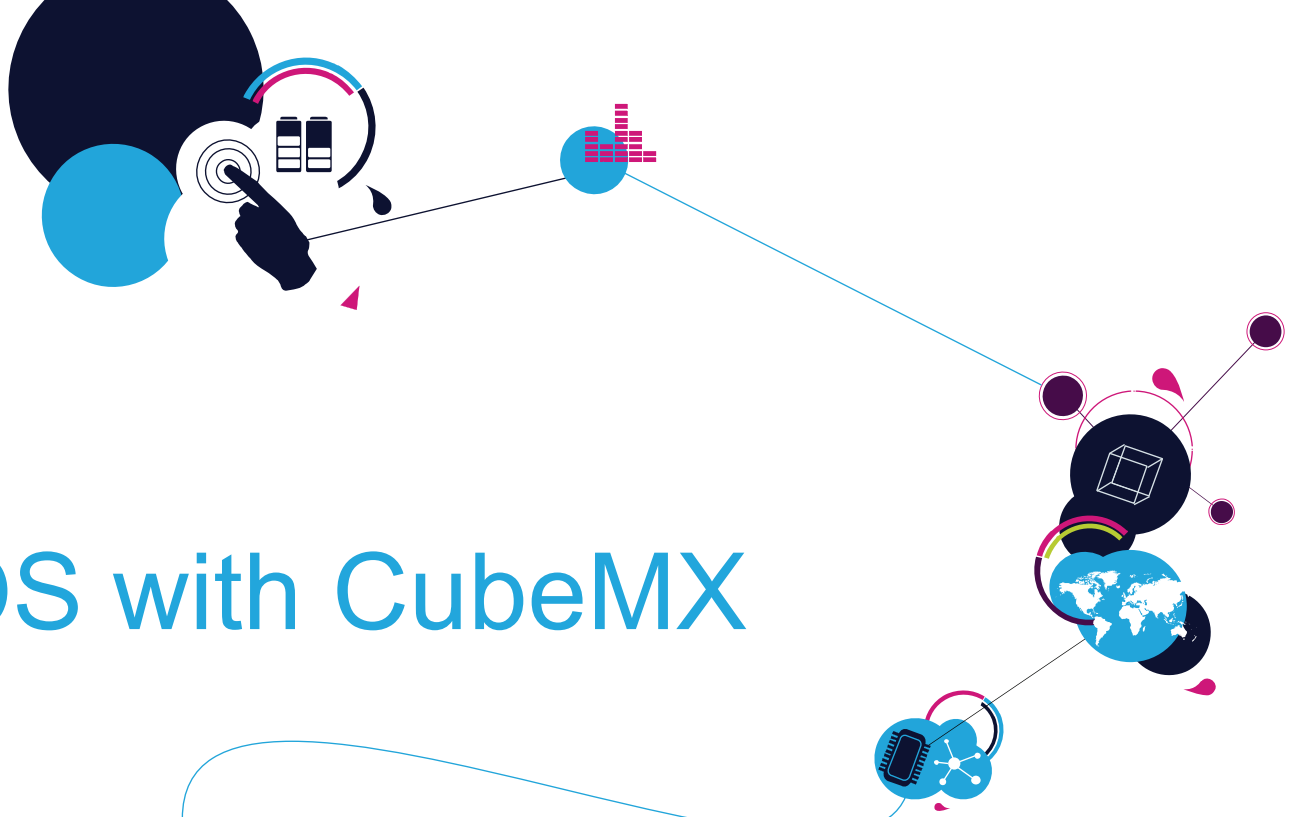


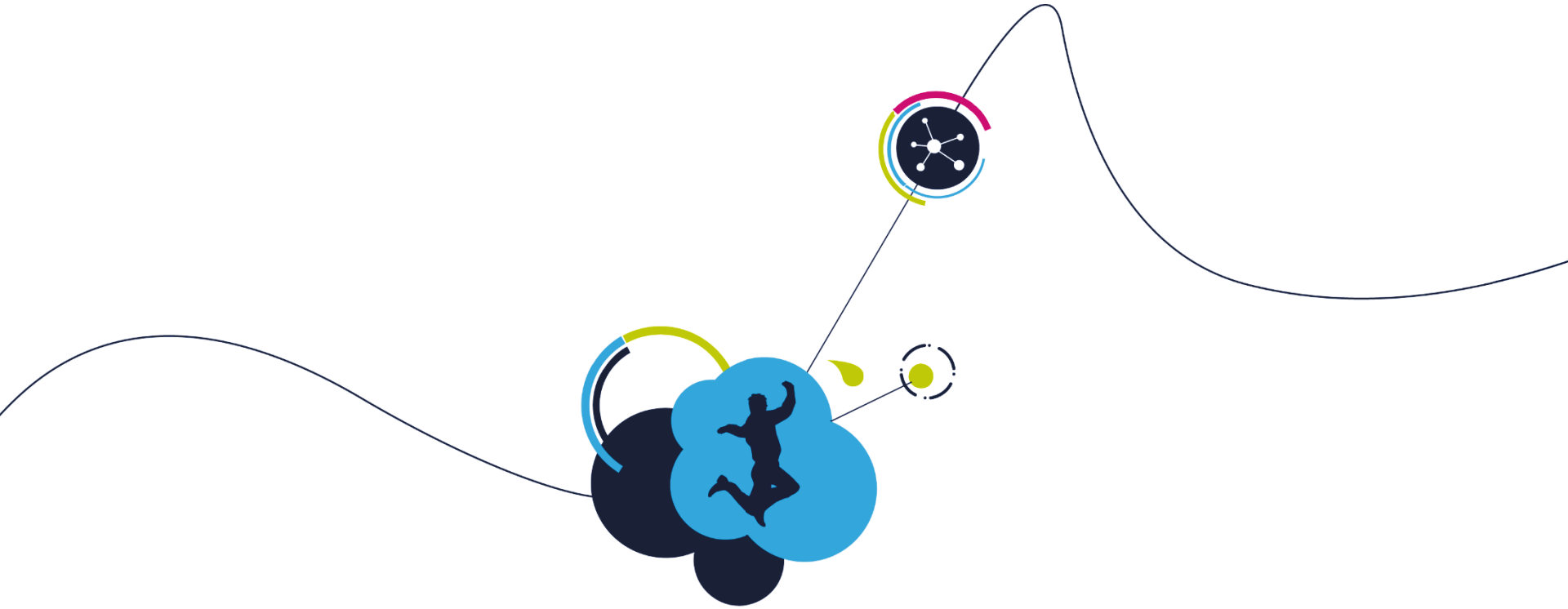
FreeRTOS with CubeMX

T.O.M.A.S – Technically Oriented Microcontroller Application Services (v0.02)

Modified by MCD Korea team

life.augmented

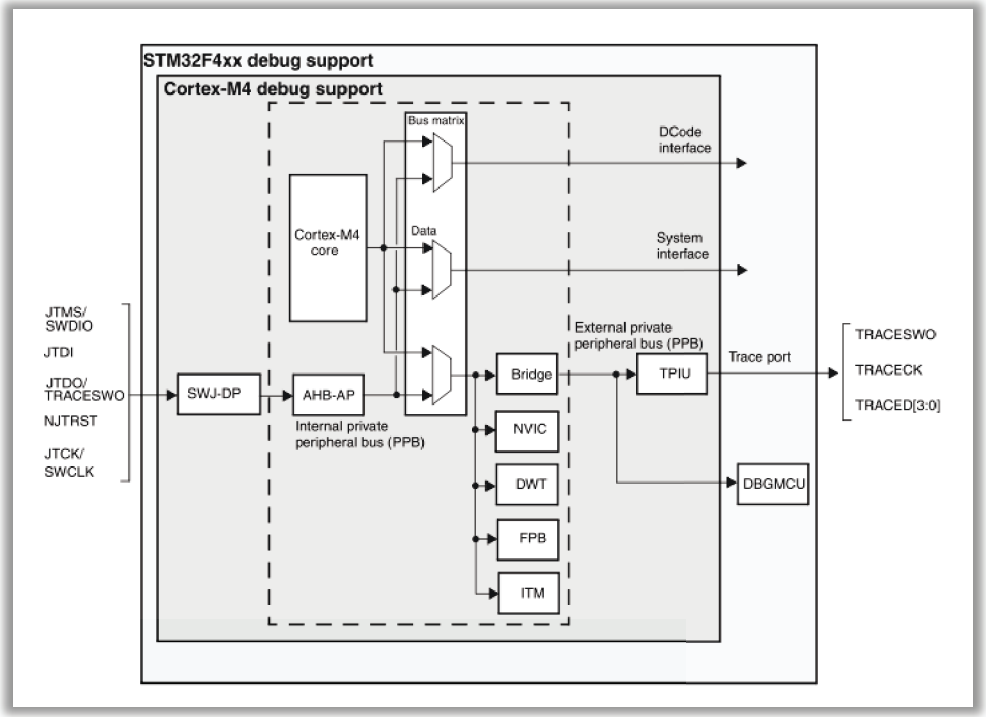
Contents



Using SWO to print info. from STM32

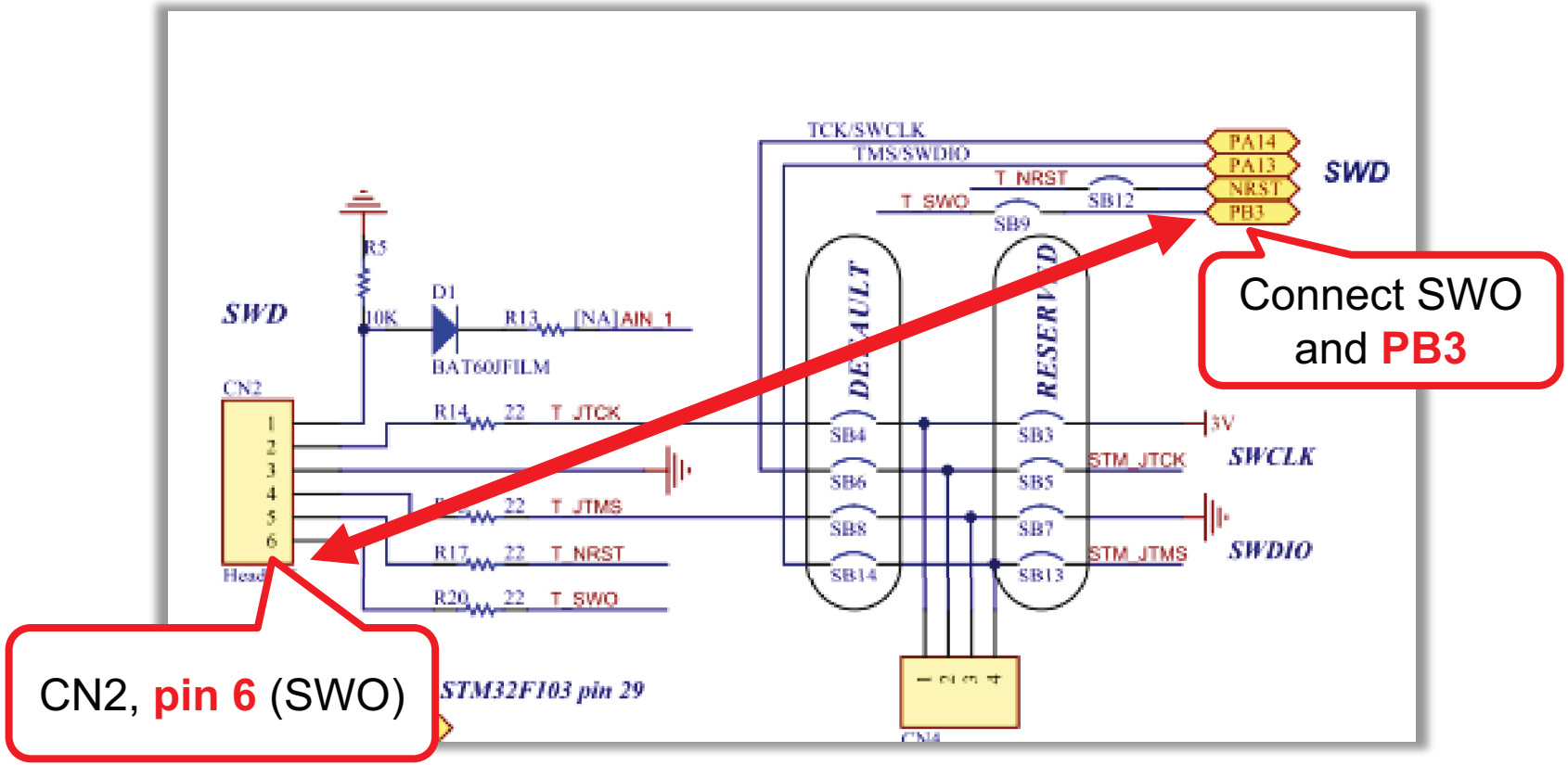
Using SWO

- On some STM32, there is periphery called ITM, not mix with ETM(real trace)
- This periphery can be used to internal send data from MCU over SWO pin
- Is possible to redirect the printf into this periphery
- And also some IDEs can display this information during debug
- It is similar to USART but we don’t need any additional wires and PC terminal



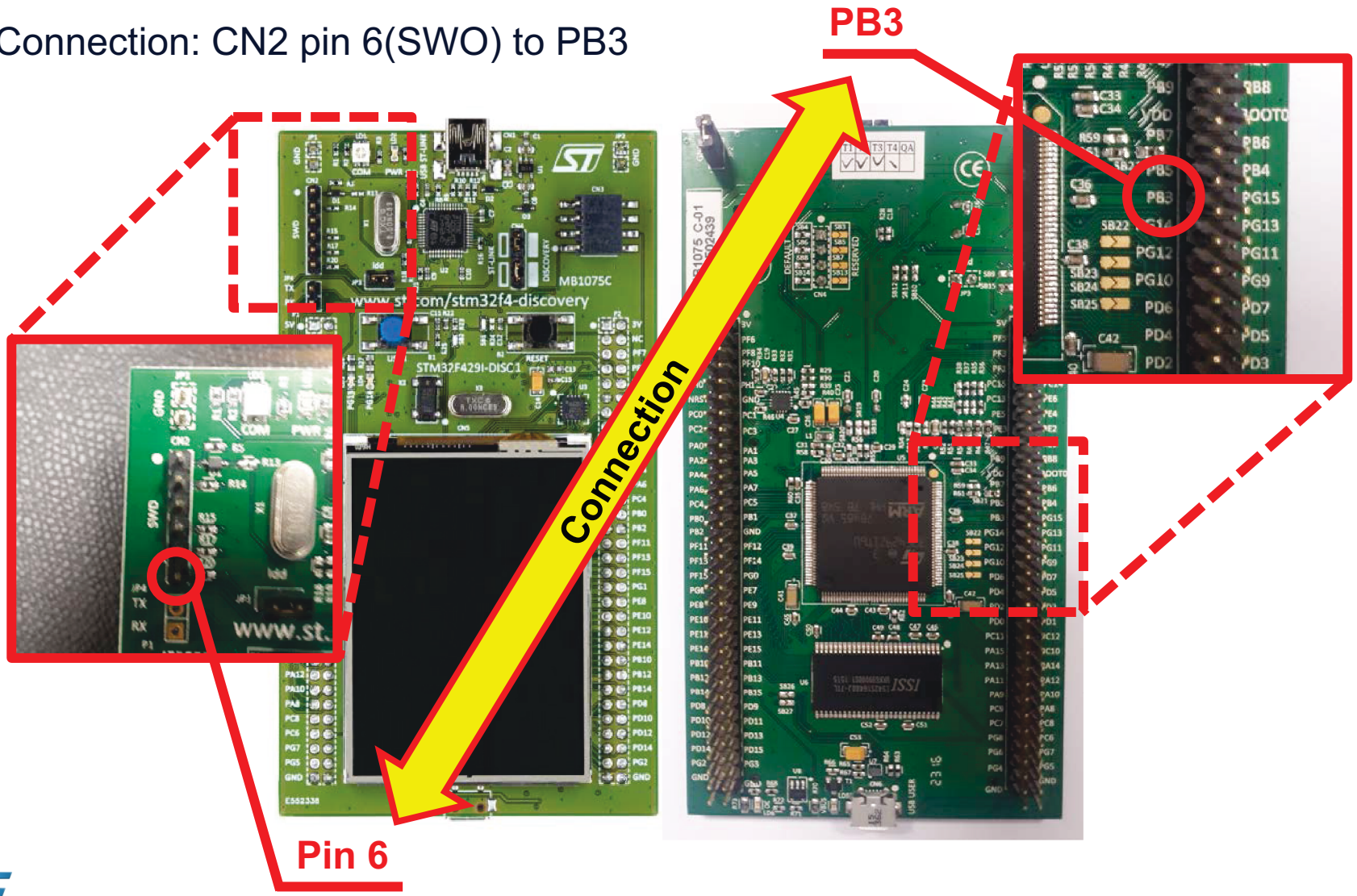
Using SWO

- To make SWO working you must connect the PB3 and Debugger SWO pin together



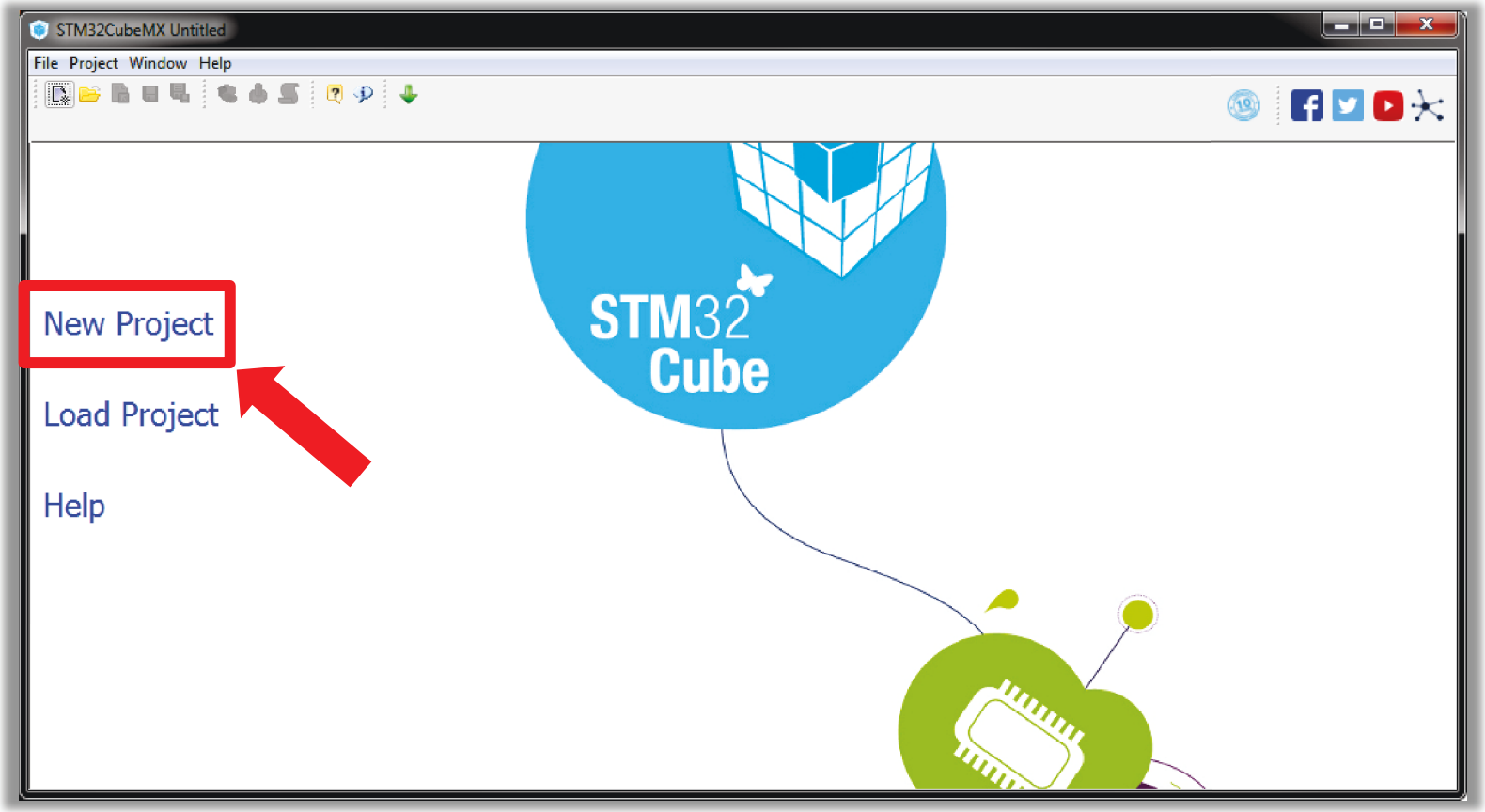
Using SWO

- Connection: CN2 pin 6(SWO) to PB3



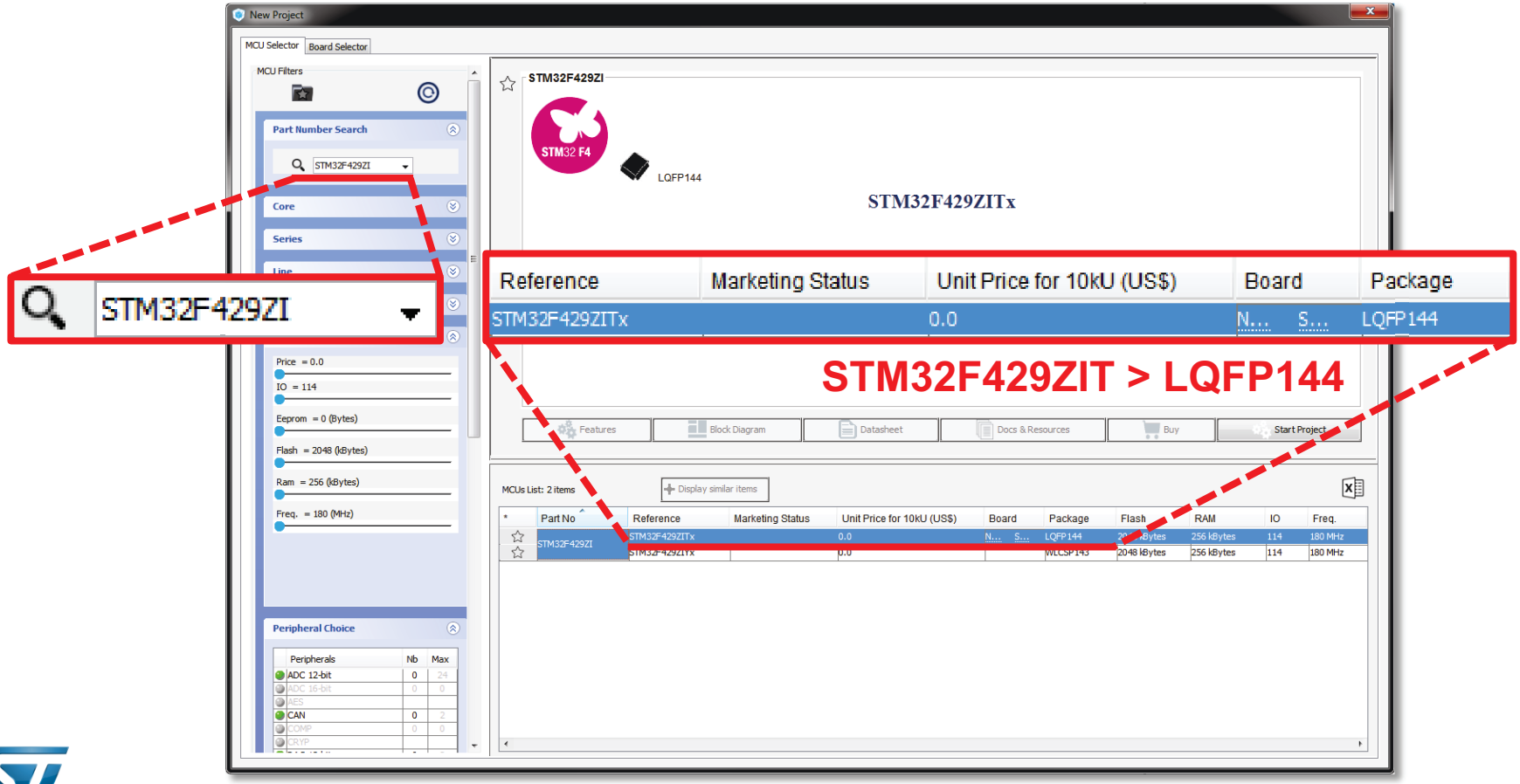
Using SWO for printf

- Create project in CubeMX
  - Menu > File > New Project



Using SWO for printf

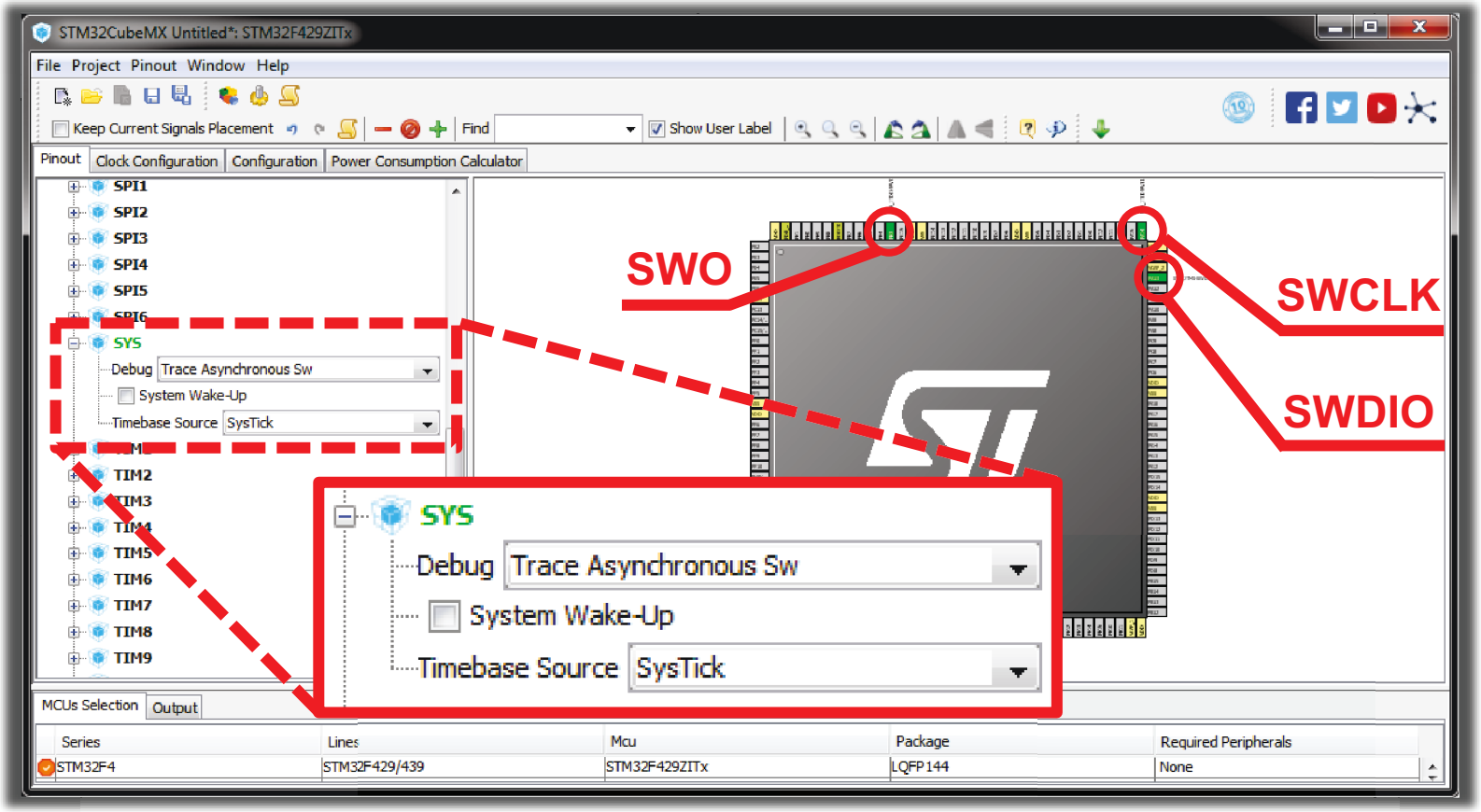
- Create project in CubeMX
  - Searching “**STM32F429ZI**”
  - Select “**STM32F429ZITx**” **(LQFP144 package)**



life.augmented

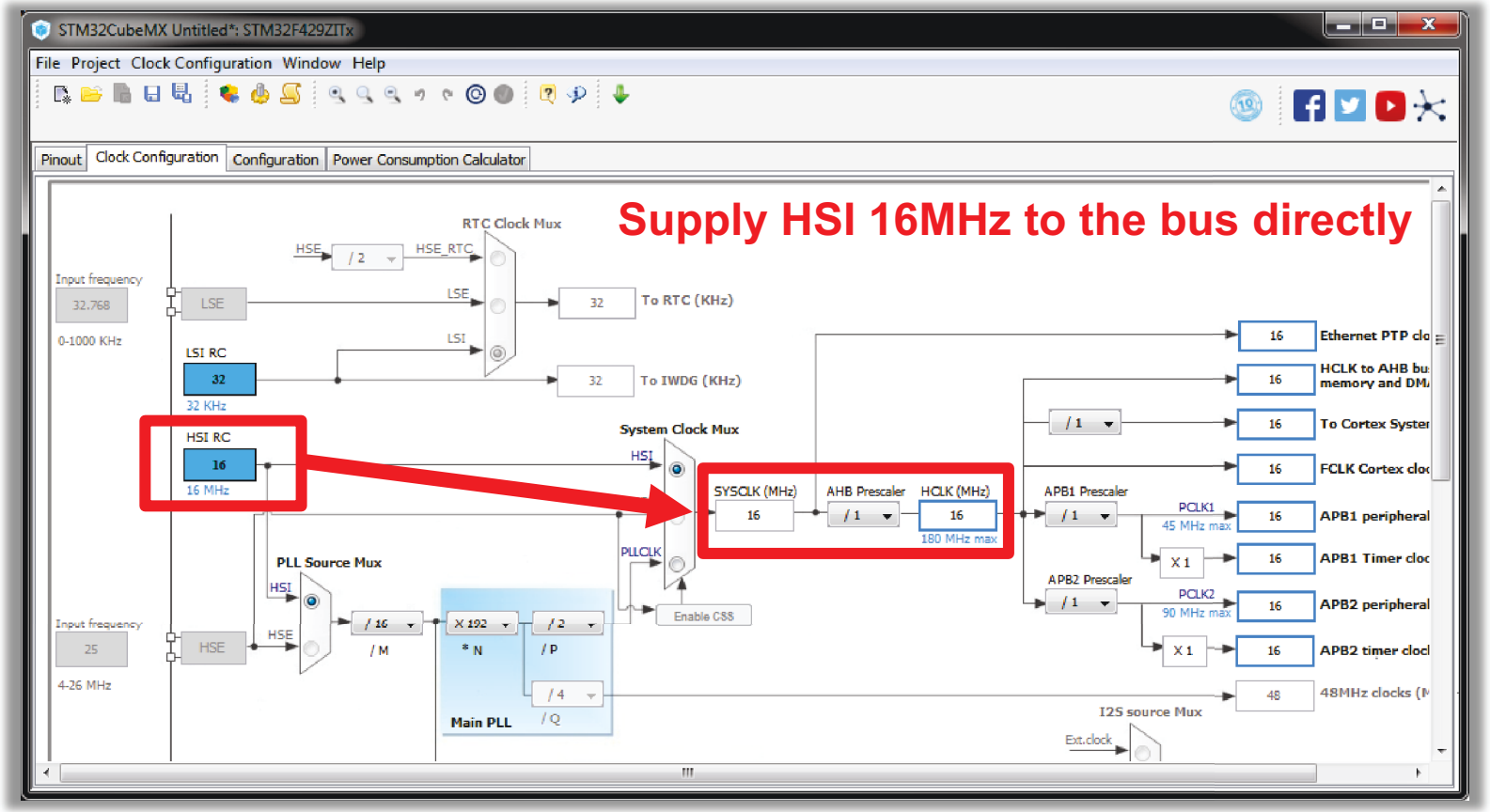
Using SWO for printf

- Pinout
  - We set the **“Debug”** to **“Trace Asynchronous Sw”**



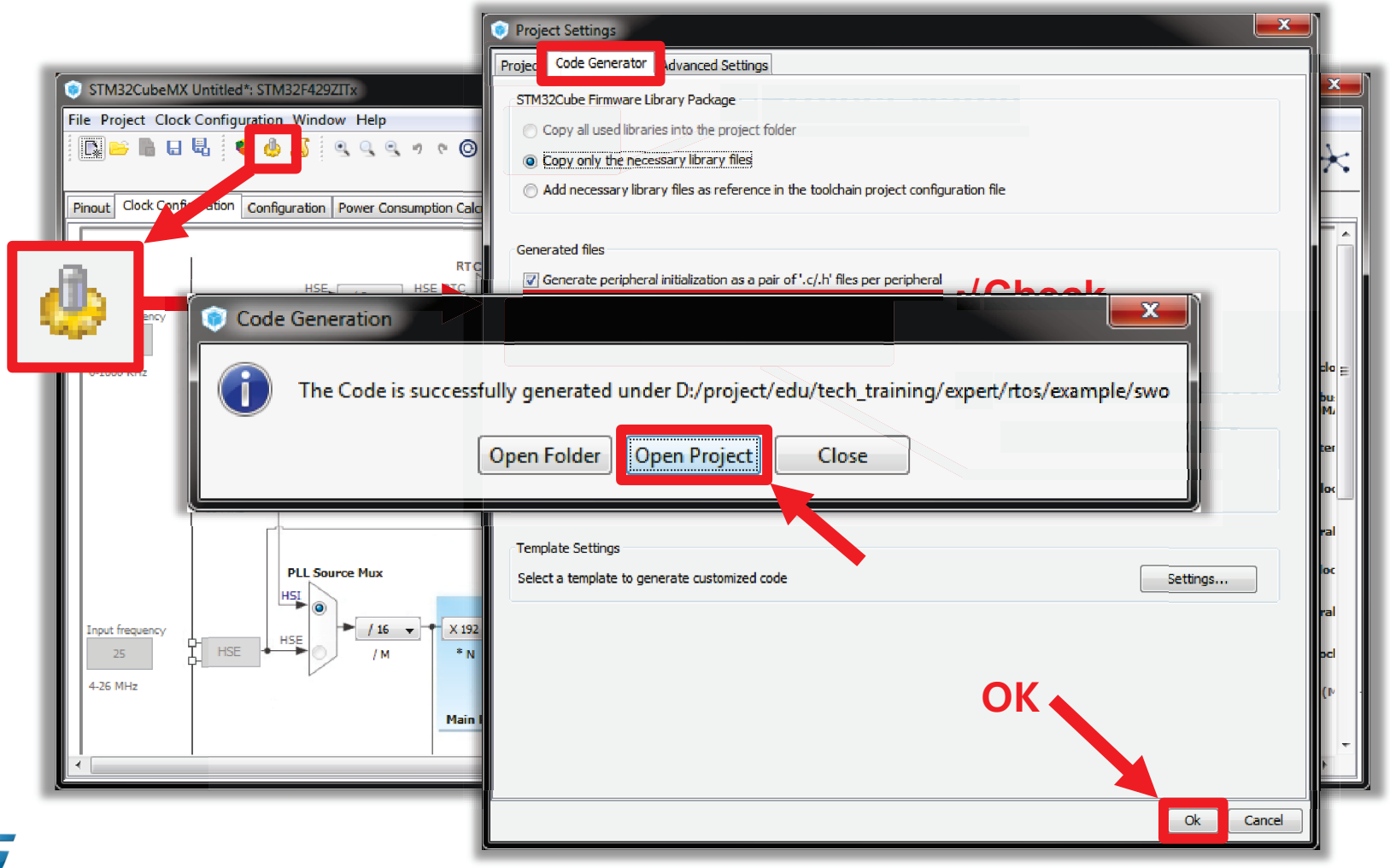
Using SWO for printf

- Clock Configuration
  - We need only blank project with clock initialization (keep core speed to default)



Using SWO for printf

- Now we set the project details for generation



Using SWO for printf in Atollic TrueSTUDIO

- To make printf() output it's data to the TrueSTUDIO debugger, simply change the _write(), like this:
  - **PATH: /Src/main.c**

```
/* USER CODE BEGIN 4 */
int _write(int pf, char *p, int len)
{
  /* Implement your write code here, this is used by puts and printf */
  for (int i = 0; i < len; i++) {
    ITM_SendChar(*p++);
  }
  return len;
}
/* USER CODE END 4 */
```

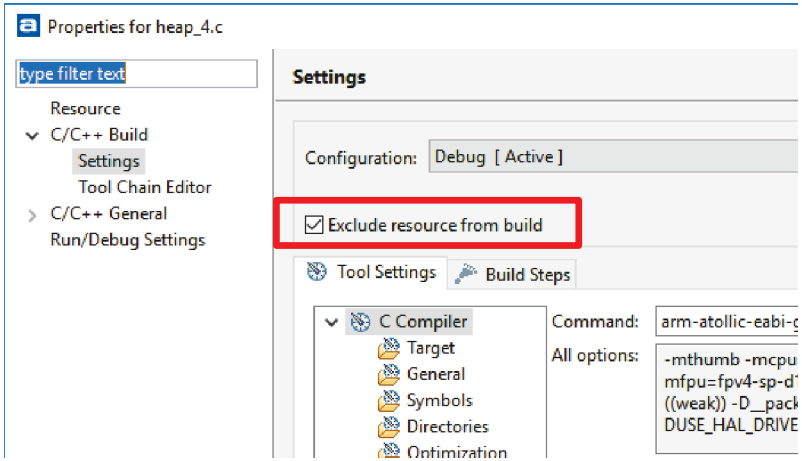
  - **ITM_SendChar() function is CMSIS code**

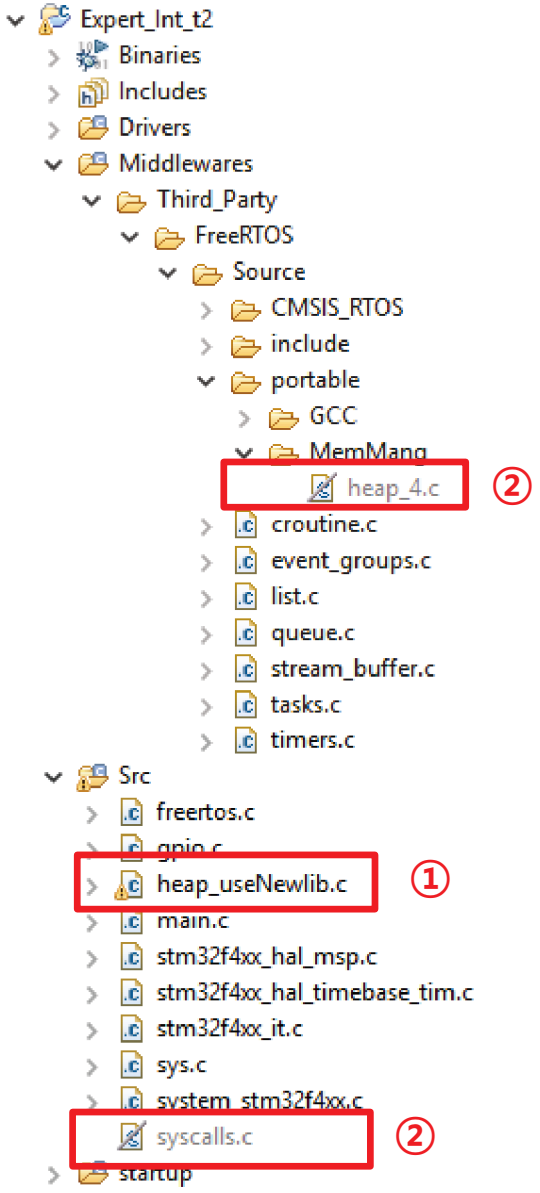
Using SWO for printf in Atollic TrueSTUDIO

- Temporary patch for FreeRTOS and Newlib issue:

  (1) Add (copy) “heap_useNewlib.c” to Src folder

  (2) Right click on “heap_4.c” and “syscalls.c”, and check “Exclude resource from build”





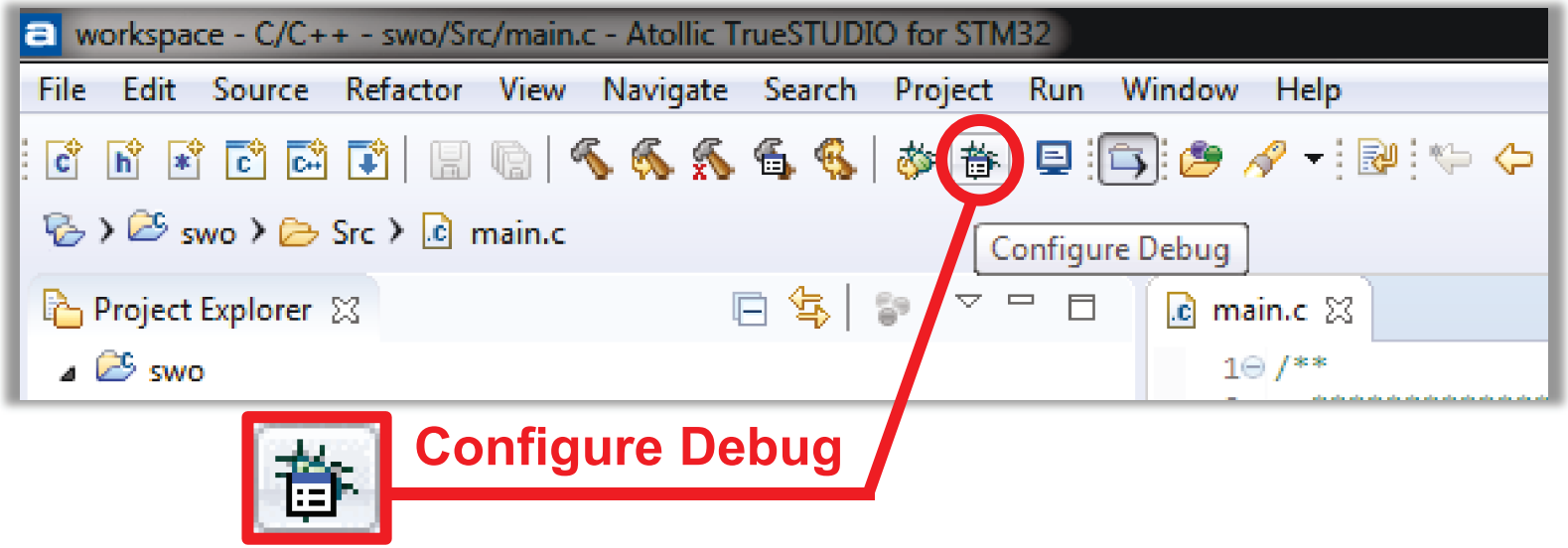
(3) Add define “#define configUSE_NEWLIB_REENTRANT 1” in FreeRTOSConfig.h

```
126 /* USER CODE BEGIN Defines */
127 /* Section where parameter definitions can be added
128 #define configUSE_NEWLIB_REENTRANT 1
129 /* USER CODE END Defines */
130
131 #endif /* FREERTOS_CONFIG_H */
```

Using SWO for printf in Atollic TrueSTUDIO

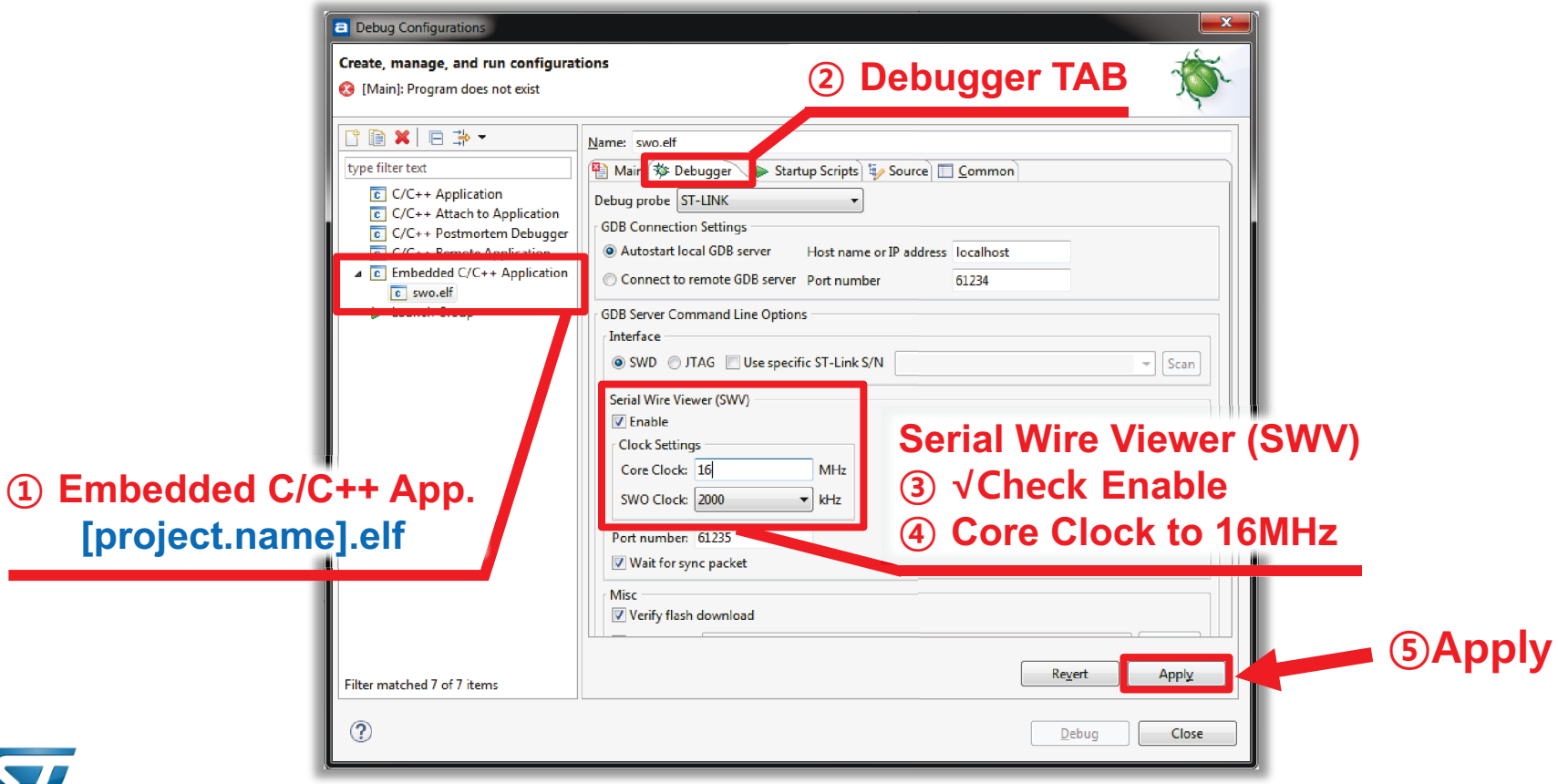
- Enable SWV tracing
  - Run [Configure Debug] → [Debugger]

    ※ *"Run" → "Debug Configurations…"*



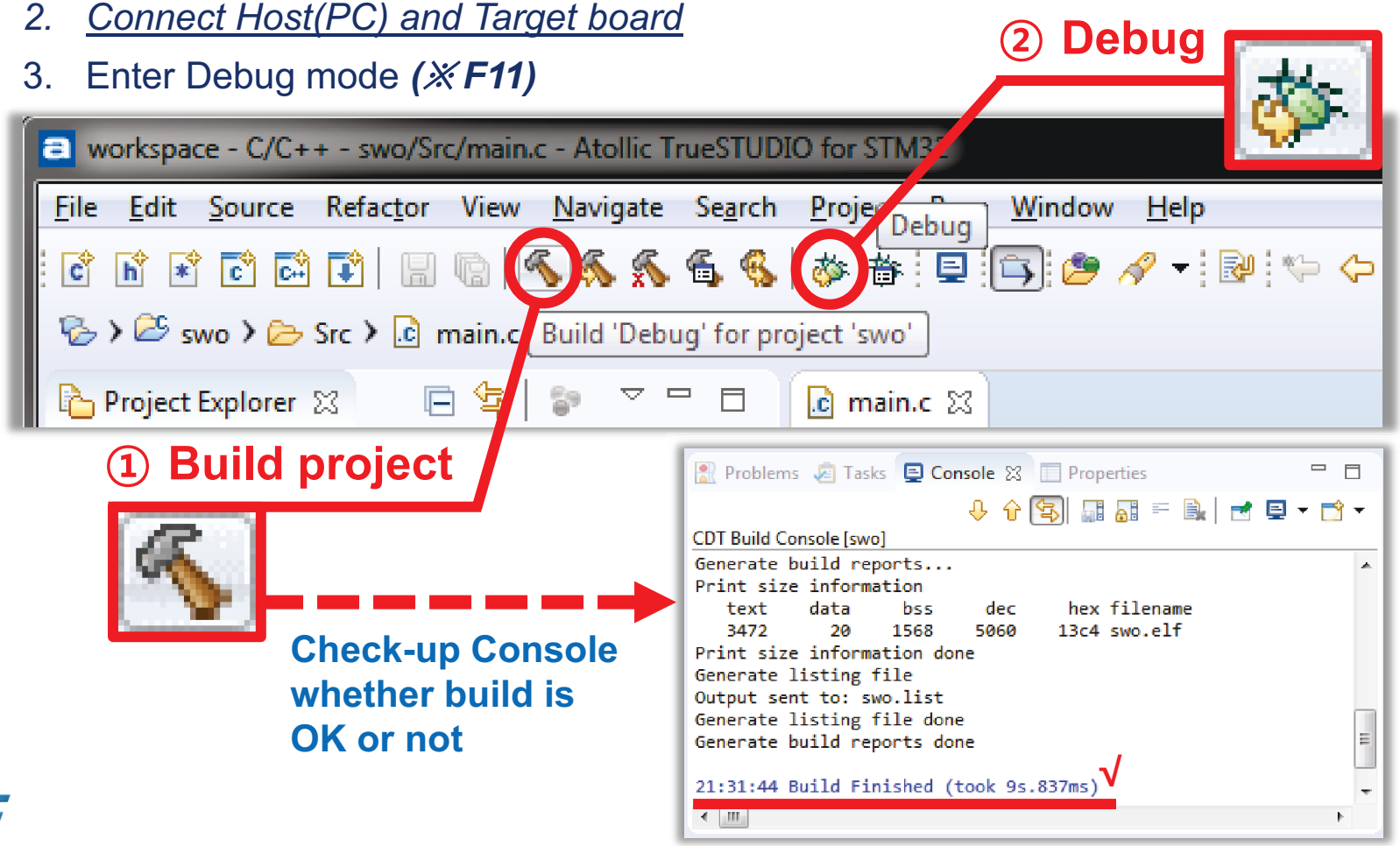
Using SWO for printf in Atollic TrueSTUDIO

- Enable SWV tracing
  - Check [Enable] checkbox in Serial Wire Viewer
  - Set a core clock according to system clock of microcontroller



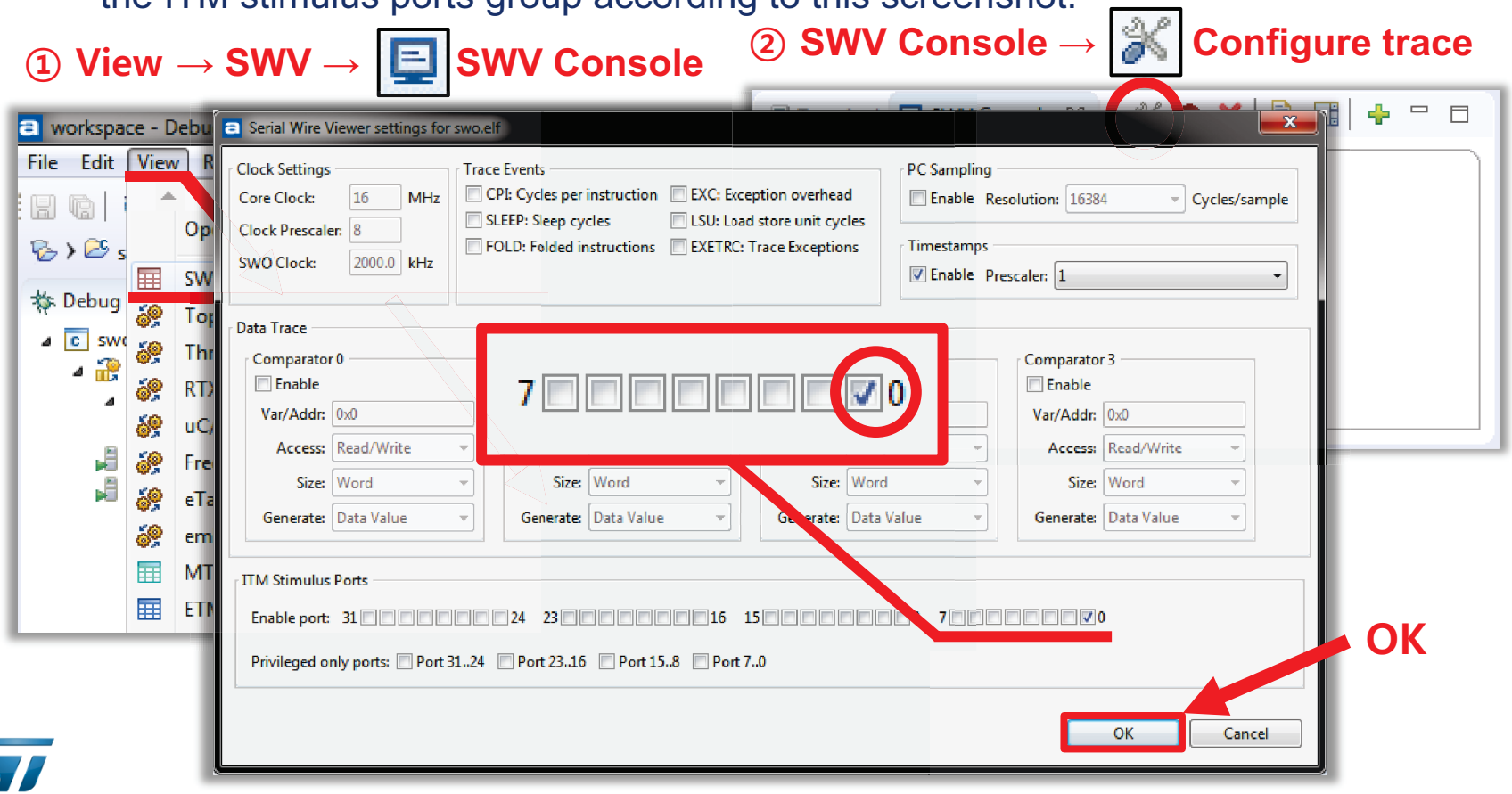
Using SWO for printf in Atollic TrueSTUDIO

- Configure SWV for ITM
  1. Build Project ***(※Ctrl + B),.*** You can also find related menu at "*Project*" → *{Build options}*
  2. *Connect Host(PC) and Target board*
  3. Enter Debug mode ***(※F11)***



Using SWO for printf in Atollic TrueSTUDIO

- Configure SWV for ITM
  - In the SWV Console configure window,
    Configure the data ports to be traced by **enabling the ITM channel 0 check box** in the ITM stimulus ports group according to this screenshot.



life.augmented

Using SWO for printf in Atollic TrueSTUDIO

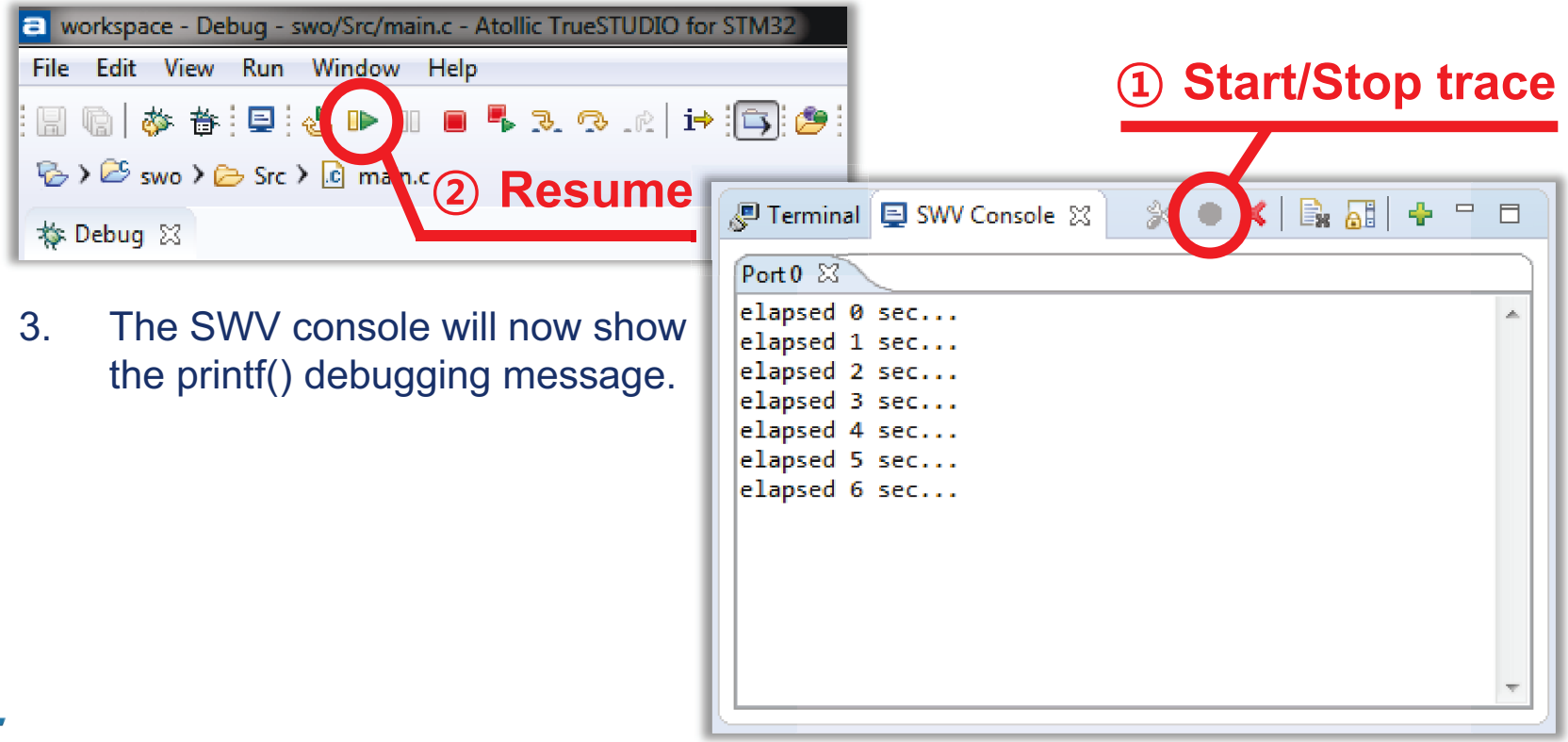
- SWO Test code
  - **FUNCTION: main()@/Src/main.c**

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
int count = 0;
while (1)
{
  HAL_Delay(1000);
  printf("elapsed %d sec...\n", count++);
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
```

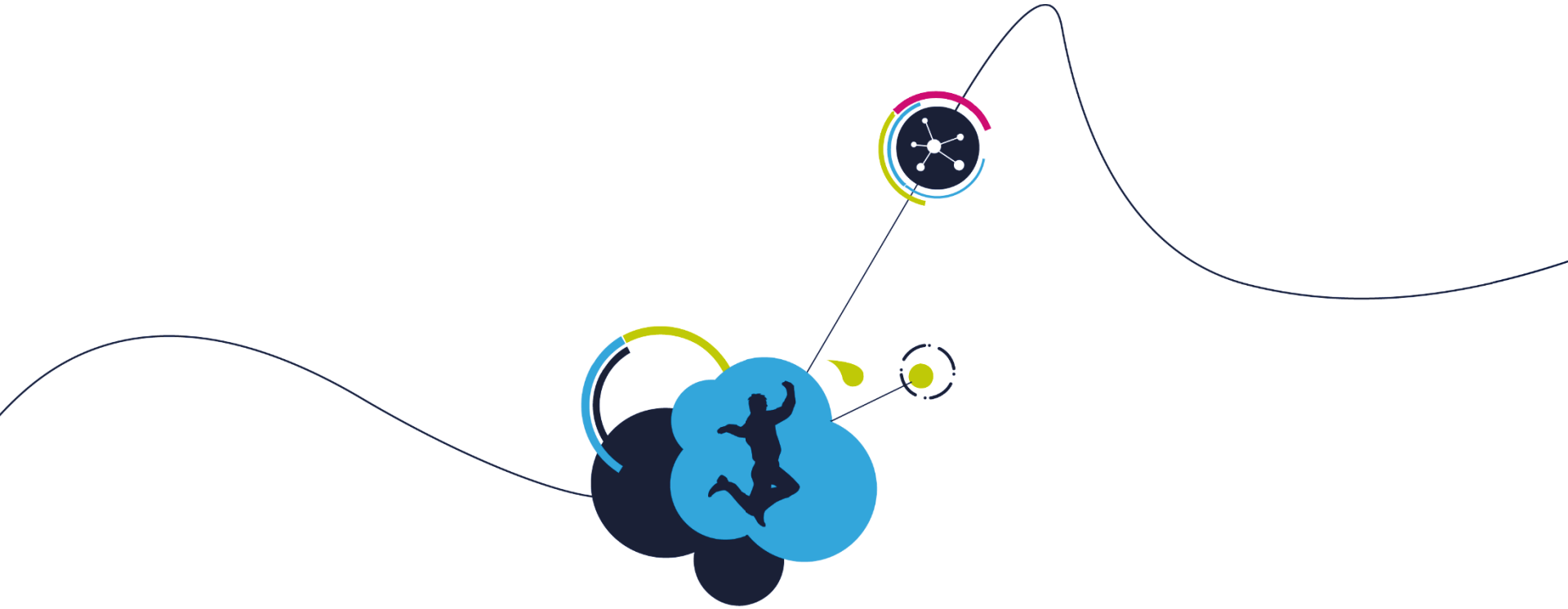
- Rebuild and Re-enter the Debug mode for SWO testing
  1. Exit from Debug mode *(※**Ctrl + F2**)*
  2. Rebuild ***(※Ctrl + B)*** and re-enter the Debug mode ***(※F11)***

Using SWO for printf in Atollic

- Output debugging message
  1. Press the *red Start/Stop Trace button*  to send the SWV configuration to the target board.
  2. Start the target execution by pressing the *Resume Debug button* 



3. The SWV console will now show the printf() debugging message.



FreeRTOS

FreeRTOS
About FreeRTOS

- Market leading RTOS by Real Time Engineers Ltd.
- Professionally developed with strict quality management
- Commercial versions available: OpenRTOS and SafeRTOS
- Documentation available on www.freertos.org
- Free support through forum (moderated by RTOS original author Richard Barry)

FreeRTOS
Main features

- Preemptive or cooperative real-time kernel
- Scalable RTOS with tiny footprint (less than 10KB ROM)
- Includes a tickless mode for low power applications
- Synchronization and inter-task communications using
  - message queues
  - binary and counting semaphores
  - mutexes
  - group events (flags)
- Software timers for tasks scheduling
- Execution trace functionality
- CMSIS-RTOS API port

FreeRTOS APIs overview (1/2)

API category	FreeRTOS API	Description
Task creation	xTaskCreate	Creates a new task
	vTaskDelete	Deletes a task
Task control	vTaskDelay	Task delay
	vTaskPrioritySet	Sets task priority
	uxTaskPriorityGet	Get task priority
	vTaskSuspend	Suspends a task
	vTaskResume	Resumes a task
Kernel control	vTaskStartScheduler	Starts kernel scheduler
	vTaskSuspendAll	Suspends all tasks
	xTaskResumeAll	Resumes all tasks
	taskYIELD	Forces a context switch
	taskENTER_CRITICAL taskEXIT_CRITICAL	Enter(Exit from) a critical section (When entering, it stops context switching)

FreeRTOS APIs overview (2/2)

API category	FreeRTOS API	Description
Message Queue	xQueueCreate	Creates a queue
	xQueueSend	Sends data to queue
	xQueueReceive	Receive data from the queue
Semaphore Mutex	xSemaphoreCreateBinary	Creates a binary semaphore
	xSemaphoreCreateCounting	Creates a counting semaphore
	xSemaphoreCreateMutex	Creates a mutex semaphore
	xSemaphoreTake	Semaphore take
	xSemaphoreGive	Semaphore give
Timers	xTimerCreate	Creates a timer
	xTimerStart	Starts a timer
	xTimerStop	Stops a timer

CMSIS-RTOS FreeRTOS implementation

- Implementation in file ***cmsis-os.c***
  - "\Middlewares\Third_Party\FreeRTOS\Source\CMSIS_RTOS"

- The following table lists examples of the CMSIS-RTOS APIs and the FreeRTOS APIs used to implement them

API category	CMSIS-RTOS API	FreeRTOS API
Kernel control	osKernelStart	vTaskStartScheduler
Thread management	osThreadCreate	xTaskCreate
Semaphore	osSemaphoreCreate	vSemaphoreCreateBinary xSemaphoreCreateCounting
Mutex	osMutexWait	xSemaphoreTake
Message queue	osMessagePut	xQueueSend xQueueSendFromISR
Timer	osTimerCreate	xTimerCreate

- Note: CMSIS-RTOS implements same model as FreeRTOS for task states

FreeRTOS CMSIS-RTOS API

- CMSIS-RTOS API is a generic RTOS interface for Cortex-M processor based devices
- Middleware components using the CMSIS-RTOS API are RTOS agnostic, this allows an easy linking to any third-party RTOS
- The CMSIS-RTOS API defines a minimum feature set including
  - Thread Management
  - Kernel control
  - Semaphore management
  - Message queue and mail queue
  - Memory management
  - …
- For detailed documentation regarding CMSIS-RTOS refer to: http://www.keil.com/pack/doc/CMSIS/RTOS/html/index.html

FreeRTOS Configuration options

- Configuration options are declared in file FreeRTOSConfig.h
- Important configuration options are:

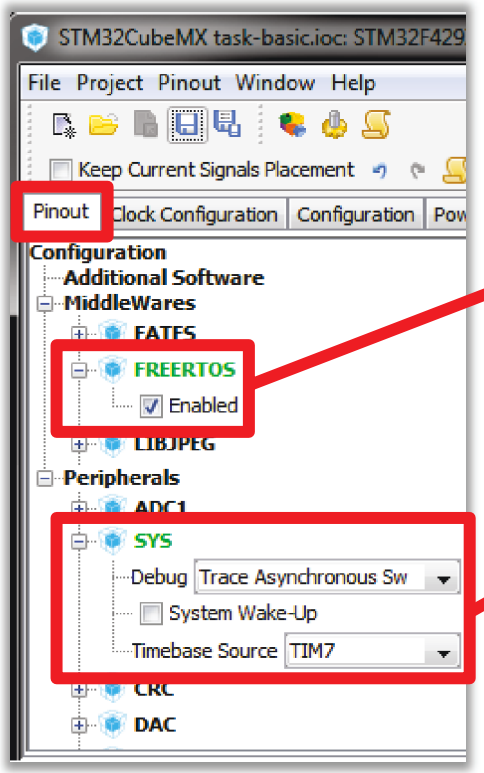
Config option	Description
configUSE_PREEMPTION	Enables Preemption
configCPU_CLOCK_HZ	CPU clock frequency in hertz
configTICK_RATE_HZ	Tick rate in hertz
configMAX_PRIORITIES	Maximum task priority
configTOTAL_HEAP_SIZE	Total heap size for dynamic allocation
configLIBRARY_LOWEST_INTERRUPT_PRIORITY	Lowest interrupt priority (0xF when using 4 cortex preemption bits)
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY	Highest thread safe interrupt priority (higher priorities are lower numeric value)

FreeRTOS
Tickless idle mode operation

- Kernel can stop system tick interrupt and place MCU in low power mode, on exit from this mode systick counter is updated
- Enabled when setting configUSE_TICKLESS_IDLE as 1
- The kernel will call a macro portSUPPRESS_TICKS_AND_SLEEP() when the Idle task is the only task able to run (and no other task is scheduled to exit from blocked state after n ticks)
  - n value is defined in FreeRTOSconf.h file
- FreeRTOS implementation of portSUPRESS_TICKS_AND_SLEEP() for cortexM3/M4 enters MCU in sleep low power mode
- Wakeup from sleep mode can be from a system interrupt/event

FreeRTOS in CubeMX

- Use CubeMX project from printf example
- *MiddleWares view in Pinout TAB*, select in **FreeRTOS**
- *Peripherals view*, *"SYS → Timebase source"* change to **TIM7**

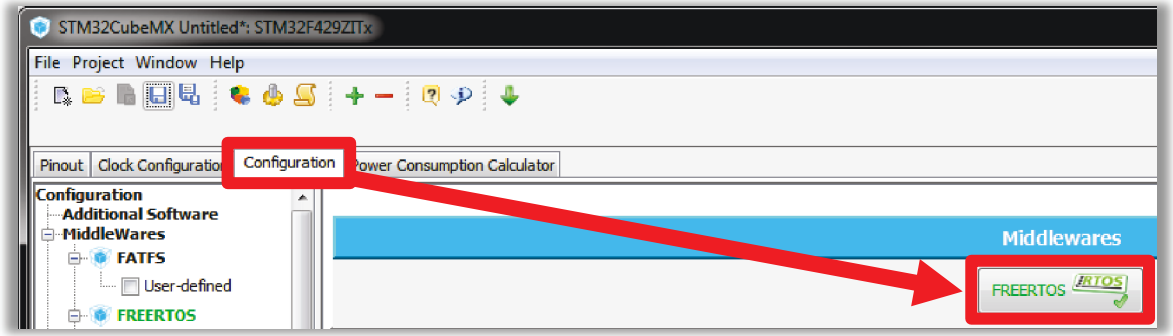


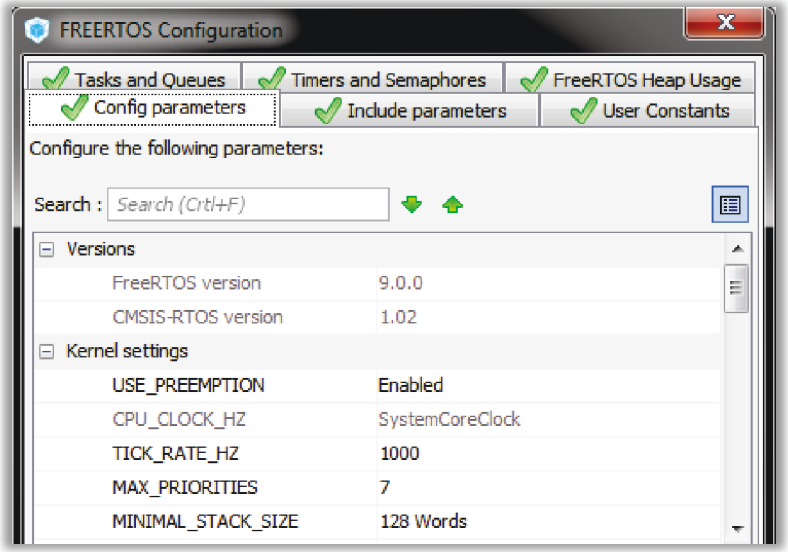
① **FREERTOS: √Check Enable**

② **Timebase: Systick → TIM7**

FreeRTOS in CubeMX

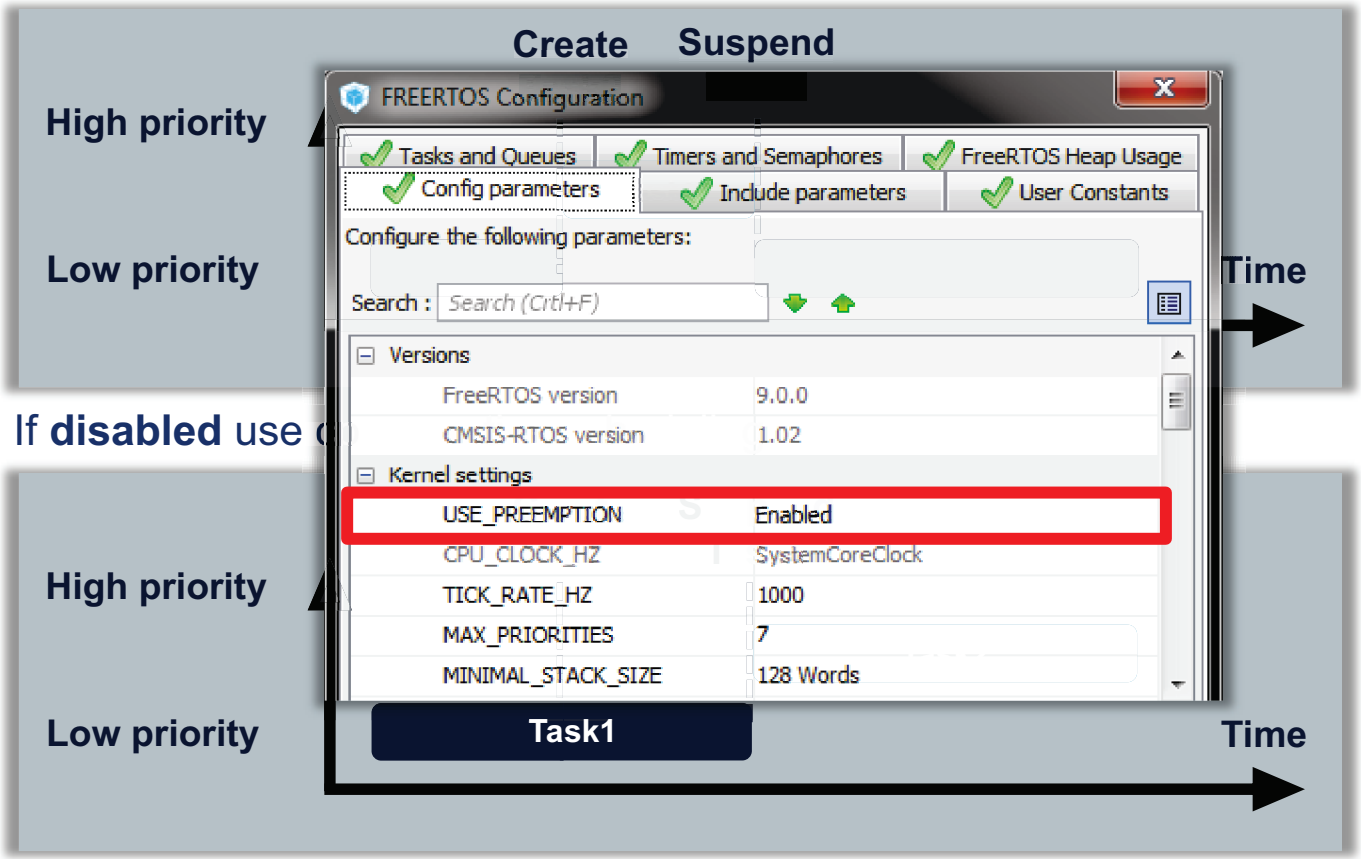
- *In Configuration TAB* is now possible to configure FreeRTOS
- FreeRTOS configuration supported by CubeMX
- Config parameters
  - About kernel features
  - About Memory mgmt.
- Include parameters
  - Additional APIs
    Not necessary for FreeRTOS run
- Tasks and Queues
  - We can easily create task or queue
- Timers and Semaphores
  - Create semaphore and timer for us
- FreeRTOS Heap Usage
  - Memory usage information



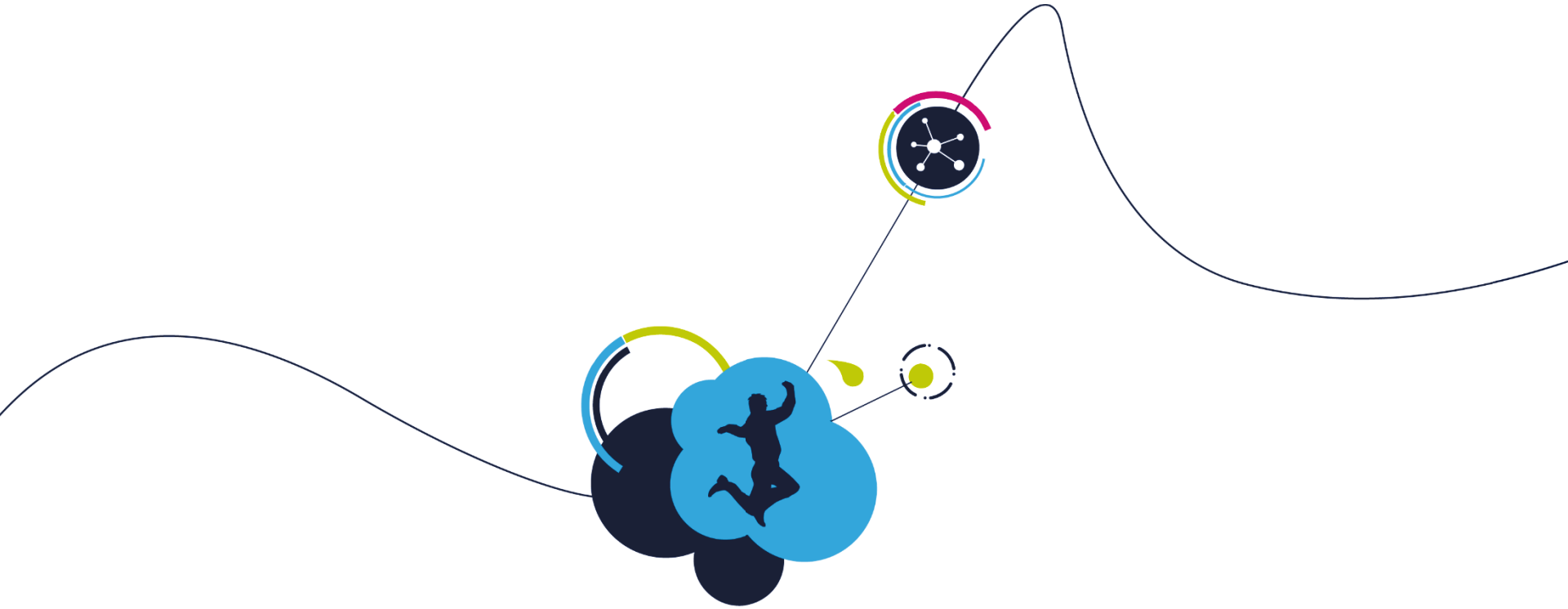


Kernel settings

- Use preemption
  - If **enabled** use pre-emptive scheduling



  - If **disabled** use c



FreeRTOS
Memory allocations types

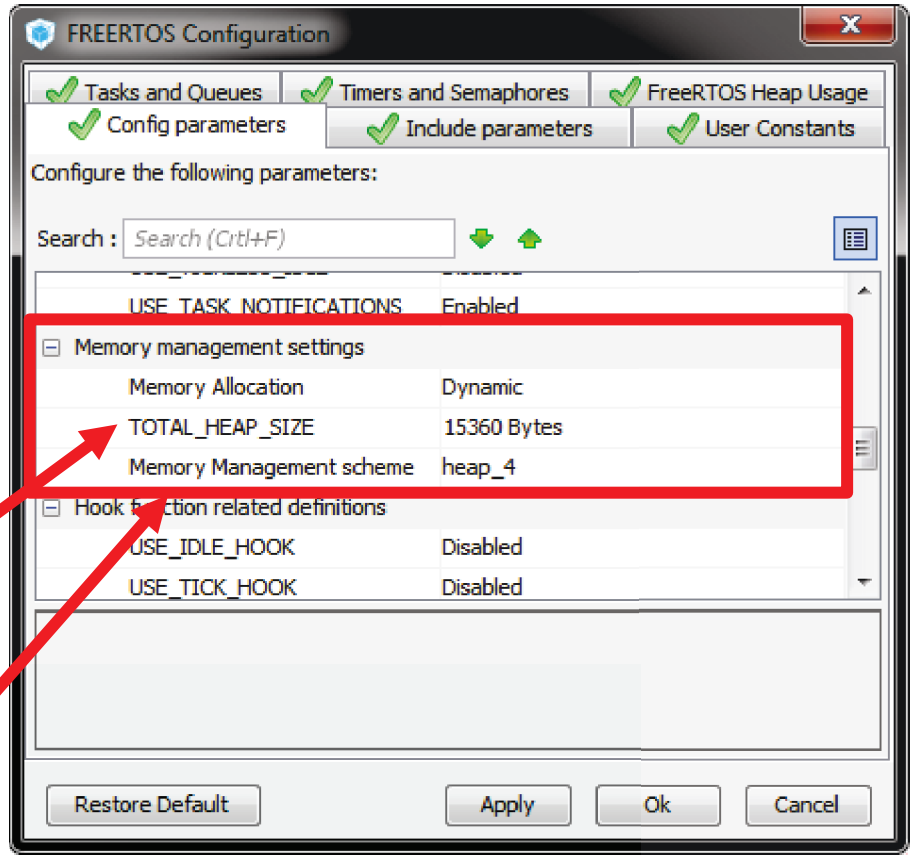
FreeRTOS
Dynamic memory management

- FreeRTOS have own heap which is use for components
  - Tasks
  - Queues
  - Semaphores
  - Mutexes
  - Dynamic memory allocation

- Is possible to select type of memory allocation

Total heap size for RTOS

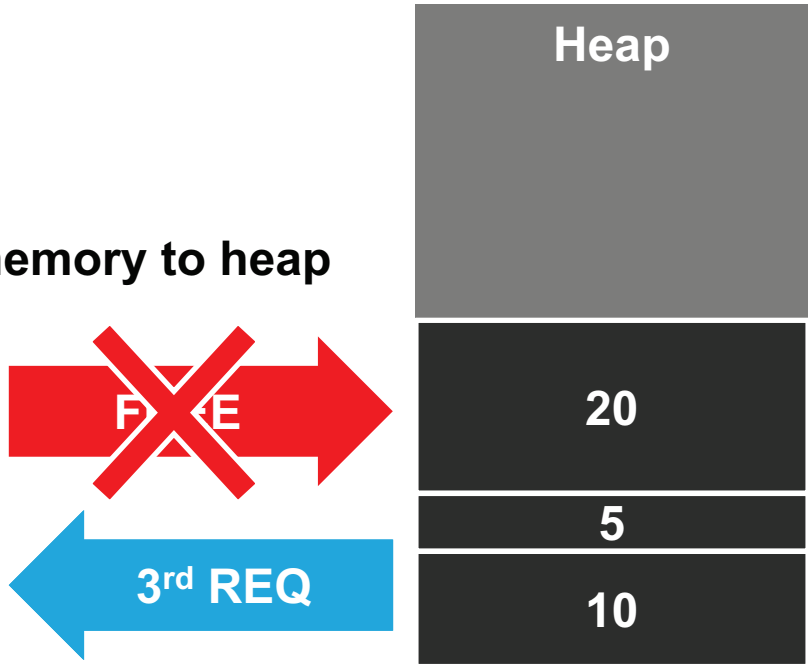
How is memory allocated and deallocated



FreeRTOS
Dynamic memory management

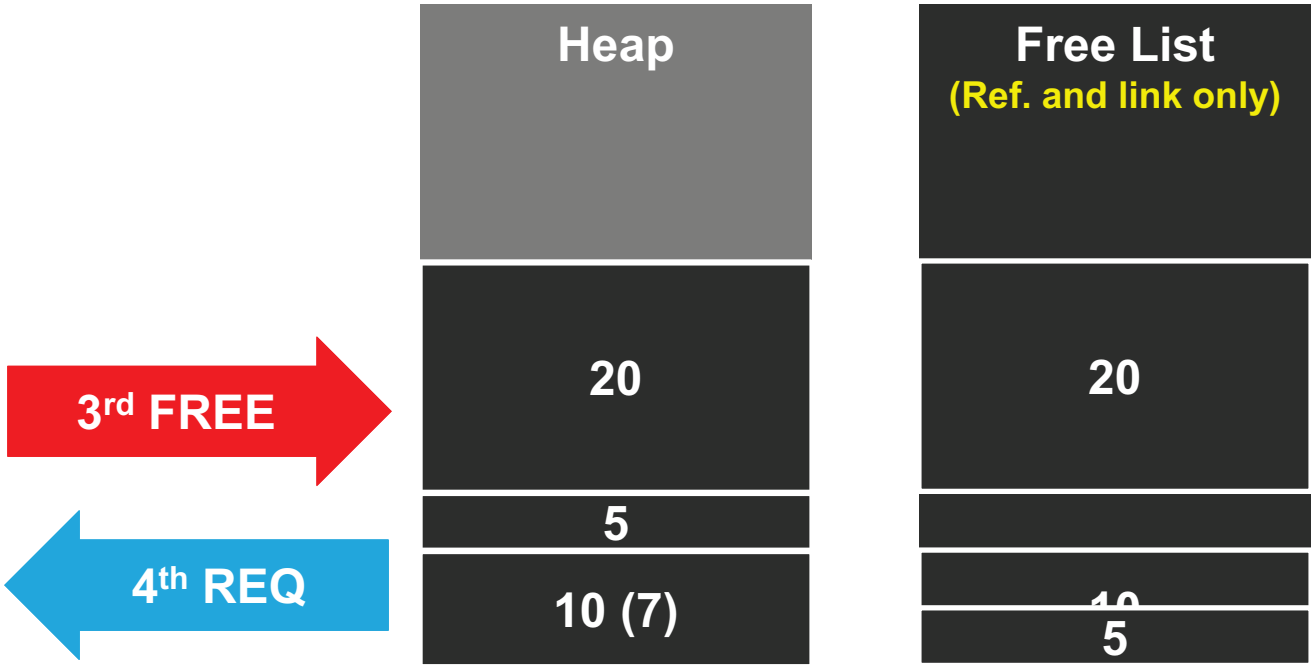
- Heap_1.c
  - Simplest allocation method (deterministic sequential allocation), but *does not allow freeing of allocated memory*
    - Could be interesting when no memory freeing is necessary

Is not possible to return memory to heap



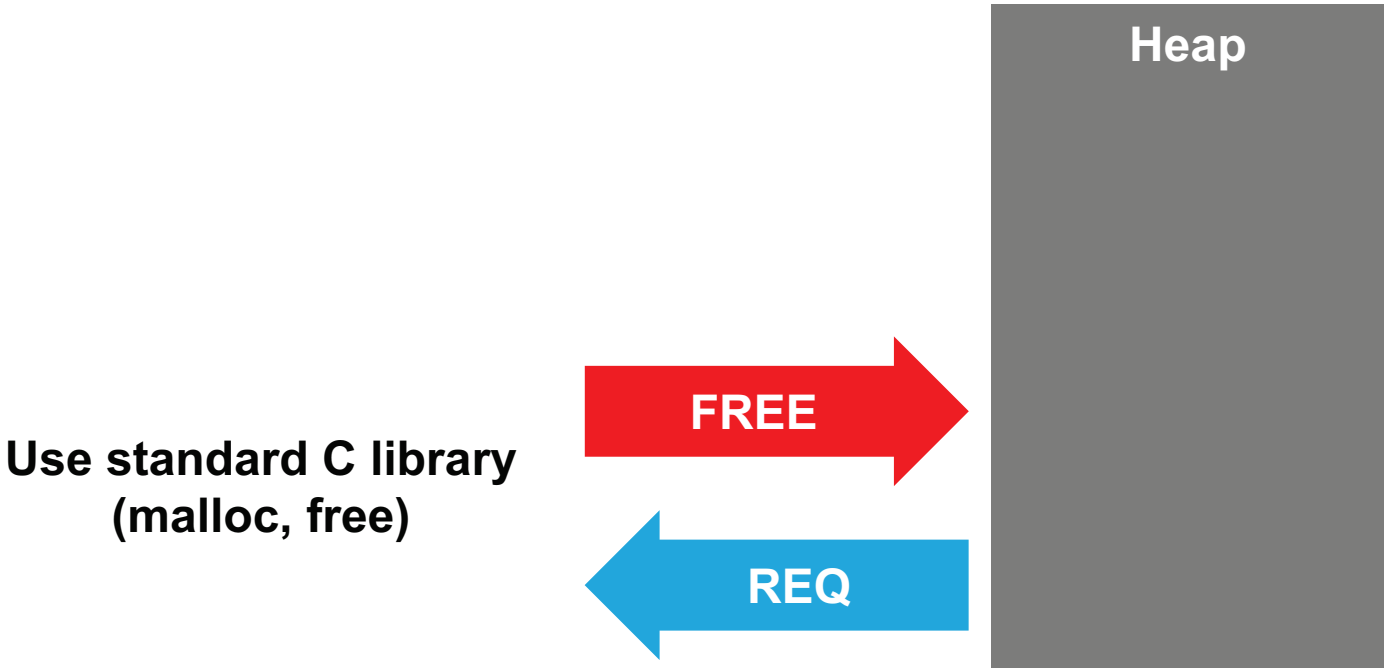
FreeRTOS
Dynamic memory management

- Heap_2.c
  - Implements a best fit algorithm for allocation
  - Allows memory free operation but does not combine adjacent free blocks
    - Risk of fragmentation



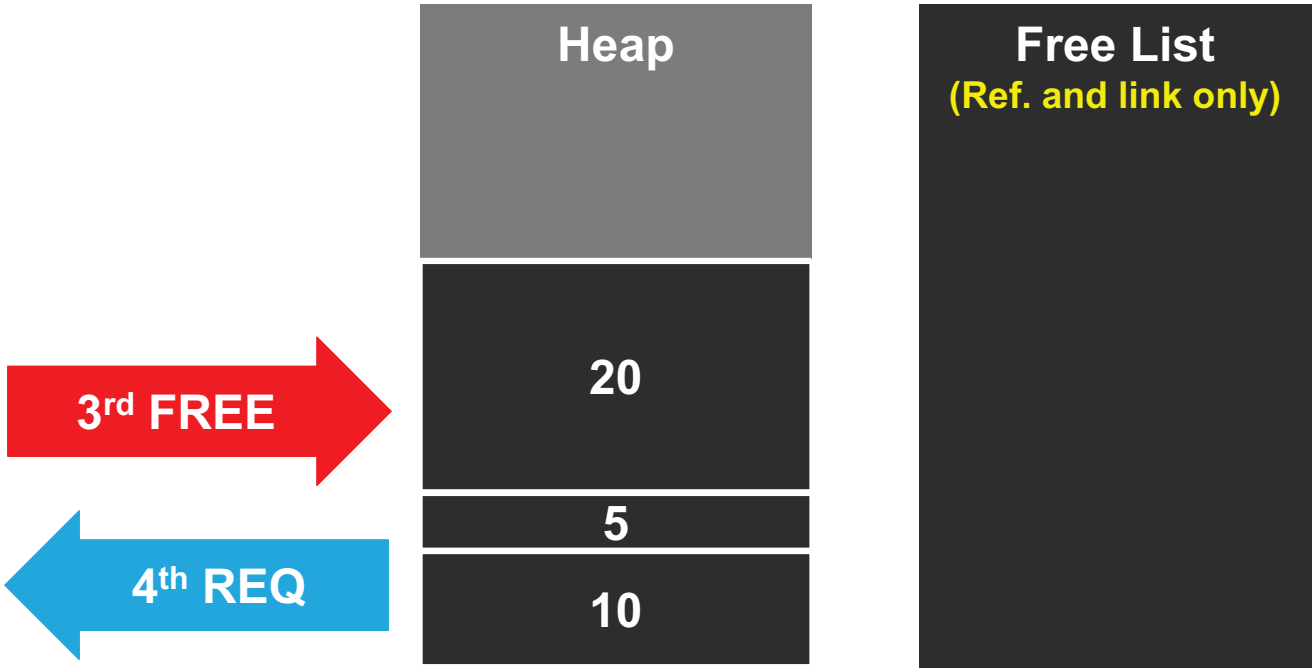
FreeRTOS
Dynamic memory management

- Heap_3.c
  - Implements a simple wrapper for the standard C library malloc() and free(), the wrapper makes these functions thread safe, but makes code increase and not deterministic



FreeRTOS
Dynamic memory management

- Heap_4.c
  - First fit algorithm and able to combine adjacent free memory blocks into a single block
    - This model is used in STM32Cube examples



Memory allocation

- Use heap_4.c
- Memory Handler definition

```
/* Private variables ---------------------------------------------
--- */
osThreadId Task1Handle;
osPoolId PoolHandle;
```

- Memory allocation

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  osPoolDef(Memory, 0x100, uint8_t);
  PoolHandle = osPoolCreate(osPool(Memory));
  uint8_t* buffer = osPoolAlloc(PoolHandle);
  /* Infinite loop */
  for (;;) {
    osDelay(5000);
  }
  /* USER CODE END 5 */
}
```

Create memory pool

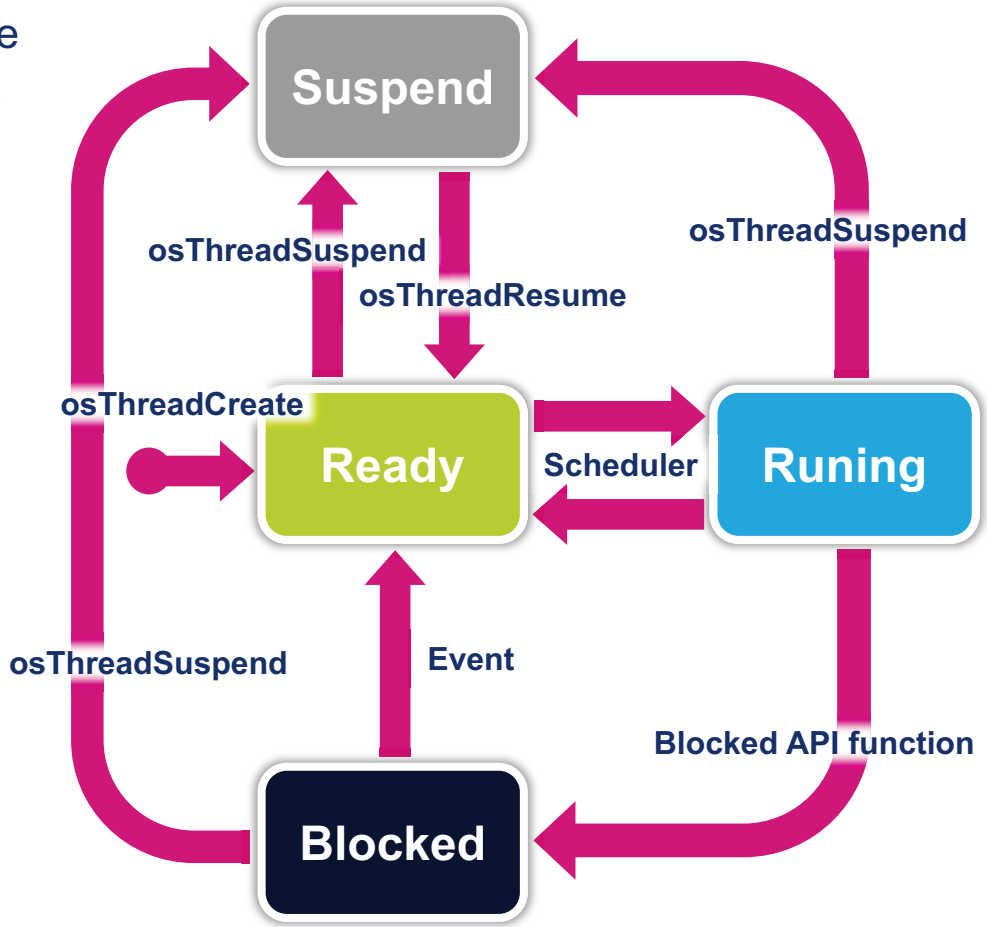
Allocate memory from pool

FreeRTOS Tasks

life.augmented

FreeRTOS Task states

- **Ready**
  - Tasks are ready to execute but are not currently executing because a different task with equal or higher priority is running
- **Running**
  - When task is actually running
- **Blocked**
  - Task is waiting for a either a temporal or external event
- **Suspended**
  - Task not available for scheduling

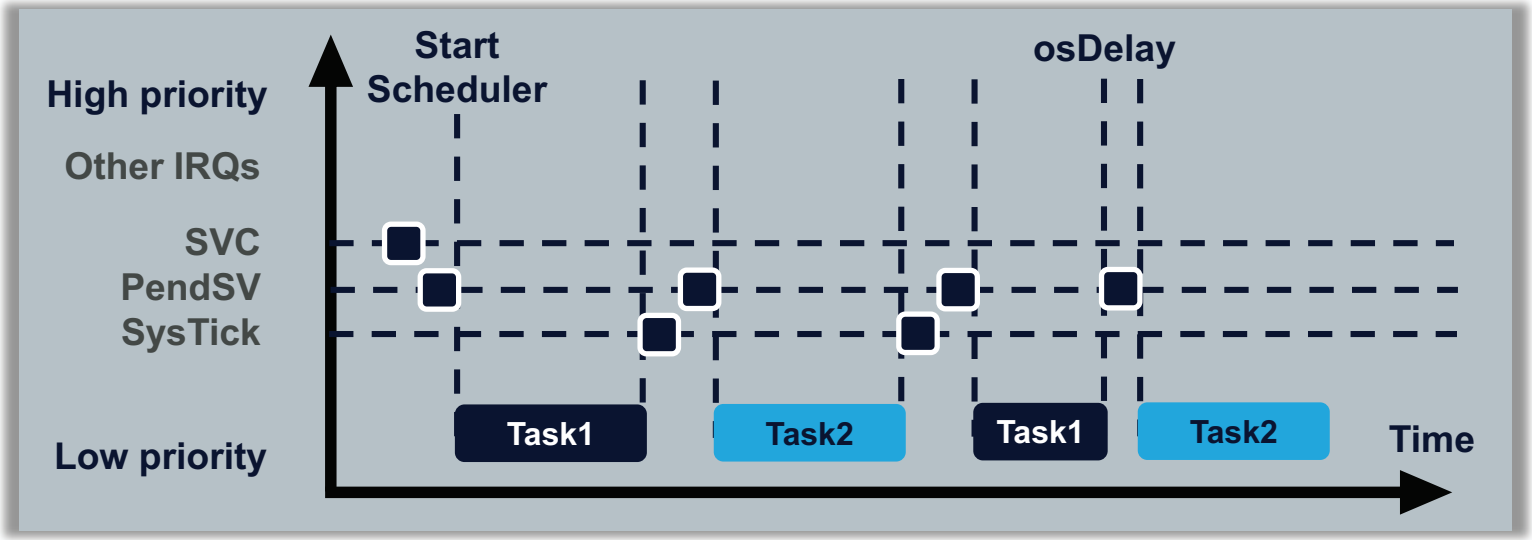


FreeRTOS
Task switch

- Task switching on STM32?
- Cortex core have implemented few features which directly support OS systems
- Two interrupts dedicated for OS
  - PendSV interrupt
  - SVC interrupt
- Two stack pointers
  - Process stack pointer
  - Main stack pointer
- SysTick timer
  - Used to periodically trigger scheduling

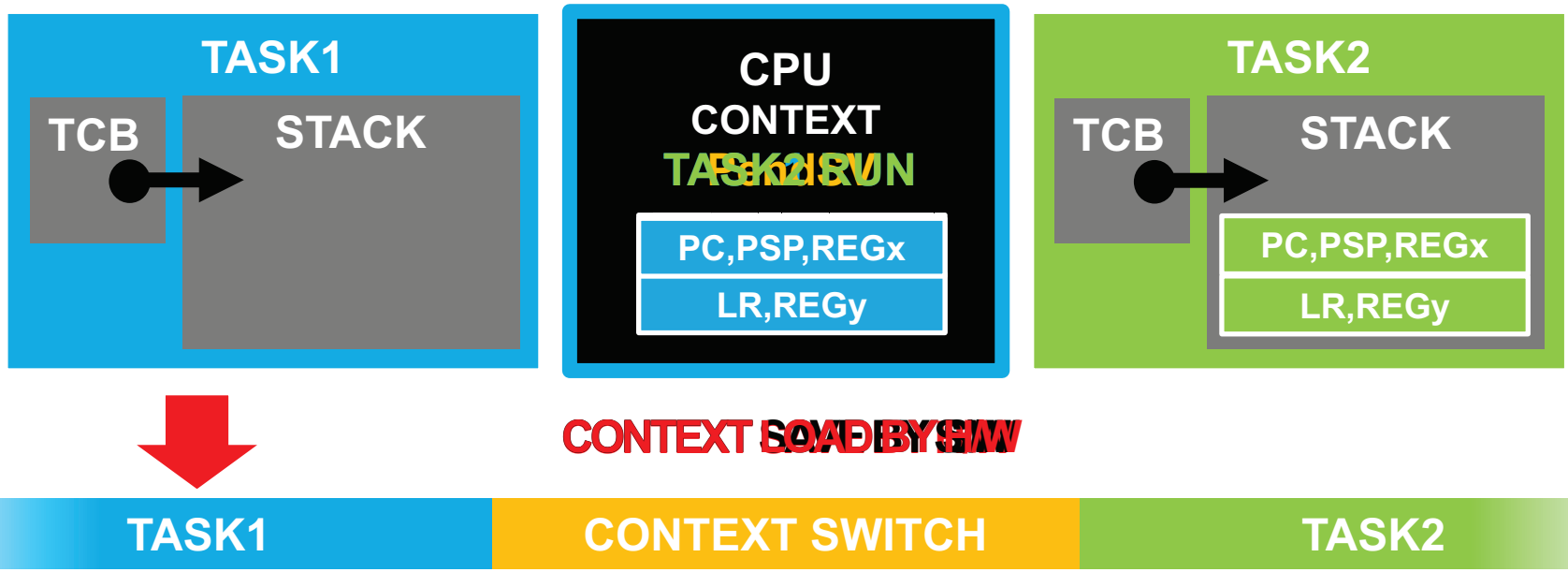
FreeRTOS OS interrupts

- PendSV interrupt
  - Manage context switching
  - Lowest NVIC interrupt priority
  - Not triggered by any peripherals
  - Pending state set from other interrupts or from task which want end earlier (non MPU version)
- SVC interrupt
  - Interrupt risen by SVC instruction
  - Called if task want end earlier (MPU version)
  - In this interrupt set pending state PendSV (MPU version)
- SysTick timer
  - Set PendSV if context switch is necessary



FreeRTOS
Stack pointer and Context Switch

- Main stack pointer
  - Used in interrupts
  - Allocated by linker during compiling
- Process stack pointer
  - Each task have own stack pointer
  - During context switch the stack pointer is initialized for correct task



Tasks API

- Create task

```
osThreadId osThreadCreate (const osThreadDef_t *thread_def, void
*argument)
```

- Delete task

```
osStatus osThreadTerminate (osThreadId thread_id)
```

- Get task ID

```
osThreadId osThreadGetId (void)
```

- Task handle definition

```
/* Private variables ---------------------------------------------
- */
osThreadId Task1Handle;
```

- Create Task

```
  /* Create the thread(s) */
  /* definition and creation of Task1 */
  osThreadDef(Task1, StartTask1, osPriorityNormal, 0, 128);
  Task1Handle = osThreadCreate(osThread(Task1), NULL);
```

Tasks API

- Check if task is suspended

```
osStatus osThreadIsSuspended(osThreadId thread_id)
```

- Resume task

```
osStatus osThreadResume (osThreadId thread_id)
```

- Check state of task

```
osThreadState osThreadGetState(osThreadId thread_id)
```

- Suspend task

```
osStatus osThreadSuspend (osThreadId thread_id)
```

- Resume all tasks

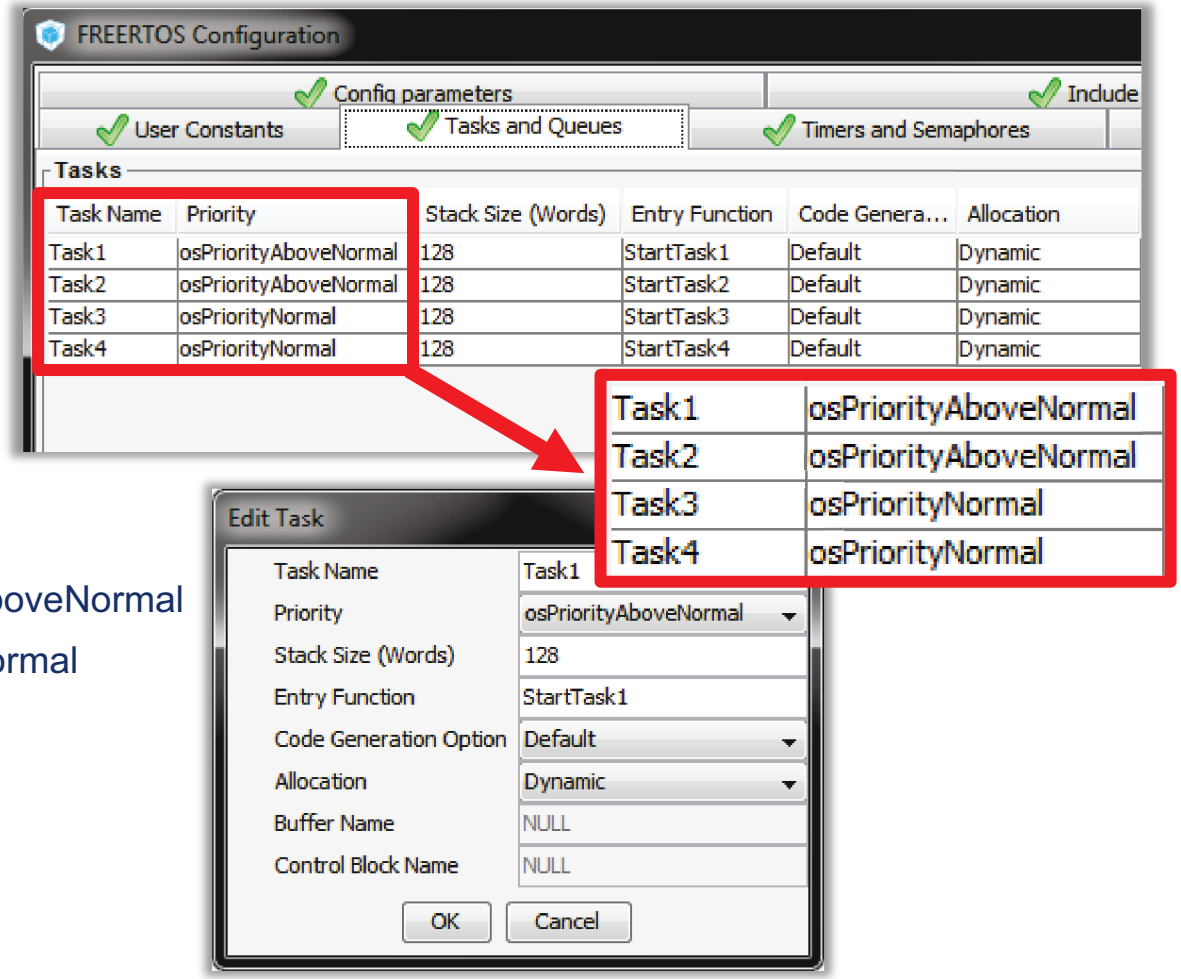
```
osStatus osThreadResumeAll (void)
```

- Suspend all tasks

```
osStatus osThreadSuspendAll (void)
```

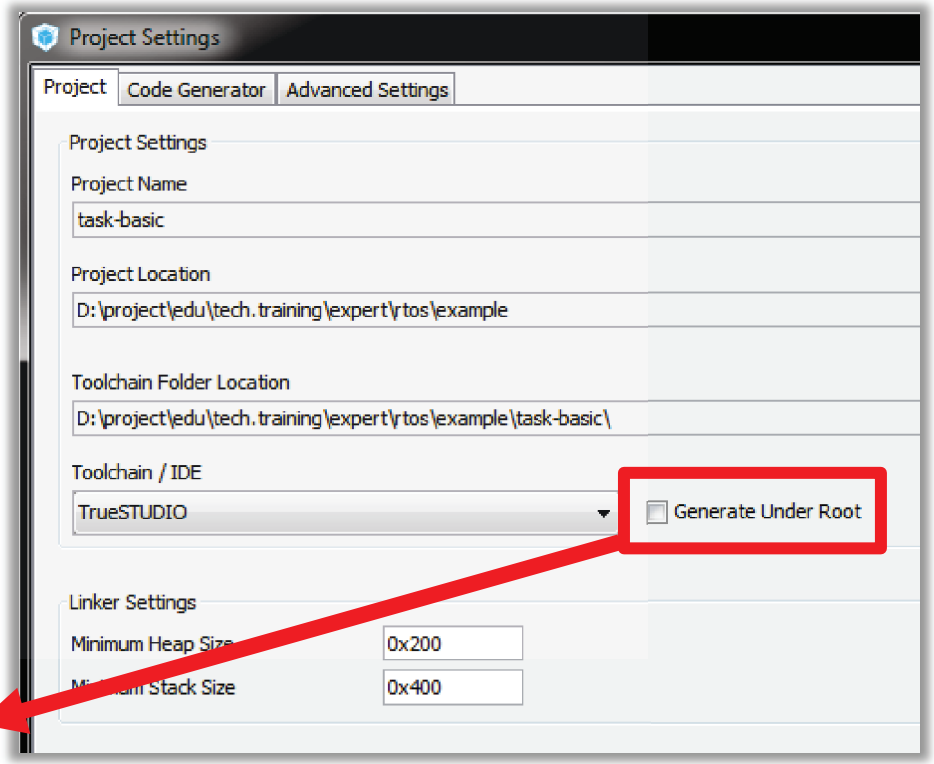
Tasks lab

- By default defined one defaultTask
- Task is defined by
  - Name
  - Priority
  - Stack size
  - Name of entry function
- Define 4 tasks
- With different priority
  - Task1, Task2 - osPriorityAboveNormal
  - Task3, Task4 - osPriorityNormal



Tasks lab

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code



Disable “Generate Under Root”

Tasks lab

- User RTOS code generated to “freertos.c” file
- Any component in FreeRTOS need to have handle, very similar to CubeMX

```
/* Private variables -----------------------------------------
--- */
osThreadId Task1Handle;
osThreadId Task2Handle;
osThreadId Task3Handle;
osThreadId Task4Handle;
```

- Task function prototypes, names was taken from CubeMX

```
/* Private function prototypes -------------------------------
--- */
void StartTask1(void const * argument);
void StartTask2(void const * argument);
void StartTask3(void const * argument);
void StartTask4(void const * argument);
```

Tasks lab

- Before the scheduler is start we must create tasks
  - *MX_FREERTOS_Init() @ freertos.c*

```
/* Create the thread(s) */
/* definition and creation of Task1 */
osThreadDef(Task1, StartTask1, osPriorityAboveNormal, 0, 128);
Task1Handle = osThreadCreate(osThread(Task1), NULL);

/* definition and creation of Task2 */
osThreadDef(Task2, StartTask2, osPriorityAboveNormal, 0, 128);
Task2Handle = osThreadCreate(osThread(Task2), NULL);

/* definition and creation of Task3 */
osThreadDef(Task3, StartTask3, osPriorityNormal, 0, 128);
Task3Handle = osThreadCreate(osThread(Task3), NULL);

/* definition and creation of Task4 */
osThreadDef(Task4, StartTask4, osPriorityNormal, 0, 128);
Task4Handle = osThreadCreate(osThread(Task4), NULL);
```

Create task, allocate memory

Define task parameters

Tasks lab

- Start the scheduler, the scheduler function newer ends

```
 /* Start scheduler */
  osKernelStart();
  /* We should never get here as control is now taken by the
scheduler */
```

- Task must have inside infinity loop in case we don’t want to end the task
  - Task1 and Task2 call “osDelay”

```
void StartTask{1 or 2}(void const * argument)
{
  /* USER CODE BEGIN StartTask1 */
  /* Infinite loop */
  for(;;) {
    printf("Task {1 or 2}\n");
    osDelay(1000);
  }
  /* USER CODE END StartTask1 */
}
```

Tasks lab

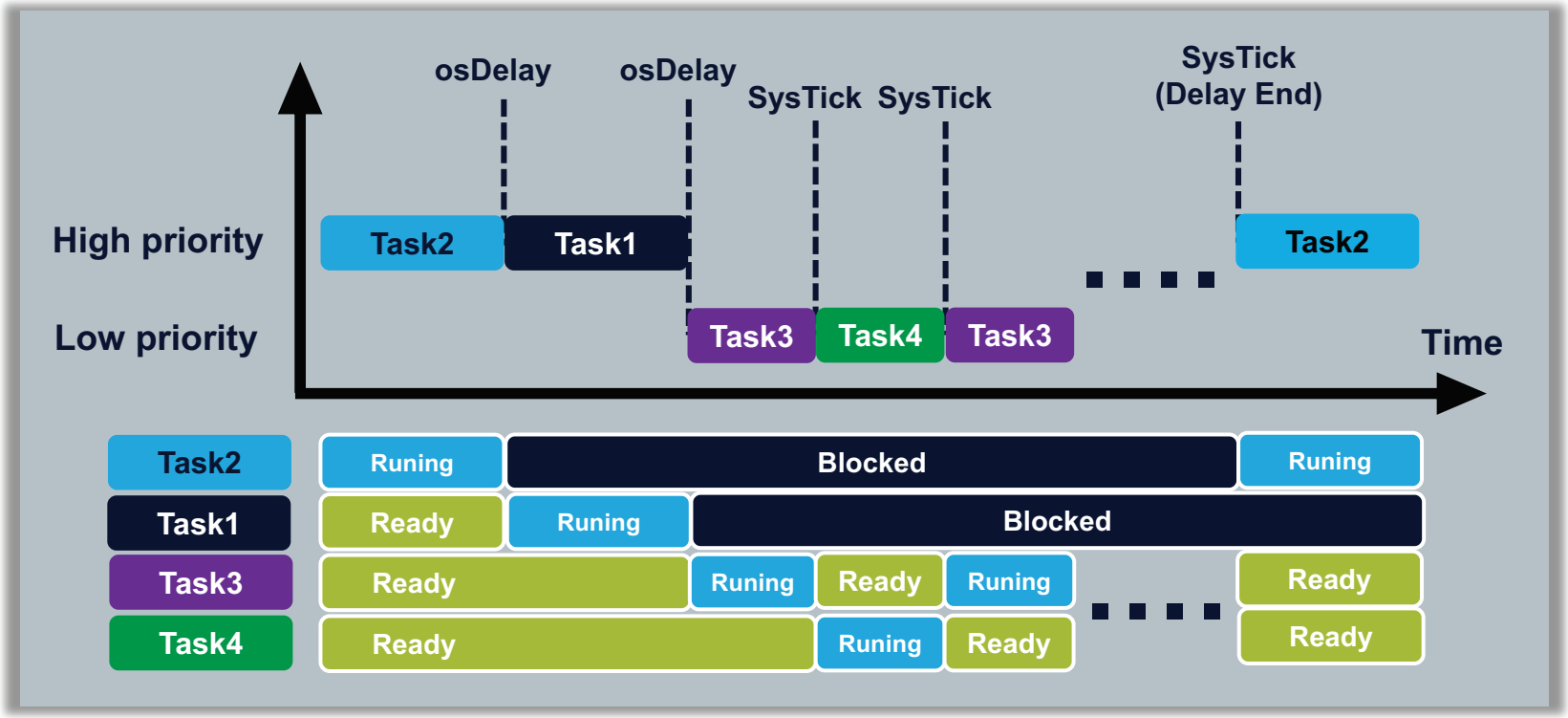
- Second loop is same as previous except “HAL_Delay”

```
/* StartTask2 function */
void StartTask{3 or 4}(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;) {
    printf("Task {3 or 4}\n");
    HAL_Delay(1000);
  }
  /* USER CODE END StartTask2 */
}
```

- Compile and run project in debug and watch terminal window

Tasks lab

- Tasks which have OS delays are yield the processing to other task
- Without OS delays the tasks will be in running state or in Ready state
  - See *“configUSE_TIME_SLICING”*



osDelay API

- Delay function

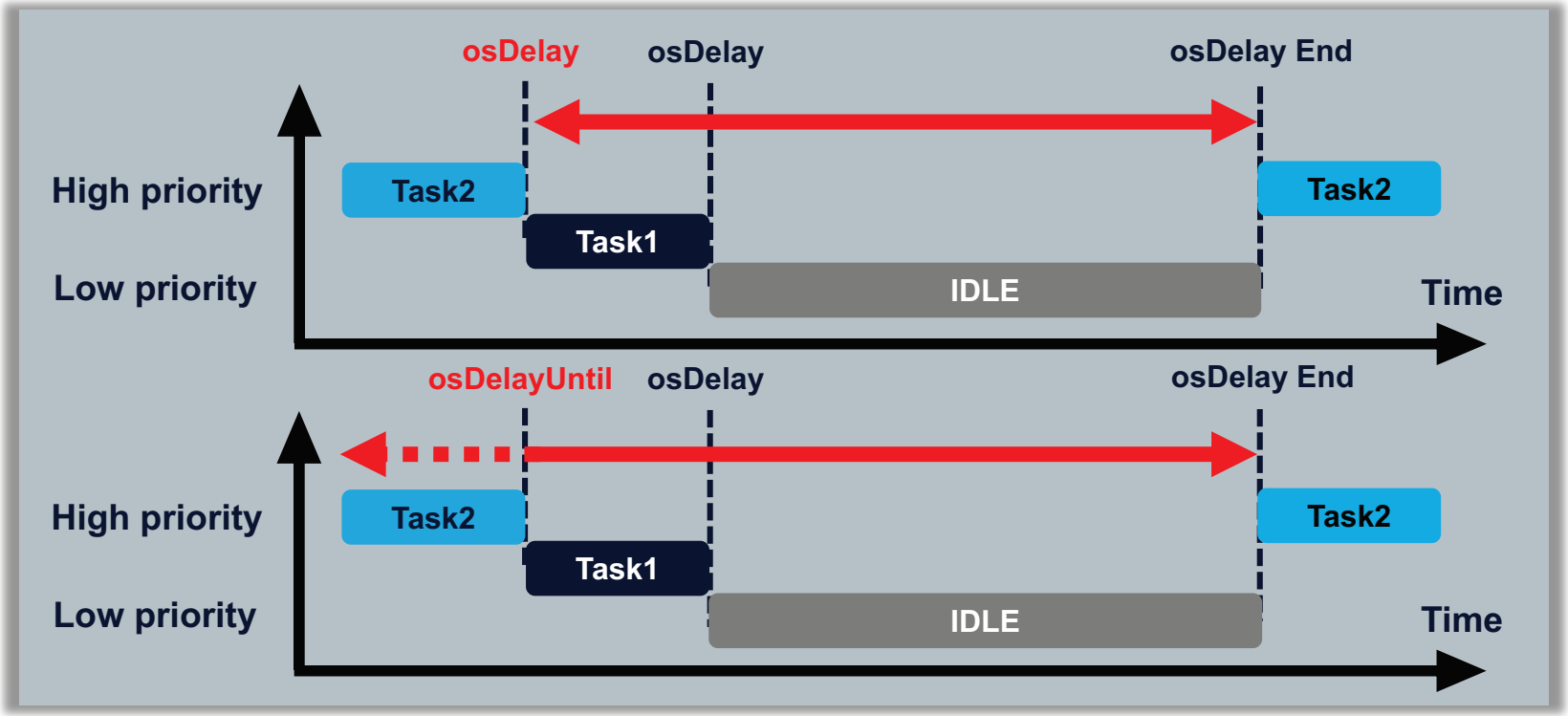
```
osStatus osDelay (uint32_t millisec)
```

- Delay function which measure time from which is delay measured

```
osStatus osDelayUntil (uint32_t PreviousWakeTime, uint32_t
millisec)
```

osDelay function

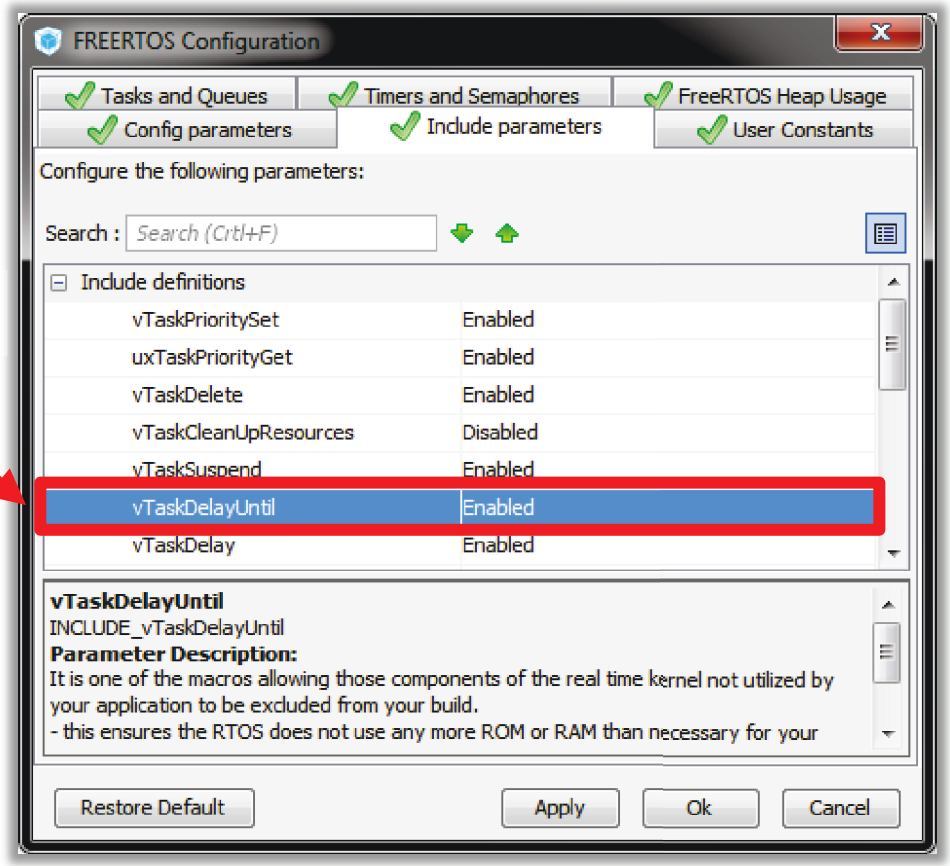
- osDelay start measure time from osDelay call
- osDelayUntil measure time from point which we selected
- This allow us to call task in regular intervals



osDelay and osDelayUntil

- Enable *"vTaskDelayUntil"* in ***"Include parameters"***

Enable "vTaskDelayUntil"



osDelay and osDelayUntil

- Only 2 tasks will used,.
- Regenerate project, modify tasks to:

```
void StartTask1(void const * argument)
{

  /* USER CODE BEGIN StartTask1 */
  uint32_t lastwakeup_t =
osKernelSysTick();
  /* Infinite loop */
  for (;;) {
    printf("Task 1 %d\n", lastwakeup_t);
    HAL_Delay(1000);
    osDelayUntil(&lastwakeup_t, 2000);
  }
  /* USER CODE END StartTask1 */

}
```

For "osDelayUntil" function, We need mark startup time

Time from which is delay measured

… Function will be executed every 2s.

osDelay and osDelayUntil

- Only 2 tasks will used,.
- Regenerate project, modify tasks to:

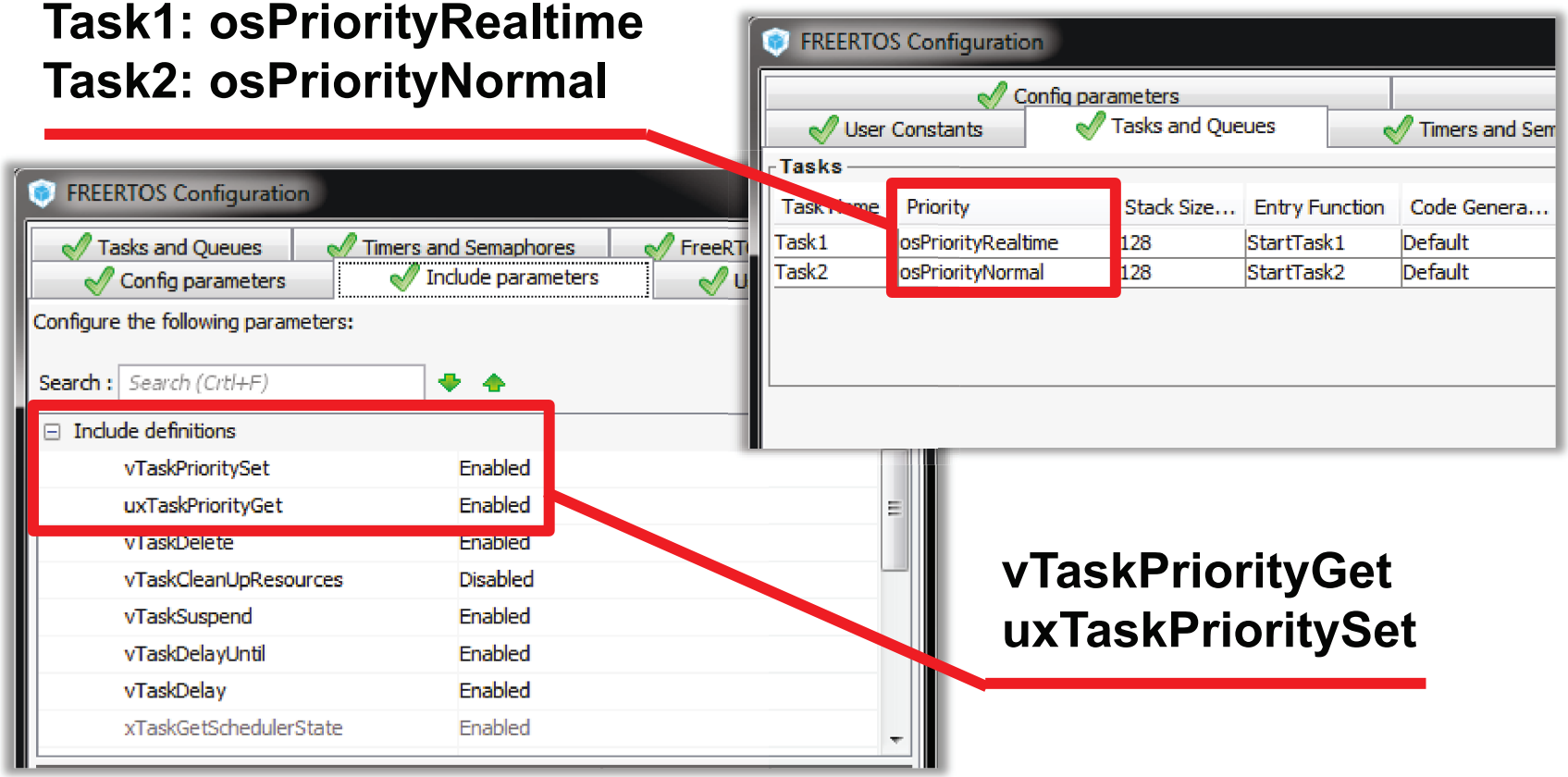
```
void StartTask2(void const * argument)
{

  /* USER CODE BEGIN StartTask2 */
  uint32_t lastwakeup_t;
  /* Infinite loop */
  for (;;) {
    lastwakeup_t = osKernelSysTick();
    printf("Task 2 %d\n", lastwakeup_t );
    HAL_Delay(1000);
    osDelay(2000);
  }
  /* USER CODE END StartTask2 */

}
```

…How long does this function execute per loop?

Priority change lab

- Task1 have higher priority than Task2
- Enable *'vTaskPriorityGet'* and *'uxTaskPrioritySet'* in ***'Include Parameters'***

Task1: osPriorityRealtime
Task2: osPriorityNormal



vTaskPriorityGet
uxTaskPrioritySet

Priority change lab

- Modify Task1 to:

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN StartTask1 */
  osPriority prior;
  /* Infinite loop */
  for (;;) {
    prior = osThreadGetPriority(Task2Handle);
    printf("Task 1\n");
    osThreadSetPriority(Task2Handle, prior +
1);
    HAL_Delay(1000);
  }
  /* USER CODE END StartTask1 */
}
```

Read Task2 priority

Increase Task2 priority

Priority change lab

- Modify Task2 to:

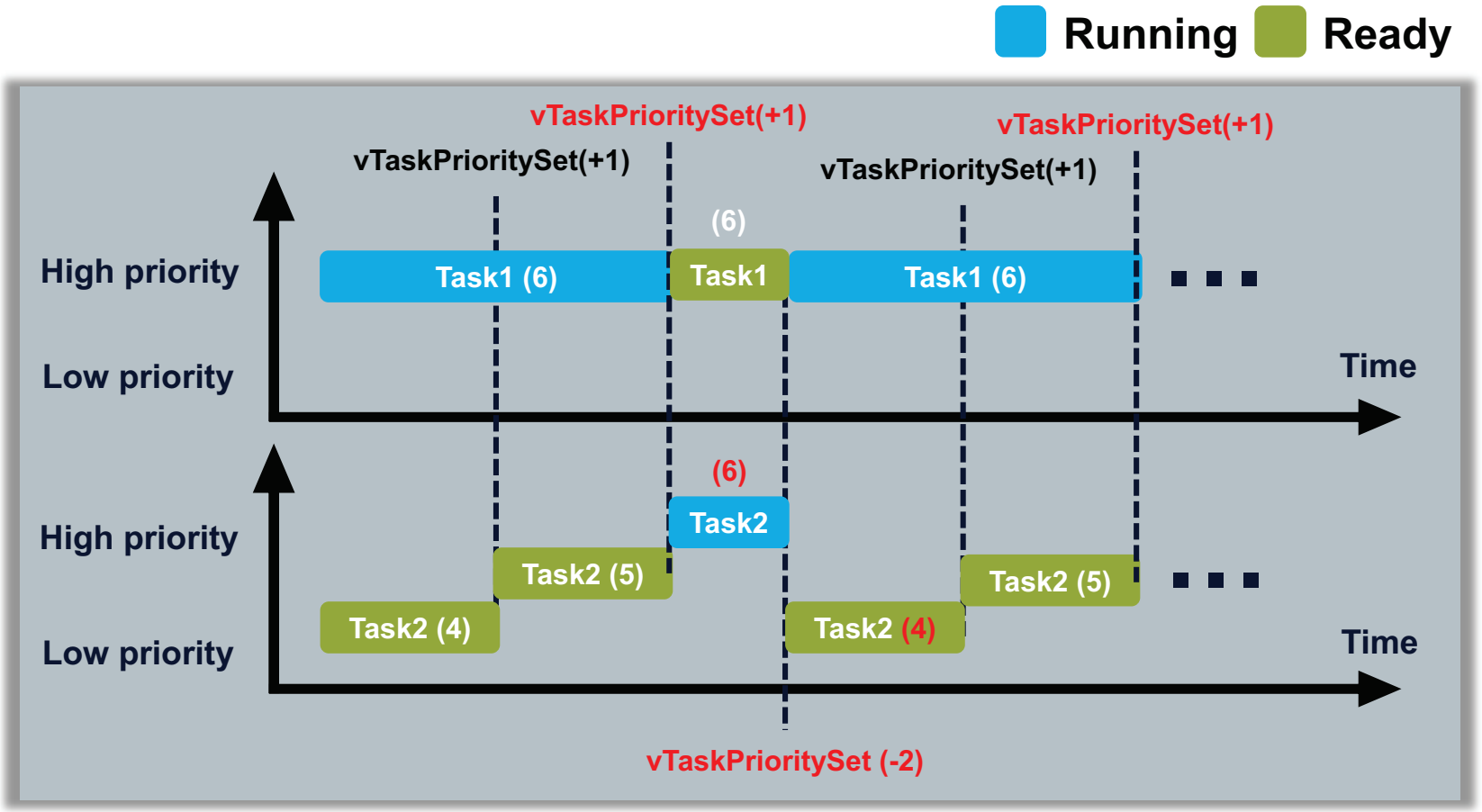
```
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  osPriority prior;
  /* Infinite loop */
  for (;;) {
    prior = osThreadGetPriority(NULL);
    printf("Task 2\n");
    osThreadSetPriority(NULL, prior - 2);
  }
  /* USER CODE END StartTask2 */
}
```

Read priority of current task

Decrease task priority

Priority change lab

- How priorities are changed?



Creating and deleting tasks lab

- Example how to create and delete tasks
- Comment Task2 creation part in main.c

```
  /* definition and creation of Task2 */
  //osThreadDef(Task2, StartTask2, osPriorityNormal, 0, 128);
  //Task2Handle = osThreadCreate(osThread(Task2), NULL);
```

- Modify Task1 to create task2

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN StartTask1 */
  /* Infinite loop */
  for (;;) {
    printf("Create task2");
    osThreadDef(Task2, StartTask2, osPriorityNormal, 0, 128);
    Task2Handle = osThreadCreate(osThread(Task2), NULL);
    osDelay(1000);
  }
  /* USER CODE END StartTask1 */
}
```

Creating and deleting tasks lab

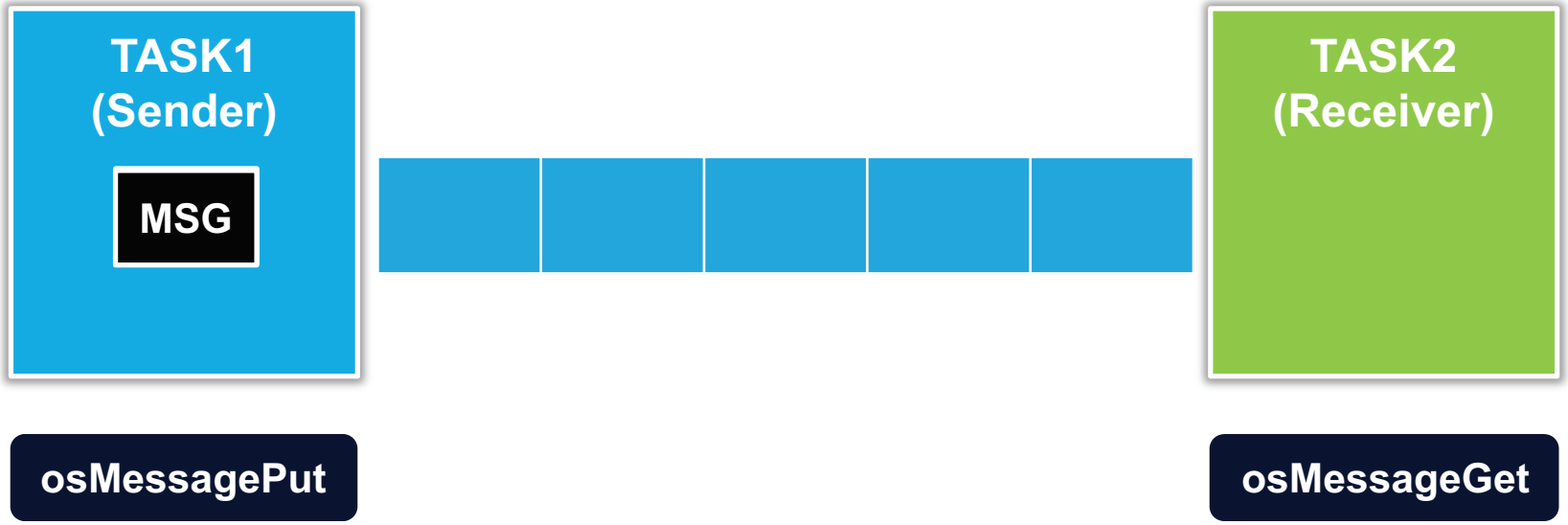
- Example how to create and delete tasks
- Modify Task2 to delete him-self:

```
/* StartTask2 function */
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for (;;) {
    printf("Delete Task2\n");
    osThreadTerminate(Task2Handle);
  }
  /* USER CODE END StartTask2 */
}
```

FreeRTOS Queues

life.augmented

Queue


TASK1
(Sender)
MSG
TASK2
(Receiver)
osMessagePut
osMessageGet

Queue

- Create Queue:

```
osMessageQId osMessageCreate(const osMessageQDef_t *queue_def,
osThreadId id)
```

- Put data into Queue

```
osStatus osMessagePut(osMessageQId id, uint32_t info, uint32_t
millisec)
```

Item to send

- Receive data from Queue

```
osEvent osMessageGet(osMessageQId id, uint32_t millisec)
```

Structure with status and with received item

Queue

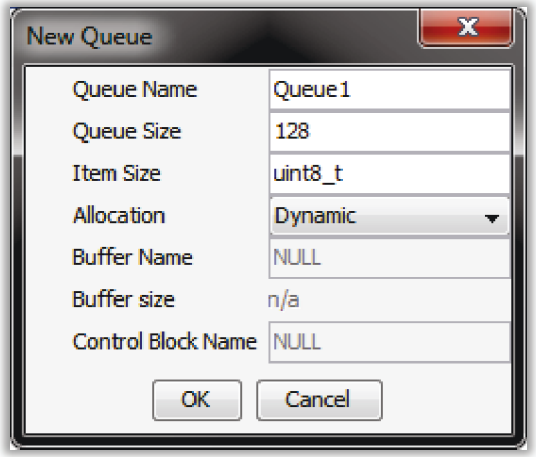
- osEvent structure

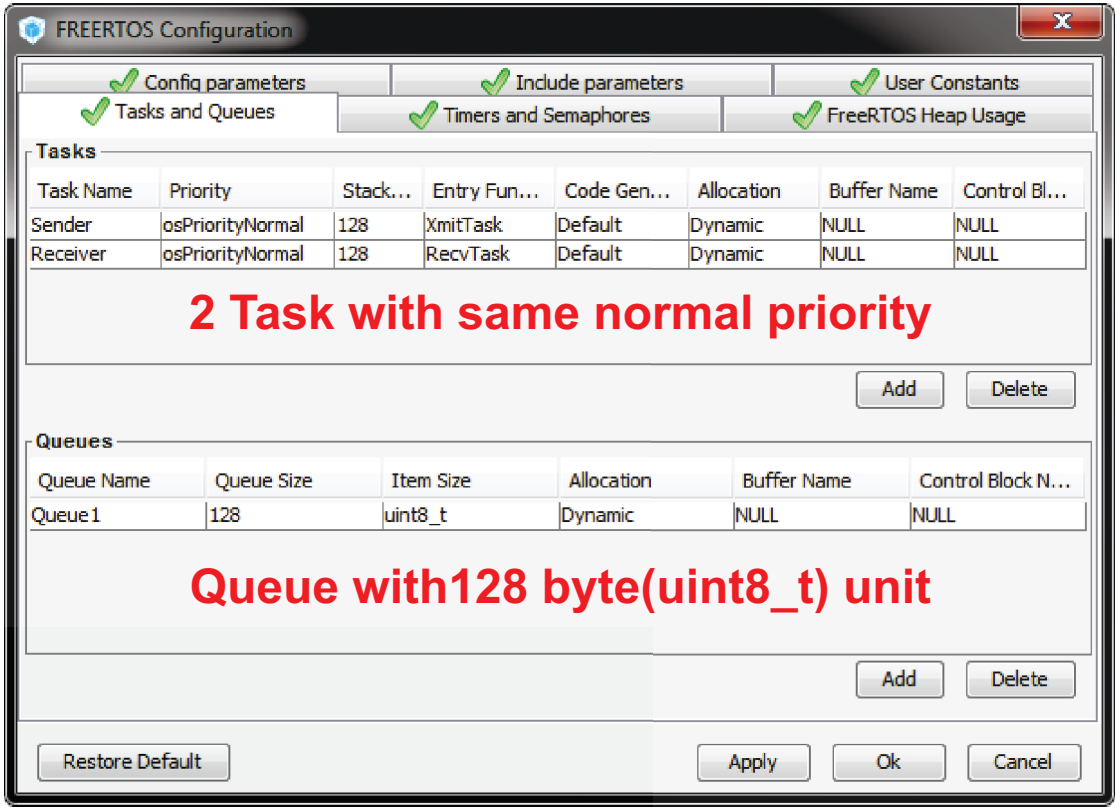
```
typedef struct  {
  osStatus     status;  ///< status code: event or error
information
  union  {
    uint32_t     v;  ///< message as 32-bit value
    void        *p;  ///< message or mail as void pointer
    int32_t  signals;  ///< signal flags
  } value;             ///< event value
  union  {
    osMailQId  mail_id;  ///< mail id obtained by \ref osMailCreate
    osMessageQId message_id;  ///< message id obtained by \ref
osMessageCreate
  } def;               ///< event definition
} osEvent;
```

- If we want to get data from osEvent we must use:
  - osEventName.v if the value is 32bit message(or 8/16bit)
  - osEventName.p and retype on selected data-type

Queue lab

- Set both tasks (Sender, Receiver) to normal priority
  - XmitTask / RecvTask
- Queue part
  - Button ‘Add’
  - Set queue size to ‘128’
  - Queue type to ‘uint8_t’
- Button ‘OK’





Queue lab

- Queue handler is now defined

```
/* Private variables ---------------------------------------------
--- */
osThreadId SenderHandle;
osThreadId ReceiverHandle;
osMessageQId Queue1Handle;
```

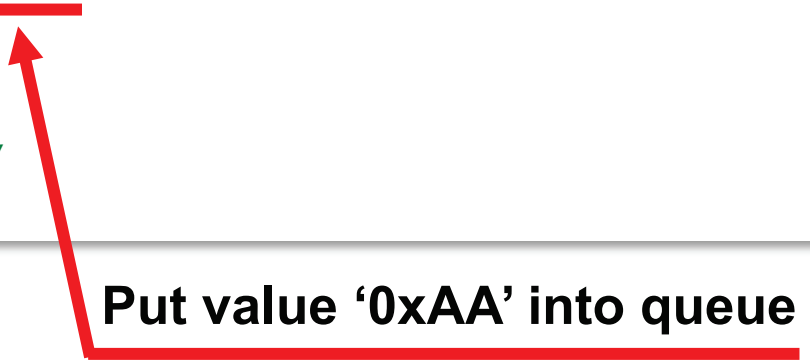
- Queue item type initialization, length definition and create of queue and memory allocation

```
/* Create the queue(s) */
/* definition and creation of Queue1 */
osMessageQDef(Queue1, 128, uint8_t);
Queue1Handle = osMessageCreate(osMessageQ(Queue1), NULL);
```

Queue lab

- Sender task

```
void XmitTask (void const * argument)
{
  /* USER CODE BEGIN XmitTask */
  /* Infinite loop */
  for (;;) {
    printf("Sender");
    osMessagePut(Queue1Handle, 0xAA, 200);
    printf("Sender delay\n");
    osDelay(1000);
  }
  /* USER CODE END XmitTask */
}
```



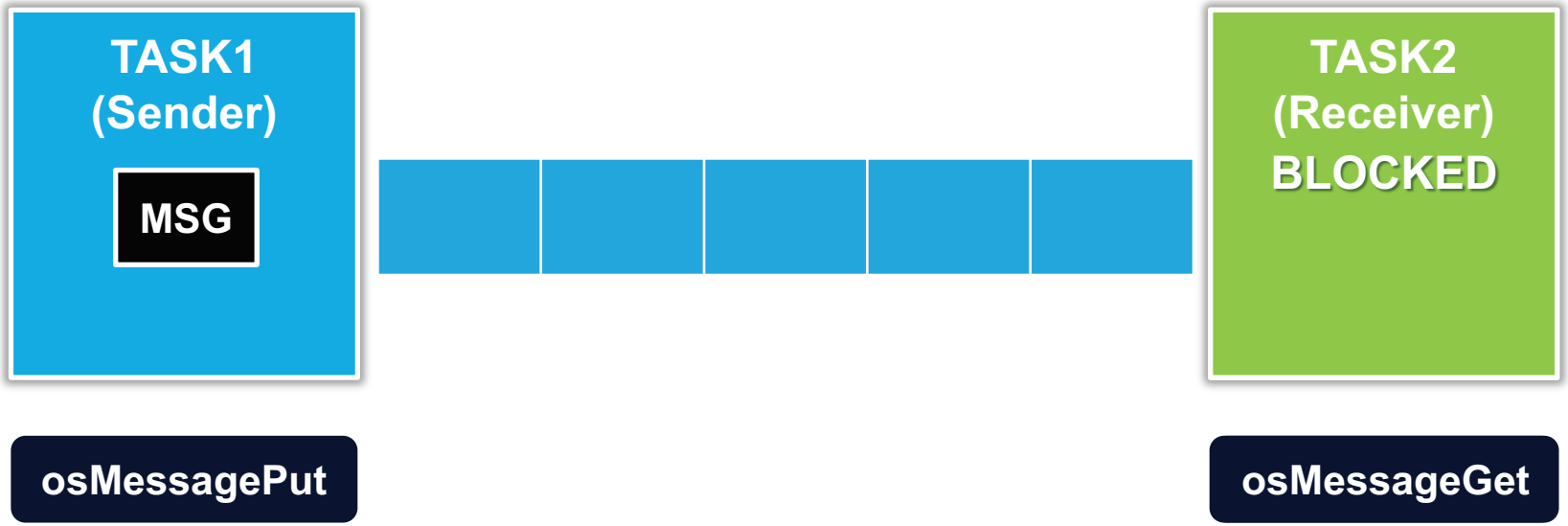
Queue lab

- Receiver task

```
/* RecvTask function */
void RecvTask(void const * argument)
{
  /* USER CODE BEGIN RecvTask */
  osEvent retval;
  /* Infinite loop */
  for (;;)  {
    printf("Receiver\n");
    retval = osMessageGet(Queue1Handle, 1000);
    printf("%d\n", retval.value.v);
  }
  /* USER CODE END RecvTask */
}
```

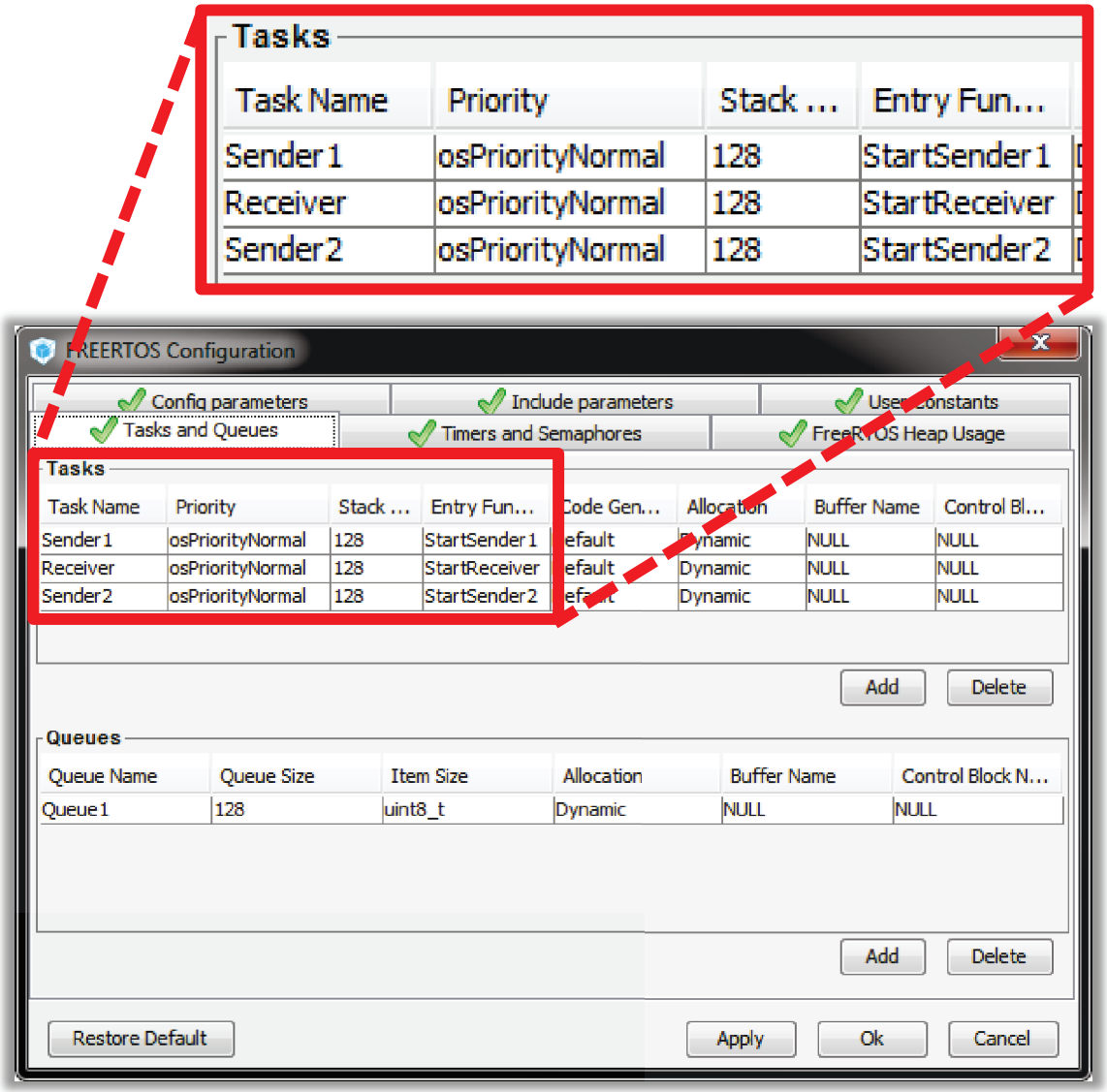
Queue Blocking

- After calling ‘osMessageGet’
- If any data are not in queue, the task I blocked for configured time
- If the data are in queue, the task will continue



Two senders lab

- Two sending tasks
- One receivers tasks
- **Same priorities**



life.augmented

Two senders lab

- Two sending tasks
- They are same no change necessary

```
void StartSender1(void const *
argument)
{
  /* USER CODE BEGIN StartSender1 */
  /* Infinite loop */
  for (;;) {
    printf("Sender1\n");

osMessagePut(Queue1Handle,0x1,200);
    printf("Sender1 delay\n");
    osDelay(2000);
  }
  /* USER CODE END StartSender1 */
}
```

```
void StartSender2(void const *
argument)
{
  /* USER CODE BEGIN StartSender2 */
  /* Infinite loop */
  for (;;) {
    printf("Sender2\n");

osMessagePut(Queue1Handle,0x2,200);
    printf("Sender2 delay\n");
    osDelay(2000);
  }
  /* USER CODE END StartSender2 */
}
```

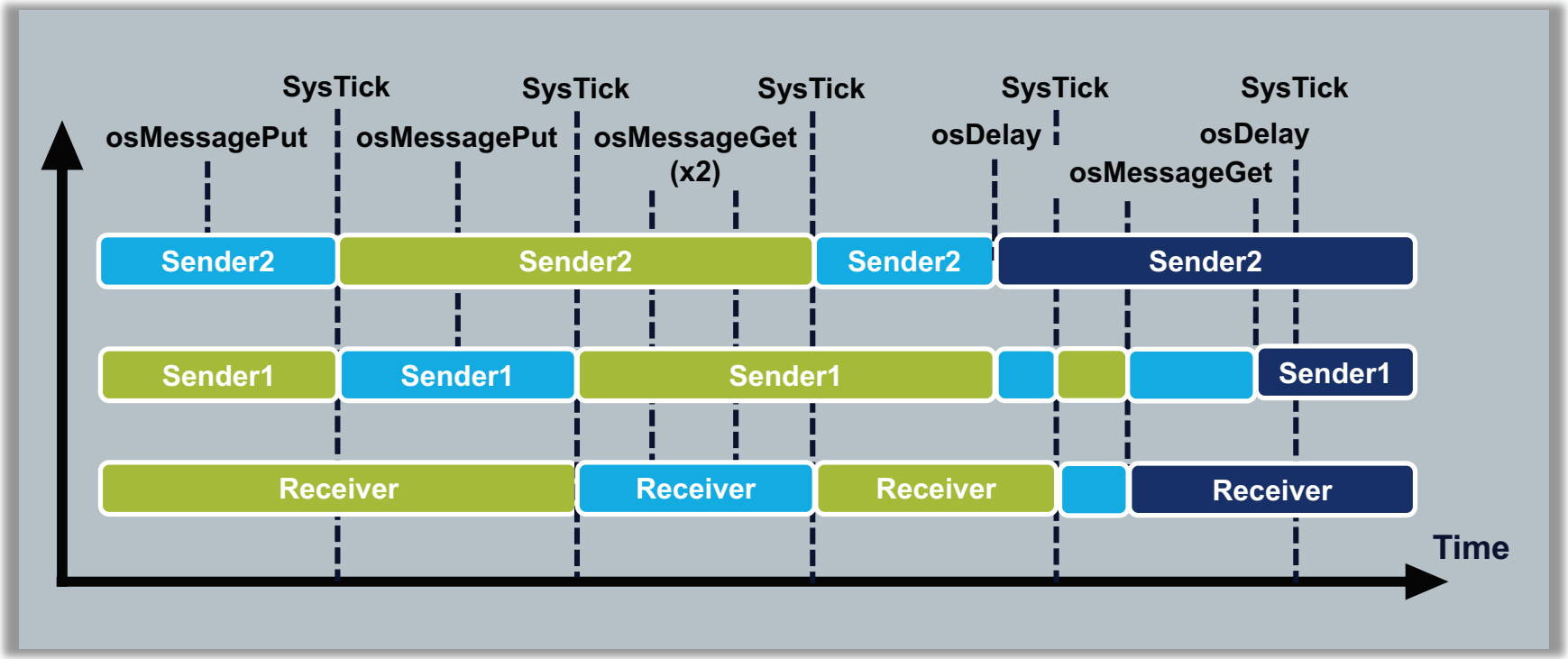
Two senders lab

- Simple receiver

```
/* StartReceiver function */
void StartReceiver(void const * argument)
{
  /* USER CODE BEGIN StartReceiver */
  osEvent retval;
  /* Infinite loop */
  for (;;) {
    retval = osMessageGet(Queue1Handle,
1000);
    printf("Receiver %d\n“, retval.value.v);
  }
  /* USER CODE END StartReceiver */
}
```

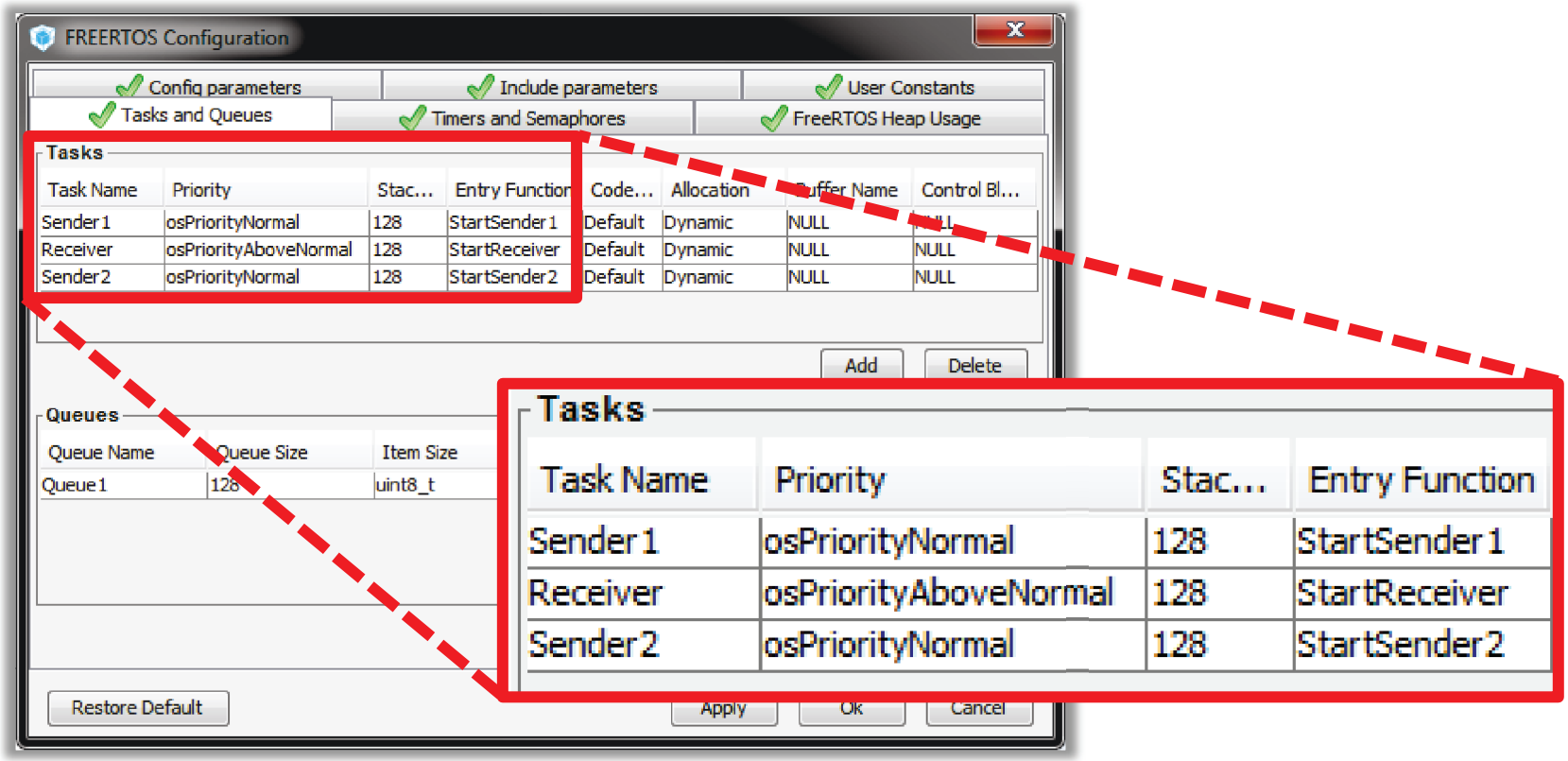
Queue Blocking

- What we can see in debug now?
- Because tasks have same priority, receiver will get data from queue after both task put data into queue
- What happened if will be more tasks?



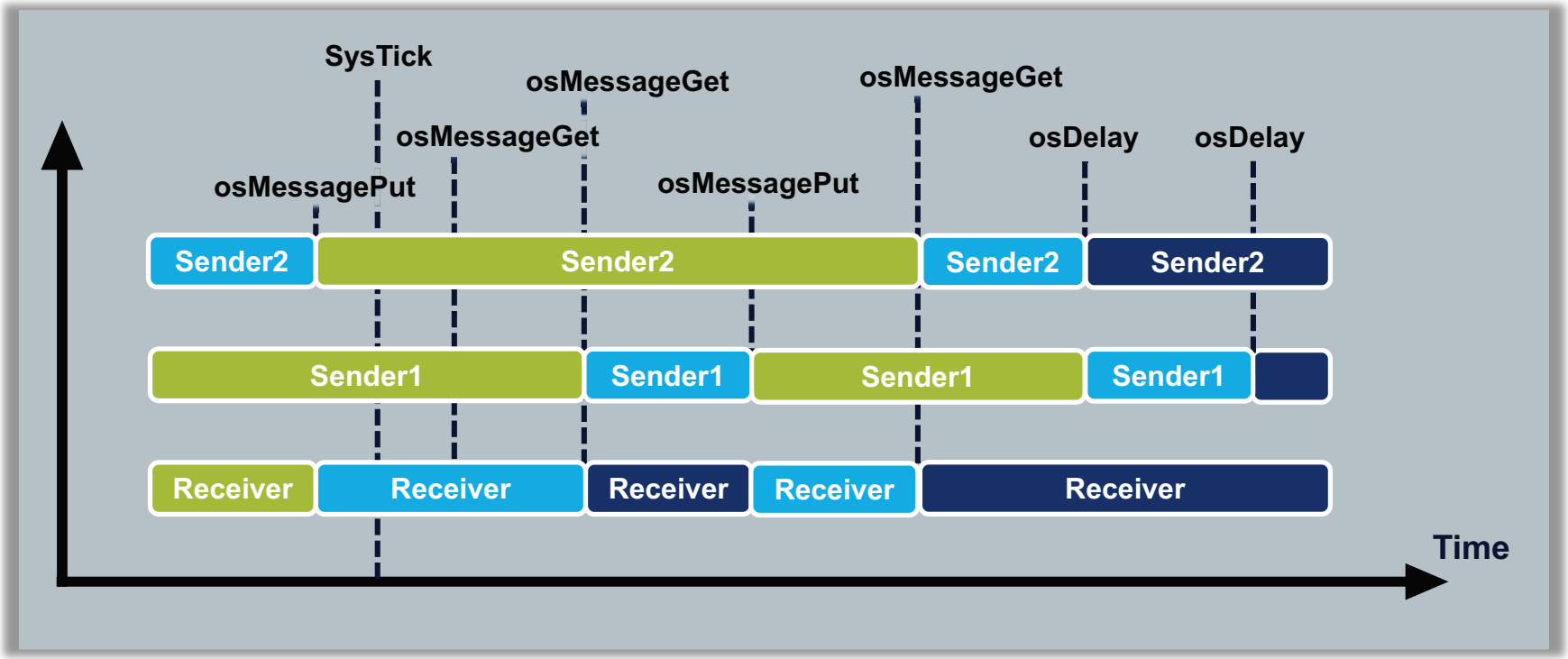
Receiver with higher priority lab

- Senders have same priority
- Receiver have higher priority than senders



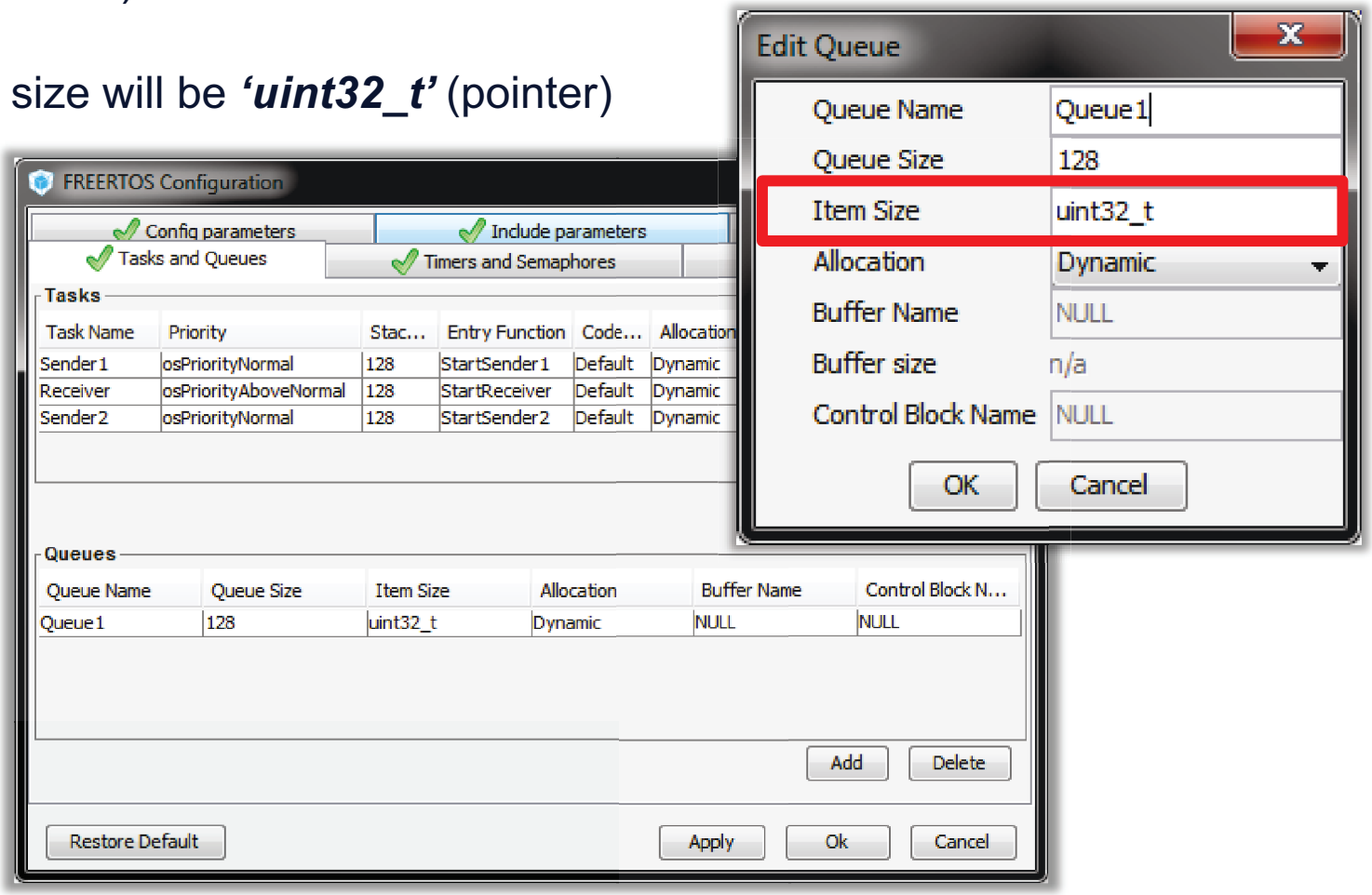
Queue Blocking

- What we can see in debug now?
- Because tasks have same priority, receiver will get data from queue after both task put data into queue
- What happened if will be more tasks?



Queue items lab

- Queues allow to define ‘type’ (different variables or structures which the queue use)
- Item size will be ***‘uint32_t’*** (pointer)



Queue items lab

- Create new structure type for data

```
/* Define string literal type that will be passed on the queue.
*/
const char *type1str = “Hello, World!”;
```

- Sent data from Sender task

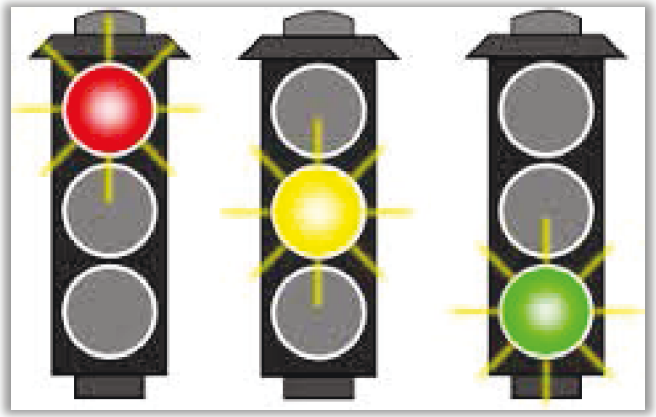
```
void StartSender1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for (;;) {
    printf("Task1\n");
    osMessagePut(Queue1Handle, (uint32_t)&type1str ,200);
    printf("Task1 delay\n");
    osDelay(2000);
  }
  /* USER CODE END 5 */
}
```

Queue items lab

- Receiver data from sender task

```
/* StartReceiver function */
void StartReceiver(void const * argument)
{
  /* USER CODE BEGIN StartReceiver */
  osEvent retval;
  /* Infinite loop */
  for (;;) {
    retval = osMessageGet(Queue1Handle, 4000);
    if (retval.value.p) {
      printf("Receive: %s\n“, (const char *)retval.value.p);
    } else {
      printf("Receive nothing\n");
    }
  }
  /* USER CODE END StartReceiver */
}
```

FreeRTOS Semaphores



life.augmented

Semaphores

- Used for synchronization between
  - Tasks
  - Interrupt and task
- Two types
  - Binary semaphores
  - Counting semaphores
- Binary semaphore
  - Have only one ‘token’
  - Using to synchronize one action
- Counting semaphore
  - Have multiple ‘tokens’
  - Synchronize multiple actions

Binary Semaphore

Binary Semaphore

- Semaphore creation

```
osSemaphoreId osSemaphoreCreate(const osSemaphoreDef_t *sema_def, int32_t
count)
```

Semaphore ‘tokens’ for binary semaphore is 1.

- Wait for Semaphore release

```
int32_t osSemaphoreWait(osSemaphoreId sema_id, uint32_t millisec)
```

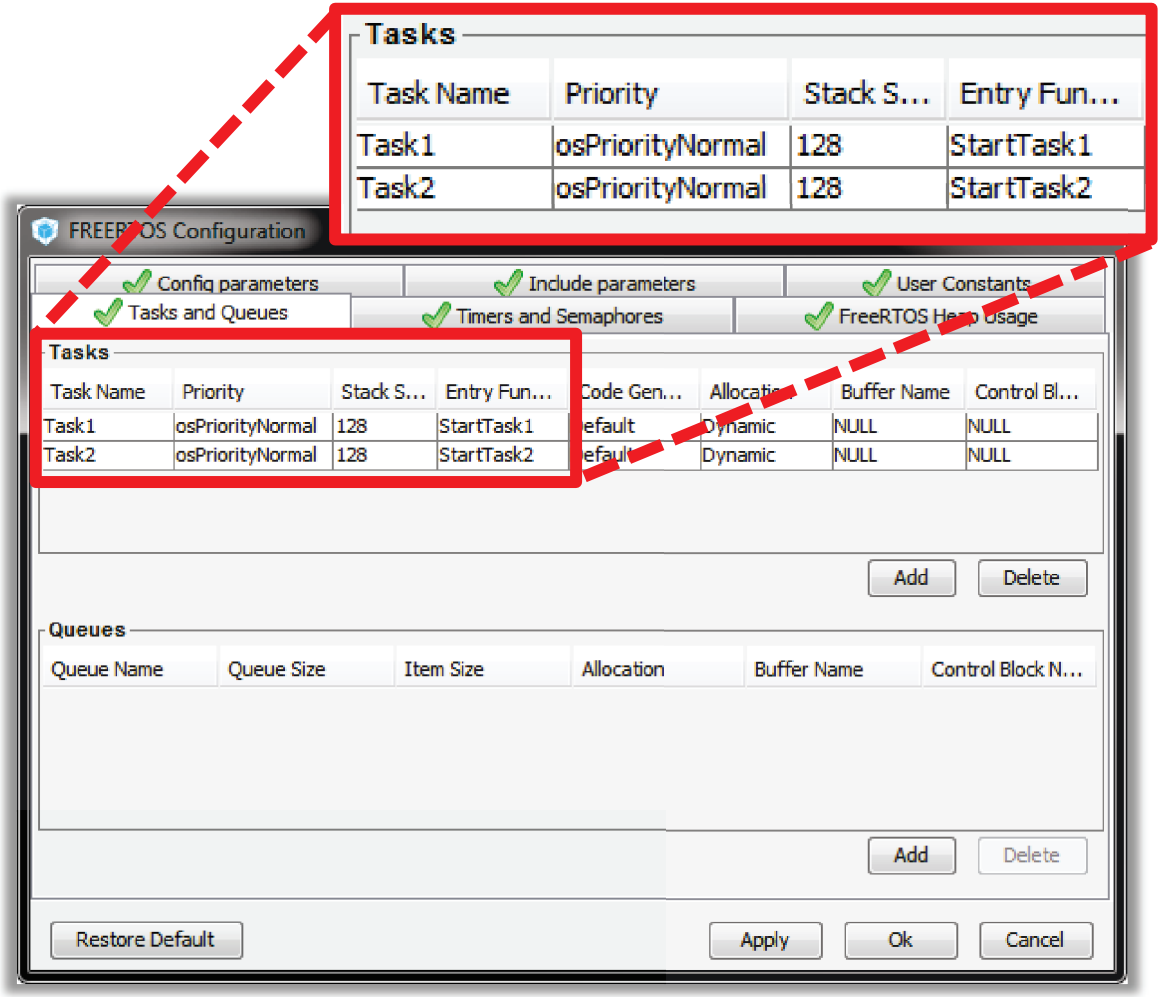
Number of ‘tokens’ in semaphore

- Semaphore release

```
osStatus osSemaphoreRelease(osSemaphoreId sema_id)
```

Binary Semaphore lab

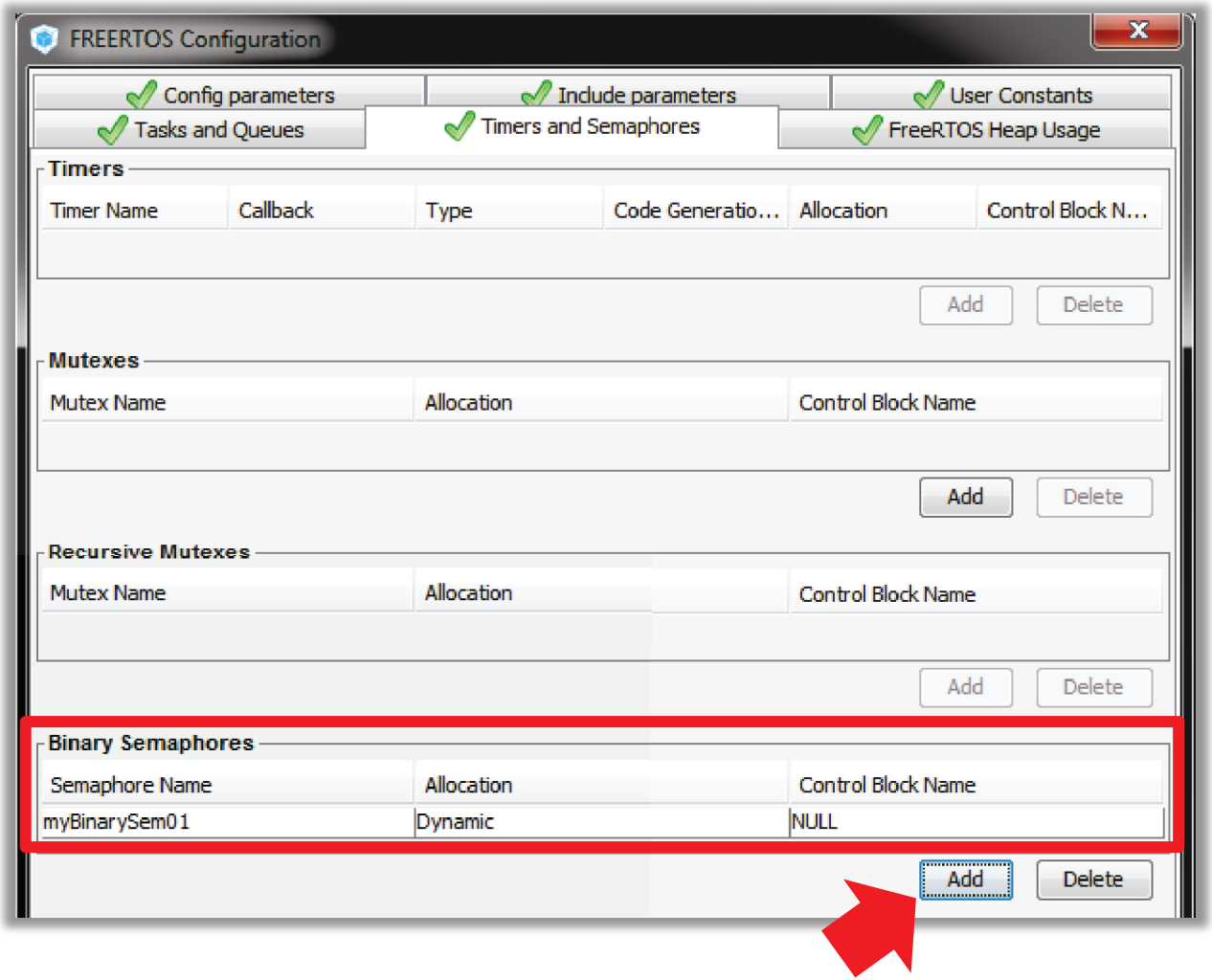
- Create two tasks
- With same priority
- Button Add
- Set parameters
- Button OK



life.augmented

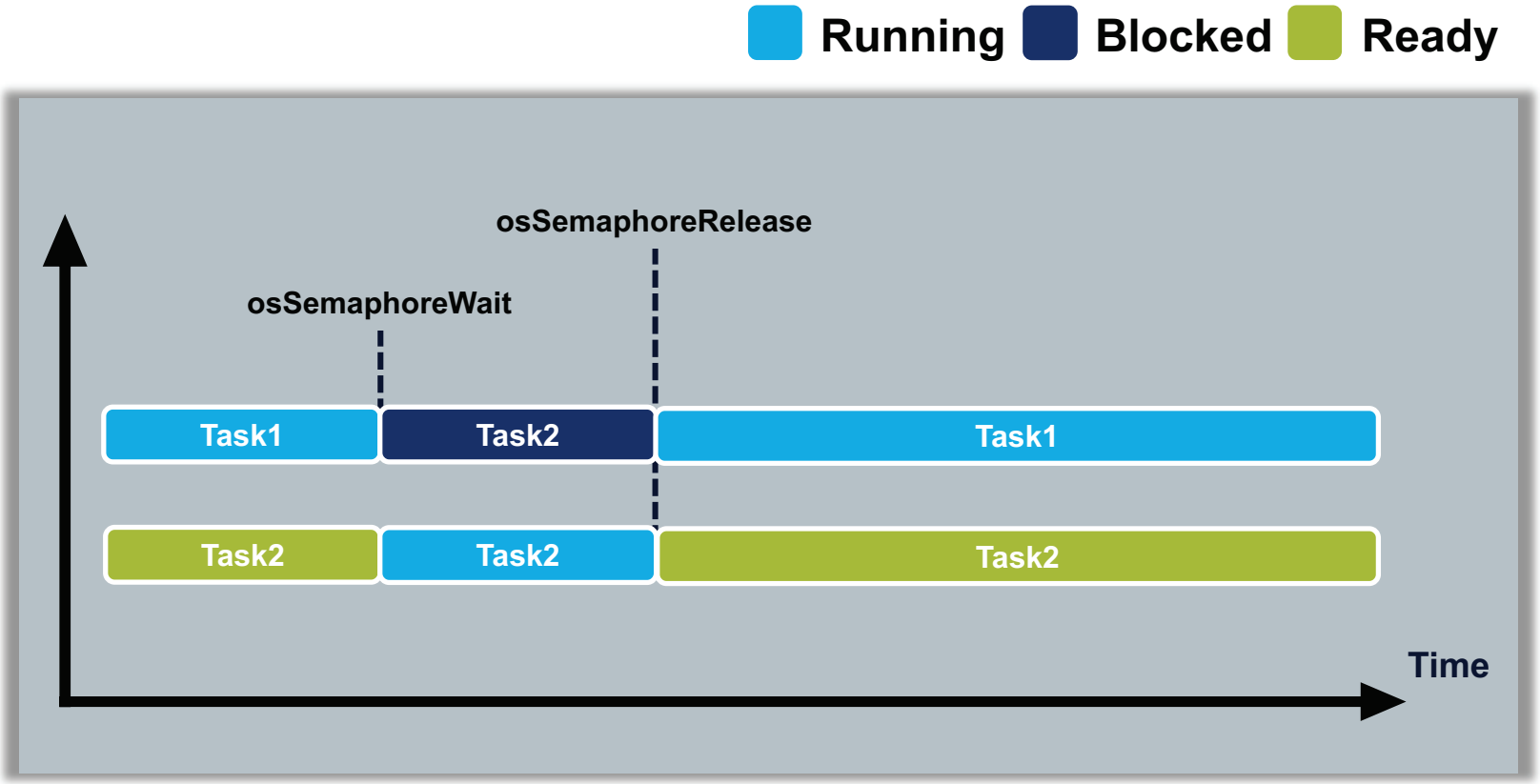
Binary Semaphore lab

- Create binary binary semaphore (in Timers and Semaphores)
- Button Add
- Set name
- Button OK



Binary Semaphore lab

- Task1 is synchronized with Task2



Binary Semaphore lab

- Semaphore handle definition

```
/* Private variables ---------------------------------------------
- */
osThreadId Task1Handle;
osThreadId Task2Handle;
osSemaphoreId myBinarySem01Handle;
```

- Semaphore creation

```
  /* Create the semaphores(s) */
  /* definition and creation of myBinarySem01 */
  osSemaphoreDef(myBinarySem01);
  myBinarySem01Handle =
osSemaphoreCreate(osSemaphore(myBinarySem01), 1);
```

Binary Semaphore lab

- Semaphore use
- If tasks/interrupt is done the semaphore is released

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN StartTask1 */
  /* Infinite loop */
  for (;;) {
    osDelay(2000);
    printf("Task1 Release semaphore\n");
    osSemaphoreRelease(myBinarySem01Handle);
  }
  /* USER CODE END StartTask1 */
}
```

Binary Semaphore lab

- Semaphore Wait
- Second task waits on semaphore release
  After releasetask is unbocked and continue in work

```
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for (;;) {
    if (osSemaphoreWait(myBinarySem01Handle, 1000) == osOK) {
      printf("Task2 synchronized\n");
    }
  }
  /* USER CODE END StartTask2 */
}
```

Counting semaphore

- OS API same as Binary semaphore
- Semaphore creation

```
osSemaphoreId osSemaphoreCreate(const osSemaphoreDef_t *sema_def, int32_t
count)
```

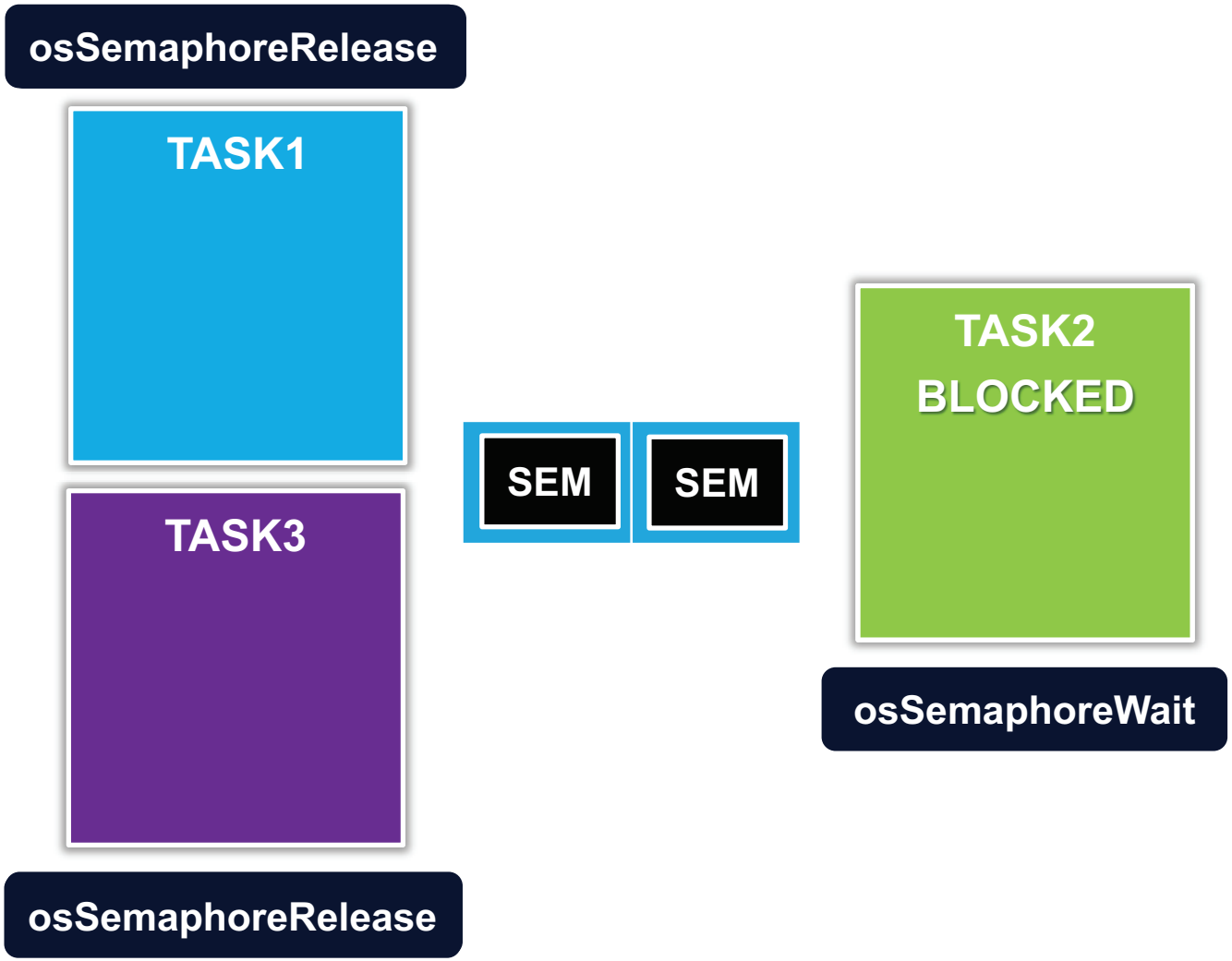
- Wait for Semaphore release

```
int32_t osSemaphoreWait(osSemaphoreId sema_id, uint32_t millisec)
```

- Semaphore release

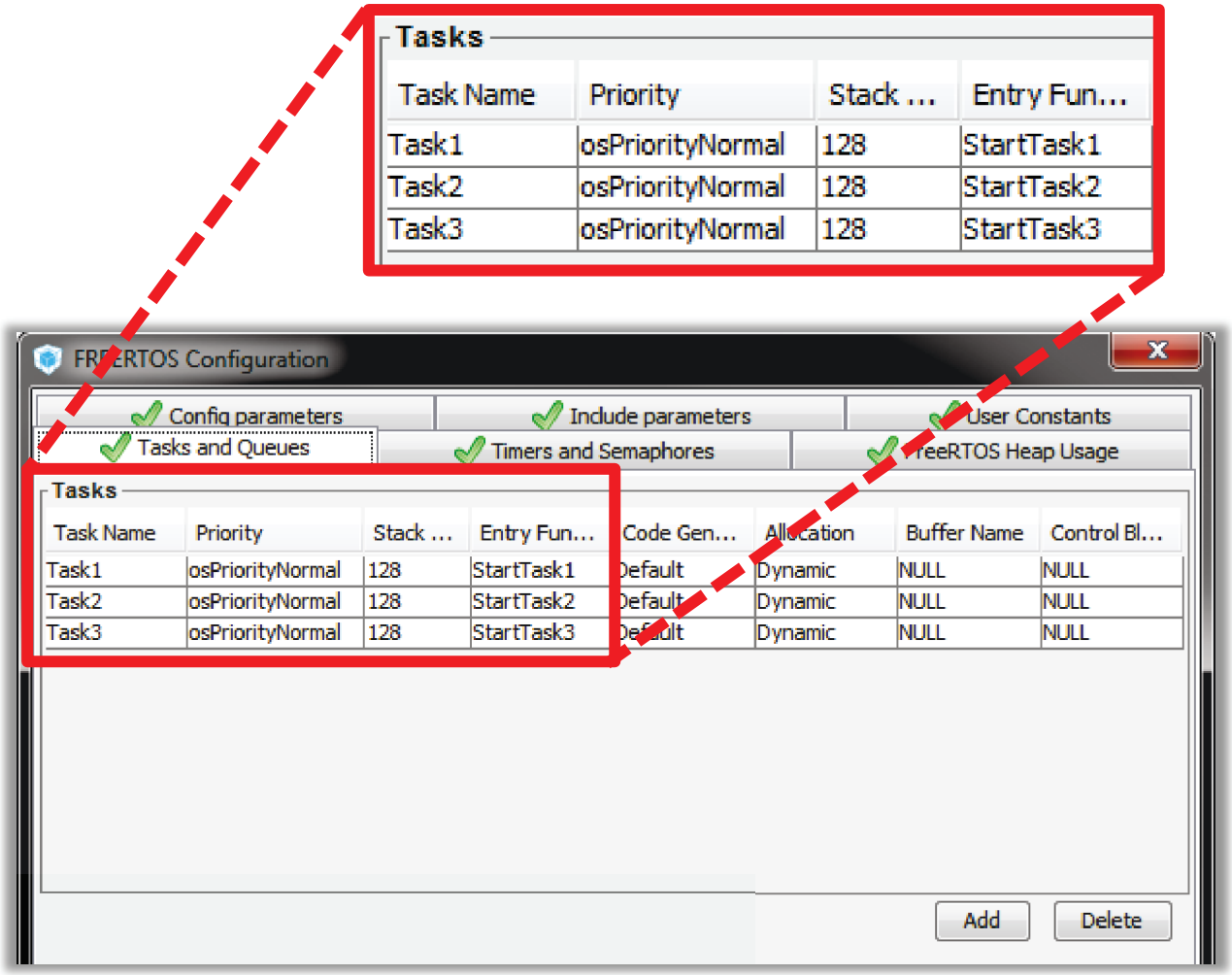
```
osStatus osSemaphoreRelease(osSemaphoreId sema_id)
```

Counting Semaphore



Counting Semaphore lab

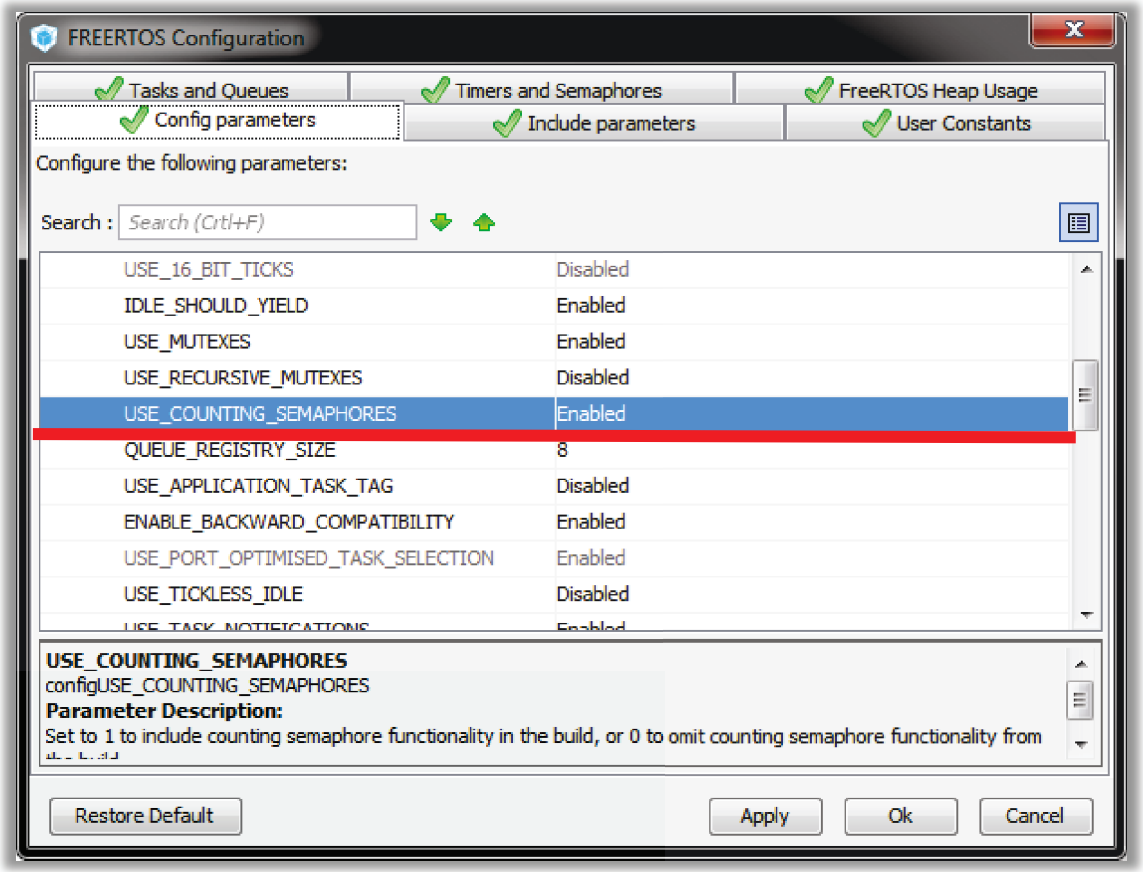
- Create three tasks
- With same priority
- Button Add
- Set parameters
- Button OK



Counting Semaphore lab

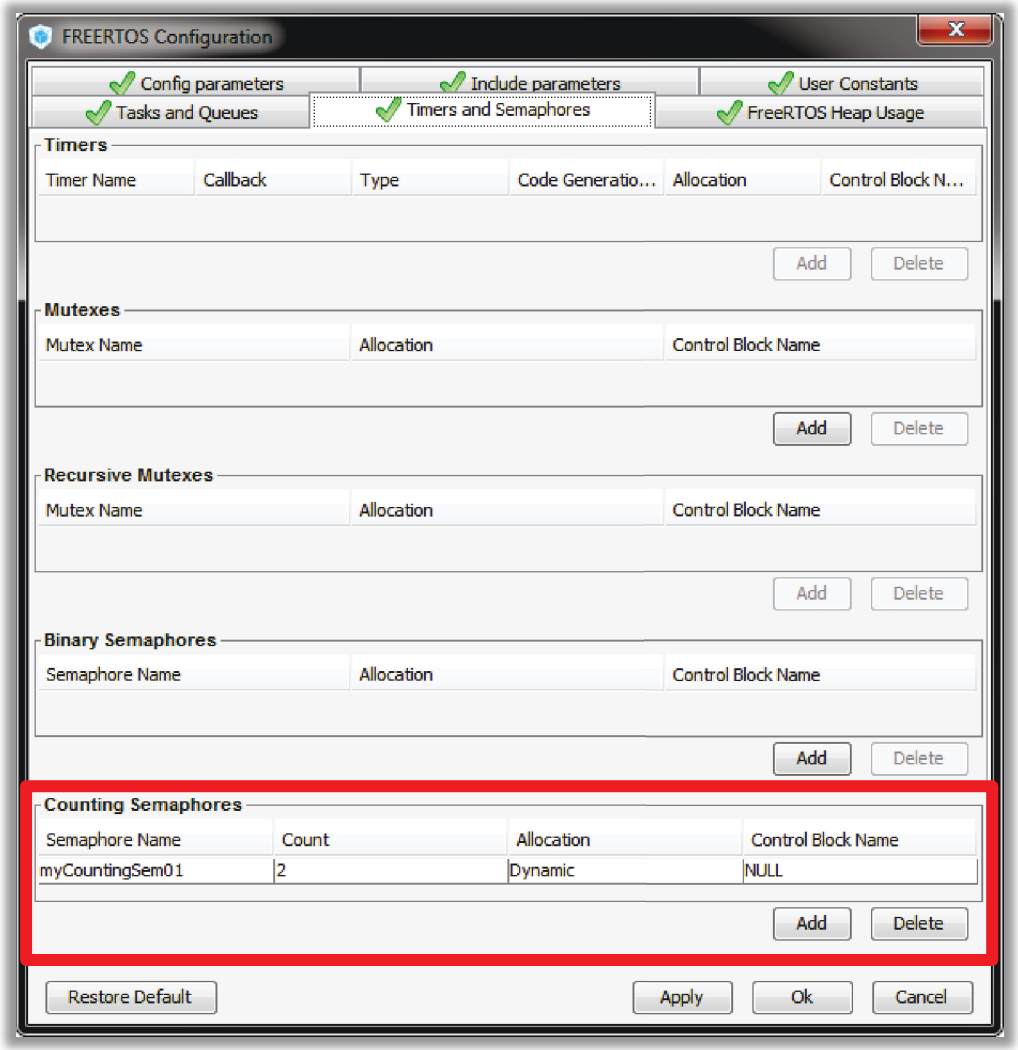
- Config parameters
- Enable kernel settings
- Button OK

USE_COUNTING_SEMAPHORES, Enabled



Counting Semaphore lab

- Create Counting semaphore
- Set count of tokens
- Button Add
- Set name
- Button OK



Counting Semaphore lab

- Create Counting semaphore

```
 /* Create the semaphores(s) */
  /* definition and creation of myCountingSem01 */
  osSemaphoreDef(myCountingSem01);
  myCountingSem01Handle =
osSemaphoreCreate(osSemaphore(myCountingSem01), 2);
```

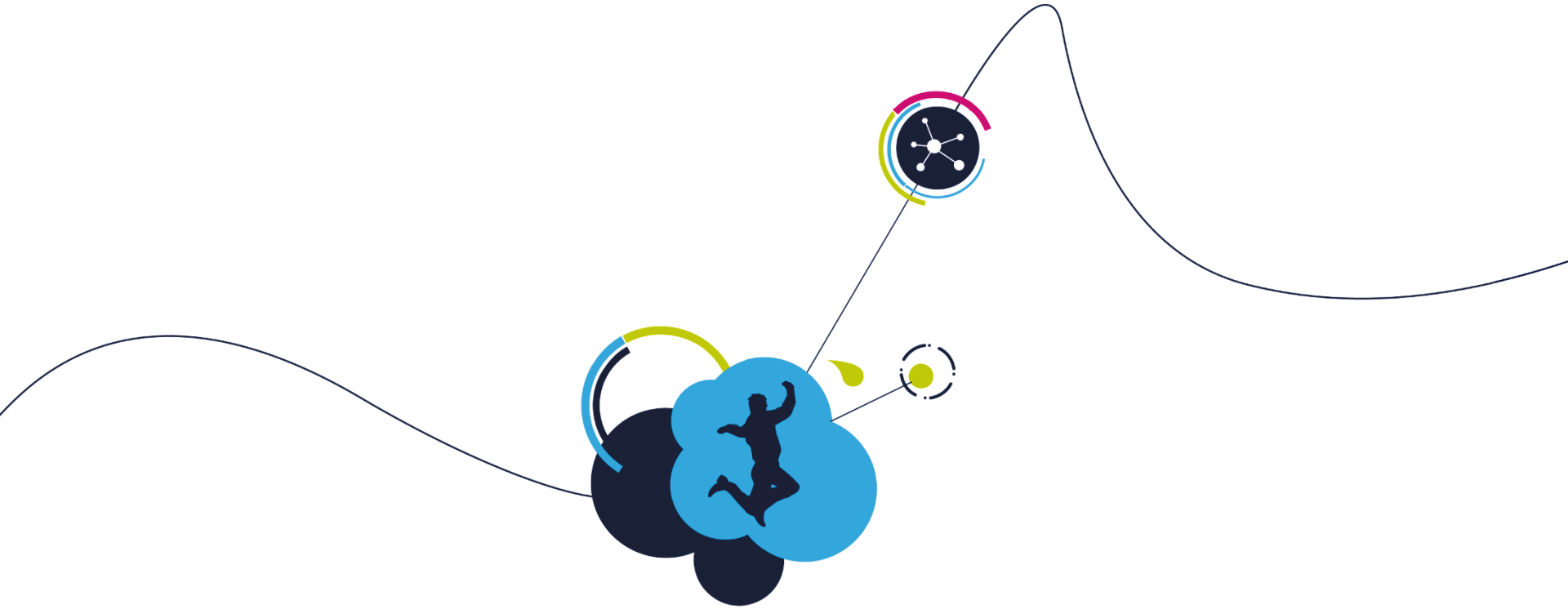
- Task1 and Task2 will be same

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for (;;) {
    osDelay(2000);
    printf("Task1 Release counting semaphore\n");
    osSemaphoreRelease(myCountingSem01Handle);
  }
  /* USER CODE END 5 */
}
```

Counting Semaphore lab

- Task3 will wait until semaphore will be 2 times released

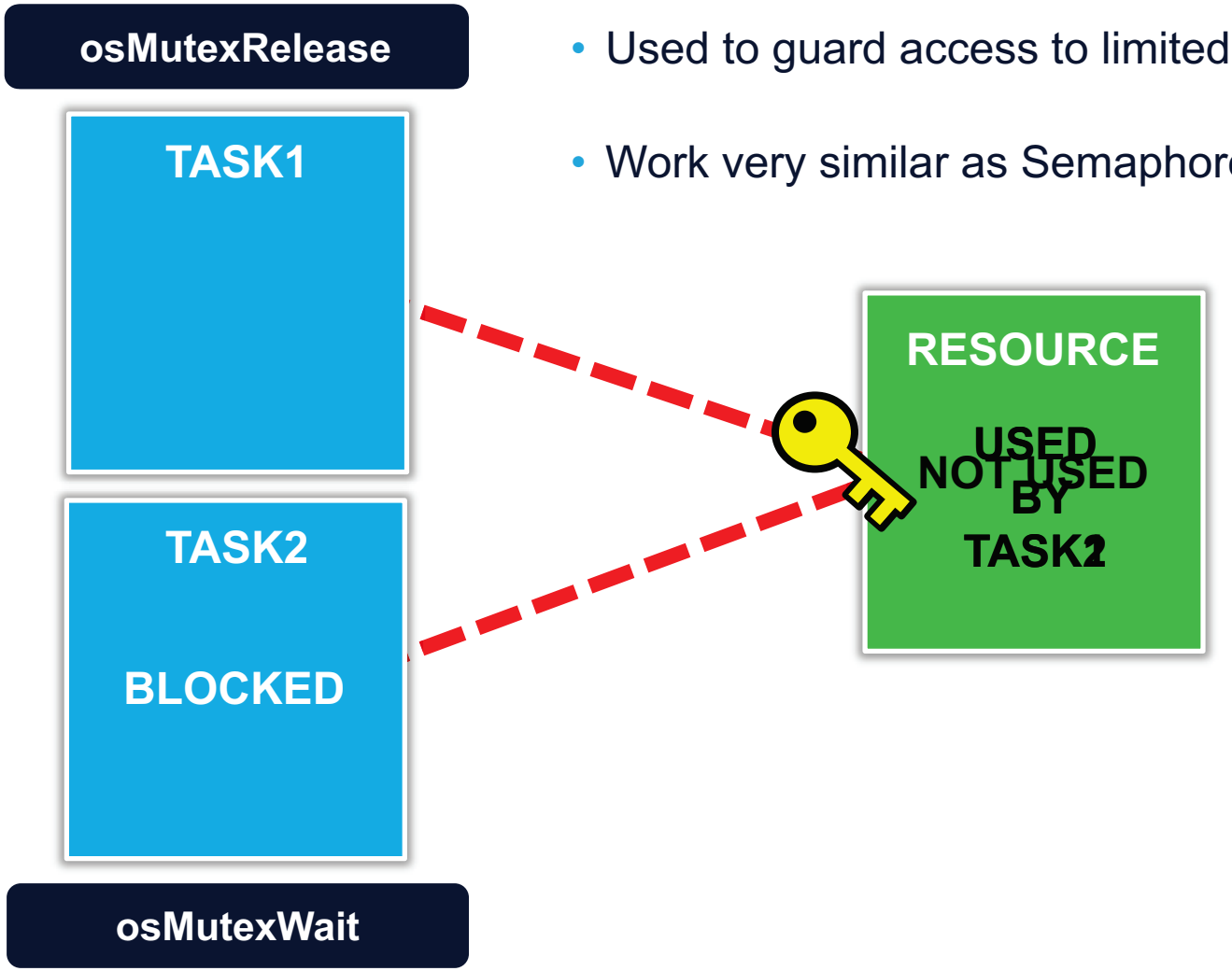
```
void StartTask3(void const * argument)
{
  /* USER CODE BEGIN StartTask3 */
  /* Infinite loop */
  for(;;)
  {
    osSemaphoreWait(myCountingSem01Handle, 4000);
    osSemaphoreWait(myCountingSem01Handle, 4000);
    printf("Task3 synchronized\n");
  }
  /* USER CODE END StartTask3 */
}
```



FreeRTOS
Mutex

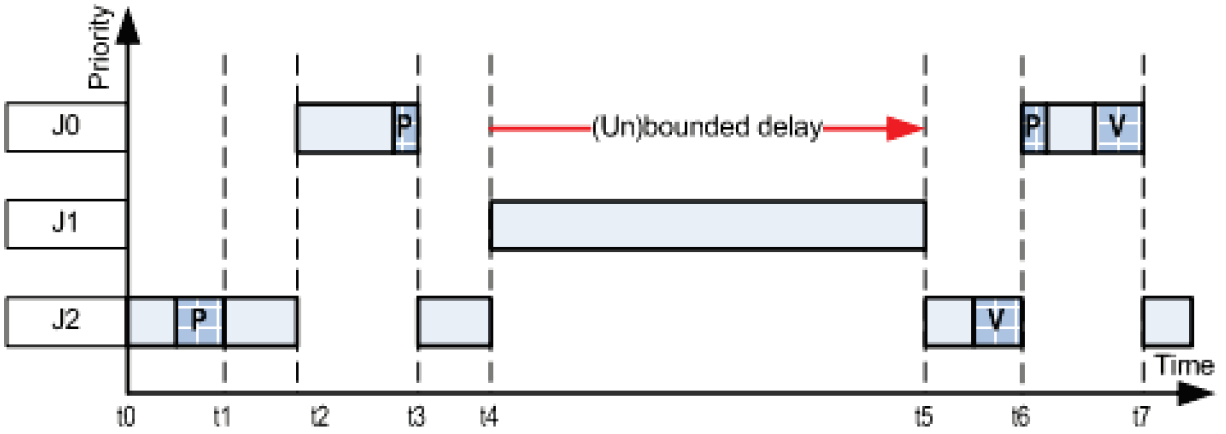
Mutex

- Used to guard access to limited recourse
- Work very similar as Semaphores

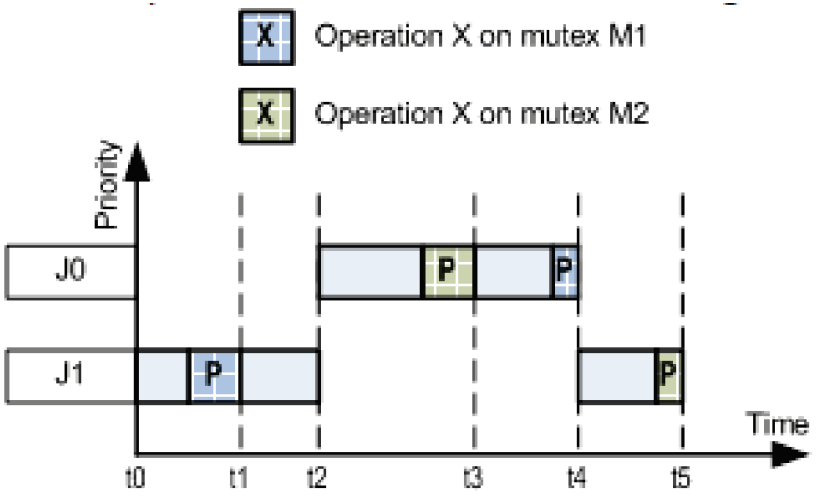


Mutex

- Priority Inversion



- Deadlock



Mutex

- Mutex creation

```
osMutexId osMutexCreate(const osMutexDef_t *mutex_def)
```

- Wait for Mutex release

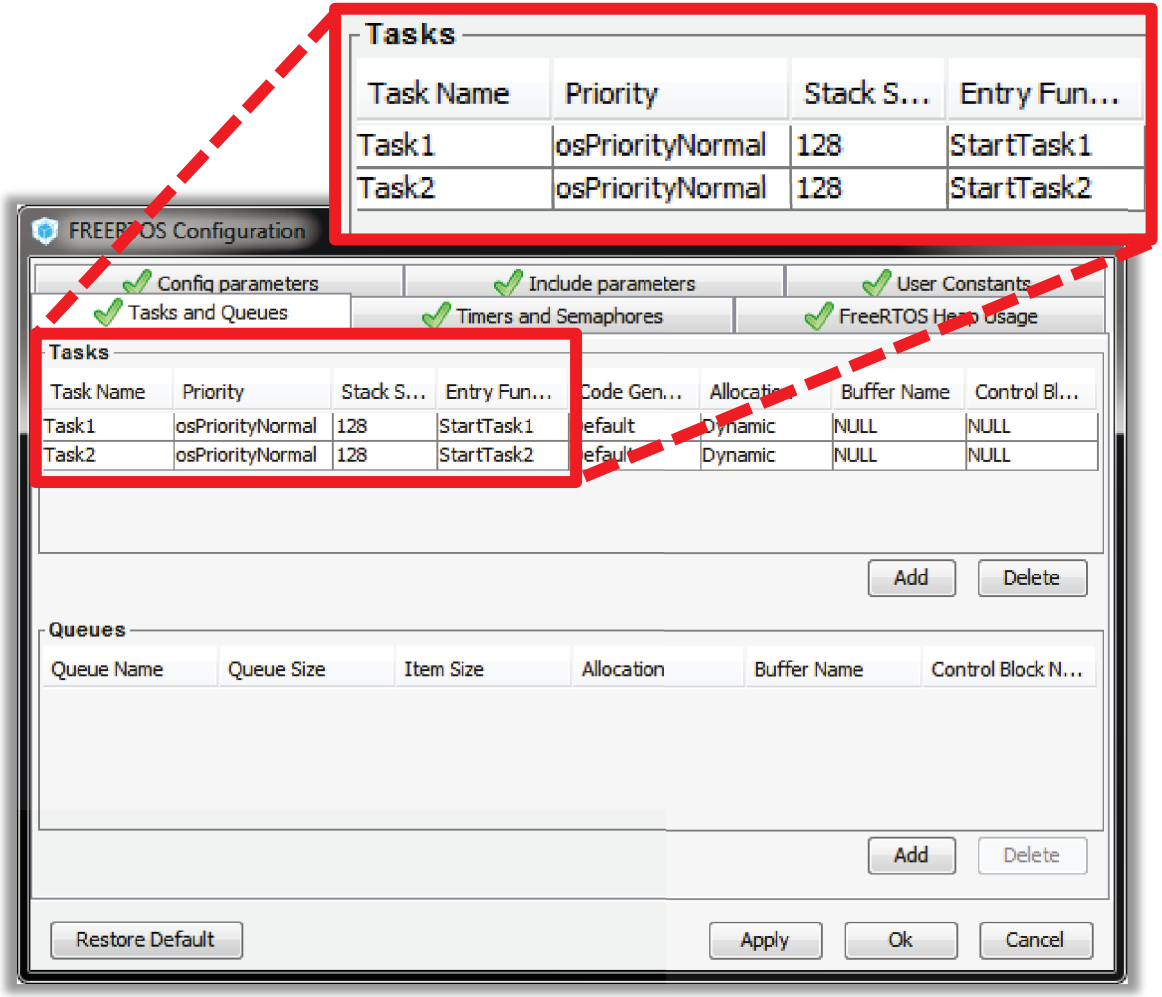
```
osMutexId osMutexCreate(const osMutexDef_t *mutex_def)
```

- Mutex release

```
osStatus osMutexRelease(osMutexId mutex_id)
```

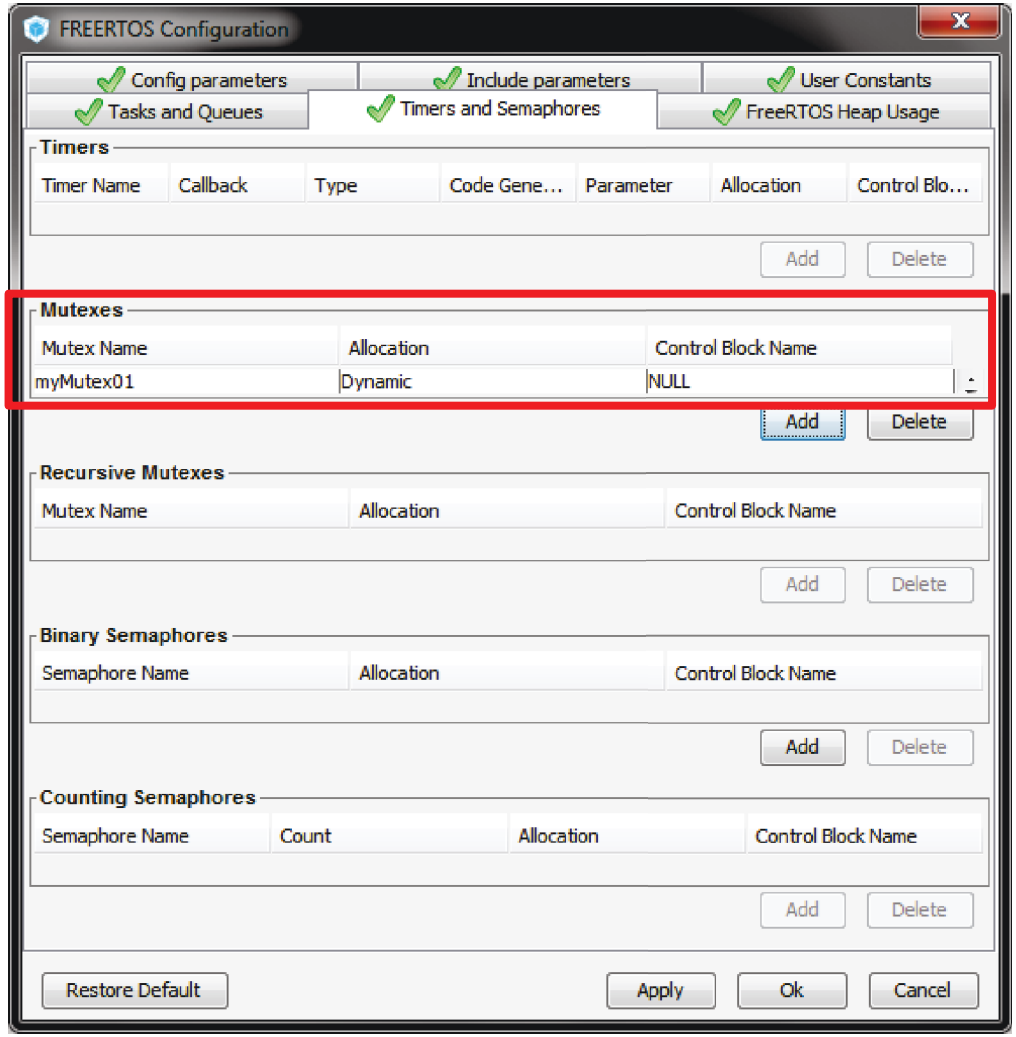
Mutex lab

- Create two tasks
- With same priority
- Button Add
- Set parameters
- Button OK



Mutex lab

- Create Mutex
- Button Add
- Set name
- Button OK



Mutex lab

- Mutex handle definition

```
/* Private variables ---------------------------------------------------*/
osThreadId Task1Handle;
osThreadId Task2Handle;
osMutexId myMutex01Handle;
```

- Mutex creation

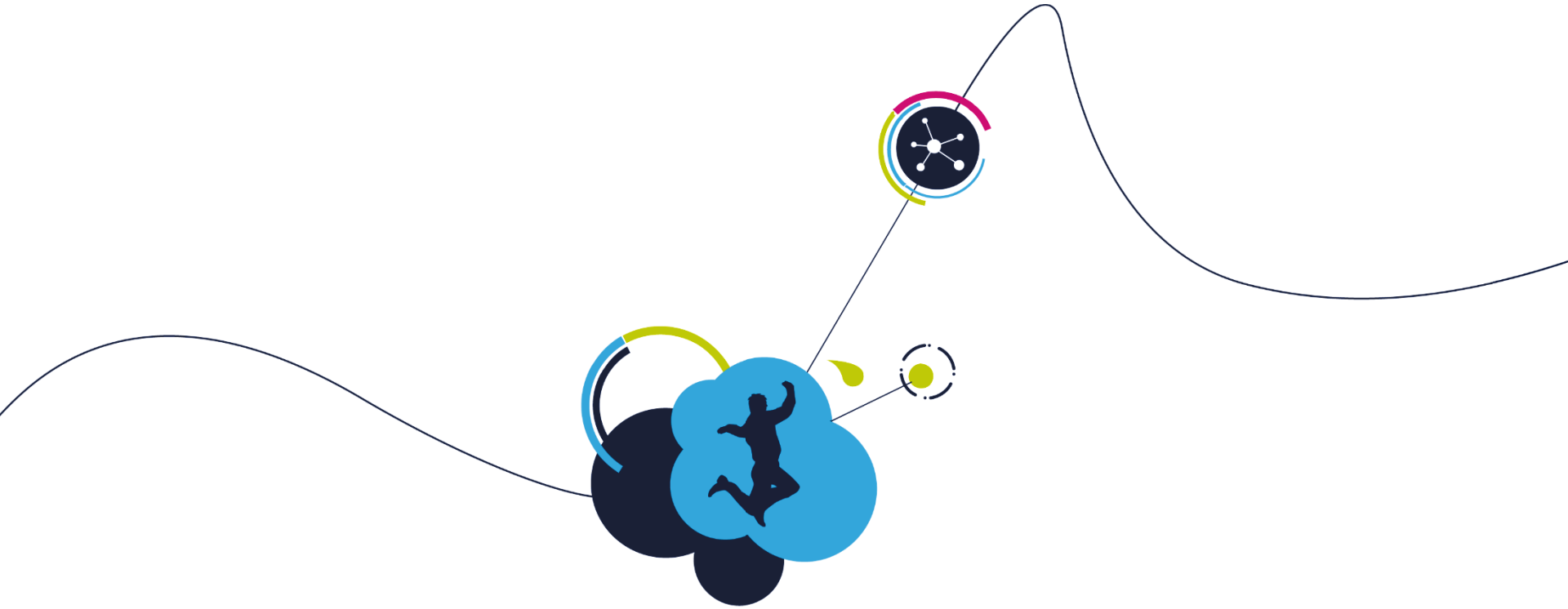
```
/* Create the mutex(es) */
/* definition and creation of myMutex01 */
osMutexDef(myMutex01);
myMutex01Handle = osMutexCreate(osMutex(myMutex01));
```

Mutex lab

- Task1 and Task2 using of Mutex

```
void StartTask1(void const *
argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(2000);

osMutexWait(myMutex01Handle,1000);
    printf("Task1 Print\n");
    osMutexRelease(myMutex01Handle);
  }
  /* USER CODE END 5 */
}
```

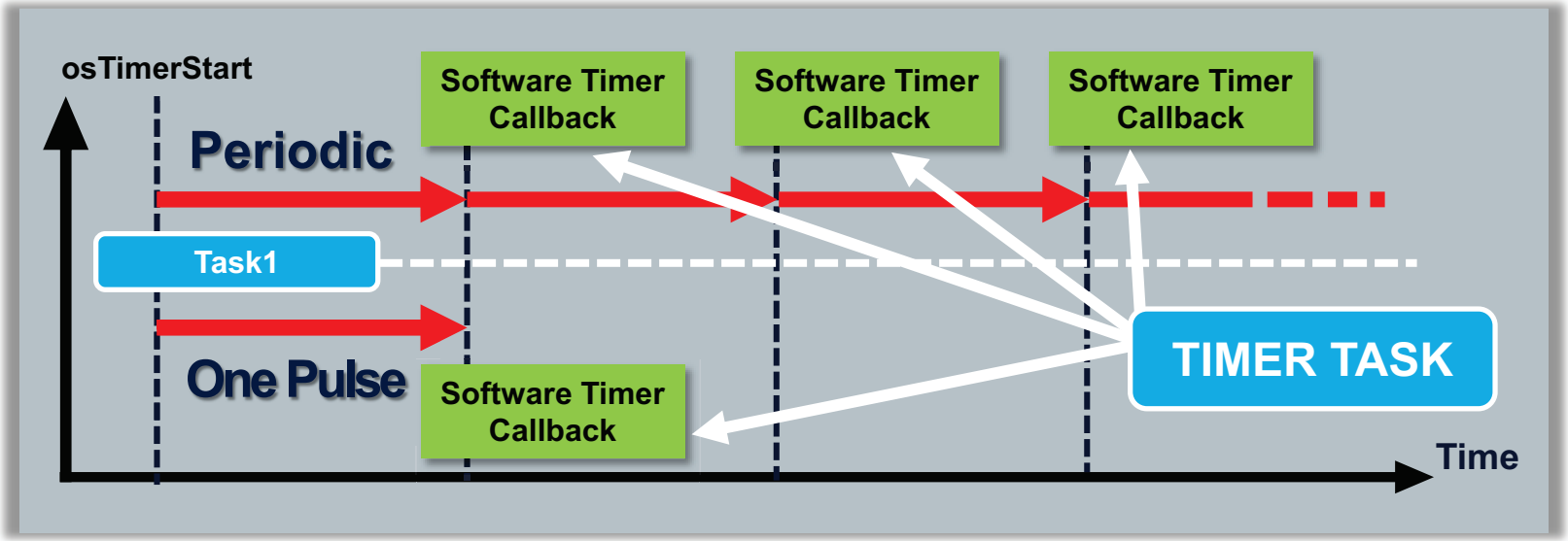
```
void StartTask2(void const *
argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(2000);

osMutexWait(myMutex01Handle,1000);
    printf("Task2 Print\n");
    osMutexRelease(myMutex01Handle);
  }
  /* USER CODE END StartTask2 */
}
```



FreeRTOS
Software Timers

Software Timers

- Software timer is one of component in RTOS
- Can extend number of Timers in STM32
- Are not precise but can handle periodic actions or delay actions
- Two modes (Periodic and One Pulse)



Software Timers

- Software timer creation

```
osTimerId osTimerCreate(const osTimerDef_t *tim_def, os_timer_type type, void *argument)
```

- Software timer start

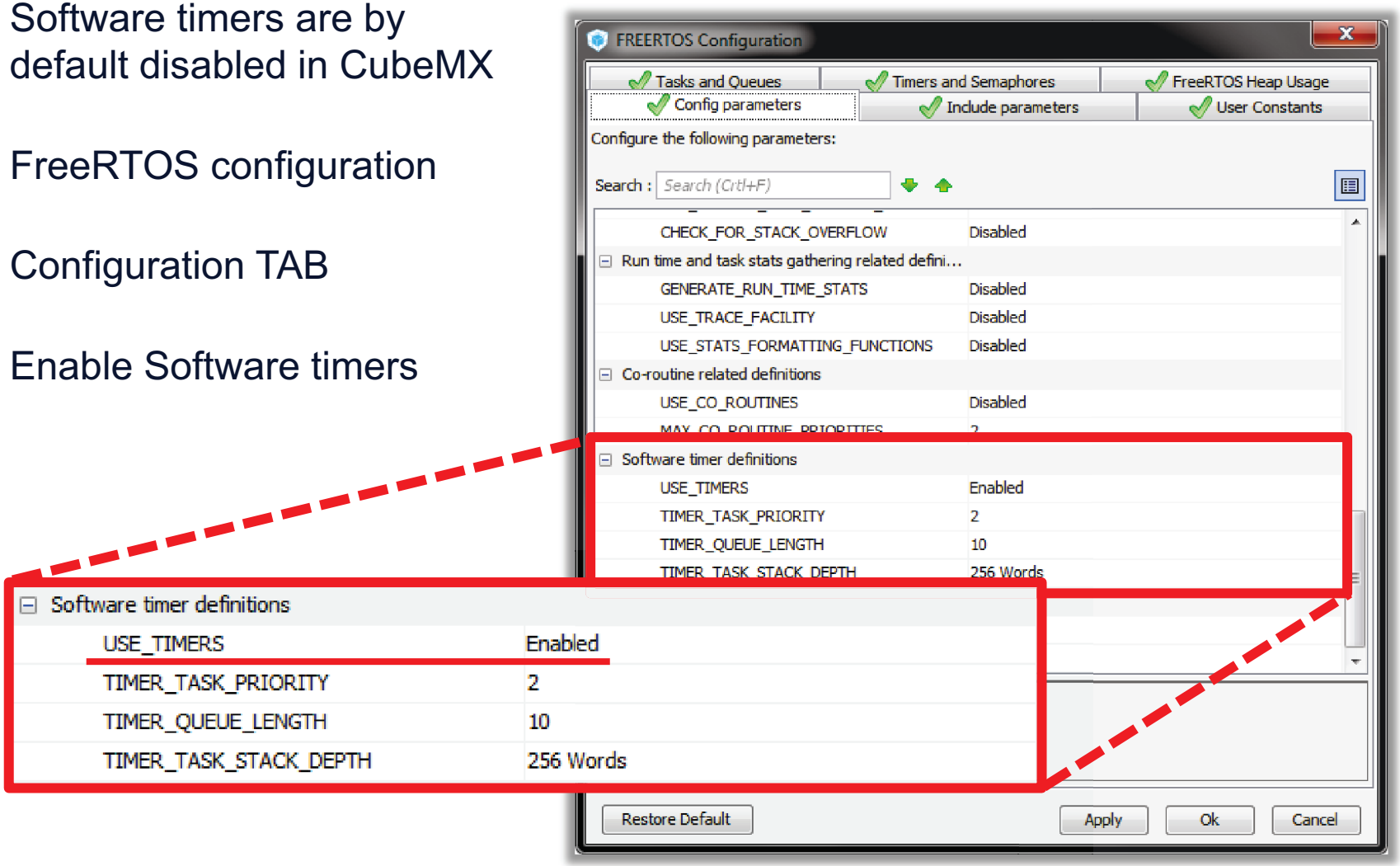
```
osStatus osTimerStart(osTimerId timer_id, uint32_t millisec)
```

- Software timer stop

```
osStatus osTimerStop(osTimerId timer_id)
```

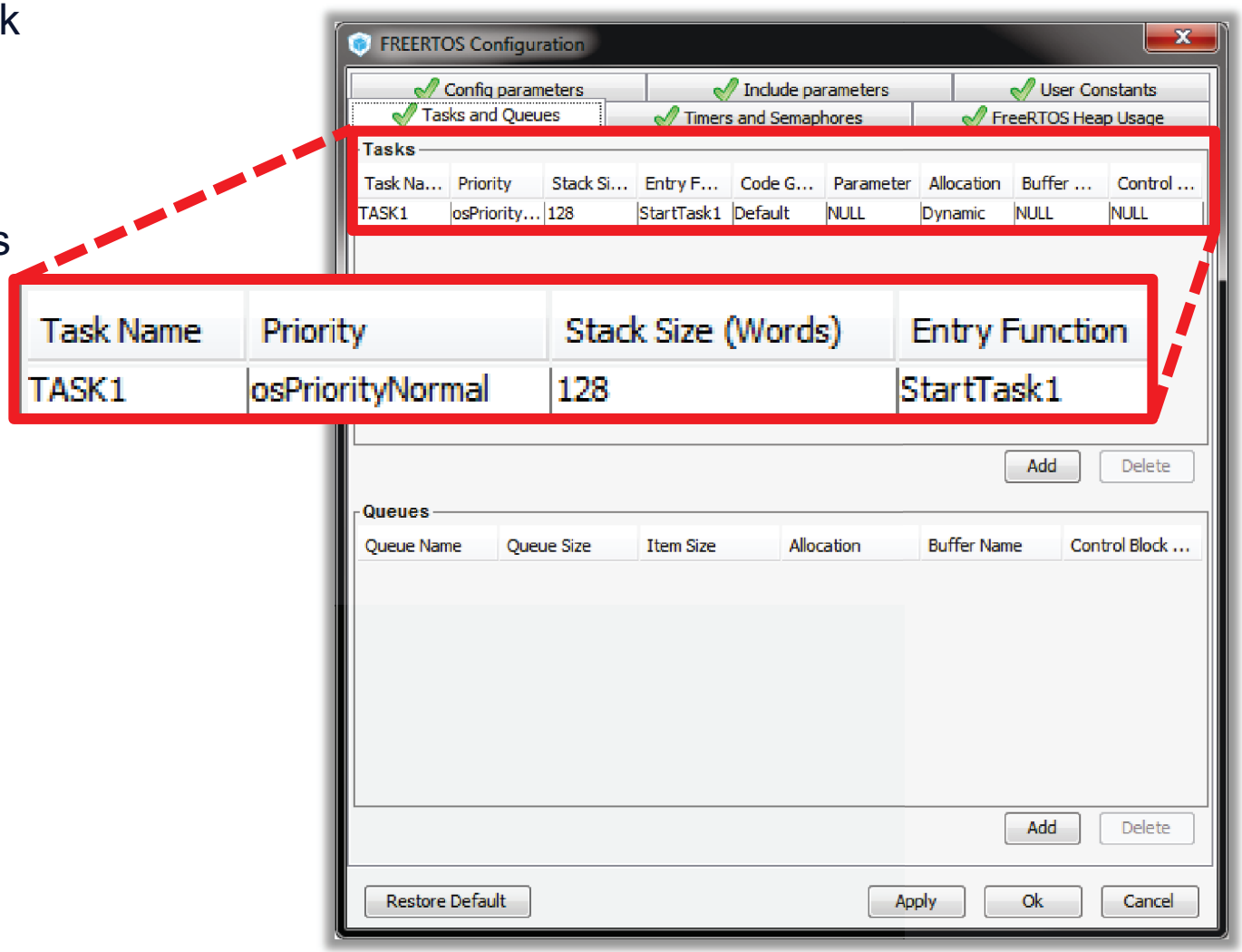
Software Timers lab

- Software timers are by default disabled in CubeMX
- FreeRTOS configuration
- Configuration TAB
- Enable Software timers



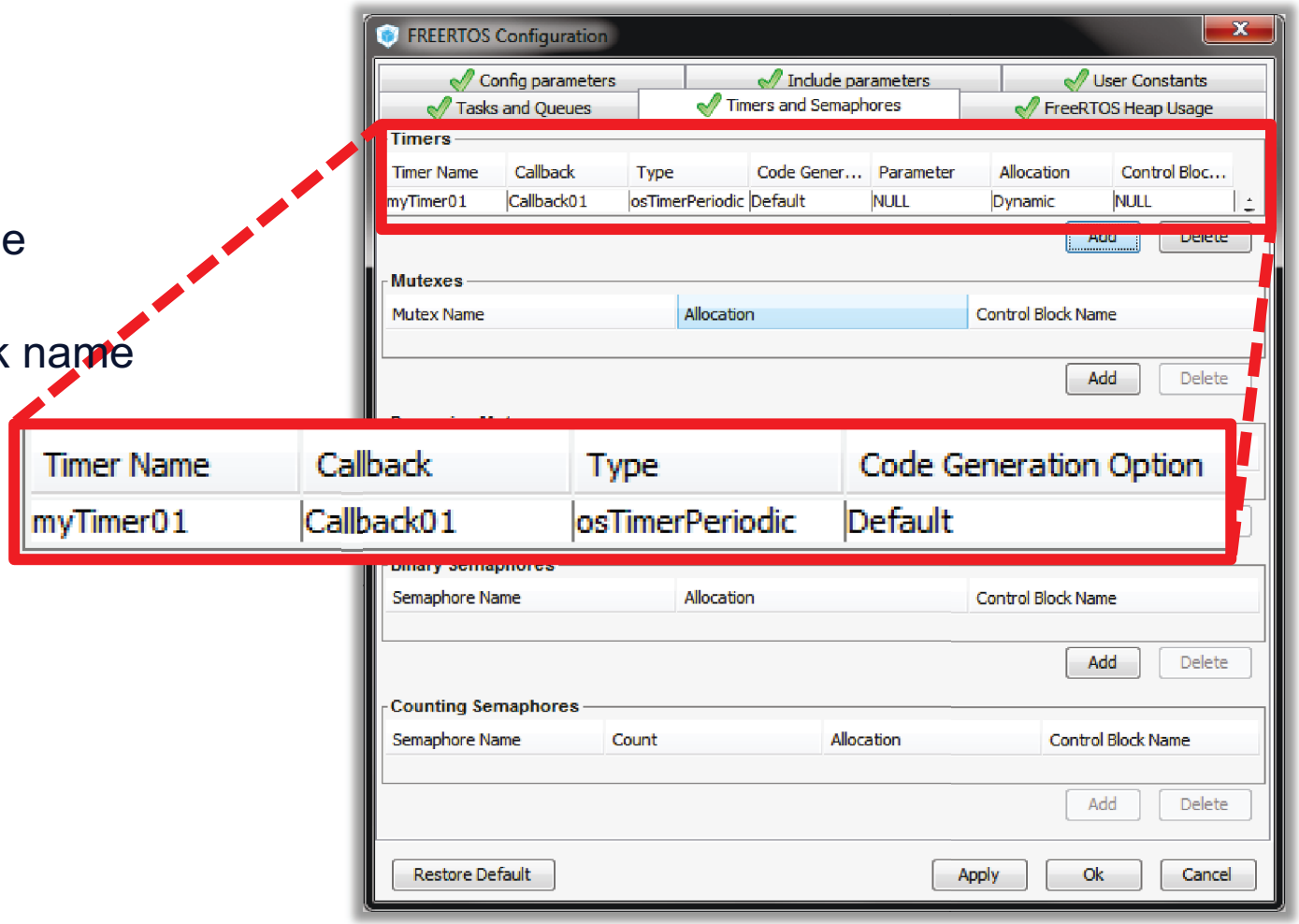
Software Timers lab

- Create one task
- Button Add
- Set parameters
- Set name
- Button OK



Software Timers lab

- Create Timer
- Button Add
- Set timer name
- Timer callback name
- Button OK



Software Timers lab

- Software timer handle definition

```
/* Private variables ---------------------------------------------------
---*/
osThreadId Task1Handle;
osTimerId myTimer01Handle;
```

- Software timer creation

```
/* Create the timer(s) */
/* definition and creation of myTimer01 */
osTimerDef(myTimer01, Callback01);
myTimer01Handle = osTimerCreate(osTimer(myTimer01), osTimerPeriodic, NULL);
```

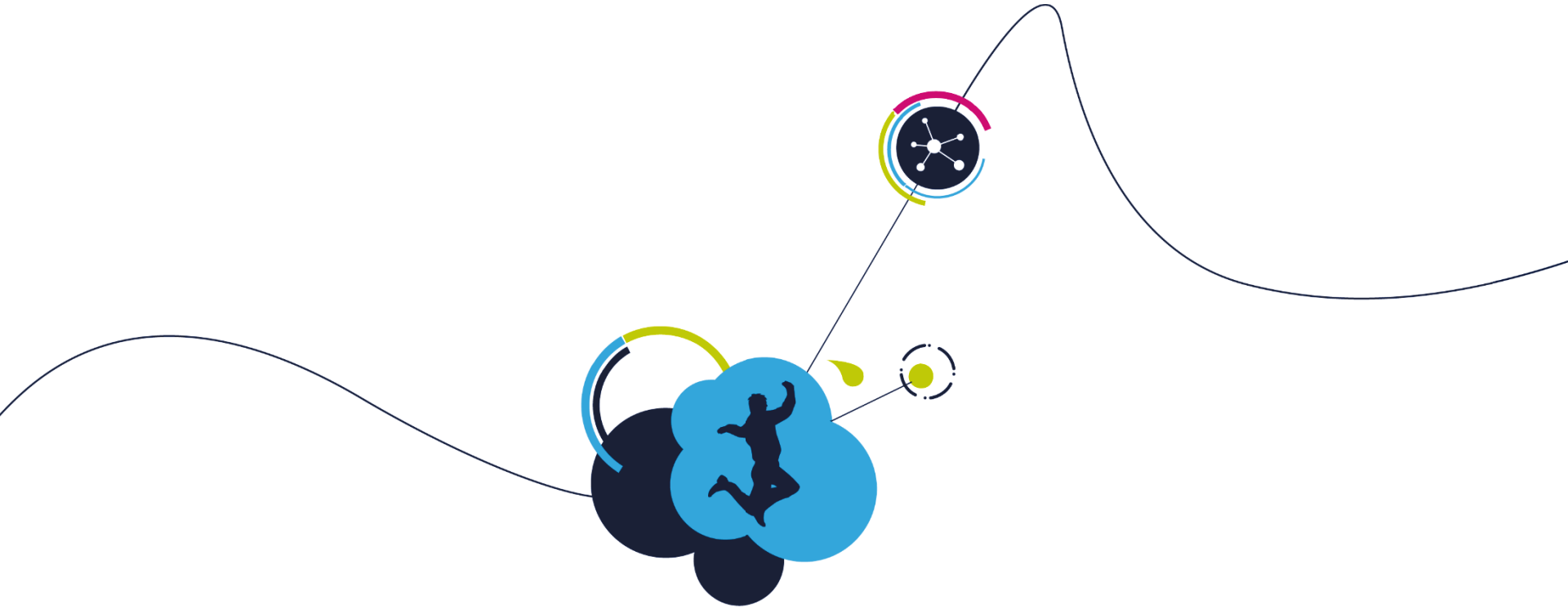
Software Timers lab

- Software timer start

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  osTimerStart(myTimer01Handle, 1000);
  for (;;) {
    osDelay(2000);
    printf("Task1 Print\n");
  }
  /* USER CODE END 5 */
}
```

- Software timer callback

```
/* Callback01 function */
void Callback01(void const * argument)
{
  /* USER CODE BEGIN Callback01 */
  printf("Timer Print\n");
  /* USER CODE END Callback01 */
}
```



End
Thanks for your attention

life.augmented