UEFI Boot Process

By: Connor Glosner

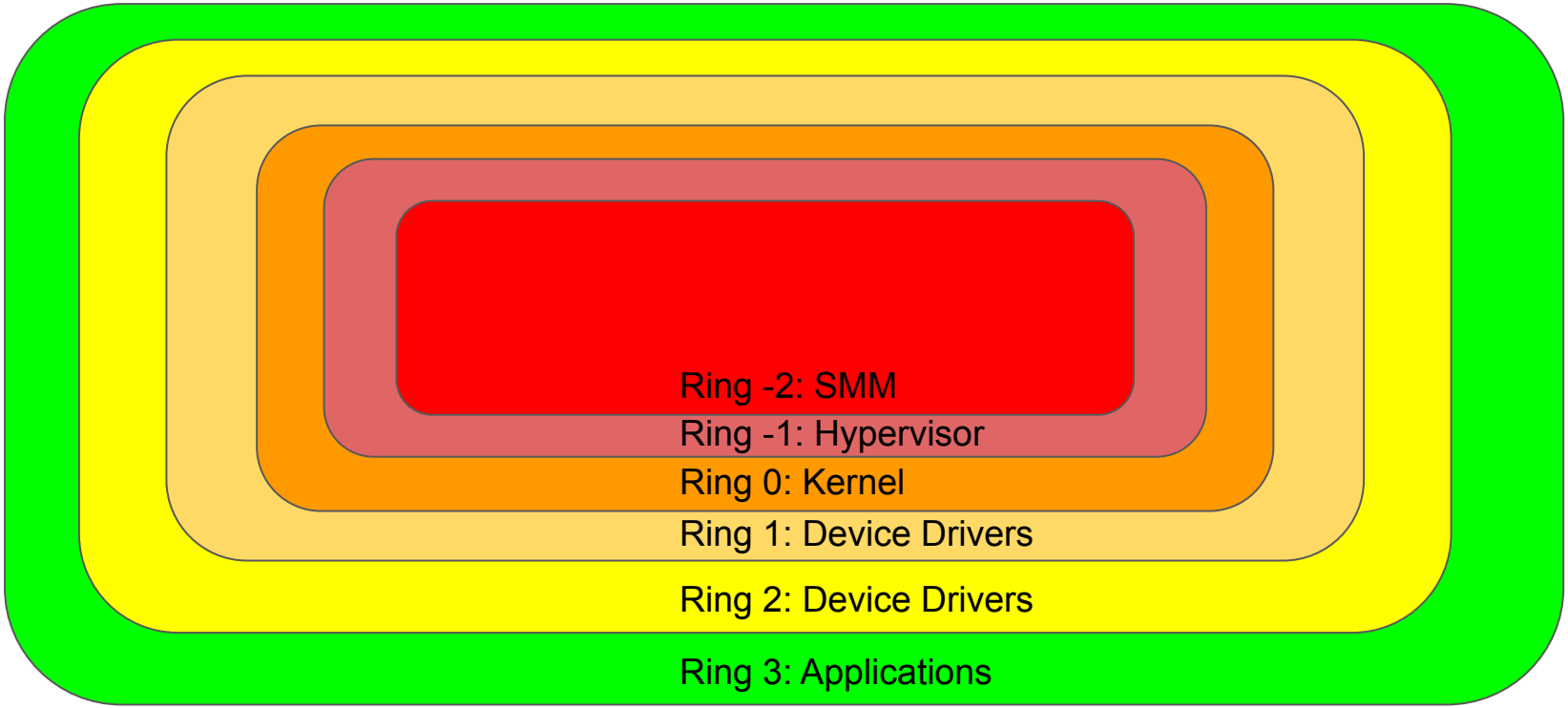
Overview

- Terms
- Legacy Boot Process
- Improvements
  - Hypervisor-Enforced Code Integrity Protection
  - Virtual Secure Mode (VSM)
  - Device Guard
  - Secure boot
- UEFI Boot Process
- UEFI Boot Walkthrough

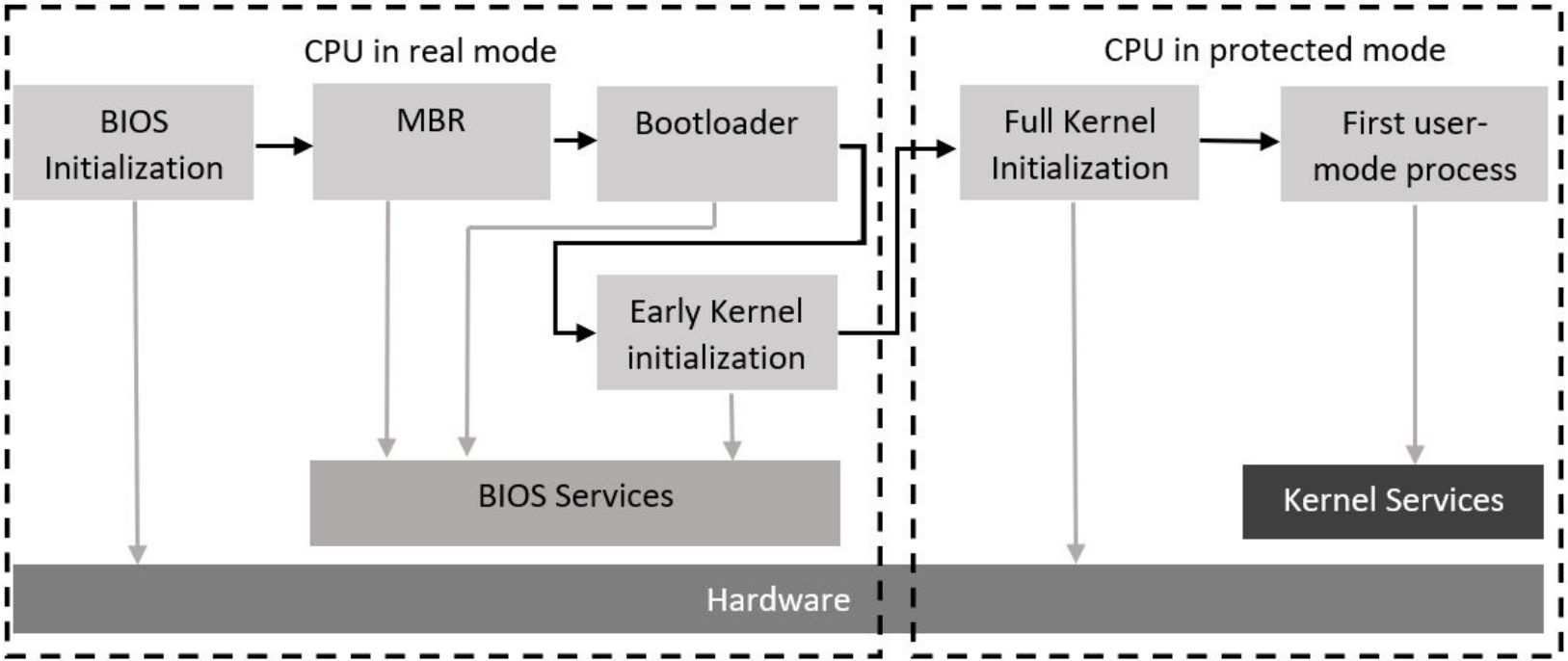
Terms

- Real Mode: Stands for real address mode and is when addresses always correspond to real locations in the memory.
- Protected Mode: Utilizes virtual memory addressing to isolate memory during different processes.
- Virtual Memory Addressing: Creates a virtual memory space that will extend the physical memory space by not requiring sequential memory addresses and allowing the use of the a disk. (requires the use of a memory management unit to translate virtual address to physical ones)

Privilege Levels



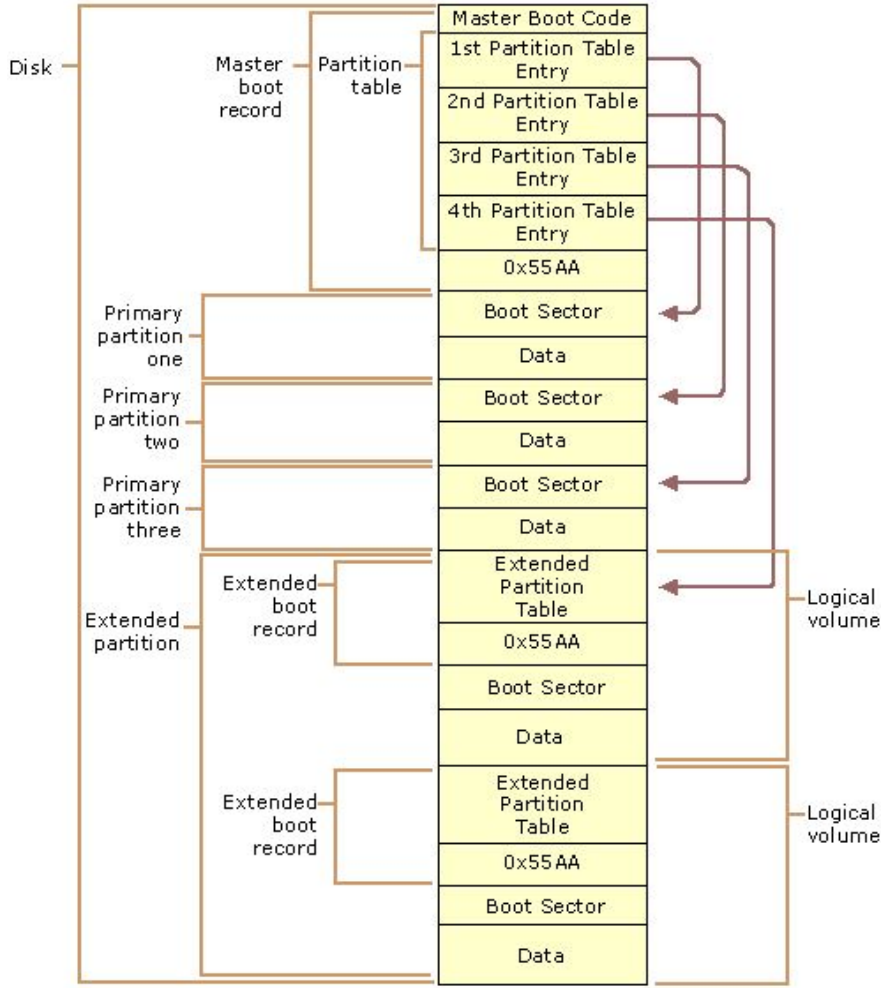
Legacy Boot Process (BIOS)



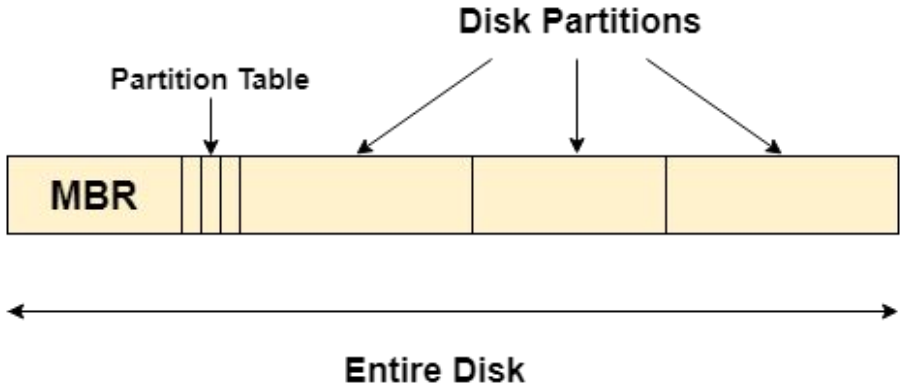
BIOS Initialization

- Basic I/O controls for hardware are initialized.
  - Ex. disk service
- Disk service has a known weakness in the access handler called interrupt 13h (INT 13h):
  - Allows arbitrary read and write permissions to any disk on the system.
  - Added an infected instruction at the end of the interrupt memory.
  - Prevented by added write protection to the MBR.

Disk Layout

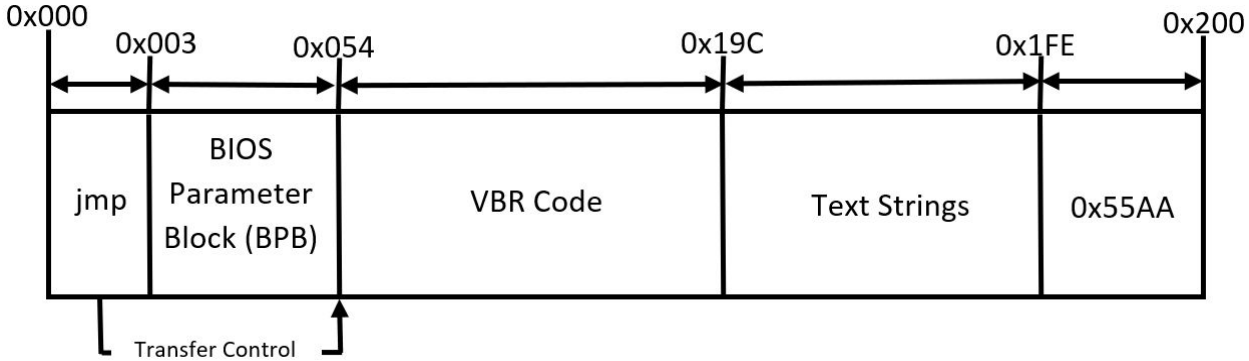

Disk
Master boot record
Partition table
Master Boot Code
1st Partition Table Entry
2nd Partition Table Entry
3rd Partition Table Entry
4th Partition Table Entry
0x55AA
Primary partition one
Boot Sector
Data
Primary partition two
Boot Sector
Data
Primary partition three
Boot Sector
Data
Extended partition
Extended boot record
Extended Partition Table
0x55AA
Boot Sector
Data
Logical volume
Extended boot record
Extended Partition Table
0x55AA
Boot Sector
Data
Logical volume

Master Boot Record (MBR)



- The MBR is only a data structure.
- It only contains 512 bytes and 446 of those are for the boot code itself.
- It is limited to 4 partitions.
- Its goal is to locate the active partition and pass control to the Volume Boot Record.

Volume Boot Record

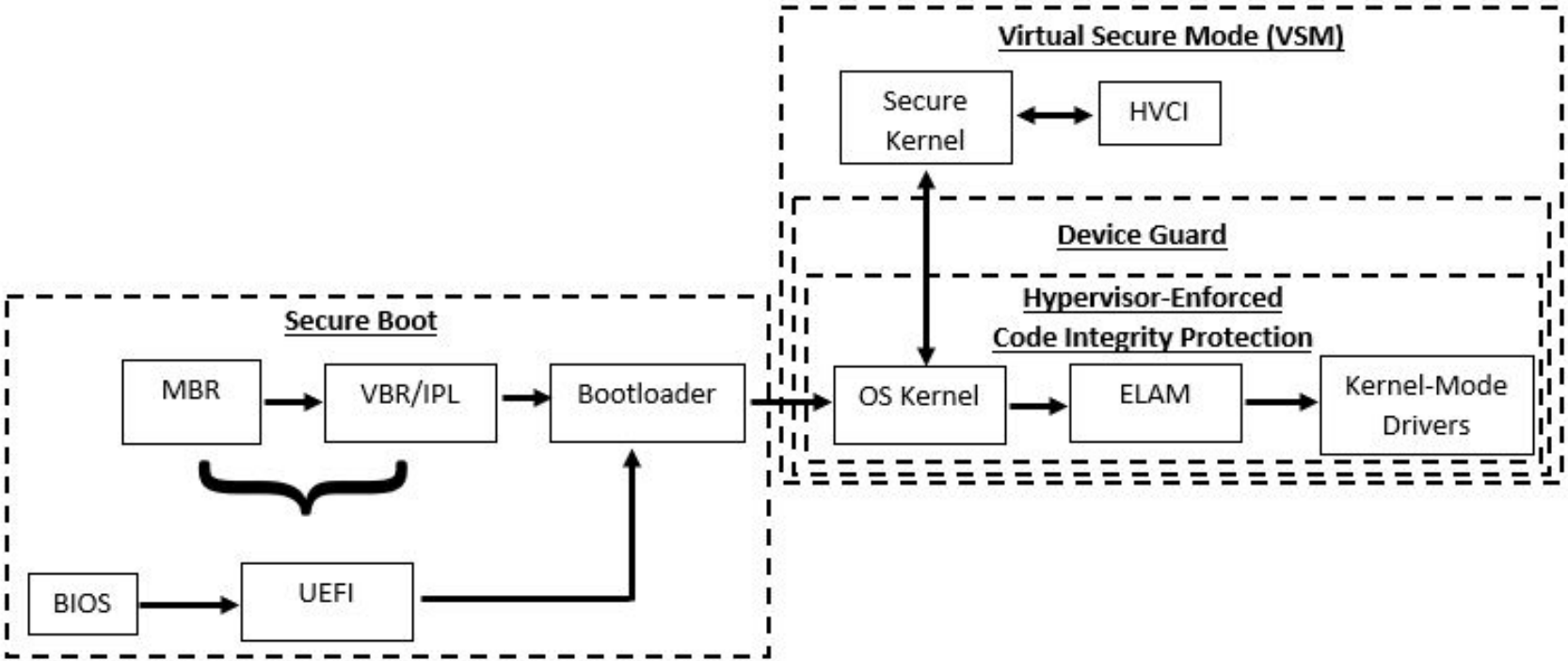


- Uses a FAT filesystem type.
- Located on the Disk
- Text strings holds error messages is an error occurs during loading.
- The IPL is stored in 15 consecutive sectors of 512 bytes.
    - The IPL is just another stage to load the boot manager (bootmgr - Windows, GRUB - Linux)

BIOS Services

- Are used to create a layer of abstraction between the hardware and the bootcode.
- Operate similarly to boot services in UEFI.
- Manufacturer specific, so each implementation is unique.

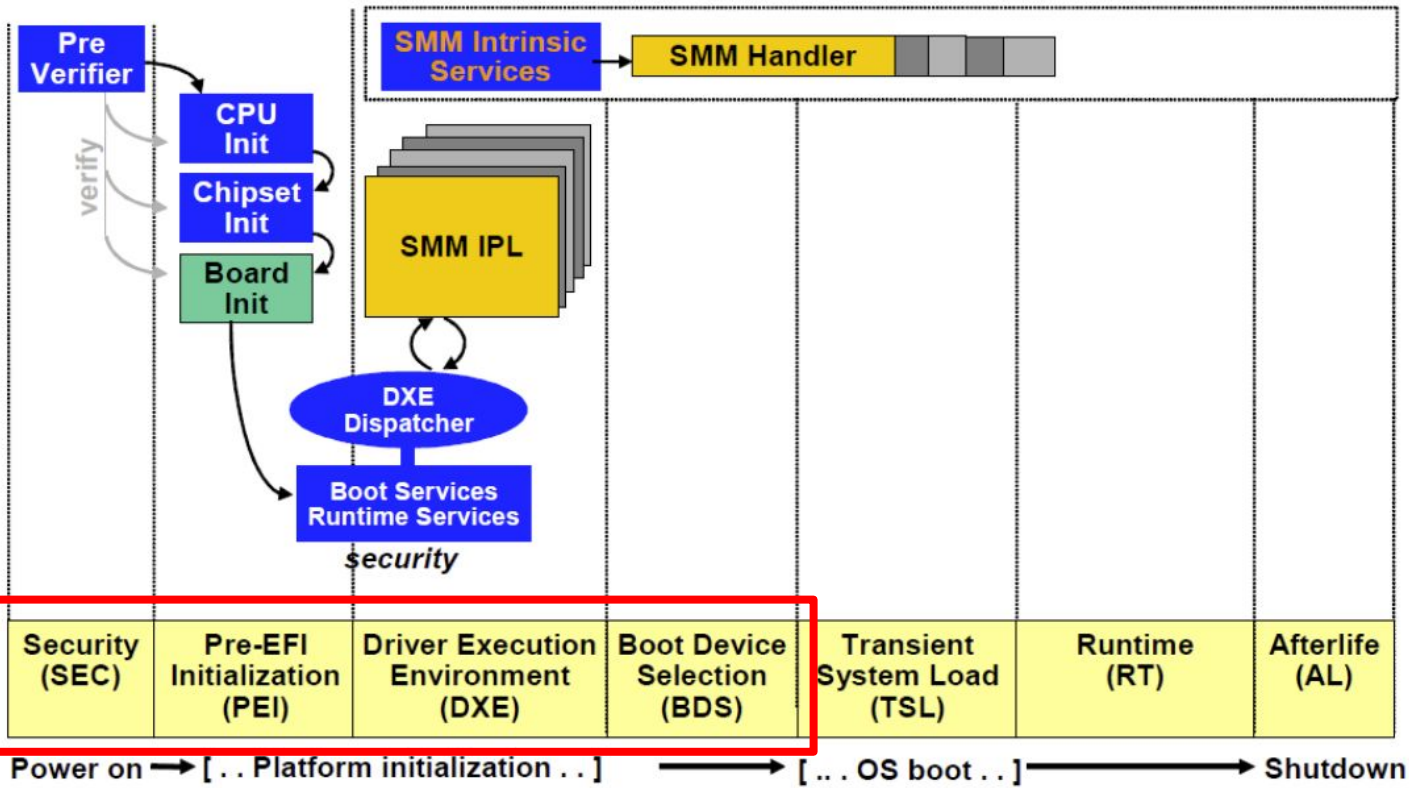
Security Improvements



Secure Boot

- It was designed to utilize UEFI to block drivers and applications that don’t have a valid digital signature during the boot process.
- It verifies the integrity of all UEFI and OS boot files before they are loaded and executed.
- It is an iterative process, so it was built using a root-of-trust methodology.
- Verification is used on 3rd party firmware code (i.e. not the manufacturers code). (PEI Example)

4 Phases of UEFI



Security (SEC)

- Executes hardware specific firmware.
  - Written in assembly.
- Creates the foundation for the root-of-trust methodology.
- Creates temporary memory using CPU caches.
- Locates the loader on the SPI flash for the PEI phase.

Pre-Environment Initialization (PEI)

- The boot code is loaded from the SPI flash in this phase.
- It initializes the permanent memory, but until then everything is executed in the CPU cache. (InitializeMemory)(FreeTemperaryMemory)
- This is where the runtime and boot services begin execution. (InitialzeDXE -> DXELoadCore)

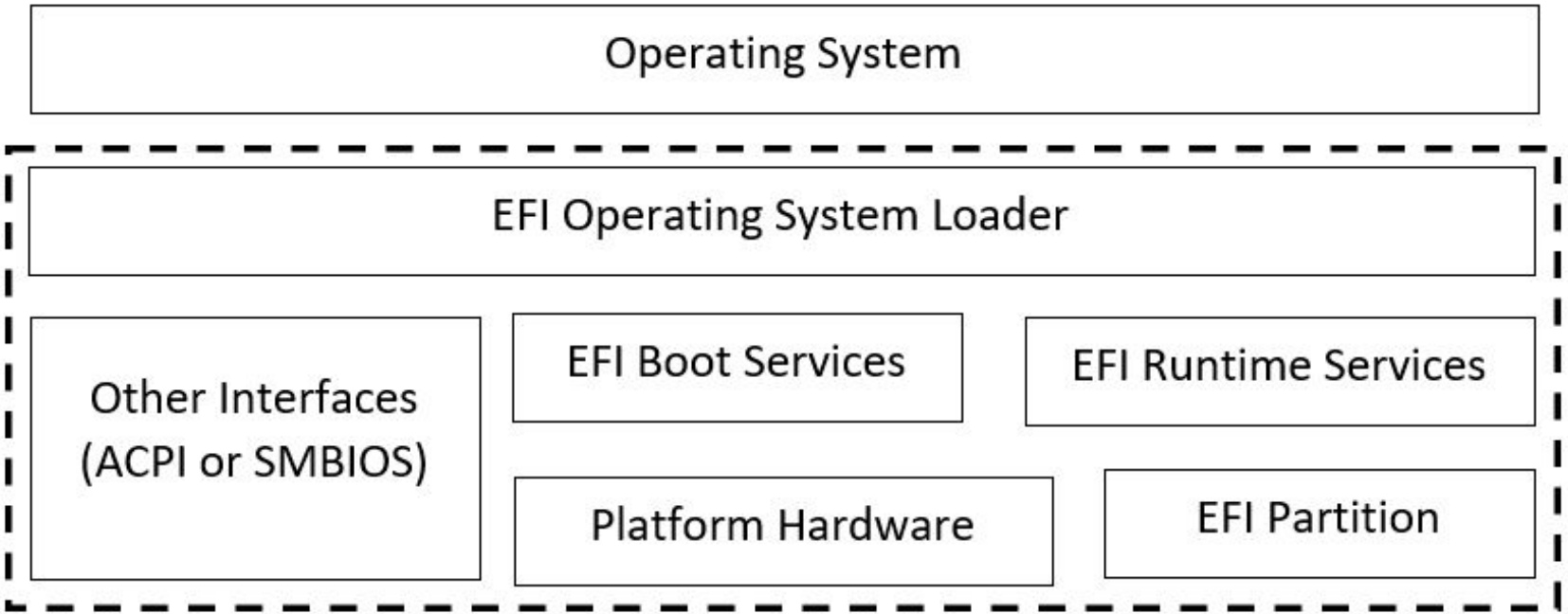
Driver Execution Environment (DXE)

- This is the main phase of the boot process. (EntryPoint)
- The System Management Mode (SMM) is initialized during this phase.
- SMM is executed in Ring -2, while everything else is in Ring 0.
- The boot and runtime services finish initialization during this phase.
- All images are loaded:
  - Driver - permanent
  - Application - temporary
- The images are located with the NVRAM variable and executed.
- The order that they are loaded in is also stored in the NVRAM.

Boot Device Selection (BSD)

- This is when the boot partition is selected.
- It is either defaulted to the active partition or will allow an option if there are multiple operating systems present.
- It will also handle executing the boot manager and OS drivers from the system partition.

Boot Systems Overview



Part 1: Platform Hardware

- This would be apart of the SEC phase.
- Hardware (CPU, Motherboard) is initialized.
- This is hardware specific, so it differs between architectures and doesn't have a standardized specification.
- This part does not initialize the memory controller.
- Other hardware is initialized later, but only system critical hardware is initialized first.

Part 2: Initializations

- This is apart of the Pre-Initialization Phase. (NVRAM)
- This is where the UEFI firmware begins to be loaded and run from memory.
- The UEFI firmware is stored in the SPI flash of on the device.
- The firmware supports 32 and 64-bit systems.
- Order of initializations is determined by the NVRAM boot order. (BootOrderApp)

```
// PE32+ Subsystem type for EFI images
#define EFI_IMAGE_SUBSYSTEM_EFI_APPLICATION          10
#define EFI_IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER  11
#define EFI_IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER       12

// PE32+ Machine type for EFI images
#define EFI_IMAGE_MACHINE_IA32                       0x014c
#define EFI_IMAGE_MACHINE_IA64                       0x0200
#define EFI_IMAGE_MACHINE_EBC                        0x0EBC
#define EFI_IMAGE_MACHINE_x64                        0x8664
#define EFI_IMAGE_MACHINE_ARMTHUMB_MIXED             0x01C2
#define EFI_IMAGE_MACHINE_AARCH64                    0xAA64
#define EFI_IMAGE_MACHINE_RISCV32                    0x5032
#define EFI_IMAGE_MACHINE_RISCV64                    0x5064
#define EFI_IMAGE_MACHINE_RISCV128                   0x5128
#define EFI_IMAGE_MACHINE_LOONGARCH32                0x6232
#define EFI_IMAGE_MACHINE_LOONGARCH64                0x6264
```

What is the NVRAM?

- The NVRAM stores the UEFI variable. (BootOrderApp)
- The UEFI variable contains all of the variables and parameters needed throughout the boot process (Definition):

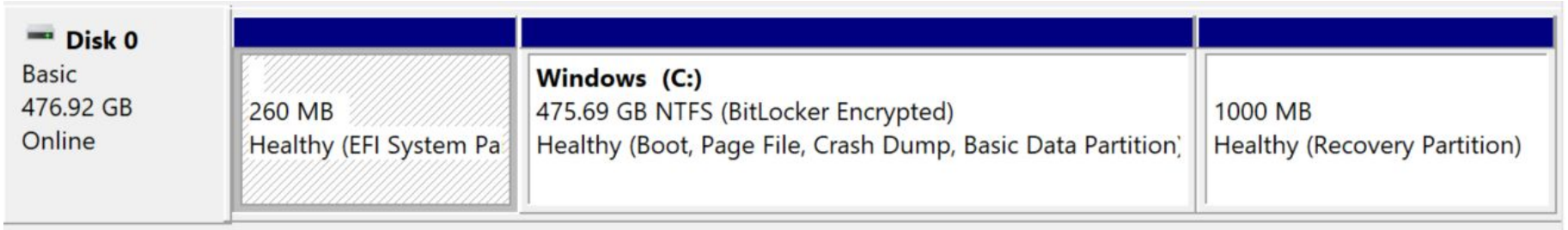
BootOrder	An in-order array of 16-bit integers that refer to the boot order.
Boot####	One of the devices that is to be booted and the #### refers to the hex identification number.
DriverOrder	An in-order array of 16-bit integers that refer to the driver order.
Driver####	A driver that is to be loaded and the #### refers to the hex identification number.

Other Interfaces

- These are the other peripherals connected to the system
  - GPU
  - CD ROM
  - USB Devices
  - Advanced Configuration and Power Interface (ACPI) (ACPI Module)
  - Etc.
- They are initialized through drivers that are operating system specific and are stored in the system partition.

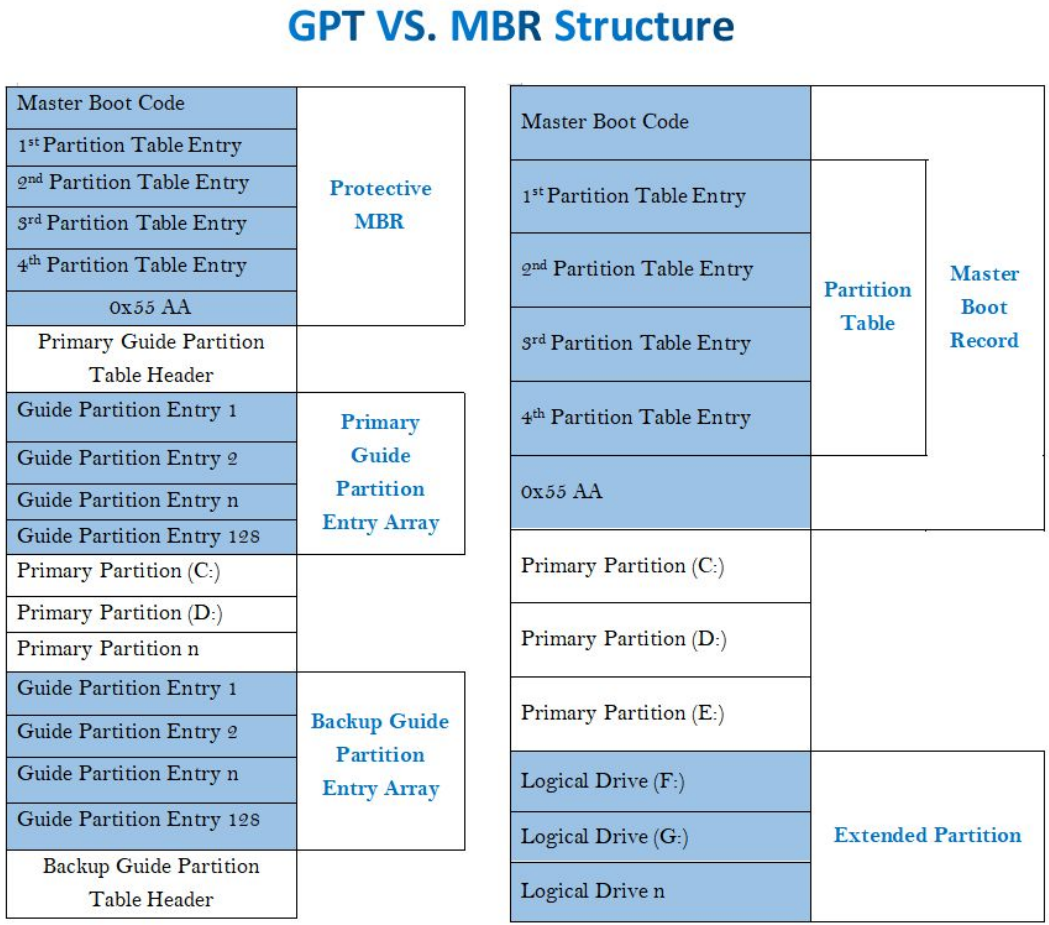
Part 3: EFI System Partition

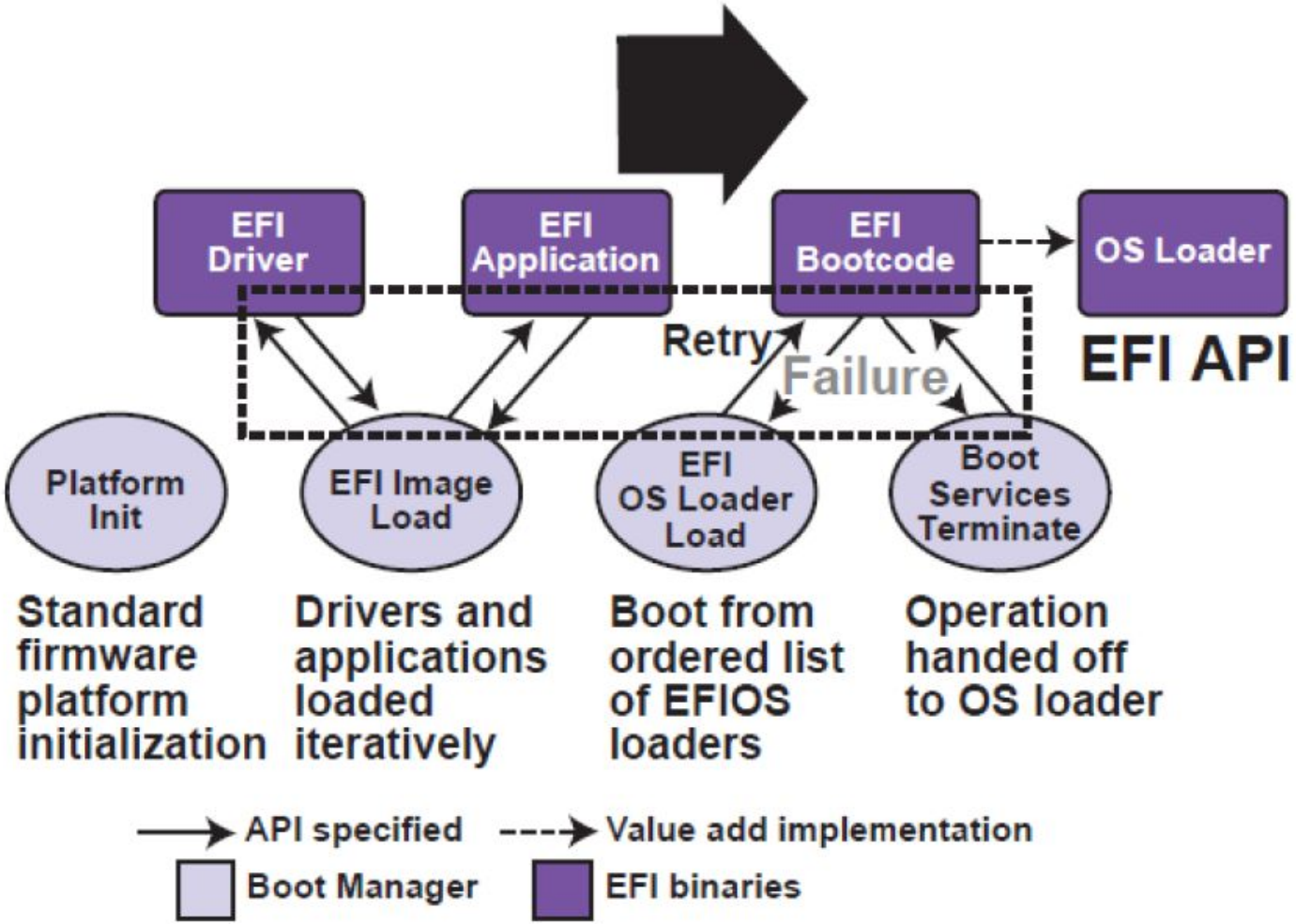
- Specially formatted partition (FAT format) that stores the UEFI OS loader.
- There is an option menu in the case of a different boot option (BootOrderApp):
  - USB drive
  - Network (HTTPS)
  - Etc.
- The structure of the system partition is managed by the GUID Partition Table (GPT).



GPT vs. MBR

- GPT can support a much larger number of partitions.
- Utilizes a 16-byte identification number.
- System partition path is stored in the NVRAM.




EFI Driver
EFI Application
EFI Bootcode
OS Loader
EFI API
Retry
Failure
Platform Init
EFI Image Load
EFI OS Loader Load
Boot Services Terminate
Standard firmware platform initialization
Drivers and applications loaded iteratively
Boot from ordered list of EFIOS loaders
Operation handed off to OS loader
API specified
Value add implementation
Boot Manager
EFI binaries

Part 4: UEFI Services

- This is an important component of the boot process.
- It consists of two components:
  - Boot Services
  - Runtime Services
- The Boot Services run in physical addressing mode while runtime services run in both physical and virtual addressing.
- These services begin initialization in the PEI phase when the permanent memory is established, but the initialization finishes during the DXE phase.

Boot Services

- Boot services are used to create, manage, and stop events during the boot process (All Services):
  - Protocol services
  - Device Protocols - how to communicate between different peripherals
  - Device handle-based boot services
  - Global boot service interface
- These services are important for communicating between drivers.
- CopyMem, which is used when copying the drivers into permanent memory or into the SMRAM is a common example.
- Primarily needed for setting everything up for the OS loader.
- They are terminated when ExitBootServices() is called in the OS Loader.
- An example is when a driver needs to connect to a hardware controller.

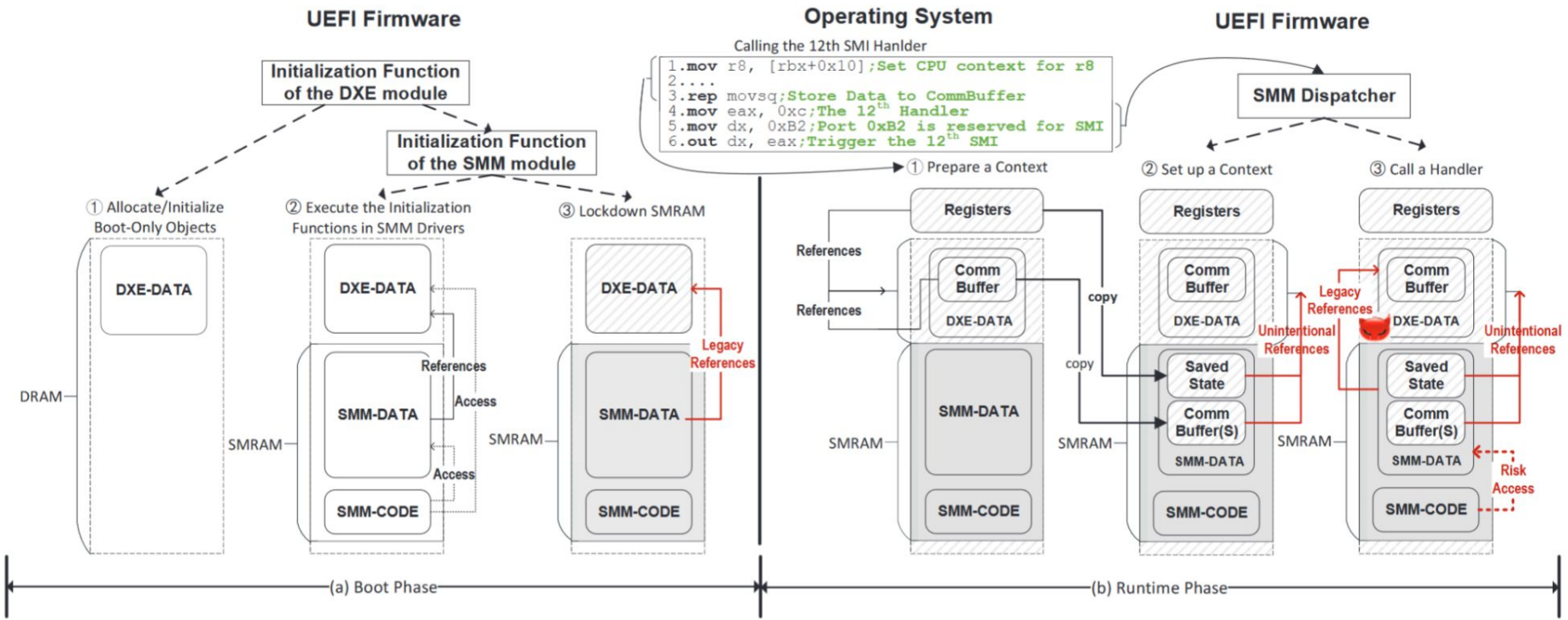
Runtime Services

- These are system call functions that create some abstraction between the kernel and the hardware.
- The service calls don’t require interrupts to be called but do use them by default.
- The memory where the runtime services are stored can’t be modified by the kernel because they interact with the hardware.
- Part of the Runtime code is stored in the SMRAM, the part pertaining to the direct hardware modification.
- The function SMMLoadImage is used to load images into SMRAM.
- RuntimeMain -> ConvertPointer

Runtime Services

- GetTime
- SetTime
- GetWakeupTime
- SetWakeupTime
- SetVirtualAddressMap
- ConvertPointer
- GetVariable
- GetNextVariableName
- SetVariable
- GetNextHighMonotonicCount
- ResetSystem
- UpdateCapsule
- QueryCapsuleCapabilities
- QueryVariableInfo

SMM



[SMT Copy](#) [LoadSMMtoSMRAM](#) [SMM IPL](#) [SMM Main](#)

Part 5: OS Loader

- The OS loader is written by the OS developer.
- It is only used to initialize the OS.
- This is where the boot services memory is freed, while the runtime services continue to run.

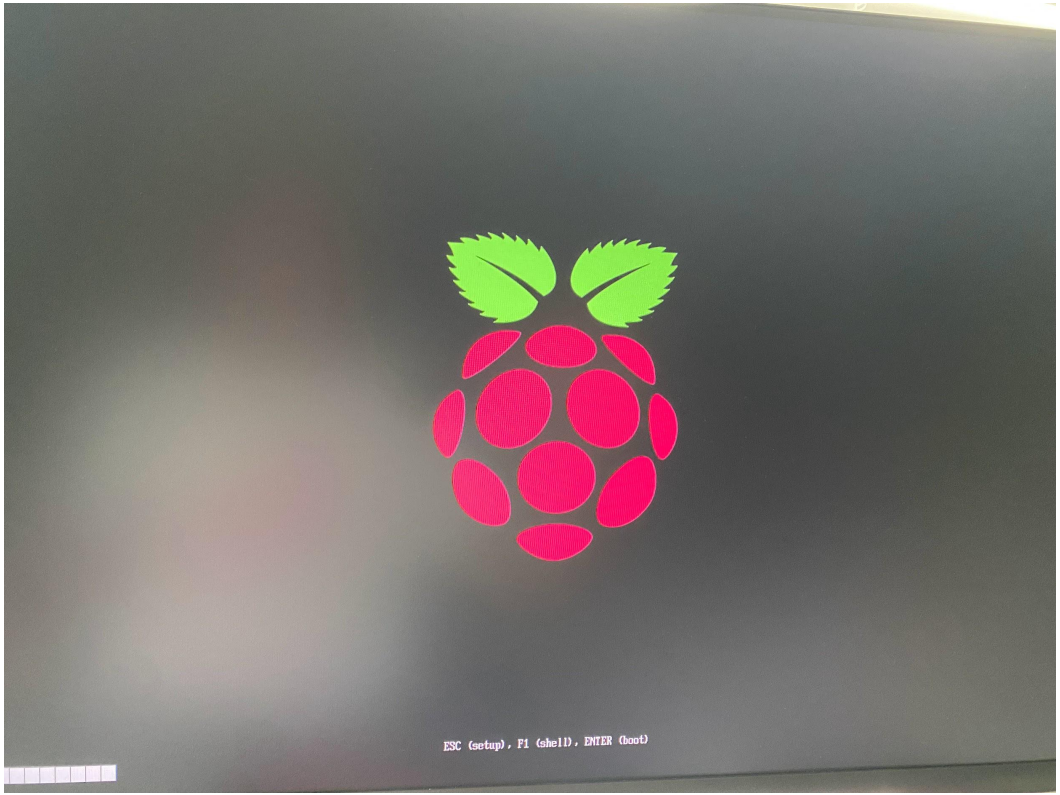
```
Windows Boot Manager
--------------------
identifier              {bootmgr}
device                  partition=P:
path                    \EFI\Microsoft\Boot\bootmgfw.efi
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
default                 {current}
resumeobject            {560d1b4a-b6aa-11ea-bf4a-b2f359f003bc}
displayorder            {current}
toolsdisplayorder       {memdiag}
timeout                 0

Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \WINDOWS\system32\winload.efi
description             Windows 11
locale                  en-US
inherit                 {bootloadersettings}
recoverysequence        {560d1b4d-b6aa-11ea-bf4a-b2f359f003bc}
displaymessageoverride  Recovery
recoveryenabled         Yes
isolatedcontext         Yes
allowedinmemorysettings 0x15000075
osdevice                partition=C:
systemroot              \WINDOWS
resumeobject            {560d1b4a-b6aa-11ea-bf4a-b2f359f003bc}
nx                      OptIn
bootmenupolicy          Standard
```

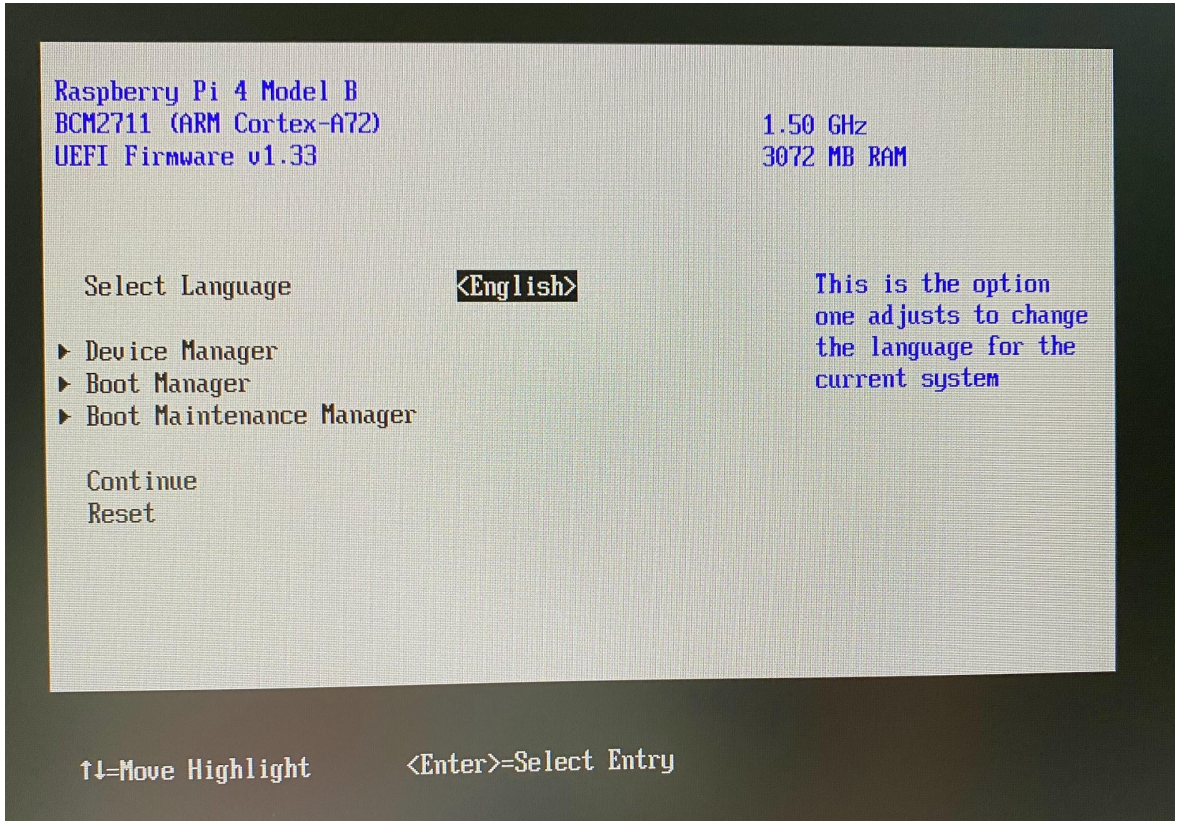
UEFI vs. Legacy Boot

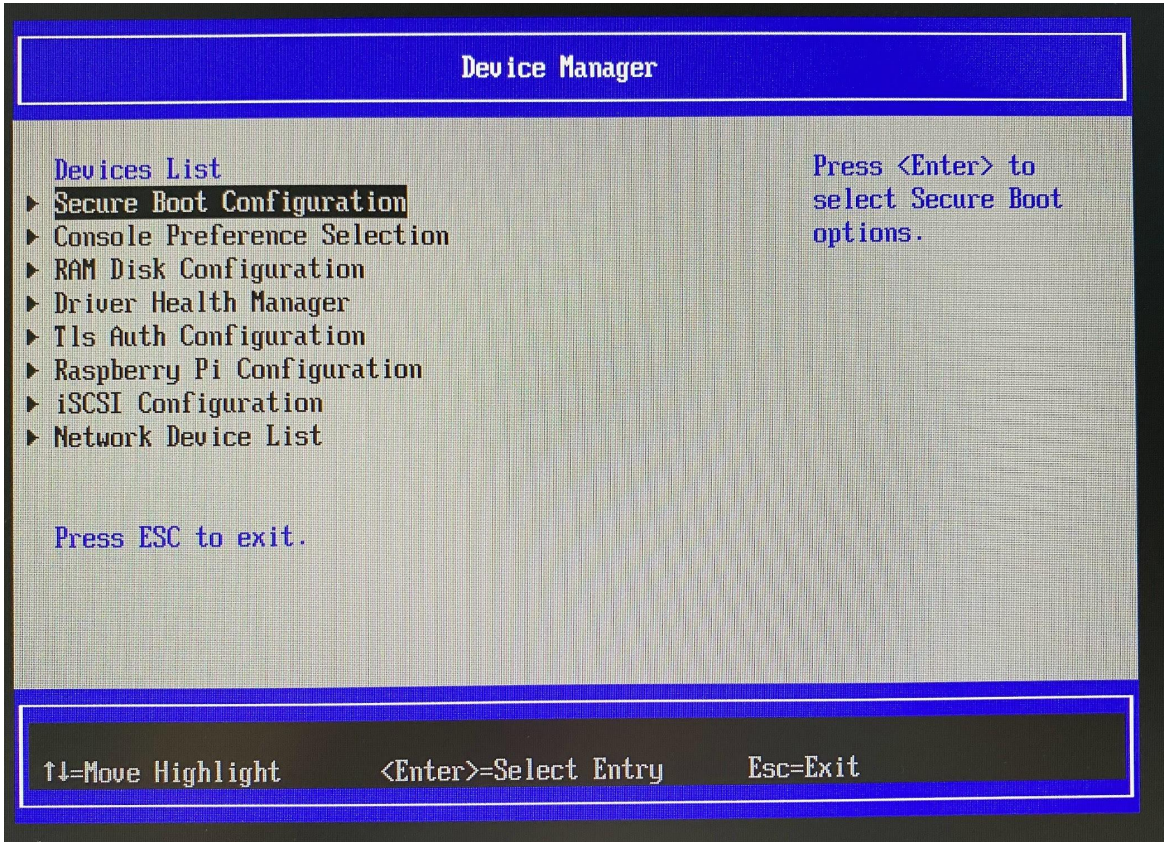
	Legacy Bios	**UEFI Firmware**
Architecture	All vendors did something different	Unified specifications (EDK1/EDK2)
Implementation	Mostly Assembly	C/C++
Memory Model	16-bit real mode	32/64-bit protected mode
Bootstrap	MBR and VBR	None
Partition	MBR	GPT
Disk I/O	System Interrupts	UEFI services
Bootloaders	Bootmgr and winload.exe	Bootmgfw.efi and winload.efi
OS Interaction	BIOS Interrupts	UEFI services
Boot Configuration	CMOS Memory	UEFI NVRAM variable

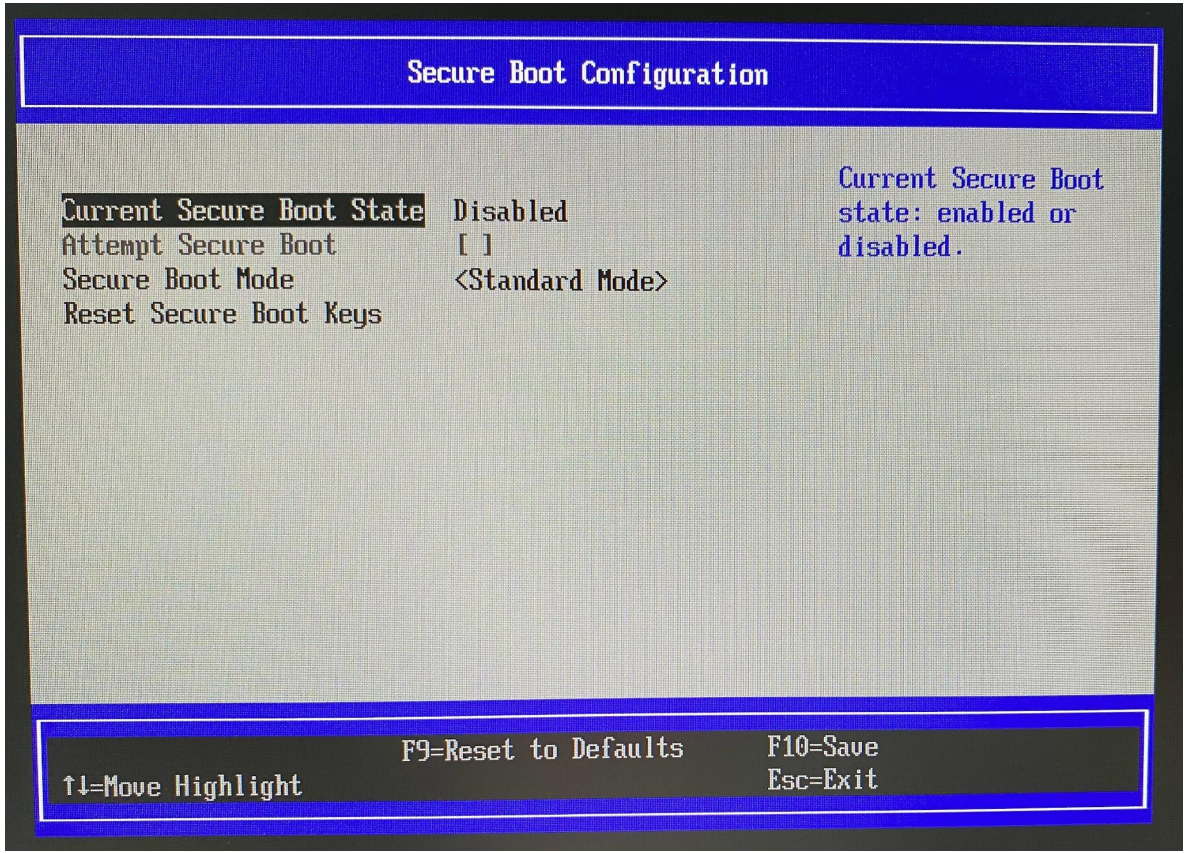
References

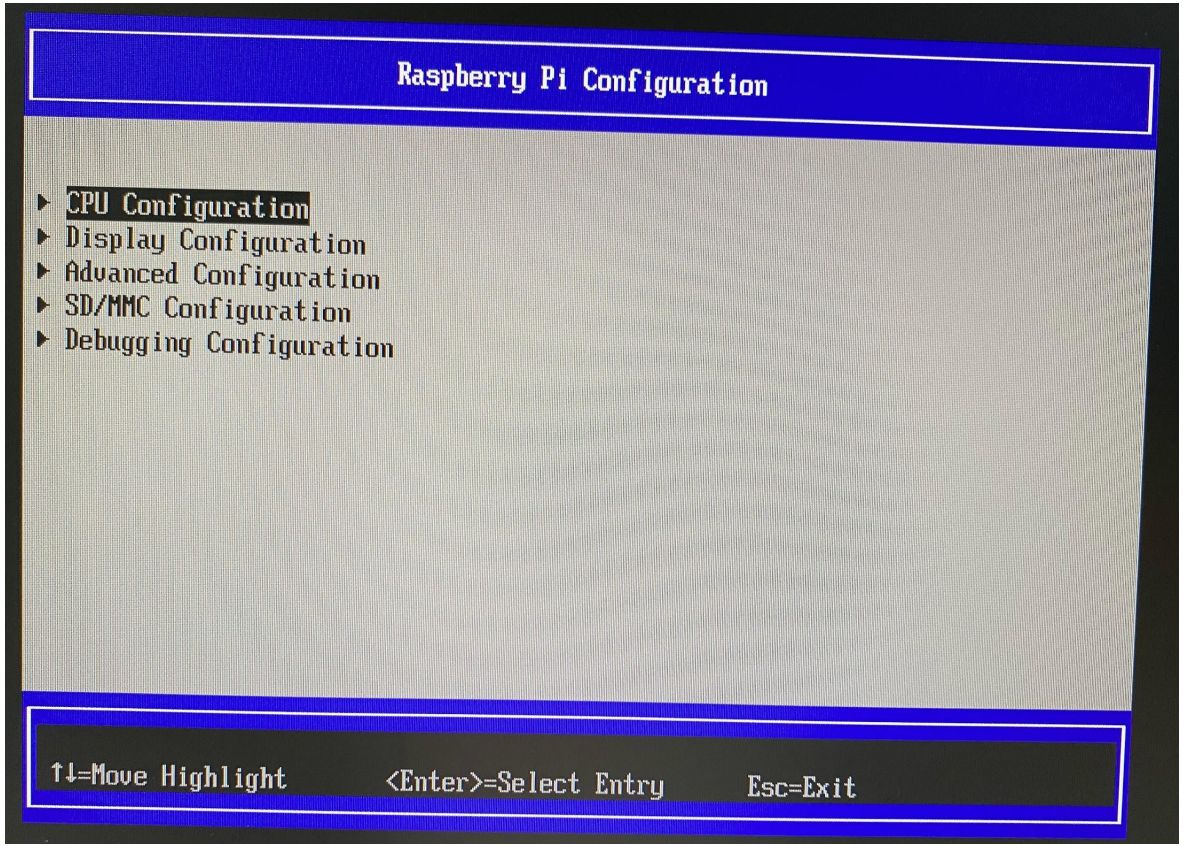
- https://www.javatpoint.com/os-master-boot-record
- https://www.intel.com/content/dam/develop/external/us/en/documents/a-tour-beyond-bios-launching-stm-to-monitor-smm-in-efi-developer-kit-ii-819978.pdf
- https://wikileaks.org/ciav7p1/cms/page_26968084.html
- https://edk2-docs.gitbook.io/edk-ii-uefi-driver-writer-s-guide/3_foundation/readme.15/31510_boot_manager_driver_list_processing

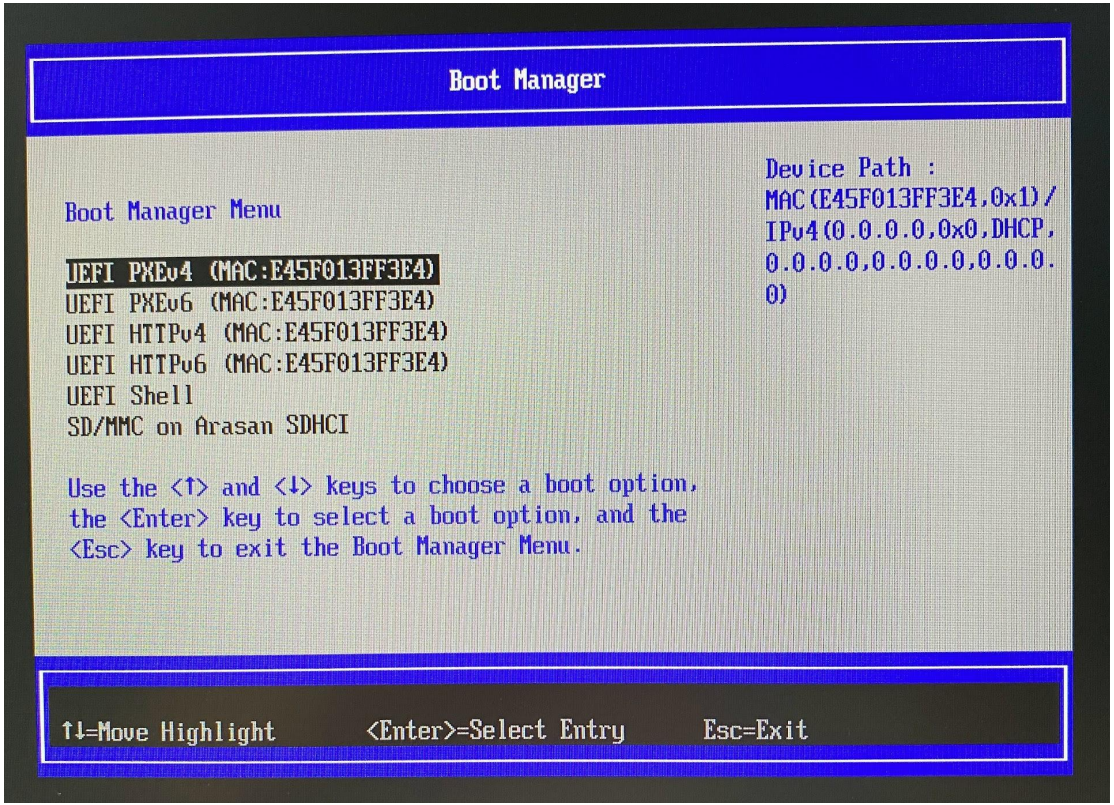

ESC (setup), F1 (shell), ENTER (boot)

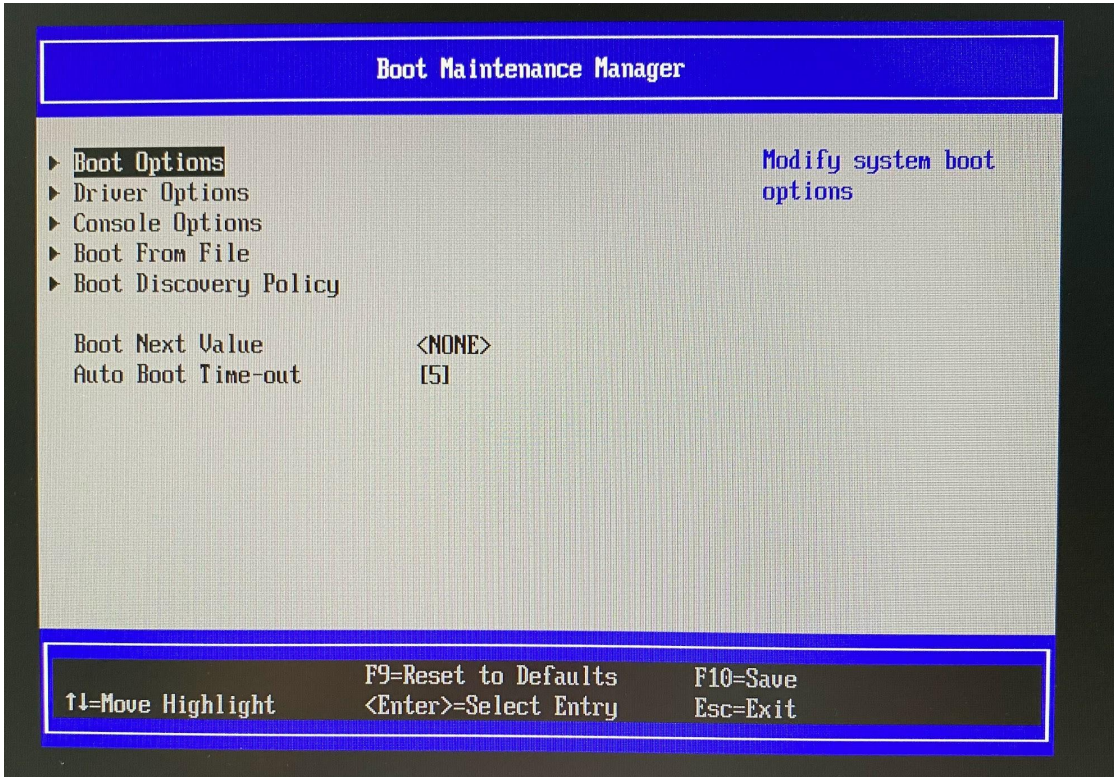

Raspberry Pi 4 Model B
BCM2711 (ARM Cortex-A72)
1.50 GHz
UEFI Firmware v1.33
3072 MB RAM
Select Language
<English>
This is the option one adjusts to change the language for the current system
Device Manager
Boot Manager
Boot Maintenance Manager
Continue
Reset
↑↓=Move Highlight
<Enter>=Select Entry

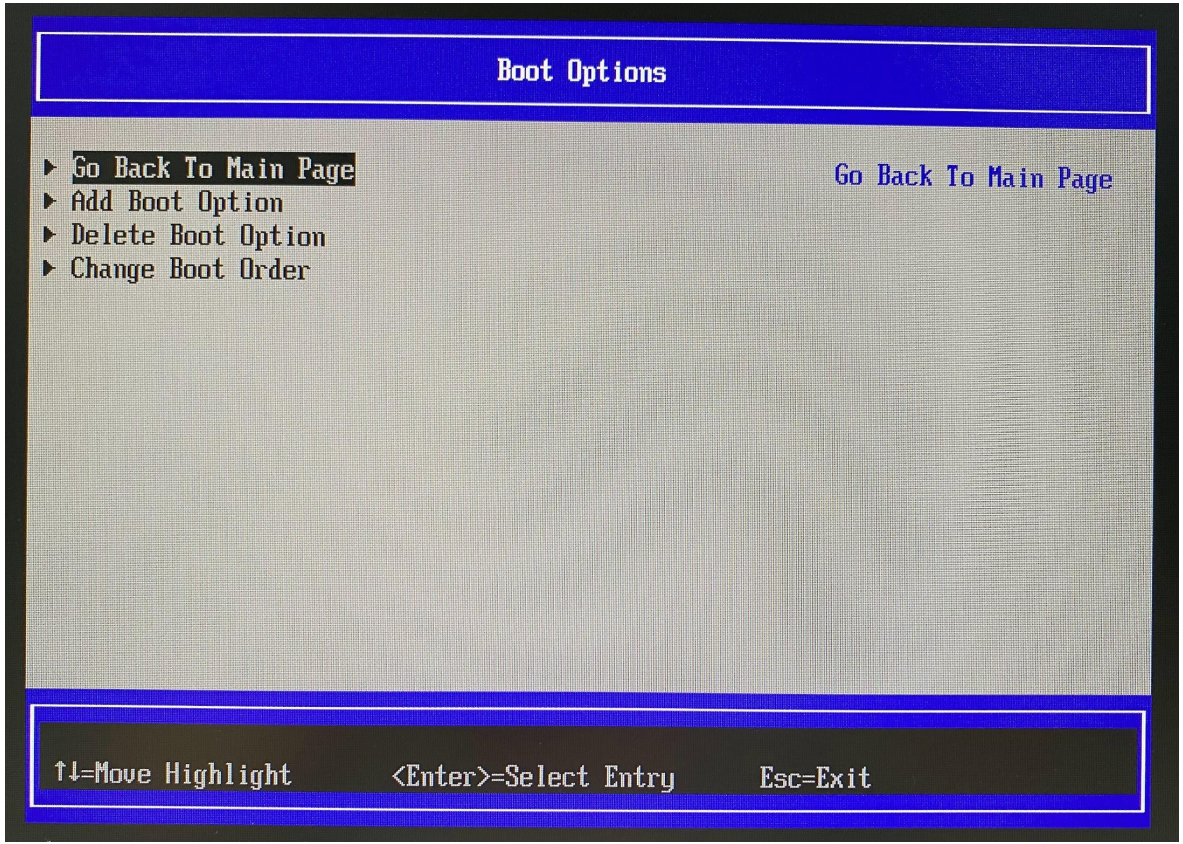

Device Manager
Devices List
Secure Boot Configuration
Console Preference Selection
RAM Disk Configuration
Driver Health Manager
Tls Auth Configuration
Raspberry Pi Configuration
iSCSI Configuration
Network Device List
Press ESC to exit.
Press <Enter> to select Secure Boot options.
↑↓=Move Highlight
<Enter>=Select Entry
Esc=Exit

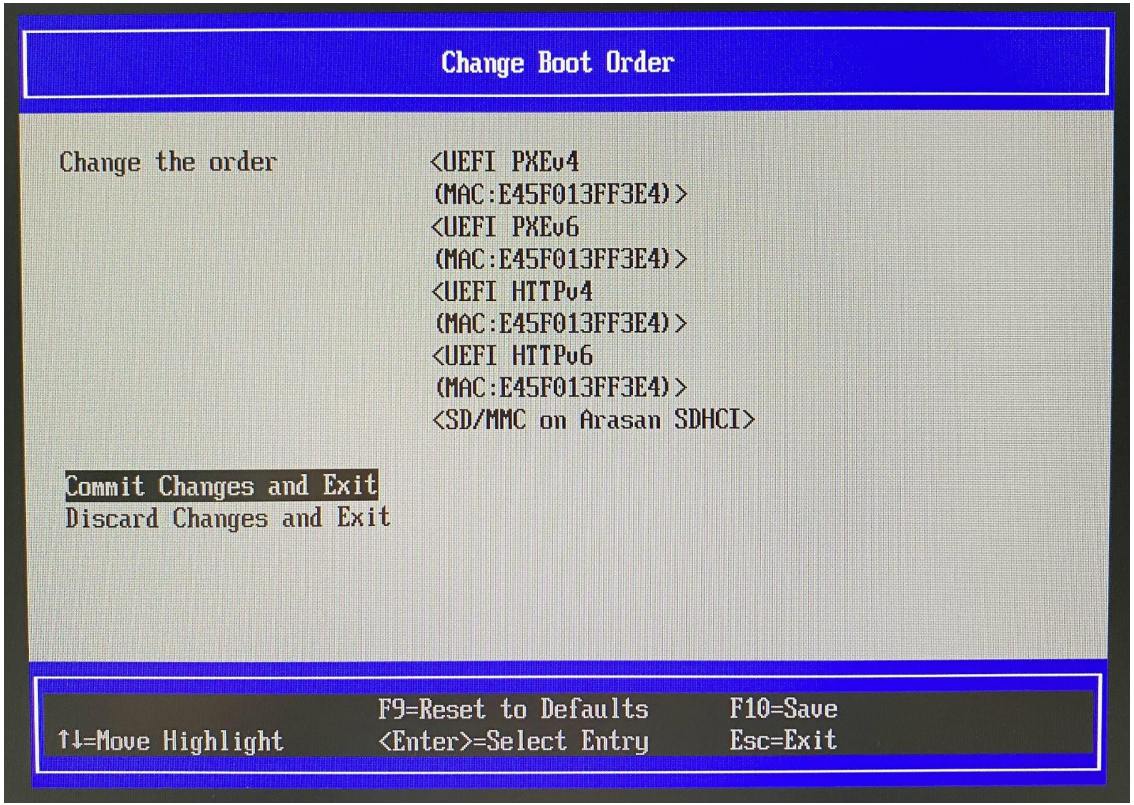

Secure Boot Configuration
Current Secure Boot State
Disabled
Attempt Secure Boot
[ ]
Secure Boot Mode
<Standard Mode>
Reset Secure Boot Keys
Current Secure Boot state: enabled or disabled.
F9=Reset to Defaults
F10=Save
↑↓=Move Highlight
Esc=Exit

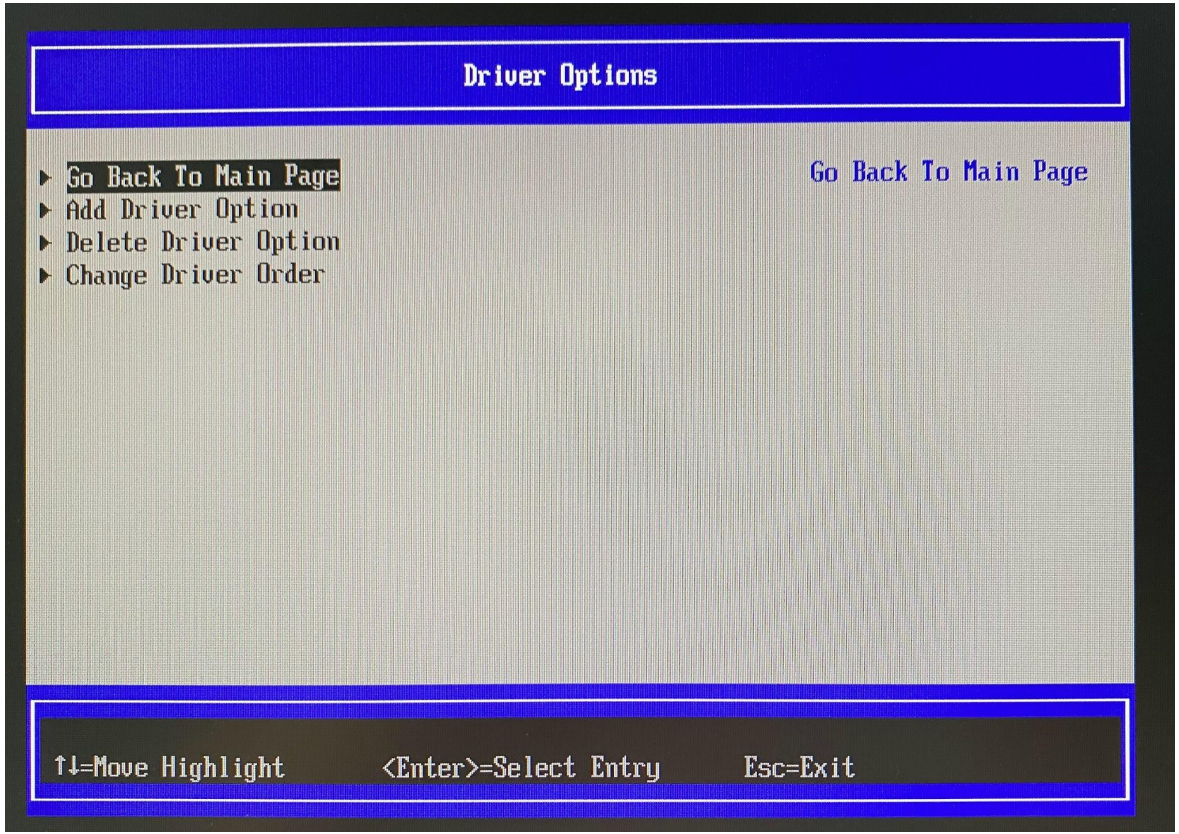

Raspberry Pi Configuration
CPU Configuration
Display Configuration
Advanced Configuration
SD/MMC Configuration
Debugging Configuration
↑↓=Move Highlight
<Enter>=Select Entry
Esc=Exit

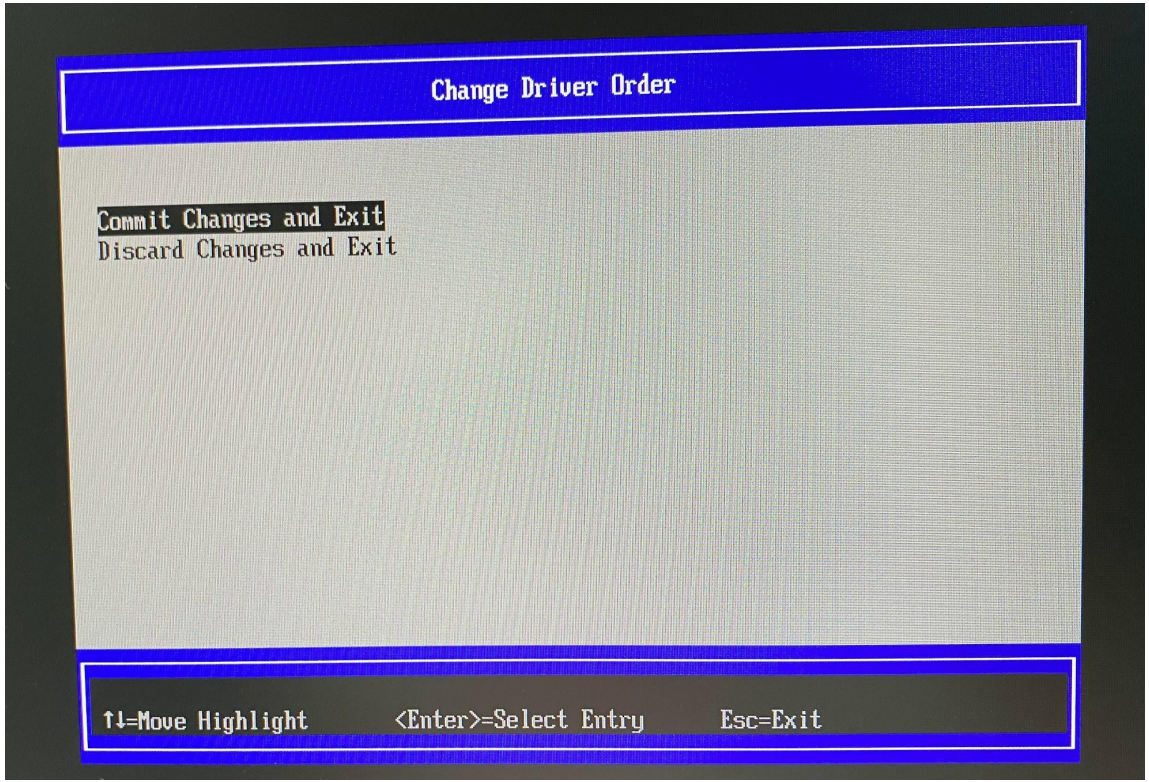

Boot Manager
Boot Manager Menu
UEFI PXEv4 (MAC:E45F013FF3E4)
UEFI PXEv6 (MAC:E45F013FF3E4)
UEFI HTTPv4 (MAC:E45F013FF3E4)
UEFI HTTPv6 (MAC:E45F013FF3E4)
UEFI Shell
SD/MMC on Arasan SDHCI
Use the <↑> and <↓> keys to choose a boot option, the <Enter> key to select a boot option, and the <Esc> key to exit the Boot Manager Menu.
Device Path :
MAC(E45F013FF3E4,0x1)/IPv4(0.0.0.0,0x0,DHCP,0.0.0.0,0.0.0.0,0.0.0.0)
↑↓=Move Highlight
<Enter>=Select Entry
Esc=Exit


Boot Maintenance Manager
Boot Options
Driver Options
Console Options
Boot From File
Boot Discovery Policy
Modify system boot options
Boot Next Value <NONE>
Auto Boot Time-out [5]
↑↓=Move Highlight
F9=Reset to Defaults
<Enter>=Select Entry
F10=Save
Esc=Exit


Boot Options
Go Back To Main Page
Add Boot Option
Delete Boot Option
Change Boot Order
Go Back To Main Page
↑↓=Move Highlight
<Enter>=Select Entry
Esc=Exit


Change Boot Order
Change the order
<UEFI PXEv4
(MAC:E45F013FF3E4)>
<UEFI PXEv6
(MAC:E45F013FF3E4)>
<UEFI HTTPv4
(MAC:E45F013FF3E4)>
<UEFI HTTPv6
(MAC:E45F013FF3E4)>
<SD/MMC on Arasan SDHCI>
Commit Changes and Exit
Discard Changes and Exit
F9=Reset to Defaults
F10=Save
↑↓=Move Highlight
<Enter>=Select Entry
Esc=Exit


Driver Options
Go Back To Main Page
Add Driver Option
Delete Driver Option
Change Driver Order
Go Back To Main Page
↑↓=Move Highlight
<Enter>=Select Entry
Esc=Exit


Change Driver Order
Commit Changes and Exit
Discard Changes and Exit
↑↓=Move Highlight
<Enter>=Select Entry
Esc=Exit