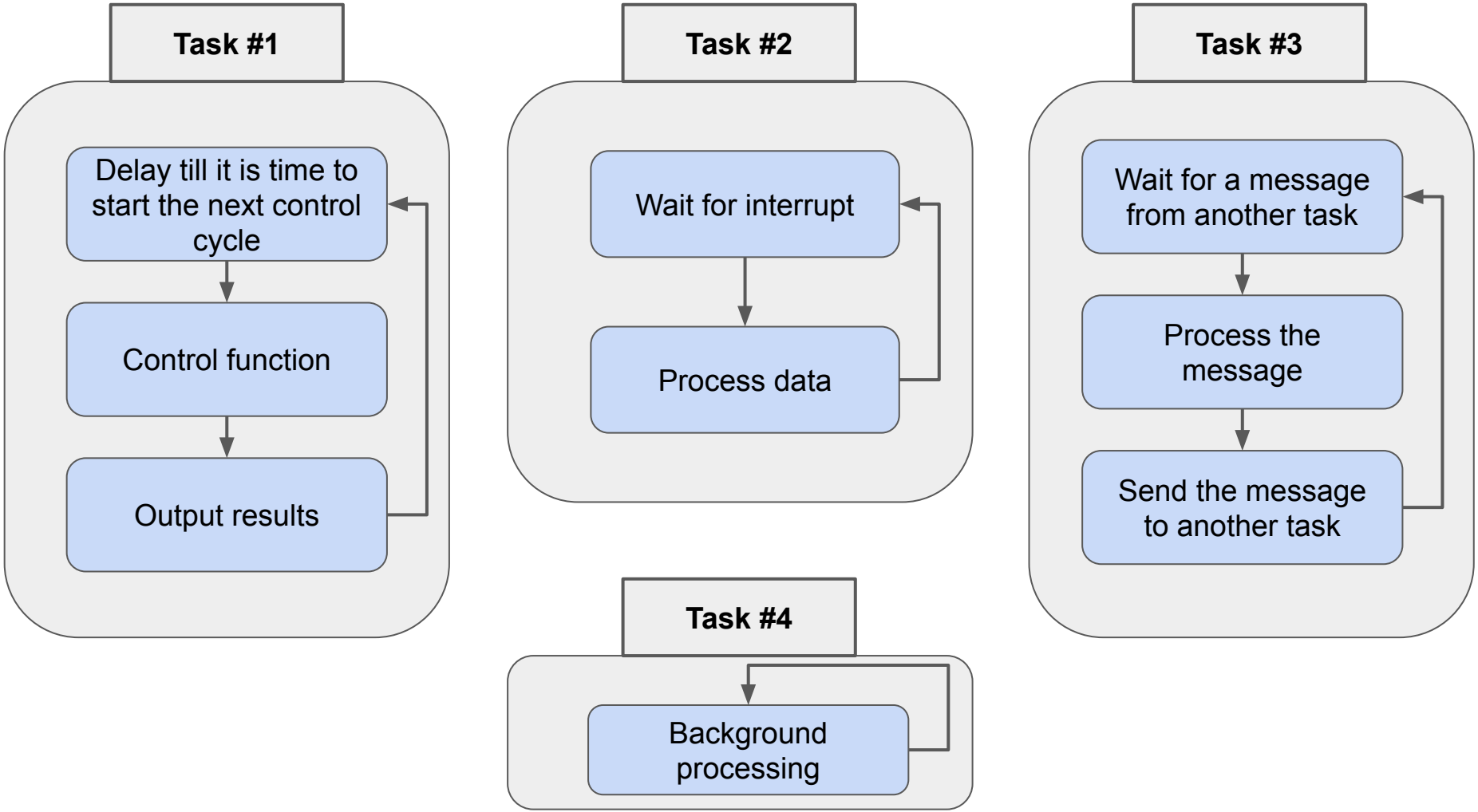
Программирование микроконтроллеров STM32

FreeRTOS and IPC

МФТИ, 2019


Task #1
Delay till it is time to start the next control cycle
Control function
Output results
Task #2
Wait for interrupt
Process data
Task #3
Wait for a message from another task
Process the message
Send the message to another task
Task #4
Background processing

FreeRTOS. Межпроцессорное взаимодействие

- Мультизадачные системы подвержены большому количеству ошибок
- Любая задача может быть прервана в любой момент
- Используемый ресурс может оказаться в неконсистентном состоянии

- Нотификация (Task Notifier)
- Мьютексы (Mutex)
- Очереди (Queue)

FreeRTOS. Примеры

Example #1

1. Запускается задача 1, которая выводит “Привет мир”
2. Задача 1 вытеснена задачей 2, причем уже было выведено слово “Привет”
3. Задача 2 пишет “STM32” и переходит в заблокированное состояние
4. Задача 1 продолжает выполнение и выводит “мир”
5. Таким образом, на дисплее оказывается “Привет STM32 мир”

FreeRTOS. Примеры

Example #2 (RMW операции) - чтобы изменить состояние порта, необходимо прочитать его (R), изменить некоторые его биты (M) и сохранить значение (W).

1. Задача 1 загружает состояние GPIOA->ODR в регистр
2. Задача 1 вытеснена Задачей во время операции изменения
3. Задача 2 изменяет состояние GPIOA->ODR и уходит в заблокированное состояние
4. Задача 1 записывает измененное значение в GPIOA->ODR, не учитывая изменения Задачи 2

Весь процесс RMW не является атомарным!

FreeRTOS. Критическая секция

Критическая секция — участок исполняемого кода программы, в котором производится доступ к общему ресурсу (данным или устройству), который не должен быть одновременно использован более чем одним потоком выполнения

- `void taskENTER_CRITICAL(void)`
- `void taskEXIT_CRITICAL(void)`
- `UBaseType_t taskENTER_CRITICAL_FROM_ISR(void)`
- `void taskEXIT_CRITICAL_FROM_ISR(UBaseType_t uxSavedInterruptStatus)`

Очень “грубый” способ разделения доступа к ресурсу, так как выключаются все прерывания.

FreeRTOS. Отключение планировщика

Данный способ не выключает все прерывания, а лишь блокирует планировщик задач.

- `void vTaskSuspendAll();`
- `BaseType_t xTaskResumeAll(void);`

FreeRTOS. Семафоры

Семафор (англ. semaphore) — примитив синхронизации работы процессов и потоков, в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение и уменьшение значения на единицу, при этом операция уменьшения для нулевого значения счетчика является блокирующейся.

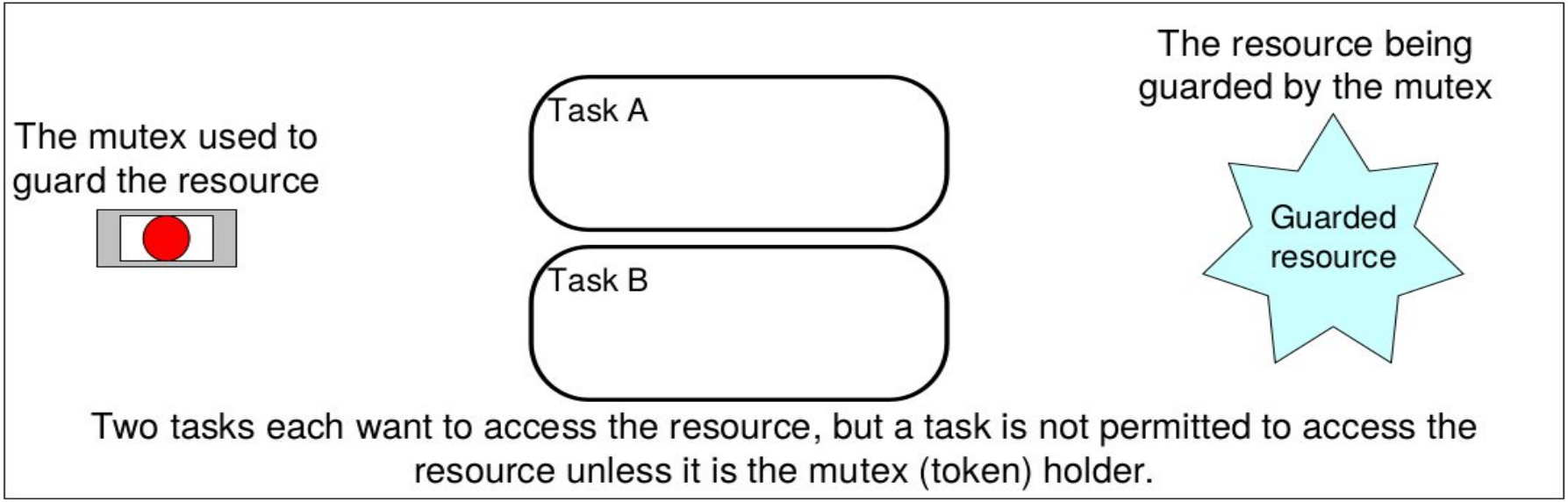
https://en.wikipedia.org/wiki/Dining_philosophers_problem

FreeRTOS. Мьютексы

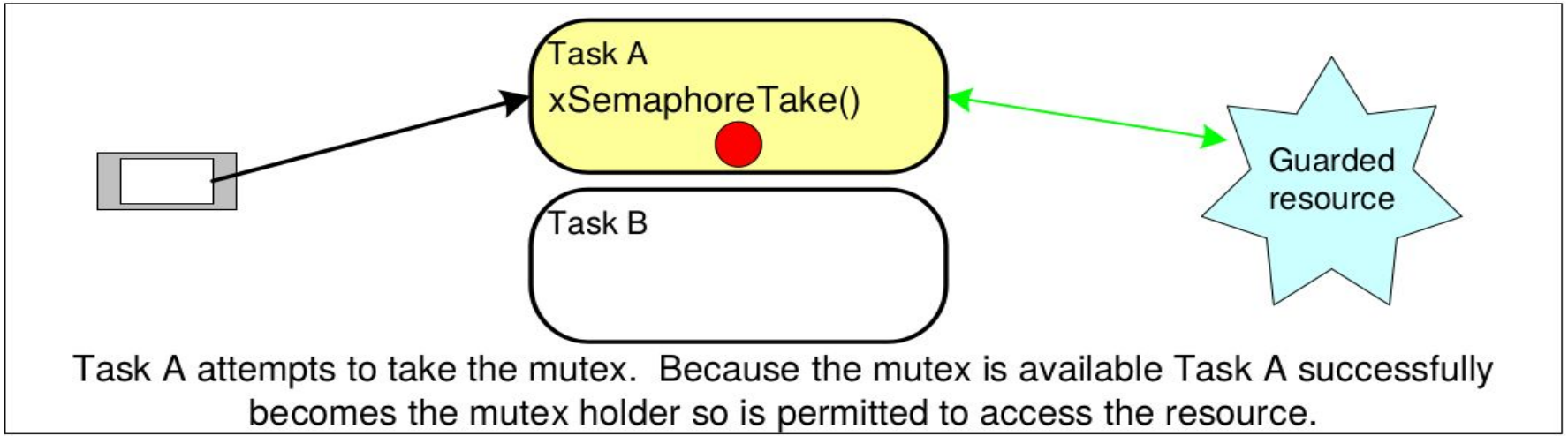
Мьютексы - упрощённая реализация семафоров, аналогичная двоичным семафорам с тем отличием, что мьютексы должны отпускаться тем же процессом или потоком, который осуществляет их захват.

Мютексы != Двоичный семафор

FreeRTOS. Мьютексы

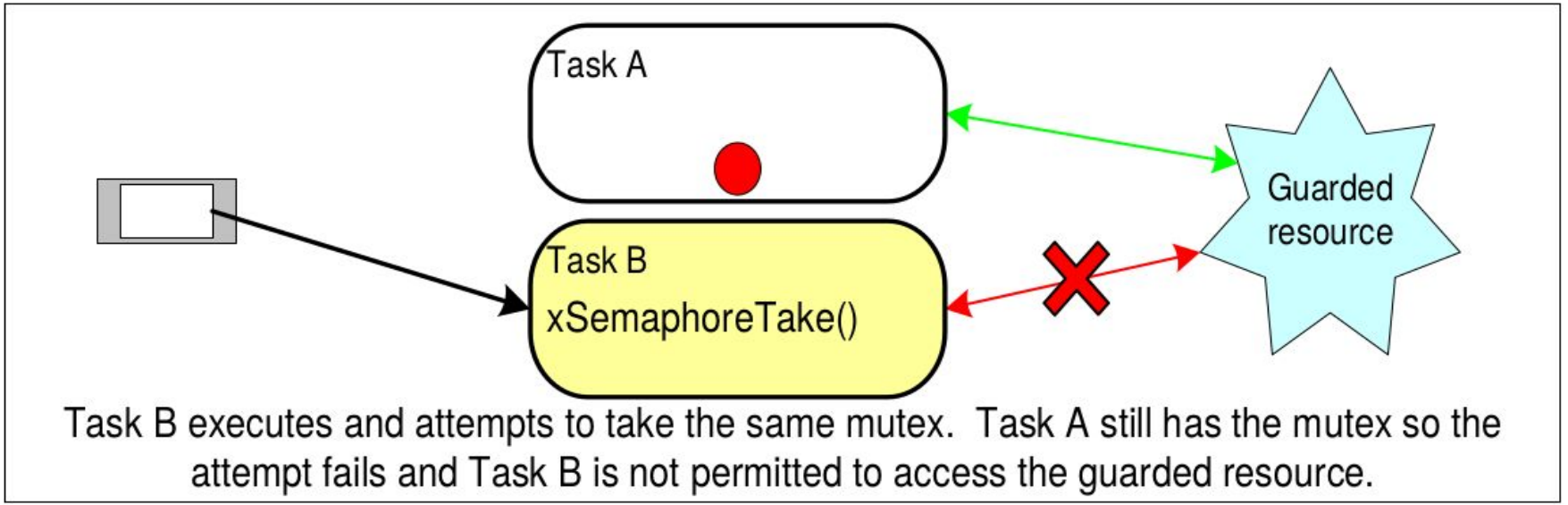


FreeRTOS. Мьютексы



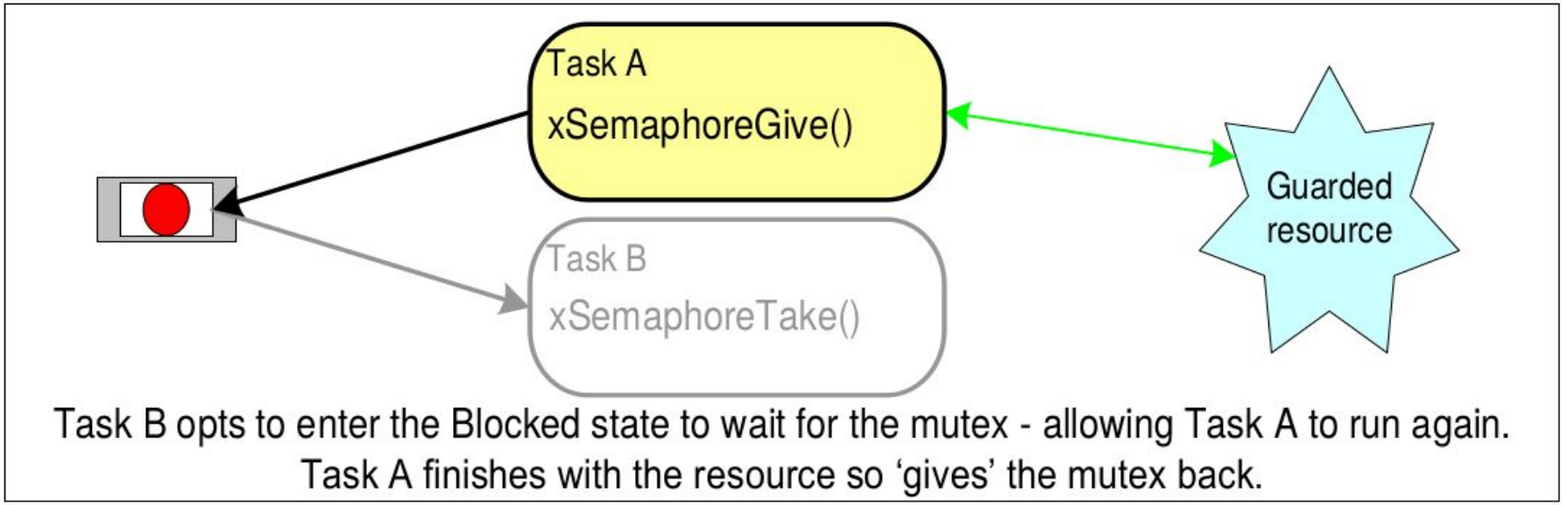
Task A attempts to take the mutex. Because the mutex is available Task A successfully becomes the mutex holder so is permitted to access the resource.

FreeRTOS. Мьютексы



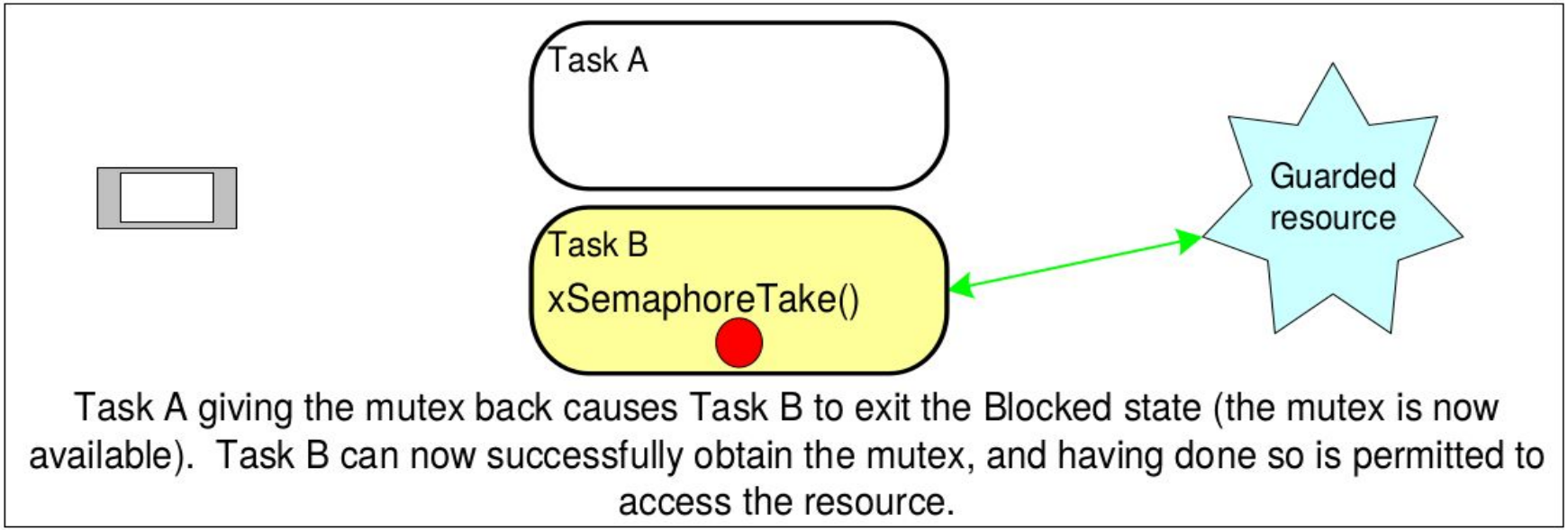
Task B executes and attempts to take the same mutex. Task A still has the mutex so the attempt fails and Task B is not permitted to access the guarded resource.

FreeRTOS. Мьютексы



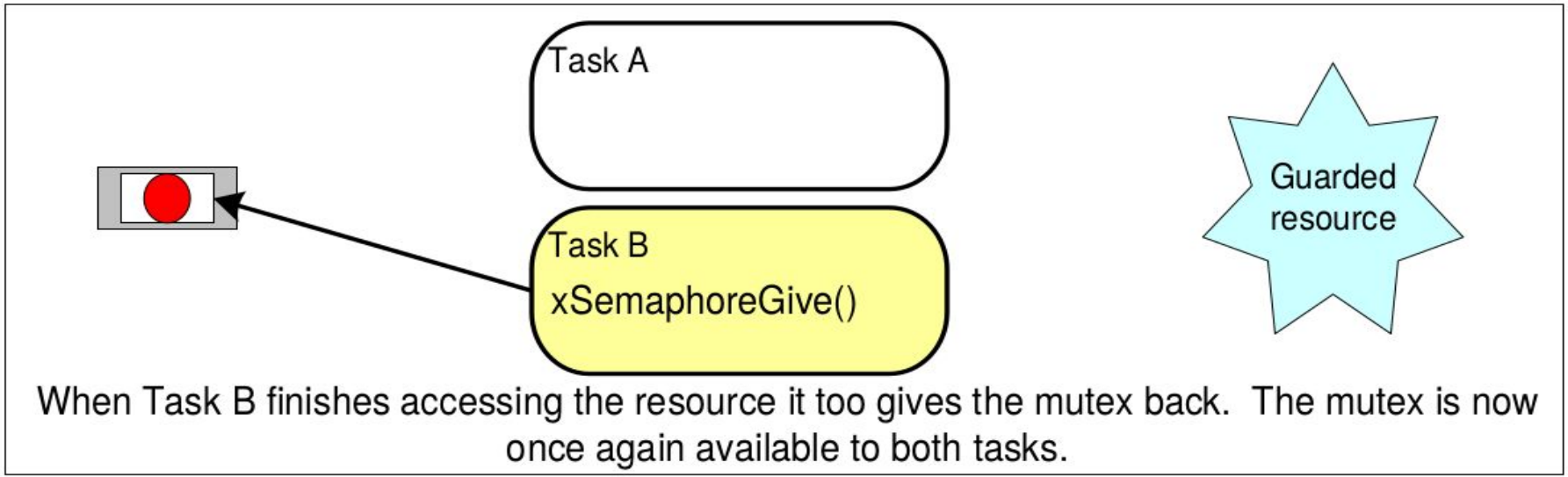
Task B opts to enter the Blocked state to wait for the mutex - allowing Task A to run again.
Task A finishes with the resource so ‘gives’ the mutex back.

FreeRTOS. Мьютексы



Task A giving the mutex back causes Task B to exit the Blocked state (the mutex is now available). Task B can now successfully obtain the mutex, and having done so is permitted to access the resource.

FreeRTOS. Мьютексы



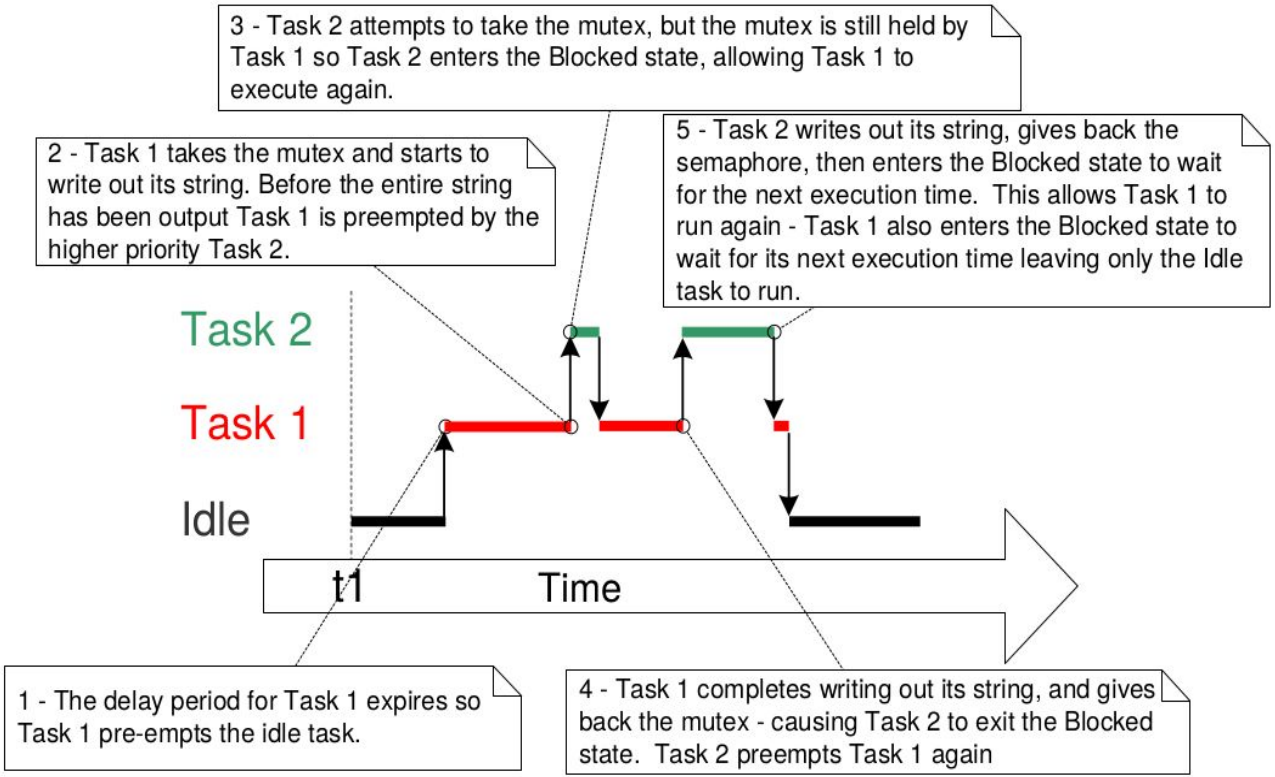
When Task B finishes accessing the resource it too gives the mutex back. The mutex is now once again available to both tasks.

FreeRTOS. Мьютексы. API

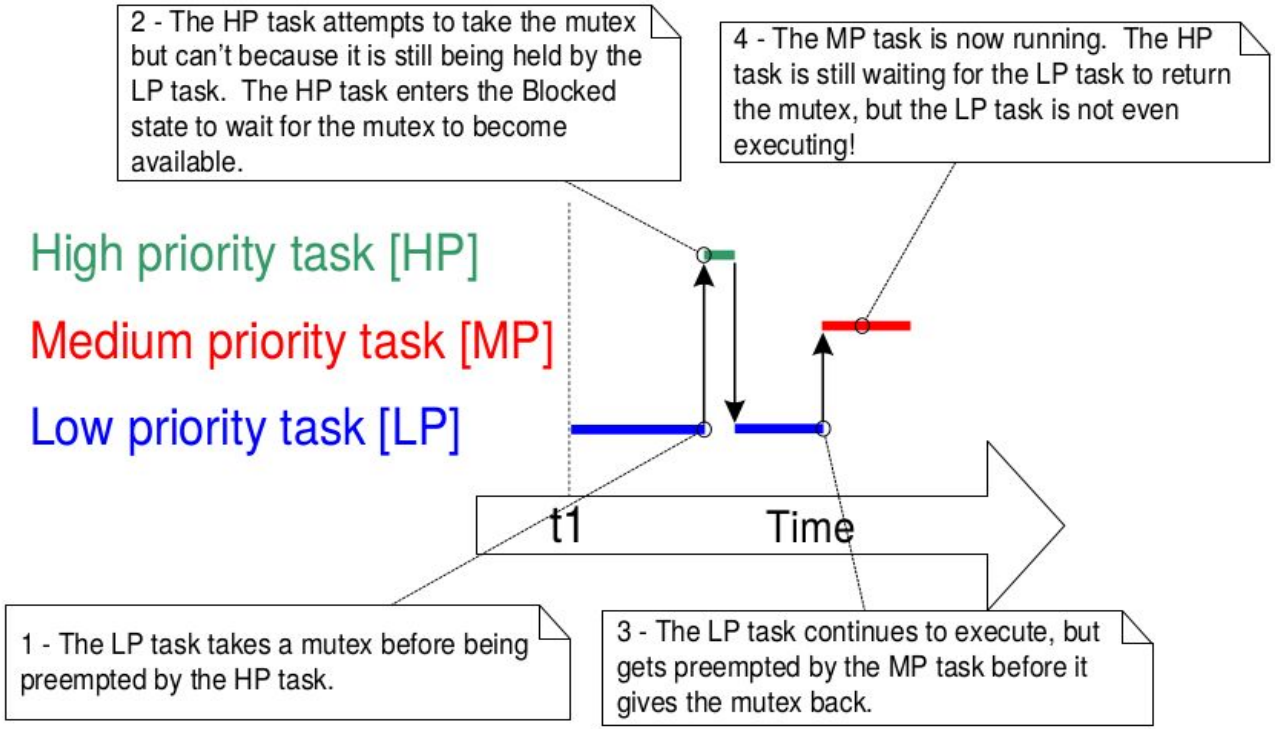
- `SemaphoreHandle_t xSemaphoreCreateMutex(void)`
- `xSemaphoreGive(SemaphoreHandle_t xMutex)`
- `xSemaphoreTake(SemaphoreHandle_t xMutex, uint32_t delay)`

Если delay = portMAX_DELAY, то блокирующий вызов

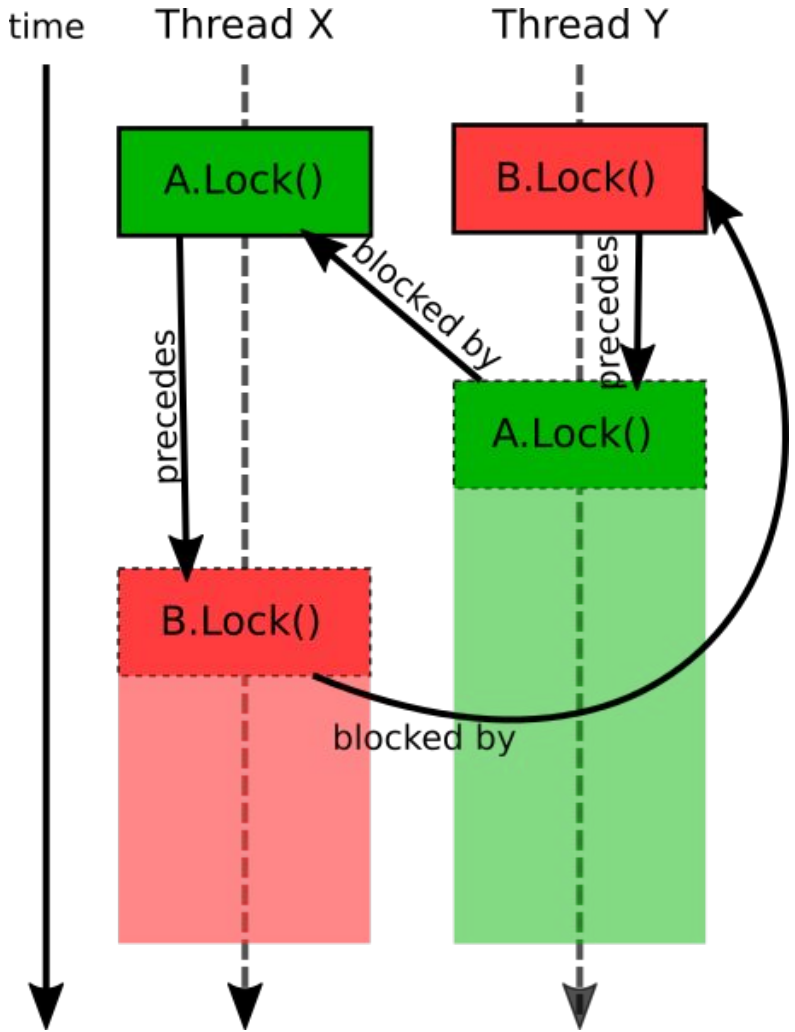
FreeRTOS. Мьютексы. Пример выполнения



FreeRTOS. Мьютексы. Инверсия приоритетов



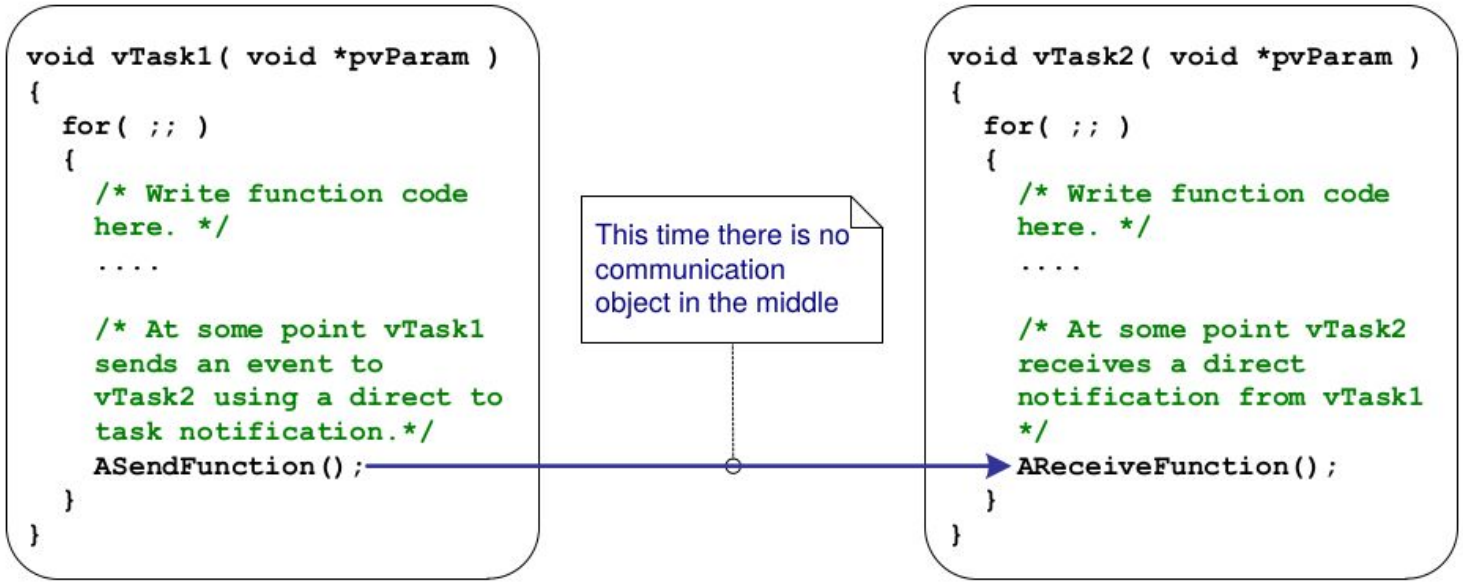
FreeRTOS.
Мьютексы. Дедлок



FreeRTOS. Нотификатор

- Позволяет синхронизироваться между задачами и прерываниями без использования дополнительно общего объекта
- Легковесный примитив
- Использование меньшего количества RAM
- Но не используем в следующих сценариях:
  - Отправка нотификации в прерывание
  - Прием происходит в нескольких задачах
  - Буфферизирование запросов

FreeRTOS. Нотификатор



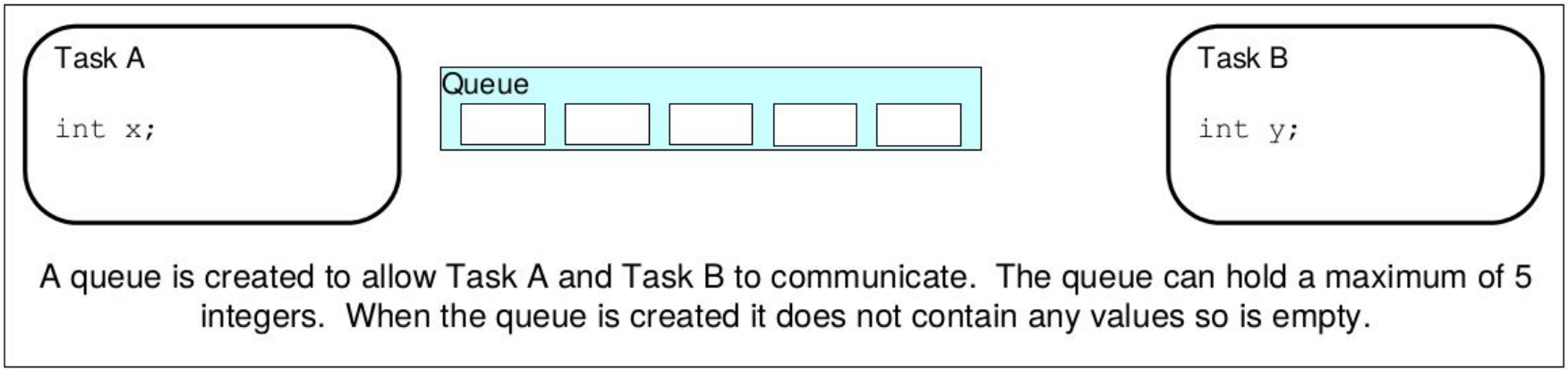
FreeRTOS. Нотификатор. API

- `TaskHandle_t xTaskGetCurrentTaskHandle()`
- `xTaskNotifyGive(TaskHandle_t task_handler)`
- `ulTaskNotifyTake(BaseType_t xClearCountOnExit, TickType_t xTicksToWait)`
- `vTaskNotifyGiveFromISR(TaskHandle_t xTaskToNotify, BaseType_t *pxHigherPriorityTaskWoken)`

FreeRTOS. Очереди

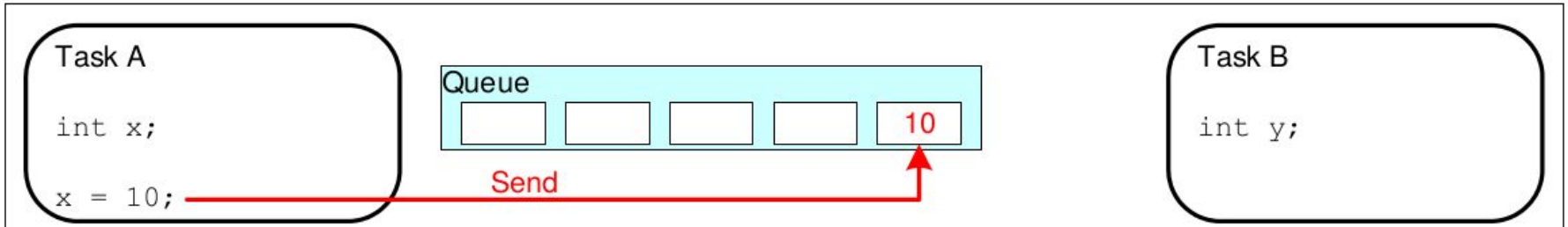
- Содержит в себе фиксированное количество элементов фиксированного размера
- Тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (FIFO)
- Взаимодействие может быть задача-задача, прерывание-задача, задача-прерывание
- Могут блокировать задачи

FreeRTOS. Очереди



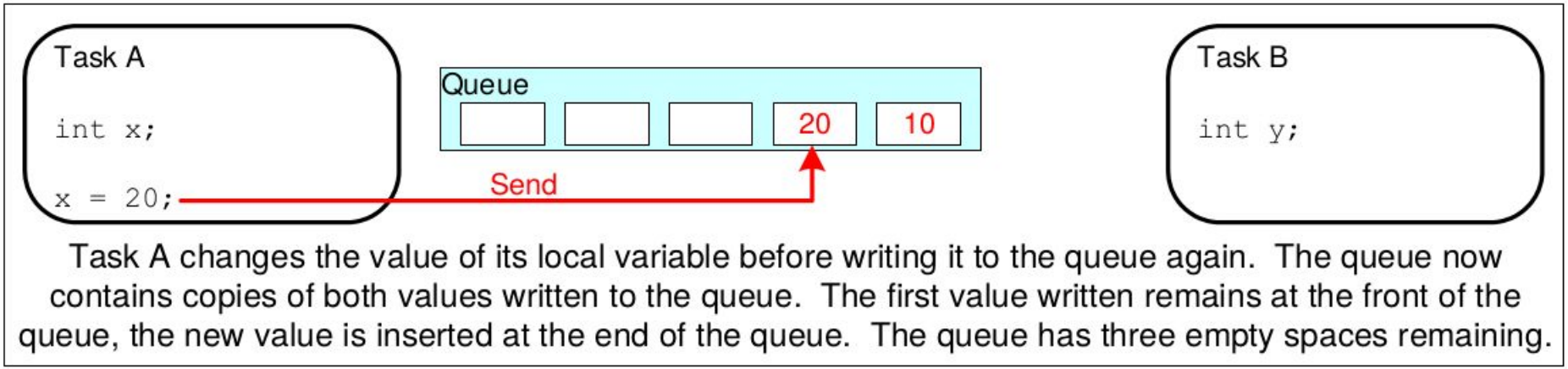
A queue is created to allow Task A and Task B to communicate. The queue can hold a maximum of 5 integers. When the queue is created it does not contain any values so is empty.

FreeRTOS. Очереди



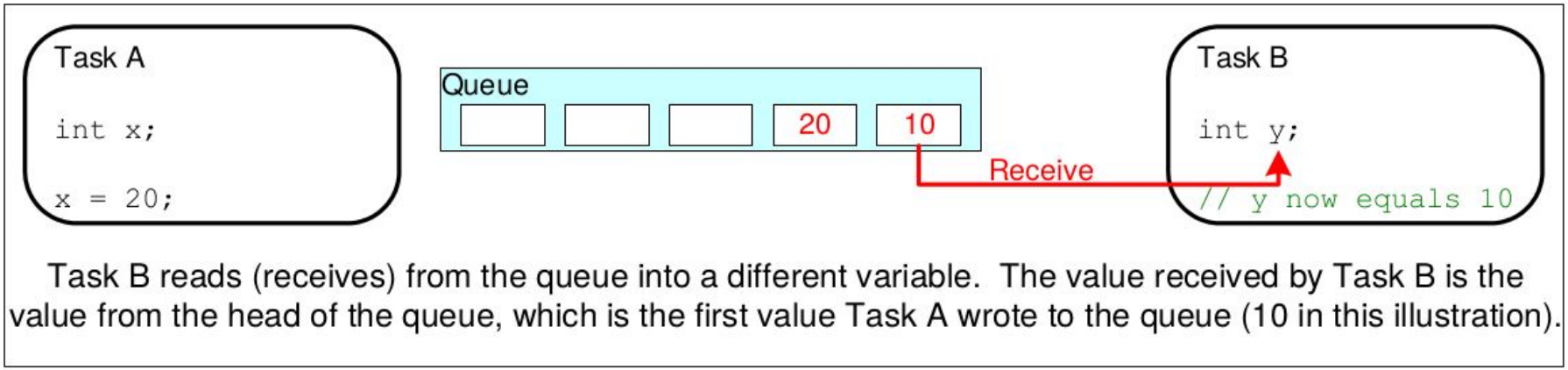
Task A writes (sends) the value of a local variable to the back of the queue. As the queue was previously empty the value written is now the only item in the queue, and is therefore both the value at the back of the queue and the value at the front of the queue.

FreeRTOS. Очереди



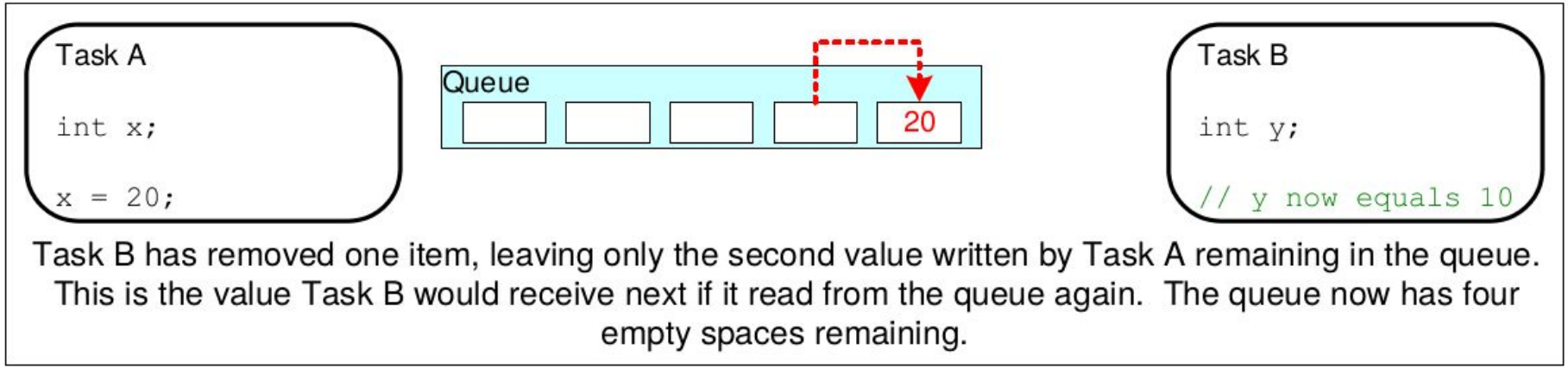
Task A changes the value of its local variable before writing it to the queue again. The queue now contains copies of both values written to the queue. The first value written remains at the front of the queue, the new value is inserted at the end of the queue. The queue has three empty spaces remaining.

FreeRTOS. Очереди



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).

FreeRTOS. Очереди



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.

FreeRTOS. Очереди. Политика копирования

- **Добавление путем копирования объекта (FreeRTOS)**
- Добавление по ссылке

FreeRTOS. Очереди. Возможности

- Can be accessed by multiple tasks
- Blocks on queue reads
- Might be blocked on write

FreeRTOS. Очереди. API

- *xQueueCreate*(UBaseType_t uxQueueLength, UBaseType_t uxItemSize)
- *xQueueSendToBack*(QueueHandle_t xQueue,const void * pvItemToQueue,TickType_t xTicksToWait)
- *xQueueSendToBackFromISR*()
- *xQueueSendToFront*()
- *xQueueSendToFrontFromISR*()
- *xQueueReceive*(QueueHandle_t xQueue,void * const pvBuffer, TickType_t xTicksToWait) (received object is removed from queue)
- *xQueueReceiveFromISR*()

Репозиторий

https://github.com/edosedgar/stm32f0_ARM