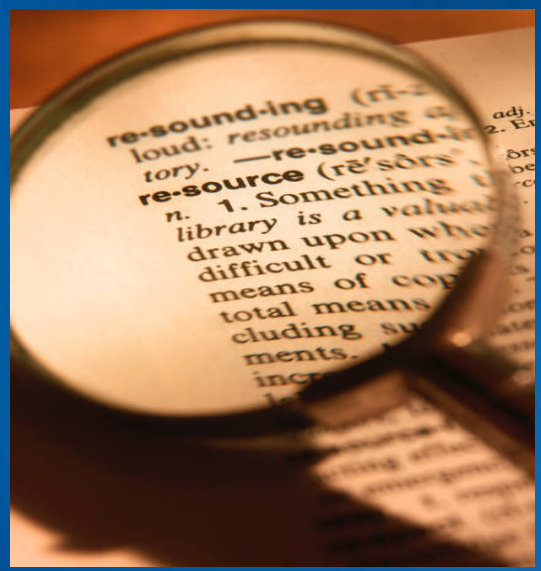
How to Support Customers Using UEFI Products

Name
Title
Department

Support Concept

- Communication
  - Listening
  - Respond
- Resources
  - Training
  - Specifications

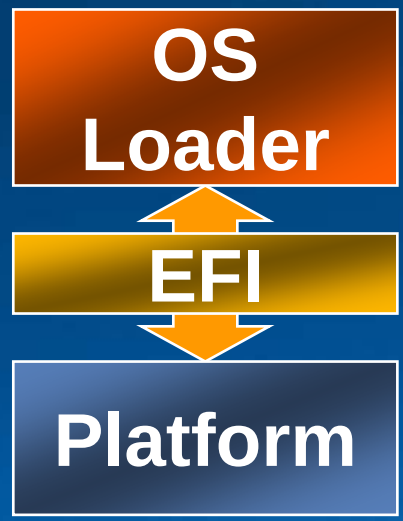


Agenda

- **UEFI Basics**
- UEFI Drivers
- EFI Shell Basics
- Where to find resources

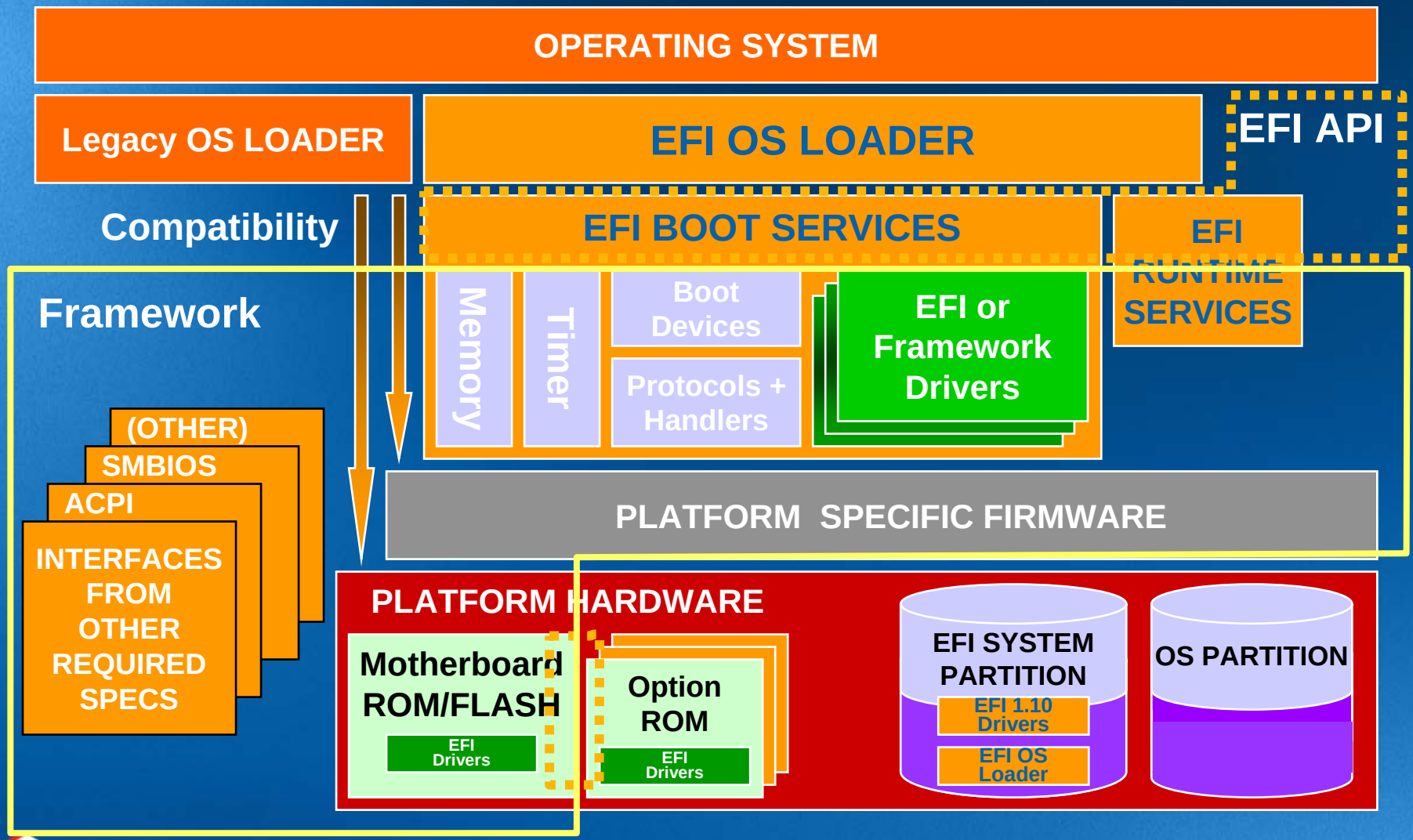
EFI Overview

- 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

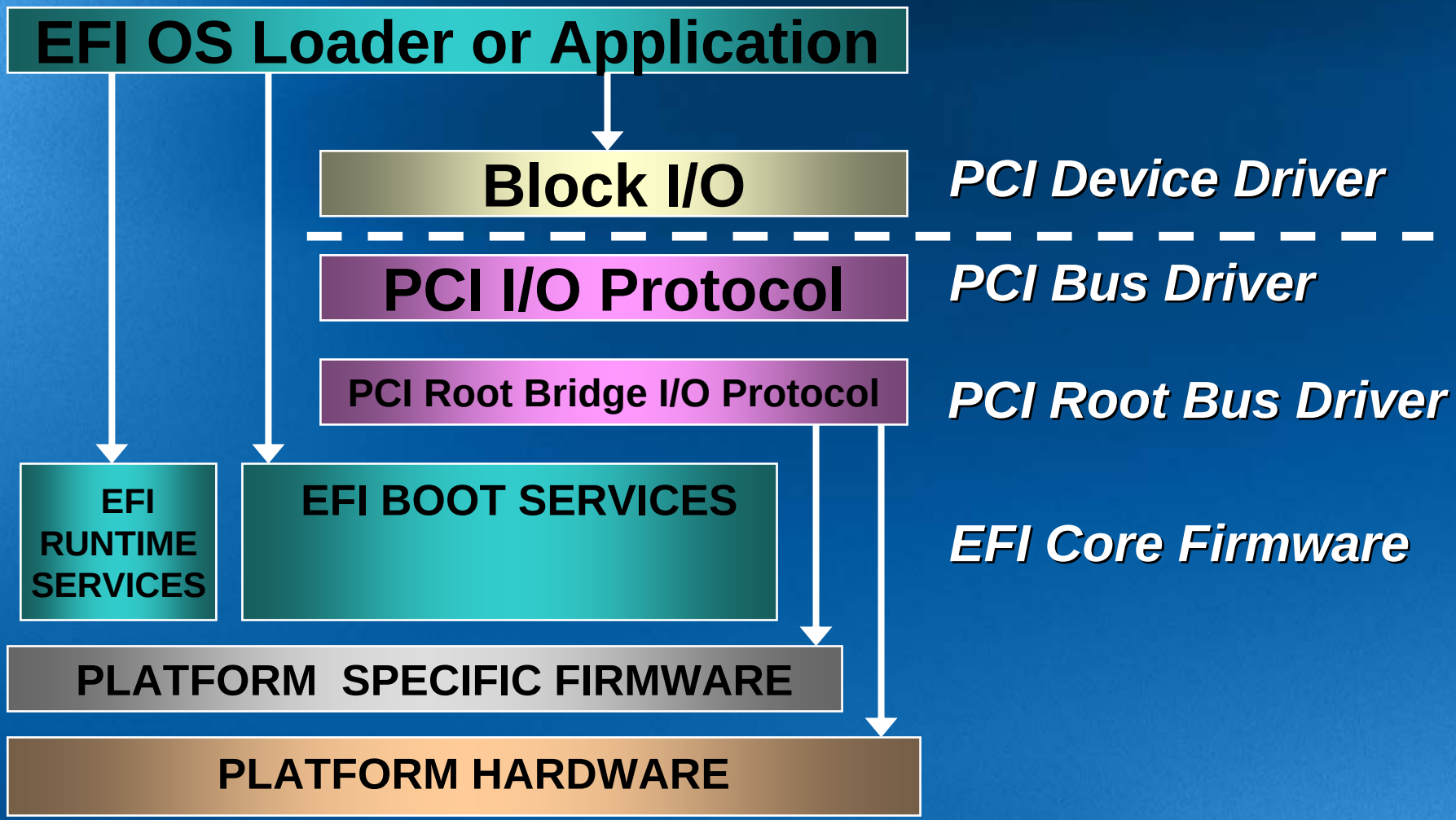


Flexible to meet existing and future needs

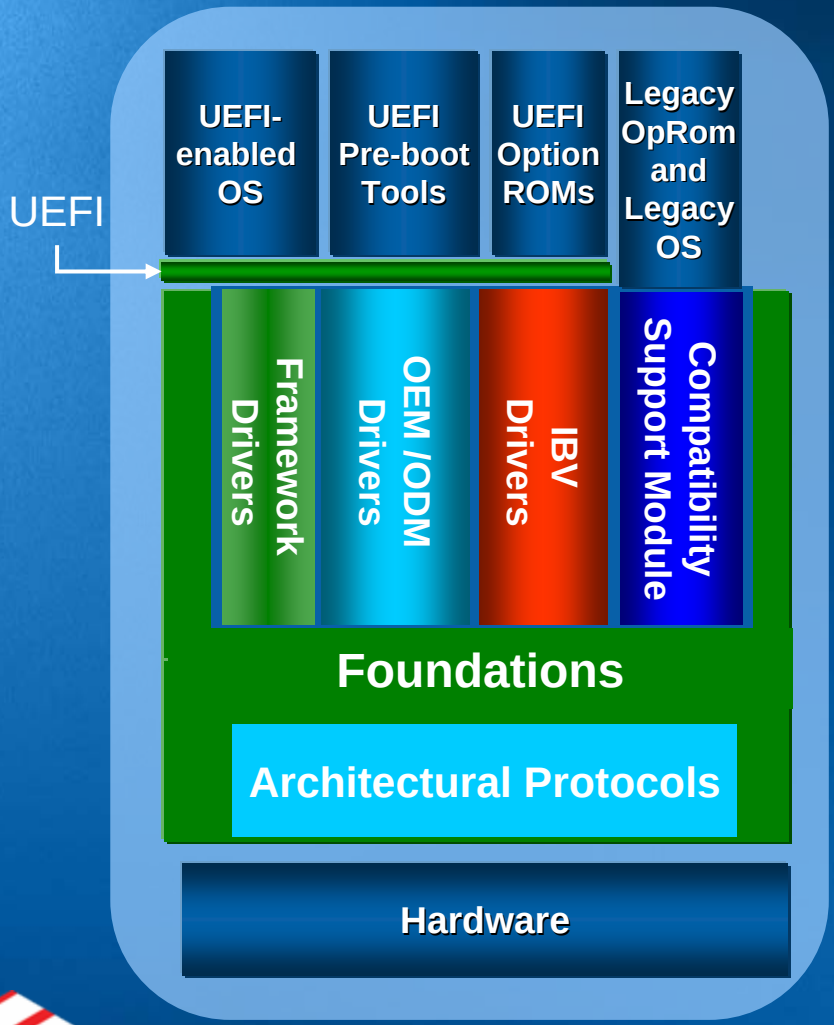
EFI Concept



Example PCI Driver Stack



Framework Concept

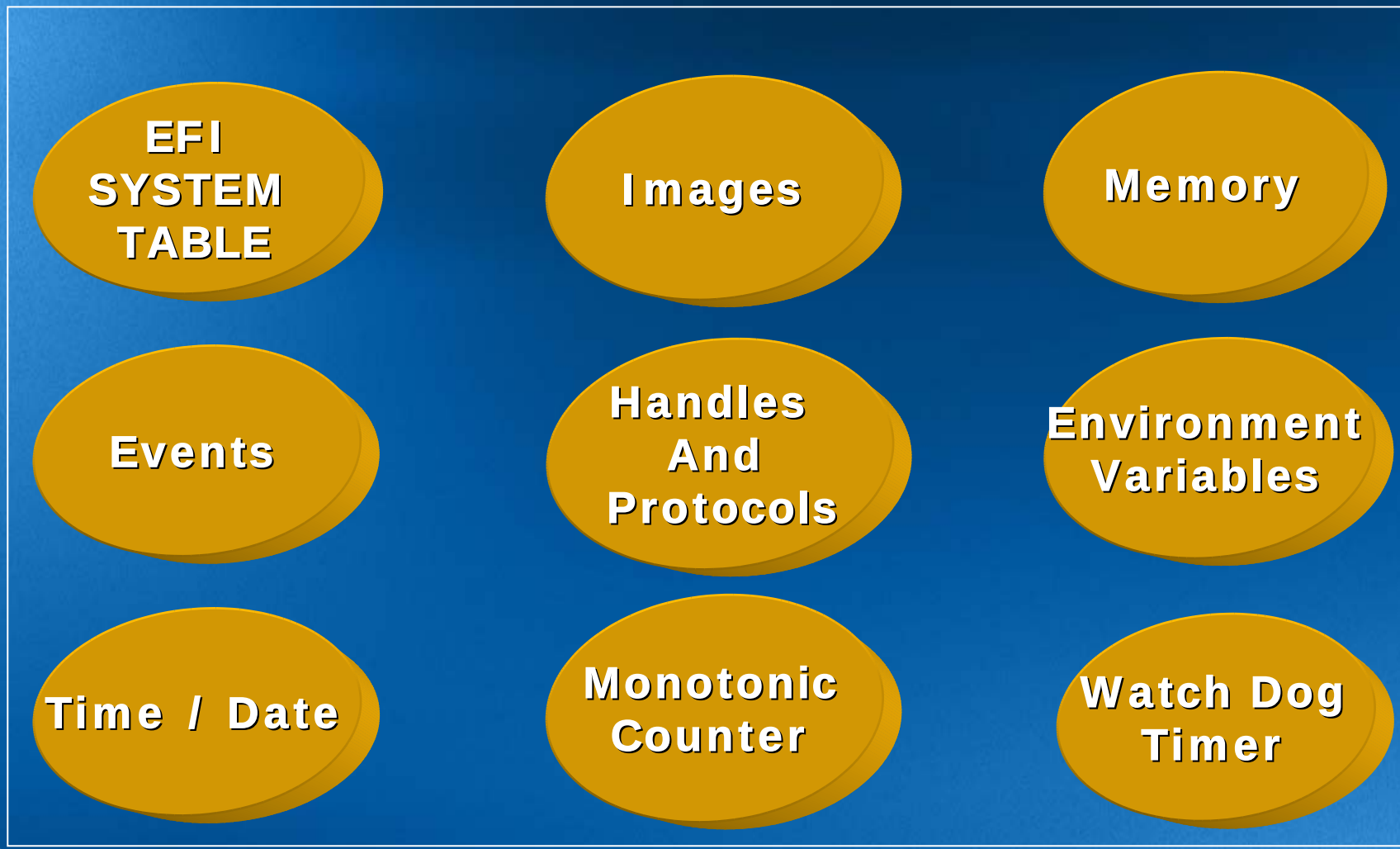


- Open Source “H”
- Starting point for firmware
- Add some NDA Stuff
- Base Drivers for platform
- SKU specific drivers
- IBV Value Add
- Full UEFI compliance
- Legacy support
- Current Firmware

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

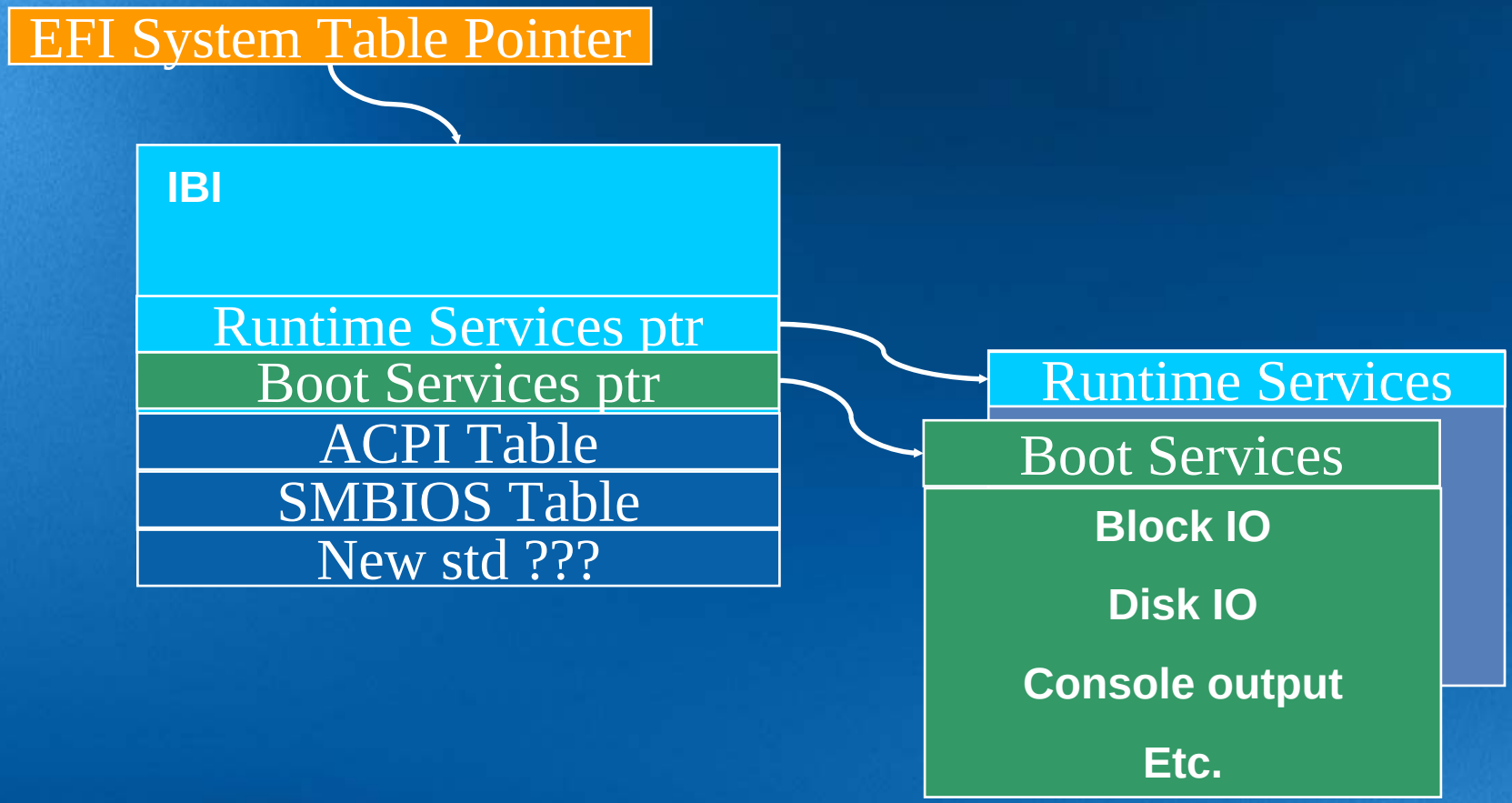


What is the UEFI System Table

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

See § 4 UEFI 2.1 Spec.

UEFI System Table



What are GUIDs

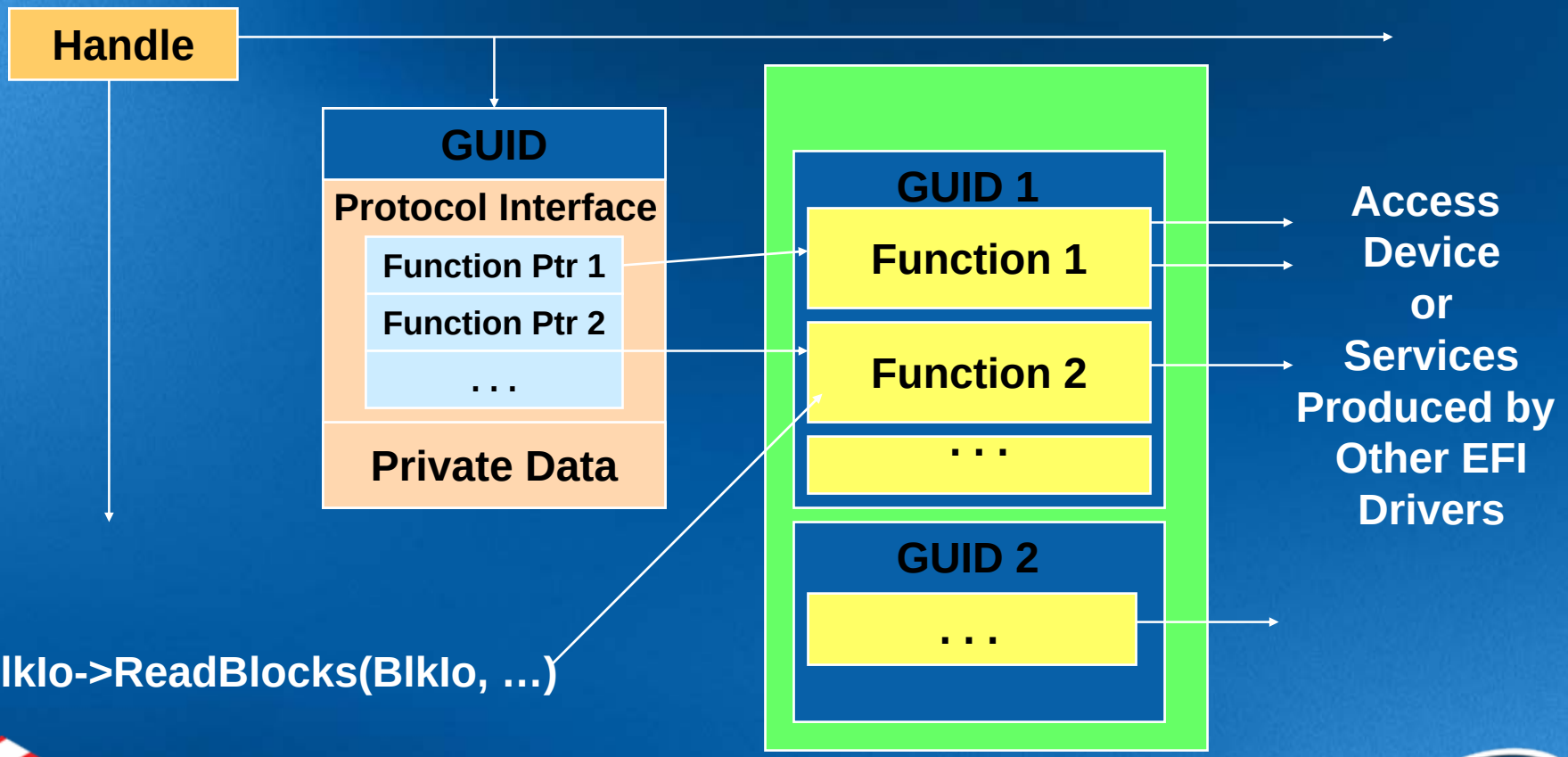
- Guaranteed Unique Identifiers
  - 128-bit quantity **
- Used to identify protocols
  - 1:1 with interfaces
- Regulate extension mechanism
  - Documented in the spec
  - Added through drivers

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

See § 5 UEFI 2.1 Spec.

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



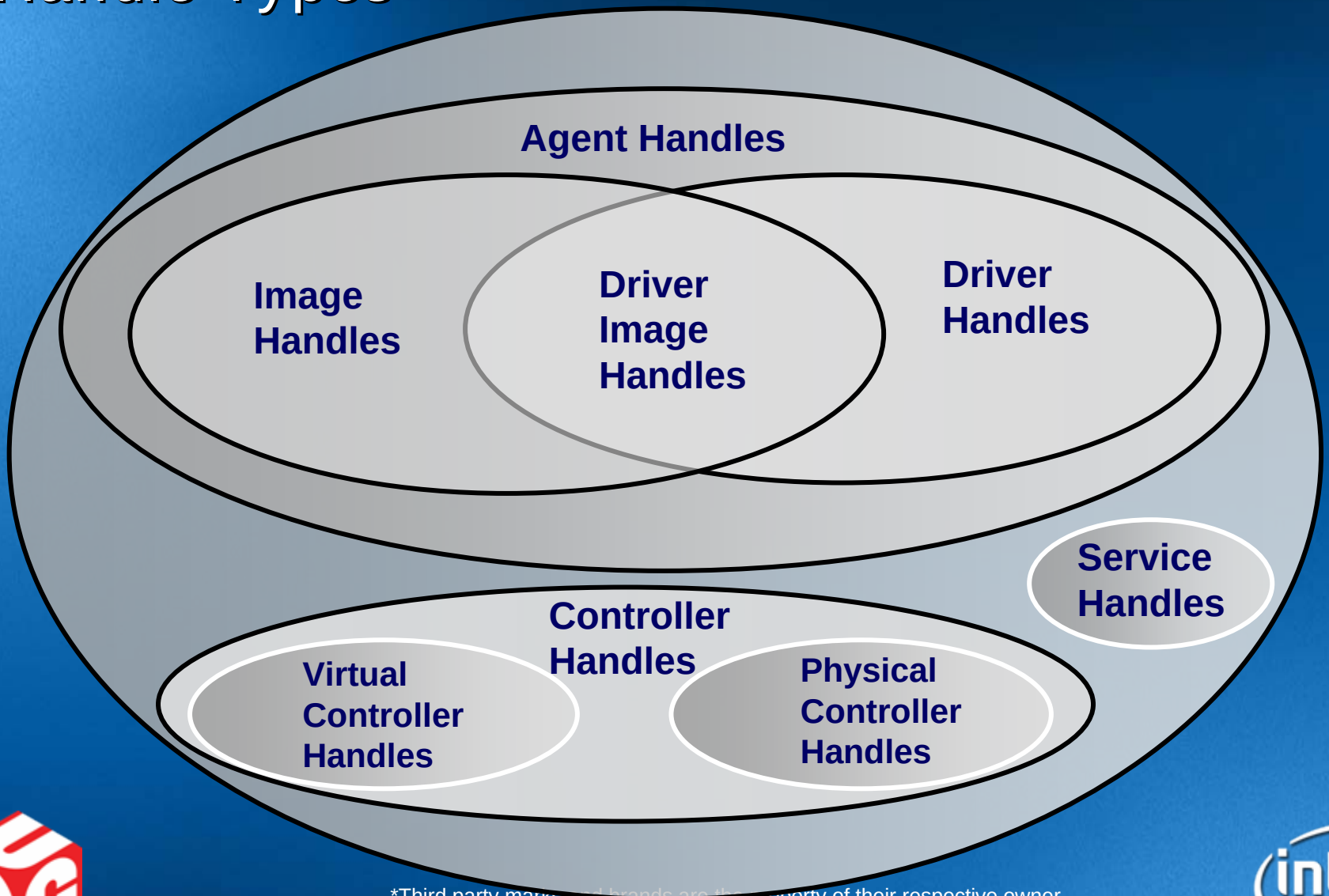
See § 2.4 UEFI 2.1 Spec.

Handles

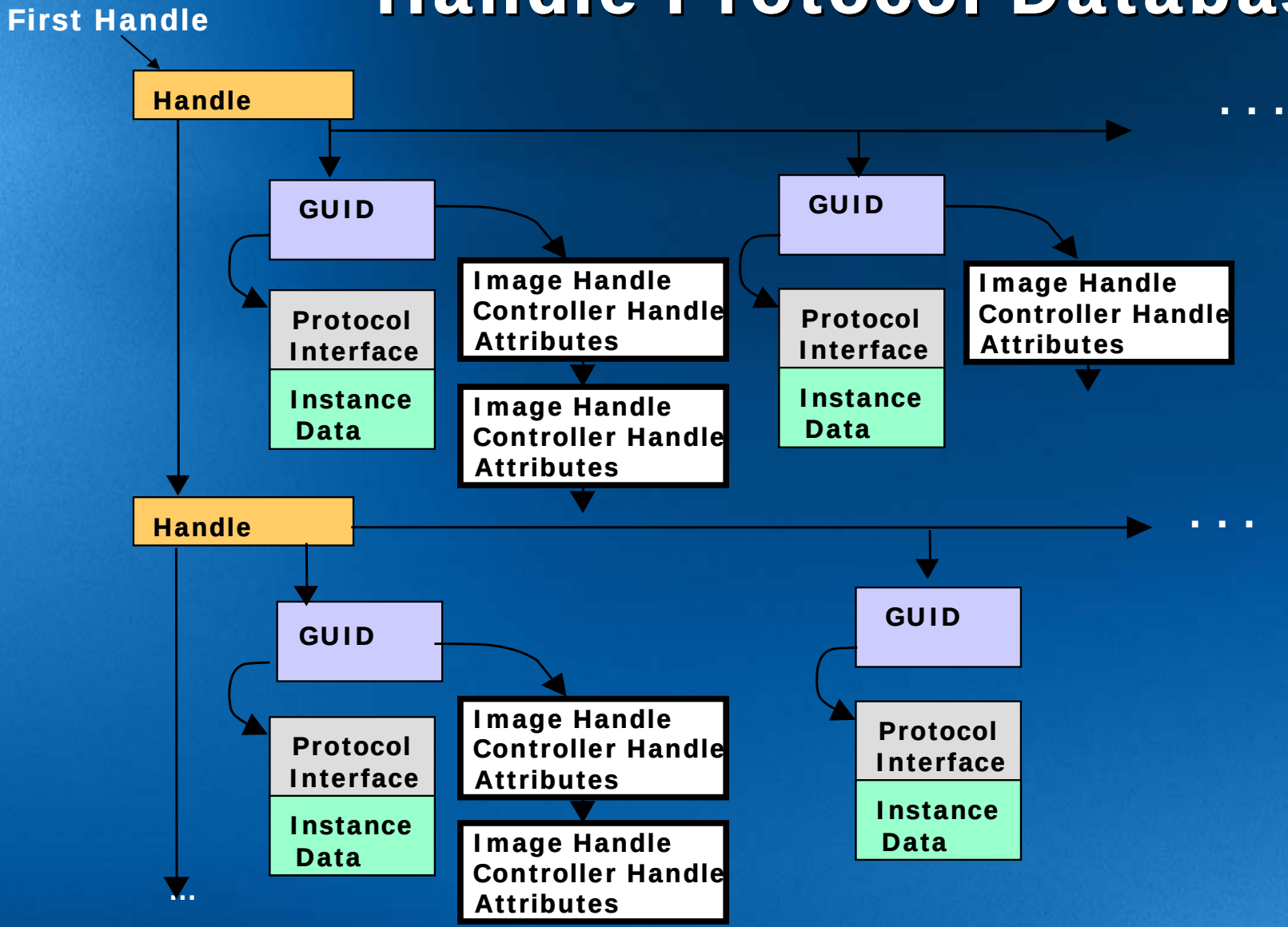
- All protocols have a handle which is associated with the protocol
- Every device and executable image in EFI has a handle protocol in the handle database

Handles

Handle Types



Handle Protocol Database



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

Agenda

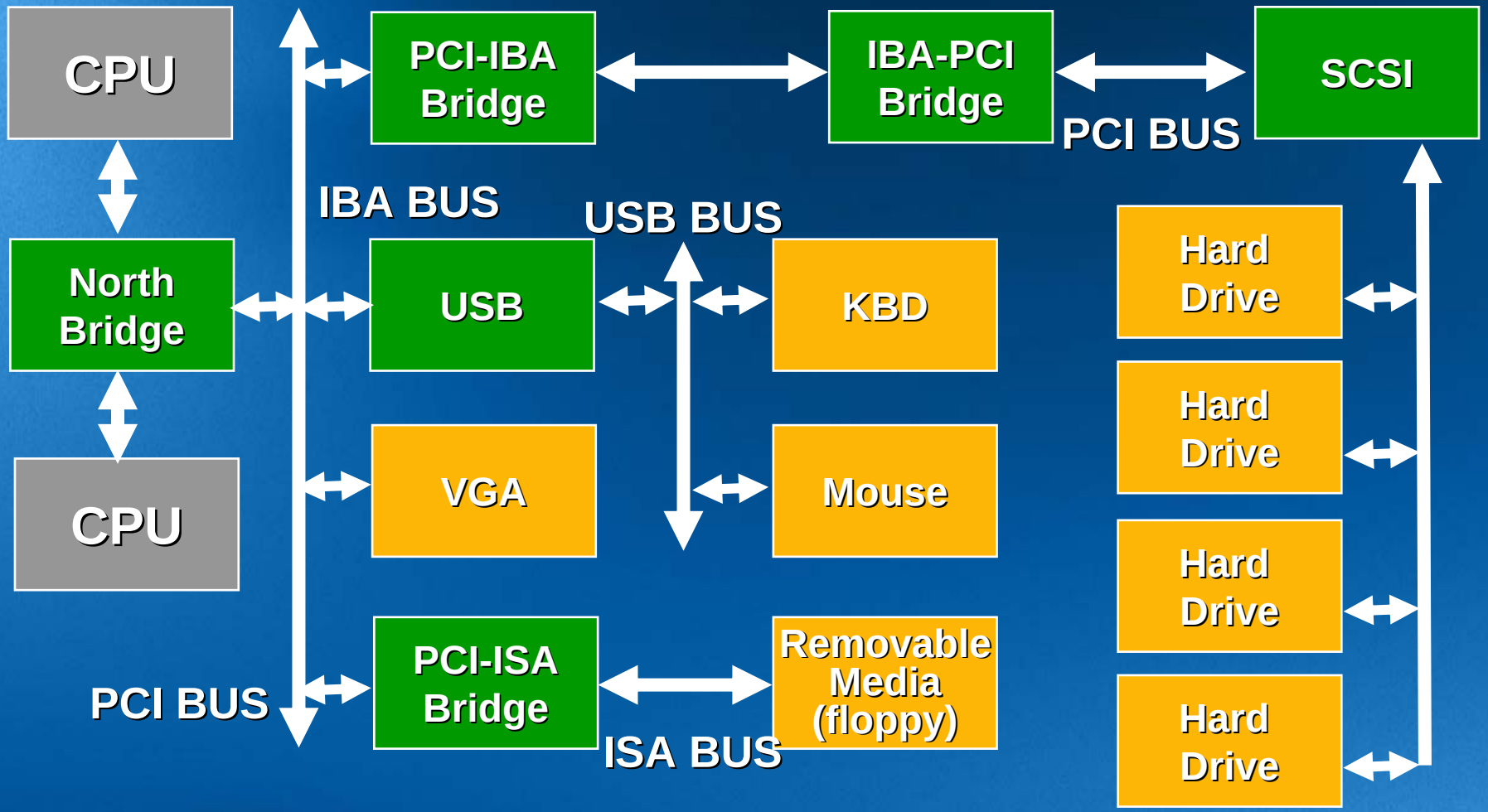
- UEFI Basics
- **UEFI Drivers**
- EFI Shell Basics
- Where to find resources

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"

Complex System Example



Manageable by UEFI Driver Model

See § 2.5 UEFI 2.1 Spec.

Legacy Vs. UEFI

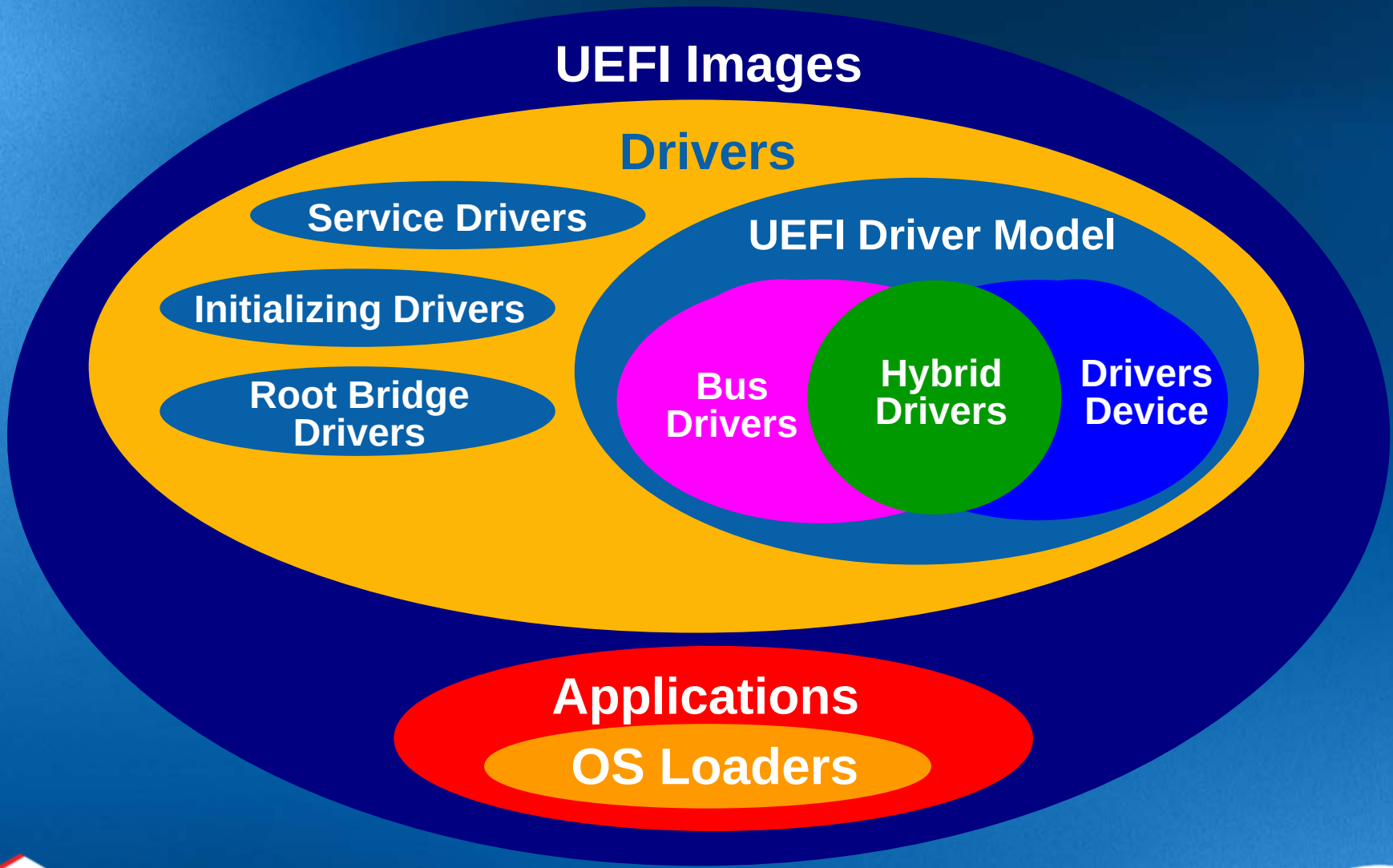
Legacy vs. UEFI

- **32-bit / 16-Bit Real Mode**
  - Legacy can not be used in platforms that do not support the execution of IA-32 real mode
  - Access to lower 1 MB of system memory only
- **Fixed Resources for Working with Option ROMs**
  - Memory, I/O, PCI, DMA have limitations
- **Issues Initializing Option ROMs to their Devices**
- **Ties to PC-AT System Design**
  - Directly access hardware and memory
- **Ambiguities in Specification and Workarounds Born of Experience**
  - no clear specifications on how to write a legacy option ROM
  - workarounds because of incompatibilities
  - not always clear which device will be the boot device

Legacy vs. UEFI

- **32-bit Real Mode**
  - Flat mode
  - Portable
  - Single Binary
- **Fixed Resources for Working with Option ROMs**
  - Full access to system components
  - Relocation capability
  - Boot services provide Memory allocation
- **Associate Matching Option ROMs to their Devices**
  - UEFI Provides deterministic matching
- **Ties to PC-AT System Design**
  - Well defined Protocols
- **Ambiguities in Specification and Workarounds Born of Experience**
  - UEFI Specification followed
  - UEFI allows for Platform overrides for flexibility

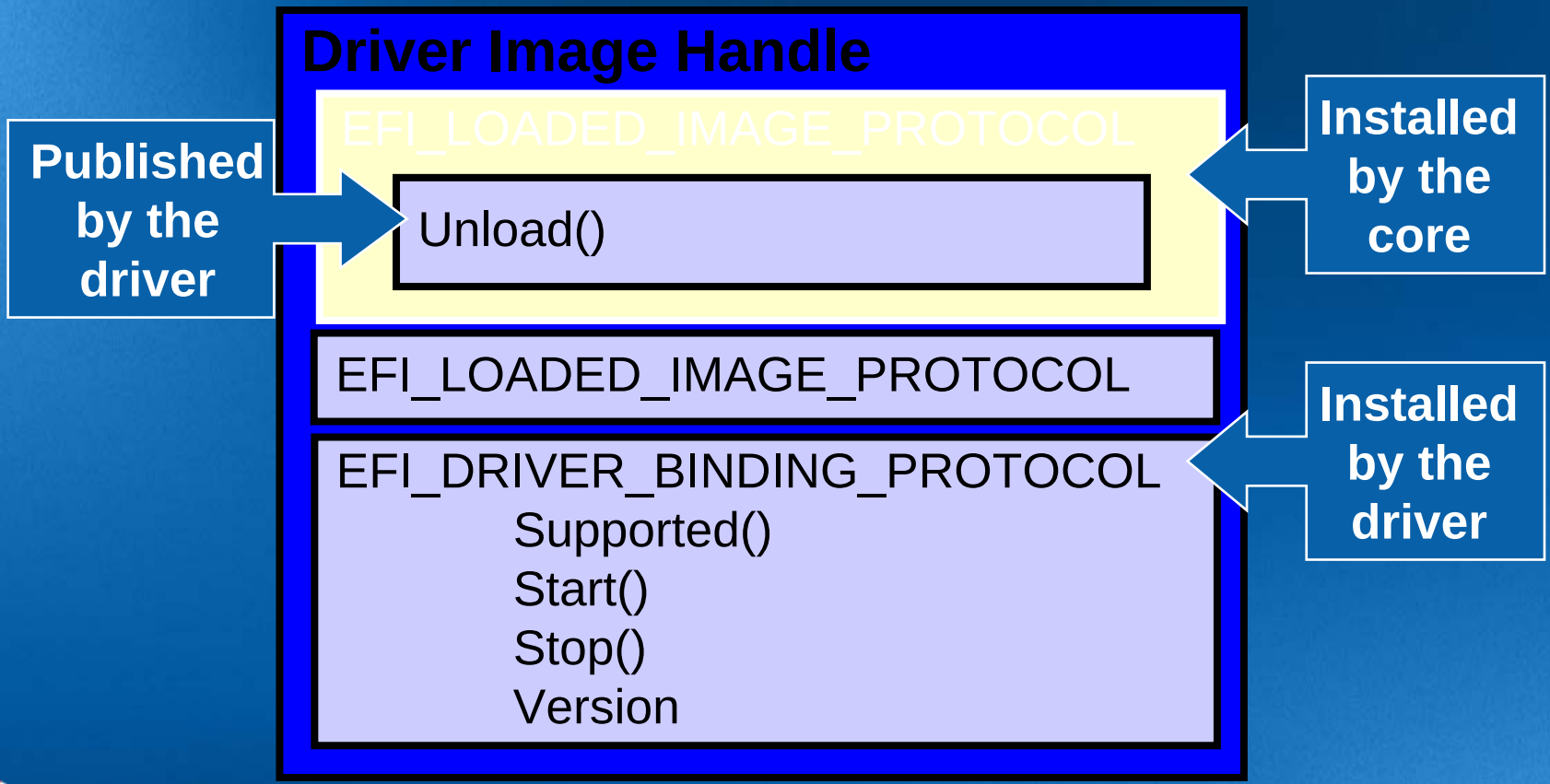
UEFI Image Types



UEFI Driver Examples in Backup

Responsibilities of Driver Writer

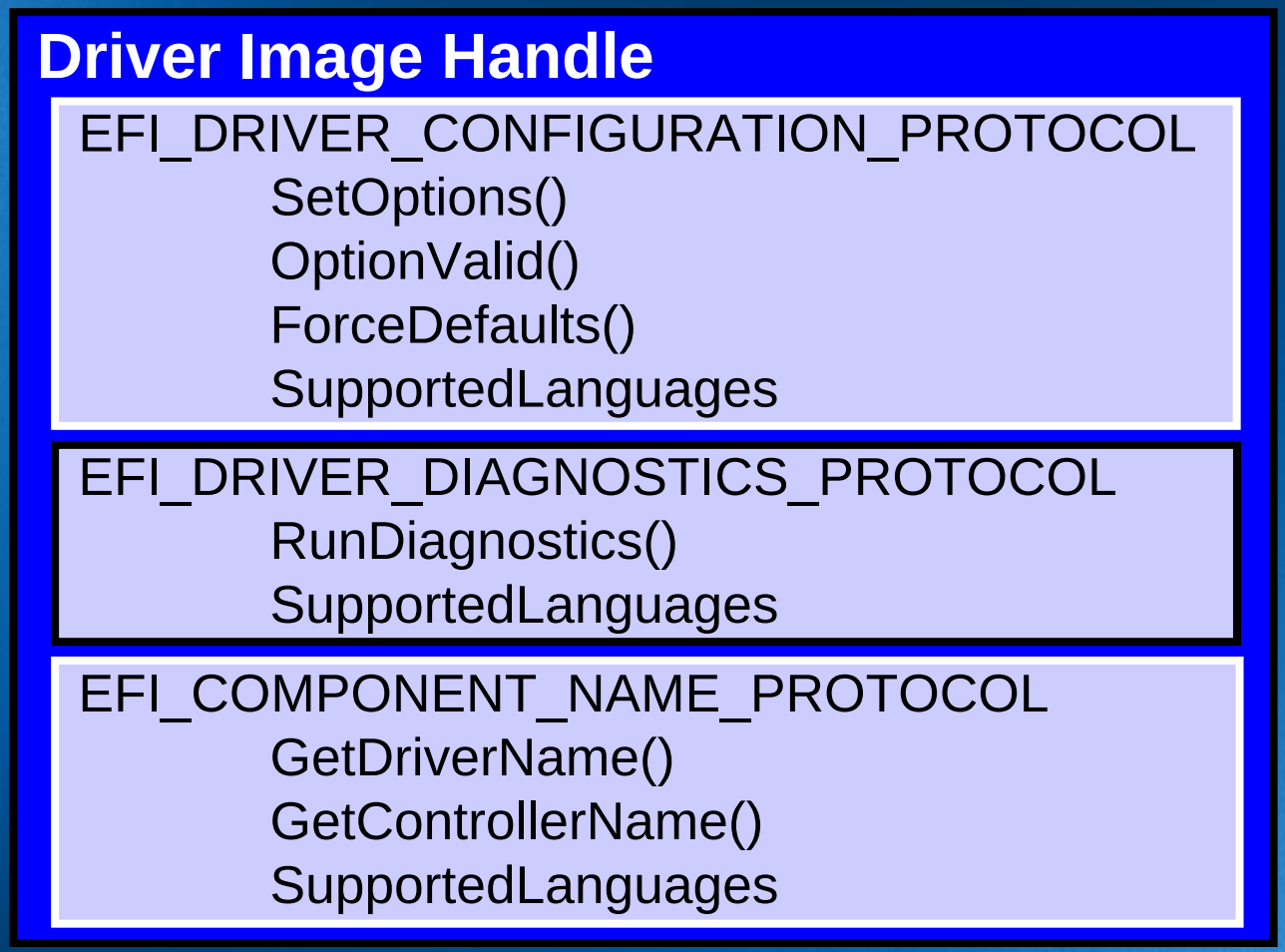
- Driver Image Handle Required Protocols



See § 2.5.2 UEFI 2.1 Spec.

More Responsibilities of Driver Writer

- Driver Image Handle Optional Protocols



Agenda

- UEFI Basics
- UEFI Drivers
- **EFI Shell Basics**
- Where to find resources

EFI Shell Overview

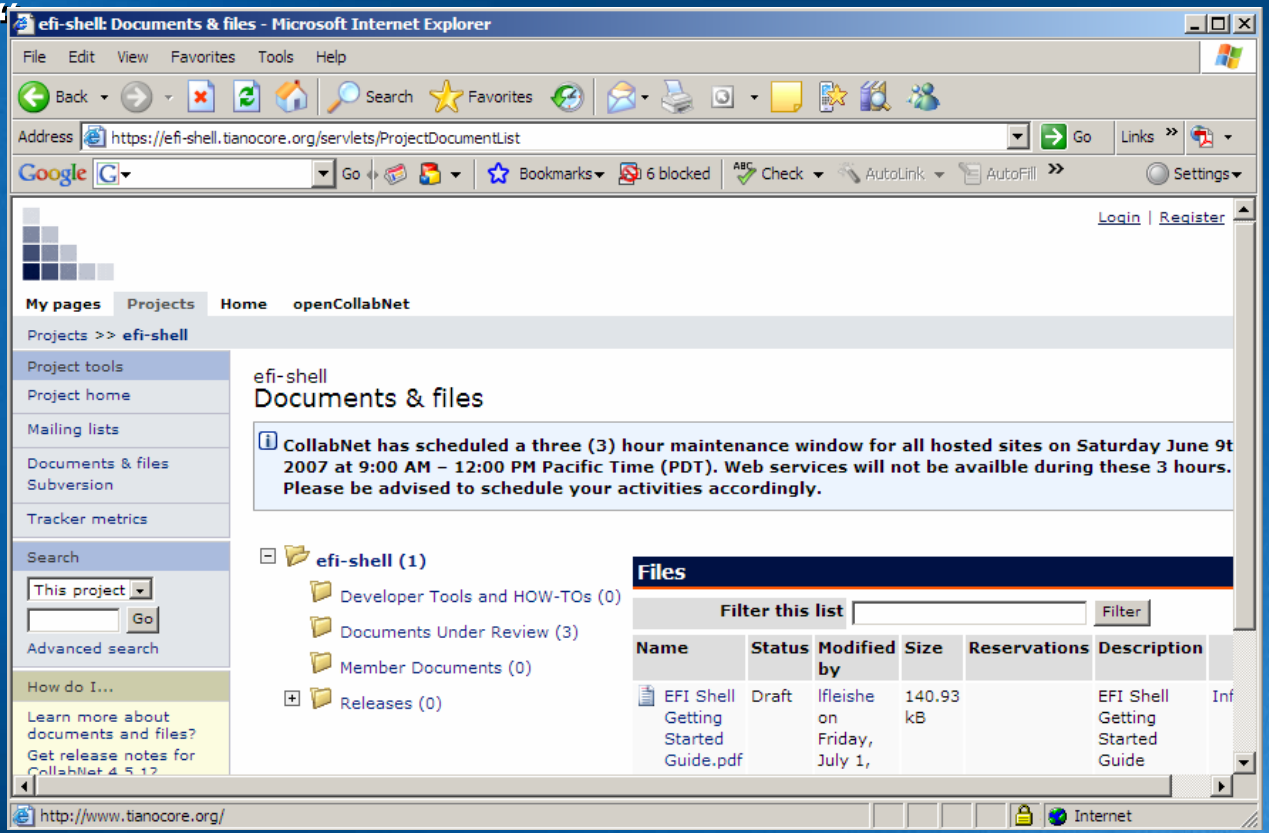
- Interactive way to use UEFI code in system
- Has command line prompt and Scripting
  - Is similar to DOS and Linux* shell but not EXACTLY - its own unique syntax
- Is an EFI executable in itself
- Knows only about EFI file systems that are FATxx
- Shell is a Sub-project on EFI Development Kit (EDK) on the EFI and Framework Open source Community Website
  - Shell programs
  - users' guide
  - EFI Shell Source
- EDK has Binary UEFI Shell for processors IA32, Intel® 64, IA-64

EFI Shell Open Source

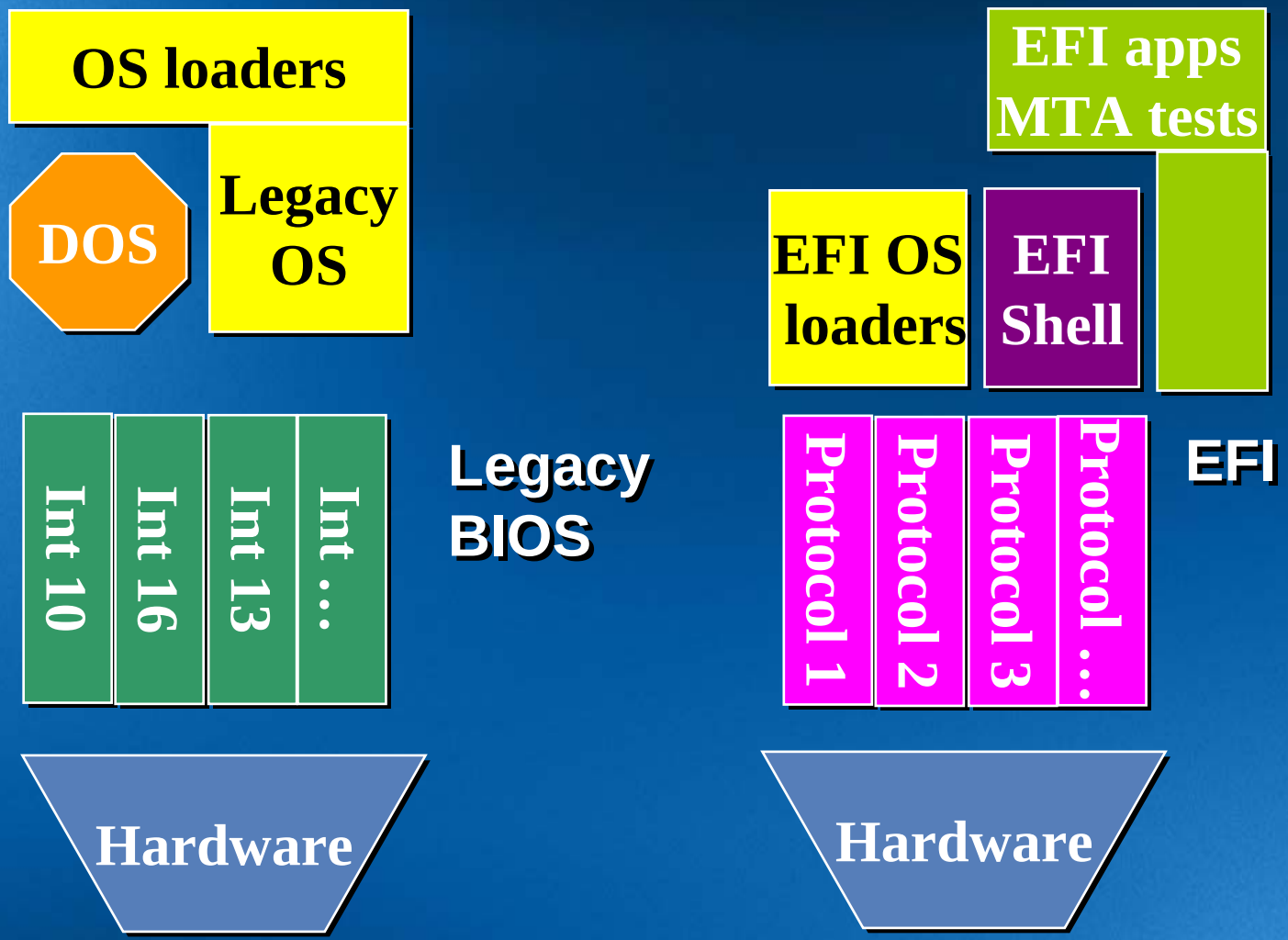
WEB Site: http:\\www.tianocore.org

Project: EFI-Shell

Documents & files : “EFI Shell Getting Started Guide “



Analogy to Old DOS: BIOS



Shell Usage

- Execute preboot programs
  - Setup
  - operating system install
  - Test
  - disk utilities
  - Driver Diagnostics, Configurations
- Move files around between the hard disk, floppy disk, CD-ROM, USB flash devices, and so on
- Load a preboot EFI driver in the system (has an .efi suffix), examples:
  - LAN stack tcpip drivers
  - Update old drivers in flash
  - New drivers for plugin cards
- Shell.efi verses Shell_full.efi
  - Shell.efi smaller to fit in Flash
  - Shell_full.efi Richer commands

EFI File System

Can manipulate EFI system fatxx partition only where boot loader and EFI application are

```
Shell> map
Device mapping table
fs0  : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/
 Scsi(Pun0,Lun0)/HD(Part1,Sig8983DFE0-F474-01C2-507B-
 9E5F8078F531)
blk0 : Acpi(PNP0A03,0)/Pci(1F|1)/Ata(Primary,Slave)
blk1 : Acpi(PNP0A03,0)/Pci(1F|1)/Ata(Primary,Master)
blk2 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/Scsi(Pun0,Lun0)
blk3 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/
  Scsi(Pun0,Lun0)/HD(Part1,Sig8983DFE0-F474-01C2-507B-
  9E5F8078F531)
blk4 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/
  Scsi(Pun0,Lun0)/HD(Part2,Sig898D07A0-F474-01C2-F1B3-
  12714F758821)
blk5 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/
  Scsi(Pun0,Lun0)/HD(Part3,Sig89919B80-F474-01C2-D931-
  F8428177D974)
```

Device Path

```
fs0  :
  Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/Scsi(Pun0,Lun0)/H
  D(Part1,
     Sig8983DFE0-F474-01C2-507B-9E5F8078F531)

Fs0:
Acpi(PNP0A03,1)
Pci(1F|0)/Pci(2|0)
Scsi(Pun0,Lun0)
HD(Part1,Sig8983DFE0-F474-01C2-507B-
  9E5F8078F531)
```

EFI Shell Commands
Help ?

dh
help ?
map
mount
load
unload
loadbmp
nshell
ver
memmap
bcfg
Dblk
alias

dmem
dmpstore
err
guid
pci
mm
reset
stall
getmtc
hexedit
Setsize
Set

drivers
devtree
devices
connect
disconnect
openinfo
reconnect
drvcfg
drvdiag
loadpcirom

Main EFI Shell Commands For EFI / UEFI Drivers

- dh
  - Displays handles in the EFI environment
- map
  - Displays or defines mappings
- drivers
  - Displays drivers and attributes in database
- connect / disconnect
  - Start and Stop managing a driver
- drvcfg
  - Run the driver's Configuration program
- drvdiag
  - Run the Driver's Diagnostic program

Back up has more on EFI Shell

Agenda

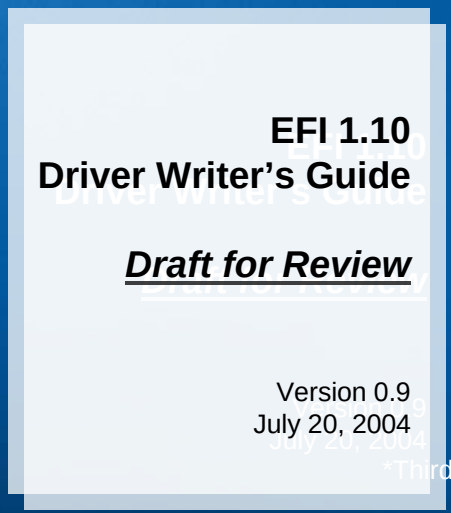
- UEFI Basics
- UEFI Drivers
- EFI Shell Basics
- **Where to find resources**

Where to find resources

- EFI Driver Writers Guide
- EFI Development Kit (EDK) – Open Source
  - DUET
- UEFI Forum Web site
  - UEFI Specification 2.1
  - PI Specification 1.0
  - UEFI SCT 2.0

EFI Driver Writer's Guide

- Captures Practical Experiences
- Use as a Recipe Book
- Must Read for all EFI Driver Developers
- Living Document
  - Content Based on Industry Feedback
  - Updated as Techniques are Refined
  - Updated as New Technologies are Introduced



Current version Intel Web site :
http://www.intel.com/technology/efi
Soon to be on UEFI with UEFI 2.1
http://www.uefi.org

General Topics

- Overview of EFI Concepts
- EFI Services
  - Commonly Used by EFI Drivers
  - Rarely Used by EFI Drivers
  - Should Not Be Used by EFI Drivers
- General Driver Design Guidelines
- Classes of EFI Drivers
- Driver Entry Point
- Private Context Data Structures
- EFI Driver Model Protocols

Platform Specific Topics

- PCI Driver Guidelines
- USB Driver Design Guidelines
- SCSI Driver Design Guidelines
- Size Optimizations
- Speed Optimizations
- EFI Byte Code (EBC) Considerations
- Building/Testing/Debugging EFI Drivers

Benefits of following EFI Driver Guidelines

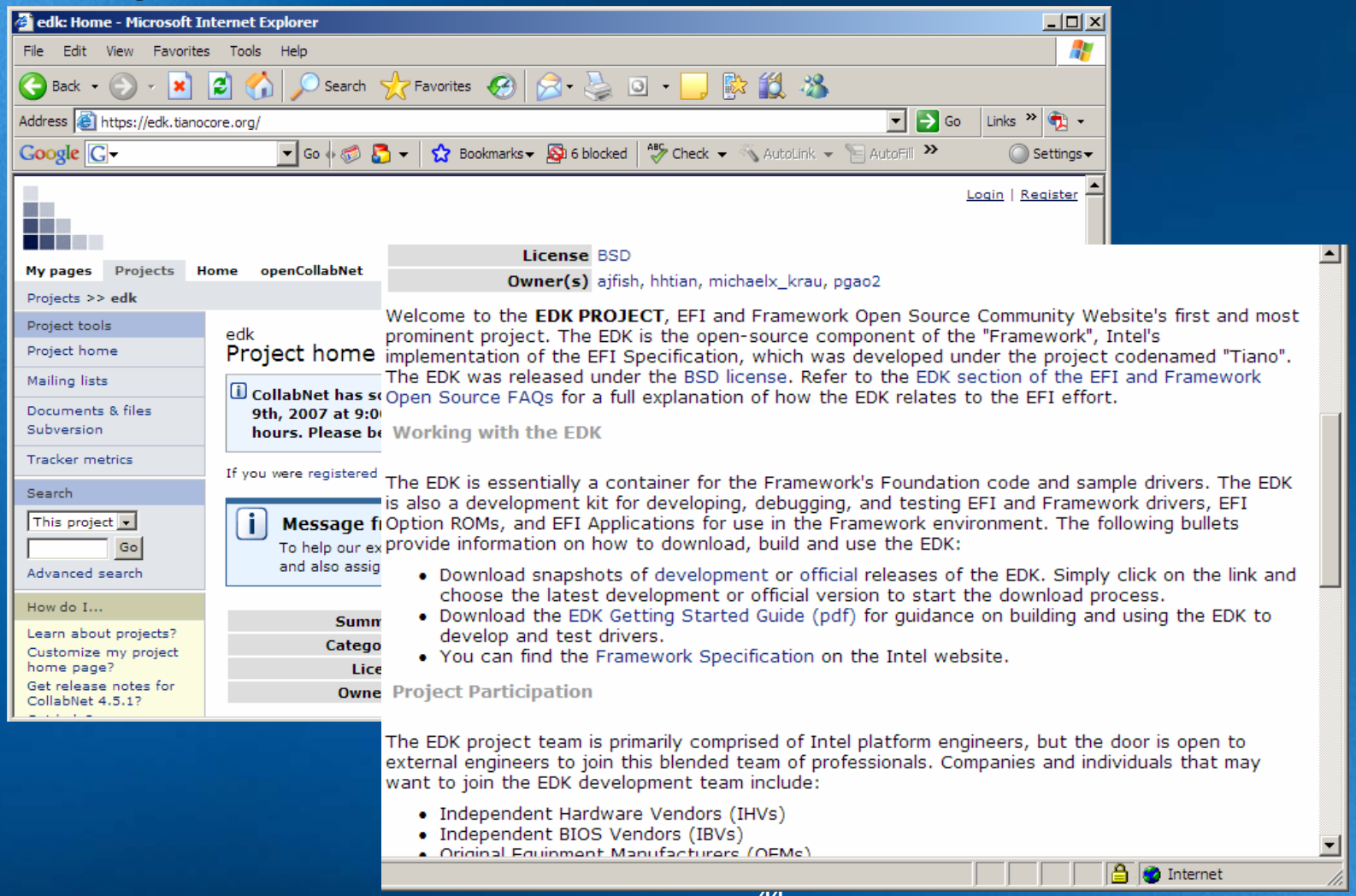
- Following EFI Driver Guidelines
  - Improves Portability, Quality, and Interoperability
  - Reduces Implementation Effort
  - May Increase Performance
  - May Reduce FLASH Overhead

EFI Driver Writer's Guide Helps Improve UEFI Drivers

EFI Developer Kit (EDK)

WEB Site: http:\\www.tianocore.org

Project: EDK



EDK Build Tips

Build Tip to use	Works on
DUET	building a development boot environment
NT32	IA-32
X64	Intel® 64
IPF	IA-64 (Itanium® processors)
EBC	IA-32 Intel® 64 IA-64

What is Developers UEFI Emulation (DUET)

- DUET – UEFI Over Legacy BIOS
- Why DUET?
  - Provide IHV an EFI/UEFI environment above legacy BIOS, to help them develop and debug their native EFI/UEFI drivers.
- Enter condition:
  - Hardware Initialization done. Legacy interfaces available.
  - Legacy boot to DUET.
- Exit condition:
  - Provide pure EFI/UEFI environment. (IA32/X64)
  - Boot to EFI/UEFI Shell/OS.

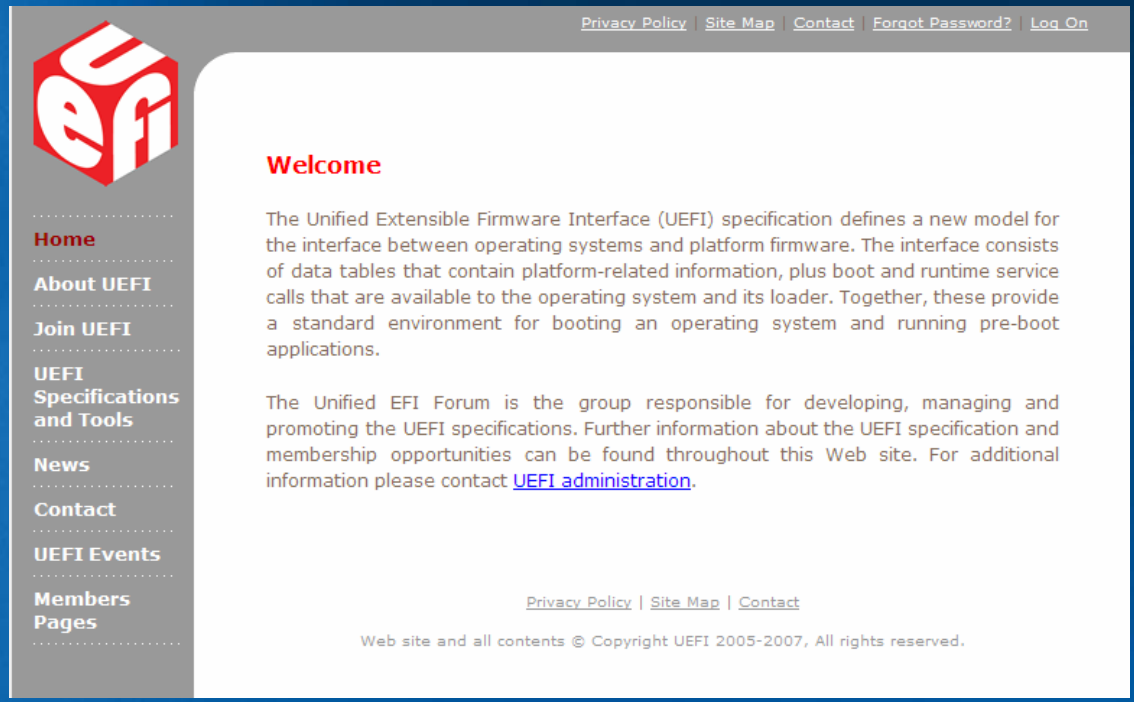
Goal of DUET

- Goal is ...
  - Export EFI/UEFI interface
  - Support IA32 and X64 architecture
  - Chipset/Platform independent
  - Boot from Floppy
  - Boot from USB (Legacy Free Consideration)
  - Boot from Hard Disk
  - Support boot to EFI/UEFI Shell

DUET Goal (Cont'd)

- Goal is not:
  - Not all Framework interfaces are supported, (for example: PEI-CIS, DXE-CIS)
  - Not support IA-64
  - Not support CSM, INTx call (except Video), and 16bit code
  - Not support boot to Legacy OS
  - Not support boot to OS

Unified EFI Web site



- http://uefi.org
- Specifications and Working Groups
- UEFI Self Certification Test Suite (SCT)
- Join

Summary

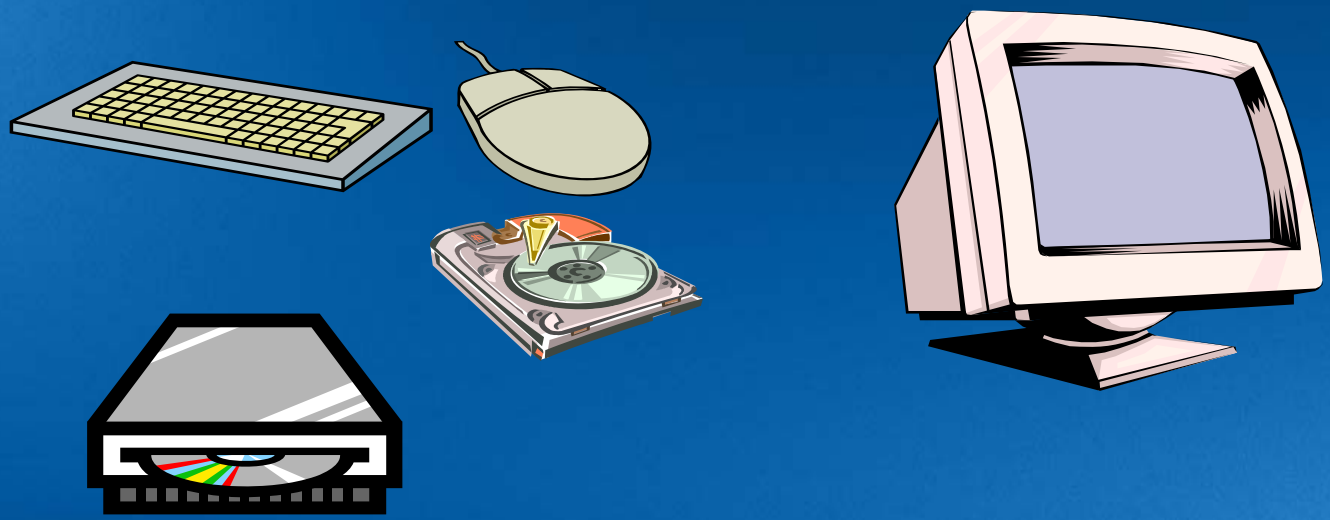
- Communication is the key for Supporting UEFI and those producing UEFI products
- Training to know
  - UEFI Basics
  - UEFI Drivers
  - EFI Shell Basics
- Where to find resources
  - UEFI 2.1 specification
    - http://www.uefi.org
  - EDK and EFI Shell + Documentation
    - http://www.TianoCore.org
  - EFI Driver Writer's Guide
    - http://developer.intel.com/technology/efi


intel®
Leap ahead™

UEFI Drivers
BACK UP

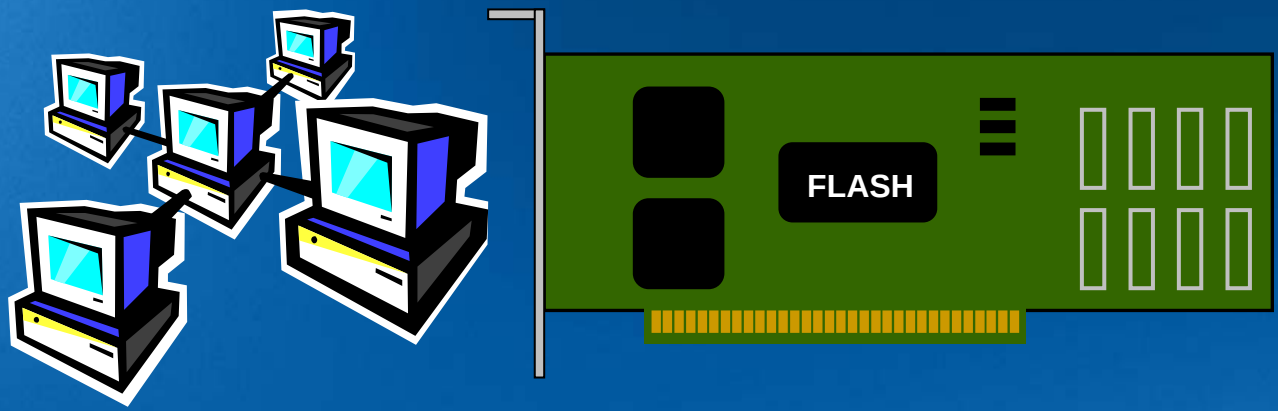
Example of Device Drivers

- PCI Video Adapters
- USB Host Controllers
- USB Keyboards / USB Mice
- PS/2 Keyboards / PS/2 Mice



Example of Bus Driver

- PCI Network Interface Controllers
- Serial UART Controllers



Hybrid Drivers

- Manages and Enumerates a Bus Controller
- Start() Creates One or More Child Handles
- Start() Produces Bus Specific I/O Protocols
  - Installed onto the Bus's Controller Handle
  - Installed onto Bus's Child Handles

Examples:

PCI SCSI Host Controllers

PCI Fiber Channel Controllers

EFI Shell Back up

Executing EFI Applications

- Enter the application name at the command prompt
- The EFI shell uses “path” environment variable
- The EFI shell automatically adds “.efi”

```
path = fs0:>\efi\tools;fs0:\;.
     fs0:> test.efi
     fs0:> test
     fs1:> test
```

If suffix is .nsh, it will execute shell commands inside of the .nsh file similar to .bat in DOS

Wild Card Expansion

- Syntax is similar to DOS commands
- '*' Matches any string
- '?' Matches any character
- [chars] Matches set of characters

```
fs0:> dir *.efi

fs0:> dir test?.efi
```

Output Redirection

- Redirect standard output to a Unicode file
```
memmap 1> Foo.txt
memmap > Foo.txt
```
- Redirect standard output to an ASCII file
```
memmap 1>a Foo.txt
memmap >a Foo.txt
```
- Append a file using redirection of standard output
```
memmap 1>>a Foo.txt
memmap >>a Foo.txt
```
- Redirect standard error to a Unicode file
```
memmap 2> Foo.txt
```
- Redirect standard error to an ASCII file
```
memmap 2>a Foo.txt
```

DH – Dump Handle

- All devices and images are in a database called the handle database
- Devices are located in the system by their device paths

```
DH [-b] [-d] [-lXXX] [-v] [handle]|[-p prot_id]
Displays the handles in the EFI environment
-b              - Displays one screen at a time
handle          - Dumps information of a certain handle
-p prot_id      - Dumps all handles of a certain protocol
-d              - Dumps EFI Driver Model related information
-lXXX           - Dumps information using the ISO 639-2 language
                  specified by XXX
-v              - Dumps information on all handles
```

DH – Dump Handle

Example from the Dump Handle command in the EFI Shell

```
Handle dump
  1: Varstore
  2: Image(EFI Core)
  3: DevIo DevPath ()
  4: Varstore
  5: UnicodeCollation
  6: Image(NVRAM API Driver 1.00)
  7:
  8: Image(BiosDisk)
  9: DevPath (Acpi(PNP0A03,0)/Pci(1F|1))
  A: Fs DiskIo BlkIo DevPath (..Pci(1F|1)/Ata(Primary,Slave))
  B: DiskIo BlkIo DevPath (..ci(1F|1)/Ata(Primary,Master))
 10: Image(Decompress)
 11: Decompress
 12: Image(PcatPciRootBridge)
 13: PciRootBridgeIo DevPath (Acpi(PNP0A03,0))
 14: PciRootBridgeIo DevPath (Acpi(PNP0A03,1))

 75: Image(VenHw(6D9FEEB1-E585-22D3-BC22-0080C73C8881))
   DriverBinding ComponentN
     ame Configuration

 A4: Image(nshell) ShellInt
 A5: Fs DiskIo BlkIo ESP DevPath (..F474-01C2-507B-9E5F8078F531))
 A6: DiskIo BlkIo DevPath (..F474-01C2-F1B3-12714F758821))
```

DH – Dump Handle

```
fs0:\> dh -d 75
 75: Image(VenHw(6D9FEEB1-E585-22D3-BC22-0080C73C8881))
  DriverBinding ComponentName Configuration
     Driver Name     : LSI Logic Ultra320 SCSI Driver
     Image Name      : VenHw(6D9FEEB1-E585-22D3-BC22-
  0080C73C8881)
     Driver Version  : 01020B00
     Driver Type     : DEVICE
     Configuration   : YES
     Diagnostics     : NO
     Managing        :
       Ctrl[46] : LSI Logic Ultra320 SCSI Controller
       Ctrl[47] : LSI Logic Ultra320 SCSI Controller
```

DH – Dump Handle

```
fs0:\> dh 46
Handle 46 (3D10E388)
   PciIo (3D10AC98)
     Location......: Seg(00)  Bus(06)  Dev(02)  Func(00)
     Configuration.: VendorID(1000)  DeviceID(0030)  ClassCode(00 00 01)
     BAR #0........: I/O      Base(            8800)  Length(100)
     BAR #1........: MEM64    Base(        F8FD0000)  Length(10000)
     BAR #2........: MEM64    Base(        F8FC0000)  Length(10000)
     ROM...........:          Base(               0)  Length(0)
     Configuration Header :
       00 10 30 00 57 01 30 02 07 00 00 01 20 40 80 00
       01 88 00 00 04 00 FD F8 00 00 00 00 04 00 FC F8
   ScsiPassThru (3D0DD720)
   Dpath (3D10EC08)
      ACPI Device Path for ACPI
       HID A0341D0, UID 1
      Hardware Device Path for PCI
       Function (0) Device (1F)
      Hardware Device Path for PCI
       Function (0) Device (2)
      AsStr: 'Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)'
```

map: Map a Short Name to a Device

```
map [-dvr] [sname] [handle]
  [-d]                  Delete a mapping
  [-v]                  Verbose listing of mappings
  [-r]                  Regenerate default mappings
  [sname]               A user defined name for the mapping
                        Can use “” to include white space
  [handle]              The handle number from the dh command

fs1:\> map
Device mapping table
  fs0  :
  Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/Scsi(Pun0,Lun0)/HD(Part1,Sig8983DFE0
              -F474-01C2-507B-9E5F8078F531)
  blk0 : Acpi(PNP0A03,0)/Pci(1F|1)/Ata(Primary,Master)
  blk1 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/Scsi(Pun0,Lun0)
  blk2 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/Scsi(Pun0,Lun0)/HD
              (Part1,Sig8983DFE0
```

UEFI Drivers

```
                T   D
D               Y C I
R               P F A
V  VERSION      E G G #D #C DRIVER NAME                          IMAGE NAME
== ======== == = = == == ================================== ==============
18 00000010 B - -  5 38 PCI Bus Driver                          PciBus
19 00000010 D - -  1  - PC-AT ISA Device Enumeration Driver     PcatIsaAcpi
1A 00000010 B - -  1  3 ISA Bus Driver                          IsaBus
1B 00000010 B - -  2  2 ISA Serial Driver                       IsaSerial
1E 00000010 D - -  1  - UGA Console Driver                      GraphicsConsole
1F 00000010 B - -  2  1 Serial Terminal Driver                  Terminal
20 00000010 D - -  2  - Platform Console Management Driver      ConPlatform
21 00000010 D - -  2  - Platform Console Management Driver      ConPlatform
25 00000010 B - -  2  2 Console Splitter Driver                 ConSplitter
26 00000010 ? - -  -  - Console Splitter Driver                 ConSplitter
27 00000010 B - -  2  2 Console Splitter Driver                 ConSplitter
28 00000010 B - -  2  2 Console Splitter Driver                 ConSplitter
2C 00000010 D - -  2  - Usb Uhci Driver                         UsbUhci
2D 00000010 B - -  2  1 USB Bus Driver                          UsbBus
2F 00000010 ? - -  -  - Usb Bot Mass Storage Driver             UsbBot
30 00000010 ? - -  -  - Usb Cbi0 Mass Storage Driver            UsbCbi0
66 00000010 D - -  1  - Simple Network Protocol Driver          Snp3264
67 00000010 D - -  1  - PXE Base Code Driver                    PxeBc
68 00000010 D - -  1  - PXE DHCPv4 Driver                       PxeDhcp4
69 00000010 ? - -  -  - Usb Mouse Driver                        UsbMouse
75 01020B00 D X -  2  - LSI Logic Ultra320 SCSI Driver          VenHw(6D9FEEB1
```

EFI Drivers

Connect / Disconnect Driver
Start / Stop Managing a Driver

```
fs0:\> dh -d 75
 75: Image(VenHw(6D9FEEB1-E585-22D3-BC22-0080C73C8881)) DriverBinding
  ComponentN
ame Configuration
     Driver Name     : LSI Logic Ultra320 SCSI Driver
     Image Name      : VenHw(6D9FEEB1-E585-22D3-BC22-0080C73C8881)
     Driver Version  : 01020B00
     Driver Type     : DEVICE
     Configuration   : YES
     Diagnostics     : NO
     Managing        :
       Ctrl[46]  : LSI Logic Ultra320 SCSI Controller
       Ctrl[47]  : LSI Logic Ultra320 SCSI Controller
fs0:\> disconnect 46
DisconnectController(3D10E388,0,0) : Status = Success

75 01020B00 D X -  1  - LSI Logic Ultra320 SCSI Driver  VenHw(6D9FEEB1-E585

fs0:\> reconnect 46
Scsi(Pun0,Lun0) MAXTOR  ATLASU320_18_SCAB120 ( 40 MBytes/sec)
Scsi(Pun6,Lun0) ESG-SHV
ReconnectController(3D10E388,0,0) : Status = Success
```