

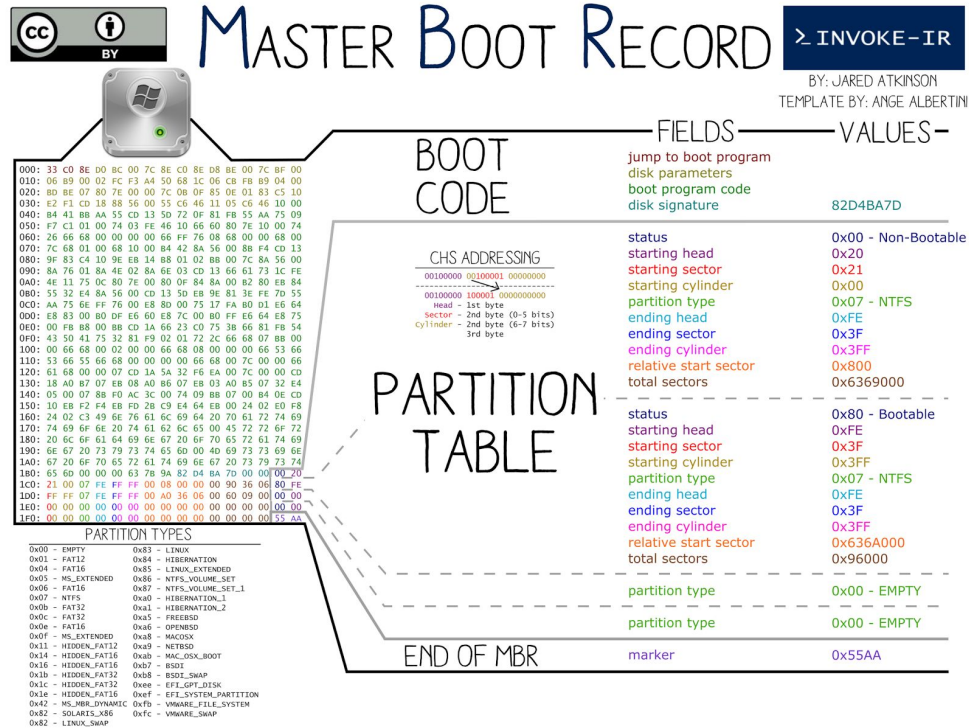


## UEFI, bootloaders & Rust

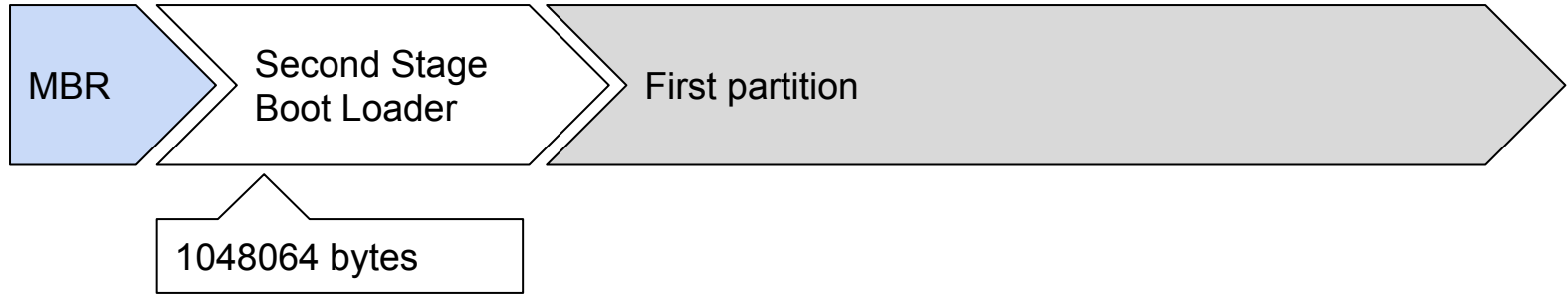
Escribiendo código seguro en el firmware de tu PC

Alberto Ruiz <aruiz@redhat.com>  
Engineering Manager - RHEL & Fedora Bootloader Team  
Red Hat

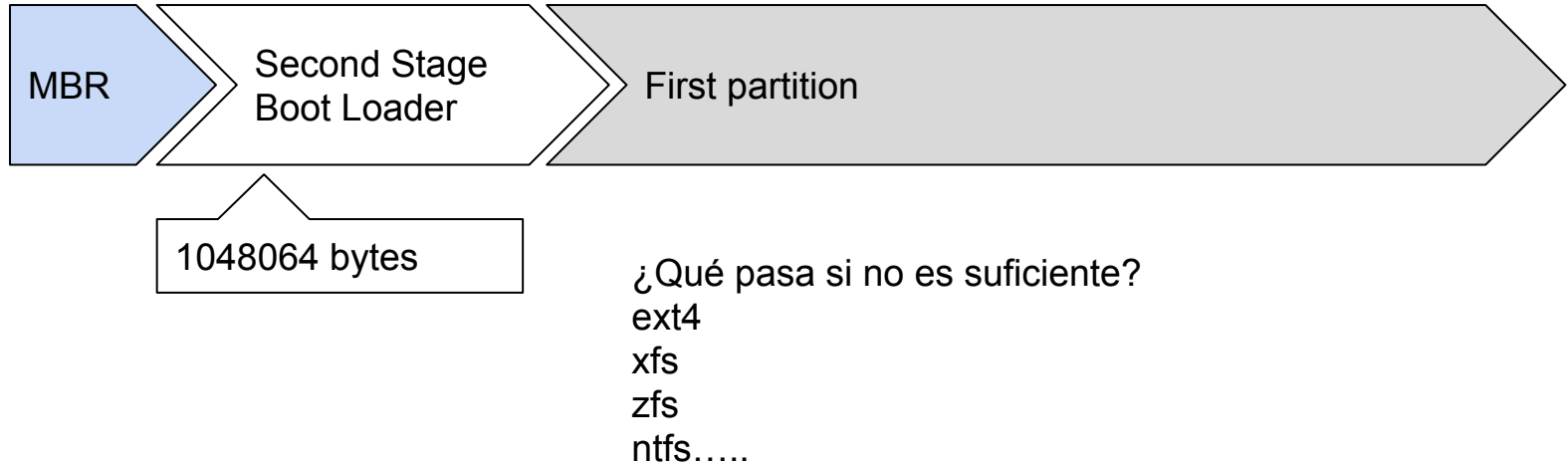
# En el principio fueron la BIOS y el MBR



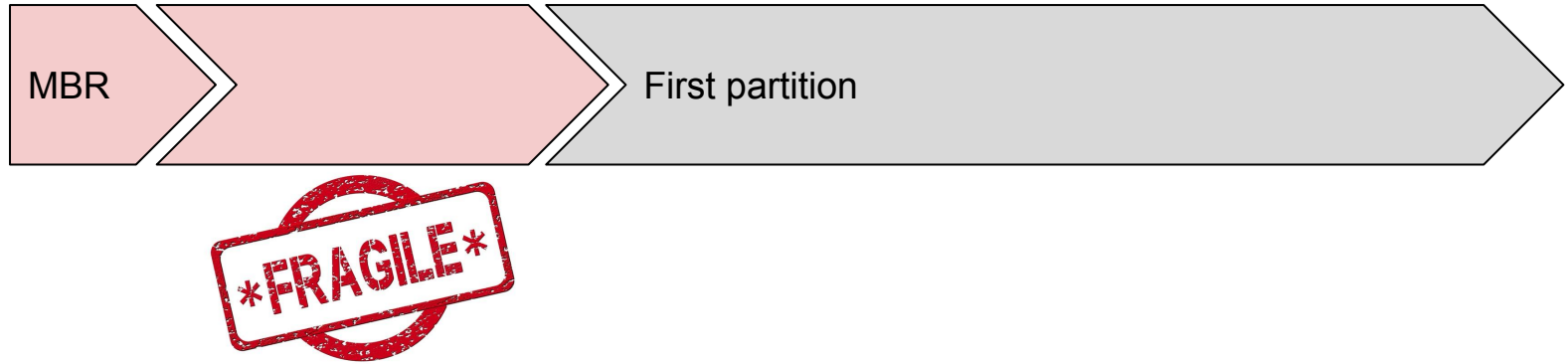
# Leer el sistema de ficheros en menos de 512bytes de código ¿cómo?



# Leer el sistema de ficheros en menos de 512bytes de código ¿cómo?



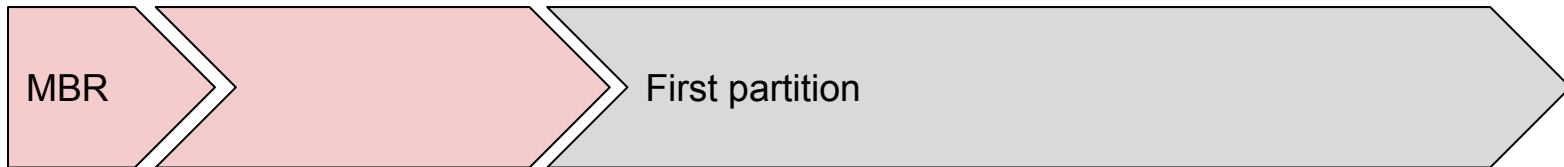
# Problemas: actualizaciones y fallos de seguridad



# Problemas: conectividad/reseteo de fábrica



# Problemas: prevención de sabotaje



# Problemas: crypto

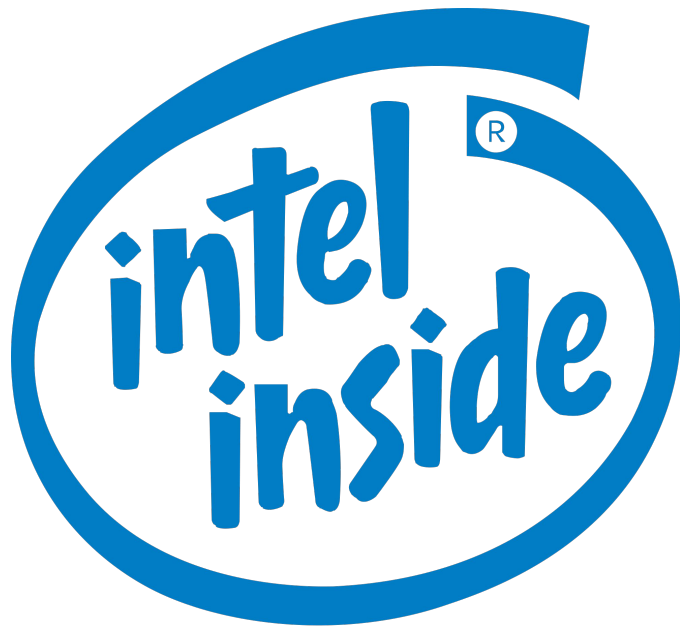




# ¡Una solución quiero!

- Seguridad
- Fiabilidad
- Extensibilidad

# Lo único bueno que nos dejó Itanium



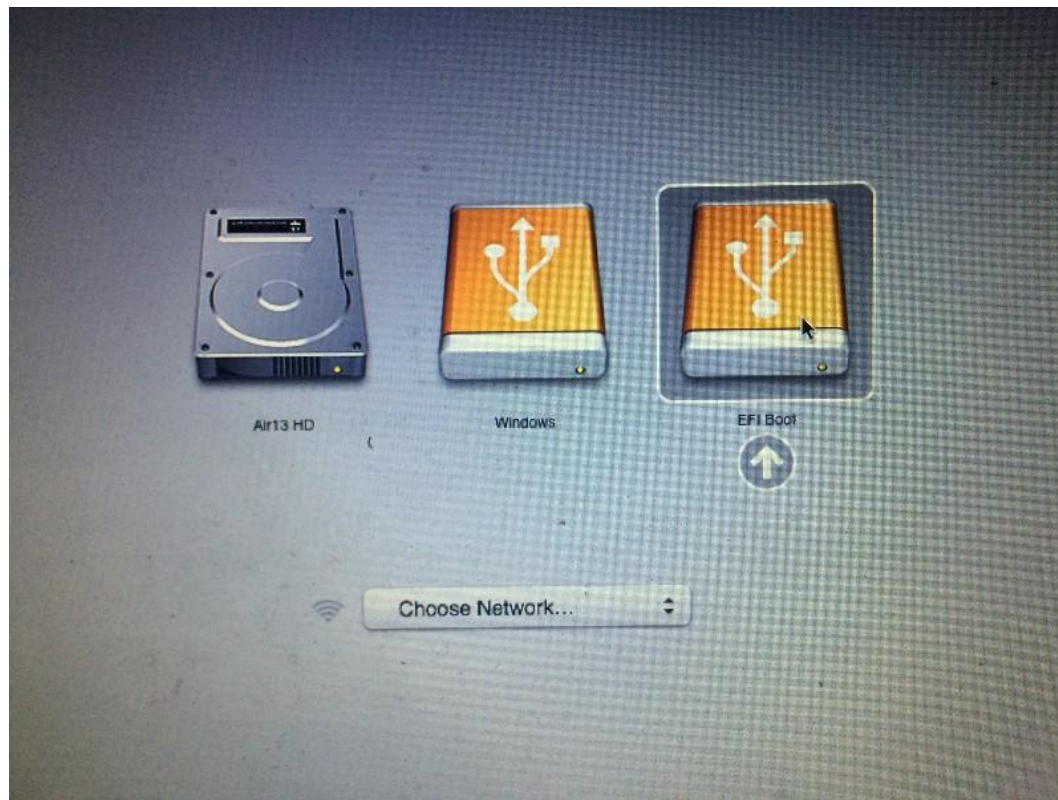
# Unified Extensible Firmware Interface



# UEFI: el conceto



UEFI es una **especificación** de un sistema operativo capaz de inicializar una plataforma de hardware, ejecutar drivers, servicios y programas con el objetivo principal de arrancar otros sistemas operativos y restaurar o verificar el sistema



# UEFI: implementación



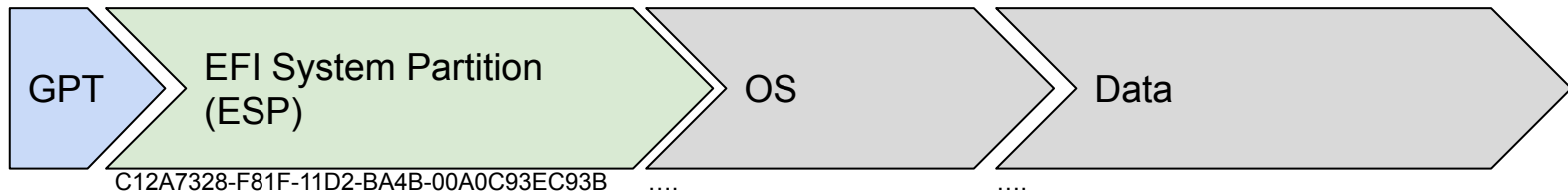
```
QEMU

Boot Failed. EFI DVD/CDROM
Boot Failed. EFI Floppy
Boot Failed. EFI Floppy 1
EFI Shell version 2.30 [1.0]
Current running mode 1.1.2
Device mapping table
  blk0 :Floppy - Alias (null)
        PciRoot (0x0) /Pci (0x1,0x0) /Floppy (0x0)
  blk1 :Floppy - Alias (null)
        PciRoot (0x0) /Pci (0x1,0x0) /Floppy (0x1)
  blk2 :BlockDevice - Alias (null)
        PciRoot (0x0) /Pci (0x1,0x1) /Ata (Secondary,Master,0x0)

Press ESC in 1 seconds to skip startup.nsh, any other key to continue.
Shell> _
```

# UEFI: Cómo funciona

Formato FAT32, >500MiB



```
EFI
├── BOOT
│   ├── BOOTX64.EFI
│   └── fbx64.efi
├── fedora
│   ├── BOOTX64.CSV
│   ├── fw
│   ├── fwupx64.efi
│   ├── grub.cfg
│   └── grub.cfg.bak
```

```
$ efibootmgr -v
```

```
BootCurrent: 0029
Timeout: 0 seconds
BootOrder: 0029,0015
Boot0029* Fedora
HD(1,GPT,0bd04bfa-51ca-4366-acf2-1ca4be877300,0x800,0xf9800)/F
ile(\EFI\fedora\shimx64.efi)
Boot002A* Windows
HD(1,GPT,0bd04bfa-51ca-4366-acf2-1ca4be877300,0x800,0xf9800)/F
ile(\EFI\Microsoft\Boot\bootmgfw.efi)
```

# UEFI: API

## Boot Services:

Servicios que se pueden consumir en la fase de arranque

## Runtime Services:

Servicios que puede ejecutar el Sistema Operativo tras el arranque (leer variables, usar el framebuffer...)



# UEFI: Portable Executable Format


- Los binarios .EFI usan el formato de ejecutables de Microsoft Windows (.EXE de toda la vida).
- Estos binarios pueden ejecutar operaciones o añadir nuevos servicios durante la fase de arranque (drivers de dispositivos o sistemas de ficheros...).

# UEFI: Ventajas

- **Unifica arquitecturas** (ARM32/64...)
- Actualizaciones de Firmware seguras sin MSDOS!
- Secure Boot (requiere que MS firme tus binarios EFI)
- Configurar la BIOS con gráficos y ratón
- Arrancar un sistema operativo desde la red, incluso con WiFi

# UEFI: Como emitir un .EFI desde Rust

- `#[no_std]`
- `xcargo`
- `nightly`

 README.md

## uefi-rs

build error

### Description

UEFI is the successor to the BIOS. It provides an early boot environment for OS loaders, hypervisors and other low-level applications. While it started out as x86-specific, it has been adopted on other platforms, such as ARM.

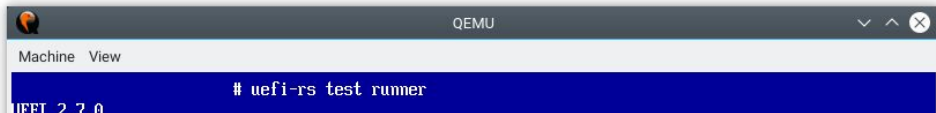
This crate makes it easy to both:

- Write UEFI applications in Rust (via the `x86_64-unknown-uefi` target)
- Call UEFI functions from an OS (usually built with a [custom target](#))

The objective is to provide **safe** and **performant** wrappers for UEFI interfaces, and allow developers to write idiomatic Rust code.

Check out @gilomendes [blog post on getting started with UEFI in Rust](#).

**Note:** due to some issues with the Rust compiler, this crate currently works and has been tested *only* with **64-bit** UEFI.



QEMU

Machine View

```
# uefi-rs test runner
UEFI 2.7.0
```

# UEFI: punto de entrada

```
#![no_std]
#![no_main]
extern crate uefi;
use uefi::table::{SystemTable, Boot};

#[no_mangle]
pub extern "win64" fn efi_main(_handle: uefi::Handle,
                               _system_table: SystemTable<Boot>) -> ! {
    loop {}
}
```

# DEMO



# GRACIAS

- [twitter.com/acruiz](https://twitter.com/acruiz)
- <https://github.com/rust-osdev/uefi-rs>
- <https://github.com/aruiz/madrust-uefi-skeleton>