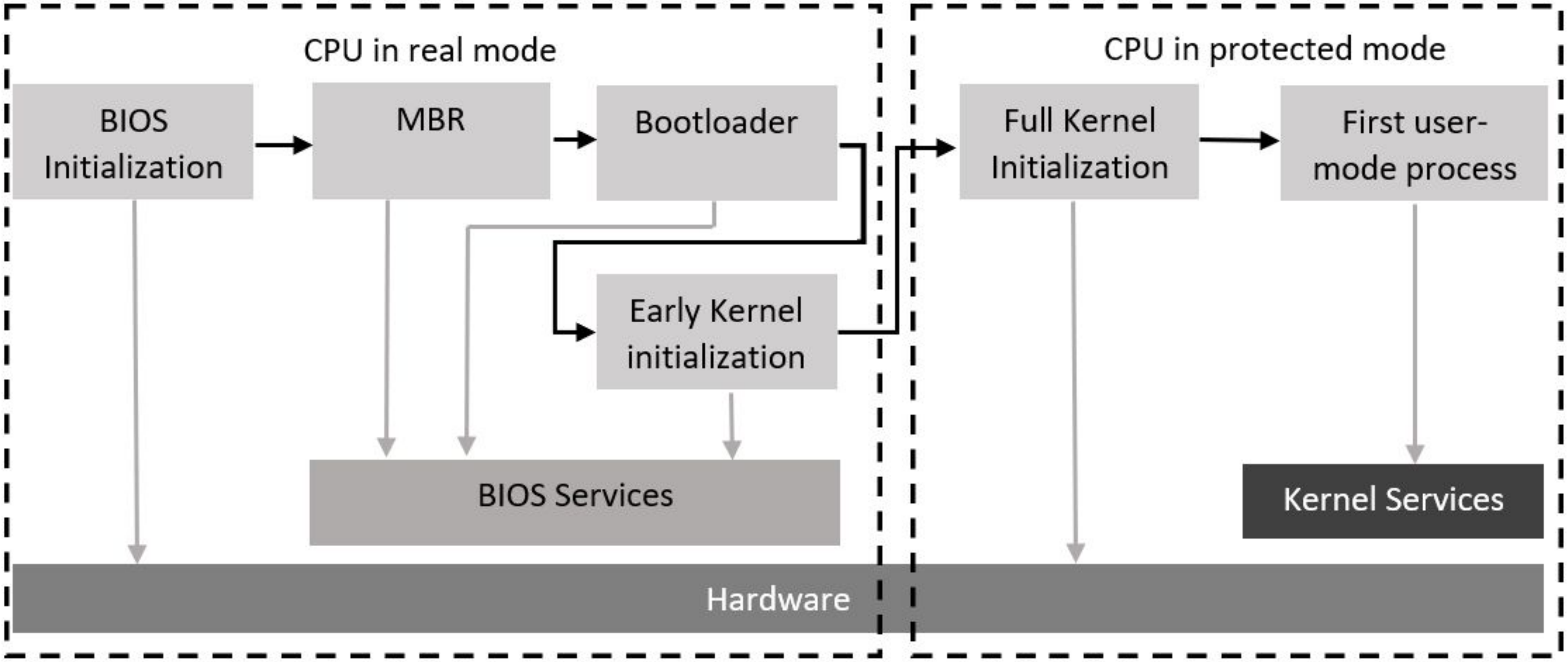
ECE469: UEFI Booting

Aravind Machiry

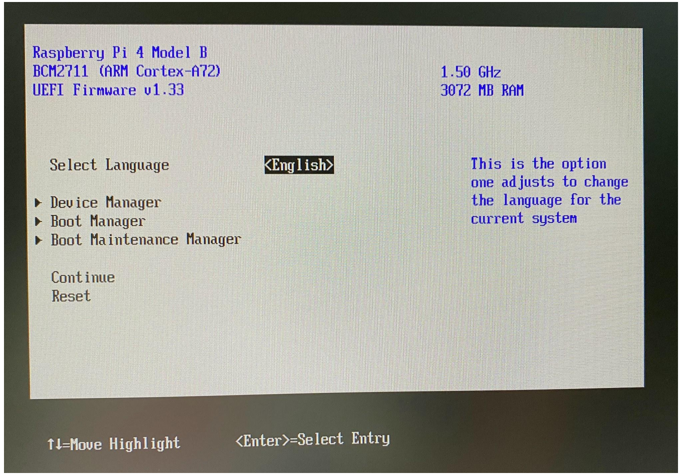
1/16/2025

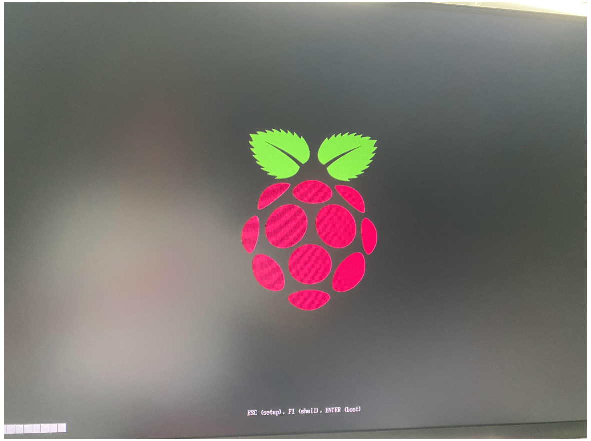
Recap: BIOS Booting



What happens, when we turn on the machine?

1. UEFI:
   a. Unified Extensible Firmware Interface.
   b. Enables basic device access.





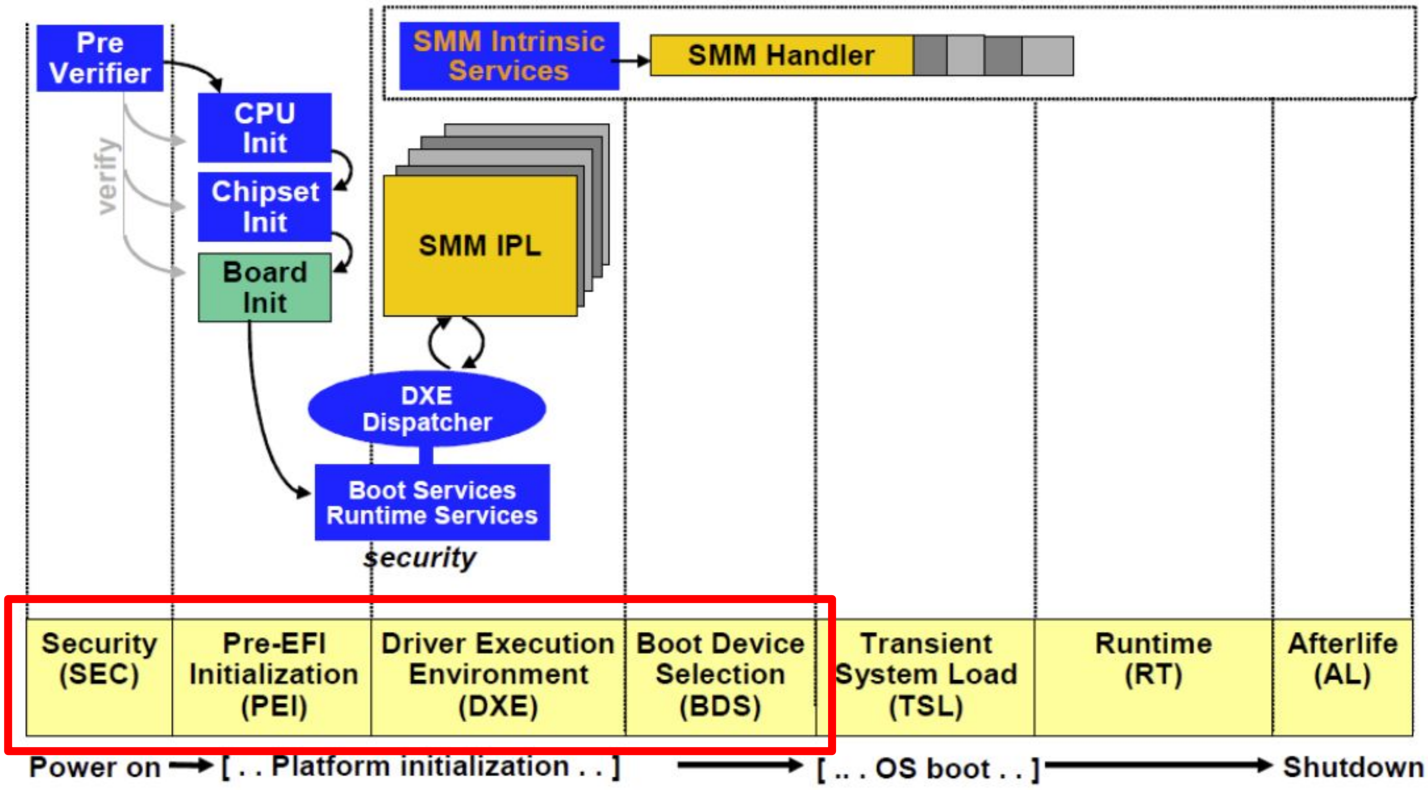
What is UEFI?

- Modular in design (uses generalized communication protocols)
- User friendly interface (sometimes with mouse support)
- Advanced security features (e.g. Secure Boot)
- Larger disk sizes (greater than 2TB)
- Only operates in protected mode
- Easier to maintain (written in C)
- Supports other boot options (e.g. Network Boot)

UEFI vs. BIOS

	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

UEFI Boot Phases



Security

- Executes hardware specific firmware.
  - Written in assembly (16-/32-bit) (SecMain).
- Creates the foundation for the root-of-trust methodology.
  - Authenticates the Pre-EFI Initialization (PEI) Foundation code.
- Creates temporary memory using CPU caches.
- Locates the PEI foundation on the SPI flash.
- The SEC phase is executed on the SPI flash.
  - Address entry point is the reset vector at address space 4GB - 0x10
  - Only the bootstrap processor(BSP) is running.

Pre-Environment Initialization

- The boot code is loaded from the SPI flash in this phase (PeiCore).
- It initializes the permanent memory, but until then everything is executed in the CPU cache. (InitializeMemory)
- This is where the runtime and boot services begin execution. (InitialzeDXE -> DXELoadCore)
- Creates hand off block (HOB) list for later phases.
- The final module is the block to load the next phase (PeimInitializeDxeIpl).
- The most architecture depend part of the code.

Driver Execution Environment

- 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
- Images are loaded and executed in two ways:
  - Through the DriverOrder option in the NVRAM
  - The default boot order

What is NVRAM?

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

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.

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.

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.

System Management Mode (SMM)

- Operates inside protected memory called SMRAM
- It is similar to Arm's TrustZone
- It has the highest privilege on the system (Ring -2)
- Operates in 16-bit mode
- It is responsible for direct hardware controls and power management
    - Flash System Firmware, write to the MMIO, etc
- A System Management Interrupt (SMI) is required to call anything inside of the SMM

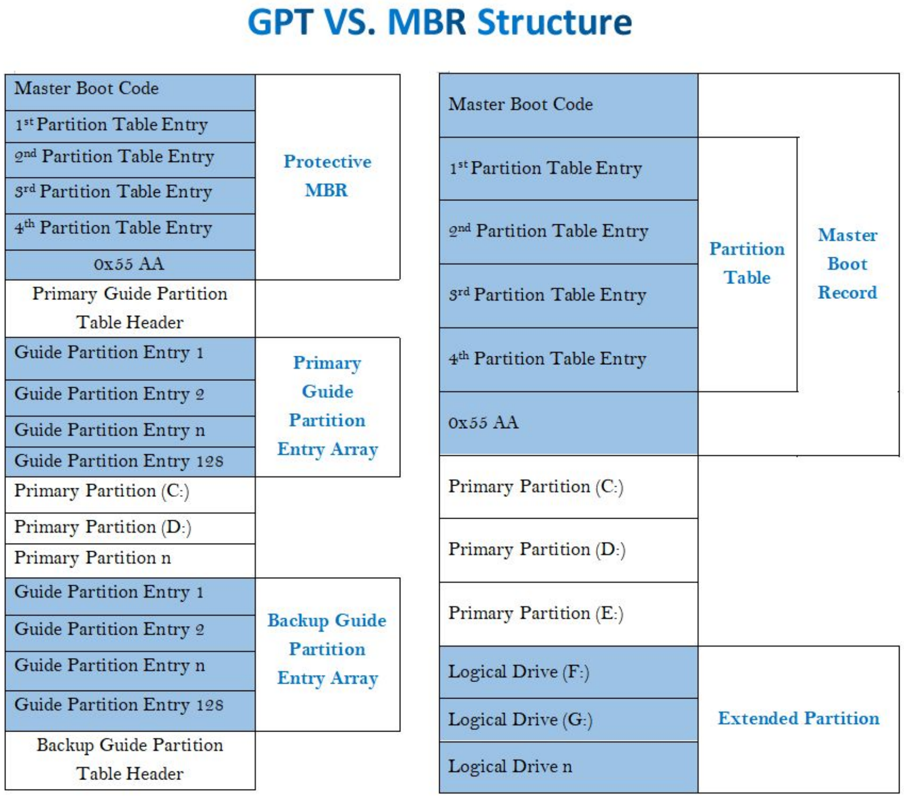
What is SMRAM?

- The SMRAM is apart of the DRAM
- SMBASE is a fixed address in a CPU register
  - Used to find the starting location of the SMRAM
- Only the SMI handler can modify the SMRAM
  - That means anyone can read the SMRAM
- There is a specific bit called the D_LCK bit
  - If set then no SMRAM configuration bits can be changed

Boot Device Selection

- 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.
- The boot manager utilizes the DXE drivers that were created to complete its tasks.
- The OS loader is stored on the EFI system partition that uses a GUID Partition Table instead of the traditional MBR.

GPT vs. MBR



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

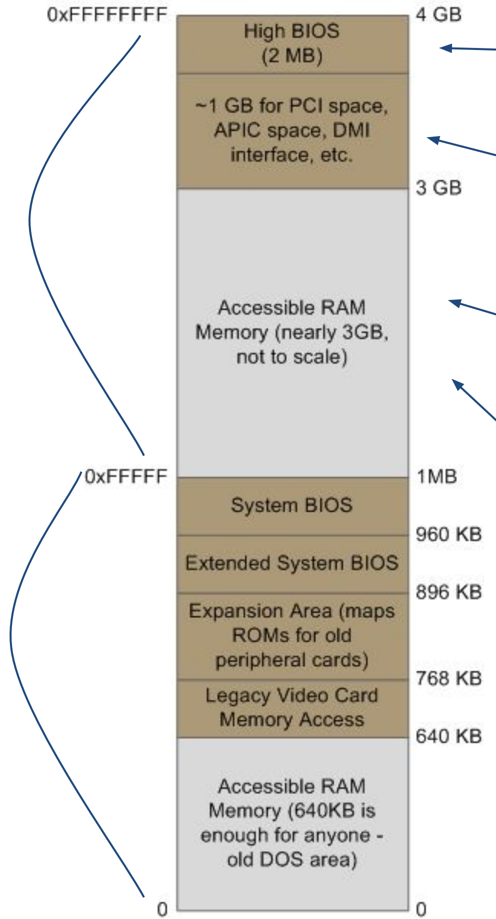
UEFI vs. BIOS (Review)

	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

Summary!

UEFI Firmware

Reserved for BIOS



Mapped Code

MMIO and other peripheral Mappings

Scratch Memory for different device drivers

The extra memory provides more flexibility in system configurations