FreeRTOS Queues

Norman McEntire
norman.mcentire@gmail.com

Textbook Reference

- Mastering the FreeRTOS Real Time Kernel by Richard Barry

  - Chapter 4: Queue Management

Topics

- 4.1 Intro and Scope
- 4.2 Characteristics of a Queue
- 4.3 Using a Queue
- 4.4 Receiving Data From Multiple Sources
- 4.5 Working with Large or Variable Sized Data
- 4.6 Receiving from Multiple Queues
- 4.7 Using a Queue to Create a Mailbox

4.1 Intro and Scope

- Queues provide communication for
  - Task-to-Task
    - Covered in next Lesson
  - Task-to-Interrupt
    - Covered in next Lesson
  - Interrupt-to-Task
    - Covered in next Lesson

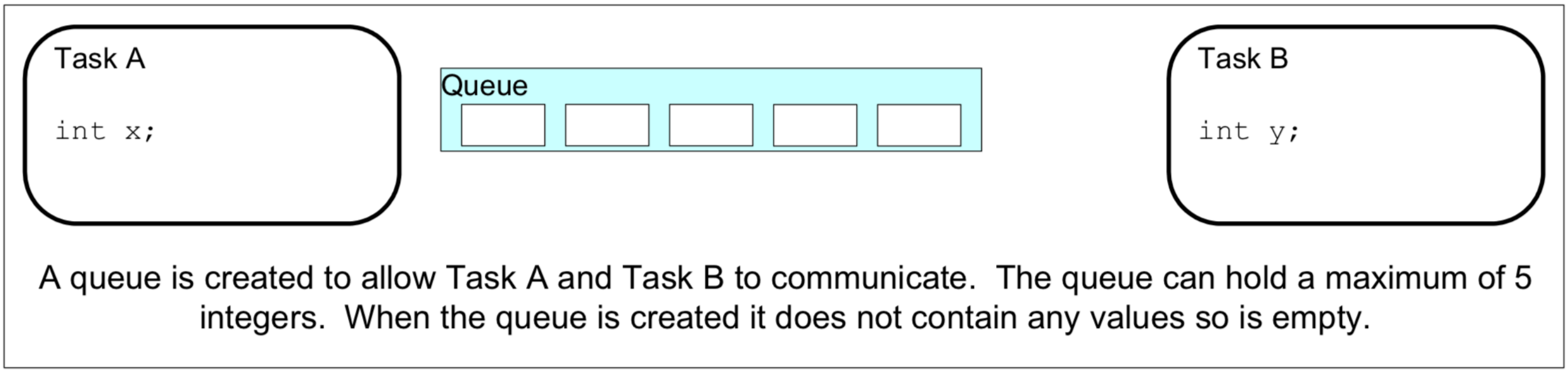
Queue Characteristics Part 1

- A queue holds….
  - A **finite number**
    - Of **fixed sized** data items
- Normally used as FIFO Buffer
  - First In (tail of queue), First Out (head of queue)

Queue Characteristics Part 2

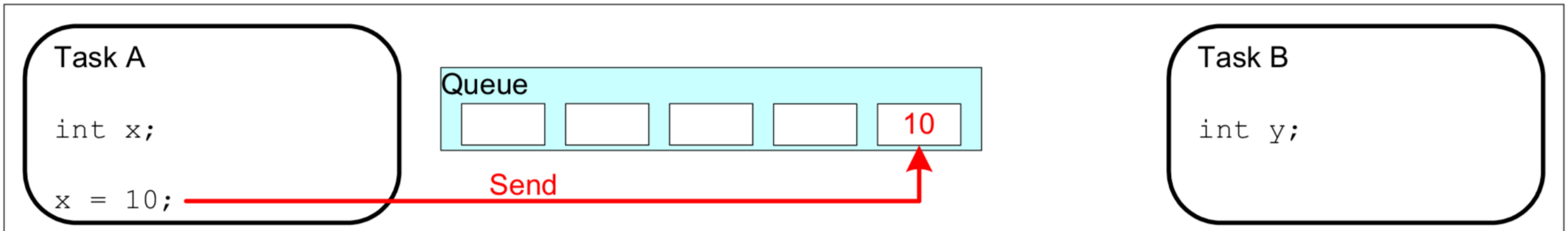
- Queue Parameters at Creation
  - **Size** of each item in queue
  - **Length** of queue

Queue Example - Part 1



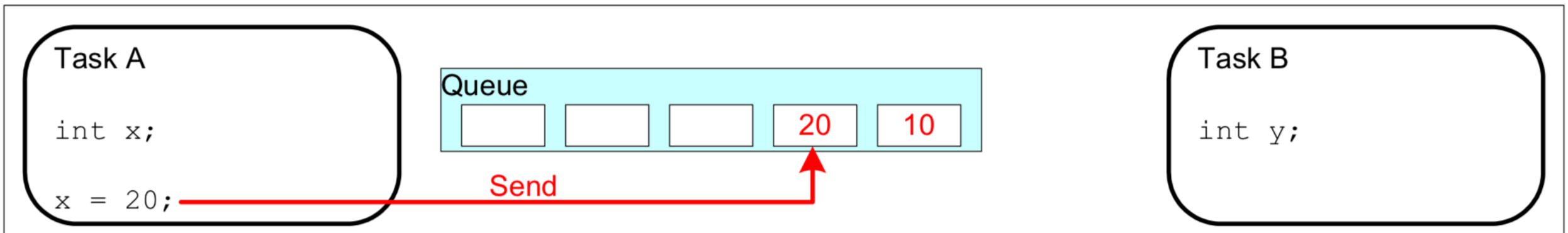
A queue is created to allow Task A and Task B to communicate. The queue can hold a maximum of 5 integers. When the queue is created it does not contain any values so is empty.

Queue Example - Part 2



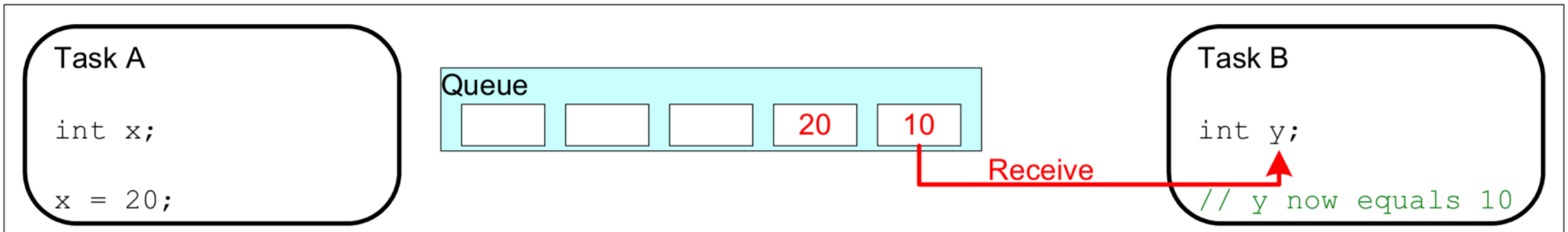
Task A writes (sends) the value of a local variable to the back of the queue. As the queue was previously empty the value written is now the only item in the queue, and is therefore both the value at the back of the queue and the value at the front of the queue.

Queue Example - Part 3



Task A changes the value of its local variable before writing it to the queue again. The queue now contains copies of both values written to the queue. The first value written remains at the front of the queue, the new value is inserted at the end of the queue. The queue has three empty spaces remaining.

Queue Example - Part 4

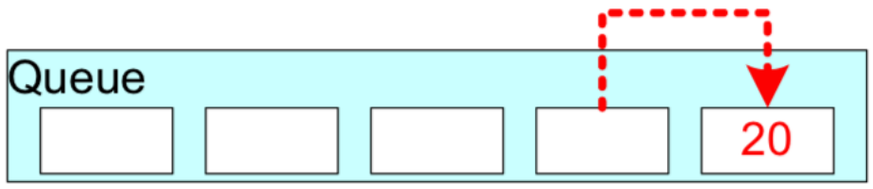


Task B reads (receives) from the queue into a different variable. The value received by Task B is the value from the head of the queue, which is the first value Task A wrote to the queue (10 in this illustration).

Queue Example - Part 5

Task A

```
int x;

x = 20;
```



Task B

```
int y;

// y now equals 10
```

Task B has removed one item, leaving only the second value written by Task A remaining in the queue. This is the value Task B would receive next if it read from the queue again. The queue now has four empty spaces remaining.

Two Ways To Implement Queue

- Queue By Copy
  - Data sent to queue is copied into the queue
  - This is method used by FreeRTOS
- Queue By Reference
  - Reference to data sent to queue
  - Not used by FreeRTOS

Queue Access by Multiple Tasks

- Queues are objects
  - Can be accessed by tasks
    - Any number of tasks can write to queue
    - Any number of tasks can read from queue
- Can be accessed by ISRs

Most Common Queue Operation

- The most common queue operation is to have one or more writers but only a single reader

Concept

Queue Writes

A task can specify a **block time** when writing to a queue

Block time is the maximum time task should be held in Blocked State waiting for room in the queue

Queue Writes

- Task can specify Block Time when writing to a queue

- Maximum time the task should be held in the Blocked Task to wait for space to become available in the queue

Queue Writes
Multiple Writers

- Possible to have full queue with multiple writers
  - Hence all writers enter Blocked state
- When space becomes available, only one task will be unblocked
  - The task with highest priority
  - If equal priority, the task waiting the longest

Queue Sets

- Queues can be grouped into sets
  - Allows a task to enter

Concept:

Queue Sets

Queues can be grouped into sets

A task can enter the blocked state to wait for data from any queue in the set

Queue Reads Blocking

- When a task reads from a queue, it can specify a "block time"

- Block time: time the task will be kept in the Blocked state to wait for data to enter the queue

    - Task will be removed from the Blocked state to the Ready state if specified block time expires

Queue Reads
Multiple readers

- If a queue has multiple readers, all will be in Blocked State until data available
- Only one task will be unblocked when data arrives
  - Task with highest priority is unblocked
  - If all tasks are equal priority, then the one that has been waiting the longest

FreeRTOS Queue Management Part 2: APIs

xQueueCreate()
xQueueCreateStatic()

- Use these APIs to create a queue

- QueueHandle_t
  xQueueCreate(
  UBaseType_t uxQueueLength,
  UBaseType_t uxItemSize)

- Returns
  - NULL if not enough space to allocate queue
  - Non-Null if queue created - use this as the handle

xQueueReset()

- Call this to reset the queue to it's original empty state

xQueue**Send**()
xQueue**SendToBac**k()
xQueue**SendToBackFromISR**()

- Use these to send data to back of the queue
- These do the exact same thing
  - Typically use xQueueSend()
- From ISR only use this one
  - xQueueSendToBackFromISR()

xQueue**SendToBack**()

- BaseType_t
  xQueueSendToBack(
  QueueHandle_t xQueue,
  const void *pvItemToQueue,
  TickType_t xTicksToWait)

- Return Values
  - pdPASS - Data sent to queue
  - errQUEUE_FULL - if data could not be written
- Notes
  - xQueue is handle returned from xQueueCreate()
  - pvItemToQueue points to data to send
  - TicksToWait is in tick periods
    - Use pdMS_TO_TICKS() macro to convert from milli to ticks
    - Use 0 to return immediately if queue full
    - Use portMAX_DELAY to wait “forever”

xQueue**SendToFront**()
xQueue**SendToFrontFromISR**()

- xQueueSendToFront()
  - Send to front of the queue
- Same parameters as xQueueSend()

xQueue**Receive**()
xQueue**ReceiveFromISR**()

- Read an item from the queue
- BaseType_t
  xQueueReceive(
    QueueHandle_t xQueue,
    void *const pvBuffer,
    TickType_t xTicksToWait)
- Returns
  - pdPASS - if data successfully returned
  - errQUEUE_EMPTY - data cannot be read from queue
- Notes
  - xQueue is handle returned from xQueueCreate()
  - pvBuffer points to data received
  - TicksToWait is in tick periods
    - Use pdMS_TO_TICKS() macro to convert from milli to ticks
    - Use 0 to return immediately if queue full
    - Use portMAX_DELAY to wait “forever”

uxQueueMessagesWaiting()

- Query the number of messages in a queue
- UBaseType_t uxQueueMessageWaiting(QueueHandle_t xQueue)
- Return value
  - If 0, then no items in queue
  - Otherwise the number of items in the queue

FreeRTOS Queue Management Part 3: Code Samples

vSenderTask()

```
• static void vSenderTask(void *pvParameters) {
      int32_t lValueToSend;
      BaseType_t xStatus;

      lValueToSend = (int32_t) pvParameters;

      for(;;) {
          xStatus = xQueueSendToBack(
                      xQueue, &lValueToSend, 0);
          if (xStatus != pdPASS) {
                vPrintString("Could not send to queue\r\n");
          }
      }
}
```

vReceiverTask()

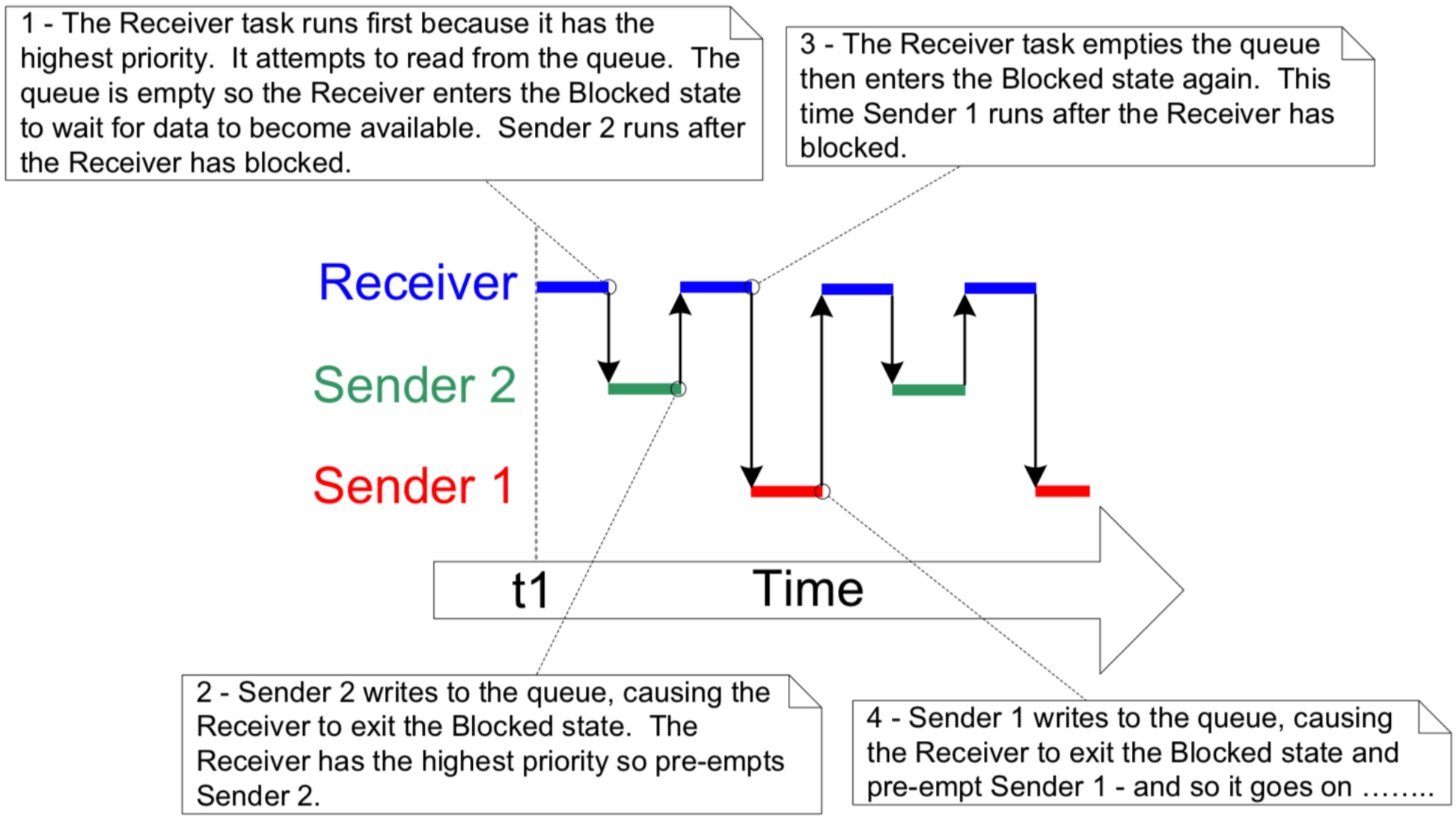
```
• static void vReceiverTask(void *pvParameters) {
      int32_t lReceiveValue;
      BaseType_t xStatus;
      Const TickType_t xTicksToWait = pdMS_TO_TICKS(100);

     for(;;) {
          if (uxQueueMessageWaiting(xQueue) != 0) {
             vPrintString(“Queue should have been empty\r\n”);
          }
          xStatus = xQueueReceive(
                     xQueue, &lReceiveValue, xTicksToWait);
          if (xStatus == pdPASS) {
                vPrintStringAndNumber(“Received: “, lReceivedValue);
          }
          else {
                vPrintString(“Could not receive from the queue\r\n”);
          }
     }
}
```

main()

- QueueHandle_t xQueue;

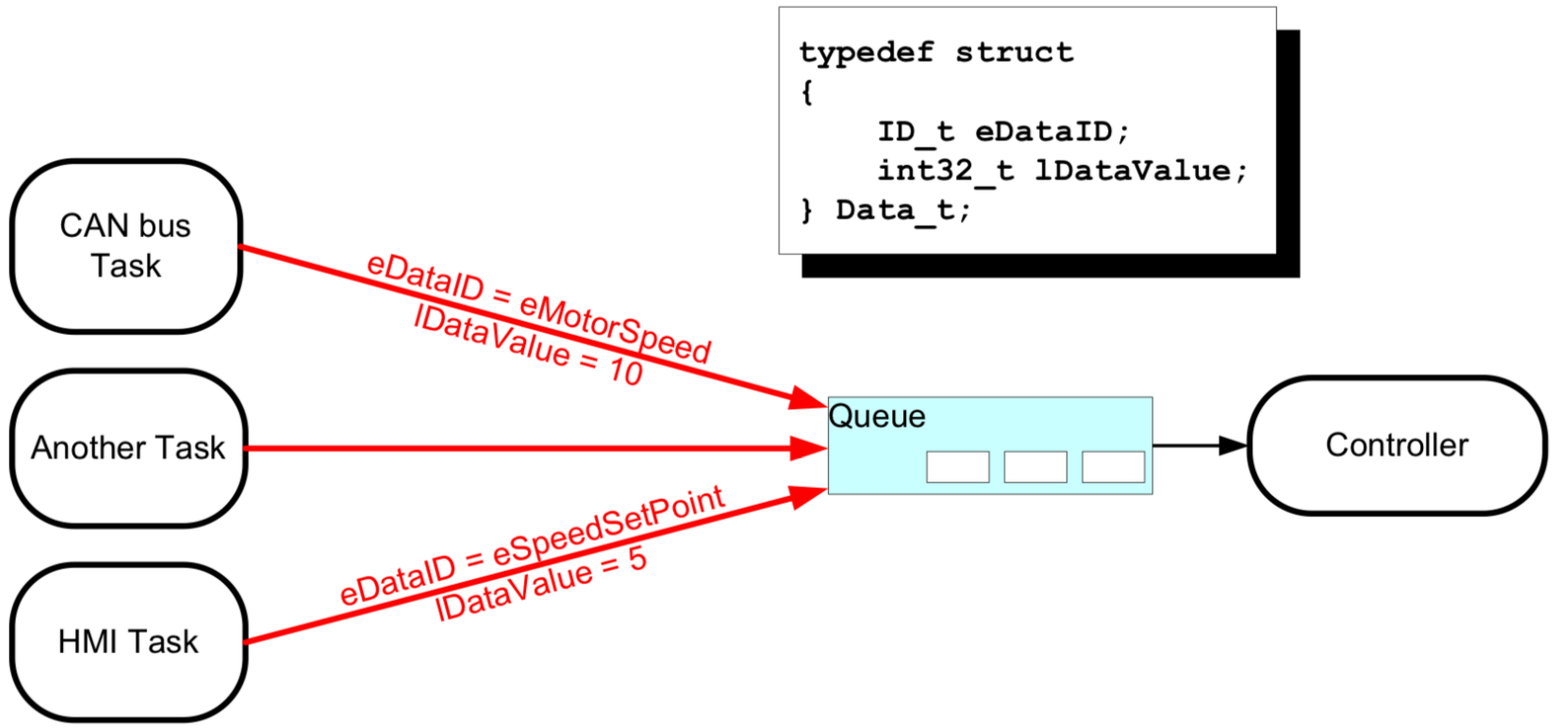
- int main(void) {
    xQueue = xQueueCreate(5, sizeof(int32_t));
    if (xQueue != NULL) {
      xCreateTask(vSenderTask,”Sender1”, 1000,
       (void *)100, 1, NULL); //Priority 1
      xCreateTask(vSenderTask,”Sender2”, 1000,
       (void *)200, 1, NULL); //Priority 1
    }
    xTaskCreate(vReceiverTask,”Receiver”, 1000,
      NULL, 2, NULL); //Priority 2
    xTaskStartScheduler();
}
else {
    vPrintString(“Could not create the queue\r\n”)
}
for (;;;);

Running Code Block Diagram



Receiving Data from Multiple Sources

- It is common in FreeRTOS designs to receive data from more than one source



NOTE: HMI = Human Machine interface

Enhance main.c - Part 1

- typedef enum {
    eSender1,
    eSender2
  } DataSource_t;

- typedef struct {
    uint8_t ucValue;
    DataSource_t eDataSource;
  } Data_t;

- Static const Data_t xStructToSend[2] = {
    { 100, eSender1 },
    { 200, eSender2 }
  };

Enhance main.c - Part 2

- int main(void) {

```
    xQueue = xQueueCreate(3, sizeof(Data_t));
    if (xQueue != NULL) {
        xTaskCreate(vSenderTask,”Sender1”, 1000,
          &(xStructureToSend[0]), 2, NULL);
       xTaskCreate(vSenderTask,”Sender2”, 1000,
          &(xStrutureToSend[1]), 2, NULL);
       xTaskCreate(vReceiverTask,”Receiver”, 1000,
          NULL, 1, NULL); //Priority 1 (lower than 2)
       vTaskStartScheduler();
    }
}
```

Enhance vReceiverTask()

- static void vReceiverTask(void *pvParameters) {
  Data_t xReceivedStructure;
  BaseType_t xStatus;
  for(;;) {
  ….
  xStatus = xQueueReceive(
  xQueue, &xReceiveStructure, 0);
  if (xStatus == pdPASS) {
  If (xReceivedStructure.eDataSource ==
  eSender1) {
  vPrintStringAndNumber(“S1: “, xReceivedStructure.ucValue)
  } else {
  vPrintStringAndNumber(“S2: “, xReceivedStructure.ucValue);
  }
  }
  }

Block Diagram of Code Execution