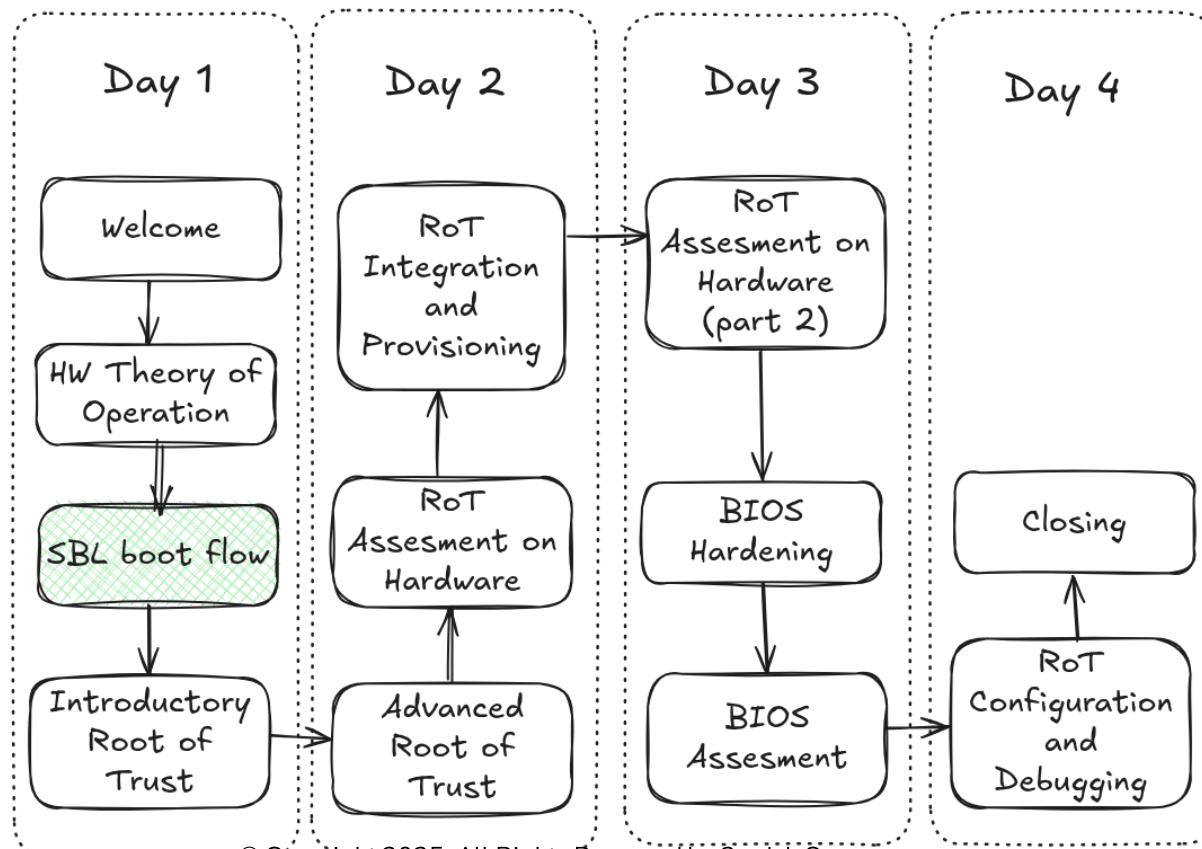


# DS09SBL: Dasharo TrustRoot Training

# Introduction

Introduction to x86 boot flow, UEFI, Root of Trust, and Chain of Trust Technologies

# Where we are in the course



# Goals of the Presentation

In this presentation, we aim to:

- Understand the basics of the typical x86 boot flow based on Hardkernel Odroid-H4 and Slim Bootloader firmware.
- Identify the location of the Root of Trust in the boot process and its role.
- Explore the Chain of Trust mechanisms for the emulated environment and how they works.

# Recommended materials

The following materials would help you understand Slim Bootloader and Intel Root of Trust challenges:

- [Architecture 4001: x86-64 Intel Firmware Attack and Defense, Xeno Kovah](#)
- [Architecture 4021: Introductory UEFI, Piotr Król](#)
- [Building Secure Firmware, Jiewen Yao , Vincent Zimmer](#)
- [Slim Bootloader Documentation](#)

# Origins



Special Publication 800-147



Special Publication 800-155  
(Draft)

NIST Special Publication 800-193

## BIOS Protection Guidelines

### BIOS Integrity Measurement Guidelines (Draft)

- Boot firmware (aka UEFI BIOS, but it could also be coreboot, U-Boot or another) has privileged role within computer system architecture, which makes it an attractive target for attackers.
- Threats vary from DoS (boot firmware corruption), through ransomware (if boot firmware vulnerability helps deploy it), and end with persistent malware (if boot firmware is implanted).
- [NIST SP 800-147](#) provides guidelines for preventing the unauthorized modification of the boot firmware by potentially malicious software running on computer systems.
- [NIST SP 800-155](#) focuses on integrity measurement, reporting, collection, transmission and assessment.
- [NIST SP 800-193](#) discusses RoT and Chain of Trust (CoT) in the context of the *protection, detection and recovery* triad.

## Platform Firmware Resiliency Guidelines

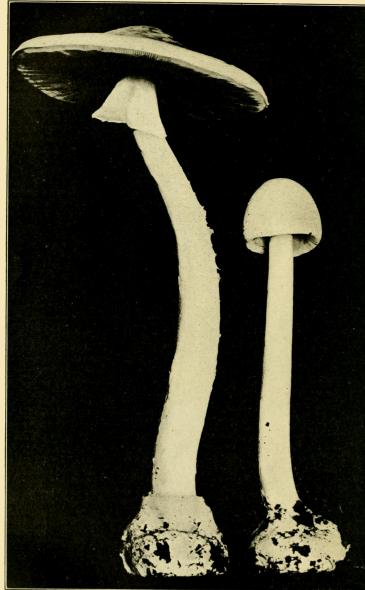
# Other standards and influences



- [TCG Root of Trust Specification](#) and [TCG Glossary](#)
- [Zimmer, Yao: Building Secure Firmware](#)
- [UEFI Specification](#)
- [IETF RFC5914: Trust Anchor Format](#)
- Current state-of-the-art Root of Trust knowledge can be found in [OpenTitan](#) and [OCP Caliptra Project](#).
  - Feb, 2025: [Nuvoton OpenTitan fabrication](#)
- Discussing firmware, Root of Trust and Chain of Trust Technologies is also complex because of multiple standards and nomenclature.

# Remember

Bad taxonomy



can KILL

[Abalg, CC BY-SA 4.0](#), via Wikimedia Commons

© Copyright 2025. All Rights Reserved by 3mdeb Sp. z o.o.

8/91

# Jargon: Root of Trust

- NIST: *Highly reliable hardware, firmware, and software components that perform specific, critical security functions. Because roots of trust are inherently trusted, they must be secure by design. Roots of trust provide a firm foundation from which to build security and trust.*
- TCG: *A component that performs one or more security-specific functions, such as measurement, storage, reporting, verification, and/or update. It is trusted always to behave in the expected manner, because its misbehavior cannot be detected (such as by measurement) under normal operation.*
- Daniel P. Smith: Root of Trust is an entity in the system which is unconditionally trusted.
  - Is just some mechanism defined as "*a process, technique, or system for achieving a result*".
  - The degree of skepticism that the mechanism's result can be subverted imparts its Strength of Mechanism.
  - Trust  $\propto$  Assurance of Strength  $\propto$  1/Skepticism

# Jargon: Chain of Trust

- NIST: *A method for maintaining valid trust boundaries by applying a principle of transitive trust, where each software module in a system boot process is required to measure the next module before transitioning control.*
- TCG: no definition of Chain of Trust
- TCG Transitive Trust: *Also known as "Inductive Trust", in this process a Root of Trust gives a trustworthy description of a second group of functions. Based on this description, an interested entity can determine the trust it is to place in this second group of functions. If the interested entity determines that the trust level of the second group of functions is acceptable, the trust boundary is extended from the Root of Trust to include the second group of functions. In this case, the process can be iterated. The second group of functions can give a trustworthy description of the third group of functions, etc. Transitive trust is used to provide a trustworthy description of platform characteristics, and also to prove that non-migratable keys are non-migratable.*

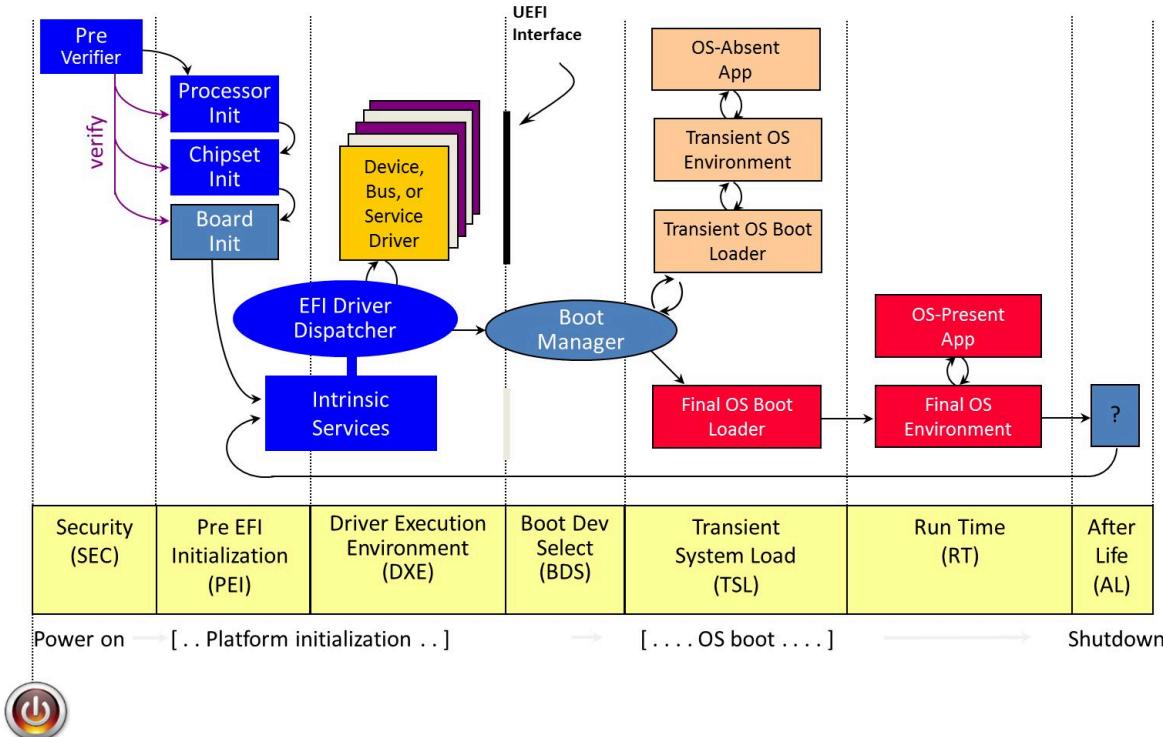
# Jargon: UEFI

- UEFI - *Unified Extensible Firmware Interface* is [specification](#) managed by [UEFI Forum](#), first published in 2007, but its origins are in mid-1990s
- TianoCore - community developing, maintaining and supporting reference implementation of UEFI Specification and related tools. [EDK II](#) is publicly available as source code on GitHub.
- The correctness of the given naming scheme is subject to some debate.
  - All major computer systems supply chain industry players keep calling their bootstrap firmware BIOS.
  - Microsoft uses UEFI term instead.
  - Apple uses EFI terms, except for security documentation where they use UEFI.
- Firmware that provide functionality of old BIOS is typically called Legacy BIOS.
- PI (*Platform Initialization*) and UEFI (*Unified Extensible Firmware Interface*) compatible firmware implementations are sometimes called UEFI BIOS, or just UEFI.
- During the presentation the terms UEFI, BIOS and UEFI BIOS are used interchangeably.

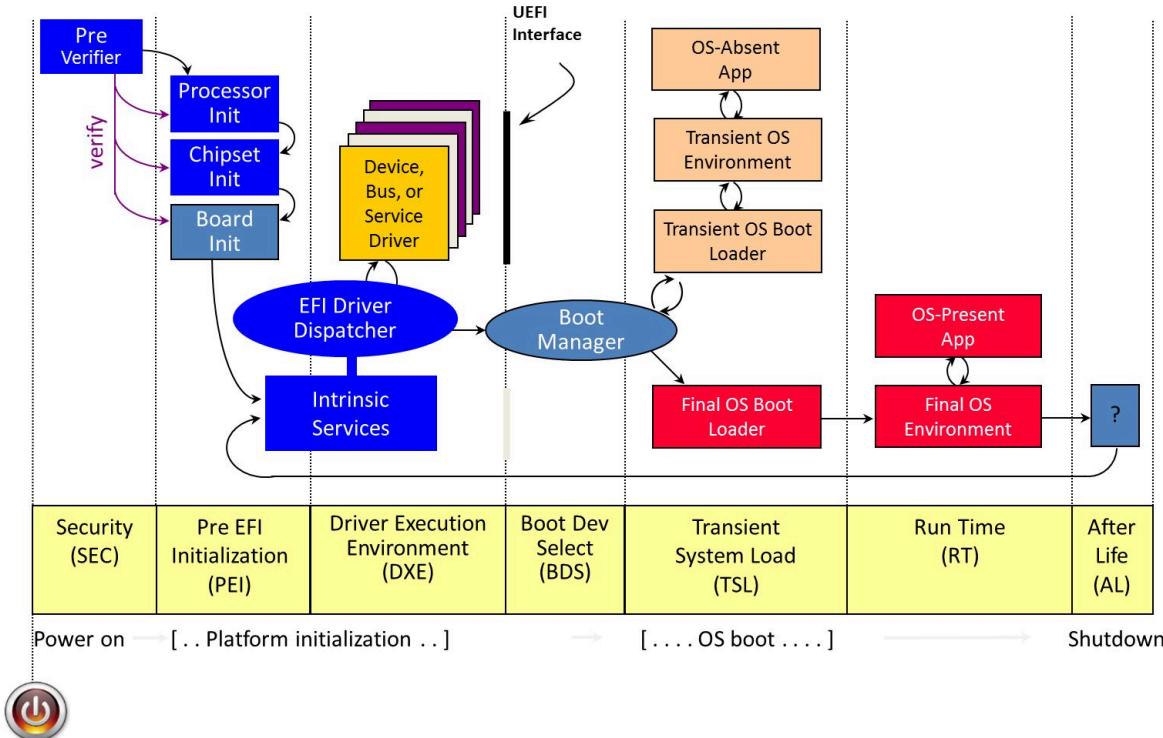
# Jargon

- **Binary blob** - proprietary software only available as a binary executable. Uneditable code which can undermine platform security posture.
- **Trusted Platform Module (TPM)** - secure crypt-processor which attached to a device to establish secure options. Serves as root of trust and supports chain of trust process through measurements, secure storage and attestation.
- **Intel Boot Guard (IBG)** - Intel Root of Trust and early boot firmware Chain of Transitive Trust technology implementation.
- **AMD Platform Secure Boot (PSB)** - AMD Root of Trust and early boot firmware Chain of Transitive Trust technology implementation.
- **System Management Mode** - operating mode of x86 CPU in which all normal execution is suspended. Type of Trusted Execution Environment on x86 which code is typically delivered by UEFI BIOS. Highly privileged mode.

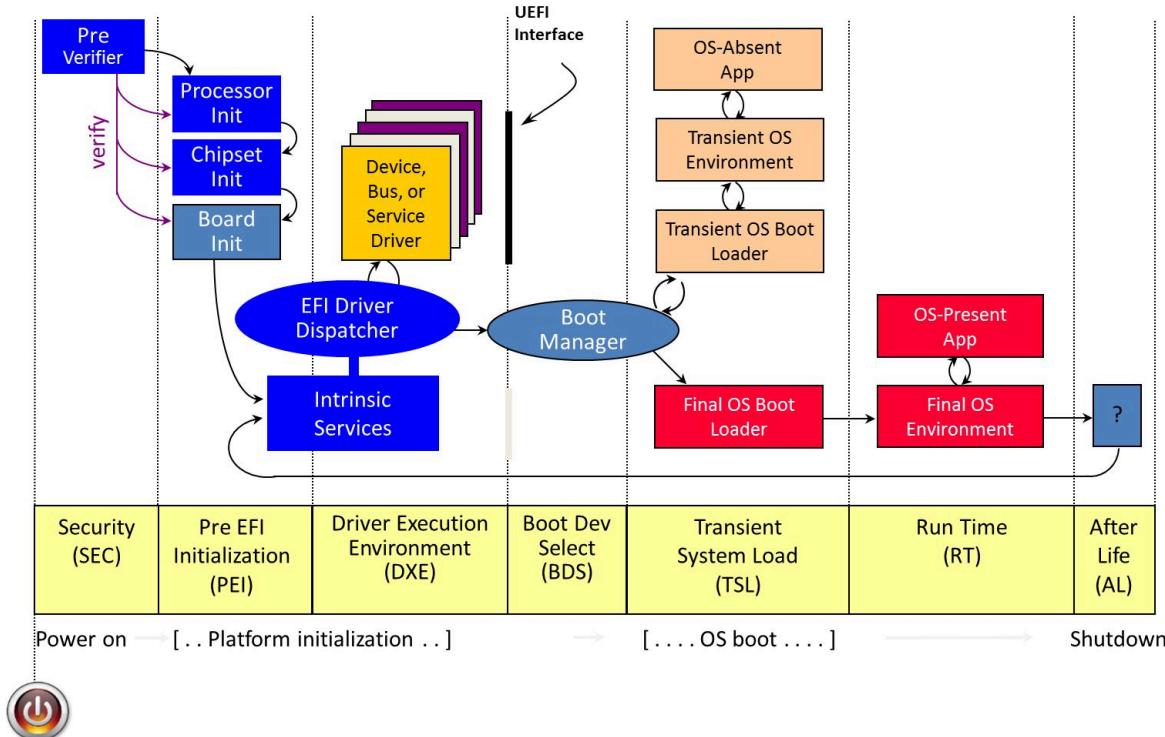
# Platform Initialization (PI) Boot Phases



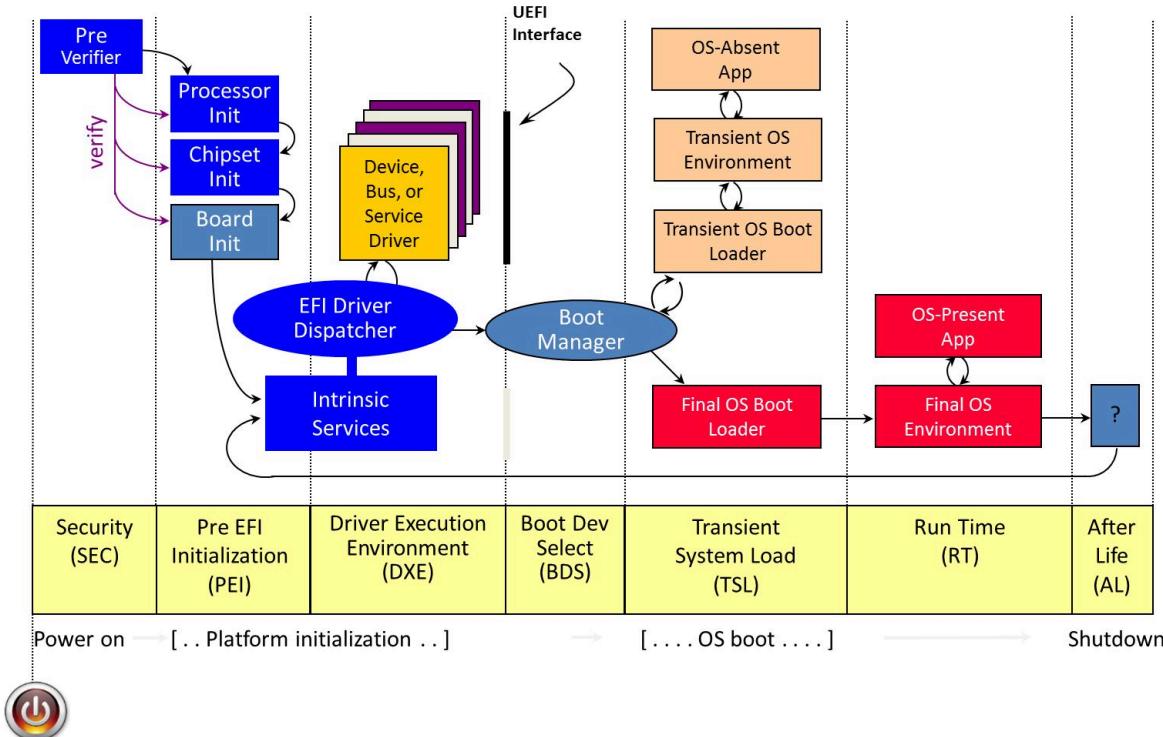
# Platform Initialization (PI) Boot Phases



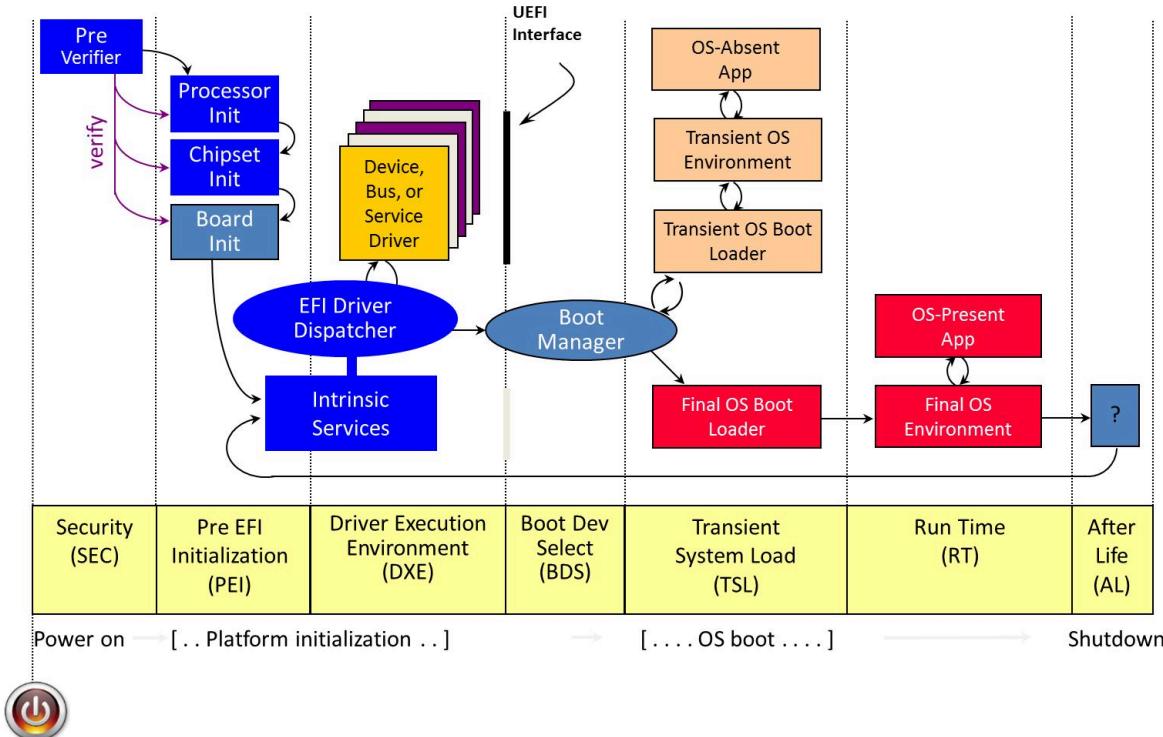
# Platform Initialization (PI) Boot Phases



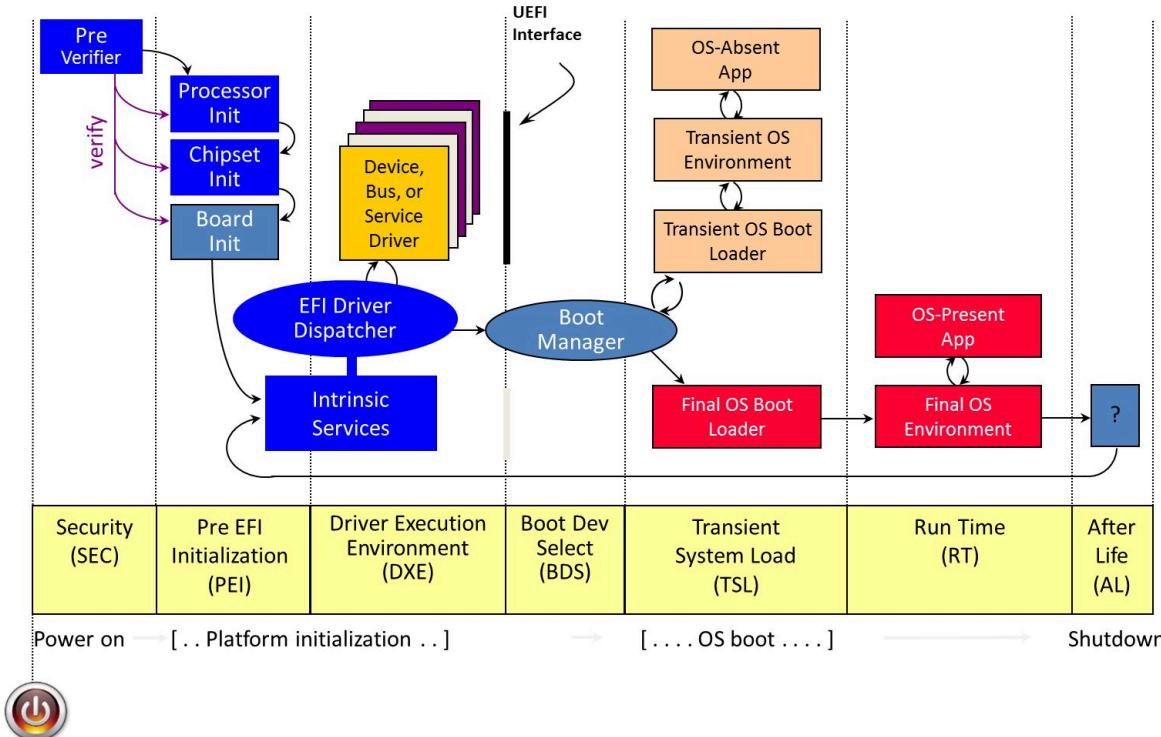
# Platform Initialization (PI) Boot Phases



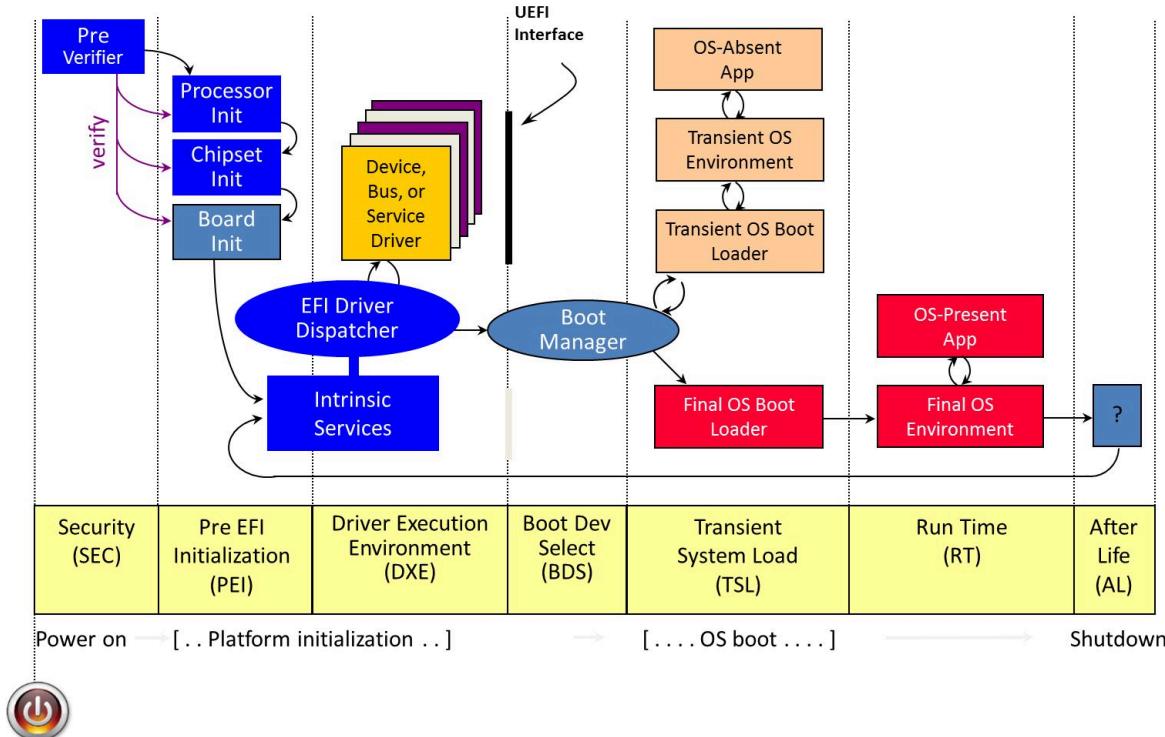
# Platform Initialization (PI) Boot Phases



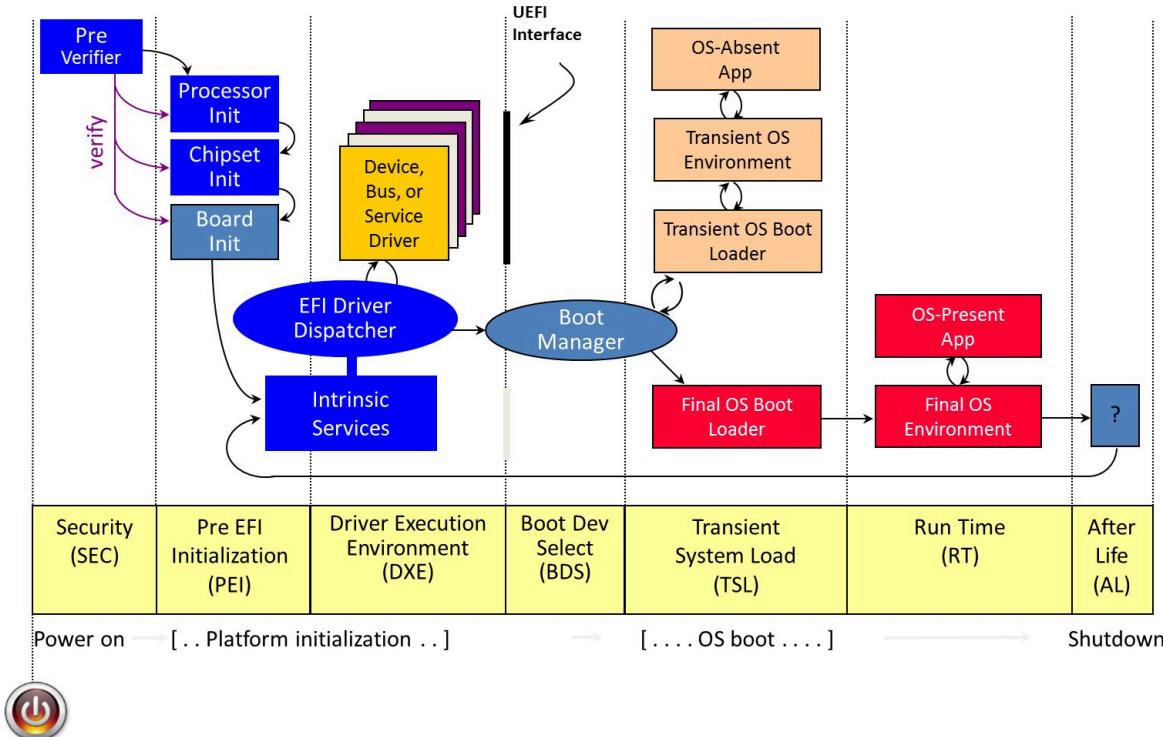
# Platform Initialization (PI) Boot Phases



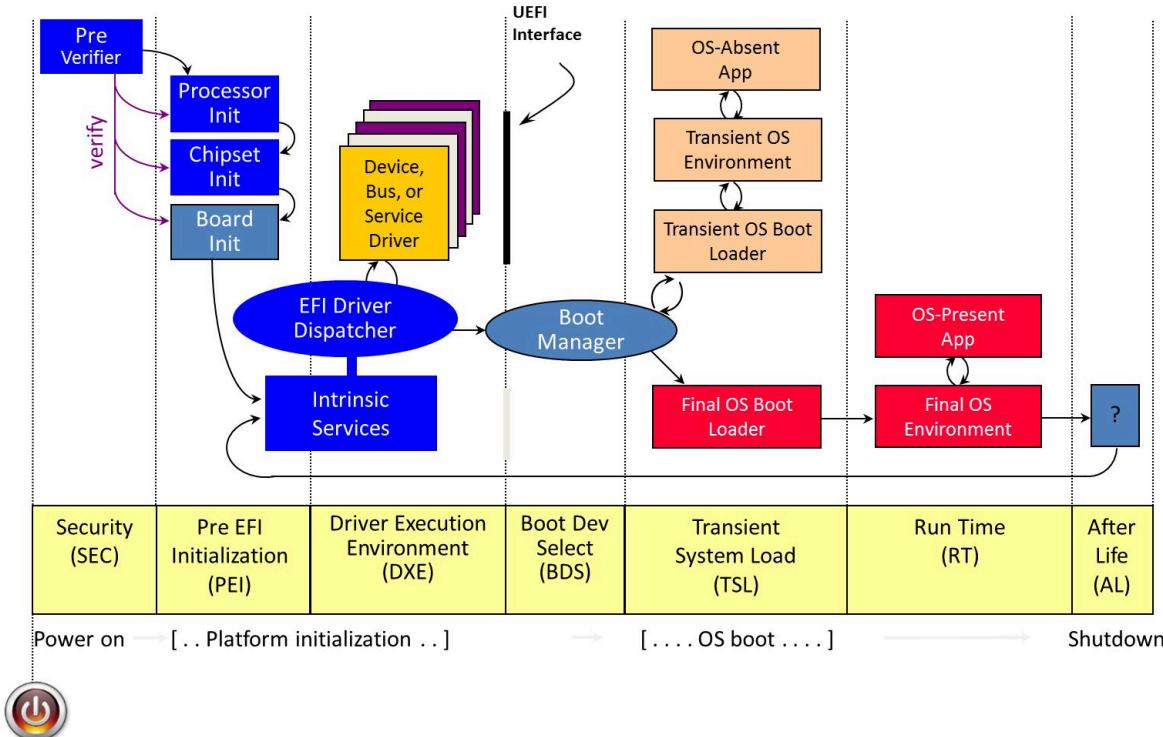
# Platform Initialization (PI) Boot Phases



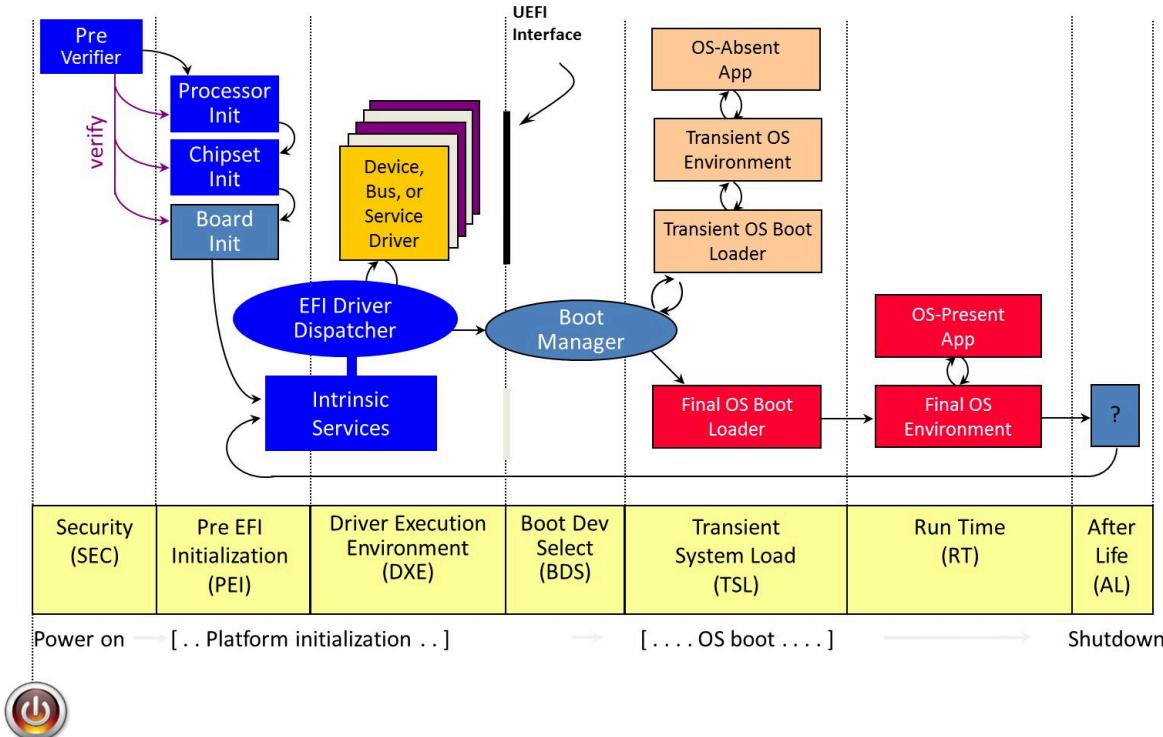
# Platform Initialization (PI) Boot Phases



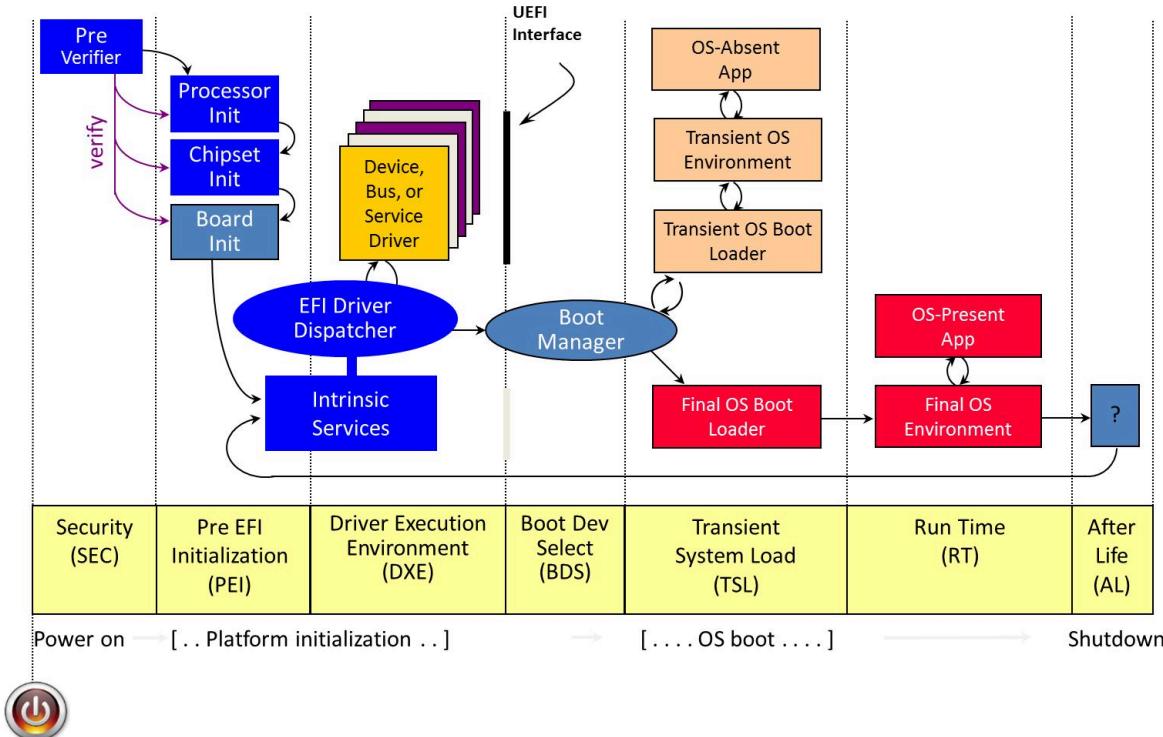
# Platform Initialization (PI) Boot Phases



# Platform Initialization (PI) Boot Phases



# Platform Initialization (PI) Boot Phases

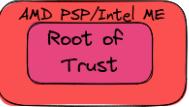


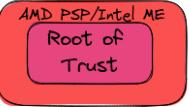


U-Boot

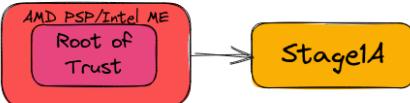
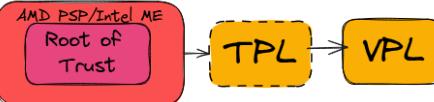
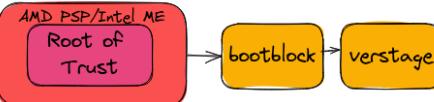
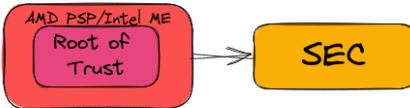


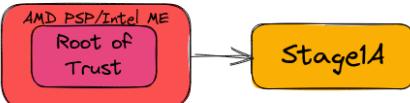
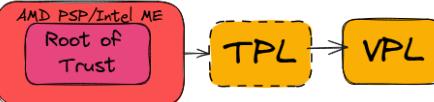
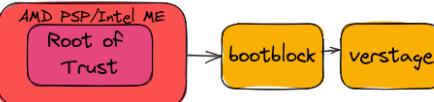
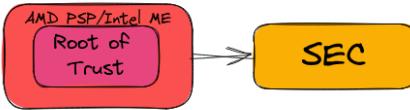


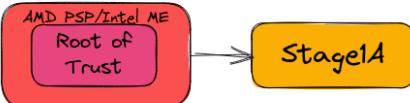
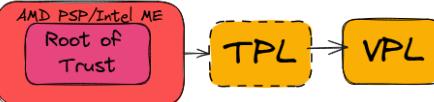
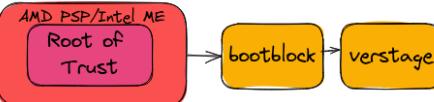
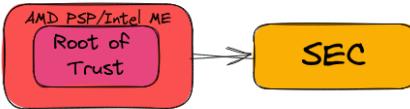


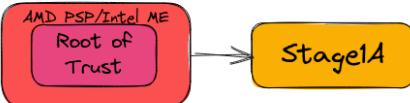
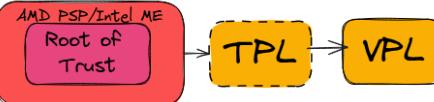
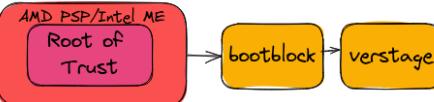
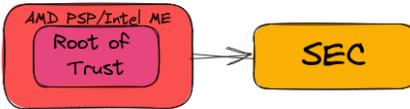


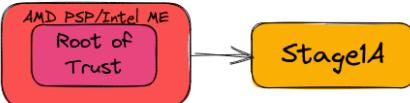
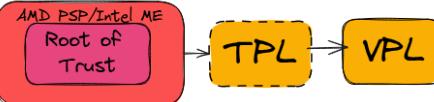
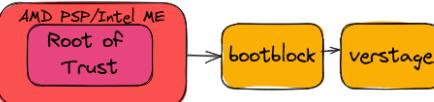
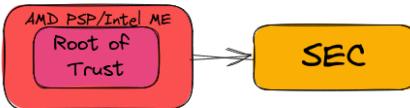


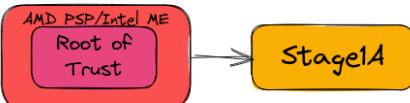
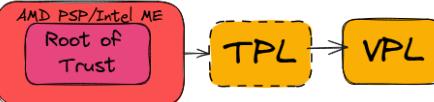
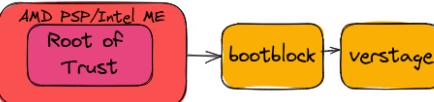
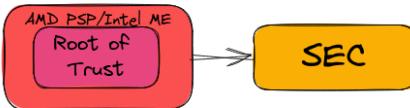


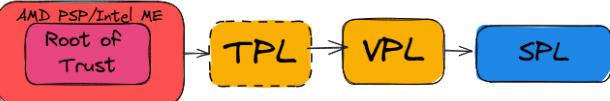
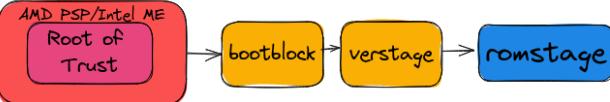


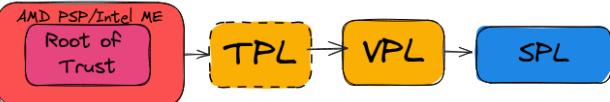
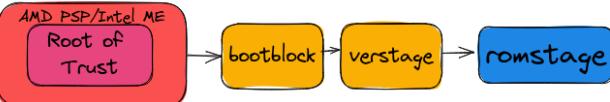


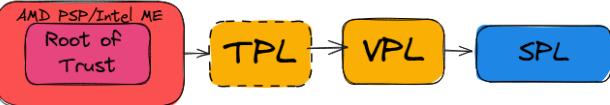
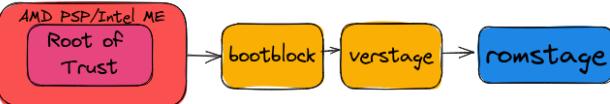


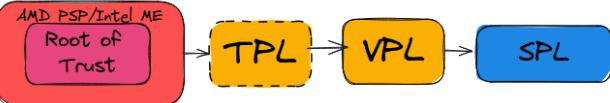
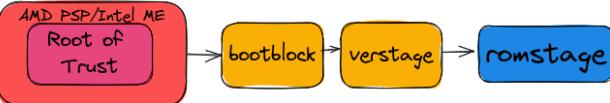


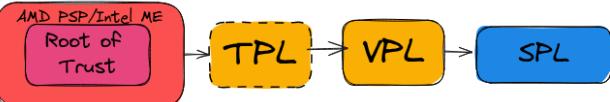
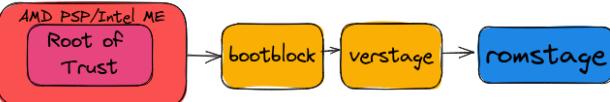


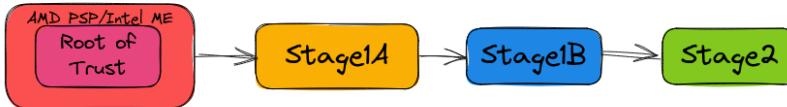
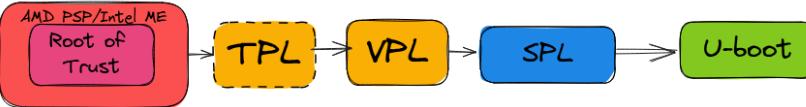
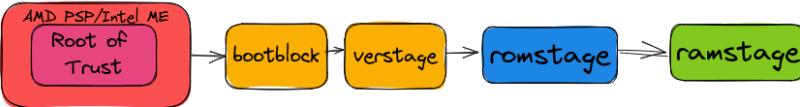
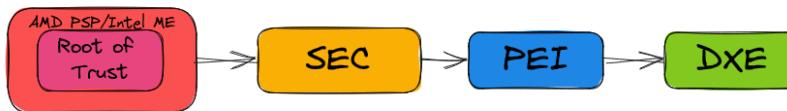


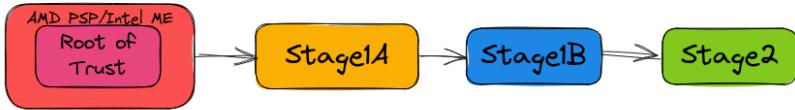
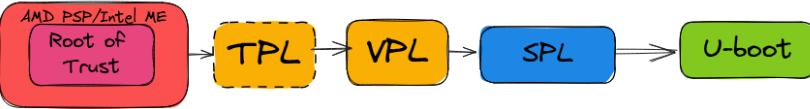
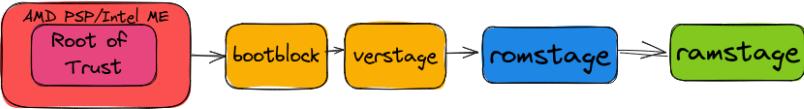
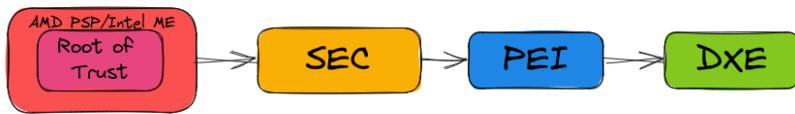


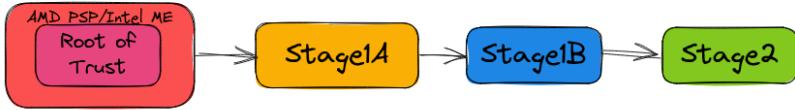
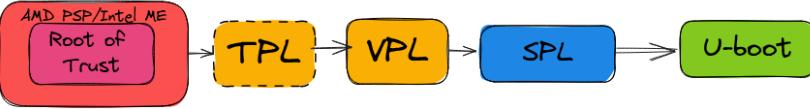
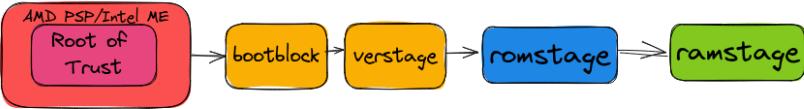
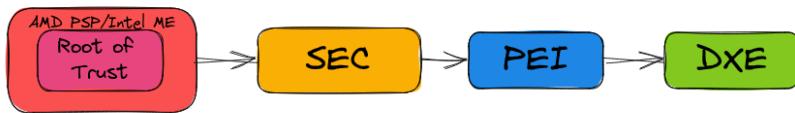


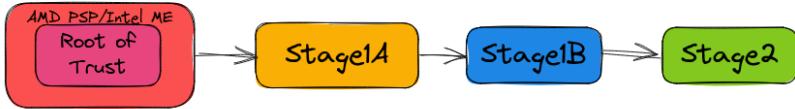
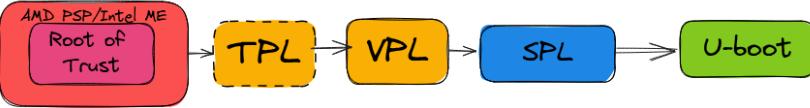
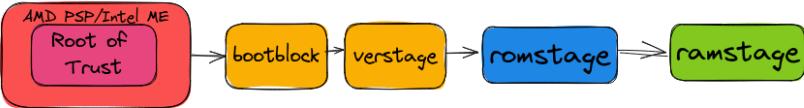
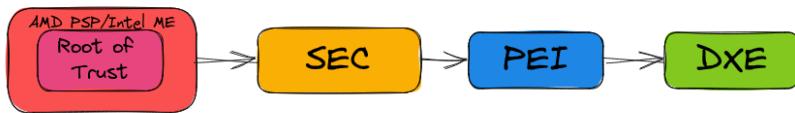


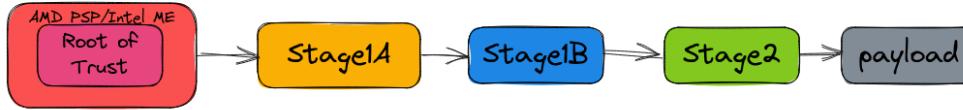
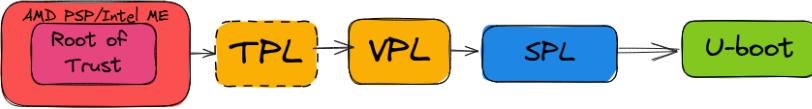
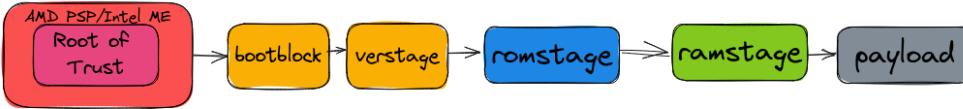
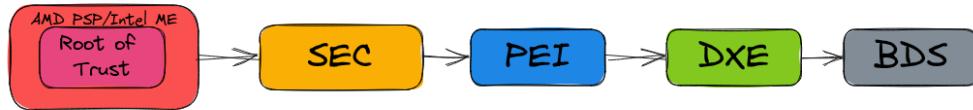


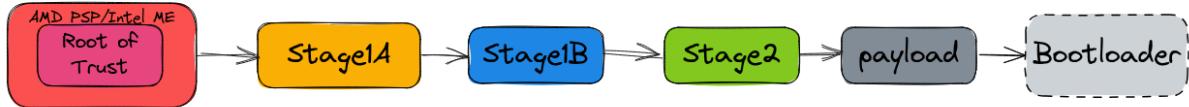
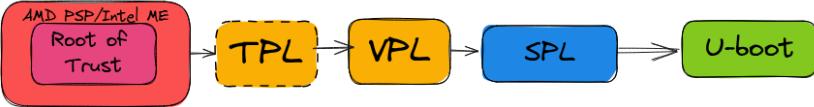
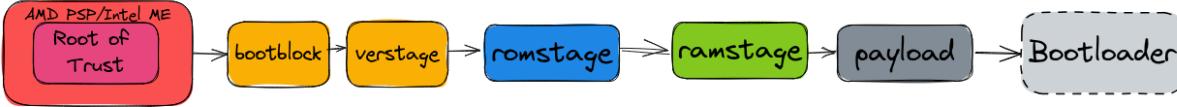
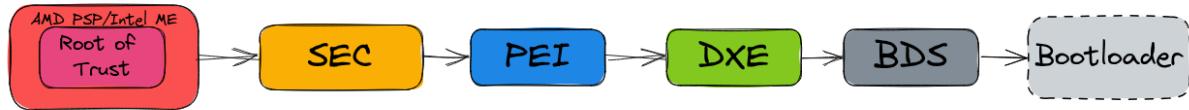


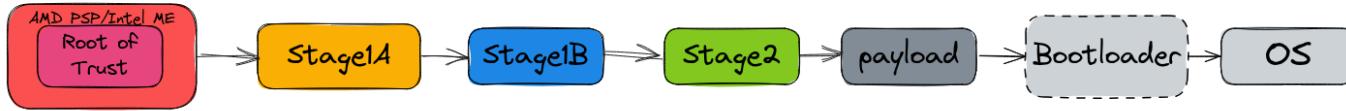
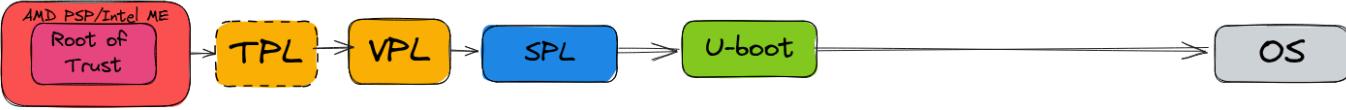
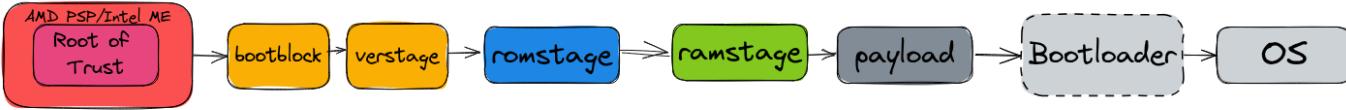
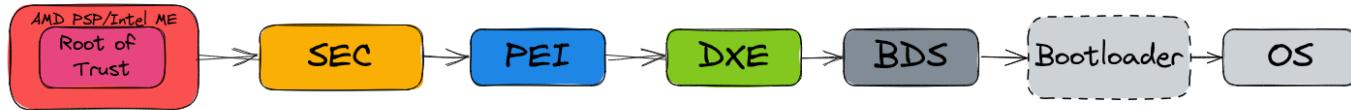


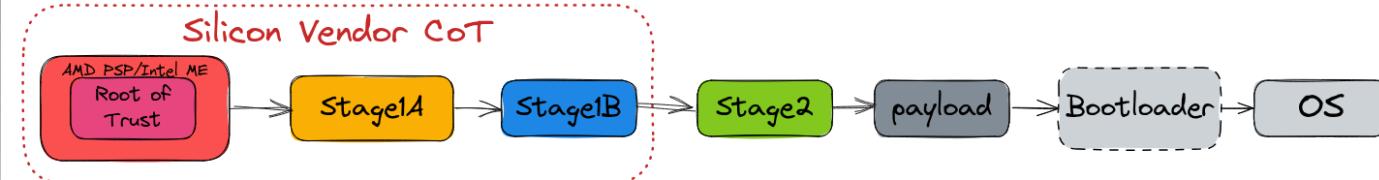
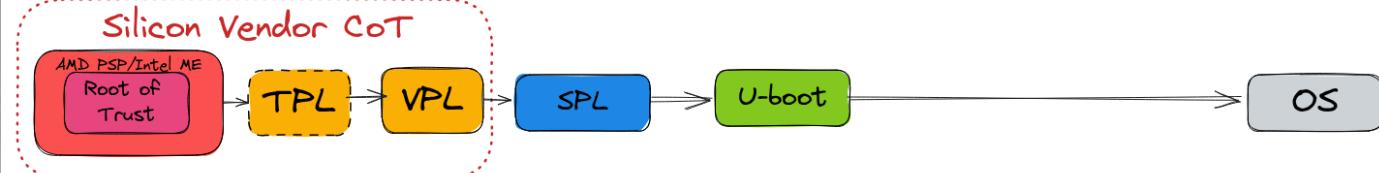
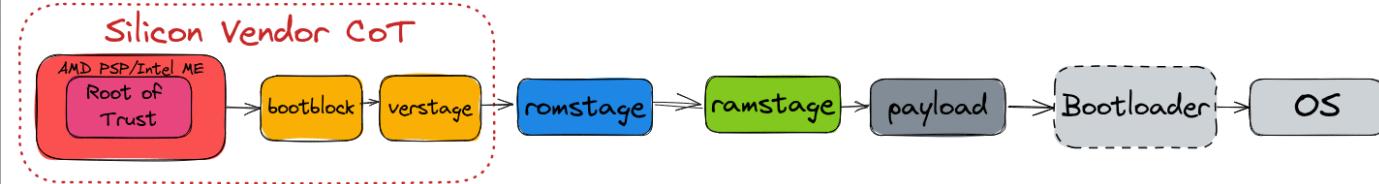
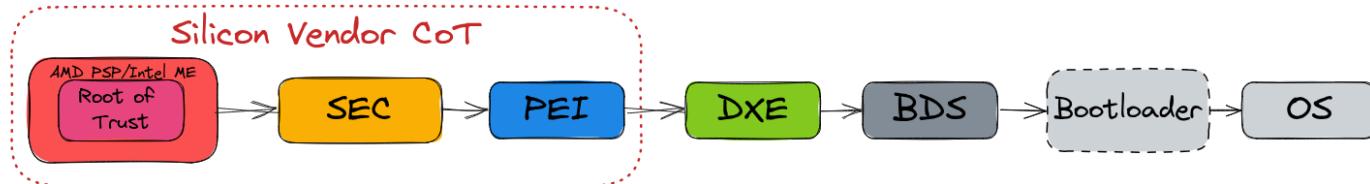


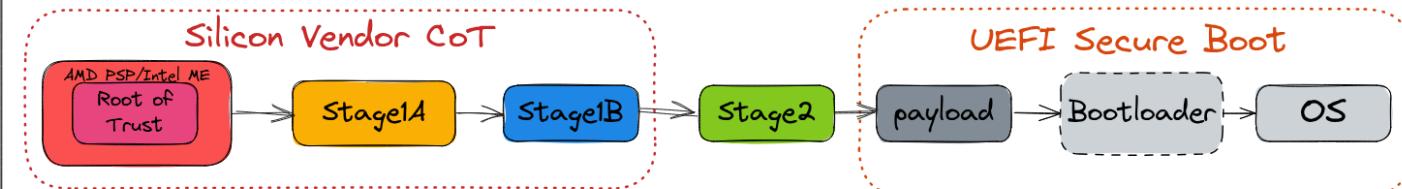
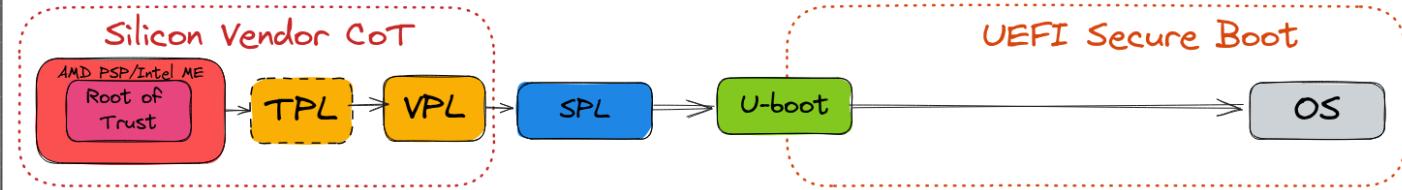
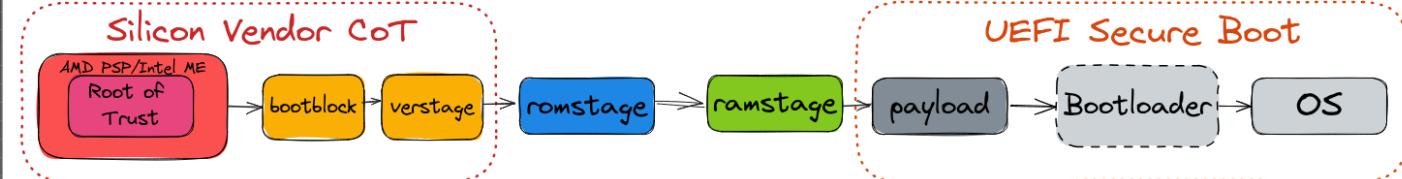
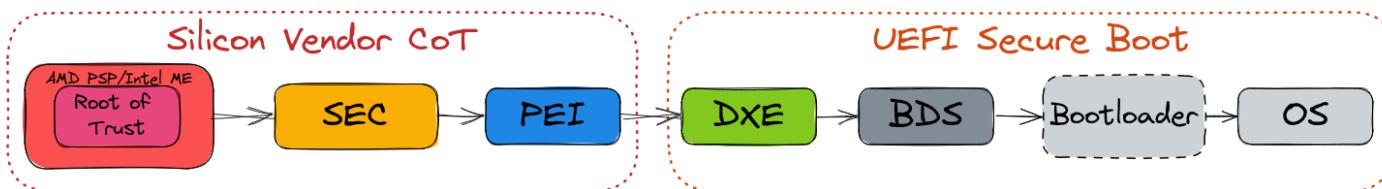


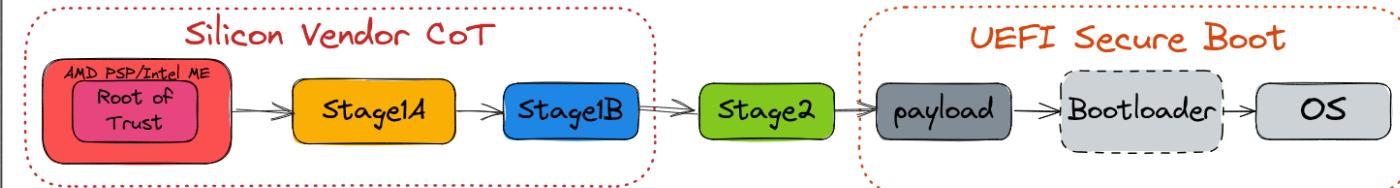
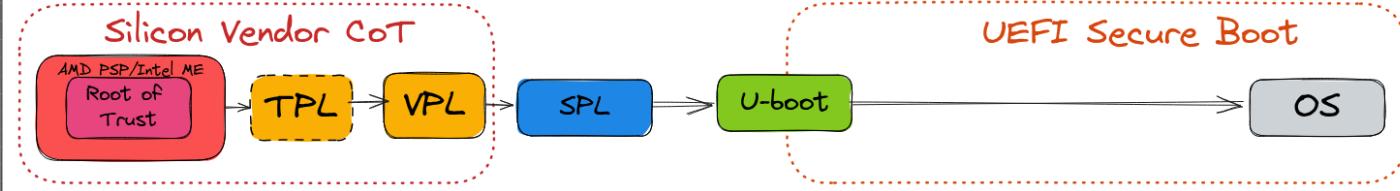
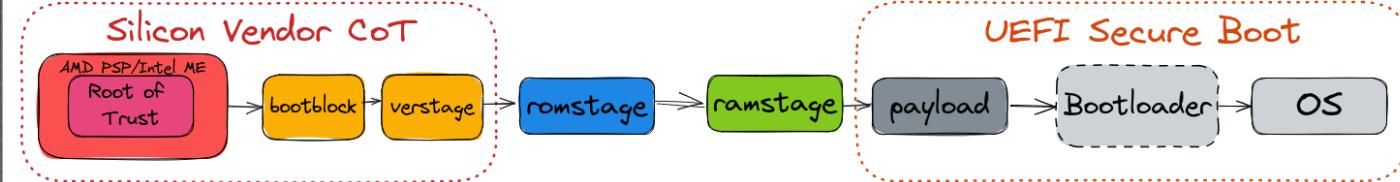
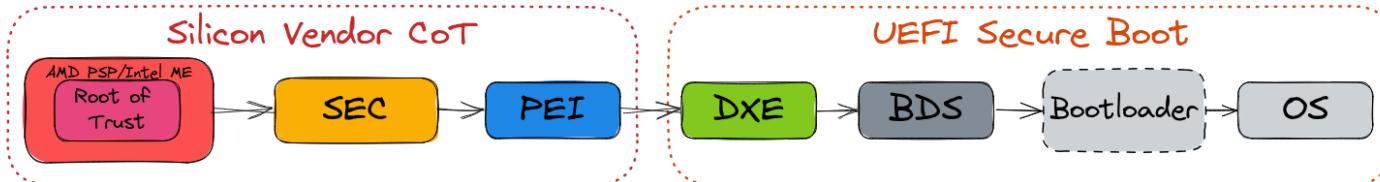


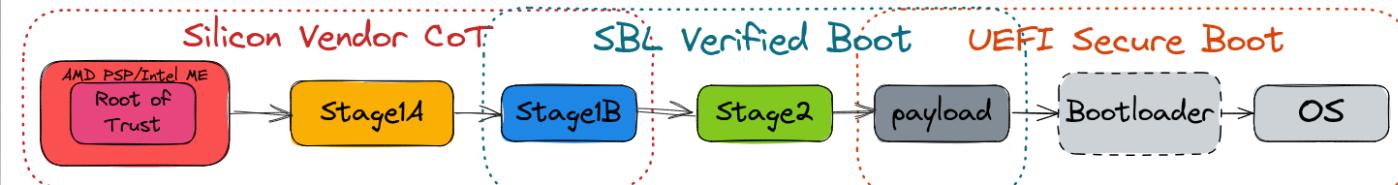
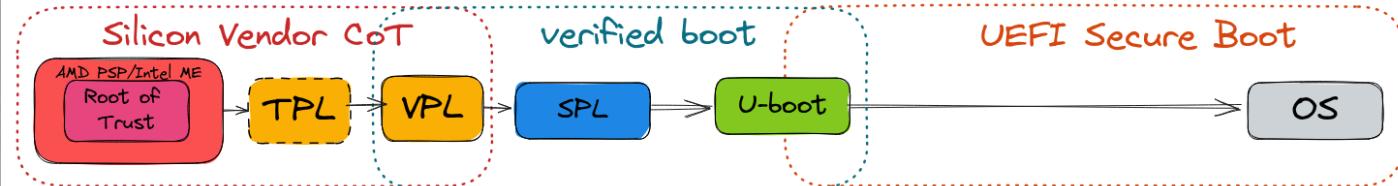
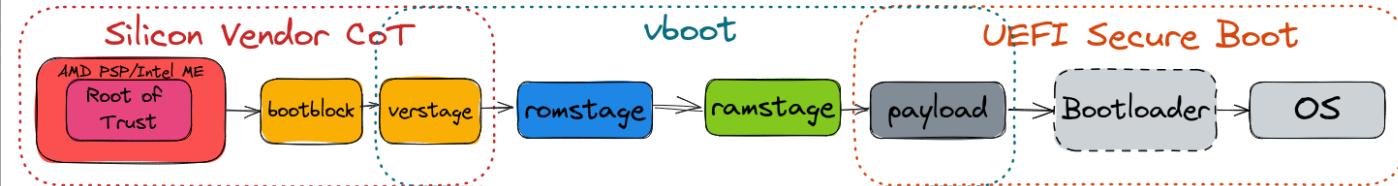
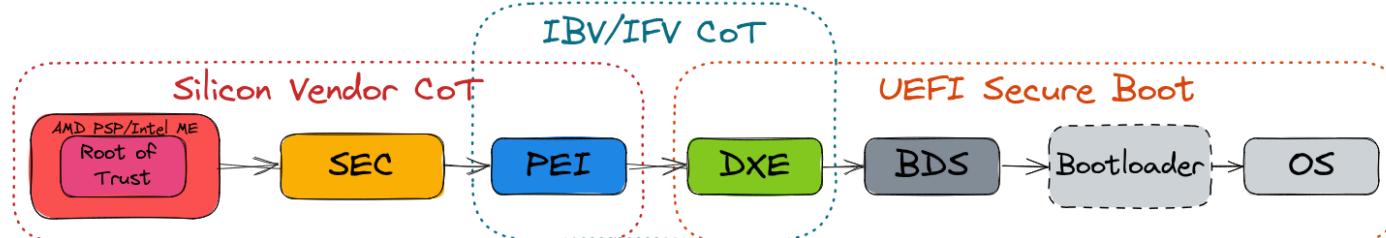


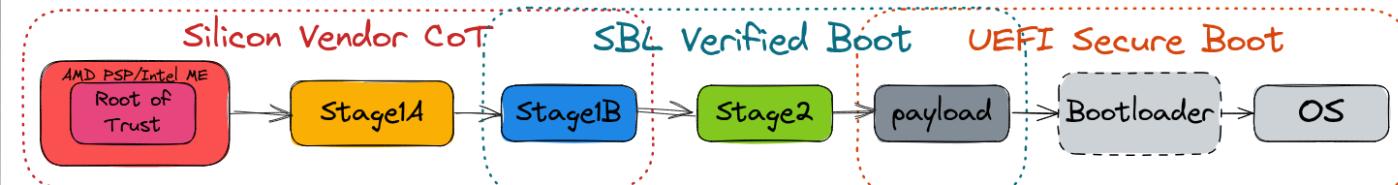
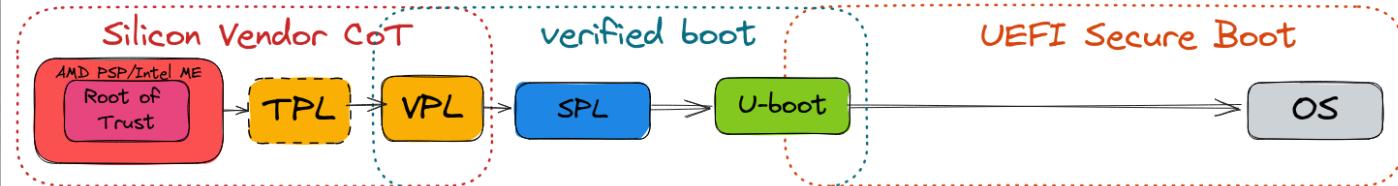
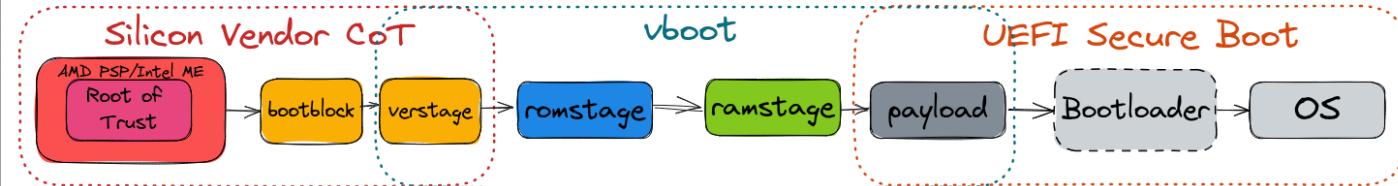
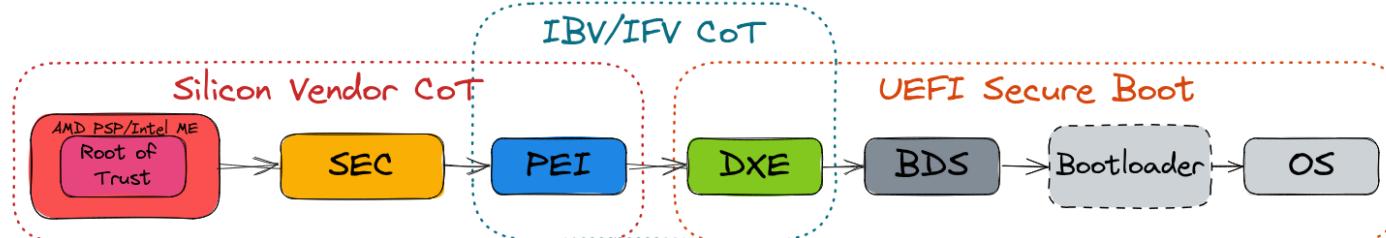


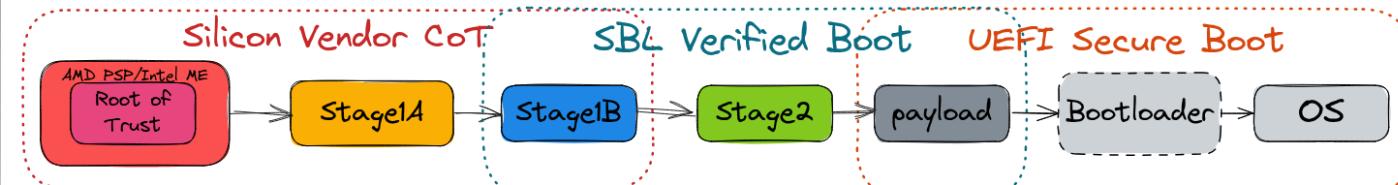
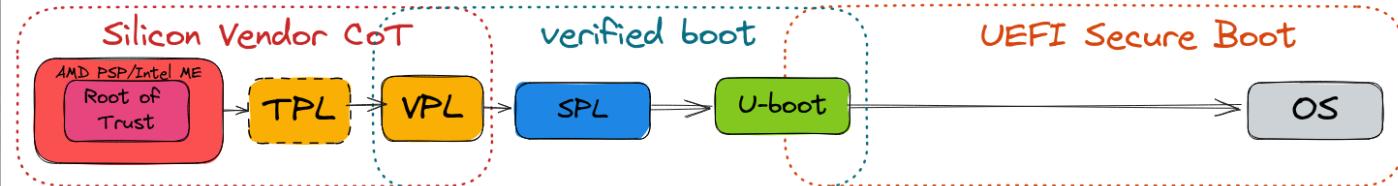
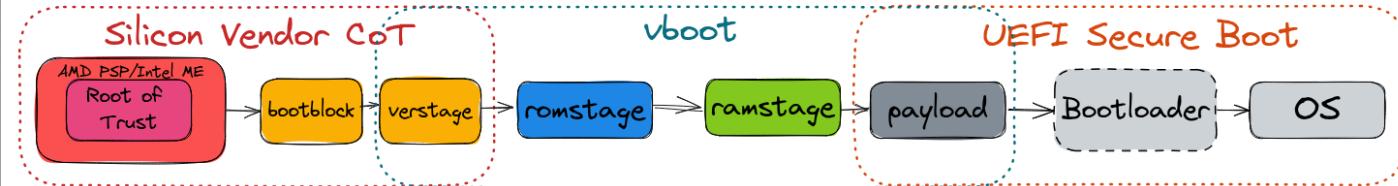
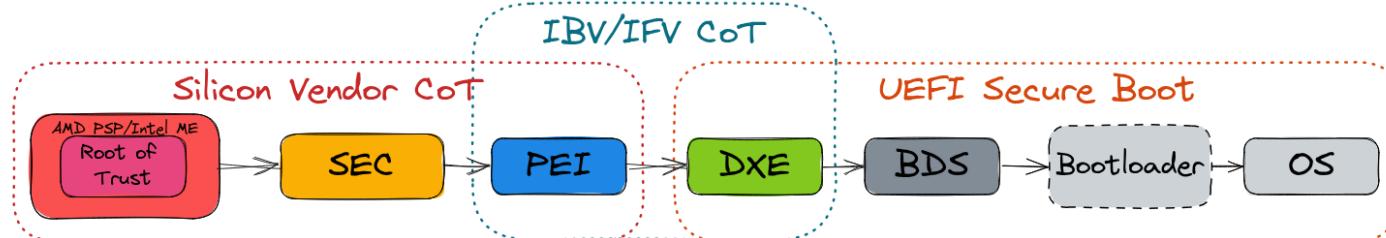


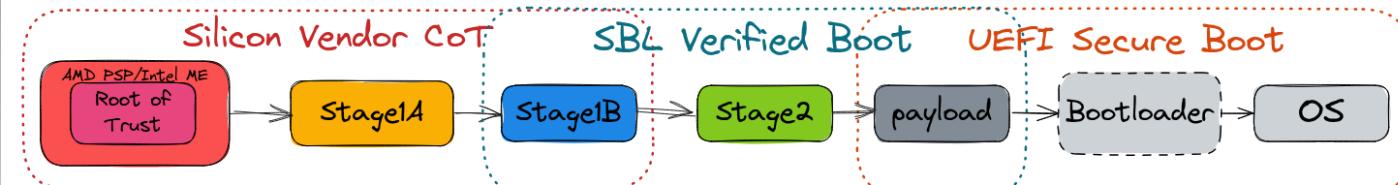
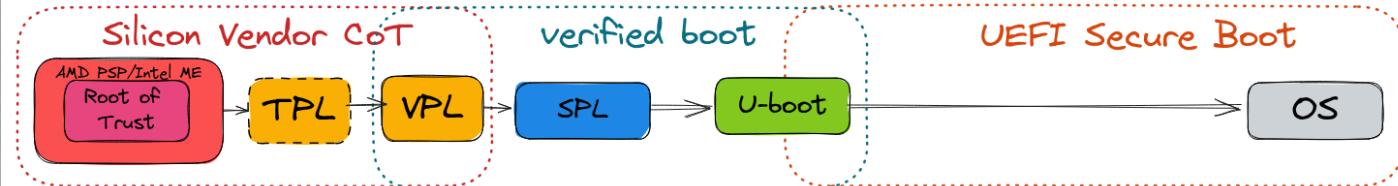
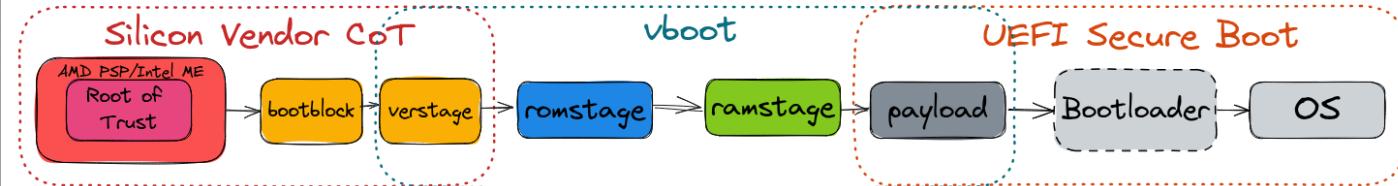
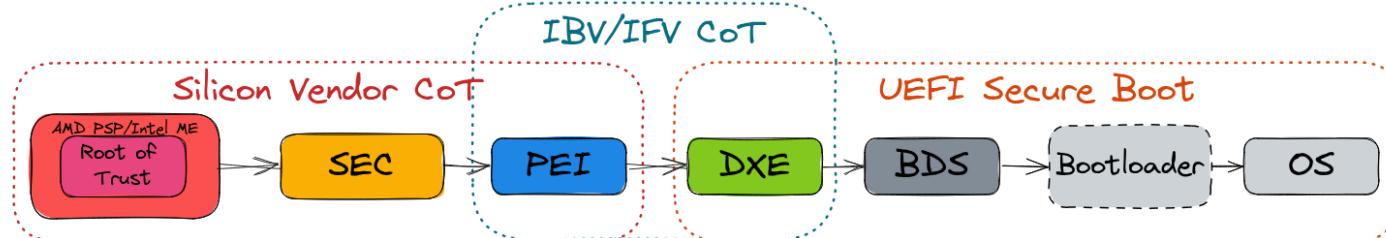


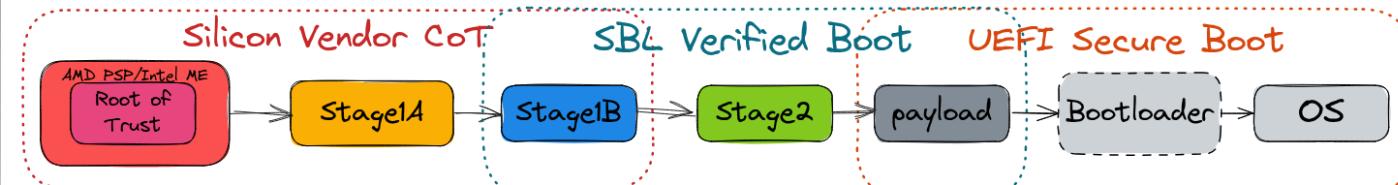
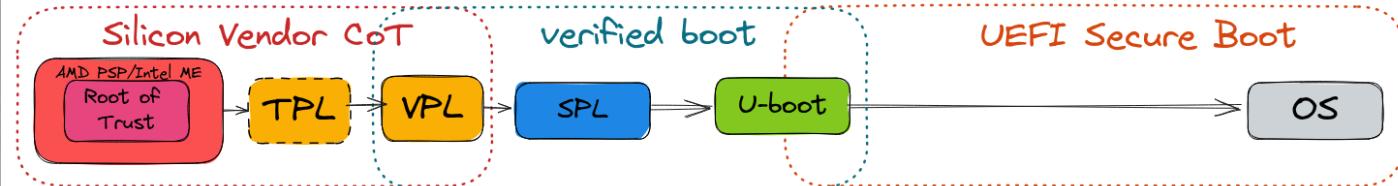
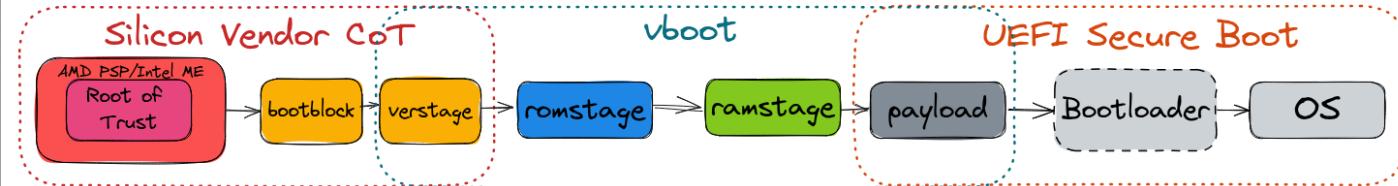
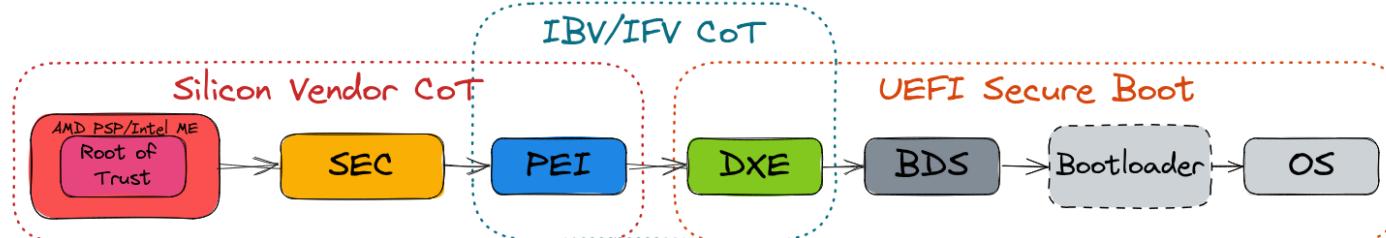


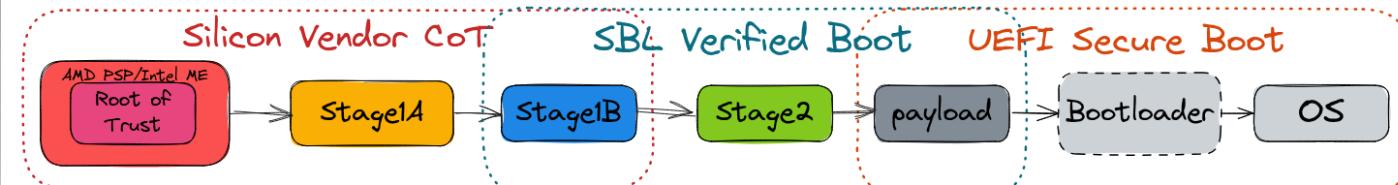
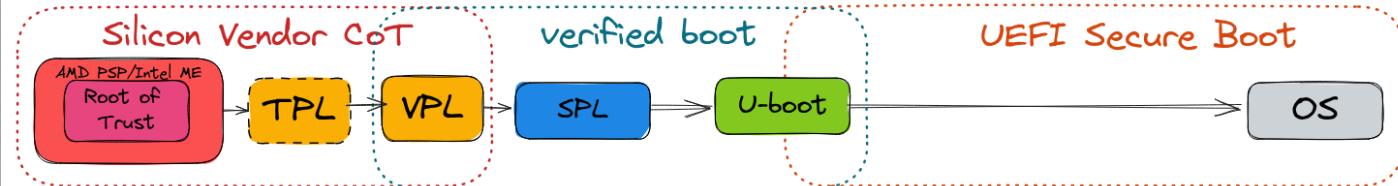
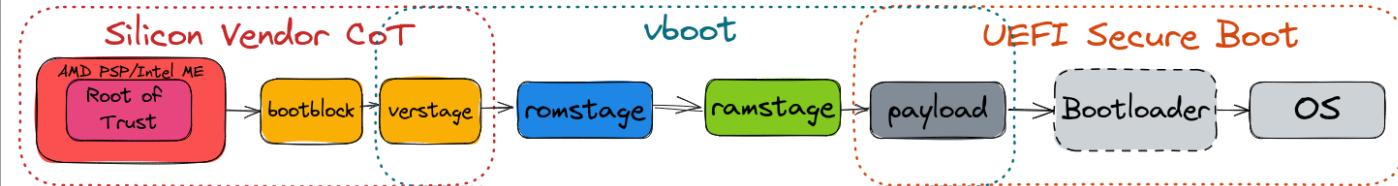
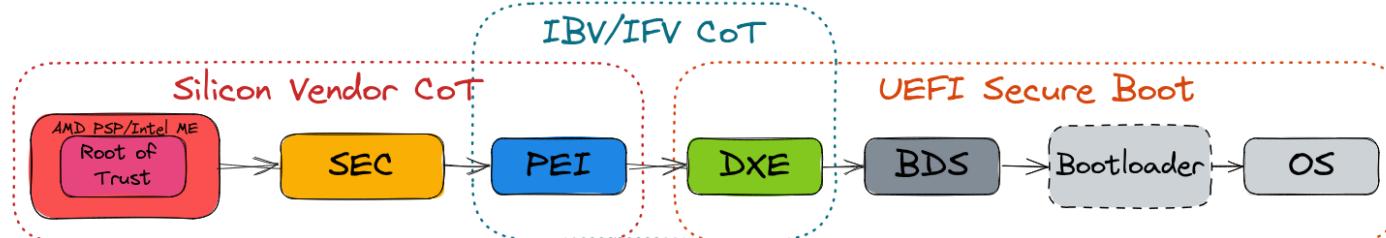












# Quiz #1

# Quiz #1

What are the potential threats of unauthorized access to boot firmware?

# Quiz #1

What are the potential threats of unauthorized access to boot firmware?

- firmware corruption (DoS),
- ransomware deployment,
- persistent malware,

# Quiz #1

What are the potential threats of unauthorized access to boot firmware?

- firmware corruption (DoS),
- ransomware deployment,
- persistent malware,

Which standard bodies were most influential in protecting boot firmware?

# Quiz #1

What are the potential threats of unauthorized access to boot firmware?

- firmware corruption (DoS),
- ransomware deployment,
- persistent malware,

Which standard bodies were most influential in protecting boot firmware?

- NIST,
- TCG,
- UEFI Forum,

# Quiz #1

What are the potential threats of unauthorized access to boot firmware?

- firmware corruption (DoS),
- ransomware deployment,
- persistent malware,

Which standard bodies were most influential in protecting boot firmware?

- NIST,
- TCG,
- UEFI Forum,

How would you define Root of Trust?

# Quiz #1

What are the potential threats of unauthorized access to boot firmware?

- firmware corruption (DoS),
- ransomware deployment,
- persistent malware,

Which standard bodies were most influential in protecting boot firmware?

- NIST,
- TCG,
- UEFI Forum,

How would you define Root of Trust?

- entity in the system which is unconditionally trusted



What is the difference between UEFI and TianoCore?

## What is the difference between UEFI and TianoCore?

- UEFI, doesn't mean anything in itself, there is UEFI Forum, which maintain UEFI Specification,
- TianoCore, is a community that maintain reference implementation of UEFI Specification,

What is the difference between UEFI and TianoCore?

- UEFI, doesn't mean anything in itself, there is UEFI Forum, which maintain UEFI Specification,
- TianoCore, is a community that maintain reference implementation of UEFI Specification,

What is the name of UEFI boot phase that passes control from firmware to OS Loader/OS Kernel or Application?

What is the difference between UEFI and TianoCore?

- UEFI, doesn't mean anything in itself, there is UEFI Forum, which maintain UEFI Specification,
- TianoCore, is a community that maintain reference implementation of UEFI Specification,

What is the name of UEFI boot phase that passes control from firmware to OS Loader/OS Kernel or Application?

- BDS (Boot Device Selection)

What is the difference between UEFI and TianoCore?

- UEFI, doesn't mean anything in itself, there is UEFI Forum, which maintain UEFI Specification,
- TianoCore, is a community that maintain reference implementation of UEFI Specification,

What is the name of UEFI boot phase that passes control from firmware to OS Loader/OS Kernel or Application?

- BDS (Boot Device Selection)

Which UEFI boot phases are covered by UEFI Secure Boot?

What is the difference between UEFI and TianoCore?

- UEFI, doesn't mean anything in itself, there is UEFI Forum, which maintain UEFI Specification,
- TianoCore, is a community that maintain reference implementation of UEFI Specification,

What is the name of UEFI boot phase that passes control from firmware to OS Loader/OS Kernel or Application?

- BDS (Boot Device Selection)

Which UEFI boot phases are covered by UEFI Secure Boot?

- DXE
- BDS
- OS Loader/OS Kernel and Application

What is the difference between UEFI and TianoCore?

- UEFI, doesn't mean anything in itself, there is UEFI Forum, which maintain UEFI Specification,
- TianoCore, is a community that maintain reference implementation of UEFI Specification,

What is the name of UEFI boot phase that passes control from firmware to OS Loader/OS Kernel or Application?

- BDS (Boot Device Selection)

Which UEFI boot phases are covered by UEFI Secure Boot?

- DXE
- BDS
- OS Loader/OS Kernel and Application

What are the generic names of other Chain of Trust technologies which can be leveraged by boot process?

What is the difference between UEFI and TianoCore?

- UEFI, doesn't mean anything in itself, there is UEFI Forum, which maintain UEFI Specification,
- TianoCore, is a community that maintain reference implementation of UEFI Specification,

What is the name of UEFI boot phase that passes control from firmware to OS Loader/OS Kernel or Application?

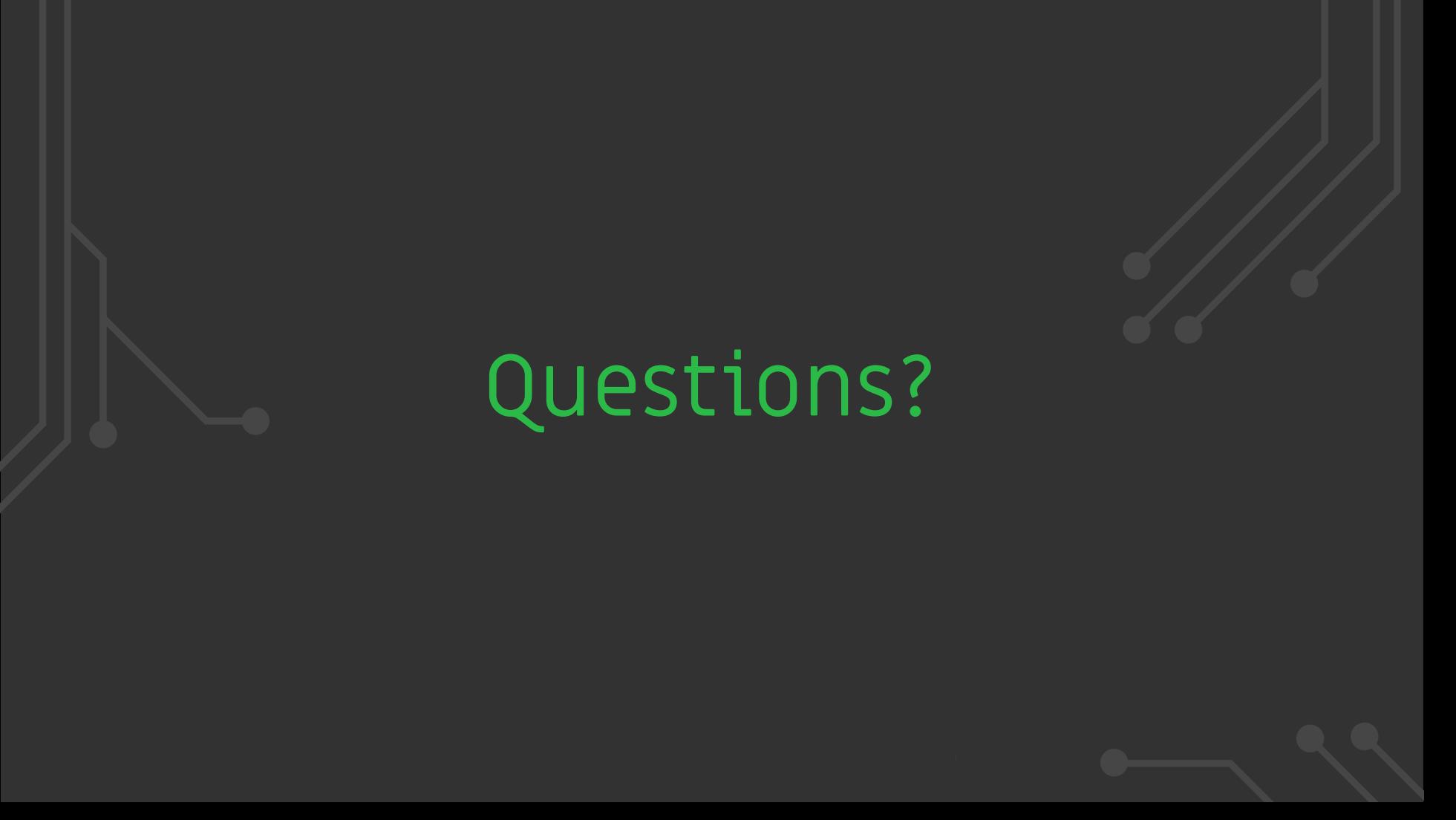
- BDS (Boot Device Selection)

Which UEFI boot phases are covered by UEFI Secure Boot?

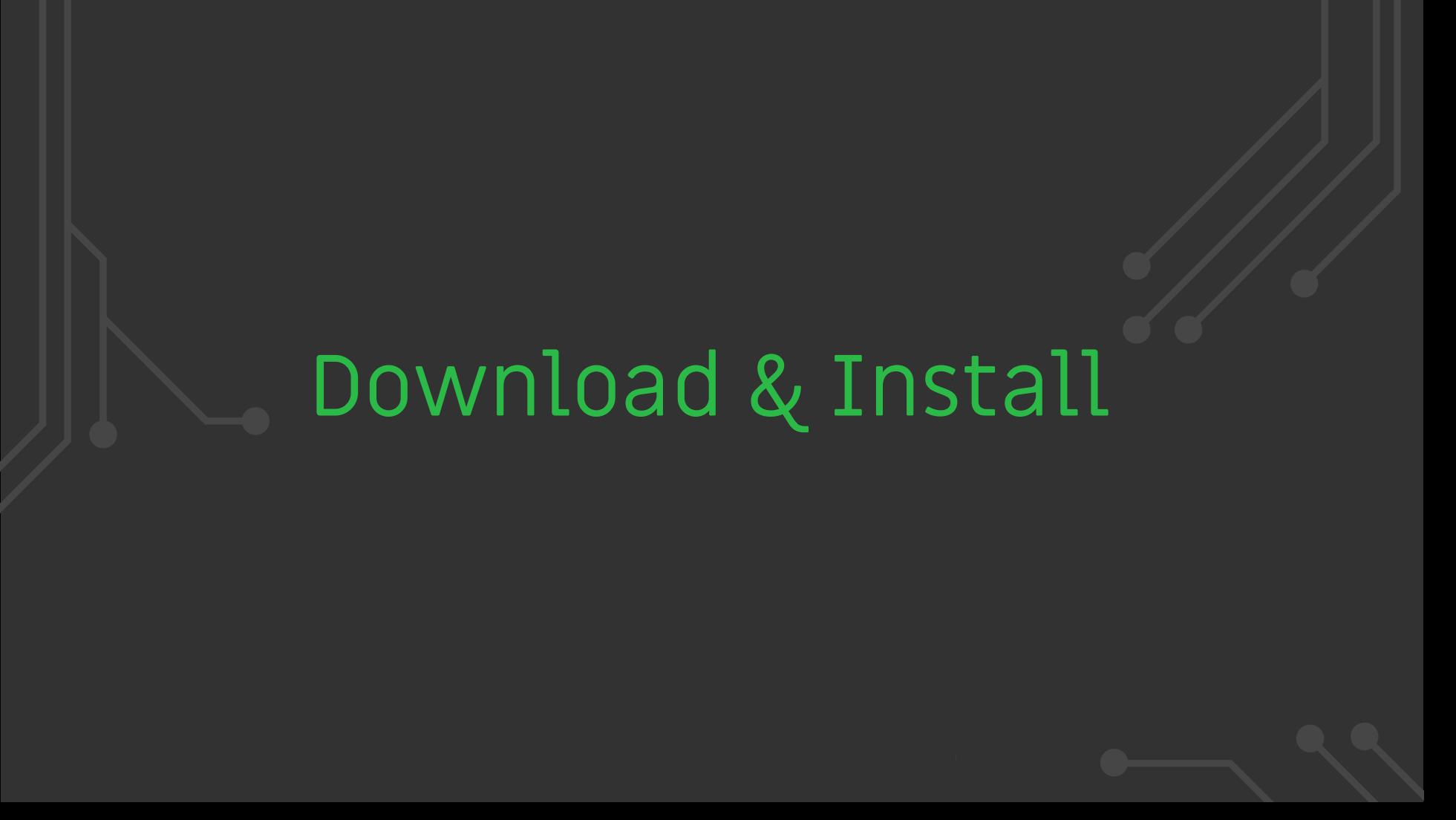
- DXE
- BDS
- OS Loader/OS Kernel and Application

What are the generic names of other Chain of Trust technologies which can be leveraged by boot process?

- SV CoT, IBV/IFV CoT and UEFI Secure Boot



# Questions?



# Download & Install

# Ubuntu Instructions

Practice #101 Exercise #0

Following instruction was tested on Debian 12 using VirtualBox 7.1.12r169651. Please report any issues [here](#). Please use following VirtualBox [installation guide](#) to install it in your operating system.

## Notice

On laptops running UEFI Secure Boot enabled there is need for signing the signing VirtualBox driver.  
dkms delivered drivers have to be resigned by PK, KEK or something in DB.

# Download and import OST2 Ubuntu VM

- Download the OST2 Ubuntu VM OVA image using command line:

```
curl -o ubuntu-24.04.2-arch5041-ds09sbl-v0.9.ova -u 'tdPZsa5MPw5tSXs:DS09sbl_2025' \
https://cloud.3mdeb.com/public.php/webdav
```

- Download SHA256 of VM OVA image:

```
curl -o ubuntu-24.04.2-arch5041-ds09sbl-v0.9.ova.sha256 -u 'apCdTa4PZMEqJPB:DS09sbl_2025' \
https://cloud.3mdeb.com/public.php/webdav
```

Or use [link](#) (password: DS09sbl\_2025) in your web browser.

- Confirm hash is correct:

```
sha256sum -c ubuntu-24.04.2-arch5041-ds09sbl-v0.9.ova.sha256
```

Output should look as follows:

```
ubuntu-24.04.2-arch5041-ds09sbl-v0.9.ova: OK
```

- Link expire on 25 September 2025.

# Extension Pack

- Some advanced features like RDP require installing Extension Pack. To install it please follow [Installing an Extension Pack](#) section.
- On Linux hosts you can:

```
wget https://download.virtualbox.org/virtualbox/7.1.12/Oracle_VirtualBox_Extension_Pack-7.1.12.vbox-extpack
```

- Hash

```
sha256sum Oracle_VirtualBox_Extension_Pack-7.1.12.vbox-extpack
```

Output:

```
c7ed97f4755988ecc05ec633475e299bbc1e0418cc3d143747a45c99df53abd3 Oracle_VirtualBox_Extension_Pack-7.1.12.vbox-extpack
```

- Install:

```
vboxmanage extpack install Oracle_VirtualBox_Extension_Pack-7.1.12.vbox-extpack
```

And follow instruction on screen

# Ubuntu Instructions

- Import using the following command (or use GUI):

```
VBoxManage import ubuntu-24.04.2-arch5041-ds09sbl-v0.9.ova
```

- To start headless:

```
VBoxHeadless -s ubuntu-24.04.2-arch5041-ds09sbl-v0.9
```

Output should look as follows:

```
Oracle VirtualBox Headless Interface 7.1.12
Copyright (c) 2008-2025 Oracle and/or its affiliates
```

```
Starting virtual machine: VRDE server is listening on port 5001.
10% ... 20% ... 30% ... 40% ... 50% ... 60% ... 70% ... 80% ... 90% ... 100%
```

# Ubuntu Instructions

## Login/pass

```
user: user  
password: ubuntu
```

## RDP access

```
vinagre rdp://0.0.0.0:5001
```

Instead of `0.0.0.0` you can use your IP address.

## SSH access

```
ssh -p 2022 user@0.0.0.0
```

Make sure your host OS (not VM) user can access and attach USB devices to VM. In Linux it means the user should be in the `vboxusers` group. To confirm that you can use the `id` command. If your user is not in `vboxusers` please add it:

```
sudo usermod -aG vboxusers $USER
```

Otherwise you will see warning like follow when calling:

```
VBoxManage list usbhost
```

Output:

```
VBoxManage: warning: VirtualBox is not currently allowed to access USB Devices:
```

```
<none>
```

It may require restarting computer or relogin after setting correct permission.

If you want to use GUI or face issues please check VirtualBox documentation for your OS [here](#).

Our goal is to prove that we can connect USB device, like USB storage or USB-to-TTL cable, to VM. To do that first list devices with `VBoxManage list usbhost`, choose UUID and pass it to `usbattach`:

Host USB Devices:

```
UUID: ffdc312e-1ba0-4996-ba11-a69649788344
VendorId: 0x067b (067B)
ProductId: 0x2303 (2303)
Revision: 3.0 (0300)
Port: 0
USB version/speed: 2/Full
Manufacturer: Prolific Technology Inc.
Product: USB-Serial Controller
Address: sysfs:/sys/devices/pci0000:00/0000:00:14.0/usb2/2-1//device:/dev/vboxusb/002/002
Current State: Busy

UUID: e7ba05ad-6ebd-41b1-8cf9-1d33935e81a0
VendorId: 0x05e3 (05E3)
ProductId: 0x0736 (0736)
Revision: 2.114 (02114)
Port: 11
USB version/speed: 2/High
( ... )
```

Our goal is to prove that we can connect USB device, like USB storage or USB-to-TTL cable, to VM. To do that first list devices with `VBoxManage list usbhost`, choose UUID and pass it to `usbattach`:

Host USB Devices:

|                    |  |
|--------------------|--|
| UUID:              | ffdc312e-1ba0-4996-ba11-a69649788344   |
| VendorId:          | 0x067b (067B)  |
| ProductId:         | 0x2303 (2303)  |
| Revision:          | 3.0 (0300)   |
| Port:              | 0  |
| USB version/speed: | 2/Full   |
| Manufacturer:      | Prolific Technology Inc.   |
| Product:           | USB-Serial Controller  |
| Address:           | sysfs:/sys/devices/pci0000:00/0000:00:14.0/usb2/2-1//device:/dev/vboxusb/002/002 |
| Current State:     | Busy   |
|                    |  |
| UUID:              | e7ba05ad-6ebd-41b1-8cf9-1d33935e81a0   |
| VendorId:          | 0x05e3 (05E3)  |
| ProductId:         | 0x0736 (0736)  |
| Revision:          | 2.114 (02114)  |
| Port:              | 11   |
| USB version/speed: | 2/High   |
| ( ... )            |  |

Our goal is to prove that we can connect USB device, like USB storage or USB-to-TTL cable, to VM. To do that first list devices with `VBoxManage list usbhost`, choose UUID and pass it to `usbattach`:

```
VBoxManage controlvm "ubuntu-24.04.2-arch5041-ds09sbl-v0.9" usbattach ffdc312e-1ba0-4996-ba11-a69649788344
```

Our goal is to prove that we can connect USB device, like USB storage or USB-to-TTL cable, to VM. To do that first list devices with `VBoxManage list usbhost`, choose UUID and pass it to `usbattach`:

```
VBoxManage controlvm "ubuntu-24.04.2-arch5041-ds09sbl-v0.9" usbdetach ffdc312e-1ba0-4996-ba11-a69649788344
```

Confirm by auditing system logs inside VM:

```
user@OST2-VM:~$ sudo dmesg
```

Successful attach operation should return clear information in OST2 Ubuntu VM system logs:

```
[198091.043315] usb 1-1: new full-speed USB device number 2 using ohci-pci
[198091.558896] usb 1-1: New USB device found, idVendor=067b, idProduct=2303, bcdDevice= 3.00
[198091.558913] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[198091.558920] usb 1-1: Product: USB-Serial Controller
[198091.558925] usb 1-1: Manufacturer: Prolific Technology Inc.
[198091.687000] usbcore: registered new interface driver usbserial_generic
[198091.687010] usbserial: USB Serial support registered for generic
[198091.690386] usbcore: registered new interface driver pl2303
[198091.690412] usbserial: USB Serial support registered for pl2303
[198091.690433] pl2303 1-1:1.0: pl2303 converter detected
[198091.742437] usb 1-1: pl2303 converter now attached to ttyUSB0
```

## Issues and contribution

The OST2 Ubuntu VM is automatically built using [Packer](#). The scripts for building VM are open-source and available [here](#). Feel free to contribute your improvements through issues and PRs.

# Command line use

- Course materials are mostly command line based. If you are not familiar with Linux command line please check [Linux Command Line Cheat Sheet](#).
- For file transfer to/from host OS to VM allow use of `scp`.
  - Verify upload by transferring files using the command line:

```
scp -P 2022 some_file user@0.0.0.0:/tmp
```
  - Verify download by transferring files using the command line:

```
scp -P 2022 user@0.0.0.0:/tmp/some_file .
```
- If your OS does not allow `scp`, consider VirtualBox features e.g. [shared folders](#).

Please make sure file sharing method work reliably both sides.

# vim

- For code inspection and editing we use `vim` without any plugins, if you prefer another editor please configure it before training inside the VM.
  - `vim` provides native support for `ctags` and `cscope` and we will use those two indexers for quick searching.
  - `tags` and `cscope` database are pre-generated inside the image.
- In `vim` you can navigate to a function, variable, or definition by pressing the `Ctrl + ]` key after placing the cursor on the symbol. To go back, press `Ctrl + t`.
- For a better understanding of how `ctags` and `cscope` can be used please check `:help cs` and `:help ctags` in `vim`.
- If you are unfamiliar with `vim` please consider looking at [Vim Cheat Sheet](#).

# Introductory tmux

Practice #101 Exercise #1

tmux is a terminal multiplexer: it enables many terminals to be created, accessed, and controlled from a single screen. tmux may be detached from a screen, continue running in the background, and then later reattached.

It is similar to the famous GNU screen application but way more powerful.

Since most of our exercises will happen over ssh and we would like to avoid opening an endless number of ssh sessions, we need tmux to manage our terminals. To run session:

```
user@OST2-VM:~$ tmux
```

To detach from session `Ctrl + b d`. The session is still running in the background. To list running sessions:

```
user@OST2-VM:~$ tmux list-sessions
```

To attach to the session:

```
user@OST2-VM:~$ tmux attach-session -t [session_name]
```

# Introductory tmux



Most useful keyboard shortcuts:

- `Ctrl-b c` - create new terminal
- `Ctrl-b n` - next terminal
- `Ctrl-b p` - previous terminal
- `Ctrl-b d` - exit terminal (generic shell shortcut)

For more please visit <https://tmuxcheatsheet.com/>.



- QEMU is a free and open-source emulator and virtualizer.
  - Written in C.
  - Version used in following mini-course: 8.2.2 (version available in Ubuntu VM)
  - License: GPLv2
  - Supported OSes: Linux OS, macOS, FreeBSD, NetBSD, OpenBSD, and Windows

## QEMU use cases

- **System emulator** - QEMU provides a virtual model of an entire machine (CPU, memory, and emulated devices)
  - Full CPU emulation
  - Virtualized CPU through hypervisors like KVM, Xen, Hax, or Hypervisor.Framework
- **User mode emulation** - QEMU runs process compiled for the target CPU on the host CPU.

# Introductory QEMU

Practice #101 Exercise #2

Let's confirm QEMU is correctly installed inside VM and can boot :

```
user@OST2-VM:~$ qemu-system-x86_64 -nographic
```

Important keyboard shortcuts inside QEMU:

```
C-a h      print this help
C-a x      exit emulator
C-a s      save disk data back to file (if -snapshot)
C-a t      toggle console timestamps
C-a b      send break (magic sysrq)
C-a c      switch between console and monitor
C-a C-a    sends C-a
```

Please try to print the above help and exit the emulator.

Let's boot Slim Bootloader in QEMU Q35 inside our OST2 Ubuntu VM:

```
user@OST2-VM:~$ cd ~/training_materials/src/slimbootloader
```

Prepare repo:

```
user@OST2-VM:~$ git remote add sbl https://github.com/slimbootloader/slimbootloader.git
user@OST2-VM:~$ git checkout -b qemu_sbl 50c9f1005c7df79c3f395b1382811ae966f9e56b
```

Prepare keys:

```
user@OST2-VM:~$ mkdir -p .. /QemuSblKeys
user@OST2-VM:~$ python BootloaderCorePkg/Tools/GenerateKeys.py -k .. /QemuSblKeys/
```

Build:

```
user@OST2-VM:~$ SBL_KEY_DIR=.. /QemuSblKeys python BuildLoader.py build qemu
```

Output should be as follows:

```
Done [qemu] !
```

Binary is in `Outputs/qemu/SlimBootloader.bin`.

To boot we can use:

```
qemu-system-x86_64 -machine q35 -nographic -serial mon:stdio -pflash Outputs/qemu/SlimBootloader.bin
```

Output should look as follows:

```
===== Try Booting with Boot Option 3 =====
BootMediumPciBase(0x400)
Getting boot image from USB
Failed to initialize USB bus !
Failed to init media - Unsupported
Failed to Initialize Boot Device - Type 5, Instance 0
Payload normal heap: 0x2000000 (0x10F000 used)
Payload reserved heap: 0x4000 (0x0 used)
Payload stack: 0x10000 (0x874 used)
```

Shell>

Press **ctrl + a** than **x**. This should exit emulator.

# Slim Bootlaoder build issues

- When working a lot with Slim Bootloader you may face problems with build system determinism.
- This is not new thing with python based wrappers. We saw that previously with EDKII.
- Despite all separation made with Docker and `build.sh` cleanups you still may face some issues.
- For reference please find some signatures:

```
Exception: Failed to apply QEMU FSP patch !
SBL_KEY_DIR is set to /home/coreboot/coreboot/SblKeys !!
Traceback (most recent call last):
  File "/home/coreboot/coreboot/BuildLoader.py", line 1692, in <module>
    main()
  File "/home/coreboot/coreboot/BuildLoader.py", line 1689, in main
    args.func(args)
  File "/home/coreboot/coreboot/BuildLoader.py", line 1525, in cmd_build
    Build(board).build()
  File "/home/coreboot/coreboot/BuildLoader.py", line 1360, in build
    self.pre_build()
  File "/home/coreboot/coreboot/BuildLoader.py", line 1192, in pre_build
    raise Exception ('Failed to prepare build component binaries !')
Exception: Failed to prepare build component binaries !
Build failed.
```

- Sometimes it is important to read `Exception` message:

```
Create FSP component file '/home/coreboot/coreboot/Build/BootloaderCorePkg/DEBUG_GCC5/FV/FSP_T.bin'
Traceback (most recent call last):
  File "/home/coreboot/coreboot/BuildLoader.py", line 1692, in <module>
    main()
  File "/home/coreboot/coreboot/BuildLoader.py", line 1689, in main
    args.func(args)
  File "/home/coreboot/coreboot/BuildLoader.py", line 1525, in cmd_build
    Build(board).build()
  File "/home/coreboot/coreboot/BuildLoader.py", line 1360, in build
    self.pre_build()
  File "/home/coreboot/coreboot/BuildLoader.py", line 1238, in pre_build
    raise Exception ('Verified Boot must also enabled to enable Measured Boot!')
Exception: Verified Boot must also enabled to enable Measured Boot!
Build failed.
```

- Some recovery steps:

- `python BuildLoader.py clean`
- `git clean -ffxd` - be caution with that it can remove some files,
- Last resort destroy VM and reimport.

Sanity check if you have correct version of VM:

```
user@OST2-VM:~$ cat ~/ost2_vm_version
```

Output:

```
ubuntu-arch5041-ds09sbl-v0.9
```

# Slim Bootloader compilation for Odroid-H4

Practice #101 Exercise #3

We will compile Slim Bootloader to obtain debug version of `ifwi-image.bin` :

```
user@OST2-VM:~$ cd ~/training_materials/src/slimbootloader
```

Let's check out tree:

```
user@OST2-VM:~$ git log --graph
```

Show `007969345502fd64cb623cd43151719168cbf644`, `hardkernel_odroid_h4_v0.9.0` tag.

```
* commit 007969345502fd64cb623cd43151719168cbf644 (grafted, HEAD)
```

```
Author: Michał Żygowski <michal.zygowski@3mdeb.com>
```

```
Date: Tue Aug 5 16:01:41 2025 +0200
```

```
build.sh: Use SOURCE_DATE_EPOCH for reproducibility
```

```
Signed-off-by: Michał Żygowski <michal.zygowski@3mdeb.com>
```

- Since we use UEFI Universal Payload we have to build it first. To simplify that process we use `build.sh` script.
- [`~/training\_materials/src/slimbootloader/build.sh`](#)
- It consists of support for building Dasharo (Slim Bootloader + UEFI) for QEMU Q35 and Odroid-H4 through following functions:
  - `build_odroid_h4`
  - `build_qemu`
- If we look into `build_odroid_h4` we can find calls to:
  - `build_edk2` function used to compile UEFI Universal Payload from EDKII,
  - `build_slimloader` function used to compile Slim Bootloader,
  - `stitch_loader` function used to put together all binary components needed for full host CPU bootstrap firmware for Odroid-H4.
  - gluing code (arguments validation, error handling and defaults)
- Let's briefly look into code.

```
DOCKER_IMAGE=${DOCKER_IMAGE:-ghcr.io/dasharo/dasharo-sdk}
DOCKER_IMAGE_VER=${DOCKER_IMAGE_VER:-v1.7.0}
SBL_KEY_DIR=${SBL_KEY_DIR:-"${PWD}/SblTestKeys"}
DEBUG=${DEBUG:-0}

EDK2_FLAGS="-D CRYPTO_PROTOCOL_SUPPORT=TRUE -D SIO_BUS_ENABLE=TRUE \
-D PERFORMANCE_MEASUREMENT_ENABLE=TRUE \
-D MULTIPLE_DEBUG_PORT_SUPPORT=TRUE -D BOOTSPLASH_IMAGE=TRUE \
-D BOOT_MANAGER_ESCAPE=TRUE"
```

```
DOCKER_IMAGE=${DOCKER_IMAGE:-ghcr.io/dasharo/dasharo-sdk}
DOCKER_IMAGE_VER=${DOCKER_IMAGE_VER:-v1.7.0}
SBL_KEY_DIR=${SBL_KEY_DIR:-"${PWD}/SblTestKeys"}
DEBUG=${DEBUG:-0}

EDK2_FLAGS="-D CRYPTO_PROTOCOL_SUPPORT=TRUE -D SIO_BUS_ENABLE=TRUE \
-D PERFORMANCE_MEASUREMENT_ENABLE=TRUE \
-D MULTIPLE_DEBUG_PORT_SUPPORT=TRUE -D BOOTSPLASH_IMAGE=TRUE \
-D BOOT_MANAGER_ESCAPE=TRUE"
```

```
DOCKER_IMAGE=${DOCKER_IMAGE:-ghcr.io/dasharo/dasharo-sdk}
DOCKER_IMAGE_VER=${DOCKER_IMAGE_VER:-v1.7.0}
SBL_KEY_DIR=${SBL_KEY_DIR:-"${PWD}/SblTestKeys"}
DEBUG=${DEBUG:-0}

EDK2_FLAGS="-D CRYPTO_PROTOCOL_SUPPORT=TRUE -D SIO_BUS_ENABLE=TRUE \
-D PERFORMANCE_MEASUREMENT_ENABLE=TRUE \
-D MULTIPLE_DEBUG_PORT_SUPPORT=TRUE -D BOOTSPLASH_IMAGE=TRUE \
-D BOOT_MANAGER_ESCAPE=TRUE"
```

```
DOCKER_IMAGE=${DOCKER_IMAGE:-ghcr.io/dasharo/dasharo-sdk}
DOCKER_IMAGE_VER=${DOCKER_IMAGE_VER:-v1.7.0}
SBL_KEY_DIR=${SBL_KEY_DIR:-"${PWD}/SblTestKeys"}
DEBUG=${DEBUG:-0}

EDK2_FLAGS="-D CRYPTO_PROTOCOL_SUPPORT=TRUE -D SIO_BUS_ENABLE=TRUE \
-D PERFORMANCE_MEASUREMENT_ENABLE=TRUE \
-D MULTIPLE_DEBUG_PORT_SUPPORT=TRUE -D BOOTSPLASH_IMAGE=TRUE \
-D BOOT_MANAGER_ESCAPE=TRUE"
```

```
DOCKER_IMAGE=${DOCKER_IMAGE:-ghcr.io/dasharo/dasharo-sdk}
DOCKER_IMAGE_VER=${DOCKER_IMAGE_VER:-v1.7.0}
SBL_KEY_DIR=${SBL_KEY_DIR:-"${PWD}/SblTestKeys"}
DEBUG=${DEBUG:-0}

EDK2_FLAGS="-D CRYPTO_PROTOCOL_SUPPORT=TRUE -D SIO_BUS_ENABLE=TRUE \
-D PERFORMANCE_MEASUREMENT_ENABLE=TRUE \
-D MULTIPLE_DEBUG_PORT_SUPPORT=TRUE -D BOOTSPLASH_IMAGE=TRUE \
-D BOOT_MANAGER_ESCAPE=TRUE"
```

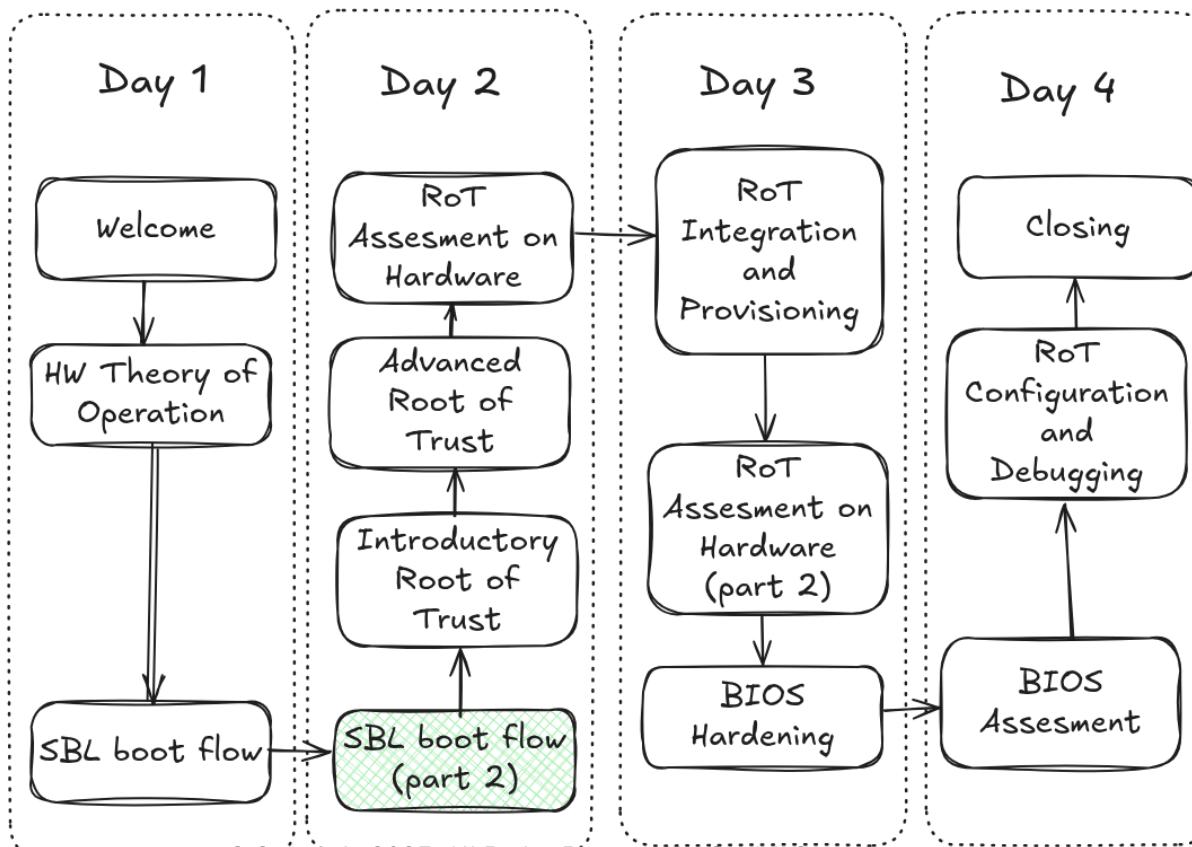
# Slim Bootloader keys production considerations

- Production ready build system should respect PKCS#11 API for HSM compatibility.
- Unfortunately [BootloaderCorePkg/Tools/SingleSign.py](#) use private key file directly.
- Production implementation would require Slim Bootloader modifications.
- It would have to change how `single_sign_file` function use private key, when `pkcs11:` prefix in `SBL_KEY_DIR` would be detected.
  - This method is defined in RFC 7512 and respected by OpenSSL.

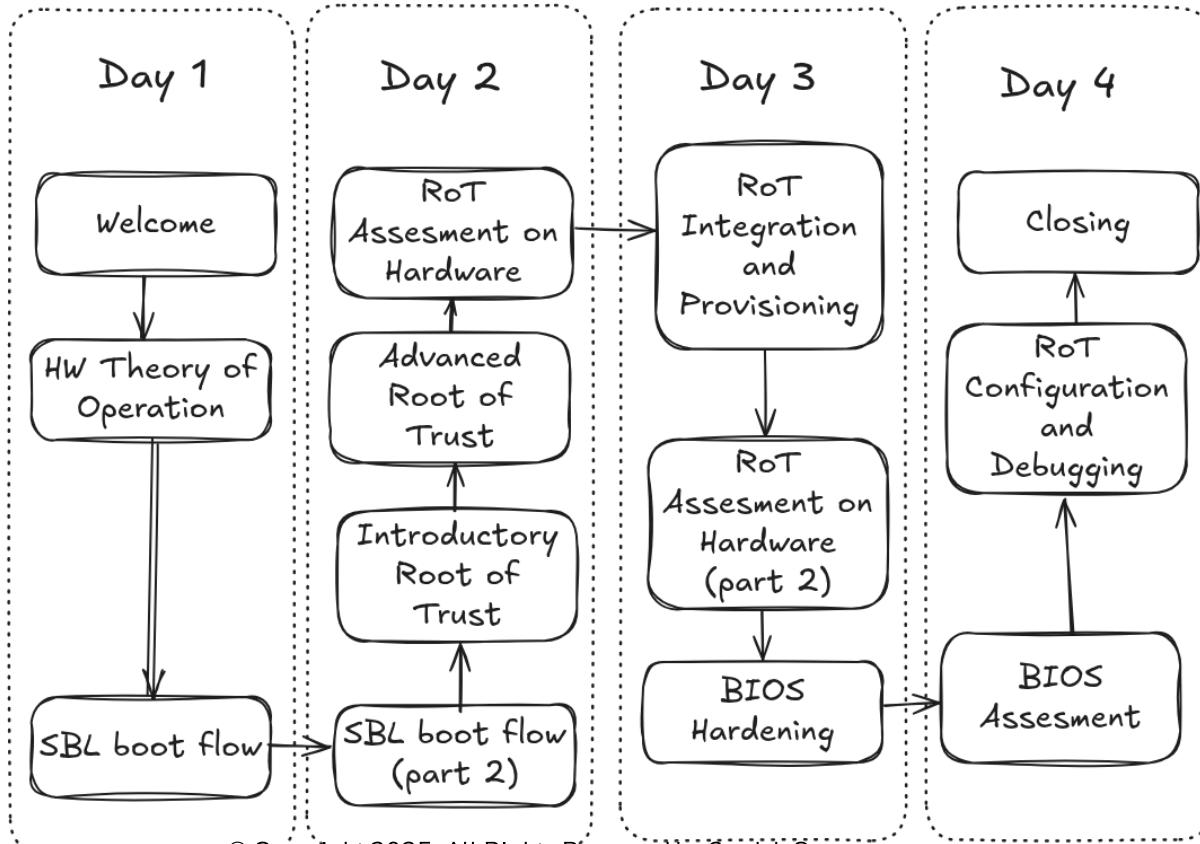
```
# --- PATCH SingleSign.py (minimal draft) -----
@@
-priv_key = os.path.join(key_dir, 'RSA3072_PRIV.pem')
+# HSM path: export SBL_HSM_URI="pkcs11:token=BuildKeys;object=SBL-RSA3072; \
+##                                     type=private;pin-value=${PIN}"
+## If that env-var is present we feed OpenSSL the URI directly.
+priv_key = os.getenv('SBL_HSM_URI') or \
+           os.path.join(key_dir, 'RSA3072_PRIV.pem')
```

- It can be even tested in our virtual/emulated environment by leveraging `softhsm2` and/or `swtpm`.

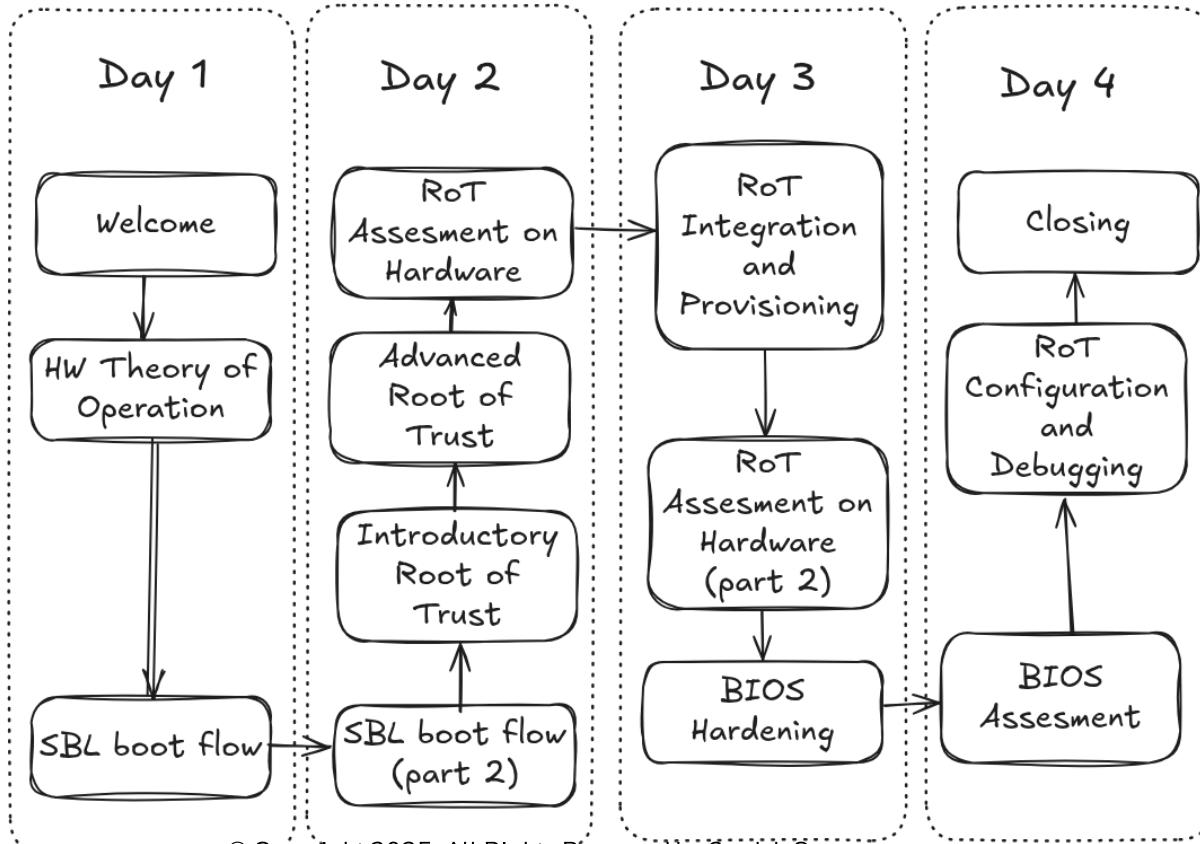
# Where we are in the course



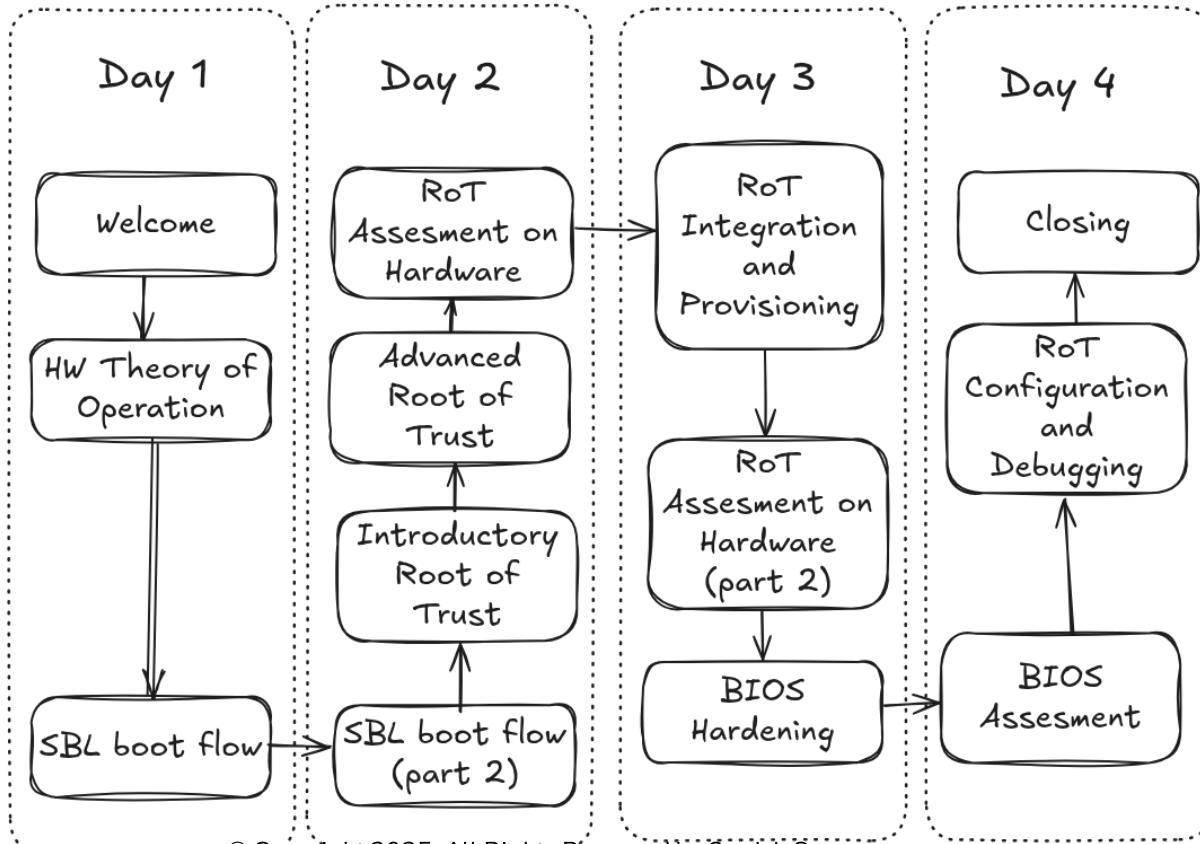
# Recap



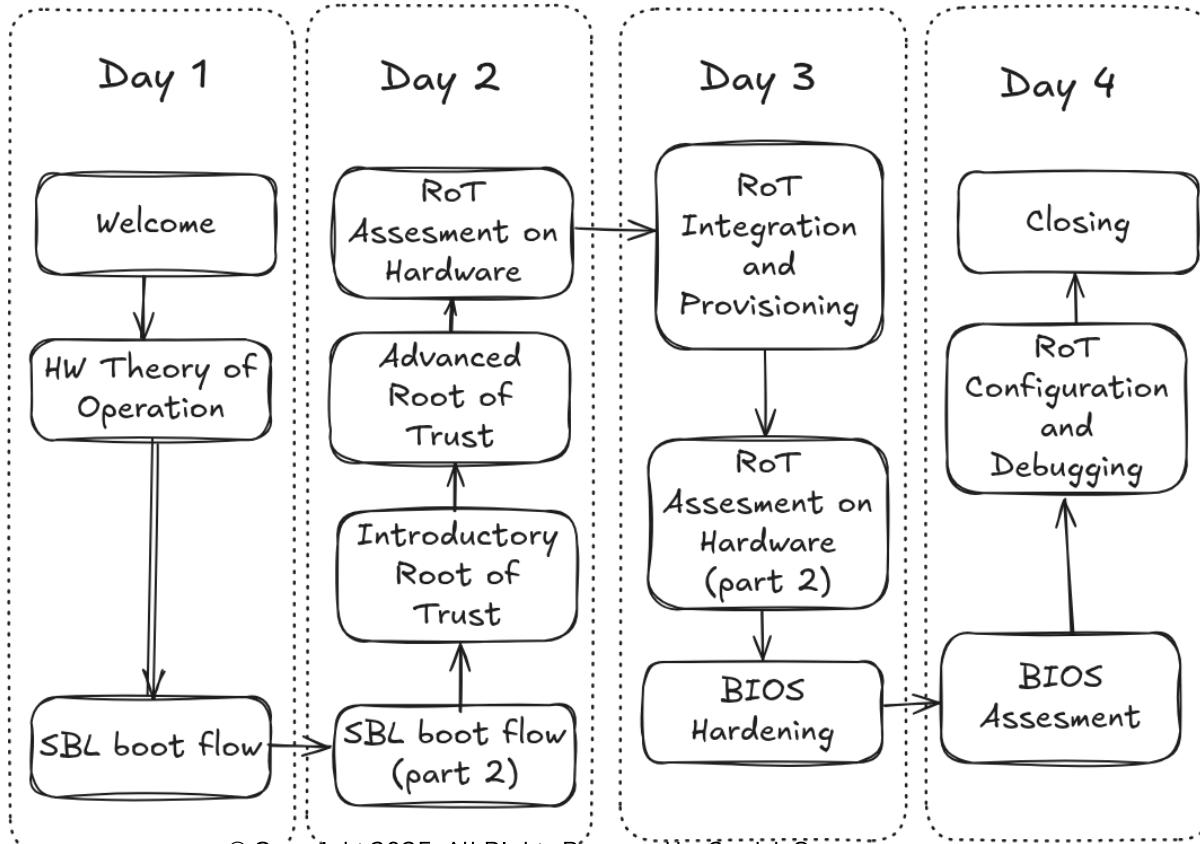
# Recap



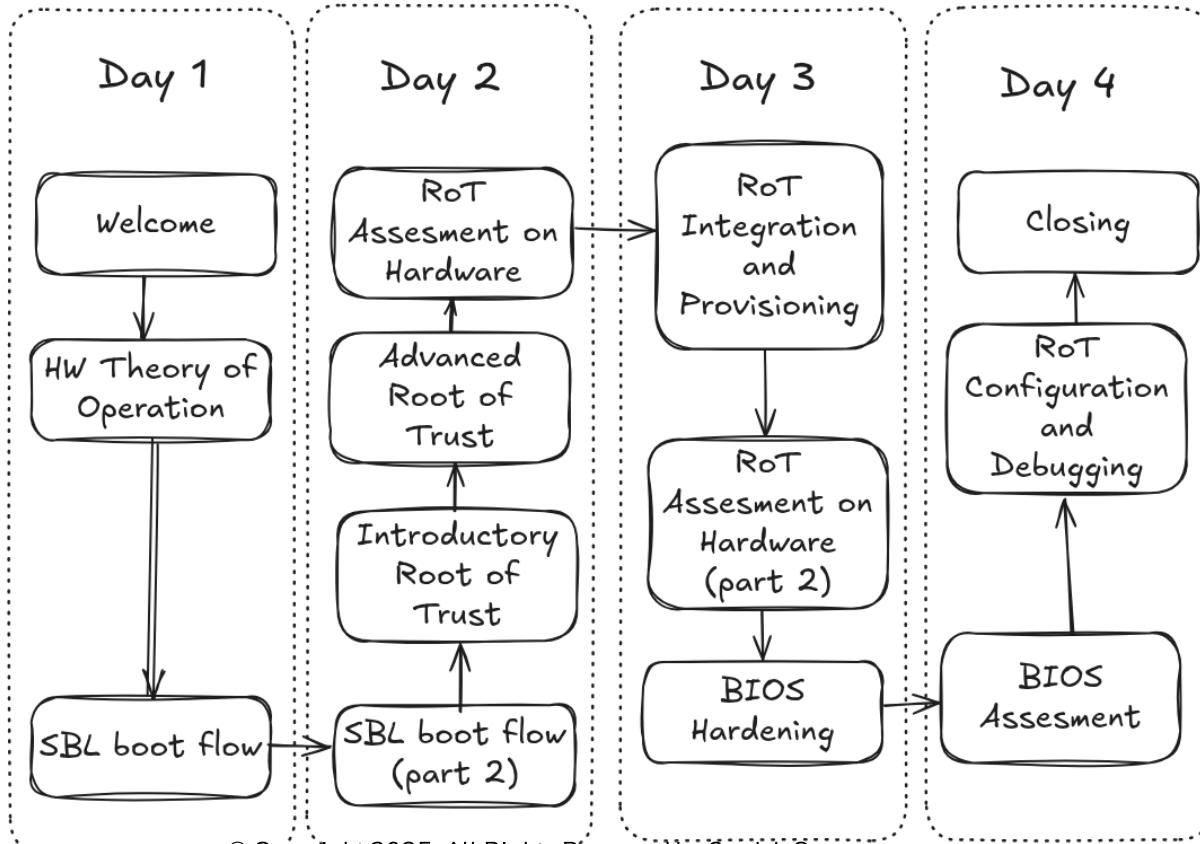
# Recap



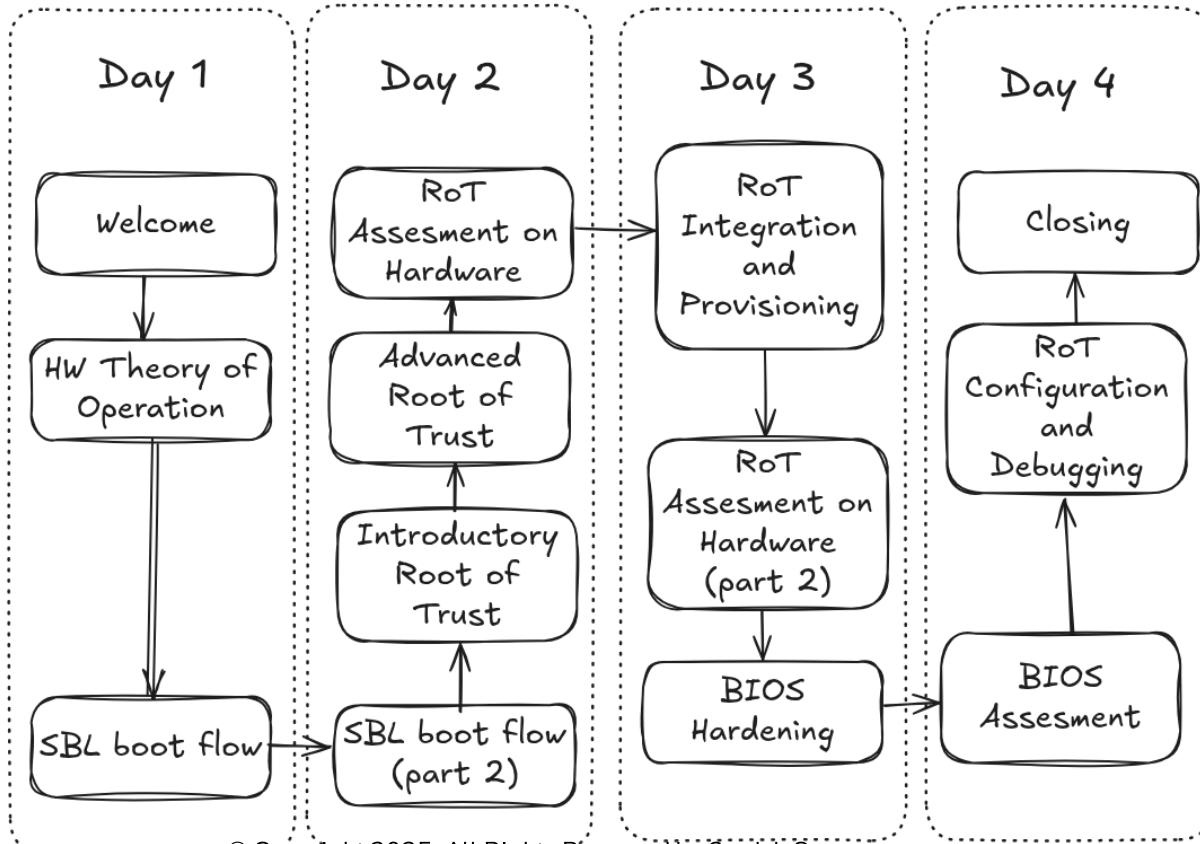
# Recap



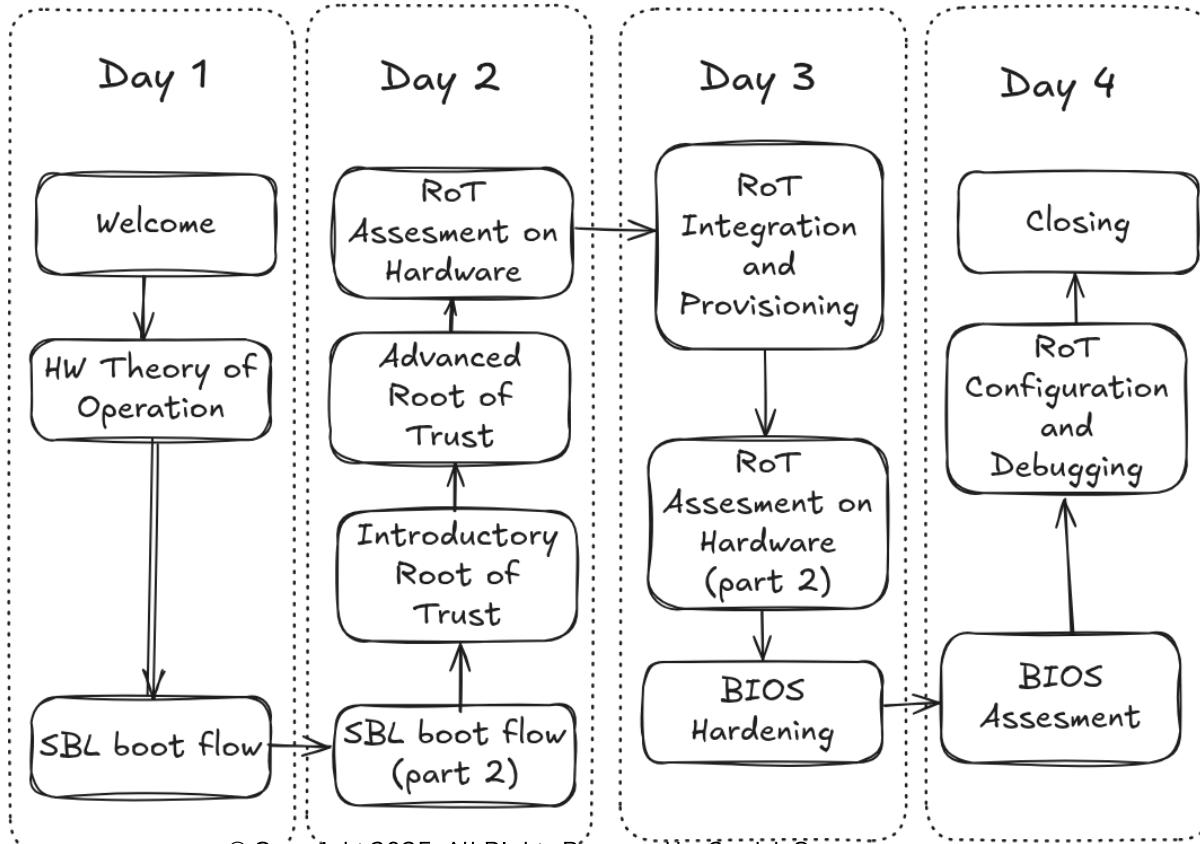
# Recap



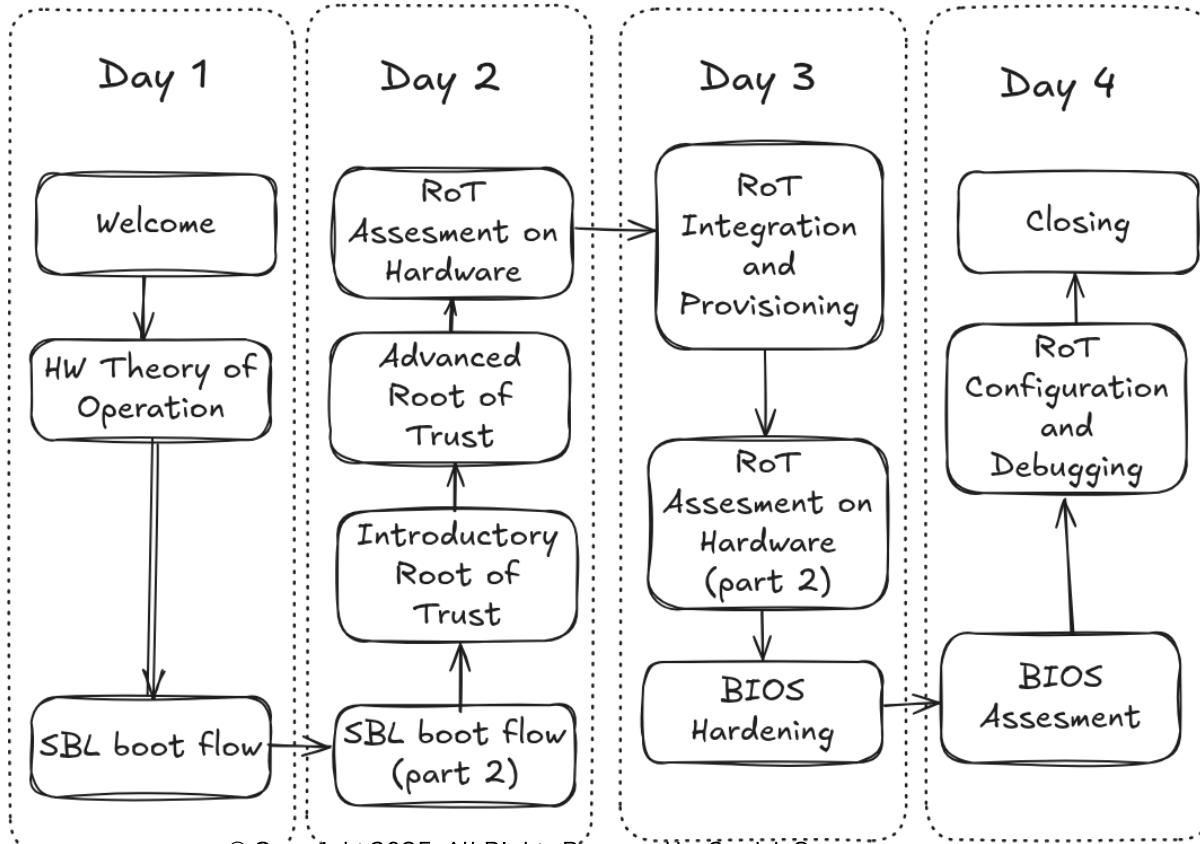
# Recap



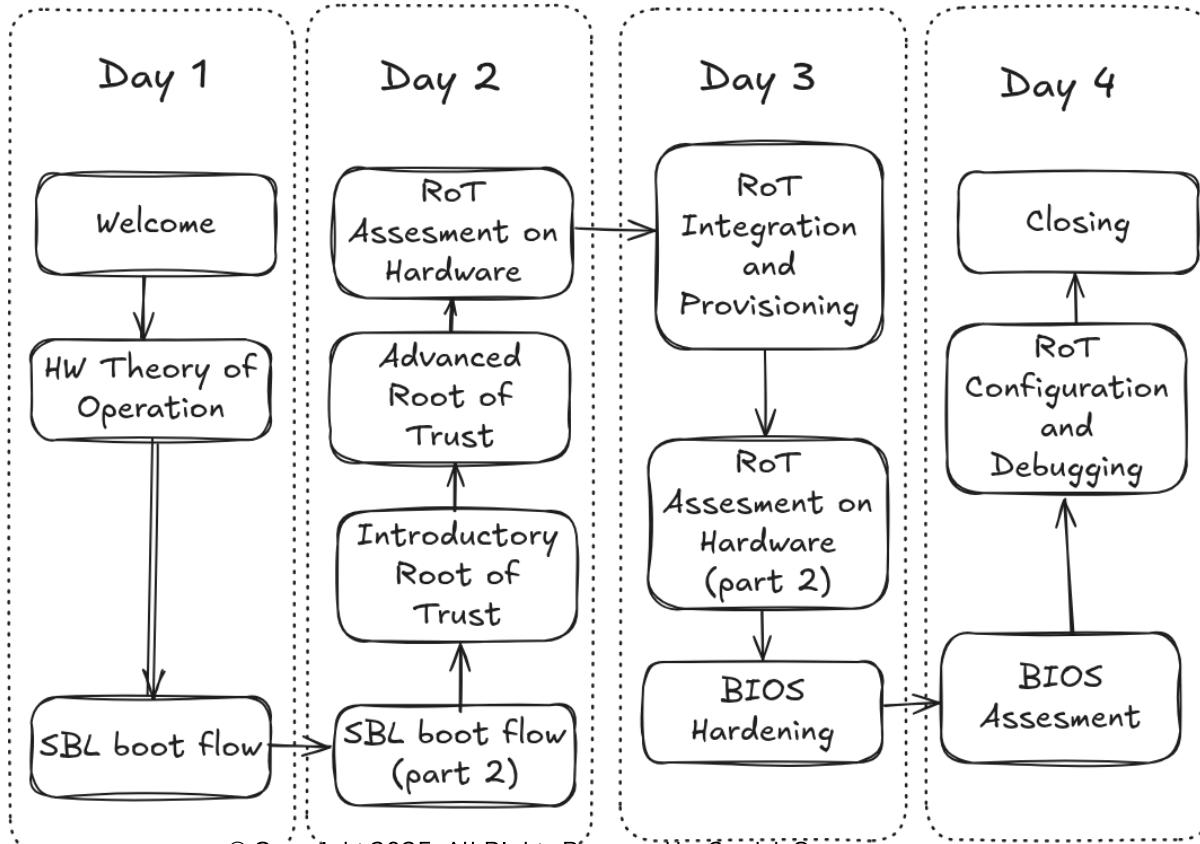
# Recap



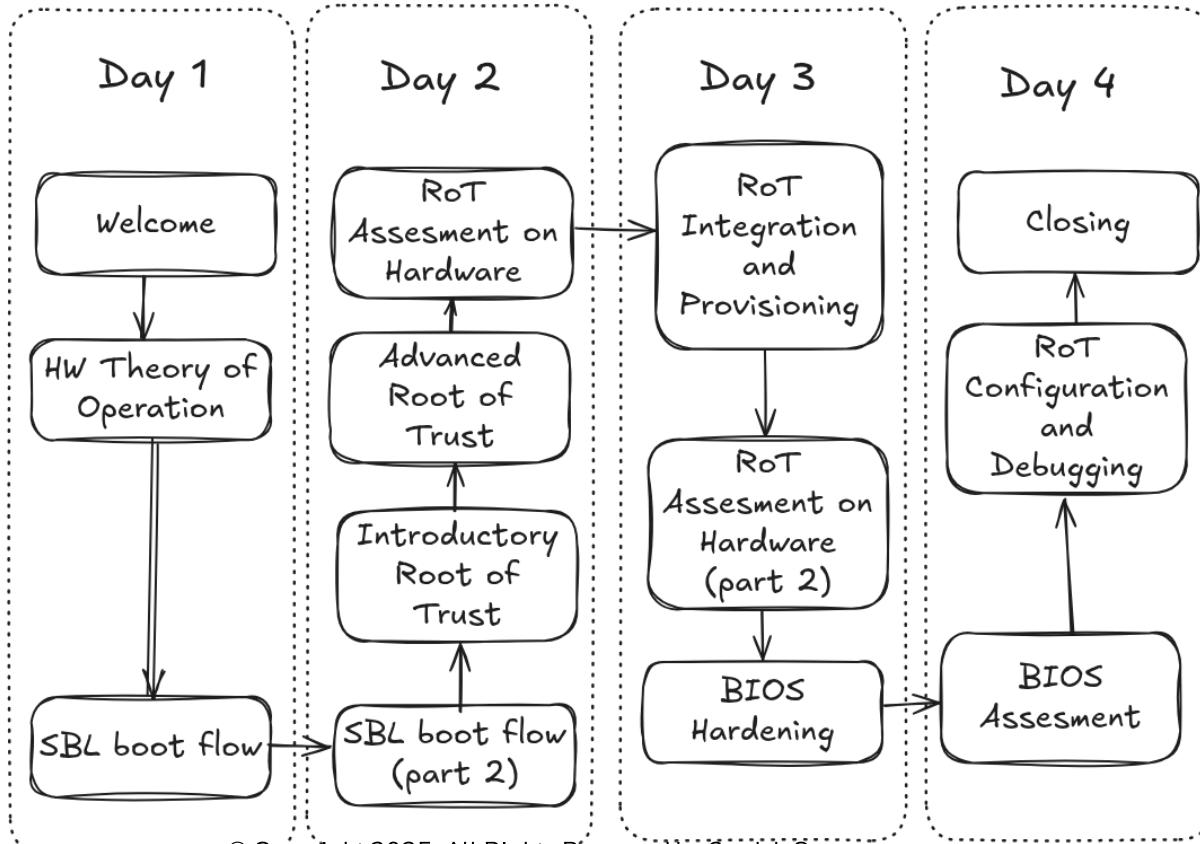
# Recap



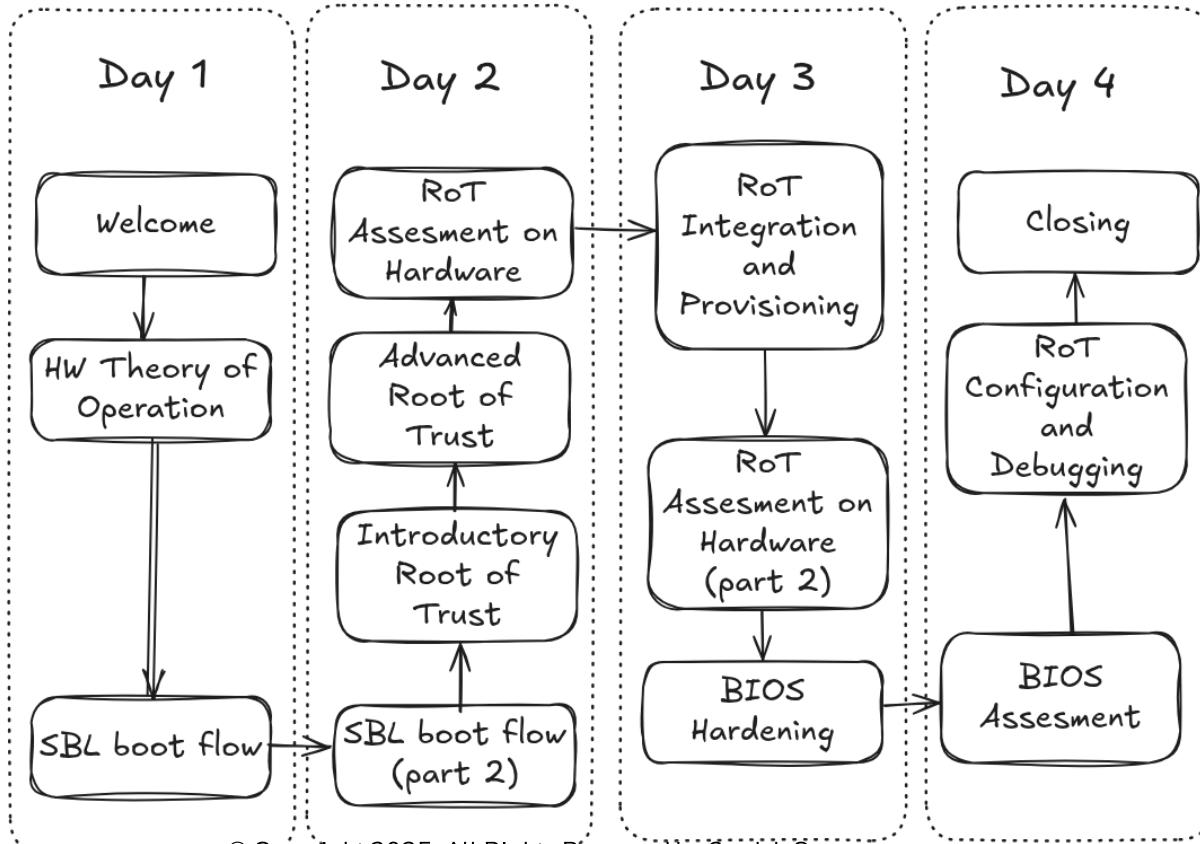
# Recap



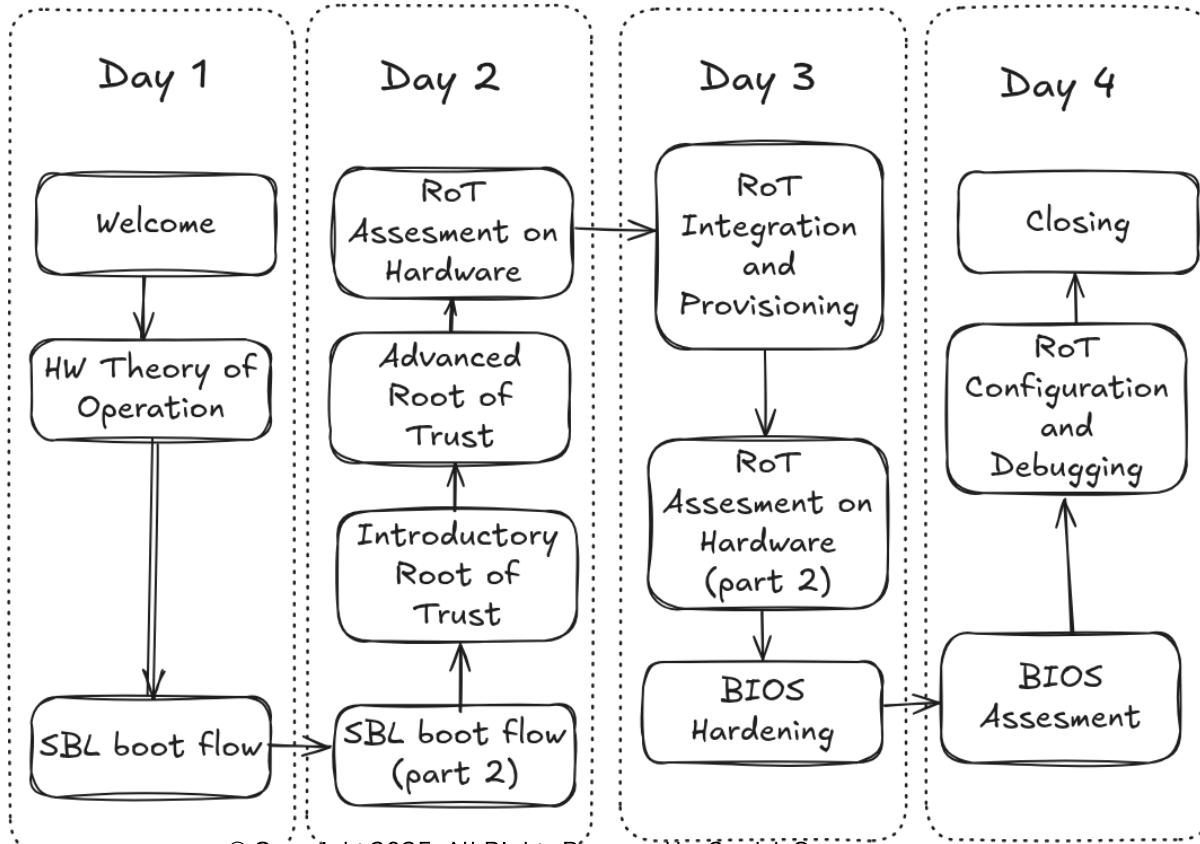
# Recap



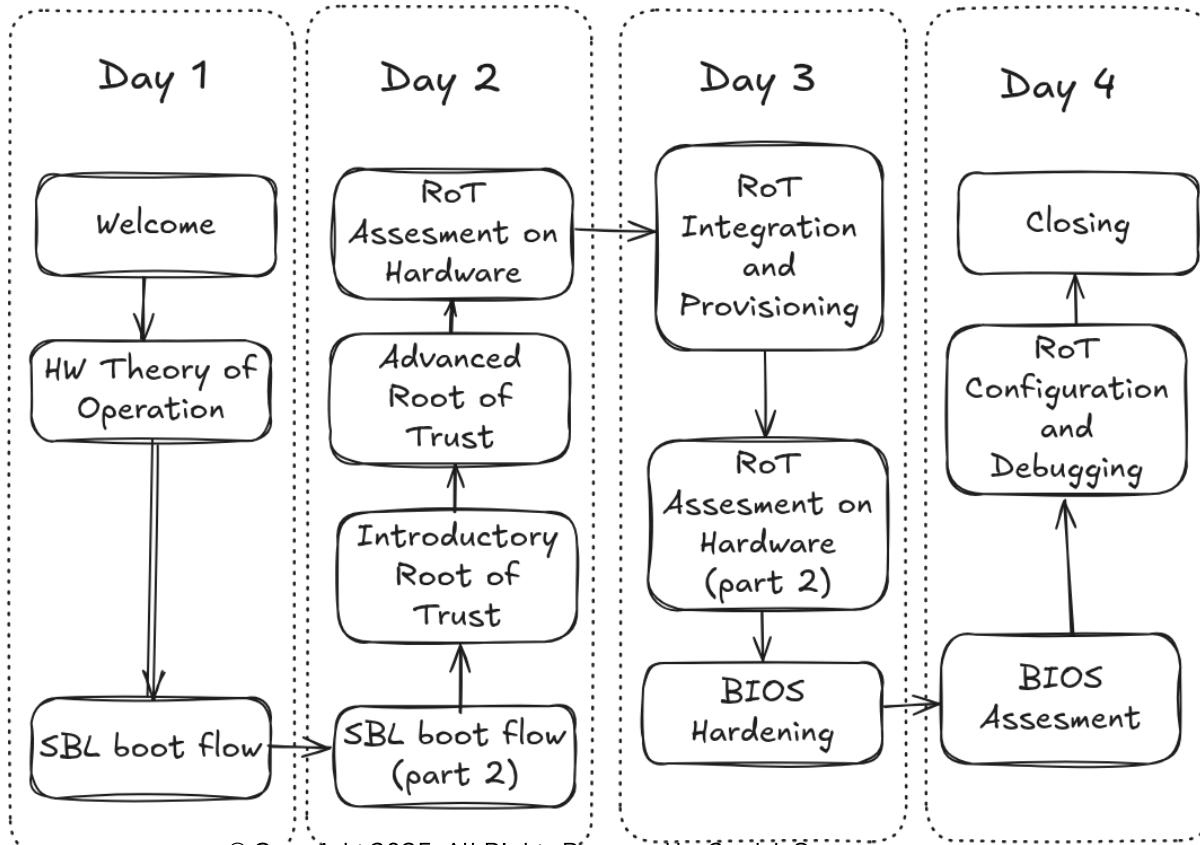
# Recap



# Recap



# Recap



```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
build_edk2() {
    local edk2_ver="$1"
    local flags="$2"
    local build_type="RELEASE"

    rm -rf edk2
    mkdir edk2
    cd edk2
    # clone one commit only
    git init
    git remote remove origin 2>/dev/null || true
    git remote add origin https://github.com/tianocore/edk2.git
    git fetch --depth 1 origin "$edk2_ver"
    git checkout FETCH_HEAD --force
    git submodule update --init --checkout --recursive --depth 1

    # Copy Dasharo logo
    cp ../../Platform/CommonBoardPkg/Logo/Logo.bmp MdeModulePkg/Logo/Logo.bmp

    if [ $DEBUG -eq 1 ]; then
        build_type="DEBUG"
    fi

    local epoch=$(git log -1 --pretty=%ct)
    ( ... )
```

```
build_edk2() {
    local edk2_ver="$1"
    local flags="$2"
    local build_type="RELEASE"

    rm -rf edk2
    mkdir edk2
    cd edk2
    # clone one commit only
    git init
    git remote remove origin 2>/dev/null || true
    git remote add origin https://github.com/tianocore/edk2.git
    git fetch --depth 1 origin "$edk2_ver"
    git checkout FETCH_HEAD --force
    git submodule update --init --checkout --recursive --depth 1

    # Copy Dasharo logo
    cp ../../Platform/CommonBoardPkg/Logo/Logo.bmp MdeModulePkg/Logo/Logo.bmp

    if [ $DEBUG -eq 1 ]; then
        build_type="DEBUG"
    fi

    local epoch=$(git log -1 --pretty=%ct)

    ( ... )
```

```
build_edk2() {
    local edk2_ver="$1"
    local flags="$2"
    local build_type="RELEASE"

    rm -rf edk2
    mkdir edk2
    cd edk2
    # clone one commit only
    git init
    git remote remove origin 2>/dev/null || true
    git remote add origin https://github.com/tianocore/edk2.git
    git fetch --depth 1 origin "$edk2_ver"
    git checkout FETCH_HEAD --force
    git submodule update --init --checkout --recursive --depth 1

    # Copy Dasharo logo
    cp ..../Platform/CommonBoardPkg/Logo/Logo.bmp MdeModulePkg/Logo/Logo.bmp

    if [ $DEBUG -eq 1 ]; then
        build_type="DEBUG"
    fi

    local epoch=$(git log -1 --pretty=%ct)

    ( ... )
```

```
build_edk2() {
    local edk2_ver="$1"
    local flags="$2"
    local build_type="RELEASE"

    rm -rf edk2
    mkdir edk2
    cd edk2
    # clone one commit only
    git init
    git remote remove origin 2>/dev/null || true
    git remote add origin https://github.com/tianocore/edk2.git
    git fetch --depth 1 origin "$edk2_ver"
    git checkout FETCH_HEAD --force
    git submodule update --init --checkout --recursive --depth 1

    # Copy Dasharo logo
    cp ../../Platform/CommonBoardPkg/Logo/Logo.bmp MdeModulePkg/Logo/Logo.bmp

    if [ $DEBUG -eq 1 ]; then
        build_type="DEBUG"
    fi

    local epoch=$(git log -1 --pretty=%ct)

    ( ... )
```

```
build_edk2() {
    local edk2_ver="$1"
    local flags="$2"
    local build_type="RELEASE"

    rm -rf edk2
    mkdir edk2
    cd edk2
    # clone one commit only
    git init
    git remote remove origin 2>/dev/null || true
    git remote add origin https://github.com/tianocore/edk2.git
    git fetch --depth 1 origin "$edk2_ver"
    git checkout FETCH_HEAD --force
    git submodule update --init --checkout --recursive --depth 1

    # Copy Dasharo logo
    cp ../../Platform/CommonBoardPkg/Logo/Logo.bmp MdeModulePkg/Logo/Logo.bmp

    if [ $DEBUG -eq 1 ]; then
        build_type="DEBUG"
    fi

    local epoch=$(git log -1 --pretty=%ct)
    ( ... )
```

```
( ... )
docker run --rm -i -u "$UID" -v "$PWD":/home/coreboot/coreboot \
-w /home/coreboot/coreboot \
-e SOURCE_DATE_EPOCH=$epoch \
$DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
source edksetup.sh
make -C BaseTools
python ./UefiPayloadPkg/UniversalPayloadBuild.py -t GCC5 -o Dasharo -b $build_type \
$flags
EOF
cd ..
}
```

```
( ... )
docker run --rm -i -u "$UID" -v "$PWD":/home/coreboot/coreboot \
-w /home/coreboot/coreboot \
-e SOURCE_DATE_EPOCH=$epoch \
$DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
source edksetup.sh
make -C BaseTools
python ./UefiPayloadPkg/UniversalPayloadBuild.py -t GCC5 -o Dasharo -b $build_type \
$flags
EOF
cd ..
}
```

```
( ... )
docker run --rm -i -u "$UID" -v "$PWD":/home/coreboot/coreboot \
-w /home/coreboot/coreboot \
-e SOURCE_DATE_EPOCH=$epoch \
$DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
source edksetup.sh
make -C BaseTools
python ./UefiPayloadPkg/UniversalPayloadBuild.py -t GCC5 -o Dasharo -b $build_type \
$flags
EOF
cd ..
}
```

```
( ... )
docker run --rm -i -u "$UID" -v "$PWD":/home/coreboot/coreboot \
-w /home/coreboot/coreboot \
-e SOURCE_DATE_EPOCH=$epoch \
$DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
source edksetup.sh
make -C BaseTools
python ./UefiPayloadPkg/UniversalPayloadBuild.py -t GCC5 -o Dasharo -b $build_type \
$flags
EOF
cd ..
}
```

```
python ./UefiPayloadPkg/UniversalPayloadBuild.py -h
usage: UniversalPayloadBuild.py [-h] [-t TOOLCHAIN] [-b TARGET] [-a {IA32,X64,RISCV64,AARCH64}]
                                 [-D MACRO] [-i IMAGEID] [-q] [-p PCD] [-s SPECREVISION]
                                 [-r REVISION] [-o PRODUCERID] [-e] [-pb PREBUILDUPLBINARY] [-sk]
                                 [-af ADDFV] [-f] [-l LOADADDRESS] [-c DSCPATH] [-ac ADD_CC_FLAGS]
                                 [-ci]
```

For building Universal Payload

options:

- h, --help show this help message and exit
- t TOOLCHAIN, --ToolChain TOOLCHAIN
- b TARGET, --Target TARGET
- a {IA32,X64,RISCV64,AARCH64}, --Arch {IA32,X64,RISCV64,AARCH64}  
Specify the ARCH for payload entry module. Default build X64 image.
- D MACRO, --Macro MACRO
- i IMAGEID, --ImageId IMAGEID  
Specify payload ID (16 bytes maximal).
- q, --Quiet Disable all build messages except FATAL ERRORS.
- p PCD, --pcd PCD
- ( ... )

```
python ./UefiPayloadPkg/UniversalPayloadBuild.py -t GCC5 -o Dasharo -b RELEASE \
$FLAGS

-t TOOLCHAIN, --ToolChain TOOLCHAIN
-o PRODUCERID, --ProducerId PRODUCERID
    A null-terminated OEM-supplied string that identifies the payload
    producer (16 bytes maximal).
-b TARGET, --Target TARGET
-D MACRO, --Macro MACRO

# EDK2_FLAGS="-D CRYPTO_PROTOCOL_SUPPORT=TRUE -D SIO_BUS_ENABLE=TRUE \
#     -D PERFORMANCE_MEASUREMENT_ENABLE=TRUE \
#     -D MULTIPLE_DEBUG_PORT_SUPPORT=TRUE -D BOOTSPLASH_IMAGE=TRUE \
#     -D BOOT_MANAGER_ESCAPE=TRUE"
```

# UEFI UniversalPayload

- Payload created using EDKII UefiPayloadPkg.
- Reference implementation of [Universal Payload Specification](#).
  - Specification goal is to describe interface between Platform Init and Payload.
  - Thanks to that interoperability payloads can be distributed in independent way from board firmware.

```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
EOF
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
EOF
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
```

EOF

© Copyright 2025. All Rights Reserved by 3mdeb Sp. z o.o.

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
EOF
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
EOF
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
EOF
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
EOF
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
EOF
```

```
build_slimbootloader() {
    local platform="$1"
    local release_build="-r"

    git submodule update --init --checkout --recursive --depth 1

    if [ $DEBUG -eq 1 ]; then
        release_build=""
    fi

    local epoch=$(git log -1 --pretty=%ct)

    mkdir -p PayloadPkg/PayloadBins/
    cp edk2/Build/UefiPayloadPkgX64/UniversalPayload.elf PayloadPkg/PayloadBins/
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \
        -v "$SBL_KEY_DIR":/home/coreboot/coreboot/SblKeys \
        -e SOURCE_DATE_EPOCH=$epoch \
        -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF
            set -e
            export SBL_KEY_DIR=/home/coreboot/coreboot/SblKeys
            export BUILD_NUMBER=0
            python BuildLoader.py clean
            python BuildLoader.py build "$platform" $release_build \
                -p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
```

EOF

© Copyright 2025. All Rights Reserved by 3mdeb Sp. z o.o.

```
python BuildLoader.py build "$platform" $release_build \
-p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
```

```
user@OST2-VM:~$ python BuildLoader.py -h
usage: BuildLoader.py [-h] {build,clean,build_dsc} ...
positional arguments:
  {build,clean,build_dsc}
    command
      build          build SBL firmware
      clean          clean build dir
      build_dsc      build a specified dsc file
options:
  -h, --help        show this help message and exit
```

```
user@OST2-VM:~$ python BuildLoader.py build -h
usage: BuildLoader.py build [-h] [-r] [-v] [-fp FSPPATH] [-fd] [-a {ia32,x64}] [-no] [-p PAYLOAD]
                            [-k] [-t TOOLCHAIN] board

positional arguments: board           Board Name (adln, adln50, adlp,
adlps, adls, azb, odroidh4, apl, arlh, arls, arlu, cfl, cml, cmlv, ehl, idv,
idvh, mtlps, mtl, qemu, qemuovrd, btls, rplps, rplp, rpls, tgl)

options:
  -h, --help            show this help message and exit
  -r, --release         Release build
  -v, --usever          Use board version file
  -fp FSPPATH           FSP binary path relative to FspBin in Silicon folder
  -fd, --fspdebug       Use debug FSP binary
  -a {ia32,x64}, --arch {ia32,x64}
                        Specify the ARCH for build. Default is to build IA32 image.
  ( ... )
  -p PAYLOAD, --payload PAYLOAD
                        Payload file name
  ( ... )
```

```
python BuildLoader.py build "$platform" $release_build \
-p "OsLoader.efi:LLDR:Lz4;UniversalPayload.elf:UEFI:Lzma"
```

"OsLoader.efi:LLDR:Lz4" -> OsLoader.efi LLDR Lz4

"UniversalPayload.elf:UEFI:Lzma" -> UniversalPayload.elf UEFI Lzma

**Payload 1:**

- Payload File: OsLoader.efi
- Payload ID: LLDR
- Compression Algorithm: Lz4

**Payload 2:**

- Payload File: UniversalPayload.elf
- Payload ID: UEFI
- Compression Algorithm: Lzma

# OsLoader

- OsLoader is a tightly-coupled payload for Slim Bootloader that natively supports the Linux Boot Protocol and is also capable of launching several different executable file formats (Multiboot ELF, ELF, PE32, UEFI-PI FV).
- If a payload is not specified in the SBL build command, OsLoader will be the default payload built with SBL.
- OsLoader expects the boot image to be packaged as an SBL Container.
- OsLoader verifies the boot image for secure boot purposes.

source: <https://slimbootloader.github.io/developer-guides/osloader.html>

```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
build_odroid_h4() {
    local blobs_rev="cbfff4d06009bc342b8638a9749fd0e286d5dcb3"

    build_edk2 "edk2-stable202505" "$EDK2_FLAGS"
    build_slimbootloader odroid_h4

    if [ -f Outputs/odroid_h4/descriptor.bin ]; then
        rm Outputs/odroid_h4/descriptor.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/descriptor.bin \
        -O Outputs/odroid_h4/descriptor.bin > /dev/null
    if [ -f Outputs/odroid_h4/me.bin ]; then
        rm Outputs/odroid_h4/me.bin
    fi
    wget https://github.com/Dasharo/dasharo-blobs/raw/$blobs_rev/hardkernel/odroid-h4/me.bin \
        -O Outputs/odroid_h4/me.bin > /dev/null
    dd if=/dev/zero of=image.bin bs=16M count=1 > /dev/null 2>&1
    cat Outputs/odroid_h4/descriptor.bin Outputs/odroid_h4/me.bin | \
        dd of=image.bin conv=notrunc > /dev/null 2>&1

    stitch_loader odroid_h4 image.bin AlderlakeBoardPkg 0xAFFFF0C
    rm image.bin

    ( ... )
}
```

```
stitch_loader() {  
    local platform="$1"      # odroid_h4  
    local ifwi_image="$2"     # image.bin  
    local platform_pkg="$3"   # AlderlakeBoardPkg  
    local platform_data="$4"  # 0xAAAAFFF0C  
  
    local out_bin="Outputs/$platform/ifwi-release.bin"  
  
    if [ $DEBUG -eq 1 ]; then  
        out_bin="Outputs/$platform/ifwi-debug.bin"  
    fi  
  
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \  
    -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF  
    python Platform/$platform_pkg/Script/StitchLoader.py \  
        -i $ifwi_image \  
        -s Outputs/$platform/SlimBootloader.bin \  
        -o $out_bin \  
        -p $platform_data  
EOF
```

```
stitch_loader() {  
    local platform="$1"      # odroid_h4  
    local ifwi_image="$2"     # image.bin  
    local platform_pkg="$3"   # AlderlakeBoardPkg  
    local platform_data="$4"  # 0xAAAAFFF0C  
  
    local out_bin="Outputs/$platform/ifwi-release.bin"  
  
    if [ $DEBUG -eq 1 ]; then  
        out_bin="Outputs/$platform/ifwi-debug.bin"  
    fi  
  
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \  
    -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF  
    python Platform/$platform_pkg/Script/StitchLoader.py \  
        -i $ifwi_image \  
        -s Outputs/$platform/SlimBootloader.bin \  
        -o $out_bin \  
        -p $platform_data  
EOF
```

```
stitch_loader() {  
    local platform="$1"      # odroid_h4  
    local ifwi_image="$2"     # image.bin  
    local platform_pkg="$3"   # AlderlakeBoardPkg  
    local platform_data="$4"  # 0xAAAAFFF0C  
  
    local out_bin="Outputs/$platform/ifwi-release.bin"  
  
    if [ $DEBUG -eq 1 ]; then  
        out_bin="Outputs/$platform/ifwi-debug.bin"  
    fi  
  
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \  
    -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF  
    python Platform/$platform_pkg/Script/StitchLoader.py \  
        -i $ifwi_image \  
        -s Outputs/$platform/SlimBootloader.bin \  
        -o $out_bin \  
        -p $platform_data  
EOF
```

```
stitch_loader() {  
    local platform="$1"      # odroid_h4  
    local ifwi_image="$2"     # image.bin  
    local platform_pkg="$3"   # AlderlakeBoardPkg  
    local platform_data="$4"  # 0xAAAAFFF0C  
  
    local out_bin="Outputs/$platform/ifwi-release.bin"  
  
    if [ $DEBUG -eq 1 ]; then  
        out_bin="Outputs/$platform/ifwi-debug.bin"  
    fi  
  
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \  
    -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF  
    python Platform/$platform_pkg/Script/StitchLoader.py \  
        -i $ifwi_image \  
        -s Outputs/$platform/SlimBootloader.bin \  
        -o $out_bin \  
        -p $platform_data  
EOF
```

```
stitch_loader() {  
    local platform="$1"      # odroid_h4  
    local ifwi_image="$2"     # image.bin  
    local platform_pkg="$3"   # AlderlakeBoardPkg  
    local platform_data="$4"  # 0xAAAAFFF0C  
  
    local out_bin="Outputs/$platform/ifwi-release.bin"  
  
    if [ $DEBUG -eq 1 ]; then  
        out_bin="Outputs/$platform/ifwi-debug.bin"  
    fi  
  
    docker run --rm -i -u $UID -v "$PWD":/home/coreboot/coreboot \  
    -w /home/coreboot/coreboot $DOCKER_IMAGE:$DOCKER_IMAGE_VER /bin/bash <<EOF  
    python Platform/$platform_pkg/Script/StitchLoader.py \  
        -i $ifwi_image \  
        -s Outputs/$platform/SlimBootloader.bin \  
        -o $out_bin \  
        -p $platform_data  
EOF
```

```
python Platform/AlderlakeBoardPkg/Script/StitchLoader.py -h
usage: StitchLoader.py [-h] -i IFWI_IN [-o IFWI_OUT] [-s SBL_IN] [-p PLAT_DATA]

options:
-h, --help            show this help message and exit
-i IFWI_IN, --input-ifwi-file IFWI_IN
                      Specify input template IFWI image file path
-o IFWI_OUT, --output-ifwi-file IFWI_OUT
                      Specify generated output IFWI image file path
-s SBL_IN, --sbl-input SBL_IN
                      Specify input sbl binary file path
-p PLAT_DATA, --platform-data PLAT_DATA
                      Specify a platform specific data (HEX, DWORD) for customization
```

```
python Platform/AlderlakeBoardPkg/Script/StitchLoader.py \
-i image.bin \
-s Outputs/odroid_h4/SlimBootloader.bin \
-o ifwi-{debug,release}.bin \
-p 0xAAAAFFF0C
```

`0xAFFFF0C -> 0xAA 0xFF 0xFF 0x0C`

```
typedef struct {
    UINT8          PlatformId : 5; // 0x0C
    UINT8          Reserved1 : 3; // 0x00
    UINT8          DebugUart;    // 0xFF
    UINT8          Reserved3;    // 0xFF
    UINT8          Marker;       // 0xAA
} STITCH_DATA;
```

# Slim Bootloader for Hardkernel Odroid-H4

## ■ Building

```
user@OST2-VM:~$ cd ~/training_materials/src/slimbootloader  
user@OST2-VM:~$ ./build.sh odroid_h4
```

Depending on the machine build takes 5-10min.

Output should looks as follows:

```
Done [odroidh4] !  
Platform data was patched for IFWI/BIOS/TS0/SG1A  
Platform data was patched for IFWI/BIOS/TS1/SG1A  
Creating IFWI image ...  
done!  
Result binary placed in /home/user/training_materials/src/slimbootloader/Outputs/odroidh4/ifwi-release.bin
```

SHA256:

```
7d9ecf7f7eb7ed56ce4f32116fa2761304cc5e0488538d3190a770c76223a6cb Outputs/odroid_h4/ifwi-release.bin
```

# Slim Bootloader boot flow on Intel N97 (ADL-N)

Practice #101 Exercise #4

Let's identify all those Slim Bootloadere boot Stages the debug log.

## Initial deployment

- Let's place Dasharo (Slim Bootloader + UEFI) on our DTS USB.

```
user@OST2-VM:~$ sudo mount /dev/sdb2 /media/
user@OST2-VM:~$ sudo cp Outputs/odroidh4/ifwi-release.bin /media/root/
user@OST2-VM:~$ sync
user@OST2-VM:~$ sudo umount /media
```

- Connect USB to Odroid-H4 and boot DTS:

```
bash-5.2# cd /root
bash-5.2# flashrom -p internal -w ifwi-release.bin
```

Output should look like this:

```
( ... )
Found Programmer flash chip "Opaque flash chip" (16384 kB, Programmer-specific) on internal.
Reading old flash chip contents... done.
Erasing and writing flash chip... Erase/write done.
Verifying flash... VERIFIED.    © Copyright 2025. All Rights Reserved by 3mdeb Sp. z o.o.
```

```
Intel Slim Bootloader
SBID: SB_ADLN
ISVN: 001
IVER: 001.000.001.000.00022
BOOT: BP0
MODE: 0
BoardID: 0x0C
Memory Init
Silicon Init
FspGfxHob is not available
MP Init
PCI Enum
FSP Requested Reboot ...
Intel Slim Bootloader
SBID: SB_ADLN
ISVN: 001
IVER: 001.000.001.000.00022
BOOT: BP0
MODE: 1
BoardID: 0x0C
Memory Init
Silicon Init
FspGfxHob is not available
MP Init
PCI Enum
ACPI Init
Jump to payload
```

3h0  
Esc or Down to enter Boot Manager Menu.  
ENTER to boot directly.

```
error: no such device: ((hd0,msdos1)/EFI/BOOT)/EFI/BOOT/
grub.cfg.
Booting `boot'
```

Let's note `dmidecode -t` output:

```
bash-5.2# dmidecode -t bios
# dmidecode 3.5
Getting SMBIOS data from sysfs.
SMBIOS 3.3.0 present.

Handle 0x0000, DMI type 0, 26 bytes
BIOS Information
    Vendor: 3mdeb
    Version: Dasharo (Slim Bootloader+UEFI) v0.9.0
    Release Date: Aug 5 2025
```

## Debug build

- Let's build debug version of Dasharo (Slim Bootloader + UEFI)

```
user@OST2-VM:~$ DEBUG=1 ./build.sh odroid_h4
```

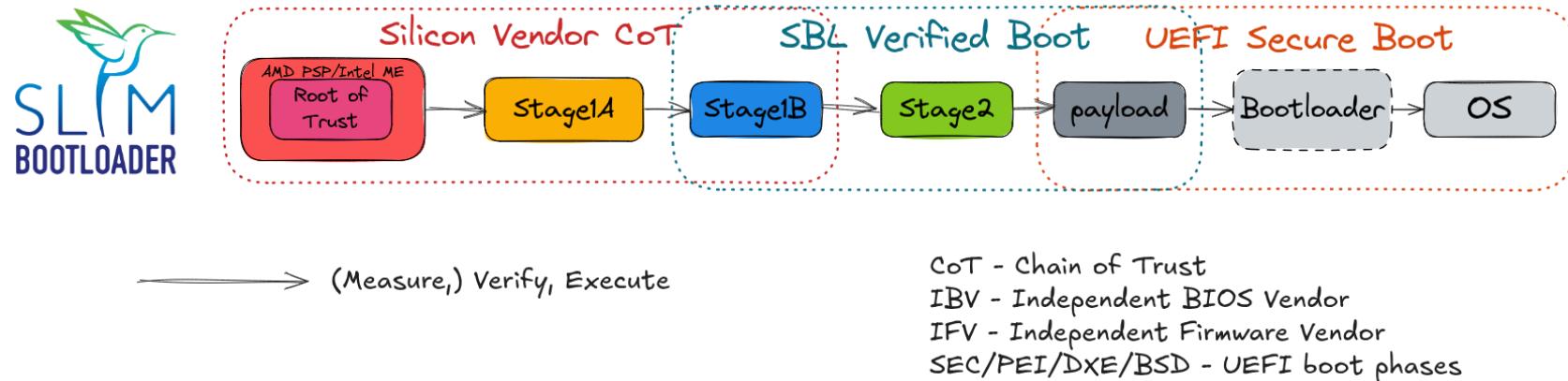
- We repeat steps for flashing and boot with that difference we change the SPI slots. This is because Dasharo (Slim Bootloader + UEFI) set the locks on SPI. We will discuss details of how that SPI protection works during further sections of the training.
- Our `odroid-serial.log` should contain following:

```
user@OST2-VM:~$ grep "Intel Slim Bootloader STAGE" odroid-serial.log
===== Intel Slim Bootloader STAGE1A =====
===== Intel Slim Bootloader STAGE1B =====
===== Intel Slim Bootloader STAGE2 =====
```

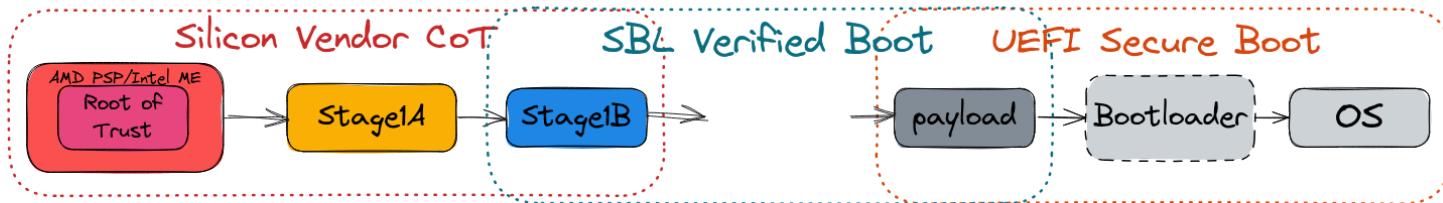
# Chain of Trust

Practice #101 Exercise #5

Critical chain of trust technology for Slim Bootloader is Verified Boot.



To prove the *Chain of Trust* works we will modify Stage2 and check how Slim Bootloader will behave with Verified Boot enabled and disabled.



→ (Measure,) Verify, Execute

CoT - Chain of Trust

IBV - Independent BIOS Vendor

IFV - Independent Firmware Vendor

SEC/PEI/DXE/BS - UEFI boot phases

# BoardConfig logic

- If we look into `Platforms` subdirectories we will find multiple files determining configuration:

```
user@OST2-VM:$ tree Platform/AlderlakeBoardPkg|grep BoardConfig
├── BoardConfigAdlN50.py
├── BoardConfigAdlN.py
├── BoardConfigAdlP.py
├── BoardConfigAdlPs.py
├── BoardConfigAdlS.py
├── BoardConfigAzb.py
└── BoardConfigOdroidH4.py
    └── BoardConfig.py
```

- Logic of loading those is defined in `BuildLoader.py`.
- It essentially pick whatever was passed as `board` parameter.

```
usage: BuildLoader.py build [-h] [-r] [-v] [-fp FSPPATH] [-fd] [-a {ia32,x64}] [-no] [-p PAYLOAD] [-k]
                           [-t TOOLCHAIN] board
```

- `board` name have to be defined in `BoardConfig*.py` :
- We use `odroid_h4` , so:

```
user@OST2-VM:$ grep odroid_h4 Platform/ -r
Platform/AlderlakeBoardPkg/BoardConfigOdroidH4.py:           self.BOARD_NAME          = 'odroid_h4'
```

- In `Platform/AlderlakeBoardPkg/BoardConfigOdroidH4.py` :

```
import BoardConfig as AlderlakeBoardConfig

class Board(AlderlakeBoardConfig.Board):
    def __init__(self, *args, **kwargs):
        super(Board, self).__init__(*args, **kwargs)

        self.VERINFO_IMAGE_ID      = 'SB_ADLN'
        self.VERINFO_PROJ_MAJOR_VER = 0
        self.VERINFO_PROJ_MINOR_VER = 9
        self.VERINFO_PROJ_PATCH_VER = 0
        self.BOARD_NAME            = 'odroid_h4'
```

## SBL Verified Boot state confirmation in configuration.

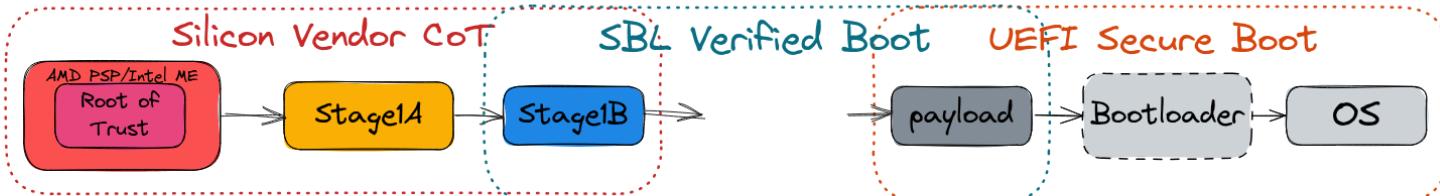
```
user@OST2-VM:$ grep VERIFIED_BOOT Platform/AlderlakeBoardPkg/BoardConfigOdroidH4.py -B6
    self.FLASH_LAYOUT_START      = 0x100000000 # 4GB Top
    self.FLASH_BASE_SIZE        = 0x01000000 # 16MB
    self.FLASH_BASE_ADDRESS     = (self.FLASH_LAYOUT_START - self.FLASH_BASE_SIZE)
    self.LOADER_ACPI_RECLAIM_MEM_SIZE = 0x000090000
    self.HAVE_FIT_TABLE         = 1
    self.HAVE_VBT_BIN           = 1
    self.HAVE_VERIFIED_BOOT     = 1
-- 
    self.ENABLE_FAST_BOOT = 0
    if self.ENABLE_FAST_BOOT:
        self.ENABLE_SPLASH          = 0
        self.ENABLE_FRAMEBUFFER_INIT = 0
        self.RELEASE_MODE           = 1
        self.HAVE_VERIFIED_BOOT     = 0
        self.HAVE_MEASURED_BOOT     = 0
        self.VERIFIED_BOOT_HASH_MASK = 0
```

Let's look for Stage2:

Flash Map Information:

| FLASH MAP<br>(RomSize = 0x00A00000) |                        |          |                    |
|-------------------------------------|------------------------|----------|--------------------|
| NAME                                | OFFSET (BASE)          | SIZE     | FLAGS              |
| TOP SWAP A                          |                        |          |                    |
| SG1A                                | 0x9e5000(0xFFFFE5000)  | 0x01b000 | Uncompressed, TS_A |
| ACM0                                | 0x980000(0xFFFF80000)  | 0x065000 | Uncompressed, TS_A |
| TOP SWAP B                          |                        |          |                    |
| SG1A                                | 0x965000(0xFFFF65000)  | 0x01b000 | Uncompressed, TS_B |
| ACM0                                | 0x900000(0xFFFF00000)  | 0x065000 | Uncompressed, TS_B |
| REDUNDANT A                         |                        |          |                    |
| SG1B                                | 0x800000(0xFFE00000)   | 0x100000 | Uncompressed, R_A  |
| KEYH                                | 0x7ff000(0xFFFFDFF000) | 0x001000 | Uncompressed, R_A  |
| CNFG                                | 0x7fb000(0xFFFFDFB000) | 0x004000 | Uncompressed, R_A  |
| FWUP                                | 0x7db000(0xFFFFDDB000) | 0x020000 | Compressed , R_A   |
| SG02                                | 0x719000(0xFFFFD19000) | 0x0c2000 | Compressed , R_A   |

```
user@OST2-VM:$ python BootloaderCorePkg/Tools/IfwiUtility.py view -i Outputs/odroid_h4/ifwi-debug.bin
IFWI [0:0x00000000 L:0x01000000]
    DESCRIPTOR [0:0x00000000 L:0x00001000]
    TXE [0:0x00001000 L:0x00413000]
    BIOS [0:0x00600000 L:0x00A00000] ← BIOS starts here
        RGN [0:0x00600000 L:0x00019000]
        EMTY [0:0x00600000 L:0x00019000]
        NVS [0:0x00619000 L:0x00001000]
        RSVD [0:0x00619000 L:0x00001000]
        NRD [0:0x0061A000 L:0x00340000]
        EMTY [0:0x0061A000 L:0x0008D000]
        IPFW [0:0x006A7000 L:0x00001000]
        VARS [0:0x006A8000 L:0x00002000]
        MRCD [0:0x006AA000 L:0x00010000]
        EPLD [0:0x006BA000 L:0x00230000]
        PYLD [0:0x008EA000 L:0x00030000]
        UVAR [0:0x0091A000 L:0x00040000]
    RD1 [0:0x0095A000 L:0x002D3000]
    UCOD [0:0x0095A000 L:0x000EC000]
    SG02 [0:0x00A46000 L:0x000C2000]
    FWUP [0:0x00B08000 L:0x00020000]
    CNFG [0:0x00B28000 L:0x00004000]
    KEYH [0:0x00B2C000 L:0x00001000]
    SG1B [0:0x00B2D000 L:0x00100000]
```



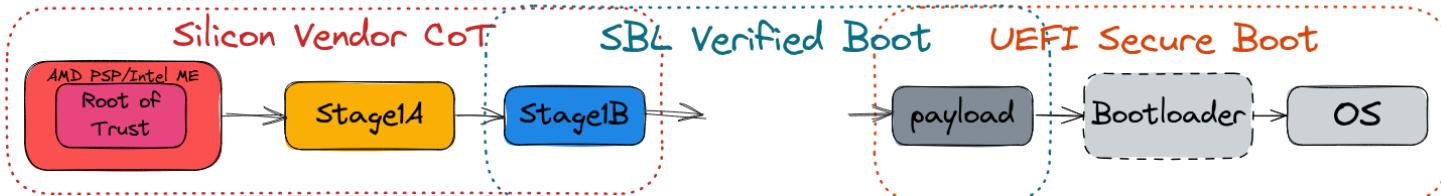
→ (Measure,) Verify, Execute

CoT - Chain of Trust

IBV - Independent BIOS Vendor

IFV - Independent Firmware Vendor

SEC/PEI/DXE/BS - UEFI boot phases



→ (Measure,) Verify, Execute

CoT - Chain of Trust

IBV - Independent BIOS Vendor

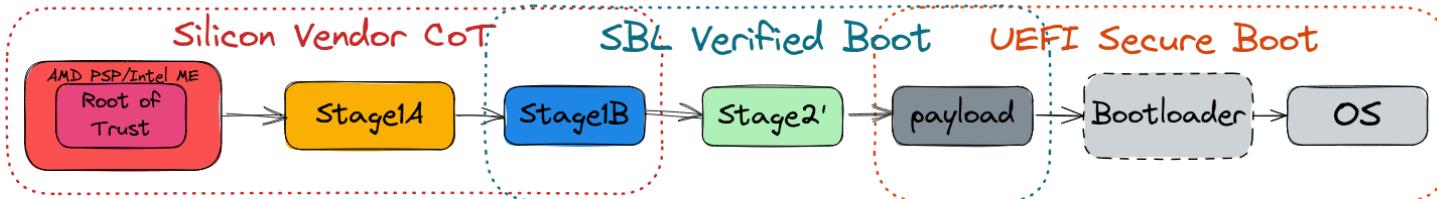
IFV - Independent Firmware Vendor

SEC/PEI/DXE/BS - UEFI boot phases

- First make copy on which we will make mods:

```
cp Outputs/odroid_h4/ifwi-debug.bin Outputs/odroid_h4/ifwi-debug.bin.vb.mod
```

- Use `hexedit` to change byte at offset 0xD19025 to 0x00.
- Put binary on USB and flash it to the hardware.



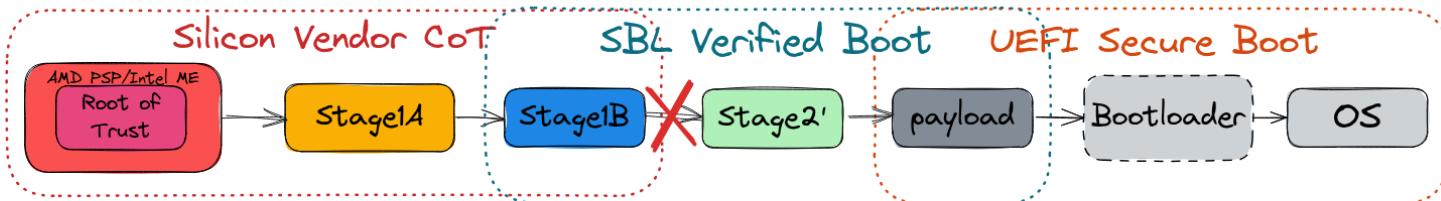
→ (Measure,) Verify, Execute

CoT - Chain of Trust

IBV - Independent BIOS Vendor

IFV - Independent Firmware Vendor

SEC/PEI/DXE/BS - UEFI boot phases



→ (Measure,) Verify, Execute

CoT - Chain of Trust

IBV - Independent BIOS Vendor

IFV - Independent Firmware Vendor

SEC/PEI/DXE/BS - UEFI boot phases

HASH verification for usage (0x00000002) with Hash Alg (0x2): Security Violation

First 48 Bytes Input Data

```
00000000: 4C 5A 34 20 5C FE 07 00-00 00 1F 00 00 00 00 00 *LZ4 \.....*
00000010: 00 00 1F 00 93 A0 05 02-01 70 00 00 01 00 01 00 *.....p....*
00000020: F1 04 78 E5 8C 00 3D 8A-1C 4F 99 35 89 61 85 C3 *..x ... =..0.5.a..*
```

Last 48 Bytes Input Data

```
00000000: FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF *.....*
00000010: FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF *.....*
00000020: FF FF FF FF FF FF FF-FF 74 50 FF FF FF FF FF *.....tP....*
```

Image Digest

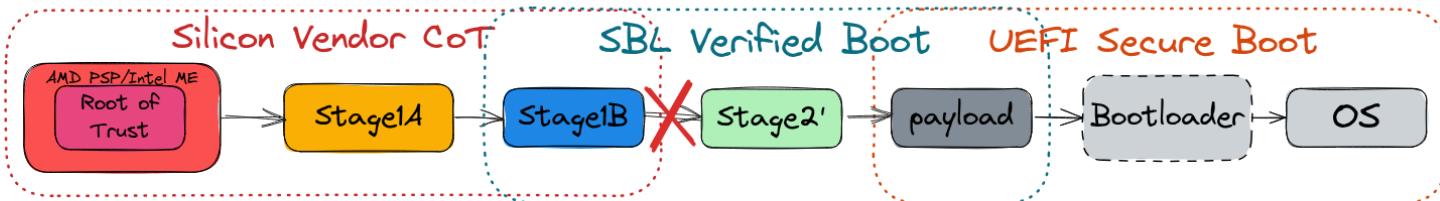
```
00000000: 56 39 A7 72 E0 39 49 39-72 95 E2 15 BB E4 16 C4 *V9.r.9I9r.....*
00000010: FB F8 71 3E BF 7B 15 29-04 F0 D9 D1 F5 3A E3 F5 *..q> ... ).....: ..*
00000020: 2B 61 B7 0B FC 1F DA 8E-45 04 94 A1 A5 C5 2D 55 *+a.....E.....-U*
```

HashStore Digest

Loading Stage2 error - Security Violation !

Failed to load Stage2!

STAGE\_1B: System halted!

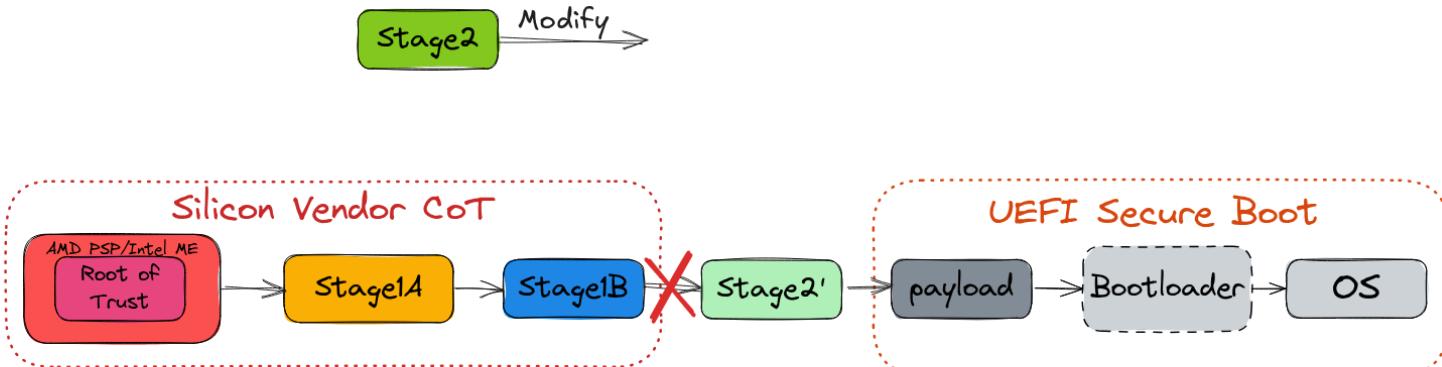


CoT - Chain of Trust

IBV - Independent BIOS Vendor

IFV - Independent Firmware Vendor

SEC/PEI/DXE/BS - UEFI boot phases



→ (Measure,) Verify, Execute

CoT - Chain of Trust

IBV - Independent BIOS Vendor

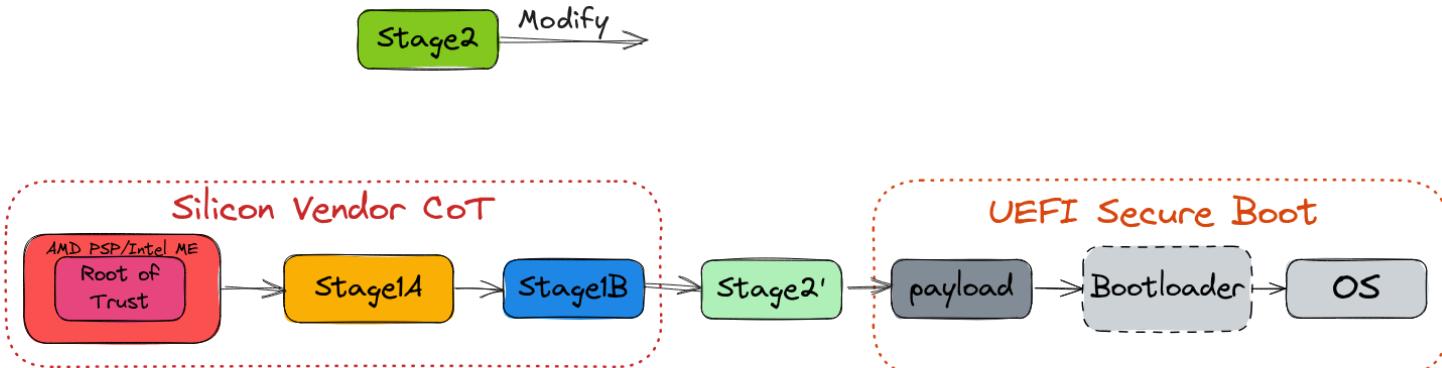
IFV - Independent Firmware Vendor

SEC/PEI/DXE/BS - UEFI boot phases

- Change Platform/AlderlakeBoardPkg/BoardConfigOdroidH4.py

```
diff --git a/Platform/AlderlakeBoardPkg/BoardConfigOdroidH4.py b/Platform/AlderlakeBoardPkg/BoardConfigOdroidH4.py
index 5da21d81..2eb41a82 100644
--- a/Platform/AlderlakeBoardPkg/BoardConfigOdroidH4.py
+++ b/Platform/AlderlakeBoardPkg/BoardConfigOdroidH4.py
@@ -47,8 +47,8 @@ class Board(AlderlakeBoardConfig.Board):
    self.LOADER_ACPI_RECLAIM_MEM_SIZE = 0x000090000
    self.HAVE_FIT_TABLE        = 1
    self.HAVE_VBT_BIN          = 1
-   self.HAVE_VERIFIED_BOOT   = 1
-   self.HAVE_MEASURED_BOOT   = 1
+   self.HAVE_VERIFIED_BOOT   = 0
+   self.HAVE_MEASURED_BOOT   = 0
    self.HAVE_FLASH_MAP        = 1
    self.HAVE_ACPI_TABLE       = 1
    self.HAVE_PSD_TABLE        = 1
```

- Build and call file ifwi-debug.bin.novb .
- Apply modification of the same byte to the image ifwi-debug.bin.novb.mod .
- Put on USB, flash and prove its booting.



→ (Measure,) Verify, Execute

CoT - Chain of Trust

IBV - Independent BIOS Vendor

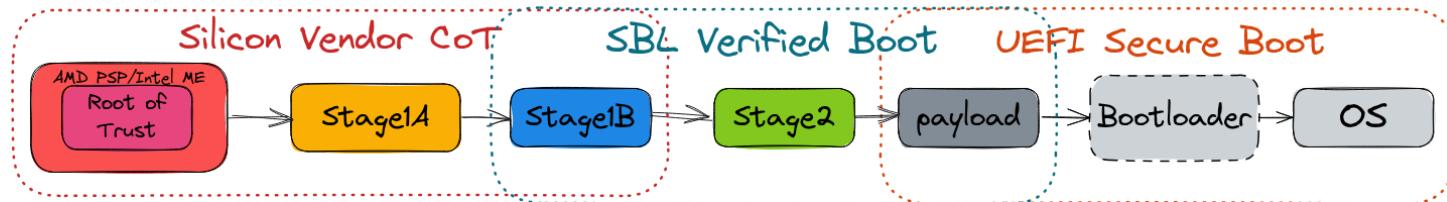
IFV - Independent Firmware Vendor

SEC/PEI/DXE/BS - UEFI boot phases

```
Dasharo Tools Suite Script 2.5.0
(c) Dasharo <contact@dasharo.com>
Report issues at: https://github.com/Dasharo/dasharo-issues
*****
**          HARDWARE INFORMATION
*****
**      System Inf.: HARDKERNEL ODROID-H4
**  Baseboard Inf.: HARDKERNEL ODROID-H4
**      CPU Inf.: Intel(R) N97
*****
**          FIRMWARE INFORMATION
*****
**  BIOS Inf.: 3mdeb Dasharo (Slim Bootloader+UEFI) v0.9.0
*****
**      1) Dasharo HCL report
**      2) Update Dasharo Firmware
**      3) Restore firmware from Dasharo HCL report
**      4) Load your DPP keys
*****
R to reboot  P to poweroff  S to enter shell
K to launch SSH server  L to enable sending DTS logs
```

Enter an option:





→ (Measure,) Verify, Execute

CoT - Chain of Trust

IBV - Independent BIOS Vendor

IFV - Independent Firmware Vendor

SEC/PEI/DXE/BSD - UEFI boot phases

- Dasharo (Slim Bootloader + UEFI) Stage1A and Stage1B is mutable because we do not have any Root of Trust technology employed.
  - Suitable RoT technology for our hardware platform (Hardkernel Odroid-H4 with Intel N97 (aka ADL-N)) is Intel Boot Guard.
- Since Intel Boot Guard is part of further sections of training, we tried to leverage next in Chain of Trust after Intel Boot Guard, which was Slim Bootloader (SBL) Verified Boot.
- We proved that Stage1B detected modification in Stage2.
- Types of Root of Trust will be explained in future modules.

# Quiz #2

# Quiz #2

Where Root of Trust is located in modern x86 platform?

# Quiz #2

Where Root of Trust is located in modern x86 platform?

- Peripheral processor or inside chip (ME/CSME/PSP/ASP)

# Quiz #2

Where Root of Trust is located in modern x86 platform?

- Peripheral processor or inside chip (ME/CSME/PSP/ASP)

What chain of trust mechanism Slim Bootloader can leverage?

# Quiz #2

Where Root of Trust is located in modern x86 platform?

- Peripheral processor or inside chip (ME/CSME/PSP/ASP)

What chain of trust mechanism Slim Bootloader can leverage?

- Verified Boot,

# Quiz #2

Where Root of Trust is located in modern x86 platform?

- Peripheral processor or inside chip (ME/CSME/PSP/ASP)

What chain of trust mechanism Slim Bootloader can leverage?

- Verified Boot,

What is the mechanism protecting Chain of Trust in Slim Bootloader in our demos/labs?

## Quiz #2

Where Root of Trust is located in modern x86 platform?