Procesos

Planificación Centralizada y Distribuida

3

Sistemas Operativos y Distribuidos

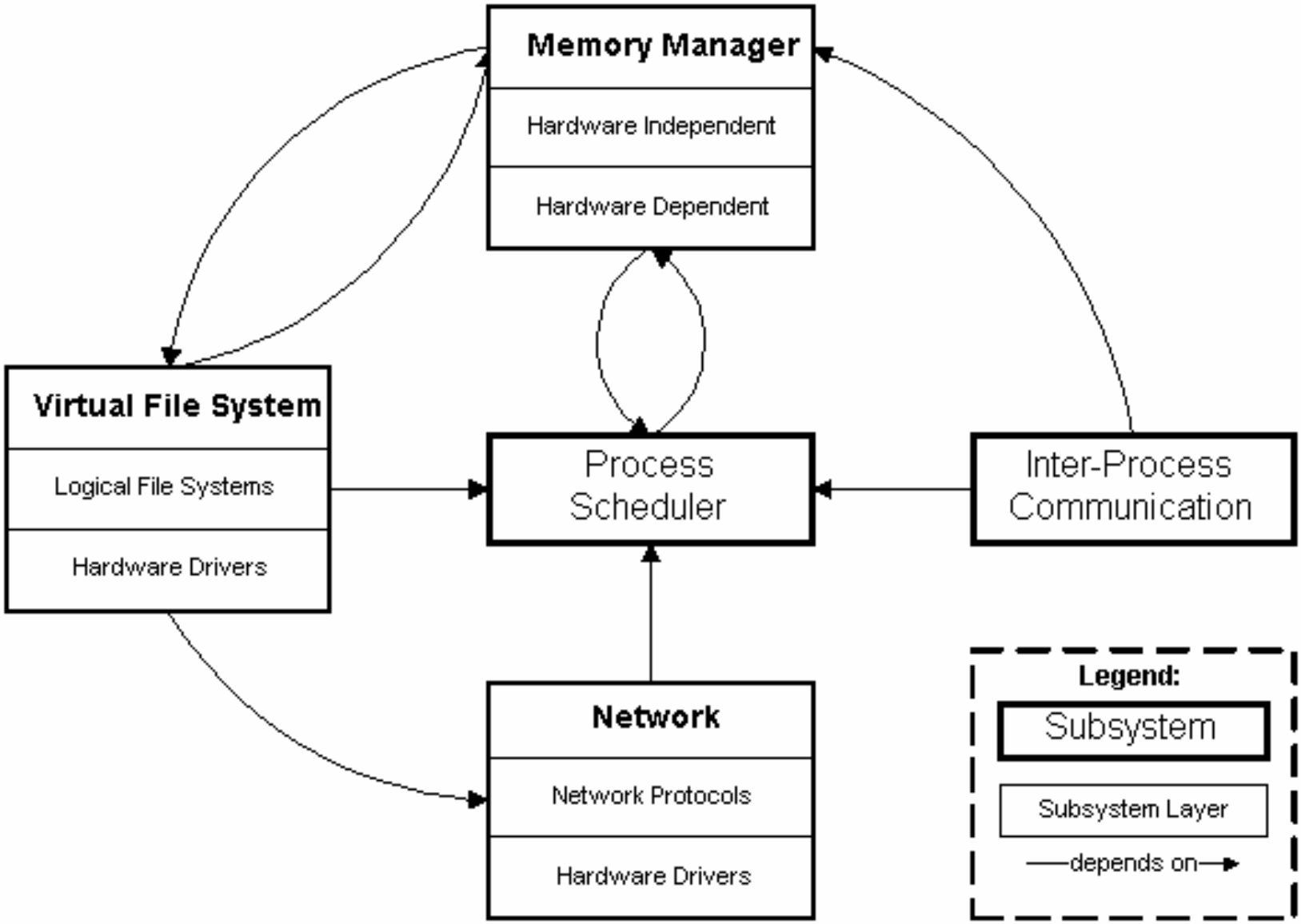
Mg. Javier Echaiz
D.C.I.C. – U.N.S.
http://cs.uns.edu.ar/~jechaiz
je@cs.uns.edu.ar

Estructura del Kernel

El kernel está conformado por 5 grandes subsistemas.

- El planificador de procesos (**sched**).
- El administrador de memoria (**mm**).
- El sistema del archivo virtual (**vfs**).
- La interfaz de red (**net**).
- La comunicación entre procesos (**ipc**).

Descomposición Conceptual



Concepto de Proceso

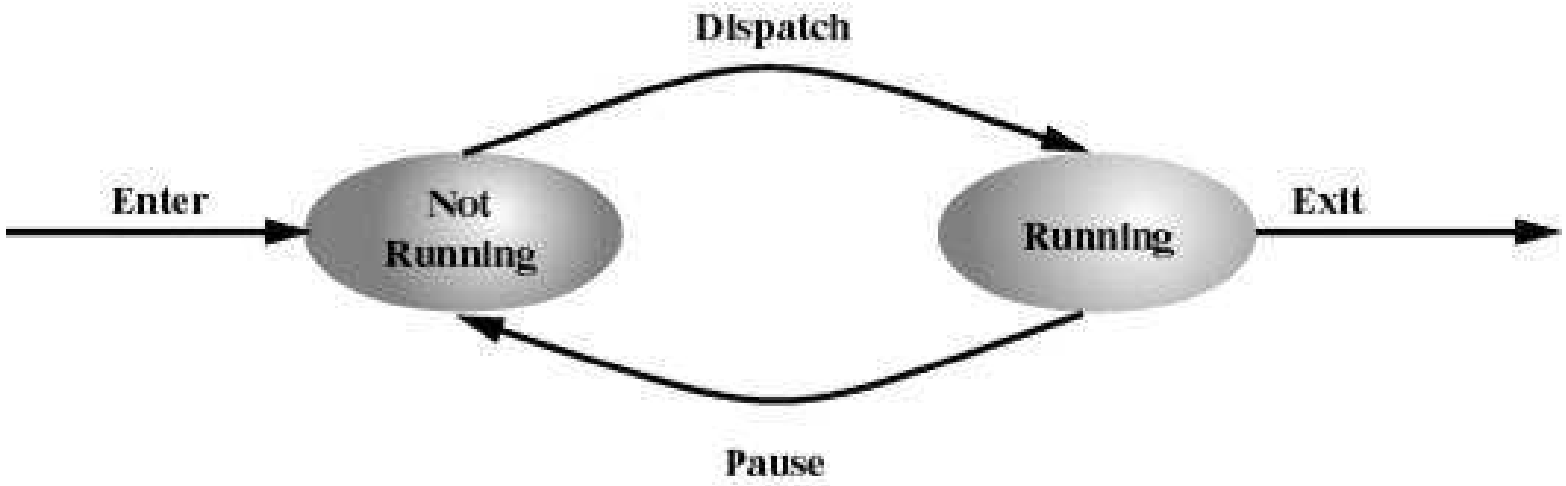
Un SO ejecuta una variedad de programas:

- Sistema Batch – jobs
- Sistemas de Tiempo Compartido – programas de usuario o tareas

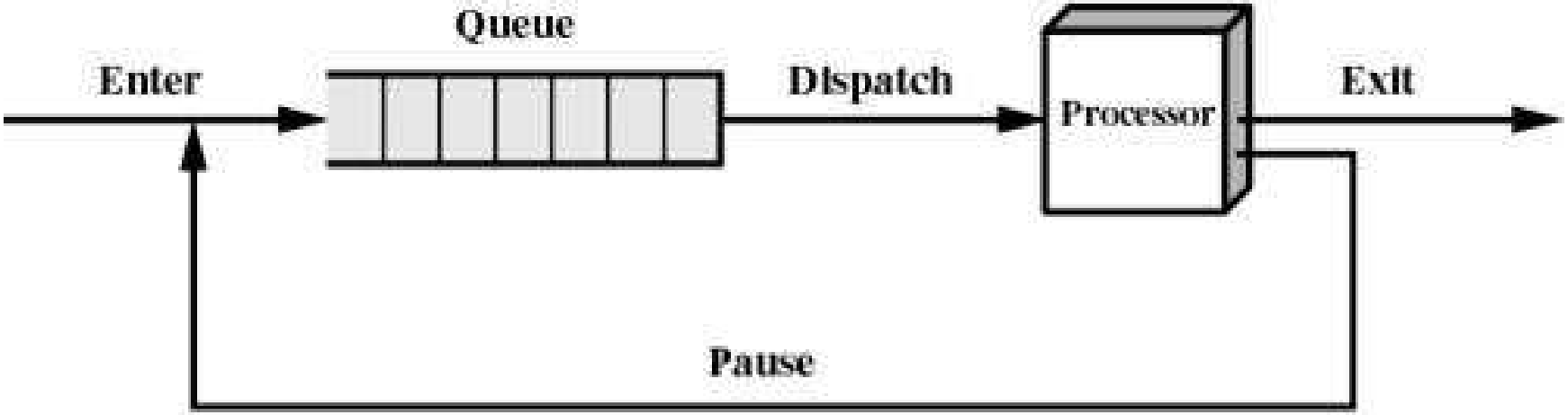
- Los términos *job* y *proceso* se usan con similar sentido.
- **Proceso** – un programa en ejecución; la ejecución de los procesos debe progresar en forma secuencial.
- Un proceso incluye:
  - contador de programa
  - stack
  - sección de datos

Modelo de Procesos con Dos Estados

- El SO controla la ejecución de los procesos (se necesita saber en qué estado se encuentra cada uno).
- Cada proceso puede tener dos estados:
  - En ejecución.
-



Cola de Procesos (dos estados)



Estado de un Proceso

- Mientras un proceso ejecuta, cambia de *estado*
  - nuevo: el proceso es creado.
  - corriendo: las instrucciones están siendo ejecutadas.
  - Espera/bloqueado: el proceso está esperando que ocurra algún evento.
  - listo: el proceso está esperando ser asignado a la CPU.
  - terminado: el proceso ha finalizado su ejecución.

Diagrama de Estados de un Proceso (1)

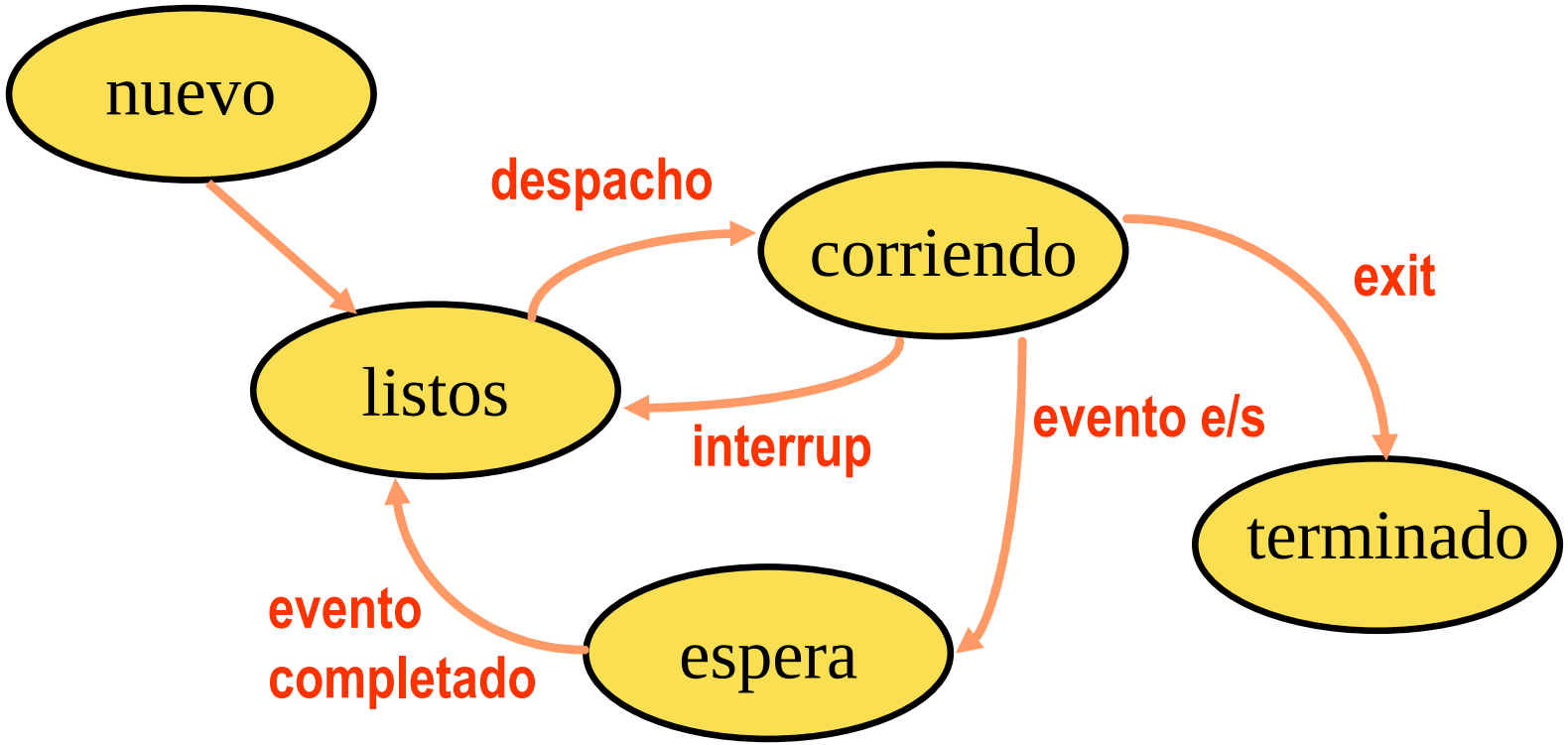
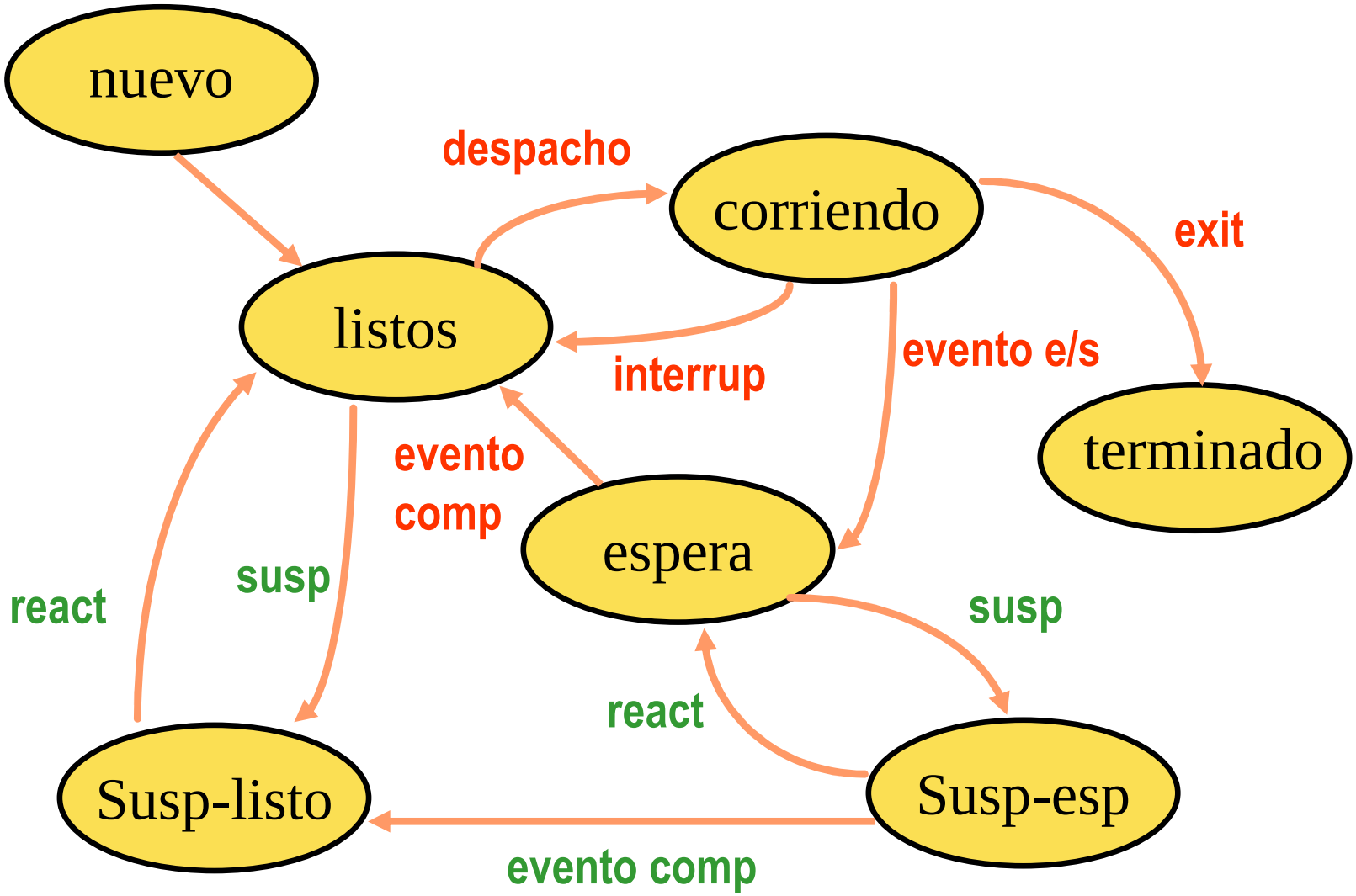


Diagrama de Estados de un Proceso (2)

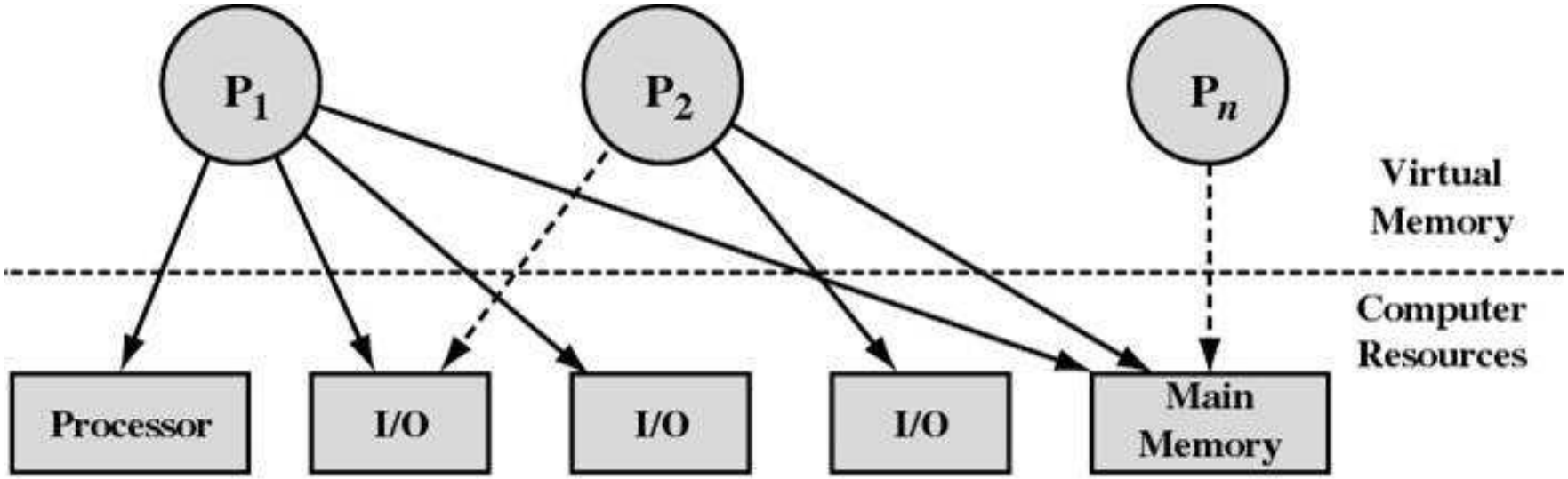


Razones para suspender procesos

- Intercambio (swapping).
  - El SO necesita liberar suficiente memoria RAM para cargar un nuevo proceso.
- Otra razón del SO.
  - El SO puede un proceso que se sospecha causa un problema.
- Solicitud del usuario.
- Por tiempo.
  - Se ejecuta con cierta frecuencia, entonces mientras no se usa se suspende.
- Solicitud del proceso padre.
  - El padre desea suspenderlo para examinar o modificar el proceso o para coordinar con otros procesos.

Descripción de Proceso

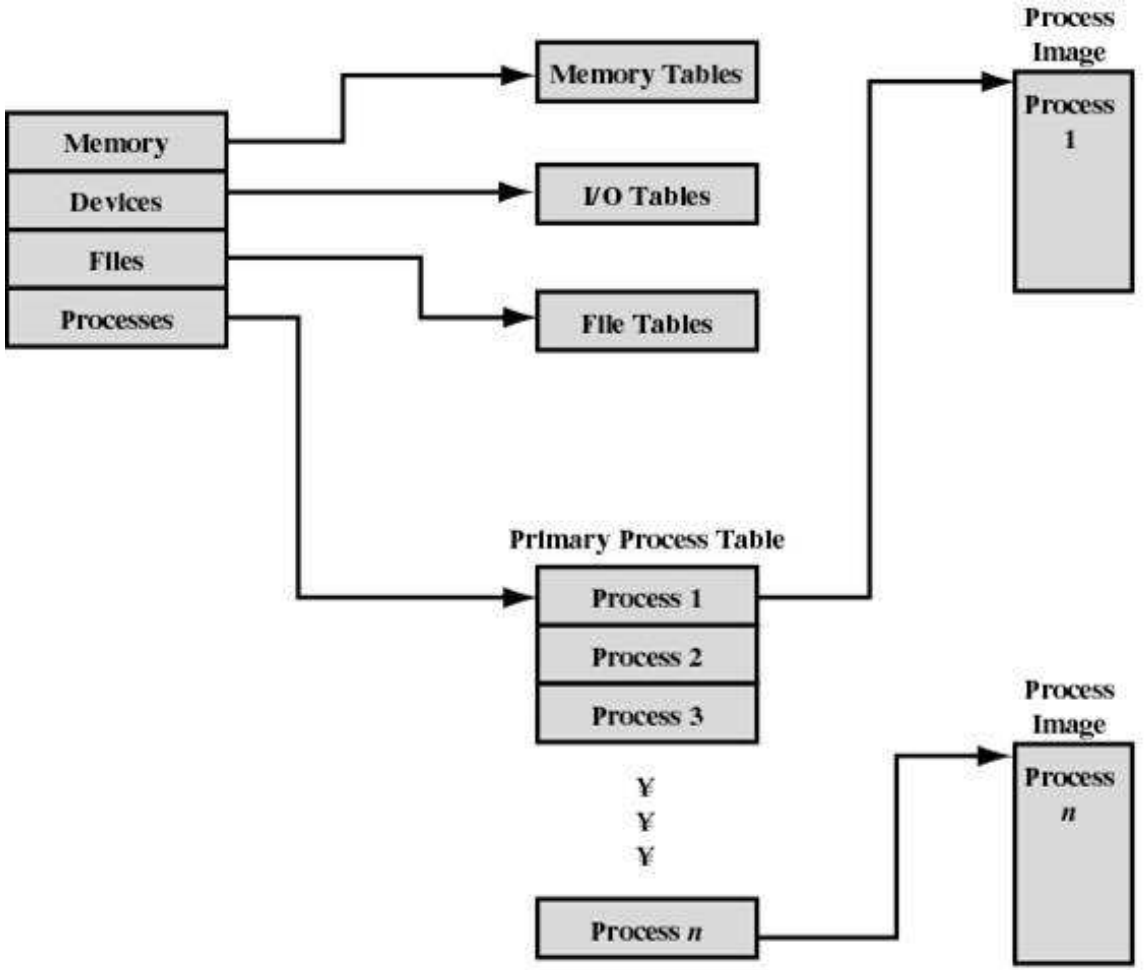
- En un entorno multiprogramado muchos procesos requieren y están haciendo uso de recursos.



- ¿Qué información necesita el SO para controlar los procesos y administrar los recursos?

Estructuras de Control del SO

- Para administrar todo lo que pasa en el sistema el SO construye y mantiene tablas de información de cada entidad que esté administrando.



Tablas de Memoria

- Se utiliza para administrar la memoria virtual y la memoria real.
  - Asignación de memoria principal a los procesos.
  - Asignación de memoria secundaria a los procesos.
  - Atributos de protección para acceso a regiones de memoria compartida.
  - Información necesaria para administrar la memoria virtual.

Tablas de E/S

- Se utiliza para administrar los dispositivos y canales DES:
  - Estado del DES: disponible o asignado.
  - Estado de una operación con el DES.
  - Ubicación en la memoria principal que ha sido usada como fuente o destino de una operación de E/S.

Tabla de Archivos

- Existencia de archivos.
  - Ubicación en la memoria secundaria.
  - Estado actual.
  - Atributos.
  - A veces esta información es mantenida por el sistema de administración de archivos (*file-management system*).

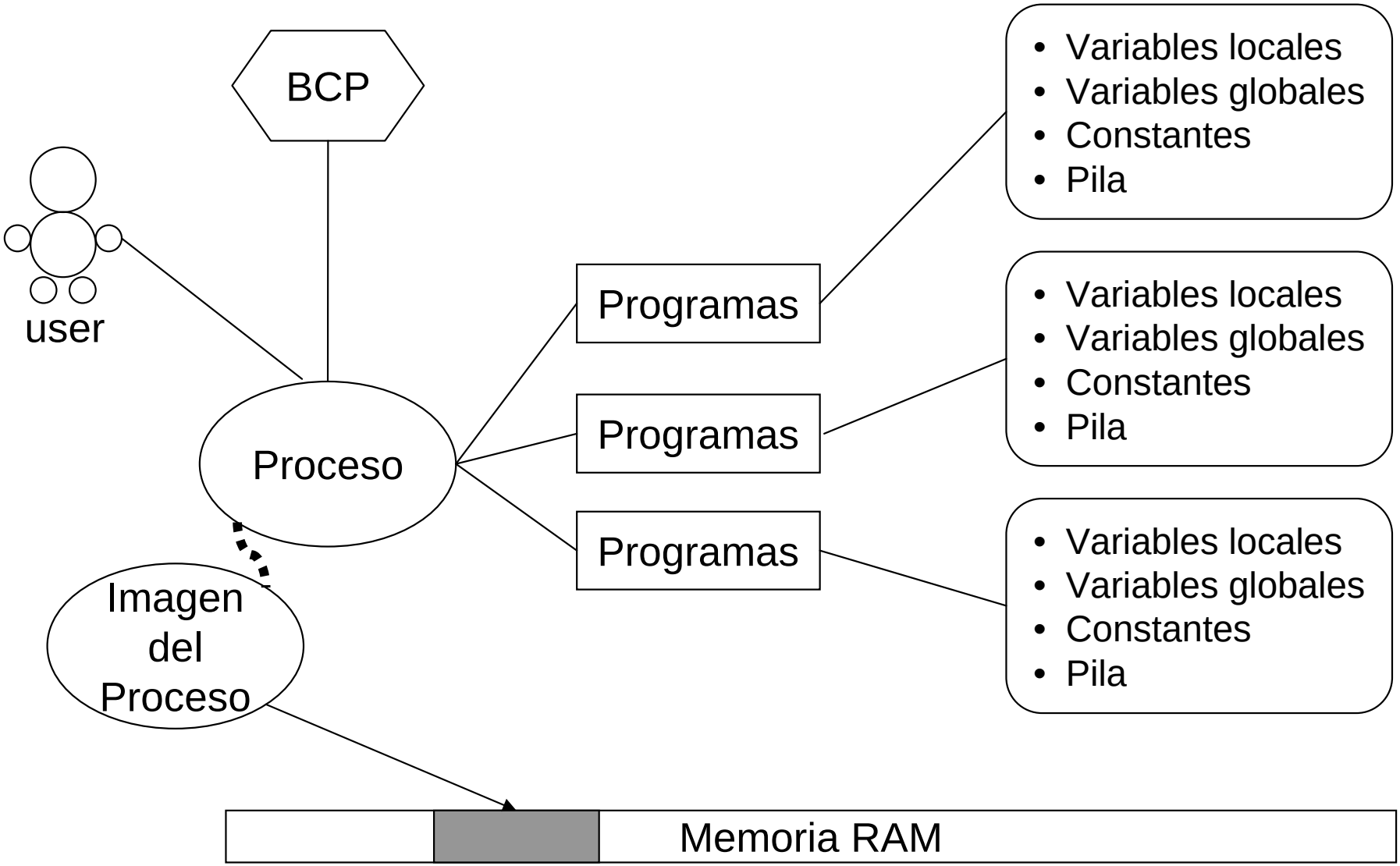
Tabla de Procesos

- Permite administrar la información de cada proceso
  - Donde está ubicado → - proceso en memoria
    - imagen del proceso
  - Atributos necesarios para este administrador.
    - Process ID
    - Process state
    - Location in memory

Estructuras de Control

- Para que el SO administre los proceso debe de conocer:
  - **Ubicación de proceso.**
  - **Atributos.**

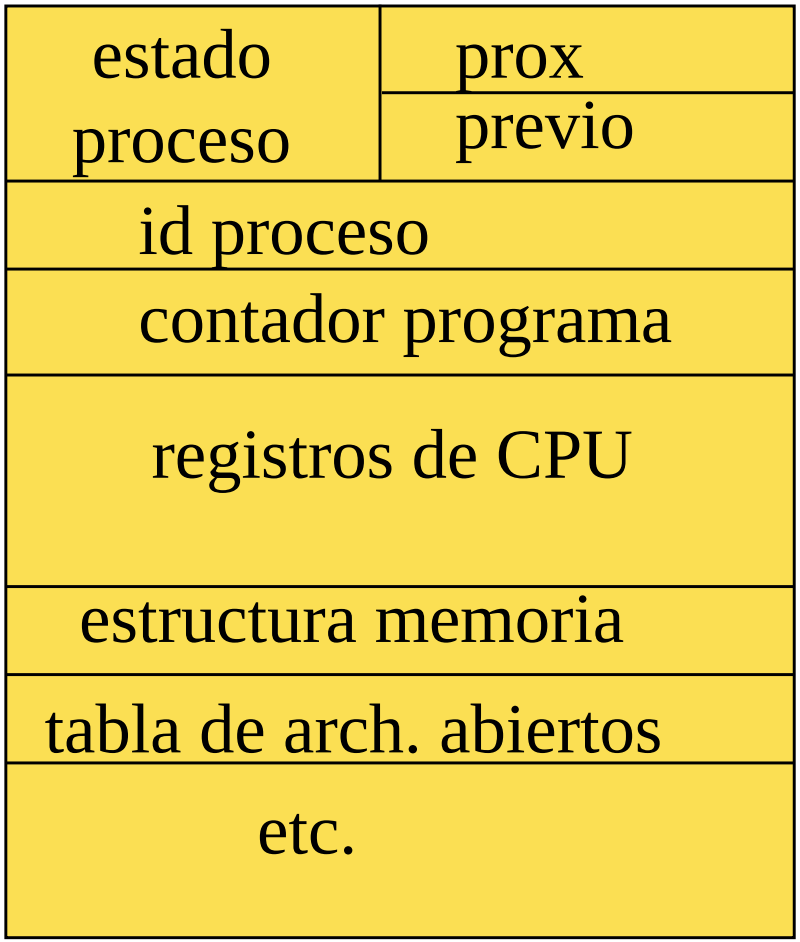
Ubicación de los Procesos



Atributos

- Bloque de Control de Proceso:
  - Identificación del proceso.
    - Identificadores
  - Información del estado del procesador.
    - Registros Visibles para el usuario.
    - Registro de control y de estado
    - Punteros de pila
  - Información de control del proceso.
    - Información de planificación y de estado
    - Estructuración de datos
    - Comunicación entre procesos
    - Privilegios de los procesos
    - Gestión de memoria
    - Propiedad de los recursos y utilización

Bloque de Control de Proceso (PCB)



Identificación del proceso

- Identificadores
  - Identificador del proceso.
  - Identificador del proceso que creó a este proceso (padre).
  - Identificador del usuario.

Información del estado del procesador

- Registros Visibles al usuario
  - Un registro visible para el usuario es aquel que puede hacerse referencia por medio de un lenguaje de máquina que ejecuta el procesador.
  - Normalmente, existen entre 8 a 32 de estos registros, aunque algunas implementaciones RISC tienen más de 100.
- Registros de Control y de Estado.

  Son registros del procesador para controlar su funcionamiento:
  - Contador de programa. Siguiente instrucción.
  - Códigos de condición. Resultado de la operación aritmética más reciente (signo, cero, acarreo, igual, desbordamiento)
  - Información de estado. Habilitación e inhabilitación de interrupciones y el modo de ejecución.
  - PSW. Palabra de estado de programa. Códigos de condición.

Palabra de estado de programa (PSW)

- La PSW almacena información pertinente sobre el programa que esté ejecutándose.
  - Códigos de condición.
  - Indicadores de habilitación de traps
  - Nivel de prioridad de interrupciones
  - Modo previo
  - Modo actual
  - Pila de interrupciones
  - Primera parte hecha (donde se quedó)
  - Traza pendiente (debug)

Información del estado del procesador

- Punteros de Pila.
  - Cada proceso tiene una o más pilas FIFO del sistema asociadas.
  - Las pilas se utilizan para almacenar los parámetros y las direcciones de retorno de los procedimientos y de las llamadas al sistema.
  - El puntero de pila siempre apunta al tope de la pila.

Información de control del procesador (1)

- Información de Planificación y de Estado
  - Esta es la información que se necesita por el SO para ejecutar sus funciones de planificador:
  - **Estado del proceso**. Define la disposición del proceso para ser planificado para ejecutar (en ejecución, listo, esperando, suspendido).
  - **Prioridad**. Se puede usar con uno o más campos para describir la prioridad de planificación de los procesos. (pueden ser omisión, actual, la más alta permitida).
  - **Información de planificación**. Depende del algoritmo de planificación utilizado (tiempo de espera, tiempo de ejecución).
  - **Eventos**. Identidad del evento que el proceso esta esperando antes de reanudarse.

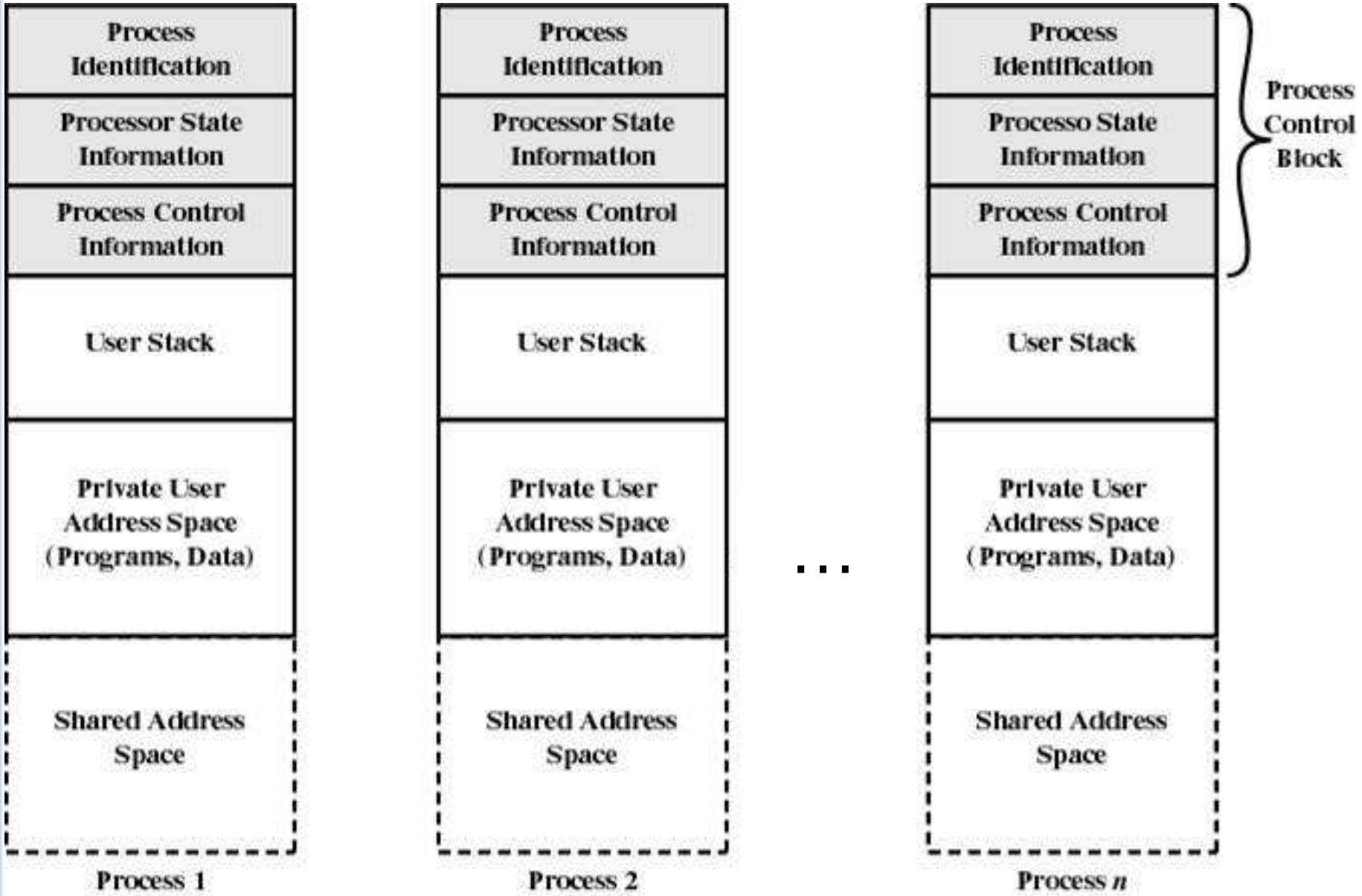
Información de control del procesador (2)

- Estructuras de datos
  - Un proceso puede estar enlazado con otros procesos en una cola, un anillo o alguna otra estructura.
  - Por ejemplo todos los procesos que están en estado de espera de un nivel determinado de prioridad pueden estar enlazados en una cola.
  - Un proceso puede mostrar una relación padre-hijo (creador-creado) con otro proceso.
  - El BCP puede contener punteros a otros procesos para dar soporte a estas estructuras.
- Comunicación entre procesos.
  - Puede haber varios indicadores, señales y mensajes asociados con la comunicación entre dos procesos independientes.
  - Una parte de esta información o toda ella se puede guardar en el BCP.

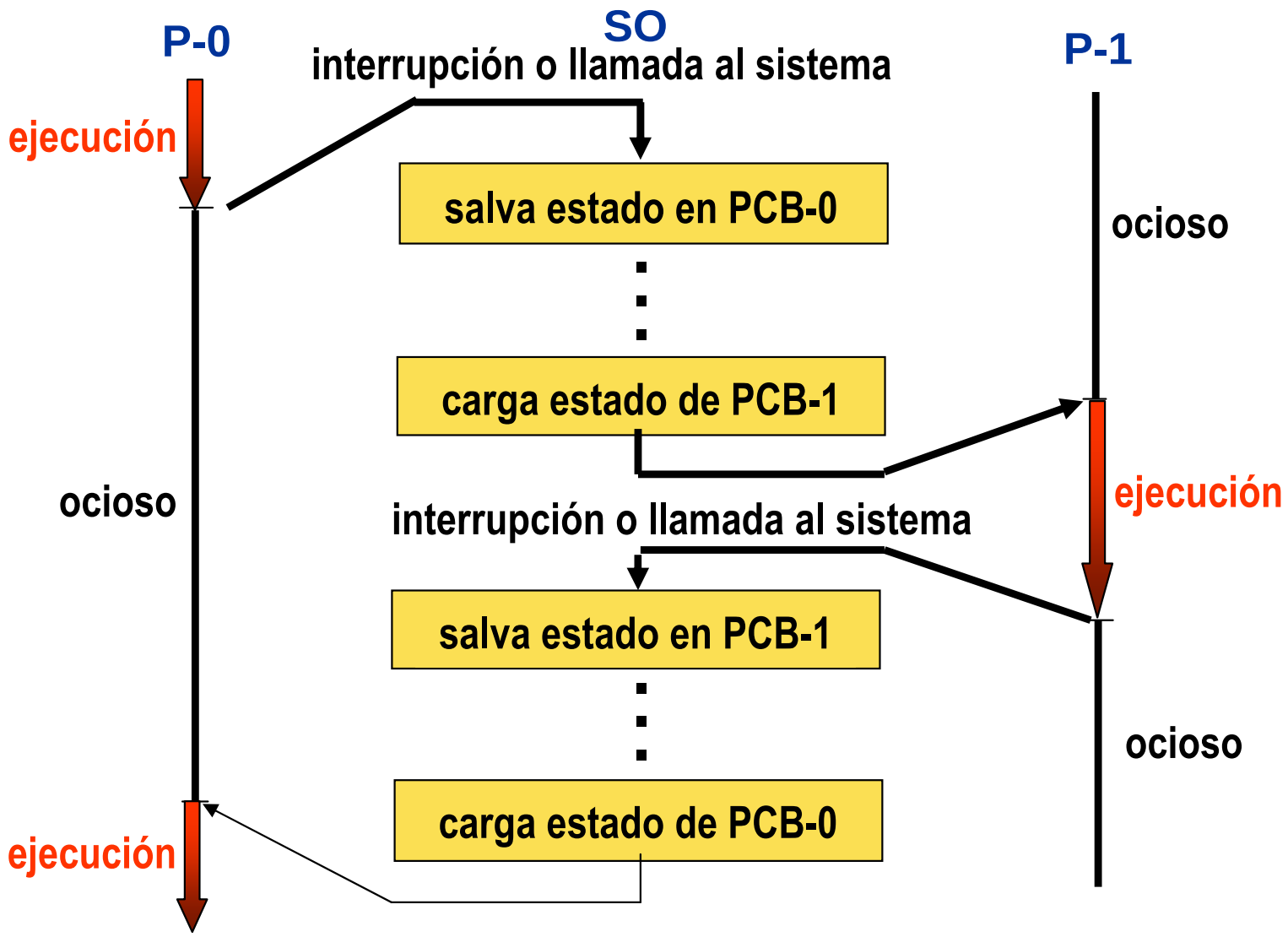
Información de control del procesador (3)

- Privilegios de los proceso.
  - A los procesos se les otorgan privilegios en términos de la memoria a la que pueden acceder y el tipo de instrucciones que pueden ejecutar. Además, también se pueden aplicar privilegios al uso de los servicios y utilidades del sistema.
- Gestión de memoria.
  - Esta sección puede incluir punteros a las tablas de páginas y/o segmentos que describen la memoria virtual asignada.
- Propiedad de los recursos y utilización.
  - Se pueden incluir los recurso controlados por el proceso, tales como los archivos abiertos.
  - Puede ser el histórico de la utilización del procesador o de otros recursos.
  - Información necesaria para el planificador.

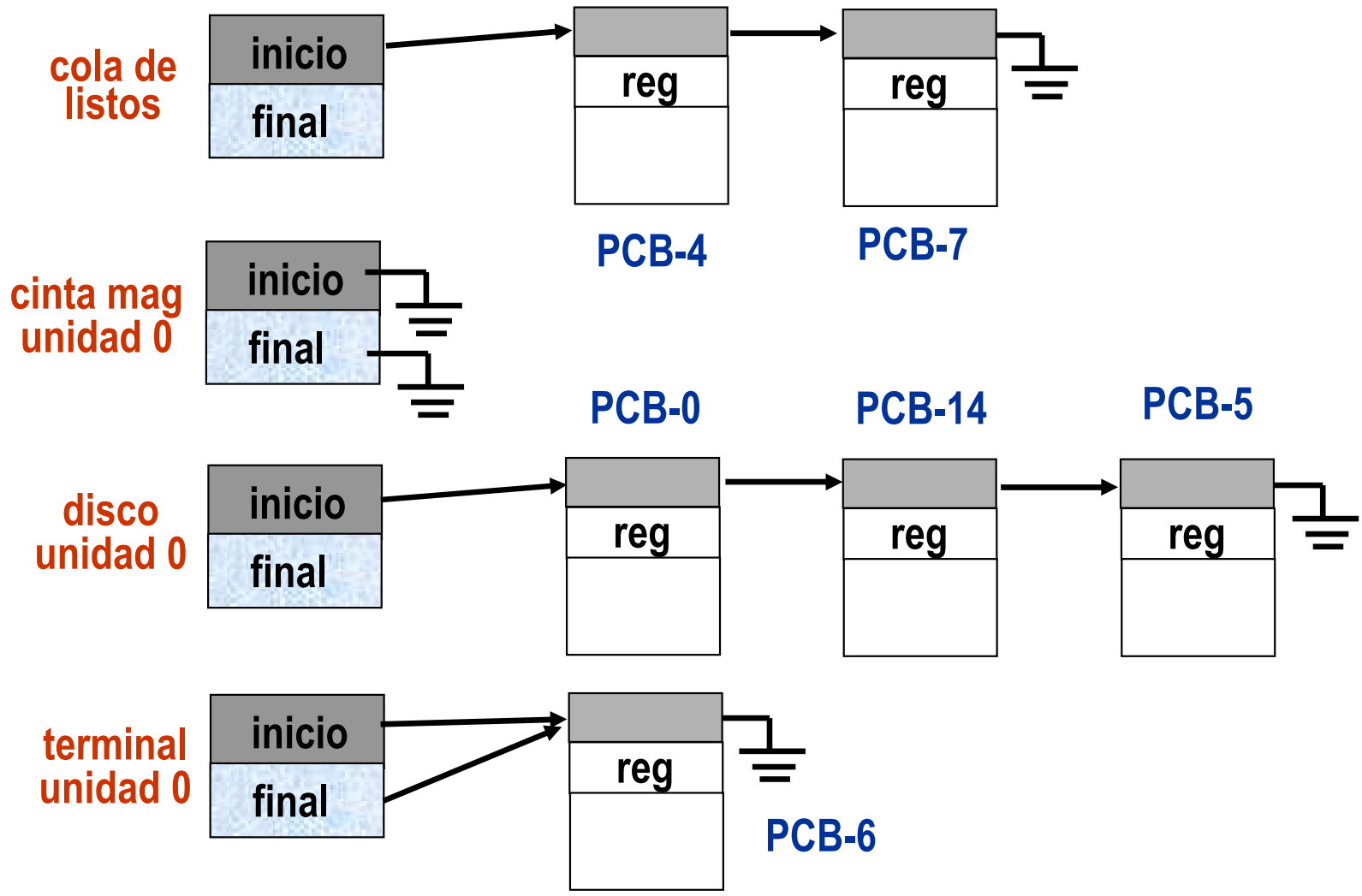
Imagen de un proceso en Memoria



Pasaje de CPU de un proceso a otro



Colas Listos y de Dispositivos I/O

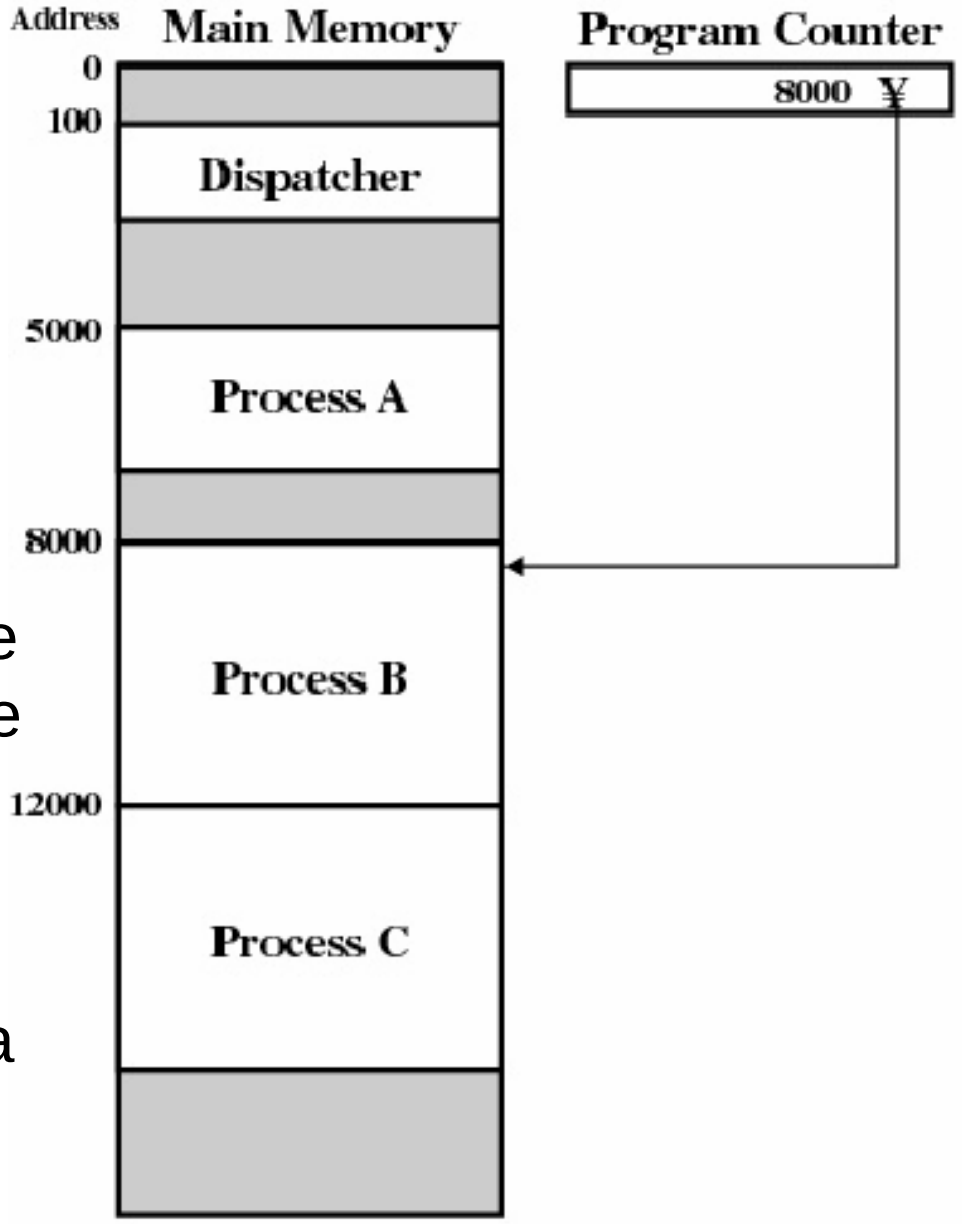


Administración de Procesos

- **Scheduler:** agrega (nuevos) y elimina (terminados) procesos a la tabla de procesos.
- **Dispatcher (planificador de corto término):** controla la asignación de *slices* de tiempo a los procesos referenciados en la tabla de procesos.
  - El final del slice se marca con una INT.
- Nótese que existen otras definiciones de scheduler / dispatcher.

Ejemplo de Procesos en mem

- No hay memoria virtual
- Tres programas cargados en memoria.
- Cada proceso representa a un programa.
- Existe un programa que asigna el procesador de un proceso a otro.
- Cada proceso tiene un tiempo de ejecución luego de lo cual ingresa el siguiente proceso



Traza de los tres procesos

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

```
1       5000
2       5001
3       5002
4       5003
5       5004
6       5005
--------------------Time out
7       100
8       101
9       102
10      103
11      104
12      105
13      8000
14      8001
15      8002
16      8003
-----------------I/O request
17      100
18      101
19      102
20      103
21      104
22      105
23      12000
24      12001
25      12002
26      12003
27      12004
28      12005
--------------------Time out
29      100
30      101
31      102
32      103
33      104
34      105
35      5006
36      5007
37      5008
38      5009
39      5010
40      5011
--------------------Time out
41      100
42      101
43      102
44      103
45      104
46      105
47      12006
48      12007
49      12008
50      12009
51      12010
52      12011
--------------------Time out
```

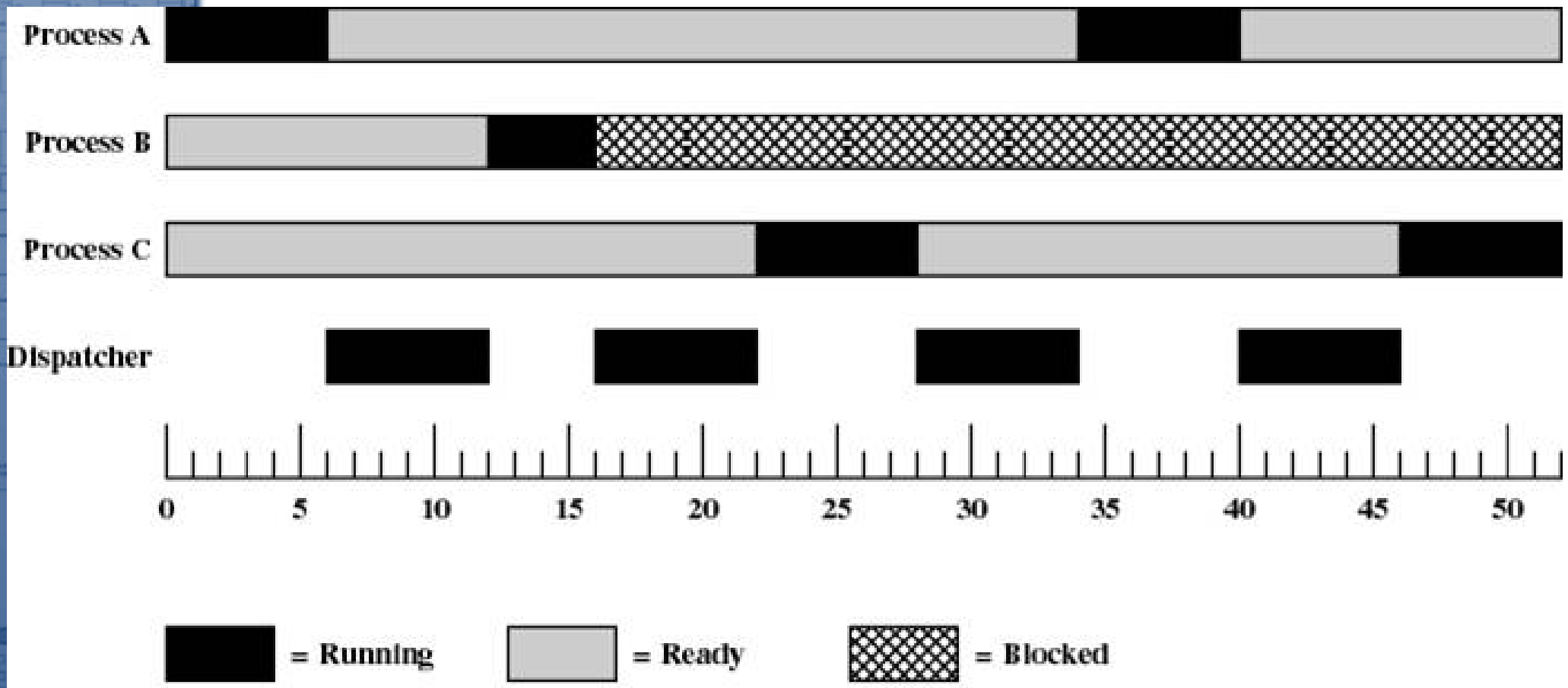
Traza de tres procesos

100 dirección de inicio para el programa distribuidor (dispatcher)

Las áreas sombreadas indican ejecución del proceso dispatcher.

La primera y tercera columna cuenta el ciclo de instrucción

La segunda y cuarta columna muestra la dirección de la que se está ejecutando.


Process A
Process B
Process C
Dispatcher
0
5
10
15
20
25
30
35
40
45
50
= Running
= Ready
= Blocked

Planificadores (1)

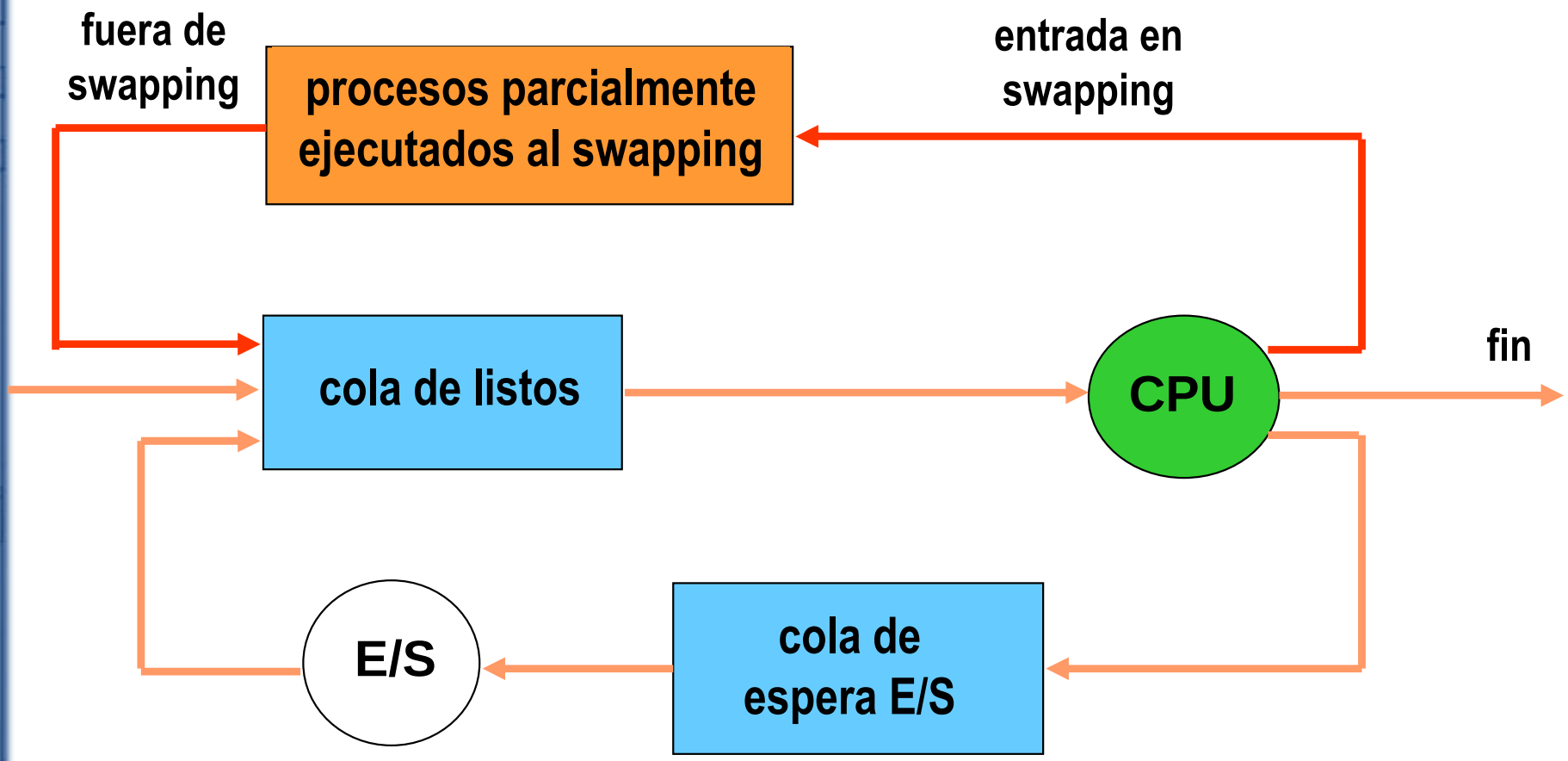
- Planificador de largo término

  (o planificador de jobs) – selecciona que procesos deberían ser puestos en la cola de listos.
- Planificador de corto término

  (o planificador de CPU / dispatcher) – selecciona que procesos deberían ser próximamente ejecutados y colocados en la CPU.

Planificadores (2)

Planificación de mediano término



Planificadores (3)

- El planificador de corto término es invocado muy frecuentemente (milisegundos) ⇒ (debe ser rápido).
- El planificador de largo término es invocado menos frecuentemente (segundos, minutos) ⇒ (puede ser muy lento).
- El planificador de largo término controla el *grado de multiprogramación*.
- Los procesos pueden ser descriptos como:
  - ***Procesos limitados por E/S*** – pasa más tiempo haciendo E/S que computaciones, ráfagas (burst) de CPU muy cortas.
  - ***Procesos limitados por CPU*** – pasa más tiempo haciendo computaciones que E/S, ráfagas (burst) de CPU muy largas.

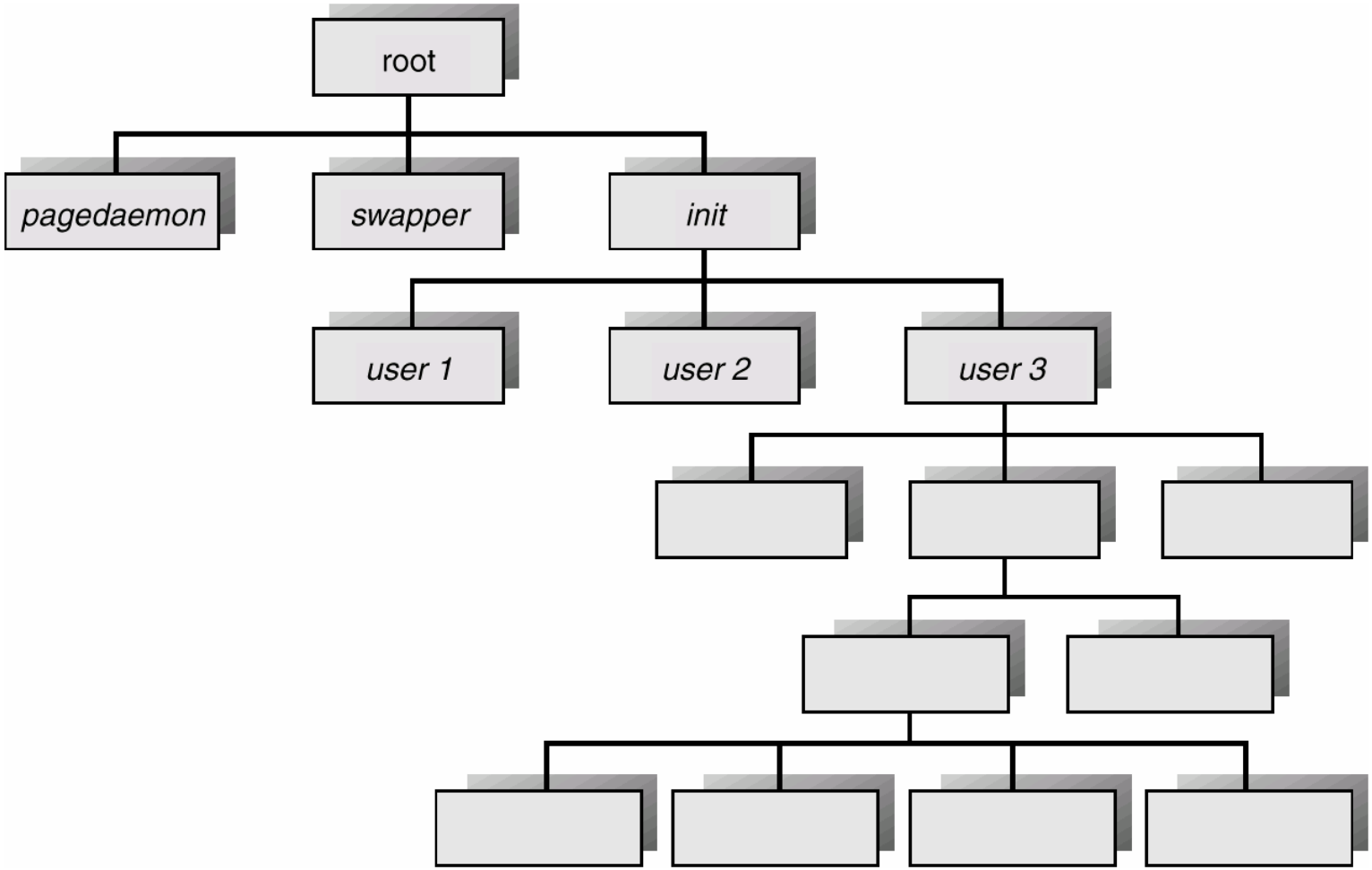
Cambio de contexto

- Cuando la CPU conmuta a otro proceso (*context switch*), el sistema debe salvar el estado del viejo proceso y cargar el estado para el nuevo proceso.
- El tiempo que lleva el cambio de contexto es *overhead*: el sistema no hace trabajo útil mientras está conmutando.
- El tiempo depende (entre otros) del soporte de hardware.

Creación de Procesos

- Procesos padres crean procesos hijos, los cuales, a su vez crean otros procesos, formando un árbol de procesos.
- Recursos compartidos
  - Padres e hijos comparten todos los recursos.
  - Hijo comparte un subconjunto de los recursos del padre.
  - Padre e hijo no comparten ningún recurso.
- Ejecución
  - Padres e hijos ejecutan concurrentemente.
  - Padres esperan hasta que los hijos terminan.

Árbol de Procesos en UNIX



Terminación de Procesos

- El proceso ejecuta la última sentencia y espera que el SO haga algo (**exit**).
  - Los datos de salida del hijo se pasan al padre (via **wait**).
  - Los recursos de los procesos son liberados por el SO.
- El padre puede terminar la ejecución del proceso hijo (**abort**).
  - El hijo ha excedido los recursos asignados.
  - La tarea asignada al hijo no es más requerida.
  - El padre está terminando.
    - El SO no permite a los hijos continuar si su padre termina.
    - Terminación en cascada.

Procesos Cooperativos

- Un proceso ***independiente*** no puede afectar ni 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
  - Información compartida
  - Aceleración de la computación
  - Modularidad
  - Conveniencia

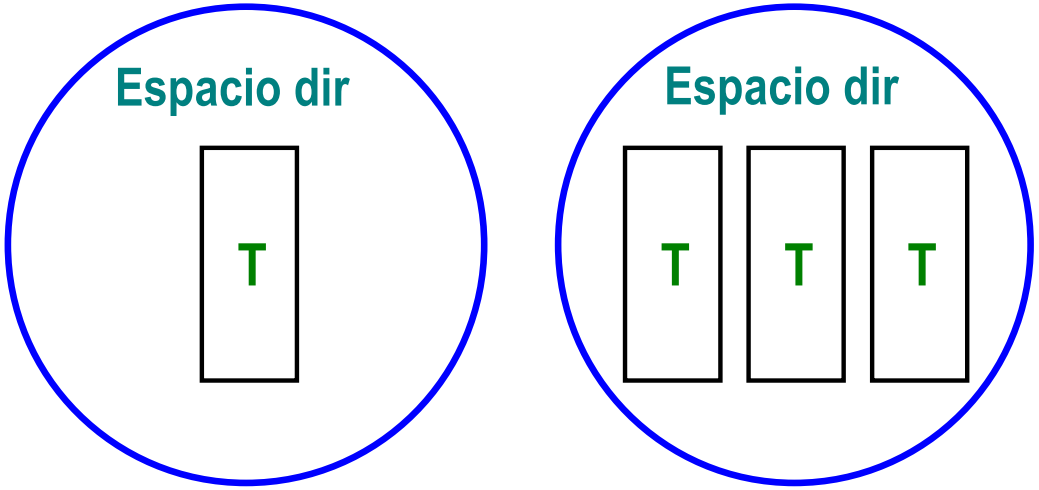
Problema del Productor-Consumidor

- Paradigma procesos cooperativos, el proceso ***productor*** produce información que es consumida por un proceso ***consumidor***.
  - *buffer ilimitado:* no tiene límites prácticos en el tamaño del buffer.
  - *buffer limitado:* supone que hay un tamaño fijo de buffer.

Hilos de Control

THREADS

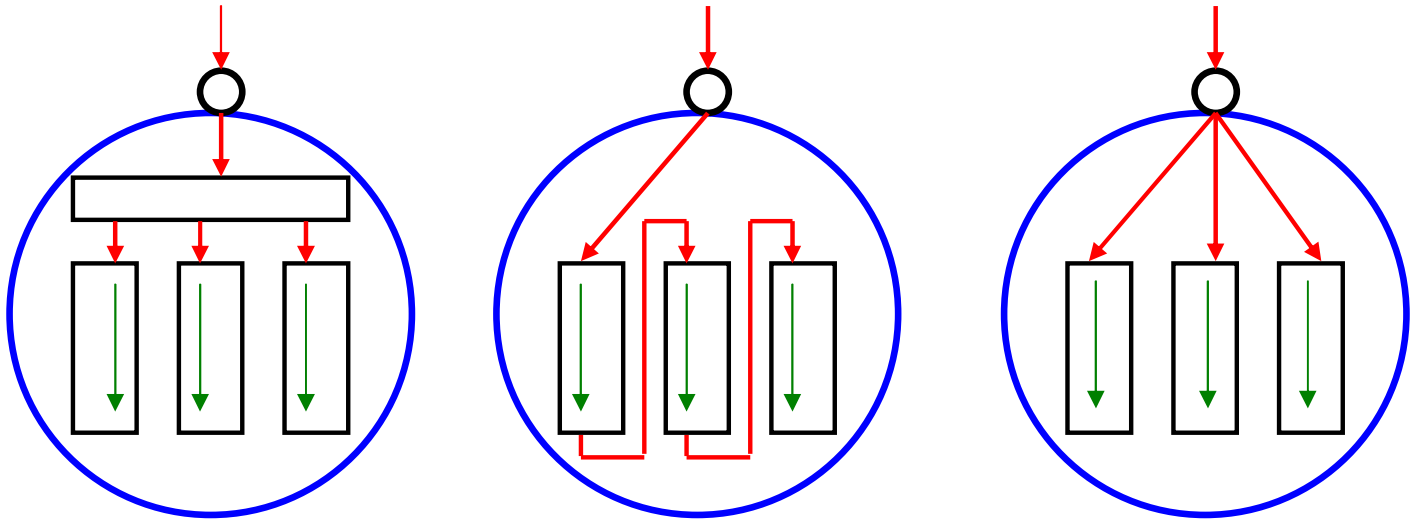
Uso de ***threads*** (*lightweight processes*)



- Igual espacio de direcciones
- No hay protección entre los mismos y no es necesaria
- Estado, stack

Modelos de Organización de Threads

- Modelo Despachador-Trabajador
- Modelo "Pipeline"
- Modelo "Team"



Procesos

Ejemplo: Servidor de Archivos

- Un *thread* simple
- *Threads* múltiples
- Máquina de estados finitos

Modelo	Características
Threads	Paralelismo, llamadas al sistema bloqueantes
Proceso simple	No paralelismo, llamadas al sistema bloqueantes
Máquinas de Estados Finitos	Paralelismo, llamadas al sistema no bloqueantes

Motivaciones para su uso

1. La sobrecarga de crear un nuevo proceso es considerable frente a la creación de un nuevo *thread* dentro de un proceso.
2. La conmutación entre *threads* compartiendo el mismo espacio de direcciones es más barata que entre procesos.
3. Los *threads* permiten paralelismo al ser combinados con ejecución secuencial y llamadas a sistemas bloqueantes.
4. Compartir recursos puede hacerse más eficiente y natural entre *threads* de un proceso que entre procesos mismos a causa de compartir el mismo espacio de direcciones.

Aspectos de Diseño de Threads

Creación de threads

- estática
- dinámica

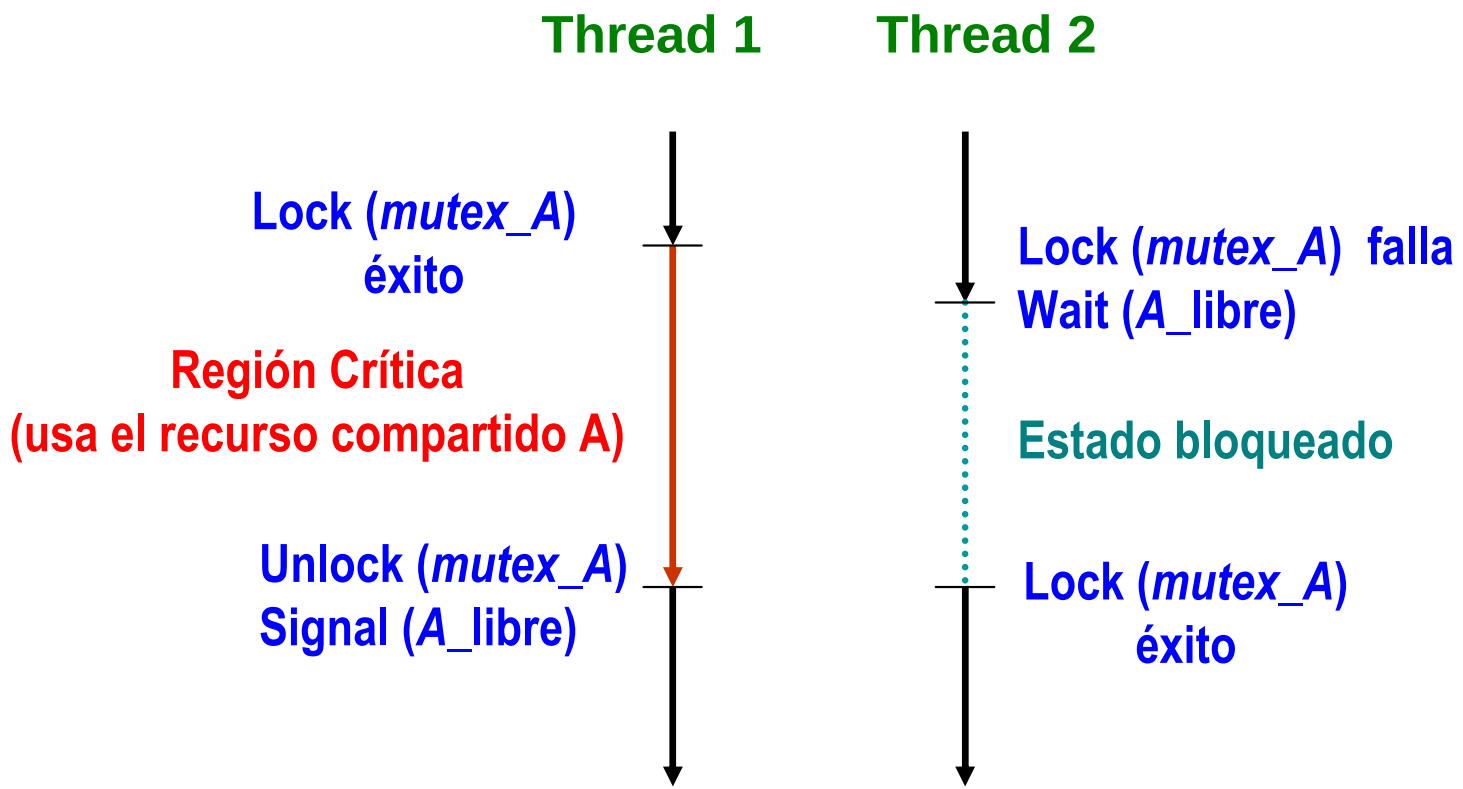
Terminación de threads

- pueden terminar convencionalmente
- pueden autodestruirse o ser destruidos

Sincronización de Threads

- ✓ usa variables *mutex*
- ✓ si encuentra *mutex=lock*
  - a) el *thread* se bloquea y espera en una cola por la variable *mutex*.
  - b) un *cod-status* indica la situación. El *thread* puede continuar con otra tarea o esperar.

Threads

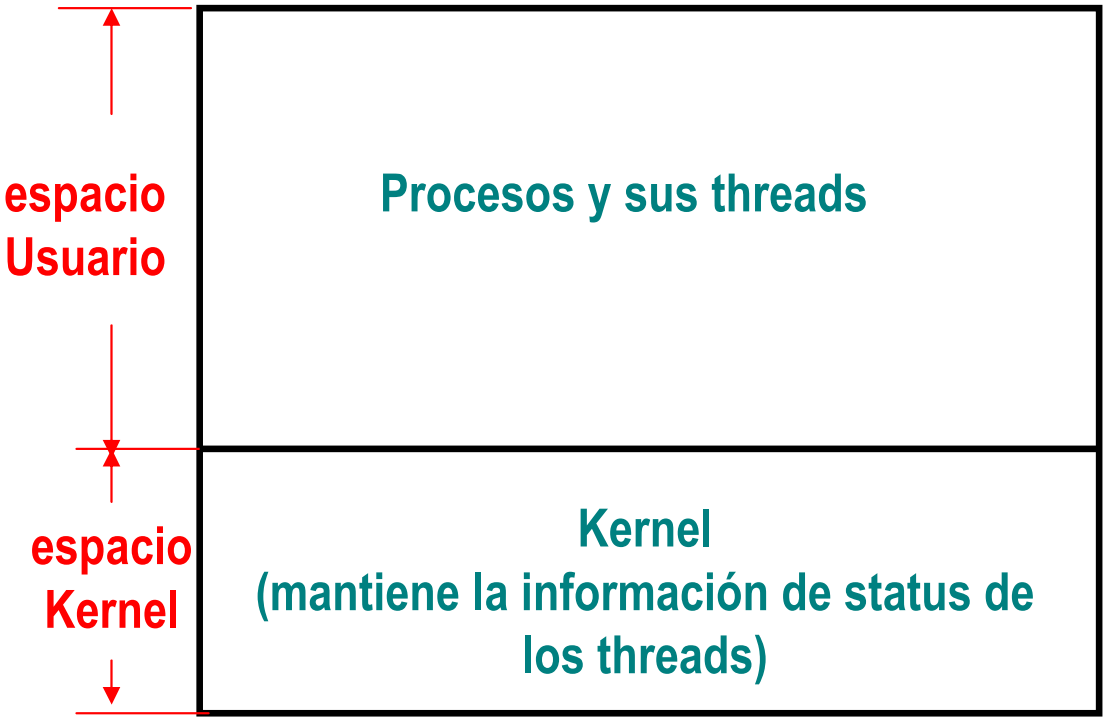

Thread 1
Thread 2
Lock (*mutex_A*)
éxito
Región Crítica
(usa el recurso compartido A)
Unlock (*mutex_A*)
Signal (*A*_libre)
Lock (*mutex_A*) falla
Wait (*A*_libre)
Estado bloqueado
Lock (*mutex_A*)
éxito

Planificación de Threads

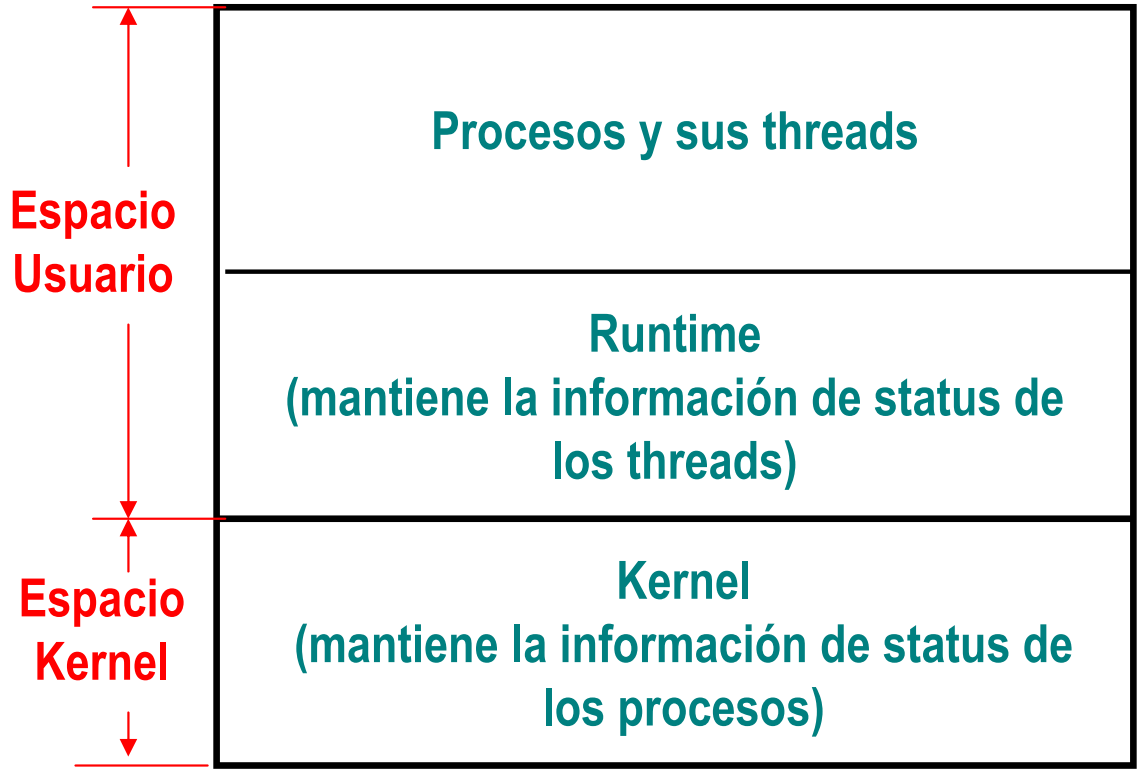
- ✓ facilidad para asignar prioridad.
- ✓ flexibilidad para cambiar el *quantum*.
- ✓ planificación forzada.
- ✓ planificación afín.

Implementación

Espacio del kernel



Espacio de Usuario



Ventajas y desventajas de las distintas alternativas (1)

- ✓ **Ventaja del nivel-usuario:** puede construirse encima del sistema operativo que no soporta *threads*. En el otro caso deben meterse en el *kernel*.
- ✓ **Ventaja del nivel-usuario:** puede usar su propio esquema de planificación. No es posible en el nivel-kernel.

Ventajas y desventajas de las distintas alternativas (2)

- ✓ Ventaja del nivel-usuario: el cambio de contexto de un *thread* en el nivel-usuario es más rápido que en el nivel-kernel. Es hecho por el *runtime*, en el nivel-kernel requiere un *trap*.
- ✓ Ventaja del nivel-usuario: la escalabilidad es mayor. Las tablas a nivel-kernel son mantenidas dentro del *kernel*.
- ✓ Desventaja del nivel-usuario: trabajan en multiprogramación pura.
- ✓ Desventaja del nivel-usuario: las *system-calls* generan problemas de bloqueo.

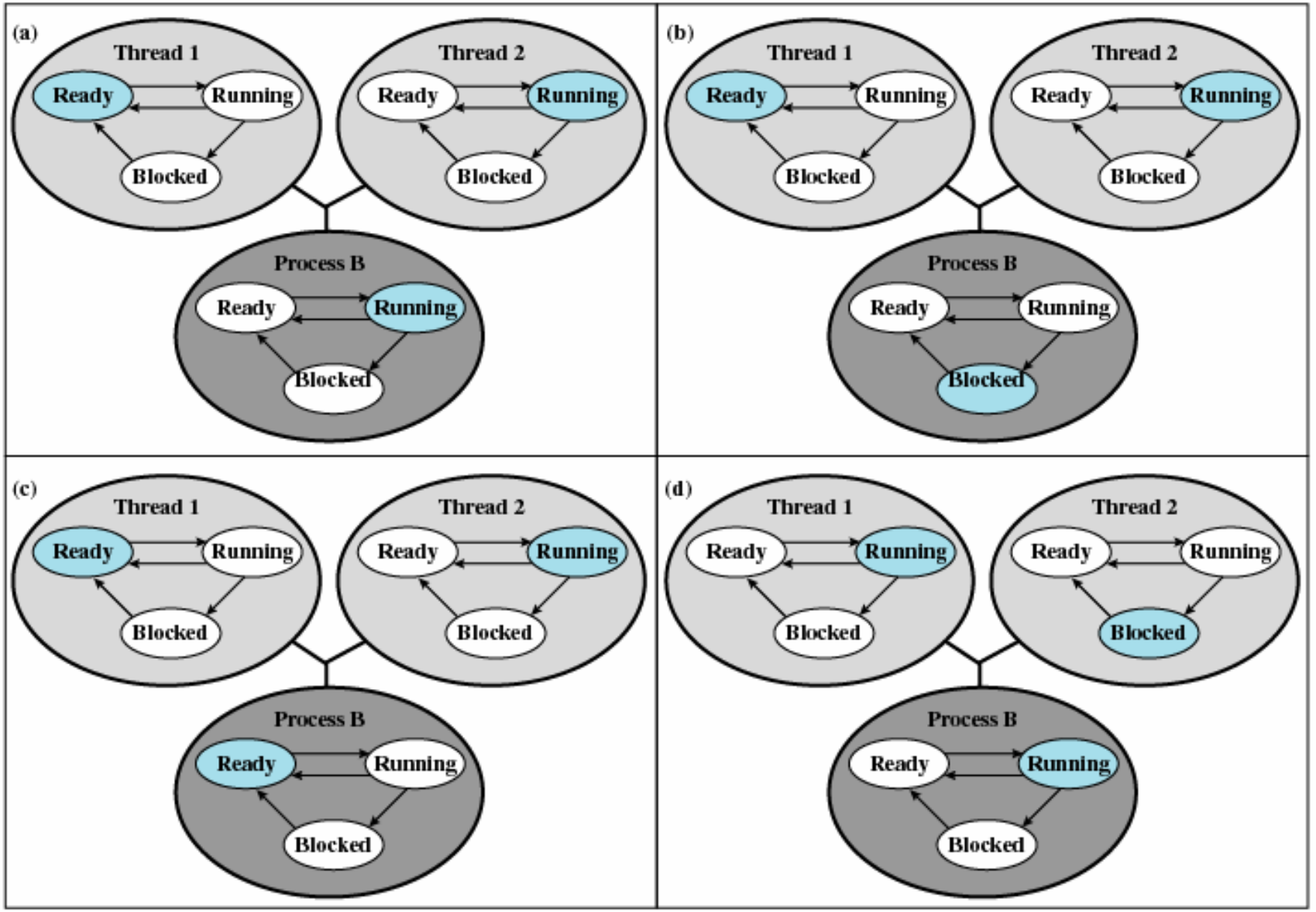
Beneficios del Uso de Threads

- Capacidad de Respuesta
- Compartir Recursos
  - Dado que los *threads* dentro de un mismo proceso comparten memoria y archivos, pueden comunicarse unos con otros sin invocar al kernel
- Economía
  - Toma menos tiempo crear un nuevo *thread* que un proceso
  - Menos tiempo terminar un *thread* que un proceso
  - Menos tiempo en conmutar entre dos *threads* dentro del mismo proceso
- Utilización de Arquitecturas Multiprocesador

Threads a Nivel de Usuario

- Manejo de threads hecho por librerías a nivel de usuario.
- Ejemplos
  - POSIX *Pthreads*
  + Native POSIX Thread Lib (NPTL)
  - Mach *C-threads*
  - Solaris *threads*

Relaciones entre estados de threads nivel usuario y estados de procesos



1) La aplicación corriendo en el thread 2 hace una llamada a sistema que bloquea B (por ejemplo: E/S). Esto transfiere el control al kernel que pone al proceso B en estado bloqueado (espera) pero de acuerdo a la estructura de datos mantenida por la librería de threads, el thread 2 permanece en "corriendo". (b)

2) Cuando el proceso B ha agotado su tiempo pasa al estado de listo, pero de acuerdo a la estructura de datos mantenida por la librería de threads, el thread se mantiene en estado "corriendo". (c)

3) El thread 2 ha alcanzado un punto donde necesita alguna acción por parte del thread 1. El thread 2 entra en estado de bloqueado y el thread 1 pasa a "corriendo". El proceso B que los contiene se mantiene en estado "corriendo". (d)

Threads a Nivel Kernel

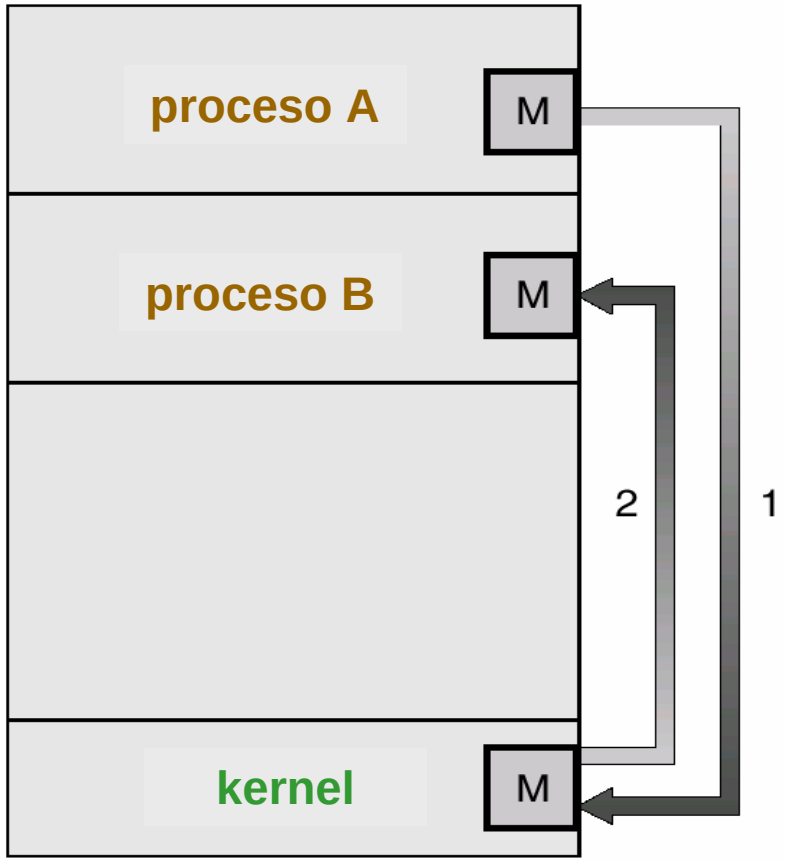
- Soportados por el Kernel

- Ejemplos
  - Windows 95/98/NT/2000/XP
  - Solaris
  - Tru64 UNIX

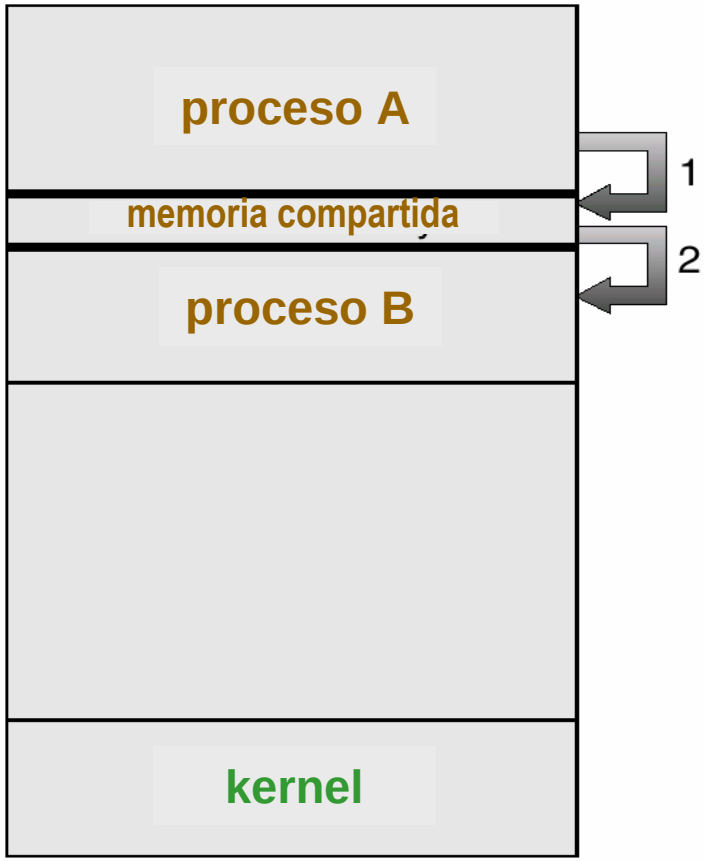
Comunicación entre Procesos

Modelos de Comunicación

Pasaje de mensajes



Memoria compartida



Comunicación entre procesos (IPC)

- Mecanismo de los procesos para comunicarse y sincronizar sus acciones.
- Sistema de mensajes – los procesos se comunican uno con otro sin necesidad de variables compartidas.
- Las facilidades de IPC provee dos operaciones:
  - **send** (*mensaje*) – mensaje de tamaño fijo o variable
  - **receive** (*mensaje*)
- Si *P* and *Q* desean comunicarse, necesitan:
  - Establecer un *vínculo de comunicación* entre ellos
  - Intercambiar mensajes via send/receive
- Implementación de un vínculo de comunicación
  - lógico (p.e., propiedades lógicas)
  - físico (p.e., memoria compartida, canal hardware)

Comunicación Directa

- Los procesos deben nombrar al otro explícitamente:
  - **send** (*P, mensaje*) – envía un mensaje al proceso P.
  - **receive** (*Q, mensaje*) – recibe un mensaje del proceso Q.
- Propiedades del vínculo de comunicación
  - Los vínculos son establecidos automáticamente.
  - Un vínculo está asociado con exactamente un par de procesos que se comunican.
  - Entre cada par existe exactamente un vínculo.
  - El vínculo puede ser unidireccional, pero es usualmente bidireccional.

Comunicación Indirecta

- Los mensajes son dirigidos y recibidos desde *mailboxes* (también conocidos como *ports*).
  - Cada mailbox tiene una única identificación.
  - Los procesos pueden comunicarse sólo si comparten un mailbox.
- Propiedades de un vínculo de comunicación
  - El vínculo se establece solo si los procesos comparten un mailbox común.
  - Un vínculo puede ser asociado con muchos procesos.
  - Cada par de procesos puede compartir varios vínculos de comunicación.
  - Los vínculos pueden ser unidireccionales o bidireccionales.

Sincronización

- El pasaje de mensajes puede ser bloqueante o no bloqueante.
- **Bloqueante** es considerado **sincrónico**
- **No bloqueante** es considerado **asincrónico**
- La primitivas ***send*** y ***receive*** pueden ser bloqueantes o no bloqueantes.

Buffering

- La cola de mensajes asociada al vínculo se puede implementar de tres maneras.
  1. **Capacidad 0 mensajes**
     El emisor debe esperar al receptor (***rendez-vous***).
  2. **Capacidad limitada** – longitud finita de *n* mensajes.
     El emisor debe esperar si el vínculo está lleno.
  3. **Capacidad ilimitada** – longitud infinita
     El emisor nunca espera.

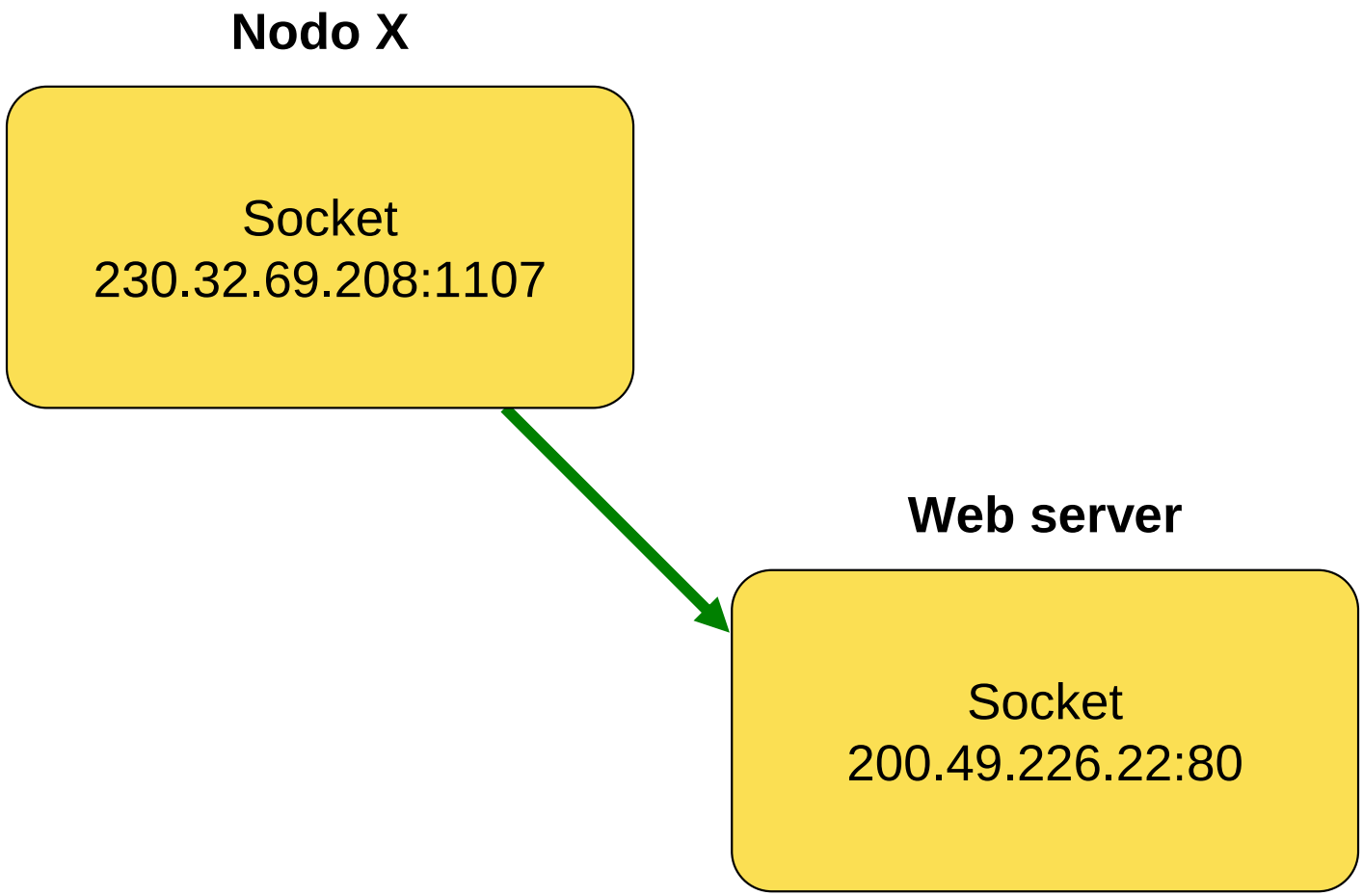
Comunicación Cliente-Servidor

- Sockets
- Llamadas a Procedimientos Remotors (**RPC**:Remote Procedure Calls)
- Invocación a Métodos Remotos (**RMI**:Remote Method Invocation (Java))

Sockets

- Un ***socket*** es definido como un *punto final de comunicación*.
- Concatenación de dirección IP y pórtico
- El socket 200.49.226.22:80 se refiere al port 80 en el host 200.49.226.22.
- La comunicación se lleva a cabo entre un par de sockets.

Comunicación por Sockets

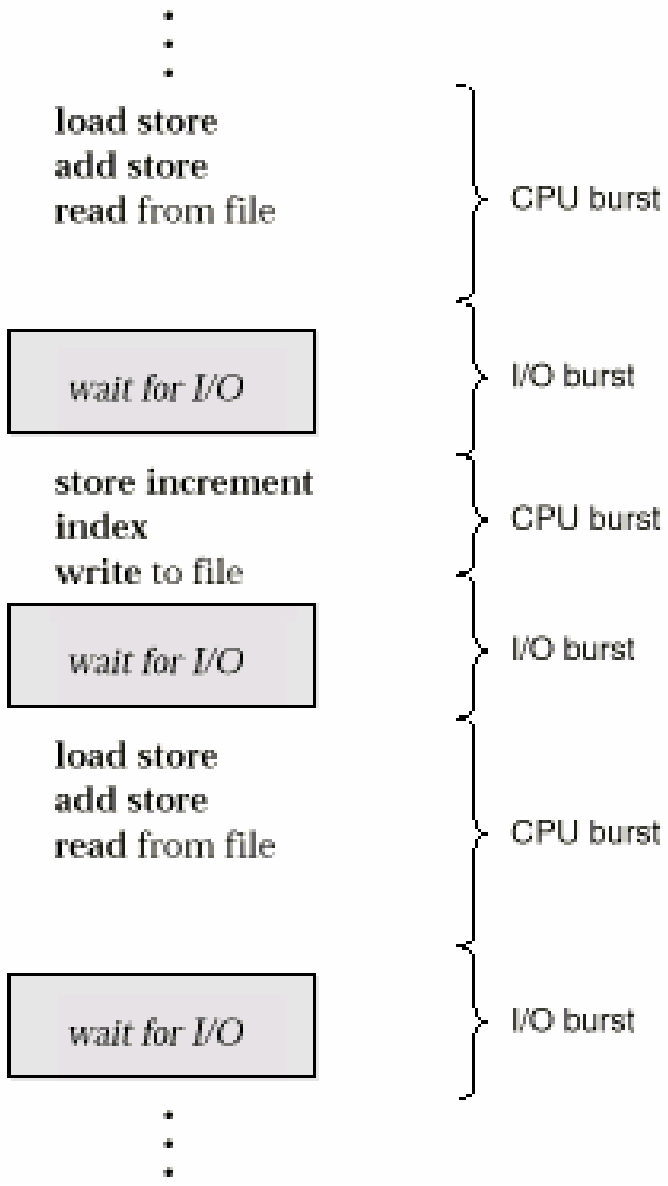


Criterios y Algoritmos de Planificación

Criterios Básicos

- Máxima utilización de CPU obtenida con multiprogramación.
- **Ciclo CPU–ráfagas de E/S** – La ejecución de procesos consiste de *ciclos* de ejecución de CPU y esperas en E/S.
- Distribución de ráfagas de CPU.

Secuencia Alternada de Ráfagas de CPU y E/S



Planificador de CPU

- Selecciona entre los procesos en memoria que están listos para ejecutar, y asigna la CPU a uno de ellos.
- La decisión de planificar la CPU puede tener lugar cuando un proceso:
  1. Conmuta de ejecutando a estado de espera.
  2. Conmuta de ejecutando a estado de listo.
  3. Conmuta de espera a listo.
  4. Termina.
- La planificación de 1 y 4 es *no apropiativa*.
- Las otras planificaciones son *apropiativas.*

Despachador

- El módulo despachador pasa el control de la CPU al proceso seleccionado (por el planificador de corto término); esto implica:
  - cambio de contexto
  - conmutación a modo usuario
  - salta a la dirección apropiada en el programa de usuario para reiniciarlo
- *Latencia de despacho* – tiempo que toma al despachador para detener un proceso e iniciar otro.

Criterios de Planificación

- Utilización de CPU – mantener la CPU tan ocupada como sea posible.
- Procesamiento total (Throughput) – número de procesos que completan sus ejecución por unidad de tiempo.
- Tiempo de retorno – cantidad de tiempo para ejecutar un determinado proceso.
- Tiempo de Espera – cantidad de tiempo que un proceso ha estado esperando en las colas.
- Tiempo de respuesta – cantidad de tiempo que transcurre desde que fue hecho un requerimiento hasta que se produce la primer respuesta, **no salida!**

Criterios de Optimización

- Maximizar la utilización de CPU
- Maximizar el procesamiento total
- Minimizar el tiempo de retorno
- Minimizar el tiempo de espera
- Minimizar el tiempo de respuesta

Planificación Primero-Entrar, Primero-Servido (FCFS)

Ejemplo:

Proceso	Tiempo de Ráfaga
P_1	24
P_2	3
P_3	3

Suponer que los procesos llegan en el orden: P_1 , P_2 , P_3

El diagrama de Gantt para la planificación es:

Tiempo de espera para P_1 = 0; P_2 = 24; P_3 = 27

Tiempo medio de espera: (0 + 24 + 27) / 3 = 17

Planificación FCFS (Cont.)

Suponer que los procesos llegan en el orden

$$P_2 , P_3 , P_1 .$$

- El diag. de Gantt para la planificación es:

- Tiempo de espera para $P_1 = 6; P_2 = 0, P_3 = 3$
- Tiempo medio de espera: (6 + 0 + 3) / 3 = 3
- Mucho mejor que el caso anterior.
- *Efecto Convoy* los procesos cortos delante de los procesos largos.

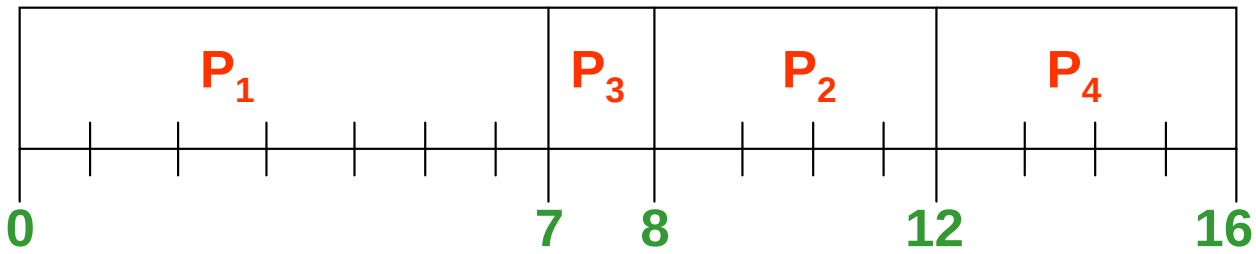
Planificación Job-Más Corto Primero (SJF)

- Se asocia con cada proceso la longitud de su *próxima* ráfaga de CPU. Se usan estas longitudes para planificar los procesos con el tiempo más corto.
- Dos esquemas:
  - No apropiativo – una vez que la CPU es dada a un proceso, no puede ser apropiada hasta que el mismo complete su ráfaga de CPU.
  - Apropiativo – si un nuevo proceso llega con una longitud de ráfaga de CPU menor que el resto del tiempo de ejecución que le queda al proceso que está ejecutando entonces se apropia de la CPU. Este esquema es conocido como El Tiempo Remanente Más Corto Primero (SRTF).
- **SJF es óptimo** – da el mínimo tiempo de espera promedio para un dado conjunto de procesos.

Ejemplo de SJF No Apropiativo

Proceso	Tiempo de llegada	Ráfaga
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (no apropiativo)

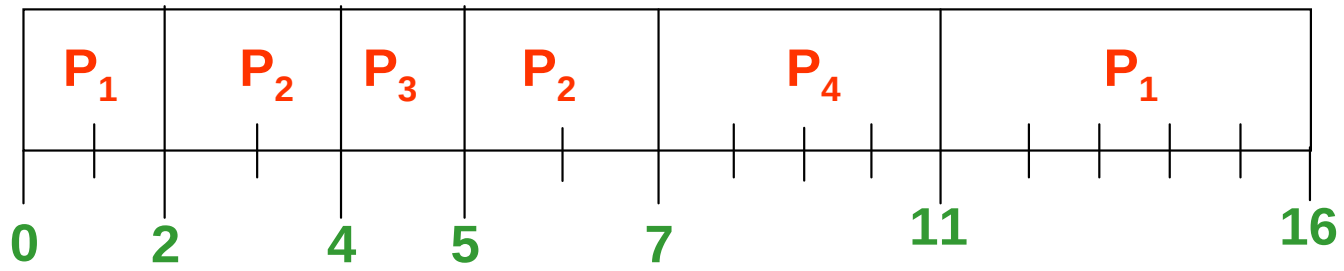


- Tiempo medio de espera = (0 + 6 + 3 + 7)/4 = 4

Ejemplo SJF Apropiativo

Proceso	Tiempo de llegada	Ráfaga
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (apropiativo)



- Tiempo medio de espera = (9 + 1 + 0 +2)/4 = 3

Planificación por Prioridad

- Con cada proceso se asocia un número (entero).
- La CPU es asignada al proceso con prioridad más alta (entero más pequeño ⇒ más alta prioridad o el entero más grande, *depende de la convención*).
  - Apropiativo
  - No apropiativo
- SJF es un algoritmo planificador con prioridad.
- **Problema** ⇒ **Inanición** – los procesos de baja prioridad pueden no llegar a ejecutarse nunca.
- **Solución** ≡ **Envejecimiento** – se incrementa en el tiempo la prioridad de los procesos en espera.

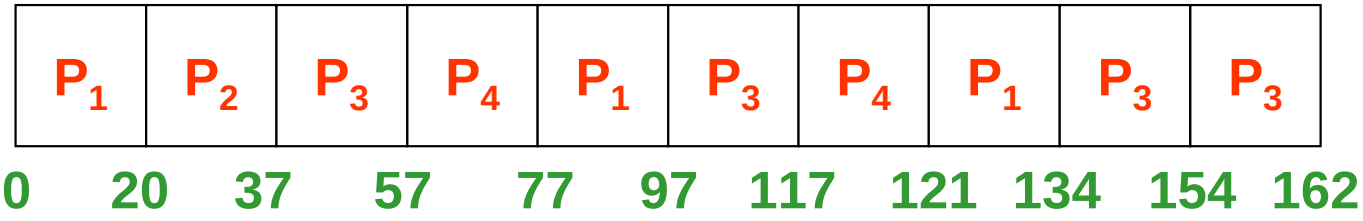
Round Robin (RR)

- Cada proceso toma una pequeña unidad de tiempo de CPU (***quantum***), usualmente 10-100 milisegundos. Luego de este tiempo el proceso es quitado de la CPU y agregado a la cola de listos.
- Si hay *n* procesos en la cola de listos y el tiempo del quantum es *q*, entonces cada proceso toma 1/*n* del tiempo de CPU en *slices* (rebanadas) de a lo sumo *q* unidades de tiempo a la vez. Los procesos no esperan más de (*n*-1)*q* unidades de tiempo.
- Rendimiento
  - *q* largo ⇒ Primero-Entrar, Primero-Salir
  - *q chico* ⇒ *q* debe ser grande con respecto al cambio de contexto, sino la sobrecarga es demasiado grande.

Ejemplo: RR con Quantum = 20

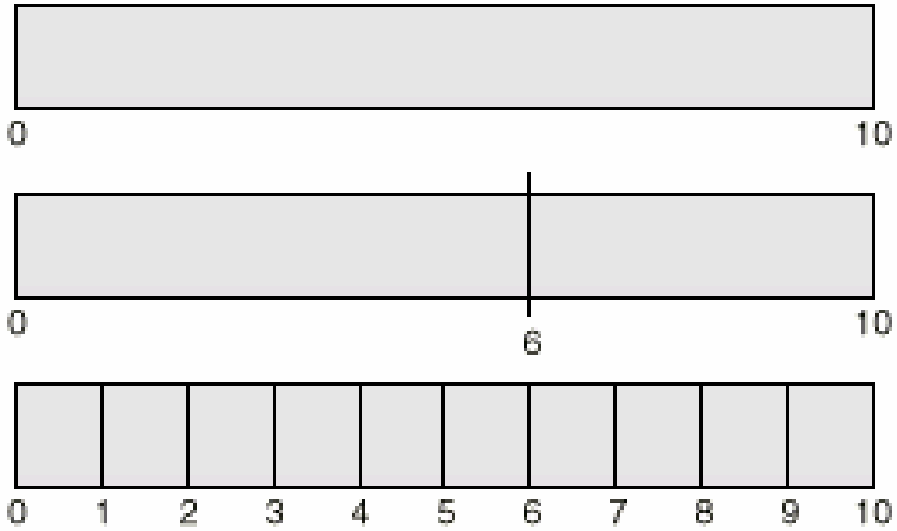
Proceso	Ráfaga
P_1	53
P_2	17
P_3	68
P_4	24

- El diagrama de Gantt:



- Típicamente, más tiempo de retorno promedio que SJF, pero mejor *respuesta*.

Quantum PEQUEÑO Incrementa los Cambios de Contexto



quantum	cambios de contexto
12	0
6	1
1	9

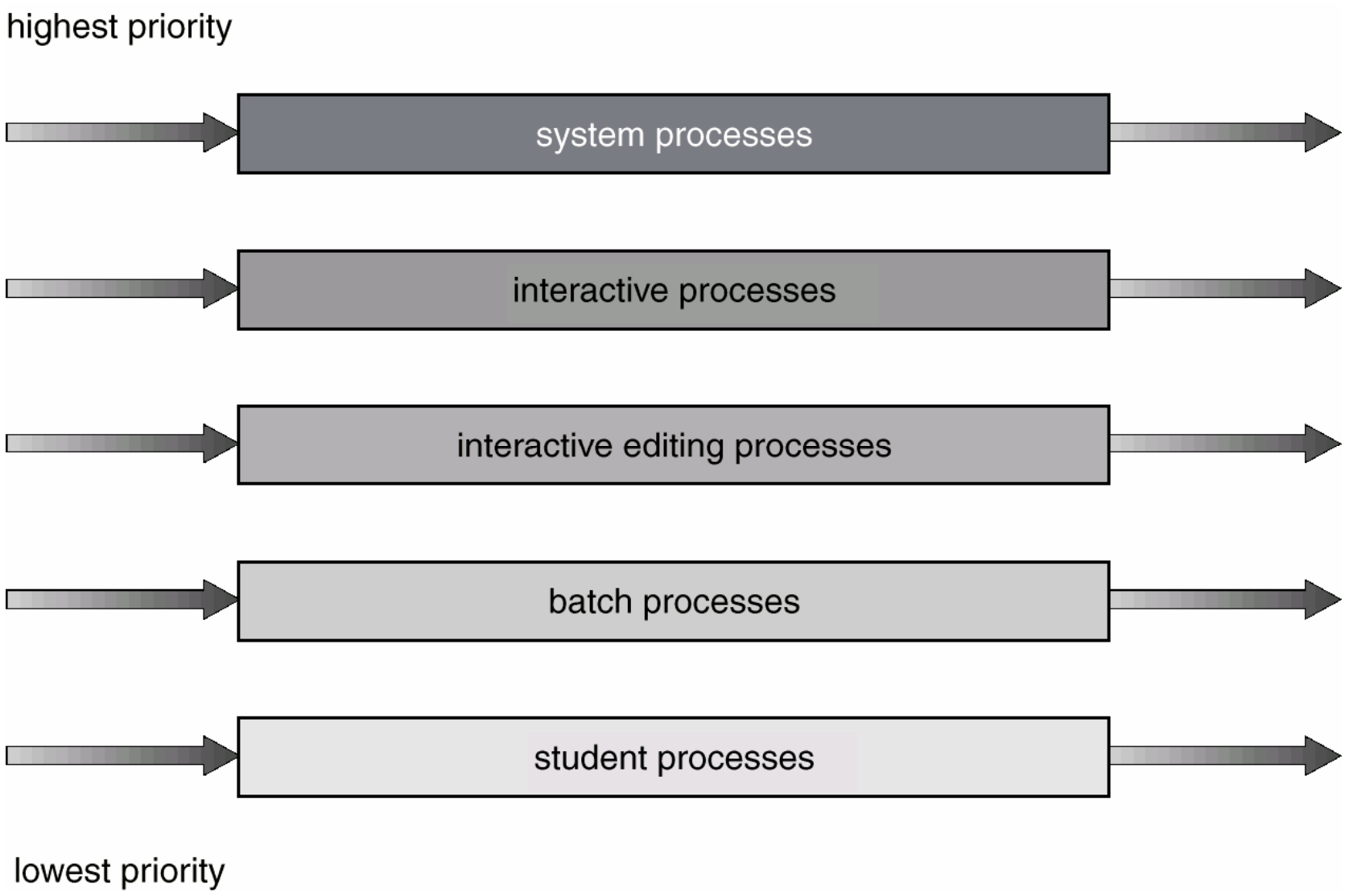
Colas Multinivel (1)

- La cola de listos está particionada en colas separadas:
  - foreground (*interactive*)
  - background (*batch*)
- Cada cola tiene su propio algoritmo de planificación,
  - foreground – RR
  - background – FCFS

Colas Multinivel (2)

- La planificación debe ser hecha entre las colas.
  - Planificación con prioridad fija; p.e., ejecutar desde el foreground y luego del background. Posibilidad de inanición.
  - *Slice* de tiempo – cada cola

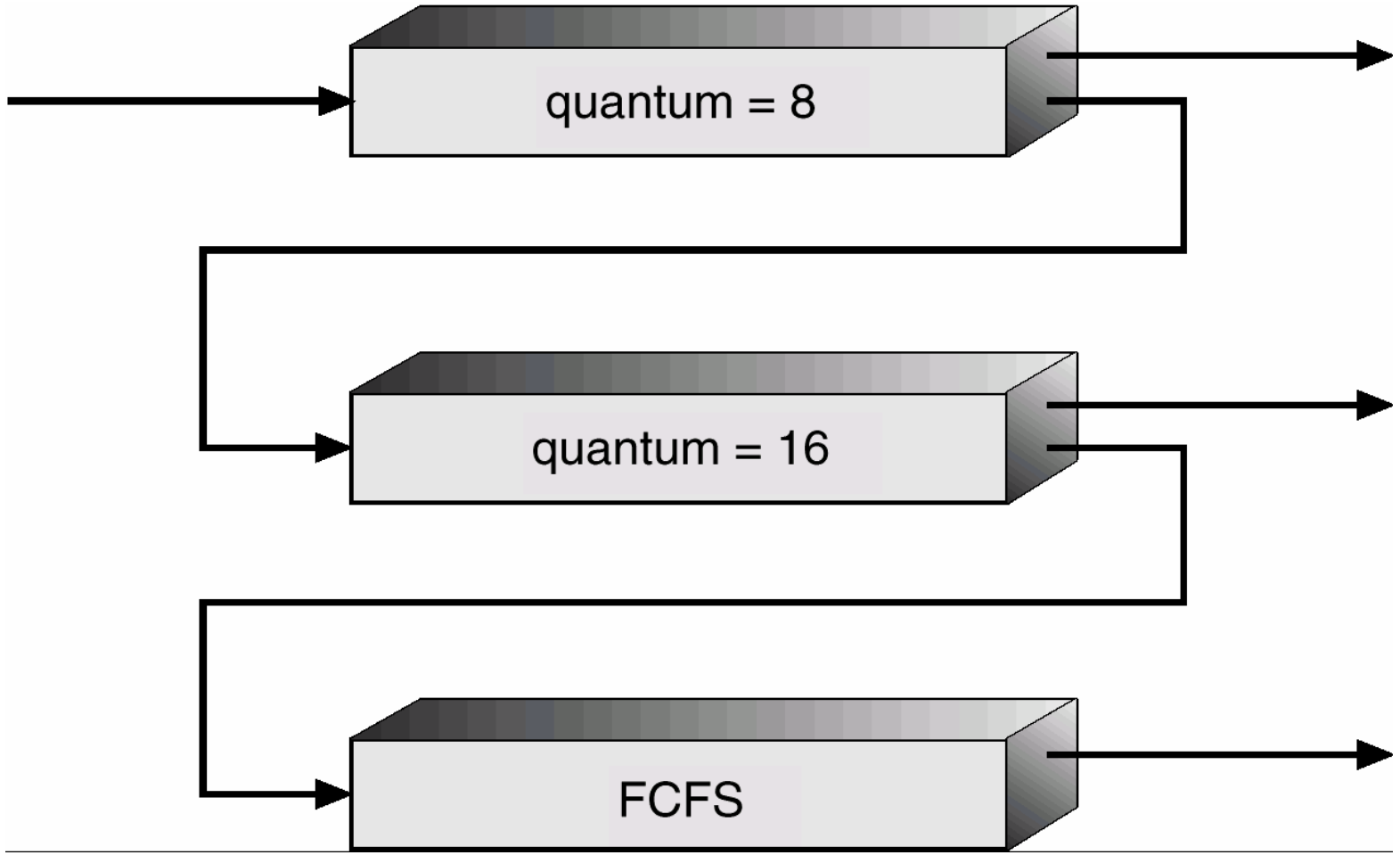
Planificación de Colas Multinivel



Colas Multinivel Realimentadas

- Un proceso puede moverse entre varias colas.
- El planificador de colas multinivel realimentadas está definido por los siguientes parámetros:
  - Número de colas.
  - Algoritmos de planificación para cada cola.
  - Método usado para determinar cuando mejorar un proceso.
  - Método usado para determinar cuando degradar un proceso.
  - Método usado para determinar en que cola entra un proceso cuando necesita servicio.

Colas Multinivel Realimentadas



Planificación Tiempo Real

- *Sistemas de Tiempo Real Duro* – requiere completar tareas críticas en una cantidad de tiempo garantizado.
- *Computación de Tiempo Real Blando* – requiere que los procesos críticos reciban prioridad sobre otros.

Planificación en UNIX (y Linux)

La planificación tradicional en UNIX emplea colas multinivel (los niveles se definen en bandas de prioridades) usando Round Robin en cada una de ellas:

$$P_j(i) = \text{Base}_j + \frac{CPU_j(i)}{2} + \text{nice}_j \qquad (1)$$

$$CPU_j(i) = \frac{CPU_j(i-1)}{2} \qquad (2)$$

Planificación en UNIX (y Linux)

$CPU_j(i)$ = Mide la utilización del procesador por el proceso *j* en el intervalo *i*.

$P_j(i)$ = Prioridad del proceso *j* en el comienzo del intervalo *i*; valores bajos implican prioridades altas.

$Base_j$ = Prioridad base del proceso *j*.

$nice_j$ = Factor de ajuste controlable por el usuario

(1) Es utilizada para ajustar dinámicamente la prioridad (producto del uso de CPU).

(2) Es usada para implementar el "envejecimiento" cuando el proceso espera. Así evita la inanición.

Planificación en UNIX (y Linux)

La prioridad de cada proceso es computada cada segundo (en los primeros UNIX, hoy es cada *quantum*).

El propósito de la prioridad base es dividir todos los procesos en bandas de niveles de prioridad.

Los componentes ***CPU*** y ***nice*** se utilizan para prevenir que los procesos migren fuera de su banda asignada (dada por la prioridad base).

Estas bandas son utilizadas para optimizar el acceso a los dispositivos que se manejan con bloques de información (discos, cintas, CD, etc) y permitir al sistema operativo responder rápidamente a las llamadas a sistema.

Planificación en UNIX (y Linux)

En orden decreciente de prioridad, las bandas son:

- Swapper.
- Control de dispositivos de E/S en bloques.
- Manipulación de archivos.
- Control de dispositivos de E/S por caracteres.
- Procesos de usuarios.

Dentro de la banda de procesos de usuario, el uso de la historia de ejecución tiende a penalizar a los procesos limitados por procesador a expensas de los procesos limitados por E/S.

Planificación de Procesos en Sistemas Distribuidos

Manejo de Recursos en Sistemas Distribuidos

Se diseña un manejador de recursos para:

- Control de asignación de recursos vs. procesos.
- Ruteo de procesos a sitios de acuerdo a la asignación de los mismos.

El objetivo es *optimizar*:

- El uso.
- Tiempo de respuesta.
- Congestión de la red.
- Sobrecarga de la planificación.

Manejo de Recursos en Sistemas Distribuidos

Técnica y metodologías para la planificación de procesos:

- Asignación de tareas.
- Balance de carga.
- Carga compartida.

La primer técnica tiene limitada aplicabilidad.

Manejo de Recursos en Sistemas Distribuidos

Características deseables

Las características deseables para un buen algoritmo de planificación global serían:

- ✓ No tener conocimiento “a priori” sobre los procesos.
- ✓ Dinámico en su naturaleza.
- ✓ Capacidad de tomar decisiones rápidas.
- ✓ Rendimiento de sistema y sobrecarga de la planificación balanceada.
- ✓ Estabilidad.
- ✓ Tolerancia a las fallas.
- ✓ Imparcialidad en el servicio.

Manejo de Recursos en Sistemas Distribuidos

Asignación de tareas

Se considera a un proceso compuesto por múltiples tareas.

Objetivo:

- Minimización de costos de IPC.
- Minimización de tiempos de retorno.
- Alto grado de paralelismo.
- Utilización de los recursos eficiente.

Puede haber conflicto entre estos objetivos.

Manejo de Recursos en Sistemas Distribuidos

Asignación óptima

Se crea un grafo estático de asignación con sitios y tareas como nodos.

Lado *tarea-sitio* es el costo de procesamiento.

Lado *tarea-tarea* es el costo de IPC.

Se busca un *cutset* que sea mínimo con la condición de que los nodos de una partición P_j son todos alcanzables desde n_i.

Manejo de Recursos en Sistemas Distribuidos

Costo de comunicación intertareas						
	t_1	t_2	t_3	t_4	t_5	t_6
t_1	0	6	4	0	0	12
t_2	6	0	8	12	3	0
t_3	4	8	0	0	11	0
t_4	0	12	0	0	5	0
t_5	0	3	11	5	0	0
t_6	12	0	0	0	0	0

Manejo de Recursos en Sistemas Distribuidos

Costos de ejecución		
	sitios	
	n_1	n_2
t_1	5	10
t_2	2	∞
t_3	4	4
t_4	6	3
t_5	5	2
t_6	∞	4

Manejo de Recursos en Sistemas Distribuidos

Asignación serial

Tarea	Sitio
t_1	n_1
t_2	n_1
t_3	n_1
t_4	n_2
t_5	n_2
t_6	n_2

Costo de ejecución = 20

Costo de comunicación = 38

Costo total = 58

Asignación óptima

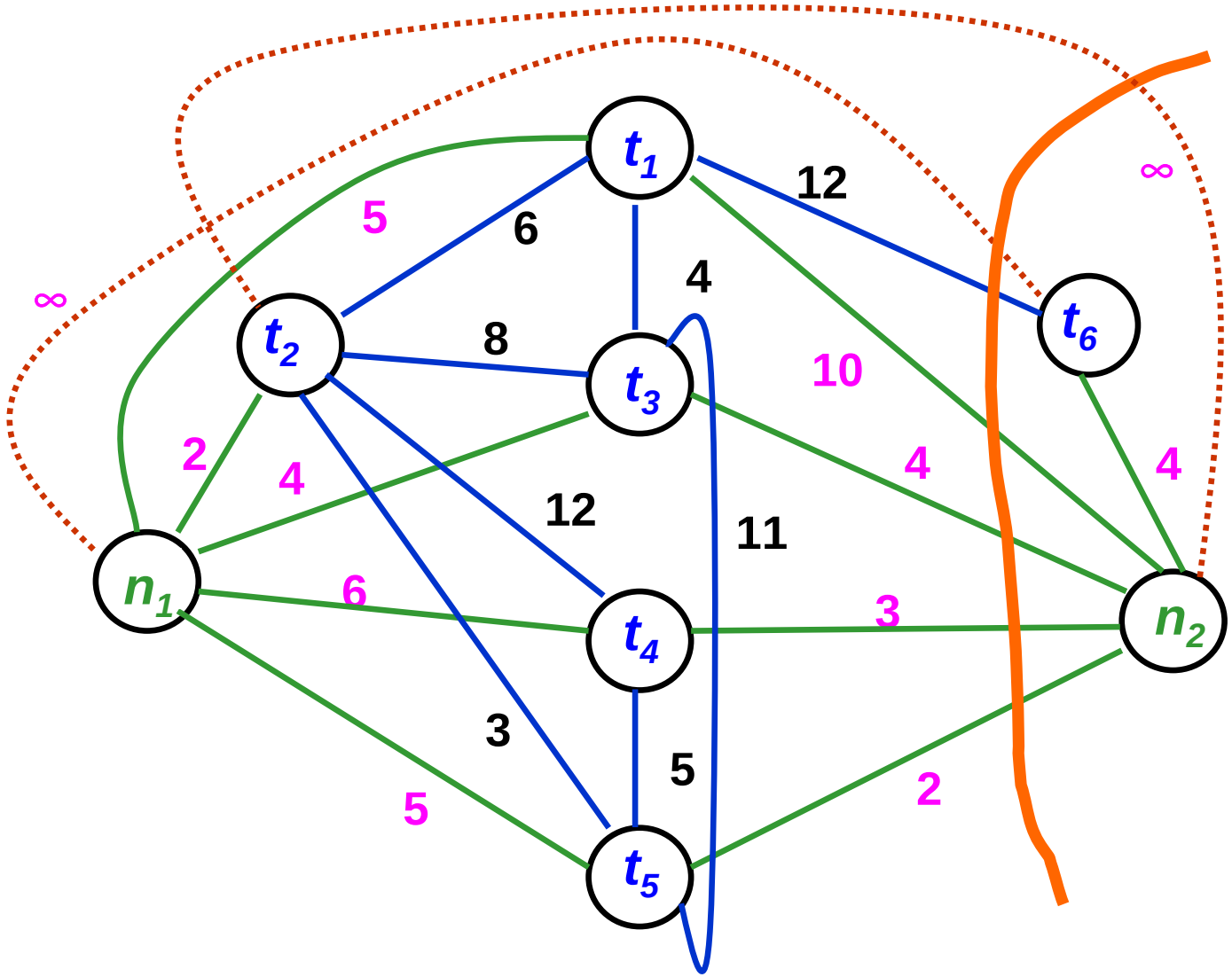
Tarea	Sitio
t_1	n_1
t_2	n_1
t_3	n_1
t_4	n_1
t_5	n_1
t_6	n_2

Costo de ejecución = 26

Costo de comunicación = 12

Costo total = 38

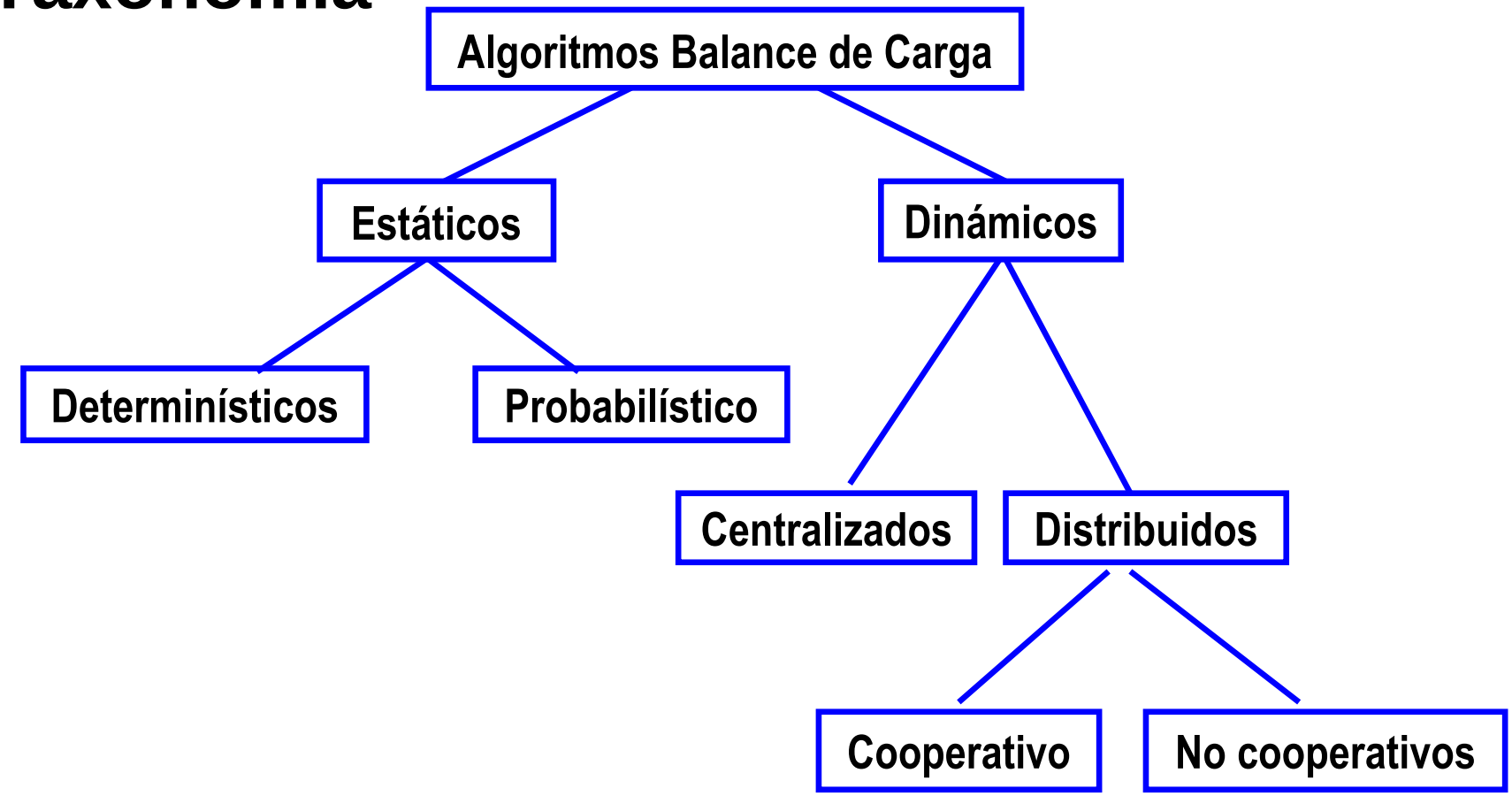
Manejo de Recursos en Sistemas Distribuidos



Manejo de Recursos en Sistemas Distribuidos

Balance de Carga

Taxonomía



Manejo de Recursos en Sistemas Distribuidos

Estáticos vs. Dinámicos

Estáticos: usan información del comportamiento medio del sistema ignorando el estado corriente del mismo.

Dinámicos: reaccionan con el estado del sistema.

Manejo de Recursos en Sistemas Distribuidos

Determinísticos vs. Probabilísticos

Determinísticos: usa info sobre las propiedades de los sitios y características de los procesadores.

Probabilísticos: usa info en cuanto a atributos estáticos del sistema como:

- Número de sitios.
- Capacidad de procesamiento de cada sitio.
- Topología de la red.

Manejo de Recursos en Sistemas Distribuidos

Centralizados vs. Distribuidos

Centralizados: la responsabilidad de la planificación se encuentra en un solo sitio (servidor).

Por confiabilidad se definen *k+1* sitios replicados para soportar *k* fallas.

Distribuidos: la planificación no queda limitada a un solo sitio.

Se compone de *k* identidades físicas distribuidas. Cada una atiende un número de sitios.

Toma las decisiones basándose en una función objetivo de todo el sistema.

Manejo de Recursos en Sistemas Distribuidos

Cooperativos vs. no cooperativos

Cooperativos: las entidades cooperan entre sí (mejor estabilidad).

No cooperativos: cada entidad actúa en forma autónoma respecto a las demás.

Manejo de Recursos en Sistemas Distribuidos

Aspectos a tener en cuenta en el diseño de balance de carga.

Es difícil obtener un “buen algoritmo” porque se debe atender a:

- *Política de estimación de carga* (determina como estimar la carga de un sitio del sistema)
- *Política de transferencia de procesos* (si el proceso se debe ejecutar local o remotamente)
- *Política de intercambio de info* (como intercambiar info entre los sitios).

Manejo de Recursos en Sistemas Distribuidos

- *Política de locación* (determina a que sitio debe enviarse el proceso seleccionado).
- *Política de asignación* (determina la prioridad de ejecución de procesos locales y remotos en un sitio particular.
- *Política que limite la migra*ción (determina el número de veces que un proceso puede migrar de un sitio a otro).

Son ***procesos locales*** a aquellos que se procesan en su sitio original y ***procesos remotos*** aquellos que se procesan en sitios distintos de los originales.

Manejo de Recursos en Sistemas Distribuidos

Política de estimación de carga

La estimación puede basarse en factores dependientes del tiempo o del sitio:

- Número total de procesos en el sitio en el momento de estimación.
- Demanda de recursos de esos procesos.
- Arquitectura y velocidad del sitio procesador.

La mejor estimación es la utilización de CPU.

Manejo de Recursos en Sistemas Distribuidos

Políticas de transferencias de procesos

Idea*: transferir procesos de sitios muy cargados a otros descargados*

Problema: ¿cómo decidir que un sitio está más cargado que otro?

Se utiliza la política del **umbral**.

Éste valor es el límite entre *pesados* y *livianos*.

Manejo de Recursos en Sistemas Distribuidos

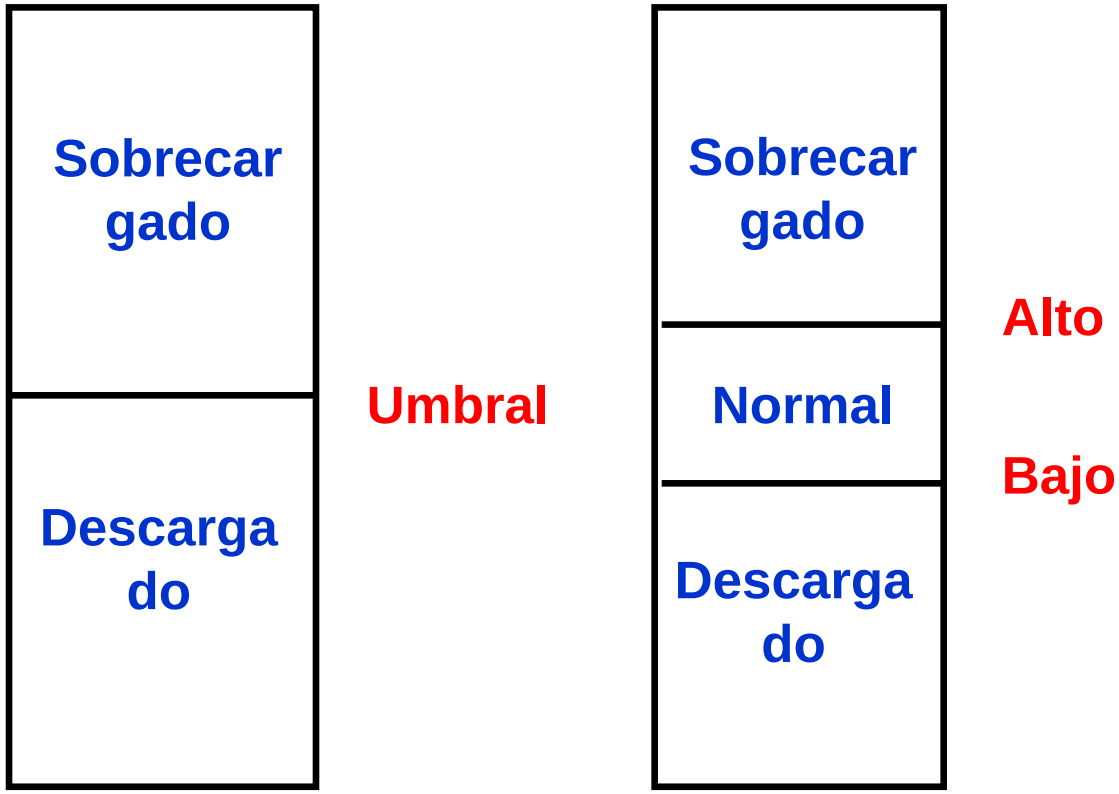
Política estática: cada sitio tiene un valor predefinido de umbral dependiendo de su capacidad de procesamiento. No requiere intercambio de información entre sitios.

Política dinámica: el umbral de cada sitio n_i se calcula como producto de la carga media de todos los sitios y una constante c_i predefinida (depende de la capacidad de procesamiento del sitio n_i relativo a los otros). Los sitios deben intercambiar información sobre sus estados.

Manejo de Recursos en Sistemas Distribuidos

Tener un solo umbral vuelve *inestable* al sistema.

Se aplica una política alto-bajo.



Manejo de Recursos en Sistemas Distribuidos

Carga Compartida

Concepto diferente, no se hace un balance dinámico de carga sino que se reparte dinámicamente la carga.

Entre las consideraciones que deben tenerse en cuenta para el diseño de algoritmos de carga compartida se requieren decisiones propias acerca de las políticas presentadas en el balance de carga.

Lo que se busca es que ningún sitio esté ocioso cuando otros están sobrecargados.

No todas las políticas varían.

Manejo de Recursos en Sistemas Distribuidos

Política de estimación de carga: idem

Política de transferencia de procesos: política alto-bajo.

Política de locación:

- Política de emisor inicial: decide donde enviar.
- Política de receptor inicial: decide de donde tomar el proceso.

Manejo de Recursos en Sistemas Distribuidos

Ambas políticas ofrecen mejoras sustanciales del rendimiento.

Se prefiere la *política de emisor* en cargas moderadamente ligeras.

Se prefiere la *política de receptor* en cargas altas solo si el costo de transferencia es comparable.

Si el costo de transferencia de un proceso bajo la *política de receptor* es significativamente mayor que bajo la política de emisor debido a las transferencias apropiativas, la *política de emisor* provee mejor rendimiento uniforme.

Manejo de Recursos en Sistemas Distribuidos

Política de intercambio de información entre estados de carga:

- *Broadcast* cuando el estado de carga cambia.
- *Polling* cuando el estado de carga cambia.

Manejo de Procesos en Sistemas Distribuidos

Manejo de Procesos

En Sistemas Operativos convencionales se trata de la forma de compartir el procesador del sistema entre los procesos.

En Sistemas Distribuidos el objetivo es similar:

Hacer el mejor uso posible de todos los recursos de procesamiento del sistema.

Manejo de Procesos en Sistemas Distribuidos

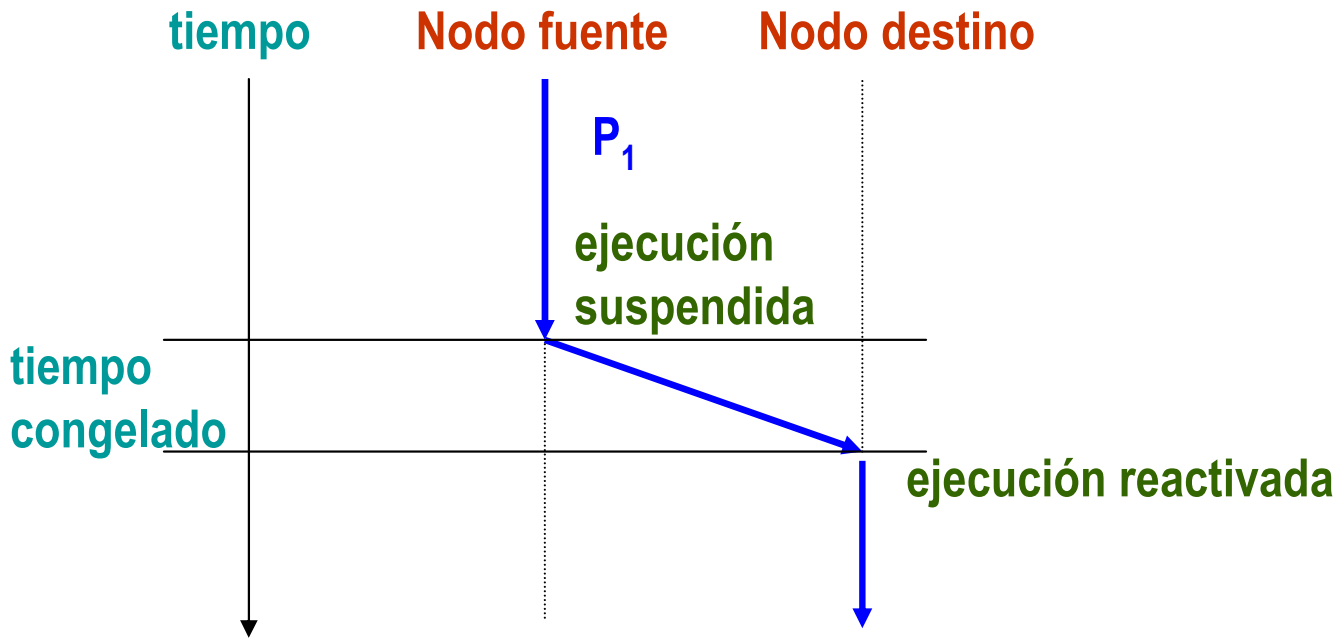
Tres conceptos se usan para lograr este objetivo:

- Asignación de procesos: qué proceso debe asignarse a qué procesador.
- Migración de procesos: movimiento del proceso al procesador que ha sido asignado.
- Threads: paralelismo mas fino para mejor utilización de la capacidad de procesamiento del sistema.

Manejo de Procesos en Sistemas Distribuidos

Migración de Procesos

Es la relocación de un proceso de su locación corriente (*sitio fuente*) en otro sitio (*sitio destino*).



Manejo de Procesos en Sistemas Distribuidos

Un proceso puede migrar antes de comenzar a ejecutar (*no apropiativo*) o durante el curso de su ejecución (*apropiativo*).

Involucra:

- Selección de un proceso a ser migrado.
- Selección de un sitio destino a donde el proceso debe ser migrado.
- Transferencia del proceso seleccionado al sitio destino.

Manejo de Procesos en Sistemas Distribuidos

Características deseables de un buen mecanismo de migración de procesos

- *Transparencia.*
- *Mínima interferencia.*
- *Dependencias residuales mínimas.*
- *Eficiencia.*
- *Robustez.*
- *Comunicación entre coprocesos de un Job.*

Manejo de Procesos en Sistemas Distribuidos

Transparencia

- *Nivel de acceso a objetos* mínimo requerido para migración no apropiativa.
- *Nivel de system-calls y comunicación entre procesos.*

Interferencia Mínima

Debe tratar de minimizar el tiempo congelado.

Manejo de Procesos en Sistemas Distribuidos

Dependencia residual mínima

La migración de procesos debe dejar lo mínimo o nada en el sitio origen.

- Impone una carga en el sitio previo.
- Una falla o *reboot* en el sitio previo hace fallar el proceso.

Manejo de Procesos en Sistemas Distribuidos

Eficiencia

La fuentes de mayores deficiencias son:

- el tiempo requerido para migrar el proceso.
- el costo de localizar el objeto.
- el costo de soportar la ejecución remota una vez que el proceso ha migrado.

Robustez

La falla en un sitio distinto del que corre el proceso no debe afectar la accesibilidad o ejecución de ese proceso.

Manejo de Procesos en Sistemas Distribuidos

Comunicación entre procesos de un Job

Cuando un Job se divide en varios coprocesos paralelos y se ejecutan en distintos sitios.

Para reducir los costos de comunicación es necesario que esos coprocesos se comuniquen directamente unos con otros independiente de su locación.

Manejo de Procesos en Sistemas Distribuidos

Mecanismos de Migración de Procesos

Involucra varias subactividades:

- Congelar el proceso en su sitio origen y reiniciarlo en su sitio destino.
- Transferir el espacio de direcciones correspondiente.
- Continuar los mensajes esperados por el proceso migrante.
- Manejar las comunicaciones entre procesos cooperativos que han sido separados como resultado de la migración.

Manejo de Procesos en Sistemas Distribuidos

Mecanismos para congelar y reiniciar un proceso

Bloqueo inmediato o retardado del proceso.

Dependiendo del estado del proceso puede ser bloquea-do inmediatamente o esperar. **Casos:**

- Si el proceso no está ejecutando un system-call puede ser bloqueado inmediatamente.
- Si el proceso está ejecutando un system-call pero durmiendo en una prioridad interrumpible esperando que ocurra un evento en el kernel, puede ser bloqueado inmediatamente.
- Si el proceso está ejecutando un system-call y está en una prioridad no interrumpible esperando que ocurra un evento en el kernel no puede ser bloqueado inmediatamente.

Manejo de Procesos en Sistemas Distribuidos

Operaciones de E/S rápidas y lentas

Debe solo esperar por las operaciones rápidas.

Respecto a las lentas, debe establecerse un mecanismo para continuarlas una vez migrado el proceso.

Información sobre archivos abiertos

Alternativa 1: Se crea un *link* al archivo, este es usado por proceso migrado.

Alternativa 2: Se recompone el nombre completo del archivo.

Manejo de Procesos en Sistemas Distribuidos

Reinstanciación del proceso en su sitio destino.

En el sitio destino se crea un PCB vacío, con identificación diferente para que existan dos copias, luego se cambia al viejo.

Puede ser necesario reiniciar los system-calls que contenía una operación de E/S lenta.

Manejo de Procesos en Sistemas Distribuidos

Mecanismos de transferencia del espacio de direcciones.

Se debe transferir:

- El estado del proceso.
- El espacio de direcciones del proceso.

Manejo de Procesos en Sistemas Distribuidos

Estado del Proceso

- Estado de ejecución
- Info de planificación
- Memoria usada
- Estados de entrada/salida
- Lista de objetos a los cuales el proceso tiene acceso
- Identificador del proceso
- Identificador de usuario y grupo del proceso
- Archivos abiertos

Manejo de Procesos en Sistemas Distribuidos

Espacio de direcciones del proceso

- Código
- Datos
- Stack

El costo de la migración está dado por la transferencia del espacio de direcciones.

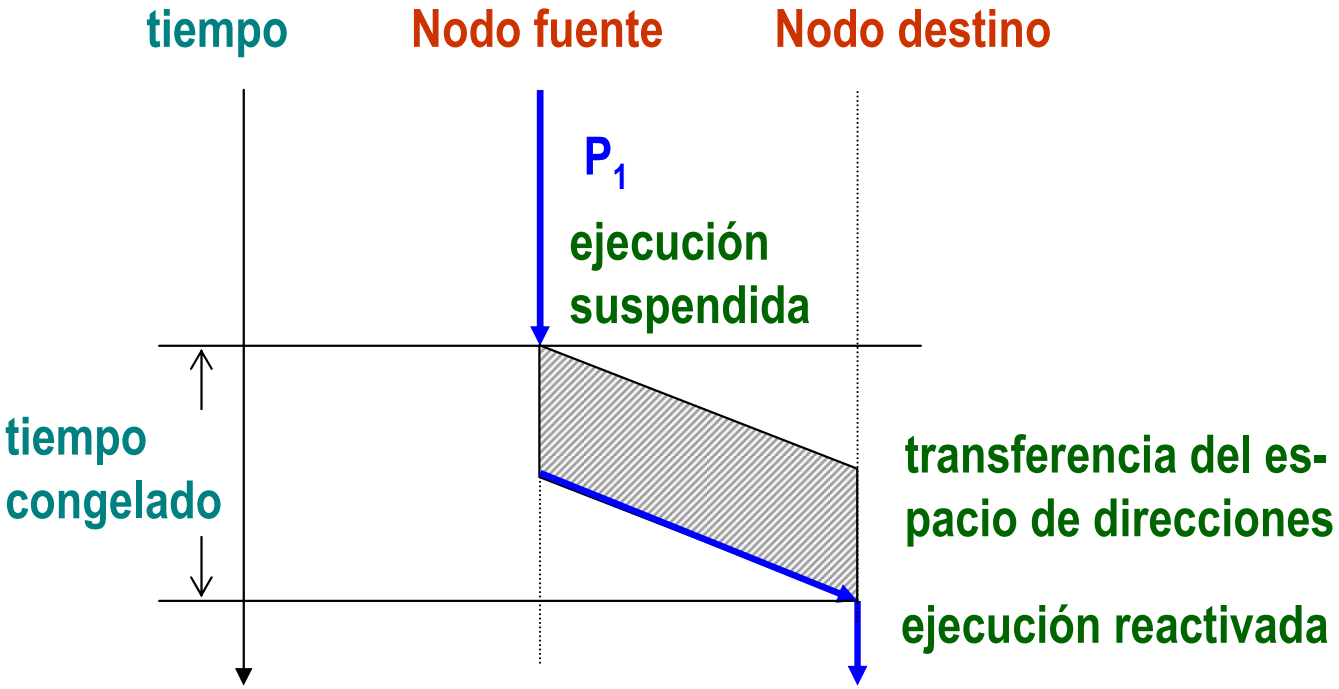
Manejo de Procesos en Sistemas Distribuidos

Se usan diferentes mecanismos para la transferencia del espacio de direcciones:

- congelamiento total
- pretransferencia
- transferencia por referencia

Manejo de Procesos en Sistemas Distribuidos

Congelamiento Total

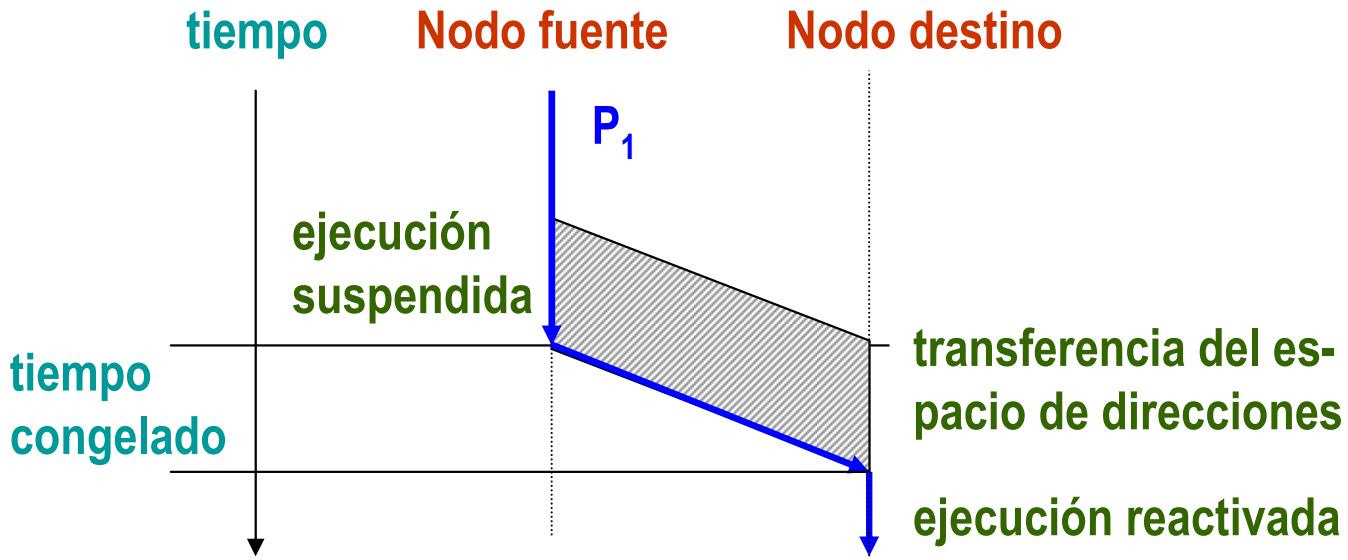


Ventaja: Fácil de implementar

Desventaja: Se pueden vencer los time-outs y los usuarios pueden notarlo.

Manejo de Procesos en Sistemas Distribuidos

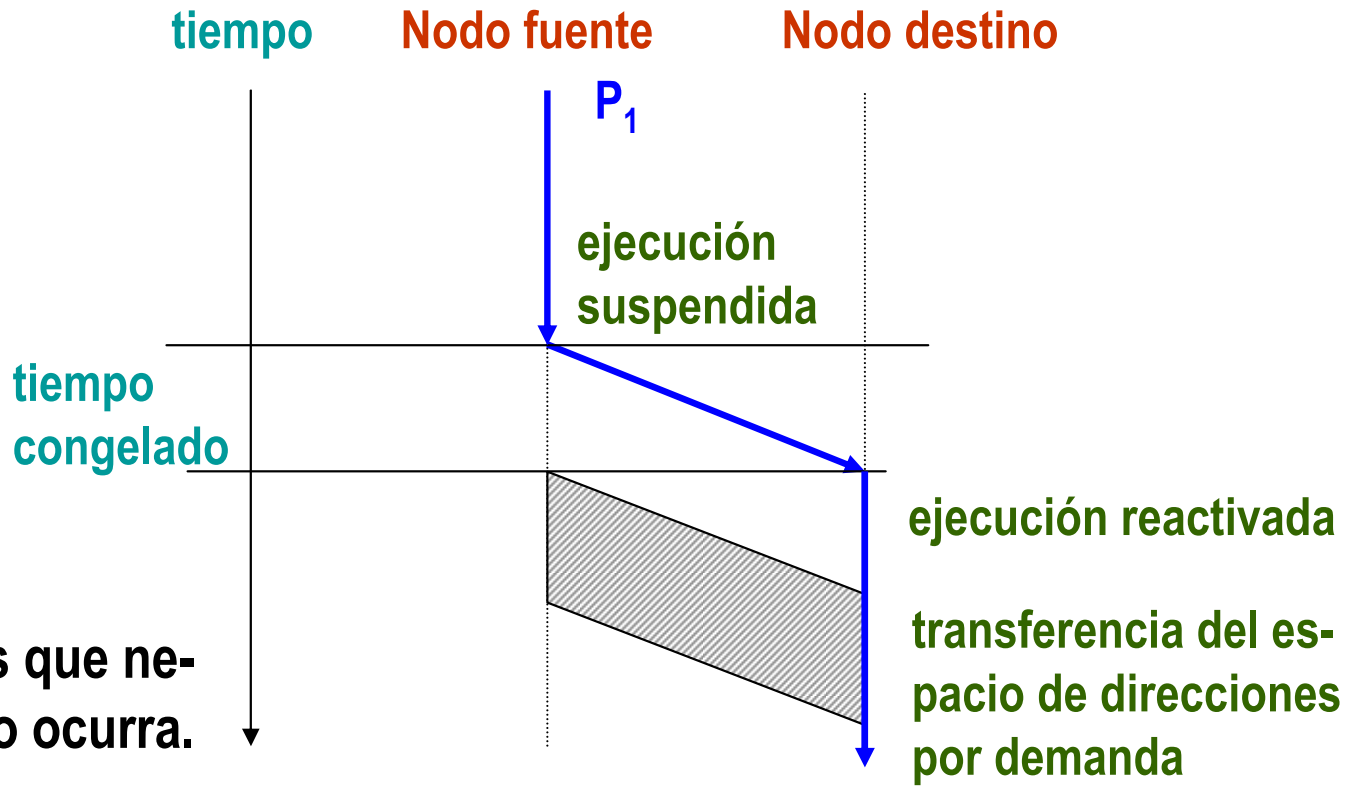
Pretransferencia



Puede transferir ***páginas redundantes***

Manejo de Procesos en Sistemas Distribuidos

Transferencia por demanda



Copia las páginas que necesita cuando ello ocurra.

Deja información en el sitio fuente, *esto es crítico*.

Manejo de Procesos en Sistemas Distribuidos

Mecanismos para envío de mensajes

Cuando se mueven los procesos debe asegurarse que lleguen a su nueva locación:

- ✓mensajes en ruta
- ✓mensajes pendientes
- ✓futuros mensajes

Manejo de Procesos en Sistemas Distribuidos

Los mensajes al proceso migrante pueden ser clasificados como:

Tipo 1: recibidos en el sitio fuente luego que se ha congelado el proceso y no se ha reiniciado en el sitio destino.

Tipo 2: recibidos en el sitio fuente cuando el proceso se inició en el sitio destino.

Tipo 3: enviados al proceso desde otros sitios luego que éste reinició en el sitio destino.

Manejo de Procesos en Sistemas Distribuidos

Los mecanismos usados son:

Reenvío de mensajes

(V-System, Amoeba)

Tipos 1 y 2 son retornados o dejados caer para que retransmitan.

Sitio origen

(AIX TCF, Sprite)

Cada sitio tiene info del traslado del proceso.

Es inseguro por caídas de los sitios intermedios, los sitios origen siguen cargados.

Manejo de Procesos en Sistemas Distribuidos

Enlace transversal

(DEMOS/MP)

Los tipo 1 son encolados en el sitio fuente, luego de notificada la ubicación del proceso le son enviados como parte del proceso de migración.

Para los tipo 2 y 3 es dejada una dirección adelantada en el sitio fuente apuntado al sitio destino llamada ***link***.

ID del Proceso (único)	última locación del proceso

Manejo de Procesos en Sistemas Distribuidos

Actualización del link

(Charlotte)

Desde el sitio origen se manda a todos los demás kernels un mensaje de actualización de la nueva ubicación.

Manejo de Procesos en Sistemas Distribuidos

Mecanismos para manejar coprocesos

Deshabilitar la separación de coprocesos

a) No permitir migración de procesos que esperan que completen uno o más subprocesos.

b) Asegurar que todos migren juntos.

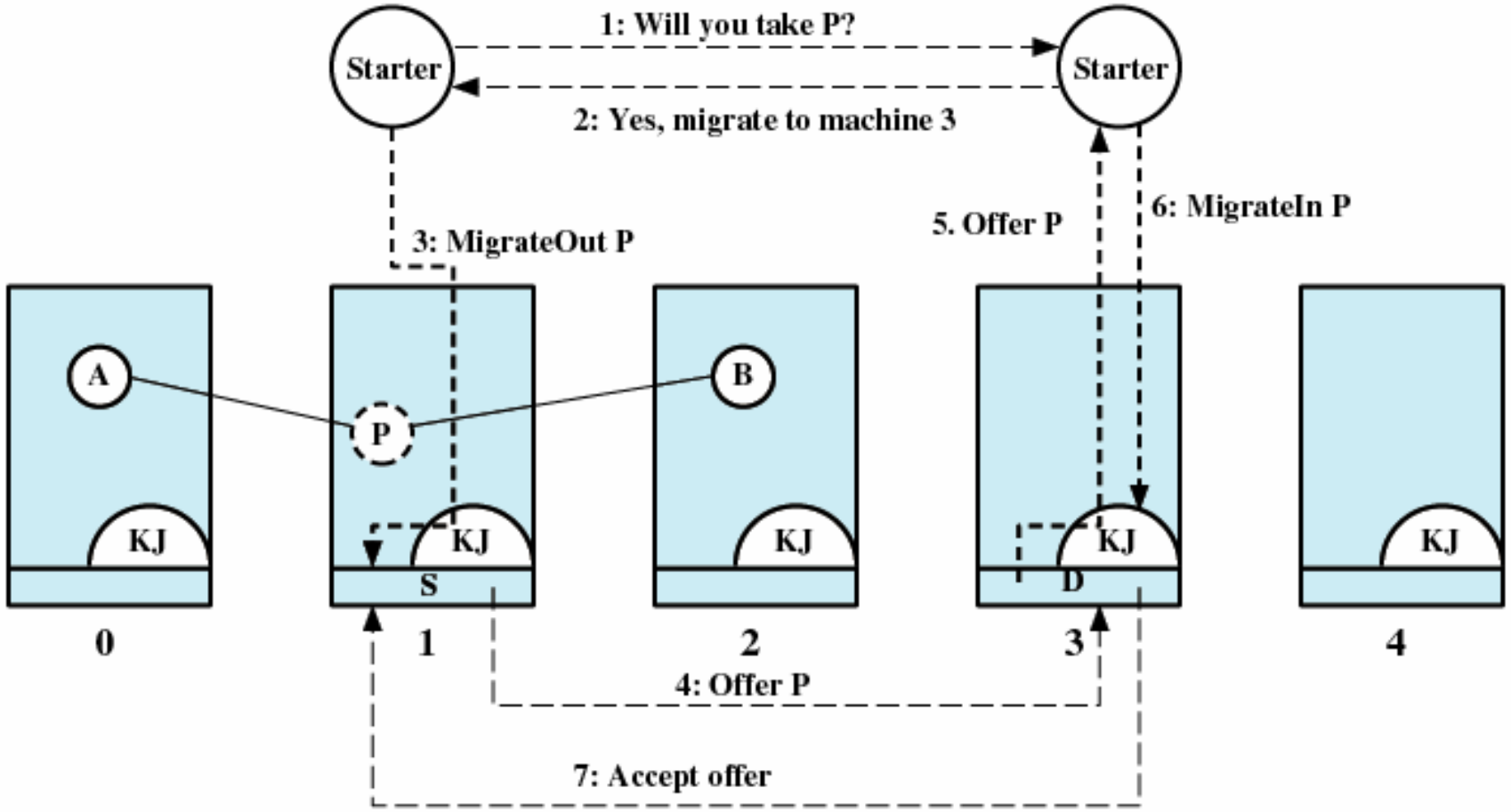
Concepto del sitio origen

La comunicación se lleva a cabo por sitio origen.

Ejemplo de Negociación en Migración

- La política de migración es responsabilidad del Starter (es un utilitario).
- El Starter es también responsable de la planificación de largo término y la alocación de memoria.
- La decisión de migrar debe ser alcanzada conjuntamente entre dos procesos Starter (uno en la fuente y otro en el destino).

Ejemplo de Negociación en Migración



Ejemplo de Negociación en Migración

- El sistema destino puede rechazar aceptar la migración de un proceso sobre él.
- Si la estación de trabajo está ociosa, el proceso puede haber sido migrado hacia allí.
  - Una vez que el usuario de esa estación de trabajo la activa es necesario rechazar el proceso migrado para proveer un adecuado tiempo de respuesta.

Manejo de Procesos en Sistemas Distribuidos

Ventajas de la Migración de Procesos

- Reducción del tiempo medio de respuesta.
- Aceleración de jobs individuales.
- Ganancia de procesamiento total.
- Efectiva utilización de recursos.
- Reducción de tráfico en la red.
- Mejora de la confiabilidad del sistema.
- Mejora de la seguridad del sistema.

Clusters

- Es una alternativa al multiprocesamiento simétrico (SMP)
- Es un grupo de computadoras interconectadas trabajando juntas como un recurso unificado.
  - La visión es como si fuera una única máquina.
  - Cada una puede correr su propio sistema.

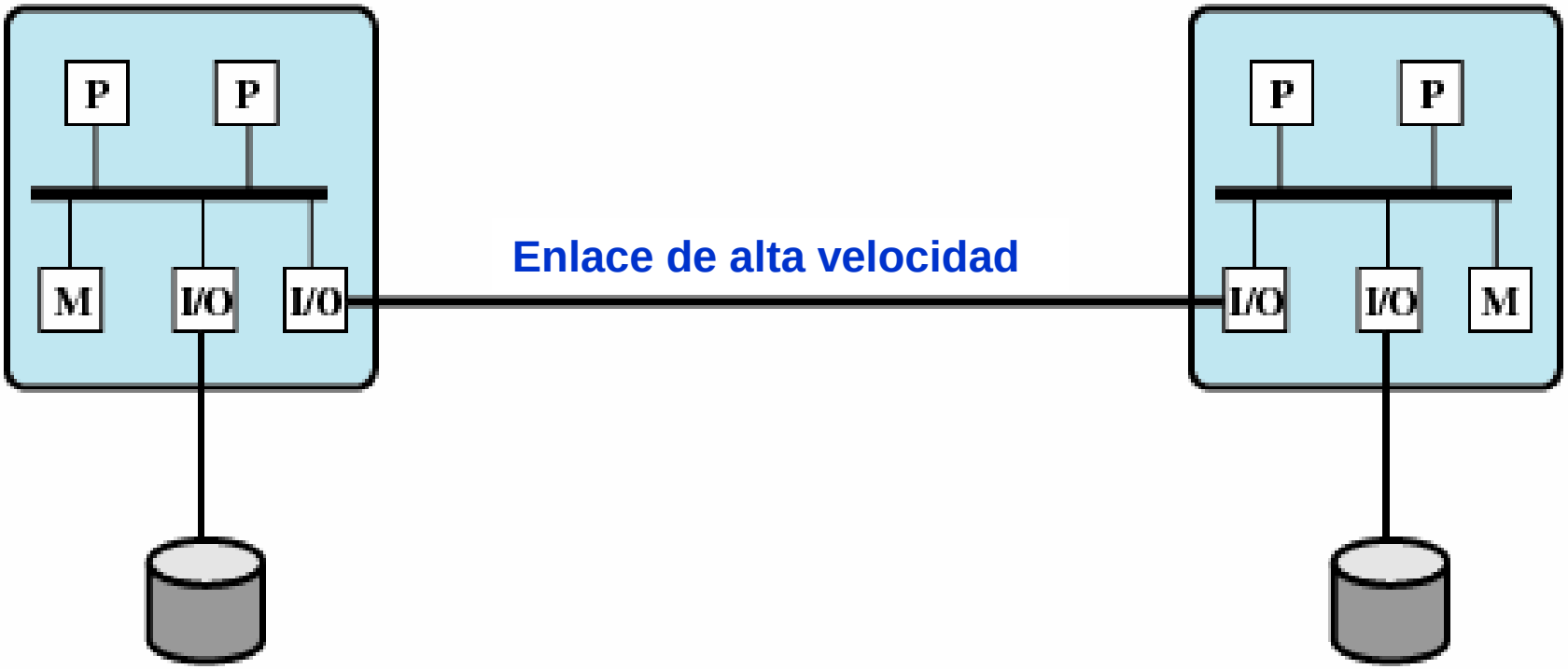
Clusters

Método de Clustering	Descripción	Beneficios	Limitaciones
Standby Pasivo	Un servidor secundario toma el servicio en caso de falla del servidor primario.	Fácil de implementar.	Alto costo a causa de la indisponibilidad para el procesamiento de otras tareas.
Secundario Activo	El servidor secundario es también utilizados para otras tareas de procesamiento.	Costo reducido dado que el servidor secundario puede ser utilizado para procesamiento.	Incrementa la complejidad.
Servidores separados	Servidores separados tienen sus propios discos. Los datos son continuamente copiados del primario al secundario.	Alta disponibilidad.	Alta sobrecarga de red y servidores debido a las operaciones de copia.
Servidores Conectados a discos	Los servidores están conectados a los mismos discos pero cada uno tiene los propios. Si un servidor falla sus discos son tomados por el otro.	Reducida actividad de red y sobrecarga del servidor debido a la eliminación de las operaciones de copia.	Usualmente requiere discos espejados o tecnología RAID para compensar el riesgo de fallas de disco.
Servidores comparten discos	Múltiples servidores comparten simultaneamente el acceso a discos.	Baja sobrecarga de red y servidores. Reduce el tiempo de parada en caso de falla en algún disco.	Requiere un software de administración de locking. Usualmente usado con discos espejados o tecnología RAID.

Clusters

- Servidor separado
  - Cada computadora es un servidor separado.
  - Los discos no son compartidos.
  - Necesita software de administración o planificación.
  - Los datos deben ser copiados constantemente entre los sistemas de modo que todos tengan los datos corrientes.

Configuraciones de Cluster

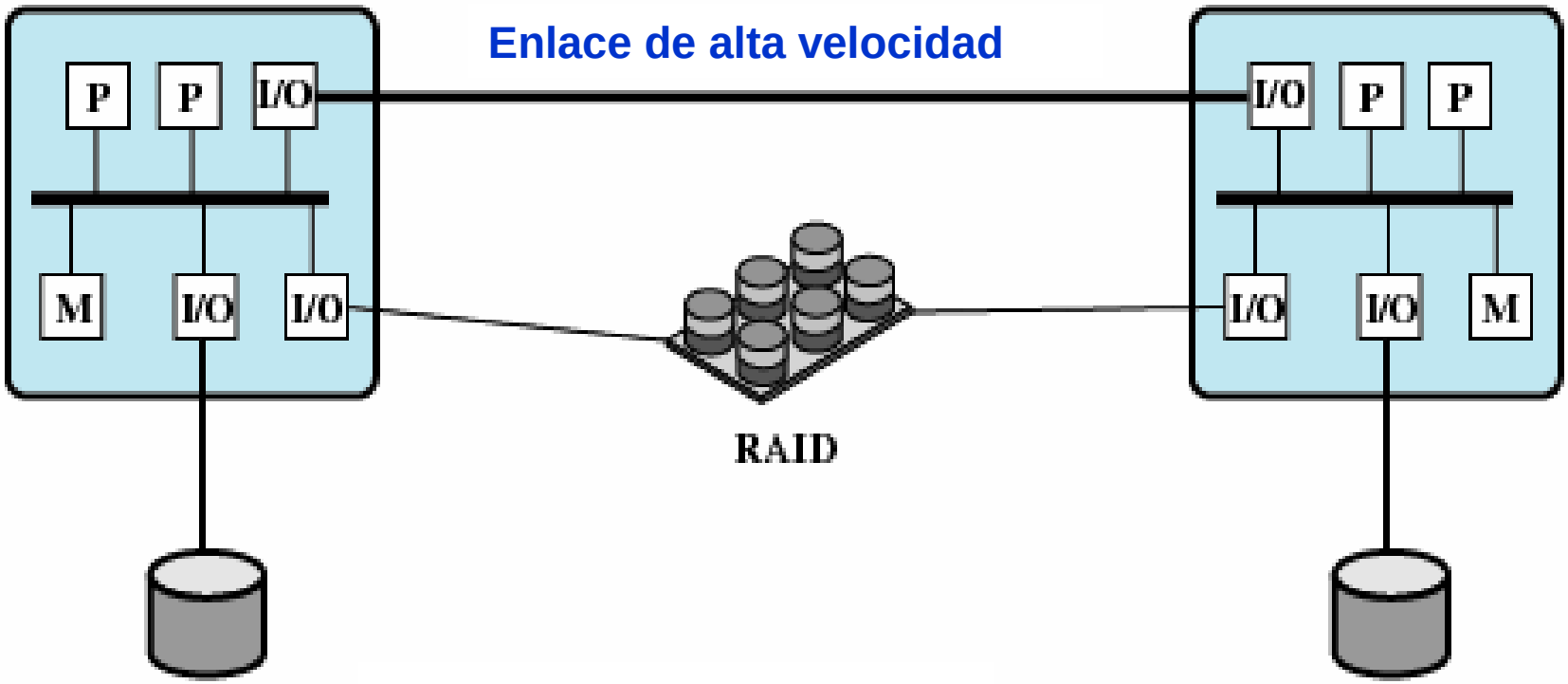


(a) Servidor Stanby con discos no compartidos

Clusters

- Nada compartido
  - Reduce la sobrecarga de comunicación
  - Los discos son particionados en volúmenes
  - Cada volumen es propiedad de alguna computadora
  - Si una computadora falla otra computadora toma la propiedad del volumen

Configuraciones de Cluster



(b) Disco compartido

(b) Shared disk

Clusters

- Disco compartido
  - Múltiples computadoras comparten los mismos discos al mismo tiempo
  - Cada computadora tiene acceso a todos los volúmenes sobre todos los discos

Aspectos de Diseño de Sistema Operativo (clusters)

- Administración de Fallas
  - La alta disponibilidad de un cluster ofrece una alta probabilidad que todos los recursos estarán en servicio.
    - Si una falla ocurre no hay garantías acerca del estado de transacciones parcialmente ejecutadas.
  - Un cluster tolerante a las fallas asegura que todos los recursos están siempre disponibles.

Aspectos de Diseño de Sistema Operativo (clusters)

- Balance de Carga
  - Cuando es agregada una nueva computadora al cluster, la facilidad del balance de carga fácilmente debería incluir a esta computadora en las aplicaciones de planificación

Aspectos de Diseño de Sistema Operativo (clusters)

- Computación Paralela

  En algunos casos el uso efectivo de un cluster requiere ejecutar software de una aplicación simple en paralelo, hay tres propuestas para atacar el problema:

  - Compilador “paralelizante”
  - Aplicación paralelizada
  - Computación paramétrica

Aspectos de Diseño de Sistema Operativo (clusters)

- Compilador “paralelizante”

  Un compilador “paralelizante” determina, en tiempo de compilación, que partes de una aplicación pueden ser ejecutadas en paralelo. El programa es dividido para ser asignado a diferentes computadoras en el cluster. El rendimiento depende de la naturaleza del problema y que tan bien está diseñado el compilador.

Aspectos de Diseño de Sistema Operativo (clusters)

- Aplicación paralelizada

El programador escribe la aplicación para ser corrida sobre un cluster y cuando son requeridos datos usa pasaje de mensajes para mover éstos entre los distintos nodos del cluster. Esto crea un gran problema para el programador pero en algunas aplicaciones es lo mejor para lograr una mejor explotación del cluster.

Aspectos de Diseño de Sistema Operativo (clusters)

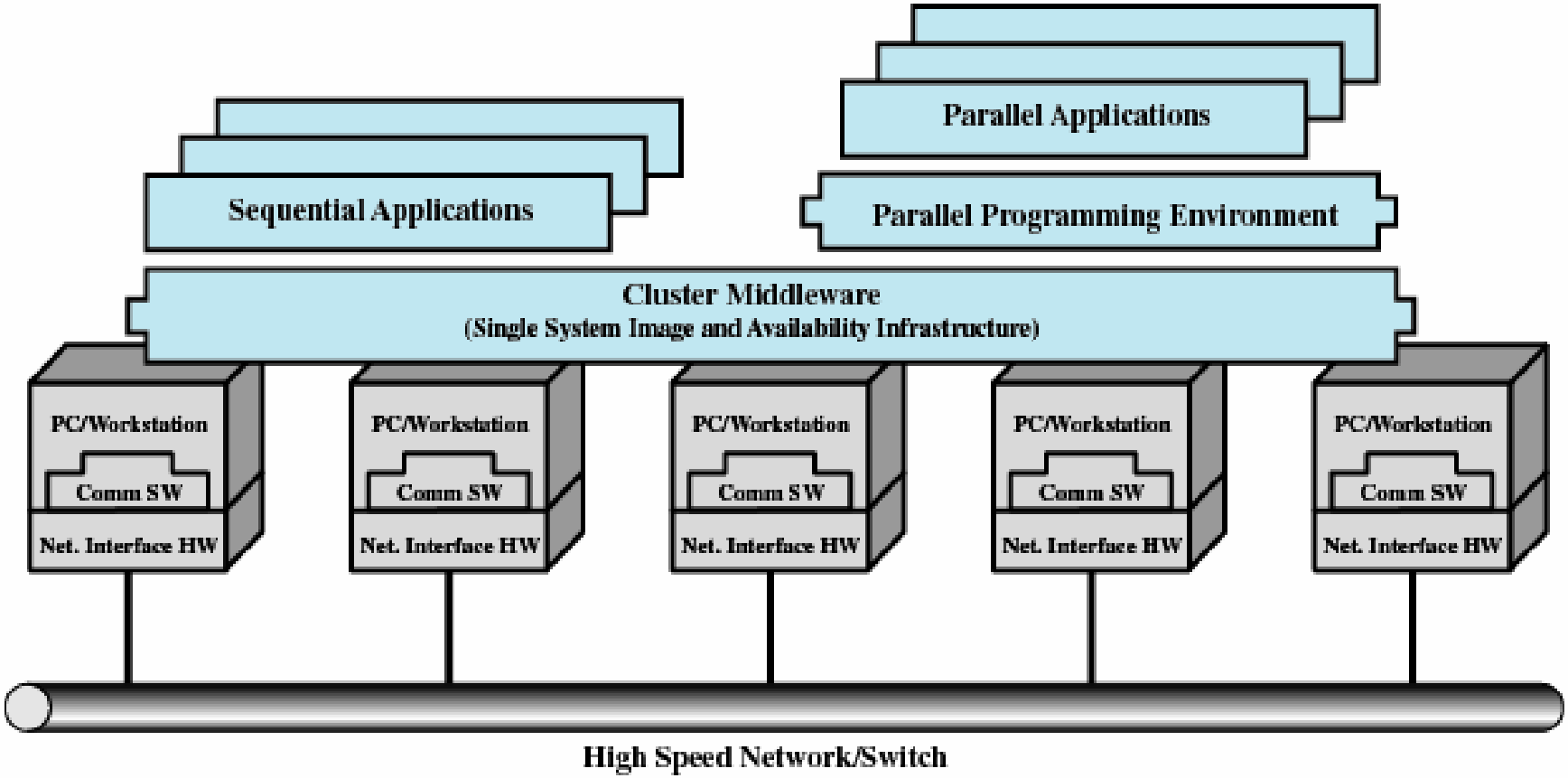
- Computación paramétrica

Esta técnica es usada cuando la esencia de la aplicación es un algoritmo que debe ser ejecutado un gran número de veces, cada vez con un conjunto diferente de condiciones iniciales o parámetros. Un ejemplo es un modelo de simulación, el cual ejecuta un gran número de escenarios diferentes y determina resúmenes estadísticos de los resultados. Para esta forma de trabajo se necesitan herramientas de procesamiento paramétrico para organizar, correr y administrar la tarea de manera ordenada.

Arquitectura de Cluster de Computadoras

- Servicios y funciones de middleware del cluster
  - Único punto de entrada
  - Jerarquía de archivos única
  - Único punto de control
  - Red virtual única
  - Espacio de memoria unificado
  - Administración de tareas de sistema unificado
  - Interfaz de usuario única
  - Espacio de E/S único
  - Espacio de proceso único
  - Checkpointing
  - Migración de procesos

Arquitectura de Cluster de Computadoras



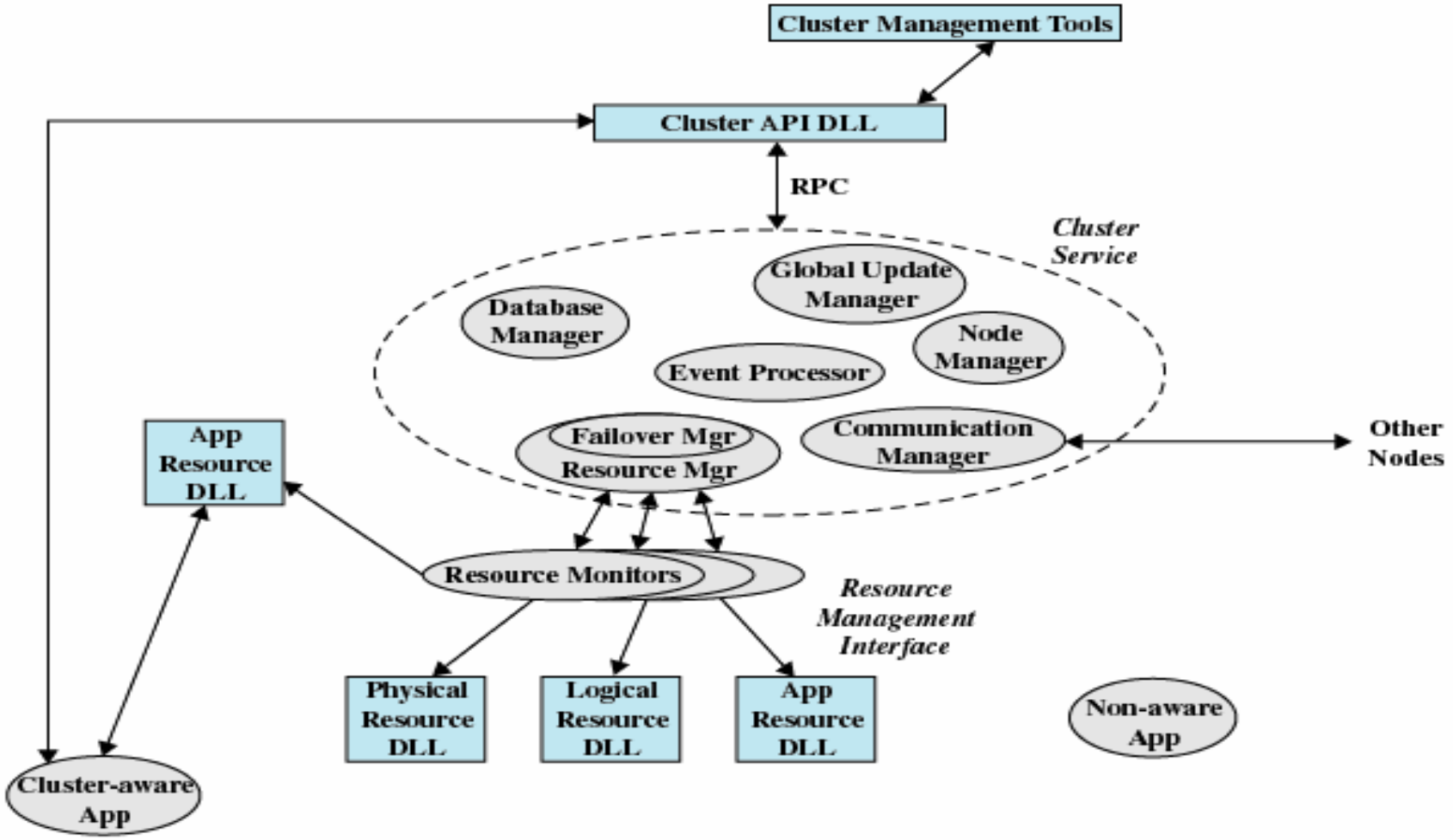
Clusters Comparados con SMP

- SMP es mas fácil de administrar y configurar
- SMP utiliza menos espacio y consume menos energía
- Los productos para SMP están bien establecidos y son estables
- Los clusters son mejores para desarrollo incremental y escalabilidad
- Los clusters son superiores en términos de disponibilidad

Servicios de Cluster de Windows (Wolfpack)

- Servicios del cluster
  - Colección de software en cada nodo que administra toda la actividad específica del cluster.
- Recursos
  - Items administrados por el servicio de cluster.
- Online
  - Un recurso se dice “online” en un nodo cuando esta provee un servicio en ese nodo específico.
  - Colección de recursos administrados como una unidad única.

Servicios de Cluster de Windows (Wolfpack)

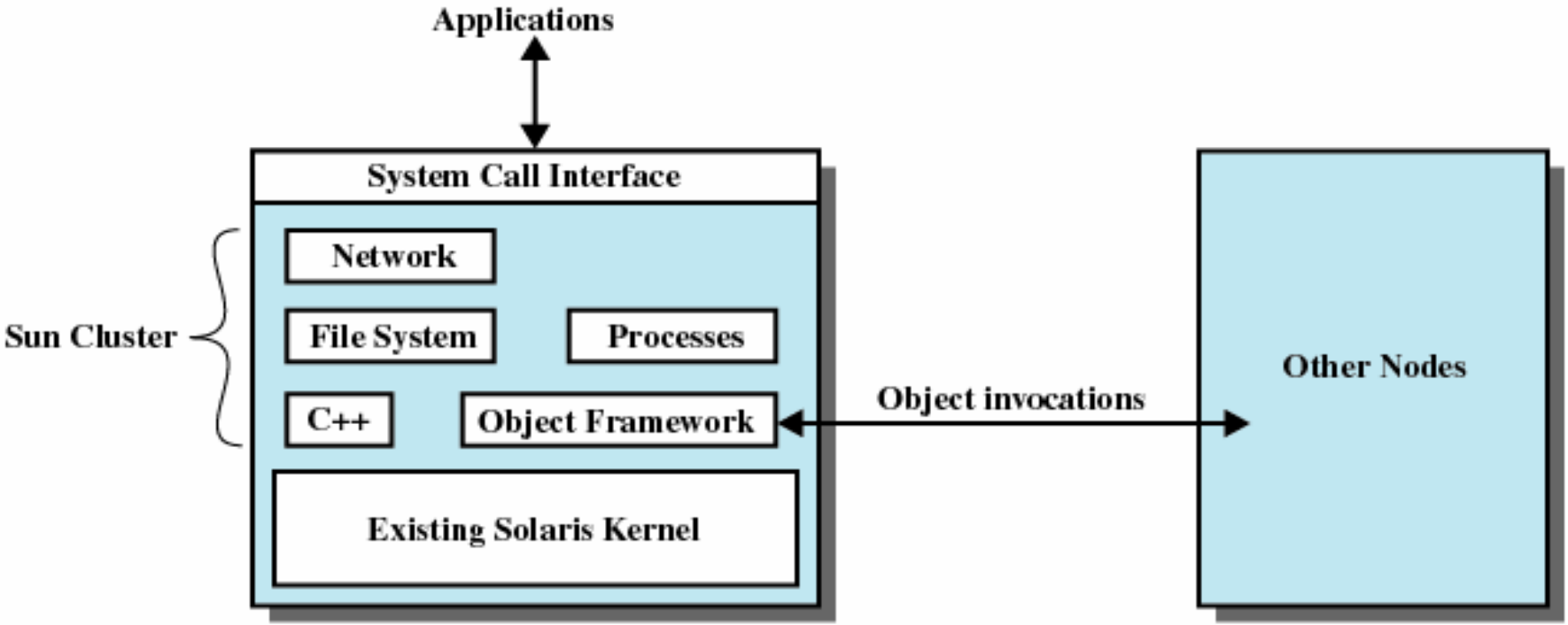


Sun Cluster

El Sun Cluster es un sistema operativo distribuido construido con una serie de extensiones sobre la base del sistema operativo Solaris. Provee una cluster con una imagen de sistema única.

- Componentes Principales
  - Soporte de objetos y comunicaciones.
  - Administración de procesos.
  - Networking.
  - Sistema de archivos distribuidos global.

Sun Cluster


Applications
System Call Interface
Network
File System
Processes
C++
Object Framework
Existing Solaris Kernel
Sun Cluster
Object invocations
Other Nodes

Sun Cluster

- Soporte de objetos y comunicaciones

La implementación es orientada a objetos. Se usa el modelo de objetos CORBA para definir objetos y el mecanismo de RPC.

El uso de un modelo de objetos uniforme e IDL (Interface Definition Language de CORBA) proveen un mecanismo para la comunicación interprocesos, internodos e intranodos.

Todo está montado sobre el tope del kernel de Solaris sin ningún cambio en el mismo.

Sun Cluster

- Administración de procesos

La administración global de procesos extiende sus operaciones de tal forma que la locación de los mismos es transparente para los usuarios. Sun Cluster mantiene una visión de los procesos tal que hay un único identificador por cada proceso en el cluster de modo que cada nodo pueda conocer la locación y estado de cada uno. Es posible la migración de procesos

Sun Cluster

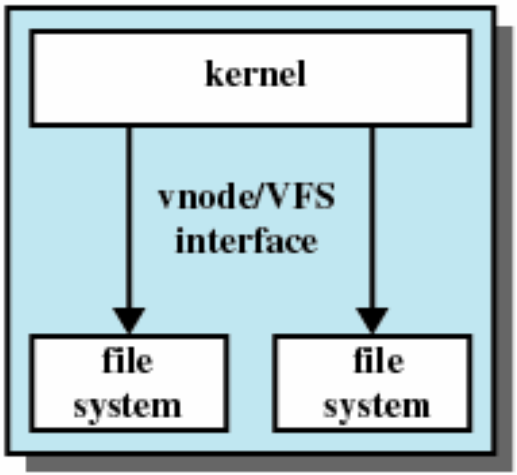
- Networking

Los diseñadores consideraron tres formas de diseño:

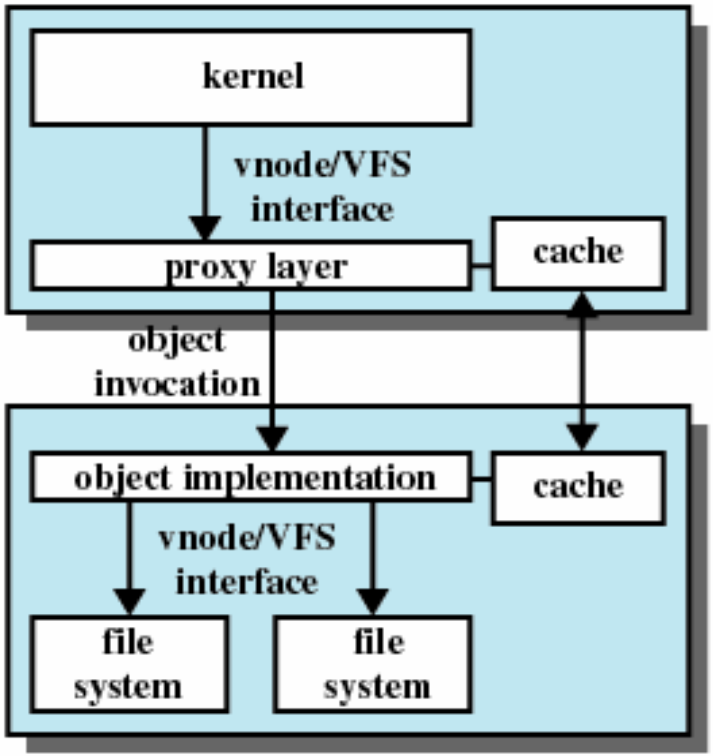
- Realiza todo el procesamiento del protocolo de red en un nodo único. Fue rechazado
- Asignan un dirección IP por nodo y corre el protocolo de red sobre la red externa directamente para cada nodo. Esta red no es transparente para el mundo exterior.
- Uso de filtro de paquetes para rutear los paquetes al nodo apropiado y realizar el protocolo de procesamiento en se nodo. Externamente, el cluster es visto como un simple servidor con una única dirección IP. Esto es lo adoptado por los diseñadores de Sun Cluster.

Sun Cluster

- Sistema de archivos distribuidos global.



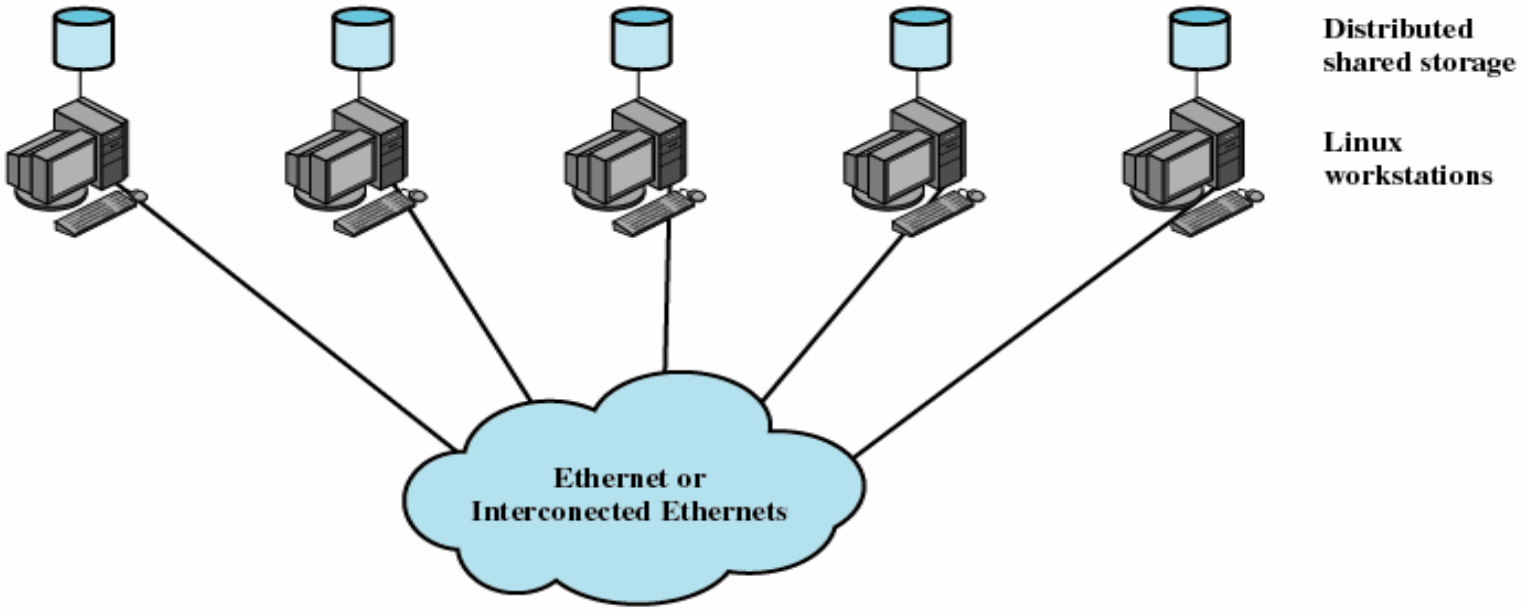
(a) Standard Solaris



(b) Sun Cluster

Beowulf y Clusters Linux

El proyecto Beowulf se inició soportado por el proyecto de la NASA “High Performance Computing and Communications” (HPCC). El objetivo fue investigar el potencial de PCs en cluster para realizar tareas computacionales importantes. Hoy en día es una de las más importantes tecnologías de cluster disponibles.



Beowulf y Clusters Linux

- Características claves
  - Componentes masivos en el mercado.
  - Procesadores dedicados (contra la técnica de robos de ciclos a las estaciones de trabajo ociosas).
  - Una red privada y dedicada (una combinación de LAN o WAN).
  - No hay componentes propietarias.
  - Replicación fácil de múltiples vendedores.
  - E/S escalables.
  - Software de base disponible libremente.
  - Uso de herramientas de computación distribuidas disponibles libremente con mínimos cambios.

Beowulf y Clusters Linux

El ambiente de software Beowulf está implementado como un agregado a las distribuciones disponibles comercialmente y libres de “royalties” de Linux.

La principal fuente *open-source* de Beowulf está disponible en el sitio de Beowulf (www.beowulf.org) pero hay muchas otras organizaciones que ofrecen libremente herramientas y utilidades.

Cada nodo en el cluster Beowulf corre su propia copia del kernel de Linux y pueden funcionar en forma autónoma.

Coming Next