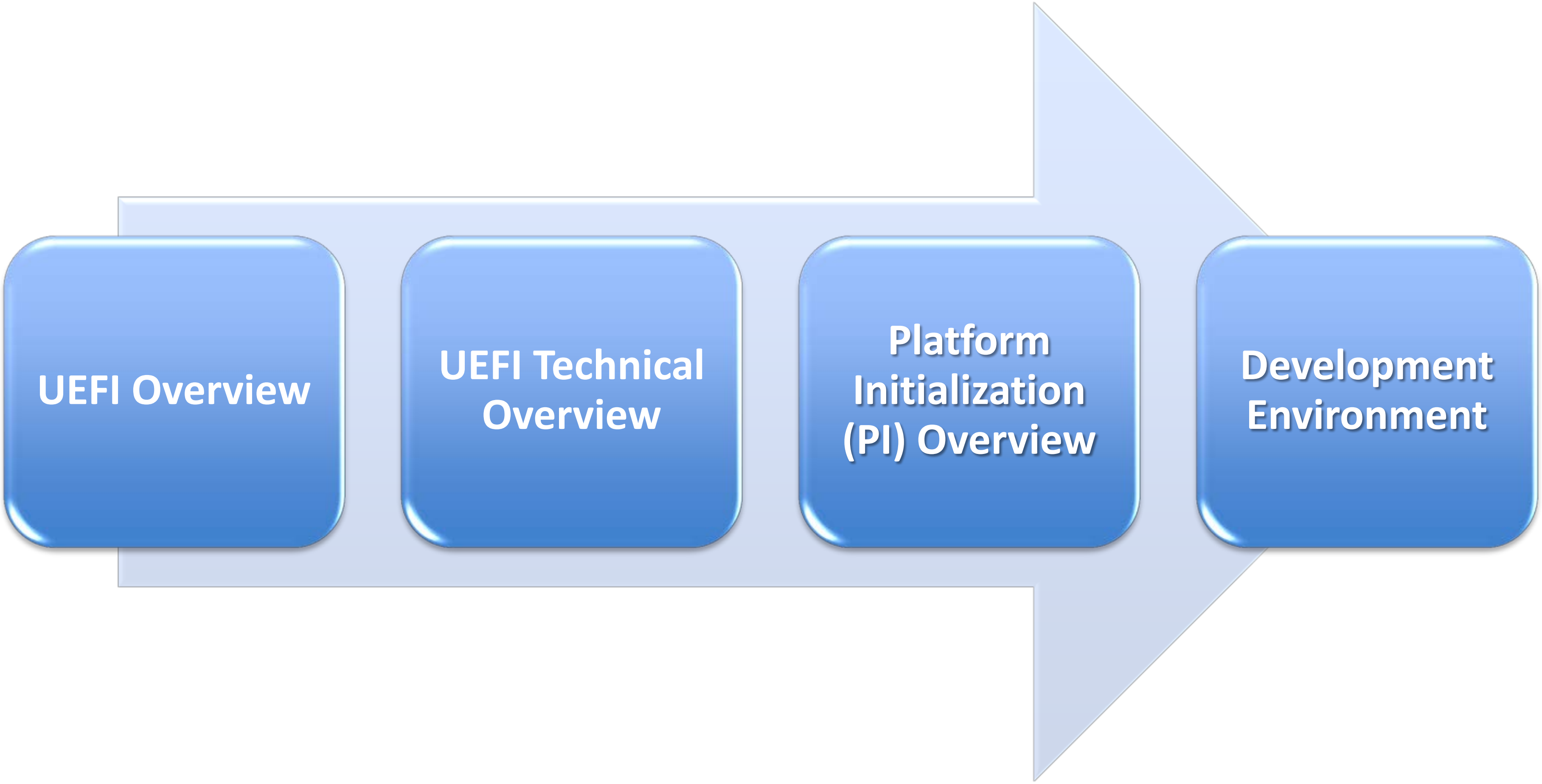
UEFI & EDK II Base Training

Unified Extensible Firmware Interface (UEFI) & Platform Initialization (PI) Overview

Intel Corporation
Software and Services Group

Agenda

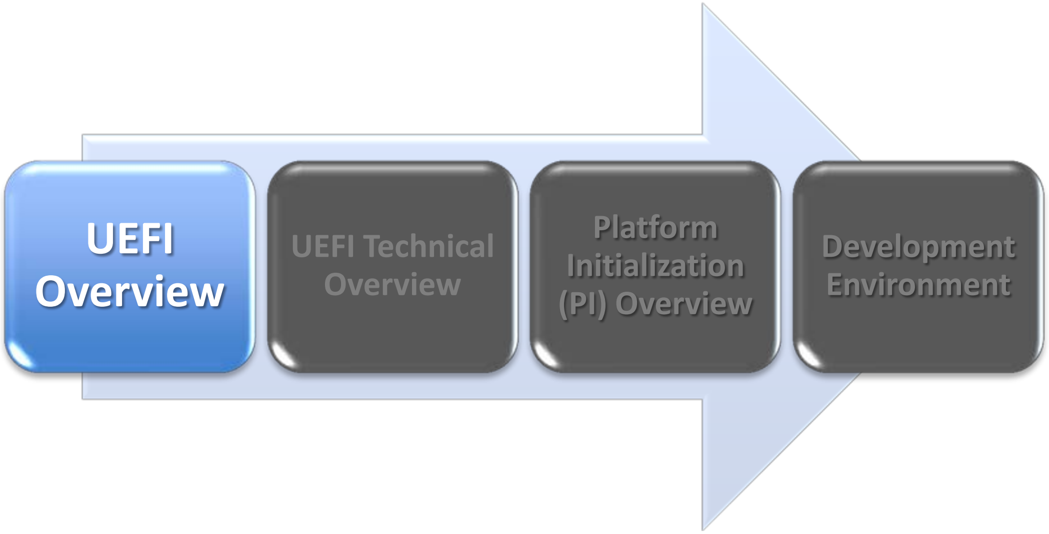

UEFI Overview

UEFI Technical Overview

Platform Initialization (PI) Overview

Development Environment

UEFI Overview

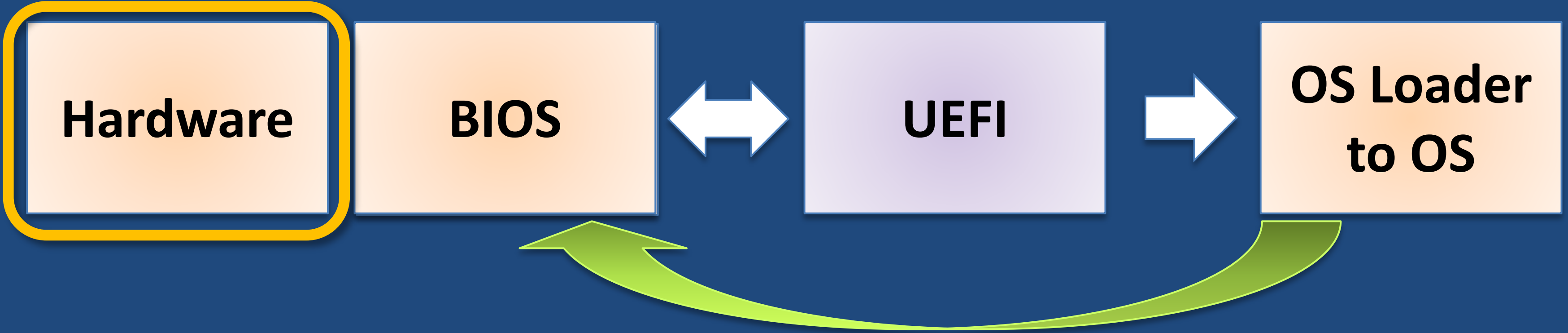


High Level Concept

EFI/UEFI Timeline

The UEFI Forum

UEFI's Role in the Booting Process



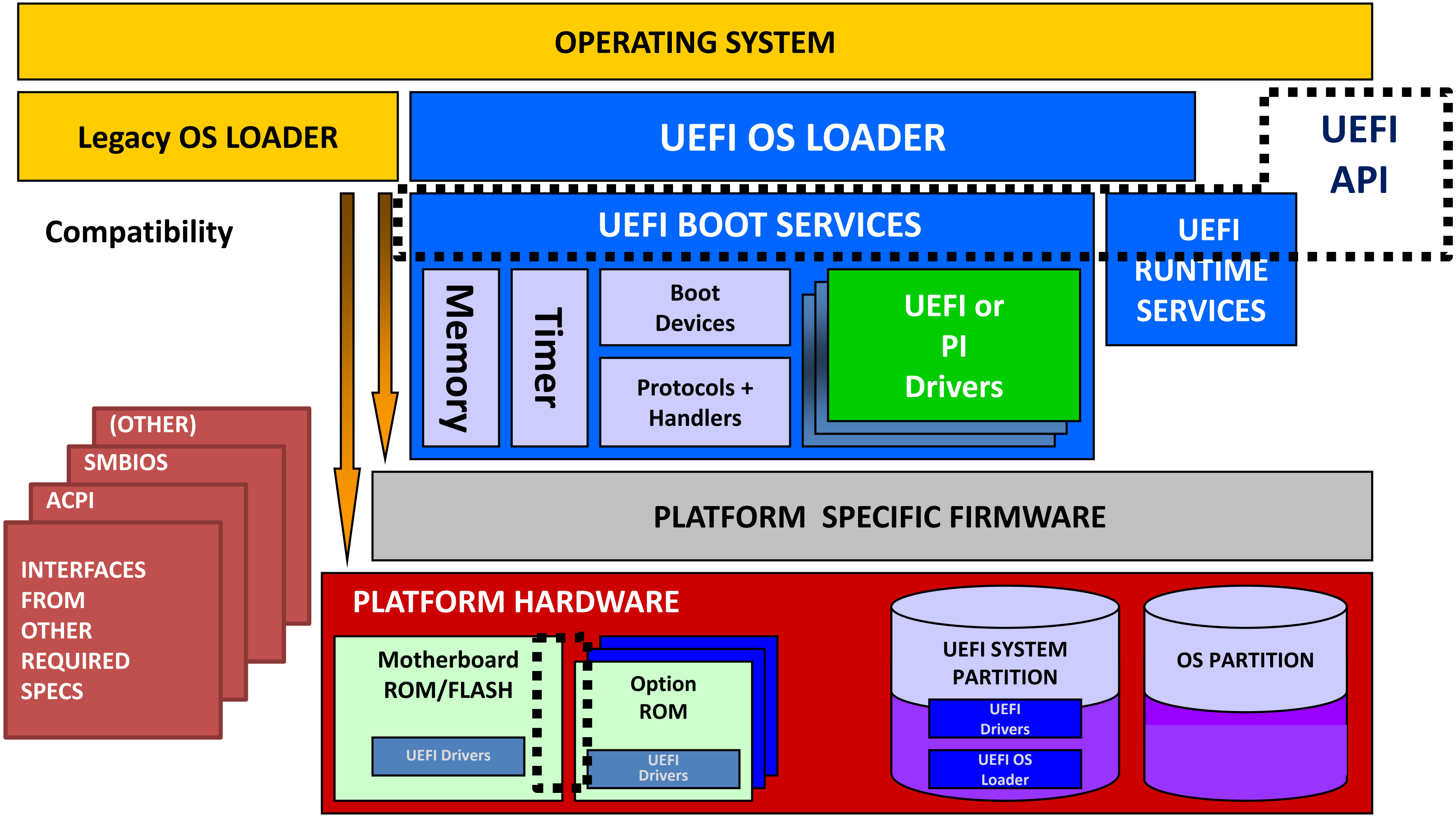
- Platform Motherboard

- Firmware Communicating to the Hardware

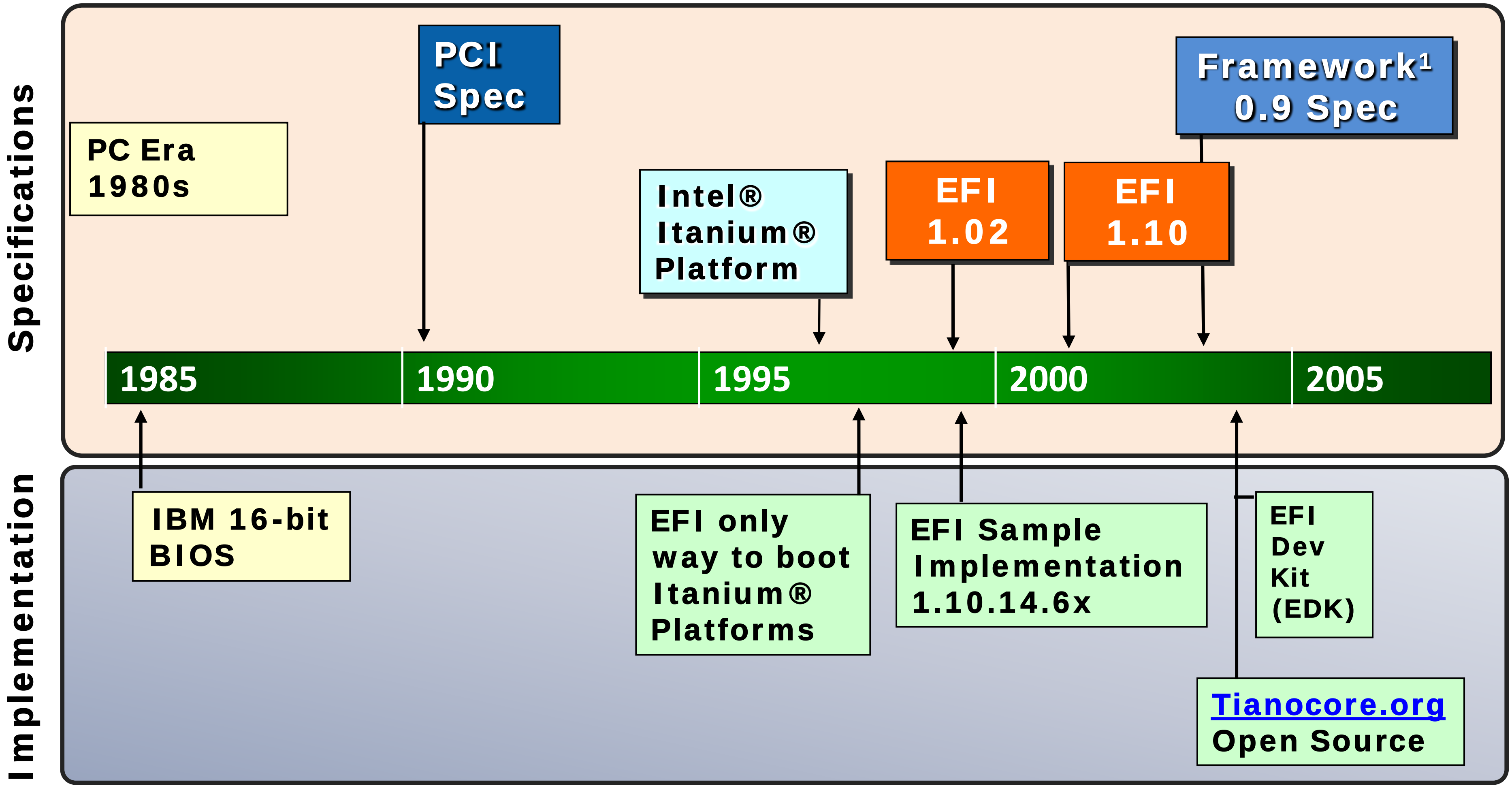
- Wrapper layer Communicating to BIOS

- Pre-Boot Layer executes UEFI Application - OS Loader
- UEFI aware OS interfaces the UEFI Layer
- Account for Legacy w/ UEFI CSM Module

High Level Concept



EFI / UEFI History Timeline

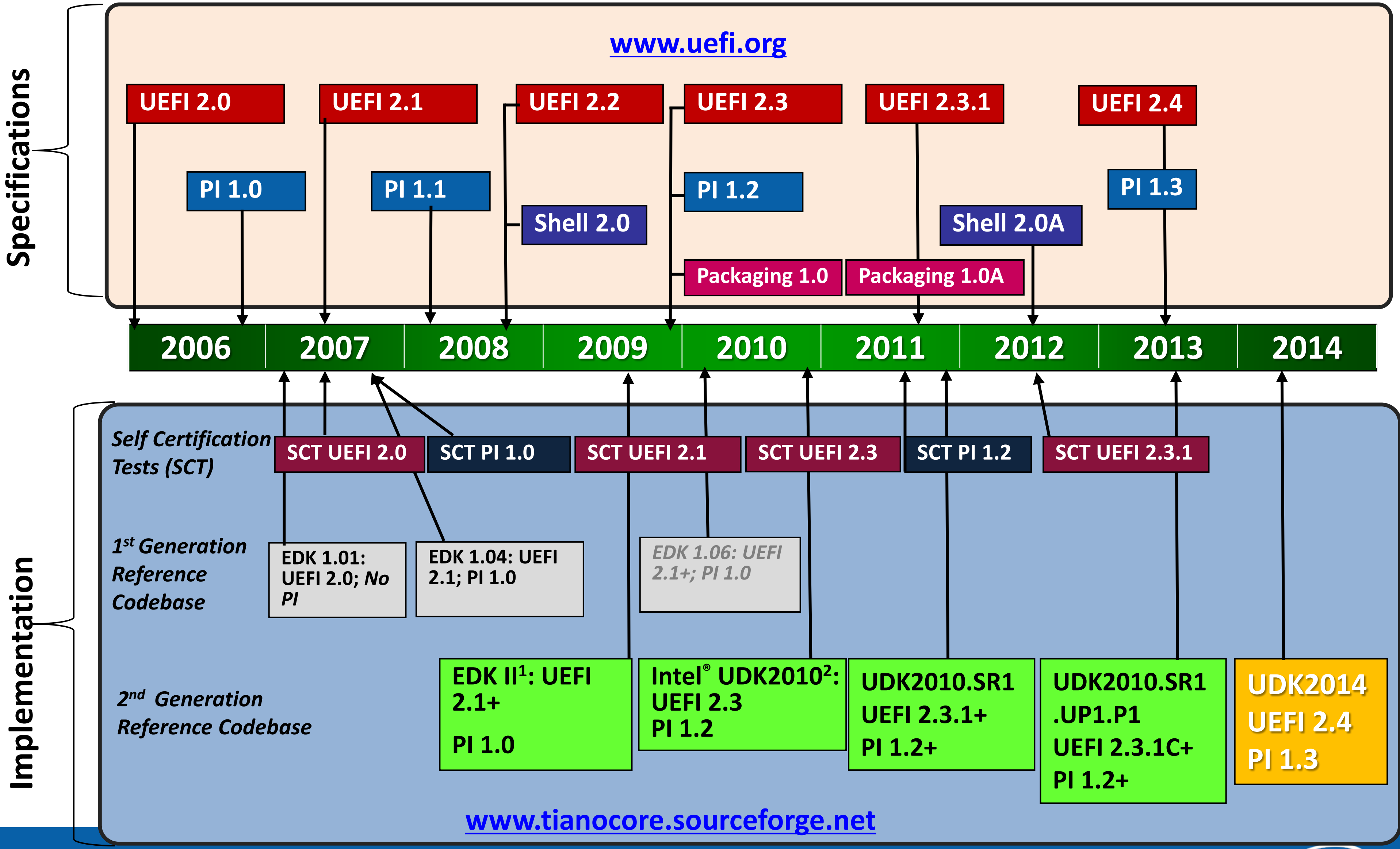


Open Source EFI Developer Kit (EDK) http://www.tianocore.Sourceforge.net

UEFI Specifications - http://www.uefi.org

[1] Intel ® Platform Innovation Framework for UEFI

UEFI Specification & Intel UEFI Reference Implementation Timeline



Specifications

www.uefi.org

UEFI 2.0

PI 1.0

UEFI 2.1

PI 1.1

UEFI 2.2

Shell 2.0

UEFI 2.3

PI 1.2

Packaging 1.0

UEFI 2.3.1

Packaging 1.0A

Shell 2.0A

UEFI 2.4

PI 1.3

2006 2007 2008 2009 2010 2011 2012 2013 2014

Implementation

Self Certification Tests (SCT)

SCT UEFI 2.0

SCT PI 1.0

SCT UEFI 2.1

SCT UEFI 2.3

SCT PI 1.2

SCT UEFI 2.3.1

1st Generation Reference Codebase

EDK 1.01: UEFI 2.0; *No PI*

EDK 1.04: UEFI 2.1; PI 1.0

EDK 1.06: UEFI 2.1+; PI 1.0

2nd Generation Reference Codebase

EDK II[1]: UEFI 2.1+ PI 1.0

Intel® UDK2010[2]: UEFI 2.3 PI 1.2

UDK2010.SR1 UEFI 2.3.1+ PI 1.2+

UDK2010.SR1 .UP1.P1 UEFI 2.3.1C+ PI 1.2+

UDK2014 UEFI 2.4 PI 1.3

www.tianocore.sourceforge.net

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

[1] **EDK II** is the main open source community codebase; same codebase as Intel UDK2010

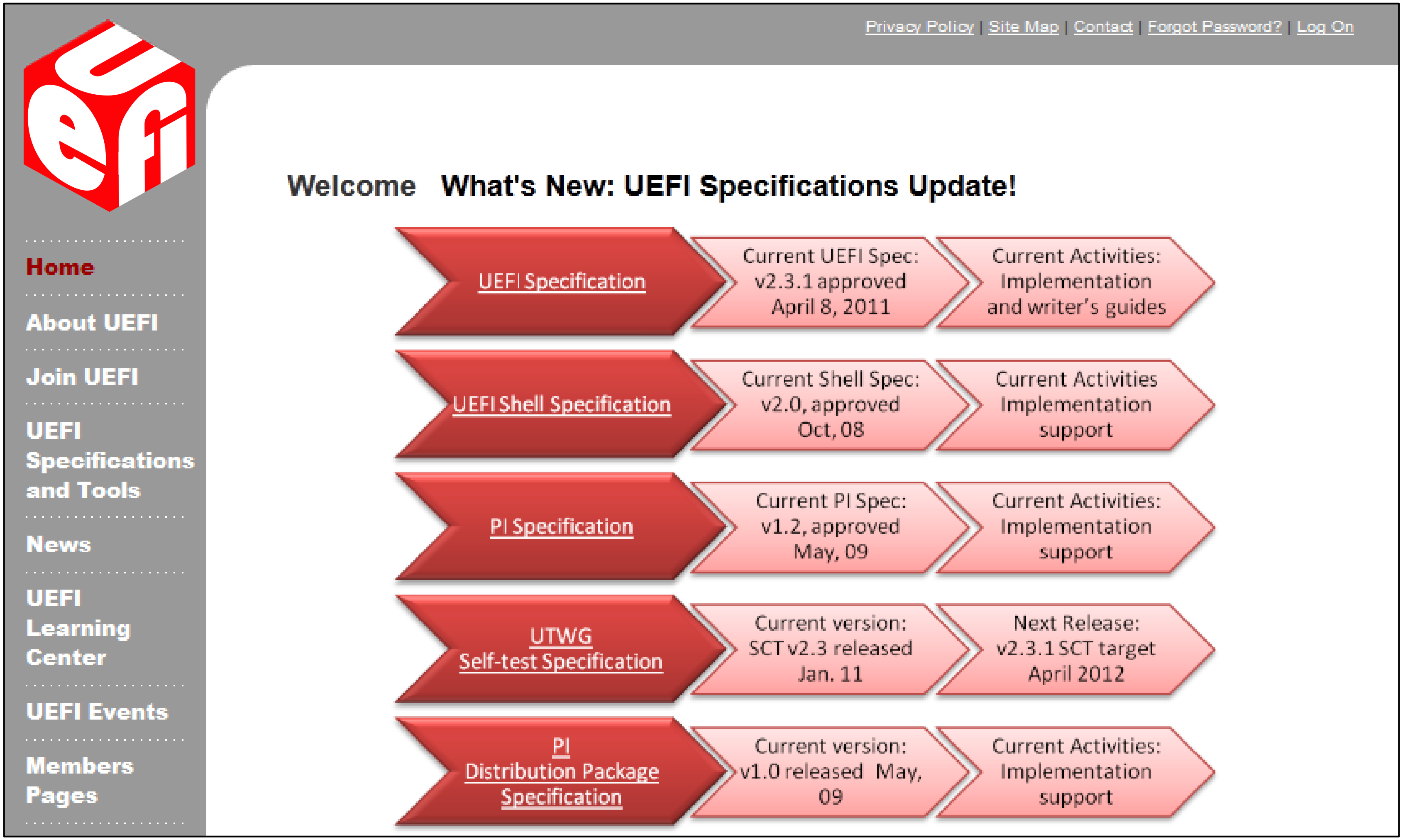
[2] **Intel® UEFI Development Kit 2010 (Intel® UDK2010)** releases are **Intel validated** snapshots of select packages from EDKII codebase

Unified EFI (UEFI) Forum – www.uefi.org

- Promoters
  - OEM: Dell, HP, IBM, Lenovo
  - IBVs: AMI, Insyde, Phoenix
  - AMD, Apple, Intel, Microsoft
- UEFI Specification
  - EFI 1.10 specification contributed to the forum by Intel and Microsoft to be used as a starting draft
  - UEFI 2.0-2.3.1 specifications have been released
  - UEFI Forum will evolve, extend, and add any new functionality as required

Purpose: Worldwide adoption and promotion of UEFI specifications

uefi.org



Privacy Policy | Site Map | Contact | Forgot Password? | Log On

- Home
- About UEFI
- Join UEFI
- UEFI Specifications and Tools
- News
- UEFI Learning Center
- UEFI Events
- Members Pages

Welcome What's New: UEFI Specifications Update!

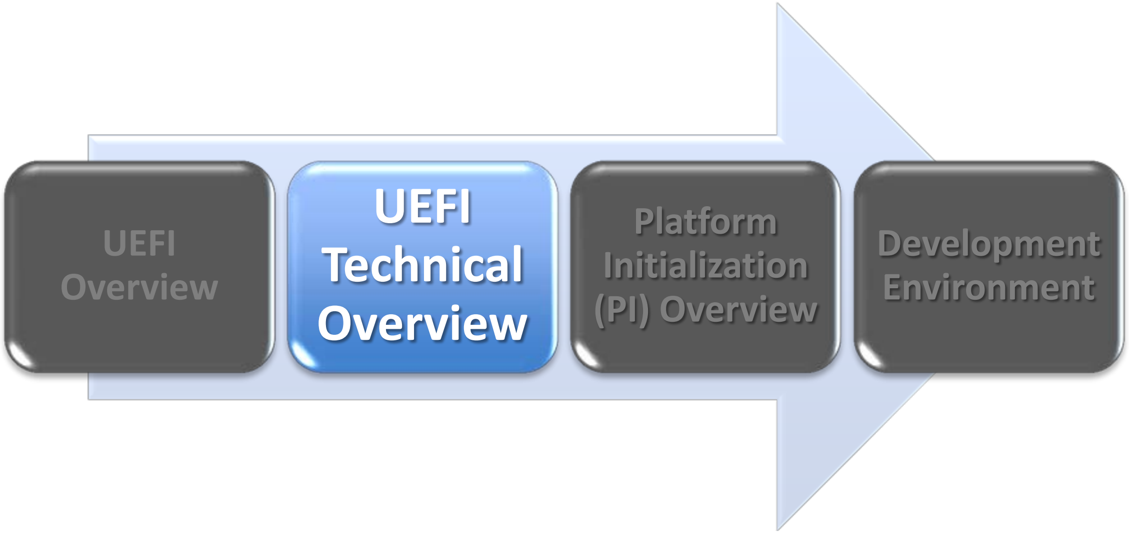
UEFI Specification	Current UEFI Spec: v2.3.1 approved April 8, 2011	Current Activities: Implementation and writer's guides
UEFI Shell Specification	Current Shell Spec: v2.0, approved Oct, 08	Current Activities Implementation support
PI Specification	Current PI Spec: v1.2, approved May, 09	Current Activities: Implementation support
UTWG Self-test Specification	Current version: SCT v2.3 released Jan. 11	Next Release: v2.3.1 SCT target April 2012
PI Distribution Package Specification	Current version: v1.0 released May, 09	Current Activities: Implementation support

Incremental UEFI Spec Version Features

Specification Version	Features
EFI 1.10 → UEFI 2.0	Industry Adoption, X64
UEFI 2.0 → UEFI 2.1	HII Features
UEFI 2.1 → UEFI 2.2	Network IPv6 and Security
UEFI 2.2 → UEFI 2.3	ARM, Firmware Management Protocol
UEFI 2.3 → UEFI 2.3.1	Secure Boot, USB 3.0

Details

UEFI Technical Overview



Boot Support

UEFI Terminology

UEFI Services

UEFI Driver Design

EFI Byte Code

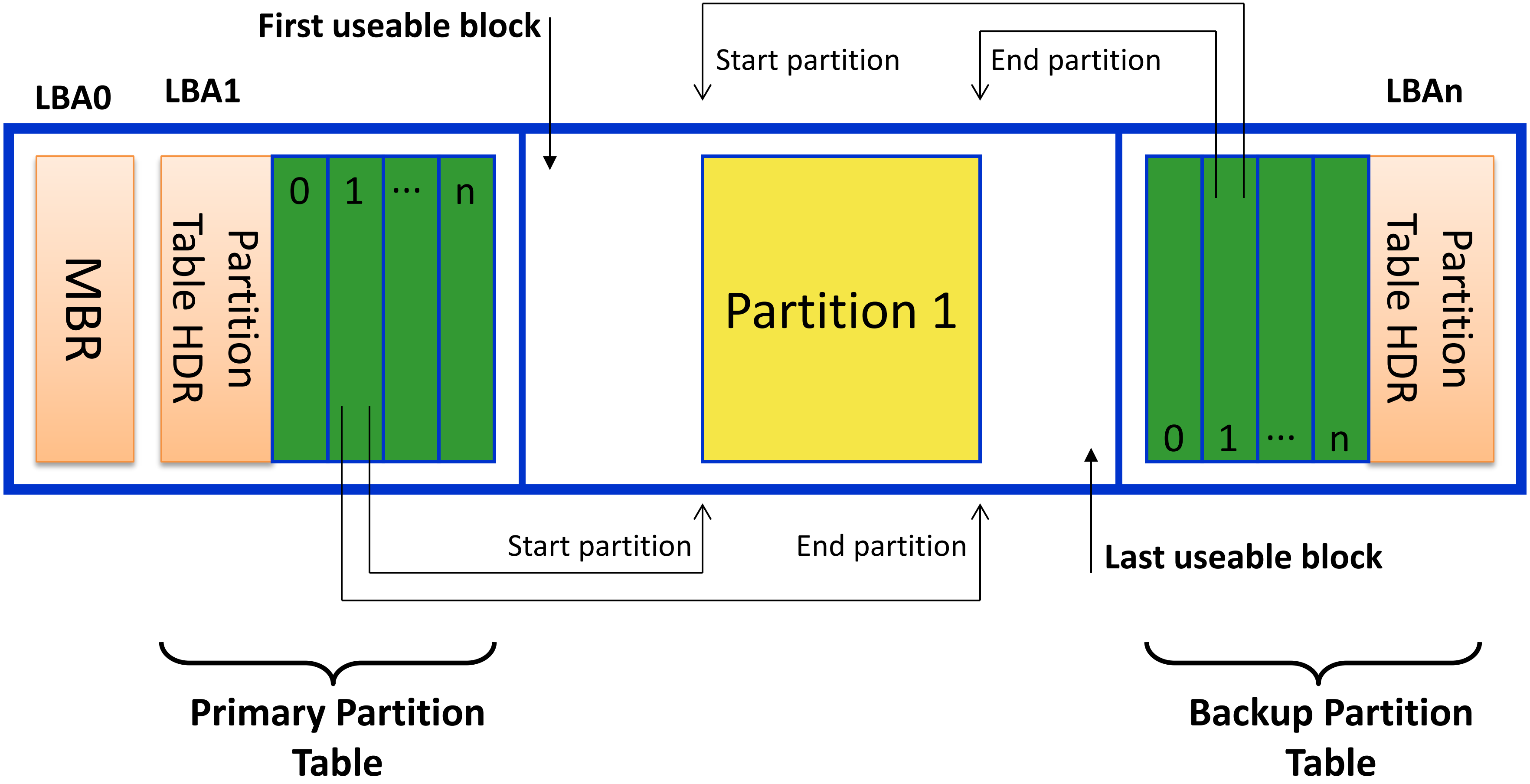
Boot Support – Device Types

- Hard disk
- Removable media
  - CD-ROM, DVD-ROM
    - El Torito 1.0 “No emulation”
  - Floppy, USB Storage, etc.
- Network
  - PXE BIOS support specification (Wire for Management)
  - iSCSI
- Future media via extensibility methods

Full Device Support

Expanded Capabilities Versus Legacy BIOS

GPT - New Partition Structure

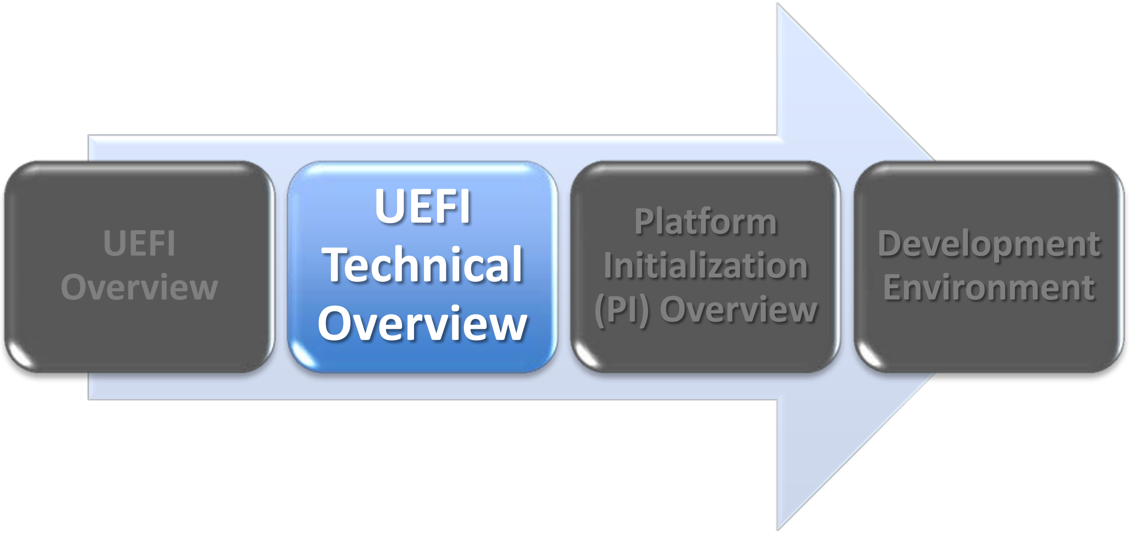


See Section 5 of the UEFI 2.3.1 Spec.

GPT Advantages over MBR Partition Table

- 64-bit LBA
  - No more 2.2TB limit
  - Up to 9.8 zettabytes
- Improved partitioning
  - Supports unlimited number of partitions
  - Uses a primary and backup table for redundancy
  - Defines a GUID for identifying each partition
  - Uses GUID & attributes to define partition type
- Uses version number and size fields for future expansion.
- Uses CRC32 fields for improved data integrity
- Each partition contains a 36 Unicode character human readable name.
- No MBR problems
  - No "magic code" must execute as part of booting

UEFI Technical Overview



Boot Support

UEFI Terminology

UEFI Services

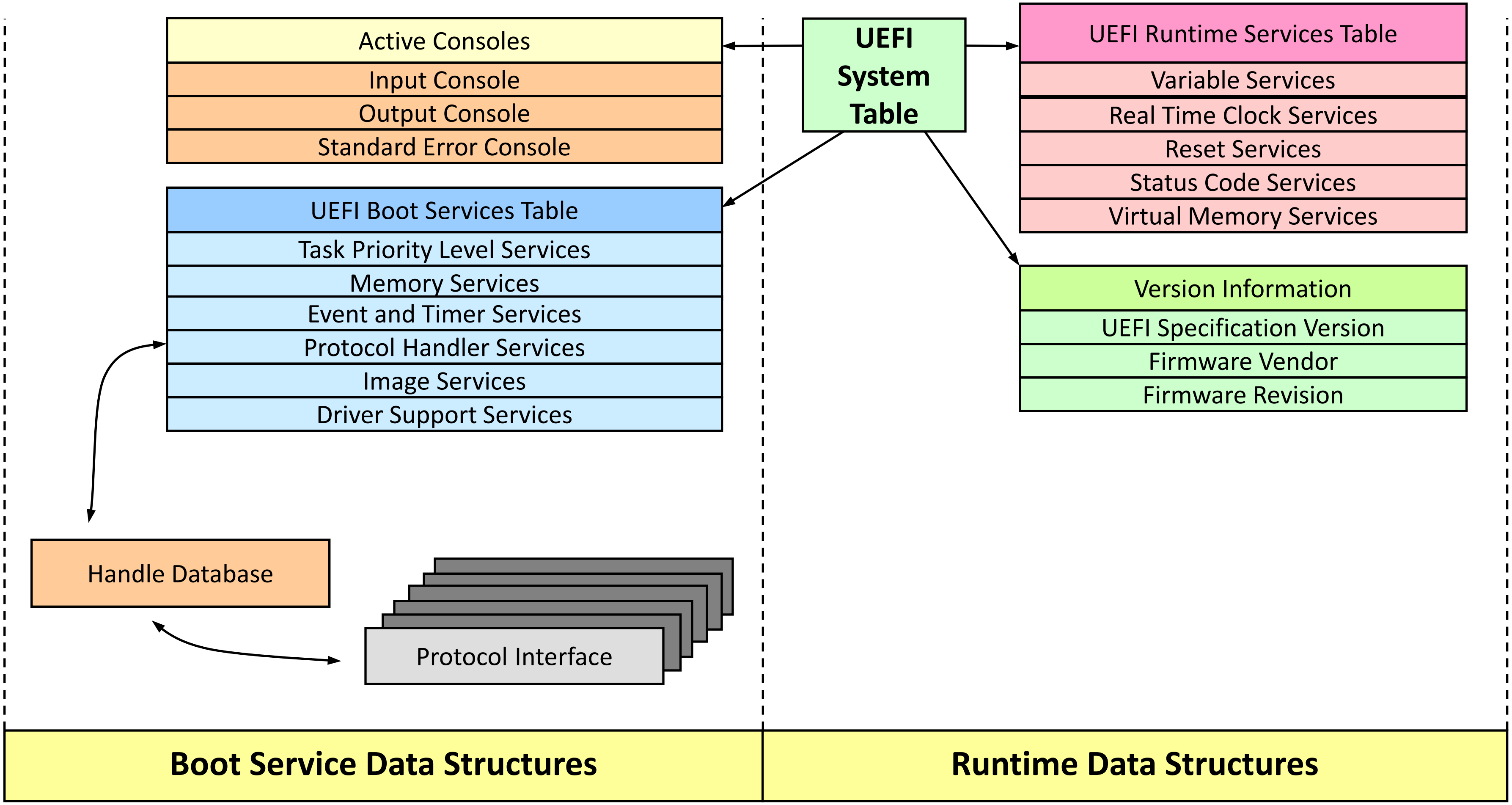
UEFI Driver Design

EFI Byte Code

UEFI Specification - Key Concepts

- **Objects** - manage system state, including I/O devices, memory, and events
- **UEFI System Table** - data structure with data information tables to interface with the systems
- **Handle Database and Protocols** - callable interfaces that are registered
- **UEFI Images** - the executable content format
- **Events** - the software can be signaled in response to some other activity
- **Device Paths** - a data structure that describes the hardware location of an entity

UEFI Data Structures: UEFI System Table



GUID

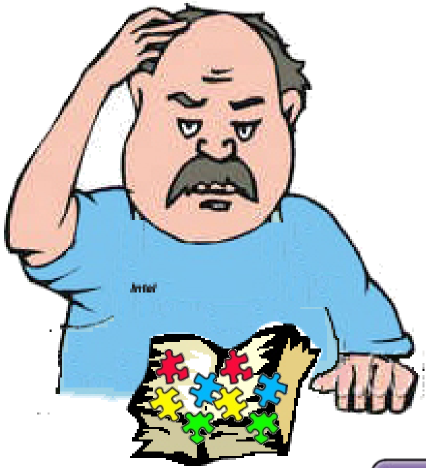
- Globally Unique Identifier
  - 128-bit quantity defined by Wired for Management (WfM) 2.0 specification **
- Used to identify protocols
  - 1:1 with interfaces
- Regulate extension mechanism
  - Documented in the spec
  - Added through drivers

Safe co-existence of 3rd party extensions

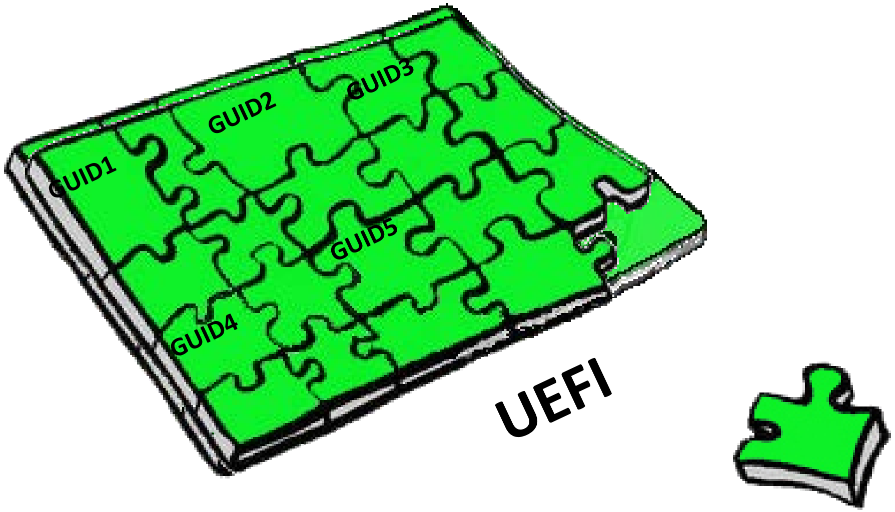
** http://www.intel.com/design/archives/wfm/index.htm

Legacy BIOS vs UEFI

Legacy BIOS
INT 10h
INT 13h
Chaining
INT 16h
INT 15h
?



Ralf Brown's Interrupt List

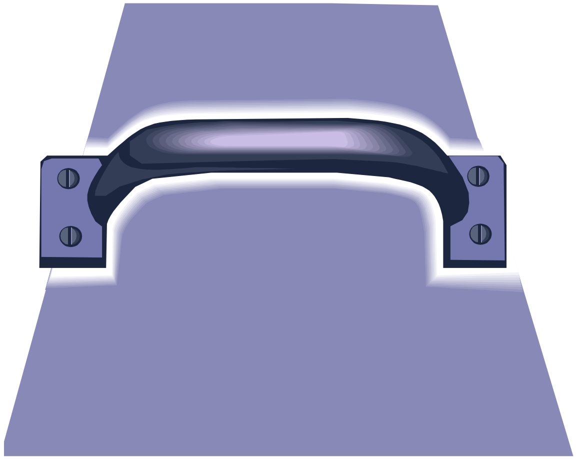


- UEFI
- GUID1 UEFI Specification
- GUID2 PI Specification
- GUID3 ODM defined
- GUID4 OEM defined
- GUID5 IBV defined

UEFI Means the Pieces all Fit and Work!

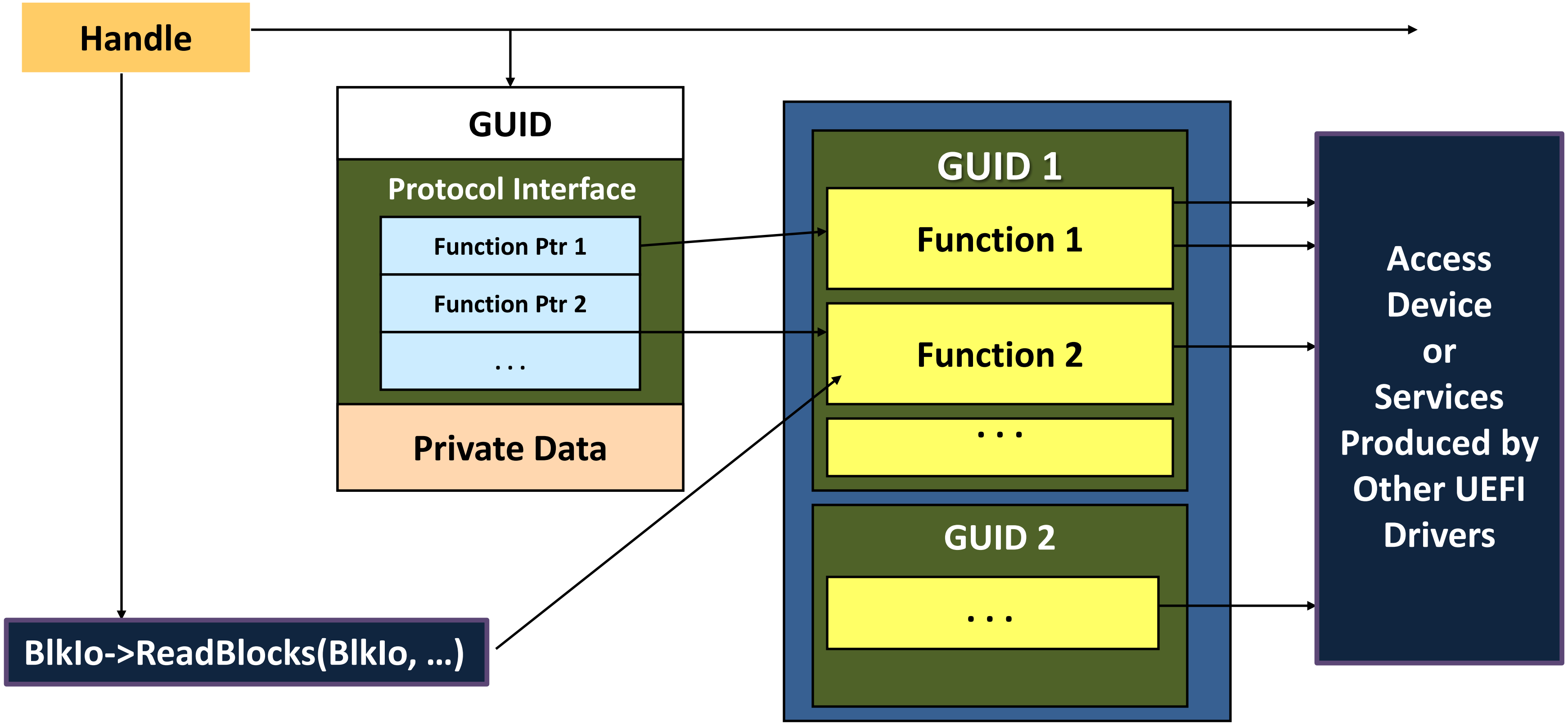
Handles

- All protocols have an associated handle
- Every device and executable image in UEFI has a handle protocol in the handle database
- Every boot device must have a device path protocol to describe it



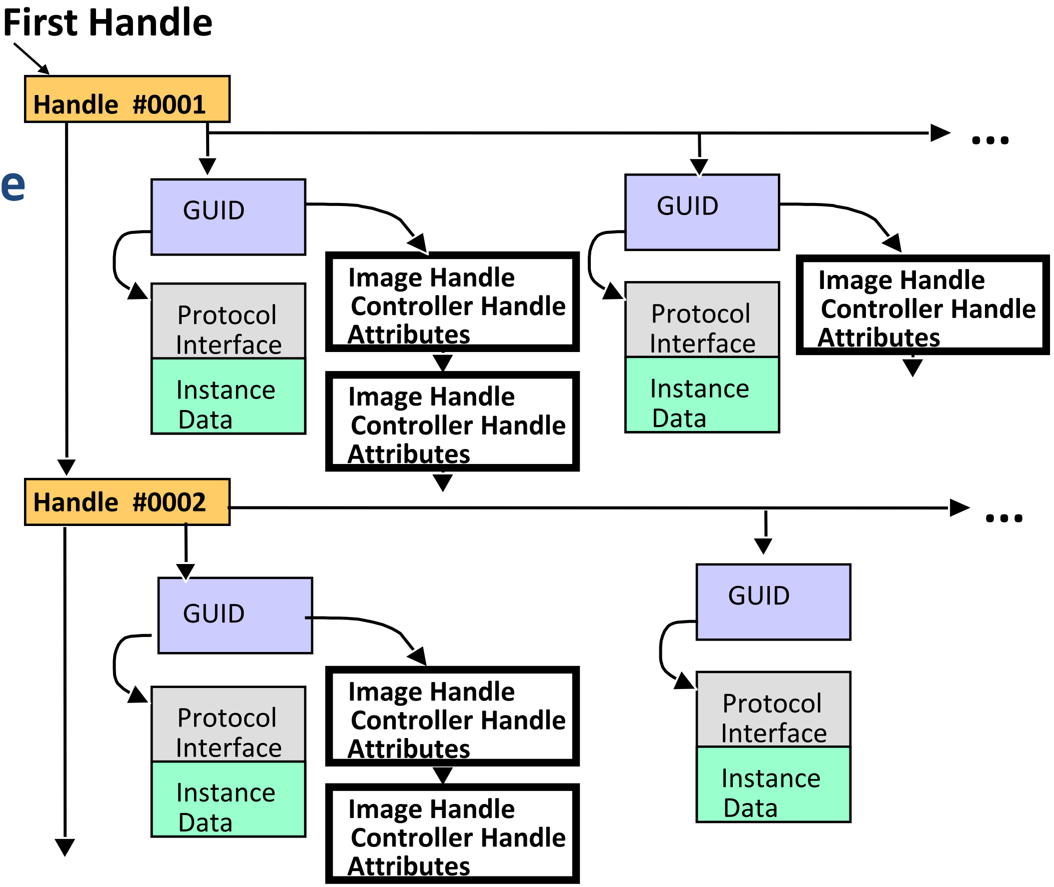
Protocols (API)

- **GUID, Interface Structure, Services**
  - `DEVICE_PATH, DEVICE_IO, BLOCK_IO, DISK_IO, FILE_SYSTEM, SIMPLE_INPUT, SIMPLE_TEXT_OUTPUT, SERIAL_IO, PXE_BC, SIMPLE_NETWORK, LOAD_FILE, UNICODE_COLLATION`



Handle Protocol Database

- The Handle Database is a list of UEFI handles that is the central repository for the objects maintained by UEFI based firmware
- Each UEFI handle is identified by a unique handle number
  - Can have >1 GUIDs
    - Device Path Protocol and an I/O abstraction Protocol
- A handle number provides a database "key" to an entry in the handle database.
- The type of protocol determines the Handle type (e.g. images, drivers, services)


First Handle
Handle #0001
...
GUID
Protocol Interface
Instance Data
Image Handle Controller Handle Attributes
Image Handle Controller Handle Attributes
GUID
Protocol Interface
Instance Data
Image Handle Controller Handle Attributes
Handle #0002
...
GUID
Protocol Interface
Instance Data
Image Handle Controller Handle Attributes
Image Handle Controller Handle Attributes
GUID
Protocol Interface
Instance Data

Handle points to more than one GUID

Device Path Protocol

- A data structure description of where a device is in the platform
- All boot devices, logical devices and images **must be described** by a UEFI device path
- The UEFI Specification defines six types of device paths

See § 9 UEFI 2.X Spec.

Six Types of Device Path Types

- **Hardware** – where is the device in the system
- **ACPI** – UID/HID of device in AML
- **Messaging** – Classifies device as LAN, Fiber Channel, ATAPI, SCSI, USB, …
- **Media** – i.e. Hard Drive, Floppy or CD-ROM
- **BIOS Boot Specification** – used to point to boot legacy operating systems
- **End of hardware** – marks end of device path

Device Path Examples

Acpi(PNP0A03,0[1])/Pci(1F|1)/Ata(Primary,Master)/HD(Part3, Sig00110011[2])

Acpi(PNP0A03,1)/Pci(1E/0)/Pci(0|0)/Mac(0002B3647D69)

Acpi(PNP0A03,0)/Pci(1F|0)/Acpi(PNP0501,0)/Uart(115200 81)

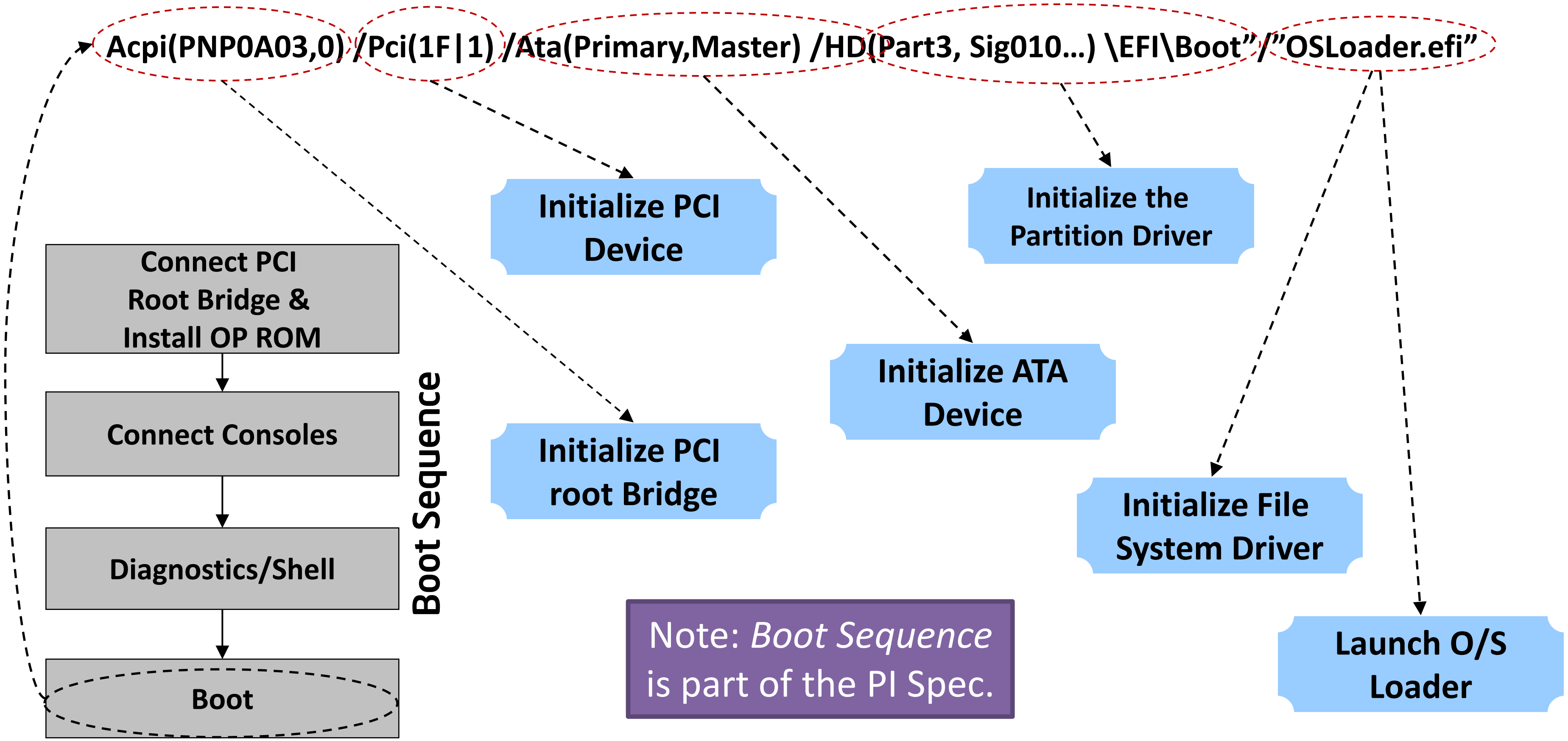
See § 9 UEFI 2.X Spec.

[1] ACPI Name space - contain HID, CID, and UID fields that match the HID, CID, and UID values that are present in the platform's ACPI tables

[2] Truncated to fit on slide, GUIDs are 128 bits

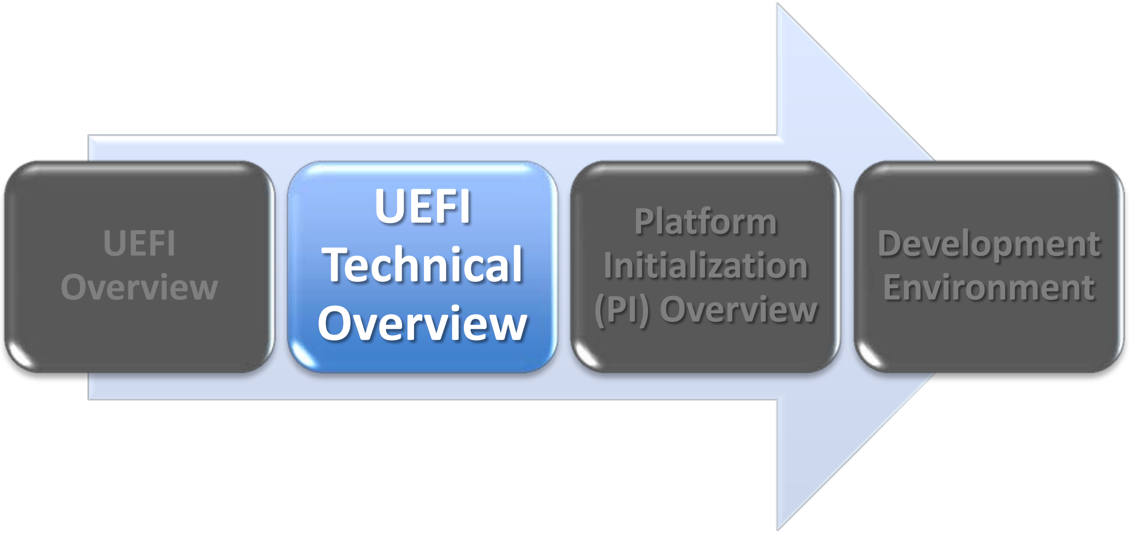
Why UEFI Device Path?

- The UEFI Device Path describes a boot target
  - Binary description of the physical location of a specific target


Acpi(PNP0A03,0) /Pci(1F|1) /Ata(Primary,Master) /HD(Part3, Sig010…) \EFI\Boot"/"OSLoader.efi"
Initialize PCI Device
Initialize the Partition Driver
Connect PCI Root Bridge & Install OP ROM
Connect Consoles
Diagnostics/Shell
Boot
Boot Sequence
Initialize ATA Device
Initialize PCI root Bridge
Initialize File System Driver
Launch O/S Loader

Note: *Boot Sequence* is part of the PI Spec.

UEFI Technical Overview



Boot Support

UEFI Terminology

UEFI Services

UEFI Driver Design

EFI Byte Code

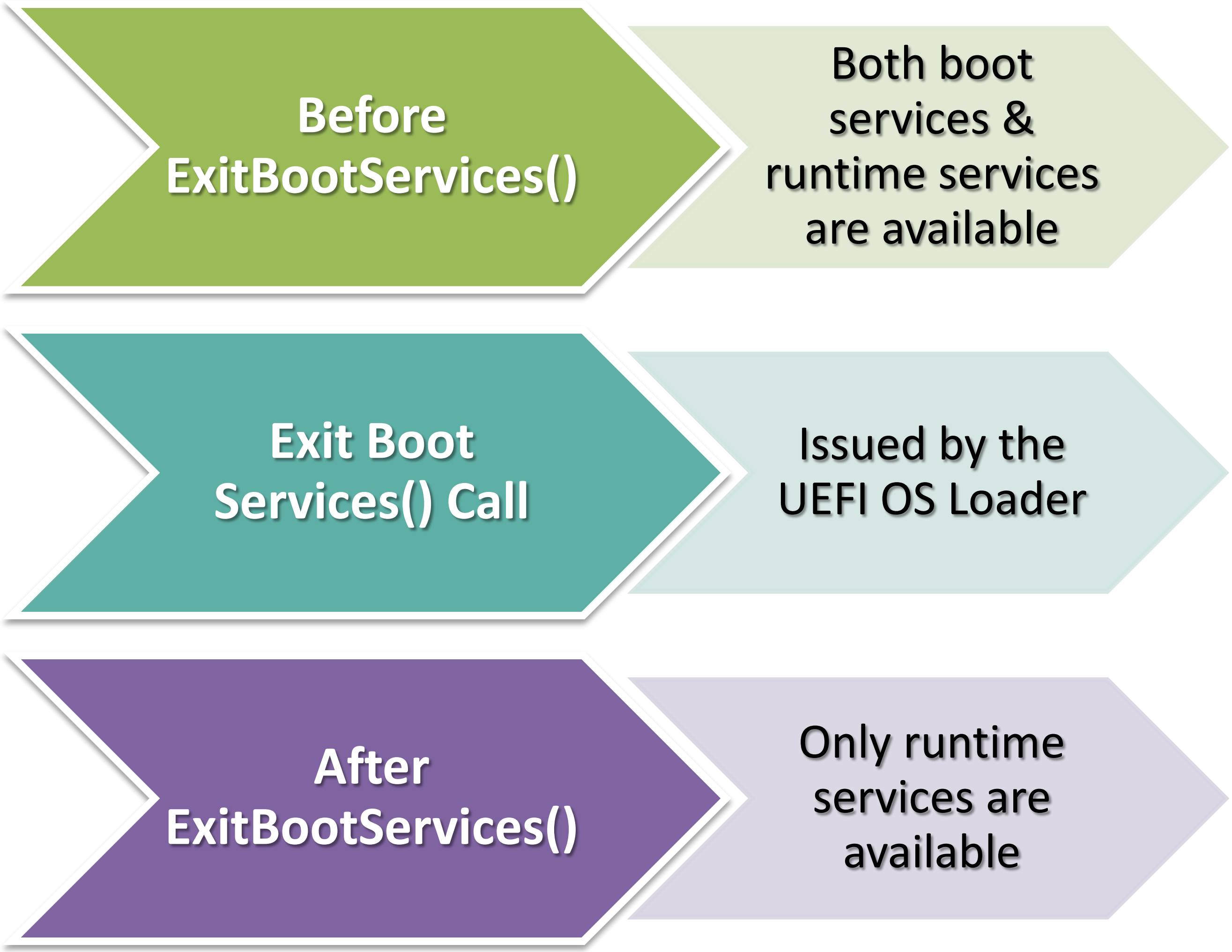
Boot Services

- *Full set of firmware services for pre-boot*
- Events and notifications
  - Polled devices, no interrupts
- Watchdog timer
  - Elegant recovery
- Memory allocation
- Handle location – for finding protocols
- Image loading
  - For drivers, applications & the OS loader

Runtime Services

- *Minimal set of services for the UEFI Aware OS*
- *Available in boot services and at OS runtime*
- Timer, Wakeup Alarm
  - Allows system to wake up or power on at a set time
- Variables
  - Boot manager handshake
- System reset

ExitBootServices()



Before ExitBootServices()

Both boot services & runtime services are available

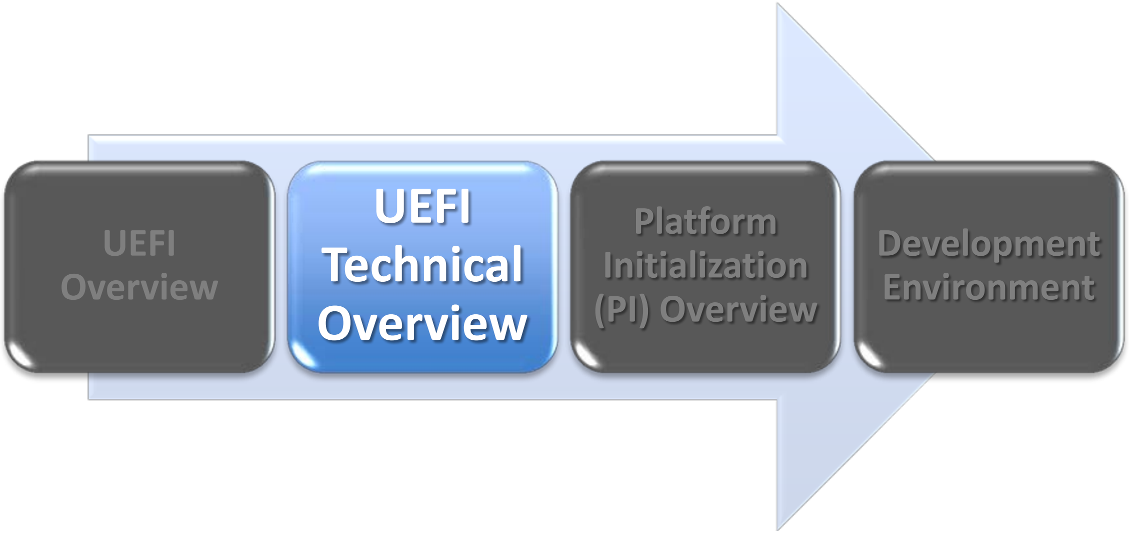
Exit Boot Services() Call

Issued by the UEFI OS Loader

After ExitBootServices()

Only runtime services are available

UEFI Technical Overview



Boot Support

UEFI Terminology

UEFI Services

UEFI Driver Design

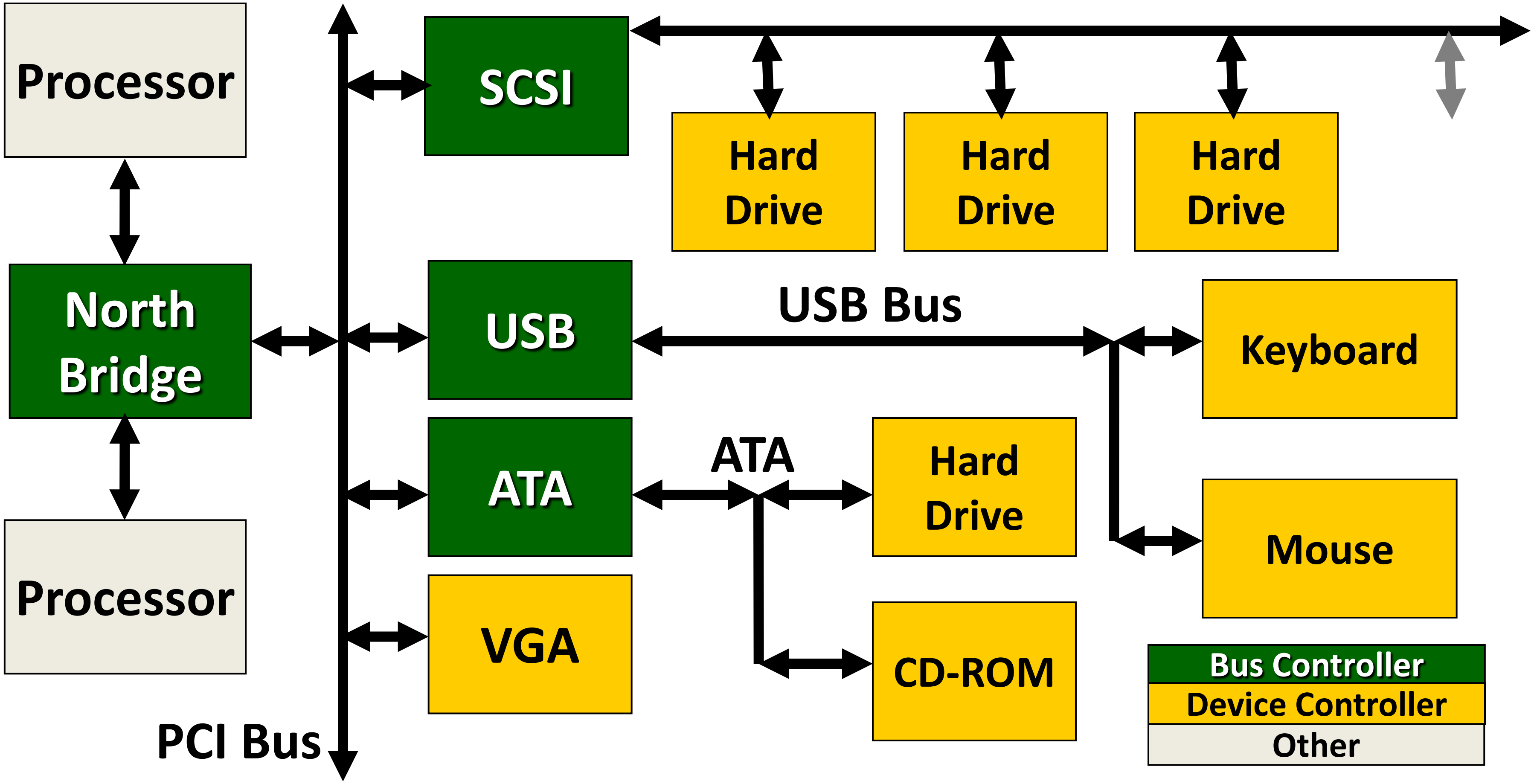
EFI Byte Code

The UEFI Driver Model

- UEFI Drivers extend firmware
  - Add support for new hardware
  - No dependence on OS or specific CPU architecture
  - Portable across platforms (IA32, x64, Itanium, …)
  - Reusable code leads to rapid platform development
- Supports complex bus hierarchies
- Driver Binding Protocol provides flexibility
  - Driver version management
  - Hot-plug and unload support
- Extensible to support future bus & device types

See § 10 UEFI 2.X Spec.

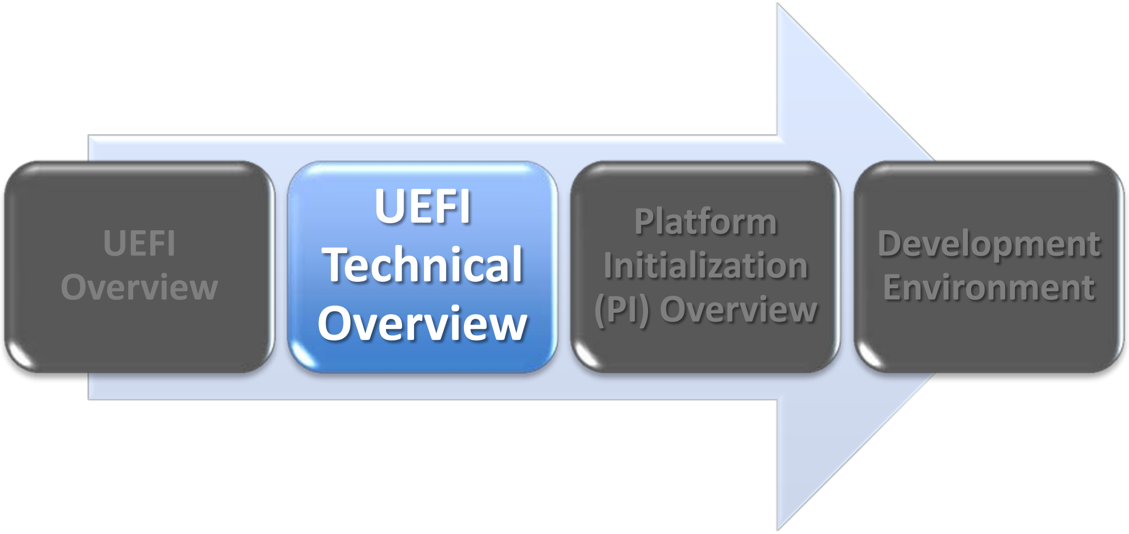
Typical System



Make the UEFI Driver Model simple and extensible so more complex systems can be described and managed in the pre-boot environment.

See § 2.5 UEFI 2.X Spec.

UEFI Technical Overview



UEFI Overview

UEFI Technical Overview

Platform Initialization (PI) Overview

Development Environment

Boot Support

UEFI Terminology

UEFI Services

UEFI Driver Design

EFI Byte Code

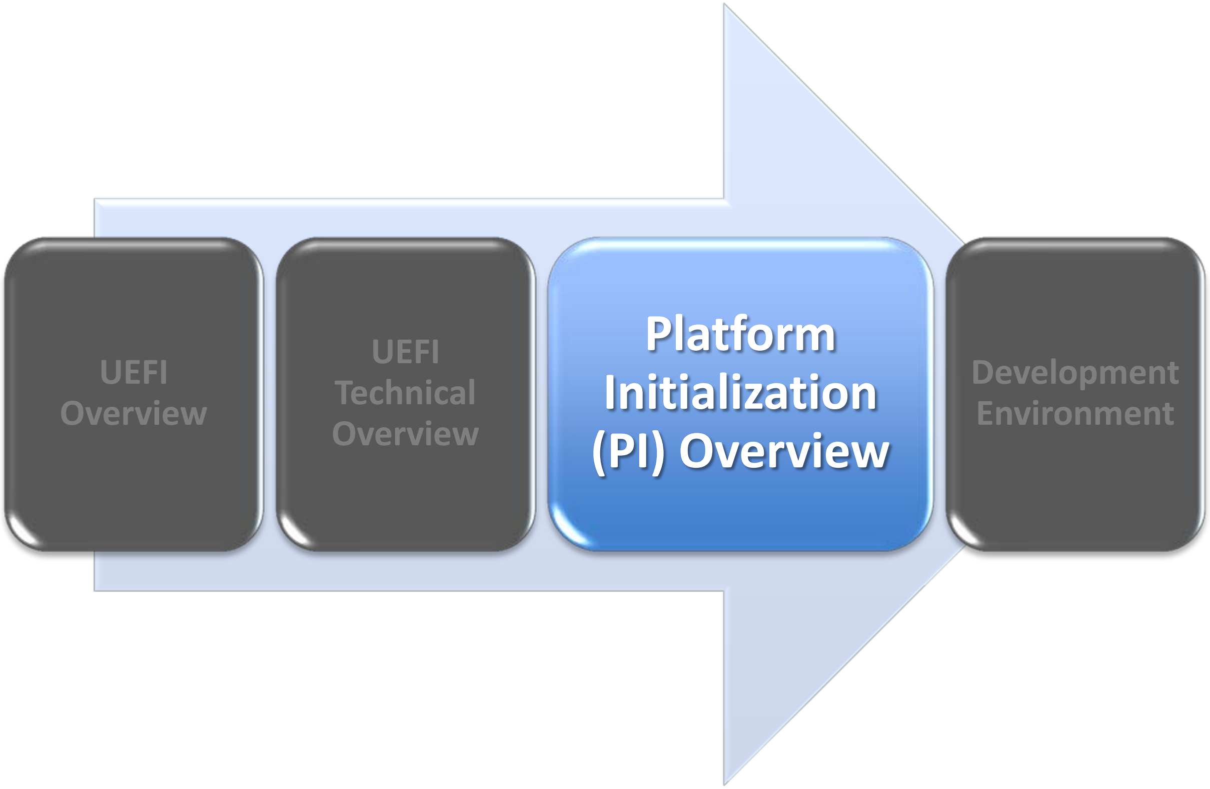
EFI Byte Code (EBC)

- Allow drivers compile to a single image to work under any platform using UEFI.
- EBC is an interpreted language, compiled form C source into the Byte code language (similar to a Risc processors Object code – this object code is defined in the UEFI specification) . Using the Interpreted version allows for a single image that is capable of supporting both Itanium and IA32 implementations
- By specification, a UEFI solution MUST contain an interpreter for EBC. This interpreter is written in native code for the architecture, so it can interpret EBC modules. Since all UEEFI platforms have the interpreter regardless of architecture, all platforms can run the same code (through interpretation).
- This allows a single driver to be executable across all architectures, reducing development time, testing, and support burdens. This was not the case Before UEFI, when Option ROMs and drivers had to be developed uniquely for each architecture to be supported by the hardware device.

When to Use EBC

- Add-in Video Adapters & Disk Controllers
- Not used for NICs (UNDI)
  - *Note: UNDI is runtime which must be native*
- Reduce driver image footprint
- Adapters supporting multiple Processor types
  - IA-32 and IA-64
  - IA-32 and Intel® 64
  - Intel® 64 and IA-64
  - IA-32, Intel® 64 and IA-64
  - Reduce adapter SKUs

Platform Initialization (PI) Overview



PI Overview

UEFI & PI Spec

Boot Execution Flow

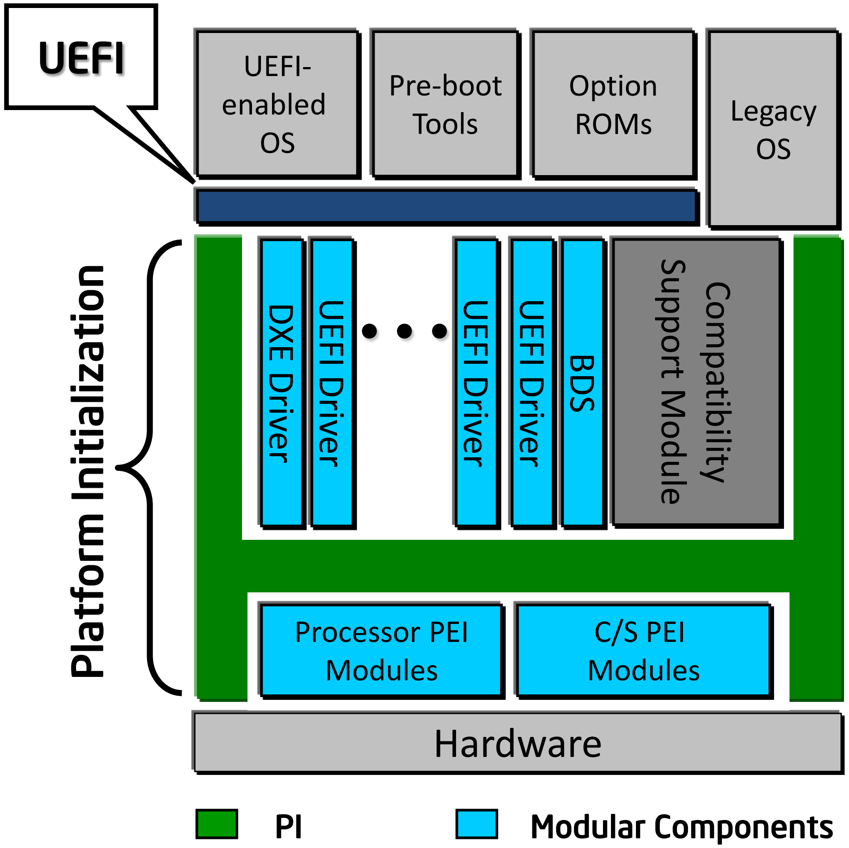
Legacy Considerations

PI – Technology not addressed by UEFI

- Memory Initialization
- Recovery
- FLASH update
- ACPI S3
- Platform Initialization
- System Management Mode (SMM)
- Setup

UEFI Separates BIOS and OS

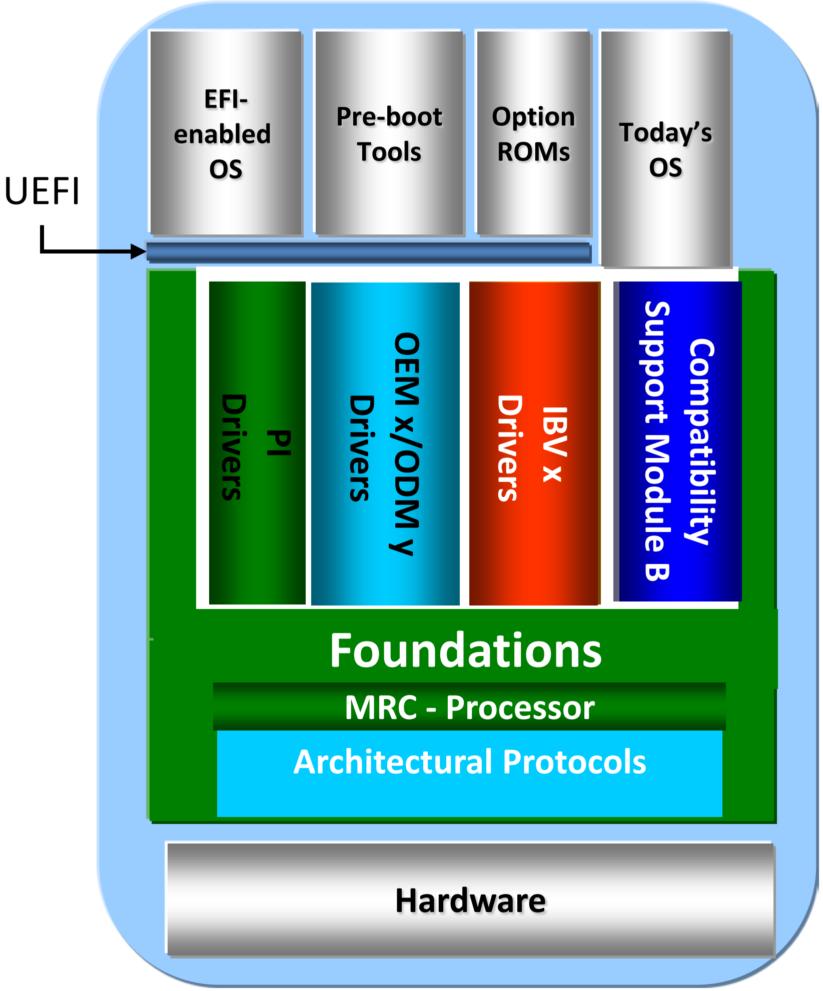
USWG/PIWG Relationship



- **UEFI Spec defines interfaces between OS, add-in driver and system firmware**
  - **A new model for the interface between the Operating Systems, other high-level software and the platform firmware add-in drivers**
- **PI Specs relate to making UEFI implementations**
  - **Promote interoperability between firmware components**
  - **Modular components like silicon drivers (e.g. PCI) and value-add drivers (security)**

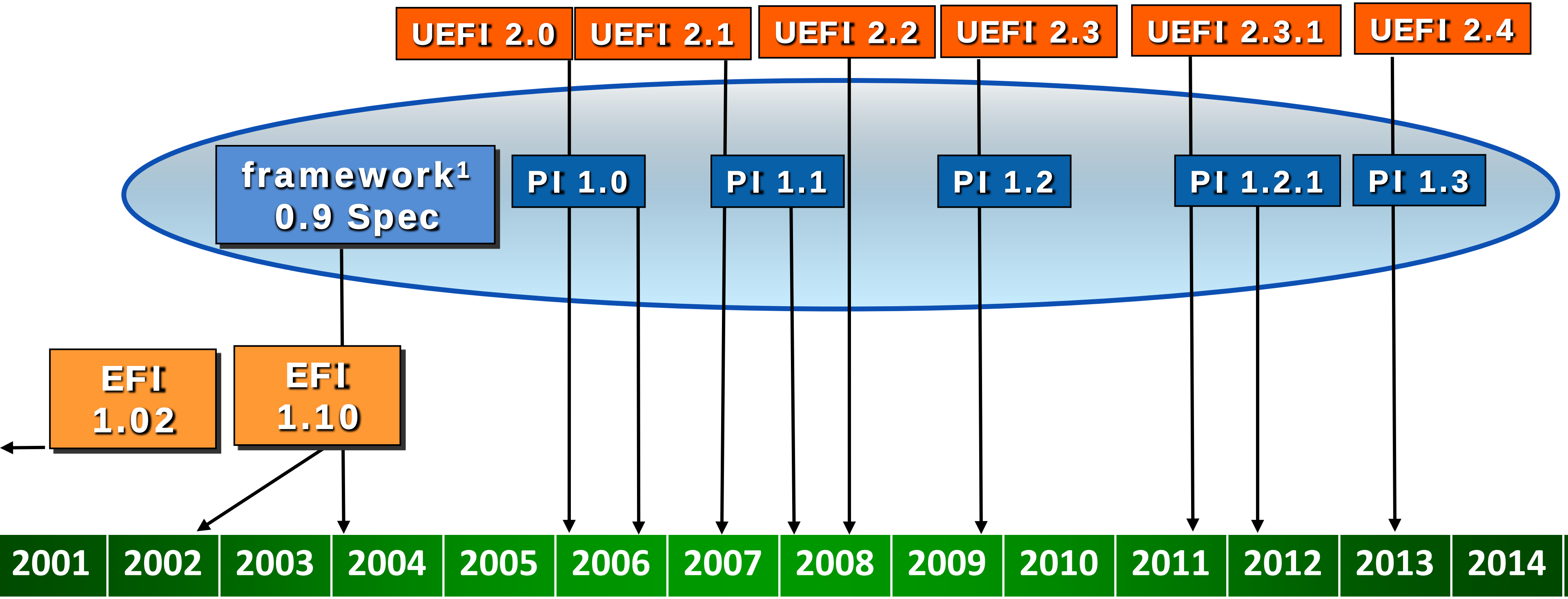
UEFI and PI are Independent Interfaces

Intel® Platform Innovation Framework for UEFI and Platform Initialization (PI)



- Base Core Foundation ("*Green H*")
  - Foundation helps teams share code
  - Developers can easily move between projects using a common foundation
- Industry standardization
  - PI spec is managed by UEFI Forum
  - Chipset code from silicon vendor
  - IBV provides value add
- The "Green H" code is Open Source on www.tianocore.org
- Intel's framework evolved into the PI Specification (uefi.org)

Timeline for the Intel® Platform Innovation Framework for UEFI and PI Specification



[1] Intel® Platform Innovation Framework for UEFI

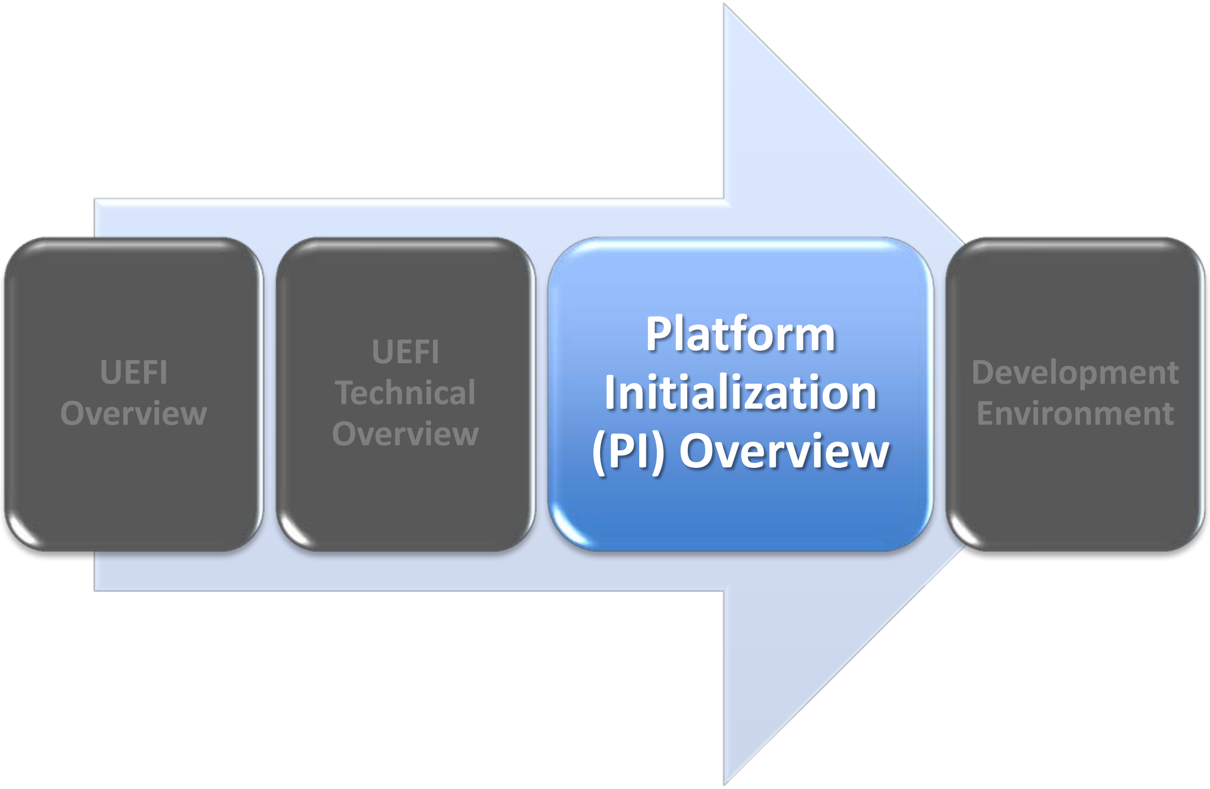
Get to "C" Code Quickly

- The Framework/PI uses modules for Processor, chipset and board
  - Only execute the minimum initialization required to get memory working
- Architecture only requires "enough" memory
  - PEI is limited, so defer to the rich DXE "C" environment for complex driver execution
- Minimize execution of PEI modules from flash
  - Quickly get to execution from memory

Standard tools & flexible memory initialization

[1] Intel® Platform Innovation Framework for UEFI

Platform Initialization (PI) Overview



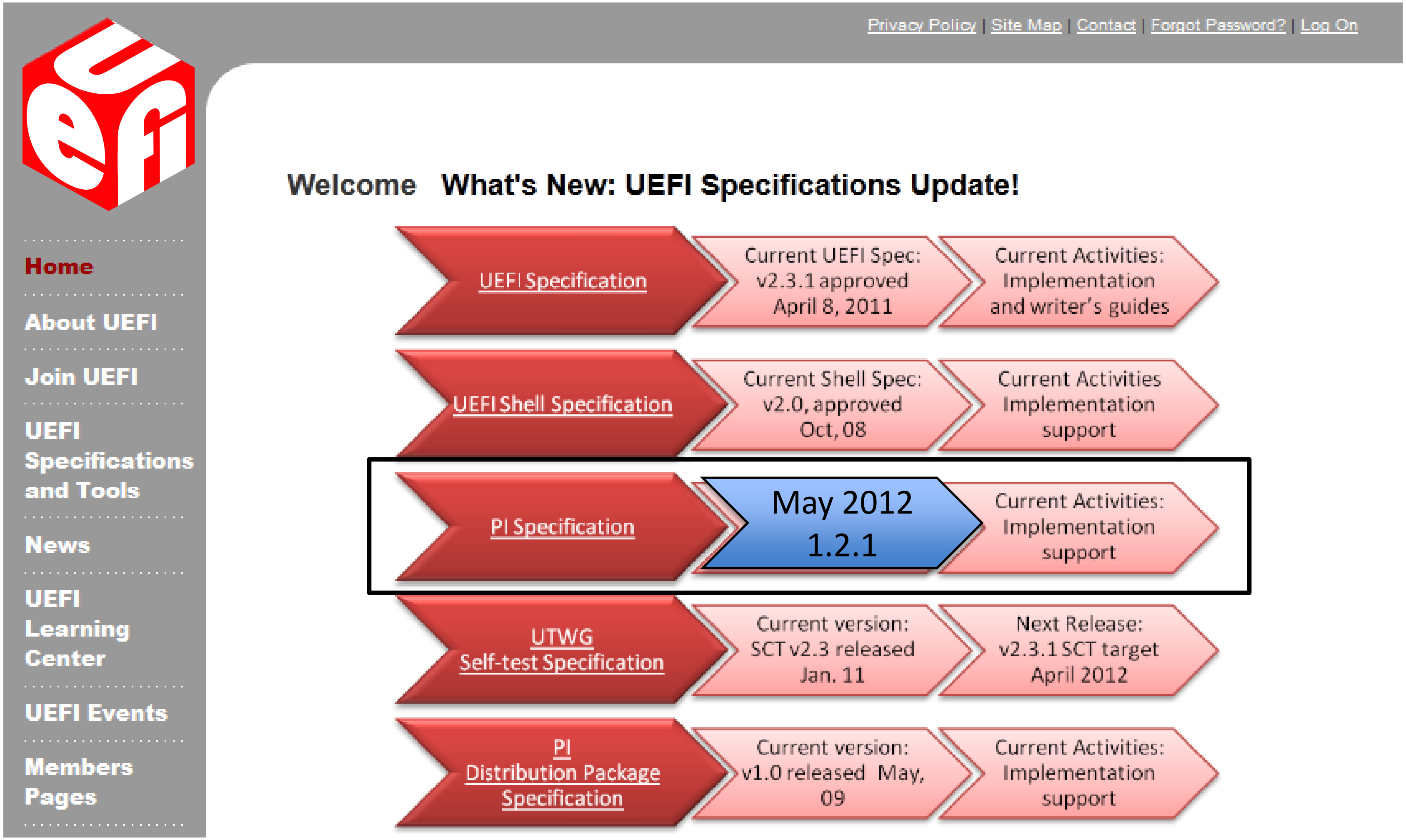
PI Overview

UEFI & PI Spec

Boot Execution Flow

Legacy Considerations

UEFI Platform Initialization Working Group (PIWG) Manages the PI Specification



Privacy Policy | Site Map | Contact | Forgot Password? | Log On

- Home
- About UEFI
- Join UEFI
- UEFI Specifications and Tools
- News
- UEFI Learning Center
- UEFI Events
- Members Pages

Welcome What's New: UEFI Specifications Update!

UEFI Specification	Current UEFI Spec: v2.3.1 approved April 8, 2011	Current Activities: Implementation and writer's guides
UEFI Shell Specification	Current Shell Spec: v2.0, approved Oct, 08	Current Activities Implementation support
PI Specification	May 2012 1.2.1	Current Activities: Implementation support
UTWG Self-test Specification	Current version: SCT v2.3 released Jan. 11	Next Release: v2.3.1 SCT target April 2012
PI Distribution Package Specification	Current version: v1.0 released May, 09	Current Activities: Implementation support

Details on PI Specification

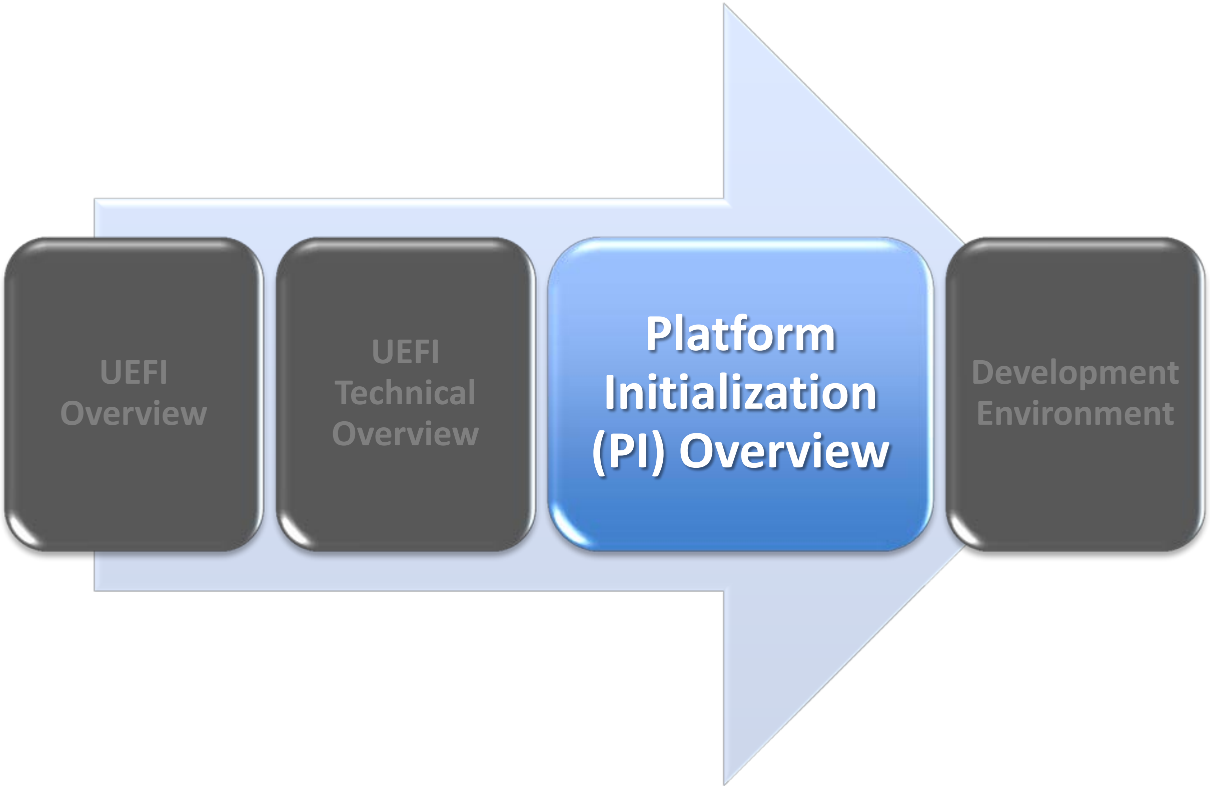
- UEFI Board first approved PI (1.0) in Oct 2006
  - Available for download from http://www.uefi.org
- Built using Intel contributed specifications
  - Built on the Framework PEI & DXE Specifications
  - Also leverages Firmware Storage, Hand Off Block (HOB) & SMBus specifications
- PI Self Certification Test (SCT) available on SourceForge
  - PI SCT v.1.2 released in 2010
- Spec Updates
  - PI 1.1 Approved 2007
  - PI 1.2 Approved 2009
  - PI 1.2.1 Approved 2012

Details

PI Specification Volumes

- VOLUME 1: Pre-EFI Initialization Core Interface (PEI-CSI)
- VOLUME 2: Driver Execution Environment Core Interface (DXE-CSI)
- VOLUME 3: Shared Architectural Elements / Firmware Storage Spec
- VOLUME 4: System Management Mode (SMM)
- VOLUME 5: Standards
- *Over 1300 pages total*

Platform Initialization (PI) Overview



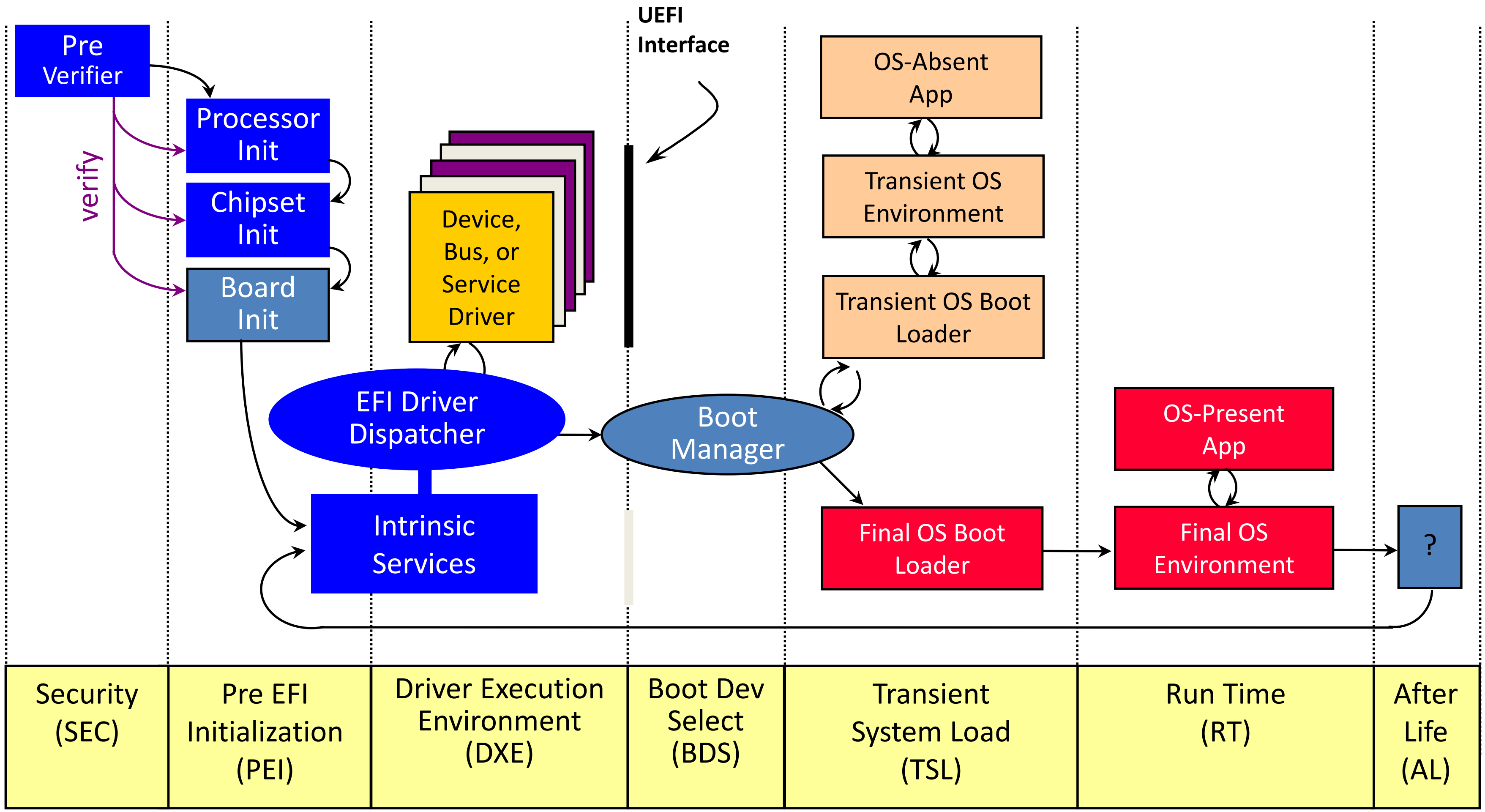
PI Overview

UEFI & PI Spec

Boot Execution Flow

Legacy Considerations

Architecture Execution Flow



Pre Verifier

verify

Processor Init

Chipset Init

Board Init

Device, Bus, or Service Driver

EFI Driver Dispatcher

Intrinsic Services

UEFI Interface

Boot Manager

OS-Absent App

Transient OS Environment

Transient OS Boot Loader

Final OS Boot Loader

OS-Present App

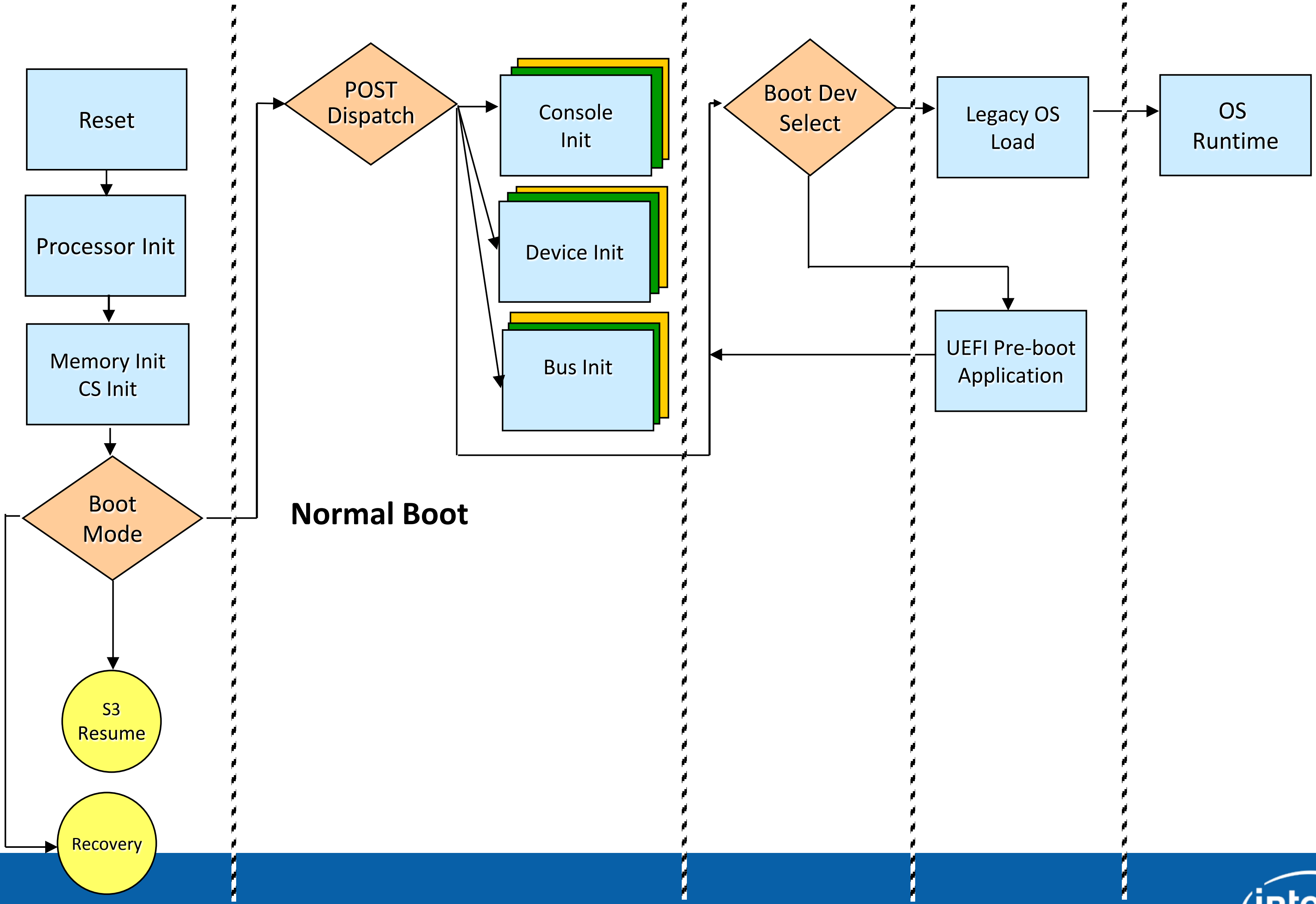
Final OS Environment

?

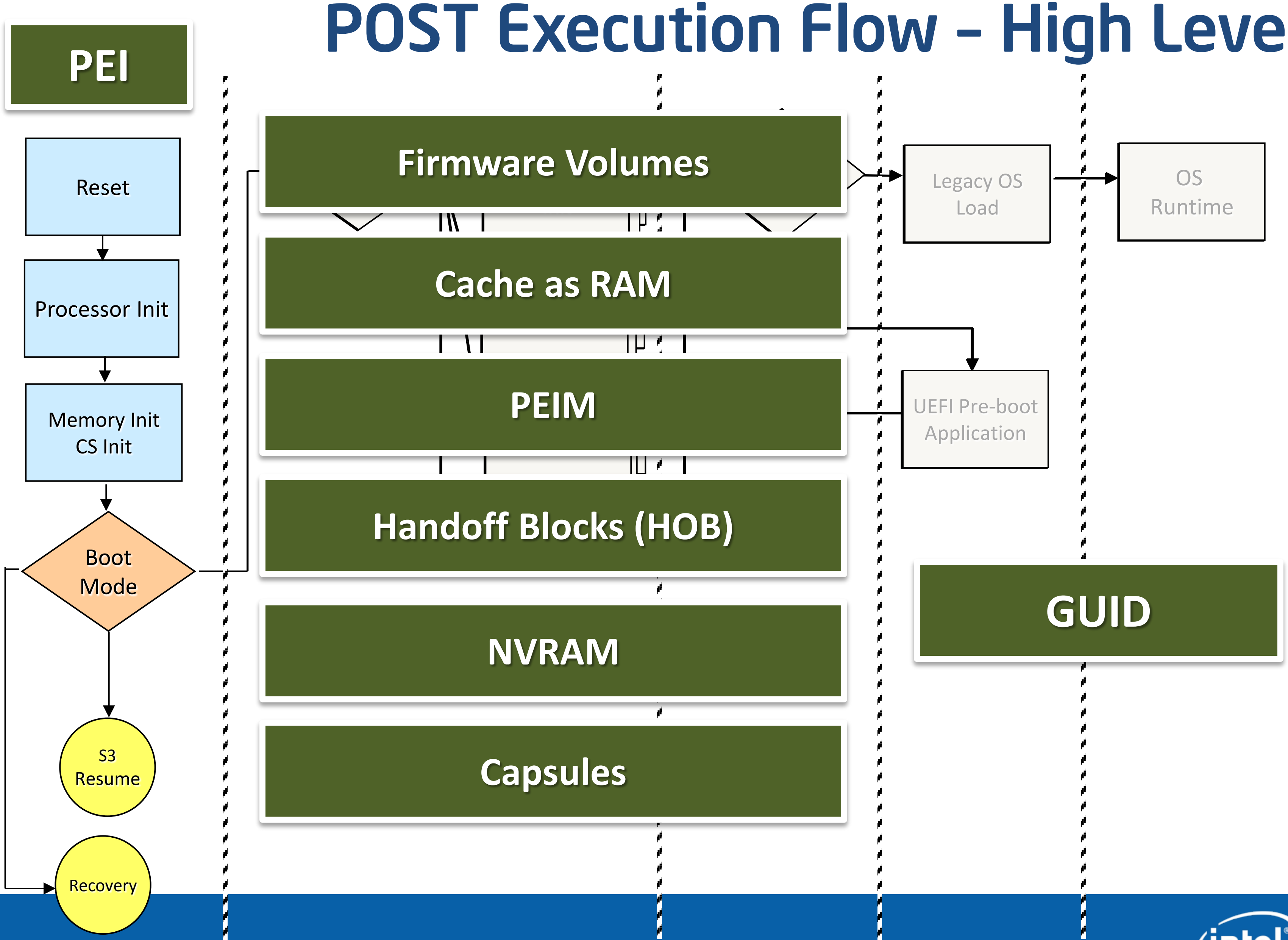
Security (SEC)	Pre EFI Initialization (PEI)	Driver Execution Environment (DXE)	Boot Dev Select (BDS)	Transient System Load (TSL)	Run Time (RT)	After Life (AL)

Power on → [. . Platform initialization . .] → [. . . . OS boot . . . .] → Shutdown

POST Execution Flow – High Level



POST Execution Flow – High Level



Firmware Volumes

Cache as RAM

PEIM

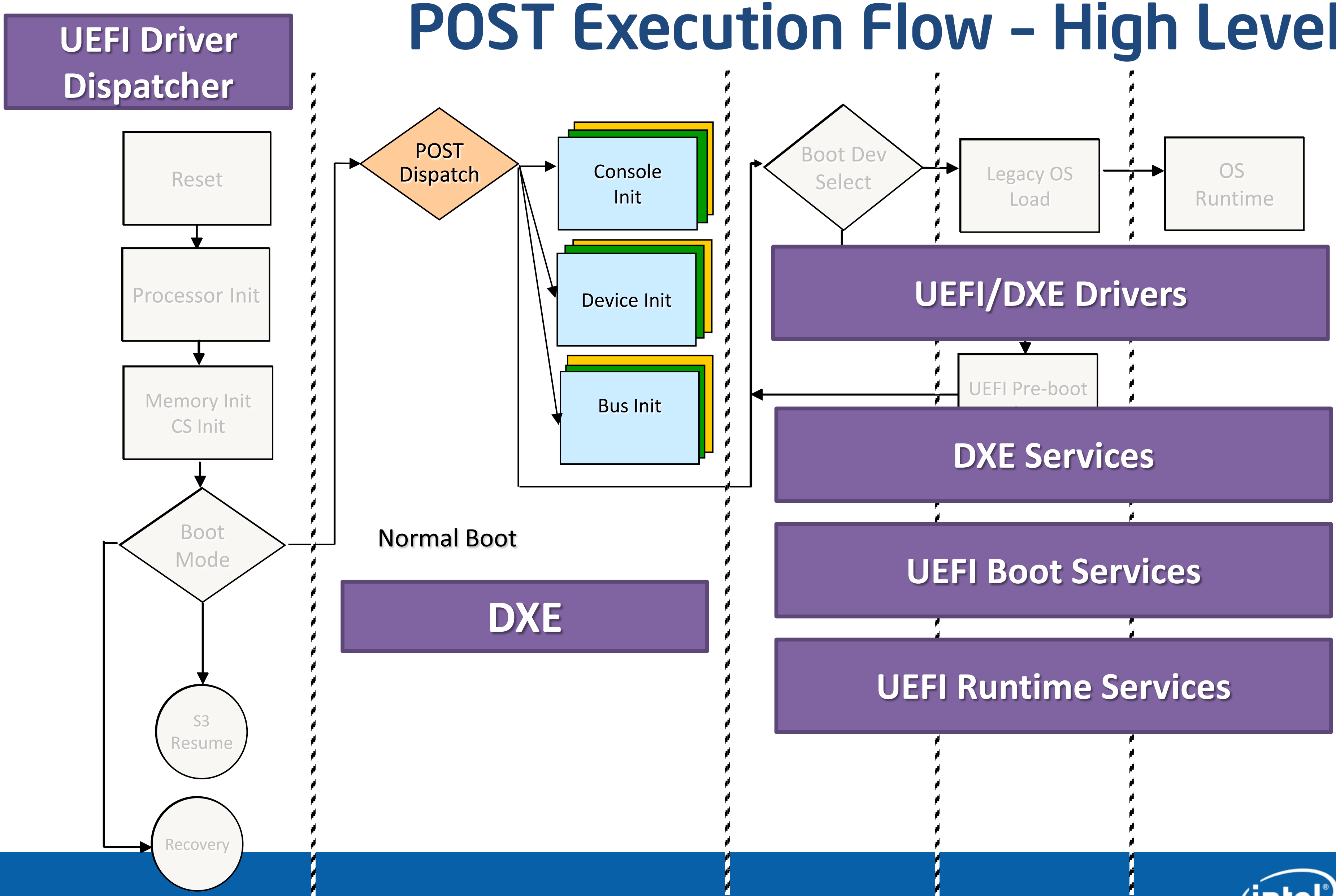
Handoff Blocks (HOB)

NVRAM

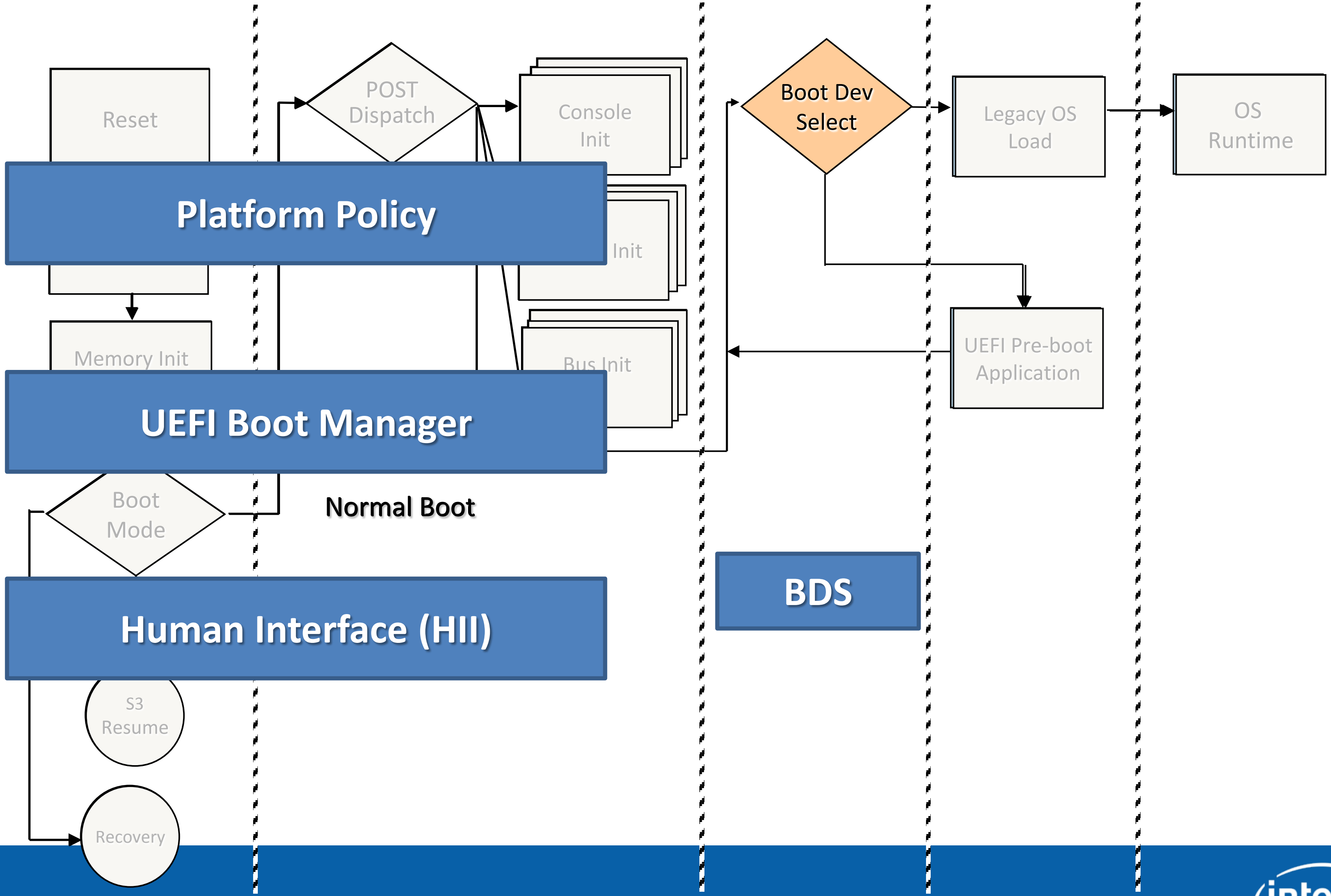
Capsules

GUID

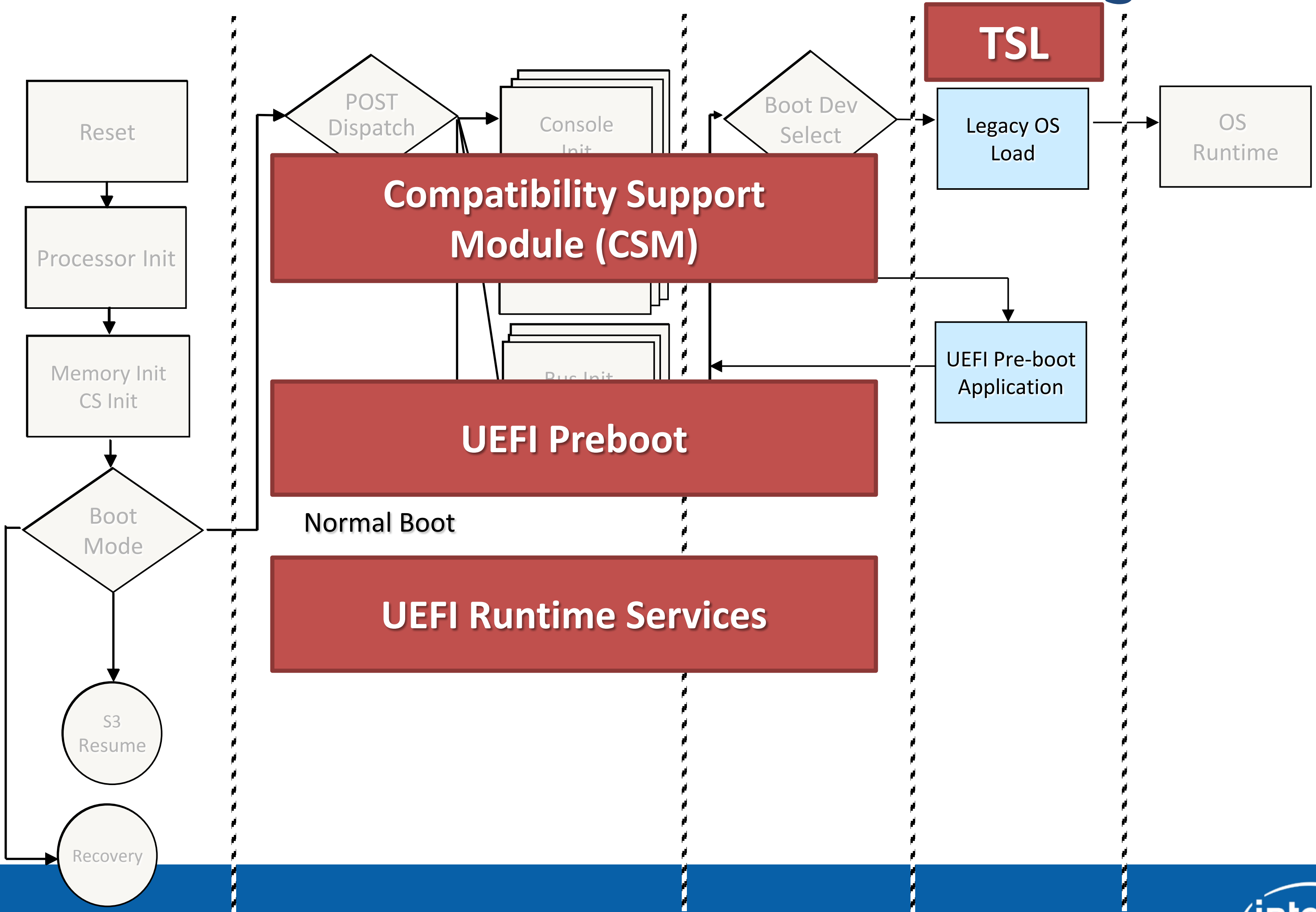
POST Execution Flow – High Level



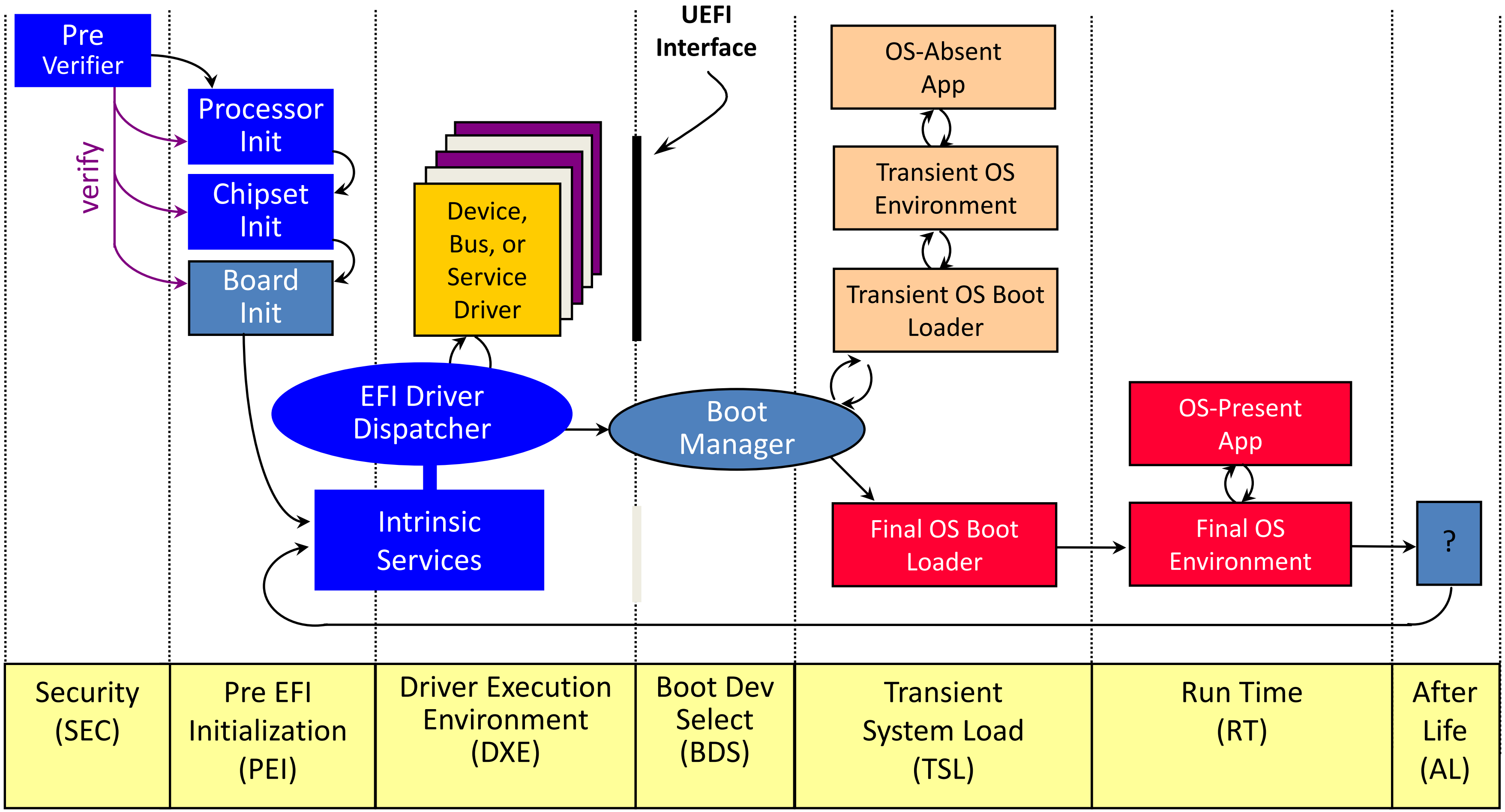
POST Execution Flow – High Level



POST Execution Flow – High Level



Architecture Execution Flow



Pre Verifier

verify

Processor Init

Chipset Init

Board Init

Device, Bus, or Service Driver

EFI Driver Dispatcher

Intrinsic Services

UEFI Interface

Boot Manager

OS-Absent App

Transient OS Environment

Transient OS Boot Loader

Final OS Boot Loader

OS-Present App

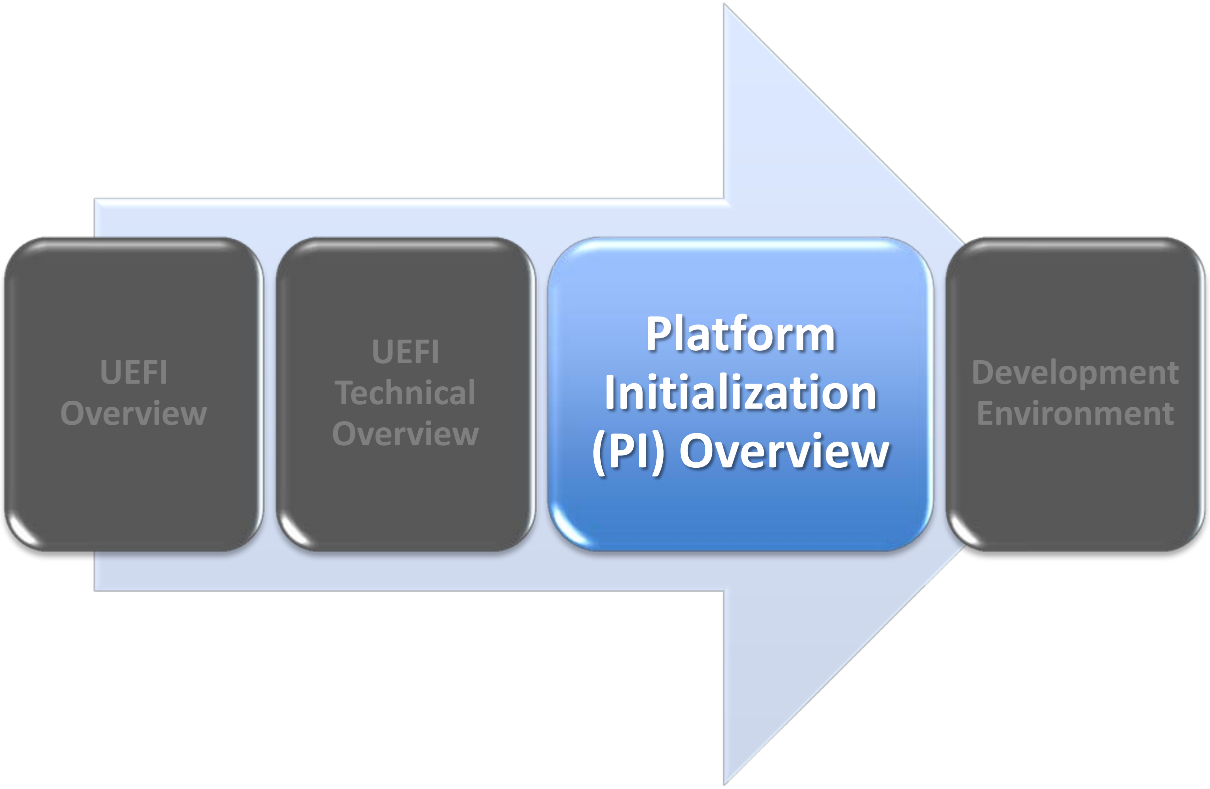
Final OS Environment

?

Security (SEC)	Pre EFI Initialization (PEI)	Driver Execution Environment (DXE)	Boot Dev Select (BDS)	Transient System Load (TSL)	Run Time (RT)	After Life (AL)

Power on → [. . Platform initialization . .] → [. . . . OS boot . . . .] → Shutdown

Platform Initialization (PI) Overview



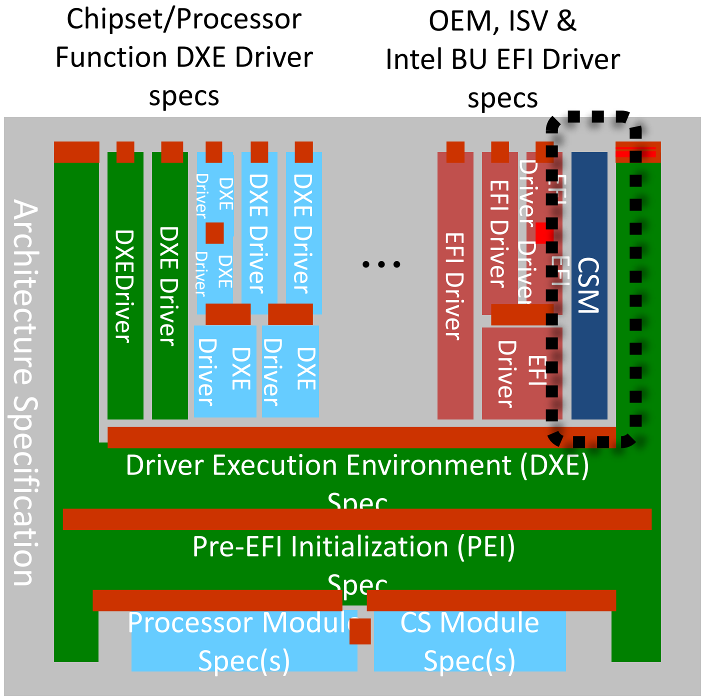
PI Overview

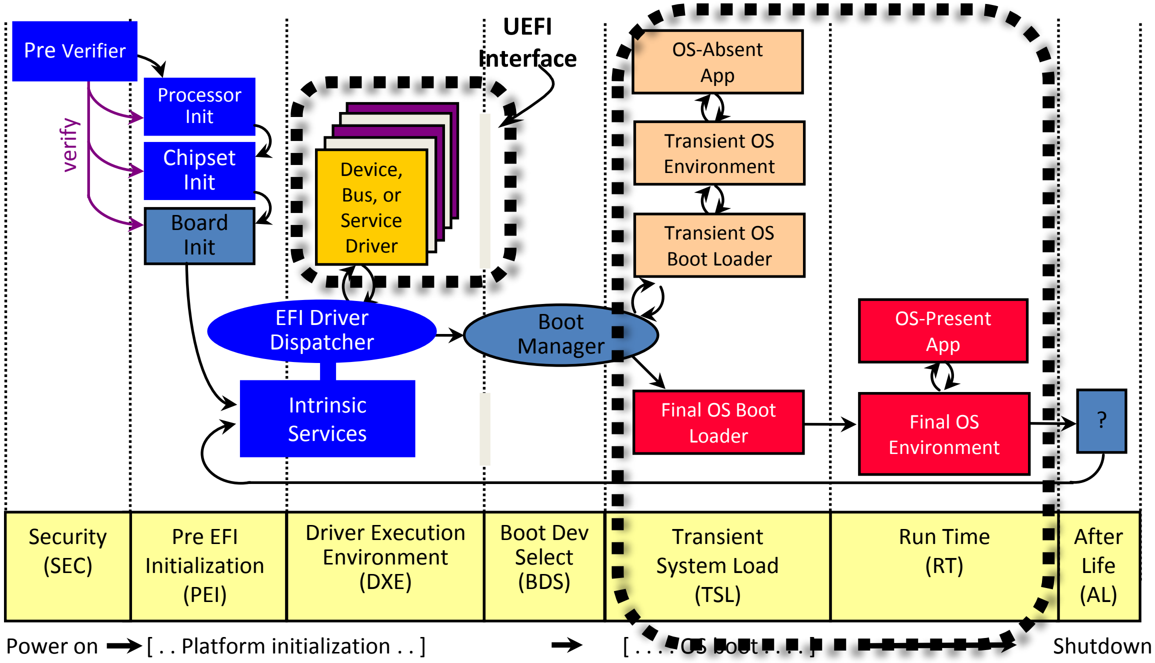
UEFI & PI Spec

Boot Execution Flow

Legacy Considerations

Compatibility Support Module





Why CSM ?

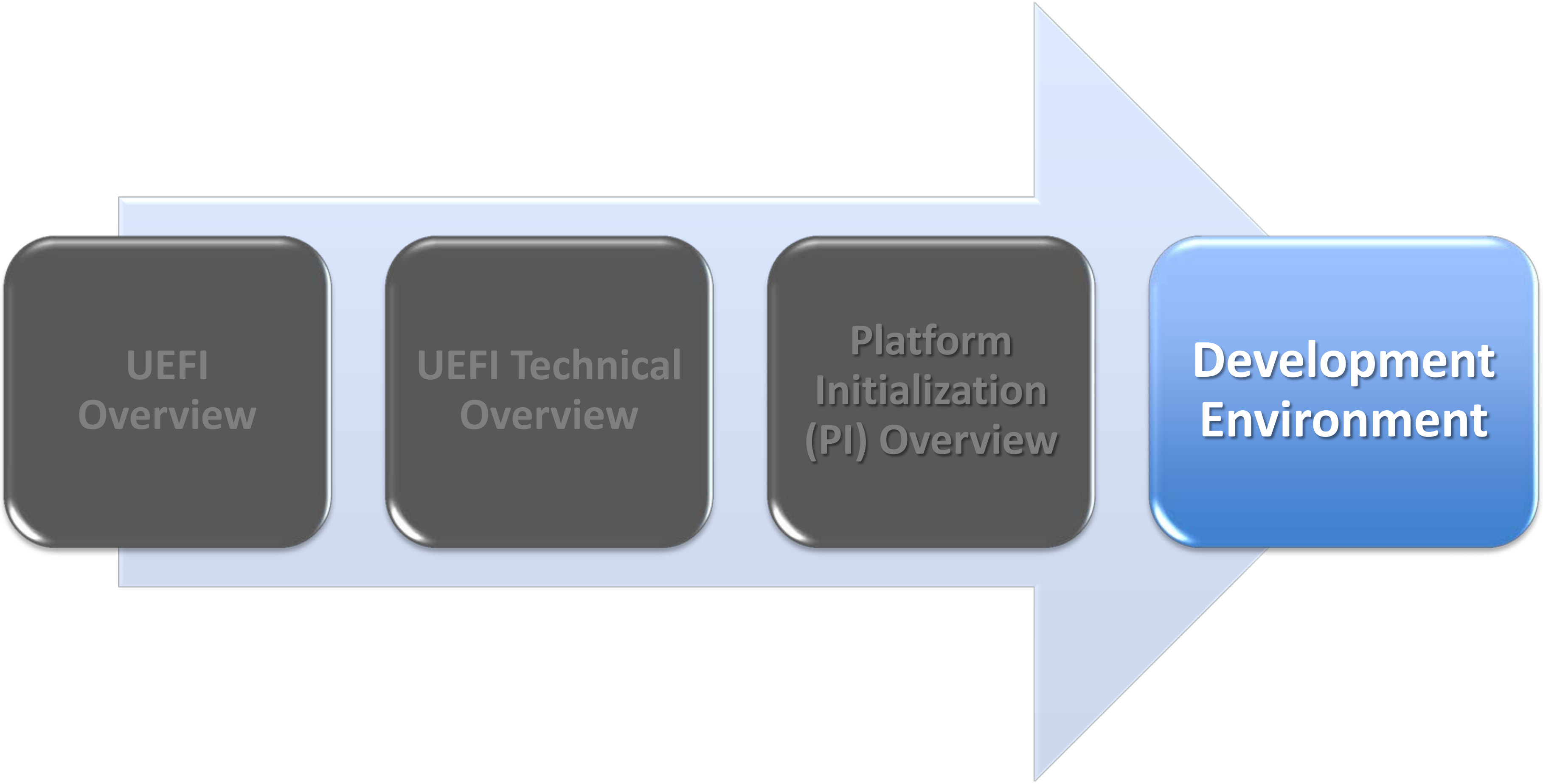
- A bridge between UEFI and legacy BIOS
- Booting a traditional or non-UEFI-aware OS
- Loading an UEFI-aware OS a device that is controlled by a traditional OpROM
- Legacy OpROM support will be required longer than legacy OS support
- **CSM is optional** … a "pure" UEFI system can be built without CSM (no legacy BIOS compatibility)

Compare The Two Environments

Legacy BIOS	UEFI
Special memory regions (A0000-FFFFF)	No special memory regions
16-bit big real	32-bit flat (or 64-bit)
Many interrupt sources	One interrupt source
Allocates PIRQs	OS does PIRQ allocation
In-flash Op ROMs for on-board devices	In-flash Op ROMs stored as firmware files
Custom INT15s	Extensible protocols
All devices enumerated (time consuming)	Normally, only enumerates boot devices (faster)

See the presentation on CSM Architecture for more information

Development Environment



Development Environment - Tools

- Start from an Open Source framework
  - EFI Development Kit (EDK II)
  - EDK II Application Development Kit (EADK)
- Build using standard C compilers
  - Commercial (Intel, MSVC) or open source (GCC)
- Self-hosted development of drivers
- Libraries created for common tasks
- Leverage open source platforms & projects to support UEFI development
  - NT32, OVMF, UEFI Shell 2.0, UEFI Driver Wizard

"Burden the Tools"

- Automated "make" file generation
  - Build configuration file selects components for image
  - Config file drives construction of make hierarchy
- Where possible, use tools in the build environment to catch problems
  - Avoid carrying error handling code in ROM
  - Ex: static check to catch driver dependency loops
- Test coverage: build in flow checking
- Performance instrumentation
  - Time spent in major phases and individual modules
  - Helps focus boot speed optimization on real problems

Debug Support

- Source level “C” debugging of components and applications with hardware emulators
- Debug Text Output
  - Formatted text to console and/or serial stream
  - Enabled early in PEI & all of DXE
- Assertions
- Status Codes
  - Logs progress, error & debug codes to DataHub
  - Output to Port 80 driver, console or serial stream
  - Enabled early in PEI & all of DXE

Open Source @ TianoCore.org

EDK II ⟷ Not a platform … a build environment for platforms
Makes use of packages, libraries and PCD values

EADK ⟷ EDK II Application Development Kit, includes Standard C Libraries for porting ANSI/POSIX applications to UEFI

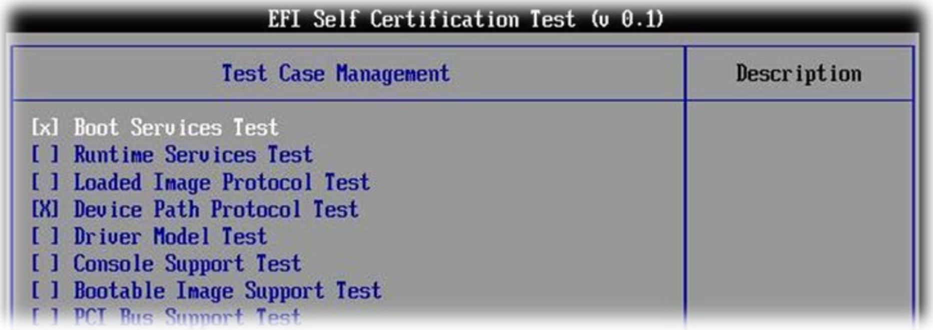
UEFI Shell ⟷ UEFI application that provides an interactive interface that allows you to launch UEFI applications and drivers
Runs on any platform using the UEFI Shell protocol

NT32 ⟷ Microsoft Windows based emulation used to run the UEFI Shell for application and driver development

OVMF ⟷ Open Virtual Machine Firmware (OVMF)
Support UEFI firmware for Virtual Machines (QEMU)

UEFI Self Certification Test (SCT)

- SCT is a set of tests used to verify compliance of UEFI implementations
- http://www.uefi.org, including
  - UEFI SCT Specs and Documents
  - UEFI SCT Source Code
  - Executable images on IA32, Intel® 64, and IA-64 platforms



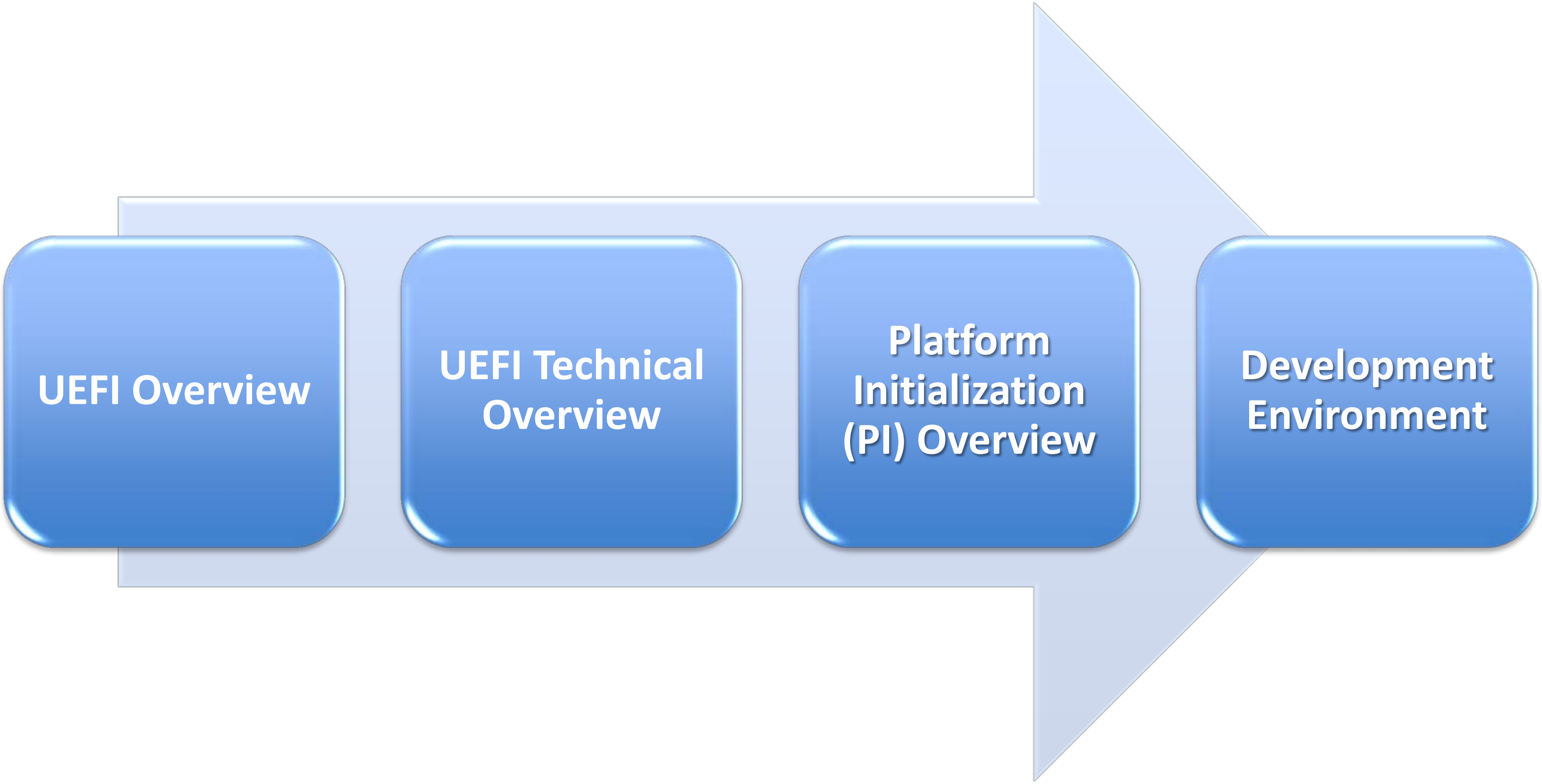
```
EFI Self Certification Test (v 0.1)

Test Case Management                          Description

[x] Boot Services Test
[ ] Runtime Services Test
[ ] Loaded Image Protocol Test
[X] Device Path Protocol Test
[ ] Driver Model Test
[ ] Console Support Test
[ ] Bootable Image Support Test
[ ] PCI Bus Support Test
```

USAGE

- **System**: UEFI SCT has two models:
  - One is native mode which is invoked as an UEFI application SCT from local EFI Shell
  - Passive mode which executes UEFI SCT Agent in EFI Shell and runs all test cases on UEFI Management side(EMS).
- **Option ROM**: UEFI IHV SCT is the toolset for the Independent Hardware Vendors(IHV)

Platform Initialization (PI) SCT 1.2 located on http://sourceforge.net/projects/pi-sct/

Summary

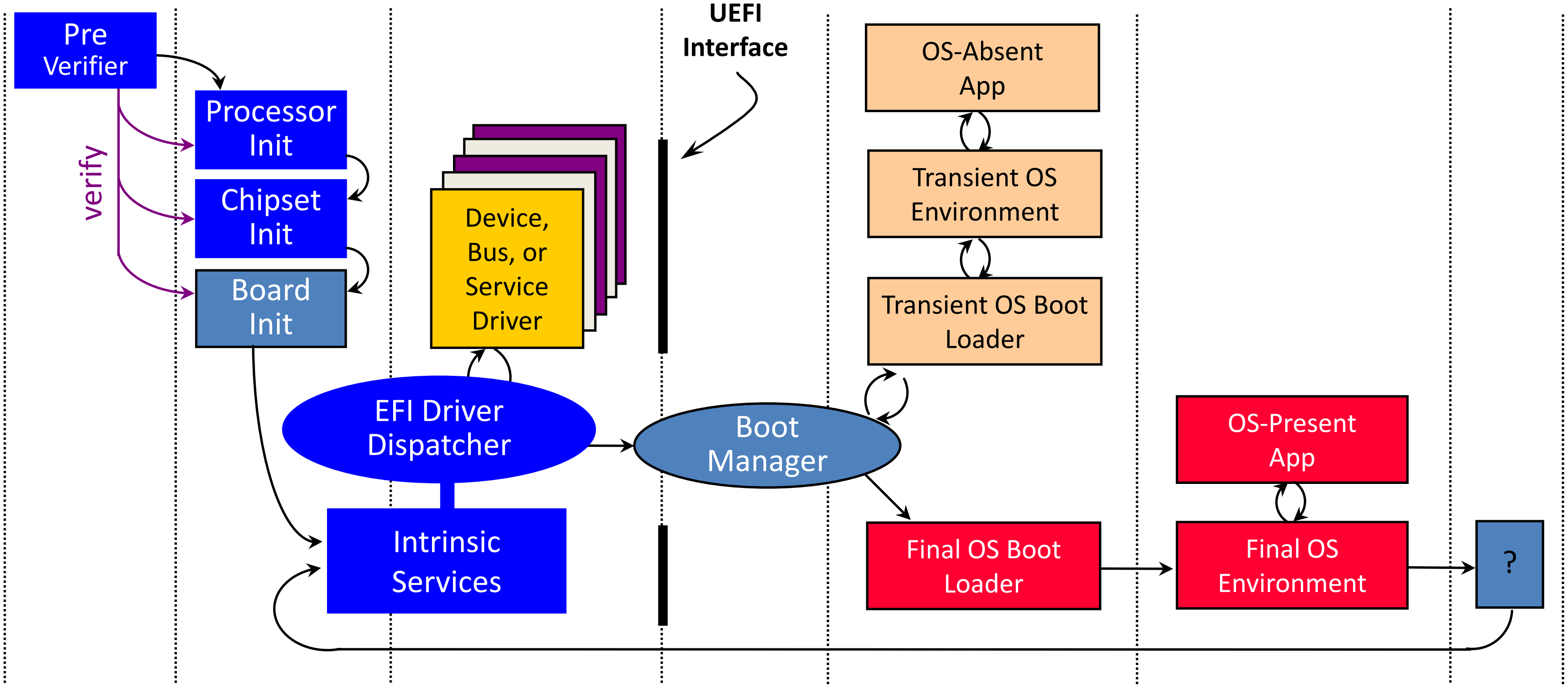

UEFI Overview

UEFI Technical Overview

Platform Initialization (PI) Overview

Development Environment

Architecture Execution Flow



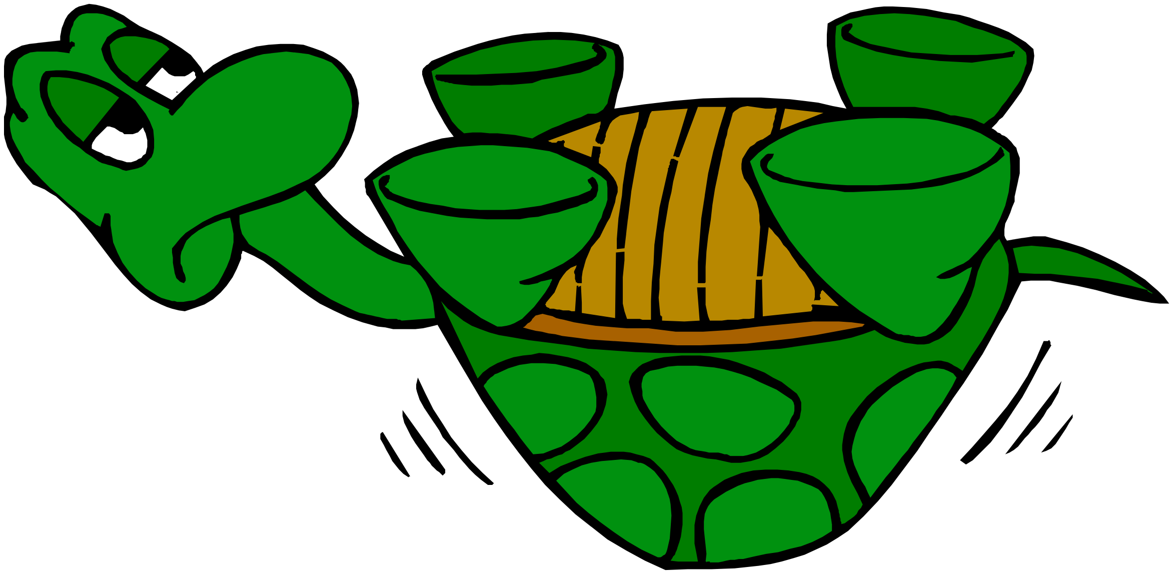
Security (SEC)	Pre EFI Initialization (PEI)	Driver Execution Environment (DXE)	Boot Dev Select (BDS)	Transient System Load (TSL)	Run Time (RT)	After Life (AL)

Q & A

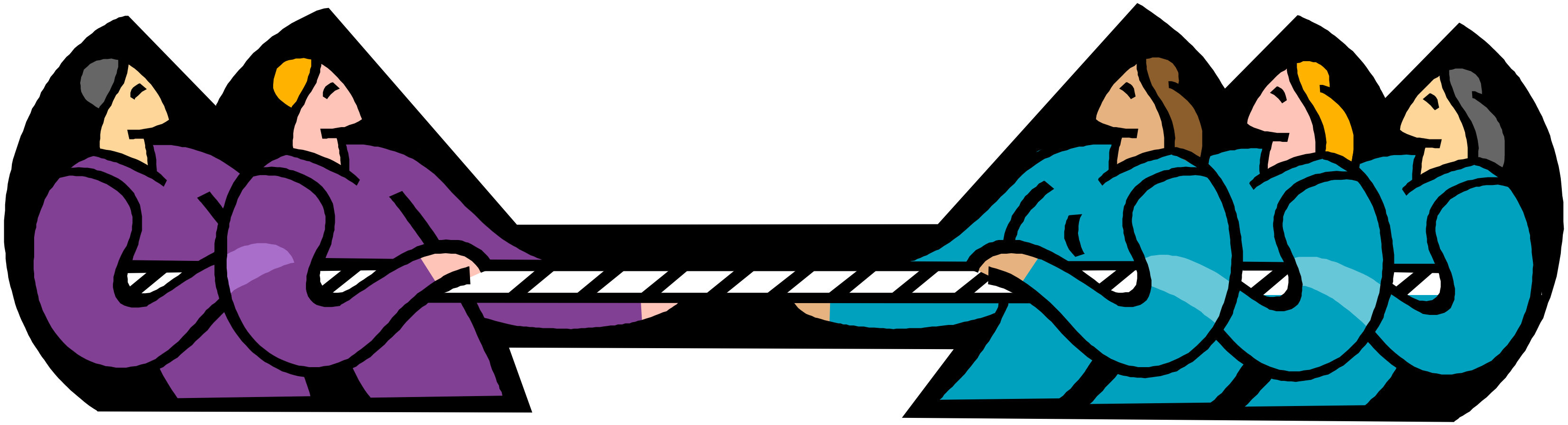
intel®

Software

Back Up



Legacy Vs. UEFI



Legacy Vs. UEFI

32-bit/16-Bit Real Mode

- No support for execution of IA-32 real mode
- Access to lower 1 MB of system memory only

- Flat mode
- Portable
- Single Binary

Fixed Resources for Working with Option ROMs

- Memory - EBDA limitations
- PMM have limitations

- Full access to system components
- Relocation ability
- Boot services provide Memory allocation

Issues Matching Option ROMs to their Devices

- UEFI Provides deterministic matching

Ties to PC-AT System Design

- Directly access HW and memory

- UEFI Has Well Defined Protocols

Ambiguities in Specification and Workarounds Born of Experience

- No clear specifications on how to write a legacy option ROM
- Workarounds because of incompatibilities
- Not clear which device will be boot device

- UEFI Specification followed
- UEFI allows for Platform overrides for flexibility

UEFI Membership

Adopters:

- Any entity wanting to implement the specification

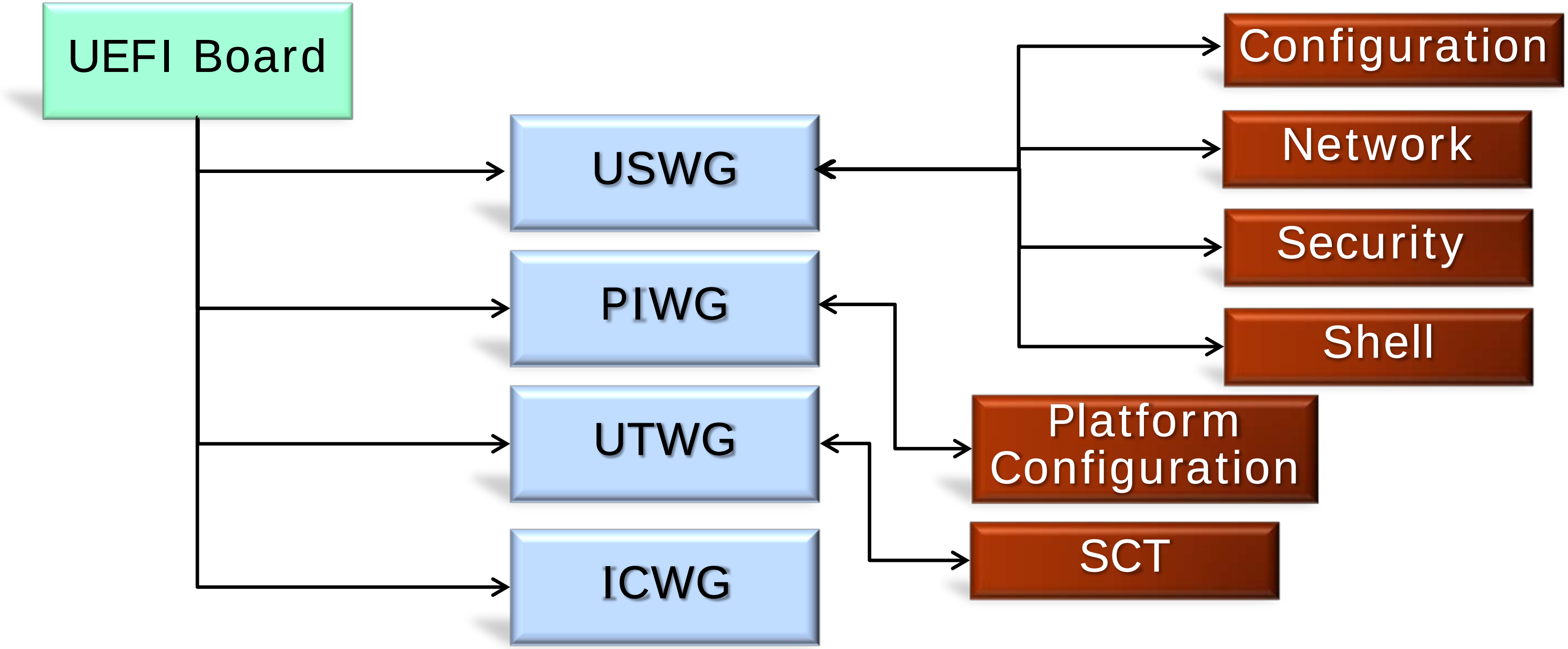
Contributors:

- Corporations, groups or individuals wanting to participate in UEFI
- Chance to join work groups and contribute to spec or test development
- Early access to drafts and work in progress

Promoters:

- Board of Directors (BoD) and corporate officers

How the Forum Works



Publications and other decisions are ratified by the board

Each work group approves and delivers different content for the public

Each sub-team focuses on specific topics and contributes material to the work group

Many Groups Working Together to Standardize Firmware

EFI 1.1 vs. UEFI 2.0

Items being changed or deprecated	UGA Protocols, SCSI Passthrough, USB Host Controller, Device I/O
New Items	Added Networking APIs, Intel® 64 binding, Service Binding, Tape I/O, Hash, DevicePath Utilities, CreateEventEx, UpdateCapsule, iSCSI Initiator, QueryCapsuleCapabilities, QueryVariableInfo, AuthenticationInfo
Items that are not changing	Loaded Image, Device Path, Driver Binding, Platform Driver Override, Bus Specific Override, Driver Configuration, Driver Diagnostics, Component Name, Simple Text Input, Simple Text Output, Simple Pointer, Serial IO, Load File, Simple File System, File Protocol, Disk IO, Block IO, Unicode Collation, PCI Root Bridge IO, PCI IO, SCSI IO, USB IO, Simple Network, PXE BC, Network Identifier Interface, BIS, Debug Support, Debug Port, Decompress, Device IO, EBC, RaiseTPL, RestoreTPL, AllocatePages, FreePages, GetMemoryMap, AllocatePool, FreePool, CreateEvent, SetTimer, WaitforEvent, SignalEvent, CloseEvent, CheckEvent, InstallProtocolInterface, ReinstallProtocolInterface, UninstallProtocolInterface, HandleProtocol, LocateHandle, LocateDevicePath, InstallConfigurationTable, LoadImage, StartImage, Exit, UnloadImage, ExitBootServices, GetNextMonotonicCount, Stall, SetWatchdogTimer, ConnectController, DisconnectController, OpenProtocol, CloseProtocol, OpenProtocolInformation, ProtocolsPerHandle, LocateHandleBuffer, LocateProtocol, InstallMultipleProtocolInterfaces, UninstallprotocolInterfaces, CalculateCrc32, CopyMem, SetMem, GetTime, SetTime, GetWakeupTime, SetWakeupTime, SetVirtualAddressMap, ConvertPointer, GetNextVariable, GetVariable, SetVariable, GetNextHighMonotonicCount, ResetSystem, ...

Unchanged Items, Deprecated Items, Changed Items, New Items

See § Appendix L UEFI 2.2 Spec.

UEFI 2.1 Published 2007

- Backlog of features not included in 2.0
  - Completed late 2006, UEFI Adoption 01/2007
- User interface presentation (HII)
  - Define interfaces that support integration of setup/configuration functions for motherboard and add-in devices
  - Removed the EFI_DRIVER_CONFIGURATION_PROTOCOL
- Security/Integrity related enhancements
  - Provide service interfaces for UEFI drivers that want to operate with high integrity implementations of UEFI
- Various other subject areas possible
  - More boot devices, error reporting, etc.

UEFI 2.0 vs. UEFI 2.1

Items changed or deprecated	GOP Protocols, Edid, Boot Service Table, SetVariable, WIN_CERTIFICATE_EFI_PKCS1_15, IP4/TCP4 variable. SCSI PassThru, iSCSI Initiator, Component Name 2, Unicode Collation, USB optional Status, SATA Device Path text presentation update, SCSI IO,Capsule Update, Device Path, Driver Configuration Protocol, *Dependency of PXE Base Code protocol on SNP
New Items	EFI HII Protocols(Font, String, Image, Database, Config Routing, Config Access, Form Browser2), APIs for Simple Text Input Ex, Absolute Pointer, ACPI Table API, *Platform to Driver Config, *Driver EFI Version, HW Error Record Support, Common Platform Error Record, Application Registration, *Authenticated Variable, *PCI Attribute change, Signaling on configuration change, RT Services w/ Interrupts enabled, FW Storage Device Path for PIWG
Items Not changing	Loaded Image, Driver Binding, Platform Driver Override, Bus Specific Override, Driver Diagnostics, Component Name, Simple Text Output, Simple Pointer, Serial IO, Load File, Simple File System, File Protocol, Disk IO, Block IO, Unicode Collation, PCI Root Bridge IO, PCI IO, USB IO, Simple Network, PXE BC, Network Identifier Interface, BIS, Debug Support, Debug Port, Decompress, Device IO, EBC, RaiseTPL, RestoreTPL, AllocatePages, FreePages, GetMemoryMap, AllocatePool, FreePool, CreateEvent, SetTimer, WaitforEvent, SignalEvent, CloseEvent, CheckEvent, InstallProtocolInterface, ReinstallProtocolInterface, UninstallProtocolInterface, HandleProtocol, LocateHandle, LocateDevicePath, InstallConfigurationTable, StartImage, Exit, UnloadImage, ExitBootServices, GetNextMonotonicCount, Stall, SetWatchdogTimer, ConnectController, DisconnectController, OpenProtocol, CloseProtocol, OpenProtocolInformation, ProtocolsPerHandle, LocateHandleBuffer, LocateProtocol, InstallMultipleProtocolInterfaces, UninstallprotocolInterfaces, CalculateCrc32, CopyMem, SetMem, GetTime, SetTime, GetWakeupTime, SetWakeupTime, SetVirtualAddressMap, Networking APIs, Intel® 64 binding, Service Binding, Tape I/O, Hash, CreateEventEx, QueryVariableInfo, AuthenticationInfo

Unchanged Items, Deprecated Items, Changed Items, New Items

* Features not supported in EDK I 1.05

UEFI 2.2 Published 2008

- Follow-on material from existing 2.1 content
  - Backlog that needed more gestation time
  - Includes incremental changes to UEFI 2.1a, 2.1b, 2.1c
- Networking IPv6 PXE+, IPSec
- Driver Signing
  - Expands the types of signatures recognized by UEFI SHA-1, SHA-256, RSA2048/SHA-1, RSA2048/SHA-256 & Authenticode
- User Identification
  - Standard framework for user-authentication devices such as smart cards, smart tokens & fingerprint sensors
- User Interface Updates
  - New form type for support of non-UEFI configuration standards (e.g. DMTF)
- Others – Many IHV driven Additions

UEFI 2.3 Published 2009

- Follow-on material from existing 2.2 content
- Addition of Firmware Management Protocol.
- Add more processor bindings to UEFI
- Added new HII callback types
- Add translator field to some of the USB2 Host Controller protocol functions
- Added a section which described “Non-removable media boot behavior”
- Fixed definitions for UEFI_CONFIG_LANG and UEFI_CONFIG_LANG_2

*

UEFI is the only place where these future technologies are defined

UEFI 2.3.1 Specification Update

Security

- Authenticated Variable & Signature Data Base
- Key Management Service (KMS)
- Storage Security Command Protocol for encrypted HDD

Network

Netboot6 client use DUID-UUID to report platform identifier

Interoperability

- New FC and SAS Device Path
- FAT32 data region alignment
- HII clarification & update
- HII Modal Form

Performance

Non-blocking interface for BLOCK oriented devices

Technology

USB 3.0

Maintenance

User Identifier, etc.

UEFI 2.3.1 Enables More Security Support

Overview of Differences
PI 1.0 vs. Framework[1] Components

Component		Actions / Exceptions
✓	Compatibility	Do not access internals of the firmware files Do not use ReportStatusCode
✓	PEI File System	Minor change to the file header and firmware volume header
✓	PPI Updates	PCI PPI for Extended PCI-express New PPI – Terminate End of Temp Memory
✓	DXE Service Table	Removed Report Status Code service
✓	New Architectural Protocol	Capsule AP / QueryVariableInfo
✓	HOB definitions	More Firmware volume information Remove Capsule HOB definition

PI 1.0 Introduces Standards To Early Boot

Details on what is in PI1.1

- Completed End of 2007
- PI1.1 includes:
  - PCI Specifications – root bridge, host bridge, hot plug, override
  - MP protocol for DXE
  - SMBIOS table creation API
  - S3 boot script infrastructure
  - SMM & PMI Component Interface specifications

Overview of Differences
PI 1.1 Vs. Framework Components

	Component	Actions / Exceptions
✓	**SMM**	SMM driver model change. Port code
✓	**S3**	New execution requirements for native callbacks. Some interfaces removed
✓	**PCI**	New event. Should be able to enhance former implementation
✓	**MP**	Clean-up. Port to use new API
✓	**DXE**	Cleaned-up DXE to be UEFI 2.0 RT compatible
✓	**SMBIOS table creation**	The framework[1] used data hub. This is a new API

PI 1.1 Expands PI 1.0 infrastructure w/ more building blocks

Details on what is in PI 1.2

- Adopted in 2009
- Support For Large Firmware Files And Firmware File Sections
- Platform Initialization Status codes
  - Enable system components to report information about their current state
- Platform Configuration Report Status Code
  - Generic status code driver in each phase.
- Platform Configuration Database (PCD) Protocol
  - A variety of current platform settings or directives that can be accessed by a driver or application.
  - Abstraction for accessing configuration content in the platform
- Platform Initialization Standards
  - IDE Controller Initialization Protocol
  - ACPI System Description Table Protocol
  - Super I/O Protocol
  - Processor I/O Protocol
  - Legacy Region Protocol

Overview of Differences
PI 1.2 Vs. Framework Components

	Component	Actions / Exceptions
✓	**Large Firmware Files**	>16M file can be integrated into FFS file.
✓	**PCD**	PCD services for Dynamic EX.
✓	**Report Status Code**	Report Status Code Router and Handler.
✓	**ACPI Protocol**	Create ACPI system description tables.
✓	**DXE**	IDE Controller and Legacy Region added.

PI 1.2 Expands PI 1.1 infrastructure w/ more building blocks

Back

intel®

Software

Optimization Notice

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

Notice revision #20110804