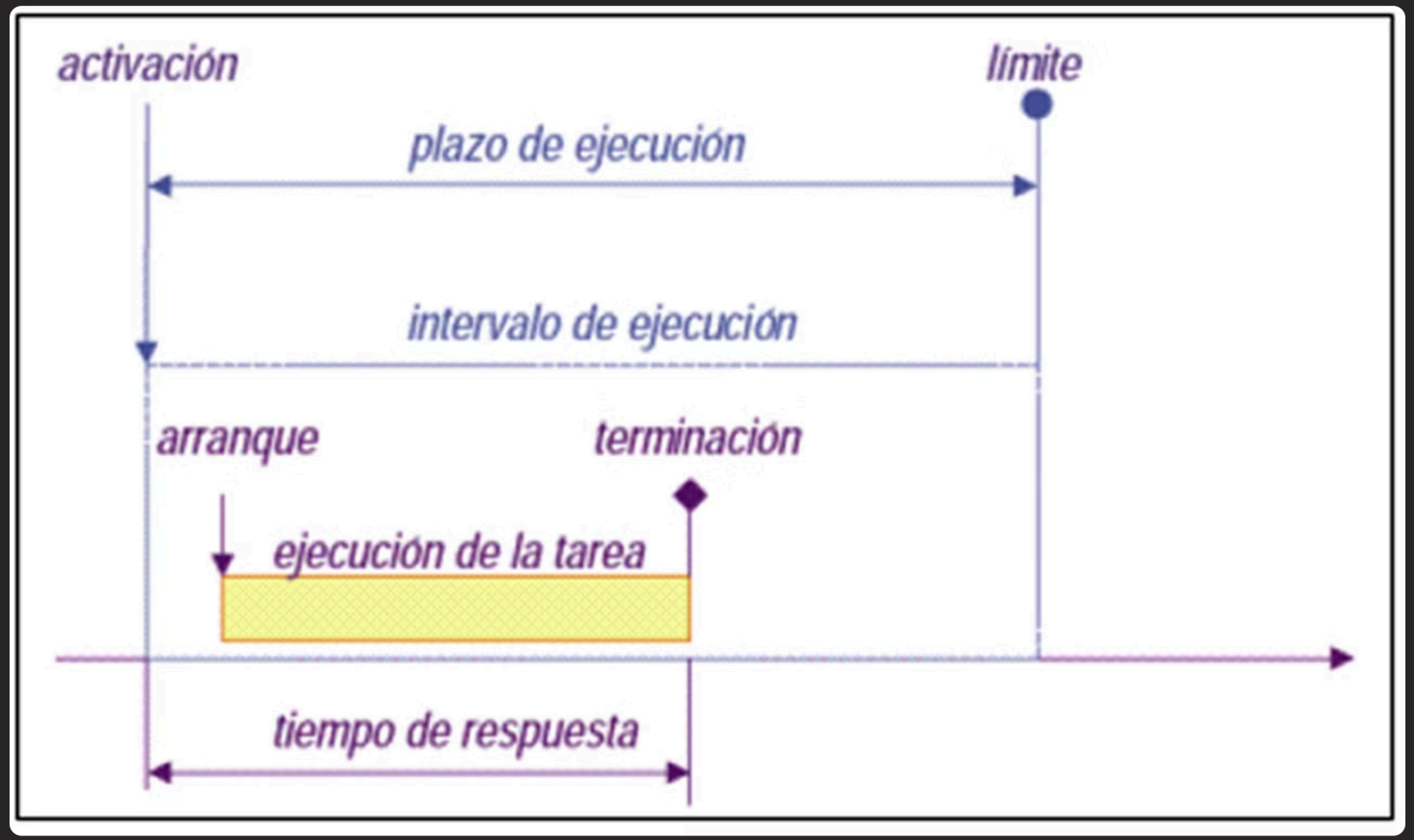
Sistemas de tiempo real

Definición:

Un **sistema de tiempo real** es un sistema donde el tiempo de respuesta es tan importante como la operación correcta. Los RTOS se utilizan en situaciones donde se requiere que el sistema responda a eventos en **plazos definidos**.

Determinismo:

El **determinismo** en un **sistema de tiempo real (RTOS)** se refiere a la capacidad del sistema para garantizar que las tareas o eventos se completen dentro de un **plazo predefinido** y de manera **predecible**. Es aquel en el que se puede predecir con precisión **cuándo** y **cómo** una tarea se ejecutará, sin importar las **condiciones externas o internas**.



Diferencia entre RTOS duro y blando

RTOS duro: La falla en cumplir un plazo (deadline) **causa consecuencias catastróficas**. Ejemplo: sistemas de control de vuelo, sistemas médicos, automóviles autónomos.

RTOS blando: Los plazos son importantes, pero no cumplirlos solo afecta el **rendimiento**, no la seguridad o funcionalidad crítica. Ejemplo: videojuegos, sistemas multimedia.

Ejemplos

- Control de un marcapasos cardíaco
- Sistema de entretenimiento multimedia en un smartphone
- Sistema de frenos antibloqueo (ABS) en automóviles
- Sistema de control de tráfico aéreo
- Sistema de monitoreo de una línea de producción en una fábrica
- Sistema de navegación de un avión
- Sistema de procesamiento de transacciones bancarias en línea

Ejemplos

- **Control de un marcapasos cardíaco** - Hard Real-Time
- **Sistema de entretenimiento multimedia en un smartphone** - Soft Real-Time
- **Sistema de frenos antibloqueo (ABS) en automóviles** - Hard Real-Time
- **Sistema de control de tráfico aéreo** - Hard Real-Time
- **Sistema de monitoreo de una línea de producción en una fábrica** - Soft Real-Time
- **Sistema de navegación de un avión** - Hard Real-Time
- **Sistema de procesamiento de transacciones bancarias en línea** - Soft Real-Time

Sistemas multitarea

Un **sistema multitarea** es aquel en el que múltiples tareas (o procesos) pueden ejecutarse de manera **simultánea** o **concurrente** bajo el control de un sistema operativo, donde cada una tiene sus propias **restricciones de tiempo**.

Importancia en sistemas de tiempo real:

- En un RTOS, **la multitarea no solo consiste en ejecutar varias tareas**, sino en **ejecutarlas dentro de plazos garantizados**. Esto asegura que todas las tareas se completen en el tiempo requerido, algo crítico en aplicaciones como sistemas médicos, automóviles autónomos o control industrial.
- Un **desafío clave** es atender múltiples tareas "al mismo tiempo", cuando en realidad solo una tarea puede ejecutarse en la CPU a la vez. El RTOS debe distribuir eficientemente los **recursos** del sistema para garantizar que las tareas importantes (de alta prioridad) se ejecuten primero, mientras las de menor prioridad esperan su turno.

Estrategias de multitarea

Gran loop o bucle principal

- En sistemas muy simples, la multitarea se implementa a través de un **gran bucle principal** (gran loop) que se repite constantemente, ejecutando cada tarea una tras otra en secuencia.
- **Ventaja**: Es fácil de implementar y entender.
- **Desventaja**: Tiene serios problemas para manejar **tareas con plazos críticos** y **eventos imprevistos**, ya que las tareas no pueden ejecutarse fuera de orden o interrumpir a otras si surge un evento urgente.

Ejemplo: Un programa de un microcontrolador que monitorea sensores de temperatura, humedad y controla motores. Cada lectura de sensor se realiza en secuencia dentro del bucle principal, pero si un sensor tiene un valor crítico, no se puede reaccionar hasta que todas las otras tareas se completen.

```
void loop() {
    leerTemperatura();
    leerHumedad();
    controlarMotor();
}
```

Estrategias de multitarea

Sistemas multitarea

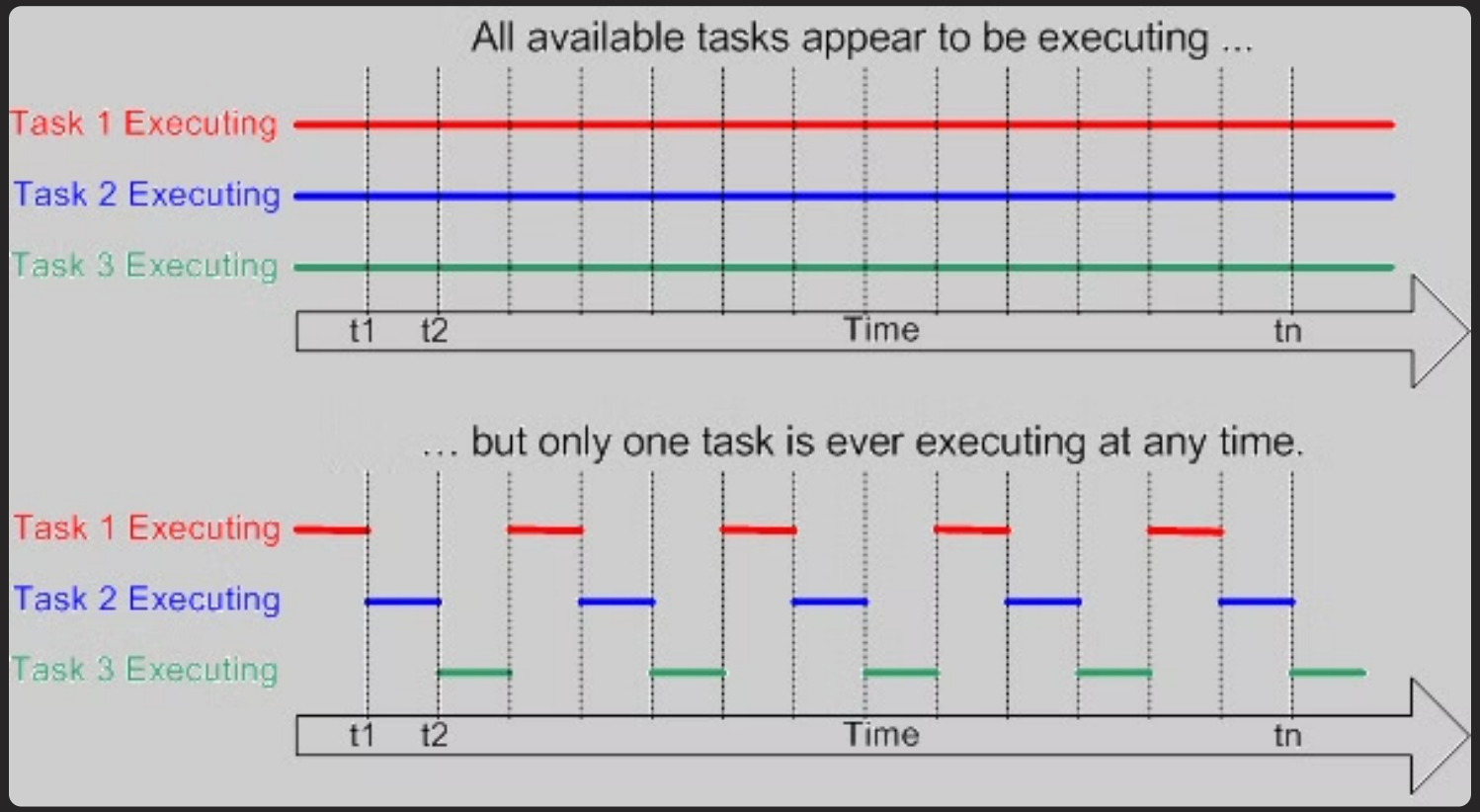
En sistemas más complejos, la multitarea se maneja utilizando un **sistema operativo** que distribuye el tiempo de CPU entre varias tareas. Las tareas no se ejecutan secuencialmente, sino que el **sistema operativo** decide cuándo y cómo ejecutarlas, dando la **apariencia de concurrencia**.

Componentes clave en un sistema multitarea:

1. **Scheduler (Planificador)**: Decide **cuál tarea** debe ejecutarse en un momento dado.
2. **Dispatcher**: Es el mecanismo que **interrumpe** una tarea y **comienza** otra, garantizando que el sistema pueda cambiar rápidamente entre tareas.
3. **Comunicación**: Proporciona métodos para que las tareas **interactúen entre sí** y compartan recursos de manera segura (semaphores, mutexes, colas, etc.).

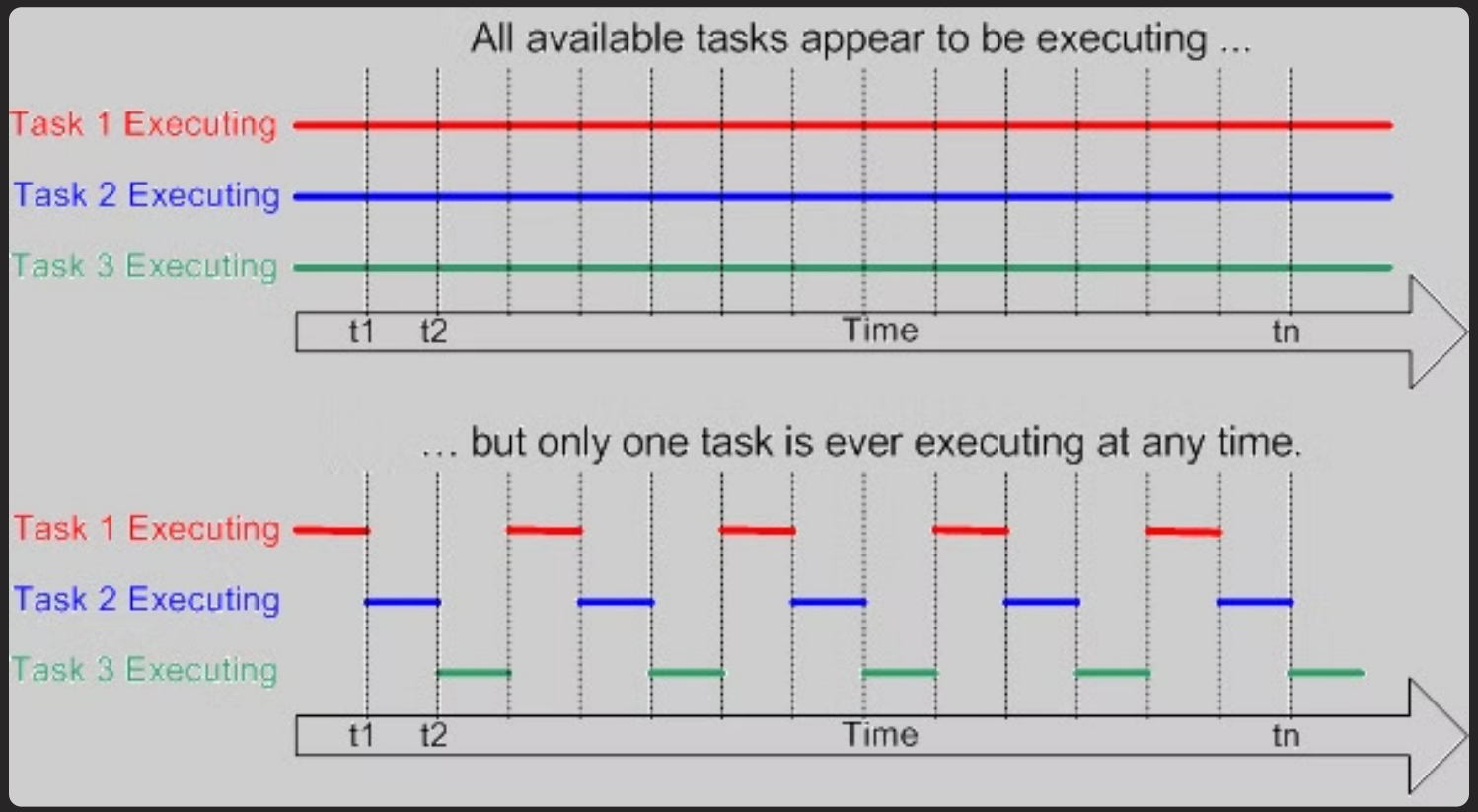
Scheduler (Planificador)

- El **scheduler** es responsable de gestionar el **tiempo de CPU** asignado a cada tarea. Es el que decide qué tarea debe ejecutarse en un momento dado, basándose en varios criterios, como **prioridad**, **estado de las tareas** (si están listas para ejecutarse, esperando, etc.), y el **plazo** (deadline) asociado a cada tarea.



Dispatcher

El **dispatcher** es el componente encargado de interrumpir la ejecución de una tarea y comenzar la ejecución de otra. Se asegura de que el cambio de contexto entre tareas se realice de manera eficiente, guardando el estado de la tarea actual y restaurando el estado de la siguiente tarea que se va a ejecutar. Es fundamental para garantizar una rápida transición entre tareas en un sistema multitarea.



Comunicación

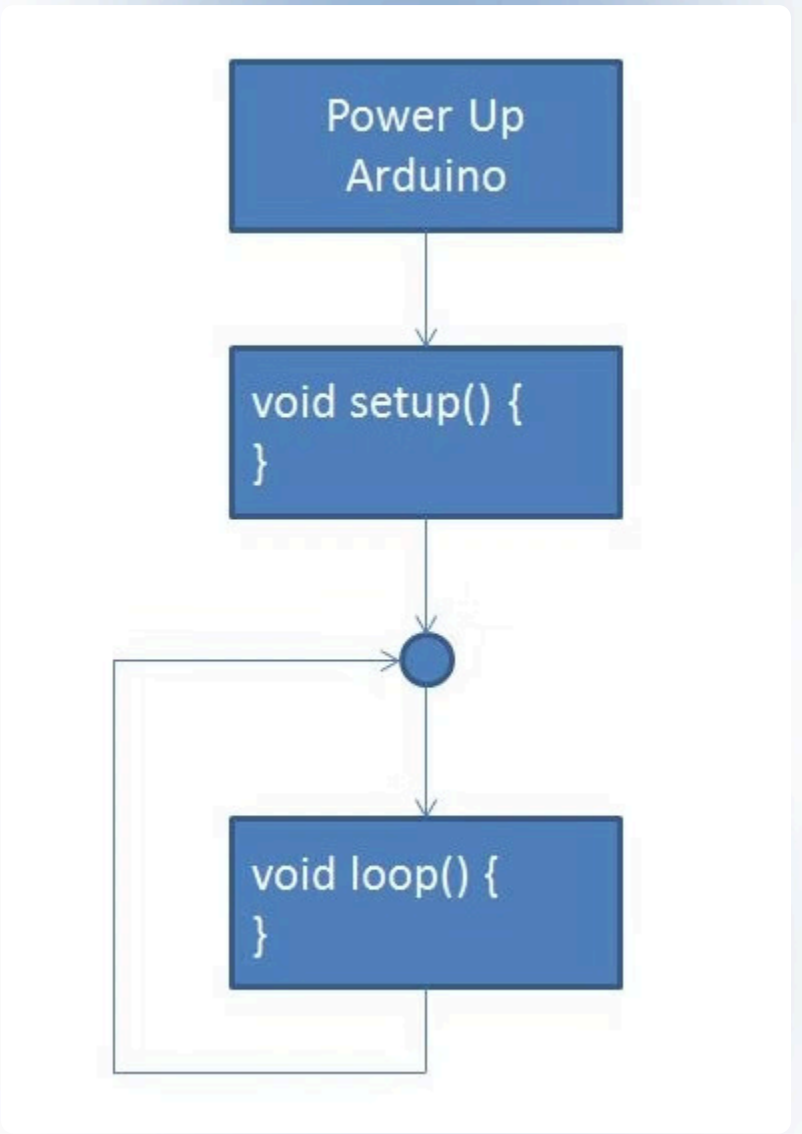
La **comunicación** entre tareas en un sistema multitarea se realiza a través de mecanismos seguros que permiten compartir recursos sin conflictos. Algunos de estos mecanismos incluyen semáforos, mutexes, y colas. Estos métodos aseguran que las tareas puedan sincronizarse, compartir datos y acceder a recursos compartidos de manera ordenada, evitando condiciones de carrera y otros problemas de concurrencia.

Tipos de sistemas multitarea

- Sistemas cíclicos
- Sistemas cooperativos
- Sistemas basados en interrupciones

Sistemas cíclicos

- Ventajas: simple y fácil de implementar.
- Desventajas: Poco flexible y no reacciona bien a eventos inesperados.



Sistemas cooperativos

- En este enfoque, las tareas se ejecutan hasta que **ceden voluntariamente** el control al sistema operativo. Esto significa que una tarea debe **terminar su trabajo** antes de permitir que otra tarea se ejecute.
- Es más eficiente que el gran loop, pero puede haber problemas si una tarea se queda "colgada" o no libera el control a tiempo.
- **Ejemplo**: Un sistema de alarma que cede el control tras revisar si hay una condición de alarma activa.

Sistemas basados en interrupciones

- En este tipo de sistemas, las tareas pueden ser **interrumpidas** por eventos externos. Una interrupción es un evento que indica que debe ejecutarse una tarea en **tiempo real** de manera inmediata.
- El sistema multitarea maneja la ejecución y las prioridades, pero las **interrupciones** permiten que tareas urgentes (como la lectura de un sensor o la respuesta a un botón) se atiendan de inmediato.
- **Ejemplo**: Un botón de emergencia en una fábrica que, al ser presionado, genera una interrupción para detener todas las máquinas.

Sistemas basados en interrupciones

Estrategias de planificación:

- Sistemas de tiempo compartido
- Sistemas preemptivos basados en prioridades
- Sistemas tipo round-robin

Tiempo compartido

Tiempo compartido es una técnica de planificación donde el **tiempo de la CPU** se divide en **bloques pequeños**. Cada tarea recibe una ***cantidad de tiempo definida*** para ejecutarse antes de que el sistema operativo cambie a la siguiente tarea.

Ventajas:

- Permite la ejecución **concurrente** de varias tareas en sistemas multitarea.
- **Equitativo**: Cada tarea obtiene tiempo de CPU.

Desventajas:

- Puede haber **ineficiencia** si las tareas importantes necesitan más tiempo del que se les asigna, o si tareas menos importantes consumen tiempo innecesario.

Ejemplo

Imagina 4 tareas: **T1** con tiempo de ejecución de 30ms y **T2**, **T3**, **T4**, con un **tiempo de ejecución** de 20 ms cada una.

Tiempo (ms)	Tarea ejecutada
0 - 30	T1
30 - 50	T2
50 - 70	T3
70 - 90	T4
90 - 120	T1

Preemptivos basados en prioridades

En un sistema **preemptivo**, las tareas se ejecutan según un esquema de **prioridades**. Las tareas de **mayor prioridad** pueden **interrumpir**a las de menor prioridad para ejecutarse de inmediato.

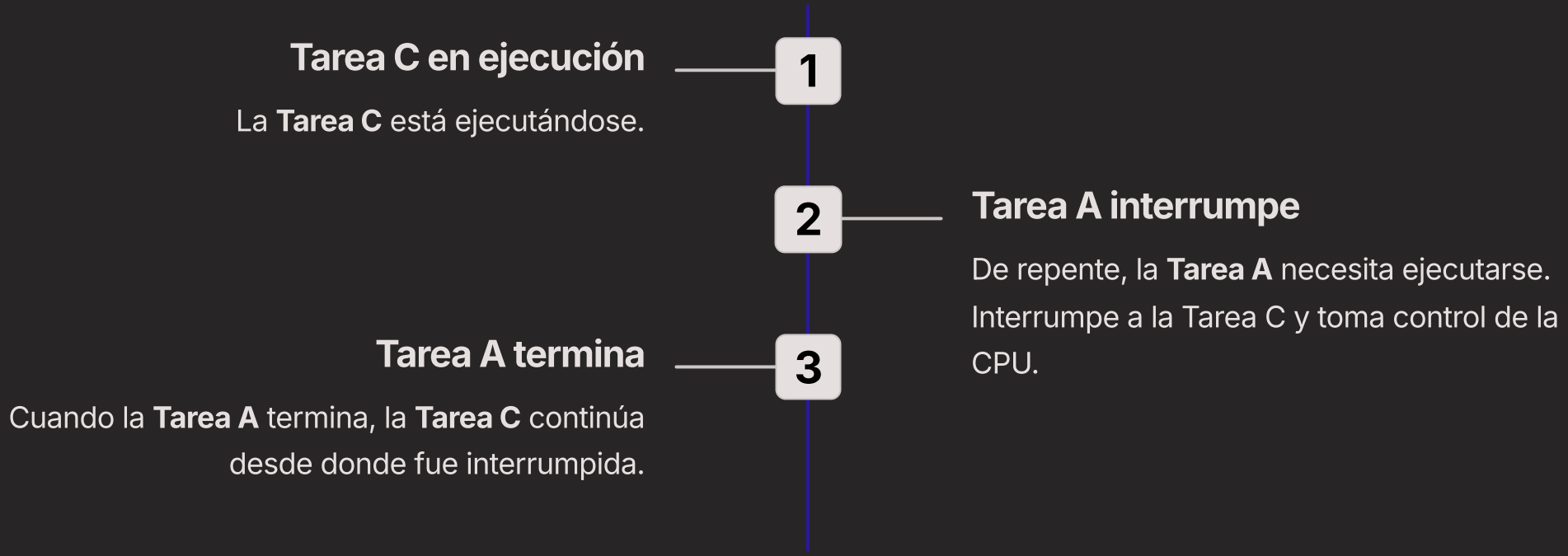
Ventajas:

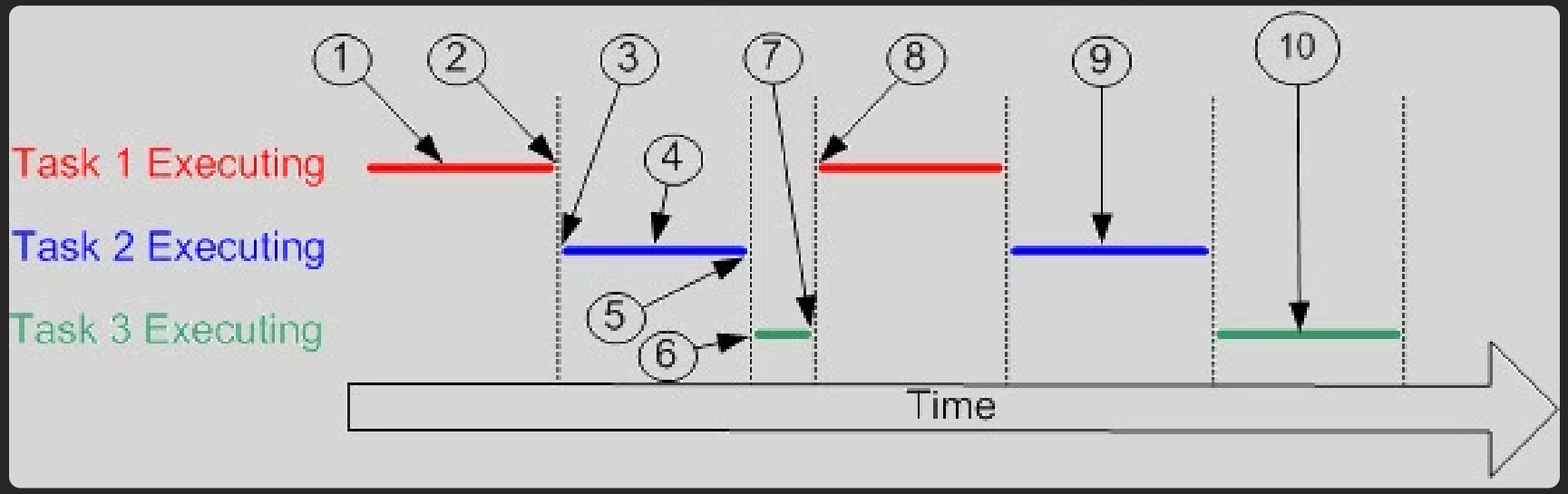
- **Determinismo**: Garantiza que las tareas **críticas** se ejecuten sin demoras.
- **Tiempos de respuesta rápidos**: Las tareas importantes no esperan en la cola si necesitan ser ejecutadas inmediatamente.

Desventajas:

- Introduce **sobrecarga** en el sistema operativo, ya que cambiar entre tareas requiere tiempo y recursos.

Ejemplo





Round-Robin

Es una técnica de planificación donde **cada tarea** recibe la misma **cantidad fija de tiempo** (quantum) para ejecutarse. Cuando el tiempo de una tarea se agota, el sistema cambia a la siguiente tarea en la cola, sin importar si la tarea ha terminado.

Ventajas:

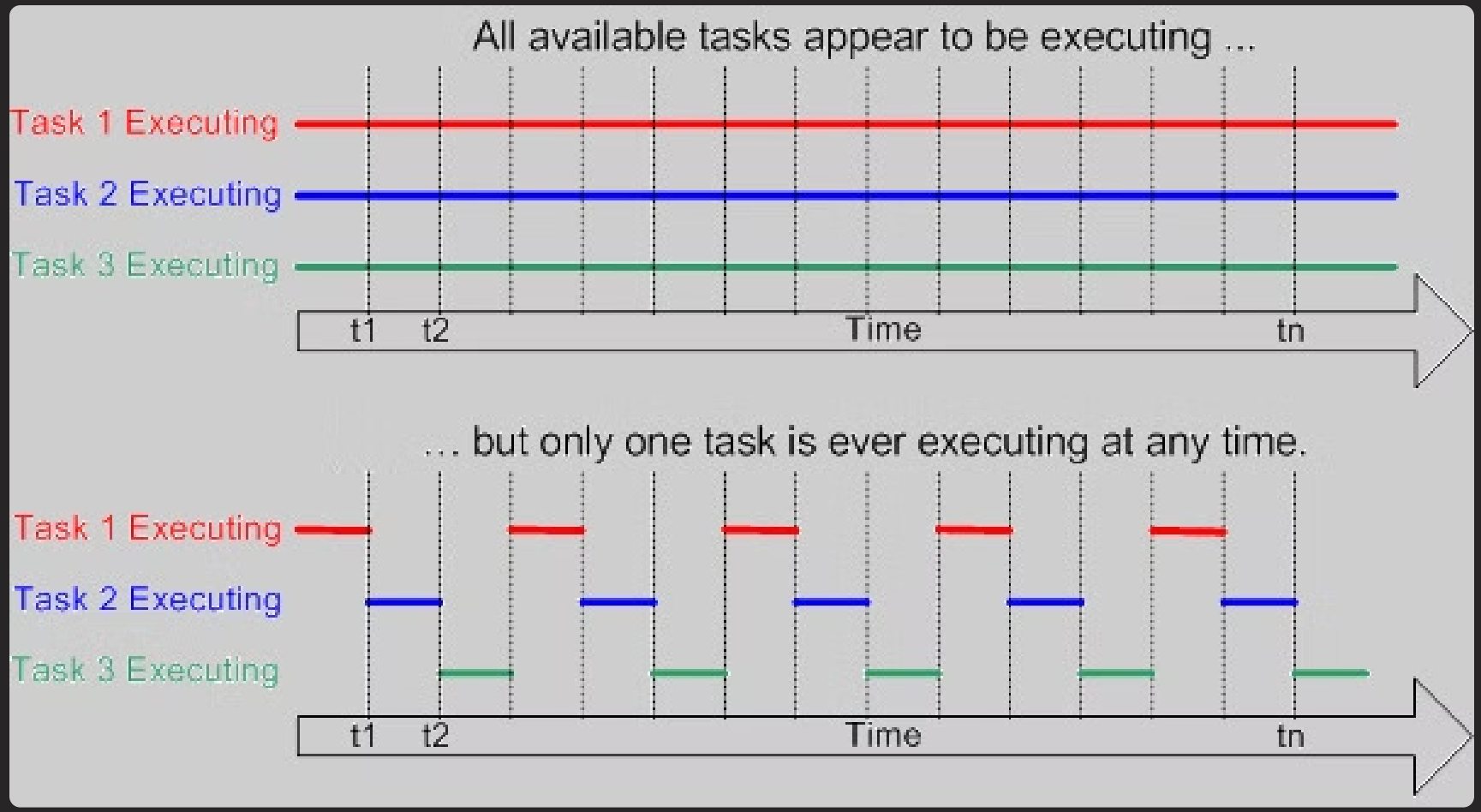
- **Justicia**: Todas las tareas reciben tiempo de CPU de manera equitativa.
- **Simples de implementar** en sistemas donde no hay tareas críticas que necesiten más atención que otras.

Desventajas:

- **Inflexible**: No toma en cuenta la **prioridad** de las tareas, por lo que una tarea importante puede quedar esperando mucho tiempo si hay muchas tareas en la cola.

Ejemplo

Tiempo (ms)	Tarea ejecutada
0 - 20	T1
20 - 40	T2
40 - 60	T3
60 - 80	T4
80 - 100	T1



Método	Tiempo compartido	Preemptivo	Round-Robin
Función principal	Dividir el tiempo de CPU en bloques para tareas	Las tareas de mayor prioridad interrumpen a las de menor	Cada tarea recibe un quantum fijo para ejecutarse
Prioridad	No considera prioridad (pero si)	Considera prioridad	No considera prioridad
Ventajas	Justo, sencillo	Tareas críticas se ejecutan sin demoras	Fácil implementación, justo para todas las tareas
Desventajas	Ineficiente para tareas críticas	Sobrecarga en el sistema	Las tareas importantes pueden quedarse esperando

Mecanismos de Coordinación entre Tareas

En un **sistema multitarea**, diferentes tareas pueden ejecutarse "simultáneamente" o de manera intercalada, y a menudo necesitan **compartir recursos** (como datos en memoria, dispositivos de hardware, etc.). Para evitar conflictos o errores al compartir recursos, los sistemas operativos en tiempo real (RTOS) ofrecen diversos **mecanismos de coordinación**

Problemas comunes en la coordinación entre tareas

Condición de carrera: Ocurre cuando dos o más tareas acceden o modifican un recurso compartido al mismo tiempo y el resultado depende del orden en que ocurren las operaciones.

Mecanismos de Coordinación Principales

Semáforos:

- **Definición**: Son contadores que controlan el acceso a recursos compartidos.
- **Tipos**:
  - **Binarios**: Solo permiten que **una tarea** acceda a un recurso (0 o 1).
  - **De conteo**: Permiten que **varias tareas** accedan simultáneamente a un recurso.
- **Uso**: Sincronización entre tareas para asegurar que no accedan a un recurso al mismo tiempo.

Mecanismos de Coordinación Principales

Mutexes (Exclusión Mutua):

- **Definición**: Similar a un semáforo, pero diseñado específicamente para garantizar que **solo una tarea** a la vez pueda acceder a un recurso.
- **Uso**: Evitar condiciones de carrera cuando se accede a **recursos críticos**.
- **Característica**: Solo la tarea que bloquea el mutex puede desbloquearlo.

Mecanismos de Coordinación Principales

Mailboxes y Colas de Mensajes:

- **Definición**: Mecanismos de **comunicación** entre tareas a través del envío y recepción de mensajes.
  - **Mailboxes**: Envían **datos pequeños** entre tareas.
  - **Colas de mensajes**: Envían **múltiples mensajes**, organizados en orden FIFO (primero en entrar, primero en salir).
- **Uso**: Transferencia de datos o notificaciones entre tareas sin necesidad de acceder a memoria compartida.

Mecanismos de Coordinación Principales

Señales (Eventos):

- **Definición**: **Notificaciones** que activan o despiertan a una tarea cuando ocurre un evento.
- **Uso**: Sincronizar tareas con eventos específicos (ej. tarea que espera a que termine otra acción).

Mecanismos de Coordinación Principales

Memoria Compartida:

- **Definición**: Un espacio de memoria común accesible por varias tareas.
- **Problema**: Puede generar **condiciones de carrera** si no se sincroniza adecuadamente.
- **Solución**: Usar semáforos o mutexes para asegurar que solo una tarea acceda a la memoria a la vez.

Mecanismo	Descripción	Uso principal
Semáforos	Controlan el acceso a recursos compartidos (binarios o de conteo)	Sincronización entre tareas
Mutexes	Exclusión mutua: garantiza que solo una tarea acceda a un recurso a la vez	Acceso exclusivo a recursos críticos
Mailboxes/Colas de mensajes	Comunicación entre tareas a través de mensajes	Transferencia de datos entre tareas
Señales	Notificaciones que despiertan tareas en espera	Eventos o condiciones específicas
Memoria compartida	Espacio de memoria accesible por varias tareas	Transferencia rápida de datos

Microcontroladores

Está diseñado para controlar **tareas específicas** en sistemas embebidos. Se utiliza en aplicaciones de **bajo costo** y **gran personalización**, como electrodomésticos, juguetes, sistemas de monitoreo, dispositivos médicos, automóviles, y proyectos de hobby como los de Arduino.

Característica	Microcontrolador (MCU)	PLC
Aplicación	Control específico en sistemas embebidos	Automatización y control industrial
Entorno de uso	Entornos controlados, no tan robusto	Entornos industriales severos, muy robusto
Capacidad I/O	Limitada, señales de baja potencia	Alta capacidad de expansión, señales industriales
Facilidad de programación	Compleja, requiere bajo nivel (C/C++)	Sencilla, lenguajes industriales (Ladder, FBD)
Costos	Bajo costo (2-10 USD)	Alto costo (cientos a miles de dólares)
Escalabilidad	Limitada, no modular	Alta escalabilidad, modularidad con expansión
Procesamiento	Personalizable, pero limitado	Menos personalizable, pero más robusto para control industrial