UEFI Overview

Ruth Li

Manager UEFI Development
Intel

Session Goals

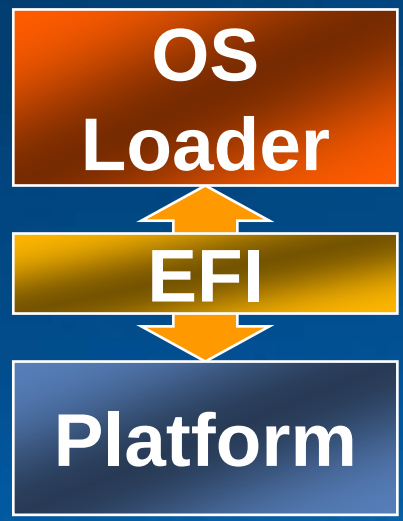
- Describe Background of EFI and UEFI
- Introduce fundamental principles of UEFI architecture and Intel's EFI implementation
- Provide a basic common dictionary of UEFI terms and concepts
- These terms and concepts are the basis of understanding UEFI and will be built upon in following sessions.

Agenda

- UEFI Overview
- UEFI Technical Overview
  - UEFI Terminology
  - What is the UEFI System Table?
  - What is a Device Path
  - What are UEFI Boot Services?
  - EFI 1.1 and UEFI 2.0 differences

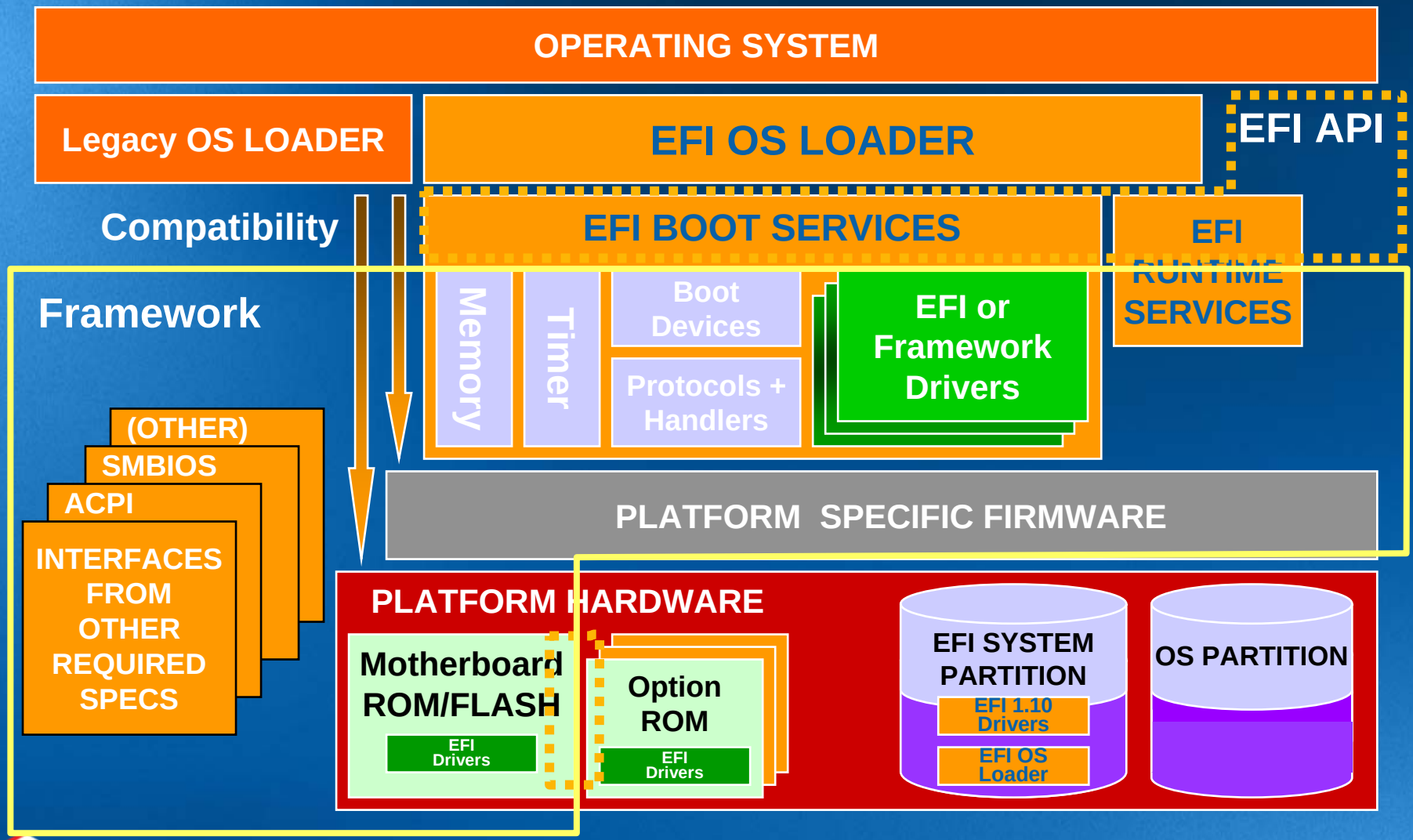
EFI Overview

- Extensible Firmware Interface
- EFI is an interface specification
- Abstracts the platform from OS
  - Decouples development
- Includes a modular driver model and CPU-independent option ROMs
  - Offers promise of improved RAS
- Compatible by design
  - Evolution, not revolution
- Modular and extensible
  - OS-Neutral value add
- Complements existing interfaces



Flexible to meet existing and future needs

EFI Concept


OPERATING SYSTEM
Legacy OS LOADER
EFI OS LOADER
EFI API
Compatibility
EFI BOOT SERVICES
EFI RUNTIME SERVICES
Framework
Memory
Timer
Boot Devices
Protocols + Handlers
EFI or Framework Drivers
(OTHER)
SMBIOS
ACPI
INTERFACES FROM OTHER REQUIRED SPECS
PLATFORM SPECIFIC FIRMWARE
PLATFORM HARDWARE
Motherboard ROM/FLASH
EFI Drivers
Option ROM
EFI Drivers
EFI SYSTEM PARTITION
EFI 1.10 Drivers
EFI OS Loader
OS PARTITION

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 2.1 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

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

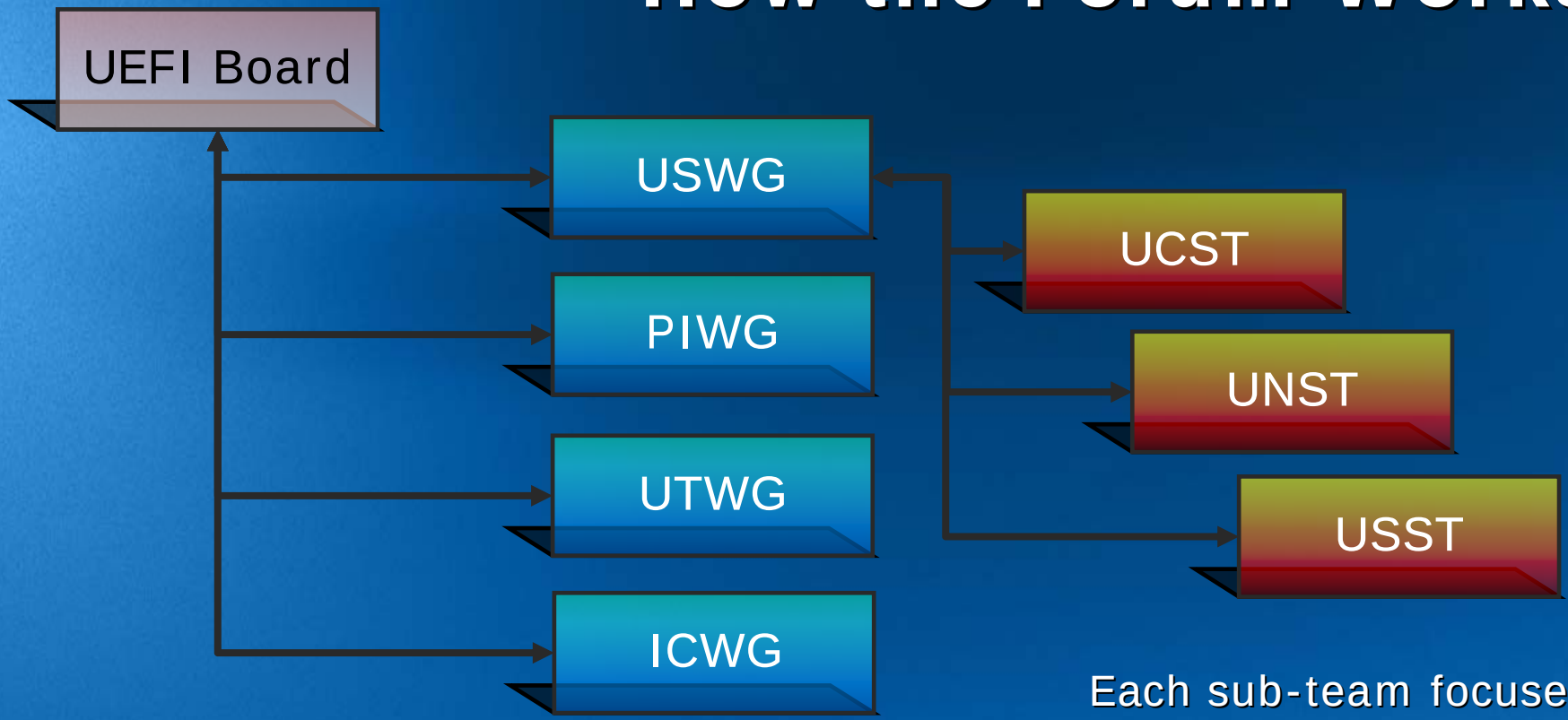
Adopters:

- Any entity wanting to implement the specification

How the UEFI Forum Works

- Board sets strategic direction
  - Charters work groups to implement direction
- Work groups drawn from membership
  - Staffed by Promoters and Contributors
- Board approves work groups proposals
- Work Groups:
  - USWG – UEFI Specification Working Group
    - responsible for UEFI Spec production
  - UTWG – UEFI Testing Working Group
    - Responsible for publishing SCT
  - PIWG – Platform Initialization Working Group
    - Responsible for the spec that will eventually replace the framework level specs
  - ICWG: Industry Communications

How the Forum Works



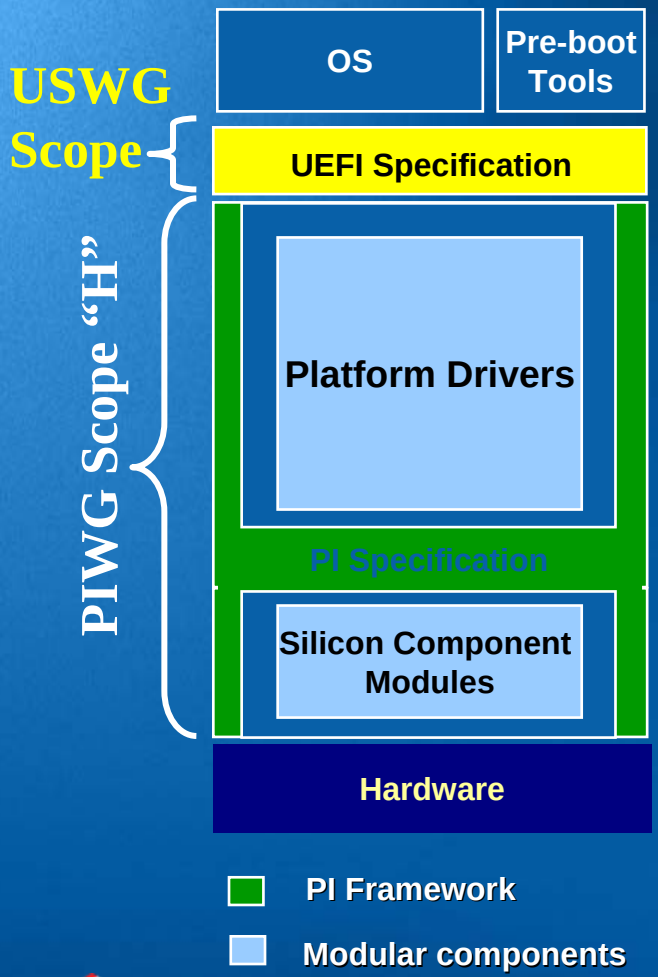
Publications/Decisions ratified by the board

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

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

UEFI Many Groups working together for Standardizing Firmware

USWG/PIWG Relationship



- UEFI Spec is about interfaces between OS, add-in driver and system firmware
  - Operating systems and other high-level software should only interact with interfaces and services defined by the UEFI Specification
- PI Specs relate to making UEFI implementations
  - Promote interoperability between firmware components providers
  - All interfaces and services produced and consumed by firmware only

UEFI and PI are Independent Interfaces

Introducing 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 Work in Progress

- Follow-on material from existing 2.1 content
  - Backlog that needed more gestation time
  - Aiming for latter portion of 2007 completion
- 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

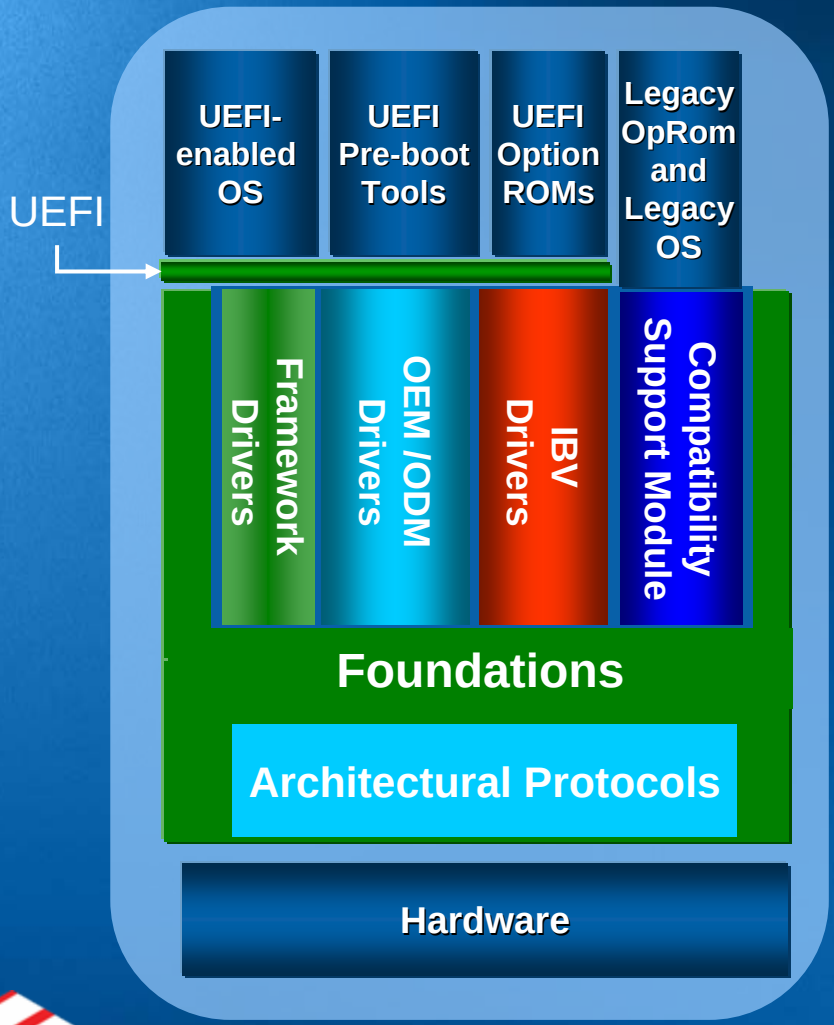
Introducing PI 1.0

- Roughly one year of Specification work
  - Builds on Intel PEI and DXE Framework specifications
- Board approved formal Adoption Oct 31st, 06
  - Available for download from http://www.uefi.org
- Starting work from Intel contributed specifications
  - Framework DXE and PEI Core Interface Specs
  - Firmware Storage, HOB, SMBus
- New content area highlights:
  - Pre-EFI Initialization Core Interface
  - Driver Execution Environment Core Interface
  - Shared Architectural Elements

PI 1.1 Work In Progress

- Based on existing Intel Framework Specifications
  - Highest priority subset chosen
  - Target completion in 2007
- PCI
  - Standardized resource allocation
- ACPI
  - Manipulation of ACPI data.
- SMM
  - Standardized SMM driver model
- MP
  - Standardized Multi Processor infrastructure.
- S3
  - Infrastructure to support power state transitions.

Framework Concept



- Open Source “H”
- Starting point for firmware
- Add some NDA Stuff
- Base Drivers for platform
- SKU specific drivers
- IBV Value Add
- Full UEFI compliance
- Legacy support
- Current Firmware

Agenda

- UEFI Overview
- UEFI Technical Overview
  - UEFI Terminology
  - What is the UEFI System Table?
  - What is a Device Path
  - What are UEFI Boot Services?
  - EFI 1.1 and UEFI 2.0 differences

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

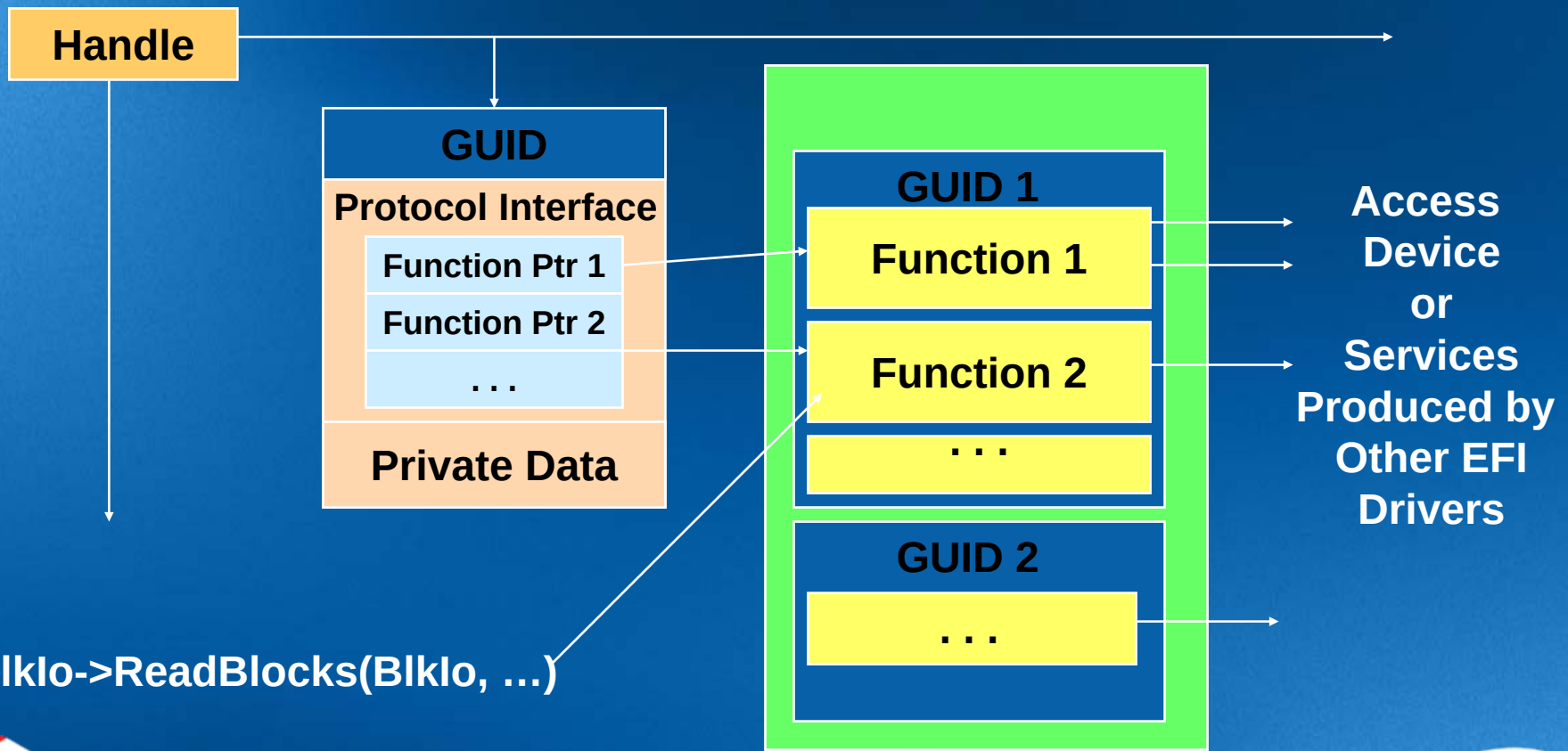
What are GUIDs

- Guaranteed Unique Identifiers
  - 128-bit quantity **
- Used to identify protocols
  - 1:1 with interfaces
- Regulate extension mechanism
  - Documented in the spec
  - Added through drivers

** as defined in the Wired for Manageability 2.0 spec
http://www.intel.com/design/archives/wfm/downloads/base20.htm

Protocols (API)

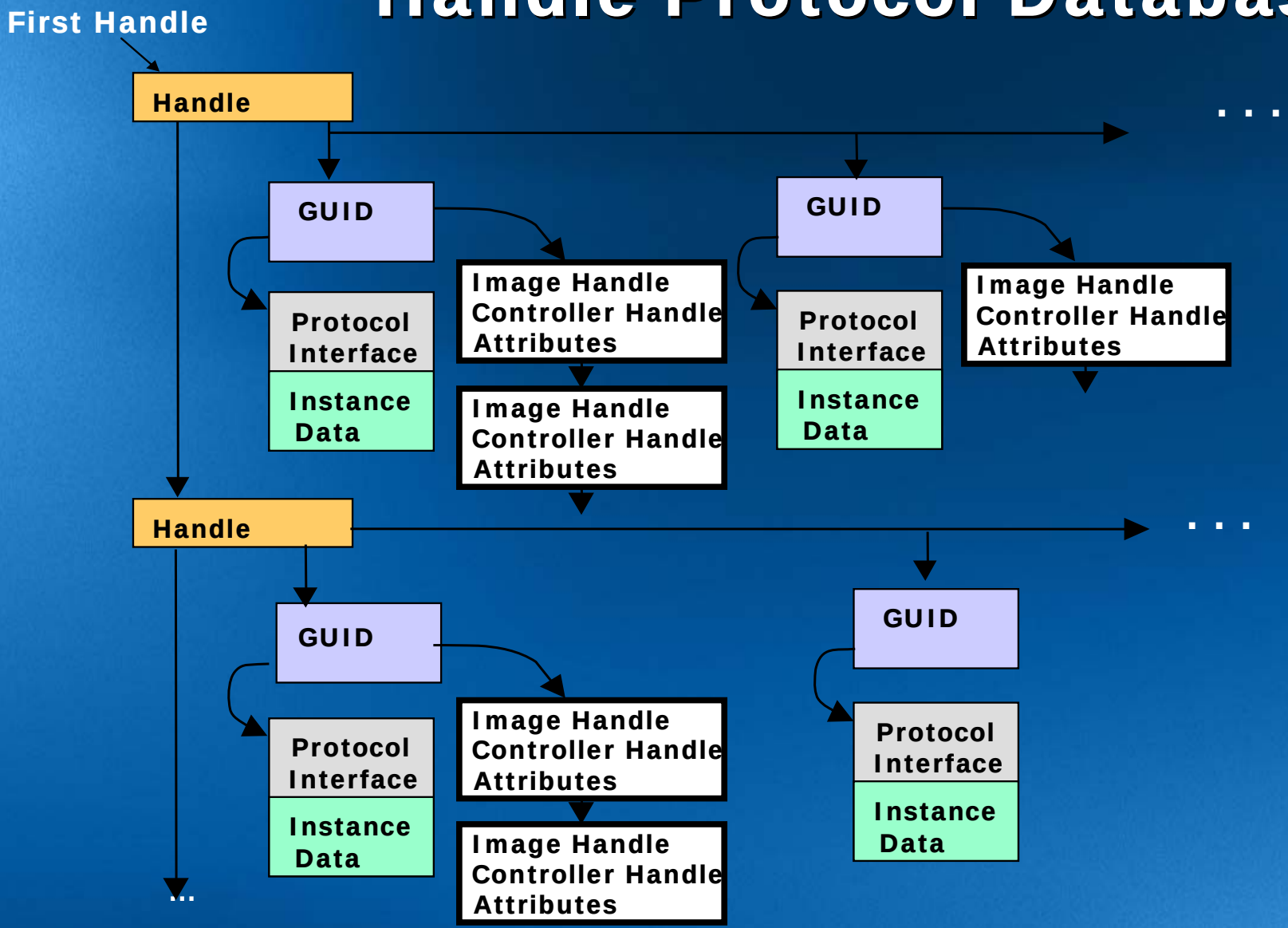
- GUID, Interface Structure, Services
  - DEVICE_PATH, DEVICE_IO, BLOCK_IO, DISK_IO, FILE_SYSTEM, SIMPLE_INPUT, SIMPLE_TEXT_OUTPUT, SERIAL_IO, PXE_BC, SIMPLE_NETWORK, LOAD_FILE, UNICODE_COLLATION



Handles

- All protocols have a handle which is associated with the protocol
- Every device and executable image in EFI has a handle protocol in the handle database

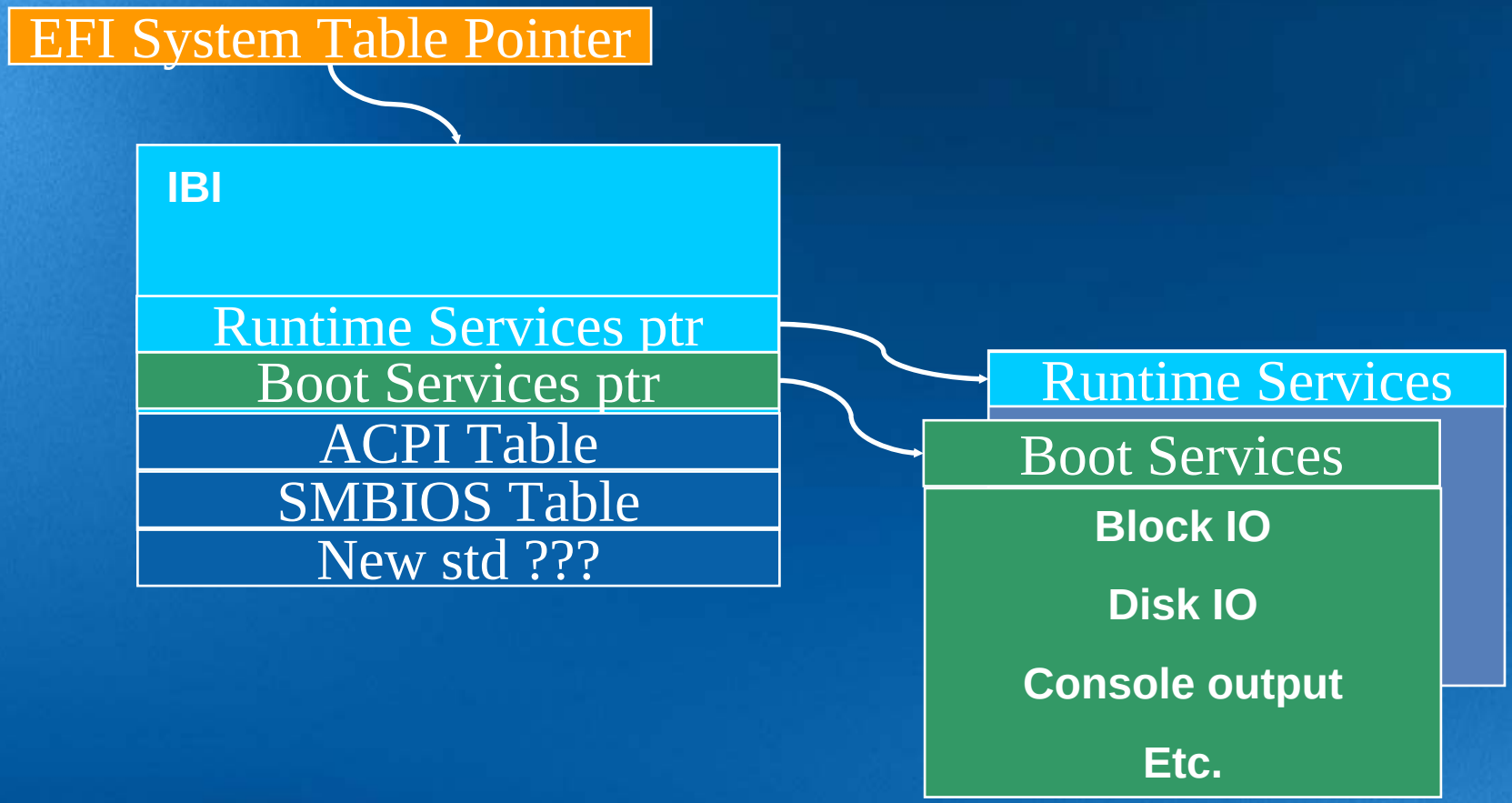
Handle Protocol Database



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



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

What are UEFI 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

UEFI 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

UEFI Driver Design

- Modular chunks of code run in preboot
  - 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

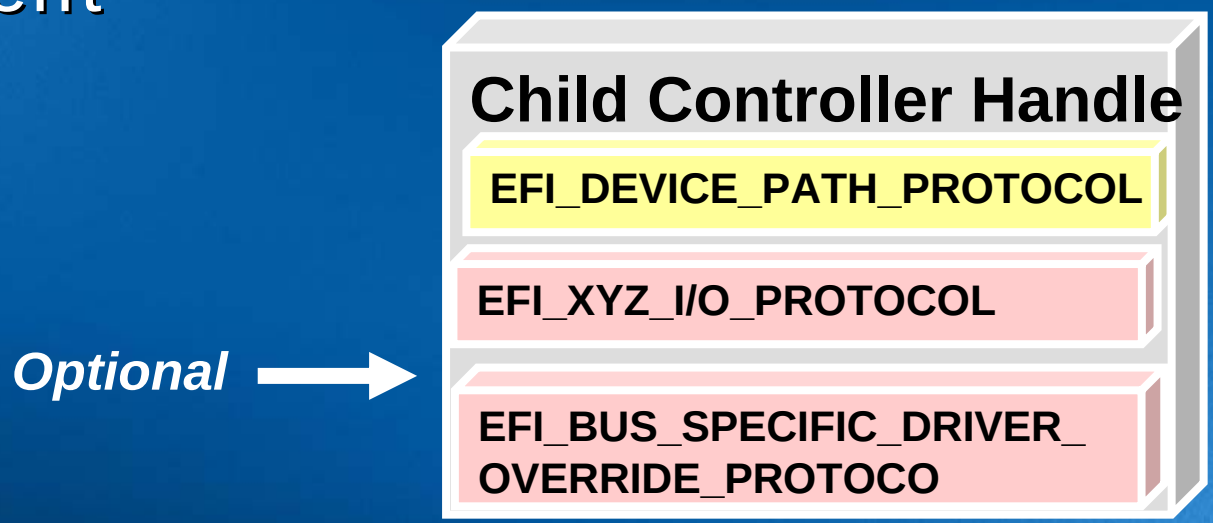
UEFI Driver Model

- 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"

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

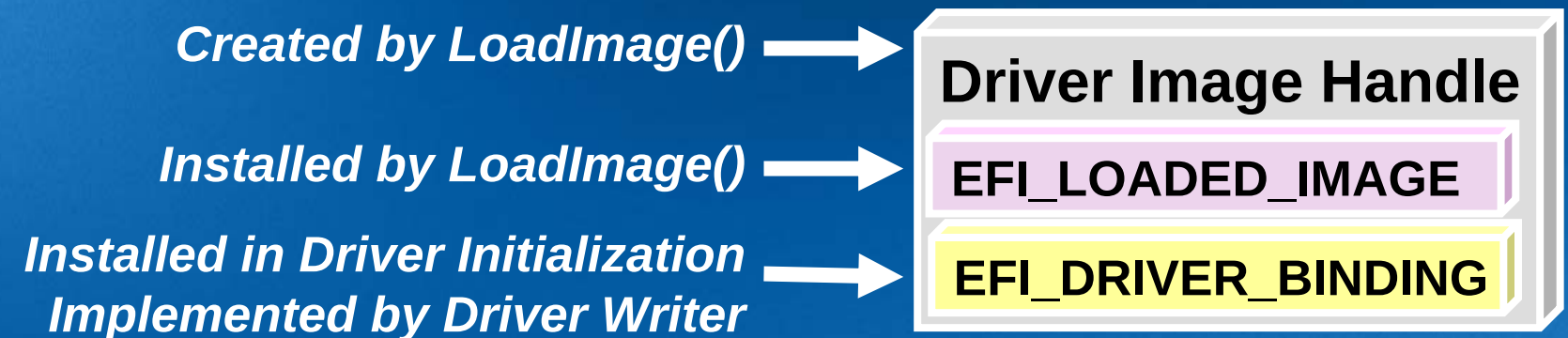


Device Driver

- 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

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
  - Aiming for mid year 2006 completion
- User interface presentation
  - Aiming to 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.

What's supported?

- The community is in the progress of switching from EFI 1.1 to UEFI 2.x.
  - EFI 1.1 protocols
  - UEFI 2.0 protocols starting to replace EFI 1.1 protocols
  - New UEFI 2.1 protocols have been added


intel®
Leap ahead™