UNIVERSIDAD COMPLUTENSE MADRID

# ESP-IDF. Polling e Interrupts

IoT Node Architecture

# How do we know if a device has something for us?

- Let's image a button. When is it pressed?
  1. We may sample the button status periodically
     - *Reading* (load) from the button controller address
  2. We may configure the system so it *notifies* us when it happens
     - Pressing a button may change the voltage in a pin (for example from 0V to 3.3V)
     - We may configure the controller to **interrupt** the CPU whenever this voltage swing occursstancia

# Samplin (*polling*)

- Periodically sample if the device is ready
  - Then, proceed with the read/write
- Some devices do not require the check-if-ready step
  - Checking the voltaje in a GPIO pin
  - Reading an infrared sensor conected to an ADC

# Interrupts

- External events that interrups the execution of a task in an arbitrary point
  - It is NOT due to the task execution (although *exceptions* may happen for certain actaions like divide by 0...
  - Usually associated to I/O interactions
    - Pressing a button, new network packet arrives...
- The asystem archictecture may define several interrupts
  - And we can write a different Interrupt Service Routine (ISR) for each of them
  - And ISR looks like a normal function to us

# Interrupt Servce Routin (ISR)

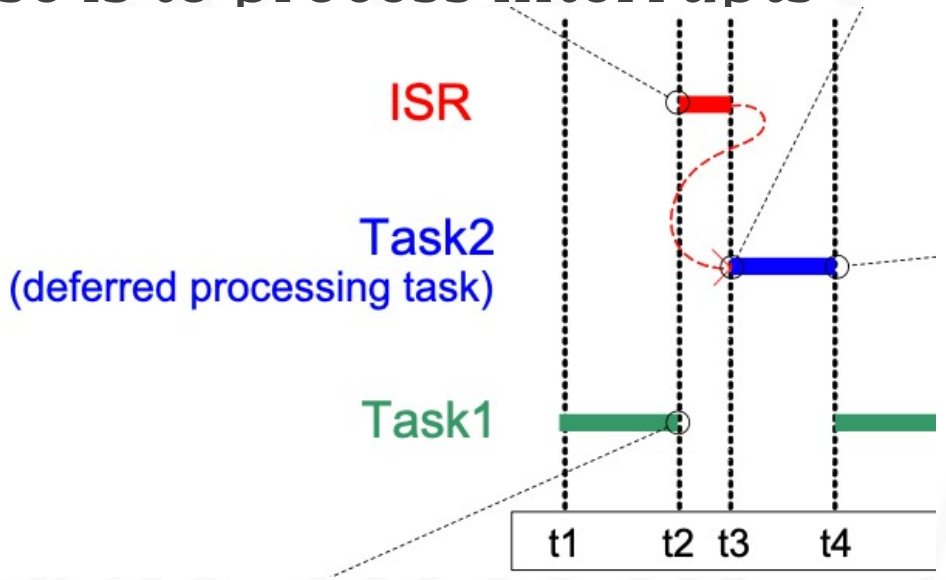
- It is a software routine (function)
  - It is automatically called when an interrupt arrives
  - They should do the minimal work possible

# ISR. FreeRTOS / ESP-IDF

- They are NOT a tasj
  - An ISR has higher priority that any task
  - A task will nevel call to a ISR
- It MUST NOT block
  - We shouldn’t use functions from ESP-IDF that potentially could get block
    - Semaphores, queues, writing to the terminal....
  - FreeRTOS defines a set of safe ISR functions
    - They append “FromISR” in its name
    - xQueueSend**FromISR**(), xTimerStart**FromISR() ....**

# Delayed interrupt management

- Often it is a good idea to delegate the processing of an interrupt to a task
- Thus, and ISR should
  - *Keep* the source of interrupt (if required later)
  - *Clean* the interrupt (to avoid it to trigger again. FreeRTOS does it for us
  - Notificar a una tarea que debe hacer el procesamiento asociado a la interrupción
- We could include tasks whose only purpose is to process interrupts
  - They will be typically blocked (semaphore)
  - ISR will wake it up
  - The could be high priority



# Using semaphores (counting semaphores)

- ISR can unblock a counting semaphore
  - A task will be waiting for in a counting sempahore
  - ISR will *post* the semaphore (*Give()* in ESP-IDF syntax)
  - *The waiting task will finish the required computation*
  - Important: use *timeout* when waiting int the semaphore to check for potential errors in the device

# ESP-IDF interrupt support

- ESP-IDF has services to register interrupts linked to one of the interrupt sources from the ESP32
  - ESP32 has 71 interrupt sources
  - Each of the two *cores* (PRO – APP) has 32 interrupt levels
    - 26 of them linked to peripherals
  - **esp_intr_alloc()** allows to associate one of the 71 available interrupts to some *free slot* of a core
  - That is what drivers do....we DO NOT develop *drivers*
    - We will use higher level functionality to register ISRs for specific devices
- ISRs must have the following prototype
  - `void ISRname(void *arg)`