UEFI as the Converged Firmware Infrastructure

Dong Wei
VP (UEFI Forum), Chief UEFI Architect (HP)
April 5, 2012

©2012 Hewlett-Packard Development Company, L.P.
The information contained herein is subject to change without notice

Motivations

- Why UEFI?

Extensible, Flexible, & Unified

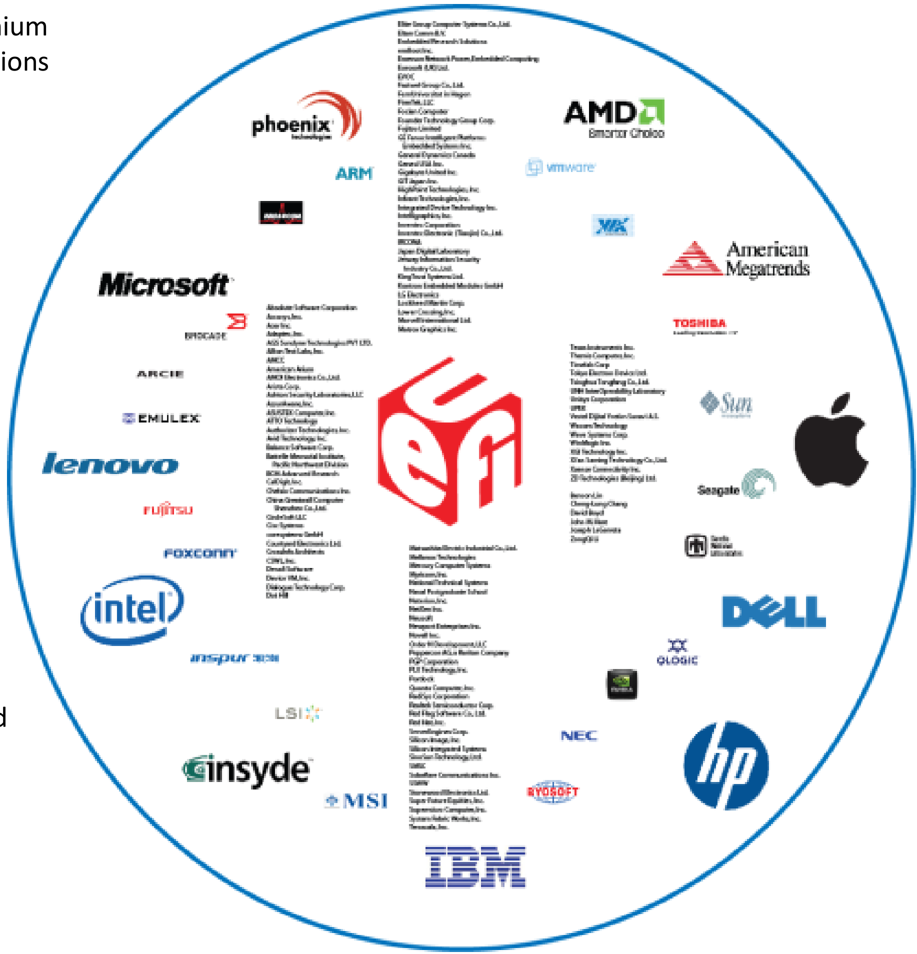
- Move away from PC-AT legacy
  - 16-bit x86 processor architecture
  - 1 MB memory space
  - Expansion ROM execution space limit
  - 2.2TB MBR formatted HDD
  - IO Port space dependency (incl. PCI Config Space CFC/CF8)
  - VGA
  - Single PCI segment group
  - PIC, PIT dependency
  - Name space collision
- Unified converged firmware infrastructure
  - Processor architecture agnostic (x86, x64, ARM, ia64, …)
  - PE-COFF executables
  - GPT formatted HDD
  - GOP
  - OSF-defined GUID as key element to avoid collision

History

- Past, Present, Future

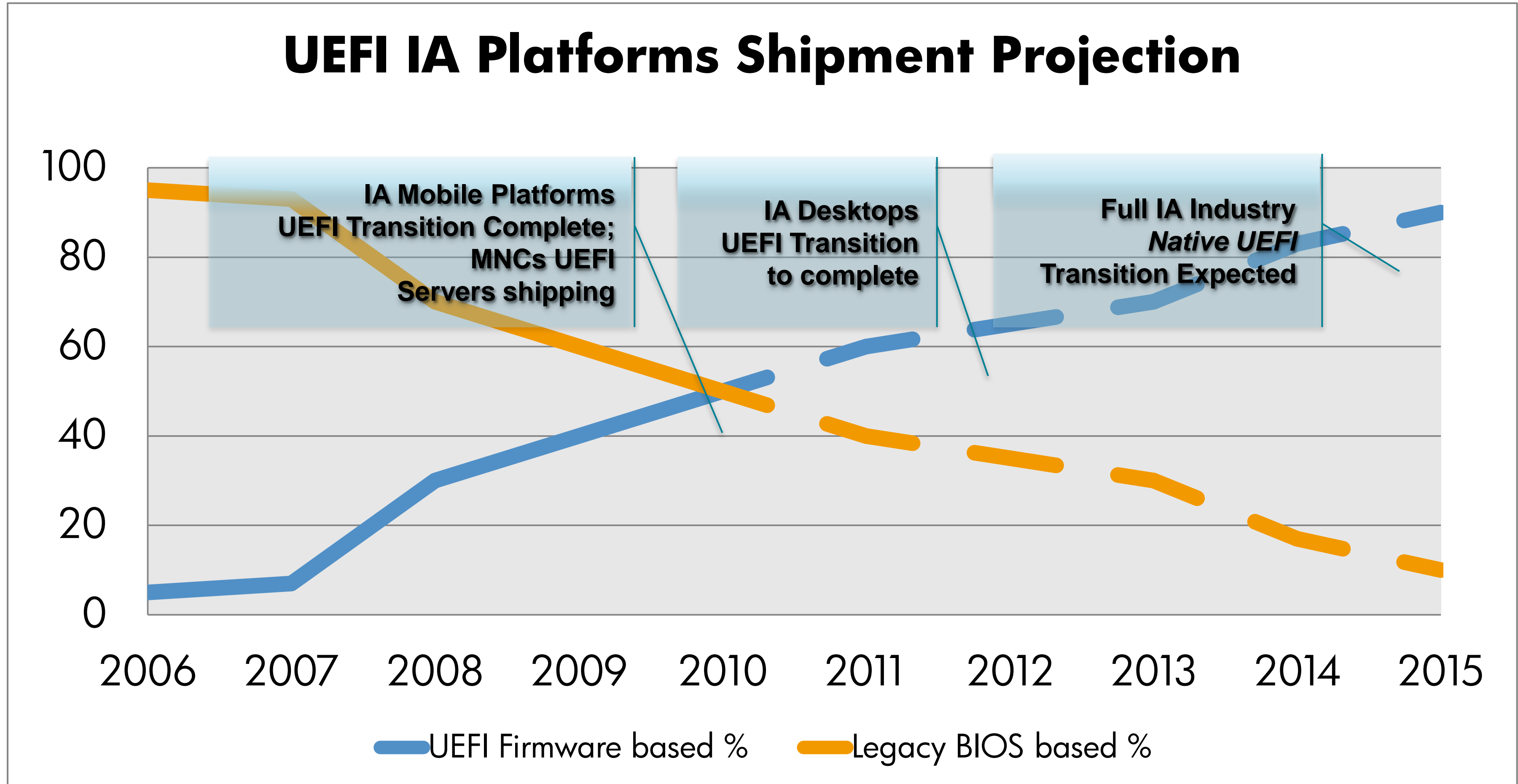
UEFI Industry Transition

1995	HP/Intel needs a boot architecture for Itanium servers that overcomes BIOS PC-AT limitations
1997-2000	Intel created EFI with HP and others in the industry, made it **processor agnostic (x86, ia64)**
2004	**tianocore.org**, open source EFI community launched
2005	**Unified EFI (UEFI)** Industry forum, with 11 promoters, was formed to standardize EFI, extended to x64
2009	UEFI extended to ARM
2012	215 members and growing! Windows 8 and ubiquitous native UEFI adoption for client PCs
Future	UEFI as the converged firmware infrastructure



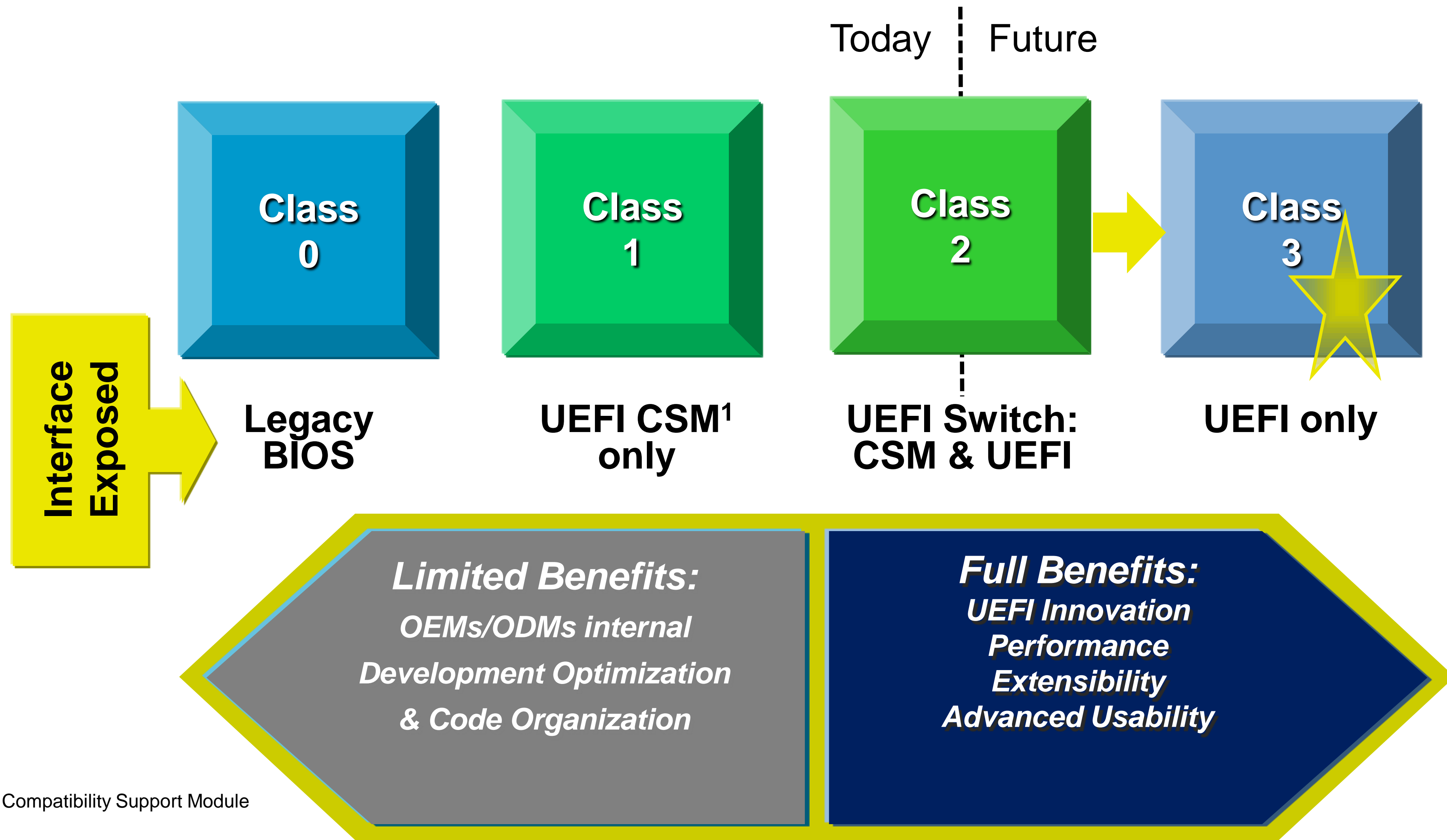
UEFI Firmware Deployments

Over 50% of worldwide IA units in 2010 and expected to reach 90% by 2015



Source: Various – UEFI Industry Communication Working Group data through 2010; Intel customers platform UEFI adoption projection data for 2011-2015

UEFI System Classes Based on Firmware I/F



[1]Compatibility Support Module

Specifications vs. Implementations

- Implementations vary

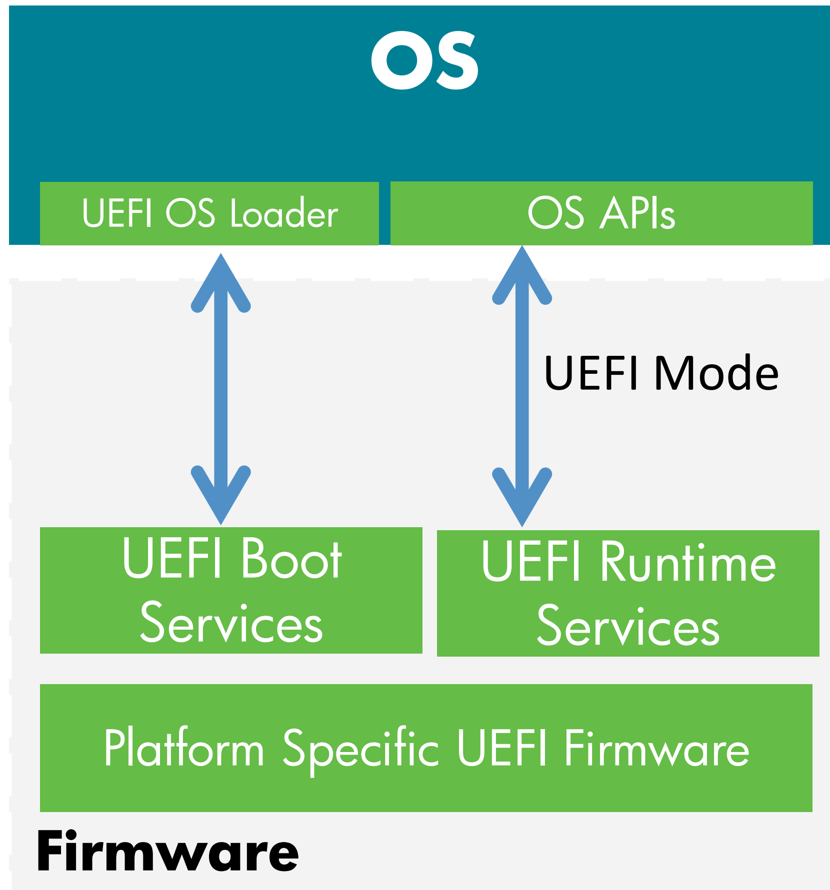
Why a Specification

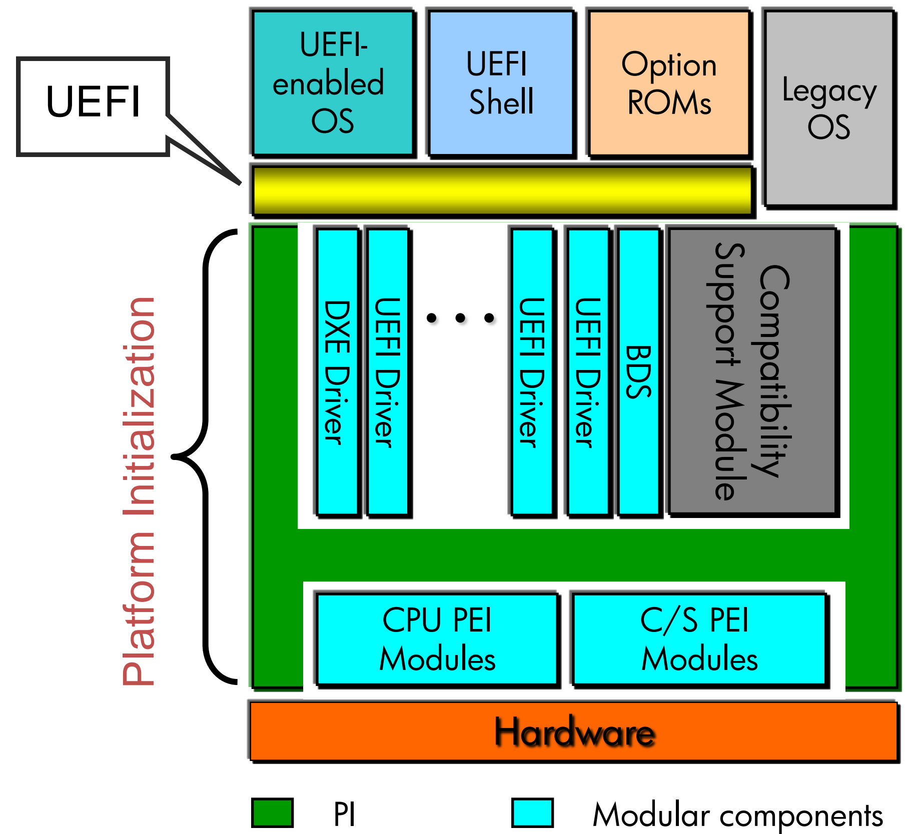
- OEMs want to differentiate their systems in hardware and firmware
- Multiple operating environments are expected to be supported
- Third party device boot drivers may be needed
- MFG/DIAG/TEST tools are needed
- Implementations differ, but all players can agree on abstracted interfaces
- UEFI provides a converged firmware infrastructure
- A "Contract" between interface producers and consumers is needed
  - Interfaces and the responsibilities of producers and consumers need to be written down clear enough for reference (Specification)
    - One way to manage interoperability, appropriate for the community served
  - Proof-of-concept needs to be provided (reference implementation)
  - Testability needs to be established (SCT)

Detailed Specification Helps Assure Interoperability

hp

UEFI View





Specifications

- The UEFI Forum defines and promotes the Specifications and SCT
- **UEFI Specification**
  - **Interface definition between OS and system firmware**
  - Interface definition between UEFI drivers/applications and system firmware
- UEFI SCT
  - Test suites for UEFI compliance
- UEFI Shell Specification
- PI Specifications & PI Packaging Specification
  - System firmware internal interface definition
  - OS developers: leave PI to the firmware developers!
- UEFI compliance does not require PI compliance

OS developers: focus on the interfaces that OS consumes!

OS developers: leave PI to the firmware developers!

Part of the Spec OS May Care

- Like a dictionary or encyclopedia, the spec is precise and complete as far as each entry goes but it's not meant to be read like a novel. It's for reference.
- Baseline (~400 out of 2214 pages)
  - 2.3 Calling Convention
  - 3.1 FW Boot Manager
  - 4.3 EFI System Table
  - Ch 5 GPT Format
  - Ch 6 Boot Services
  - Ch 7 Runtime Services
  - Ch 8 EFI Loaded Images
  - Ch 9 Device Path Protocol
  - Ch 11 Console Support
  - 12.4, 12.7, 12.8, 12.9 Media Access
  - 18.5 Decompress Protocol
- PXE support, if supported (44 pages)
  - 21.1, 21.3
- Secure Boot, if supported (33 pages)
  - 27.1~27.8

Implementations

- Implementations vary
  - Reference Implementations
    - EFI Sample Implementation
    - EDK (e.g., EDK1117) reference implementation
    - EDK II (e.g., UDK2010 SR1) reference implementation
  - IBV implementations and value-add
  - OEM in-house implementations and value-add
  - HP shipped systems with all of these flavors
- File numbers vary
  - Some implementations may maximize the reusability, others may minimize the file numbers
- ROM code sizes vary
  - Not all systems are created equal (from embedded systems to enterprise servers)
    - From hundreds of KB (e.g.,200KB) to several MB (e.g.,12MB)(note: SPI protocol traditionally limits to 16MB ROM)
  - No systems need to include all the reference sample code
  - UEFI enables modular designs, move away from spaghetti assumption

OS developers: design to the UEFI Specification, don't assume a particular codebase!

UEFI Deployment @ HP

Embedded Systems

Printers and Scanners

Scanjet Enterprise 7000n*, Color Laserjet CM4540 MFP*, Color LaserJet CP5525*, LaserJet M4555MFP*.

Storage

Network

PC Clients

Notebook PCs and Tablets

Commercial group has shipped Class 2 systems since 2008

Consumer group is shipping Class 1 systems

Desktops and Workstations

Adopted a common UEFI codebase

Shipped Class 2 systems since 2H'2010

Enterprise Servers

Integrity Servers

Always Class 3 systems

HP-UX, VMS, Integrity VM Operating Environment

Collaborate on HP UEFI features provided enhanced manageability, security and ease of code with shared UEFI-based diagnostics

UEFI Secure Boot

- Motivations, History, Definition

Why Implement UEFI Secure Boot?

- **As OS becomes more resistant to attack the threat targets the weakest element in the chain**
- **And 16-bit Legacy Boot is not secure!**

It should be no surprise that a TDL Gang botnet climbed into the number one position in the Damballa Threat Report – Top 10 Botnets of 2010. "RudeWarlockMob" … applied effective behaviors of old viruses and kits. It combined techniques that have been effective since the days of 16-bit operating systems, like Master Boot Record (MBR) infection … with newer malware techniques.
(from http://blog.damballa.com)

  - **http://www.theregister.co.uk/2011/09/14/bios_rootkit_discovered/**
  - http://www.theregister.co.uk/2011/11/18/windows_8_bootkit/
- **Secure Boot based on UEFI 2.3.1 removes the Legacy Threat and provides software identity checking at every step of boot – Platform Firmware, Option Cards, and OS Bootloader**

History

- Phoenix initiated the discussion on the need for secure boot
- USST (UEFI Security Subteam) formed to address the topic
- The secure boot architecture was defined in the UEFI 2.3 Specification
- Microsoft contributed additional capabilities for UEFI 2.3.1 Specification
  - Append support for the authenticated variables
  - Time-based authenticated variable for roll-back protection
  - Authenticode specification for use in UEFI
  - UEFI Secure Boot support in Windows 8

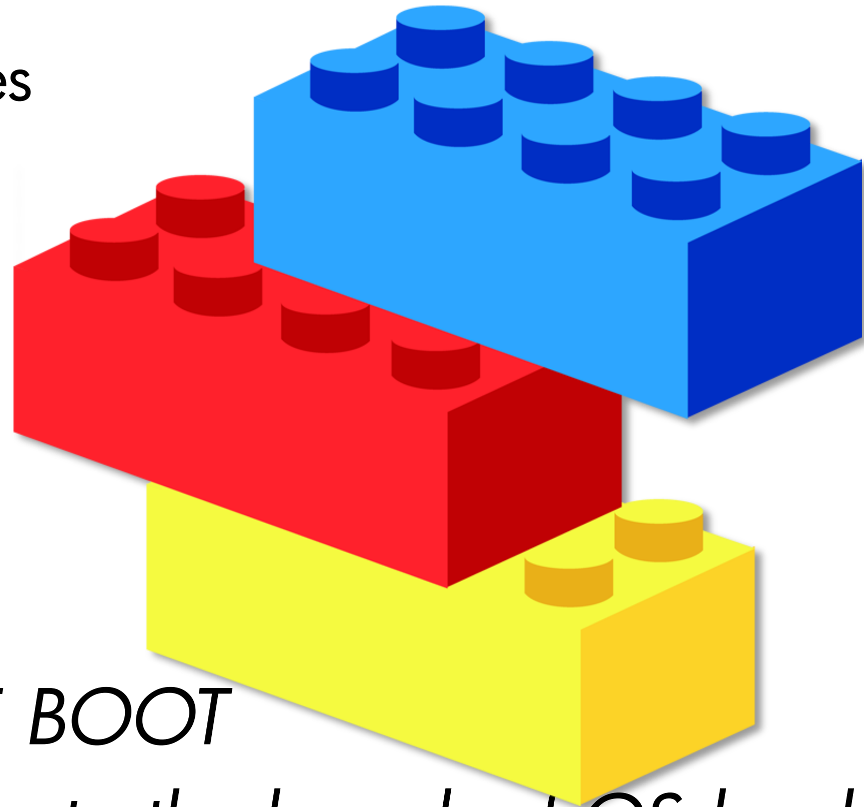
UEFI Secure Boot support is an industry effort.

Secure Boot – Three Components

1. Authenticated Variables

2. Driver Signing

3. System Defined Variables



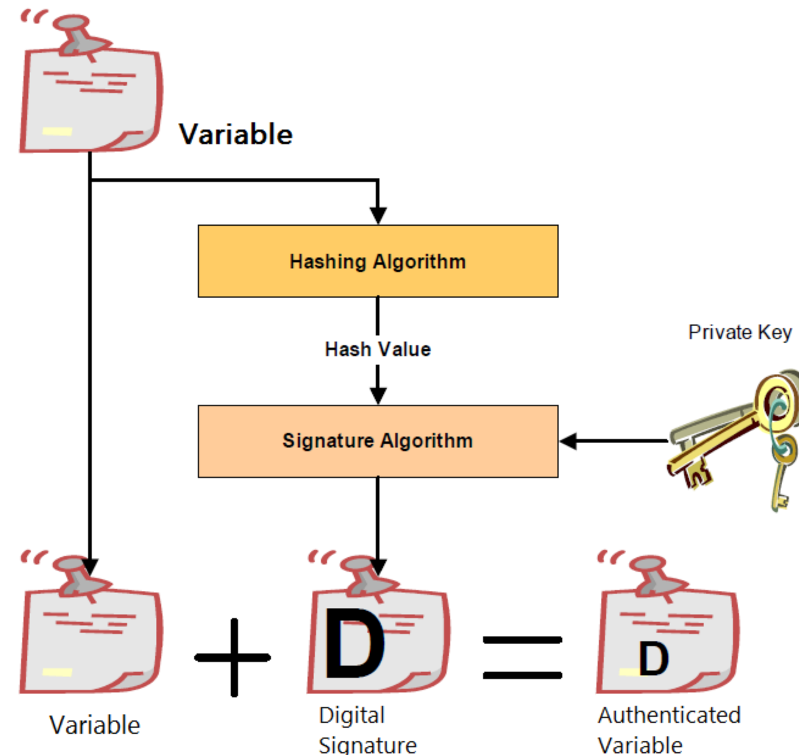
UEFI 2.3.1 SECURE BOOT

Scope: From power-on to the launch of OS loader

NO dependency on TPM

UEFI Authenticated Variables

- Uses standard UEFI Variable Functions
- Available Pre-boot and also Runtime
- Typically stored in Flash
- Variable Creator signs Variable Hash with Private Key (PKCS-7 Format)
- Signature & Variable Passed Together for Create, Replace, Extend, or Delete
- Several System-defined variables for Secure Boot



Extensible Integrity Architecture

Updating Authenticated Variable

- Support for Append added (UEFI 2.3.1)
- Counter-based authenticated variable (UEFI 2.3)
  - Uses monotonic count to protect against suspicious replay attack
  - Hashing algorithm – SHA256
  - Signature algorithm – RSA-2048

- Time-based authenticated variable (UEFI 2.3.1)
  - Uses timestamp as rollback protection mechanism
  - Hashing algorithm – SHA256
  - Signature algorithm – X.509 certificate chains
    - Complete X.509 certificate chain
    - Intermediate certificate support (non-root certificate as trusted certificate).

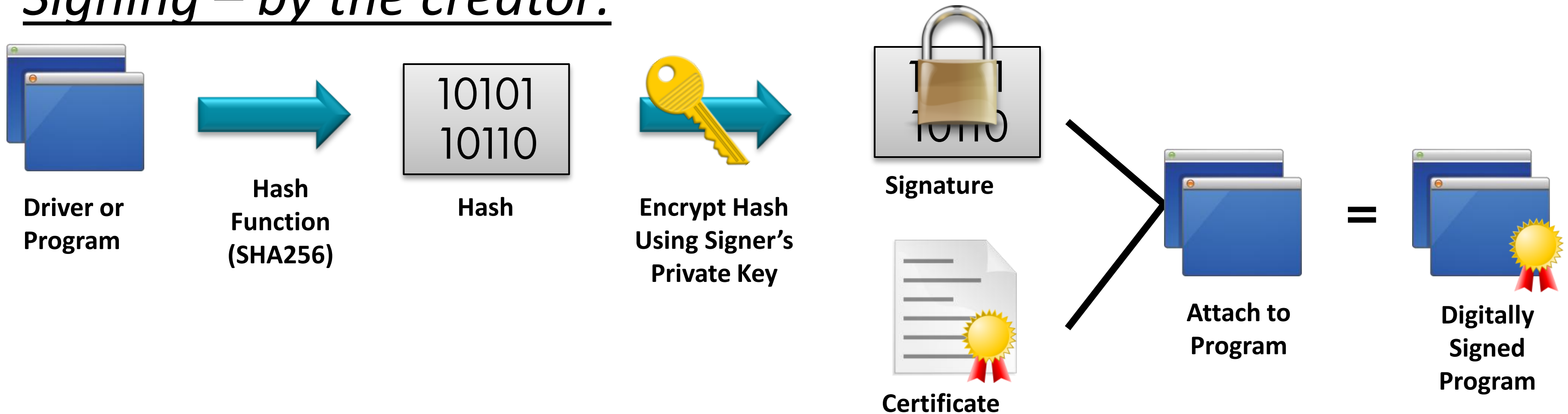
Protected Variables that can be Securely Updated

UEFI Driver Signing

- In Secure Boot, signatures should be checked:
  1. UEFI Drivers loaded from PCI-Express Cards
  2. Drivers loaded from mass storage
  3. Pre-boot EFI Shell Applications, FW updaters
  4. OS UEFI Boot-loaders
- UEFI Signing is not applied to
  1. Drivers in the Factory BIOS
  2. Legacy components
- UEFI Driver Signing Utilizes Microsoft* Authenticode* Technology to sign UEFI executables
  - Authenticode and PE/COFF are licensed for use in UEFI

UEFI Driver Signing

Signing – by the creator:



Verification – In the PC:

Digitally Signed Driver or Program

Program

Hash Function

10101
10110

Hash

=?

Signature

Decrypt Hash with Signer's Public Key

10101
10110

Hash

Check local databases for certificate. If certificate found and not revoked, run UEFI Executable.

Secure Boot Authenticated Variables

PK	**P**latform **K**ey – Root key set to enable Secure Boot
KEK	**K**ey **E**xchange **K**ey List of Cert. Owners with db, dbx update privilege
db	List of Allowed Driver or App. Signers (or hashes)
dbx	List of Revoked Signers (or hashes)
SetupMode	1= in Setup Mode, 0 = PK is Set (User Mode)
SecureBoot	1 = Secure Boot in force

Notes:

- Owner of cert. in KEK can update db, dbx
- Owner of cert. in PK can update KEK

UEFI Defines System Databases for Secure Boot

hp

Secure Boot Begins @ the Factory

Pre-production —— **Production** —— **User**

Certificate Generating Station @ OEM

1

2

3

OEM collects certificates provided by OSVs, Partners, and OEM's own keys.

"DB Generator" creates the Initial Security Load for new computers.

Initial Security Load is installed onto each computer at the factory, enabling Secure Boot.
1) Initial db and dbx
2) KEK with allowed updaters
3) Platform Key (PK)

After delivery, the OEM or OSV can update with new certificates or revoked certificates*

OEM Responsible for Initializing Secure Boot

**And OEM can allow User To Disable Secure Boot in 'Setup'*

Secure Boot Protects the User

User attempts to boot a compromised system

OS Boot-loader image checked against pre-loaded database

Root-kit fails checks, user protected by Secure Boot

Secure Boot Tests Signatures to Reject Potential Threats

UEFI Forum: Although an optional feature in the UEFI Specification, the UEFI Forum expects to see platforms with UEFI Secure Boot supporting both commercial and open source operating systems. Our members view that as a significant tool for both in the fight against pre-OS malware.

UEFI Signing Service

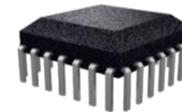
- The UEFI Forum decided not to host a UEFI Signing Service
  - Liability too high, cost prohibitive
  - Instead, worked with the Industry to provide UEFI Signing Service(s)
  - Microsoft offered an independent certificate authority (CA) for UEFI, with a well defined and fair process to blacklist a signature, including themselves
- Microsoft plans to use Winqual to provide a UEFI Signing Service
  - Winqual hosts 11000+ companies (including some Linux distros and Apple)
  - Minimal one-time administrative costs
  - Free signing of UEFI images
- Other entities can work with the UEFI Forum to provide UEFI Signing Service(s) as well
- Would like to keep the number of CAs at the minimum
  - Keys need to be stored in flash

UEFI Secure Boot & TCG Trusted Boot

- Complementary

What is a TPM?

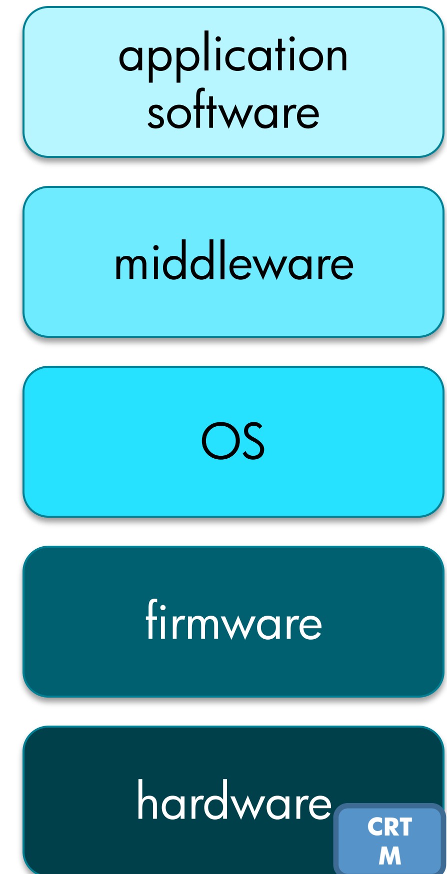
- A hardware security component
  - Think smartcard on the motherboard
  - Could be integrated firmware, but should provided tamper resistance
- A root of trust in the platform
  - Support for asymmetric cryptography
  - Support for protected storage
  - Support for migratable and non-migratable crypto keys



- At the core of the TCG Trusted Boot architecture
  - TPM can be used to record and report software hashes
  - TPM enables attestation of measurements to remote entities
  - TCG Trusted Boot requires an external entity to validate the measurements and to apply the right policies

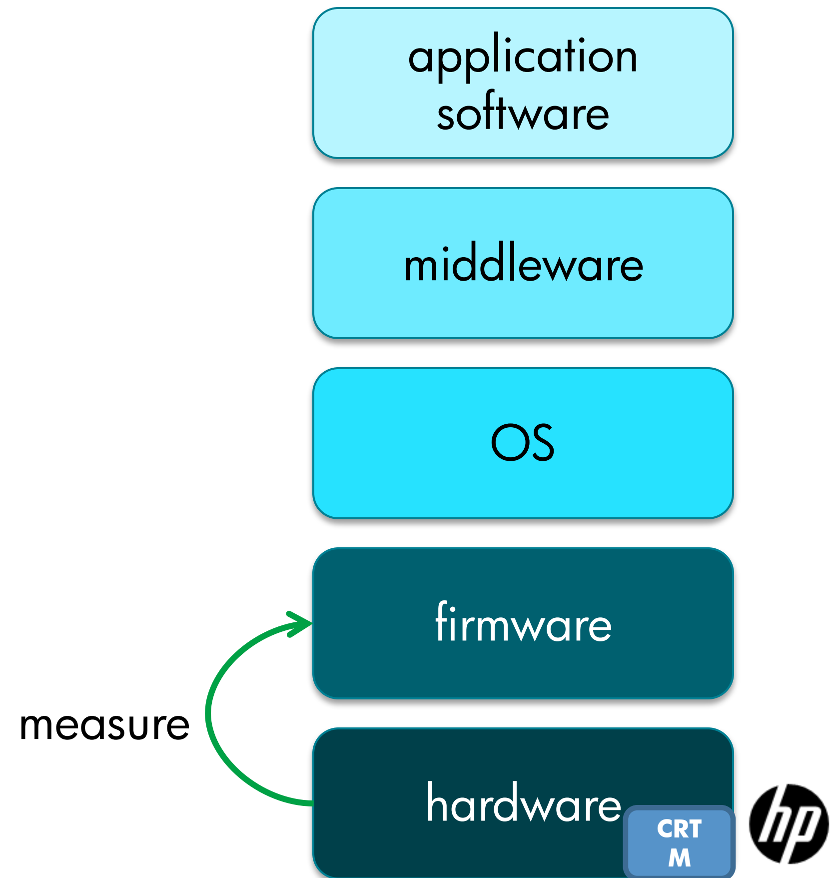
TCG Trusted Boot: Building a Chain of Trust

- The concept of a TCG Trusted Boot is to have each component in the chain be *measured* by the preceding one and recorded in a way that can be reliably reported
- TCG Trusted Boot makes no judgment on the trustworthiness of any module
- TCG Trusted boot relies on TPM and on a trusted boot block (CRTM)



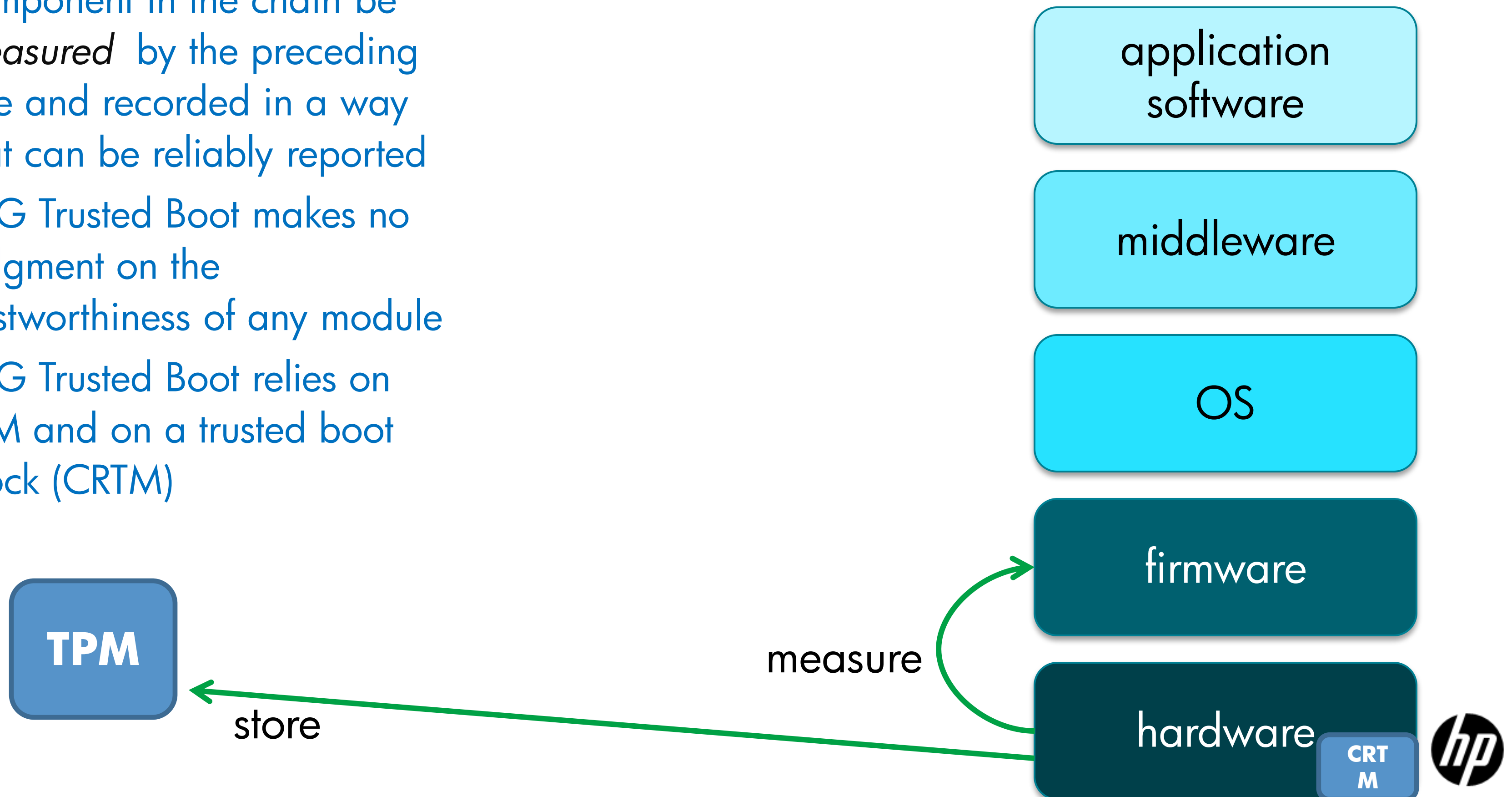
TCG Trusted Boot: Building a Chain of Trust

- The concept of a TCG Trusted Boot is to have each component in the chain be *measured* by the preceding one and recorded in a way that can be reliably reported
- TCG Trusted Boot makes no judgment on the trustworthiness of any module
- TCG Trusted Boot relies on TPM and on a trusted boot block (CRTM)



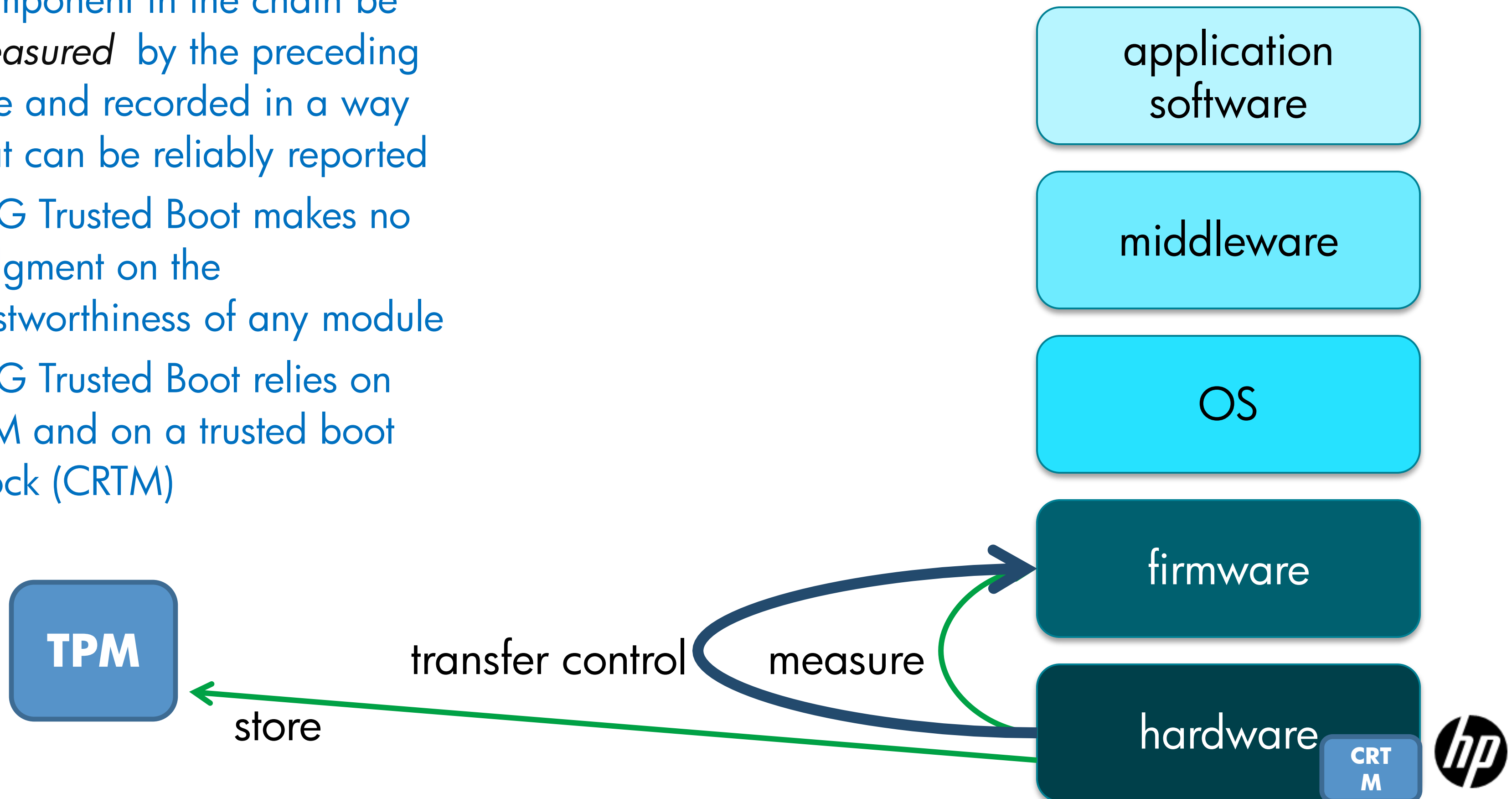
TCG Trusted Boot: Building a Chain of Trust

- The concept of a TCG Trusted Boot is to have each component in the chain be *measured* by the preceding one and recorded in a way that can be reliably reported
- TCG Trusted Boot makes no judgment on the trustworthiness of any module
- TCG Trusted Boot relies on TPM and on a trusted boot block (CRTM)



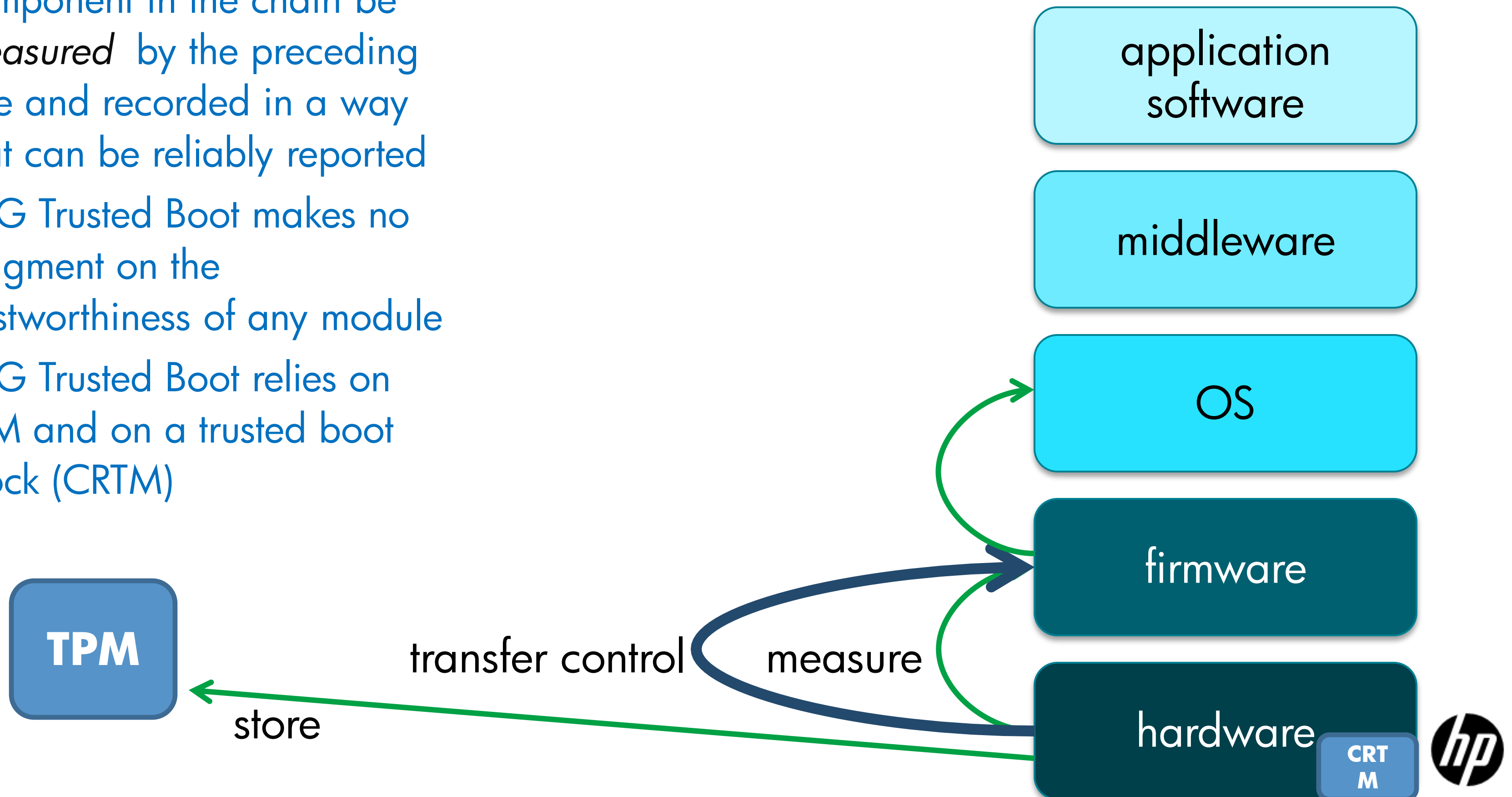
TCG Trusted Boot: Building a Chain of Trust

- The concept of a TCG Trusted Boot is to have each component in the chain be *measured* by the preceding one and recorded in a way that can be reliably reported
- TCG Trusted Boot makes no judgment on the trustworthiness of any module
- TCG Trusted Boot relies on TPM and on a trusted boot block (CRTM)



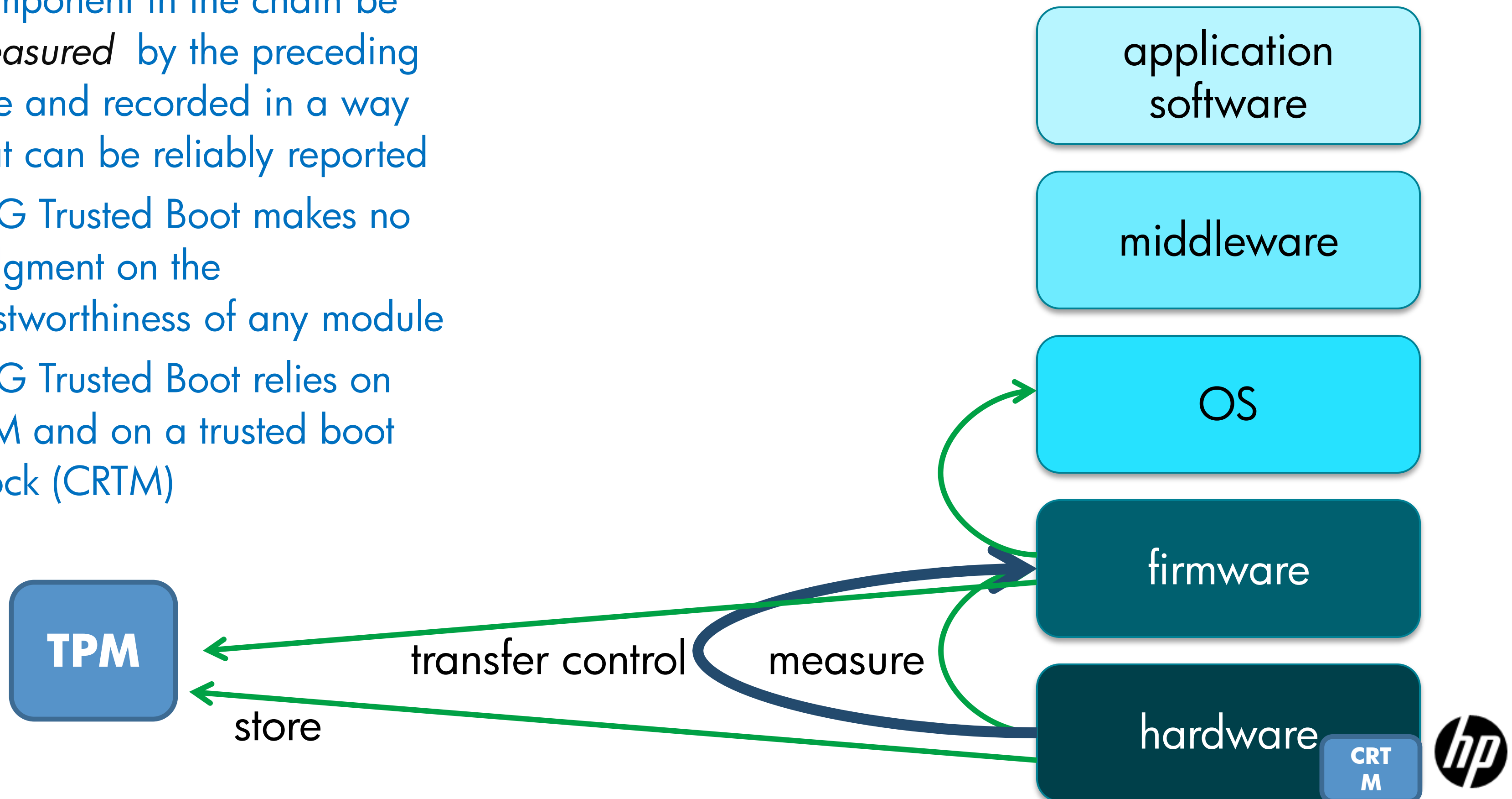
TCG Trusted Boot: Building a Chain of Trust

- The concept of a TCG Trusted Boot is to have each component in the chain be *measured* by the preceding one and recorded in a way that can be reliably reported
- TCG Trusted Boot makes no judgment on the trustworthiness of any module
- TCG Trusted Boot relies on TPM and on a trusted boot block (CRTM)



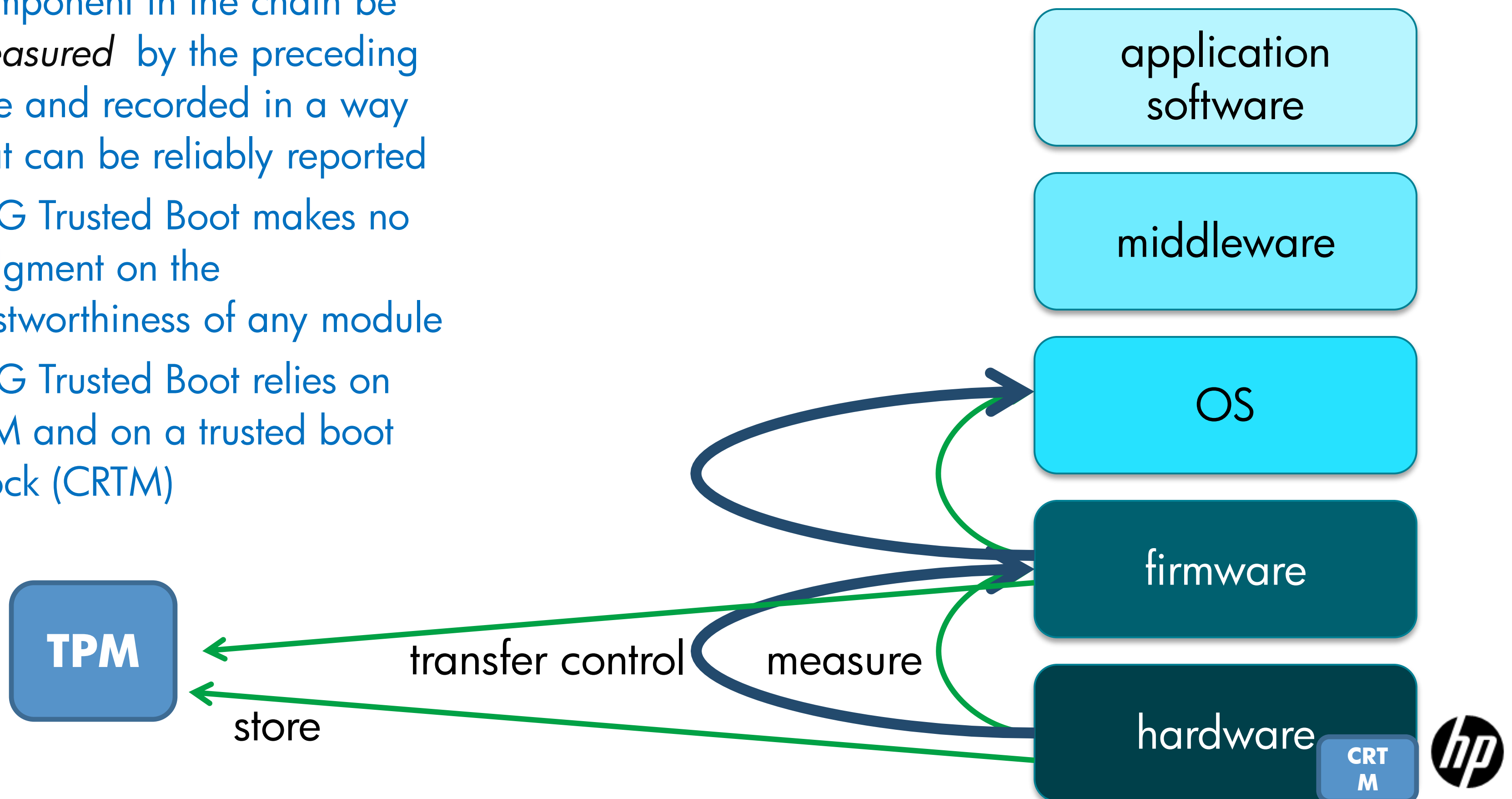
TCG Trusted Boot: Building a Chain of Trust

- The concept of a TCG Trusted Boot is to have each component in the chain be *measured* by the preceding one and recorded in a way that can be reliably reported
- TCG Trusted Boot makes no judgment on the trustworthiness of any module
- TCG Trusted Boot relies on TPM and on a trusted boot block (CRTM)



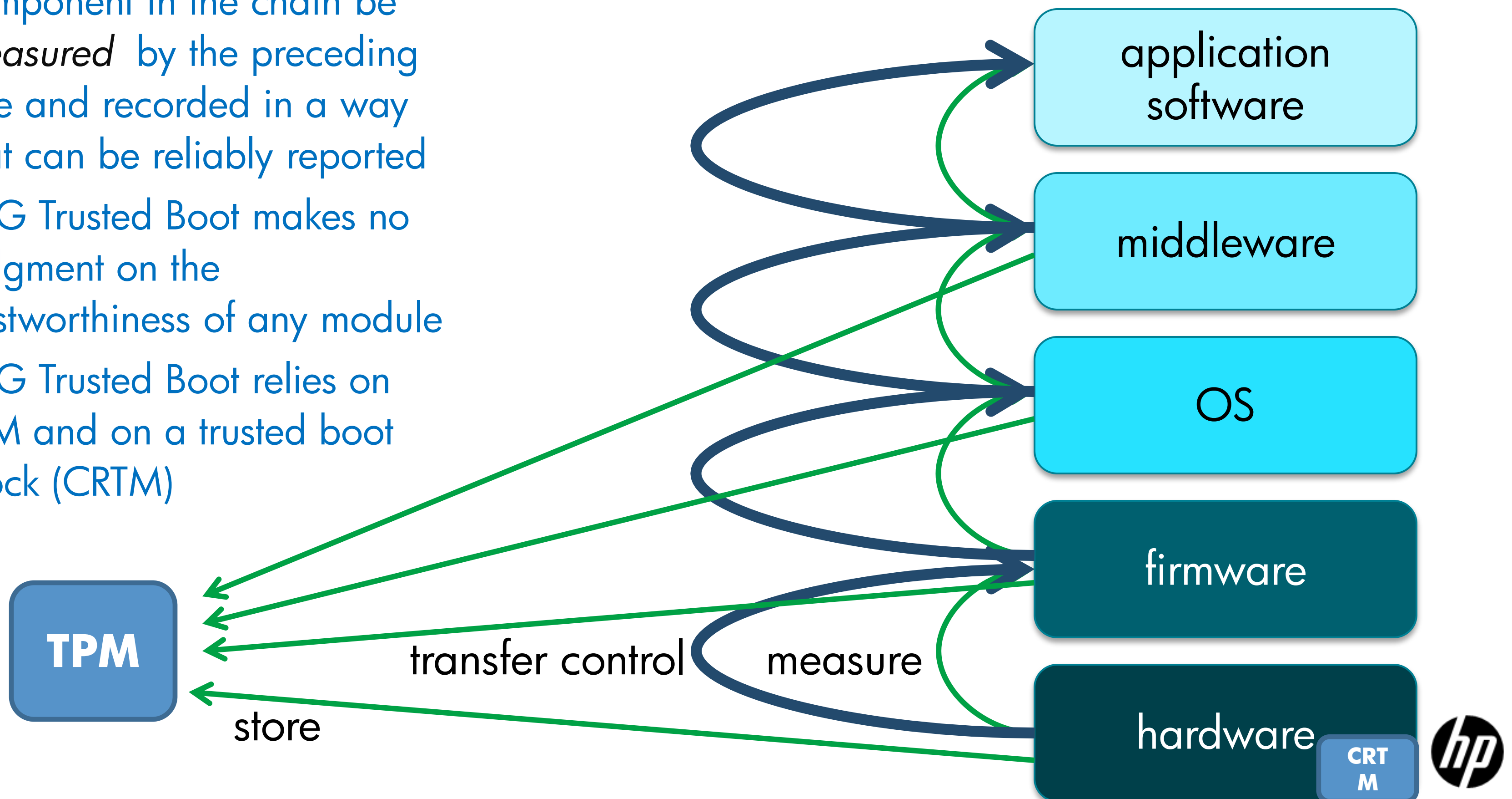
TCG Trusted Boot: Building a Chain of Trust

- The concept of a TCG Trusted Boot is to have each component in the chain be *measured* by the preceding one and recorded in a way that can be reliably reported
- TCG Trusted Boot makes no judgment on the trustworthiness of any module
- TCG Trusted Boot relies on TPM and on a trusted boot block (CRTM)



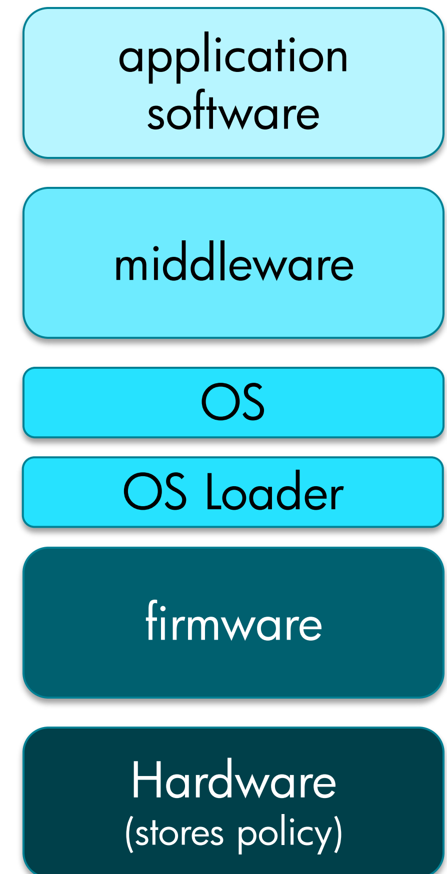
TCG Trusted Boot: Building a Chain of Trust

- The concept of a TCG Trusted Boot is to have each component in the chain be *measured* by the preceding one and recorded in a way that can be reliably reported
- TCG Trusted Boot makes no judgment on the trustworthiness of any module
- TCG Trusted Boot relies on TPM and on a trusted boot block (CRTM)



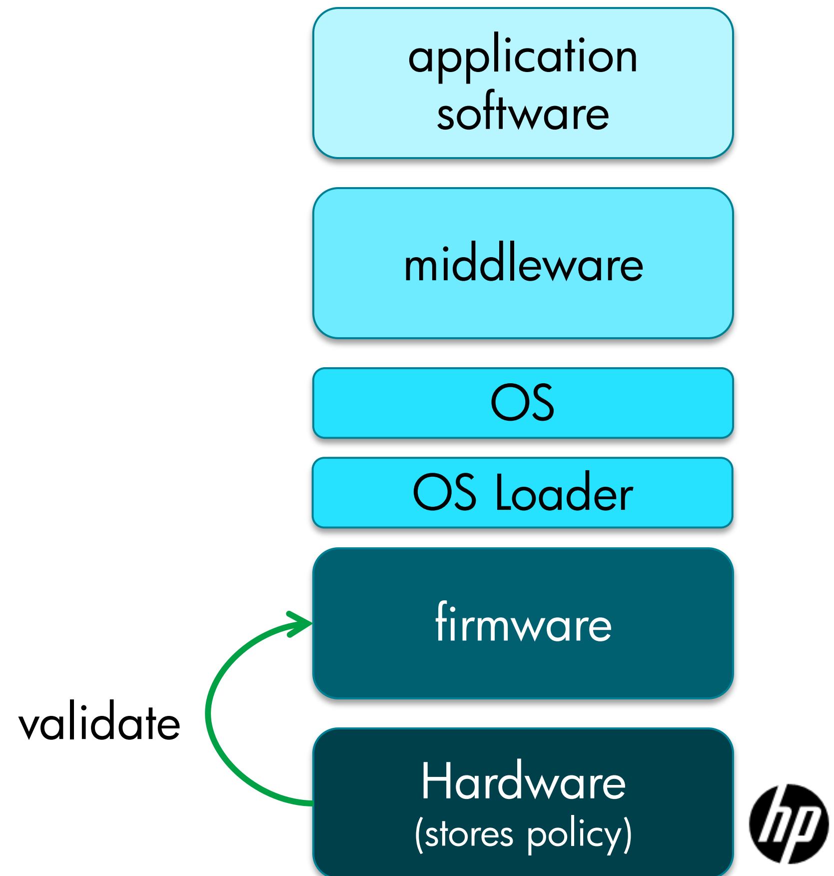
UEFI Secure boot: Enforcing Boot Policy

- The concept of UEFI secure boot is to have each component in the chain be *validated and authorized* against a given policy before allowing it to execute
- UEFI secure boot policy implementations can range from digital signatures to preloaded hash values…



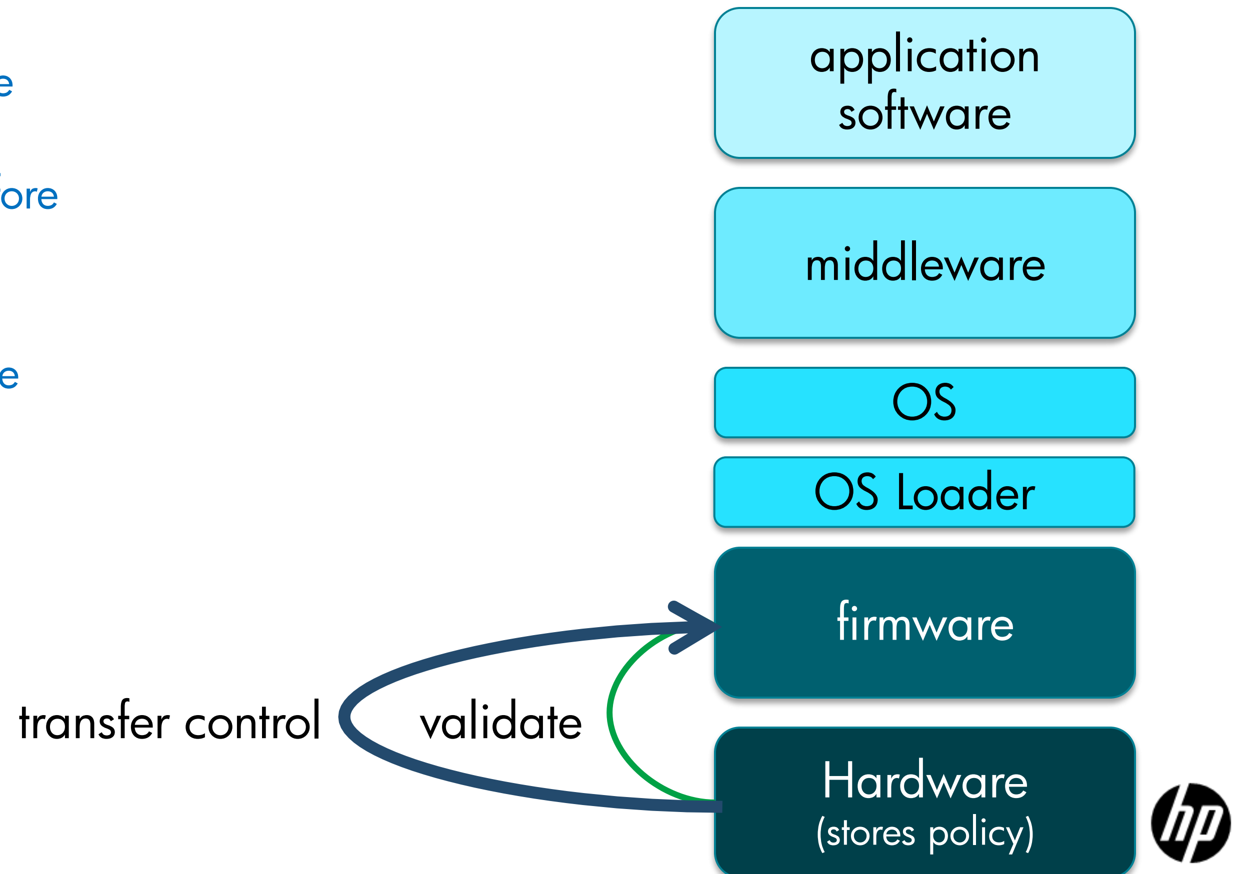
UEFI Secure boot: Enforcing Boot Policy

- The concept of UEFI secure boot is to have each component in the chain be *validated and authorized* against a given policy before allowing it to execute
- UEFI secure boot policy implementations can range from digital signatures to preloaded hash values…



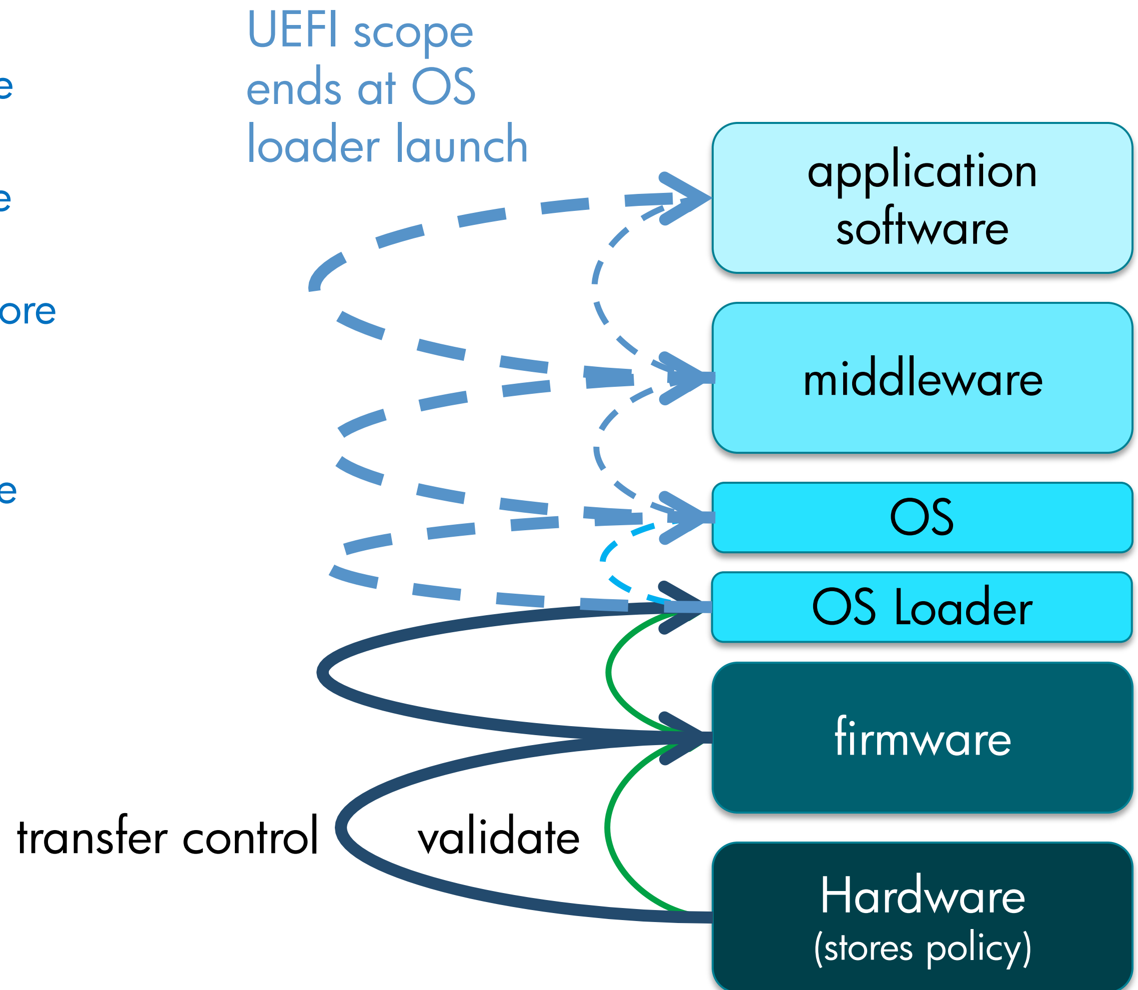
UEFI Secure boot: Enforcing Boot Policy

- The concept of UEFI secure boot is to have each component in the chain be *validated and authorized* against a given policy before allowing it to execute
- UEFI secure boot policy implementations can range from digital signatures to preloaded hash values…



UEFI Secure boot: Enforcing Boot Policy

- The concept of UEFI secure boot is to have each component in the chain be *validated and authorized* against a given policy before allowing it to execute
- UEFI secure boot policy implementations can range from digital signatures to preloaded hash values…



UEFI Secure Boot and TPM

- No interdependency
  - UEFI Secure boot does not require TPM
  - TPM does not require UEFI Secure Boot
- Some complementarity
  - TPM can be used to implement other types of Secure Boot
    - i.e. to store certificates in NVRAM, or prevent modification of boot policies
  - UEFI Secure Boot can be used in combination to TCG Trusted Boot
    - Typically to simplify state complexity and increase robustness

Securing the Software Stack

- UEFI 2.3.1 security enhancements specifically address the "secure boot" issue
- Securing the firmware itself further strengthens the UEFI Secure Boot concept
  - *How is the firmware update protected?*
  - *How is the firmware put into "admin mode"?*
- NIST has created *BIOS Protection Guidelines*
  - Secure flash update requirements
  - Maintain firmware core root of trust
- UEFI 2.3.1 contains the framework to develop secure flash update

NIST Implementation Requirements

The NIST BIOS Protection Guidelines break down to three basic requirements…

1. Authenticity: BIOS updates must be signed
2. Integrity: BIOS must be protected
3. Non-bypassability: only the authenticated BIOS update mechanism can modify the BIOS

hp

Linux support for UEFI Secure Boot

- Issues & Solutions

Issues Identified and Possible Solutions

Secure boot may not be supported by an OS

- Platforms, if supporting secure boot, provide a switch to enable/disable secure boot

Corporate IT or enthusiasts may want to enroll their own keys

- Users can move the system into setup mode, if platform supports
- Take the full responsibility by providing their own PK
- Do not assume OEM PK can be re-enrolled!
- OEM tools can still work (use a separate OEM key to sign these tools)

Key databases are stored in NVM space

- Resource limited, cannot accommodate each and every possible self-signers
- All UEFI drivers and applications are expected to be signed with the UEFI CA Service hosted by Microsoft
- Ideally, OS bootloaders (a special form of UEFI application) can also be signed with the UEFI CA service

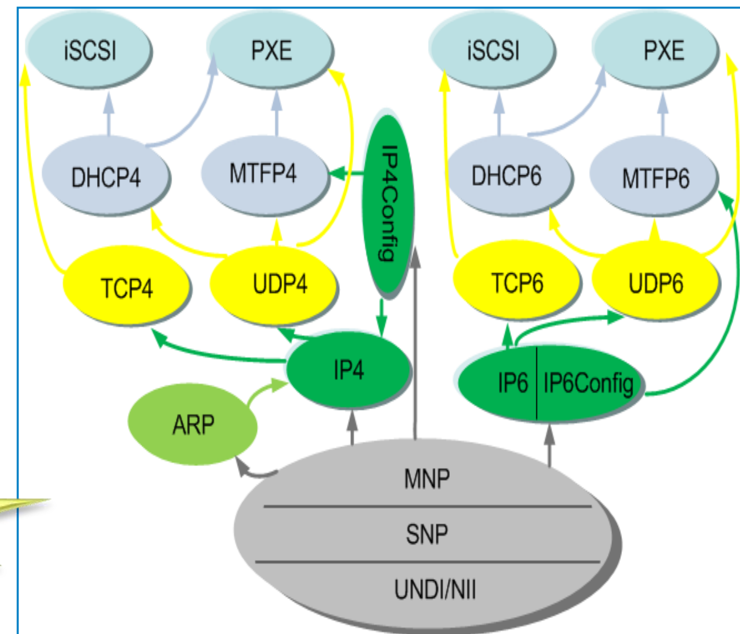
Complete secure boot support needs more than UEFI

- Secure boot beyond the execution of the OS bootloader is beyond the scope of UEFI
- OS needs to provide its own secure infrastructure once the bootloader takes over

IPv6 support via UEFI

Focus: IPv6 Networking

- IPv6 protocol compliance
  - "IPv6 ready" logo approved
  - Requirements for IPv6 transition (PDF)
- UEFI IPv6 Features
  - IP4/6, UDP4/6, TCP4/6
  - DHCP4/6, MTFP4/6
  - iSCSI, PXE, IPsec
  - Allows for concurrent network applications
    Dual stack (IPv4 and/or IPv6)
  - DUID-UUID support
    - New in UEFI 2.3.1
    - Use SMBIOS system GUID as UUID



OS Support for Netboot6[1]

- SUSE* Linux Enterprise Server 11 Service Pack 2 x86_64 Beta 4* (SLES 11 SP2 x86_64 Beta 4)
  - Supports UEFI 2.3.1 PXE Netboot6
  - Can support at the same time requests for booting PXE to both IPV4 and IPV6 UEFI 2.3.1 clients

[1] Details on Netbvoot6 can be found in the UEFI2.3.1 Specification

Call to Action

Call to Action

- Engage the UEFI community and make sure UEFI is Linux-ready
  - Linux is in a sense coming late to the UEFI party on x86.
  - There's been a LOT of testing of UEFI code with commercial operating systems. Not so much with Linux.
    - Opportunity to improve testing and hence quality of code measured by Linux-readiness if together we can work on producing tests that exercise UEFI APIs the way Linux code wants to do that.
  - There's no reason why Linux can't get the same or more mindshare with the firmware implementation world if folks on both sides are willing to learn how the other side works and how to communicate across what has been traditionally a wide divide

Q&A

Thank You!