

Sistemas Operativos II

Sistemas Operativos de Tiempo Real: FreeRTOS

Temario

- Porque un RTOS?
- Estructura FreeRTOS
- Manejo de Memoria en FreeRTOS
- Manejo del Tiempo en FreeRTOS
- Debug + Trace
- TP

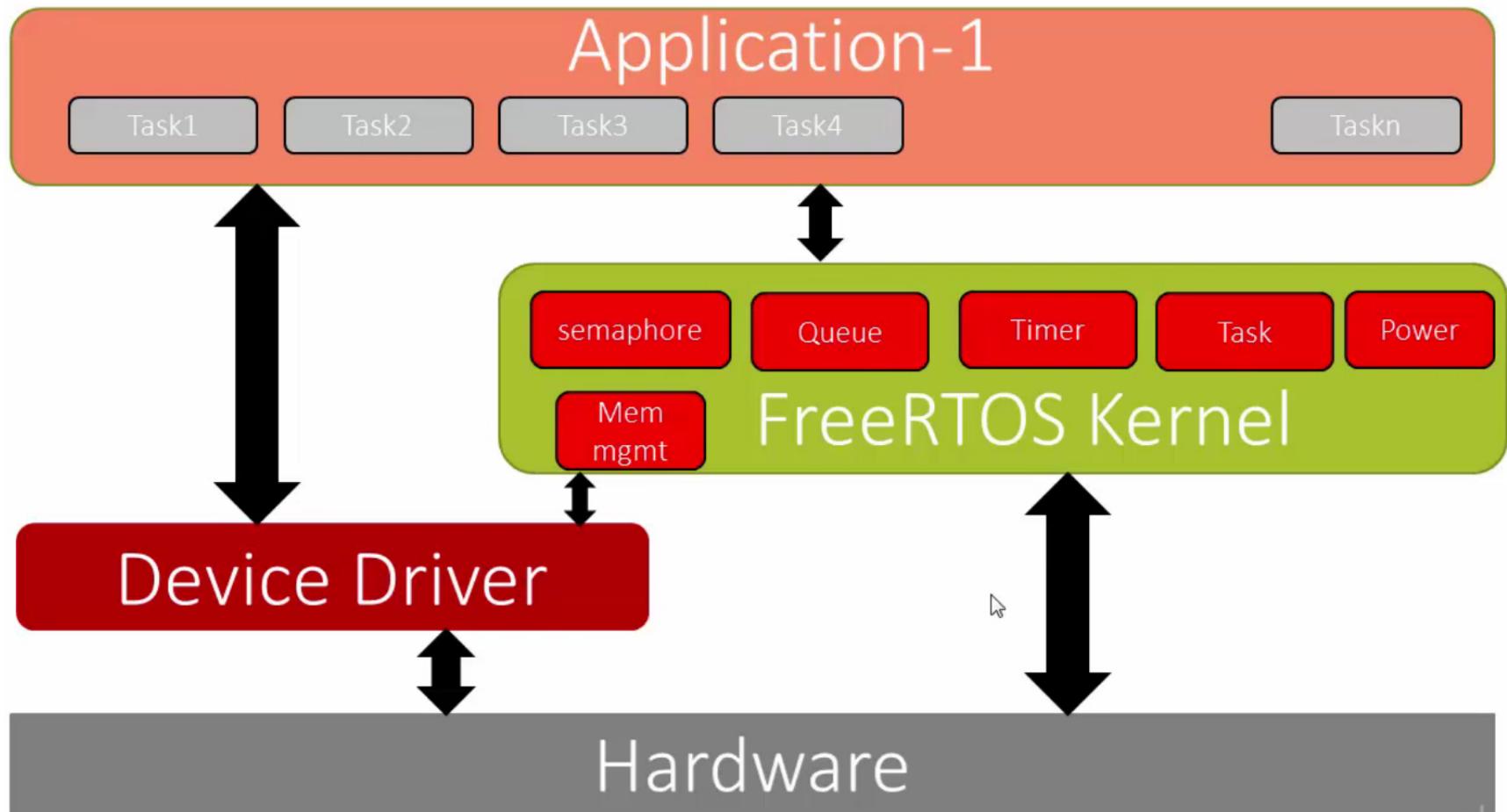
Porque un RTOS?

- Abstracción de la información de tiempo
- Independencia entre tareas
- Tarea idle: Bajo Consumo

Consecuencias del RTOS

- Se gasta tiempo de CPU para determinar que tarea se debe ejecutar (scheduler)
- Se gasta tiempo de CPU para realizar los cambios de contexto
- El kernel utiliza memoria de programa
- Cada tarea consume RAM por la TCB
- El programador deja de tener control de cuando se ejecutan las tareas y pasa a depender del scheduler

FreeRTOS



FreeRTOS

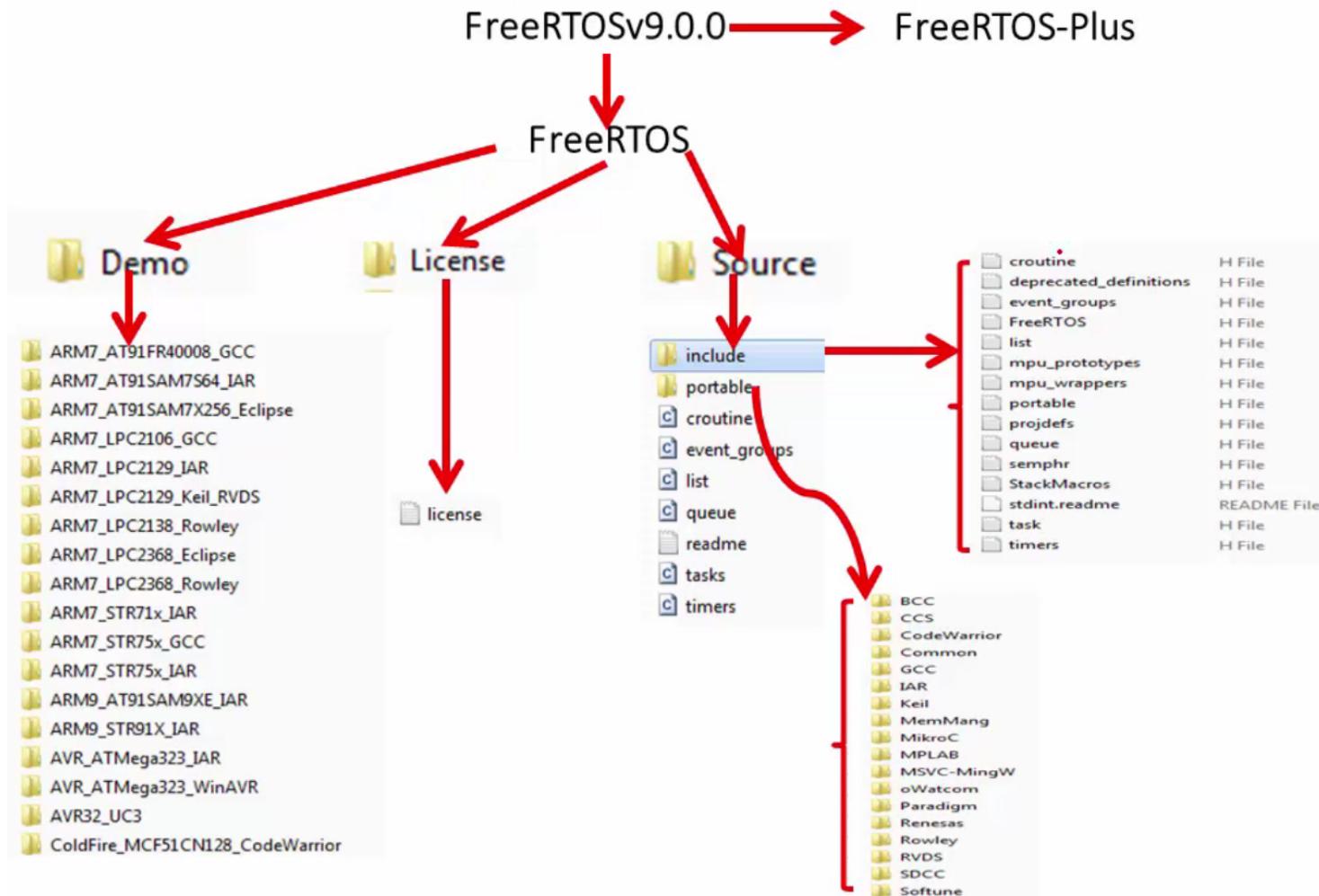
- FreeRTOS incluye:
 - Scheduler
 - Manejo dinámico de memoria
 - Elementos de sincronización:
 - Mutex, Semaforos, Colas
 - Timers por software

FreeRTOS

- FreeRTOS Plus:

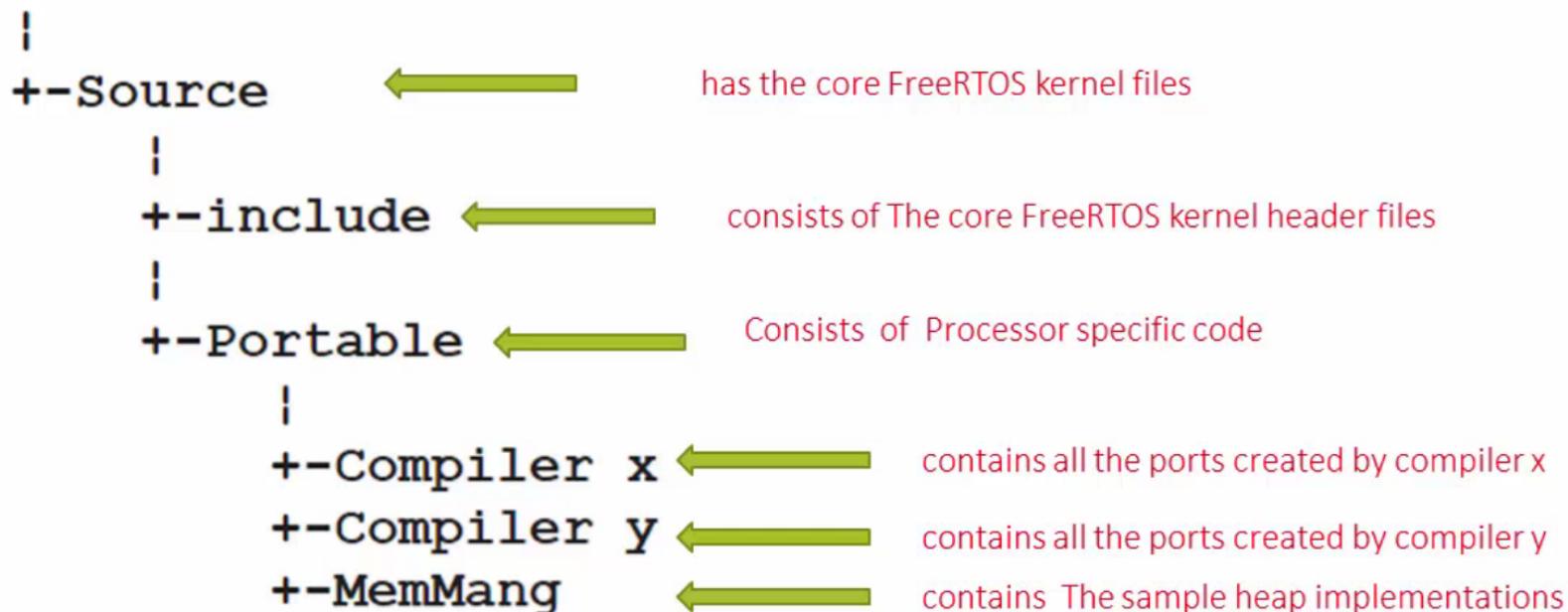
- TCP - UDP/IP
- FAT
- CLI
- Trace

FreeRTOS: Estructura

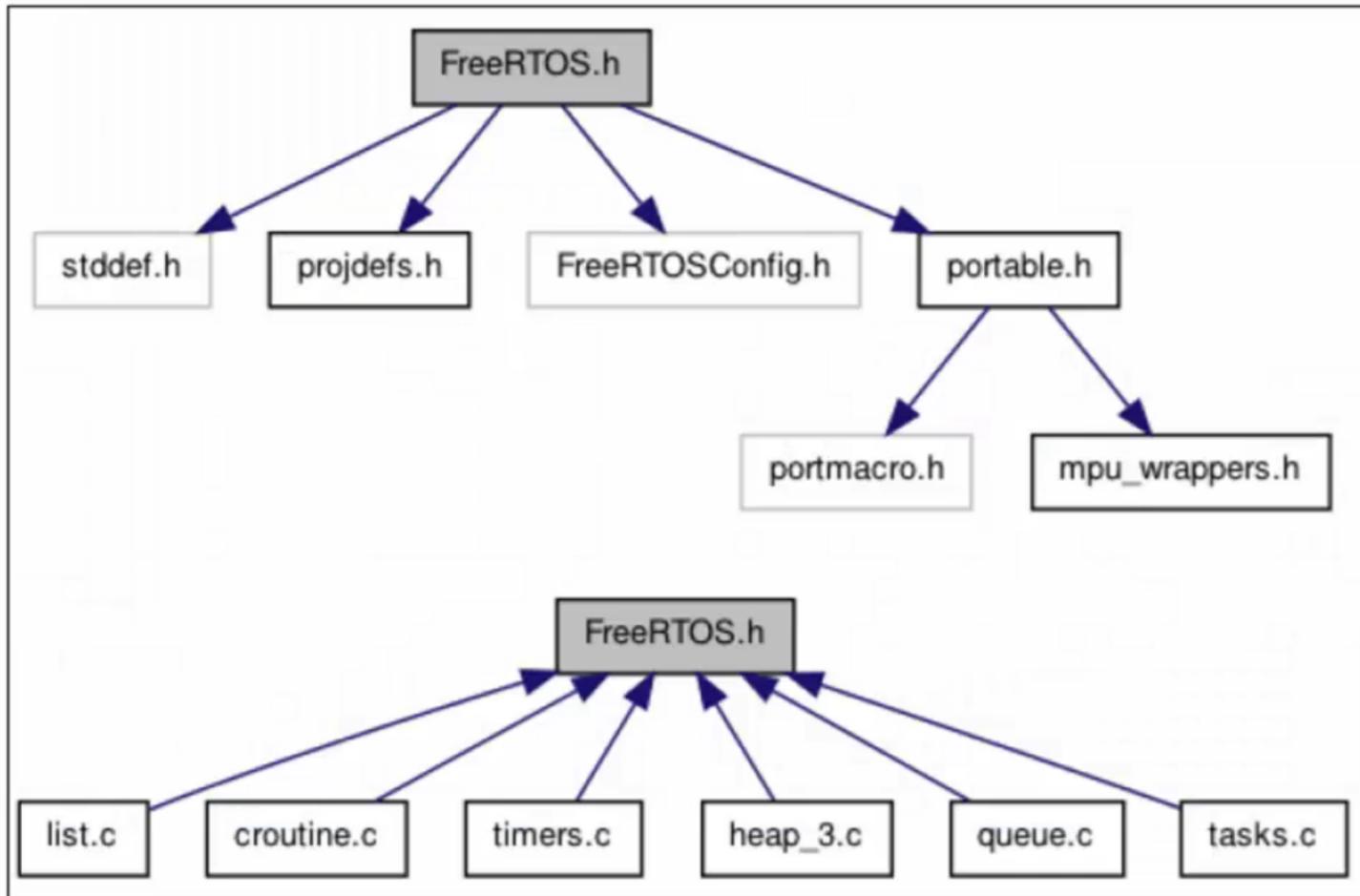


FreeRTOS: Estructura

FreeRTOS



FreeRTOS: Estructura



Convenciones

- Macros
 - configUSE_PREEMPTION
 - portNVIC_INT_CTRL_REG
- Archivos con configuracion con Macros:
 - FreeRTOSConfig.h – Configuración General
 - portmacro.h – Definiciones propias de la arquitectura

Convenciones

■ Variables

- El nombre de cada variable comienza con un prefijo con el tipo de dato.
 - Unsigned Long: “ul”
 - Unsigned Short: “us”
 - PointerVoid: “pv”
 - Unsigned Char: “uc”
 - Enum: “e”
 - Non Standard: “x”

Convenciones

■ Funciones

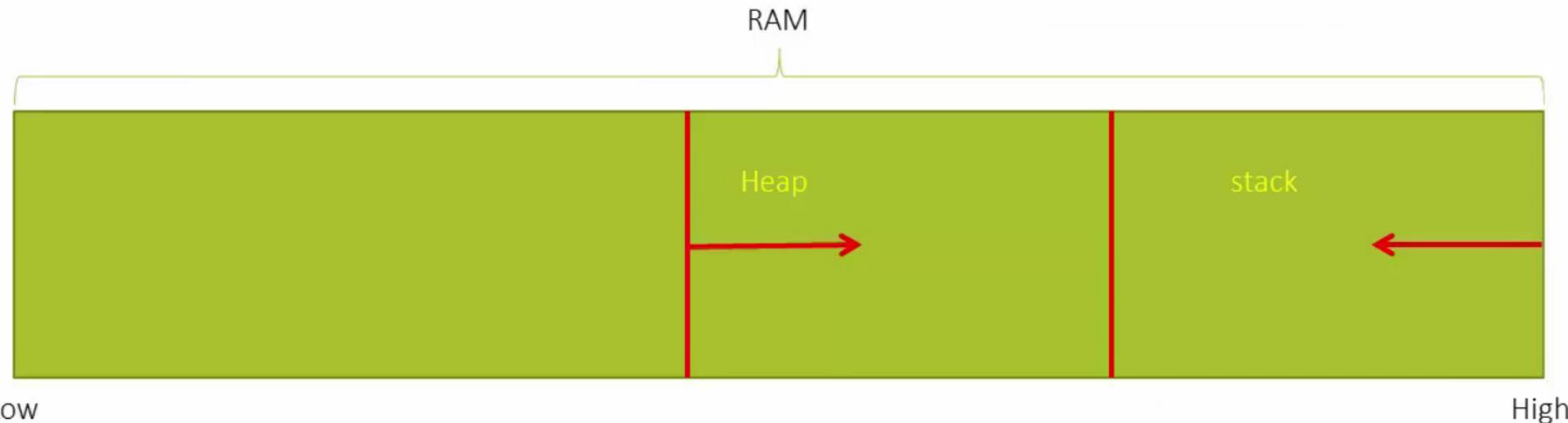
- Prefijo con el tipo de retorno
- Posterior al prefijo, nombre del archivo adonde esta definida.
- “prv” – Funciones Estáticas

Memoria

RAM	FLASH
<p>Datos (Variables Globales)</p> <p>Ejecución de código temporal (parches)</p> <p>Stack:</p> <ul style="list-style-type: none">- Variables Locales,- Argumentos de Funciones- Direcciones de Retorno <p>Heap (Memoria Dinámica)</p>	<p>Código de Aplicación</p> <p>Constantes</p> <p>Tabla de Interrupciones y Expciones</p>

Memoria RAM

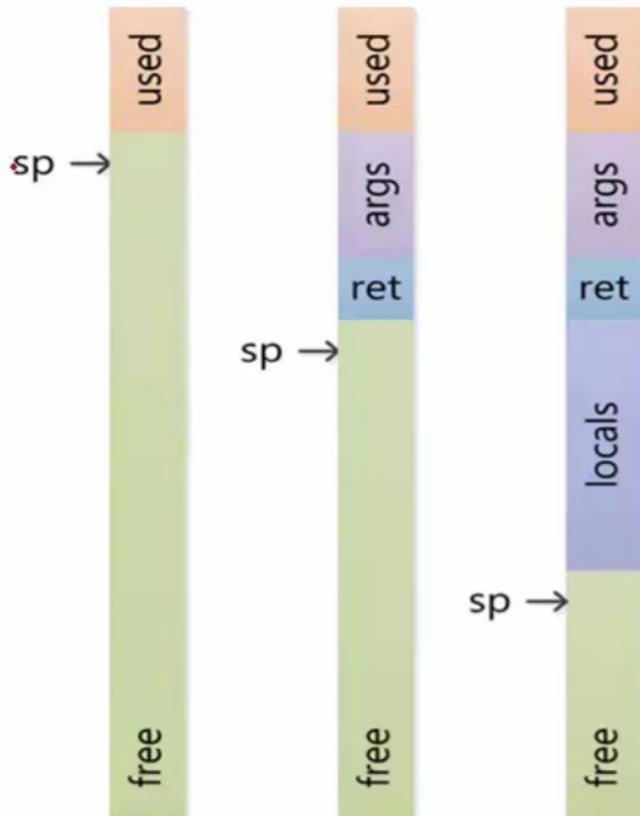
Stack and Heap in embedded Systems



- **Stack Pointer (SP):** Antes de ejecutar la primera instrucción, se inicializa el SP con la dirección más alta de la RAM

Memoria RAM: Stack

Stack



sp : Stack Pointer
used : Unavailable stack (already used)
args : function arguments
ret : return address
locals : local variable
free : available stack

Memoria FILO
Acceso Secuencial

Memoria RAM: Stack

Stack

```
char do_sum(int a, int b, int c)
{
    char x=1;
    char y=2;
    char *ptr; } Local variables

ptr = malloc(100);
x = a+b+c; } Some operations

return x; } Exiting [R0 = x]
```

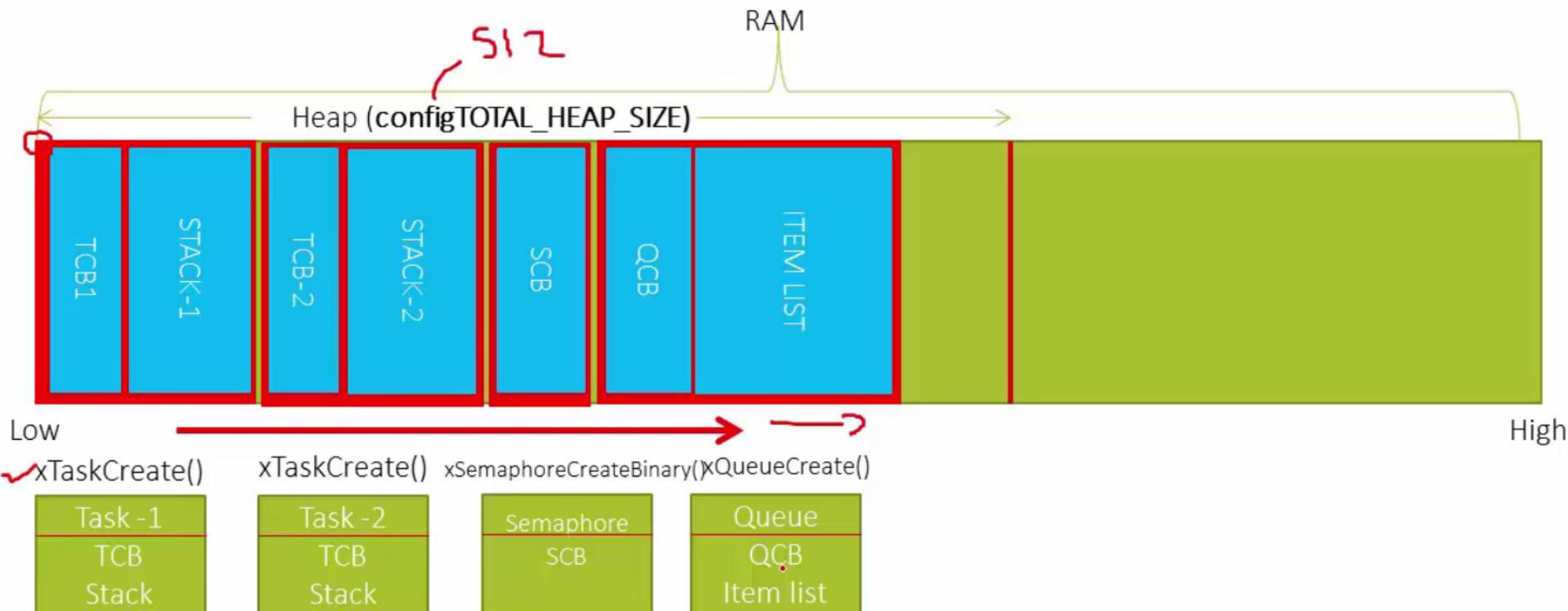


Memoria RAM: Heap

- Memoria Heap: Acceso Aleatorio (No existe Heap Pointer)
 - C: Malloc() y Free()
 - C++: New y Delete
- Para embebidos, malloc y free tienen las contras:
 - Falta de Determinismo
 - Fragmentación de Memoria
 - Footprint elevado

Memoria RAM: Heap

FreeRTOS Stack and heap



- Por defecto, el kernel indica el inicio de la Heap, para hacerlo manual:
 - configAPPLICATION_ALLOCATED_HEAP 1

Memoria RAM: Heap

FreeRTOS Heap management Schemes

heap_1.c

heap_2.c

heap_3.c

heap_4.c

heap_5.c

Your_own
_mem.c

pvPortMalloc()

pvPortMalloc()
vPortFree()

pvPortMalloc()
vPortFree()

pvPortMalloc()
vPortFree()

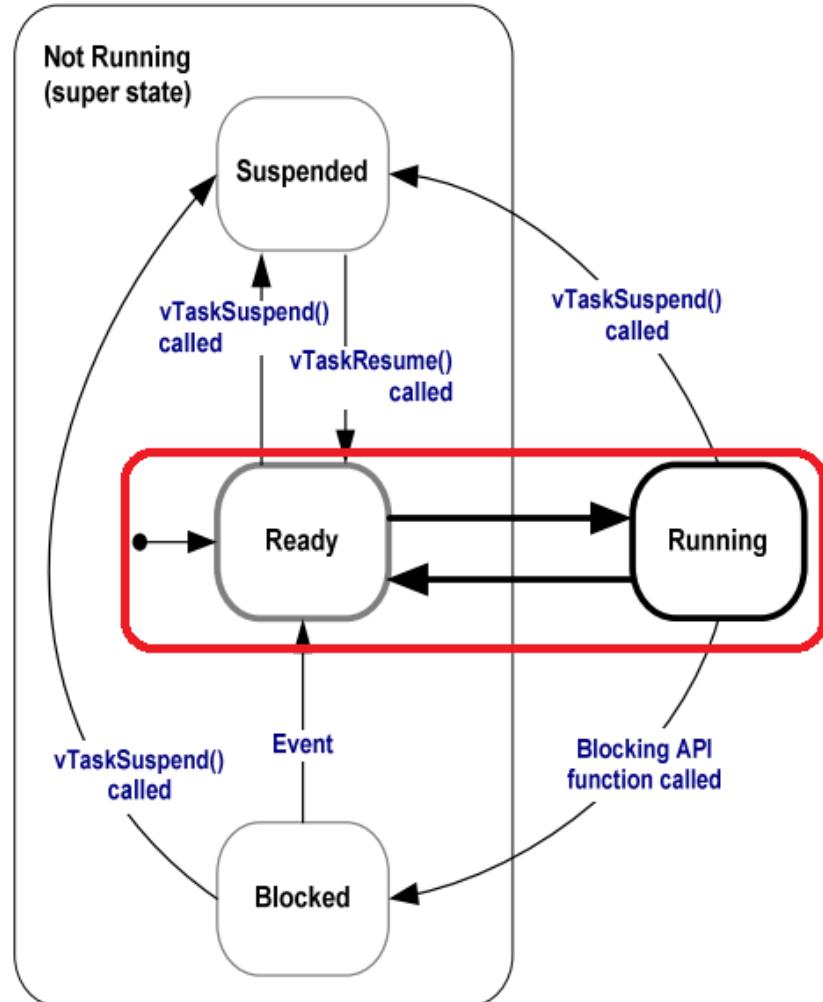
pvPortMalloc()
vPortFree()

pvPortMalloc()
vPortFree()

Application uses any one of these Schemes according to its requirements

FreeRTOS APIs and
Applications

FreeRTOS: Tareas



- `xTaskCreate()`
- `vTaskStartScheduler()`

FreeRTOS: Ejemplo de main

```
// Punto de entrada de la aplicación con FreeRTOS
int main( void )
{
    // Creo las dos tareas
    xTaskCreate( vTask1, "Task 1", 240, NULL, 1, NULL );
    xTaskCreate( vTask2, "Task 2", 240, NULL, 2, NULL );

    // Inicio el scheduler
    vTaskStartScheduler();

    // No se llega a este punto si no hubo problemas al
    // iniciar el scheduler
    for( ; ; );
    return 0;
}
```

FreeRTOS: Ejemplo de Tarea

```
void vTask1(void * pvParameters) {  
    char *pcTaskName;  
    pcTaskName = (char *) pvParameters;  
    for (;;) {  
        // Task Code  
    }  
}
```

FreeRTOS: Manejo del Tiempo

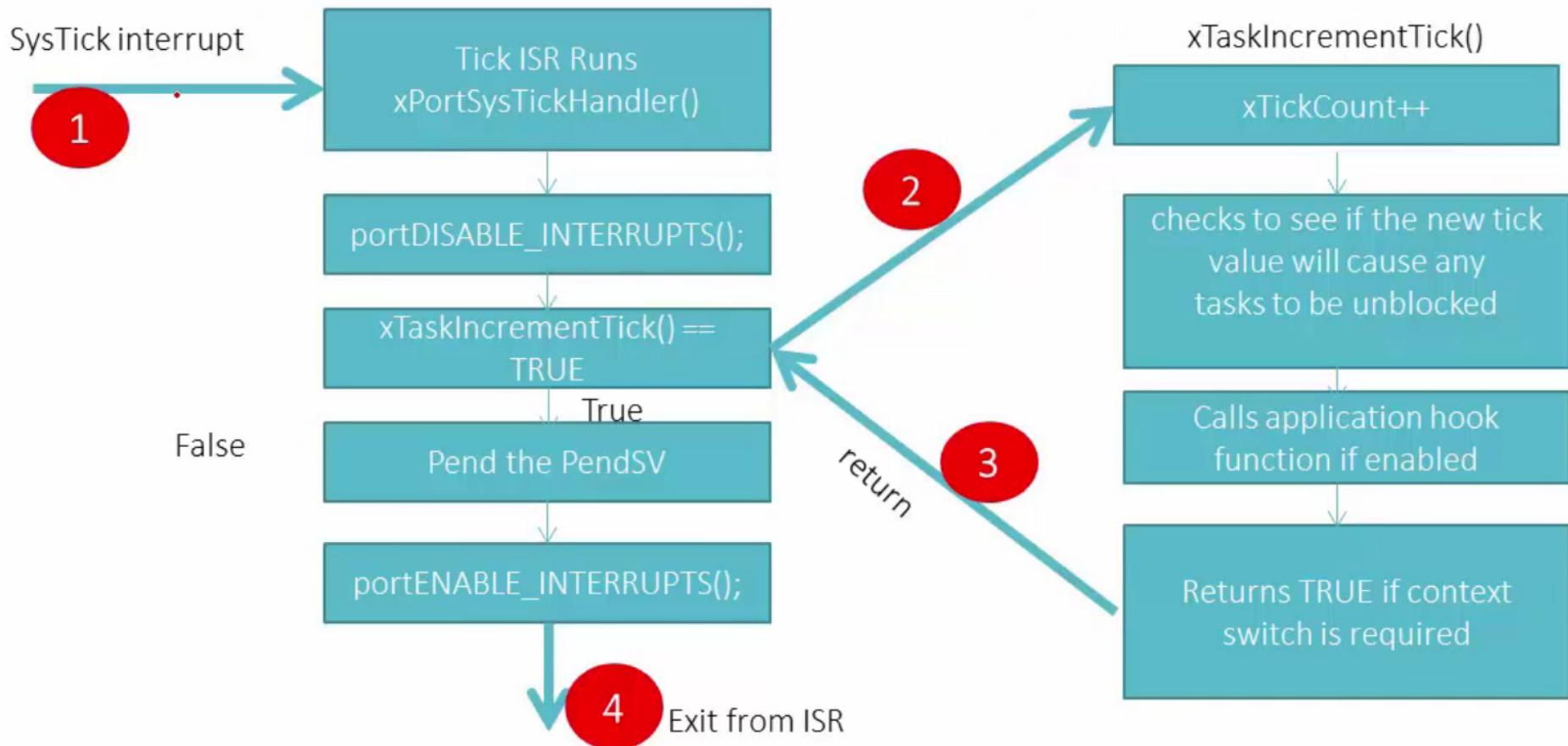
- OS Tick Interrupt (**TickISR**)
 - configTICK_RATE_HZ
 - Con cada interrupción se incrementa **tick_count**
 - Usada para:
 - Scheduling / cambio de context
 - Cálculo de delays
 - Función que ejecuta la ISR:
 - xPortSysTickHandler()

FreeRTOS: Manejo del Tiempo

- OS Tick Interrupt (**TickISR**)
 - xPortSysTickHandler()
 1. Se escanean todas las tareas en Ready en búsqueda de la de mayor prioridad
 2. Se determina la próxima tarea a ser ejecutada
 3. Si hay necesidad de cambio de tarea se ejecuta el handler de cambio de contest (**PendSV**).

FreeRTOS: Manejo del Tiempo

What RTOS tick ISR does ? : Conclusion

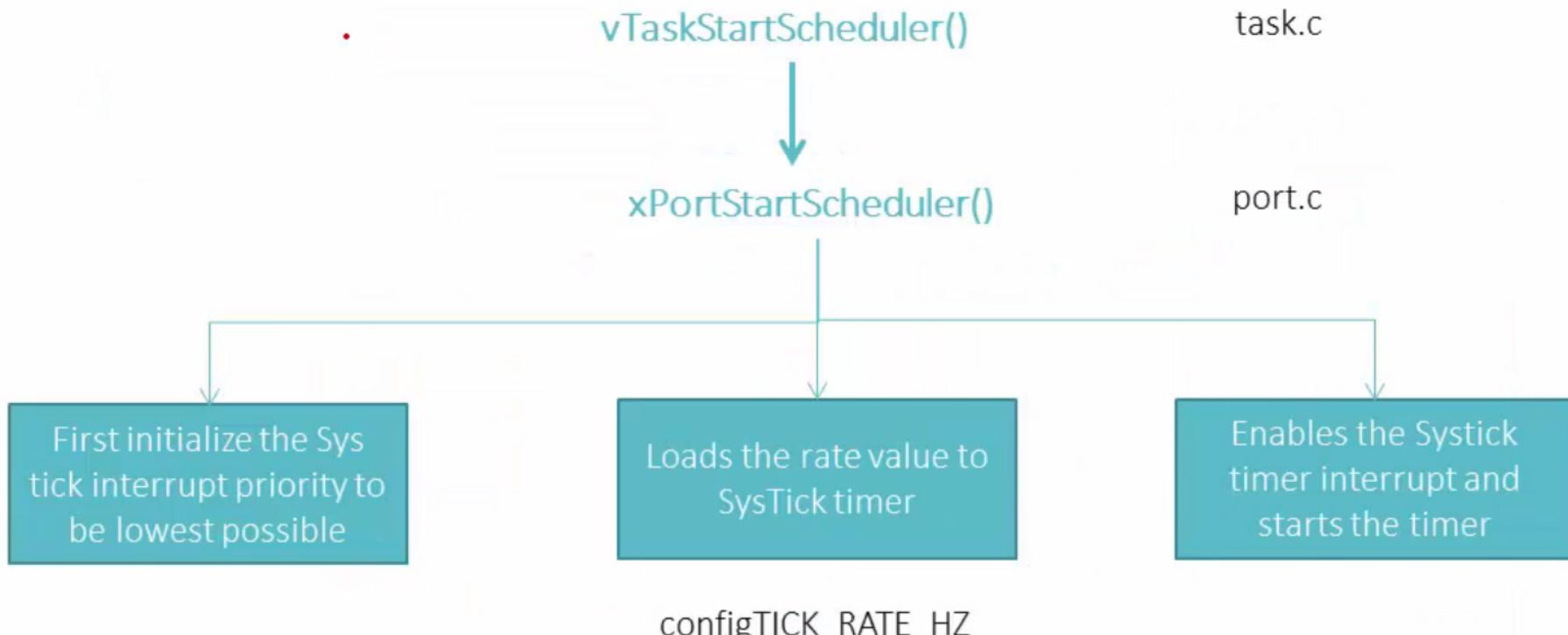


FreeRTOS: Manejo del Tiempo

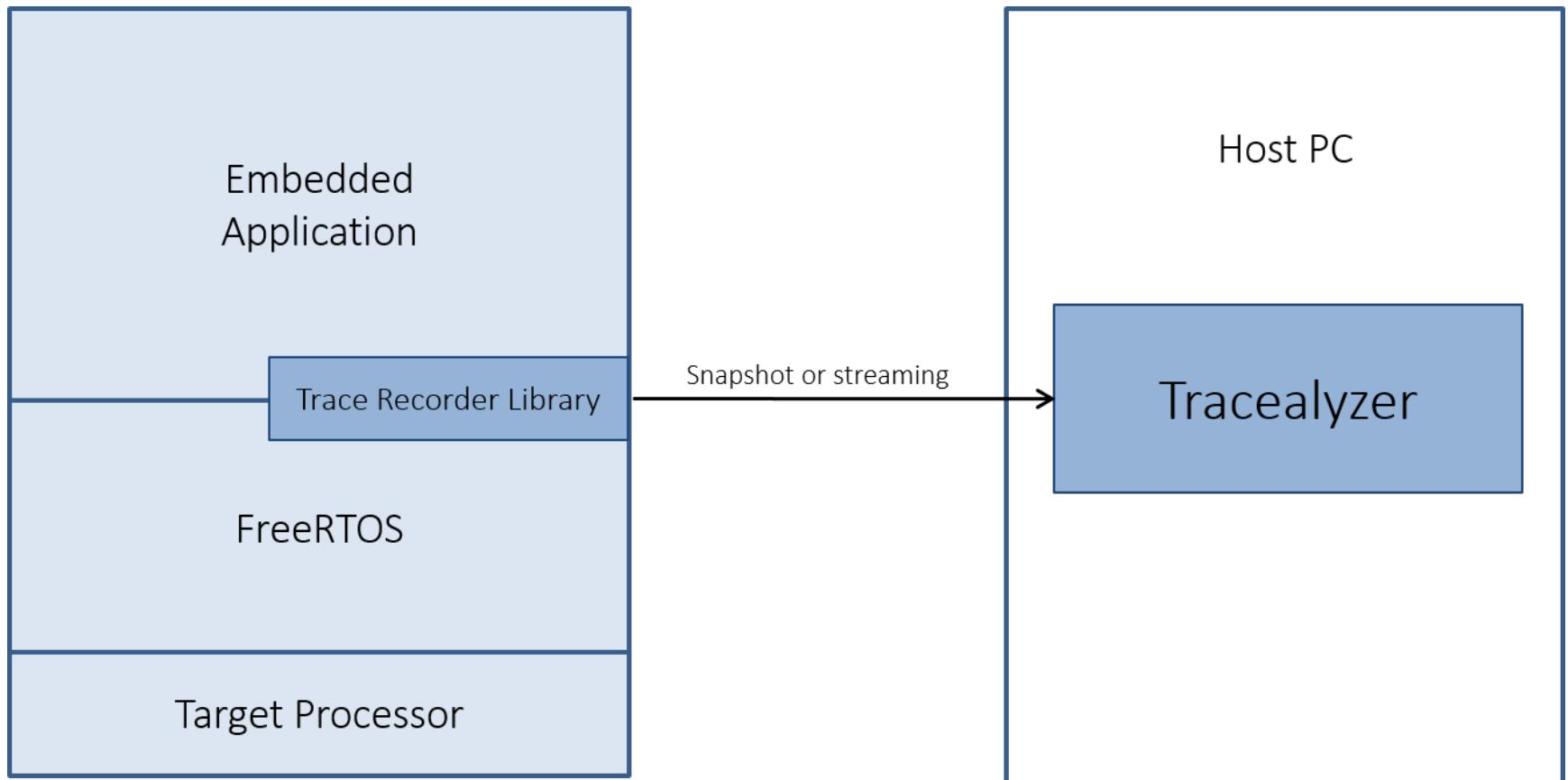
- En Cortex M, el HW que dispara el Tick es el “SysTickTimer”.

FreeRTOS: Manejo del Tiempo

Who configures OS tick Timer ?



Debug + Trace: *Tracealyzer*

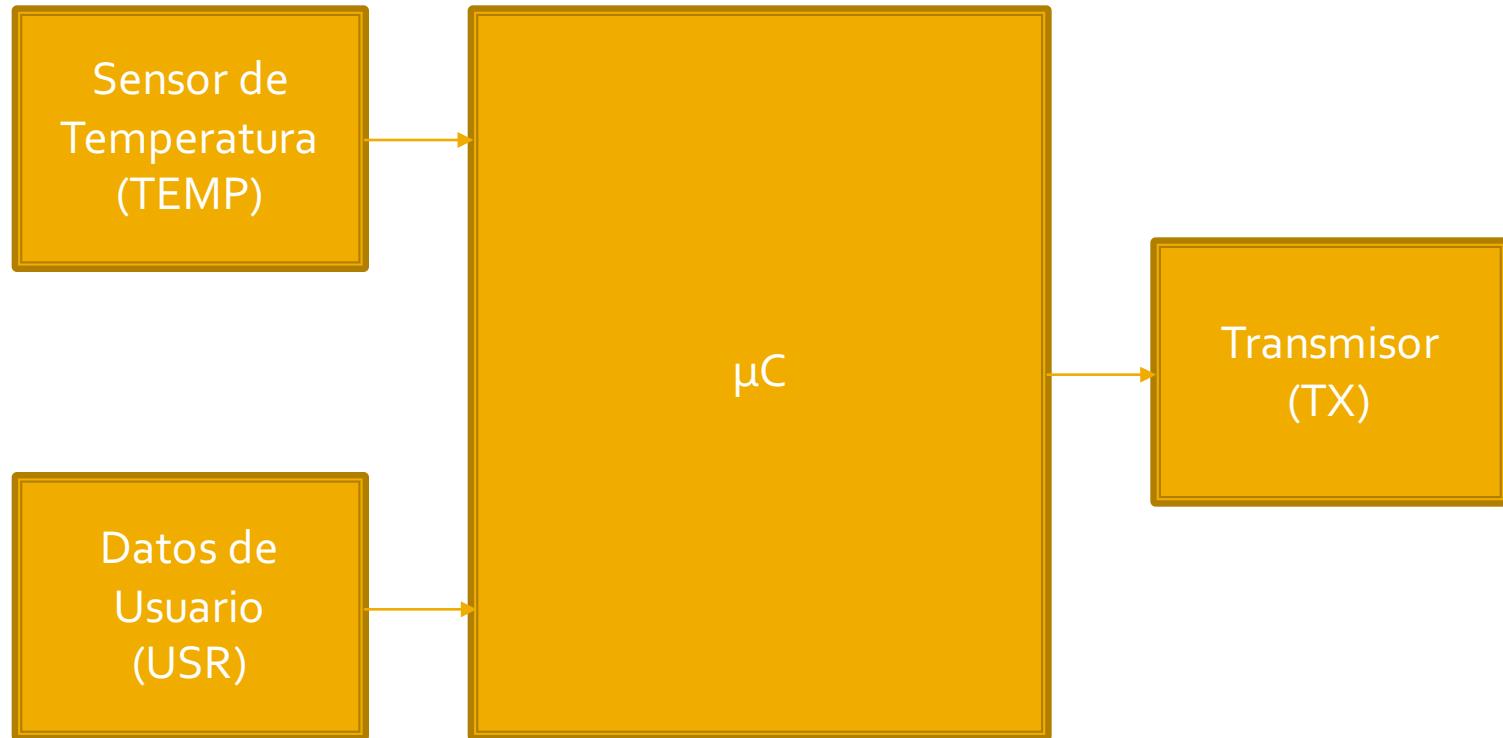


FreeRTOS: Trabajo Practico

1. Instalar FreeRTOS en cualquier placa de desarrollo con CortexM.
2. Ejecutar 2 tareas simples y realizar un análisis completo del Sistema con Tracealyzer (tiempos de ejecución, memoria).

FreeRTOS: Trabajo Practico

3.



TEMP: 1 byte por lectura

USR: Tamaño variable por cada ingreso