Booting Linux system

Contents

1. The overview of booting
2. Kernel loading
3. Starting system services – run level
4. Initing the working environment
5. Operations with processes during booting progress
6. Manage the booting and process with systemd

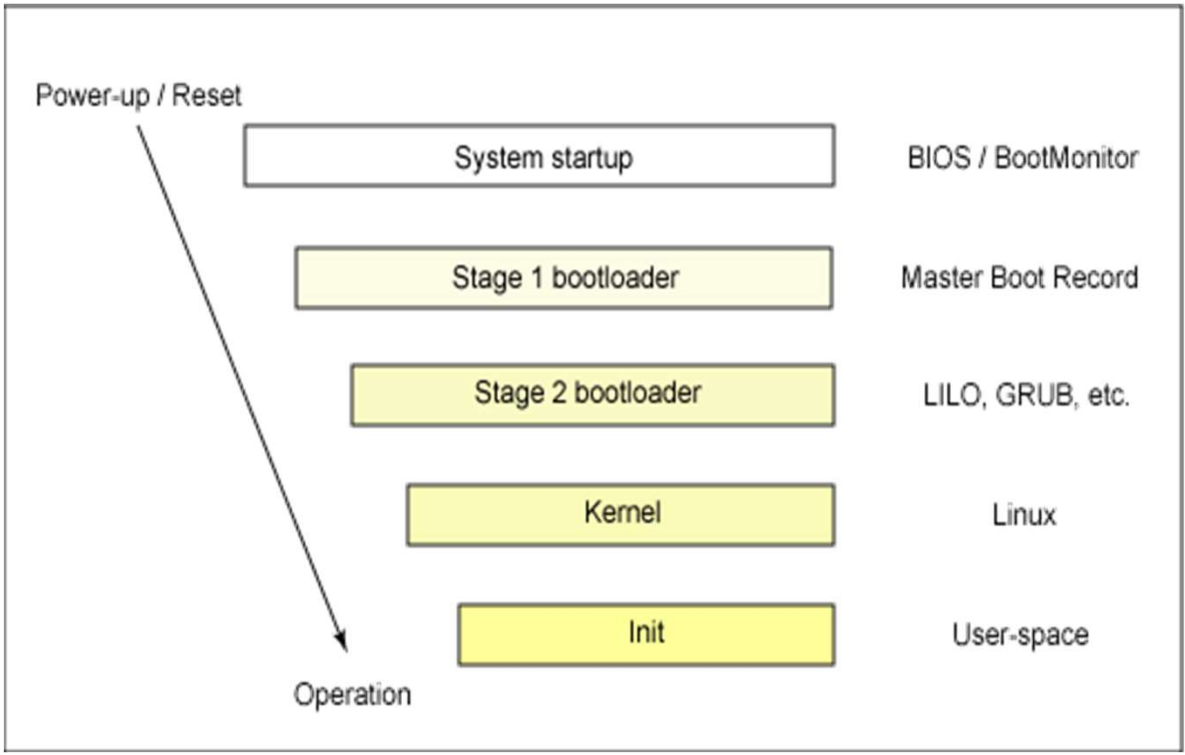
1. The overview of booting

The goals of booting

- Start up hardware
- Check device status
- Boot up applications for users

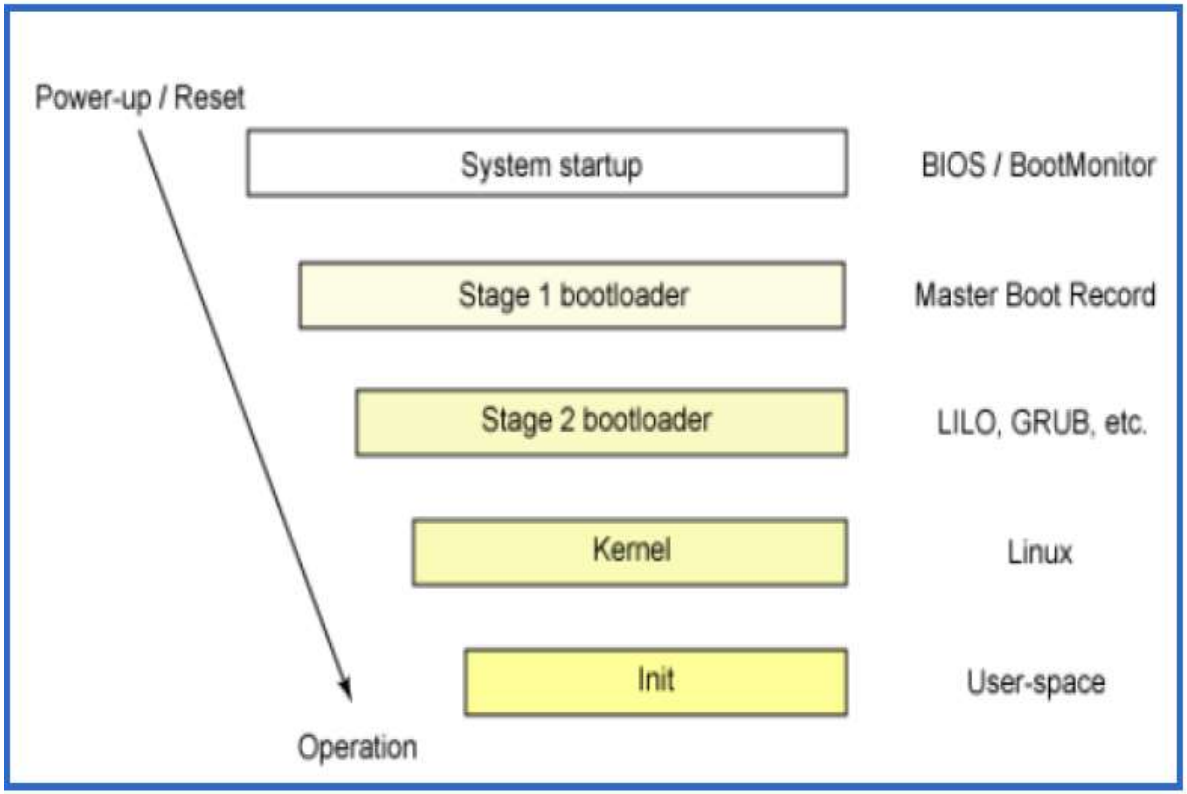
Specifically, when booting a PC

- Start up hardware
- Start up MBR
- Execute booting software (OS menu)
- Start up OS kernel
- Start up user applications
- Depending on systems, some stages can be combined together

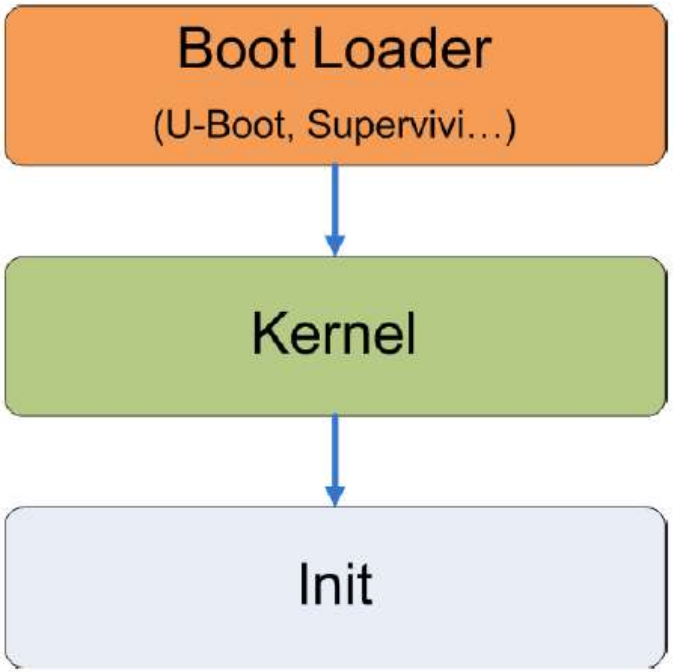


Booting progress in PC and embedded Linux

Boot Linux on PC



Boot Linux on embedded systems



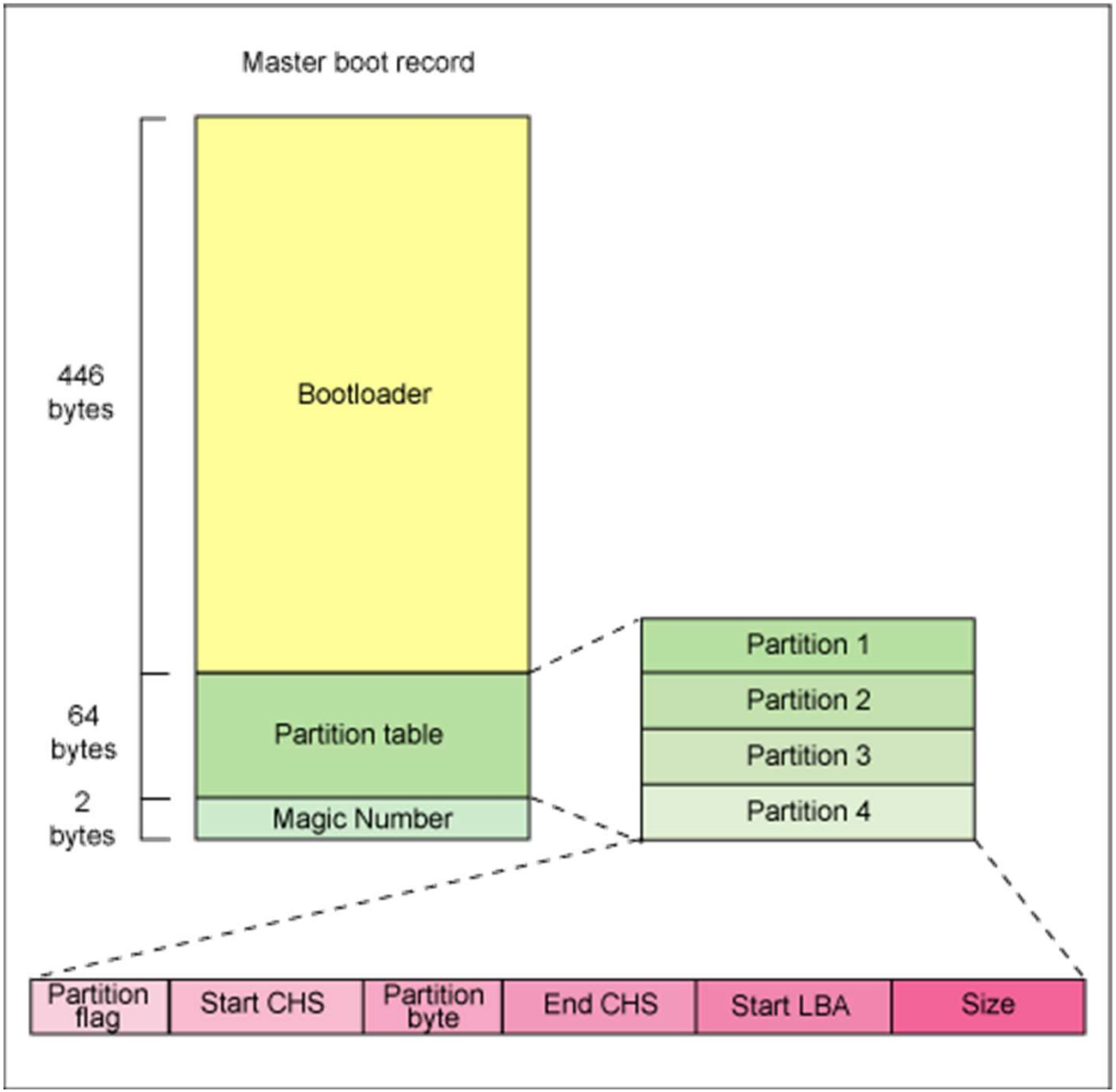
Booting hardware devices

- Depending on physical systems
- On PC: BIOS
  - POST
  - Allocate and mark peripherals
  - Determine the booting device
  - Execute MBR
  - MBR
    - Bootloader
    - Partition Table: how the logical partitions are organized
  - Execute boot record

MBR-Master Boot Record

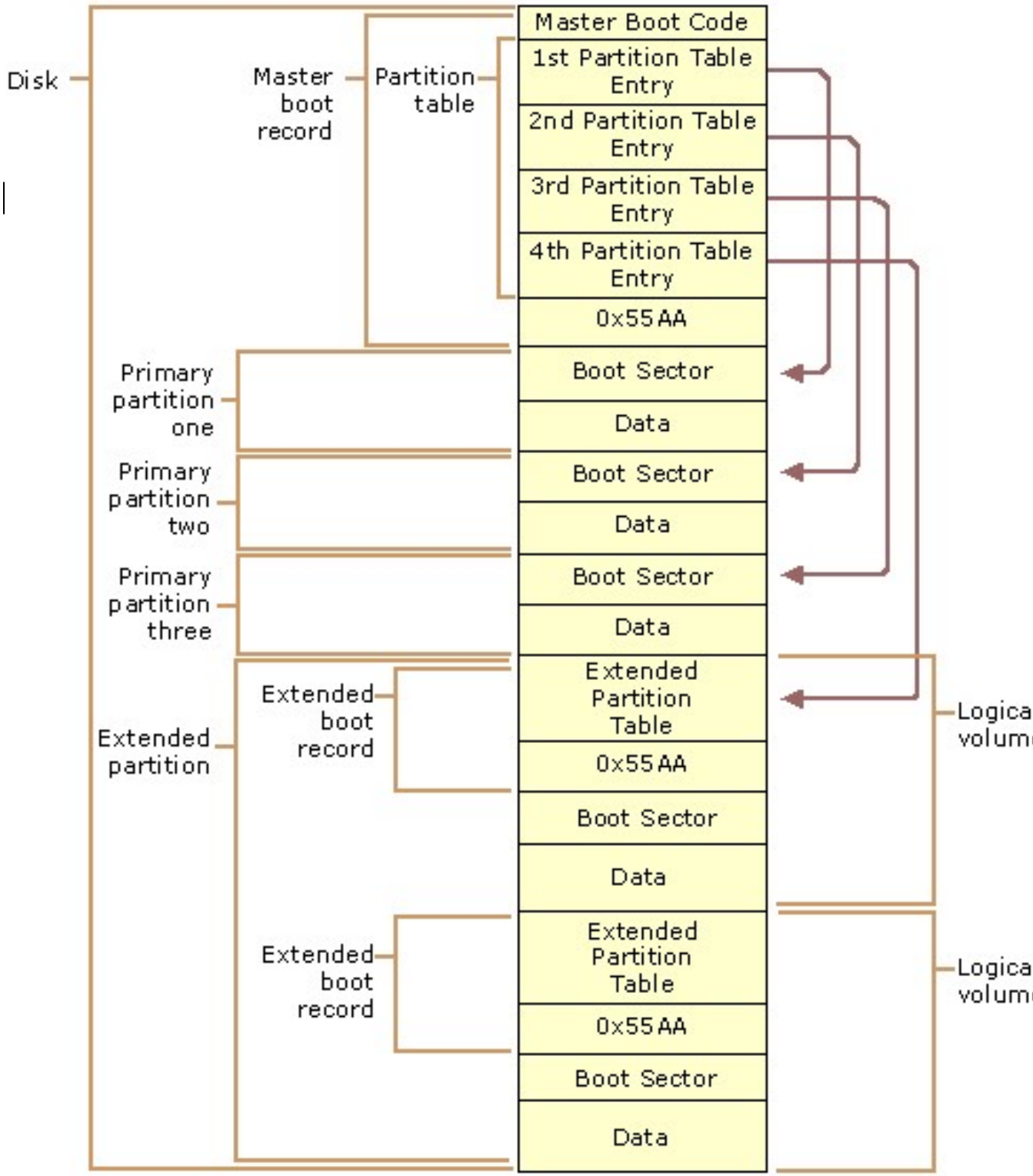
- MBR the first sector of a physical hard drive
- MBR is outside of Partition Table
- MBR:
  - Contains Partition Table
  - Contains bootloader
- Each partition has its own Boot Record, containing script/program to start up OS inside the partition table

MBR-Master Boot Record



Example of a complex MBI

1. Load partition table of the Active partition
2. Find the first sector of the active partition
3. Load first sector into memory
4. Switch the control to the bootloader



Notes

- Each physical disk can have up to 4 primary partitions
- One of 4 primary partitions can be converted to many logical partitions

Bootloader

- A small program to load OS kernel
- Location
  - 1st sector of HDD: 1st stage boot loader, in MBR
  - 1st sector of each partition : 2nd stage boot loader.
- Functions
  - Load OS kernel to memory
  - Load 2nd stage bootloader to memory
  - Call bootloader in boot sector of other partitions.
- Simple
  - No authentication
  - No protection (Boot sector virus)

Bootloader

- Main bootloaders by architecture:
  - x86 => GRUB / LILO
  - ARM => U-Boot
  - PowerPC => U-Boot
  - MIPS => U-Boot
  - SuperH => U-Boot
  - M68k => U-Boot

2. Load OS kernel

- MBR or boot sector can directly load OS kernel
  - Only use simple and low-level disk-reading operations
  - Cannot read big files, complex locations (Ex: LBA)
- Practical method
  - MBR loads a small program (but larger than MBR) and let this program load OS kernel
  - More complex, more steps
  - OS kernel can be more sophisticated
- Example: ntosloader, lilo, grub

Lilo Boot Loader

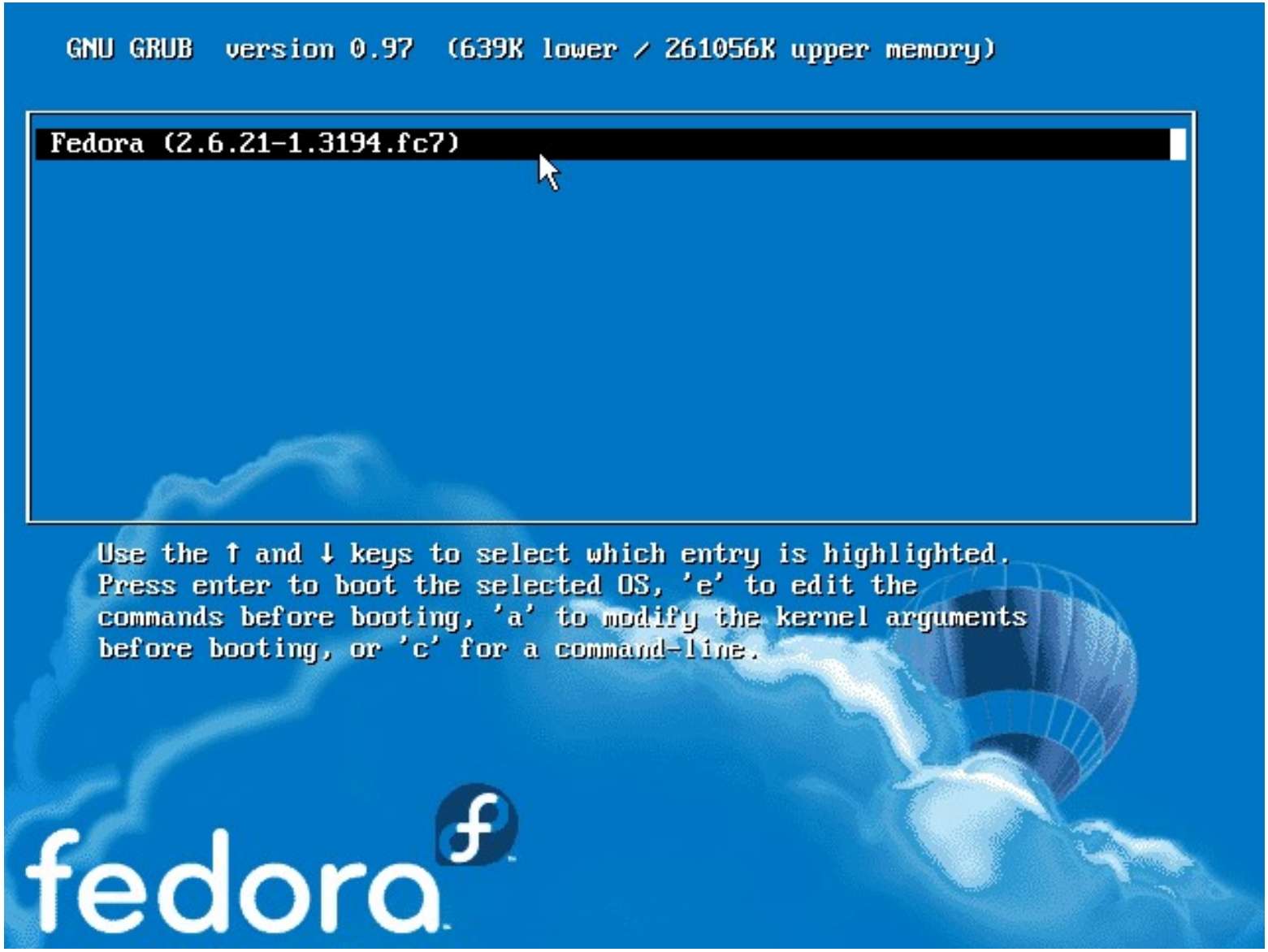
```
boot = /dev/hda #boot loader ở MBR
delay = 40
compact
vga = normal
root = /dev/hda1
read-only
image = /zImage-2.5.99
        label = try # tên ở menu khởi
động
image = /zImage-1.0.9
        label = 1.0.9
other = /dev/hda3
        label = dos
        table = /dev/hda
```

- Location: MBR of HDD or first sector of a partition
- To simplifty, OS kernel is stored in /boot
- Allow to select OS to boot
- Lilo configuration
  - /etc/lilo.conf
  - Use lilo command to
    - Read configuration file
    - Write changes to MBR
  - Can check the configuration before writing

LILO Boot step

- L- Loader OK
- LI- Second stage Loader OK
- LIL? Found Kernel but cannot load
- LIL- Wrong kernel format
- LILO- Sucessful

Grub bootloader



Grub bootloader

- Grand Unified Bootloader
- At MBR
- Use to load grub loader
- Grub loader loads kernel OS in /boot

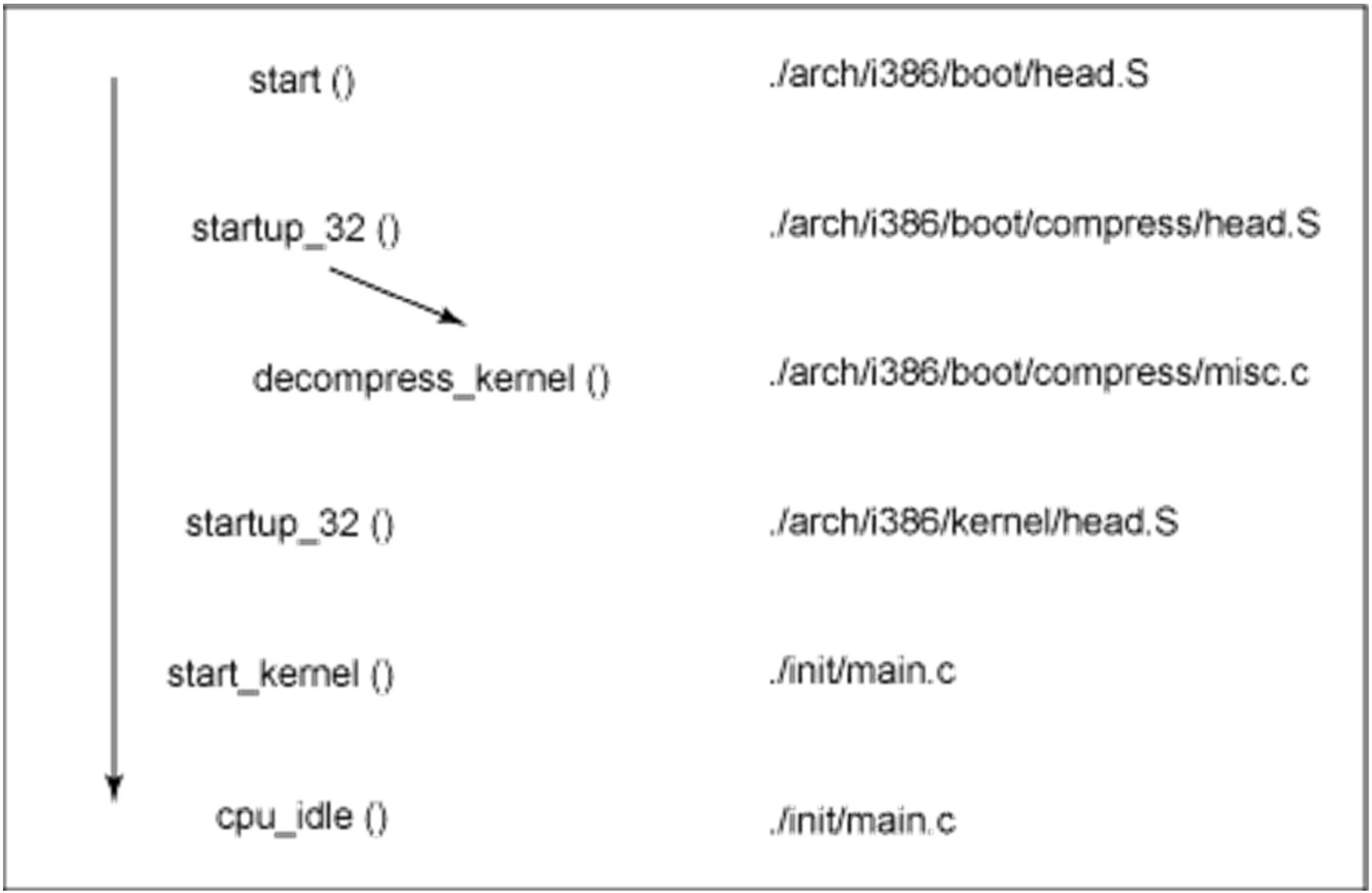
Grub configuration

- Grub 2: /boot/grub2/grub.conf
- MBR doesn’t change after modifying /boot/grub2/grub.conf
- The updating progress will be done by the grub of step 2
- Allow to change parameters while booting

Booting parameters

- Vga: text screen while booting
- Root: the disk of root directory /
- Label: Name of OS when booting
- Other parameters

Kernel boot


start ()
./arch/i386/boot/head.S
startup_32 ()
./arch/i386/boot/compress/head.S
decompress_kernel ()
./arch/i386/boot/compress/misc.c
startup_32 ()
./arch/i386/kernel/head.S
start_kernel ()
./init/main.c
cpu_idle ()
./init/main.c

2. SysVInit and run-level

- Run-level
  - After loading OS kernel, some tasks will be executed
  - The first one is init
  - Other tasks/processes will be loaded according to user requirements
  - There are 6 different run-levels
  - Each run-level includes different activated tasks

Run-level in Debian distributions

Run-level	Description
0	Halt
1	Single user, no graphic, no network
2-5	Multiple user, graphic, network
6	Restart

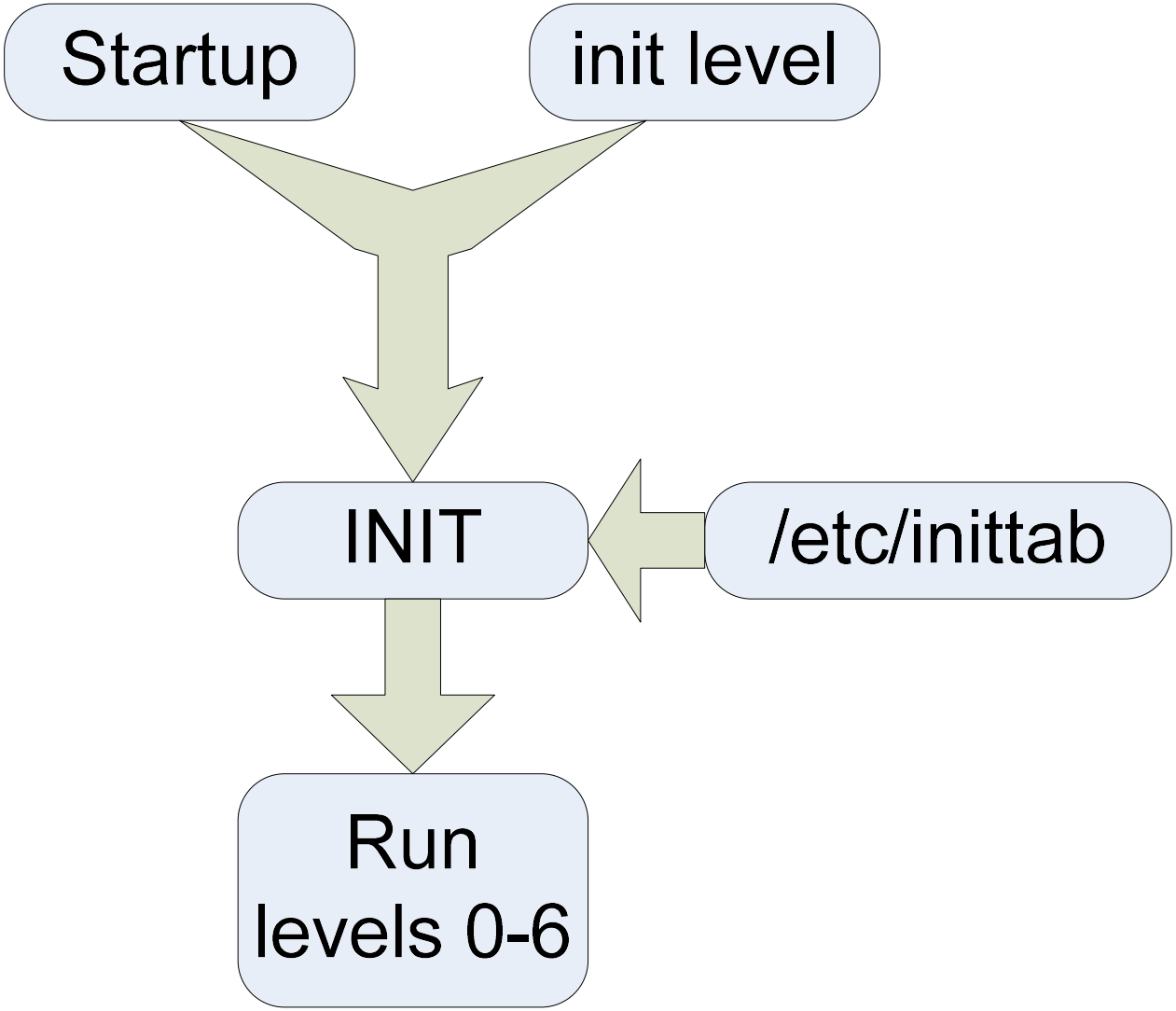
Run-level in Redhat distributions

Run-level	Description
0	Turn off
1	Single user, no graphic, no network, no service
2	Multiple user, no graphic, no network
3	Multiple user, no graphic with network
4	No use
5	Multiple user, graphic, network
6	Restart
S	Single user, no graphic, no network, no service

Manage run-levels

- runlevel: find the current and previous runlevels
- init
- telinit: change system run-level
- initctl: allows a system administrator to communicate and interact with the Upstart init(8) daemon

Manage run-levels



inittab

ID : runlevel : action : command
(1) (2) (3) (4)

	Field	Description
(1)	ID	Identifier uniquely assigned to the entry.
(2)	Runlevel	Runlevel in which the description of the entry is reflected. The action of this entry will be executed in all runlevels when omitting it.
(3)	Action	Definition that shows how process of the entry is executed sysinit : Execute it before accessing the console. initdefault : Runlevel of the default to give to 'init' powerfail : Execute when it receive the 'POWER FAIL' signal. wait : Wait for the termination of the process. respawn : Keep the started process status constant.
(4)	Command	Command to be executed

Most of current Linux distribution abandoned init / inittab !!!!!!!!

Why not init / inittab?

- Init is a process with PID=1
- If init cannot start, the system will fall to “kernel panic” state.
- A starup process can only be started after the previous one was successfully loaded → Slow and unstable booting
- Replacement
  1. **Upstart** – WAS used for Ubuntu GNU/Linux (support up to Ubuntu 16). It was designed to start up asynchronously.
  2. **Epoch** – was built around the simplification of managing services and processes.
  3. **Mudar** – was written on Python, is used for Pardus GNU/Linux, is designed to start up asynchronously.
  4. **systemd** – was designed to run process parallel, is used for many current Linux systems such as Ubuntu, Fedora, OpenSuSE, Arch, RHEL, CentOS, etc.

Target in systemd

- Target is used to group services to synchronous points for services
- Replace runlevel
- Each target is considered as an OS state and services will be activated depending on OS states

Runlevel at systemd

- Target system: set of OS states and regulation of services and processes need to be run at that state
- New targets are equivalent with old runlevel
  - poweroff.target *(runlevel 0)*: Power off
  - rescue.target *(runlevel 1)*: start rescue shell
  - multi-user.target *(runlevel 2,3,4)*: multiple users at console mode
  - graphical.target *(runlevel 5)*: graphical mode and network service
  - reboot.target *(runlevel 6)*: restart system

Examples

```
[intran@localhost init.d]$ systemctl list-units --type target
UNIT                    LOAD   ACTIVE SUB    DESCRIPTION
basic.target            loaded active active Basic System
cryptsetup.target       loaded active active Local Encrypted Volumes
getty.target            loaded active active Login Prompts
graphical.target        loaded active active Graphical Interface
local-fs-pre.target     loaded active active Local File Systems (Pre)
local-fs.target         loaded active active Local File Systems
multi-user.target       loaded active active Multi-User System
network-online.target   loaded active active Network is Online
network-pre.target      loaded active active Network (Pre)
network.target          loaded active active Network
nss-user-lookup.target loaded active active User and Group Name Lookups
paths.target            loaded active active Paths
remote-fs.target        loaded active active Remote File Systems
slices.target           loaded active active Slices
sockets.target          loaded active active Sockets
sshd-keygen.target      loaded active active sshd-keygen.target
swap.target             loaded active active Swap
sysinit.target          loaded active active System Initialization
timers.target           loaded active active Timers
```

Structure of a service in systemd

- Each service is represented by a “service unit” (with the tail of name as .service), saved in /lib/systemd/system/ (or /run/systemd/system or /etc/systemd/system)
  - etc > run > lib : priority
- Content of a service service-name.service is similar to a INI file in Windows
- Each service’s configuration can be modified by file *service-name.service.d/*.conf* in the same folder
- Change files of a service
  - systemctl edit [--full] application
  - Note: while modifying, the directory with the name “*.d” will be created inside /…/systemd/system for each service. Removing this folder means deleting modified configurations
  - systemctl daemon-reload

How to create a service with systemd

- Create a file inside the directory systemd/system under a service-name.service with the following format

```
[Unit]
Description=<description about this service>

[Service]
User=<user e.g. root>
WorkingDirectory=<directory_of_script e.g. /root>
ExecStart=<script which needs to be executed>
Restart=always

[Install]
WantedBy=multi-user.target
```

- Reload the daemon service by systemctl
  - sudo systemctl daemon-reload
- Restart the service
  - sudo systemctl start your-service.service

Example of sshd.service

```
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.target
Wants=sshd-keygen.target

[Service]
Type=notify
EnvironmentFile=-/etc/crypto-policies/back-ends/opensshserver.config
EnvironmentFile=-/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS $CRYPTO_POLICY
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target
```

systemctl

- The command is used to control other parts of systemd
- Only root or users with root privilege (through sudo)
- Simplify the configuration of service/ application while booting
- Manage all through UNIT

Some commands of systemctl

- List information
  - systemctl [list-units] [--all] [--state=inactive]
  - systemctl [list-units] [--all] [--type=service]
  - systemctl list-unit-files
- Manage information of UNIT
  - systemctl cat UNIT
  - systemctl list-dependencies UNIT
  - systemctl show sshd.service
- Some commands to change runlevel
  - systemctl rescue/halt/poweroff/reboot

Manage services/programs with systemd

- systemctl [start/stop/restart/reload] application
  - start/stop/restart/reload a service
  - Restart vs reload?
  - Nor sure? → reload-or-restart
- systemctl [enable/disable] application
  - enable/disable a application while booting
- systemctl status application.service
- systemctl [is-active/is-enabled/is-failed] application.service