Unified Extensible Firmware Interface (UEFI) Overview

Intel Corporation
Software and Services Group

Agenda

- BIOS Background
- Why Change to UEFI
- UEFI Overview
- UEFI Technical Overview
  - Boot Support
  - UEFI Terminology
  - UEFI Driver Design
  - EFI 1.1 and UEFI 2.0/2.1/2.2 differences
  - EFI Byte code
- Technology not addressed by UEFI

What's Legacy BIOS

- Basic Input - Output System for original IBM PC/XT and PC/AT
- Originated in 1980s
- Based on 8086 architecture
- A group of clearly defined OS-independent interface for hardware
  - Int10 for Video service
  - Int13 disk service
  - Int16 keyboard service
  - Int18 BIOS ROM loader
  - Int19 bootstrap loader
- Availability of MS-DOS outside of IBM allowed applications to run equally well across different brands of box "PC clones".

EFI / UEFI History Time Line

		PCI	Intel® Itanium® Platforms 64 bit develops	EFI 1.02	EFI 1.10 - 2002	UEFI 2.0 PI 1.0 Web: UEFI.org	UEFI 2.1 PI 1.1	UEFI 2.2 Shell 2.0
PC Era	IBM 16 Bit BIOS		EFI only way to boot Itanium® Platforms	Intel Sample Imple-menta-tion	EDK - Tianocore .org Open Source	EDK I 1.01	EDK I 1.03 EDK I 1.04 EDK II	EDK I 1.04 P1
1980s	1985	1990	1998	1999	2004	2006	2007	2008

- Open Source EFI Developer Kit (EDK) http://www.tianocore.org
- UEFI Specifications - http://www.uefi.org

Agenda

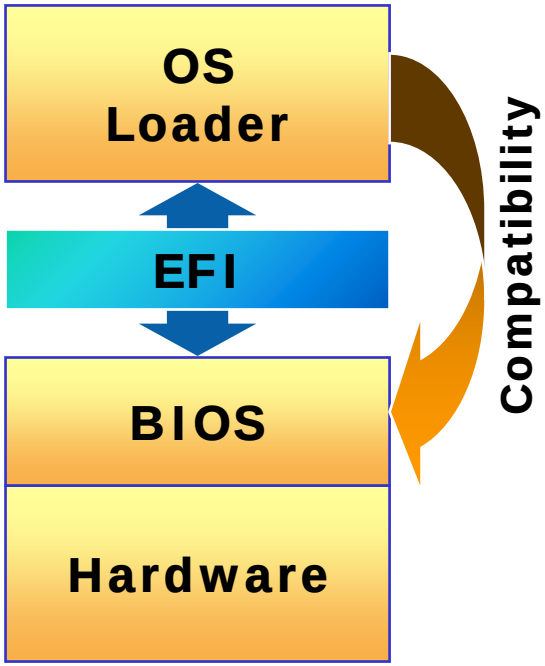
- BIOS Background
- Why Change to UEFI *–see Backup*
- UEFI Overview
- UEFI Technical Overview
  - Boot Support
  - UEFI Terminology
  - UEFI Driver Design
  - EFI 1.1 and UEFI 2.0 - 2.2 differences
  - EFI Byte code
- Technology not addressed by UEFI

Agenda

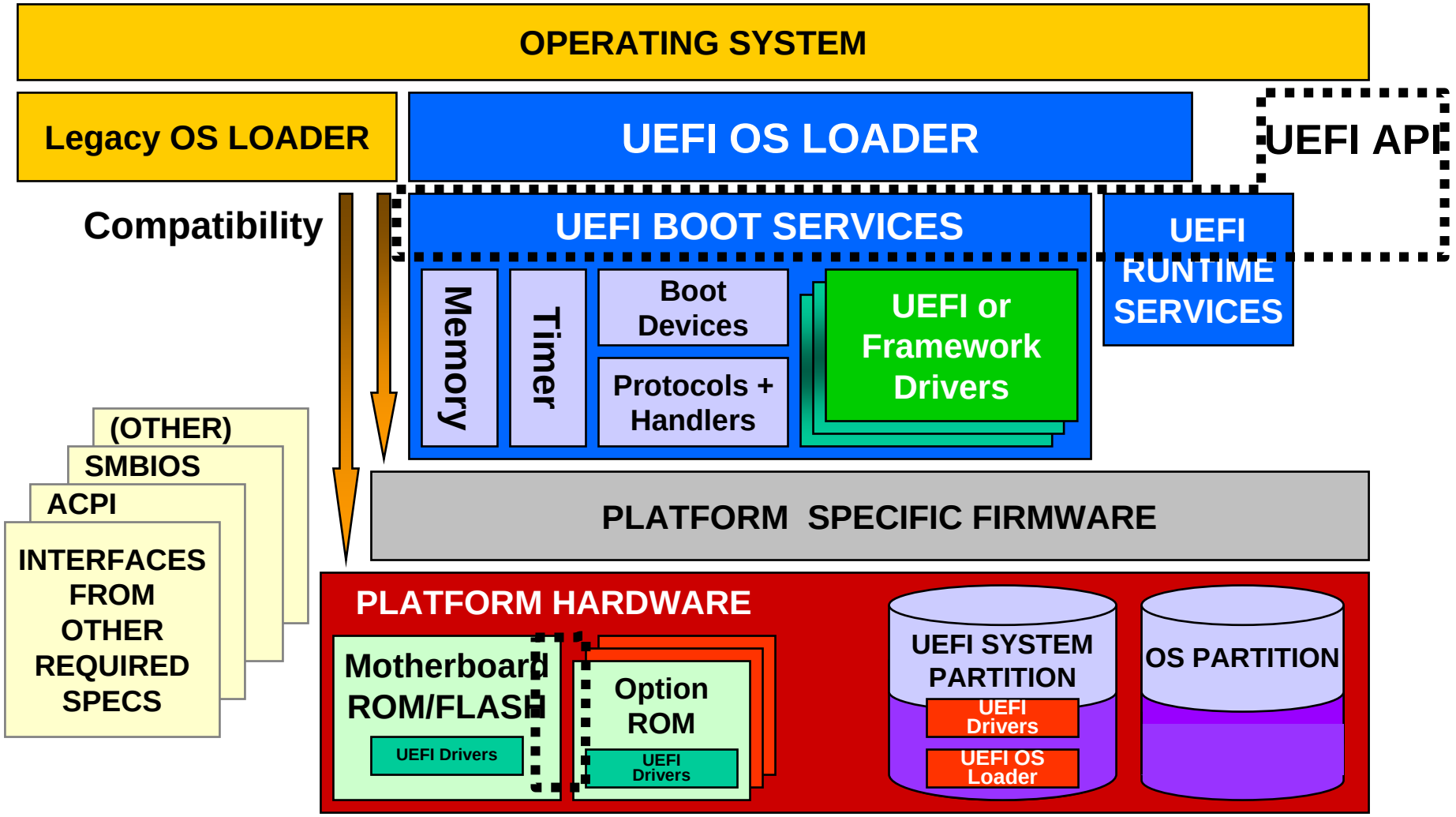
- BIOS Background
- Why Change to UEFI
- UEFI Overview
- UEFI Technical Overview
  - Boot Support
  - UEFI Terminology
  - UEFI Driver Design
  - EFI 1.1 and UEFI 2.0 - 2.2 differences
  - EFI Byte code
- Technology not addressed by UEFI

What is UEFI?

- Unified (EFI) / Extensible Firmware Interface (EFI)
- UEFI is an interface specification
- Abstracts BIOS from OS
  - Decouples development
- Compatible by design
  - Evolution, not revolution
- Modular and extensible
  - OS-Neutral value add
- Provide efficient Option ROM Replacement
  - Common source for multiple CPU architectures
- Complements existing interfaces



Concept



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

- Promoters
  - OEMs: 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 - 2.2 specification released.
  - Forum will evolve, extend, and add any new functionality as required
  - Intel contributed EFI 1.10 SCT being used as starting base for UEFI conformance tests (UEFI SCT 2.1 released TBD)

Purpose: Worldwide adoption and promotion of UEFI specifications

UEFI Membership

Promoters: board and corporate officers

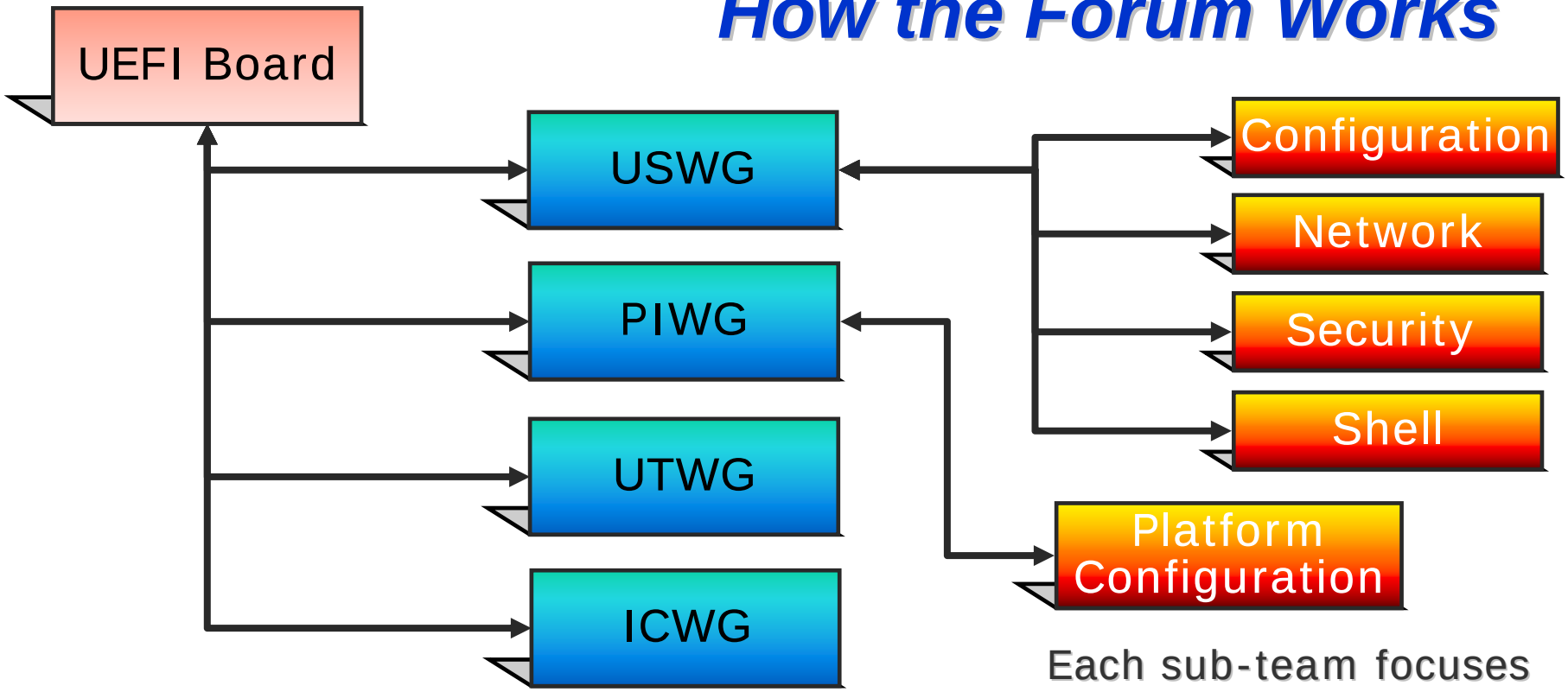
Contributors:

- 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

Adopters:

- Any entity wanting to implement the specification

How the Forum Works



Publications/Decisions ratified by the board

Each work group approves/delivers different content to the public.

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

Many groups working together to Standardize Firmware

Agenda

- BIOS Background
- Why Change to UEFI
- UEFI Overview
- UEFI Technical Overview
  - Boot Support
  - UEFI Terminology
  - UEFI Driver Design
  - EFI 1.1 and UEFI 2.0 - 2.2 differences
  - EFI Byte code
- Technology not addressed by UEFI

System Partition

Architectural Sharing

- System partition
- Location for OS loaders
- Applications and drivers

FAT32 Format

- FAT32 spec now “public”
- Tried and tested format
- Readily available tools

Interoperability layout

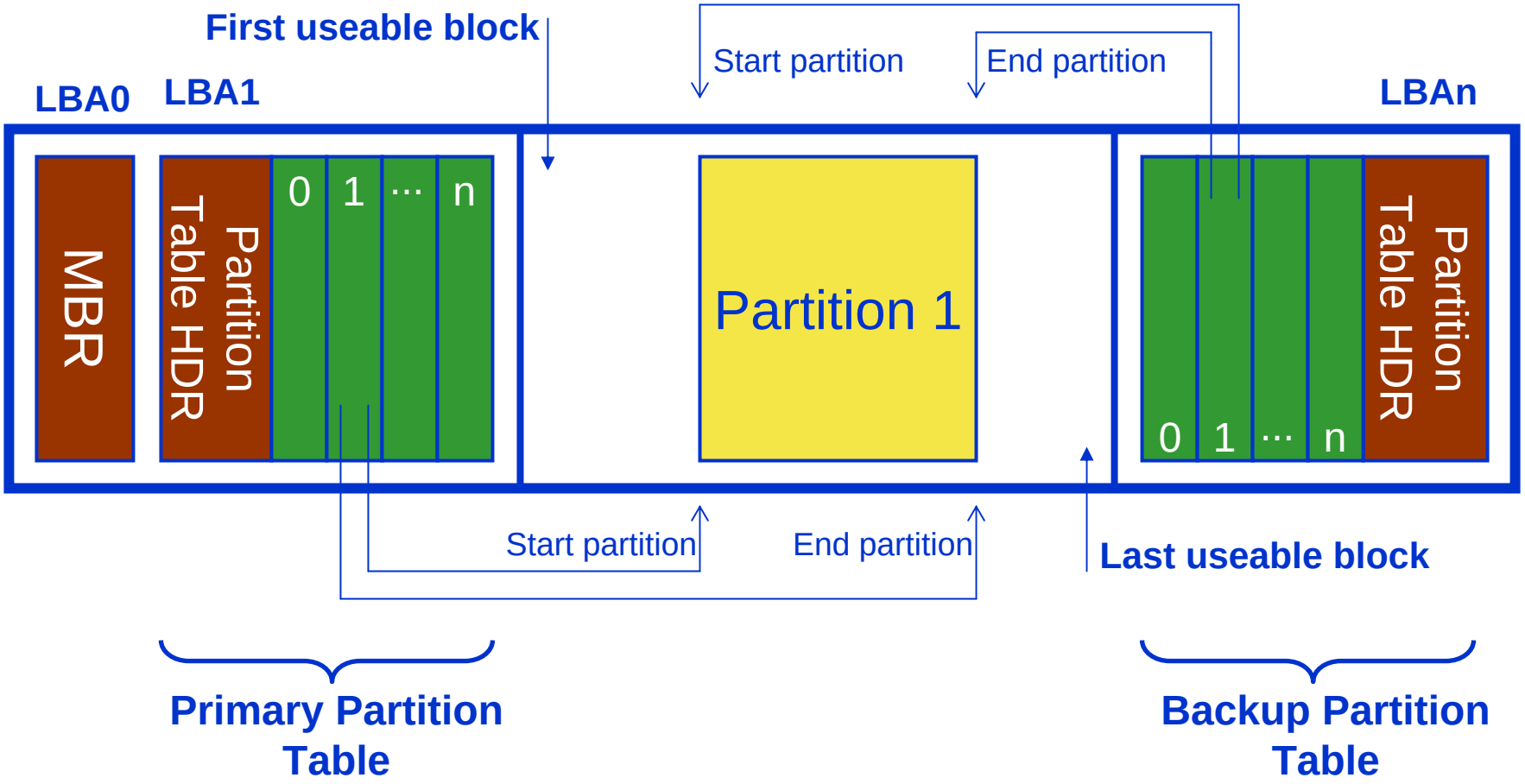
- Multiple system partitions
- Supports multiple OS installs

New Partition Structure

- 64 bit partition sizes
- Unlimited # of partitions
- Co-exists w/ legacy MBR

Designed for Flexibility

New Partition Structure



Boot device support

- Hard disk
- Removable media
  - CD-ROM, DVD-ROM
    - El Torito 1.0 “No emulation”
  - Floppy, LS-120 SuperDisk*, Iomega* Zip, Fujitsu* MO etc.
- Network
  - PXE BIOS support specification (WfM)
- Future media via extensibility methods

Full Device Support

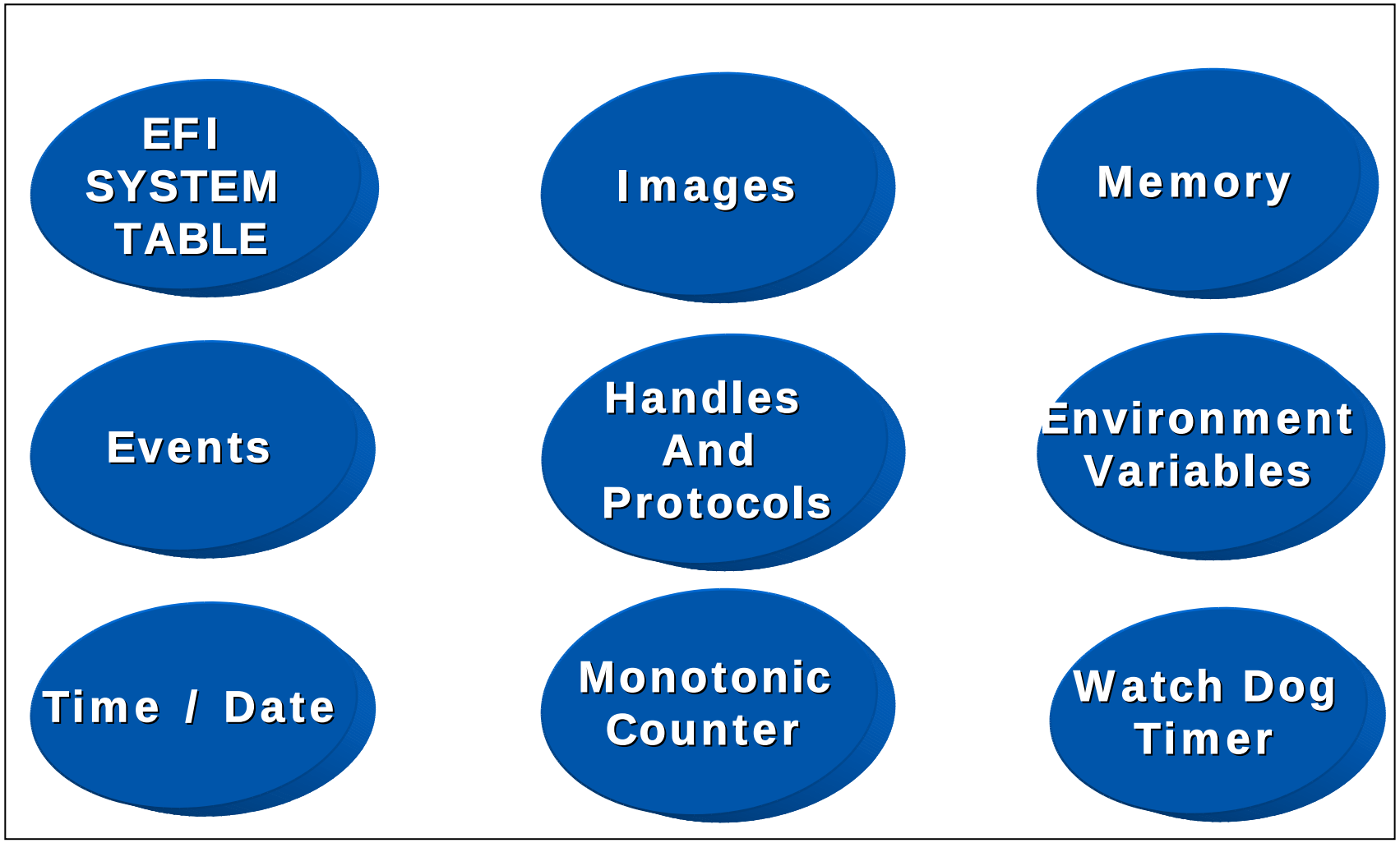
Agenda

- BIOS Background
- Why Change to UEFI
- UEFI Overview
- UEFI Technical Overview
  - Boot Support
  - UEFI Terminology
  - UEFI Driver Design
  - EFI 1.1 and UEFI 2.0 - 2.2 differences
  - EFI Byte code
- Technology not addressed by UEFI

UEFI Specification - Key Concepts

- Objects - manage system state, including I/O devices, memory, and events
- The UEFI System Table - data structure with data in-formation 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

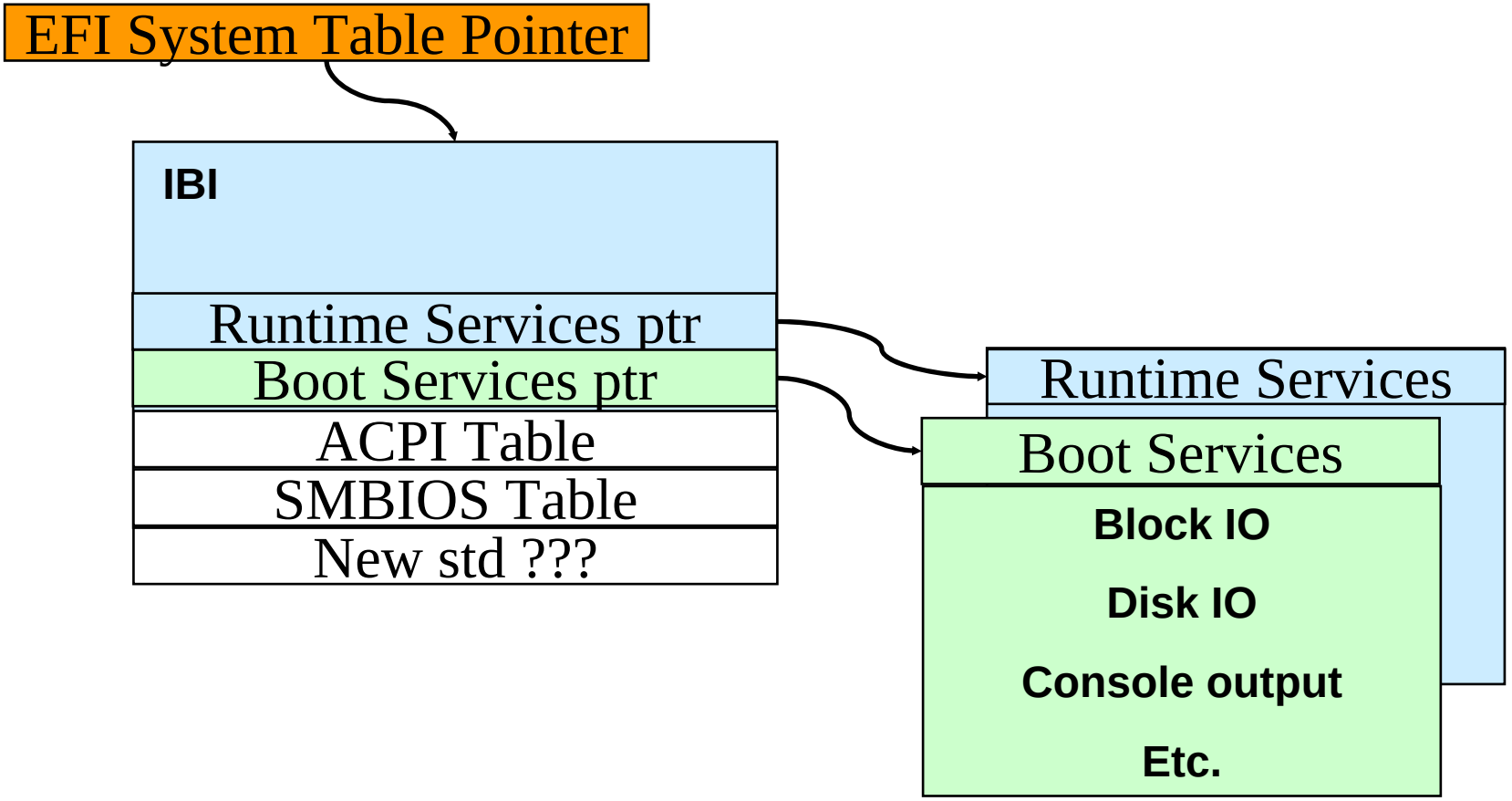
Objects Managed by UEFI Firmware



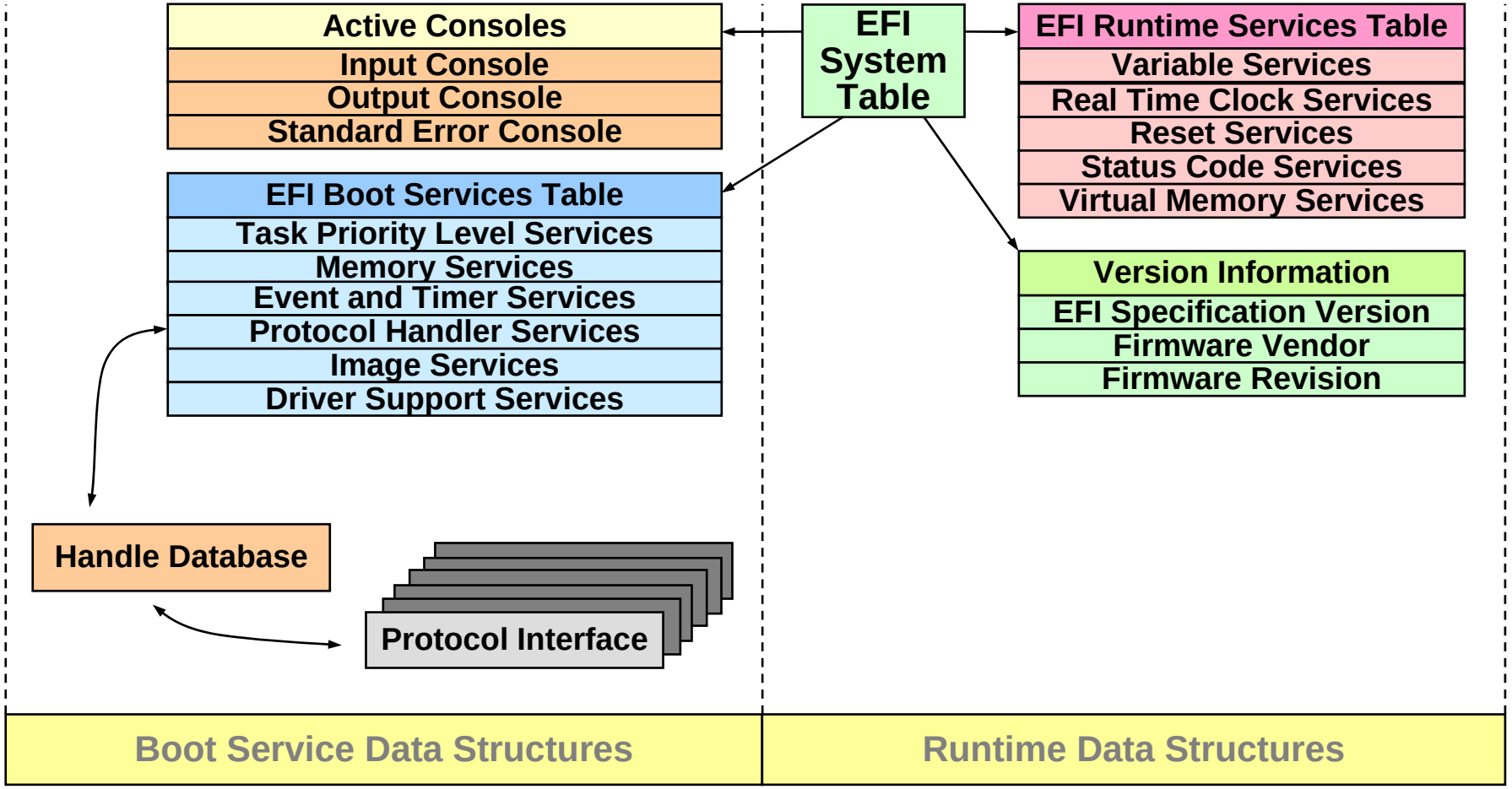
What is the UEFI System Table

- Firmware implementation information
  - Read only for peripheral drivers
  - Specification version
  - Interface to UEFI protocols
  - Interface to other standards…

UEFI System Table



UEFI Data Structures - EFI System Table



GUID

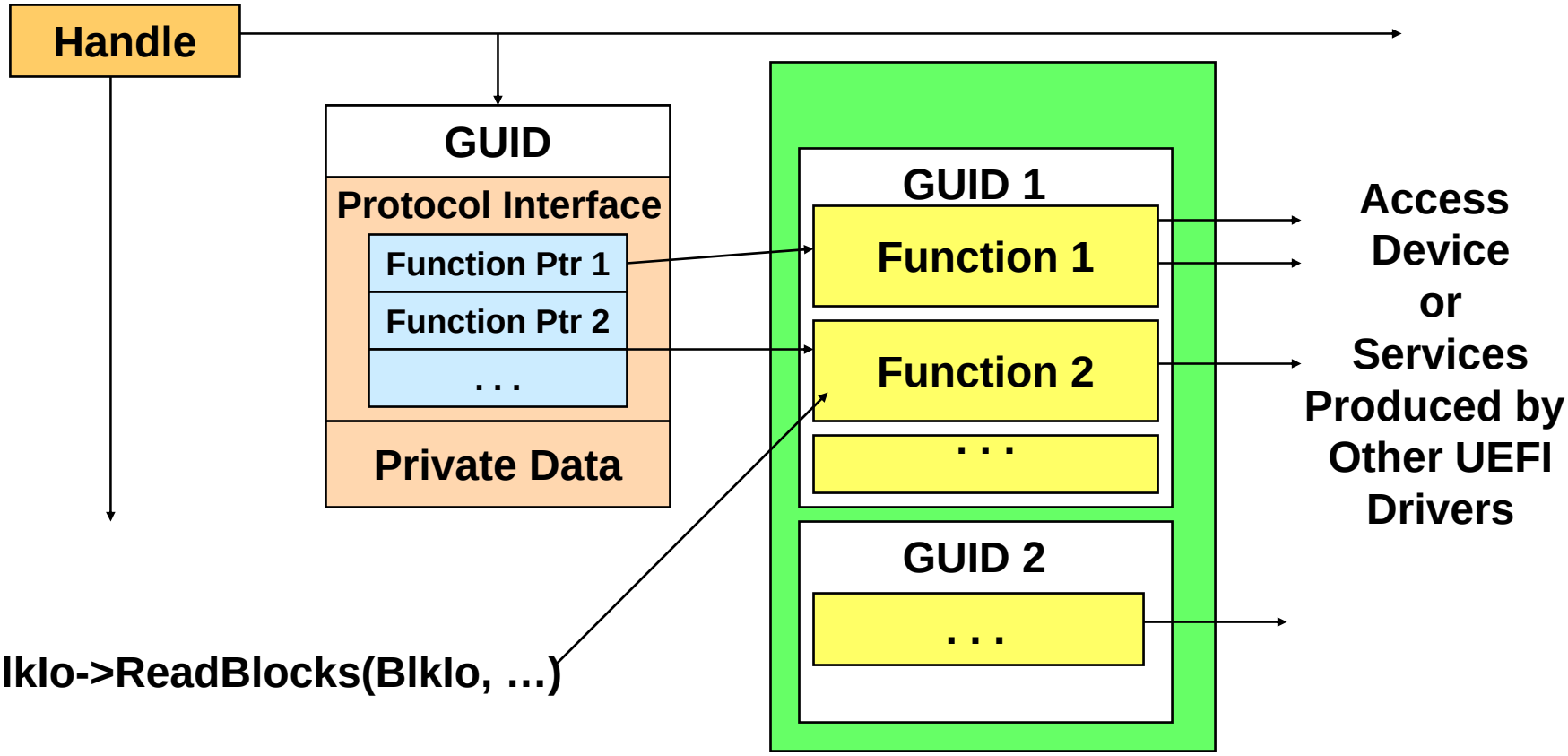
- "Globally" Unique Identity
  - 128-bit quantity defined by Wired for Management WfM 2.0 specification **
- Used to identify protocols
  - 1:1 with interfaces
- Regulate extension mechanism
  - Documented in the spec
  - Added through drivers

Safe co-existence of 3rd party extensions

** http://www.intel.com/labs/manage/wfm/wfmspecs.htm

Protocols (API)

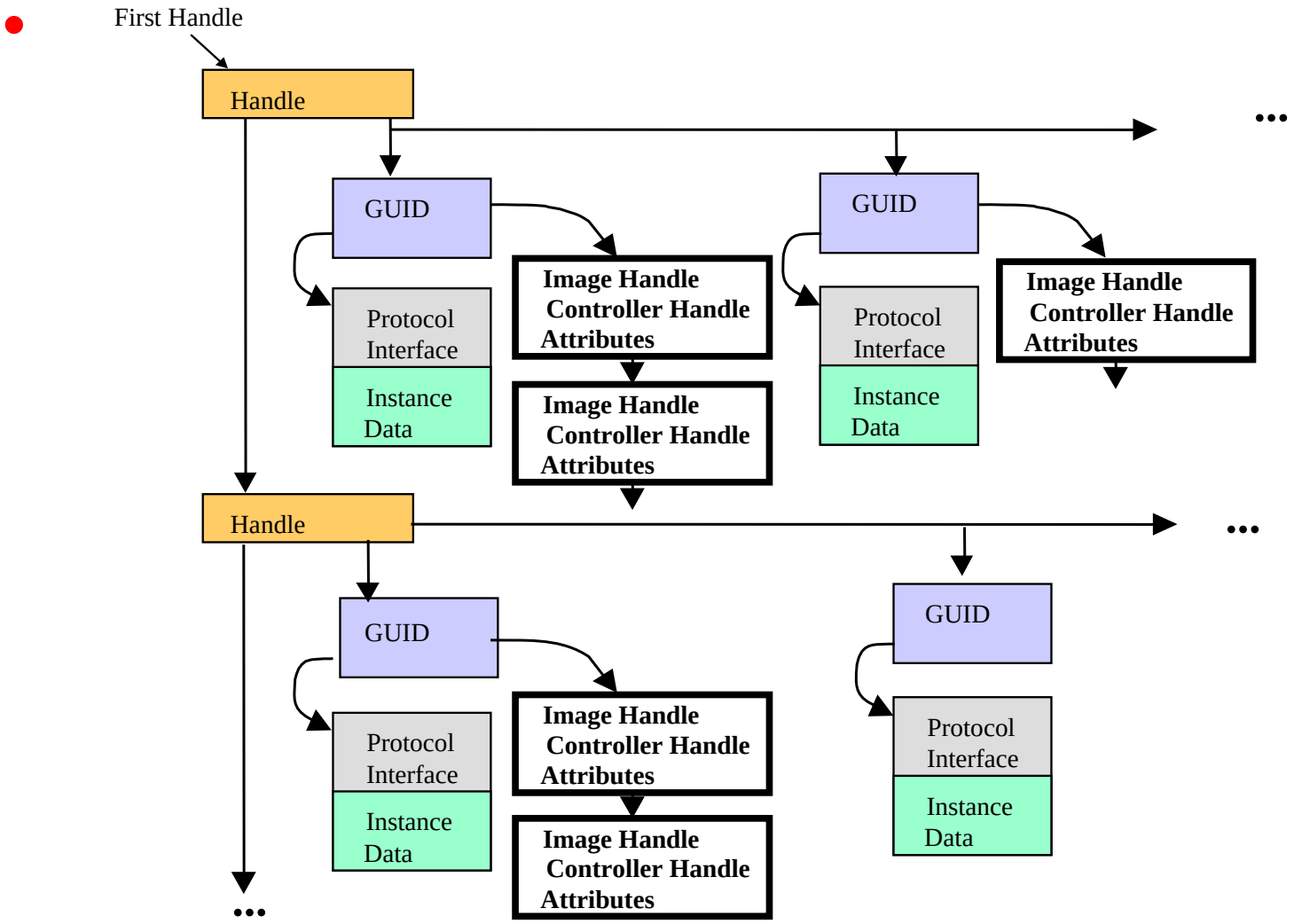
- GUID, Interface Structure, Services
  - DEVICE_PATH, DEVICE_IO, BLOCK_IO, DISK_IO, FILE_SYSTEM, SIMPLE_INPUT, SIMPLE_TEXT_OUTPUT, SERIAL_IO, PXE_BC, SIMPLE_NETWORK, LOAD_FILE, UNICODE_COLLATION



Handles

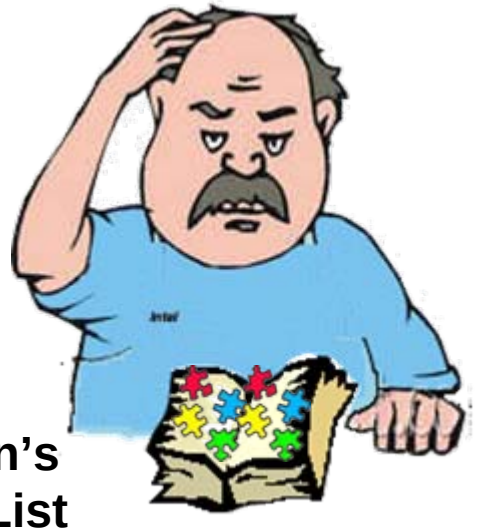
- All protocols have a handle which is associated with the protocol
- Every device and executable image in UEFI has a handle protocol in the handle database
- Every boot device must have a device path protocol to describe it

Handle Protocol Database

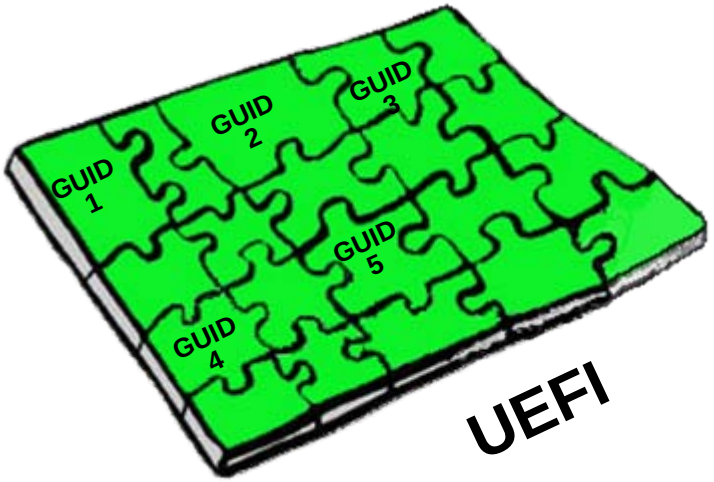


Legacy BIOS vs UEFI

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



Ralf Brown's Interrupt List



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

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 device path
- 6 types of device paths:
  - Hardware
  - ACPI – UID/HID of device in AML
  - Messaging – i.e. LAN, Fiber Channel, ATAPI, SCSI, USB
  - Media – i.e. Hard Drive, Floppy or CD-ROM
  - EDD 3.0 boot device – see EDD 3.0 spec int13 48
  - End of hardware – marks end of device path

Device Path Examples

- Acpi(PNP0A03,0)/Pci(1F|1)/Ata(Primary,Master)/HD(Part3, Sig00110011)
- Acpi(PNP0A03,1)/Pci(1E/0)/Pci(0|0)/Mac(0002B3647D69)
- Acpi(PNP0A03,0)/Pci(1F|0)/Acpi(PNP0501,0)/Uart(115200 81)

Boot Services

- Events and notifications
  - Polled devices, no interrupts
- Watchdog timer
  - Elegant recovery
- Memory allocation
- Handle location – for finding protocols
- Image loading
  - Drivers, applications, OS loader

Complete and size efficient

Runtime Services

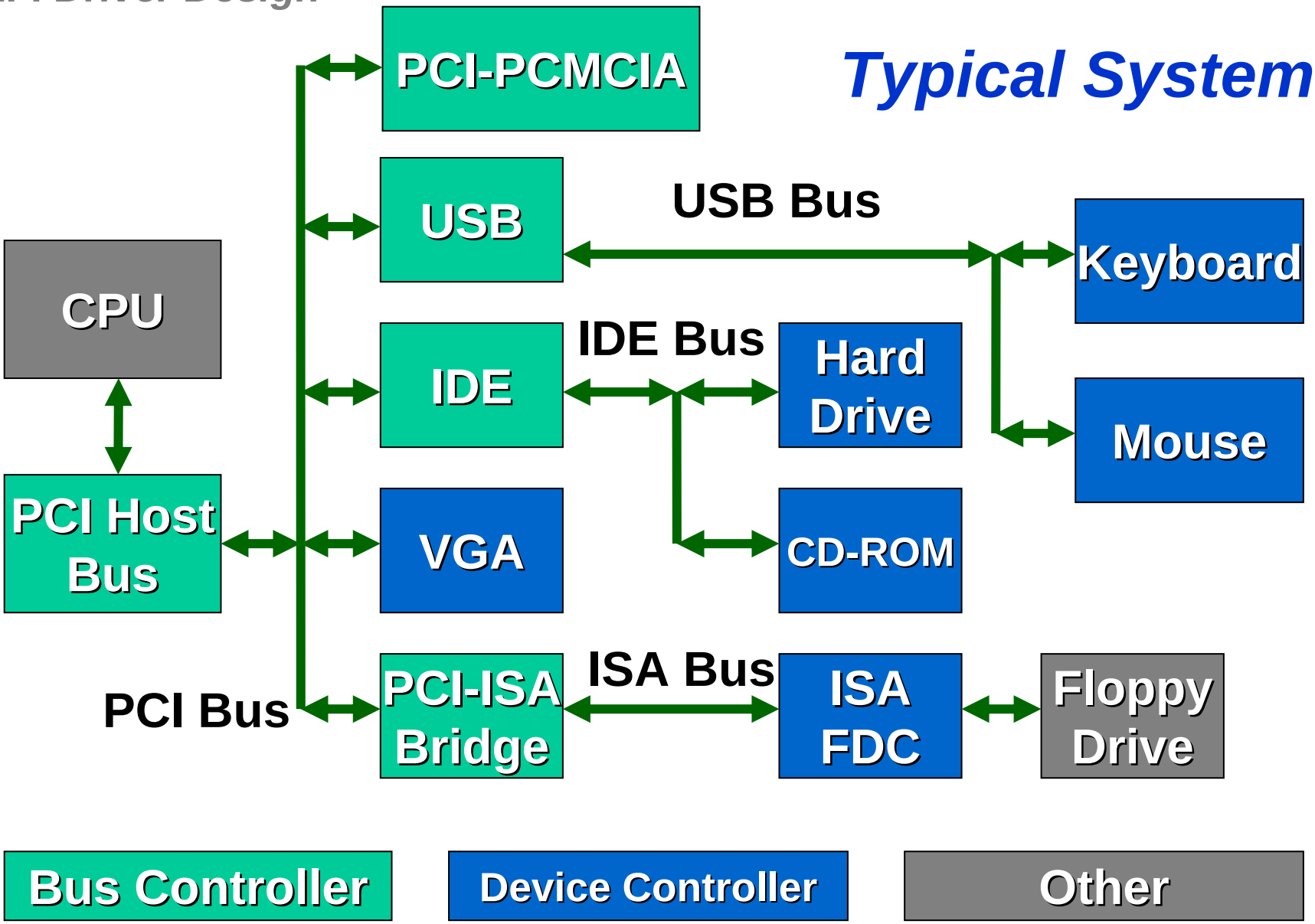
- Services available at both boot time and runtime
- Timer, Wakeup alarm
  - Allows system to wake up or power on at a set time.
- Variables
  - Boot manager handshake
- System reset

Minimal set to meet OSV needs

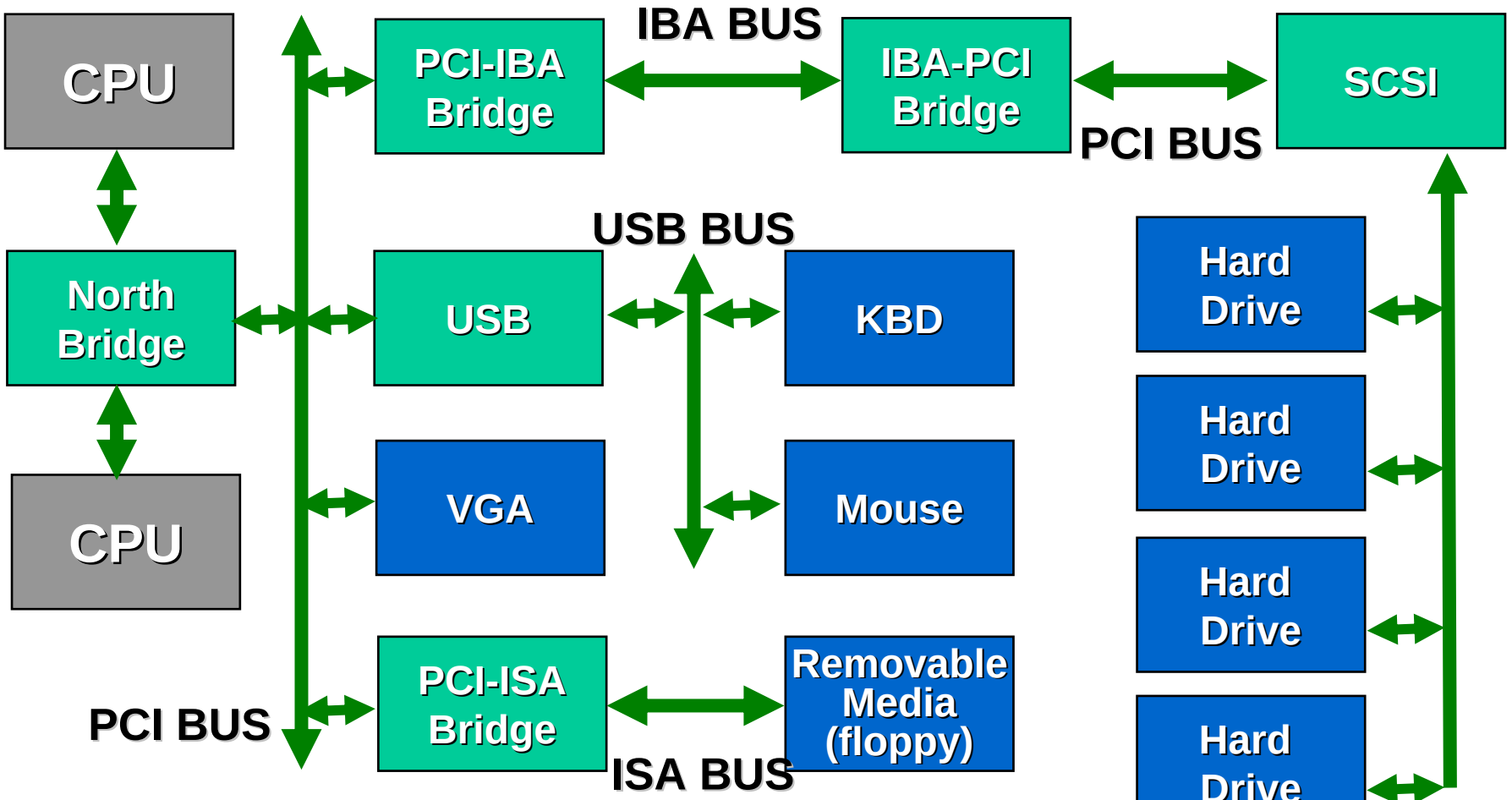
Agenda

- BIOS Background
- Why Change to UEFI
- UEFI Overview
- UEFI Technical Overview
  - Boot Support
  - UEFI Terminology
  - UEFI Driver Design -*See also Backup*
  - EFI 1.1 and UEFI 2.0 - 2.2 differences
  - EFI Byte code
- Technology not addressed by UEFI

Typical System



Complex System Example

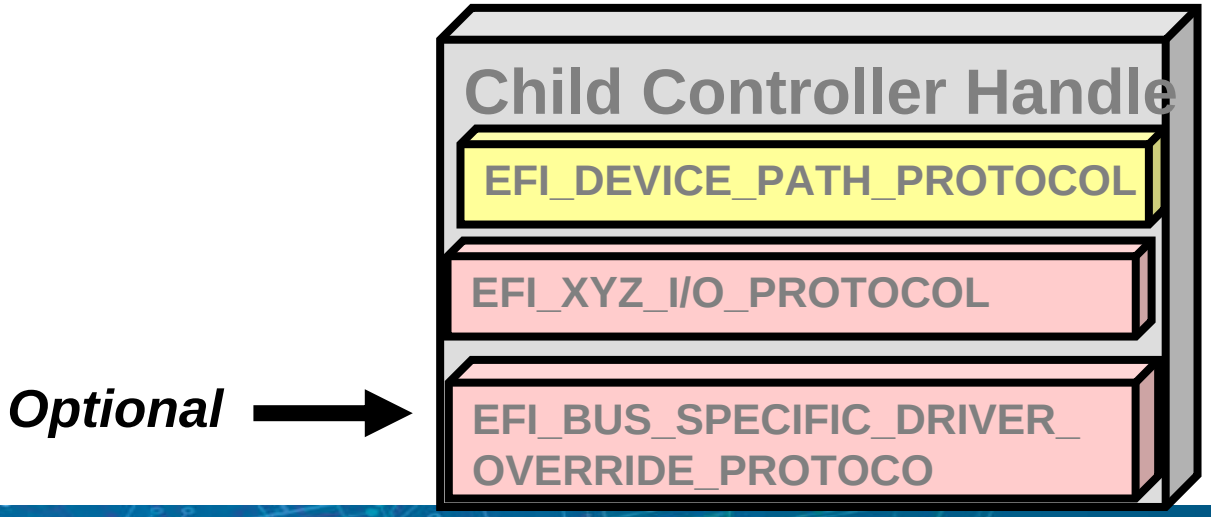


- **Manageable by UEFI Driver Model**

See § 2.5 UEFI 2.1 Spec.

Bus Driver

- Consumes Parent Bus I/O Abstraction(s)
- Initializes Bus Controller
- Allocates Resources for Child Controllers
- Creates Handles for Child Controllers
- Loads drivers from Option ROMs if present

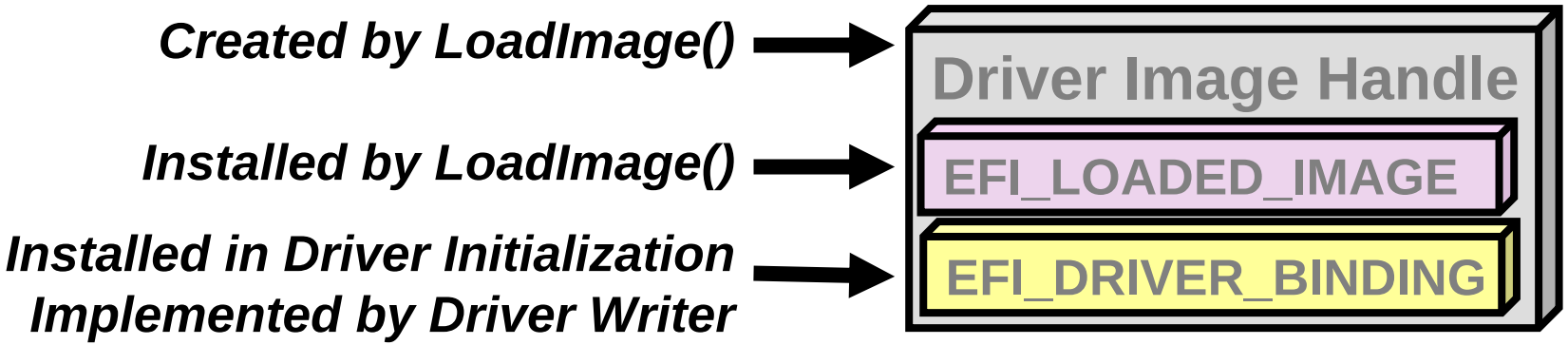


Device Driver

- Consumes Bus I/O Abstraction(s)
- Initializes Device Controller
- Produces Device Abstraction(s)
  - Block I/O Protocol
  - Simple Text Output Protocol
  - Simple Network Protocol
- Does Not Create Any Child Handles
- Can still be a “Parent” Controller

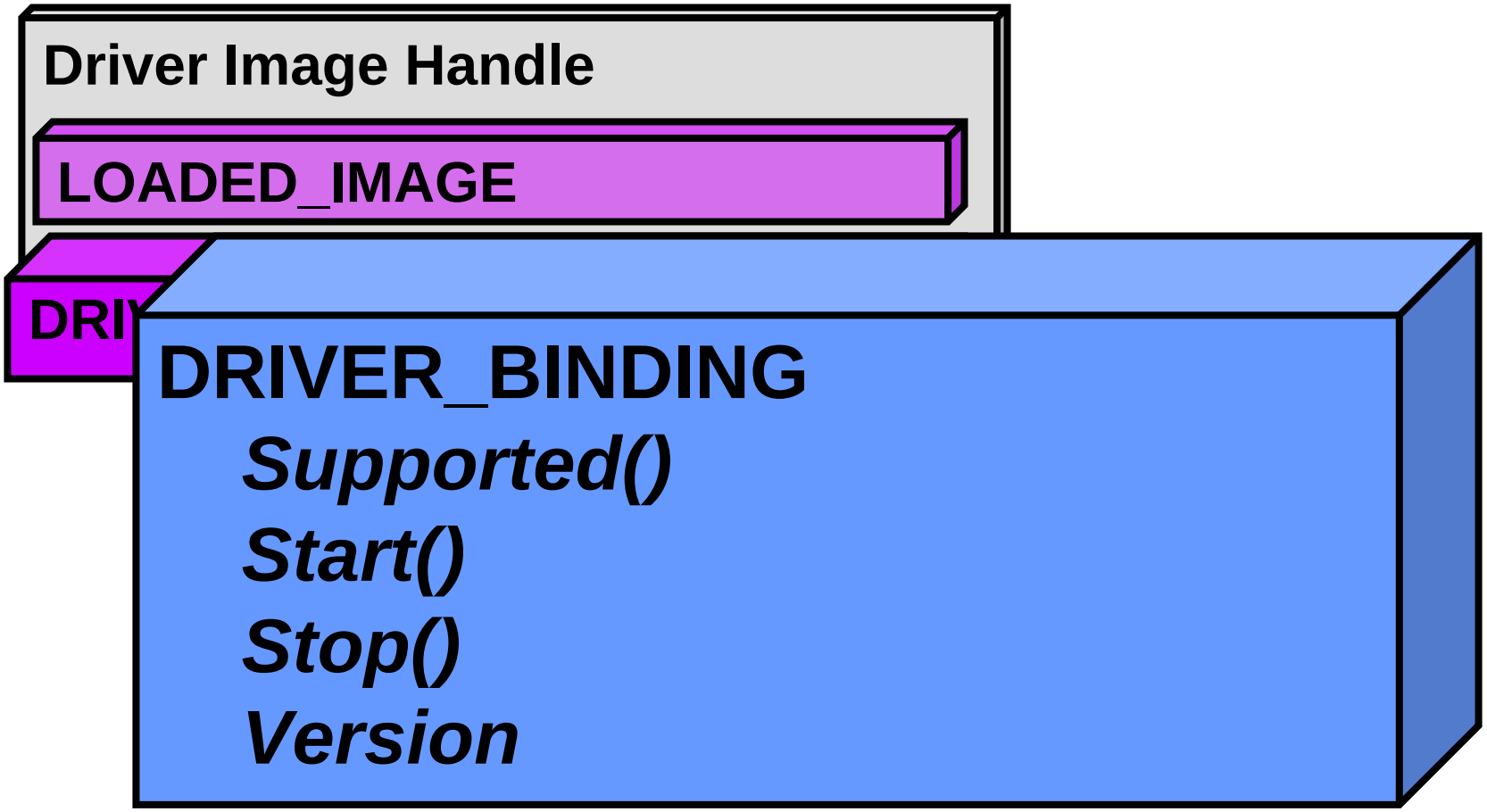
Driver Initialization

- EFI Driver Handoff State
- Not Allowed to Touch Hardware Resources
- Installs Driver Binding on Driver Image Handle

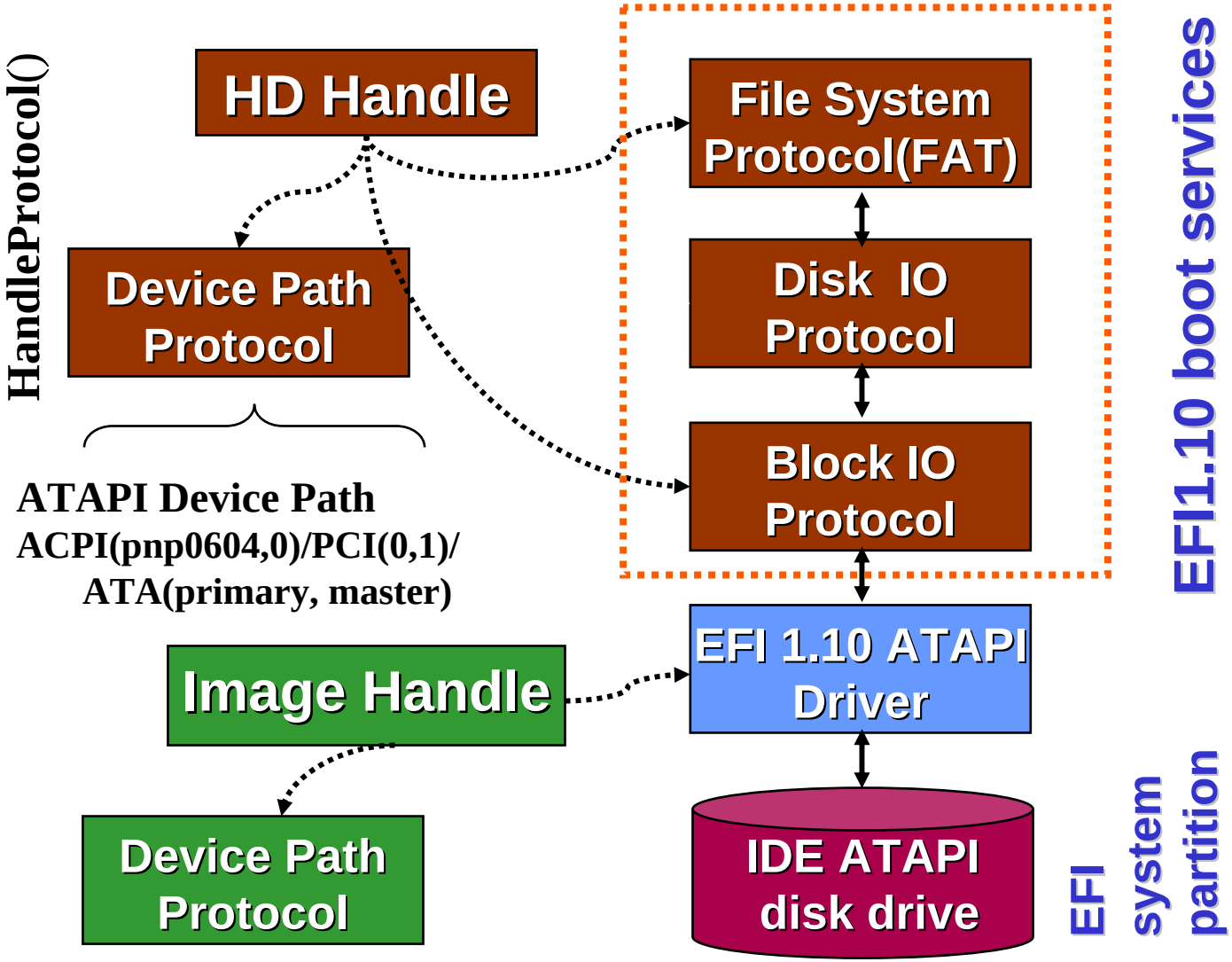


Registers Driver for Later Use

Driver Binding Protocol



Example: UEFI ATAPI Driver Stack



Agenda

- BIOS Background
- Why Change to UEFI
- UEFI Overview
- UEFI Technical Overview
  - Boot Support
  - UEFI Terminology
  - UEFI Driver Design
  - EFI 1.1 and UEFI 2.0 - 2.2 differences
  - EFI Byte code
- Technology not addressed by UEFI

EFI 1.1 vs. UEFI

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 AND Deprecated Items, New Items

UEFI 2.1 Published 2007

- A smaller update than the 2.0 iteration
  - Backlog that needed more gestation time
  - 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
- 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 2.2 Published 2008

- Follow-on material from existing 2.1 content
  - Backlog that needed more gestation time
- 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 is the only place where these future technologies are defined

What's supported?

- The community is in the progress of switching from EFI 1.1 to UEFI 2.x.
  - EFI 1.1 protocols Starting Point
  - UEFI 2.0 protocols since 2007 to replace EFI 1.1 protocols added to open source community Mid 2007
  - UEFI 2.1 protocols added beginning of 2007 and then added to the open source community End 2007 (EDK I 1.04)
  - New SCT UEFI 2.1 Mid 2008
  - UEFI 2.2 support through EDK II

Agenda

- BIOS Background
- Why Change to UEFI
- UEFI Overview
- UEFI Technical Overview
  - Boot Support
  - UEFI Terminology
  - UEFI Driver Design
  - EFI 1.1 and UEFI 2.0/2.1/2.2 differences
  - EFI Byte code
- Technology not addressed by UEFI

Why Do More Than UEFI?

- EFI Sample Implementation on IA32 too large for today's desktop, mobile and handheld flash allotment.
- Opportunity to drive C-code and modularity much closer to Si with all architectures in mind.
- Make IA firmware development competitive with RISC implementations.

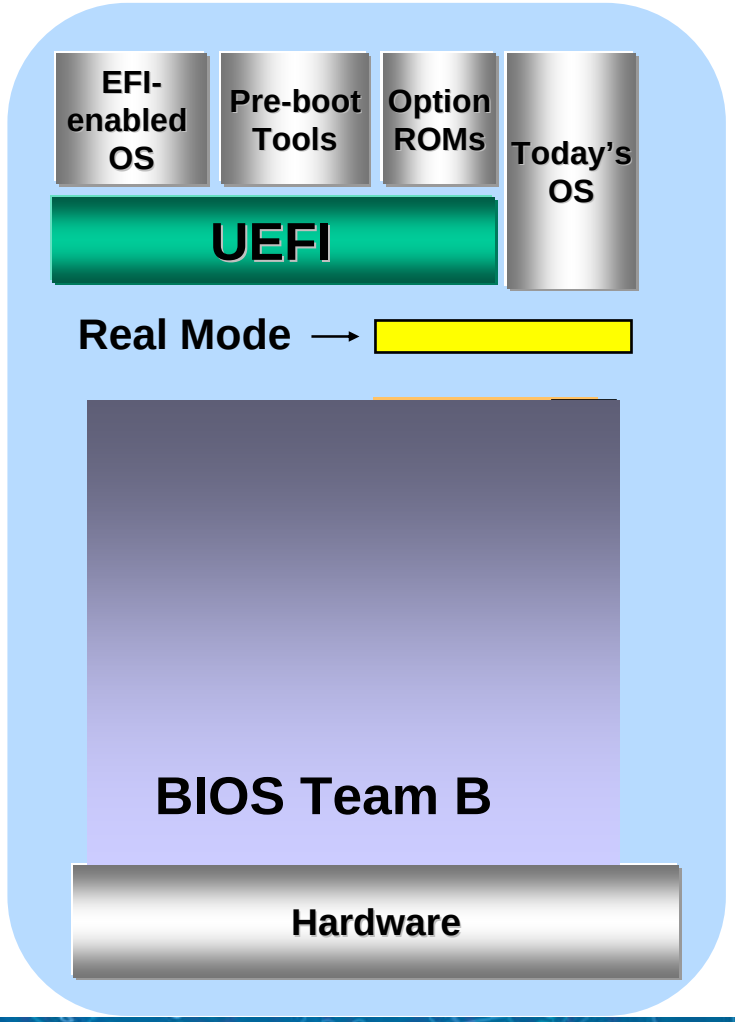
UEFI on BIOS is NOT Good Enough

Technology not addressed by UEFI

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

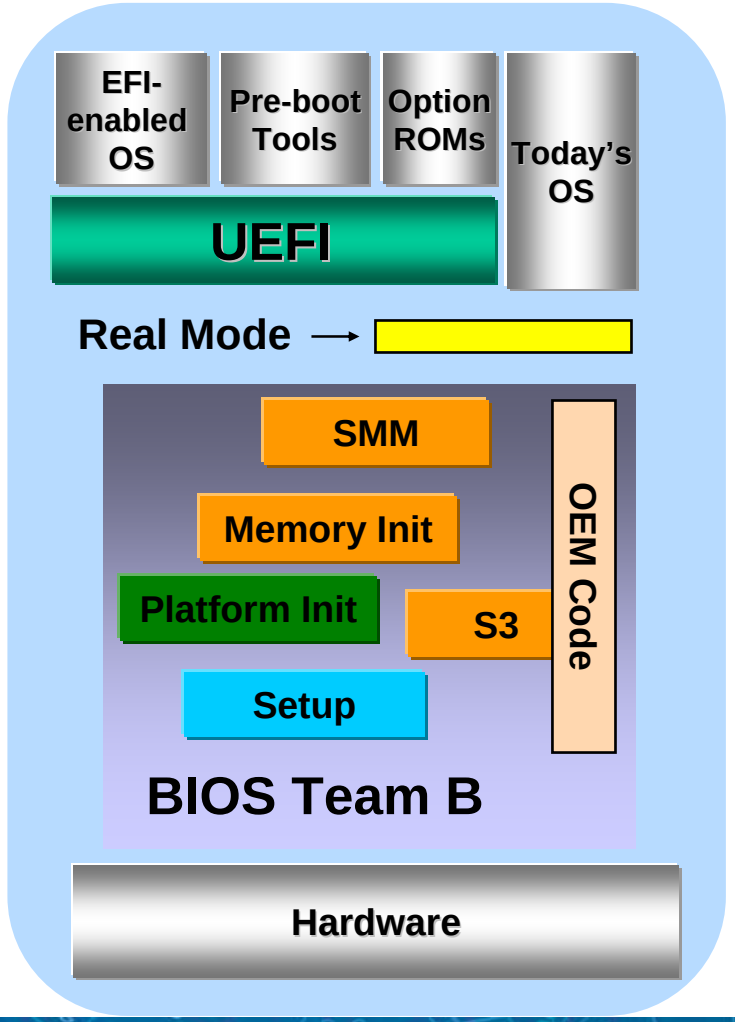
UEFI Separates BIOS and OS

UEFI Firmware Software Stack



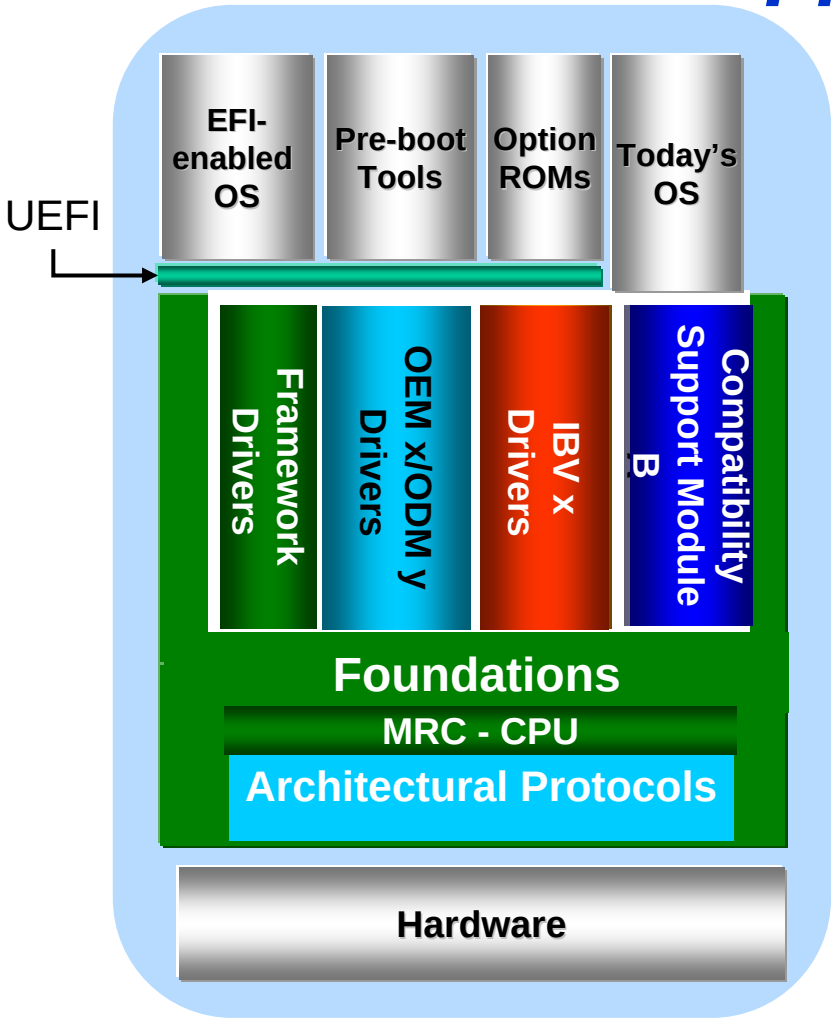
- UEFI on top of Legacy isn't good enough.
- Inherit the same issue
- Legacy is still the heart of the system
- BIOS's between teams not standardized
  - Changes like this mean major overhaul of System BIOS
  - Much rework

UEFI Firmware Software Stack



- UEFI on top of Legacy isn't good enough.
- Inherit the same issue
- Legacy is still the heart of the system
- BIOS's between vendors not standardized
  - Changes like this mean major overhaul of System BIOS
  - Much rework

Framework Software Stack

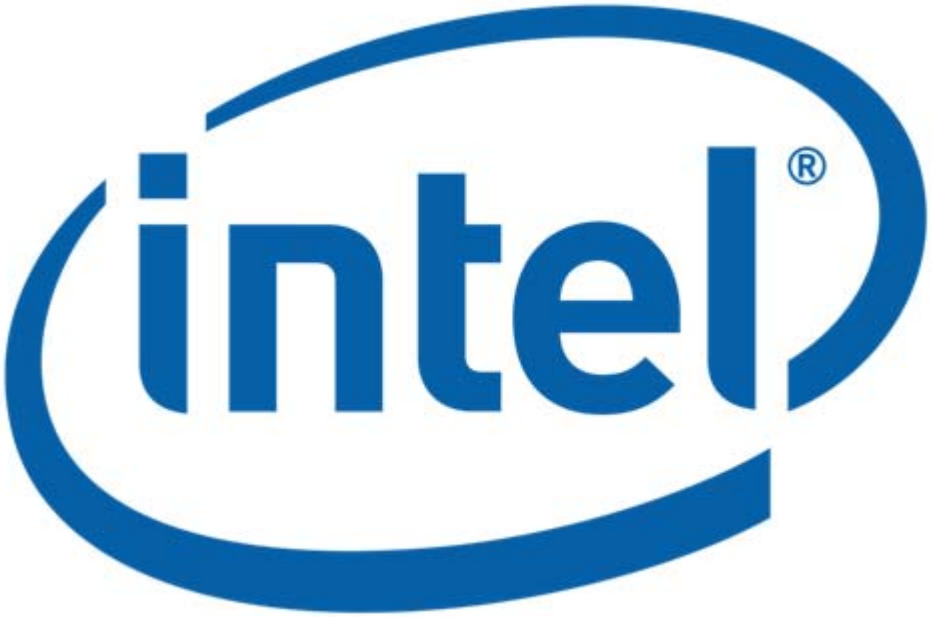


- Foundation lets different teams share code
- Developers can easily move between projects
- Chipset code enabled by Silicon vendor
- Standardization benefits the industry
- IBV provides value add
- Glue code "Big H" is Open Source on Tianocore.org
- Framework Start point for Platform Initialization (PI) Specification on UEFI.org

Summary

- UEFI as an interface is settled –firmly on the Industry's Roadmap
- UEFI breaks through fundamental BIOS barriers
- UEFI Solves Option ROM problem
- The Framework is not just another "core" code base … It's a new purpose-built architecture for firmware
  - UEFI / EFI on top of BIOS is not a good solution

Q & A



Back up

Session Goals

- Describe Background of Legacy BIOS and UEFI
- Introduce fundamental principles of UEFI architecture and Intel's EFI implementation
- Provide a basic common dictionary of UEFI terms and concepts
- These terms and concepts are the basis of understanding UEFI and will be built upon in following sessions.

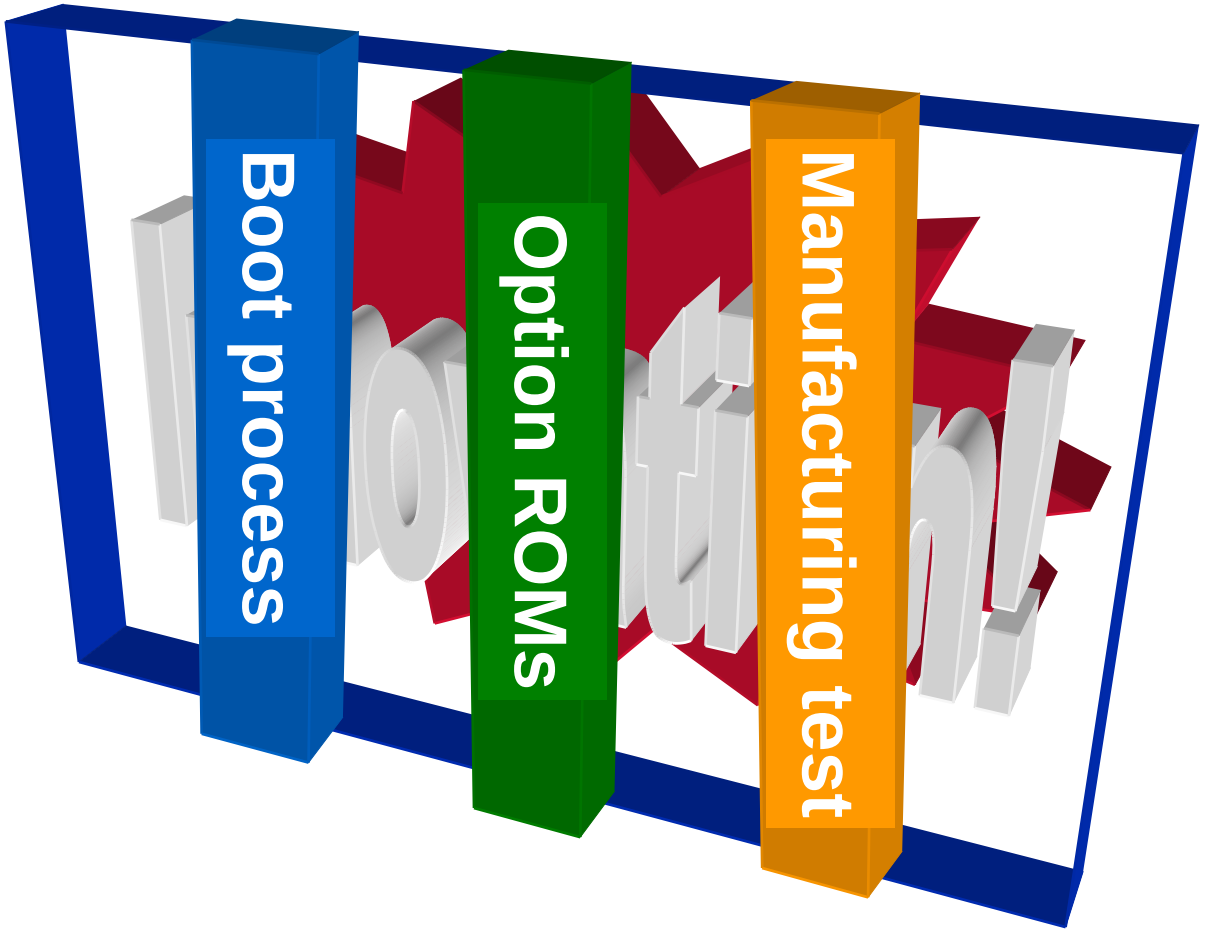
Problem of Legacy BIOS

- It is not platform independent and it highly depends on Intel 8086 Software Interrupt model
  - Real mode
  - 16-bits register access
  - No memory address beyond 1MB
  - Option ROM space limited to below 1MB, limitation on OPROM size
- Modern Intel CPU architecture has come to 32bit, 64bits protection mode, with Hyper-Threading Technology and Multi-core, but for compatibility, it still needs to boot in 16 bit real mode

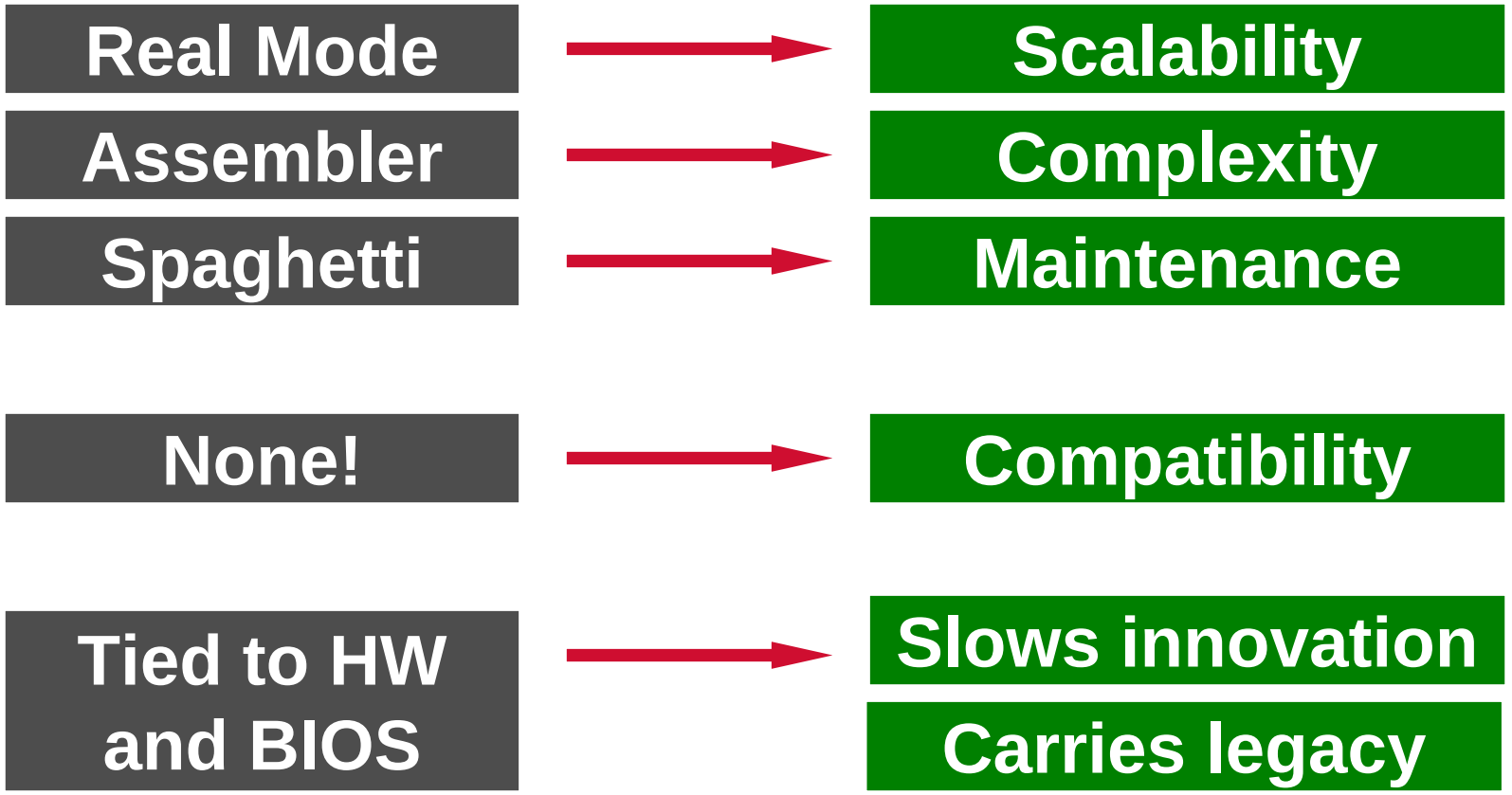
Problem of Legacy BIOS (Cont.)

- Calls into 16 bit legacy firmware is inaccessible with boot to 64 bit system and Itanium® processor family systems.
- Hard to enable new technology quickly for the whole eco-system using Legacy BIOS. (i.e. Intel® Virtualization Technology (VT) and Intel® Trusted Execution Technology (Intel® TXT))
- There is a need for processor architecture independency that Legacy does not account for where a single driver can work across multiple processors (i.e. x86, Itanium®, XScale)
- .

The pre-boot dilemma

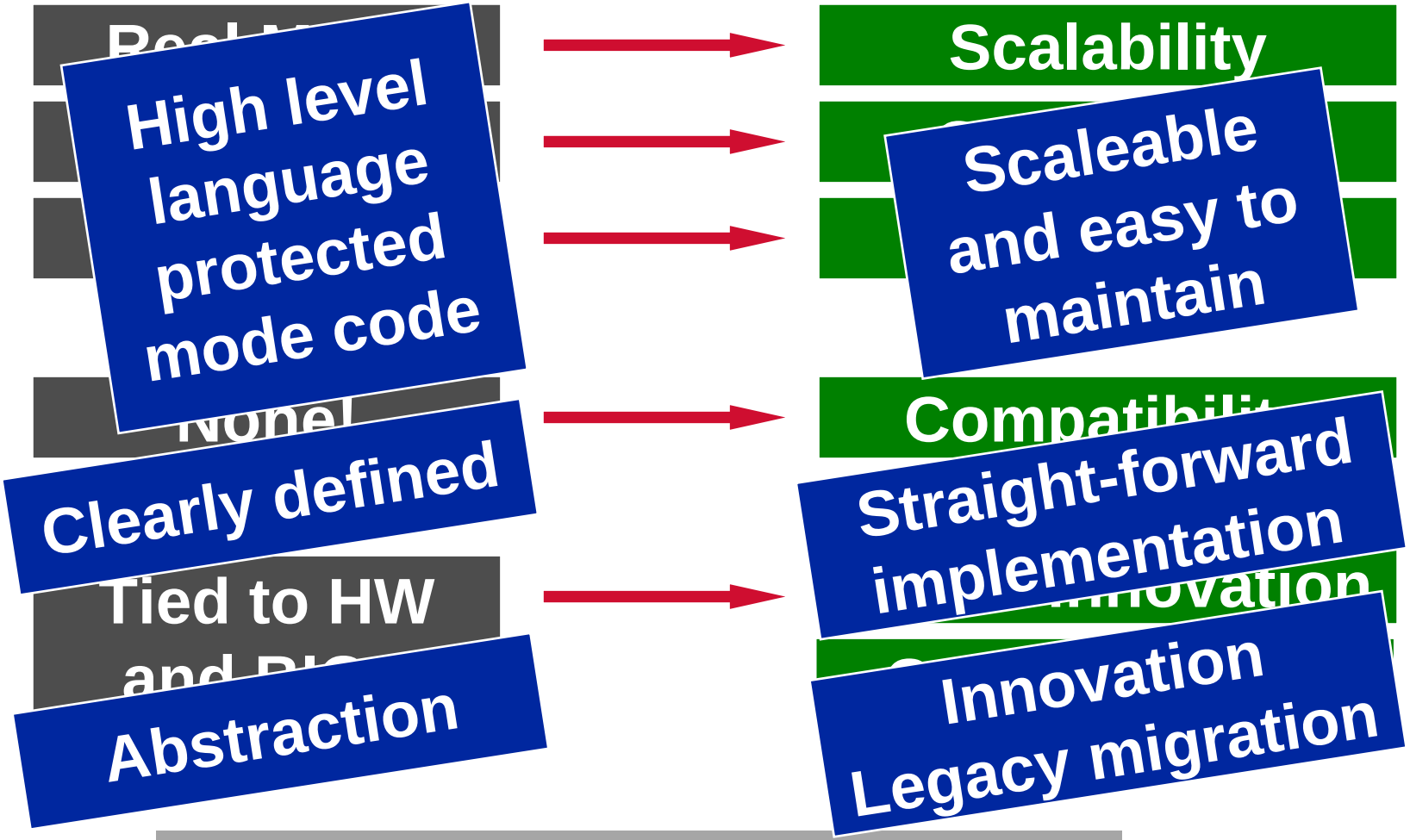


Issues with existing boot



New Architecture Required

UEFI Delivers . . .



The right solution

Breaking away

Golden opportunity for change

UEFI 2.1

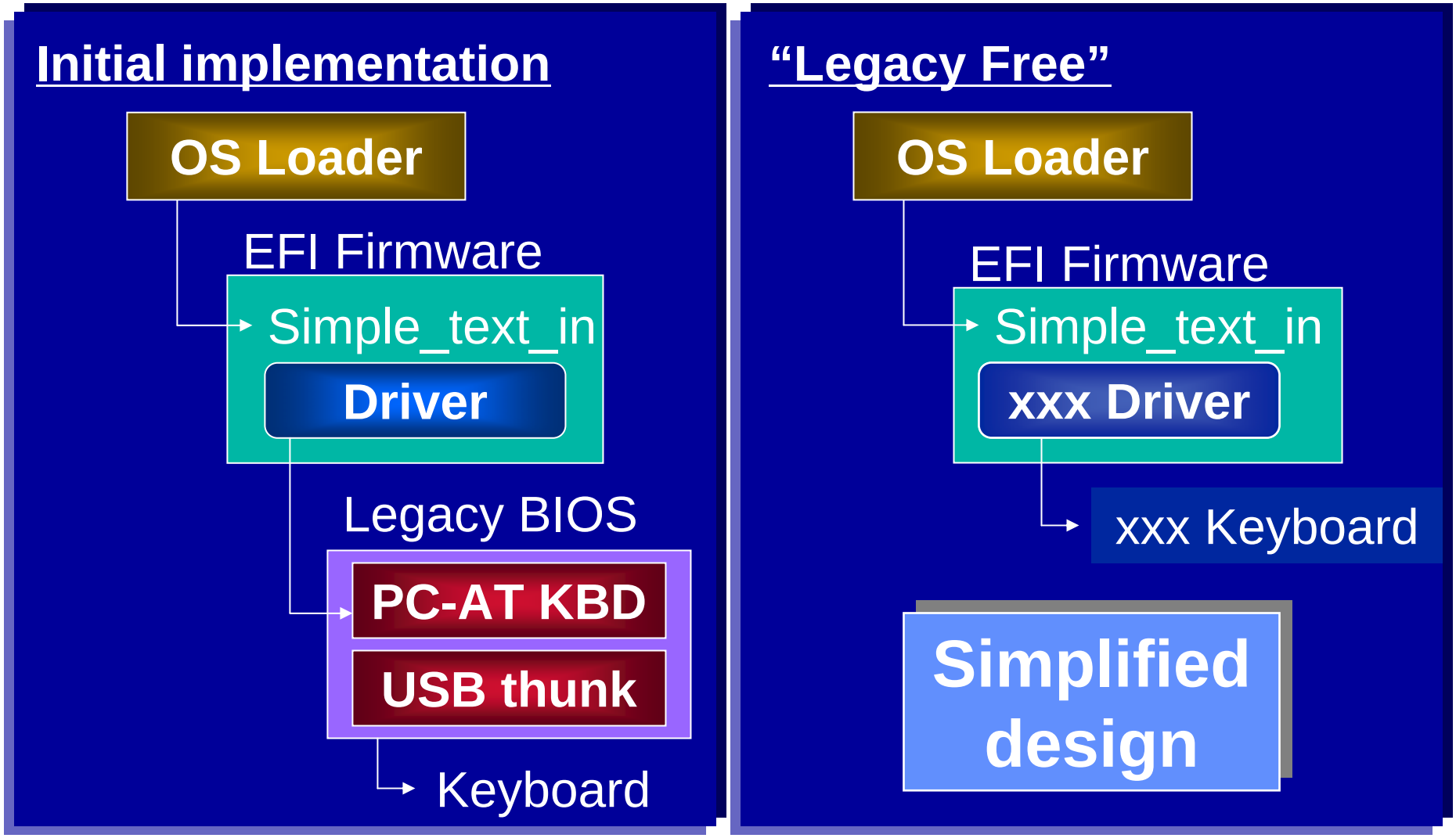
- Roughly one year of Specification work
  - Builds on UEFI 2.0
- Board approved formal Adoption Jan 23rd, 07
  - Available for download from www.uefi.org
- New content area highlights:
  - Human Interface Infrastructure
  - Hardware Error Record Support
  - Authenticated Variable Support
  - Simple Text Input Extensions
  - Absolute Pointer Support

UEFI 2.2

- Follow-on material from existing 2.1 content
  - Backlog that needed more gestation time
  - Complete August 2008
- 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.
- Network – IPv6, PXE+, IPSec
- Various other subject areas possible
  - More boot devices, more authentication support, etc.
- Establishing a formal relationship with DMTF (Distributed Management Task Force)

UEFI is the only place where these future technologies are defined

Protocols Example



UEFI Driver Design

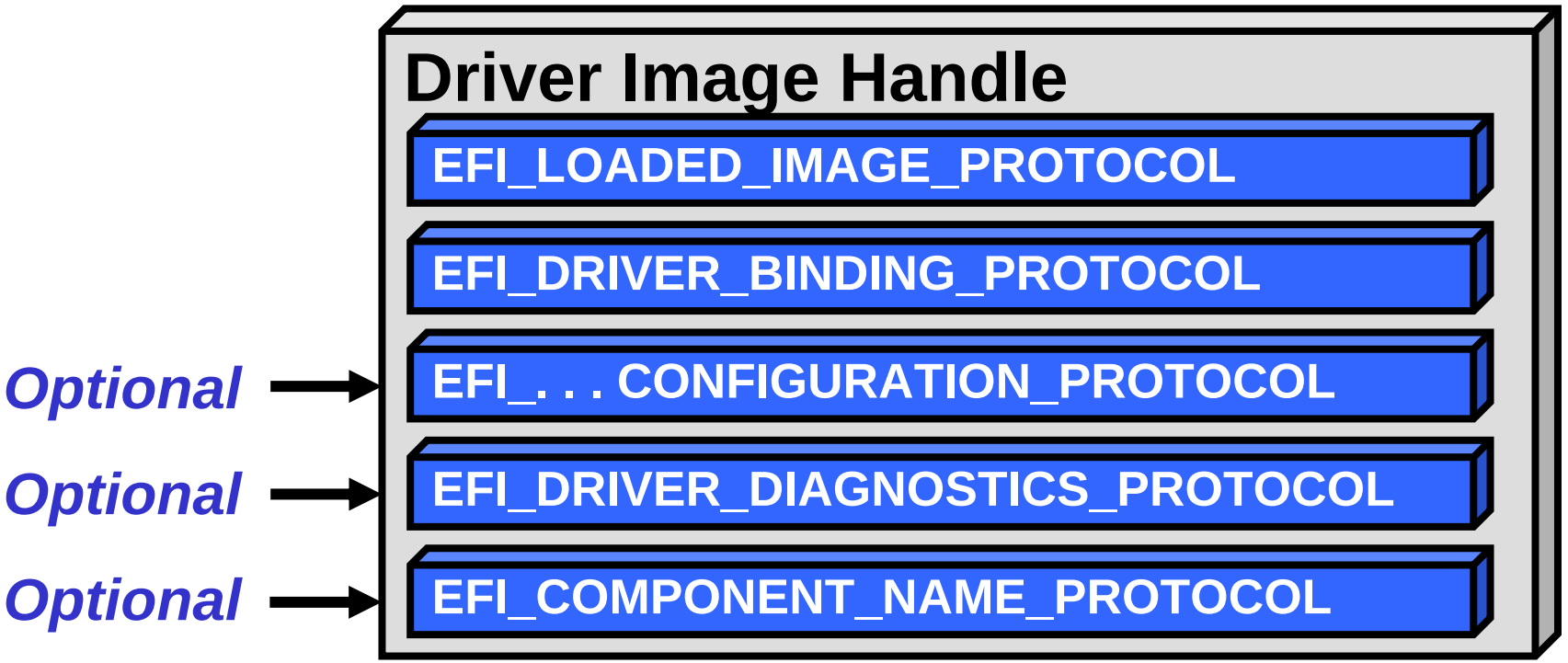
- Modular chunks of code run in pre-boot
  - Manage devices or services
  - …they are NOT OS-present drivers!
- Drivers export protocol interfaces
  - Protocol = instance data + access methods
  - Like C++ classes but more code space efficient
  - Identified by GUID to avoid collisions
  - Version numbers and signatures provide means to manage driver management policy
- Drivers may consume protocol interfaces
  - Self-describing dependencies
  - E.g. Memory initialization may depend on SMBUS service

UEFI Driver Model

- Used for devices on industry standard buses
  - "boot devices"
- Structured model of device/bus hierarchy
  - Device Drivers and Bus Drivers
  - Device Drivers are topology agnostic
- Benefits
  - Simpler Device Drivers
    - Moves complexity into Bus Drivers and core services
    - Smaller driver footprint
  - Deterministic driver selection by the platform
    - Which driver controls which device
  - Describes complex bus hierarchies
    - Embedded, Desktop, Workstation, Server
  - Extensible to future bus types

Use of multilayer modularity means more scenarios "just work"

Driver Initialization



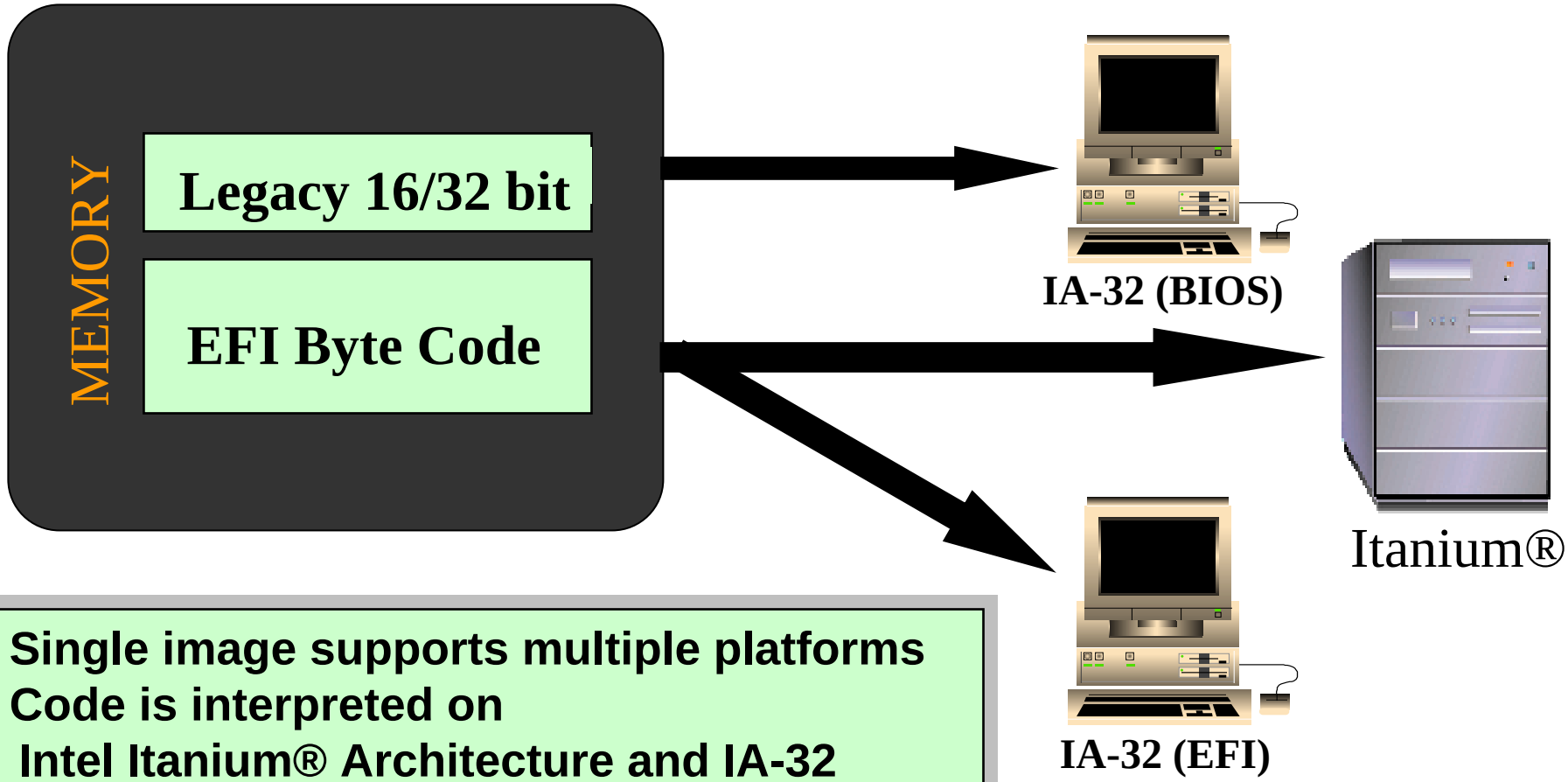
Optional Protocols Add Value
Used by Platform Management Utility

See § 10 UEFI 2.1 Spec.

Driver Binding Protocol

- *DriverBinding.Supported()*
  - Checks to see if a driver supports a controller
  - Check should not change hardware state of controller
  - Minimize execution time, move complex I/O to Start()
  - May be called for controller that is already managed
  - Child is optionally specified
- *DriverBinding.Start()*
  - *Starts a driver on a controller*
  - *Can create ALL child handles or ONE child handle*
  - *A driver is not required to support starting ONE child handle. It may always create ALL child handles*
- *DriverBinding.Stop()*
  - *Stops a driver from managing a controller*
  - *Destroys all specified child handles*
  - *If no children specified, controller is stopped*
  - *Stopping a bus controller requires 2 calls*

EFI Byte Code Overview



- Single image supports multiple platforms
- Code is interpreted on
  Intel Itanium® Architecture and IA-32 systems
- IHVs develop only once