Sistemas Operativos

- Planificadores de CPU

  Como vimos, en un SO multiprograma varios procesos o threads listos pueden competir por la CPU.

- Politicas
  - FCFS
  - RR
  - PRIORIDAD
  - SJF

Definición (Planificador)

El planificador (scheduler) es la parte del SO que decide a qué proceso preparado se le da paso a CPU.

- Funcion
  - Asignar unidades de ejecucion al procesador/es
- Objetivo
  - Uso eficiente / poca sobrecarga
  - Equidad / no inanicion
  - Muchos otros en base a fines especificos

Definición (Planificador)

- Existen distintos algoritmos de planificación (scheduling algorithms).
- Planificación no apropiativa (non-preemptive): deja ejecutar al proceso en CPU hasta que éste se detiene, por bloqueo (inicio E/S), espera por otro proceso o terminación voluntaria.
- Planificación apropiativa: el planificador puede desalojar al proceso en CPU durante su ejecución y cambiarlo por otro. Necesita una interrupción de reloj para poder ejecutarse en períodos regulares de tiempo (quantum).

Objetivos del planificador

En la mayoria de los entornos:

- **Justicia (fairness):** que el proceso obtenga una porción de CPU “justa” o razonable.
- **Política:** que se satisfaga un determinado criterio establecido (ej.prioridades).
- **Equilibrio:** que las partes del sistema estén ocupadas haciendo algo.
- **Productividad o rendimiento (throughput) :** número de trabajos / unidad de tiempo. Maximizar
- **Tiempo de paso o de retorno (turnaround):** tiempo transcurrido entre que se lanza un proceso y termina. Minimizar.
- **Capacidad de ejecución:** mantener la CPU ocupada el mayor tiempo posible.

Principales Metricas Cuanticas

- Alto Rendimiento
  - Numero de trabajos completados por unidad de tiempo
- Alta utilizacion del Procesador
  - Porcentaje de tiempo que el procesador esta ocupado
- Tiempo de retorno
  - Tiempo transcurrido desde que nace un trabajo, hasta su finalizacion
- Tiempo de respuesta
  - Tiempo transcurrido desde el envio de un trabajo, hasta que se inicia su ejecucion

Medidas de tiempo

- **Tiempo de retorno (turnaround) (tR):** el total transcurrido desde que se inicia (Ti) hasta que finaliza (Tf).

$$TR = Tf - Ti$$

Incluye:

- Tiempo de carga en memoria
- Tiempo en la cola de preparados
- Tiempo de ejecución en CPU tCPU
- Tiempo en operaciones E/S (bloqueado) tE/S

Medidas de tiempo

- **Tiempo de espera (tE):** es el tiempo de retorno quitando CPU y E/S.

$$TE = tR - tCPU - tE/S$$

- **Tiempo de servicio (tS) :** Es el tiempo que consumiría si fuese el único proceso existente (y no precisase carga). Es decir, el tiempo de retorno menos el tiempo de espera.

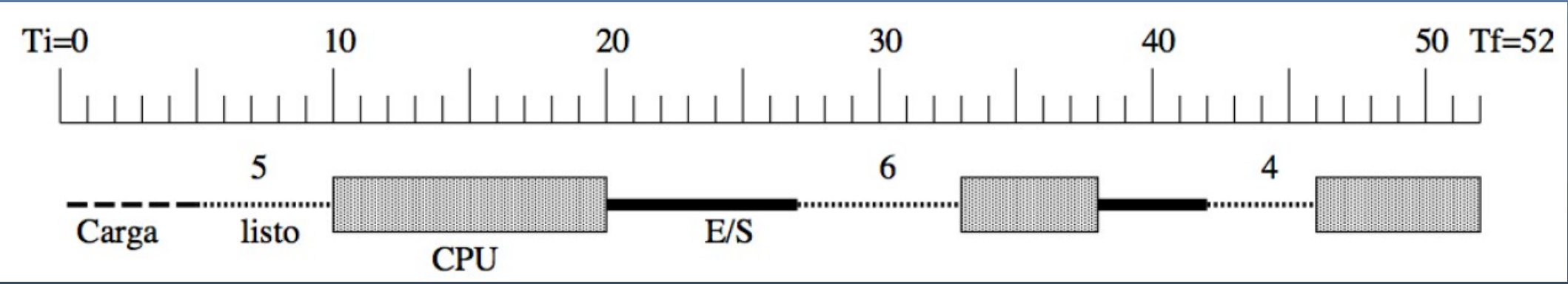
$$TS = tR - tE = tCPU + tE/S$$

- **Índice de servicio (i):**

$$iS = tS/tR$$

Medidas de tiempo

- Un ejemplo . . .



- Tiempo de retorno: tR = Tf - Ti = 52 - 0 = 52
- Tiempo de CPU: tCPU = 10 + 5 + 6 = 21
- Tiempo de E/S: tE=S = 7 + 4 = 11
- Tiempo de servicio: tS = tCPU + tE/S = 32
- Tiempo de espera: tE = tR - tS = 52 - 32 = 20
- Tiempo de índice de servicio: iS = 32/52 = 0;615

Evaluación de la planificación

Modelos deterministas

- Tomamos una carga de trabajo concreta y evaluamos los algoritmos sobre ella. Importante: seleccionar casos representativos.
- Comparamos los algoritmos en función de alguna de las medidas de rendimiento (ej. tiempo medio de retorno, productividad, etc).
- Ventajas: sencilla. Proporciona medidas exactas.
- Desventaja: engañosa si la carga de trabajo no es representativa.

Evaluación de la planificación

Modelos no deterministas (teoría de colas)

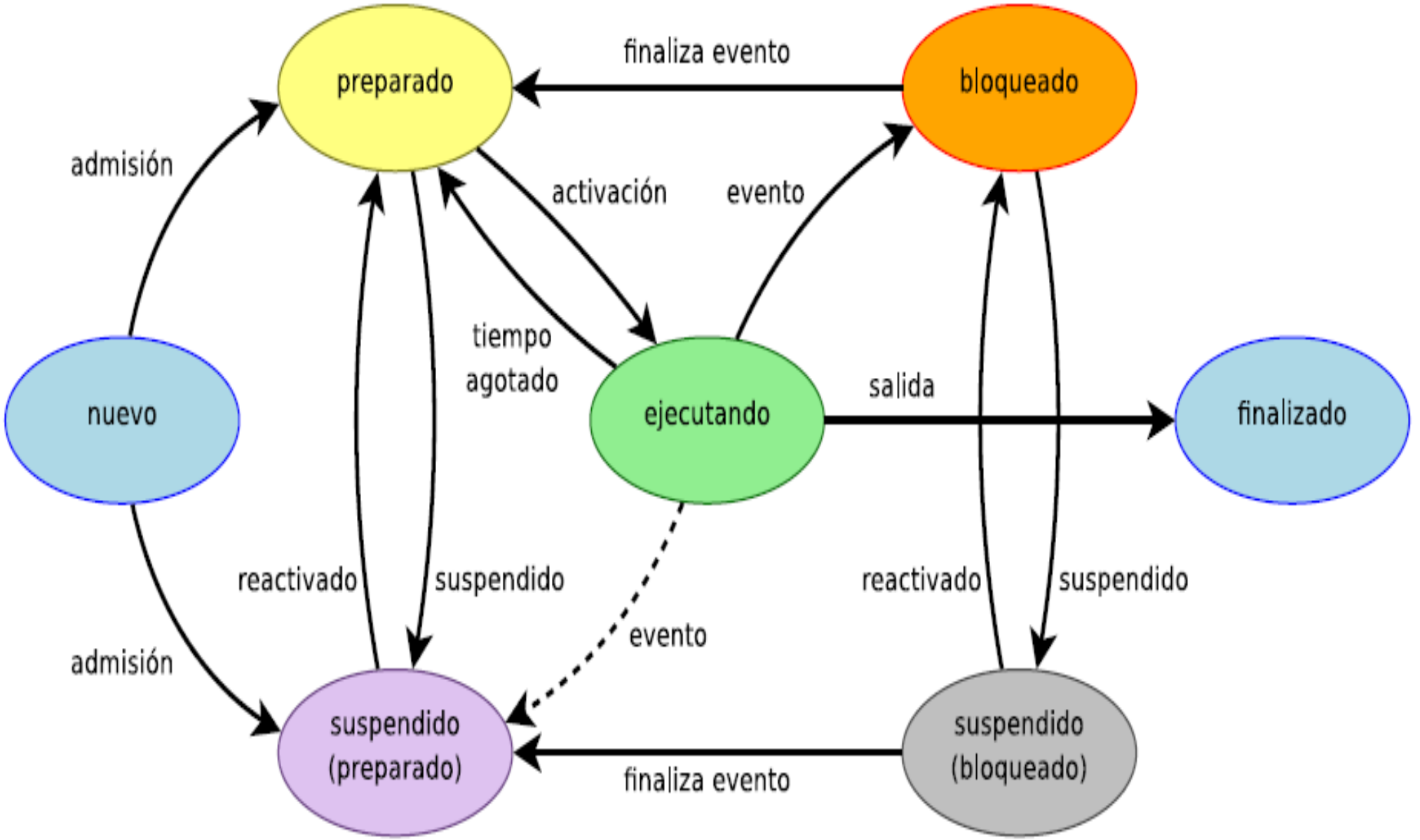
- El sistema informático se describe como una red de “servidores”.Cada servidor tiene una cola de trabajos en espera. La CPU es un servidor de su cola de preparados, así como el sistema de E/S lo es de su cola de dispositivo.
- Si conocemos los ritmos de llegada y de servicio, podemos calcular la utilización, la longitud media de cola, el tiempo de espera medio, etc. Esto se conoce como análisis de redes de colas.

Evaluación de la planificación

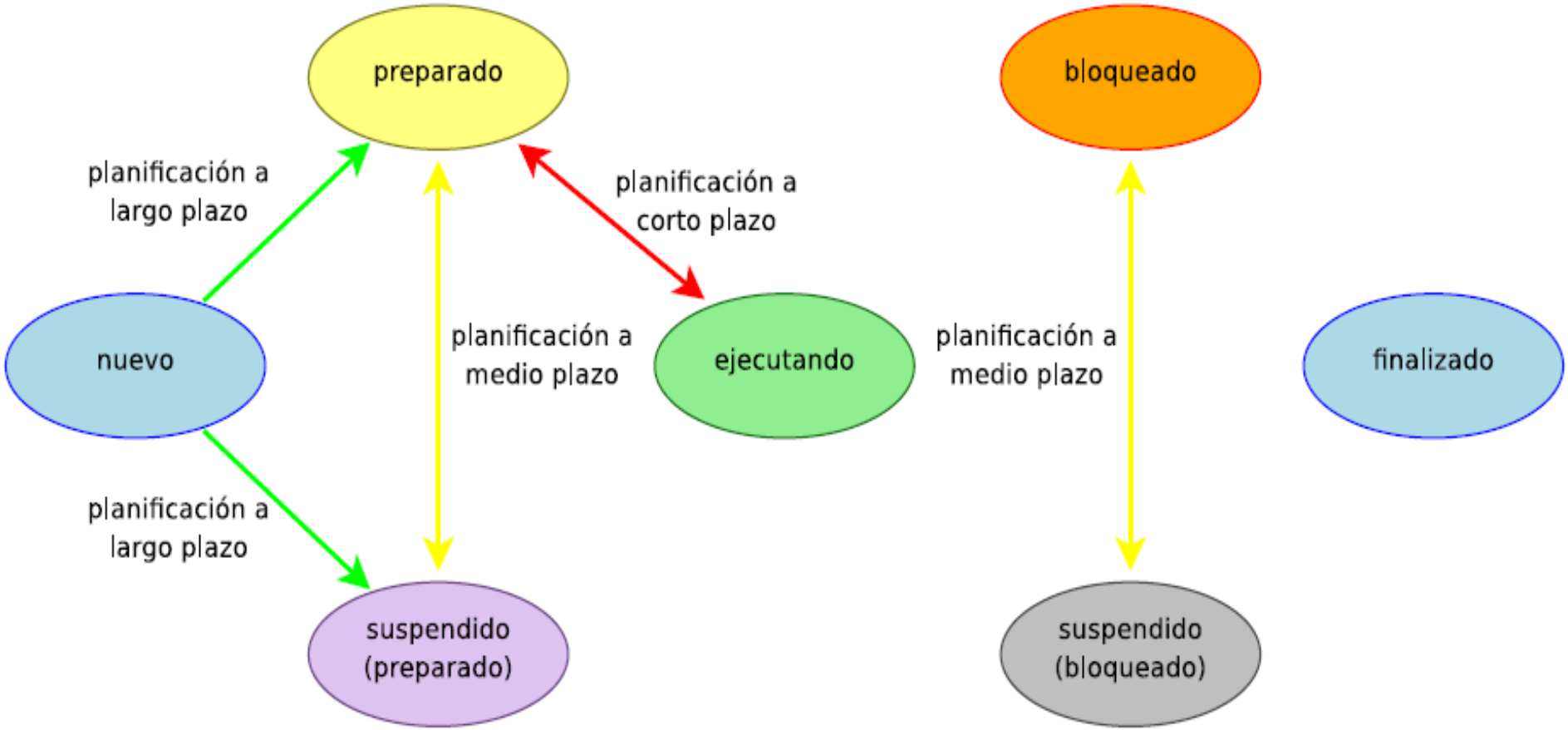
Simulación

- Una tercera opción es realizar simulaciones del comportamiento del sistema.
- Los datos de procesos y ráfagas se generan aleatoriamente o se obtienen de trazas reales.
- Permiten una evaluación cercana a casos reales.
- Sin embargo tienen alto coste (obtención de datos, tiempo de simulación, mediciones, etc).

Estado de un proceso



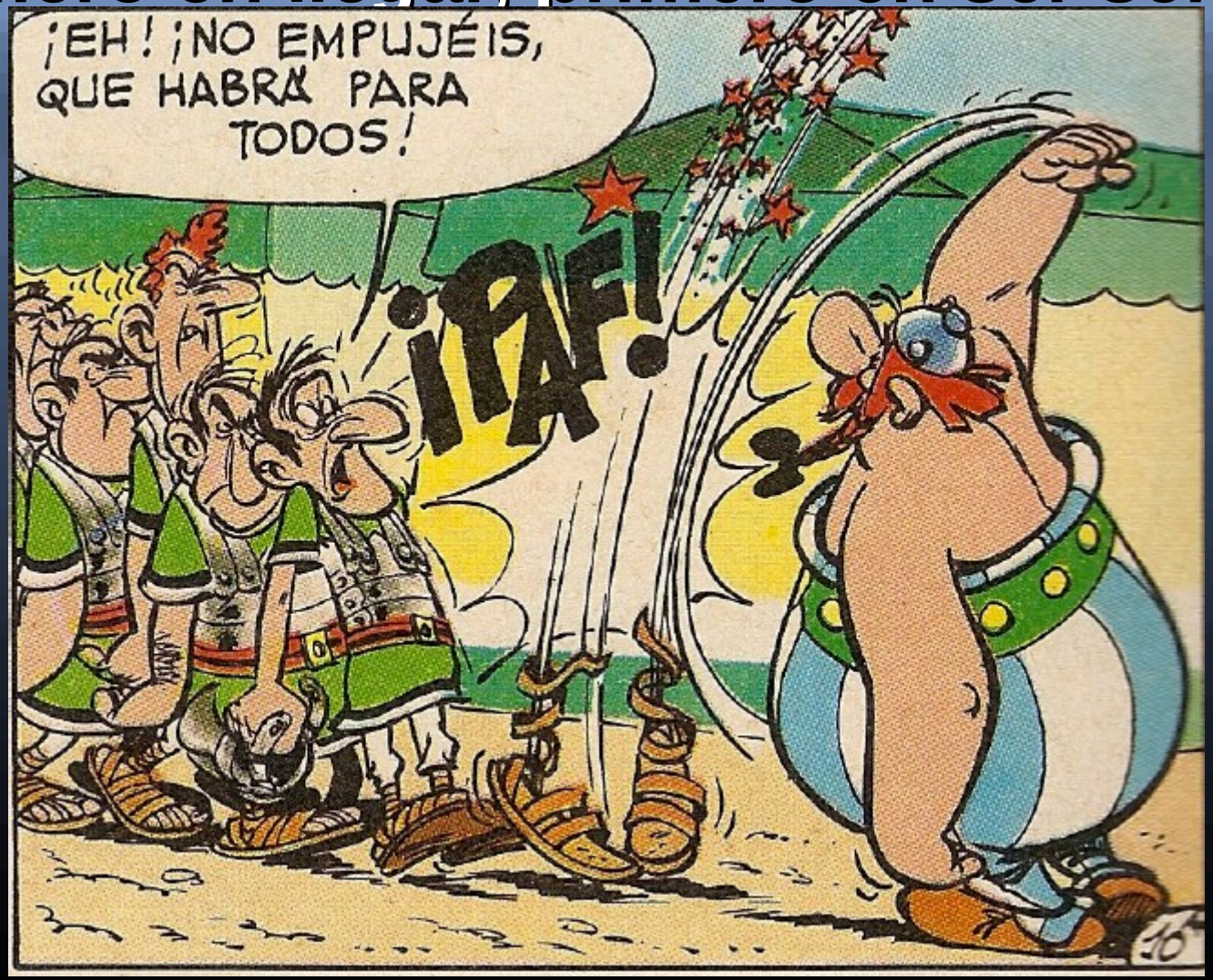
Tipos de Planificacion



- largo plazo: que procesos **admitir** al sistema $\longrightarrow$ [s - min].
- medio plazo: que procesos **intercambiar** $\longrightarrow$ [ms - s].
- corto plazo: que proceso **ejecutar** $\longrightarrow$ [μs - ms].

Algoritmos no apropiativos

Algoritmo First-Come-First-Served (FCFS): Primero en llegar, primero en ser servido



Algoritmos no apropiativos

Ventajas:

- Fácil de implementar. Basta una cola FIFO.
- Es bastante justo, si entendemos que procesos con menos CPU tienen menos derecho a usarla.

Desventaja:

- Puede provocar baja productividad; efecto “convoy”.

Algoritmos no apropiativos

Algoritmo Shortest Job First (SJF): Primero el más corto

- Tiene sólo utilidad teórica, ya que precisa conocer el tiempo que va a usarse la CPU antes de usarla.
- Es el óptimo para minimizar el tiempo de paso o de retorno (turnaround) con varios procesos listos en llegada simultánea
- A tiempos iguales, se usa FCFS.

Algoritmos no apropiativos

Algoritmo Shortest Process Next

- En la práctica SJF se modifica usando una estimación de las siguiente ráfaga de CPU en función de las anteriores

$$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$$

donde:

$$\begin{aligned} \tau_{n+1} &= \text{valor de la estimación} \\ t_n &= \text{última ráfaga} \\ \tau_n &= \text{valor anterior de la estimación} \\ \alpha &\in [0, 1] \text{ factor de ajuste} \end{aligned}$$

Algoritmos Apropiativos

Prioridades-Definición

La prioridad de un proceso es un valor numérico que se usa como factor para determinar si debe entrar en CPU antes que otro(s).

Tipos de prioridades:

- **Internas:** asignadas por el S.O. a partir de información de los procesos. Ej: tiempo en CPU, uso de memoria, ficheros abiertos, relación entre ráfagas CPU y E/S, etc.
- **Externas:** asignadas por S.O. (privilegios del usuario) o incluso por preferencias del propietario (ej: comando nice en UNIX).

Algoritmos Apropiativos

Prioridades

- **Principal desventaja:** inanición (starvation).Un proceso queda siempre esperando.
- Se suele resolver mediante asignación dinámica de prioridades.

  Dos ejemplos:

  - Usar como prioridad la fracción q=tCPU donde tCPU fue la última ráfaga.
  - Envejecimiento (aging): cuanto más tiempo CPU consume va disminuyendo su prioridad.

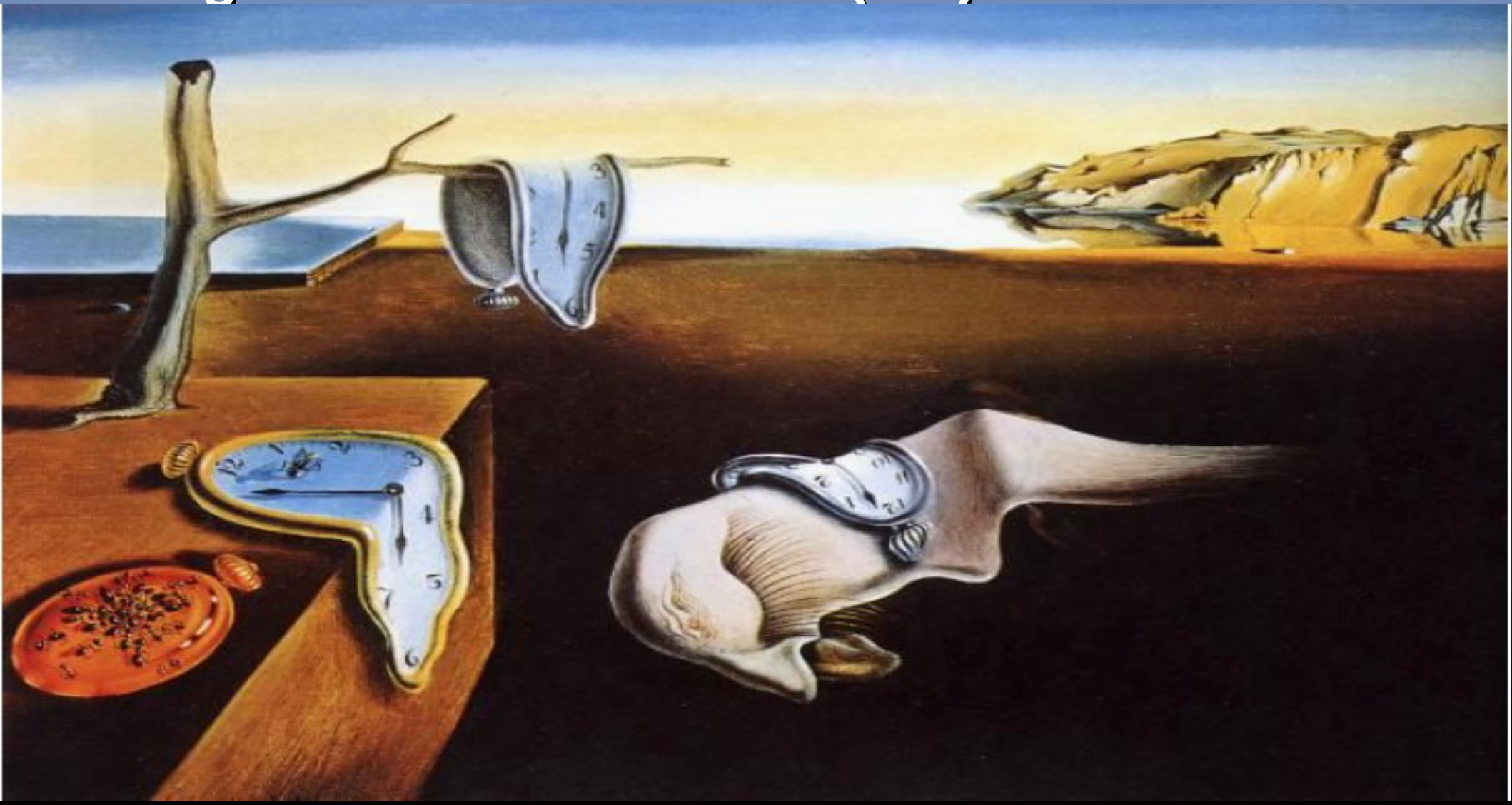
Algoritmos Apropiativos

Algoritmo Shortest Remaining Time First (SRTF)

- Es una versión apropiativa de SJF
- Cada vez que entran trabajos se interrumpe el actual y se compara el tiempo restante de éste con el de los entrantes.
- Si hay un trabajo nuevo más corto que lo que le falta al actual en CPU, echamos el actual y metemos el nuevo.
- De nuevo, se supone que se conocen los tiempos de uso futuro de CPU de antemano. Una versión práctica debe hacer uso de una estimación.

Algoritmos Apropiativos

Algoritmo Round-Robin (RR)



Algoritmos Apropiativos

Algoritmo Round-Robin (RR)

- Podemos traducirlo como asignación circular o por torneo
- Cada proceso tiene un tiempo límite de uso de CPU llamado quantum q.
- Los procesos preparados se organizan en una cola FIFO
- Si A está ejecutando y alcanza el quantum ==> cambio de contexto: se pasa el primero de la cola a CPU y se inserta A al final de la cola.
- Un temporizador (interrupción de reloj) se encarga de despertar al planificador para que compruebe si debe actuar o no.

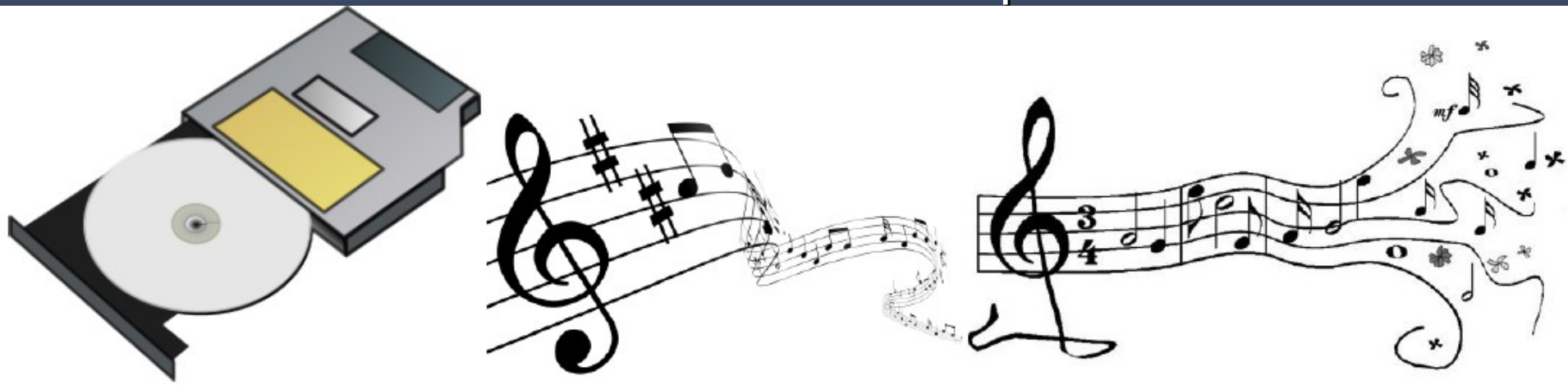
Algoritmos Apropiativos

Algoritmo Round-Robin (RR)

- Ventajas: es fácil de implementar. Es el algoritmo más justo: todos los procesos tienen garantizada su porción de CPU.
- Problema: fijar el valor de q es crítico
  - Si q muy pequeño, provoca muchos cambios de contexto. P.ej. si el cambio tarda ej. 1 ms, y las ráfagas son p.ej. de 4 ms un 20% del tiempo se desperdicia en tareas S.O.
  - Si q demasiado grande, degenera en un FCFS
- Empíricamente, da mejor resultado cuando un 80% de las ráfagas son más cortas que q. Valor habitual 20 ms <= q =< 50 ms.

Planificación en Sistemas Tiempo Real

- En sistemas de tiempo real (STR) el tiempo juega un papel crucial.
- Uno o más dispositivos físicos generan estímulos y el ordenador debe reaccionar a ellos dentro de un tiempo limitado.
- Ejemplo: un reproductor de CD toma información del disco y debe irla convirtiendo en sonido al ritmo preciso.



- Si no se atiende debidamente: pierde calidad o suena raro.

Planificación en Sistemas Tiempo Real

- Tiempo real estricto: el plazo de tiempo límite es obligatorio
- Tiempo real no estricto: perder un plazo límite es indeseable, pero a veces tolerable.
- Un programa se suele dividir en distintos procesos cortos y predecibles cuya duración se conoce de antemano.
- El planificador debe organizar los procesos de modo que se cumplan los plazos límite.
- En un STR podemos tener eventos:
  - Periódicos: suceden a intervalos regulares
  - Aperiódicos: suceden de forma impredecible.
- Un STR puede tener que responder a varios flujos (streams) de eventos periódicos. Si cada evento requiere mucho tiempo, puede incluso ser inmanejable.
- Supongamos 1;...;m flujos de eventos periódicos, y en cada flujo i:

  Pi = Período de tiempo con que sucede un evento

  Ci = Tiempo CPU que cuesta atender un evento

Planificación en Sistemas Tiempo Real

Definición (STR Planificable)

Decimos que un STR con m flujos es planificable si satisface:

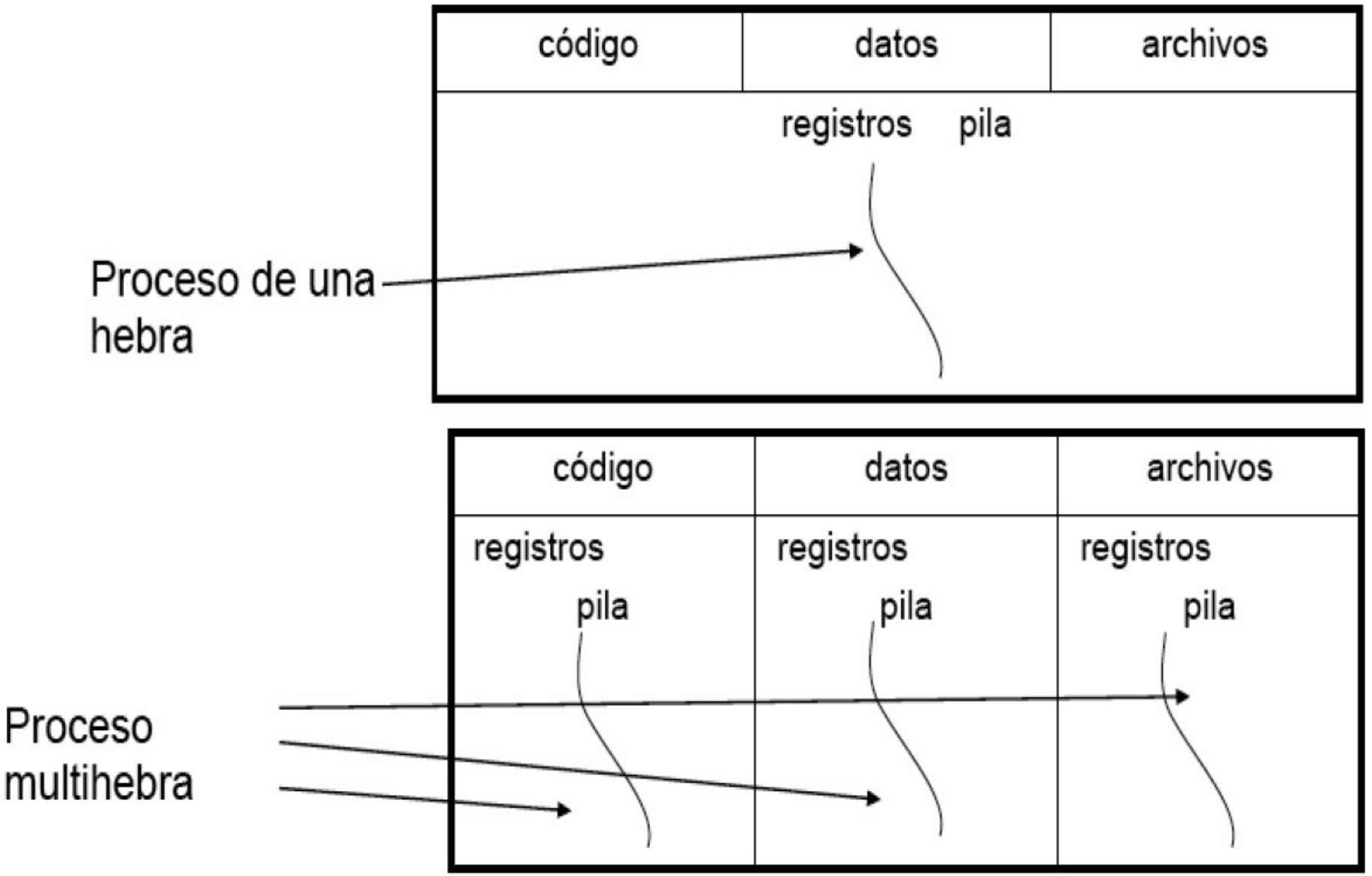
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

- Ejemplo: 3 flujos con $P_1 = 100$, $P_2 = 200$ y $P_3 = 500$ y consumos de CPU de $C_1 = 50$, $C_2 = 30$ y $C_3 = 100$ todo en ms.
- La suma da $0,5 + 0,15 + 0,2 < 1$.
- Si añadimos un cuarto flujo con $P_4 = 1000$ ¿cuánto podría valer C_4 como máximo para seguir siendo planificable?

Planificación con Hebras

- La hebra (thread) se puede definir como la unidad básica de utilización de CPU.
- Un proceso tiene como mínimo una hebra. Si tiene varias, puede realizar varias tareas concurrentemente.
- Las hebras de un proceso comparten: segmento de código, segmento de datos, recursos (archivos abiertos, señales, etc).
- Por cada hebra tenemos: identificador, contador de programa, registros, pila.

Planificación con Hebras



Planificación con Hebras

Ventajas

- Mayor capacidad de respuesta: si una hebra se bloquea, las demás pueden seguir ejecutándose.
- Puede haber varias hebras compartiendo los mismos recursos (memoria, ficheros, etc).
- Menos costoso que crear procesos, el cambio de contexto también es más ligero.
- Pueden aprovechar arquitecturas multiprocesador.

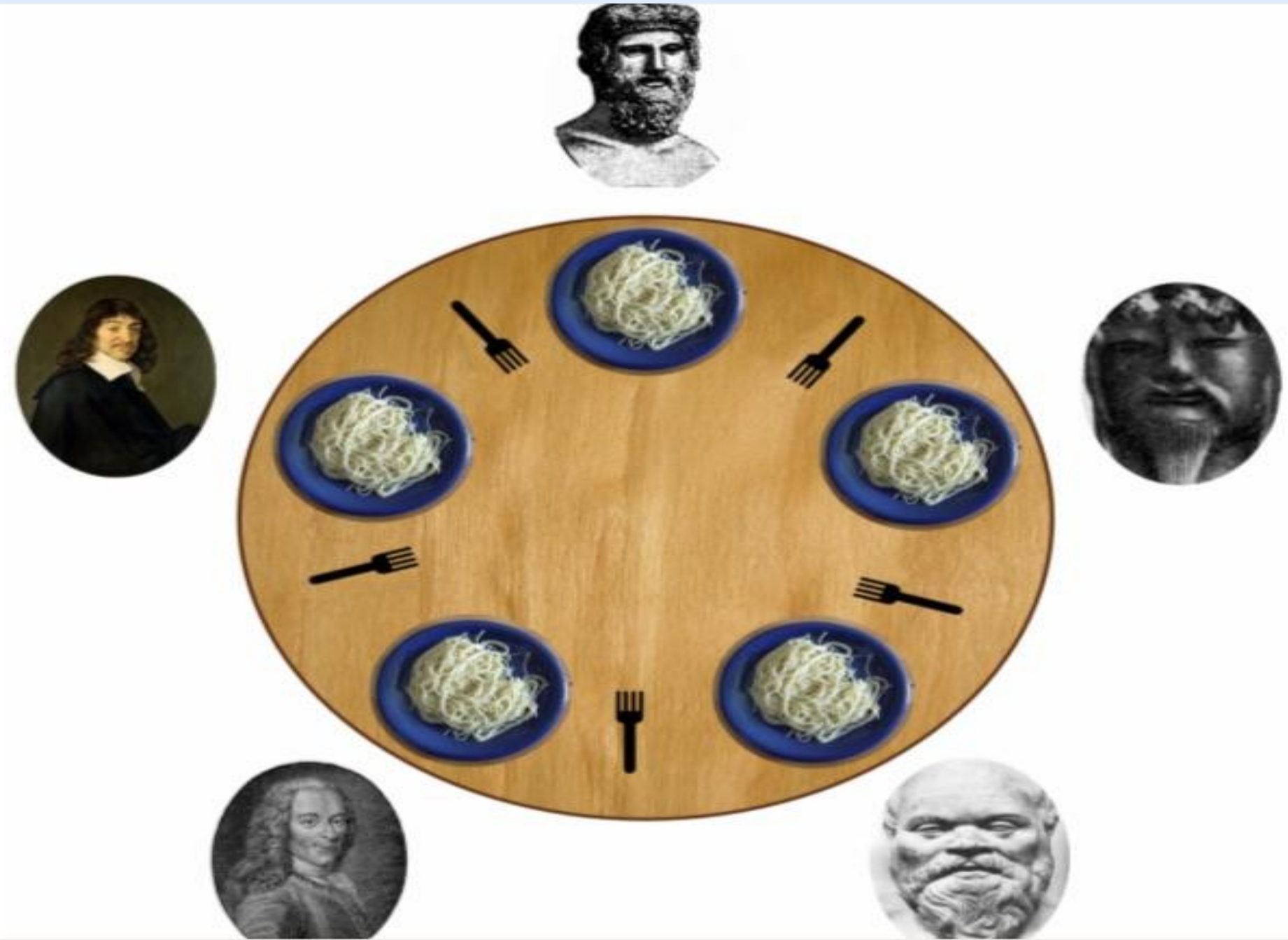
Planificación con Hebras

- La planificación se separa a dos niveles: procesos; hebras.
- Un planificador de procesos elige el proceso. Después un planificador de hebra escoge la hebra.
- No existe apropiación entre hebras. Si la hebra agota el quantum del proceso, se salta a otro proceso. Cuando vuelva, seguirá con la misma hebra
- Si la hebra no agota el quantum, el planificador de hebra puede saltar a otra hebra del mismo proceso.

ABRAZO MORTAL - DEADLOCK

Un conjunto de procesos está en estado de "DEADLOCK" cuando cada proceso del conjunto está esperando por un evento que solo puede ser causado por otro proceso que está dentro de ese conjunto.

ABRAZO MORTAL - DEADLOCK



ABRAZO MORTAL - DEADLOCK

CONDICIONES NECESARIAS

- **Exclusión Mutua:** debe haber recursos no compartidos de uso exclusivo.

- **Espera y Retenido (Hold & Wait):** un proceso retiene un recurso y espera por otro

- **Sin Desalojo:** no es posible quitarle el recurso retenido a un proceso. Lo libera voluntariamente

- **Espera circular:** hay un ciclo de espera entre procesos