FreeRTOS Tasks

Norman McEntire
norman.mcentire@gmail.com

Textbook Reference

- Mastering the FreeRTOS Real Time Kernel
  by Richard Barry
    - Chapter 3: Task Management

Topics

- 3.1 Intro and Scope
- 3.2 Task Functions
- 3.3 Top Level Task States
- 3.4 Creating Tasks
- 3.5 Task Priorities
- 3.6 Time Measurement and Tick Interrupts
- 3.7 Expanding the “Not Running” State
- 3.8 The Idle Task and the Idle Task Hook
- 3.9 Changing the Priority of a Task
- 3.10 Deleting a Task
- 3.12 Scheduling Algorithms

3.1 Intro and Scope

- Concepts Covered
  - How FreeRTOS allocates processing time to each task
  - How FreeRTOS chooses which task should execute
  - How FreeRTOS handles task priorities
  - How FreeRTOS Task States work

3.1 Intro and Scope

- Skills and APIs Covered
  - How to create tasks
  - How to use the task parameter
  - How to change the priority of a task
  - How to delete a task
  - How to implement periodic processing of a task
  - How to use the idle task

This is the most detailed chapter in the whole book - many key concepts that are used throughout the remainder of the book

3.2 Task Functions

- Tasks are implemented as C functions with the following prototype
  - void MyTaskFunction(void *pvParameters)
- Each task runs independent of other tasks
  - It has an entry point
  - Has it's own stack
  - It never ends - runs forever
    - for (;;);
    - NOTE: Then task function should never return - rather use TaskDelete if for some reason the task should stop

Code Segment

- ```
  void MyTaskFunction(void *pvParams) {
      int32_t lVariableExample = 0;
      for ( ;; ) {
          // The code to implement the task goes here
      }
  }
  ```
  ```

3.3 Top Level Task States

- At the highest level, a Task is in one of two states:
    - Running
    - Not Running
- The above is actually a simplification, but for now this works fine - more info later in the course

Block Diagram

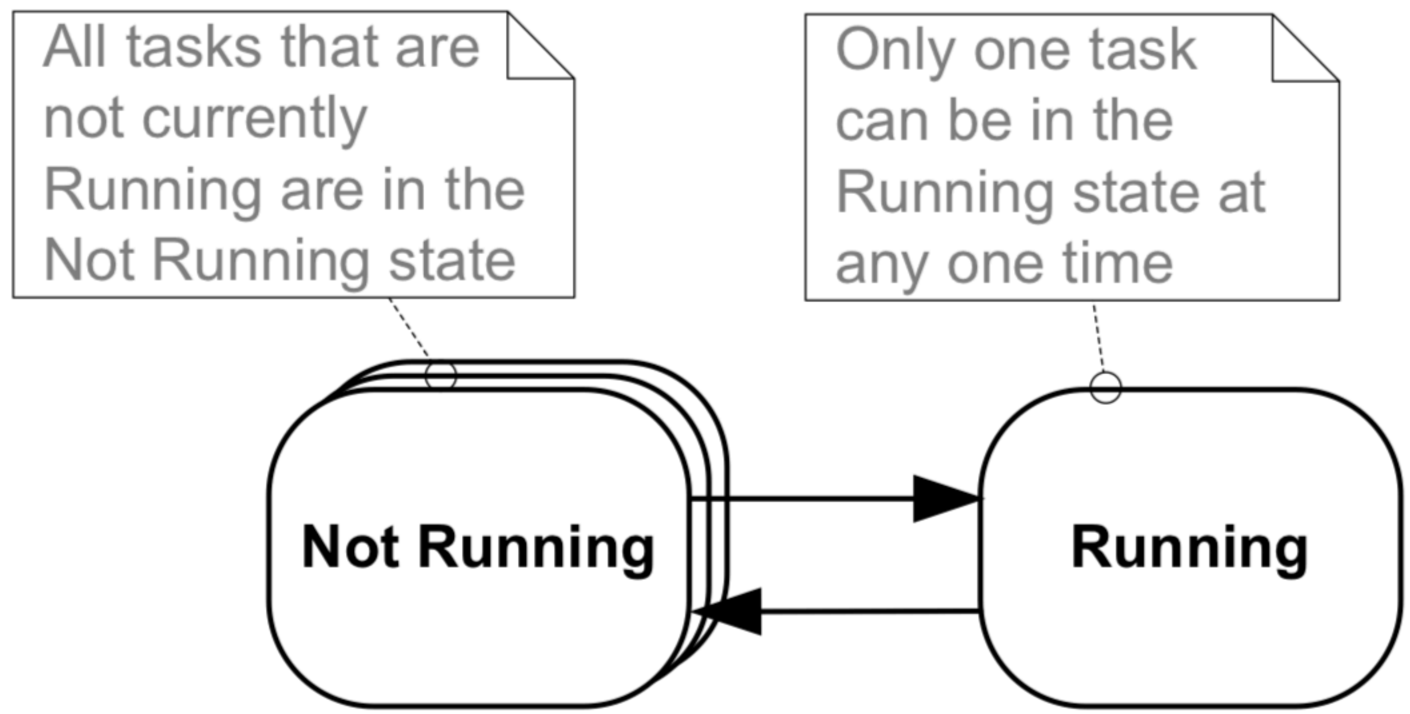


Figure 9. Top level task states and transitions

Task States

- The FreeRTOS scheduler manages the transition of task states
  - Swapped In
    - Transition from Not Running to Running
  - Swapped Out
    - Transition from Running to Not Running

3.4 Creating Tasks

- Two APIs to create tasks
  - xTaskCreate()
  - xTaskCreateStatic()

xTaskCreate()

- BaseType_t
  xTaskCreate(
  TaskFunction_t pvTaskCode, //The C function
  const char * const pcName, //Descriptive Name
  unit16_t usStackDepth, //Number of words (not bytes)
  void *pvParameters, //Optional pointer to parameters
  UBaseType_t uxPriority, //0 = lowest,
  configMAX_PRIORITIES-1 = max
  TaskHandle_t *pxCreatedTask) //Set to NULL if not needed

- Return Values
  - pdPASS or pdFAIL

Code Demo - Part 1

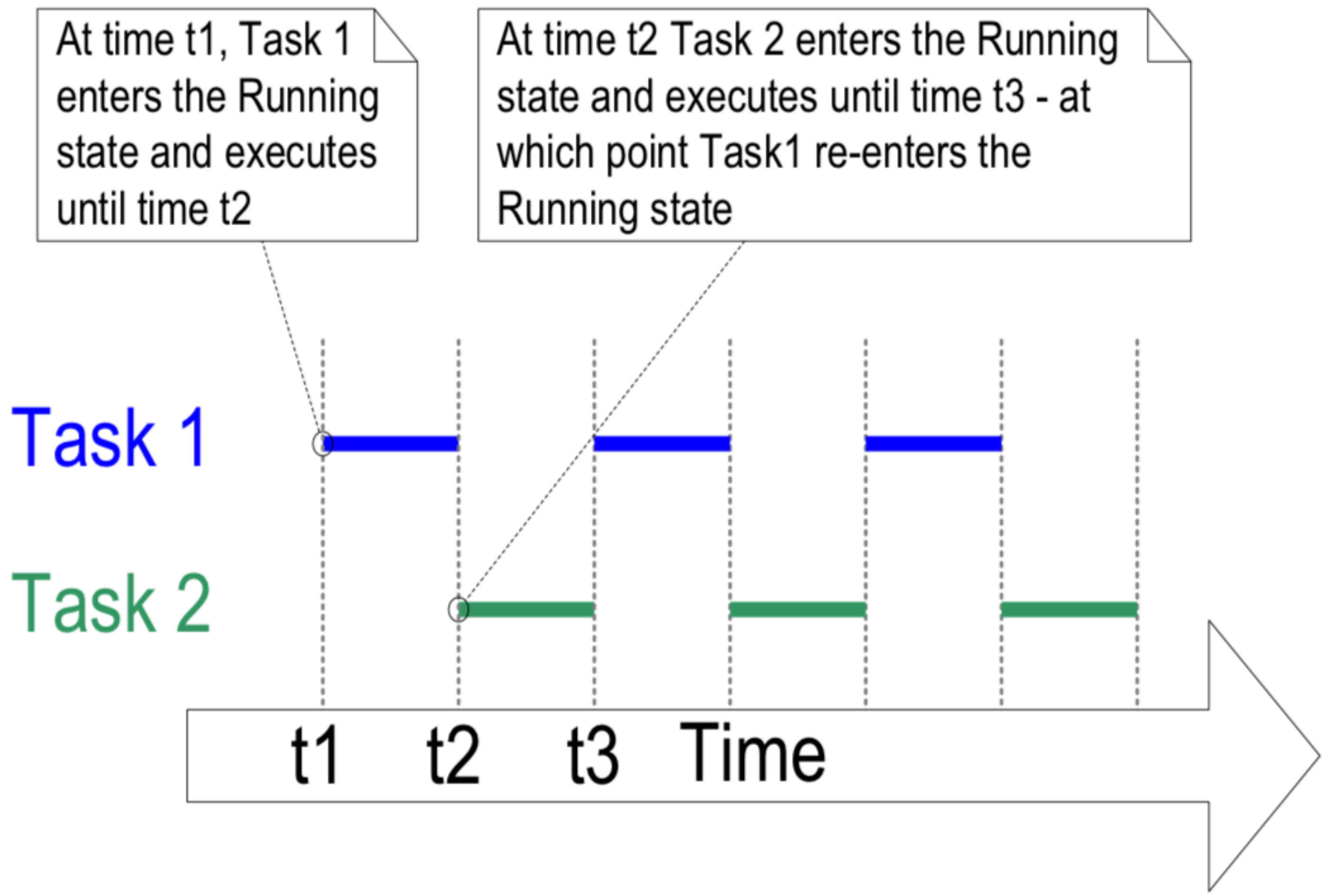
- ```
  void vTask1(void *pvParams) {
      const char *pcMsg = “Task 1 is running\r\n”;
      for (;;) {
          vPrintString(pcMsg);
          …
      }
  }
  ```
  ```

Code Demo - Part 2

- int main(void) {

```
    xTaskCreate( vTask1,
                 "Task1",
                 1000, //1000 words, 4K bytes
                 NULL, //No parameter needed
                 1, //Priority 1
                 NULL); //Task handle not needed
    vTaskStartScheduler();
    for(;;); //Should never get here
}
```

Block Diagram



Code Demo - Part 1
Passing Parameters

- ```
  void vTaskFunction(void *pvParams) {
      char *pcMsg = (char *) pvParams;
      for (;;) {
          vPrintString(pcMsg);
          …
      }
  }
  ```
  ```

Code Demo - Part 2
Passing Parameters

- static const char *pcMsg1 = “Task 1\r\n”;
- static cost char *pcMsg2 = “Task 2\r\n”;

Code Demo - Part 3 Passing Parameters

- int main(void) {
    xTaskCreate(vTaskFunction, “Task 1”, 1000,
      (void *)pcMsg1, 1, NULL);
    xTaskCreate(vTaskFunction, “Task 2”, 1000,
      (void *)pcMsg2, 1, NULL);
    vStartScheduler();
    for (;;); //Never get here
  }

3.5 Task Priorities

- uxPriority parameter of xTaskCreate() assigns an initial priority
- Use vTaskPrioritySet() to change priority
- configMAX_PRIORITIES configures the maximum number of priorities
- Priority 0 is always the lowest priority
- configMAX_PRIORITIES-1 is always max priority

Scheduler Options

- Option 1
  - configUSE_PORT_OPTIMISED_TASK_SELECTION set to 0
  - Uses generic C code to do task selection
- Option 2
  - configUSE_PORT_OPTIMIZED_TASK_SELECTION set to 1
  - Faster than generic method
  - Uses port optimized assembly code for task selection
  - configMAX_PRIORITIES cannot be greater than 32
  - Not available on all ports

Scheduler always runs highest priority task

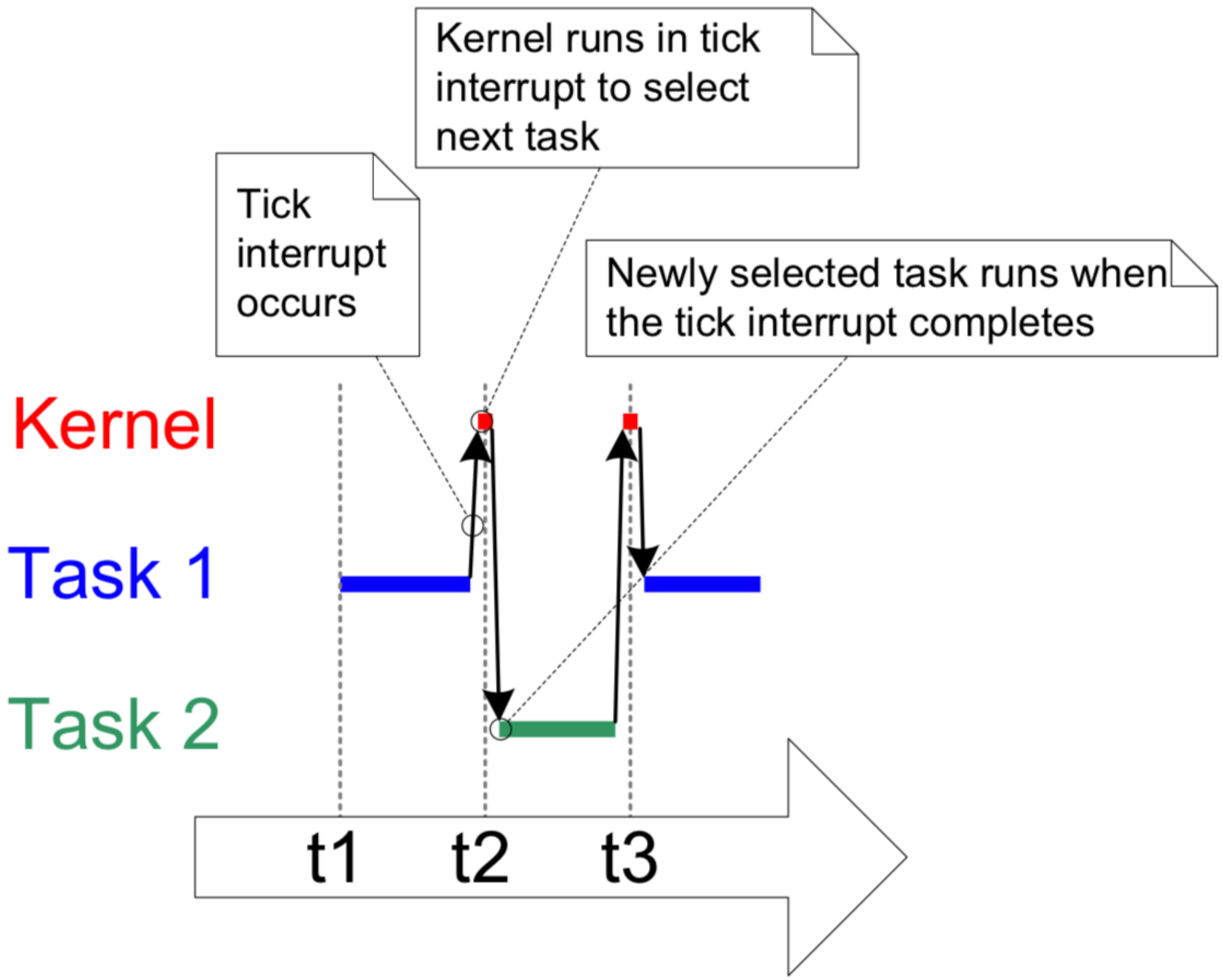
- Scheduler always runs the highest priority task that is ready to run

- If two or more tasks are at same priority, and are ready to run, then scheduler will transition each task out into and out of the Running state in turn

3.5 Time Measurement and Tick Interrupts

- A periodic interrupt is used to enable time slicing
- At each time slice the scheduler runs to see what task runs next
- configTICK_RATE_HZ sets the frequency of the timer
    - Example: configTICK_RATE_HZ 100 is every 10msec

Tick Interrupt and Scheduler



pdMS_TO_TICKS()

- TickType_t pdMS_TO_TICKS(BaseType_t millisec)
- Use this macro to convert from millsecs to time ticks used by FreeRTOS
- FreeRTOS API always specifies time in ticks, hence this macro useful to convert msec to ticks
- Example - Convert 200 milliseconds
  - TickType_t xTimeInTicks = pdMS_TO_TICKS(200)

Code Demo - Part 1

- ```
  void vTaskFunction(void *pvParams) {
      char *pcMsg = (char *) pvParams;
      for (;;) {
          vPrintString(pcMsg);
          …
      }
  }
  ```
  ```

Code Demo - Part 2

- static const char *pcMsg1 = “Task 1 Running\r\n”;
  static const char *pcMsg2 = “Task 2 Running\r\n”;

- int main(void) {
  xTaskCreate(vTaskFunction, ”Task 1”, 1000,
  (void *) pcMsg1, 1, NULL);

  xTaskCreate(vTaskFunction, “Task 2”, 1000,
  (void *) pcMsg2, 2, NULL);

  vTaskStartScheduler();

  return 0; //Will never get here
  }

3.7 Expanding the “Not Running” State

- So far we have only look at two task states:
    - Running
    - Not Running
- This was a simplification - there is more to Not Running
    - Specifically, we need to focus on event-driven tasks

Event-Driven Tasks

- An event-driven task performs work only after an event triggers it
    - The task does not enter the running state until the event occurs
- Using event-driven tasks means that tasks can be created at different priorities without the highest priority starving all the lower priority tasks

The Blocked State

- A subset of the “Not Running” state is the Blocked State
- Tasks enter the Blocked State for three types of events
  - #1. Temporal (time-related) Events
    - The event is after a delay period or an absolute time being reached
  - #2. Synchronization Events
    - Events originate from other tasks or interrupts
  - #3. Both Sync and Time Event
    - That is, sync events that include a timeout of how long to wait

Kernel Objects that Create Synchronization Events

- Queues
- Binary Semaphores
- Counting Semaphores
- Mutexes
- Recursive Mutexes
- Event Groups
- Task Notifications

We cover
all of these
in future lessons

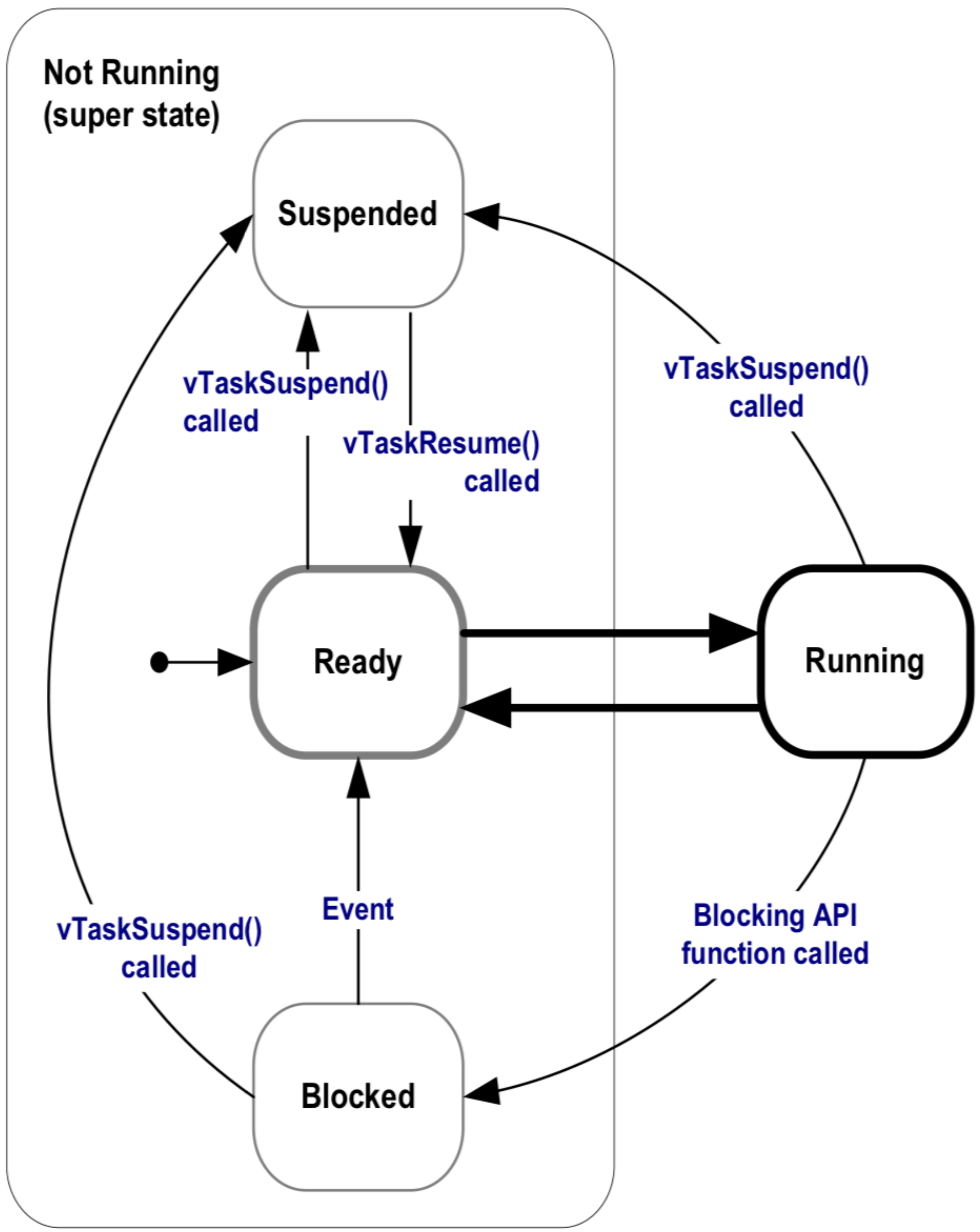
The Suspended State

- Suspended State is another sub-state of "Not Running"
- The only way **into** the Suspended State
    - vTaskSuspend()
- The only way **out of** Suspended State
    - vTaskResume()
- NOTE: Most apps do not use Suspended State

The Ready State

- Tasks that are in the Not Running state, but are NOT Blocked or NOT Suspended are in the Ready State
    - They are Ready to run, but not yet running

Full Task State Machine



vTaskDelay()

- void vTaskDelay(TickType_t xTicksToDelay)
- Notes
    - Puts the task into the Blocked State
    - Use pdMS_TO_TICKS() macro to set the xTicksToDelay()

Code Demo - Part 1
Using vTaskDelay() Create Delay

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

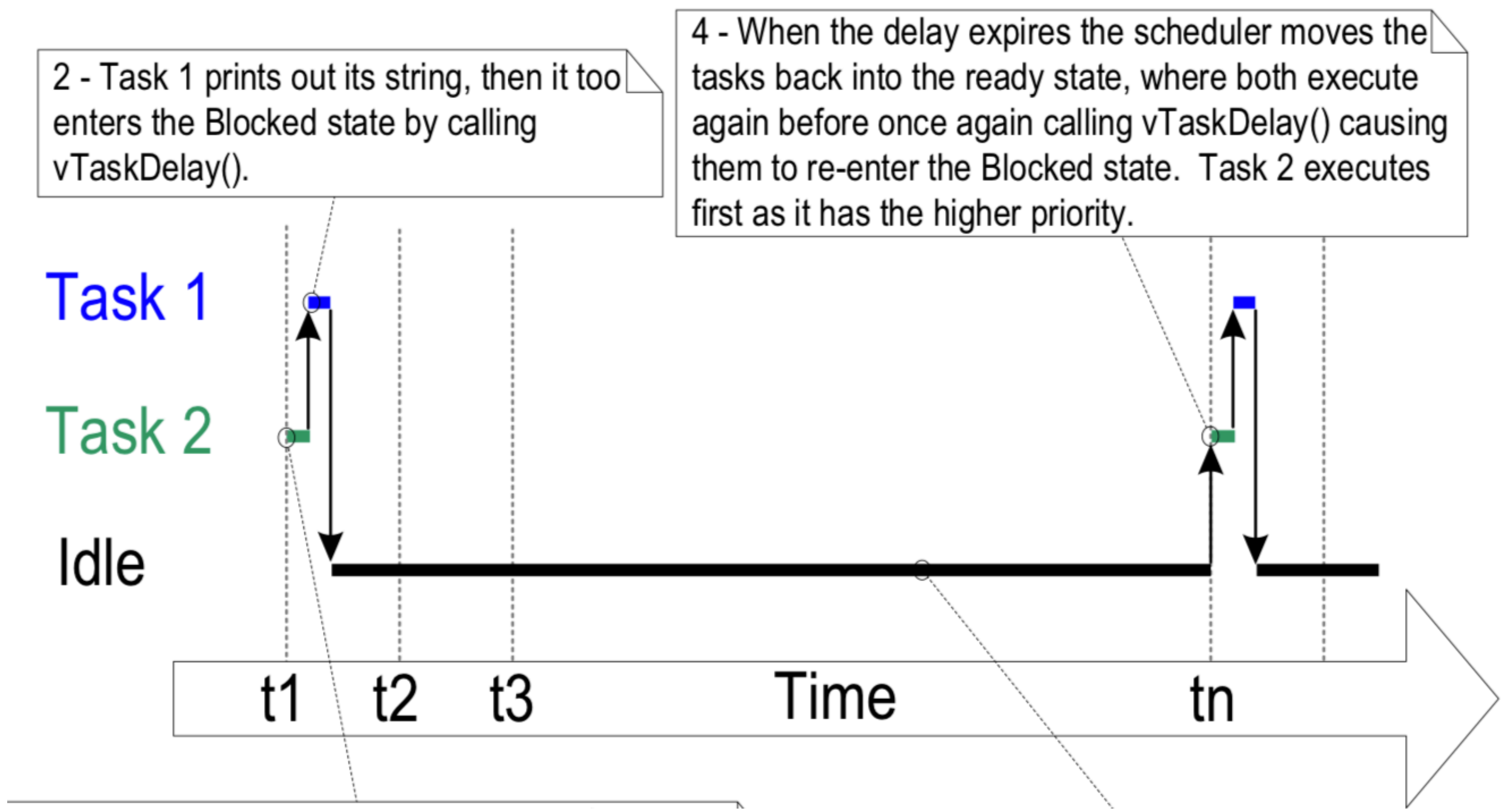
      pcTaskName = (char *)pcParams;

      for ( ;; ) {
          vPrintString(pcTaskName);
          vTaskDelay(xDelay250ms);
      }
  }
  ```
  ```

Idle Task

- The Idle Task is created automatically when the scheduler is started
  - Hence always at least one task in the Ready state

Block Diagram



vTaskDelayUntil()

- Void
  vTaskDelayUntil(
      TickType_t *pxPreviousWakeTime,
      TickType_t xTimeIncrement);

- Notes

  - Similar to vTaskDelay(), except the time delay is absolute instead of relative

  - Use xTaskGetTickCount() to initialize value of pxPreviousWakeTime

Code Demo vTaskDelayUntil()

- ```
  void vTaskFunction(void *pvParams) {
      char *pcTaskName;
      TickType_t xLastWakeTime;

      pcTaskName = (char *) pvParams;

      xLastWakeTime = xTaskGetTickCount();

      for( ;; ) {
          vPrintString(pcTaskName);
          vTaskDelayUnti(&xLastWakeTime, pdMS_TO_TICKS(250));
      }
  }
  ```
  ```

Combining Blocking and Non-Blocking Tasks

- What we will demonstrate in the following example
  - Two tasks created at priority 1
    - Just spin in loop and use time - never in the Blocked state
  - A third task at priority 2 (higher priority than 1)
    - Prints out string periodically
    - Use vTaskDelayUntil()

Code Demo - Part 1

- ```
  void
  vContinuousProcessingTask(void *pvParams) {
      char *pcTaskName;
      pcTaskName = (char *) pvParams;

      for (;;) {
          vPrintString(pcTaskName)
      }
  }
  ```
  ```

Code Demo - Part 2

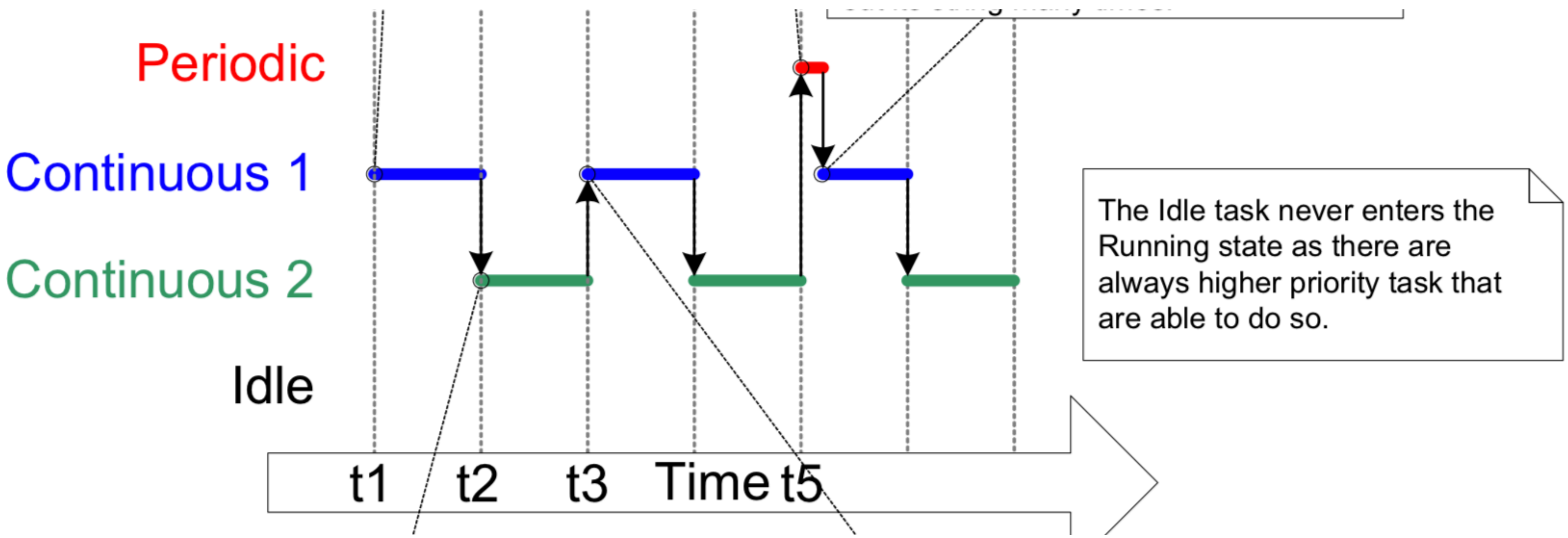
- void

```
vPeriodicTask(void *pvParams) {
    TickType_t xLastWakeTime;
    const TickType_t xDelay3ms = pdMS_TO_TICKS(3);

    xLastWakeTime = xTaskGetTickCount();

    for( ;; ) {
        vPrintString(“Periodic task running\r\n”);
        vTaskDelayUntil(&lastWakeTime, xDelay3ms);
    }
}
```

Block Diagram



3.8 The Idle Task and the Idle Task Hook

- There must always be at least one task that can enter the Running state
  - Hence the need for the Idle task - if no other tasks need the Run state, then it will be the Idle task
- Idle task has lowest possible propriety - 0
- If needed, you can add app specific functions to the Idle task
  - See next slide

Some uses of Idle Hook Function

- Executing low priority, background, or continuous processing functionality
- Measuring the amount of spare processing capacity
- Placing the processor into low-power mode

Idle Task Hook Function

- void vApplicationIdleHook(void)
- Notes
  - configUSE_IDLE_HOOK must be 1
  - Must call the hook function with name show above
  - Must never attempt to block or suspend

Code Demo
Idle Hook Function

- votatile unit32_t ulIdleCycleCount = 0UL;

- void vApplicationIdleHook(void) {
    ulIdleCycleCount++;
  }

3.9 Changing the Priority of a Task

- Use vTaskPrioritySet() to change priority of task after the scheduler has been started
  - See next slide

vTaskPrioritySet()

- void
  vTaskPrioritySet(
      TaskHandle_t pxTask,
      UBaseType_t uxNewPriority)

- Notes
  - Must have INCLUDE_vTaskPrioritySet to 1 in FreeRTOSConfig.h

uxTaskPriorityGet()

- UBaseType_t uxTaskPriorityGet(TaskHandle_t pxTask)
- Notes
  - Set pxTask to NULL to query your own priority

3.9 Deleting a Task

- A task can use vTaskDelete() API to
  - Delete itself
  - Delete any other task

vTaskDelete()

- Void TaskDelete(TaskHandle_t pxTaskToDelete)
- Notes
  - INCLUDE_vTaskDelete must be set to 1 in FreeRTOSConfig.h
  - The Idle task will free any remaining memory for this task
    - Hence if you call vTaskDelete() do not starve the processor for time - the Idle Task needs to run

3.12 Scheduling Algorithms

- Review of Task States
  - Running - the task that is actually executing
  - Ready - ready to run when processor available
  - Blocked - waiting on an event
  - Suspended - suspended until resume API called

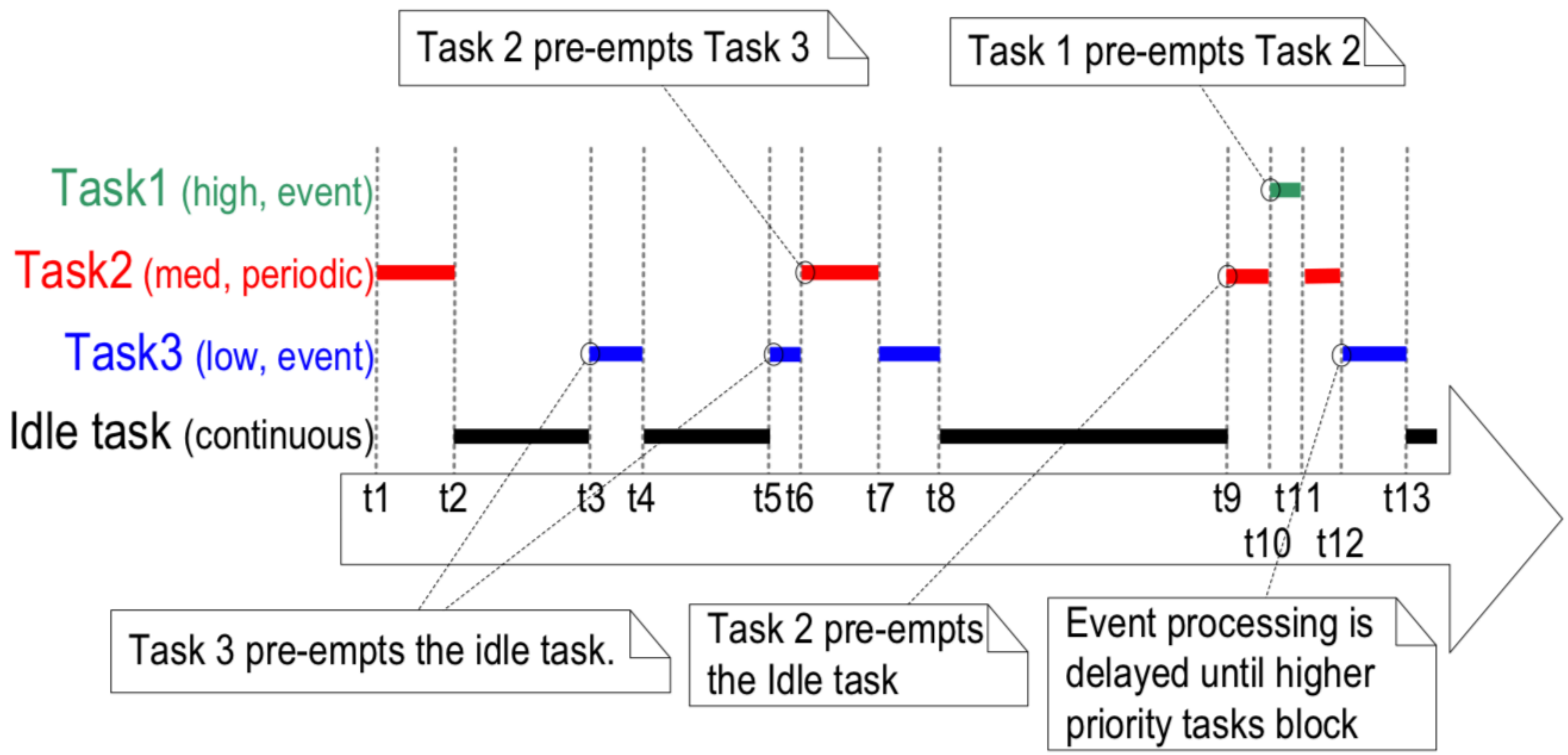
Configuring Scheduling Algorithm

- Three configuration constants related to scheduling
  - configUSE_PREEMPTION
  - configUSE_TIME_SLICING
  - configUSE_TICKLESS_IDLE
    - For use in very low power situations where tick clock is turned off

Fixed Priority Pre-Emptive Scheduling

- configUSE_PREEMPTION 1
- configUSE_TIME_SLICING 1
- Definitions
    - Fixed Priority - Priority of tasks not changed
    - Pre-Emptive - Running task immediately pre-empted if higher priority task enters running state
    - Time Slicing - used to share processing time between Ready tasks at the same priority

Block Diagram



Prioritized Pre-Emptive Scheduling (without Time Slicing)

- configUSE_PREEMPTION 1
- configUSE_TIME_SLICING 0
- Definitions
  - Pre-Emptive - Running task immediately pre-empted if higher priority task enters running state
  - But no Time Slicing for tasks at same priority - they much release processor on their own

Co-Operative Scheduling

- configUSE_PREEMPTION 0
- configUSE_TIME_SLICING 0
- Definitions
  - No pre-emption or time slicing - tasks must co-operate to share time
  - Least used of the scheduling options

Summary

- 3.1 Intro and Scope
- 3.2 Task Functions
- 3.3 Top Level Task States
- 3.4 Creating Tasks
- 3.5 Task Priorities
- 3.6 Time Measurement and Tick Interrupts
- 3.7 Expanding the “Not Running” State
- 3.8 The Idle Task and the Idle Task Hook
- 3.9 Changing the Priority of a Task
- 3.10 Deleting a Task
- 3.12 Scheduling Algorithms