4 PROCESOS

1 CONCEPTO DE PROCESO

2 ITINERACIÓN DE PROCESOS

3 OPERACIONES SOBRE PROCESOS

4 PROCESOS COOPERATIVOS

5 COORDINACIÓN ENTRE PROCESOS

4.1 CONCEPTO DE PROCESO

1 El Proceso
2 Estado del Proceso
3 Bloque de Control de Proceso

4.1.1 El Proceso

- Un sistema operativo ejecuta varios programas:
  - En Sistemas Batch : jobs
  - En Sistemas de tiempo compartido : programas de usuario o tareas
- Job y proceso se usan en forma equivalente.
- Proceso: un programa en ejecución; la ejecución del proceso debe progresar de manera secuencial.
- Un proceso incluye:
  - Contador de programa (program counter)
  - Stack: variables locales, paso de parámetros
  - Sección de datos: variables globales

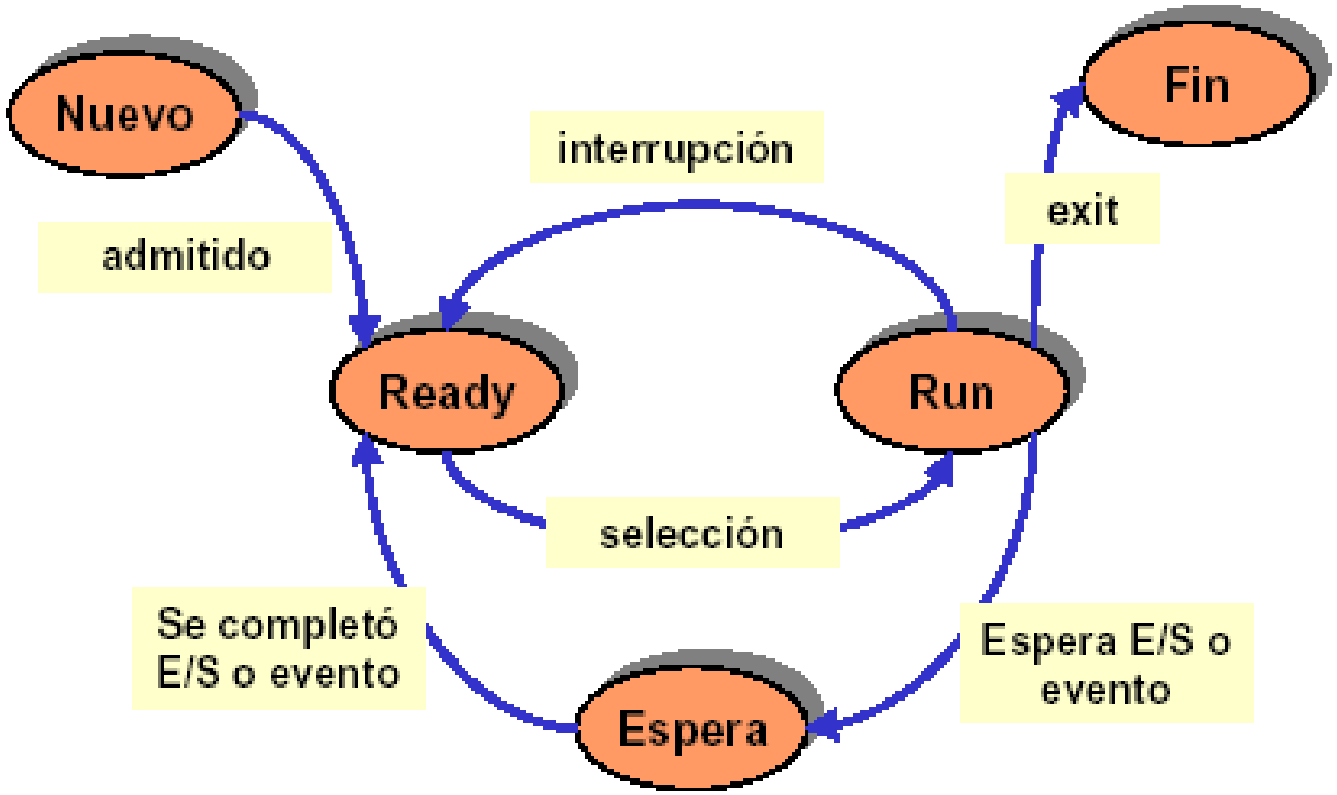
4.1.2 Estado del Proceso

A medida que un proceso se ejecuta, va cambiando de estado:

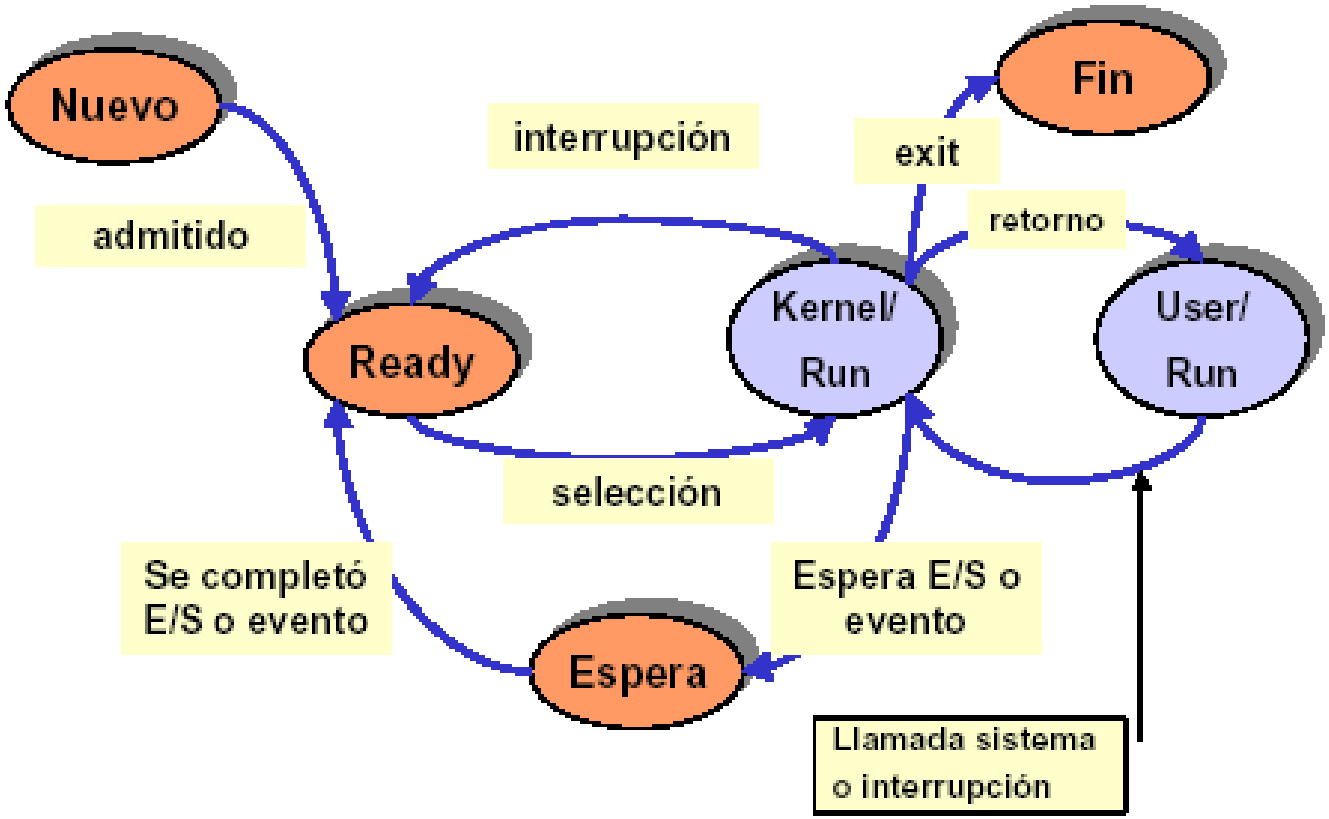
- **Nuevo** (New): El proceso está siendo creado.
- **Ejecución** (Running): Sus instrucciones están siendo ejecutadas.
- **Espera** (Waiting): El proceso está esperando a que un evento ocurra (generalmente la completación de una operación de E/S).
- **Listo** (Ready): El proceso está esperando que le sea asignado el procesador.
- **Terminado** (Terminated): El proceso ha finalizado su ejecución.

4.1.2 Estado del Proceso

Diagrama de Estados de un Proceso



Modos Kernel y Modo Usuario



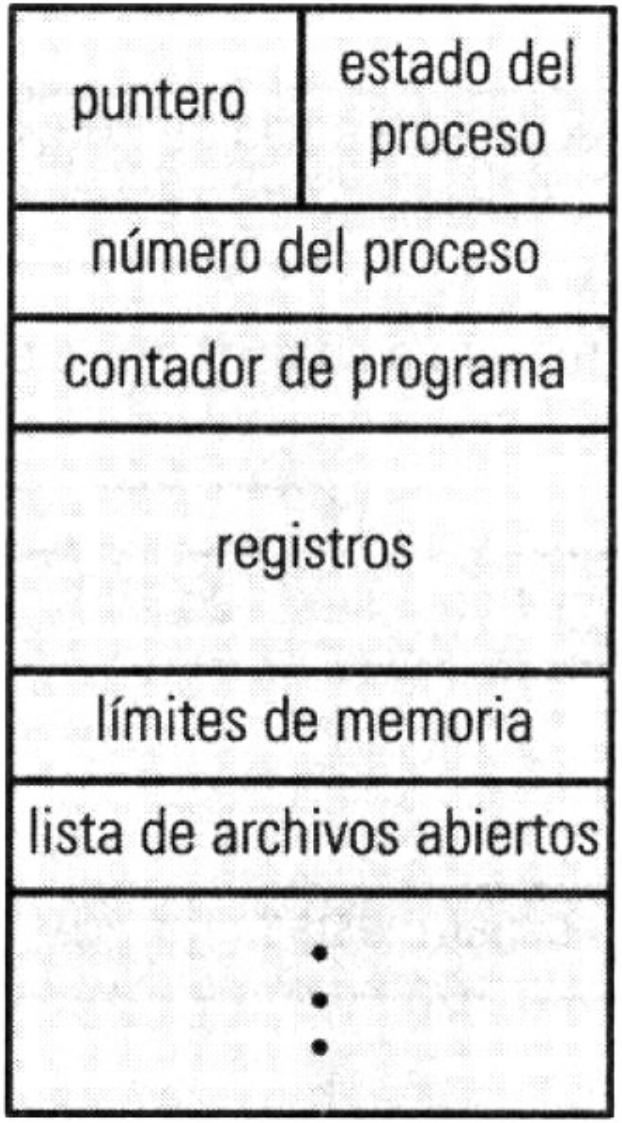
4.1.3 Bloque de Control de Proceso

El **PCB** contiene información asociada con cada proceso.

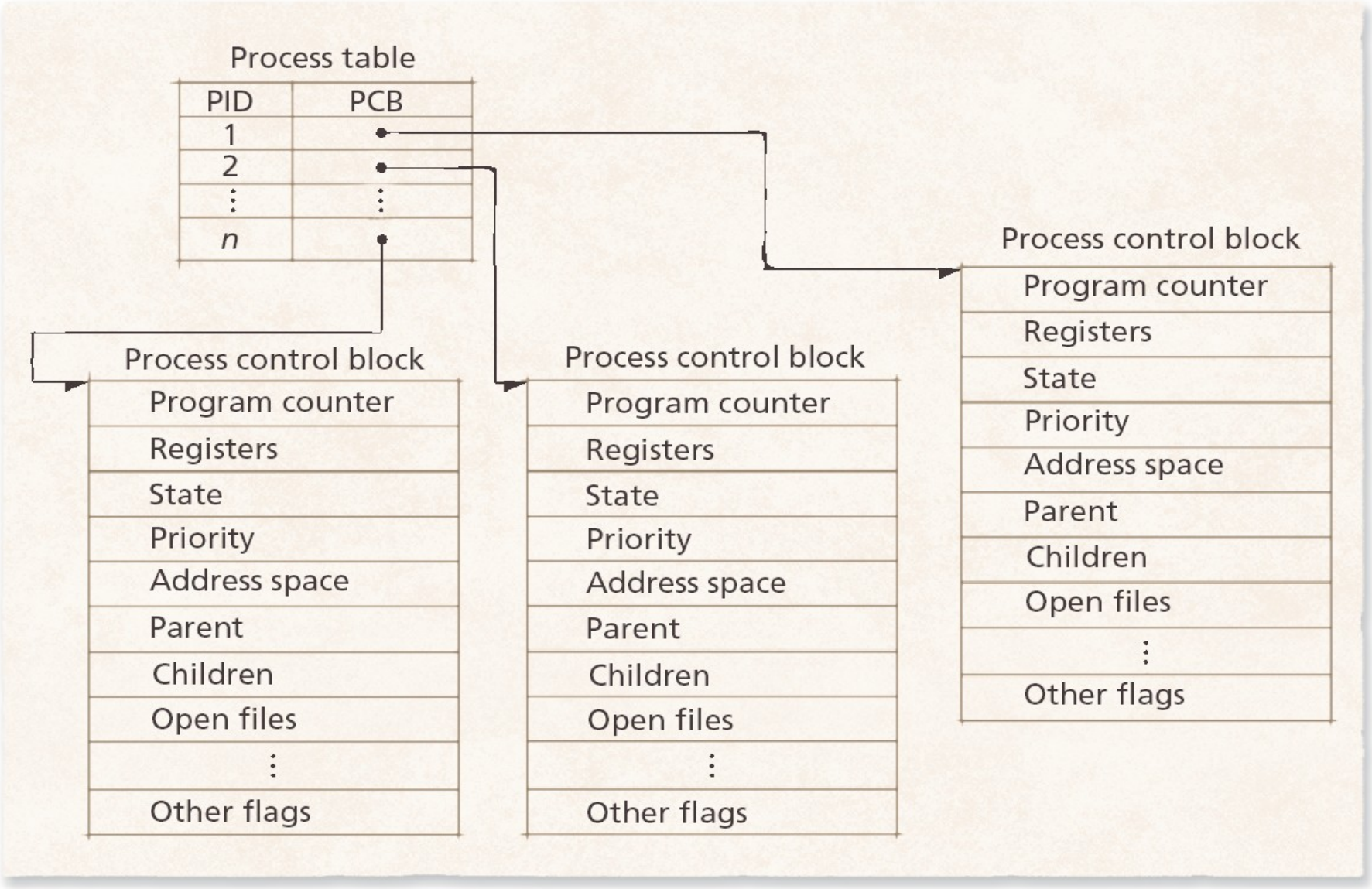
- Estado
- Contador de Programa
- Registros de la CPU (acumuladores, indices,punteros)
- Itineración de la CPU(prioridad, punteros a colas de planificación)
- Administración de la Memoria (reg base y límite)
- Contabilidad (accounting)
- Estado de la E/S

4.1.3 Bloque de Control de Proceso

Bloque de Control de Proceso (PCB)

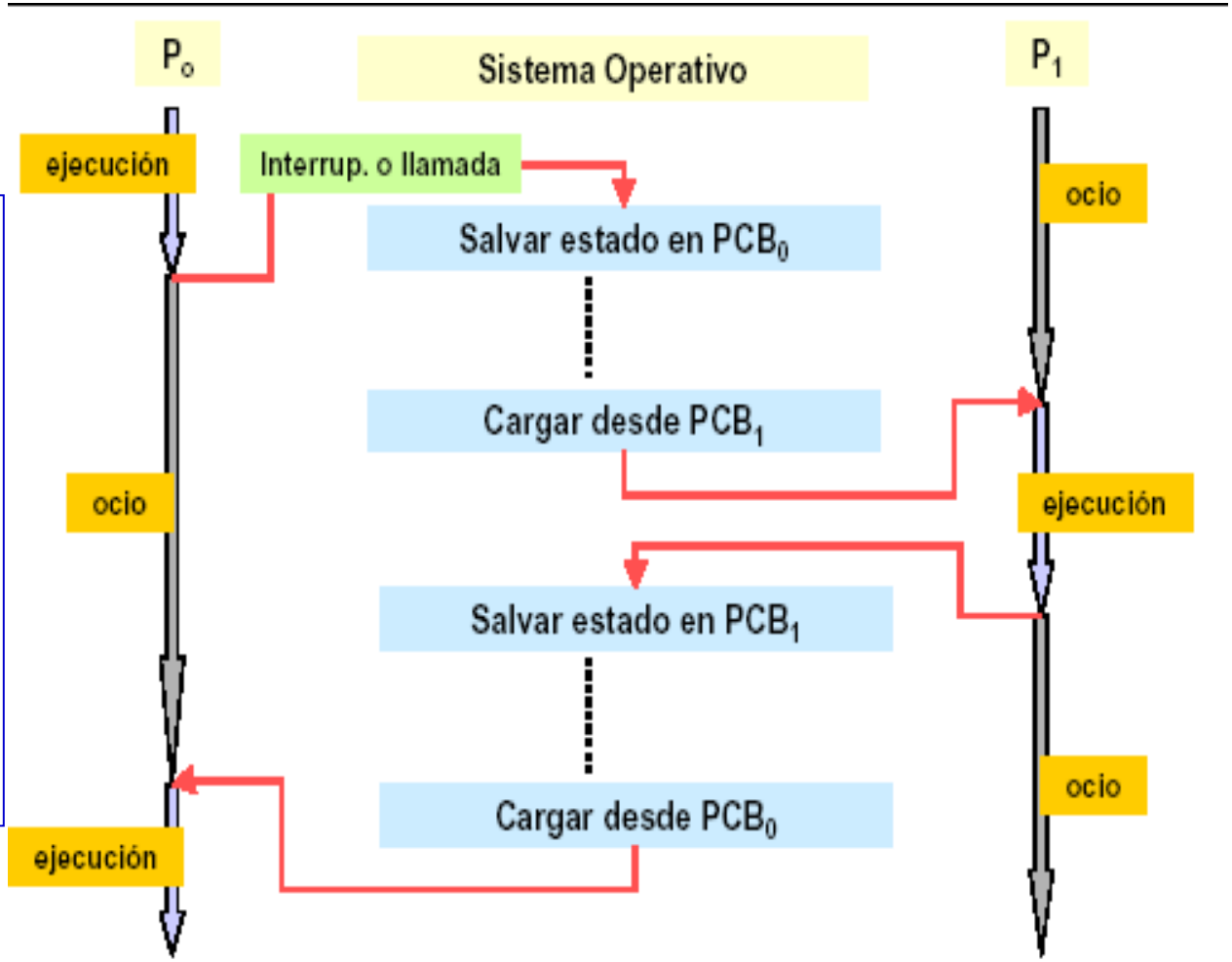


4.1.3 Bloque de Control de Proceso



4.1.3 Bloque de Control de Proceso

el PCB refleja el cambio de la CPU de un proceso a otro (conmutación)

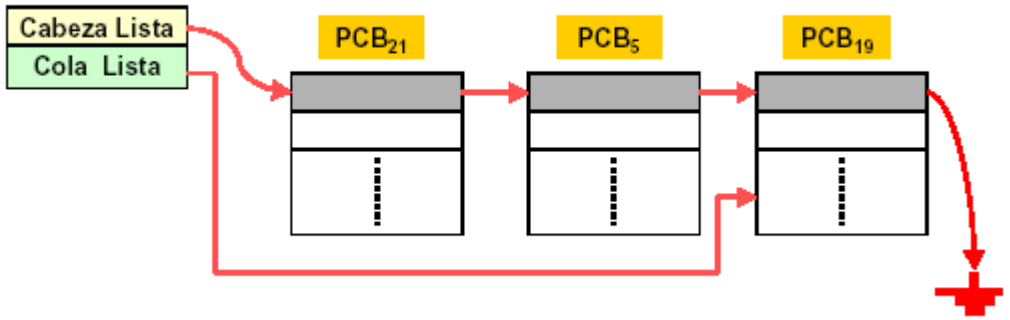


4.2 ITINERACIÓN DE PROCESOS

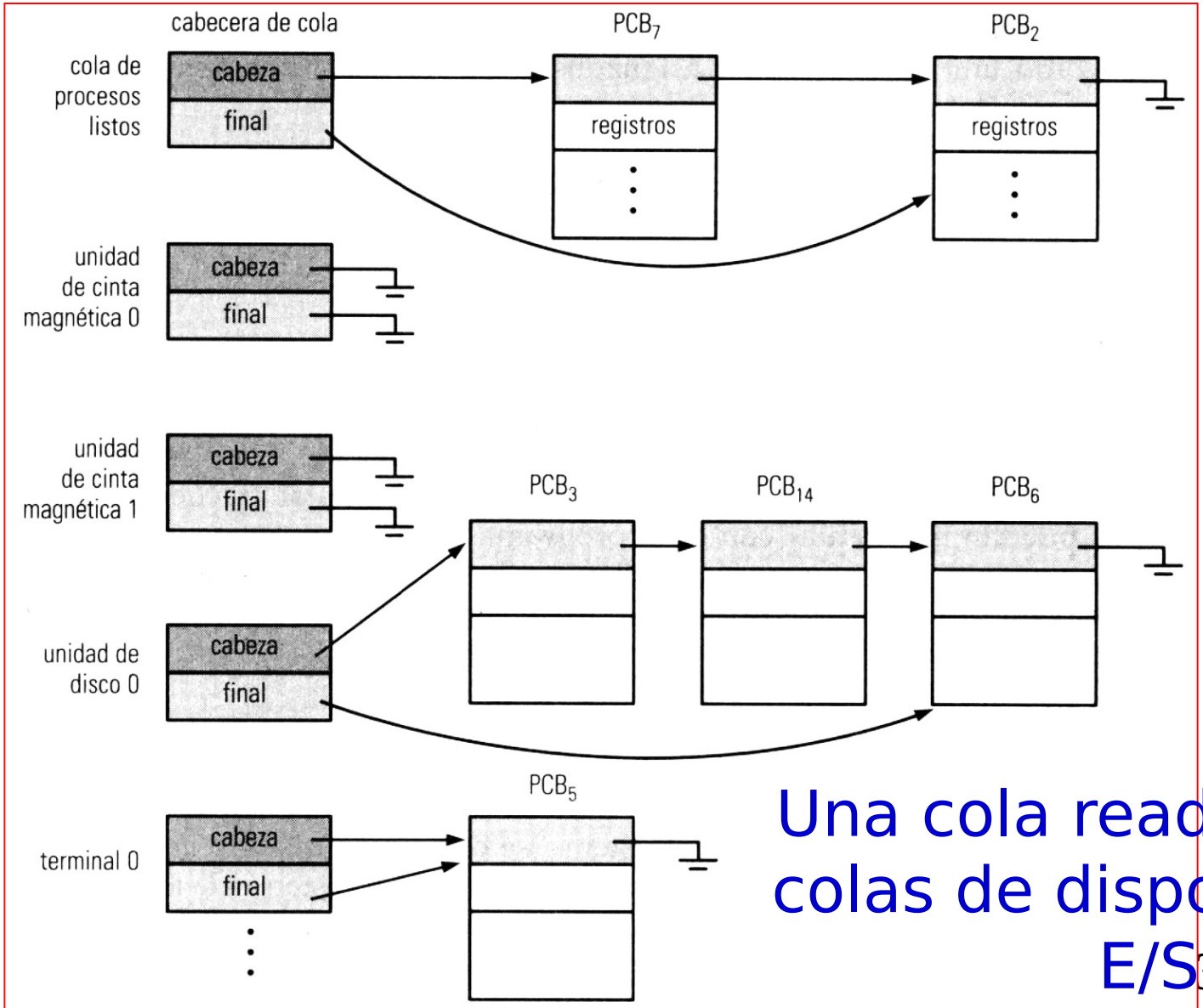
1 Colas de Itineración
2 Itineradores
3 Cambio de Contexto
4 Procesamiento de Interrupciones

4.2.1 Colas de Itineración

- **Colas de jobs**: conjunto de todos los procesos en el sistema.
- **Cola Ready**: conjunto de todos los procesos que residen en memoria principal, listos para ejecutarse.
- **Colas de Dispositivos**: conjunto de todos los procesos que están esperando por un dispositivo de E/S.
- Migración de procesos entre las distintas colas.

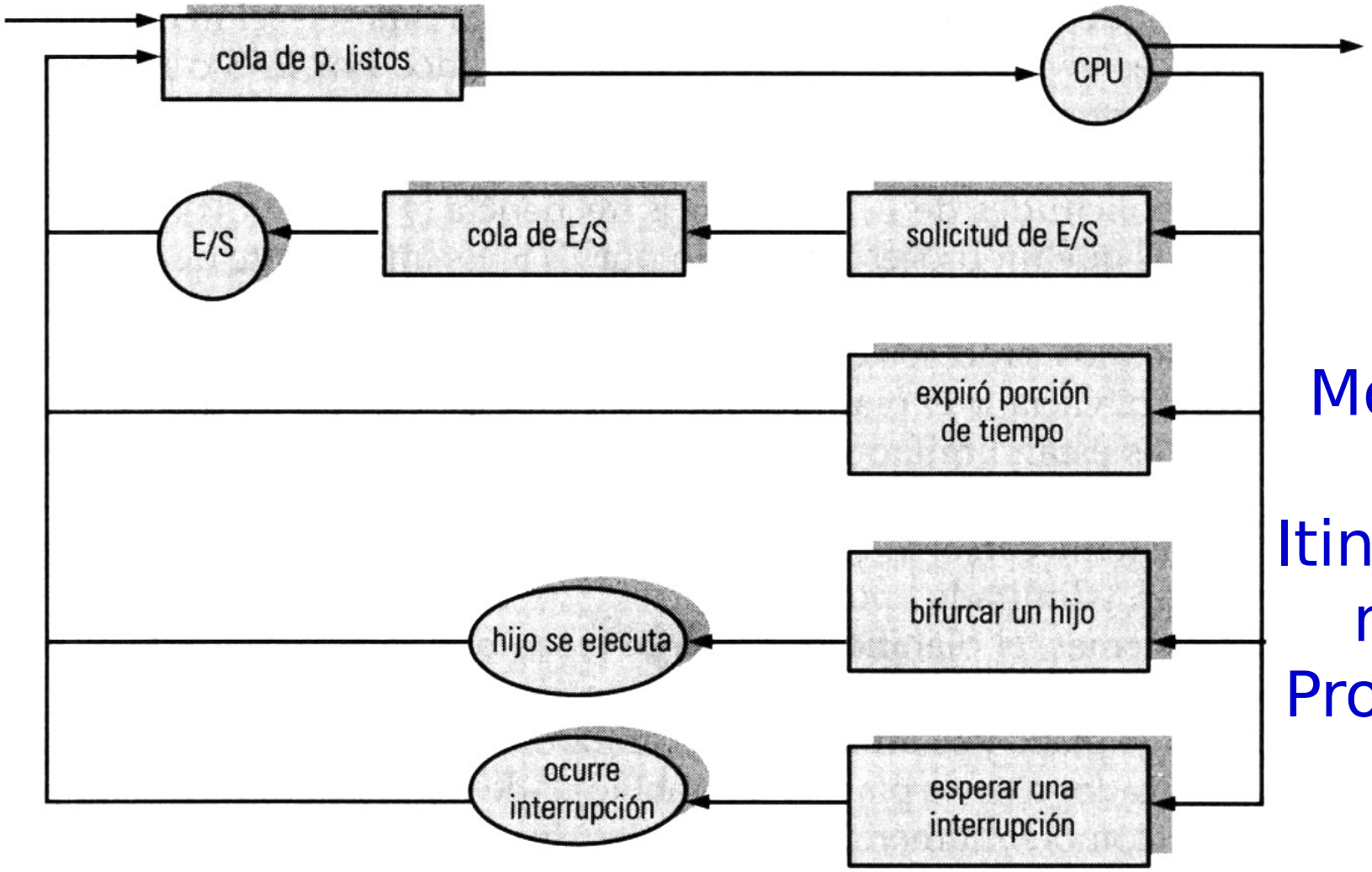


4.2.1 Colas de Itineración



Una cola ready y varias colas de dispositivos de E/S

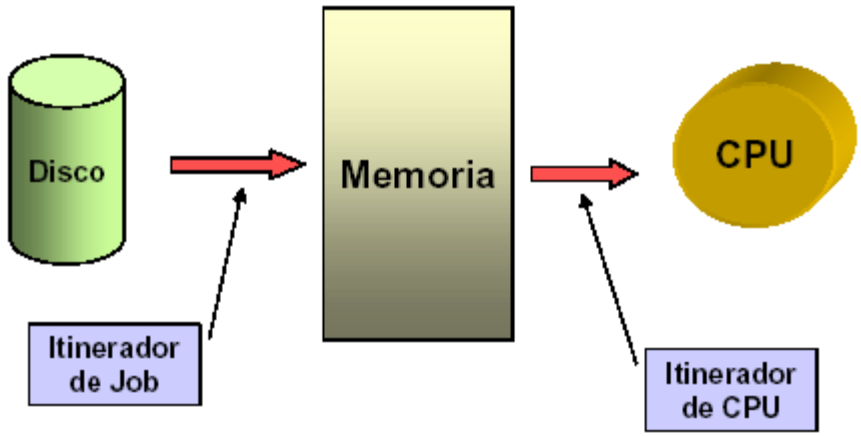
4.2.1 Colas de Itineración



Modelo de Itineración de Procesos

4.2.2 Itineradores

- Itinerador de **largo plazo** (o itinerador de jobs): selecciona qué procesos serán traídos a la cola ready.
- Itinerador de **corto plazo** (o itinerador de la CPU): selecciona qué proceso (de la cola ready) será ejecutado a continuación y le asigna la CPU.

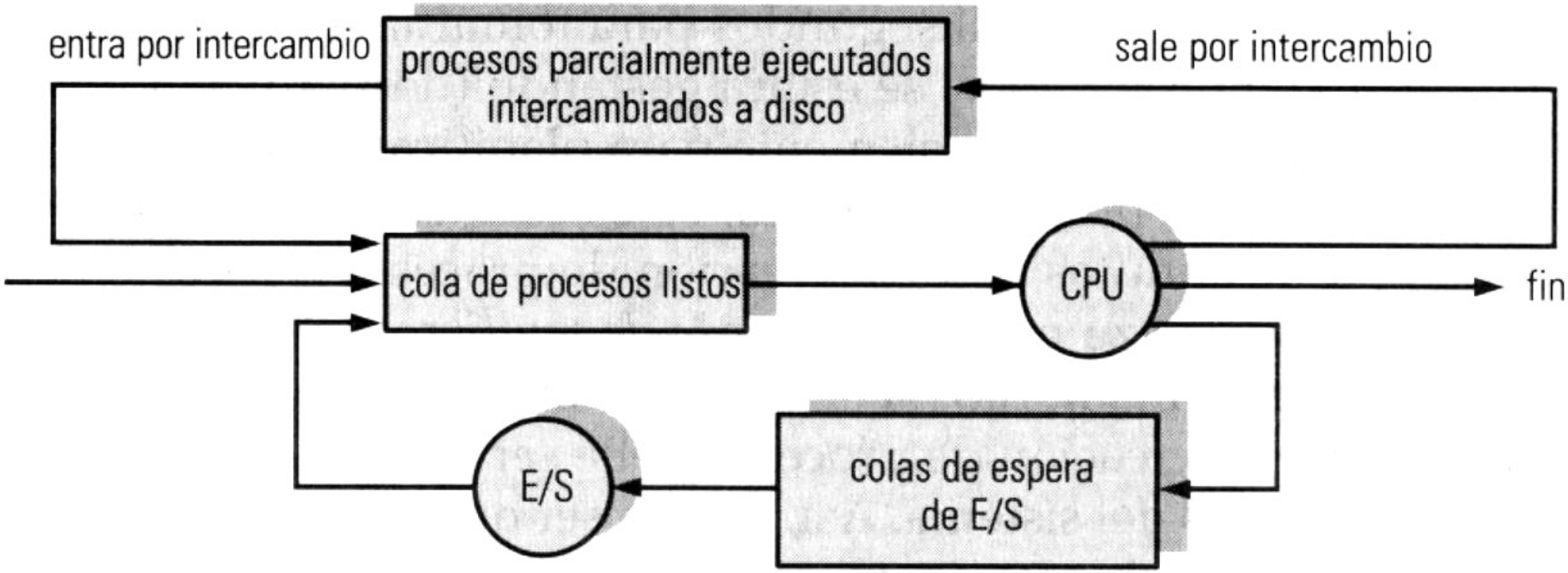


4.2.2 Itineradores

- El itinerador de corto plazo es invocado muy frecuentemente (miliseg) ⇒ (debe ser rápido).
- El itinerador de largo plazo es invocado poco frecuentemente (seg, min) ⇒ (puede ser lento).
- El itinerador de largo plazo controla el grado de multiprogramación.
- Los procesos pueden ser descritos ya sea:
  - Limitados por la E/S: ocupan más el tiempo haciendo E/S que cálculos; tienen muchas y muy pequeñas ráfagas de CPU (CPU Bursts).
  - Limitados por la CPU: ocupan más el tiempo haciendo cálculos; tienen pocas y extensas ráfagas de CPU.

4.2.2 Itineradores

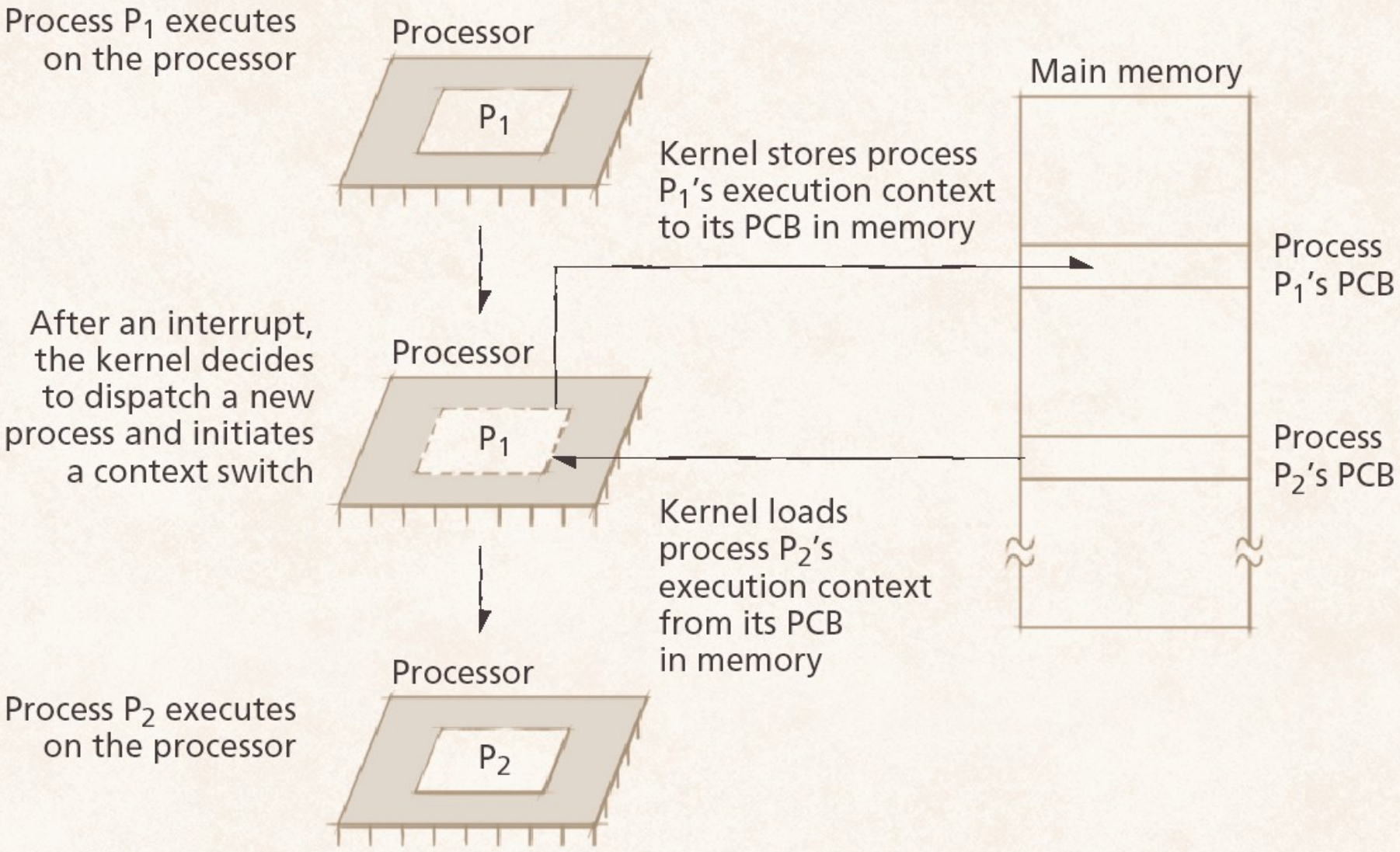
Agregación de un Itinerador de Mediano Plazo



4.2.3 Cambio de Contexto

- Cuando la CPU cambia a otro proceso, el sistema debe salvar el estado del antiguo proceso y cargar el estado (previamente salvado) del nuevo proceso.
- El tiempo de cambio de contexto es *overhead*; el sistema no hace trabajo útil mientras hace el cambio.
- El tiempo depende del soporte de hardware.

4.2.3 Cambio de Contexto



4.3 OPERACIONES SOBRE PROCESOS

1 Creación

2 Terminación

4.3.1 Creación de un Proceso

- Un proceso padre crea procesos hijos, los cuales, a su vez, crean otros procesos, formando un árbol de procesos.
- La compartición de recursos puede ser :
  - Padre e hijos comparten todos los recursos.
  - Los hijos comparten un subconjunto de los recursos del padre.
  - Padre e hijos no comparten los recursos.
- La ejecución puede ser :
  - Padre e hijo se ejecutan concurrentemente.
  - El padre espera a que los hijos terminen.

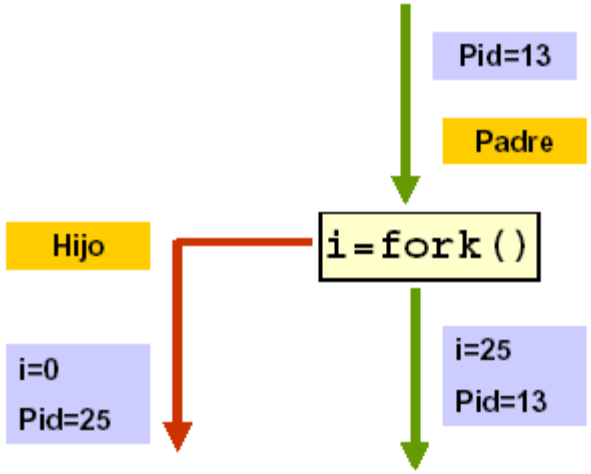
4.3.1 Ej. Creación de un Proceso

- En cuanto al espacio de direcciones:
  - El hijo es una duplicación del padre.
  - El hijo tiene su propio espacio de direcciones.
- Ejemplos de UNIX
  - La llamada al sistema *fork* crea un nuevo proceso.
  - La llamada al sistema *execute* es usada después de *fork* para reemplazar el espacio de memoria del proceso con un nuevo programa

- La llamada al sistema fork tiene la siguiente estructura:

```
i = fork();
```

$$i = \begin{cases} 0 & \text{Si es proceso hijo} \\ \text{pid del hijo si es padre} \end{cases}$$



4.3.1 Creación de un Proceso

```
{
    pid_t pid;

    if ( (pid=fork()) == 0 )
    { /* hijo */
        printf("Soy el hijo (%d, hijo de %d)\n", getpid(),
        getppid());
    sleep(10);
    }
    else
    { /* padre */
        printf("Soy el padre (%d, hijo de %d)\n", getpid(),
        getppid());
    sleep(10);
    }
```

4.3.1 Creación de un Proceso

Árbol de procesos en un sistema UNIX típico

4.3.2 Término de un Proceso

- El proceso ejecuta su última sentencia y solicita al SO ser eliminado (salida normal o exit).
  - Envía los datos desde el hijo al padre (vía wait).
  - Los recursos del proceso son desasignados por el sistema operativo.
- Un padre puede terminar la ejecución de los procesos hijos (término anormal o abort), porque:
  - El hijo ha excedido los recursos asignados.
  - La tarea asignada al hijo ya no es requerida.
  - El padre está terminando.
    - El sistema operativo no permite que un hijo continúe si su padre termina.
    - Terminación en cascada.

4.4 PROCESOS COOPERATIVOS

- Un proceso **independiente** no puede afectar o ser afectado por la ejecución de otro proceso.
- Un proceso **cooperativo** puede afectar o ser afectado por la ejecución de otro proceso.
- Ventajas de los procesos cooperativos:
  - Compartir información
  - Aumento en la velocidad de computación
  - Modularidad
  - Conveniencia
- En el paradigma de los procesos cooperativos; un proceso *productor* produce información, la cual es consumida por un proceso *consumidor*.

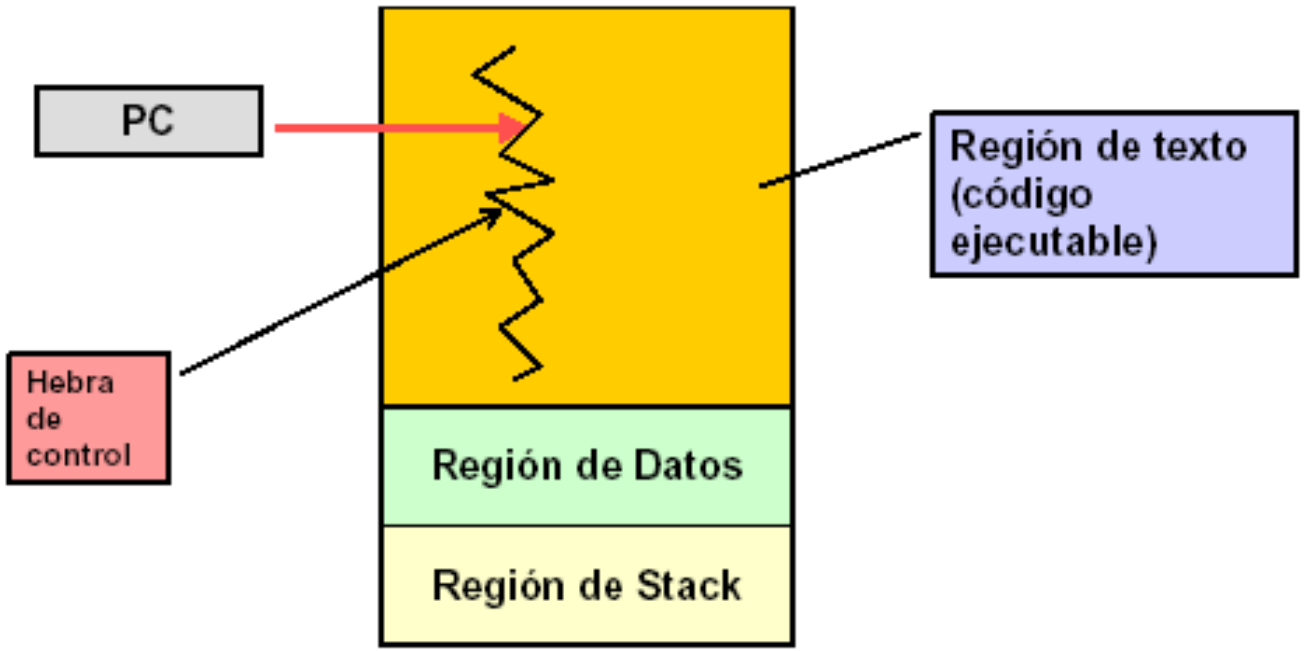
4.5 HEBRAS (THREADS)

- Una hebra (o *proceso liviano*) es una unidad básica de utilización de CPU; consiste de:
  - program counter
  - registros
  - stack
- Una hebra comparte con sus pares:
  - sección de código
  - sección de datos
  - recursos del sistema operativo

  Es conocido colectivamente como tarea (*task*).
- Un proceso tradicional o *pesado* es igual a una tarea con una sola hebra.

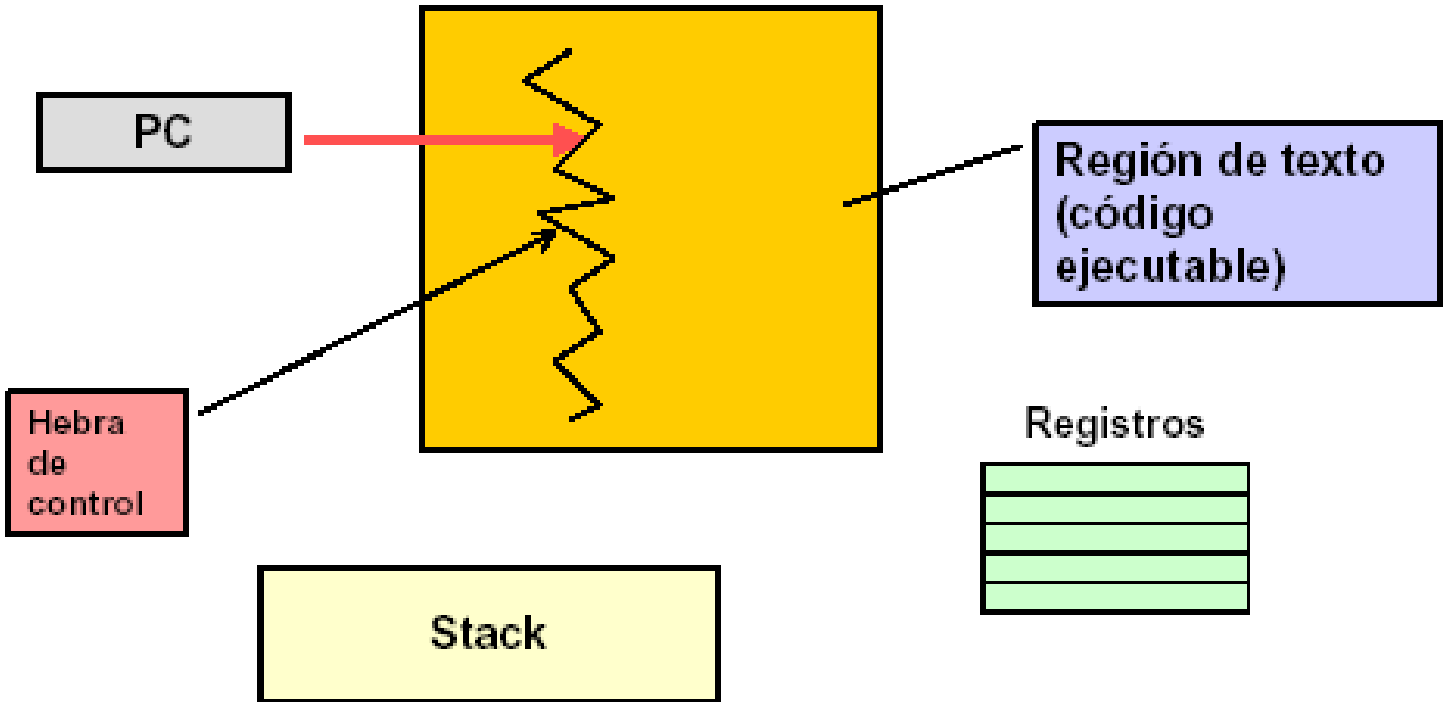
HEBRAS

Un proceso normal tiene la siguiente estructura



HEBRAS (Thread)

Un Thread tiene los siguientes recursos:

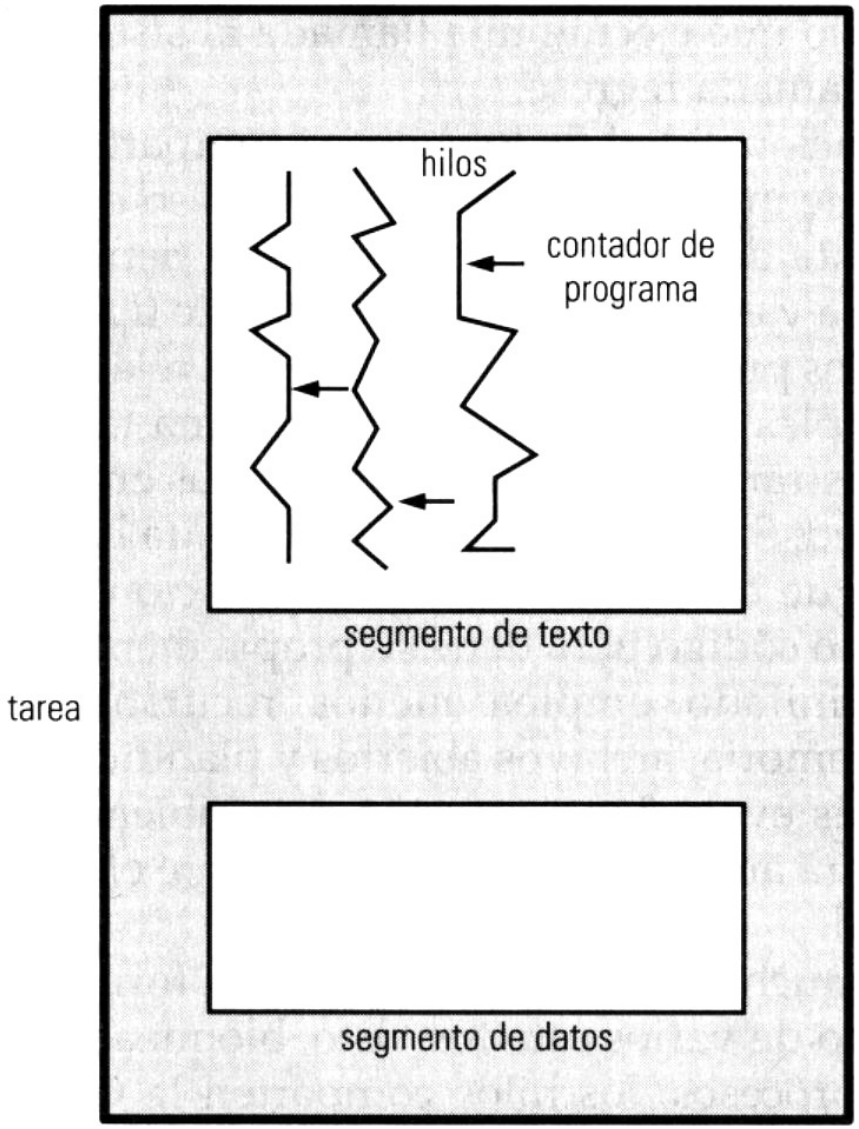


4.5 HEBRAS (THREADS)

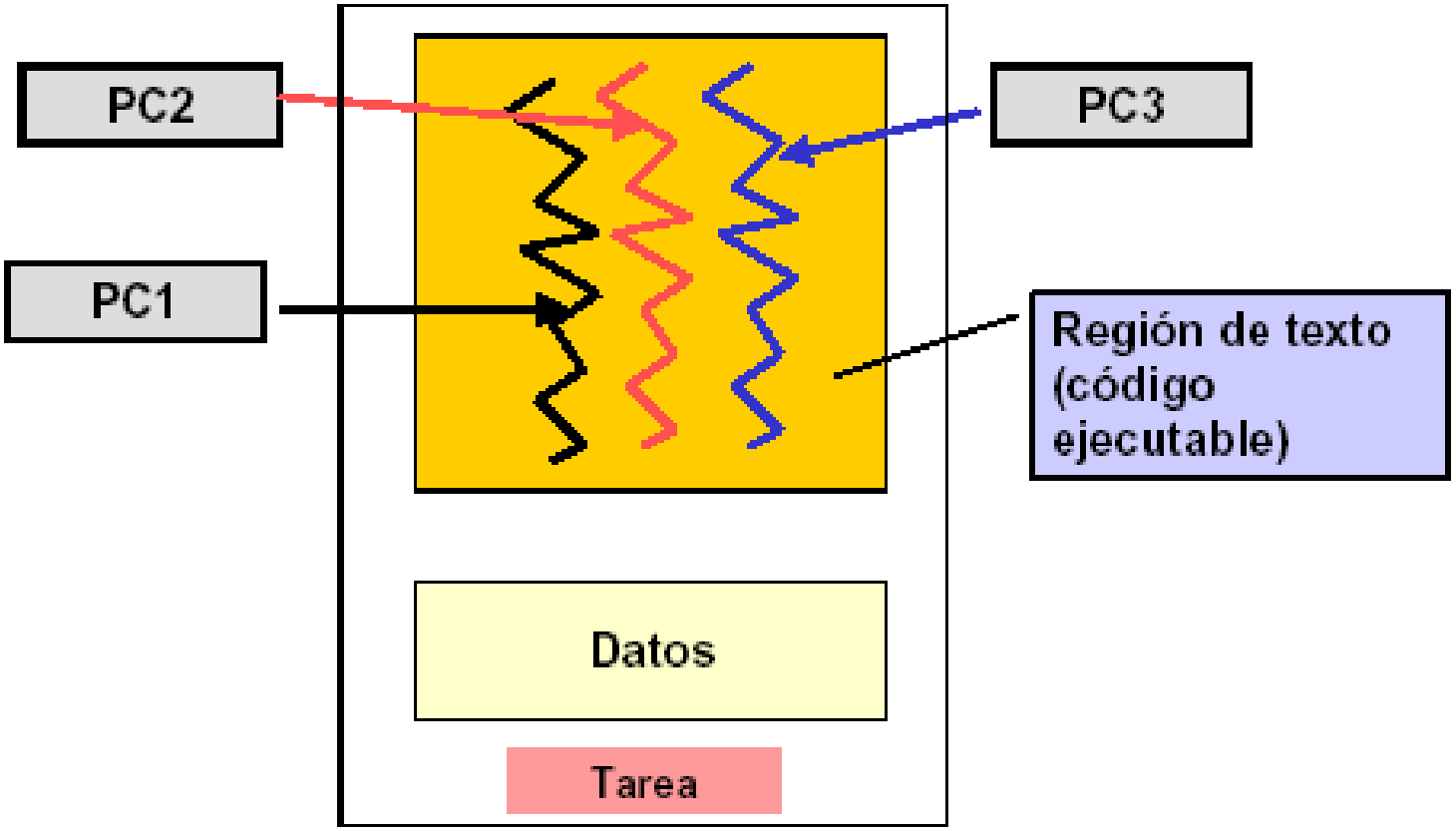
- En una tarea con múltiples hebras, mientras una está bloqueada, otra de la misma tarea puede ejecutarse.
- La cooperación de múltiples hebras en el mismo job da un throughput y rendimiento mayor.
- Las hebras proveen un mecanismo que permite a procesos secuenciales hacer llamadas al sistema con bloqueo y también lograr paralelismo.
- Hebras soportadas por kernel (Mach y OS/2).
- Hebras de nivel de usuario; por encima del kernel, via llamadas a biblioteca a nivel de usuario (Andrew).
- Híbrido que implementa ambos tipos (Solaris).

4.5 HEBRAS (THREADS)

Múltiples hebras dentro de una tarea



4.5 HEBRAS (THREADS)



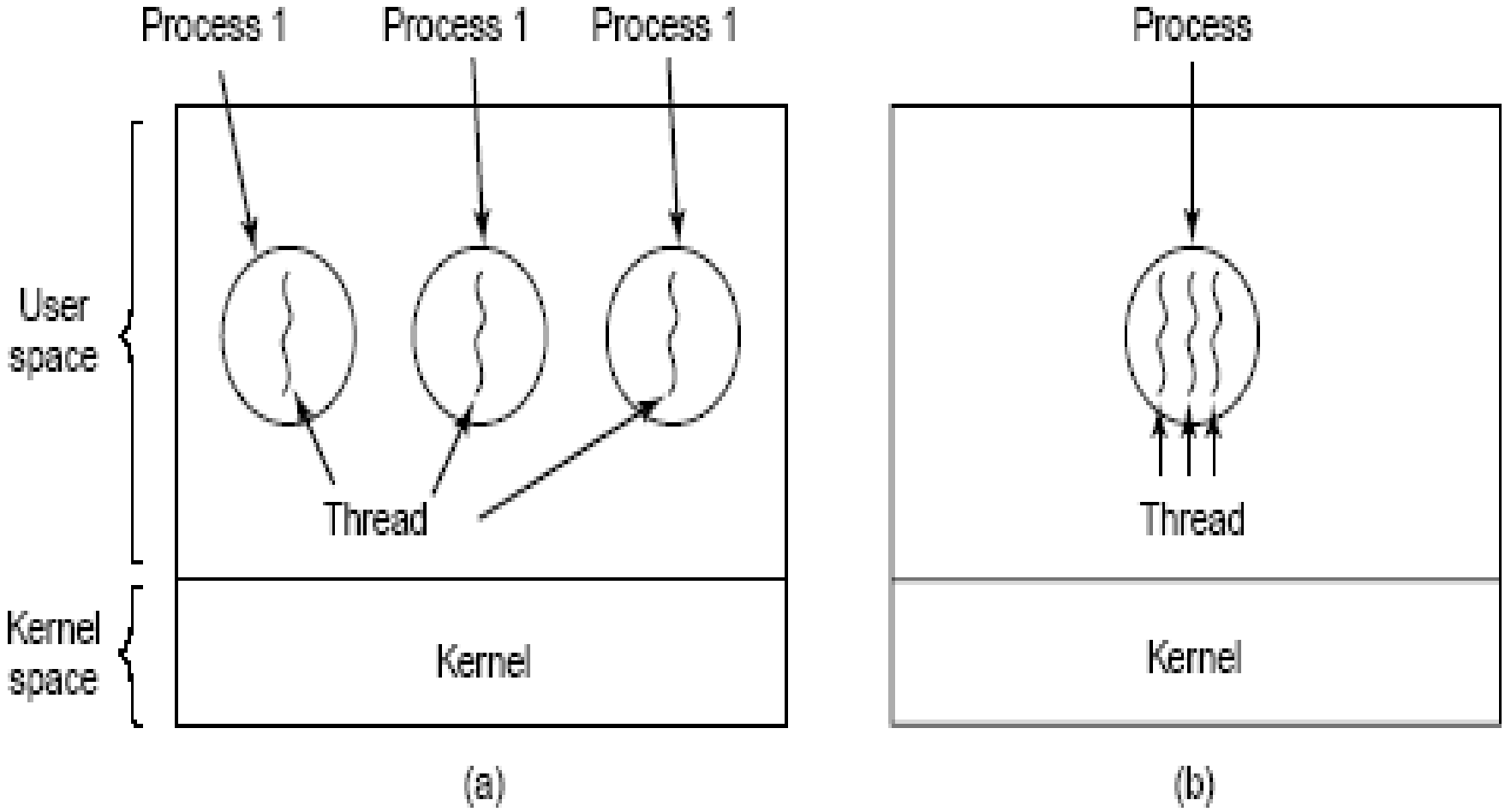


Figure 2-1. (a) Three processes each with one thread. (b) One process with three threads.

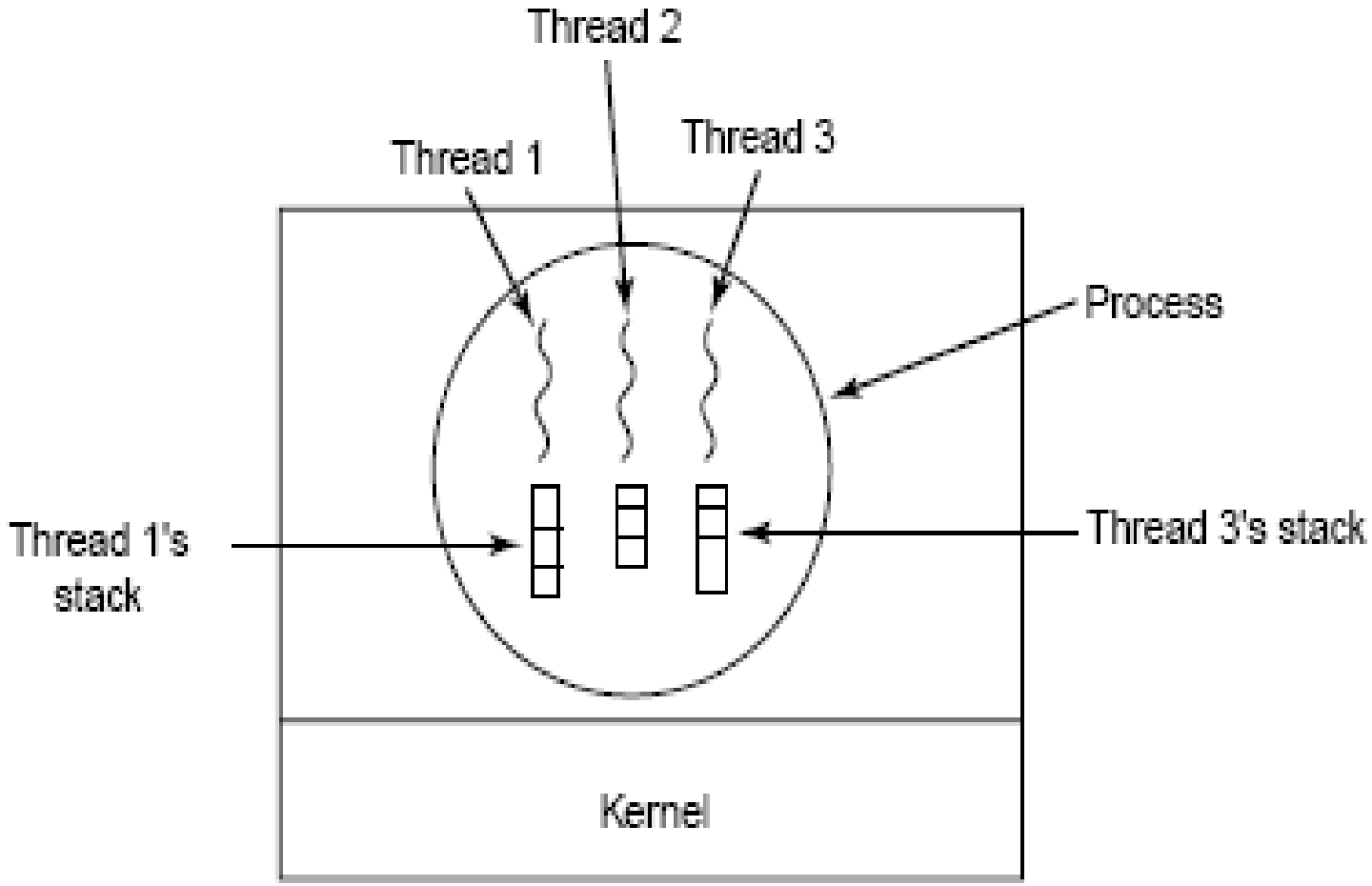


Figure 2-3. Each thread has its own stack.

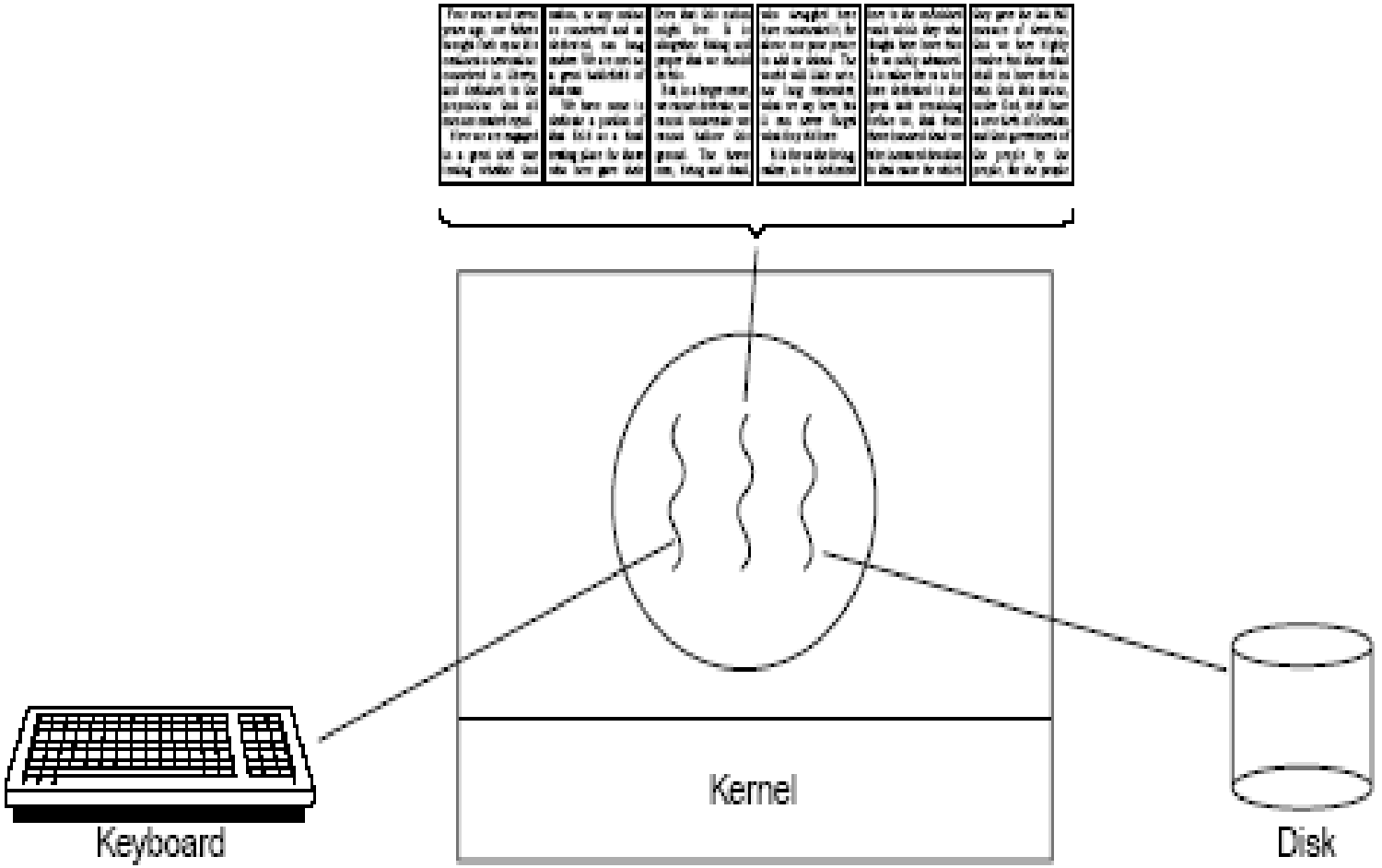


Figure 2-4. A word processor with three threads.

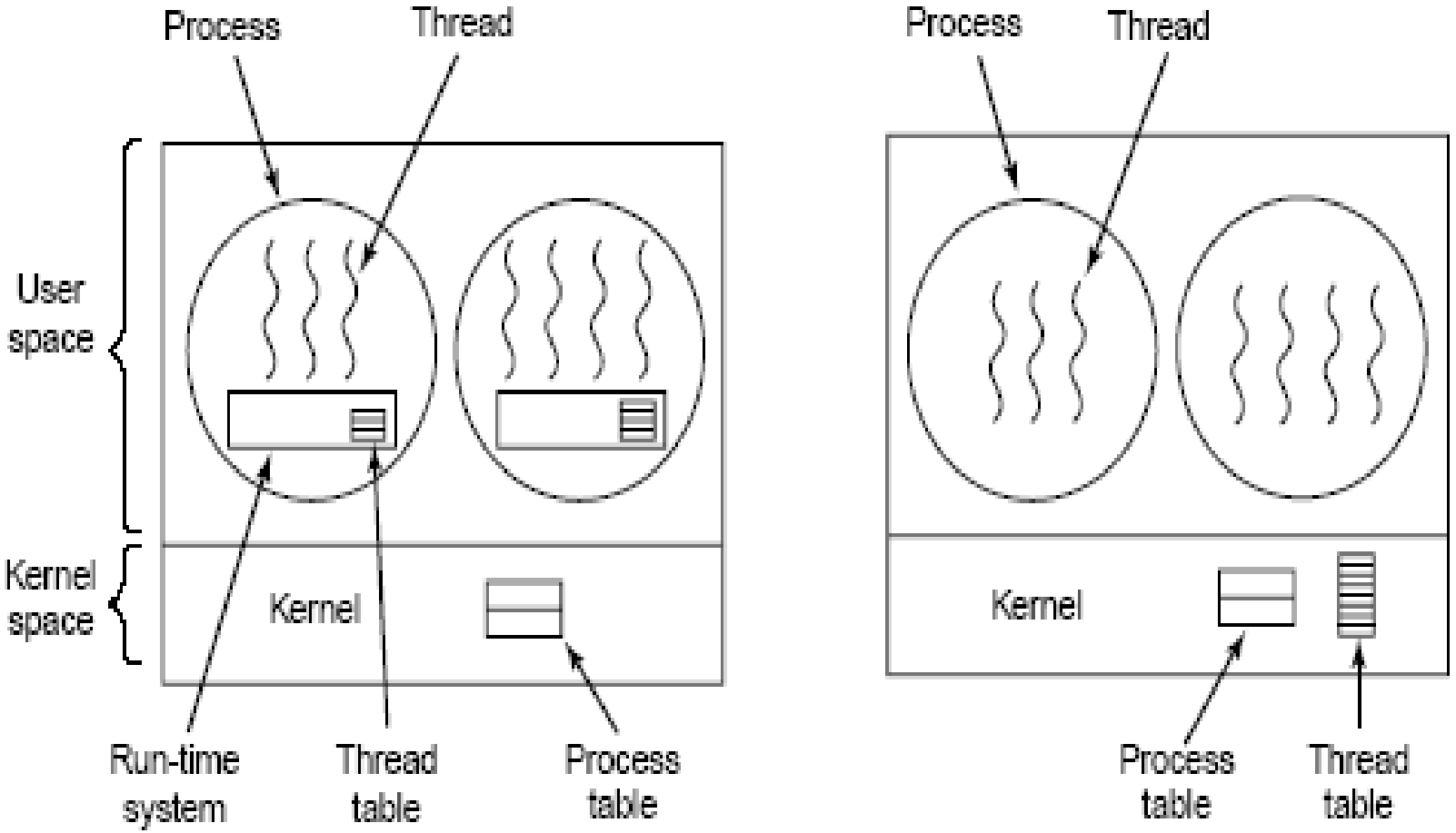


Figure 2-8. (a) A user-level threads package. (b) A threads package managed by the kernel.

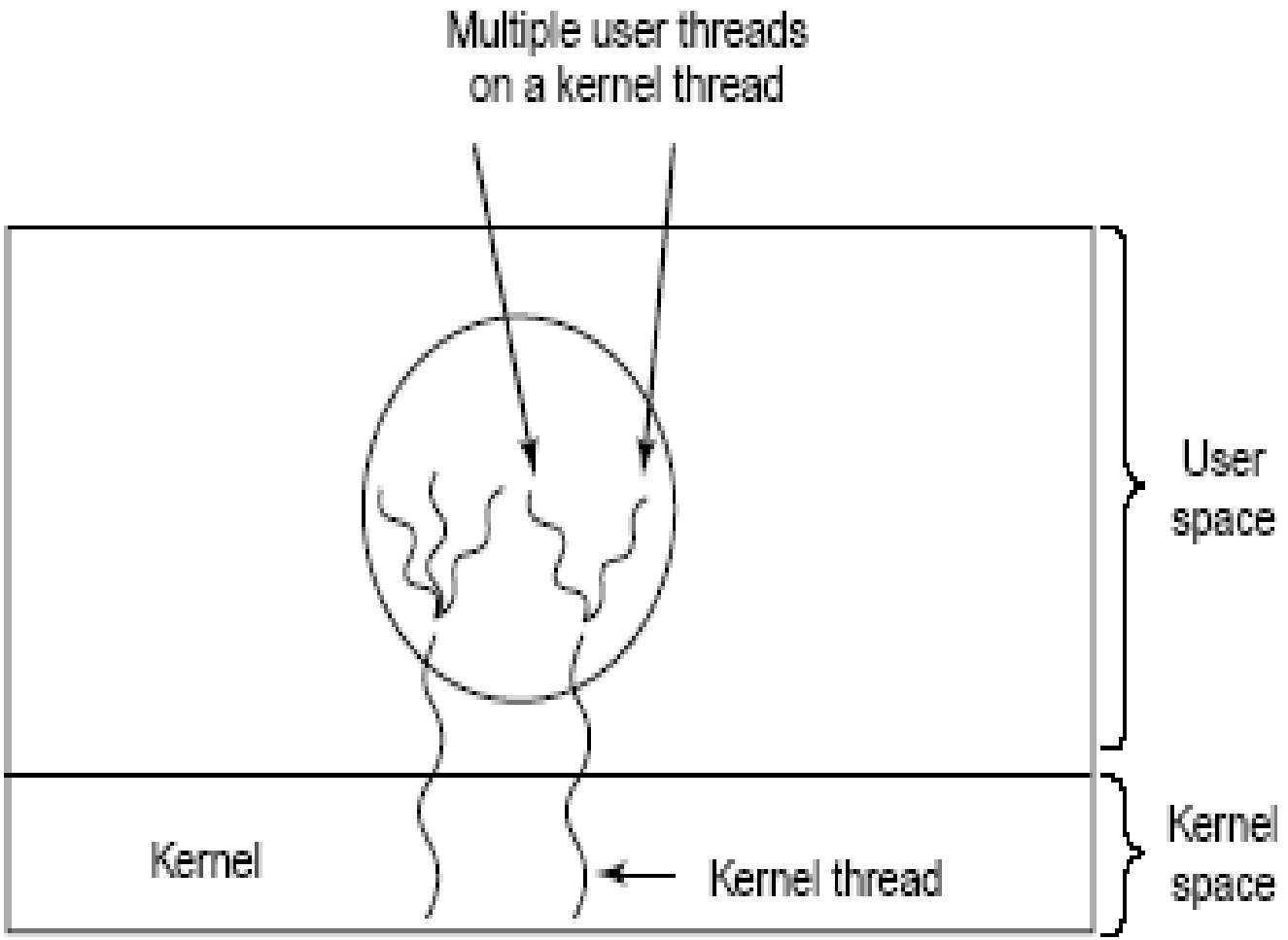


Figure 2-9. Multiplexing user-level threads onto kernel-level threads.

Comunicación entre procesos

- Los procesos cooperativos pueden comunicarse en un entorno de memoria compartida.
- El S.O. ofrece un mecanismo de comunicación entre procesos IPC (interprocess communication)
- Esquema de memoria compartida y transferencia de mensajes no son mutuamente exclusivos (dentro de un mismo S.O. o un mismo proceso)

Estructura básica

- Sistema de mensajes: Sin recurrir a variables compartidas.
- Al menos dos operaciones:
  - Enviar (mensaje) o send
  - Recibir (mensaje) o receive

Estructura Básica

- Si P y Q desean comunicarse, necesitan:
  - Establecer un enlace de comunicación entre ellos
  - Intercambiar mensajes vía send/receive
- Métodos para implementar un enlace:
  - Comunicación directa o indirecta
  - Comunicación simétrica o asimétrica
  - Uso de Buffers
  - Envío por copia o por referencia
  - Mensajes de tamaño fijo o variable

Comunicación Directa

- Cada proceso nombra explícitamente al destinatario o el remitente.
  - Enviar (P,mensaje)
  - Recibir (Q,mensaje)

- Propiedades:
  - Se establece automáticamente un enlace entre cada par de procesos
  - Un enlace se asocia a exactamente dos procesos
  - Entre cada par de procesos sólo existe un enlace
  - El enlace puede ser unidireccional, pero suele ser bidireccional

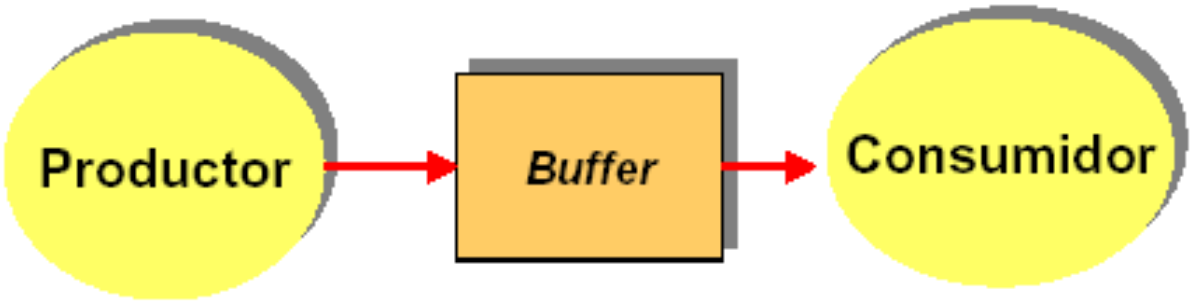
Problema Productor-Consumidor

- Productor:
  - Repeat
  - …
  - Producir un elemento
  - …
  - Enviar (consumidor,elemento);
  - Until false:

Consumidor:

- Repeat
- Recibir (productor,elemento);
- …
- Consumir el elemento
- …
- Until false:

Productor - Consumidor



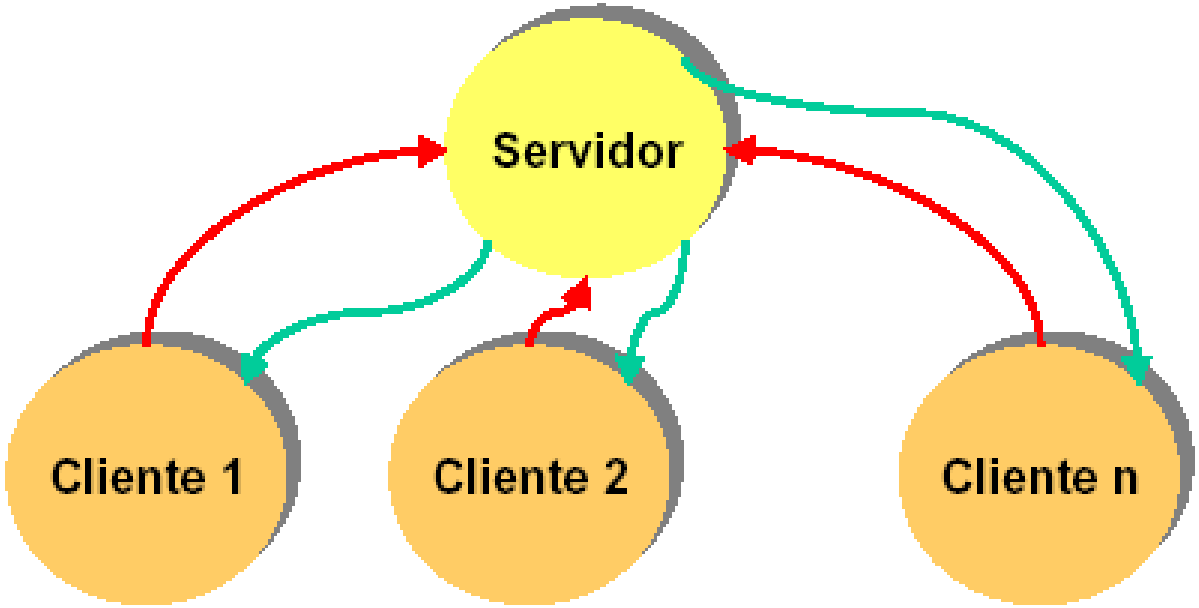
Como los procesos Productor y Consumidor tienen distintas velocidades, normalmente se necesita un buffer para suavizar esta diferencia. El IPC provee de este Buffer en forma automática

Simetría - Asimetría

- El esquema anterior es simétrico: procesos nombran al otro explícitamente para comunicarse
- Asimetría:
  - Enviar (P,mensaje)
  - Recibir (id, mensaje) id: nombre del proceso con el que hubo comunicación.
- Desventaja de ambos esquemas: poca modularidad, se cambia un nombre de proceso, revisar donde se haya nombrado ese proceso.

Asimetría

El esquema asimétrico permite un tipo de interacción llamado Cliente/Servidor



Comunicación Indirecta

- Los mensajes se envían y reciben de buzones o mailbox.
- Cada buzón tiene una identificación única
  - Enviar (A, mensaje) enviar un mensaje al buzón A
  - Recibir (A, mensaje) recibir un mensaje del buzón A

Comunicación Indirecta

- Se establece un enlace entre un par de procesos si tienen un buzón compartido
- Un enlace puede estar asociado a más de dos procesos
- Cada par de procesos puede compartir varios enlaces (buzones)
- El enlace puede ser unidireccional o bidireccional

Comunicación Indirecta

- Compartir buzones:
  - P1, p2, p3 comparten el buzón A
  - P1 envía; p2 y p3 reciben
  - ¿quién recibe el mensaje?
- Soluciones:
  - Enlace asociado con a lo más, dos procesos
  - Sólo un proceso a la vez ejecute una operación receive
  - El sistema selecciona arbitrariamente al receptor. El remitente es notificado de quién fue el receptor.

S.O. establece un mecanismo que permite a un proceso:

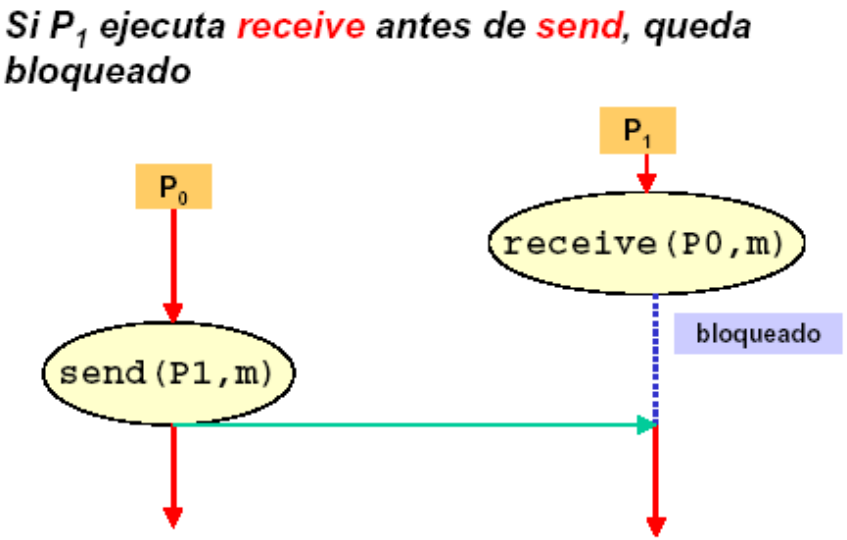
- Crear un buzón nuevo
- Enviar y recibir mensajes a través del buzón
- Destruir un buzón

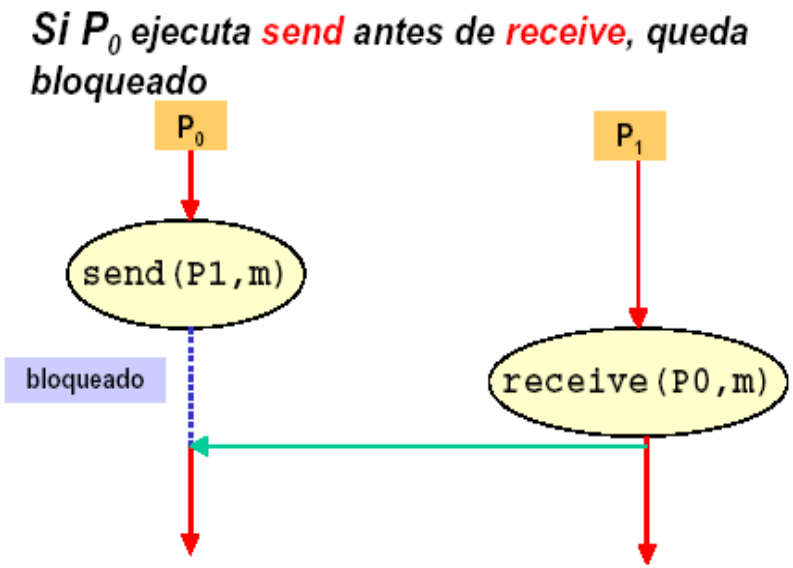
Uso de Buffers

- Cola de mensajes unida al enlace, se implementa como:
  - Capacidad cero: longitud 0; procesos deben sincronizarse (encuentro o rendezvous) (sin buffers)
  - Capacidad limitada: longitud n; si está lleno el emisor deberá esperar hasta que haya espacio libre en la cola. (almacenamiento automático en buffers)
  - Capacidad ilimitada: longitud infinita; el emisor nunca espera (almacenamiento automático en buffers)

Sincronización

- Capacidad cero: los procesos podrían quedar bloqueados

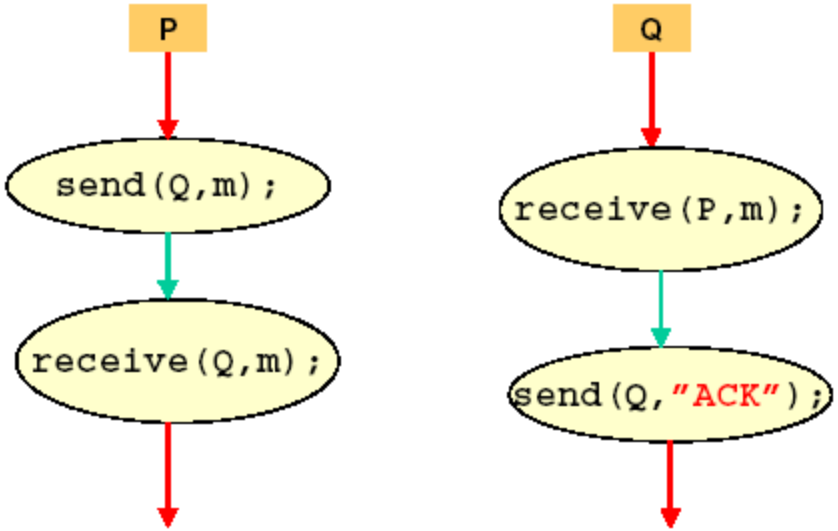




Sincronización

- Capacidad limitada e ilimitada: es necesario señalar cuando un mensaje ha llegado a su destino. Esto se puede hacer con un mensaje corto de ACUSE de recibo al emisor en cierto plazo de tiempo

¿qué pasa si se pierde este mensaje?



Mensajes perdidos o alterados

- El S.O. tiene la obligación de detectar este suceso y retransmitir el mensaje o avisar al emisor que el mensaje se perdió o se alteró. El proceso emisor decide qué hacer.

- El proceso emisor tiene la obligación de detectar este suceso y retransmitir, si desea.