Processes

Process - Concept

An OS executes a variety of programs

- Batch system – jobs
- Time-shared systems – user programs or tasks

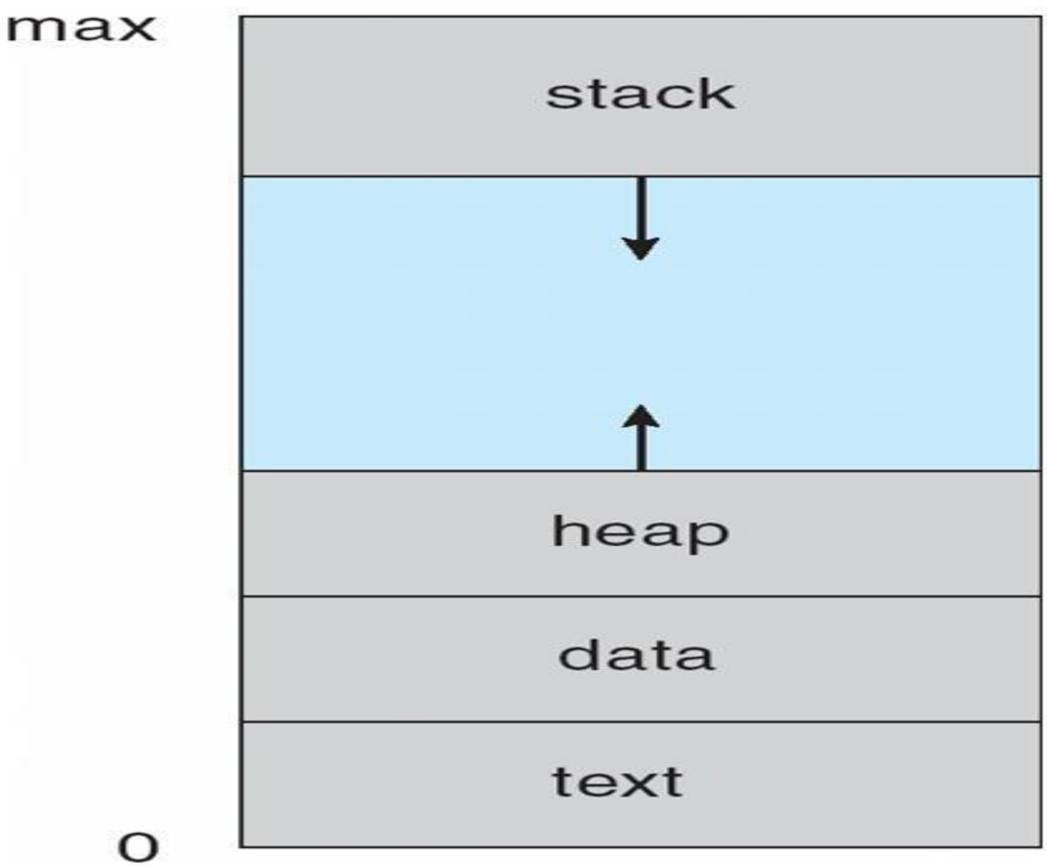
Process

- a program in execution;
- process execution must progress in sequential fashion

A process includes

- program counter
- stack pointers
- data section

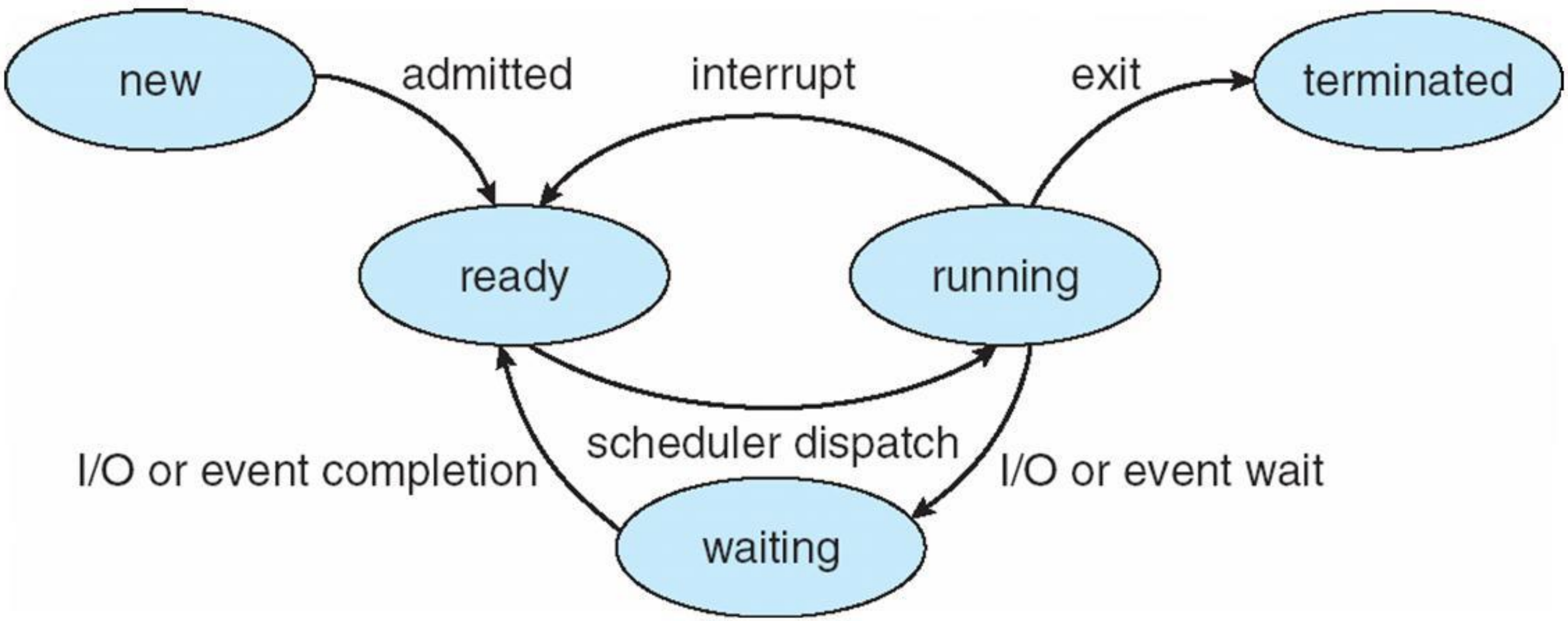
Process in memory



Process 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

Diagram of Process States



Process Control Block

- Information associated with each process

Process state

Program counter

CPU registers

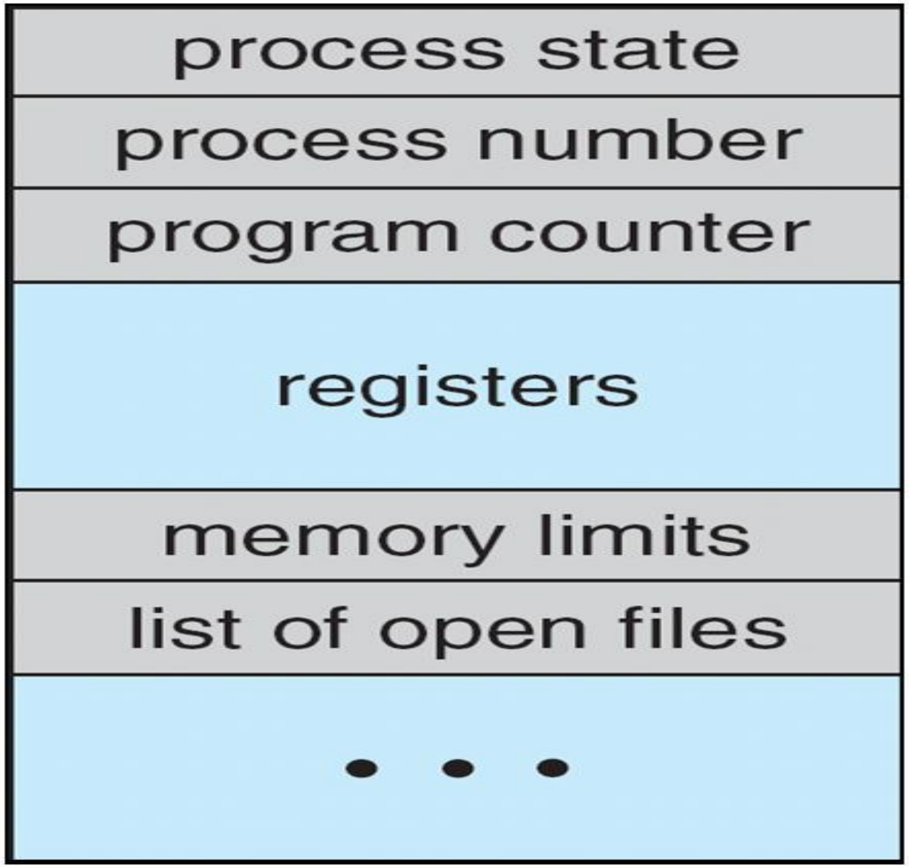
CPU scheduling information

Memory-management information

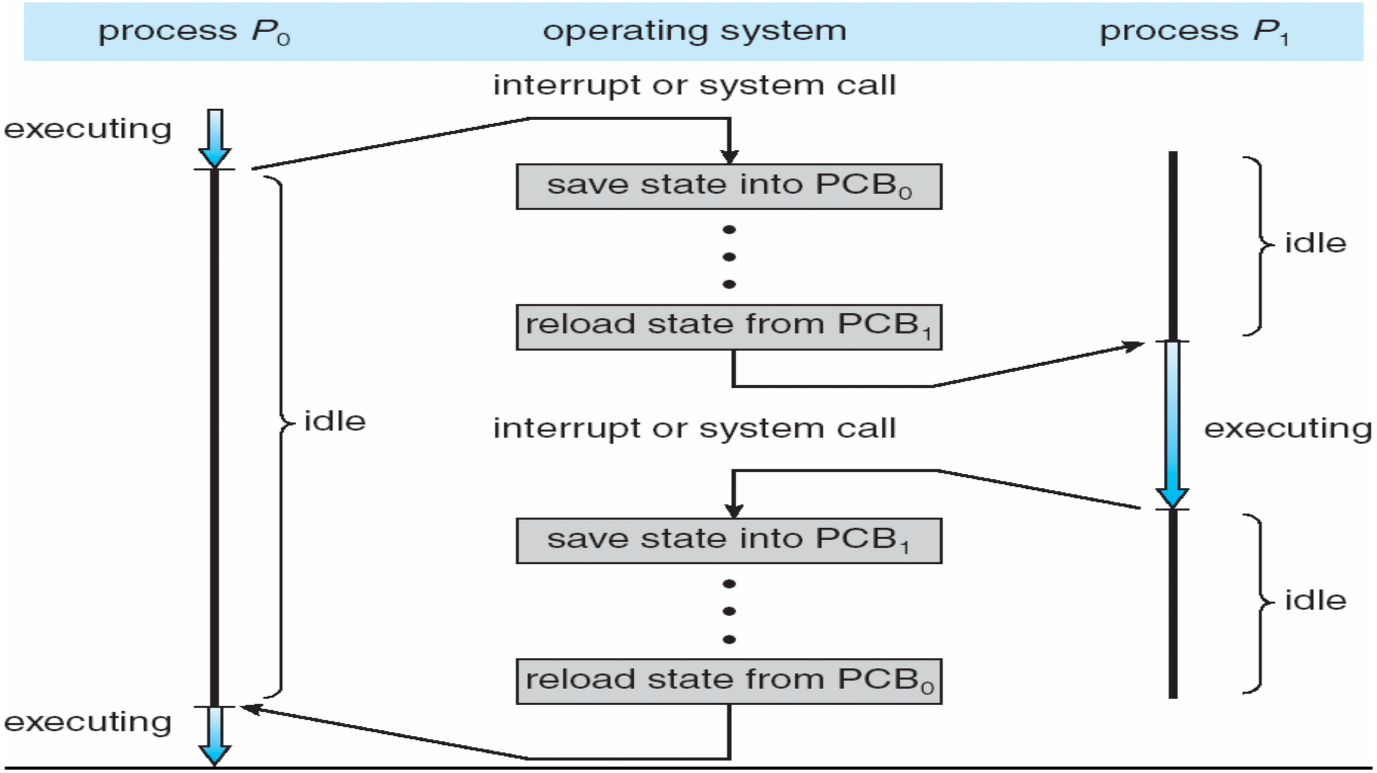
Accounting information

I/O status information

Process Control Block



CPU Switch from Process to Process



Process Scheduling Queue

Job queue

- set of all processes in the system

Ready queue

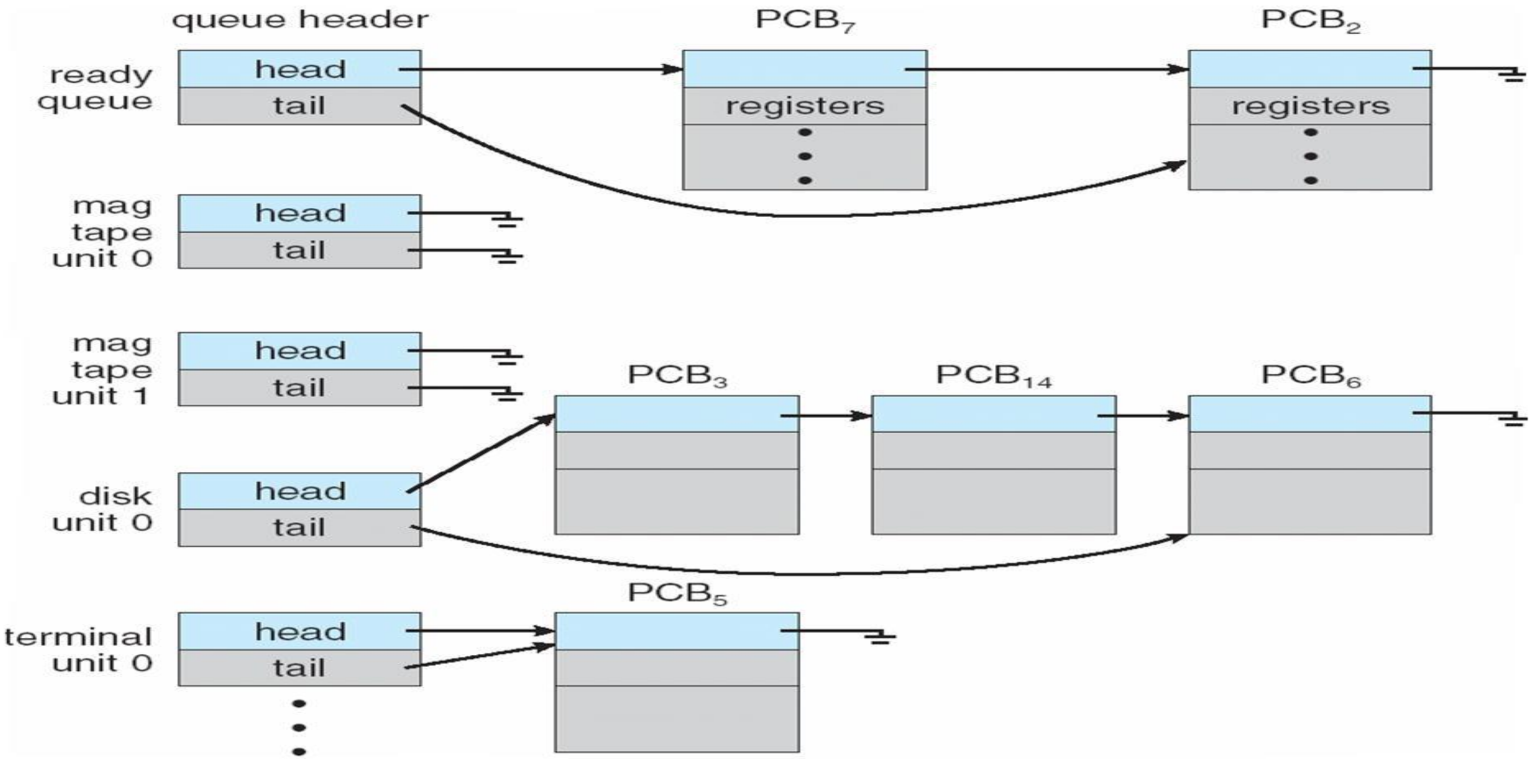
- set of all processes residing in main memory, ready and waiting to execute

Device queues

- set of processes waiting for an I/O device

Processes migrate among the various queues

Ready Queue and Various I/O Device queues



Schedulers

Long-term scheduler (or job scheduler)

- selects which processes should be brought into the ready queue

Short-term scheduler (or CPU scheduler)

- selects which process should be executed next and allocates CPU

Schedulers

Short-term scheduler is invoked very frequently (milliseconds) -> (must be fast)

Long-term scheduler is invoked very infrequently (seconds, minutes) -> (may be slow)

The long-term scheduler controls the *degree of multiprogramming*

Processes can be described as either:

- **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
- **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Context Switches

Context Switching

When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch.

Context of a process represented in the PCB

Context-switch time is overhead; the system does no useful work while switching

Time dependent on hardware support

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 - possibilities

- Share all resources
- Share subset of resources
- Share no resources

Execution

- Parent and children execute concurrently
- Parent waits until children terminate

Process Creation

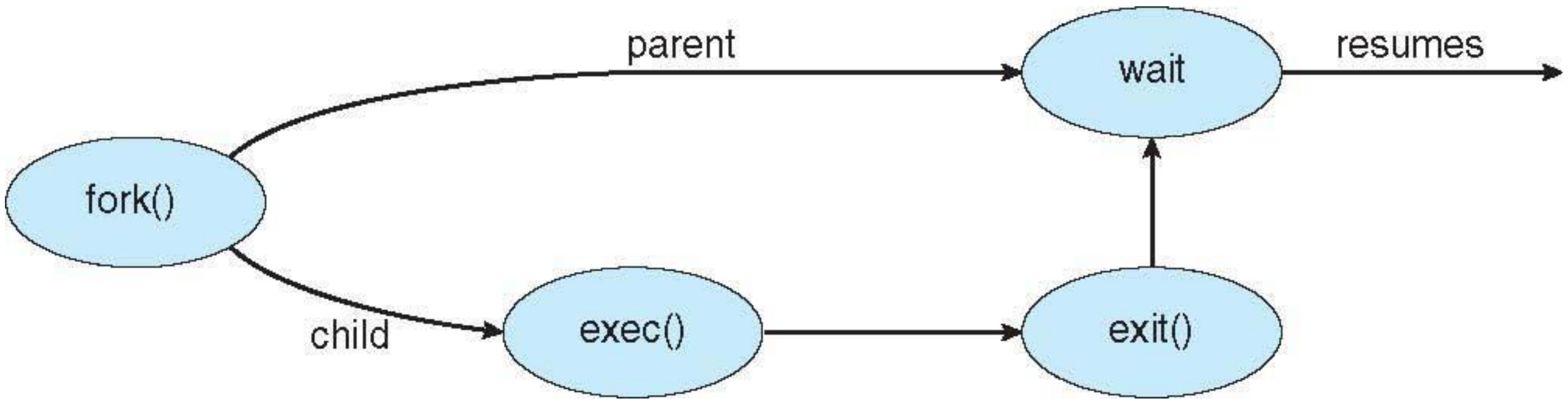
Address space

- Child duplicate of parent
- Child has a program loaded into it

UNIX examples

- **fork** system call creates new process
- **exec** system call used after a **fork** to replace the process' memory space with a new program

Process Creation



Process Termination

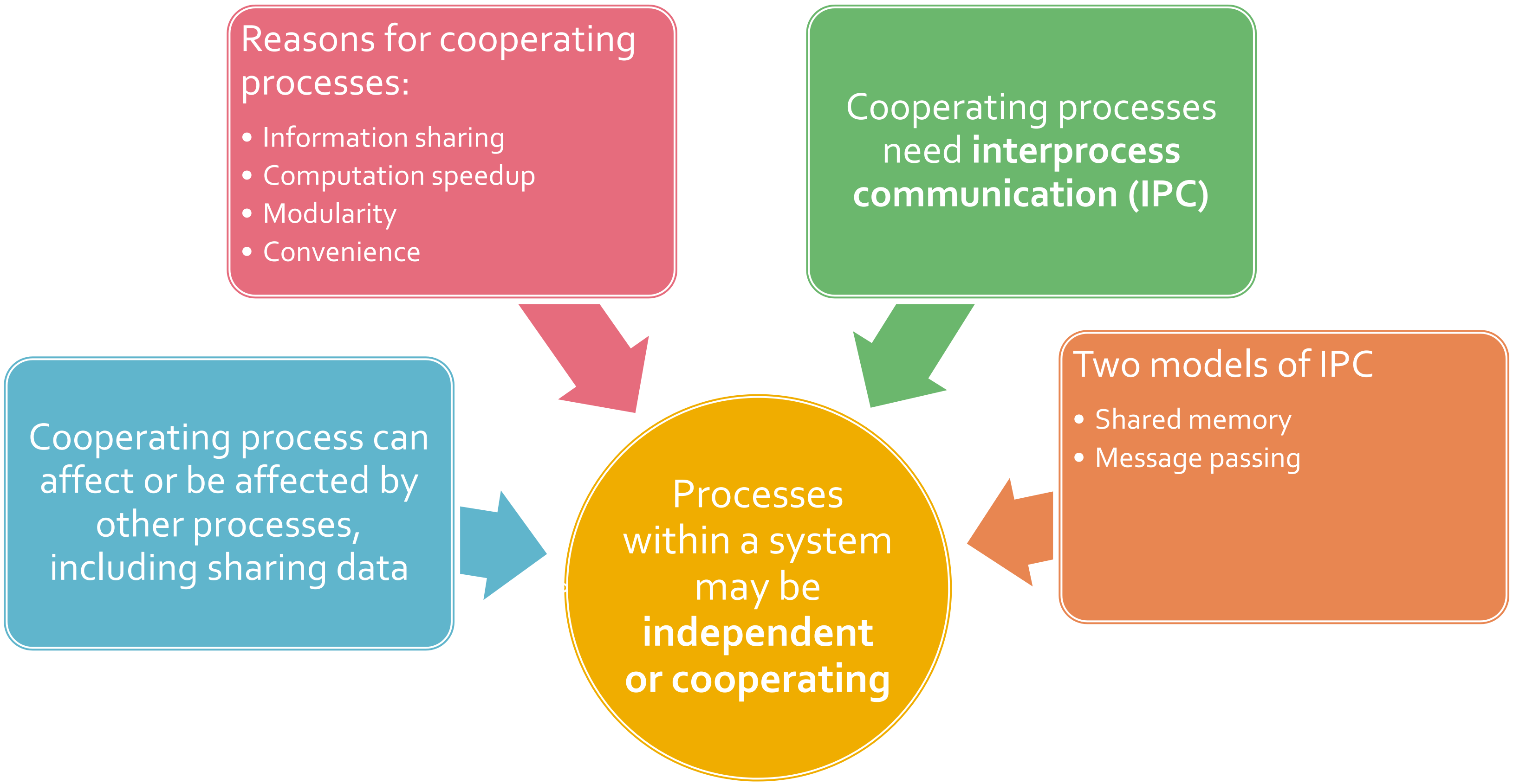
Process executes last statement and asks the operating system to delete it (**exit**)

- Output data from child to parent (via **wait)**
- Process' resources are deallocated by operating system

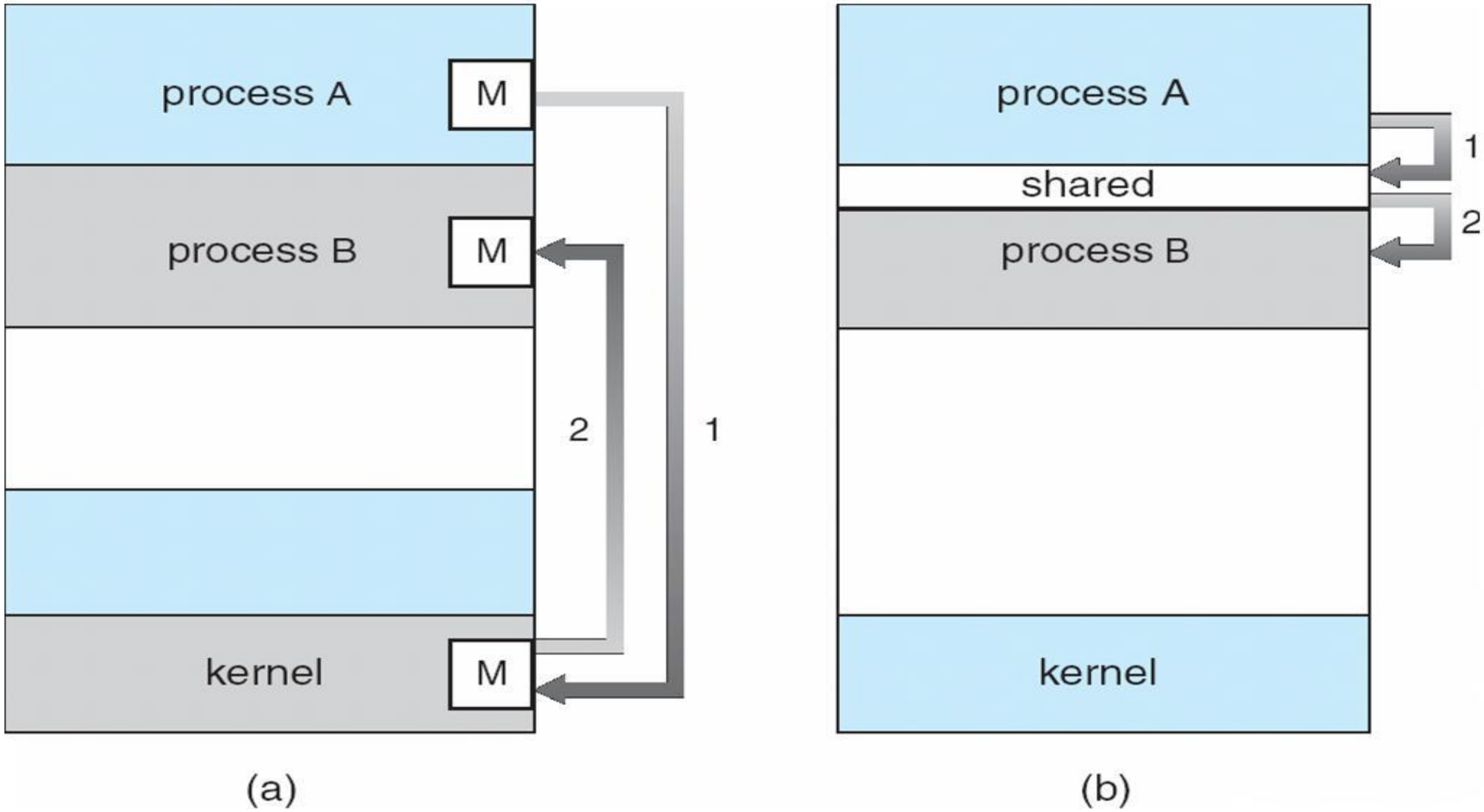
Parent may terminate execution of children processes (**abort**)

- Child has exceeded allocated resources
- Task assigned to child is no longer required
- If parent is exiting
  - Some operating system do not allow child to continue if its parent terminates
    - All children terminated - **cascading termination**

Interprocess Communication



Communications Models



Inter-process Communication. Message Passing

Mechanism for processes to communicate and to synchronize their actions

Message system – processes communicate with each other without resorting to shared variables

IPC facility provides two operations:

- **send(*message*)** – message size fixed or variable
- **receive(*message*)**

If *P* and *Q* wish to communicate, they need to*:*

- establish a *communication link* between them
- exchange messages via send/receive

Implementation of communication link

- physical (e.g., shared memory, hardware bus)
- logical (e.g., logical properties)

Direct Communication

Processes must name each other explicitly:

- **send (*P, message*)** – send a message to process P
- **receive(*Q, message*)** – receive a message from process Q

Properties of communication link

- Links are established automatically
- A link is associated with exactly one pair of communicating processes
- Between each pair there exists exactly one link
- The link may be unidirectional, but is usually bi-directional

Indirect Communication

Messages are directed and received from mailboxes (also referred to as ports)

- Each mailbox has a unique id
- Processes can communicate only if they share a mailbox

Properties of communication link

- Link established only if processes share a common mailbox
- A link may be associated with many processes
- Each pair of processes may share several communication links
- Link may be unidirectional or bi-directional

Indirect Communication

Operations

- create a new mailbox
- send and receive messages through mailbox
- destroy a mailbox

Primitives are defined as:

- **send(*A, message*)** – send a message to mailbox A
- **receive(*A, message*)** – receive a message from mailbox A

Synchronization

Message passing may be either blocking or non-blocking

Blocking is considered **synchronous**

Non-blocking is considered **asynchronous**

Blocking send has the sender block until the message is received

Blocking receive has the receiver block until a message is available

Non-blocking send has the sender send the message and continue

Non-blocking receive has the receiver receive a valid message or null

IPC Systems Examples-Windows

- Message-passing centric via **local procedure call (LPC)** facility

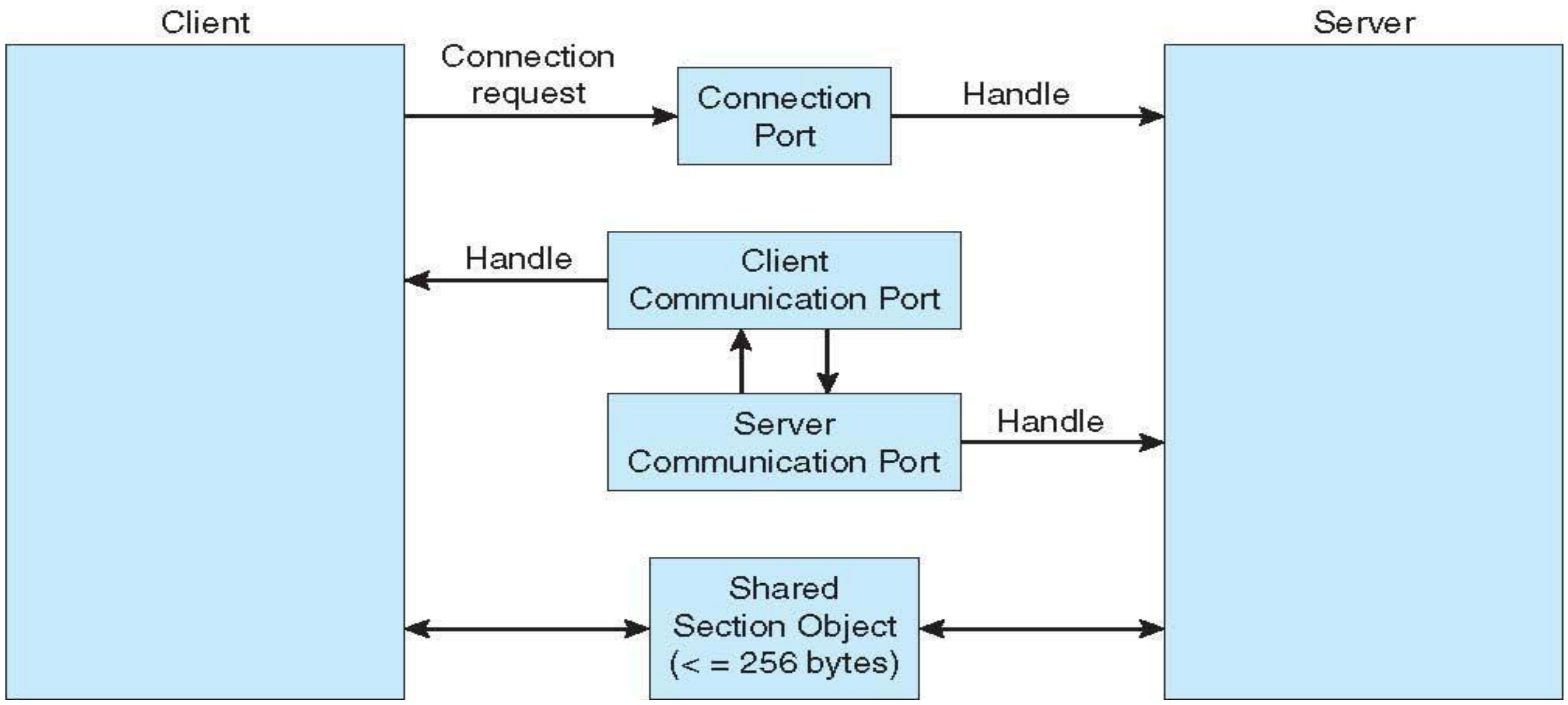
Only works between processes on the same system

Uses ports (like mailboxes) to establish and maintain communication channels

Communication works as follows:

- The client opens a handle to the subsystem's connection port object.
- The client sends a connection request.
- The server creates two private communication ports and returns the handle to one of them to the client.
- The client and server use the corresponding port handle to send messages or callbacks and to listen for replies.

Local Procedure Calls in Windows



Communications in Client-Server Systems.

- Sockets
- Remote Procedure Calls
- Pipes
- Remote Method Invocation (Java)
- .NET Remoting

Sockets

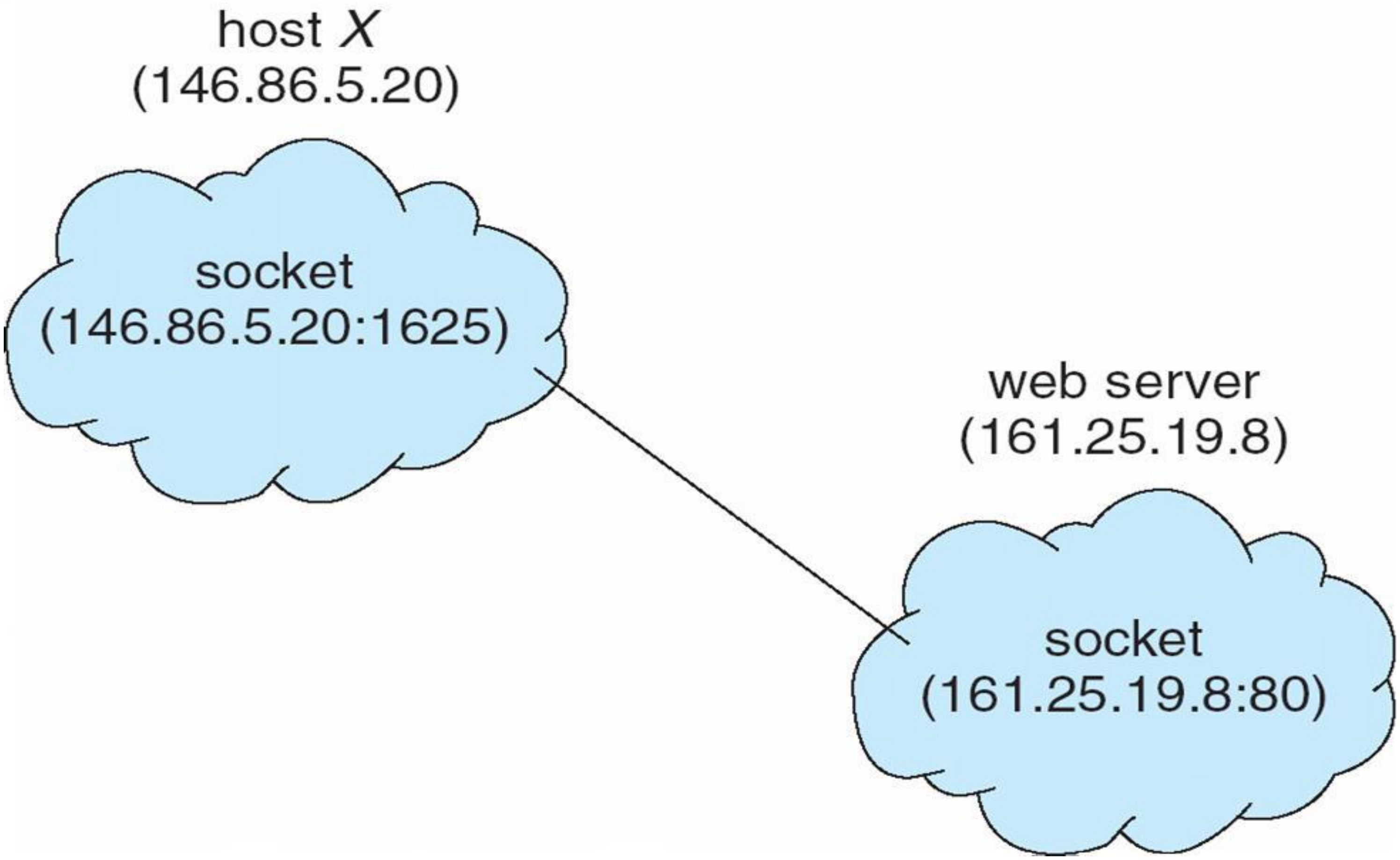
A socket is defined as an *endpoint for communication*

Concatenation of IP address and port

The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**

Communication consists between a pair of sockets

Socket Communications



Remote Procedure Calls

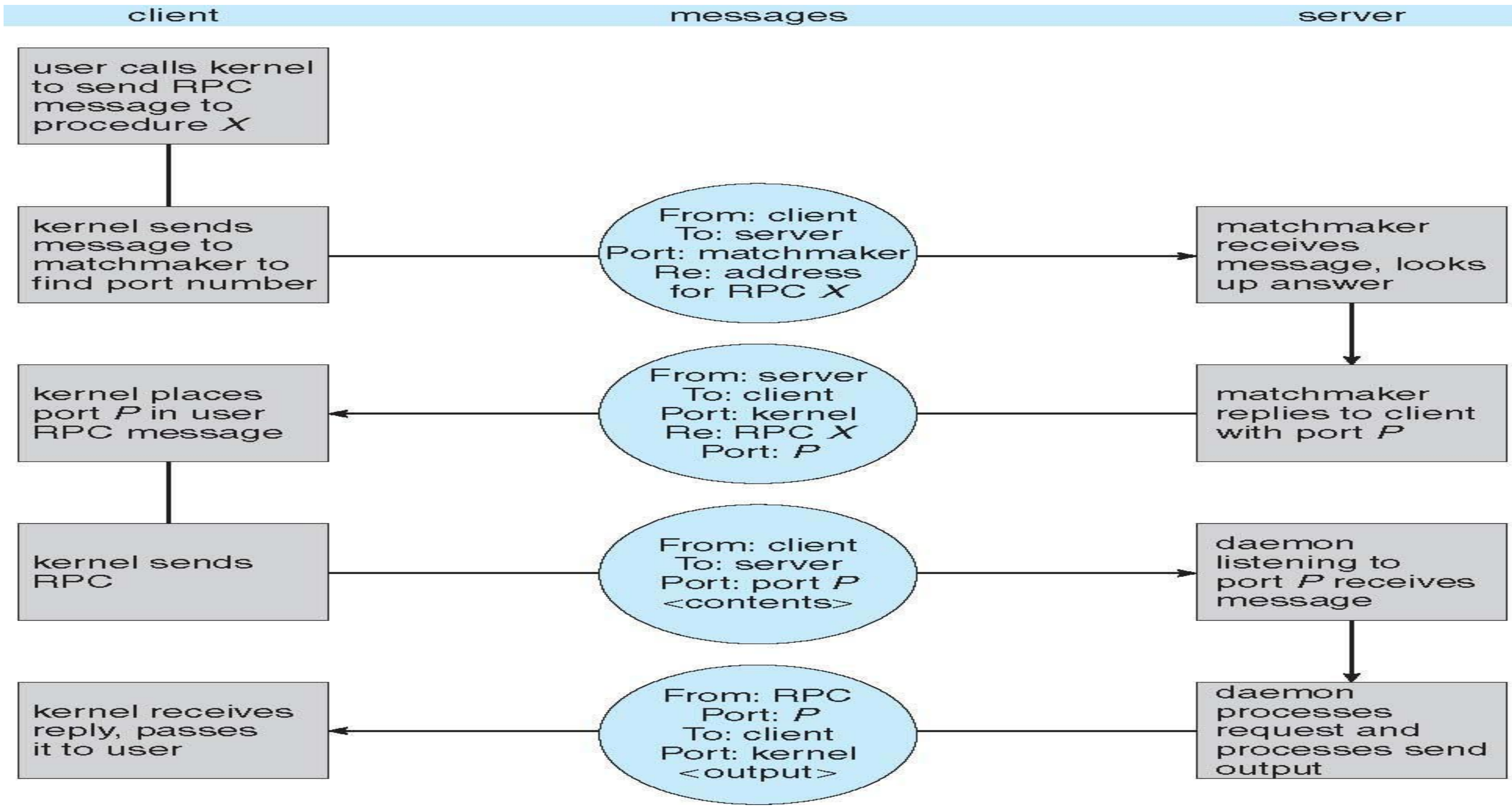
Remote procedure call (RPC) abstracts procedure calls between processes on networked systems

Stubs – client-side proxy for the actual procedure on the server

The client-side stub locates the server and *marshalls* the parameters

The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server

Execution of RPC



Named Pipes

- Named Pipes are more powerful than ordinary pipes
- Communication is bidirectional
- No parent-child relationship is necessary between the communicating processes
- Several processes can use the named pipe for communication
- Provided on both UNIX and Windows systems