Dr.Dobb's
THE WORLD OF SOFTWARE DEVELOPMENT

The Evolution of the Unified Extensible Firmware Interface

Moving beyond BIOS with the Unified Extensible Firmware Interface

November 22, 2010
URL:http://www.drdobbs.com/embedded-systems/the-evolution-of-the-unified-extensible/228300362

Vincent Zimmer and Michael Rothman are engineers in the Software and Services Group at Intel. Suresh Marisetty is a software and systems architect at Intel focusing on security and manageability. They are the authors of Beyond BIOS: Developing with the Unified Extensible Firmware Interface

---

In essence, the modern day BIOS is one that has survived for more than 25 years. However, the program and instructions themselves have been dramatically changed and adapted over time to match the ever quickening pace of technological advancement. The goals of the Extensible Firmware Interface (EFI), and now the Unified Extensible Firmware Interface (UEFI), is to eventually replace the aging BIOS, learning from both the mistakes and successes of the past 25 years. This article provides an overview of the evolution of the Extensible Firmware Interface (EFI) to the Unified Extensible Firmware Interface (UEFI) and from the Intel Framework specifications to the UEFI Platform Initialization (PI) specifications.

When we discuss UEFI, we need to emphasize that UEFI is a pure interface specification that does not dictate how the platform firmware is built; the "how" is relegated to PI. The consumers of UEFI include but are not limited to operating system loaders, installers, adapter ROMs from boot devices, pre-OS diagnostics, utilities, and OS runtimes (for the small set of UEFI runtime services). In general, though, UEFI is about booting, or passing control to a successive layer of control, namely an operating system loader, as shown in Figure 1. UEFI offers many interesting capabilities and can exist as a limited runtime for some application set, in lieu of loading a full, shrink-wrapped multi-address space operating systems like Microsoft Windows+, Apple OS X+, HP-UX+, or Linux, but that is not the primary design goal.

[Click image to view at full size]

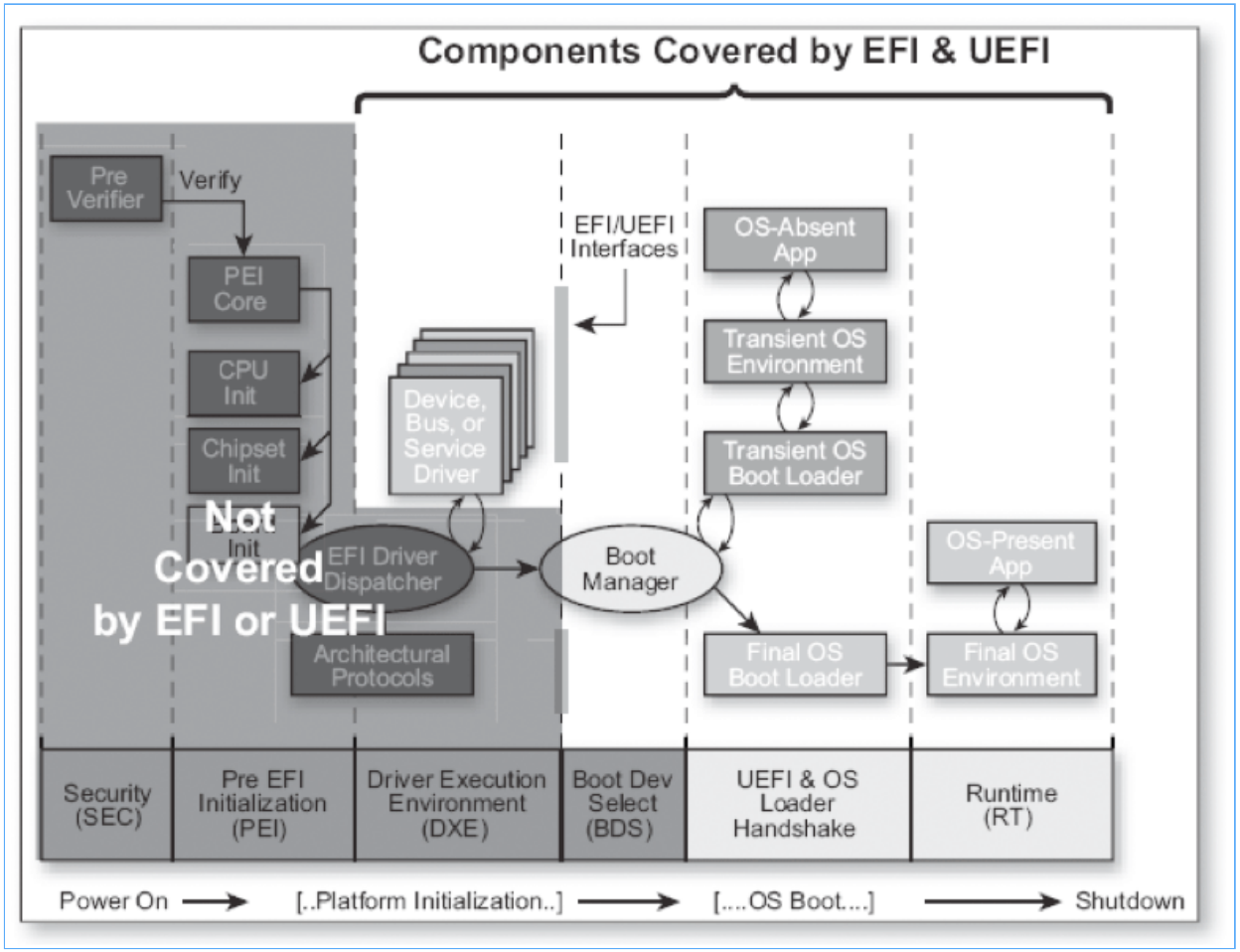

Figure 1: Where EFI and UEFI Fit into the Platform Boot Flow.

PI, on the other hand, should be largely opaque to the pre-OS boot devices, operating systems, and their loaders since it covers many software aspects of platform construction that are irrelevant to those consumers. PI instead describes the phases of control from the platform reset and into the success phase of operation, including an environment compatible with UEFI, as shown in Figure 2. In fact, the PI DXE component is the preferred UEFI core implementation.

[Click image to view at full size]

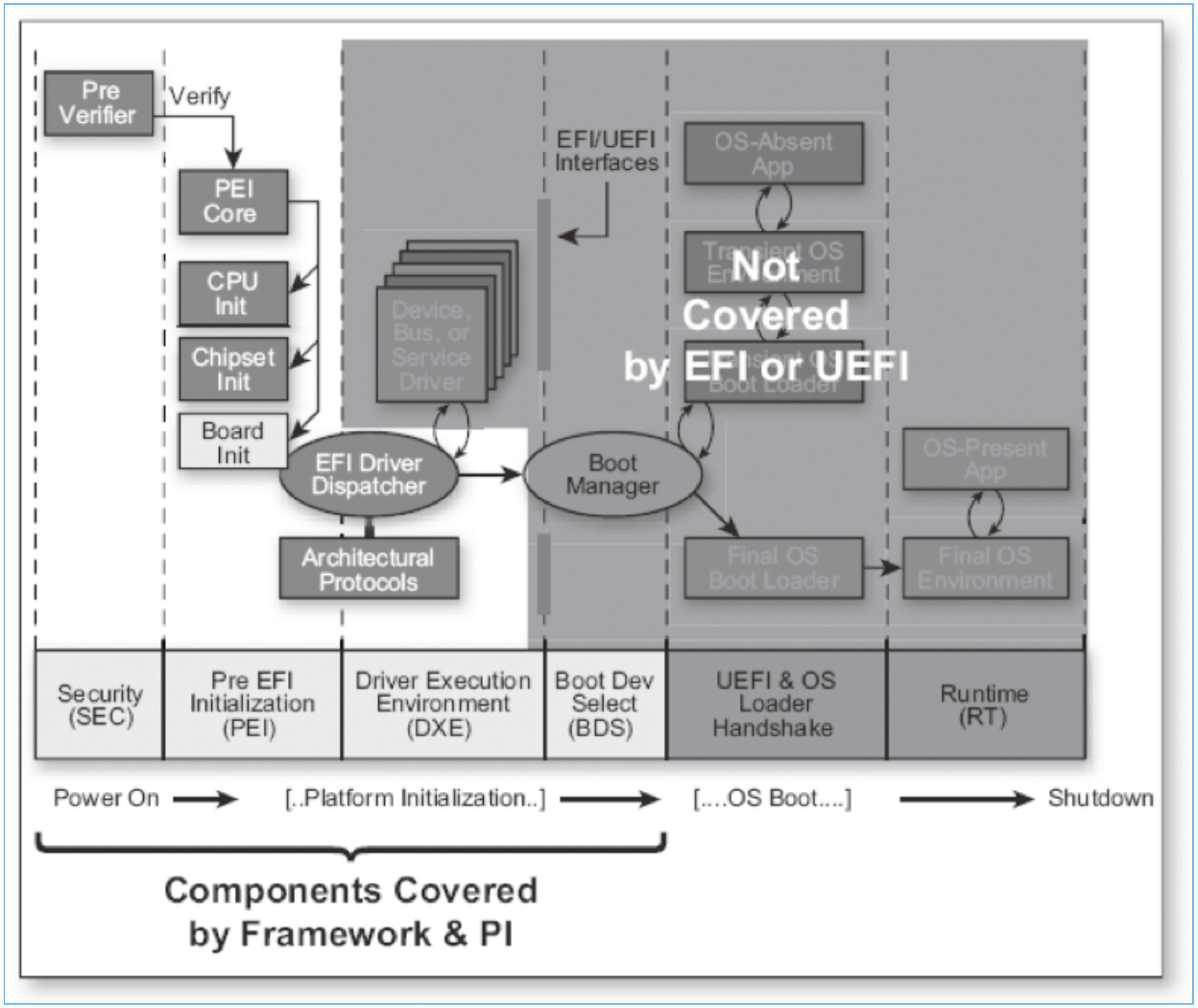

Figure 2: Where PI and Framework Fit into the Platform Boot Flow.

Within the evolution of Framework to PI, some things were omitted from inclusion in the PI specifications. As a result of these omissions, some subjects that were discussed in the first edition of *Beyond BIOS*, such as the compatibility support module (CSM), have been removed from the second edition in order to provide space to describe the newer PI and UEFI capabilities. This omission is both from a scope perspective, namely that the PI specification didn't want to codify or include the CSM, but also from a long-term perspective. Specifically, the CSM specification abstracted booting on a PC/AT system. This requires an x86 processor, PC/AT hardware complex (for example, 8254, 8259, RTC). The CSM also inherited other conventional BIOS boot limitations, such as the 2.2-TB disk limit of Master Boot Record (MBR) partition tables. For a world of PI and UEFI, you get all of the x86 capabilities (IA-32 and x64, respectively), ARM+, Itanium, and future CPU bindings. Also, via the polled driver model design, UEFI APIs, and the PI DXE architectural protocols, the platform and component hardware details are abstracted from all consumer software. Other minor omissions also include data hub support. The latter has been replaced by purpose-built infrastructure to fill the role of data hub in Framework-based implementations, such as SMBIOS table creation and agents to log report status code actions.

What has happened in PI beyond Framework, though, includes the addition of a multiprocessor protocol, Itanium E-SAL and MCA support, the above-listed report-status code listener and SMBIOS protocol, an ACPI editing protocol, and an SIO protocol. With Framework collateral that moved to PI, a significant update was made to the System Management Mode (SMM) protocol and infrastructure to abstract out various CPU and chipset implementations from the more generic components. On the DXE front, small cleanup was added in consideration of UEFI 2.3 incompatibility. Some additions occurred in the PEI foundation for the latest evolution in buses, such as PCI Express+. In all of these cases, the revisions of the SMM, PEI, and DXE service tables were adjusted to ease migration of any SMM drivers, DXE drivers, and PEI module (PEIM) sources to PI. In the case of the firmware file system and volumes, the headers were expanded to comprehend larger file and alternate file system encodings, respectively. Unlike the case for SMM drivers, PEIMs, and DXE drivers, these present a new binary encoding that isn't compatible with a pure Framework implementation.

The notable aspect of the PI is the participation of the various members of the UEFI Forum, which will be described below. These participants represent the consumers and producers of PI technology. The ultimate consumer of a PI component is the vendor shipping a system board, including multinational companies such as Apple, Dell, HP, IBM, Lenovo, and many others. The producers of PI components include generic infrastructure producers such as the independent BIOS vendors (IBVs) like AMI, Insyde, Phoenix, and others. And finally, the vendors producing chipsets, CPUs, and other hardware devices like AMD, ARM, and Intel would produce drivers for their respective hardware. The IBVs and the OEMs would use the silicon drivers, for example. If it were not for this business-to-business transaction, the discoverable binary interfaces and separate executable modules (such as PEIMs and DXE drivers) would not be of interest. This is especially true since publishing GUID-based APIs, marshaling interfaces, discovering and dispatching code, and so on take some overhead in system board ROM storage and boot time. Given that there's never enough ROM space, and also in light of the customer requirements for boot-time such as the need to be "instantly on," this overhead must be balanced by the business value of PI module enabling. If only one vendor had access to all of the source and intellectual property to construct a platform, a statically bound implementation would be more efficient, for example. But in the twenty-first century with the various hardware and software participants in the computing industry, software technology such as PI is key to getting business done in light of the ever-shrinking resource and time-to-market constraints facing all of the UEFI forum members.

There is a large body of Framework-based source-code implementations, such as those derived or dependent upon EDK I (EFI Developer Kit version 1), which can be found on http://www.tianocore.org. These software artifacts can be recompiled into a UEFI 2.3, PI 1.2-compliant core, such as UDK2010 (the UEFI Developer Kit revision 2010), via the EDK Compatibility Package (ECP). For new development, though, the recommendation is to build native PI 1.2, UEFI 2.3 modules in the UDK2010 since these are the specifications against which long-term silicon enabling and operating system support will occur, respectively.

Terminology

The following list provides a quick overview of some of the terms that may be encountered later in the article and have existed in the industry associated with the BIOS standardization efforts.

- UEFI Forum. The industry body which produces UEFI, Platform Initialization (PI), and other specifications.
- UEFI Specification. The firmware-OS interface specification.
- EDK. The EFI Development Kit, an open sourced project that provides a basic implementation of UEFI, Framework, and other industry standards. It, is not however, a complete BIOS solution. An example of this can be found at http://www.tianocore.org.
- UDK. The UEFI Development Kit is the second generation of the EDK (EDK II), which has added a variety of codebase related capabilities and enhancements. The inaugural UDK is UDK2010, with the number designating the instance of the release.
- Framework. A deprecated term for a set of specifications that define interfaces and how various platform components work together. What this term referred to is now effectively replaced by the PI specifications.
- Tiano. An obsolete codename for an Intel codebase that implemented the Framework specifications.

A Short History of EFI

The Extensible Firmware interface (EFI) project was developed by Intel, with the initial specification released in 1999. At the time, it was designed as the means by which to boot Itanium-based systems. The original proposal for booting Itanium was the SAL (System Architectural Layer) SAL_PROC interface, with an encapsulation of the PC/AT BIOS registers as the arguments and parameters. Specifically, the means to access the disk in the SAL_PROC proposal was **"SAL_PROC (0x13, 0x2, ...)"**, which is aligned with the PC/AT conventional BIOS call of **"int13h."**

Given the opportunity to clean up the boot interface, various proposals were provided. These included but were not limited to Open Firmware and Advanced RISC Computing (ARC). Ultimately, though, EFI prevailed and its architecture-neutral interface was adopted.

The initial EFI specification included both an Itanium and IA-32 binding. EFI evolved from the EFI 1.02 interface into EFI1.10 in 2001. EFI1.10 introduced the EFI Driver model.

With the advent of 64-bit computing on IA-32 (for example, x64) and the industry's need to have a commonly owned specification, the UEFI 2.0 specification appeared in 2005. UEFI 2.0 is largely the same as EFI 1.0, but also included the modular networking stack APIs for IPv4 and the x64 binding. In Figure 3, we illustrate the evolution of the BIOS from its legacy days through 2010.

[Click image to view at full size]

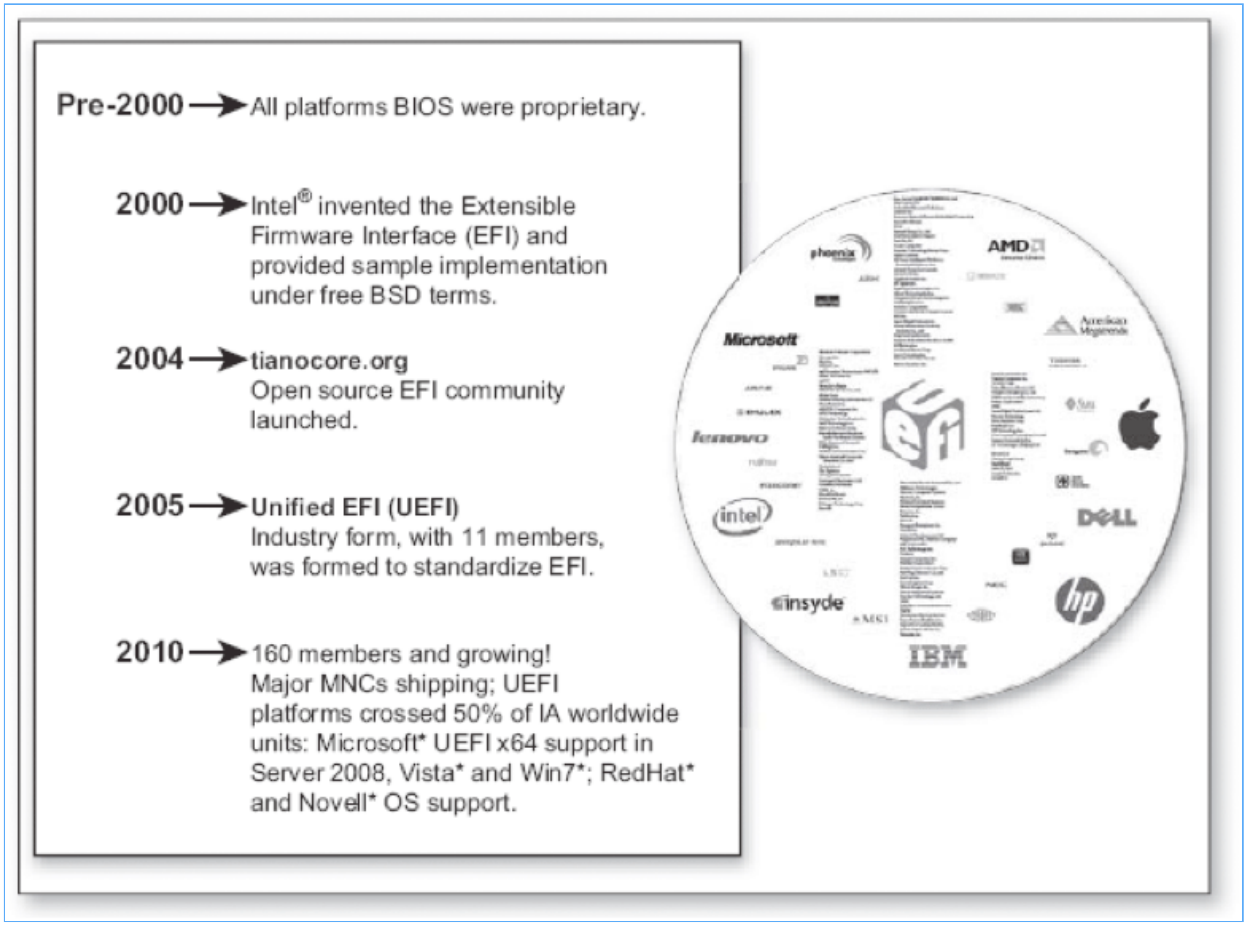

Figure 3: BIOS Evolution Timeline.

EFI Becomes UEFI -- The UEFI Forum

Regarding the UEFI Forum, there are various aspects to how it manages both the UEFI and PI specifications. Specifically, the UEFI forum is responsible for creating the UEFI and PI specifications.

When the UEFI Forum first formed, a variety of factors and steps were part of the creation process of the first specification:

- The UEFI forum stakeholders agree on EFI direction
- Industry commitment drives need for broader governance on specification
- Intel and Microsoft contribute seed material for updated specification
- EFI 1.10 components provide starting drafts
- Intel agrees to contribute EFI test suite As this had established the framework of the specification material that was produced which the industry used, the forum itself was formed with several thoughts in mind:

- The UEFI Forum is established as a Washington non-profit Corporation
  - Develops, promotes and manages evolution of Unified EFI Specification
  - Continue to drive low barrier for adoption
- The Promoter members for the UEFI forum are:
  - AMD, AMI, Apple, Dell, HP, IBM, Insyde, Intel, Lenovo, Microsoft, Phoenix
- The UEFI Forum has a form of tiered Membership:
  - Promoters, Contributors, and Adopters
  - More information on the membership tiers can be found at http://www.uefi.org
- The UEFI Forum has several work groups:
  - Figure 4 illustrates the basic makeup of the forum and the corresponding roles.

[Click image to view at full size]

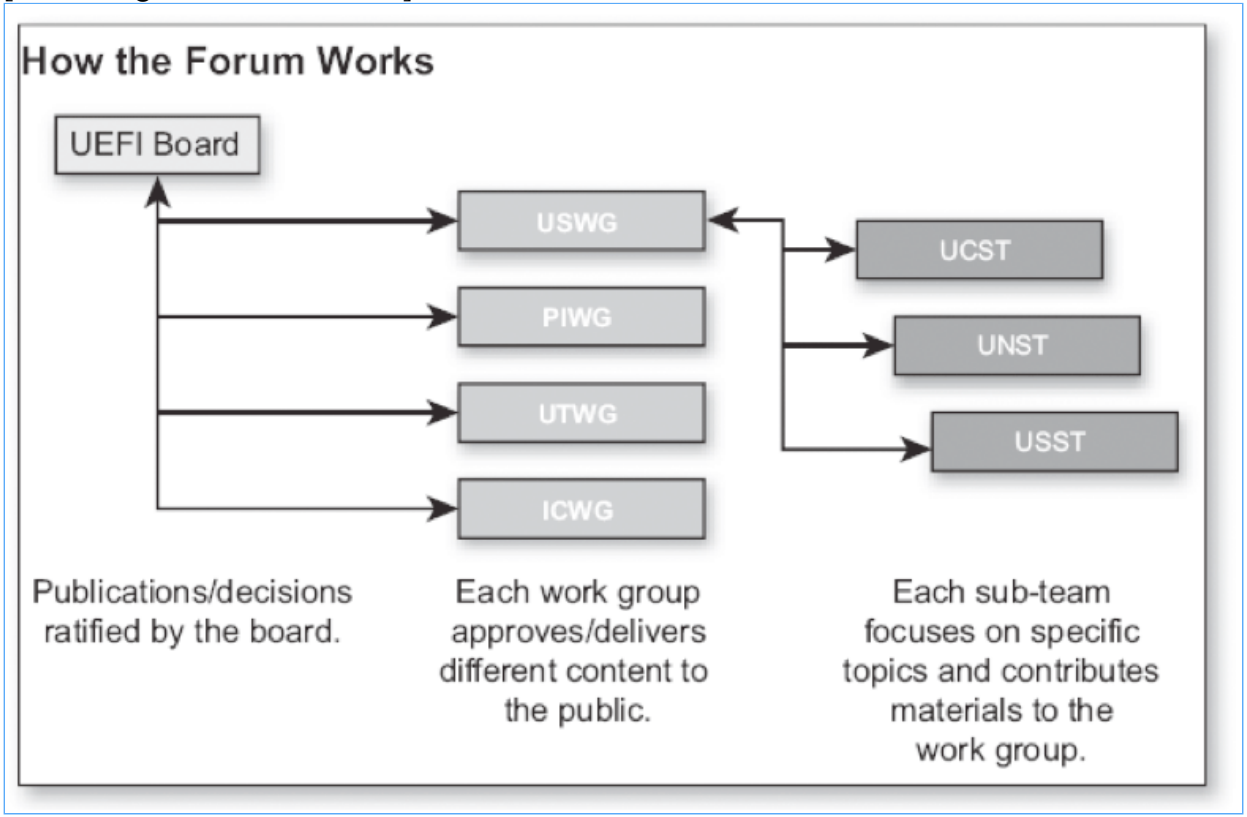

Figure 4: Forum group hierarchy.

- Sub-teams are created in the main owning workgroup when a topic of sufficient depth requires a lot of discussion with interested parties or experts in a particular domain. These teams are collaborations amongst many companies who are responsible for addressing the topic in question and bringing back to the workgroup either a response or material for purposes of inclusion in the main working specification. Some examples of sub-teams that have been created are as follows
  - UCS/UEFI Configuration Sub-team

    - Chaired by Michael Rothman (Intel)
    - Responsible for all configuration related material and the team has been responsible for the creation of the UEFI configuration infrastructure commonly known as HII, which is in the UEFI Specification.
- UNST/UEFI Networking Sub-team
    - Chaired by Vincent Zimmer (Intel)
    - Responsible for all network related material and the team has been responsible for the update/inclusion of the network related material in the UEFI specification, most notably the IPv6 network infrastructure.
- USST/UEFI Security Sub-team
    - Chaired by Tim Lewis (Phoenix)
    - Responsible for all security related material and the team has been responsible for the added security infrastructure in the UEFI specification.

PIWG and USWG

The Platform Initialization Working Group (PIWG) is the portion of the UEFI forum that defines the various specifications in the PI corpus. The UEFI Specification Working Group (USWG) is the group that evolves the main UEFI specification. Figure 5 illustrates the layers of the platform and what the scope that the USWG and PIWG cover.

[Click image to view at full size]

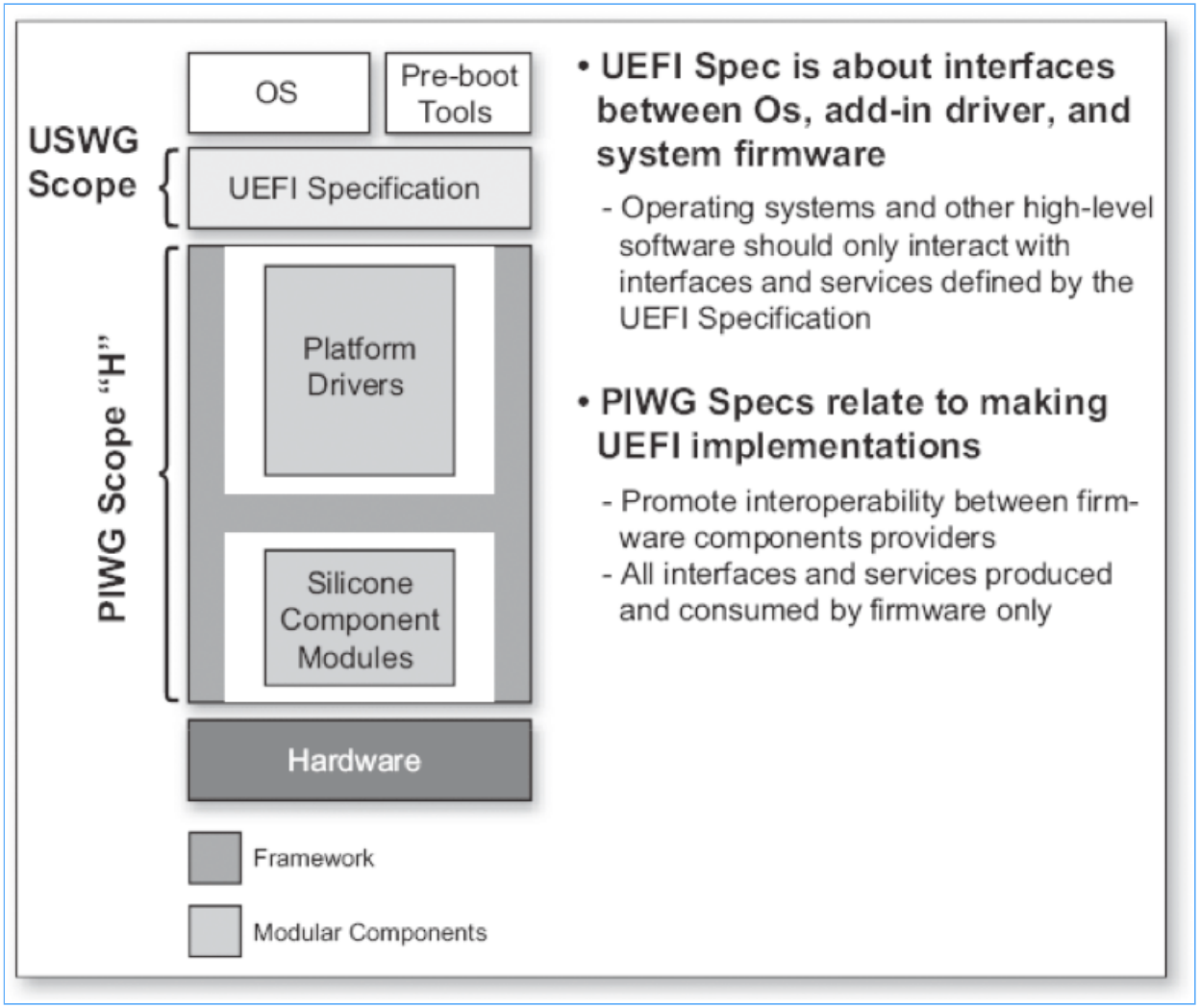

Figure 5: PI/UEFI layering.

Over time, these specifications have evolved. Below we enumerate the recent history of specifications and the work associated with each:

- UEFI 2.1
  - Roughly one year of Specification work
    - Builds on UEFI 2.0
  - New content area highlights:
    - Human Interface Infrastructure
    - Hardware Error Record Support
    - Authenticated Variable Support
    - Simple Text Input Extensions

        - Absolute Pointer Support
    - UEFI 2.2
        - Follow-on material from existing 2.1 content
            - Backlog that needed more gestation time
        - Security/Integrity related enhancements
            - Provide service interfaces for UEFI drivers that want to operate with high integrity implementations of UEFI
        - Human Interface Infrastructure enhancements
            - Further enhancements pending to help interaction/configuration of platforms with standards-based methodologies.
            - Networking
                - IPv6, PXE+, IPsec
            - Various other subject areas possible
                - More boot devices, more authentication support, more networking updates, etc.
    - UEFI2.3
        - ARM binding
        - Firmware management protocol

To complement the layering picture in Figure 5, Figure 6 shows how the PI elements evolve into the UEFI. The left half of the diagram with SEC, PEI, and DXE are described by the PI specifications. BDS, UEFI+OS Loader handshake, and RT are the province of the UEFI specification.

[Click image to view at full size]

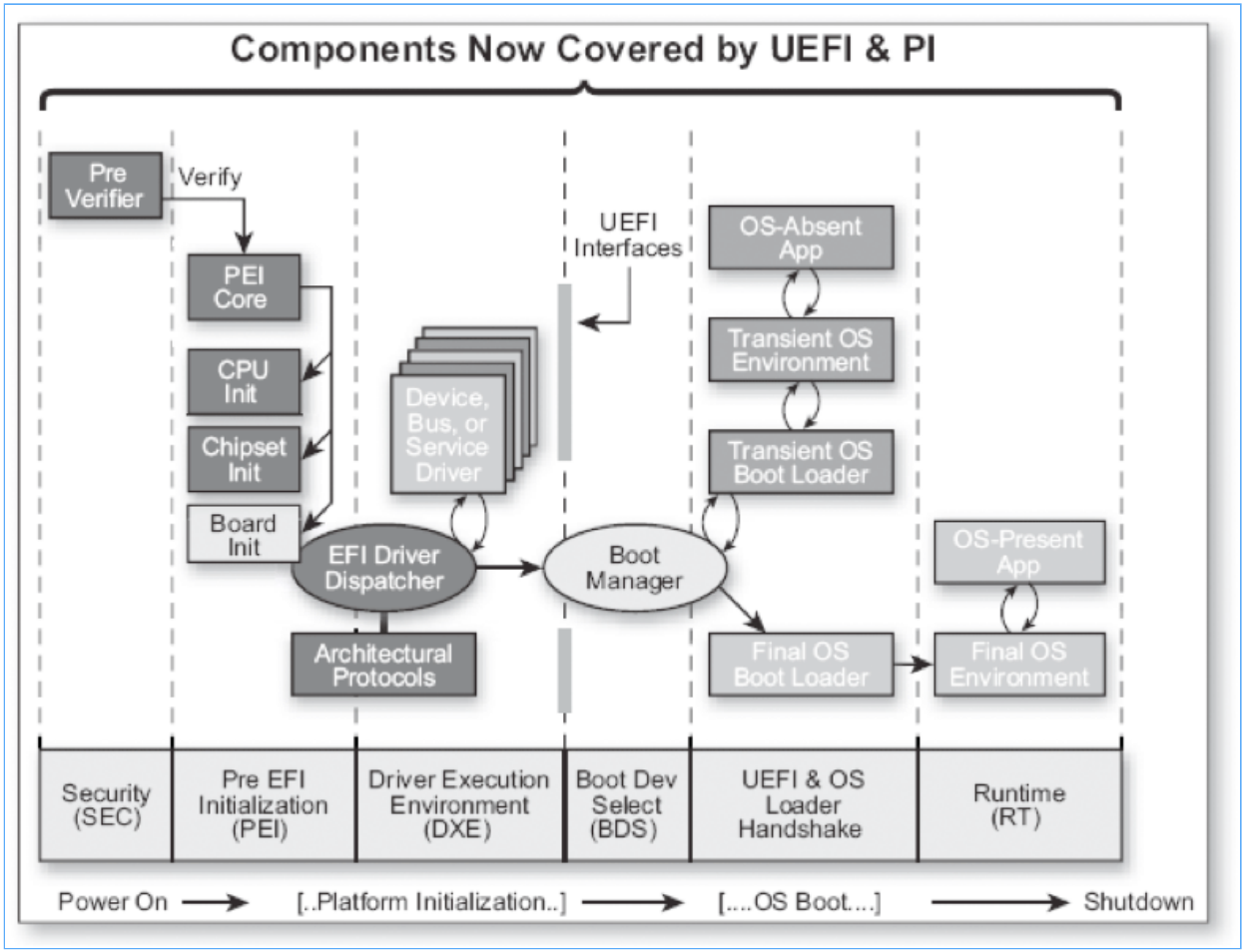

Figure 6: Where PI and Framework Fit into the Platform Boot Flow.

In addition, as time has elapsed, the specifications have evolved. Figure 7 is a timeline for the specifications and the implementations associated with them.

[Click image to view at full size]

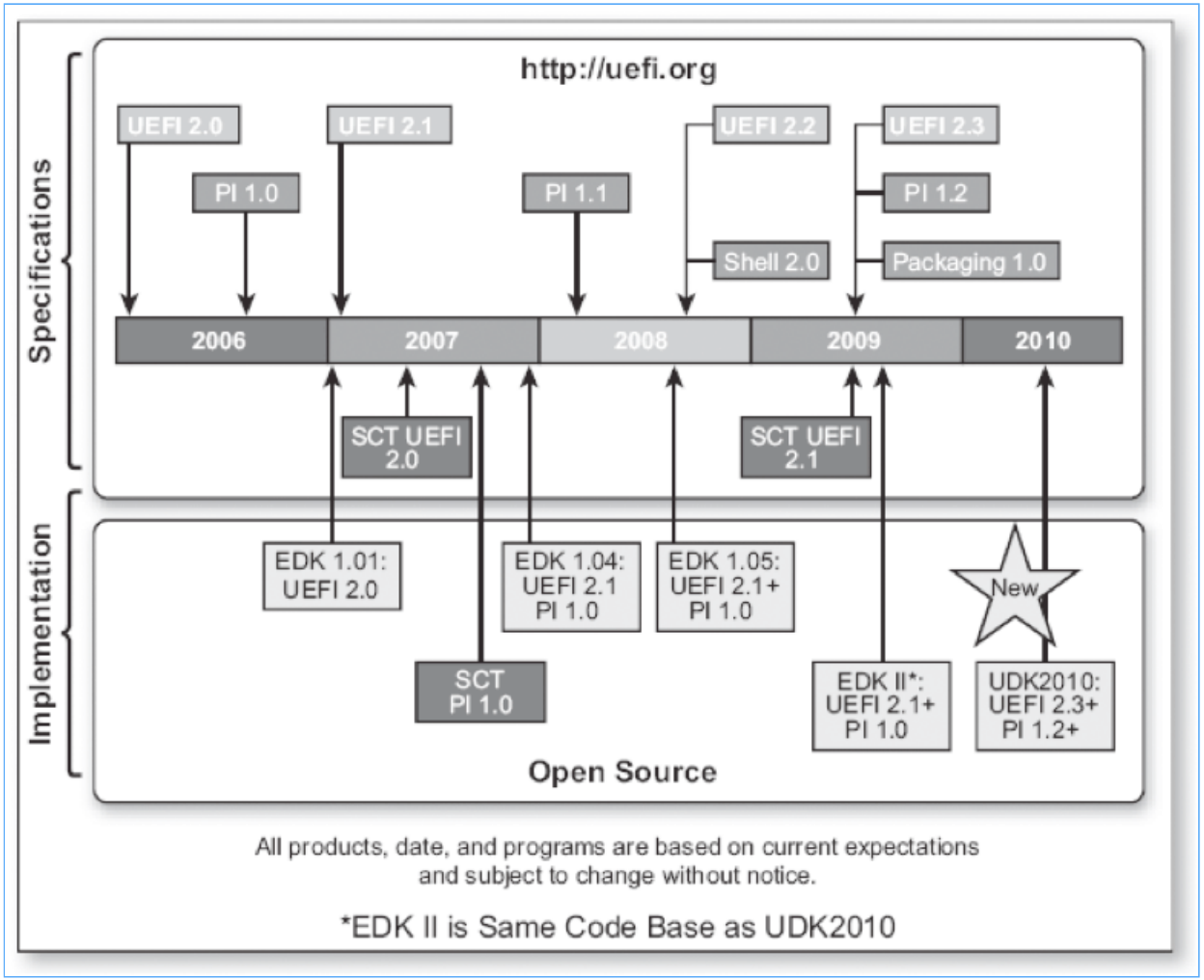

Figure 7: Specification and Codebase Timeline.

Platform Trust/Security

Recall that PI allowed for business-to-business engagements between component providers and system builders. UEFI, on the other hand, has a broader set of participants. These include the operating system vendors that built the OS installers and UEFI-based runtimes; BIOS vendors who provide UEFI implementations; platform manufacturers, such as multi-national corporations who ship UEFI-compliant boards; independent software vendors who create UEFI applications and diagnostics; independent hardware vendors who create drivers for their adapter cards; and platform owners, whether a home PC user or corporate IT, who must administer the UEFI-based system.

PI differs from UEFI in the sense that the PI components are delivered under the authority of the platform manufacturer and are not typically extensible by third parties. UEFI, on the other hand, has a mutable file system partition, boot variables, a driver load list, support of discoverable

option ROMs in host-bus adapters (HBAs), and so on. As such, PI and UEFI offer different issues with respect to security. In general, the security dimension of the respective domains include the following: PI must ensure that the PI elements are only updateable by the platform manufacturer, recovery, and PI is a secure implementation of UEFI features, including security; UEFI provides infrastructure to authenticate the user, validate the source and integrity of UEFI executables, network authentication and transport security, audit (including hardware-based measured boot), and administrative controls across UEFI policy objects, including write-protected UEFI variables.

A fusion of these security elements in a PI implementation is shown in Figure 8.

[Click image to view at full size]

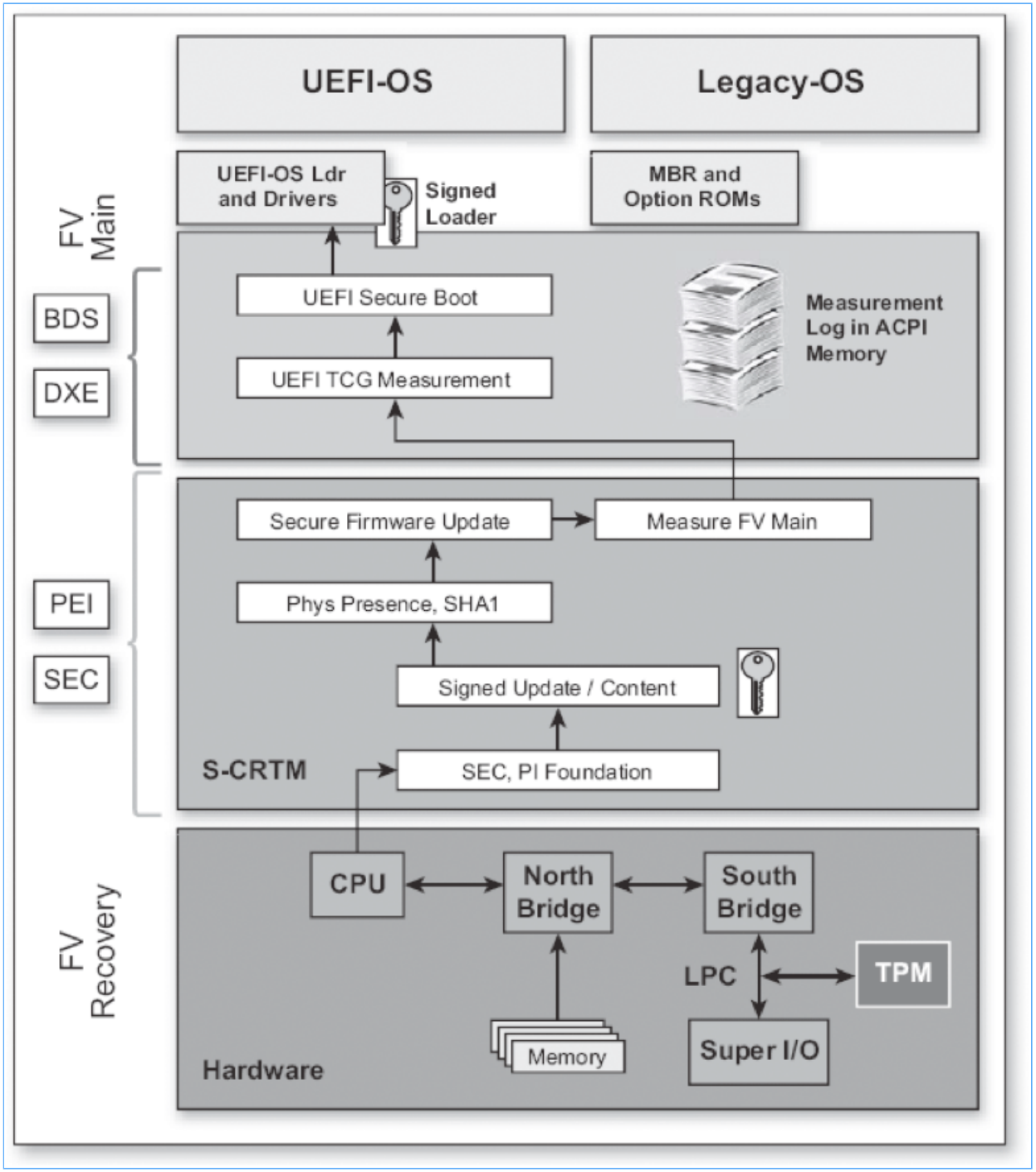

Figure 8: Trusted UEFI/PI stack.

Embedded Systems: The New Challenge

As the UEFI took off and became pervasive, a new challenge has been taking shape in the form of the PC platform evolution to take on the embedded devices, more specifically the consumer electronic devices, with a completely different set of requirements driven by user experience factors like instant power-on for various embedded operating systems. Many of these operating systems required customized firmware with OS-specific firmware interfaces and did not fit well into the PC firmware eco-system model.

The challenge now is to make the embedded platform firmware have similar capabilities to the traditional model such as the being OS-agnostic, being scalable across different platform hardware, and being able to lessen the development time to port and to leverage the UEFI standards.

Does the Boot Process Differ?

Figure 9 indicates that between the normal boot and an optimized boot, there are no design differences from a UEFI architecture point of view. Optimizing a platform's performance does not mean that one has to violate any of the design specifications. It should also be noted that to comply with UEFI, one does not need to encompass all of the standard PC architecture, but instead the design can limit itself to the components that are necessary for the initialization of the platform itself. Chapter 2 in the UEFI 2.3 specification does enumerate the various components and conditions that comprise UEFI compliance.

[Click image to view at full size]

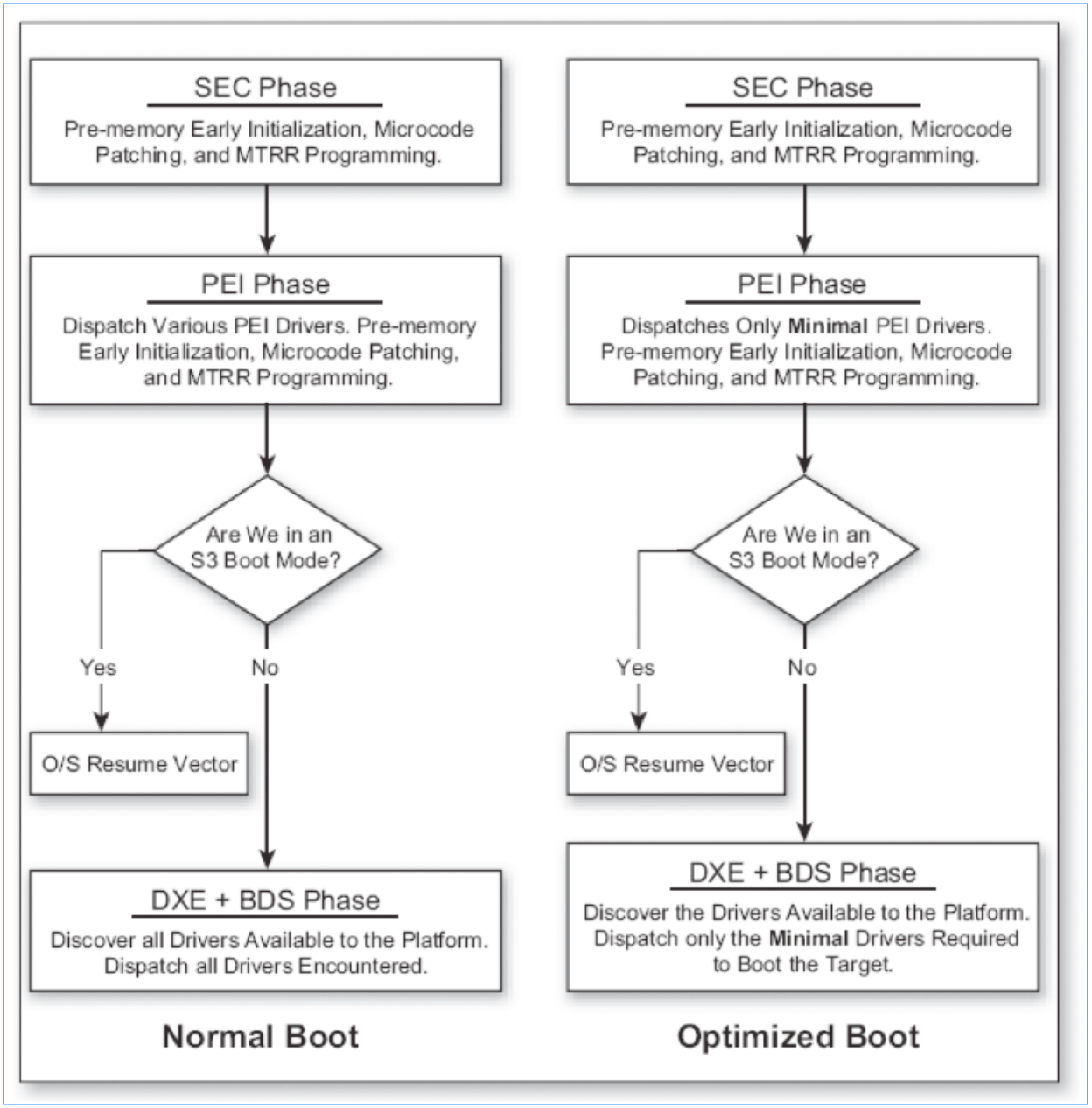

Figure 9: Architectural Boot Flow Comparison.

Conclusion

We have provided some rationale in this article for the changes from *Beyond BIOS: Implementing the Unified Extensible Firmware Interface with Intel's Framework* to *Beyond BIOS: Developing with the Unified Extensible Firmware Interface*. These elements include the industry members' ownership and governance of the UEFI specification. Beyond this sea change, we've described the migration of the Framework specifications to PI specification and the evolution of PI over the former Framework feature set; the evolution of the UEFI specification; and some of the codebase technology.

For more information on implementing UEFI with Intel's Framework, please refer to our book *Beyond BIOS: Developing with the Unified Extensible Firmware Interface, 2nd Edition* .

This article is based on material found in book Beyond BIOS: Developing with the Unified Extensible Firmware Interface, 2nd Edition *by Vincent Zimmer, Michael Rothman, and Suresh Marisetty. More on similar subjects may be found in the* Intel Technology Journal.