IEN-003 - Stage 2 — The Bootloader (GRUB)

At the end of the firmware stage, the computer has completed its hardware initialization and identified a bootable storage device.

However, the firmware still faces one fundamental problem:

> It knows **where** the operating system is stored, but it does **not know how to load it**.

This is where the bootloader comes into play.

The bootloader acts as the bridge between the firmware and the operating system. Its primary responsibility is to locate the Linux kernel, load it into memory, provide the necessary boot parameters, load the initial RAM disk (`initrd` or `initramfs`), and finally transfer execution to the kernel.

Without a bootloader, the firmware has no knowledge of Linux-specific file formats, filesystem layouts, or kernel images.

The bootloader is therefore the **first piece of software that understands how to start Linux**.

Why Do We Need a Bootloader?

A common misconception is that the firmware could simply load the Linux kernel directly.

While this is technically possible in certain specialized systems, it is not how general-purpose computers are designed.

The firmware is intentionally kept generic. It can initialize hardware and locate a bootable device, but it does not understand:

- Linux kernels
- Windows boot managers
- FreeBSD kernels
- Multiple operating systems
- Kernel command-line arguments
- Recovery modes

Instead, the firmware loads a small program whose sole purpose is boot management.

That program is the bootloader.

Its responsibilities include:

- Presenting a boot menu.
- Selecting an operating system.
- Loading the kernel into memory.
- Loading the initial RAM disk.
- Passing boot parameters to the kernel.
- Supporting recovery and rescue modes.

The Boot Sequence So Far

At this point, the startup sequence looks like this:

```
Power On
   |
   ▼
Firmware (BIOS / UEFI)
   |
   ▼
POST
   |
   ▼
Boot Device Selected
   |
   ▼
Bootloader (GRUB)
```

Notice that Linux itself is **still not running**.

GRUB is responsible for preparing everything the kernel needs before handing over control.

A Brief History of Linux Bootloaders

Before GRUB became the de facto standard, Linux systems used several different bootloaders.

Some of the most well-known include:

Bootloader	Status	Notes
LILO	Legacy	One of the earliest Linux bootloaders. Required reinstallation after kernel changes.
GRUB Legacy	Legacy	Widely used by RHEL 5, CentOS 5, and many older distributions.
GRUB2	Current	Successor to GRUB Legacy. More modular and feature-rich.
systemd-boot	Current	Lightweight bootloader primarily used on UEFI systems.

Because much of the *Infrastructure Engineering Notes* series focuses on enterprise environments—including legacy systems such as RHEL 5—we will encounter both **GRUB Legacy** and **GRUB2**.

Understanding their differences is essential when performing recovery operations.

GRUB Legacy vs GRUB2

Although both bootloaders share the same goal, their internal design differs considerably.

Feature	GRUB Legacy	GRUB2
Typical distributions	RHEL 5, CentOS 5	RHEL 7+, Ubuntu, Debian, Rocky Linux
Configuration file	`/boot/grub/grub.conf`	`/boot/grub2/grub.cfg` or `/boot/efi/EFI/...`
Configuration generation	Manual editing	Automatically generated
Module support	Limited	Modular architecture
Filesystem support	Basic	Extensive
UEFI support	No	Yes

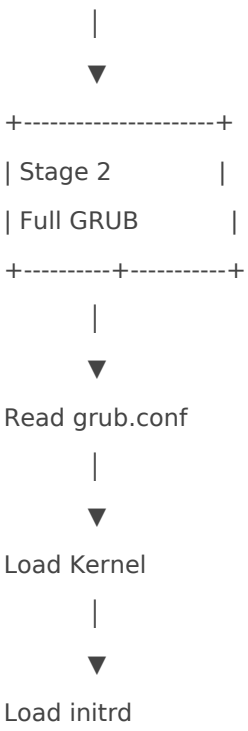
One important point is that **GRUB2 is not simply a newer version of GRUB Legacy**. It is a complete redesign.

For example, on RHEL 5 you typically edit `grub.conf` directly. On modern systems, editing `grub.cfg` manually is discouraged because it is generated automatically from configuration templates.

This distinction becomes important during recovery scenarios.

Inside GRUB Legacy

Under legacy BIOS systems, GRUB is traditionally divided into multiple stages.



Let's examine each stage.

Stage 1

Stage 1 occupies the first 446 bytes of the Master Boot Record (MBR).

Its size is extremely limited, so it cannot understand filesystems or locate the Linux kernel directly.

Its only responsibility is to load the next stage.

Stage 1.5

Stage 1.5 resides immediately after the MBR in the so-called *embedding area*.

Unlike Stage 1, it contains basic filesystem drivers.

This allows GRUB to read files from partitions such as ext2 or ext3.

Without Stage 1.5, Stage 1 would have no way to locate the full GRUB program stored within the filesystem.

Stage 2

Stage 2 is the full bootloader.

It provides:

- Interactive command line.
- Boot menu.
- Configuration parsing.
- Filesystem access.
- Kernel loading.
- initrd loading.
- Recovery options.

When users refer to "GRUB," they are usually interacting with Stage 2.

The GRUB Configuration File

Once Stage 2 starts, GRUB searches for its configuration file.

On GRUB Legacy systems, this file is usually:

```
/boot/grub/grub.conf
```

or

```
/boot/grub/menu.lst
```

The configuration file defines:

- Available operating systems.
- Default boot entry.
- Boot timeout.
- Kernel image location.
- initrd image location.

- Kernel parameters.

A simplified example looks like this:

```
title Red Hat Enterprise Linux

root (hd0,0)

kernel /vmlinuz-2.6.18-128.el5 ro root=/dev/VolGroup00/LogVol00

initrd /initrd-2.6.18-128.el5.img
```

Each line has a specific purpose:

- `root (hd0,0)` tells GRUB where to find the `/boot` partition.
- `kernel` specifies the kernel image and its boot parameters.
- `initrd` points to the initial RAM disk required during early boot.

If any of these paths are incorrect, GRUB may fail before the Linux kernel even begins execution.

Engineering Insight

One of the most common production issues after restoring or cloning a server is that the kernel and `initrd` files no longer match the entries in `grub.conf`. GRUB itself is functioning correctly, but it is instructed to load files that no longer exist or belong to a different kernel version. The result can range from a simple "File not found" message to a kernel panic later in the boot process.

This is exactly the type of issue we encountered during our RHEL 5.2 recovery project, and we will revisit it in detail later in this series.

Loading the Linux Kernel

Once a boot entry is selected, GRUB performs one of its most important tasks: loading the Linux kernel into memory.

At this stage, the kernel is still just a compressed binary file stored on disk.

GRUB reads this file from the `/boot` filesystem, copies it into RAM, and prepares the execution environment.

Immediately afterward, it loads the corresponding `initrd` (or `initramfs`) image into memory as well.

Only when both components are ready does GRUB transfer control to the Linux kernel.

From that moment onward, the bootloader's job is complete.

The Linux kernel takes over, and the boot process enters its next stage.

Common Bootloader Failures

Understanding common failures helps quickly identify whether a problem belongs to the bootloader stage.

Symptom	Likely Cause
`grub>` prompt	Missing or incorrect configuration
`grub rescue>`	GRUB cannot locate its modules or filesystem
`Error 15: File not found`	Kernel or initrd path is incorrect
`Unknown filesystem`	Filesystem driver unavailable or partition damaged
Boot menu does not appear	Corrupted GRUB installation or wrong boot target

A key observation is that **none of these errors originate from the Linux kernel**. They occur **before** the kernel starts executing.

Summary

The bootloader is responsible for bridging the gap between firmware and the Linux kernel. It locates the kernel image, loads the initial RAM disk, passes kernel parameters, and transfers control to the operating system.

A failure at this stage prevents Linux from starting entirely, making it essential to distinguish bootloader issues from kernel or userspace problems.

Revision #2
Created 4 July 2026 04:13:35 by Kapak Maut
Updated 4 July 2026 04:14:20 by Kapak Maut