# Linux Bootloaders using QEMU for RISCV

Internal Group P-4

# Agenda

- What exactly a Bootloader is?
- What and Why QEMU?
- RISC-V Bootflow
- U-boot
- EDK II
- Coreboot

# Bootloader

- A bootloader, also known as a bootstrap loader, is a crucial piece of software responsible for initiating the startup process of the computer.

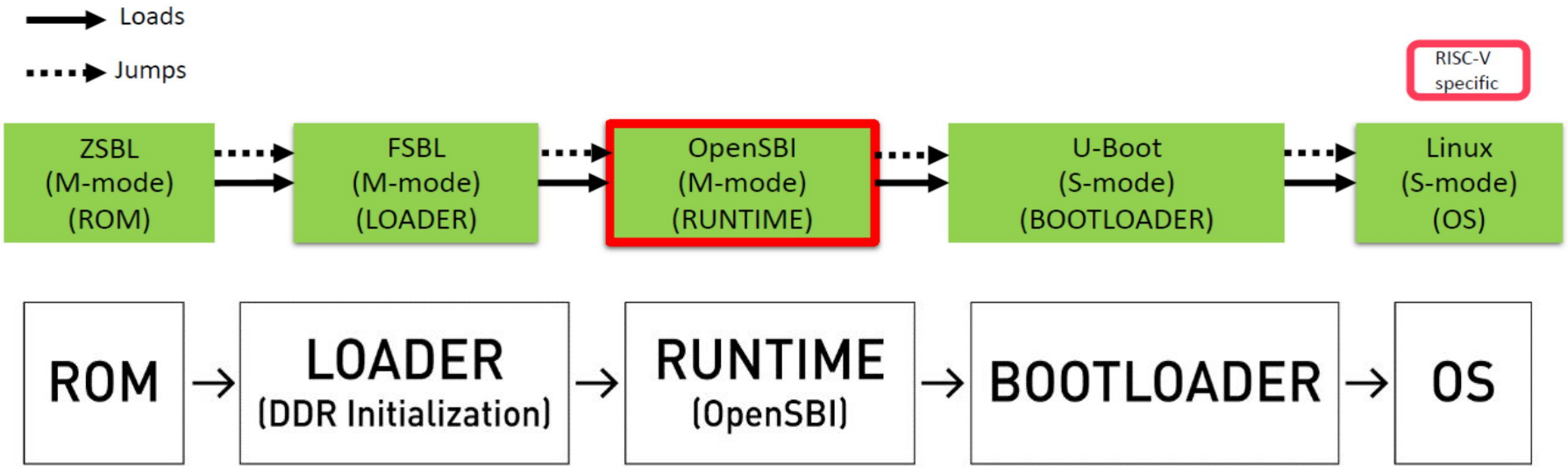
- It is a small program that places the operating system (OS) of a computer into memory

# QEMU: Quick Emulator

- QEMU is a free and open-source software that allows you to emulate and virtualize computer systems.

- QEMU can emulate a wide variety of computer architectures, including x86, ARM, PowerPC, RISC-V, and more. This means that you can use QEMU to run operating systems that were designed for a different type of computer.

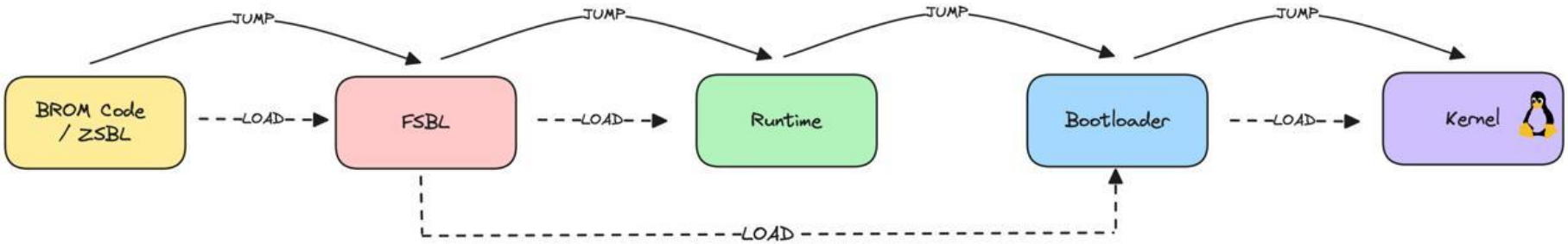
# RISC-V Boot Flow



# U-Boot

- U-Boot, also known as Das U-Boot (German for "The Submarine") is a highly popular open-source bootloader used in a wide range of embedded devices. It performs several crucial tasks during the boot process, ensuring the device starts up and can run its intended operating system.
- Some key features are:
  - Hardware Initialization.
  - Device Tree Support
  - Boot Source Selection
  - Kernel Loading
  - Command-Line Interface
  - Open Source

# U-Boot




- Flashed in on-chip Mask ROM
- Runs from embedded SRAM
- Performs the bare-minimum Power-up and Clock setup
- It's task is to load another appropriate bootloader into the SRAM

- Runs in the embedded SRAM
- Main task is DDR init and load the next stage boot-loader into the DDR memory
- Eg : u-boot spl
  Coreboot

- Runs from embedded SRAM or more typically DDR memory
- SoC security setup
- Provides low-level abstraction for kernel such as power control and privileged register access
- Persists in memory

Eg : UEFI, ATF. OpenSBI

- Runs from DDR
- Filesystem support
- Network booting
- Boot configuration
- Eg : u-boot, grub

jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project

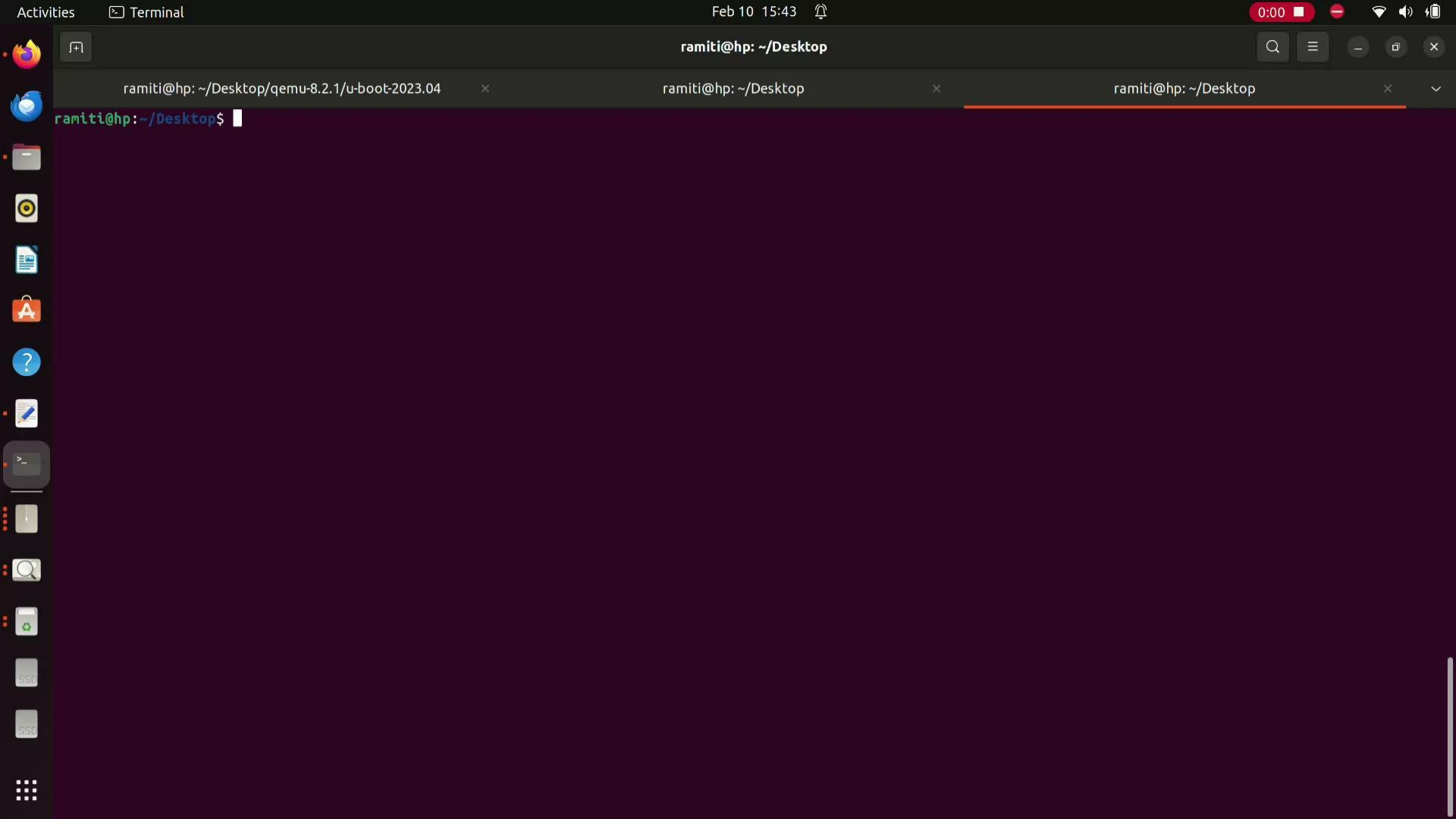
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project

```
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA:~/cdac_project$ qemu-system-riscv64   -bios u-boot-spl.bin   -nographic   -machine virt   -smp 6   -m 6G   -
device loader,file=u-boot.itb,addr=0x80200000   -object rng-random,filename=/dev/urandom,id=rng0   -device virtio-rng-device,rng=rng0   -device virtio-blk-dev
ice,drive=hd0   -drive file=Fedora-Developer-38-20230825.n.0-sda.raw,format=raw,id=hd0   -device virtio-net-device,netdev=usernet   -netdev user,id=usernet,ho
stfwd=tcp::10000-:22
```

jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project

jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project

```
[  OK  ] Reached target remote-fs.target - Remote File Systems.
         Starting systemd-user-sess…vice - Permit User Sessions...
         Starting virtqemud.service…0m - Virtualization qemu daemon...
[  OK  ] Started sshd.service - OpenSSH server daemon.
[  OK  ] Finished systemd-user-sess…ervice - Permit User Sessions.
         Starting plymouth-quit-wai… until boot process finishes up...
         Starting plymouth-quit.ser… Terminate Plymouth Boot Screen...

Welcome to the Fedora/RISC-V disk image
https://fedoraproject.org/wiki/Architectures/RISC-V

Build date: Fri Aug 25 07:41:12 UTC 2023

Kernel 6.4.12-200.0.riscv64.fc38.riscv64 on an riscv64 (ttyS0)

The root password is 'fedora_rocks!'.
root password logins are disabled in SSH starting Fedora 31.
User 'riscv' with password 'fedora_rocks!' in 'wheel' and 'mock' groups
is provided.

To install new packages use 'dnf install ...'

To upgrade disk image use 'dnf upgrade --best'

If DNS isn't working, try editing '/etc/yum.repos.d/fedora-riscv.repo'.

For updates and latest information read:
https://fedoraproject.org/wiki/Architectures/RISC-V

Fedora/RISC-V
-------------
Koji:               http://fedora.riscv.rocks/koji/
SCM:                http://fedora.riscv.rocks:3000/
Distribution rep.:  http://fedora.riscv.rocks/repos-dist/
Koji internal rep.: http://fedora.riscv.rocks/repos/
fedora-riscv login: riscv
Password:
Last login: Wed Feb 14 07:29:02 on ttyS0
[riscv@fedora-riscv ~]$
```


Activities
Terminal
Feb 10 15:43
0:00
ramiti@hp: ~/Desktop
ramiti@hp: ~/Desktop/qemu-8.2.1/u-boot-2023.04
ramiti@hp: ~/Desktop
ramiti@hp: ~/Desktop
ramiti@hp:~/Desktop$

# Difference between BIOS and UEFI

BIOS and UEFI are both firmware interfaces responsible for starting up your computer and initializing the operating system. However, they differ in several key aspects:

**Functionality:**

- **BIOS (Basic Input/Output System):** It has been the traditional standard for decades, but is essentially a basic program written in 16-bit code. BIOS initializes hardware and then hands over control to the operating system bootloader.
- **UEFI (Unified Extensible Firmware Interface):** It's the modern successor to BIOS, written in 32-bit or 64-bit code, offering more powerful features and flexibility. UEFI acts as a mini operating system itself, allowing for advanced tasks like network access and secure boot.

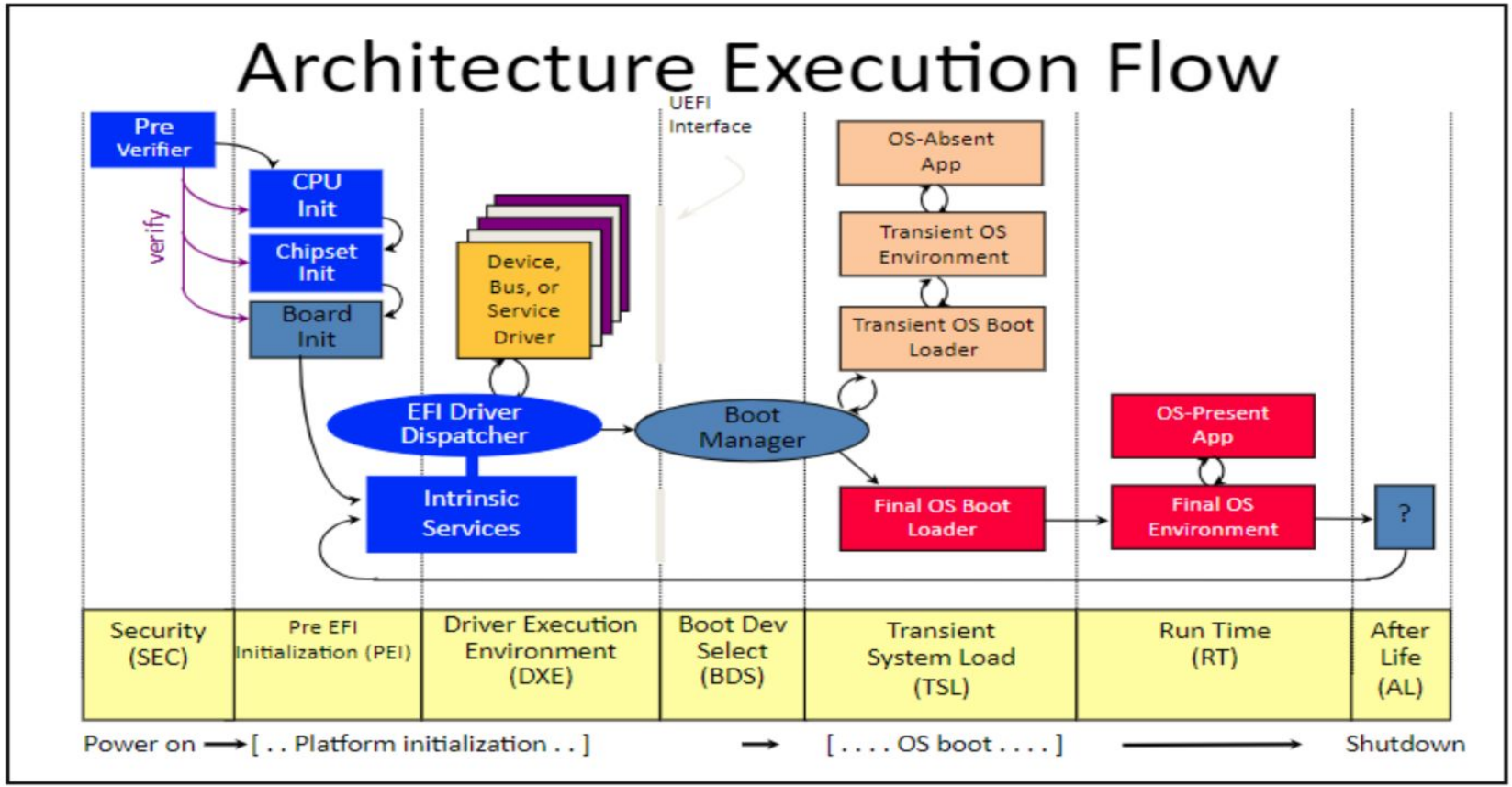
**Key Differences:**

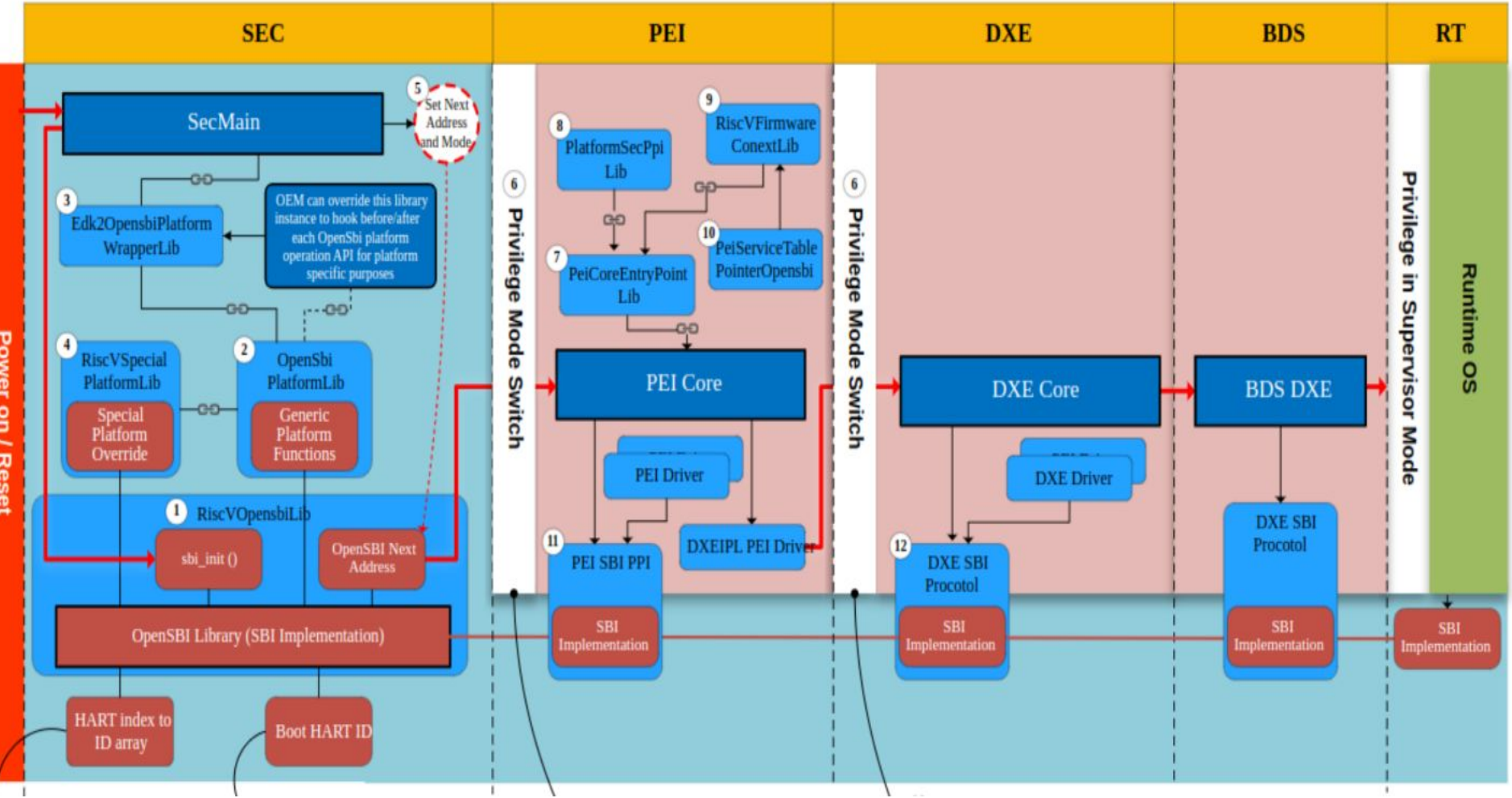
- **Boot Speed:** UEFI generally offers faster boot times due to its more efficient architecture and optimizations.
- **Storage Capacity:** BIOS is limited to drives under 2.2TB due to its use of the Master Boot Record (MBR). UEFI uses the GUID Partition Table (GPT), supporting massive drives up to 9 zettabytes.
- **Security:** UEFI offers enhanced security features like Secure Boot, preventing unauthorized operating systems from booting. This can hinder dual-booting capabilities if not configured correctly.
- **User Interface:** BIOS features a text-based interface controlled by keyboard commands. UEFI boasts a graphical user interface (GUI) with mouse support, making it more user-friendly.
- **Extensibility:** UEFI is designed to be modular and extensible, allowing for additional drivers and features to be loaded during boot.

# EDK II

- EDK II, short for Extensible Firmware Interface (EFI) Development Kit II, is a powerful and feature-rich open-source environment for developing platform firmware based on the UEFI and PI specifications. It's widely used in various scenarios, including:
    - Embedded systems: Routers, switches, industrial controllers, single-board computers (e.g., Raspberry Pi)
    - Servers: High-performance computing clusters, cloud infrastructure
    - Personal computers: Some manufacturers utilize EDK II for their UEFI firmware

# UEFI for X86


Architecture Execution Flow
Pre Verifier
verify
CPU Init
Chipset Init
Board Init
Device, Bus, or Service Driver
EFI Driver Dispatcher
Intrinsic Services
UEFI Interface
Boot Manager
OS-Absent App
Transient OS Environment
Transient OS Boot Loader
Final OS Boot Loader
OS-Present App
Final OS Environment
?
Security (SEC)
Pre EFI Initialization (PEI)
Driver Execution Environment (DXE)
Boot Dev Select (BDS)
Transient System Load (TSL)
Run Time (RT)
After Life (AL)
Power on
[ . . Platform initialization . . ]
[ . . . . OS boot . . . . ]
Shutdown


SEC
PEI
DXE
BDS
RT
Power on / Reset
SecMain
5
Set Next Address and Mode
3
Edk2OpensbiPlatform WrapperLib
OEM can override this library instance to hook before/after each OpenSbi platform operation API for platform specific purposes
4
RiscVSpecial PlatformLib
Special Platform Override
2
OpenSbi PlatformLib
Generic Platform Functions
1
RiscVOpensbiLib
sbi_init ()
OpenSBI Next Address
OpenSBI Library (SBI Implementation)
HART index to ID array
Boot HART ID
6
Privilege Mode Switch
8
PlatformSecPpi Lib
9
RiscVFirmware ConextLib
7
PeiCoreEntryPoint Lib
10
PeiServiceTable PointerOpensbi
PEI Core
PEI Driver
11
PEI SBI PPI
SBI Implementation
DXEIPL PEI Driver
6
Privilege Mode Switch
DXE Core
DXE Driver
12
DXE SBI Procotol
SBI Implementation
BDS DXE
DXE SBI Procotol
SBI Implementation
Privilege in Supervisor Mode
Runtime OS
SBI Implementation

# UEFI execution flow…

- Security (SEC)
- Handling all platform restart events
- Creating a temporary memory store
- Serving as the root of trust in the system
- Passing handoff information to the PEI Foundation

- Pre-EFI Initialization (PEI)
- Initializing some permanent memory complement.
- Describing the memory in Hand-Off Blocks (HOBs)
- Describing the firmware volume locations in HOBs
- Passing control into the Driver Execution Environment (DXE)phase

# UEFI execution flow…

## Drive Execution Environment (DXE)

- The state of the system at the end of the PEI phase is passed to the DXE phase through a list of position-independent data structures called Hand-Off Blocks (HOBs).

• DXE Foundation

• DXE Dispatcher

• A set of ‘DXE Drivers’

• **Boot Device Selection (BDS)**

• Initializing console devices

• Loading device drivers

• Attempting to load and execute boot selections

• Transient System Load (TLS) and Runtime (RT)

# BDS continue….

The Transient System Load (TLS) primarily the OS vendor provided boot loader, along with the Runtime Services (RT)phases may access persistent UEFI drivers and applications.Drivers in this category include PCI Option ROMs.

- After Life (AL)

- The After Life (AL) phase consists of persistent UEFI drivers used for storing the state of the system during the OS orderly shutdown, sleep,hibernate or restart processes.

# EDK II work progress


## Obtaining source code

1. Create a new folder (directory) on your local development machine for use as your workspace. This example uses `/work/git/tianocore`, modify as appropriate for your needs.

```
$ export WORKSPACE=/work/git/tianocore
$ mkdir -p $WORKSPACE
$ cd $WORKSPACE
```

export WORKSPACE=/home/riscv

2. Into that folder, clone:

   i. edk2
   ii. edk2-platforms
   iii. edk2-non-osi (if building platforms that need it)

```
$ git clone https://github.com/tianocore/edk2.git
$ git submodule update --init
...
$ git clone https://github.com/tianocore/edk2-platforms.git
$ git submodule update --init
...
```

3. Set up a **PACKAGES_PATH** to point to the locations of these three repositories:

```
$ export PACKAGES_PATH=$PWD/edk2:$PWD/edk2-platforms:$PWD/edk2-non-osi
```

export PACKAGES_PATH=/home/riscv/edk2:/home/riscv/edk2-platforms:/home/riscv/edk2-non-osi

## Manual building

1. Set up the build environment (this will modify your environment variables)

   `$ . edk2/edksetup.sh`

   (This step *depends* on **WORKSPACE** being set as per above.)

2. Build BaseTools

   `make -C edk2/BaseTools`

## Build options

There are a number of options that can (or must) be specified at the point of building. Their default values are set in `edk2/Conf/target.txt` . If we are working only on a single platform, it makes sense to just update this file.

| target.txt option | command line | Description |
|---|---|---|
| ACTIVE_PLATFORM | `-p` | Description file (.dsc) of platform. |
| TARGET | `-b` | One of DEBUG, RELEASE or NOOPT. |
| TARGET_ARCH | `-a` | Architecture to build for. |
| TOOL_CHAIN_TAG | `-t` | Toolchain profile to use for building. |

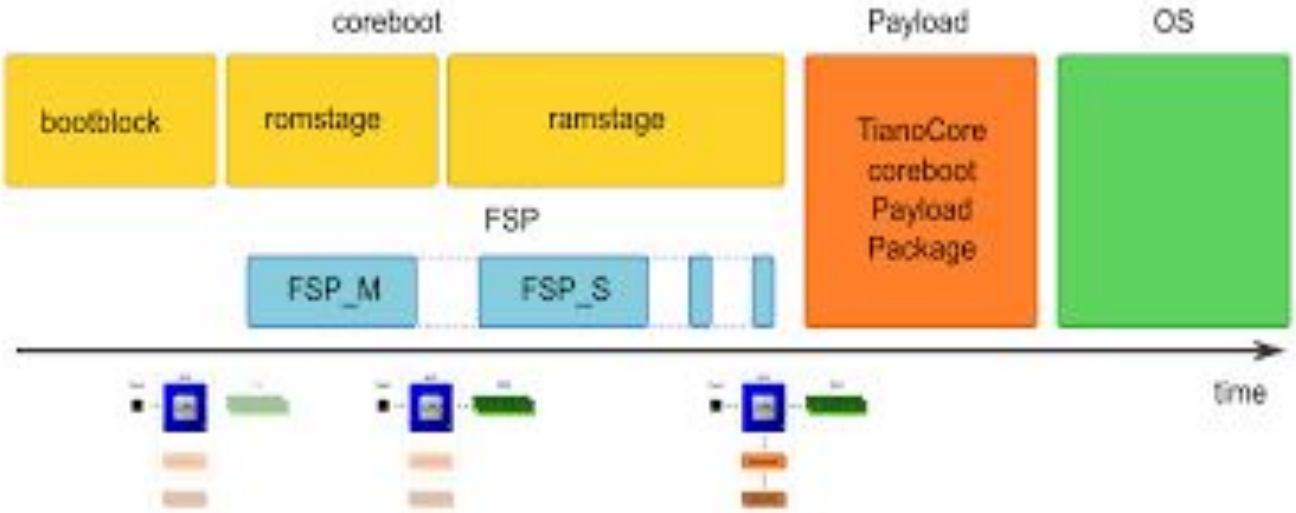
build -n 4 -a RISCV64 -p home/riscv/edk2-platforms/Platform/RISC-V/PlatformPkg/RiscVPlatformPkg.dsc

```
CONF_PATH        = /home/riscv/edk2/Conf
PYTHON_COMMAND   = /usr/bin/python3


Architecture(s)  = RISCV64
Build target     = DEBUG
Toolchain        = VS2015x86

Active Platform          = /home/riscv/edk2-platforms/Platform/RISC-V/PlatformPkg/RiscVPlatformPkg.dsc

Processing meta-data ..

build.py...
/home/riscv/edk2-platforms/Platform/RISC-V/PlatformPkg/RiscVPlatformPkg.dsc(...): error 4000: Instance of library class [RegisterFilterLib] is not found
        in [/home/riscv/edk2/MdePkg/Library/BaseIoLibIntrinsic/BaseIoLibIntrinsic.inf] [RISCV64]
        consumed by module [/home/riscv/edk2-platforms/Platform/RISC-V/PlatformPkg/Universal/Sec/SecMain.inf]


- Failed -
Build end time: 08:03:03, Feb.14 2024
Build total time: 00:00:02

[riscv@fedora-riscv ~]$
[riscv@fedora-riscv ~]$
```

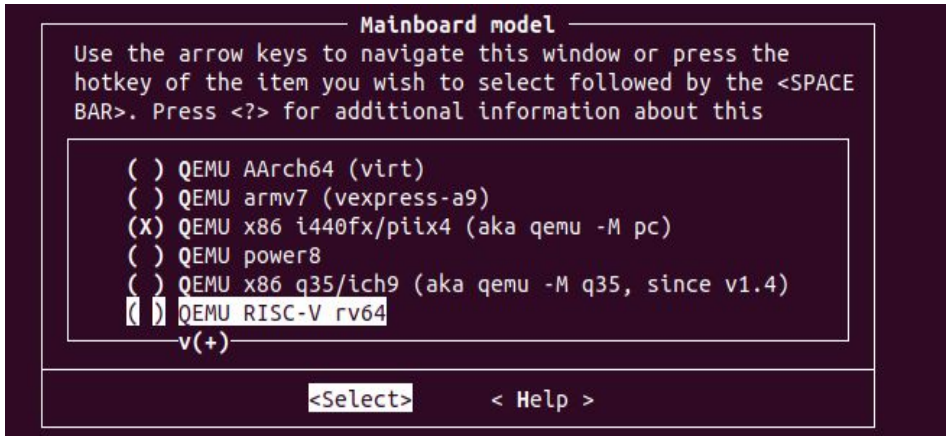
# Coreboot

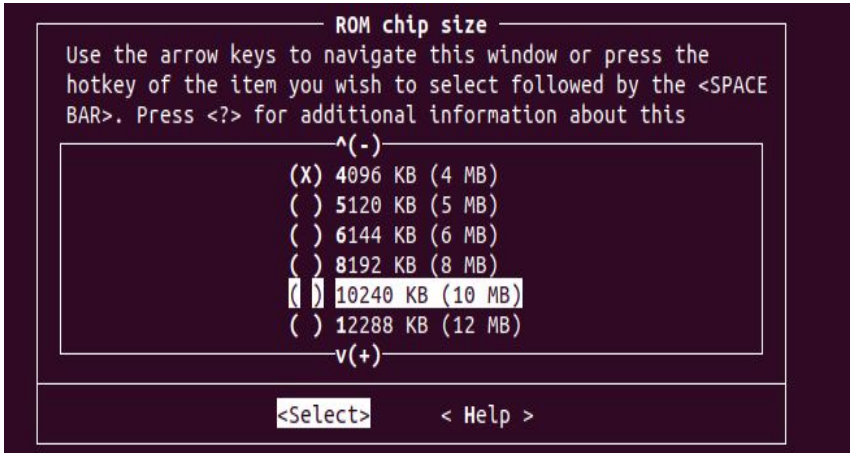
- Booting speed is high.
- Written in Assembly and c language
- Can work with different operating systems.

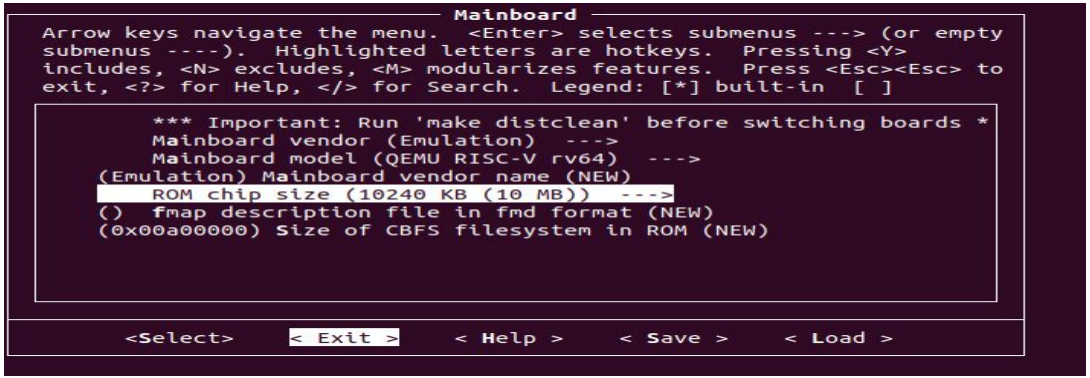
# Coreboot Stages:

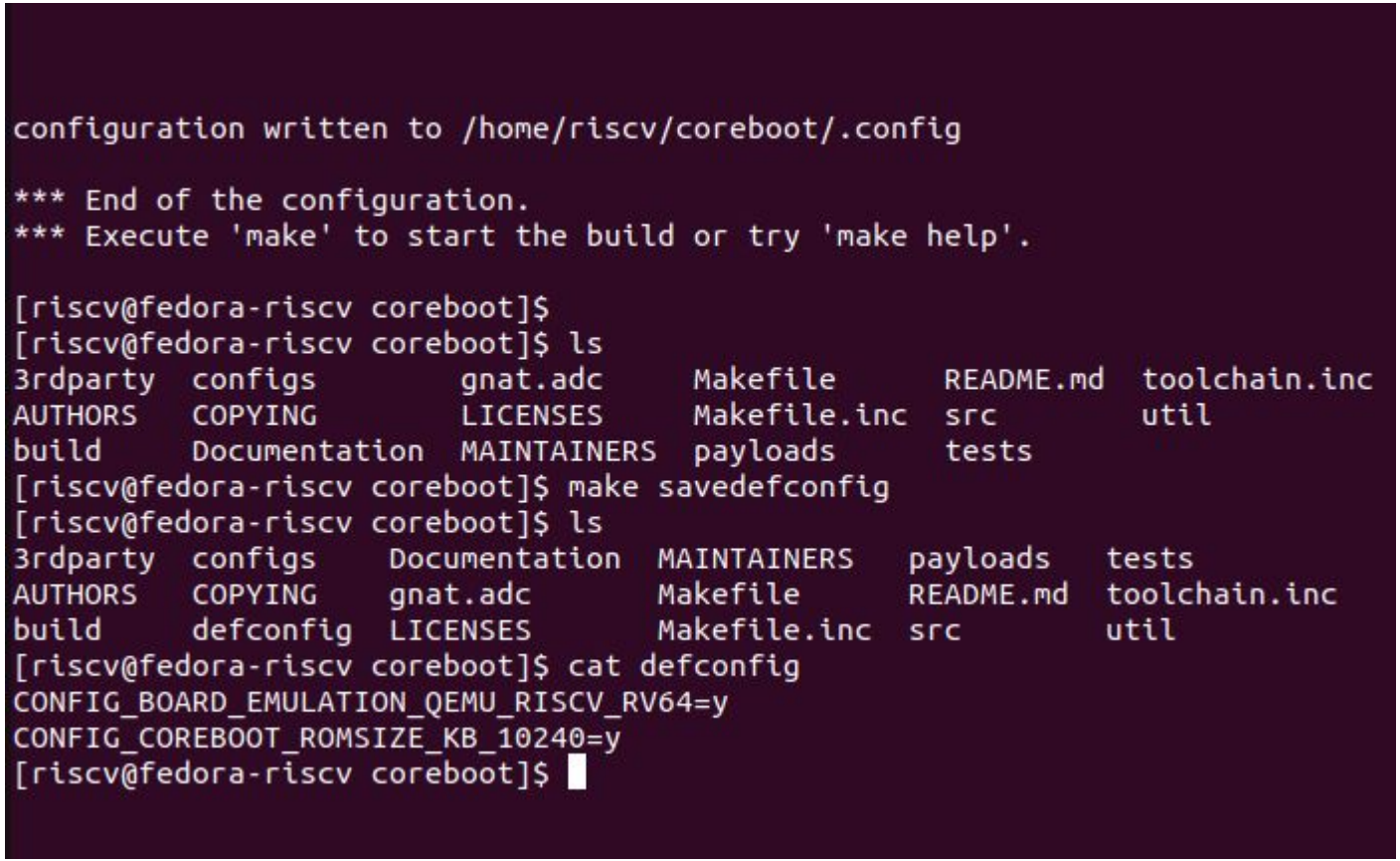


# Coreboot Setup:




configuration written to /home/riscv/coreboot/.config
*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.
[riscv@fedora-riscv coreboot]$
[riscv@fedora-riscv coreboot]$ ls
3rdparty configs gnat.adc Makefile README.md toolchain.inc
AUTHORS COPYING LICENSES Makefile.inc src util
build Documentation MAINTAINERS payloads tests
[riscv@fedora-riscv coreboot]$ make savedefconfig
[riscv@fedora-riscv coreboot]$ ls
3rdparty configs Documentation MAINTAINERS payloads tests
AUTHORS COPYING gnat.adc Makefile README.md toolchain.inc
build defconfig LICENSES Makefile.inc src util
[riscv@fedora-riscv coreboot]$ cat defconfig
CONFIG_BOARD_EMULATION_QEMU_RISCV_RV64=y
CONFIG_COREBOOT_ROMSIZE_KB_10240=y
[riscv@fedora-riscv coreboot]$

# Further Task…

- Till now we have achieved complete booting through u-boot.
- Booting through EDK II and Coreboot is in progress.
- Further tasks are to study and complete booting through EDK II and Coreboot.

# THANK YOU