Интернет вещей

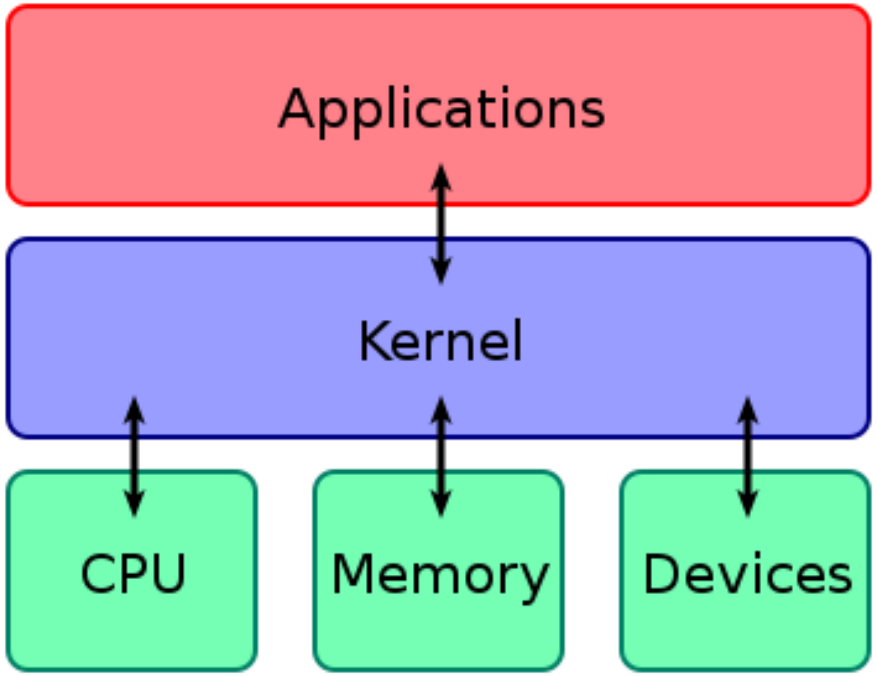
Упреждающие операционные системы

Операционная система

система управления ресурсами вычислительной системы

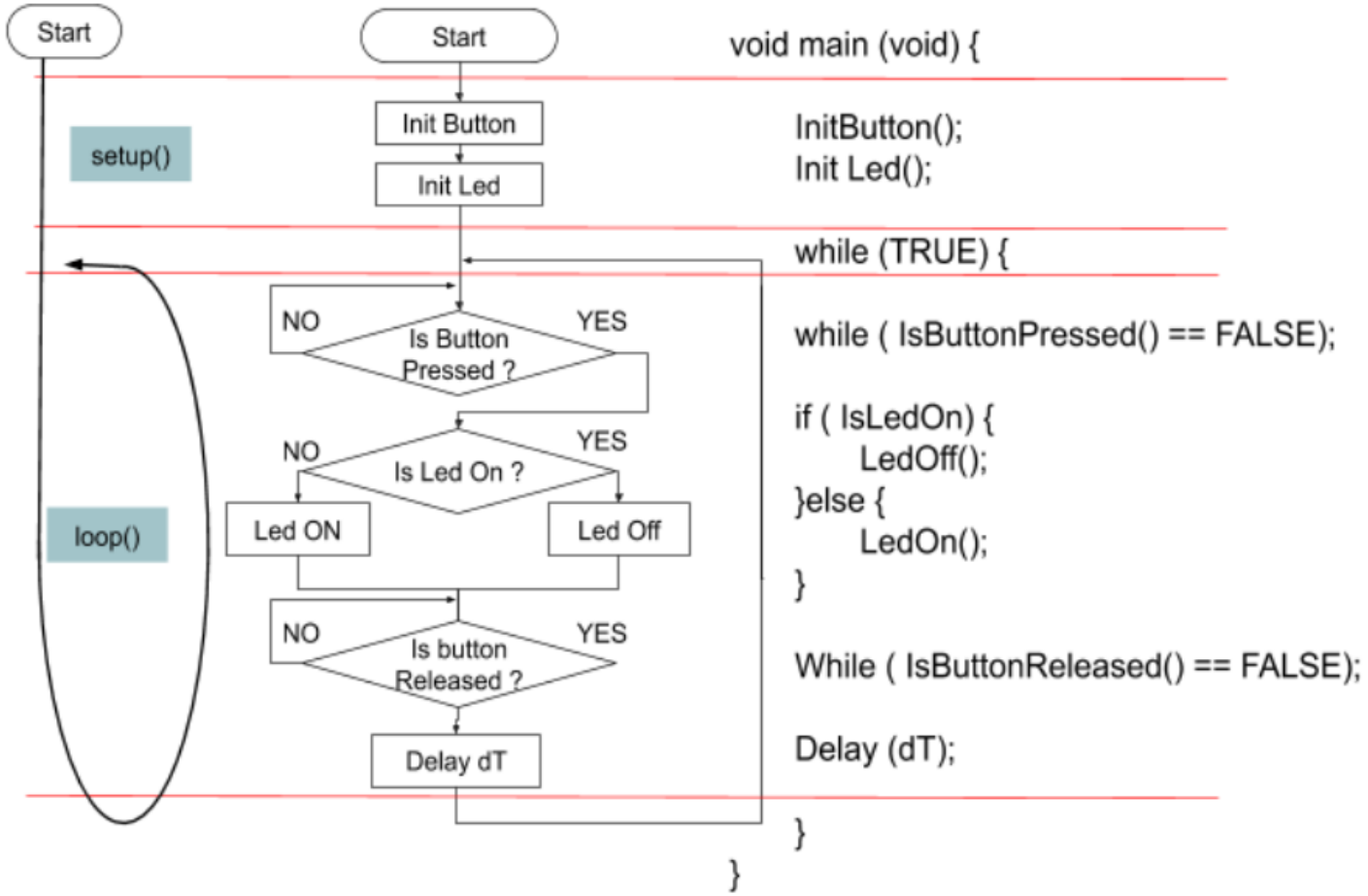
Управляемые ресурсы:

- Память
- Переферия
- Время обработки

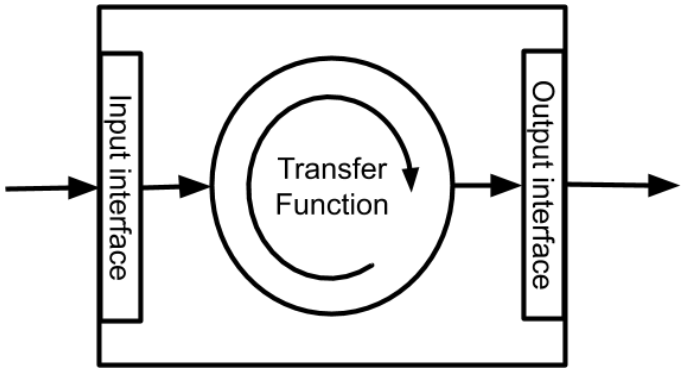


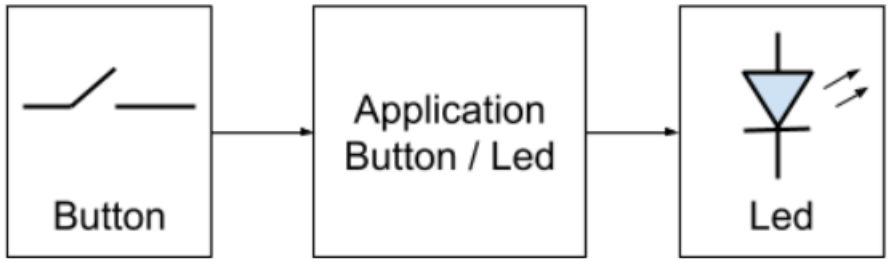
Single-process – Infinite loop

Классическое приложение– Buton / Led



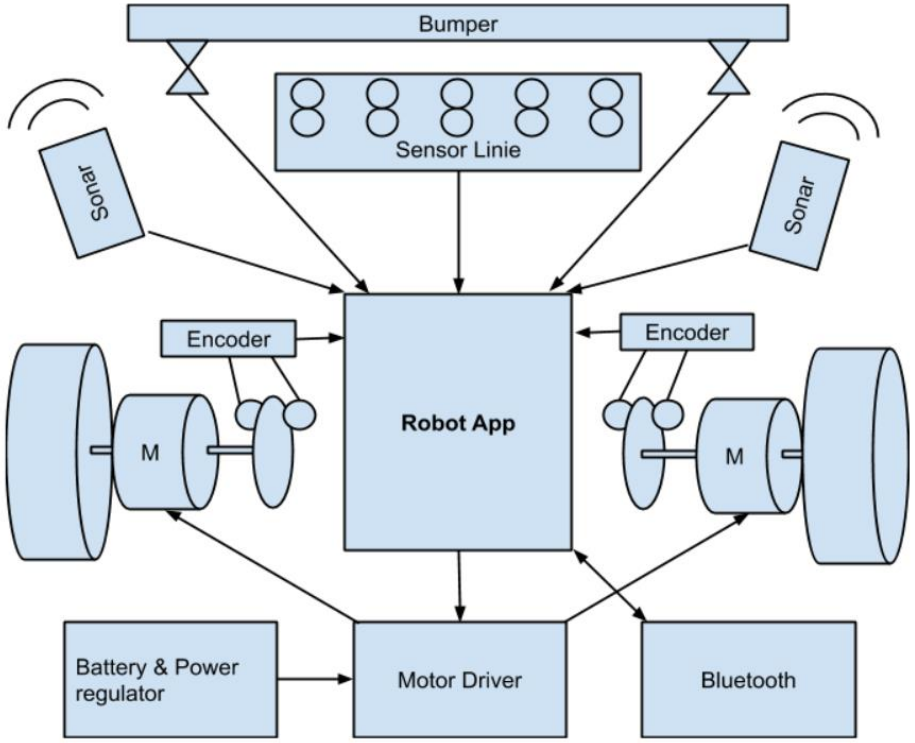

```
void main (void) {

    InitButton();
    Init Led();

    while (TRUE) {

    while ( IsButtonPressed() == FALSE);

    if ( IsLedOn) {
        LedOff();
    }else {
        LedOn();
    }

    While ( IsButtonReleased() == FALSE);

    Delay (dT);

    }
}
```



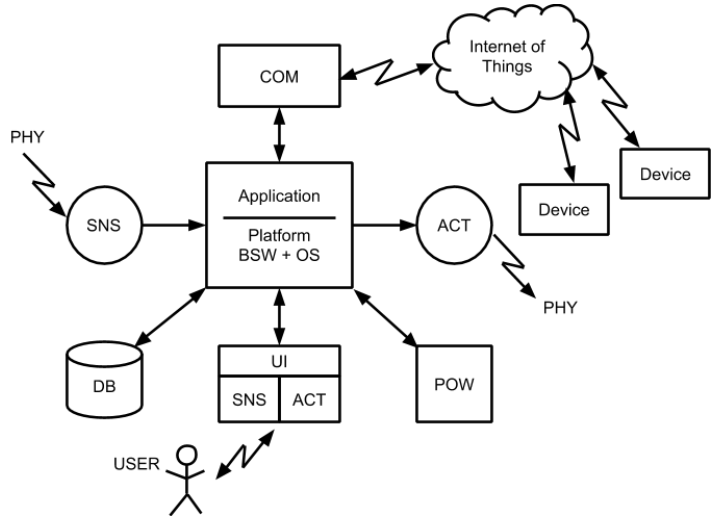


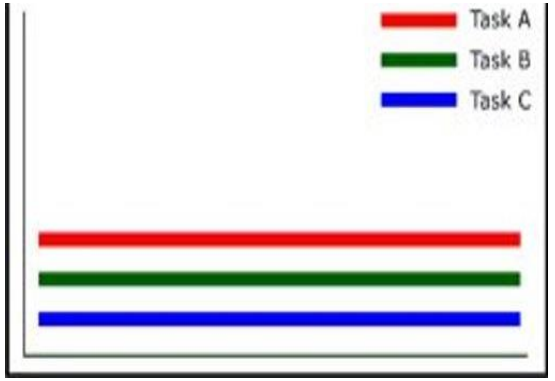
Multi tasking - Problem

сосуществование нескольких функций одновременно

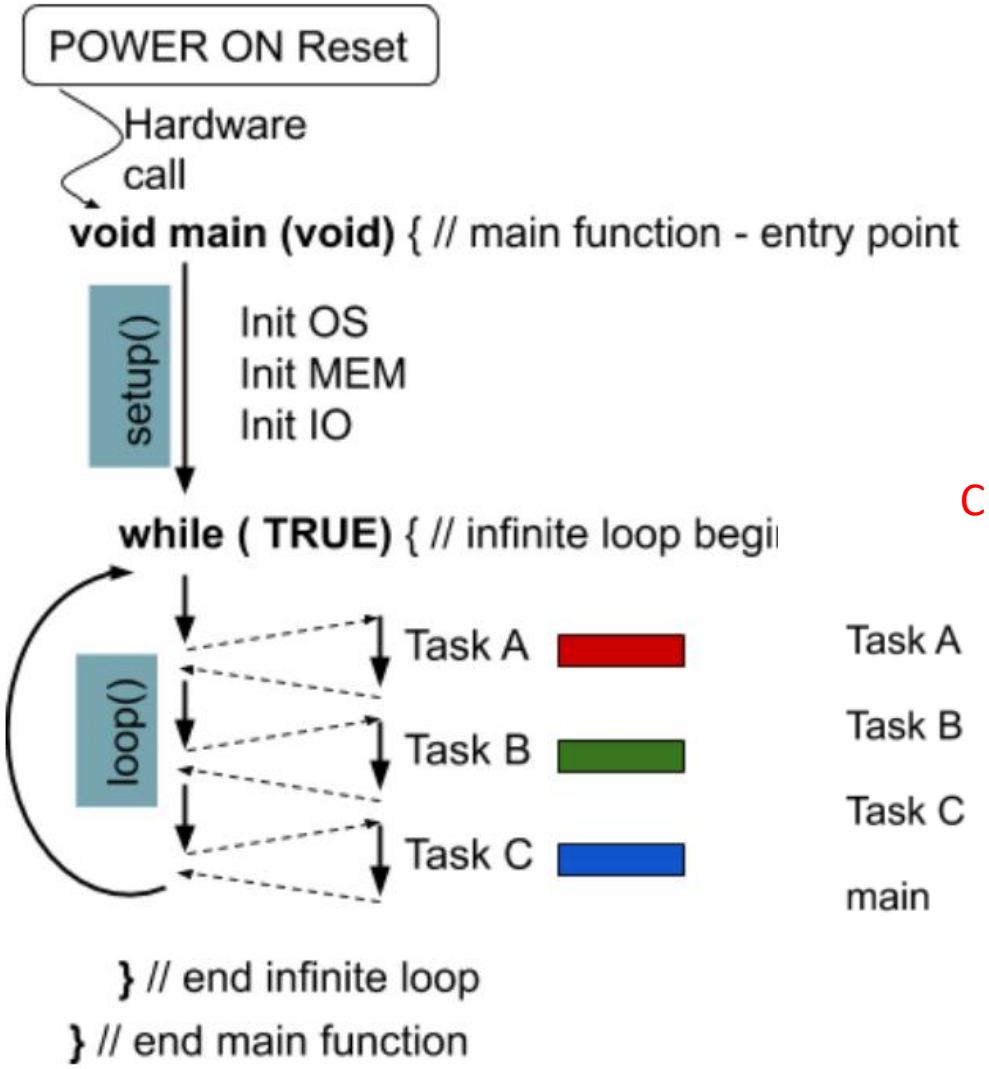


- Взаимодействие с пользователем
- Сенсор
- Привод
- Управление
- Коммуникация

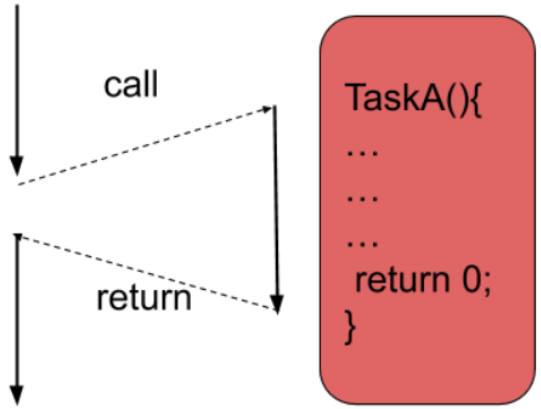




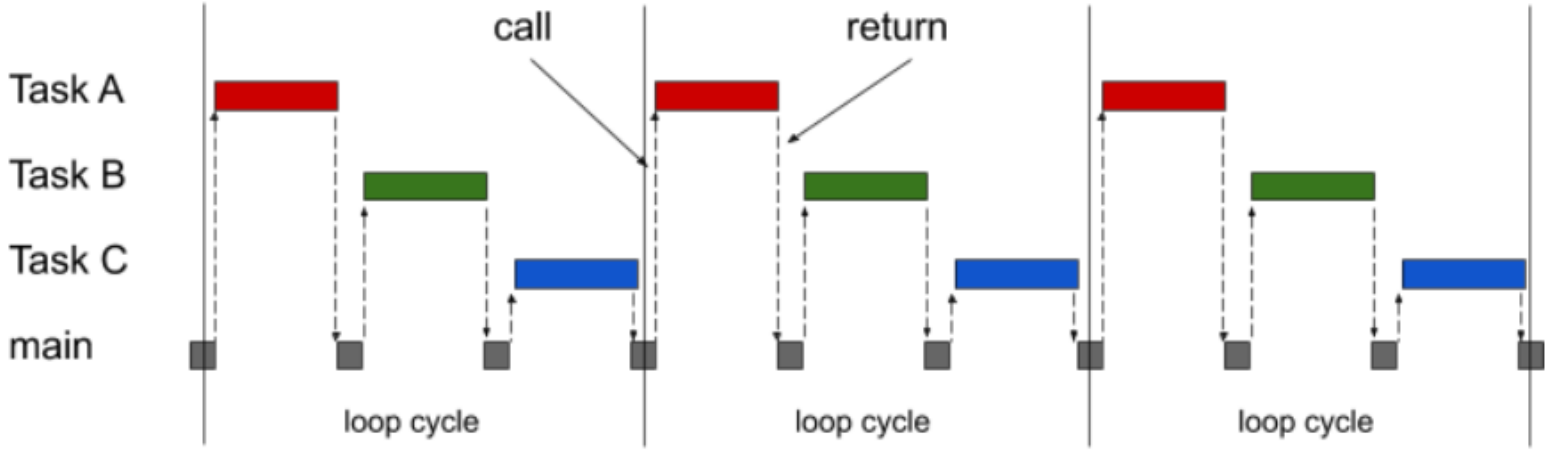
Multi-tasking - Single process



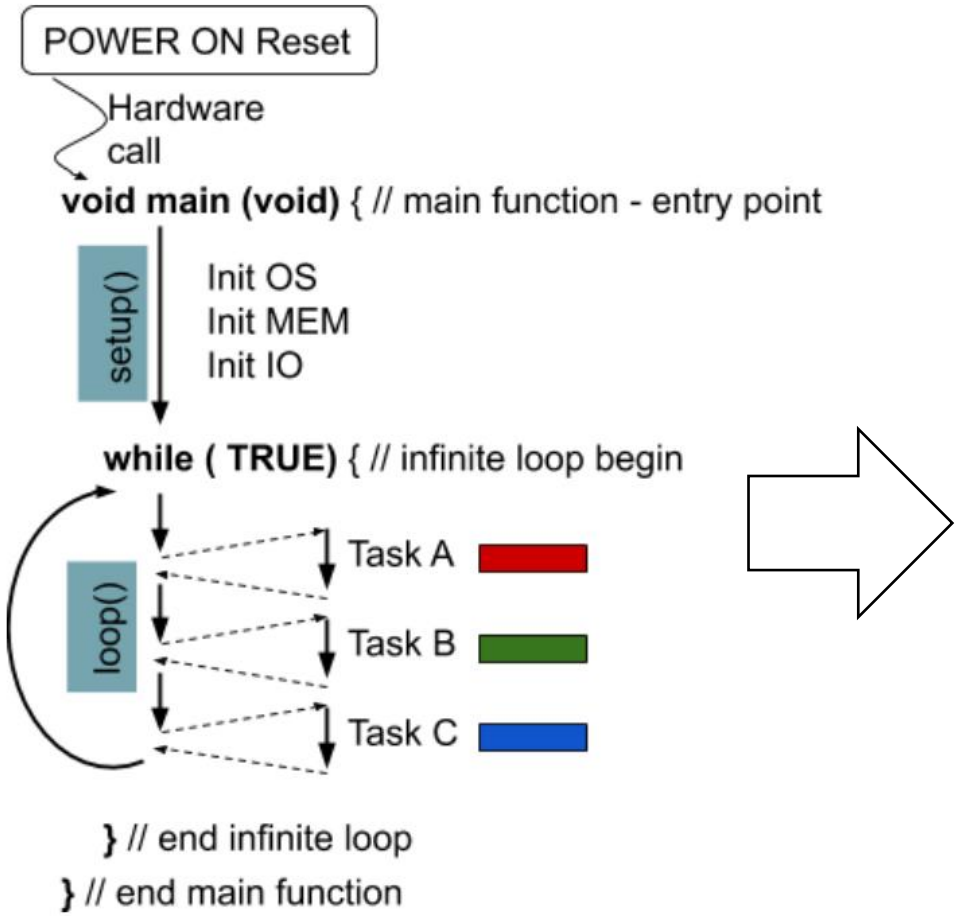
Task switch context

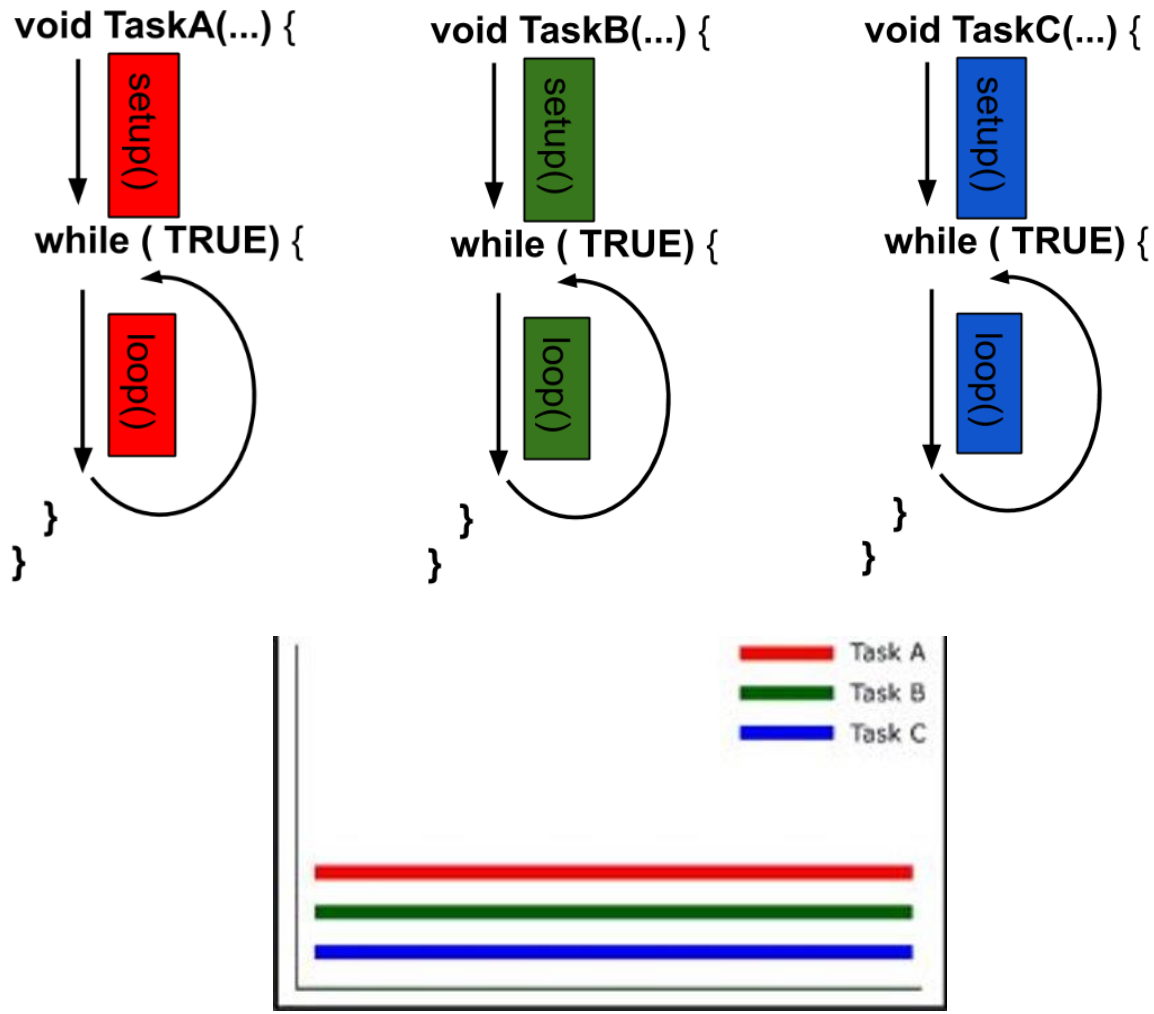


CPU Load – 100%

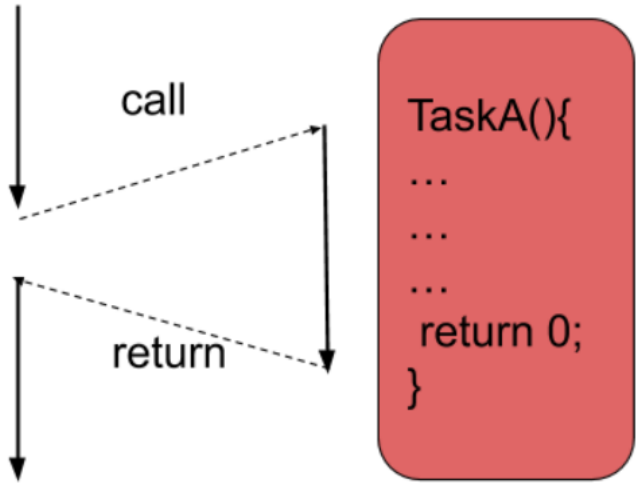


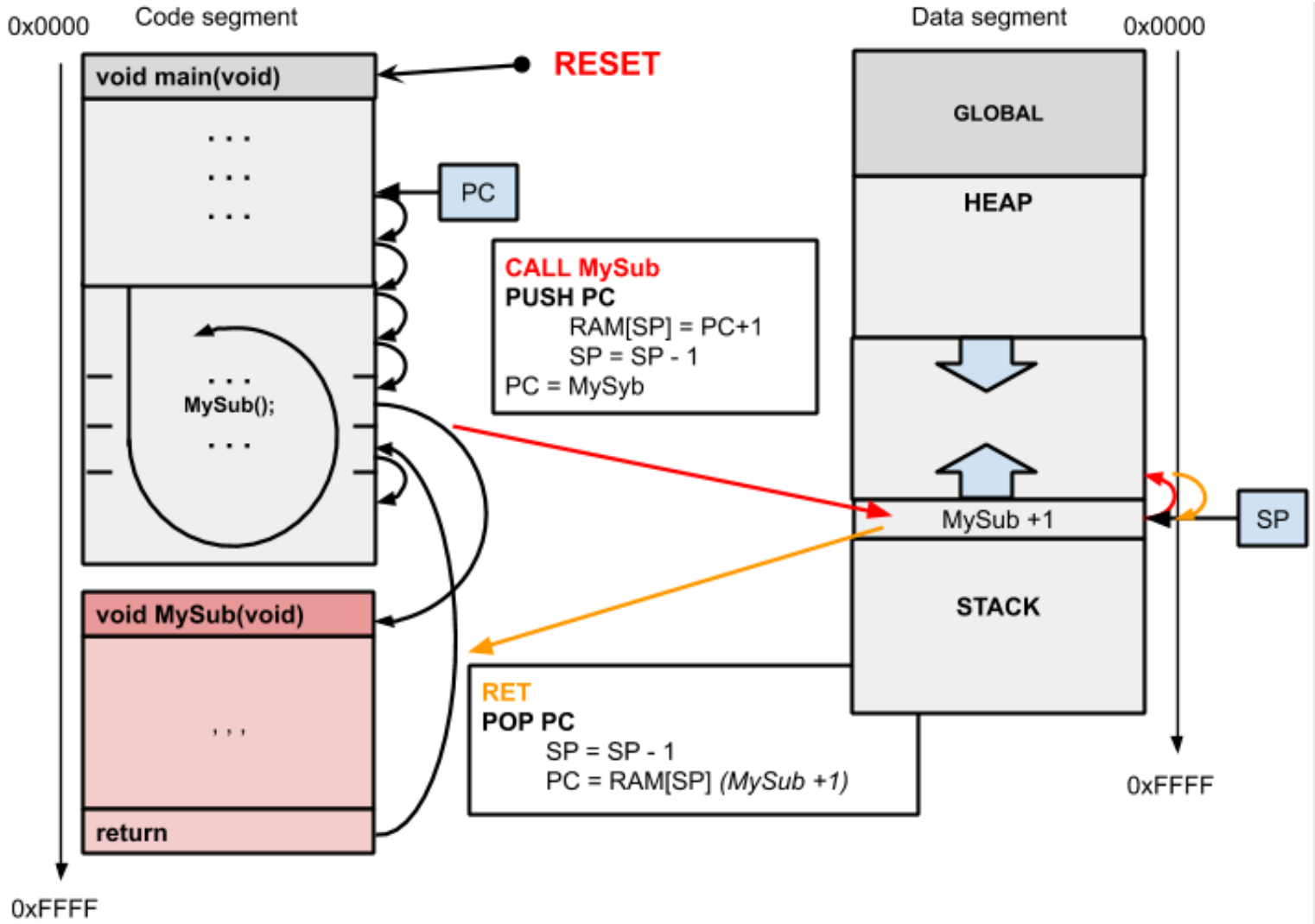
Multi-Tasking Multi-process





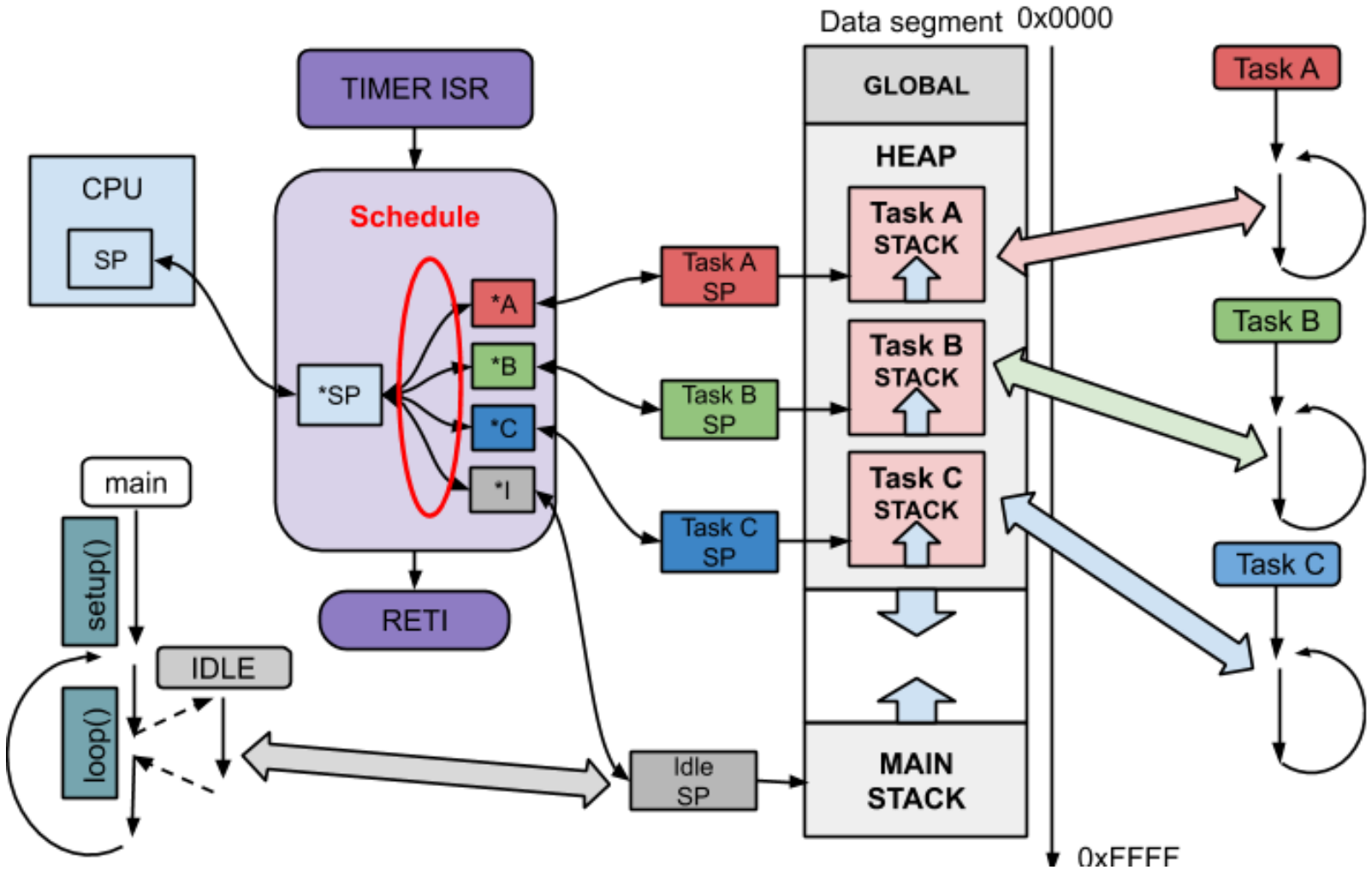
Stack – Subroutine call





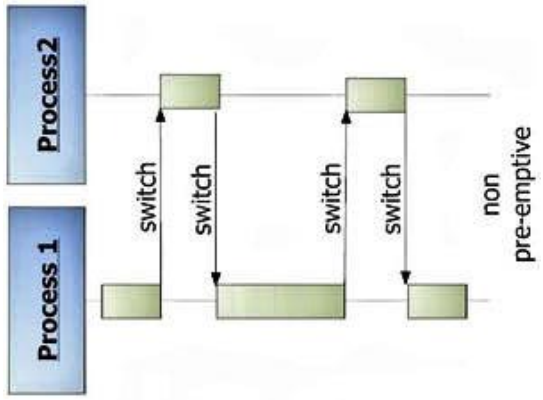
Multi-tasking – Switch Context – Pre-emption

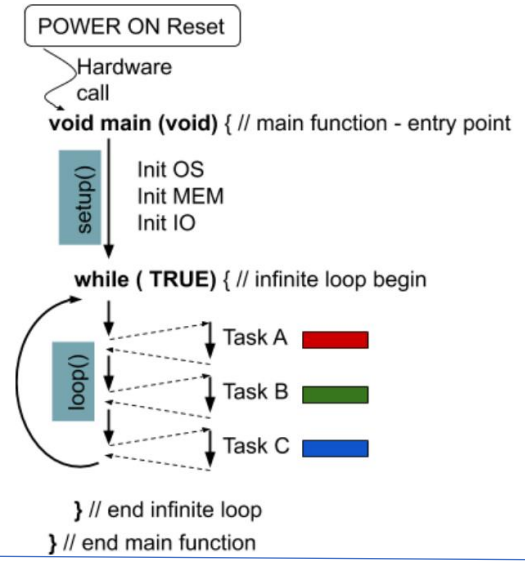
1. Store current state GPR
2. Arbiter Tasks
3. Change Stack Pointer
4. Restore new state GPR

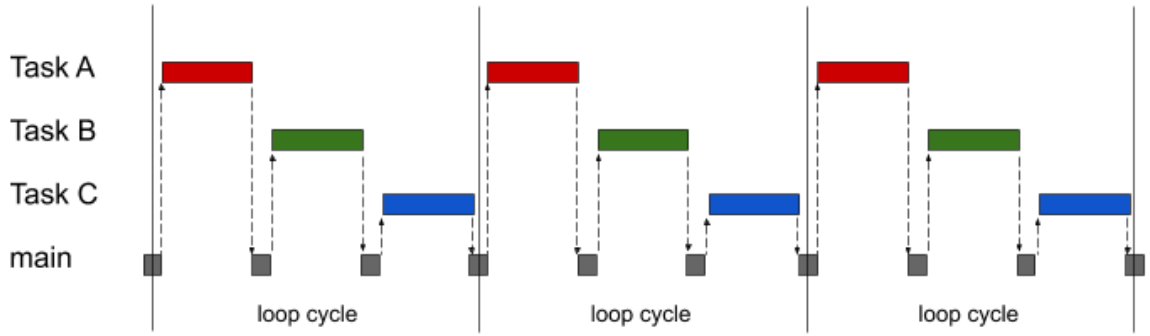


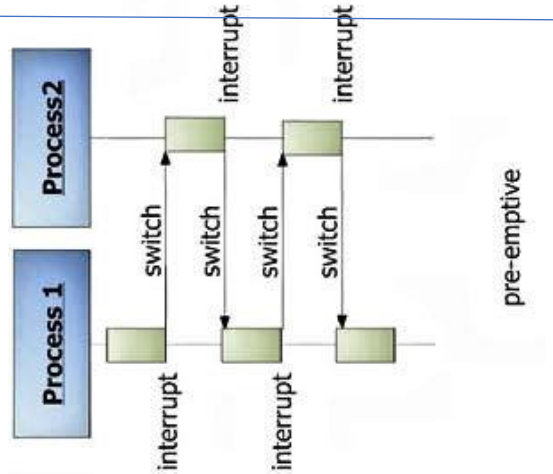
*GPR – general purpose registers (CPU)

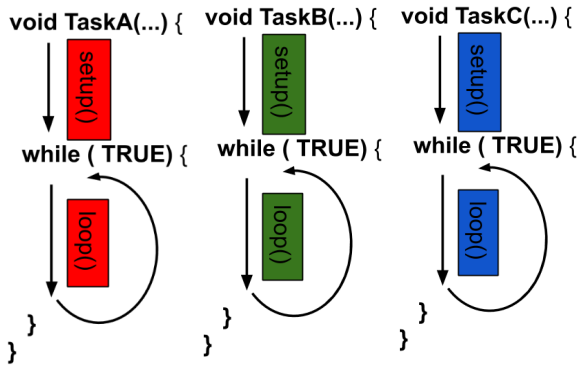
Multi-tasking : pre-emptive vs non pre-emptive

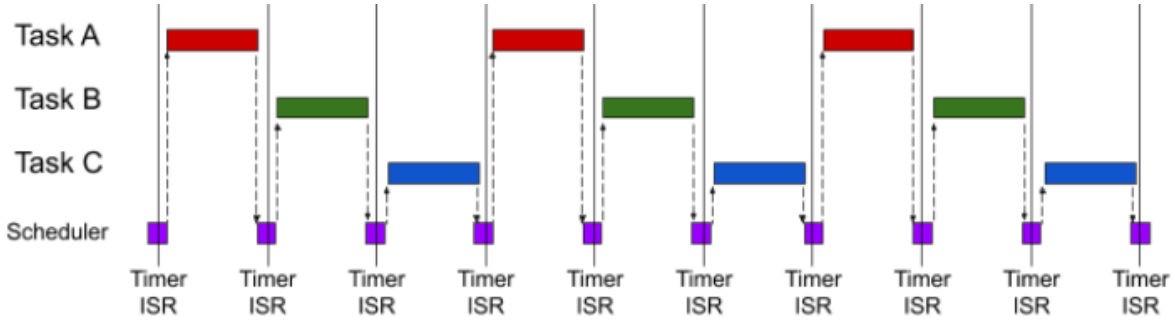




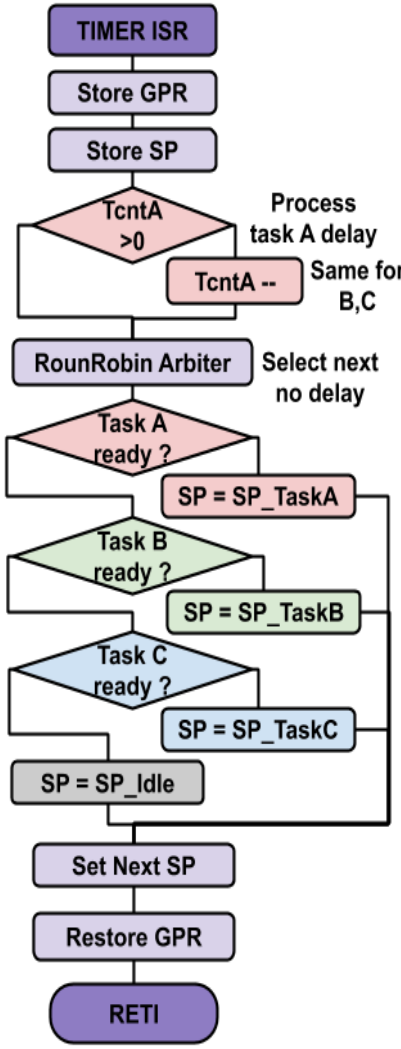




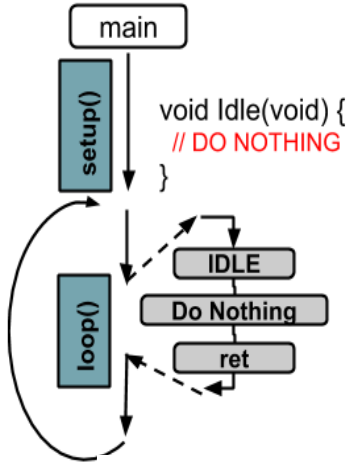


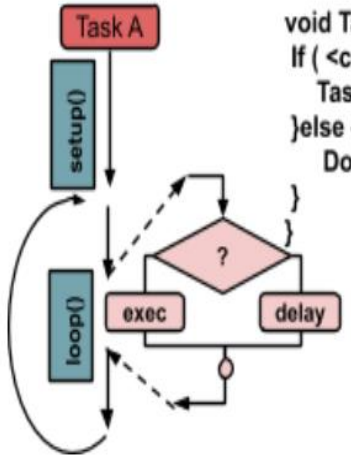


Multi-tasking : Task Delay

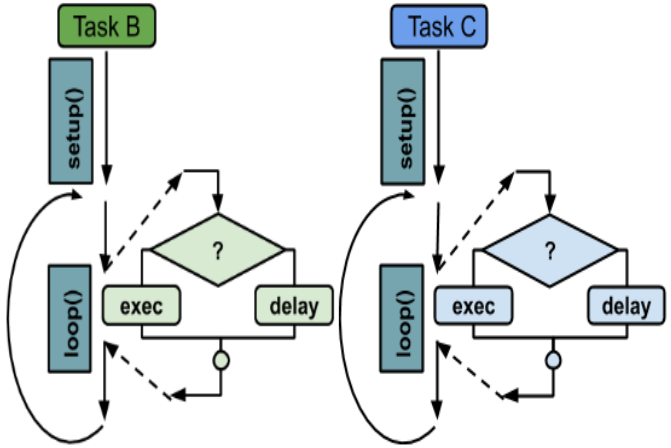


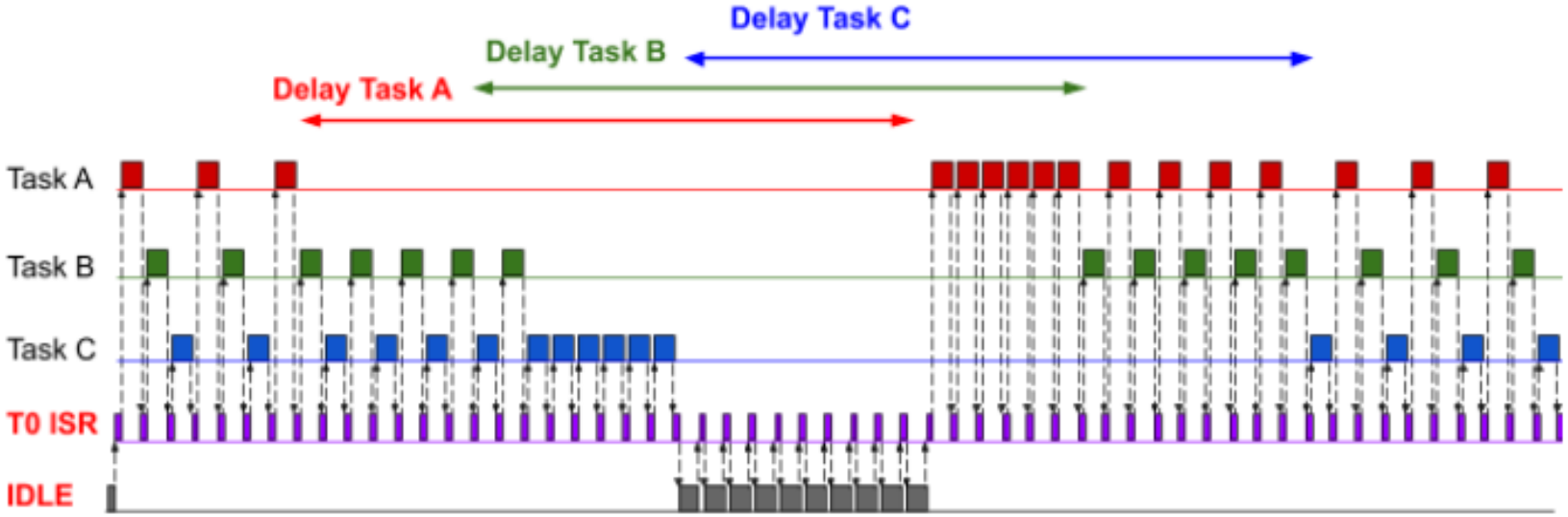

```
ISR (T0_OVF_vect){

  OS_StoreGPR();

  OS_SToreSP( )

  if(TcntA > 0) TcntA--;
  if(TcntB > 0) TcntB--;
  if(TcntC > 0) TcntC--;

  TaskID = OS_Arbiter();

  if(TaskID == TASK_A)
       nextSP = SP_TaskA;

  else if(TaskID == TASK_B)
       nextSP = SP_TaskB;

  else if(TaskID == TASK_C)
       nextSP = SP_TaskC;

  else
       nextSP = SP_Idle;

  OS_SetSP(nextSP )

  OS_RestoreGPR();

}
```






```
void TaskA(){
 If ( <condition A> )
    TaskDelay(T_DELAY)
 }else {
    DoSomething()
 }
```





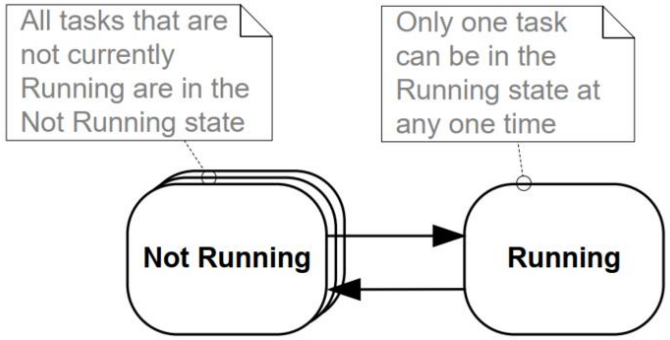
FreeRTOS – Task

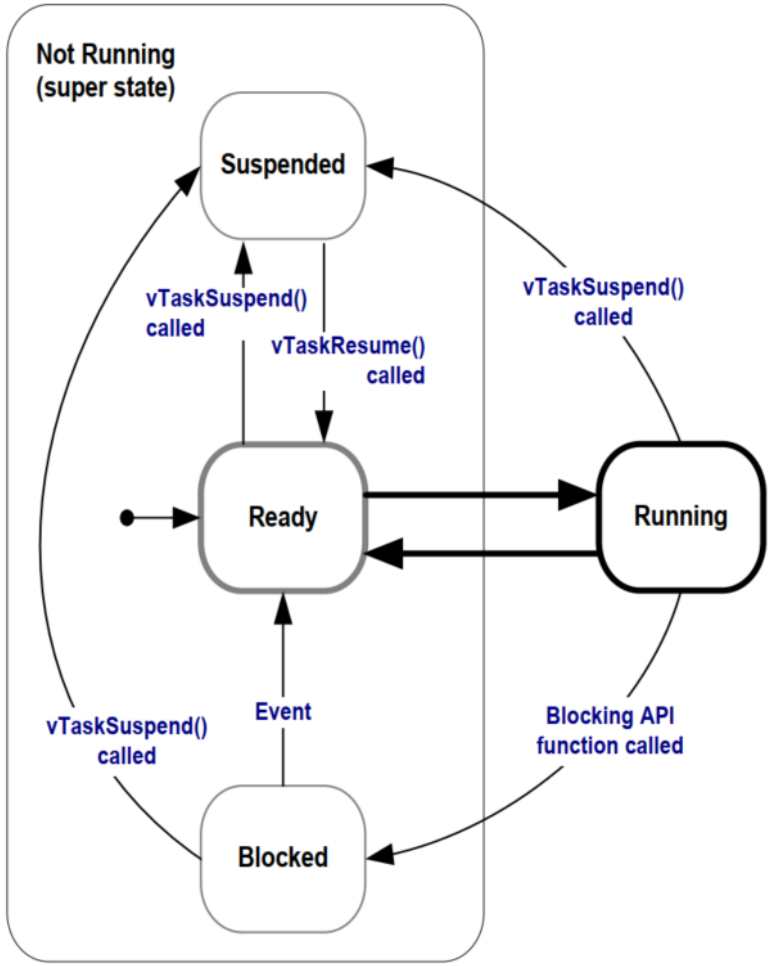
```
void vTask1( void *pvParameters )
{
const char *pcTaskName = "Task 1 is running\r\n";
volatile uint32_t ul; /* volatile to ensure ul is not optimized away. */

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /* This loop is just a very crude delay implementation.  There is
            nothing to do in here.  Later examples will replace this crude
            loop with a proper delay/sleep function. */
        }
    }
}
```

```
 void vTask2( void *pvParameters )
 {
 const char *pcTaskName = "Task 2 is running\r\n";
 volatile uint32_t ul; /* volatile to ensure ul is not optimized away. */

     /* As per most tasks, this task is implemented in an infinite loop. */
     for( ;; )
     {
         /* Print out the name of this task. */
         vPrintString( pcTaskName );

         /* Delay for a period. */
         for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
         {
             /* This loop is just a very crude delay implementation.  There is
             nothing to do in here.  Later examples will replace this crude
             loop with a proper delay/sleep function. */
         }
     }
 }
```

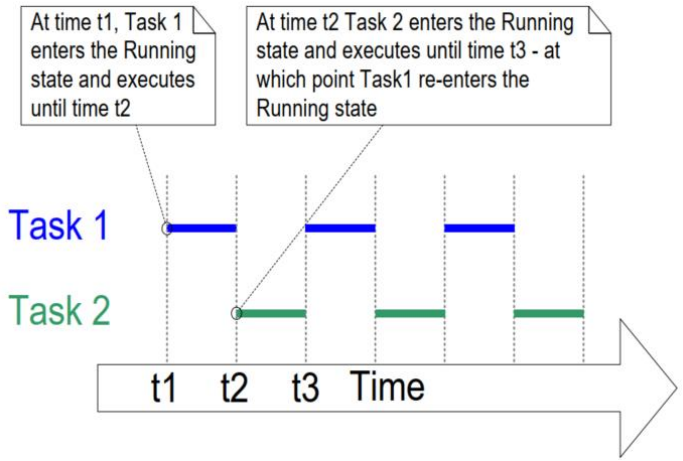
```
int main( void )
{
    /* Create one of the two tasks.  Note that a real application should check
    the return value of the xTaskCreate() call to ensure the task was created
    successfully. */
    xTaskCreate(    vTask1,  /* Pointer to the function that implements the task. */
                    "Task 1",/* Text name for the task.  This is to facilitate
                             debugging only. */
                    1000,    /* Stack depth - small microcontrollers will use much
                             less stack than this. */
                    NULL,    /* This example does not use the task parameter. */
                    1,       /* This task will run at priority 1. */
                    NULL );  /* This example does not use the task handle. */

    /* Create the other task in exactly the same way and at the same priority. */
    xTaskCreate( vTask2, "Task 2", 1000, NULL, 1, NULL );

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* If all is well then main() will never reach here as the scheduler will
    now be running the tasks.  If main() does reach here then it is likely that
    there was insufficient heap memory available for the idle task to be created.
    Chapter 2 provides more information on heap memory management. */
    for( ;; );
}
```

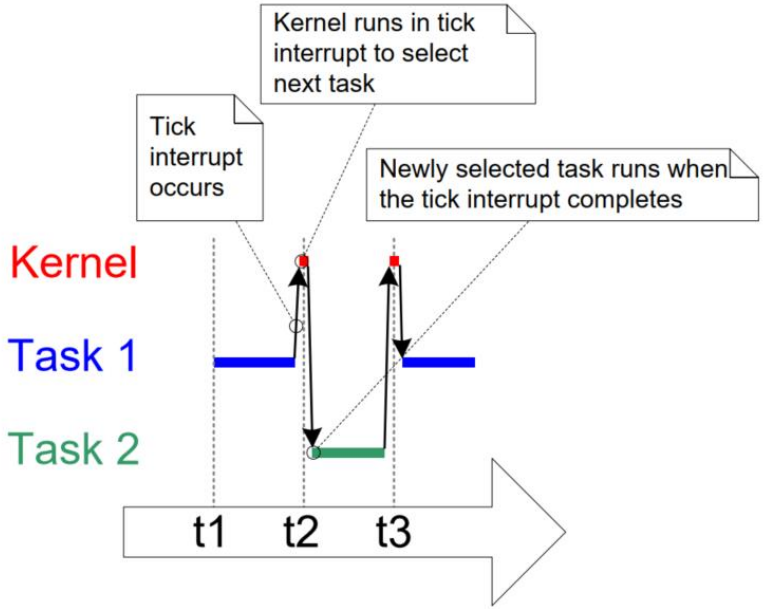
FreeRTOS – State Flow

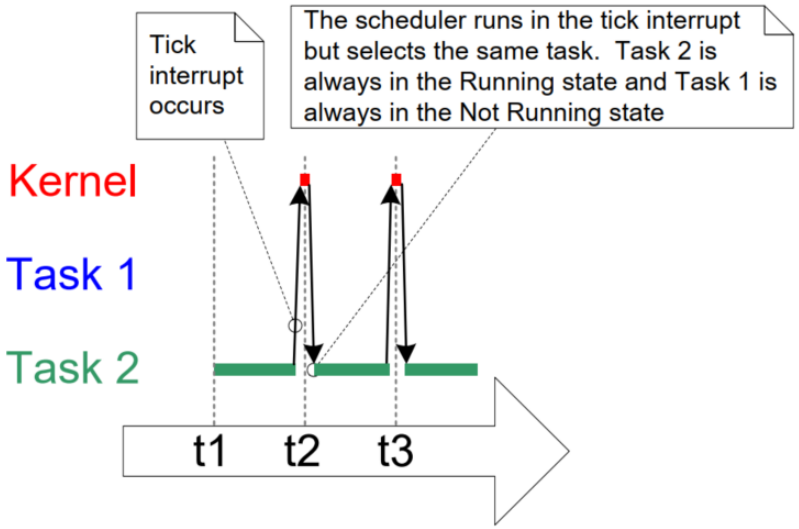




FreeRTOS – Task Running

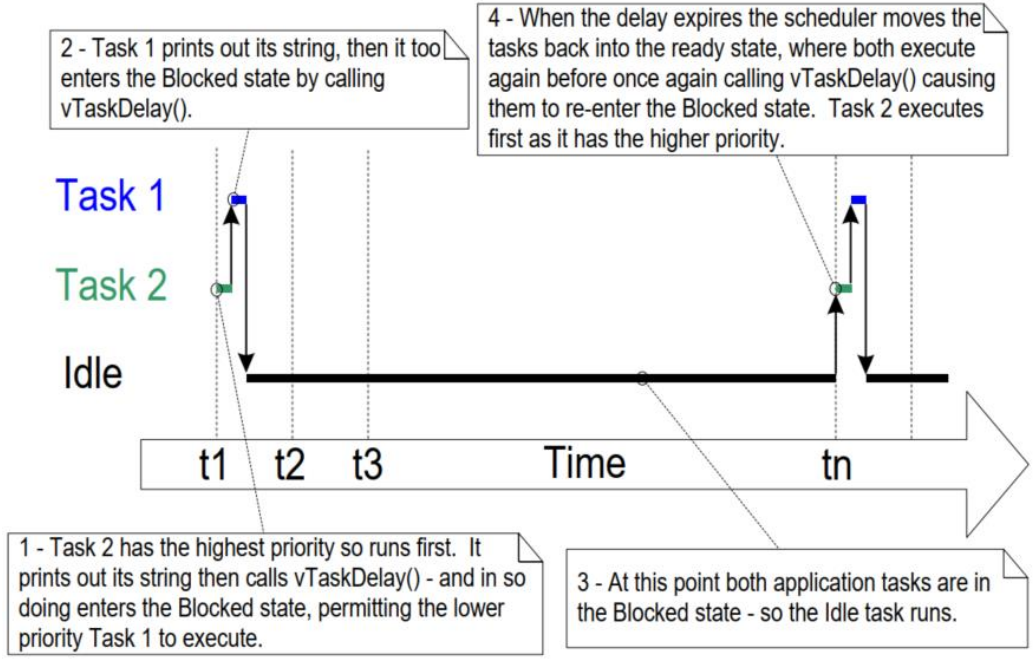




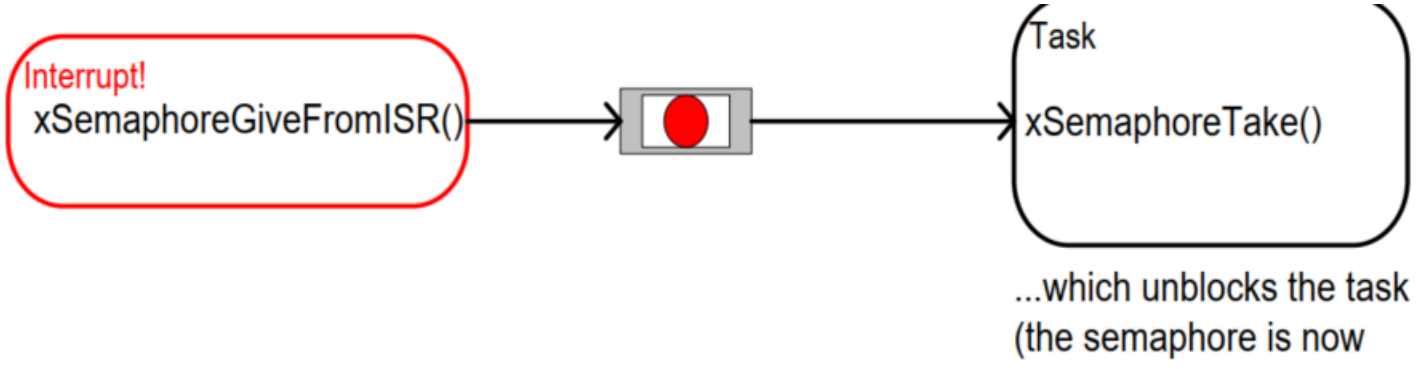


FreeRTOS - TaskDelay

```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
const TickType_t xDelay250ms = pdMS_TO_TICKS( 250 );

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = ( char * ) pvParameters;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Delay for a period.  This time a call to vTaskDelay() is used which places
        the task into the Blocked state until the delay period has expired.  The
        parameter takes a time specified in 'ticks', and the pdMS_TO_TICKS() macro
        is used (where the xDelay250ms constant is declared) to convert 250
        milliseconds into an equivalent time in ticks. */
        vTaskDelay( xDelay250ms );
    }
}
```



FreeRTOS – IPC Sync -Semaphores

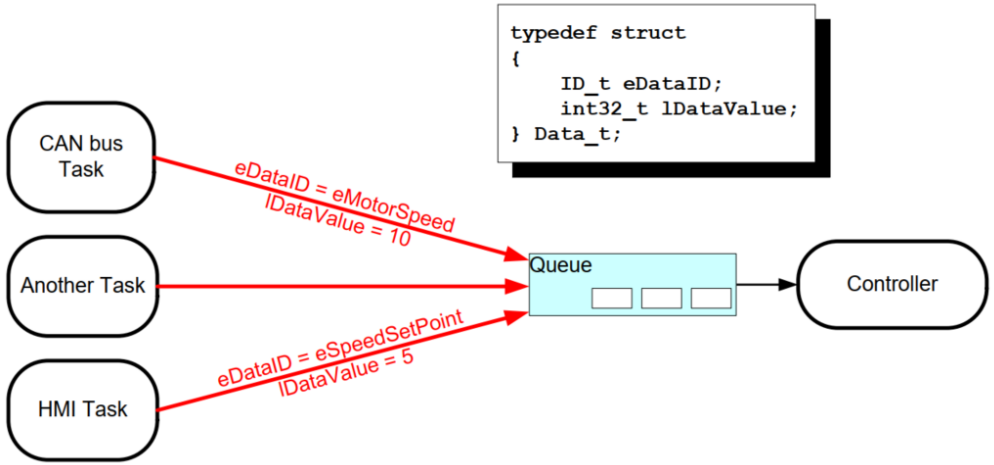



```
SemaphoreHandle_t xSemaphoreCreateBinary( void );
```

```
BaseType_t xSemaphoreTake( SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait );
```

```
BaseType_t xSemaphoreGiveFromISR( SemaphoreHandle_t xSemaphore,
                                  BaseType_t *pxHigherPriorityTaskWoken );
```

FreeRTOS – IPC Communication Queue




```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength, UBaseType_t uxItemSize );

BaseType_t xQueueReceive( QueueHandle_t xQueue,
                          void * const pvBuffer,
                          TickType_t xTicksToWait );

BaseType_t xQueueSendToFront( QueueHandle_t xQueue,
                              const void * pvItemToQueue,
                              TickType_t xTicksToWait );
```

Minimal FreeRTOS

- FreeRTOS/Source/tasks.c
- FreeRTOS/Source/queue.c
- FreeRTOS/Source/list.c
- FreeRTOS/Source/portable/[compiler]/[architecture]/port.c.
- FreeRTOS/Source/portable/MemMang/heap_x.c where ‘x’ is 1, 2, 3, 4 or 5.
- FreeRTOS/Source/include
- FreeRTOS/Source/portable/[compiler]/[architecture].
- Whichever directory contains the FreeRTOSConfig.h