Introduction to embedded programming on STM32

FreeRTOS Tasks

Skoltech, 2019

Scope

- How FreeRTOS allocates processing time to each task within an application.
- How FreeRTOS chooses which task should execute at any given time.
- How the relative priority of each task affects system behavior.
- The states that a task can exist in

Why use FreeRTOS

- Abstracting away timing information
- Maintainability/Extensibility
- Modularity
- Team development
- Easier testing
- Code reuse
- Improved efficiency
- Idle time
- Flexible interrupt handling

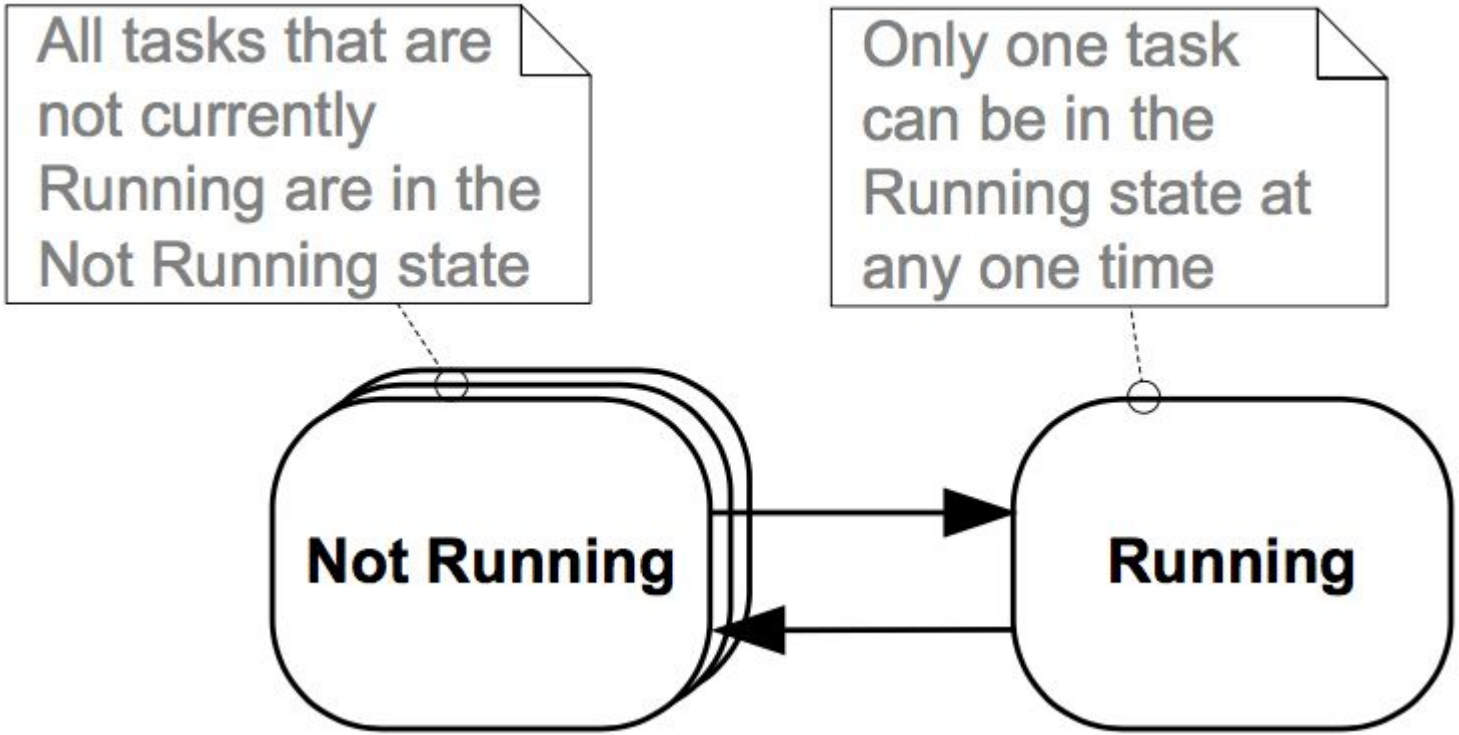
Task function

```
void my_task (void * args)

{

        // initialization

        while(1)

        {

                // task functionality implementation

        }

        vTaskDelete(NULL);

}
```

Task must :

- be void type
- take void * type parameters
- not return anything
- be explicitly deleted if task is no longer required

Top Level Task States



Creating Tasks

```
xTaskCreateStatic(TaskFunction_t pxTaskCode,                // pointer to the function
                  const char * const pcName,                // descriptive name
                  const uint32_t ulStackDepth,              // stack size (in words)
                  void * const pvParameters,                // task parameters
                  UBaseType_t uxPriority,                   // task priority
                  StackType_t * const puxStackBuffer,       // Stack buffer
                  StaticTask_t * const pxTaskBuffer )       // Task data structure
```

Creating Tasks

```
int main(void)

{

        rcc_config();

        xTaskCreateStatic(...);   // Create first task

        xTaskCreateStatic(...);   // Create another task

        vTaskStartScheduler(); // Start scheduler

        return 0;

}
```

Task Priorities

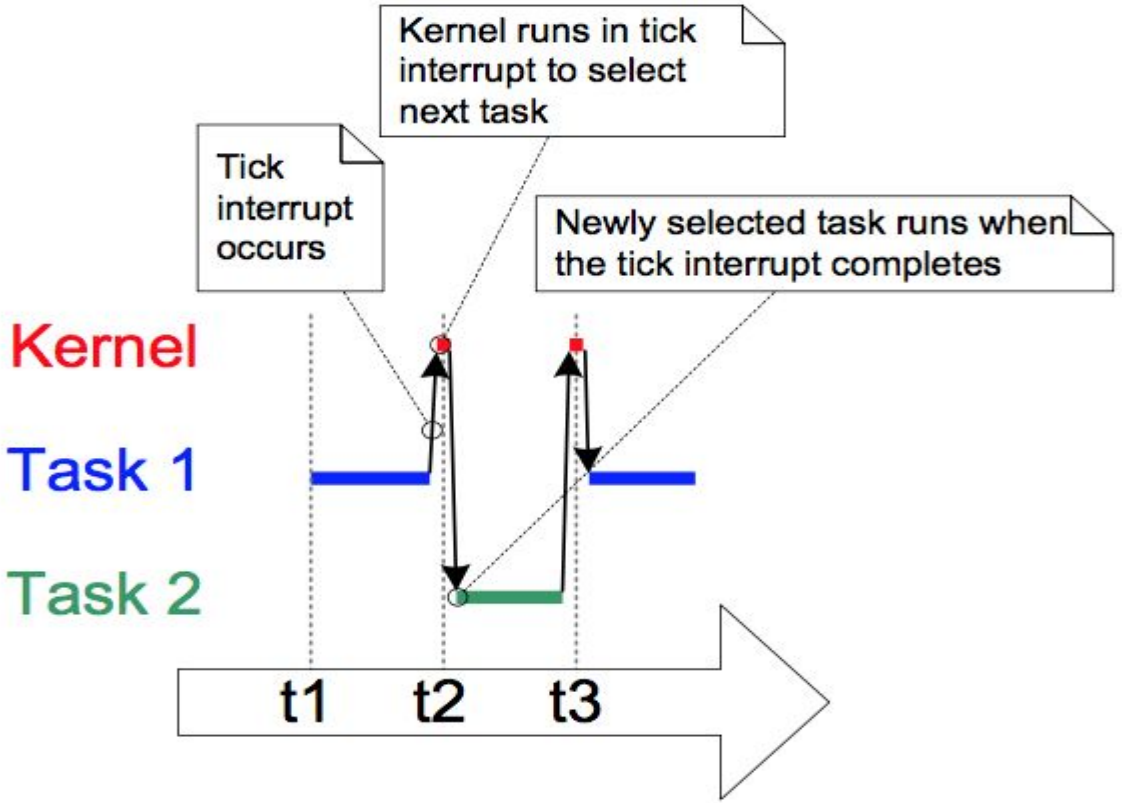
uxPriority

the lowest priority:

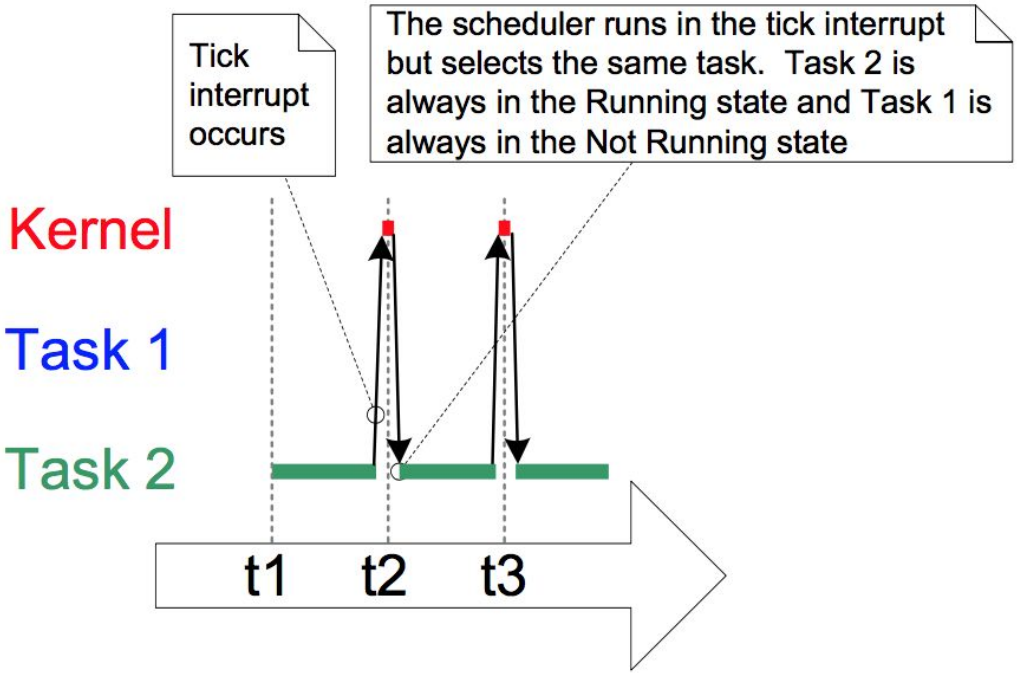
0

the highest priority:

configMAX_PRIORITIES – 1



Task Priorities



Task 1 priority less than Task 2

Not Running States

Blocked state - wait for event

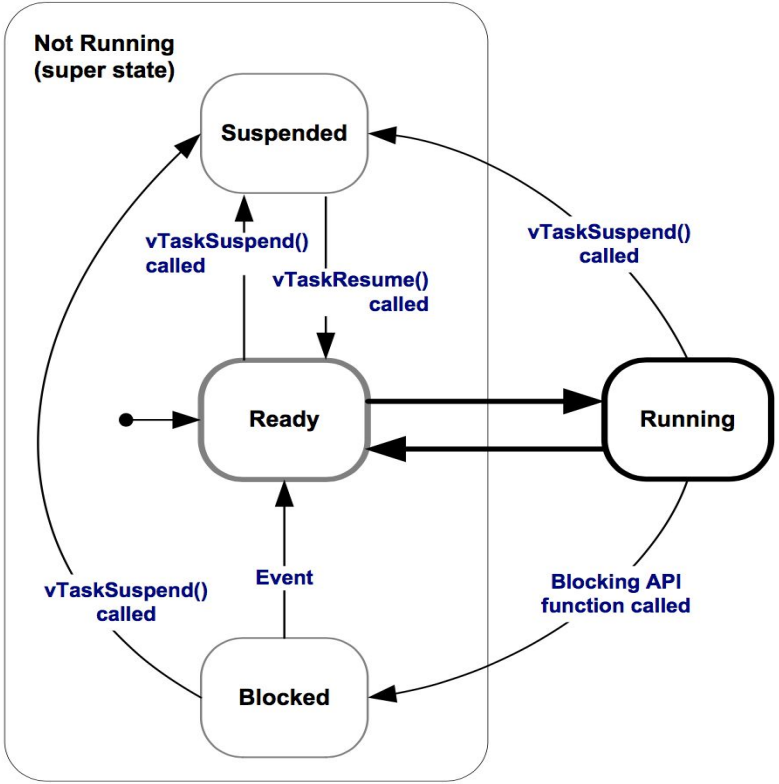
1. Temporal (time related) event
2. Synchronization event (queues, semaphores, mutexes, task notifications …)

Suspend state - entered manually with vTaskSuspend() and vTaskResume()

Ready state - ready to be executed

Not Running States

- Suspended
- Ready
- Blocked



Idle Task and Idle Hook

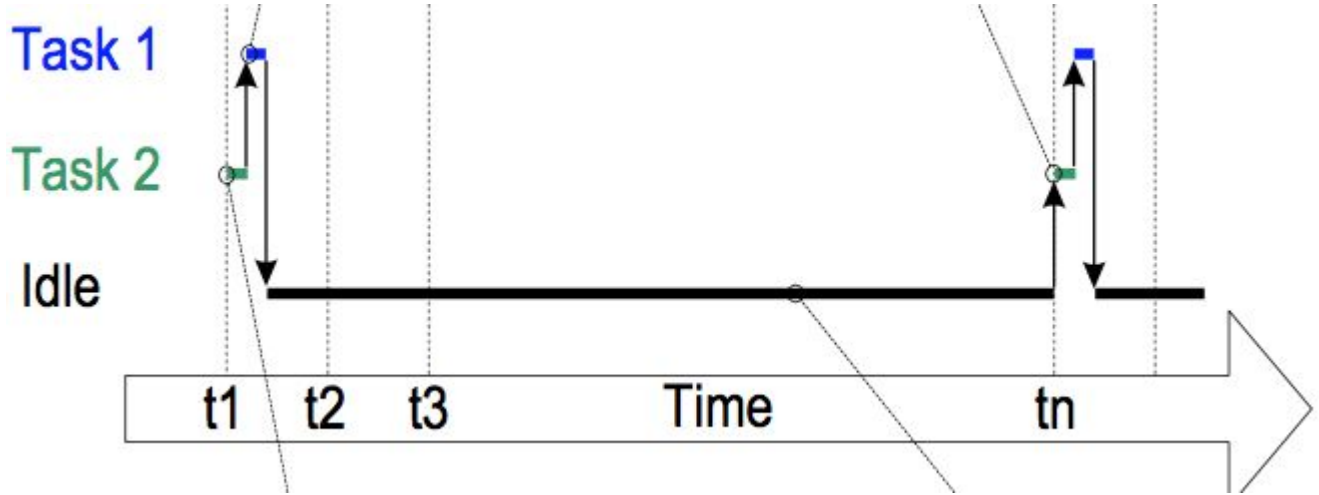
Idle task

- Always in running state
- Has lowest priority
- Responsible for cleaning up kernel resources after a task has been deleted.

Idle hook

- vApplicationIdleHook(void) - will be called by Idle task each loop iteration
- Measuring the amount of spare processing capacity.
- Placing the processor into a low power mode

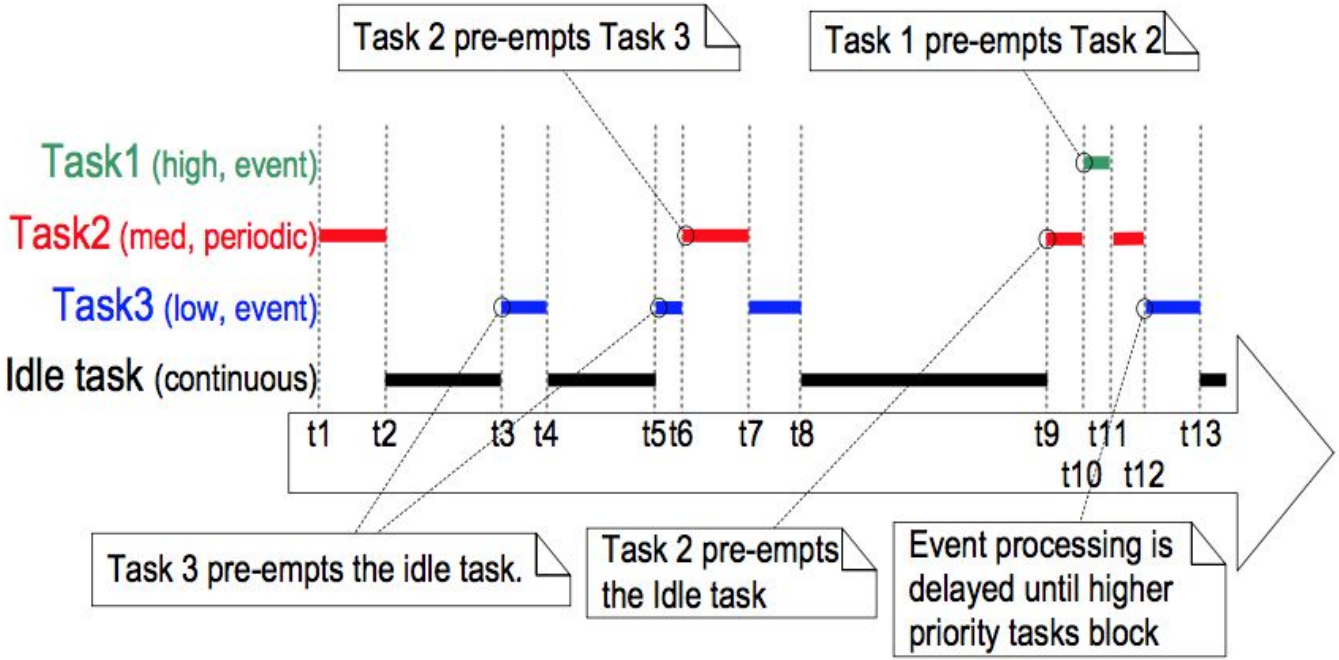
Idle Task and Idle Hook



Task 1 priority less than Task 2

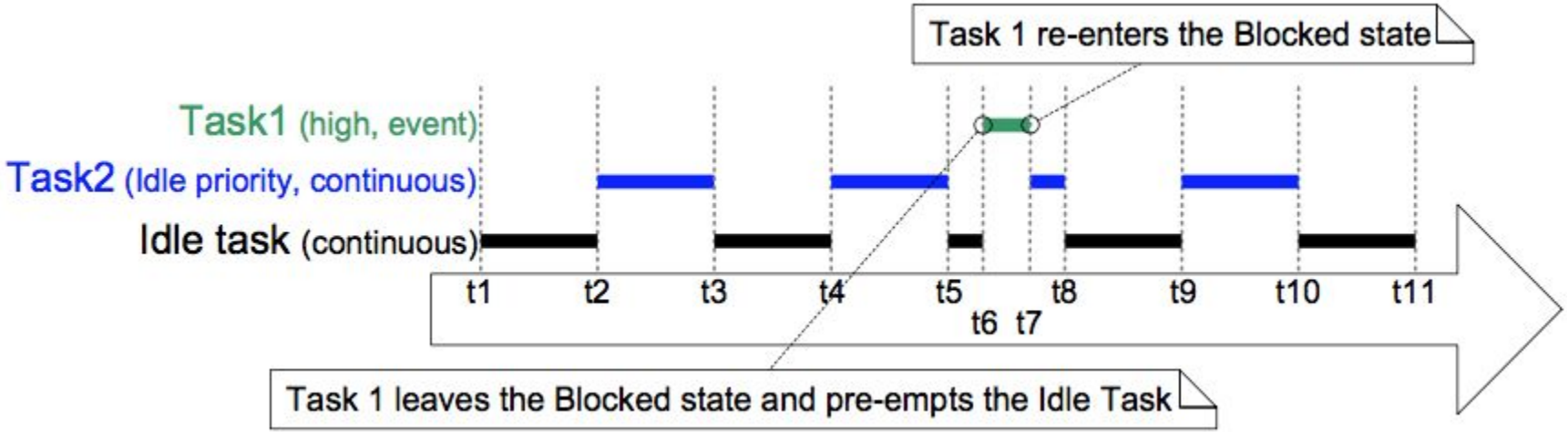
Scheduling Algorithms

configUSE_PREEMPTION = 1, configUSE_TIME_SLICING = 1



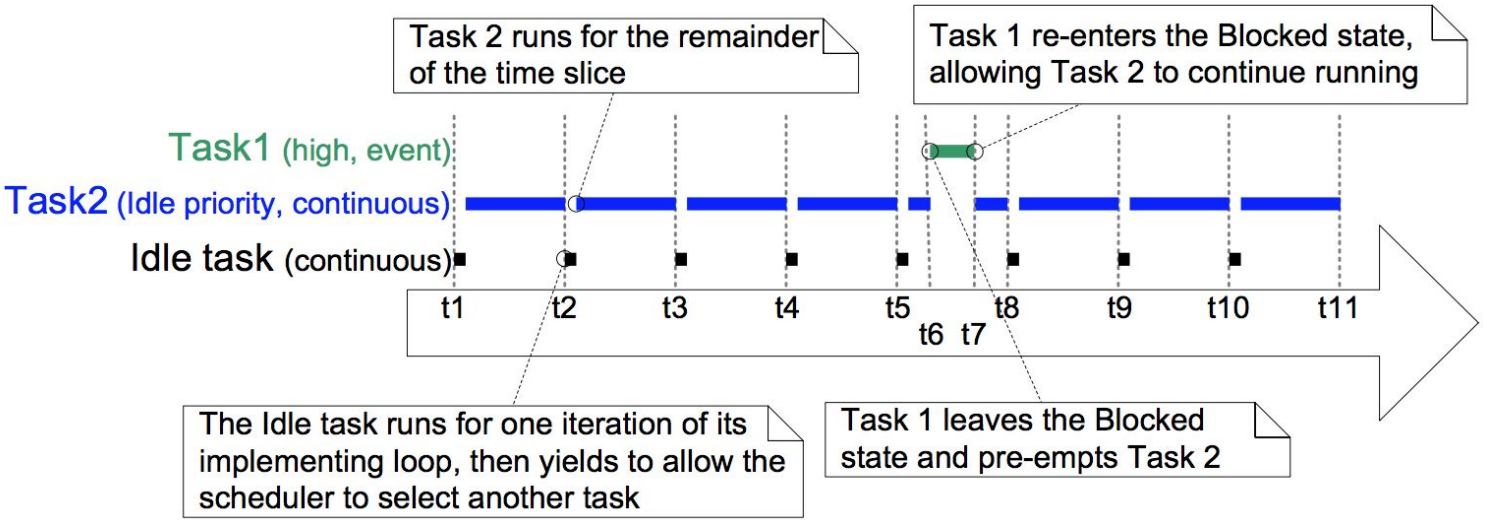
Scheduling Algorithms

configUSE_PREEMPTION = 1, configUSE_TIME_SLICING = 1



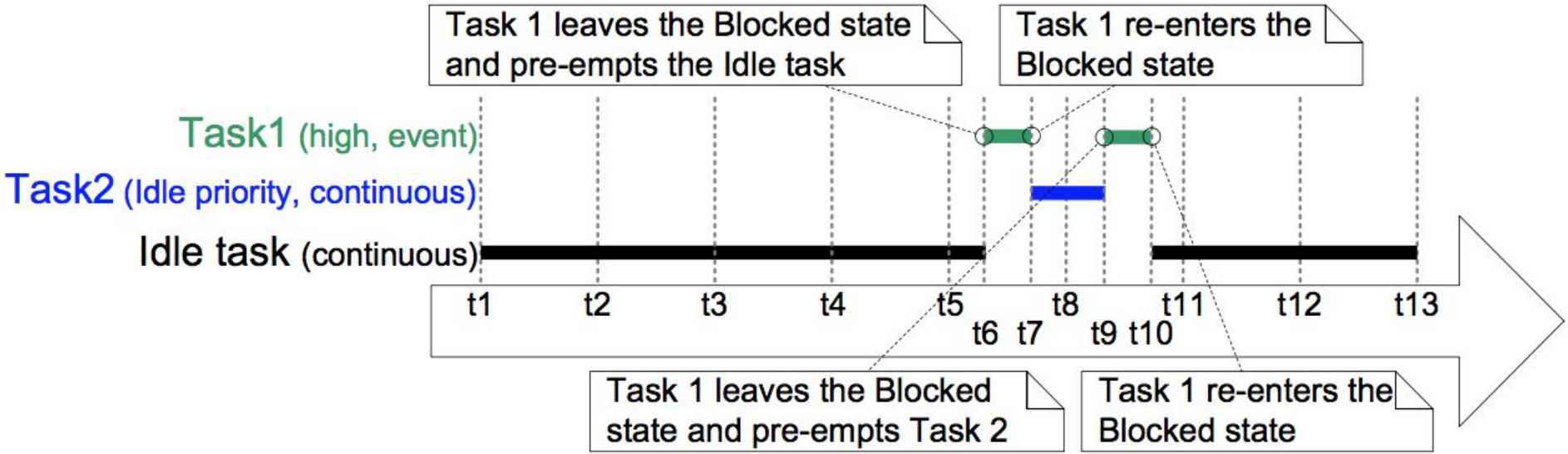
Scheduling Algorithms

configUSE_PREEMPTION = 1, configUSE_TIME_SLICING = 1, configIDLE_SHOULD_YIELD = 1



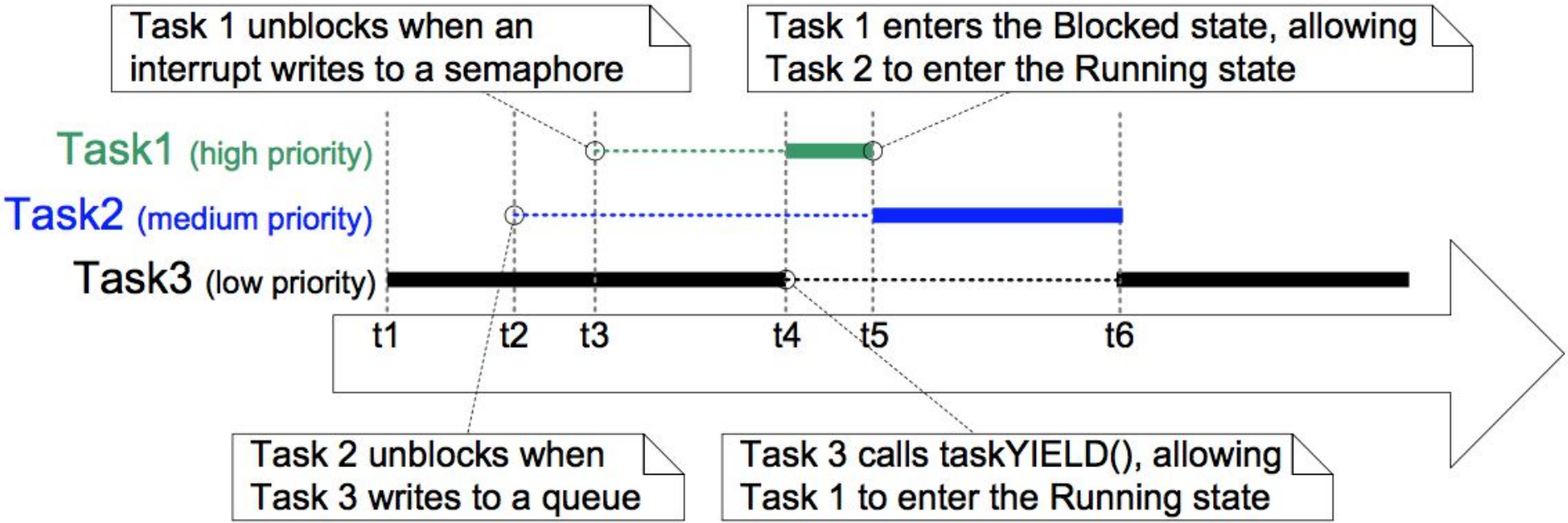
Scheduling Algorithms

configUSE_PREEMPTION = 1, configUSE_TIME_SLICING = 0



Scheduling Algorithms

configUSE_PREEMPTION = 0, configUSE_TIME_SLICING = Any value



Example

go to stm32f0_ARM directory

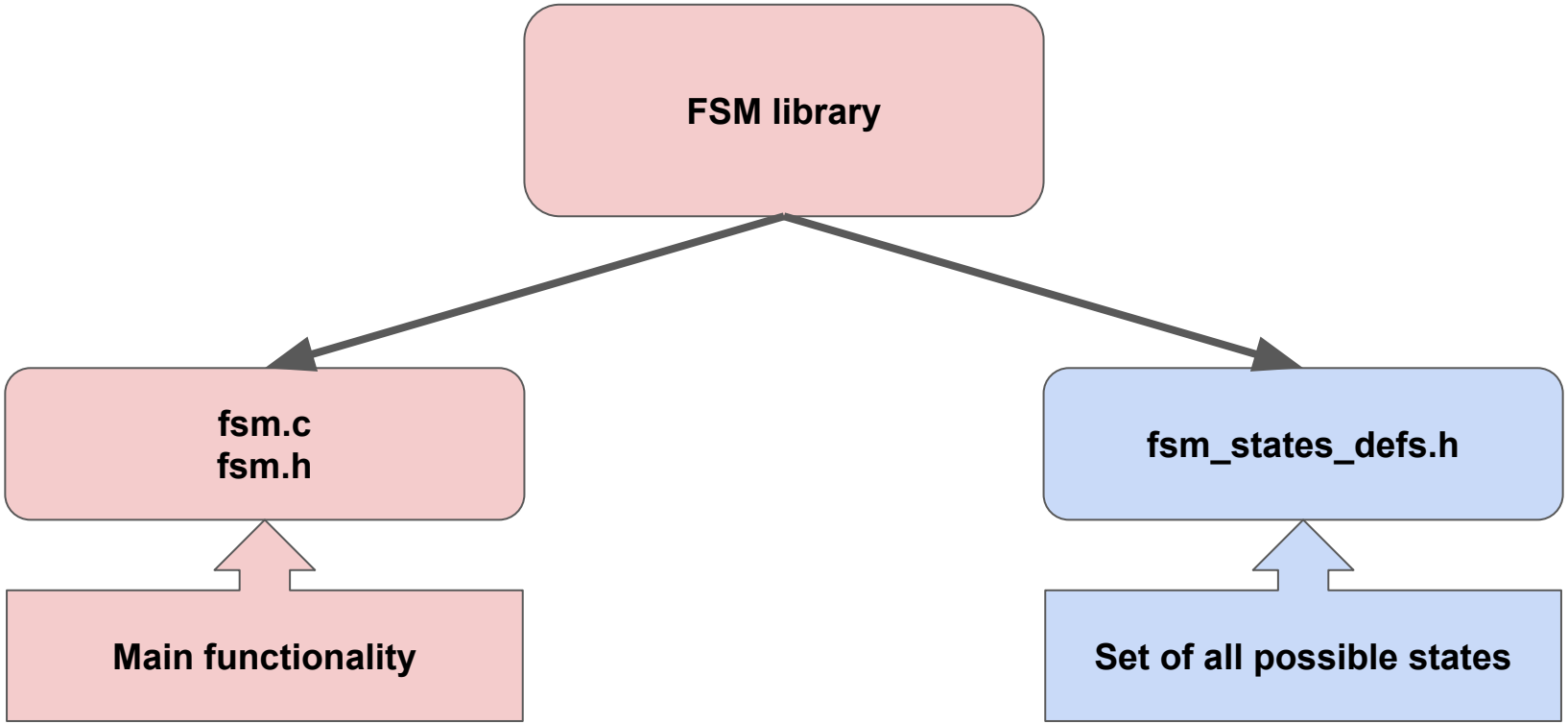
git pull

cd rtos

make

make flash

FSM. Library architecture



FSM. Example of state list in fsm_states_defs.h

```
FSM_STATE_DEF(DUMMY,                    dummy)
FSM_STATE_DEF(ERROR,                    error)
FSM_STATE_DEF(GLOBAL_INIT,              global_init)
FSM_STATE_DEF(TERM_MAIN,                term_main)
```

FSM. fsm.h. States identifiers generator

```
/*
 * Generate enum of states available in
 * fsm_defs.h
 * Note: All commands start with 'FSM_' prefix
 */
#define FSM_STATE_DEF(state_aliase, state_handler) \
        FSM_##state_aliase,
enum {
        LOWER_BOUND_CASE,
        #include "fsm_states_defs.h"
        UPPER_BOUND_CASE
};
#undef FSM_STATE_DEF
```

FSM. fsm.h. Unrolling enum defines

```
enum {
        LOWER_BOUND_CASE = 0,
        FSM_DUMMY = 1,
        FSM_ERROR = 2,
        FSM_GLOBAL_INIT = 3,
        FSM_TERM_MAIN = 4,
        UPPER_BOUND_CASE = 5
};
```

FSM. fsm.h. States handlers generator

```
/*
 * Array of cases for finite state machine
 */
#define FSM_STATE_DEF(state_aliase, state_handler) \
        [FSM_##state_aliase] = fsm_##state_handler,
void (* const fsm_states_handlers[])(void *) = {
        #include "fsm_states_defs.h"
};
#undef FSM_STATE_DEF
```

FSM. Unrolling states generator

```
void (* const fsm_states_handlers[])(void *) = {
        [FSM_DUMMY] = fsm_dummy,
        [FSM_ERROR] = fsm_error,
        [FSM_GLOBAL_INIT] = fsm_global_init,
        [TERM_MAIN] = fsm_term_main
};
```

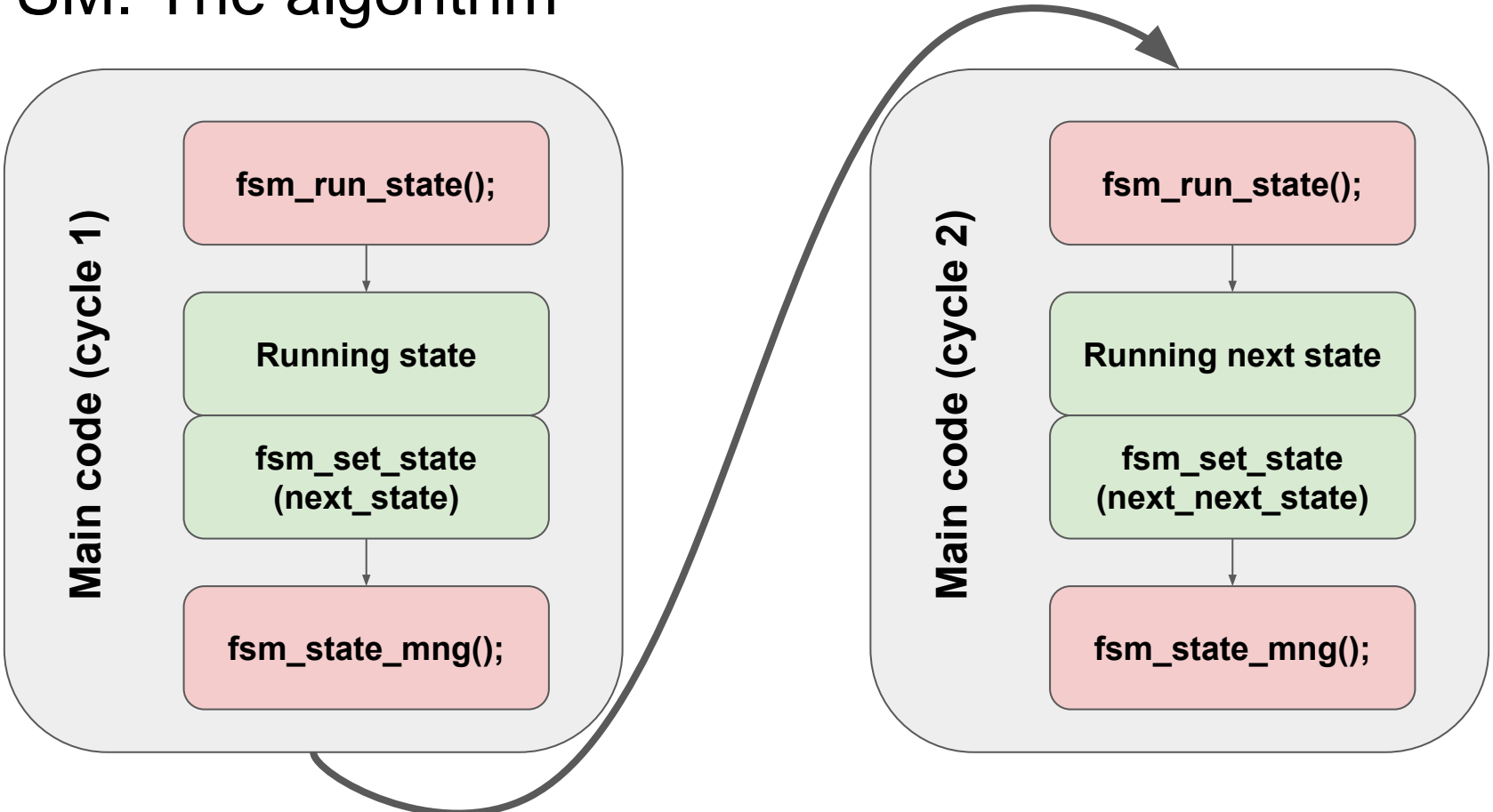
FSM. fsm.h. States prototypes generator

```
/*
 * declare list of handlers prototypes
 * note: all handler function names start
 * with 'fsm_' prefix
 */
#define fsm_state_def(state_aliase, state_handler) \
void fsm_##state_handler(void *args);
#include "fsm_states_defs.h"
#undef fsm_state_def
```

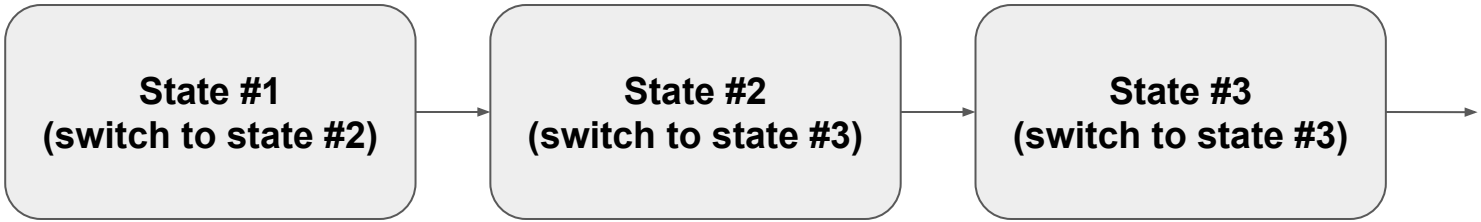
FSM. Unrolling prototypes generator

```
void fsm_dummy(void *args);
void fsm_error(void *args);
void fsm_global_init(void *args);
void fsm_term_main(void *args);
```

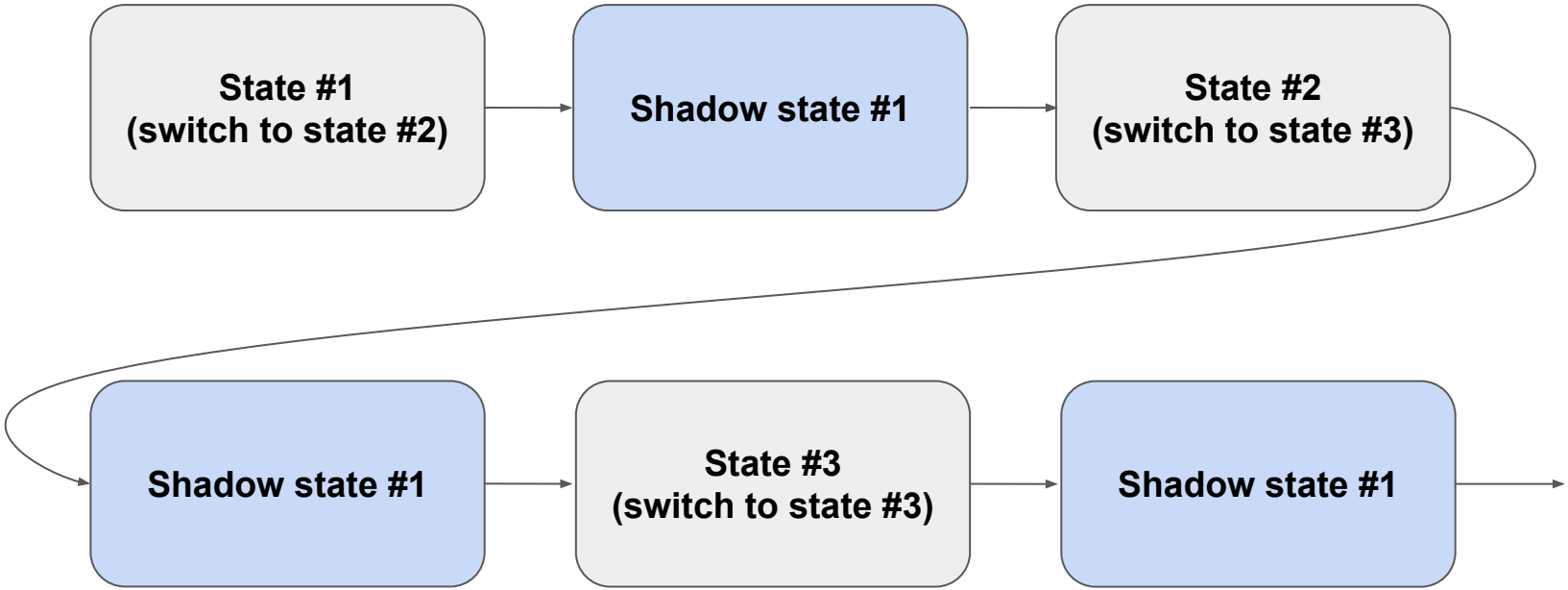
FSM. The algorithm



FSM. The calling sequence

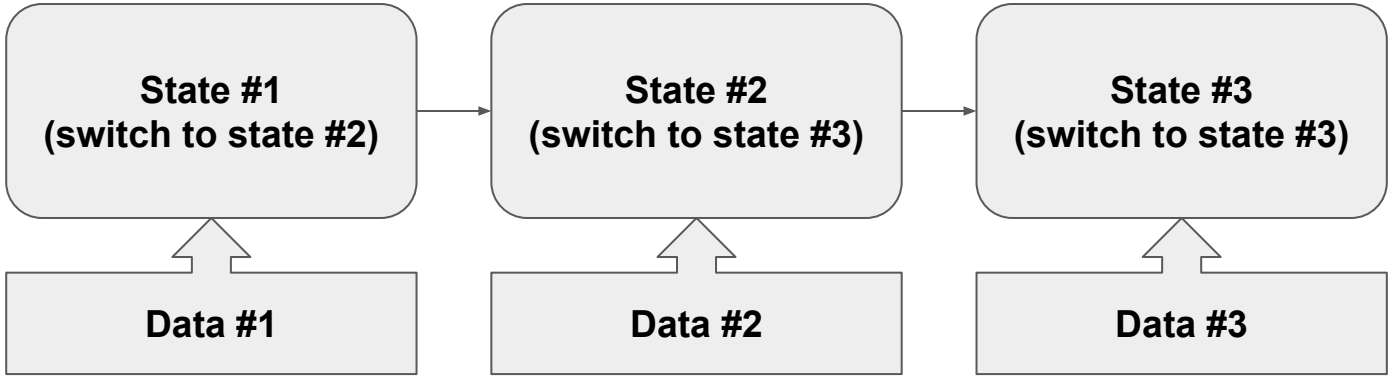


FSM. Shadow states. Example with error_catching



fsm_add_shadow_state()

FSM. Data bypassing. Terminal example



fsm_add_set_data(state, state_data)