

# $\mu$ Kernel

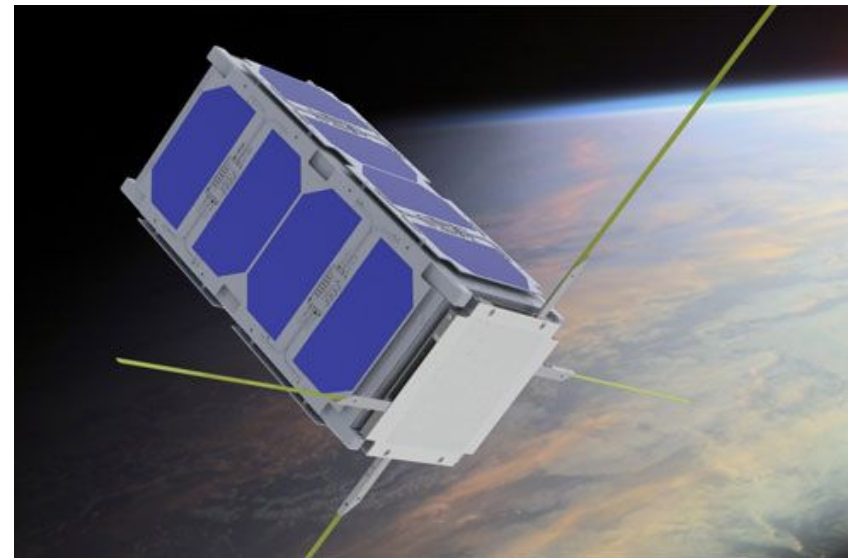
## El sistema operativo de Antelsat

Gustavo De Martino - 28/04/2015

gdemartino@fing.edu.uy

# Temas

- Antelsat
- ¿Porqué usar un OS?
- ¿Porqué no usar un RTOS existente?
- Requerimientos del OS
- Diseño
- Implementación
- Estructuras de datos
- Código



# ¿Qué es Antelsat?

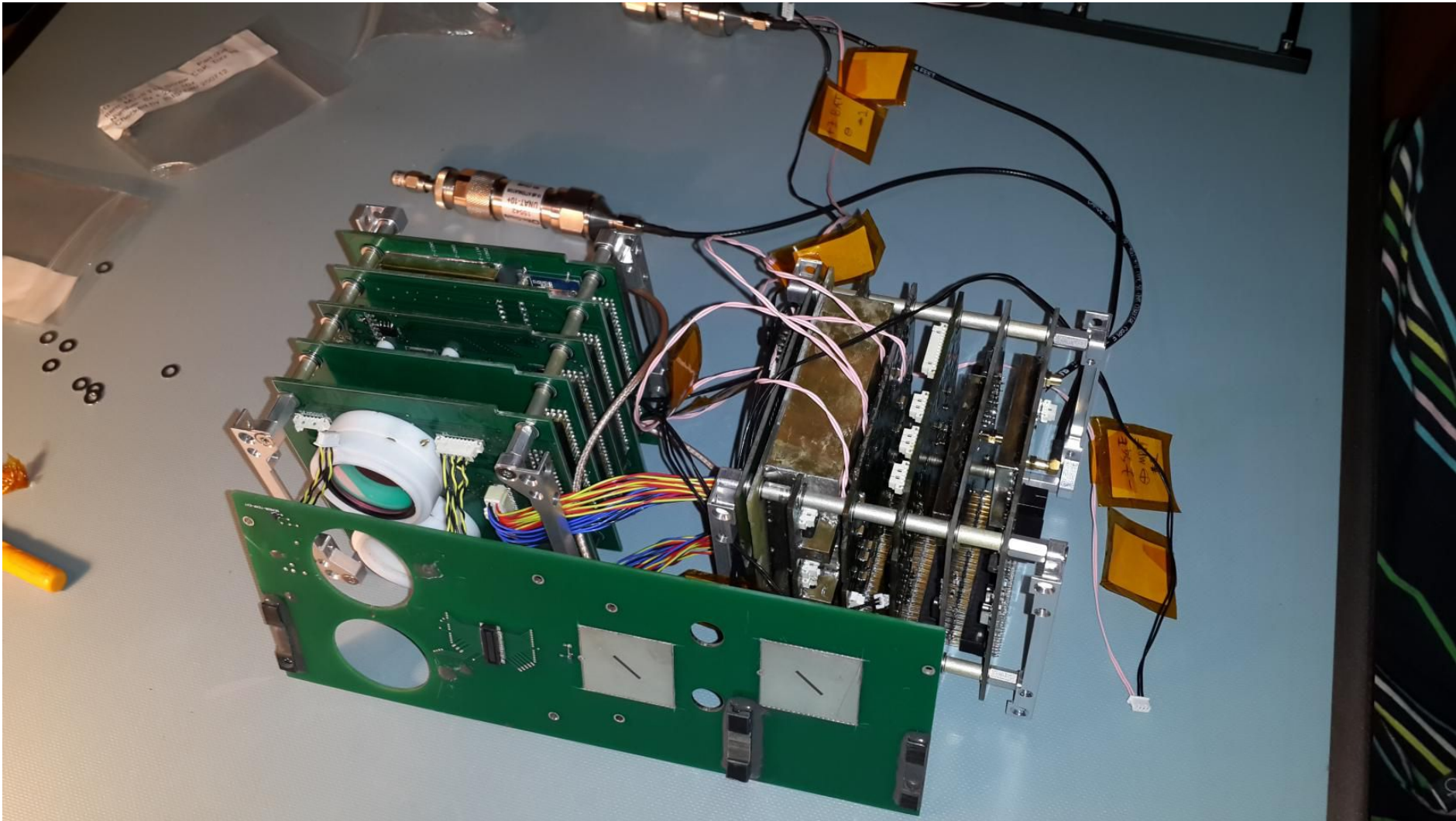
Nano satélite desarrollado según el estándar Cubesat.

Dimensiones 10 x 10 x 20 cm

6 módulos desarrollados por separado

1 bus de comunicaciones I2C

- Energía MSP430F5438A
- Control y registro MSP430F5438A
- Orientación MSP430F5438A + ARM
- Comunicaciones (2) MSP430F6638
- Payload MSP430



# ¿Porqué usar un OS?

Gran cantidad de requisitos independientes

Ejemplo: En EMS

- Maximizar la potencia obtenida de los paneles
- Encender y apagar módulos a demanda
- Generar y transmitir telemetría Morse
- Monitorear el bus de comunicación
- Mantener la hora del sistema
- Enviar y recibir mensajes de otros módulos

Las aplicaciones deben ser simples y fáciles de verificar.

# ¿Porqué no usar un RTOS existente?

**Bajo consumo de energía**

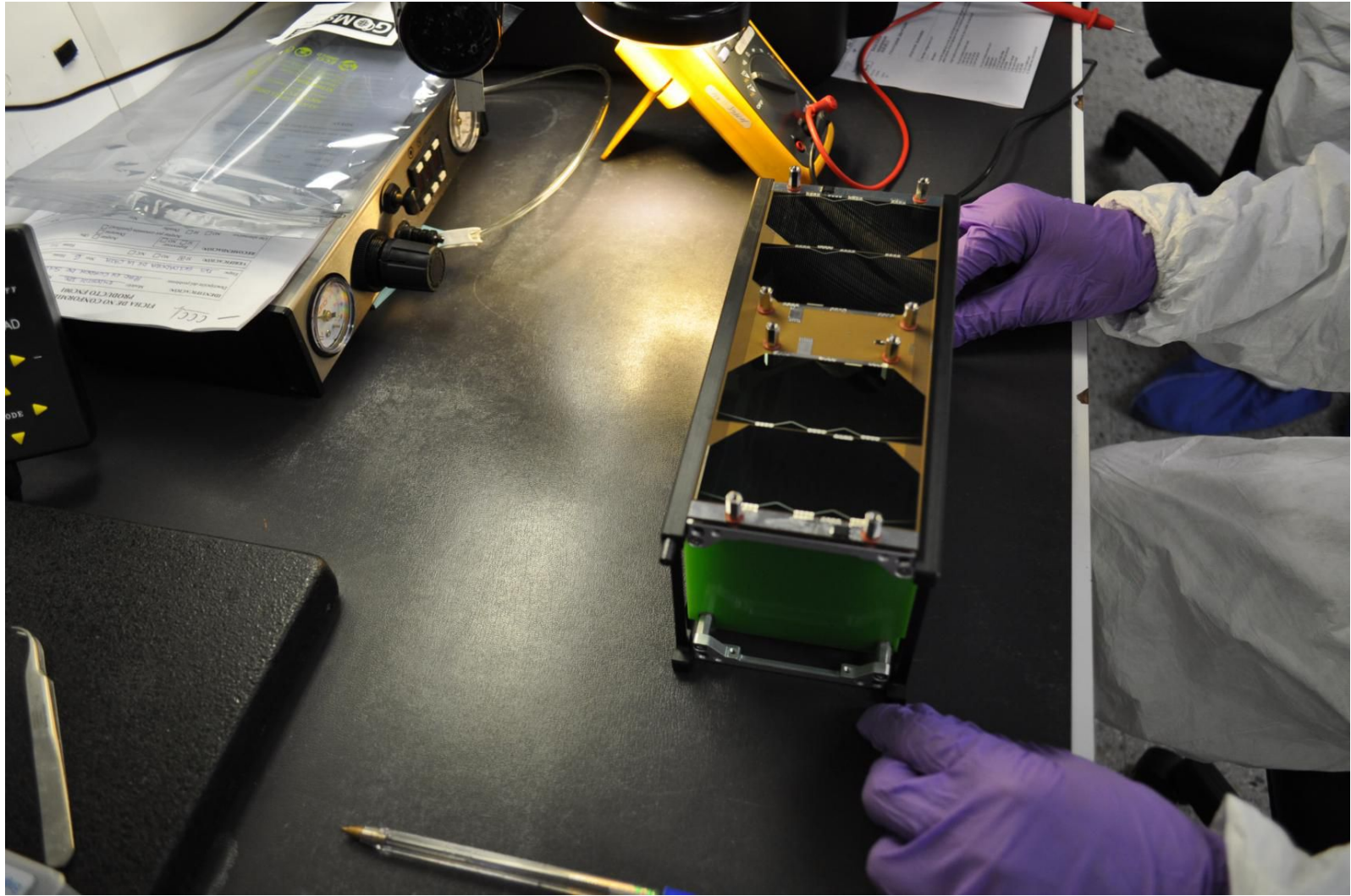
**Tolerancia a fallos**

- Alteraciones del código
- Alteraciones del contenido de la memoria RAM

**Decisión de diseño**

¿Modificar un RTOS existente para cumplir los requisitos o crear un OS tan simple como sea posible para satisfacer las necesidades?





# Requerimientos del OS

- Administración de CPU
- Sincronización de procesos
- Comunicación entre procesos

## ¿Administración de memoria?

- No es posible. No hay MMU en MSP430
- Memoria compartida



# Diseño

## Dos conceptos

- Proceso (en realidad hilos)
- Recurso: Herramienta de sincronización y comunicación. Una mezcla de semáforo y socket
- Planificador colaborativo de prioridad fija.
- Delegación temporal del procesador (IDLE)
- Tiempo máximo de ejecución por proceso (WDT)
- Tiempos del kernel con granularidad de 100 ms (tick)
- Control de CRC sobre datos del proceso
- Control de desbordamiento de stack

# Implementación (visión del proceso)

**Load**      Cargar un proceso en memoria

**Run**        Ejecutar los procesos

**Lock**        Tomar propiedad de un recurso

**Unlock**    Liberar un recurso

**Read**        Recibir datos a través de un recurso

**Write**       Entrega datos a través de un recurso

# Implementación (visión del kernel)

- Load**      Preparar el contexto para un proceso
- Run**      Ejecutar el planificador
- Lock**      Colocar al proceso en la cola de espera por el recurso solicitado hasta que este esté libre.
- Unlock**    Desbloquear al primer proceso esperando por el recurso, marcarlo como libre si la cola está vacía.
- Read**      Bloquear el proceso hasta que se hayan recibido datos a través del recurso.
- Write**      Bloquear el proceso hasta que se puedan entregar los datos. Liberar al propietario del recurso si está bloqueado a la espera de los datos.

# Otras funciones

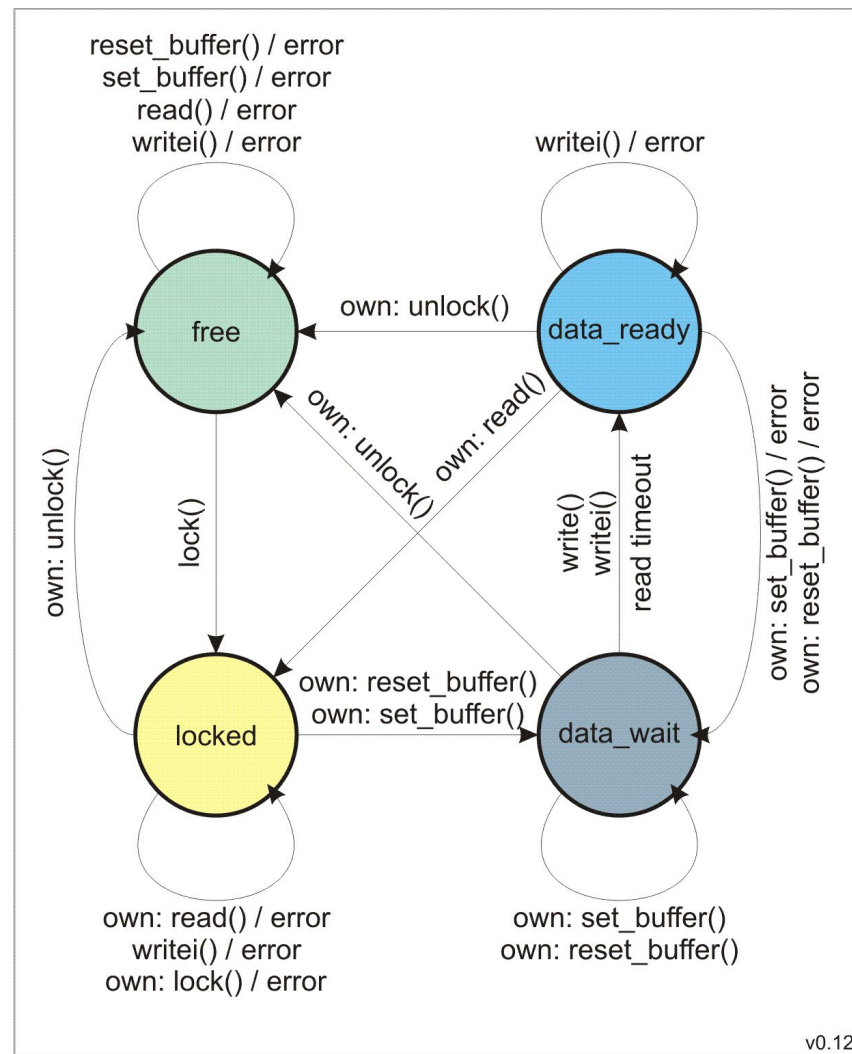
**Kernel\_init**      Inicializa las estructuras de datos

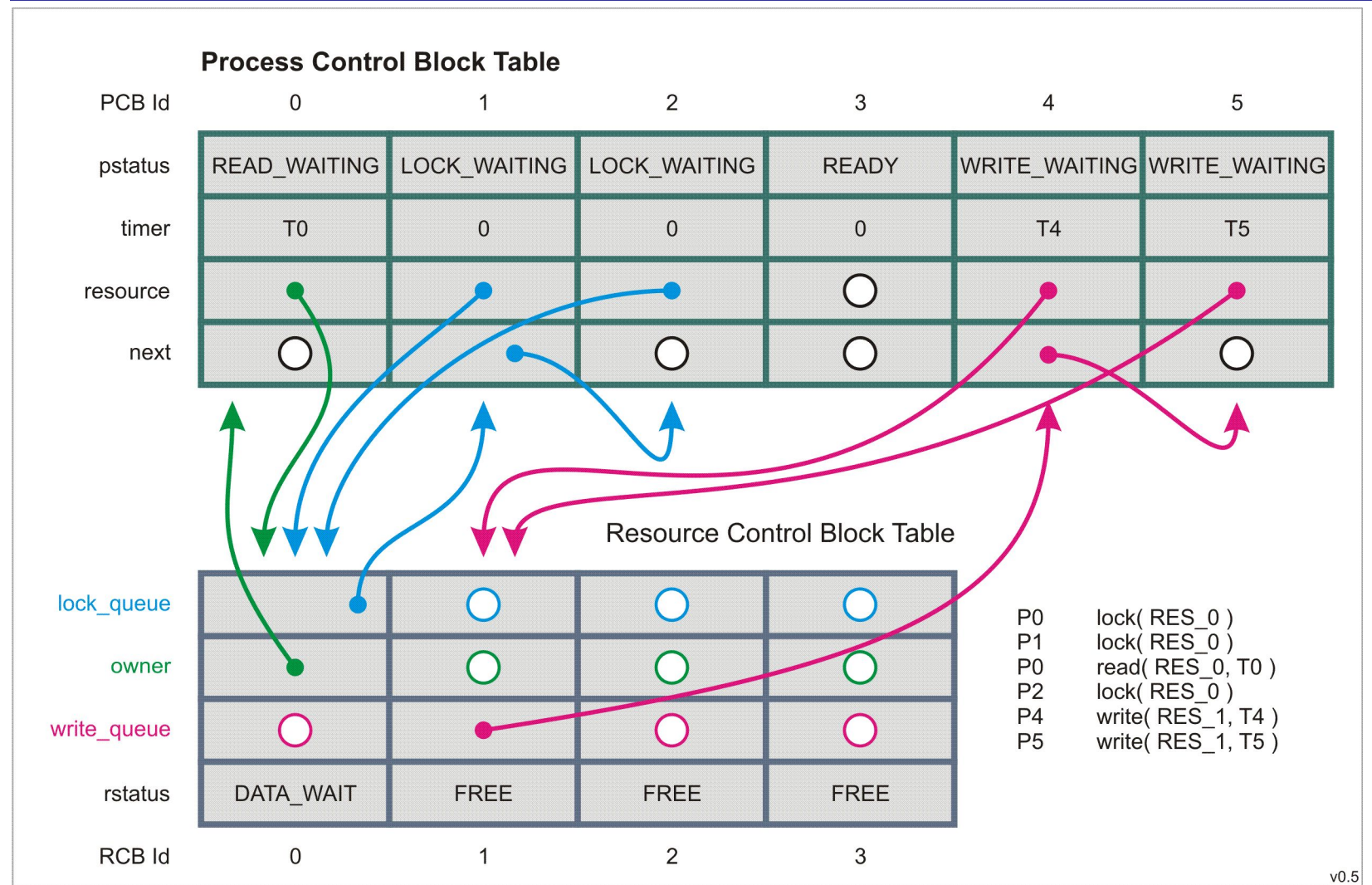
**Set\_buffer**      Define el espacio de transferencia

**Reset\_buffer**    Reinicia el espacio de transferencia

**Writei**            Write no bloqueante para ISR

# Datos de kernel







# Una mirada al código

# Preguntas