Processes

Objectives

- To introduce the notion of a process – a program in execution
- To describe the various features of processes
  - Scheduling
  - Creation
  - Termination
  - Communication
  - Etc.
- To describe communication in client-server systems.

Outline

- Process Concept ...
- Process Scheduling ...
- Operations on Processes ...
- Cooperating Processes ...
- Interprocess Communication ...
- Communication in Client-Server Systems ...
- Summary ...

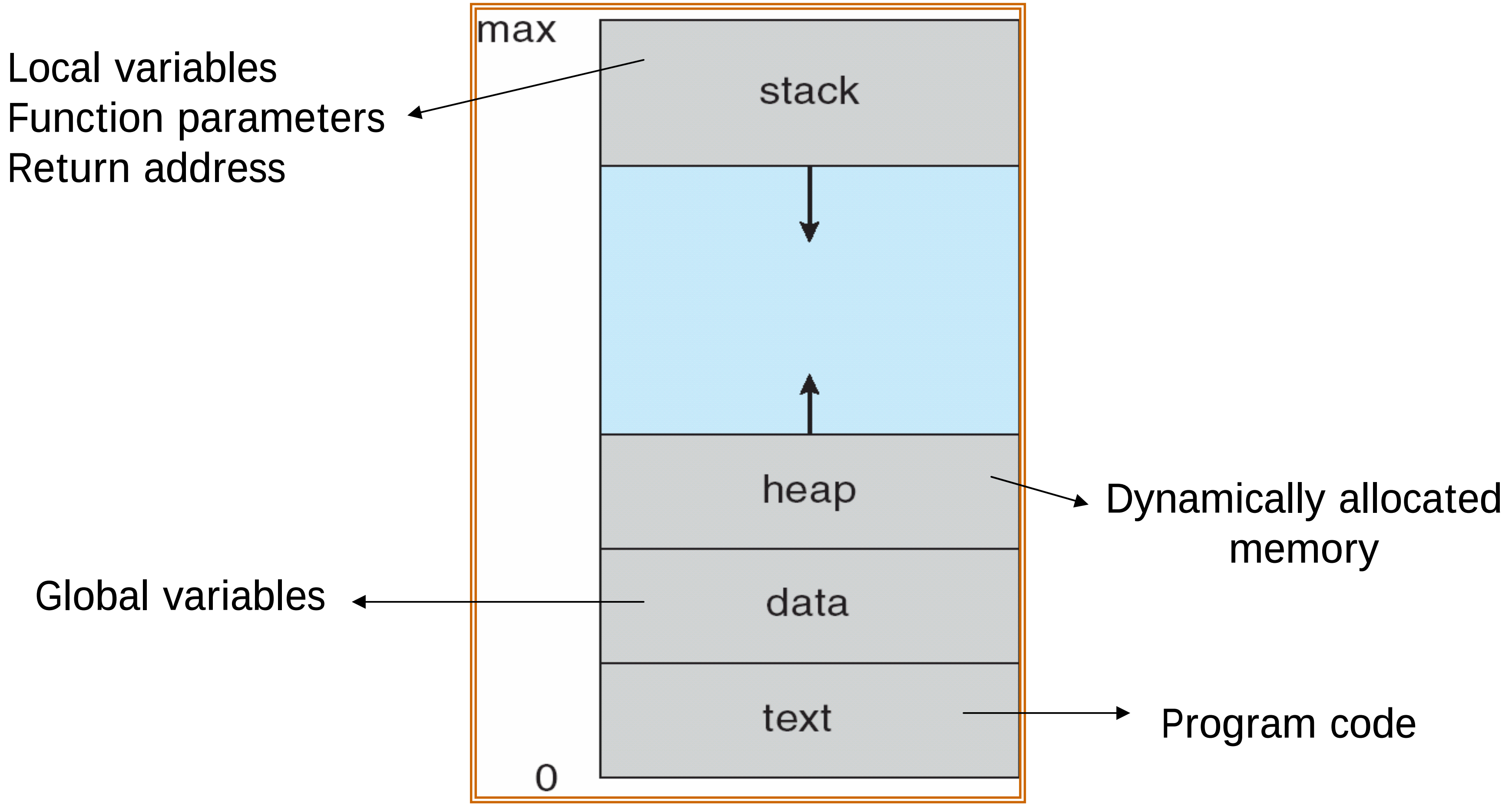
- Process Concept

- Basic Concepts ...
- Process State ...
- Process Control Block (PCB) ...

-- Basic Concept

- An operating system 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:
  - Code
  - stack
  - data section
- 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.

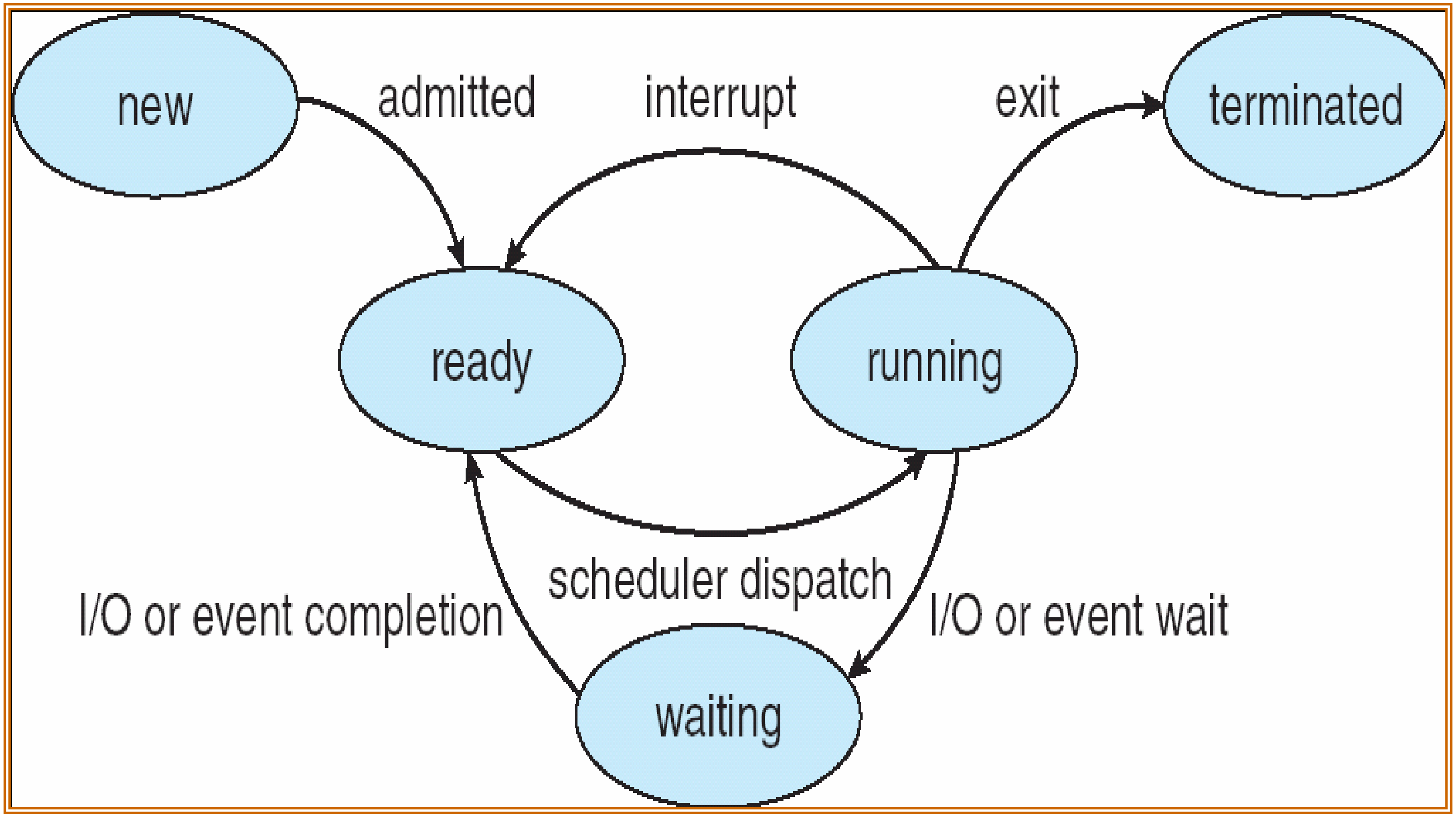
--- Process in Memory



-- Process State

- As a process executes, it changes *state*
  - 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 process.
  - terminated: The process has finished execution.

--- Diagram of Process State

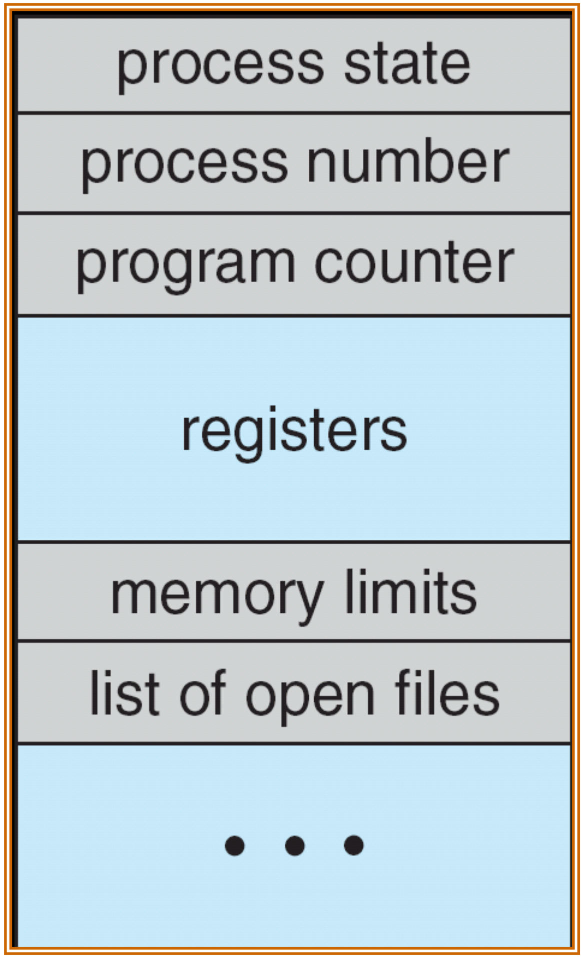


-- Process Control Block (PCB)

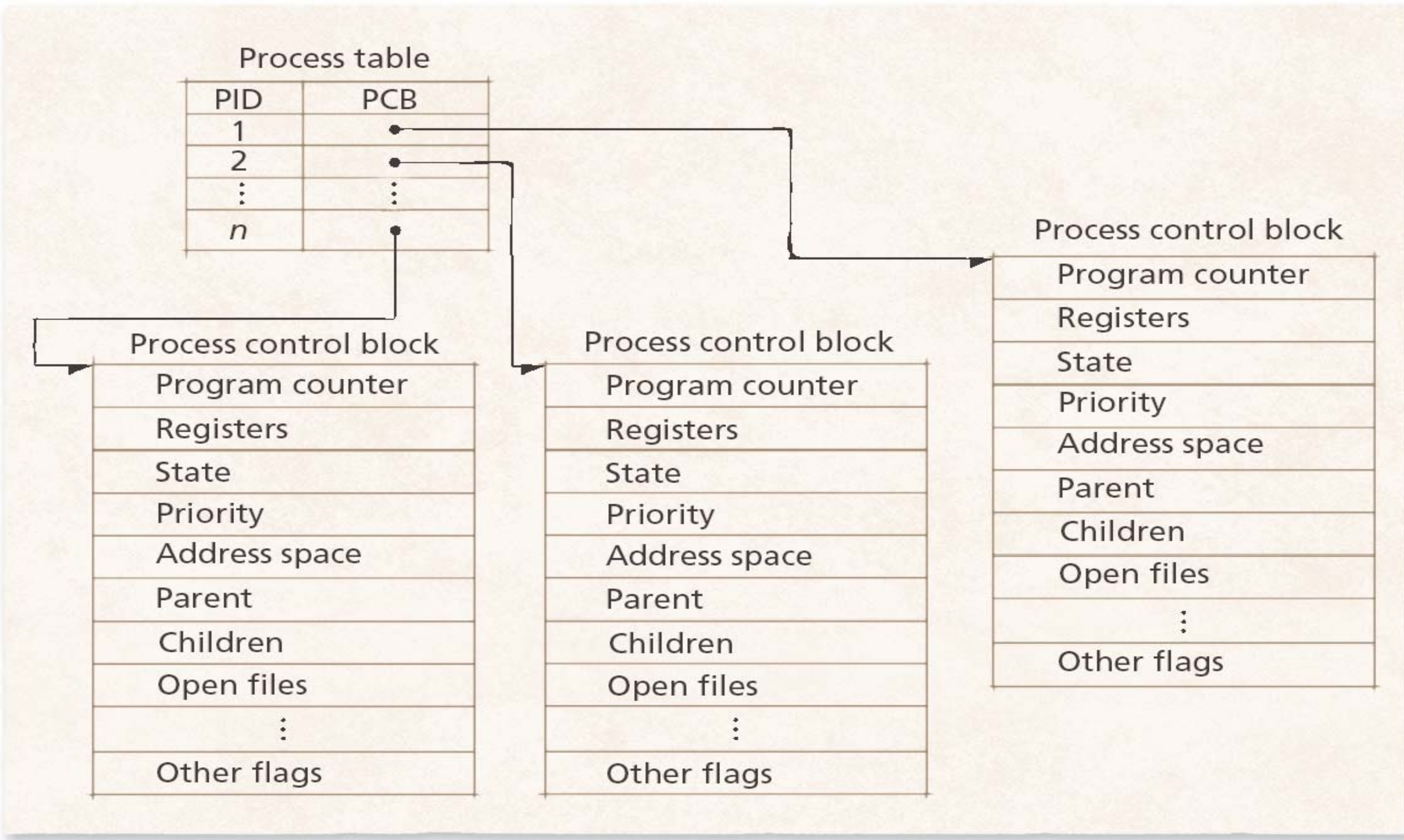
Information associated with each process.

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

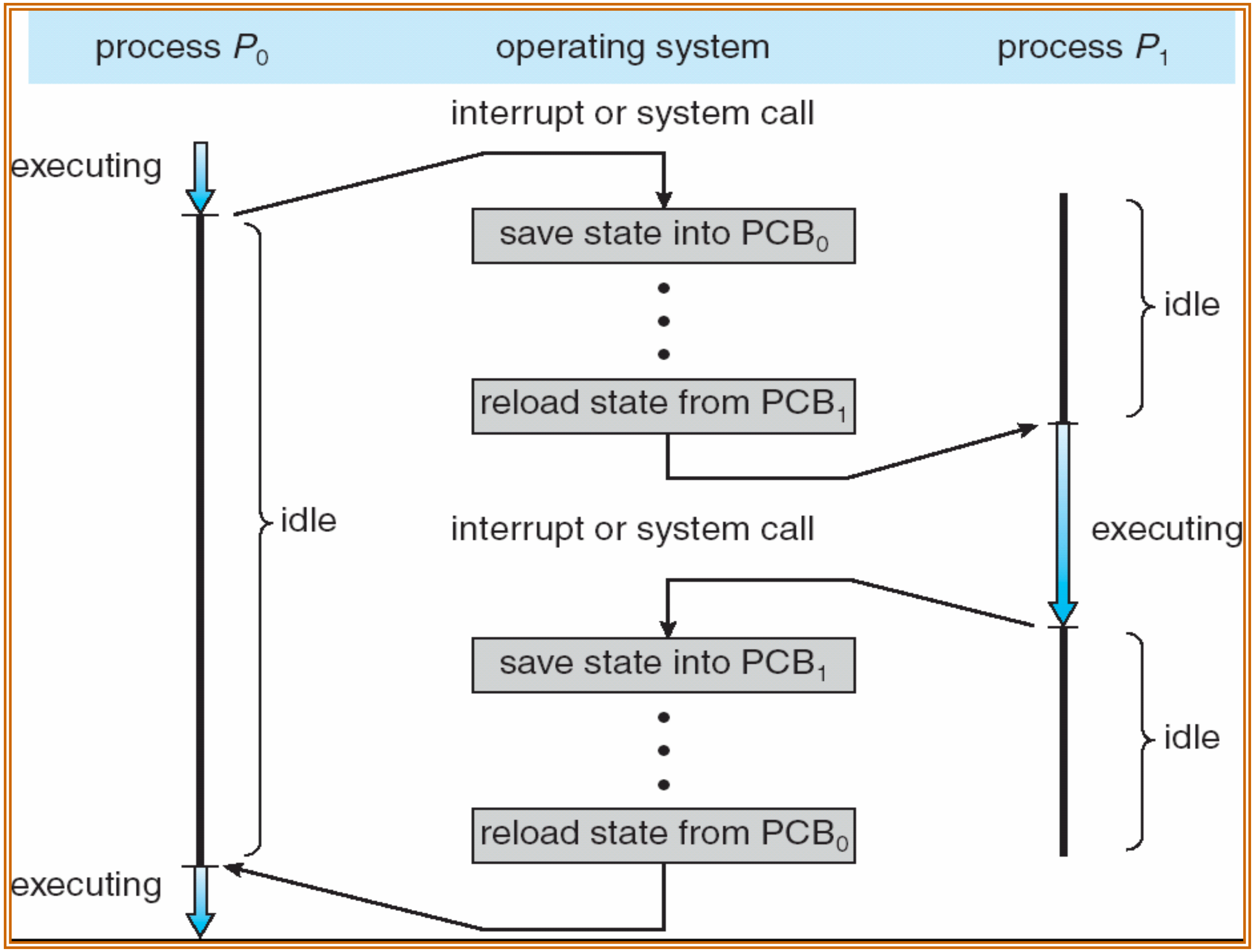
--- Process Control Block (PCB)



--- Process Table and Process Control Block (PCB)



--- CPU Switch From Process to Process



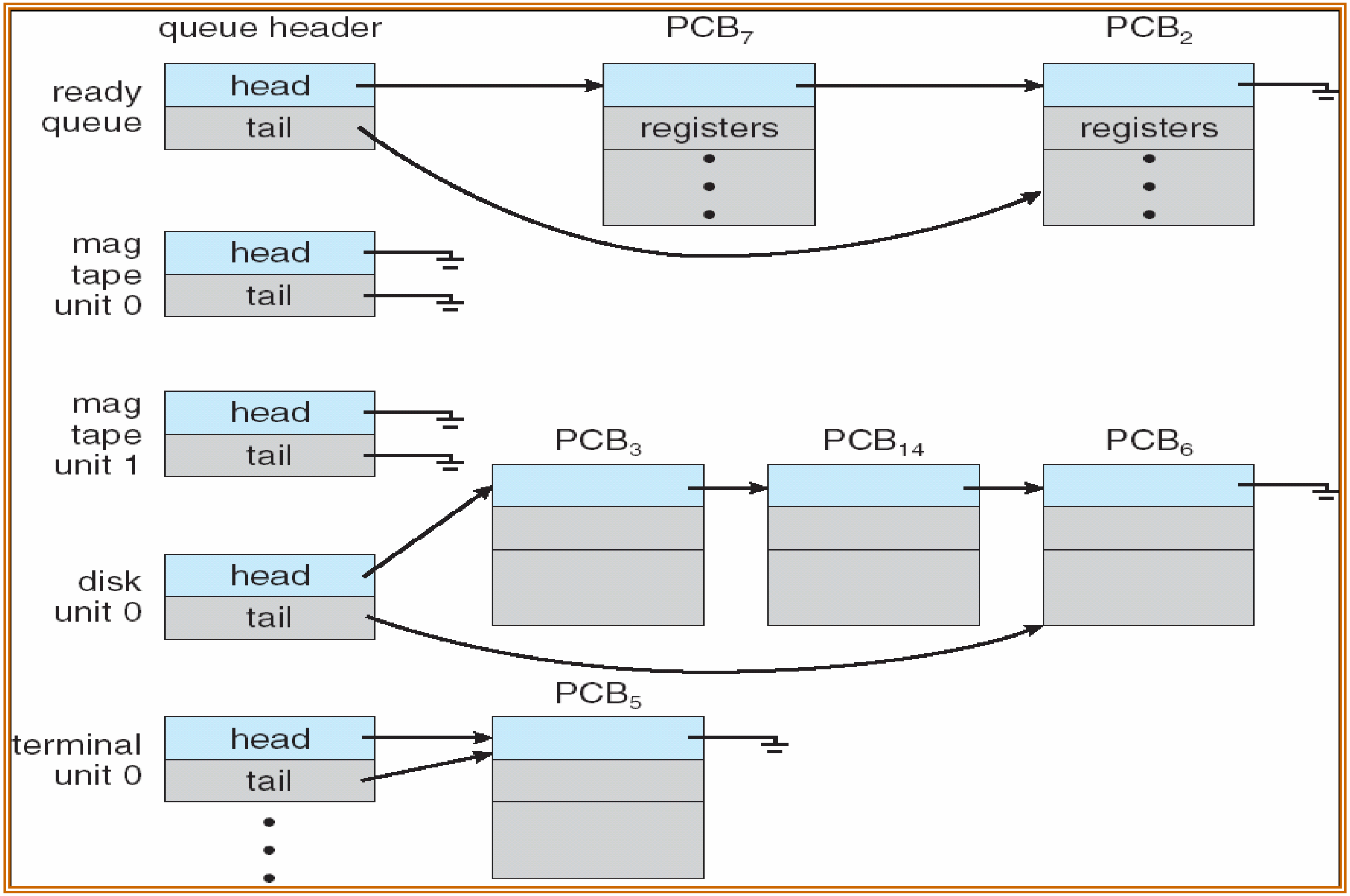
- Process Scheduling

- In multi-programming environment processes compete to get resources. Because the number of resources are limited, the OS efficiently schedules processes before it assigns them a resource. In this section we will cover:
  - Scheduling Queues ...
  - Schedulers ...
  - Context Switching ...

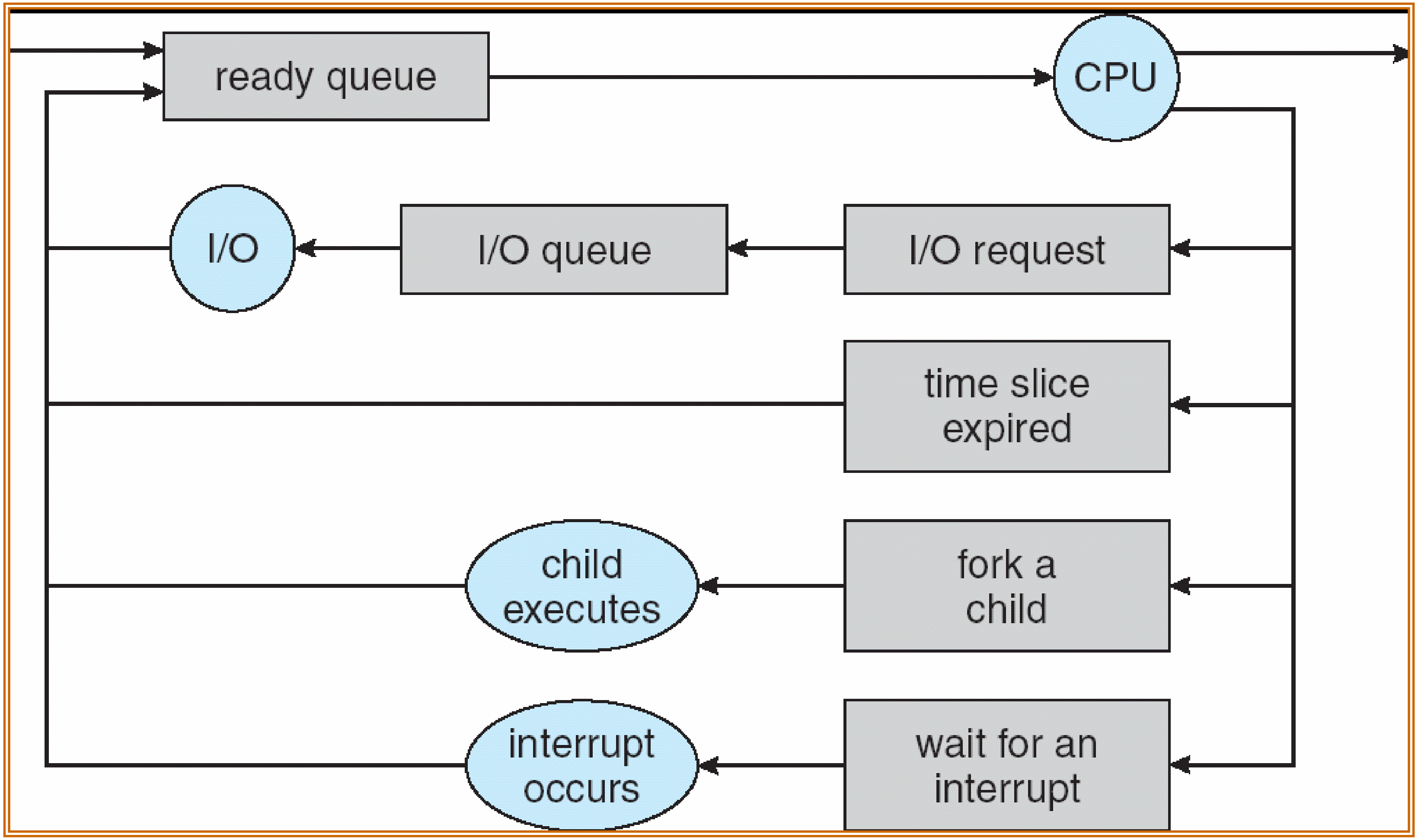
-- Scheduling Queues

- 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.
- Process migration between the various queues.

--- Ready Queue And Various I/O Device Queues



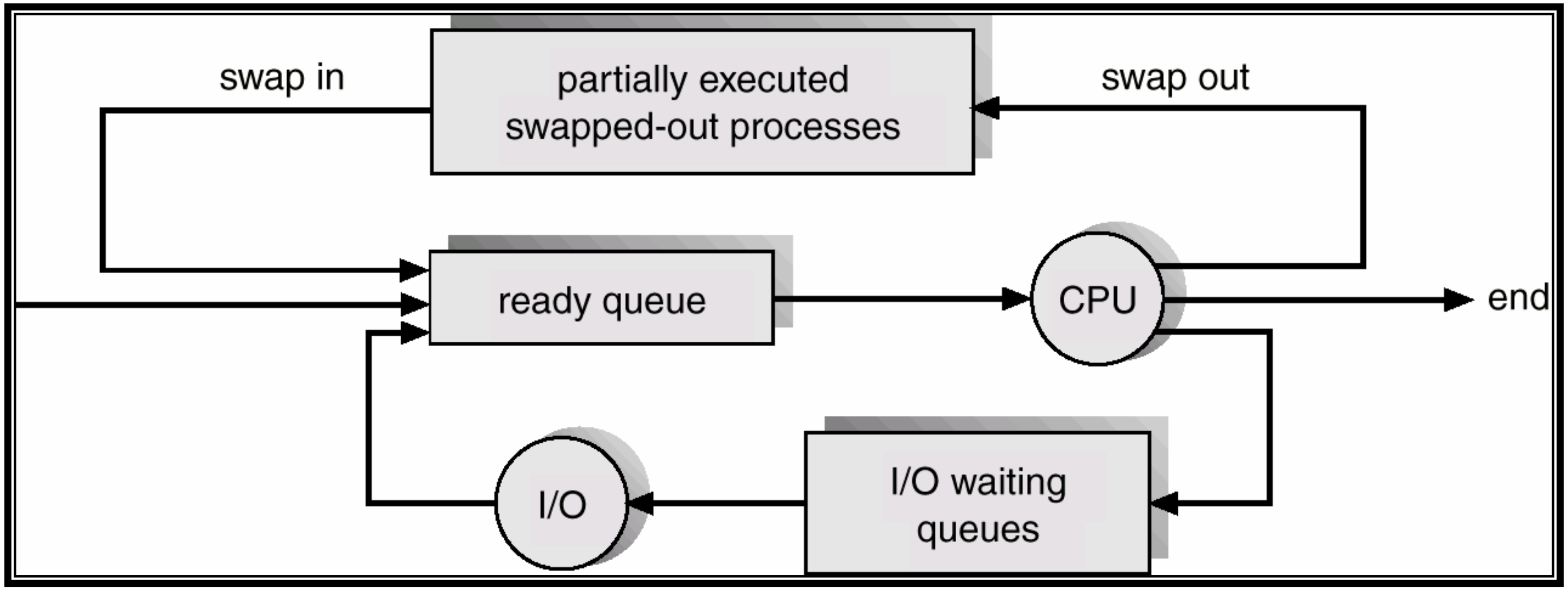
--- Representation of Process Scheduling



-- Schedulers

- Long-term scheduler:
  - Is also called job scheduler
  - Selects which processes should be brought into the ready queue
  - Is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
  - Controls the degree of multiprogramming
- Short-term scheduler:
  - Is also called CPU scheduler
  - Selects which process should be executed next and allocates CPU
  - Is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Medium-term scheduler:
  - Swaps in and out jobs form memory to improve efficiency.
  - Found in some time-sharing systems.

--- Medium Term Scheduling



--- Context Switch

- 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.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

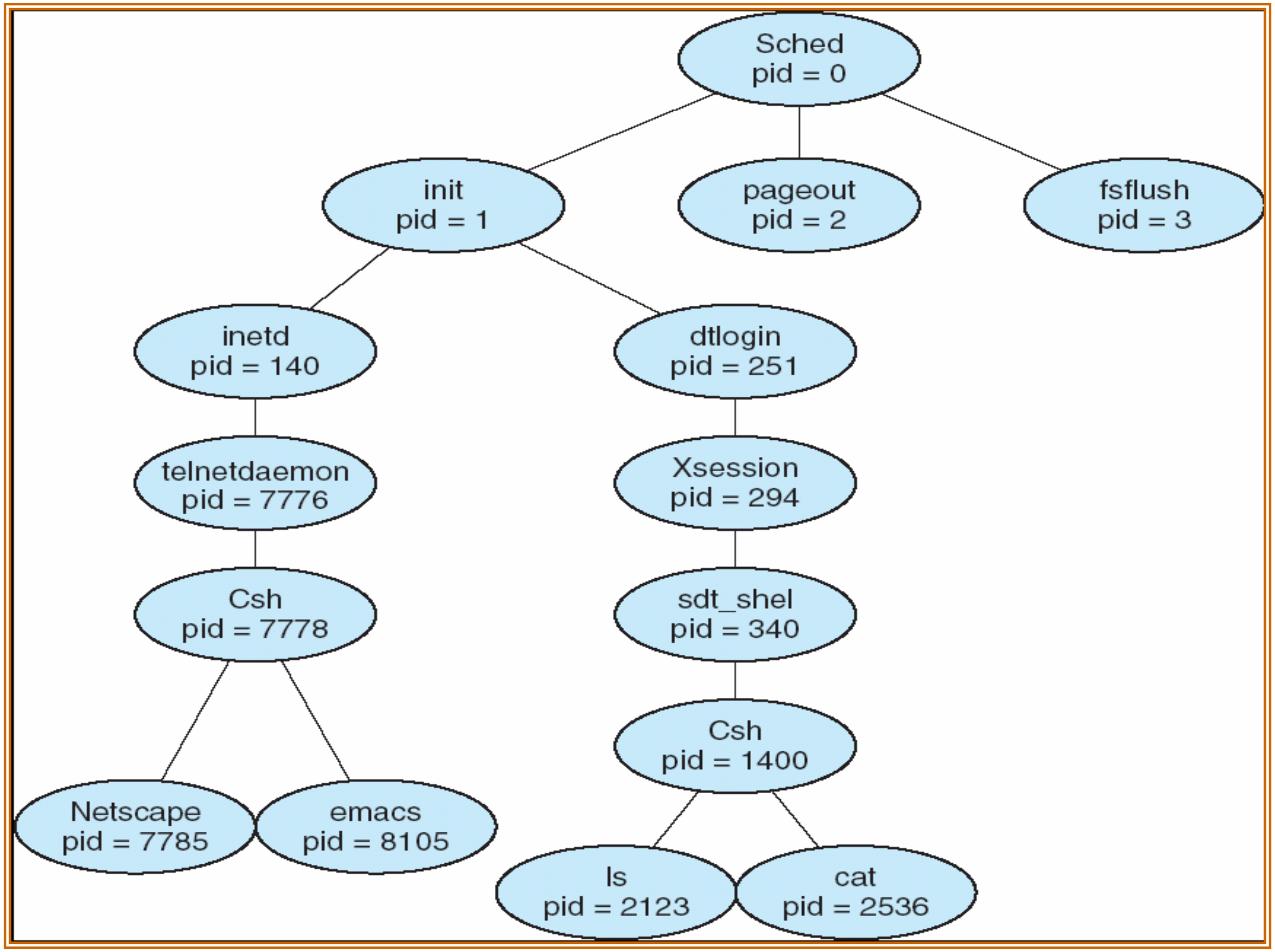
- Operations on Processes

- Process Creation ...
- Process Termination ...

-- Process Creation ...

- Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

A tree of processes on a typical Solaris

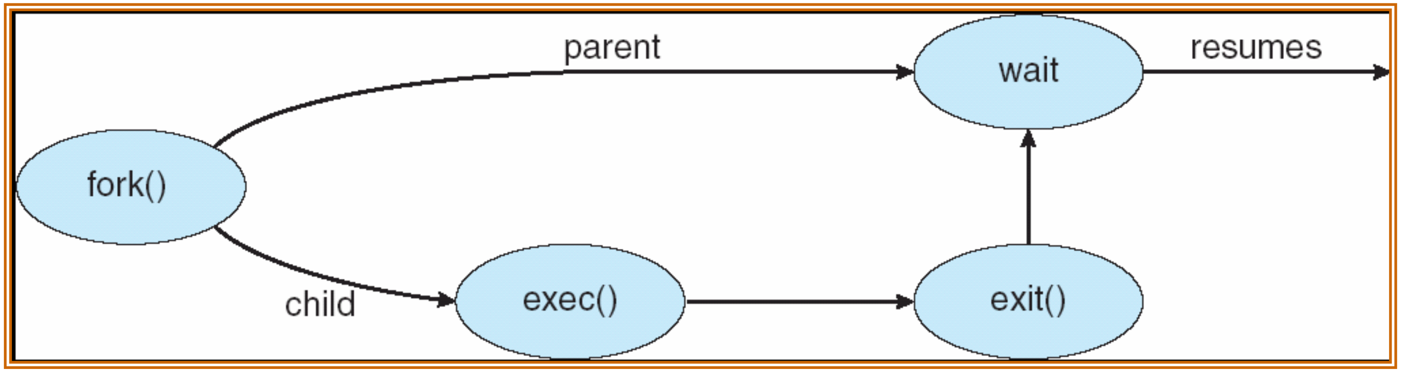


... -- Process Creation

- 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



--- C Program forking a separate process

```
#include <stdio.h>

Main(int argc, char *argv[ ])
{
    int pid;
    pid = fork( );     /* child process created */
    if (pid < 0 ) {    /* Error occurred   */
       fprintf(stderr, "Fork Failed");
    }
    else if (pid == 0) {     /* Child process   */
        execlp("/bin/ls", "ls", NULL);
    }
    else  {      /* Parent process    */
       wait(NULL);
       printf("Child Complete");
       exit(0);
    }
}
```

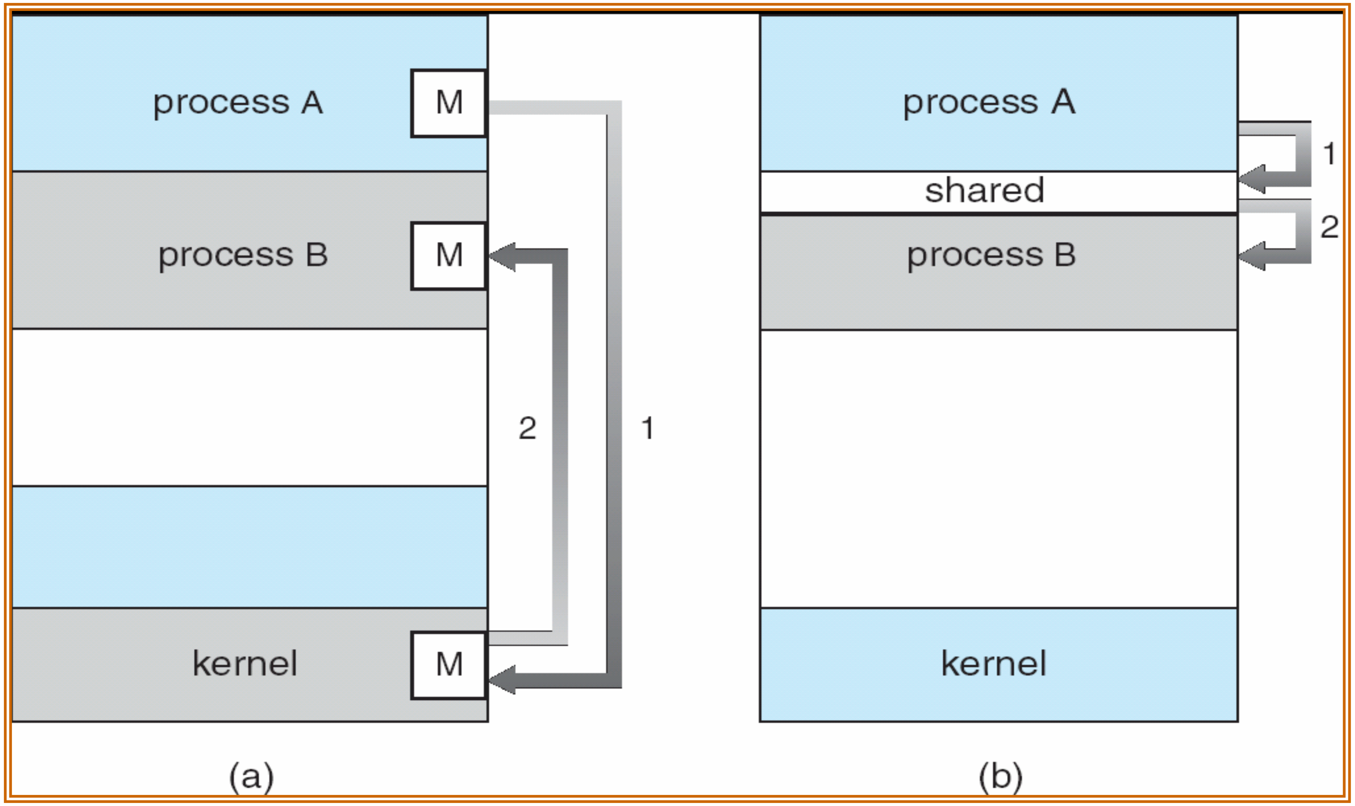
-- Process Termination

- Process executes last statement and asks the operating system to 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.
  - Parent is exiting.
    - Operating system does not allow child to continue if its parent terminates.
    - Cascading termination.

- Interprocess Communication (IPC)

- IPC is a mechanism for processes to communicate and to synchronize their actions.
- Independent process cannot affect or be affected by the execution of another process.
- Cooperating process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience
- There are two fundamental models of IPC
  - Shared memory
  - Message passing

--- Communications Models



(a) Message passing (b) Shared memory

-- Shared Memory

- Communicating processes establish a shared memory
- Faster than message passing – memory speed
- Not easy to implement when processes are in separate computers connected by a network.
- Accessing and manipulating the shared memory be written explicitly by the application programmer

--- Example of Producer-Consumer Process ...

- Paradigm for cooperating processes, producer process produces information that is consumed by a consumer process.
  - unbounded-buffer places no practical limit on the size of the buffer.
  - bounded-buffer assumes that there is a fixed buffer size.

... --- Example of Producer Consumer Process

Shared Variables

```
#define BUFFER-SIZE 10
Typedef struct {
. . .
} item;

Item buffer[BUFFER_SIZE];
Int in = 0;
Int out = 0;
```

Producer

```
while(1) {
    while (((in + 1) % BUFFER_SIZE) == out
          ; /* do nothing */
    buffer[in] = nextProduced
     in = (in + 1) % BUFFER_SIZE;
}
```

Consumer

```
while(1)  {
        while (in == out)
             ;  /*  do nothing */
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
}
```

-- Message passing

- Basic Concepts ...
- Direct Communication ...
- Indirect communication ...
- Synchronization ...
- Buffering ...

-- Basic Concepts

- Message-passing 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
    - physical (e.g., shared memory, hardware bus)
    - logical (e.g., logical properties like direct or indirect; symmetric or asymmetric)
  - exchange messages via send/receive

--- Direct Communication

- Processes must name each other explicitly:
  - Symmetry
    - **send** (*P, message*) – send a message to process P
    - **receive**(*Q, message*) – receive a message from process Q
  - Asymmetry
    - **send** (*P, message*) – send a message to process P
    - **receive**(id*, message*) – receive message from any process.

- 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

... --- Indirect Communication

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A.
  - P_1, sends; P_2 and P_3 receive.
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes.
  - Allow only one process at a time to execute a receive operation.
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

--- Synchronization

- Message passing is may be either blocking or non-blocking
  - Blocking: is considered synchronous
    - **Blocking send** has the sender block until the message is received
    - **Blocking receive** has the receiver block until a message is available
  - non-blocking: is considered asynchronous
    - **Non-blocking** send has the sender send the message and continue
    - **Non-blocking** receive has the receiver receive a valid message or null

-- Buffering

- Queue of messages attached to the link; implemented in one of three ways.

1. Zero capacity – 0 messages
   Sender must wait for receiver (rendezvous).

2. Bounded capacity – finite length of *n* messages
   Sender must wait if link full.

3. Unbounded capacity – infinite length
   Sender never waits.

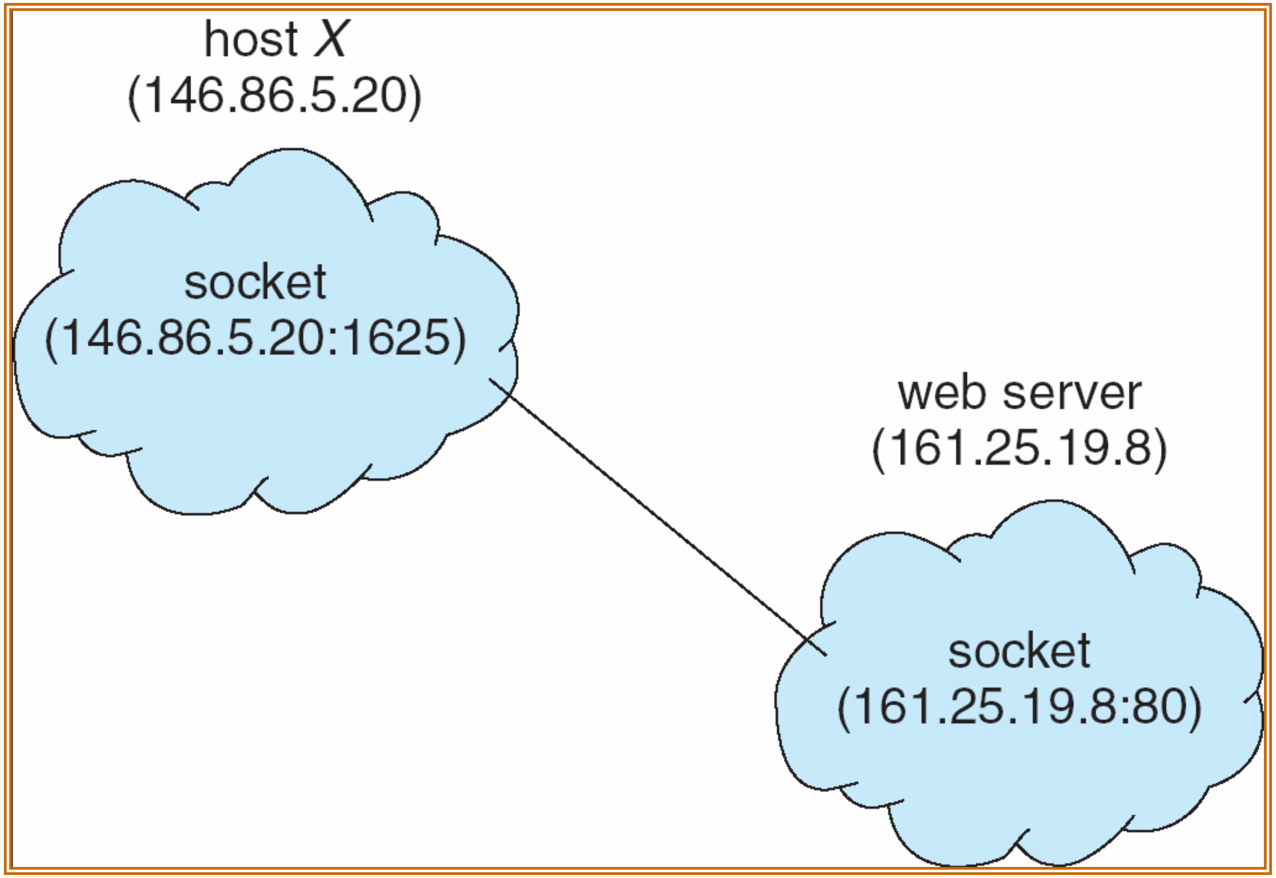
- Client-Server Communication

- Sockets ...
- Remote Procedure Calls ...
- Remote Method Invocation (Java) ...

-- 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.

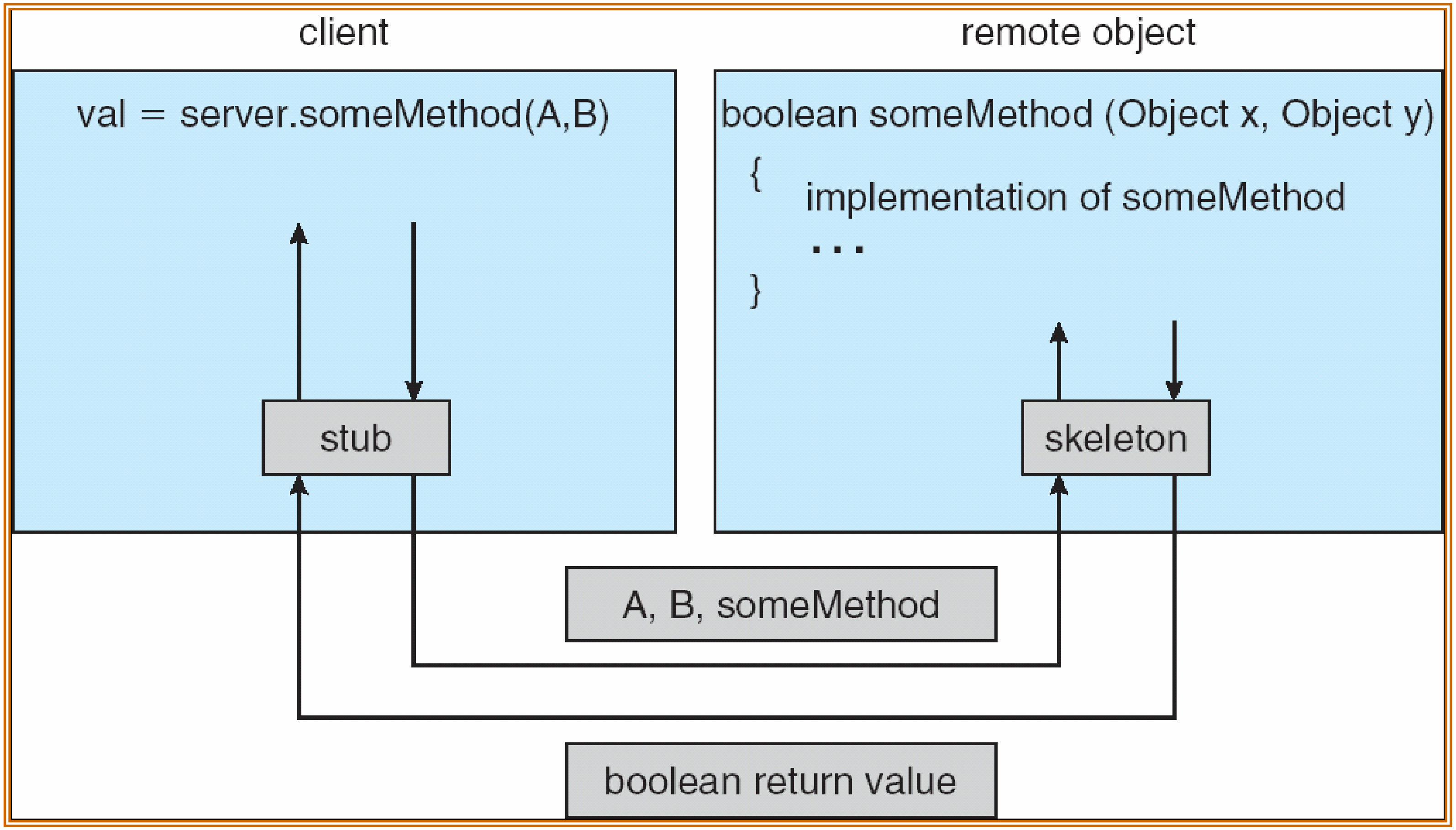
--- Socket Communication



-- Remote Procedure Calls

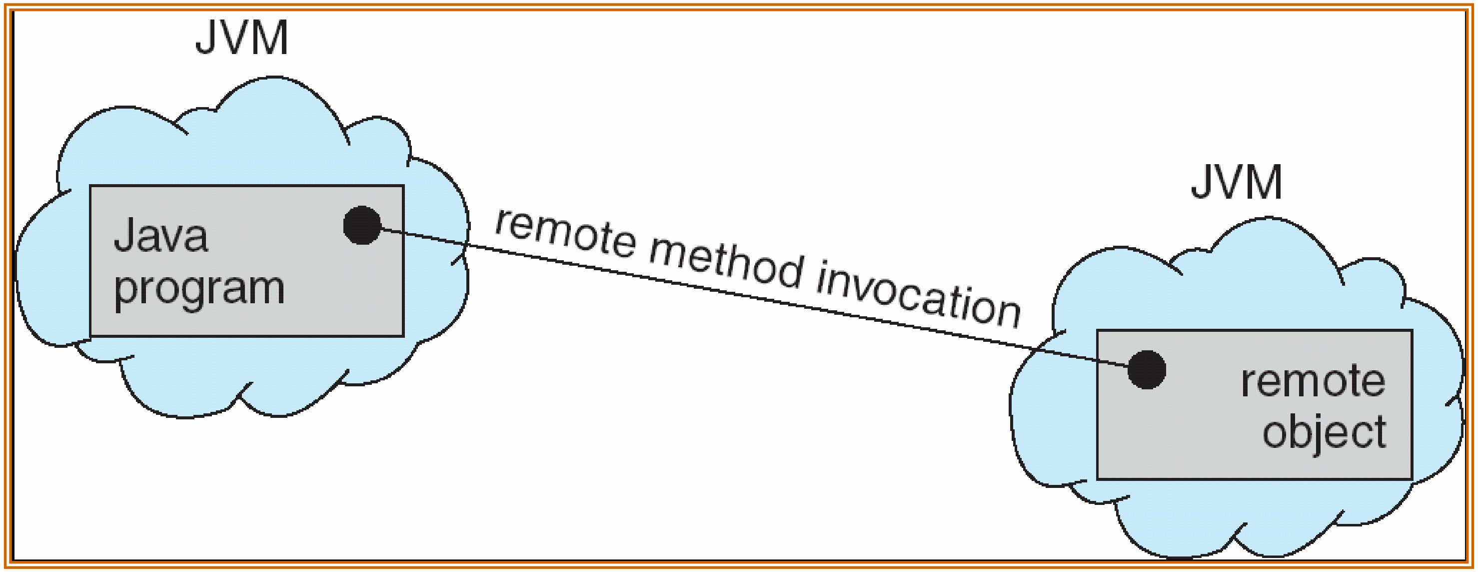
- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
- Stubs – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and marshals the parameters.
- The server-side stub receives this message, unpacks the marshaled parameters, and performs the procedure on the server.

--- Marshalling Parameters

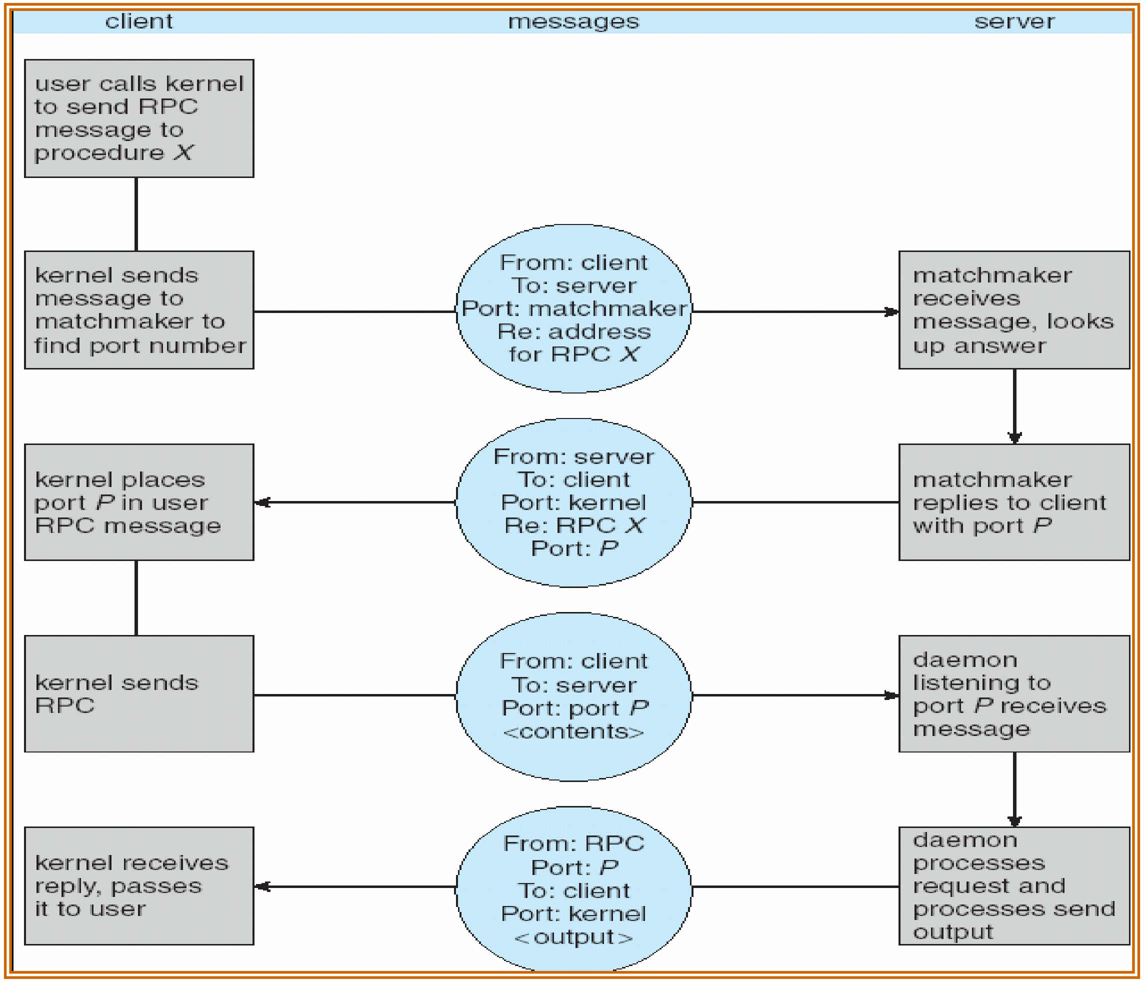


-- Remote Method Invocation

- Remote Method Invocation (RMI) is a Java mechanism similar to RPCs.
- RMI allows a Java program on one machine to invoke a method on a remote object.



--- Execution of RPC



- Summary

- Process: A program in execution
  - Batch vs. time sharing
  - I/O bound process vs. CPU bound process
- Process state: new, ready, running, waiting, terminated
- Context switching: PCB
- Process scheduling: Short, medium, long term schedulers
- Operations on processes: process creation & termination.
- IPC:
  - shared memory
    - Producer consumer
  - message passing
    - Direct vs. Indirect communication; Synchronization; Buffering
- Client-Server communication: Sockets, RPC, Stub, RMI

End of Chapter 3