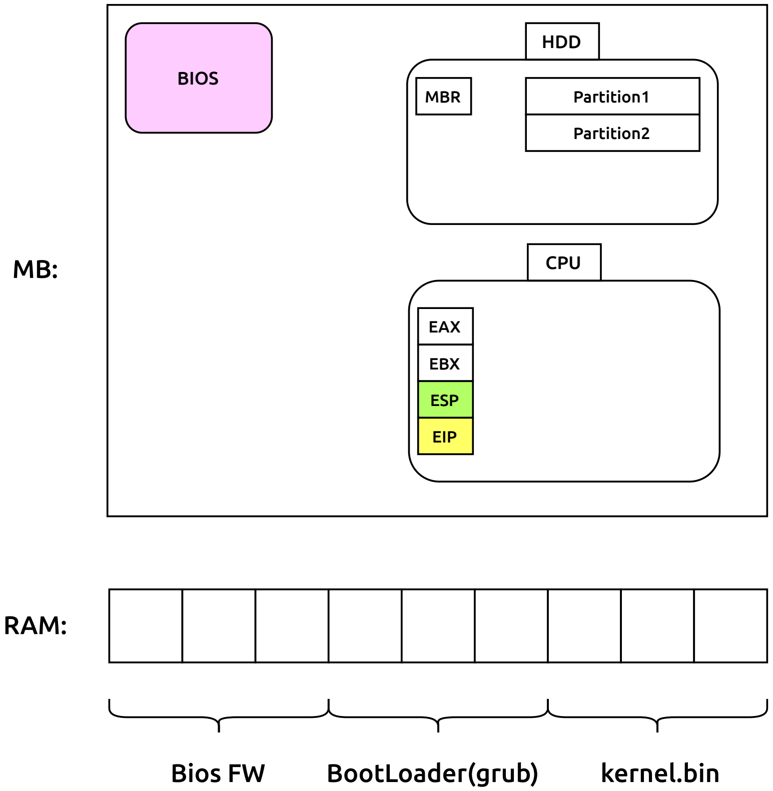

Challanges:

- When Bootloader loads the kernel into the ram, it doesn't update ESP.
- C/C++ programs expect the ESP always be set before our application starts to run.
- To set ESP manually, we need to do it manually.
    - We'll create an assembly file called: loader.s for this purpose.
        - loader.s will jump to our kernel.c at the end.
- loader.s will compile with: *as* --> produce: loader.o
- kernel.c will compile with: *gcc* --> produce: kernel.o
- *ld* will link loader.o and kernel.o into **kernel.bin**
- We then put this kernel.bin into /boot and write an entry in grub.cfg for it.

## NOTES

Items in your computer:

- MotherBoard (MB)
    - A HardDrive attached to it. (HDD)
    - CPU (has several registers)
        - General purpose: EAX, EBX, ECX, EDX
        - Instruction Pointer: EIP
        - Stack Pointer: ESP
        - Base Pointer: EBP
        - Status Register: EFLAGS
        - Control Registers: CR0, CR2, CR3, CR4, CR8

What happens when you start your computer?

1. MB copies data from BIOS and load it into RAM. (Bios Firmware)
2. Then EIP will point to the beginning of RAM (which contains Bios FW)
    1. CPU will read and execute the instruction at that position.
3. FW then tell cpu: "Hello CPU!, please talk to the HDD"
4. CPU read raw data from HDD. (the first 2 MB which called Boot Sector or Master Boot Record, MBR)
    1. There's no concept of partition table or file system at this level.
5. BootLoader will load from MBR into the RAM.
    1. EIP points to the memory address of BootLoader in RAM.
    2. CPU starts executing instructions.
6. BootLoaders knows about partition tables and file systems.
    1. So it can deal with Directories and Files.
7. BootLoader loads: **/boot/grub/grub.cfg**
    1. This file contains some menu entries for each OS on your HDD.
8. BootLoader prints all menu entries in grub.cfg on the screen.
9. User can use arrow-keys on his keyboard to select one of the OSs.
10. When you select an OS, it will load it's appropriate bin related to that OS. (**/boot/kernel.bin**)
11. For linux, it's: **kernel.bin**.
12. BootLoader, loads kernel.bin into the RAM.
13. EIP will update to point to the beginning of the kernel.bin.
14. CPU starts executing instructions from the entry point of kernel.bin.
15. This is exactly where our code will start to execute.
16. *When you start your computer, CPU starts in 32-bit compatibility mode.*
    1. So everything in our kernel is in 32-bit mode.