OPEN SOURCE FIRMWARE CONFERENCE 2021 NOV. 30TH & DEC. 1ST

Universal Scalable Firmware

SPEAKER

Vincent Zimmer

Agenda

- Universal Scalable Firmware Overview
- Operation System Interface
- Universal Payload
- Platform Orchestration Layer (POL)
- Scalable Intel® Firmware Support Package (sFSP)
- YAML Format Boot Configuration

Universal Scalable Firmware (USF)

- Simplify & scale firmware development from edge to cloud across all layers of the firmware stack

  Silicon → IP Firmware → system-on-chip (SoC) Abstraction Layer → Platform Orchestration Layer → OS payloads interface

- Modular flexible architecture enables developers and technology partners to accelerate the integration of new silicon and platform technologies

OPEN SOURCE FIRMWARE CONFERENCE

Layers of **Universal** Scalable Firmware (USF)

OS Payload Interfaces

Universal Payload API

Platform Orchestration Layer (POL)

Scalable Firmware Support Package (sFSP) API

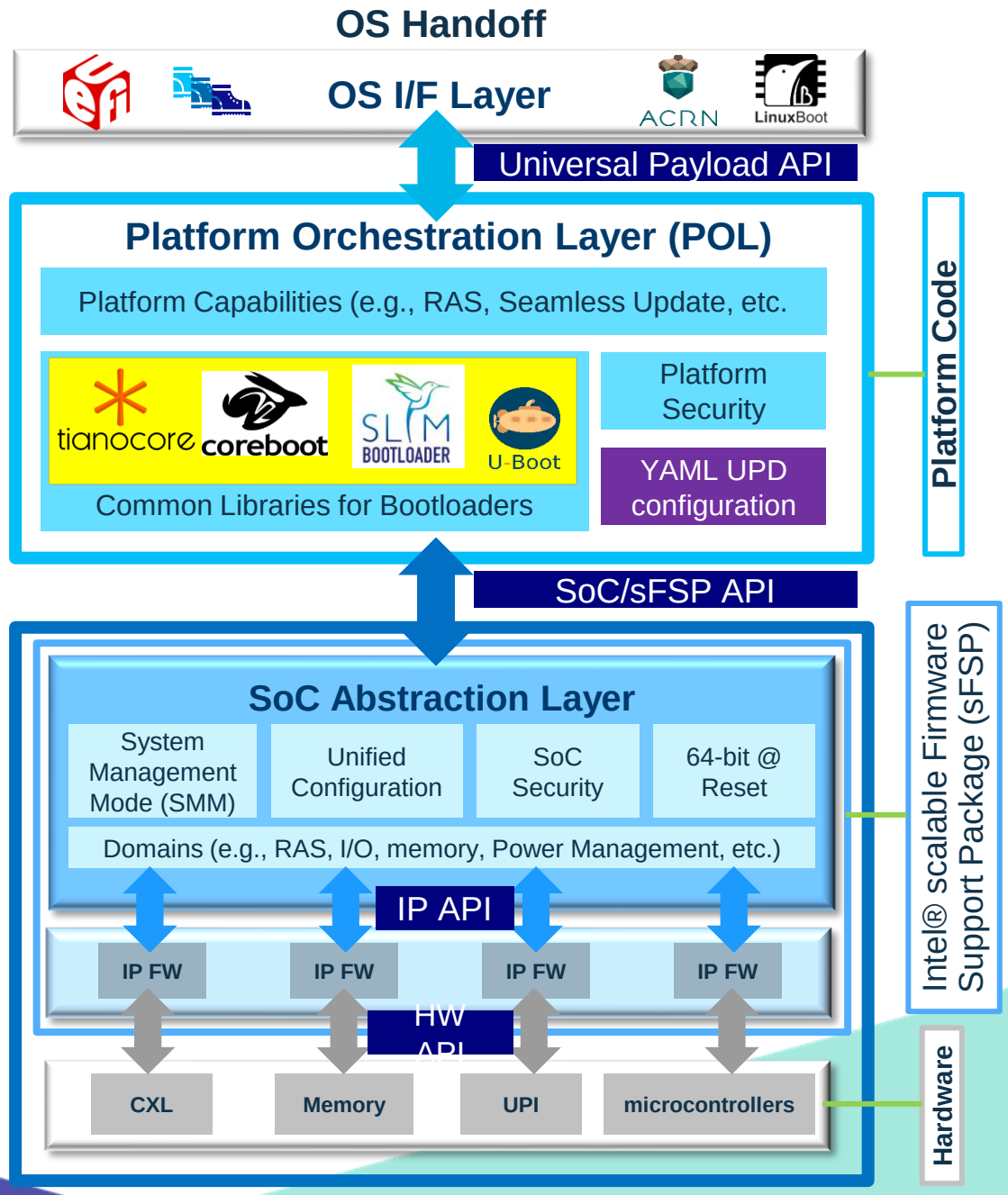
SOC Abstraction Layer(SAL)

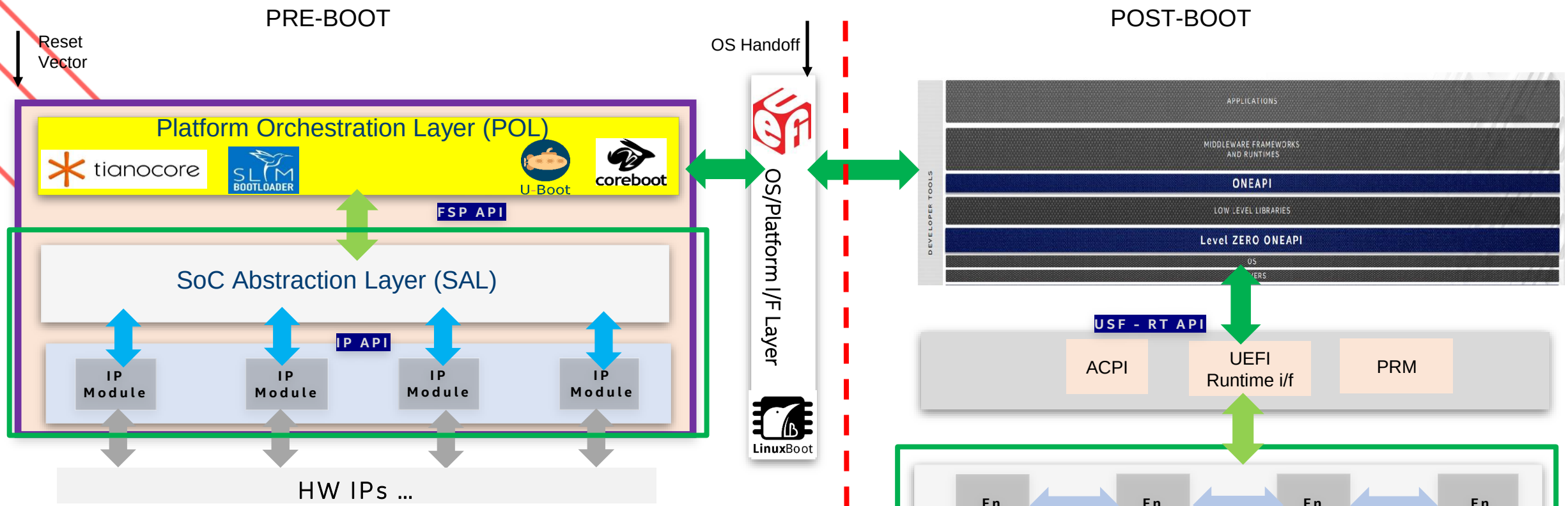
IP API

IP Firmware Interface

HW API

HW Interface





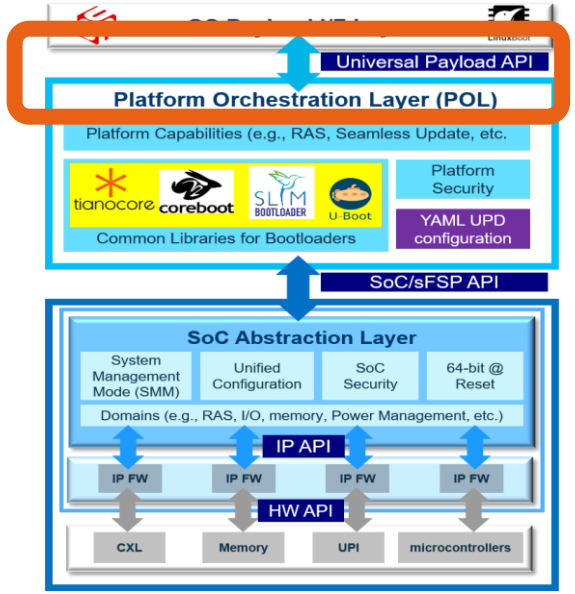
- POL - Orchestrates boot from Reset vector to Payload launch.
- Invokes USF API to perform Si Init based on Boot milestones

- SAL - Provides SoC/Platform level stitching (e.g Memory Map etc).
- Exposes APIs for POL

- FW IP Modules – Initialize SoC Independent IP building blocks
- Binds to the HW IP and moves with the HW IP

IP API | SoC API | OS API | HW API

Universal Payload

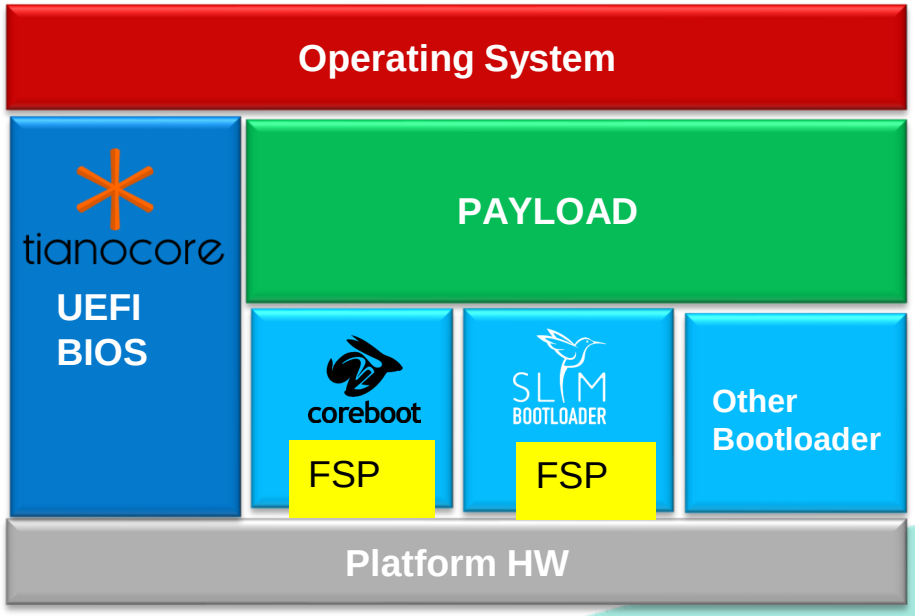


Bootloader

- Platform initialization including memory, silicon, GPIO, ACPI, etc.
- coreboot, Slim Bootloader (SBL), Uboot

Payload

- Boot media initialization, file system, OS boot, etc.
- EDK II UEFI Payload, LinuxBoot, Uboot



Payload is part of boot firmware to initialize boot media and boot OS

Goal 1: Platform Independent

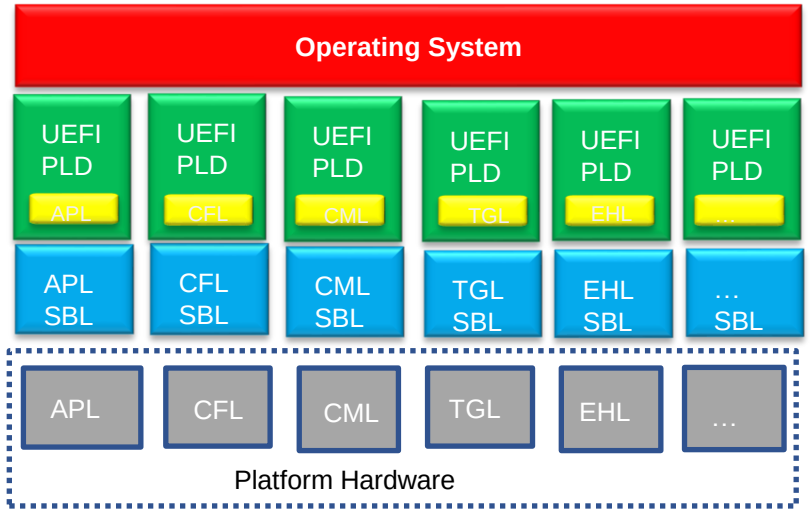


Fig1. Before Change

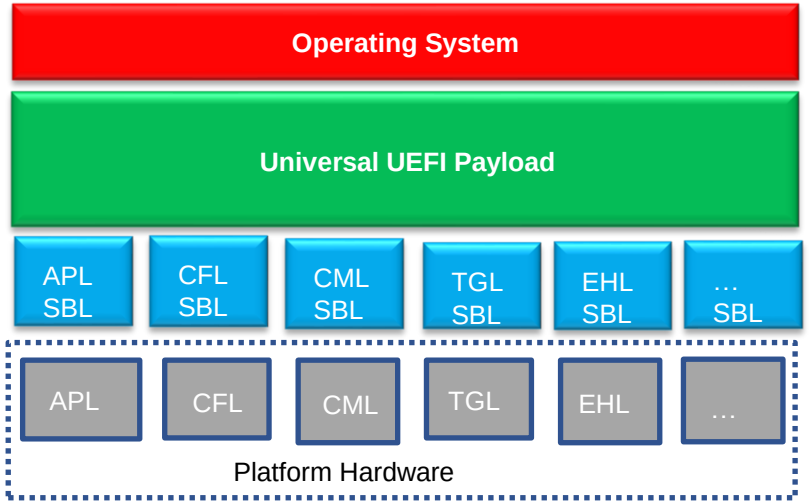


Fig2. After Change

Universal payload is platform independent

Goal 2: Bootloader independent

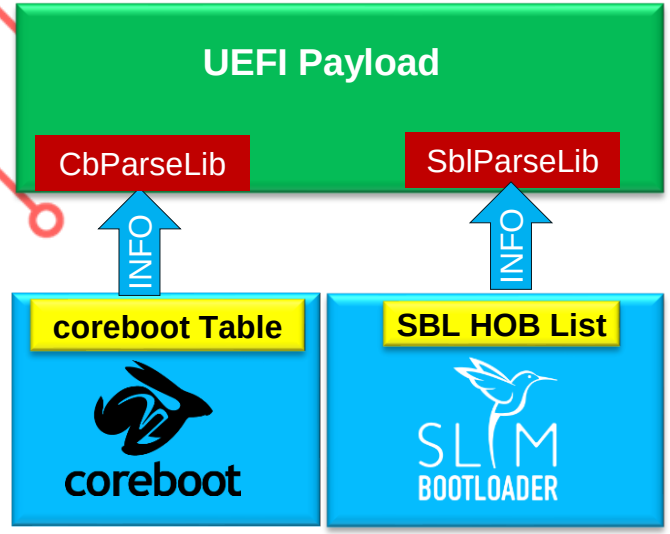


Fig1. Current status

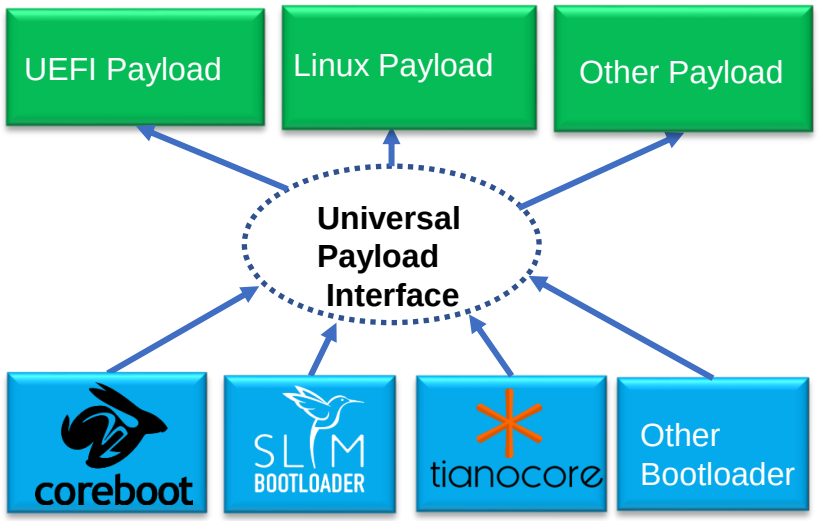


Fig2. Expected goal

Universal payload is bootloader independent

Universal Payload Image Format Sections

- Universal Payload Information Section
  - Have section name defined as ".upld_info"
  - Have section aligned at 4-byte boundary within the ELF image.
  - Contain `UNIVERSAL_PAYLOAD_INFO` structure in its section

Format is using ELF (Executable and Linkable Format)

1. Universal Payload Information Section
2. Universal Payload Loaded Image Section
3. Optional universal payload extra image sections with unique section name ".upld.*"

Universal Payload Entry Point

- The prototype of payload entry point

```
typedef
void
(*PAYLOAD_ENTRY) (
    EFI_HOB_HANDOFF_INFO_TABLE            *HobList
    );
```

- The Hand Off-Block (HOB) is passed to the payload entry

Universal Payload hand-off

Payload Hand-Off State

- Memory and silicon initialized
- PCI enumeration complete
- Stack
- Interrupt
- Registers
- Page table
- …

Payload Hand-Off Block List

HOBs in the hoblist

- ACPI Table HOB
- SMBIOS Table HOB
- Device Tree HOB
- Resource Descriptor HOB
- Graphics Information HOB
- Serial Information HOB
- Cpu Information HOB
- Would add more HOBs for advanced features.

Hand-Off Block (HOB) List

Required HOB Type	Usage
Phase Handoff Information Table (PHIT) HOB	This HOB is required.
One or more Resource Descriptor HOB(s) describing physical system memory	The DXE Foundation will use this physical system memory for DXE.
Boot-strap processor (BSP) Stack HOB	The DXE Foundation needs to know the current stack location so that it can move it if necessary, based upon its desired memory address map. This HOB will be of type EfiConventionalMemory
One or more Resource Descriptor HOB(s) describing firmware devices	The DXE Foundation will place this into the GCD.
One or more Firmware Volume HOB(s)	The DXE Foundation needs this information to begin loading other drivers in the platform.
A Memory Allocation Module HOB	This HOB tells the DXE Foundation where it is when allocating memory into the initial system address map.

Universal Payload Info Structure

```
typedef struct {
  UINT32     Identifier;
  UINT32     HeaderLength;
  UINT16     SpecRevision;
  UINT8      Reserved[2];
  UINT32     Revision;
  UINT32     Attribute;
  UINT32     Capability;
  CHAR8      ProducerId[16];
  CHAR8      ImageId[16];
} UNIVERSAL_PAYLOAD_INFO_HEADER;
```

Byte Offset	Size in Bytes	Field	Description
0	4	Identifier	‘PLDH’ Identifier for the universal payload info.
4	4	HeaderLength	Length of the structure in bytes.
8	2	SpecRevision	Indicates compliance with a revision of this specification in the BCD format. 7 : 0 - Minor Version 15 : 8 - Major Version For revision v0.75 the value will be 0x0075.
12	4	Revision	Revision of the Payload binary. Major.Minor .Revision.Build The ImageRevision can be decoded as follows: 7 : 0 - Build Number 15 :8 - Revision 23 :16 - Minor Version 31 :24 - Major Version
16	4	Attribute	Bit-field attribute indicator of the payload image. BIT 0: Build Type. 0: Release Build 1: Debug Build
20	4	Capability	Bit-field capability indicator that the payload image can support. BIT 0: Support SMM rebase
24	16	ProducerId	A null-terminated OEM-supplied string that identifies the payload producer.
40	16	ImageId	A null-terminated ASCII string that identifies the payload name.

Payload Execution Environment Intel® 64 and IA-32 Architectures

- Executes on Bootstrap Processor (BSP)
- 32bit protected or 64bit long-mode
- Registers
  - HOBS Pointer in Registers:
    - ESP+4 for 32bit
    - RCX for 64bit
  - EFLAGS – Direction Flag clear
  - Floating Point Control = 0x027F
  - MMX control word = 0x1f80
    - All exceptions masked
  - CR0.EM is clear
  - CR0.TS is clear

Interrupts disabled
Page Table
Selectors set to flat
32bit may have paging mode
64bit Paging mode enabled
All memory space is identity mapped
Stack – 4KB for payload
Payload may use its own stack
Application Processors (AP)– in halt state

Are payloads new?

- From 2015 page 56, 139
  https://link.springer.com/book/10.1007/978-1-4842-0070-4

- From OSFC 2020
  https://cfp.osfc.io/osfc2020/talk/VUNDSC/

- From coreboot practice 1999+

The Philosophy of coreboot

coreboot is built on the belief that users and vendors deserve an open, fast, customizable, and purpose-built firmware for silicon and mainboard initialization. coreboot is designed to do critical hardware initialization before passing control to a payload.

CHAPTER 6 ■ INTEL FSP AND UEFI INTEGRATION

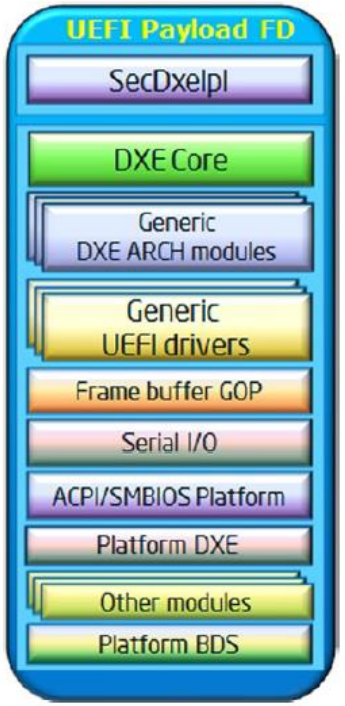
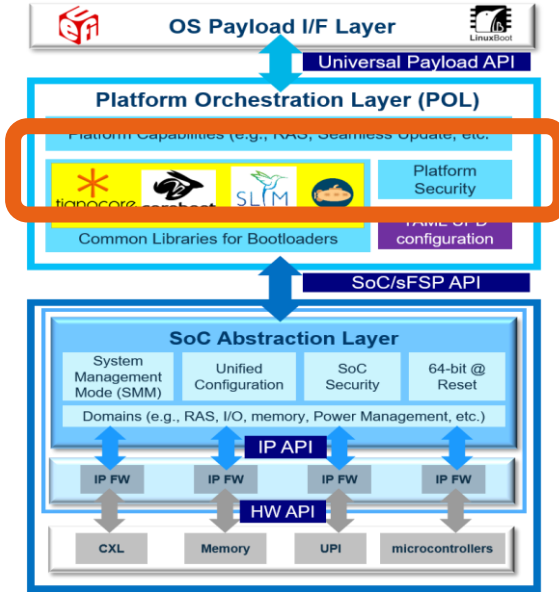


Figure 6-17. UEFI payload based upon EDK II

Platform Orchestration Layer (POL)



Design Principles

Providing vendor specific features and mainboard specific initialization

Compatibility	• Configuration to/from sFSP
Portability	• Between different open-source platforms, EDK II, coreboot, U-Boot, Etc.
Determinism /Simplicity	• Abstract SOC initialization & have simple boot flow

POL Boot Flow Stages w/ Bootloaders

coreboot tianocore SLIM BOOTLOADER

Early Initialization

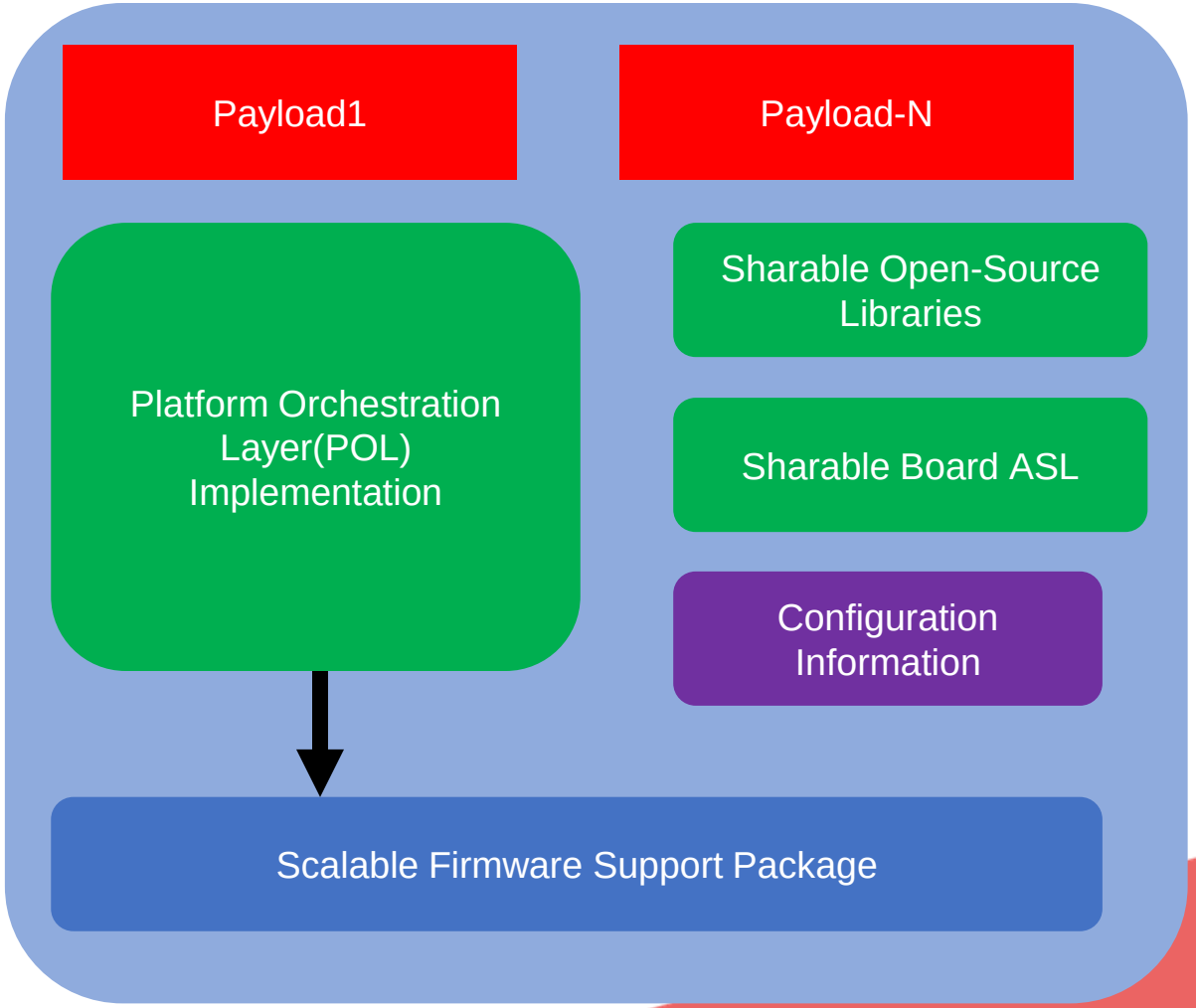
- coreboot romstage
- EDKII SEC/PEI
- Slim bootloader stage 1

Late Initialization

- coreboot ramstage
- EDKII DXE
- Slim bootloader stage 2

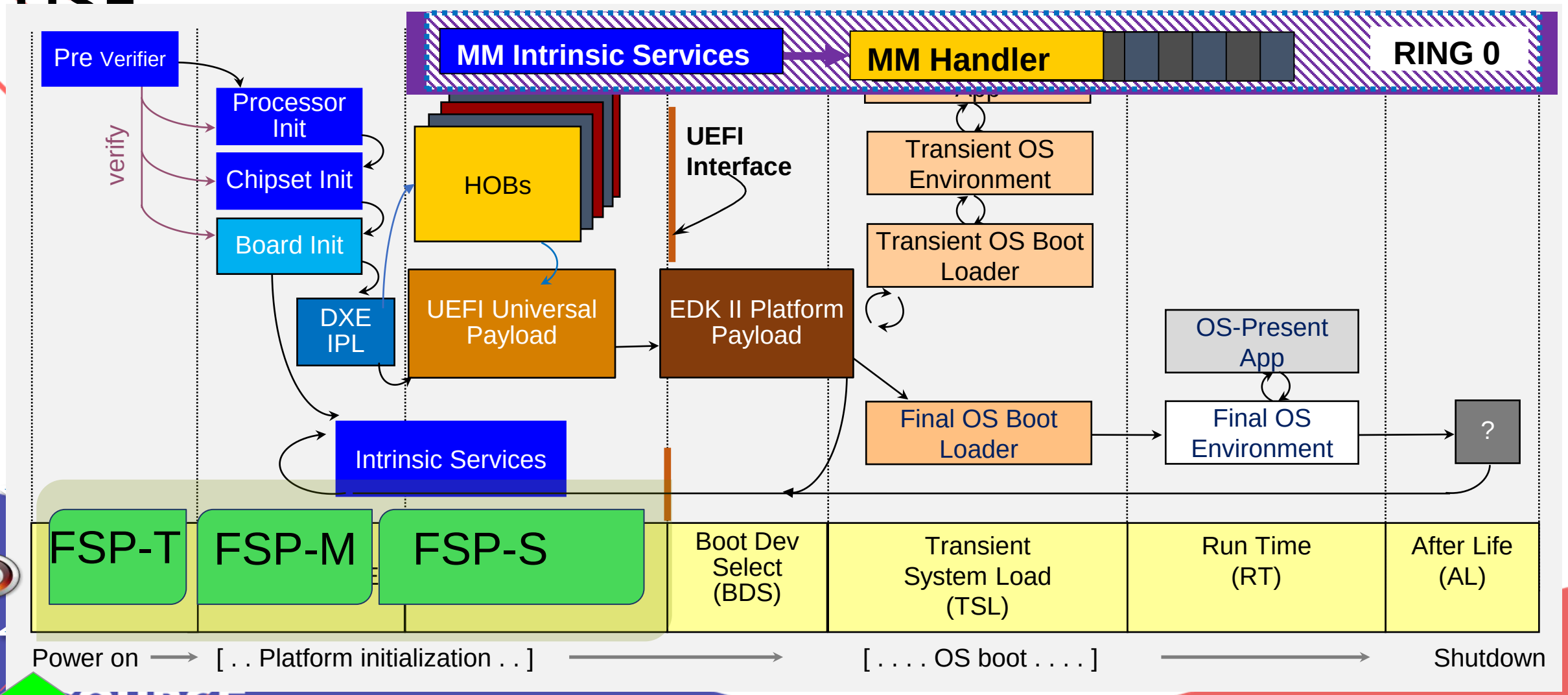
Runtime

- OEM SMM
- ACPI

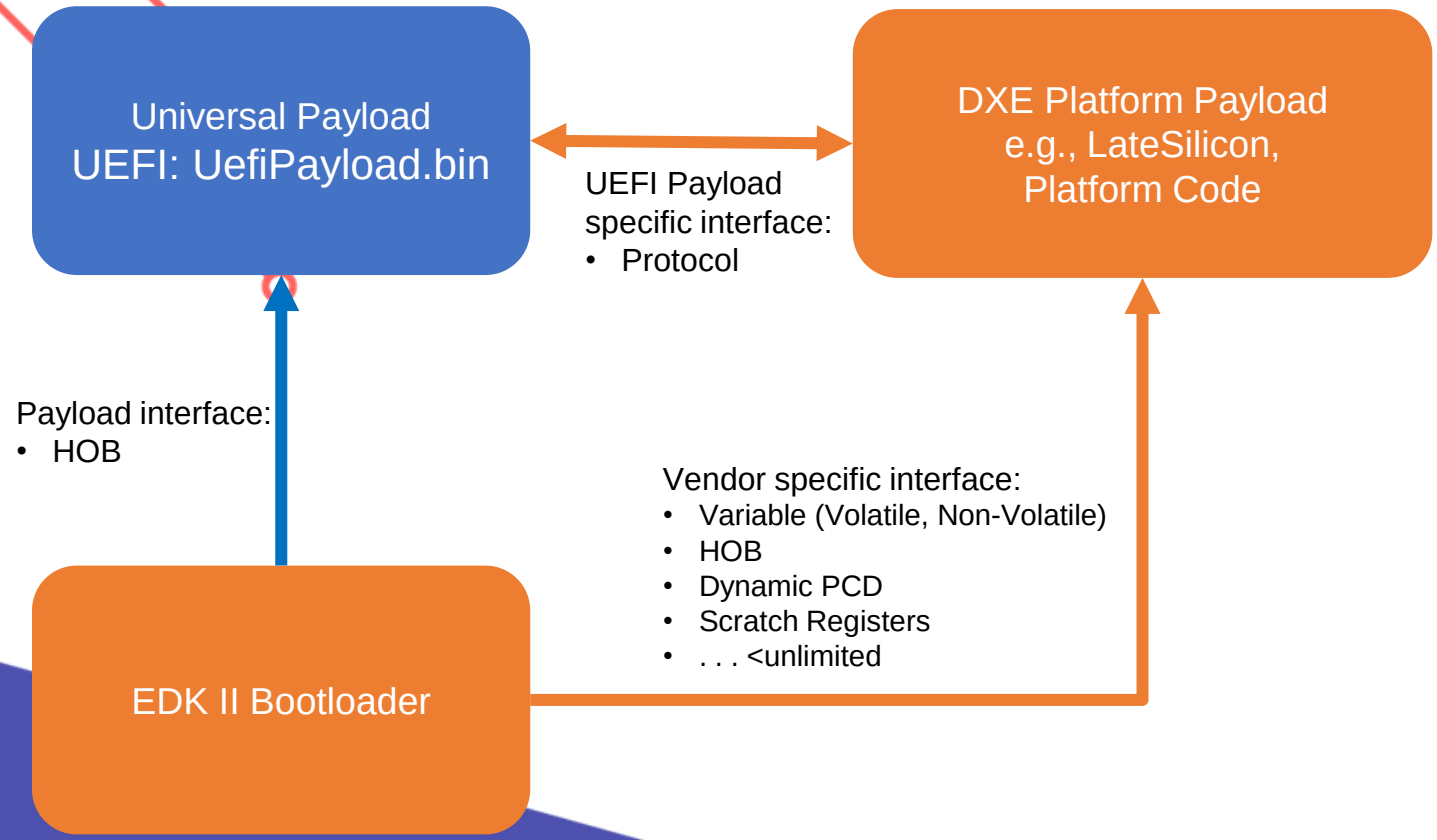


POL High Level Architecture

UEFI – PI & EDK II Boot Flow – Intel® FSP w/ UCE



UEFI – PI & EDK II Data Flow w/ Universal Payload



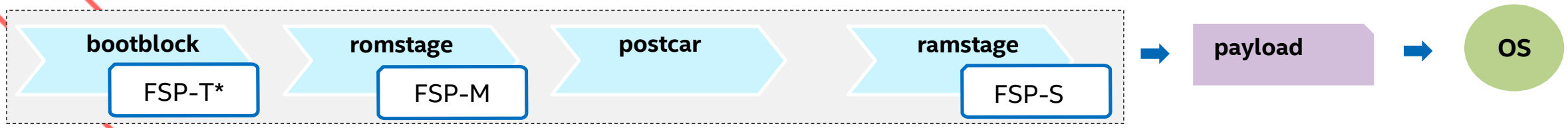
Universal Payload,
EFI Platform Payload and
EDK II Bootloader

EDKII Bootloader passes SOC and platform information through HOBs to UEFI Universal Payload.

UEFI Universal Payload interacts with EDKII UEFI Platform Payload through Protocols.

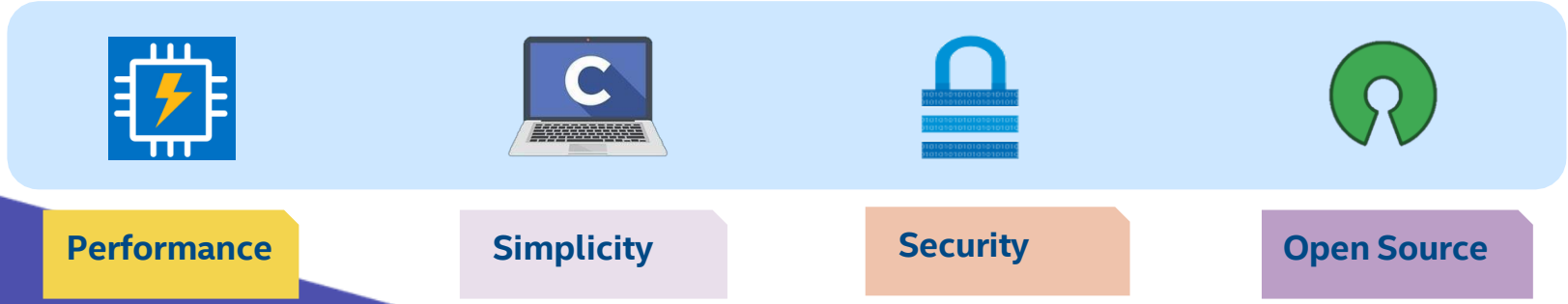
EDKII Bootloader can use any mechanism to pass information to EDKII UEFI Platform Payload since both are owned by the platform vendor.

- Core (essential, stripped-down firmware) boot (to boot the platform)



- Idea: Perform basic hardware initialization before passing control to a payload** that boots the OS***

- Principles:



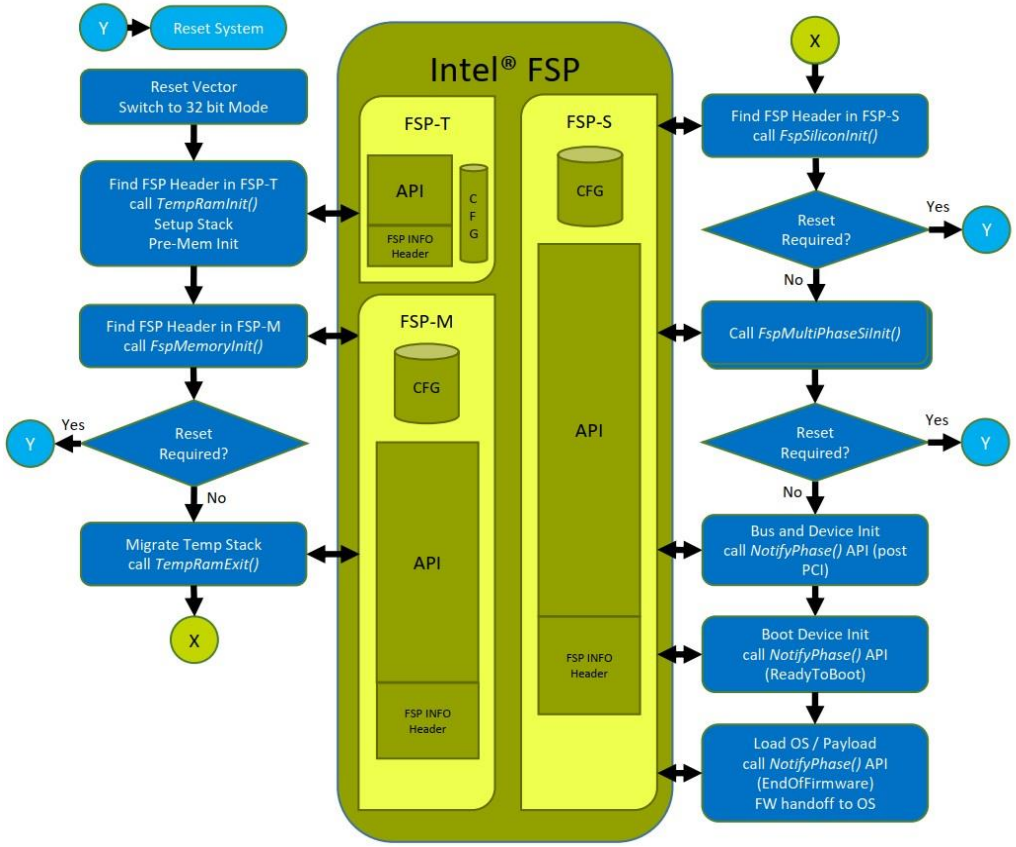
*FSP-T: Optional for Chrome Platform

**Payload: Depth charge, Linux Boot, Tiano Core, SeaBios .... Almost all payload are compatible with coreboot

***OS: Any market standard operating system

coreboot boot flow using Intel® FSP v2.3

- coreboot owns reset vector
- coreboot contains the real mode reset vector handler cod
- Optionally coreboot can call FSP-T for CAR setup and create stack
- coreboot to fill in required UPDs before calling FSP-M for memory
- On exit of FSP-M, coreboot to tear down CAR and do the required Silicon programming including filling up UPDs for FSP-S before calling FSP-S to initialize Chipset
- If supported by the FSP and the bootloader enables multi-phase silicon initialization by setting FSPS_ARCH_UPD.EnableMultiPhaseSiliconInit to a non-zero value:
- On exit of FSP-S, coreboot to perform PCI enumeration and resource allocation.
- Bootloader calls the FspMultiPhaseSiInit() API with the EnumMultiPhaseGetNumberOfPhases parameter to discover the number of silicon initialization phases supported by the bootloader.
- Bootloader must call the FspMultiPhaseSiInit() API with the EnumMultiPhaseExecutePhas parameter n times, where n is the number of phases returned previously. Bootloader may perform board specific code in between each phase as needed.
- The number of phases, what is done during each phase, and anything the bootloader may need to do in between phases shall be describes in the Integration Guide.
- coreboot calls NotifyPhase at proper stages before handing over to payload.



Coreboot

FSP

Slim bootloader Boot Stages

- Stage 1A
  - Reset Vector stage starts with assembly code
  - Basic initialization including setting up temporary memory, debug output
- Stage 1B
  - Memory initialization stage
  - Loads configuration data

Stage 2

Post Memory stage

Silicon initialization

ACPI, PCI Enumeration, etc

OsLoader / FWU Payload

OS boot logic

Media drivers

Universal Payloads & Bootloader Payload Interfaces

UEFI / EDK II Payload

- Provides UEFI Architectural Protocols
- Uses UEFI HOB
- POC: https://github.com/universalscalablefirmware/edk2/tree/universal_payload

Slim Bootloader OS Loader

- Supports Linux Boot Protocol, ELF, PE and Multi-Boot
- uses UEFI HOB
- POC: https://github.com/universalscalablefirmware/slimbootloader/tree/universal_payload

Linux Payload

- Like UEFI DXE with Linux kernel https://www.linuxboot.org
- POC: https://github.com/universalscalablefirmware/linuxpayload

coreboot

- coreboot Tables – comparable to ACPI RSDT or MP Tables
- Similar to UEFI HOB
- POC: https://github.com/universalscalablefirmware/coreboot/tree/universal_payload

Universal payload specification is open sourced as part of main USF specification

https://github.com/universalscalablefirmware/documentation

Specification HTML version

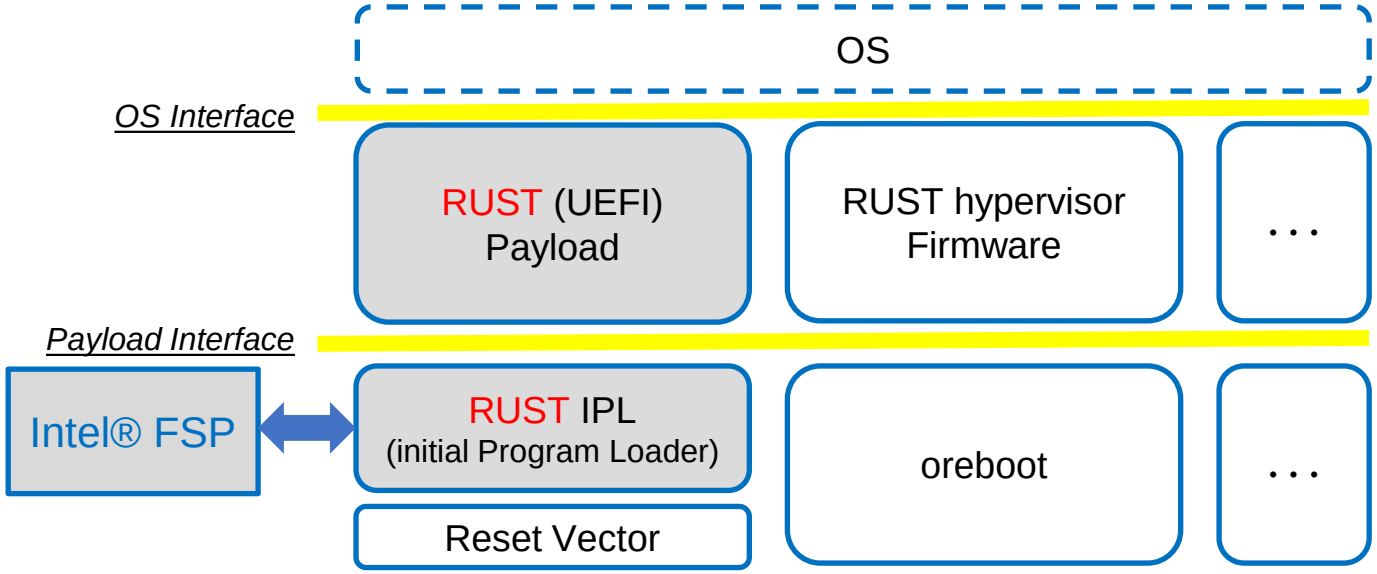
https://universalscalablefirmware.github.io/documentation/

Tools

https://github.com/universalscalablefirmware/tools

Platform Orchestration Layer (POL) – Modern Language

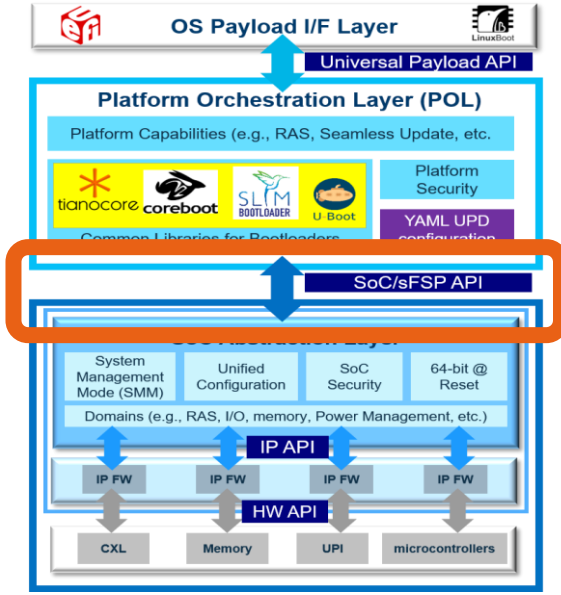
- Modern Language – RUST-based firmware as either a payload or the platform layer implementation



RUST architecture is based upon https://github.com/jyao1/rust-firmware

The RUST API for FSP wrapper is at https://github.com/jyao1/rust-firmware/tree/master/rust-fsp-wrapper

Scalable FSP



Since the Platform Orchestration Layer POL uses Intel® FSP as its SOC abstraction, scalable FSP is an evolution of the Intel® FSP

What is a Firmware Support Package

- Intel® Firmware Support Package (Intel® FSP) includes:
  - A binary file
  - An integration guide
  - A rebasing tool
  - An IDE configuration tool / Boot Setting File (BSF) - USF can use YAML
- Provide silicon initialization code:
  - Initializes processor core, chipset as explained in BIOS Writers' Guide
  - Is relocatable in ROM
  - Can be configured for platform customization
- Boot loader agnostic and can be easily integrated with many options:
  - Open source boot loaders: UEFI –EDK II, coreboot, U-boot, etc.
  - RTOS
  - Others

Firmware Support Package components

- CPU, memory controller, and chipset initialization functions as a binary package
- Provides silicon initialization ingredients
- Plugs into existing firmware frameworks
- Integration guide, includes API documentation

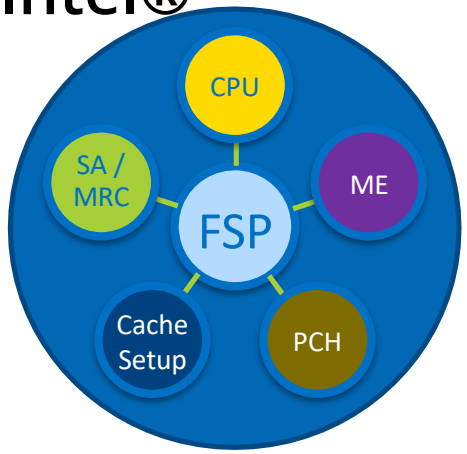
- Intel FSP is currently available for the many Intel hardware-producing divisions
- See: About Intel FSP (Intel® FSP 2.3 July 2021)
- White Paper Example: Open Braswell - Design and Porting Guide

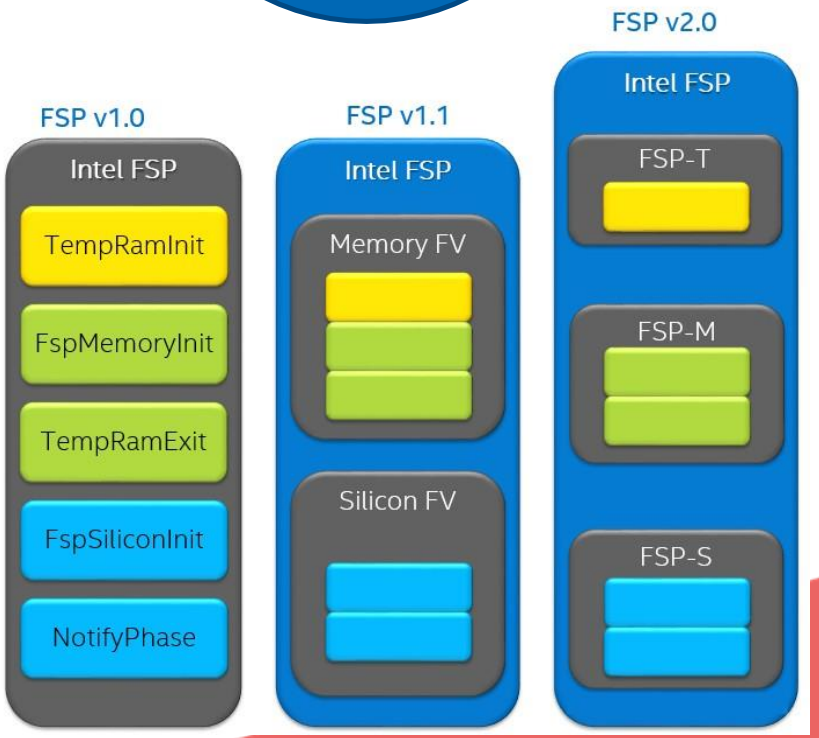
OPEN SOURCE FIRMWARE CONFERENCE

Intel® Firmware Support Package (Intel® FSP)

The Intel ® FSP provides processor & chipset initialization in a format that can easily be incorporated into many existing boot loader frameworks.

- Responsible for most SOC setup
  - Memory, CPU, PCH, GPU, ME, etc
- Configuration via UPD
  - Turned into “Policy” settings for reference code
- Specification evolved from version 1.0 to latest 2.0
- FSP v2.0 has provided with 3 Entry Points
- FSP 2.1 since Aug 2018
  - Incremental improvement over FSP2.0 specification
  - Reduced Temporary RAM usage after using same Bootloader Stack
  - Introduction of Dispatch Mode
- Intel® FSP 2.2 (May 2020)
  - Incremental improvement over FSP2.1 specification
  - Added multi-phase silicon initialization to increase the modularity of the FspSiliconInit() API.

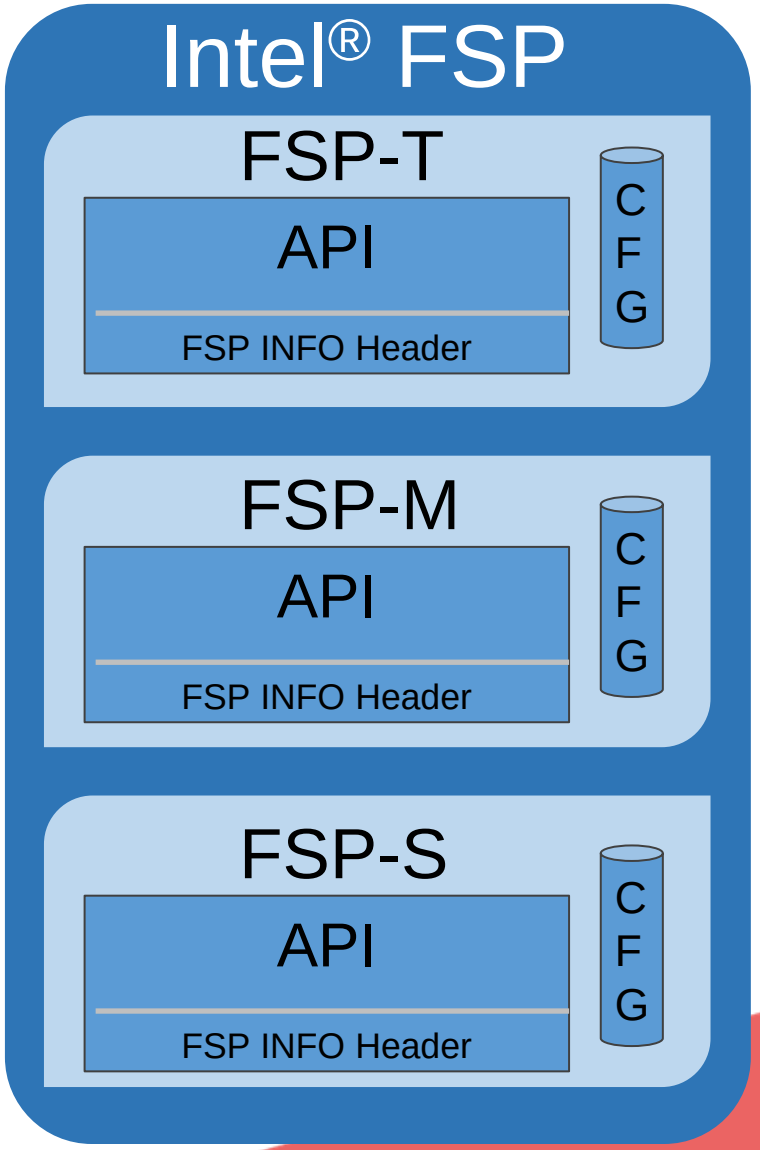




Intel® FSP V2.3 Binary Component View

Firmware Volume Layout of the Intel FSP Binary

- FSP-T: Temporary RAM initialization phase
  - `TempRamInit()`
- FSP-M: Memory initialization phase
  - `FspMemoryInit()`
  - `TempRamExit()`
- FSP-S: Silicon initialization phase
  - `FspSiliconInit()`
  - `NotifyPhase()`
  - `FspMultiPhaseSiInit()`



Intel® FSP Spec 2.3 Figure 1

Intel® FSP 2.Next or FSP 3.0

FSP-V – Validation

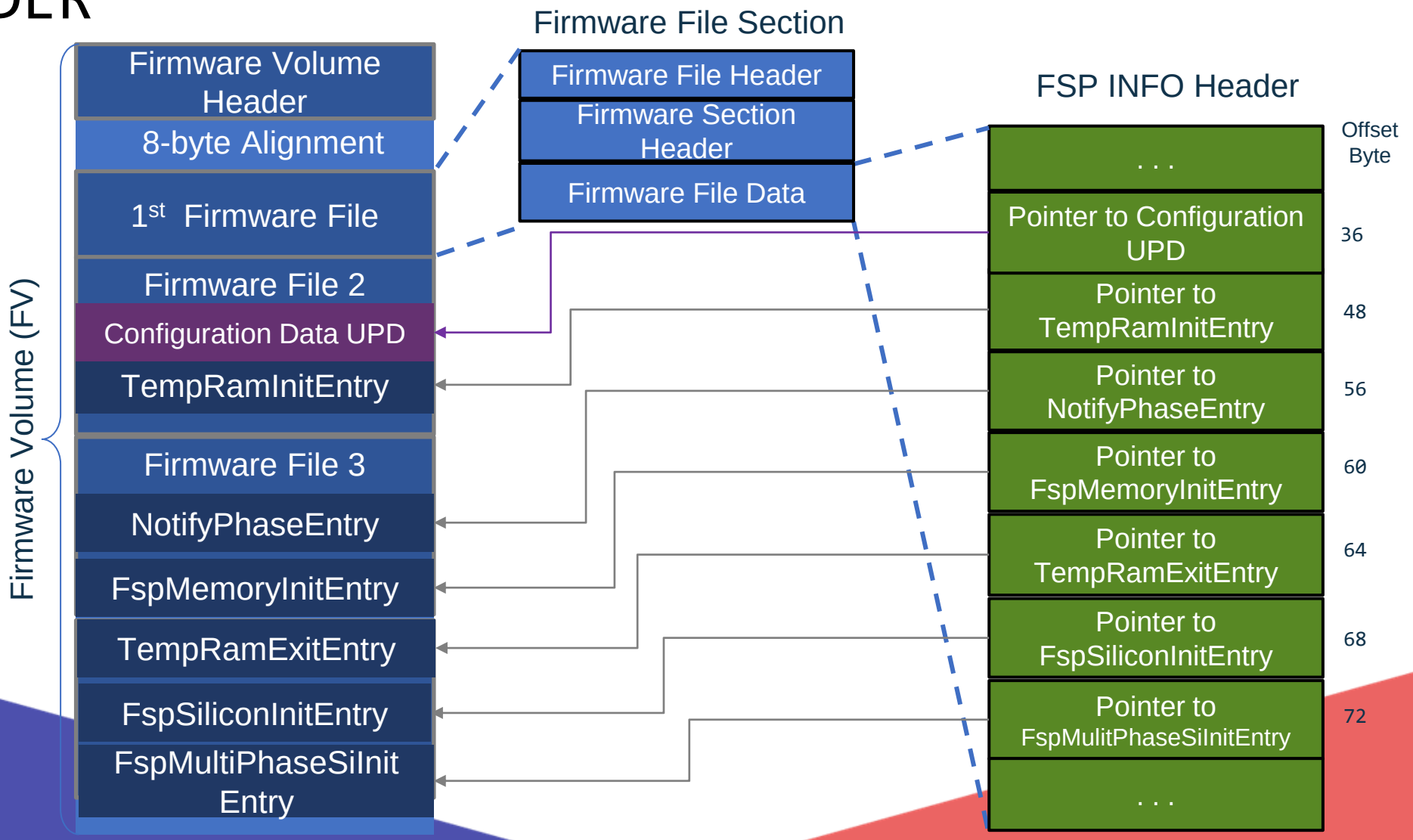
FSP-R - Runtime

Intel® FSP Binary Structure

FSP_INFO_HEADER

FSP INFO Header is the first Firmware File within each of the FSP Component's FV

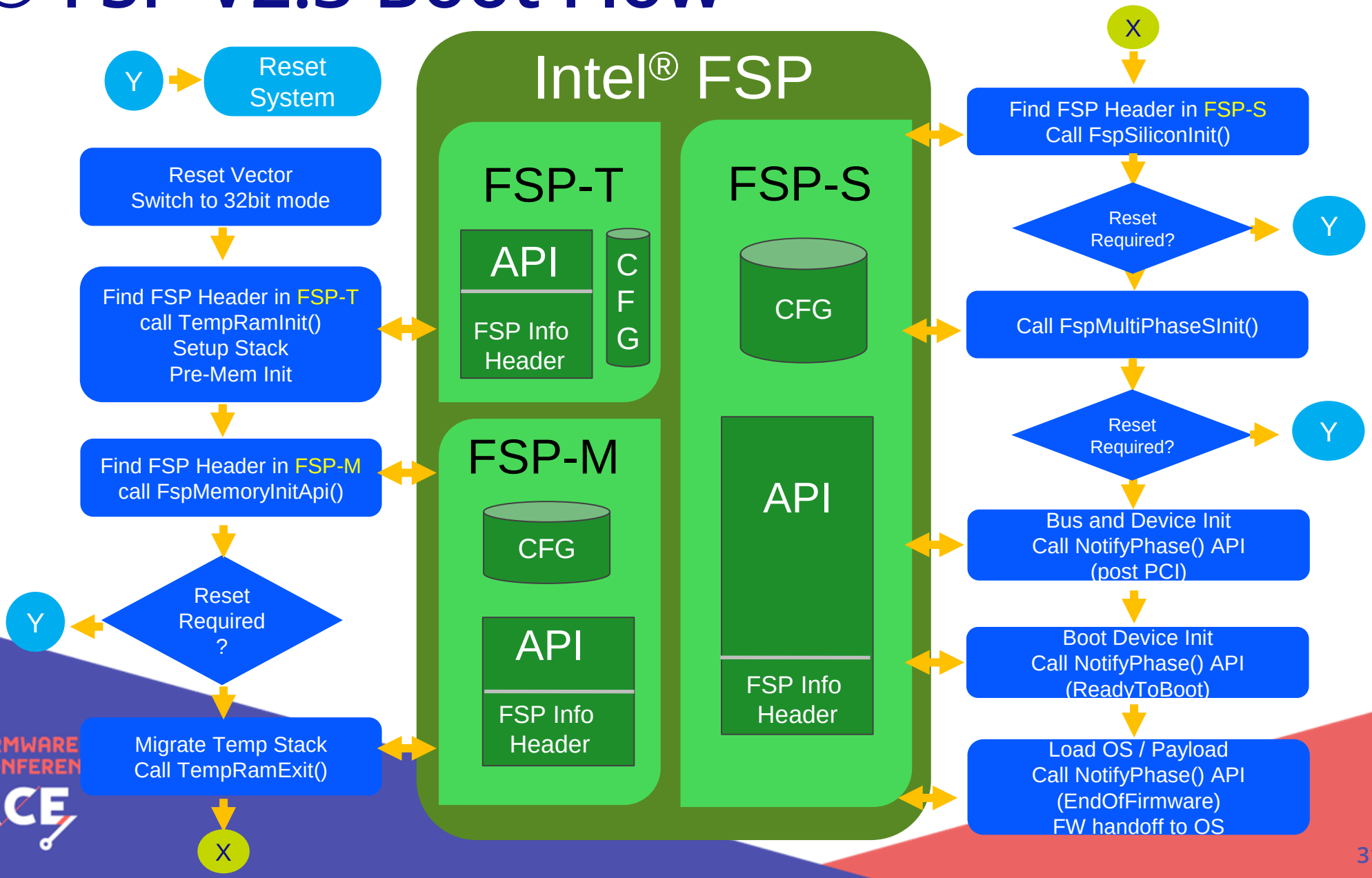
If a pointer in the FSP INFO Header is 0x00000000 then API not available in this component



FSP-T FV only `TempRamInitEntry` is non-zero

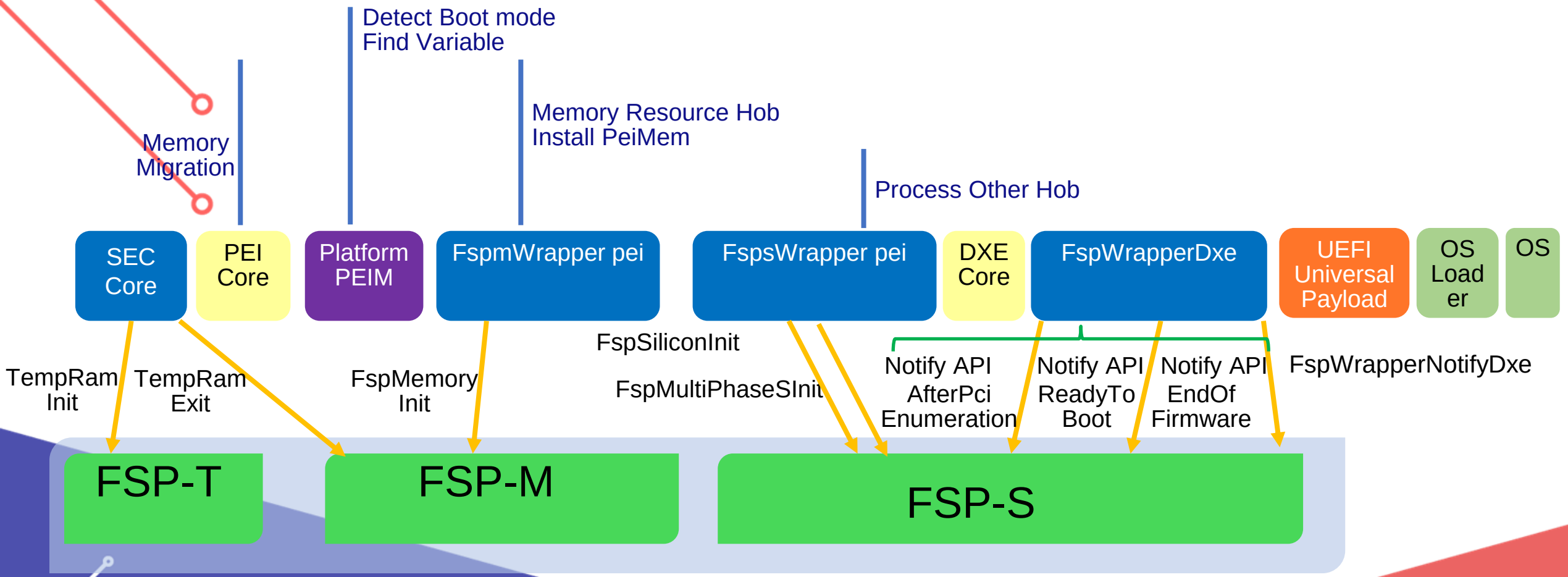
Intel® FSP V2.3 Boot Flow

Using Intel® FSP w/ EDK II: PDF

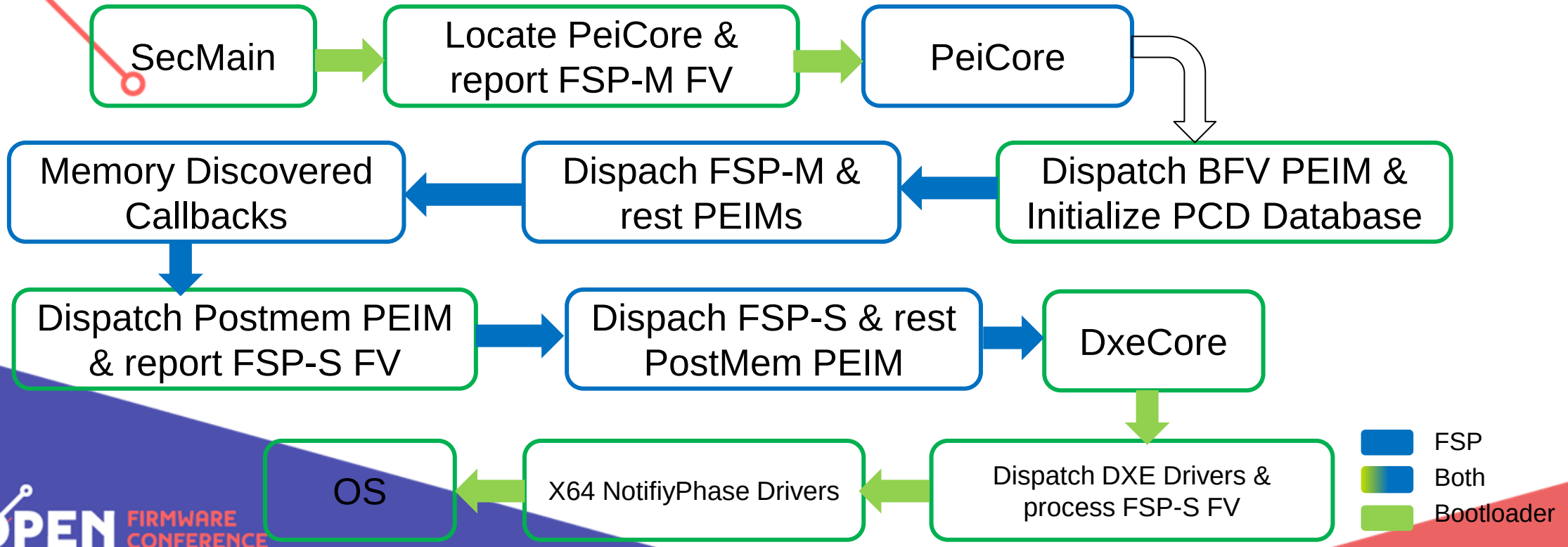


Producer
Consumer

Boot Flow with UEFI & Intel® FSP



Starting with Intel FSP 2.1 Dispatch Mode Boot Flow

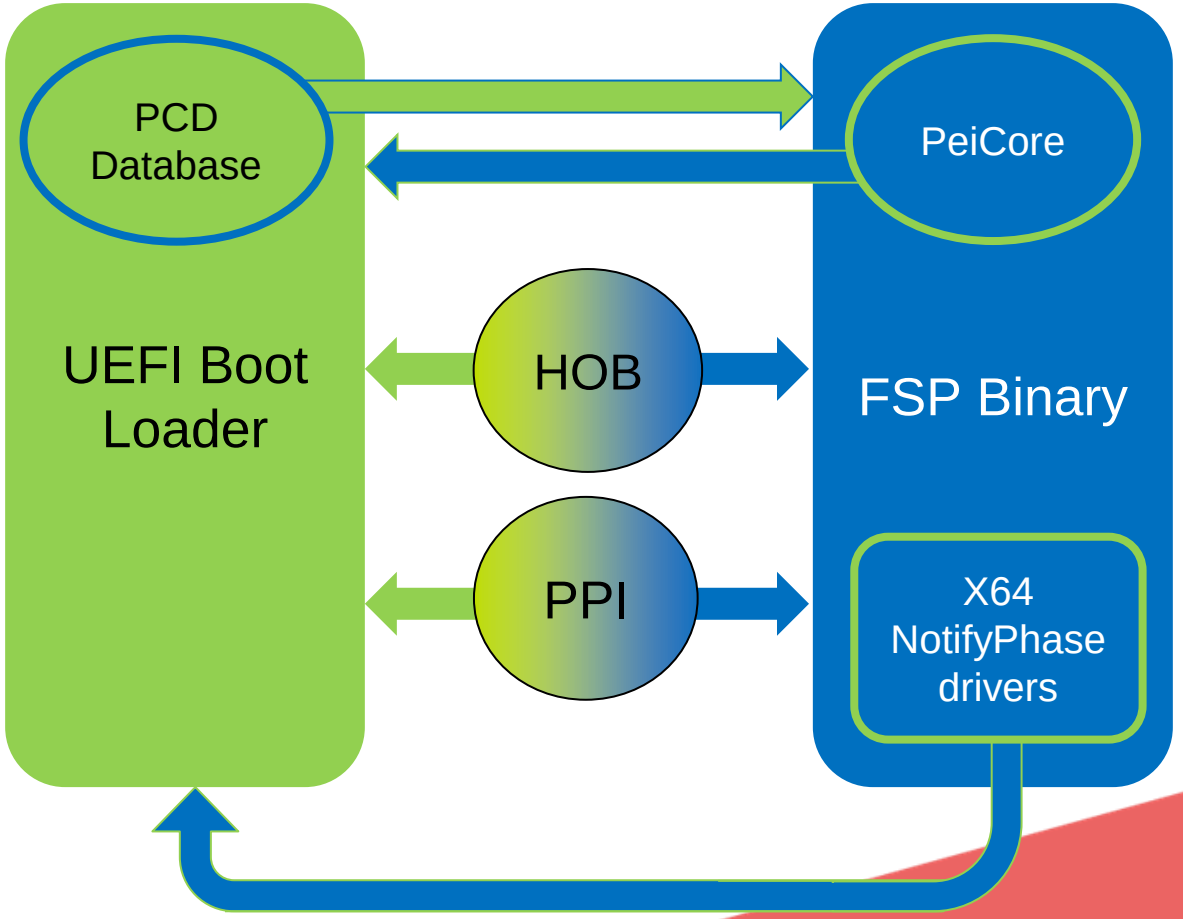


FSP Spec 2.3

Figure 5

Intel FSP Dispatch Mode Interface

- Optional boot flow intended to enable Intel FSP to integrate well in to UEFI bootloader implementations.
- Conforms to UEFI & PI Specifications
- The FSP-T, FSP-M, and FSP-S are containers that expose firmware volumes (FVs) directly to the bootloader.
- UPD Mechanism to pass Config data is not needed
- PCD Database Required



FSP Spec 2.3

Figure 6

Intel® FSP Producer

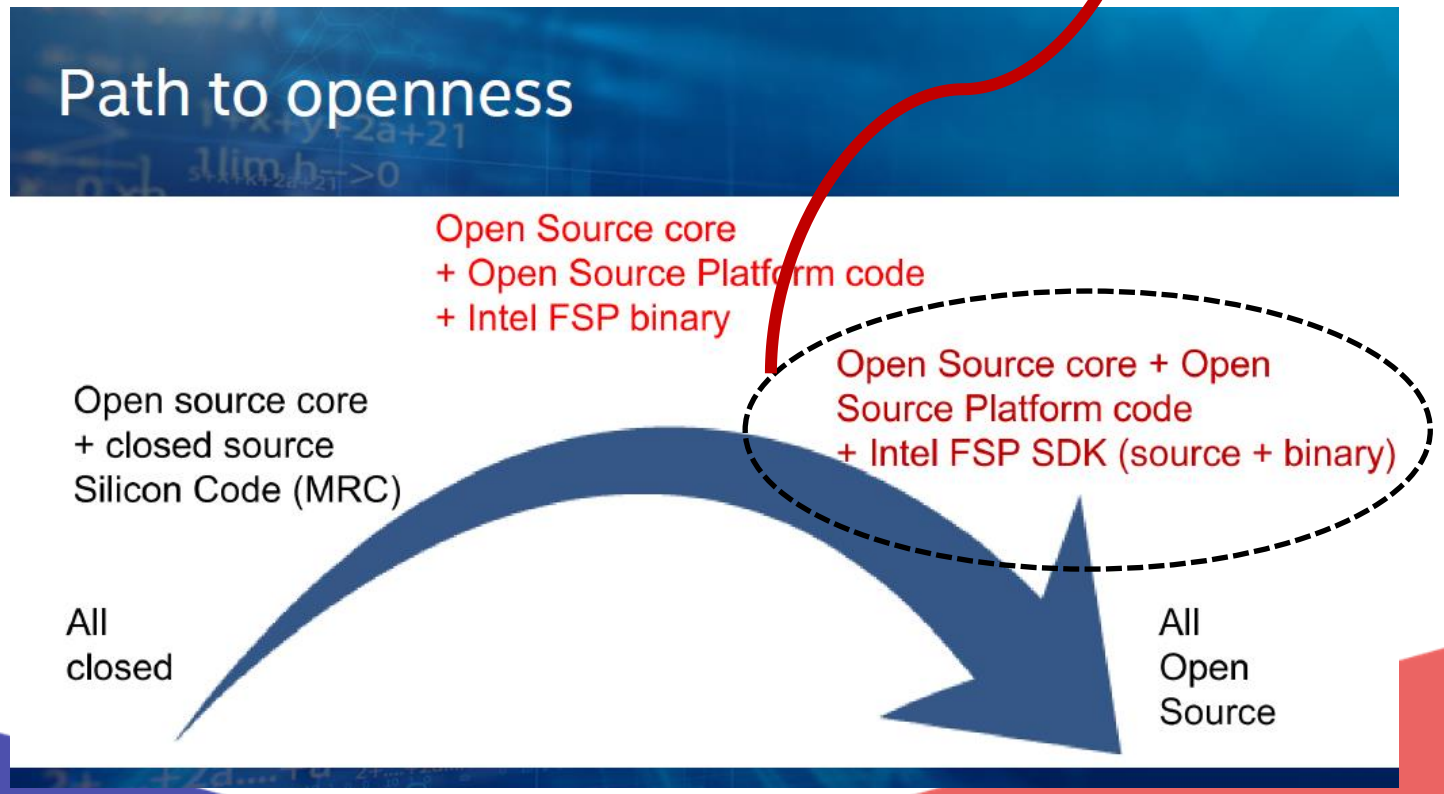
- Examples of binary instances on http://www.intel.com/fsp with integration guides
  - This includes hardware initialization code that is EDK II based PEI Modules (PEIM's)
- Modules are encapsulated as a UEFI PI firmware volume w/ extra header
- Configure w/Vital Product Data (VPD)-style Platform Configuration Data (PCD) externalized from the modules
- Resultant output state reported via UEFI Platform Initialization (PI) Hand Off Block (HOB)

Intel® Firmware Support Package (Intel® FSP) External Architecture Specification (EAS) v2.3 Link v2.0

Resource: https://software.intel.com/content/www/us/en/develop/articles/intel-firmware-support-package.html

https://github.com/UniversalScalableFirmware/fspsdk

- From https://2018.osfc.io/talks/keynote.html



FSP Authentication

- Intel Firmware Support Package (FSP)
  - **A binary** to perform silicon initialization.
  - Released by Intel.
  - Can be integrated into OEM BIOS.

- Question
  - Is the FSP binary in OEM BIOS from Intel?
  - Is it the latest FSP binary with known bug fix?

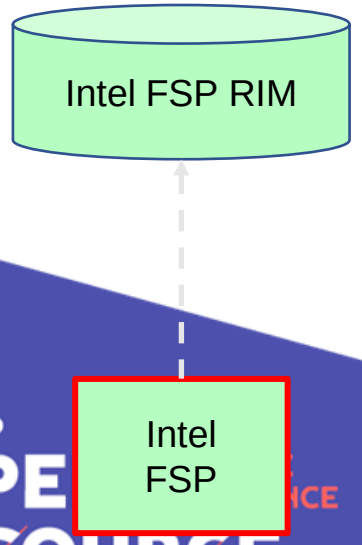
Reference Integrity Manifest

Base RIM (SWID Tag)
==========

Name: AmberLakeFspBin
version: 3.7.6
tagID: CC92BA16-8450-4C4B-8EFA-F34D5299D5E0
Link: href: http://...
Role: softwareCreator tagCreator
BindingSpec: PC Client RIM
BindingSpecVersion: 1.2
SupportRIMFormat: TCG_EventLog_Assertion
SupportRIMURL: http://...
...
DigitalSignature: XXXXXXXX

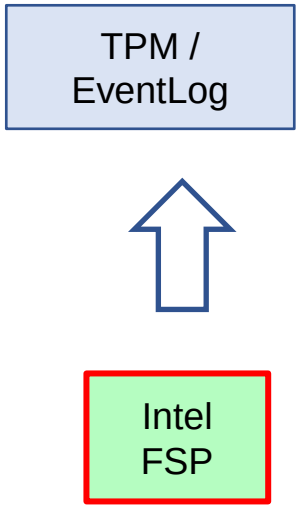
Support RIM
==========

EV_EFI_PLATFORM_FIRMWARE_BLOB2:
Description: FSPT
Base: 0x00000000FFED0000
Length: 0x0000000000130000
Digest: AAAAAAAA
EV_EFI_PLATFORM_FIRMWARE_BLOB2:
Description: FSPM
Base: 0x00000000FFDF0000
Length: 0x0000000000066000
Digest: BBBBBBBB
EV_EFI_PLATFORM_FIRMWARE_BLOB2:
Description: FSPS
Base: 0x00000000FFD90000
Length: 0x000000000002E000
Digest: CCCCCCCC



** Source: Intel FSP 2.x Measurement and Attestation,* https://cdrdv2.intel.com/v1/dl/getContent/644001

Boot time measurement



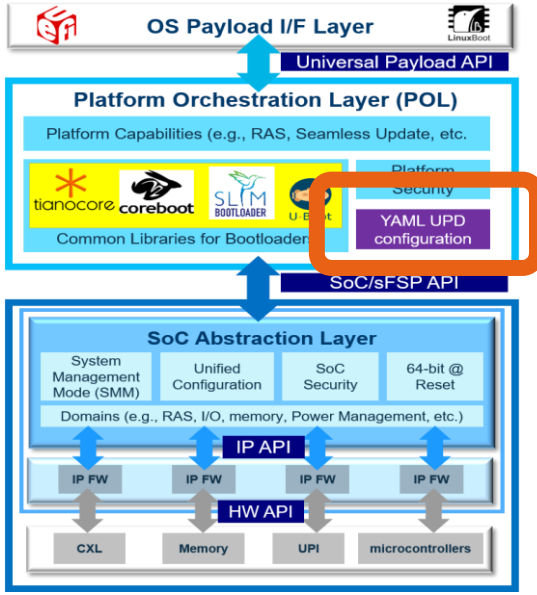
TCG Event Log
==========

```
TCG_Sp800_155_PlatformId_Event2:
        ManufactureId: <OEM_ID>
        ManufactureStr: <OEM>
        RimGuid: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
TCG_Sp800_155_PlatformId_Event2:
        ManufactureId: 343
        ManufactureStr: Intel
        RimGuid: CC92BA16-8450-4C4B-8EFA-F34D5299D5E0
EV_EFI_PLATFORM_FIRMWARE_BLOB2:
        Description: FSPT
        Base: 0x00000000FFED0000
        Length: 0x0000000000130000
        Digest: AAAAAAAA
EV_EFI_PLATFORM_FIRMWARE_BLOB2:
        Description: FSPM
        Base: 0x00000000FFDF0000
        Length: 0x0000000000066000
        Digest: BBBBBBBB
EV_EFI_PLATFORM_FIRMWARE_BLOB2:
        Description: FSPS
        Base: 0x00000000FFD90000
        Length: 0x000000000002E000
        Digest: CCCCCCCC
```

** Reference:*
https://github.com/tianocore/edk2/blob/master/IntelFsp2WrapperPkg/Include/Library/FspMeasurementLib.h

Configuration

YAML Format Boot Configuration



Configuration flow today

- For FSP configuration
- BIOS build process generates FSP collaterals, such as FD and header files.
- Bootloader engineers consume these collaterals.

- For BIOS setup menu configuration
- BIOS build process generates HII related files (UNI/HFR/VFR/HPK/I)
- No UI to render BIOS configurations without booting platform.

Opportunity

- DSC, DEC, VFR, UNI, HFR, BSF, PCD -> YAML to enable single data source, compared to many places to change for configuration
- Streamline configuration process across UEFI and bootloaders.
- Open-source Config Editor tool support, compared to closed source tool such as BCT.

What's YAML

- Human-readable data-serialization language
- List of key/value pairs. Superset of JSON.
- 19 years of history, widely adopted. Many tools/libraries available.
- Slim bootloader currently using YAML as single configuration source

Comparison between BSF and YAML

Server.bsf

```
611 Page "SoC"
612     Combo $gEagleStreamFspPkgTokenSpaceGuid_BifurcationPcie0, "PCIe
        Controller 0 Bifurcation",
        &gEagleStreamFspPkgTokenSpaceGuid_BifurcationPcie0,
613         Help "Configure PCI Express controller 0 bifurcation."
614     Combo $gEagleStreamFspPkgTokenSpaceGuid_BifurcationPcie1, "PCIe
        Controller 1 Bifurcation",
        &gEagleStreamFspPkgTokenSpaceGuid_BifurcationPcie1,
615         Help "Configure PCI Express controller 1 bifurcation."
616     Combo $gEagleStreamFspPkgTokenSpaceGuid_ActiveCoreCount, "Active
        Core Count", &gEagleStreamFspPkgTokenSpaceGuid_ActiveCoreCount,
617         Help "Select # of Active Cores (Default: 0, 0:ALL, 1..15 =
            1..15 Cores)"
618     Combo $gEagleStreamFspPkgTokenSpaceGuid_EnablePcie0, "PCIe
        Controller 0", &EN_DIS,
```

Server.yaml

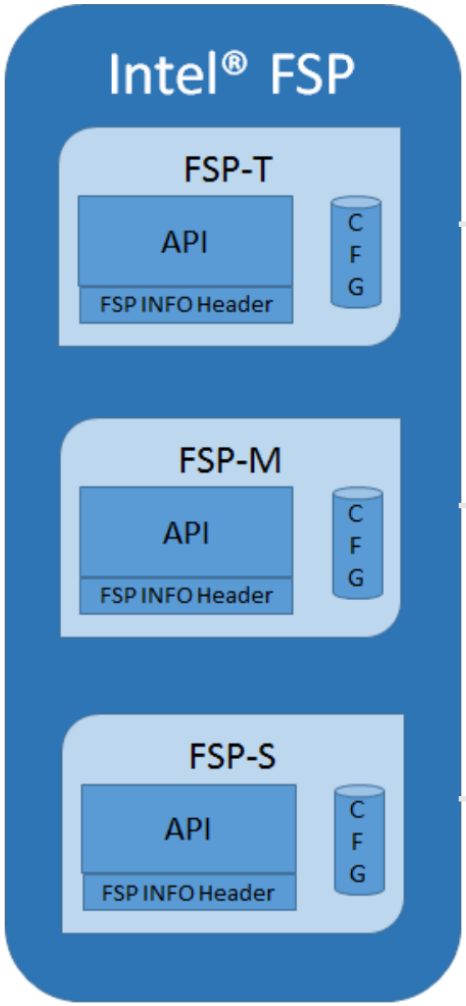
```
968             value         : 3
969             help          : >
970                             Configure PCI Express controller 1 bifurcation.
971             length        : 0x01
972             option        : 0:X2X2X2X2, 1:X2X2X4, 2:X4X2X2, 3:X4X4, 4:X8
973         - ActiveCoreCount :
974             type          : Combo
975             name          : Active Core Count
976             value         : 0
977             help          : >
978                             Select # of Active Cores (Default- 0, 0:ALL,
                                1..15 = 1..15 Cores)
979             length        : 0x01
980             option        : 0:ALL, 1:1, 2:2, 3:3, 4:4, 5:5, 6:6, 7:7, 8:8,
                9:9, 10:10, 11:11, 12:12, 13:13, 14:14, 15:15
981         - CpuMicrocodePatchBase :
```

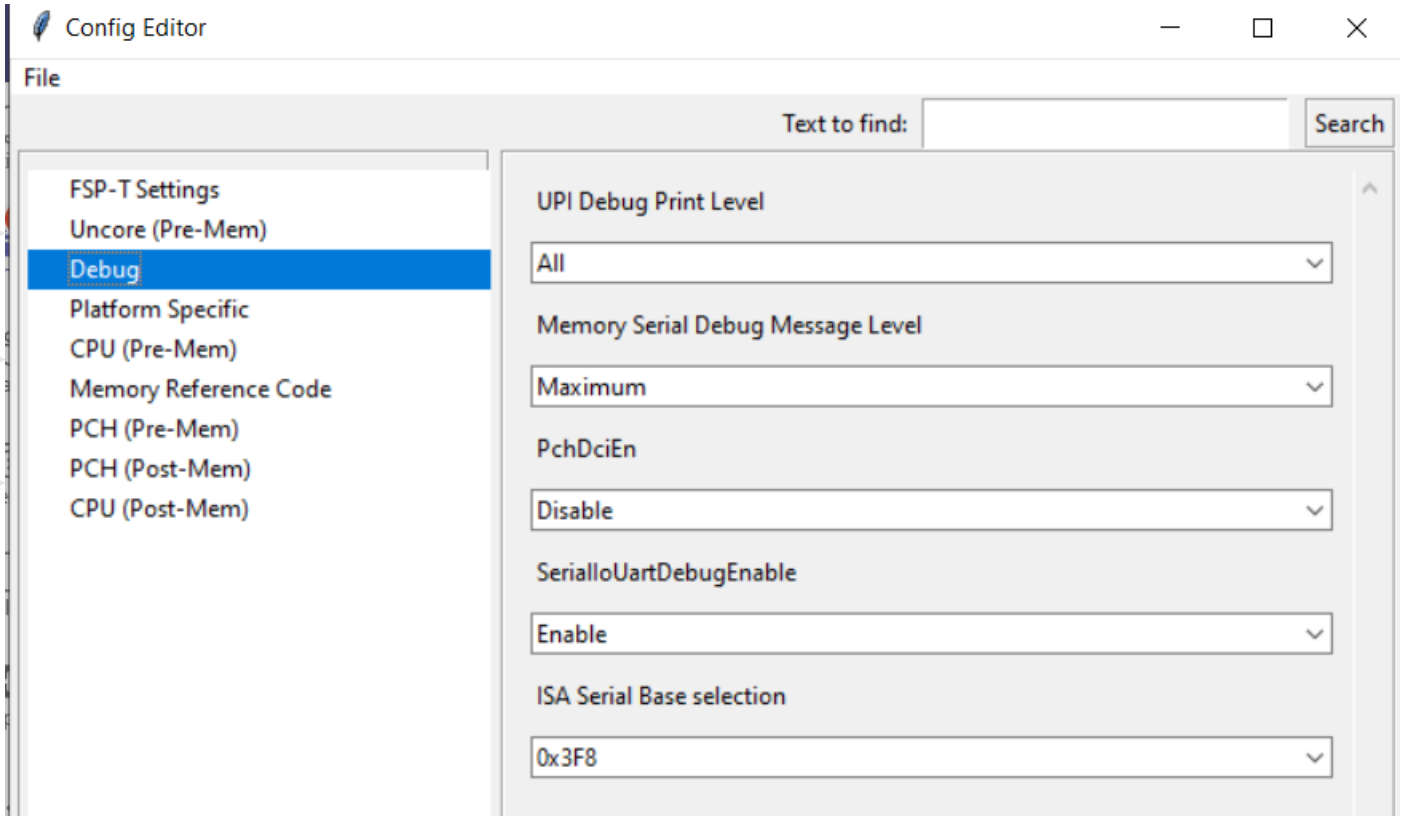
Config Editor

- Features support:
  - Read FSP binary information
  - Allow patching any BIOS/IFWI image containing FSP UPDs
  - Read YAML config format while BSF backward compatible
  - Bit format FSP support instead of bytes
  - Modifying BSF parameters and export loadable delta files
  - FSP 1.x and 2.x format backward compatible
  - Search function

AmberLakeFspBinP...	Update Readme.md for Coffee Lake Refresh	5 months ago
ApolloLakeFspBinP...	Apollo Lake MR9 FSP.	6 months ago
BraswellFspBinPkg	Convert Braswell BSF file to CR/LF format	7 months ago
CoffeeLakeFspBinP...	Update Readme.md for Coffee Lake Refresh	5 months ago
CometLakeFspBinP...	Comet Lake FSP 9.3.7B.20	2 months ago
DenvertonNSFspBi...	Update DenvertonNSFsp.bsf	2 years ago
ElkhartLakeFspBinP...	Elkhart Lake PR1 FSP	2 months ago
IceLakeFspBinPkg	Ice Lake FSP 8.0.52.40	2 years ago
KabylakeFspBinPkg	Kaby Lake FSP 3.7.6	2 years ago
SkylakeFspBinPkg	Reorganize the FSP repo to have all FSPs i...	3 years ago
TigerLakeFspBinPkg	Tiger Lake FSP A.0.51.31	2 months ago
Tools	Update SplitFspBin.py to latest from edk2	5 months ago
FSP_License.pdf	Add files via upload	3 years ago
README.md	Update README.md	12 days ago

Firmware

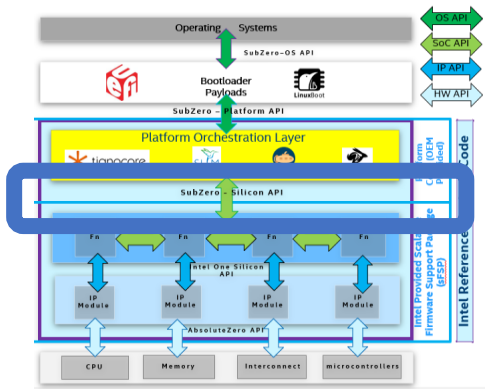

Intel® FSP
FSP-T
API
FSP INFO Header
CFG
FSP-M
API
FSP INFO Header
CFG
FSP-S
API
FSP INFO Header
CFG


Config Editor
File
Text to find:
Search
FSP-T Settings
Uncore (Pre-Mem)
Debug
Platform Specific
CPU (Pre-Mem)
Memory Reference Code
PCH (Pre-Mem)
PCH (Post-Mem)
CPU (Post-Mem)
UPI Debug Print Level
All
Memory Serial Debug Message Level
Maximum
PchDciEn
Disable
SerialIoUartDebugEnable
Enable
ISA Serial Base selection
0x3F8

Steps to run config editor

- Clone Intel FSP at https://github.com/intel/FSP
- Clone edk2 code at https://github.com/tianocore/edk2
- ConfigEditor is located at IntelFsp2Pkg/Tools/ConfigEditor
- Run “python ConfigEditor.py”

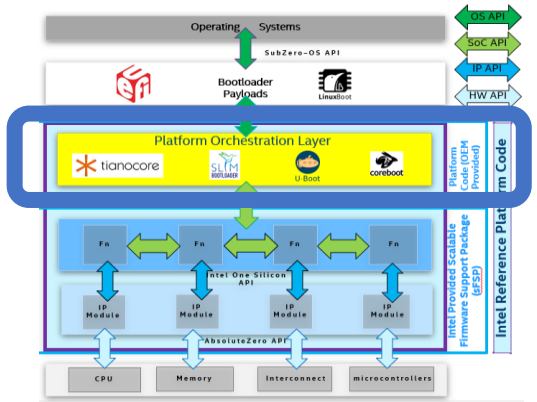
Industry USF spec key capabilities



SOC I/F – HAL

64-bit HAL
SMM and ACPI
Authentication
Unified configuration
HAL @ reset vector
HAL for PreSi Val

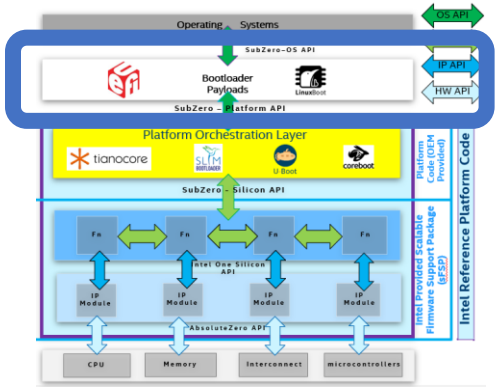
https://github.com/universalscalablefirmware/fspsdk



Platform I/F - POL

- Simplified ACPI
- Common Libs for bootloaders including Rust safe language
- Standard binary configuration mechanism (YAML)
- Attestation/ authentication/ update/ measurement

https://github.com/tianocore/edk2/tree/master/IntelFsp2Pkg/Tools/ConfigEditor



OS I/F - Payload

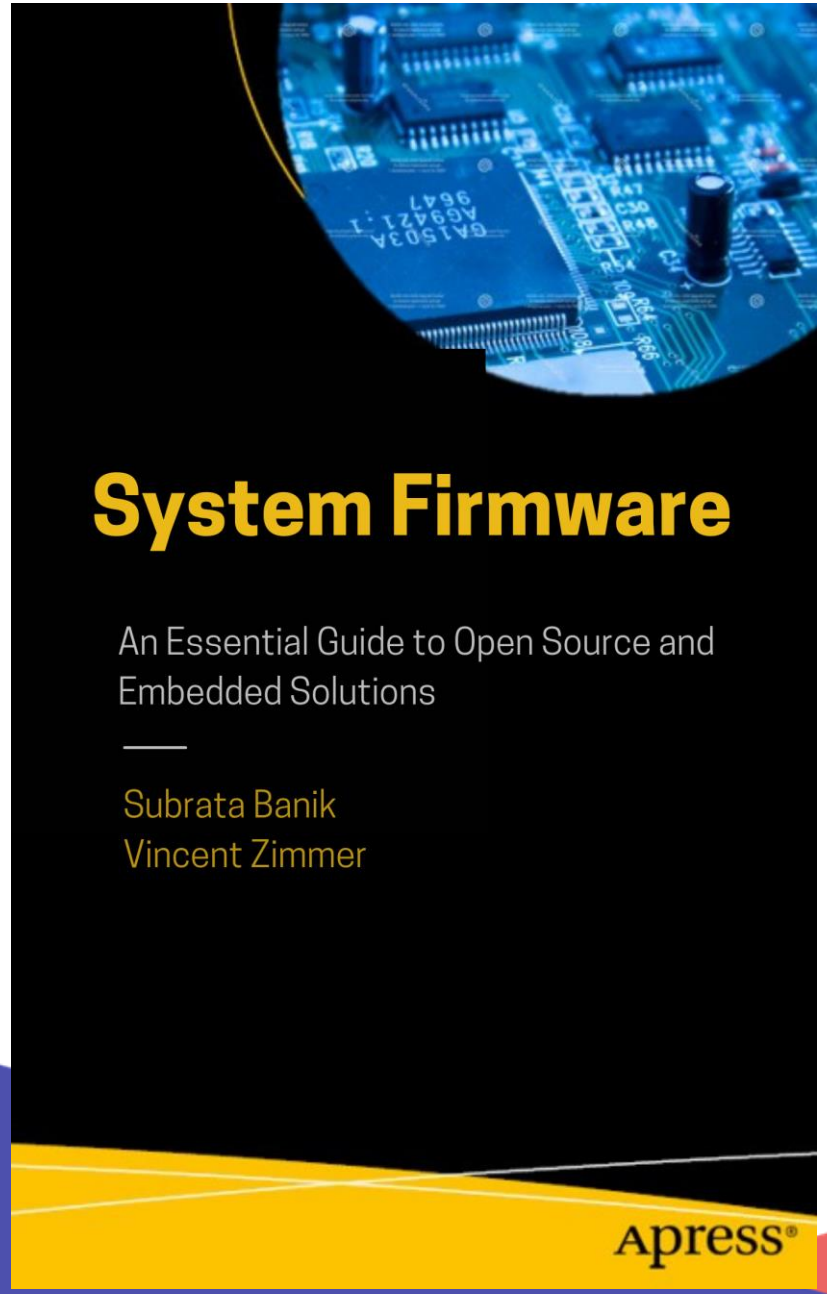
- Universal API for different payloads (i.e. UEFI, LinuxBoot)
- Support various bootloaders (tianocore, coreboot, slim boot)
- Embedded hypervisor (e.g. ACRN)

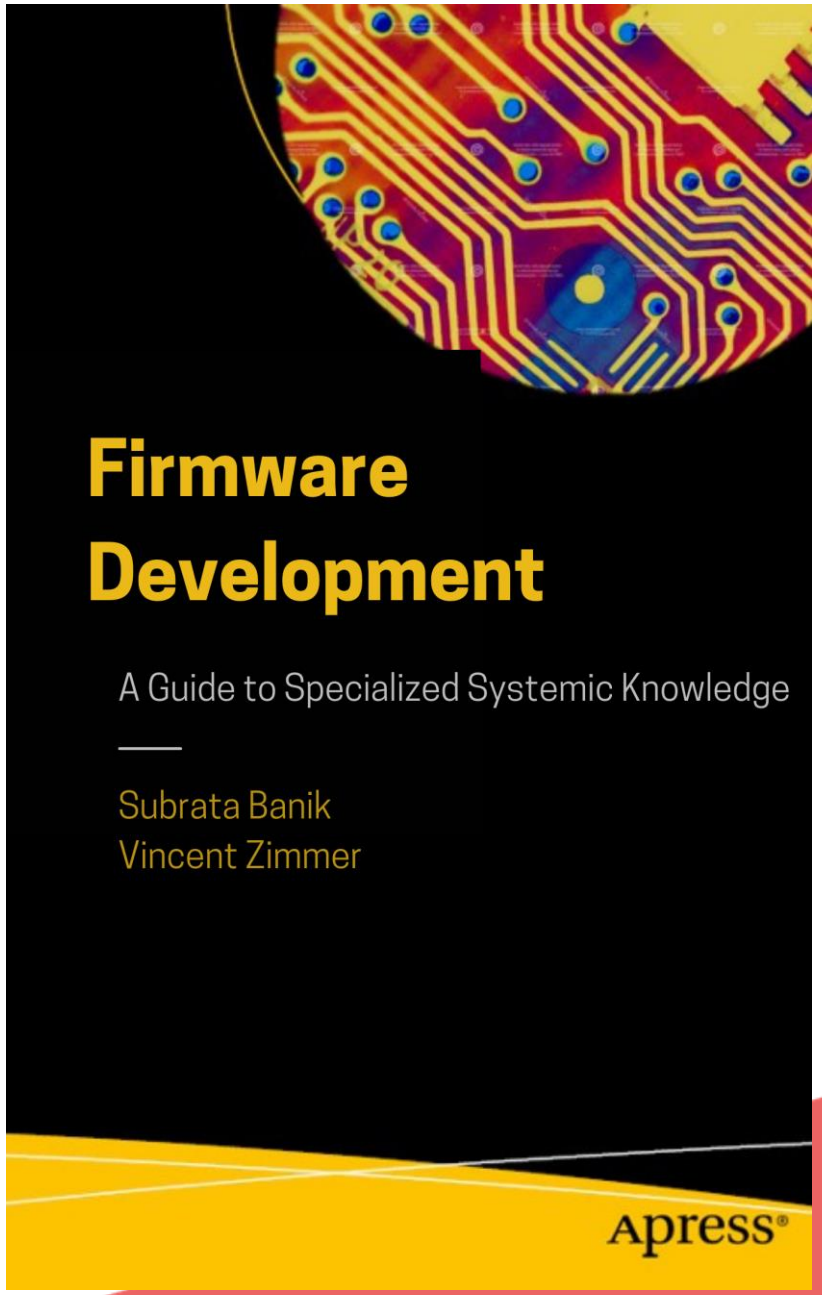
https://github.com/universalscalablefirmware

More information

- USF site https://github.com/universalscalablefirmware
- USF Specification https://universalscalablefirmware.github.io/documentation/
- FSP https://www.intel.com/fsp

Learn more about system firmware development*





*Soon to be available at https://apress.com and other leading e-commerce websites for purchase

OPEN SOURCE FIRMWARE CONFERENCE

OPEN SOURCE FIRMWARE CONFERENCE