

# Session 11

Omnia Ibrahim

# Table of contents

01

## Boot Process

**sequence of steps**  
your computer goes  
through when you  
turn it on

02

## Processes

instance of a running  
program

03

## Systemd

**init system** (the first  
real program started  
by the kernel).



**HI!**

---



01

# Boot Process

# WHAT?

## what? , why? And when?

- **boot** (short for "bootstrap") refers to the process of starting up a computer or device from a powered-off state to a fully functional state.
- The boot process involves loading the operating system (OS) and all the necessary system software into memory to enable the device to perform tasks.



# what? , why? And when?

## WHY?

Understanding the Linux boot and startup processes is important to being able to both configure linux and to resolving startup issues.

# what? , why? And when?

## WHEN?

**The boot process can be initiated in two ways:**

- If power is turned off, turning on the power will begin the boot process.
- If the computer is already running, the boot sequence can be initiated through a reboot, which will first do a shutdown and then restart the computer.

# Boot Process Stages

Hardware Initialization

## 1. BIOS Stage

After the selected kernel is loaded into memory and begins executing

## 3. Kernel Stage

## 2. Bootloader stage

program responsible for loading the Linux kernel and the initial RAM disk.

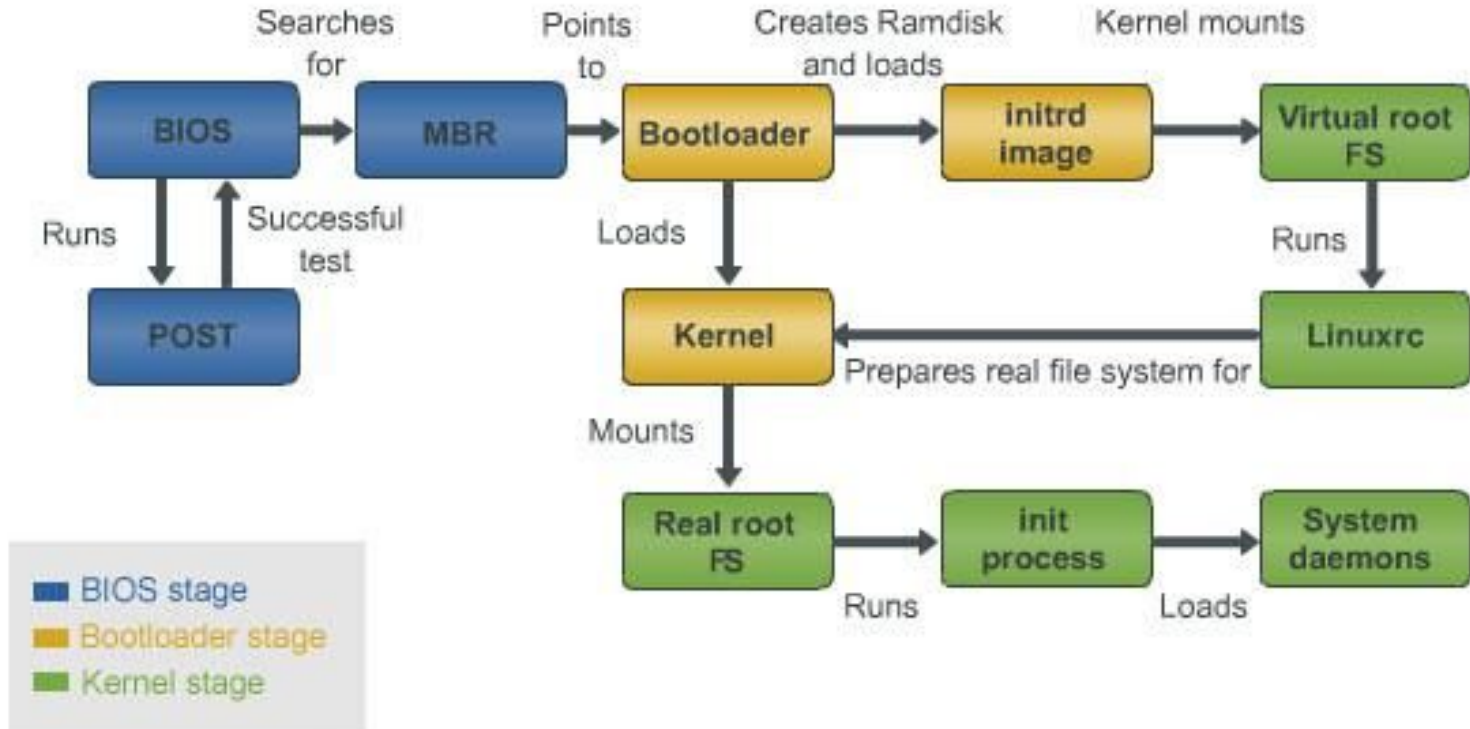
## 4. Init/Systemd

The parent of all Processes





# Boot Process

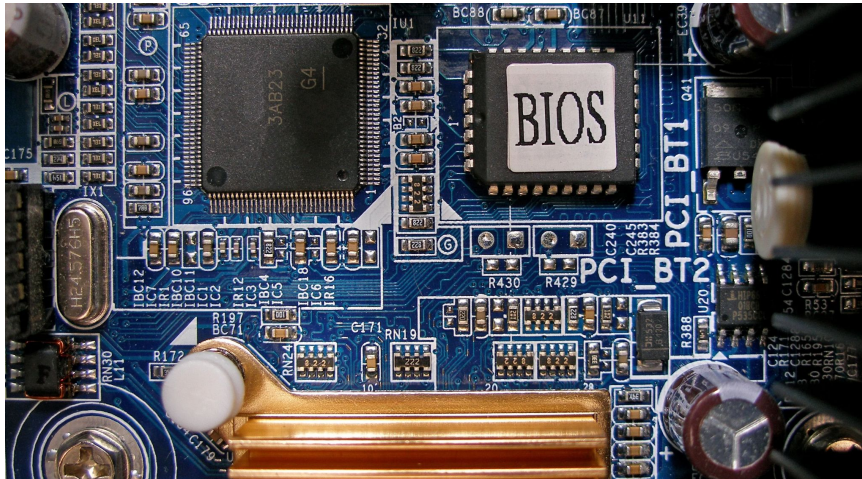


# BIOS Stage

- BIOS (Basic Input/Output System) is the **firmware** that initializes and tests the hardware components of a computer when it is powered on. It also provides a runtime environment for the operating system and other programs to interact with the hardware.
  - **Firmware** is a special kind of software that is embedded into hardware devices. It controls low-level operations and is usually the first code that runs when a device powers on.
- ★ Basic → it's minimal, just enough to get things started.
  - ★ Input/Output → it handles basic communication between your hardware and software.
  - ★ System → it's a core part of the system's startup process.

# BIOS Stage

- BIOS is stored in ROM or Flash memory on a chip on the motherboard.



# BIOS steps

performing the POST

01

02

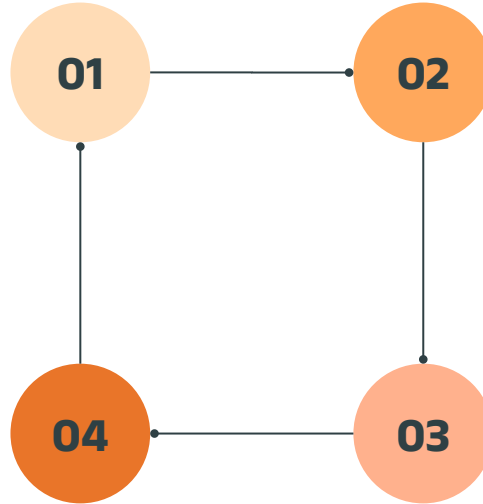
Initializing hardware

load the MBR ,  
run the bootloader

04

03

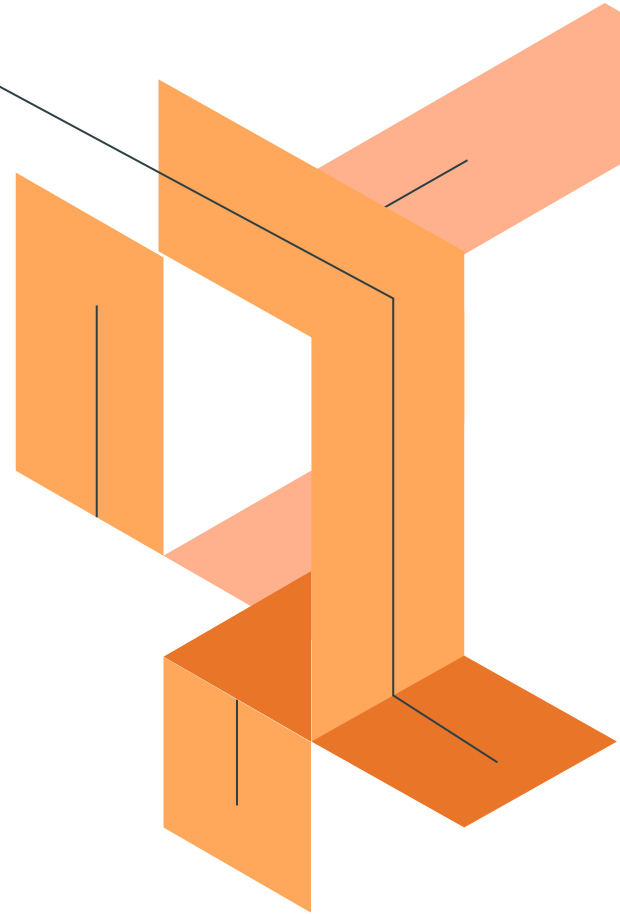
Search for Bootable  
Device



# BIOS Stage

## *1\_POST (Power On Self Test):*

- POST is software running in the BIOS chip whose task is to ensure that the computer hardware functioned correctly.
- POST is a diagnostic testing sequence that checks the processor, RAM, storage devices, and other critical components to ensure they are functioning correctly.



# BIOS Stage

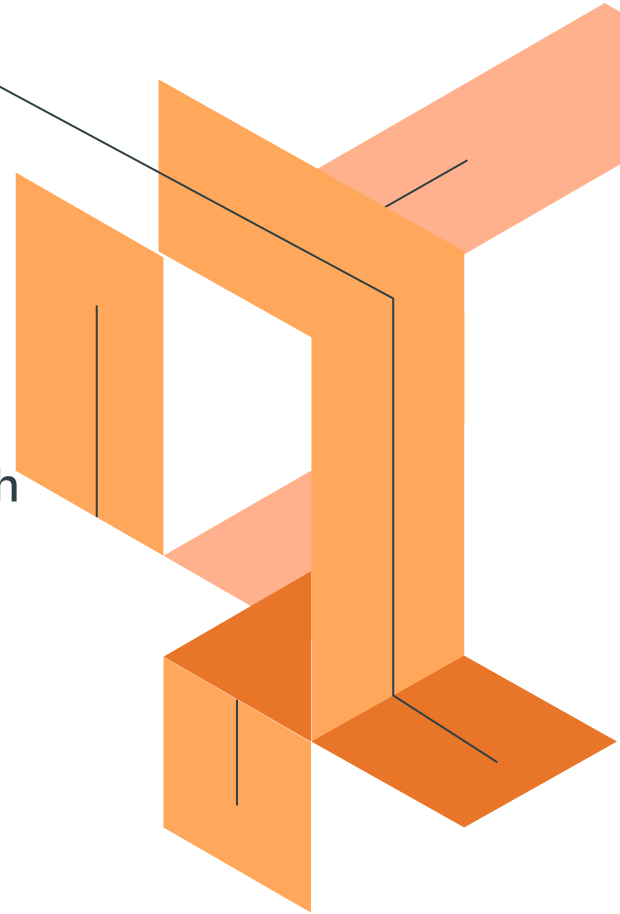
## ***1\_POST (Power On Self Test):***

The CPU starts executing POST code from BIOS.

The POST code includes commands to interact with each device.

These commands cause the CPU or BIOS chip to:

- Try reading from or writing to a device
- Wait for expected responses or signals



# There is two scenarios

## Issues

- ❖ If the BIOS finds any errors during the POST, it will notify you by a series of beeps or a text message displayed on the screen.
- ❖ An error at this point is almost always a hardware problem.
- ❖ If POST fails, the computer may not be usable and so the boot process does not continue.

## No issues

- ❖ Once POST is completed, BIOS initialize the hardware then looks for the bootable device and checks its master boot record (MBR) to locate the boot loader and load it into memory.
- ❖ Control is then passed from BIOS to the boot loader.

# BIOS Stage

## *1\_POST (Power On Self Test):*

### *Common problems:*

- No display + beeps → RAM or GPU error
- Message like "Keyboard not detected"? → POST failed keyboard check
- "No boot device found" → POST couldn't find a working storage device





8 beep error  
&  
blank display

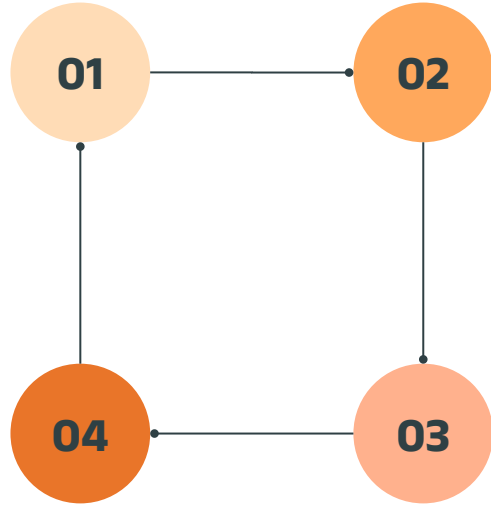
The image shows a laptop screen with red text. The text is arranged in four lines: '8 beep error', '&', 'blank display'. The background of the image is a blurred workshop or lab setting with various electronic equipment and a wooden desk.

# BIOS steps

performing the POST



load the MBR ,  
run the bootloader



Initializing hardware

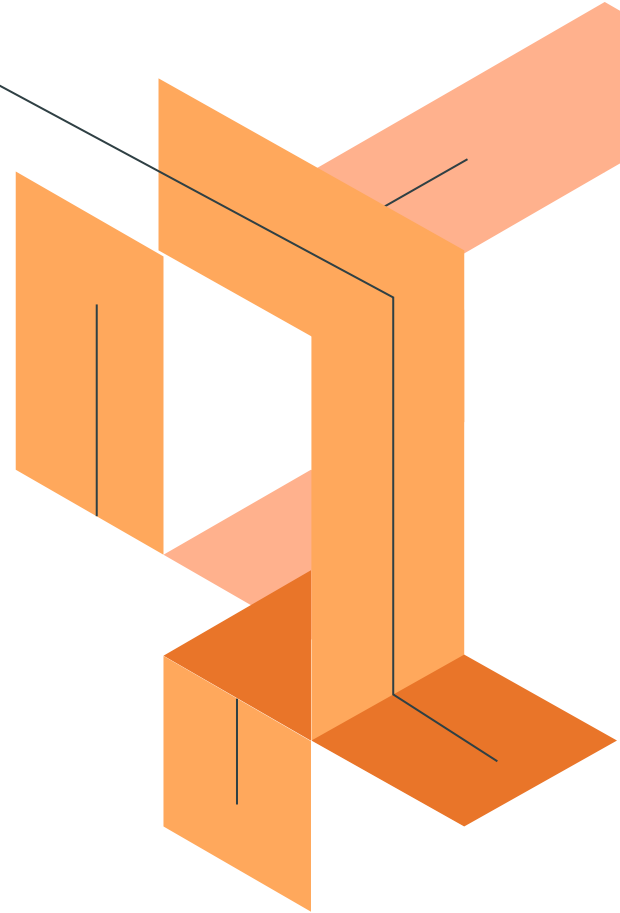
Search for Bootable  
Device

# BIOS Stage

## 2\_Hardware initialization:

- The first step of the BIOS process is Checks if hardware is functioning then, **hardware initialization**.
- **Hardware initialization** refers to the process of detecting and preparing the physical components of your computer (like the CPU, RAM, hard drives, keyboard, GPU, etc.) for use when the system is powered on or rebooted.

Hardware initialization = "Get everything ready so the operating system can control the hardware."



# BIOS Stage

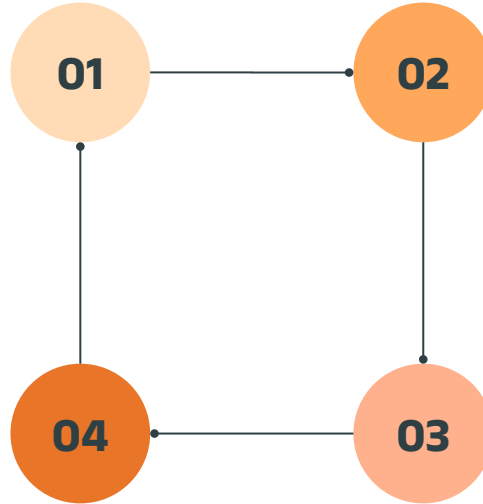
## ***2\_Hardware initialization:***

- the BIOS runs more setup code (still stored in the BIOS chip).
- The BIOS communicates with each device using low-level instructions through buses.  
like:
  - Configure device settings (e.g., screen resolution, memory timings)
  - Send initialization commands (e.g., “keyboard, prepare to receive input”)
  - Enable features (like turning on the display or USB legacy support)
  - Set communication channels (like enabling interrupts or DMA)

# BIOS steps

~~performing the POST~~

load the MBR ,  
run the bootloader



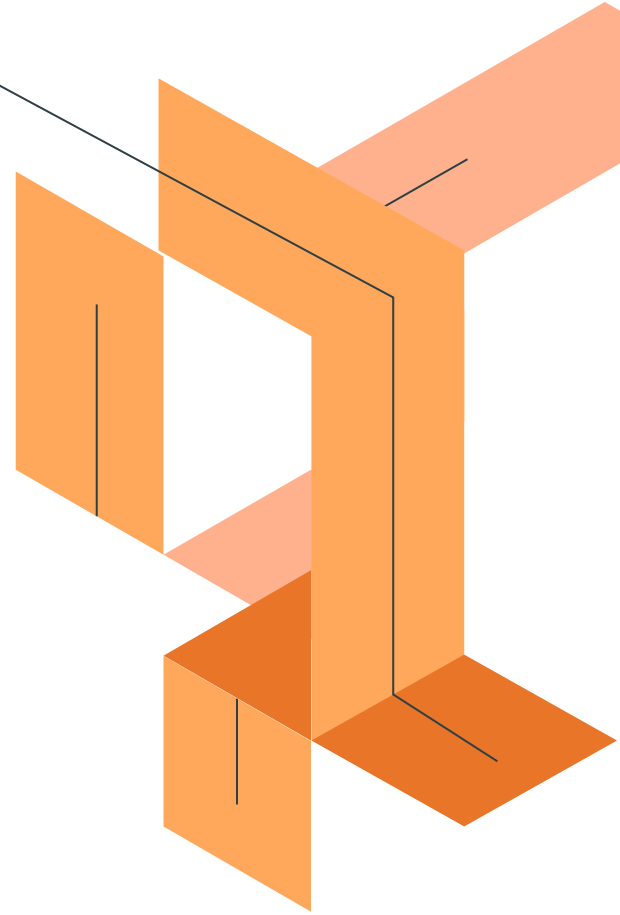
~~Initializing hardware~~

Search for Bootable  
Device

# BIOS Stage

## 3\_Search for Bootable Device:

- A bootable device is any storage device (like a hard drive, SSD, USB, CD/DVD, etc.).
- it contains a bootloader In a format the BIOS or UEFI can understand at a specific location (like MBR)



# BIOS Stage

## 3\_Search for Bootable Device:

For each device:

1. Checks if the device is connected and ready.
2. Reads the first sector (called MBR → Master Boot Record) of the device.
3. Looks at the first 446 bytes of the MBR this should contain the pre-bootloader code.
4. Checks for a boot signature at the end (0x55AA) this confirms it's bootable.
5. If everything checks out it jumps to that pre-bootloader code and hands over control.

If not bootable → try next device in list.

# BIOS Stage

## 3\_Search for Bootable Device:

### What is the Boot Sequence?

- ★ The **boot sequence** is the order in which your BIOS or UEFI firmware checks devices to find a **bootable one** (a device with an OS or bootloader).
- ★ It may be something like this:
  1. USB Drive
  2. Hard Disk
  3. CD/DVD
  4. Network Boot (PXE)



# BIOS Stage

## 3\_Search for Bootable Device:

We can change the boot sequence in two ways:

- ❖ Boot Menu (Temporary)
  - Lets you choose a boot device *just for this one time*
  - Useful for booting from a USB stick or recovery disk

Use the ↑(Up) and ↓(Down) arrow keys to move the pointer to the desired boot device.  
Press [Enter] to attempt the boot or ESC to Cancel. (\* = Password Required)

Boot mode is set to: UEFI; Secure Boot: ON

UEFI BOOT:

Windows Boot Manager  
Onboard NIC(IPV4)  
Onboard NIC(IPV6)

OTHER OPTIONS:

BIOS Setup  
Device Configuration  
BIOS Flash Update  
**Diagnostics**  
SupportAssist OS Recovery  
Intel(R) Management Engine BIOS Extension (MEBx)  
Change Boot Mode Settings  
Exit Boot Menu and Continue

# BIOS Stage

## 3\_Search for Bootable Device:

We can change the boot sequence in two ways:

- ❖ BIOS/UEFI Settings (Permanent Boot Order)
  - You can find a Boot Order or Boot Priority section

- ▶ CPU Configuration
- ▶ Power & Performance
- ▶ Thunderbolt(TM) Configuration
- ▶ **Trusted Computing**
- ▶ Network Stack Configuration
- ▶ NVMe Configuration
- ▶ USB Configuration

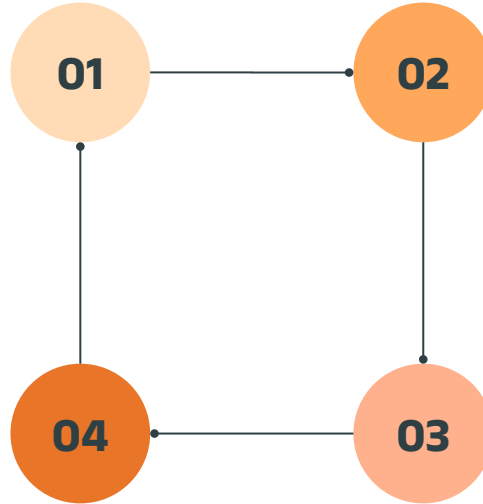
Trusted Computing Settings


→←: Select Screen  
↑↓: Select Item  
Enter: Select  
+/-: Change Opt.  
F1: General Help  
F2: Previous Values  
F3: Optimized Defaults  
F4: Save & Exit  
ESC: Exit

# BIOS steps

 perform the POST

load the MBR ,  
run the bootloader



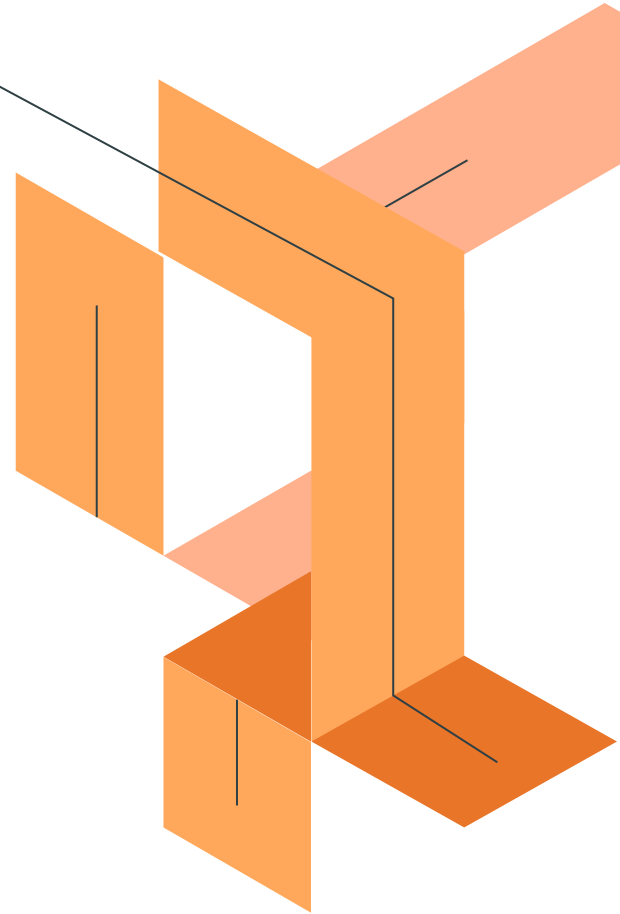
 Initializing hardware

 Search for Bootable  
Device

# BIOS Stage

## ***4\_MBR (Master Boot Record) and Boot Loader:***

- After POST, BIOS's job is to find something that can boot the system.
- The MBR is the first small section (512 bytes) of your hard drive.
- It contains: A tiny bootloader program (just enough to start things) A partition table (tells where the OS is installed) If BIOS finds an MBR, it loads it into memory and runs it.

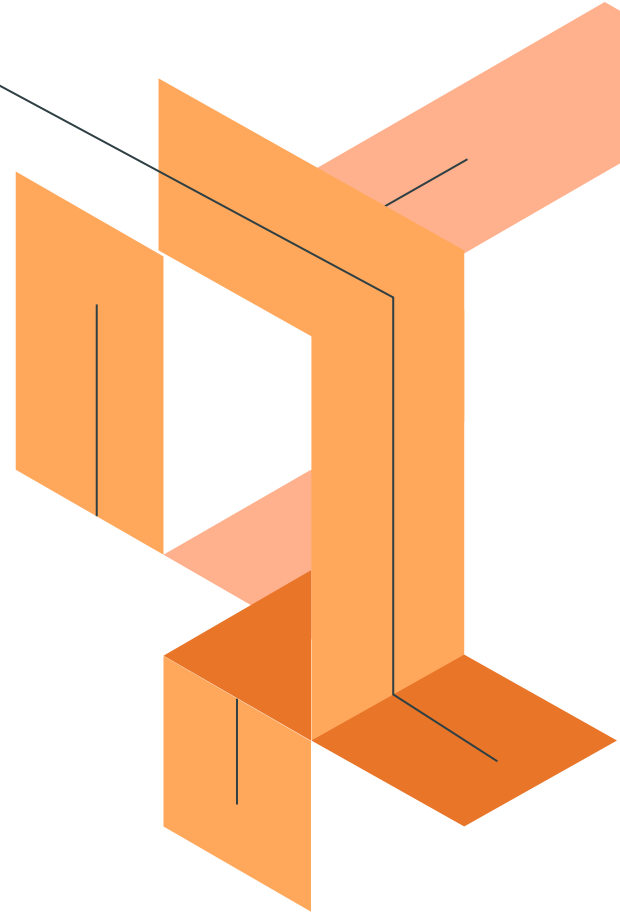


# BIOS Stage

## ***4\_MBR (Master Boot Record) and Boot Loader:***

The pre-bootloader in the MBR:

- Reads the partition table
- Finds the active partition (the one marked bootable)
- Loads the real bootloader from that partition



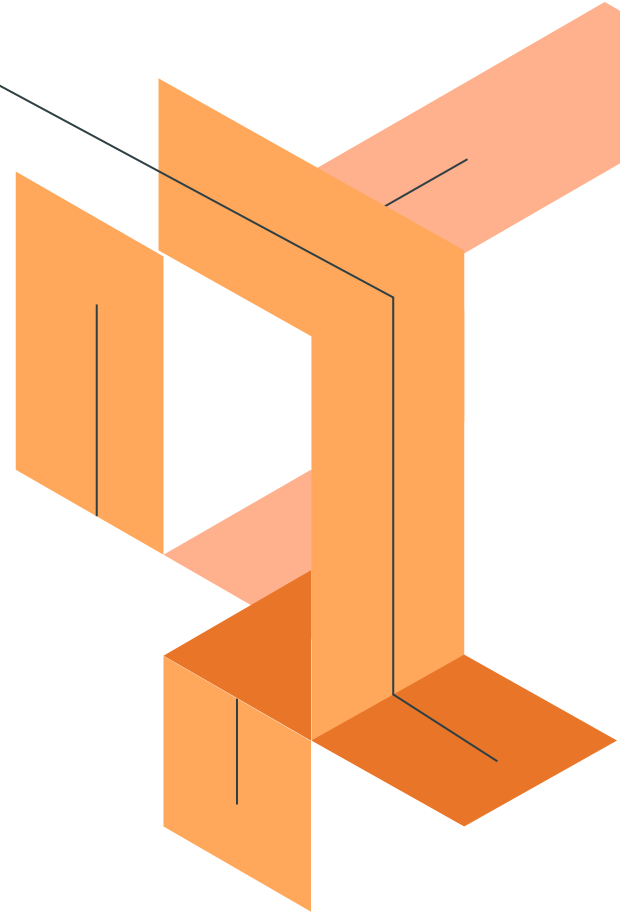
# BIOS Stage

## *4\_MBR (Master Boot Record) and Boot Loader:*

Imagine your hard drive is a big storage box.

- The MBR is a little label stuck to the outside of the box.
- That label says:

"Here's a small piece of code to get started, and by the way the operating system is inside drawer #2, go check there!"





# BIOS steps

performing the POST

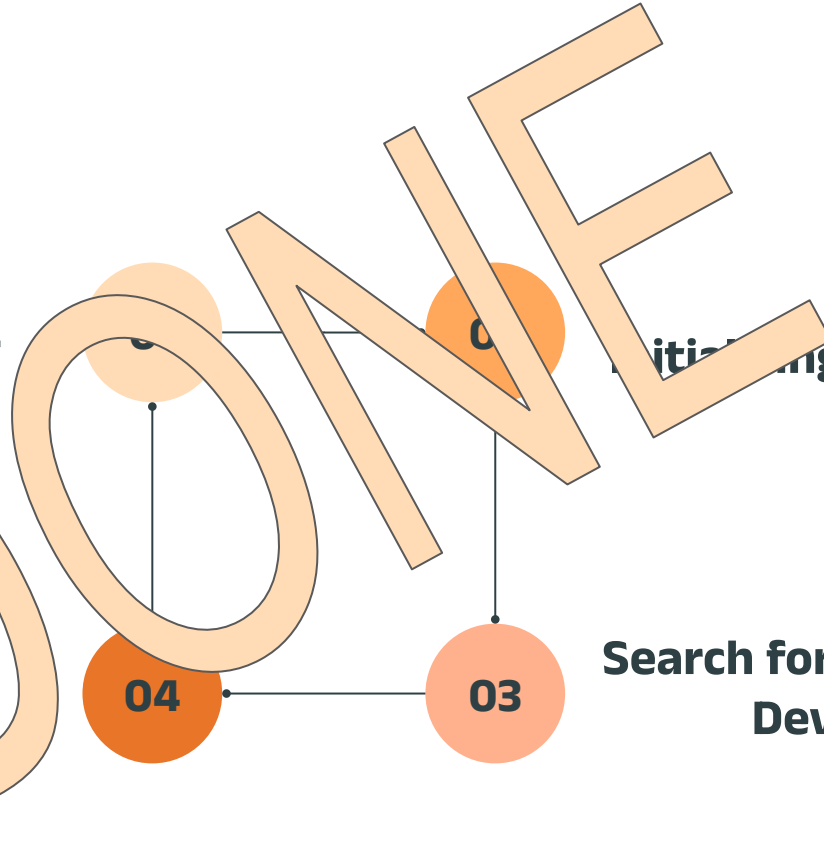
initializing hardware

load the MBR ,  
run the boot loader

Search for Bootable  
Device

04

03



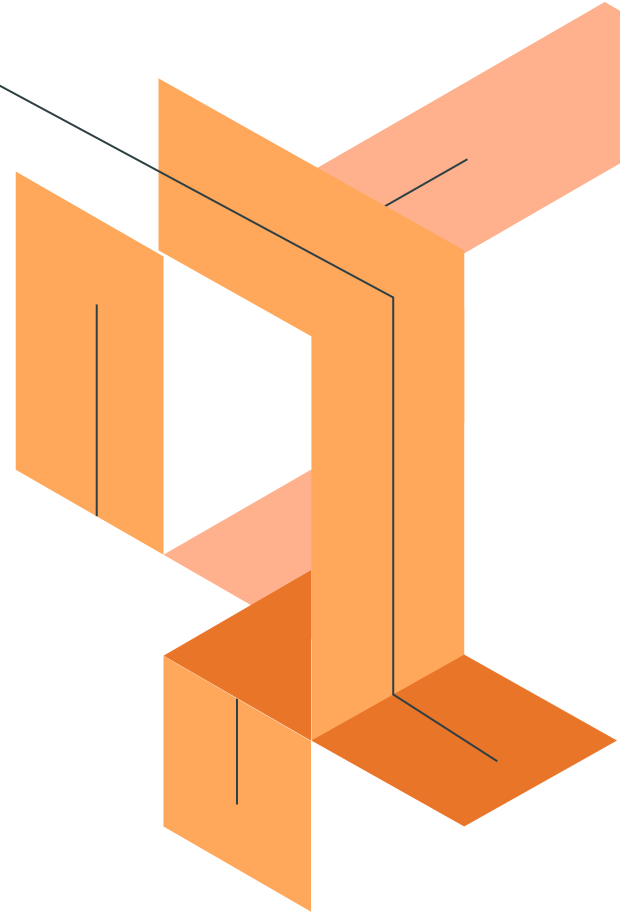
# BIOS Stage

## ***NOTE:***

- The first step really has nothing whatever to do with Linux.

This is the hardware portion of the boot process and is the same for any operating system.

When power is first applied to the computer it runs the POST.



# Summary

- Checks if hardware is functioning (Power-On Self Test or **POST**).
- Initializes devices like keyboard, monitor, RAM, and storage.
- Finds a bootable device (like your hard drive or USB) then runs the MBR that runs the bootloader.

Think of it as:

- ★ BIOS is a tiny built-in program whose job is to make sure the hardware is working, then pass control to a much larger program → the Operating System.

# We didn't finish yet

- ★ As computers advanced, a modern alternative to BIOS was introduced.
- ★ it's called UEFI.
- ★ You can think of it as BIOS's smarter cousin with better features, faster performance, and support for new tech.

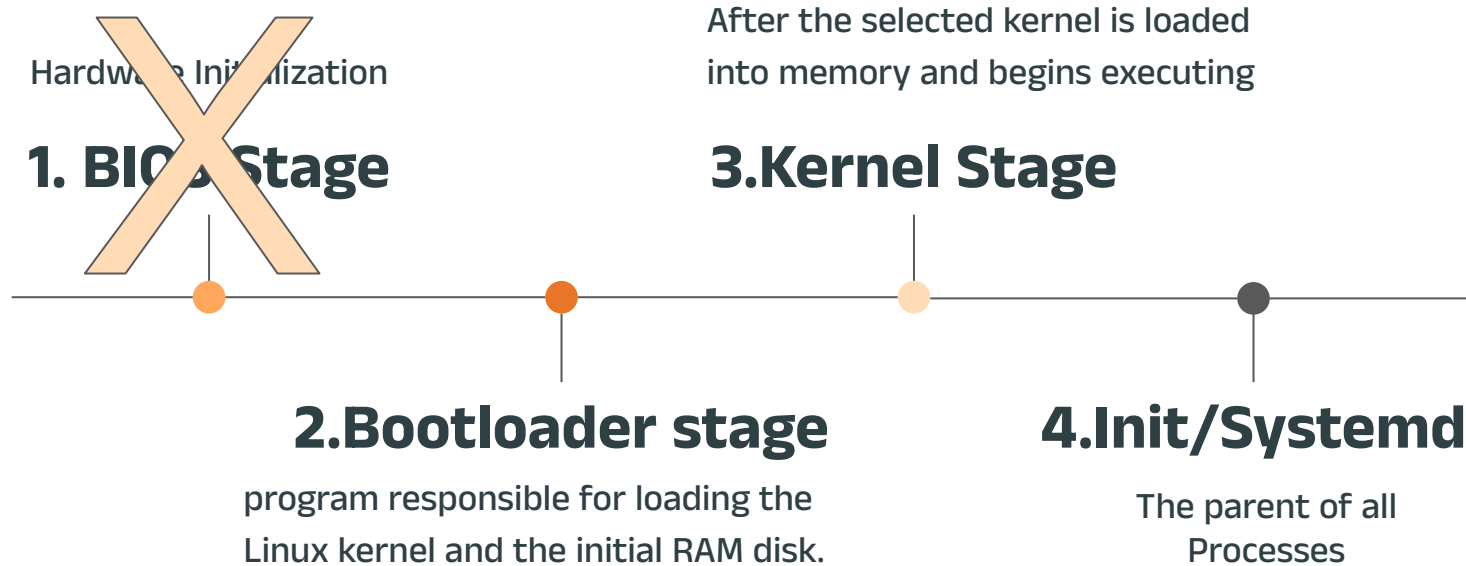
## Limitations of BIOS:

- ★ Can't boot from disks > 2TB.
- ★ Only supports 4 primary partitions on MBR.
- ★ No mouse support.
- ★ Legacy code, not secure.

# UEFI

- ★ Stands for: Unified Extensible Firmware Interface
- ★ Faster boot times
- ★ Graphical interface with mouse and keyboard
- ★ Secure Boot support (helps prevent malware during boot)
- ★ Supports disks larger than 2TB Uses GPT (GUID Partition Table) instead of MBR
- ★ Modular and more customizable
- ★ Most PCs made after ~2010 support UEFI

# Boot Process Stages

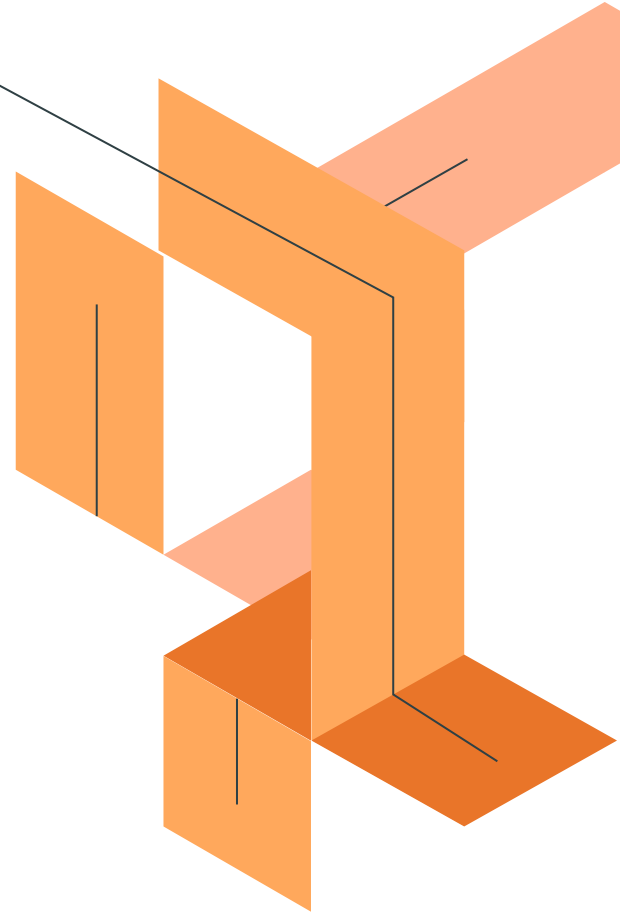


# Bootloader stage

## What is a bootloader?

- The bootloader is a small program that loads your operating system (OS).
- It runs after the BIOS finishes its job and before the OS starts.

★ "Load" means: Copy the file from storage (disk) → into RAM (memory) And prepare it to run (or start running it)



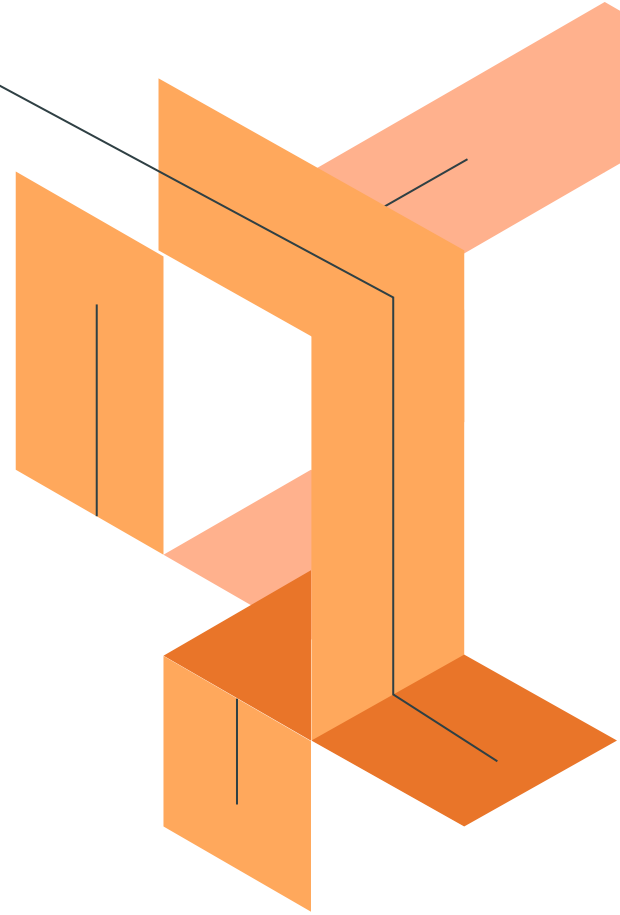
# Bootloader stage

## Why Do We Need a Bootloader?

Because:

1. BIOS is too simple to load an entire OS.
2. The OS (like Windows or Linux) is too big to fit in the early memory.

So we use a small "middle man" → the bootloader.





# Bootloader stage

## Where Is It?

The bootloader is stored in two parts:

| Part                                 | Where It Is  | What It Does                                   |
|--------------------------------------|--|--|
| ♦ <b>Pre-bootloader</b>              | First 446 bytes of the <b>MBR</b> (sector 0 of disk) | Tiny code to find and load the real bootloader |
| ♦ <b>Real bootloader</b> (like GRUB) | In a special <b>boot partition</b> (e.g., /boot)     | Loads the OS (like Linux or Windows)           |

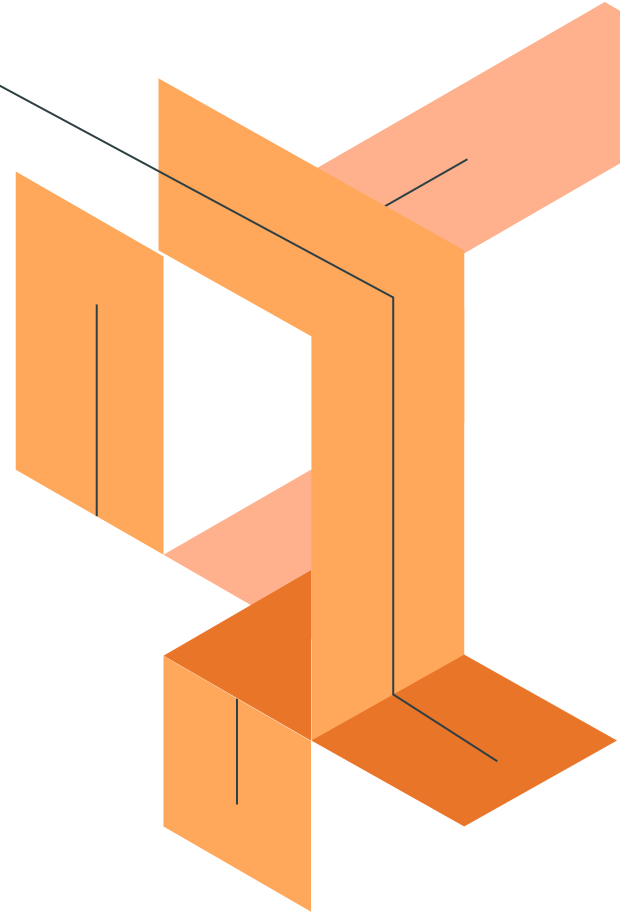
# Bootloader stage

## What it actually do?

The bootloader's job is to:

- Load the Linux kernel (the brain of Linux)
- Load a temporary helper file called `initrd` or `initramfs`

It loads them together into memory.



# Bootloader stage

## What is initrd?

- **initrd = initial RAM disk**
  - **Initial** → Because it's used at the **very beginning** of the boot process
  - **RAM** → Because it gets **loaded into memory (RAM)** not run from the hard disk
  - **Disk** → Because it acts like a **tiny virtual disk** (a mini-filesystem), even though it's not a physical disk
- It's a temporary mini-filesystem packaged into a file
- The kernel can't always boot your system alone, it needs help finding and mounting the real root filesystem.
- The initrd is used by the Linux kernel as a temporary filesystem for loading critical files into memory before the real root file system can be mounted.

# Bootloader stage

## What's inside initrd?

It's A small Linux filesystem (like a zip archive) Contains:

- ❖ Small programs (binaries)
- ❖ Shell scripts
- ❖ Drivers (kernel modules)
- ❖ Tools to find and mount your real root filesystem

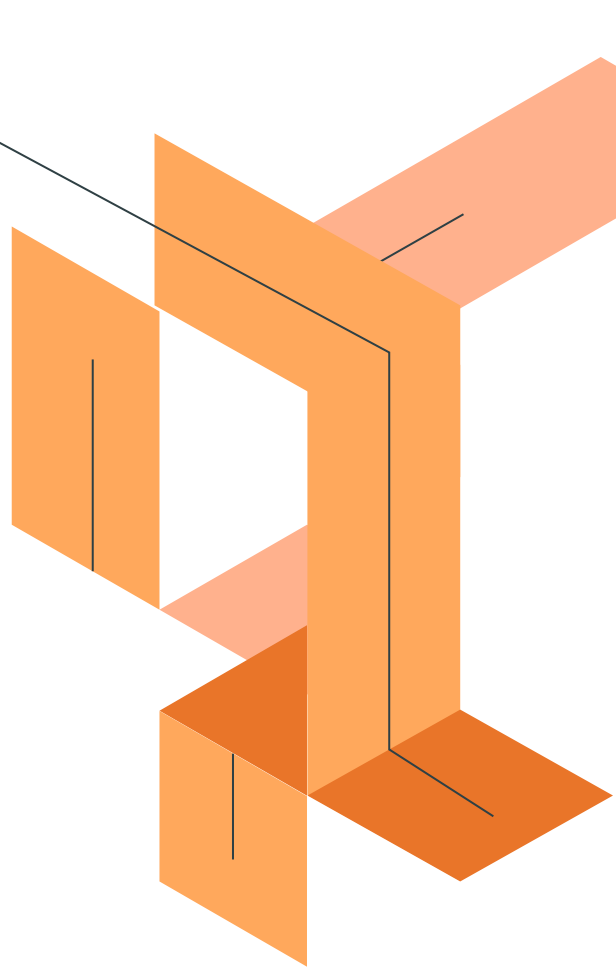


What is the filesystem?

It's where all your Linux files live Like /home, /etc, /bin, /usr



Mount" means: make a storage device (like a disk or USB) ready to use.



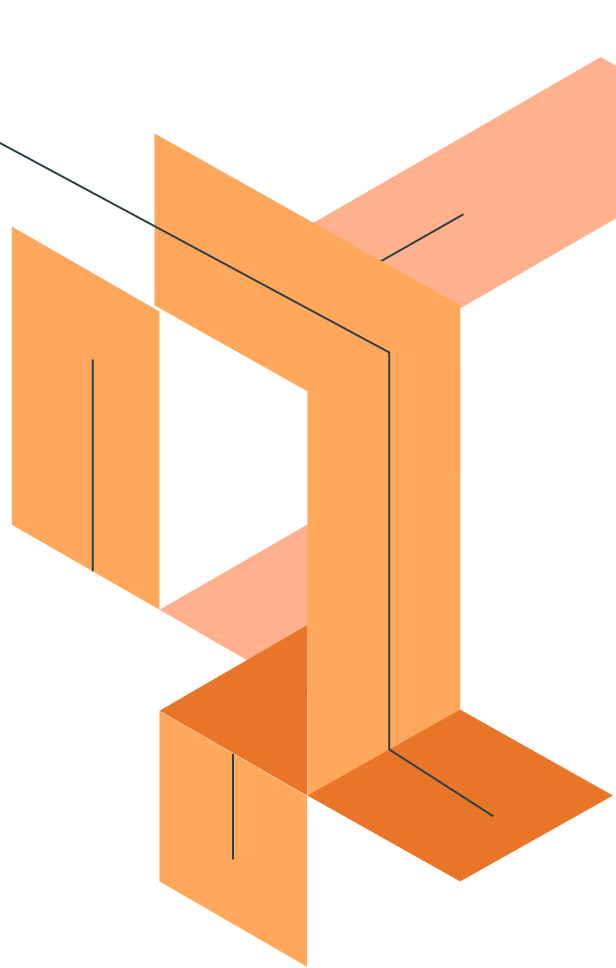
# Bootloader stage

## Why don't it just load the kernel only?

1\_ The modular Kernel's Problem that It doesn't yet have drivers to talk to your hard disk, SSD, or file system

It doesn't know:

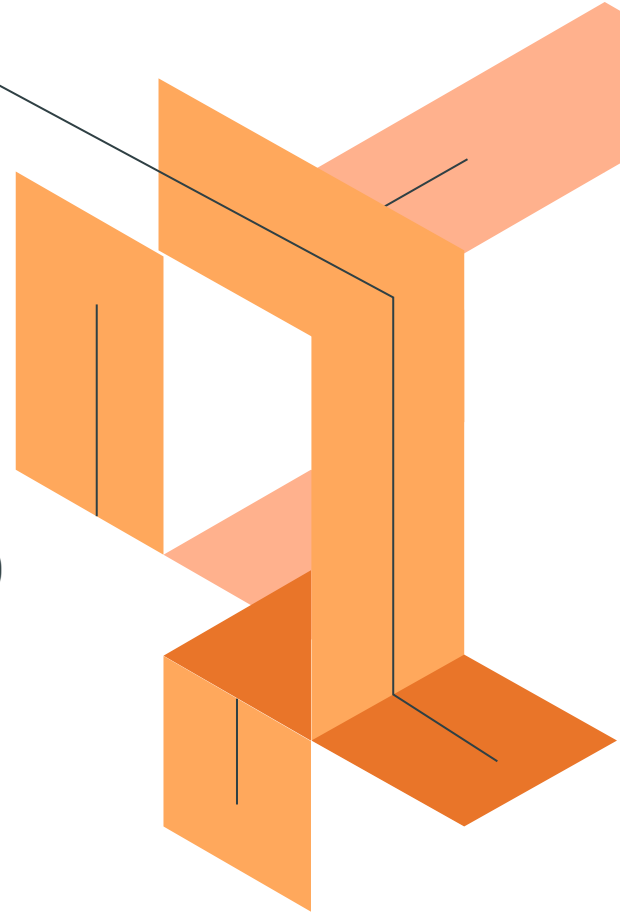
- What kind of hardware you have
- Where the root filesystem is
- What filesystems are supported (ext4, xfs, btrfs, etc.)



# Bootloader stage

## What is the modular kernel?

- ★ Modular kernel is a kernel that's designed to be flexible, it doesn't include all possible features and drivers built-in.  
Instead, it can load extra features (called modules) when needed.
- ★ Many modern Linux distributions use a modular kernel.
- ★ The initrd contains the essential drivers and scripts required to detect and load the modules needed to access the root filesystem.

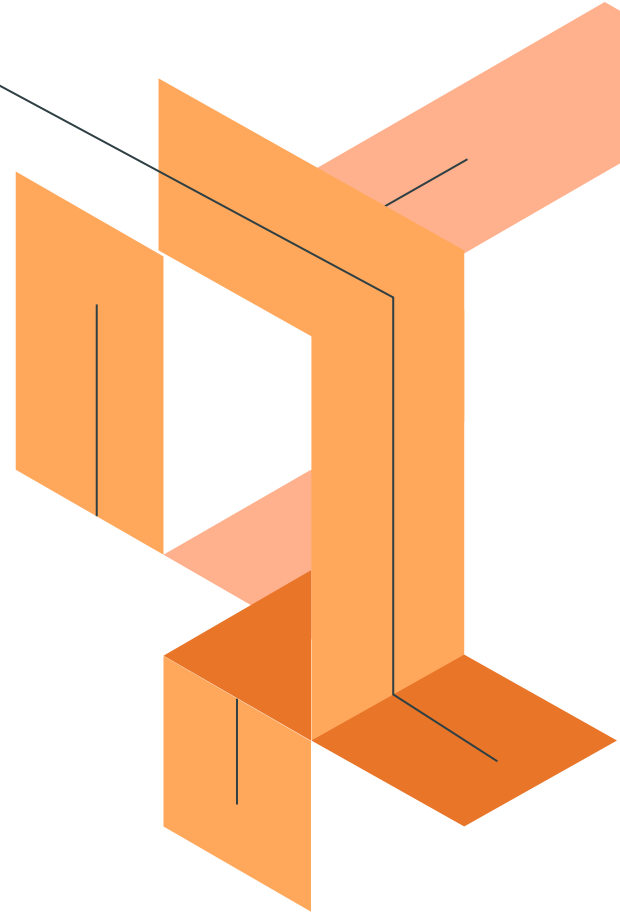


# Bootloader stage

**Why don't it just load the kernel only?**

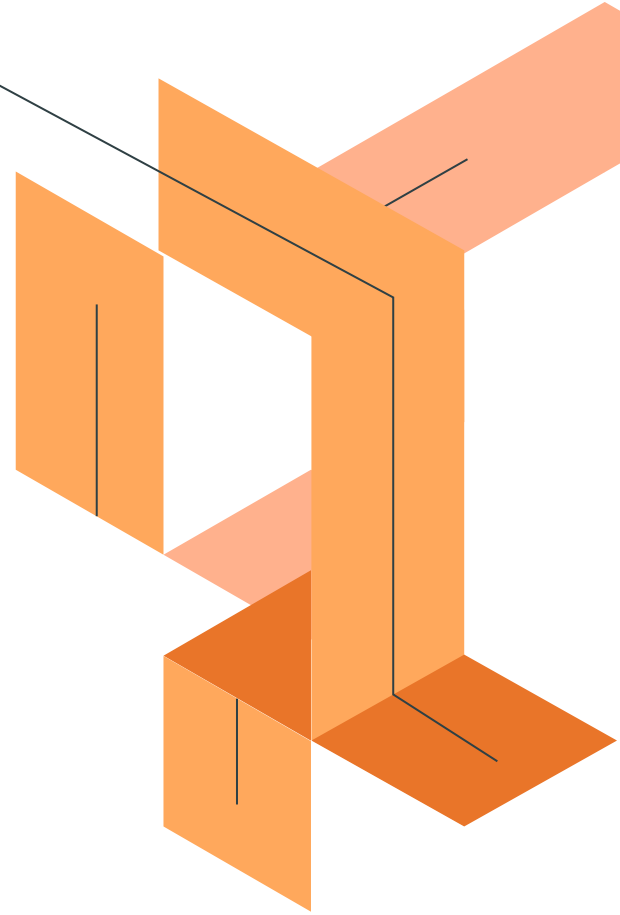
**2\_ Pre-boot Tasks:**

initrd can perform other necessary tasks such as decrypting encrypted filesystems, or setting up networking for network-based filesystems.



# Bootloader stage

Most Linux boot loaders, such as **GRUB** (Grand Unified Bootloader), can present a user interface for choosing alternative options for booting Linux, and even other operating systems that might be installed (e.g., MS Windows). This is called dual-boot (or multi-boot).





# Bootloader stage

| Term                    | Meaning   |
|-------------------------|---|
| <b>Bootloader</b>       | Small program that loads Linux Kernel and initrd                  |
| <b>Kernel</b>           | Core part of Linux, manages everything                            |
| <b>initrd/initramfs</b> | Temporary tools used by kernel before loading the real filesystem |
| <b>init/systemd</b>     | First main Linux program that starts everything                   |

# Grub Menu -> Bootloader menu

GNU GRUB version 2.04

\*Linux Mint 20 Xfce

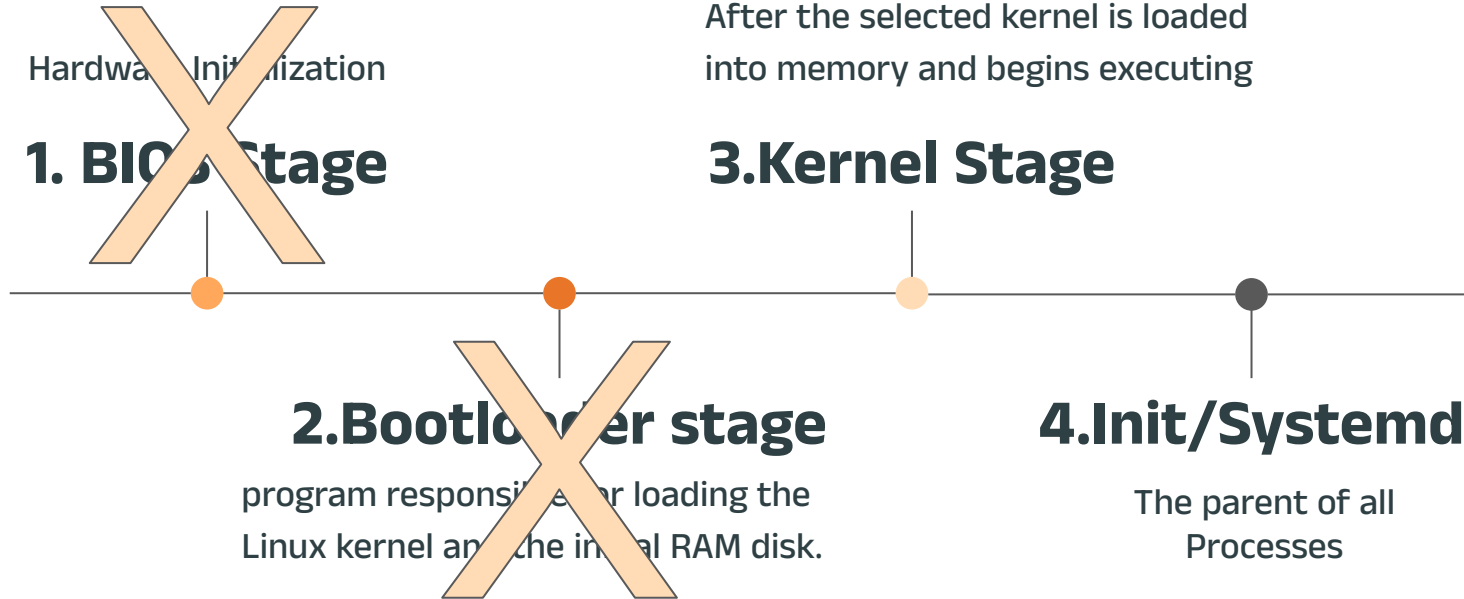
Advanced options for Linux Mint 20 Xfce

Ubuntu 19.04 (19.04) (on /dev/sda1)

Advanced options for Ubuntu 19.04 (19.04) (on /dev/sda1)

Use the ↑ and ↓ keys to select which entry is highlighted.  
Press enter to boot the selected OS, 'e' to edit the commands  
before booting or 'c' for a command-line.

# Boot Process Stages



# Kernel Stage

## *What is the Kernel?*

- The kernel is a compressed, self-extracting file.
- located in the `/boot` directory.
- Filenames usually start with `vmlinuz` (stands for Virtual Memory Linux).
- You can list installed kernels using: `ls /boot`.

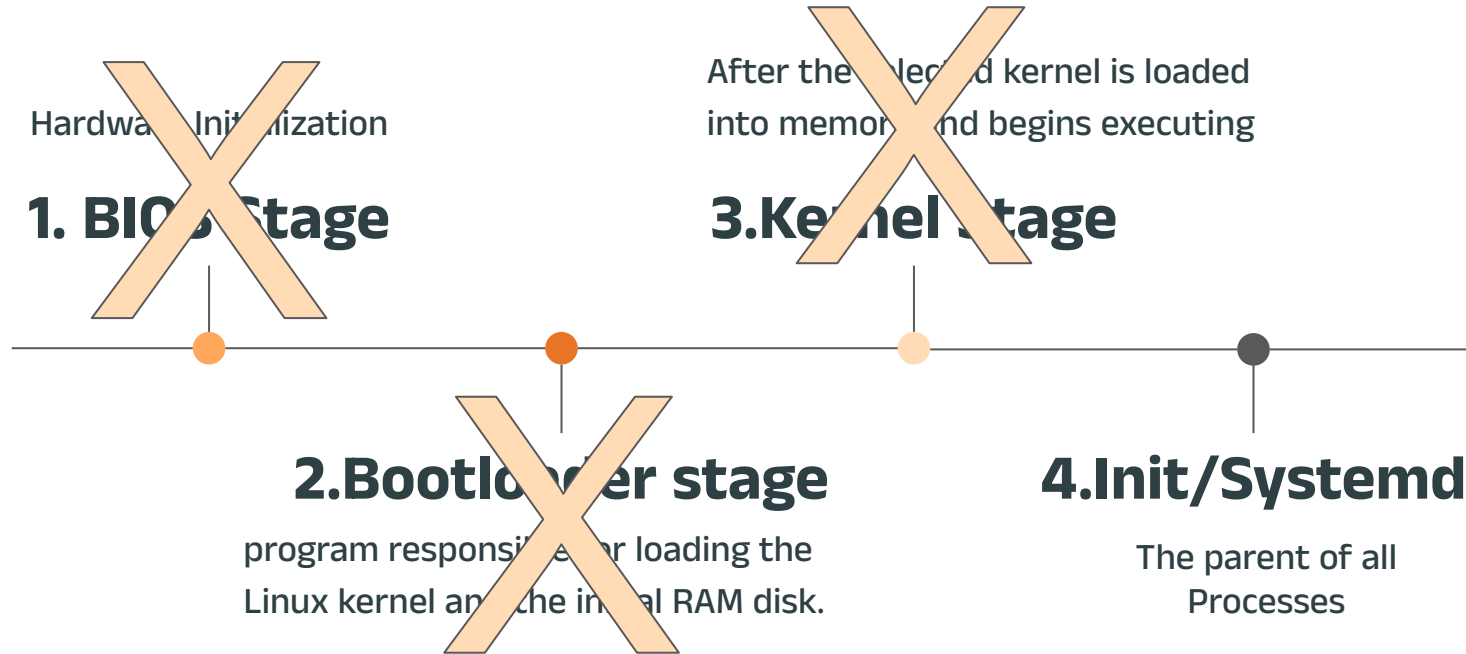
# Kernel Stage

*Once the kernel starts running, it:*

1. Decompresses itself
2. Initializes essential hardware interfaces
3. Uses the `initrd` to mount the real root filesystem
4. Starts `systemd`, the first user space process

At this point, the boot process ends, and the system is ready to start background services and user-level programs.

# Boot Process Stages



# Init/Systemd

- ★ **After the boot process is completed and the kernel is loaded into memory.**
- ★ **it starts the init process, which brings the Linux computer to an operational state for productive work.**
- ★ **But to talk about the init process, we first need to understand Linux processes. So, let's skip forward in time, and imagine that the system has started successfully.**

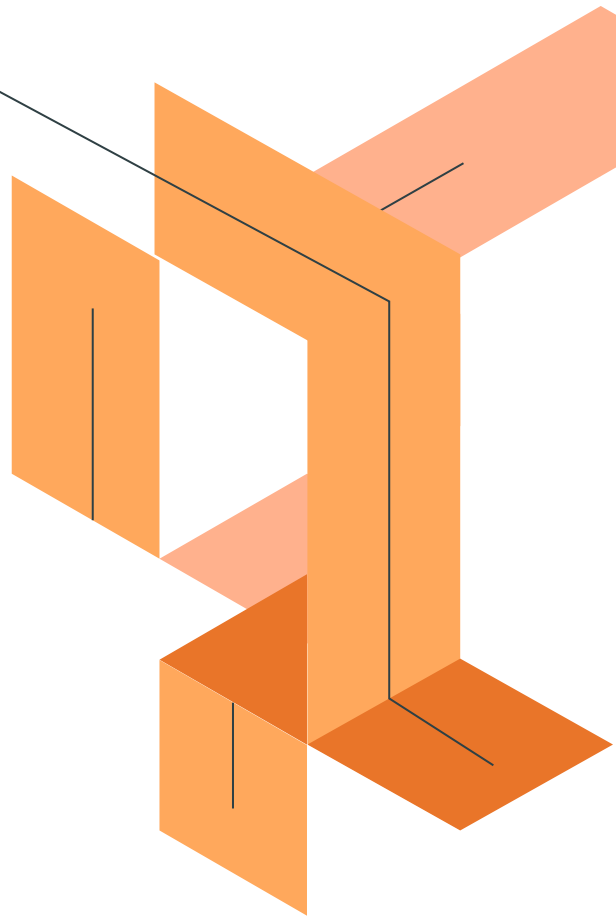
# HANDS ON





# Hands on #1

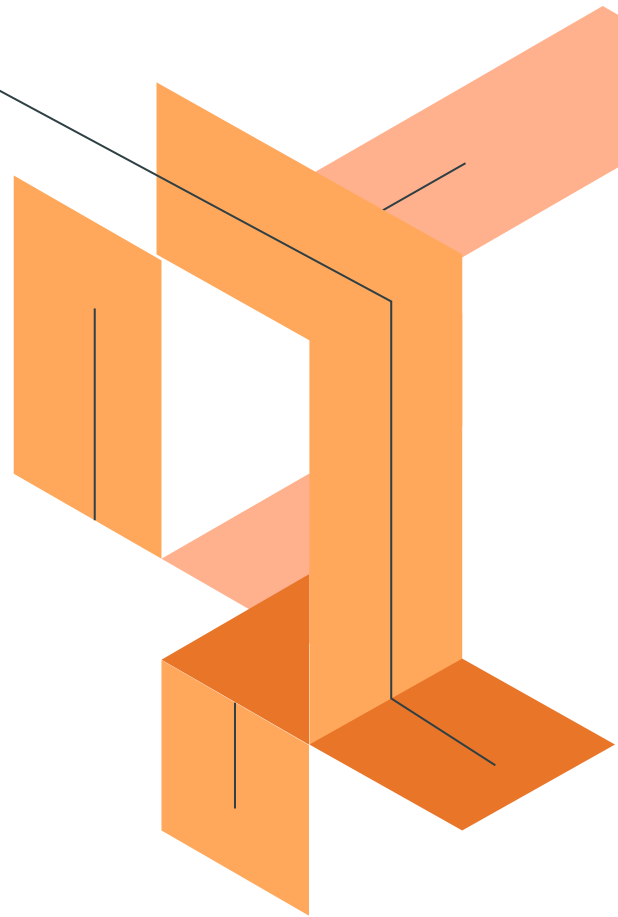
- List the kernels installed on your system.
- Kernel files start with `vmlinuz`. Filter the output to show only those.



# Hands on #1

→ Solution:

```
ls /boot | grep vmlinuz
```





# Let's Play!

---

# Table of contents

01

## Boot Process

sequence of steps  
your computer goes  
through when you  
turn it on

02

## Processes

instance of a running  
program

03

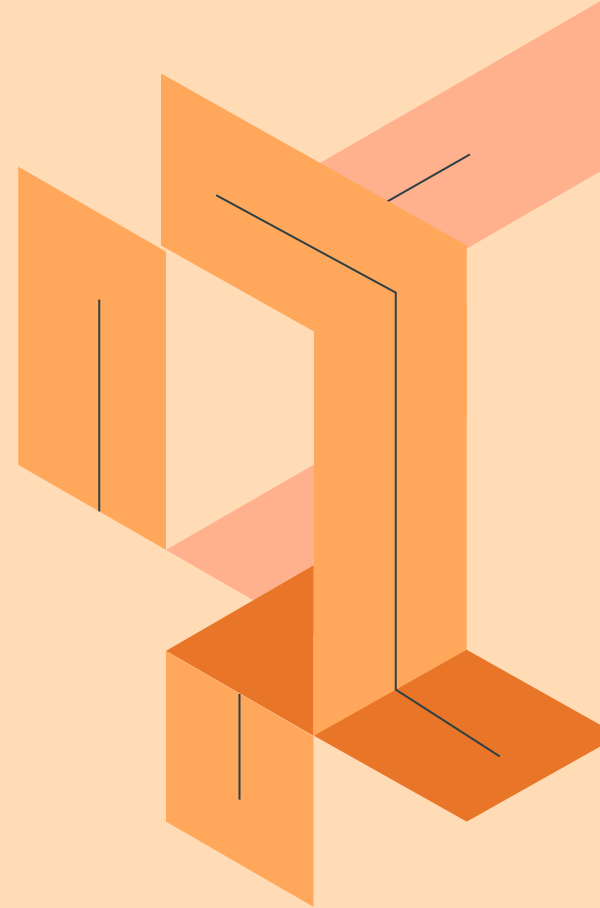
## Systemd

**init system** (the first  
real program started  
by the kernel).

02

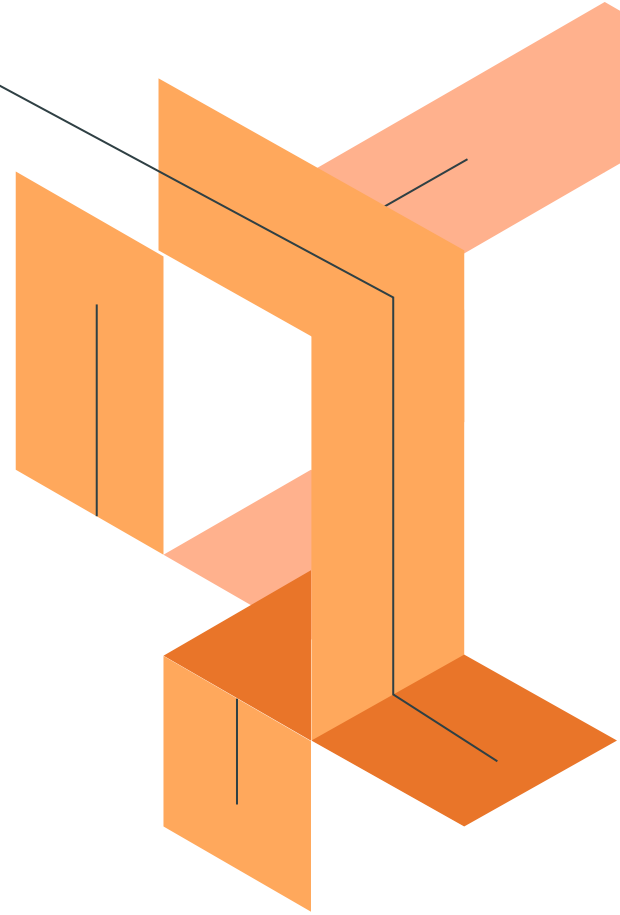
---

# Processes



# Processes

- A process (or task) is an instance of a program.
- Each process has its own memory space and resources allocated to it.
- Each process has the illusion that it is the only process on the computer, but in reality all processes share system resources like the CPU and memory.



# Processes

- A bash shell itself is a process, and running a script or program in it starts a new process.
- Each process in Linux has a process ID (PID) and is associated with a specific user and group.
- Linux is a multitasking operating system, which means that multiple processes can run at the same time.
- The kernel uses a scheduler to manage how different programs share the CPU.

# Commands





# Processes

## **ps**

Lists processes associated with the current terminal session.

## **ps -e or ps -A**

prints all processes within the system.

## **ps -f**

prints a more detailed output.

## **ps -u [user name]**

prints a more detailed output.

## **ps -C [process name]**

searches for a particular process name.



# Processes

## **top**

Shows a continuously updating display of the system processes with more information.

## **pstree**

Display processes in a tree format.

## **ps aux --sort=-%mem**

Sort processes based on memory usage

## **ps aux --sort=-%cpu**

Sort processes based on CPU usage



# Jobs



# Jobs

- A job is a concept used by the shell.
- It is a unit of work performed by the user, which may consist of one or more processes.
- For example, `ls | head` is a job consisting of two processes.

# Jobs

- A job is identified by a job ID (JID).
- You can list the jobs of the current shell by running `jobs`

# Process VS Jobs



# Process VS Jobs

- We will not go deep into the differences between processes and jobs, but just note that they are distinct but related concepts, and that some commands take PID while others take JID.

# Foreground Processes





# Foreground Processes

- These are initialized and controlled through a terminal session.
- There must be a user connected to the system to start such processes, as they don't start automatically as part of the system functions/services.
- Examples are user programs like web browsers, image editors, etc.
- When a process is run in the foreground, no other process can be run on the same terminal until the process is finished or killed.

# **Background Processes**



# Background Processes

- These are not connected to a terminal session and don't expect any user input.
- They usually start automatically as part of the system functions/services.
- Examples are update managers, network managers, etc.

# Commands



# Foreground and background processes

## **[command] &**

- user programs can be manually run as background processes.
- To run a process in the background, add an ampersand & at the end of the command.
- This launches the command in the background and returns control of the terminal to the user.

# Foreground and background processes

## **fg %JID**

- To bring a background or stopped (suspended) process to the foreground, use the fg command followed by the job ID.

## **bg %JID**

- To resume a suspended process in the background, use the bg command

# Deamon Process



# Deamon Process

- A daemon is a type of background process that runs continuously, typically performing system-related tasks.
- Daemon processes are often started during system boot-up and run in the background without any user interaction.
- Daemons typically have no controlling terminal, which is indicated by a ? in the TTY field of the ps command's output.



# **Process Relationships**



# Process Relationships

- 1 — **Parent Process**
- 2 — **Child Process**
- 3 — **Orphan Process**
- 4 — **Zombie Process**

# Process Relationships

1

## Parent Process

- ★ A parent process creates another process, either directly or indirectly.
- ★ Every process has a parent process, except for `systemd`.
- ★ When a process is created, it inherits various attributes from its parent process, such as its working directory.

# Process Relationships

2

## Child Process

- ★ A child process is one created by another process (its parent process).
- ★ Child processes inherit most of their attributes from their parent process but can also have their own unique attributes.
- ★ For example, a shell may create a child process that has its own environment variables and command-line arguments.

# Process Relationships

3

## Orphan Process

- ★ An orphan process is a process whose parent has terminated before them.
- ★ Orphan processes are re-parented to the init system, typically systemd, which continues to manage them.

# Process Relationships

4

## Zombie Process

- ★ A zombie process is a process that has completed execution but still has an entry in the process table, as its parent process has not yet retrieved its exit status.
- ★ Although zombies do not consume system resources like memory or CPU, they do occupy a slot in the process table
- ★ Zombie processes are usually terminated once the parent process retrieves the exit status.

# **Process Management**



# Process Management

## Kill [signal] PID

- The kill command doesn't exactly “kill” processes;
- rather, it sends them signals.
- Signals are one of several ways that the operating system communicates with programs.



# Process Management

Signals can be specified in three ways:



**kill -15 PID**

1\_ By number

**kill -TERM PID**

2\_ By name

**kill -SIGTERM PID**

3\_ By name prefixed with SIG

# Process Management

There are many signals, but the most common ones are:

| Number | Signal | Description                       |
|--------|--------|-----------------------------------|
| 9      | KILL   | Terminate immediately, hard kill. |
| 15     | TERM   | Terminate whenever, soft kill.    |
| 19     | STOP   | Pause the process. (CTRL+Z)       |
| 18     | CONT   | Resume the process.               |

The default signal is 15 or TERM (terminate).

# Process Management

## **killall [name]**

- This command sends a signal to all instances (processes) of a specific program name.
- Unlike kill, which targets a specific PID, killall targets all processes with the specified name.



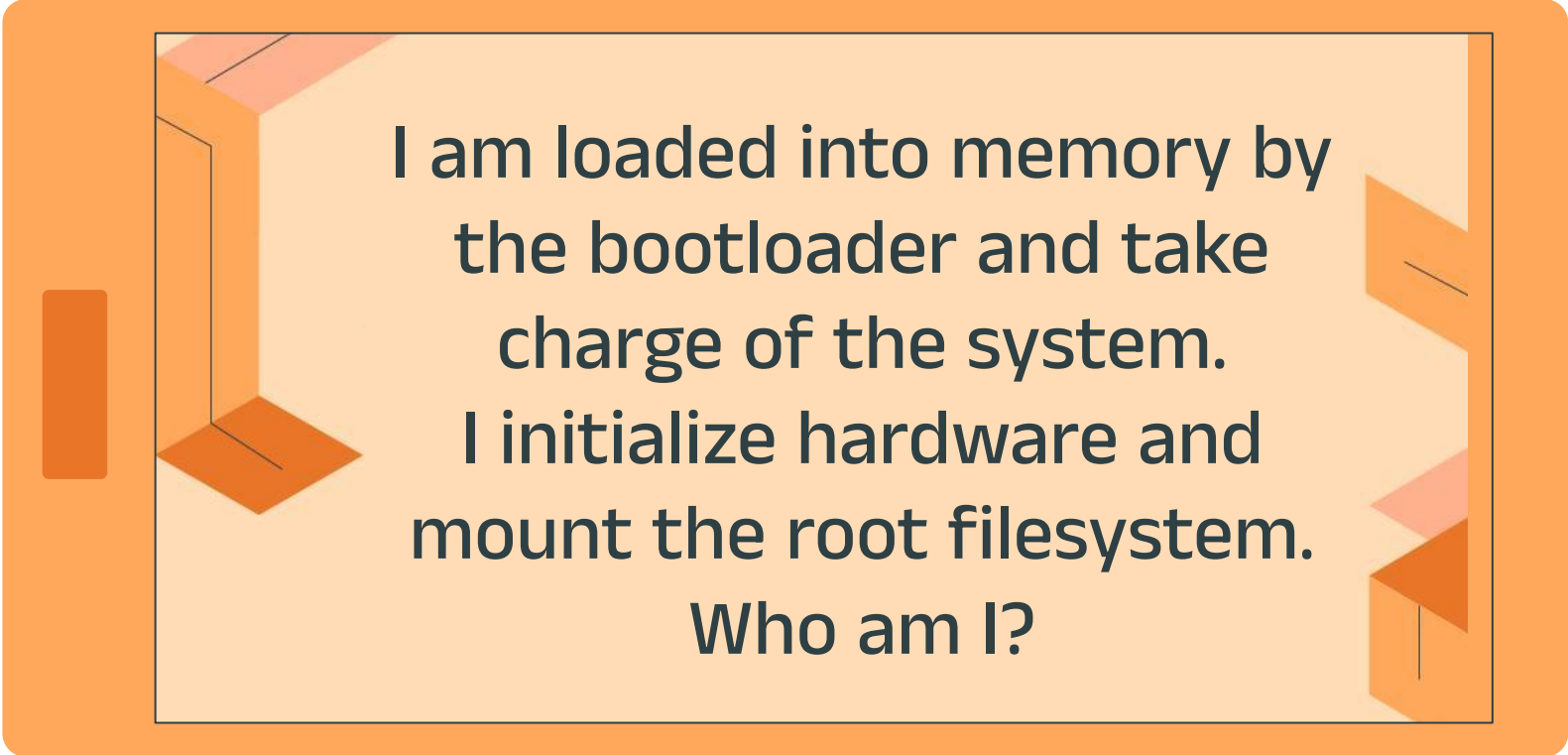
# Let's Play!

---

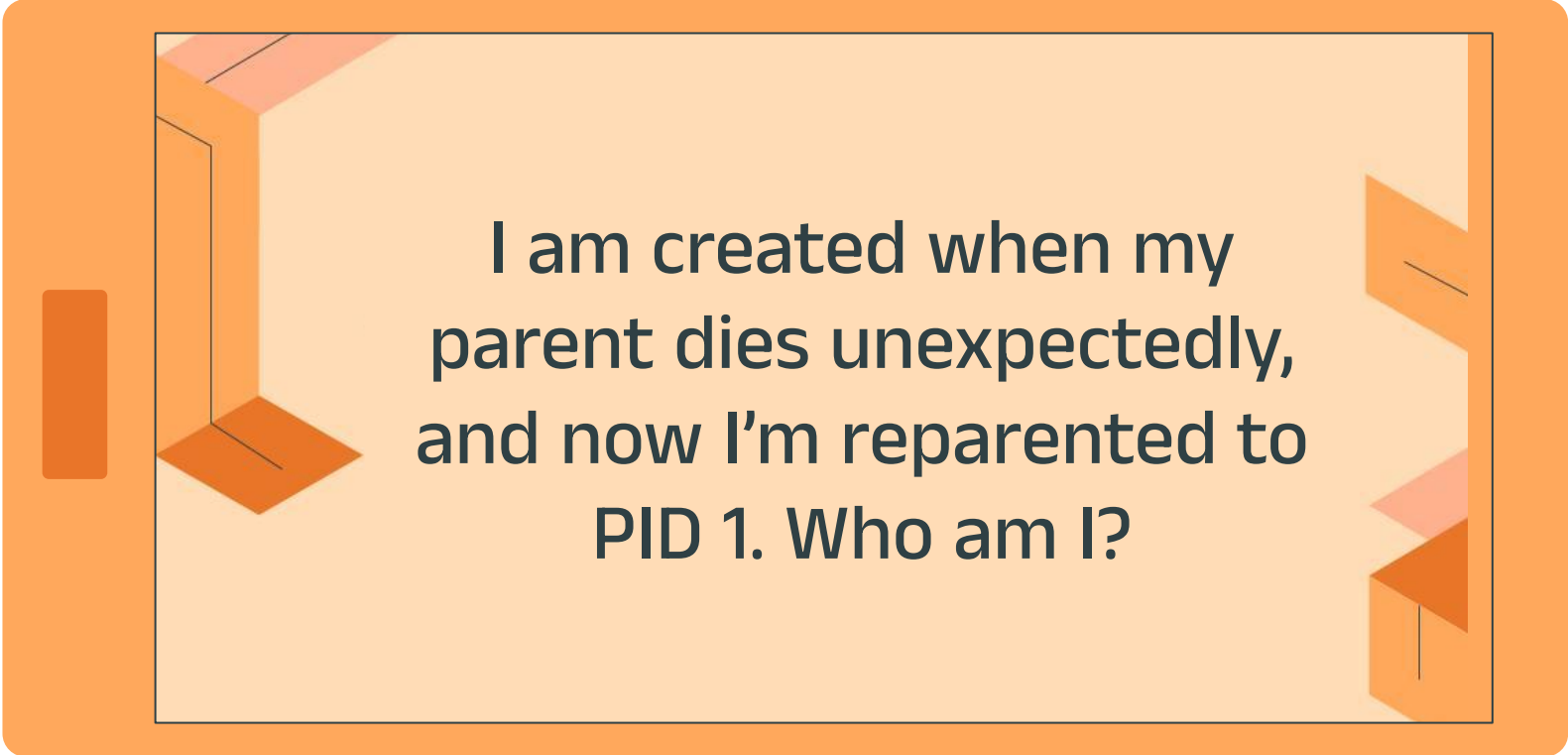


# Who am I?

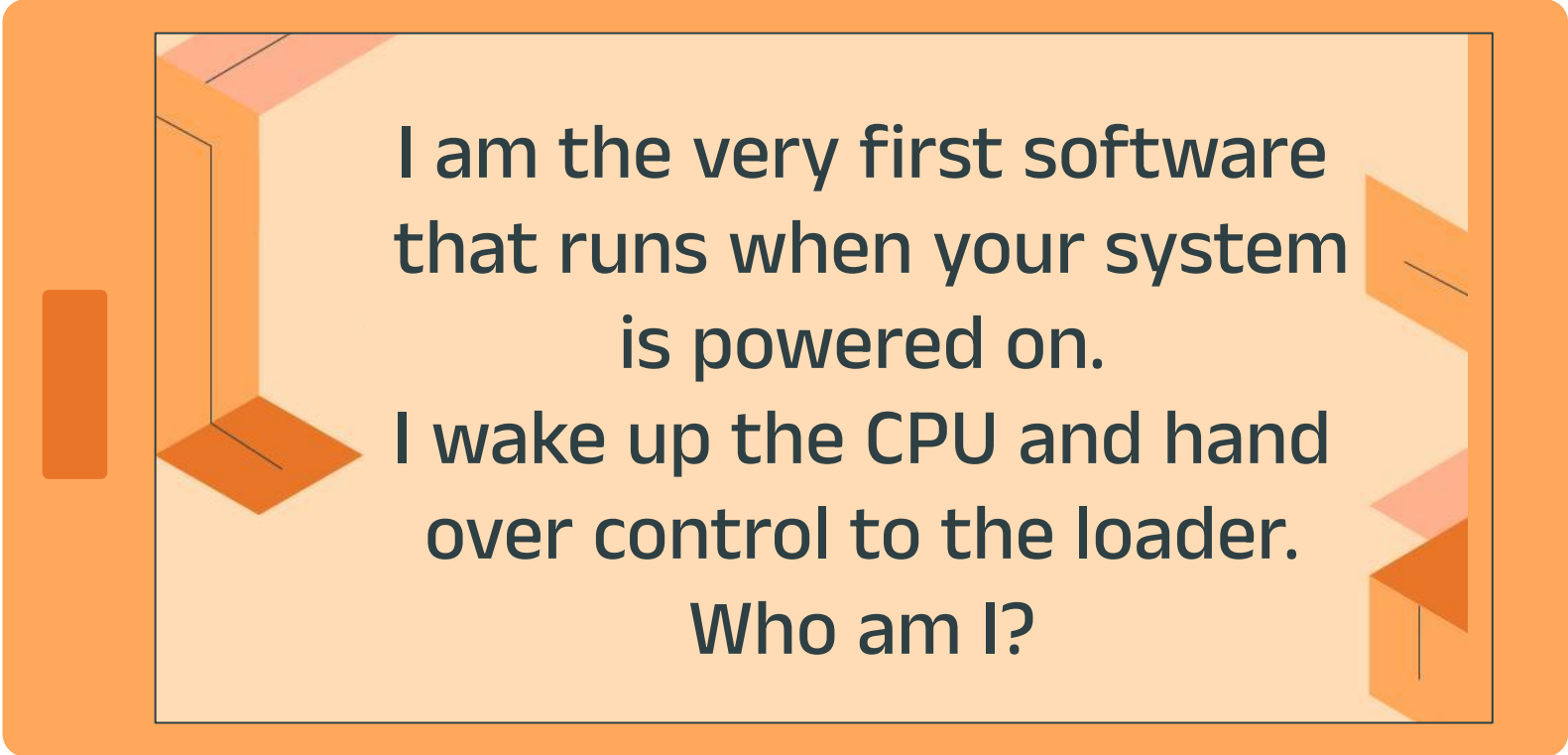
---



I am loaded into memory by  
the bootloader and take  
charge of the system.  
I initialize hardware and  
mount the root filesystem.  
Who am I?



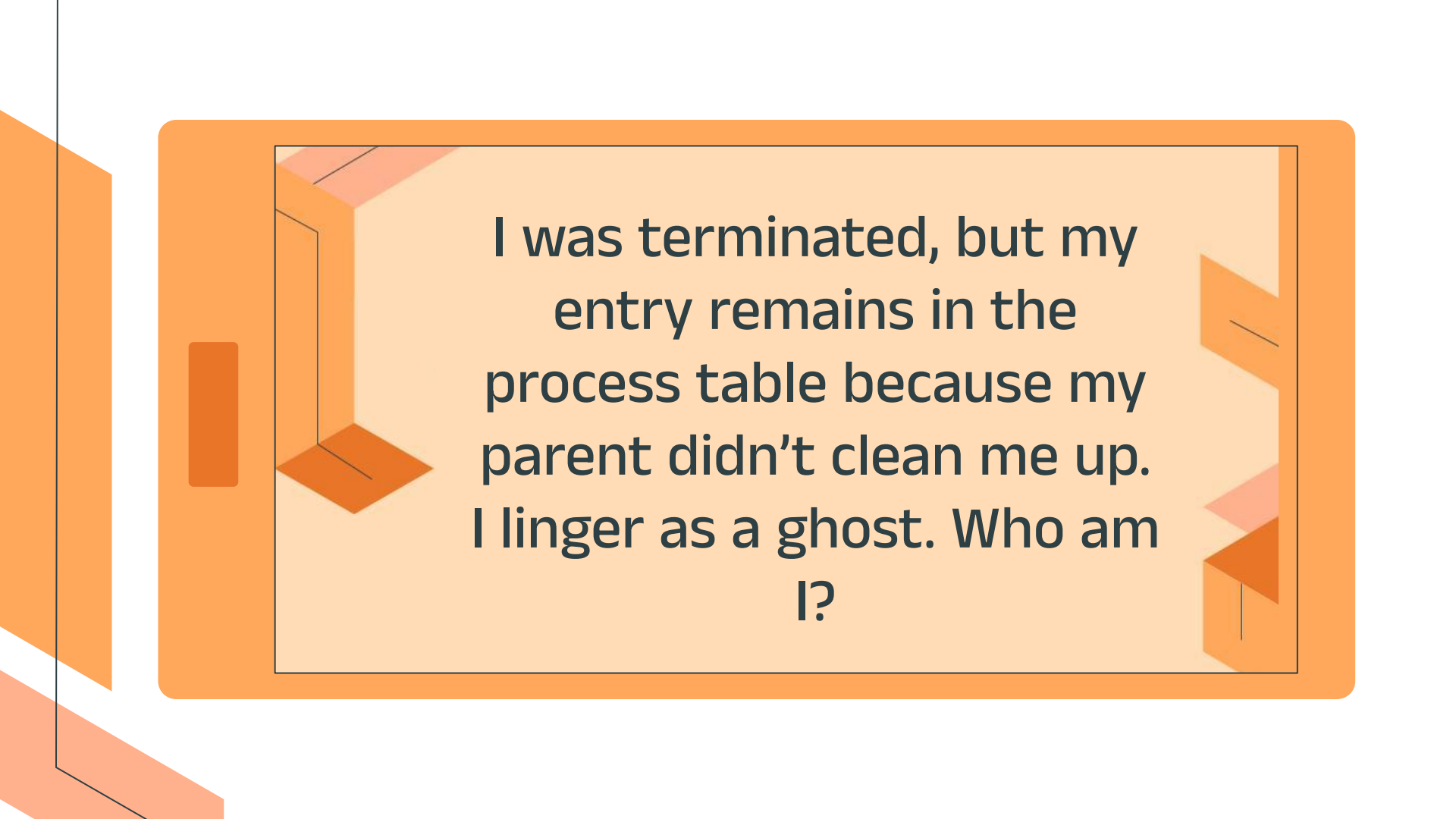
I am created when my  
parent dies unexpectedly,  
and now I'm reparented to  
PID 1. Who am I?



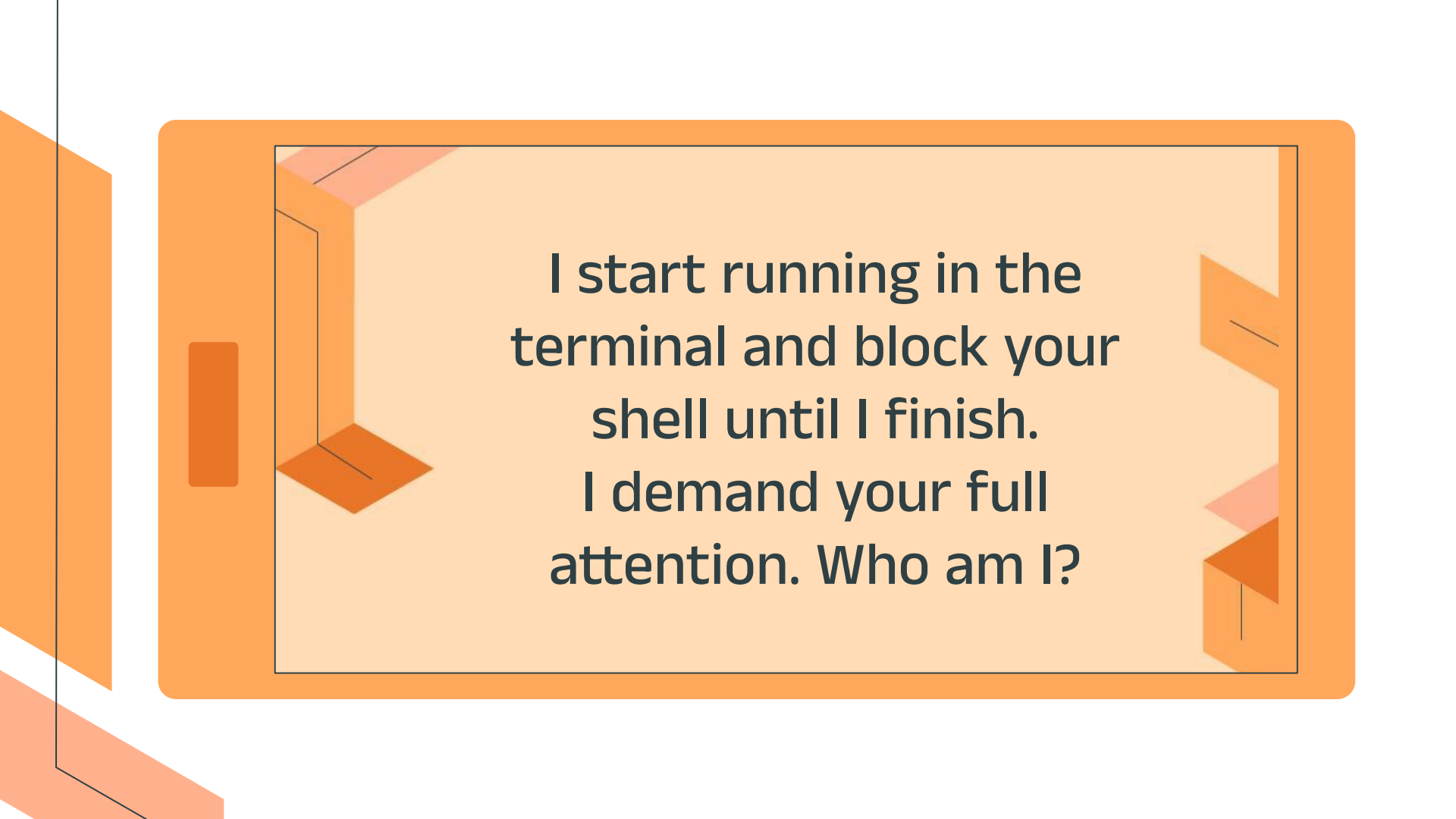
I am the very first software  
that runs when your system  
is powered on.

I wake up the CPU and hand  
over control to the  
bootloader. Who am I?

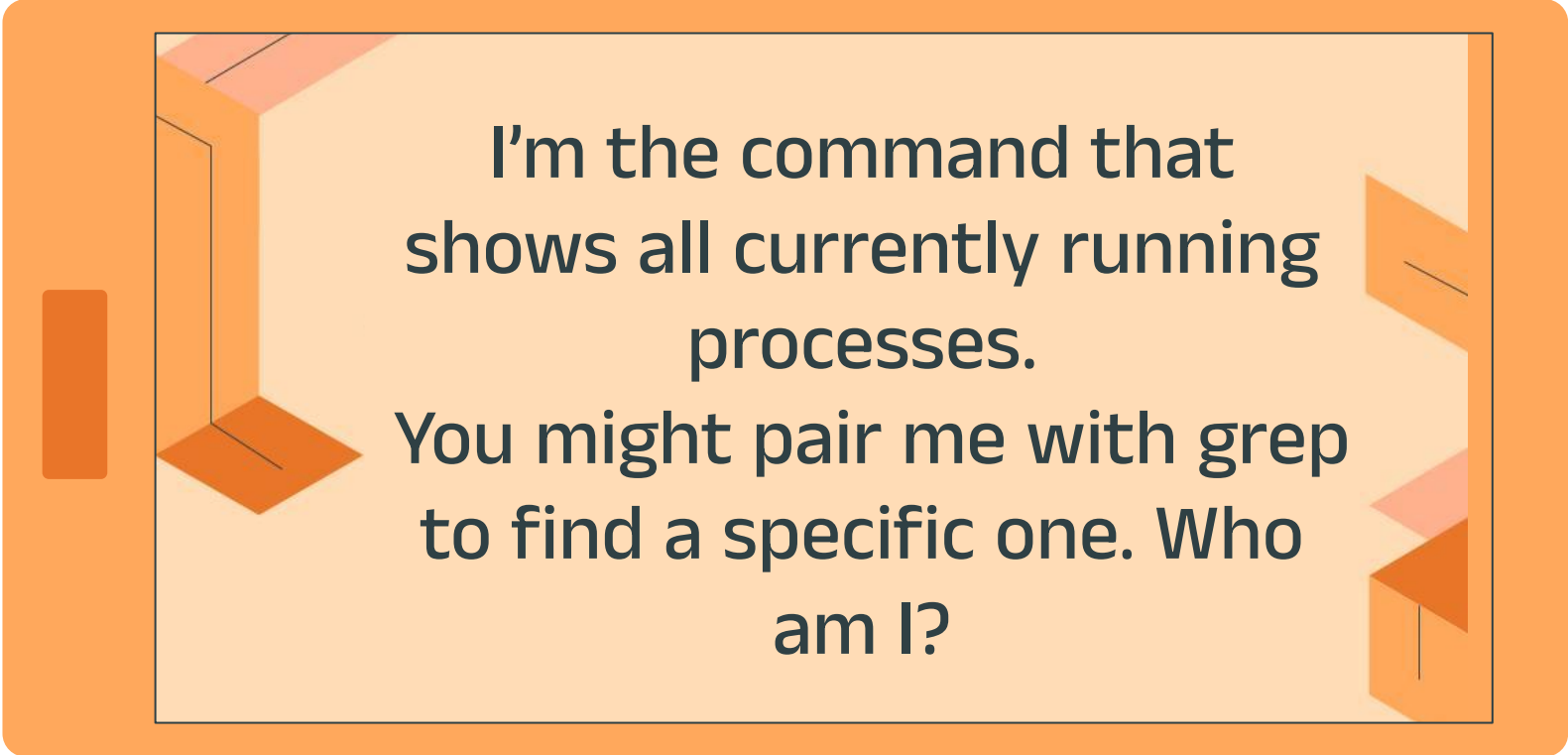




I was terminated, but my  
entry remains in the  
process table because my  
parent didn't clean me up.  
I linger as a ghost. Who am  
I?

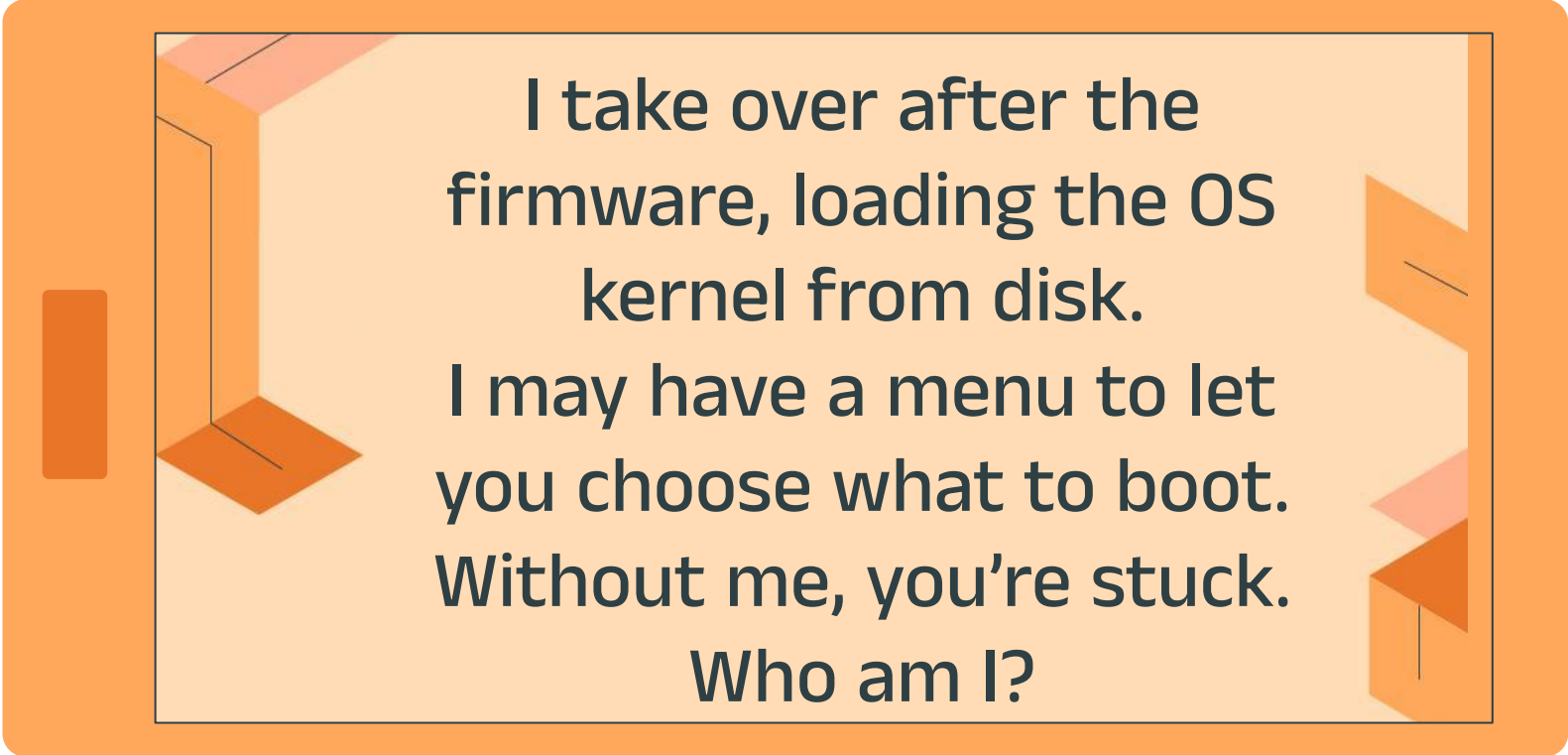


I start running in the  
terminal and block your  
shell until I finish.  
I demand your full  
attention. Who am I?



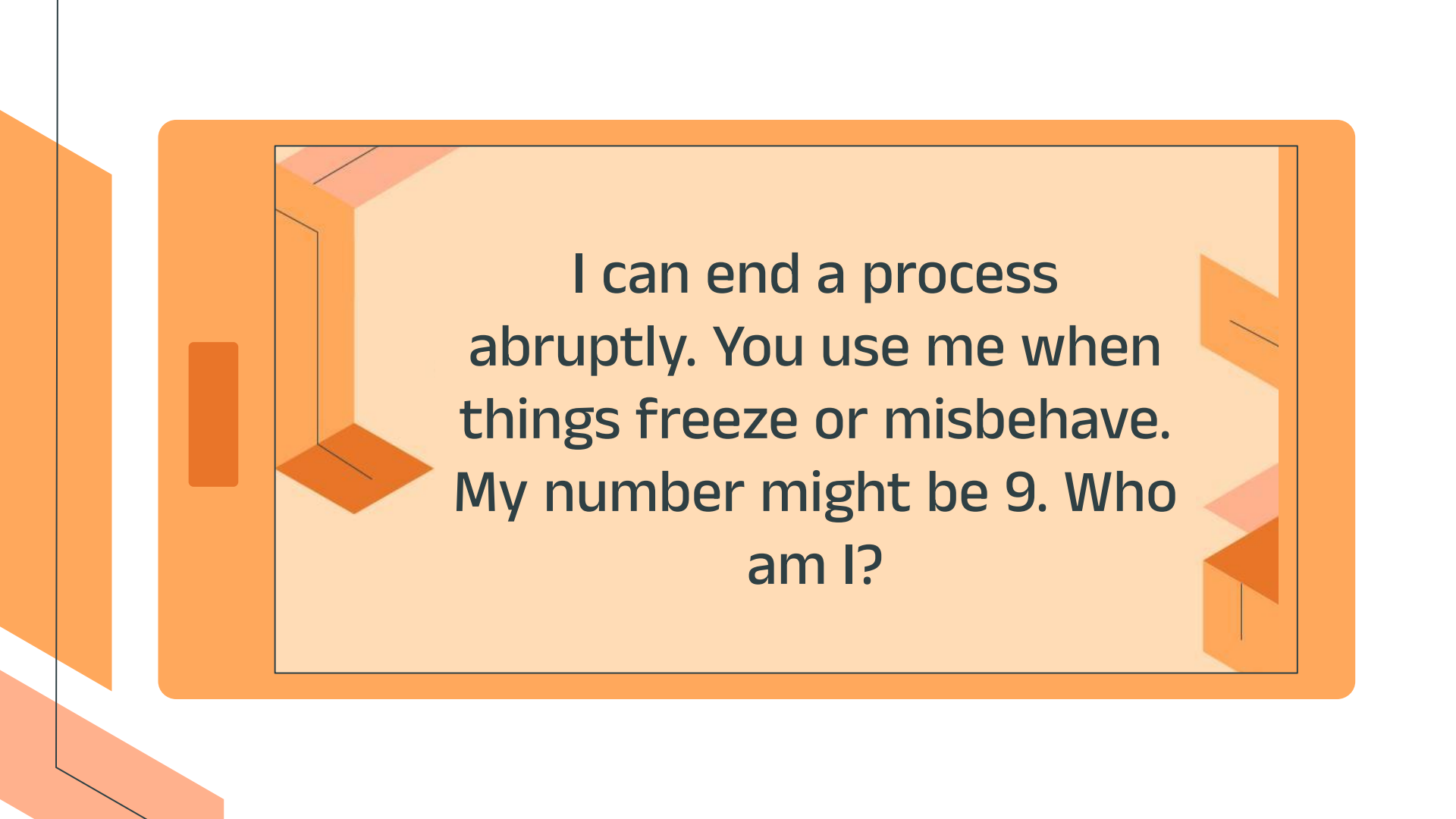
I'm the command that  
shows all currently running  
processes.

You might pair me with grep  
to find a specific one. Who  
am I?

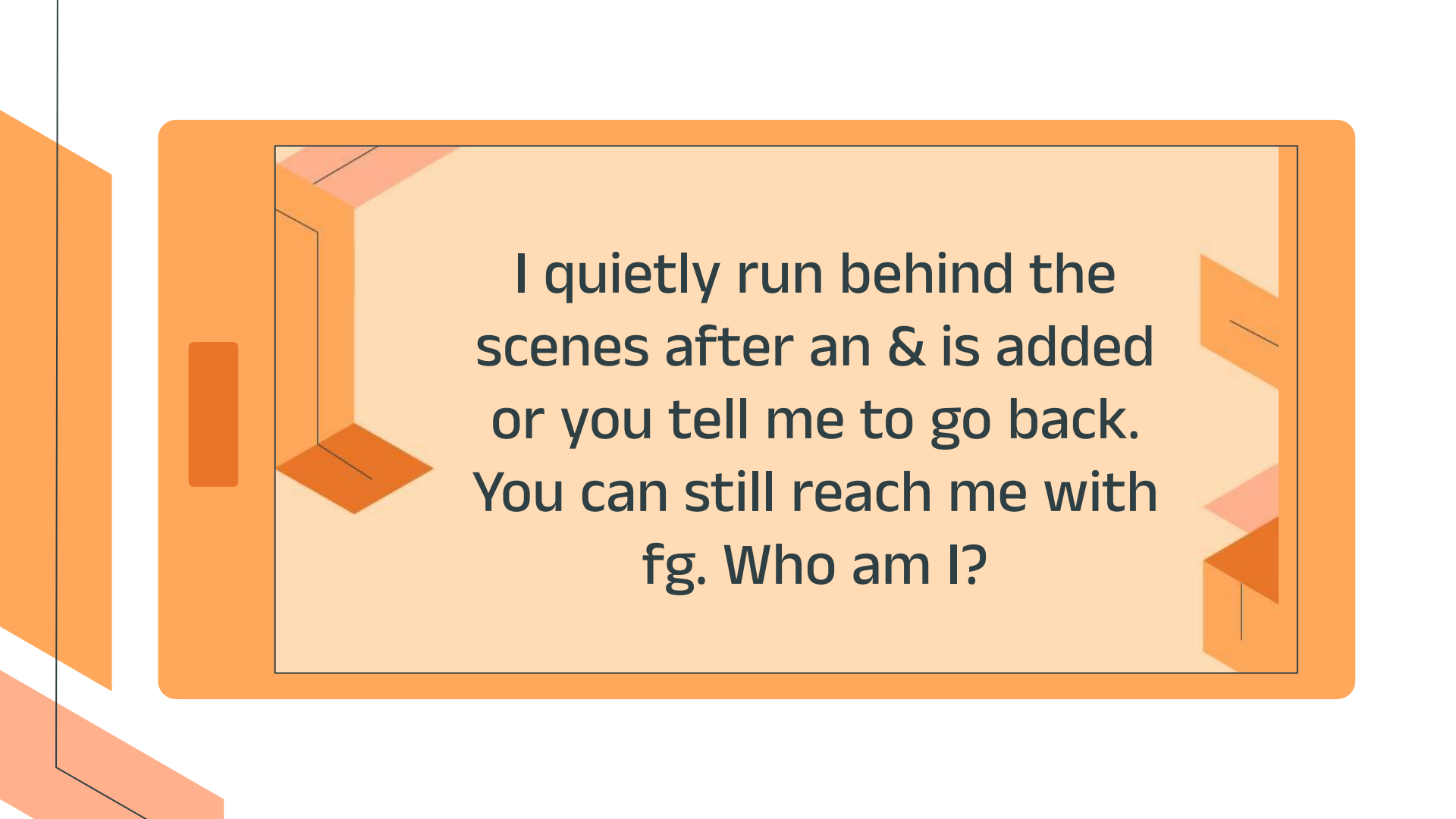


I take over after the  
firmware, loading the OS  
kernel from disk.

I may have a menu to let  
you choose what to boot.  
Without me, you're stuck.  
Who am I?



I can end a process  
abruptly. You use me when  
things freeze or misbehave.  
My number might be 9. Who  
am I?

The background is a solid orange color. On the left side, there are several overlapping orange geometric shapes, including a large rectangle and a smaller one. In the center, there is a large, light-orange rounded rectangle with a thin black border. Inside this rectangle, the text is displayed. On the right side of the central rectangle, there are more orange geometric shapes, including a large rectangle and a smaller one. The text is centered within the central rectangle.

I quietly run behind the  
scenes after an & is added  
or you tell me to go back.  
You can still reach me with  
fg. Who am I?

# HANDS ON



# Hands on #2

1. Open a terminal and run `xlogo`.
2. Suspend (stop) the `xlogo` process.
3. Run another `xlogo` process in the background.
4. Search for the two Xlogo processes.
5. List the current shell jobs.
6. Bring the suspended job back to the foreground.
7. Open a new terminal and kill every `xlogo` process.



## → Solution:

1. Open a terminal and run `xlogo`.

```
xlogo
```

2. Suspend (stop) the `xlogo` process.

```
cntrl+Z , kill -STOP PID
```

3. Run another `xlogo` process in the background.

```
xlogo &
```

4. Search for the two Xlogo processes.

```
ps -C xlogo
```

5. List the current shell jobs.

```
jobs
```

6. Bring the suspended job back to the foreground.

```
fg&JID
```

7. Open a new terminal and kill every `xlogo` process.

```
killall xlogo
```

# Table of contents

01

## Boot Process

sequence of steps  
your computer goes  
through when you  
turn it on

02

## Processes

instances of a running  
program

03

## Systemd

init system (the first  
real program started  
by the kernel).



03

---

# Systemd

# Systemd

- systemd is the parent of all processes on modern Linux systems, and it is responsible for bringing the Linux host up to a state in which productive work can be done.
- systemd is the primary init system on modern Linux distributions, replacing older systems like SysVinit.
- The init system is the very first process to start on your system, and schedules all other processes on your system.

# Systemd

- The PID (Process ID) of systemd process is always 1. It persists till the computer halts.
- On some systems, you might find `/sbin/init`, which is typically a symlink to `/lib/systemd/systemd`.
- systemd provides a powerful and flexible way to manage system services.

# Systemd

- Services in Linux are processes that run in the background to perform specific tasks.
- These can be anything from web servers, database servers, or even system-related tasks like logging and monitoring.
- A great example of such a service is cron, which is used for job scheduling. You can start, stop, restart, and check the status of any service using systemd commands.

# Basic systemctl Commands





# Commands

**sudo systemctl status <service>**

- Checking the Status of a Service

**sudo systemctl start <service>**

- Starting a service.

**sudo systemctl stop <service>**

- Stopping a service

# Commands

**sudo systemctl restart <service>**

- Restarting a service

**sudo systemctl enable <service>**

- Enabling a service at boot

**sudo systemctl disable <service>**

- Disabling a service at boot

# Table of contents

01

## Boot Process

sequence of steps  
your computer goes  
through when you  
turn it on

02

## Processes

instance of a running  
program

03

## Systemd

init system (the first  
real program started  
by the kernel).



# Thanks!

**Do you have any questions?**

# Resources



- <https://github.com/Open-Source-Community/BeRoot/tree/main/Boot%20Process>
- <https://github.com/Open-Source-Community/BeRoot/tree/main/Boot%20Process>

