution; process execution must progress in sequential fashion
- Program is a ***passive*** entity stored on disk (**executable file**), and it becomes a process when executable file is loaded into memory
- One program can be executed multiple times generating multiple processes

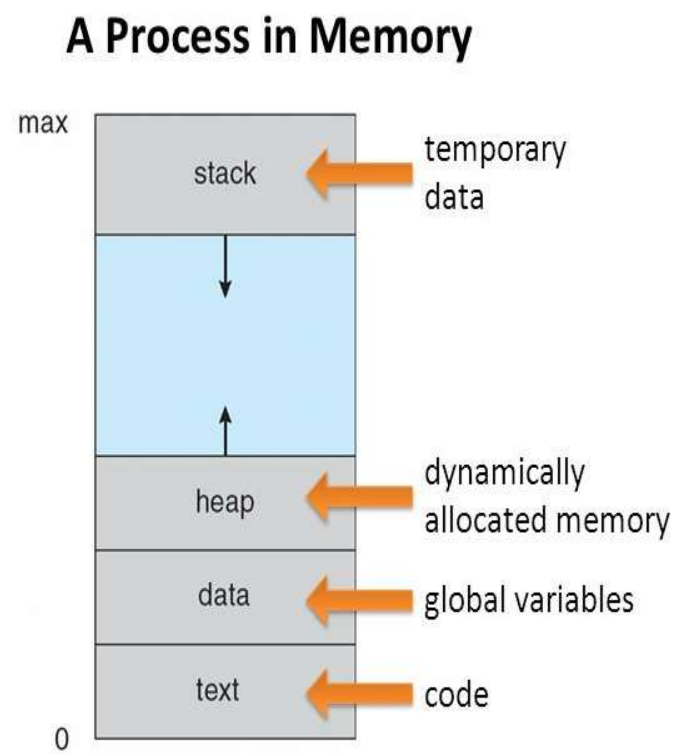
Practical Case
Process Explorer (Freeware)

- It is a freeware shows a list of the currently active processes,
- It is useful for tracking down problems and provide insight into the way Windows and applications work.



3.2 Process Parts

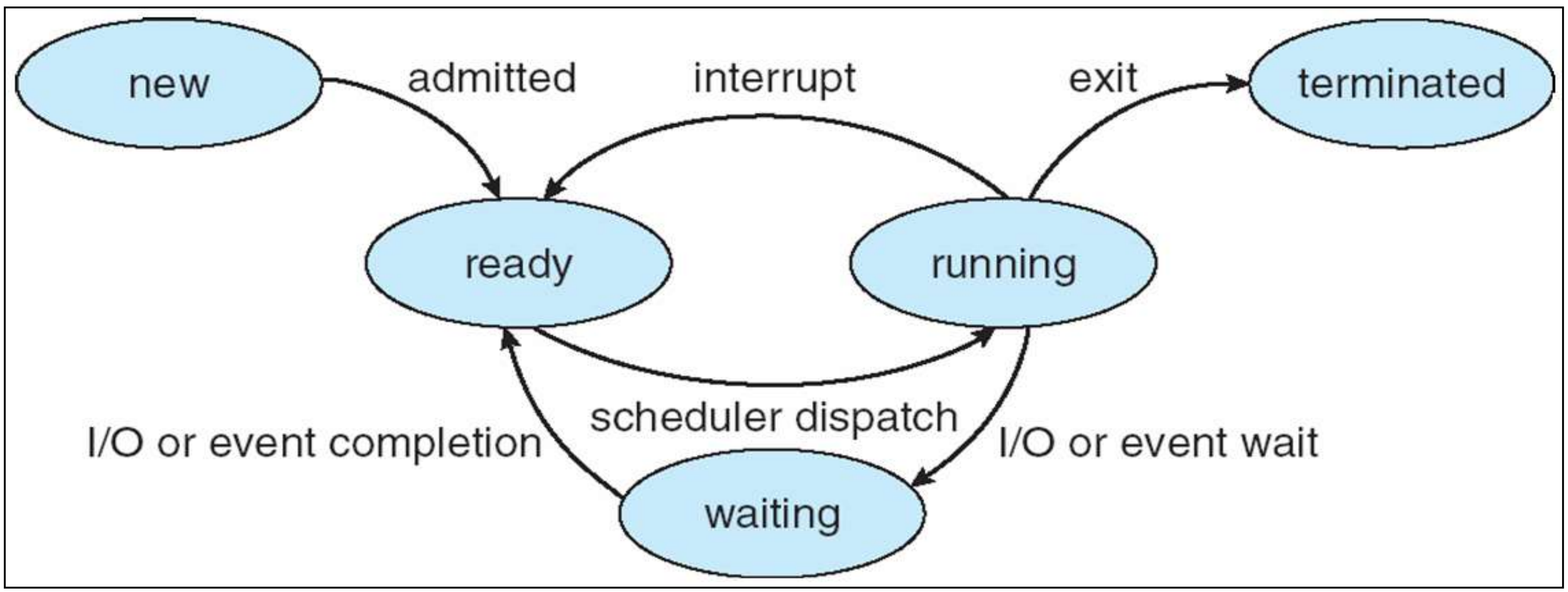
- **Text Section** contains the program code.
- **Data section** contains the global variables
- **Stack** contains function parameters, return addresses, and local variables.
- **Heap** contains dynamically allocated memory during run time.



3.3 Process States

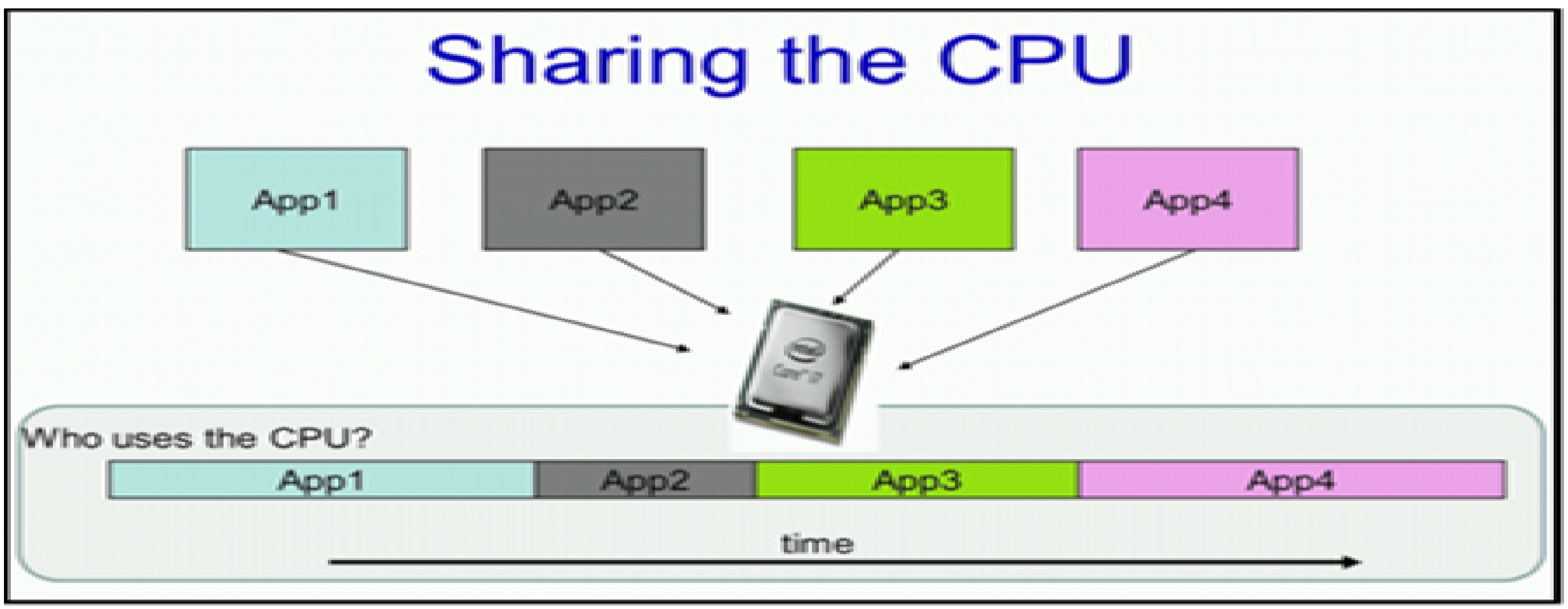
As a process executes, it changes **state** to below five states

- **new**: The process is being created
- **running**: Instructions are being executed
- **waiting**: The process is waiting for some event to occur
- **ready**: The process is waiting to be assigned to a processor
- **terminated**: The process has finished execution



Practical Analogy: CPU vs. Doctor (not required in the exam)

Patient 1 Patient 2 Patient 3



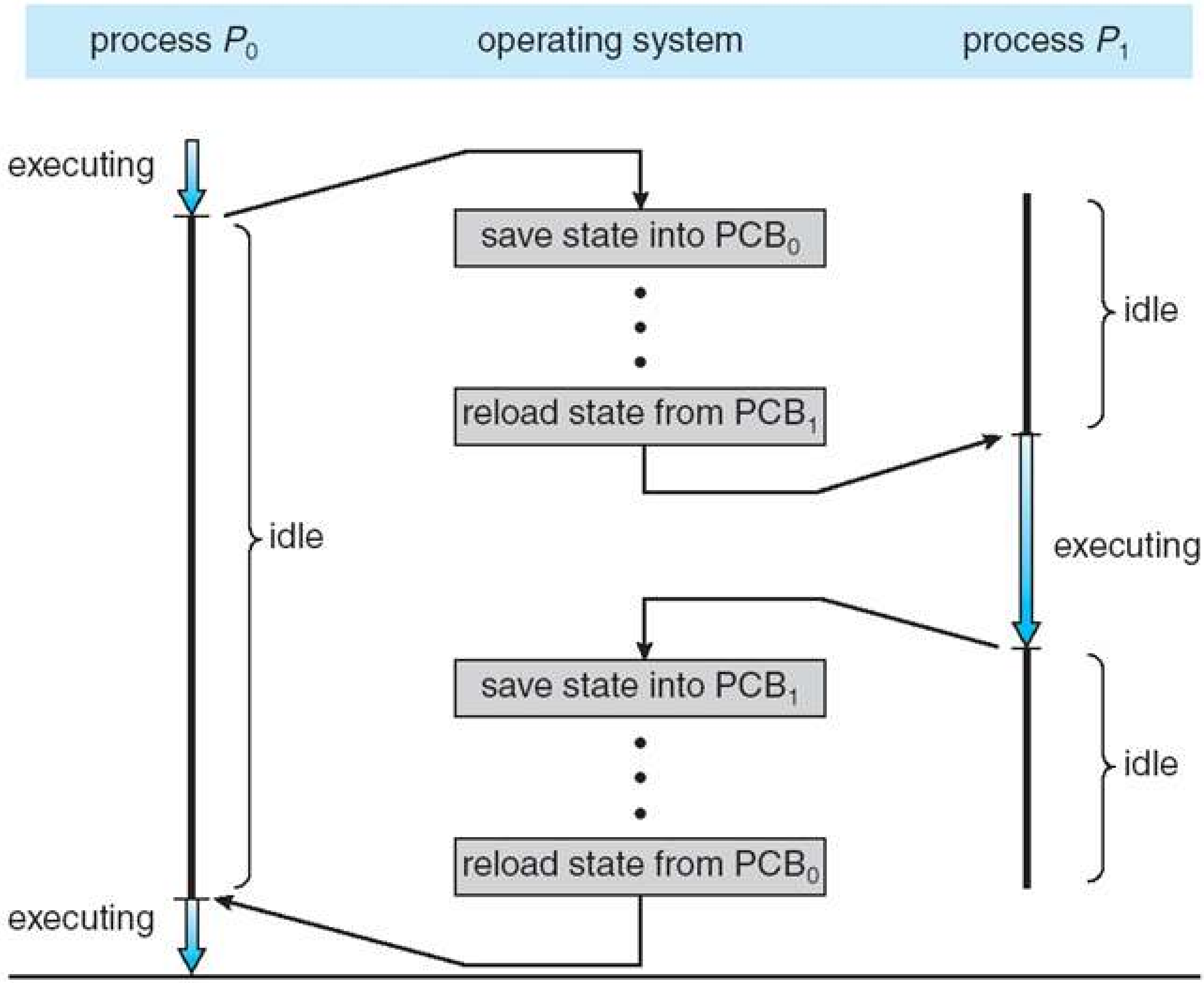
3.4 Process Control Block (PCB)

PCB is the Information associated with each process, which includes:

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Analogy
PCB = Patient Information

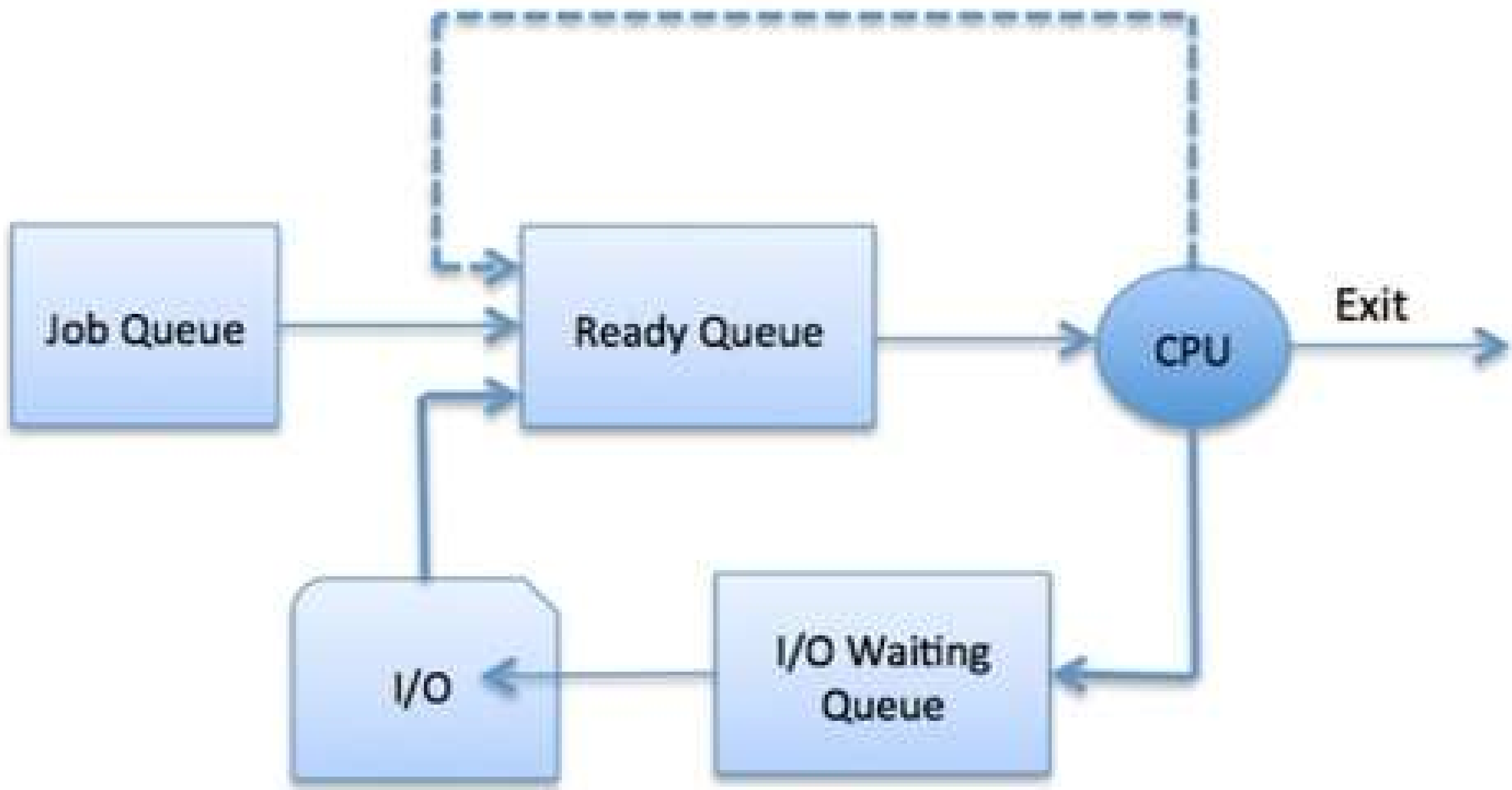
3.5 CPU Switch From Process to Process (not required in the exam)



3.6 Process Scheduling Queues

- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

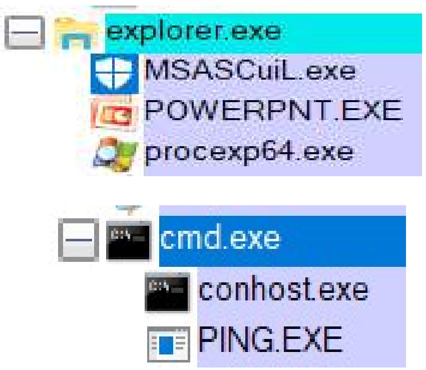
Process Scheduling Queues Diagram



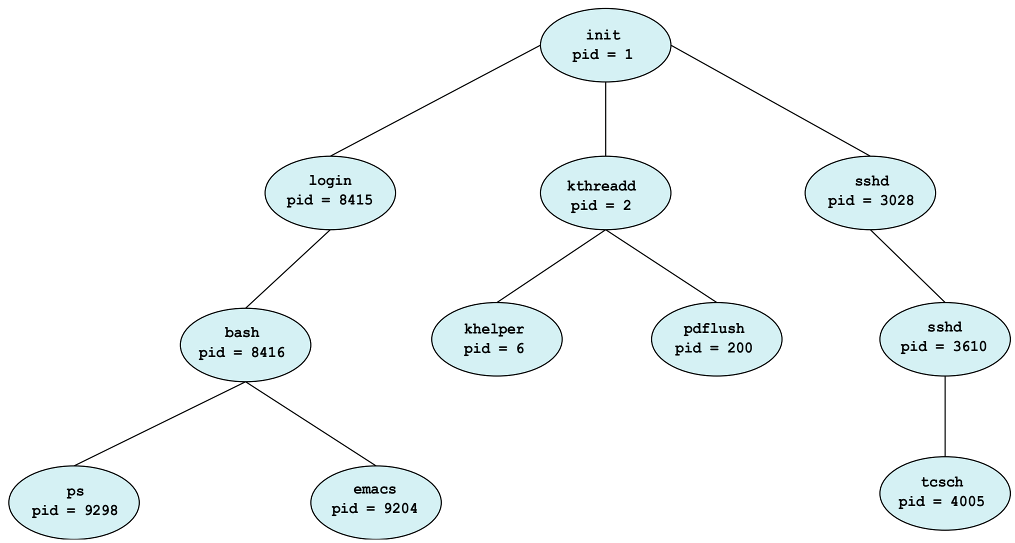
3.7 Operations on Processes

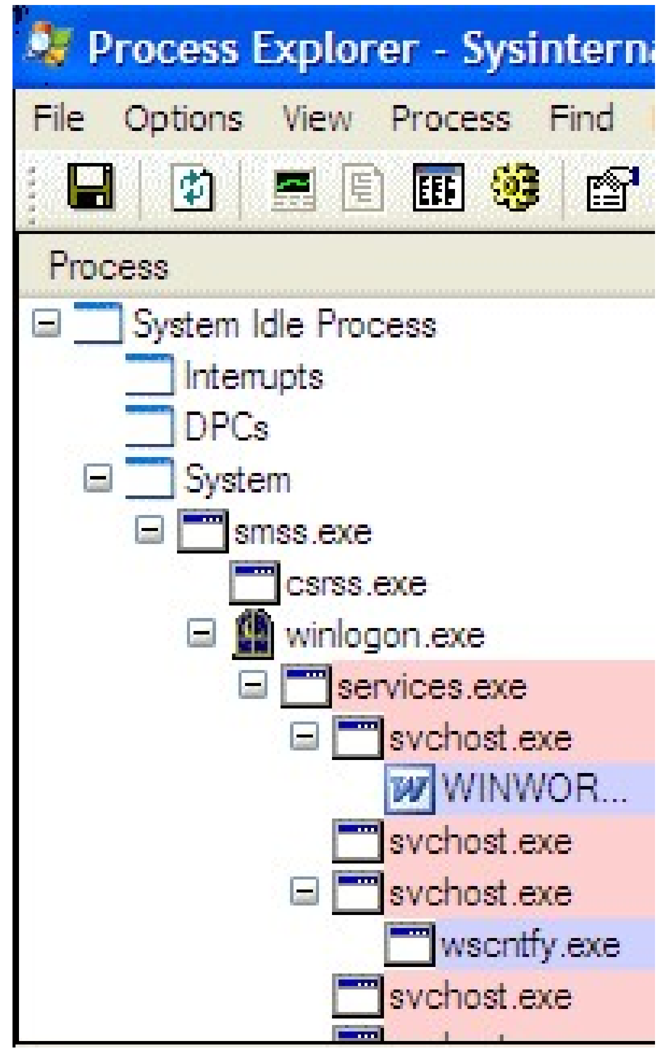
1) Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier** (**pid**)
- Resource sharing options:
  1. Parent and children share all resources
  2. Children share subset of parent's resources
  3. Parent and child share no resources
- Execution options:
  1. Parent and children execute concurrently
  2. Parent waits until children terminate



Examples of Tree of Processes (not required in the exam)





2. Process Termination

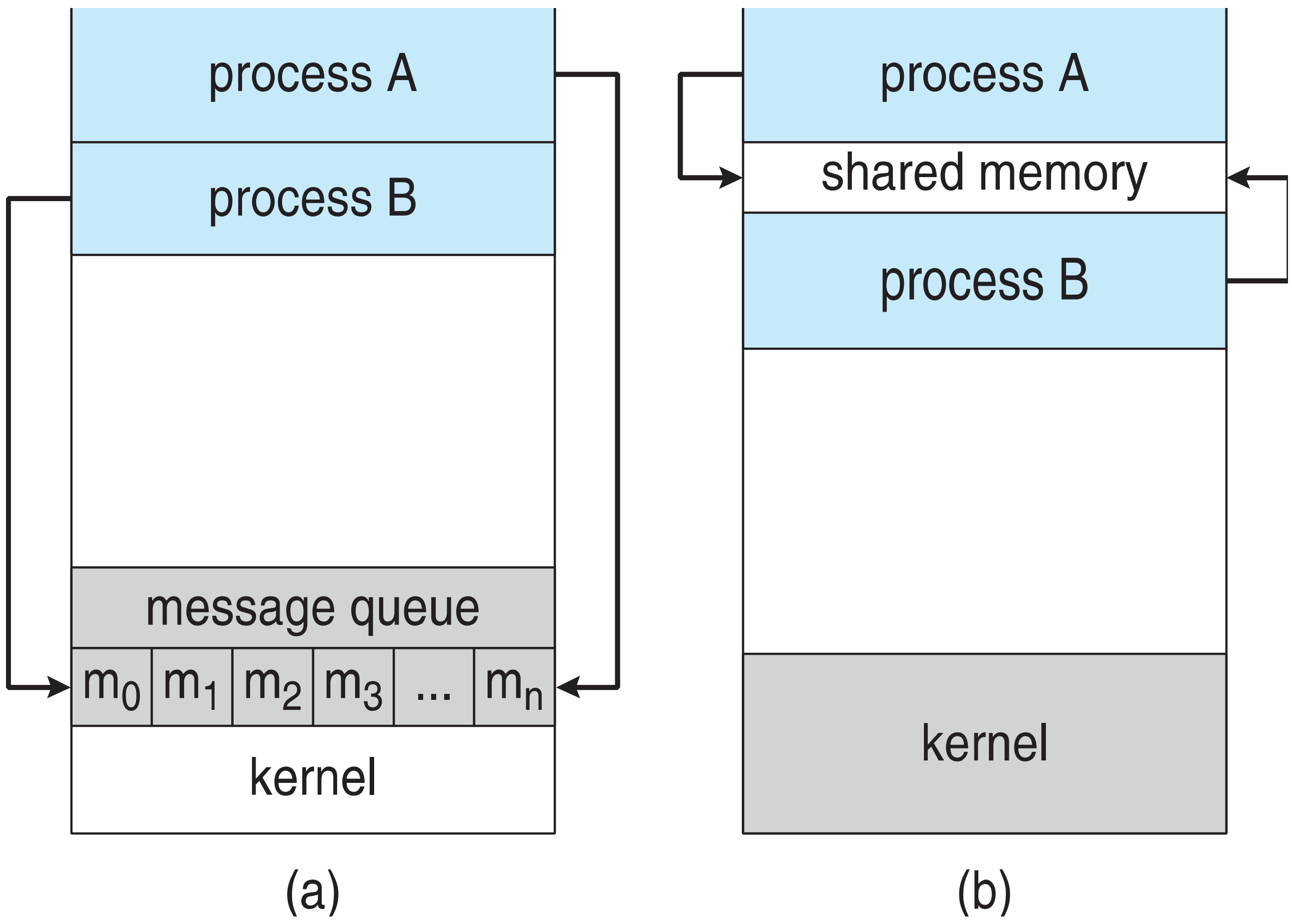
- Process executes last statement and then asks the operating system to delete it using a system call.
  - Returns status data from child to parent
  - Process' resources are de-allocated by operating system
- Parent may terminate the execution of children processes using a system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting

3.8 Inter-process Communication

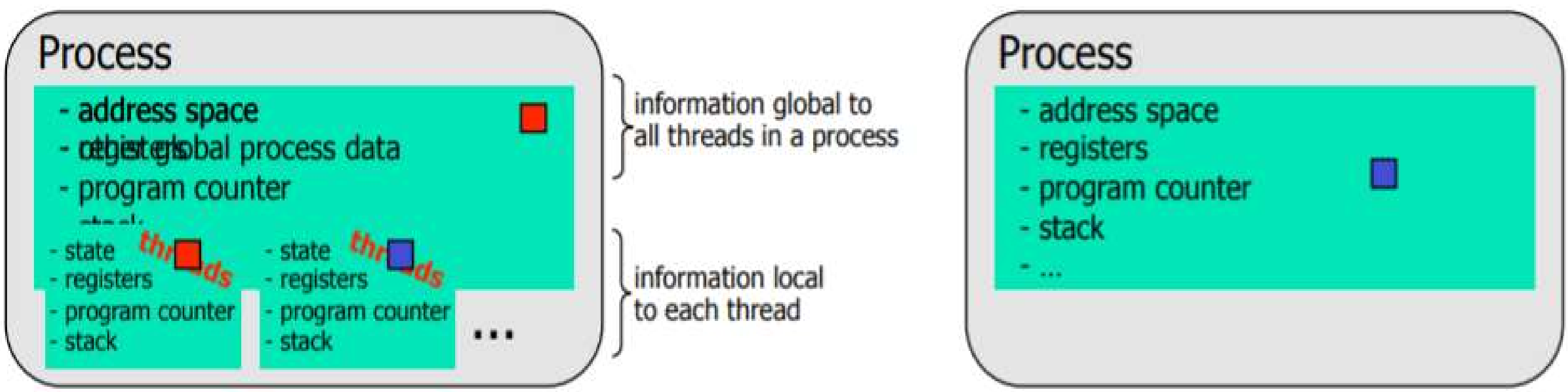
- Processes within a system may be ***independent*** or ***cooperating*** when they need to share data
- There are two models of **inter-process communications** (**IPC**)
  - **Shared memory**
  - **Message passing**

Inter-Process Communications Models

(a) Message passing. (b) shared memory.



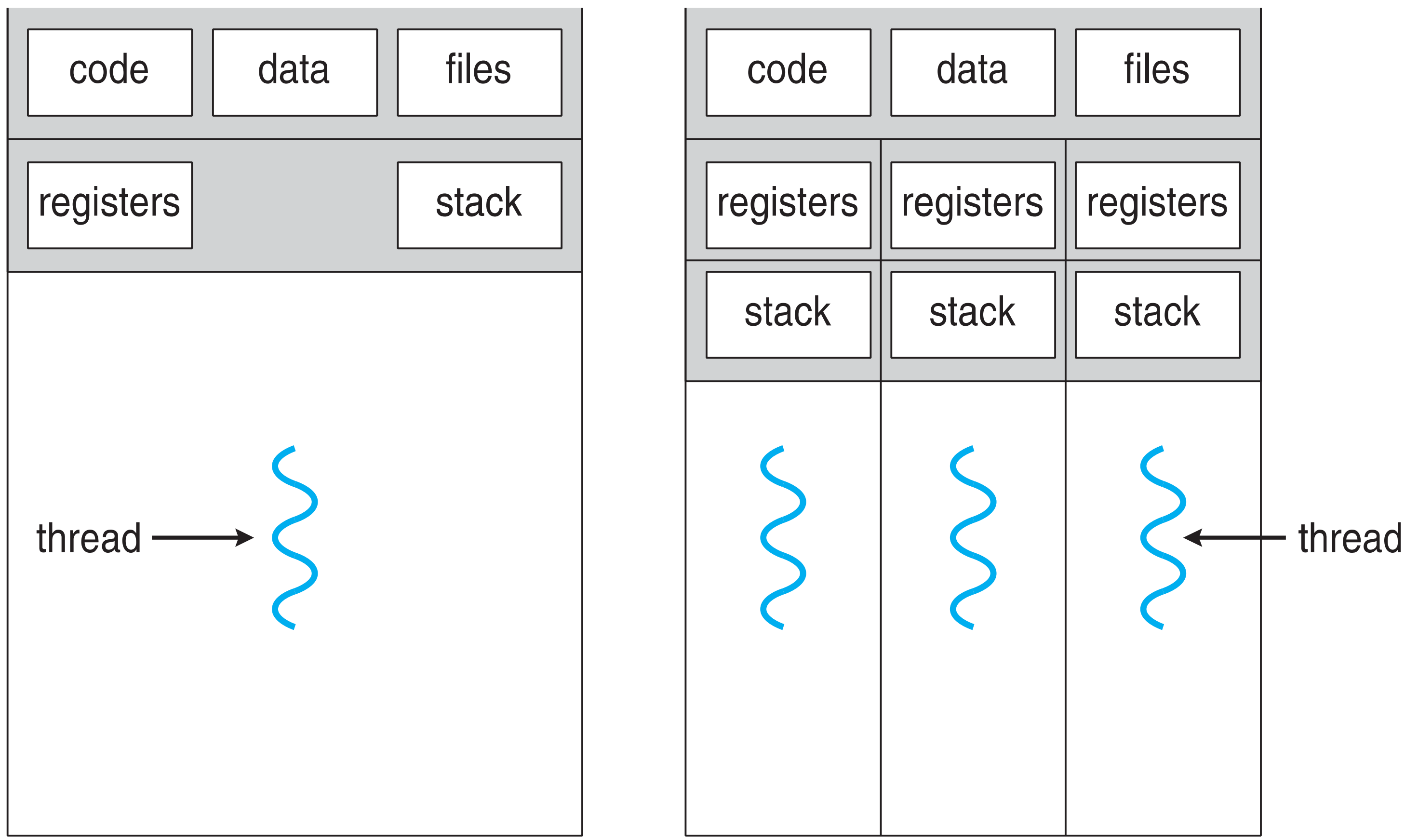
PART 2
Threads



3.9 Thread Concept

- The thread is a component of the process and it is the smallest sequence of instructions that can be managed by the scheduler
- Multiple threads can exist within one process, executing concurrently and sharing resources such as memory.
- Implicit Threading where the creation and management of threads done by compilers rather than programmers.
- Most modern applications are multithreaded, so tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking

Single and Multithreaded Processes (not required in the exam)

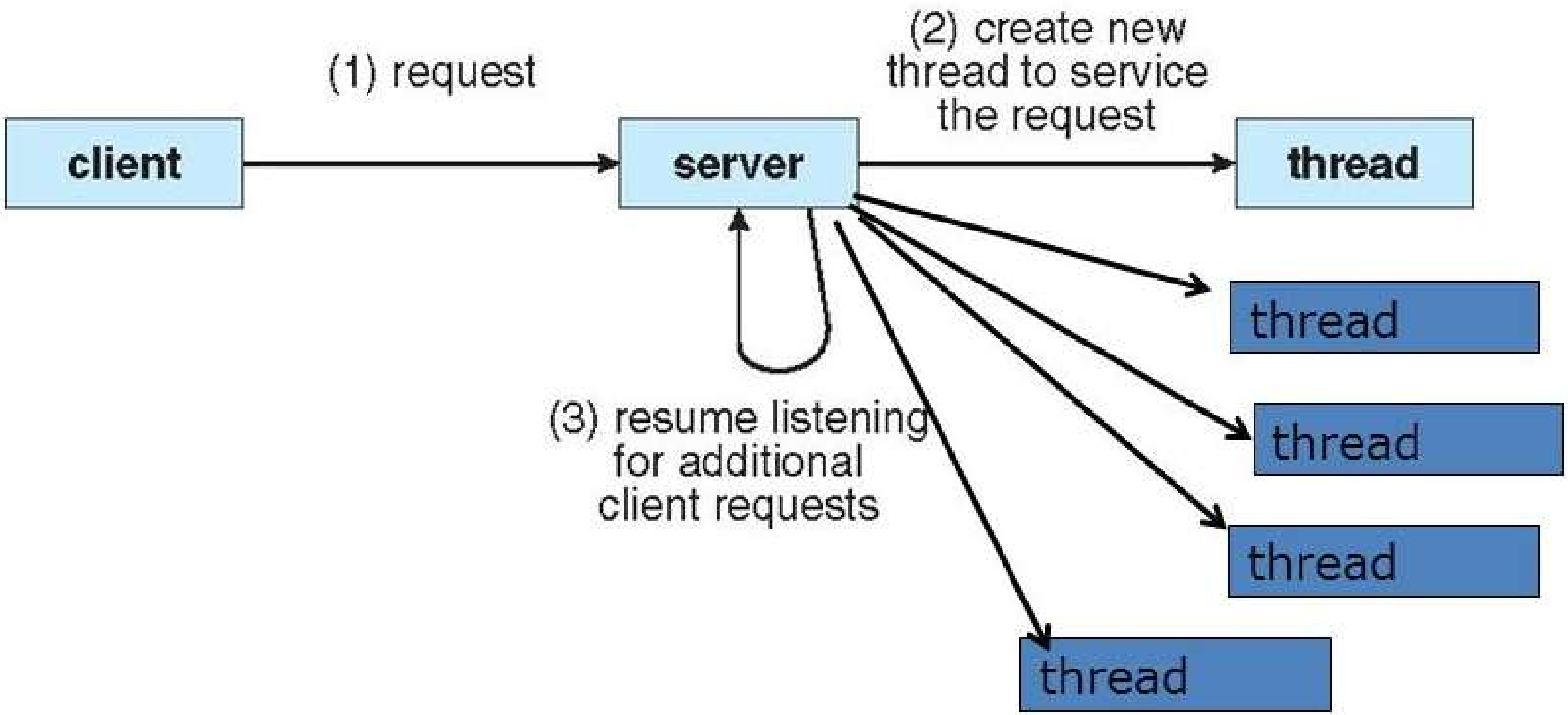


single-threaded process

multithreaded process

Multi-threaded Process Analogy

3.10 Multithreaded Server Architecture Example



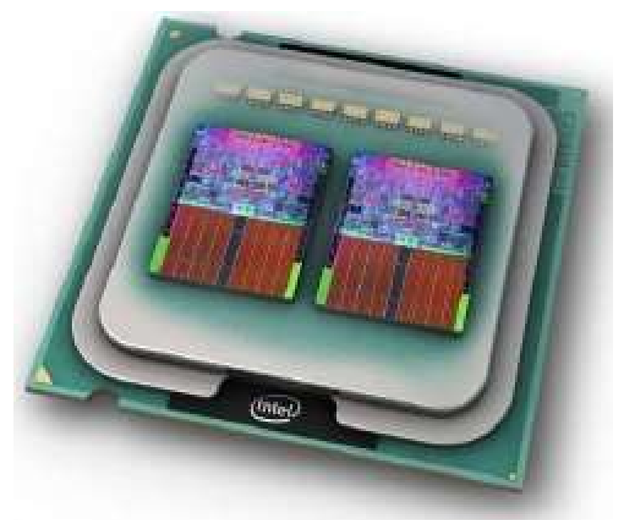
3.11 Multithreading Benefits

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multiprocessor architectures

3.12 Multi-Processor Systems

- The systems can have single processor or multiple processors

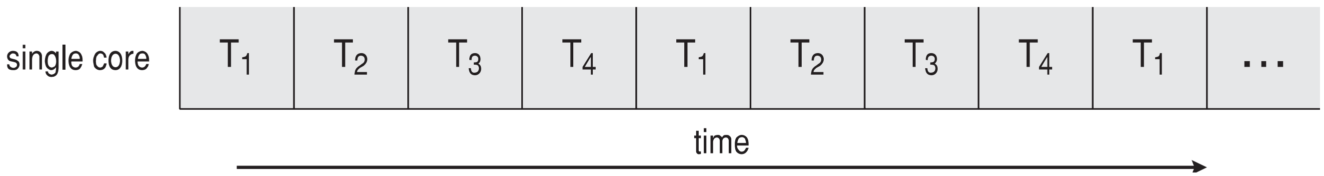
- A system can have independent CPUs in single motherboard



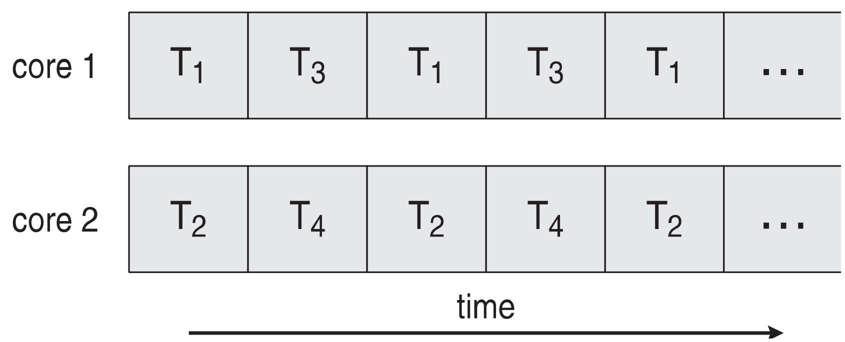
- A <u>multi-core processor</u> is one which combines two or more independent processors into a single chip.

3.13 Concurrency vs. Parallelism

- ***Concurrency*** supports more than one task making progress, this can be implemented by a Single processor / core with a scheduler.



- ***Parallelism*** implies a system can perform more than one task simultaneously on multi-core system



3.14 Amdahl's Law

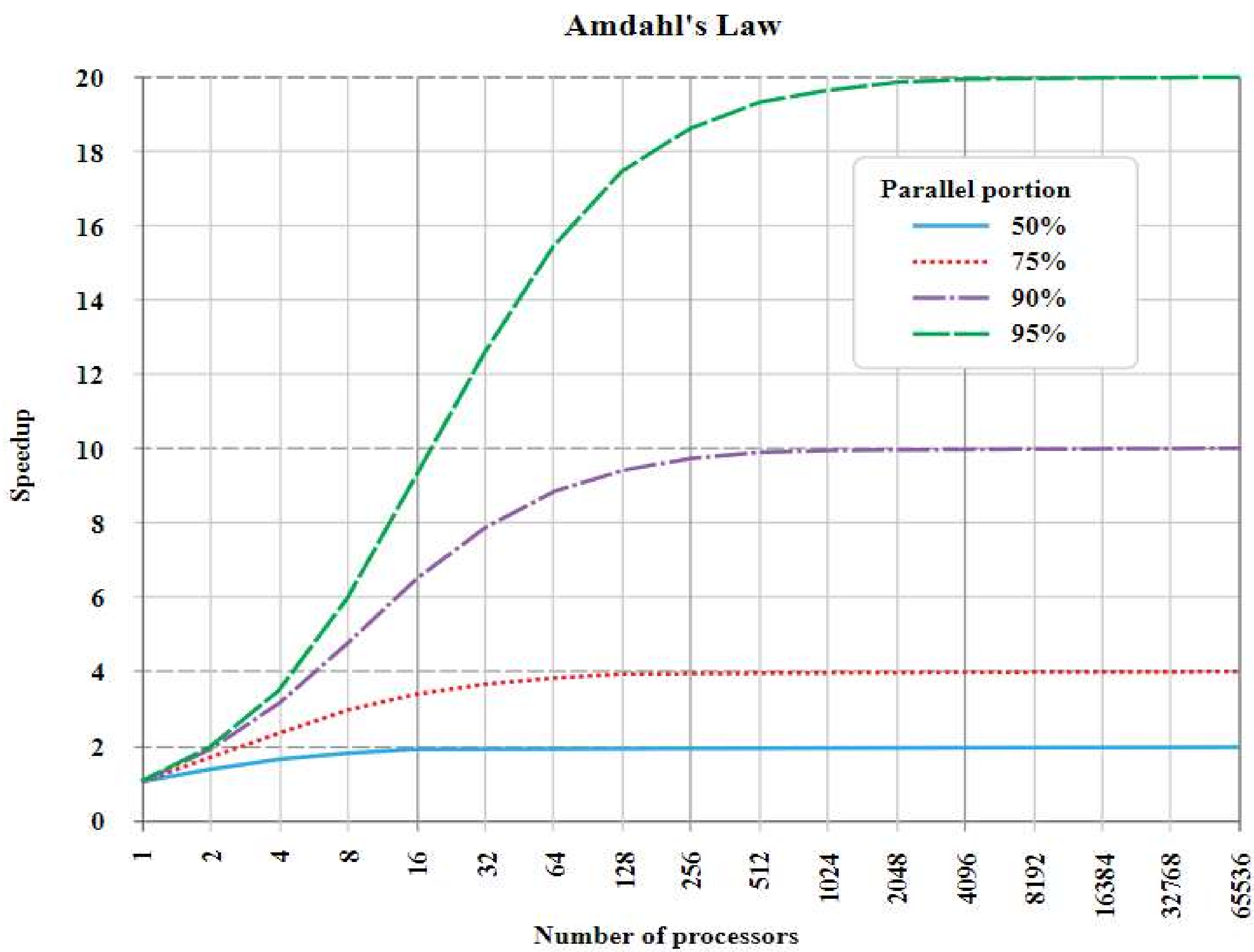
- This law Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- P is parallel portion
- *S* is serial portion

$$speedup = \frac{1}{S + \frac{(1-S)}{N}}$$

S = 1 - P

- *N* processing cores
- As *N* approaches infinity, speedup approaches 1 / *S*
- **Serial portion of an application is limiting the performance gained by adding additional cores**
- Adding more processors leads to successively smaller returns in terms of speedup

Amdahl's Law Graph (not required in the exam)



Amdahl's Law Example

Q\ Using Amdahl's Law, calculate the speed up factor for moving from single processor to four processors with an algorithm that has %80 parallel part.

Answer\ Amdahl's Law state that:

$$speedup = \frac{1}{S + \frac{(1-S)}{N}}$$

N= 4

S = Serial Part

P = Parallel Part

S = 1 – P = 1 - 0.8 = 0.2

$$Speedup = \frac{1}{\left[0.2 + \frac{(1-0.2)}{4}\right]} = 2.5$$