OPEN.

FOR BUSINESS.

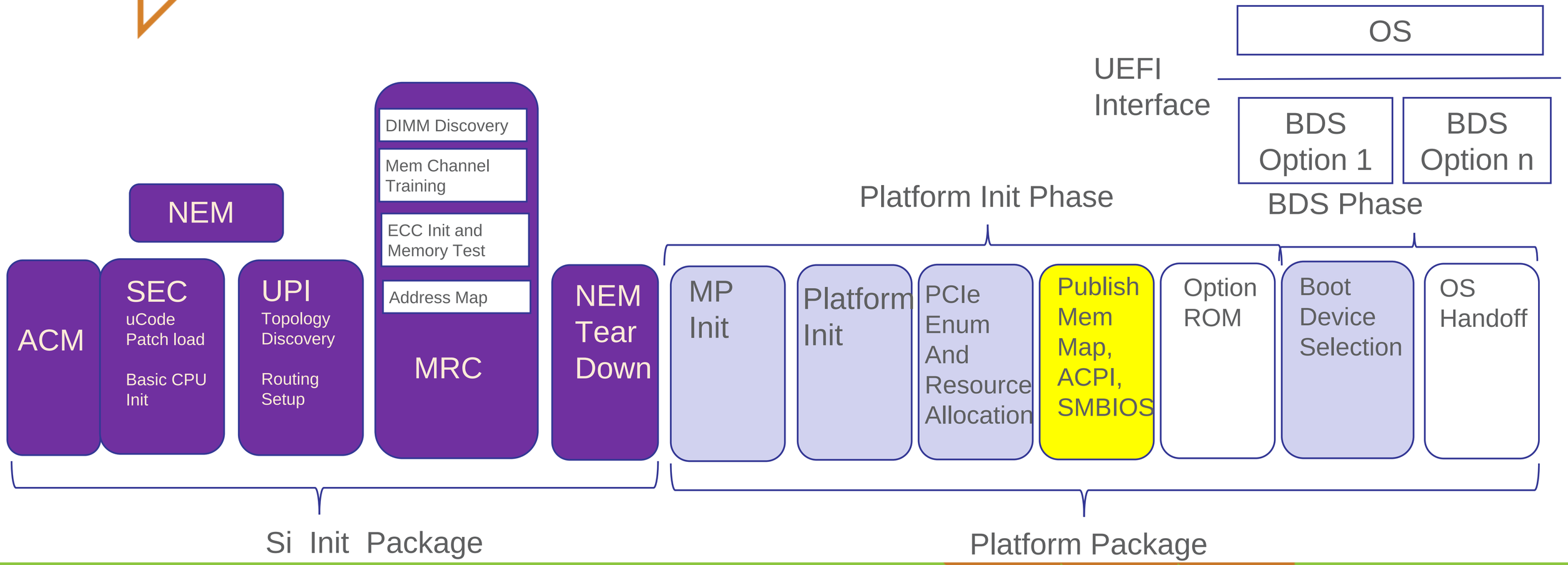
OCP
SUMMIT

Optimized UEFI Implementation on Intel® Xeon Based OCP Platform

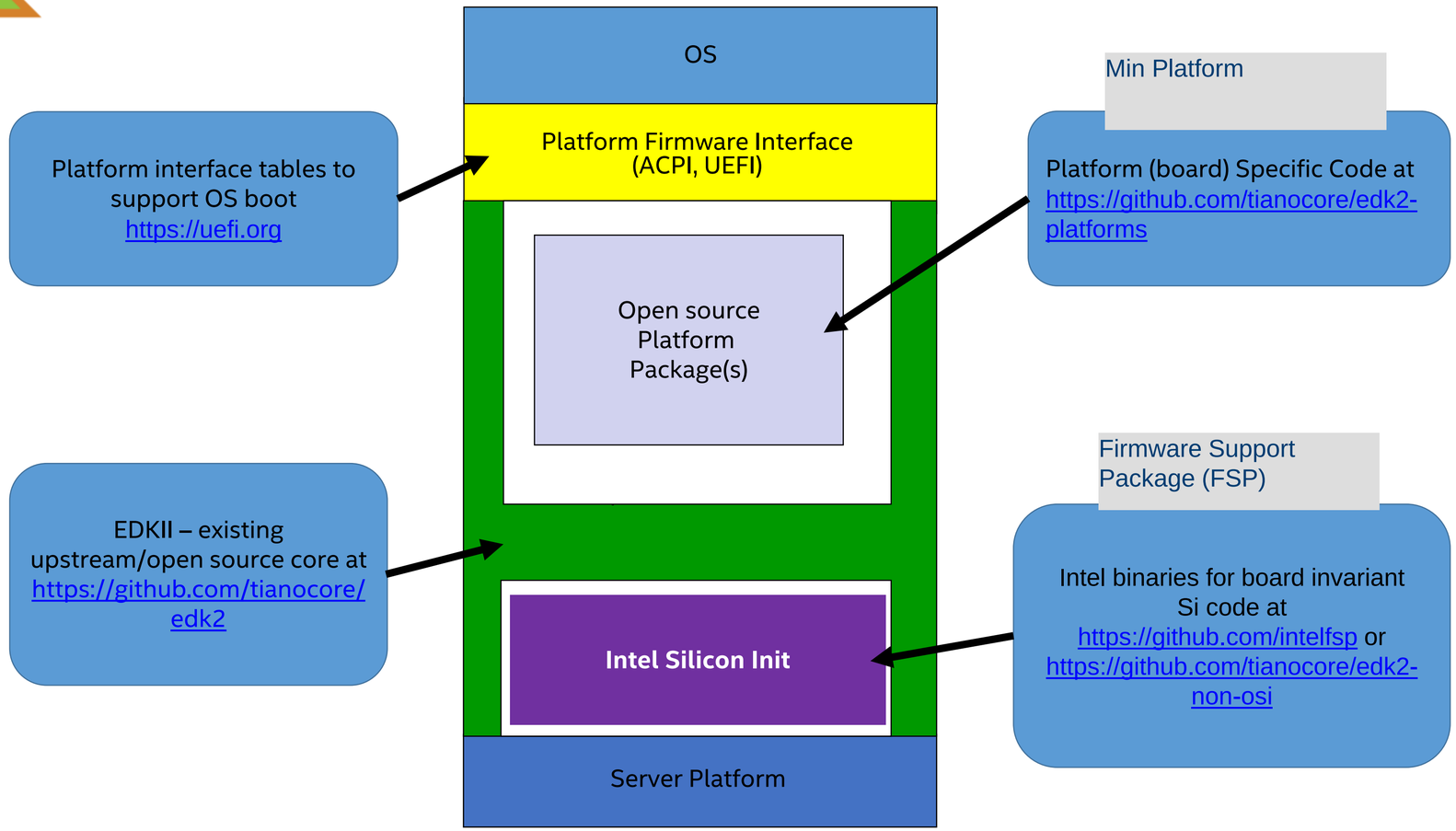
Sarathy Jayakumar, Principal Engineer, Intel Corp

Mohan J. Kumar, Fellow, Intel Corp

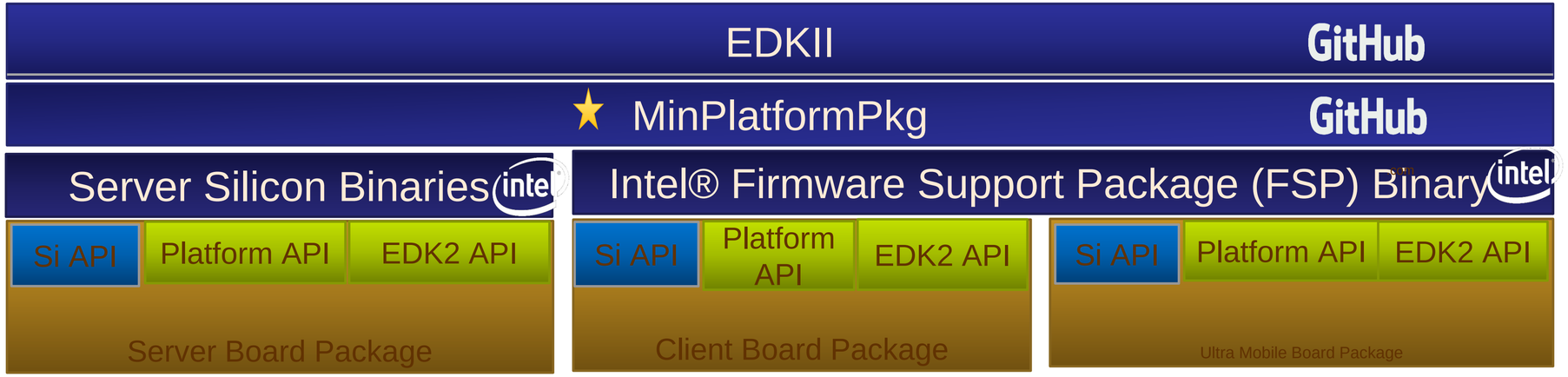
Baseline Boot Flow



UEFI-based Open Firmware
(for Intel® Xeon® Processor-based Platforms)



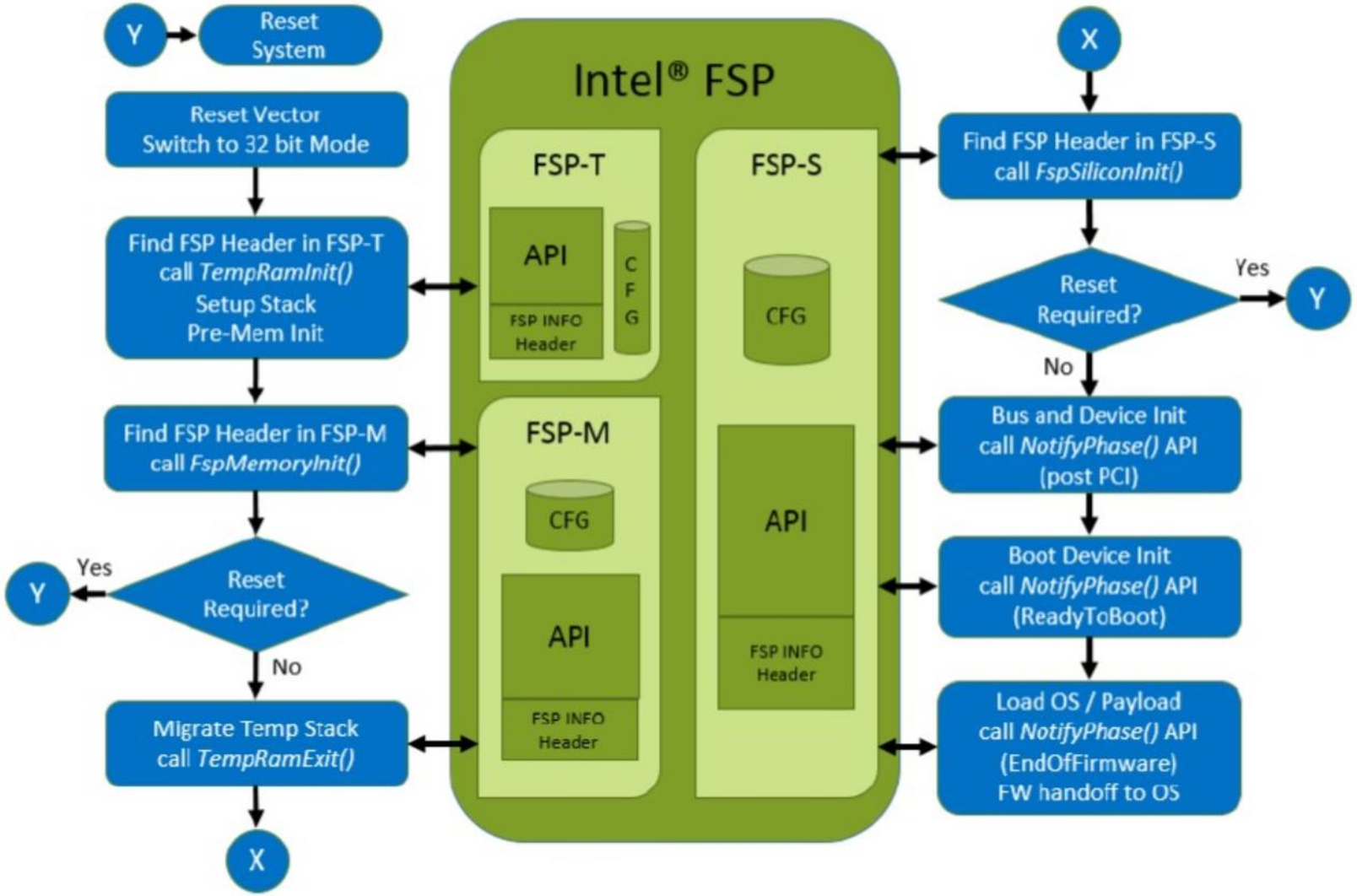
Minimum Platform overview



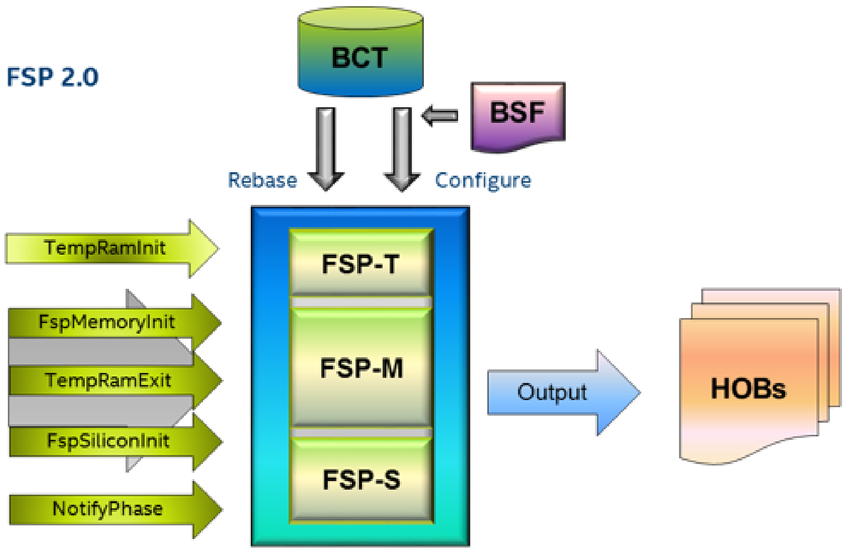
Developer Focused
Easier to share and secure code

Consistent boot flows and interfaces
Approachable across the ecosystem
Scalable from pre-silicon **to derivatives**

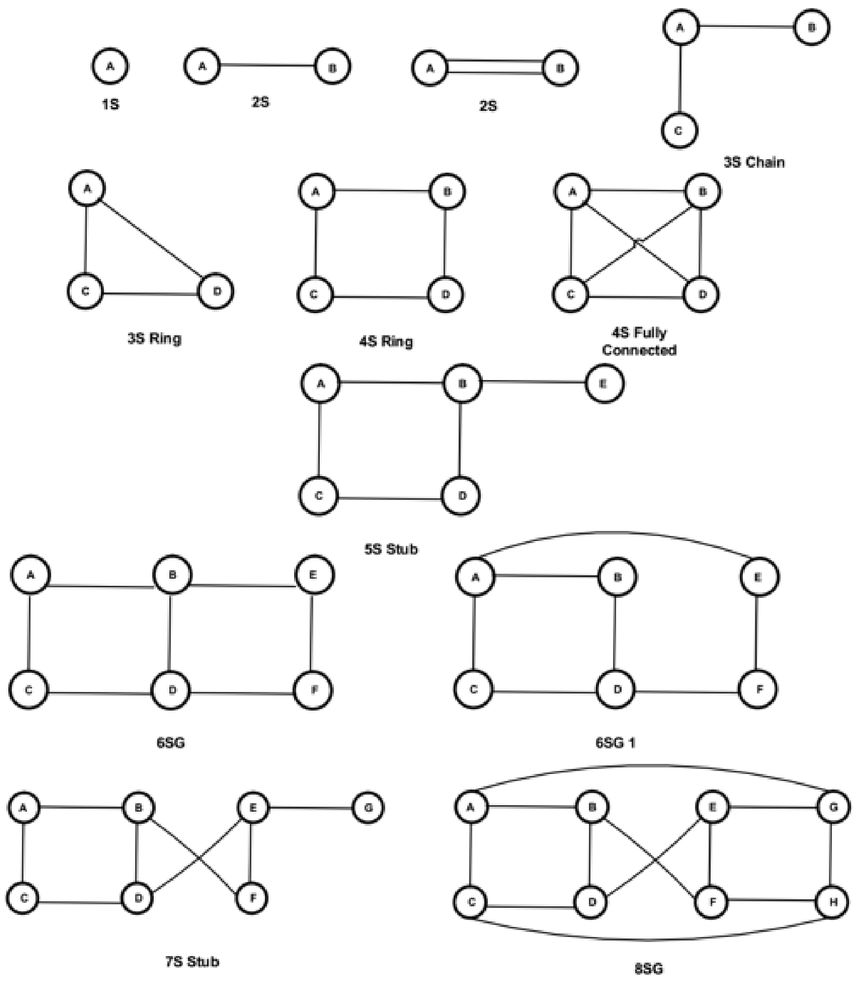
https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/MinPlatformPkg
https://github.com/tianocore/edk2-non-osi/tree/devel-MinPlatform/Silicon/Intel/PurleySiliconBinPkg
https://github.com/tianocore/edk2


Y
Reset System
Reset Vector
Switch to 32 bit Mode
Find FSP Header in FSP-T
call TempRamInit()
Setup Stack
Pre-Mem Init
Find FSP Header in FSP-M
call FspMemoryInit()
Reset Required?
Yes
Y
No
Migrate Temp Stack
call TempRamExit()
X
Intel® FSP
FSP-T
API
FSP INFO Header
CFG
FSP-M
CFG
API
FSP INFO Header
FSP-S
CFG
API
FSP INFO Header
X
Find FSP Header in FSP-S
call FspSiliconInit()
Reset Required?
Yes
Y
No
Bus and Device Init
call NotifyPhase() API
(post PCI)
Boot Device Init
call NotifyPhase() API
(ReadyToBoot)
Load OS / Payload
call NotifyPhase() API
(EndOfFirmware)
FW handoff to OS

FSP Extensions for Servers



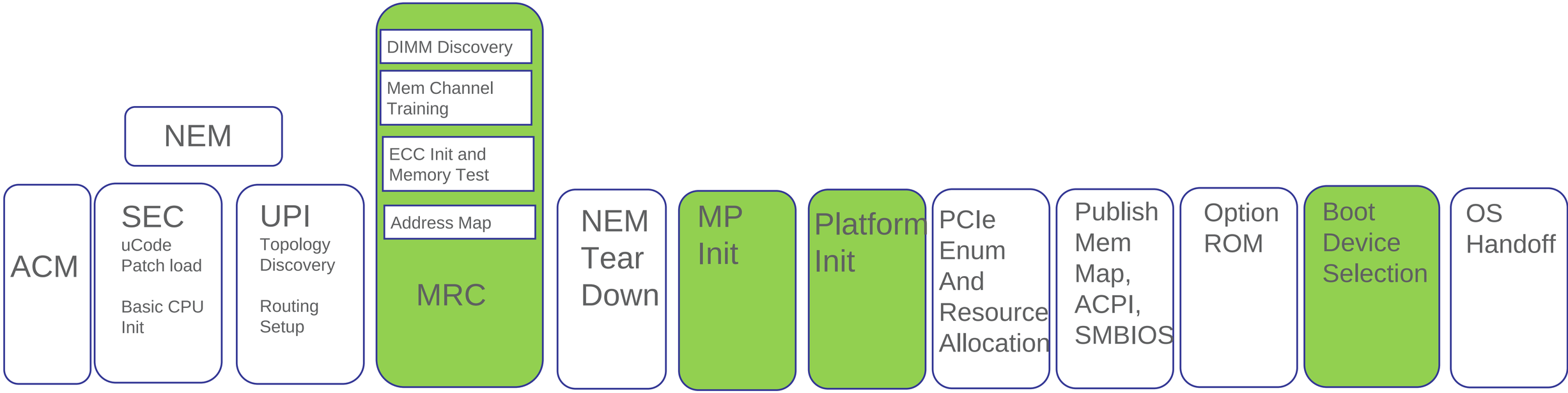
1. Out of Phase processor check-in at Reset Vector
2. Multi-Socket parallel MRC
3. Early and Late IIO Init (Pre and Post Memory)
4. Platform Abstraction (PCIe HP Slots, Retimers etc)
5. Runtime Handling



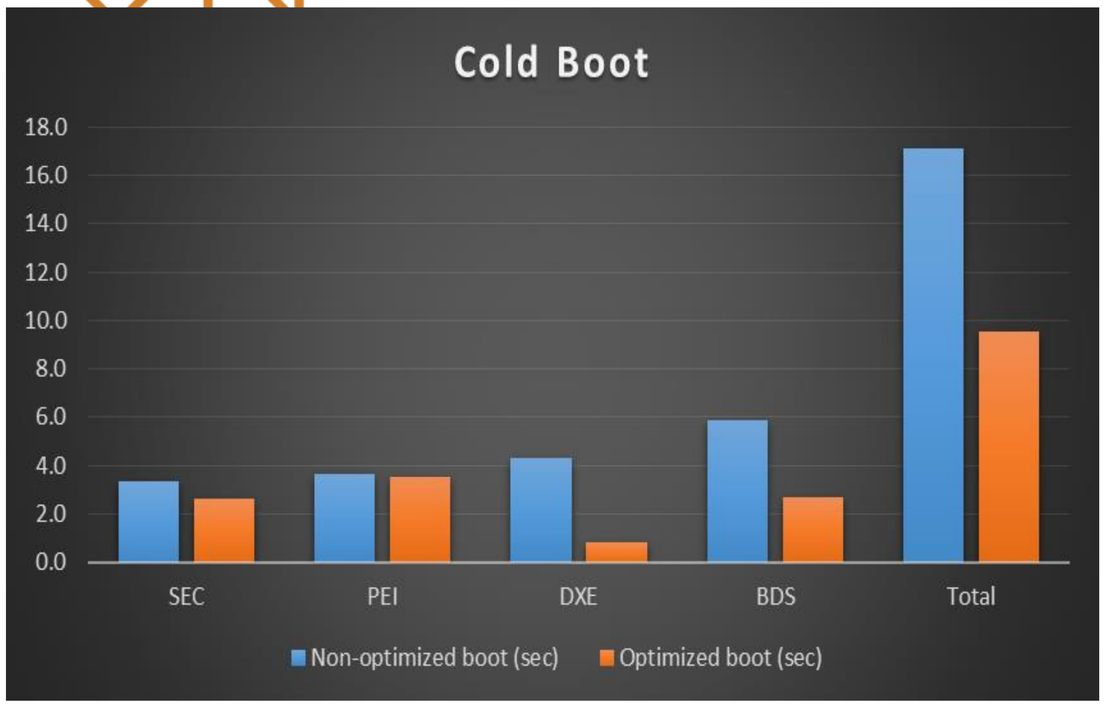
Boot Time Optimization

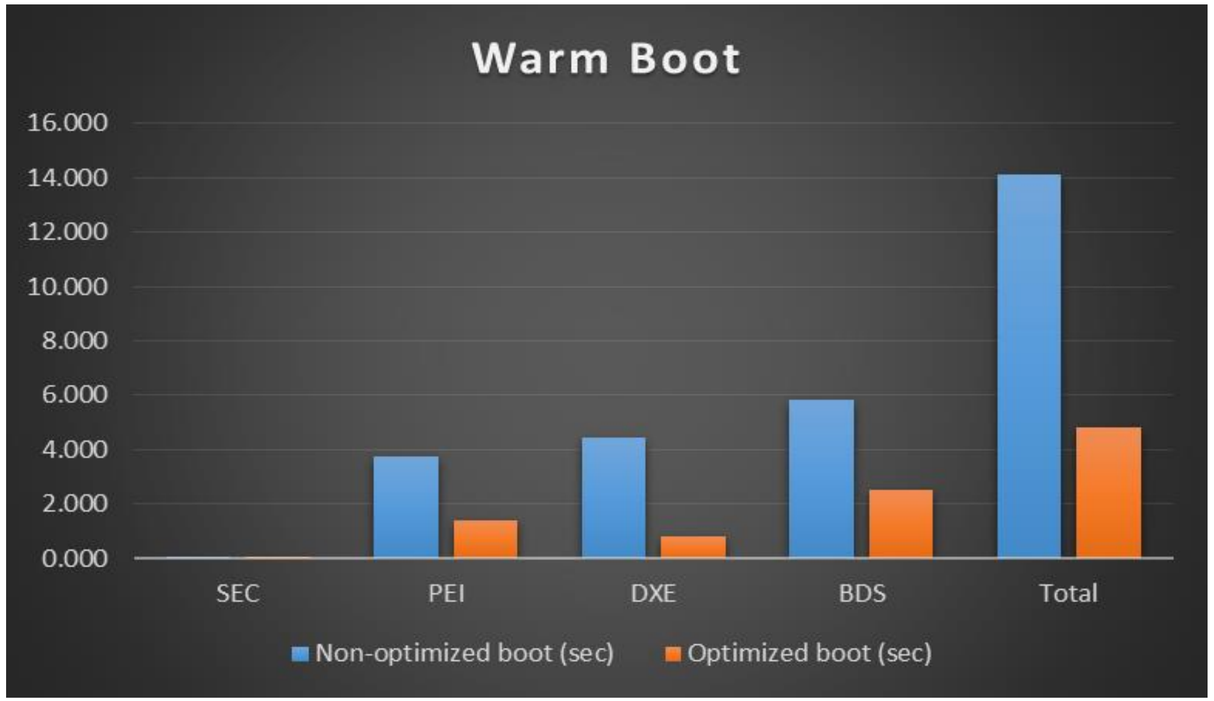
Boot Time Optimization

Boot Time Optimization Touch points



Boot optimized Purley MinPlatform





Mt. Olympus Platform with 32GB of memory, SKX Processor
OS handoff to OS Load – 13sec

Boot Optimizations…

- BMC Communications
- Optimized flash settings
- Memory ECC init on cold boot only
- BDS Fast boot path

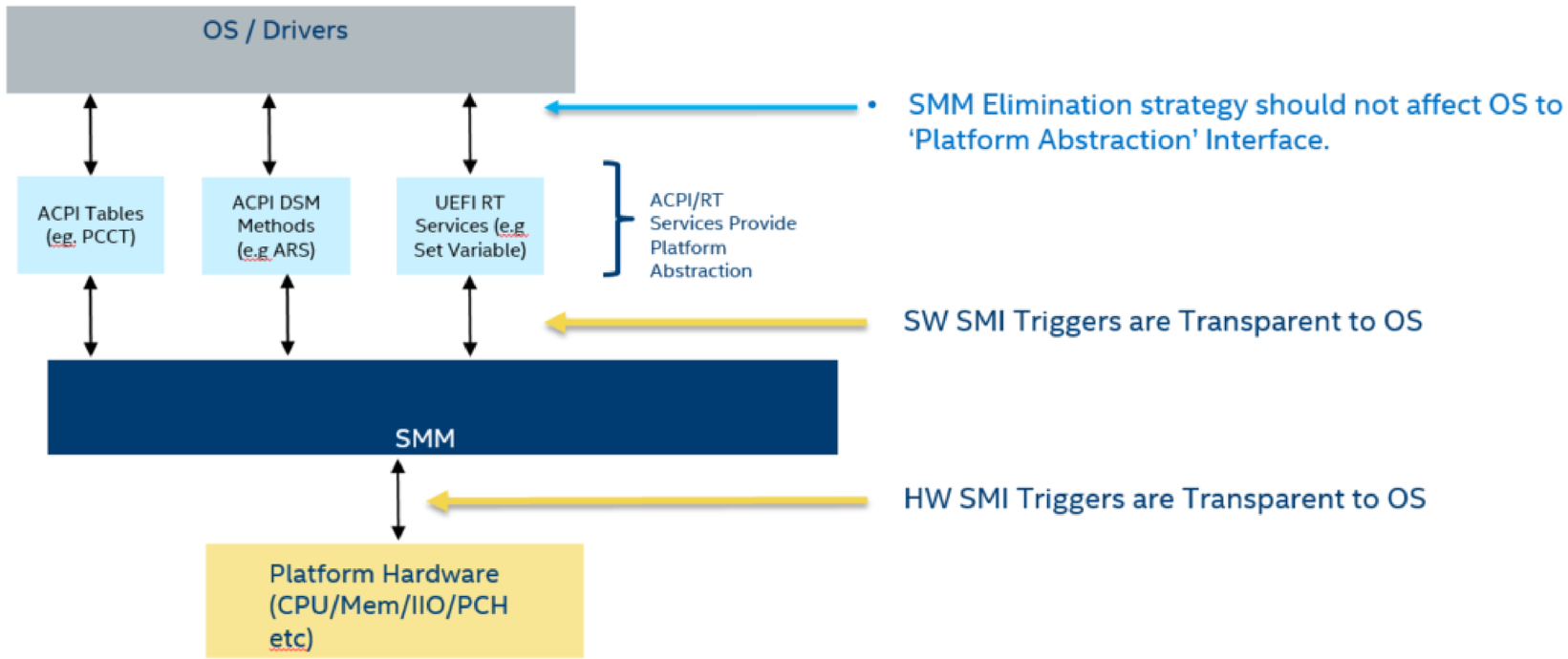
Please visit Azure Booth for more info

Proposal for reducing SMM Usage

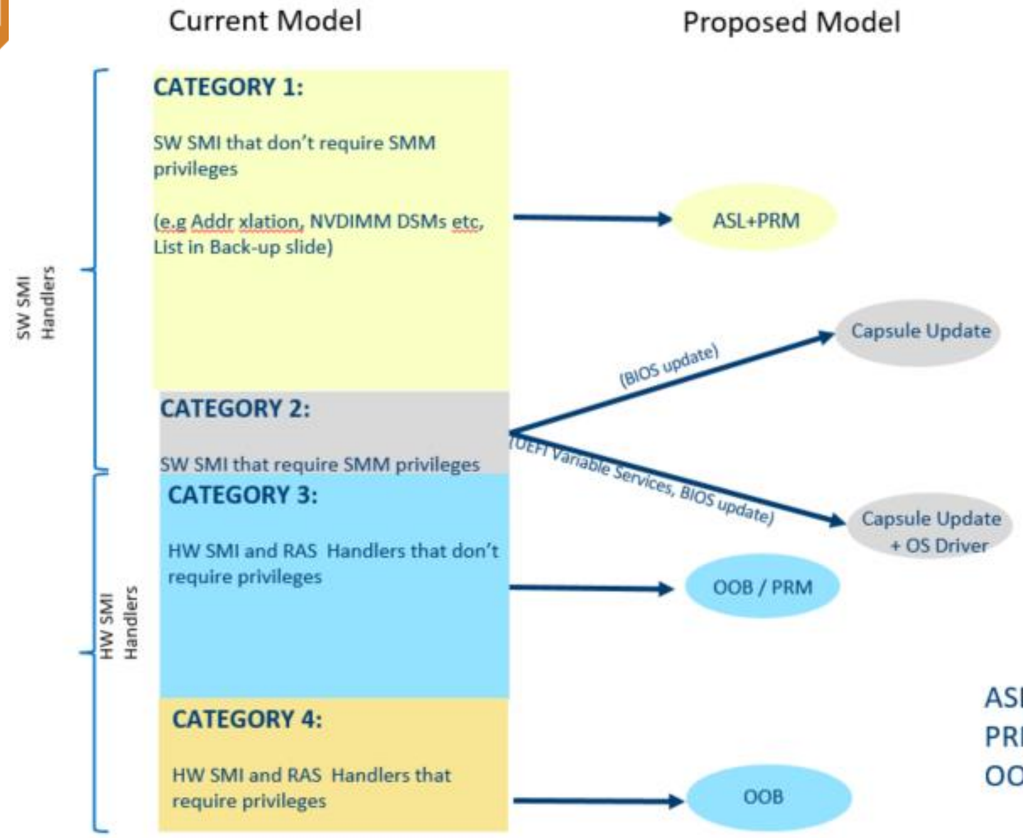
Problem Summary

- Broadcast SMM model has served the industry well for many generations, but running into some fundamental problems
  - **Perf/QoS degradation**
    - SMM latency increases with core count.
    - Several RAS features require platform FW to monitor *every* correctable error.
  - **Firmware Complexity**
    - SMM was never designed to handle so many asynchronous events in many-core environment.
    - No concept of interrupt priority, edge/level or reentrancy
    - Discovering more race conditions as the number of RAS events and new features are added.
    - SMM handler code has crossed complexity threshold, SW can’t take advantage of HW enhancements.
  - **Security concerns**
    - SMM code is more privileged than VMM/OS and an attractive target for attackers.

OS View of SMM



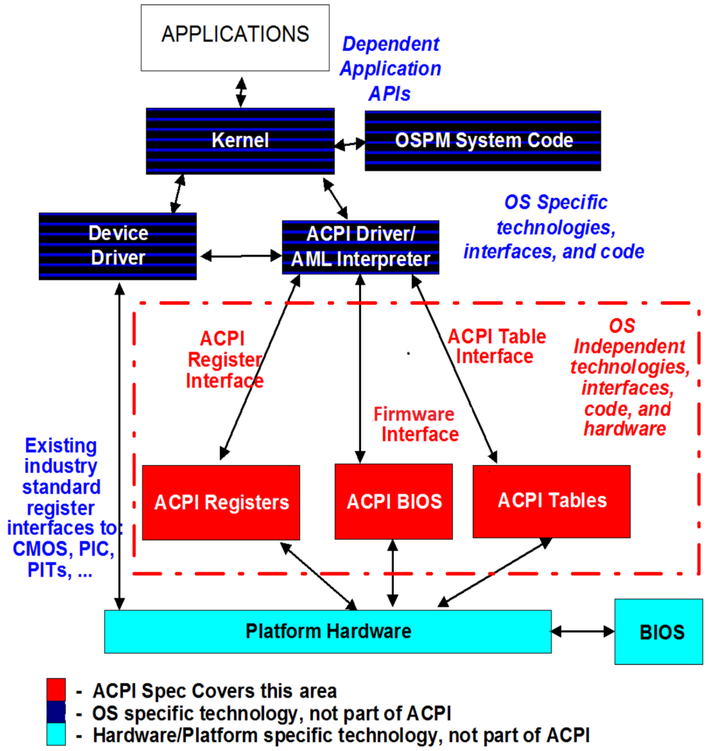
Categories of SMM Handler



Category	Elimination Proposal	Comments
1	ASL + PRM	e.g Addr Traslation, eLog, NVDIMM DSM etc.
2	Capsule Update	BIOS and Variable Update are the only two usages
3 and 4	BMC	Error Handling. Server implementation might choose to use BMC assistance.

ASL – ACPI Source Language
PRM – Protected Runtime Mechanism (Explained in Subsequent Slides)
OOB – Out of Band (e.g BMC)

Category 1: Background on ASL and SMM Usage



- Platform Event Handling in runtime requires intimate platform knowledge
- ACPI provides mechanism for runtime platform abstraction (ASL)
- ASL is very primitive interpreted code, and very limited in scope
  - ASL is architecture neutral, hence can't do architecture specific tasks
  - To overcome the ASL limitations, SMI is invoked to provide native code execution
  - ASL is slow, hard to develop, maintain and debug
- PRM (Protected Runtime Mechanism) provides a secure mechanism for ASL extension
  - Makes usage of ASL viable, eliminating the need to drop into SMI
  - Platform Runtime code will be carried as part of the BIOS image (tied to the platform)

Why not Driver Based Model

1. OSVs do not want to carry platform specific knowledge in the driver
2. Need intimate platform and silicon knowledge to carry out certain tasks (such Address Translation for RAS)
3. These vary between platform generations, and even between implementations in the same generation

Examples:

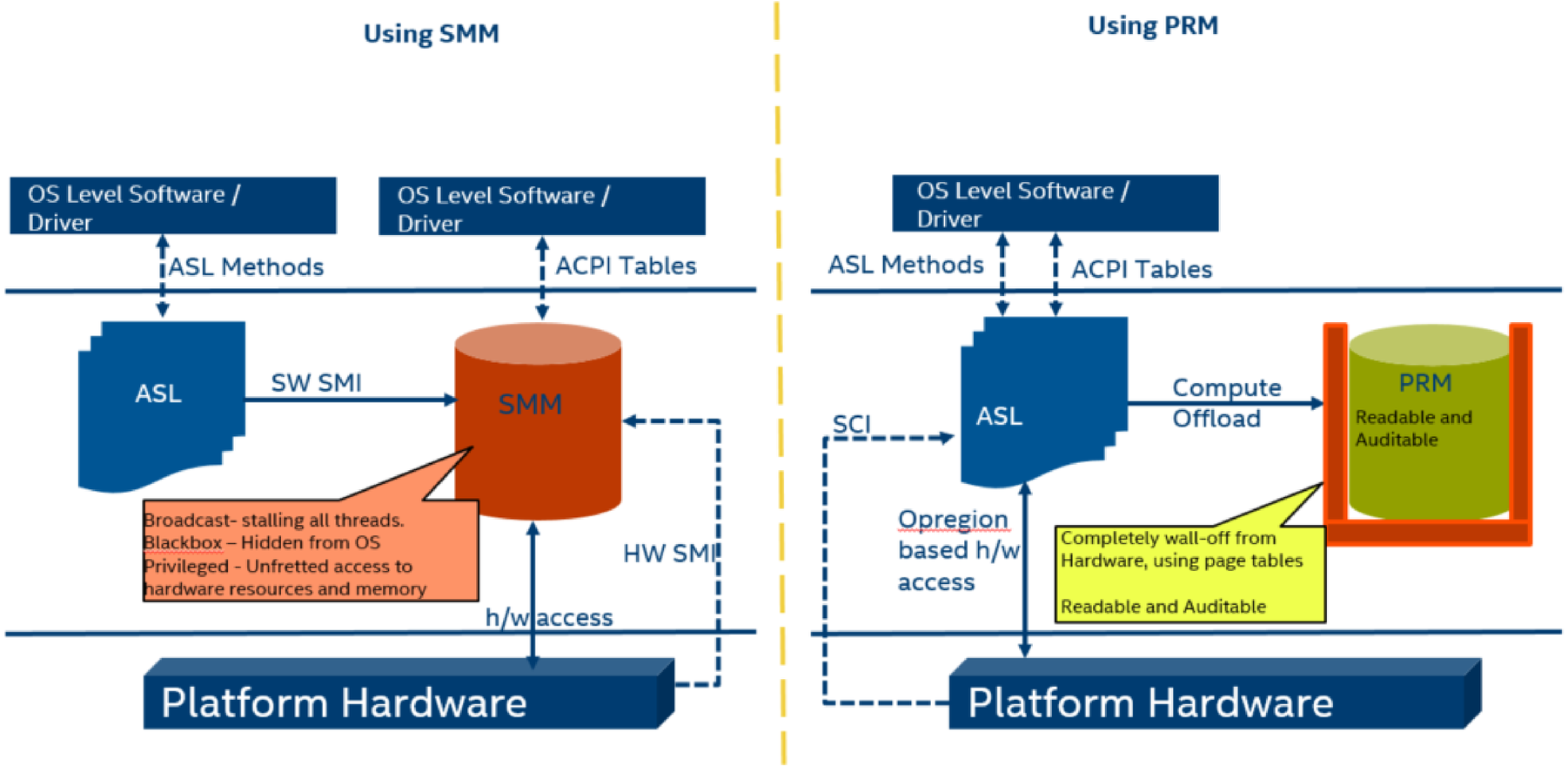
1. PSHED Plug-in
   - Not a viable deployment model and hence ACPI abstraction was used which drops into SMI for handling complex tasks
2. Address Translation
   - Originally pushed to EDAC drivers
   - OSVs are pushing it back into ACPI to keep the driver generic
   - The ACPI method drops into SMI to handle complex algorithms.

What is Protected Runtime Mechanism

- PRM is a mechanism to invoke native code from ACPI context
- ASL as the landing point for SW or HW based runtime events
- ASL will invoke PRM if required (think of this as ASL Assist)

- ***PRM is not a new capability***
- ***It is piecing together of existing capabilities to create a mechanism.***

SMM vs PRM


Using SMM
OS Level Software / Driver
OS Level Software / Driver
ASL Methods
ACPI Tables
ASL
SW SMI
SMM
Broadcast- stalling all threads.
Blackbox – Hidden from OS
Privileged - Unfretted access to hardware resources and memory
HW SMI
h/w access
Platform Hardware
Using PRM
OS Level Software / Driver
ASL Methods
ACPI Tables
SCI
ASL
Compute Offload
PRM
Readable and Auditable
Opregion based h/w access
Completely wall-off from Hardware, using page tables
Readable and Auditable
Platform Hardware

SMM Vs PRM …Continued

	SMI	PRM
Global/Local	SMI is global. All the threads are stalled.	Behaves like a device interrupt (SCI). Only one thread executes the handler
Privilege	SMM has unlimited access to system resources	PRM is a contained environment (Ring-0)
Resource conflict	Non Atomic IO (Index/Data) access	ASL Based access to hardware resources. OS aware.
Locked registers	SMM has access to all the locked registers	PRM can’t modify locked registers.
VMM Security	VMM pass-through of SW SMI leads to attack vectors	PRM prevents this by restricting access to memory
Blocking	SMM is blocking. No other interrupts can make progress	PRM runs in a kernel driver context. So, can be interrupted by high priority interrupts
Visibility	Black Box	Visible to OS (just link another kernel mode driver).

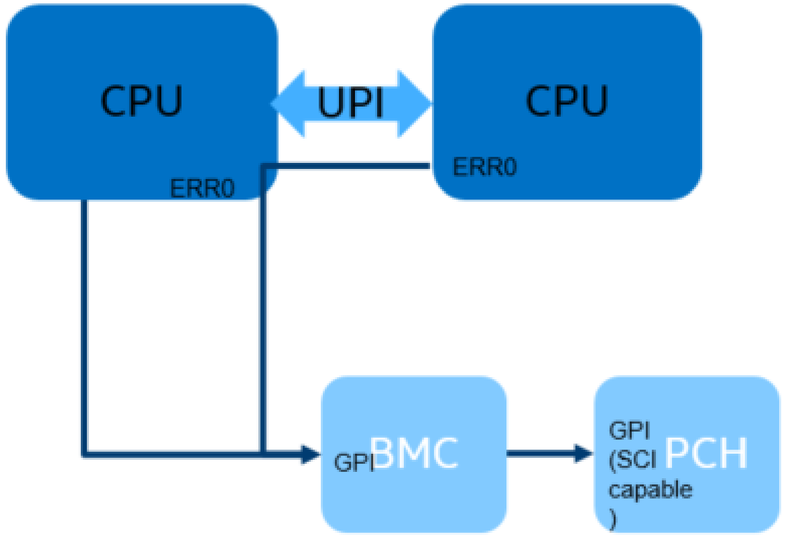
Category 2: Authenticated Variables and BIOS Update

- Authenticated Variables
  - Consumed by BIOS during boot (e.g SMT enable/disable)
  - Use capsule update style for updating the authenticated variables
  - The RO region for Authenticated Variables gets locked out at EndofDXE
  - Staged by the OS, updated on reset and BIOS acts on the settings

- BIOS Update
  - Capsule based update

<u>Server Class systems might choose to use BMC/OOB based Variable and BIOS update based on Redfish standard.</u>

Category 3 and 4: HW SMI



Configure CPU so that correctable errors assert ERR0 pin instead of SMI

ERR0 from all sockets connected to BMC GPI, configured to interrupt BMC FW

Feedback?

- Let’s together accelerate OSF Development
- Increase adoption of MinPlatform
- Get Involved
- Feedback? How can we help

OCP SUMMIT

OPEN. FOR BUSINESS.