URJC Escuela de Ingeniería de Fuenlabrada

RoboticsLabURJC
Programming Robot Intelligence

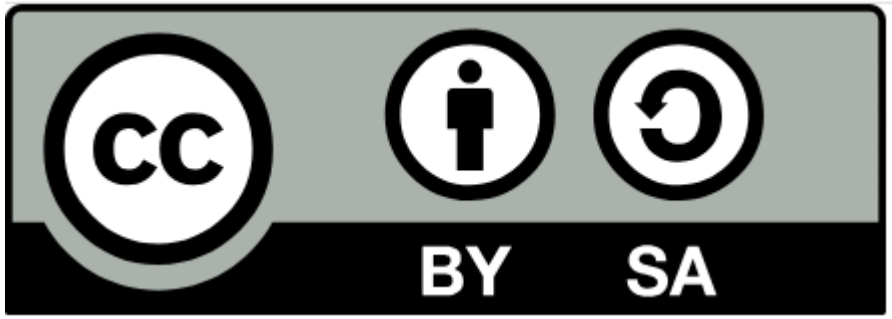
Sistemas Empotrados y de Tiempo Real

Introducción

Grado en Ingeniería de Robótica Software

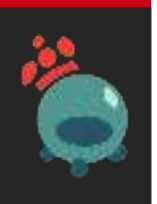
Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es



¿Qué es sistema empotrado?

- Un sistema **empotrado** es un sistema de computación diseñado para realizar **una o algunas pocas** funciones de control.
  - El contexto y dominio del problema es específico
  - No tienen un propósito generalista
  - Recursos limitados
  - No disponen de interfaces de entrada/salida (no siempre se cumple)
  - Llevan años en auge gracias a *Internet-of-Things* (IoT)

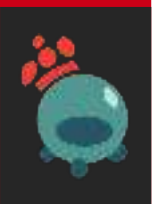


¿Qué es sistema empotrado?

¿Qué es sistema empotrado?

¿Qué es un sistema empotrado?

- Ventajas
  - Reducción de costes
  - Diseño modular
  - Corto tiempo de respuesta
  - Facilidad en la integración
  - Accesibilidad



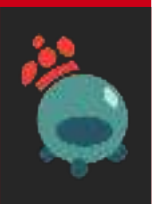
¿Qué es un sistema empotrado?

- Inconvenientes
  - Complicado realizar mejoras
  - Cifrado Débil
  - Diseños cerrados (puertas traseras)
  - Puertos de entrada y salida
  - Los fabricantes suelen descuidar bastante su seguridad.
  - Ataques DOS/DDoS

¿Qué es sistema empotrado?

- 2016: Mirai Botnet
- Gran Cyber-Ataque DDoS al proveedor de nombres de dominio (Dyn)
- Tambień se vió afectado Twitter(X), Reddit, Netflix, etc.
- Numerosos dispositivos IoT fueron hackeados (impresoras, webcams, routers, …, etc)
- En algunos dispositivos la clave estaba “hardcodeada” en la ROM del dispositivo.

https://es.wikipedia.org/wiki/Ciberataque_a_Dyn_de_octubre_de_2016



¿Qué es sistema empotrado?

- 2019: Cámaras NEST
- Acceso remoto, privacidad, …

- LG smartThinkQ hackeado (info)
- Acceso a los electrodomésticos y dispositivos

- En la primera mitad de **2023**, los ataques de malware a dispositivos IoT aumentaron un 37%, con un total de **77.9 millones** de ataques a nivel global
- Cámaras de seguridad, routers y otros sistemas conectados.

¿Qué es sistema empotrado?

- Algunos ejemplos

Arduino

Esp8266

LoRa

Raspberry Pi?

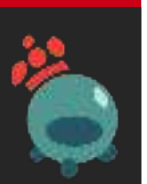
Ejemplos

Ejemplos

- SmartHomes

Ejemplos

Sistemas en Tiempo Real

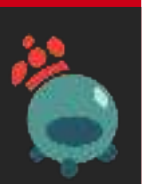


¿Qué es un sistema de tiempo real?

- Un sistema en **tiempo real** es un sistema informático en el que se asegura los tiempos de ejecución para cada tarea.
  - Usualmente es un sistema empotrado
  - Interacciona repetidamente con su entorno físico
  - Cada tarea tiene asignada una prioridad diferente
  - No basta con que las acciones del sistema sean correctas, tienen que ejecutarse dentro de un intervalo de tiempo determinado
  - Restricción de tiempo no significa que las tareas se ejecuten rápidamente.

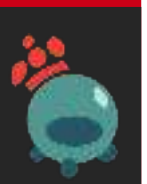
¿Qué es un sistema de tiempo real?

- Características de un sistema de tiempo real (STR)
  - **Determinismo**: Es importante asegurar con una alta probabilidad que la tarea *(T)* siempre va a durar el mismo tiempo *(t)*
  - **Responsividad**: Tiempo que tarda una tarea (*T*) en ejecutarse desde que el sistema operativo ha decidido ejecutar dicha tarea (*T*)
  - **Confiabilidad**: El sistema debe de seguir en funcionamiento a pesar de catástrofes, o fallas mecánicas. Usualmente una degradación en el servicio en un sistema de tiempo real lleva consecuencias catastróficas.
  - **Operación a prueba de fallos**: En caso de fallar, el sistema debe preservar los datos y ejecutar tareas de mayor prioridad.



¿Qué es un sistema de tiempo real?

- Clasificación (según restricciones temporales):
  - **Tiempo real duro** (hard real time): Cuando es absolutamente necesario que la respuesta se produzca dentro del límite de tiempo especificado. Ej.: control de vuelo, coches autónomos.
  - **Tiempo real firme** (firm real time): Cuando se permite la pérdida ocasional de los tiempos límites de ejecución en las tareas, pero dicha perdida no implica que el sistema falle. Ej.: video-conferencia, GPS.
  - **Tiempo real suave** (soft real time): Cuando se permite frecuentemente los tiempos límites de ejecución en las tareas, siempre y cuando la ejecución de las tareas tengan sentido con el valor obtenido (aunque haya llegado tarde). Ej.: sistema de reserva de tickets.

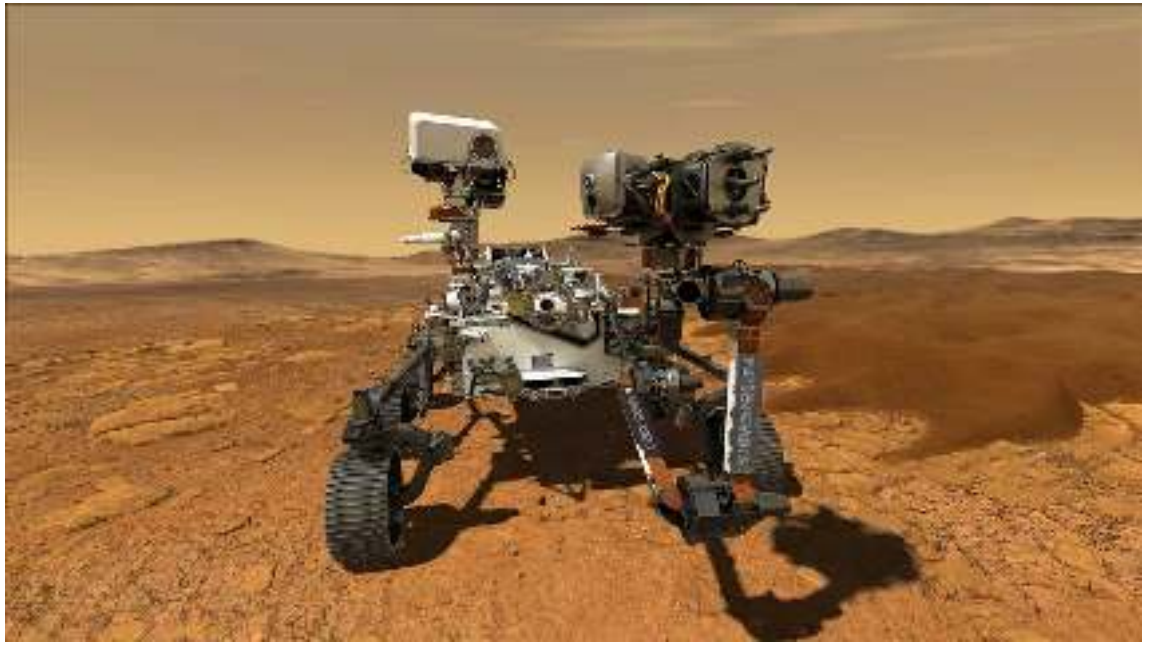


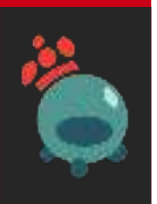
Ejemplos

- Computador de Navegación del Apolo 11
  - Controlar la navegación del modulo de mando y del modulo lunar
  - Diseñado en el MIT
  - Luminary (SO) permitía máximo 8 tareas.
  - Tareas priorizadas
  - 2Kb de memoria para el área de datos.
  - El Apolo 11 generó errores durante el alunizaje, pero consiguió recuperarse gracias a la gestión de prioridades.

Ejemplos

- Rover Perseverance – Mars 2020
- Dispone de 2 GB memoria flash, 256 MB RAM y opera a 200 MHz
- Open Souce real-time operating system RTEMS 4.5.0
  - https://www.rtems.org/





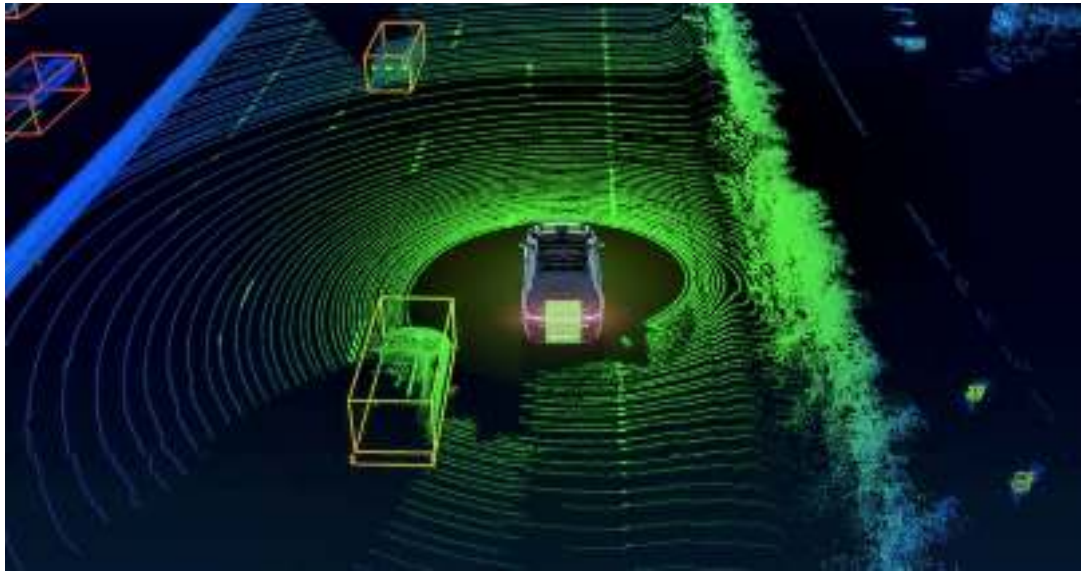
Ejemplos

- Automoción tradicional
  - ESP o ABS
  - Airbags
  - Sistemas de alerta de ángulo muerto.

- Automoción moderna
  - Análisis del entorno con visión o radar
  - Detección de peatones y frenada de emergencia.

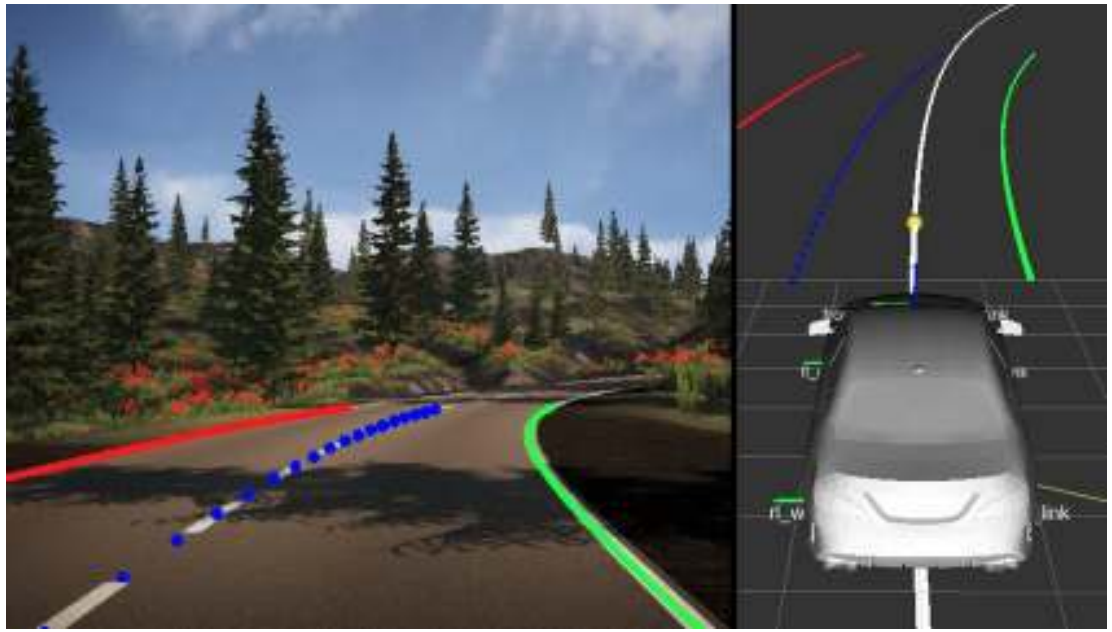
Ejemplos

- Conducción Autónoma:
  - Sistema de detección y evasión de obstáculos



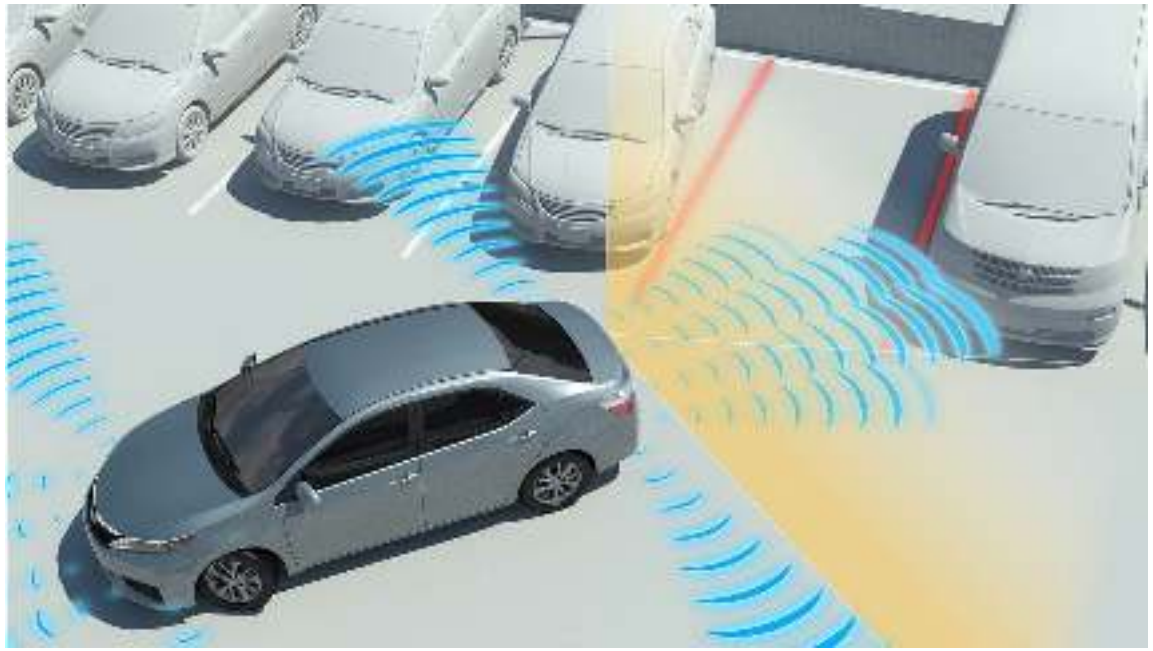
Ejemplos

- Conducción Autónoma:
  - Sistema de asistencia de carril



Ejemplos

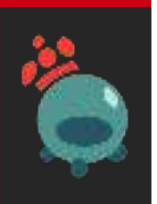
- Conducción Autónoma:
  - Sensores de proximidad y detección ultrasónica



Ejemplos

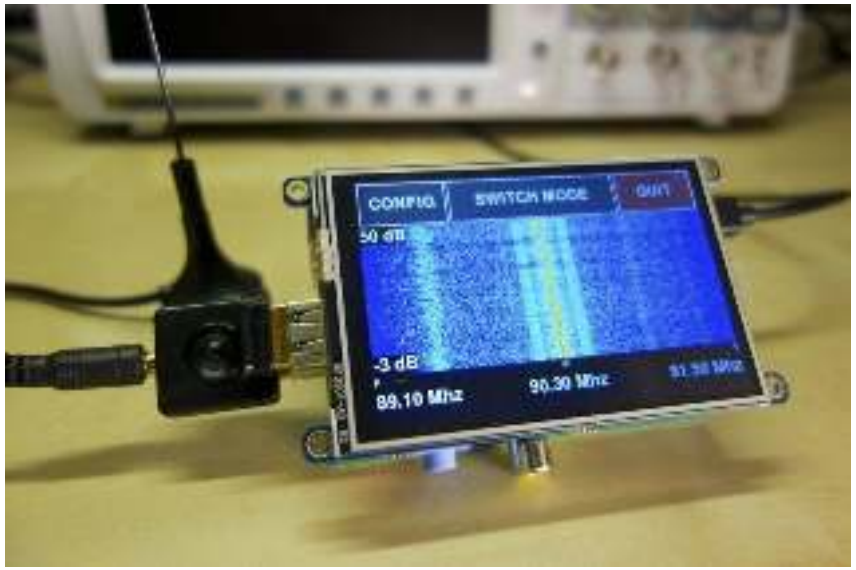
- Aviación: Se espera que todas las tareas terminen en un tiempo determinado
  - Control de Flaps
  - Sistemas de vuelo
  - Sistemas de colisión

- Es una industria donde los sistemas empotrados y de tiempo real, tienen una importancia máxima.
  Salvan vidas!



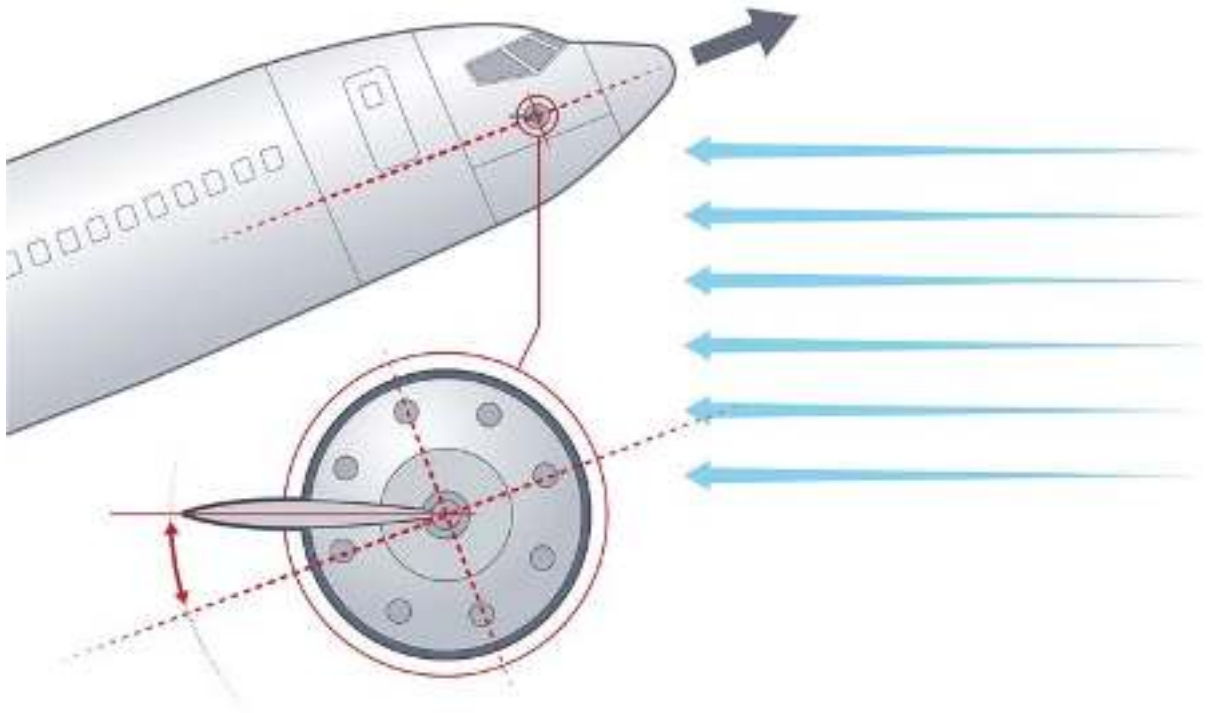
Ejemplos

- RTL-SDR
- Software Defined Radio
- Típicas funciones implementadas en hardware analógico, ahora son implementadas en software dentro un sistema empotrado.



Ejemplos

- Boeing 737 MAX-8
  - 2 accidentes en menos de 5 meses (Indonesia y Etiopía).
  - MCAS sistema empotrado/tiempo real para estabilizar el avión.



Ejemplos

- Entonces … ¿Qué son los smartphones?

URJC Escuela de Ingeniería de Fuenlabrada

RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

RoboticsLabURJC
Programming Robot Intelligence

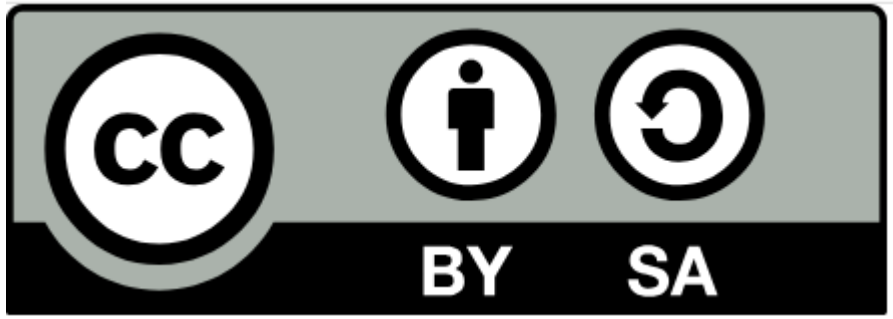
Sistemas Empotrados y de Tiempo Real

Planificación de Tareas

Grado en Ingeniería de Robótica Software

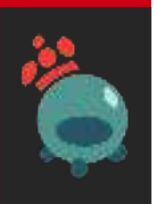
Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es



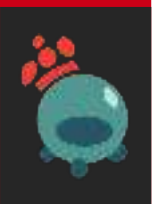
Introducción

- Usualmente los sistemas empotrados de tiempo real disponen de **mayor número de tareas** a ejecutar que numero de núcleos o unidades de procesamiento.
- Las tareas tienen que **competir** para obtener acceso a la unidad de procesamiento.
- El **planificador** es el encargado de decidir que tareas y recursos se ejecutan en cada momento.
- Estados de las tareas:
  - Creación
  - Listo
  - Ejecución
  - Terminación



Tareas en Sistemas en Tiempo Real

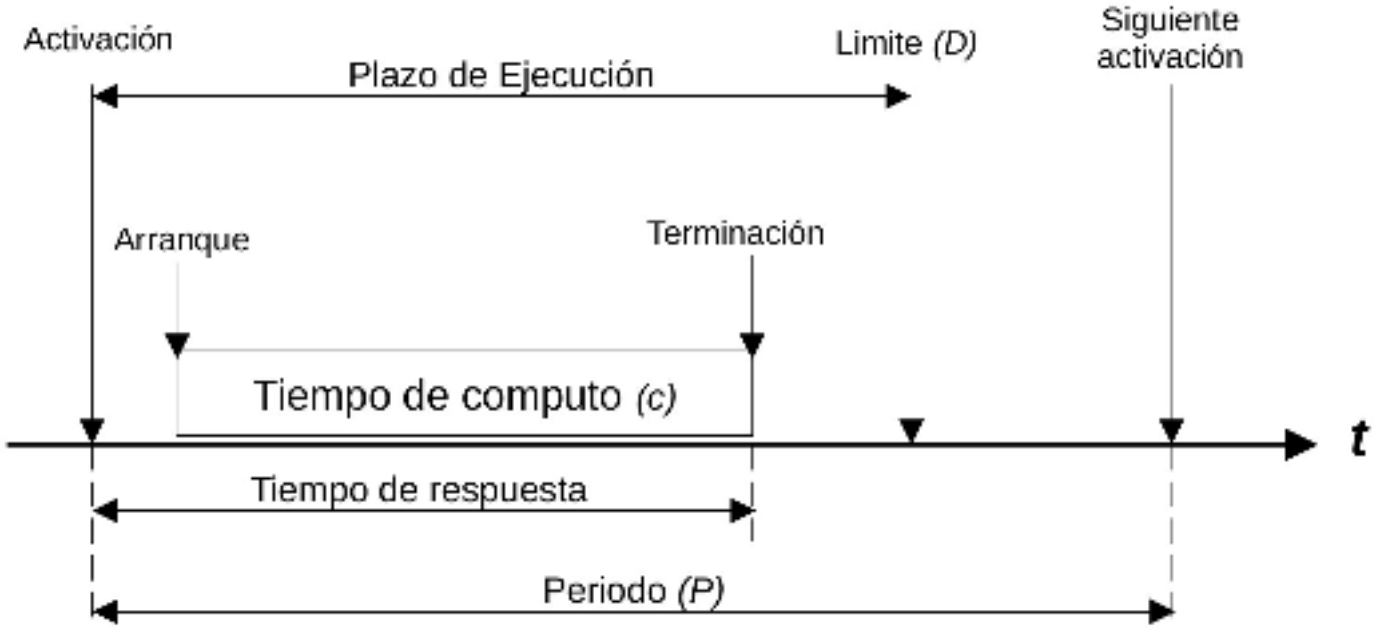
- Clasificación de los STR según el flujo de ejecución
  - **Sistemas monotarea:**
    - Un único flujo de ejecución
    - Bucle infinito
    - Muestreo de las entradas (sondeo – polling)
    - Sencillos pero poco flexibles
    - Difícil añadir nueva funcionalidad.
  - **Sistemas multitarea:**
    - Compuestos por un conjunto de tareas
    - Procesos concurrentes
    - Controlar recursos
    - Comunicación entre tareas



Tareas en Sistemas en Tiempo Real

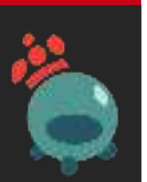
- En función de la forma de ejecución, las tareas se clasifican:
  - **Tareas periódicas**: Se activan repetidamente a intervalos de tiempo constantes.
    - Período de activación *(p)*
    - Período de ejecución *(d)*
    - Tiempo de cómputo *(c)*

Se debe cumplir que $0 \leq c \leq d < p$



Tareas en Sistemas en Tiempo Real

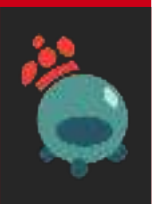
- **Tareas esporádicas.**
  - Se activan en instantes aleatorios y tienen requisitos temporales críticos.
  - Características:
    - Normalmente su ejecución depende de eventos externos no planificados.
    - Son críticas, tienen que ser atendidas dentro de los plazos.
  - Ejemplos:
    - Accionamiento de ABS
    - Interrupción causada por un interruptor.



Tareas en Sistemas en Tiempo Real

- **Tareas aperiódicas**.

  - Tiempos de llegada aleatorios e irregulares.
  - Podrían tener o no requisitos temporales estrictos.
  - Sus tiempos límites, son establecidos siempre desde la llegada del evento.
  - No deberían existir número altos de tareas aperiódicas en un sistema: difícil planificar.



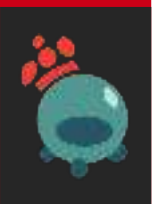
Tareas en Sistemas en Tiempo Real

- Las tareas se clasifican, atendiendo a su semántica en:
  - **Críticas**: El fallo de una de estas tareas puede ser catastrófico.
  - **Opcionales** (no críticas): Se pueden utilizar para refinar el resultado dado por una tarea crítica, o para monitorizar el estado del sistema, etc.

  ¿ Ejemplos ?

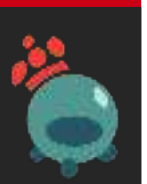
Planificador de Tareas

- Es el sub-sistema encargado de **decidir** qué tareas se ejecutan en el procesador y en qué momento.
  - Cada procesador debe estar asignado como máximo a una sola tarea en cada instante.
  - Cada tarea debe estar asignada como máximo a un solo procesador.
  - La cantidad total de tiempo de procesador asignada a cada tarea debe ser igual a su tiempo máximo de ejecución.
  - Se deben satisfacer todas las restricciones de precedencia y uso de recursos comunes.
  - Dadas ciertas restricciones, el planificador puede no conseguir una correcta planificación de tareas (ya que no existe)



Tipos de Planificadores

- Planificadores **cíclicos**
  - Cuando los parámetros de las tareas con plazos de finalización estrictos son conocidos antes de que comience la ejecución del sistema
- Planificadores por **prioridades**
  - Cada tarea tiene una prioridad debido a su importancia.
  - La asignación de prioridades puede ser estática o dinámica.
  - Planificadores con prioridades estáticas:
    - Rate Monotonic (RM): Tareas son prioridad alta son planificadas más frecuentemente.
    - Deadline Monotonic (DM). Tareas con el plazo más corto son asignadas con prioridades altas.
  - Planificadores con prioridades dinámicas:
    - Earliest Deadline First (EDF) y el Least Laxity First (LLF)

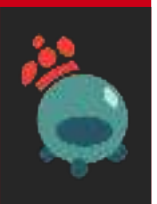


Planificación Cíclica de Tareas

- Ejecutan de forma iterativa un conjunto de tareas **periódicas** con un solo procesador.
- A partir de los requisitos temporales del conjunto de tareas, se define la secuencia de tareas que deben ejecutarse durante un período fijo de tiempo (**ciclo principal**)
- El plan principal se divide en **planes** secundarios. Secuencia de procesos que se deben ejecutar durante un período de tiempo fijo

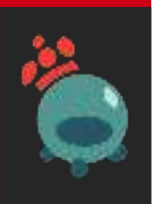
Planificación Cíclica de Tareas

- Los planificadores cíclicos son aplicables únicamente cuando existe un grado alto de determinismo. Supuestos:
  - En el sistema existen un número N de tareas periódicas que permanece constante durante toda la vida del sistema.
  - Los parámetros de todas las tareas periódicas son conocidos.
  - Existe un único procesador en el sistema.



Planificación Cíclica de Tareas

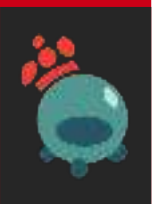
- Los métodos de planificación estática basados en una planificación **cíclica** son comúnmente utilizados en sistemas de tiempo real críticos. El diseño se realiza off-line.
- Ventajas:
  - Determinista y predecible
  - No hay sobrecarga por cambios de contexto ni exclusión mutua.
  - La implementación es sencilla y el planificador se ocupa de activar los procesos por turnos.
  - Las tareas se ejecutan según el plan definido por el diseñador/programador.
  - El sistema operativo se reemplaza por una ejecución cíclica
- Desventajas:
  - El diseño de los planes es bastante complejo
  - No son fáciles de mantener y actualizar.



Planificación Cíclica de Tareas

- Sistemas mono-procesador
- Conjunto de tareas estáticas
- Solo admite tareas periódicas
  - Tiempo máximo de ejecución igual al periodo.
- Sistema síncrono: Todas las tareas piden ejecución al mismo tiempo. Todas las tareas se activan en *t=0*
- Hiper-periodo es el mínimo común múltiplo de los periodos de todas las tareas.
- Tareas periódicas no se suspenden por ellas mismas
  - Únicamente al final de la ejecución.

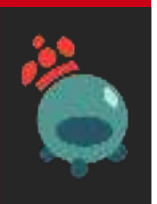
$$tiempo\_computo(c) \leq plazo\_ejecucion\ (d) < periodo\_activacion\ (p)$$



Planificación Cíclica de Tareas

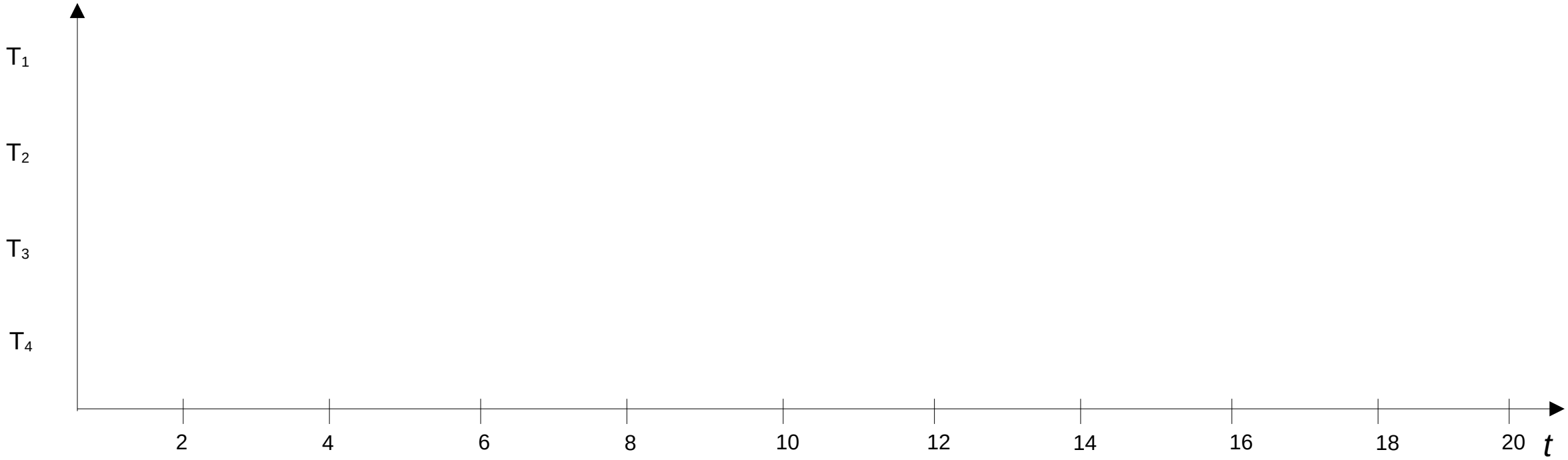
- Una condición necesaria para que sea planificable el conjunto de procesos periódicos *{P1,P2,…,Pn}* con requisitos temporales representados por *(pi,di,ci)* es que la utilización del procesador u sea menor o igual que uno

$$U = \sum_{i=1}^{k} \frac{C_i}{P_i} \leq 1$$



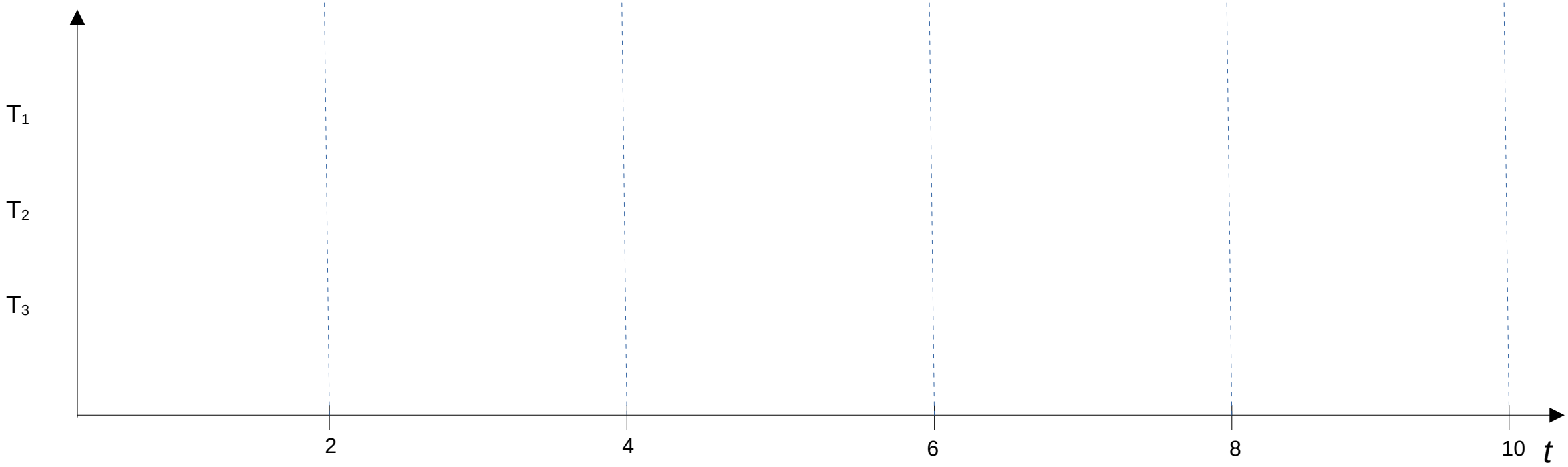
Planificación Cíclica de Tareas

Proceso	p	d	c
T_1	4	4	1
T_2	5	5	1.8
T_3	20	20	1
T_4	20	20	2



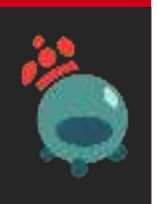
Planificación Cíclica de Tareas

Proceso	p	d	c
T_1 (lectura odometria)	2	2	1
T_2 (calculo bloqueo ruedas)	4	4	1
T_3 (accionar abs)	10	10	1



Tareas interrumpibles (PREEMPT)

- Un sistema en tiempo real puede permitir que sus tareas puedan ser **interrumpidas** en cualquier momento para que una tarea de mayor prioridad se ejecute.
- La interrupción de una tarea se denomina **PREEMPTION**
- El sistema podría tener tareas que no son interrumpibles.
- Al intercambio de estados en la CPU y memoria, cuando una tarea se interrumpe para dar paso de ejecución a una segunda tarea se denomina ***cambio de contexto***
- Para sistemas Linux tenemos
  - Kernel por defecto:
    - Antiguamente: No PREEMPT
    - Actualmente: PREEMPT_DYNAMIC
  - PREEMPT y PREEMPT-RT

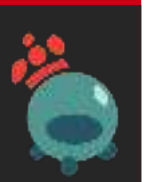


Tareas interrumpibles

- **NO_PREEMPT**
  - Cuando un proceso entra en el kernel (llamada al sistema), no puede ser interrumpido por otro proceso.
  - Ventajas:
    - Menos cambios de contexto, menos sobrecargar.
    - Mayor rendimiento en entornos no críticos en términos de latencia.
  - Desventajas:
    - Latencia de respuesta puede ser alta ya que los procesos pueden ocupar mucho tiempo en el kernel.

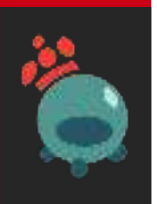
Tareas interrumpibles

- **PREEMPT_VOLUNTARY**
  - Permite varios puntos “seguros” dentro del código del kernel para poder ser interrumpido. Utilizado normalmente en servidores.
  - Ventajas:
    - Reduce latencia con respecto a NO_PREEMPT
    - Compatible con mayor numero de procesos que requieren cierta capacidad de respuesta.
  - Desventajas:
    - Latencia aún relativamente alta para sistemas tiempo real.



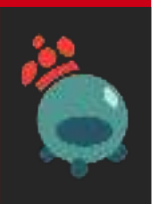
Tareas interrumpibles

- **PREEMPT**
  - Cualquier proceso en el kernel puede ser interrumpido por tareas de mayor prioridad en la mayoría de las secciones no críticas del kernel. Utilizado en sistemas de tiempo real suaves.
  - Ventajas:
    - Baja latencia y alta capacidad de respuesta.
    - Ideal para sistemas que requieren tiempo de respuestas altos.
  - Desventajas:
    - Sobrecarga por los continuos cambios de contexto.
    - Impacto alto en tareas computacionalmente altas.



Tareas interrumpibles

- **PREEMPT_DYNAMIC**
  - Incorporado a partir de kernel 5.12, permite cambiar entre modos PREEMPT y PREEMPT_VOLUNTARY sin necesidad de recompilar.
  - Ventajas:
    - Flexibilidad
    - Recursos de tiempo
  - Desventajas:
    - Puede ser más difícil de optimizar debido a la naturaleza cambiante de la interrupción.



Tareas interrumpibles

- **PREEMPT_RT**
  - Modo que permite la interrupción total y garantías temporales. Modifica el kernel para que linux funcione como un sistema de tiempo real estricto.
  - Ventajas:
    - “Garantiza” tiempos de respuesta estrictos.
  - Desventajas:
    - Impacto significativo en el rendimiento de tareas generales.
    - Mayor complejidad en el kernel.

Tareas interrumpibles

- Tipo de kernel

```
root vega ~ # uname -a
Linux vega 6.1.0-9-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.27-1 (2023-05-08) x86_64 GNU/Linux
```

- Configuración actual

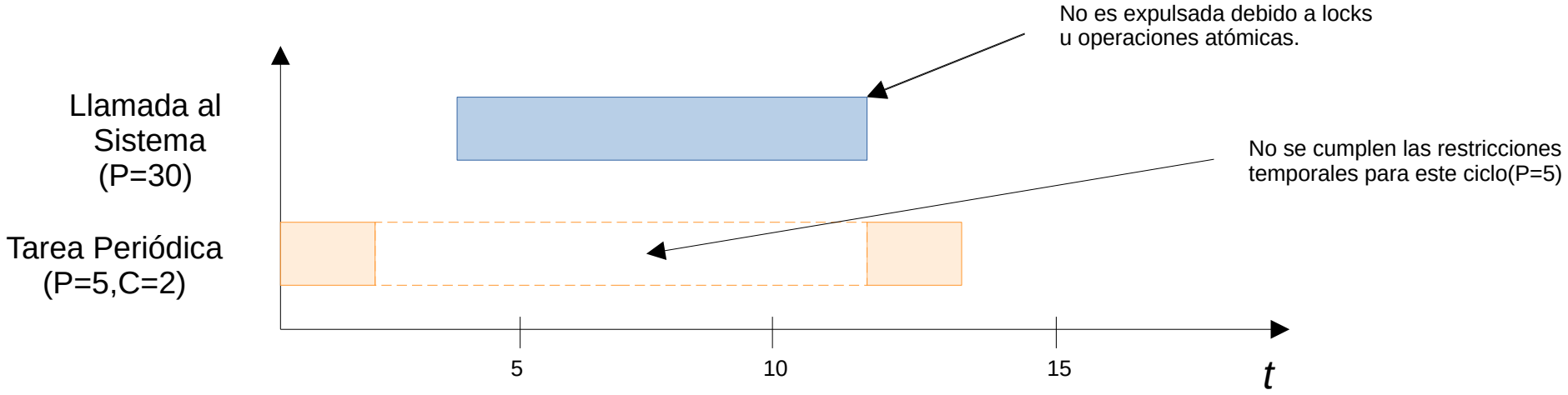
```
root vega ~ # cat /sys/kernel/debug/sched/preempt
none (voluntary) full
```

- Cambiar la configuración

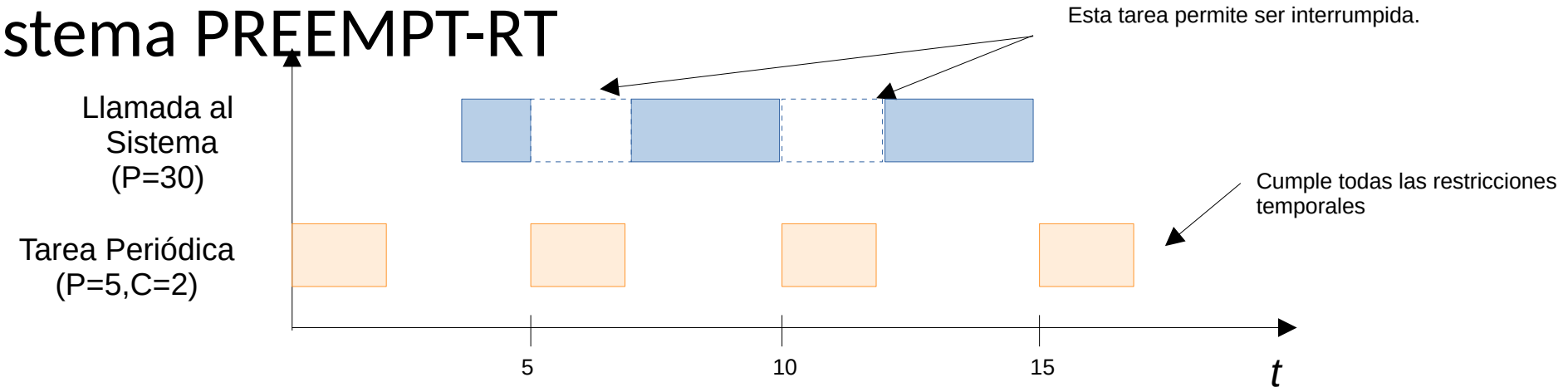
```
root vega ~ # echo voluntary > /sys/kernel/debug/sched/preempt
```

Tareas interrumpibles (PREEMPT)

- Sistema NO PREEMPT

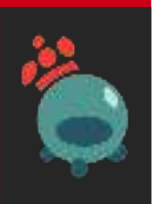


- Sistema PREEMPT-RT



Planificadores con prioridades estáticas

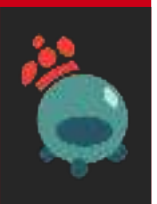
- En los planificadores por **prioridades** se asigna una prioridad a cada tarea en base a su importancia.
- En tiempo de **ejecución** se ejecutará siempre la tarea de más prioridad que este activa en cada instante.
- En este caso es el planificador quién decide en cada instante que tarea debe ejecutarse.
- El Planificador ***Rate Monotonic (RM)*** es uno de los más usados en sistemas de planificación con prioridades estáticas para tareas periódicas simples.



Rate Monotonic (RM)

- Las **prioridades** (Pr) de las tareas deciden el orden de su ejecución
- A mayor ***Pr*** antes entrará en la CPU a ejecutar, siempre dentro de su periodo de activación.
  (Ej. *P99* entra antes que *P1* en el procesador)
- **La tarea más corta tiene la prioridad más alta.**
- Todas las tareas son **periódicas** e independientes unas de otras.
- No es un planificador óptimo, **excepto** cuando las tareas periódicas son simples. Cálculo de utilización de las tareas:

$$\sum_{i=1}^{N} \frac{C_i}{P_i} < N\left(2^{1/N} - 1\right)$$



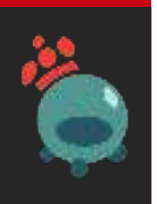
Utilización

- En la política de planificación de Rate Monotonic, todos los deadlines de N tareas periódicas están garantizados si:

$$U \leqslant N\left(2^{1/N}-1\right)$$

  - Condición suficiente pero no necesaria (Liu & Layland, 1973)
- Para calcular la utilización mínima garantizada para *N* tareas

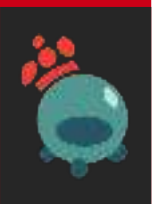
$$U_0 = N\left(2^{1/N}-1\right)$$



Utilización mínima garantizada

N	$U_0(N)$
1	1,000
2	0,828
3	0,779
4	0,756
5	0,743

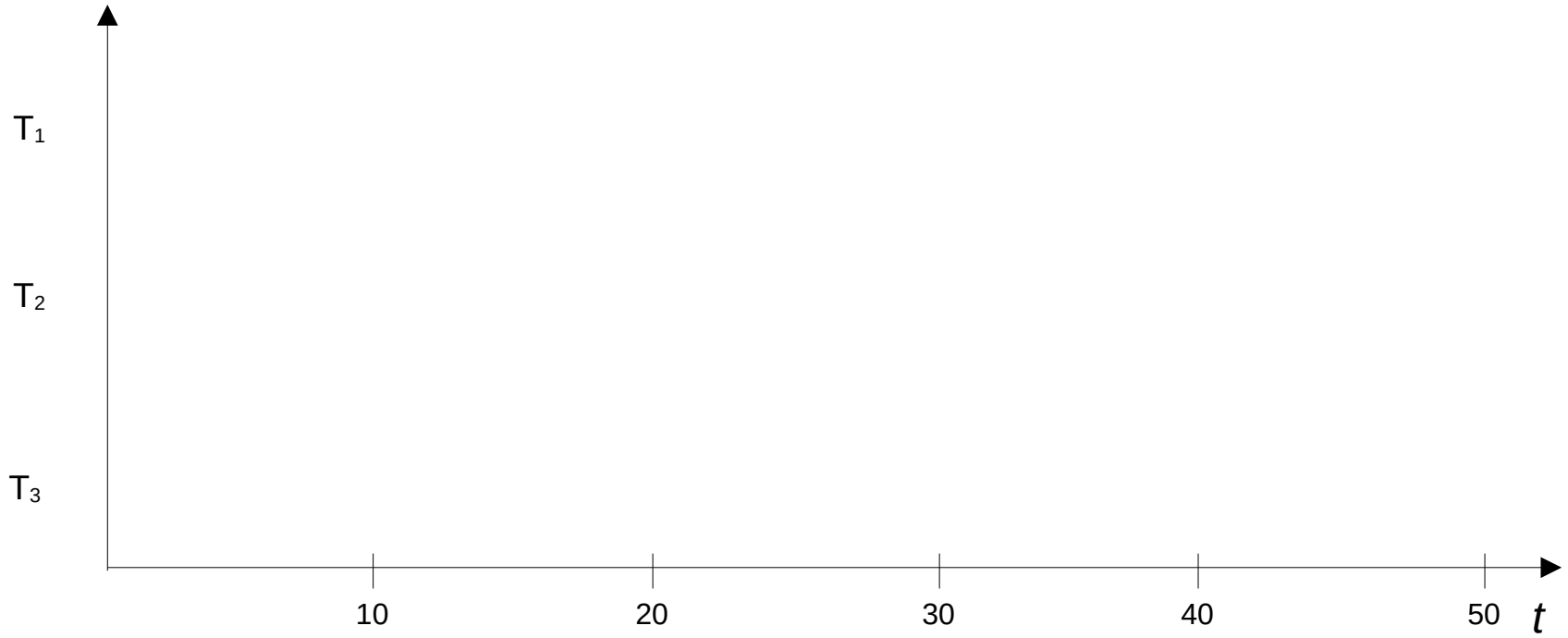
- Si $U \leqslant U_0(N)$
  - Deadlines están garantizados.
- Si $U > 1$
  - Deadlines no están garantizados.
- Si $U > U_0(N) \leq 1$
  - No es posible decidir nada mediante este método. Tendríamos que realizar el **cronograma temporal.**



Rate Monotonic (RM)

Tarea	P	C	D	Pr	U
T_1	20	5	20		
T_2	30	12	30		
T_3	50	10	50		

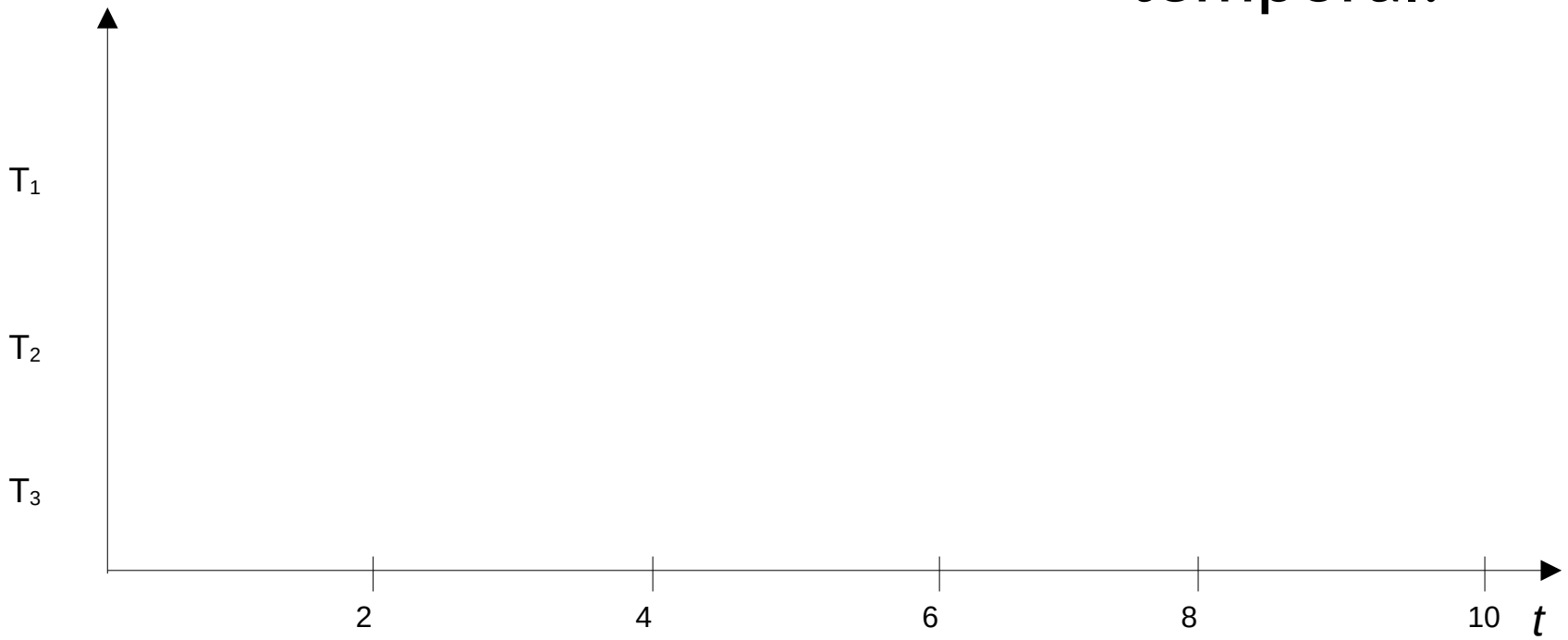
- Calculo de U=
- Calculo de $U_0(N)$ =
- Realizar el cronograma temporal.



Rate Monotonic (RM)

Tarea	P	C	D	Pr	U
T_1	5	3	5		
T_2	3	2	3		
T_3	8	4	8		

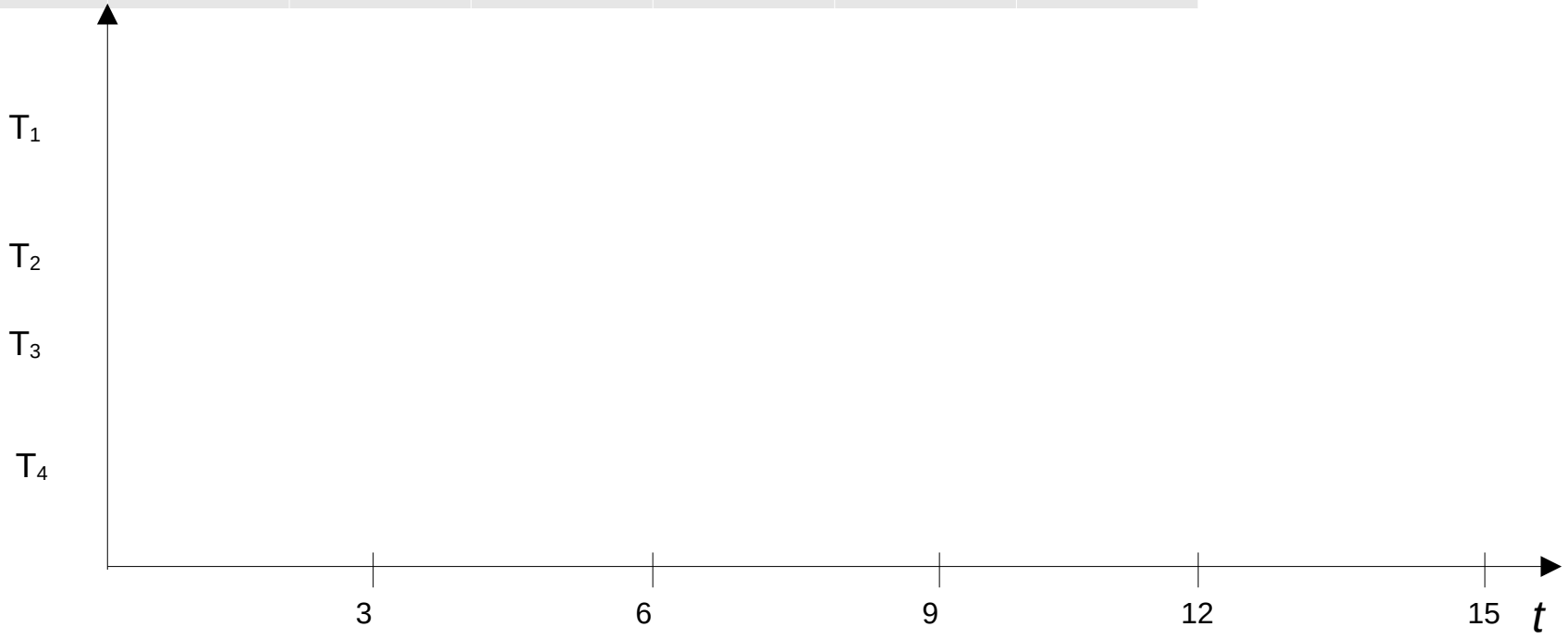
- Calculo de U =
- Calculo de $U_0(N)$ =
- Realizar el cronograma temporal.



Rate Monotonic (RM)

Tarea	P	C	D	Pr	Pr*	U
T_1	3	1	3		1	
T_2	3	1	3		3	
T_3	8	2	8		1	
T_4	13	1	13		1	

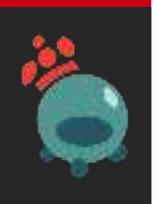
- Calculo de U =
- Calculo de $U_0(N)$ =
- Realizar el cronograma temporal.



Sistema Tiempo Real

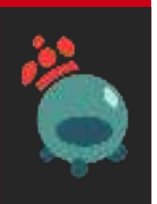
- Tareas
  - [T1] Actuador servo p=20, c=5, pr=2
    - Necesita el valor de la temperatura
  - [T2] Lector sensor temperatura: p=10, c=2, pr=3
  - [T3] Encendido de un led, p=30, c=1, pr=1
    - Se enciende después del actuador

- ¿Podrías además añadir otra tarea al sistema que lea sensor de humedad (p=20, c=3) ?
- U, U_0, cronograma temporal.

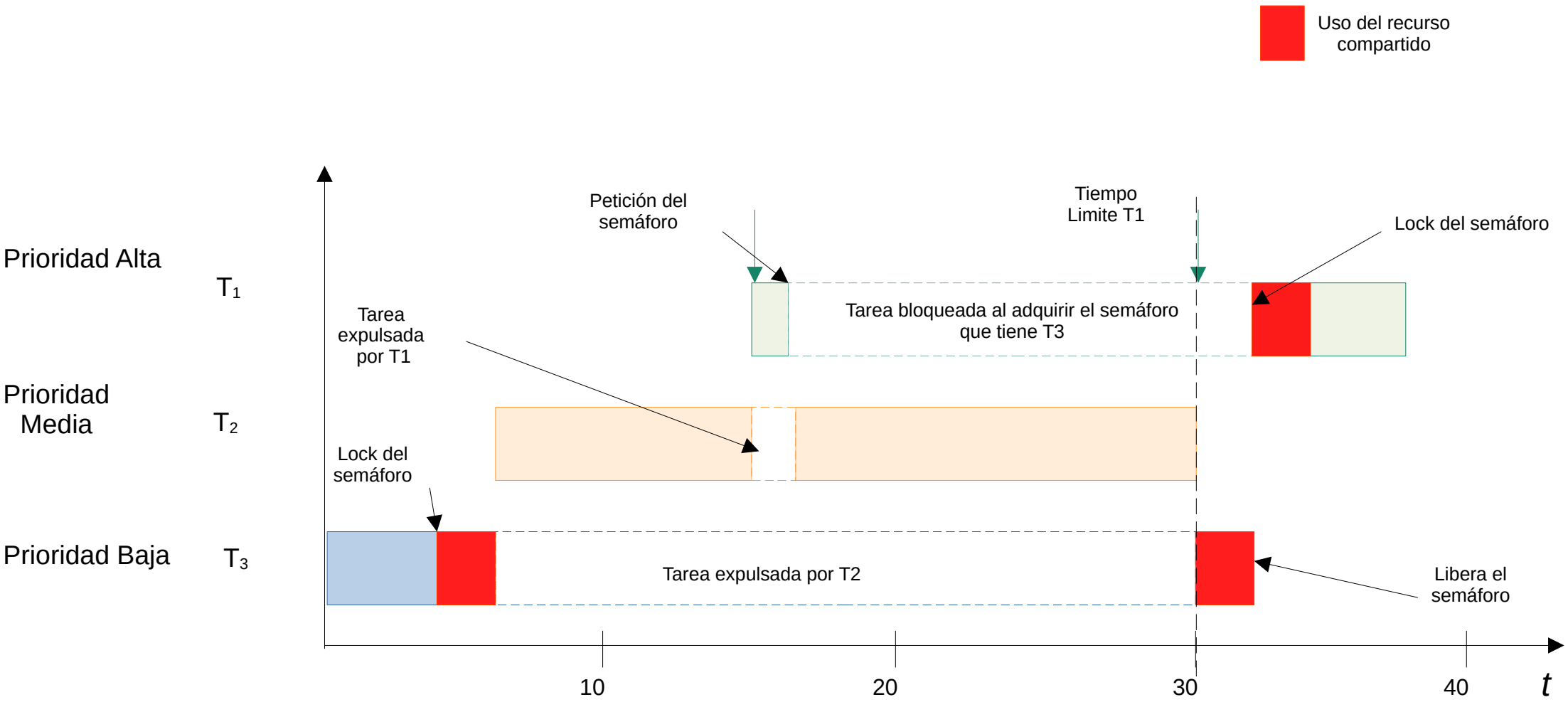


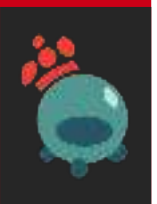
Inversión de Prioridades

- La **inversión de prioridades** es un problema que se puede presentar en todos los sistemas de multi-programación cuando se usan mecanismos para controlar la exclusión mutua al acceso de recursos compartidos.
- Escenario:
  - T1 con prioridad alta, T2 con prioridad media y T3 con prioridad baja
  - T3 coge recurso compartido (M), pero es expulsada de la CPU para ejecutar T2
  - T2 es expulsada porque T1 entra en ejecución e intenta coger (M)
  - T1 se queda bloqueada y el planificador pone a ejecutar T2
- Solución:
  - Herencia de Prioridad. El planificador incrementa la prioridad de la tarea que tiene el **cerrojo** (T3) a la máxima prioridad de otra tarea (T1) que está bloqueada esperando la liberación de dicho **cerrojo.**

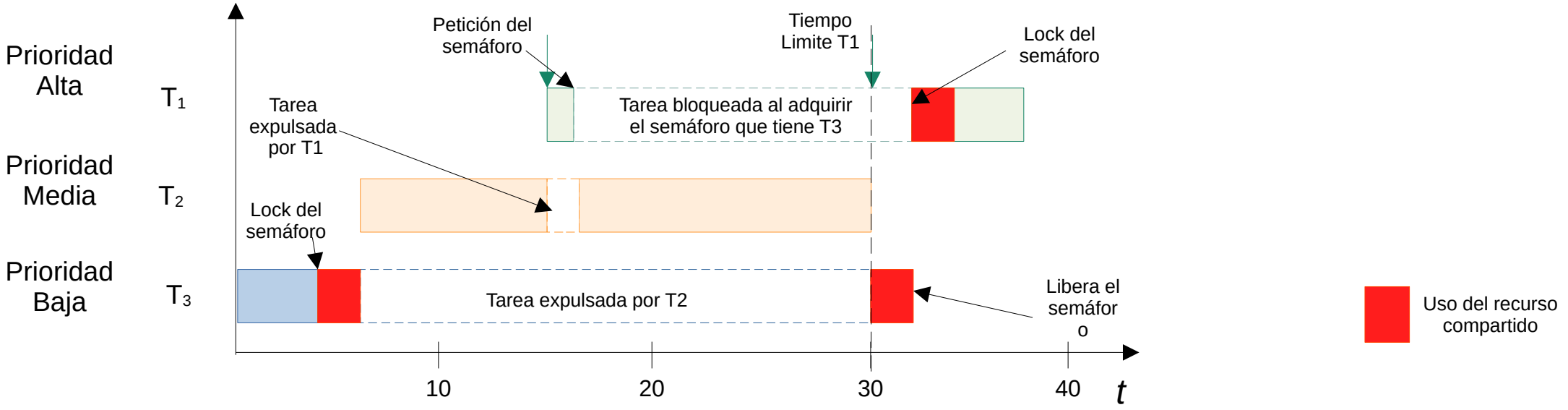


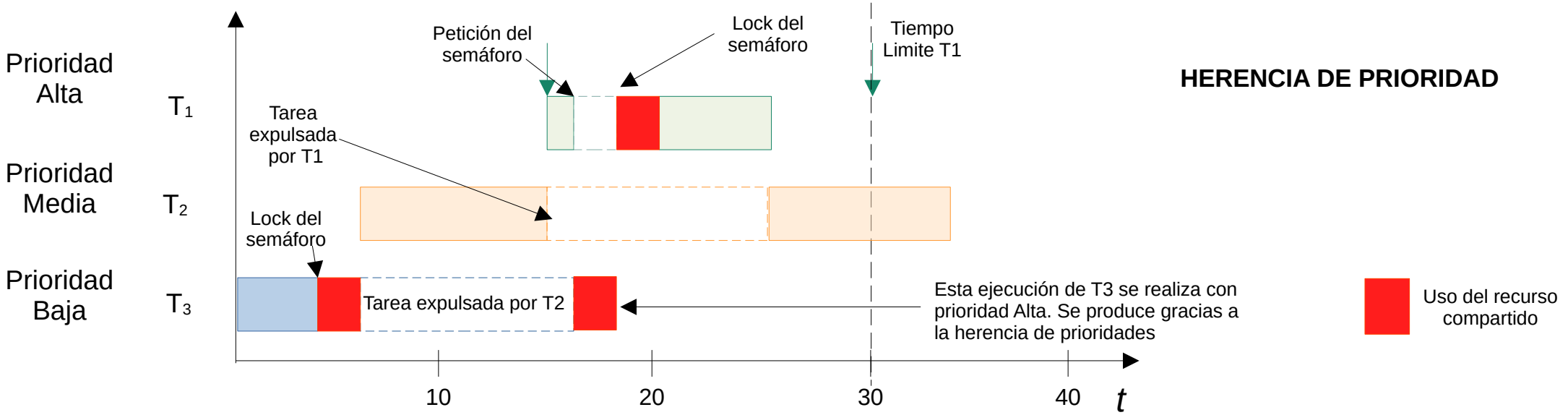
Inversión de Prioridades





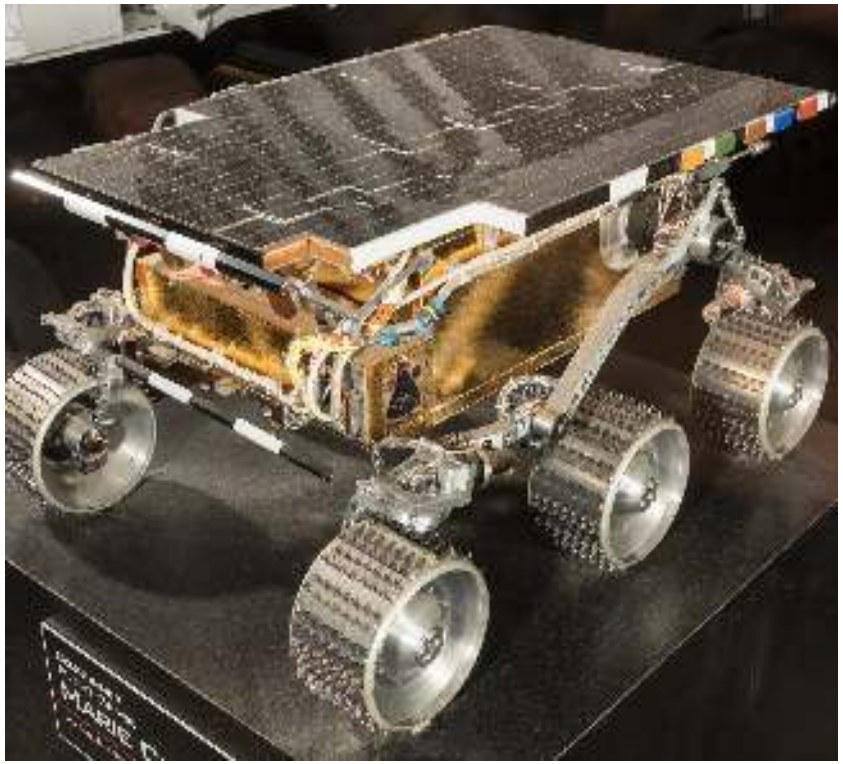
Inversión de Prioridades

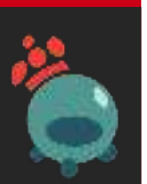




Inversión de Prioridad (en Marte!)

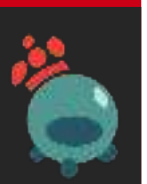
- Mars PathFinder: nava cuya misión fue estudiar Marte desde la superficie (1996)
- Sistema de tiempo real duro VxWorks RTOS
- Tareas con prioridades y “expulsables” (PREEMPT)
- Detectaron continuos reset del sistema después de varios días en Marte.
- El sistema contenía un recurso común “information bus” usado para compartir información entre diferentes componentes.



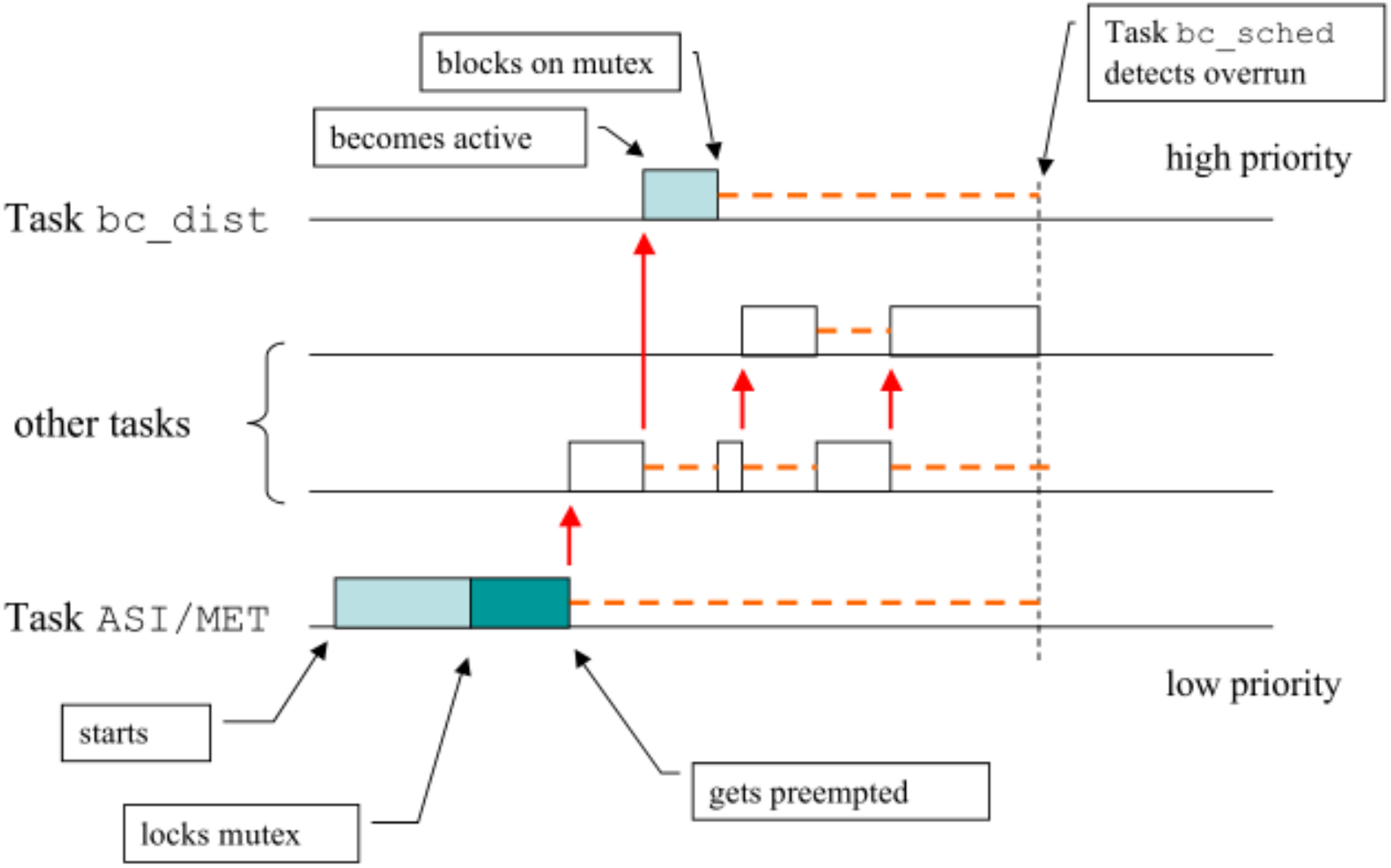


Inversión de Prioridad (en Marte!)

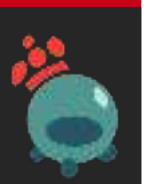
- 3 de las numerosas tareas a ejecutar por el PathFinder
  - T1: Information Bus Thread (Periodicidad y prioridad Alta)
  - T2: Communication Thread (Periodicidad y prioridad Media)
  - T3: Weather Thead (Periodicidad y prioridad Baja)
- Watchdog que comprueba que la anterior tarea ejecutó por completo en el ciclo previo.
  - Si esta comprobación falla, se considera violación del sistema de tiempo real duro y el sistema se reinicia (tolerante a fallos)
  - En cada reinicio, se perdían datos recolectados de los últimos ciclos.
- Comunicación entre tareas:
  - Memoria compartida era usada para pasar información de la T3 a la T2 via de bus (T1).



Inversión de Prioridad (en Marte!)



http://www.cse.nd.edu/~cpoellab/teaching/cse40463_fall05/lecture11.pdf

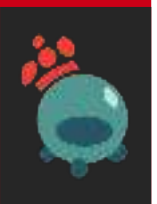


Inversión de Prioridad (en Marte!)

- Solución: habilitar la herencia de prioridad.
  - Habilitar un flag en la llamada de adquisición del semáforo.
  - Afecta al rendimiento del sistema
  - Se generó el parche y se envió a Marte por radio.

Bibliografía

- Real-Time Embedded Systems
(Ivan Cibrario Bertolotti, Gabriele Manduchi)
- Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications (Giorgio C. Buttazzo)
- A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems
(Mark Klein , Thomas Ralya , Bill Pollak , Ray Obenza , Michael González Harbour)
- Fixed priority pre-emptive scheduling: An historical perspective
(Neil C. Audsley, Alan Burns, Robert I. Davis, Ken W. Tindell & Andy J. Wellings)
- Priority Inversion on Mars
https://es.slideshare.net/jserv/priority-inversion-30367388
- L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization.
In IEEE Transactions on Computers, vol. 39, pp. 1175-1185, Sep. 1990.



URJC Escuela de Ingeniería de Fuenlabrada

RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

RoboticsLabURJC
Programming Robot Intelligence

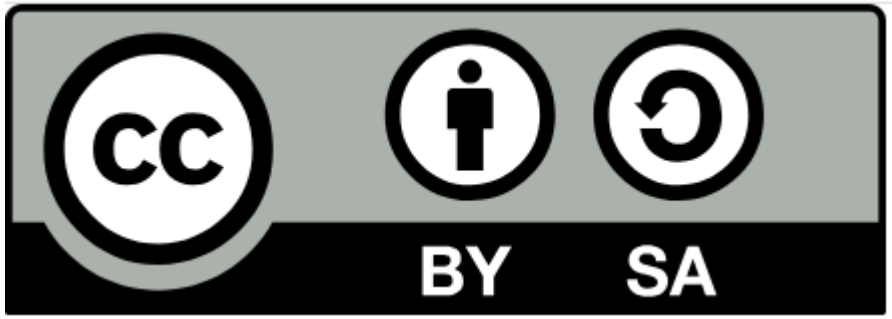
Sistemas Empotrados y de Tiempo Real

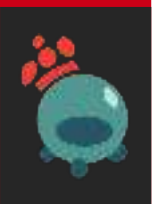
RTOS: Real Time Operative System

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

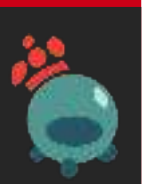
Roberto Calvo Palomino
roberto.calvo@urjc.es





Características

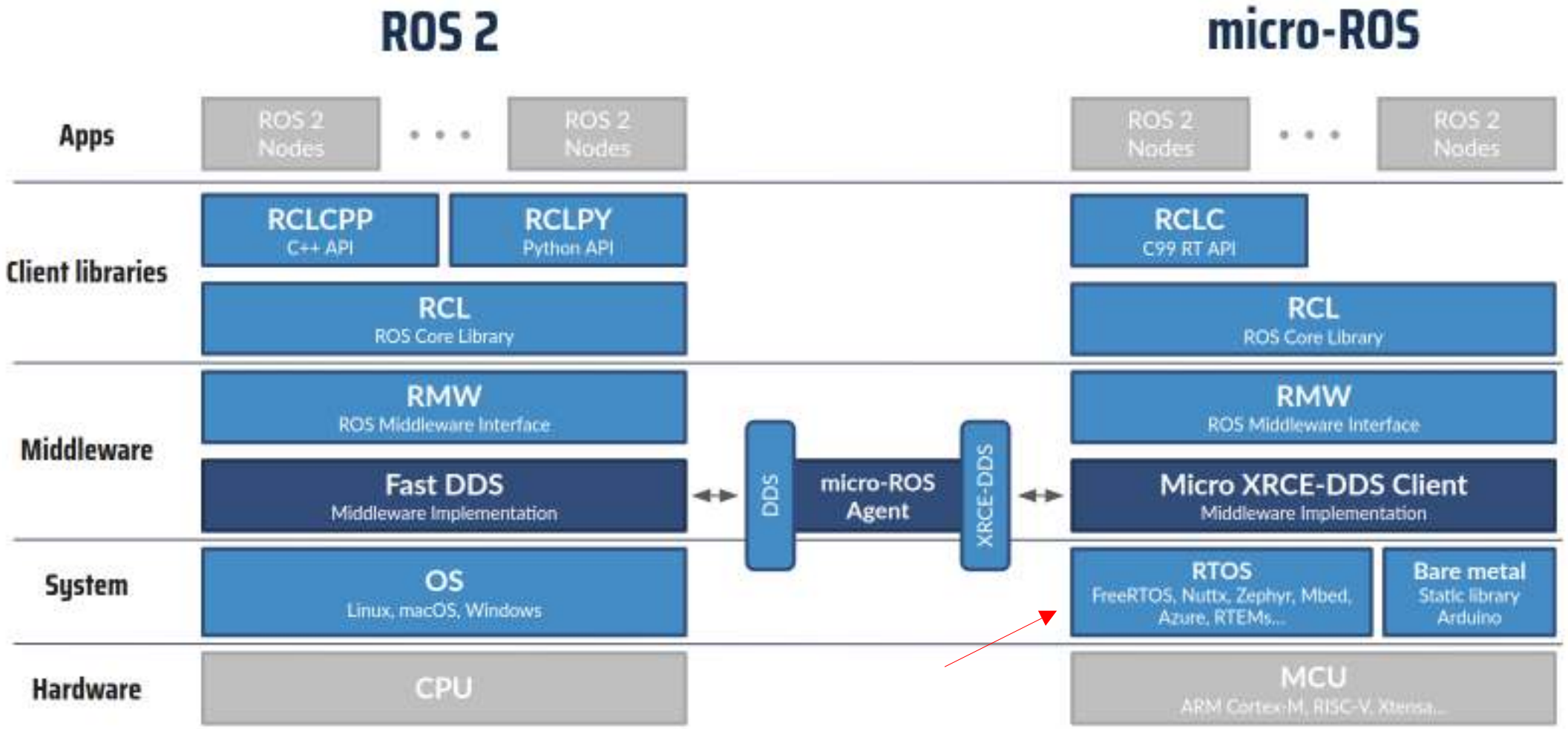
- Un RTOS es un sistema oparativo **multi-tarea** diseñado para ejecutar aplicaciones de tiempo real.
- Comportamiento predecible y determinista.
- Garantizar la ejecución completa de la tarea en un tiempo determinado.
- Debe implementar herencia de prioridades para evitar bloqueos y mal funcionamiento del sistema
- 2 tipos:
  - Soft RTOS
  - Hard RTOS



RTOS

- Existen multitud de RTOS
  - https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems
- Dependiendo del uso, características y plataforma soportada habrá RTOS que se ajusten más a nuestros requisitos
- Existen RTOS Open Source y propietarios.
- Compatibles con POSIX (Linux)
- FreeRTOS, VxWorks, QNX, RTLinux

RTOS en ROS 2

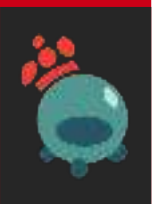

ROS 2
micro-ROS
Apps
ROS 2 Nodes
ROS 2 Nodes
ROS 2 Nodes
ROS 2 Nodes
Client libraries
RCLCPP
C++ API
RCLPY
Python API
RCLC
C99 RT API
RCL
ROS Core Library
RCL
ROS Core Library
Middleware
RMW
ROS Middleware Interface
RMW
ROS Middleware Interface
Fast DDS
Middleware Implementation
DDS
micro-ROS Agent
XRCE-DDS
Micro XRCE-DDS Client
Middleware Implementation
System
OS
Linux, macOS, Windows
RTOS
FreeRTOS, Nuttx, Zephyr, Mbed, Azure, RTEMs...
Bare metal
Static library
Arduino
Hardware
CPU
MCU
ARM Cortex-M, RISC-V, Xtensa...

ESP32 + FreeRTOS

ESP32 + FreeRTOS

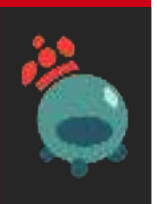
RTOS: Funciones básicas

- Manejo de tareas
  - Tiempo de ejecución, periodo y tiempo limite.
- Manejo de interrupciones
  - Síncronas y asíncronas
- Administración de memoria
  - Normalmente no hay memoria virtual, ni reserva dinámica.
- Excepciones
  - Timeouts, bloqueos, tiempo limite no cumplido, …
- Sincronización de tareas
  - Semáforos, mutex, spinlock, cerrojos para lectura/escritura
- Planificador de tareas
  - Basados en prioridad, RMS, EDFS, RR, …
- Manejo del tiempo
  - Reloj hardware programado para interrumpir al procesador a intervalos fijos.



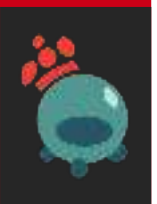
¿Qué RTOS elegir?

- Dependiendo de las características del sistema:
  - Menor **latencia** en cambio de contexto
  - Menor **latencia** en tratar interrupciones
  - Menor tamaño de kernel
  - Mayor flexibilidad en los algoritmos de planificación
  - Mayor rango de arquitectura suportadas
  - Soporte para Mono o multi núcleo
  - APIs starndars, POSIX, etc.

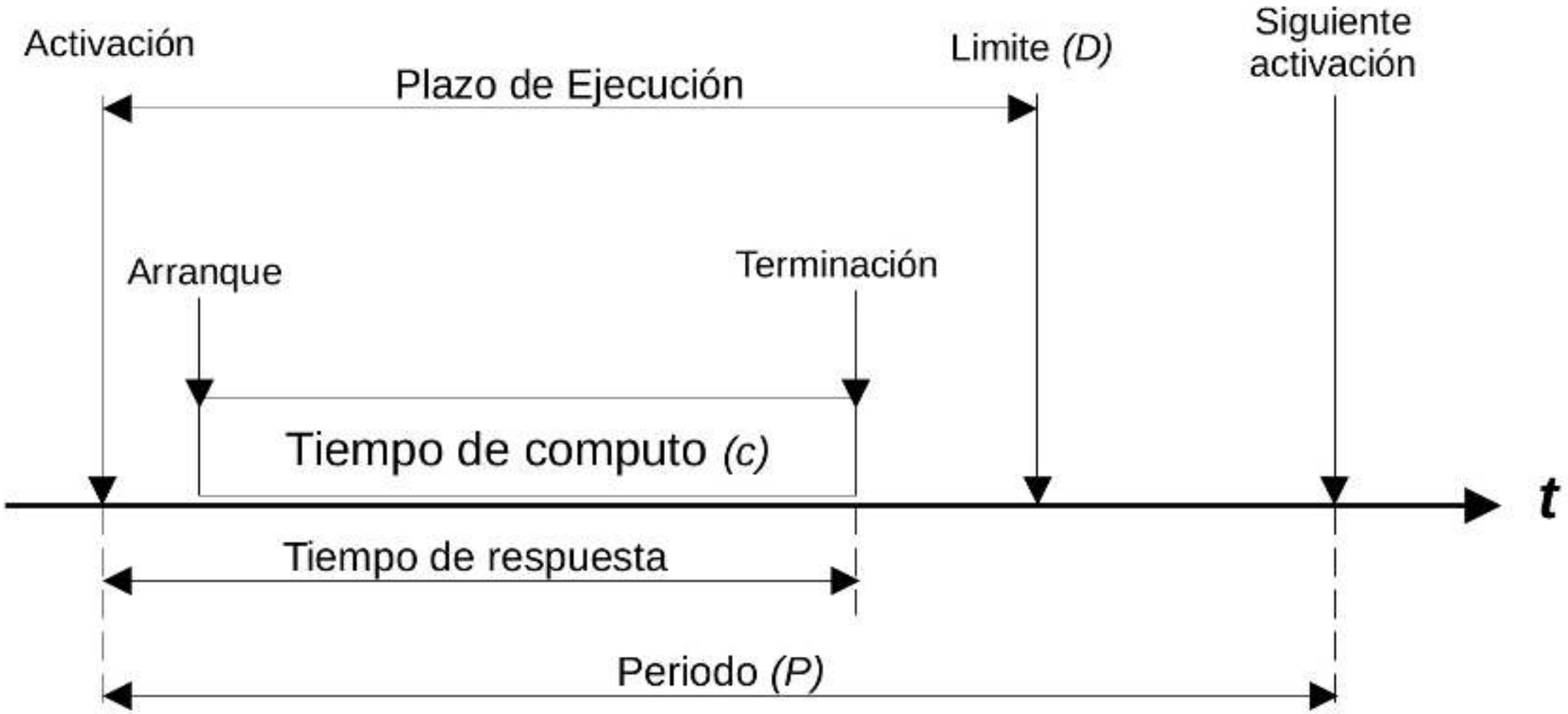


Latencia

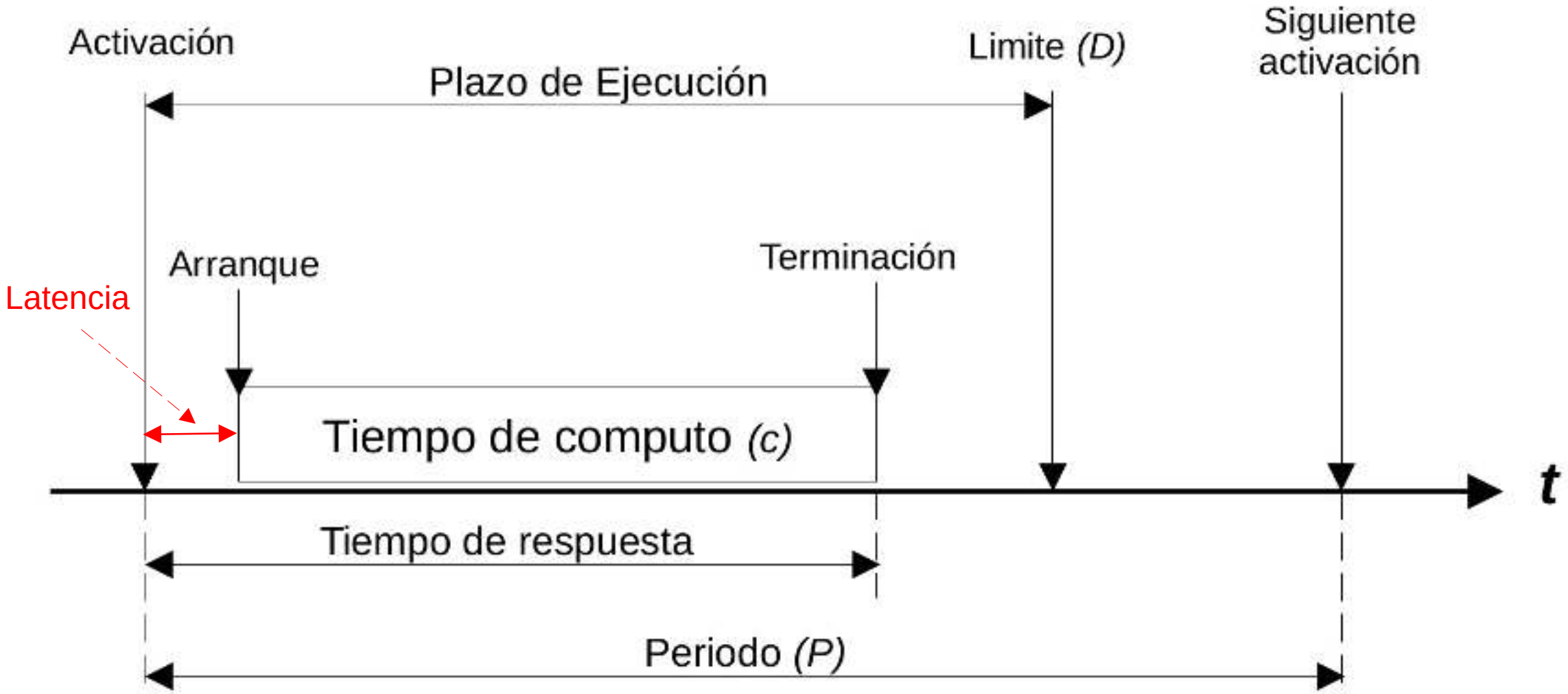
- **Latencia**: intervalo de tiempo entre estímulo y respuesta.
- Normalmente es el indicador más importante a la hora de elegir un RTOS
  - Latencia de interrupción: tiempo entre que una interrupción se genera y el instante que la interrupción es atendida (ISR: Interrupt Service Routine). Depende del microprocesador, controladores, sistema operativo, etc.
  - Latencia de planificación: tiempo entre que una tarea debe ejecutarse y el instante en el que verdaderamente se ejecuta.
- Las capacidades y rendimiento de un RTOS puede medirse mediante la latencia de planificación.



Latencia de planificación

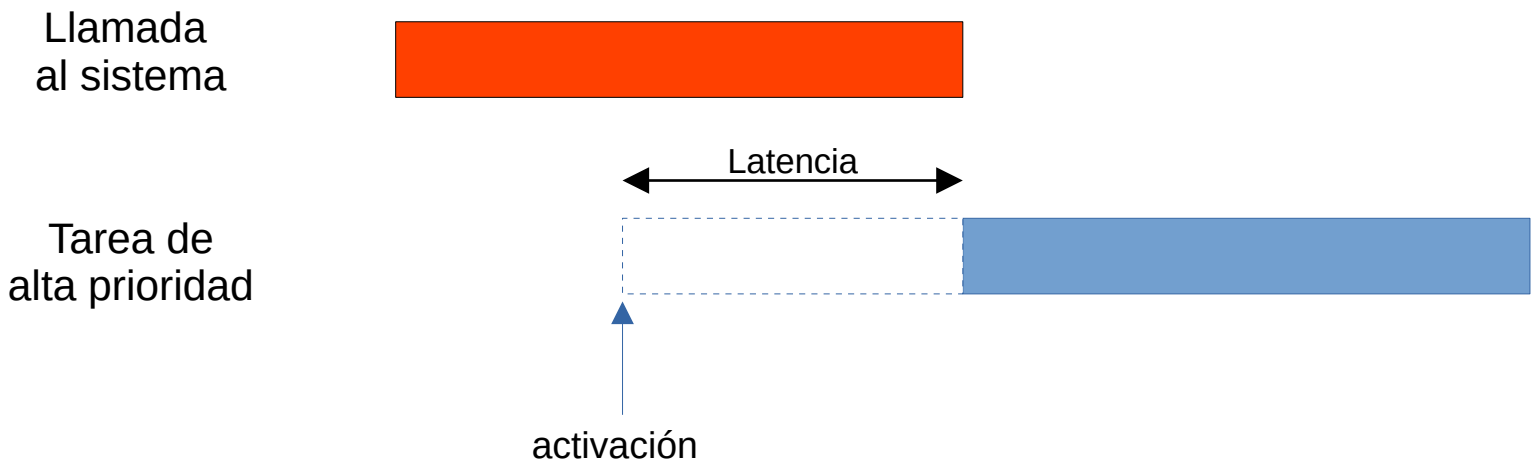


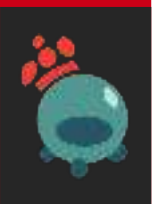
Latencia de planificación



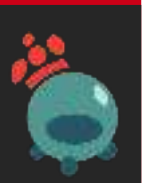
Causas de alta latencia

- Prioridades ilimitadas.
- Latencia en el algoritmo de planificación.
- Latencia en las interrupciones y en acceso a memoria.
- En SO generalistas, procesos que usan llamadas al sistema no son “expulsables” de la CPU en cualquier momento.





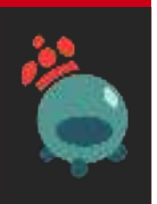
Kernel de Linux



Kernel de Linux

- **No** es un sistema operativo de tiempo real.
- El planificador no es determinista ni predictivo.
- Las **llamadas al sistema** tienen preferencia y no pueden expulsarse de la CPU en cualquier momento.
- Los procesos que ejecutan en **espacio de usuario** son siempre expulsables (preemptible).
  - Un bucle infinito en espacio de usuario no bloquea el sistema.

- Aún así, el kernel de linux es usado en dispositivos empotrados como routers y smartphones.

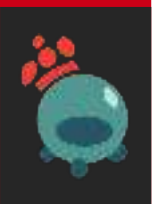


Kernel de Linux

SOFTWARE		
	OS	Ubuntu LTS 64-bits, RT Preempt
	Open source middleware	ROS LTS
	Periodic update/patches	✓
	Arm control mode	Position / velocity / effort control

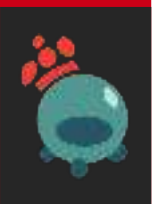
Scheduler en el kernel de Linux

- El planificador/scheduler de Linux en versiones mayores a 2.6.23 es **CFS: "Completely Fair Scheduler"**
- Este algoritmo tiene como objetivo último ser **justo** con todos los procesos del sistema y asegurar su ejecución.
- CFS utiliza una granularidad de nanosegundos y no hace uso de los "**timeslices**" de antiguos planificadores.
- Está basado en un árbol binario de búsqueda equilibrado (árbol rojo-negro, RBTREE).
- Complejidad de O(log(N)).



Scheduler en el kernel de Linux

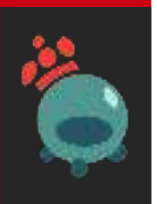
- **EEVDF: Earliest Eligible Virtual Deadline First**
  Implementa un algoritmo descrito en un paper 90's
- Incorporado a partir del kernel 6.6 como una opción. Por defecto sigue veniendo CFS
- Mismo objetivo CFS: Ser justo, pero mejorado.
- En principio no tiene prioridades, asigna deadlines virtuales para escoger qué tarea entra en ejecución.
- Optimizado para taras de baja latencia
- Mejora el funcionamiento de CFS en entonos multi-core.



CFS

- CFS implementa 3 colas de ejecución:
  - SCHED_NORMAL/ SCHED_OTHER: Usado para tareas regulares
  - SCHED_BATCH: No expulsa tan a menudo como en tareas regulares. Favorece tareas de ejecución largas que hace mejor uso de caches a costa de la interactividad.
  - SCHED_IDLE: Usado para tareas de muy baja prioridad.
- CFS ofrece políticas de “real-time”
  - SCHED_FIFO: se basa en política First-In, First-Out
  - SCHED_RR: se basa en política round robin (quantum fijo).
  - SCHED_DEADLINE: se basa en GEDF (Global Earliest Deadline First)

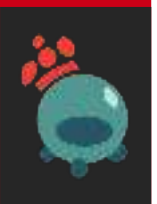
```
$ man sched
```



CFS

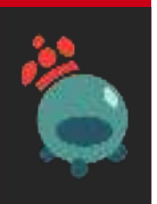
- Configurar nuestro thread para ejecutar en la cola SCHED_FIFO con prioridad 99

```
struct sched_param param;
param.sched_priority=99;

pthread_setschedparam (pthread_self(), SCHED_FIFO, &param);
```

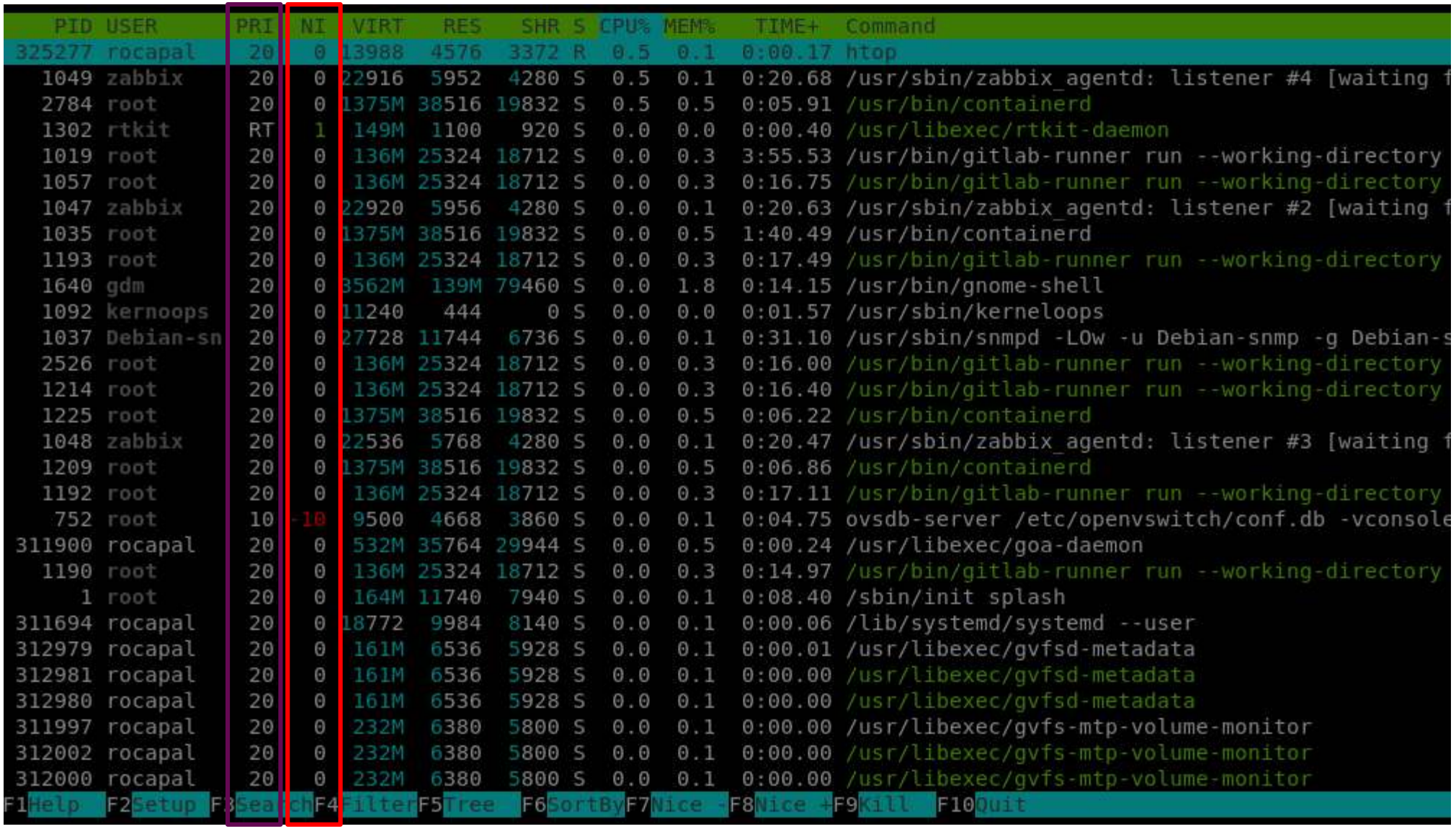


Prioridades y NICE

- El valor de **prioridad** de un proceso se visualiza como PR o PRI (usando *ps* o *htop*). Es el valor de prioridad que el kernel/planificador utiliza.
  - Prioridad a nivel de espacio de usuario 100-139
  - Prioridad real-time: 1 (baja) a 99(alta) (SCHED_FIFO, SCHED_RR)
- **Nice**: Valor utilizado en espacio de usuario para cambiar la prioridad de un proceso
  - Rango de -20 (alta) a 19(baja) , 0 por defecto.
- Prioridad = 20 + Nice , nice=[-20, 19]
  - Los procesos linux tiene una prioridad de **20** por defecto (nice=0)

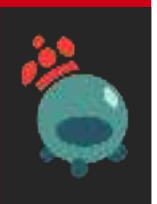


Prioridades y NICE



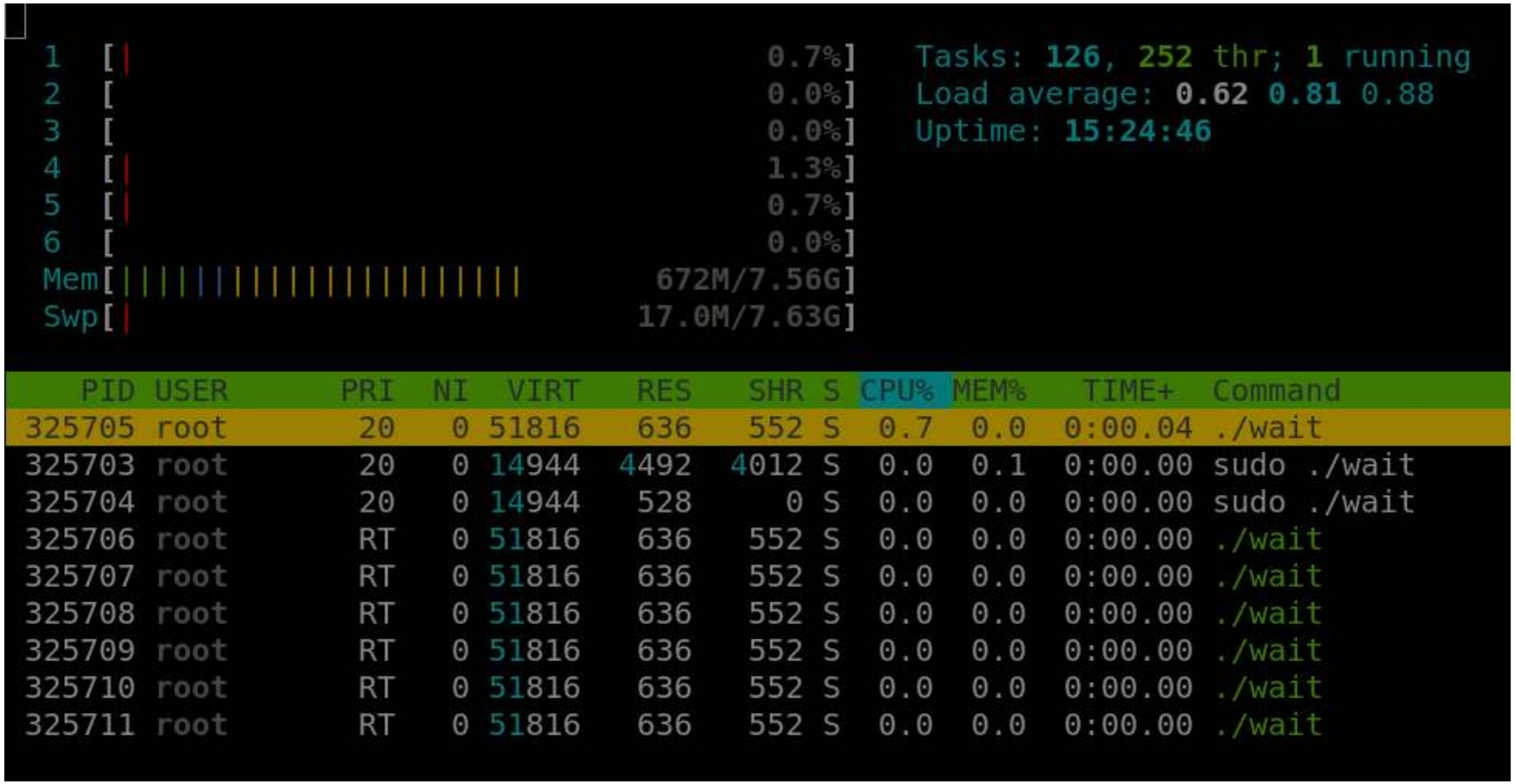
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
325277	rocapal	20	0	13988	4576	3372	R	0.5	0.1	0:00.17	htop
1049	zabbix	20	0	22916	5952	4280	S	0.5	0.1	0:20.68	/usr/sbin/zabbix_agentd: listener #4 [waiting f
2784	root	20	0	1375M	38516	19832	S	0.5	0.5	0:05.91	/usr/bin/containerd
1302	rtkit	RT	1	149M	1100	920	S	0.0	0.0	0:00.40	/usr/libexec/rtkit-daemon
1019	root	20	0	136M	25324	18712	S	0.0	0.3	3:55.53	/usr/bin/gitlab-runner run --working-directory
1057	root	20	0	136M	25324	18712	S	0.0	0.3	0:16.75	/usr/bin/gitlab-runner run --working-directory
1047	zabbix	20	0	22920	5956	4280	S	0.0	0.1	0:20.63	/usr/sbin/zabbix_agentd: listener #2 [waiting f
1035	root	20	0	1375M	38516	19832	S	0.0	0.5	1:40.49	/usr/bin/containerd
1193	root	20	0	136M	25324	18712	S	0.0	0.3	0:17.49	/usr/bin/gitlab-runner run --working-directory
1640	gdm	20	0	3562M	139M	79460	S	0.0	1.8	0:14.15	/usr/bin/gnome-shell
1092	kernoops	20	0	11240	444	0	S	0.0	0.0	0:01.57	/usr/sbin/kerneloops
1037	Debian-sn	20	0	27728	11744	6736	S	0.0	0.1	0:31.10	/usr/sbin/snmpd -LOw -u Debian-snmp -g Debian-s
2526	root	20	0	136M	25324	18712	S	0.0	0.3	0:16.00	/usr/bin/gitlab-runner run --working-directory
1214	root	20	0	136M	25324	18712	S	0.0	0.3	0:16.40	/usr/bin/gitlab-runner run --working-directory
1225	root	20	0	1375M	38516	19832	S	0.0	0.5	0:06.22	/usr/bin/containerd
1048	zabbix	20	0	22536	5768	4280	S	0.0	0.1	0:20.47	/usr/sbin/zabbix_agentd: listener #3 [waiting f
1209	root	20	0	1375M	38516	19832	S	0.0	0.5	0:06.86	/usr/bin/containerd
1192	root	20	0	136M	25324	18712	S	0.0	0.3	0:17.11	/usr/bin/gitlab-runner run --working-directory
752	root	10	-10	9500	4668	3860	S	0.0	0.1	0:04.75	ovsdb-server /etc/openvswitch/conf.db -vconsole
311900	rocapal	20	0	532M	35764	29944	S	0.0	0.5	0:00.24	/usr/libexec/goa-daemon
1190	root	20	0	136M	25324	18712	S	0.0	0.3	0:14.97	/usr/bin/gitlab-runner run --working-directory
1	root	20	0	164M	11740	7940	S	0.0	0.1	0:08.40	/sbin/init splash
311694	rocapal	20	0	18772	9984	8140	S	0.0	0.1	0:00.06	/lib/systemd/systemd --user
312979	rocapal	20	0	161M	6536	5928	S	0.0	0.1	0:00.01	/usr/libexec/gvfsd-metadata
312981	rocapal	20	0	161M	6536	5928	S	0.0	0.1	0:00.00	/usr/libexec/gvfsd-metadata
312980	rocapal	20	0	161M	6536	5928	S	0.0	0.1	0:00.00	/usr/libexec/gvfsd-metadata
311997	rocapal	20	0	232M	6380	5800	S	0.0	0.1	0:00.00	/usr/libexec/gvfs-mtp-volume-monitor
312002	rocapal	20	0	232M	6380	5800	S	0.0	0.1	0:00.00	/usr/libexec/gvfs-mtp-volume-monitor
312000	rocapal	20	0	232M	6380	5800	S	0.0	0.1	0:00.00	/usr/libexec/gvfs-mtp-volume-monitor

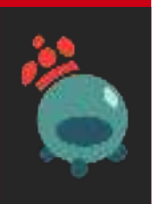
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice - F8Nice + F9Kill F10Quit



Prioridades y NICE

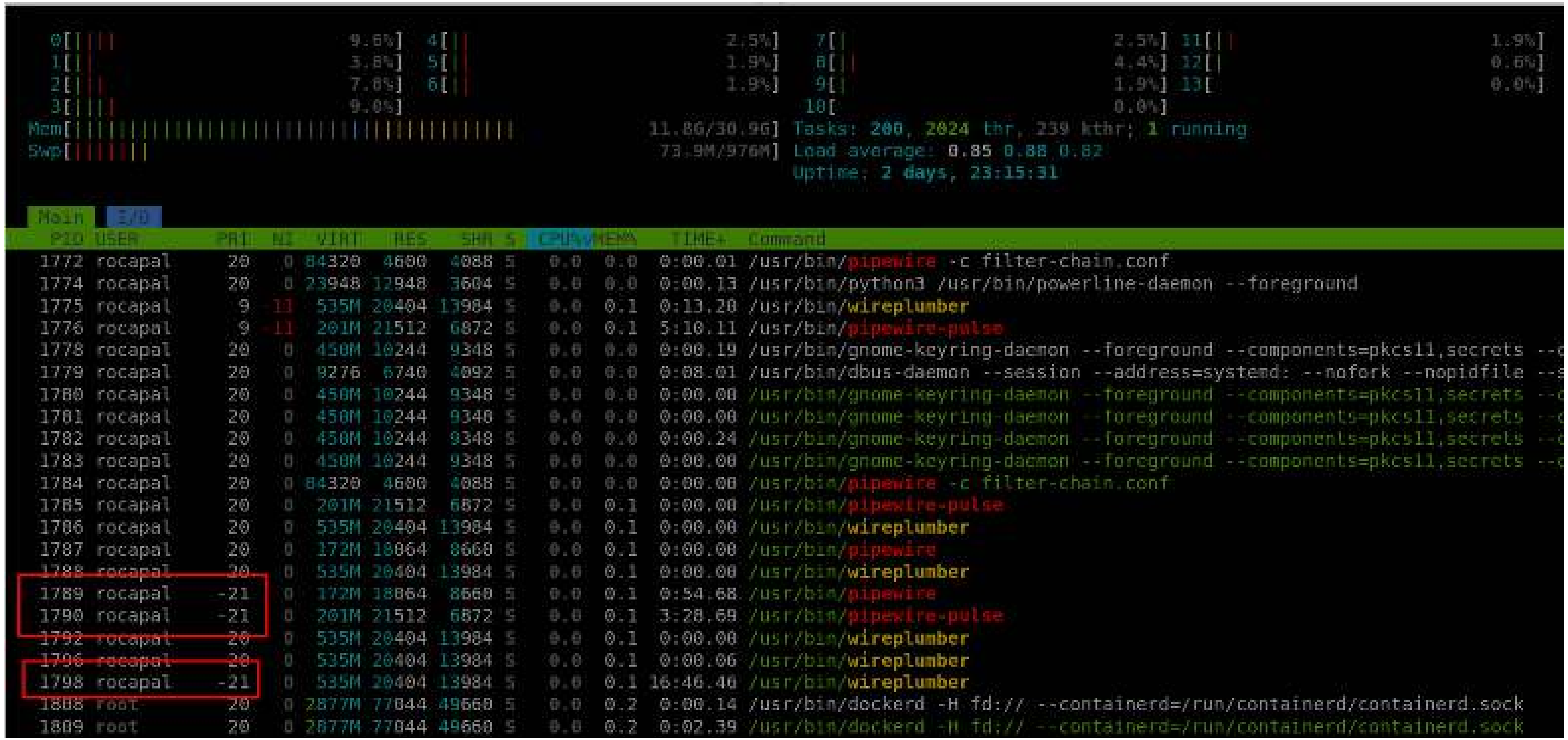
- Cuando un proceso están en la cola de prioridades de real time (SCHED_RR ó SCHED_FIFO), aparece **RT** en la columna *PRI* de *htop*.





Prioridades y NICE

- En versiones más modernas podría aprecer como



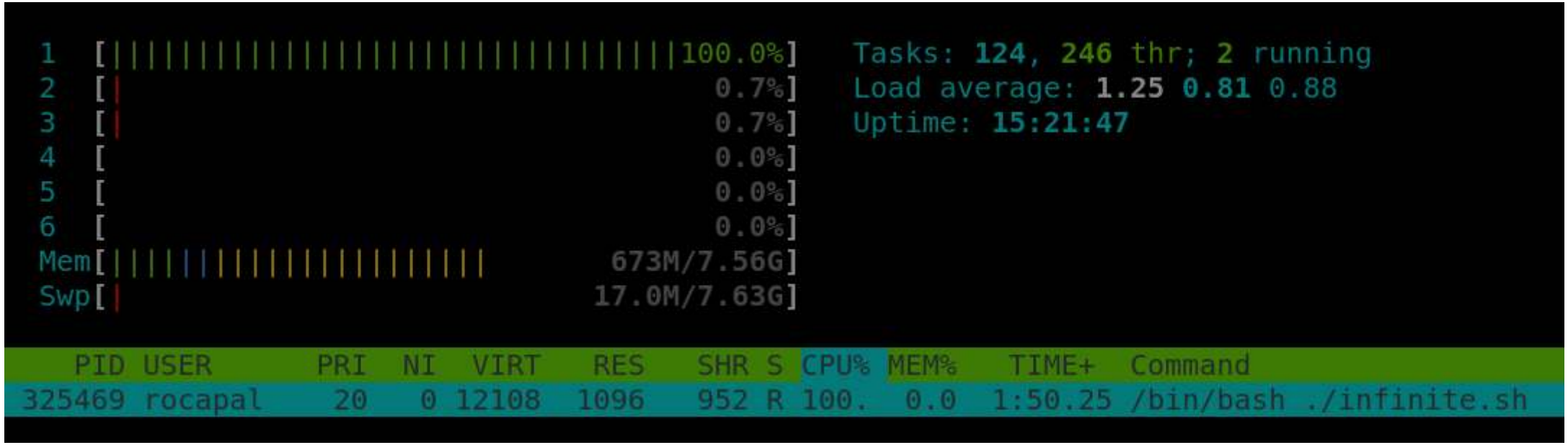
Prioridades y NICE

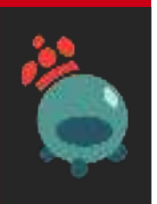
- Utiliza chrt (man) para comprobar más información sobre colas de ejecución y prioridades

```
base rocapal vega ~ $ chrt -p 1789
política actual de planificación del pid 1789: SCHED_RR|SCHED_RESET_ON_FORK
política actual de planificación del pid 1789: 20
```

Prioridades y NICE

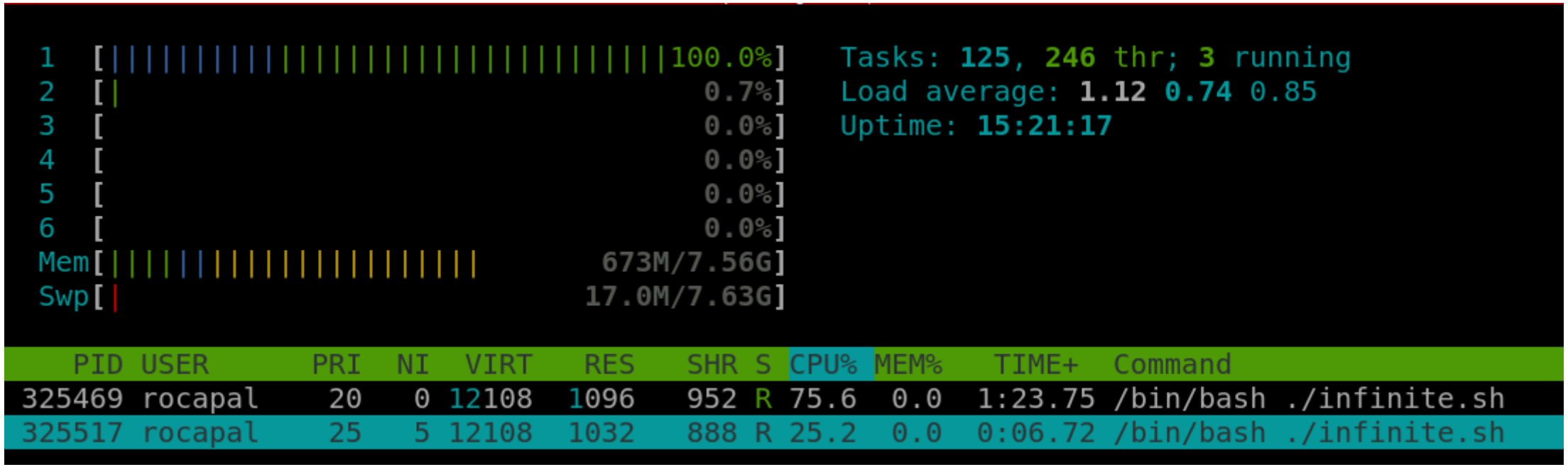
- Lanzamos un proceso en la CPU-0
  - taskset -c 0 ./infinite.sh &



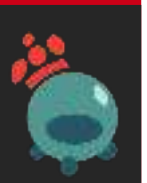


Prioridades y NICE

- Lanzamos otra instancia del mismo proceso con un valor de **nice** especifico.
  - taskset -c 0 nice -n 5 ./infinite.sh &



Gestion de tiempos



Gestion de tiempos: Relojes

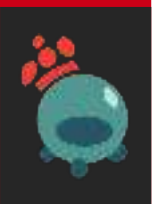
- Relojes **Hardware**
  - RTC (Real Time Clock), mantiene la gestión del tiempo cuando el equipo está apagado.

- Relojes **Software** (system clock o kernel clock)
  - CLOCK_REALTIME: Reloj del sistema. Puede sufrir ajustes para corregir la fecha (NTP).
  - CLOCK_MONOTONIC : Igual que CLOCK_REALTIME pero no se realizan ajustes, por tanto su cuenta es creciente sin saltos bruscos. Avanza constantemente por cada tick.
    Útil para medir duraciones entre eventos.

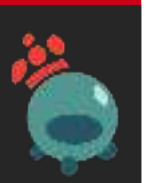
Gestion de tiempos: Relojes

- CLOCK_MONOTONIC vs CLOCK_REALTIME

```
clock_gettime(CLOCK_MONOTONIC, &begin);
[...]
// Transcurren 1000 ms
clock_gettime(CLOCK_MONOTONIC, &end);
end – begin ~= 1000 ms



clock_gettime(CLOCK_REALTIME, &begin);
[...]
// Transcurren 1000 ms
clock_gettime(CLOCK_REALTIME, &end);
end – begin pueden no ser 1000 ms
```

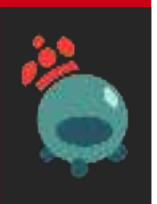


Kernel de Linux de Tiempo Real RTLinux



Historia

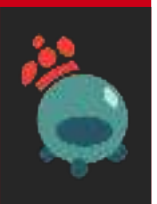
- **RTLinux** fue inicialmente desarrollado por Michael Barabanov y Victor Yodaiken, y fue adquirido por Wind River en 2007.
- Primeras versiones ofrecía un API muy reducido y no POSIX.
- En Octubre de 2015 el proyecto pasa a estar bajo la **Linux Foundation**[1] junto con la colaboración de Google, Intel, IBM, Qualcomm o ScanDisk
- RTLinux no es un código independiente, es un parche que se aplica sobre una versión del kernel de linux
  - `https://git.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt.git`



[1] `https://www.linuxfoundation.org/press-release/2015/10/the-linux-foundation-announces-project-to-advance-real-time-linux/`

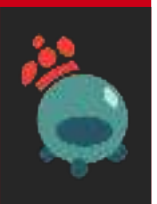
RT Linux

- Permite un sistema híbrido de:
  - Tareas de tiempo real
  - Tareas regulares
- La mayoría del código realizado en Linux puede ejecutarse con muy pocas modificaciones en RTLinux (POSIX)
- Minimiza la latencia lo máximo posible
- Minimiza el código de kernel no PREEMPT.
- Sistema operativo de tiempo real estricto.
- Extensiones para entorno multiprocesador SMP.



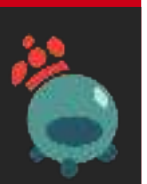
RT Linux

- ¿Dónde está la magia del parche?
- Fuerza la interrupción de threads
  - Permite priorizar los controladores de interrupciones.
- Permitir que los locks se duerman
  - rt_spinlocks, rt_mutexes, semaphores, …
- Eliminar secciones críticas
  - Evitar zonas no expulsables
  - Interrupciones
  - Spinlocks
- Herencia de prioridad



RT Linux

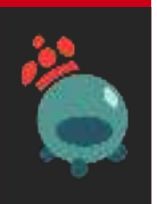
- Sistema de tiempo real duro / 95%
- Latencia estable pero con algunos picos
  - CPU idle states
    - /sys/devices/system/cpu/cpu0/cpuidle/
  - Voltaje y frecuencia dinámica
  - Multi-Threading (cambio de contexto)
  - Hardware



Políticas de Tiempo Real

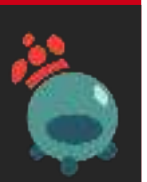
- SCHED_FIFO, SCHED_RR
- SCHED_DEADLINE: se basa en GEDF (Global Earliest Deadline First)
  - Modelo para tareas esporádicas
    - Requisitos temporales críticos, activaciones a instantes aleatorios.

```
struct sched_attr task;
pid_ttid = syscall(SYS_gettid);
task.size = sizeof(struct sched_attr);
task.sched_policy = SCHED_DEADLINE;
task.sched_flags = 0;
task.sched_nice = 0;
task.sched_priority = 0;
task.sched_runtime = 100 * 1000;
task.sched_deadline = 200 * 1000;
task.sched_period = 200 * 1000;

if (sched_setattr(tid, &task, 0) < 0)
    handle_err("Could not set SCHED_DEADLINE attributes");
```



Políticas de Tiempo Real

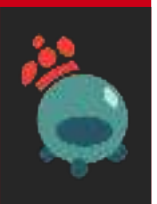
- SCHED_DEADLINE:
  - Muy utilizado en sistemas de tiempo real
  - Puede contar las veces que se no se ha cumplido el deadline.
  - NO es capaz de controlar tareas que no han cumplido el deadline.



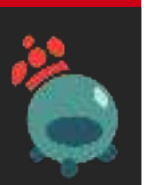
Non PREEMPT y PREEMPT RT

```
$ uname -a

Linux f-l3202-pc10 5.4.0-47-generic #51-Ubuntu
SMP Fri Sep 4 19:50:52 UTC 2020 x86_64 x86_64
x86_64 GNU/Linux
```

```
$ uname -a

Linux f-l3202-pc11 5.4.66-rt38 #1 SMP PREEMPT_RT
Sun Oct 4 19:59:00 CEST 2020 x86_64 x86_64 x86_64
GNU/Linux
```

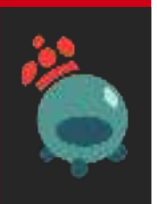


EJEMPLO RT



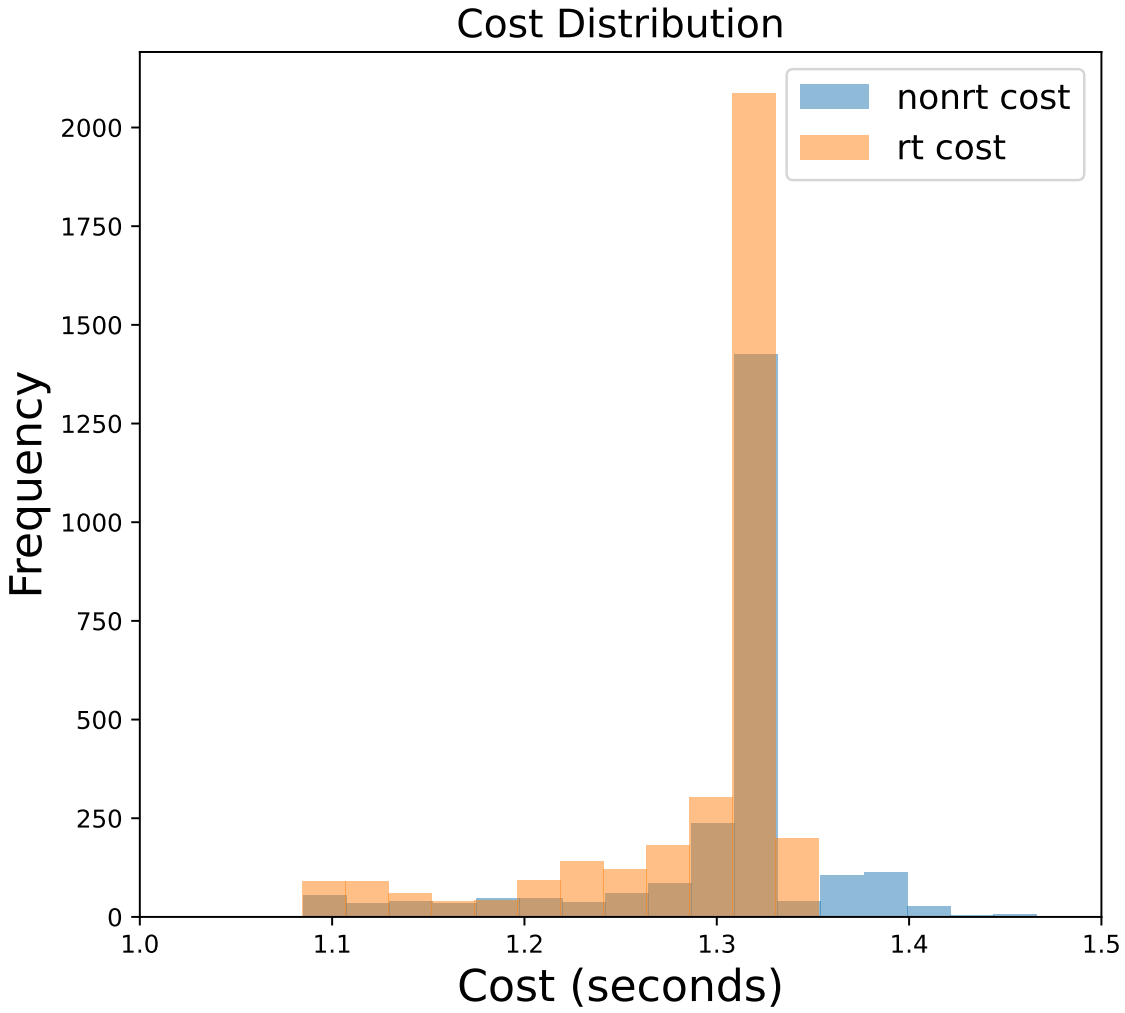
Parpadeo de pantalla

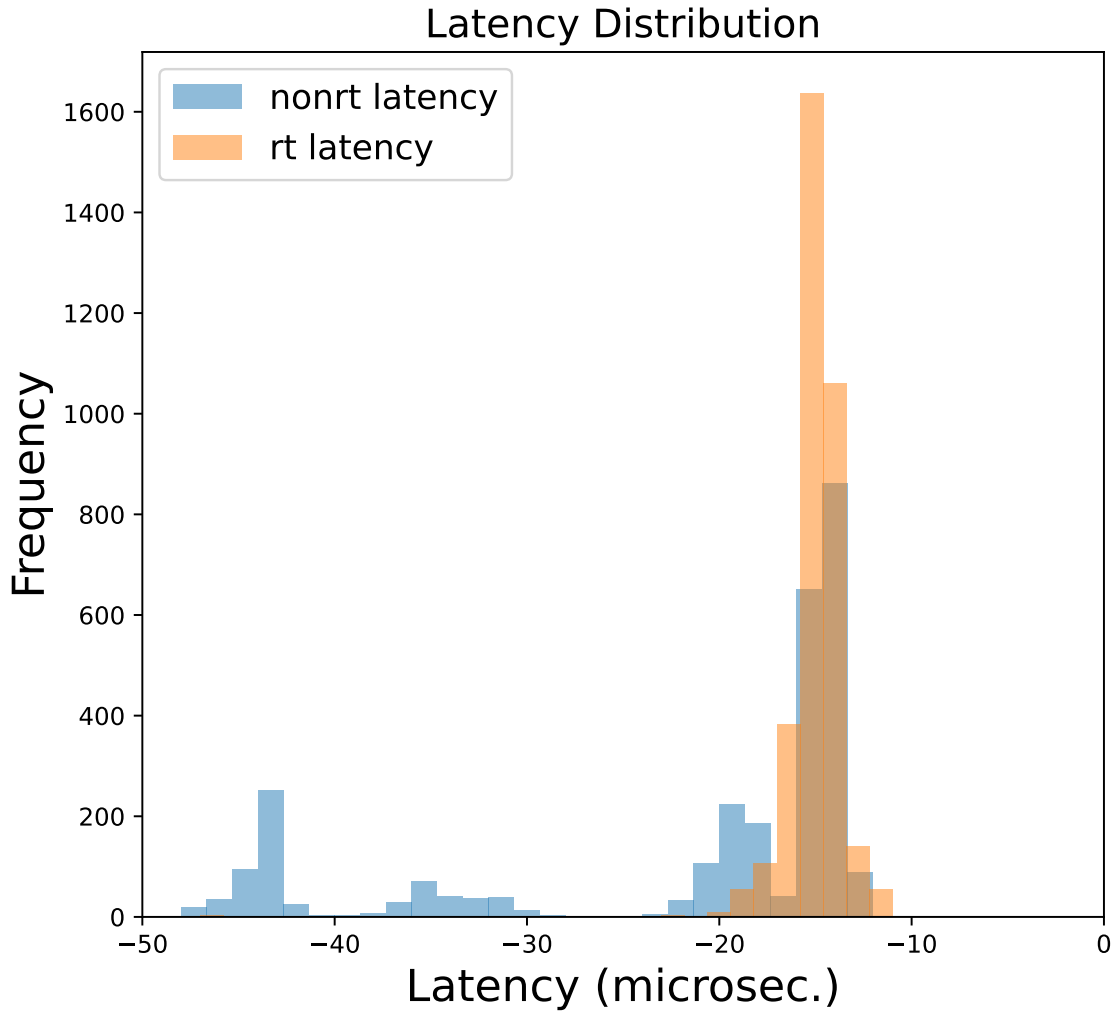
- Ejemplo de código que hace parpadera la pantalla durante ~1 segundo, durante ese segundo el programa realiza un uso de CPU alto.
- Además, realiza un cálculo en cuanto a la latencia de planificación.
- Por tanto tendremos 2 métricas a analizar:
  - Coste del procesamiento
  - Latencia de planificación



Parpadeo Pantalla

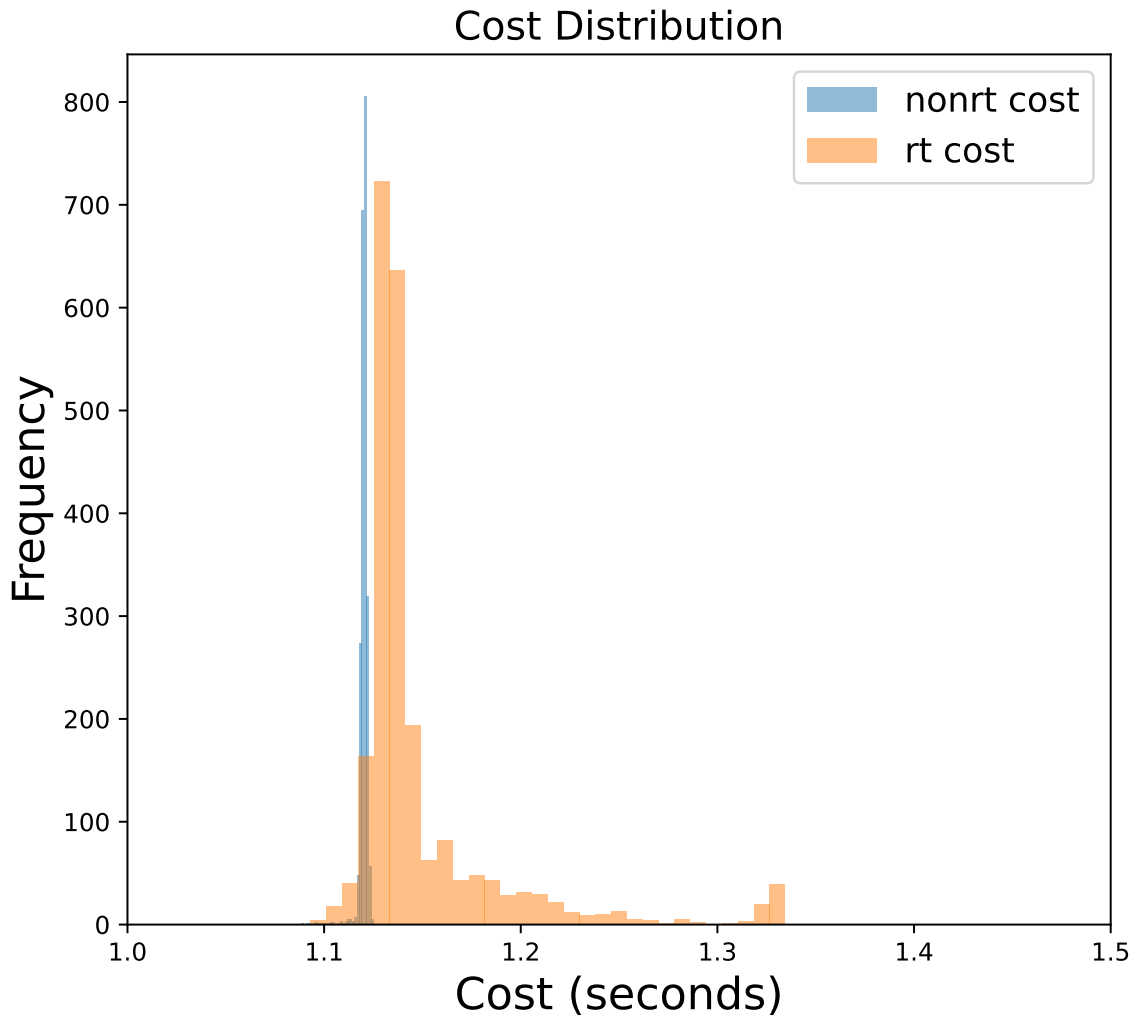
- IDLE

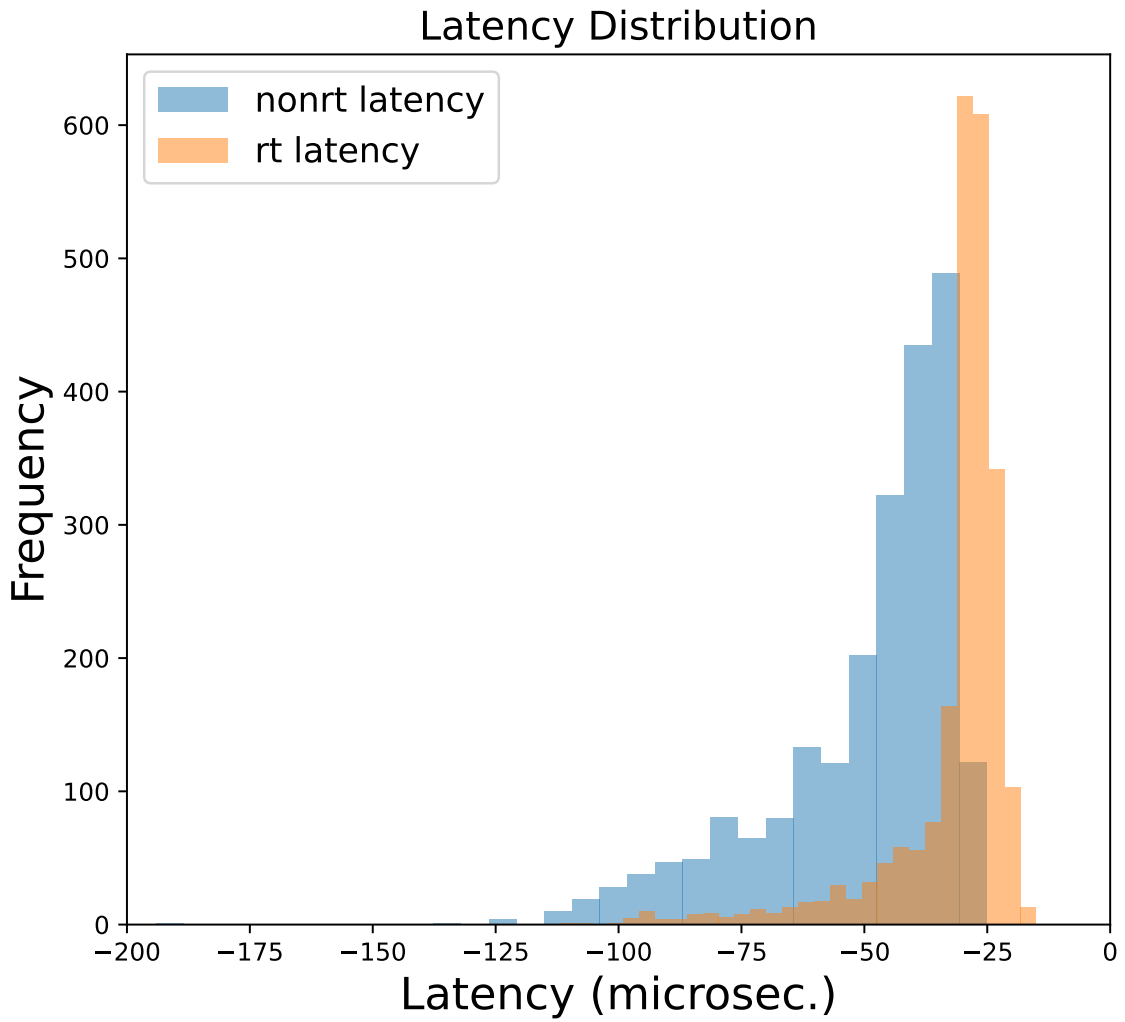




Parpadeo Pantalla

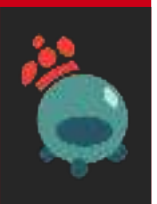
- HACKBENCH



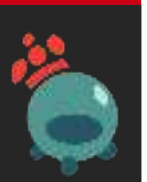


Buenos prácticas programación RT

- Manejo de memoria eficiente (no malloc / free)
- Manejo de excepciones
- Operaciones I/O
- Evitar inversión de prioridades
- Evita spinlocks
- Fork() no es aconsejable su uso.

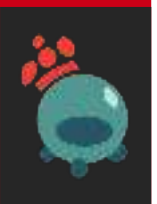


¿Cómo podemos medir la latencia de planificación de un sistema Linux ?



Cyclictest

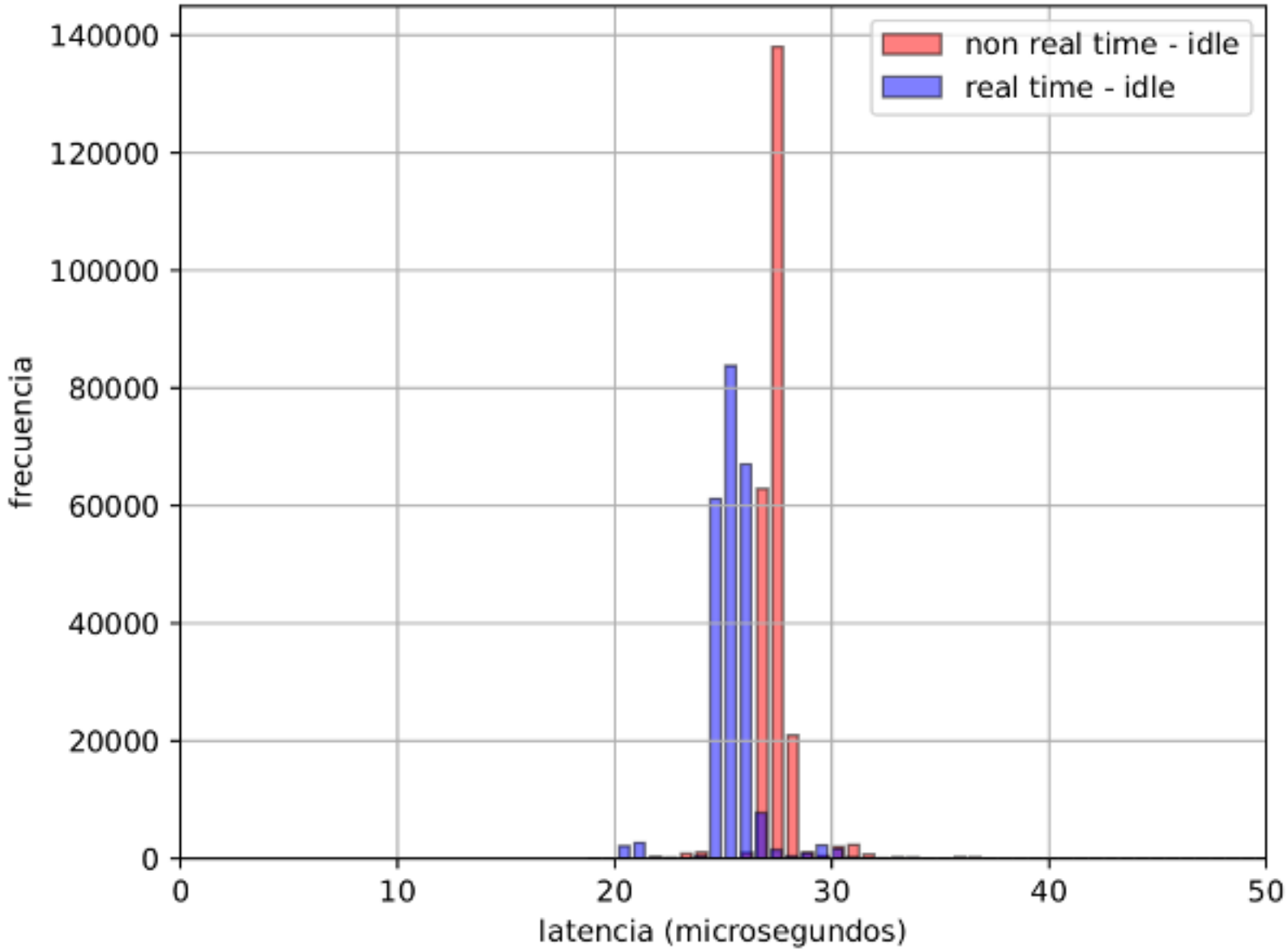
- Utilidad para medir la latencia en sistemas operativos
- Mide la latencia a nivel de planificador
  - Tiempo entre que una tarea se despierta (wake up) y el verdadero instante en el que empieza a ejecutar
- Adoptado como banco de pruebas para entornos RT basados en GNU/Linux.
- La latencia se debe medir en diferentes escenarios
  - IDLE
  - Numerosas operaciones I/O (interrupciones)
  - Multitud de procesos ejecutando en los cores (stress o hackbench)



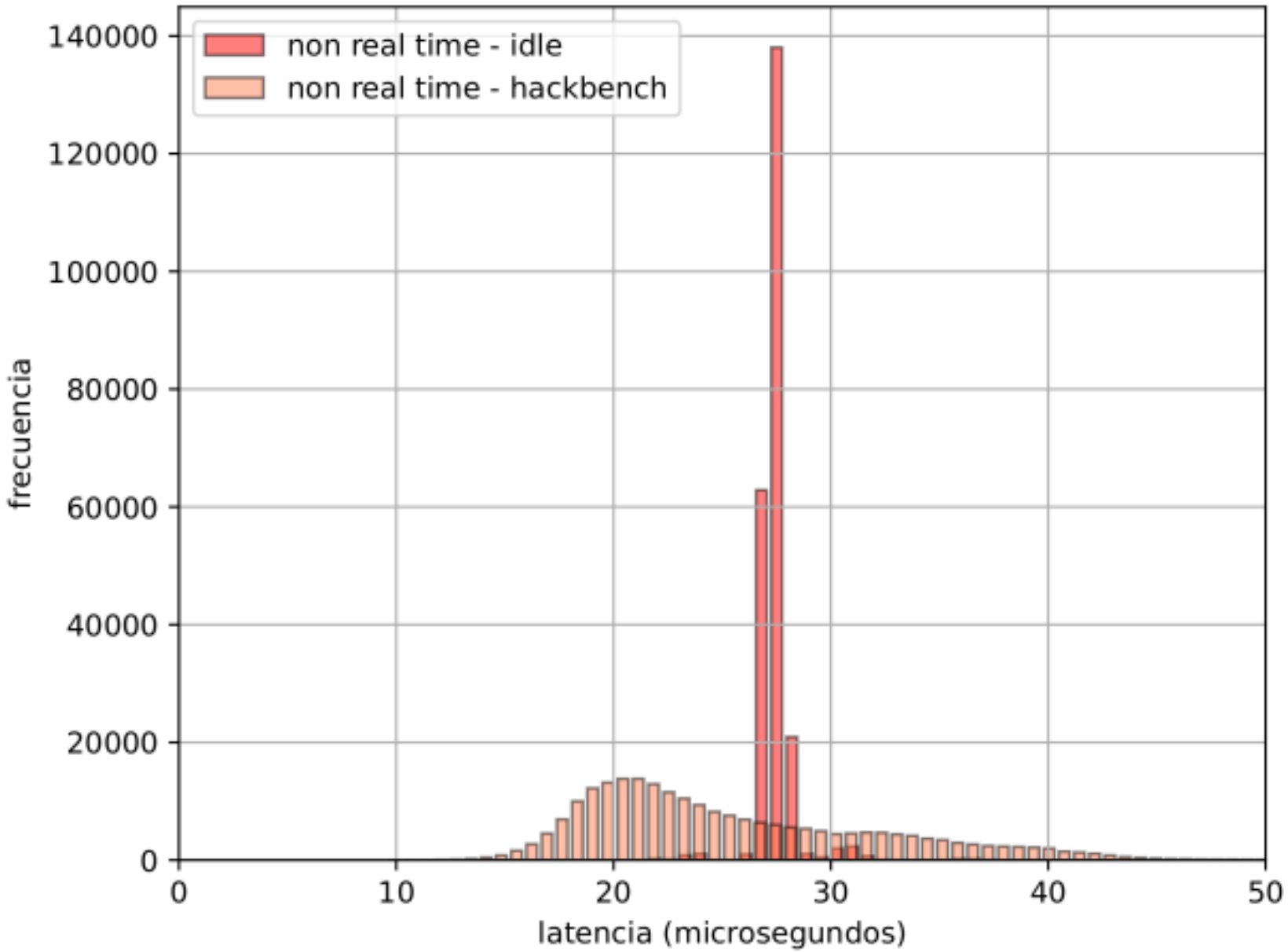
Otras herramientas

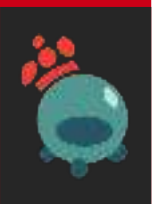
- **stress**: Utilidad para generar sobrecarga computacional en el sistema
- **hackbench**: Utilidad para testear y estresar el planificador.
  - Define número de threads o procesos hijo.
  - Define el número de mensajes intercambiados entre threads.
  - Mensajes enviados a través de pipes en vez de sockets.
- **boonie++**: Utilidad para testear I/O y sistema de ficheros.
  - Define modos de escritura (buffering, no buffering).
  - Genera muchas operaciones I/O con sincronización protegida por semáforos.

Histograma

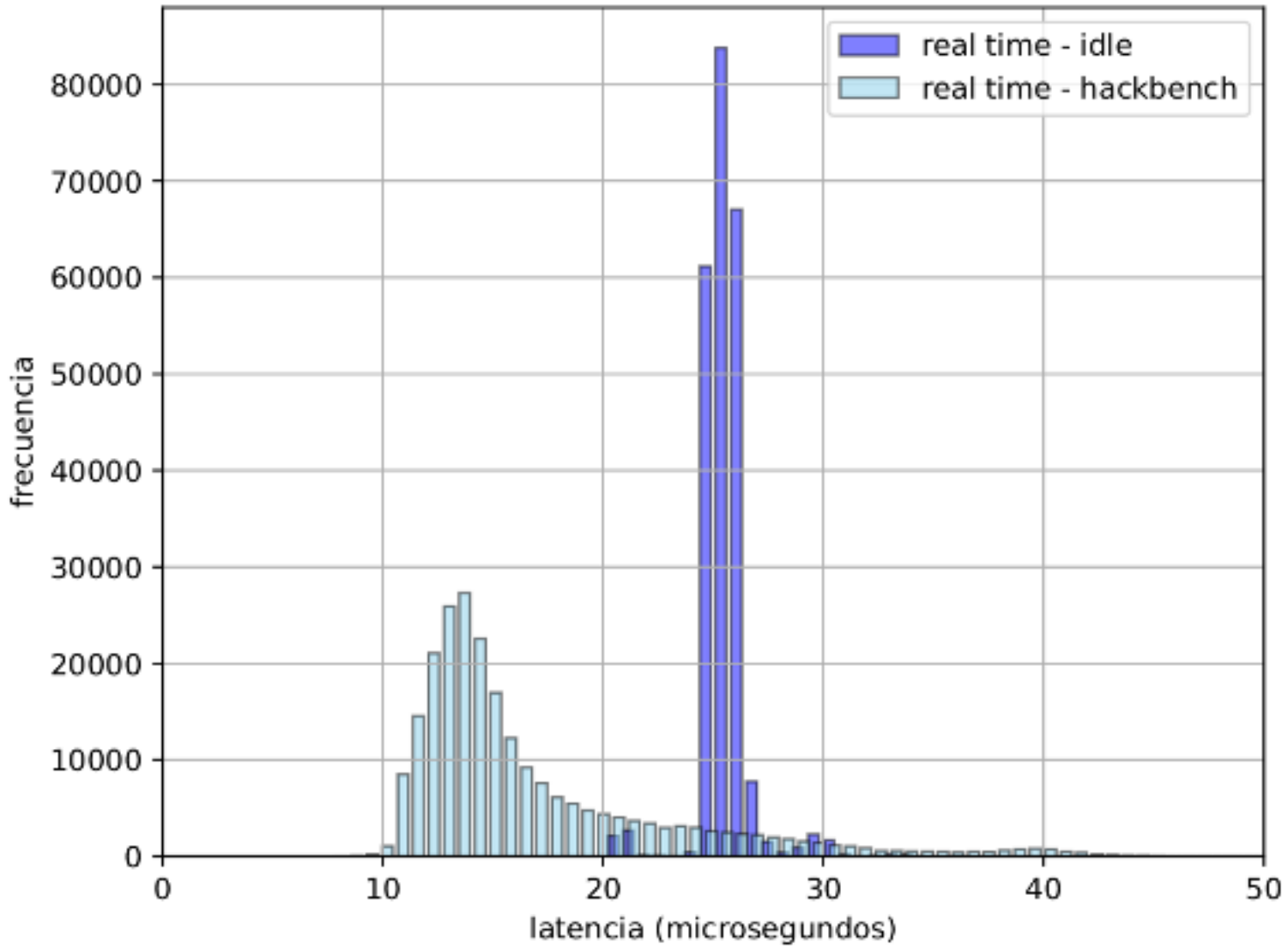

non real time - idle
real time - idle
frecuencia
latencia (microsegundos)
0
20000
40000
60000
80000
100000
120000
140000
0
10
20
30
40
50

Histograma

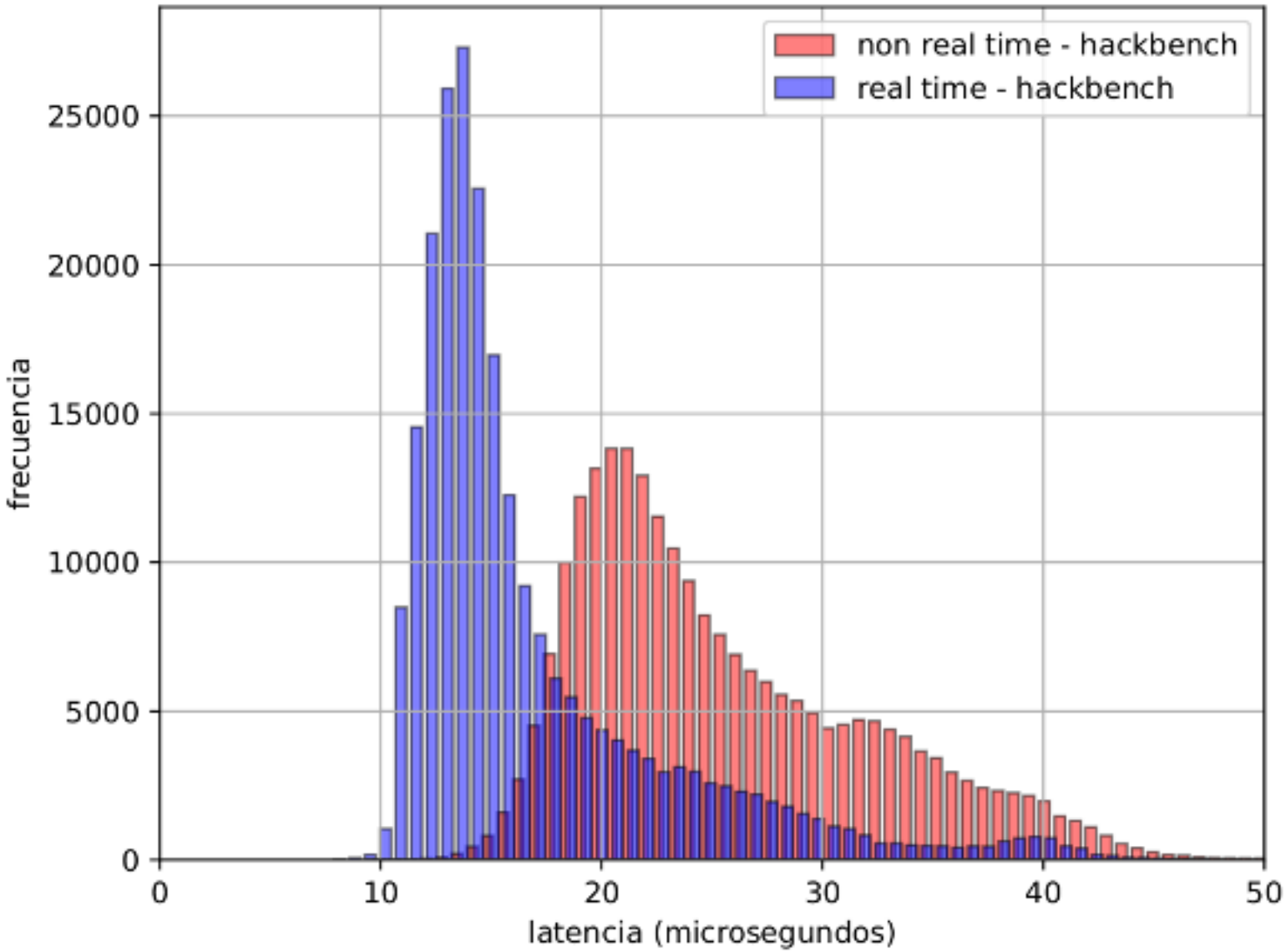

non real time - idle
non real time - hackbench
frecuencia
latencia (microsegundos)
0
20000
40000
60000
80000
100000
120000
140000
0
10
20
30
40
50



Histograma


real time - idle
real time - hackbench
frecuencia
80000
70000
60000
50000
40000
30000
20000
10000
0
0
10
20
30
40
50
latencia (microsegundos)

Histograma


non real time - hackbench
real time - hackbench
frecuencia
25000
20000
15000
10000
5000
0
0
10
20
30
40
50
latencia (microsegundos)

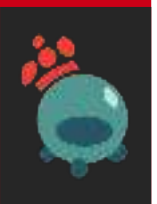
RTLinux en Ubuntu y RaspBerryPi

- Ubuntu
  - https://chenna.me/blog/2020/02/23/how-to-setup-preempt-rt-on-ubuntu-18-04/

- RaspBerryPi
  - https://www.get-edi.io/Real-Time-Linux-on-the-Raspberry-Pi/
  - https://metebalci.com/blog/latency-of-raspberry-pi-4-on-standard-and-real-time-linux-4.19-kernel/
  - https://lemariva.com/blog/2019/09/raspberry-pi-4b-preempt-rt-kernel-419y-performance-test

Bibliografía

- CFS Scheduler
  - https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt
- A Comparison of Scheduling Latency in Linux, PREEMPTRT, and LITMUS
  - https://people.mpi-sws.org/~bbb/papers/pdf/ospert13.pdf
- The Real Time Linux Project
  - https://wiki.linuxfoundation.org/realtime/start
- A realtime preemption overview
  - https://lwn.net/Articles/146861/



URJC Escuela de Ingeniería de Fuenlabrada

RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC Programming Robot Intelligence

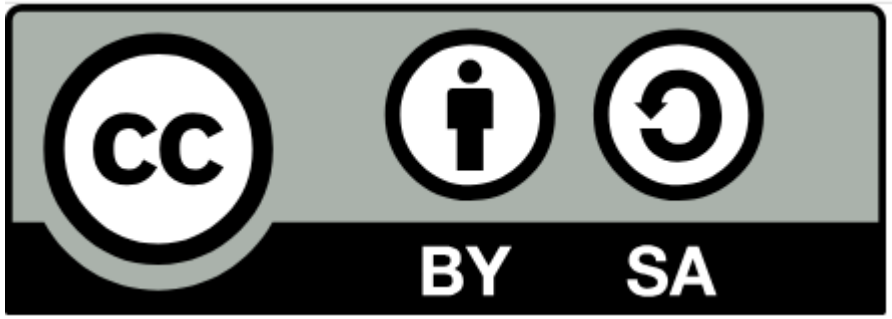
Sistemas Empotrados y de Tiempo Real

Microcontroladores

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

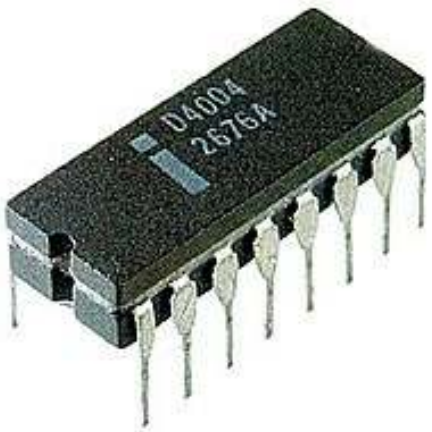
Roberto Calvo Palomino
roberto.calvo@urjc.es



Microcontrolador vs Microprocesador

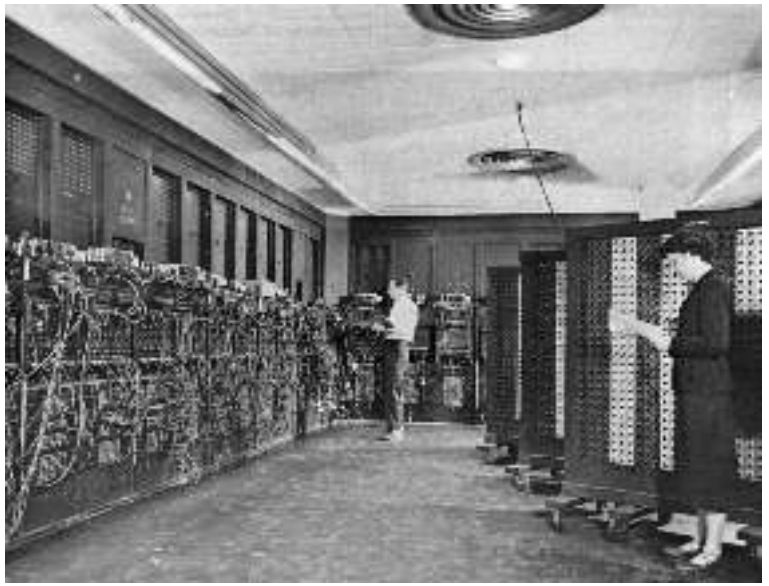
Micro-procesadores

- Intel 4004: Primero microprocesador comercial (4 bits) ensamblado en un chip (1971)
- 740 kHz
- 2.300 transistores



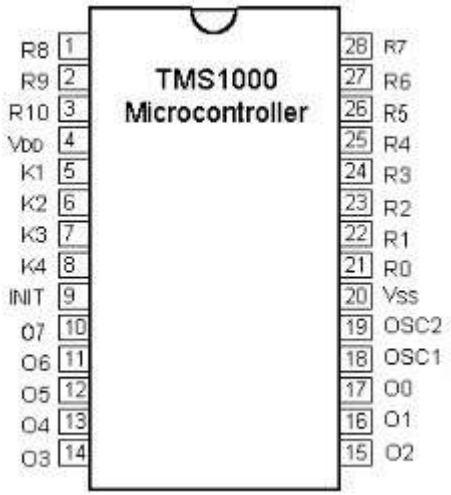
Micro-procesadores

- ¿Y antes de los micro-procesadores?
- ENIAC o Electronic Numerical Integrator And Computer
  - Primeros ordenadores de propósito general.
  - Diseñada para calcular tablas de tiro de artillería destinadas al Laboratorio de Investigación Balística del Ejército de USA.
    - 27 toneladas de peso
    - superficie de 167 m^2
    - 17.500 válvulas de vacío, 7.200 diodos d
    - 1.500 relés, 70.000 resistencias,
    - 10.000 condensadores



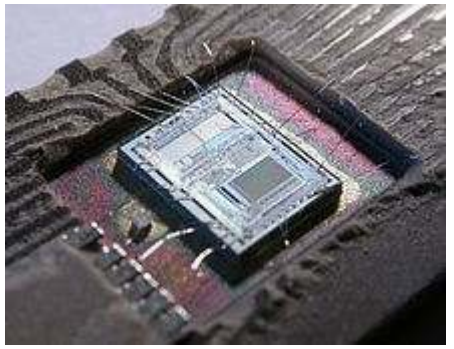
Micro-controladores

- TMS 1000 (1971)
- 4-bits
- Inventado en Texas Instruments
- ROM / RAM integrada



Microcontroladores

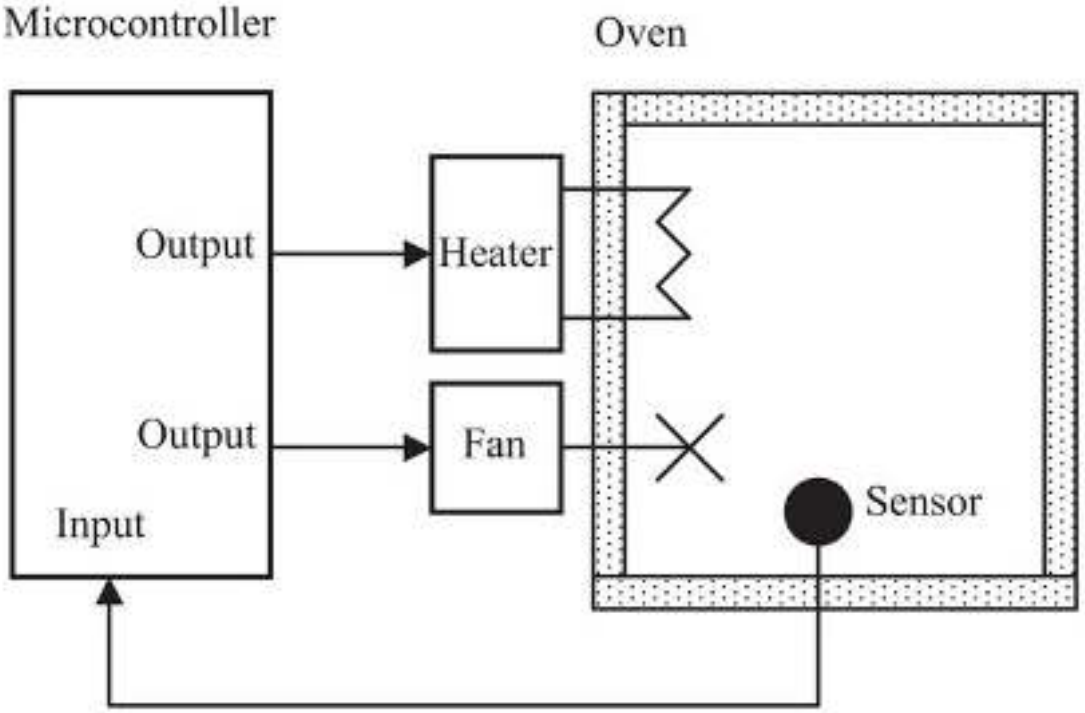
- Definición clásica:
  - Un **microcontrolador** (µC, UC o MCU) es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria.
  - Está compuesto de varios bloques funcionales que cumplen una **tarea específica**
  - Un microcontrolador incluye en su interior las **tres principales unidades funcionales** de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.

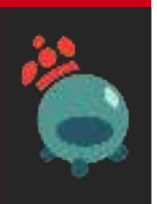


Microcontroladores

- Aparecieron a principios de los años 70 y supusieron un gran avance para la electrónica de aquel entonces.

- Realizan tareas específicas
- Disminuyen el coste económico de los sistemas
- Reducen el consumo de energía considerablemente
- Arquitectura Harvard y RISC

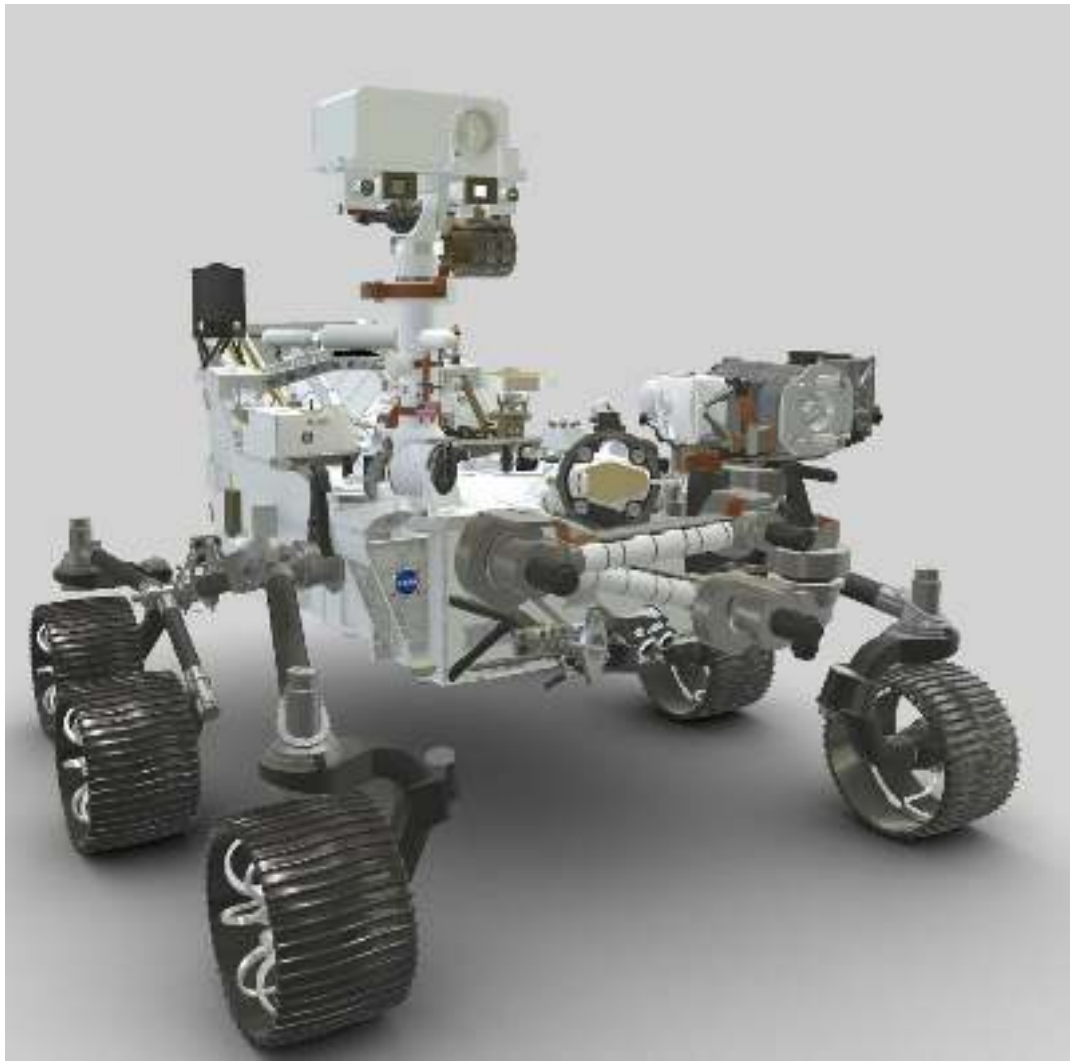




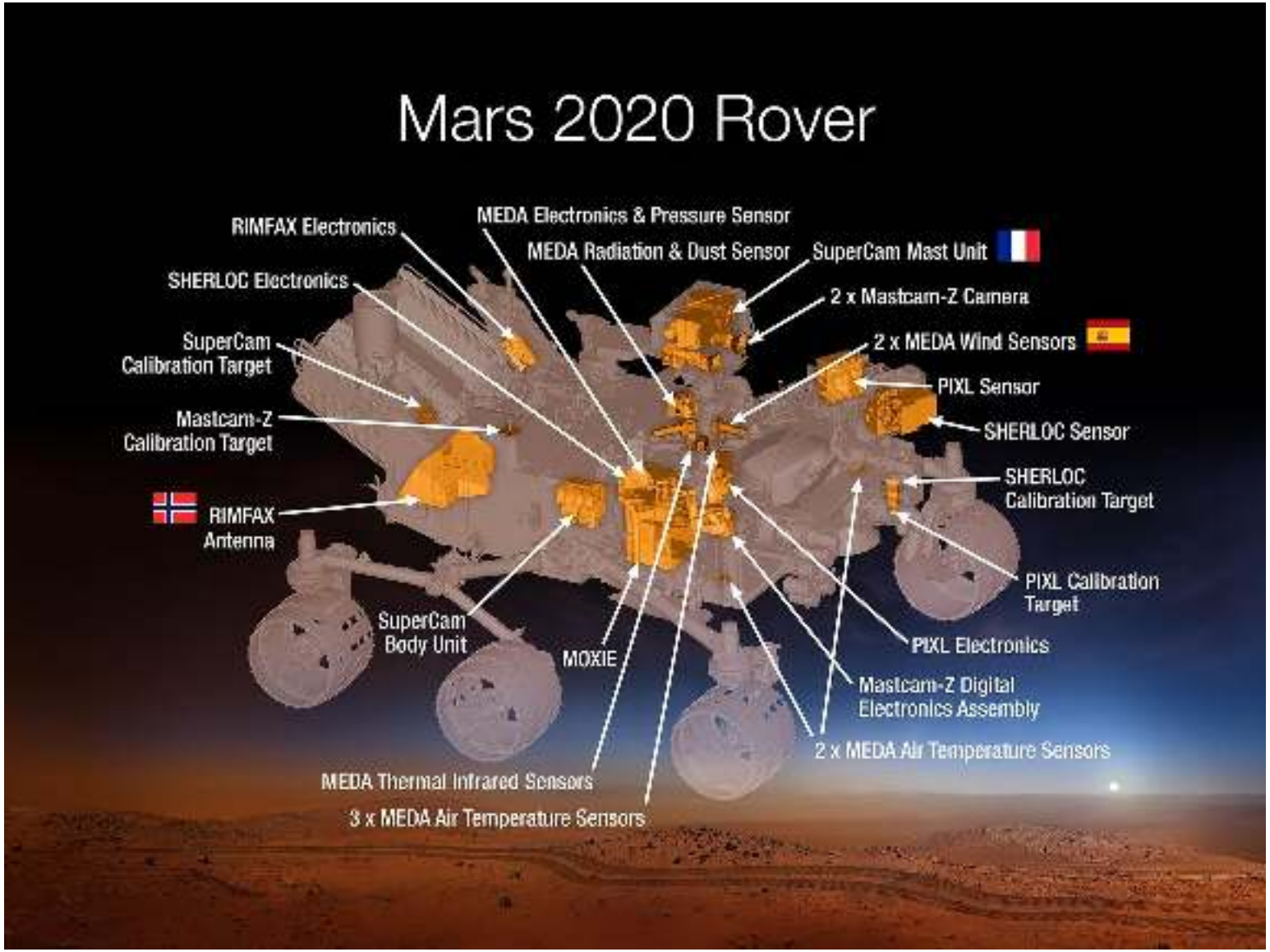
Microcontroladores

Mars 2020 – Rover Perseverance

- Su objetivo es encontrar vida microbiótica
- Es parte de una misión mucho más compleja
- Despegó con éxito el 30 de julio de 2020
- Aterrizó con éxito en el cráter Jezero el 18 de febrero de 2021
- VIDEO



Mars Environmental Dynamics Analyzer – MEDA

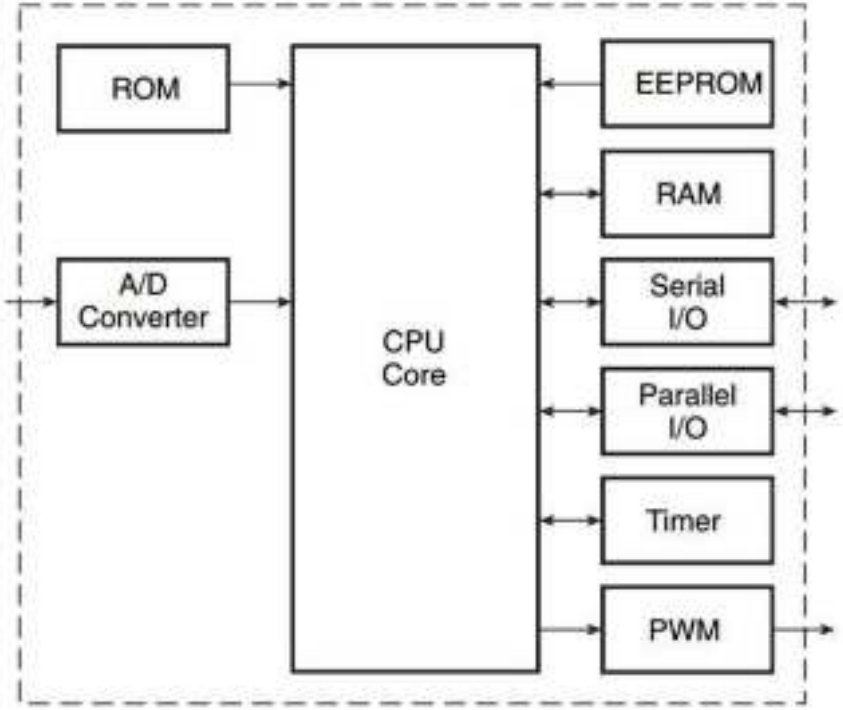


Microcontroladores

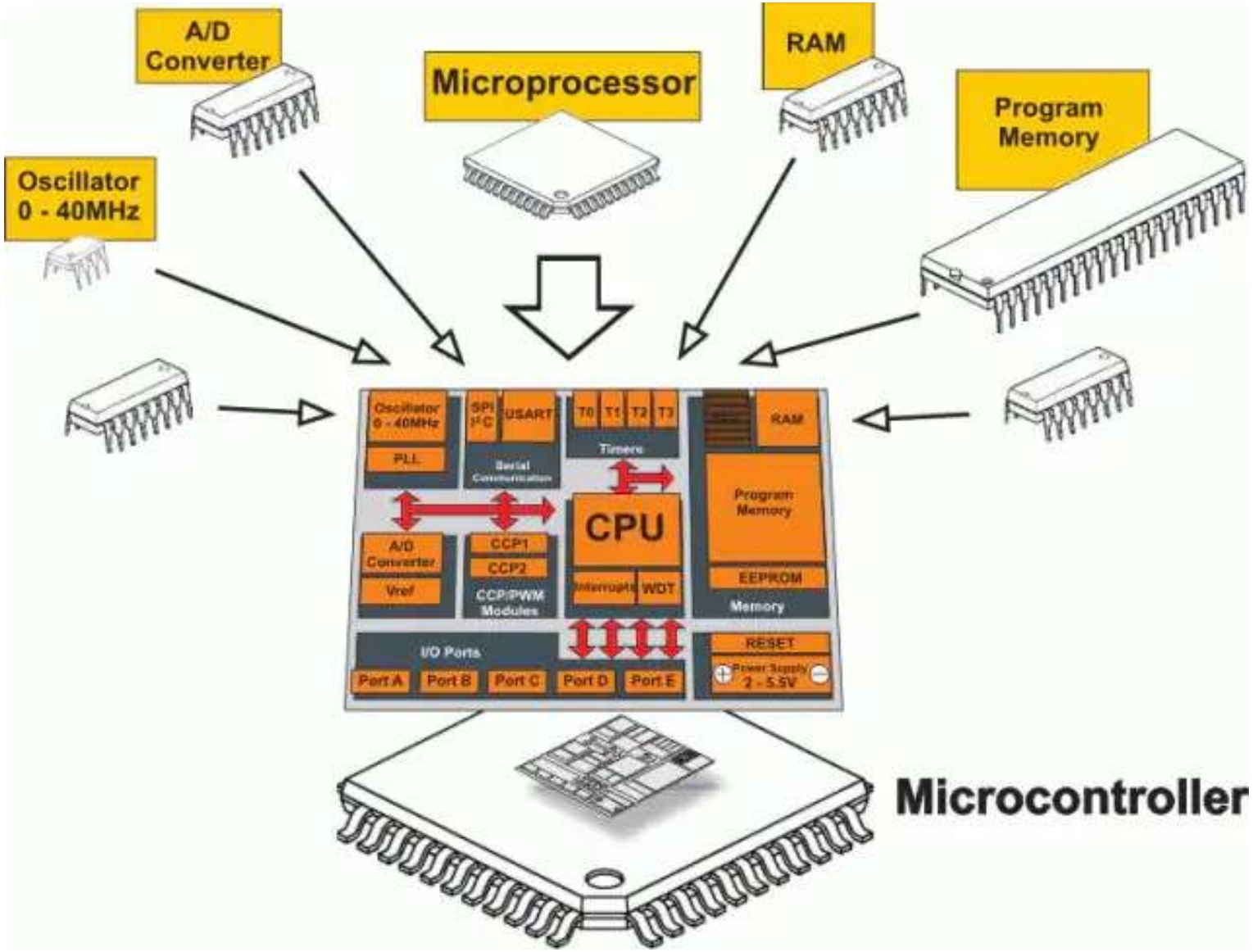
- Típicamente, encontraremos los siguientes componentes dentro de un microcontrolador
  - Unidad de proceso central (CPU)
  - ALU, Registros, Buses
  - Memoria de programa (no volátil)
  - Memoria datos (lectura/escritura)
  - Puertos de entrada/salida
  - Timers y contadores
  - Interrupciones
  - Convertidores analógico/digital
  - Interfaces de comunicación (serial, i2c)

Microcontroladores

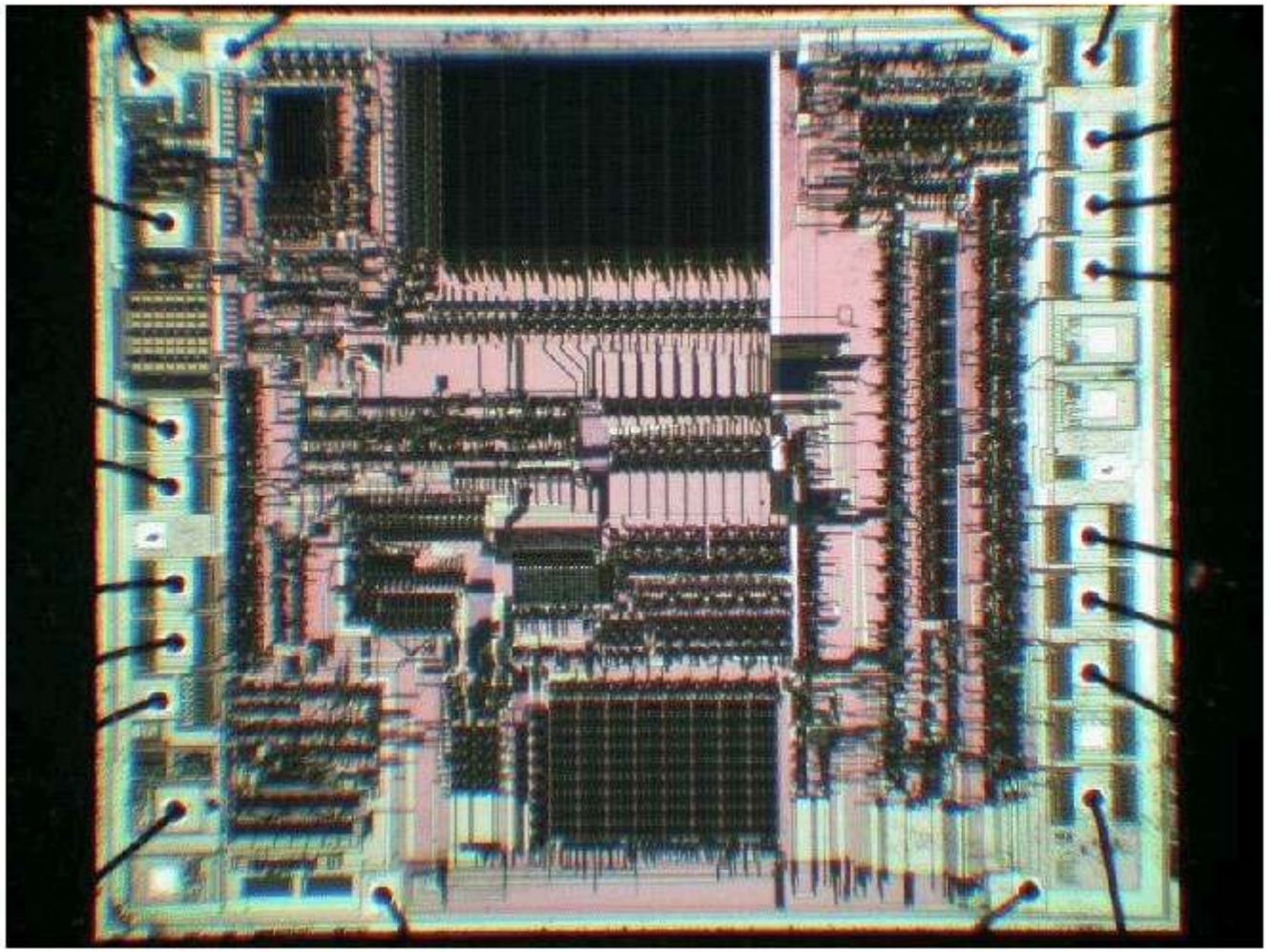
- Algunos recursos auxiliares
  - Circuito de reloj
  - Modulador de ancho de pulsos (PWM)
  - Puestos de comunicación (USB, CANBUS)
  - Sistema de protección
  - Estados de reposo
  - Watchdog



Microcontroladores

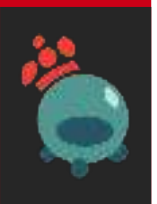


Microcontroladores



Microcontroladores

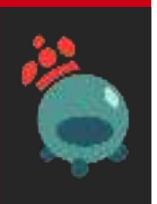
- Arquitectura **Von Neumann**
  - Utilizada en los ordenadores personales.
  - Entre otras cosas, se caracteriza por tener una sola memoria principal donde se almacenan datos e instrucciones.
  - Un único bus de dirección, datos y control.
- Arquitectura **Harvard**
  - Diferencia entre memoria de datos (SRAM) y de programas (ROM, EEPROM, flash)
  - Buses y memoria segregados
  - Conjunto reducido de instrucciones RISC
  - Utilizada en super computadores y microcontroladores.



Microcontroladores

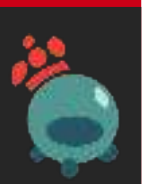
Microcontroladores

- Unidades de cómputo, memoria y procesamiento muy limitado
  - Por ejemplo: 32 KB de flash, y 4 KB memoria EEPROM.
- Se clasifican según el numero de **bits** de procesamiento:
  - 4/8 bits: Muy usados en el pasado … y ahora! (ATmega328P)
    - La tecnología Atmel AVR es una de las arquitecturas de 8 bits líderes en la industria.
  - 16/32 bits: Cuando los diseños exigen mayor potencia de procesamiento, estos chips ofrecen un rendimiento más de diez veces superior del MCU de 8 bits (MIPS32, ARM Cortex,..)
    - Smartphones, etc.



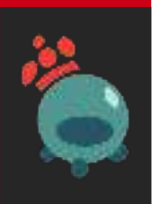
Microcontroladores

- Memoria:
  - Memoria ROM: El microcontrolador se fábrica con el programa ya grabado en la memoria.
  - Memoria PROM: (Programmable Read-Only Memory). Se pueden programar una sola vez.
  - Memoria EPROM: (Erase Programmable Read-Only Memory). Memoria reprogramable, pero hay que exponerla a una fuente de luz ultravioleta.
  - Memoria EEPROM: (Electrical Erase Programmable Read-Only Memory). Sustituto natural de las EPROM, se pueden borrar eléctricamente.
  - Memoria Flash: Son el último avance tecnológico, basada en impulso eléctricos, y ofrecen mayor velocidad.



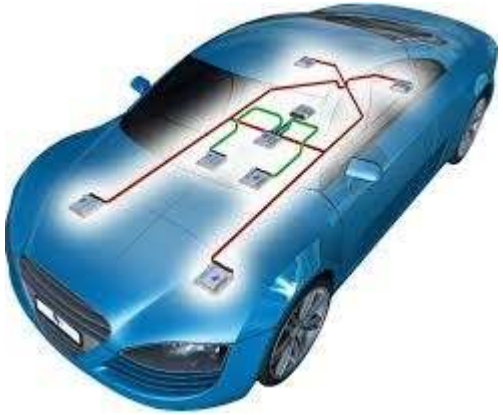
Microcontroladores

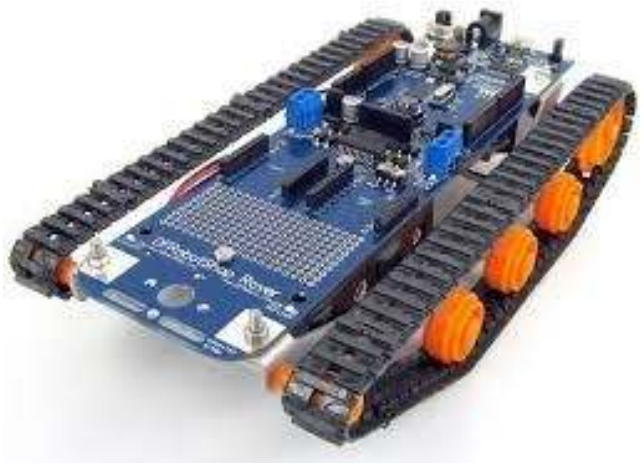
- El alto rendimiento de los microcontroladores se debe a tres técnicas:
  - Arquitectura Harvard
    - Memorias separadas y buses distintos para acceso.
    - Propicia el paralelismo.
  - Arquitectura RISC
    - Instrucciones reducidas y de tamaño fijo
  - Segmentación del procesador (pipelining)
    - Descomposición de la instrucción en varias etapas
    - Ejecutar etapas de varias instrucciones a la vez.



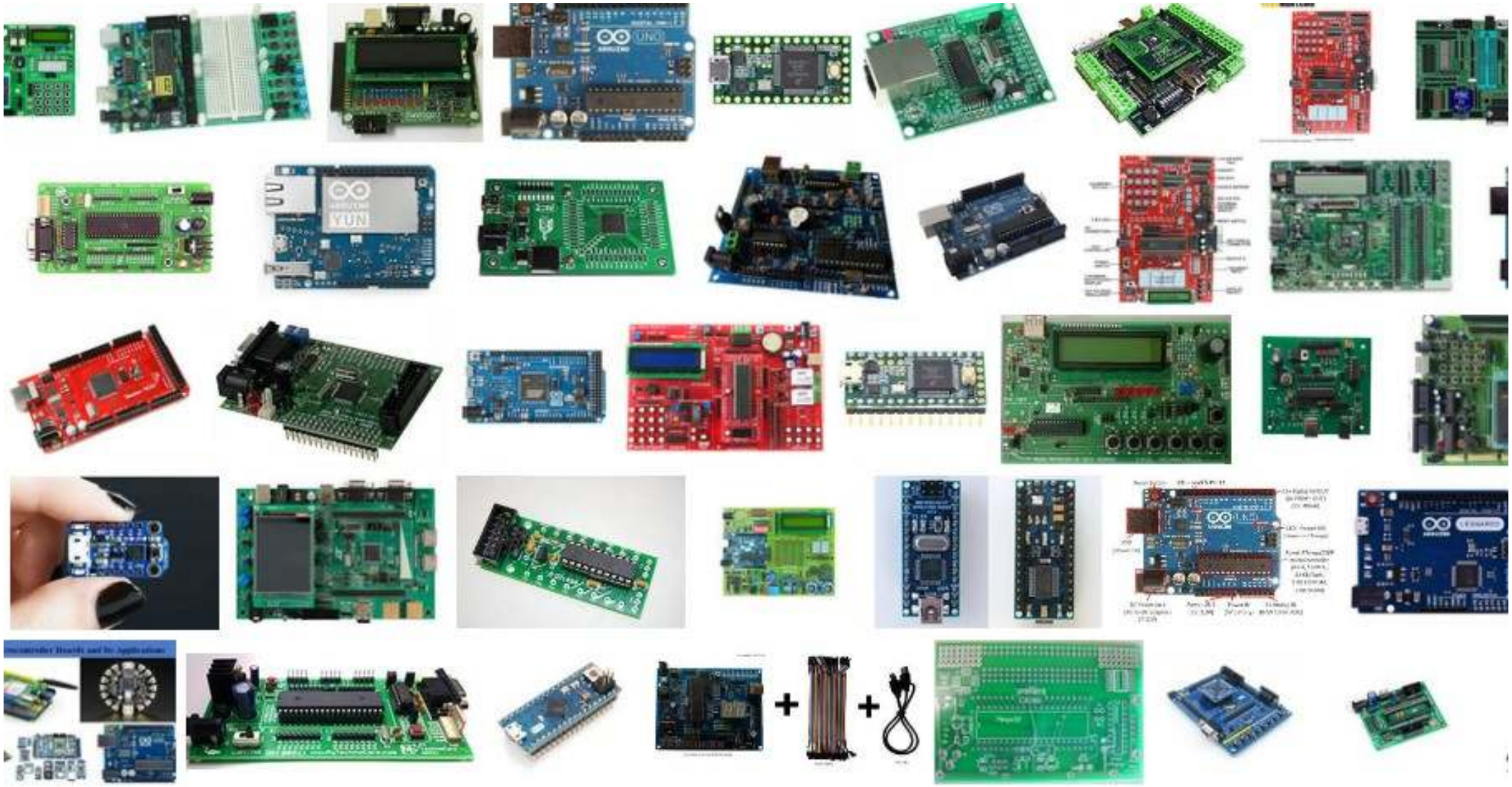
Áreas de uso

- TV, radio, electrodomésticos, equipos de música
- Sistemas domóticos
- Robots
- Industria del automóvil
- Aviones
- Industria aeroespacial (satélites, rovers, etc)
- Presente en todo el área de IoT





... Microcontroladores ...

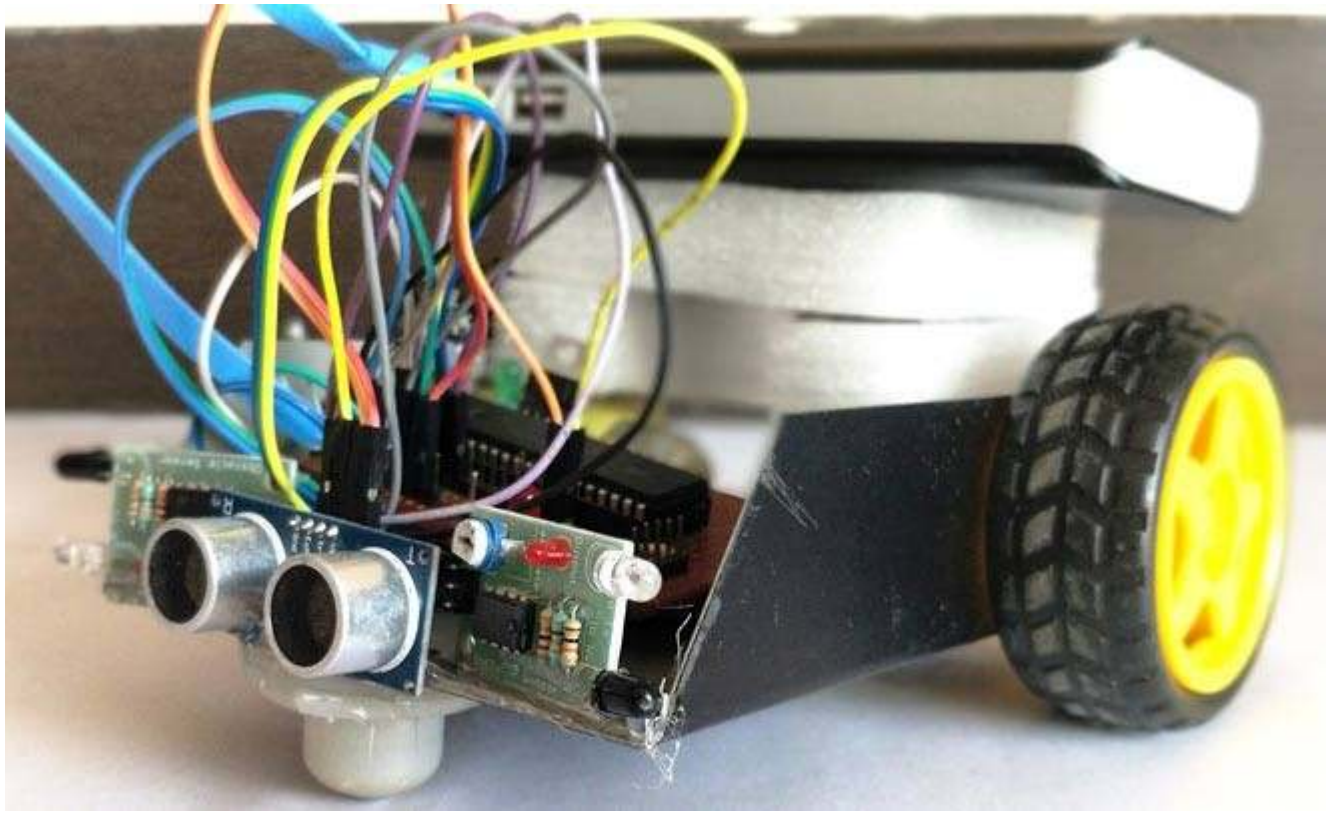


PIC 16F877A

- PIC 16F877A uno de los más populares de la familia PIC. Son fabricados Microchip Technology.
- 8-bit, RISC
- RAM: 368 bytes
- EEPROM: 256 KB
- Velocidad CPU (MIPS): 5
- 33 GPIO
- Coste: 4-5 €

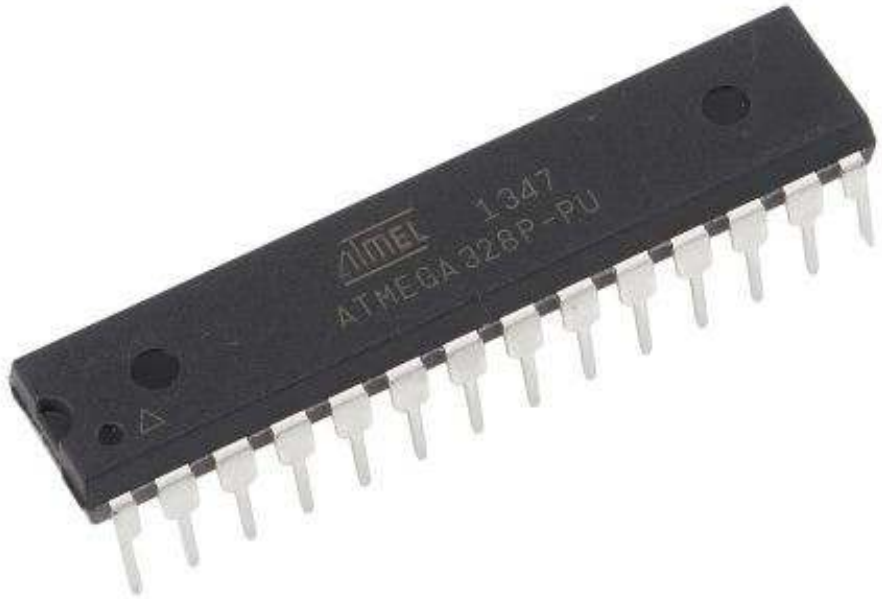
PIC 16F877A

- Detectar y evitar obstáculos.
- https://www.youtube.com/watch?v=TlzyRJipfAk



ATmega328

- Utilizado en las placas arduino
- 8-bit, RISC
- EEPROM: 32 KB
- SRAM: 2 KB
- Velocidad CPU (MIPS): 20 @ 20 MHz.
- 23 GPIO
- Arduino Nano, Uno, pro-mini.
- Coste: 2-3 €



ATmega328

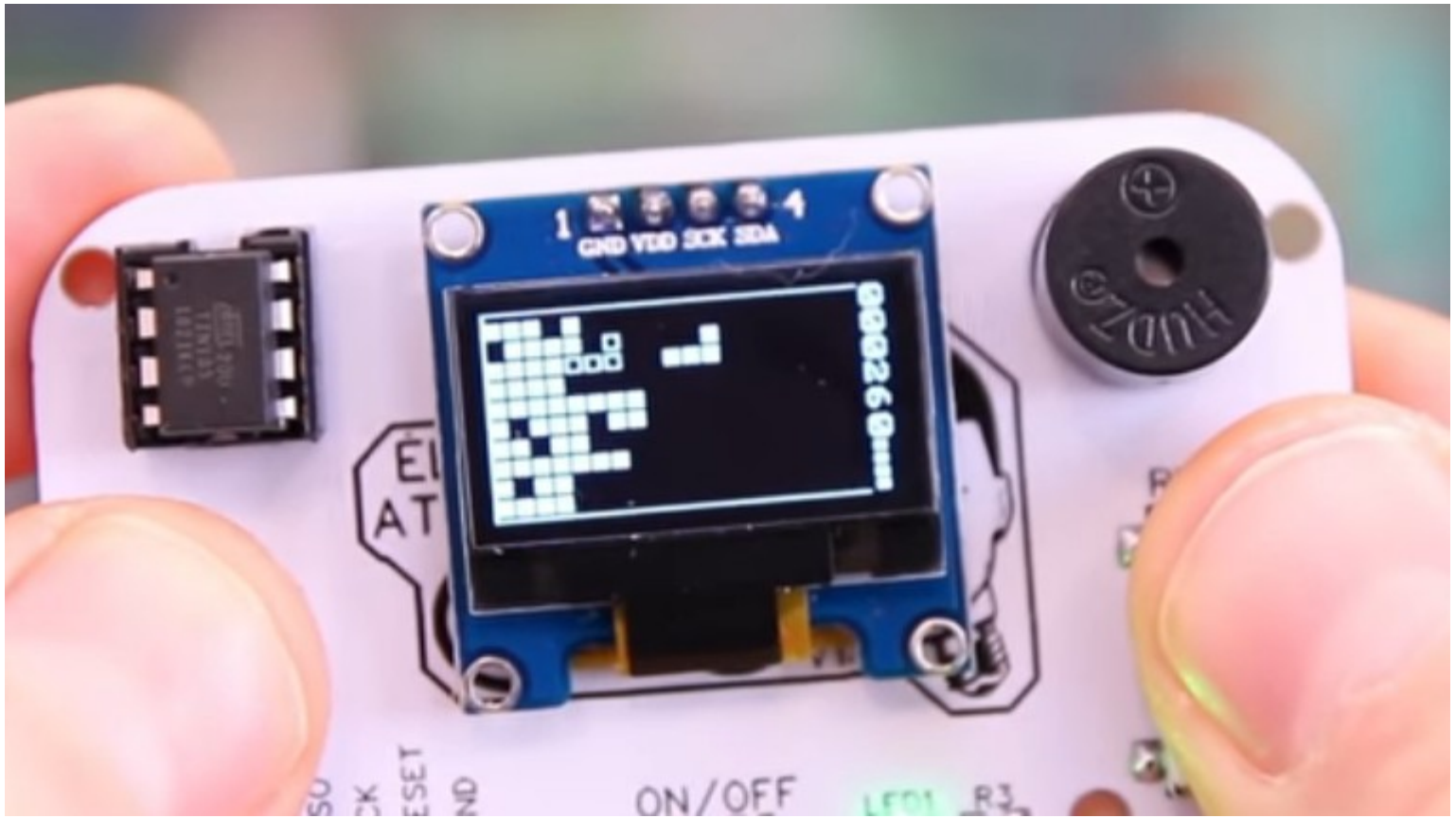
- Robot self-balancing
- https://youtu.be/I6z26LVu5y0?t=489

Attiny85

- Micro de muy pequeño formato.
- 8-bit, RISC
- EEPROM: 512 Bytes
- RAM: 512 Bytes
- Velocidad CPU (MIPS): 20 @ 20 MHz
- 6 GPIO
- Mini ATtiny85 USB, Digispark ATtiny85
- Coste: 2-3 €

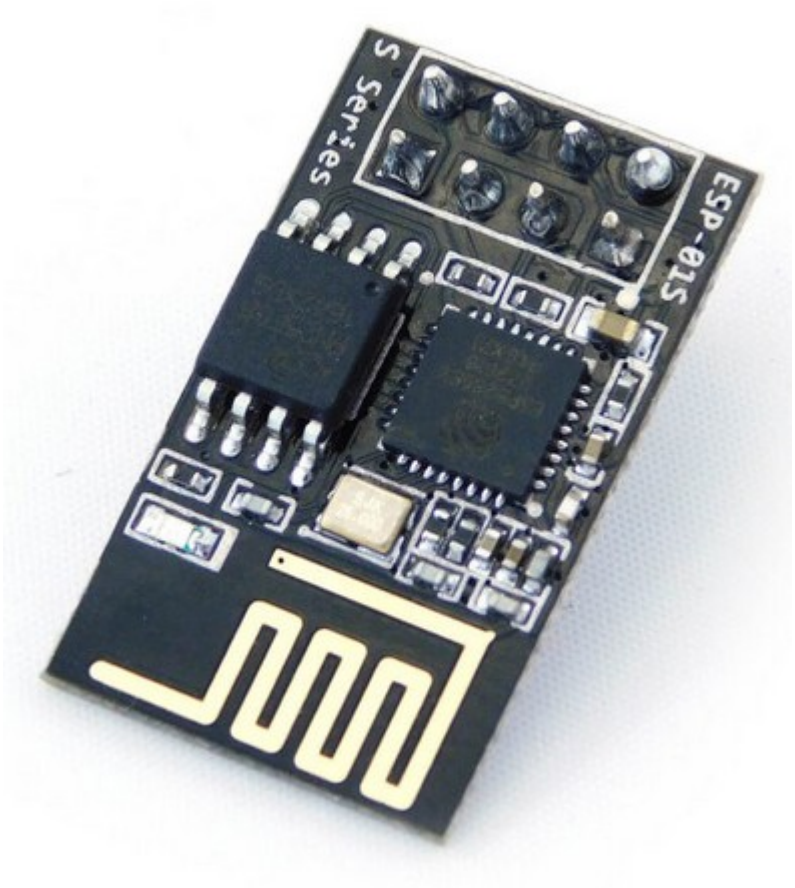
Attiny85

- Consola de Juegos.
- https://youtu.be/ddHbItTluKU?t=711



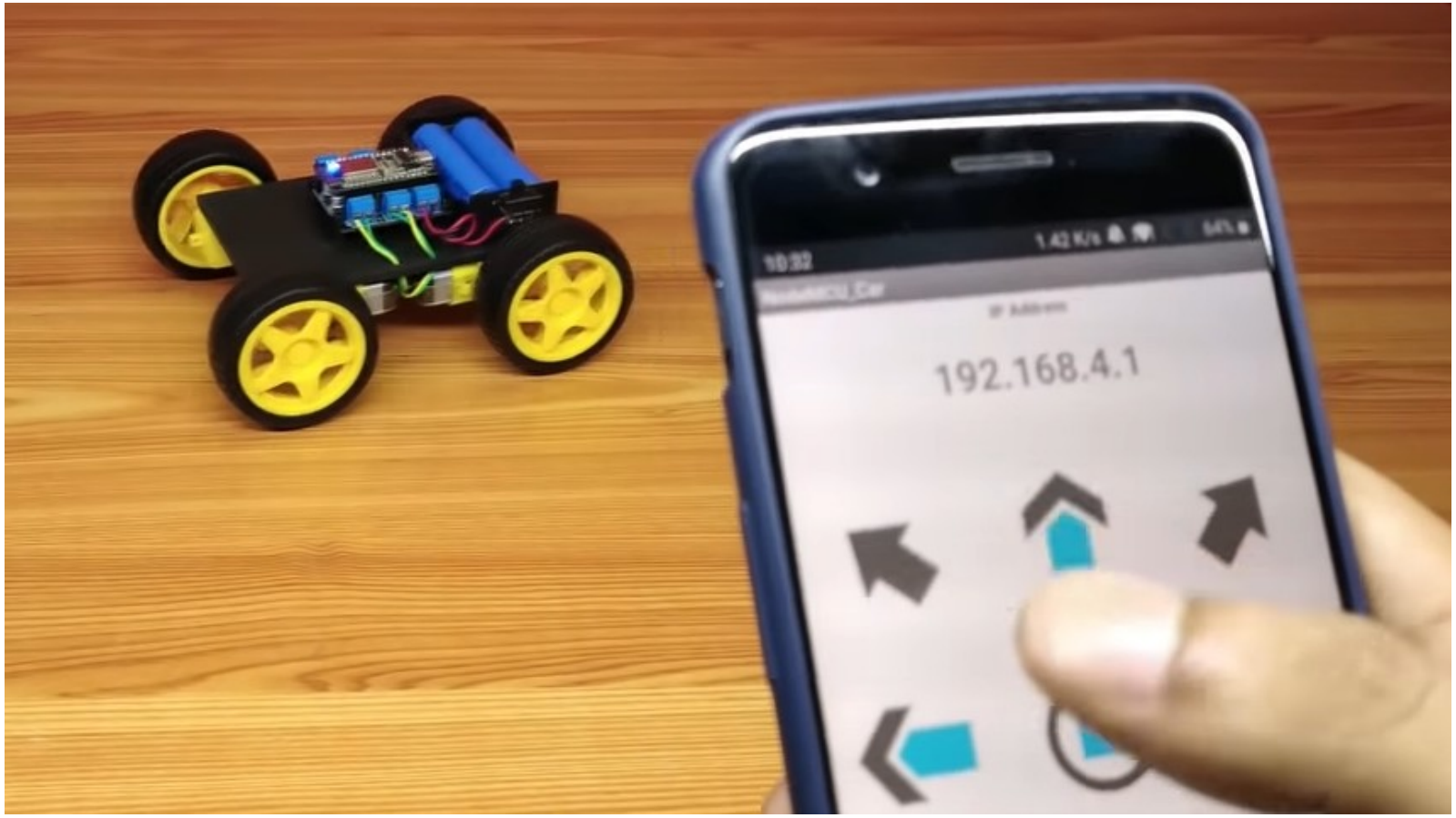
ESP8266

- Microchip potente y versátil de la última decada
- Microcontrolador Tensilica Xtensa LX106
- 32-bit, RISC
- EEPROM: 512 KB / 4BM
- RAM: 96 KBytes
- Velocidad CPU (MIPS): 80 MHz
- 16 GPIO
- IEEE 802.11 b/g/n Wi-Fi
- Modo de “deep sleep”
- Coste: 2-5 €



ESP8266

- Coche teledirigido
- https://youtu.be/zJnDbdefeCA?t=371



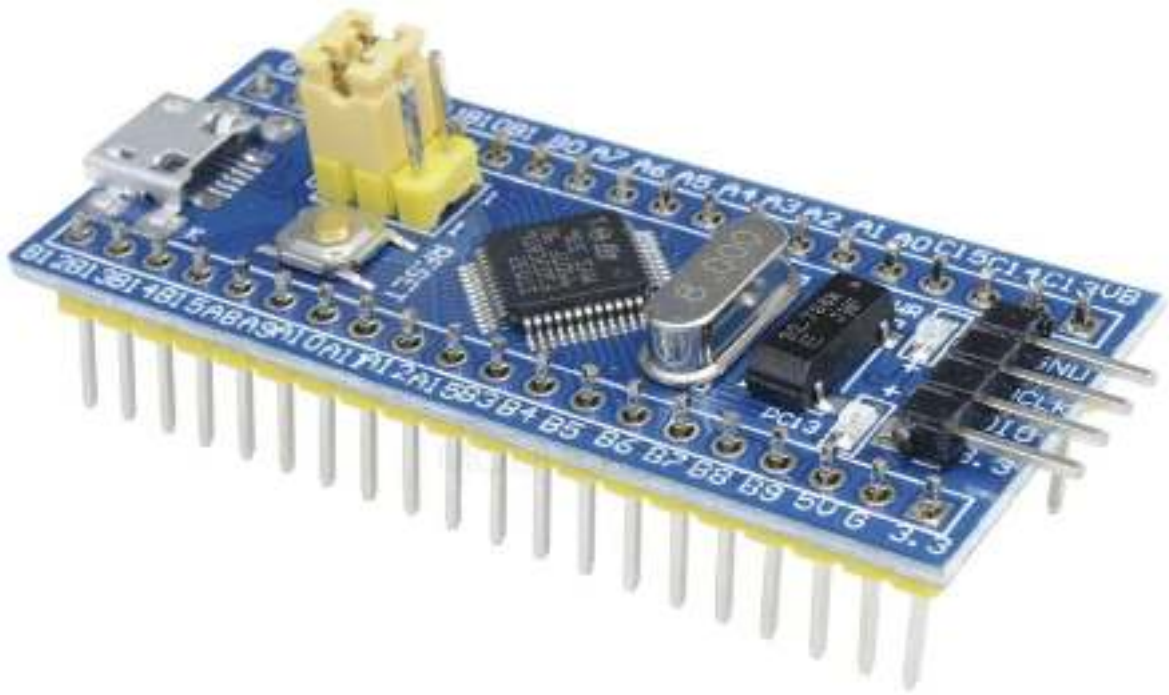
ESP8266

ESP8266 (TASMOTA)

- Open source firmware para dispositivos ESP
- https://tasmota.github.io/docs/
- Comunicación basada en MQTT

STM32F103C8T6

- ARM® Cortex® -M3
- 32-bit RISC
- Flash: 64/128 Kbytes
- SRAM: 20 Kbytes
- Reloj a 72 MHz, MIPS=90
- Oscilador 32 kHz (RTC)
- Timers, DMA, i2C
- Coste: 4-8 €

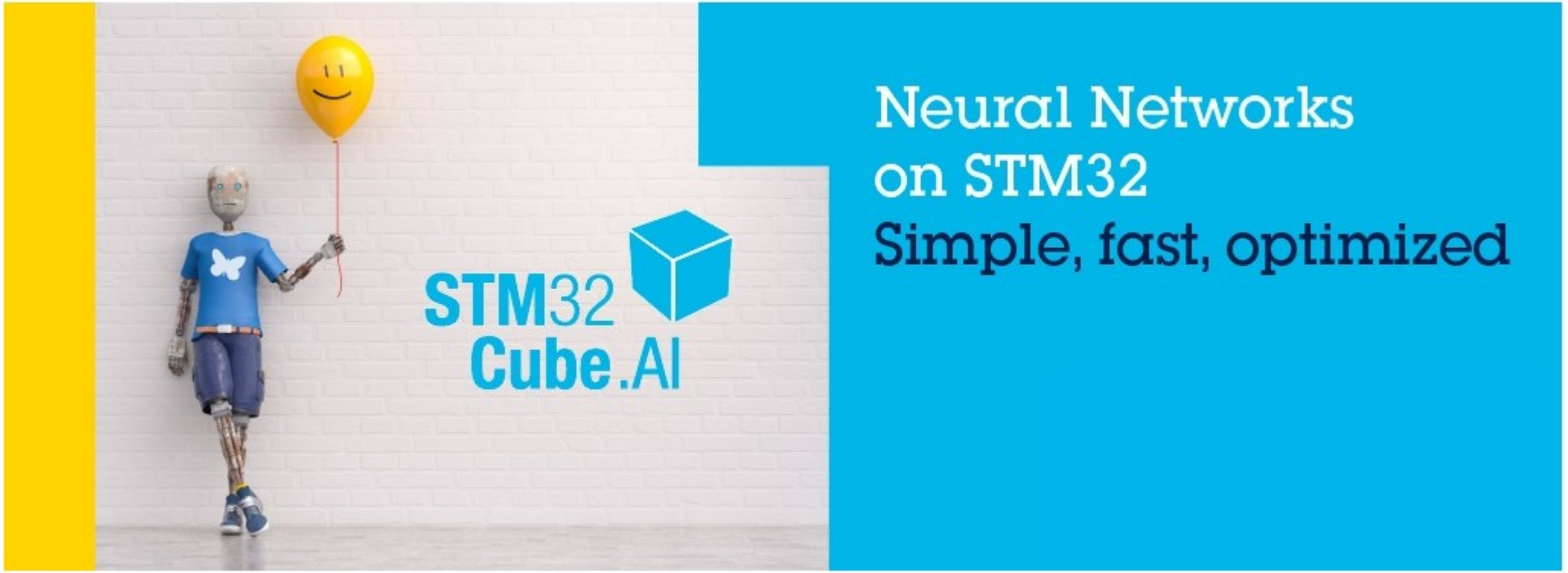


STM32F103C8T6

- STM32 quadcopter
- https://www.youtube.com/watch?v=dWnF3sAvONM

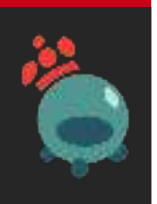
STM32

- Posibilidad de ejecutar redes pre-entrenadas en el STM32
- Video



Consumo

	PIC 16F877A	ATmega328	Attiny85	ESP8266	STM32F103C8T6
Voltaje Operación (V)	2 - 5.5	1.8 - 5.5	1.8 - 5.5	2.5 – 3.3	2.0 - 3.6
Consumo (mA)	0.3-0.6	3-6	5	35	1.19



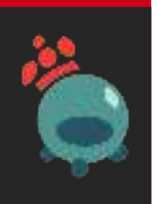
Consumo

	PIC 16F877A	ATmega328	Attiny85	ESP8266	STM32F103C8T6
Voltaje Operación (V)	2 - 5.5	1.8 - 5.5	1.8 - 5.5	2.5 – 3.3	2.0 - 3.6
Consumo (mA)	0.3-0.6	3-6	5	35	1.19

¡¡¡¡¡ 5V – 3A !!!!

¡Vamos a construir un Drone!

- Características del sistema
  -
  -
  -
  -
- RT o no RT
- Microcontrolador/(es) / Microprocesador



URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC Programming Robot Intelligence

Sistemas Empotrados y de Tiempo Real

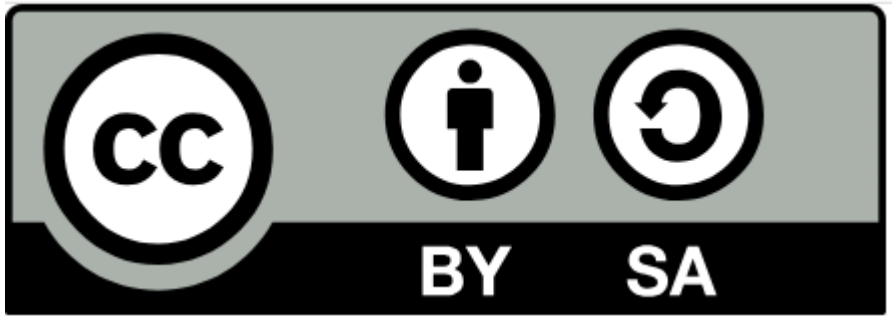
Desarrollo con Arduino

Entradas/Salidas Analógicas Digitales

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

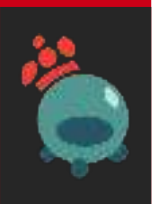
Roberto Calvo Palomino
roberto.calvo@urjc.es



Esqueleto del sketch

- Todo sketch tendrá 2 funciones obligatorias:
  - ***setup()***: Función que se utiliza para inicializar datos o puertos, y que se ejecuta 1 única vez al inicio del programa.
  - ***loop()***: El contenido de esta función ejecuta repetidamente mientras la placa arduino siga encendida.


```
// the setup routine runs once when you press reset:
void setup() {

}

// the loop routine runs over and over again forever:
void loop() {

}
```

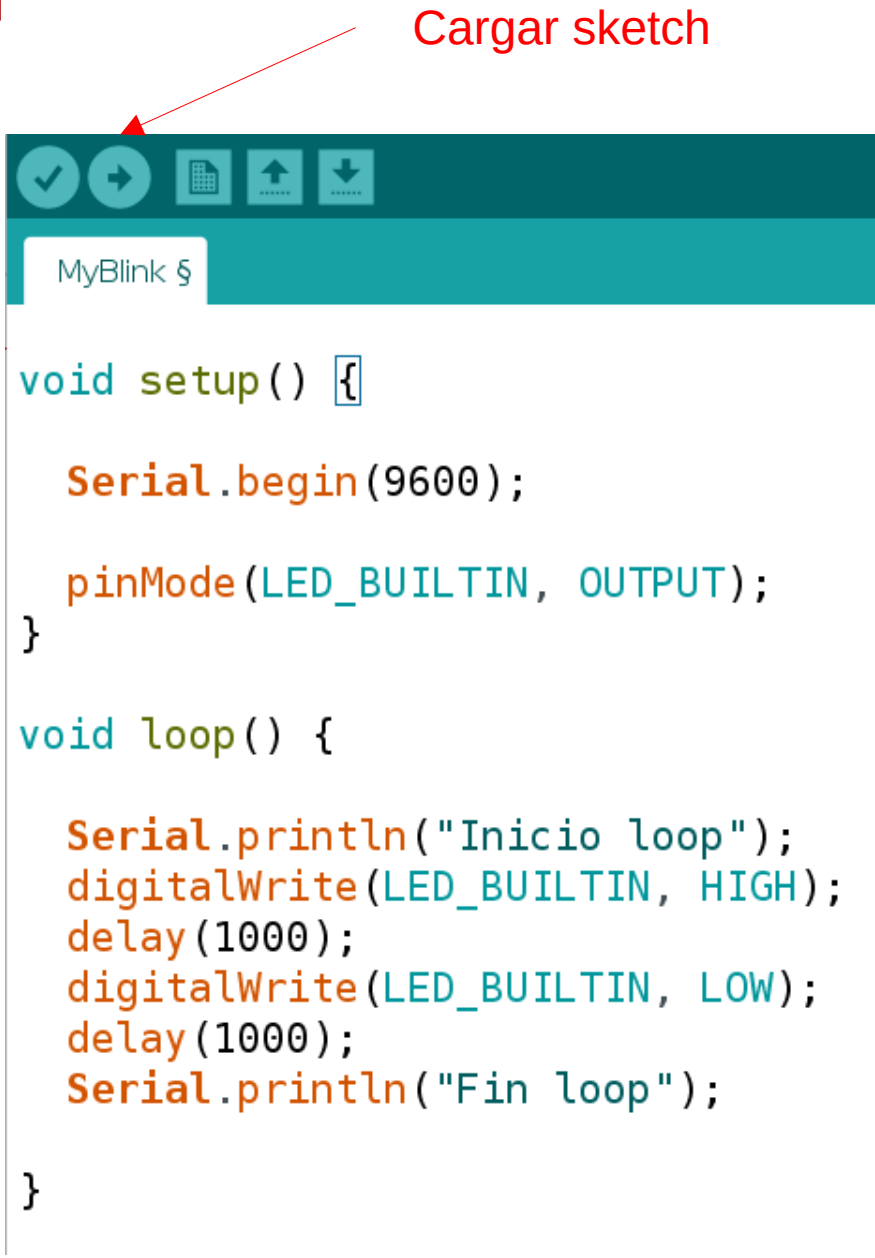


Depuración

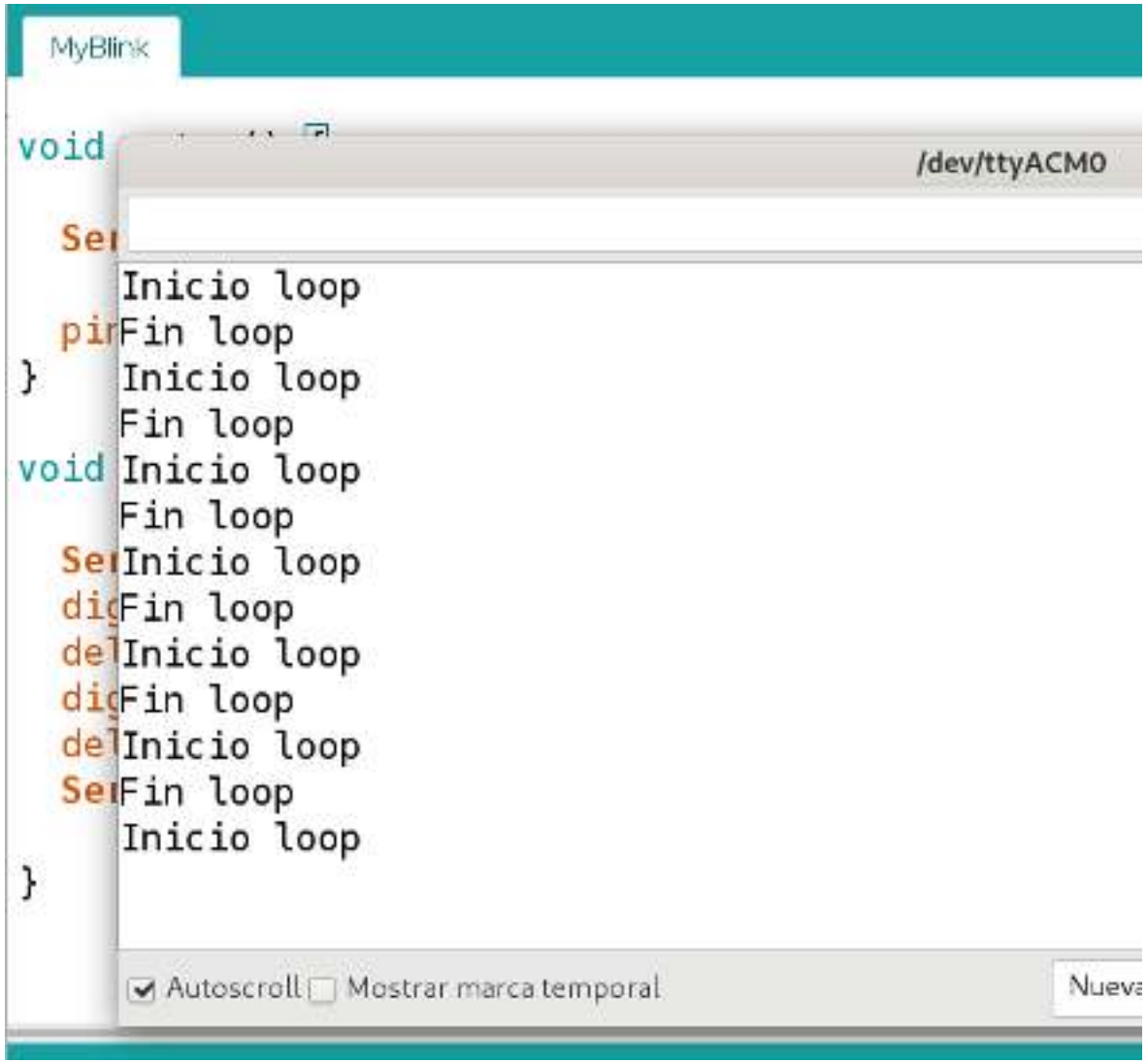
- A través del puerto serie (USB)
- Configurar la velocidad del puerto serie en la función *setup()*

  `Serial.begin(SPEED)`
- Imprimir mensajes con

  `Serial.println(MESSAGE)`
- Todos los mensajes son mostrados por la interfaz serial de Arduino.
- Normalmente conectada a la interfaz USB.



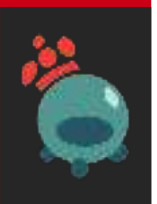
Depuración


MyBlink
void
Ser
pir
}
void
Ser
dig
del
dig
del
Ser
}
/dev/ttyACM0
Inicio loop
Fin loop
Inicio loop
Fin loop
Inicio loop
Fin loop
Inicio loop
Fin loop
Inicio loop
Fin loop
Inicio loop
Fin loop
Inicio loop
Fin loop
Inicio loop
Autoscroll
Mostrar marca temporal
Nueva


Enter

Funciones Comunes

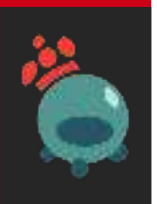
- **`Serial.println(value)`**
  - Imprime el valor por el puerto serie.
- **`pinMode(pin, mode)`**
  - Configura un pin digital para leer (entrada), o escribir (salida)
- **`digitalRead(pin)`**
  - Lee un valor digital (HIGH o LOW) en un pin establecido como entrada.
- **`digitalWrite(pin,value)`**
  - Escribe un valor digital (HIGH o LOW) en un pin establecido como salida.



Tipos de Datos

Tipos Numéricos	Bytes(*)	Rango	Uso
`int`	2	-32768 to 32767	Enteros negativos y positivos
`unsigned int`	2	0 to 65535	Solo enteros positivos
`long`	4	-2147483648 to 2147483647	Rango largo de enteros positivos/negativos
`unsigned long`	4	4294967295	Solo largo rango de enteros positivos
`float`	4	3.4028235E+38 to -3.4028235E+38	Representa numero coma flotante
`double`	4	Igual que float	En Arduino, igual que float.
`bool`	1	false (0) o true (1)	Valores condicionales
`char`	1	-128 to 127	Representa un carácter ASCII
`byte`	1	0 to 255	Enteros en un rango corto

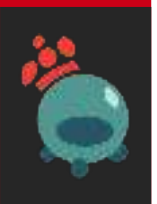
(*) Tipos de datos para placas arduino basadas en microcontroladores de 8-bit.



Grupo de Valores

- Genera arrays de pines y asigna su modo fácilmente.

```
int ledPins[] = {10, 11, 12, 13};

void setup()
{
  for (int index = 0; index < 4; index++)
    pinMode(ledPins[index], OUTPUT);
}
```



String

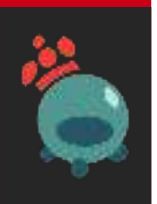
- La clase String nos ayudará a trabajar con cadenas de caracteres de una manera muy fácil

```
char oldString[] = "this is a character array";
```

```
String newString = "this is a string object";
```

- Al ser una clase nos provee numerosos métodos para trabajar con strings (conversiones, comparaciones, concatenaciones, busquedas, reemplazamientos, ...)

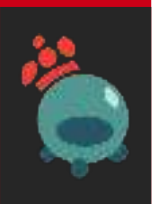
  - https://docs.arduino.cc/built-in-examples/strings/StringConstructors/



Muy similar a C

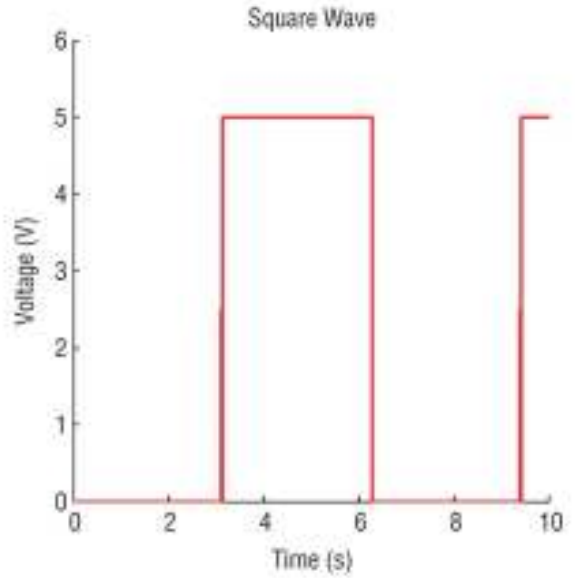
- Structs
- Sentencias condicionales o de control
- Sentencias repetitivas
- Comparadores numéricos y lógicos

Para mas detalle consultar la bibliografía de la asignatura



Señales Digitales

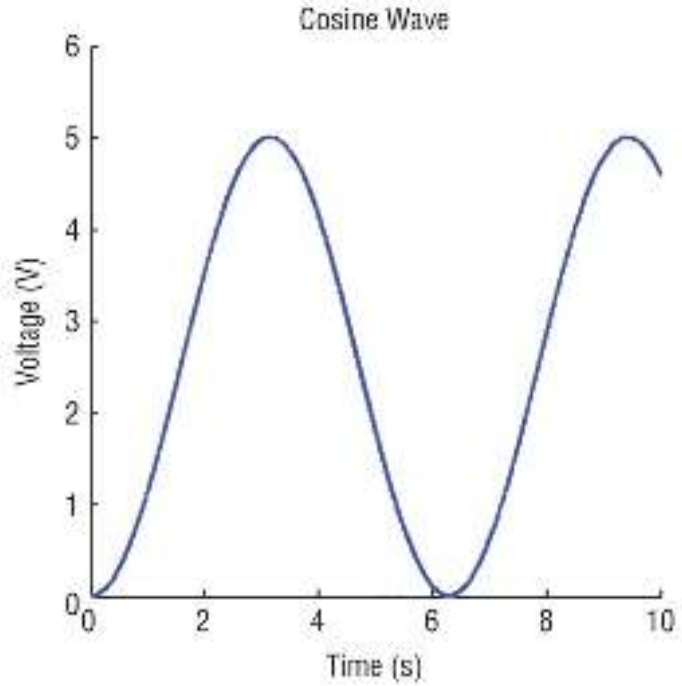
- Una **señal digital** es un tipo de señal (normalmente generada por un fenómeno electromagnético) en que cada símbolo que codifica se puede analizar a través de valores discretos.
- Los microcontroladores usan lógica de 2 estados representados por 2 niveles de tensión eléctrica (H=High y L=Low). A alto nivel representado como 0 y 1.
- Ejemplos:
  - LED
  - Switch / Interruptor



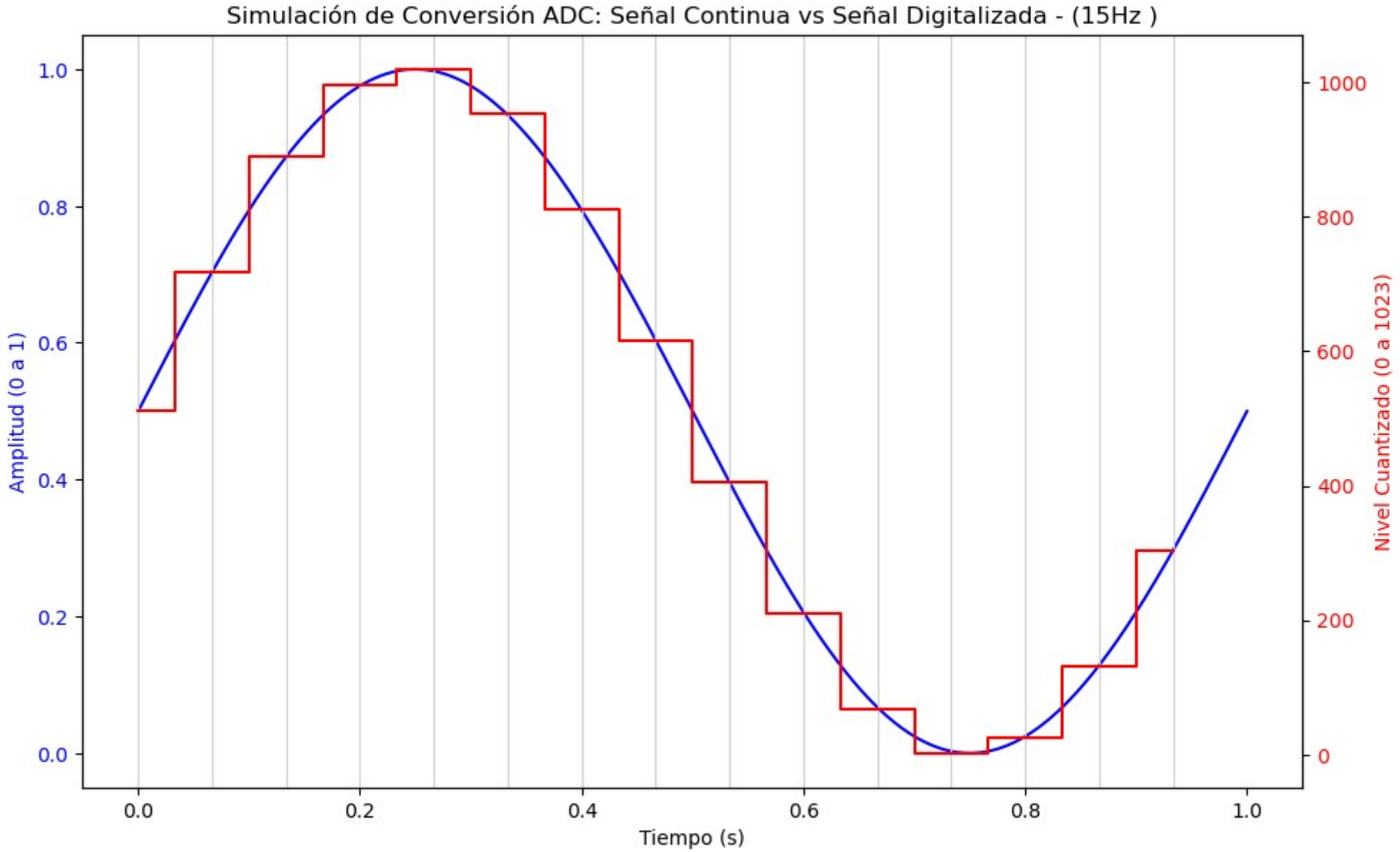
Señales Analógicas

- Una señal analógica es aquella que los valores de la tensión del voltaje varían constantemente durante el tiempo.
- Arduino UNO (atmega328p) contiene un conversor (ADC) con una resolución de 10 bits, devolviendo valores en el rango de [0-1023] (valor máximo 2^{10})
- Frecuencia reloj ADC: 125 kHz
- El proceso de conversión 10-Bit A/D: 13 ADC clocks
- ADC Sampling rate: 1/9.6 kHz ~ 100 microsec.

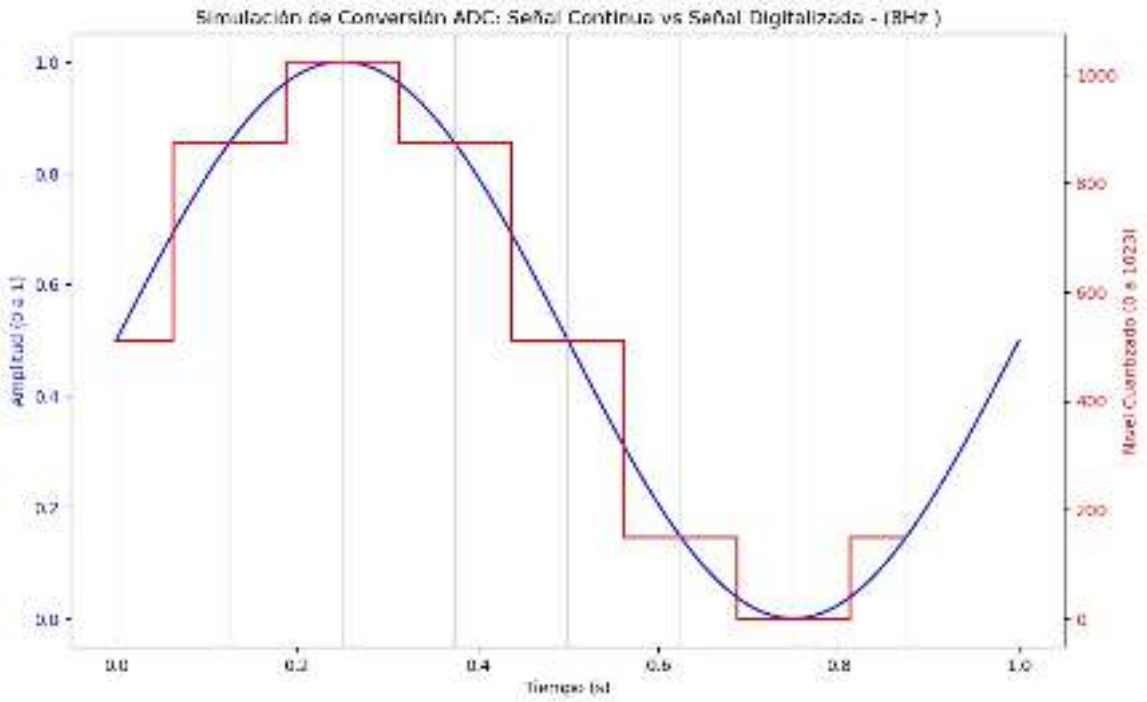
- Muchos sensores son analógicos:
  - Temperatura, PIR, ultrasonido, etc.

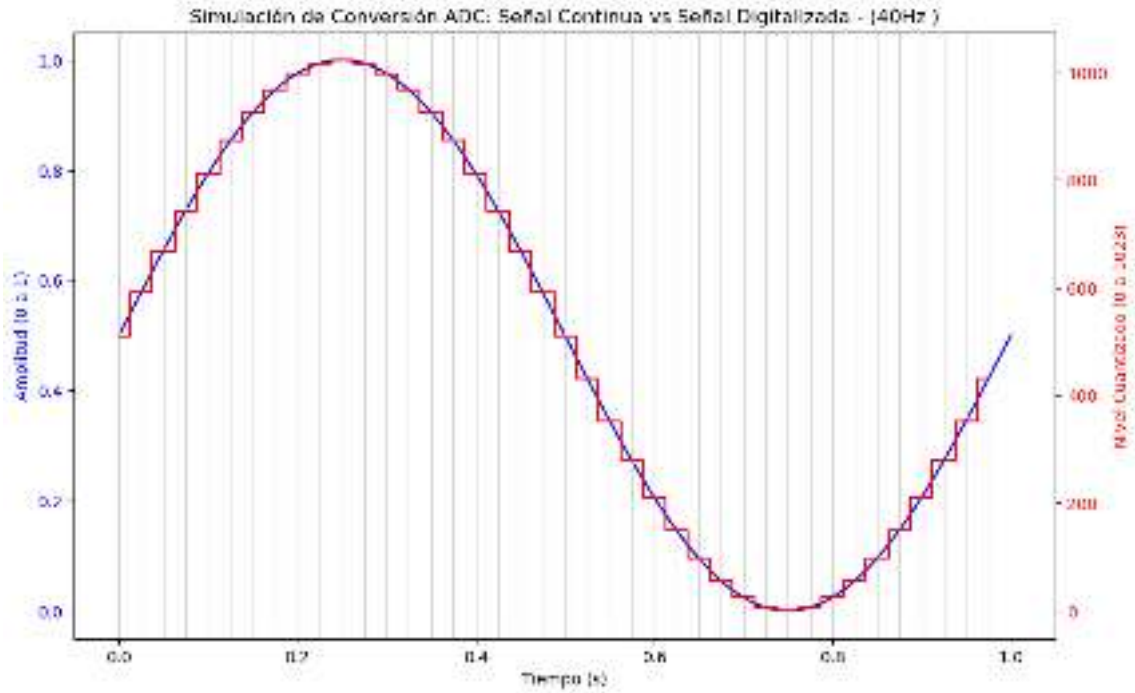


Señales Analógicas (ADC)



Señales Analógicas (ADC)



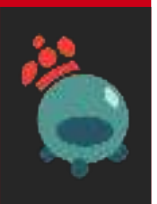


- Recuerda: Arduino UNO
  - ADC Sampling rate: 1/9.6 kHz ~ 100 microsec.

PWM (Pulse Width Modulation)

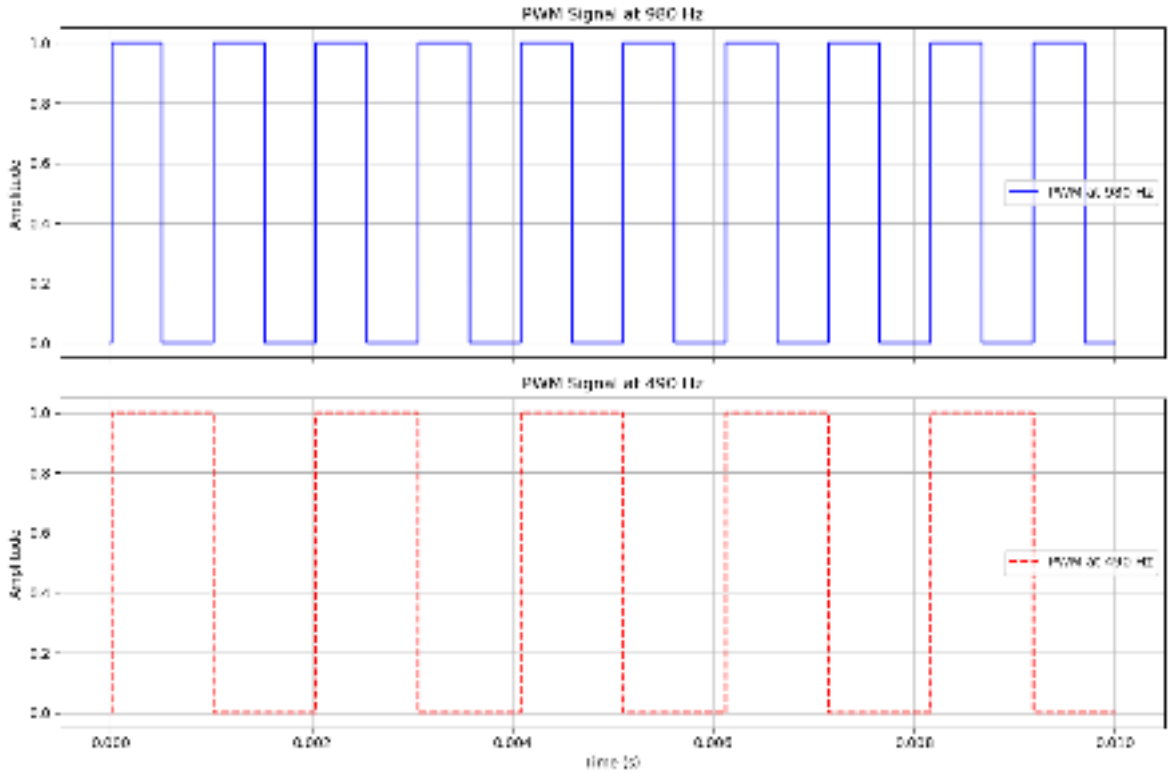
- PWM nos permite simular comportamientos analógicos a través de de los pines digitales de Arduino.
- En una placa Arduino Uno, los pines PWM son: 3, 5, 6, 9, 10, y 11.
- PWM hace uso de relojes/contadores y timers.
- Problema
  - Un led se enciende (HIGH) o se apaga (LOW)
  - Un motor está girando (HIGH) o está pagado (LOW)

¿Cómo podemos generar diferentes intensidades para el Led o diferentes velocidades para el motor?



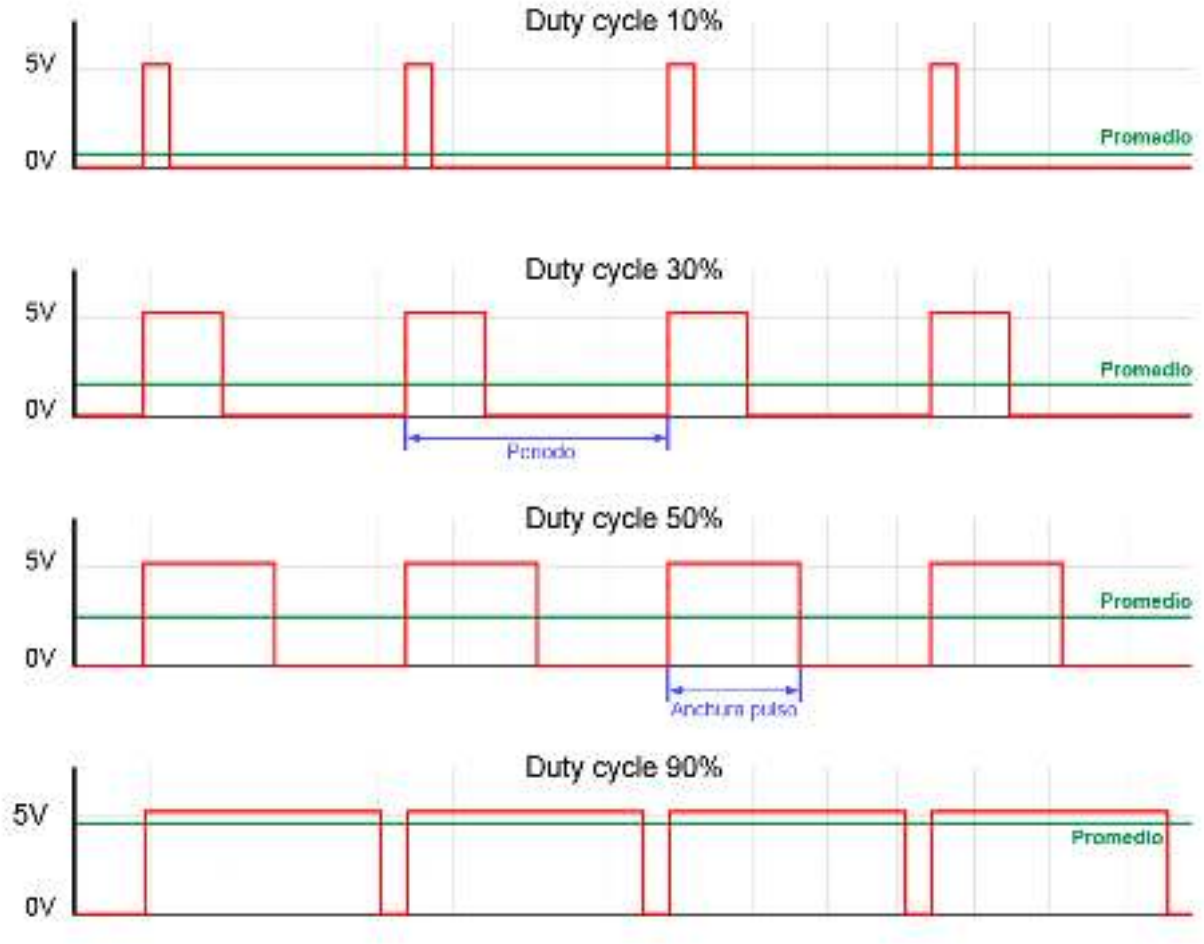
PWM (Pulse Width Modulation)

- PWM en arduino UNO usa 8 bits (0-255)
- Frecuencia 490 Hz (Timer1 y Timer2): pines 3,9,10,11
- Frecuencia 980 Hz (Timer 0): pines 5 y 6



PWM (Pulse Width Modulation)

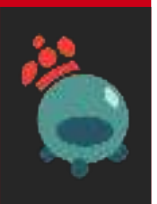
- La proporción de tiempo que está la señal HIGH respecto al total del ciclo, se denomina "duty cycle".



PWM (Pulse Width Modulation)

- Arduino incorpora la función analogWrite() que puede ser usada para generar una señal PWM en un pin digital.
  - analogWrite(PIN, 0): es una señal de ciclo de trabajo del 0%.
  - analogWrite(PIN, 127): es una señal de un ciclo de trabajo del 50%.
  - analogWrite(PIN, 255): es una señal de ciclo de trabajo del 100%.

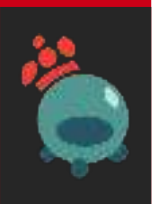
- IMPORTANTE: analogWrite genera señales digitales de 5V (incluso a un duty cycle muy bajo), si el sensor solo soporta 3V puede sufrir daños.



Trabajando con el tiempo

- Usa las siguientes funciones para obtener el contador
- millis(): Número de milisegundos desde que la placa empezó a ejecutar el sketch actual.
- micros(): Número de microsegundos desde que la placa empezó a ejecutar el sketch actual.
  - En placas de 16 MHz:
    - resolución de 4 microsegundos
  - En placas de 8 MHz:
    - resolución de 8 microsegundos

```
unsigned long time;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("Time: ");
  time = micros();

  Serial.println(time);
  delay(1000);
}
```

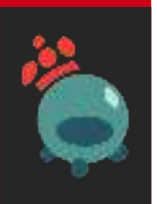


Esperas y pausar la ejecución

- *delay(value_ms)*
  - Pausa el sketch durante la cantidad de milisegundos especificados
- *delayMicroseconds(value_micros)*
  - delay bastante exacto para valores en el rango [3 - 16383]
  - Para valores mayores usar delay()

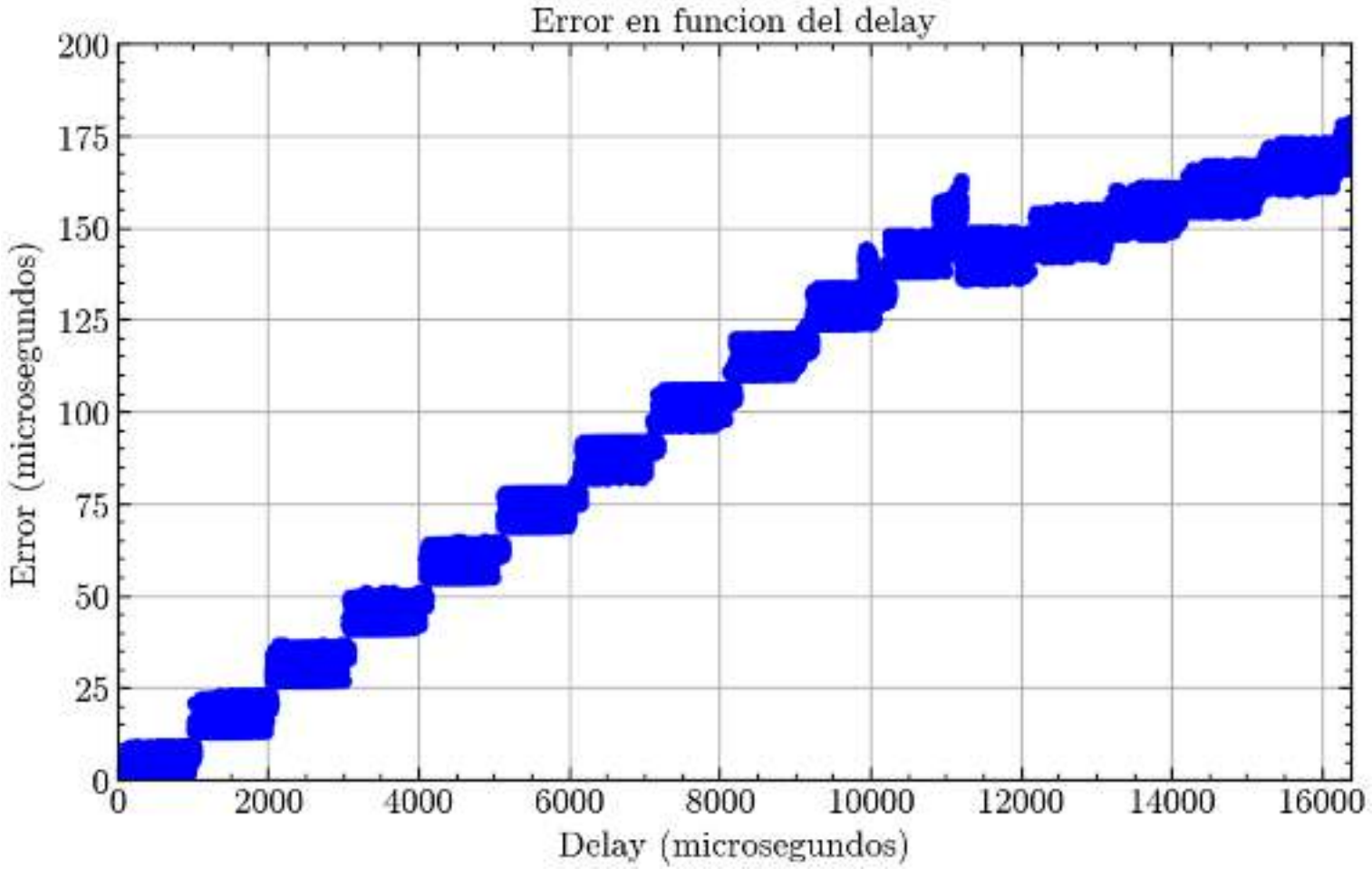
Estas funciones son bloqueantes, pero no se asemeja su funcionamiento a sleep en sistemas GNU/Linux

¿Como afecta la ejecución de delay() en el microcontrolador?



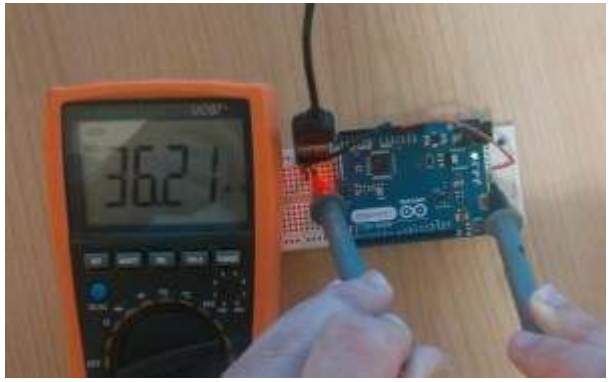
delayMicroseconds

- Delay bastante exacto para valores en el rango [3 – 16383]
- No utiliza Timer, utiliza ciclos de procesador.



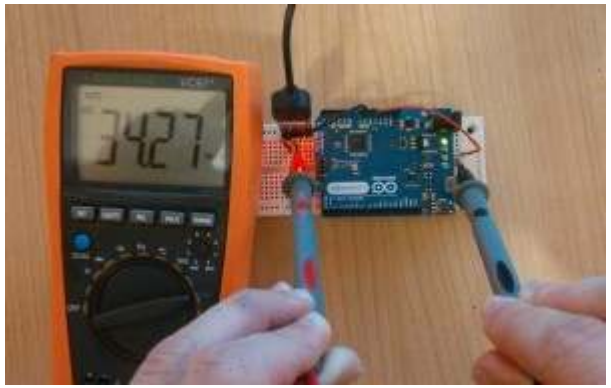
Uso de delay()

```
void setup() {}

void loop() {
  delay(99999);
}
```



(*)

```
void setup() {}

void loop() {
  while(1);
}
```



- delay() no libera la CPU del microcontrolador.
- Veremos mecanismos para mejorar este comportamiento.

(*) https://todohacker.com/tutoriales/arduino-usar-delays-o-evitarlos

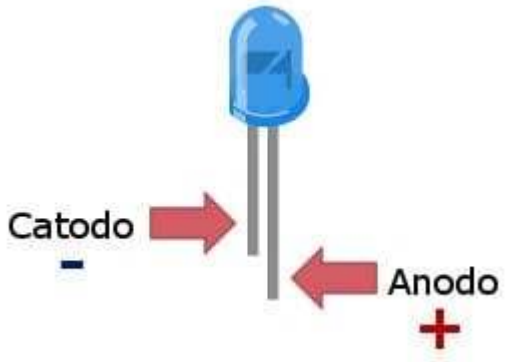
delay()

```
void delay(unsigned long ms)
{
        uint32_t start = micros();

        while (ms > 0) {
                yield();
                while ( ms > 0 && (micros() - start) >= 1000) {
                        ms--;
                        start += 1000;
                }
        }
}
```

https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/wiring.c#L106

Sensores

- LED
  - Sensor Digital
  - Diodo: cátodo, ano
  - 1.2 - 3.6 v.



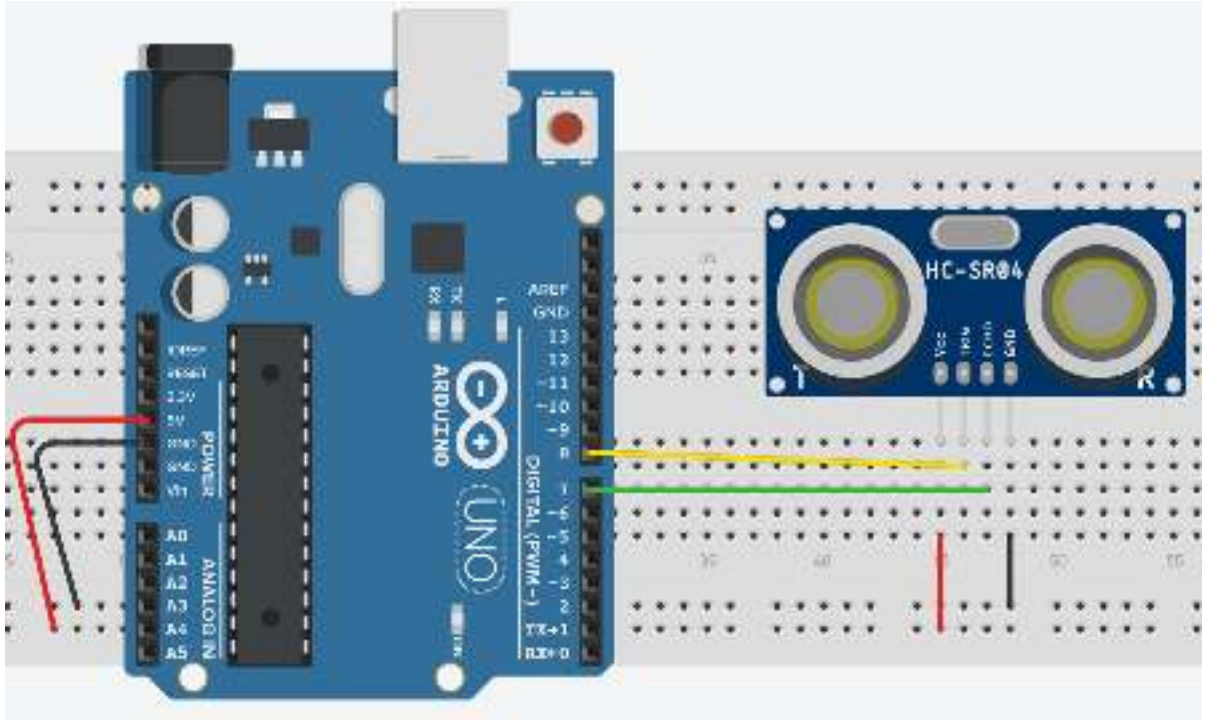
$$R = \frac{V_{\text{fuente}} - V_{\text{LED}}}{I}$$

$$R = \frac{5V - 1.8V}{0.02A} = \frac{3.2V}{0.02A} = 160\,\Omega$$

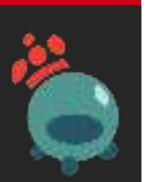
- Botón/switch
  - Mantiene cerrado el circuito
  - Hasta que se presiona
  - ¿Cómo podemos saber si está presionado?

Sensores: HC-SR04

- Ultra-Sonido (HC-SR04)
  - Rango: 2-400 cm
  - Precisión: 3mm
  - Ángulo de medición: 15°
- VCC y GND
- Trigger y ECHO



- Mide distancias a objetos utilizando la misma técnica que un SONAR. Emite un sonido a frecuencias determinadas (~40 KHz) y mide del tiempo de vuelo que tarda en recibirse su eco (rebote sobre el objeto)



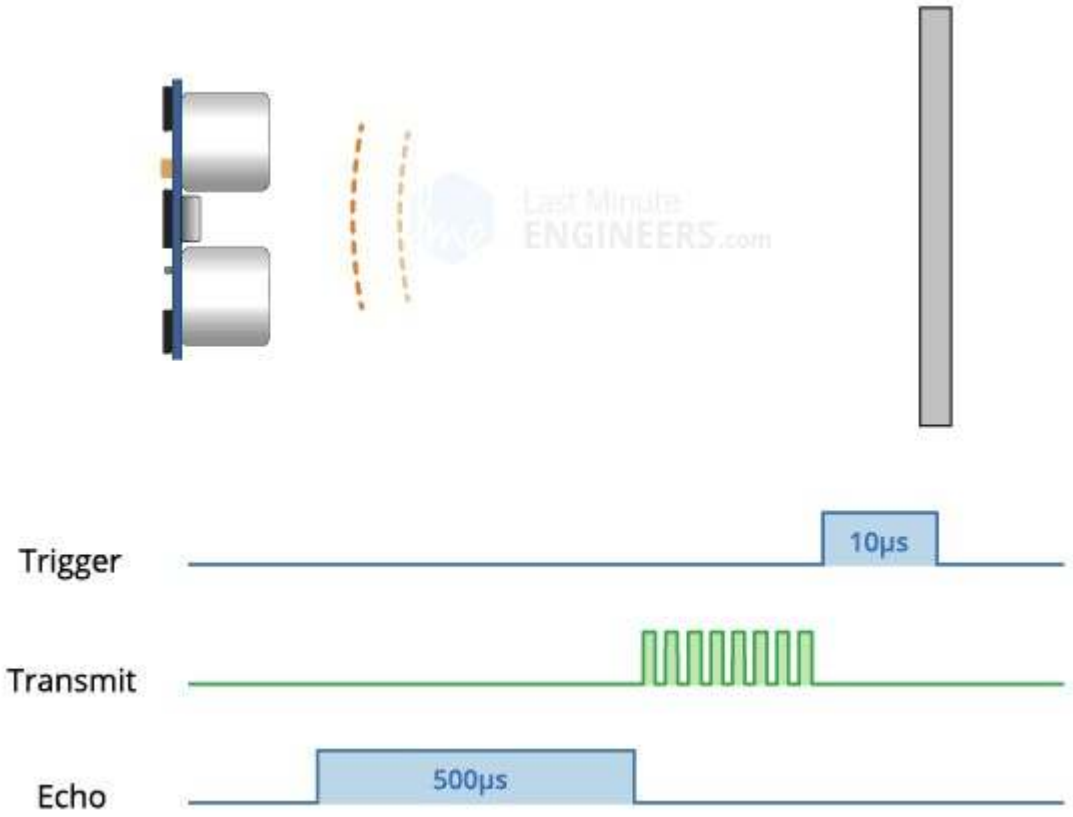
Sensores: HC-SR04

```
digitalWrite(PIN_TRIGGER,HIGH);
delayMicroseconds(10);
digitalWrite(PIN_TRIGGER,LOW);

distancia=pulseIn(PIN_ECHO,HIGH);
```

- *pulseIn*: Devuelve el tiempo en micro-segundos hasta que aparece un pulso (HIGH o LOW).

Sensores: HC-SR04

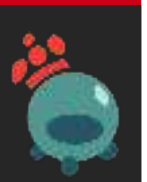
- Ejemplo de funcionamiento: LINK



Sensores: HC-SR04

```
digitalWrite(PIN_TRIGGER,HIGH);
delayMicroseconds(10);
digitalWrite(PIN_TRIGGER,LOW);

distancia=pulseIn(PIN_ECHO,HIGH);
```

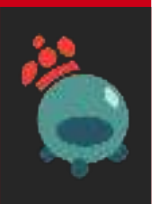
- *pulseIn*: Devuelve el tiempo en microsegundos hasta que aparece un pulso (HIGH o LOW).
- Velocidad del sonido: 343 m/s



Sensores: HC-SR04

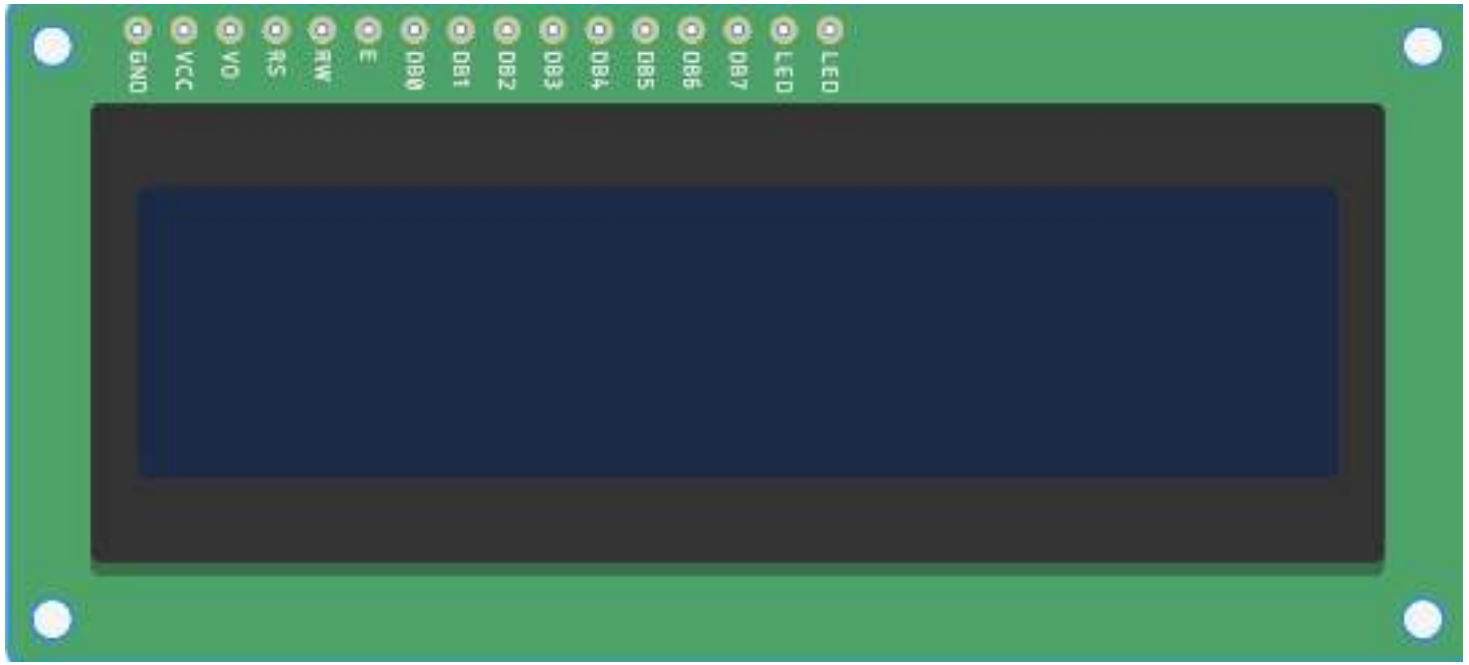
```
digitalWrite(PIN_TRIGGER,HIGH);
delayMicroseconds(10);
digitalWrite(PIN_TRIGGER,LOW);

distancia=pulseIn(PIN_ECHO,HIGH);
```

- *pulseIn*: Devuelve el tiempo en microsegundos hasta que aparece un pulso (HIGH o LOW).
- Velocidad del sonido: 343 m/s
  - 343 * $1e^2$ / $1e^6$ / 2 = 0.01715 cm/microsegundo.

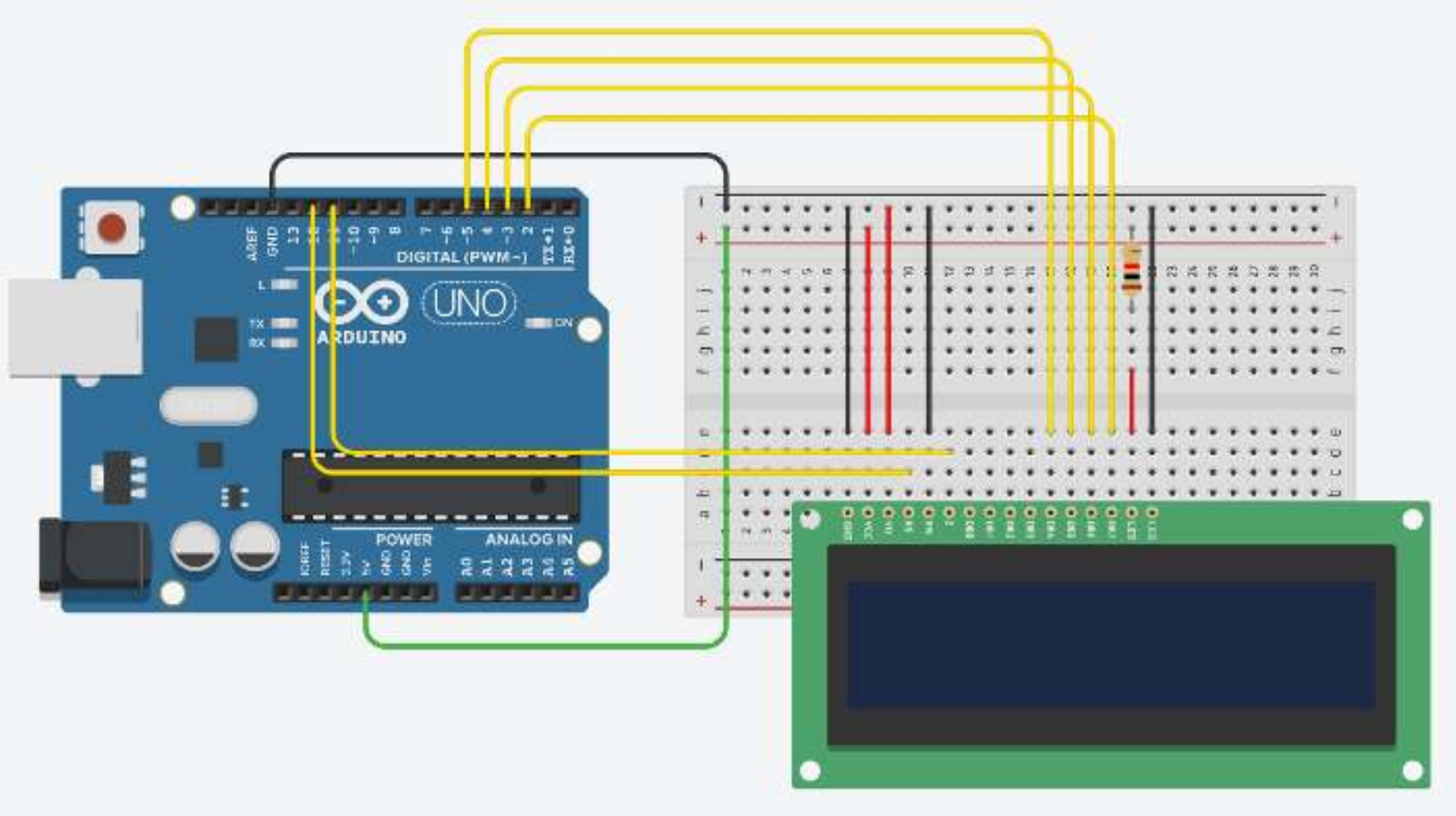


Sensores: LED 16x2

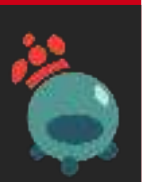
- LED 16x2
- Bus de 8 líneas
- Manejo de contraste
- https://www.arduino.cc/en/Reference/LiquidCrystal



Sensores:LED 16x2

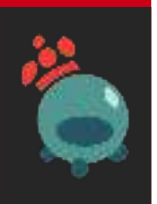


¿Como calcularías la latencia del planificador en Arduino?



Bibliografía

- [Libro] Arduino Cookbook, 3rd Edition, Abril 2020
  - Michael Margolis, Brian Jepson, Nicholas Robert Weldin
- [Libro] Exploring Arduino, 2nd Edition, Septiembre 2019
  - Jeremy Blum
- https://deepbluembedded.com/arduino-adc-analogread-analog-input/



URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC Programming Robot Intelligence

Sistemas Empotrados y de Tiempo Real

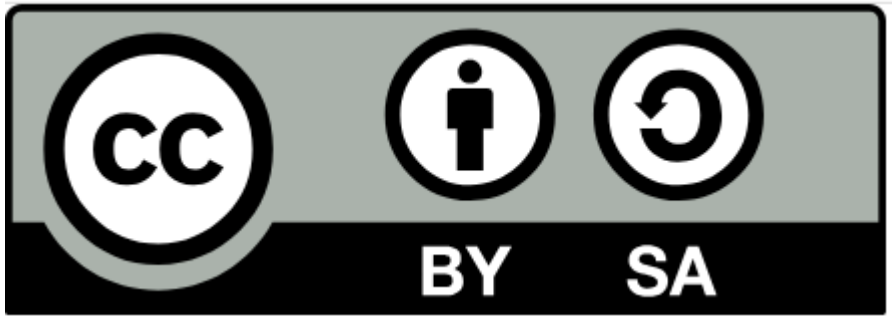
Desarrollo con Arduino

Watchdog y Lecturas Digitales

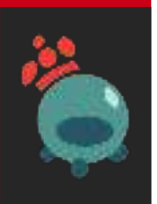
Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es

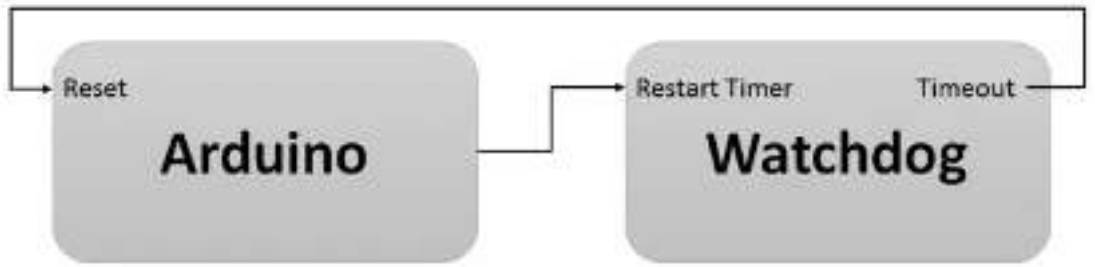


Watchdog



Watchdog

- Un watchdog es un mecanismo de seguridad que provoca un reset del sistema en caso de que éste se haya bloqueado o no responde al comportamiento deseado.
- El watchdog es un timer que va dentro de la arquitectura del microcontrolador y es independiente del resto de timers y del sketch que se está ejecutando
- El watchdog utiliza una fuente de reloj interna de 128 kHz.



Watchdog

- Incluir librería en el scketch

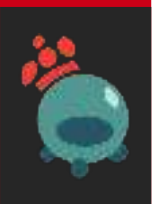
```
#include <avr/wdt.h>
```

- Desactivar el watchdog y configurarlo

```
wdt_disable();
wdt_enable(tiempo);
```

- Actualiza el watchdog para que no produzca el reseteo

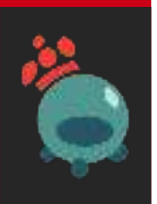
```
wdt_reset();
```



Watchdog

- Un watchdog es un mecanismo de seguridad que provoca un reset del sistema en caso de que éste se haya bloqueado o no responde al comportamiento deseado.

Tiempo	wtd_enable()
15 ms	WDTO_15MS
30 ms	WDTO_30MS
60 ms	WDTO_60MS
120 ms	WDTO_120MS
250 ms	WDTO_250MS
500 ms	WDTO_500MS
1 s	WDTO_1S
2 s	WDTO_2S
4 s	WDTO_4S
8 s	WDTO_8S

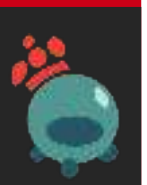


Watchdog

- Código ejemplo:

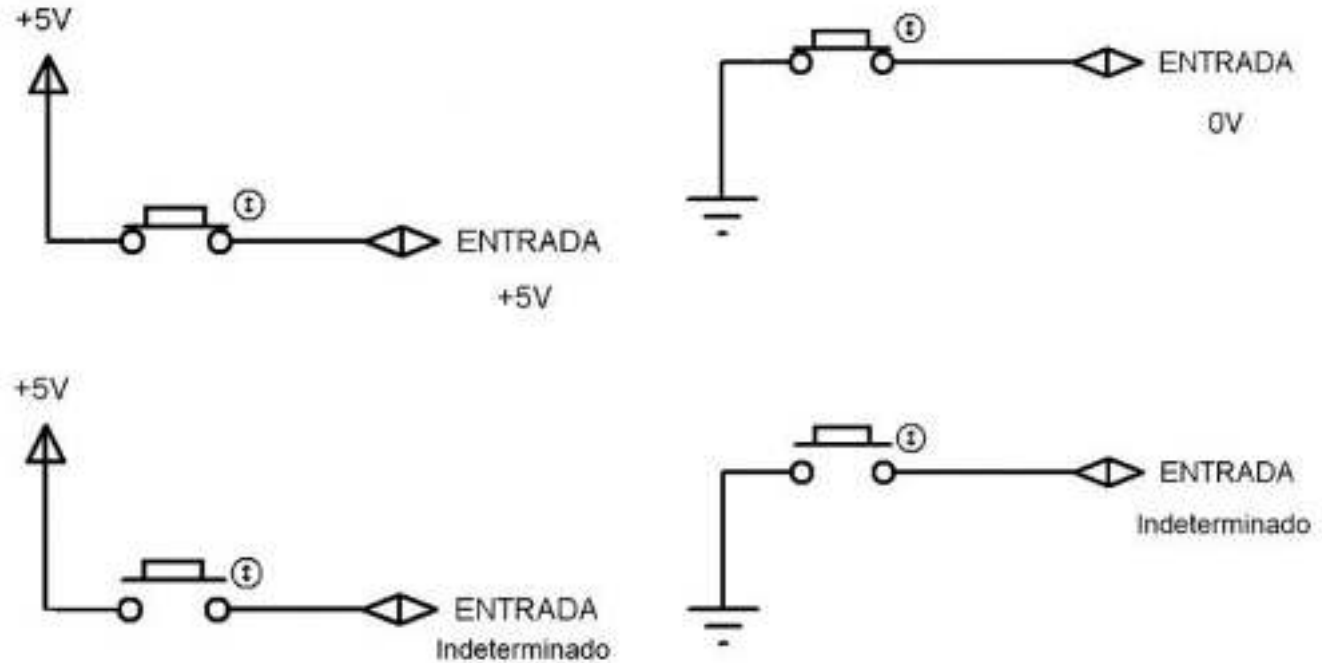
```
#include <avr/wdt.h>

int loops = 0;

void setup()
{
  Serial.begin(9600);
  wdt_disable();
  wdt_enable(WDTO_8S);
  Serial.println("iniciando sketch");
  delay(1000);
}

void loop()
{
  Serial.println("Iteracion: " + String(loops));
  delay(1000);
  if (loops >= 10) {
    Serial.println("Dentro de bucle largo");
    for (int i = 0; i < 100; i++) {
      Serial.println(i);
      delay(1000);
    }
  }

  wdt_reset();
  loops++;
}
```

Lecturas Digitales



Lecturas Digitales

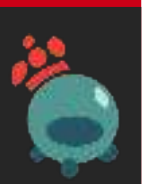
- Lecturas confiables desde un botón o switch
- Cuando una entrada digital no está conectada a nada se define que tiene un estado de *alta impedancia* o *flotante*.
- Un estado flotante nos puede llevar a valores no esperados en las entradas digitales.



Lecturas Digitales

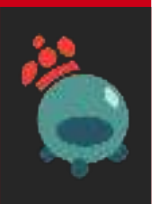
- Código ejemplo:

```
const int inputPin = 2;


void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(inputPin, INPUT);
}

void loop(){
  int val = digitalRead(inputPin);
  Serial.println(val);
  digitalWrite(LED_BUILTIN, val);

  delay(500);
}
```



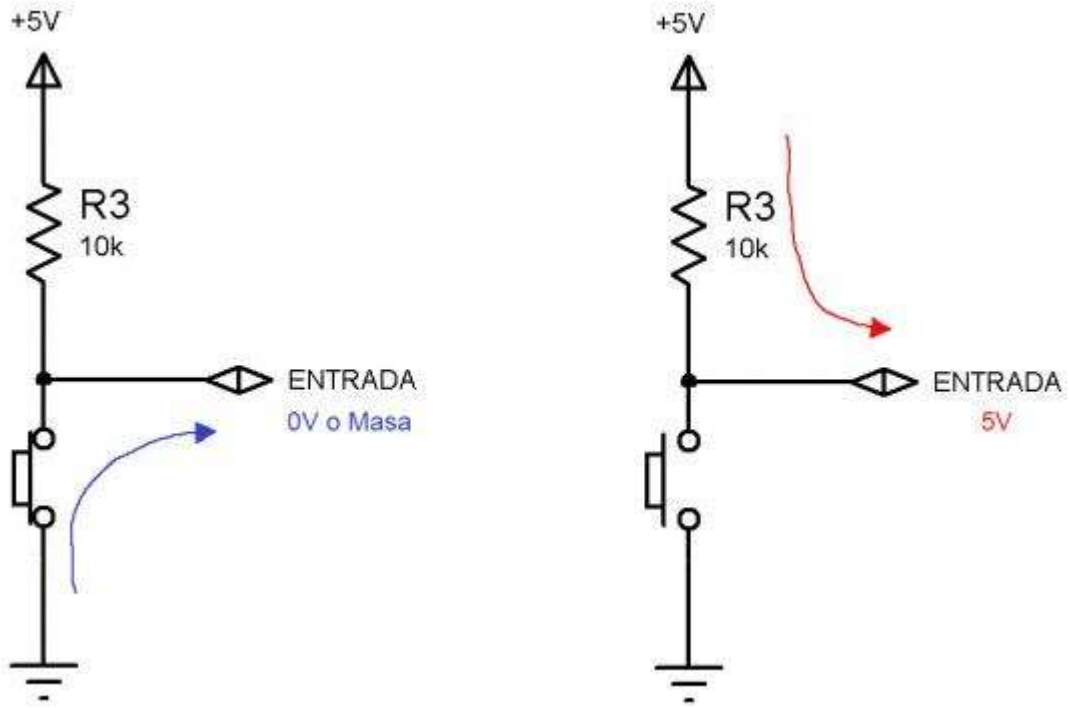
Lecturas Digitales

- Resistencias *pull up* o *pull down*
- Resistencias normales pero que debido a su disposición en un circuito electrónico establecen el estado lógico de un pin cuando se encuentra en estado reposo.
- Normalmente se utilizan resistencias de 10KΩ



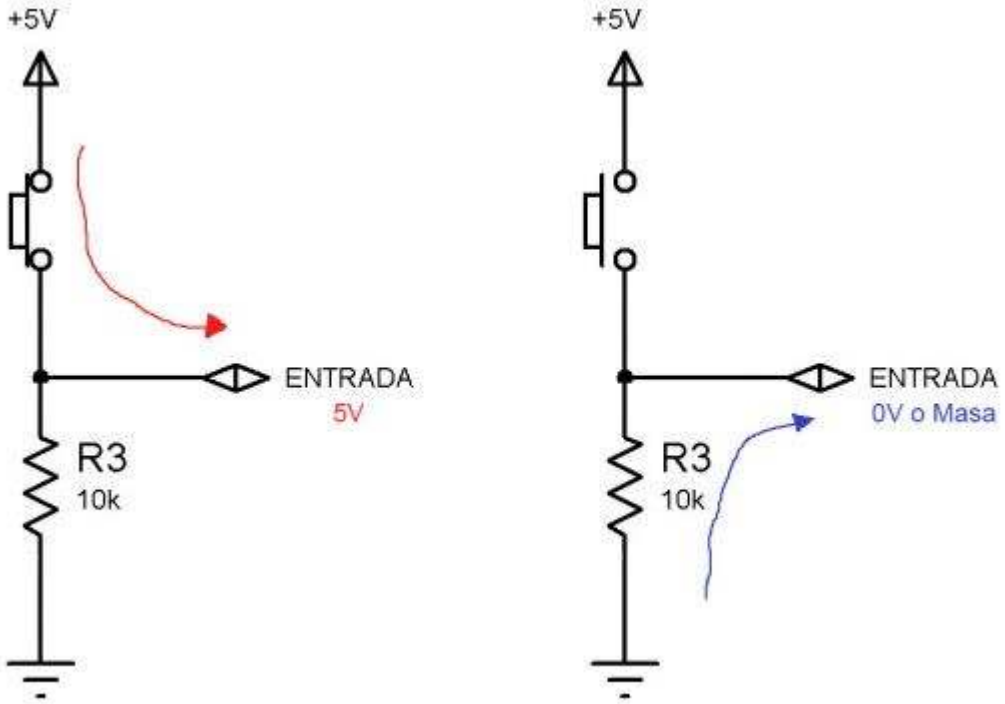
Resistencias Pull-up

- Nos aseguramos que la entrada digital va a tener un valor HIGH hasta que se produzca la pulsación del pulsador, en ese momento el valor se establece a LOW



Resistencias Pull-down

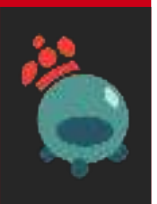
- Nos aseguramos que la entrada digital va a tener un valor LOW hasta que se produzca la pulsación del pulsador, en ese momento el valor se establece a HIGH



Resistencia pull-up con Arduino

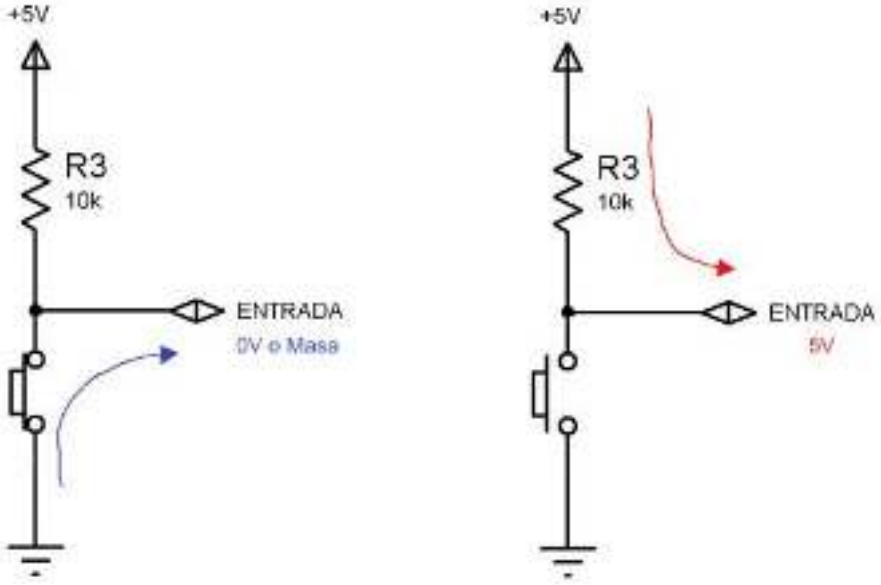
- El microcontrolador Atmega del Arduino tiene resistencias pull-up internas
- Evitamos mas componentes en nuestro circuito
- Solo podemos usar PULLUP.

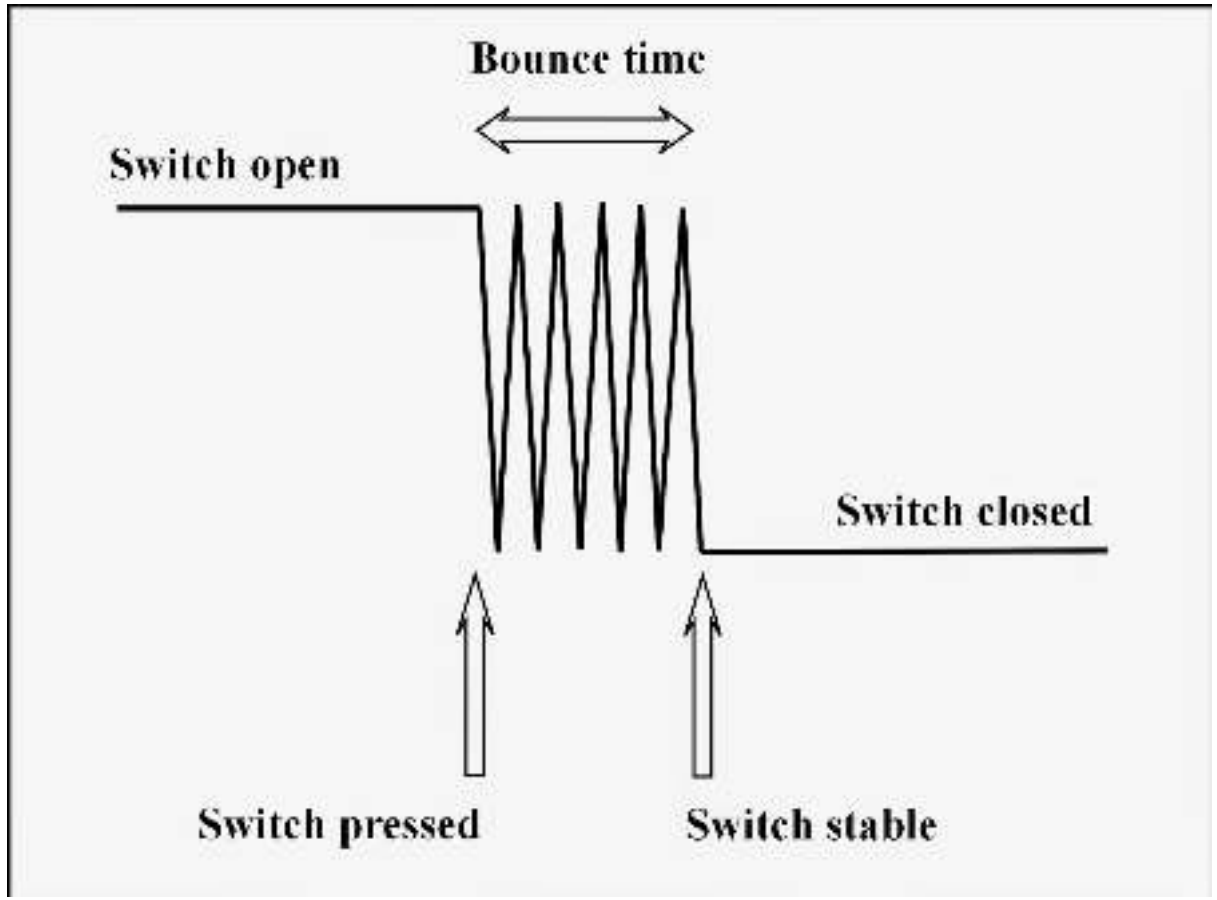
```
pinMode(inputPin, INPUT_PULLUP);
```



Rebotes (Bounce)

- En el instante de abrir o cerrar un interruptor, se pueden producir rebotes hasta que se estabilice la señal.



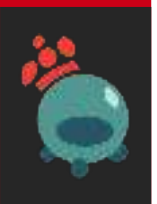


Rebotes (Bounce)

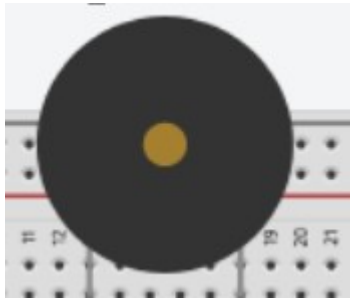
- Soluciones:
  - Uso de condensadores en la entrada
  - Añadir código para detectar los “rebotes”

- Ideas sobre código a añadir

- Revisar
  - Capitulo 5.4 de Arduino Cookbook, 3rd Edition
    - Determining How Long a Switch Is Pressed

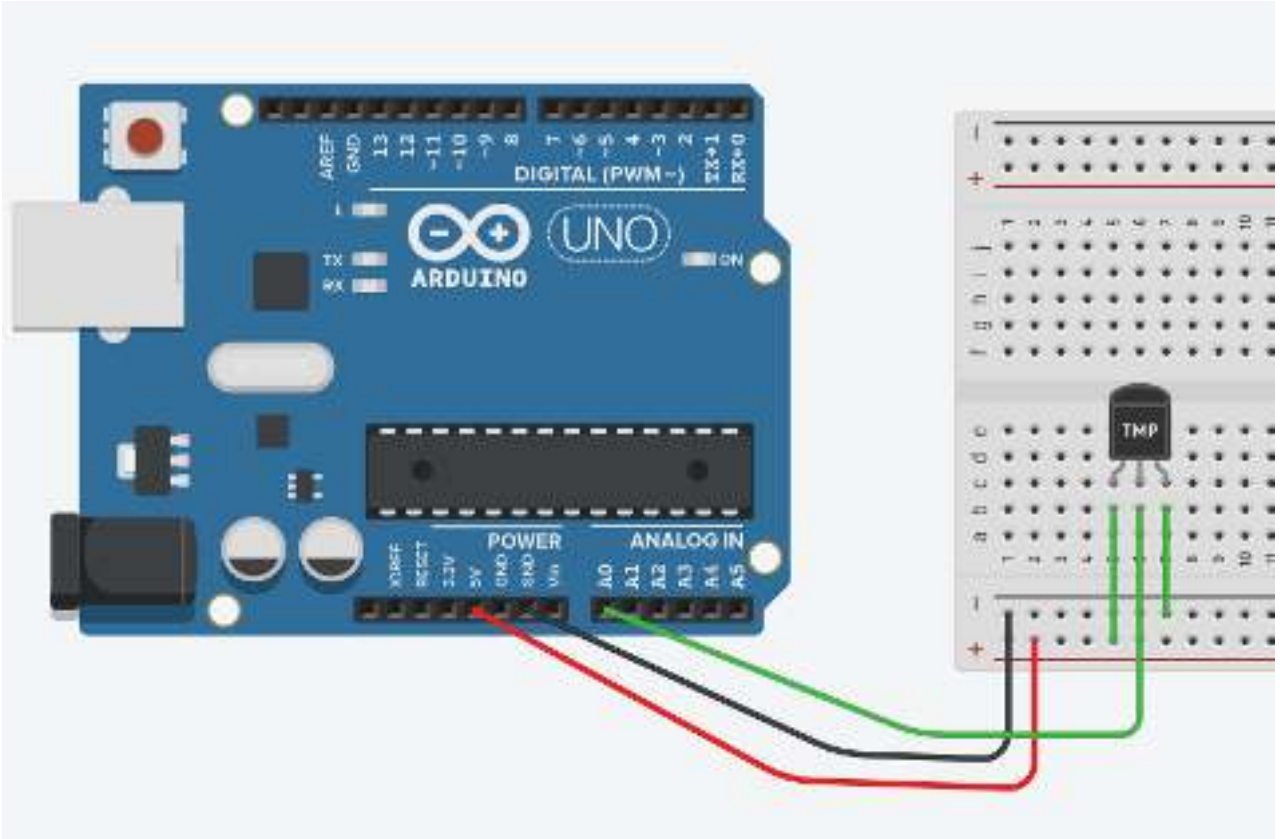


Sensores



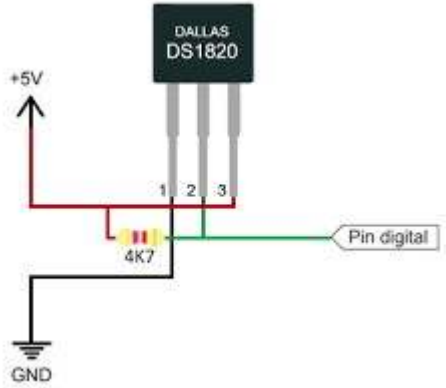
Sensores: TMP36

- Sensor Temperatura (TMP36)
- Sensor Analógico
- Rango: −40°C to +125°C
- Resolución: 0.5°C
- 10 mV/°C



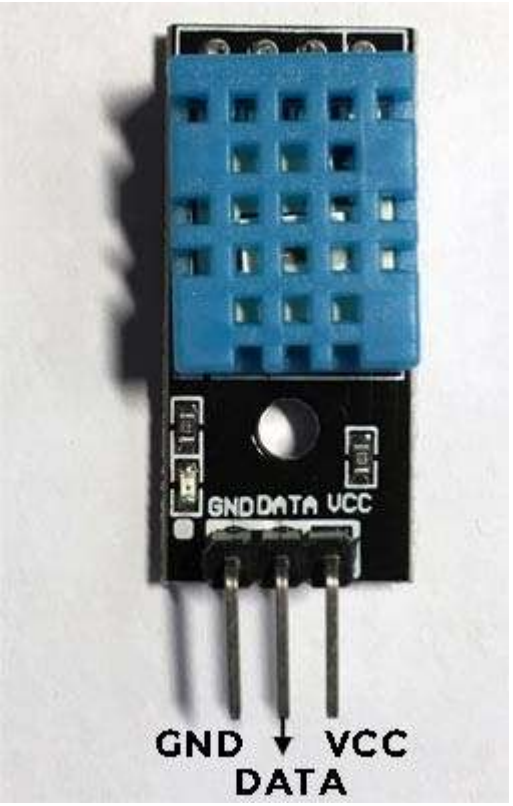
Sensores: TMP36

- TEMP36, DS1820, …



Sensores DHT-11/20

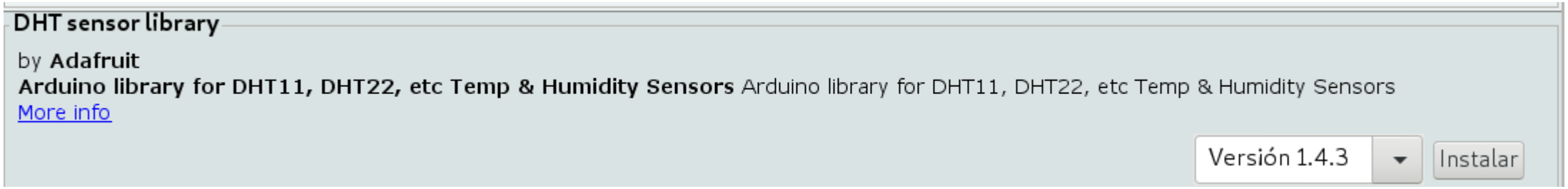
- Sensor de temperatura y humedad
- Alta fiabilidad y estabilidad debido a su señal digital calibrad
- Se trata de un sensor analógico, pero dentro de la PCB se realiza la conexión digital.
- La salida digital siempre va conectada en modo pull-up



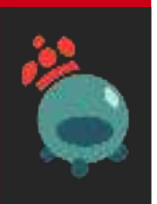
0011 0101	0000 0000	0001 1000	0000 0000	0100 1001
8 bits humedad	**8 bits humedad**	**8 bits temperatura**	**8 bits temperatura**	**bits de paridad**

Sensores DHT11

- Necesidad de instalar librerías adicionales desde el IDE de arduino.

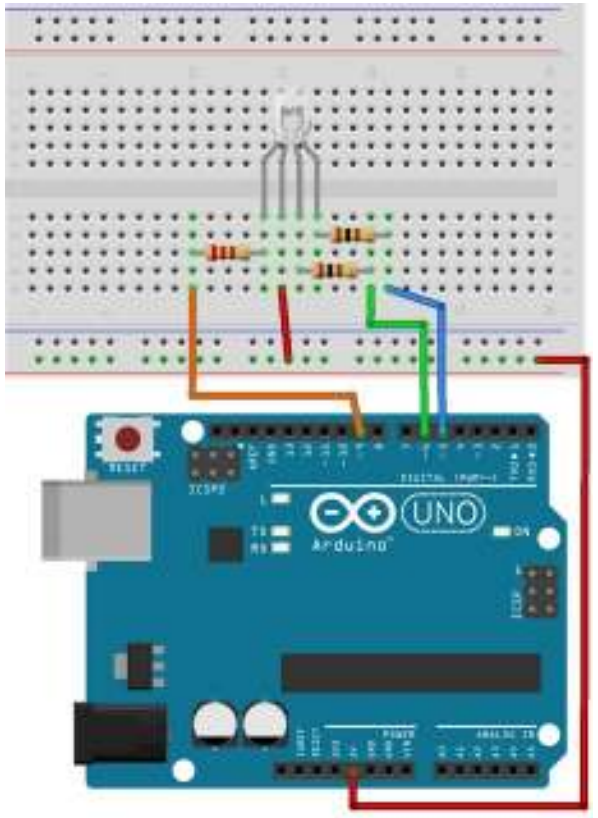


- Documentación
  - https://github.com/adafruit/DHT-sensor-library



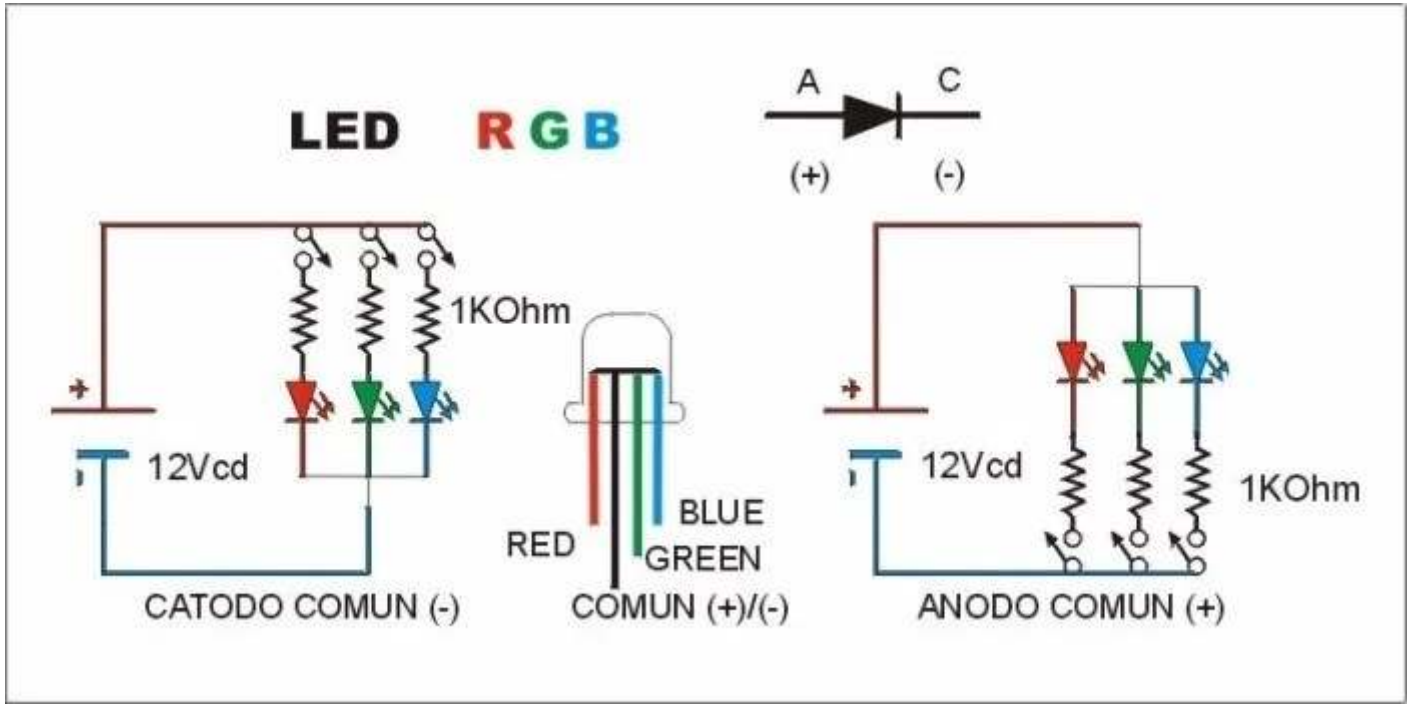
Sensores: LED RGB

- LED RGB
- Utilizan los pines “analógicos” PWM
- Basado en el concepto “duty cycle”



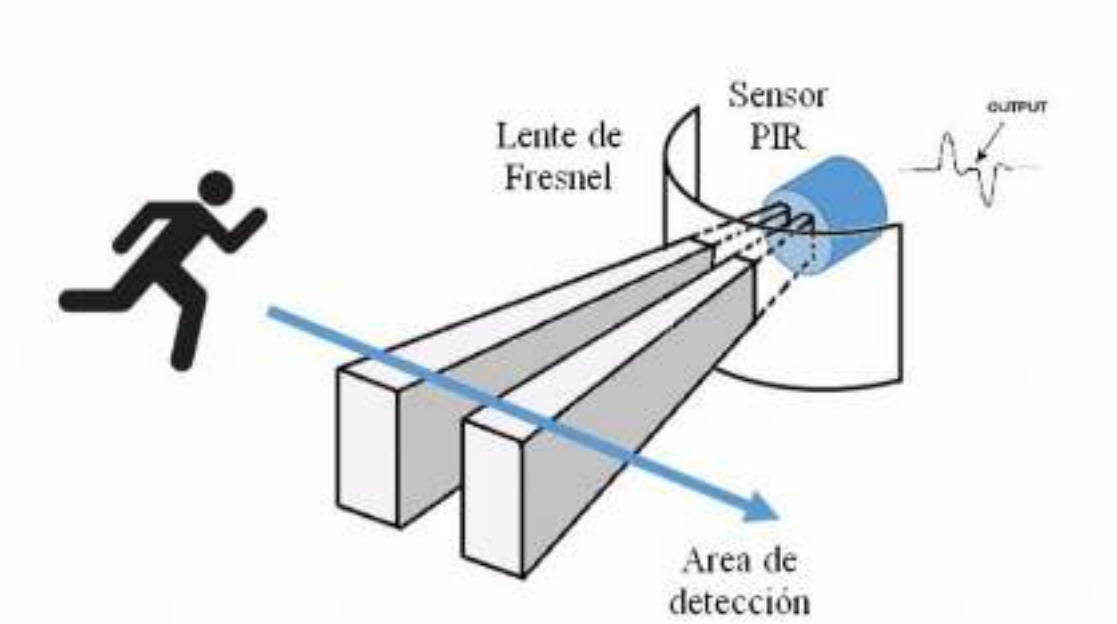
Sensores: LED RGB

- Ánodo o Cátodo común
- 3 pines (R,G,B)
- Uso de analogWrite



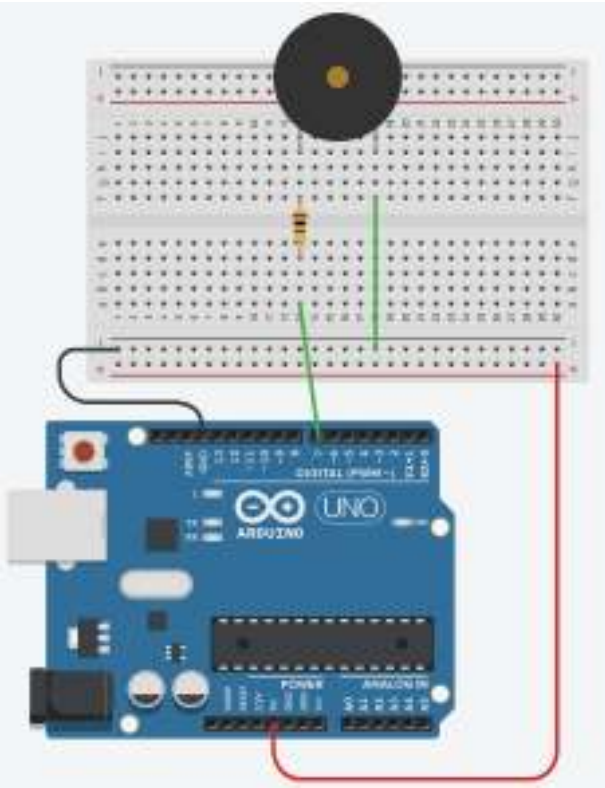
Sensores: PIR

- Los sensores infrarrojos pasivos (PIR) son dispositivos usados para la detección de movimiento.
- Se basan en la medición (recepción) de la radiación infrarroja
- Lectura digital (HIGH presencia, LOW no-presencia)



Sensores: Zumbador

- Los Zumbadores (buzzer), en ocasiones denominados zumbadores, son dispositivos que generan un sonido de una frecuencia determinada y fija cuando son conectados a tensión.
- Arduino dispone de 2 funciones para hacer funcionar estos dispositivos.
  - Tone() y noTone()
- Genera una onda cuadrada a la frecuencia especificada.



https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC Programming Robot Intelligence

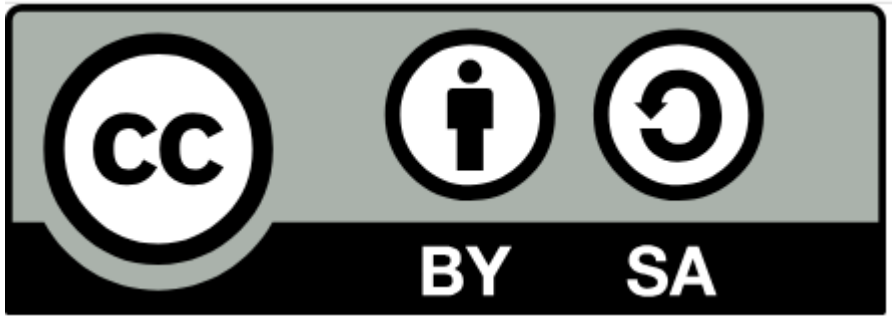
Sistemas Empotrados y de Tiempo Real

Arduino

Grado en Ingeniería de Robótica Software

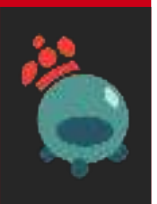
Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es



Arduino

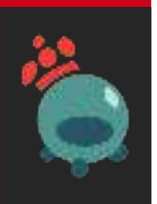
- Arduino es una compañía de software y hardware libre.
- Desarrollo de placas hardware de bajo coste para construir dispositivos digitales que interaccionan con el mundo real.
- Focalizado en acercar y facilitar el uso de dispositivos electrónicos y programación en sistemas empotrados.
- Tiene una gran comunidad de desarrolladores alrededor.
- La funcionalidad de las placas arduino pueden extenderse gracias a los ‘shields’ o ‘hats’



ARDUINO

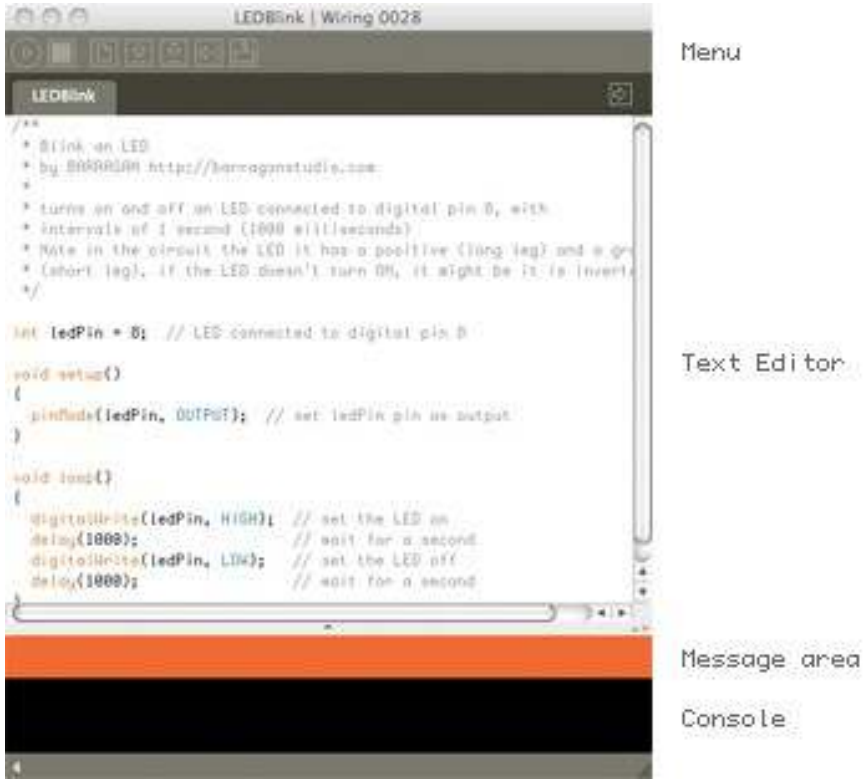
Historia

- El proyecto **Arduino** comenzó en 2005, como un proyecto enfocado a estudiantes (Instituto IVREA Italia).
- El objetivo del proyecto era reducir costes de las placas usadas en asignaturas para crear proyectos digitales.
- Ya existía un proyecto denominado **Wiring**, que permitía a través de un IDE programar de una manera sencilla una placa basada en ATmega168.
- Massimo Banzi, David Cuartielles y David Mellis, dieron soporte a Wiring para el micro ATmega8 (mucho más baerato).
- Crearon el proyecto Arduino, que tiene su origen en el bar donde se reunían "Bar di Re Arduino"
- En el año 2017, Massimo Banzi anunció la creación de la «Fundación Arduino»
- Documental ARDUINO: https://www.youtube.com/watch?v=mltWc9_C9gs



Wiring / Arduino

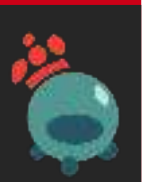
- **Wiring** fue un proyecto creado por Hernando Barragán en 2003 como parte de su trabajo de maestría. Su objetivo era facilitar el uso de la electrónica a artistas y diseñadores sin experiencia en programación ni electrónica.



Software/Hardware Libre

- Los diseños de referencia de hardware se distribuyen bajo licencia Creative Commons Attribution Share-Alike 2.5 y están disponibles en el sitio web de Arduino

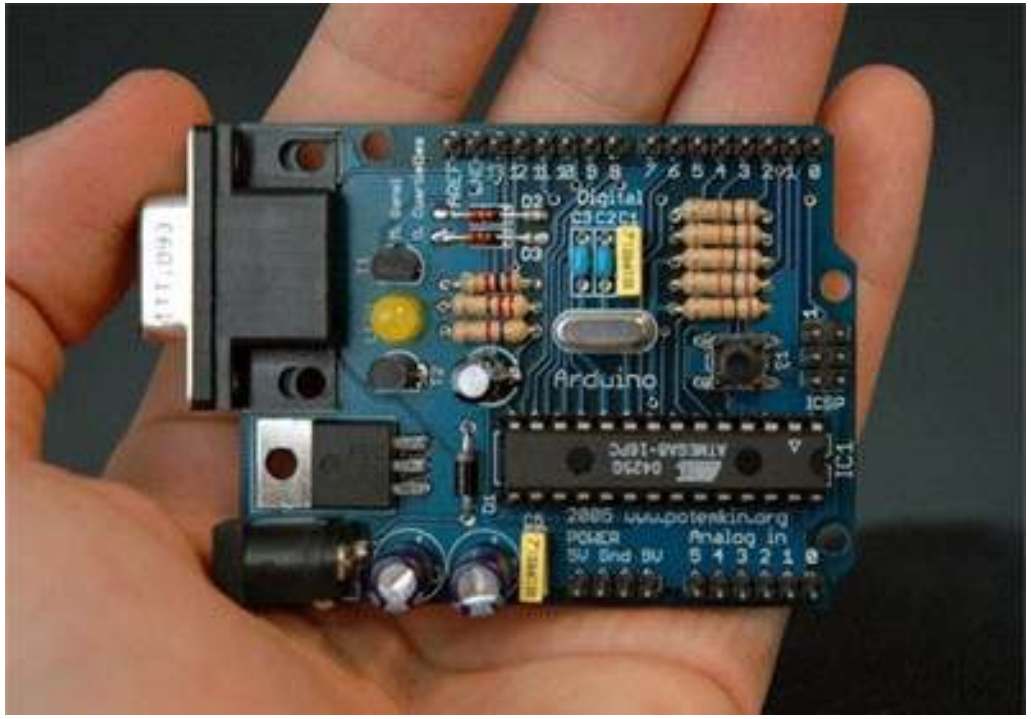
¿Implicaciones de software/hardware libre?



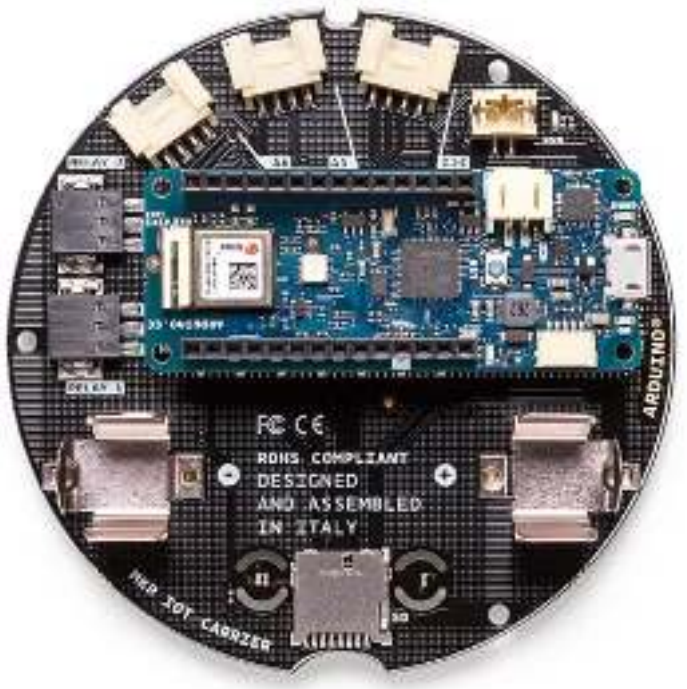
Evolución

2006, primer arduino liberado.
Conexión serie.

No SMD (Surface Mounting Device)

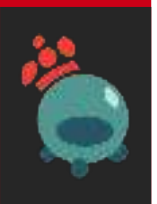


2020, Oplà IoT Kit.
Conectividad WiFi.



Características

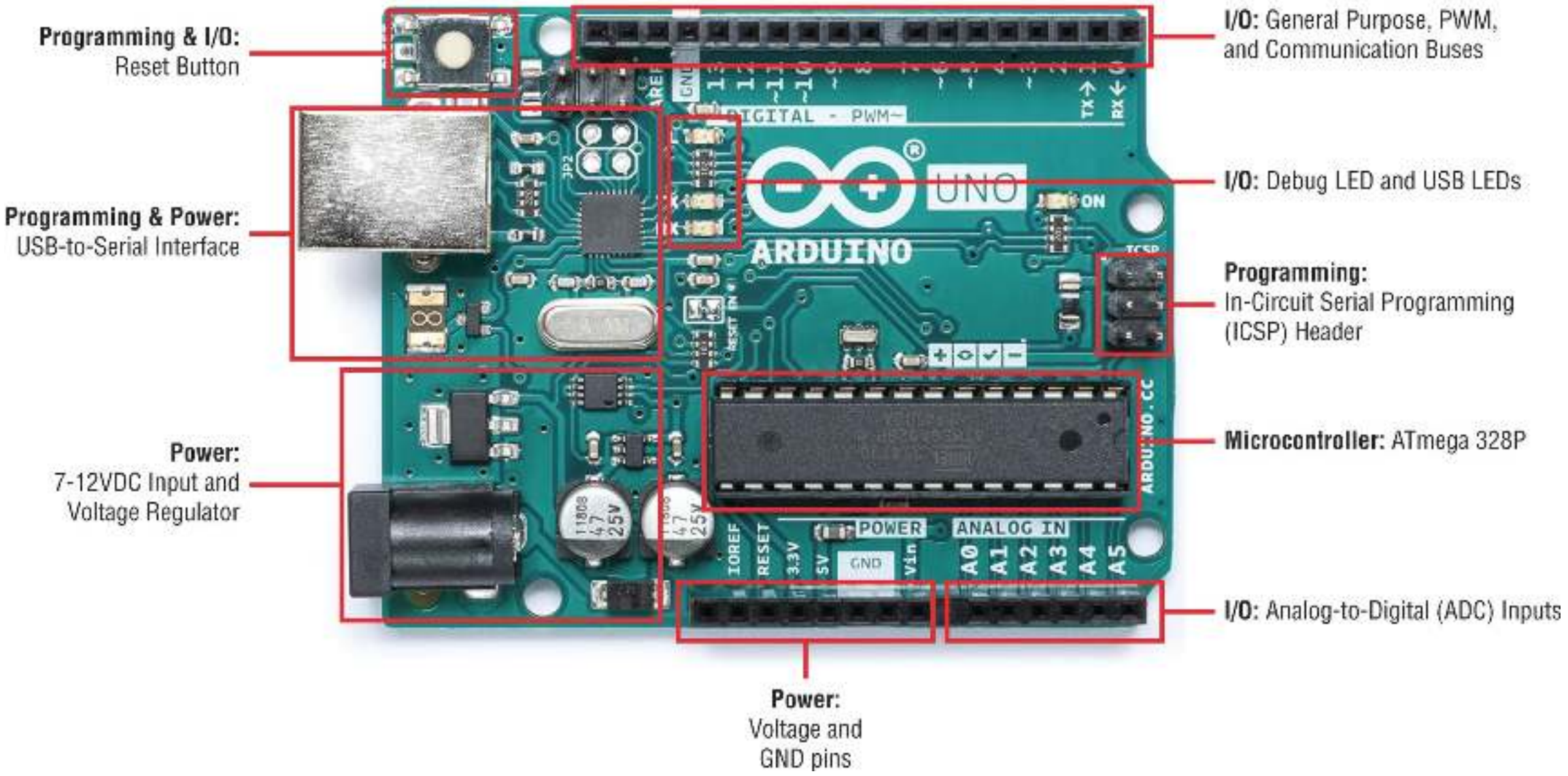
- Circuitos/placas de bajo coste.
- La mayoría de las placas Arduino se basan en un microcontrolador **AVR Atmel-8 bits** (ATmega8, ATmega168, ATmega328, ATmega1280, ATmega2560),
- Cada microcontrolador consta de diversas cantidades de memoria flash, pines y funciones.
- Las placas arduino además incorporan otros componentes para facilitar el uso y programación (conexión usb, regulador de voltaje, pines analógicos y digitales, etc).



ATmega328p

- AVR de 8 bits
- RISC
- 32 KB Flash
- 1 KB EEPROM
- 2 KB de SRAM
- 23 líneas de E/S
- Interrupciones, watchdog, ADC, serie
- 1 MIPS
- No es hardware libre.

Características



* Imagen obtenida de Exploring Arduino, 2nd Edition

Placas Arduino

ATmega2560
54 pines digitales

ARM
Bajo coste
WiFi

ARDUINO **NANO**

ARDUINO **MEGA**

ARDUINO **LEONARDO**

ATmega32u4
Soporte USB

ATmega328P
Modelo de referencia

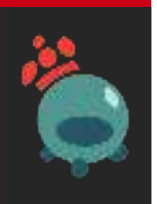
ARDUINO **UNO**

ARDUINO **YUN**

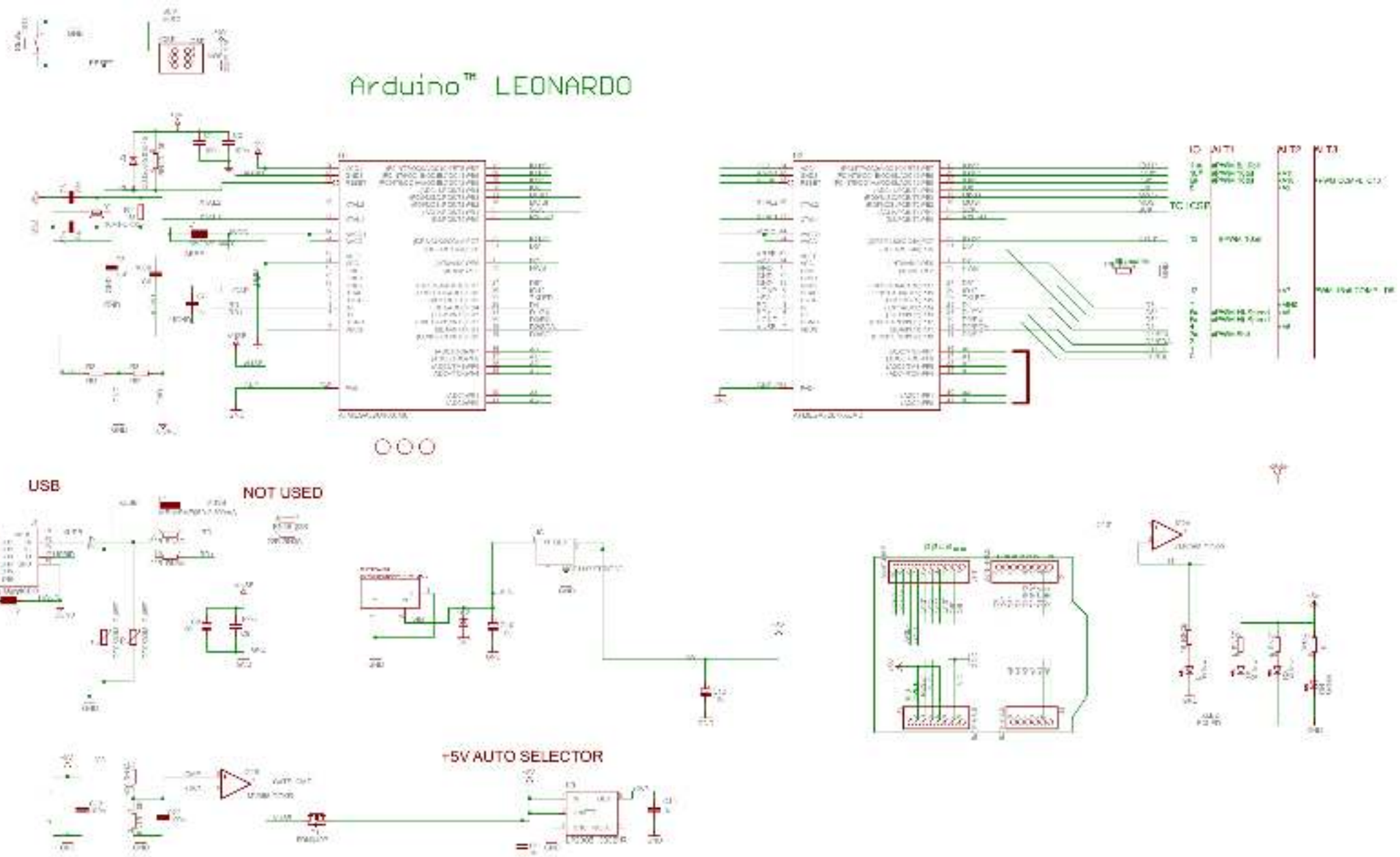
Linux
USB
Ethernet

Arduino Leonardo

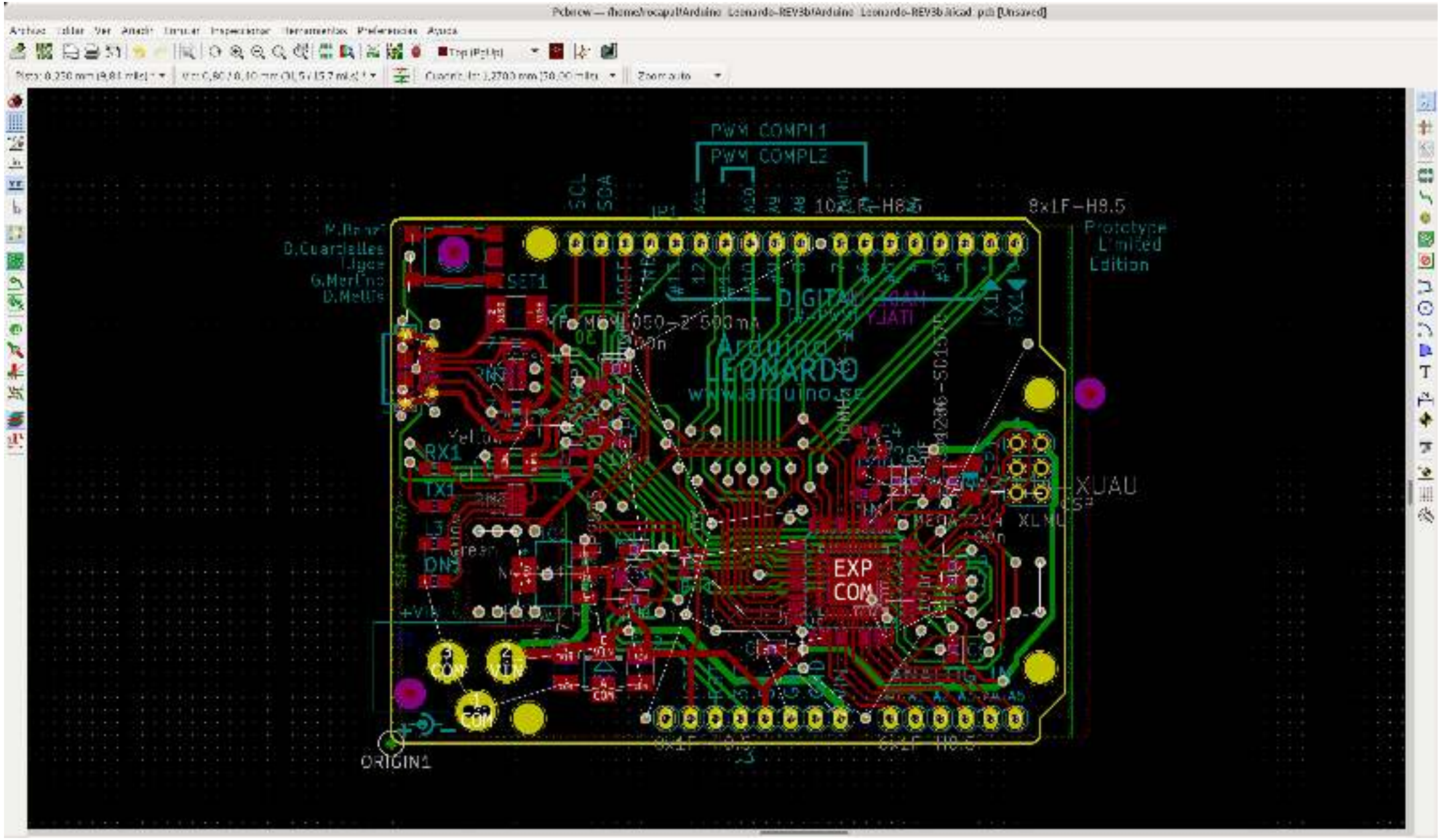
- Los beneficios del hardware libre es que toda la documentación está disponible, pudieron replicar y clonar cualquier placa arduino.
- Datasheet del microcontrolador Atmel ATmega de Arduino Leonardo
  - https://www.microchip.com/wwwproducts/en/atmega32u4
- EAGLE files con esquemas para los makers.
  - https://www.arduino.cc/en/uploads/Main/arduino-leonardo-reference-design_3b.zip
- Esquemas electrónicos de la placa Arduino UNO.
  - https://www.arduino.cc/en/uploads/Main/arduino-leonardo-schematic_3b.pdf



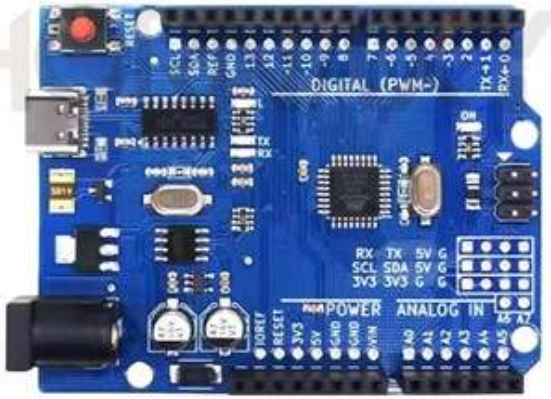
Arduino Leonardo



Arduino Leonardo



Hardware Libre



Ecosistema Arduino

- Familia Clásica

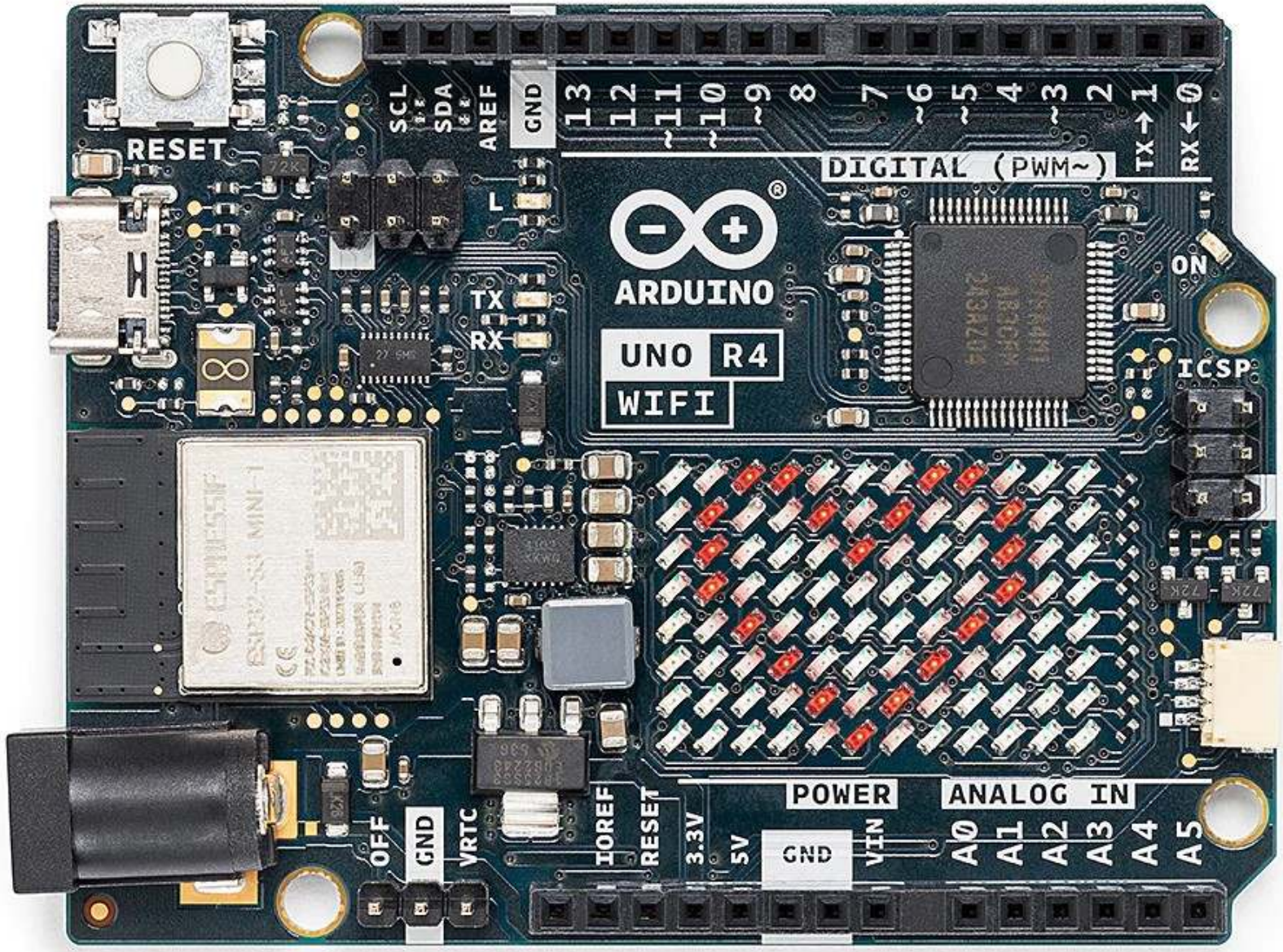
			
Arduino UNO R3	Arduino Mega 2560 Rev3	Arduino Leonardo	Arduino UNO Mini Limited Edition
			
Arduino Due	Arduino Micro	Arduino Zero	Arduino UNO WiFi Rev2

Shields

Shields

		
Arduino Motor Shield Rev3	Arduino 4 Relays Shield	Arduino Ethernet Shield Rev2

Arduino UNO R4 WiFi



Ecosistema Arduino

Kits

		
Arduino Starter Kit	Arduino Oplà IoT Kit	Arduino Sensor Kit

Ecosistema Arduino

Familia Nano

- Compactos
- Bluetooth
- Wifi
- Sensores
- MicroPython
- ML

- https://projecthub.arduino.cc/?value=tinyml

Ecosistema Arduino

Familia MKR

- WiFi
- Bluetooth
- LoRa
- Sigfox
- NB-IoT
- ARM Cortex-M0 32-bit
  - SAMD21

https://microchipdeveloper.com/32arm:samd21-mcu-overview

Arduino Pro

- Arm® Cortex® -M7 at 480 MHz
- Arm® Cortex® -M4 at 240 MHz
- Tensor-flow lite
- Wifi-Bluetooth
- Pensado para la industria
- IA, IoT

Ecosistema Arduino

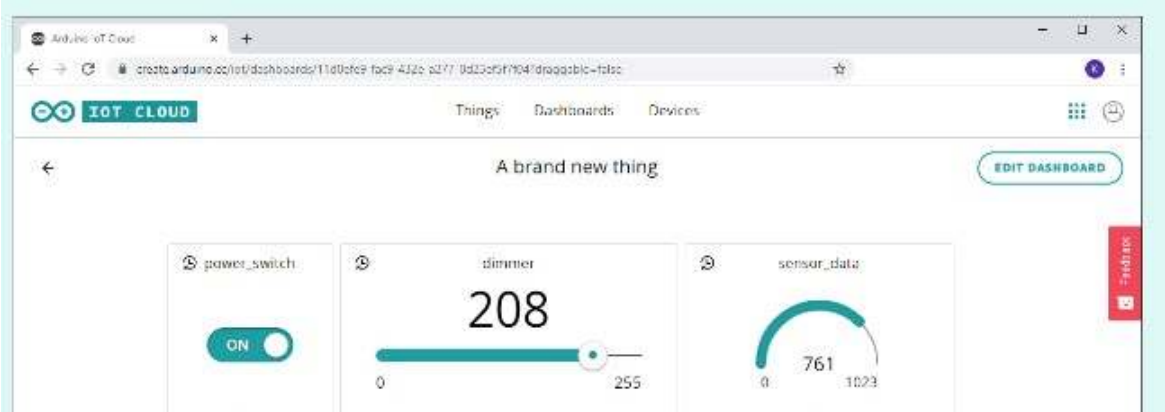
- Un **shield** es una placa de circuito impreso que se coloca sobre la placa Arduino para extender su funcionalidad.
- Se conecta a ella mediante el acoplamiento de sus pines sin necesidad de alguna otra conexión externa.
- No tienen funcionamiento autónomo.
- Mismo concepto aplicado en RaspberryPi (hats)

Arduino Oplà IoT Kit

- Arduino irrumpe con fuerza en IoT (Septiembre 2020)

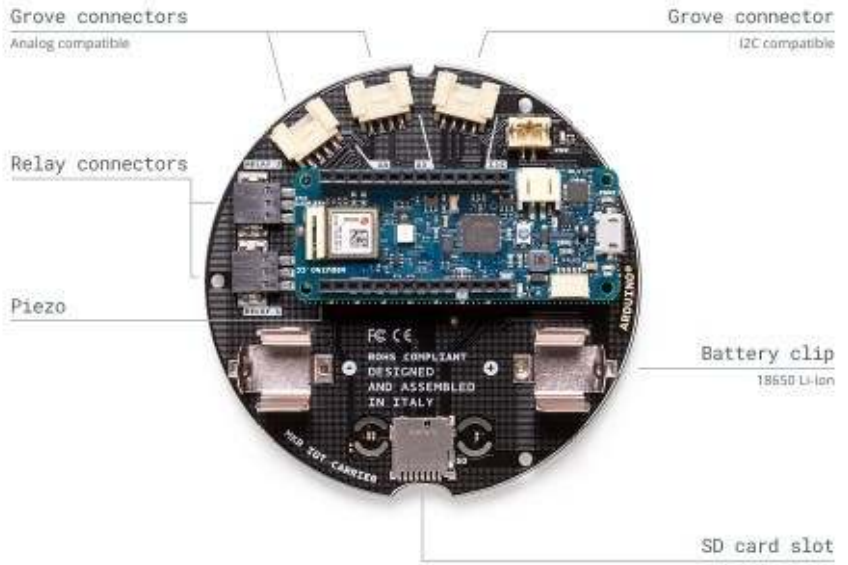
Arduino Oplà IoT Kit

- Incorpora el material necesario para montar 8 proyectos IoT sin necesidad de soldar.
  - Control remoto de luces, estación meteorológica personal, alarma de seguridad, jardín inteligente, control termostato, etc.
- Incorpora sensores de temperatura, presión, movimiento, leds, botones táctiles y conectividad WIFI
- Interactúa con Arduino IoT Cloud.



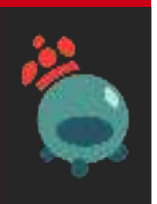
Arduino Oplà IoT Kit

Arduino Oplà IoT Kit

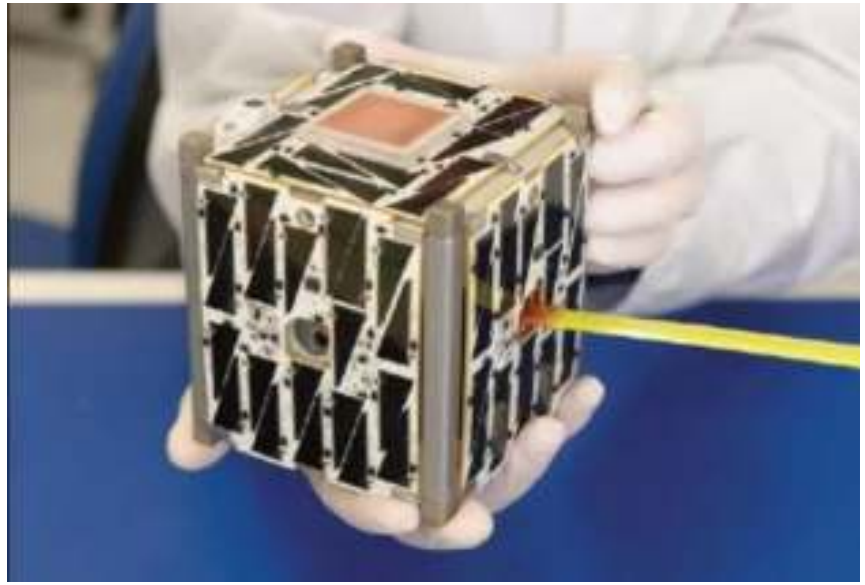


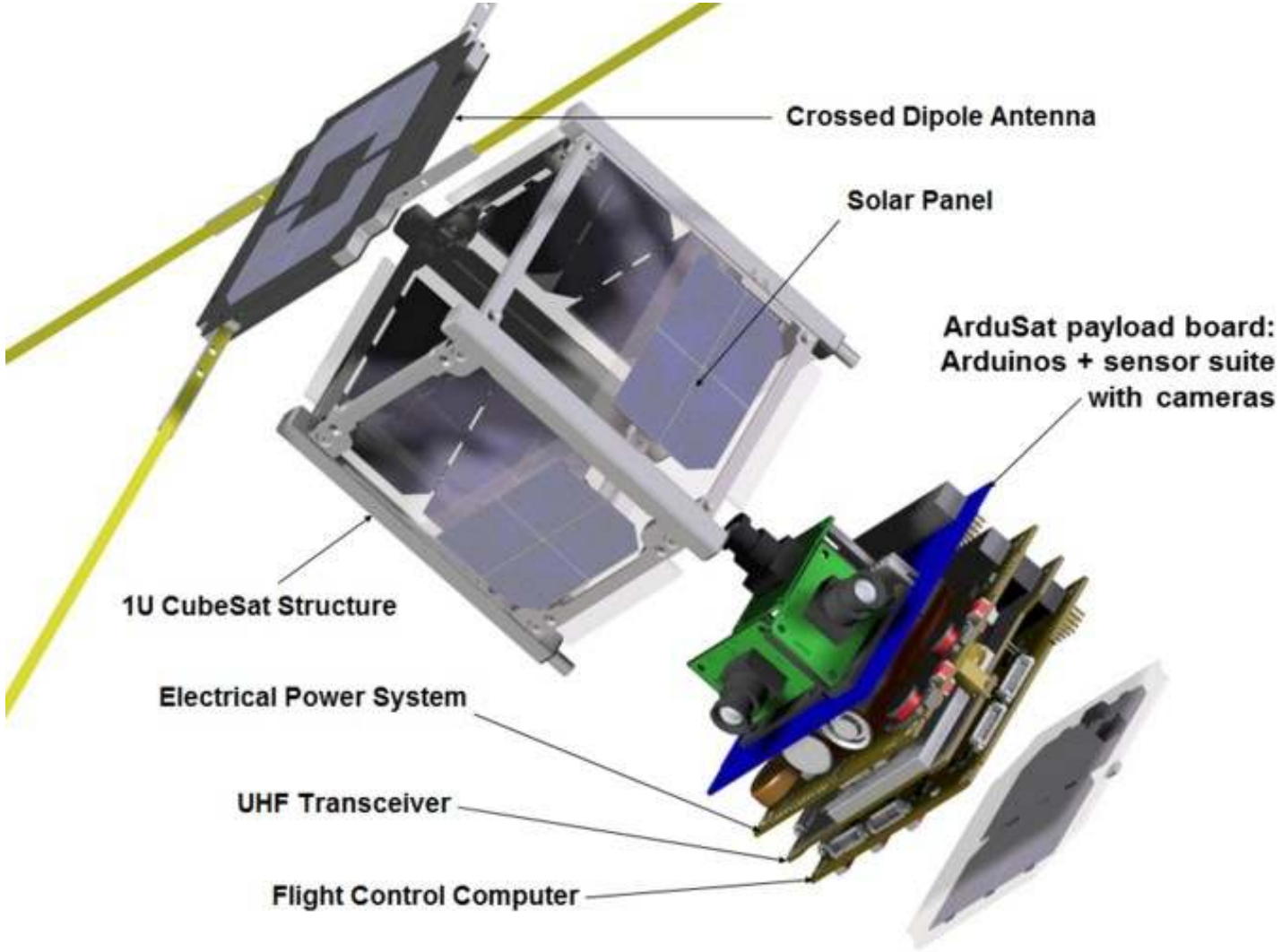
ArduSat

- **ArduSat** es un satélite en miniatura con software de código abierto, desarrollado por la empresa Arduino , basado en el CubeSat estándar.
- ArduSat es el primer satélite de código abierto que proporciona acceso al público en general al espacio.
- 16 nodos procesadores, ATmega328P, y 1 nodo máster: ATmega2561
- La idea era proporcionar una plataforma experimental para estudiantes interesados en diseñar sus propios experimentos.
- Contiene magnetómetro, giroscopio, acelerómetro, sensor temperatura infrarrojo, sensores luminosidad, contadores geiger, espectómetro óptico, cámara.
- 2 satélites idénticos ArduSat-1 y ArduSat-X, fueron lanzados el 2 de agosto de 2013, reentrando de nuevo en la atmósfera en Abril 2014.



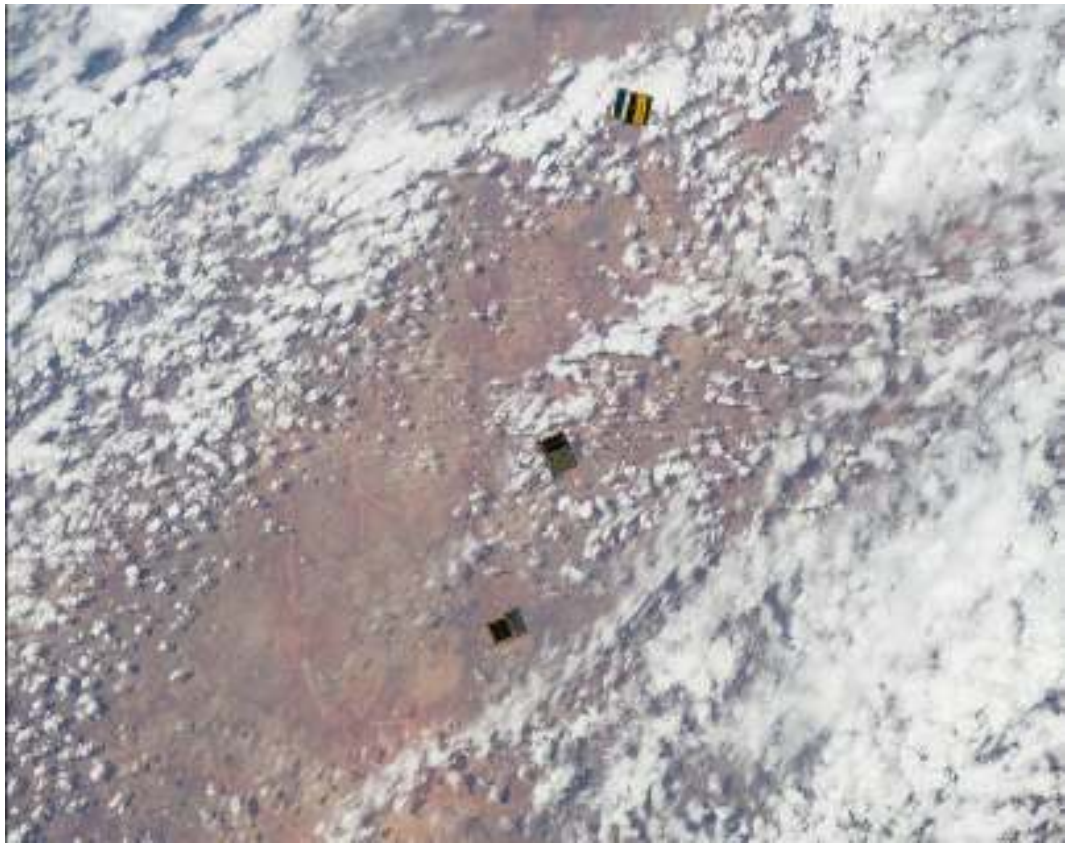
ArduSat




Crossed Dipole Antenna
Solar Panel
ArduSat payload board:
Arduinos + sensor suite
with cameras
1U CubeSat Structure
Electrical Power System
UHF Transceiver
Flight Control Computer

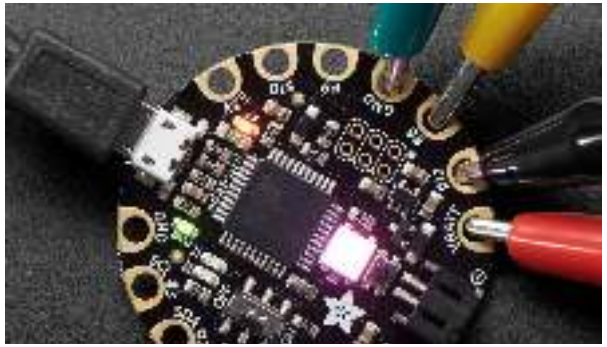
ArduSat

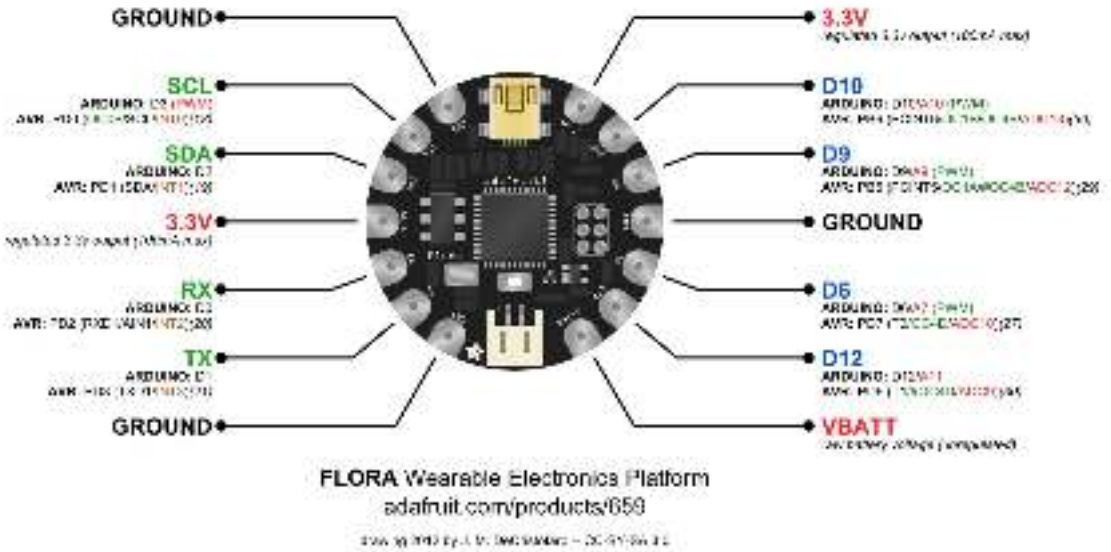
- ArduSat-1, ArduSat-X y Pico Dragon fotografiados desde la estación espacial después de su lanzamiento el 19/11/2013.



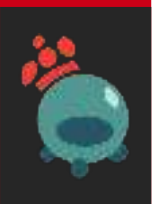
FLORA

- Plataforma electrónica para “wearables” basados y compatibles con Arduino
- Pensado para el mundo textil
- Desarrollado por Adafruit



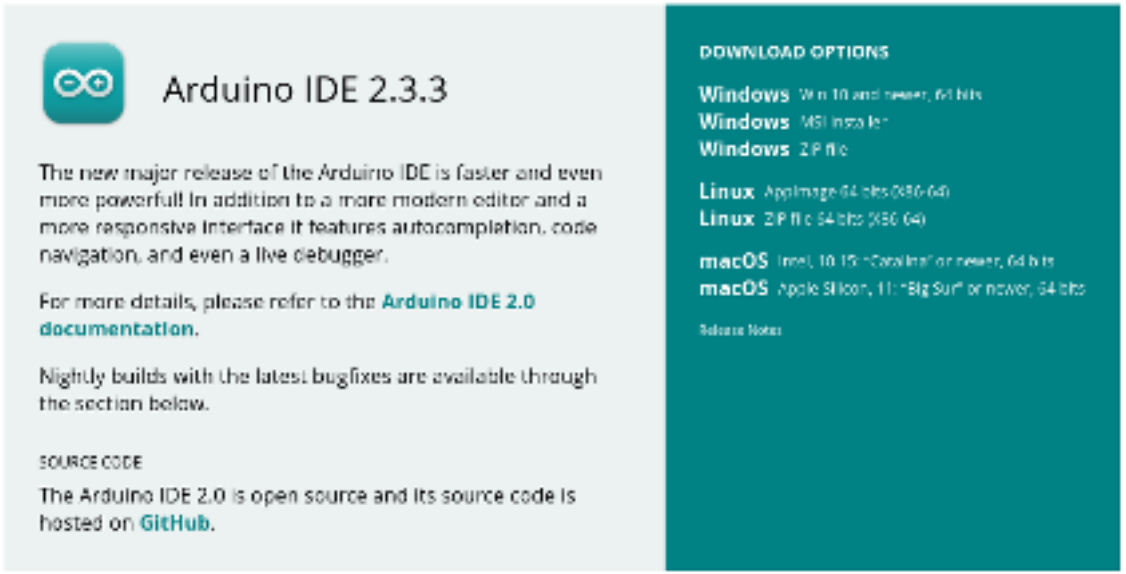


Desarrollo en Arduino

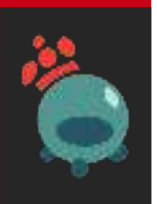


Instalación IDE

- Descargar desde https://www.arduino.cc/en/software
- Linux AppImage 64 bits (X86-64)

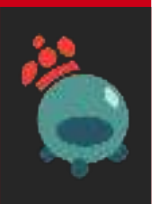


- Tienes la opción de usar Arduino Cloud, IDE integrado en el navegador.

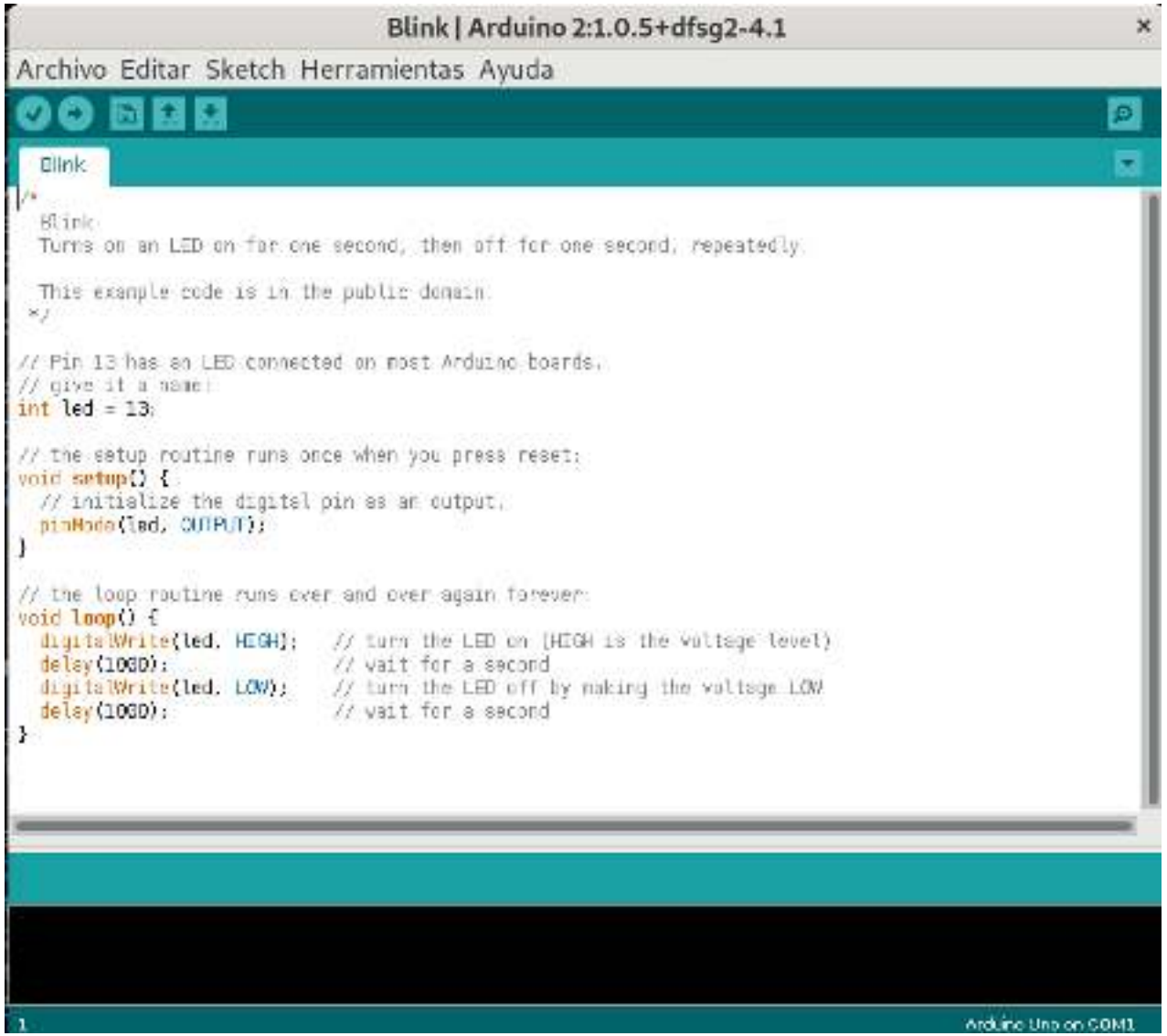


Desarrollo en Arduino

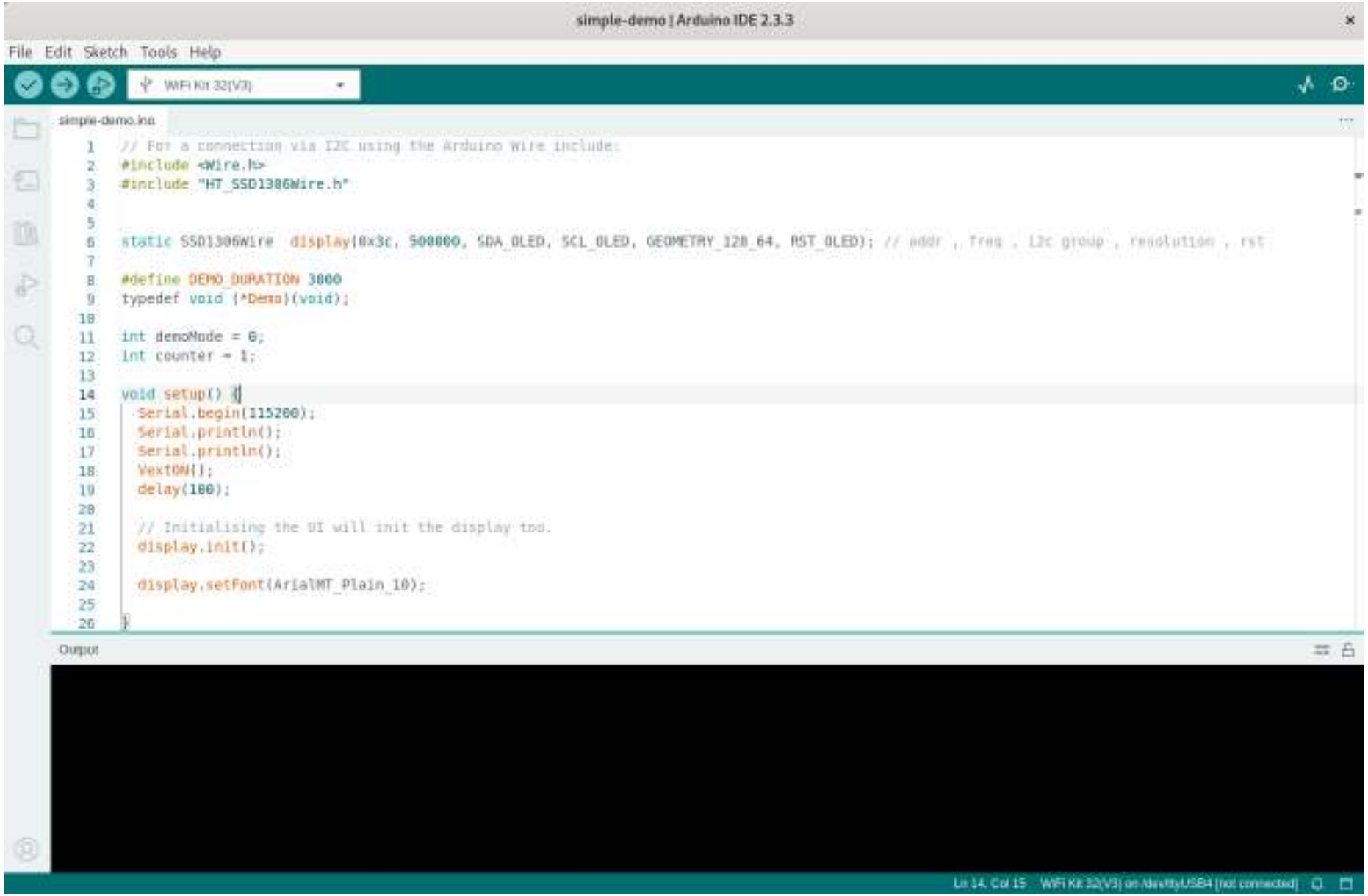
- Para desarrollar en **Arduino** utilizaremos el IDE arduino que a simple vista puede parecer un tipo de C/C++
- Arduino es la conjunción de Processing y Wiring
  - **Processing**: Es un lenguaje de programación y entorno de desarrollo basado en Java, de código abierto y bajo una licencia GNU GPL. Pensado para no programadores, para diseñadores audiovisuales que quieran crear proyectos multimedia
  - **Wiring**: Plataforma de prototipado electrónico enfocada a diseñadores y artistas.
- Los programas escritos con el IDE de Arduino Software, se denominan **sketches** y tienen extensión .ino



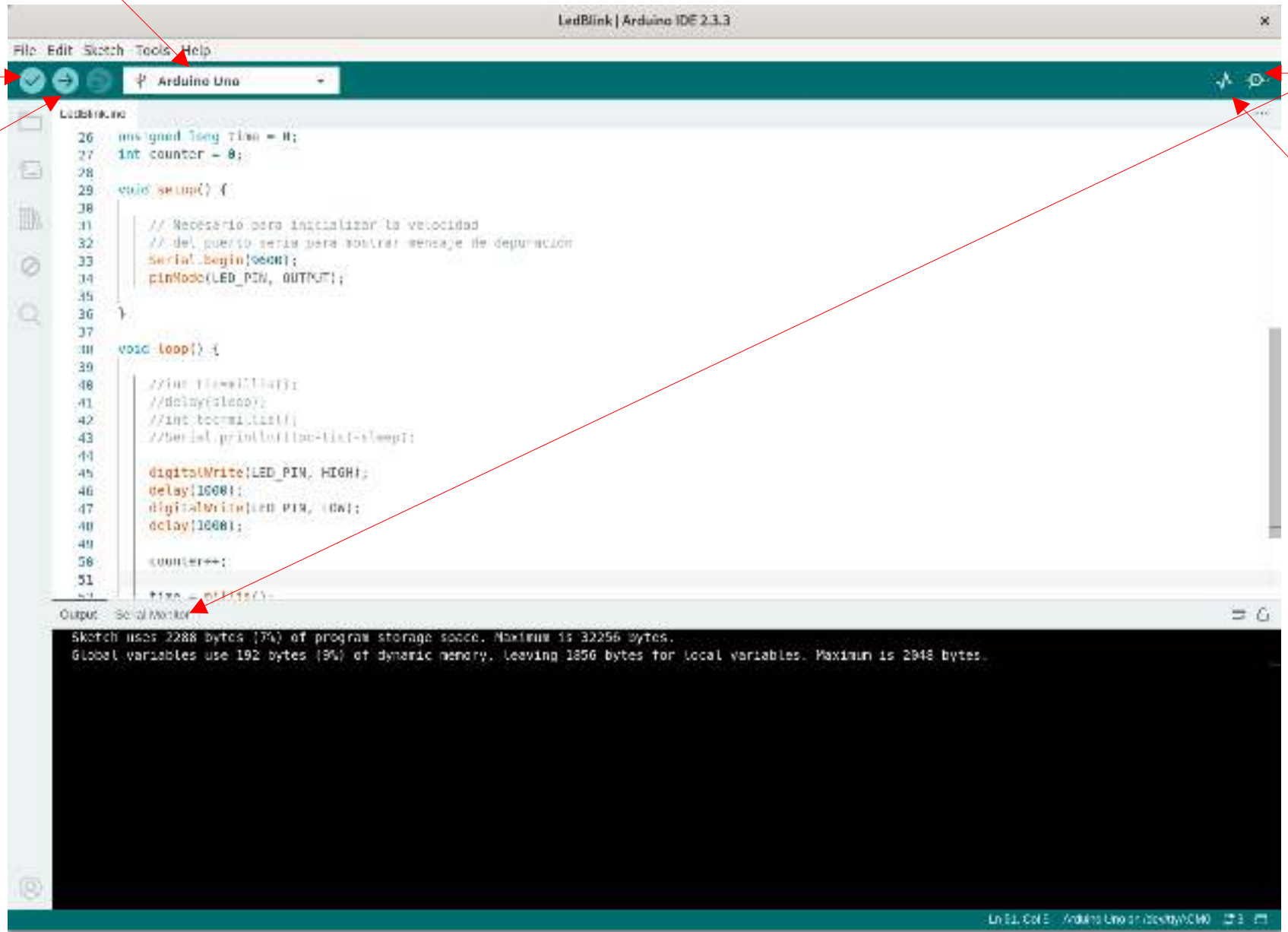
Arduino Software IDE



Arduino IDE 2.3.3



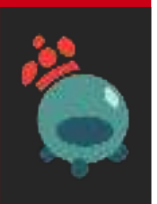
Arduino IDE 2.3.3



Esqueleto del sketch

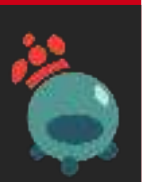
- Todo sketch tendrá 2 funciones obligatorias:
  - ***setup()***: Función que se utiliza para inicializar datos o puertos, y que se ejecuta 1 única vez al inicio del programa.
  - ***loop()***: El contenido de esta función ejecuta repetidamente mientras la placa arduino siga encendida.


```
// the setup routine runs once when you press reset:
void setup() {

}

// the loop routine runs over and over again forever:
void loop() {

}
```



Librerías Oficiales

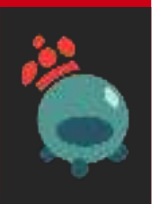
- Arduino ofrece librerías oficiales de soporte:
  - ArduinoTestSuite
  - EEPROM
  - SD
  - Ethernet
  - Firmata
  - LiquidCrystal
  - Servo
  - Stepper
  - SPI
  - Wire
  - SoftwareSerial



Ejemplo: Parpadeo de un LED

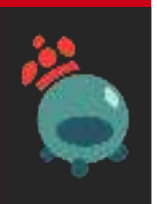
- Parpadeo intermitente de un LED conectado al pin 13.

```
25  int LED_PIN=13
26
27  // the setup function runs once when you press reset or power the board
28  void setup() {
29    // initialize digital pin LED_BUILTIN as an output.
30    pinMode(LED_PIN, OUTPUT);
31  }
32
33  // the loop function runs over and over again forever
34  void loop() {
35    digitalWrite(LED_PIN, HIGH);       // turn the LED on (HIGH is the voltage level)
36    delay(1000);                       // wait for a second
37    digitalWrite(LED_PIN, LOW);        // turn the LED off by making the voltage LOW
38    delay(1000);                       // wait for a second
39  }
40
```



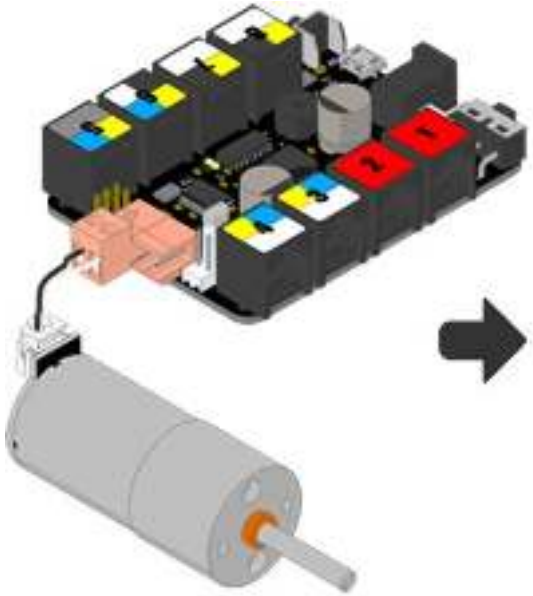
Bootloader y Firmware

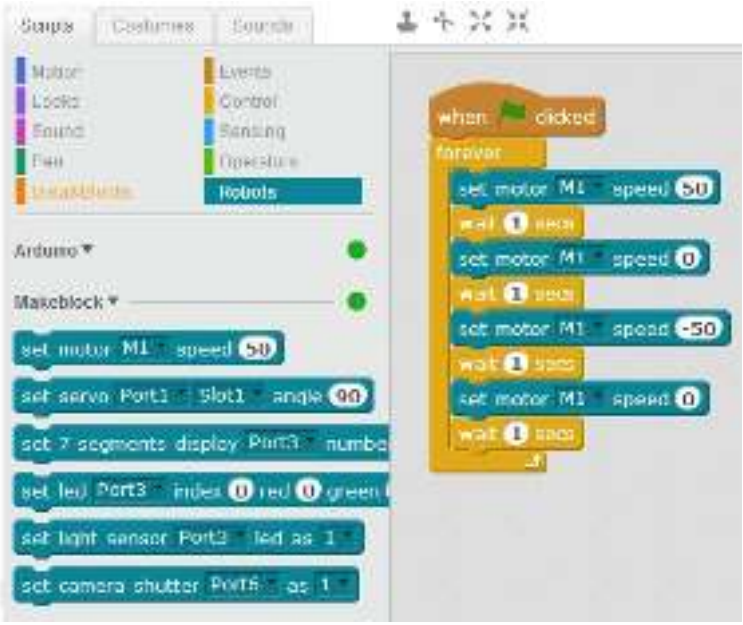
- **Bootloader** es un código que reside en un espacio reservado de la memoria del MCU de Arduino.
- Cuando arranca la placa Arduino, ejecuta el bootloader por un tiempo determinado de tiempo.
- Si durante ese tiempo, la UART del MCU de Arduino recibe el comando de “reprogramación” desde el IDE, entonces graba el programa recibido en la memoria de programa.
- Si no recibe ningún comando, entonces la placa Arduino arranca con el sketch alojado en la memoria de programa.
- 2 consideraciones importantes:
  - Bootloader ocupa espacio en memoria: 2 KB
  - El inicio de tu sketch siempre tendrá un retraso de unos segundos.



Programación Visual

- Scratch (S4A): https://scratch.mit.edu/
- Ardublock: http://blog.ardublock.com/
- Visualino: http://www.visualino.net/
- mBlock: http://www.mblock.cc/





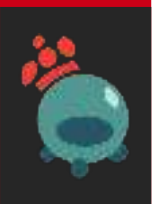
Simulador Arduino

- Programa gratuito de modelado 3D y circuitos electrónicos.
- Editor y emulador de Arduino
- https://www.tinkercad.com/

AUTODESK®
TINKERCAD®

Bibliografía

- [Libro] Arduino Cookbook, 3rd Edition, Abril 2020
  - Michael Margolis, Brian Jepson, Nicholas Robert Weldin
- [Libro] Exploring Arduino, 2nd Edition, Septiembre 2019
  - Jeremy Blum
- Arduino: El Documental
  - https://www.youtube.com/watch?v=mltWc9_C9gs
- Getting started with Arduino productos
  - https://www.arduino.cc/en/Guide
- A Cross Platform and Open Source Electronics Design Automation Suite
  - https://kicad.org/



URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC Programming Robot Intelligence

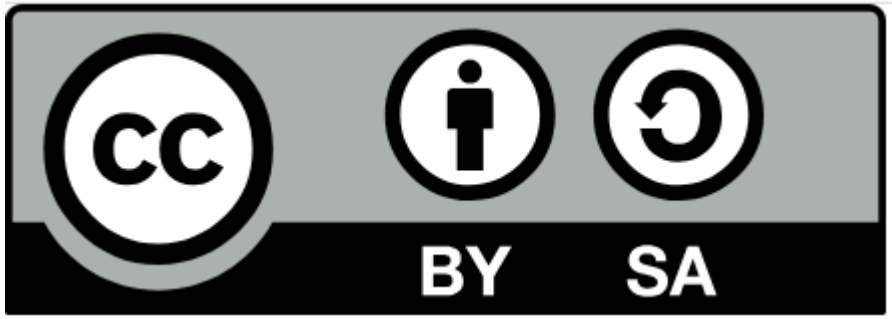
Sistemas Empotrados y de Tiempo Real

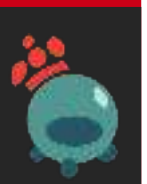
Desarrollo con Arduino MultiTasking

Grado en Ingeniería de Robótica Software

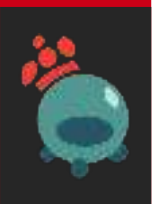
Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es

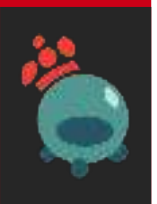




MultiTasking

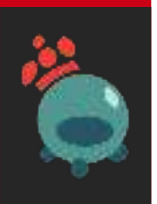


MultiTasking



Sketch

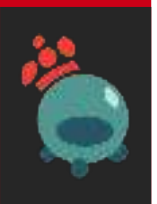
- Los sketch en Arduino son simples y fáciles de utilizar al principio cuando los ejemplos son muy básicos
  - Enceder/apagar un led.
  - Leer un pulsador y encender un led
- Pero la complejidad aumenta cuando el número de tareas a realizar es alta y cada una necesita una periodicidad diferente.
  - Led intermitente cada 500 ms
  - Leer entrada digital pulsador cada 170 ms
  - Leer sensor temperatura cada segundo
  - ….



Scketch

- Posible implementación con millis()

```
void loop() {

  /* Updates frequently */
  unsigned long currentTime = millis();

  /* This is the event */
  if (currentTime - previousTime >= eventInterval1) {
    /* Event code */
    Serial.println("Task1");
   /* Update the timing for the next time around */
    previousTime1 = currentTime;
  }
  if (currentTime - previousTime >= eventInterval2) {
    /* Event code */
    Serial.println("Task2");
   /* Update the timing for the next time around */
    previousTime2 = currentTime;
  }
  if (currentTime - previousTime >= eventInterval3) {
    /* Event code */
    Serial.println("Task3");
   /* Update the timing for the next time around */
    previousTime3 = currentTime;
  }
}
```



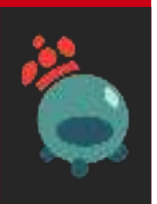
Scketch

- Posible implementación con millis()

```
void l

  /* Up                    ntly */
  unsig                   entTime

  /* This                    */
  if (curr                          ventInterval
    /* Even
    Serial.p
   /* Update                      ext time around
    previousTi
  }
  if (currentTi                     >= eventInterval
    /* Event co
    Serial.pri
   /* Update                       round */
    previou
  }
  if (cu                 previousTi
    /*                  */
    Se                  tln("Task3");
   /*                the timing for the next time around */
                       usTime3 = currentTime;
  }
}
```

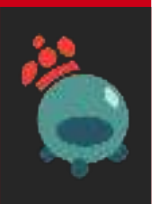
ArduinoThread

- Arduino incorpora una librería que nos permite ejecutar “threads” en Arduino.
- Compatible con todas las arquitecturas Arduino
- No ejecuta threads en paralelo
- Basado en ProtoThreads
- Planificar y manejar fácilmente tareas periódicas
- Definir tiempos fijos o variables entre ejecuciones
- Organizar el código limpiamente:
  - Todas las lecturas de sensores en un thread.
  - Mantiene el bucle principal limpio.
- Encapsula la complejidad de cada thread.



ArduinoThread

- Documentación:
  - https://www.arduino.cc/reference/en/libraries/arduinothread/
  - https://github.com/ivanseidel/ArduinoThread



ArduinoThread

- Añadir las librerías necesarias.

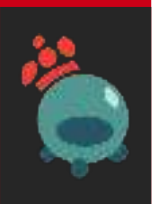
```
#include <Thread.h>
#include <StaticThreadController.h>
#include <ThreadController.h>
```

- Creamos un thread asignándole una periodicidad y callback

```
  Thread myThread = Thread();
  myThread.enabled = true;
  myThread.setInterval(300);
  myThread.onRun(callback_thread1);
```

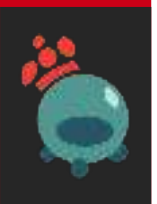
- El callback puede implementar cualquier código

```
void callback_thread1() {
  Serial.println("Hello! " + String(millis()));
}
```



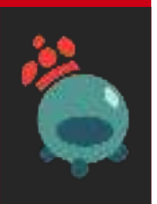
ArduinoThread

```ino
#include <Thread.h>

void callback_thread1() {
  Serial.println("Hello! " + String(millis()));
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  delay(100);

  myThread.enabled = true;
  myThread.setInterval(300);
  myThread.onRun(callback_thread1);

}

void loop() {

  // Check si podemos ejecutar el thread
  if(myThread.shouldRun()){
    myThread.run();
  }
}
```



ArduinoThread: Controller

- ¿Y qué ocurre si tenemos 20 threads distintos?
- Podemos usar el controlador de ArduinoThread
- El controlador permite manejar, activar y ejecutar numerosos threads al mismo tiempo.



ArduinoThread: Controller

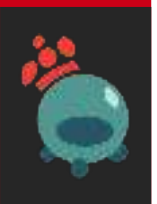
```
#include <Thread.h>
#include <StaticThreadController.h>
#include <ThreadController.h>

ThreadController controller = ThreadController();
Thread myThread = Thread();


void callback_thread1() {
  Serial.println("Hello! " + String(millis()));
}


void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  delay(100);

  myThread.enabled = true;
  myThread.setInterval(300);
  myThread.onRun(callback_thread1);

  controller.add(&myThread);
}

void loop() {


  controller.run();

}
```

Añadimos las librerías

Instanciamos el controlador

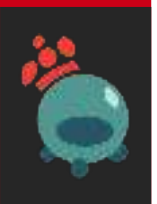
Añadimos el thread al controlador

El bucle principal solo llama al controlador en cada iteración



ArduinoThread

¿Cómo generamos un thread que permita parpadear un LED sin que utilice más recursos de los necesarios utilizando ArduinoThread?



ArduinoThread

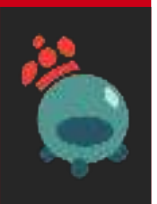
```
class LedThread: public Thread {

public:
  int pin;
  bool state;

  LedThread(int _pin): Thread() {
    pin = _pin;
    state = true;

    pinMode(pin, OUTPUT);
  }

  bool shouldRun(unsigned long time){
    return Thread::shouldRun(time);
  }

  void run(){

    Thread::run();

    digitalWrite(pin, state ? HIGH : LOW);
    state = !state;

  }
};
```

Heredamos de Thread

Constructor recibe el pin del LED

shouldRun es llamado siempre antes de run(). Solo ejecutará run() si esta función devuelve TRUE

Run(), ejecuta el código deseado de este Thread. Siempre hay que llamar al padre Thread::run()



ArduinoThread

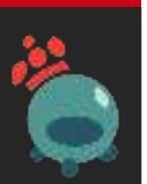

```
#define PIN_LED 8

ThreadController controller = ThreadController();

void setup() {

  Serial.begin(9600);
  delay(100);

  LedThread* ledThread = new LedThread(LED_BUILTIN);
  ledThread->setInterval(1000);
  controller.add(ledThread);

  LedThread* ledThread2 = new LedThread(PIN_LED);
  ledThread2->setInterval(2000);
  controller.add(ledThread2);

}
```

Primer objeto LedThread con intervalo de 1 segundo y LED_BUILTIN

Segundo objeto LedThread con intervalo de 2 segundos y PIN_LED = 8

ArduinoThread

- Primer aproximación a multi-tarea en Arduino mono-core
- Nos permite programar y organizar la funcionalidad de nuestro sketch en diferentes hilos/threads de ejecución.
- Si un thread se queda bloqueado, todo el programa se queda bloqueado.
- Nos permite programar en Arduino sin necesidad de delays.



URJC Escuela Técnica Superior de Ingeniería de Telecomunicación

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC Programming Robot Intelligence

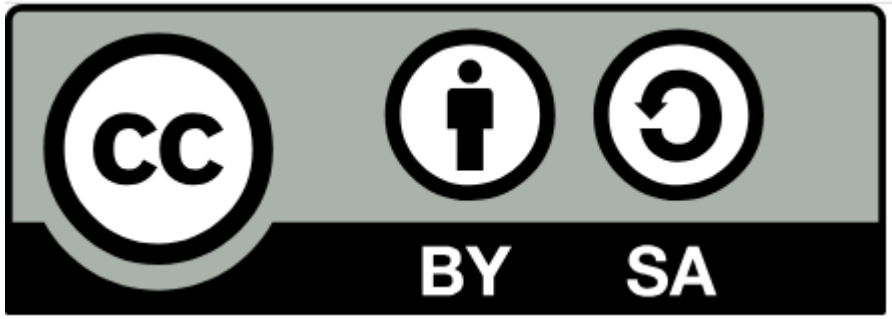
Sistemas Empotrados y de Tiempo Real

Desarrollo con Arduino
Tiempo y RTC

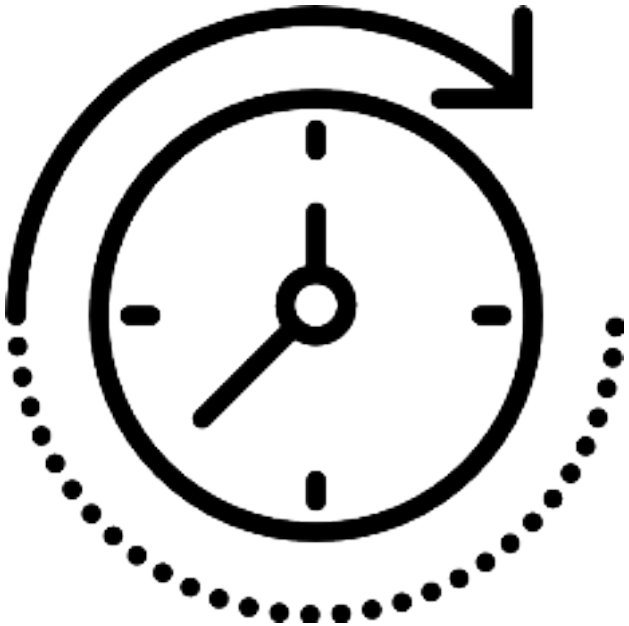
Grado en Ingeniería de Robótica Software

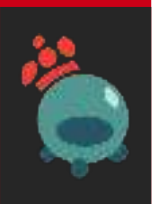
Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es



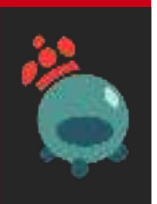
Tiempo





Relojes

- La mayoría de las placas Arduino no incorporan ningún componente que nos proporcione hora y fecha en UTC.
- Recordad que *milis()* y *micros()* devuelven contadores que se inicializan a 0 cuando arranca el sketch.
- Para obtener y almacenar hora y fecha en nuestros proyectos podemos usar:
  - RTC (Real Time Clock)
  - NTP
  - RTC + NTP



Real Time Clock (RTC)

- Normalmente son circuitos integrados que mantienen la hora y fecha actual aunque el sistema esté apagado, ya que incluyen una pequeña batería.
- No confundir con Real Time Computing (RTC)
- Bajo consumo energético.
- Más preciso que otros mecanismos.

- La mayoría de los RTC utilizan un oscilador de cristal a una frecuencia de 32.768 kHz
- $32768 = 2^{15}$, es un ratio muy práctico para usar con circuitos de contadores binarios simples.

DS3231 RTC

- Oscilador de 32 kHz
- Comunicación I2C
- Cuenta segundos, minutos, horas y años
- Exactitud (accuracy)
  - +2ppm / -2ppm → 0°C , +40°C
  - 3.5ppm / -3.5ppm -> -40°C , +85°C

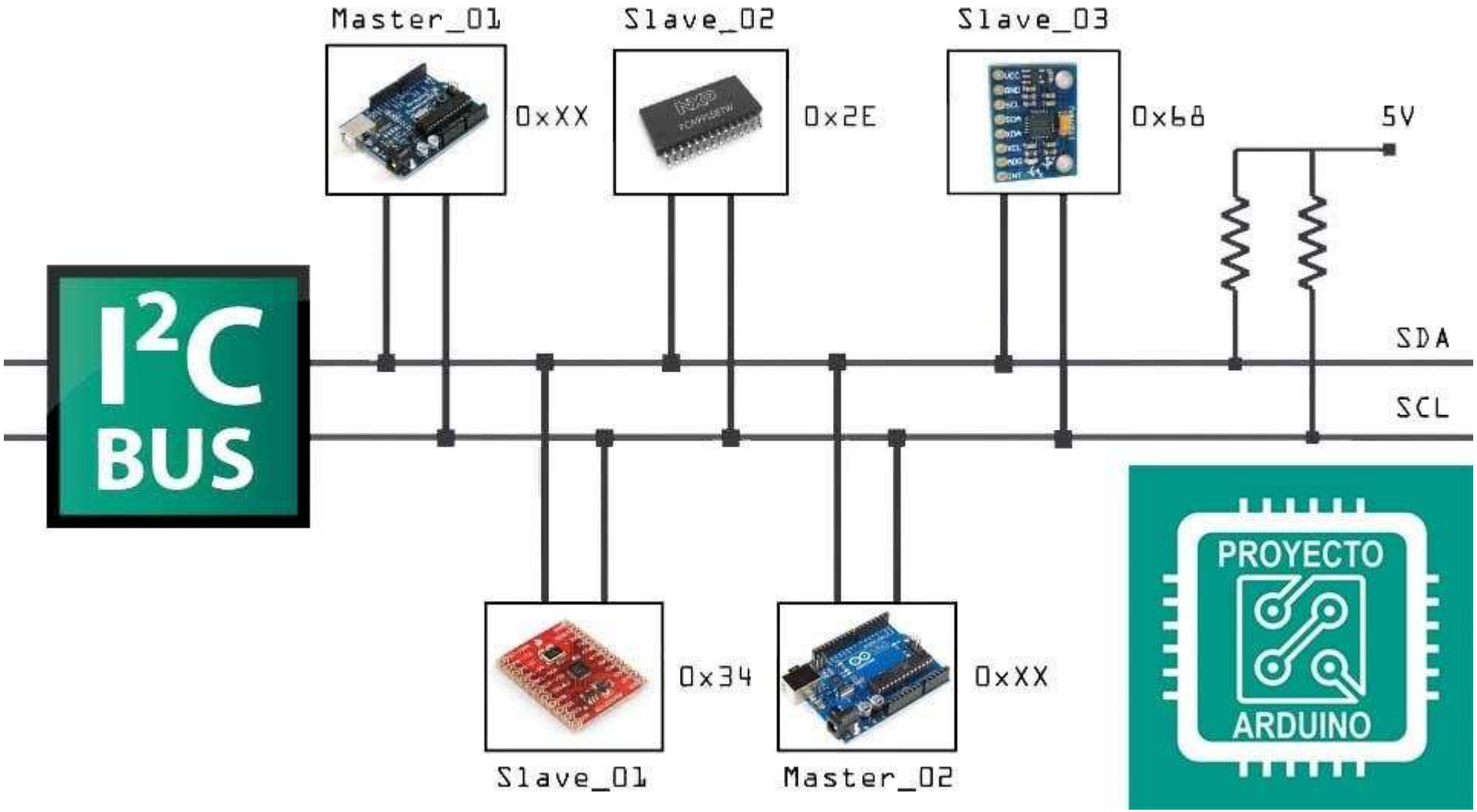
DS1307 RTC

- Oscilador de 32 kHz
- Comunicación I2C
- ~ 500 nA operado con bateria (5-10 años)
- Exactitud (accuracy)
  - 15/30 ppm
  - Alrededor de un par de segundos al dia.

I2C

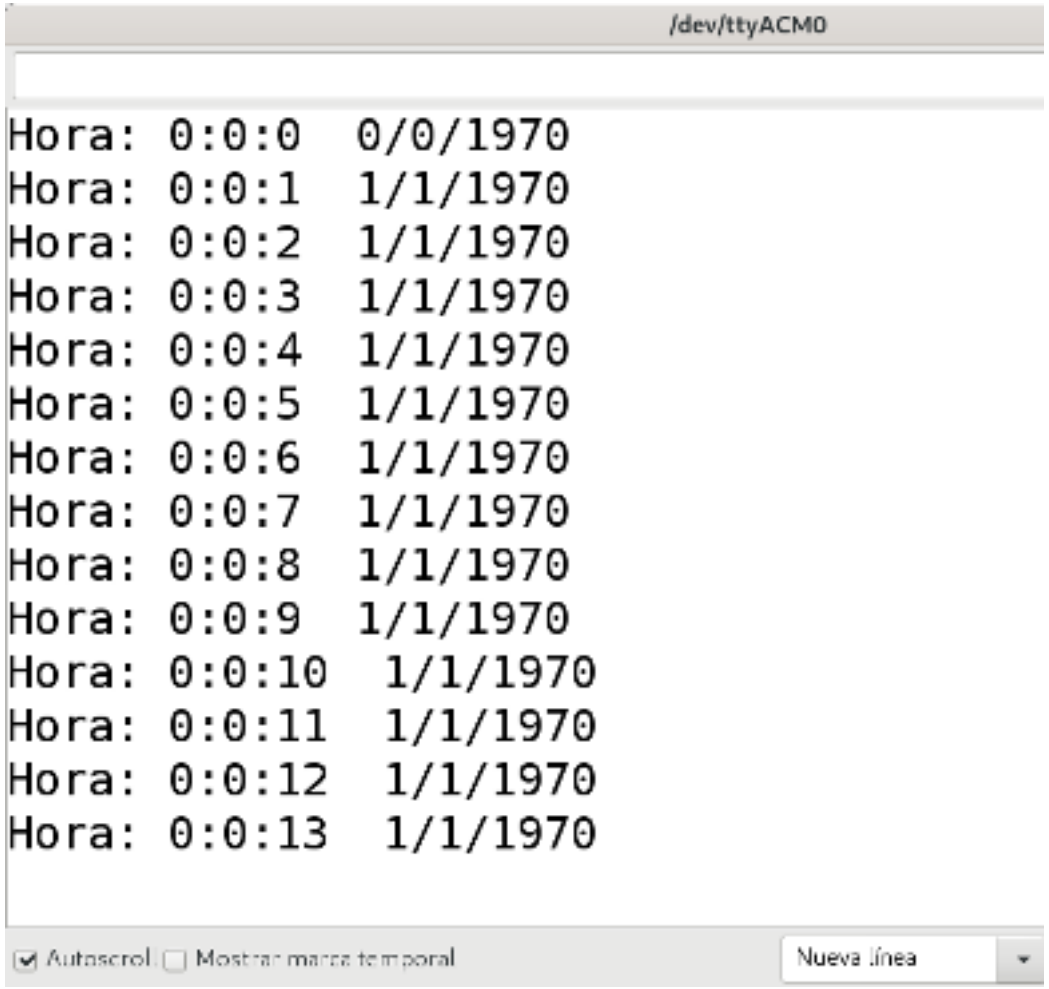
- Protocolo de comunicación serial desarrollado por Philips
- Lider-seguidor (maestro-esclavo)
- Es un protocolo síncrono
- Velocidades: 100 kbps – 5 Mbps
- Numero ilimitado de lideres y un máximo de 1008 seguidores.
- I2C utiliza 2 vías de comunicación:
  - SDA (Serial Data). Canal de intercambio de información.
  - SCL (Serial Clock). Canal donde viaja la señal de reloj.

I2C


Master_01
0xXX
Slave_02
0x2E
Slave_03
0x68
5V
I²C BUS
SDA
SCL
0x34
Slave_01
0xXX
Master_02
PROYECTO
ARDUINO

Hora y Fecha

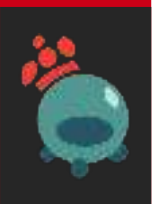
```
#include <Time.h>
#include <TimeLib.h>

void setup() {
  Serial.begin(9600);
}

void loop() {

  Serial.print("Hora: ");
  Serial.print(hour());
  Serial.print(":");
  Serial.print(minute());
  Serial.print(":");
  Serial.print(second());

  Serial.print("  ");
  Serial.print(day()); Serial.print("/");
  Serial.print(month()); Serial.print("/");
  Serial.println(year());

  delay(1000);
}
```



Hora y Fecha

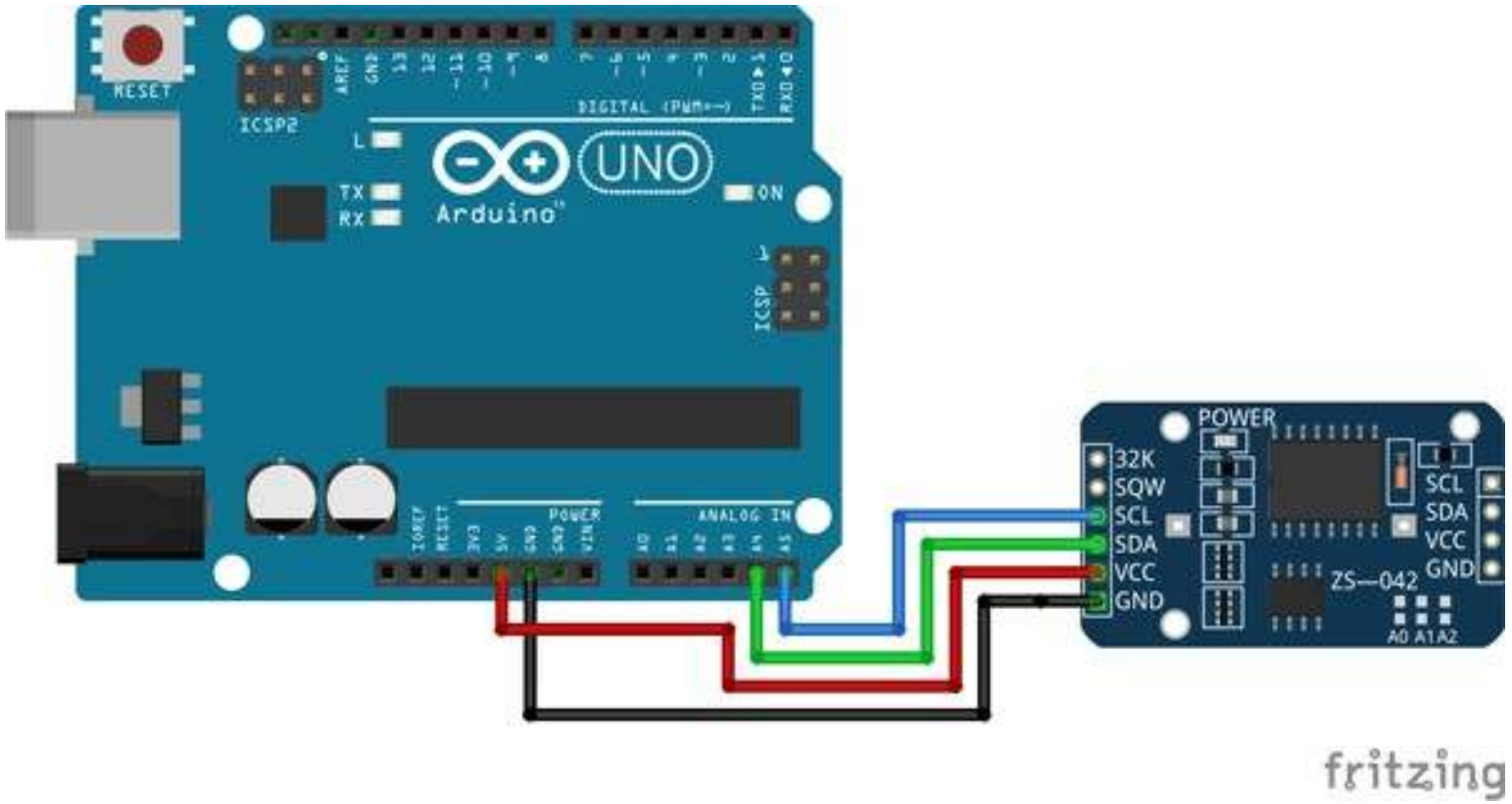
- Para establecer una nueva fecha y hora:

```
#include <Time.h>
#include <TimeLib.h>

void setup() {

  Serial.begin(9600);

  setTime(10, 58, 0, 24, 11, 2020);

}
```



Añadir RTC

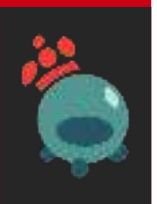
- Bus I2C
- Arduino UNO -> SDA = A4 y SCL = A5



Añadir RTC

```
#include <Wire.h>
#include <RTClib.h>
#include <Time.h>
#include <TimeLib.h>

// Declaramos un RTC DS3231
RTC_DS3231 rtc;

void setup() {
  Serial.begin(9600);

  delay(3000);

  // Comprobamos si tenemos el RTC conectado
  if (! rtc.begin())
    Serial.println("No hay un módulo RTC!");

  // Ajustamos la hora y fecha con la compilación del
scketch
  // Solo ejecutamos la primera vez este código.
  //rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
}
```

```
void loop() {

  DateTime now = rtc.now();

  Serial.print("Hora: ");
  Serial.print(now.hour());
  Serial.print(":");
  Serial.print(now.minute());
  Serial.print(":");
  Serial.print(now.second());

  Serial.print("  ");
  Serial.print(now.day()); Serial.print("/");
  Serial.print(now.month()); Serial.print("/");
  Serial.println(now.year());

  delay(1000);
}
```



URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

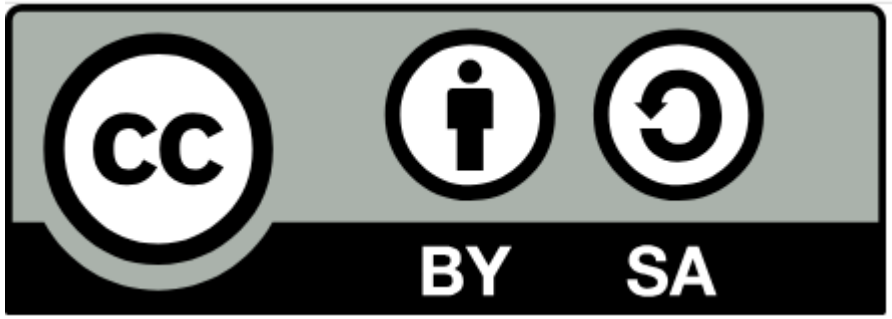
GSyC RoboticsLabURJC Programming Robot Intelligence

Sistemas Empotrados y de Tiempo Real
Interrupciones

Grado en Ingeniería de Robótica Software

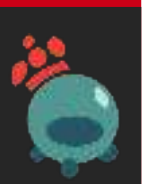
Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es



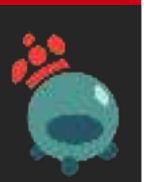
Polling

- El **polling** es una técnica antigua por la cual se sondea un estado de manera continua y constante.
- Técnica ineficiente:
  - El microcontrolador gasta ciclos sondeando, cuando la mayoría de las veces no es necesario ya que no hay datos disponibles.
  - Se pueden perder eventos/lecturas.
  - No existe prioridad.
  - Consume más energía.
  - No escala correctamente con numerosas entradas digitales.



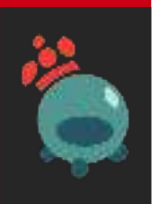
Interrupciones

- Las interrupciones son **mecanismos** del microcontrolador para responder a **eventos**.
- Suspende **temporalmente** el flujo del programa principal.
- Ejecuta una subrutina de servicio de interrupción (ISR)
- Una vez terminada dicha subrutina, se **reanuda** la ejecución del programa principal.
- Las **interrupciones** son esenciales en **sistemas empotrados**, ya que le permite atender procesos del mundo exterior sin descuidar la ejecución del programa principal.
- Ejemplos: adquisición de datos, monitoreo de sensores, cálculos numéricos, envío de comandos al robot, etc.

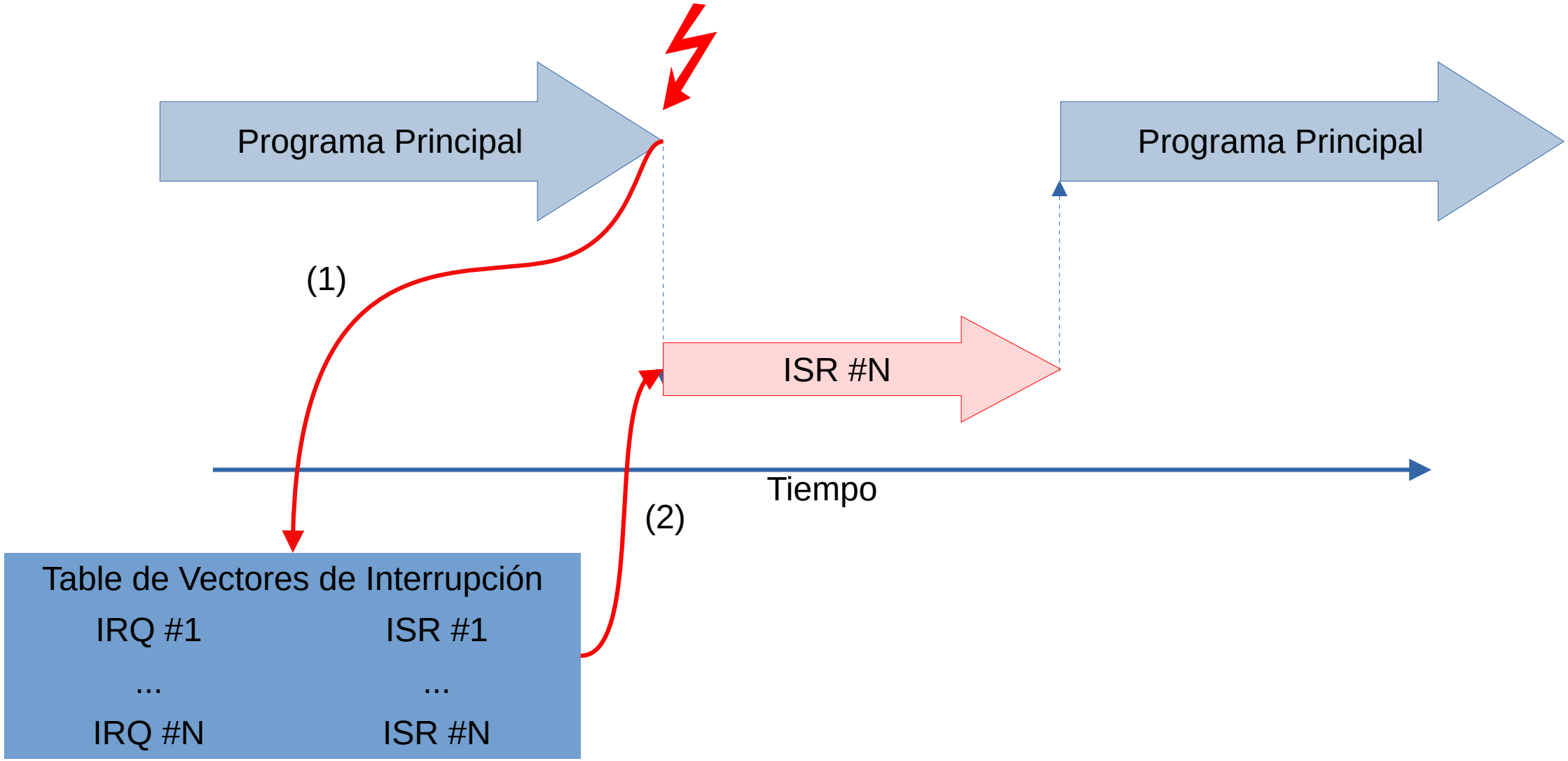


¿Cómo funciona una interrupción?

- Cuando se genera una interrupción:
  - La unidad de interrupciones indica al microcontrolador que un evento **requiere** solicitud de interrupción.
  - El microcontrolador determina qué **tipo** de interrupción se está generando.
  - El microcontrolador salva el **contador de programa** y contexto.
  - Cada tipo de interrupción tiene una **ISR** asignada que se guarda en la tabla de vectores de interrupción.
  - Se ejecuta el código de la ISR.
  - Se restaura el contexto y se devuelve el control al programa que estaba en ejecución.

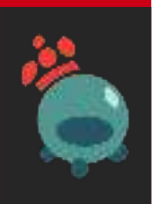


¿Cómo funciona una interrupción?



Tipos de Interrupciones

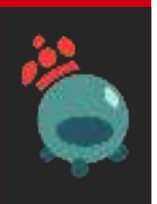
- En los microcontroladores Atmega nos encontrados con interrupciones hardware y software.
- Interrupciones **hardware** nos ayudan a determinar cambios en los pines físicos de entrada.
  - *Arduino UNO* dispone de 2 pines para interrupciones por hardware:
    - Int 0: Pin 2
    - Int 1: Pin 3
  - Arduino MEGA dispone de 6 pines.
- Utilizados para sensores y lecturas del entorno exterior.



Tipos de Interrupciones

- Interrupciones **software** se generan a través de timers o contadores. Son generadas cuando pasa un tiempo determinado.
- Arduino UNO dispone de:
  - 2 Timers de 8 bits (250 kHz): Timer0, Timer2
  - 1 Timer de 16 bits (16 MHz): Timer1

- Timer0 es usado para *millis() y delay()*, por lo que el uso de este timer hará que la función millis() no funcione correctamente.

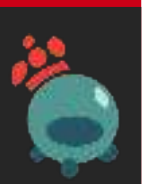


Interrupciones ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B

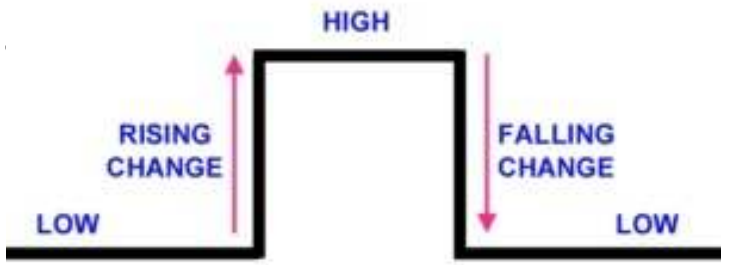
Interrupciones ATmega328P

14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready



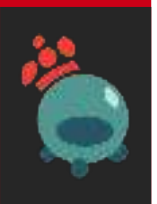
Interrupciones Hardware

- Las interrupciones hardware puede ser configuradas para detectar estados lógicos (voltaje) o transición de pulsos.
  - ***LOW***: La interrupción salta cuando el pin está en estado LOW
  - ***HIGH***: La interrupción salta cuando el pin está en estado HIGH (sólo disponible en Arduino DUE).
  - ***CHANGE***: La interrupción salta cuando el pin cambia de estado, ya sea LOW a HIGH, o viceversa.
  - ***RISING***: La interrupción salta cuando el pin pasa de LOW a HIGH.
  - ***FALLING***: Esta es la inversa de la de arrib cuando el pin pasa de HIGH a LOW



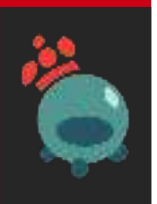
ISR

- **Interrupt Service Routine**
- Deben ejecutar el mínimo tiempo posible.
- Preferiblemente código sencillo y simple.
- Solo puede ejecutar una ISR a la vez, en caso de varias, se ejecutan **secuencialmente**.
- Normalmente la función de la ISR se limita a activar un flag, incrementar un contador, o modificar una variable.
- No añadir en una ISR:
  - Cálculos complejos
  - Comunicación (serial, I2C y SPI)
- Siempre que sea posible, no realizar cambios lógicos en entradas salidas/digitales.



Efecto de la interrupción

- Arduino utiliza interrupciones para realizar el conteo del tiempo mediante *millis()* y *micros()*
- Efectos dentro de la ISR:
  - Durante la ejecución de la ISR, arduino no actualiza los valores de las funciones *millis()*, y *micros()* funcionaría erróneamente después de 1-2 ms.
  - No se contabiliza el tiempo de ejecución de la ISR.
- Efectos fuera de la ISR:
  - *millis()* no actualiza su valor, y *delay()* no podría funcionar correctamente.
- El uso de interrupciones provee de funcionalidad muy potente pero hay que ser cuidadosos con su uso.

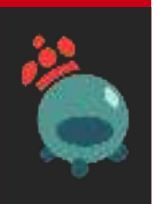


Interrupciones Hardware

- Las interrupciones hardware las activaremos usando la siguiente llamada:

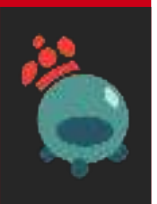
```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

  - Pin: Es el pin físico asociado a la interrupción
  - ISR: callback a la función de tratamiento de interrupción
  - Mode: Estado lógico para activar la interrupcción
    - HIGH, LOW
    - CHANGE
    - RISING, FALLING



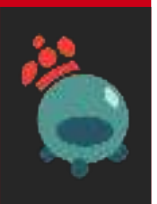
Interrupciones Hardware

- Otras funciones interesantes para la gestión de interrupciones son:
  - Anular interrupción
    - `detachInterrupt(interrupt)`
  - Desactivar ejecución de interrupciones
    - `noInterrupts()`
  - Reactivar Interrupciones
    - `interrupts()`



Volatile

- Directiva del compilador asociada a definición de **variables**.
- Fuerza al compilador a cargar esa variable siempre desde memoria **RAM** y en una operación atómica (y no desde uno de los registros internos, cache, optimizaciones, etc).
- La variable tiene que ser **cargada siempre** antes de su lectura, ya que ha podido ser modificada de forma ajena al flujo principal del programa.
- En arduino, este caso solo se da en **interrupciones**.
  - Una interrupción puede cambiar el estado de una variable que luego se usará en el loop() principal.
- El uso de volatile **ralentiza** la ejecución. Solo usar en situaciones donde sea realmente necesario.



Interrupciones Hardware

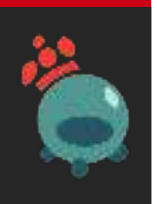
- Enceder/Apagar un LED a través de interrupciones (botón)

```
const int INT_PIN = 2;

volatile byte state = LOW;

void blink() {
  state = !state;
  digitalWrite(LED_BUILTIN, state);
}

void setup() {
  Serial.begin(9600);

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(INT_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(INT_PIN), blink, CHANGE);
}

void loop() {
  delay(10);
  Serial.println(state);
}
```

Una variable debe ser definida como ‘volatile’ si es modificada en una ISR y queremos que se actualice el valor de esa variable en otras funciones.

Cambiamos el estado de la salida digital en la ISR

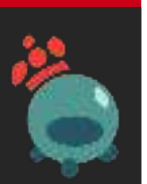
Asociamos la interrupción del pin #2 a la ISR definida por blink(), para que se ejecute cuando cambie el estado de la señal lógica (CHANGE)



Interrupciones por Hardware

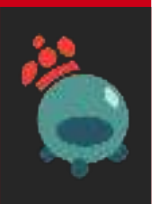

```
const int INT_PIN = 2;

volatile byte state = LOW;
long startTime = 0;
const int timeThreshold = 200;

void blink() {
  if (millis() - startTime > timeThreshold) {
    startTime = millis();
    Serial.println("Interruption: "
        + String(millis()));
    state = !state;
    digitalWrite(LED_BUILTIN, state);
  }
}
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(INT_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(INT_PIN), blink, CHANGE);

}

void loop() {
  // put your main code here, to run repeatedly:
  delay(100);
  Serial.println(state);
}
```



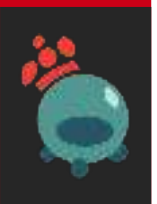
Interrupciones por Software

- Las interrupciones por **software** son sensibles a la precisión y exactitud del reloj interno del sistema.
- Las interrupciones software basadas en timers utilizan la misma idea de una **alarma** en un despertador.
- El contador del timer se va incrementando hasta que finaliza el conteo y salta la interrupción.
- Son muy utilizadas para implementar sistemas **multitasking** en entornos mono-core.



Interrupciones por Software

- Los Timers en Arduino permiten una configuración a bajo nivel escribiendo en los registros apropiados.
- Utilizaremos la librería *TimerOne* para interactuar con el timer1 de Arduino
  - https://www.arduino.cc/reference/en/libraries/timerone/
- Compatible con la arquitectura AVR, por lo que soporta:
  - Arduino Micro
  - Arduino Leonardo
  - Arduino Mega
  - Arduino Nano
  - Arduino Uno
  - Arduino Yún



Interrupciones por Software

- Encender/Apagar un LED

```
#include <TimerOne.h>

int ledstate = LOW;

void blinkLED() {

  digitalWrite(LED_BUILTIN, ledstate);
  ledstate = !ledstate;

}

void setup() {

  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  Timer1.initialize(500000);
  Timer1.attachInterrupt(blinkLED);

  Serial.begin(9600);
}

void loop() {

  delay(100);
}
```

Incluimos la librería TimerOne

Definimos la ISR que tratará la interrupción por software

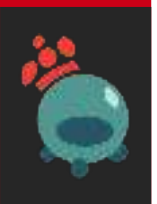
Inicializamos el timer con el tiempo periódico en el cual se lanzará una interrupción software.

Asignamos la ISR al Timer1

Conflicto TIMERS – Arduino UNO

- Timer0 → millis() y delay()
- Counter0/Timer0 → PWM de los pines 2,5,6
- Es **seguro** utilizar Timer0 para PWM, millis() y delay() a la vez mientras no se cambie la frecuencia de trabajo de Timer0.

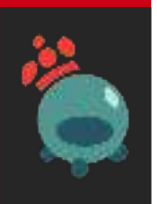
- Counter1/Timer1 → PWM de los pines 9, 10, 11
- Librería TimerOne hace uso de Counter/Timer1.
- Usar a la vez TimerOne y PWM (9,10,11) podría llevar a comportamientos nos deseados.



Evitar interrupciones anidadas

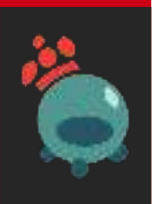
- Es posible que recibamos más interrupciones de las que podemos/queremos tratar.
- Eso puede llevar a un mal funcionamiento del sistema.
- Es posible inhabilitar las interrupciones mientras estamos ejecutando la ISR.

```
void ISR() {
  noInterrupts()
  ... //código que atiende la interrupción
  interrupts()
}
```



Más sobre interrupciones

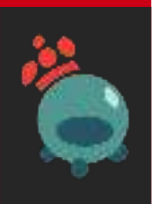
- La interrupción para *timer0* es usado para *millis()*
- Muchas librerías externas usan directamente *millis()* para generar ejecuciones periódicas, por lo que usan internamente interrupciones.
- No uses '*volatile*' si no es necesario, ralentiza la ejecución ya que el microcontrolador tiene que copiar valores de variables constantemente
- Es posible configurar la frecuencia del timer, eso es, la rapidez con la que cuenta. A una *f=100Hz*, tenemos un T=*1/100*, que son 10 ms.
  - Un contador de 8bits [0-255], generaría una interrupción cada 255*10 ms = 2.55 segundos.



Más sobre interrupciones

Si arduino dispone de un reloj de 16 Mhz …

¿Por qué Timer0, que es utilizado para el calculo de millis() y micros(), me da una resolución de 4 microsegundos?



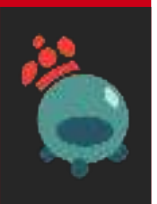
Más sobre interrupciones

- Timer0, usado por millis, recuerda que es un temporizador de 8 bits (no es un reloj). El reloj del sistema en Arduino UNO es de 16 MHz (1 tick cada ~ 62 nano-segundos) .
- Cada vez que cuenta 0-255 y se da el overflow es cuando se produce la interrupción software.
- Timer0 está configurado con un **pre-escaler** de 64, lo que quiere decir → 16 MHz / 64 = 250 kHz (~4 microsegundos)
- Por tanto, cada overflow del Timer0 supondrá una interrupción cada ~1024 millisegundos.

Más sobre interrupciones

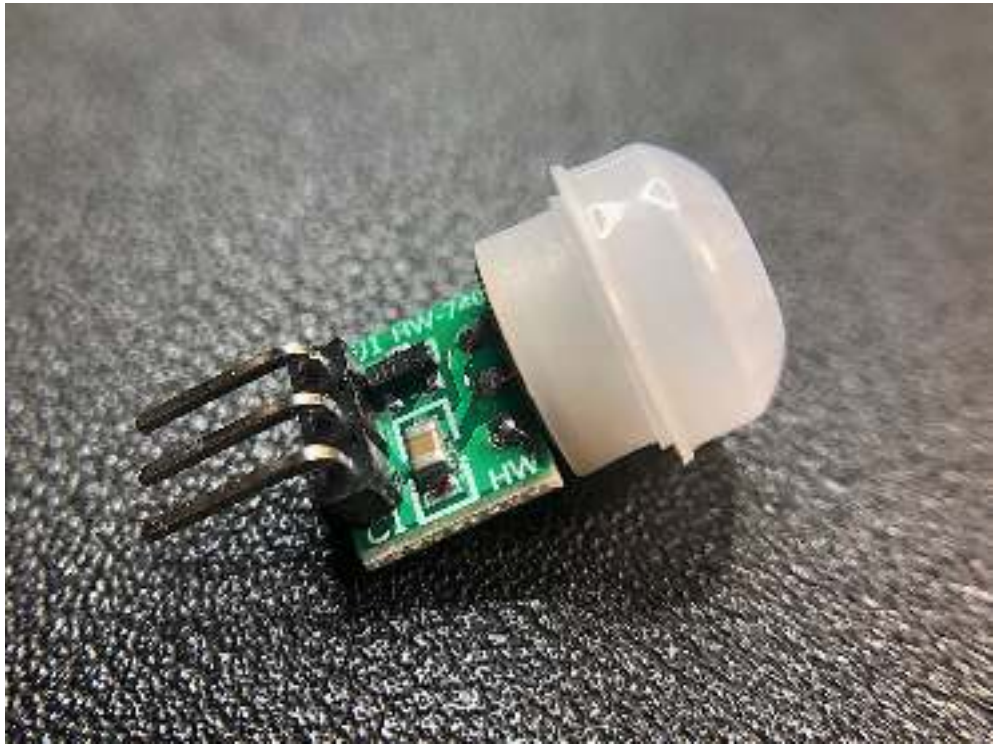
- El prescaler es una propiedad del temporizador/contador, y su función es dividir la frecuencia del reloj del sistema antes de que este llegue al temporizador, para controlar la frecuencia con la que el temporizador cuenta los "ticks"
- Es posible cambiar el factor de pre-escalado en Timer0, pero con cuidado.

```
Setting                          Prescale_factor
TCCR0B = _BV(CS00);              1
TCCR0B = _BV(CS01);              8
TCCR0B = _BV(CS00) | _BV(CS01);  64
TCCR0B = _BV(CS02);              256
TCCR0B = _BV(CS00) | _BV(CS02);  1024
```



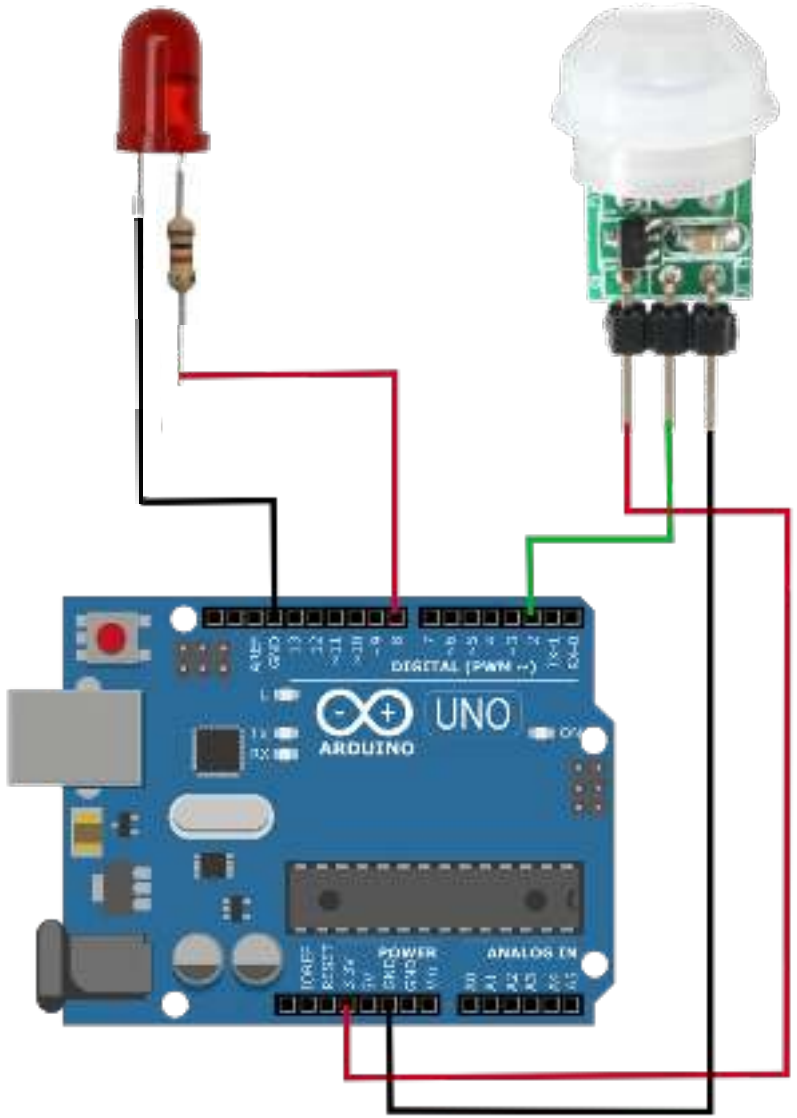
Ejemplo

- Sensor de presencia mediante interrupciones



1. VCC
2. OUT
3. GND

Ejemplo



Bibliografia

- ATmega328P (datasheet)
  - https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
- Arduino Software Internals: A Complete Guide to How Your Arduino Language and Hardware Work Together

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC Programming Robot Intelligence

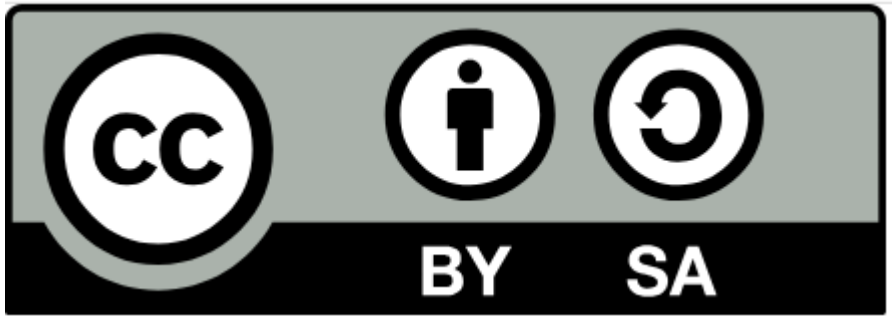
Sistemas Empotrados y de Tiempo Real

Comunicaciones en Arduino

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es



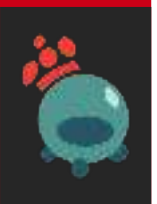
Comunicaciones

- UART
- I2C
- SPI
- CANBUS

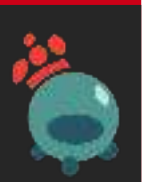
- Ethernet/WiFi
- MQTT y NodeRed

Comunicación Serie

- La **comunicación serie** en telecomunicaciones es el proceso de envío de datos de un bit a la vez, de forma secuencial, sobre un canal de comunicación o bus.
- Multitud de protocolos y sistemas
  - UART, RS-232, I2C, SPI, CAN, USB, JTAG, …
  - ¡Incluso código morse!
- Comunicación puede ser:
  - Asíncrona: No hay reloj común (UART, RS-232)
  - Síncrona: Hay reloj común para la transferencia (I2C, SPI)

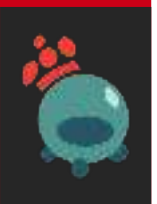


Comunicación UART

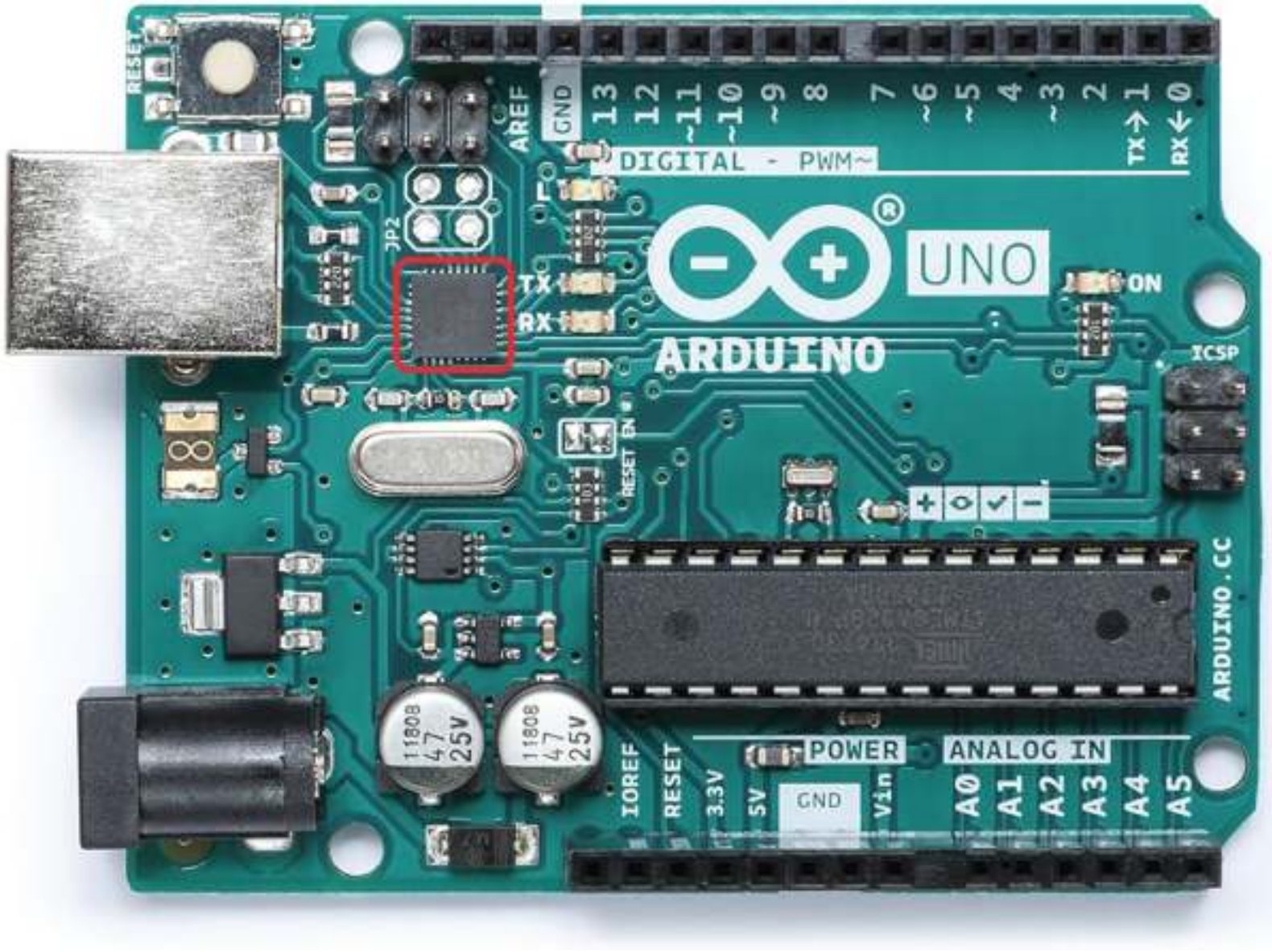


Comunicación UART

- Todas las placas Arduino contienen al menos un puerto serie para comunicarse con otro ordenador o dispositivos.
- También denominado UART:
  - Universal Asynchronous Receiver-Transmitter
- Inicialmente utilizado para imprimir mensajes y verlos a través del monitor de logs
- También podemos utilizar la comunicación serie para comunicar diferentes Arduinos entre si, o con otros dispositivos.
  - Chips GPS normalmente incorporan puerto serie para la comunicación.

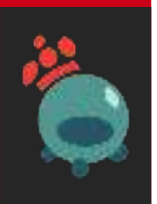


Comunicación UART



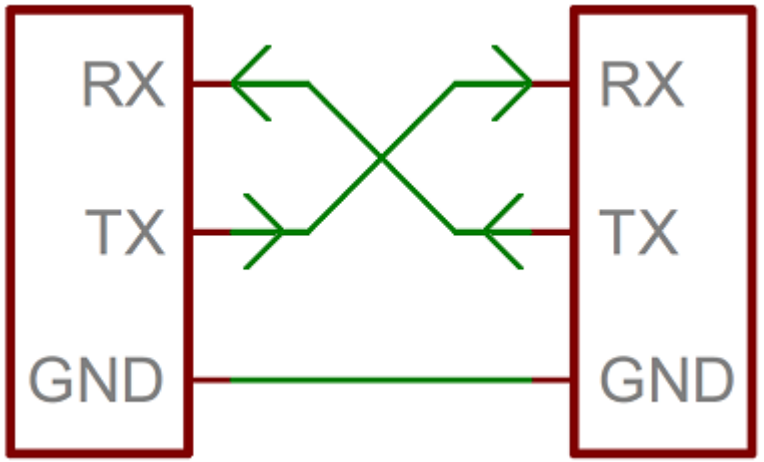
Comunicación UART

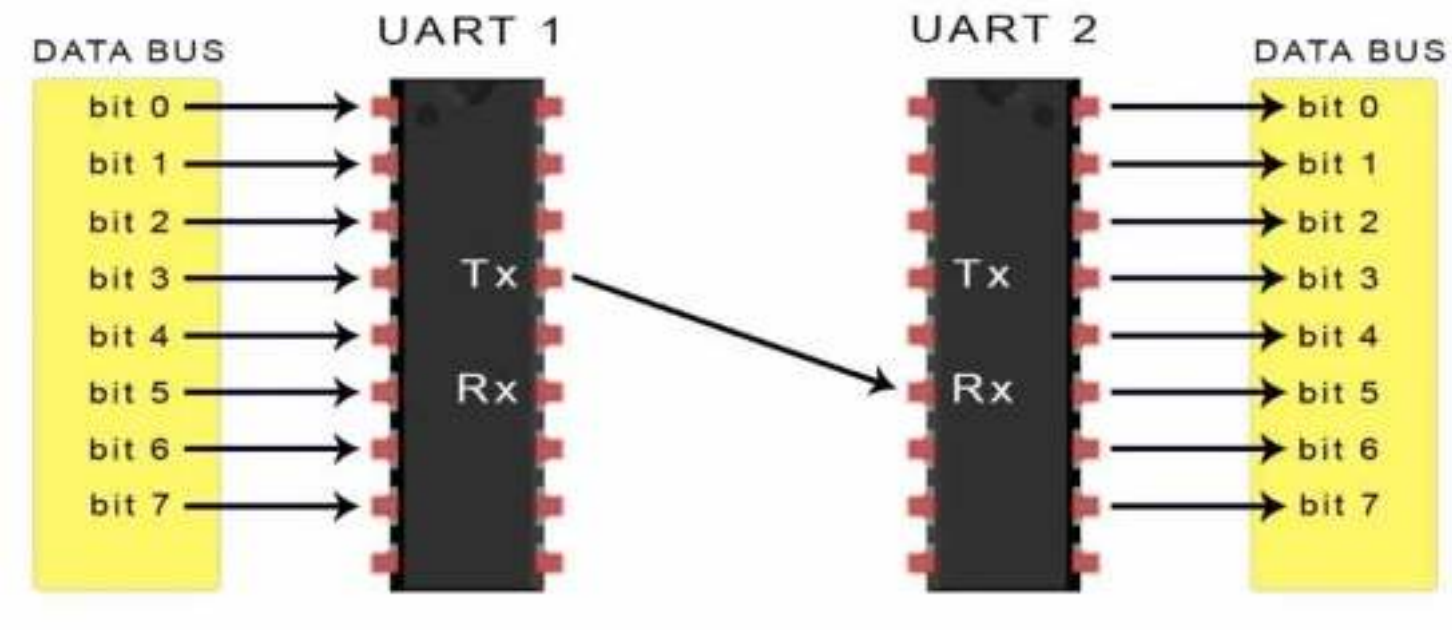
- Comunicación serie solo utiliza 2 pines:
  - TX para envío
  - RX para recepción
- No hay señal de reloj
- Los pines de comunicación TX/RX utiliza niveles lógicos TTL(Transistor-Transistor Logic).
  - Arduino utiliza 5V o 3.3V
  - Pero cuidado con RS232, utiliza 12V
- Utilizar TTL-USB para conectar con seguridad.



Comunicación UART

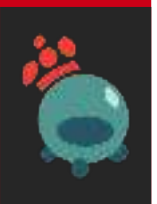
- Siempre conectar:
  - TX #1 con RX #2
  - RX #1 con TX #2





Comunicación UART

- Velocidades (en baudios)
  - 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200
- En telecomunicaciones, **baudios** es una medida utilizada que representa el número de símbolos por segundo en un medio de transmisión digital.
- Cada símbolo puede comprender 1 o más bits, dependiendo del esquema de modulación.
- En el esquema de comunicación serie 1 símbolo = 1bit
- 9600 baudios = 9600 bits/s



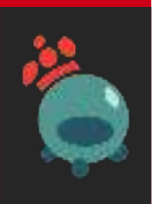
Comunicación UART

- Es posible comunicarse con otro Arduino mediante comunicación serie utilizando las siguientes funciones
  - print()-readBytes() y write()-read()

```
void setup() {
  Serial.begin(9600);
}
void loop() {

  if (Serial.available()) {
    buffer[Serial.readBytesUntil('\n', buffer, 256)] = '\0';

    Serial.println(String("ECHO: ") + buffer);
  }
  delay(1);
}
```

- Referencia a la librería:

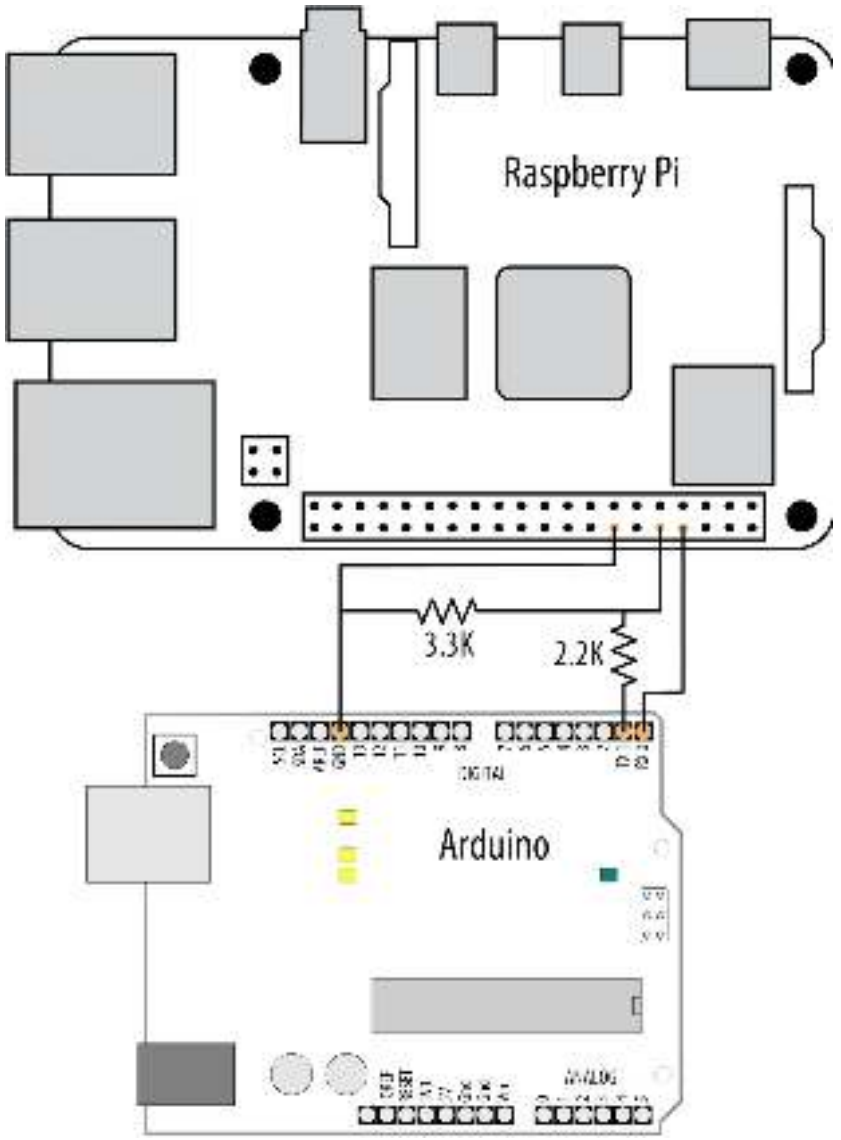
  https://www.arduino.cc/reference/en/language/functions/communication/serial/



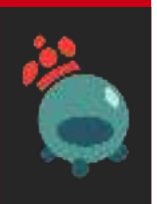
Comunicación UART: Ejemplos

- Comunicar Arduino con RaspberrypPi (python)

```python
#!/usr/bin/env python

import serial
from time import sleep

ser = serial.Serial('/dev/serial0', 9600)
ser.write('P13=1')
sleep(1)
ser.write('P13=0')
```



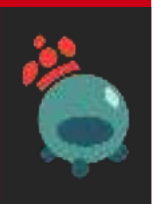
- Arduino Cookbook (3rd edition)



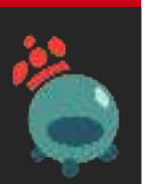
Comunicación UART: Resumen

- ✔ 2 líneas para envío y recepción
- ✔ Comunicación Asíncrona (no se comparte reloj).
- ✔ Velocidades entre 300 bps y 460 kbps
- ✔ Mecanismo simple

- ✗ Problemas a más de 15 metros.
- ✗ Limitado para la comunicación entre 2 dispositivos
- ✗ La velocidad se debe acordar al inicio, si no es la misma habrá problemas en la integridad de los datos recibidos.

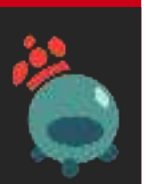


Comunicación I2C



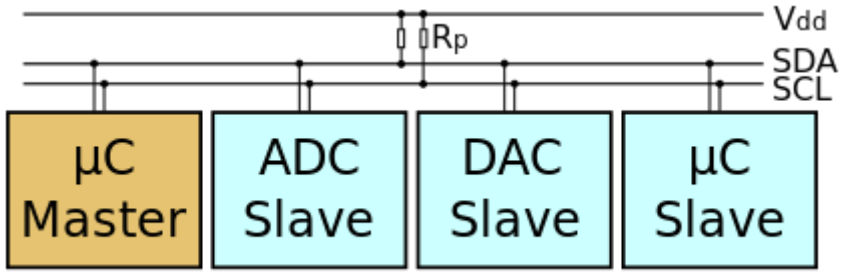
Comunicación I2C

- I2C: Inter-Integrated Circuit
- Protocolo de comunicación serie desarrollado por Philips (1982)
- Muy utilizado para añadir dispositivos de baja velocidad en microcontroladores en una distancia corta.
- Pensado para comunicación dentro de una placa.
- Lider-seguidor (maestro-esclavo)
- Numero ilimitado de lideres y un máximo de 1008 seguidores.
- Es un protocolo **síncrono** y hay una señal de reloj común.
- Velocidades: 100 kbps – 400 kbps (max 5 Mbps)
- Los dispositivos esclavo conectados al bus I2C se identifican por la dirección de chip, definida por el hardware/software

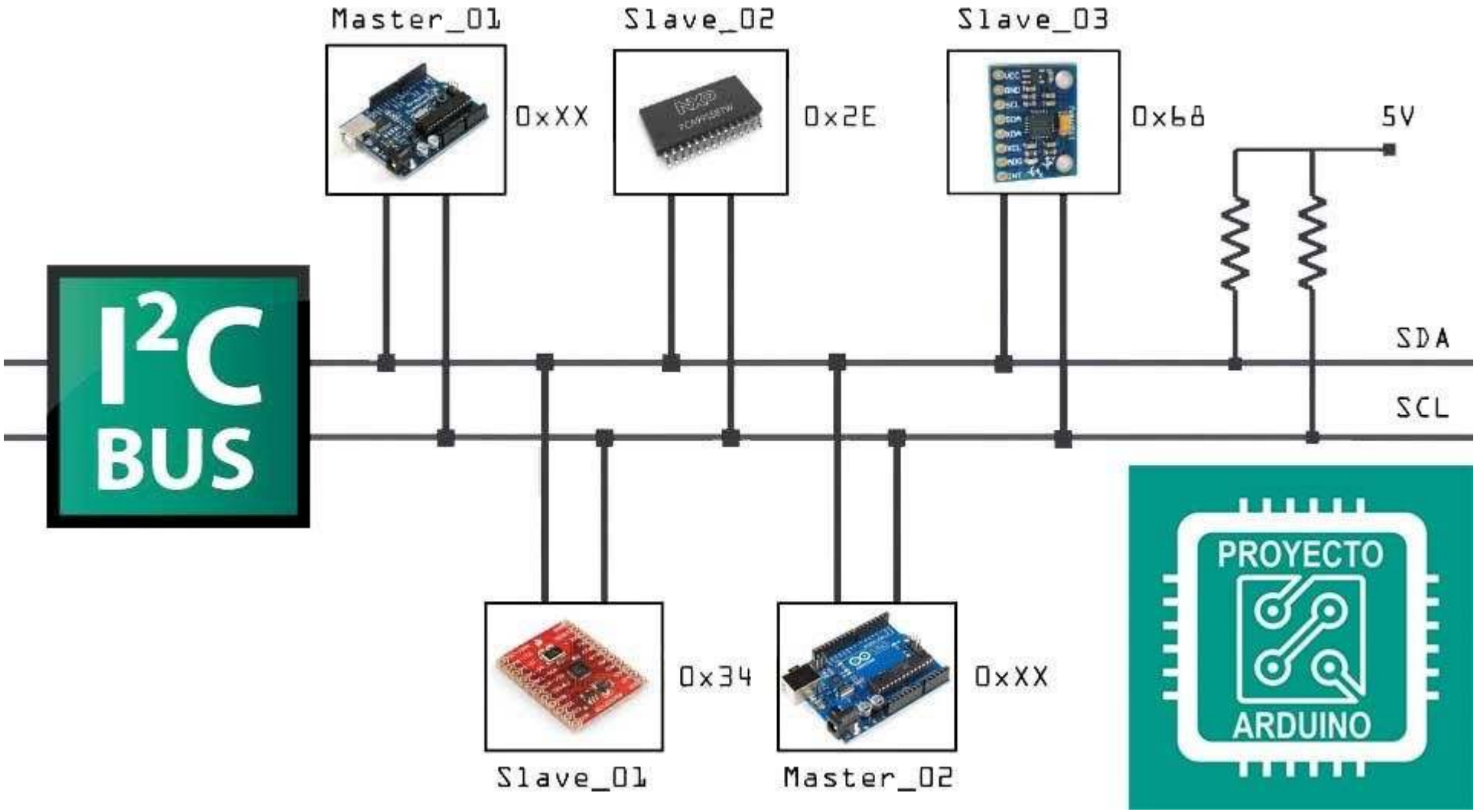


Comunicación I2C

- I2C utiliza sólo 2 pines de comunicación:
  - SDA (Serial Data). Canal de intercambio de información.
  - SCL (Serial Clock). Canal donde viaja la señal de reloj.
- Voltajes típicos son 3.3V o 5V
- El diseño de referencia de I2C tiene direcciones de 7bits

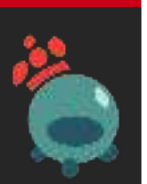


I2C


Master_01
0xXX
Slave_02
0x2E
Slave_03
0x68
5V
I²C BUS
SDA
SCL
Slave_01
0x34
Master_02
0xXX
PROYECTO
ARDUINO

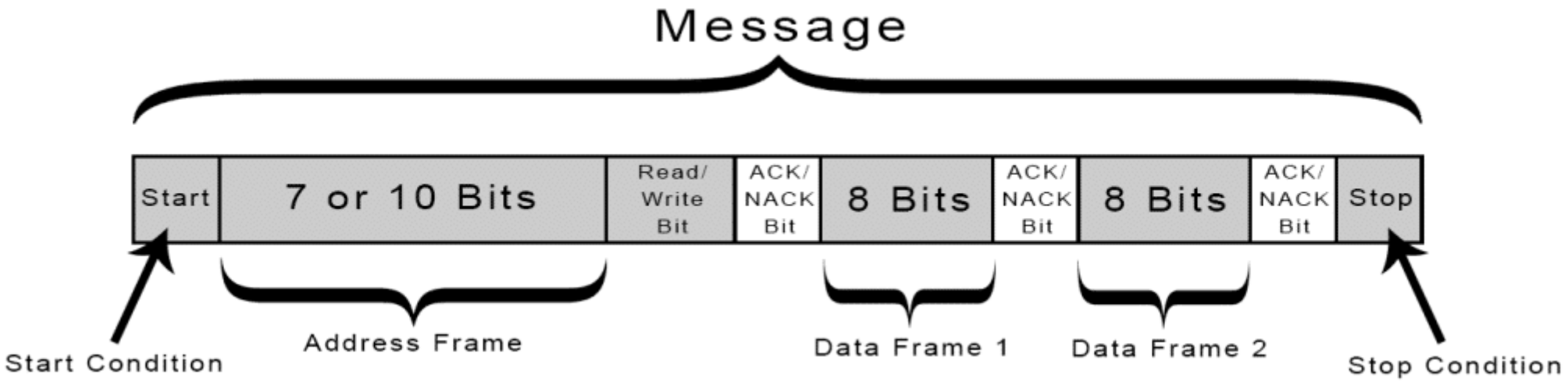
Comunicación I2C

- El diseño del bus I2C tiene dos roles diferenciados
  - **Nodo Líder** (máster): Genera la señal de reloj e inicia la comunicación con los nodos seguidores.
  - **Nodo Seguidor** (esclavo): Recibe la señal de reloj y responde cuando es definido por el líder.
- El bus de comunicaciones es un **multi-lider** bus, lo que significa que varios nodos lideres pueden estar presentes.
- Los roles (líder, seguidor) pueden ser intercambiados por ciertos nodos.



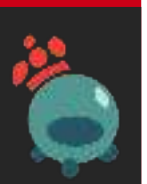
Comunicación I2C

- Los mensajes en I2C se componen de 2 partes:
  - **Address frame**: El nodo líder especifica a qué nodo seguidor va dirigido el mensaje.
  - **Data frame**: El mensaje de 8 bits que es enviado desde el nodo líder hacía el nodo seguidor (o viceversa).



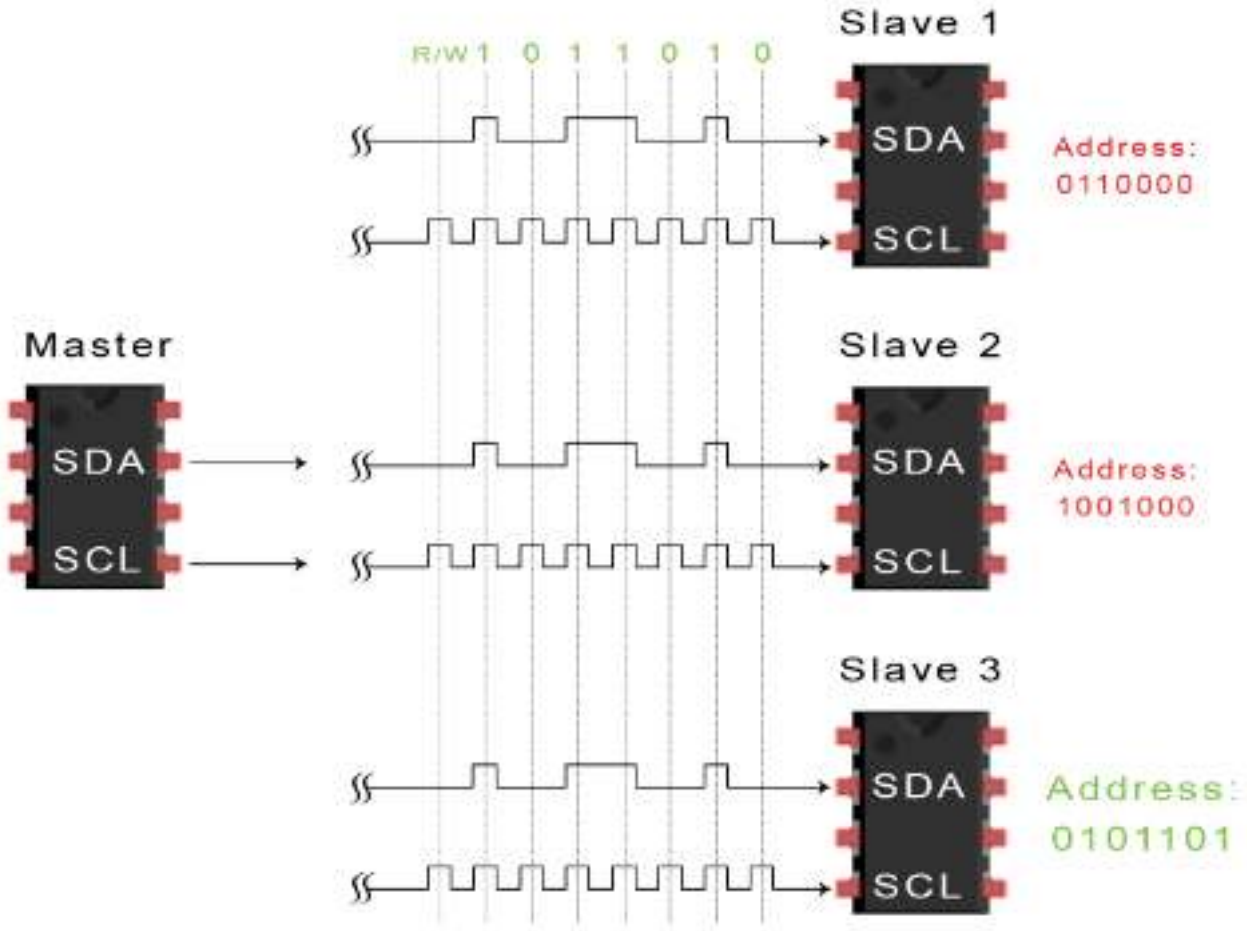
Comunicación I2C

- Para iniciar una comunicación el nodo lider:
  - Mantiene SCL en HIGH
  - Pone SDA en LOW
- Los nodos seguidores se preparan para la comunicación que va a comenzar.
- ¿Qué ocurre si 2 nodos lideres requieren el control del bus al mismo tiempo?
- Obtendrá el control de bus aquel nodo que ponga antes la señal SDA a LOW



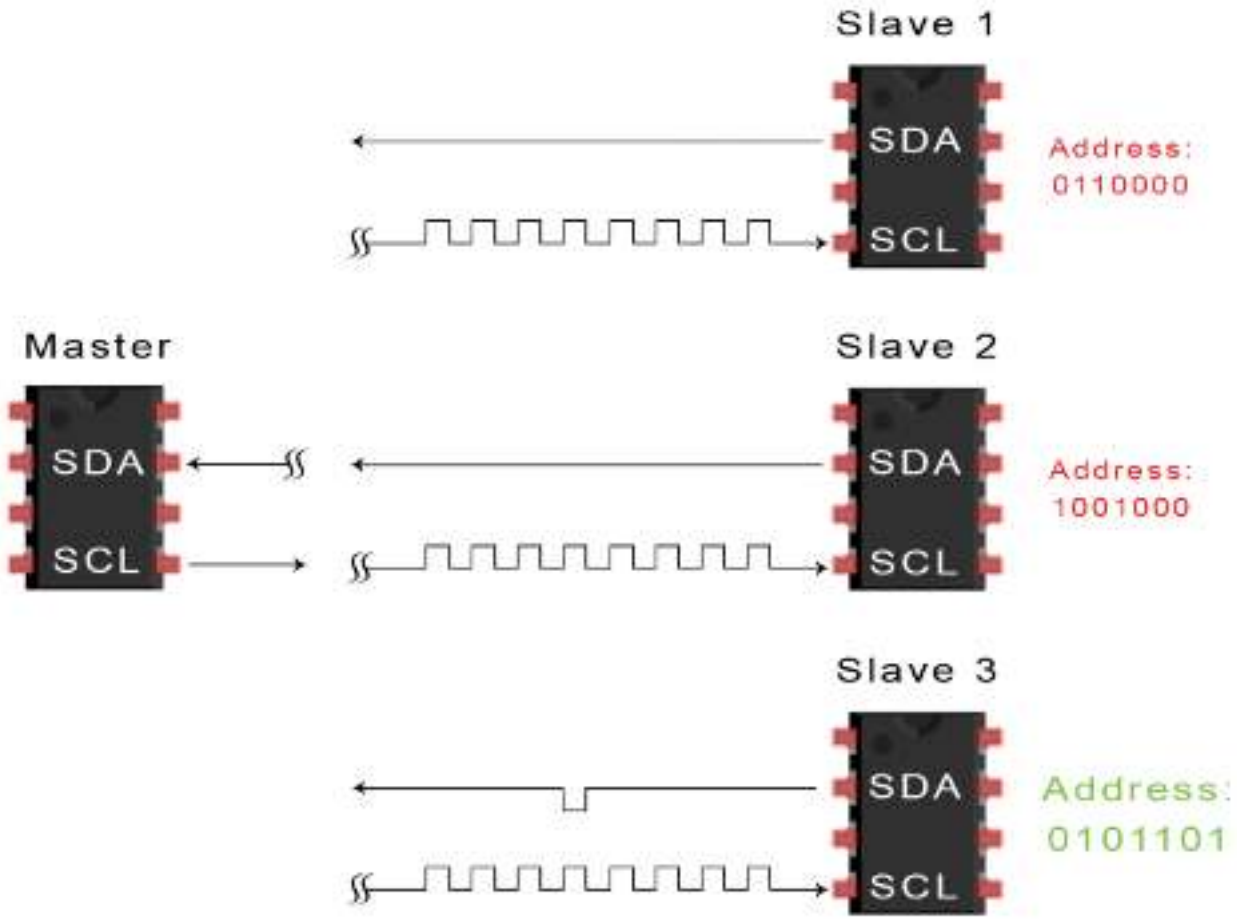
Comunicación I2C

- Una vez iniciada la comunicación con START, el nodo líder envía el “address frame” con la dirección del nodo seguidor.



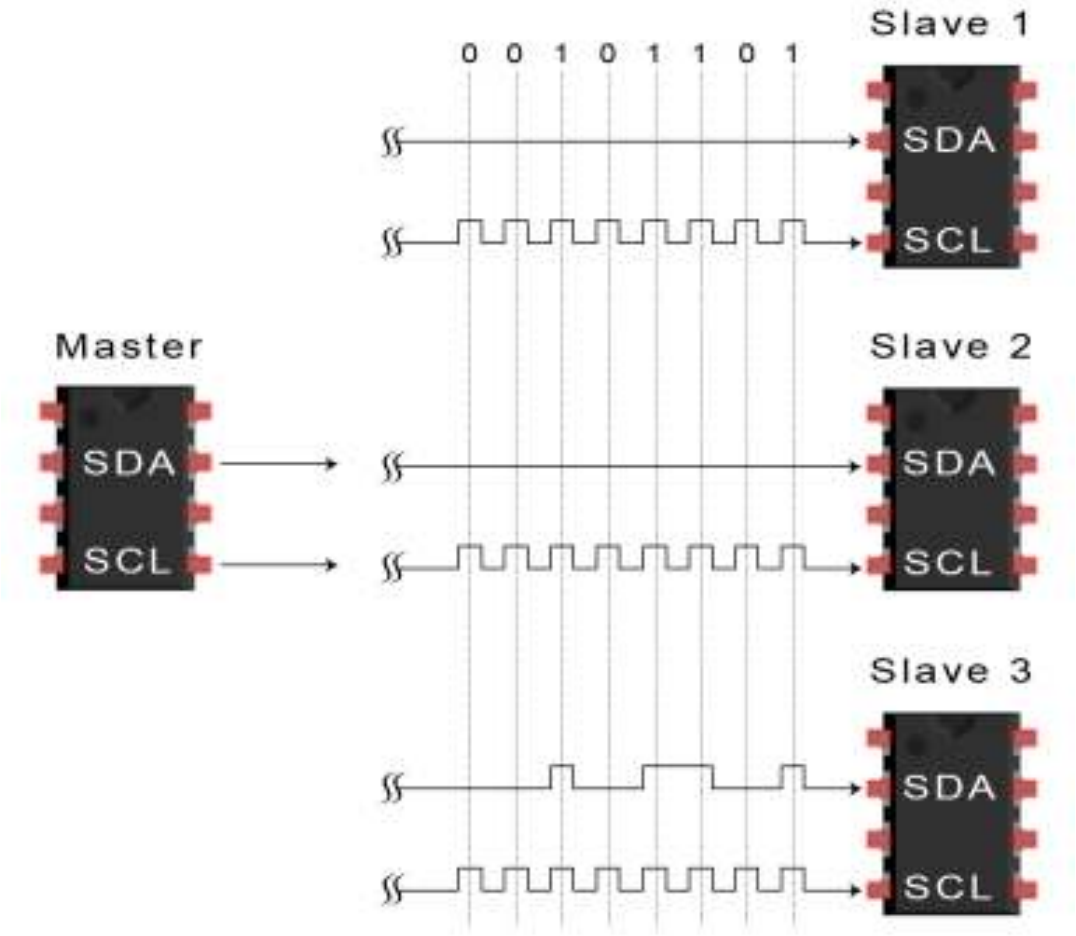
Comunicación I2C

- El nodo seguidor que coincida con la dirección especificada contestará con el bit ACK



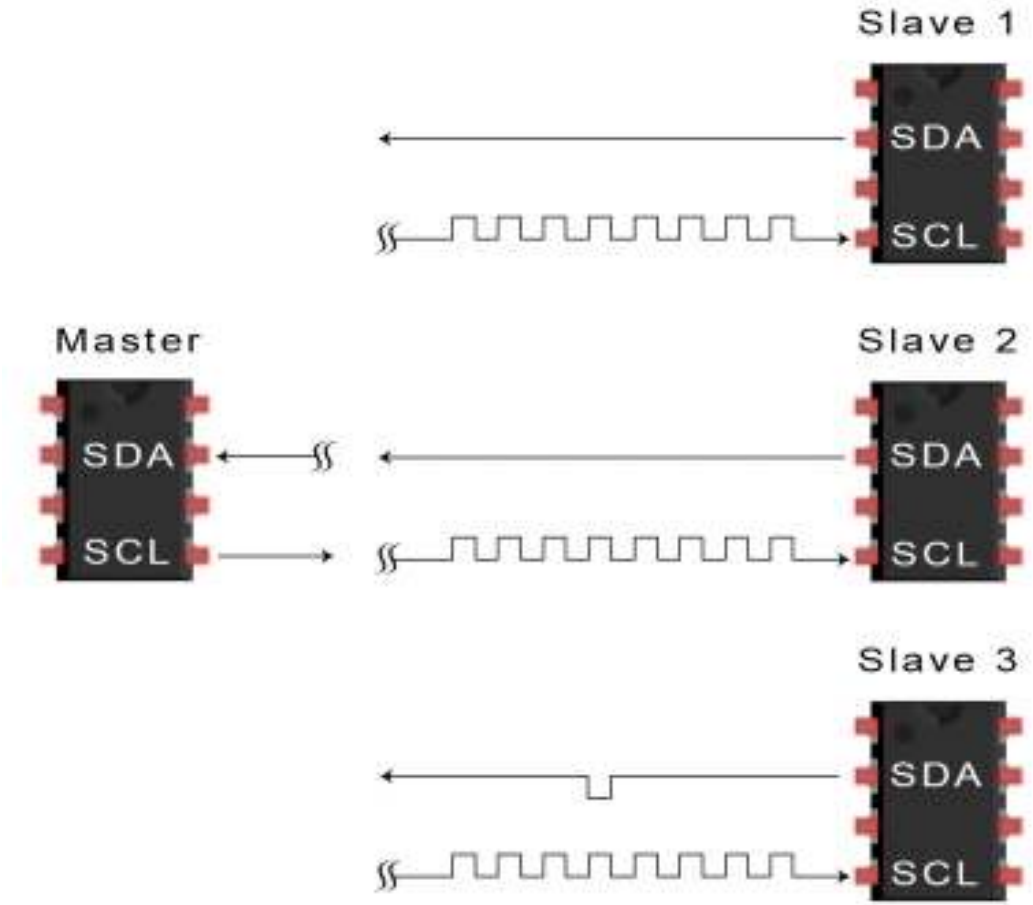
Comunicación I2C

- El nodo líder envía o recibe el data frame:



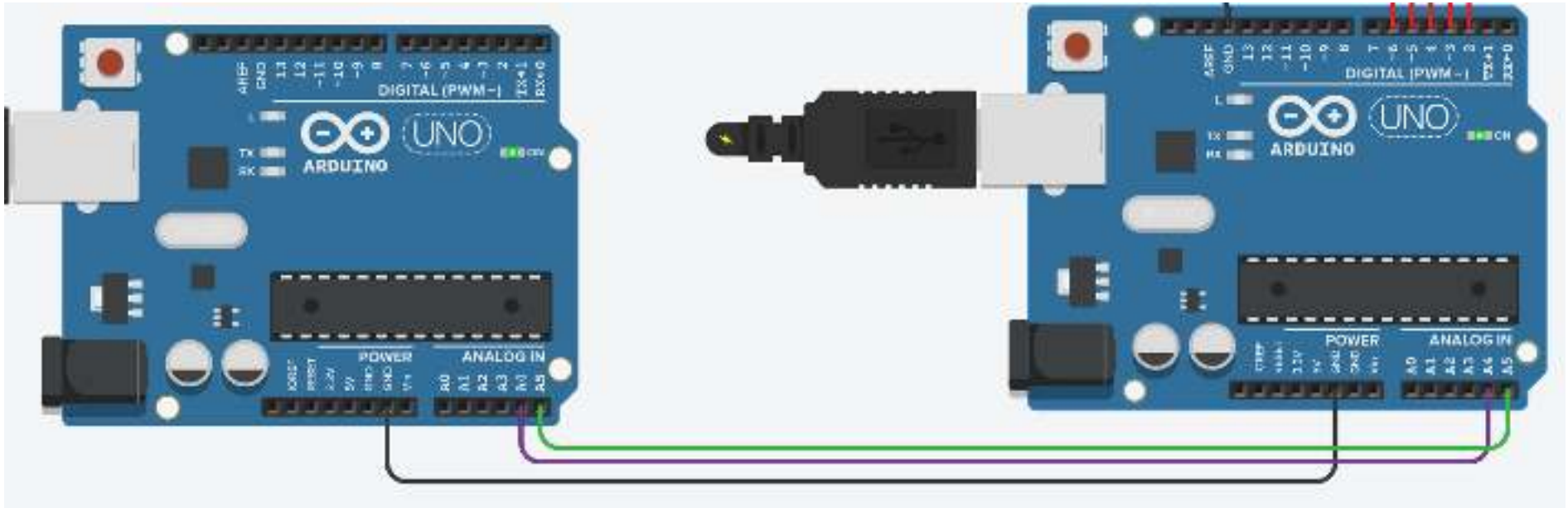
Comunicación I2C

- Después de cada envío, el nodo que recibe envía el ACK al nodo que envía para notificar la recepción correcta del mensaje.



Comunicación I2C: Arduino

- En Arduino disponemos de la librería ‘Wire’ que implementa la comunicación I2C
  - https://www.arduino.cc/en/reference/wire
- En la placa Arduino UNO:
  - A4 corresponde con SDA y A5 corresponde con SCL



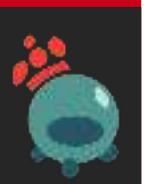
Comunicación I2C: Arduino

Master/Lider

```
#include <Wire.h>

const byte I2C_MASTER_ADDR = 0x10;
const byte I2C_SLAVE_ADDR = 0x20;

const byte SEND_PIN = 2;
const byte SEND_STATE = 1;

void setup() {

  // Añadimos este dispositivo al bus
  Wire.begin(I2C_MASTER_ADDR);
}

void loop() {

  Wire.beginTransmission(I2C_SLAVE_ADDR);
  Wire.write(SEND_PIN);
  Wire.write(SEND_STATE);
  Wire.endTransmission();

  delay(1);
}
```

Slave/Seguidor

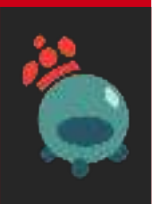
```
#include <Wire.h>

const byte byte I2C_SLAVE_ADDR = 0x20;

void setup() {

  // Añadimos este dispositivo al bus
  Wire.begin(I2C_SLAVE_ADDR);
  // Registramos evento de recepción
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
}

void loop() {
  delay(300);
}

void receiveEvent(int howMany) {

  if (Wire.available() == 1) {
     pin = Wire.read();
     Serial.print("PIN: " + String(pin));
  }
  if (Wire.available() == 2) {
     state = Wire.read();
     Serial.print("STATE: " + String(state));
  }
}
```



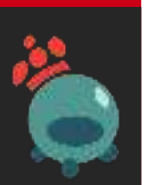
Comunicación I2C: Resumen

- ✔ Mayor velocidad que comunicación serie
- ✔ Muy extendido
- ✔ Utiliza únicamente 2 líneas
- ✔ Múltiples líderes y seguidores
- ✔ ACK/NACK permite confirmación de la transferencia

- ✗ Más lento que SPI
- ✗ Hardware más complicado
- ✗ Data frame limitado a 8 bits.

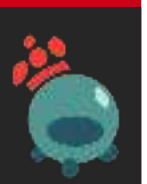


Comunicación SPI



Comunicación SPI

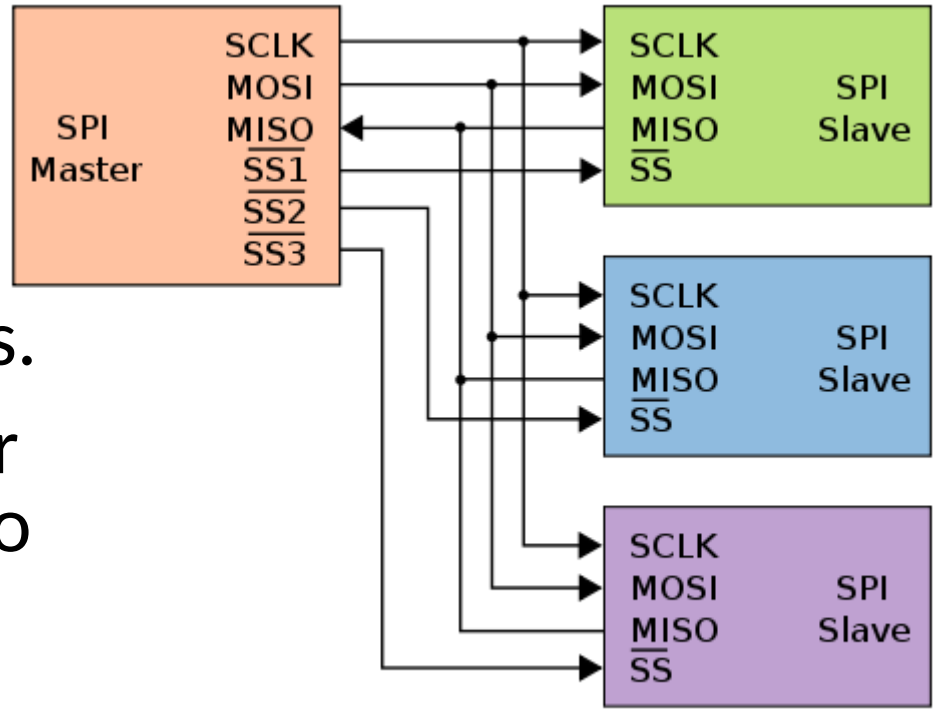
- SPI (Serial Peripheral Interface Bus)
- Desarrollado por Motorola (1980)
- Comunicación síncrona.
- Utilizado para comunicación a distancias cortas (on board)
- Utiliza arquitectura líder-seguidor (maestro-esclavo)
- Permite comunicación full-duplex
- Permite un único líder en el sistema.
- Hace uso de 4 líneas de comunicación.
- Velocidades



Comunicación SPI

- SCLK: señal de reloj para sincronizar los datos.
- MOSI: Master Output Slave Input
- MISO: Master Input Slave Output
- SS: Selección del Slave

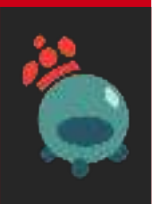
- Multitud de seguidores (slaves) están soportados, utilizando una línea individual por cada uno de ellos.
- Cualquier numero de bits pueden ser enviados/recibidos en un mismo flujo continuo.



Comunicación SPI

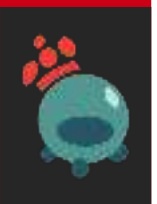
- Funcionamiento del protocolo

1) El nodo líder configura el reloj (SCLK)
2) El nodo líder selecciona el seguidor con el que quiere comunicarse activando la linea de selección (SS)
3) Durante cada ciclo de SPI, la comunicación full-duplex es permitida.
    - El nodo líder manda información (MOSI)
    - El nodo seguidor manda información (MISO)
4) La transmisión puede continuar por los ciclos de reloj necesarios.
5) Cuando la transmisión está completada, el líder para la señal de reloj y de-selecciona al nodo seguidor.



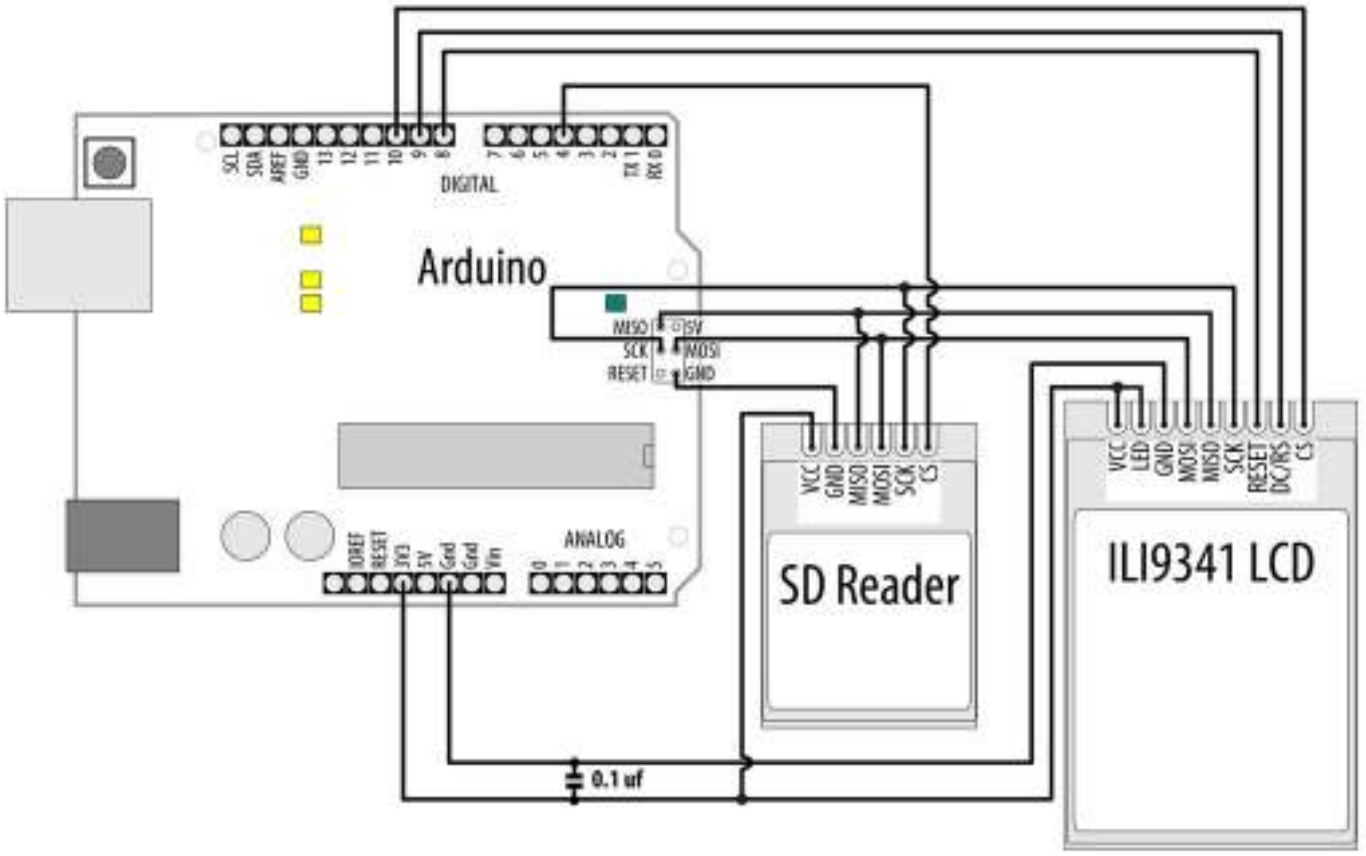
Comunicación SPI: Arduino

- En Arduino disponemos de la librería ‘SPI’ que implementa la protocolo de comunicación SPI
- En Arduino UNO los pines a utilizar son:
  - SCK: Pin 13
  - MOSI: Pin 11
  - MISO: Pin 12
  - SS: Pin 10
- Muchos de los sensores que usan SPI tienen sus propias librerías para interactuar con ellos.



Comunicación SPI: Ejemplo

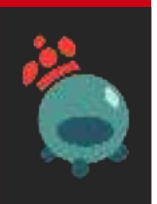
- Arduino CookBook (3rd edition)
- Lector SD y LCD funcionan con el protocolo SPI



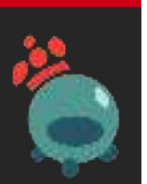
Comunicación SPI: Resumen

- ✔ Mayor velocidad
- ✔ Full-duplex
- ✔ No está limitado a palabras de 8 bits.
- ✔ Hardware más sencillo, multitud sensores compatibles.

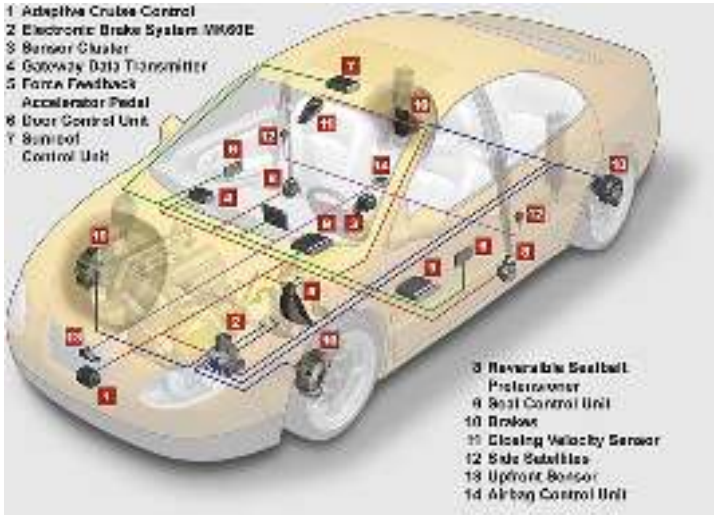
- ✗ Requiere más lineas.
- ✗ Direccionamiento a través de lineas específicas
- ✗ Longitud del mensaje debe ser conocido por ambas partes.
- ✗ No hay ACK en la transmisión.
- ✗ Solo soporta un único nodo líder (máster)



CAN BUS



CAN BUS

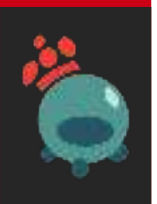


- Controller Area Network (CAN)
- Protocolo de comunicaciones
- Creado por BOSH en 1983
- Mercedes fue uno de los primeros en usarlo (1992)
- Primeros estándares ISO en 1993-1994
- Su misión principal es intercomunicar las diferentes ECU (Electronic control units) de un sistema.

CAN BUS

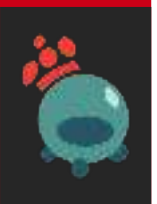
- Conexión entre multitud de microcontroladores y ECU a través de únicamente 1 par de cables.
- Alta velocidad (1MBit/s) y transmisión en tiempo real
- Alta inmunidad a interferencias en un entorno con alto ruido electromagnético
- Bajo coste
- Distancias de BUS de hasta 1 Km.

Bus Length (m)	Signaling Rate (Mbps)
40	1
100	0.5
200	0.25
500	0.10
1000	0.05



CAN BUS

- Es una red cerrada, por lo que no se implementan característica de seguridad, login o cifrado.
- Prioridad de mensajes.
- Garantía de tiempos de latencia.
- Flexibilidad en la configuración.
- Recepción por multidifusión (multicast)
- Sistema multimaestro.
- Retransmisión automática de tramas erróneas
- Funciona a través de señales lógicas CAN_HIGH y CAN_LOW

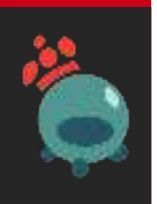


CAN BUS

- Tramas (mensajes)

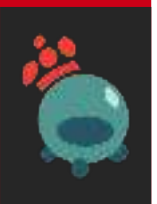
Figure 3. Standard CAN: 11-Bit Identifier

Figure 4. Extended CAN: 29-Bit Identifier



CAN BUS

- Normalmente en un automóvil nos encontramos 2 redes CANBUS diferenciadas
  - Una red de alta velocidad (hasta 1 Mbit/s), bajo el estándar ISO 11898-2
    - Destinada para controlar el motor e interconectar las unidades de control electrónico (ECU);
  - Una red de baja velocidad tolerante de fallos (menor o igual a 125 kbit/s), bajo el estándar ISO 11519-2/ISO 11898-3
    - Comunicación de los dispositivos electrónicos internos de un automóvil como son control de puertas, ventanillas, luces y asientos

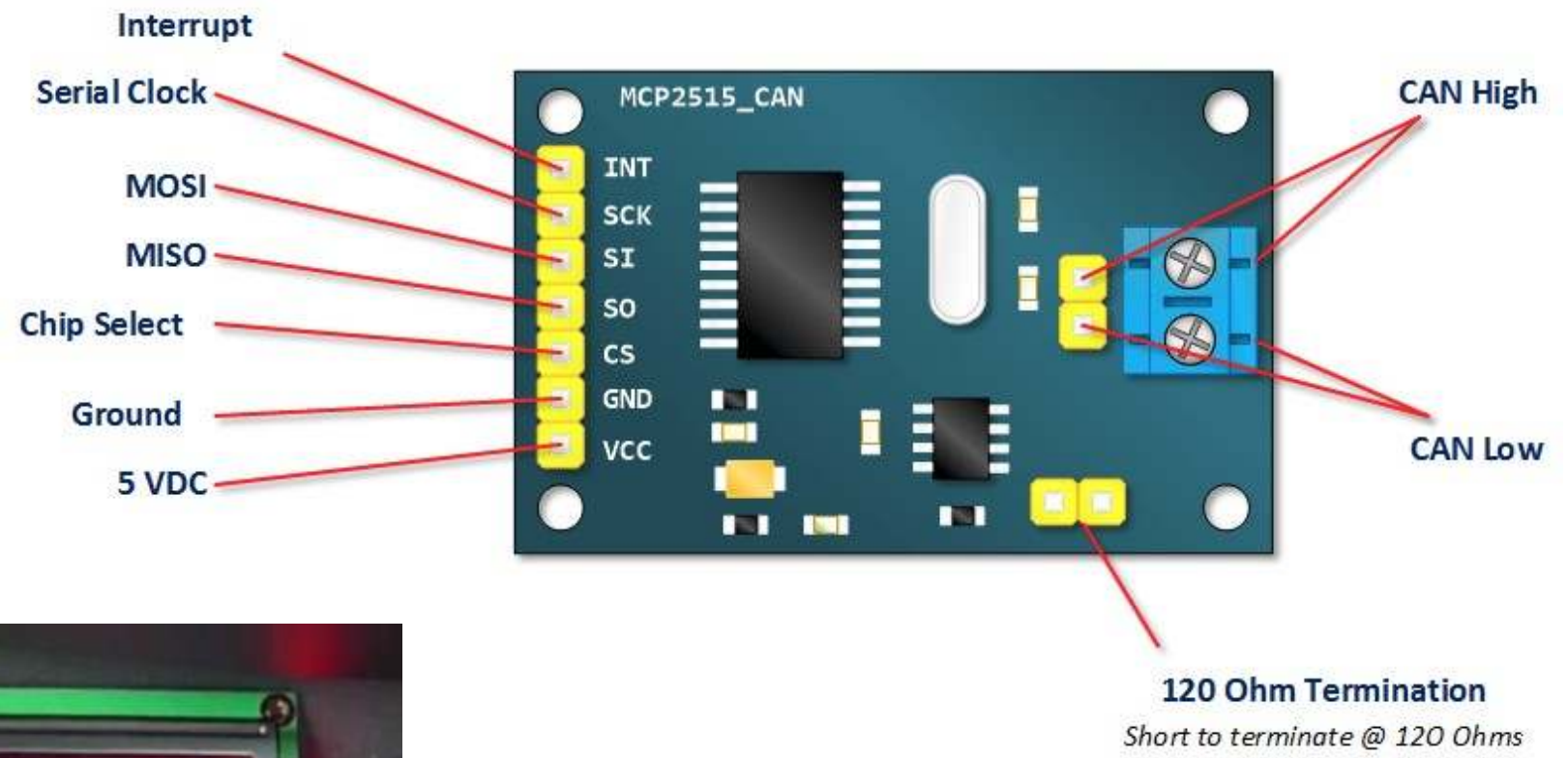


CAN BUS - OBD

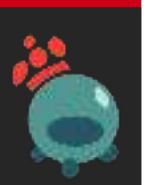
- OBD es un protocolo de diagnóstico a bordo que se comunica con el CAN BUS para realizar diagnosis o detectar fallos en sistemas de automoción.
- OBD y CAN BUS son protocolos estándar, pero cada fabricante puede crear sus propias tramas con códigos de error diferentes.

CAN BUS - Arduino

- MCP2515: Módulo de comunicación CAN BUS



Ethernet/WiFi



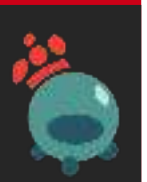
Ethernet

Arduino Shield Ethernet

Arduino Ethernet

Ethernet

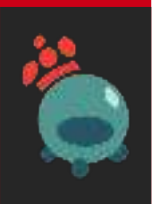
- Arduino provee de una librería para utilizar la tarjeta de red a alta nivel:
  - https://www.arduino.cc/en/Reference/Ethernet
- Librería diseñada para funcionar con Arduino Ethernet Shield, Arduino Ethernet Shield 2, Leonardo Ethernet, y cualquier otro dispositivo basado en W5100/W5200/W5500
- Utiliza SPI para la comunicación
- Posibilidad de utilizar TCP y UDP



Ethernet

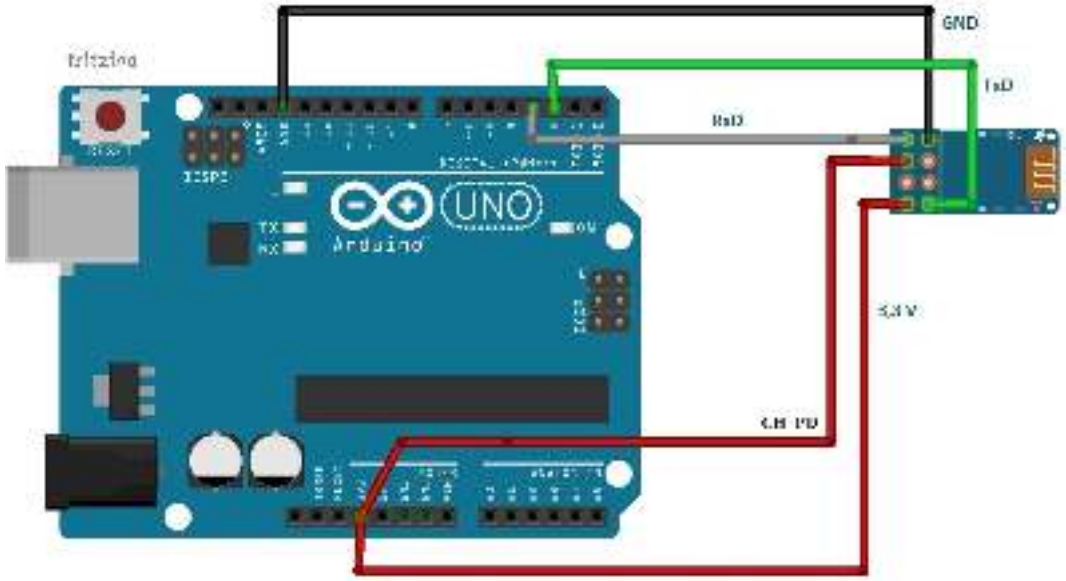
- Ejemplo de servidor web
  - https://www.arduino.cc/en/Tutorial/LibraryExamplesWebServer
- Muy útil para mostrar información adquirida desde los sensores.

- Consultar el código:
  - https://create.arduino.cc/example/library/ethernet_2_0_0/ethernet_2_0_0%5Cexamples%5CWebServer/WebServer/preview



WiFi

ESP8266 + Arduino

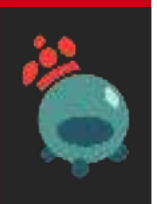


Arduino MKR WiFi 1010

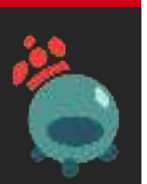
(Arduino Oplà IoT Kit)

WiFi

- Librería
  - https://www.arduino.cc/en/Reference/WiFiNINA
- Ejemplo de servidor web sencillo
  - https://www.arduino.cc/en/Tutorial/WiFiNINAAPSimpleWebServer

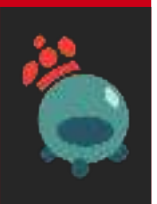


MQTT



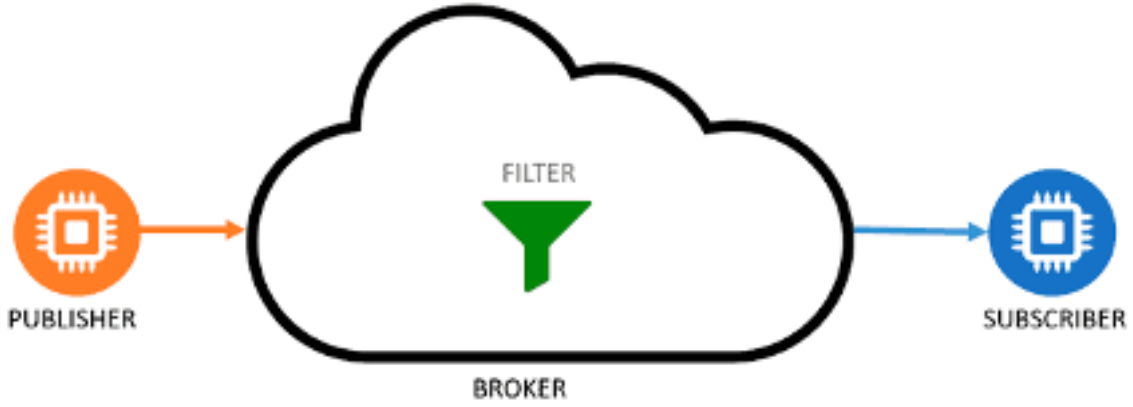
MQTT

- Protocolo de comunicación M2M (Machine-to-Machine)
- Basado en Publicador/Subscriptor
- Muy liviano y sencillo de procesar.
- Normalmente ejecuta sobre TCP/IP
- Pensado para comunicar remotamente hardware de bajo coste, sin muchos recursos computacionales.
- Consumo energético bajo.



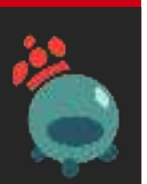
MQTT

- Se le denomina **protocolo de IoT** ya que se ha extendido su uso en estos dispositivos gracias a su sencillez y ligereza.
- MQTT precisa de un servidor central por donde se encaminan todos los mensajes (**Broker**)
- Hay nodos que publican mensajes a través de topics (**Publisher**)
- Hay nodos que se subscriben a ciertos topis pare recibir mensajes (**Subscriber**)



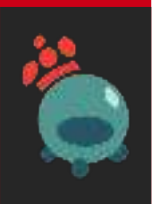
MQTT

- MQTT dispone de mecanismos de calidad de servicio (QoS).
- Diferentes niveles de QoS para la robustez de envíos:
  - **QoS 0** (at most one): El mensaje solo se envía una vez, en caso de fallo puede que no se reciba correctamente.
  - **QoS 1** (at least one): El mensaje se envía hasta que se garantice la entrega (subscriptor podría recibir mensajes duplicados)
  - **QoS 2** (exactly one): Se garantiza que el mensaje se entrega al subscriptor una única vez.
- Dependiendo de las características y necesidades de nuestro sistema usaremos el nivel apropiado de QoS.



MQTT

- Incluye capa de transporte SSL/TSL para autenticación mediante contraseña o certificados.
- Estándar en IoT
- Seguridad y calidad del servicio (QoS)
- Requiere un ancho de banda mínimo.
- Menor consumo de energía



MQTT - Demo Linux

- Mosquitto es una implementación libre de MQTT y funciona en Debian, Ubuntu, Rpi, Windows, Mac. https://mosquitto.org/

- Ejemplos:
  - Subscribir a todos los topics posibles

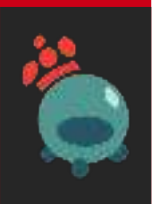
```
mosquitto_sub -v -h 0.0.0.0 -p 1883 -t '#'
```

  - Subscribir a un topic específico

```
mosquitto_sub -v -h 0.0.0.0 -p 1883 -t '/Robot/odom/'
```

  - Enviar un mensaje en un topic concreto

```
mosquitto_pub -h 0.0.0.0 -p 1883 -t /Robot/speed/ -m "speed:12.1"
```

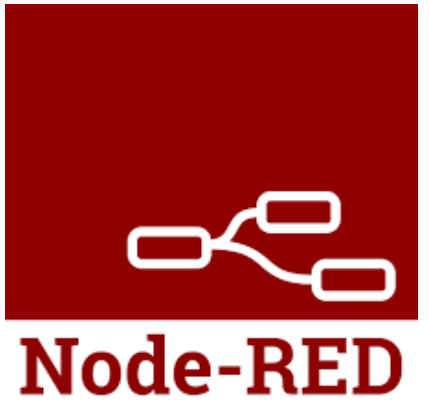


MQTT - Arduino

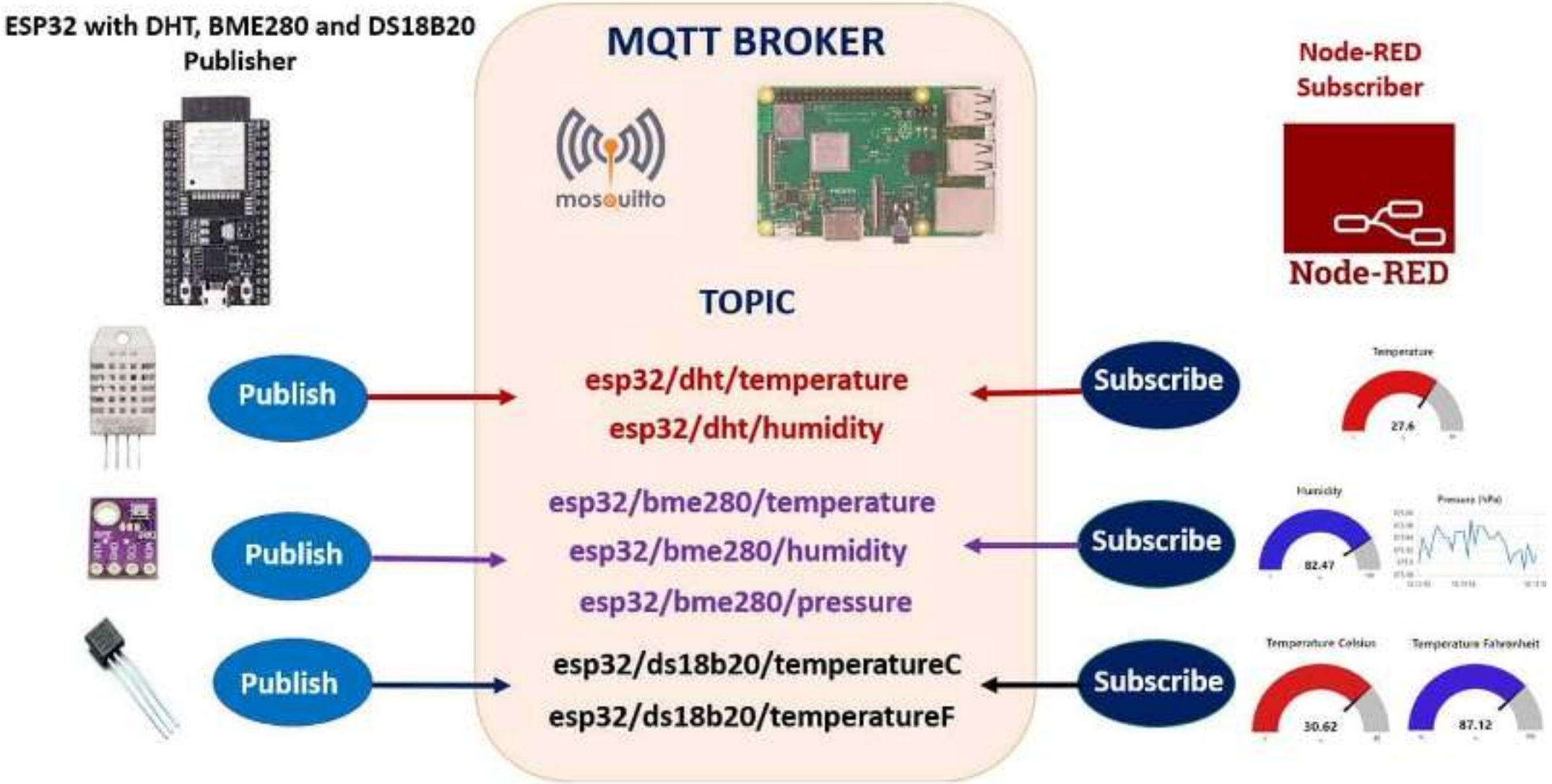
- Existen multitud de librerías para dar soporte MQTT a microcontroladores, entre ellos Arduino.
- Adafruit
  - https://github.com/adafruit/Adafruit_MQTT_Library
- Arduino Client for MQTT
  - https://github.com/knolleary/pubsubclient

MQTT - NodeRed

- Herramienta de programación visual que permite interconectar diferentes dispositivos de nuestro sistema
- Muy utilizado para gestión y procesado de datos en tiempo real y muy unido a proyectos IoT.
- Está desarrollado en NodeJS e internamente los componentes intercambian información a través de JSON.
- https://nodered.org/

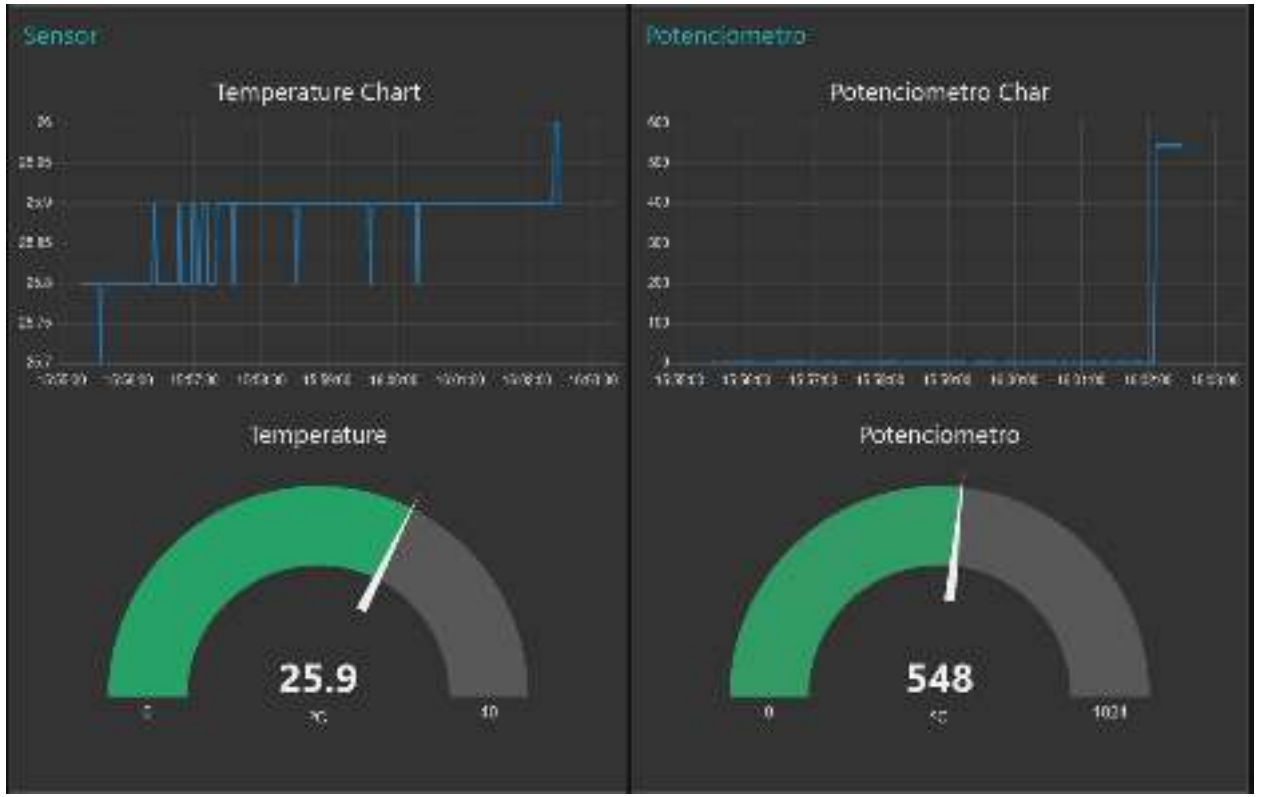


MQTT - NodeRed



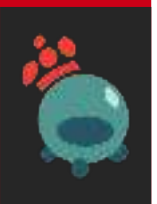
NodeRed

- Ejemplo demostración
- https://gitlab.etsit.urjc.es/roberto.calvo/setr/-/tree/main/MQTT-NodeRed



Bibliografia

- Comunicación I2C
  - https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/
- Arduino Communication Peripherals: UART, I2C and SPI
  - https://www.seeedstudio.com/blog/2019/11/07/arduino-communication-peripherals-uart-i2c-and-spi/
- Controller Area Network Physical Layer Requirements
  - https://www.ti.com/lit/an/slla270/slla270.pdf



URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela Técnica Superior de Ingeniería de Telecomunicación

GSyC

RoboticsLabURJC
Programming Robot Intelligence

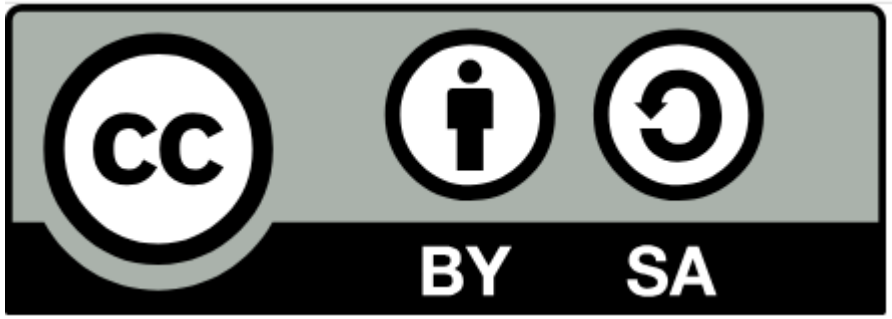
Sistemas Empotrados y de Tiempo Real

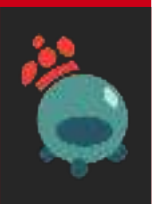
Compilación Cruzada

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

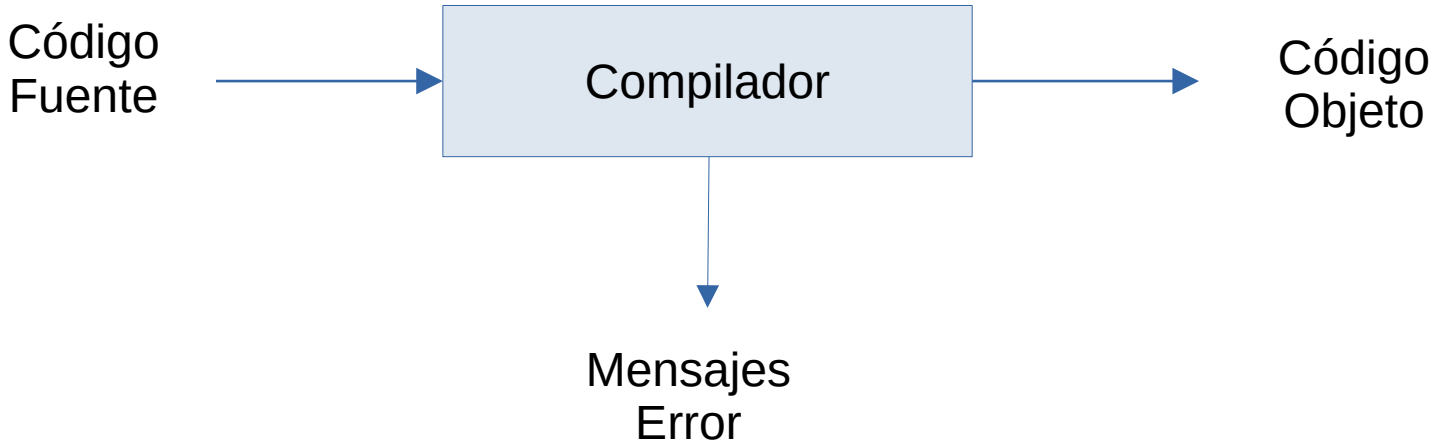
Roberto Calvo Palomino
roberto.calvo@urjc.es



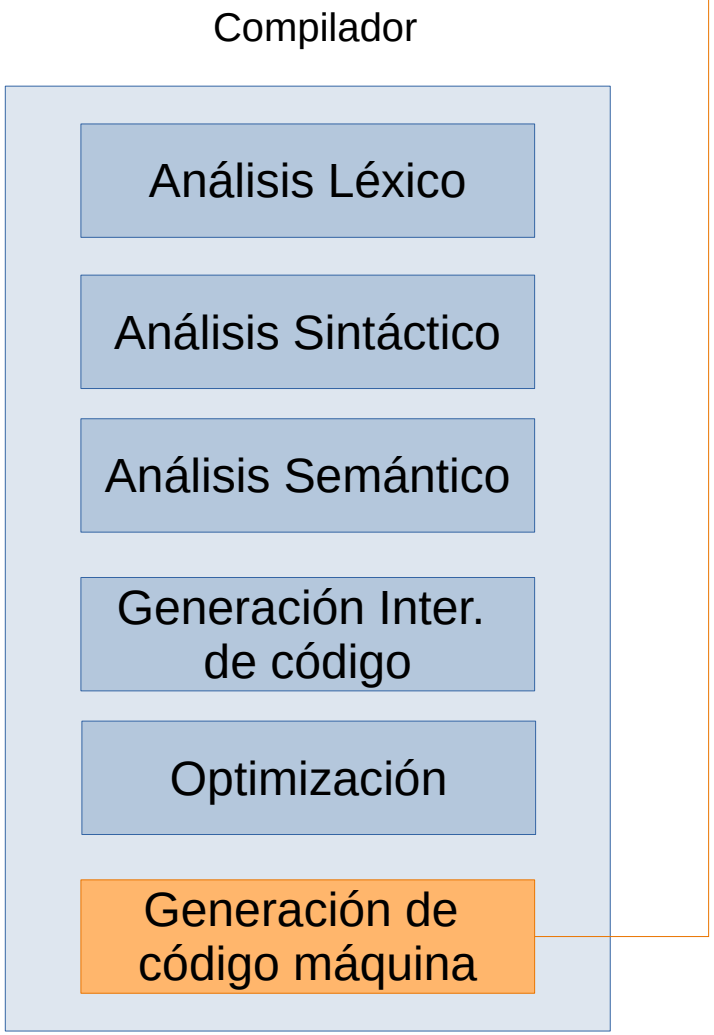


Compilador

- Programa que lee un código escrito en un lenguaje y lo traduce a otro lenguaje.
- En el ámbito de la programación, nos referimos a que un código fuente es compilado a un código objeto.



Compilador



- La generación de código máquina necesita conocer en qué arquitectura correrá el programa compilado.
- Intel, ARM, atmega, PIC
- El mismo código fuente escrito en C, puede ser compilado para diferentes arquitecturas.

Compilación en entorno empotrados

- Normalmente el proceso de compilación en entornos empotrados puede requerir más recursos (memoria, disco, procesamiento) que la que la propia arquitectura empotrada dispone.
- Además, debido al procesamiento limitado de los dispositivos empotrados la compilación puede llegar a tardar mucho más tiempo:
  - Compilación OPENCV en RBPi (arm) ~ 4 horas
  - Compilación OPENCV en laptop (x86_64/amd64) para arquitectura arm ~ 30 min

Compilación cruzada

- **Compilación cruzada** o **cross-compilation** se refiere a la técnica de compilar un código fuente en una arquitectura *A1* generando un código objeto para una arquitectura *A2*

Compilación en entorno empotrados

- Podemos compilar el mismo código fuente para:
  - x86_64/amd64 gcc
  - ARM arm-linux-gnueabihf-gcc
  - Atmel AVR 8-bit avr-gcc

```c
#include <stdio.h>

int main (void) {

        printf("Hello World\n");
        return 0;
}
```

Compilación en entorno empotrados

- Compilación en x86_64/amd64

```
$ gcc hello.c -o hello
$ file hello
hello:     ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=e905fd610172e547ea572f6f04ab803c43f2a17e, for GNU/Linux 3.2.0, not stripped
```

-

```
$ arm-linux-gnueabihf-gcc hello.c -o hello_arm
$ file hello_arm
hello_arm: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 2.6.26,
BuildID[sha1]=b52c2315a46258931c7890edc67ab56f307d1421, not stripped
```

-

```
$ /usr/bin/avr-gcc hello.c -o hello_avr
$ file hello_avr
hello_avr: ELF 32-bit LSB executable, Atmel AVR 8-bit, version 1 (SYSV), statically linked,
not stripped
```

Ventajas / Desventajas

- ✔ Velocidad en el proceso de compilación.
- ✔ Facilidad en el proceso de depuración.
- ✔ No es necesario disponer del hardware del sistema empotrado para realizar compilaciones.
- ✔ Mismo código genera diferentes *targets*.

- ✗ El compilador cruzado normalmente es más limitado que el compilador nativo.
- ✗ Complejidad de código para mantener la compatibilidad con las N arquitecturas *targets/objeto* soportadas.

URJC Escuela Técnica Superior de Ingeniería de Telecomunicación

GSyC

RoboticsLabURJC

Programming Robot Intelligence

URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC Programming Robot Intelligence

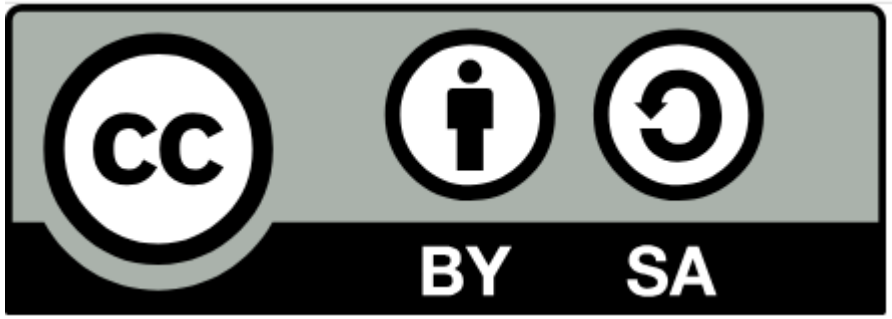
Sistemas Empotrados y de Tiempo Real

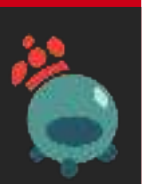
FreeRTOS en Arduino

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

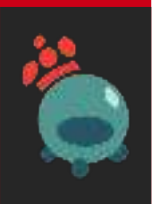
Roberto Calvo Palomino
roberto.calvo@urjc.es





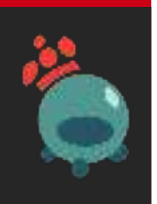
FreeRTOS

- **FreeRTOS** es un kernel de tiempo real para dispositivos embebidos que está portado a 35 plataformas de microcontroladores.
  - Intel, ARM, Atmel, ESP, Xilinx, ….
- Liberado como software libre (MIT)
  - https://www.freertos.org/
- Permite organizar aplicaciones para microcontroladores como una colección de hilos independientes de ejecución.
- Ejecución de hilos basada en prioridades.
- En FreeRTOS, cada hilo de ejecución se denomina **tarea**



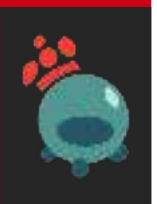
FreeRTOS - Beneficios

- Abstracción de la información del tiempo
- Escalabilidad
- Modularidad
- Facilita el desarrollo en equipo
- Utilización del tiempo “idle”

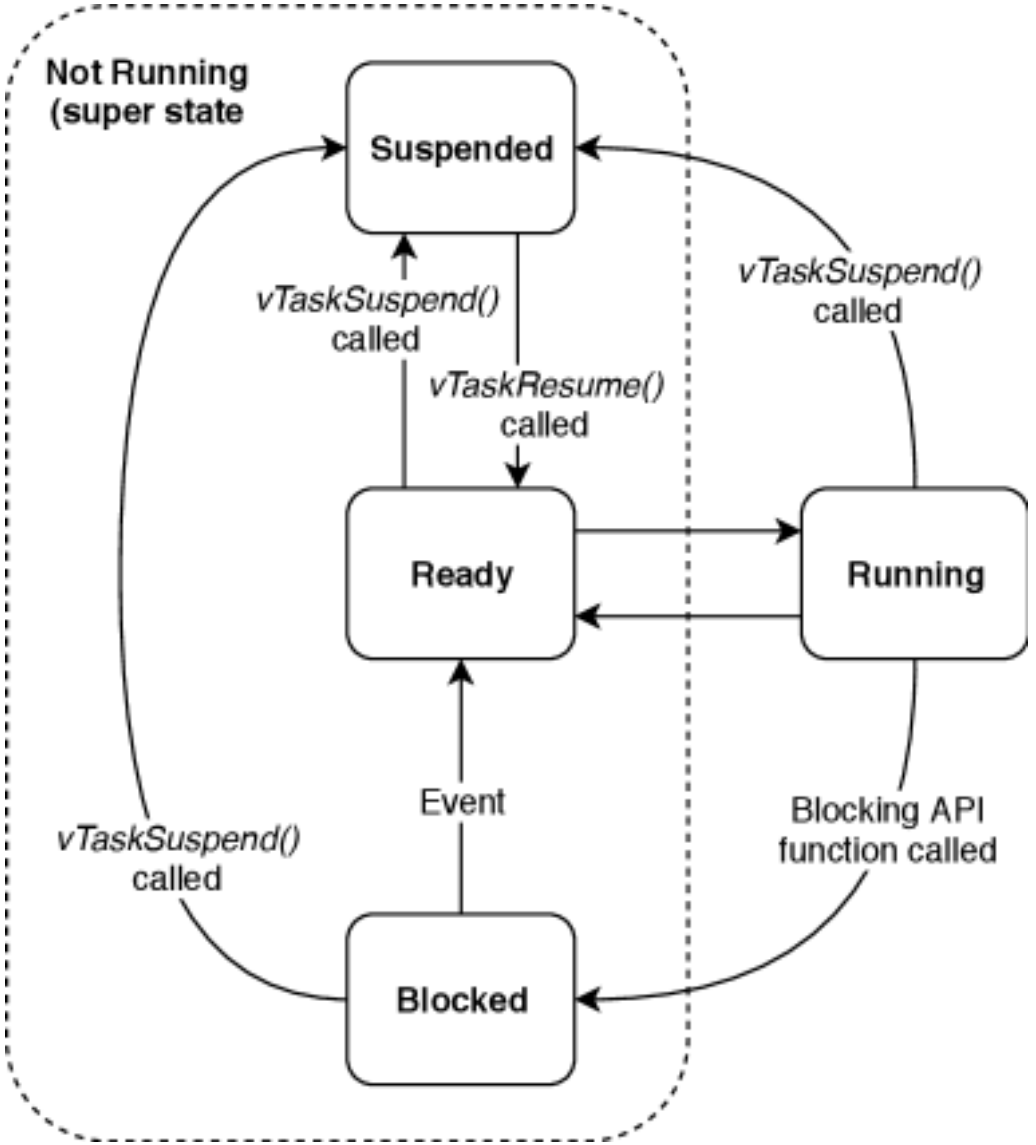


FreeRTOS - Características

- Operación “pre-emptive”
- Asignación de prioridad a tareas muy flexible.
- Colas, utilizadas para el paso de mensajes entre tareas.
- Uso de semáforos binarios, recursivos, de conteo y de exclusión mutua.
- Funciones para consultar Tick/Idle
- Interrupciones anidadas



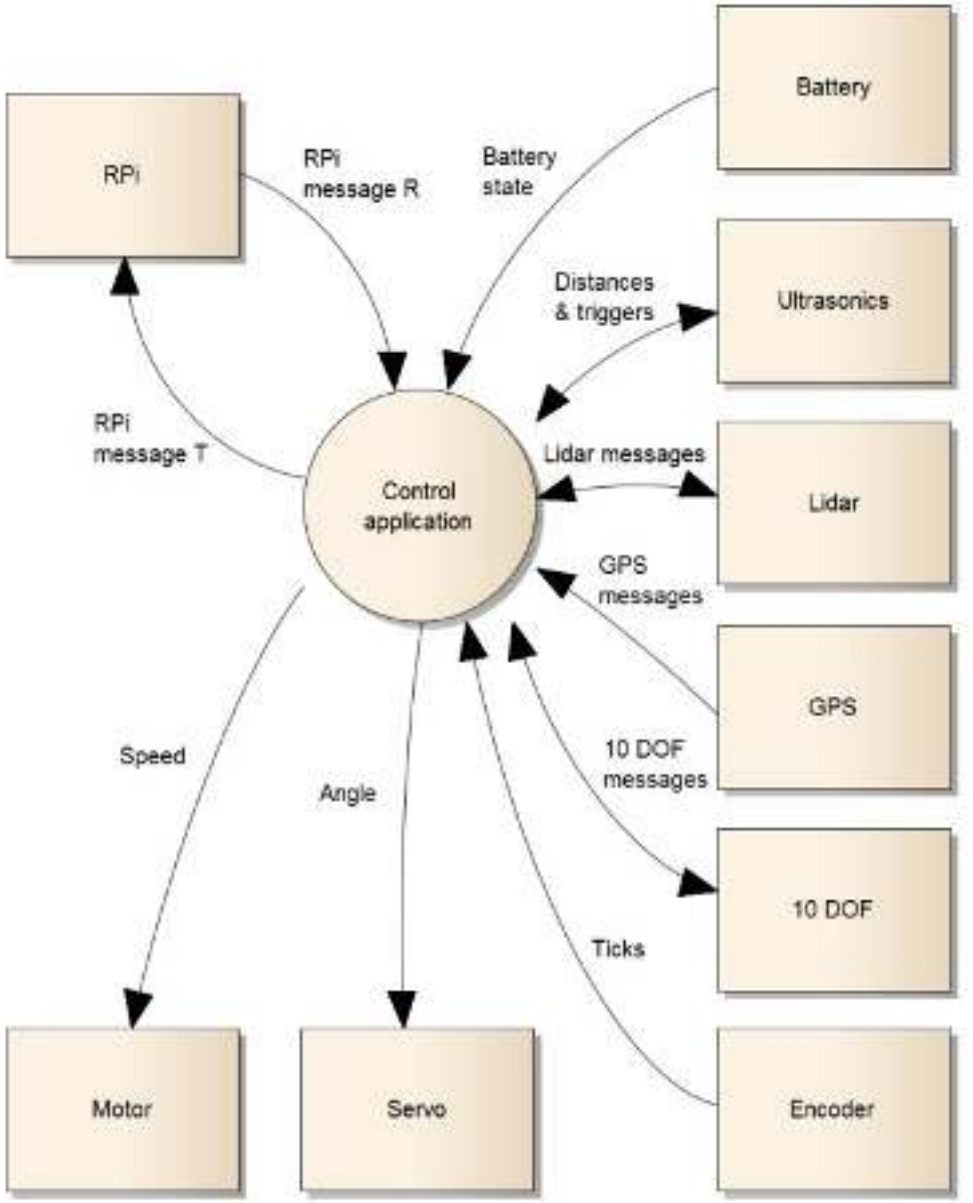
FreeRTOS


Not Running
(super state
Suspended
vTaskSuspend()
called
vTaskResume()
called
vTaskSuspend()
called
Ready
Running
Event
vTaskSuspend()
called
Blocking API
function called
Blocked

FreeRTOS

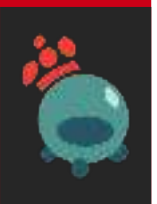
- Para comportamientos muy sencillos (1 actuador + 1 sensor) seguramente un sketch simple en Arduino es más que suficiente para asegurar el muestreo constante del sensor y comandar acciones al actuador.

FreeRTOS


RPi
RPi message R
Battery state
Battery
Distances & triggers
Ultrasonics
RPi message T
Control application
Lidar messages
Lidar
GPS messages
GPS
Speed
Angle
10 DOF messages
10 DOF
Ticks
Motor
Servo
Encoder

FreeRTOS

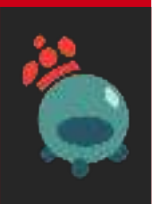
- La versión FreeRTOS para AVR se puede encontrar en:
  - https://github.com/feilipu/Arduino_FreeRTOS_Library
- **FreeRTOS** puede configurarse para diferentes time slices:
  - 15ms, 30ms, 60ms, 120ms, 250ms, 500ms, 1seg y 2seg
  - Si una tarea acaba antes del time-slice, ese tiempo restante es gestionado por el planificador.
- Tareas de la misma prioridad ejecutaran en base al TIME SLICE en round-robin
- Compatible con ATmega328, ATmega1284p, ATmega2560.
- Ejemplos:
  - https://github.com/feilipu/avrfreertos



FreeRTOS

```
#include <Arduino_FreeRTOS.h>

xTaskCreate(MyTask1, "TaskLed1", 100, NULL, 1, NULL);
```

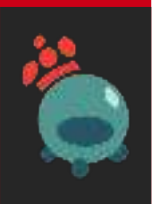
- MyTask1: función que ejecutar la tarea
- “TaskLed1”: Nombre identificativo
- 100: Numero de palabras reservadas en la pila.
- NULL: Parámetros pasados a la función dela tarea
- 1: Prioridad asignada a la tarea
- NULL: puntero opcional para creación de tarea.



FreeRTOS

- Ejemplo de uso de FreeRTOS en Arduino.
- Prioridades en FreeRTOS → [0 – MAX_PRIORITY]
- 3 Tareas:
  - La primera tarea con prioridad 1 enciende el Led-1 durante 100 ms cada 1000ms.
  - La segunda tarea con prioridad 2 enciende el Led-2 durante 800 ms cada 1000ms.
  - La tercera tarea “IDLE” enciende el Led-3 cuando el sistema está ocioso

```
xTaskCreate(MyTask1, "TaskLed1", 100, NULL, 1, NULL);
xTaskCreate(MyTask2, "TaskLed2", 100, NULL, 2, NULL);
xTaskCreate(MyIdleTask, "IdleTask", 100, NULL, 0, NULL);
```

FreeRTOS

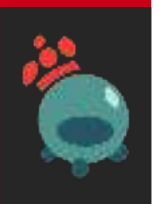

```
static void MyTask1(void* pvParameters)
{
  TickType_t xLastWakeTime, aux;

  while(1)
  {
    xLastWakeTime = xTaskGetTickCount();
    aux = xLastWakeTime;

    digitalWrite(PIN_T1,HIGH);
    digitalWrite(PIN_T2,LOW);
    digitalWrite(PIN_IDLE,LOW);


    while ( (aux - xLastWakeTime)*portTICK_PERIOD_MS < TIME_ON_T1) {
      aux = xTaskGetTickCount();
    }

    Serial.println("Task1");
    xTaskDelayUntil( &xLastWakeTime, ( DELAY_T1 / portTICK_PERIOD_MS ) );

  }
}
```

Si nuestra tarea es periódica debemos programarla como bucle infinito.

Definimos los encendidos y apagados de los LEDS de la aplicación.

Definimos el tiempo que dura el encendido de los LEDS

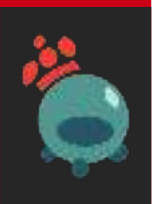
Utilizamos xTaskDelayUntil para definir el delay necesario antes de realizar la siguiente iteración.



FreeRTOS

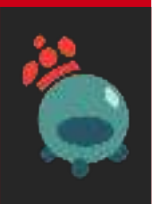
```
// IDLE Task
static void MyIdleTask(void* pvParameters)
{
  while(1)
   {
    digitalWrite(PIN_T1,LOW);
    digitalWrite(PIN_T2,LOW);
    digitalWrite(PIN_IDLE,HIGH);
    Serial.println("Idle state");
    xTaskDelayUntil( &xLastWakeTime, ( PERIODIC_IDLE / portTICK_PERIOD_MS ) );
  }
}
```

Nuestra tarea IDLE solo ejecutará cuando
el MCU esté “ocioso”, esto es, cuando no ejecuta ni la Tarea1 ni la Tarea2



Bibliografía

- FreeRTOS
  - https://es.wikipedia.org/wiki/FreeRTOS
- FreeRTOS documentation
  - https://www.freertos.org/Documentation/RTOS_book.html
- Libro
  - Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC Warren Gay, Apress, 2018.
- Paper:
  - Control System Based on FreeRTOS for Data Acquisition and Distribution on Swarm Robotics Platform



URJC Escuela de Ingeniería de Fuenlabrada

GSyC RoboticsLabURJC

Programming Robot Intelligence