

# UEFI & EDK II Training

## UEFI Aware Operating System

[tianocore.org](http://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 MM (Formerly Known as SMM)
-  What about coreboot?

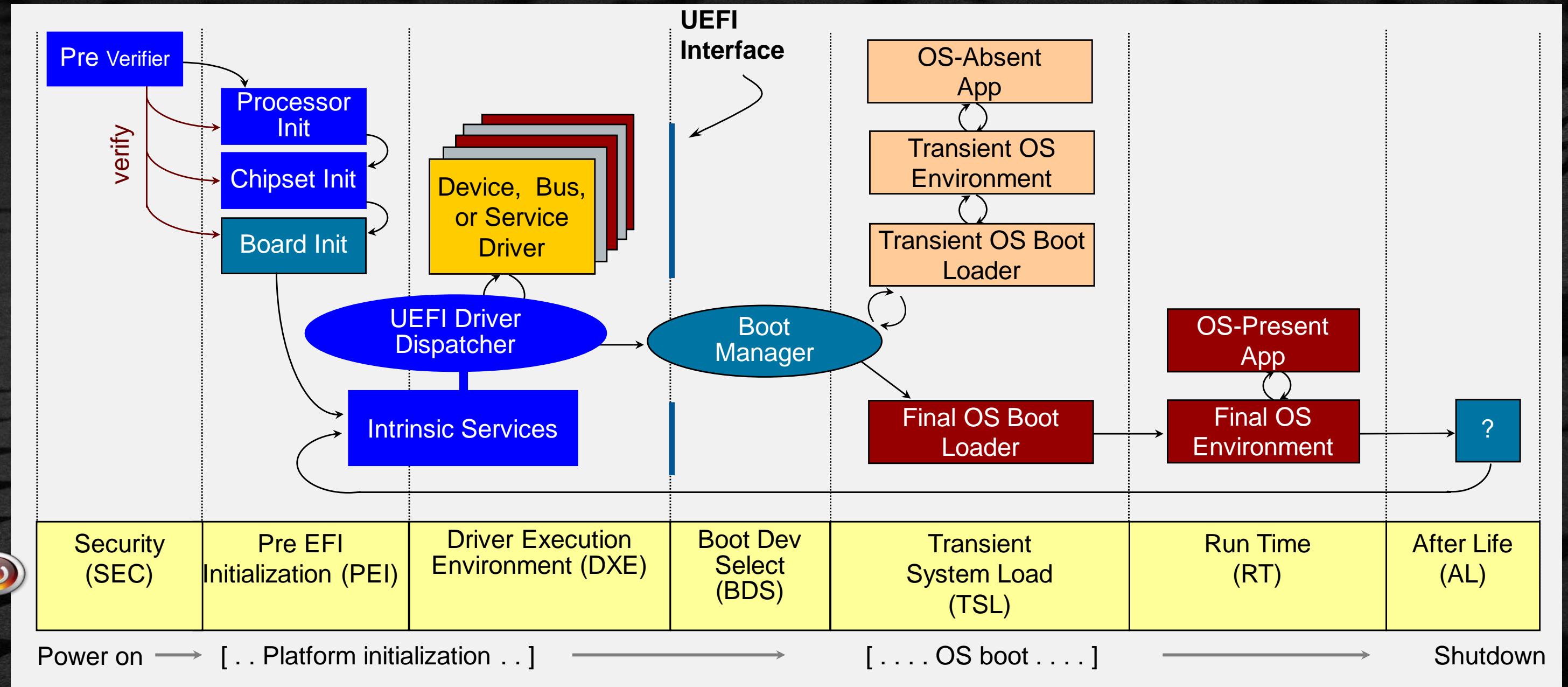
# UEFI AWARE OS REQUIREMENTS

Common Requirements

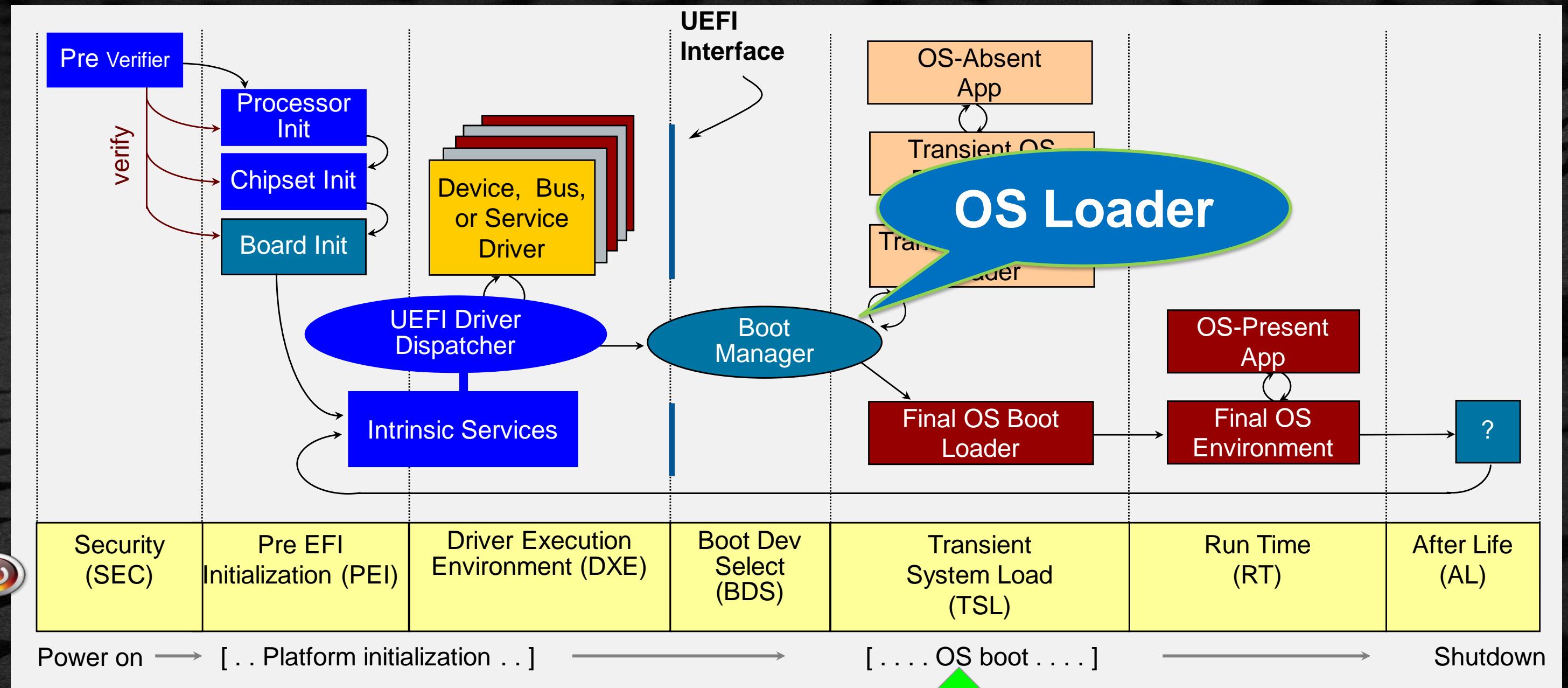
# UEFI OPERATING SYSTEMS



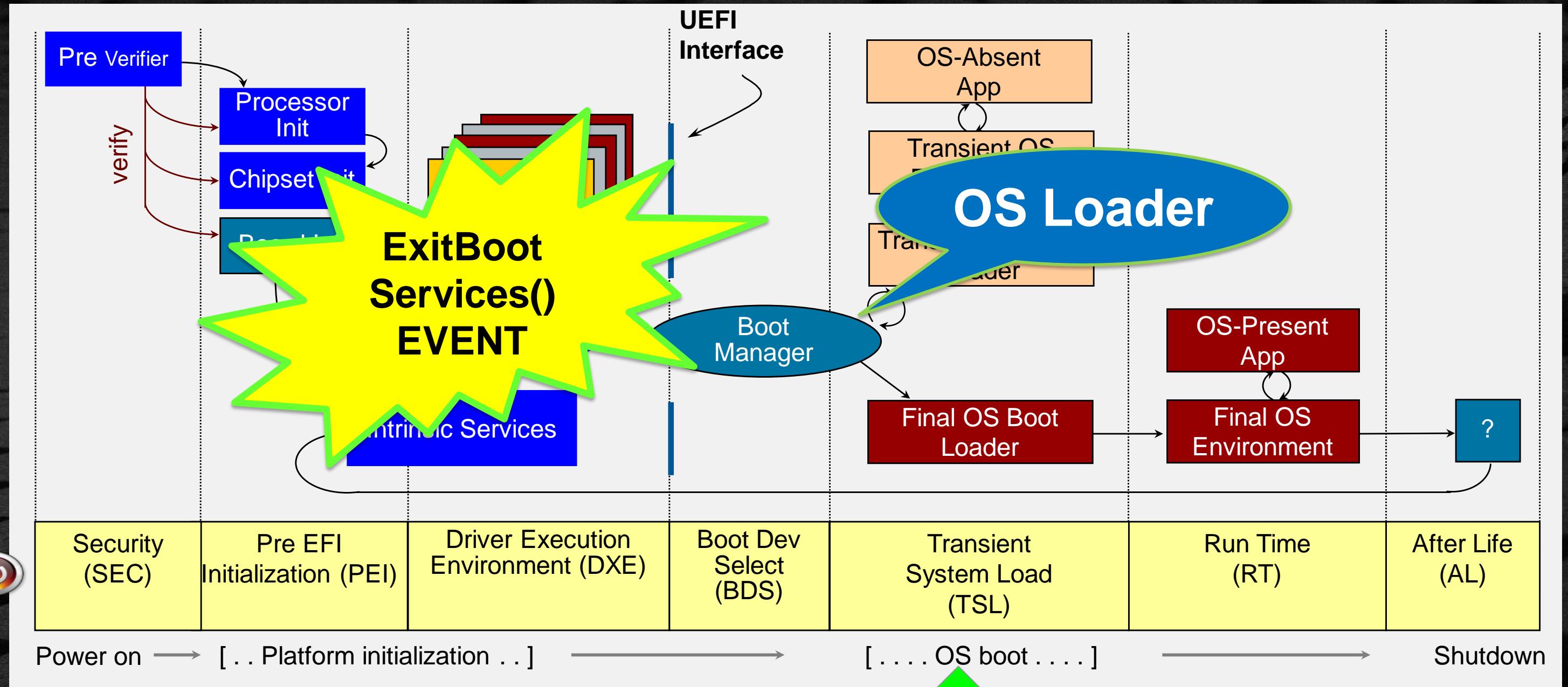
# UEFI - PI & EDK II Boot Flow - Review



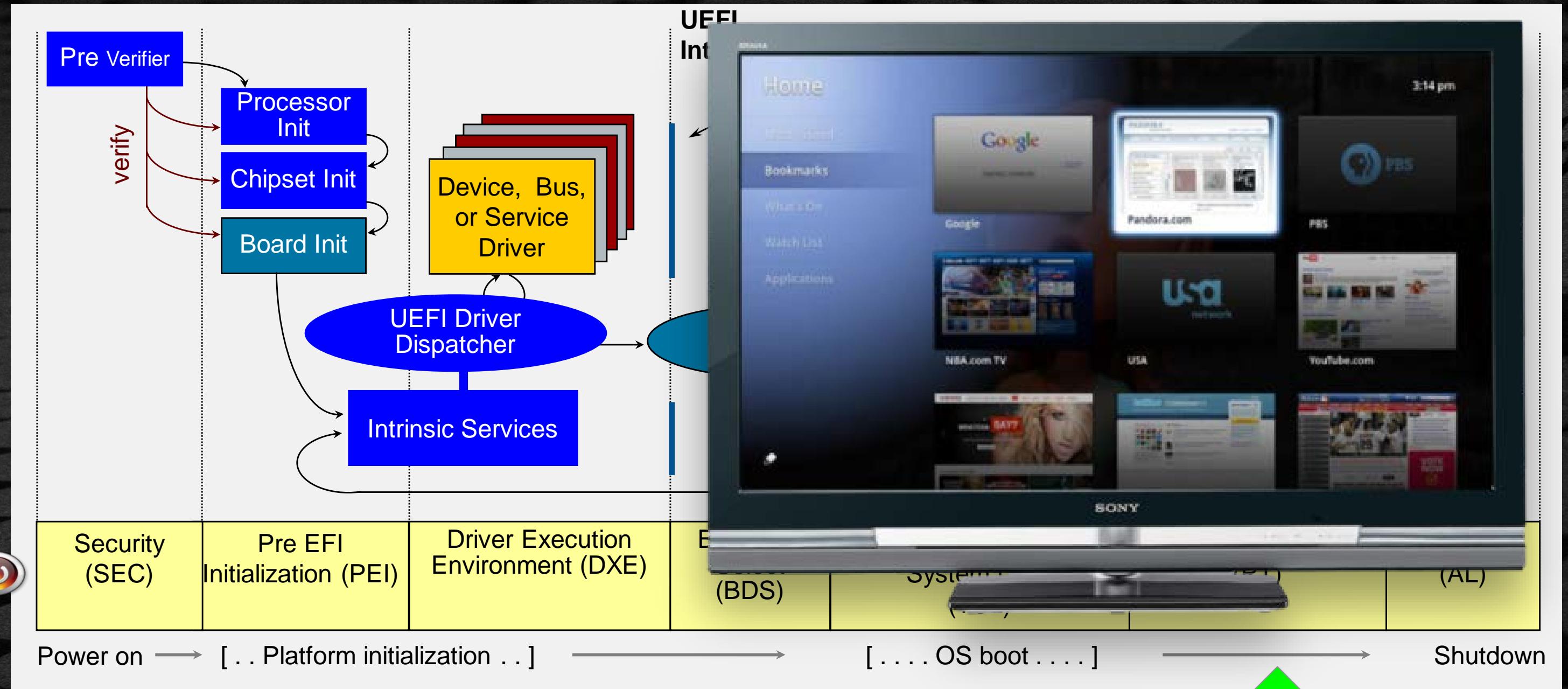
# UEFI - PI & EDK II Boot Flow - Review



# UEFI - PI & EDK II Boot Flow - Review



# UEFI - PI & EDK II Boot Flow - Review



# UEFI OS REQUIREMENTS

UEFI Drivers: Boot devices/console

# UEFI OS REQUIREMENTS

UEFI Drivers: Boot devices/console

UEFI OS installer

# UEFI OS REQUIREMENTS

UEFI Drivers: Boot devices/console

UEFI OS installer

UEFI OS Loader

# UEFI OS REQUIREMENTS

UEFI Drivers: Boot devices/console

UEFI OS installer

UEFI OS Loader

Disk Partition/Formats

# UEFI OS REQUIREMENTS

UEFI Drivers: Boot devices/console

UEFI OS installer

UEFI OS Loader

Disk Partition/Formats

Firmware Requirements

# UEFI OS REQUIREMENTS

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)

## UEFI Class 0

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

# UEFI System Classes

(based on firmware interfaces)

## 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
- Runtime exposes only legacy BIOS runtime interfaces

# UEFI System Classes

(based on firmware interfaces)

## 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
- Runtime exposes only legacy BIOS runtime interfaces

## UEFI Class 2

- Boots legacy - int 19 or UEFI
- Uses UEFI /PI interfaces
- Runtime exposes UEFI & legacy BIOS interface
- CSM

# UEFI System Classes

(based on firmware interfaces)

## 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
- Runtime exposes only legacy BIOS runtime interfaces

## UEFI Class 2

- Boots legacy - int 19 or UEFI
- Uses UEFI /PI interfaces
- Runtime exposes UEFI & legacy BIOS interface
- CSM

## Limited Benefits

- ✓ OEMs/ODMs Internal
- ✓ Double code development & organization
- ✓ Compromised security –(MBR exposure)

# UEFI System Classes

(based on firmware interfaces)

## 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
- Runtime exposes only legacy BIOS runtime interfaces

## UEFI Class 2

- Boots legacy - int 19 or UEFI
- Uses UEFI /PI interfaces
- Runtime exposes UEFI & legacy BIOS interface
- CSM

## UEFI Class 3

- Boots only UEFI
- Uses UEFI / PI interfaces
- Runtime exposes only UEFI interfaces

# UEFI System Classes

(based on firmware interfaces)

## UEFI Class 3

### Full Benefits

- ✓ UEFI Innovation
- ✓ Smaller code size/  
Validation
- ✓ Extensibility

## UEFI Class 1

- Boots Legacy - int 19
- Uses UEFI /PI interfaces
- Runtime exposes only legacy  
BIOS runtime interfaces

## UEFI Class 3

- Boots only UEFI
- Uses UEFI / PI interfaces
- Runtime exposes only UEFI  
interfaces

# UEFI System Classes

(based on firmware interfaces)

## UEFI Class 3 +

### Full Benefits

- ✓ UEFI Innovation
- ✓ Smaller code size/  
Validation
- ✓ Extensibility

**Only Class after 2020**

Enabling *Secure Boot*  
creates another Class

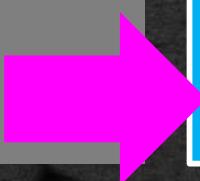
## UEFI Class 1

- Boots Legacy - int 19
- Uses UEFI / PI interfaces
- Runtime exposes only legacy  
BIOS runtime interfaces

## UEFI Class 3 +

- Boots only UEFI
- Uses UEFI / PI interfaces
- Runtime exposes only UEFI  
interfaces

**UEFI Secure Boot “ON”**



# Required UEFI Drivers: OS Install & Boot

Boot Device

# Required UEFI Drivers: OS Install & Boot

Boot Device

Console Output

# Required UEFI Drivers: OS Install & Boot

Boot Device

Console Output

Console Input

# Required UEFI Drivers: OS Install & Boot

Boot Device

Console Output

Console Input

NVRAM Driver

# UEFI OS LOADER

- 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 variable **BootXXXX**

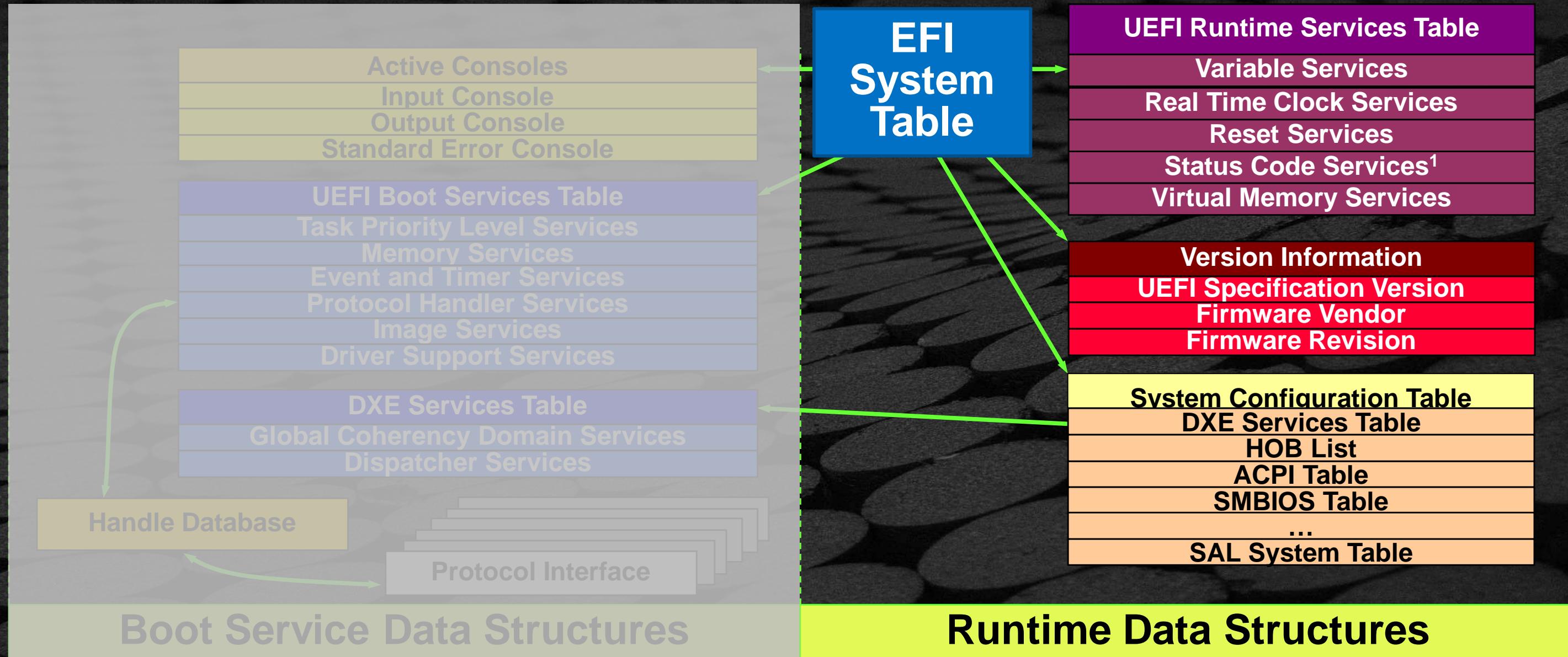
# Disk Partition and Format



# INTERFACE INSIDE OS RUNTIME

## UEFI Runtime Services

# Runtime Services Available to the UEFI Aware OS



# Accessing RT services from Windows API

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

```
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?

# BOOT SECURITY TECHNOLOGIES

Hardware Root of Trust

Measured Boot

Verified Boot

- Boot Guard, Intel® TXT
- Using TPM to store hash values
- Boot Guard + UEFI Secure Boot

TPM – Trusted Platform Module

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

# HARDWARE ROOT OF TRUST

## Boot Guard

CPU verifies signature

Verification occurs before BIOS starts

Hash of signature is fused in CPU

## Intel® TXT

Uses a Trusted Platform Module (TPM)  
& cryptographic

Provides Measurements

# HARDWARE ROOT OF TRUST

## Boot Guard

CPU verifies signature

Verification occurs before BIOS starts

Hash of signature is fused in CPU

Verification

## Intel® TXT

Uses a Trusted Platform Module (TPM)  
& cryptographic

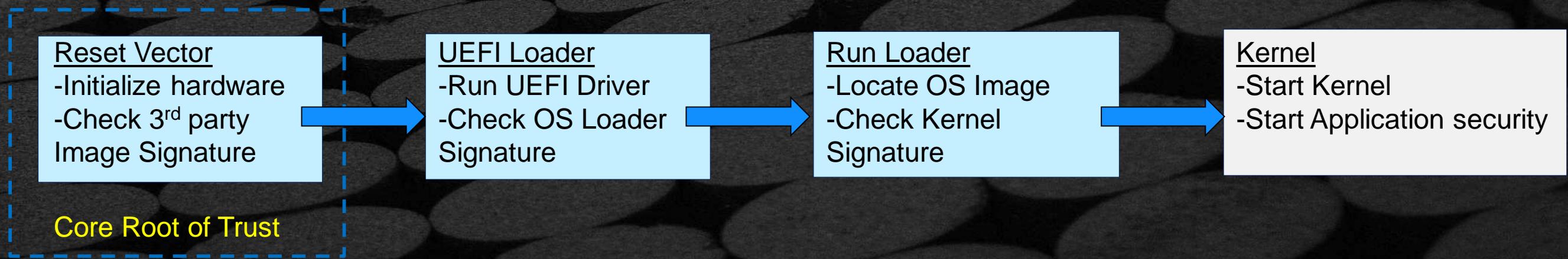
Provides Measurements

Measurements

# UEFI SECURE BOOT

Software ID checking during every step of the boot flow:

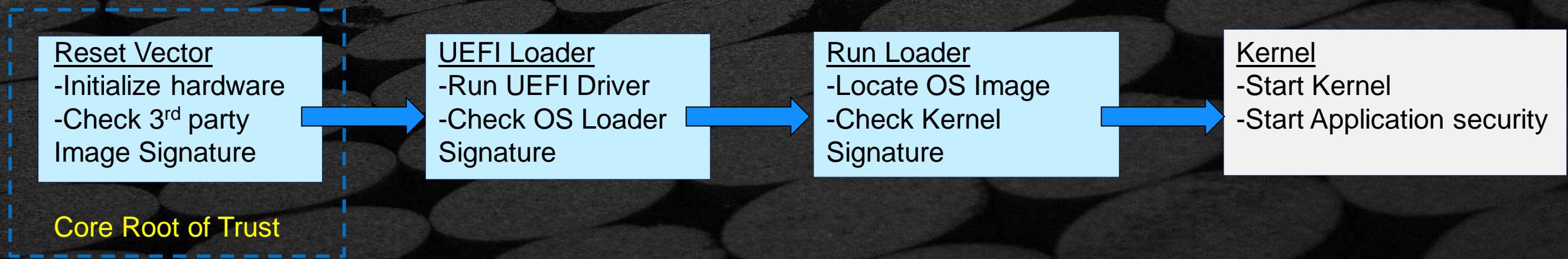
1. UEFI System BIOS (updated via secure process)



# UEFI SECURE BOOT

Software ID checking during every step of the boot flow:

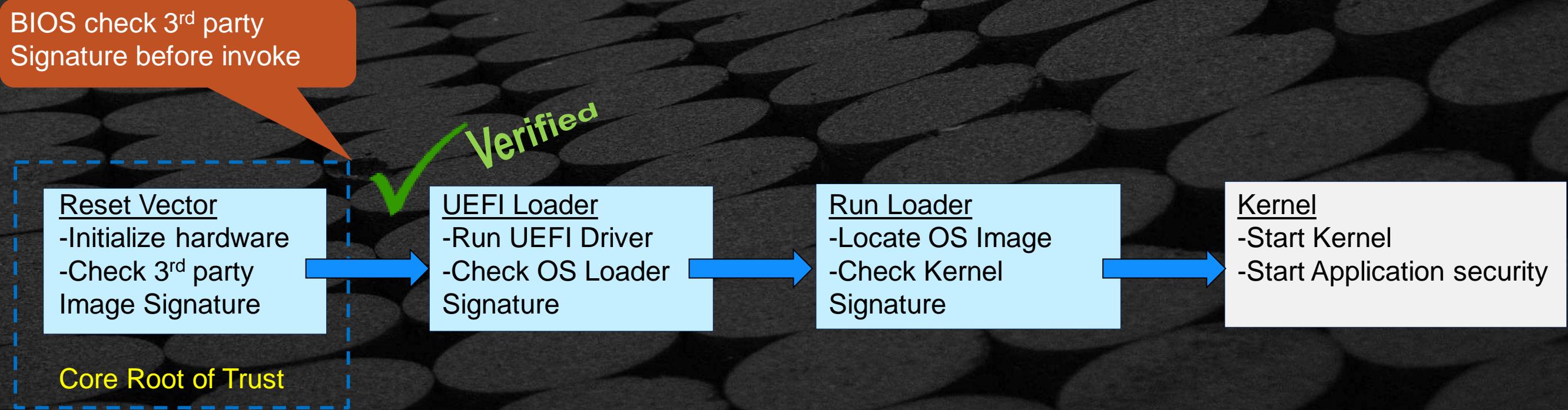
1. UEFI System BIOS (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)



# UEFI SECURE BOOT

Software ID checking during every step of the boot flow:

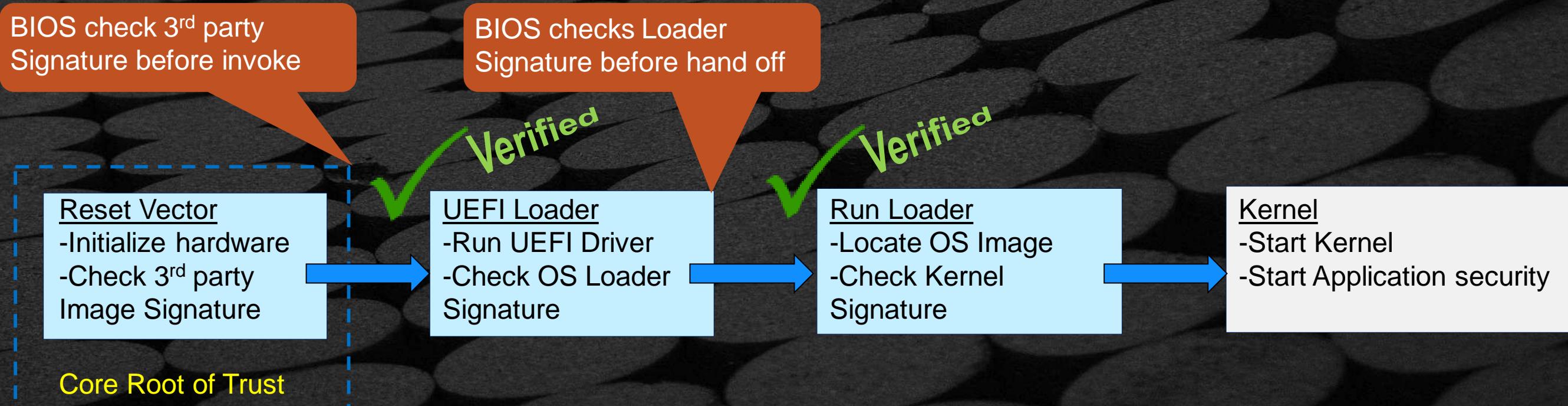
1. UEFI System BIOS (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)



# UEFI SECURE BOOT

Software ID checking during every step of the boot flow:

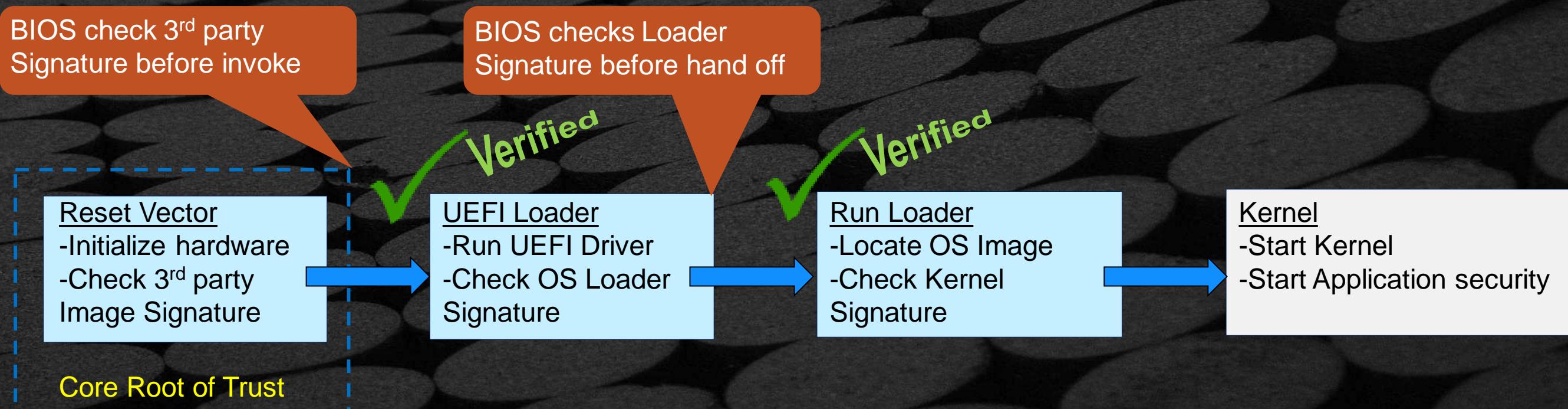
1. UEFI System BIOS (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)



# UEFI SECURE BOOT

Software ID checking during every step of the boot flow:

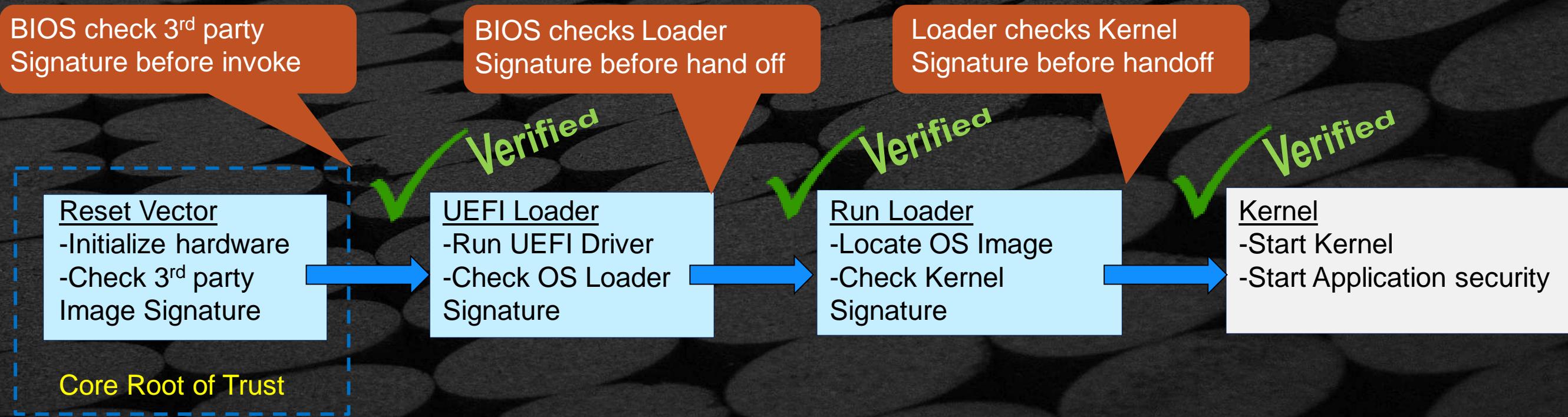
1. UEFI System BIOS (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)
3. OS Boot Loader (checks for “secure mode” at boot)



# UEFI SECURE BOOT

Software ID checking during every step of the boot flow:

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



# AUTHENTICATED VARIABLES

PK

KEK

DB

**DBX**

# SetupMode

# SecureBoot

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

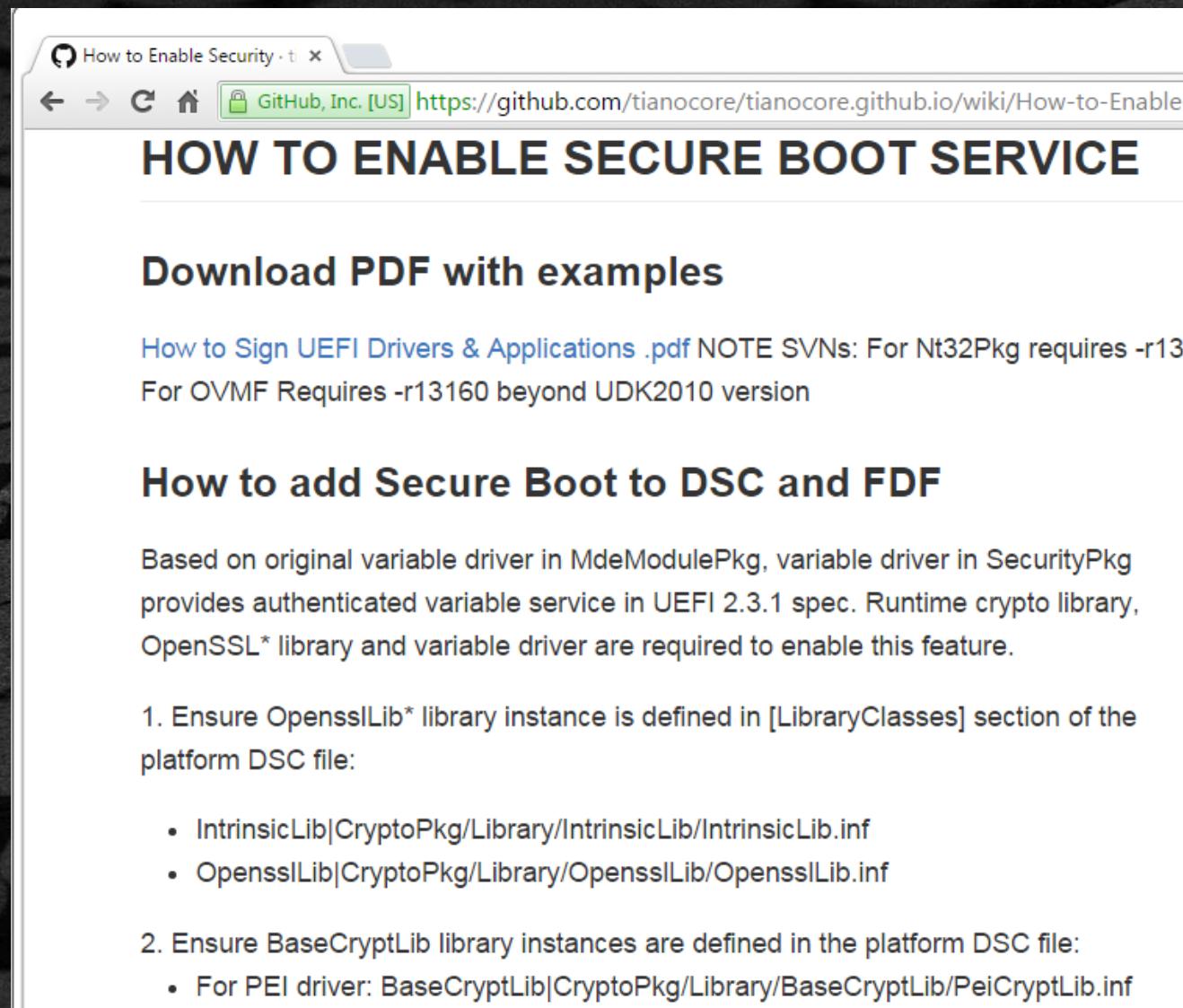
# Security Package Project Page Wiki Link

- Wiki Link: [How-to-Enable-Security](#)
- PDK: [How to Sign UEFI Images](#)  
[V1.31](#)
- Build command line switch -  
SECURE\_BOOT\_ENABLE = TRUE
- Install the OpensslLib CryptoPkg :  
[OpenSSL-Howto.txt](#)

**How To Enable  
Secure Boot Service**

# Security Package Project Page Wiki Link

- Wiki Link: [How-to-Enable-Security](#)
- PDK: [How to Sign UEFI Images](#)  
[V1.31](#)
- Build command line switch -  
SECURE\_BOOT\_ENABLE = TRUE
- Install the OpensslLib CryptoPkg :  
[OpenSSL-Howto.txt](#)



The screenshot shows a web browser window with the title "How to Enable Security · tianocore". The URL in the address bar is <https://github.com/tianocore/tianocore.github.io/wiki/How-to-Enable-Security>. The main content of the page is titled "HOW TO ENABLE SECURE BOOT SERVICE". Below this, there is a section titled "Download PDF with examples" which links to a PDF file named "How to Sign UEFI Drivers & Applications.pdf". A note states: "NOTE SVNs: For Nt32Pkg requires -r13160 beyond UDK2010 version For OVMF Requires -r13160 beyond UDK2010 version". Another section titled "How to add Secure Boot to DSC and FDF" provides instructions based on the original variable driver in MdeModulePkg, mentioning the SecurityPkg variable driver, UEFI 2.3.1 spec, Runtime crypto library, and OpenSSL\* library. It lists steps for defining OpenSSLLib\* library instances in the platform DSC file and for defining BaseCryptLib library instances.

**HOW TO ENABLE SECURE BOOT SERVICE**

**Download PDF with examples**

[How to Sign UEFI Drivers & Applications.pdf](#) NOTE SVNs: For Nt32Pkg requires -r13160 beyond UDK2010 version  
For OVMF Requires -r13160 beyond UDK2010 version

**How to add Secure Boot to DSC and FDF**

Based on original variable driver in MdeModulePkg, variable driver in SecurityPkg provides authenticated variable service in UEFI 2.3.1 spec. Runtime crypto library, OpenSSL\* library and variable driver are required to enable this feature.

1. Ensure OpenSSLLib\* library instance is defined in [LibraryClasses] section of the platform DSC file:
  - IntrinsicLib|CryptoPkg/Library/IntrinsicLib/IntrinsicLib.inf
  - OpenSSLLib|CryptoPkg/Library/OpenSSLLib/OpenSSLLib.inf
2. Ensure BaseCryptLib library instances are defined in the platform DSC file:
  - For PEI driver: BaseCryptLib|CryptoPkg/Library/BaseCryptLib/PeiCryptLib.inf

# Windows Secure Boot Key Creation and Management Guidance

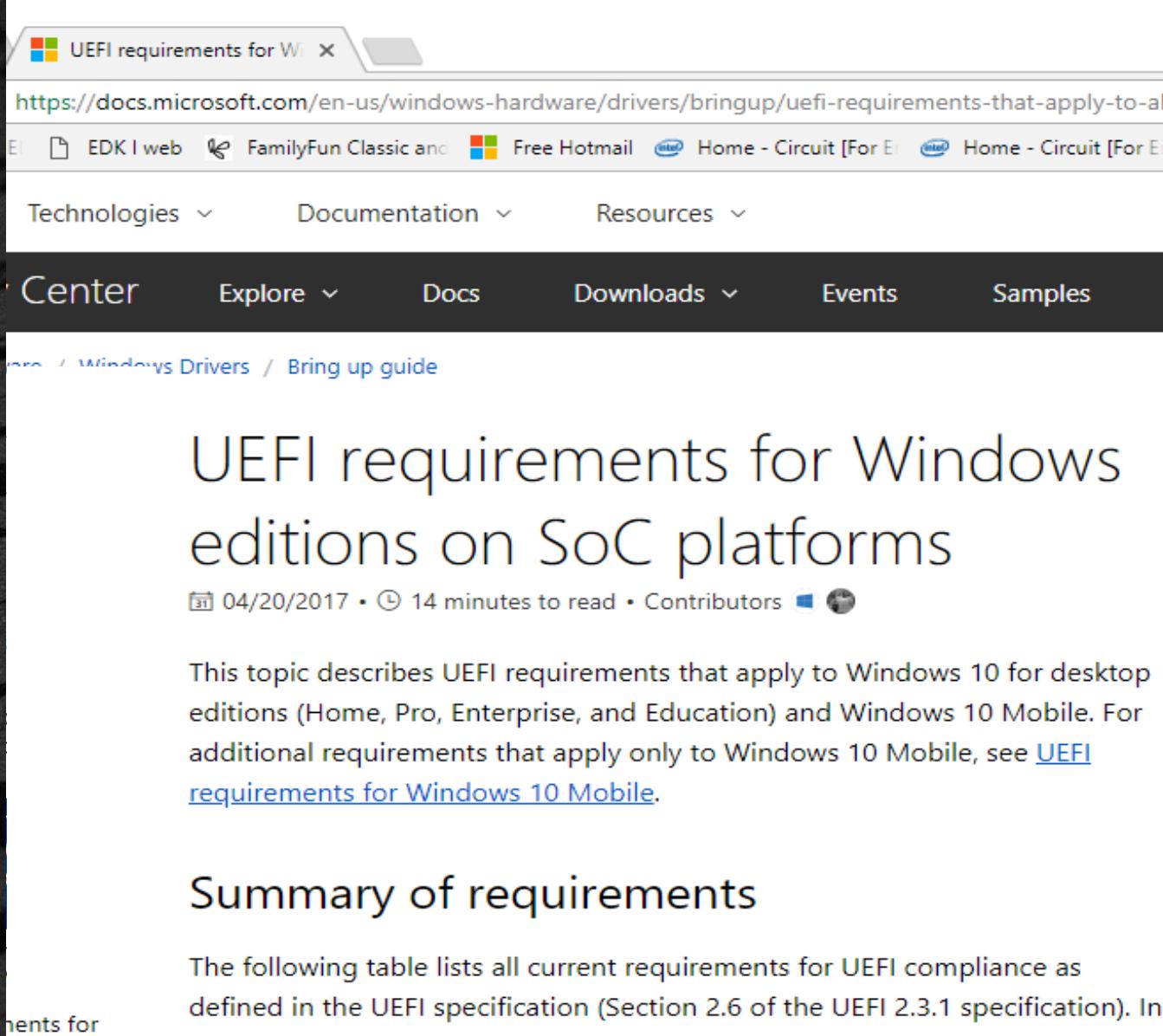
- Windows - Secure Boot Key Creation & Management Guide
- Creation and management of the Secure Boot keys and certificates in a manufacturing environment.
- Addresses questions related to creation, storage and retrieval of Platform Keys (PKs), secure firmware update keys, and third party Key Exchange Keys (KEKs).



The screenshot shows a Microsoft MSDN page titled "Windows Secure Boot Key Creation and Management Guidance". The page is dated 10/14/2016 and is attributed to Vishal Manan and Arie van der Hoeven. The content describes the document's purpose of guiding OEMs and ODMs in creating and managing Secure Boot keys and certificates in a manufacturing environment, specifically addressing Platform Keys (PKs), secure firmware update keys, and third-party Key Exchange Keys (KEKs). The page includes a "Hardware Dev Center" navigation bar and a "Developer resources" dropdown.

# 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



The screenshot shows a Microsoft Edge browser window with the following details:

- Title Bar:** UEFI requirements for Windows editions on SoC platforms
- Address Bar:** https://docs.microsoft.com/en-us/windows-hardware/drivers/bringup/uefi-requirements-that-apply-to-all-windows-editions-on-soc-platforms
- Toolbar:** EDK I web, FamilyFun Classic and, Free Hotmail, Home - Circuit [For E, Home - Circuit [For E]
- Navigation Bar:** Technologies, Documentation, Resources
- Content Area:**
  - Section: UEFI requirements for Windows editions on SoC platforms
  - Published: 04/20/2017 • 14 minutes to read • Contributors
  - Description: This topic describes UEFI requirements that apply to Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) and Windows 10 Mobile. For additional requirements that apply only to Windows 10 Mobile, see [UEFI requirements for Windows 10 Mobile](#).
  - Section: Summary of requirements
  - Description: The following table lists all current requirements for UEFI compliance as defined in the UEFI specification (Section 2.6 of the UEFI 2.3.1 specification). In this table, the terms “For Kit Windows Drivers” and “UEFI” refer to the UEFI specification.

# OS BOOTS, NOW WHAT?

UEFI and Management Mode (formerly known as SMM)

# Platform Initialization (PI) Specification Introduces Management Mode (MM)\*\*

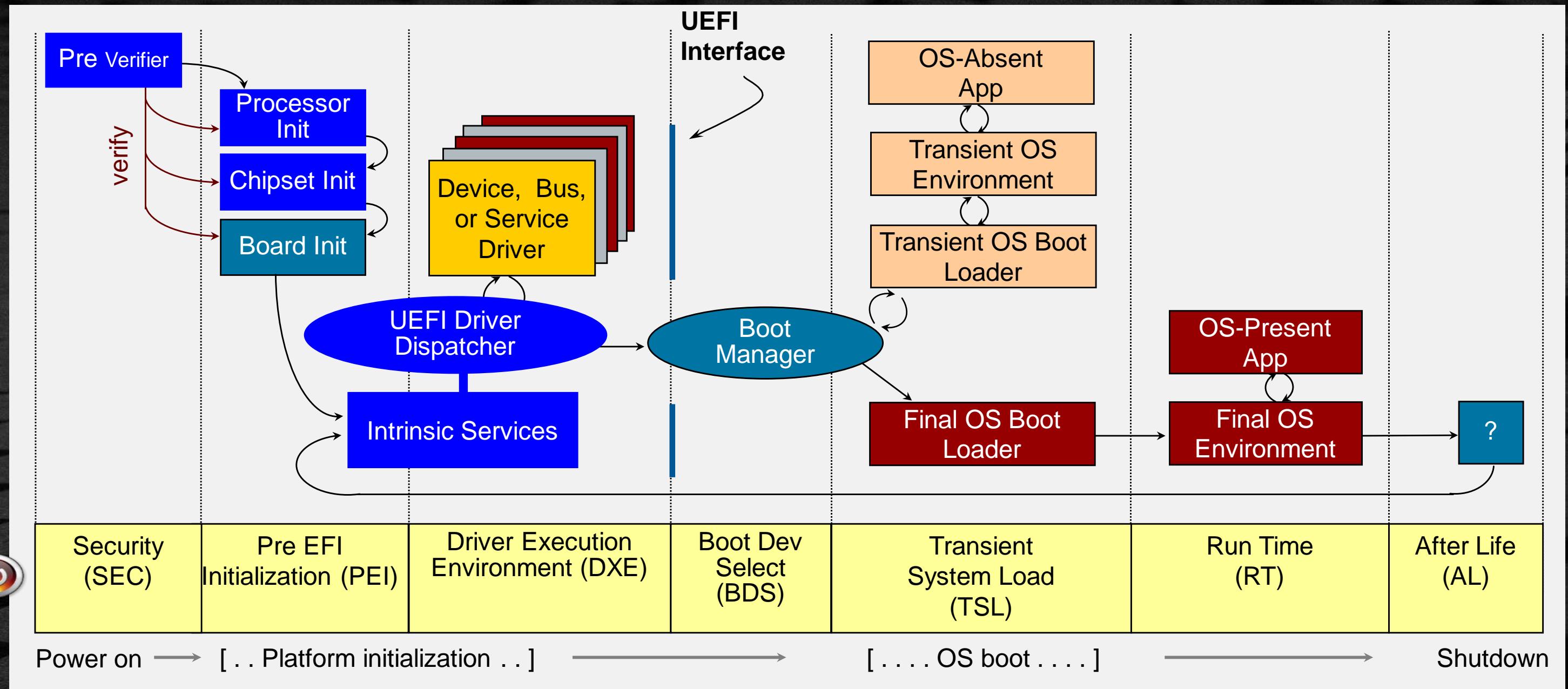
UEFI PI-standard for creating a protected execution environment using hardware resources

- Dedicated, protected memory space, entry point and hardware resources, such as timers and interrupt controllers
- Implemented using SMM (Intel® Architecture) or TrustZone(Arm)
- Highest-privilege operating mode (Ring 0) with greatest access to system memory and hardware resources

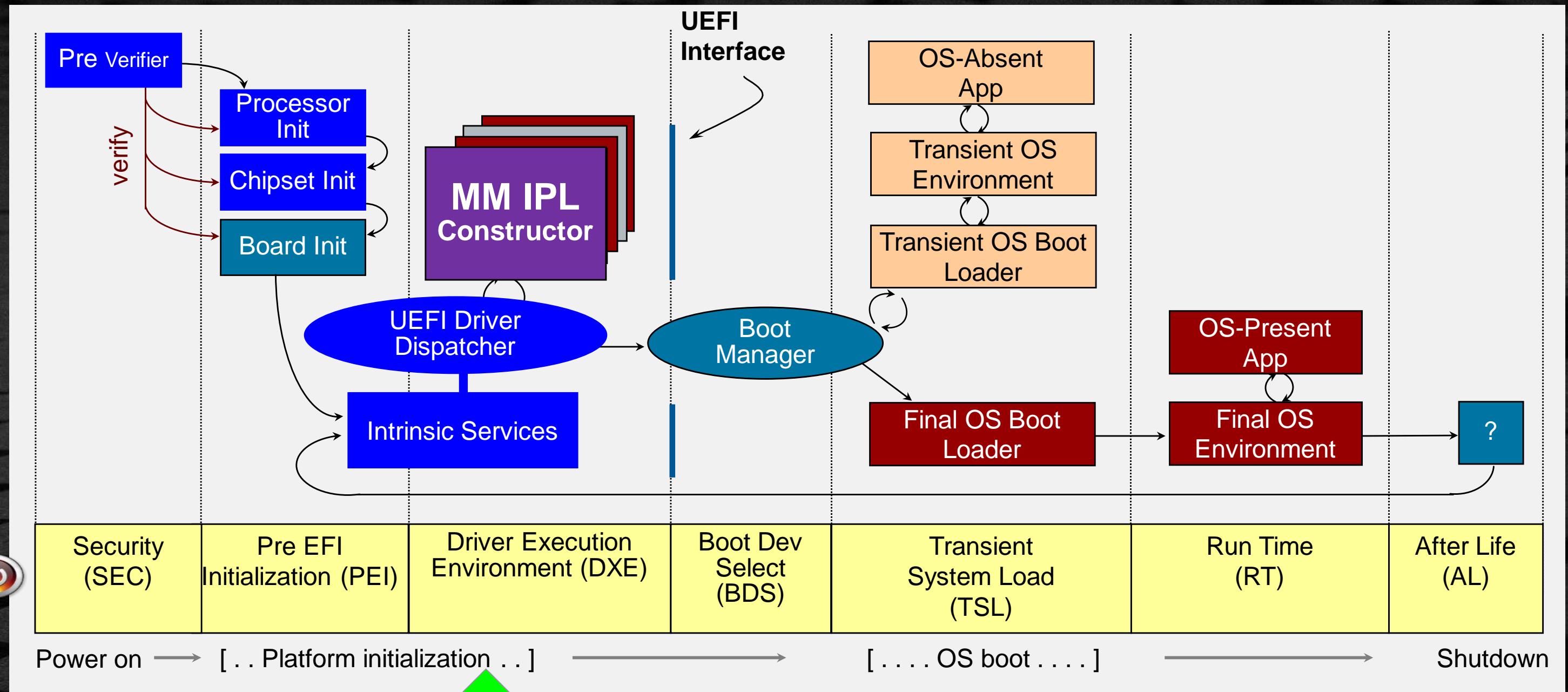
Presented at UEFI Plugfest Fall 2017: [Presentation link](#)

\*\*Formerly known as SMM in PI specification

# UEFI - PI & EDK II Boot Flow - SMM

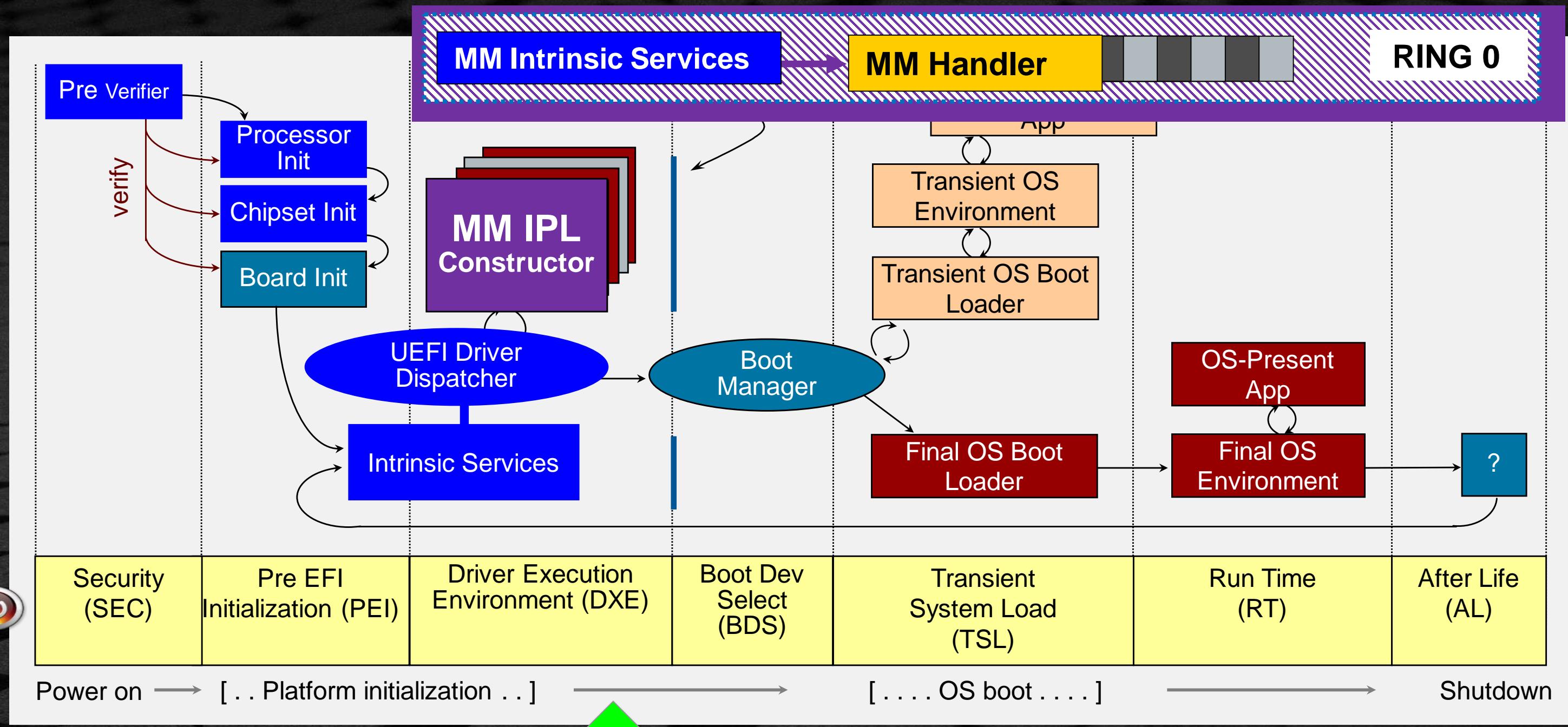


# UEFI - PI & EDK II Boot Flow - SMM



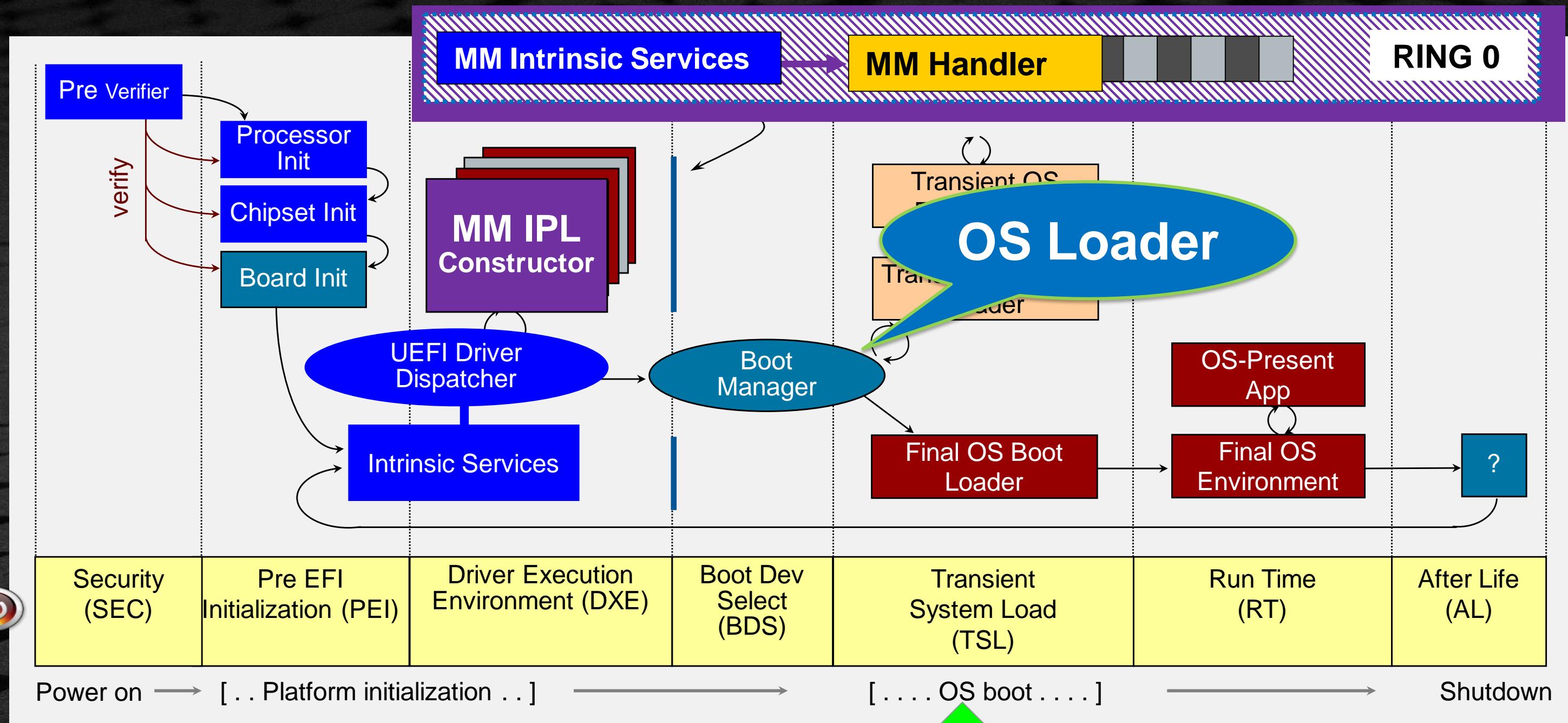
# UEFI - PI & EDK II Boot Flow

- SMM



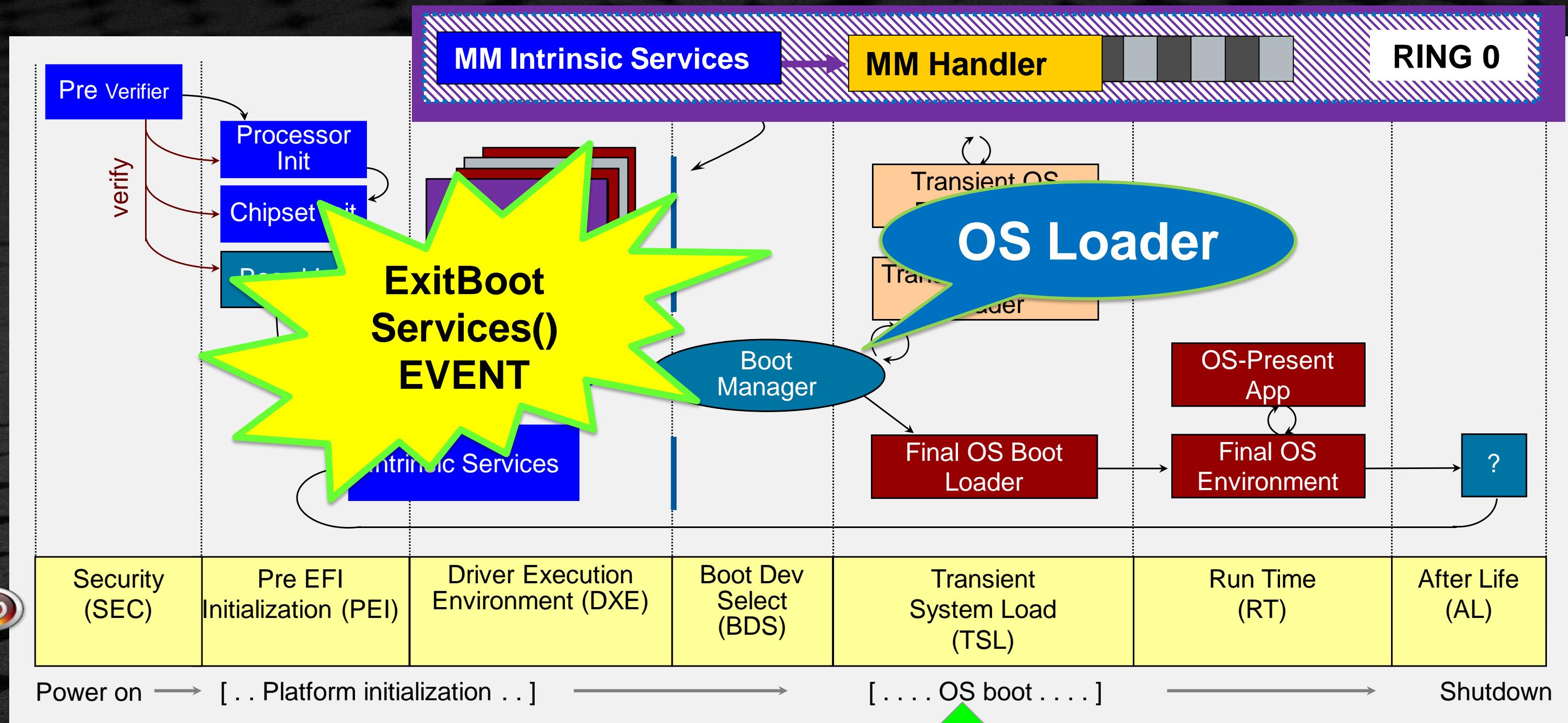
# UEFI - PI & EDK II Boot Flow

- SMM



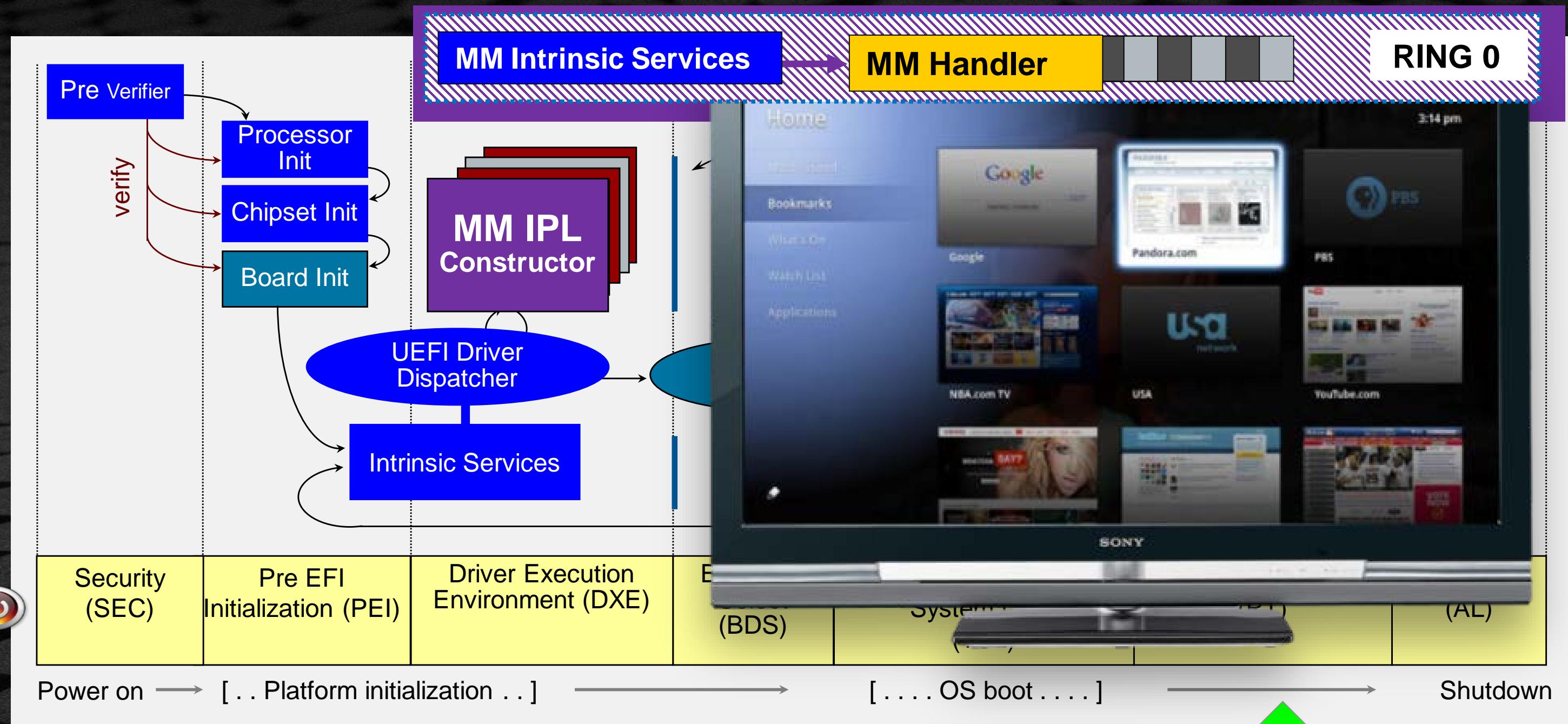
# UEFI - PI & EDK II Boot Flow

- SMM



# UEFI - PI & EDK II Boot Flow

- SMM



# Software Management Mode Interrupts (Software MMIs)

- Management Mode Interrupts generated by software synchronously are called Software MMIs
  - Generated using I/O resources or CPU instructions
- Used to provide firmware services to the
  - OS (ACPI, TPM)
  - OS drivers (device handoff, CPU management)
  - UEFI runtime support (variables, capsule, etc.)
  - BIOS vendor applications (flash utilities, setup access)
  - OEM/ODM applications

# Why are Software MMI Vulnerabilities Dangerous?



# Why are Software MMI Vulnerabilities Dangerous?

Software MMIs can be asked to perform:

- Privileged operations: Flash BIOS, flash EC, write to MMIO, write to MMRAM, etc.
- Overwrite OS code/data
- Copy protected OS data to another unprotected location
- Copy protected firmware data to another unprotected location
- Overwrite BIOS code/data



# UEFI Platform Firmware Assumptions

- Memory protected by the OS cannot be snooped while in use by the OS application or OS driver
  - No protection from MM, VMs or hardware snooping

# UEFI Platform Firmware Assumptions

- Memory protected by the OS cannot be snooped while in use by the OS application or OS driver
  - No protection from MM, VMs or hardware snooping
- Flash protected by hardware cannot be modified outside of MM after the end of DXE
  - Not worried about snooping since no secrets are stored in BIOS
  - Not worried about flash-altering hardware attacks

# UEFI Platform Firmware Assumptions

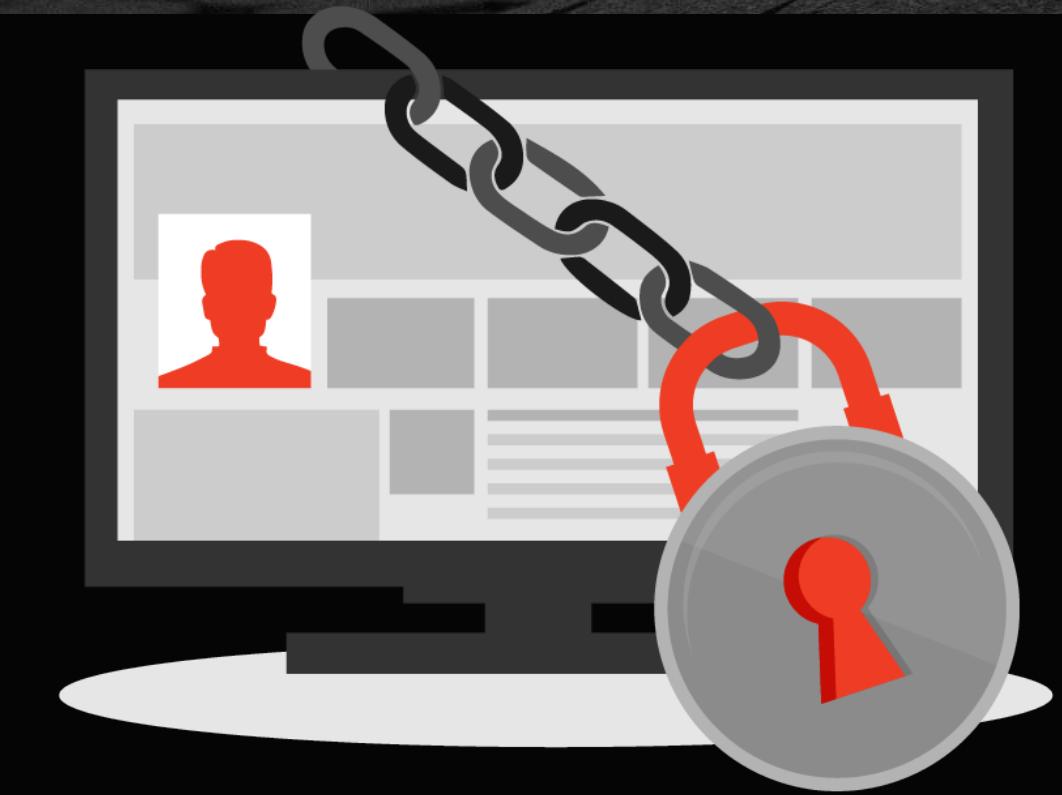
- Memory protected by the OS cannot be snooped while in use by the OS application or OS driver
  - No protection from MM, VMs or hardware snooping
- Flash protected by hardware cannot be modified outside of MM after the end of DXE
  - Not worried about snooping since no secrets are stored in BIOS
  - Not worried about flash-altering hardware attacks
- Software MMIs cause CPUs to enter SMM in SMRAM at a fixed location

# UEFI Platform Firmware Assumptions

- Memory protected by the OS cannot be snooped while in use by the OS application or OS driver
  - No protection from MM, VMs or hardware snooping
- Flash protected by hardware cannot be modified outside of MM after the end of DXE
  - Not worried about snooping since no secrets are stored in BIOS
  - Not worried about flash-altering hardware attacks
- Software MMIs cause CPUs to enter SMM in SMRAM at a fixed location
- MMRAM cannot be altered from outside SMM

# Key Points for More Secure Software MMI Handlers

- Allocate The Buffer In PEI/DXE
- Never Trust That Pointers Point To The Buffer
- Prohibit Input/Output Buffer Overlap
- Don't Trust Structure Sizes
- Verify Variable-Length Data

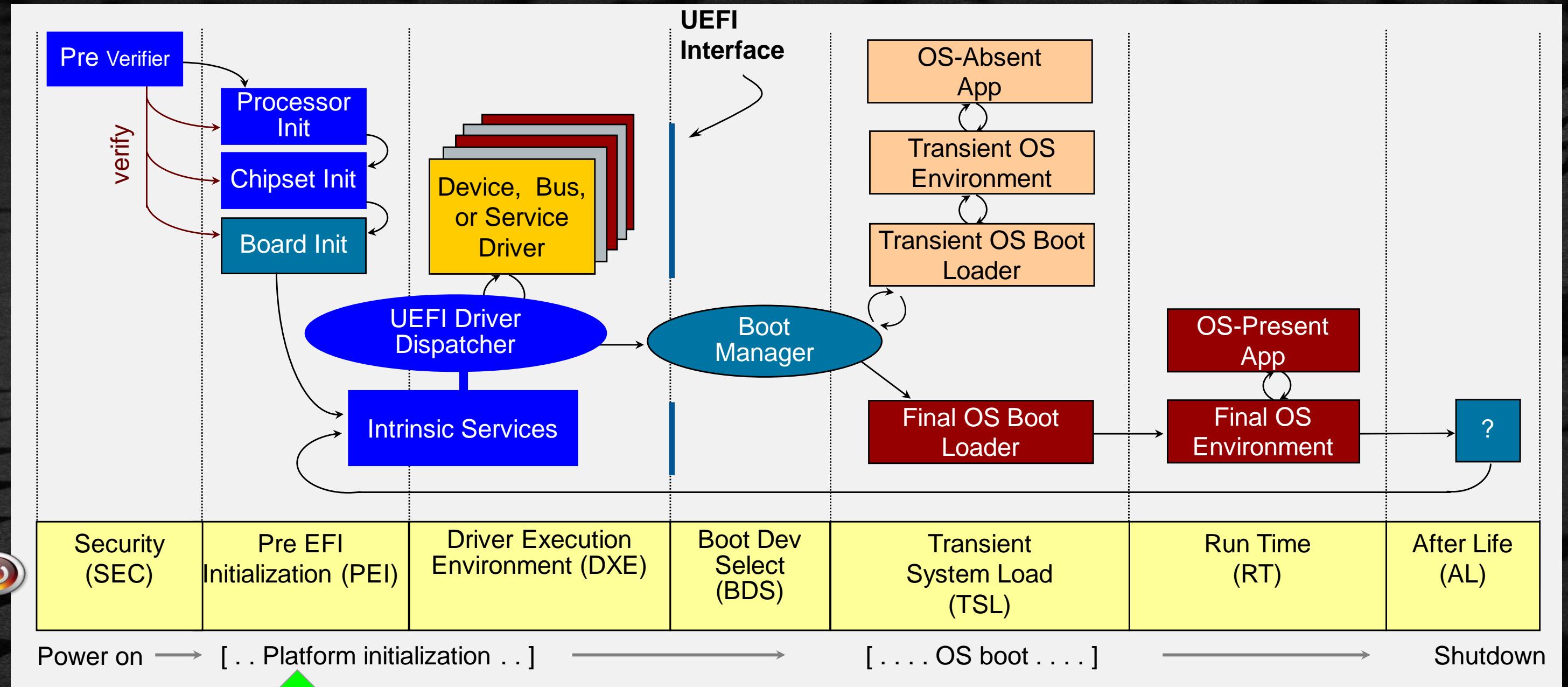


# COREBOOT

How does coreboot work with UEFI

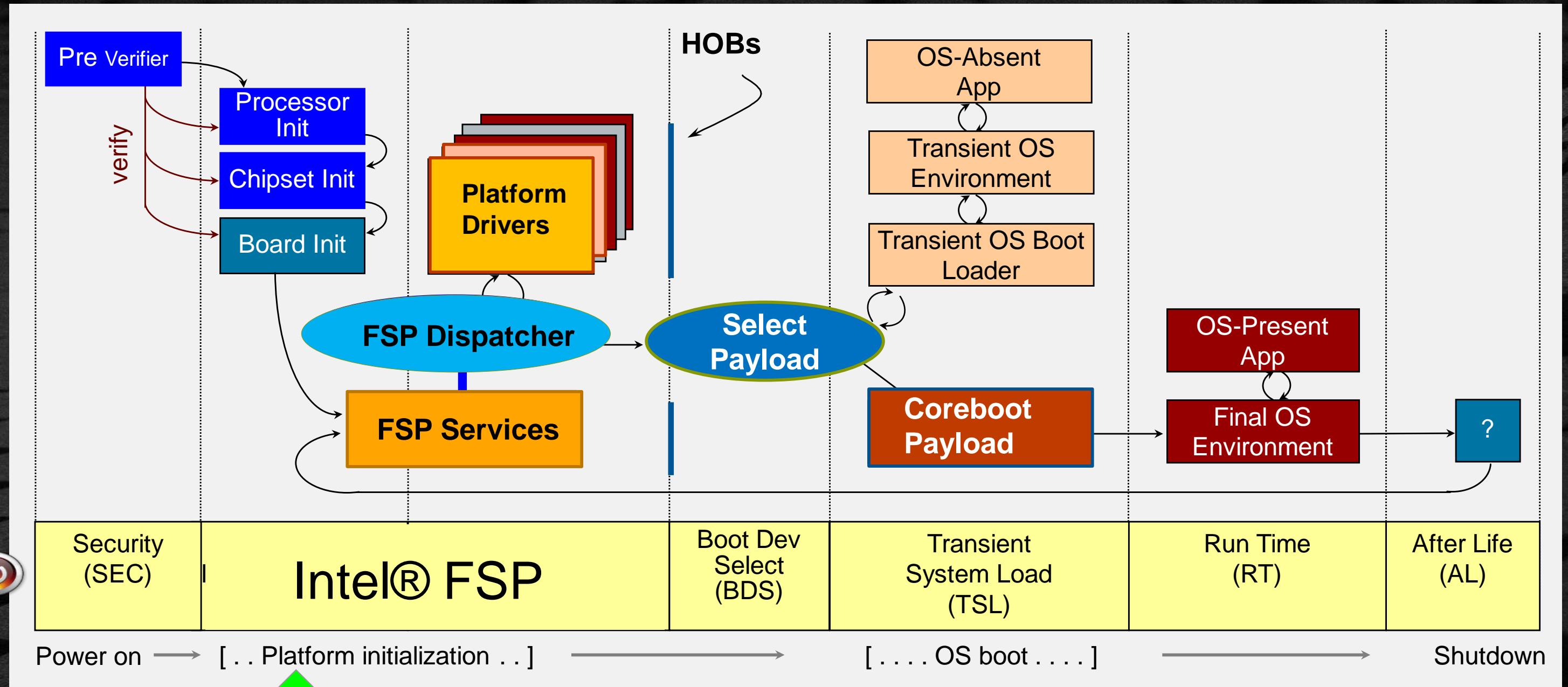
# UEFI - PI & EDK II Boot Flow

- coreboot



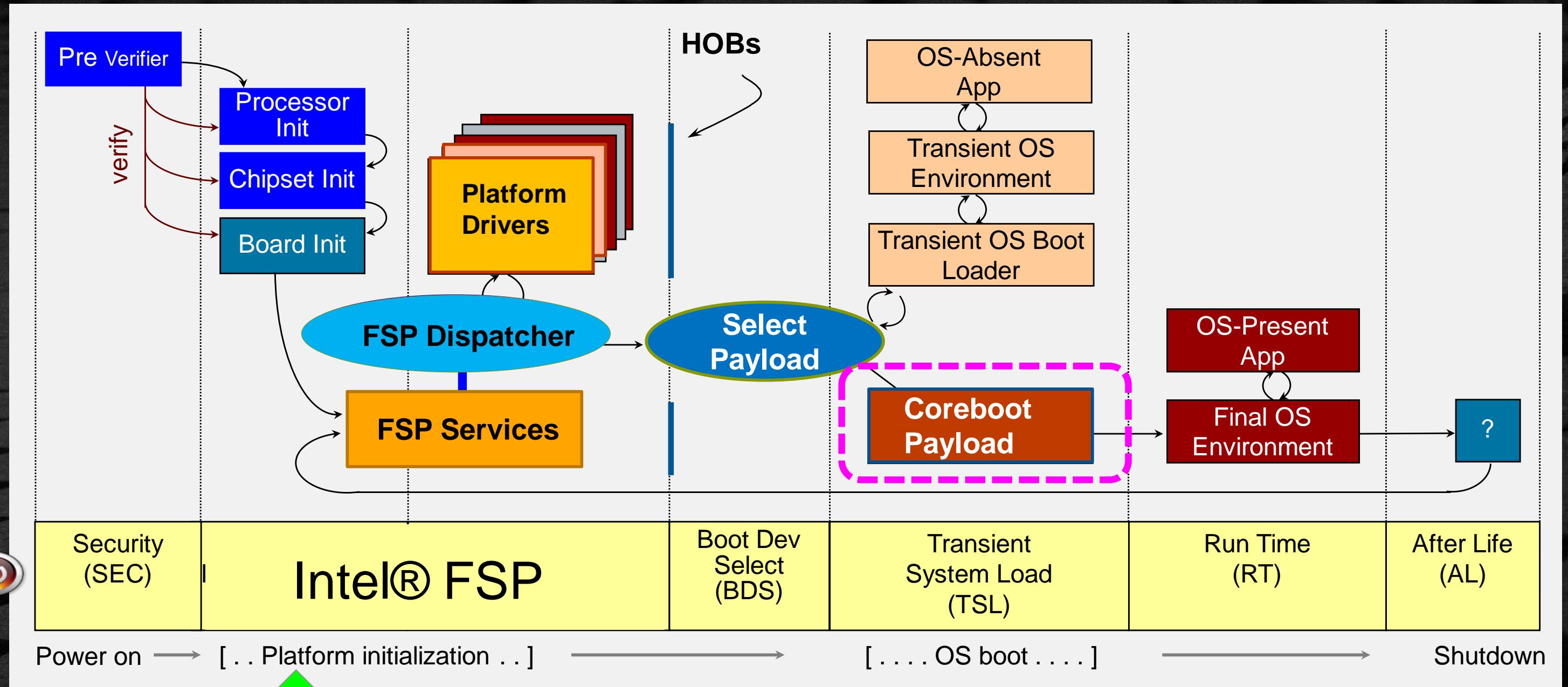
# UEFI - PI & EDK II Boot Flow

- coreboot

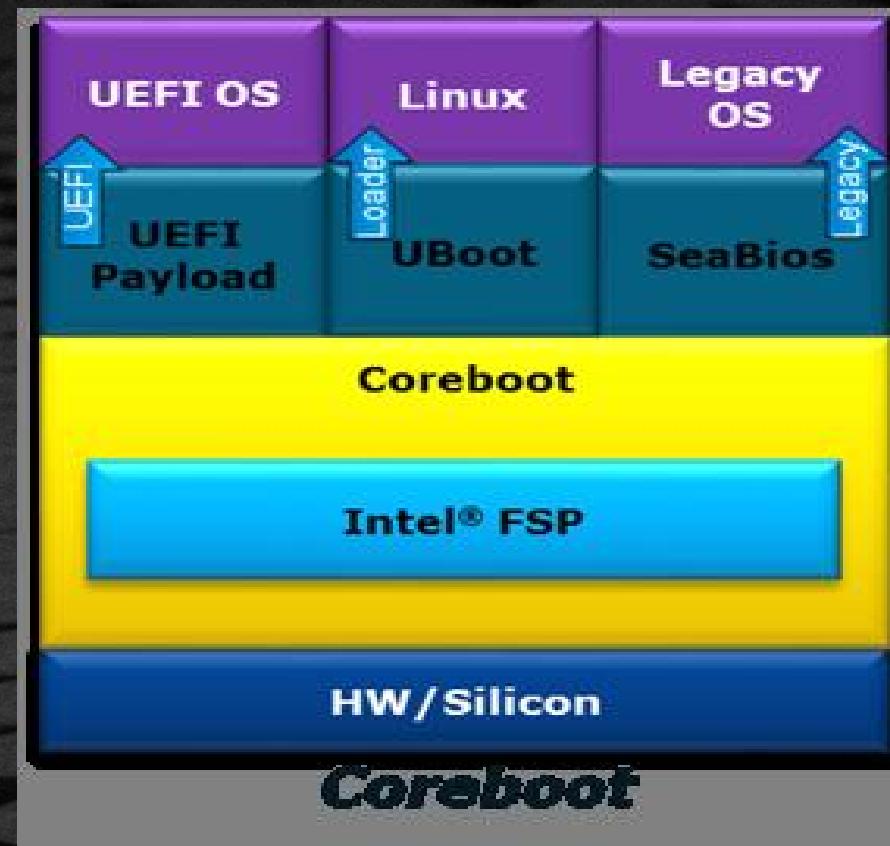


# UEFI - PI & EDK II Boot Flow

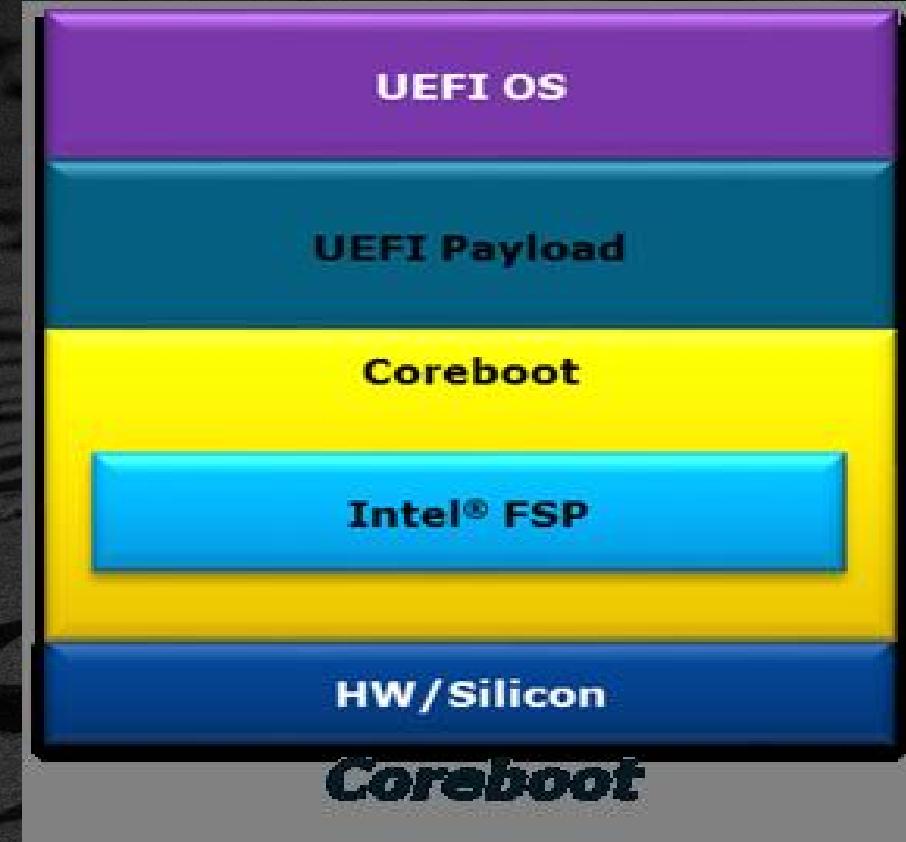
- coreboot



# Coreboot Payload



Coreboot in itself is "only" minimal code for initializing hardware. It does not have the media drivers nor does it provide services required by an OS. After the initialization, it jumps to a payload which can provide services required to load and boot an OS.



UEFI payload provides UEFI services on top of coreboot that allows UEFI OS boot.

# Consumers: EDK II firmware and coreboot

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 

# Demo Windows with Secure Boot



# 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 MM (Formerly Known as SMM)
-  What about coreboot?

# Questions?





# tianocore



# Backup

# UEFI Secure Boot

- 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
- 3<sup>rd</sup> party drivers signed using CA-holding trusted keys
- Trusted signing key's database factory-initialized and OS-updated

# WHY??? SECURE Boot WITH UEFI

**Without**

**Possible corrupted or destroyed data**

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



**With**

- Data integrity
- Trusted boot to OS
- Trusted drivers
- Trusted Applications



# UEFI Secure Boot Flow

