UNIVERSIDAD
COMPLUTENSE
MADRID

RTOS. ESP-IDF

Tasks, communication and synchronization

IoT Node Architecture

(Real Time-) Operating Systems

- What is the role of an OS?
- What is an RTOS?
- Operating system as...
  - Service provider
  - Shared hardware ressources manager
    - CPU
    - Memory
    - Storage
    - Other devices

By HikingArtist.com

OS manages / provides....

- Tasks (~ threads)
  - Task scheduling: share (multiple) CPU across tasks
  - Multiple task scheduling algorithms available
- Task communication and synchronization
  - Task may cooperate to perform a work: they must be able to cooperate
  - And we need to synchronize those tasks (stablish sync. Points...)
- Memory
  - Memory is a shared resource: every task needs its “own” memory
  - What is *free* memory? OS must keep track of it
  - Should Task A be able to read/write memory from Task B?
- I/O devices
  - Provide an abstraction to access devices
- Storage and file systems
  - Provide abstractions to use storage devices (files, folders...)

Are all the OS the same?

- Which Operating Systems do you know? (use the chat!)

Traditional OS

- Windows, Linux, Mac OS, IRIX
- Android, iOS....

- Do you think we can use any of them in our ESP32 device? Why?

Embeded RTOS

- A looooot of them
  - RIOT
  - Contiki
  - Tiny OS
  - chibiOS
  - RTEMS
  - RTLinux
  - Free RTOS
  - Micrium
  - eCos
  - Windows 10 IoT
  - QNX
  - VxWorkrs
  - Integrity (Greenhills)
  - Nucleus (Mentor Graphics)
  - ARMmbed
  - TI RTOS
  - ….

RTOS: Real-Time OS

- Not every RTOS is really for real-time applications
  - But still different from “traditional” ones

	RTOS	GPOS
Scheduling	Critical task first scheduling	*throughput* and/or interactivity based
	Predictability and fast response time	Latency not bounded and response time is not critical
	interruptible kernel	*non-interruptible kernel*
HW ressources	It mus require just a few ressources (memory, CPU…)	Allocates a lot of ressources
	Less implemented services	Many services (and more complete)

Choose the right OS...

- Cost / Licence
  - Free?
  - King Midas licences
  - Tech support
- Check our HW limitations
- Power saving module
  - Many OS allow to manage the power saving futures
- Task protection
  - Does the OS build a MMU? Do we need it?
- Is there a BSP (*Board Support Package*) for our SoC / Board?
- API
- Documentation

http://www.smxrtos.com/articles/How%20to%20Pick%20an%20RTOS.pdf
http://www.embeddeddeveloper.com/documents/mentorgraphics_selecting_an_operating_system_for_embedded_applications.pdf
http://download.dedicated-systems.com/
https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

Some links

- http://www.differencebetween.net/technology/difference-between-rtos-and-os/
- http://www.circuitstoday.com/gpos-versus-rtos-for-an-embedded-system
- https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems
- http://www.smxrtos.com/articles/How%20to%20Pick%20an%20RTOS.pdf
- http://www.embeddeddeveloper.com/documents/mentorgraphics_selecting_an_operating_system_for_embedded_applications.pdf

TASKS in ESP-IDF

- Similar to threads in Linux
  - Single memory space for them all
    - Do they share global variables?
    - What about local variables?
  - Fast switching time
- No protection mechanism
  - There is no Virtual Memory to isolate processes and protect memory

Task creation

- FreeRTOS defines **`xTaskCreate(....)`**
  - When bboting the system, several tasks are automatically created
  - One of them finally calls `app_main()`
- ESP32 has 2 cores (PRO y APP)
  - Defines **`xTaskCreatePinnedToCore(...)`** fix the core where the task must run
  - https://docs.espressif.com/projects/esp-idf/en/stable/api-reference/system/freertos.html

Example

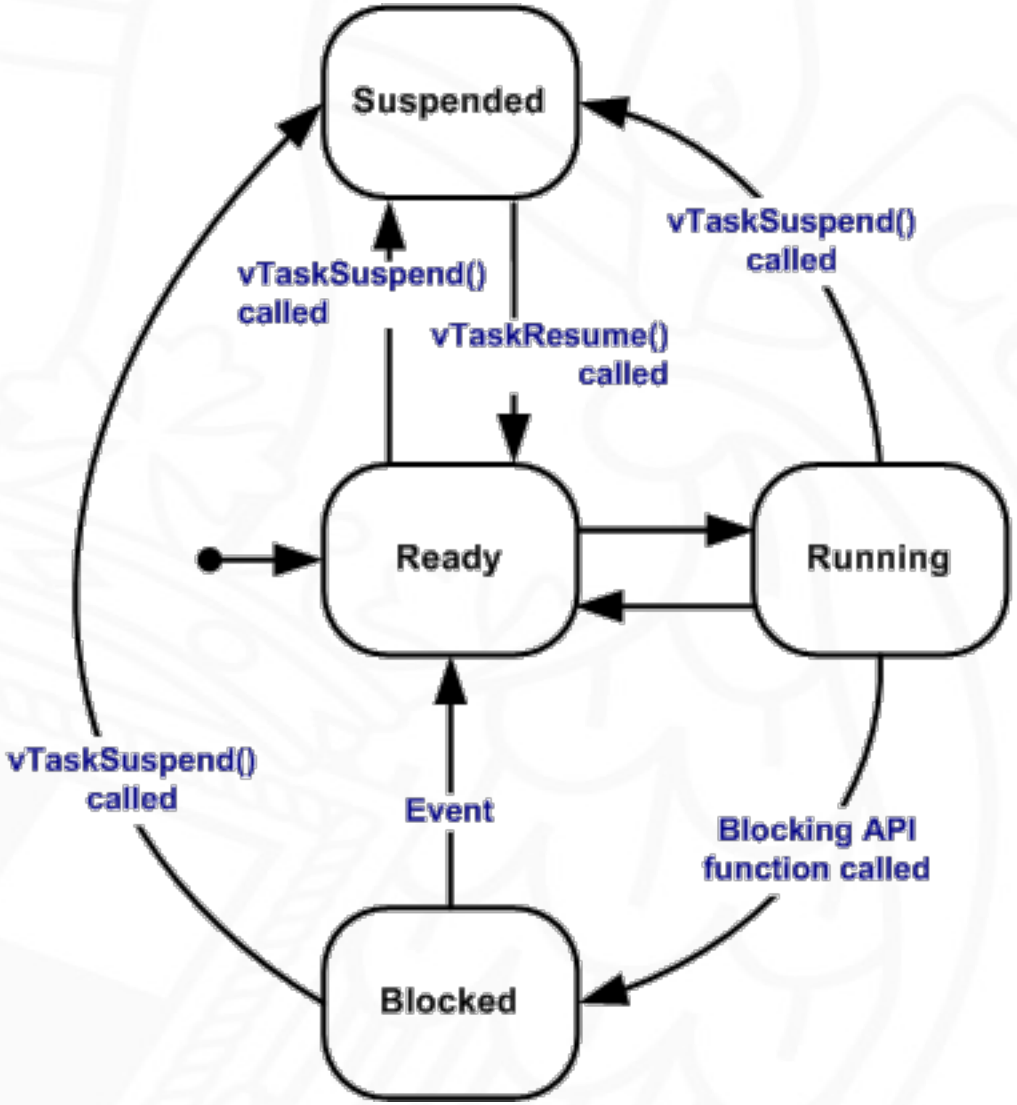
```
...
TaskHandle_t xHandle = NULL;
xTaskCreate(&exampleTask, ”Example", 3072, NULL, 5, &xHandle);
...
xTaskCreatePinnedToCore(& exampleTask, ”ExamplePinned", 3072, NULL, 5, NULL, 0);
....
        void exampleTask( void *pvParameters )
        {

        int32_t varExample = 0;

        while( 1 ) {
          // body of task
        }
        // Important: explictely remove the task
        (including app_main())
        vTaskDelete( NULL );
        }
```

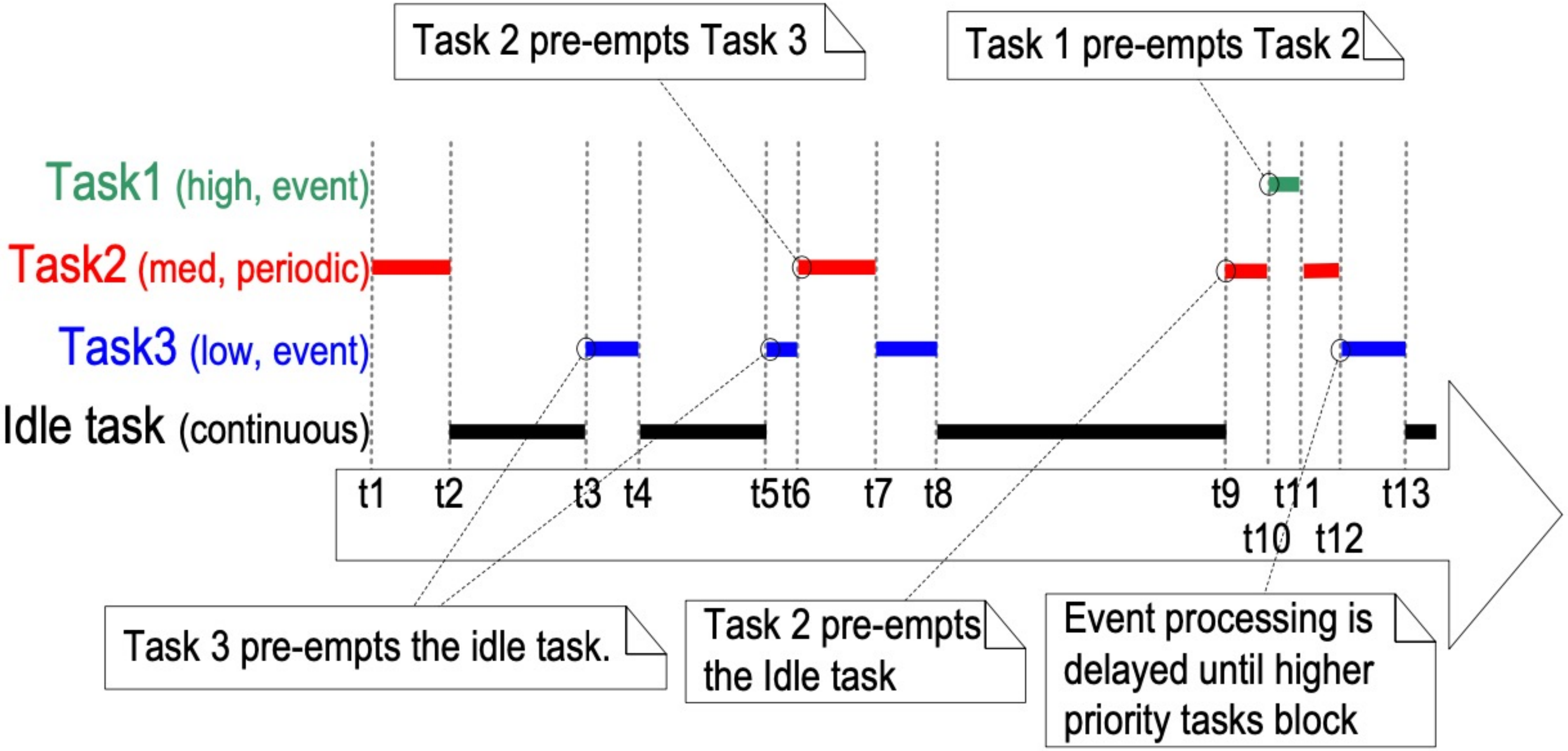
Task State

- ❑ *Ready* to run (waiting for CPU to be available)
- ❑ *Running:* running in the *core*
- ❑ *Blocked:* waiting for some external event (timer....)
- ❑ *Suspended:* waiting for other task to explicit wake this one up



Fuente: https://www.freertos.org/RTOS-task-states.html

FreeRTOS (IDF) scheduling

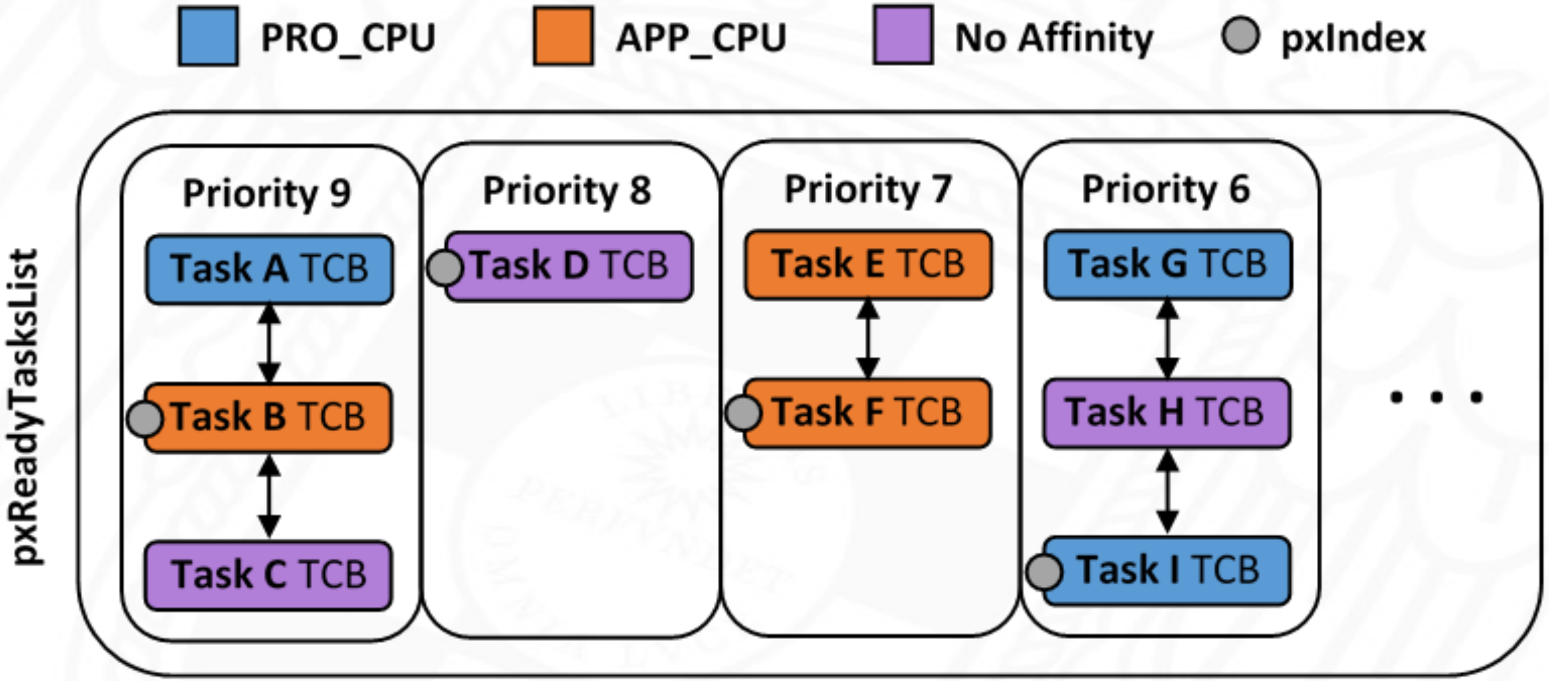


Source: R. Barry, *Mastering the FreeRTOSTM Real Time Kernel*

- Preemptive scheduling
  - Priority based
  - Round robin (time multiplex) across same priority tasks

ESP-IDF: 2 cores

- Each *core* (PRO y APP) runs its own independent scheduler
  - CPU 0 is PRO. CPU 1 is APP
  - vTaskSuspendAll() only targets the Task in the core where it is executed
- Single *Ready* task queue for both cores
  - Actually, one per priority level



FreeRTOS. Task Communications

- Single memory space → everything shared?

```
...
xTaskCreate(&exampleTask, ”Example", 3072, NULL, 5, NULL);
...
xTaskCreate(&exampleTask, ”PinnedExample", 3072, NULL, 5, NULL, 0);
....
```

```
int32_t exVar = 0;
void exampleTask( void *pvParameters )
{
for (i=0;i<10;i++) {
        exVar++;
 }
 vTaskDelete( NULL );
}
```

¿Final value of ***exVar***?

FreeRTOS. Task Communications

- Single memory space → everything shared?

```
...
xTaskCreate(&exampleTask, ”Example", 3072, NULL, 5, NULL);
...
xTaskCreate(&exampleTask, ”PinnedExample", 3072, NULL, 5, NULL, 0);
....
```

```
void exampleTask( void *pvParameters )
{
 int32_t exVar = 0;
 for (i=0;i<10;i++) {
        exVar++;
 }
 vTaskDelete( NULL );
}
```

¿Final value of ***exVar***?

Task Synchronization

- ❑ We also need to control the *relative ordering* of memory accesses
  - Which sequence of values does *exVar* have during the execution?

```
...
xTaskCreate(&exampleTask, ”Example", 3072, NULL, 5, NULL);
...
xTaskCreate(&exampleTask2, ”PinnedExample", 3072, NULL, 5, NULL, 0);
....
```

```
int32_t exVar = 0;
void exampleTask( void *pvParameters )
{
for (i=0;i<4;i++) {
        exVar = exVar + 1;
 }
 vTaskDelete( NULL );
}
```

```
void exampleTask2( void *pvParameters )
{
for (i=0;i<4;i++) {
        exVar = exVar +2;
 }
 vTaskDelete( NULL );
}
```

FreeRTOS Communication/Syncrho mechanisms

- FreeRTOS Queues: provide communication and synchronization
  - Writes get block if queue is full (*xQueueSend*)*
    - Data is **copied** into the queue
    - But we can still pass a pointer of course
  - Reads get block if queue is empty (xQueueReceive)
    - Reading removes an element of a queue

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength, UBaseType_t uxItemSize );
BaseType_t xQueueSendToFront( QueueHandle_t xQueue, const void * pvItemToQueue,
                              TickType_t xTicksToWait );
BaseType_t xQueueSendToBack( QueueHandle_t xQueue, const void * pvItemToQueue,
                             TickType_t xTicksToWait );
BaseType_t xQueueReceive( QueueHandle_t xQueue, void * const pvBuffer,
                          TickType_t xTicksToWait );
```

https://docs.espressif.com/projects/esp-idf/en/stable/api-reference/system/freertos.html#queue-api

Example

```
static void vSenderTask( void *pvParameters ) {
 int32_t lValueToSend;
 BaseType_t xStatus;
 lValueToSend = ( int32_t ) *pvParameters;
 for( ;; ) {
   xStatus = xQueueSendToBack( xQueue,
&lValueToSend, 0 );
   if( xStatus != pdTRUE ) {
     printf(“Error sneding...\n”);
   }
 }
}
```

```
static void vReceiverTask( void *pvParameters )
{
  int32_t lReceivedValue;
  BaseType_t xStatus;
  const TickType_t xTicksToWait = pdMS_TO_TICKS( 100 );
  while (1) {
    xStatus = xQueueReceive( xQueue, &lReceivedValue,
xTicksToWait );
           If ( xStatus == pdTRUE ) {
       vPrintStringAndNumber( "Received = ",lReceivedValue );
    }
  }
}
```

```
QueueHandle_t xQueue;
void app_main( void ) {
  xQueue = xQueueCreate( 5, sizeof( int32_t ) );
  if( xQueue != NULL ) {
    xTaskCreate( vSenderTask, "Sender1", 1000, ( void * ) 100, 1, NULL );
    xTaskCreate( vSenderTask, "Sender2", 1000, ( void * ) 200, 1, NULL );
    xTaskCreate( vReceiverTask, "Receiver", 1000, NULL, 2, NULL );
   }
  ...
}
```

FreeRTOS. Semaphores

- Provide synchronization across tasks. A task can...
  - ... *wait* in a semaphore (*xSemaphoreTake*(...)).
    - Potentially blocking call (but MAY not be blocking)
  - ... *release* a semaphore (*xSemaphoreGive(..))*
    - *Never blocking call*
- Two *flavors*: binary or counting semapohores
  - Binary: two states (open | closed)
  - Counting: integer associated (if 0 or negative, a call to xSemaphoreTake() will block the calling task)

Example (careful: it is not a good implementation)

```
int32_t sharedValue;
static void vSenderTask( void *pvParameters ) {
 BaseType_t xStatus;
 sharedValue = ( int32_t ) *pvParameters;
 for( ;; ) {
    sharedValue++;
    xSemaphoreGive( mySem );
  }
}
```

```
static void vReceiverTask( void *pvParameters )
{
while (1) {
     xSemaphoreTake( mySem, portMAX_DELAY);
     printf( "Received = %d\n", sharedValue );
    }
  }
}
```

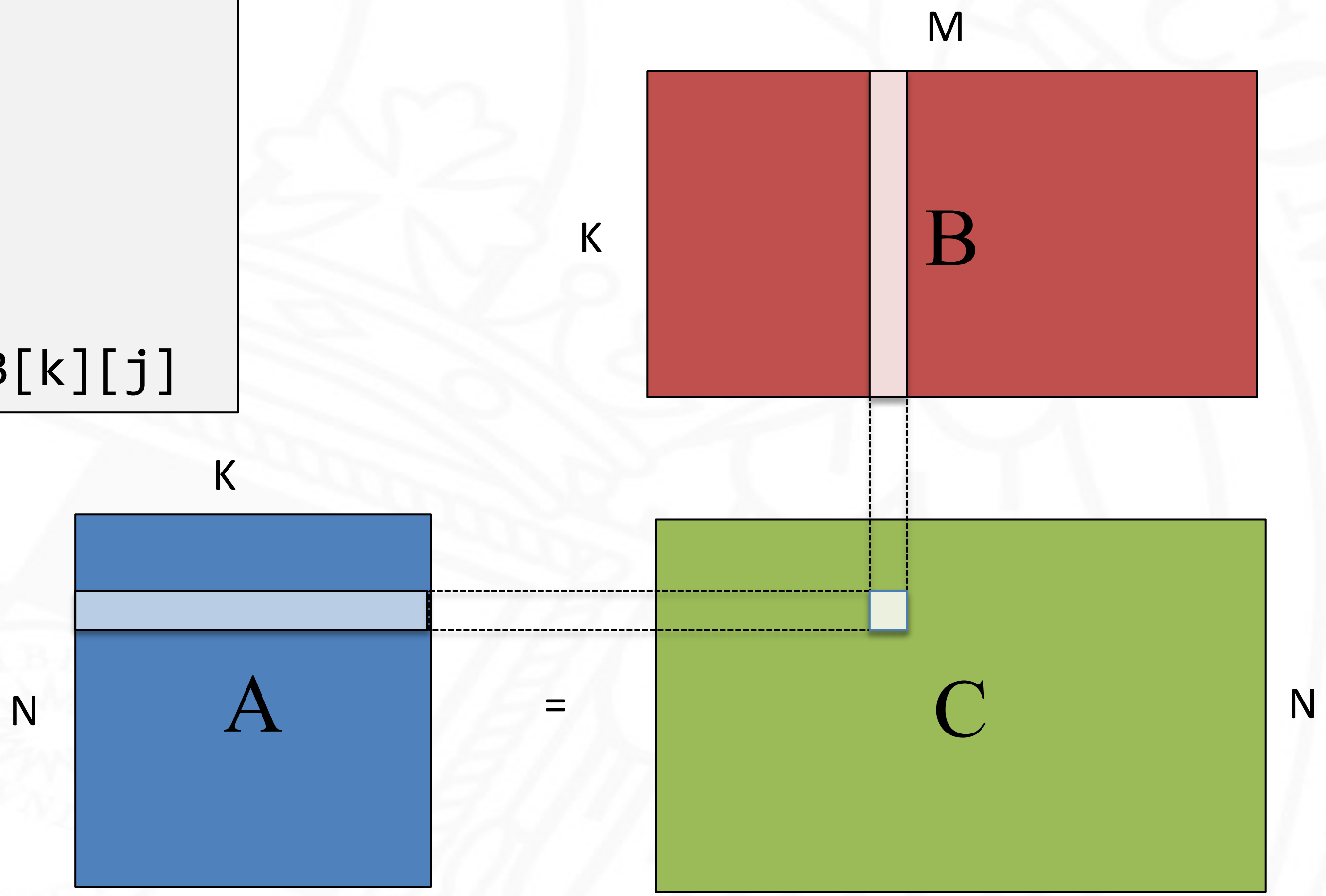
```
SemaphoreHandle_t mySem;
void app_main( void ) {
  mySem = xSemaphoreCreateBinary()
  if( mySem != NULL ) {
    xTaskCreate( vSenderTask, "Sender1", 1000, ( void * ) 100, 1, NULL );
    xTaskCreate( vReceiverTask, "Receiver", 1000, NULL, 2, NULL );
   }
  ...
}
```

Lab assignment

- Multitask Matrix multiply

$C_{n \times m} = A_{n \times k} * B_{k \times m}$

```
for i = 0 to N
  for j = 0 to M
          C[i][j] = 0
          for k=0 to K
                 C[i][j] += A[i][k]*B[k][j]
```

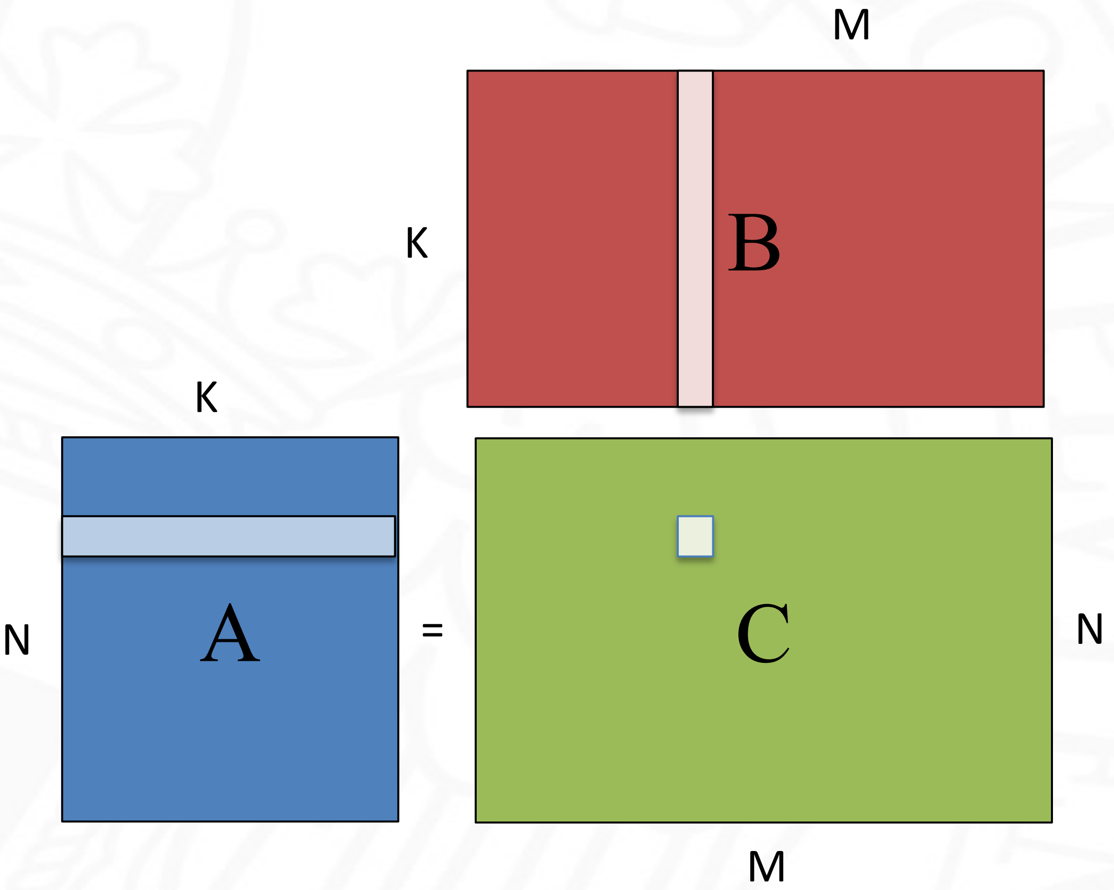


Lab assignment

- Matrix Multiply: massively parallel
  - Every element of matrix C may be computed independently (loops *i* and *j)*
  - Even loop *k* could be partially paralelized

$\mathbf{C}_{nxm} = \mathbf{A}_{nxk} * \mathbf{B}_{kxm}$

```
for i = 0 to N
  for j = 0 to M
        C[i][j] = 0
        for k=0 to K
             C[i][j] += A[i][k]*B[k][j]
```



Lab Assignment

- Implement a parallel version of matrix multiply
- Proposed implementations:
  1. Create two tasks. One task will multiply the odd rows of A and the other tasks will multiply the even rows
  2. One task (*Controller*) provides work to the other tasks (*Workers*)
     - Controller task writes pair of integers <*i,j*> into a queue
     - *Worker* task reads one pair and multiply row <*i*> by column <*j*>
     - More Worker tasks could be created dynamically