Inter-Task Communication

Networks and Embedded Software

Module 6.2.2

by Wolfgang Neff

Inter-Task Communication

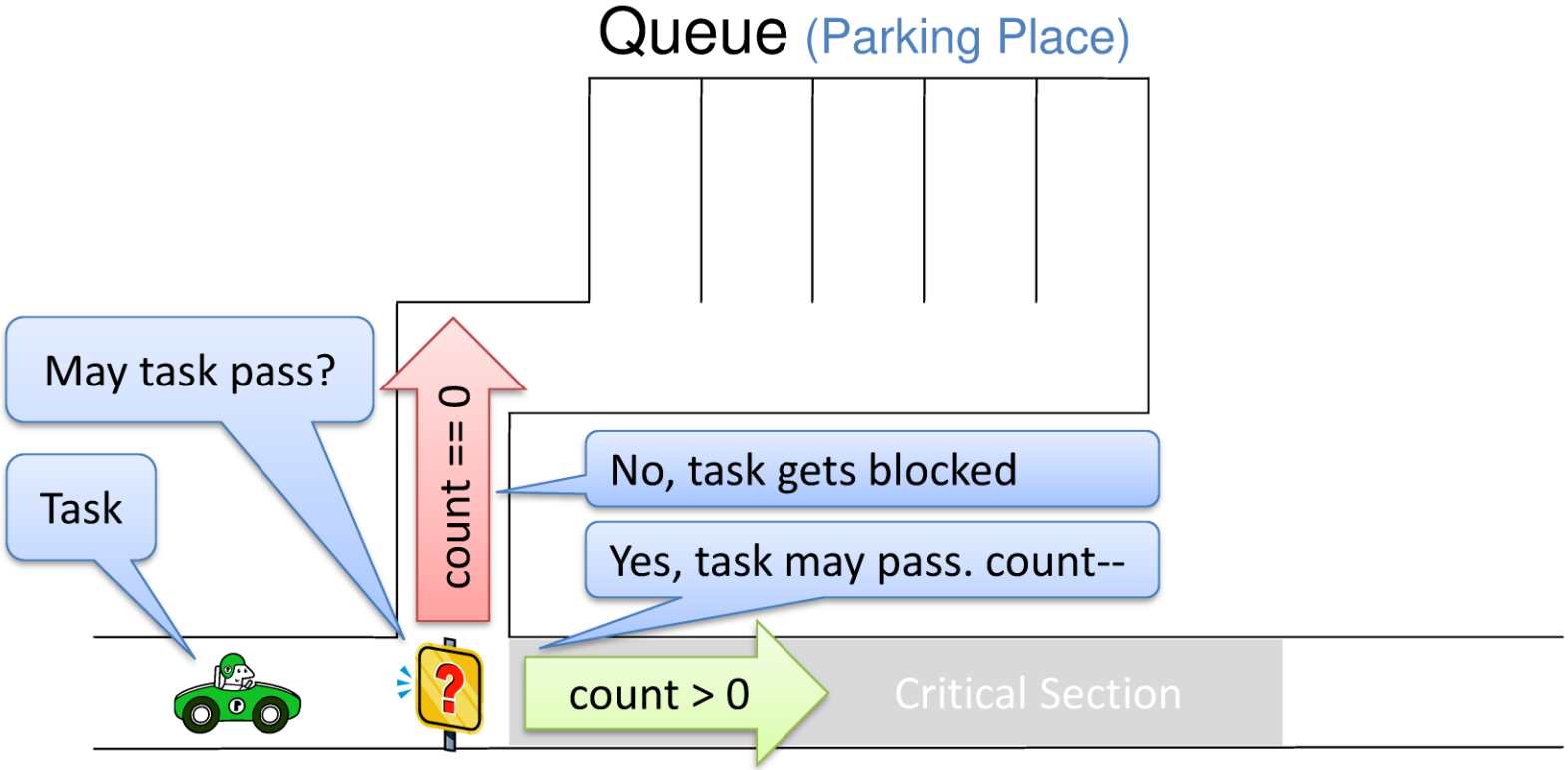
- Communication between tasks
  - Synchronisation of tasks
    - Binary semaphores
  - Management of common resources
    - Mutual exclusion
  - Data exchange between tasks
    - Shared memory
    - Message queues

Semaphores (1)

- Basics
  - Datatype
    - Counter for the resources
    - Queue for the blocked tasks
    - Operation for taking resources
    - Operation for giving resources back
  - Implementation
    - Atomic
      - No pre-emption may occur

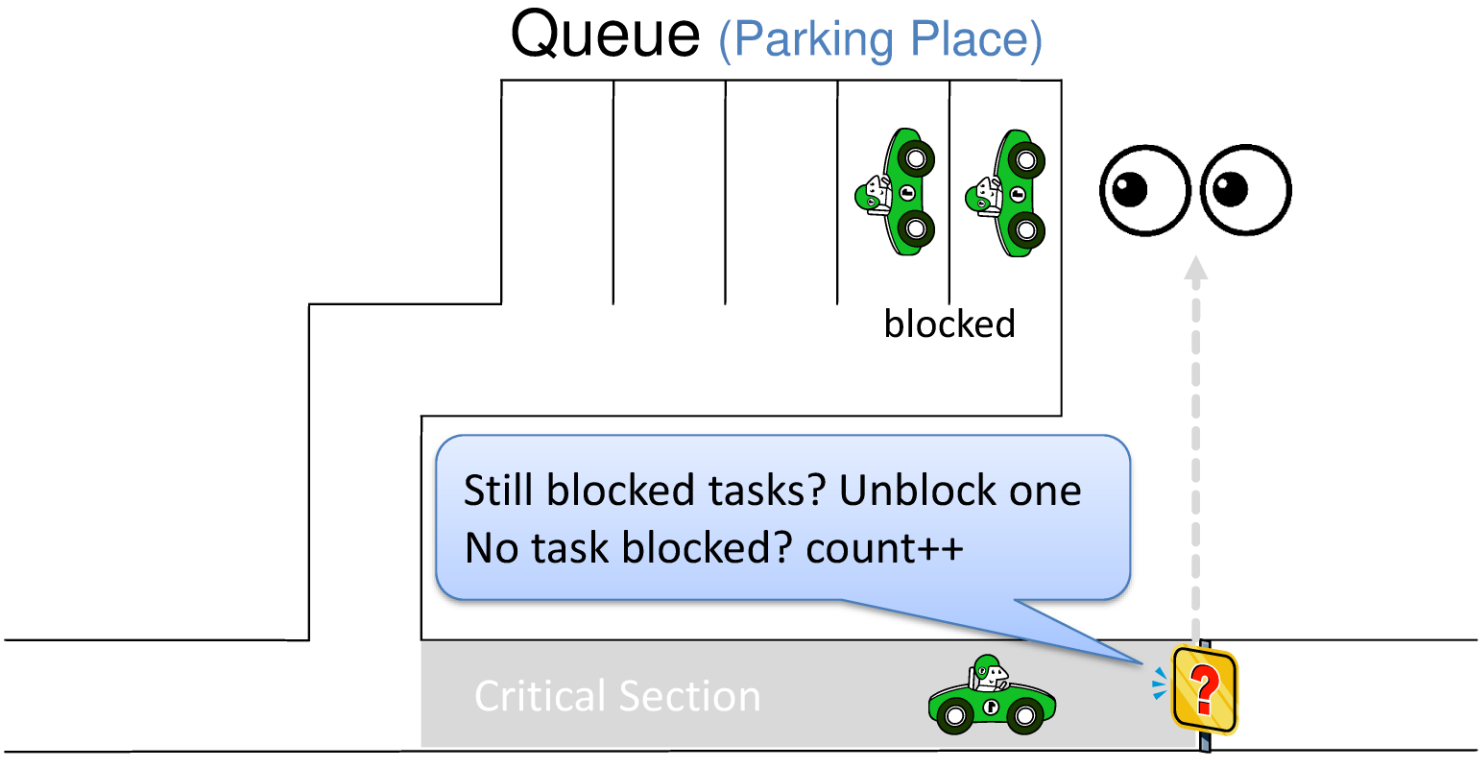
Semaphores (2)

- Operation Take



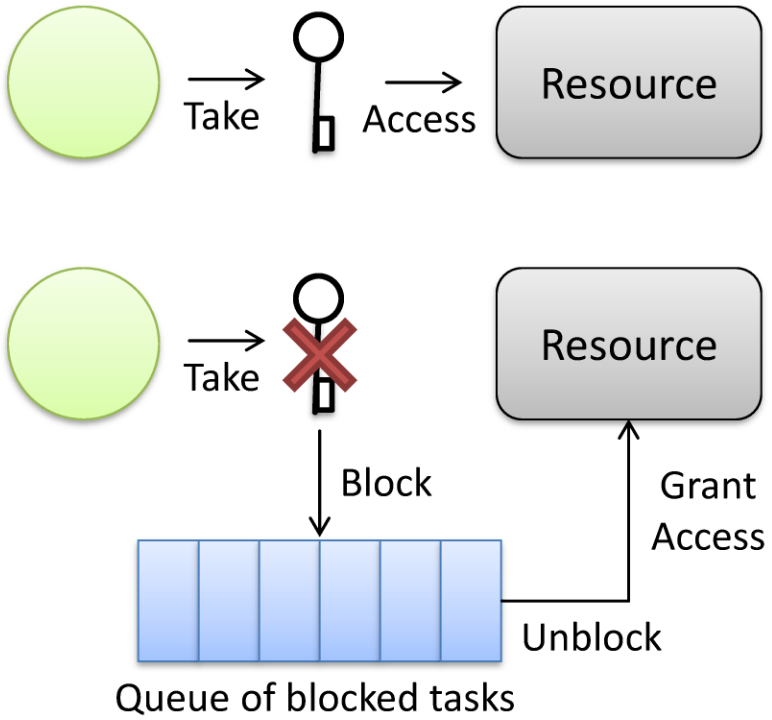
Semaphores (3)

- Operation Give



Semaphores (4)

- Mode of Operation
  - Resource free?
    - Yes
      - Grant access
      - Lock resource
    - No
      - Block task
      - Wait until free
      - Unblock task
      - Grant access

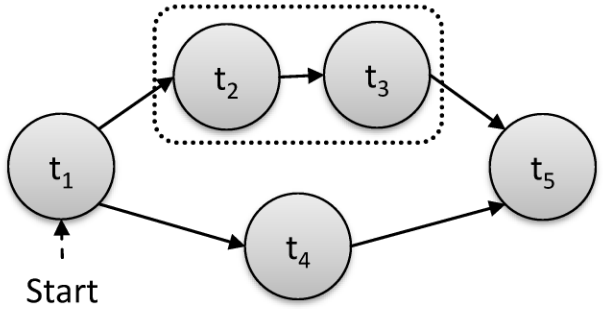


Synchronization (1)

- Task Dependencies
  - One task depends on another.
  - It must not start before the other has finished.
  - Representation
    - Task-dependency graph
  - Implementation
    - Binary semaphores
    - *Barriers*

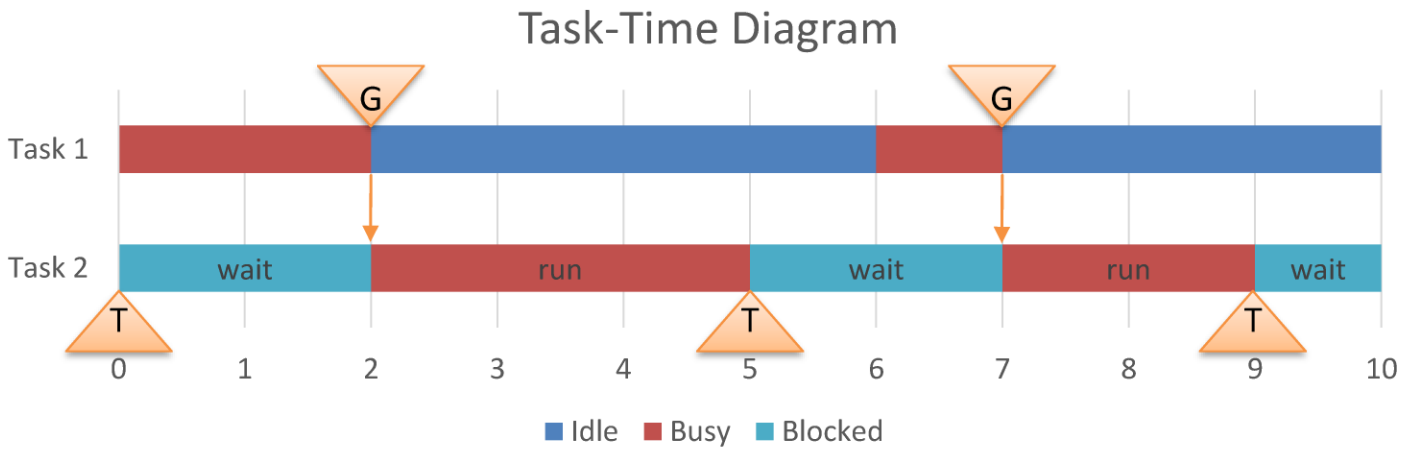
Synchronization (2)

- Dependency Graph (example)
  - Task t_1 starts at first.
  - Task t_2 and t_4 start when task t_1 has finished.
  - Task t_3 starts when task t_2 has finished.
  - Task t_2 and t_3 run sequentially.
    - Sequential tasks can be merged.
  - Task t_2 and t_4 run parallelly.
  - Task t_5 starts when task t_3 and t_4 have finished.



Synchronization (3)

- Binary Semaphores
  - Like token passing
  - One task gives the token
  - Another task takes the token

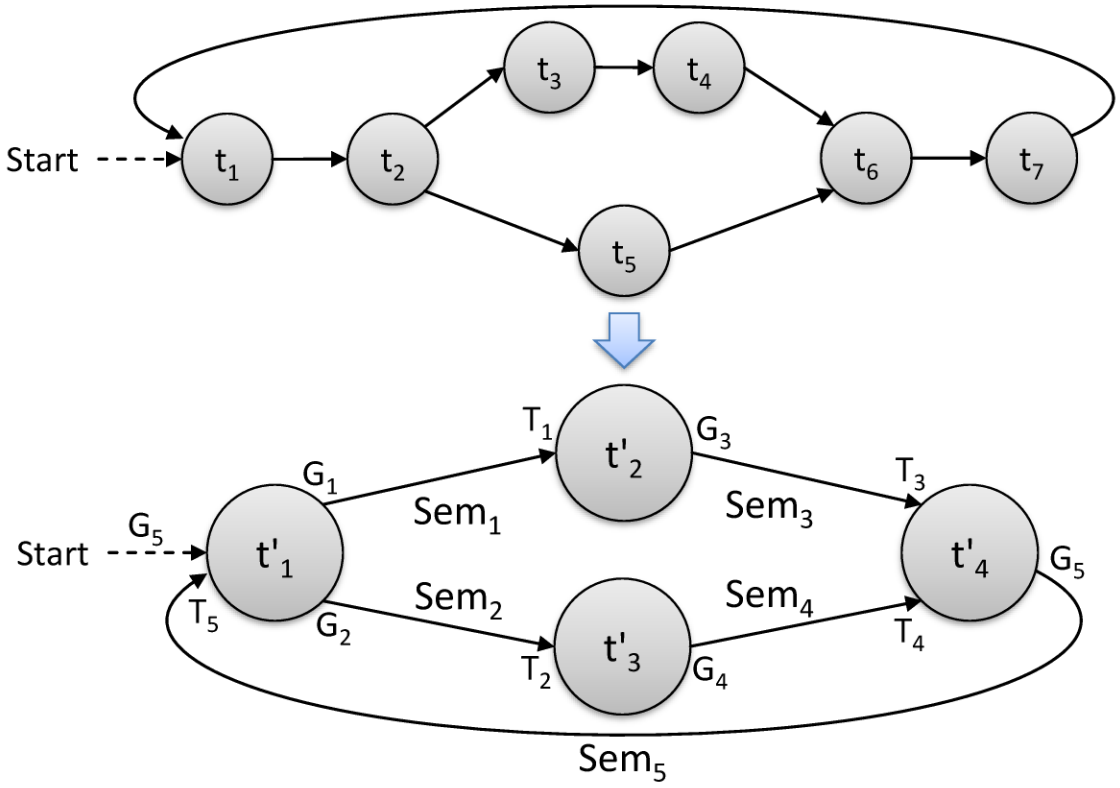


Synchronization (4)

- Best Practice
  - Start with a plenty of potential tasks
    - Tasks synchronize only at the start and at the end
  - Draw the dependency graph
  - Merge sequences of tasks
  - Handle the remaining edges (arrows)
    - Create one semaphore per edge
    - Every head stands for a take operation
    - Every tail stands for a give operation

Synchronization (5)

- Best Practice (example)



Synchronization (6)

- FreeRTOS Prototypes
  - Binary Semaphores
    - `#include <semphr.h>`
    - `SemaphoreHandle_t xSemaphoreCreateBinary(void);`
    - `xSemaphoreGive(SemaphoreHandle_t xSemaphore);`
    - ```
      xSemaphoreTake(
          SemaphoreHandle_t xSemaphore,
          TickType_t xTicksToWait
      );
      ```
    - Use *FromISR* versions if called from ISR
      - `xSemaphoreGiveFromISR(…);`
      - `xSemaphoreTakeFromISR(…);`
      ```

Synchronization (7)

- FreeRTOS Example
  - Binary Semaphores
    - Create Semaphore
      - `SemaphoreHandle_t xSemaphore;`
      - `xSemaphore = xSemaphoreCreateBinary();`
    - Use Case with Real-Time Aspect

```
if (xSemaphoreTake(xSemaphore, deadline)) {
    // Semaphore successfully taken.
}
else {
    // Semaphore not available within deadline
}
```

Synchronization (8)

- FreeRTOS Example (continued)
  - Binary Semaphores
    - Task that gives the semaphore

```
void vTaskA (void* pvParameters)
{
    …
    while (1) {
        …
        xSemaphoreGive(xSemaphore);
    }
}
```

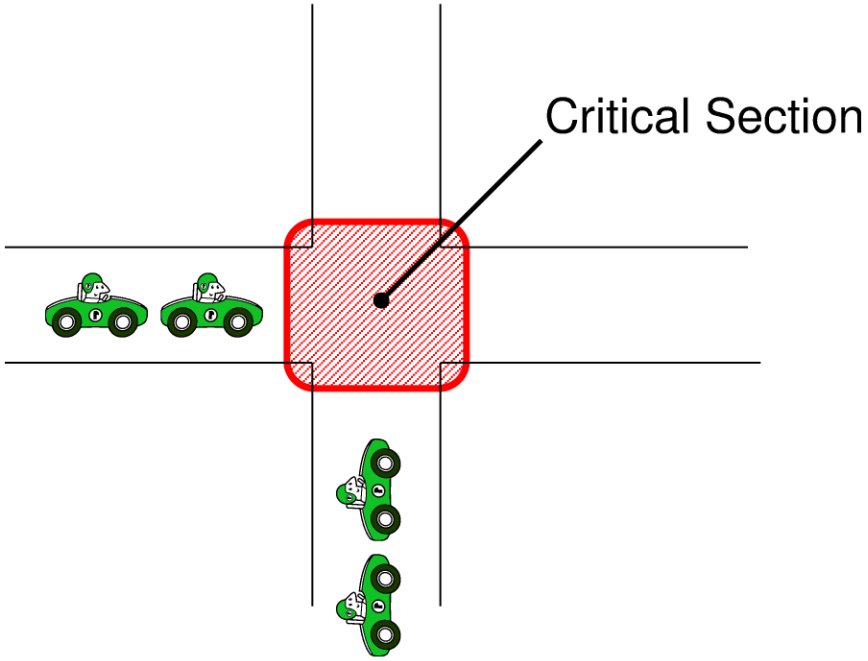
Synchronization (9)

- FreeRTOS Example (finished)
  - Binary Semaphores
    - Task that takes the semaphore

```
void vTaskB (void* pvParameters)
{
    …
    // If task is starting task
    xSemaphoreGive(xSemaphore);
    while (1) {
        xSemaphoreTake(xSemaphore, portMAX_DELAY);
        …
    }
}
```

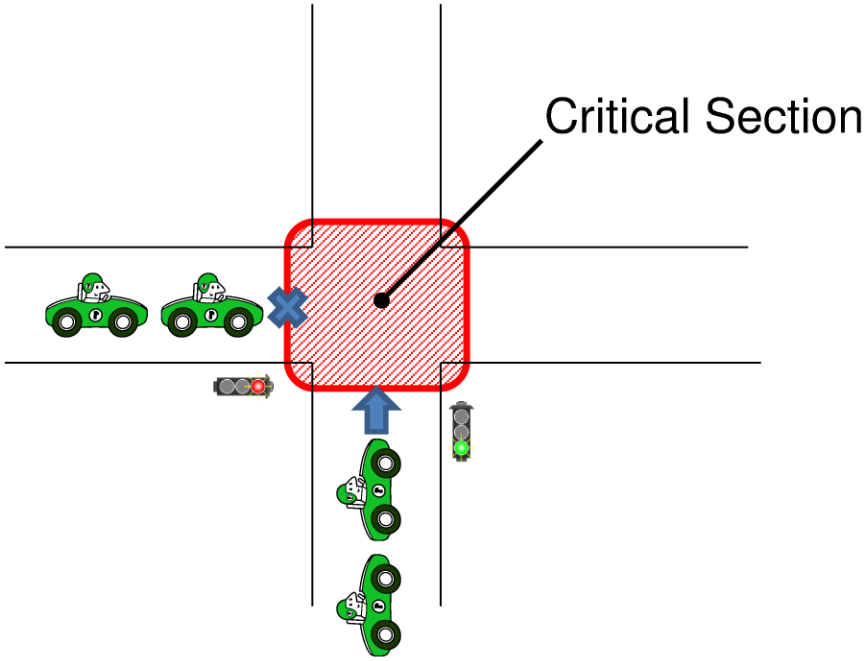
Mutual Exclusion (1)

- Critical Section
  - Two tasks compete for a common resource.



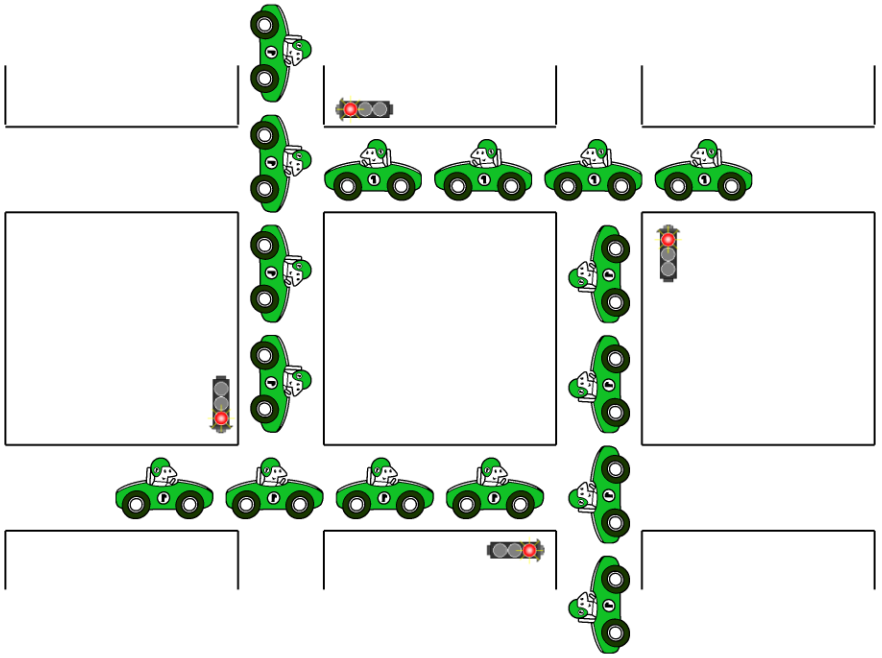
Mutual Exclusion (2)

- Mutual Exclusion
  - Just one task may enter the critical section.



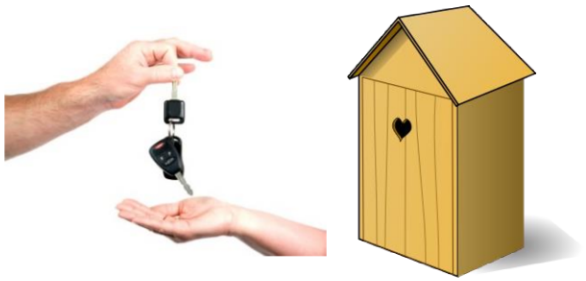
Mutual Exclusion (3)

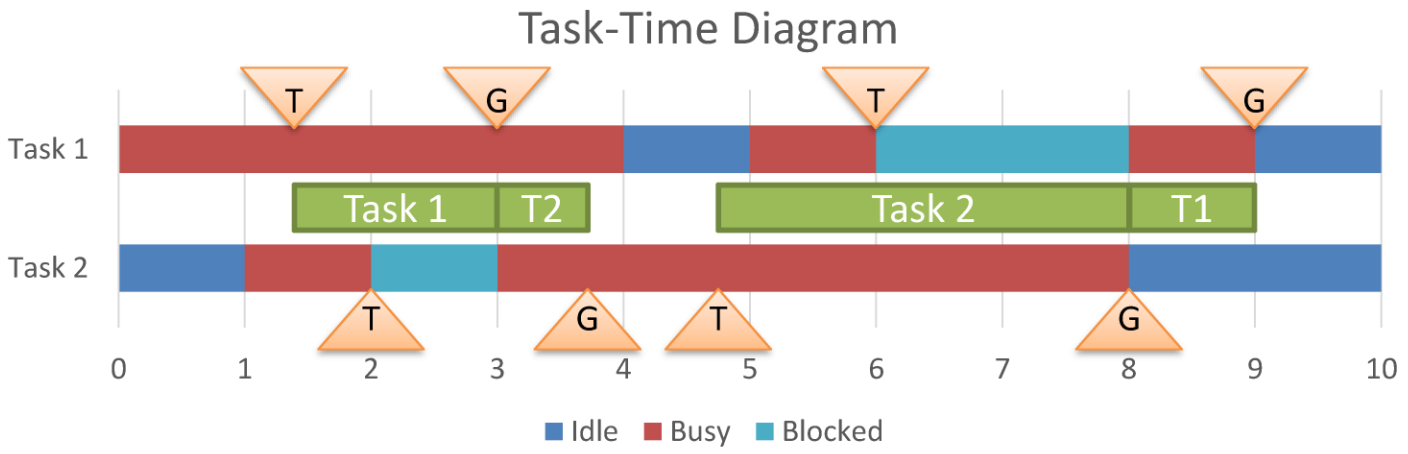
- Deadlocks
  - Mutual exclusions may cause deadlocks.



Mutual Exclusion (4)

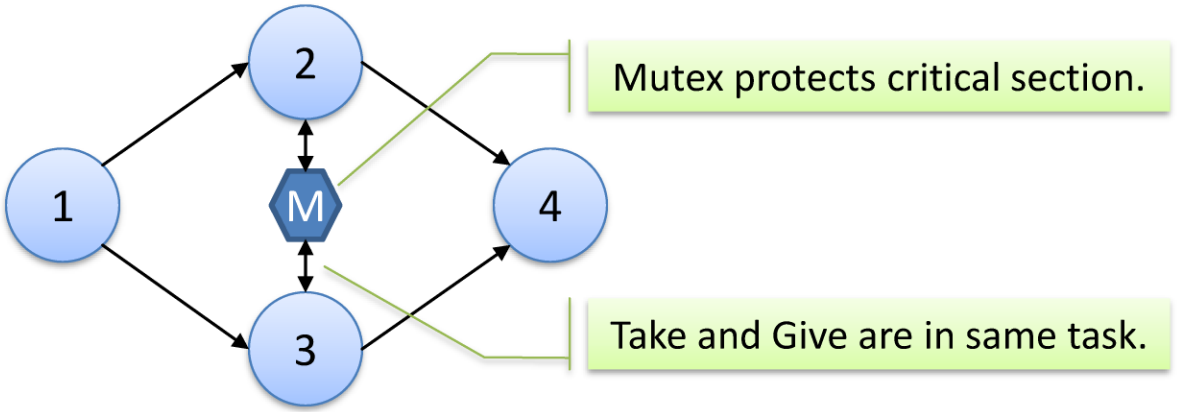
- Mode of Operation
  - One Task takes the mutex.
  - Other tasks get blocked.
  - Task gives mutex back.





Mutual Exclusion (5)

- Best Practice
  - Draw the dependency graph.
  - Identify common resources.
  - Use one mutex per critical section.
  - Give and take are in the same task.



Mutual Exclusion (6)

- FreeRTOS Example
  - Mutexes
    - Create a mutex (prototype)
      - `SemaphoreHandle_t xSemaphoreCreateMutex(void);`
    - Use the mutex (use case with real-time aspect)

```
if (xSemaphoreTake(xMutex, deadline)) {
    // In critical section, now.
    …
    xSemaphoreGive(xMutex);
}
else {
    // Missed deadline to enter critical section.
}
```

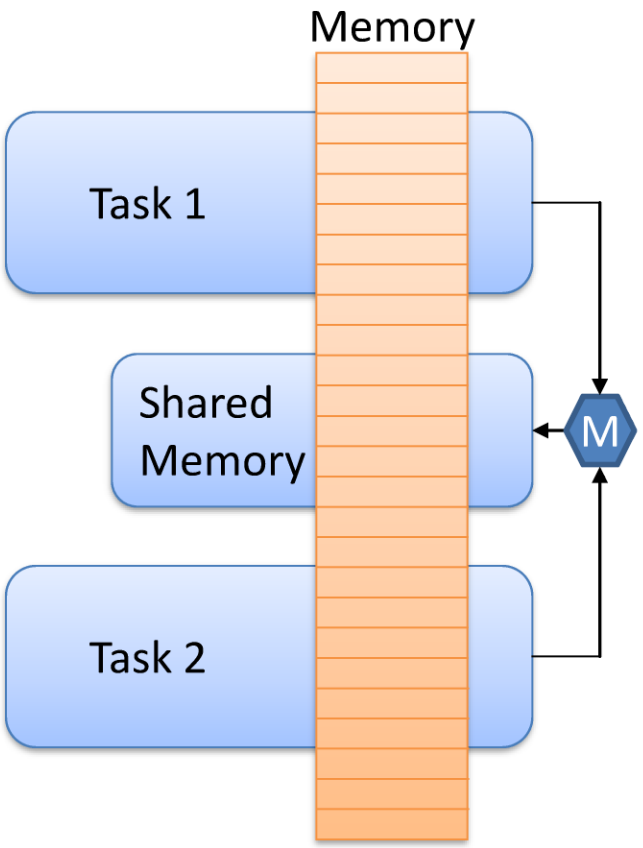
Mutual Exclusion (7)

- FreeRTOS Example (continued)
  - Mutexes
    - Task with a critical section

```
void vTask(void* pvParameters)
{
    …
    while (1) {
        …
        xSemaphoreTake(xMutex, portMAX_DELAY);
        … // In critical section, now.
        xSemaphoreGive(xMutex);
        …
    }
}
```

Data Exchange (1)

- Shared Memory
  - Often global variables
  - Accessed by serval tasks
  - Common Resource
    - Race conditions
    - Avoided by mutex
  - Very fast



Data Exchange (2)

- FreeRTOS Example
  - Shared Memory
    - Declare shared memory

```
volatile uint8_t buffer;
volatile uint8_t buffer_full;
```

    - Task with write access to shared memory

```
…
// Data races avoided by a mutex.
xSemaphoreTake(xMutex, portMAX_DELAY);
buffer = 5;
buffer_full = 1;
xSemaphoreGive(xMutex);
…
```

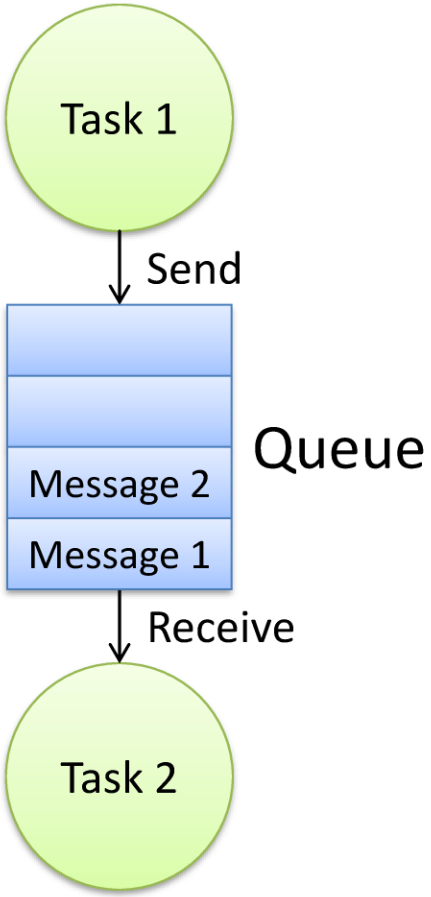
Data Exchange (3)

- FreeRTOS Example (continued)
  - Shared Memory
    - Task with read access to shared memory

```
…
// Data races avoided by a mutex.
xSemaphoreTake(xMutex, portMAX_DELAY);
data = buffer;
buffer_full = 0;
xSemaphoreGive(xMutex);
…
```

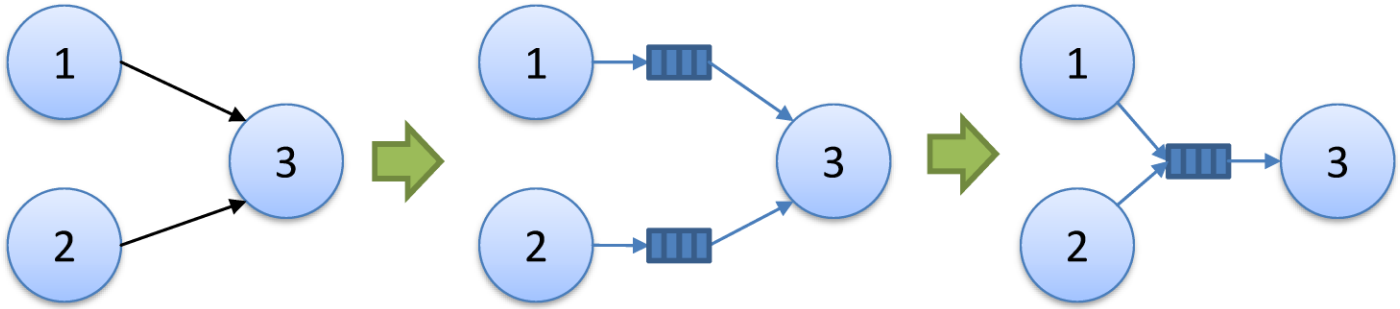
Data Exchange (4)

- Message Queues
  - Queue is a fixed-size FIFO
    - First In, First Out
    - All boxes the same size
  - One or more tasks write
    - Send data
  - One or more tasks read
    - Receive data
  - Multitasking safe



Data Exchange (5)

- Best Practice
  - Draw the dependency graph
  - Identify data exchange
  - Use message queue for data exchange
  - Merge queues if possible



Data Exchange (6)

- FreeRTOS Prototypes
  - Message queues
    - `#include <queue.h>`
    - ```
      QueueHandle_t xQueueCreate(
          UBaseType_t uxQueueLength,
          UBaseType_t uxItemSize
      );
      ```
    - ```
      BaseType_t xQueueSend(
          QueueHandle_t xQueue,
          const void* pvItemToQueue,
          TickType_t xTicksToWait
      );
      ```

Data Exchange (7)

- FreeRTOS Prototypes (continued)
  - Message queues
    - ```
      BaseType_t xQueueReceive(
          QueueHandle_t xQueue,
          void* pvBuffer,
          TickType_t xTicksToWait
      );
      ```
    - Use *FromISR* versions if called from ISR
      - `xQueueSendFromISR(…);`
      - `xQueueReceiveFromISR(…);`
      ```

Data Exchange (8)

- Simple FreeRTOS Example
  - Message queues
    - Create Queue
      - `QueueHandle_t xQueue;`
      - `xQueue = xQueueCreate(4, sizeof(int));`
    - Send a message
      - `int iMessageTaskA = 10;`
      - `xQueueSend(xQueue, &iMessageTaskA, portMAX_DELAY);`
    - Receive a message
      - `int iMessageTaskB;`
      - `xQueueReceive(xQueue, &iMessageTaskB, portMAX_DELAY);`

Data Exchange (9)

- Advanced FreeRTOS Example
  - Message queues

```
#define CAPACITY 8
typedef struct {
      int a;
      int b;
} Message_t;
QueueHandle_t xQueue
xQueue = xQueueCreate(CAPACITY, sizeof(Message_t));
Message_t xMessage;    // = { 10, 11 };
xMessage.a = 10;       // ^^^^^^^^^^^^^
xMessage.b = 11;       //  Alternative
xQueueSend(xQueue, &xMessage, portMAX_DELAY);
```