Enabling Runtime RAS for Xeon Platforms using Open System Firmware

Daocheng “Amos” Bu, Firmware Engineer (Intel)

Anjaneya “Reddy” Chagam, Sr. Principal Engineer (Intel)

Intel Confidential

intel

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Your costs and results may vary.

No product or component can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks .

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/benchmarks .

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

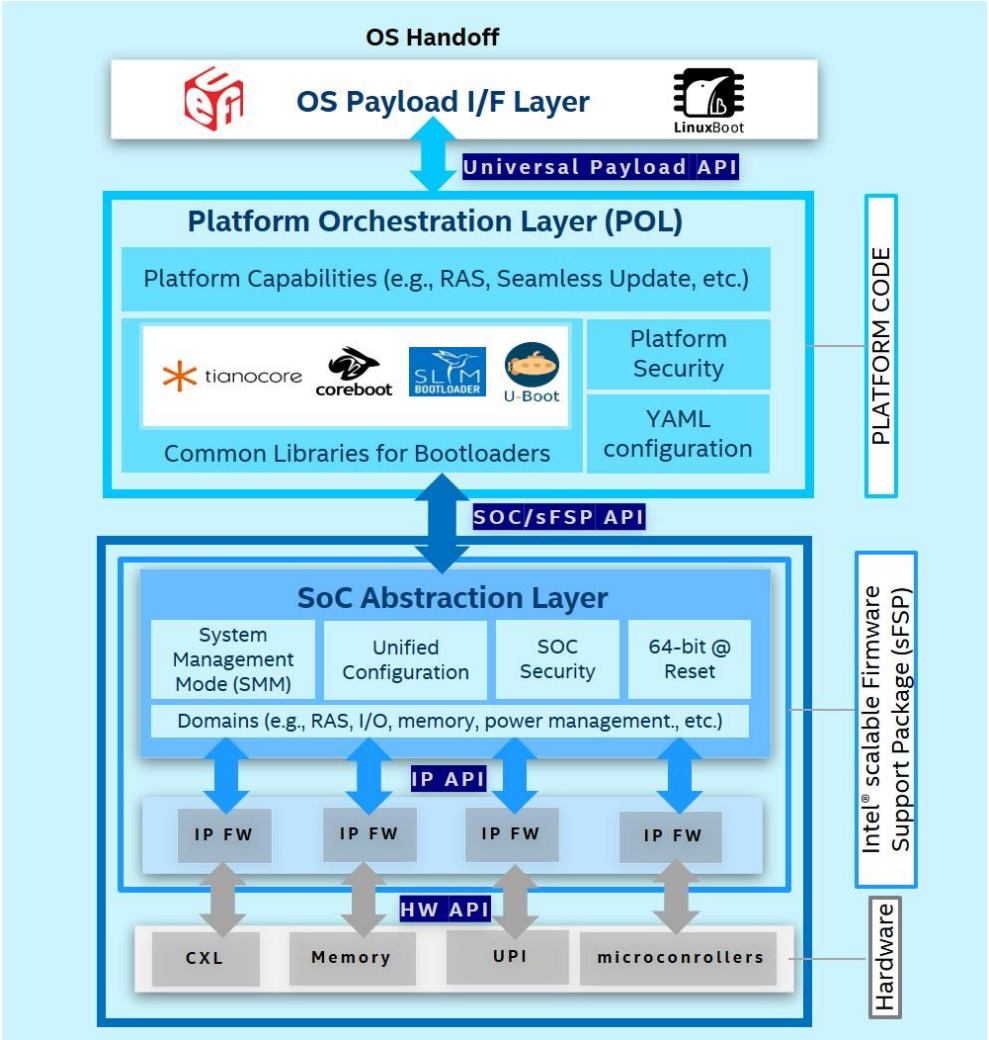
© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Agenda

- Background
- FSP-SMM API
- FSP-SMM binary layout
- SMM traditional Mode VS. Standalone Mode
- FSP-SMM flow
- OSF(coreboot)→FSP-SMM
- Traditional SMM transition to MM
- Converged SMM Solution (Future)

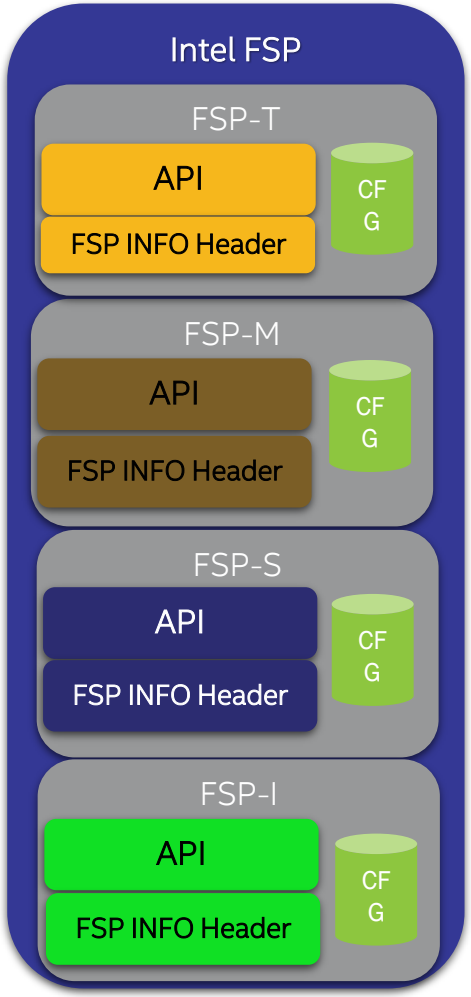
Background

- Current FSP2.3 does not include SMM related function, bootloaders own SMM foundation and feature itself.
- However, for some SMM functions like server RAS runtime features, it is not easy for bootloader (like coreboot/SlimBoot) to support it.
- It will be a solution for non-EDKII bootloader with FSP-SMM to support SMM foundation and features.

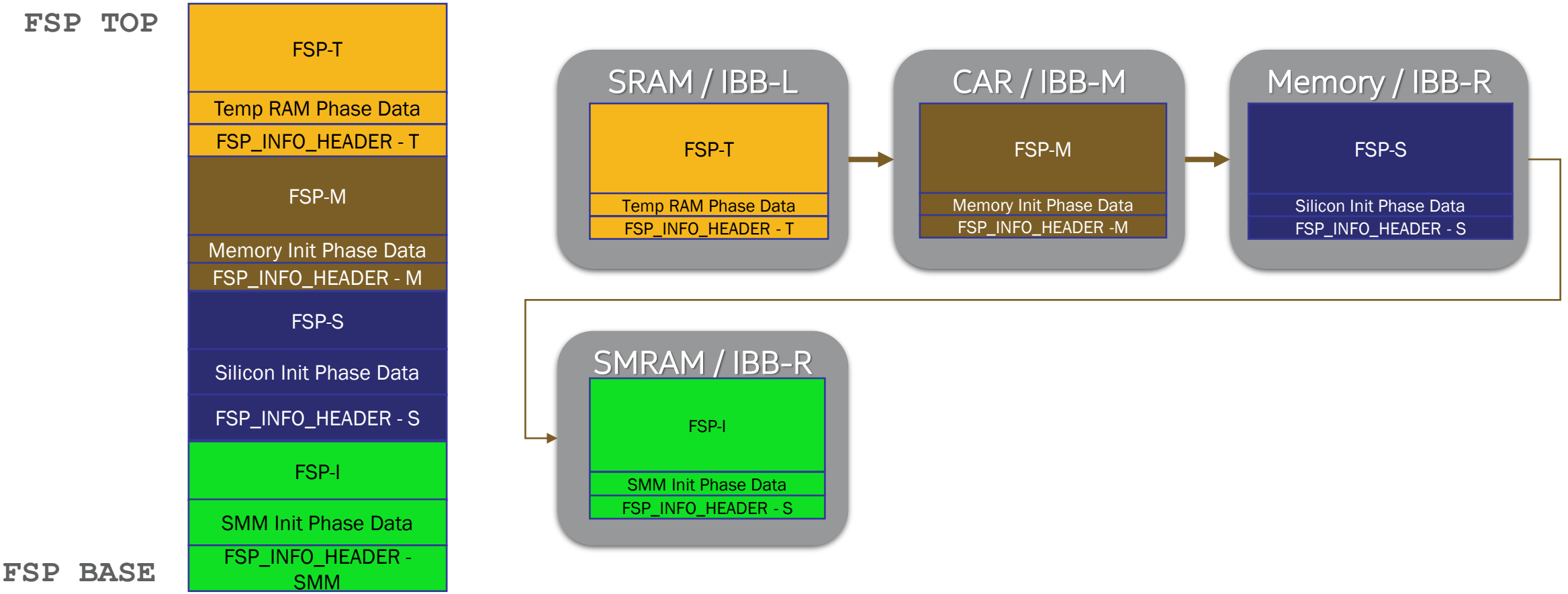


FSP-SMM API

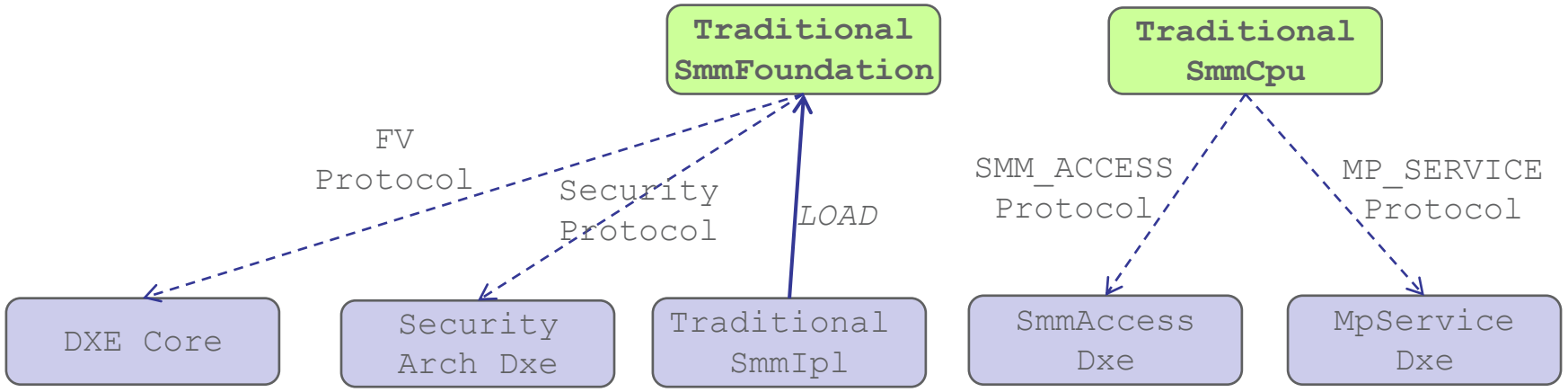
- The Intel® FSP binary follows the UEFI Platform Initialization Firmware Volume Specification format.
  - Split into 4 separate FVs
- FSP-SMM: SMM initialization phase (FSP owns SMRAM)
  - Primary purpose of this phase is to provide a collection of **silicon** SMI handlers that provide value-add services that a bootloader can use.
    - **FspSmmInit()** : Initialize SMM Foundation, dispatch all Standalone MM drivers, build SMM environment, install all SMI handlers.



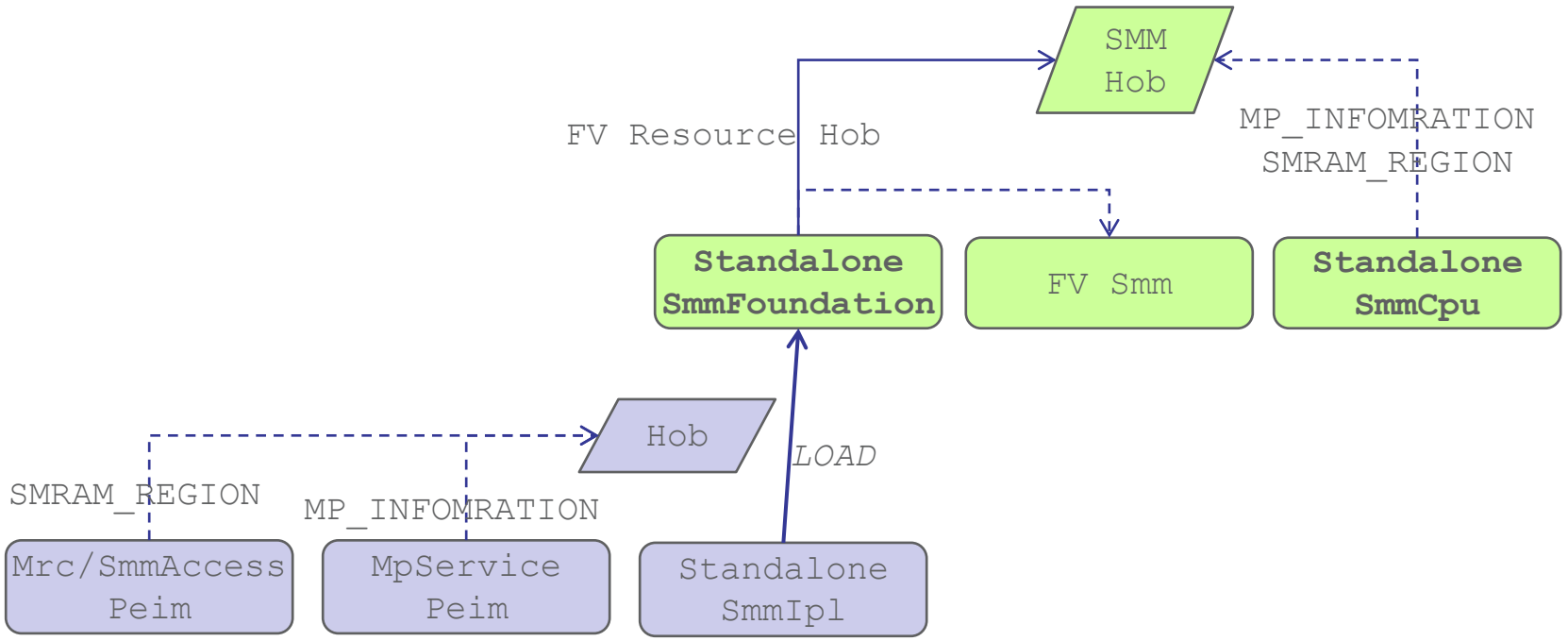
FSP-SMM binary layout



SMM Traditional Mode

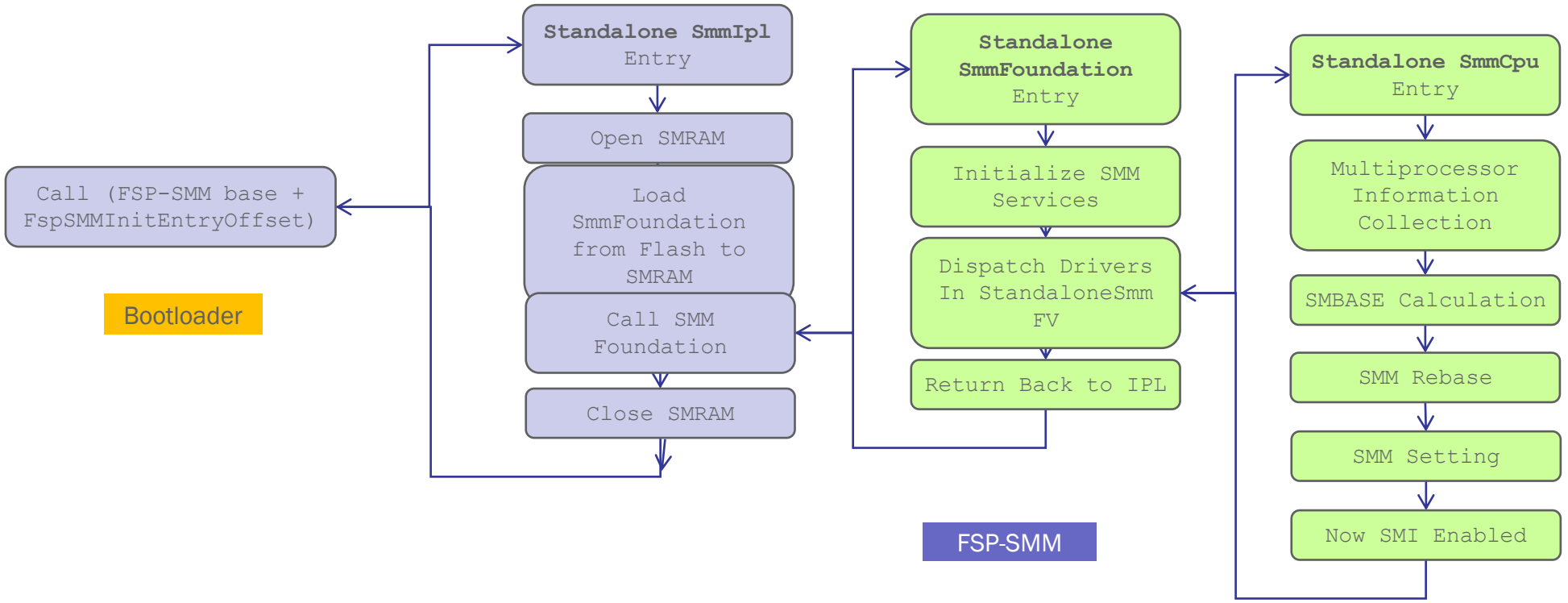


Standalone Mode



Standalone: SMM will not run DXE code.

FSP-SMM Boot Flow

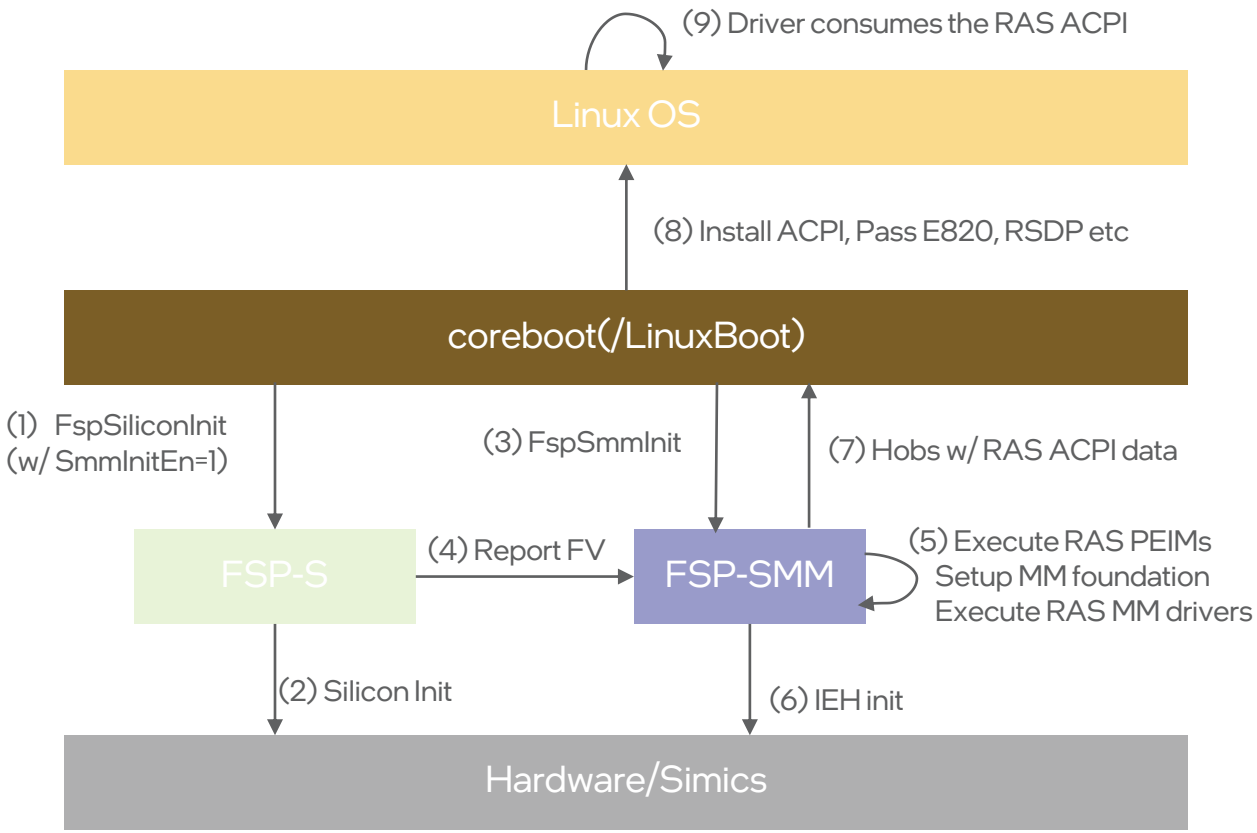


FSP-SMM Modules changes

FV	Status	Build target	Driver name	Description
FSP-S FV	Existed	IA32	SmmAccessPei.inf	Provide access SMM Ram ppi
	Existed	IA32	CpuMpPei.inf	Provide MP service ppi
FSP-SMM FV	New	IA32	SmmControlPei.inf	Provide smi trigger/clear ppi
	New	IA32	CpuMpInfoPei.inf	Provide MP info hob
	New	IA32	StandaloneMmIplPei.inf	Load MMCore to SMM Ram
	New	X64	StandaloneMmCore.inf	Dispatch all MM driver and register all handlers
	New	X64	PiSmmCpuStandaloneSmm.inf	Build CPU SMM enviroment
	New	X64	CpuIo2StandaloneMm.inf	Provide CPU io/memory access protocol
	New	X64	PchSmiDispatcherServer.inf	Install sw smi handler for verify

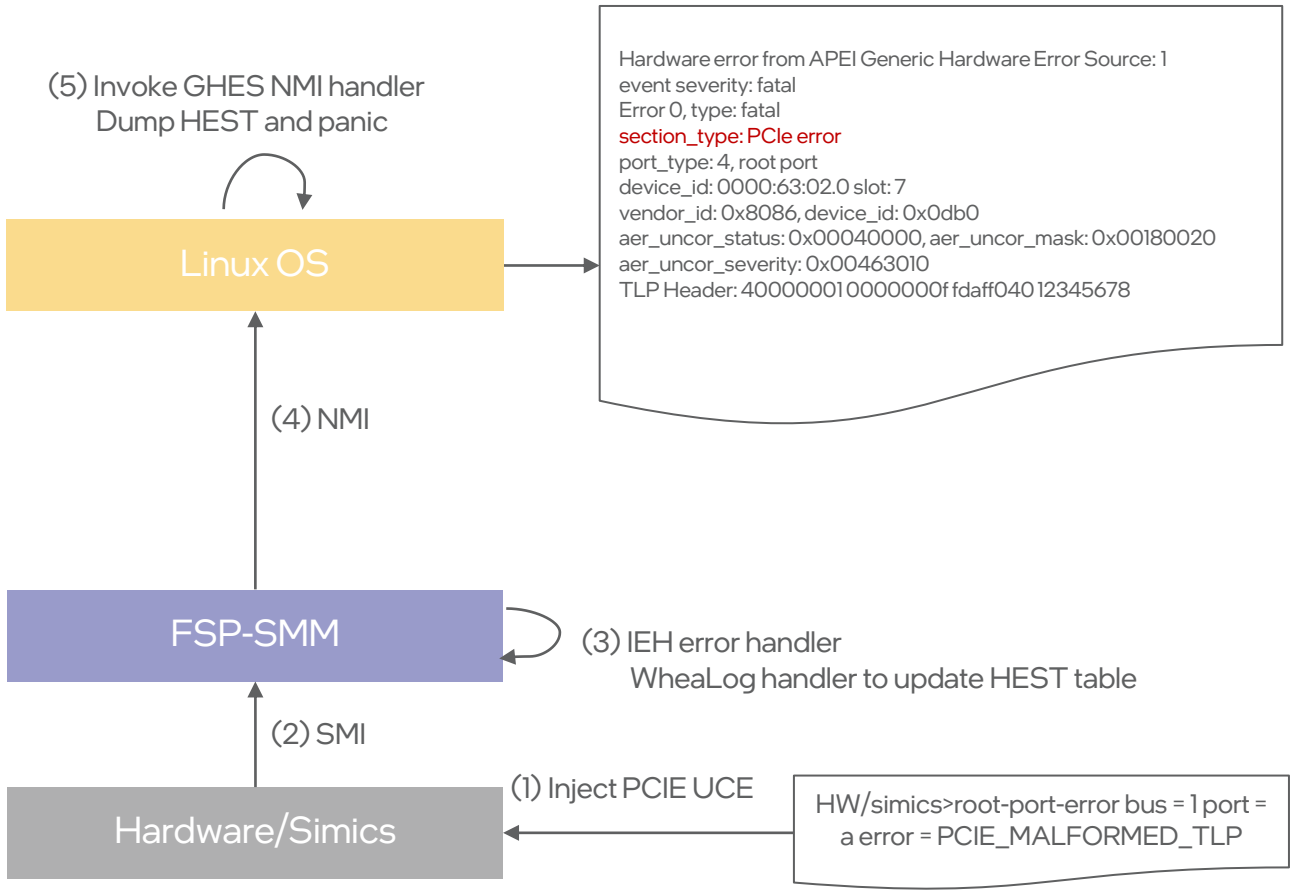
coreboot FSP-SMM (boot) integration

Boot Time

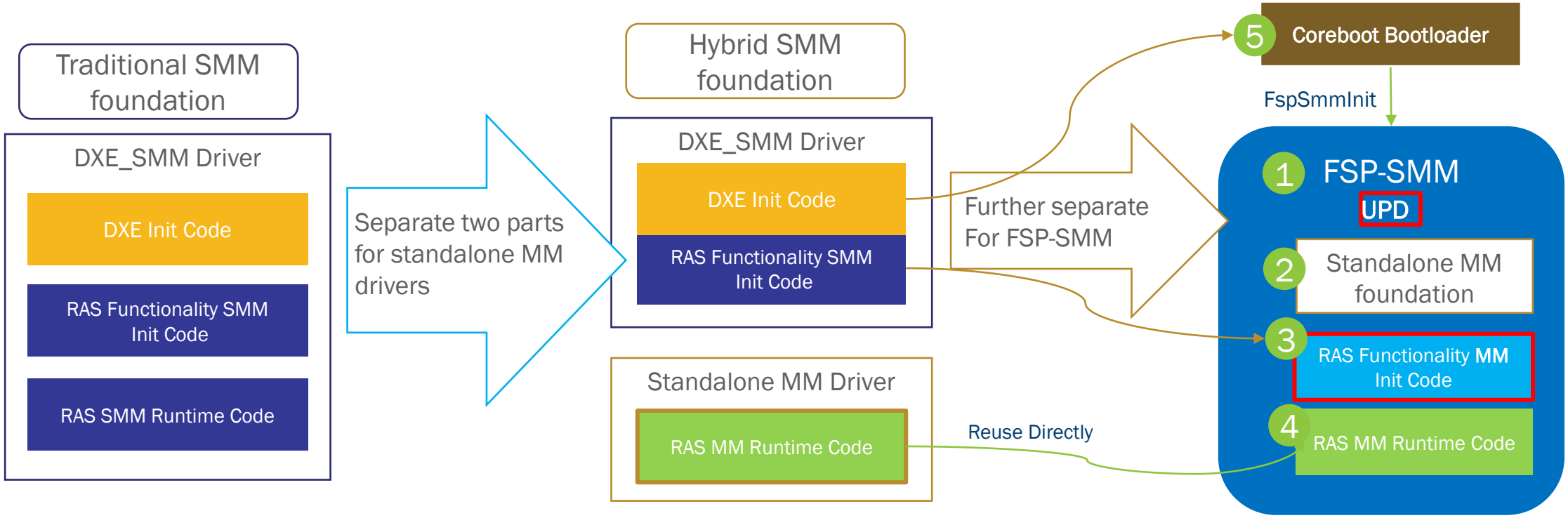


FSP-SMM (runtime) Execution

Runtime



Traditional SMM transition to Stanalone MM



Thanks & QA

- Xeon Server coreboot credits:
  - Meta, Byte Dance, Google, Amazon, Intel
  - 9Elements, SysPro Consulting
  - Wiwynn, Quanta, Inspur
- Reference link:
  - https://github.com/universalscalablefirmware
  - 644852_2.3_Firmware-Support-Package-External-Architecture-Specification
    - FSP 2.4 EAS is coming..