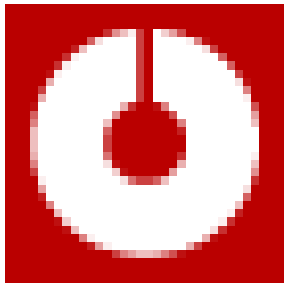


CanSecWest 2015 Vancouver, Canada

UEFI, Open Platforms, and the Defender's Dilemma

Vincent Zimmer

@vincentzimmer, vincent.zimmer @intel.com | @gmail.com

Agenda

- Background on UEFI
- Security Features
- Trust Model
- EDK II on MinnowMax
- EDK II on Galileo
- Futures

Background on UEFI

Old Day

19XX

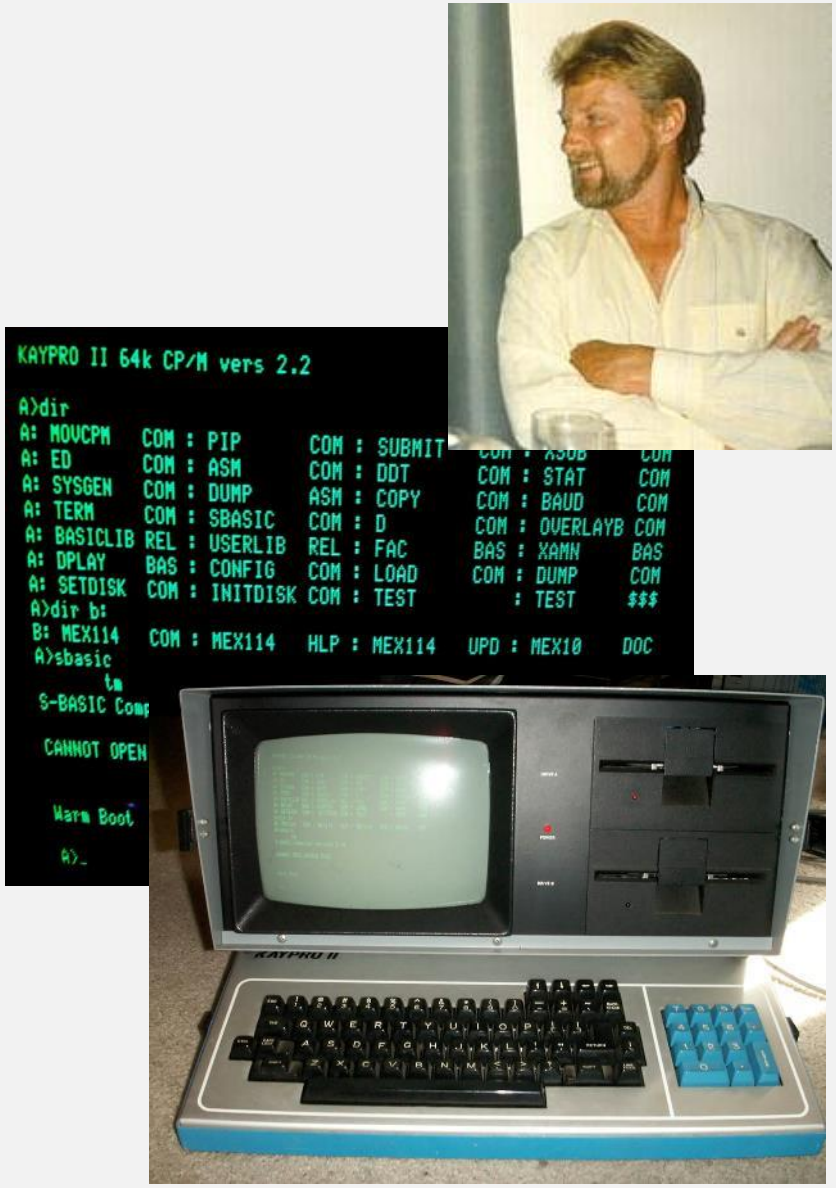
Pioneer

CP/M

BIOS
(machine specific CP/M)

8080/Z80

1974 Basic I/O (Sub) System
by Gary Kildall in CP/M



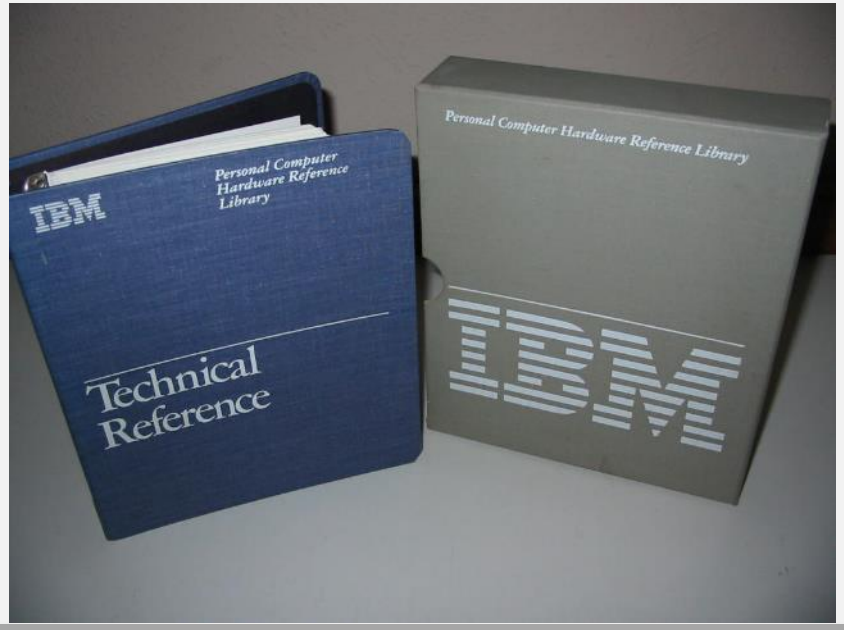
PC/AT BIOS

DOS

BIOS
(de facto standard)

8088

1981 IBM PC



PC/AT BIOS -> EFI

IPF Windows/Linux

EFI

(Intel Standard)

IPF (Merced)

intel

Extensible Firmware Interface Specification

Version 1.02

December 12, 2000

2000 Extensible Firmware Interface

Intel/HP IPF

Broader adoption

Windows/Linux

UEFI

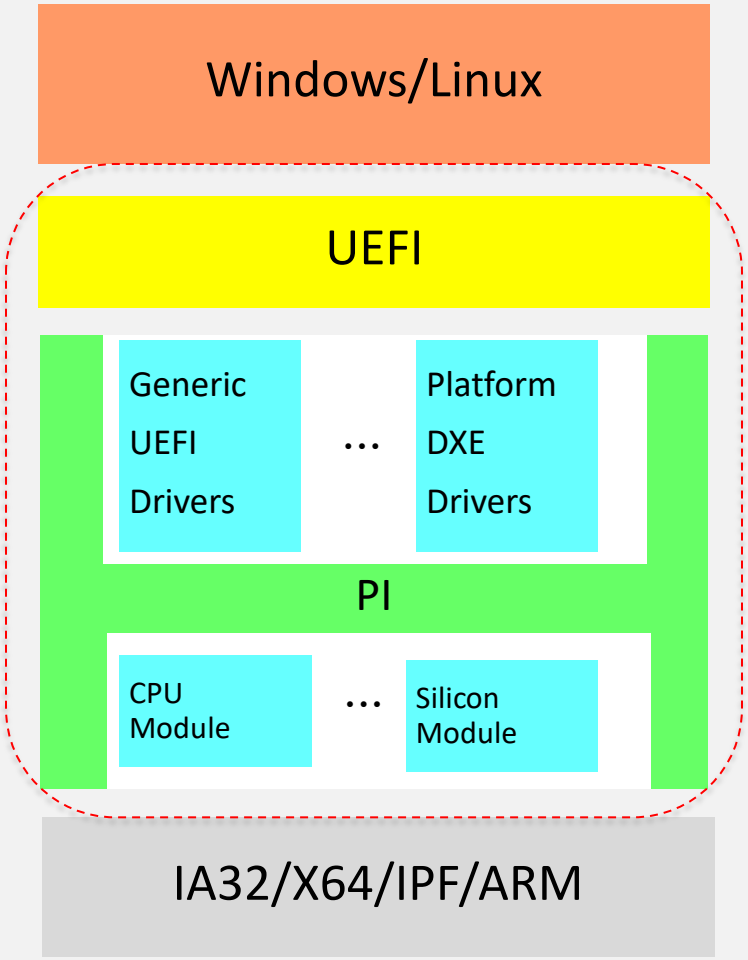
(Industry Standard)

IA32/X64/IPF/ARM

2006 January

Unified Extensible Firmware Interface

Today



UEFI + PI

2006 Aug Platform Initialization

Industry Transition

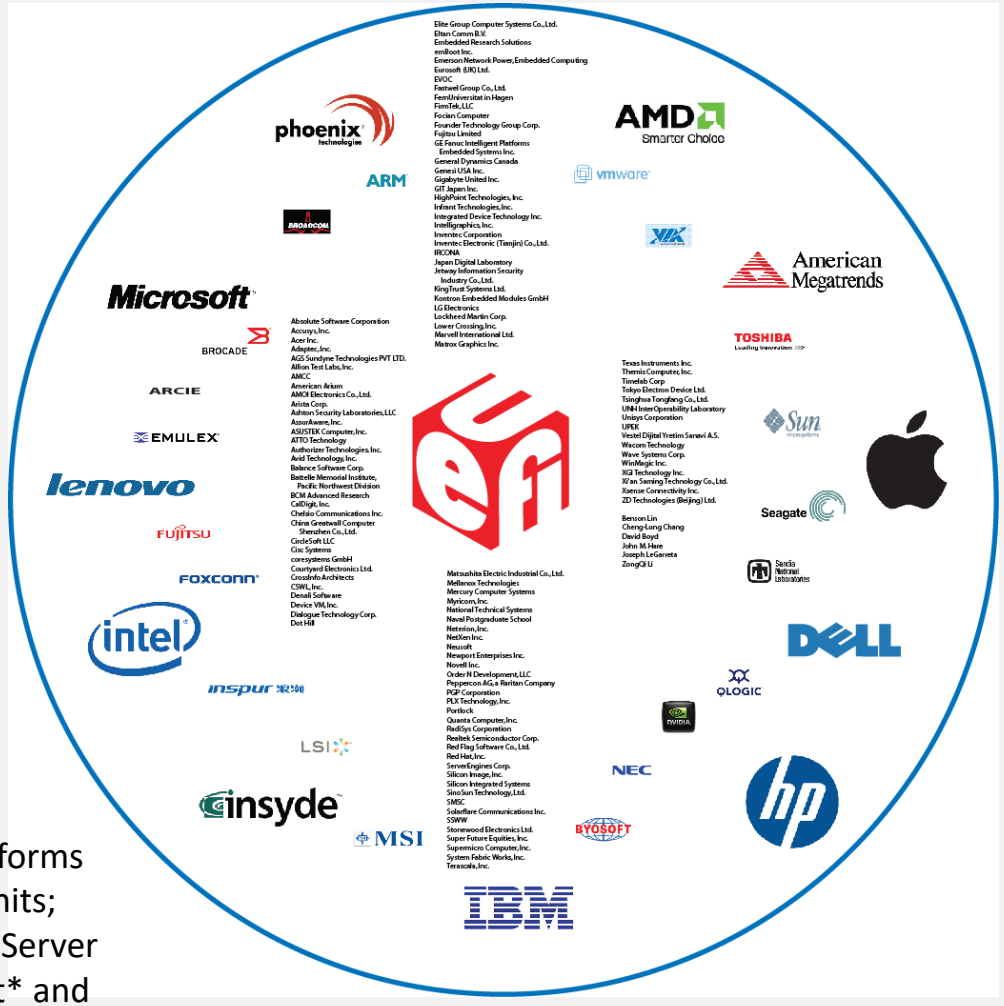
Pre-2000 All Platforms BIOS were proprietary

2000 Intel invented the Extensible Firmware Interface (EFI) and provided sample implementation under free BSD terms

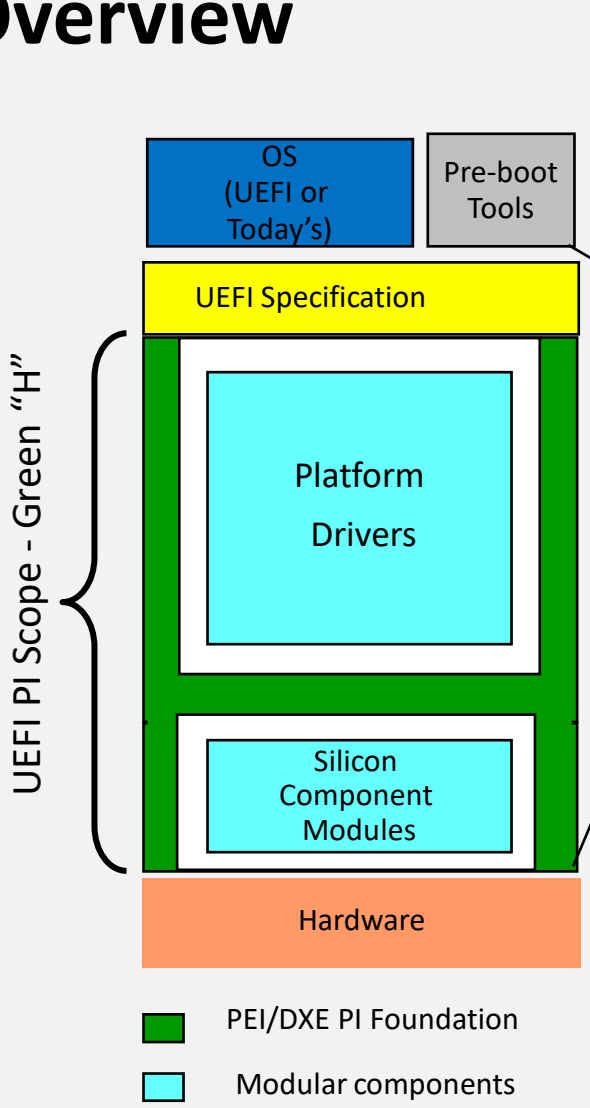
2004 **tianocore.org**, open source EFI community launched

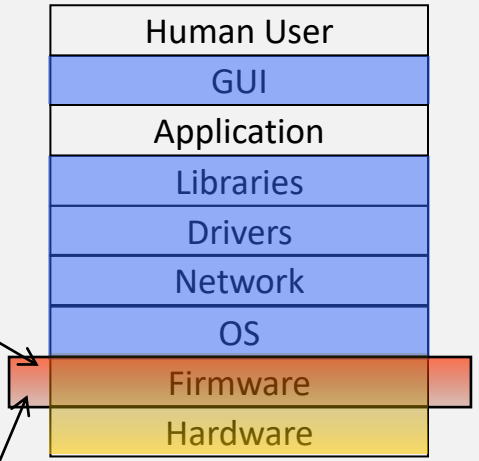
2005 **Unified EFI (UEFI)** Industry forum, with 11 members, was formed to standardize EFI

2015 240 members and growing! Major MNCs shipping; UEFI platforms crossed most of IA worldwide units; Microsoft* UEFI x64 support in Server 2008, Vista* and Win7*; RedHat* and SuSEl* OS support. Mandatory for Windows 8 client. ARM 32 and 64 bit support. ACPI added.



What is UEFI? UEFI Platform Initialization Overview





Full system stack
(user -> hardware)

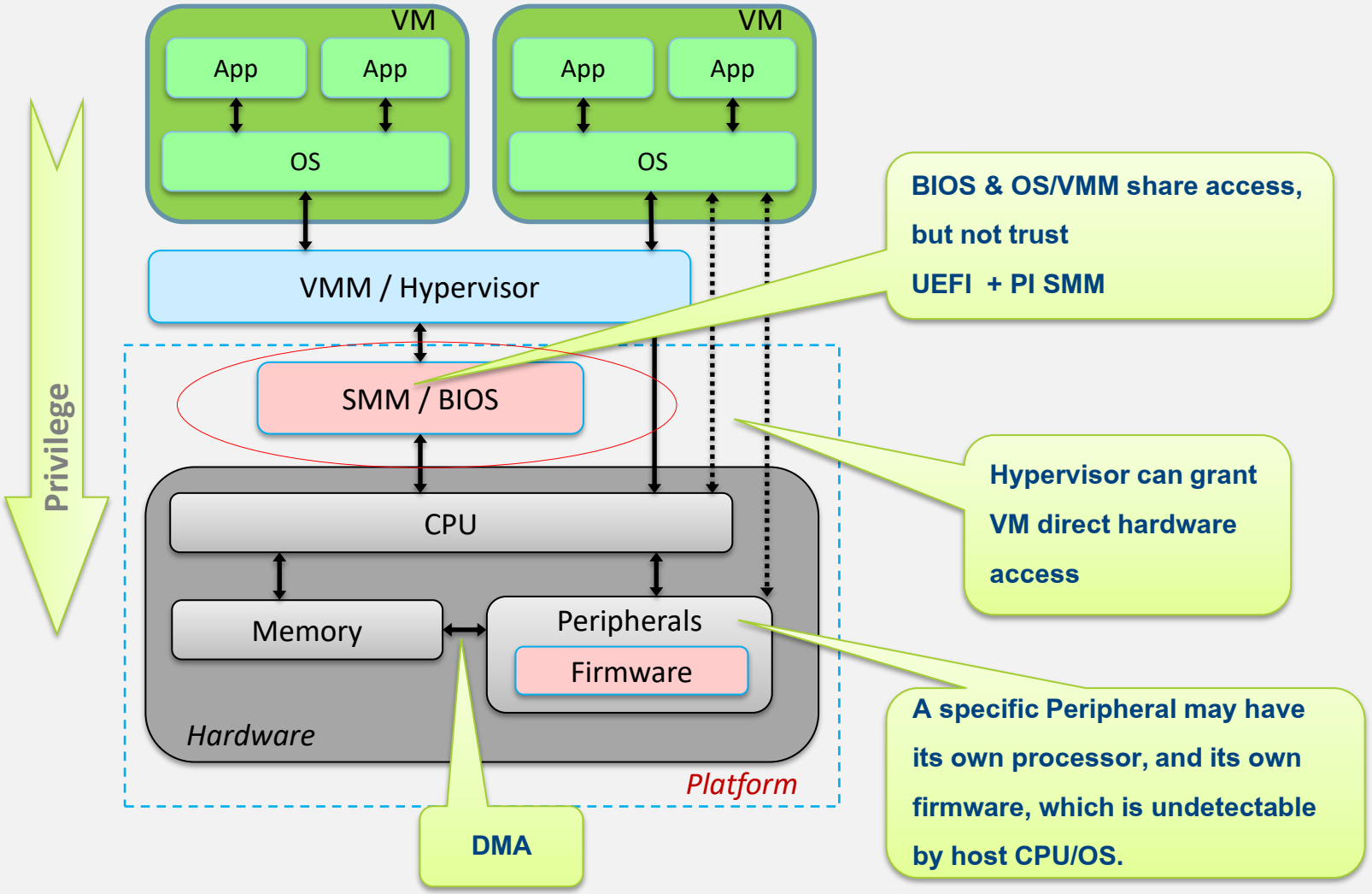
UEFI 2.4 specifies how firmware boots OS loader

UEFI's Platform Initialization (PI) 1.3 Architecture specifies how Driver Execution Environment (DXE) Drivers and Pre-EFI Initialization (PEI) Modules (PEIMs) initialize SI and the platform

DXE is preferred UEFI Implementation

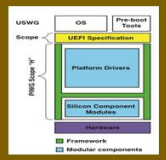
PEIMs, UEFI and DXE drivers implement networking, Update, other security features

Where are we (UEFI firmware)?



What's in UEFI

Mostly written in C.
High code re-use.

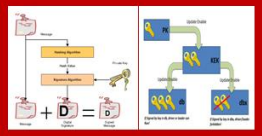


Emphasis on Specifications.
Standards compliance.

Better platform scaling. For e.g. removes shadow ROM limits.

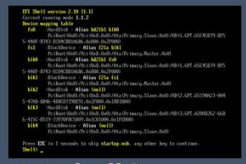
Storage.
GPT removes 2.2 TB MBR restriction.

CPU Architecture independent. Platform design flexibility.

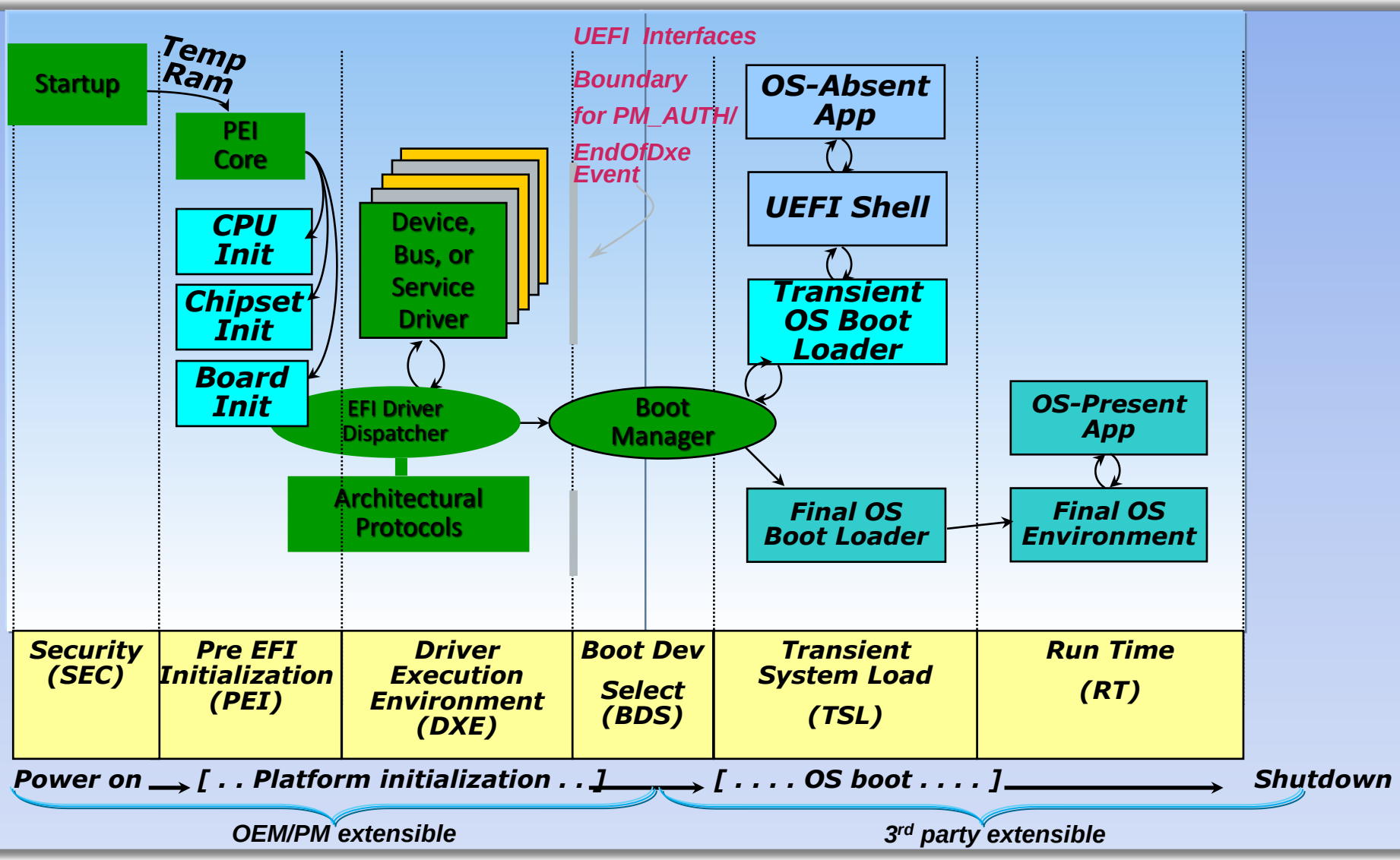


Secure boot solves "trust" related system integration challenges.

Pre-boot Networking.
Ipv4, Ipv6, PXE,
VLAN, iSCSI etc.

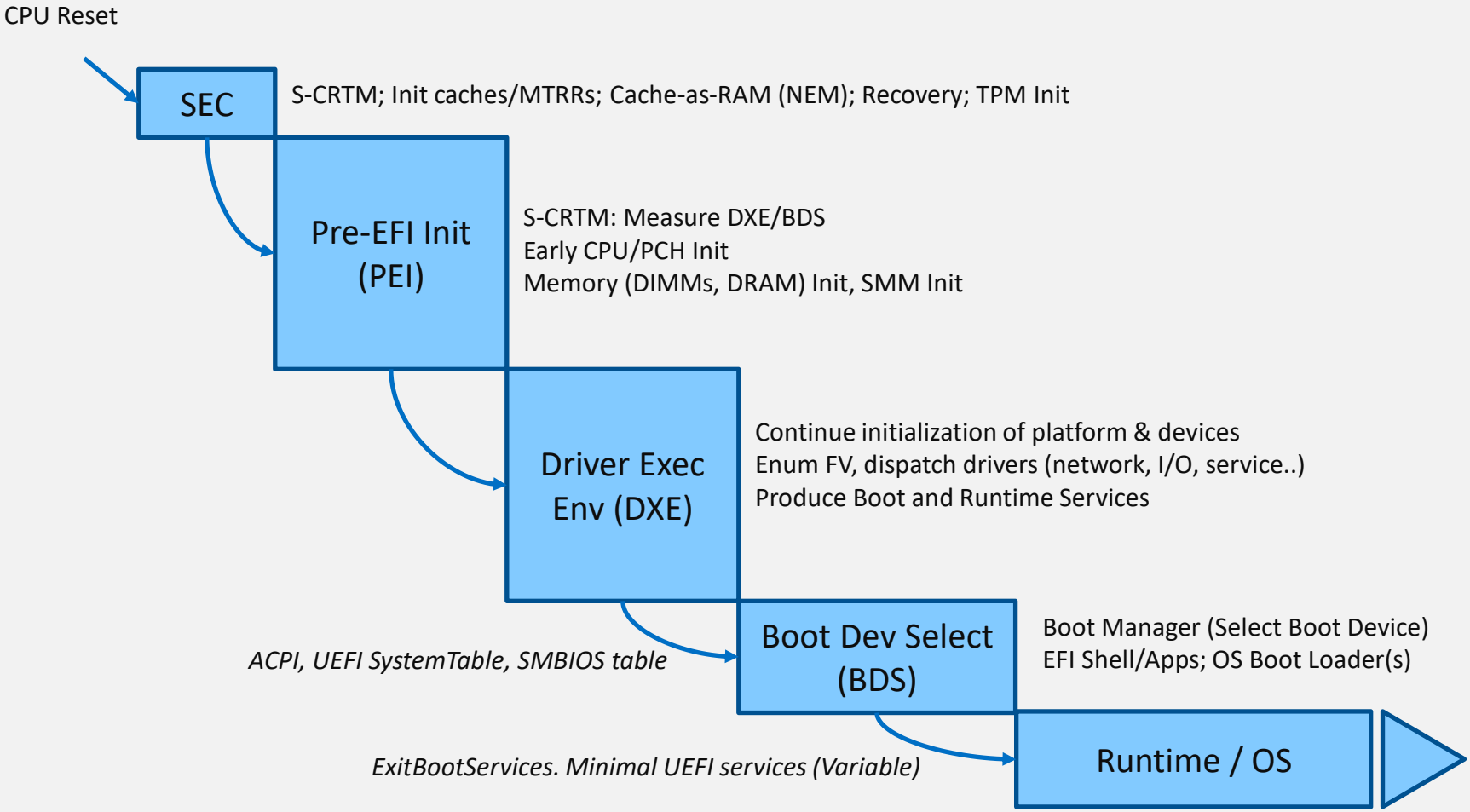


UEFI shell improves pre-boot testing & diagnostics experience.

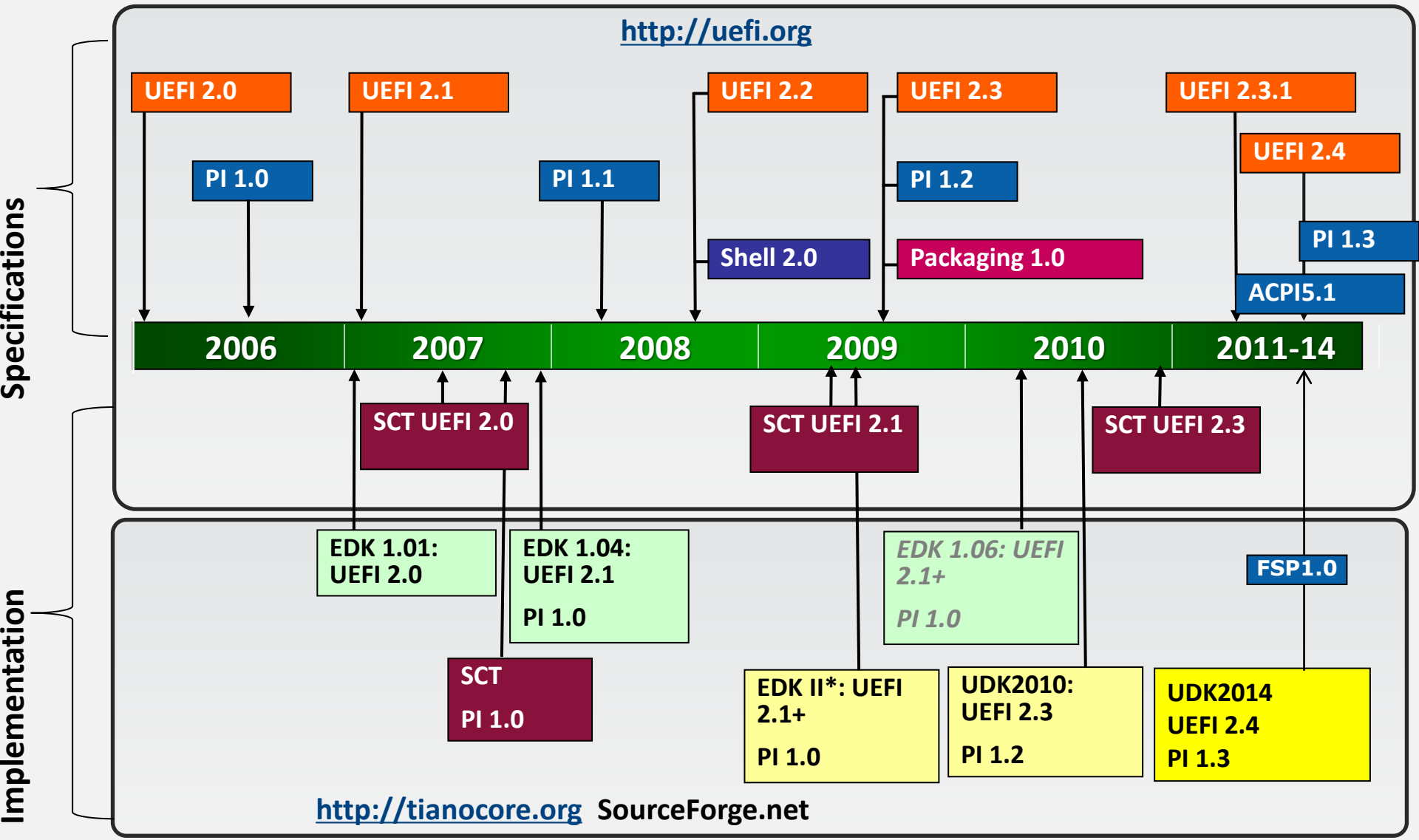


Overall UEFI Boot Timeline

UEFI [Compliant] Firmware

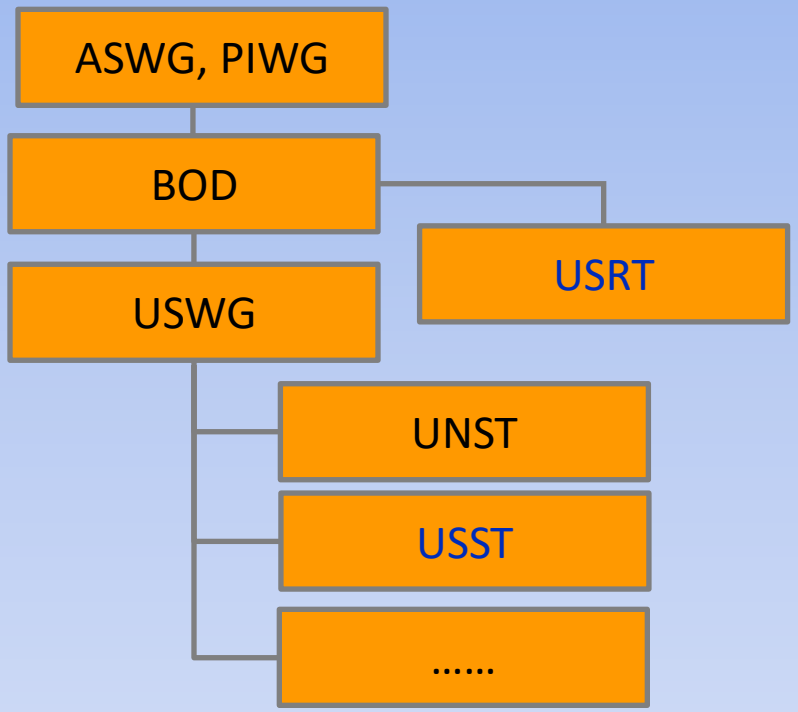


Specification & Tianocore.org Timeline



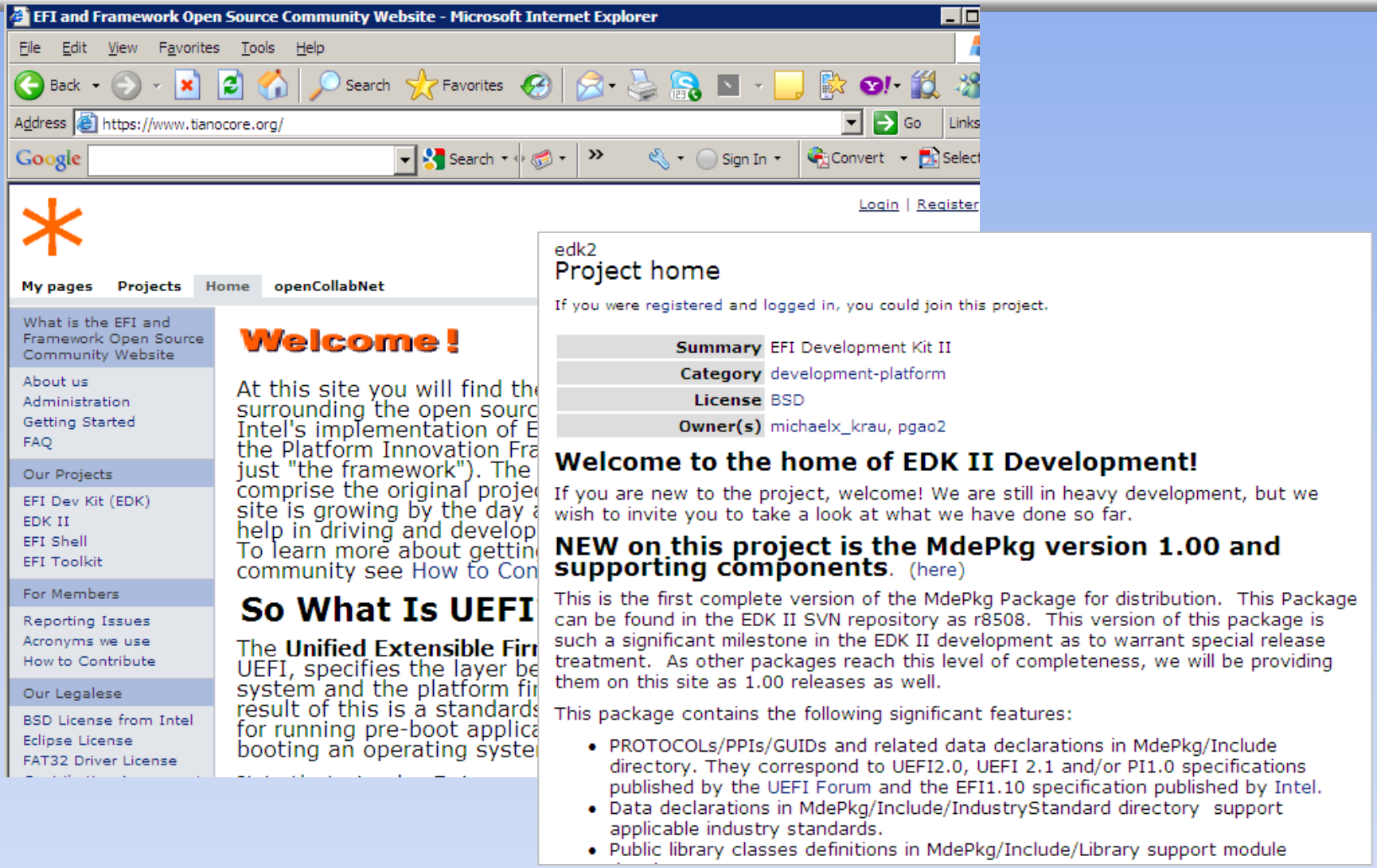
All products, dates, and programs are based on current expectations and subject to change without notice.

www.uefi.org



Note: Engaged in firmware/boot
Related WG's of Trusted Computing Group (TCG), IETF, DMTF

- **USWG**
  - **U**EFI **S**pecification **W**orking **G**roup
- **PIWG**
  - **P**latform **I**nitialization **W**orking **G**roup
- **ASWG**
  - **A**CPI **S**pecification **W**orking **G**roup
- **BOS**
  - **B**oard **O**f **D**irectors
- **USST**
  - **U**SWG **S**ecurity **S**ub-**t**eam
  - Chaired by Vincent Zimmer (Intel)
  - Responsible for all security related material and the team that has added security infrastructure in the UEFI spec
- **USRT**
  - UEFI Security Response Team
  - Chaired by Dick Wilkins (Phoenix)
  - Provide response to security issues.
- **UNST**
  - **U**EFI **N**etwork **S**ub-**t**eam (VZ chairs, too)
  - Evolve network boot & network security infrastructure for UEFI Specification



UDK2014 Available on Tianocore.org

How to build it? UDK2014

Industry Standards Compliance
- UEFI 2.0, UEFI 2.1, UEFI 2.2, UEFI 2.3, UEFI 2.4; PI 1.0, PI 1.1, PI 1.2, PI1.3, ACPI 5.1

Extensible Foundation for Advanced Capabilities
- Pre-OS Security
- Rich Networking
- Manageability

Support for UEFI Packages
- Import/export modules source/binaries to many build systems

Maximize Re-use of Source Code**
- Platform Configuration Database (PCD) provides "knobs" for binaries
- ECP provides for reuse of EDK1117 (EDK I) modules
- Improved modularity, library classes and instances
- Optimize for size or speed

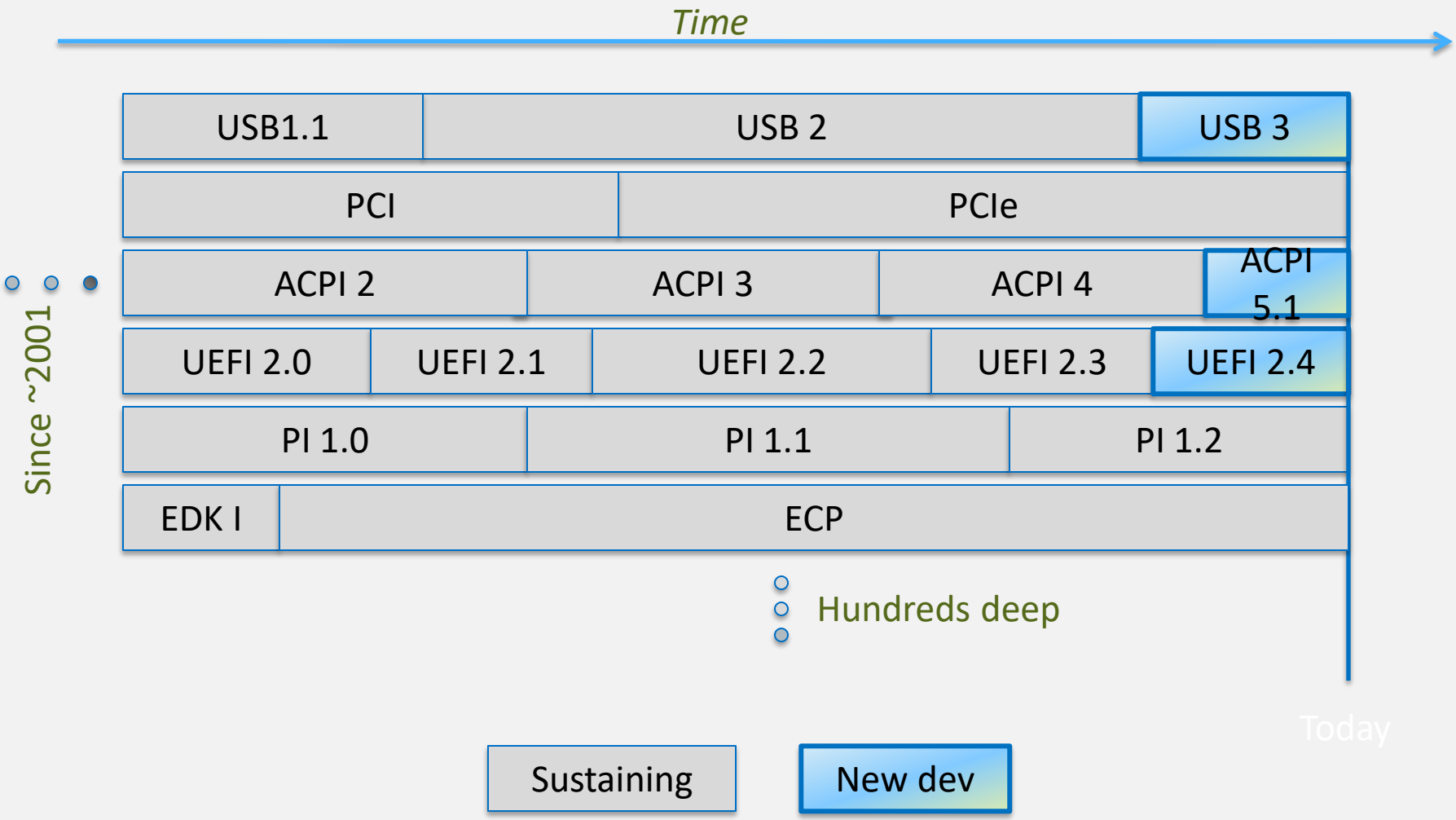
Multiple Development Environments and Tool Chains**
- Windows, Linux, OSX
- VS2003, VS2005, WinDDK, Intel, GCC

Fast and Flexible Build Infrastructure**
- 4X+ Build Performance Improvement (vs EDKI)
- Targeted Module Build Flexibility

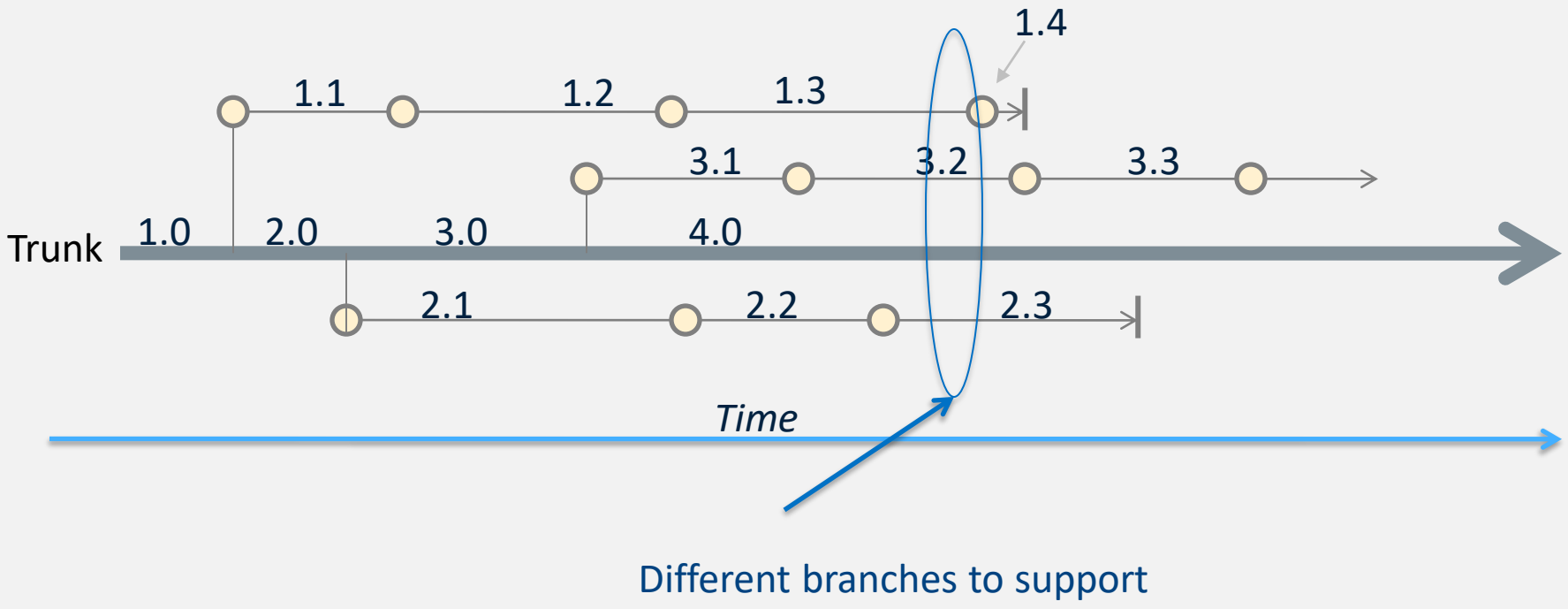
*** benefit of EDK II codebase*

Maximize the open source at www.tianocore.org

Contents

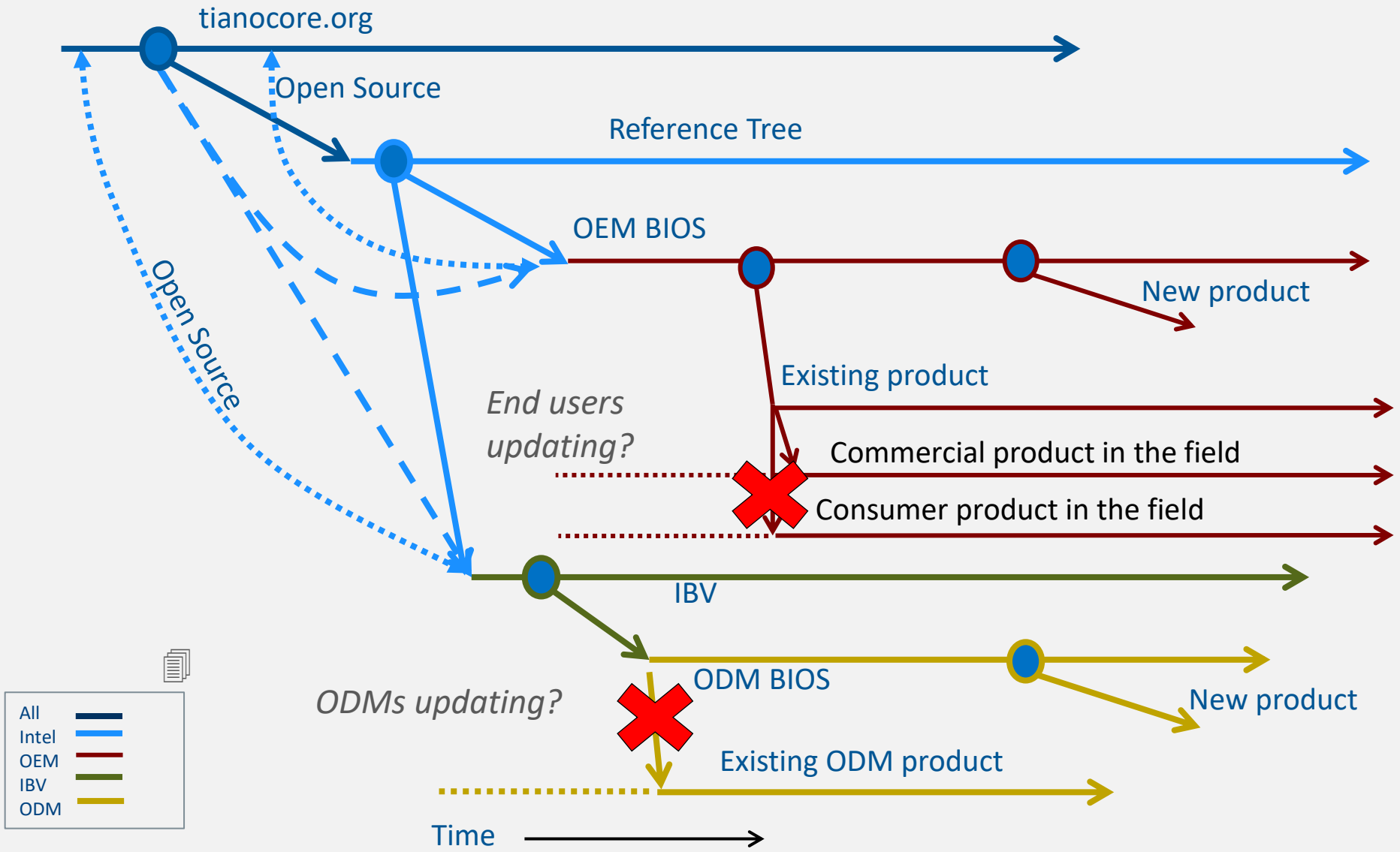

Time
USB1.1
USB 2
USB 3
PCI
PCIe
ACPI 2
ACPI 3
ACPI 4
ACPI 5.1
UEFI 2.0
UEFI 2.1
UEFI 2.2
UEFI 2.3
UEFI 2.4
PI 1.0
PI 1.1
PI 1.2
EDK I
ECP
Since ~2001
Hundreds deep
Today
Sustaining
New dev

Core evolution



Different branches to support

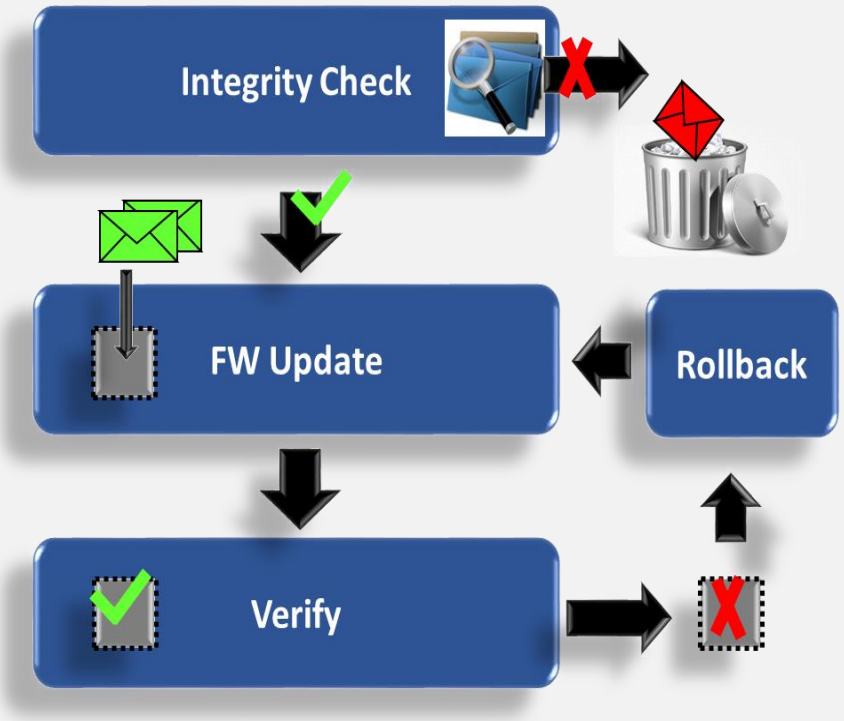
The road from core to platform



Security Features

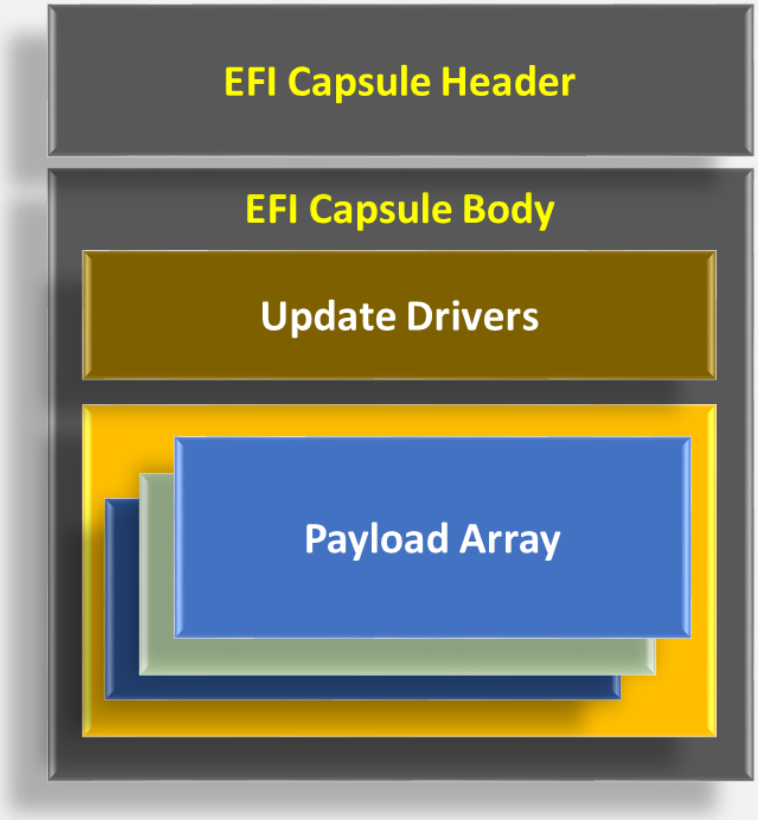
Solving Firmware Update

- Reliable update story
  - Fault tolerant
  - Scalable & repeatable
- How can UEFI Help?
  - Capsule model for binary delivery
  - Bus / Device Enumeration
  - Managing updates via
    - EFI System Resource Table
    - Firmware Management Protocol
    - Capsule Signing

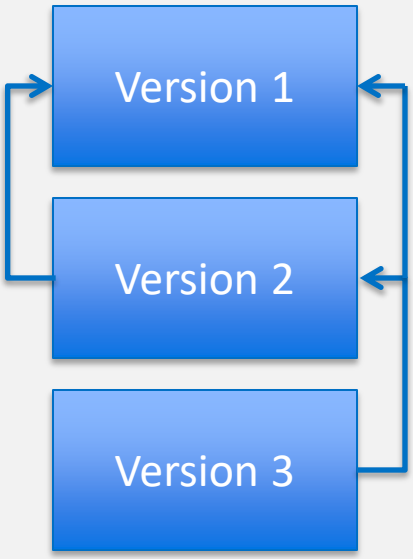


Delivering Firmware Binaries

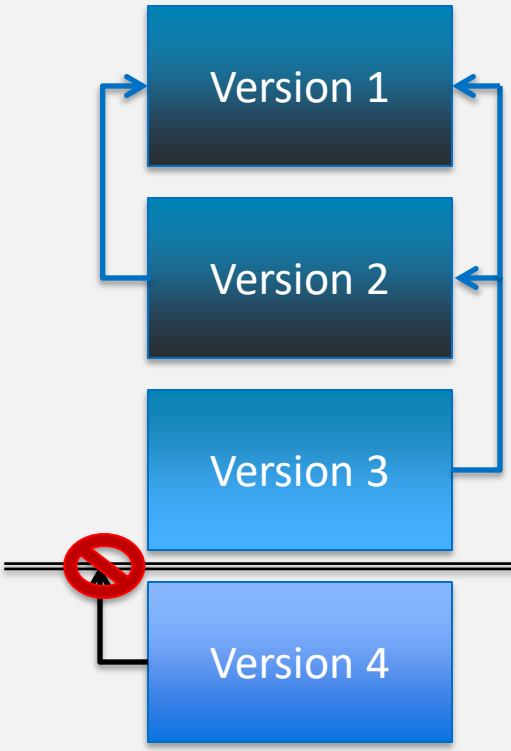
- UEFI supports Capsule format
  - Tools for capsule generation
  - Core logic for capsule handling
- Extensible Capsule format
  - Self-contained
  - Discrete updates
  - Composite updates
- Firmware Management Protocol allows
  - Reading / updating firmware
  - Integrity checks



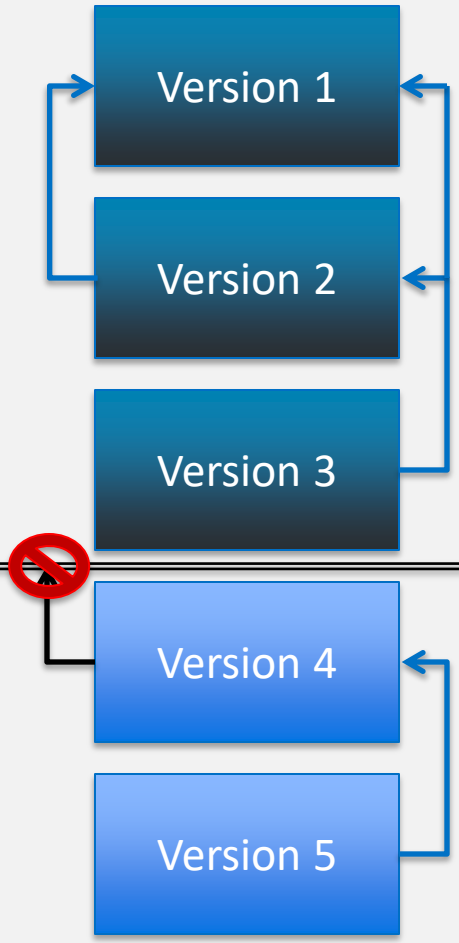
Rollbacks with fences



Each version fixes some issues with the previous. Since none are known to have security flaws, each new version allows updates to all older versions.



In V4, one of the issues fixed in V3 is realized to be a security fix. V4 will not allow updates to earlier versions, even V3 since it allows update to V2.

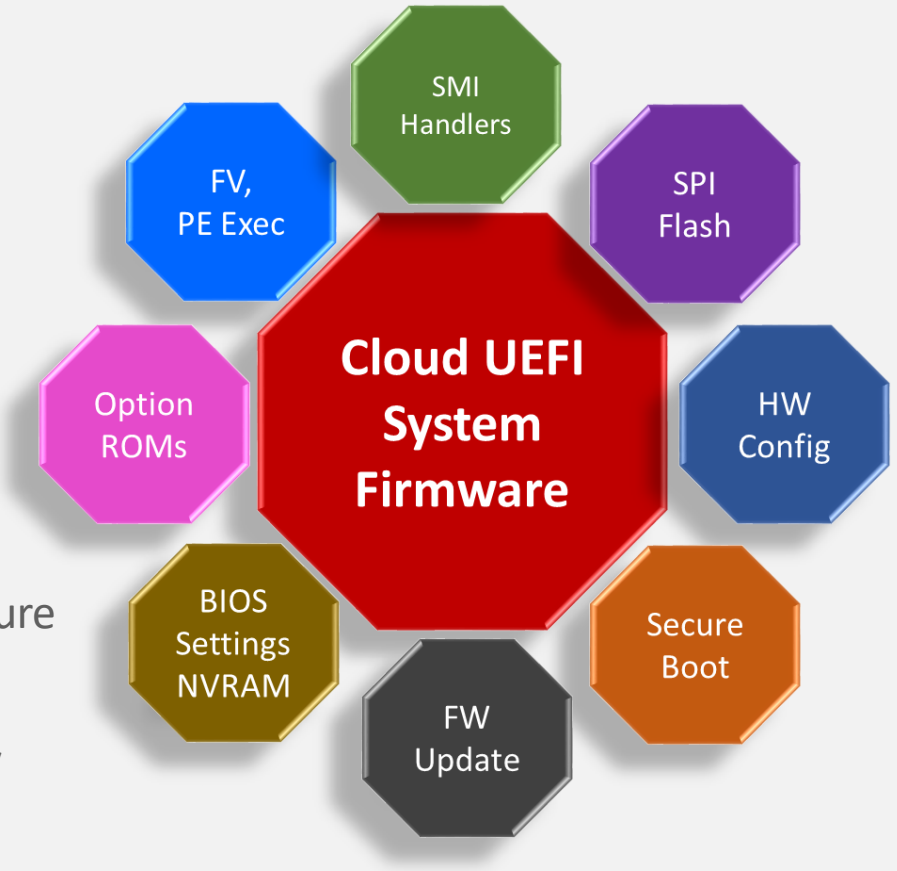


"Fence"

Version 5 can now accept only versions 5 and 4.

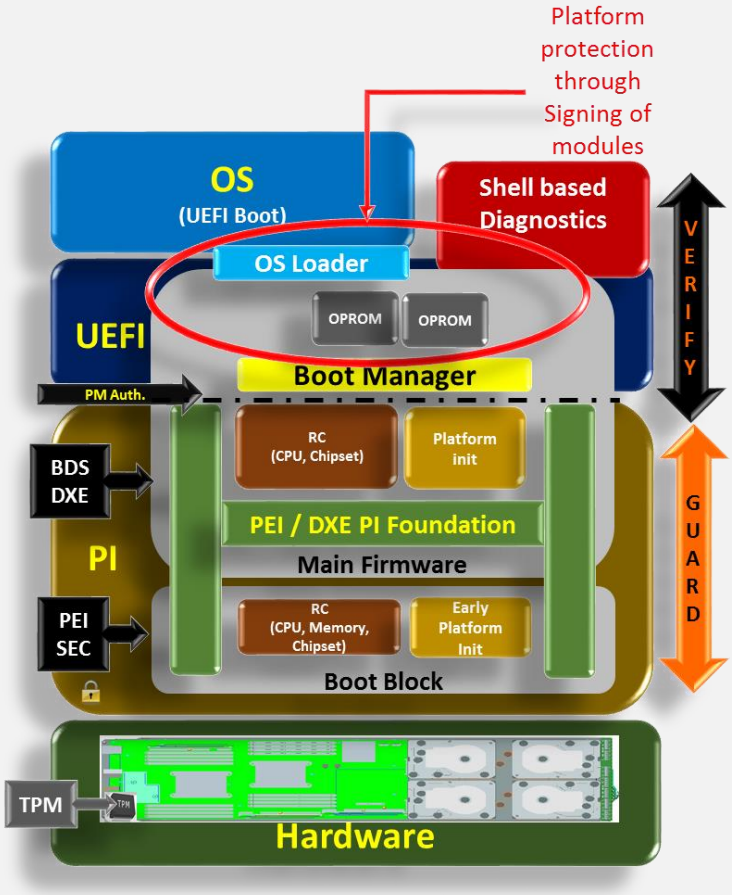
Security Solutions

- Signed capsule updates
- UEFI Secure boot
  - local / network
- TPM on the platform
  - Measured boot
  - Root of Trust for Reporting
  - Storage
- Protect machine configuration & UEFI Secure boot trust anchors
- In-band and out-of-band network security



Guarding & Verifying in Pre-boot

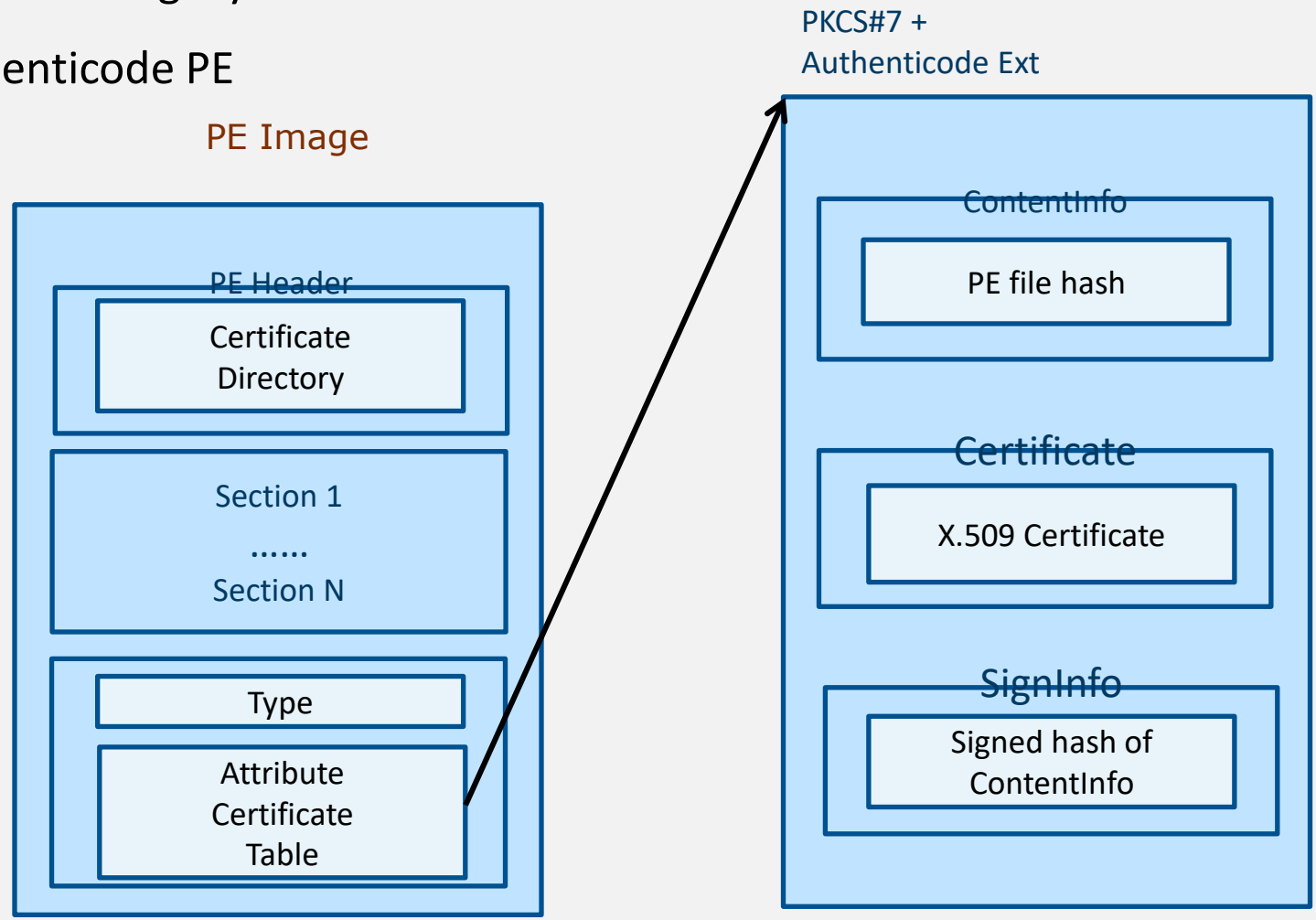
- PI & UEFI complement each other to impart **platform security** through guarding and verification during pre-boot.
- PI facilitates **platform hardening** by guarding internal firmware ingredients that consume reset vector, initialization of CPU, Memory, Chipset etc.
- UEFI signing allows **robust platform scaling** through verified inclusion of external firmware ingredients such as OPROMS into the trust chain



UEFI Driver Signing

Why? – Origin & Integrity

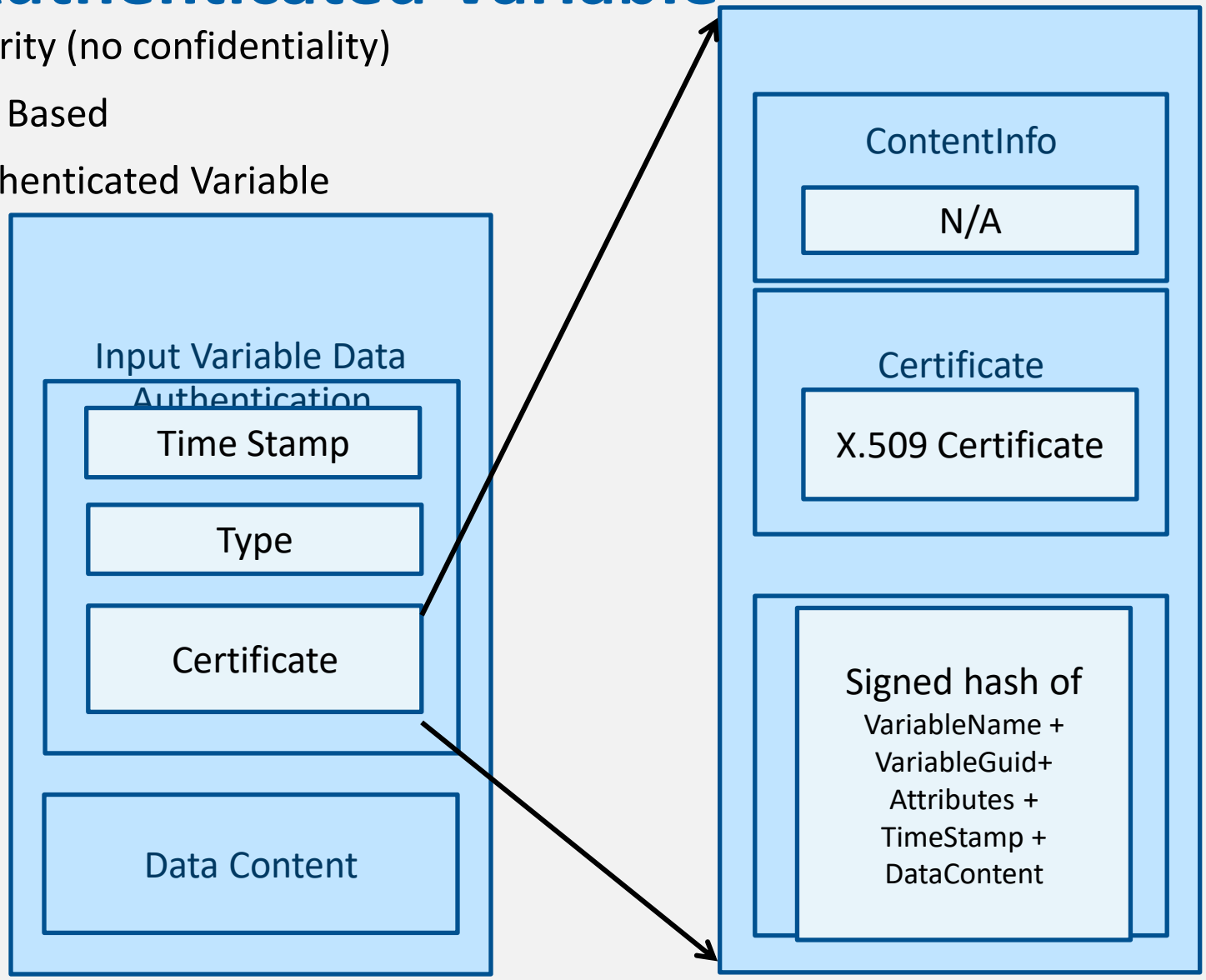
How? – Authenticode PE



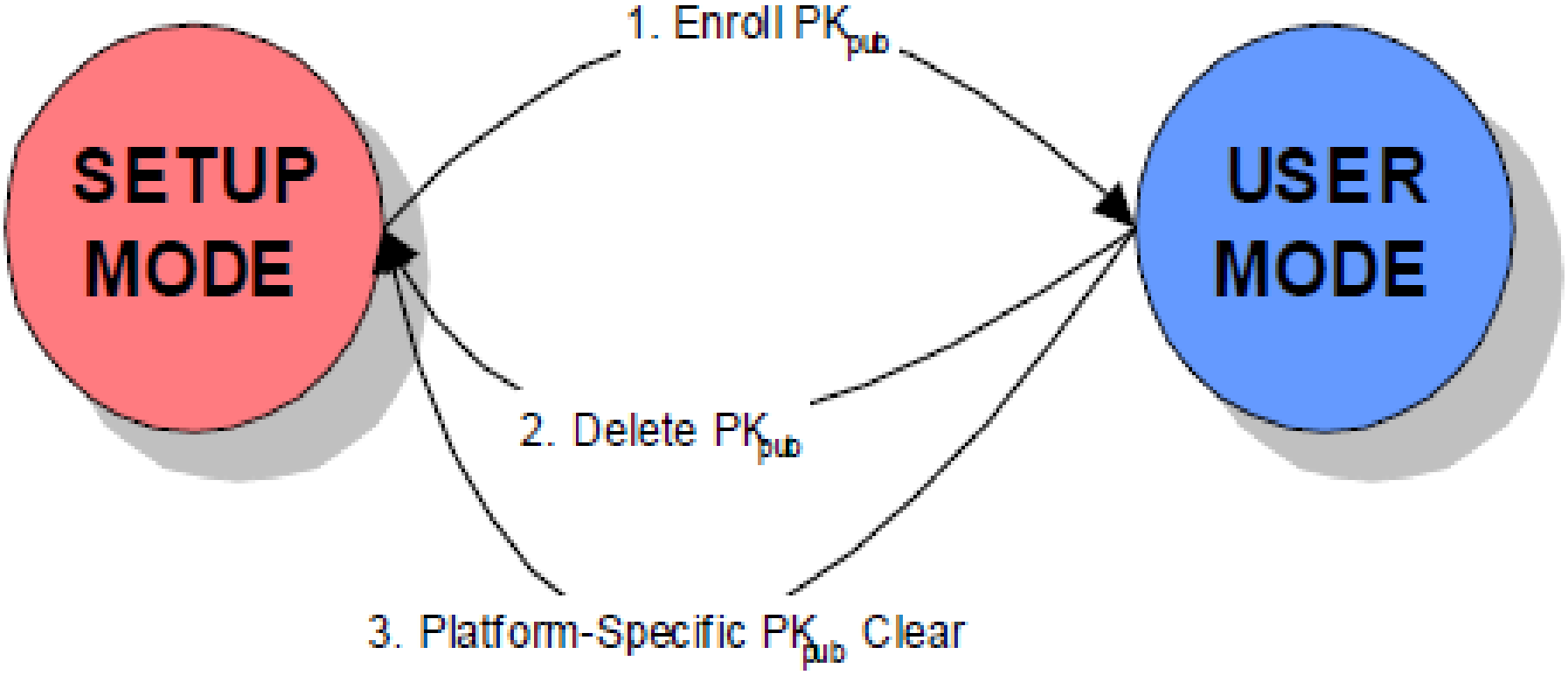
UEFI Authenticated Variable

Why? – Integrity (no confidentiality)

How? – Time Based



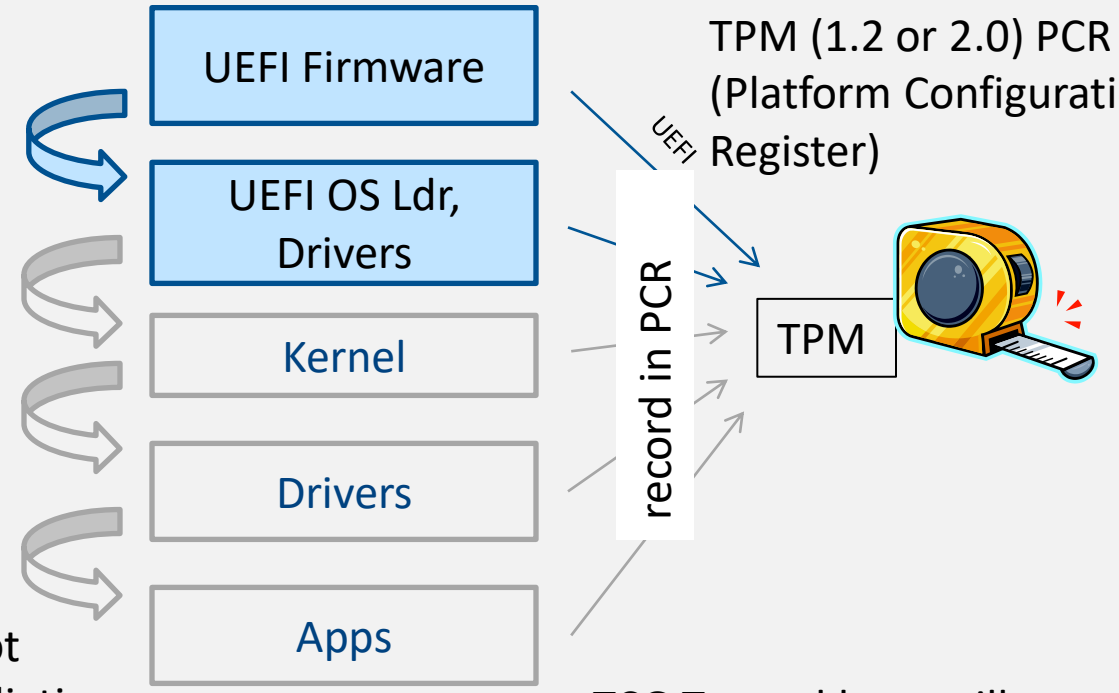
Put them altogether: UEFI Secure Boot



UEFI Secure Boot VS TCG Trusted Boot

UEFI authenticate OS loader
(pub key and policy)

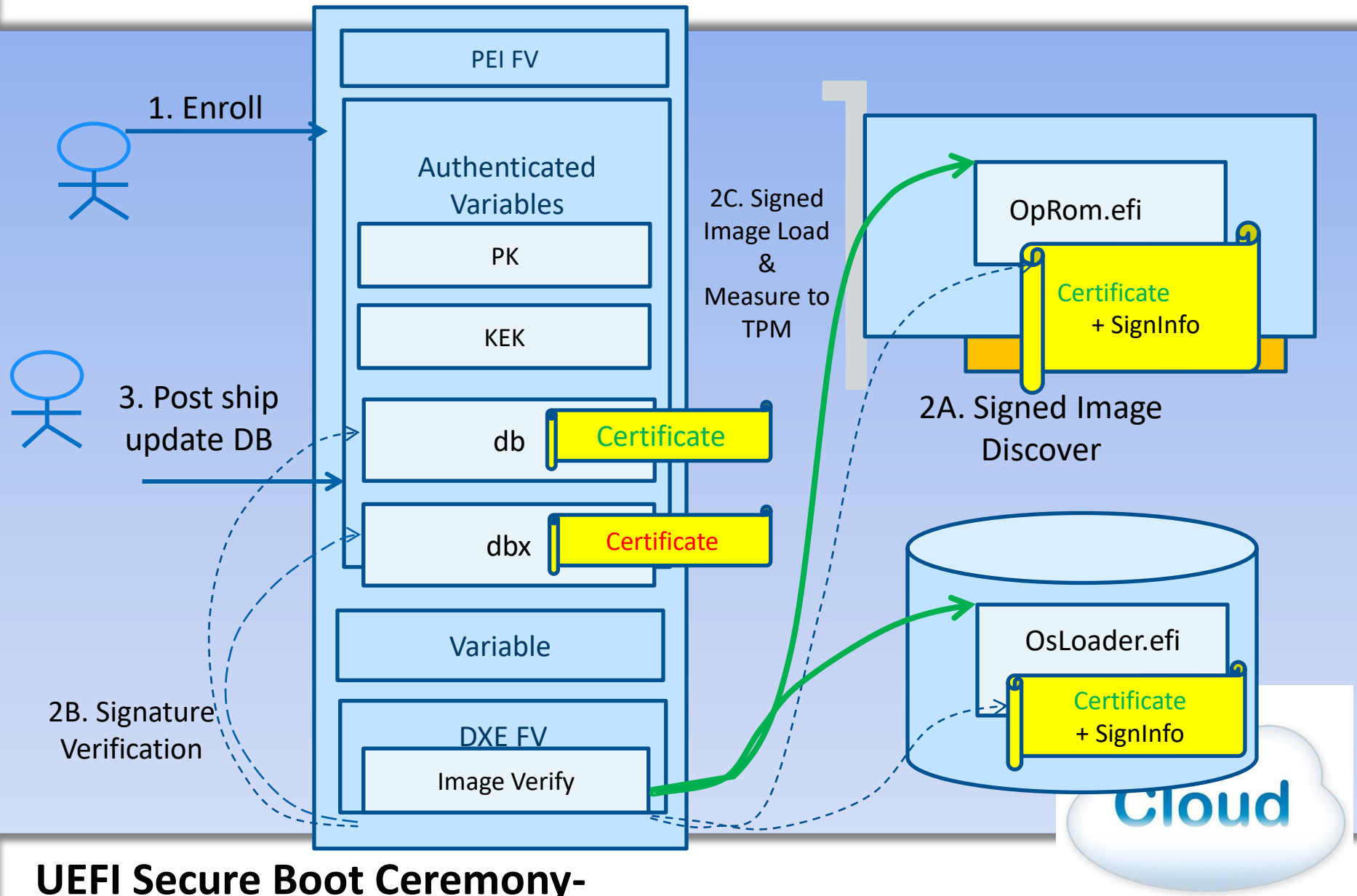
Check signature of before loading



UEFI PI will measure OS loader & UEFI drivers into TPM (1.2 or 2.0) PCR (Platform Configuration Register)

- UEFI Secure boot will stop platform boot if signature not valid (OEM to provide remediation capability)
- UEFI will require remediation mechanisms if boot fails

- TCG Trusted boot will never fail
- Incumbent upon other SW to make security decision using attestation



UEFI Secure Boot Ceremony-

Ellison, Carl M. "**Ceremony** Design and Analysis," IACR Cryptology ePrint Archive (2007): 399. https://eprint.iacr.org/2007/**399**.**pdf**

Relevant open source software packages/routines for Authorization flow

MdeModulePkg

LoadImage Boot Service

gBS->LoadImage
CoreLoadImage()

EFI_SECURITY_ARCH_PROTOCOL
SecurityStubDxe

SecurityStubAuthenticateState()

DxeSecurityManagementLib

RegisterSecurityHandler()
ExecuteSecurityHandlers()

SecurityPkg
DxeImageVerificationLib

DxeImageVerificationHandler()
HashPeImage()
HashPeImageByType()
VerifyWinCertificateForPkcsSignedData()
DxeImageVerificationLibImageRead()
IsSignatureFoundInDatabase()
IsPkcsSignedDataVerifiedBySignatureList()
VerifyCertPkcsSignedData()

Authenticated Variables
gRT->GetVariable

MdePkg
BasePeCoffLib

PeCoffLoaderGetImageInfo()

CryptoPkg
BaseCryptLib

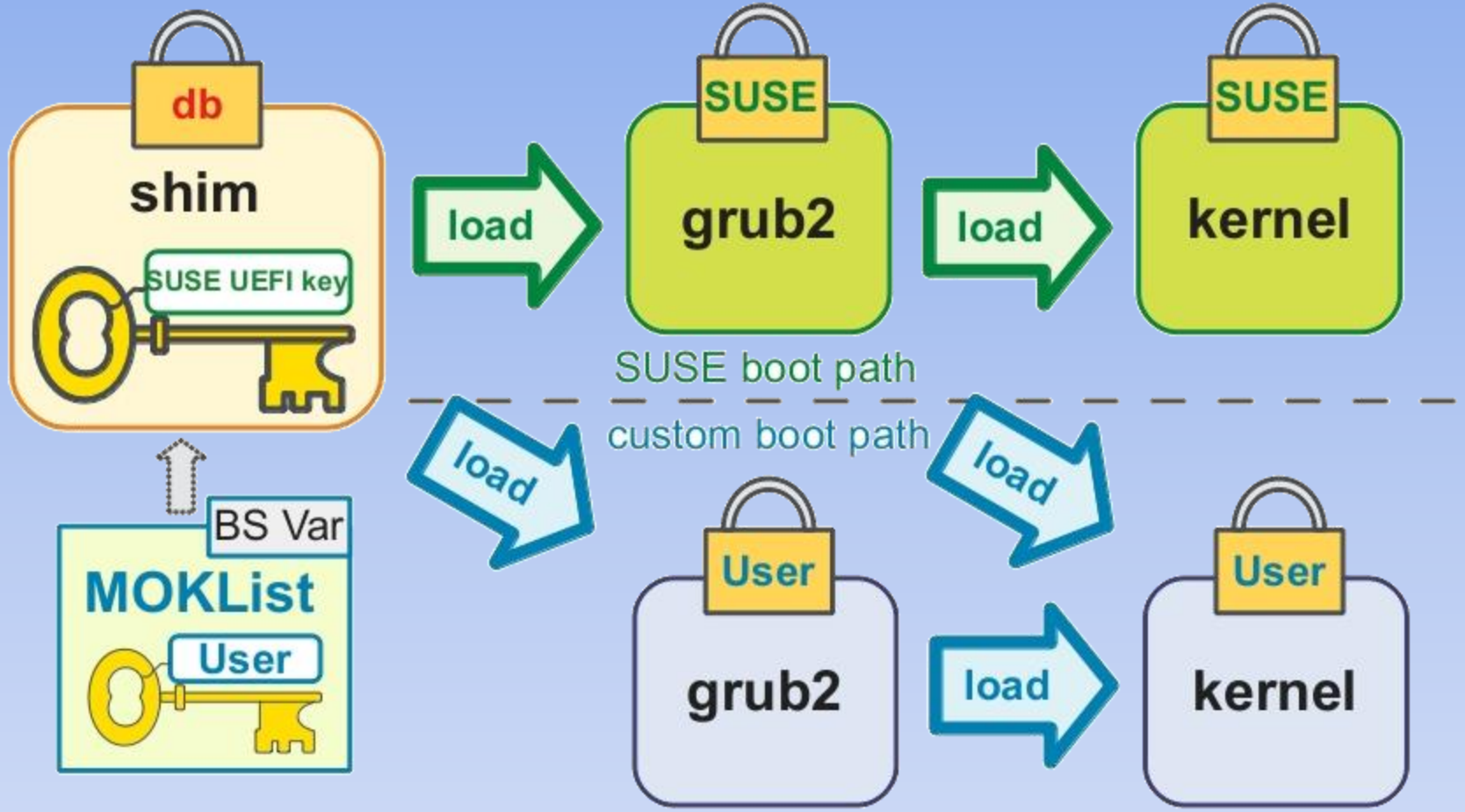
Sha256Init()
Sha256Update()
Sha256Final()
Sha256GetContextSize()

AuthenticodeVerify()
Pkcs7Verify()
WrapPkcs7Data()

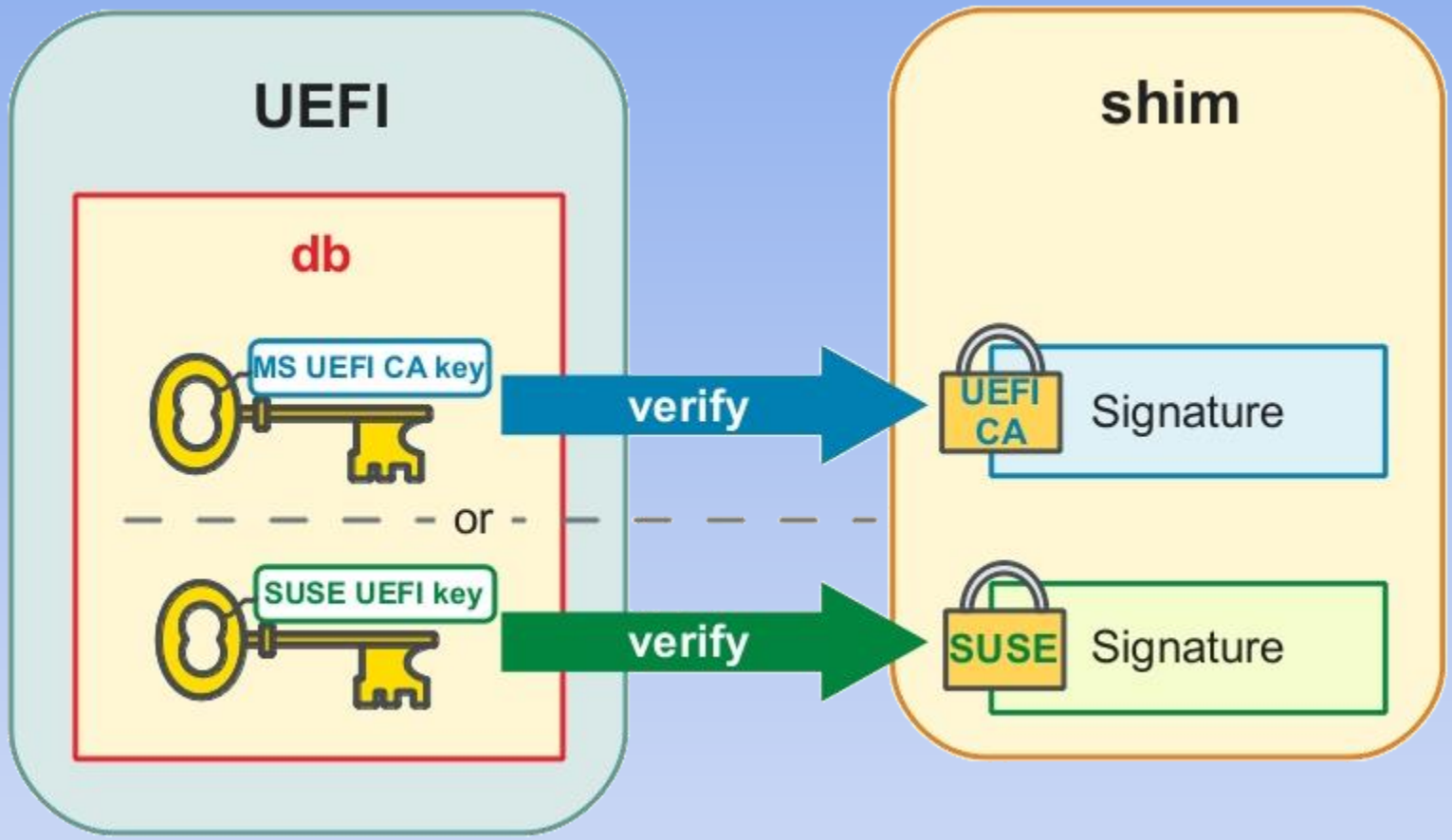
OpenSslLib
Openssl-0.9.8w

IntrinsicLib

See *Rosenbaum, Zimmer, "A Tour Beyond BIOS into UEFI Secure Boot,"* for more details



Load the UEFI image as long as it is trusted



Either the UEFI CA key or SUSE key will let the shim boot with UEFI secure boot

RandomNumberGenerator

UEFI driver implementing the EFI_RNG_PROTOCOL from the UEFI2.4 specification

TCG

PEI Modules & DXE drivers implementing Trusted Computing Group measured boot

EFI_TCG_PROTOCOL and EFI_TREE_PROTOCOL from the TCG and Microsoft MSDN websites, respectively

UserIdentification

DXE drivers that support multi-factor user authentication

Chapter 31 of the UEFI 2.4 specification

Library

DxeVerificationLib for “UEFI Secure Boot”, chapter 27.2 of the UEFI 2.4 specification + other support libs

VariableAuthenticated

SMM and runtime DXE authenticated variable driver, chapter 7 of the UEFI2.4 specification

https://svn.code.sf.net/p/edk2/code/trunk/edk2/SecurityPkg

Variable Lock Protocol

Make variables read-only

https://github.com/tianocore/edk2/blob/master/MdeModulePkg/Include/Protocol/VariableLock.h

Lock Box

Protect content across re-starts

https://github.com/tianocore/edk2-MdeModulePkg/blob/master/Include/Protocol/LockBox.h

Capsule Update

Generic capsule update driver support

http://comments.gmane.org/gmane.comp.bios.tianocore.devel/8402

https://svn.code.sf.net/p/edk2/code/trunk/edk2/

Analyze and Mark external Interfaces where input can be attacker controlled data, comment headers

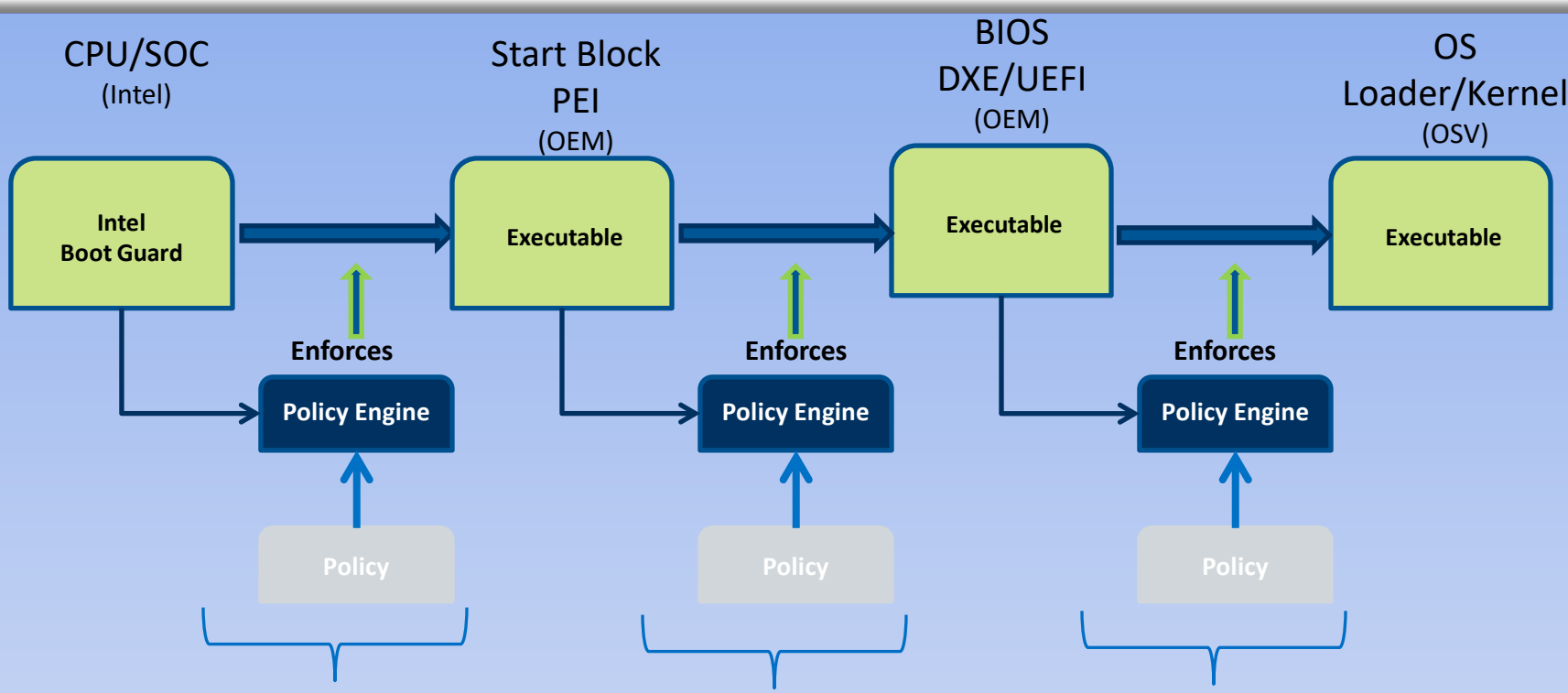
```
/**
  Install child handles if the Handle supports GPT partition structure.


  Caution: This function may receive untrusted input.
  The GPT partition table is external input, so this routine
  will do basic validation for GPT partition table before install
  child handle for each GPT partition.


  @param[in]  This       Calling context.
  @param[in]  Handle     Parent Handle.
  @param[in]  DevicePath Parent Device Path.


**/
EFI_STATUS
PartitionInstallGptChildHandl
```

UDK2010 example: http://edk2.svn.sourceforge.net/svnroot/edk2/trunk/edk2/MdeModulePkg/Universal/Disk/PartitionDxe/Gpt.c



Intel® Device Protection Technology with Boot Guard

http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/4th-gen-core-family-mobile-brief.pdf

OEM PI Verification Using PI Signed Firmware Volumes

Vol 3, section 3.2.1.1 of PI 1.3 Specification

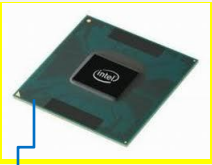
OEM UEFI 2.4 Secure Boot

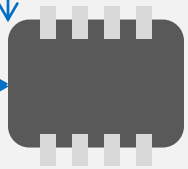
Chapter 27.2 of The UEFI 2.4 Specification

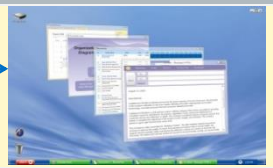
Technologies – putting it together

Reset

TCG Measurements into PCRs 0..7







Assets	Threats	
BIOS Flash	ROM Swap Bit rot	H/W Spec
System BIOS NIST SP800-147. PEI recovery. DXE SMM, UEFI Core	Erase flash part Overwrite flash part	SP800 -147
Option ROMs BIOS device drivers	Erase op ROM Overwrite op ROM	UEFI 2.4
Network Boot IPv6 for the cloud	Network attacks	
OS Boot loader BIOS loads the OS To BIOS, VM is an OS	Spoof boot loader	

Different colors for different vendors

Trust Model

System

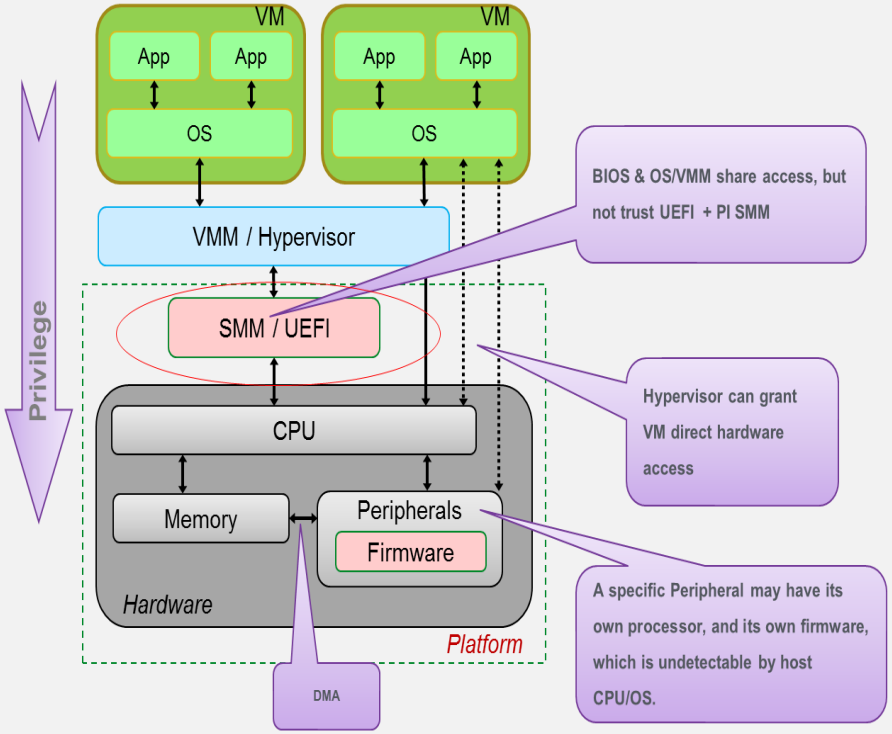
Data	CP/M (1974)	IBM PC (1981)	PC (1999)	PC (2012)
BIOS ROM	<4K	40K	512K	4M
Processor	8080	8088	Pentium III	Ivy Bridge
OS	CP/M	DOS	Win98	Win8

Security

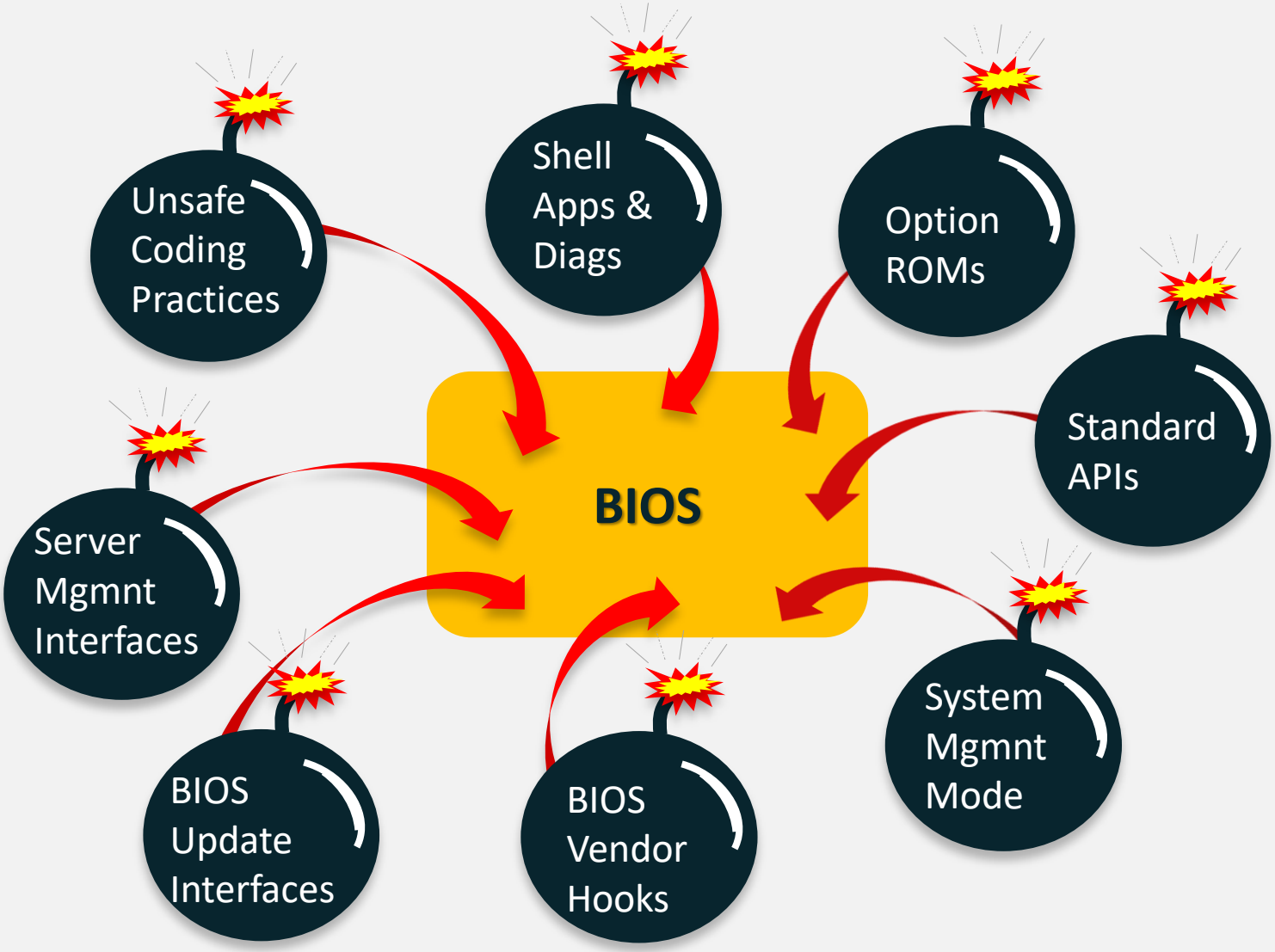
Data	CP/M (1974)	IBM PC (1981)	PC (1999)	PC (2012)
Malware	**[1949]**: **John von Neumann** - Theory of **self-reproducing** automata **[1971]**: Bob Tomas: Creeper - **"I'm Creeper: Catch me If you Can"**	**[1984]**: **Fred Cohen**- Computer **Viruses** - Theory and Experiments (VAX, UNIVAC) **[1986]**: Farooq Alvi – **Brian** (IBM PC)	**[1999]**: **CIH** (Intel 430TX)	**[2011]**: **BMW** /Mebromi (Award BIOS) *[2005~] BootKit *[2006~] BiosRootkit *[2006~] SMM *[2008] Password *[2008~] NIC *[2009] BMP *[2009] ME *[2009~] KBC/EC *[2011] Battery Bootkit S3 SMM
Security	**[1972]** Anderson - Computer Security Technology Planning **[1973]** Bell-LaPadula **[1977]** Biba Integrity [1989] Clark Wilson	Bios Password	Flash Protection **[1992] SMM** in i486SL	**[2003]** TCG **[2006]** TXT **[2006]** UEFI Secure Boot **[2009] S**MRR **[2014]** Intel ® Boot and BIOS Guard

Security Challenges

- Different elements in platform from many vendors
- How to establish trust anchor in the hardware
- How to protect elements
- How to protect the platform
- How to allow platform scaling



BIOS Potential Attack Surfaces



BIOS Malware

UEFI Rootkits

Bootkits

SMM Rootkits

Device FW Malware

ACPI Rootkits

Option ROM Malware

Evil Maid

HVM Rootkits (Blue Pill)

HW Trojans

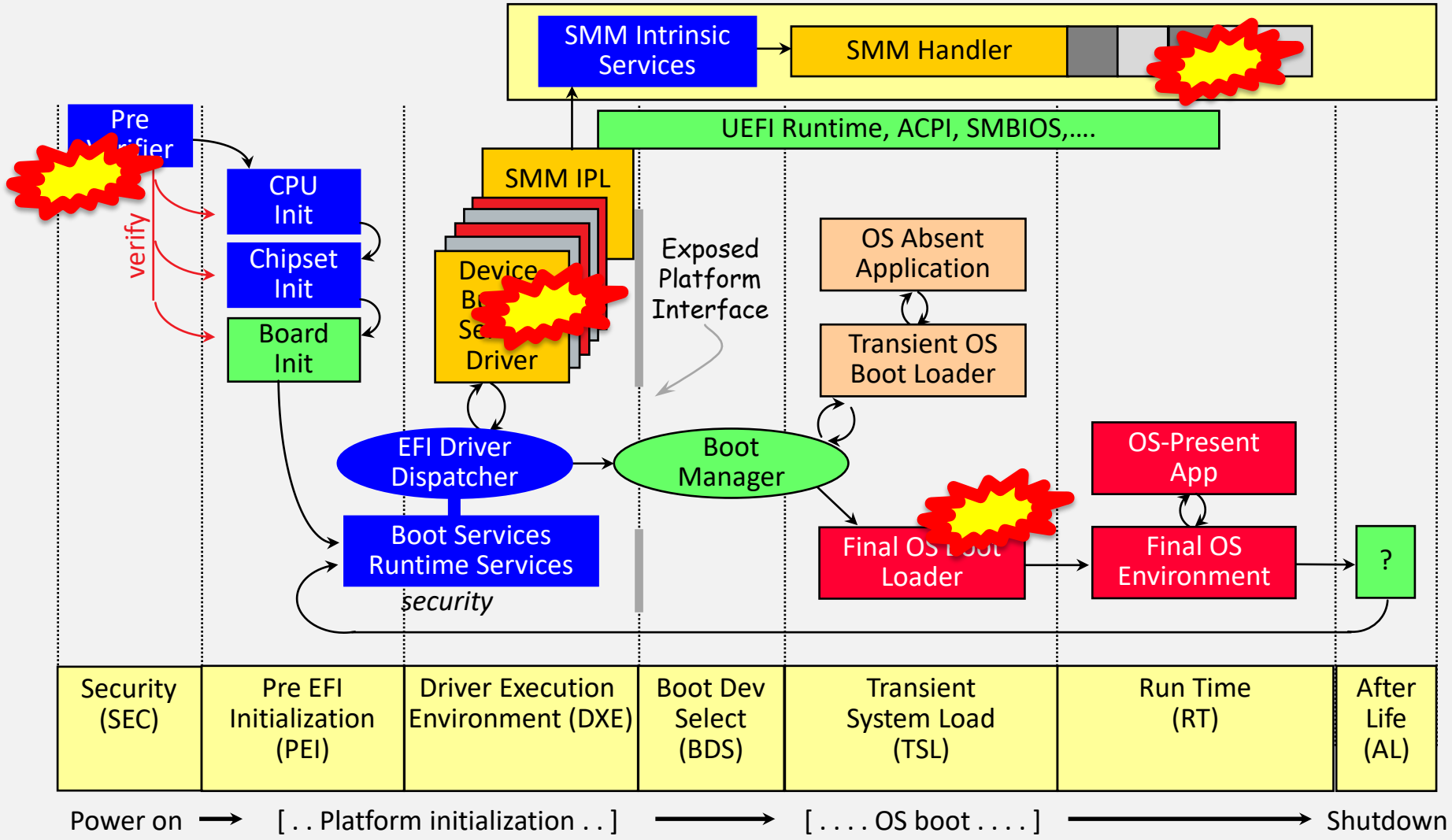
Missed a Detail

Source: Jeff Forristal

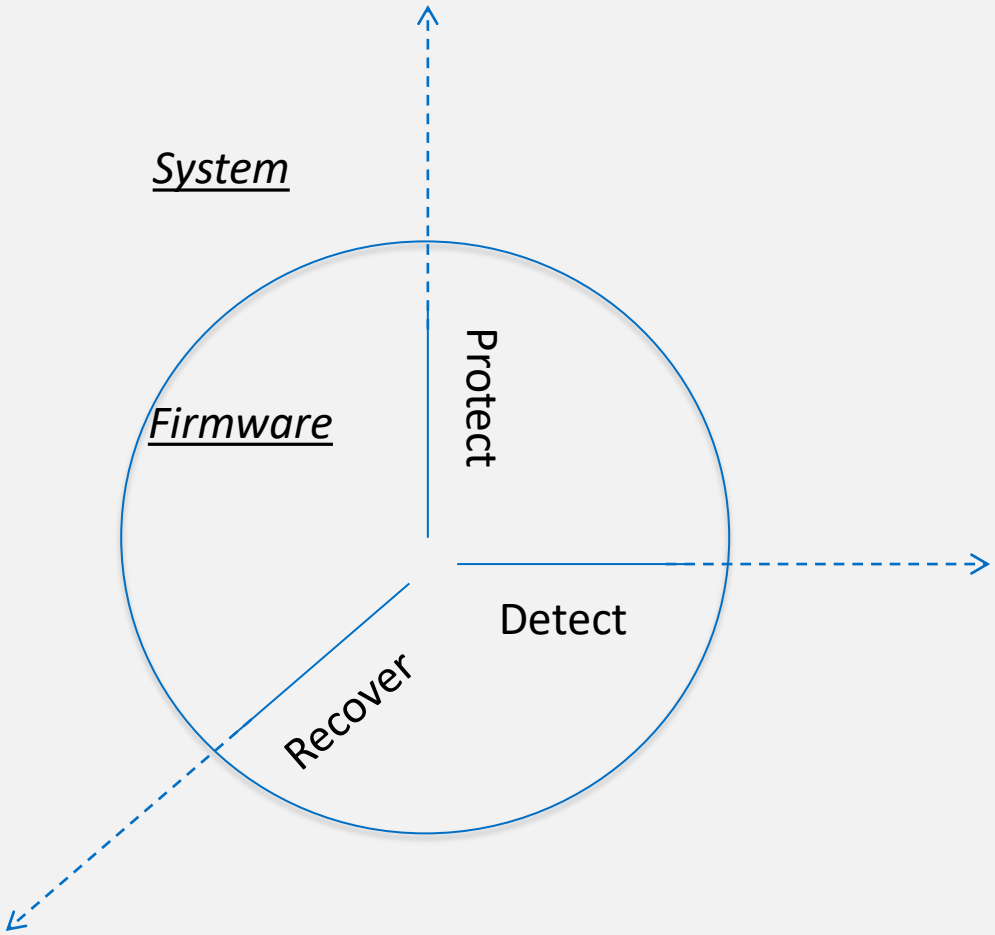
Security Fundamentals

Security Fundamentals

What Could Possibly Go Wrong???



Things need consider



What to build & defend – Rationale for a threat model

"My house is secure" is almost meaningless

- Against a burglar? Against a meteor strike? A thermonuclear device?

"My system is secure" is almost meaningless

- Against what? To what extent?

Threat modeling is a process to define the goals and constraints of a (software) security solution

- Translate user requirements to security requirements

We used threat modeling for our UEFI / PI codebase

- We believe the process and findings are applicable to driver implementations as well as UEFI implementations in general

Defining, using a threat model

A Threat Model (TM) defines the security assertions and constraints for a product

- Assets: What we're protecting
- Threats: What we're protecting it against
- Mitigations: How we're protecting our Assets

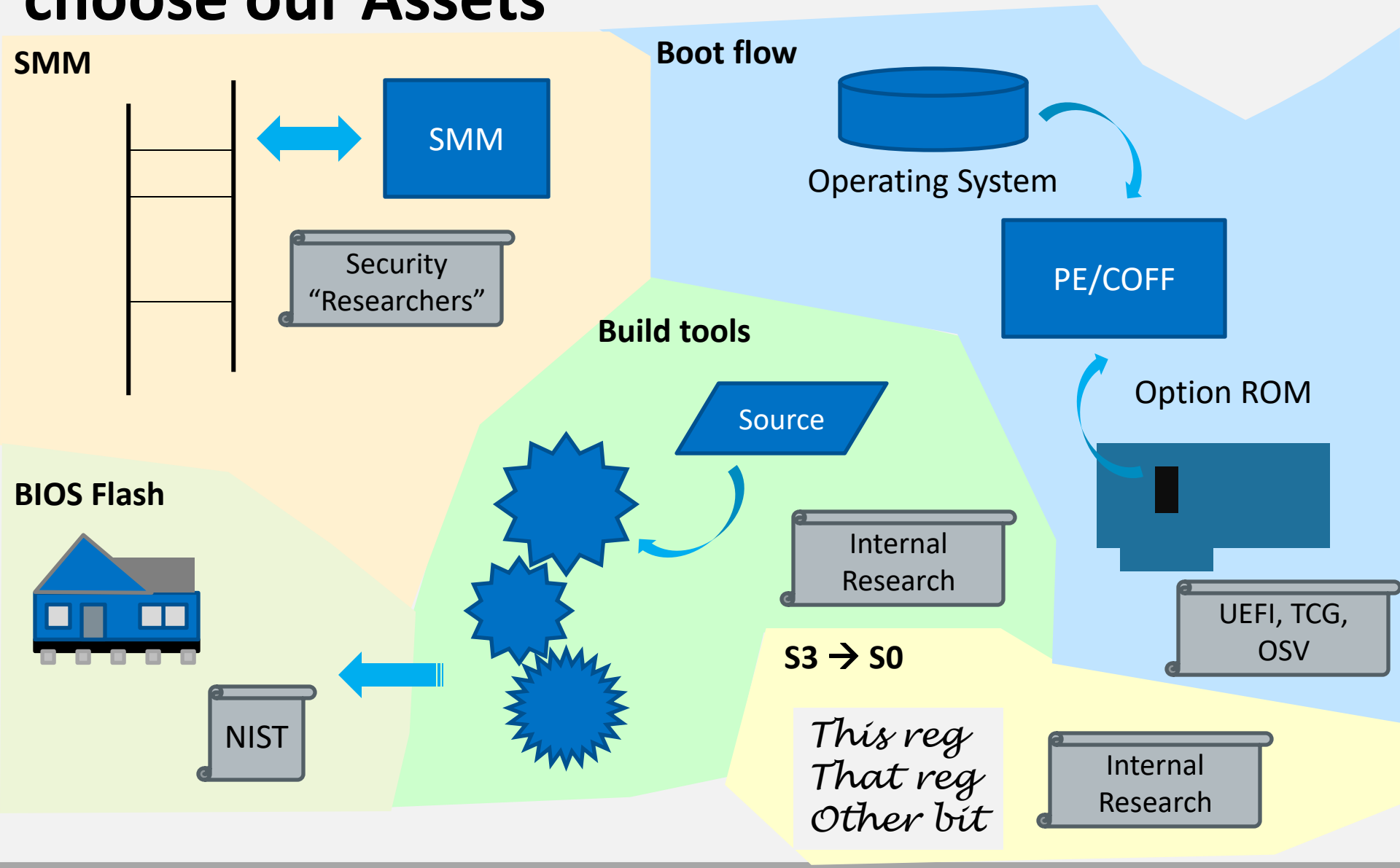
Use TM to narrow subsequent mitigation efforts

- Don't secure review, fuzz test all interfaces
- Select the ones that are critical

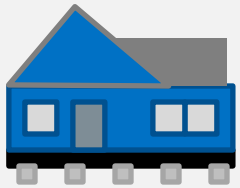
TM is part science, part art, part experience, part nuance, part preference

- Few big assets vs lots of focused assets

We don’t always get to choose our Assets



Flash**



NIST SP800-147 says

- Lock code flash except for update before Exit Mfg Auth
- Signed update (>= RSA2048, SHA256)
- High quality signing servers
- Without back doors (“non-bypassability”)

Threats

- PDOS – Permanent Denial of Service
  - System into inefficient room heater
- Elevation of privilege
  - Owning the system at boot is an advantage to a virus

Known attacks

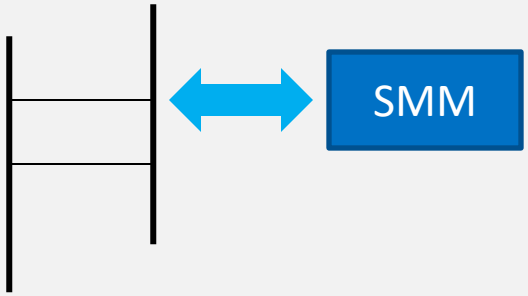
- CIH / Chernobyl 1999-2000
- Mebroni 2010

Mitigations include

- Reexamining flash protection methods – use the best even if its new
- Using advanced techniques to locate and remove (un)intentional backdoors

** or tomorrow’s equivalent NV storage

SMM



SMM is valuable because

- It's invisible to Anti Virus, etc
- SMM sees all of system RAM
- Not too different from PCI adapter device firmware

Threats

- Elevation
  - View secrets or own the system by subverting RAM

Known attacks

- See e.g Duflot, Legbacore

Mitigations include

- Validate "external" / "untrusted" input
- Remove calls from inside SMM to outside SMM

Resume from S3

This reg
That reg
Other bit

ACPI says that we return the system to the S5→S0 configuration at S3→S0

- Must protect the data structures we record the cold boot config in

Threats

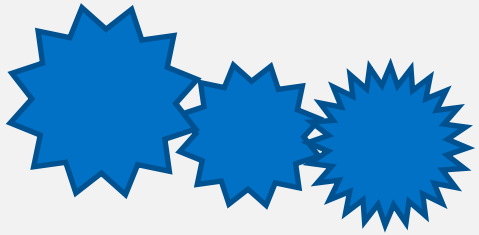
- Changing data structures could cause security settings to be incorrectly configured leaving S3
- Reopen the other assets' mitigated threats

Known attacks

Mitigations include

- Store data in SMM -or-
- Store hash of data structures and refuse to resume if the hashes don't compare

Tool chain



Tools create the resulting firmware

- Rely on third party tools and home grown tools
- Incorrect or attacked tools leave vulnerabilities

Threats

- Disabled signing, for example

Known attacks

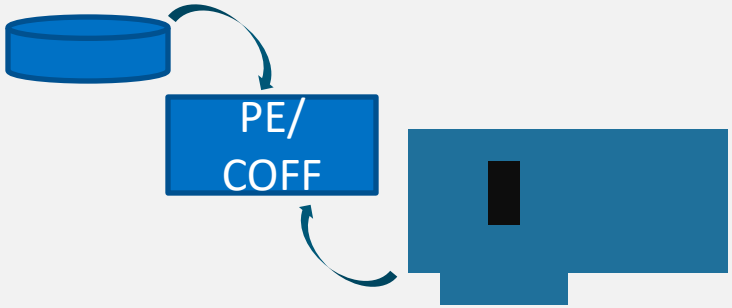
- See e.g. *Reflections on Trust*, Ken Thompson**

Mitigation

- Difficult: For most tools, provided as source code
- Review for correct implementation
- Use static, dynamic code analysis tools
  - PyLint for Python, for example

** CACM, Vol 27, No 8, Aug, 1984, pp. 761-763

Boot flow



Secure boot

- Authenticated variables
- Based on the fundamental Crypto being correct
- Correct location for config data

Threats

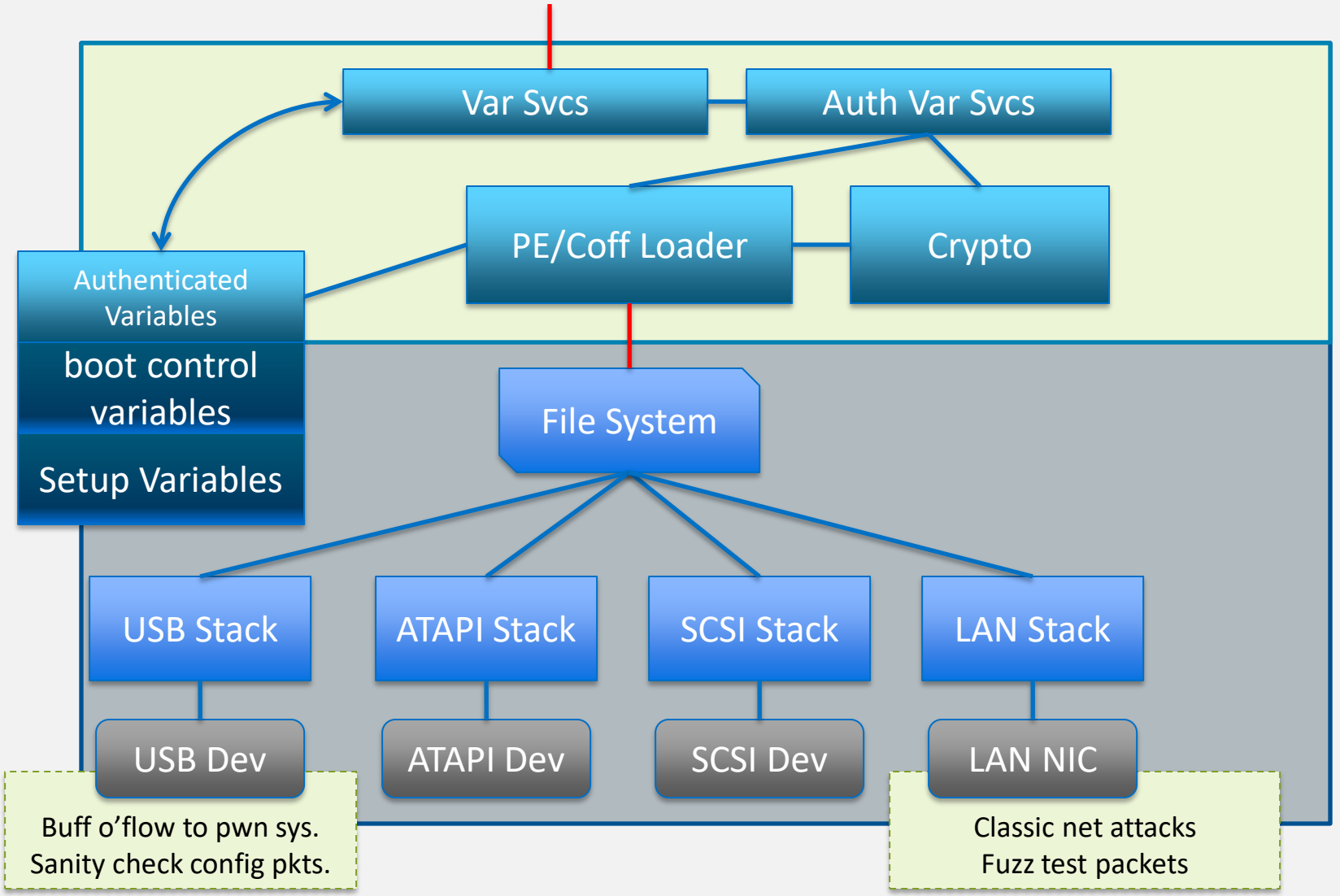
- Run unauthorized op roms, boot loaders
- PDOS systems with bad config variables

Known attacks

Mitigations include

- Sanity check config vars before use, use defaults
- Reviews, fuzz checking, third party reviews, etc.

TM to Modules: Boot flow



Assets or not?

Variable content sanity checking?

- If you randomly fill in your Setup variables, will your system still boot?
- Fit in as a part of boot flow

ACPI? We create it but don’t protect it

TPM support? We fill in the PCRs but don’t use them (today)

Quality ≠ Security

Vulnerability VS Threat

Vulnerability Cases:

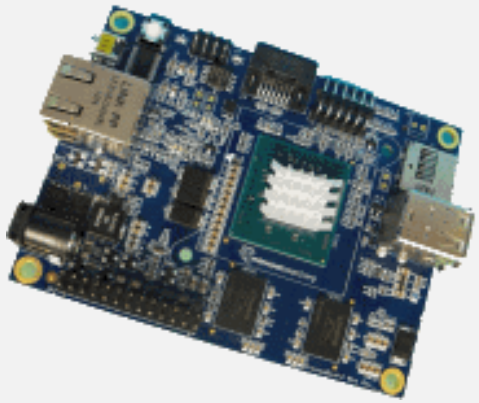
- Unauthorized Firmware Update
- Unauthorized 3rd Party Code
- Critical Register Unlocked
- Buffer Overflow
- Secret Used but not Cleared
- Default Passphrase to Access

Threat:

- S: Spoof user identity
- T: Tampering
- R: Repudiation
- I: Information disclosure
- P: Permanent Denial of Service
- E: Elevation of privilege
- D: Denial of service

EDK II on MinnowMax

MinnowMax



Open hardware platform

Baytrail single or dual core

From http://firmware.intel.com/projects

This project focus in on the firmware source code (and binary modules) requried to create the boot firmware image for the MinnowBoard MAX. The UEFI Open Source (EDKII project) packages for MinnowBoard MAX are available at http://tianocore.sourceforge.net/wiki/EDK2. To learn more about getting involved in the UEFI EDKII project visit the How to Contribute page.

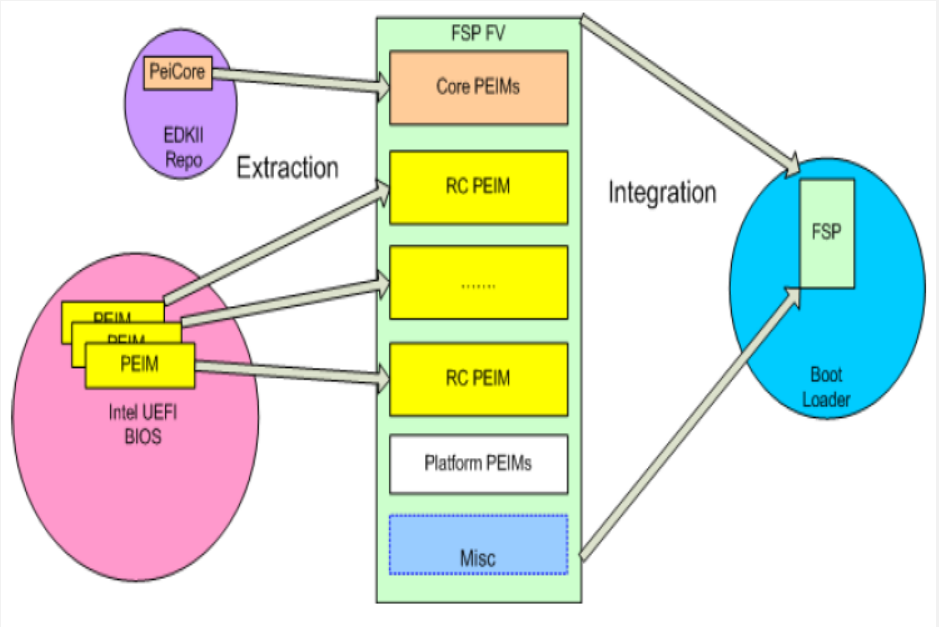
The source code builds using Microsoft Visual Studios and GNU C Compiler (for both 32 and 64 bit images) - production and debug execution environments. The source code builds the same UEFI firmware image shipping on MinnowBoard MAX.

- See more at: http://firmware.intel.com/projects#sthash.1oOc8srY.dpuf

Intel® Firmware Support Package (FSP) Overview

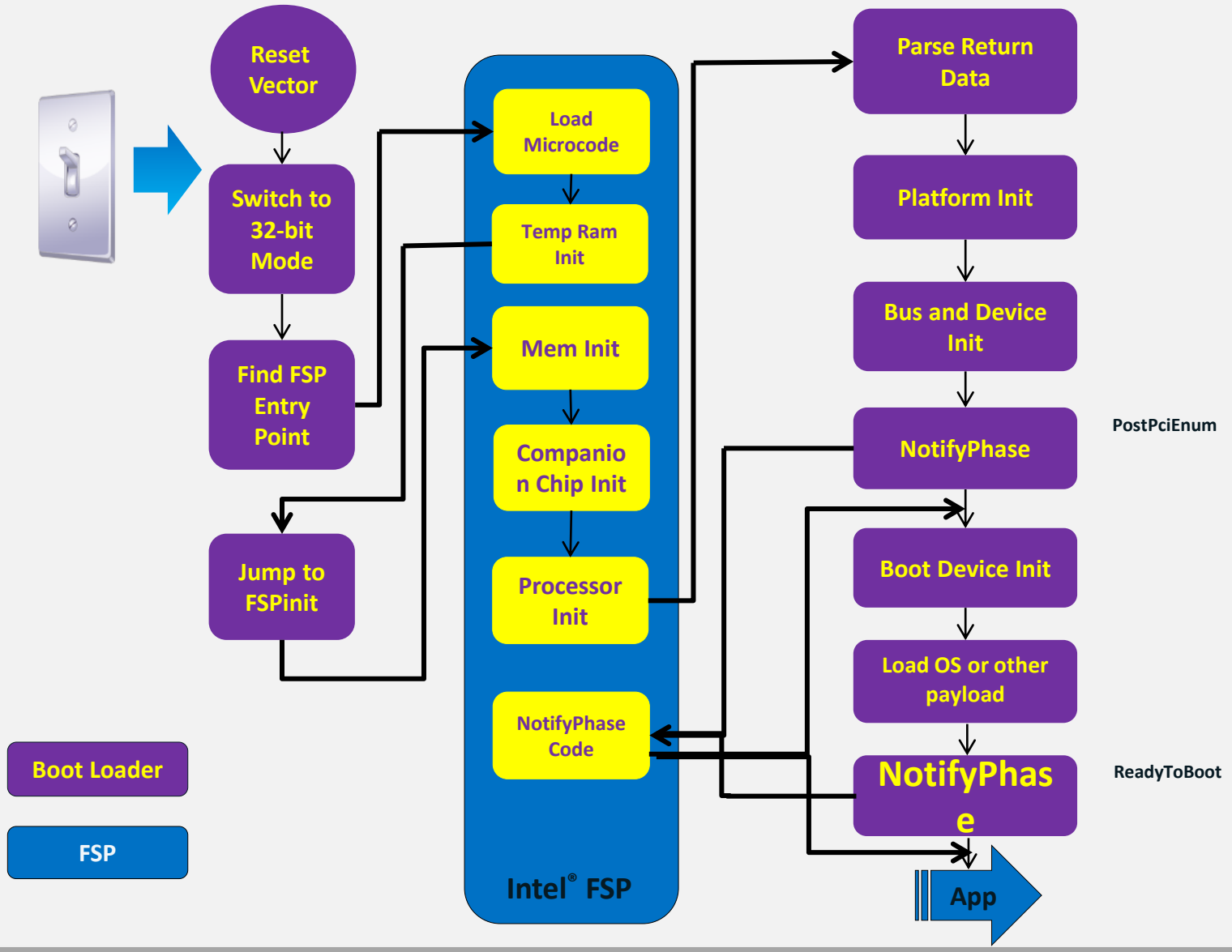
The Intel ® FSP provides processor & chipset initialization in a format that can easily be incorporated into many existing boot loader frameworks without exposing the Intellectual Property (IP) of Intel.

- Distributed as single binary
- Silicon PEIMs packaged into FSP
- Plugs into existing f/w frameworks
- Binary customization



More information at www.intel.com/fsp

Intel® FSP Boot Flow



MinnowMax

Focused on the maker community, but….

64-bit Intel® Atom™ E38xx Series SOC

Has UEFI Secure Boot

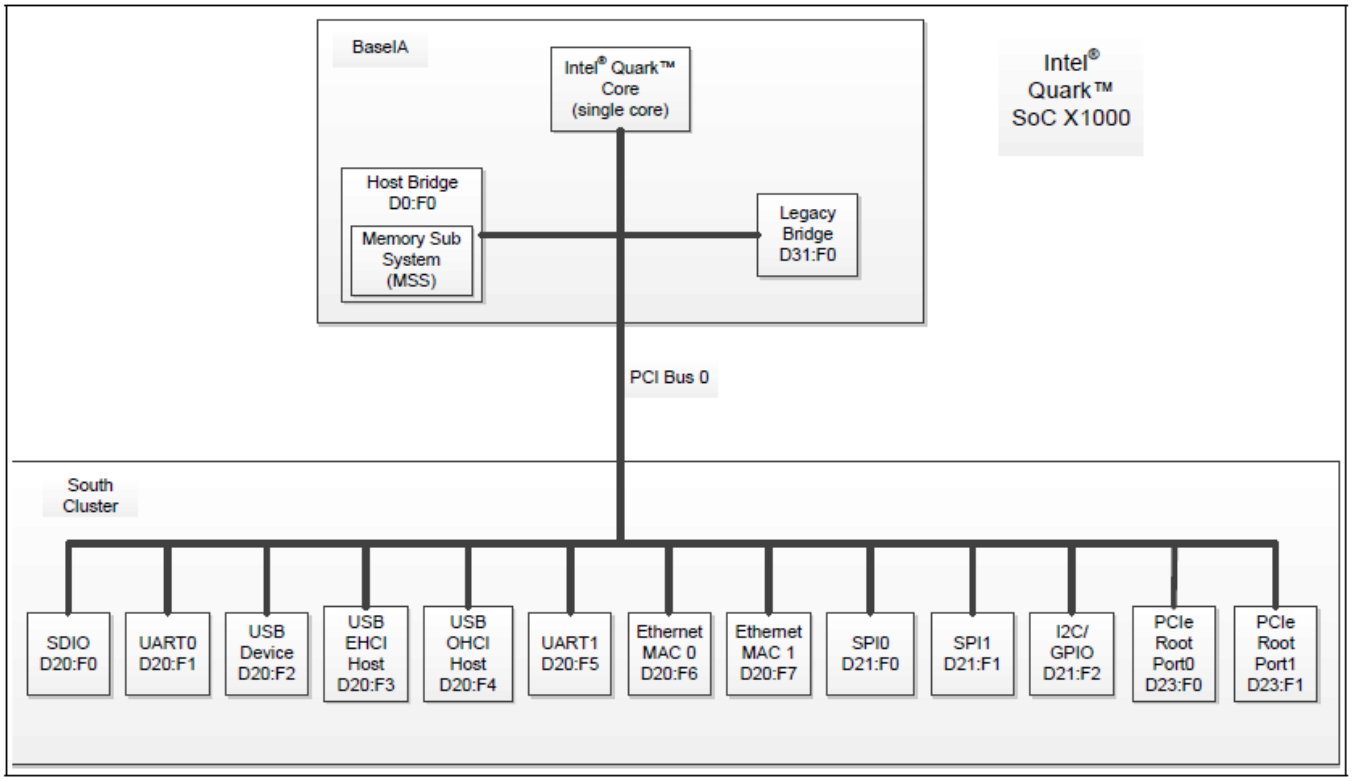
Built off of live tree

Ability to update w/ latest capabilities on http://www.tianocore.org

EDK II on Galileo

Intel® Quark™ SoC – Hardware Overview

- 32 bit Intel® Pentium® ISA-class processor
- PCI
- USB
- I2C
- Single core



UEFI for Intel® Quark™ SoC

First fully open source Intel-based platform

Builds on Intel® UDK2014 packages like MdePkg, MdeModulePkg w/ a 32-bit build, adding

- IA32FamilyCpuBasePkg
- QuarkPlatformPkg
- QuarkSocPkg

Standard build is 1 Mbyte image w/full features

- Capsule update, SMM, S3, PCI, recovery, full UEFI OS support, FAT OS support, UEFI variables

Quark and security

Support for I2C-attached TPM

Hardware Secure Boot option

UEFI Secure Boot implementation

UEFI Capsule update support w/ hardware verification assist

…..

Demonstrates one way to build out UEFI Security Features w/ a full open source platform tree

Futures

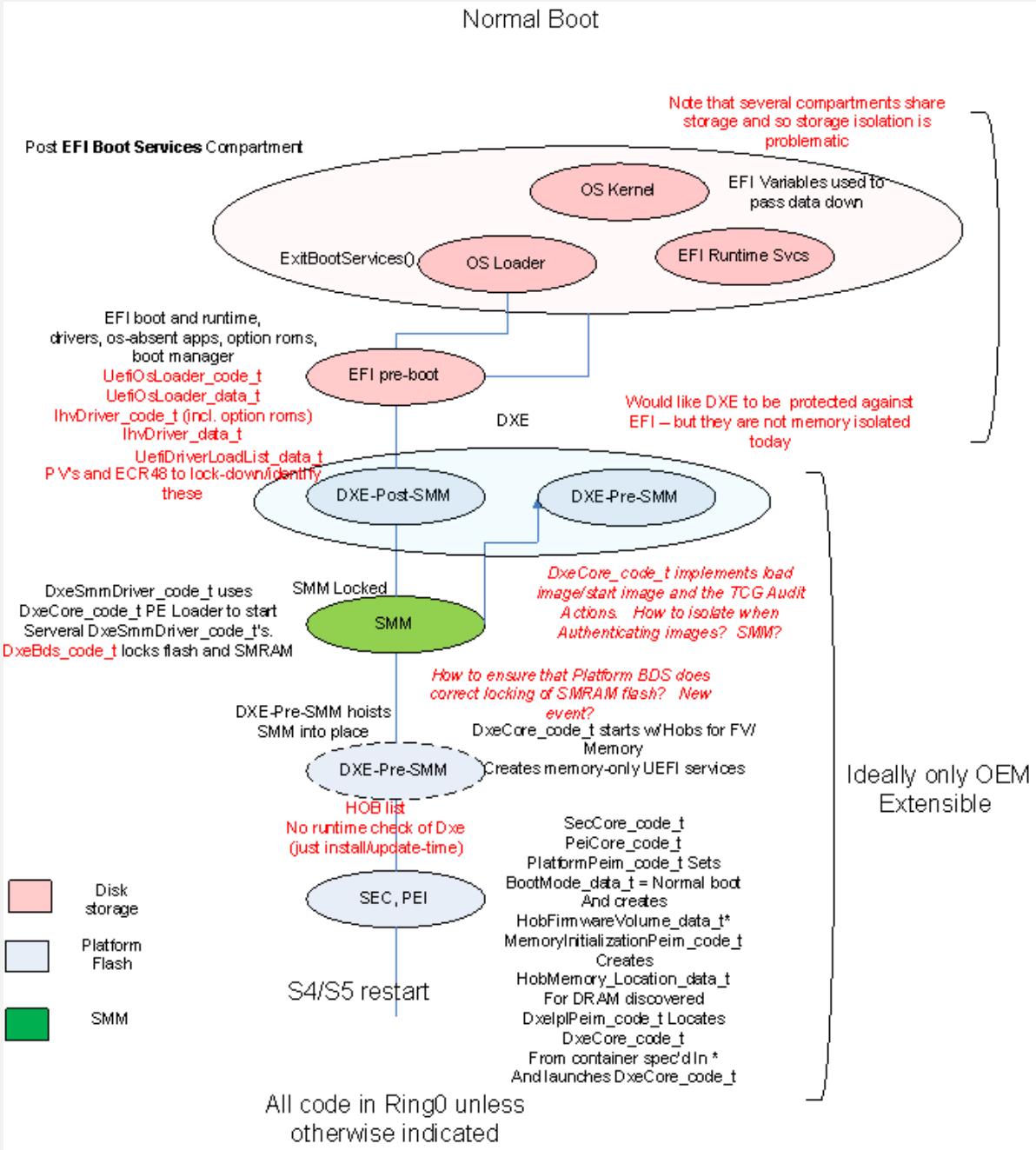
Integrity analysis of Pre-OS via Compartments – CW/BIBA

Business goals dictate isolation boundaries called compartments (cpts)

- Platform Manufacturer wants to protect self from 3rd party pre-OS and OS/Hv
- OS/Hv- protect self from pre-boot extensibility
- Idealized isolation boundary is a compartment (cpt)–
  - Not a thing like process, interface table, handle etc
- Security types are actual data types that are used in the cpt.
  - Eg: Smm_code_t, efi_iface_tbl_data_t in OEM cpt

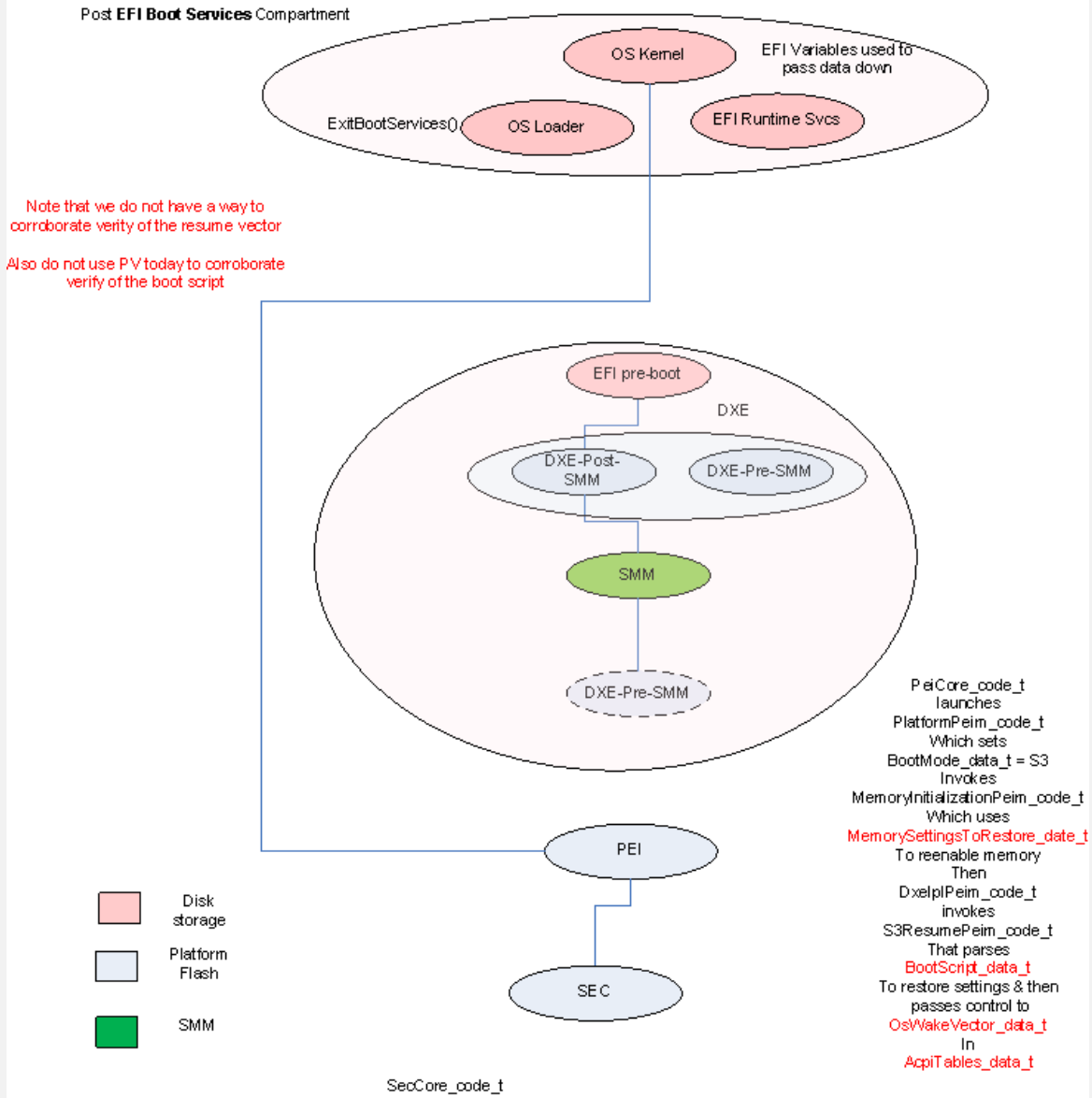
Security analysis checks if something is a compartment

- Checking a number of information flow rules
  - Code and data in compartment have verifiable integrity
    - Compartment needs storage isolated from other compartments
    - Compartment needs execution isolation
  - Compartment transitions have chokepoints and are protected via guards
    - Interfaces into the compartment for code and data must be validated
  - Admin model of compartment must be fully specified
    - Admin tasks must be “minimized” to reduce TCO & chance of bugs
  - Audit (e.g.,TPM measurements)– All integrity affecting operations must be audited
    - Availability of audit log is also a requirement


Normal Boot
Note that several compartments share storage and so storage isolation is problematic
Post EFI Boot Services Compartment
OS Kernel
EFI Variables used to pass data down
ExitBootServices()
OS Loader
EFI Runtime Svcs
EFI boot and runtime, drivers, os-absent apps, option roms, boot manager
UefiOsLoader_code_t
UefiOsLoader_data_t
IhvDriver_code_t (incl. option roms)
IhvDriver_data_t
UefiDriverLoadList_data_t
PV's and ECR48 to lock-down/identify these
EFI pre-boot
DXE
Would like DXE to be protected against EFI – but they are not memory isolated today
DXE-Post-SMM
DXE-Pre-SMM
DxeSmmDriver_code_t uses DxeCore_code_t PE Loader to start Serveral DxeSmmDriver_code_t's.
DxeBds_code_t locks flash and SMRAM
SMM Locked
SMM
DxeCore_code_t implements load image/start image and the TCG Audit Actions. How to isolate when Authenticating images? SMM?
How to ensure that Platform BDS does correct locking of SMRAM flash? New event?
DXE-Pre-SMM hoists SMM into place
DxeCore_code_t starts w/Hobs for FV/ Memory
DXE-Pre-SMM
Creates memory-only UEFI services
HOB list
No runtime check of Dxe (just install/update-time)
SecCore_code_t
PeiCore_code_t
PlatformPeim_code_t Sets BootMode_data_t = Normal boot
And creates
HobFirmwareVolume_data_t*
MemoryInitializationPeim_code_t
Creates
HobMemory_Location_data_t
For DRAM discovered
DxeIplPeim_code_t Locates
DxeCore_code_t
From container spec'd In *
And launches DxeCore_code_t
SEC, PEI
Disk storage
Platform Flash
SMM
S4/S5 restart
Ideally only OEM Extensible
All code in Ring0 unless otherwise indicated

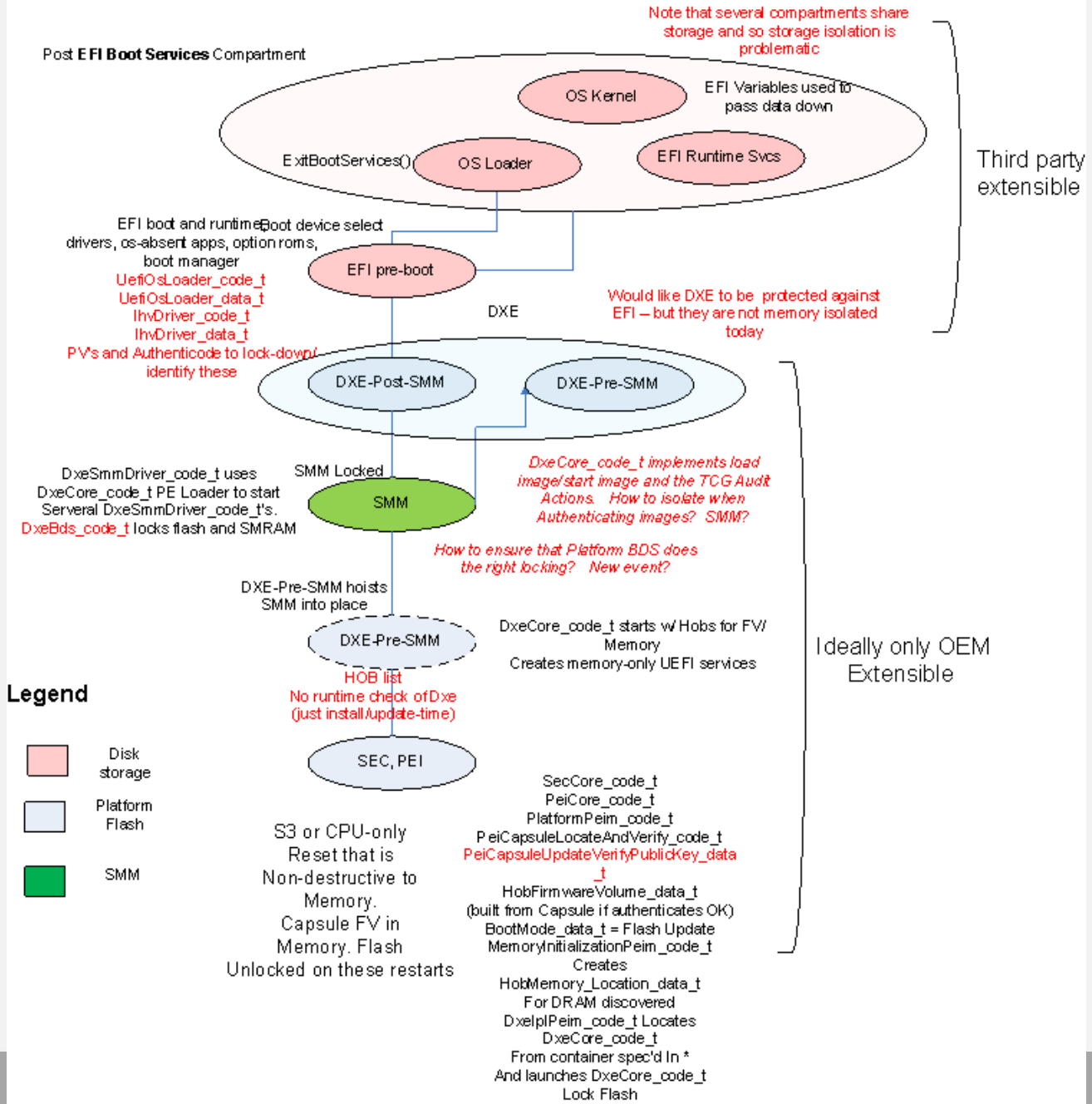
Normal boot

S3 Wake Event



S3 wake event

Flash update for system upgrade



Moving Forward

More open source examples

Test tools complement the SCT, but **the community can do more!**

Continue to evolve development philosophy

- "Testing shows the presence, not the absence of bugs" (*Dijkstra,1970)*
- Better Living Through Tools? (*Zimmer, 2013*)

Getting code coverage closer to 100%?

- Early effort using DDT with EDK II, Moving to KLEE (open source)

More fuzzers, from custom to public (e.g., Peach)

More Isolation – lots of hardware, esp. that built for OS/Hv. Leverage for platform firmware where it makes sense

- Map the CW/BIBA CPT analysis to specs & code
- Information flow and analysis
- NX, ALSR, Stack Canaries

References

1. UEFI Forum http://www.uefi.org
2. EFI Developer Kit II http://www.tianocore.org
3. White papers, training, projects http://firmware.intel.com
4. CHIPSEC: https://github.com/chipsec/chipsec
5. Trianocore security advisories
6. UEFI Forum USRT (security@uefi.org , PGP key)
7. Intel PSIRT (secure@intel.com, PGP key)
8. Books http://www.apress.com/apressopen UEFI SHELL
9. UEFI Overview UEFI Intel Technology Journal
10. Quark Soc X1000 Version 1.1.0 BIOS https://downloadcenter.intel.com/download/23197/Intel-Quark-BSP
11. MinnowMax http://www.minnowboard.org/meet-minnowboard-max/

Thank You CanSecWest 2015