PROCESOS

Rodrigo García Carmona
Universidad San Pablo-CEU
Escuela Politécnica Superior

OBJETIVOS

- Profundizar en el **concepto de proceso**, conociendo sus tres visiones complementarias.
- Analizar las diferentes formas de modelar los **estados de un proceso**, incluyendo el estado "suspendido".
- Entender la **implementación** del modelo de procesos, así como los componentes de la misma, destacando el **Bloque de Control de Proceso (PCB)** y el **Bloque de Control de Sistema (SCB)**.
- Comprender el concepto de ***thread***, entendiendo qué lo diferencia de un proceso y los diferentes niveles a los que puede encontrarse. Conocer la biblioteca de ***threads* POSIX**.
- Entender en qué consiste la **planificación** de procesos y por qué es necesaria.
- Conocer los **criterios** bajo los que puede evaluarse la planificación en un procesador.
- Estudiar las diferentes **políticas de planificación** existentes, así como los algoritmos que las implementan.
- Aprender a **evaluar** algoritmos para determinar su efectividad.

CONTENIDO

- Procesos y sus estados.
- Implementación de procesos.
- Hilos de ejecución (*threads*).
- Concepto de planificación.
- Criterios de planificación.
- Algoritmos de planificación.

Bibliografía

- W. Stallings: **Sistemas Operativos**.
  - Capítulos 3, 4 y 9.
- A.S. Tanenbaum: **Modern Operating Systems**.
  - Capítulo 2.

PROCESOS Y SUS ESTADOS

CONCEPTO DE PROCESO

- Existen numerosas razones para permitir la **ejecución concurrente de procesos**:
  - Compartir recursos físicos.
  - Compartir recursos lógicos.
  - Acelerar los cálculos.
  - Modularidad.
  - Comodidad.
- Existen diferentes visiones complementarias del concepto de proceso:
  - **Programa en ejecución**.
  - Proceso como **procesador virtual**.
  - Unidad de **asignación de recursos**.

PROCESO COMO PROGRAMA EN EJECUCIÓN

- **Proceso como programa en ejecución:**
  - **Programa:** lista de instrucciones. Ente pasivo.
  - **Proceso:** Programa en ejecución. Ente activo. Pasa por una serie de estados
- Un programa reside normalmente en un fichero que se encuentra físicamente en el disco y es **cargado en memoria cuando se crea el proceso** para ejecutarlo.
- Puede haber dos o más procesos asociados a la ejecución de un mismo programa. Es habitual que un proceso genere más procesos durante su ejecución.

PROCESO COMO PROCESADOR VIRTUAL (I)

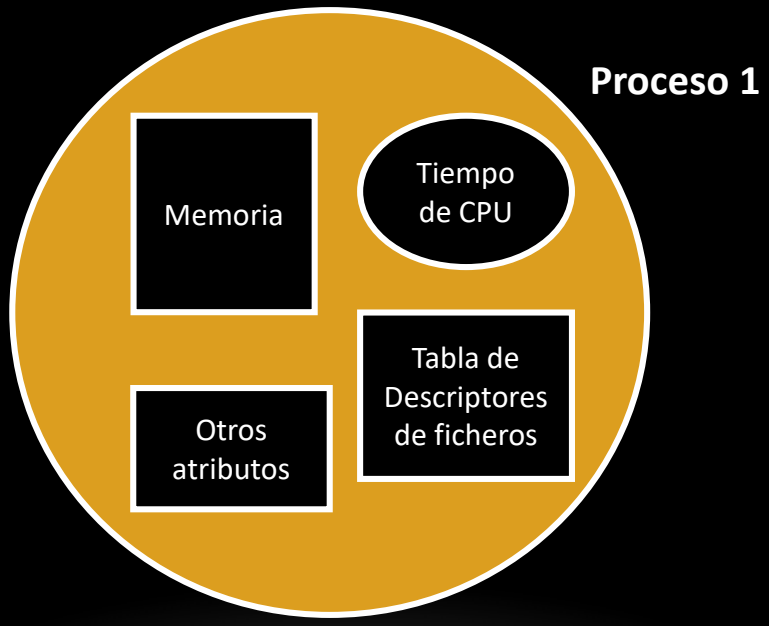
- **Proceso como procesador virtual:**
  - **Proceso:** cada una de las actividades paralelas que se ejecutan en la máquina. Da la apariencia de que todas ellas se ejecutan sobre procesadores diferentes en paralelo.
- **Ejemplos:**
  - **Procesos de usuario:**
    - Un proceso de edición de un sistema de tiempo compartido.
    - Un trabajo en un sistema por lotes.
    - La compilación de un programa fuente en C: el programa que ejecuta el proceso es el propio compilador.
  - **Procesos de sistema:**
    - "*Swapper*", que decide qué procesos hay que extraer de la memoria e introducir en ella cuando la cantidad de memoria libre no llega o supera ciertos límites.
    - "Page daemon", el encargado de utilizar reemplazo para liberar marcos y pasarlos a la reserva.

PROCESO COMO PROCESADOR VIRTUAL (II)

- El sistema operativo simula la existencia de N procesadores virtuales (procesos) a partir de una CPU o procesador físico.
  - Cada procesador virtual ejecuta **secuencialmente** un único programa.
  - Todos los procesos se ejecutan **concurrentemente**.
- **Ejecución secuencial:** La ejecución de un proceso se dice que es secuencial porque sus operaciones son ejecutadas por la CPU una tras otra, en el orden que dicte el programa.
- **Ejecución concurrente:** La ejecución de dos procesos se dice que es concurrente porque éstos se pueden ejecutar en paralelo.
  - **Concurrencia real:** si cada proceso se ejecuta sobre una CPU.
  - **Concurrencia virtual:** la CPU reparte su tiempo entre los procesos para simular su ejecución paralela.

PROCESO COMO UNIDAD DE ASIGNACIÓN DE RECURSOS

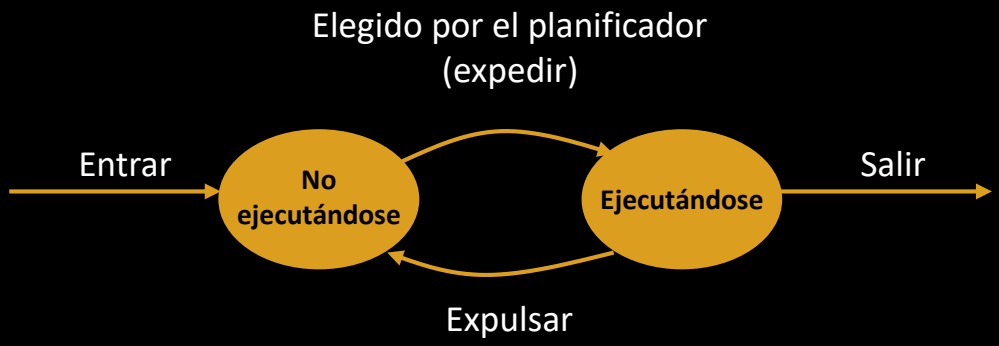
- Para ejecutar un programa se necesita un entorno formado por una serie de recursos: memoria, descriptores de ficheros y otros atributos del proceso.
- Un proceso se puede considerar como la unidad propietaria de todos esos recursos.

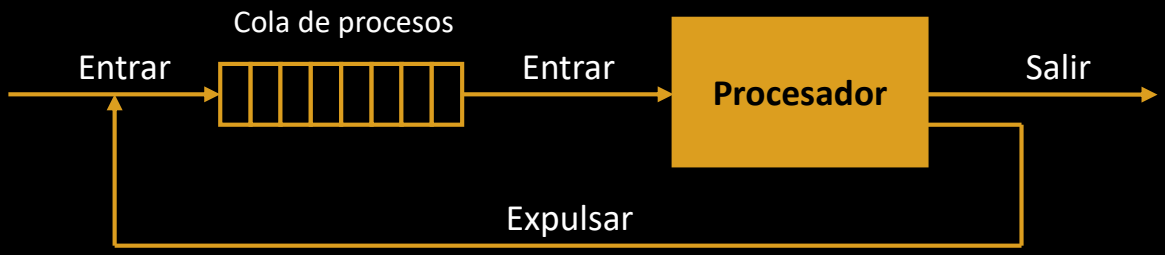


ESTADOS DE UN PROCESO

- Al ejecutar un proceso éste va cambiando de estado. El **estado** de un proceso se define como el **comportamiento que presenta en un instante** dado:
  - **Activo:** El proceso se puede ejecutar. No hay impedimentos en asignarle alguna CPU.
    - **En ejecución:** el proceso tiene asignada una CPU, las instrucciones se están ejecutando. Sólo puede haber un proceso en este estado por CPU.
    - **Preparado:** el proceso puede ser ejecutado pero está esperando a que se le asigne una CPU libre. Puede haber varios procesos en este estado.
  - **Bloqueado:** No puede ser ejecutado porque el proceso se encuentra esperando un evento determinado como:
    - La finalización de una operación de E/S (una lectura de teclado).
    - La comunicación con otro proceso, etc.

MODELO DE 2 ESTADOS (BÁSICO)





OPERACIONES SOBRE PROCESOS

- Hacia un modelo más realista…
- **Creación:** asignar todos los recursos que el proceso necesita para su ejecución.
  - **En Unix:** fork(). La utiliza el proceso init para crear un proceso que gestione cada terminal y el shell cada vez que recibe una nueva orden del usuario (si ésta es externa).
- **Terminación:** liberar los recursos previamente asignados al proceso. Puede ser:
  - **Terminación normal:** el proceso invoca su propia terminación. **En Unix:** exit().
  - **Terminación anormal:** el proceso termina por iniciativa del sistema operativo al detectar alguna condición anómala de error (violación de límites, errores aritméticos) o por iniciativa de algún otro proceso. **En Unix:** kill() y señales.
- **Bloquear:** pasar un proceso al estado bloqueado para que espere un evento u operación de E/S.
  - **En Unix:** ejecutar un read() sobre un pipe vacío.
- **Activación:** Pasar un proceso al estado activo cuando se produce el evento que esperaba.
  - **En Unix:** otro proceso efectúa un write() sobre un pipe en el que había un lector esperando. El lector se reactiva.

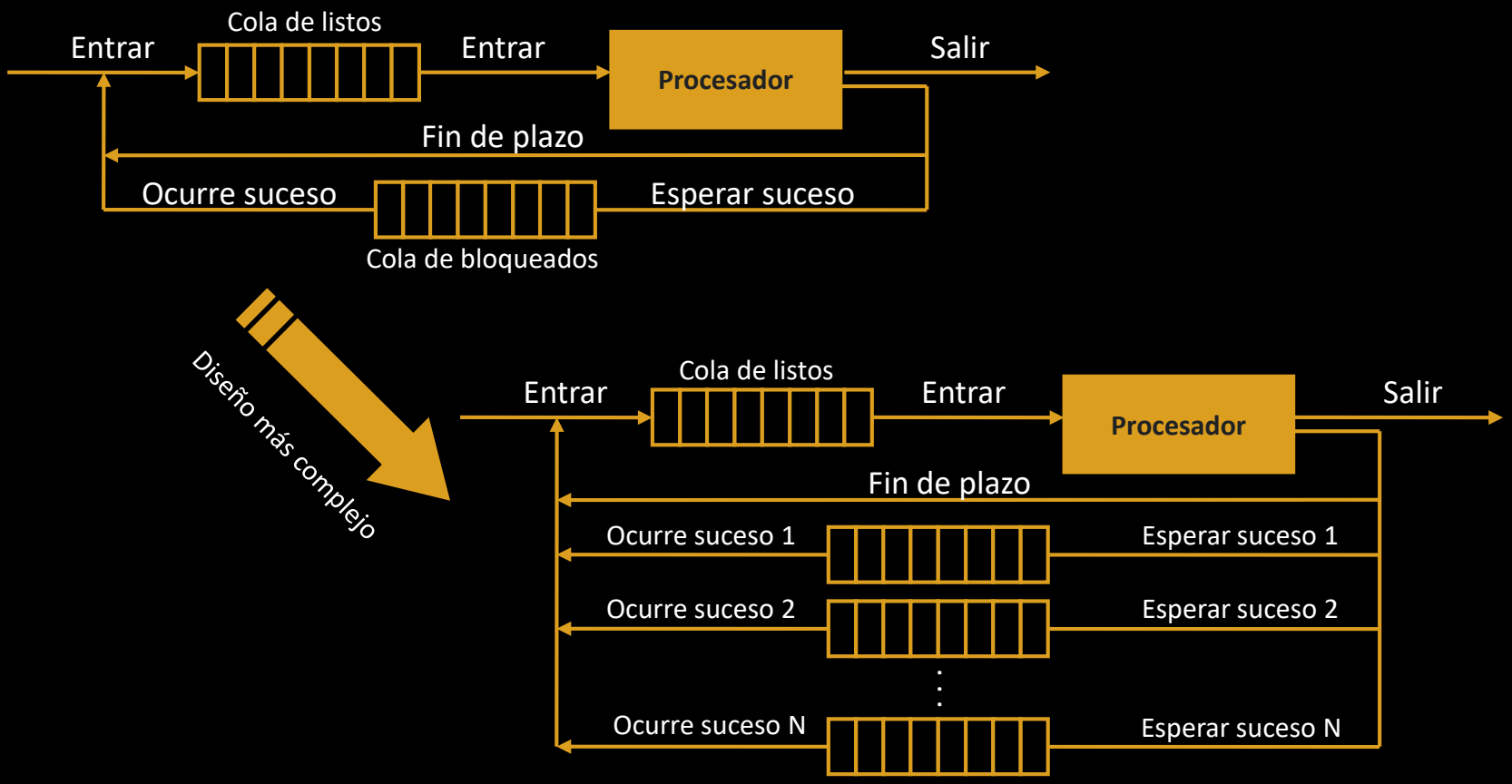
CREACIÓN Y TERMINACIONES DE PROCESOS

- **Razones para la creación de un proceso:**
  - Nuevo trabajo por lotes.
  - Conexión interactiva.
  - Creada por el sistema operativo para dar un servicio.
  - Generada por un proceso existente.

- **Razones para la terminación de un proceso:**
  - Terminación normal.
  - Tiempo límite excedido.
  - No hay memoria disponible.
  - Violación de límites.
  - Error de protección.
  - Error aritmético.
  - Tiempo máximo de espera rebasado.
  - Fallo de E/S.
  - Instrucción ilegal.
  - Instrucción privilegiada.
  - Mal uso de los datos.
  - Intervención del operador o del SO.
  - Terminación del padre.
  - Solicitud del padre.

MODELO DE 5 ESTADOS (AVANZADO)

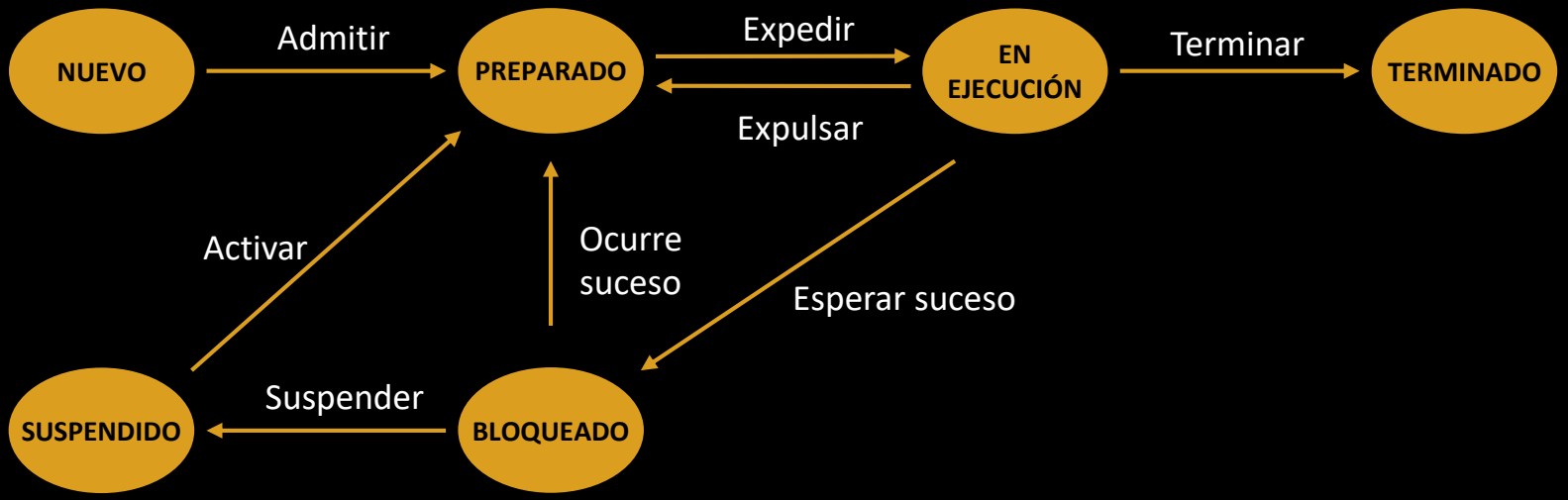
COLAS PARA EL MODELO DE 5 ESTADOS



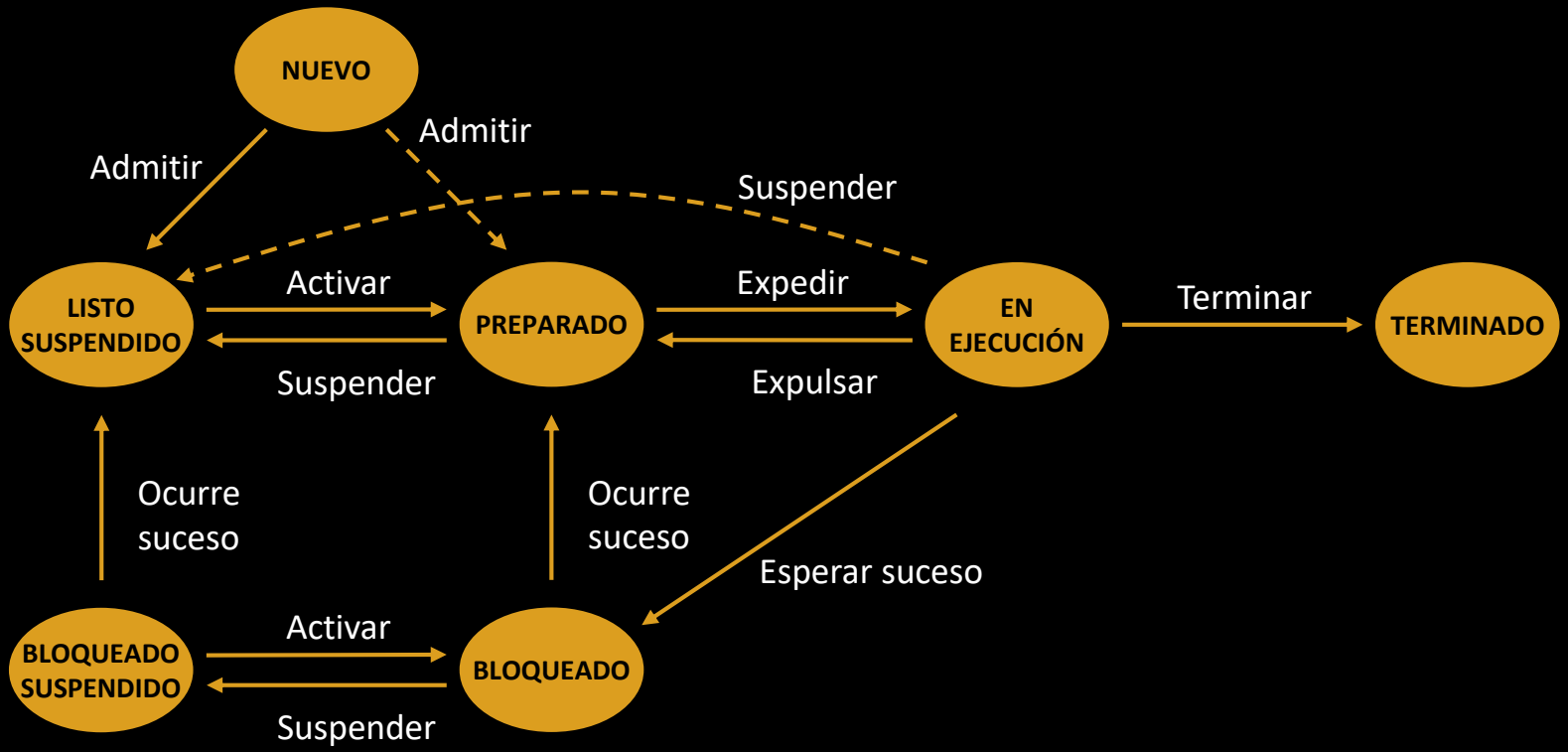
SUSPENSIÓN DE PROCESOS

- **Proceso suspendido:**
  - Un proceso que no está disponible de inmediato para su ejecución.
  - Ha sido movido a memoria virtual.
  - Hay que traerlo de vuelta a la memoria principal antes de poder ejecutarlo.
  - Puede estar esperando o no un suceso (bloqueado o no).
  - Debe permanecer en ese estado hasta que el agente que lo suspendió revoque la orden explícitamente.
- **Razones para la suspensión:**
  - Intercambio (*swapping*): se necesita espacio en memoria.
  - Solicitud del sistema operativo: se sospecha del proceso.
  - Solicitud de un usuario interactivo: para depurar o liberar recurso.
  - Ejecución de periódica: espera al siguiente intervalo.
  - Solicitud del proceso padre: para analizar al proceso hijo.
- Se puede distinguir en **suspendido/bloqueado** y **suspendido/listo**.

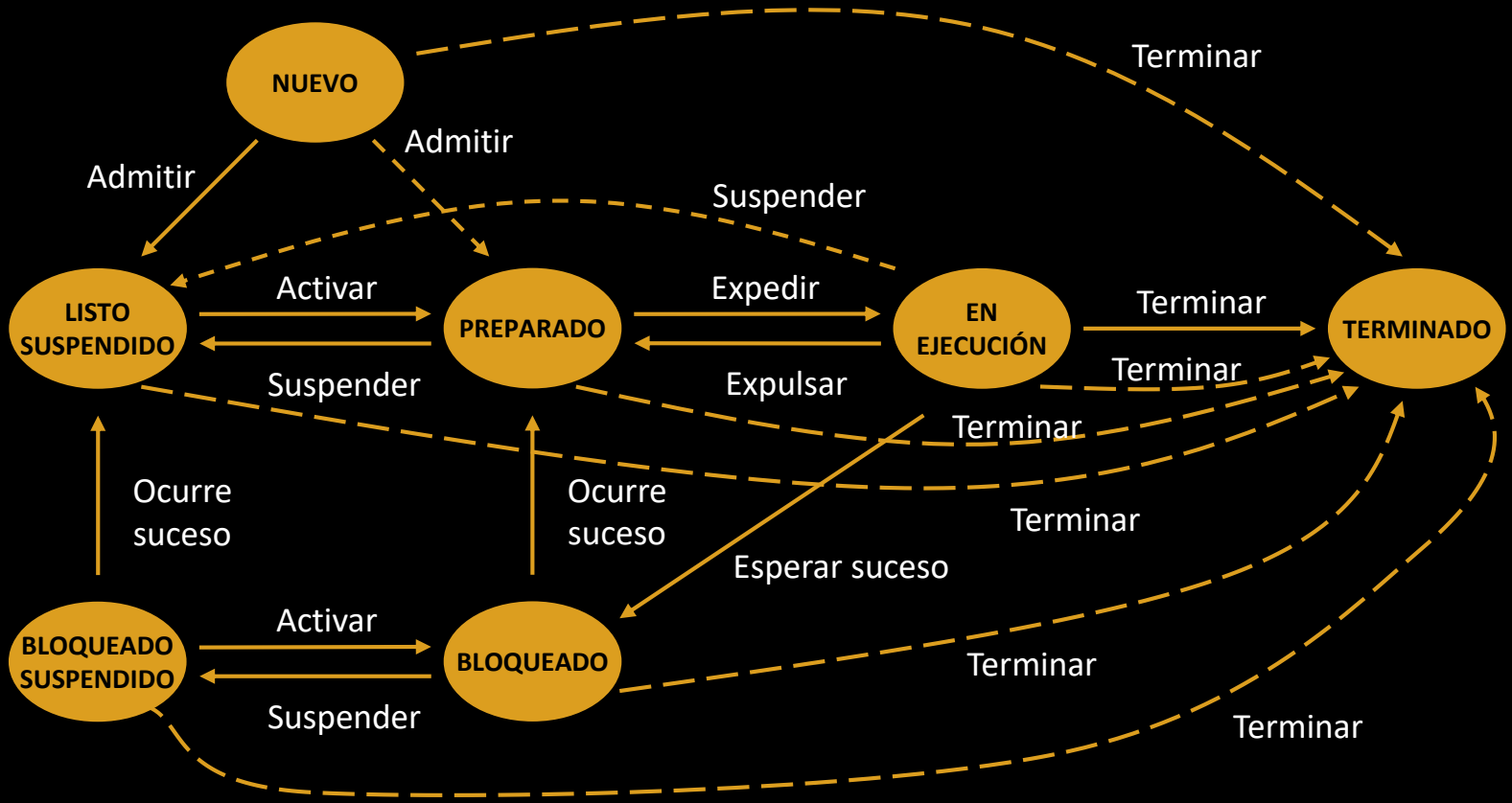
MODELO DE 6 ESTADOS (SUSPENSIÓN)



MODELO DE 7 ESTADOS (DOS SUSPENSIONES) INCOMPLETO



MODELO DE 7 ESTADOS (DOS SUSPENSIONES) COMPLETO



IMPLEMENTACIÓN DE PROCESOS

IMPLEMENTACIÓN DE PROCESOS

- Es conveniente pensar en un proceso como un **procesador virtual** que ejecuta un programa y que se implementa a partir de un procesador físico.
- La implementación de procesos requiere:
  - **Bloque de control de un proceso** PCB (*Process Control Block*), una estructura de datos para administrar el proceso. Almacena información de **un proceso**.
  - **Bloque de control del sistema** SCB (*System Control Block*), una estructura de datos que para controlar la ejecución de los procesos. Almacena información de **todos los procesos**.
  - **Asignar los recursos** necesarios para su ejecución, fundamentalmente la memoria para almacenar el código, los datos y la pila.
  - **Multiplexar la CPU** entre los procesos: repartir el tiempo de la CPU entre los diferentes procesos para simular la ejecución paralela (concurrencia virtual).

BLOQUE DE CONTROL DE UN PROCESO

- Es una estructura de datos donde se almacenan los atributos de un proceso, es decir, la información asociada al mismo.
- En un sistema de multiprogramación se requiere una gran cantidad de información de cada proceso para su administración.
- Dicha información puede agruparse en **tres categorías:**
  - **Identificación de Proceso:** a cada proceso se le asigna un identificador numérico único. Puede ser simplemente un índice en una tabla de procesos.
  - **Información de estado del procesador (Contexto):** el contenido de los registros del procesador. Cuando se interrumpe un proceso, este contenido debe salvarse de forma que pueda restaurarse cuando el proceso reanude su ejecución.
  - **Información de control del proceso:** información necesaria para que el sistema operativo controle y coordine los diferentes procesos.

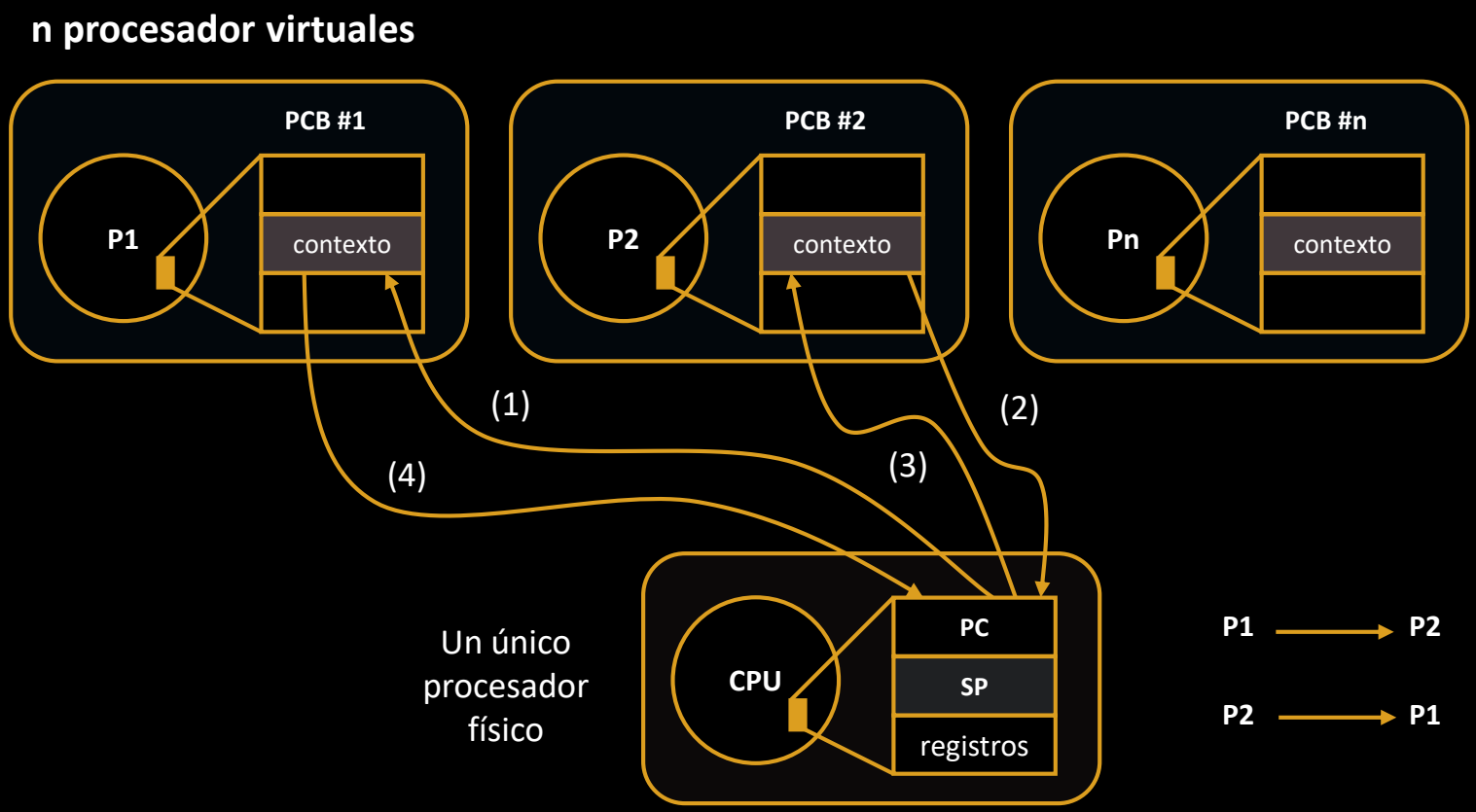
PARTES DEL BLOQUE DE CONTROL DE PROCESO

- **Identificación de Proceso:**
  - **Identificador de este proceso.**
  - **Identificador del proceso que creó a este proceso (identificador del proceso padre).**
  - **Identificador del usuario.**
- **Información de estado del procesador (Contexto):**
  - Registros generales (visibles para el usuario).
  - Contador del programa: contiene la dirección de la próxima instrucción a ejecutar
  - Códigos de condición: resultado de la operación aritmética/lógica más reciente (signo, cero, acarreo, desbordamiento)
  - Información de estado: indicadores de habilitación e inhabilitación de interrupciones.
  - Puntero de pila: cada proceso tiene una o más pilas LIFO asociadas, para almacenar parámetros y direcciones de retorno de procedimientos y de las llamadas al sistema. El puntero siempre apunta a la cima de la pila.
- **Información de control del proceso:**
  - Estado del proceso: preparado, en ejecución, suspendido, etc.
  - Prioridad de un proceso.
  - Información de planificación: depende del algoritmo de planificación, por ejemplo, la cantidad de tiempo que el proceso ha estado esperando y que se ejecutó la última vez.
  - Suceso: identidad del suceso que el proceso está esperando antes de poder reanudarse (solicitudes de E/S pendientes).
  - Mapa de memoria: memoria de código, memoria de datos y memoria de pila.

IMPLEMENTACIÓN DE VARIOS PROCESOS

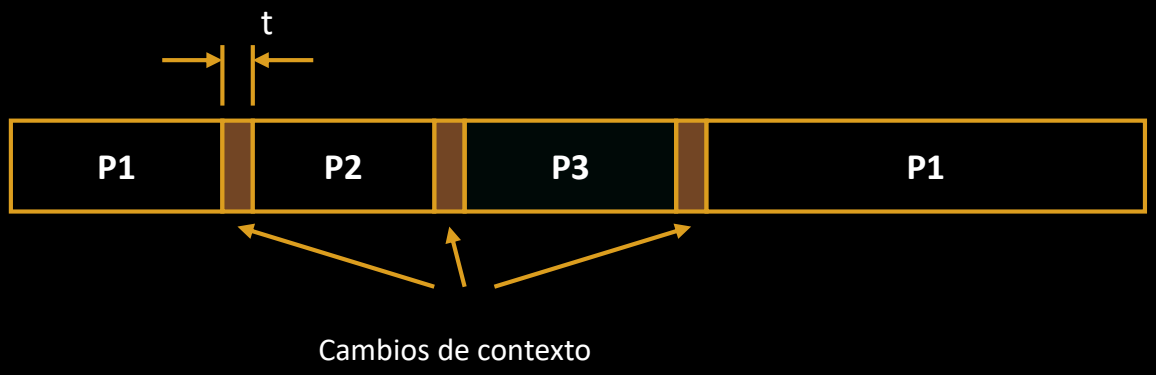
- La ejecución aparentemente simultánea de varios procesos en un mismo sistema, requiere repartir el tiempo de CPU entre los procesos a ejecutar (**concurrencia virtual**).
- Ello implica desasignar la CPU al proceso en ejecución y asignarla a un proceso preparado. Esta actividad se conoce con el nombre de **cambios de contexto**.
- Realizar un cambio de contexto **conlleva:**
  - Salvar el contexto del proceso en ejecución en su PCB.
  - Poner en la CPU el contexto del nuevo proceso (nuevo contador de programa).
  - Actualizar la información de control de los procesos.
- **La implementación de varios procesos implica cambios de contexto.**

CAMBIOS DE CONTEXTO



UTILIZACIÓN DE CPU EN CAMBIOS DE CONTEXTO

- Los cambios de contexto no son trabajo útil e implican una sobrecarga importante si se hacen con frecuencia: reducen la utilización.
- Ratio de utilización de CPU:

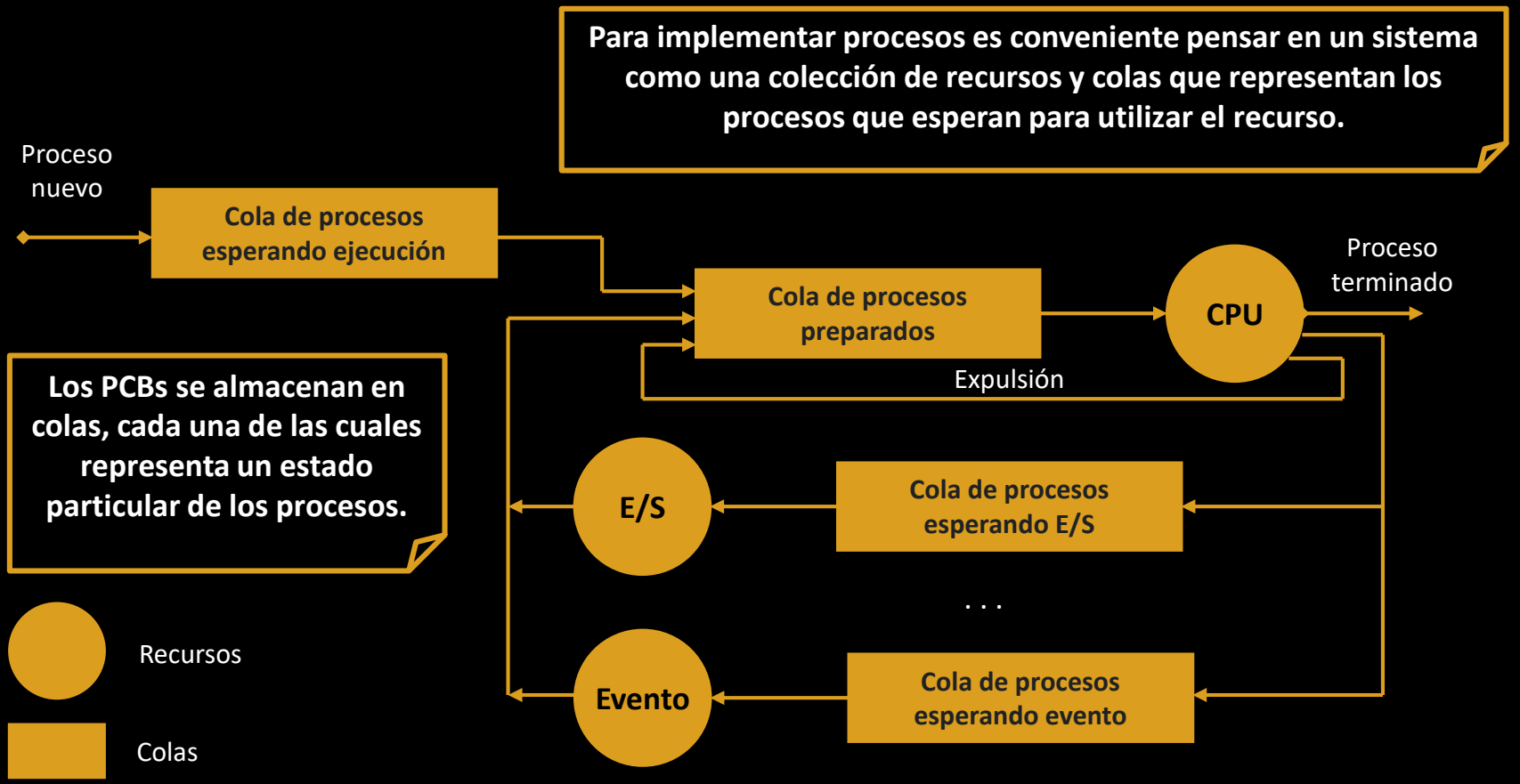


$$\text{Utilización de CPU} = \frac{T(P1) + T(P2) + T(P3)}{T(P1) + T(P2) + T(P3) + \mathbf{3t}}$$

MOTIVOS Y REPRESENTACIÓN DE LOS CAMBIOS DE CONTEXTO

- **Terminación** normal del proceso, el proceso en ejecución se acaba.
- El proceso en ejecución es **bloqueado** en espera de una E/S o de un evento.
- El proceso en ejecución es **expulsado** bien porque se ha agotado la cantidad de tiempo de CPU asignada bien porque se decide asignarla a otro proceso más prioritario.
- El sistema puede representar como un **diagrama de colas** donde:
  - Los círculos representan recursos.
  - Los rectángulos representan colas de procesos que esperan acceder a los recursos.

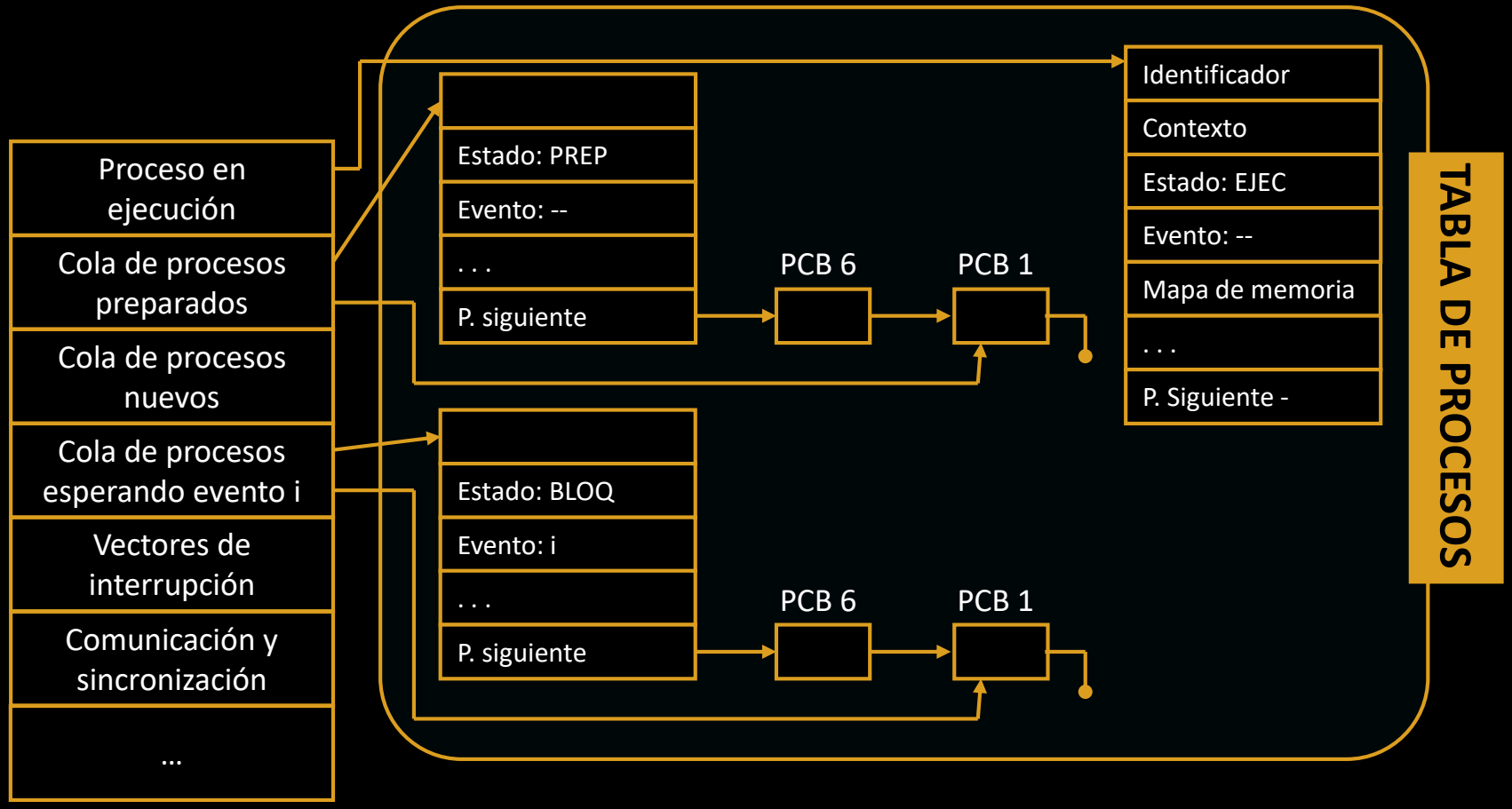
DIAGRAMA DE COLAS



BLOQUE DE CONTROL DEL SISTEMA

- Los datos que el sistema utiliza para controlar la ejecución de los procesos se encuentran en una estructura llamada **Bloque de Control del Sistema (SCB)**.
- **El SCB contiene en la gran mayoría de los casos la siguiente información:**
  - Tabla de procesos.
  - Puntero al PCB del proceso que está haciendo uso de la CPU.
  - Un puntero a la cola de PCBs de los procesos que están esperando ejecución.
  - Un puntero a la cola de PCBs de los procesos nuevos.
  - Un puntero a la cola de PCBs de los procesos que están esperando a que se produzca un evento, para poder volver a ejecutarse, no pudiendo hacerlo hasta que tenga lugar el evento esperado.
  - Los identificadores de las rutinas necesarias para tratar las interrupciones producidas por el hardware, software o errores indeseados.
  - Estructuras de datos relacionadas con la comunicación y sincronización de procesos.
  - Otras tablas y datos relacionados con la gestión de memoria y ficheros.

ESTRUCTURA DEL BLOQUE DE CONTROL DEL SISTEMA

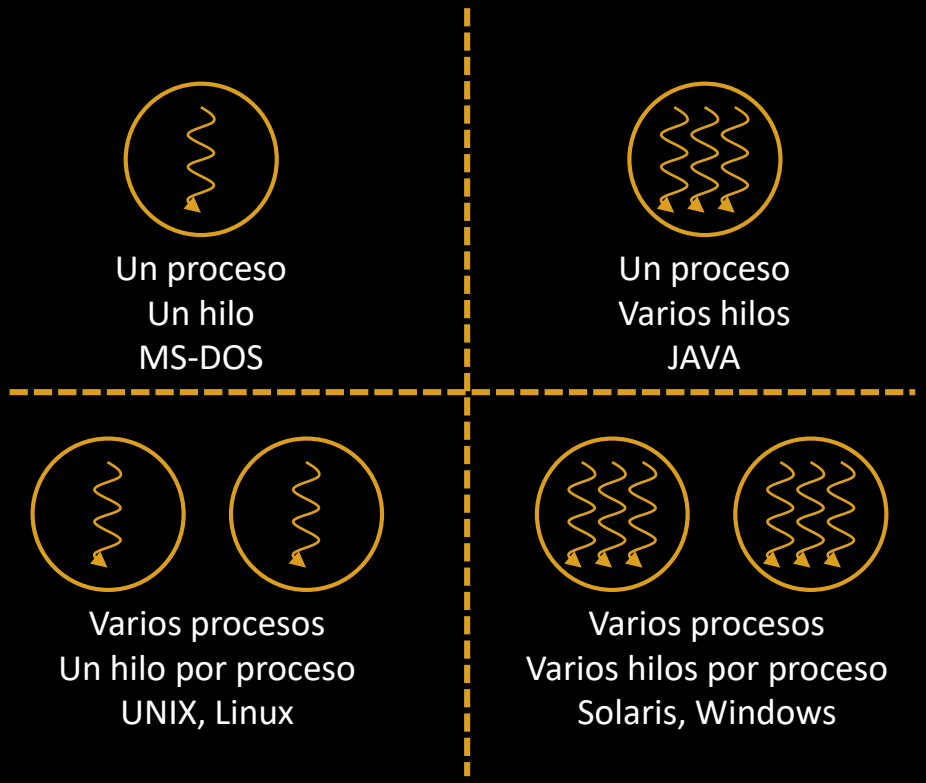


HILOS DE EJECUCIÓN *(THREADS)*

CONCEPTO DE *THREAD*

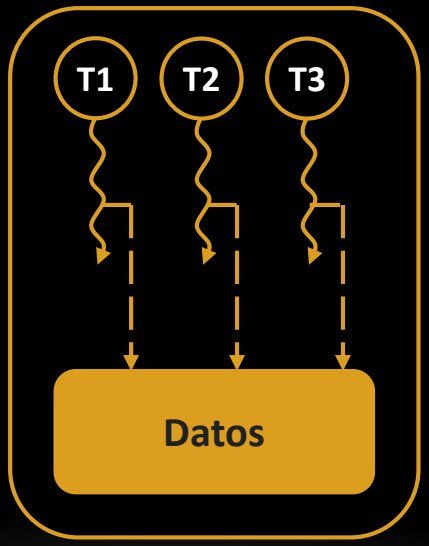
- Hasta ahora hemos definido un proceso dentro de un SO como:
  - **Unidad mínima de propiedad de recursos:** Espacio de memoria, tiempo de CPU, dispositivos de E/S, ficheros.
  - **Unidad mínima de planificación o ejecución:** Un camino de ejecución a través de un programa, que puede ser intercalada con otros procesos.
- **Entorno multihilo:** Estas dos características se separan en dos conceptos:
  - **Proceso:** Unidad mínima de propiedad de recursos.
  - **Hilo *(thread):*** Unidad mínima de planificación o ejecución.
- **Características de los hilos:**
  - Todos los hilos de un proceso comparten los mismos recursos.
  - Cada hilo se ejecuta independientemente.
  - Un hilo puede encontrarse en varios estados.
- **Ejemplos:**
  - Un videojuego puede tener un hilo para cada elemento móvil de la pantalla.
  - La máquina virtual de Java.

HILOS Y PROCESOS



COMPARTICIÓN DE MEMORIA EN HILOS

- Los procesos no comparten memoria (código o datos). Cada uno tiene su propio espacio de direcciones.
- Los hilos de ejecución de un mismo proceso pueden tener rutinas o variables comunes.
- No obstante, cada hilo tendrá su propia pila donde se guardarán las variables locales y argumentos de invocación.



VENTAJAS E INCONVENIENTES DE LOS HILOS

- **Ventajas:**
  - ¡Comparten recursos!
    - Mejora la comunicación entre hilos.
  - Crear o finalizar un hilo es más rápido (¡10 veces más!) que un proceso.
  - El cambio entre hilos de un mismo proceso es prácticamente despreciable.
- **Inconvenientes:**
  - ¡Comparten recursos!
    - Aumenta el peligro de fallo de sincronización/concurrencia.
  - Dos niveles de planificación: mayor complejidad.
  - Suspender un proceso implica suspender sus hilos.

FORMAS DE IMPLEMENTAR HILOS: A NIVEL DE USUARIO

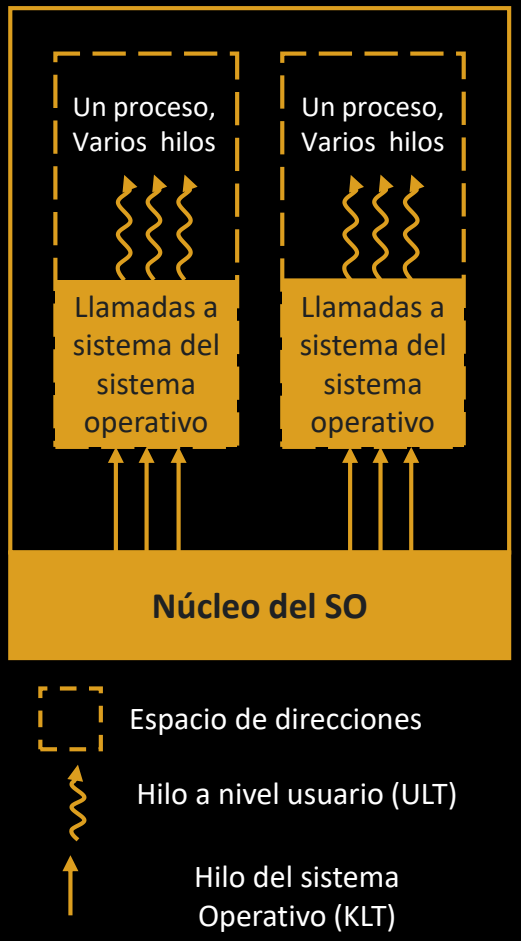
- **(*User-Level Threads* ULT):**

  Los hilos se crean a nivel del proceso de usuario vía un conjunto de procedimientos de biblioteca o vía el soporte de ejecución del lenguaje de programación.

  - El sistema operativo sólo crea un hilo de ejecución en el núcleo por cada espacio de direcciones (proceso).
  - El resto de hilos son implementados por el entorno de ejecución del lenguaje.
  - Es la implementación utilizada en los sistemas operativos que no soportan hilos.
  - Los cambios de contexto entre los hilos a nivel de usuario son rápidos, ya que **no involucran llamadas al sistema**.
  - La política de **planificación** de estos hilos **no está restringida a la propia del sistema operativo**.
  - Cuando este tipo de hilos invocan llamadas al sistema bloqueantes, normalmente **se bloquea todo el proceso**.
  - **No pueden explotar la capacidad de un sistema multiprocesador**, ya que el sistema operativo ve un único proceso.

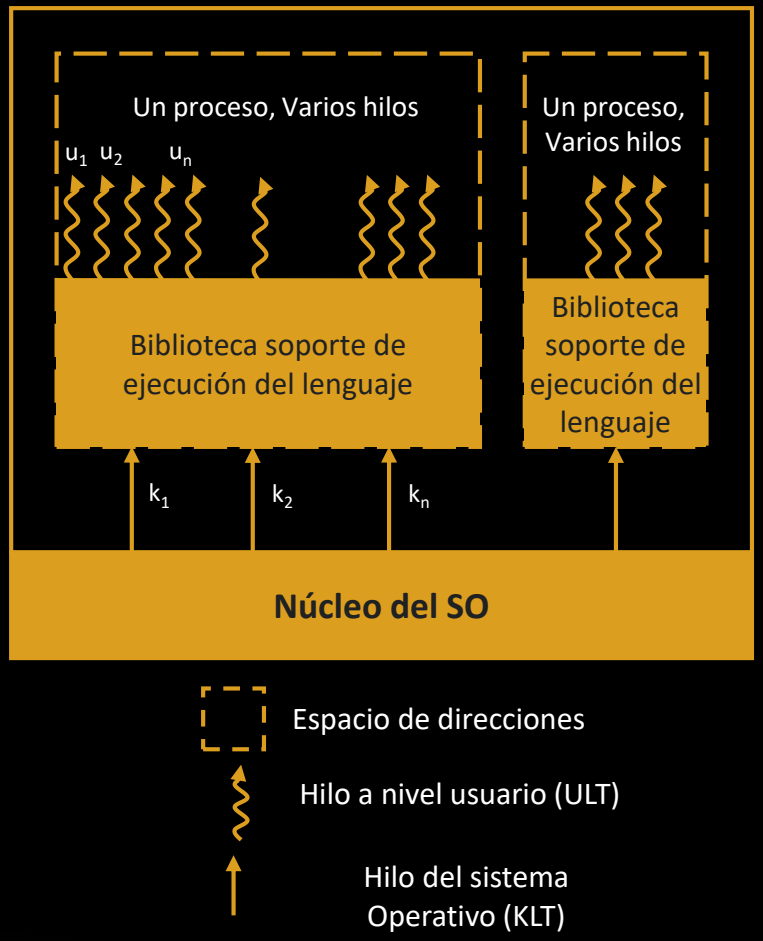
FORMAS DE IMPLEMENTAR HILOS: A NIVEL DE NÚCLEO

- **(*Kernel-Level Threads* KLT):**
  - El sistema operativo soporta hilos de ejecución y proporciona un conjunto de llamadas al sistema para su manipulación.
  - El sistema crea un hilo de ejecución por cada hilo a nivel de usuario.
  - Son la **unidad de planificación del sistema**.
  - El **bloqueo y activación** de hilos **corre a cargo del núcleo**.
  - Los cambios de contexto son manejados por el núcleo. Son más lentos que en el caso de los hilos a nivel de usuario, pero más rápidos que en el caso de los procesos.



FORMAS DE IMPLEMENTAR HILOS: APROXIMACIÓN HÍBRIDA

- Se implementan las dos clases de hilos anteriores.
- El sistema operativo permite **más de un hilo por proceso**.
- El soporte del lenguaje de programación utiliza **un hilo del núcleo para implementar un grupo de hilos de usuario**.
- Proporcionan flexibilidad y el máximo rendimiento potencial al programador de aplicaciones.
- Aumenta la complejidad del sistema operativo.



UTILIZACIÓN DE *THREADS*

- Los hilos pueden ser creados y manejados desde dos orígenes distintos:
  - **Utilizando un lenguaje de programación convencional y llamadas al sistema:**
    - Ejemplo: C y threads en POSIX.
  - **Utilizando construcciones lingüísticas de un lenguaje de programación que admita concurrencia:**
    - **Ejemplo:** Tareas en Ada95, threads en Java.

THREADS EN POSIX

Llamada
pthread_create(thread_id, attr, func, args) Crear un nuevo hilo de ejecución. El hilo de ejecución empieza en func y se le pasan los parámetros args.
pthread_exit(status) El hilo que la invoca finaliza su ejecución.
pthread_join(thread) Suspende la ejecución del hilo que invoca esta llamada, hasta que el thread_id acabe.
pthread_t pthread_self() Devuelve el identificador del thread que la invoca.

```
#include <stddef.h>
#include <pthread.h>

void * process(void * args){
  printf("%s",(char *)arg);
  fflush(stdout);
  pthread_exit(0);
}

int main(){
  pthread_t th_a,th_b;

  pthread_create
    (&th_a, NULL, process, "Hello");
  pthread_create
    (&th_b, NULL, process, "World");
  sleep(1);
}
```

CONCEPTO DE PLANIFICACIÓN

CONCEPTO DE PLANIFICACIÓN

- **Recursos reutilizables en serie:**

  Aquellos que sólo pueden estar asignados a un único proceso en un instante de tiempo dado.

  - **Ejemplos:** CPU, impresoras…

- **Características:**

  - **Escasez de recursos:** el número de recursos es inferior al número de procesos que compiten por ellos.
  - **Planificación del uso de recursos:** es necesario que el sistema operativo aplique una política para la asignación de dichos recursos a los procesos.
  - **Objetivos:** equidad, eficiencia, predicción, mínimo gasto…

Problema complejo: **Criterios de planificación**

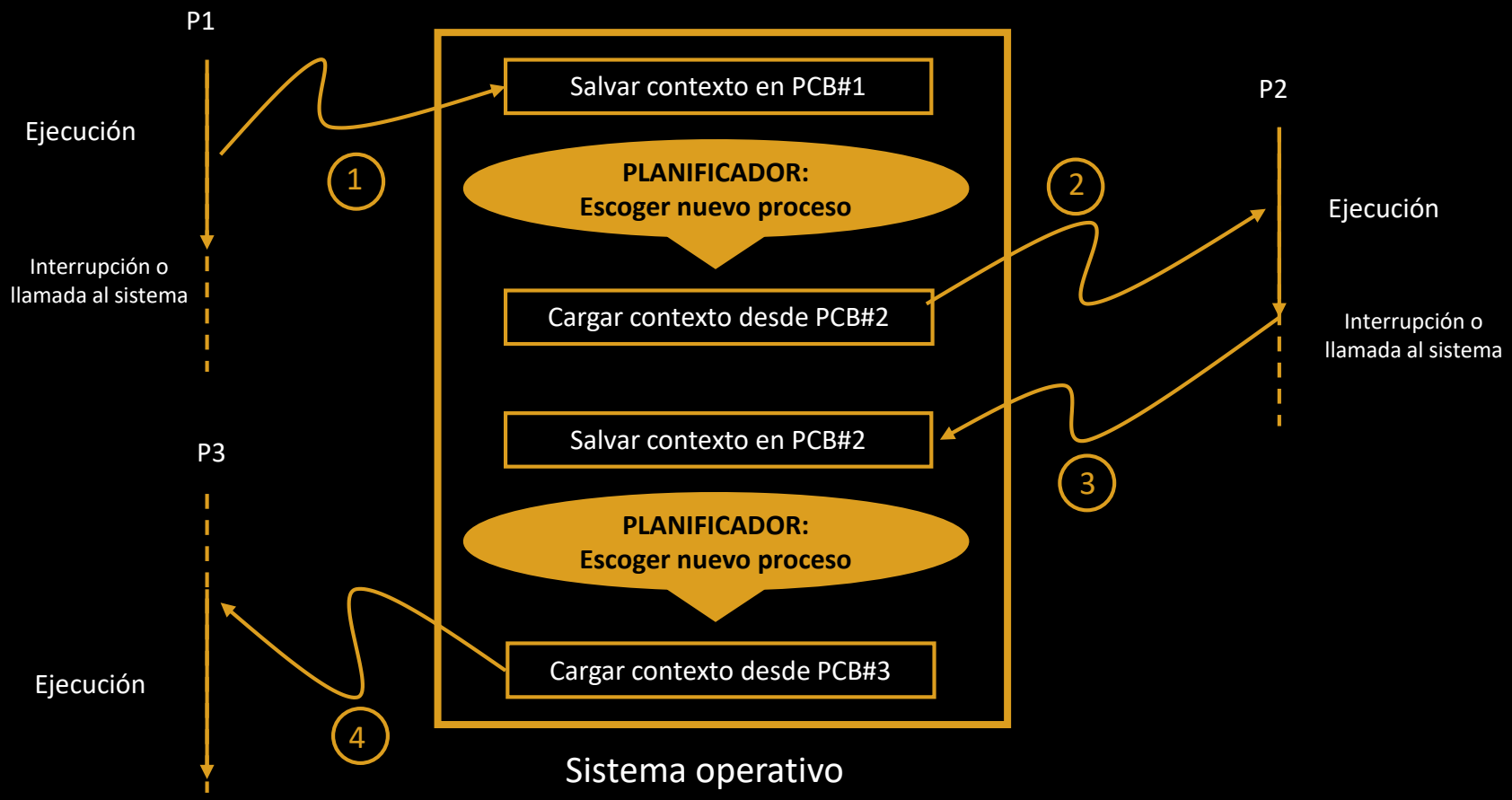
PLANIFICADOR

- Elemento del sistema operativo que determina a qué proceso se le asigna un determinado recurso (como la CPU) en cada instante de tiempo, de acuerdo con alguna política.
  - Procesos orientados a CPU u orientados a E/S:
    - Un proceso **orientado a CPU** es aquel que invierte la mayor parte de su tiempo en efectuar **cálculos** y genera solicitudes de E/S con poca frecuencia.

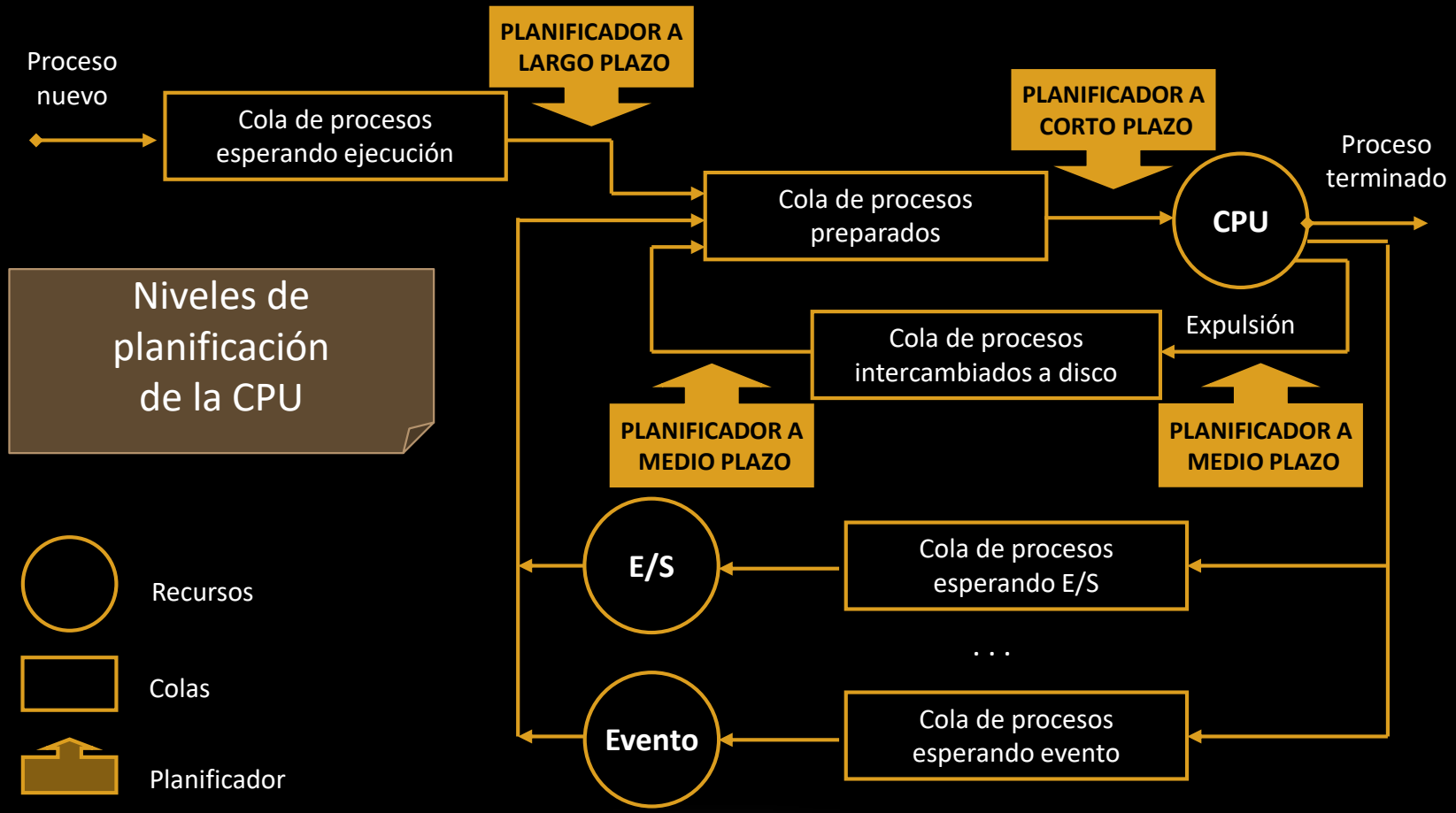
    - Un proceso **orientado a E/S** es aquel que emplea más tiempo en realizar **operaciones de E/S** que en efectuar cálculos.

  - En el caso de que el recurso a asignar sea la CPU se distinguen tres planificadores:
    - **Planificador a corto plazo.**
    - **Planificador a medio plazo.**
    - **Planificador a largo plazo.**

PLANIFICADOR EN FUNCIONAMIENTO



NIVELES DE PLANIFICACIÓN



PLANIFICACIÓN A LARGO PLAZO

- **También denominada admisión.**
- En un sistema de proceso por lotes, los procesos recién incorporados permanecen detenidos en una cola de procesamiento por lotes, en el disco. El planificador a largo plazo **creará procesos** a partir de la cola cuando sea posible.
- Dos son las decisiones que toma el planificador a largo plazo:
  - **Cuándo crear un nuevo proceso:** controla el grado de multiprogramación.
  - **Cuál va a ser el siguiente proceso a admitir:** usando un algoritmo FCFS (*first come, first served*), teniendo en cuenta prioridades, tiempos de ejecución esperados, exigencias E/S…
- **Grado de multiprogramación:** conjunto de procesos que residen simultáneamente en memoria y se ejecutan concurrentemente.
- Selecciona procesos de la cola de procesos que están esperando ser ejecutados y los carga en memoria.
- Se ejecuta con poca frecuencia, ya que pueden transcurrir minutos entre la creación de nuevos procesos en el sistema.

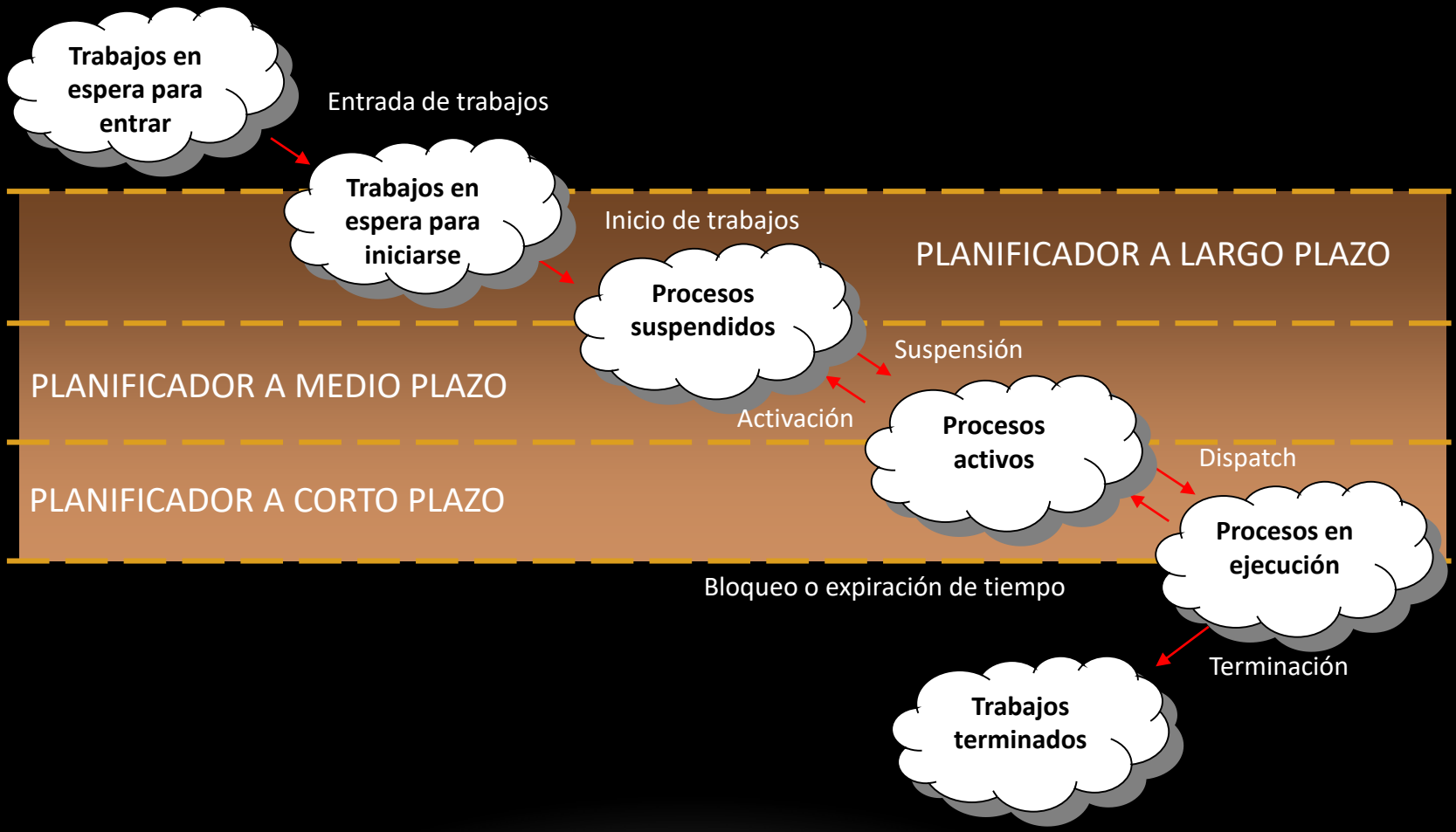
PLANIFICACIÓN A MEDIO PLAZO

- **También denominada *swapping*.**
- En ocasiones es interesante sacar procesos de memoria para reducir el grado de multiprogramación o para modificar la proporción entre procesos de los dos tipos (orientados a CPU o E/S).
- Estos procesos son movidos al espacio de intercambio (*swap* o memoria virtual).
- Se encarga de controlar qué procesos, de entre todos los iniciados deben estar en memoria y cuáles deben estar en el espacio de intercambio.
- El planificador a medio plazo puede sacar procesos de memoria y volverlos a introducir más adelante. El proceso continuará su ejecución a partir del punto donde se había quedado.

PLANIFICACIÓN A CORTO PLAZO

- **También denominada *dispatching*.**
- Selecciona un proceso de la cola de procesos preparados para su ejecución y le asigna la CPU.
- Se lleva a cabo con mucha frecuencia. El proceso seleccionado quizás se ejecute una únicamente durante unos milisegundos antes de iniciar una solicitud de E/S.
- Se ejecuta cuando ocurre un **evento** que conduce a la **interrupción** del proceso actual, expulsando el proceso a favor de otro.
- Ejemplos de eventos:
  - Interrupciones de reloj.
  - Interrupciones de E/S.
  - Llamadas al sistema operativo.
  - Señales.

VISIÓN GLOBAL DE LOS TRES NIVELES DE PLANIFICACIÓN



CRITERIOS DE PLANIFICACIÓN

CRITERIOS DE PLANIFICACIÓN

- Dependen del tipo de sistema.
- **Comunes:**
  - **Equidad** – cada proceso recibe la proporción justa de CPU.
  - **Prioridades** – cumplimiento de las políticas de prioridades.
  - **Balanceo (equilibrio de recursos)** – todos los recursos del sistema deben estar ocupados.
- **Sistemas por lotes:**
  - **Productividad** – maximizar el número de trabajos completados en la unidad de tiempo.
  - **Tiempo de retorno** – minimizar el tiempo entre la entrada y la salida.
  - **Utilización de CPU** – mantener la CPU ocupada todo el tiempo.
- **Sistemas interactivos:**
  - **Tiempo de respuesta** – minimizar el tiempo entre la recepción y el comienzo de los procesos.
  - **Proporcionalidad** – los procesos simples deben completarse rápidamente.
- **Sistemas de tiempo real:**
  - **Cumplir plazos** – maximizar el número de trabajos terminados a tiempo.
  - **Previsibilidad** – un proceso debe ejecutarse en el mismo tiempo y con el mismo coste independientemente de la carga del sistema.

MEDIDAS DE EFECTIVIDAD

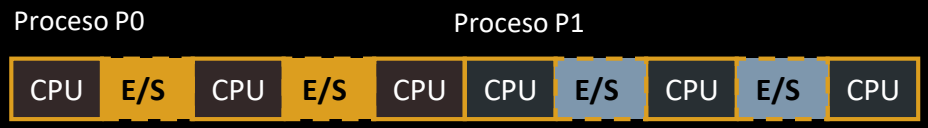
- Para determinar si se cumplen los criterios.
- **Utilización:** Los recursos se han de mantener tan ocupados como sea posible:
  - Tiempo de recurso ocupado / Tiempo total
- **Productividad:** Maximizar el número de tareas procesadas por unidad de tiempo:
  - Número de trabajos terminados / Tiempo total
- **Tiempo de retorno:** Tiempo que tarda en ejecutarse un proceso:
  - Tiempo de salida – Tiempo de entrada = $T_{CPU} + T_{E/S} + T_{Colas}$
- **Tiempo de espera:** Tiempo que un proceso está en la cola de procesos preparados.
- **Tiempo de respuesta:** Tiempo que transcurre desde que se presenta una solicitud hasta que el sistema comienza a contestar
- **Equidad:** Cada proceso obtiene la proporción justa de CPU. Es decir, que los procesos sean tratados de manera igualitaria. Lo opuesto a equidad sería inanición.

OPTIMIZACIÓN DE LOS CRITERIOS DE PLANIFICACIÓN

- No es posible optimizar todos los criterios a la vez puesto que algunos de ellos persiguen objetivos contrapuestos.
- Cada tipo de sistema tiene sus prioridades:
  - **Sistema por lotes:**
    - Maximizar: utilización y productividad.
    - Minimizar: tiempo de retorno y tiempo de espera.
  - **Sistemas interactivos:**
    - Proporcionar equidad.
    - Propiciar un tiempo de respuesta razonable y predecible.
- La **multiprogramación** en sí misma supone una mejora de muchos de los criterios anteriores respecto a la ejecución secuencial.
- Los **algoritmos de planificación** también tienen como objetivo mejorar algunos de los criterios mencionados.

EFECTO DE LA MULTIPROGRAMACIÓN

Sin multiprogramación

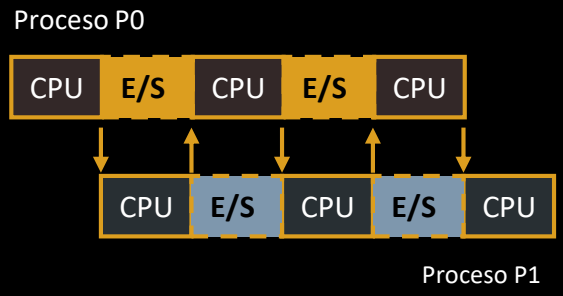


Utilización de CPU = 6/10 100% = 60%

Productividad = 2 trabajos / 10 = 0.2

Tiempo de retorno = (5+10) / 2 = 7.5

Con multiprogramación: (Intercalando ráfagas de CPU con ráfagas de E/S)

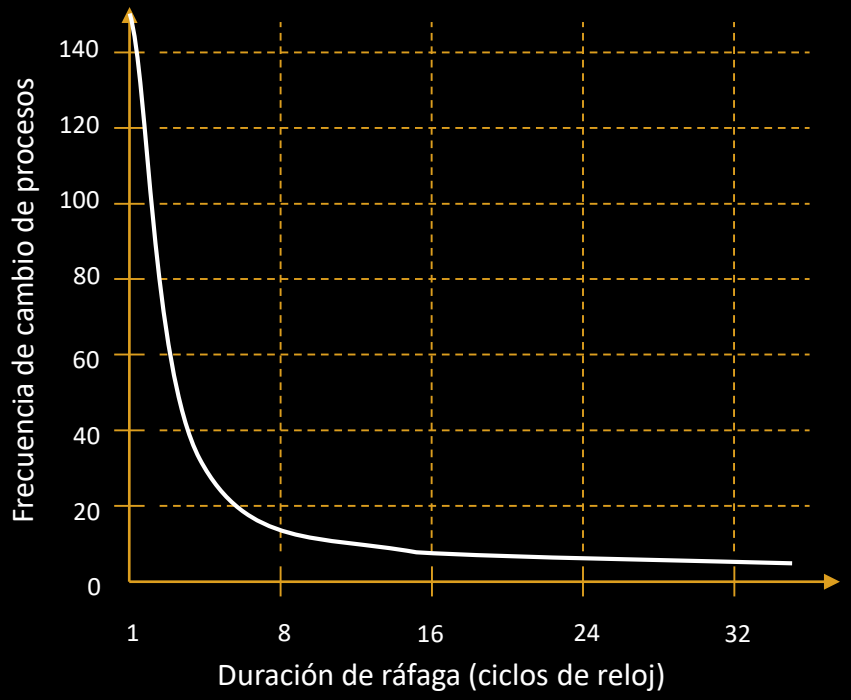


Utilización de CPU = 6/6 100% = 100%

Productividad = 2 trabajos / 6 = 0.33

Tiempo de retorno = (5+6) / 2 = 5.5

UTILIZACIÓN DE LA CPU



- Gran número de ráfagas de CPU de corta duración.
- Pequeño número de ráfagas de CPU de larga duración.

- Para mejorar la utilización de la CPU se puede:
  - Aumentar el grado de multiprogramación mientras la memoria lo permita.
  - Adoptar un algoritmo de planificación adecuado que optimice el orden de ejecución de los procesos.
- Para conseguir un algoritmo de planificación óptimo se puede:
  - Determinar qué parámetros se quieren optimizar.
  - Conocer el tipo de comportamiento de los procesos.

ALGORITMOS DE PLANIFICACIÓN

ALGORITMOS DE PLANIFICACIÓN

- **Objetivo:** Decidir a cuál de los procesos que están en la cola de procesos preparados se le asignará la CPU.
- **Clasificación de algoritmos de planificación:**
  - **Planificación sin expulsión:**
    - Una vez que la CPU ha sido asignada a un proceso, la mantiene hasta que termina o se bloquea.
    - Óptimo en el caso de proceso por lotes.
  - **Planificación con expulsión:**
    - La CPU puede ser liberada de un proceso en ejecución y asignarse a otro proceso.
    - Necesaria en sistemas interactivos o de tiempo real.

ALGORITMO FCFS (I)

- **Servicio por orden de llegada (FCFS: *first come, first served*):**

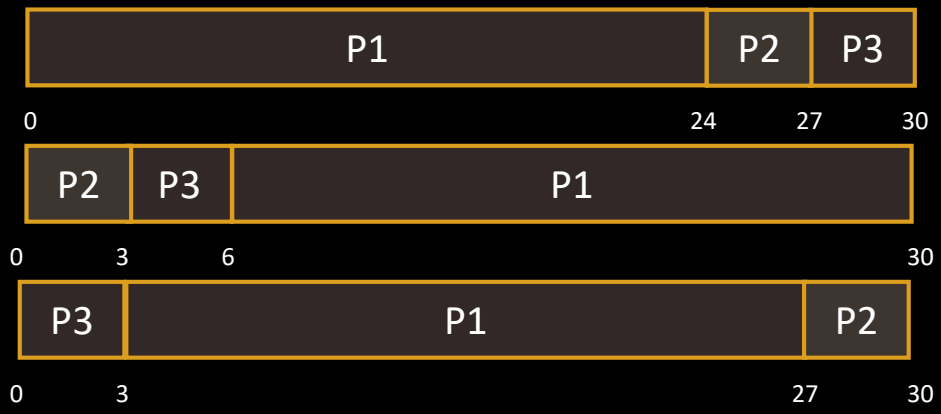
  La CPU es asignada a todos los procesos en el mismo orden que lo solicitan.

Proceso	$T_{llegada}$	T_{CPU}
P1	0	24
P2	0	3
P3	0	3

Caso 1: Orden de llegada P1, P2, P3
T de espera medio:
(0 + 24 + 27) / 3 = 17

Caso 2: Orden de llegada P2, P3, P1
T de espera medio:
(6 + 0 + 3) / 3 = 3

Caso 3: Orden de llegada P3, P1, P2
T de espera medio:
(3 + 27 + 0) / 3 = 10



ALGORITMO FCFS (II)

Proceso	$T_{llegada}$	T_{CPU}
P1	0	7
P2	2	4
P3	4	1
P4	5	4

T de espera medio: (0 + 5 + 7 + 7) / 4 = 4,75

ALGORITMO FCFS (III)

- La CPU es asignada a todos los procesos en el mismo orden que lo solicitan.
- **Propiedades:**
  - **Sin expulsión:** cuando un proceso tiene asignada la CPU, la conserva hasta que desee liberarla, bien porque finaliza o por una solicitud de E/S.
- **Ventajas:**
  - Fácil de implementar.
- **Inconvenientes:**
  - **No optimiza el tiempo de espera:** es muy variable en función del orden de llegada de los procesos y la duración de los intervalos de CPU.
  - **Efecto convoy:** los trabajos largos retrasan a los cortos (por ejemplo: un sistema con un único trabajo con largas ráfagas de CPU y muchos trabajos con ráfagas cortas de CPU).
  - **No es adecuado para sistemas interactivos:** por ser sin expulsión un trabajo con una ráfaga de CPU larga puede provocar una espera larga a otros usuarios.

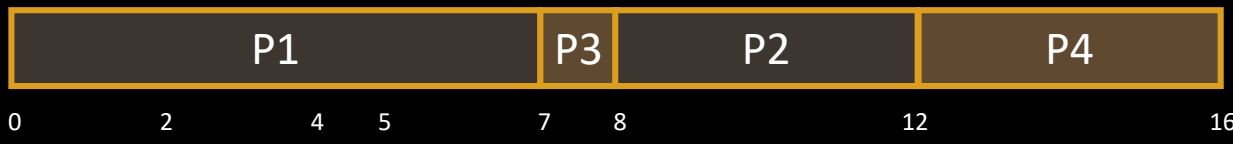
ALGORITMO SJF (I)

- **Prioridad al trabajo más breve (SJF: *shortest job first*, SPN *shortest process next*):**
- Se asocia a cada trabajo (proceso) un tiempo de ejecución estimado.
  - Especificado por el programador.
  - Obtenido de un histórico de estadísticas.
  - Estimado por la CPU.
  - …
- Se asigna la CPU al trabajo con menor tiempo de ejecución estimado.
- **Sin expulsión:**
  - Cuando un proceso tiene asignada la CPU, la conserva hasta que desee liberarla.

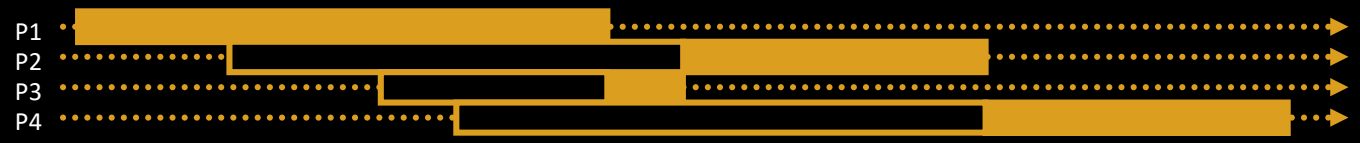
ALGORITMO SJF (II)

Proceso	$T_{llegada}$	T_{CPU}
P1	0	7
P2	2	4
P3	4	1
P4	5	4

SJF (sin expulsión)



Cronograma por procesos



Tiempo de espera

Tiempo de CPU

T de espera medio: (0 + 6 + 3 + 7) / 4 = 4

ALGORITMO SRTF (I)

- **Variante con expulsión del SJF (SRTF: *shortest remaining time first*):**

  Prioridad al que le falta menos tiempo para finalizar.

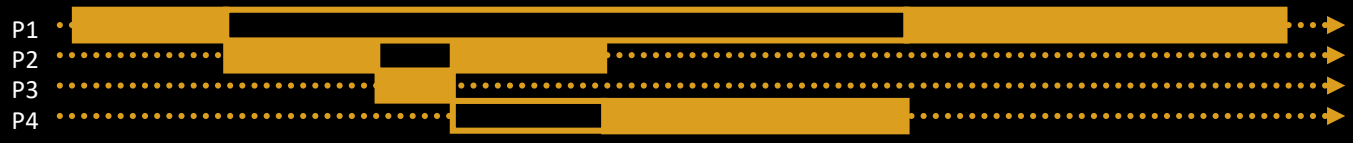
  - La CPU es asignada al proceso que le queda menos tiempo para acabar el intervalo de CPU en curso.
  - **Variante con expulsión de SJF:** si llega un proceso con un intervalo de CPU inferior al tiempo que le falta al proceso en ejecución para abandonar la CPU, entonces el nuevo proceso se hace con la CPU.
- **Ventajas:**
  - SRTF optimiza la media del tiempo de espera.
- **Inconvenientes:**
  - El tiempo del siguiente intervalo de CPU es difícil de predecir. Esto lo hace difícil de implementar para una planificador a corto plazo.
    - En los sistemas por lotes el usuario puede aportar información.
  - **Posibilidad de inanición:** los trabajos largos no se ejecutarán mientras haya trabajos cortos.

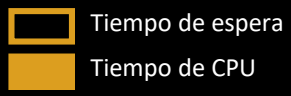
ALGORITMO SRTF (II)

Proceso	$T_{llegada}$	T_{CPU}
P1	0	7
P2	2	4
P3	4	1
P4	5	4

SRTF (SJF con expulsión)

Cronograma por procesos





T de espera medio: (9 + 1 + 0 + 2) / 4 = 3

PLANIFICACIÓN ALGORITMOS SJF/SRTF

- Aunque no se conoce la longitud de la siguiente ráfaga se puede predecir su valor esperando que sea de longitud similar a las anteriores.
- **Estimación del siguiente intervalo de CPU:**
  - Se utiliza la media exponencial: $S_{n+1} = \alpha T_n + (1- \alpha)S_n$
    - S_n = tamaño estimado del n-ésimo intervalo
    - T_n = tamaño real del n-ésimo intervalo
    - α = coeficiente exponencial ($0 \leq \alpha \leq 1$)
  - Caso $\alpha = 1$: los datos históricos son irrelevantes y sólo tiene importancia la ráfaga más reciente de CPU. ($S_{n+1} = T_n$)
  - Caso $\alpha = 0$: la historia reciente no tiene efecto, se supone que las condiciones actuales son transitorias. ($S_{n+1} = S_n$)
  - Habitualmente $\alpha = ½$, por lo que la historia reciente y antigua se ponderan de igual manera.

ESTIMACIÓN DEL SIGUIENTE INTERVALO DE CPU

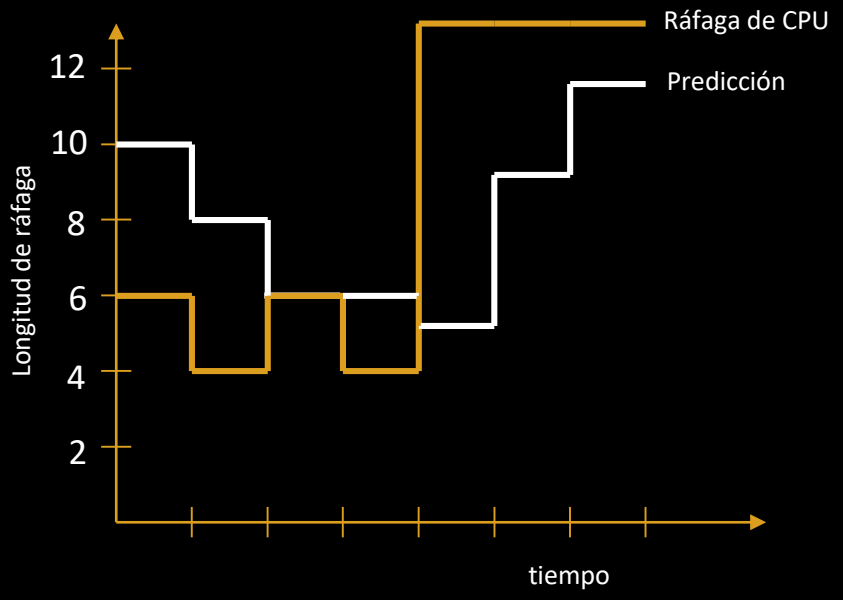
- Desarrollando la fórmula y sustituyendo sucesivamente S_n :

  $S_{n+1} = \alpha T_n + (1-\alpha)\alpha T_{n-1} + (1-\alpha)^2\alpha T_{n-2}+...+(1-\alpha)^j\alpha T_{n-j} + ... +(1-\alpha)^{n-1}\alpha T_1 + (1-\alpha)^n S_1$

- Puesto que tanto $(1-\alpha)$ como α son menores o iguales que 1, cada término sucesivo tiene menos peso que el anterior

Ejemplo de promedio exponencial para $\alpha = ½$ y $S_1 = 10$

Ráfaga de CPU:	6	4	6	4	13	13	13
Predicción:	10	8	6	6	5	9	11



ALGORITMO RR (I)

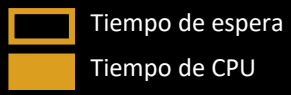
- **Planificación circular (RR: *round-robin*):**
  - A cada proceso se le asigna una pequeña cantidad de tiempo de CPU, llamada **quantum** de tiempo, normalmente 10ms - 100ms.
  - Si el proceso tiene un intervalo de CPU mayor que el quantum, entonces es expulsado de la CPU y añadido a la cola de procesos listos.
  - Si hay n procesos, cada uno de ellos obtiene 1/n del tiempo de la CPU en intervalos de q unidades, como máximo.
- **Valor del quantum de tiempo:**
  - **q grandes ($q \rightarrow \infty$):** este algoritmo degenera en un algoritmo FCFS.
  - **q pequeños:** ha de ser grande respecto al tiempo necesario para realizar los cambios de contexto, de lo contrario la sobrecarga introducida es muy alta.
  - **Regla práctica:** el 80% de los intervalos de CPU han de ser inferiores al quantum de tiempo.
- **Propiedades:**
  - Equitativo.
  - El tiempo de espera máximo está limitado por (n-1)q, antes de recibir su siguiente cuanto de tiempo.
  - El tiempo de retorno medio varía con el cuanto de tiempo.
  - En general es peor que el algoritmo SRTF. Mejora si un porcentaje alto de trabajos acaban antes de que acabe el cuanto de tiempo.

ALGORITMO RR (II)

Proceso	$T_{llegada}$	T_{CPU}
P1	0	7
P2	2	4
P3	4	1
P4	5	4

RR (q = 1)

Cronograma por procesos

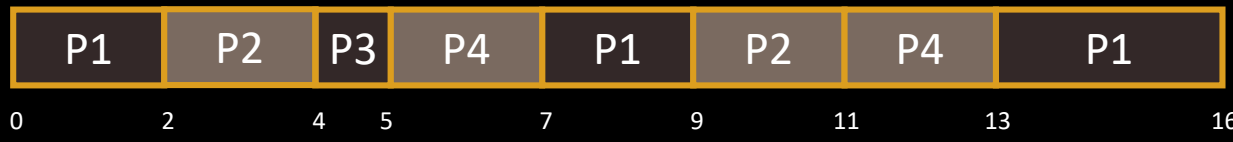


T de espera medio: (9 + 6 + 1 + 6) / 4 = 5,5

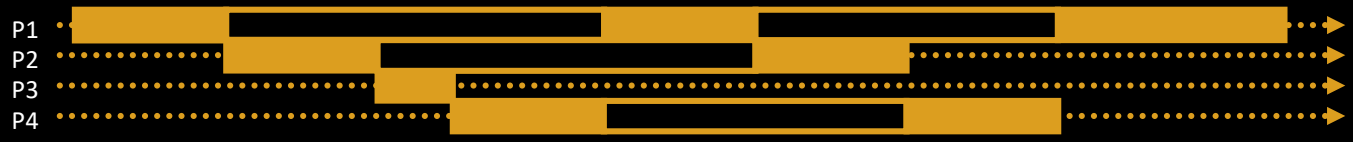
ALGORITMO RR (III)

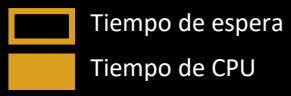
Proceso	$T_{llegada}$	T_{CPU}
P1	0	7
P2	2	4
P3	4	1
P4	5	4

RR (q = 2)



Cronograma por procesos



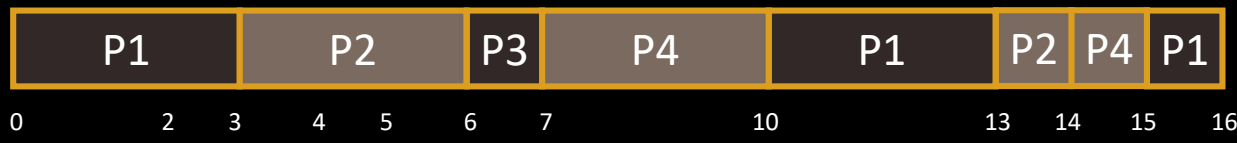


T de espera medio: (9 + 5 + 0 + 4) / 4 = 4,5

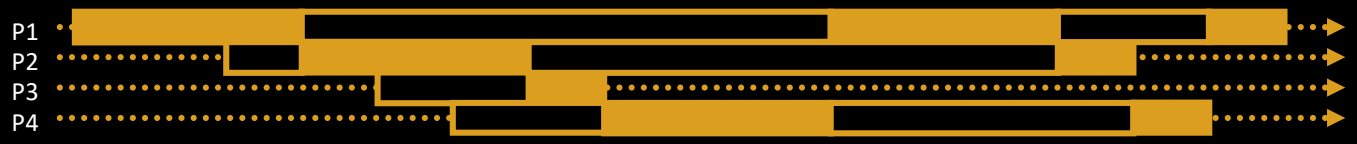
ALGORITMO RR (IV)

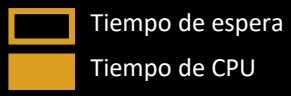
Proceso	$T_{llegada}$	T_{CPU}
P1	0	7
P2	2	4
P3	4	1
P4	5	4

RR (q = 3)



Cronograma por procesos



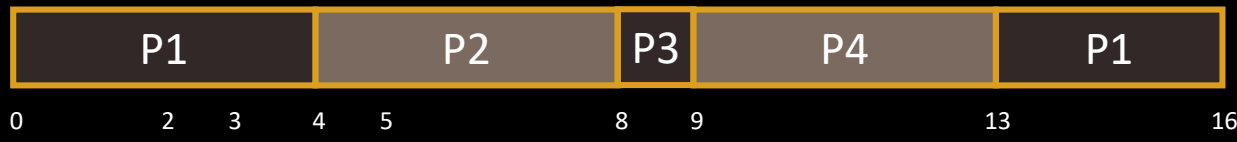


T de espera medio: (9 + 8 + 2 + 6) / 4 = 6,25

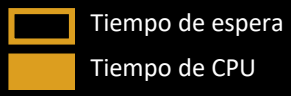
ALGORITMO RR (V)

Proceso	$T_{llegada}$	T_{CPU}
P1	0	7
P2	2	4
P3	4	1
P4	5	4

RR (q = 4)



Cronograma por procesos



T de espera medio: (9 + 2 + 4 + 4) / 4 = 4,75

PLANIFICACIÓN POR PRIORIDADES (I)

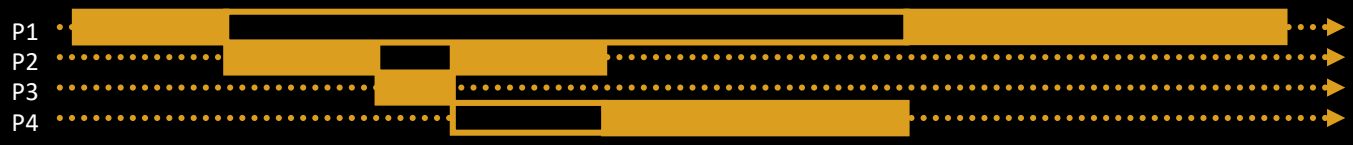
- Se asocia a cada proceso un número (entero), llamado prioridad, de acuerdo con algún criterio.
- Se asigna la CPU al trabajo con mayor prioridad: nº más bajo (Unix) o nº más alto (Windows).
  - SJF es un caso particular de prioridades en el que la prioridad es T (Unix) o 1/T (Windows).
- **Inconvenientes:**
  - Un algoritmo de prioridades es **poco equitativo**. El problema extremo es:
    - **Inanición:** los procesos con baja prioridad no se ejecutan nunca.
    - **Solución:** actualización de prioridades, esquema de prioridades dinámicas, donde la prioridad de un proceso aumenta con el tiempo.
  - **Inversión de prioridades:** Un proceso de baja prioridad puede hacer que un proceso de mayor prioridad no progrese bloqueando un recurso.
- **Variantes:**
  - Algoritmos con expulsión / sin expulsión.
  - Prioridades estáticas / dinámicas:
    - **Prioridades estáticas:** la prioridad se asigna antes de la ejecución y no cambia.
    - **Prioridades dinámicas:** la prioridad cambia con el tiempo.

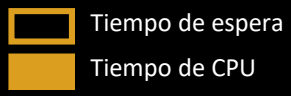
PLANIFICACIÓN POR PRIORIDADES (II) (TIPO WINDOWS)

Proceso	$T_{llegada}$	T_{CPU}	Prioridad
P1	0	7	5
P2	2	4	10
P3	4	1	15
P4	5	4	10

Planificación por prioridades

Cronograma por procesos





T de espera medio: (9 + 1 + 0 + 2) / 4 = 3

PLANIFICACIÓN MLQ

- **Planificación con múltiples colas (*Multi-Level Queue scheduling* MLQ):**

  Los procesos se pueden clasificar fácilmente en distintos grupos (interactivos, por lotes, etc.). La cola de procesos preparados consiste en realidad en diversas colas.

  - Cada cola ha de tener su propio algoritmo de planificación.
  - Ha de haber un algoritmo de planificación entre colas.

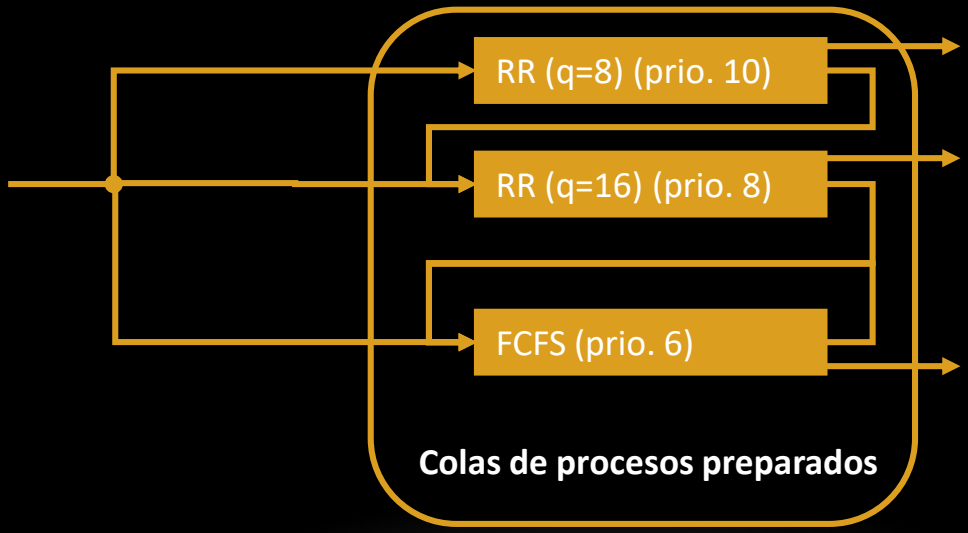
- **Ejemplo de algoritmo de planificación entre colas:**
  - **Prioridades estáticas:** Los procesos en cola con menor prioridad no se ejecutan mientras haya procesos en cola con mayor prioridad. Posibilidad de inanición.
  - **Cuotas de tiempo:** Cada cola está asignada a un porcentaje de tiempo de CPU.
    - Ejemplo: 80% a trabajos interactivos, 20% a trabajos por lotes.

MÚLTIPLES COLAS REALIMENTADAS (I)

- **Colas multinivel realimentadas (*Multi-Level Feedback Queue scheduling* MLFQ):**
- Existen diferentes colas de procesos preparados.
- Cada cola posee:
  - Política de planificación.
  - Una prioridad asignada.
- Un proceso puede cambiar de cola de acuerdo con un esquema de actualización de prioridades:
  - Los procesos con un tiempo de espera acumulado son **promocionados** a una cola con prioridad superior. (**envejecimiento**).
  - Los procesos con una utilización de la CPU elevado son **degradados** a una cola de prioridad inferior.

MÚLTIPLES COLAS REALIMENTADAS (II)

- Parámetros de las colas realimentadas:
  - Número de colas.
  - Prioridad de cada cola.
  - Método de promoción de cada cola.
  - Método de degradación de un proceso.
  - Método para determinar la cola de entrada de un proceso.



ALGORITMO HRRN

- **Mayor tasa de respuesta (*Highest Response Ratio Next* HRRN):**
  - Es necesario estimar el tiempo de servicio de igual forma que ocurría en SRTF o SJF.
  - Intenta hacer competitivos a los procesos largos.
  - Es necesario evaluar la **tasa de respuesta:** tiempo de servicio esperado mas tiempo de espera dividido por el tiempo de servicio esperado:
    - R = (w + s) / s
  - Se toma la R mayor.
  - Si s es pequeño la razón es grande: se favorecen los procesos cortos.
  - Si w es grande la razón es grande: se favorecen los procesos largos que llevan esperando bastante tiempo sin ejecutarse (envejecidos).
  - Normalmente **sin expulsión**.

ALGORITMO FSS (I)

- **Planificación por reparto equitativo (*Fair Share Scheduling* FSS):**
  - Agrupar procesos por usuarios y grupos o departamentos.
  - Asegurar que cada grupo recibe una parte proporcional de CPU
    - Se asigna una tasa de CPU a cada grupo.
    - Esas partes no tienen por qué ser iguales (¡a veces se compran!).
  - La prioridad de un proceso depende de su propia prioridad y del histórico de la totalidad de su grupo.

ALGORITMO FSS (II)

- **Cálculo de la prioridad en FSS:**
- $CPU_j(i)$: Utilización de la CPU por el proceso j en el intervalo i.
  - CPU(i) se incrementa en 1 por cada ciclo de CPU que use el proceso.
  - Al inicio de cada intervalo se recalcula CPU(i): CPU(i)=CPU(i-1)/2
- $GCPU_k(i)$: Utilización de la CPU por el grupo k en el intervalo i.
  - GCPU(i) se incrementa en 1 por cada ciclo de CPU que use un proceso del grupo.
  - Al inicio de cada intervalo se recalcula GCPU(i): GCPU(i)=GCPU(i-1)/2
- $P_j(i)$: Prioridad del proceso j al comienzo del intervalo i.
  - Un número menor equivale a una prioridad mejor.
  - Al inicio de cada intervalo se recalcula $P_j(i)$: $P_j(i) = Base_j + CPU_j(i)/2 + GCPU_k(i)/4*W_k$
  - $Base_j$ es la prioridad base del proceso j.
  - W_k es la parte proporcional de cada grupo. $0 < W_k < 1$ y $\Sigma W_k = 1$
  - Se usan los CPU(i) y GCPU(i) que se han recalculado al inicio del intervalo.

EVALUACIÓN DE ALGORITMOS

- **Etapas en la selección de un algoritmo:**
  1. **Selección de criterios** (utilización de CPU, minimizar tiempos de espera, respuesta, …).
  2. **Evaluar la adaptación** de cada algoritmo a esos criterios.
  - **Evaluación analítica:** dada la carga de trabajos intentar producir una fórmula matemática del rendimiento para cada algoritmo de planificación.
    - **Modelo determinista:** evaluar el rendimiento para cada caso particular de carga.
    - **Modelo de colas:** caracterizar la carga con una distribución estadística:
      - Caracterizar la tasa de llegada de trabajos.
      - Caracterizar el tiempo de servicio de un recurso.
      - Hacer análisis estadístico de las redes de colas:
        - Ejemplo: Fórmula de Little $n = \tau E$, dónde n es el tamaño medio de la cola, τ la tasa media de llegada y E es el tiempo medio de espera en la cola.
  - **Simulación:** Hacer un modelo, programarlo y ejecutarlo.