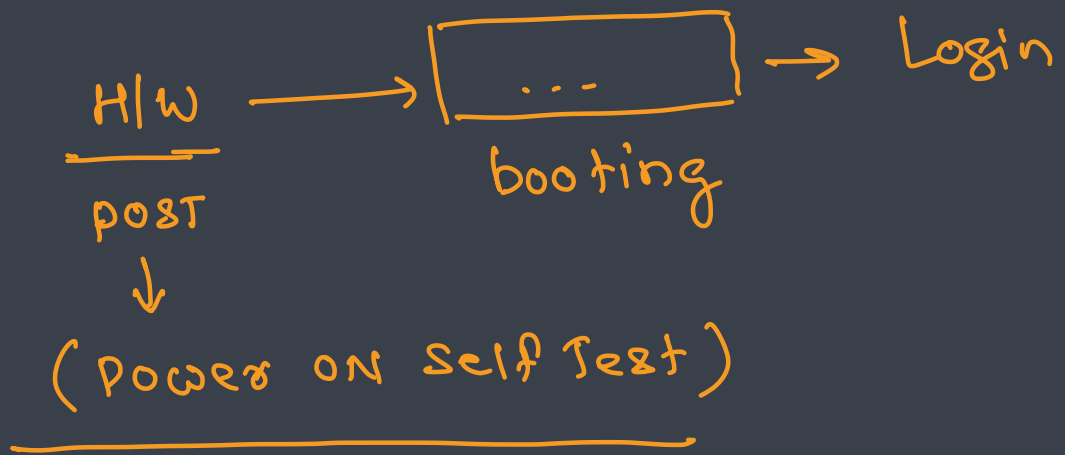
Booting

Booting

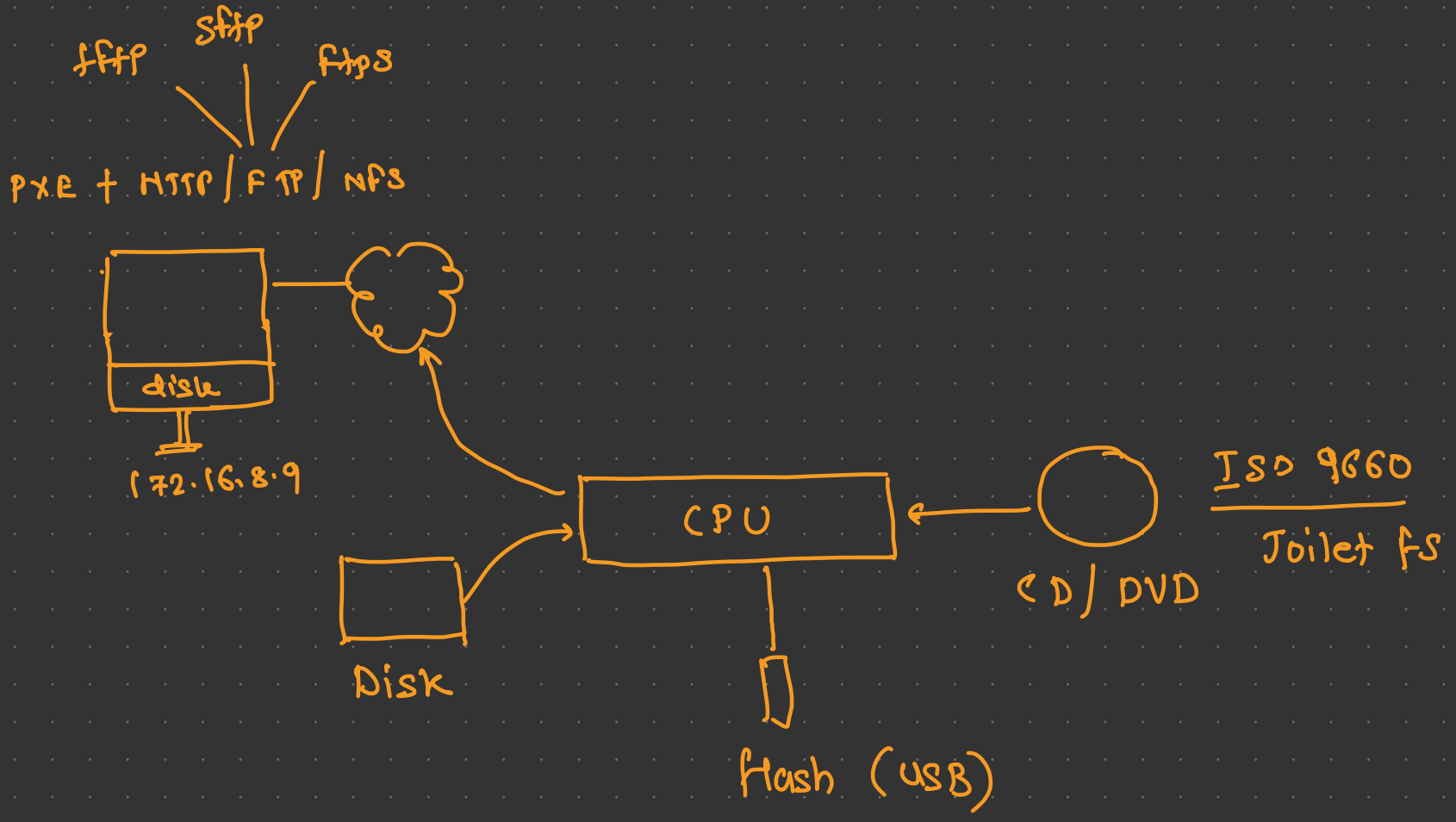
- The boot process consists of many steps, which begin with the hardware layer and move up through the BIOS, boot loader, Linux kernel, and authentication layers
- Understanding this process is helpful for troubleshooting and performance tuning
- Not only does the hardware need to be initialized and the core operating system need to be brought up and shut down, but an impressive list of services and processes must also be started and stopped at the right moment



① bootloader

② kernel

③ system



Boot Sources

- Installation files may be stored on various media and available from several sources. Most organizations will have a standard Linux distribution for deployment on internal systems
- Storage Disk
  - Typically the system BIOS/UEFI is configured to boot from the internal storage disk → HD / SSD
  - Source files may have been copied to the disk from a remote source or from a currently installed operating system that you will replace
- USB → connector → flash drives (pen drives) or external disks →
  - In this configuration, Linux would boot from a removable USB flash drive or storage disk that contains the Linux source files
  - The system BIOS may need to be adjusted to boot from USB storage rather than an internal storage disk or DVD drive
  - Typically, you must configure USB disks to be bootable
- Virtual Machine File → cloud (AWS / Azure / GCP) → AWS → Amazon Machine Image (AMI)
  - You can also boot from a preconfigured virtual machine configuration file specifying CPU, memory, storage, and network configurations
  - The hypervisor virtualization layer constructs the VM and boots it, initiating the installation from the storage media
- ISO Image
  - An ISO image is a system image, originally that of an optical disc (CD)
  - It is a common file format for packaging and distributing images of operating systems that users can boot from to install the OS
  - Typically, you write the ISO image to an optical disc or USB thumb drive, insert the media into the computer, and instruct a boot environment like UEFI to boot from that media
  - ISOs are also commonly used to construct virtual machines

Boot Sources

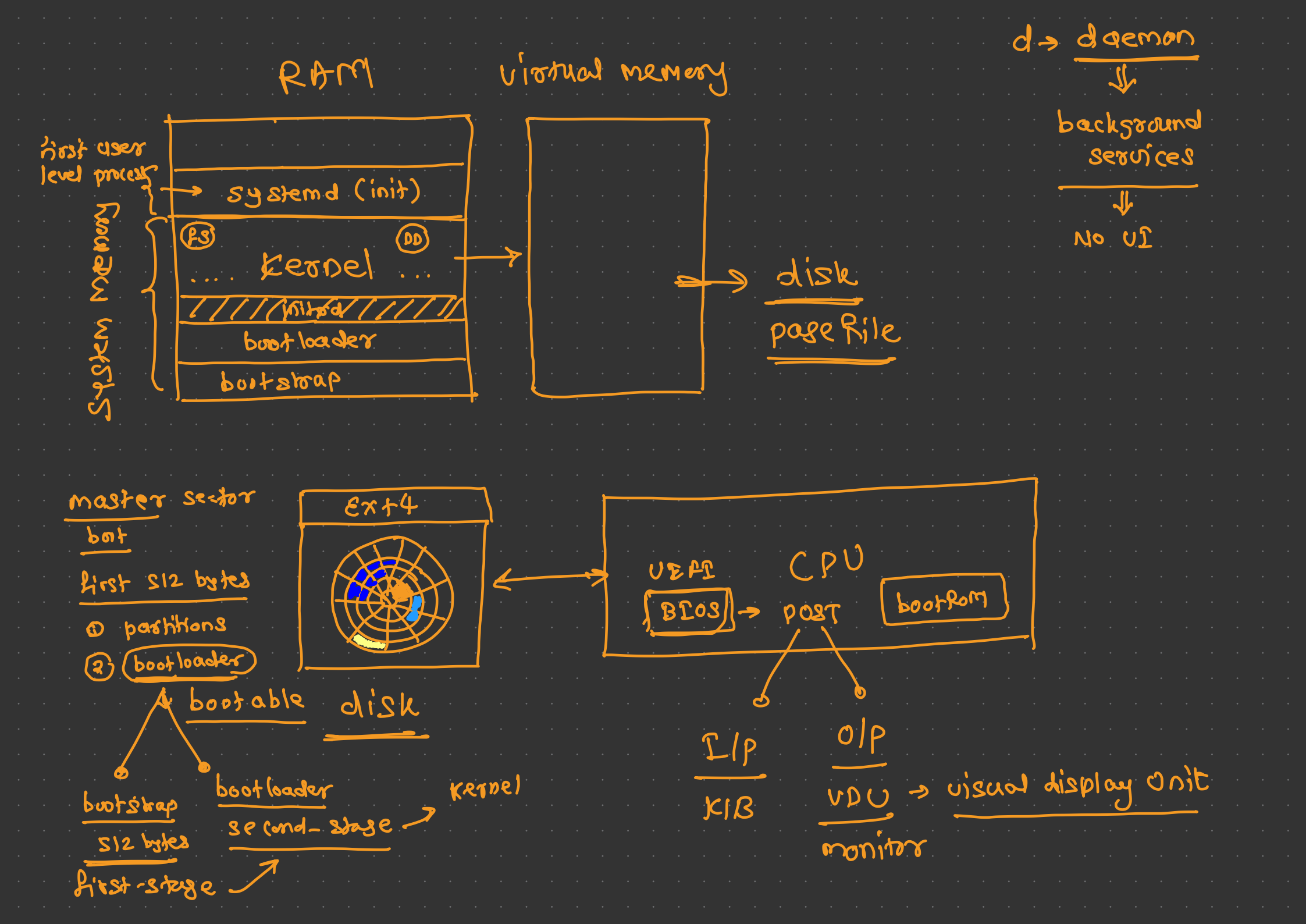
- PXE
  - Preboot Execution Environment (PXE) is a part of the UEFI standard that enables a client to retrieve the necessary boot loader and system files from a server over the network
  - The client configures UEFI to boot from PXE, and during the startup process, it will search for Dynamic Host Configuration Protocol (DHCP) servers that also act as PXE servers
  - Once the proper server is found, it transfers the necessary boot files to the client over the Trivial File Transfer Protocol (TFTP)
- HTTP and FTP
  - Clients can also acquire boot data over a network from content delivery protocols like Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP)
  - These are typically faster, more reliable, and more secure than the standard TFTP protocol used in PXE. Open-source implementations of PXE, like iPXE, extend PXE support to include these protocols
- Network File System
  - This is another network boot option
  - Rather than store system files on a local storage drive, a client will mount a Network File System (NFS) share as its root file system
  - The share must be prepared ahead of time and stored on an NFS server that the client can retrieve the files from
  - Therefore, the client does not store data locally, but rather stores it on the NFS server
  - DHCP, TFTP, and other network protocols communicate the necessary boot data in such an environment

Boot Process

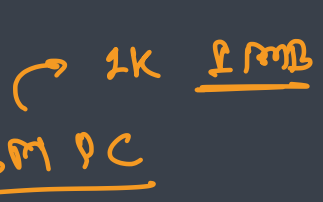
Boot Process

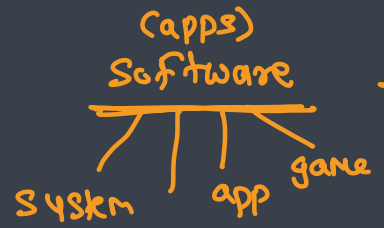
POST

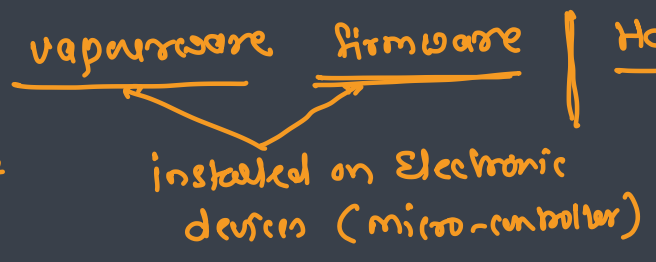
1. BIOS/UEFI
2. MBR/GPT
3. Boot Loader
4. Kernel
5. Init [systemd]
6. Run Level

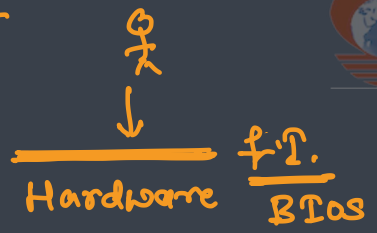

RAM
virtual memory
first user level process
system memory
systemd (init)
FS
DD
.... kernel ...
initrd
bootloader
bootstrap
disk
pageFile
d → daemon
background services
No UI
master sector
bot
first 512 bytes
① partitions
② bootloader
bootable
disk
Ext4
UEFI
BIOS
CPU
POST
bootrom
I/p
KIB
O/p
VDU → visual display unit
monitor
bootstrap
512 bytes
first-stage
bootloader
second_stage
kernel

BIOS/UEFI





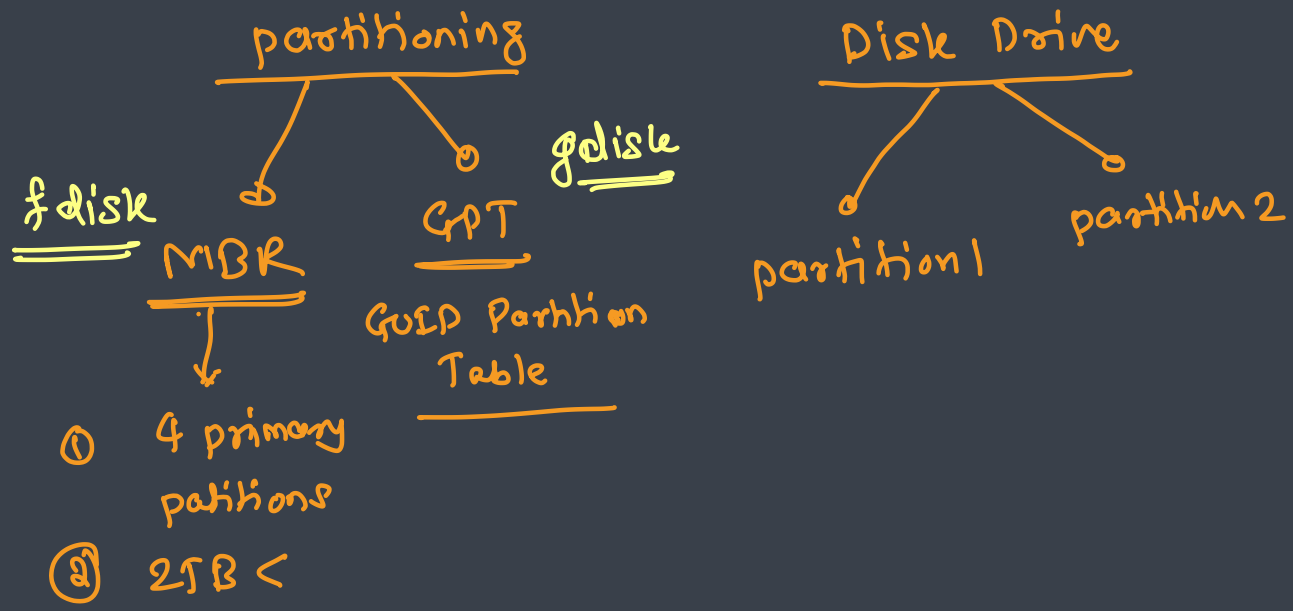




- Basic Input/Output System (BIOS)
  - It is a standard for firmware interfaces stored on a computer motherboard's ROM chip
  - The BIOS firmware runs when the computer powers on, enabling it to test the various hardware components in a computer and run the boot loader to start the operating system
  - The BIOS has access to the ports used by basic hardware input devices like a mouse and keyboard. Users can also load up a BIOS interface instead of an operating system to make various hardware-level changes
  - BIOS was the dominant standard for home and enterprise computers for several decades
- Unified Extensible Firmware Interface (UEFI)
  - It is newer firmware technology that has largely replaced BIOS by bringing with it several key advantages
  - UEFI runs faster than BIOS, can operate within a greater amount of memory, can access storage drives of currently unattainable sizes, can access more hardware types, and has improved security protections
  - Most modern motherboards, as well as the pre-assembled PCs that use them, ship with UEFI. The secure boot features of UEFI are critical
  - Like BIOS, UEFI provides an environment to execute a boot loader and ultimately start up the operating system
- One secure boot feature that both BIOS and UEFI include is the ability to set a password.
- If this password is not provided at boot time, the system will not start
- Since BIOS/UEFI firmware differs between hardware manufacturers, the process of setting this password is not consistent

MBR → Master Boot Record

- Master Boot Record is the first sector of a hard disk and the place where all the information about the disk and booting can be found
- It is the most essential part of the booting process
- Along with the bootloader program, MBR also contains details regarding the partitions of the hard disk
- The size of MBR is commonly less than or equal to 512 bytes



Boot Loader → load kernel →

① LiLo → Linux Loader
② GRUB → Grand Unified Bootloader 2

- Booting is starting or restarting a computer and loading an operating system for the user to access
- A booting environment reads a small program stored in read-only memory (ROM)
- This program then executes various operations in RAM that bootstrap the operating system and make it available for use
- A boot loader is a small program stored in ROM that loads the kernel from a storage device and then starts the operating system
- A boot environment like BIOS reads the boot loader from ROM so that the boot loader can execute the necessary operations to begin the process
- Boot loaders can protect the boot process with a password to prevent unauthorized system (→ Linux) startup
- In addition, boot loaders can load more than one operating system into the computer's memory, but the user needs to select the desired operating system to use during boot
- The boot loader uses three main components that work together to load the operating system in stages:
  - The boot sector program is loaded by a boot environment on startup and has a fixed size of 512 bytes. Its main function is to load the second-stage boot loader; however, it can also load another sector or a kernel
  - The second-stage boot loader loads the operating system and contains a kernel loader
  - Finally, the boot loader installer controls the installation of drive sectors and can only be run when booting from a drive. It coordinates the activities of the boot sector and the boot loader

Kernel

- The initial ramdisk (initrd) refers to the root file system that is temporarily loaded into memory upon system boot
- The initrd loads along with the kernel, which controls its functionality
- The initrd enables the system to start in two phases. In the first phase, the system boots with the minimal set of modules required to load the main or the permanent root file system
- In the second phase, when the main root file system is mounted, the previously mounted initrd file system is removed and the user-space boot process continues
- The initrd is useful because many potential variables can complicate the boot process
- For example, the kernel needs to find and load the necessary device driver modules and the actual root file system itself
- There's also the possibility that the root file system uses one of several advanced storage methods, like LVM or NFS, which has different mount requirements than a standard partition has
- Rather than hardcode all of this behavior in the kernel and introduce bloat, the initrd's temporary root file system can handle these tasks
- The Linux initrd image is an archive file containing all the essential files required for booting the operating system
- It can be built or customized to include additional modules, remove unnecessary modules, or update existing modules
- Typically, the /boot directory stores this image

Init Process → Run Level

- SysV Init — First user level process
  - On the traditional System V (SysV) style of Linux distros, the init process is the first non-kernel process that is started
  - It always gets the process ID number of 1
  - init reads its configuration file, /etc/inittab, and determines the runlevel where it should start
  - Essentially, a runlevel dictates the system's behavior
  - Each level (designated by an integer between 0 and 6) serves a specific purpose
  - A runlevel of initdefault is selected if it exists; otherwise, you are prompted to supply a runlevel value
- systemd
  - Most modern Linux distros have substituted the functionality previously provided by SysV init with a new startup manager systemd
  - The notion of runlevels is different in systemd, and instead are referred to as target
  - There are different runlevels in the traditional SysV world as well as their equivalent in the systemd world

Runlevel	Systemd Targets	Description
0	poweroff.target, runlevel0.target	Halt the system
1	rescue.target, runlevel1.target	Enter single-user mode
2	runlevel2.target, multi-user.target	Multiuser mode, traditionally without Network File System (NFS)
3	multi-user.target, runlevel3.target	Full multiuser mode (normal)
4	multi-user.target, runlevel4.target	Unused or user defined
5	graphical.target, runlevel5.target	Same as runlevel 3, except using an X Window (graphical) System login rather than a text-based login
6	reboot.target, runlevel6.target	Reboot the system
emergency	emergency.target	Emergency shell

Summary

- The processor checks for the BIOS/UEFI firmware and executes it
- BIOS/UEFI checks for bootable media from internal storage devices or peripherals like USB flash drives and DVD-ROMs. It locates a valid device to boot the system
- BIOS/UEFI loads the primary boot loader (probably GRUB2) from the MBR/GPT partition into memory. It also loads the partition table along with it.
- GRUB2 prompts the user to select an operating system to boot. If the user does not respond, then the default operating system is booted.
- The boot loader determines the kernel and locates the corresponding kernel binary
- It then uploads the respective initrd image into memory and transfers control of the boot process to the kernel
- The kernel configures the available hardware drivers, including processors, I/O subsystems, and storage devices. It decompresses the initrd image and mounts it to load the necessary drivers. If the system implemented any virtual devices, such as LVM or software RAID, then they are initialized
- The kernel mounts the main root partition and releases unused memory
- The systemd program runs to set up the user environment. It becomes process ID 1

Summary

- The systemd program searches for the default.target file, which contains details about the services to start
- It mounts the file system based on the /etc/fstab file or .mount files and begins the process of starting services
- On most systems, the selection is either multi-user.target or graphical.target
- If graphical mode is selected, then a display manager starts and the login window is displayed on the screen
- The user enters a user name and password to log in to the system
- The system authenticates the user. If the user is valid, then various profile files are executed
- The shell starts, and the system is ready for the user