# Carrera de Especialización en Sistemas Embebidos

# Sistemas Operativos en Tiempo Real

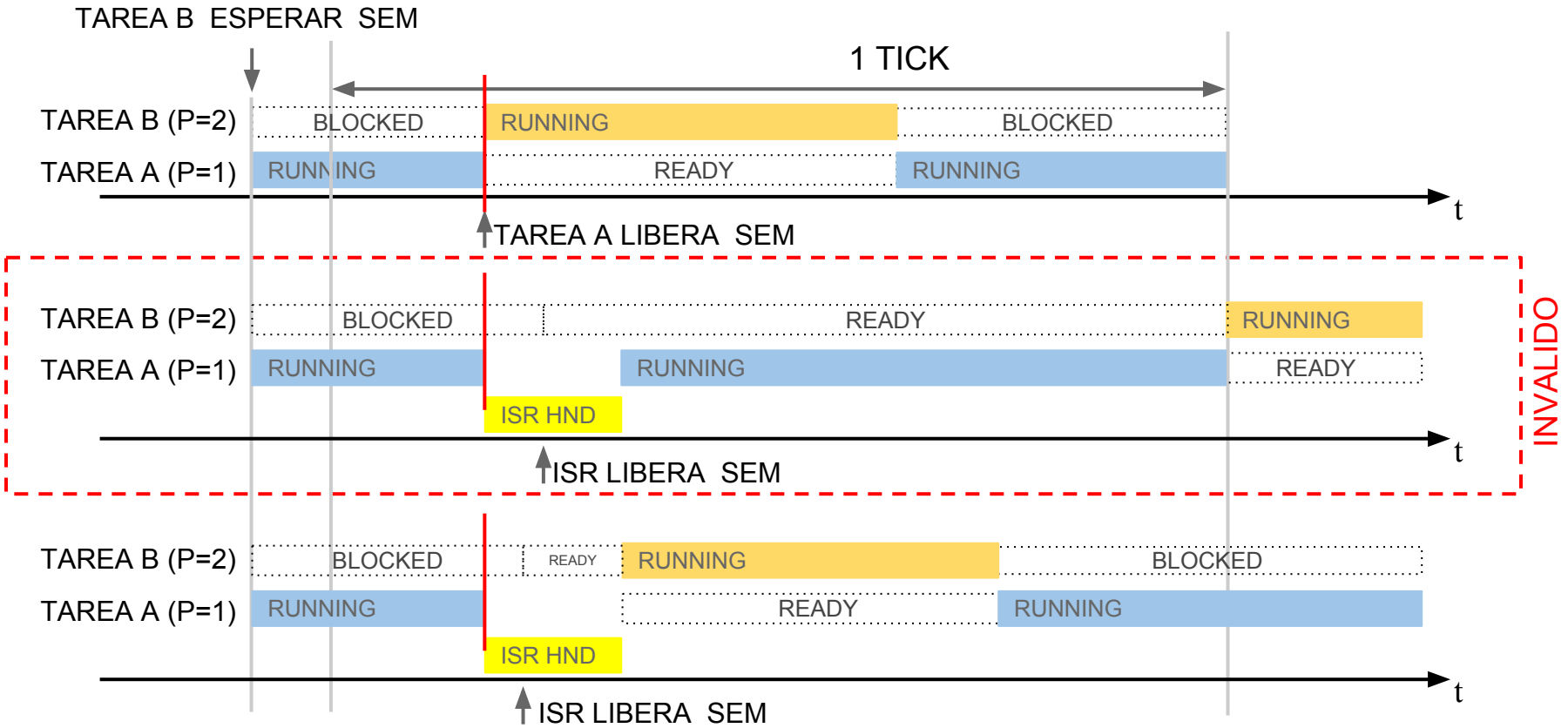
## Clase 5: Gestión de Interrupciones.

ACSE
Asociación Civil para la Investigación, Promoción y Desarrollo de los Sistemas Electrónicos Embebidos
FACULTAD DE INGENIERIA
Universidad de Buenos Aires
CIM
CADIEEL

# ¿COMO SE DEBEN USAR LAS ISR EN UN RTOS?

- ¿ Se pueden usar de igual forma que en un baremetal ?
- ¿ Se puede usar la API del RTOS vista hasta el momento ?
- ¿ Como se asegura que el uso de ISR conlleve una respuesta RT ?

# ¿COMO SE DEBEN USAR LAS ISR EN UN RTOS?



# Interrupciones en FreeRTOS

- Todas las funciones de la API que se pueden usar en un handler de interrupción finalizan con FromISR()
- Todas las funciones ...FromISR() llaman al scheduler, para conocer cuál es la siguiente tarea a ejecutar.
- A todas las funciones FromISR() habrá que pasarle un nuevo parámetro que indicará si hay que realizar un cambio de contexto o no.
- Como NO SE PUEDE utilizar bloqueos en un Handler, NINGUNA de las funciones ...FromISR() poseen el timeout tradicional.
- Si la función determinó que debe ocurrir un cambio de contexto, este ocurre a la salida del handler, generando una latencia adicional.

# Ejemplos

```
void vTimerISR( )
{
    static unsigned char ucLocalTickCount = 0;
    static signed BaseType_t xHigherPriorityTaskWoken= pdFALSE;

    /* A timer tick has occurred. */

    /* Is it time for vATask() to run? */
    ucLocalTickCount++;


    if( ucLocalTickCount >= TICKS_TO_WAIT )
    {
        /* Unblock the task by releasing the semaphore. */
        xSemaphoreGiveFromISR( xSemaphore, &xHigherPriorityTaskWoken );

        /* Reset the count so we release the semaphore again in 10 ticks time. */
        ucLocalTickCount = 0;
    }

    /* If xHigherPriorityTaskWoken was set to true you we should yield. */
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

# Tareas e Interrupciones

- Las tareas son elementos de Software cuya prioridad las define el programador de la aplicación y las cuales son controladas por el scheduler.

- Los handler de interrupción, son controlados por el hardware. Su prioridad no tiene nada que ver con la prioridad asignada a las tareas.

- Las tareas se ejecutan siempre y cuando no haya ningún handler de interrupción pendiente.
  - El handler de interrupción de menor prioridad (por hardware) será el techo de prioridad de todas las tareas implementadas.
  - O sea, cualquier handler interrumpirá a cualquier tarea, cualquiera sea su prioridad.

# Buenas prácticas

- Los handlers deben ser lo más corto posible porque:
  - Cualquier tarea va a ser retrasada por la ejecución del mismo.
  - El tiempo de ejecución de un handler puede agregarle mucha latencia variable a una tarea crítica.
  - Interrupciones múltiples, podrían empeorar el escenario.
- Los handlers deberán señalizar la ocurrencia del evento, pero el proceso de ese evento realizarse en la tarea (Application Controlled Deferred Interrupt Handling)
- Los handlers deberán acolar un callback al deamon que posee FreeRTOS, para ejecutarlo y procesar el evento. (Centralised Deferred Interrupt Handling)

# Bibliografia


- Amazon FreeRTOS – Interrupt management
- Introducción a los Sistemas operativos de Tiempo Real, Alejandro Celery - 2014
- Interrupciones, CAPSE, Franco Bucafusco, 2017
- Interrupciones - Ejemplos con FreeRTOS, CESE, Franco Bucafusco, 2017