UEFI & EDK II Training

UEFI Aware Operating System

tianocore.org

LESSON OBJECTIVE

- Explain How the OS and UEFI Work together
- Explain the UEFI Requirements for UEFI aware OS
- Explain How Secure Boot Fits with UEFI
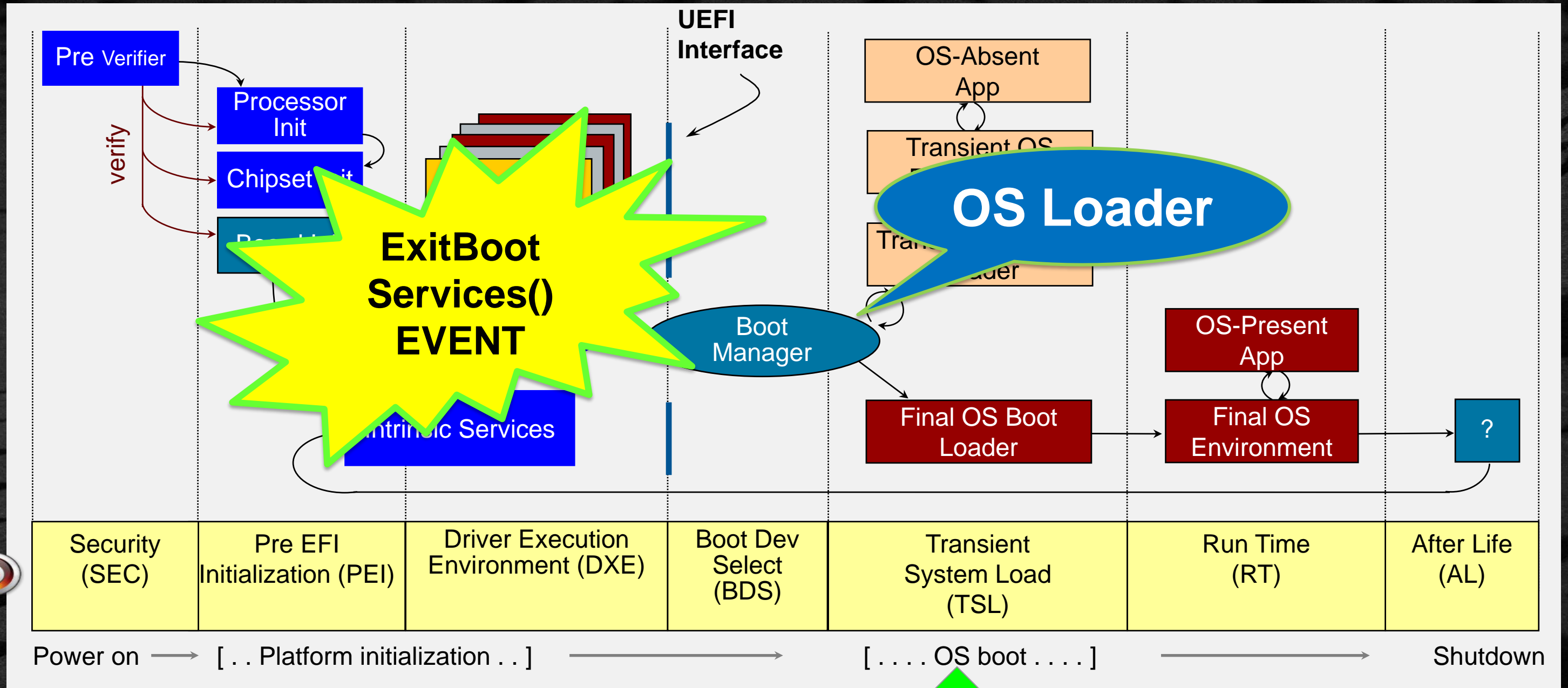- What about coreboot?

UEFI AWARE OS REQUIREMENTS

Common Requirements

UEFI - PI & EDK II Boot Flow - Review

UEFI Interface

Pre Verifier

verify

Processor Init

Chipset Init

Board Init

ExitBoot Services() EVENT

Intrinsic Services

OS-Absent App

Transient OS

Transient Loader

OS Loader

Boot Manager

Final OS Boot Loader

OS-Present App

Final OS Environment

?

Security (SEC)	Pre EFI Initialization (PEI)	Driver Execution Environment (DXE)	Boot Dev Select (BDS)	Transient System Load (TSL)	Run Time (RT)	After Life (AL)

Power on ⟶ [. . Platform initialization . .] ⟶ [. . . . OS boot . . . .] ⟶ Shutdown

5

tianocore

UEFI
Int

Pre Verifier

verify

Processor Init

Chipset Init

Board Init

Device, Bus, or Service Driver

UEFI Driver Dispatcher

Intrinsic Services

Security (SEC)	Pre EFI Initialization (PEI)	Driver Execution Environment (DXE)	(BDS)	System	(DT)	(AL)

Power on ⟶ [. . Platform initialization . .] ⟶ [. . . . OS boot . . . .] ⟶ Shutdown

6

tianocore

UEFI Drivers:
Boot devices/console

UEFI OS installer

UEFI OS Loader

Disk Partition/Formats

Firmware Requirements

Set Boot Path to Boot to UEFI OS

UEFI System Classes (based on firmware interfaces)

tianocore

UEFI Class 0

- Boots Legacy - int 19 ONLY
- Legacy BIOS Only (16 bit)
- No UEFI or UEFI PI Interfaces

UEFI Class 1

- Boots Legacy - int 19 ONLY
- Uses UEFI / PI Interfaces
- Only legacy BIOS runtime Interfaces

UEFI Class 2

- Boots Legacy - int 19 or UEFI
- Uses UEFI / PI Interfaces
- Legacy BIOS runtime Interfaces w/ **CSM**

Limited Benefits

- ✓ OEMs / ODMs Internal
- ✓ Double code development
- ✓ Compromised security – MBR exposure

UEFI System Classes (based on firmware interfaces)

tianocore

UEFI Class 0
- Boots Legacy - int 19 ONLY
- Legacy BIOS Only (16 bit)
- No UEFI or UEFI PI Interfaces

UEFI Class 1
- Boots Legacy - int 19 ONLY
- Uses UEFI / PI Interfaces
- Only legacy BIOS runtime Interfaces

UEFI Class 2
- Boots Legacy - int 19 or UEFI
- Uses UEFI / PI Interfaces
- Legacy BIOS runtime Interfaces w/ **CSM**

UEFI Class 3
- Boots **ONLY** UEFI
- Uses UEFI / PI Interfaces
- Runtime exposes only UEFI interfaces

UEFI System Classes (based on firmware interfaces)

Full Benefits

✓ UEFI Innovation
✓ Smaller code size/ Validation
✓ Extensibility

Only Class after 2020

Enabling *Secure Boot* creates another Class

UEFI Class 3 +

- Boots **ONLY** UEFI
- Uses UEFI / PI Interfaces
- Runtime exposes only UEFI interfaces

UEFI Secure Boot "ON"

- OS install process includes UEFI loader

  `/efi/boot/bootx64.efi /efi/redhat/grub.efi`

- Call UEFI boot & runtime services to start OS
- Exit UEFI Boot Services
- Transfer control to native OS

UEFI OS INSTALLER

- Discover UEFI storage devices
- Setup storage device: GPT w/ FAT32 boot partition
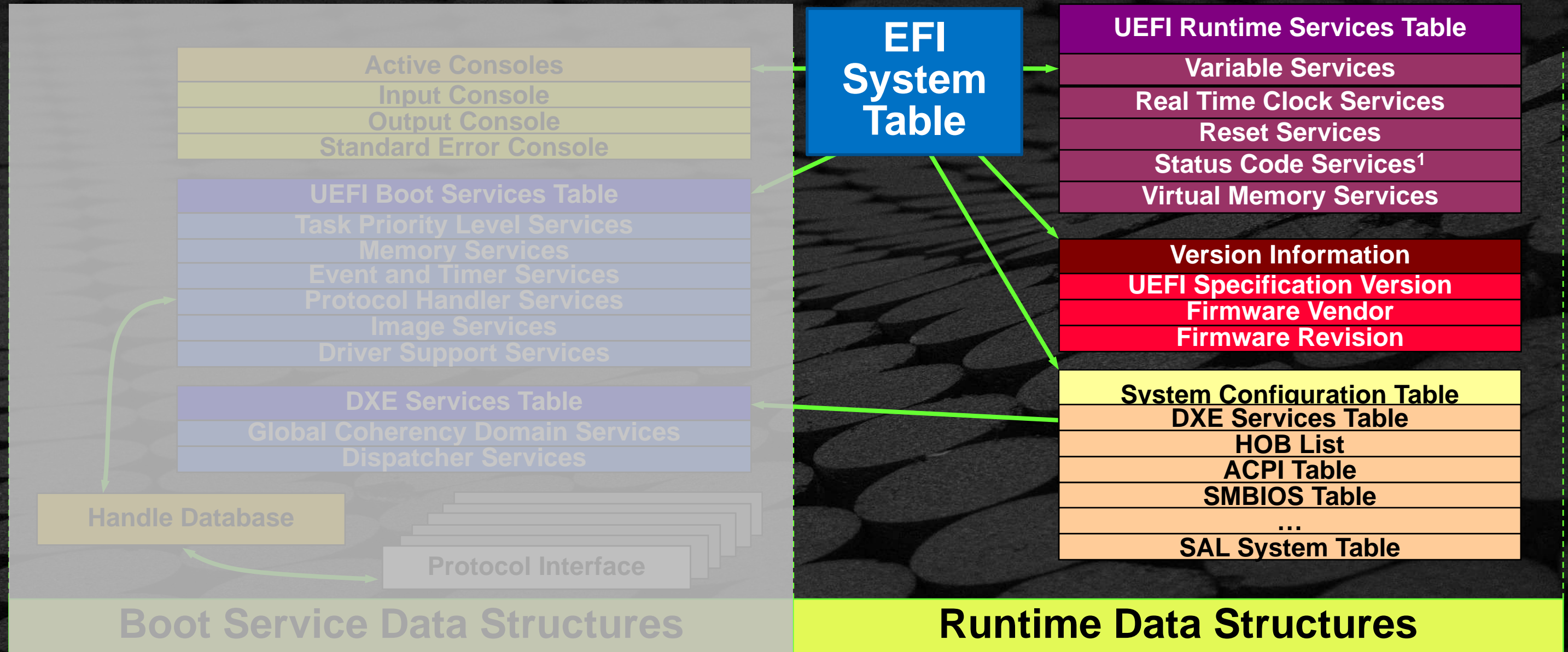- Create boot variables `BootXXXX` and set the `BootNext`

GPT
DISK

FAT32 partition
/EFI/BOOT/OS-loader.efi

Xyz partition (OS)

Reserved partition
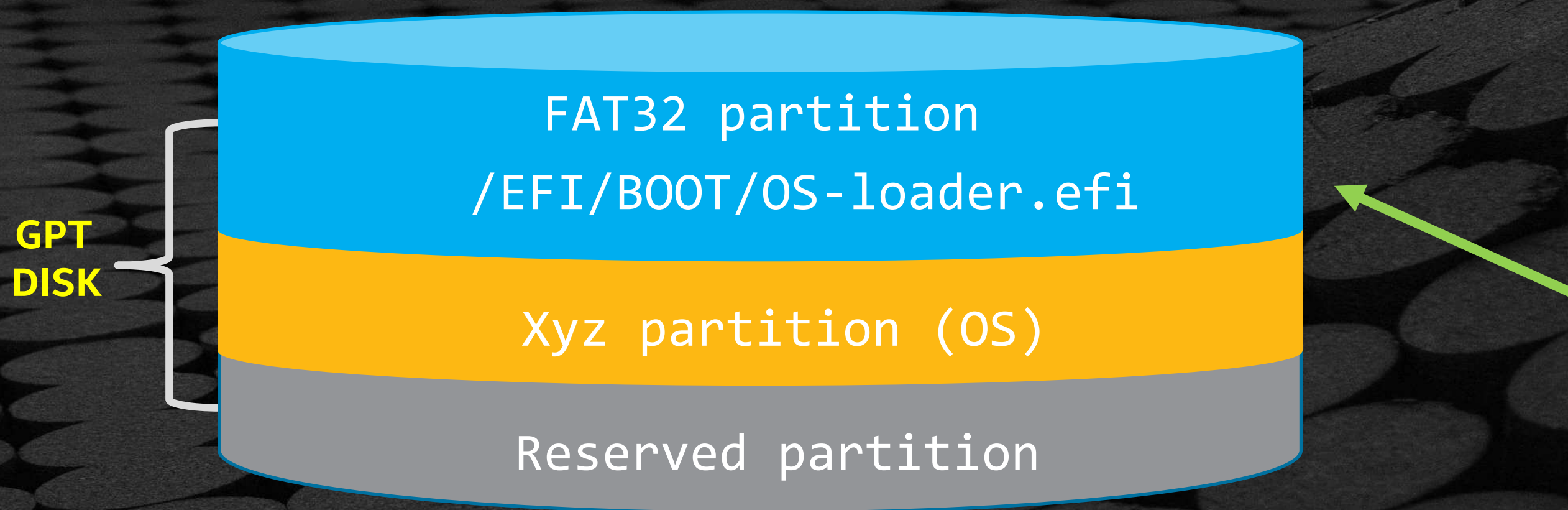
INTERFACE INSIDE OS RUNTIME

UEFI Runtime Services

Runtime Services Available to the UEFI Aware OS

EFI System Table

UEFI Runtime Services Table
- Variable Services
- Real Time Clock Services
- Reset Services
- Status Code Services[1]
- Virtual Memory Services

Active Consoles
- Input Console
- Output Console
- Standard Error Console

UEFI Boot Services Table
- Task Priority Level Services
- Memory Services
- Event and Timer Services
- Protocol Handler Services
- Image Services
- Driver Support Services

Version Information
- UEFI Specification Version
- Firmware Vendor
- Firmware Revision

DXE Services Table
- Global Coherency Domain Services
- Dispatcher Services

System Configuration Table
- DXE Services Table
- HOB List
- ACPI Table
- SMBIOS Table
- …
- SAL System Table

Handle Database

Protocol Interface

Boot Service Data Structures

Runtime Data Structures

Accessing RT services from Windows API

- GetFirmwareEnvironmentVariable: MSDN Link
- SetFirmwareEnvironmentVariable: MSDN Link
- Example: (determine if UEFI or Legacy BIOS)

```c
int main(int argc, char*argv[])
{
        GetFirmwareEnvironmentVariableA("",
            "{00000000-0000-0000-0000-000000000000}",NULL,0);
        if (GetLastError() ==ERROR_INVALID_FUNCTION) {
                printf("Legacy"); // This.. is.. LEGACY BIOS....
                return 1;
        } else{
                printf("UEFI"); // This.. is.. UEFI
                return 0;
        }
        return 0;
}
```

ACCESSING RT SERVICES FROM LINUX OS

Firmware Test Suite, it includes a Linux kernel driver to help with it's interactions with UEFI. Note that this is a Linux-centric test suite, solution won't work for other OSes.

- http://kernel.ubuntu.com/git/hwe/fwts.git
- https://bugs.launchpad.net/ubuntu/+source/linux/+bug/1633506
- https://patchwork.kernel.org/patch/9323781/
- http://www.basicinputoutput.com/2016/03/introduction-to-firmware-test-suite-fwts.html

SECURITY WITH UEFI
How does UEFI ensure the Operating System is trusted?

Security Resources: https://github.com/tianocore/tianocore.github.io/wiki/EDK-II-Security-White-Papers

BOOT SECURITY TECHNOLOGIES

Hardware Root of Trust   Boot Guard, Intel® TXT

Measured Boot   Using TPM[1] to store hash values

Verified Boot   Boot Guard +
– → UEFI Secure Boot

[1]TPM – Trusted Platform Module

Resources: https://firmwaresecurity.com/2015/07/29/survey-of-boot-security-technologies/

tianocore

Boot Guard

CPU verifies signature

Verification occurs before system FW starts
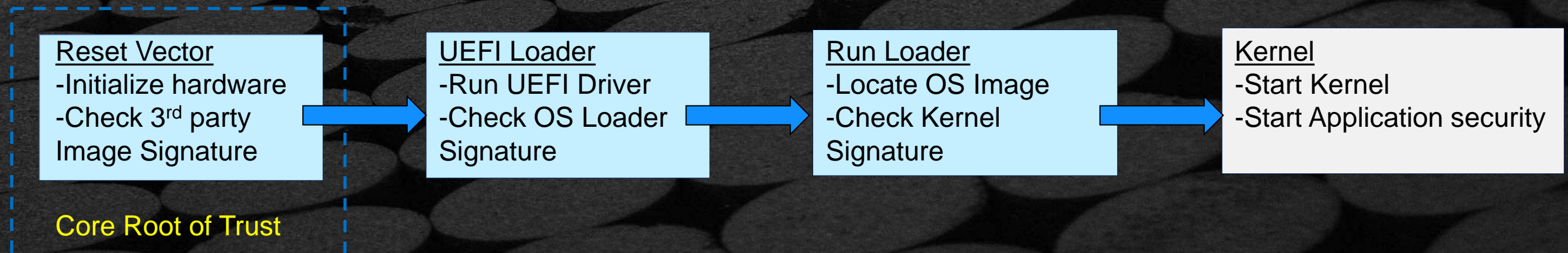
Hash of public key is fused in CPU

Intel® TXT

Uses a Trusted Platform Module (TPM) & cryptographic

Provides Measurements

Verification

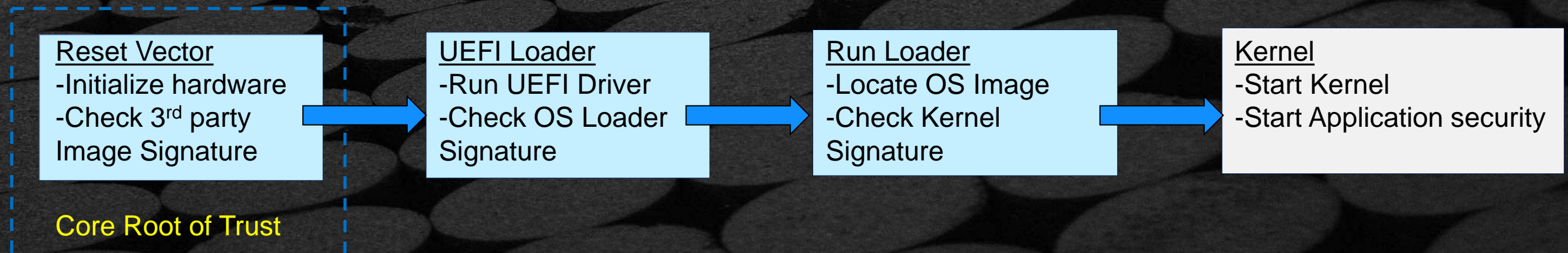## Measurements

tianocore

Software ID checking during every step of the boot flow:
1. UEFI System FW (updated via secure process)

Reset Vector
-Initialize hardware
-Check 3rd party
Image Signature

Core Root of Trust

→

UEFI Loader
-Run UEFI Driver
-Check OS Loader
Signature

→

Run Loader
-Locate OS Image
-Check Kernel
Signature

→

Kernel
-Start Kernel
-Start Application security

Software ID checking during every step of the boot flow:

1. UEFI System FW (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)

Reset Vector
-Initialize hardware
-Check 3rd party
Image Signature

→

UEFI Loader
-Run UEFI Driver
-Check OS Loader
Signature

→

Run Loader
-Locate OS Image
-Check Kernel
Signature

→

Kernel
-Start Kernel
-Start Application security

Core Root of Trust

Software ID checking during every step of the boot flow:

1. UEFI System FW (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)

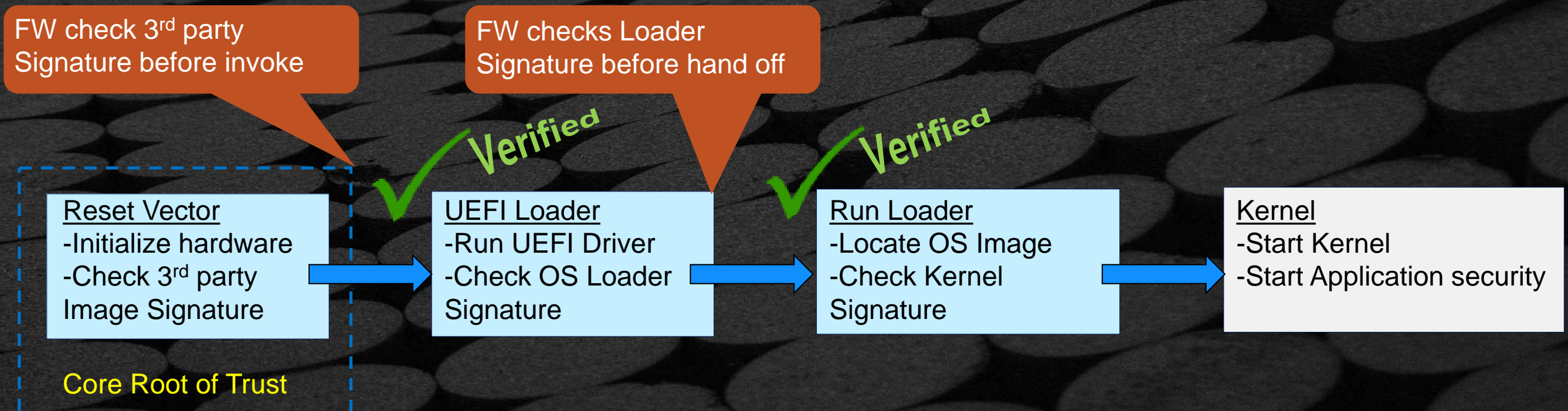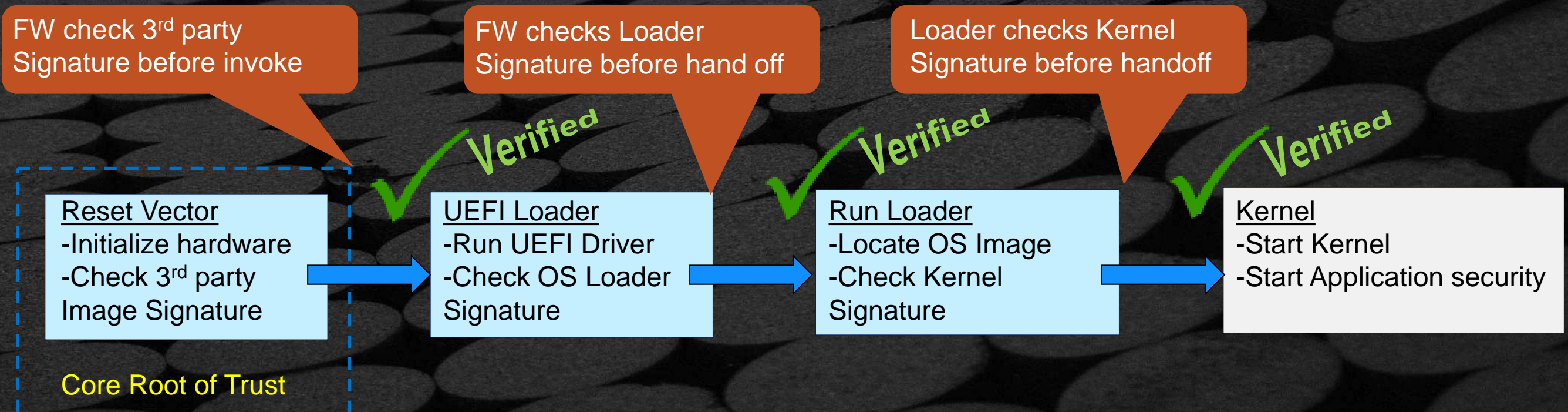FW check 3rd party
Signature before invoke

Verified ✓

| Reset Vector
-Initialize hardware
-Check 3rd party
Image Signature | → | UEFI Loader
-Run UEFI Driver
-Check OS Loader
Signature | → | Run Loader
-Locate OS Image
-Check Kernel
Signature | → | Kernel
-Start Kernel
-Start Application security |

Core Root of Trust

tianocore

Software ID checking during every step of the boot flow:

1. UEFI System FW (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)

FW check 3rd party
Signature before invoke

FW checks Loader
Signature before hand off

Verified

Verified

✓

✓

Reset Vector
-Initialize hardware
-Check 3rd party
Image Signature

UEFI Loader
-Run UEFI Driver
-Check OS Loader
Signature

Run Loader
-Locate OS Image
-Check Kernel
Signature

Kernel
-Start Kernel
-Start Application security

Core Root of Trust

*tianocore

Software ID checking during every step of the boot flow:

1. UEFI System FW (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)
3. OS Boot Loader (checks for "secure mode" at boot)

FW check 3rd party
Signature before invoke

FW checks Loader
Signature before hand off

✓ Verified

✓ Verified

Reset Vector
-Initialize hardware
-Check 3rd party
Image Signature

UEFI Loader
-Run UEFI Driver
-Check OS Loader
Signature

Run Loader
-Locate OS Image
-Check Kernel
Signature

Kernel
-Start Kernel
-Start Application security

Core Root of Trust

tianocore

Software ID checking during every step of the boot flow:

1. UEFI System FW (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)
3. OS Boot Loader (checks for "secure mode" at boot)

FW check 3rd party
Signature before invoke

FW checks Loader
Signature before hand off

Loader checks Kernel
Signature before handoff

✓ Verified ✓ Verified ✓ Verified

Reset Vector
-Initialize hardware
-Check 3rd party
Image Signature

UEFI Loader
-Run UEFI Driver
-Check OS Loader
Signature

Run Loader
-Locate OS Image
-Check Kernel
Signature

Kernel
-Start Kernel
-Start Application security

Core Root of Trust

PK

KEK

DB

DBX

SetupMode

SecureBoot

```
2.0 Shell> dmpstore SecureBoot
Variable - RS+BS -   '8BE4DF61-93CA-11D2-AA0D-00E098032B8C:SecureBoot' - DataSize
= 0x01
  00:                                                            00  *.*
2.0 Shell>
```

Security Package Project Page Wiki Link

- Wiki Link: How-to-Enable-Security

- PDF: How to Sign UEFI Images V1.31

- Build command line switch -
  SECURE_BOOT_ENABLE = TRUE

- Install the OpensslLib CryptoPkg :
  OpenSSL-Howto.txt
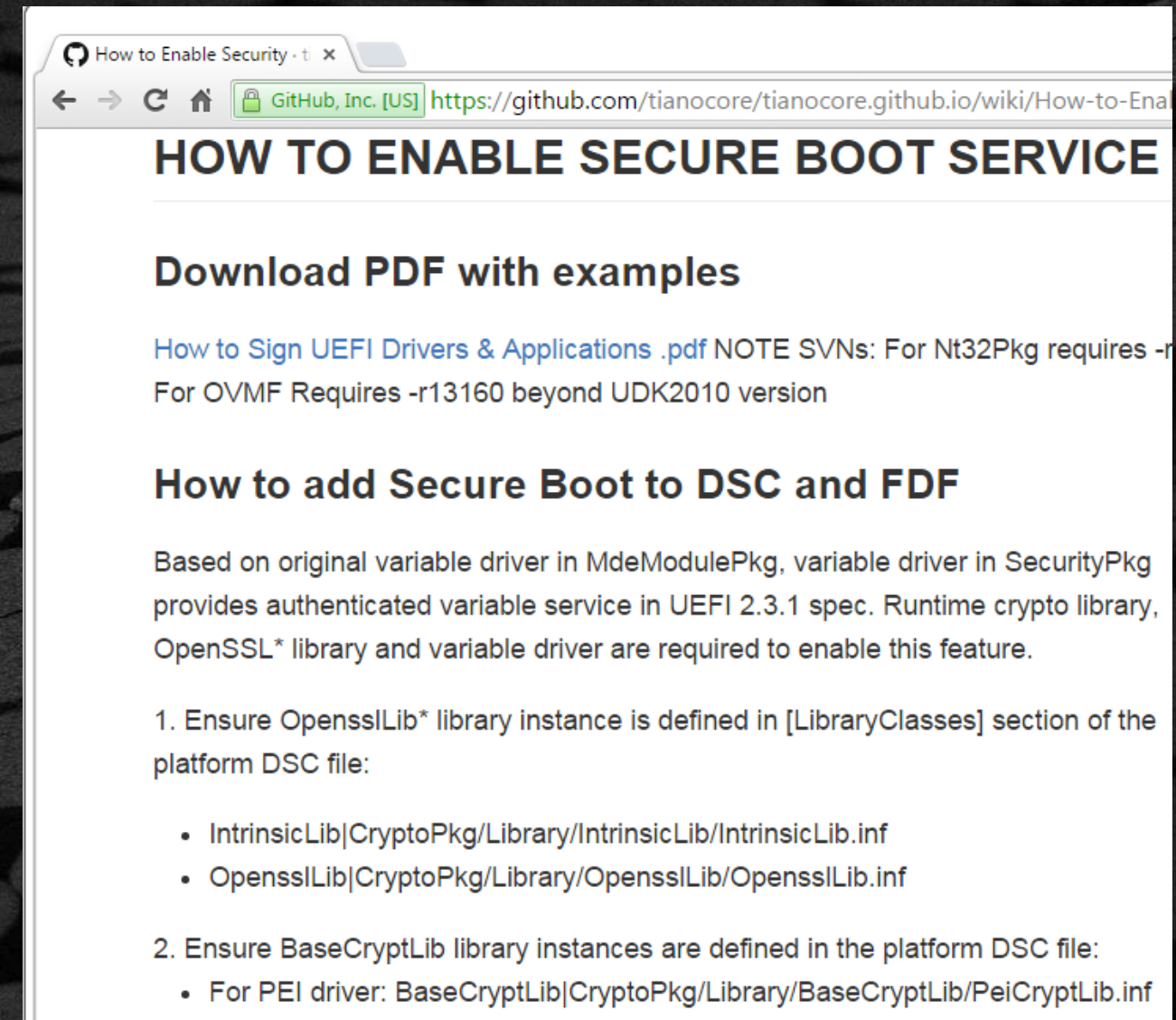
Windows Secure Boot Key Creation and Management Guidance

- Windows - Secure Boot Key Creation & Management Guide

- Creation and management of the Secure Boot keys and certificates in a manufacturing environment.

- Addresses questions related to creation, storage and retrieval of Platform Keys (PKs), secure firmware update keys, and third party Key Exchange Keys (KEKs).

Many Platforms are Requiring UEFI Secure Boot Enabled

- Secure Boot now mandated for specific platforms

- See "Security requirements" on UEFI requirements for Windows editions on SoC Platforms
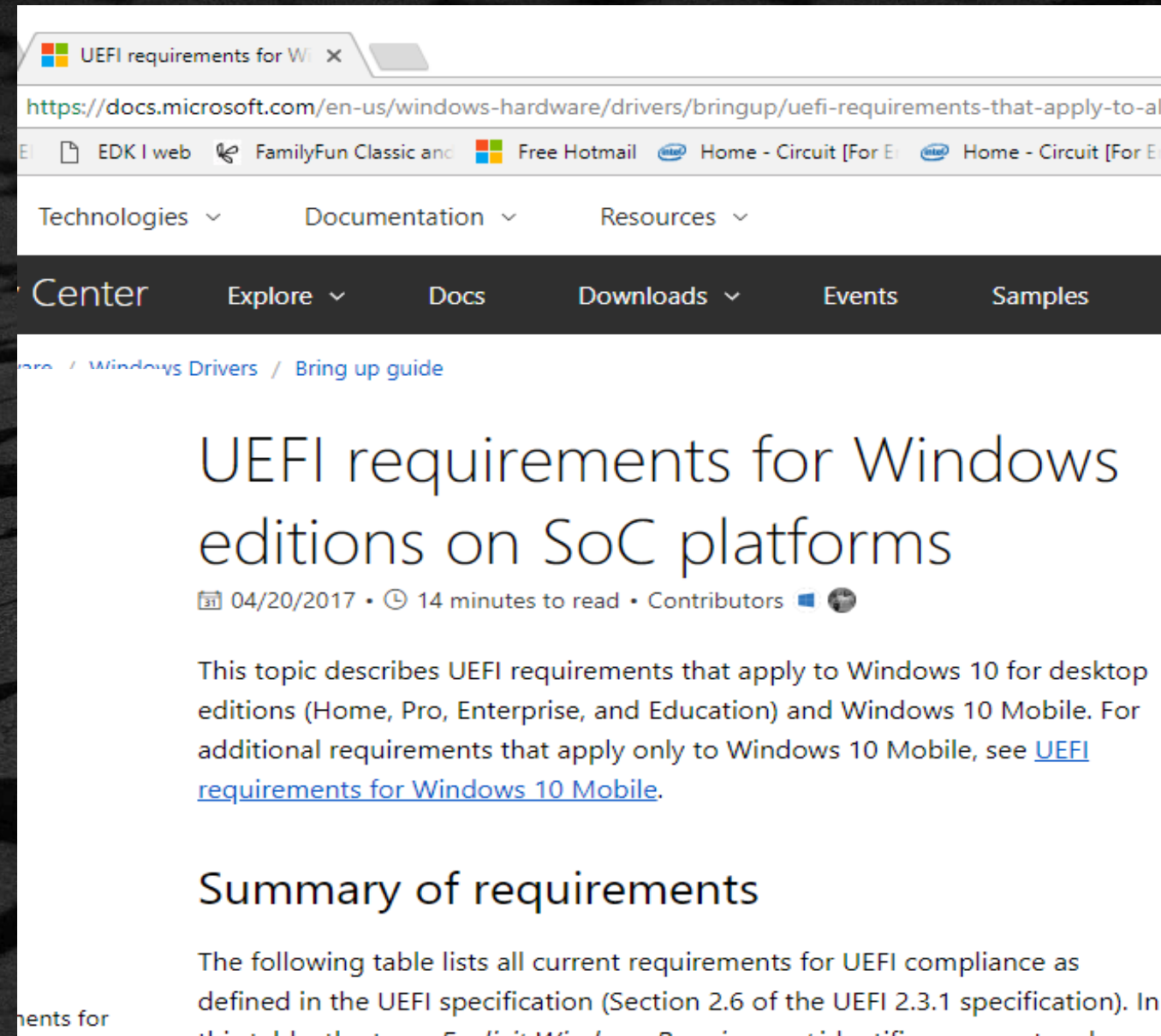
COREBOOT

How does coreboot work with UEFI

31

Intel® FSP "Produced" to "Consuming" Intel® Architecture Firmware

tianocore

UEFI/PI Scope - Green "H" w/ EDK2

OS
UEFI Specification

Platform

Drivers

Intel® FSP

Hardware/Silicon

Intel® FSP

Glue Code (PEI Core / Arch PPIs)

Single Si Init Binary

| CPU | SA | PCH | ME |

OS

Payload

coreboot ramstage

Intel® FSP

coreboot romstage

Hardware/Silicon
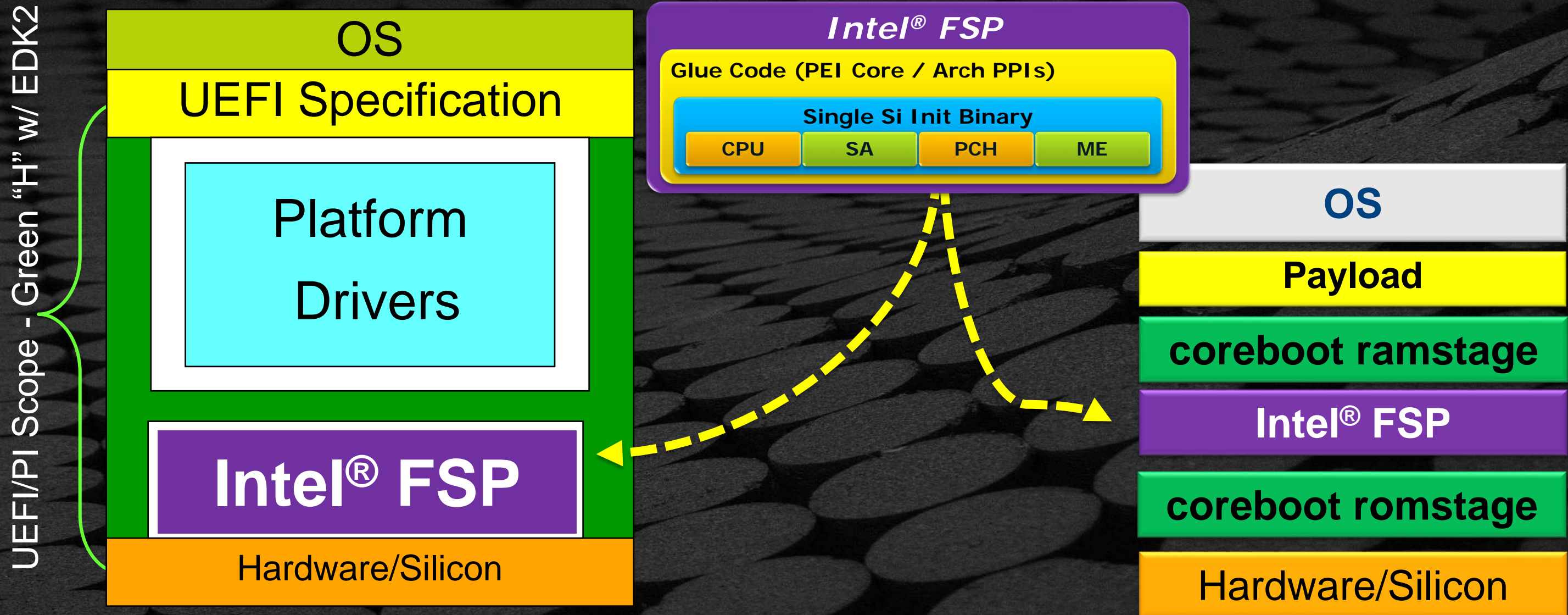
Intel FSP is independent of the bootloader solutions

tianocore

Functionality	coreboot	UEFI / PI
The reset vector and pre cache-as-ram setup	bootblock	Security Phase (SEC)
Cache as Ram setup, early silicon initialization, memory setup. Covered largely by Intel® Firmware Support Package	romstage	Pre-EFI Initialization (PEI) Create HOBs
Normal device setup and mainboard configuration. Publish SMBIOS/ACPI Tables	ramstage	Early Driver Execution Environment (DXE)
Memory map hand-off	CBMEM	UEFI Memory Map
The OS or application bootloader	payload	DXE BDS and UEFI Drivers

SUMMARY

Explain How the OS and UEFI Work together

Explain the UEFI Requirements for UEFI aware OS

Explain How Secure Boot Fits with UEFI

What about coreboot?

Questions?

Return to Main Training Page

Return to Training Table of contents for next presentation link

www.tianocore.org

Backup

tianocore

- Deficiency: Boot path malware targets
- UEFI and Secure Boot harden the boot process
- Firmware/software in the boot process must be signed by a trusted Certificate Authority (CA)
- Firmware image is hardware-protected
- 3rd party drivers signed using CA-holding trusted keys
- Trusted signing key's database factory-initialized and OS-updated

WHY??? SECURE BOOT WITH UEFI

Without

With

Possible corrupted or destroyed data

- BootKit virus – MBR Rootkits
- Network boot attacks e.g. PXESPOILT
- Code Injection Attacks

Data integrity

- **Trusted boot to OS**
- **Trusted drivers**
- **Trusted Applications**

UEFI Secure Boot Flow

tianocore

1. Enroll

PEI FV

Authenticated Variable

PK

KEK

db — Certificate

dbx — Certificate

Variable

DXE FV

Image Verify

2C. Signed Image Load And measure Into TPM

OpRom.efi
Certificate + SignInfo

2A. Signed Image Discover

OsLoader.efi
Certificate + SignInfo

2B. Signature Verification