

# **Carrera de Especialización en Sistemas Embebidos Sistemas Operativos en Tiempo Real**

## **Clase 3: Sincronización de tareas (1ra parte)**



Asociación Civil para la Investigación,  
Promoción y Desarrollo de los  
Sistemas Electrónicos Embebidos

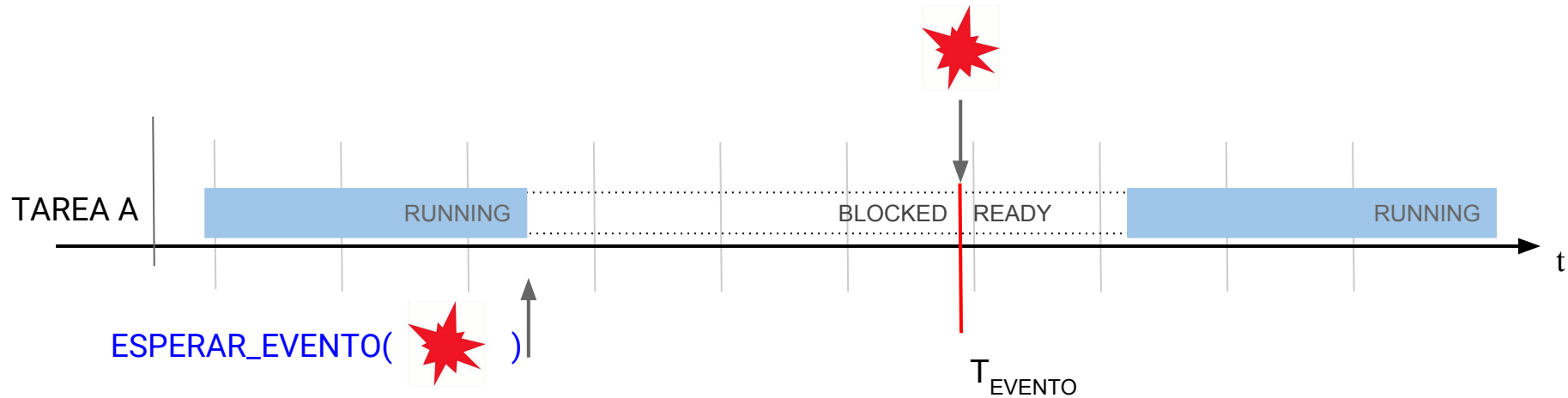


**FACULTAD  
DE INGENIERIA**  
Universidad de Buenos Aires

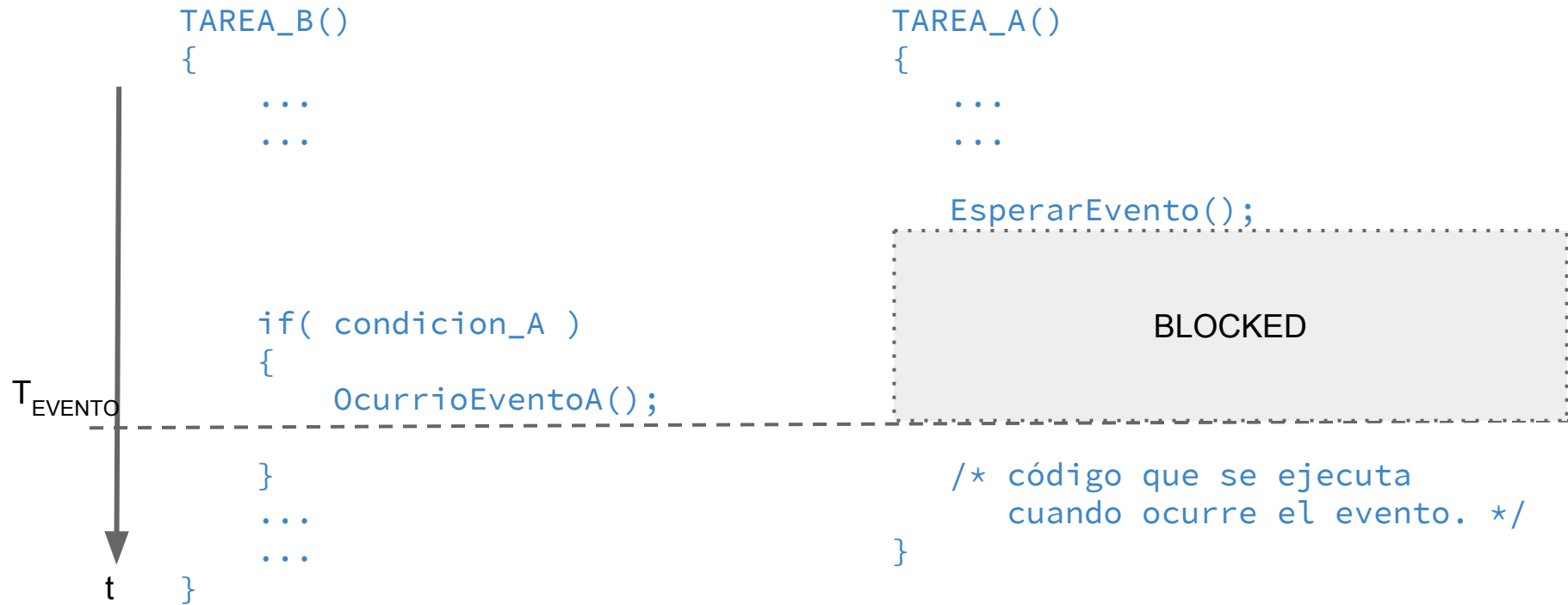


# Problema 1: Sincronización

- ¿Como se puede sincronizar el evento que ocurre en un contexto con una acción que requiere ser disparada en otro contexto a causa de dicho evento ?



# Problema 1: Sincronización



# SEMÁFORO

- Es un tipo de dato abstracto que permite sincronizar tareas mediante un mecanismo sencillo.
- Restringe el acceso a una sección del código hasta que alguien lo señalice (lo ponga "en verde")
- Tipos:
  - BINARIOS
  - CONTADORES
  - MUTEX



# SEMÁFOROS BINARIOS EN FREERTOS

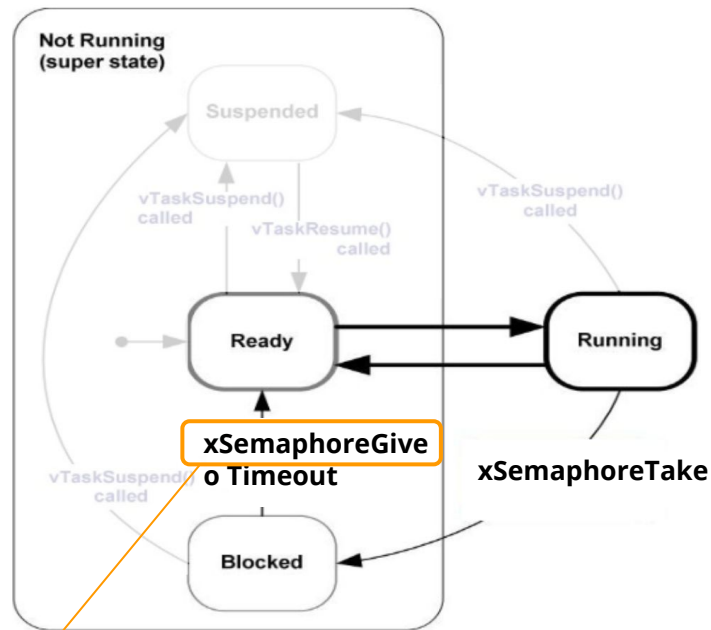


- El semáforo binario sólo puede estar en dos estados: 0 o 1
- Una tarea puede tomar (mediante la acción "take") el semáforo. Si el semáforo ya está tomado, entonces la tarea esperará a que alguien lo libere (mediante la acción "give")
- El semáforo no tiene "dueño". Es decir, lo puede tomar una tarea y lo puede liberar otra.

# API: Semáforos Binarios



- SemaphoreHandle\_t **mi\_semaforo** = **xSemaphoreCreateBinary()**;
  - Crea el semáforo (arranca "tomado").
- **xSemaphoreTake**( **mi\_semaforo** , TickType\_t xTicksToWait);
  - Toma el semáforo.
  - Retorna pdTRUE si se pudo tomar correctamente y pdFALSE si durante xTicksToWait ticks nadie lo liberó.
- **xSemaphoreGive**( **mi\_semaforo** )
  - Libera el semáforo.



Ejecutado por otra contexto

# SEMÁFOROS BINARIOS EN FREERTOS



```
SEM = xSemaphoreCreateBinary();           /* se instancia un semáforo  
                                           se inicializa "tomado" */
```

```
TAREA_A()  
{
```

```
    while(1)
```

```
    {
```

```
        xSemaphoreTake( SEM, portMAX_DELAY );
```

Esta macro obliga esperar "por siempre"

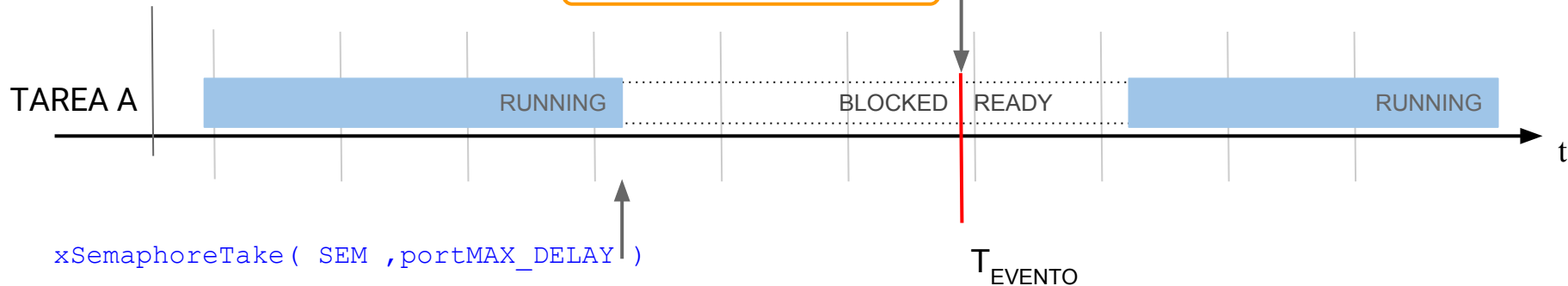
```
        /* código que ocurre cuando otro contexto llama a xSemaphoreGive */
```

```
    }
```

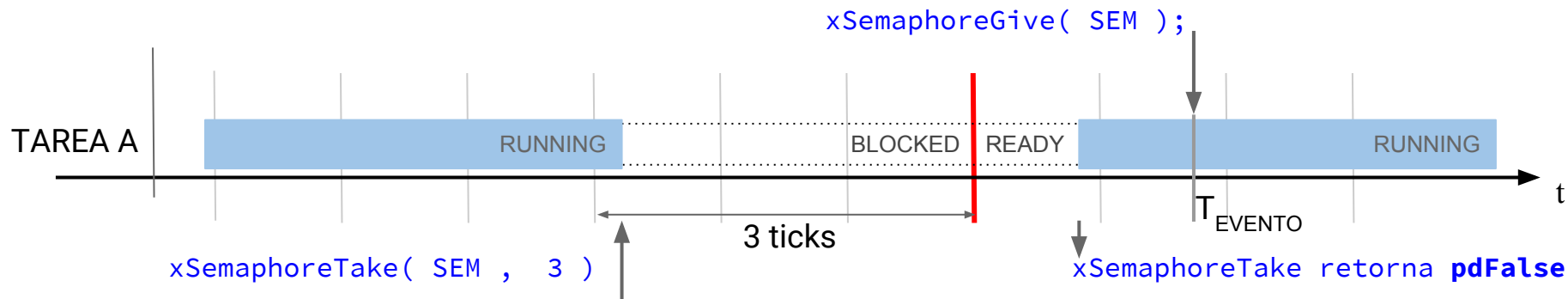
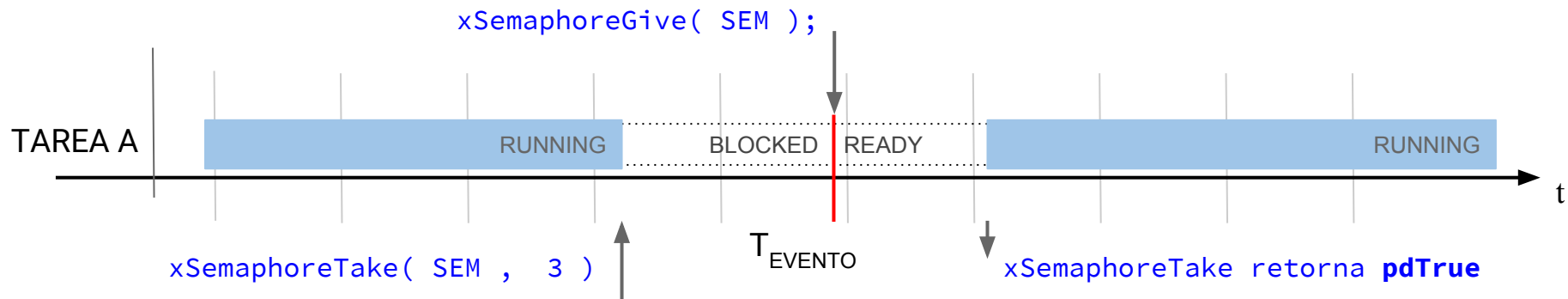
```
}
```

```
xSemaphoreGive( SEM );
```

Ejecutado por otra contexto



# SEMÁFOROS BINARIOS EN FREERTOS





# SEMÁFOROS BINARIOS EN FREERTOS



```
SEM = xSemaphoreCreateBinary();           /* se instancia un semáforo
                                           se inicializa "tomado" */

TAREA_A()
{
    while(1)
    {
        if( xSemaphoreTake( SEM , 3 ) == pdTRUE )
        {
            /* otro contexto liberó el semáforo */
        }
        else
        {
            /* pasaron más de 3 ticks, sin que otro contexto libere el semáforo */
        }
    }
}
```

- El semáforo contador puede estar en N estados: 0 a N-1
- Más de una tarea puede tomar (mediante la acción "take") el semáforo. Si el semáforo ya está tomado, entonces la tarea esperará a que alguien lo libere (mediante la acción "give")
  - "Tomar" un semáforo significa decrementar el contador (y si ya estaba en cero, bloquearse)
  - "Dar" un semáforo significa incrementar el contador.
- El semáforo no tiene "dueño". Es decir, lo puede tomar una tarea y lo puede liberar otra.
- SemaphoreHandle\_t **mi\_semaforo** = **xSemaphoreCreateCounting( MAX , INICIAL);**
  - Crea el semáforo contador con un valor máximo MAX y un valor INICIAL preestablecido (recordar que 0 es tomado, y >0 liberado N veces)

# SEMÁFOROS CONTADORES: EJ1

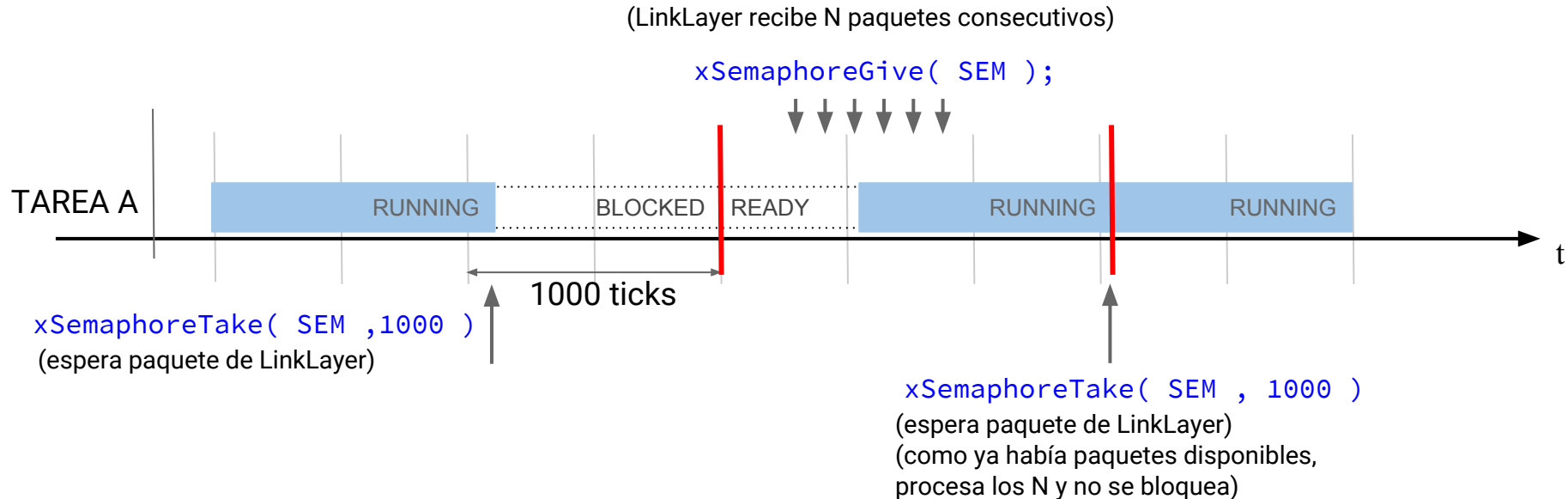


APLICACIÓN
LINK LAYER
PHY = UART

Tarea\_A()

procesa paquetes y los envía a capa superior

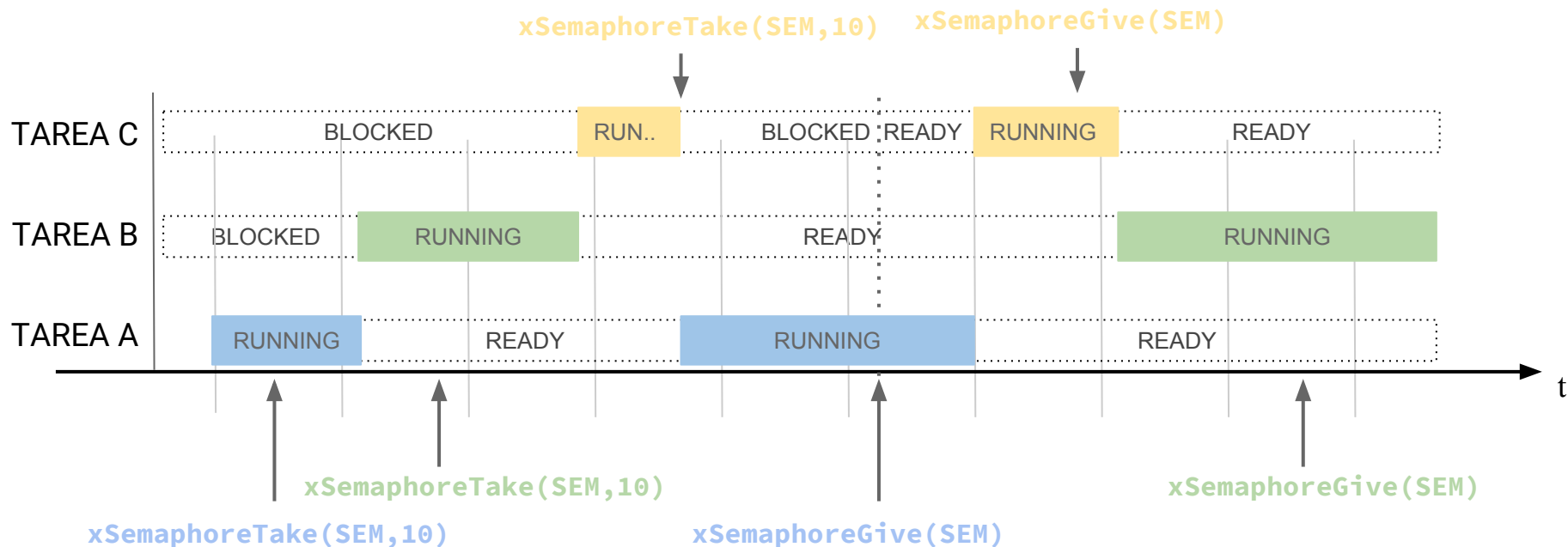
procesa bytes y los envía a capa superior



# SEMÁFOROS CONTADORES: EJ2



Tarea\_A(), Tarea\_B() y Tarea\_C() eventualmente solicitan un bloque de memoria a un gestor de bloques de memoria con solo 2 bloques disponibles.



# Bibliografía

---

- <https://www.freertos.org>
- Introducción a los Sistemas operativos de Tiempo Real, Alejandro Celery - 2014
- FreeRTOS - Semáforos, Cusos INET, Franco Bucafusco, 2017