Tema 2
Procesos y Hebras

1. 2.1 Generalidades sobre Procesos, Hilos y Planificación.
2. 2.2. Diseño e implementación de procesos e hilos en Linux.
3. 2.3 Planificación de la CPU en Linux.

Objetivos

- Conocer y diferenciar los conceptos de **proceso y hebra.**
- Conocer los **estados** básicos de un proceso/hebra y las posibles transiciones entre los estados.
- Saber en qué consiste un **cambio de contexto** y los costes que éste tiene para el sistema operativo.
- Contenido y utilidad **de las estructuras de datos** que el SO utiliza para la gestión de los procesos/hebras.
- Conocer y comparar las distintas implementaciones de las hebras.
- Conocer la utilidad de la **planificación de procesos** y de los distintos planificadores que pueden existir.
- Comparar los distintos **algoritmos de planificación** de CPU. Conocer las **operaciones** básicas sobre **procesos/hebras** que pueden realizar los usuarios.

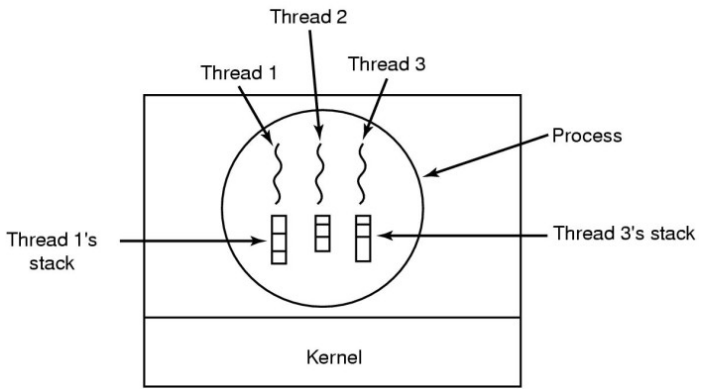
Bibliografía

- [Sta05] W. Stallings, Sistemas Operativos. Aspectos Internos y Principios de Diseño (5/e), Prentice Hall, 2005.
- [Car07] J. Carretero et al., Sistemas Operativos (2ª Edición), McGraw-Hill, 2007.
- [Lov10] R. Love, *Linux Kernel Development (3/e)*, Addison-Wesley Professional, 2010.
- [Mau08] W. Mauerer, Professional Linux Kernel Architecture, Wiley, 2008.

Ejecución del SO

Núcleo fuera de todo proceso:

- Ejecuta el núcleo del sistema operativo fuera de cualquier proceso.
- El código del sistema operativo se ejecuta como una entidad separada que opera en **modo privilegiado**.



Ejecución dentro de los procesos de usuario:

- Software del sistema operativo en el contexto de un proceso de usuario.
- Un proceso se ejecuta en modo privilegiado cuando se ejecuta el código del sistema operativo.

Bloque de control de procesos

Identificadores únicos del proceso, su padre y el usuario propietario.

Guarda el **contexto del CPU** (estado de ejecución) para poder detener y reanudar el proceso correctamente.

Contiene los **parámetros de planificación y control** que usa el sistema operativo para administrar el proceso.

Contiene datos locales y variables temporales del proceso mientras ejecuta en **modo usuario**

Segmento donde está cargado el programa ejecutable y sus datos. Ejemplo: código del editor de texto, archivos abiertos (texto.txt).

Usada cuando el proceso cambia a **modo núcleo** (por ejemplo, al hacer una llamada al sistema o interrupción).

Regiones de memoria que el proceso puede compartir con otros

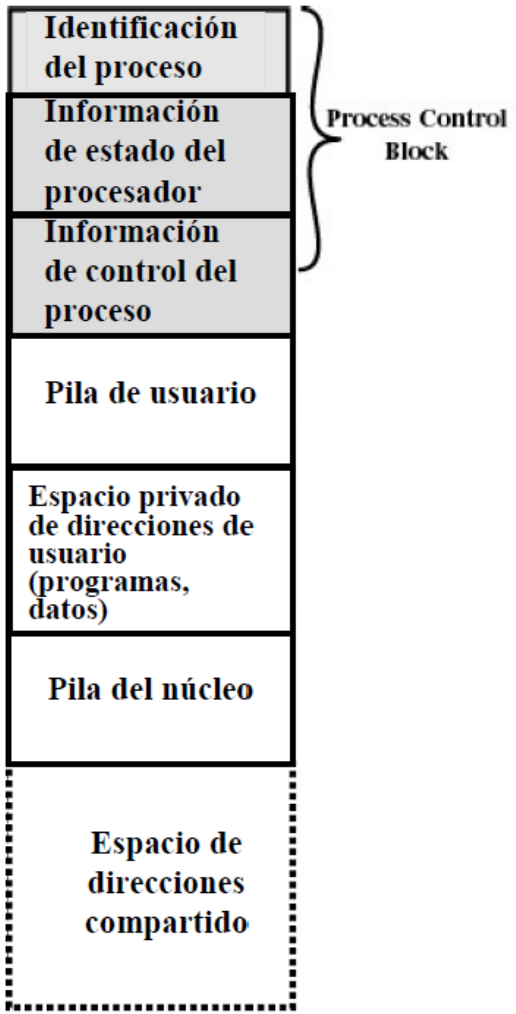
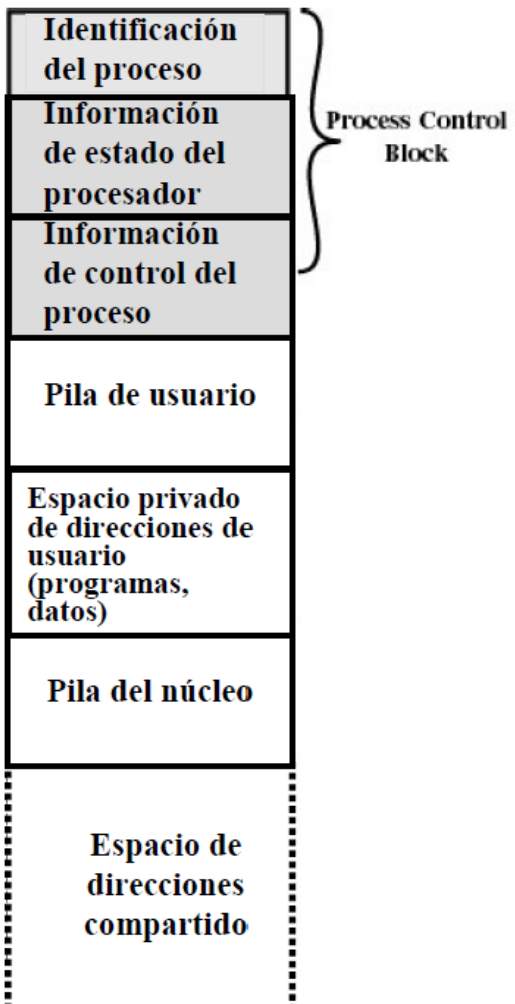


Figura 3.15. Imagen de un proceso: el sistema operativo se ejecuta dentro del proceso de usuario.

Bloque de control de procesos

Campo	Valor	Correspondencia según el gráfico
PID	1532	🔹 Identificación del proceso
PPID	145 (shell bash que lo lanzó)	🔹 Identificación del proceso
UID	1001 (usuario docente)	🔹 Identificación del proceso
Estado del proceso	*Running*	🔹 Información de control del proceso (estado actual)
Prioridad	15	🔹 Información de control del proceso (planificación)
Contador de programa (PC)	0x00407A23	🔹 Información de estado del procesador (dirección de la próxima instrucción)
Registros CPU	AX=120, BX=008, CX=450...	🔹 Información de estado del procesador (valores de registros del CPU)
Puntero de pila (SP)	0xFF12A0	🔹 Información de estado del procesador (posición actual de la pila)
Dirección base de memoria	0x00400000	🔹 Espacio privado de direcciones de usuario
Tamaño de segmento	2 MB	🔹 Espacio privado de direcciones de usuario (tamaño asignado al proceso)
Archivos abiertos	`/etc/diccionario.txt` , `/home/docente/texto.txt`	🔹 Información de E/S dentro del control del proceso
Tiempo de CPU usado	00:02:13	🔹 Información de control del proceso (estadísticas de ejecución)
Dispositivo de E/S asignado	Consola TTY1	🔹 Información de E/S (recursos asociados al proceso)
Cola de planificación	Lista de procesos interactivos	🔹 Información de control del proceso (ubicación en la cola del scheduler)
Tabla de páginas	0x8000F123	🔹 Información de memoria (traducción de direcciones virtuales a físicas)



Estructuras usadas por el SO

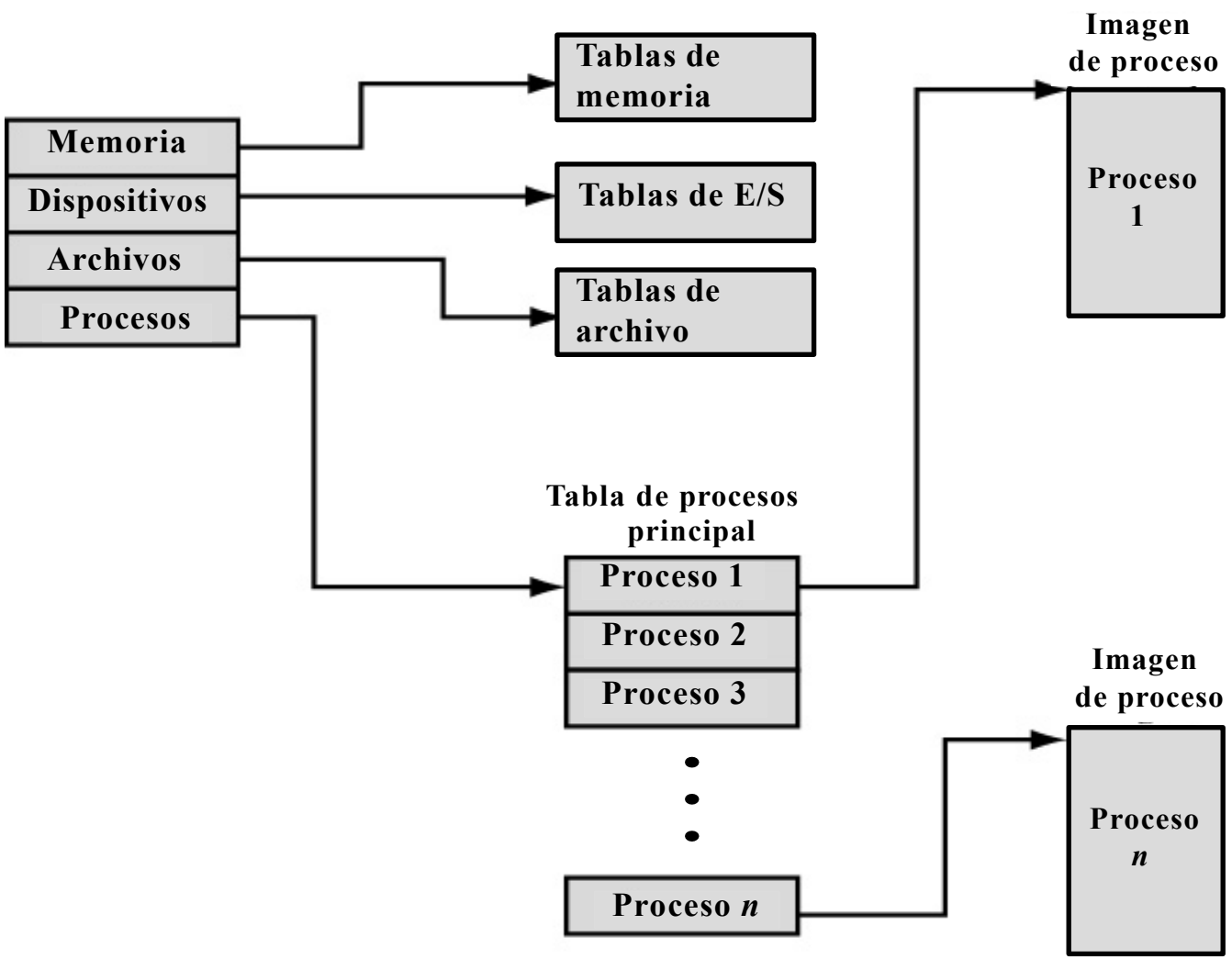


Figura 3.10. Estructura general de las tablas de control del sistema operativo.

Cambio de contexto

- Cuando un proceso esta ejecutándose, su PC (contador de programa), puntero a pila, registros, etc., están cargados en la CPU (es decir, los registros hardware contienen los valores actuales).
- Cuando el SO detiene un proceso ejecutándose, salva los valores actuales de estos registros (**contexto**) en el PCB de ese proceso.
- La acción de conmutar la CPU de un proceso a otro se denomina ***cambio de contexto***. Los sistemas de tiempo compartido realizan de 10 a 100 cambios de contexto por segundo. Este trabajo es sobrecarga.

Cambio de contexto (y II)

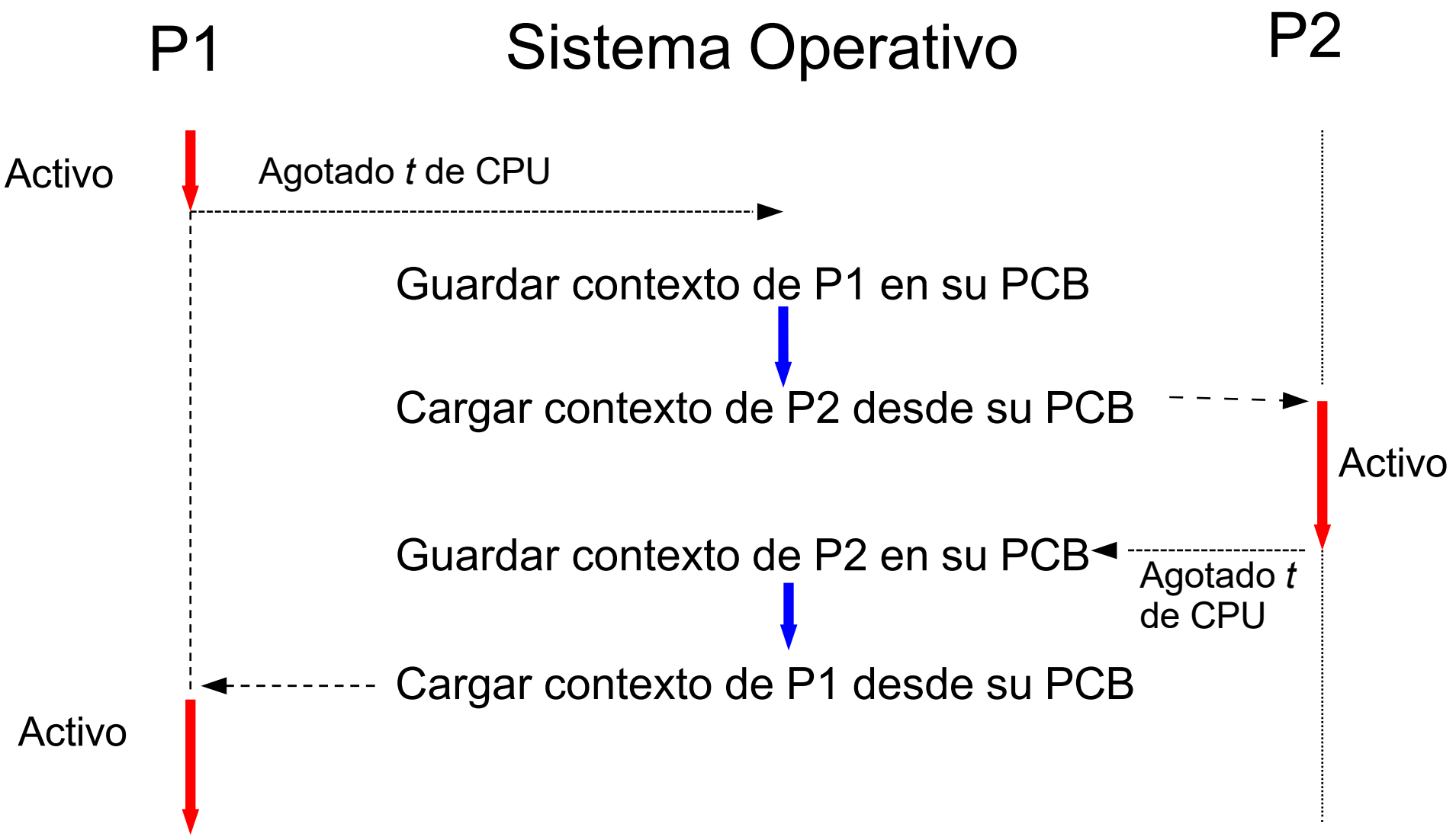
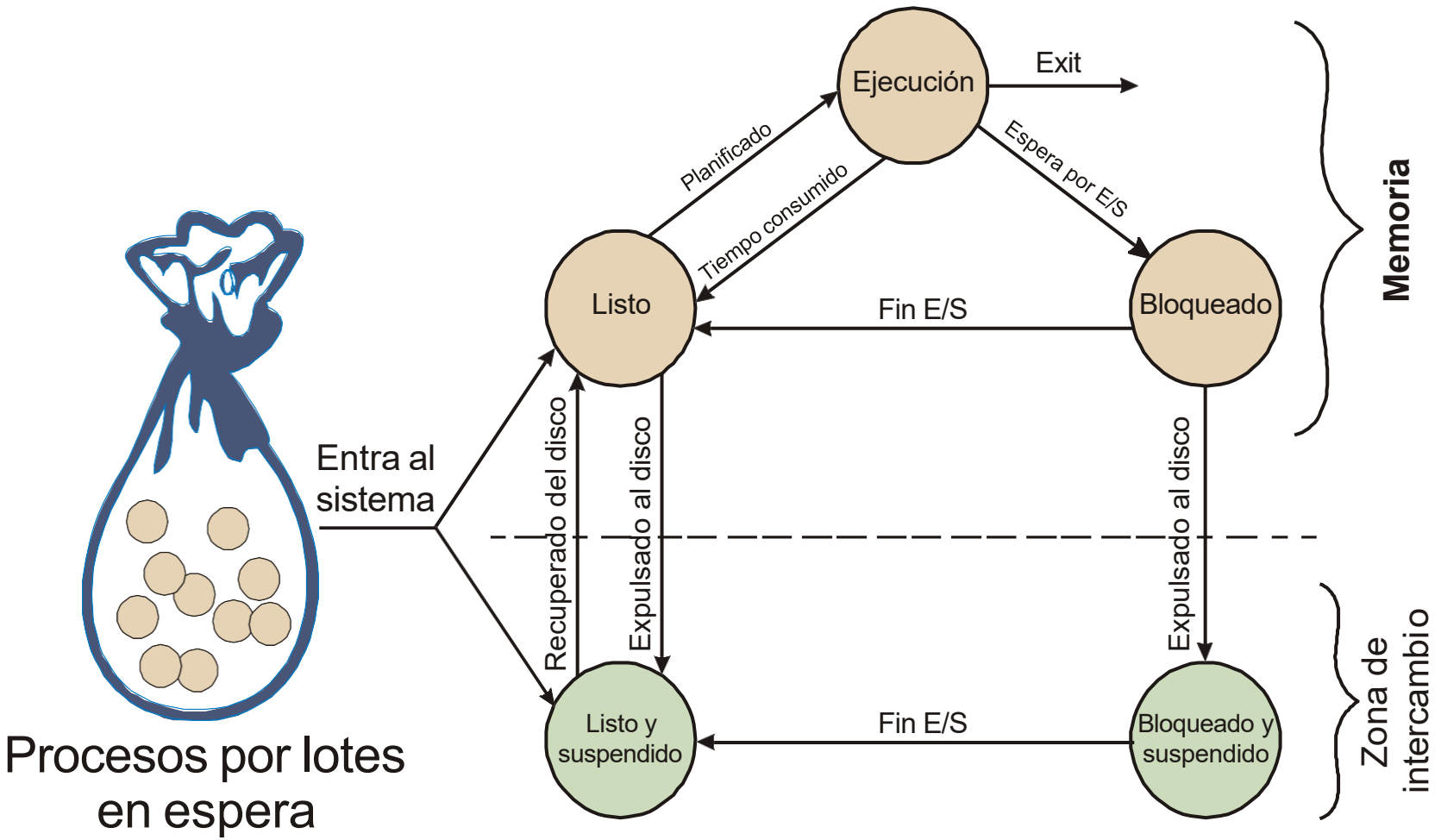


Diagrama de estados modificado



Operaciones sobre procesos
Creación de procesos

- ¿Qué significa crear un proceso?
  - » Asignarle el espacio de direcciones que utilizará.
  - » Crear las estructuras de datos para su administración.
- Cuando se crea, los sucesos comunes son:
  - » En sistemas por lotes en respuesta a la recepción y admisión de un trabajo.
  - » En sistemas interactivos cuando el usuario se conecta, el SO crea un proceso que ejecuta el intérprete de órdenes.
  - » El SO puede crear un proceso para llevar a cabo un servicio solicitado por un proceso de usuario.
  - » Un proceso puede crear otros procesos formando un árbol de procesos. Hablamos de relación padre-hijo (creador-creado).

Creación de procesos (y II)

- Cuando un proceso crea a otro, ¿cómo obtiene sus recursos el proceso hijo?
  - » Los obtiene directamente del SO: padre e hijo no comparten recursos.
  - » Comparte todos los recursos con el padre.
  - » Comparte un subconjunto de los recursos del padre.

Creación de procesos (y III)

- Ejecución:
  - » Padre e hijo se ejecutan concurrentemente.
  - » Padre espera al que el hijo termine.
- Espacio de direcciones:
  - » Hijo es un duplicado del padre (Unix, Linux).
  - » Hijo tiene un programa que lo carga (Virtual
- Memory System, W2K) Ejemplo: En UNIX
  - » La llamada al sistema **`fork`** (bifurcar) crea un nuevo proceso.
  - » La llamada **`exec`** después de `fork` reemplaza el espacio de direcciones con el programa del nuevo proceso.

Creación de procesos (y III)

```c
// fork_exec_subconjunto.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>

int main(void) {
    // Ejemplo: un descriptor de archivo heredado (subconjunto de recursos)
    int fd = open("salida_padre_hijo.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);
    if (fd < 0) { perror("open"); return 1; }

    pid_t pid = fork();
    if (pid < 0) { perror("fork"); return 1; }

    if (pid == 0) { // Hijo
        dprintf(fd, "Hijo: escribo antes del exec(). PID=%d\n", getpid());
        // Reemplaza imagen del proceso: tras exec, el espacio de direcciones cambia
        // pero el descriptor fd heredado sigue abierto (a menos que tenga FD_CLOEXEC).
        char *argv[] = {"/bin/echo", "Hijo tras exec(): hola desde echo", NULL};
        execv("/bin/echo", argv);
        perror("execv"); // solo si falla
        _exit(1);
    } else { // Padre
        dprintf(fd, "Padre: escribo mientras el hijo existe. PID_hijo=%d\n", pid);
        int status = 0;
        waitpid(pid, &status, 0);
        dprintf(fd, "Padre: hijo terminó con status=%d\n", status);
        close(fd);
        printf("Listo. Revisa 'salida_padre_hijo.txt'\n");
    }
    return 0;
}
```

- El **hijo** reemplaza **toda** su imagen de proceso por `/bin/echo`.
- **PID no cambia**, pero **memoria, código y pila** sí (todo es del nuevo programa).
- Los **descriptores** abiertos **permanecen abiertos** (a menos que tengan `FD_CLOEXEC`).
- **Nunca se vuelve** a la línea siguiente a `execv` (solo retornaría si falla).

```
Padre: escribo mientras el hijo existe. PID_hijo=12345
Hijo: escribo antes del exec(). PID=12345
Padre: hijo terminó con status=0
```

- `PID_hijo=12345` es el PID real del hijo (variará).
- `status=0` suele indicar que el hijo terminó normalmente (el `echo` salió con código 0).

Creación de procesos (y IV)

Por tanto, ¿qué pasos, en general, deben realizarse en una operación de creación?

- Nombrar al proceso: asignarle un PID único.
- Asignarle espacio (en MP y/o memoria secundaria).
- Crear el PCB e inicializarlo.
- Insertarlo en la Tabla de procesos y ligarlo a la cola de planificación correspondiente.
- Determinar su prioridad inicial.

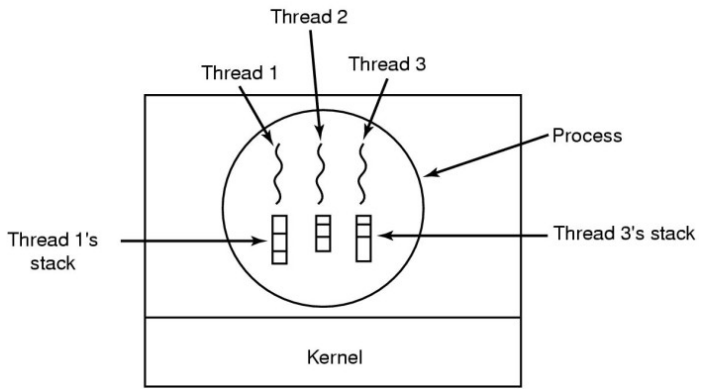
Terminación de procesos

¿Qué sucesos determinan la finalización de un proceso?

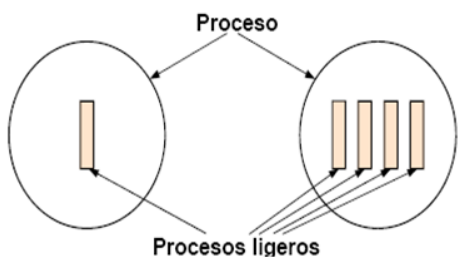
- Cuando un proceso ejecuta la última instrucción, solicita al SO su finalización (**exit**)
  - Envío de datos del hijo al padre
  - Recursos del proceso son liberados por el SO
- El padre puede finalizar la ejecución de sus hijos (**abort o kill**)
  - El hijo ha sobrepasado los recursos asignados
  - La tarea asignada al hijo ya no es necesaria
  - El padre va a finalizar: el SO no permite al hijo continuar (terminación en cascada)
- El SO puede terminar la ejecución de un proceso porque se hayan producido errores o condiciones de fallo

Hebras
(threads, hilos o procesos ligeros)

- Una *hebra* (o *proceso ligero*) es la unidad básica de utilización de la CPU. Consta de:
  - » Contador de programa.
  - » Conjunto de registros.
  - » Espacio de pila.
  - » Estado

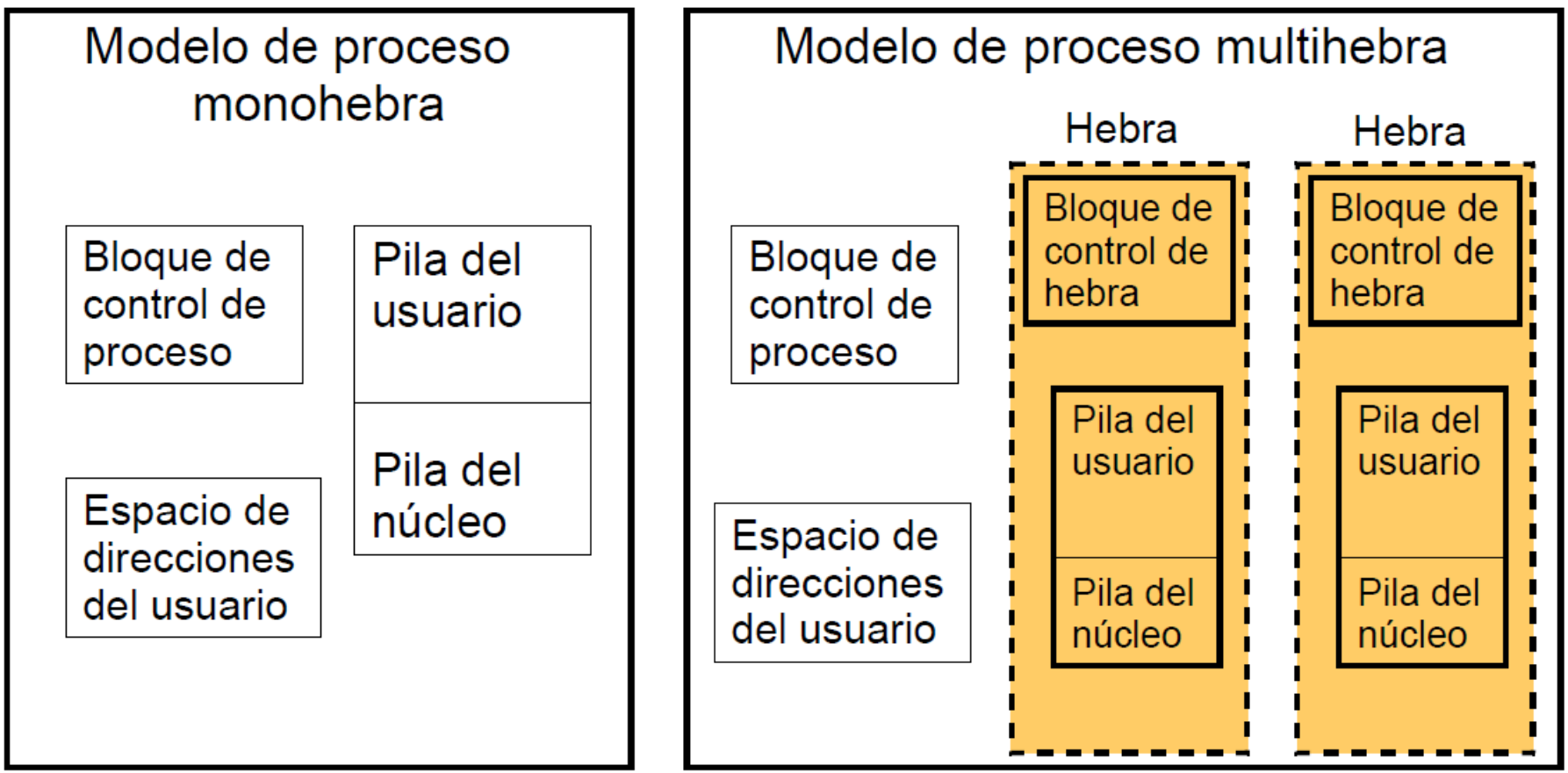


- Una hebra comparte con sus hebras pares una tarea que consiste en:
  - » Sección de código.
  - » Sección de datos.
  - » Recursos del SO (archivos abiertos, señales,..).



- Un *proceso pesado* o tradicional es igual a una tarea con una hebra.

Hebras (y II)


Modelo de proceso monohebra
Bloque de control de proceso
Pila del usuario
Pila del núcleo
Espacio de direcciones del usuario
Modelo de proceso multihebra
Hebra
Hebra
Bloque de control de proceso
Bloque de control de hebra
Bloque de control de hebra
Pila del usuario
Pila del usuario
Espacio de direcciones del usuario
Pila del núcleo
Pila del núcleo

Ventajas de las hebras

Se obtiene un mayor rendimiento y un mejor servicio debido a :

- Se reduce el tiempo de cambio de contexto, el tiempo de creación y el tiempo de terminación.
- En una tarea con múltiples hebras, mientras una hebra esta bloqueada y esperando, una segunda hebra de la misma tarea puede ejecutarse (depende del tipo de hebras).
- La comunicación entre hebras de una misma tarea se realiza a través de la memoria compartida (no necesitan utilizar los mecanismos del núcleo).
- Las aplicaciones que necesitan compartir memoria se benefician de las hebras.

Funcionalidad de las hebras

Al igual que los procesos las hebras poseen un estado de ejecución y pueden sincronizarse.

- **Estados de las hebras**: Ejecución, Lista o Preparada y Bloqueada.
- Operaciones básicas relacionadas con el **cambio de estado** en hebras:
  - Creación
  - Bloqueo
  - Desbloqueo
  - Terminación
- **Sincronización entre hebras (**mutex, semáforos, entre otros**)**.

Tipos de hebras

Tipos de hebras: *Núcleo (Kernel), de usuario y enfoques híbridos*

Hebras de usuario

- Todo el trabajo de gestión de hebras lo realiza la aplicación, el núcleo no es consciente de la existencia de hebras.
- Se implementan a través de una biblioteca en el nivel usuario. La biblioteca contiene código para gestionar las hebras:
  - crear hebras, intercambiar datos entre hebras, planificar la ejecución de las hebras y salvar y restaurar el contexto de las hebras.
- La unidad de planificación para el núcleo es el proceso.

Tipos de hebras (y II)

Hebras Kernel (Núcleo)

- Toda la gestión de hebras lo realiza el núcleo.
- El SO proporciona un conjunto de llamadas al sistema similares a las existentes para los procesos (Mach, OS/2).
- El núcleo mantiene la información de contexto del proceso como un todo y de cada hebra.
- La unidad de planificación es la hebra.
- Las propias funciones del núcleo pueden ser multihebras.

Tipos de hebras (y III)

- Ventajas del uso de las hebras tipo usuario frente a las de tipo núcleo:
  - » Se evita la sobre carga de cambios de modo, que sucede cada vez que se pasa el control de una hebra a otra en sistemas que utilizan hebras núcleo.
  - » Se puede tener una planificación para las hebras distinta a la planificación subyacente del SO.
  - » Se pueden ejecutar en cualquier SO. Su utilización no supone cambio en el núcleo.

Tipos de hebras (y IV)

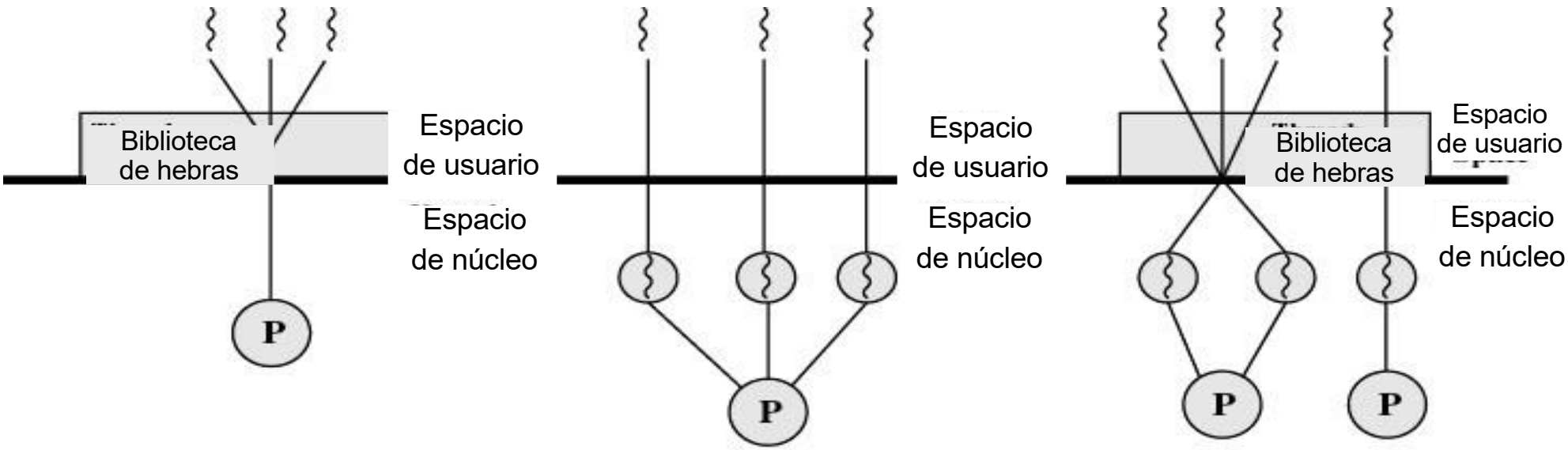
- Desventajas del uso de las hebras tipo usuario frente a las de tipo núcleo.
  - » Cuando un proceso realiza una llamada al sistema bloqueado no sólo se bloquea la hebra que realizó la llamada, sino todas las hebras del proceso.
  - » En un entorno multiprocesador, una aplicación multihebra no puede aprovechar las ventajas de dicho entorno ya que el núcleo asigna un procesador a un proceso.

Tipos de hebras (y V)

Enfoques híbridos

- Implementan tanto hebras a nivel *kernel* como usuario (p. ej. Solaris 2, WindowsNT/XP).
- La creación de hebras, y la mayor parte de la planificación y sincronización se realizan en el espacio de usuario.
- Las distintas hebras de una aplicación se asocian con varias hebras del núcleo (mismo o menor número), el programador puede ajustar la asociación para obtener un mejor resultado.
- Las múltiples hebras de una aplicación se pueden paralelizar y las llamadas al sistema bloqueadoras no necesitan bloquear todo el proceso.

Tipos de hebras (y VI)



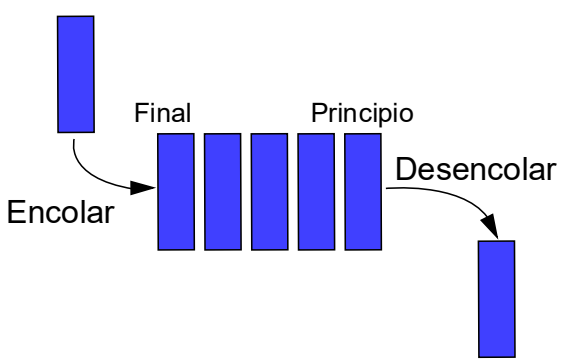
(a) Nivel de usuario puro **(b) Nivel de núcleo puro** **(c) Sistema Híbrido**

Hebra a nivel de usuario Hebra a nivel de núcleo P Proceso

Hebras a nivel de usuario y a nivel de núcleo (*Stalling*)

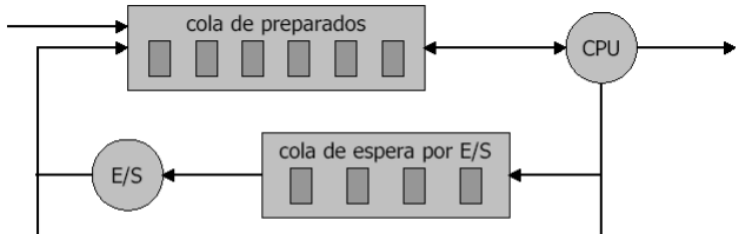
Planificación: PCB's y Colas de Estados

- El SO mantiene una colección de colas que representan el estado de todos los procesos en el sistema.
- Típicamente hay una cola por estado.
- Cada PCB está encolado en una cola de estado acorde a su estado actual.
- Conforme un proceso cambia de estado, su PCB es retirado de una cola y encolado en otra.



Colas de estados

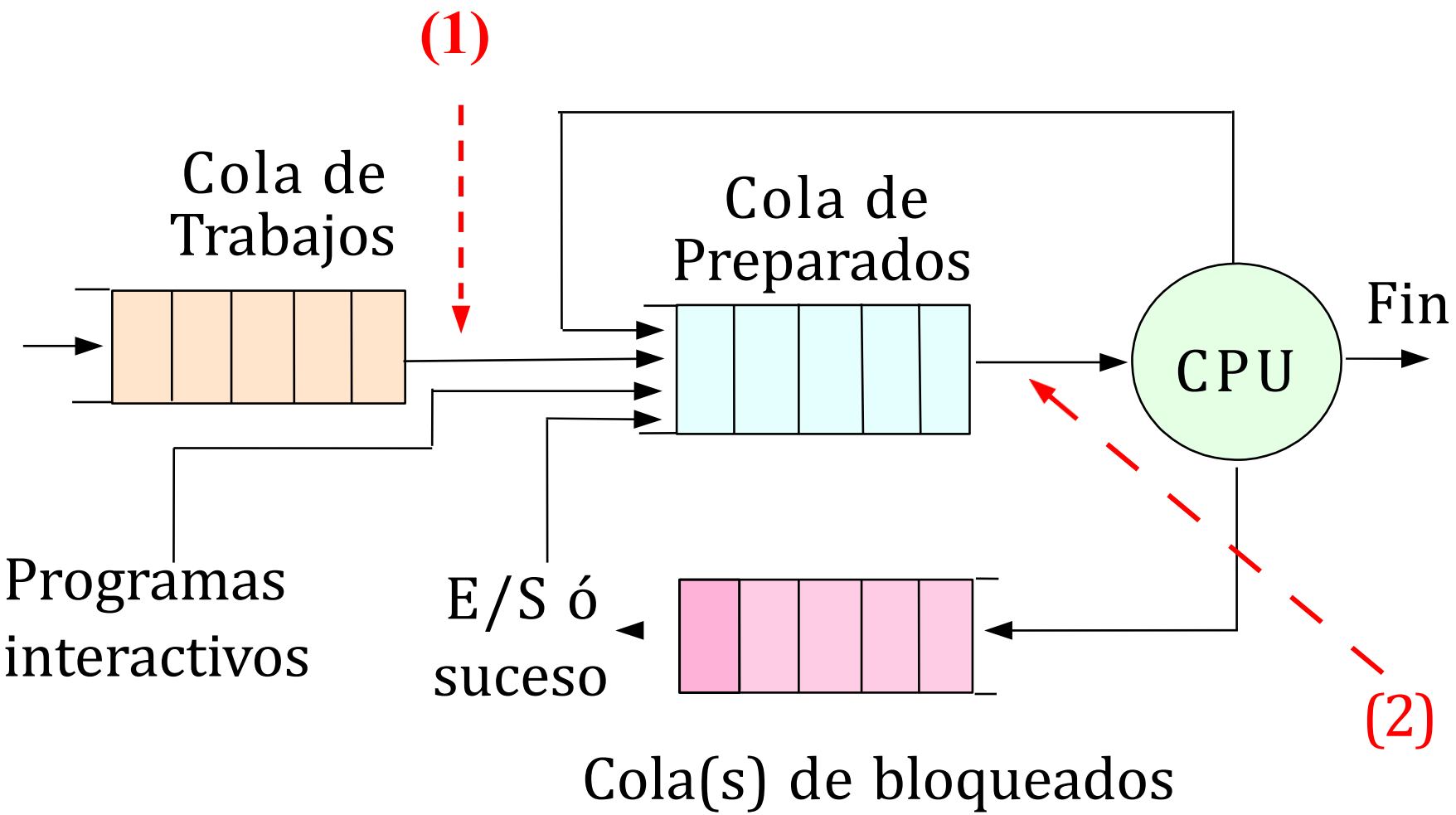
- *Cola de trabajos* -- conjunto de los trabajos pendientes de ser admitidos en el sistema (trabajos por lotes que no están en memoria).
- *Cola de preparados* -- conjunto de todos los procesos que residen en memoria principal, preparados y esperando para ejecutarse.
- *Cola(s) de bloqueados* -- conjunto de todos los procesos esperando por un dispositivo de E/S particular o por un suceso.



Tipos de planificadores

- Planificador. Parte del SO que controla la utilización de un recurso.
- Tipos de planificadores de la CPU:
  - » *Planificador a largo plazo* (planificador de trabajos): selecciona los procesos que deben llevarse a la cola de preparados; (1) en figura siguiente.
  - » *Planificador a corto plazo* (planificador de la CPU): selecciona al proceso que debe ejecutarse a continuación, y le asigna la CPU; (2) en figura siguiente.
  - » *Planificador a medio plazo* (más adelante).

Migración entre colas



Características de los planificadores

- *El planificador a corto plazo* :

  »trabaja con la cola de preparados.

  »se invoca muy frecuentemente (milisegundos) por lo que debe ser rápido.

El planificador a largo plazo

»Permite controlar el *grado de multiprogramación*.

»Se invoca poco frecuentemente (segundos o minutos), por lo que puede ser más lento.

Clasificación de procesos

- *Procesos limitados por E/S o procesos cortos* -- dedican más tiempo a realizar E/S que cómputo; muchas ráfagas de CPU cortas y largos períodos de espera.

- *Procesos limitados por CPU o procesos largos* -- dedican más tiempo en computación que en realizar E/S; pocas ráfagas de CPU pero largas.

Mezcla de trabajos

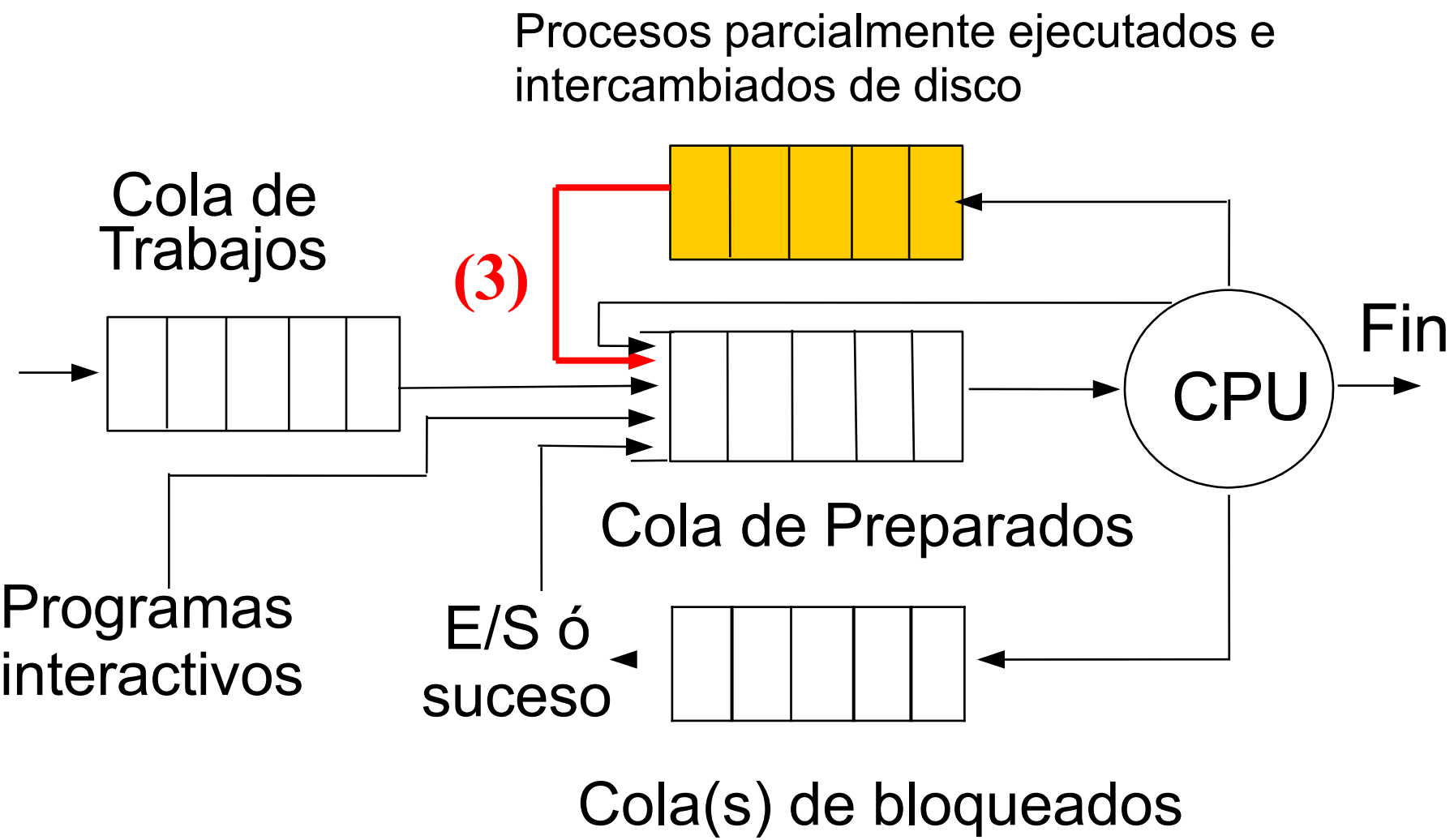
Es importante que el planificador a largo plazo seleccione una buena mezcla de trabajos, ya que:

- Si todos los trabajos están limitados por E/S, la cola de preparados estará casi siempre vacía y el planificador a corto plazo tendrá poco que hacer.
- Si todos los procesos están limitados por CPU, la cola de E/S estará casi siempre vacía y el sistema estará de nuevo desequilibrado.

Planificador a medio plazo

- En algunos SO's, p. ej. SO de tiempo compartido, a veces es necesario sacar procesos de la memoria (reducir *el grado de multiprogramación*), bien para mejorar la mezcla de procesos, bien por cambio en los requisitos de memoria, y luego volverlos a introducir (intercambio o *swapping*).
- El *planificador a medio plazo* se encarga de devolver los procesos a memoria. Transición (3) en la siguiente figura.

Planificador a medio plazo (y II)



Despachador

- El despachador (*dispatcher*) es el módulo del SO que da el control de la CPU al proceso seleccionado por el planificador a corto plazo; esto involucra:
  - » Cambio de contexto (se realiza en modo kernel).
  - » Conmutación a modo usuario.
  - » Salto a la posición de memoria adecuada del programa para su reanudación.
- *Latencia de despacho* -- tiempo que emplea el despachador en detener un proceso y comenzar a ejecutar otro.

Activación del Despachador

El despachador puede actuar cuando:

1. Un proceso no quiere seguir ejecutándose (finaliza o ejecuta una operación que lo bloquea)
2. Un elemento del SO determina que el proceso no puede seguir ejecutándose (ej. E/S o retiro de memoria principal)
3. El proceso agota el quantum de tiempo asignado
4. Un suceso cambia el estado de un proceso de bloqueado a ejecutable

Políticas de planificación: Monoprocesadores

- Objetivos:
  - » Buen rendimiento (productividad)
  - » Buen servicio
- Para saber si un proceso obtiene un buen servicio, definiremos un conjunto de medidas: dado un proceso *P*, que necesita un tiempo de servicio (ráfaga) *t*
  - *Tiempo de respuesta (T)*-- tiempo transcurrido desde que se remite una solicitud (entra en la cola de preparados) hasta que se produce la primera respuesta (no se considera el tiempo que tarda en dar la salida)
  - *Tiempo de espera (M)*-- tiempo que un proceso ha estado esperando en la cola de preparados: T - t
  - *Penalización (P)* -- T / t
  - *Índice de respuesta (R)* -- t / T : fracción de tiempo que P está recibiendo servicio.

Políticas de planificación: Monoprocesadores

- **T** = tiempo de respuesta total
- **t** = tiempo de servicio o ráfaga del CPU
- **M** = tiempo de espera
- **P** = penalización
- **R** = índice de respuesta

Supongamos el **proceso P_1** que:

- Entra al sistema en el tiempo **0 s**.
- Empieza a ejecutarse en el tiempo **4 s** (debido a otros procesos en cola).
- Termina completamente en el tiempo **9 s**.
- Tiene un **tiempo de servicio (ráfaga)** `t = 5 s`.

Métrica	Símbolo	Fórmula	Resultado	Interpretación
Tiempo de respuesta	T	9 – 0	9 s	Tiempo total en el sistema
Tiempo de espera	M	T – t	4 s	Tiempo en cola sin CPU
Penalización	P	T / t	1.8	Cuánto más tardó en total
Índice de respuesta	R	t / T	0.56	% de tiempo que estuvo en CPU

🔹 1 Tiempo de respuesta (T)

Tiempo total desde que entra el proceso hasta que termina su ejecución.

$$T = \text{tiempo de finalización} - \text{tiempo de llegada}$$

$$T = 9 - 0 = 9 \text{ s}$$

🔹 2 Tiempo de espera (M)

Tiempo que el proceso permaneció en la cola **sin ejecutar**.

$$M = T - t = 9 - 5 = 4 \text{ s}$$

🔹 3 Penalización (P)

Mide **cuánto más tiempo** tardó el proceso en completarse, **comparado con el tiempo que realmente necesitaba de CPU**

$$P = \frac{T}{t} = \frac{9}{5} = 1.8$$

🟢 **Interpretación:**

El proceso pasó 1.8 veces su tiempo de ejecución total dentro del sistema (incluyendo espera).

🔹 4 Índice de respuesta (R)

Fracción del tiempo total en que el proceso estuvo recibiendo servicio.

$$R = \frac{t}{T} = \frac{5}{9} \approx 0.56$$

🟢 **Interpretación:**

El proceso recibió servicio (usó CPU) el **56 % del tiempo** que estuvo en el sistema.

Políticas de planificación: Monoprocesadores (y II)

Otras medidas interesantes son:

- *Tiempo del núcleo* -- tiempo perdido por el SO tomando decisiones que afectan a la planificación de procesos y haciendo los cambios de contexto necesarios. En un sistema eficiente debe representar entre el 10% y el 30% del total del tiempo del procesador.
- *Tiempo de inactividad* -- tiempo en el que la cola de ejecutables está vacía y no se realiza ningún trabajo productivo.
- *Tiempo de retorno* – cantidad de tiempo necesario para ejecutar un proceso completo.

Políticas de planificación: Monoprocesadores (III)

- Las políticas de planificación se comportan de distinta manera dependiendo de la clase de procesos
  - » Ninguna política de planificación es completamente satisfactoria, cualquier mejora en una clase de procesos es a expensas de perder eficiencia en los procesos de otra clase.
- Podemos clasificarlas en:
  - » **No apropiativas (no expulsivas)**: una vez que se le asigna el procesador a un proceso, no se le puede retirar hasta que éste voluntariamente lo deje (finalice o se bloquee).
  - » **Apropiativas (expulsivas)**: al contrario, el SO puede apropiarse del procesador cuando lo decida.

Políticas de planificación de la CPU (y IV)

- FCFS
- El más corto primero:
  - no apropiativo
  - apropiativo
- Planificación por prioridades
- Por turnos (*Round-Robin*)
- Colas múltiples
- Colas múltiples con realimentación.

FCFS (First Come First Served)

- Los procesos son servidos según el orden de llegada a la cola de ejecutables.
- Es *no apropiativo*, cada proceso se ejecutará hasta que finalice o se bloquee.
- Fácil de implementar pero pobre en cuanto a prestaciones.
- Todos los procesos pierden la misma cantidad de tiempo esperando en la cola de ejecutables independientemente de sus necesidades.
- Procesos cortos muy penalizados.
- Procesos largos poco penalizados.

El más corto primero (SJF)

- Es *no apropiativo.*
- Cuando el procesador queda libre, selecciona el proceso que requiera un tiempo de servicio menor.
- Si existen dos o más procesos en igualdad de condiciones, se sigue FCFS.
- Necesita conocer explícitamente el tiempo estimado de ejecución (tº servicio) ¿Cómo?.
- Disminuye el tiempo de respuesta para los procesos cortos y discrimina a los largos.
- Tiempo medio de espera bajo.

El más corto primero apropiativo (SRTF)

- Cada vez que entra un proceso a la cola de ejecutables se comprueba si su tiempo de servicio es menor que el tiempo de servicio que le queda al proceso que está ejecutándose. Casos:
  - » **Si es menor**: se realiza un cambio de contexto y el proceso con menor tiempo de servicio es el que se ejecuta.
  - » **No es menor**: continúa el proceso que estaba ejecutándose.
- El tiempo de respuesta es menor excepto para procesos muy largos.
- Se obtiene la menor penalización en promedio (mantiene la cola de ejecutables con la mínima

Planificación por prioridades

- Asociamos a cada proceso un número de prioridad (entero).
- Se asigna la CPU al proceso con mayor prioridad (enteros menores = mayor prioridad)
  - Apropiativa
  - No apropiativa
- **Problema**: Inanición -- los procesos de baja prioridad pueden no ejecutarse nunca.
- **Solución**: Envejecimiento -- con el paso del tiempo se incrementa la prioridad de los procesos.

Por Turnos (Round-Robin)

- La CPU se asigna a los procesos en intervalos de tiempo (quatum).
- Procedimiento:
  - » Si el proceso finaliza o se bloquea antes de agotar el quantum, libera la CPU. Se toma el siguiente proceso de la cola de ejecutables (la cola es FIFO) y se le asigna un quantum completo.
  - » Si el proceso no termina durante ese quantum, se interrumpe y se coloca al final de la cola de ejecutables.
- Es apropiativo.

***Nota**: En los ejemplos* supondremos que si un proceso *A* llega a la cola de ejecutables al mismo tiempo que otro *B* agota su quantum, la llegada de *A* a la cola de ejecutables ocurre antes de que *B* se incorpore a ella.

Por Turnos (y III)

- Los valores típicos del quantum están entre 1/60sg y 1sg.
- Penaliza a todos los procesos en la misma cantidad, sin importar si son cortos o largos.
- Las ráfagas muy cortas están más penalizadas de lo deseable.
- ¿valor del quantum?
  - muy grande (excede del tº de servicio de todos los procesos) □ se convierte en FCFS
  - muy pequeño □ el sistema monopoliza la CPU haciendo cambios de contexto (tº del núcleo muy alto)

Colas múltiples

- La cola de preparados se divide en varias colas y cada proceso es asignado permanentemente a una cola concreta P. ej. interactivos y batch
- Cada cola puede tener su propio algoritmo de planificación P. ej. interactivos con RR y batch con FCFS
- Requiere una planificación entre colas
  - » Planificación con prioridades fijas. P. ej. primero servimos a los interactivos luego a los batch
  - » Tiempo compartido -- cada cola obtiene cierto tiempo de CPU que debe repartir entre sus procesos. P. ej. 80% interactivos en RR y 20% a los batch con FCFS

Colas múltiples con realimentación

- Un proceso se puede mover entre varias colas
- Requiere definir los siguientes parámetros:
  - » Número de colas
  - » Algoritmo de planificación para cada cola
  - » Método utilizado para determinar cuando trasladar a un proceso a otra cola
  - » Método utilizado para determinar en qué cola se introducirá un proceso cuando necesite un servicio
  - » Algoritmo de planificación entre colas
- Mide en tiempo de ejecución el comportamiento real de los procesos
- Disciplina de planificación más general (Unix, Linux, Windows NT)

Ejercicios sobre políticas de planificación

- FCFS (First-Come, First-Served)

Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})
P1	0	18
P2	3	5
P3	7	11
P4	10	4
P5	15	7

- Construye el Diagrama de Gantt.
- Calcula el Tiempo de Finalización para cada proceso.
- Calcula el Tiempo de Retorno Promedio

Ejercicios sobre políticas de planificación

- El más Corto Primero: No Apropiativo (SJF)

Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})
P1	0	9
P2	2	4
P3	5	7
P4	8	5
P5	12	2

  - Construye el Diagrama de Gantt.

Ejercicios sobre políticas de planificación

- El más Corto Primero: Apropiativo (SJF)

Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})
P1	0	10
P2	1	5
P3	3	2
P4	6	8
P5	7	4

  - Construye el Diagrama de Gantt, mostrando todas las apropiaciones.
  - Calcula el Tiempo de Retorno Promedio y el Tiempo de Respuesta Promedio.

Ejercicios sobre políticas de planificación

- Planificación por Prioridades (No Apropiativa)

Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})	Prioridad
P1	0	14	3 (Media)
P2	2	5	1 (Alta)
P3	4	7	4 (Baja)
P4	9	3	2 (Media-Alta)
P5	11	6	1 (Alta)

- Construye el Diagrama de Gantt, mostrando todas las apropiaciones.
- Calcula el Tiempo de Retorno Promedio y el Tiempo de Respuesta Promedio.

Ejercicios sobre políticas de planificación

- Round-Robin (RR)

Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})
P1	0	11
P2	1	5
P3	2	9
P4	3	4
P5	4	7

Algoritmo: Round-Robin con **quantum (q) = 3**.

- Construye el Diagrama de Gantt, mostrando todas las apropiaciones.
- Calcula el Tiempo de Retorno Promedio.

Ejercicios sobre políticas de planificación

- Planificación por Prioridades (No apropiativo)

Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})	Prioridad
P1	0	14	3 (Media)
P2	2	5	1 (Alta)
P3	4	7	4 (Baja)
P4	9	3	2 (Media-Alta)
P5	11	6	1 (Alta)

- Construye el Diagrama de Gantt, mostrando todas las apropiaciones.
- Calcula el Tiempo de Retorno Promedio.

Ejercicios sobre políticas de planificación

- Planificación por Prioridades (Apropiativo)

Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})	Prioridad
P1	0	8	4 (Baja)
P2	2	10	1 (Alta)
P3	5	4	3 (Media)
P4	8	5	2 (Media-Alta)
P5	11	2	1 (Alta)

  - Construye el Diagrama de Gantt, mostrando todas las apropiaciones.
  - Calcula el Tiempo de Retorno Promedio.

Ejercicios sobre políticas de planificación

- Colas Múltiples con Realimentación (MLFQ)

Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})
P1	0	16
P2	1	6
P3	3	5
P4	5	4
P5	7	8

- Construye el Diagrama de Gantt, mostrando todas las apropiaciones.
- Calcula el Tiempo de Respuesta Promedio.

Ejercicios sobre políticas de planificación

- Colas Múltiples (MQ)

Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})	Tipo de Proceso
P1	0	10	Interactivo (Q1)
P2	2	15	Lote (Q2)
P3	5	5	Interactivo (Q1)
P4	8	12	Lote (Q2)
P5	10	3	Interactivo (Q1)

Políticas de las Colas:

•**Cola 1 (Interactiva):** Se ejecuta con **Round Robin (RR)**, q=4.

•**Cola 2 (Lote):** Se ejecuta con **FCFS**.

Planificador Global: Las Colas tienen prioridad estática: **Q1 tiene mayor prioridad que Q2**. Q1 solo cede la CPU si está vacía

- Construye el Diagrama de Gantt, mostrando todas las apropiaciones.
- Calcula el Tiempo de Retorno Promedio para todos los procesos.

Ejercicios sobre políticas de planificación

- Colas Múltiples con Realimentación (MLFQ)

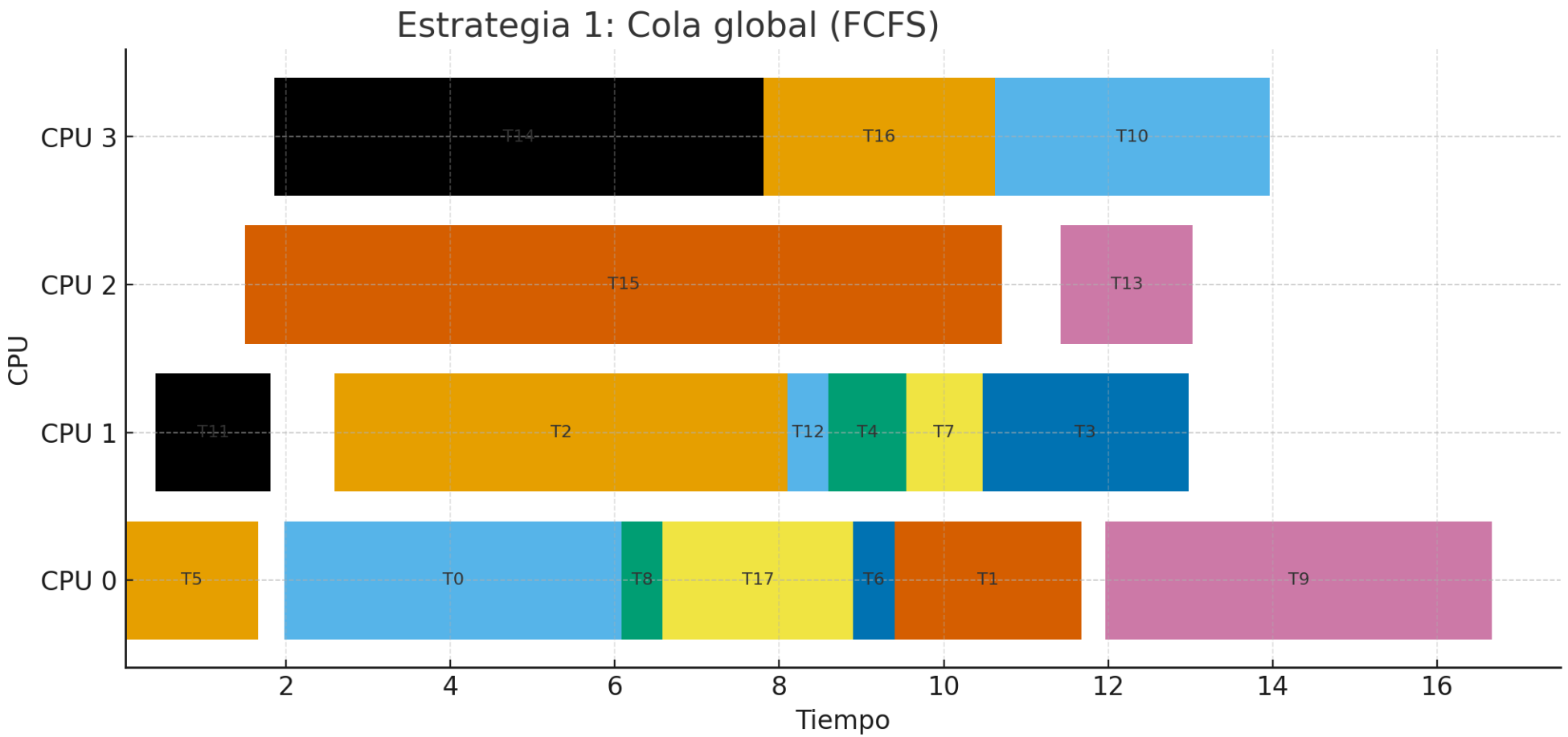
Proceso	Tiempo de Llegada (T_L)	Ráfaga de CPU (R_{CPU})
P1	0	16
P2	1	6
P3	3	5
P4	5	4
P5	7	8

Políticas de las Colas:
•**Cola 1 (Q1 - Alta Prioridad):** Round Robin con q=2.
•**Cola 2 (Q2 - Media Prioridad):** Round Robin con q=4.
•**Cola 3 (Q3 - Baja Prioridad):** FCFS.
Reglas de Realimentación:
1.Todos los procesos entran inicialmente en **Q1**.
2.Si un proceso en Q1 consume todo su q (2 unidades), pasa a **Q2**.
3.Si un proceso en Q2 consume todo su q (4 unidades), pasa a **Q3**.
4.Si un proceso termina antes de agotar su *quantum* (en Q1 o Q2), sale del sistema.
5.La **prioridad es estática entre colas (Q1 > Q2 > Q3)**: la cola de mayor prioridad se ejecuta primero, y solo cede la CPU cuando está vacía.

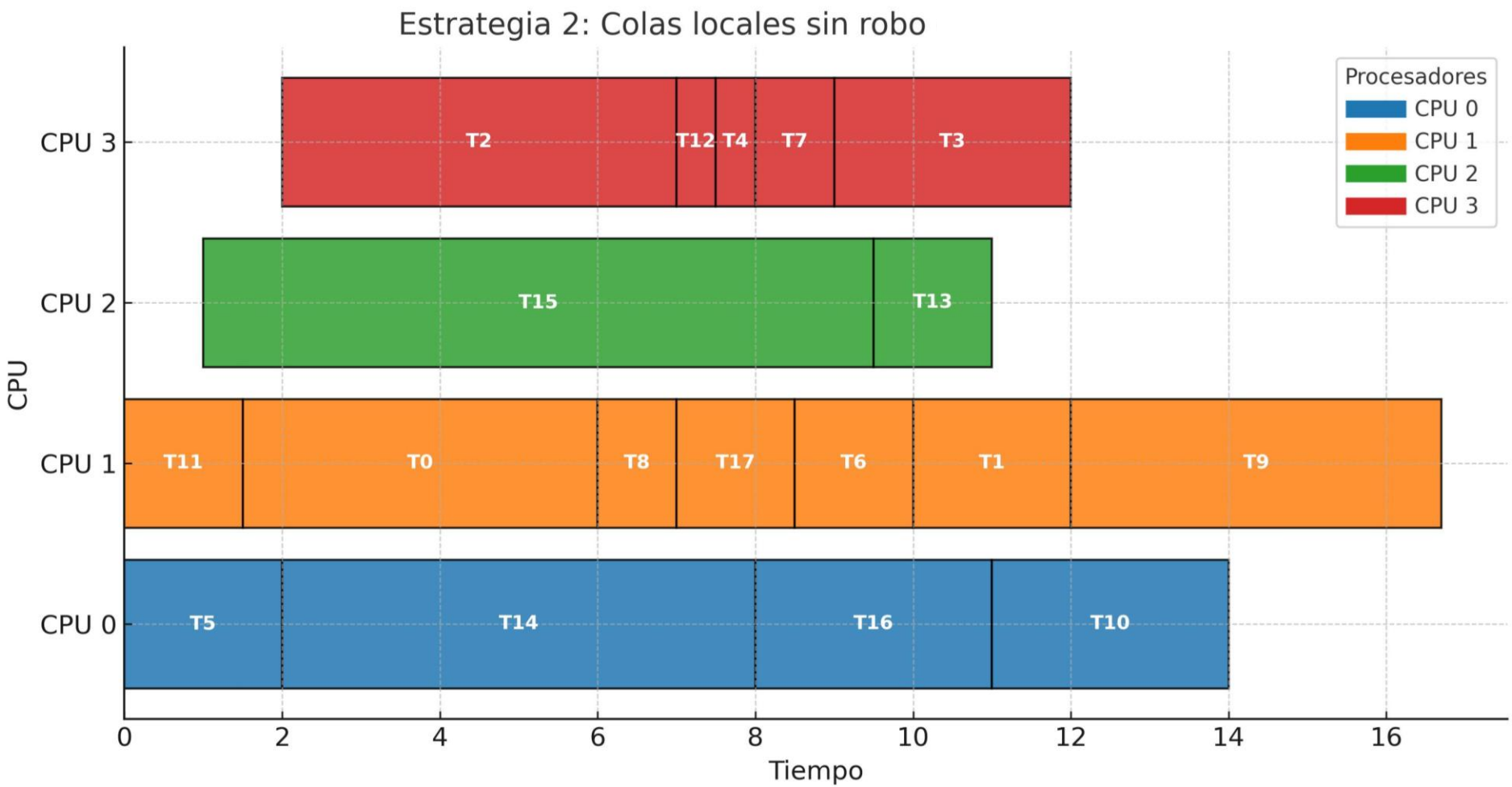
Planificación en multiprocesadores

- Tres aspectos interrelacionados:
  - Asignación de procesos a procesadores
    - Planificación por procesador. Cola dedicada para cada procesador
    - Planificación global. Cola global para todos los procesadores
    - Planificación híbrida. Cada CPU tiene su cola local. Cuando hay desequilibrio, se realiza migración o "stealing" de procesos desde colas más cargadas hacia las menos cargadas.
  - Uso de multiprogramación en cada procesador individual
  - Activación del proceso. Momento en que el sistema operativo asigna un procesador físico (CPU) a un proceso listo para ejecutarse.

Planificación en multiprocesadores (y II)



Planificación en multiprocesadores (y II)



Planificación en multiprocesadores (y II)

- Planificación de procesos

  → igual que en monoprocesadores;
  pero teniendo en cuenta:

  - Número de CPUs
  - Asignación/Liberación proceso-procesador

- Planificación de hilos

  → permiten explotar el paralelismo real dentro de una aplicación

Planificación de hilos en multiprocesadores

1) Compartición de carga

- Cola global de hilos preparados
- Cuando un procesador está ocioso, se selecciona un hilo de la cola (método muy usado)

2) Planificación en pandilla

- Se planifica un conjunto de hilos afines (de un mismo proceso) para ejecutarse sobre un conjunto de procesadores al mismo tiempo (relación 1 a 1)
- Útil para aplicaciones cuyo rendimiento se degrada mucho cuando alguna parte no puede ejecutarse (los hilos necesitan sincronizarse)

Planificación de hilos en multiprocesadores

3) Asignación de procesador dedicado

- Cuando se planifica una aplicación, se asigna un procesador a cada uno de sus hilos hasta que termine la aplicación
- Algunos procesadores pueden estar ociosos → No hay multiprogramación de procesadores

4) Planificación dinámica

- La aplicación permite que varíe dinámicamente el número de hilos de un proceso
- El SO ajusta la carga para mejorar la utilización de los procesadores

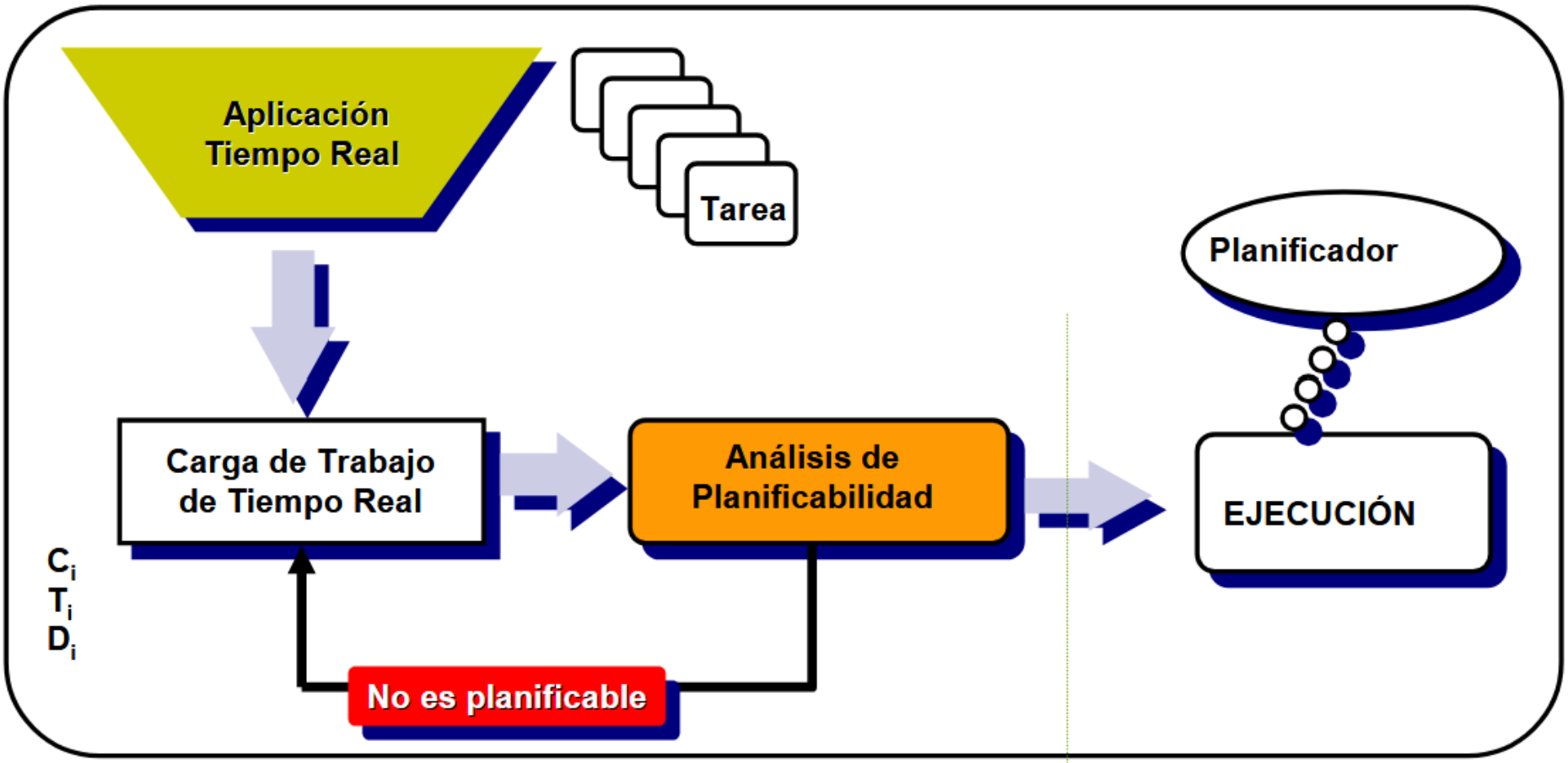
Sistemas de tiempo real

- La exactitud del sistema no depende sólo del resultado lógico de un cálculo sino también del instante en que se produzca el resultado.
- Las tareas o procesos intentan controlar o reaccionar ante sucesos que se producen en “tiempo real” (eventos) y que tienen lugar en el mundo exterior.
- Características de las tareas de tiempo real:
  - Tarea de tiempo real duro: debe cumplir su plazo límite
  - Tarea de tiempo real suave: tiene un tiempo límite; pero no es obligatorio
  - Periódicas: se sabe cada cuánto tiempo se tiene que ejecutar
  - Aperiódicas: tiene un plazo en el que debe comenzar o acabar o restricciones respecto a esos tiempos; pero son impredecibles

Planificación de sistemas de tiempo real

- Los distintos enfoques dependen de:
  - Cuando el sistema realiza un análisis de viabilidad de la planificación
    - Estudia si puede atender a todos los eventos periódicos dado el tiempo necesario para ejecutar la tarea y el periodo
  - Si se realiza estática o dinámicamente
  - Si el resultado del análisis produce un plan de planificación o no

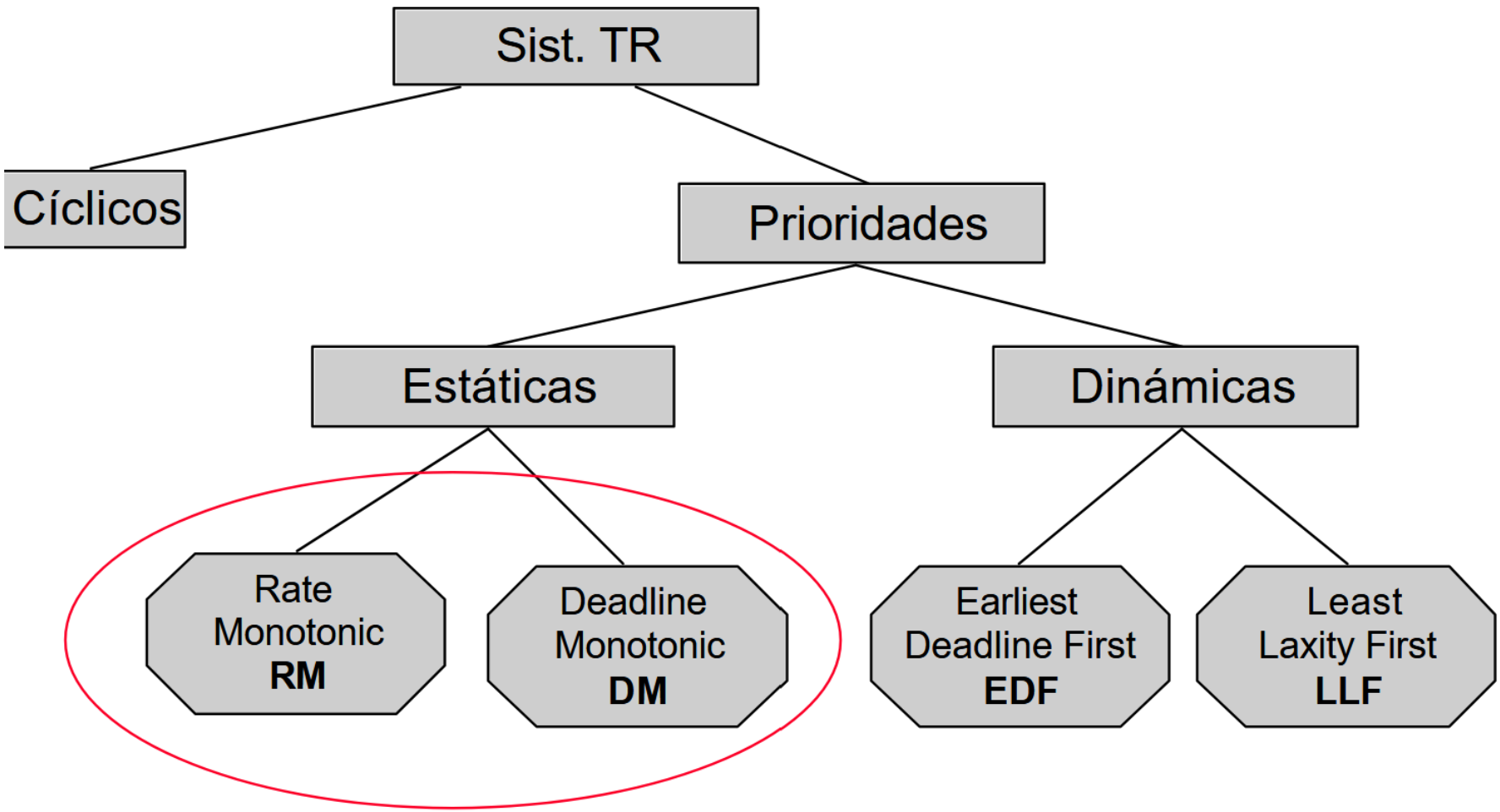
Planificación de sistemas de tiempo real



Planificación en sistemas de tiempo real (y II)

- Enfoques estáticos dirigidos por tabla
  - Análisis estático que genera una planificación que determina cuándo empezará cada tarea
- Enfoques estáticos expulsivos dirigidos por prioridad
  - Análisis estático que no genera una planificación, sólo se usa para dar prioridad a las tareas. Usa planificación por prioridades
- Enfoques dinámicos basados en un plan
  - Se determina la viabilidad en tiempo de ejecución (dinámicamente): se acepta una nueva tarea si es posible satisfacer sus restricciones de tiempo
- Enfoques dinámicos de menor esfuerzo (el más usado)
  - No se hace análisis de viabilidad. El sistema intenta cumplir todos los plazos y aborta ejecuciones si su plazo ha finalizado

Planificadores de Tiempo Real



Planificador de Prioridades Monótonas en Frecuencia (RM; Rate Monotonic)

El Rate Monotonic Scheduling (RMS) es un algoritmo de planificación estática y apropiativa diseñado para sistemas de tiempo real periódicos.

Su principio básico es: A menor período (mayor frecuencia de ejecución), mayor prioridad.

Tarea	Tiempo de Ejecución (C)	Período (T)	Prioridad (RM)
T1	1 ms	4 ms	Alta (1)
T2	2 ms	6 ms	Baja (2)

- Según RM → **T1** tiene mayor prioridad porque su período (4 ms) es menor que el de **T2 (6 ms)**.

Planificador de Prioridades Monótonas en Frecuencia (RM; Rate Monotonic)

Teorema:

Un conjunto de N tareas será planificable bajo la política de planificación Rate Monotonic si se cumple la siguiente desigualdad:

Utilización total del CPU $$U = \sum_{i=1}^{N} \frac{Ci}{Ti} < N\left(2^{1/N} - 1\right)$$

Condición de factibilidad (criterio de Liu & Layland):

Planificador de Prioridades Monótonas en Frecuencia (RM; Rate Monotonic)

Tarea	Tiempo de Ejecución (C)	Período (T)	Prioridad (RM)
T1	1 ms	4 ms	Alta (1)
T2	2 ms	6 ms	Baja (2)

- Según RM → **T1** tiene mayor prioridad porque su período (4 ms) es menor que el de **T2 (6 ms)**.

Teorema:

Un conjunto de N tareas será planificable bajo la política de planificación Rate Monotonic si se cumple la siguiente desigualdad:

$$U = \sum_{i=1}^{N} \frac{Ci}{Ti} < N\left(2^{1/N} - 1\right)$$

Para 2 tareas, el límite ≈ **0.828**

Utilización total del CPU $U = \frac{1}{4} + \frac{2}{6} = 0.25 + 0.333 = 0.583 < 0.828$

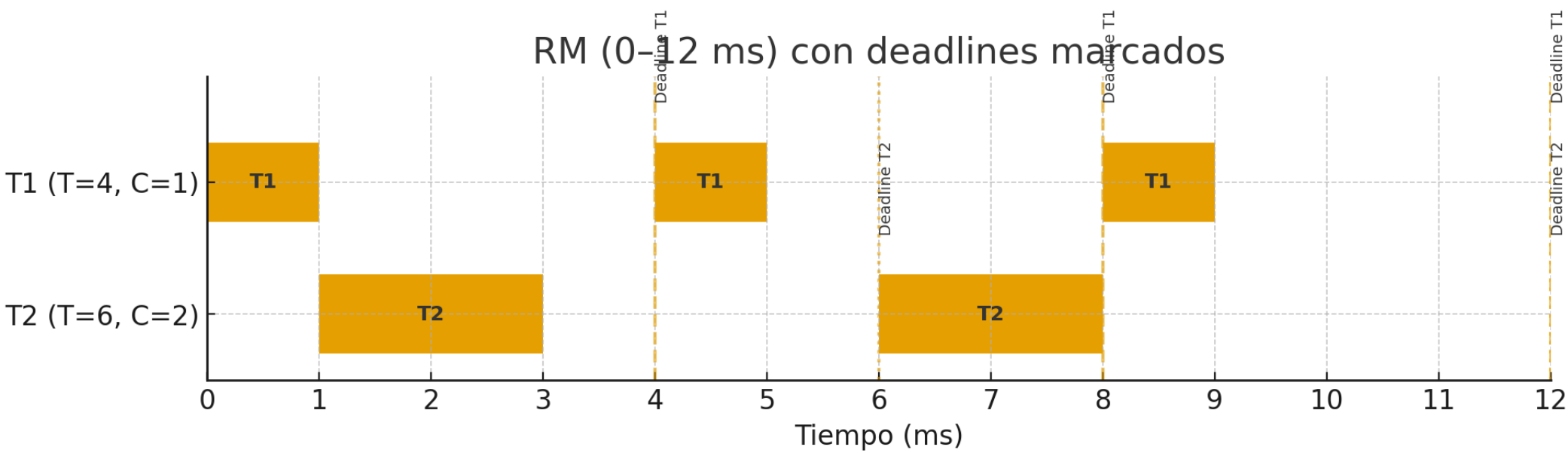
El sistema es **planificable con RM**.

Planificador de Prioridades Monótonas en Frecuencia (RM; Rate Monotonic)

Las tareas se repiten según sus períodos:

- T1 → cada 4 ms
- T2 → cada 6 ms

Mínimo común múltiplo (LCM) = 12 ms **hiperperiodo**



La prioridad se asigna según el **período (P)**.

Planificador de Prioridades Monótonas en Frecuencia (RM; Rate Monotonic)

Tarea	C (ms)	T (ms)
T1	1	4
T2	2	6
T3	1	8

- Realizar los cálculos del análisis de viabilidad de la planificación.
- Construye el Diagrama de Gantt, mostrando todos los periodos.
- Calcula el Tiempo de Retorno Promedio para todos los procesos.

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Esta política de planificación es idéntica a la Rate Monotic, pero ahora los plazos de finalización pueden ser distintos a los periodos (menores o iguales):

En **RM**, se asume que $D_i = T_i$.

En **DM**, se admite $D_i \leq T_i$.

La asignación de prioridades se hace en este caso en orden inverso al plazo de finalización, de forma que la tarea con plazo de finalización más breve tenga mayor prioridad.

Teorema:

Un sistema de tareas independientes e interrumpibles, que están en fase y sus plazos de finalización son menores o iguales que sus respectivos periodos, pueden ser planificadas en un único procesador bajo la política DM sí y solo sí pueden ser planificadas con cualquier otro planificador de prioridades fijas.

DM ≠ RM cuando $D_i \neq T_i$.

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C (Tiempo de Ejecución)	P (Período)	D (Plazo o Deadline)
T_1	10 ms	50 ms	50 ms
T_2	20 ms	100 ms	100 ms
T_3	5 ms	200 ms	200 ms

$$D_1 = T_1, \quad D_2 = T_2, \quad \ldots$$

$$\Rightarrow \text{Prioridades (DM)} = \text{Prioridades (RM)}$$

Tarea	P (Período)	Prioridad Asignada
T_1	50 ms	**P1 (Más Alta)**
T_2	100 ms	P2 (Media)
T_3	200 ms	**P3 (Más Baja)**

Asignación de prioridades:

- El plazo más corto obtiene la prioridad más alta (P1).
- El plazo más largo obtiene la prioridad más baja (P3).

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i}$$

$$U = \frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{C_3}{P_3}$$

$$U = \frac{10}{50} + \frac{20}{100} + \frac{5}{200}$$

Utilización total del CPU $U = 0.20 + 0.20 + 0.025 = \mathbf{0.425} \quad (\mathbf{42.5\%})$

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Criterio de L.L. (Liu & Layland):

La utilización máxima teórica para n tareas es: $U_{\text{L.L.}} = n \times (2^{1/n} - 1)$.

Para $n = 3$ tareas:

$$U_{\text{L.L.}} = 3 \times (2^{1/3} - 1) \approx 3 \times (1.2599 - 1) \approx \mathbf{0.7798} \quad (\mathbf{77.98\%})$$

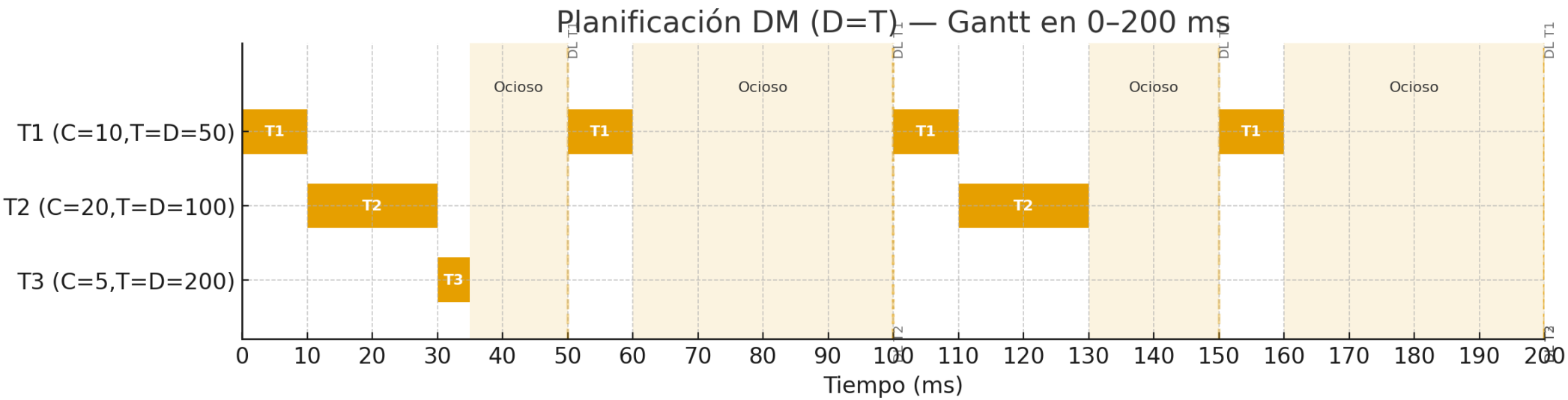
Dado que $\mathbf{U(42.5\%) < U_{L.L.}(77.98\%)}$, el conjunto de tareas es **planificable** con el algoritmo DM, y todas las tareas cumplirán su plazo.

> **Nota:** El criterio L.L. es una condición **suficiente**, no necesaria. Si la utilización supera este límite, aún podría ser planificable; en ese caso, se requiere un análisis de tiempo de respuesta más complejo.

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C (Tiempo de Ejecución)	P (Período)	D (Plazo o Deadline)
T_1	10 ms	50 ms	50 ms
T_2	20 ms	100 ms	100 ms
T_3	5 ms	200 ms	200 ms

mcm(50,100,200)=200 (**hiperperíodo**)



La prioridad se asigna según el **plazo o deadline (D)**.

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C	T	D	Prioridad RM	Prioridad DM
T1	10	50	50	Alta	Alta
T2	15	100	70	Media	**Alta (más que T3)**
T3	20	150	150	Baja	Baja

$R_i \leq D_i \Rightarrow$ **el conjunto es planificable con DM**

Test de garantía para planificación fija por prioridades (DM) usando análisis de tiempo de respuesta (RTA)

En RTA, para cada tarea τ_i (ordenadas por prioridad), se itera:

$$R_i^{(k+1)} = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j$$

R_i: **tiempo de respuesta** de la tarea i,
C_i: tiempo de ejecución de la tarea i,
T_j: período de la tarea j (más prioritaria),
$hp(i)$: conjunto de tareas con **prioridad más alta** que i,
$\lceil x \rceil$: redondeo hacia arriba (porque una tarea puede ser interrumpida varias veces).

Es planificable si $R_i \leq D_i$.

Una tarea i **cumple su deadline** si:

$$R_i \leq D_i$$

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C	T	D	Prioridad RM	Prioridad DM
T1	10	50	50	Alta	Alta
T2	15	100	70	Media	**Alta (más que T3)**
T3	20	150	150	Baja	Baja

$$R_i^{(k+1)} = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j$$

(1) T1 — la más prioritaria

No tiene interferencias (nadie puede preemptarla).

$$R_1 = C_1 = 10$$

$R_1 = C_1 = \boxed{10 \text{ ms}} \leq D_1 = 50$ ✓

Parámetro	Valor
$C_1 = 10$ ms	Tiempo de ejecución
$T_1 = 50$ ms	Período
$D_1 = 50$ ms	Deadline
$hp(1) = \emptyset$	No hay tareas más prioritarias (T_1 es la más alta)

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C	T	D	Prioridad RM	Prioridad DM
T1	10	50	50	Alta	Alta
T2	15	100	70	Media	**Alta (más que T3)**
T3	20	150	150	Baja	Baja

(2) T2 — interferida por T1

$\lceil 0.3 \rceil = 1$ (significa que T1 interrumpe 1 vez durante la ejecución de T2)

- **T2**: $C_2 = 15\ \text{ms},\ T_2 = 100\ \text{ms},\ D_2 = 70\ \text{ms}$
- Tareas de mayor prioridad $hp(2) = \{\mathbf{T1}\}$ (porque en DM el menor deadline manda)
- **T1**: $C_1 = 10\ \text{ms},\ T_1 = 50\ \text{ms},\ D_1 = 50\ \text{ms}$

$$R_2^{(k+1)} = C_2 + \sum_{\tau_j \in hp(2)} \left\lceil \frac{R_2^{(k)}}{T_j} \right\rceil C_j = C_2 + \left\lceil \frac{R_2^{(k)}}{T_1} \right\rceil C_1$$

Iteraciones

Inicialización **1.** $w_0 = C_2 = 15$

2. $w_1 = 15 + \left\lceil \frac{15}{50} \right\rceil \cdot 10 = 15 + 1 \cdot 10 = 25$ $\quad w_1 = C_2 + \left\lceil \frac{w_0}{T_1} \right\rceil C_1 \rightarrow w_1 = 15 + \left\lceil \frac{15}{50} \right\rceil \times 10$

3. $w_2 = 15 + \left\lceil \frac{25}{50} \right\rceil \cdot 10 = 15 + 1 \cdot 10 = 25$ → converge $\quad w_2 = 15 + \left\lceil \frac{w_1}{T_1} \right\rceil C_1 \rightarrow w_2 = 15 + \left\lceil \frac{25}{50} \right\rceil \times 10$

Resultado:

$$\boxed{R_2 = 25\ \text{ms}}$$

$R_2 = \boxed{25\ \text{ms}} \leq D_2 = 70$ ✓

$\lceil 0.5 \rceil = 1$

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C	T	D	Prioridad RM	Prioridad DM
T1	10	50	50	Alta	Alta
T2	15	100	70	Media	**Alta (más que T3)**
T3	20	150	150	Baja	Baja

(3) T3 — interferida por T1 y T2

- T3: $C_3 = 20,\ T_3 = 150,\ D_3 = 150$
- hp(3)={T1,T2}
  - T1: $C_1 = 10,\ T_1 = 50,\ D_1 = 50$
  - T2: $C_2 = 15,\ T_2 = 100,\ D_2 = 70$

$$R_3^{(k+1)} = C_3 + \left\lceil \frac{R_3^{(k)}}{T_1} \right\rceil C_1 + \left\lceil \frac{R_3^{(k)}}{T_2} \right\rceil C_2$$

1. **Inicialización**

   $w_0 = C_3 = 20$

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C	T	D	Prioridad RM	Prioridad DM
T1	10	50	50	Alta	Alta
T2	15	100	70	Media	**Alta (más que T3)**
T3	20	150	150	Baja	Baja

(3) T3 — interferida por T1 y T2

- T3: $C_3 = 20,\ T_3 = 150,\ D_3 = 150$
- hp(3)={T1,T2}
  - T1: $C_1 = 10,\ T_1 = 50,\ D_1 = 50$
  - T2: $C_2 = 15,\ T_2 = 100,\ D_2 = 70$

$$R_3^{(k+1)} = C_3 + \left\lceil \frac{R_3^{(k)}}{T_1} \right\rceil C_1 + \left\lceil \frac{R_3^{(k)}}{T_2} \right\rceil C_2$$

Paso 2 — Primera iteración

$w_0 = C_3 = 20$

$$w_1 = C_3 + \left\lceil \frac{w_0}{T_1} \right\rceil C_1 + \left\lceil \frac{w_0}{T_2} \right\rceil C_2 \longrightarrow w_1 = 20 + \left\lceil \frac{20}{50} \right\rceil \times 10 + \left\lceil \frac{20}{100} \right\rceil \times 15$$

$$w_1 = 20 + (1 \times 10) + (1 \times 15) = 45$$

$$\boxed{w_1 = 45 \text{ ms}}$$

$\frac{20}{50} = 0.4 \Rightarrow \lceil 0.4 \rceil = 1$

$\frac{20}{100} = 0.2 \Rightarrow \lceil 0.2 \rceil = 1$

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C	T	D	Prioridad RM	Prioridad DM
T1	10	50	50	Alta	Alta
T2	15	100	70	Media	**Alta (más que T3)**
T3	20	150	150	Baja	Baja

- **(3) T3 — interferida por T1 y T2**
  - T3: $C_3 = 20,\ T_3 = 150,\ D_3 = 150$
  - hp(3)={T1,T2}
    - T1: $C_1 = 10,\ T_1 = 50,\ D_1 = 50$
    - T2: $C_2 = 15,\ T_2 = 100,\ D_2 = 70$

$$R_3^{(k+1)} = C_3 + \left\lceil \frac{R_3^{(k)}}{T_1} \right\rceil C_1 + \left\lceil \frac{R_3^{(k)}}{T_2} \right\rceil C_2$$

Paso 3 — Segunda iteración

$$w_2 = C_3 + \left\lceil \frac{w_1}{T_1} \right\rceil C_1 + \left\lceil \frac{w_1}{T_2} \right\rceil C_2 \longrightarrow w_2 = 20 + \left\lceil \frac{45}{50} \right\rceil \times 10 + \left\lceil \frac{45}{100} \right\rceil \times 15$$

$$\frac{45}{50} = 0.9 \Rightarrow \lceil 0.9 \rceil = 1$$

$$\frac{45}{100} = 0.45 \Rightarrow \lceil 0.45 \rceil = 1$$

$$w_2 = 20 + 10 + 15 = 45$$

$$\boxed{w_2 = 45 \text{ ms}}$$

$$R_3 = \boxed{45 \text{ ms}} \leq D_3 = 150 \checkmark$$

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C	T	D	Prioridad RM	Prioridad DM
T1	10	50	50	Alta	Alta
T2	15	100	70	Media	**Alta (más que T3)**
T3	20	150	150	Baja	Baja

Tarea	R_i (ms)	D_i (ms)	Cumple
T1	10	50	✅
T2	25	70	✅
T3	45	150	✅

(Complemento rápido) Utilización total

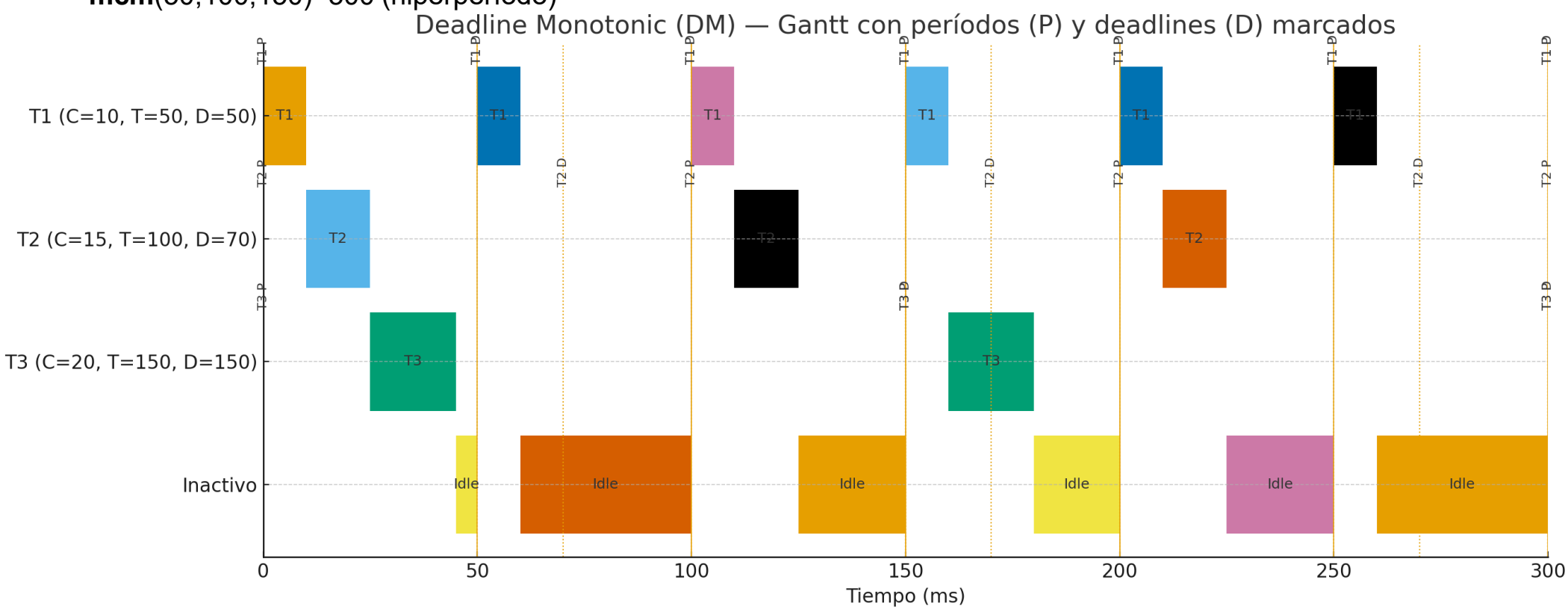
$U = \frac{10}{50} + \frac{15}{100} + \frac{20}{150} = 0.20 + 0.15 + 0.133\overline{3} \approx \mathbf{0.483} < 0.779$

(el límite de Liu–Layland para $n = 3$), por lo que también pasa el test de utilización suficiente.

Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C	T	D	Prioridad RM	Prioridad DM
T1	10	50	50	Alta	Alta
T2	15	100	70	Media	**Alta (más que T3)**
T3	20	150	150	Baja	Baja

mcm(50,100,150)=300 (hiperperíodo)



Planificador de Plazo Monótono (DM; Deadline Monotonic)

Tarea	C	T	D
T1	2	10	6
T2	3	15	10

- Realizar los cálculos del análisis de viabilidad de la planificación.
- Construye el Diagrama de Gantt, mostrando todos los periodos.

Planificador de Prioridades Estáticos - RT

Criterio	RM (Rate Monotonic)	DM (Deadline Monotonic)
Criterio de prioridad	Tareas con **menor período (T)** tienen **mayor prioridad**	Tareas con **menor deadline (D)** tienen **mayor prioridad**
Suposición fundamental	Se asume que $D_i = T_i$ (el plazo es igual al período)	Se permite $D_i \leq T_i$ (el plazo puede ser menor que el período)
Cuándo usarlo	Cuando **todas las tareas** deben terminar **antes de su siguiente activación** (es decir, D = T). Ejemplo: control de motores, muestreo periódico, tareas sensoriales.	Cuando **algunas tareas** deben completarse **antes de su siguiente liberación** (D < T). Ejemplo: tareas con respuesta crítica o límites intermedi
Ventaja	Más simple, óptimo entre planificadores de prioridad fija bajo $D = T$.	Más general: funciona también si los deadlines son menores que los períodos.
Desventaja	No funciona bien si $D < T$, porque podría violarse un plazo aunque se cumpla el período.	Requiere conocer y usar los deadlines de cada tarea (má parámetros).
Ejemplo típico	Control de temperatura: sensor cada 100 ms, actuador cada 50 ms → RM.	Sistema de frenado: lectura de sensor cada 100 ms, perc reacción debe ocurrir antes de 40 ms → DM.

Planificador de Prioridades Dinámicas

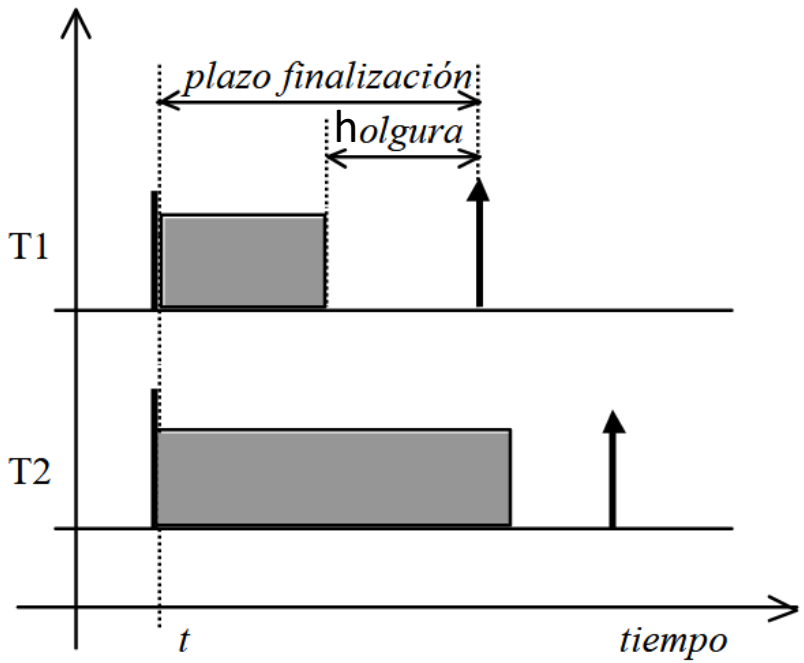
Los planificadores dinámicos son, al igual que los estáticos, planificadores basados en prioridades, pero se caracterizan por que las prioridades no son estáticas durante toda la vida del sistema; sino que varían en el tiempo siguiendo un algoritmo que define el planificador.

Básicamente se utilizan dos planificadores dinámicos, estos son:

- **EDF (Earliest Deadline First).** Se asignan las prioridades dando preferencia a las tareas con plazo de finalización más próximo.
- **LLF (Least Laxity First).** Se asignan las prioridades dando preferencia a las tareas con menor holgura.

Planificador de Prioridades Dinámicas

- En el tiempo t, según el planificador EDF T1 tendría más prioridad que T2, pero según el LLF, T2 tendría más prioridad que T1.
- El inconveniente de los planificadores dinámicos es que requieren un elevado tiempo de cálculo durante la ejecución del sistema.
- Por este motivo, el único que resulta realmente útil es el EDF.



Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)

Teorema: El planificador EDF puede producir una planificación válida en un solo procesador de un conjunto de trabajos J independientes con expulsión permitida y con plazos de ejecución arbitrarios, si y solo si J tiene planificaciones válidas.

```
Para cada unidad de tiempo
{
   Para cada proceso listo
   {
        El proceso con plazo cerrado se ejecutará primero
   }
 }
```

Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)

Test de planificabilidad: Definimos la densidad de un conjunto de tareas como:

$$D = \sum_{i=1}^{N} \frac{Ci}{Di}$$

- C_i: tiempo de ejecución de la tarea i
- D_i: *deadline* relativo (plazo máximo para terminar la tarea)
- N: número total de tareas

(D) mide **cuán "denso"** o **cargado** está el sistema en función de los *deadlines* (no de los períodos).

Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)

$$D = \sum_{i=1}^{N} \frac{Ci}{Di} \leq 1$$

- C_i: tiempo de ejecución de la tarea i
- D_i: *deadline* relativo (plazo máximo para terminar la tarea)
- N: número total de tareas

(D) mide **cuán "denso"** o **cargado** está el sistema en función de los *deadlines* (no de los períodos).

Caso A — $D_i = T_i$ (condición necesaria

Una condición es **necesaria** si **debe** cumplirse para que la planificabilidad sea posible. Si esta condición no se cumple, es **imposible** que el sistema sea planificable.

Significado: El hecho de que se cumpla la condición necesaria **no garantiza** la planificabilidad, pero es un **prerrequisito** indispensable.

Tarea	C_i	$T_i = D_i$
τ_1	2	5
τ_2	1	4
τ_3	2	8

$$D = \sum \frac{C_i}{D_i} = 0.40 + 0.25 + 0.25 = \mathbf{0.90} < 1$$

✅ Con EDF este caso es **planificable (garantizado)**.

Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)

Test de planificabilidad: Definimos la densidad de un conjunto de tareas como:

$$D = \sum_{i=1}^{N} \frac{Ci}{Di}$$

- C_i: tiempo de ejecución de la tarea i
- D_i: *deadline* relativo (plazo máximo para terminar la tarea)
- N: número total de tareas

(D) mide **cuán "denso"** o **cargado** está el sistema en función de los *deadlines* (no de los períodos).

Caso B — $D_i \neq T_i$ (condición suficiente)

Una condición es **suficiente** si, al cumplirse, **garantiza** que el sistema es planificable.

Significado: Si se cumple, tienes la **certeza** de la planificabilidad. Sin embargo, el sistema **podría ser planificable** incluso si esta condición suficiente **no se cumple**. Es una prueba sencilla pero a veces estricta.

Tarea	C_i	T_i	D_i
τ_1	1	6	3
τ_2	2	10	6
τ_3	1	12	5

$$D = \sum \frac{C_i}{D_i} = 0.333 + 0.333 + 0.200 = \mathbf{0.866} < 1$$

✅ Como $D < 1$, **EDF es planificable (suficiente).**

Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)

Test de planificabilidad: Definimos la densidad de un conjunto de tareas como:

$$D = \sum_{i=1}^{N} \frac{Ci}{Di}$$

- C_i: tiempo de ejecución de la tarea i
- D_i: *deadline* relativo (plazo máximo para terminar la tarea)
- N: número total de tareas

(D) mide **cuán "denso"** o **cargado** está el sistema en función de los *deadlines* (no de los períodos).

Caso C — $D_i = T_i$ (condición exacta)

Una condición es **necesaria y suficiente** (o **exacta**) si su cumplimiento **garantiza** la planificabilidad, y su incumplimiento **garantiza** la no planificabilidad.

Significado: Esta es la prueba **ideal**. Define la línea divisoria perfecta entre sistemas planificables y no planificables.

Tarea	C_i	$T_i = D_i$
T1	3	5
T2	4	8
T3	2	10

$$D = 0.60 + 0.50 + 0.20 = \mathbf{1.30} > \mathbf{1}$$

D=1.30>1, por tanto, **el sistema no puede ser planificado sin violar deadlines**, sin necesidad de simular.
La CPU debería estar disponible el 130 % del tiempo para atender las tres tareas, lo cual es imposible.

Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)

Tarea	C (Tiempo de ejecución)	T (Período)	D (Deadline relativo)
T1	1 ms	4 ms	4 ms
T2	2 ms	6 ms	6 ms

Calculamos la utilización

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{1}{4} + \frac{2}{6} = 0.583 \qquad U = 0.583 < 1$$

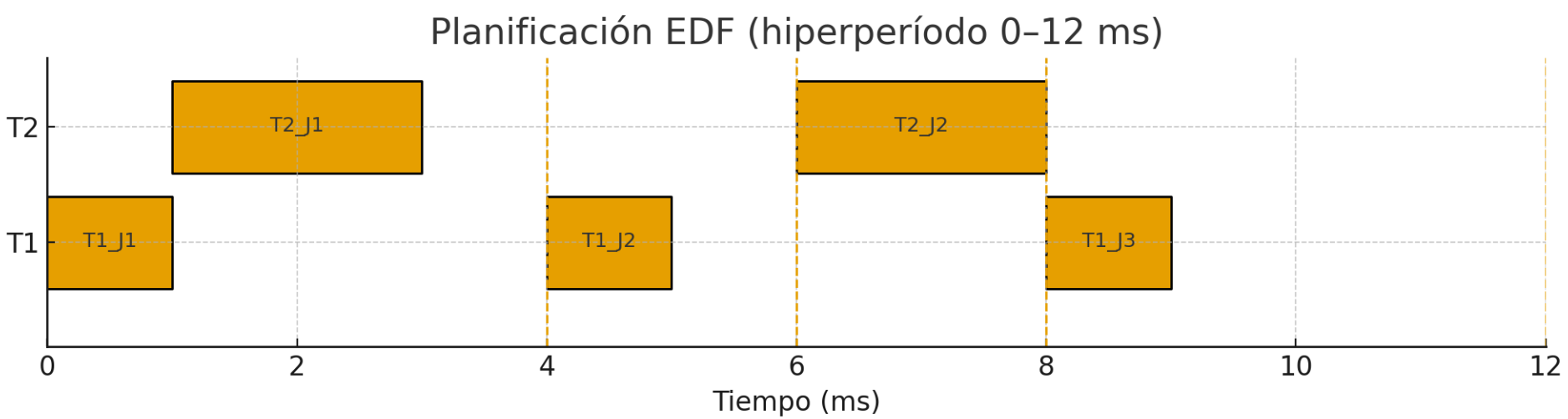
El **hiperperíodo** (mínimo común múltiplo de 4 y 6) es **12 ms**.

Prioridades cambian dinámicamente:

En 0 ms → T1 tiene deadline más cercano (4 ms).

En 6 ms → T2 tiene deadline 12 ms → se ejecuta.

En 8 ms → ambas tienen deadline 12 ms → empate.



Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)

Planteamiento (EDF, uniprocesador, apropiativo)

Tarea	C	T	D
T1	1	6	4
T2	2	10	7

- Realizar los cálculos del análisis de viabilidad de la planificación.
- Construye el Diagrama de Gantt, mostrando todos los periodos.

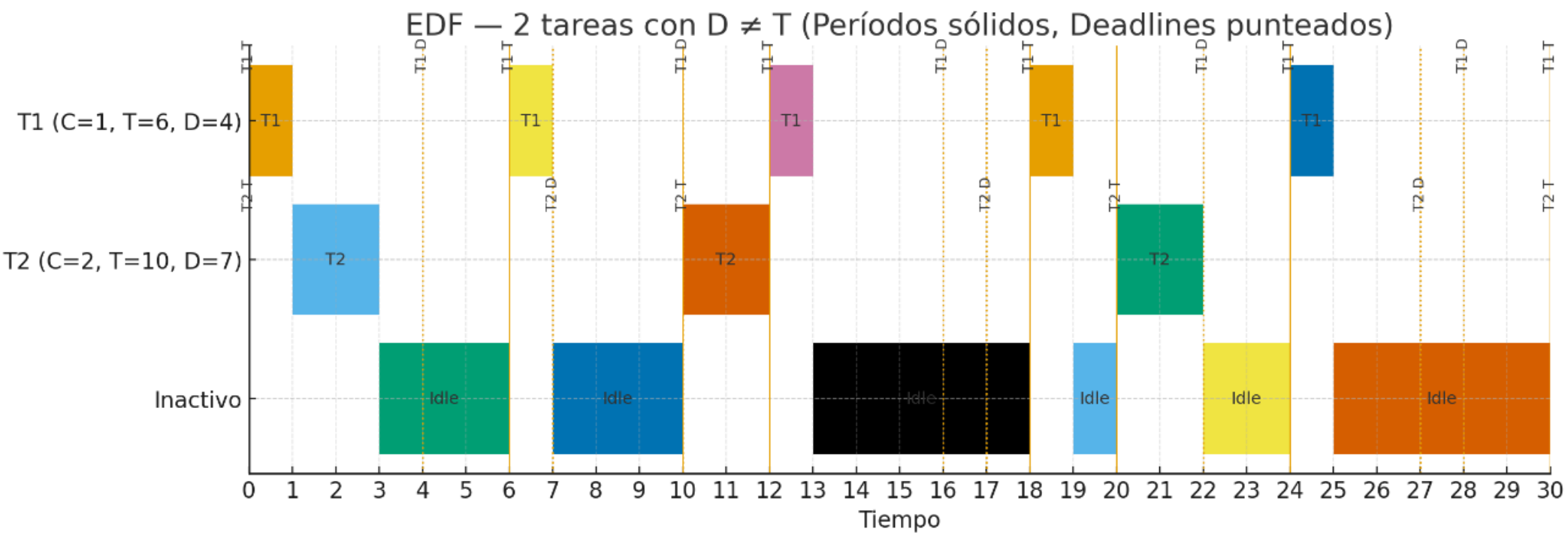
Densidad por deadlines (condición suficiente cuando $D \neq T$):

$$D = \frac{1}{4} + \frac{2}{7} = 0.25 + 0.2857 = \mathbf{0.536} < 1$$

Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)

Planteamiento (EDF, uniprocesador, apropiativo)

Tarea	C	T	D
T1	1	6	4
T2	2	10	7



Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)

Proceso	Tiempo de Ejecución (Execution Time)	Periodo (Period)	Deadline
P1	1	3	**3**
P2	1	4	**4**
P3	2	8	**8**

Procesos (asumo $D_i = T_i$):

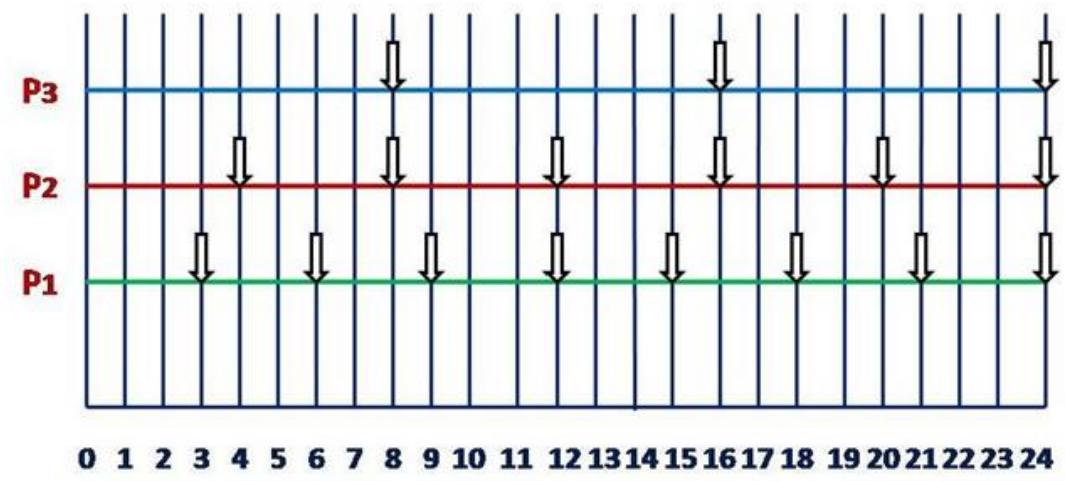
- **P1:** $C_1 = 1,\ T_1 = D_1 = 3$
- **P2:** $C_2 = 1,\ T_2 = D_2 = 4$
- **P3:** $C_3 = 2,\ T_3 = D_3 = 8$

1) Condición exacta EDF (como $D_i = T_i$)

$$U = \sum \frac{C_i}{T_i} = \frac{1}{3} + \frac{1}{4} + \frac{2}{8} = \frac{1}{3} + 0.25 + 0.25 = \frac{5}{6} \approx 0.833 \ \leq 1$$

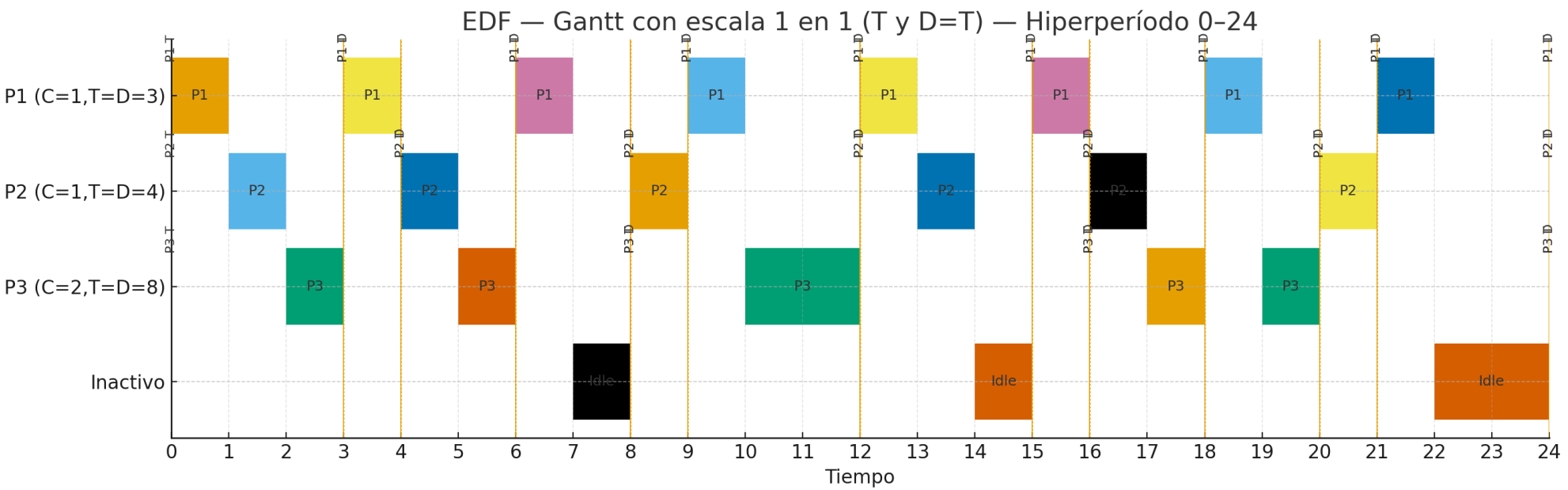
✅ **Planificable con EDF** (condición necesaria y suficiente).

Planificador de Prioridades Dinámicas EDF Early Deadline First (plazo más cercano primero)



mcm (3,4,8)=24

Proceso	Tiempo de Ejecución (Execution Time)	Periodo (Period)	Deadline
P1	1	3	**3**
P2	1	4	**4**
P3	2	8	**8**



Planificador de Prioridades Dinámicas LLF Early Deadline First (plazo más cercano primero)

El algoritmo LLF da prioridad a la tarea que tiene la mínima holgura (Laxity). La holgura representa cuánto tiempo puede esperar una tarea antes de que comience su ejecución sin incumplir su fecha límite (deadline).

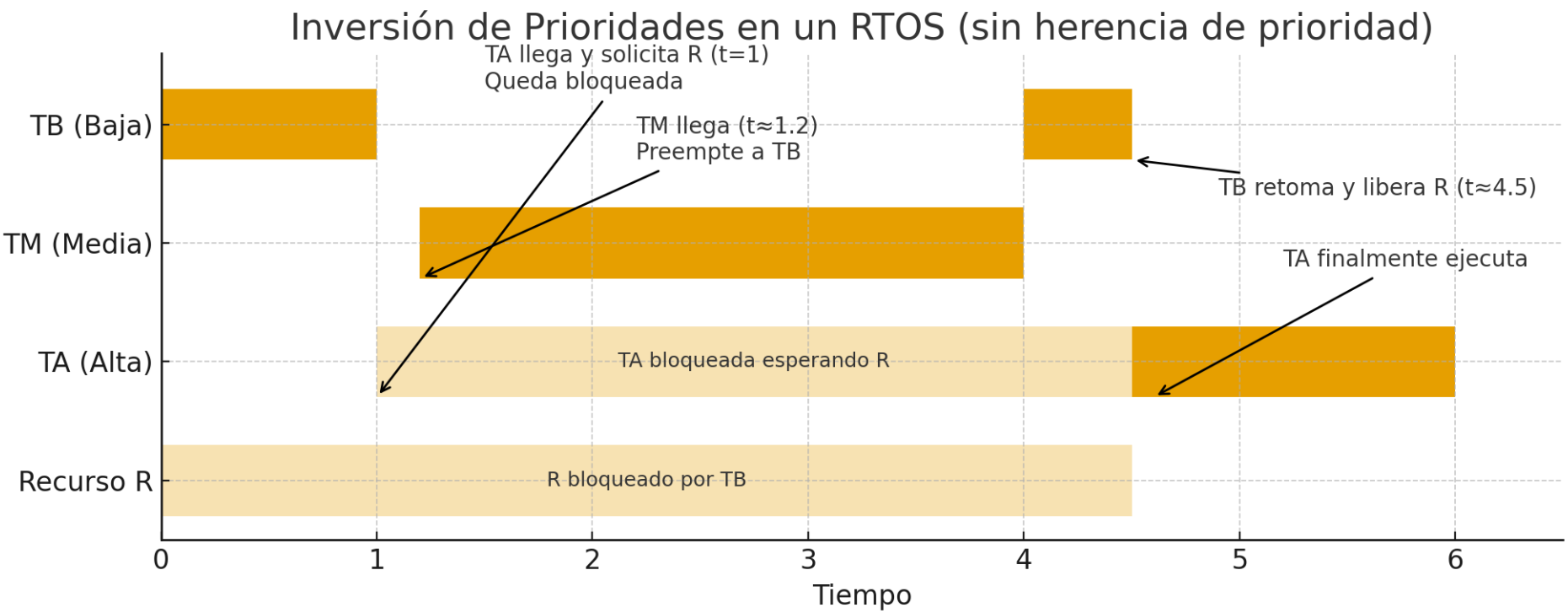
$$\text{Holgura} = \text{Deadline Absoluto} - \text{Tiempo Restante de Ejecución} - \text{Tiempo Actual}$$

Problema: Inversión de Prioridad

- Se da en un esquema de planificación de prioridad y cuando:
  - Una tarea de mayor prioridad espera por una tarea (proceso) de menor prioridad debido al bloqueo de un recurso de uso exclusivo (no compartible).
- La inversión de prioridades viola la premisa fundamental de los sistemas de tiempo real: que las tareas críticas (de alta prioridad) deben ejecutarse inmediatamente y sin interrupción por tareas menos importantes.

1. **Tarea de Baja Prioridad (T_B)**: Comienza a ejecutarse y adquiere un recurso compartido (R).
2. **Tarea de Media Prioridad (T_M)**: Llega una tarea de prioridad media que es ejecutable, pero no necesita el recurso R.
3. **Tarea de Alta Prioridad (T_A)**: Llega una tarea de alta prioridad (T_A) que **sí** necesita el recurso R.

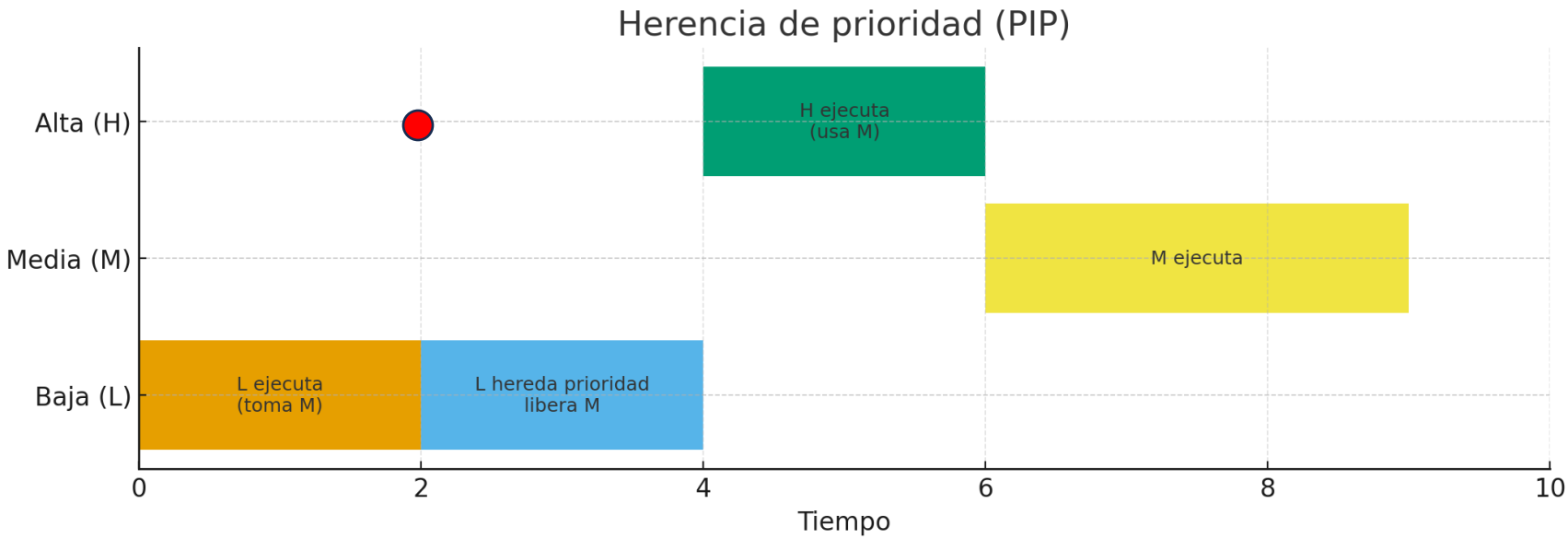
Problema: Inversión de Prioridad



Problema: Inversión de Prioridad

Herencia de prioridad: Cuando una tarea de alta prioridad (TA) se bloquea al intentar adquirir un recurso que está en manos de una tarea de baja prioridad (TB), el sistema operativo transfiere temporalmente la prioridad de TA a TB.

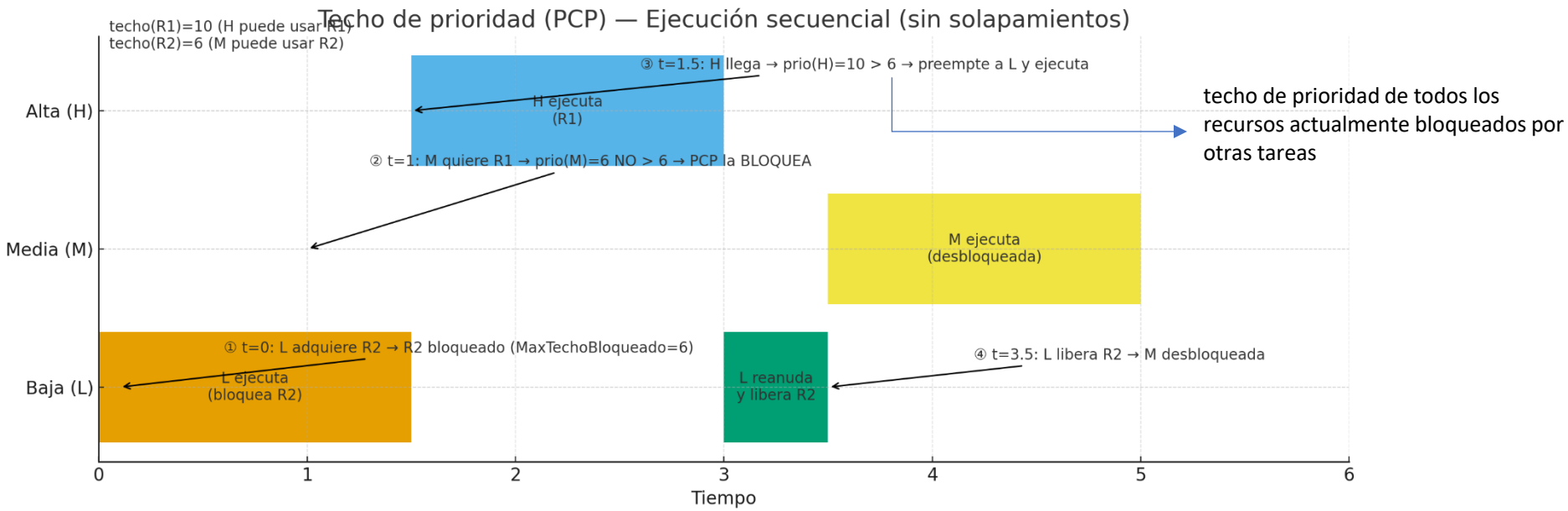
- TB ahora se ejecuta con la prioridad alta de TA, lo que le permite terminar rápidamente, liberar el recurso R, y salir de la sección crítica.
- Una vez que TB libera R, su prioridad se restablece a su nivel original. TA puede adquirir R y ejecutarse de inmediato.



Problema: Inversión de Prioridad

Techo de prioridad: Este es un enfoque más conservador y preventivo. A cada recurso (R) se le asigna un techo de prioridad igual a la prioridad más alta de cualquier tarea que pueda acceder a él.

- Una tarea solo puede entrar en una sección crítica (adquirir R) si su prioridad es estrictamente mayor que el techo de prioridad de cualquier otro recurso bloqueado por otras tareas.



En ambos, la tarea menos prioritaria vuelve a tener el valor de prioridad que tenía cuando libere el recurso.