Advanced Linux System Administration

Topic 3. Booting & shutting down

Pablo Abad Fidalgo
José Ángel Herrero Velasco

Departamento de Ingeniería Informática y Electrónica

Index

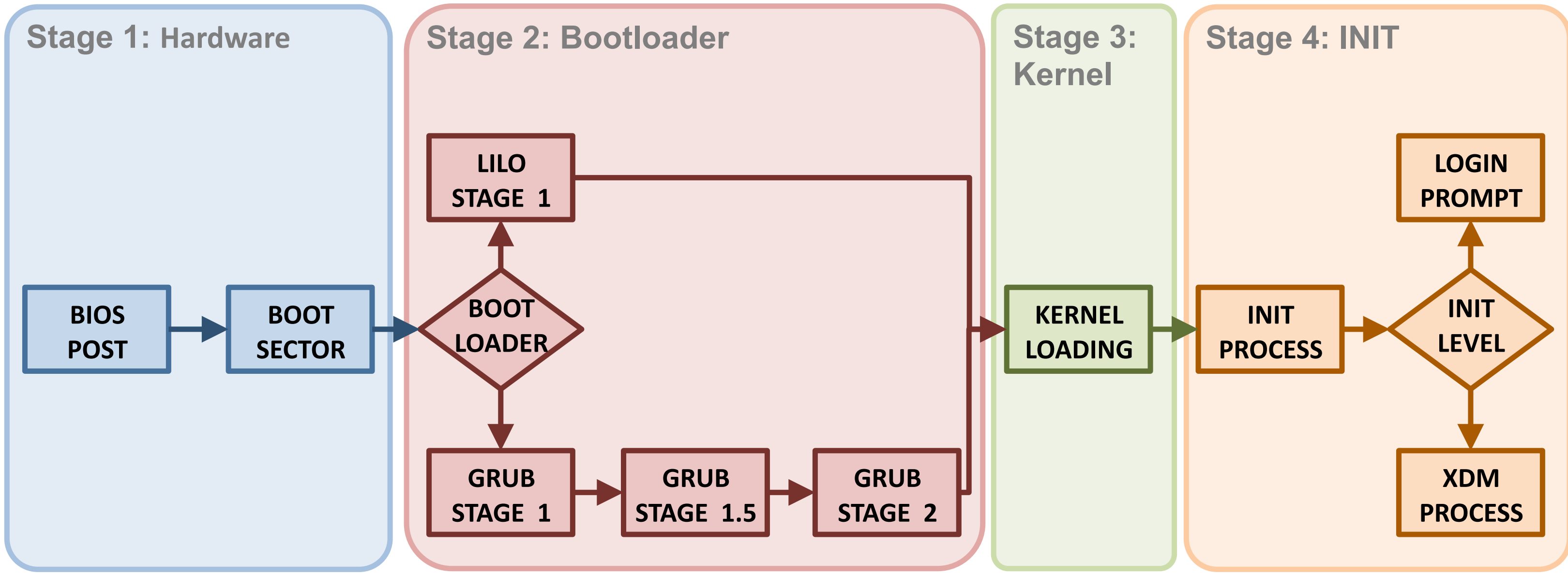
- Introduction.
- Booting, Stage 1: Hardware.
- Booting, Stage 2: Bootloader:
  - LILO.
  - GRUB.
- Booting, Stage 1+2 (UEFI).
- Booting, Stage 3: Kernel.
- Booting, Stage 4: SysV / Systemd.
- Shutting Down.

Introduction

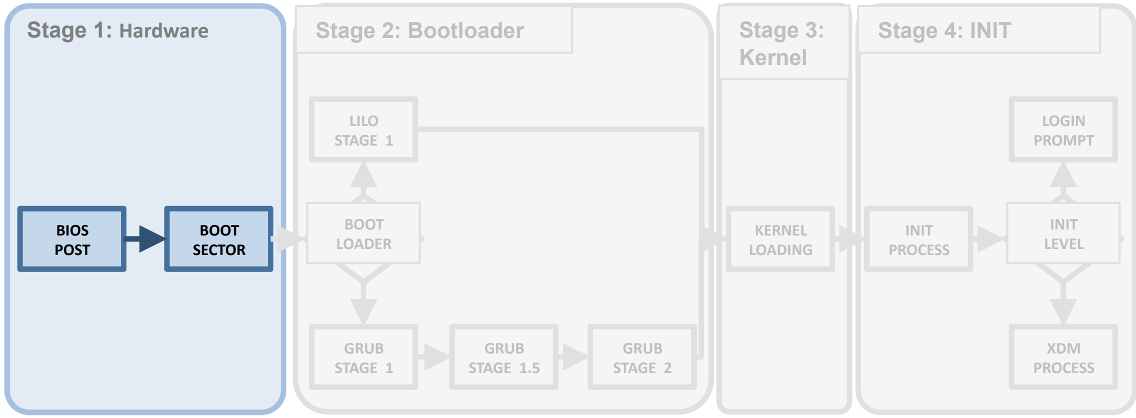
- Booting/Shutting Down are complex procedures, but system provides mechanisms to deal with them.
- …However, this is one of the potential troubles of administration.
- Goals of this Chapter:
  - To understand the basic operation of both procedures.
  - Being able to customize them.
  - Being able to solve generic problems related to Boot process.
- **Bootstrapping**. Where does the name come from?:
  - Allusion to “Baron Münchausen”.
  - Defines a process where a simple system starts up another one with higher complexity (starting the system forms a small portion of the system itself).

Introduction

- The main objective of the Booting process is to load the kernel in the memory and to start executing it:
  - Where is the kernel before booting?
  - What's the memory content before booting?
- It is a sequential process divided into 4 stages:

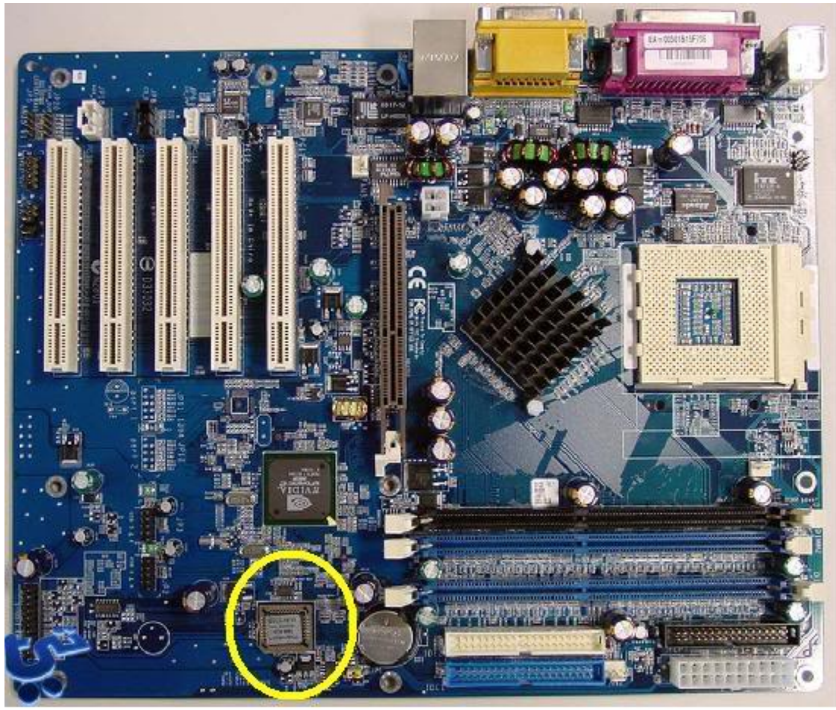


Index



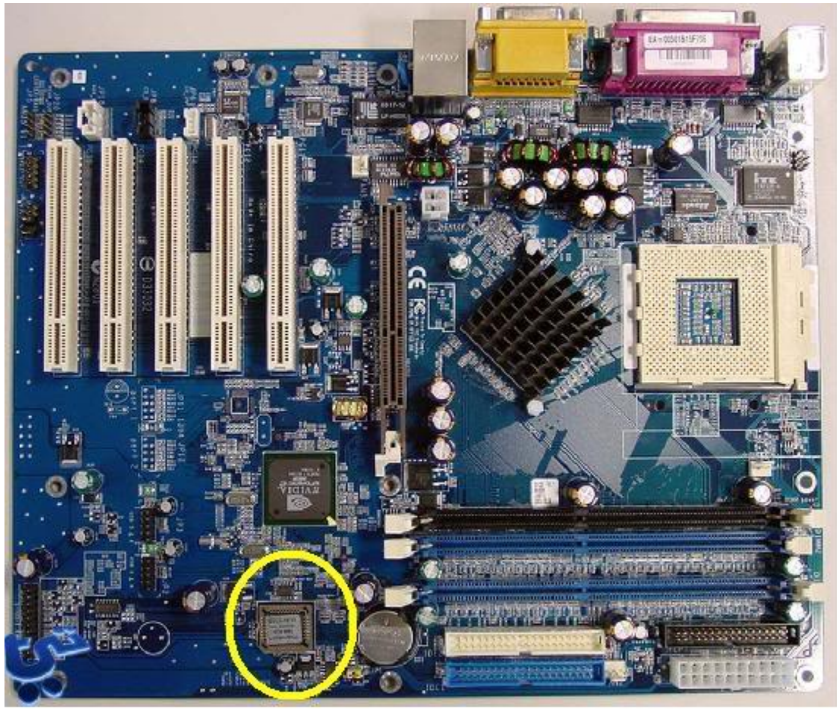
- Introduction.
- **Booting, Stage 1: Hardware.**
- Booting, Stage 2: Bootloader:
  - LILO.
  - GRUB.
- Booting, Stage 1+2 (UEFI).
- Booting, Stage 3: Kernel.
- Booting, Stage 4: SysV / Systemd.
- Shutting Down.

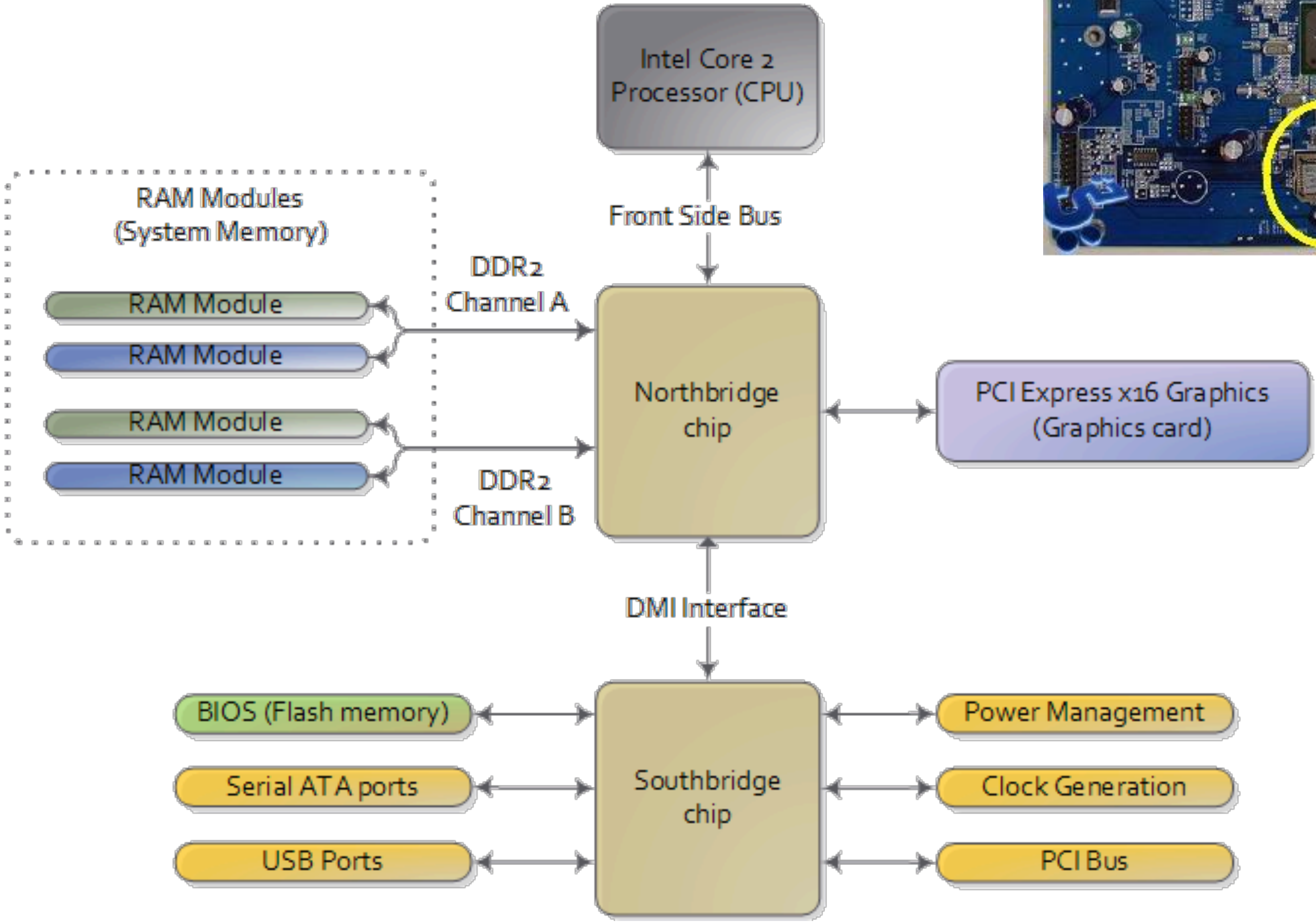
Stage 1: Hardware



- First Steps:
  - After pushing the Power-On button, the **Reset Vector** tells the CPU the address of the first instruction to be executed (FFFFFFF0h for x86).
  - Such direction corresponds to an EPROM/Flash (motherboard) that stores the code corresponding to the Firmware (memory-mapped I/O).
- Firmware:
  - Stores Hardware configuration for the system.
  - Some configuration parameters with its own power supply (battery).
- Want detailed description? (hardcore…):
  - http://www.drdobbs.com/parallel/booting-an-intel-architecture-system-par/232300699.

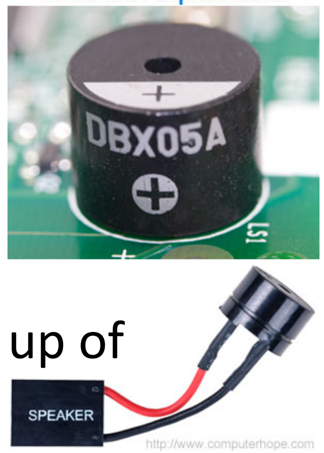
Stage 1: Hardware





Stage 1: Hardware

Internal Speakers

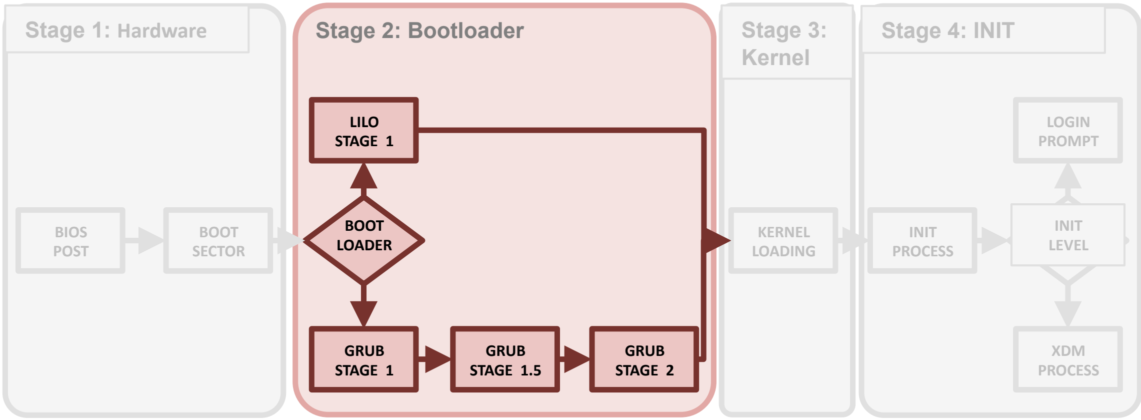


- Tasks to be performed:
  - **Power-on-self-test** (POST): examination, verification and start up of hardware devices (CPU, RAM, Controllers, etc.).
  - Configuration of previous aspects, independent of OS (Virtualization extensions, security, etc.).
  - Starting up the Operating System: in the case of BIOS, look for the OS loader in the first block (512 bytes) [**Master Boot Record (MBR)**] from the booting device in the configured order. When found, the contents are loaded into the memory.
- Two main kinds of Firmware:
  - BIOS: Basic Input Output System.
  - EFI: Extensible Firmware Interface.

Stage 1: Hardware

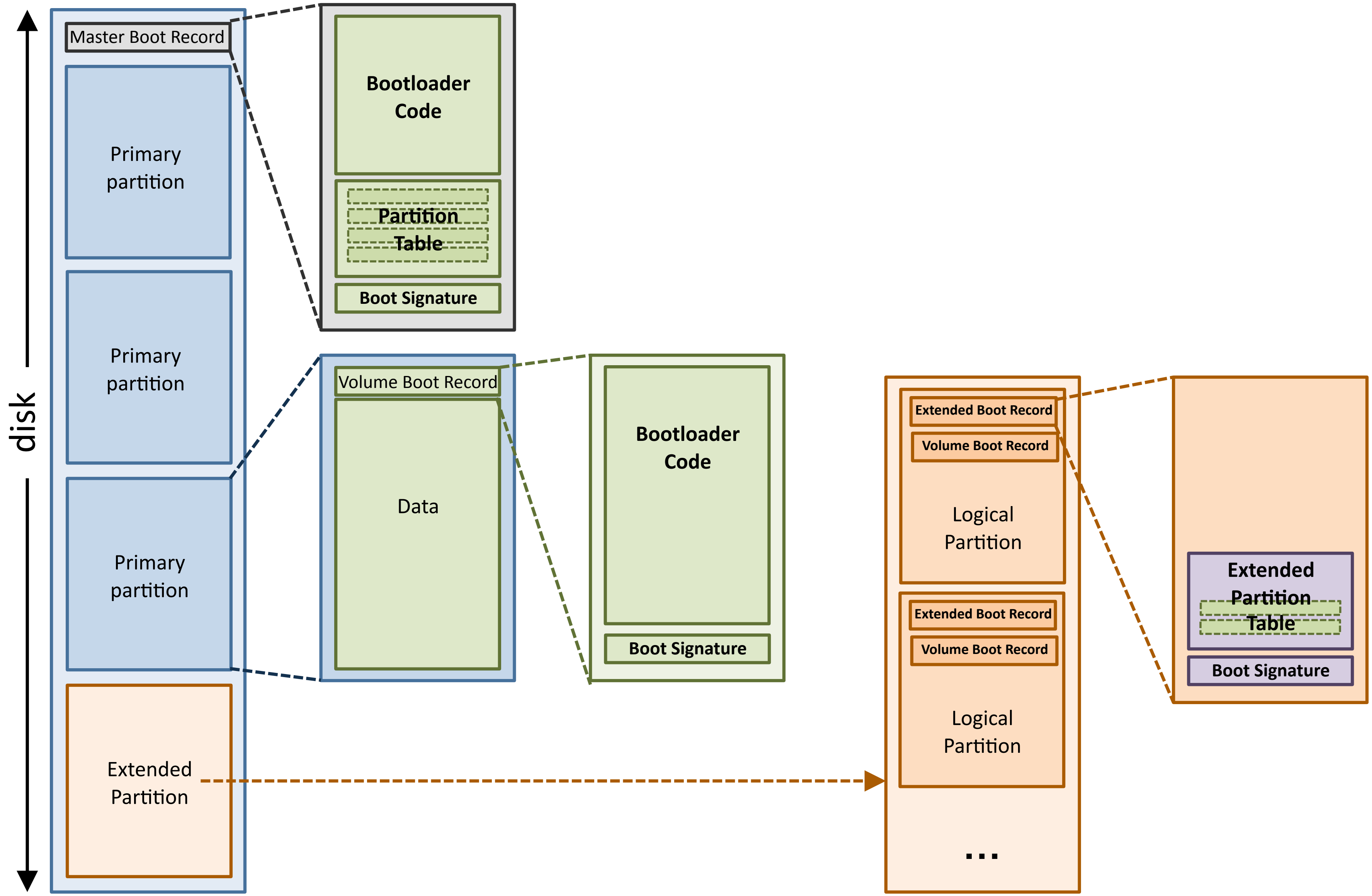
- **BIOS** (Basic Input/Output System):
  - 1975: first appeared in the CP/M Operating System.
  - It runs in real address mode (16 bit): 1MB of addressable memory.
  - 1990: “BIOS setup utility” appears: allows the user to define some configuration options (boot priority).
  - ROM customized for a particular HW. Provides a small library with I/O functions to work with peripherals (keyboard, screen). Very slow (protected to real mode).
  - Emerging applications require more and more BIOS support: security, temperature/power metrics (ACPI), virtualization extensions, turbo-boost… (hard to put all that in 1MB).
  - 2002: intel develops an alternative firmware: EFI (/UEFI).

Index



- Introduction.
- Booting, Stage 1: Hardware.
- **Booting, Stage 2: Bootloader:**
  - **LILO.**
  - **GRUB.**
- Booting, Stage 1+2 (UEFI).
- Booting, Stage 3: Kernel.
- Booting, Stage 4: SysV / Systemd.
- Shutting Down.

Previous: MBR Disks & Partitions



Previous: MBR Disks & Partitions

- **Master Boot Record (MBR):**
  - First block of the Disk, 512 Bytes.
  - **Partition Table:** information about four primary partitions: begin and end blocks, size, etc. (64 bytes).
  - **Boot Signature:** numerical value indicating the presence of valid bootloader code in the code field (0x55AA) (2 bytes).
- **Volume Boot Record (VBR):**
  - First block of each primary partition.
  - Could contain bootloader code (indicated by Boot Signature).
- **Extended Partition:**
  - Partition that can be sub-divided into multiple **logical partitions**.
  - **Extended Boot Record (EBR):** first block of each logical partition. It only contains a partition table with two fields. **Extended partition table** forms a linked list with all logical partitions.

Previous: MBR Disks & Partitions

- **Linux Naming Convention:**
  - Remember: I/O devices are treated as files. Under directory /dev we find all system disks.
  - Generic PC: 2 IDE controllers, each can have two devices (master/slave):
    - /dev/**hda:** first device (master) of the first IDE controller.
    - /dev/**hdb:** second device (slave) of the first IDE controller.
    - /dev/**hdc:** first device of the second controller.
    - /dev/**hdd:** second device of the second controller.
  - In a disk, each **primary partition** is identified with a number from 1 to 4:
    - /dev/**hda1:** first primary partition of the hda disk.
  - **Logical partitions** start from 5:
    - /dev/**hda5:** first logical partition of hda disk.
  - In **SCSI devices** same naming convention, changing “hd” to “sd”:
    - /dev/**sda1**.

Stage 2: Bootloader

- Hardware requires an OS in charge of providing all the functionality in a computer.
- Target: to load the OS kernel into memory and start running it. Loader with different locations: USB, CD, Disk…
- **Stage 2.1:**
  - Located in **MBR:** 512 first bytes (block 0) of the **active device**.
  - Loaded into memory by BIOS (Stage 1).
  - Triggers, when executed, the load and execution of Stage 2.2.
- **Stage 2.2:**
  - Located in the **active partition**, where the kernel is placed.
  - Loads the kernel into memory and transfers control to it (data initialization, drivers, check CPU, etc.).
  - After this process, the **init** process is executed (Stage 3).

Stage 2: LILO

- **LI**nux **Lo**ader:
  - Two stage Bootloader.
  - Does not “understand” about operating system or about file system. Only works with physical locations.
  - Obsolete (but easy to follow for academic purposes).
- Steps:
  - Master boot loads LILO from the first active partition and runs it:
    - LILO can be in the MBR or in the Boot Block of a primary partition. In the second case, MBR contains the necessary code to load LILO from another block.
  - LILO asks the user what kind of boot he wants (partition, kernel, mode). Through a prompt.
  - LILO loads the kernel and a ramdisk.
  - The kernel starts running once it is loaded into memory.

Stage 2: LILO

- Configuration: /etc/**lilo.conf:**

```
boot=/dev/hda #o by ID
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
linear
default=linux

image=/boot/vmlinuz-2.6.2-2
        label=linux
        read-only
        root=/dev/hda2 #o by UUID
        initrd=/boot/initrd-2.4.2-2.img

other=/dev/hda1
        label=dos
        optional
```

Device where LILO is installed (IDE/SATA/Floppy...).

File with information about disk blocks with the files required to boot system.

Loader Assembly code.

Kernel for booting and its options.

Linux system partition (/). Not necessarily a disk (usb loader).

Filesystem loaded into memory as a ramdisk. Software support not provided by the kernel to initialize the system.

Link to other loader (boot a different OS).

Stage 2: LILO

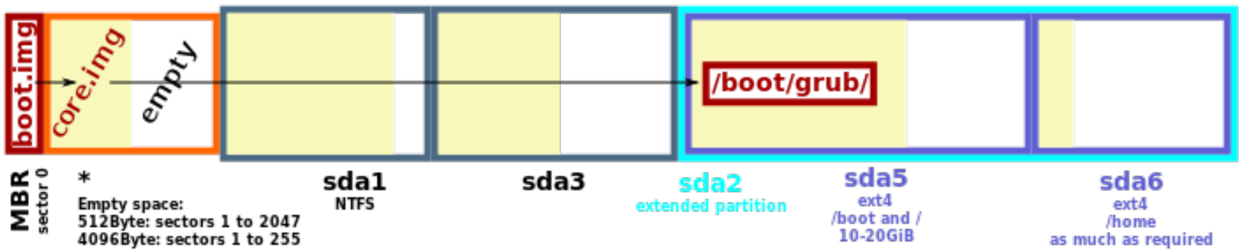
- Configuration: /etc/**lilo.conf:**
  - Any change in the files employed in boot process (boot.b, kernel, ramdisk) requires loader update:
    - Map file must reflect those changes, otherwise booting process is corrupted.
    - Check if map file is updated: # lilo -q.
    - Update map file: # lilo [-v].
- A booting error cannot be fixed from the shell…
- Possible error sources:
  - Installation of a new OS overwriting MBR (M$).
  - Failed kernel compilation.
  - Modification in boot files without map updating.
- Rescue Systems:
  - mkbootdisk.
  - Installation Live CD (option rescue) or specialized (SystemRescueCD).

Stage 2: GRUB/GRUB2

- **GR**and **U**nified **B**ootloader: linux loader:
  - Bootloader with three stages.
  - Can work with file systems (ext2, ext3, ext4…), directly accessing partitions (no map files).
  - UEFI version available (grub.efi).
  - Much more flexible, has its own mini-shell (grub>):
    - Booting parameters can be decided through that prompt. It is possible to indicate the kernel and the ramdisk before startup (booting an OS which was not in the boot menu).
    - "c" from the startup window opens the console with the values for the selected input.
    - "e" edits each input in n-curses format.
    - "kernel", "initrd" loads a kernel or a ramdisk.
    - "boot" boots your OS.
    - Access to the file system and command has auto-complete (TAB).
  - Currently GRUB2 is the most commonly used bootloader.

Stage 2: GRUB/GRUB2

- **GR**and **U**nified **B**ootloader:
  - Configuration:
    - More complex scripts than LILO. Advantage: modifications in files required to boot (kernel or initrd) are processed "automatically".
    - Everything in /etc/default/grub and /etc/grub.d/.
    - Final configuration (/boot/grub) is performed through the command "update-grub".
  - Stages:
    - Stage 1. Boot.img stored in MBR (or VBR), loaded into memory and executed (loads the first sector of core.img).
    - Stage 1.5. Core.img stored in the blocks between MBR and first partition (MBR gap), loaded into memory and executed. Loads its configuration file and drivers for the file system.
    - Stage 2. Load Kernel and ramdisk, accessing directly to the file system (/boot/grub).

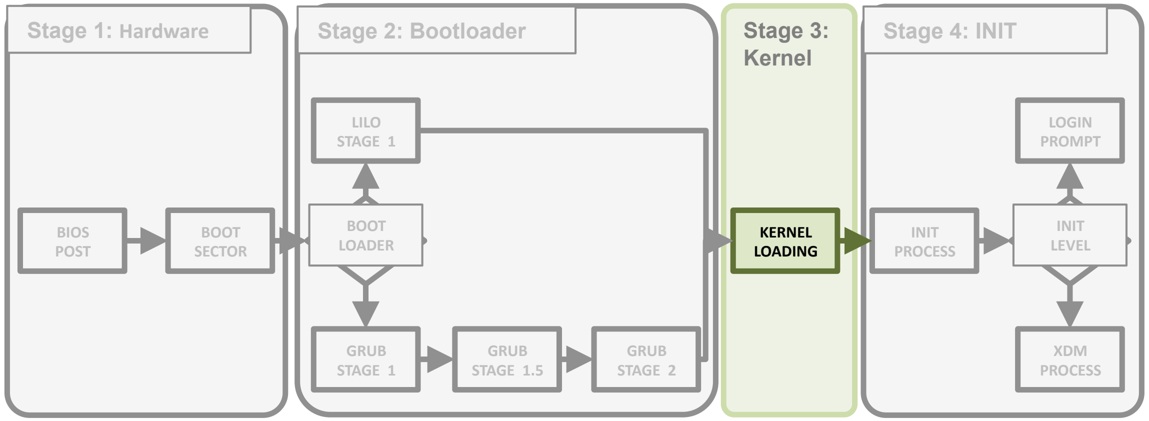


Stage 2: Bootloader

- Having physical access to a system, stages 1 & 2 can become a weakness:
  - Modifying boot options we could obtain superuser privileges.
- Protect BIOS and loader with password.
- Example: protection of GRUB2 with password:
  - Edit /etc/grub.d/00_header and at the end of the file add (remember to perform update-grub after that):

```
cat << EOF
set superusers="alumno"
password alumno <<<<<secuencia de grub-mkpasswd-pbkdf2>>> o <<password-plano>>
export superusers
EOF
```

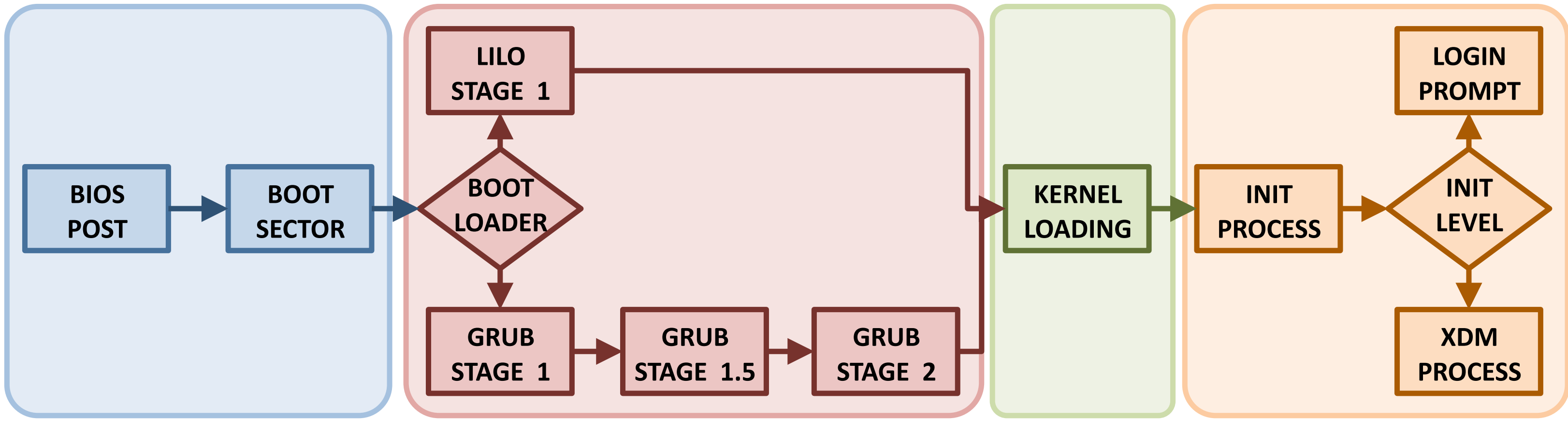
Index



- Introduction.
- Booting, Stage 1: Hardware.
- Booting, Stage 2: Bootloader:
  - LILO.
  - GRUB.
- **Booting, Stage 1+2 (UEFI).**
- Booting, Stage 3: Kernel.
- Booting, Stage 4: SysV / Systemd.
- Shutting Down.

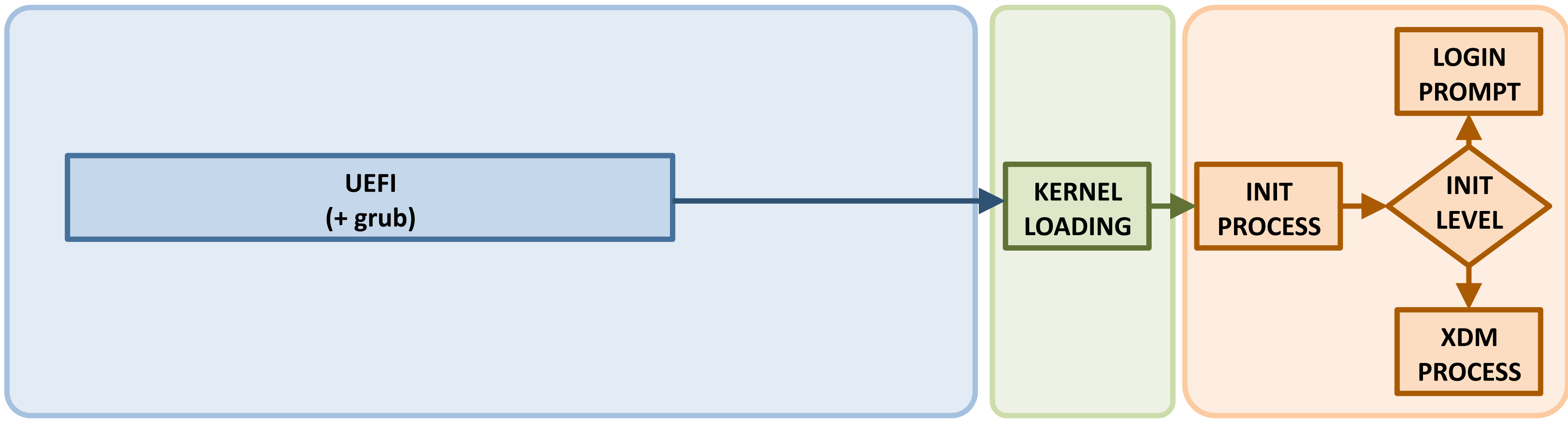
UEFI

- From:



BIOS POST
BOOT SECTOR
BOOT LOADER
LILO STAGE 1
GRUB STAGE 1
GRUB STAGE 1.5
GRUB STAGE 2
KERNEL LOADING
INIT PROCESS
INIT LEVEL
LOGIN PROMPT
XDM PROCESS

- To:

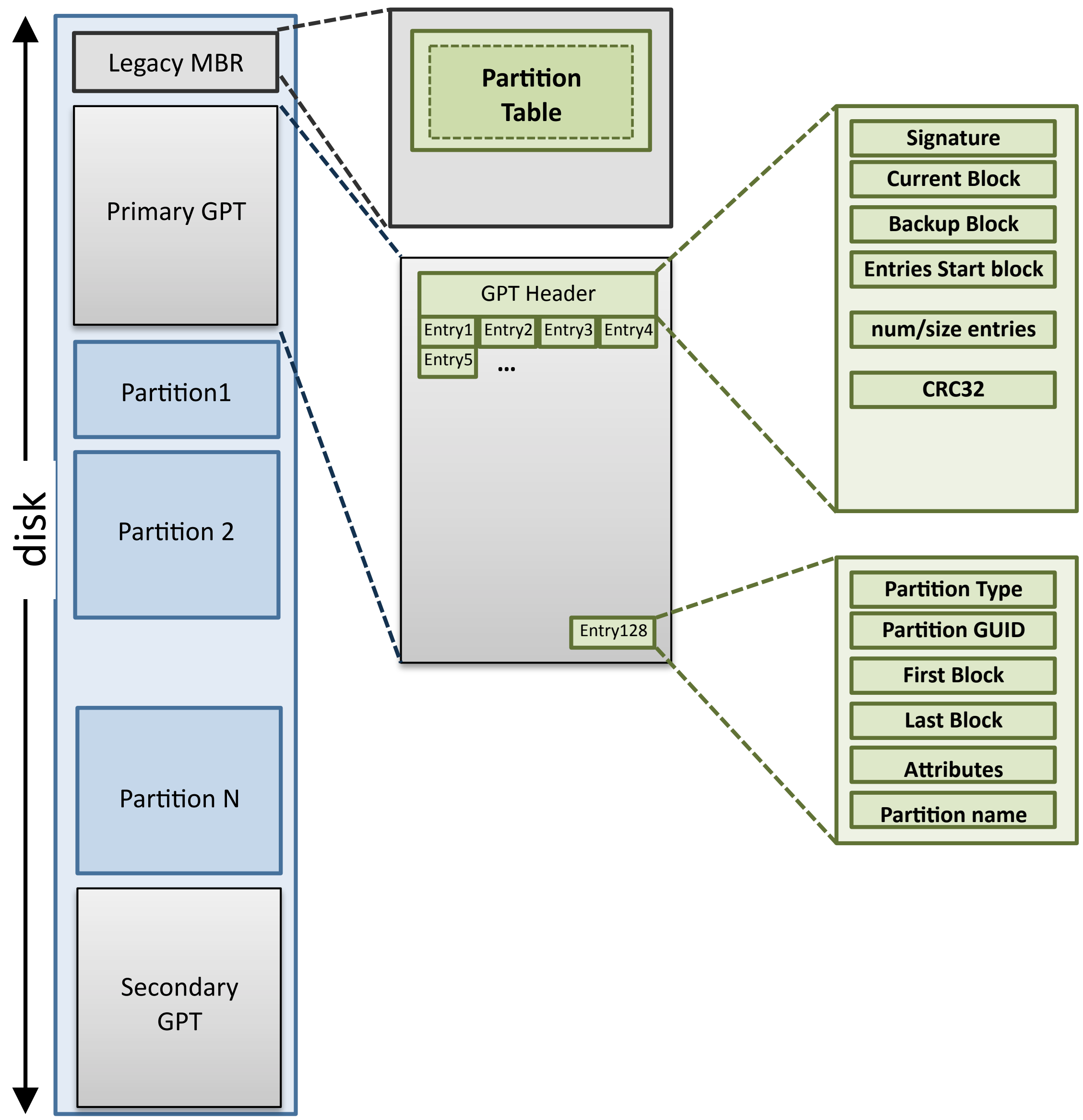


UEFI (+ grub)
KERNEL LOADING
INIT PROCESS
INIT LEVEL
LOGIN PROMPT
XDM PROCESS

UEFI

- **EFI/UEFI** (Unified Extensible Firmware Interface):
  - 2002: itanium platform from intel provides EFI firmware.
  - 2005: UEFI. Consortium of companies takes control over the firmware. Unified EFI Forum.
  - Works in 32/64 bits mode.
  - Much more flexible than BIOS:
    - Supports big disks (MBR: 32-bit block addresses. GPT: 64-bit block addresses):
      - MBR: 512KB block: 2TB disk.
    - Supports more booting devices (network).
    - Can eliminate the need for a bootloader (no stage 2).
    - Improved Security (network authentication, signed start up).
    - Extends bootloader operation (load the OS) to a UEFI-capable shell (interaction).
  - Requires support from the OS (Linux, OSX, Windows8).
  - Can emulate BIOS.
  - VirtualBox supports UEFI.

Previous: GPT Disks & Partitions



Previous: GPT Disks & Partitions

- **Protective/Legacy MBR:**
  - Backward compatibility, first block reserved.
  - Prevent MBR-based disk utilities from misrecognizing/overwriting GPT disks.
  - Single partition of special type (identifies a GPT disk). OS & tools which cannot read GPT recognize the disk and typically refuse to modify it.
- **Primary GPT Header:**
  - Defines the usable blocks on the disk.
  - Also defines the partition table (number & size of the partition entries). Minimum table: 128 entries, each 128 bytes long.
  - Also contains disk UUID, CRC32 checksum, its own size and location (always LBA 1) and the size and location of the secondary GPT header & table (always last disk sectors).

Previous: GPT Disks & Partitions

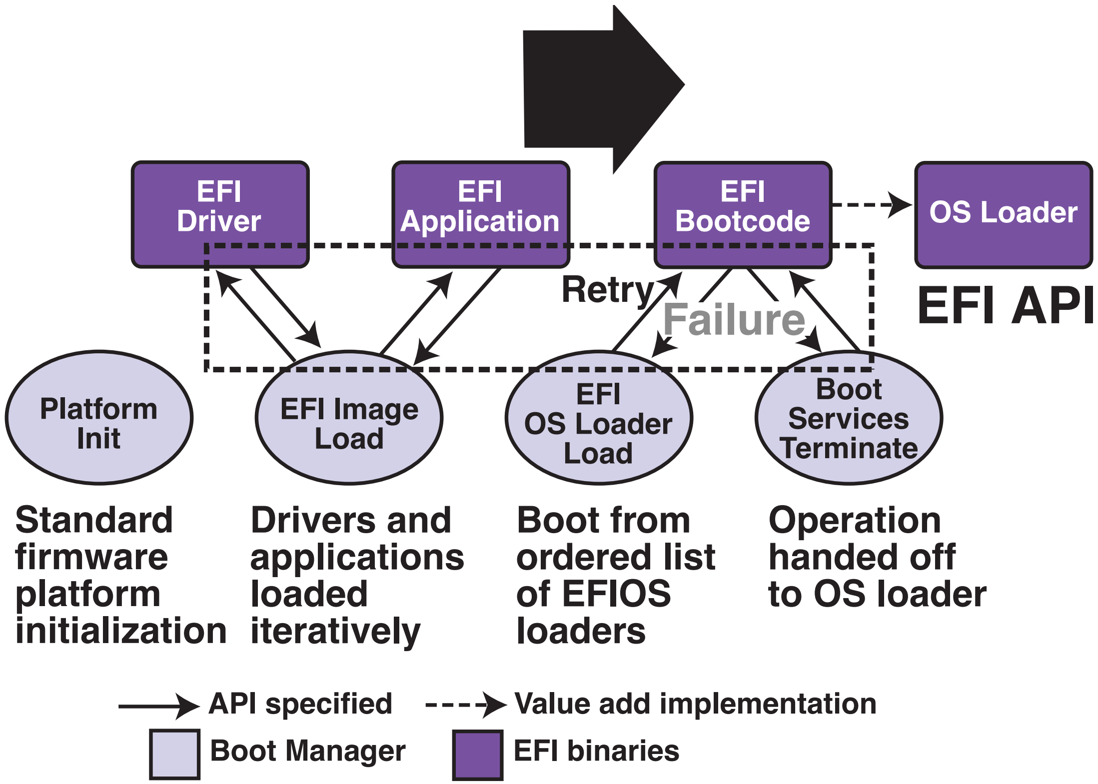
- **Partition Entries:**
  - 128 bytes for each entry block.
  - Each partition includes the following contents: Type, unique ID, First and last blocks, Attributes (e.g. read only) & partition name.
- **Secondary GPT Header:**
  - Copy of the Primary GPT header, placed in last disk blocks.
  - If checksum of primary header fails, secondary is employed.

UEFI

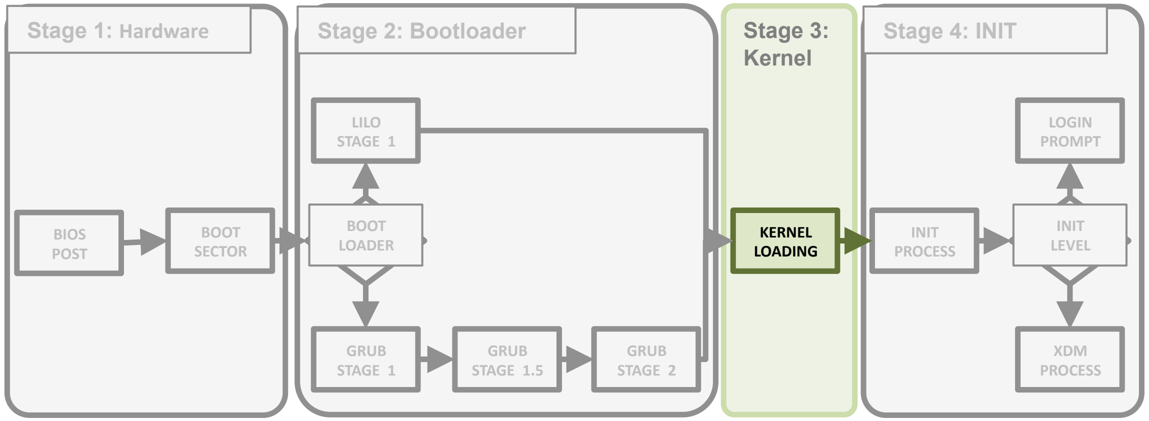
- Instead of a 512 MBR and some boot code, UEFI has its own filesystem, with files and drives (FAT32, 200-500Mb).
- UEFI marks one GPT partition with the boot flag:
  - But this is an EFI partition, never any of the OS partitions.
- Each installed OS has its own directory in EFI partition:
  - All necessary files for loading the OS are under these directories.
  - In Linux, after computer boot-up the EFI partition is sometimes mounted under the boot partition.
- Taking a look at the UEFI boot process, you realize it reminds you of a mini-OS.

UEFI

- Boot Manager:
  - Firmware policy engine that can be configured by modifying architecturally defined global NVRAM variables.
  - In charge of loading UEFI drivers and UEFI applications (including UEFI OS boot loaders). Boot order defined by the global NVRAM variables.



Index

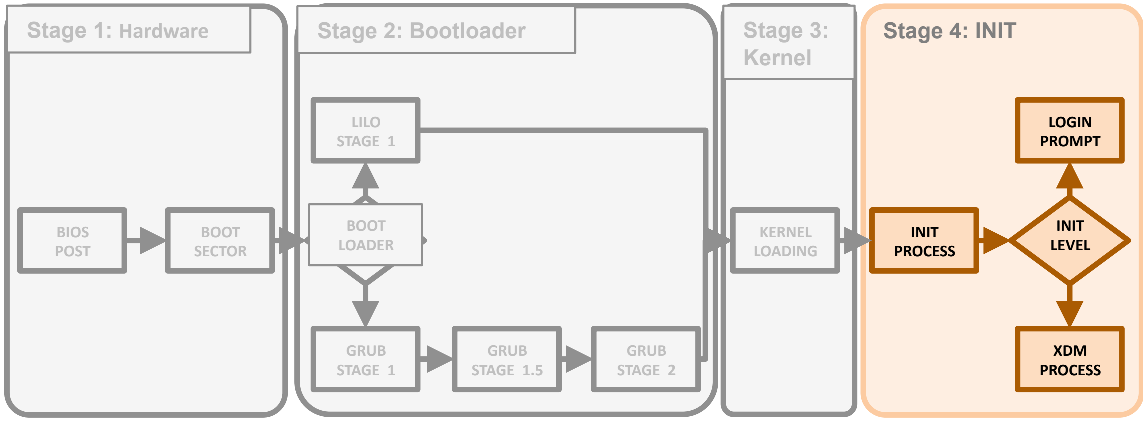


- Introduction.
- Booting, Stage 1: Hardware.
- Booting, Stage 2: Bootloader:
  - LILO.
  - GRUB.
- Booting, Stage 1+2 (UEFI).
- **Booting, Stage 3: Kernel.**
- Booting, Stage 4: SysV / Systemd.
- Shutting Down.

Stage 3: Loading the Kernel

- The bootloader has loaded kernel & ramdisk files into memory:
  - vmlinuz-2.6.26-2-686.
  - initrd.img-2.6.26-2-686.
- Once Stage 2 is complete, kernel execution starts:
  - The Kernel **uncompresses** itself.
  - Detects memory map, the **CPU** and its features supported.
  - Starts the **display** (console) to show information through the screen.
  - Checks the **PCI bus**, creating a table with the peripheral detected.
  - Initializes the system in charge of **virtual memory** management, including swapper.
  - Initializes the **drivers** for the peripherals detected (Monolithic or modular).
  - Mount **file system** root ("/").
  - Calls the **init** process (Stage 4): PID 1, father of the rest of processes.

Index



- Introduction.
- Booting, Stage 1: Hardware.
- Booting, Stage 2: Bootloader:
  - LILO.
  - GRUB.
- Booting, Stage 1+2 (UEFI).
- Booting, Stage 3: Kernel.
- **Booting, Stage 4: SysV.**
- Shutting Down.

Stage 4: INIT (SysV)

- The init process performs the following tasks:
  - Step 1. **Configuration:** read from the file **/etc/inittab** the initial configuration of the system: Operation mode, runlevels, consoles,…
  - Step 2. **Initialization:** runs the command **/etc/init.d/rc.S** (debian), which performs a basic initialization of the system.
  - Step 3. **Services:** according to the runlevel configured, runs the scripts/ services pre-established for that runlevel.
- Runlevels (Operation modes):
  - Standard: 7 levels. Each distribution has its own configuration (here Debian).
  - Level **S.** Only executed at boot time (replaces /etc/rc.boot).
  - Level 0. **Halt:** employed to Shut down the system.
  - Level 1. **Single User:** maintenance tasks (no active network).
  - Level 2-5. **Multiuser:** all the network and Graphical services activated.
  - Level 6. **Reboot:** similar to level 0.

Stage 4: INIT (SysV)

- **Step 1. Configuration. The file /etc/inittab:**

```
# /etc/inittab: init(8) configuration.

# The default runlevel.
id:2:initdefault:

# Boot-time system configuration/initialization
# script. This is run first except when booting in
# emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes S and K scripts upon change
# of runlevel.
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
```

```
# Normally not reached, but fallthrough in case of
# emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
…

# Note that on most Debian systems tty7 is used by
# the X Window System, so if you want to add more
# getty's go ahead but skip tty7 if you run X.
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

Stage 4: INIT (SysV)

- **Step 1. Configuration. The file /etc/inittab:**
  - Line format: **id:runlevels:action:process**.
  - **id:** identifier for the entry inside inittab.
  - **Runlevels:** execution levels for that entry (empty means all).
  - **Action:** what must init do with the process:
    - Wait: wait until it finishes.
    - Off: ignore the entry (deactivated).
    - Once: run only once.
    - Respawn: rerun the process if it dies.
    - Sysinit: ask the user what to do with that entry.
    - Special: ctrlaltdel.
  - **Process:** sh line tells init which process to start when this entry is reached.

Stage 4: INIT (SysV)

- **Step 2. Initialization. The file /etc/init.d/rc:**
  - Input parameters: the runlevel. Example rc 2: multiuser.
  - Tasks:
    - Establishes PATHs.
    - Loads swap space: swapon.
    - Checks and mounts local filesystems (/ets/fstab).
    - Activates and configures the network.
    - Removes unnecessary files (/tmp).
    - Configures the kernel. Loads modules: drivers (managing dependencies).
    - Triggers the startup of the services associated with the runlevel.
  - Modifying the runlevel: command init, telinit:
    - Allows changing from one runlevel to another.
    - Single User?
    - Restores original state.

Stage 4: INIT (SysV)

- **Step 3. Services. The directories /etc/init.d and /etc/rcN.d:**
  - All the services available are found in /etc/init.d:
    - Examples: cron, ssh, lpd…
  - How do we tell each runlevel which services to start?:
    - With a special directory, /etc/rcN.d/ (being N the runlevel).
    - In these directories a list of links to the services is found.
  - The directory /etc/rcN.d/:
    - The links begin with letters “S” or “K” plus two digits (execution order).
    - “S”: executed in ascending order when a runlevel is started (ssh start).
    - “K”: executed in descending order when shutting down (ssh stop).
    - These links are controlled with “update-rc.d”.
  - S99local: script to perform local configurations:
    - Minor booting aspects: auxiliary kernel modules, personalized services…
    - Employed by the administrator.
    - It really runs the script /etc/rc.local.

Stage 4: INIT (SysV)

- **Step 3. Services. The directories /etc/init.d and /etc/rcN.d:**
  - The directory /etc/rcN.d/.

```
pablo@si:/etc/rc2.d$ ls
README            S03cgroupfs-mount   S03vboxdrv           S05cups
S01bootlogs       S03cron             S04avahi-daemon      S05cups-browsed
S01rsyslog        S03dbus             S04docker            S05saned
S02apache2        S03exim4            S04lightdm           S06plymouth
…
```

```
pablo@si:/etc/rc6.d$ ls
K01alsa-utils     K01network-manager     K02avahi-daemon     K06rpcbind
K01apache2        K01plymouth            K02vboxdrv          K07hwclock.sh
…
```

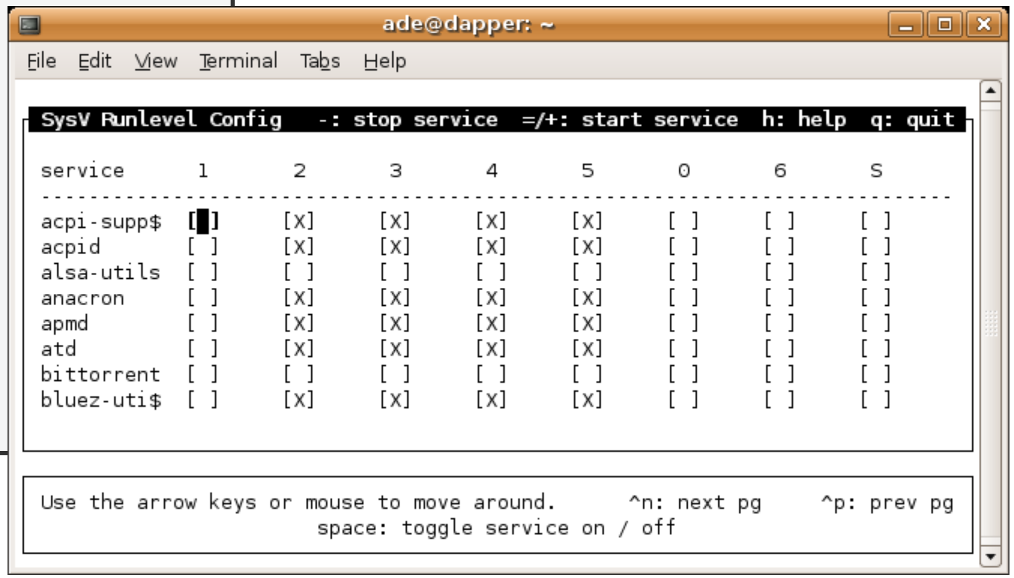
Stage 4: INIT (SysV)

- **Manual** administration of services:
  - After booting process, services can be modified (stop running services or start new services).
  - Directly through its script (example ssh):
    - # /etc/init.d/ssh [stop/start/restart/status].
  - Or through the command service:
    - Service --status-all: reads /etc/init.d/ verifying service state [+] [-] [?].
  - These changes are volatile (lost after reboot):
    - Permanent with update.rc-d.
  - Checking possible errors concerning boot process:
    - # tail -f /var/log/messages (Other important files: syslog, daemon.log).
    - # ls -lart /var/log.

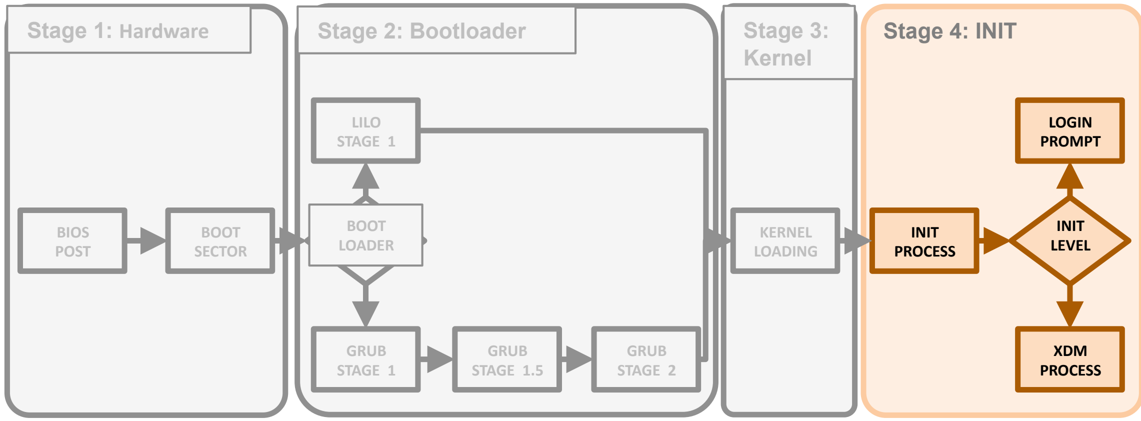
Stage 4: INIT (SysV)

- **Manual** administration of services:
  – Examples of start script and services command:

```
#!/bin/sh
#SIMPLIFICADO

[ -f /usr/local/sbin/sshd2 ] || exit 0

PORT=

PORT=`grep Port /etc/ssh2/sshd2_config | awk '{ x = $2 } END {print x}' -`
if [ "X$PORT" = "X" ]
then
        PORT=22
fi

# See how we were called.
case "$1" in
  start) # Start daemons.
        echo -n "Starting sshd2 in port $PORT: "
        /usr/local/sbin/sshd2
        echo "done."
        ;;
  stop) # Stop daemons.
        echo -n "Shutting down sshd2 in port $PORT: "
        kill `cat /var/run/sshd2_$PORT.pid`
        echo "done."
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  *)
        echo "Usage: sshd2 {start|stop|restart}"
        exit 1
esac

exit 0
```



Index



- Introduction.
- Booting, Stage 1: Hardware.
- Booting, Stage 2: Bootloader:
  - LILO.
  - GRUB.
- Booting, Stage 1+2 (UEFI).
- Booting, Stage 3: Kernel.
- **Booting, Stage 4: Systemd.**
- Shutting Down.

Stage 4: Systemd

- SysV is not the only available init system:
  - BSD init, ubuntu's Upstart, **systemd**.
- What are systemd benefits?:
  - Faster Startup:
    - Sysvinit is slow: it starts processes one at a time, performs dependency checks on each one, and waits for daemons to start so more daemons can.
    - Daemons don't need to know if the daemons they depend on are actually running (only need the inter-process communication sockets to be available).
    - Step 1. Create all sockets for all daemons. Step 2: start all daemons.
    - Client requests for daemons not yet running buffered in the socket, filled when the daemons are up and running.
  - Hotplugging and On-Demand Services:
    - After startup sysvinit goes to sleep and doesn't do any more.
    - Systemd (making use of D-Bus) can expand init duties, working as a full-time Linux process babysitter.

Stage 4: Systemd

- Systemd **Unit:** any resource that system can operate/manage:
  - This is the primary object that the systemd tools know how to deal with.
- Available Systemd unit types:
  - **.service:** a system service.
  - **.target:** a group of systemd units.
  - **.automount:** a file system automount point.
  - **.device:** a device file recognized by the kernel.
  - **.mount:** a file system mount point.
  - **.path:** a file or directory in a file system.
  - **.socket:** an inter-process communication socket.
  - **.swap:** a swap device or a swap file.
  - **.timer:** a systemd timer.
  - …

Stage 4: Systemd

- Location of the Unit files:
  - /usr/lib/systemd/system/, /run/systemd/system/, /etc/systemd/system/.
- General Characteristics of Unit files:
  - Internal structure organized with sections, denoted as: [section_name].
  - At each section, behavior is defined through key-value directives (one per line).

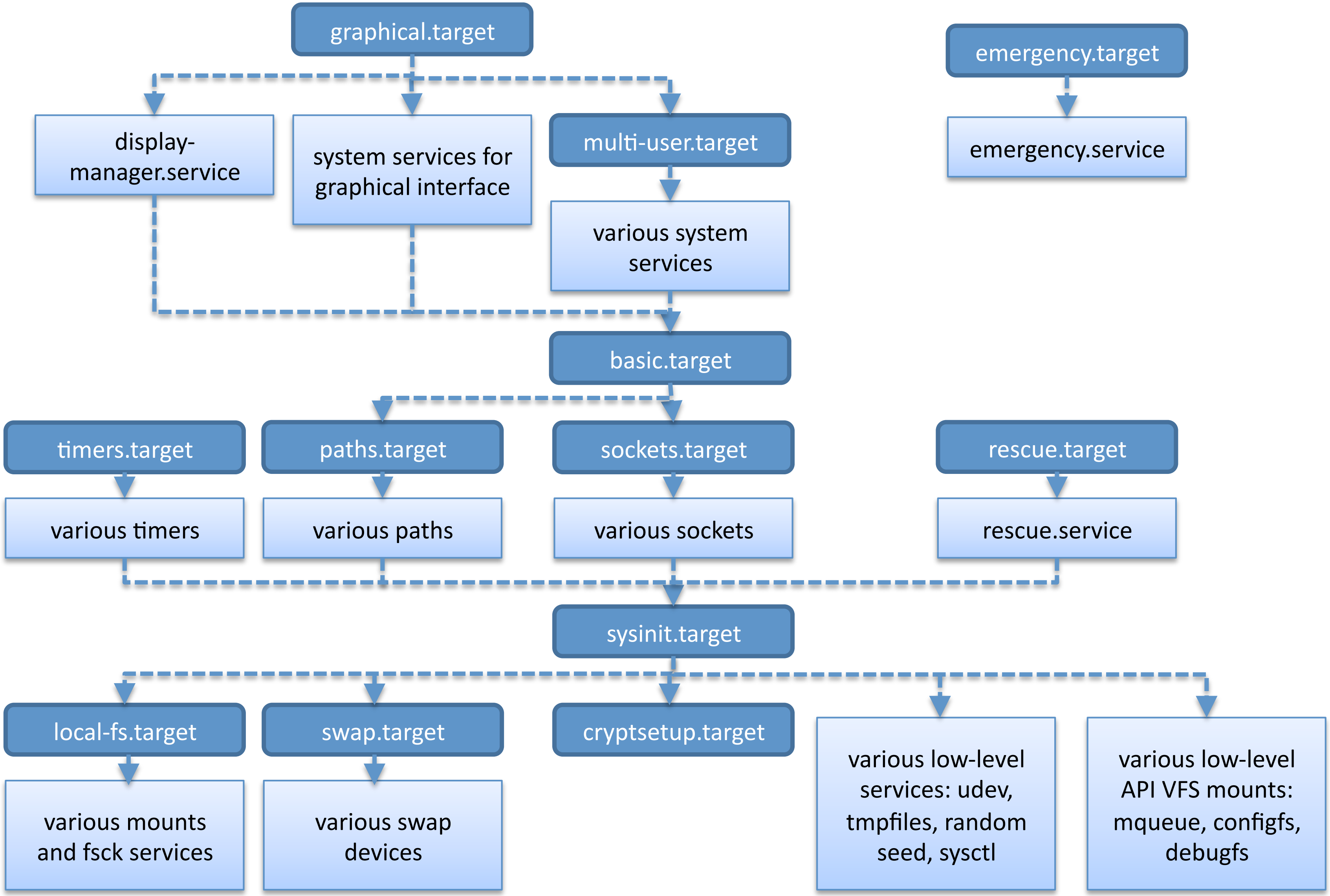
```
[Unit]
Description=Simple firewall

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/local/sbin/simple-firewall-start
ExecStop=/usr/local/sbin/simple-firewall-stop

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

Stage 4: Systemd

- Systemd **boot process:**
  - Configure grub2 for systemd:
    - GRUB_CMDLINE_LINUX="init=/lib/systemd/systemd" (run update-grub afterwards).
  - Systemd handles boot & service management using **Targets:**
    - Target: special unit employed to group boot units and start up synchronization processes.
  - First target executed: **default.target:**
    - Usually a symbolic link to graphical.target.
  - Target Unit File main options:
    - Requires: hard dependencies. Must start before your own service.
    - Wants: soft dependencies (not required to start). Can be replaced by a directory, named foo.target.wants.
    - After: boots after these services.
  - Runlevels: Specific Target units.

```
[Unit]

Description=foo boot target
Requires=multi-user.target
Wants=foobar.service
After=multi-user.target rescue.service rescue.target
```

Stage 4: Systemd



Stage 4: Systemd

- Service administration through the **systemctl** command:
  - Table: comparison of the service utility with systemctl.

service (sysV)	systemctl (systemd)	Description
service *name* start	systemctl start *name*.service	Starts a service
service *name* stop	systemctl stop *name*.service	Stops a service
service *name* restart	systemctl restart *name*.service	Restarts a service
service *name* status	systemctl status *name* .service	Checks if a service is running
service –status-all	systemctl list-units –type service	Displays the status of all services

- System & Boot performance statistics through **systemd-analyze** command:
  - Alternatives for SysV: Bootchart.

Index

- Introduction.
- Booting, Stage 1: Hardware.
- Booting, Stage 2: Bootloader:
  - LILO.
  - GRUB.
- Booting, Stage 1+2 (UEFI).
- Booting, Stage 3: Kernel.
- Booting, Stage 4: INIT.
- **Shutting Down.**

Shutting Down

- Never shut down directly (reset!):
  - If this rule is not followed, there is a high probability of losing or corrupting system files (with a bit of bad luck, fully broken system).
  - Intermediate Buffers for disk read/write. Synchronization.
- Never shut down without warning all system users:
  - Periodically programmed shut-downs.
- Steps for a correct shut down:
  - Warn all users beforehand.
  - Stop all services associated with (/etc/rcN.d/Kxxservice stop).
  - Send the specific signal to all the processes to end their execution.
  - Users and processes still present, killed.
  - Subsystems shut down sequentially.
  - File System unmounted (synchronizes pending changes with disk).

Shutting Down

- Command **shutdown:**
  - Format: /sbin/shutdown -<options> time message:
    - Option **–r:** reboot instead power off.
    - Option **–h:** stop the system(with ACPI).
  - Message: message sent to all users.
  - Time: delay to begin the shutdown (mandatory):
    - Format: hh:mm.
    - Supports now+,minutes.
- /etc/shutdown.allow or inittab:
  - Avoid Ctrl+Alt+Del.
- Other commands: /sbin/halt, /sbin/poweroff.