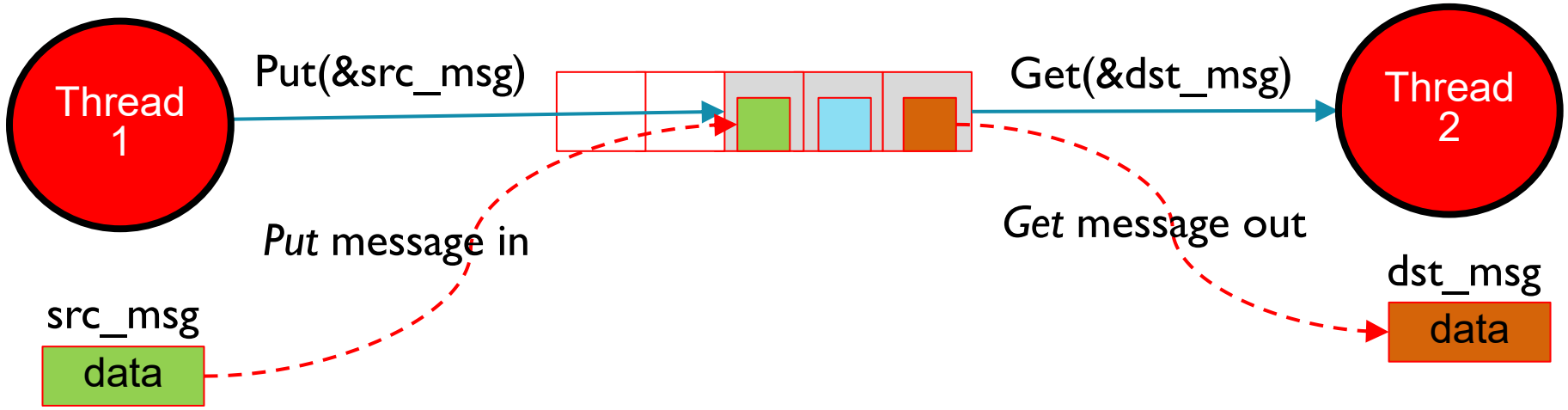
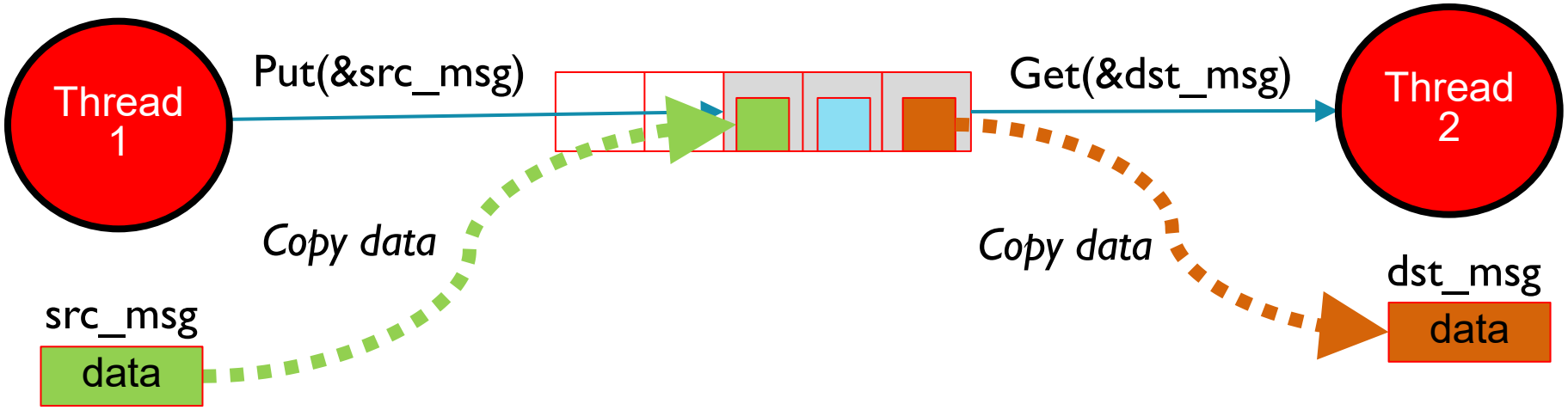
Message Queues

Message Queues



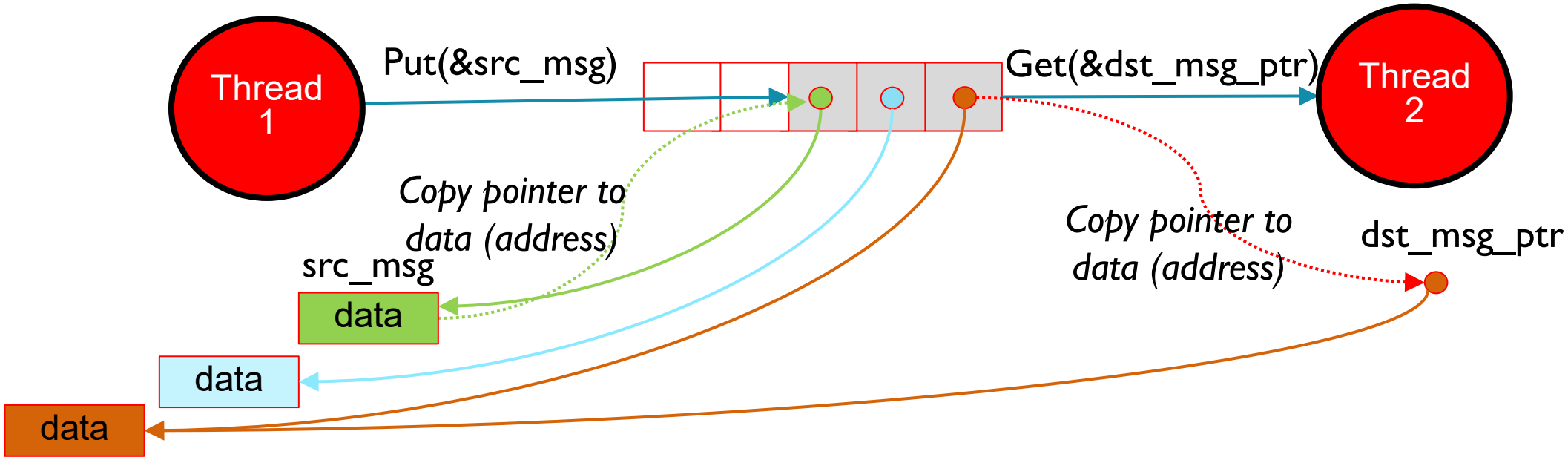
- Message data is a user-defined structure
- Queue can hold multiple messages, allows accumulation if processing is delayed
- Basic message queue operations
  - *Put* a message in the queue
  - *Get* a message from the queue
  - Priority: messages can be prioritized within queue

What Do We Pass? The Data or a Pointer to Data?



- Pass the message itself?
- Pass by Value copies the **data**
  - Takes more time to copy data.
    - Does it matter to the application? Maybe, maybe not.

What Do We Pass? The Data or a Pointer to Data?

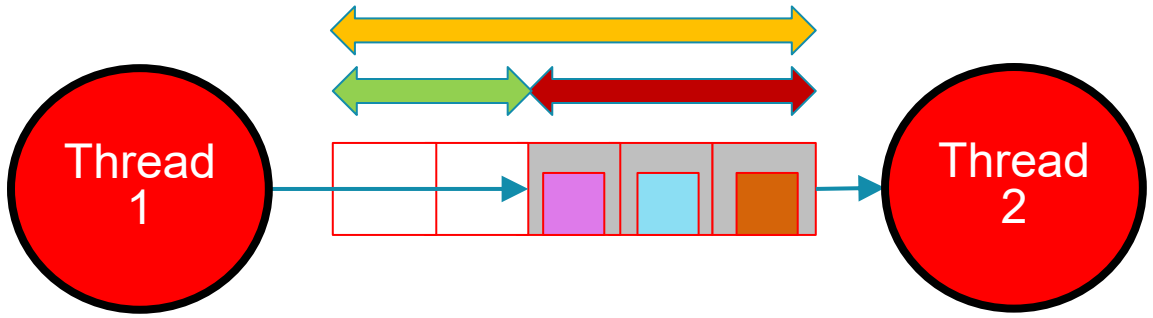


- Pass a pointer to the message?
  - Pass by Reference: Copies data **address**
  - Faster, copies less data.
  - Will fail if the data is freed before the corresponding ***get*** operation. Don't pass automatic variable (on stack) by reference unless you're sure it will work!
- CMSIS-OS2 uses Pass by Value
  - *Putting* a message in the queue copies the data from the source object into the queue
  - *Getting* a message from the queue copies the data from the queue to the destination object

CMSIS-RTOS2 Message Functions

- osMessageQueueId_t osMessageQueueNew (msg_count, msg_size, attr)
  - Creates and initializes a message queue and space for its messages
  - msg_count: maximum number of messages to hold
  - msg_size: maximum size of a single message
  - Returns ID for message queue
- osStatus_t osMessageQueuePut(mq_id, msg_ptr, msg_prio, timeout)
  - Put message msg_ptr into queue mq_id with location based on priority msg_prio
  - What if queue is full? Depends on timeout
    - 0: return immediately (try)
    - osWaitForever: wait until space is available
    - other: return when space is available or timeout passes
  - Return type osStatus_t
    - osOK, osErrorTimeout, osErrorResource, osErrorParameter
- osStatus_t osMessageQueueGet(mq_id, msg_ptr, msg_prio, timeout)
  - Get next message from queue
  - What if queue is empty? Depends on timeout
    - 0: return immediately (try)
    - osWaitForever: wait until message is available
    - other: return when message is available or timeout passes
  - Return type osStatus_t
    - osOK, osErrorTimeout, osErrorResource, osErrorParameter

More Message Queue Functions



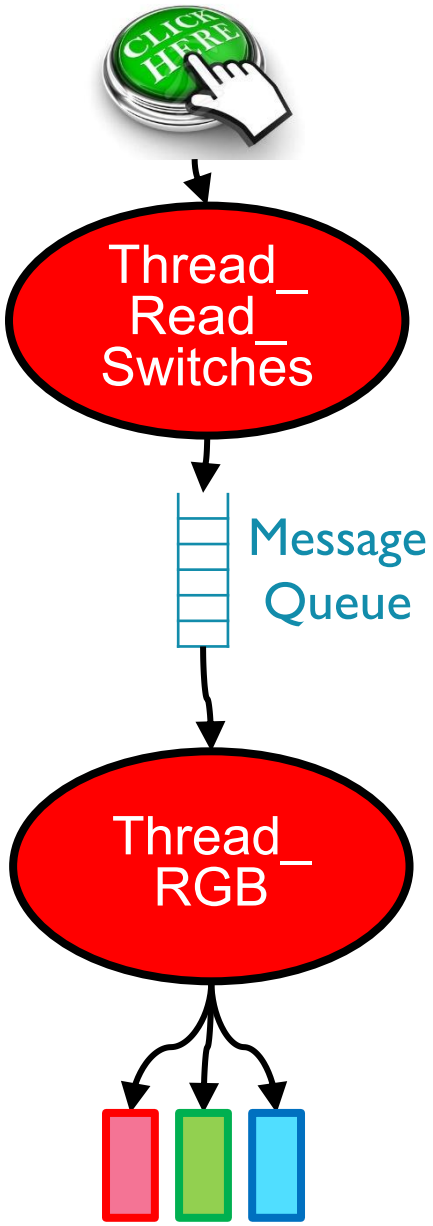
- osMessageQueueGetCapacity: maximum number of messages
- osMessageQueueGetSpace: number of available slots for messages
- osMessageQueueGetCount: number of queued messages
- osMessageQueueGetName
- osMessageQueueGetMsgSize: maximum message size in bytes
- osMessageQueueReset: re-initialize message queue
- osMessageQueueDelete: delete queue and free up storage

RTX5 Demo: Message Queues

- Thread_Read_Switches polls switches
  - If SW2 is pressed, then upon release send message to Thread_RGB
    - value: number of sequences
    - letter: *L* or *S*

```
typedef struct {
  char letter;
  int value;
} MY_MSG_T;
```

- Thread_RGB waits for message
  - Repeats RGB sequence based on value field in message
  - Speed up sequence if received *L* in letter



Demo Message Code

```
void Thread_Read_Switches(void  * arg) {
  int count = 0;
  MY_MSG_T msg;

  msg.value = 0;
  msg.letter = ' ';
  while (1) {
    osDelay(100);
    if (SWITCH_PRESSED(SW2_POS)) {
      count++;
      Control_RGB_LEDs(0, 1, 0);
      osDelay(g_RGB_delay/30);
      Control_RGB_LEDs(0, 0, 0);
    } else { // send message on release
      if (count > 0) {
        msg.value = count;
        if (count > 10)
          msg.letter = 'L';
        else
          msg.letter = 'S';
        osMessageQueuePut(switch_msgq_id,
          &msg, NULL, osWaitForever);
        count = 0;
      }
    }
  }
}
```

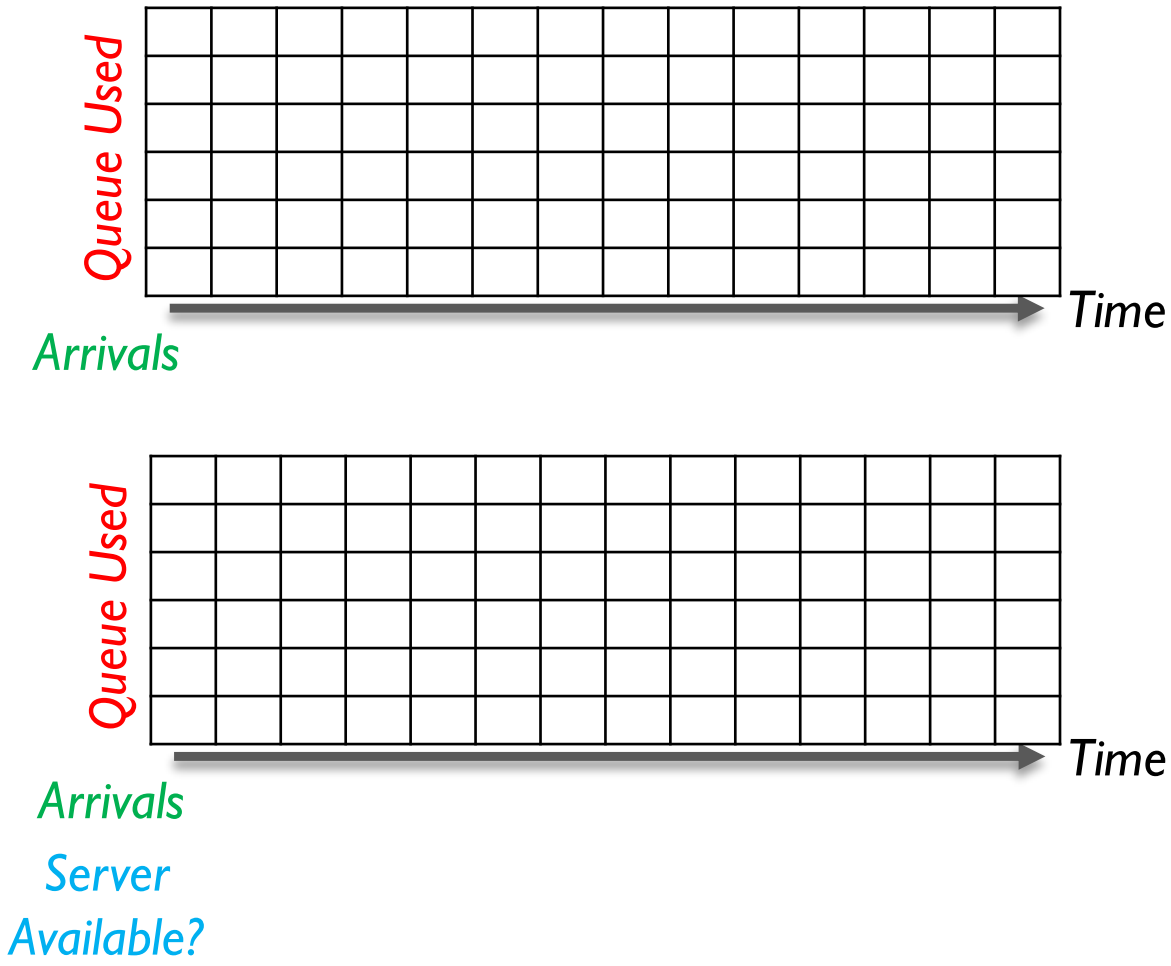
```
void Thread_RGB(void * arg) {
  osStatus_t result;
  MY_MSG_T dest_msg;
  uint32_t delay;

  while (1) {
    result = osMessageQueueGet(switch_msgq_id,
      &dest_msg, NULL, osWaitForever);
    if (result==osOK) {
      if (dest_msg.letter == 'L')
        delay = g_RGB_delay/5;
      else
        delay = g_RGB_delay;
      while (dest_msg.value-- > 0) { // Do RGB
        Control_RGB_LEDs(1, 0, 0);
        osDelay(delay);
        Control_RGB_LEDs(0, 1, 0);
        osDelay(delay);
        Control_RGB_LEDs(0, 0, 1);
        osDelay(delay);
      }
      Control_RGB_LEDs(0, 0, 0);
    }
  }
}
```

- Measure how long switch 2 is pressed
- Then send data to Thread_RGB in a message

How Long Can the Queue Get?

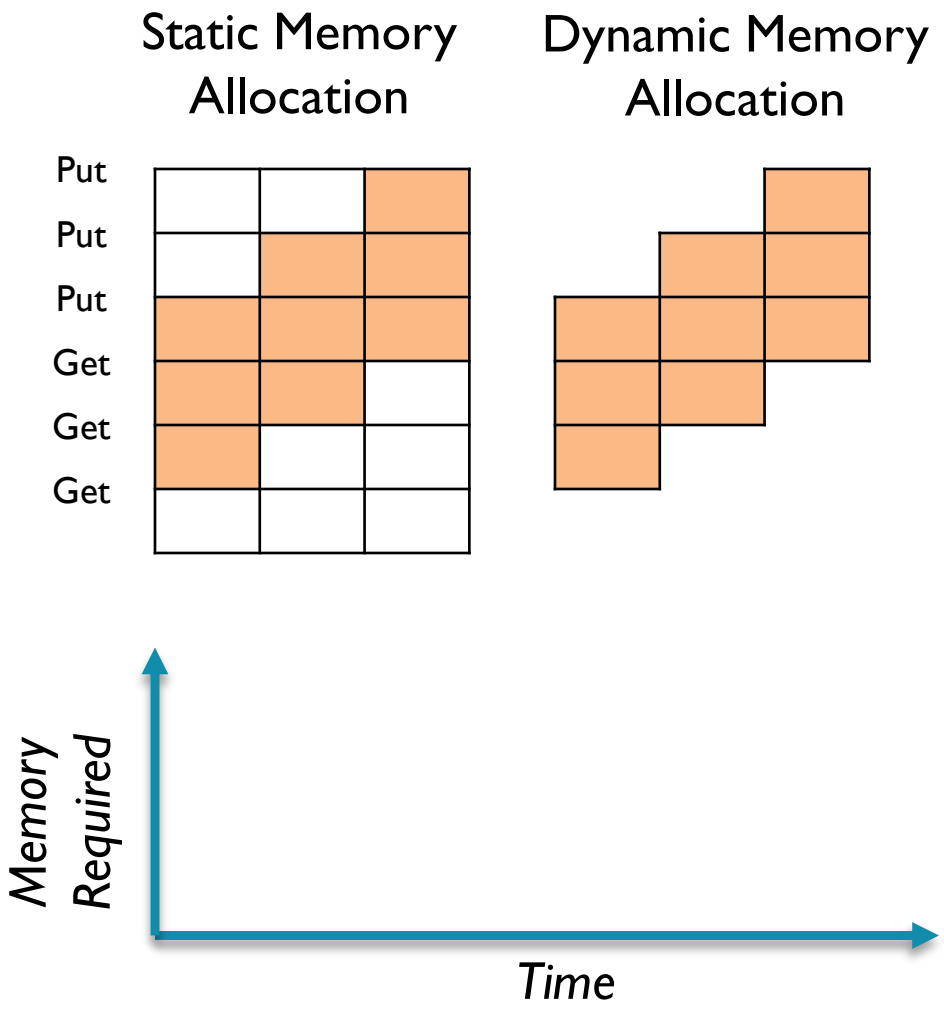
- It depends on…
  - Event arrival rate
  - Service time required per event
- Is server thread in system with prioritized, preemptive scheduling? Then consider…
  - Delay until service begins
  - Preemption during service
- How parameters are estimated (models)
  - Constant? Periodic? Sporadic?
  - Exact, typical, worst-case bounds…
- *Queueing theory* covers this in depth
  - https://queue-it.com/blog/queuing-theory/
  - https://en.wikipedia.org/wiki/Queueing_theory



Message Queues and Memory Allocation

Where is the message data held?

- How much memory is needed?
  - Message queue data needs up to msg_count * msg_size of space
- Two approaches for managing this memory
  - Static allocation: allocate enough memory for largest possible queue length
    - Always uses maximum memory. Too much?
    - Must make consumer thread more responsive to reduce maximum queue size needed
  - Dynamic allocation: ***put*** allocates, ***get*** deallocates
    - Slower, more complex, but less memory required (except in worst case)
    - Dynamic memory allocation will be discussed soon
- *CMSIS-RTOS2/RTX uses dynamic memory allocation*
  - *Keil_v5\ARM\PACK\ARM\CMSIS\<version>\CMSIS\RTOS2\RTX\Source\rtx_msgqueue.c*



Message Queues and Memory Allocation

Where is the message data held?

- How much memory is needed?
  - Message queue data needs up to msg_count * msg_size of space
- Two approaches for managing this memory
  - Static allocation: allocate enough memory for largest possible queue length
    - Always uses maximum memory. Too much?
    - Must make consumer thread more responsive to reduce maximum queue size needed
  - Dynamic allocation: ***put*** allocates, ***get*** deallocates
    - Slower, more complex, but less memory required (except in worst case)
    - Dynamic memory allocation will be discussed soon
- CMSIS-RTOS2/RTX uses dynamic memory allocation
  - Keil_v5\ARM\PACK\ARM\CMSIS\<version>\CMSIS\RTOS2\RTX\Source\rtx_msgqueue.c