# GRUB

- Briefly, a boot loader is the first software program that **runs when a computer starts**.
- It is responsible for **loading and transferring control** to an operating system kernel software
- The kernel, in turn, initializes the rest of the operating system.

- **GNU GRUB is a very powerful boot loader, which can load a wide variety of free operating systems, as well as proprietary operating systems with chain-loading1**.

- One of the important features in GRUB is **flexibility**; GRUB understands file systems and kernel executable formats, so you can load an arbitrary operating system the way you like, without recording the physical position of your kernel on the disk.
- Thus you can load the kernel just by specifying its file name and the drive and partition where the kernel resides.

- When booting with GRUB, you can use either a **command-line interface, or a menu interface.**
- Using the **command-line interface**, you type the drive specification and file name of the kernel manually.
- In the **menu interface**, you just select an OS using the arrow keys.

# Differences from previous versions

- GRUB 2 is a rewrite of GRUB.
- The configuration file has a new name (‘grub.cfg’ rather than ‘menu.lst’ or grub.conf’)
- ‘grub.cfg’ is typically automatically generated by grub-mkconfig.
- Partition numbers in GRUB device names now start at 1, not 0.
- The configuration file is now written in something closer to a full scripting language: variables, conditionals, and loops are available.

- **A small amount of persistent storage is available across reboots,** *using the save_env and load_env commands in GRUB and the grub-editenv utility. This is not available in all configurations.*

- **GRUB 2 has more reliable ways to find its own files and those of target kernels on multiple-disk systems, and has commands to find devices using file system labels or Universally Unique Identifiers (UUIDs).**

- **GRUB 2 can read files directly from LVM and RAID devices.**
- GRUB 2 puts many facilities in dynamically loaded modules, allowing the core image to be smaller, and allowing the core image to be built in more flexible ways.

# Functions of GRUB

- **Basic functions must be straightforward for end-users.**
- **Rich functionality to support kernel experts and designers.**
- **Supports chain-loading function.**
- **Recognize multiple executable formats**
- **Support non-Multiboot kernels**
- **Load multiples modules**
- **Fully support the Multiboot feature of loading multiple modules.**
- **Load a configuration file**

- **Support a human-readable text configuration file with preset boot commands.**
- **Provide a menu interface . A menu interface listing preset boot commands, with a programmable timeout,is available.**
- **Have a flexible command-line interface**
- **A fairly flexible command-line interface, accessible from the menu, is available to edit any preset commands, or write a new boot command set from scratch.**

- **If no configuration file is Support multiple file system types**

  **Support multiple filesystem types transparently, plus a useful explicit blocklist notation. resent, GRUB drops to the command-line.**
- **Access data on any installed device .**
- **Support reading data from any or all floppies or hard disk(s) recognized by the BIOS, independent of the setting of the root device.**
- **Detect all installed ram**
- **GRUB can generally find all the installed ram on a PC-compatible machine.**

# Feature

# 1. Support Logical Block Address mode

*In traditional disk calls (called CHS mode), there is a geometry translation problem, that is, the BIOS cannot access over 1024 cylinders, so the accessible space is limited to at least 508 MB and to at most 8GB. GRUB can't universally solve this problem, as there is no standard interface used in all machines.However, several newer machines have the new interface, Logical Block Address (LBA) mode. GRUB automatically detects if LBA mode is available and uses*

*it if available. In LBA mode, GRUB can access the entire disk.*

- **Support network booting .GRUB is basically a disk-based boot loader but also has network support.**

- **Support remote terminals.**

*To support computers with no console, GRUB provides remote terminal support, so that you can control GRUB from a remote host. Only serial terminal support is implemented at the moment.*

- For example, under Linux the following will install GRUB into the

  MBR of the first IDE disk:

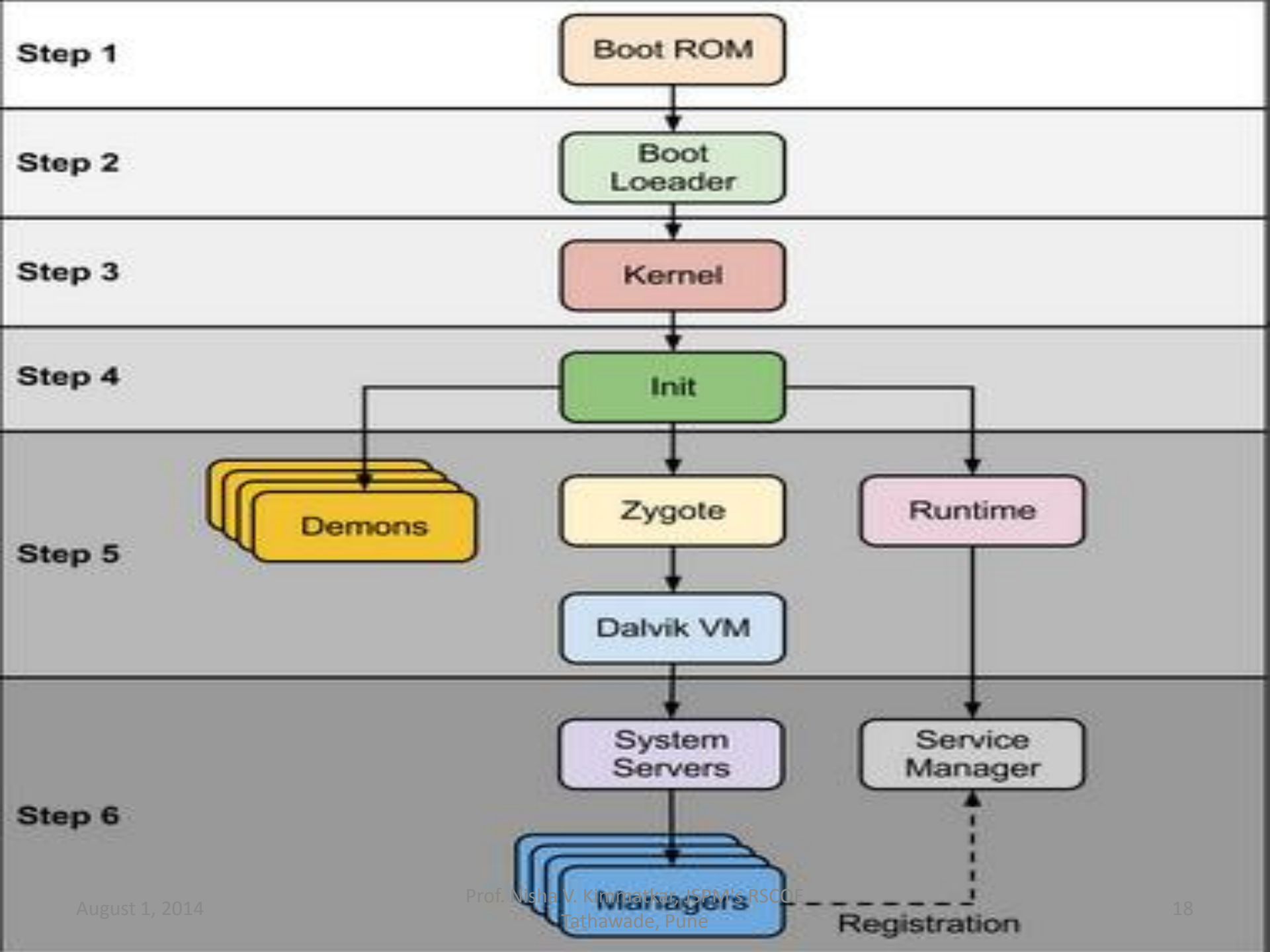
  # grub-install /dev/had

- *But the above examples assume that GRUB should put images under the '/boot' directory. If you want GRUB to put images under a directory other than '/boot', you need to specify the option '--boot-directory'. The typical usage is that you create a GRUB boot floppy with a file system. Here is an example:*
- **# mke2fs /dev/fd0**
- **# mount -t ext2 /dev/fd0 /mnt**
- **# mkdir /mnt/boot**
- **# grub-install --boot-directory=/mnt/boot /dev/fd0**
- **# umount /mnt**

# Android mobile init() Process

1. **Android is linux based open source operating system, x86 (x86 is a series of computer microprocessor instruction set architectures based on the Intel 8086 CPU.)**
2. **linux kernel is deployed, all Android devices are running on ARM process (ARM - Advanced RISC Machine)**

# Android mobile booting sequence


Step 1
Boot ROM
Step 2
Boot Loeader
Step 3
Kernel
Step 4
Init
Step 5
Demons
Zygote
Runtime
Dalvik VM
Step 6
System Servers
Service Manager
Managers
Registration

# Step 1 : Power On and System Startup

- When power start Boot ROM code start execution from pre defined location which is hardwired on ROM.
- It load Bootloader into RAM and start execution.

# Step 2 : Bootloader

- Bootloader is small program which runs before Android operating system running.
- Bootloader is first program to run so It is specific for board and processor.
- Device manufacturer either use popular bootloaders like redboot,uboot, qi bootloader or they develop own bootloaders.

- **Bootloader perform execution in two stages,**

a. **first stage It to detect external RAM and load program which helps in second stage,**
b. **In second stage bootloader setup network, memory, etc. which requires to run kernel, bootloader is able to provide configuration parameters or inputs to the kernel for specific purpose.**

- Android bootloader can be found at

**<Android Source>\bootable\bootloader\legacy\usbloaderlegacy**

loader contain two important files that need to address here.

1. init.s - Initializes stacks, zeros the BSS segments, call _main() in main.c
2. main.c - Initializes hardware (clocks, board, keypad, console), creates Linux tags

# Step 3: Kernel

1. Android kernel start similar way as desktop linux kernel starts.
2. As kernel launch it start setup cache, protected memory, scheduling, loads drivers.
3. When kernel finish system setup first thing it look for “init” in system files and launch root process or first process of system.

# Step 4: init process

- init it very first process, we can say it is root process or grandmother of all processes.
- init process has two responsibilities

  **1. mount directories like /sys, /dev, /proc and**
  **2. run init.rc script.**

- init process can be found at init :

<android source>/system/core/init

- init.rc file can be found in source tree at <android source>/system/core/rootdir/init.rc
- readme.txt file can be found in source tree at <andorid source>/system/core/init/readme.txt

# Step 5: Zygote and Dalvik

- **In a Java, We know that separate Virtual Machine(VMs) instance will popup in memory for separate per app, In case of Android app should launch as quick as possible, If Android os launch different instance of Dalvik VM for every app then it consume lots of memory and time. so, to overcome this problem Android OS as system named “Zygote”.**
- **Zygote enable shared code across Dalvik VM, lower memory footprint and minimal startup time.**

- **Zygote is a VM process that starts at system boot time as we know in previous step. Zygote preloads and initialize core library classes.**
- **Normally there core classes are read-only and part of Android SDK or Core frameworks. In Java VM each instance has it’s own copy of core library class files and heap objects.**

# Dalvik

- **Dalvik is the process virtual machine (VM) in Google's Android operating system.**
- **It is the software that runs the applications on Android devices.**
- **Dalvik is thus an integral part of Android, which is typically used on mobile devices such as mobile phones and tablet computers as well as more recently on embedded devices such as smart TVs and media streamers.**

- Programs are commonly written in Java and compiled to bytecode for the Java virtual machine, which is then translated to Dalvik bytecode and stored in .dex (Dalvik EXecutable) and .odex (Optimized Dalvik EXecutable) files, giving rise to the related terms odex and de-odex. The compact Dalvik Executable format is designed for systems that are constrained in terms of memory and processor speed.
- Dalvik is open-source software. It was originally written by Dan Bornstein, who named it after the fishing village of Dalvík, Iceland.

# Step 6: System  Service or Services

- After complete above steps, runtime request Zygote to launch system servers. System Servers are written in native and java both, System servers we can consider as process, The same system server is available as System Services in Android SDK. System server contain all system services.

# Core Services:

1. Starting Power Manager
2. Creating Activity Manager
3. Starting Telephony Registry
4. Starting Package Manager
5. Set Activity Manager Service as System Process
6. Starting Context Manager
7. Starting System Context Providers
8. Starting Battery Service
9. Starting Alarm Manager
10. Starting Sensor Service
11. Starting Window Manager
12. Starting Bluetooth Service
13. Starting Mount Service

# Other services

1. Starting Status Bar Service
2. Starting Hardware Service
3. Starting NetStat Service
4. Starting Connectivity Service
5. Starting Notification Manager
6. Starting DeviceStorageMonitor Service
7. Starting Location Manager
8. Starting Search Service
9. Starting Clipboard Service
10. Starting Checkin Service
11. Starting Wallpaper Service
12. Starting Audio Service
13. Starting HeadsetObserver
14. Starting AdbSettingsObserver

# Step 7 : Boot Completed

- Once System Services up and running in memory, Android has completed booting process, At this time "ACTION_BOOT_COMPLETED" standard broadcast action will fire.