phoenix technologies

UEFI Overview

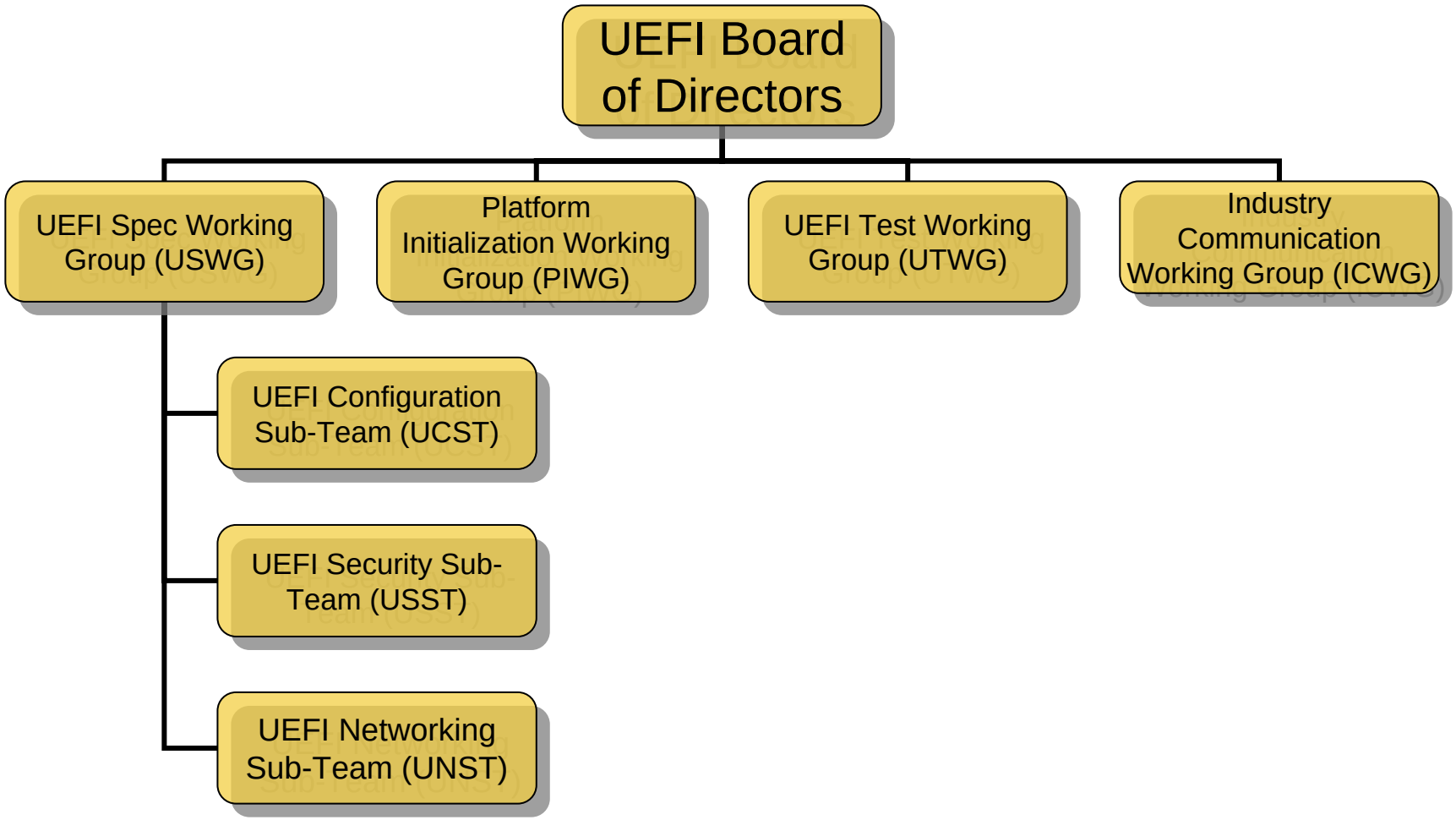
Tim Lewis

6 June 2007

Agenda

- UEFI Organization
- How UEFI Changes BIOS
- UEFI Key Ideas
- Summary

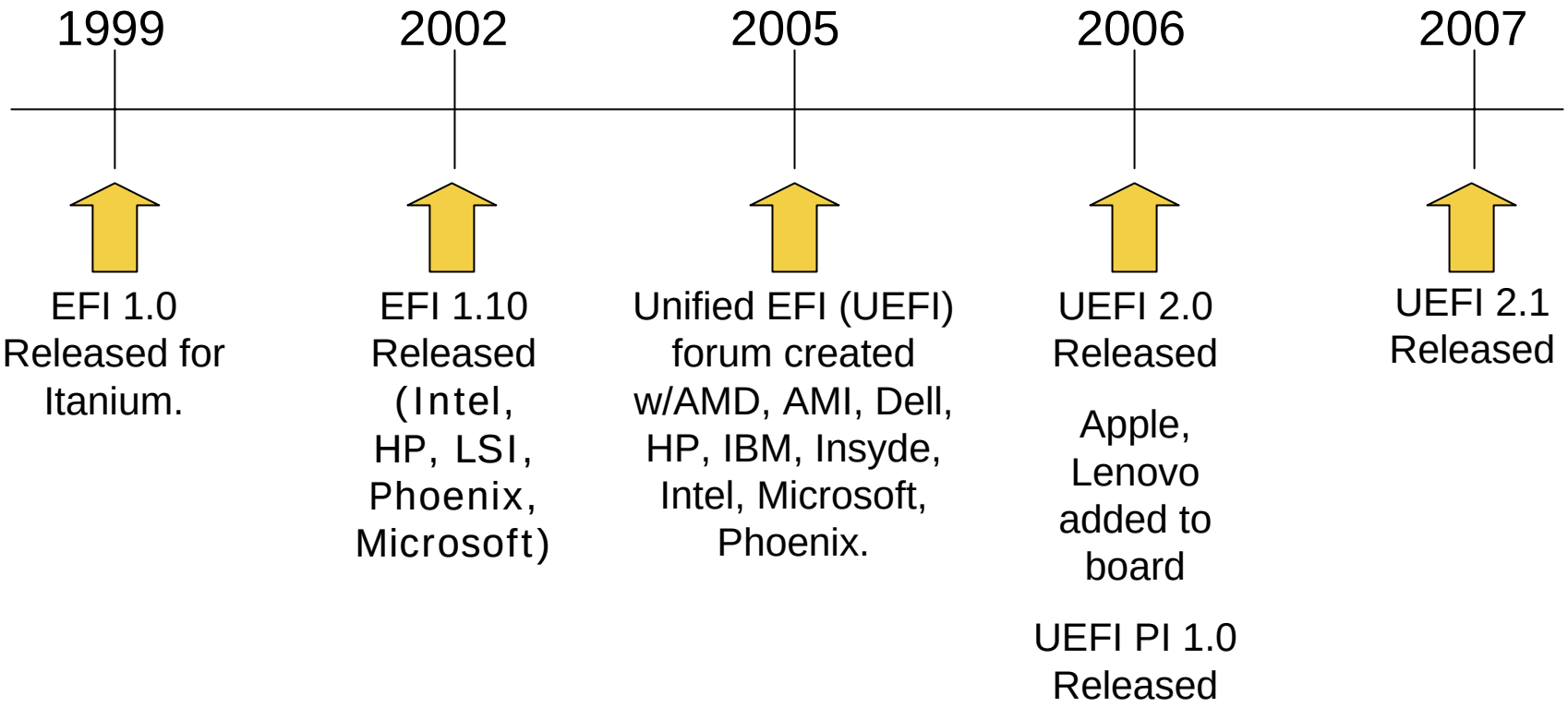
UEFI Forum Organization



UEFI Goals

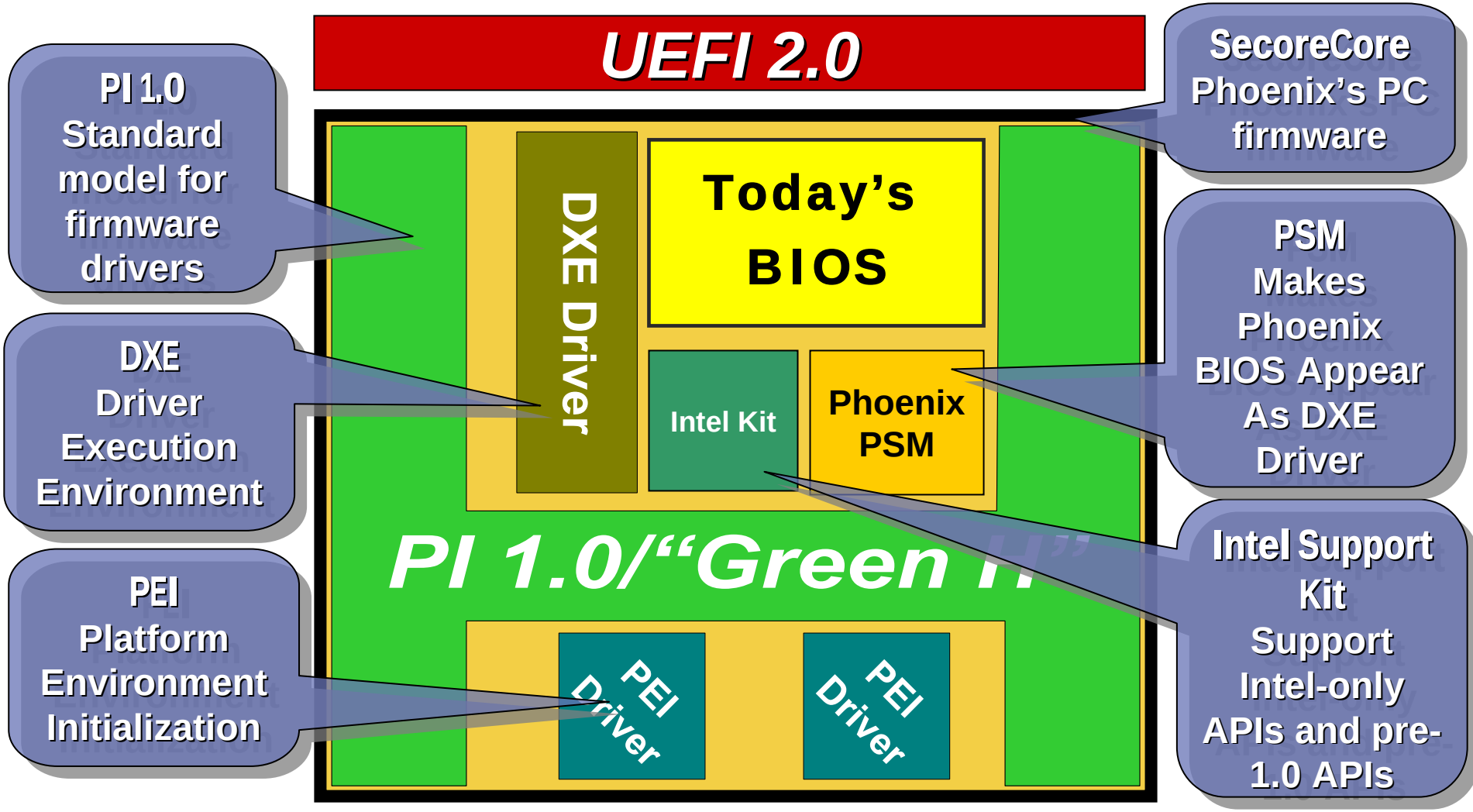
- Open (UEF**I**)
  - Clear specification of the boot environment allows 3rd party drivers and 3rd party applications, as long as they followi the interface.
- Extensible (U**E**FI)
  - New interfaces and capabilities can be added & prototyped, even w/o changing the UEFI specification.
- Industry Controlled (**U**EFI)
  - Provides balance between OS vendors, OEMs, firmware providers and 3rd party developers
- Modular
  - Can be created and delivered separately.
- Scalable
  - Applicable to a wide variety of platforms, from embedded and special purpose, up through multi-node servers.

Trend #1: Innovation Speeding Up



BIOS and UEFI Coexist For A Long Time (Compatibility)

UEFI Trend - Compatibility

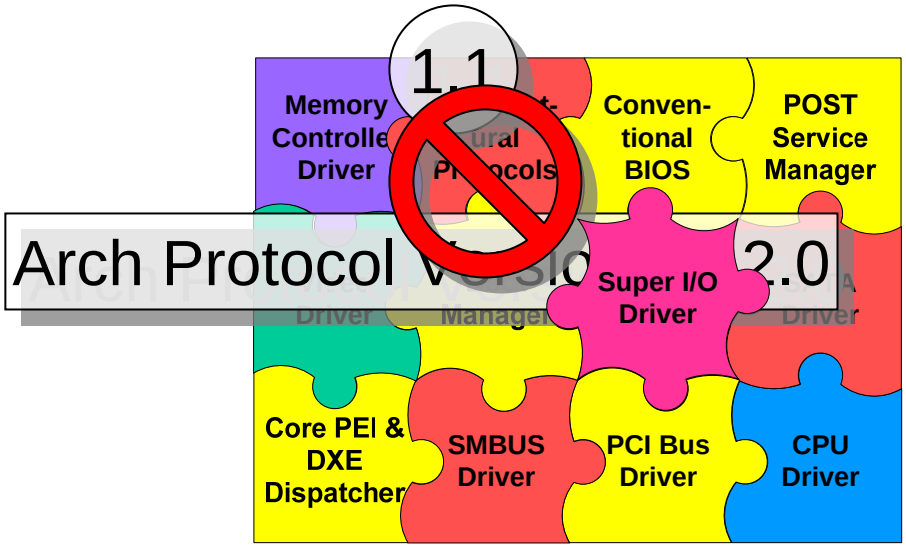


Trend #2: Monolithic To Modular

- BIOS will be a mix of drivers and applications provided by Phoenix, silicon vendors, OS vendors, 3rd party ISVs, OEMs and ODMs.
  - Tiano, UEFI 2.0, etc.
- Drivers and applications are **delivered *separately*** rather than a single deliverable.
- Drivers and applications are **installed *separately***
  - Even after the system is shipped!

Many Pieces Plug Into BIOS (Integration)

UEFI Trend - Integration



Do I have all of the drivers I need?

Do I have the right versions?

Is it configured for my platform?

What if I want to make a change?

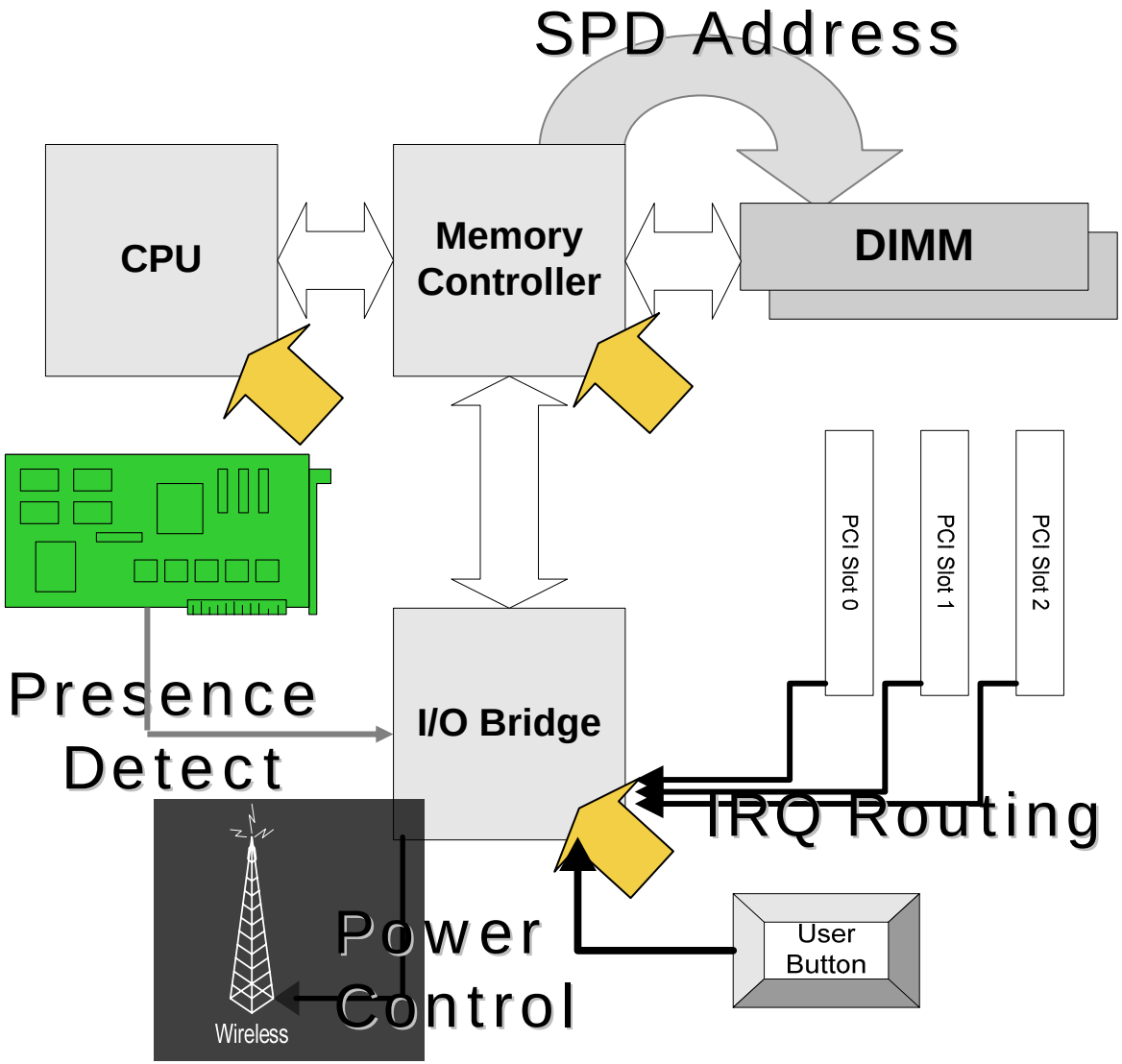
Trend #3: Product To Platform

- Stable, standards-based platforms allow value-add development with assurance of longevity.
  - Testing insures compatibility
- Every BIOS vendor and major system vendor either has or is planning UEFI-capable systems.
- Features and applications can be moved from platform to platform.

It's Not How You Build Your BIOS

But How You Build Your Feature (Development)

UEFI Trend - Configuration



Driver for each of the components

But drivers need to be configured

How to do it?

UEFI Specification Overview

What Is The UEFI Specification?

- Standard boot process
  - Method for finding drivers
  - Method for finding an OS loader
  - Method for launching an OS loader
  - Standard boot and runtime services
- On PC/AT systems, replaces BIOS real-mode interface
  - Works alongside of ACPI and SMBIOS
- Manages initial program load (IPL) devices
  - Boot devices, video devices and input devices
- Managed by the UEFI Specification Working Group
- Current Version Is 2.1

Key Idea #1: In UEFI, Everything Is An EXE

- UEFI Drivers, Applications, OS Loaders Are EXEs
  - Called *images* in UEFI
  - Same file format used by Windows (PE32+)
  - Three new subsystem types:
    - EFI RUNTIME, EFI BOOT SERVICE, EFI APPLICATION
- UEFI Images Can Be 32-Bit or 64-Bit
  - Flat mode. If paged, then 1-to-1 linear/virtual mapping
- UEFI Images Can Be For EBC, x86-32, Itanium & x86-64
  - EFI Byte Code (EBC) is interpreted assembly language
- UEFI Images Have A Single Entry Point (Not Many)

Key Idea #2: Globally Unique Identifier (GUID)

- GUIDs are identifiers
  - Sometimes called UUIDs
  - UEFI uses for interfaces and data structures
- GUIDs are unique
  - Extremely difficult to generate two duplicate GUIDs
- Easy to create your own GUID
  - Run UUIDGEN or GUIDGEN from Microsoft or Linux
- If you create your own GUID, you can create your own interface or data structure in UEFI.
  - GUID1 = UEFI Specification
  - GUID2 = Phoenix Specification
  - GUID3 = OEM Defined
  - GUID4 = ODM Defined

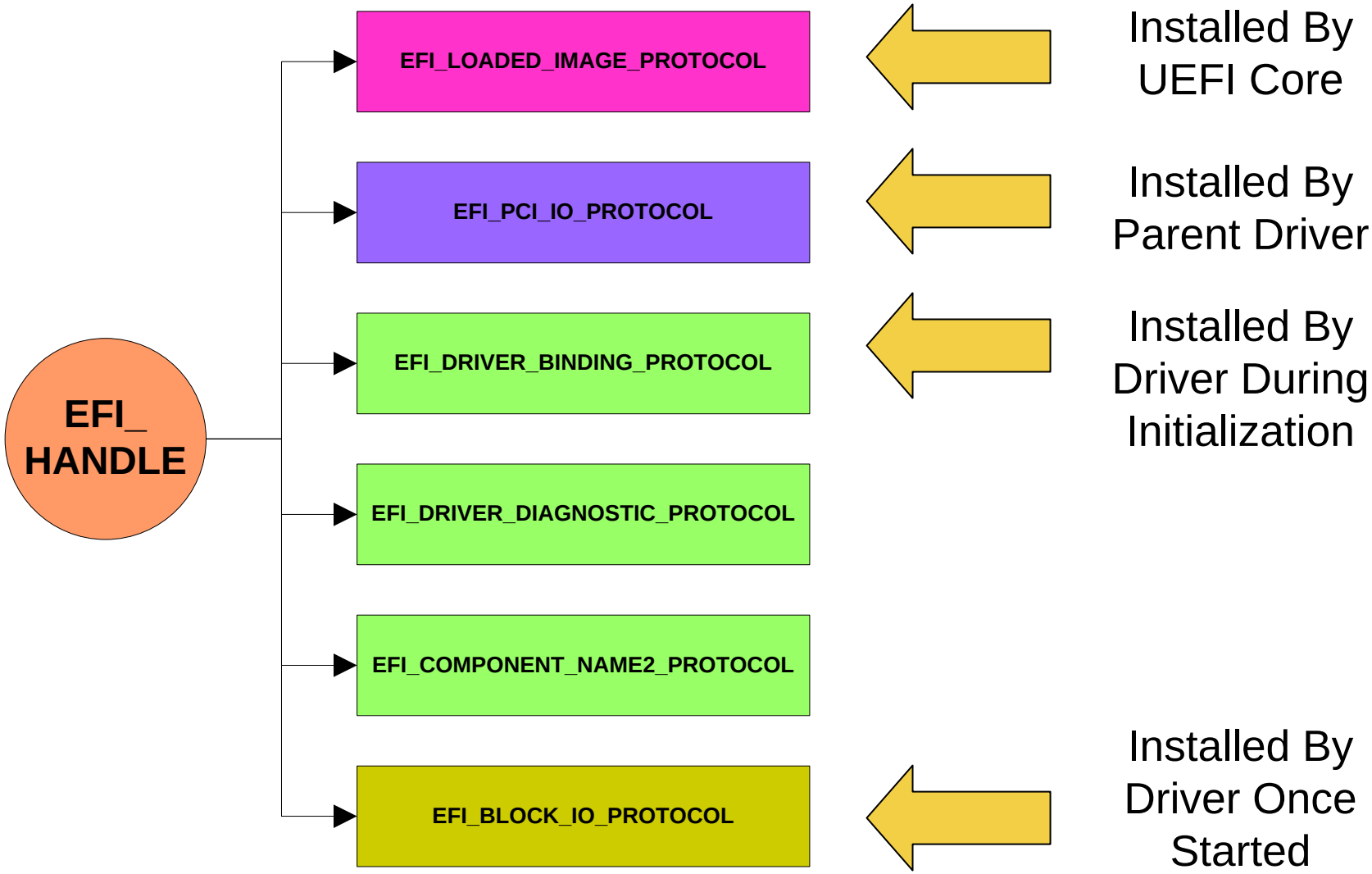
Key Idea #3: Drivers Communicate Using Protocols

- Protocols consist of an interface data structure and a GUID
  - If you know the GUID, you know the data structure
  - Interface data structures contain data or function pointers
- Drivers can *produce* an interface
  - Using **`InstallProtocolInterface()`**
  - Maximum: one of each type per handle
- Drivers can *consume* an interface
  - Using **`HandleProtocol()`** or **`OpenProtocol()`**
- Drivers can be *notified* when a new interface is produced
  - Using **`RegisterProtocolNotify()`**

Key Idea #4: Protocols Are Installed On Handles

- UEFI supports an object model with runtime binding
- Each object is called a handle (**`EFI_HANDLE`**)
- Each handle can have up to one of each interface
- Handles can be created in two ways:
  - **`LoadImage()`** creates a new handle for the image and installs **`EFI_LOADED_IMAGE_PROTOCOL`** on it.
    - Called an *image handle*
  - **`InstallProtocolInterface()`** creates a new handle if it is passed a NULL handle as an input parameter.
- Protocols can be installed on a handle by the UEFI core, a parent driver or by the driver itself.

Where Do Protocols Come From?



Key Idea #5: Device Paths Match Hardware With Handles

- *Device paths* are variable-length binary data structures which describe how to get to a device (software perspective)
- Consists of one or more *device nodes*
  - Hardware – PCI, Memory Mapped
  - ACPI – HID/CID, UID
  - Messaging – ATAPI, SCSI, USB, IPv4, UART
  - Media – Partition, File Path
  - BIOS Boot Specification
- Installed on a handle as (**`EFI_DEVICE_PATH_PROTOCOL`**)
- There are GUIDed device nodes for vendor usage

Device Path Examples

- Example #1: IDE Hard Drive
  - **`/PciRootBridge(0)/Pci(0x1F,1)/Ata(Primary,Master)`**
- Example #2: Legacy Floppy
  - **`/Floppy(0)`**
- Example #3: Partition On IDE Hard Drive
  - **`/PciRootBridge(0)/Pci(0x1F,1)/Ata(Primary,Master)/Mbr(EfiGuid,`*`guid`*`)`**
- Example #4: File On IDE Hard Drive Partition
  - **`/PciRootBridge(0)/Pci(0x1F,1)/Ata(Primary,Master)/Mbr(EfiGuid,`*`guid`*`)/EfiLoader.efi`**
- Example #5: iSCSI
  - **`/PciRootBridge(0)/PCI(2,0)/MAC(…)/IPv4(…)/iSCSI(`*`iSCSITargetName,PortalGroupTag,LUN`*`)`**

Summary

- UEFI Forum Controls The UEFI Specification
- There Are 5 Key Ideas For Understanding UEFI
  - EXEs
  - GUIDs
  - Protocols
  - Handles
  - Device Paths
- UEFI Driver Model Creates Robust, Re-usable Drivers

Where To Find More Information…

- Phoenix's BIOS Developer Blog
  - http://blogs.phoenix.com
- UEFI Site
  - www.uefi.org