IEN-005 - Stage 4 — initrd / initramfs: Preparing the System to Mount the Root Filesystem

By the end of the previous stage, the Linux kernel is fully operational.

It has:

- initialized the processor,
- taken control of memory management,
- started its internal subsystems,
- detected hardware,
- initialized built-in device drivers.

Despite all of this progress, the system still cannot boot completely.

One critical problem remains.

> **The kernel still does not know how to access the real root filesystem.**

This may sound surprising.

After all, the kernel was loaded from the disk.

Shouldn't it already know where the operating system is?

The answer is:

Not necessarily.

Modern Linux systems often place the root filesystem behind one or more abstraction layers.

For example:

```
Disk
 |
 ▼
RAID
 |
 ▼
LVM
 |
 ▼
Filesystem
 |
 ▼
Root (/)
```

or

```
Disk
 |
 ▼
Multipath
 |
 ▼
LVM
 |
 ▼
Filesystem
 |
 ▼
Root (/)
```

or even

```
SAN Storage
 |
 ▼
```

```
Fibre Channel
  |
  ▼
Multipath
  |
  ▼
LVM
  |
  ▼
XFS
  |
  ▼
Root (/)
```

At this moment, the kernel has not yet assembled these storage layers.

It cannot simply mount `/`.

Something else must prepare the environment first.

That "something" is the **Initial RAM Disk**, commonly known as **initrd** or **initramfs**.

Why Does initrd Exist?

To understand why initrd exists, let's look at the history of Linux.

In the early days, Linux systems were much simpler.

A typical machine looked like this:

```
IDE Disk
   |
   ▼
ext2 Filesystem
   |
   ▼
```

Root Filesystem

The kernel already contained all required drivers.

Booting was straightforward.

Kernel

↓

Mount Root

↓

Start init

No temporary filesystem was necessary.

As Linux evolved, systems became significantly more complex.

Modern enterprise servers may use:

- LVM
- Software RAID
- Hardware RAID
- Multipath
- Fibre Channel SAN
- iSCSI
- NVMe
- Encrypted disks (LUKS)
- Network boot

Bundling every possible driver into the kernel would make it unnecessarily large and difficult to maintain.

Instead, Linux adopted a modular design.

Only essential components remain built into the kernel.

Additional modules are loaded during early boot from a temporary filesystem.

That temporary filesystem is initrd (or, on modern systems, initramfs).

---

initrd vs initramfs

Although the terms are often used interchangeably, they are not identical.

initrd	initramfs
Older implementation	Modern implementation
Block device backed	RAM filesystem
Used heavily by RHEL 5 and older distributions	Used by modern Linux distributions
Typically created with `mkinitrd`	Typically created with `dracut`
Less flexible	More flexible and efficient

Because this series includes both legacy and modern Linux systems, we will use the term **initrd** when discussing older environments such as RHEL 5, and **initramfs** when referring to newer systems.

What Is Inside initrd?

Many administrators think of initrd as a mysterious binary blob.

In reality, it is simply a compressed filesystem image containing everything needed to prepare the system for mounting the real root filesystem.

Typical contents include:

```
/init
/bin
/sbin
/lib
/lib/modules
/dev
/proc
/sys
```

More importantly, it contains:

- Storage drivers
- Filesystem drivers
- LVM tools
- RAID utilities
- Multipath utilities
- Device mapper support
- udev components

Think of initrd as a **miniature Linux system** whose only purpose is to prepare the real Linux system.

Boot Flow with initrd

Once the kernel has completed its initialization, it unpacks the initrd into memory.

The process now looks like this:

```
GRUB
 |
 ▼
Kernel
 |
 ▼
Unpack initrd
 |
 ▼
Temporary Root Filesystem
 |
 ▼
Execute /init
```

Notice something important.

The kernel has **not** mounted the real root filesystem yet.

Instead, it mounts the temporary root filesystem provided by initrd.

From this point onward, the `/init` program inside initrd becomes responsible for preparing the real storage environment.

What Happens Inside initrd?

This is where most enterprise boot logic takes place.

The `/init` program performs tasks such as:

1. Loading additional kernel modules.
2. Starting udev.
3. Detecting storage devices.
4. Loading storage controller drivers.
5. Detecting RAID arrays.
6. Activating LVM volume groups.
7. Starting multipath services.
8. Locating the root filesystem.
9. Mounting the real root filesystem.
10. Switching execution to the real system.

A simplified workflow looks like this:

```
Kernel
   |
   ▼
Execute /init
   |
   ▼
Load Modules
   |
   ▼
Detect Storage
   |
   ▼
Activate LVM
   |
   ▼
Locate Root Filesystem
   |
   ▼
```

```
Mount Root
    |
    ▼
switch_root
```

LVM Activation

Let's focus on one of the most important steps.

Many enterprise Linux installations store the root filesystem inside LVM.

For example:

```
Disk
 |
 ▼
Physical Volume (PV)
 |
 ▼
Volume Group (VG)
 |
 ▼
Logical Volume (LV)
 |
 ▼
XFS / ext4
 |
 ▼
Root Filesystem
```

The kernel cannot mount:

```
/dev/mapper/vg_root-lv_root
```

unless the Volume Group has already been activated.

That activation happens inside initrd.

Internally, tools equivalent to:

```
vgscan
vgchange -ay
```

(or their initramfs equivalents) make the logical volumes available before the root filesystem is mounted.

Without this step, the kernel has no device to mount as `/`.

Engineering Insight

This explains why LVM-related boot failures almost always appear **before** the system reaches `systemd` or the login prompt. If the Volume Group cannot be activated during the initrd stage—because a Physical Volume is missing, metadata is corrupted, or the necessary LVM tools are absent—the root filesystem never becomes available. The kernel is left waiting for a device that does not exist, and the boot process cannot continue.

This is exactly the class of problem we encountered during the RHEL 5.2 recovery after the Acronis restore. The issue was not with the kernel itself, but with the early userspace environment failing to expose the expected storage layout.

Device Discovery

While LVM is being activated, initrd also waits for storage devices to appear.

This includes:

- SATA disks
- SAS disks
- NVMe drives
- Fibre Channel LUNs
- iSCSI targets
- Multipath devices

Enterprise storage may require several seconds before all devices become available.

The initrd environment is responsible for waiting, probing, and assembling these devices before attempting to mount the root filesystem.

---

switch_root

Once the real root filesystem has been successfully mounted, the temporary initrd environment is no longer needed.

Instead of rebooting or restarting, Linux performs a handoff.

This operation is called **`switch_root`** (historically, some systems used `pivot_root`).

The transition looks like this:

```
Temporary Root (initrd)
        |
        ▼
Real Root Filesystem
        |
        ▼
switch_root
        |
        ▼
Execute /sbin/init
```

After this point, the initrd environment is discarded, and the real operating system takes over.

This is the first moment during the boot process where the system begins executing programs from the permanent root filesystem.

Common initrd / initramfs Failures

Because initrd is responsible for preparing the storage stack, failures here are often related to storage, drivers, or root filesystem discovery.

Symptom	Likely Cause
`dracut-initqueue timeout`	Storage device never appeared
`Volume Group not found`	LVM activation failed
`No root device found`	Incorrect `root=` parameter or missing driver
`Kernel panic - unable to mount root fs`	Root filesystem could not be mounted
`Waiting for root device`	Storage initialization incomplete

One important observation is that these errors occur **after the kernel has started**, but **before the real operating system begins**. Understanding that distinction helps avoid troubleshooting the wrong component.

Summary

The initrd/initramfs stage bridges the gap between a running kernel and a usable operating system. It provides a temporary userspace where additional drivers can be loaded, storage technologies such as LVM and RAID can be activated, and the real root filesystem can be located and mounted.

Without this stage, many modern Linux systems would be unable to boot because the kernel alone lacks the information required to discover complex storage configurations.

Revision #2
Created 4 July 2026 04:17:03 by Kapak Maut
Updated 4 July 2026 04:18:42 by Kapak Maut