Extensible Firmware Interface:

booting the new generation of Intel Architecture platforms

Mark Doran
Program Manager
Intel Corporation

September 1, 1999

intel®

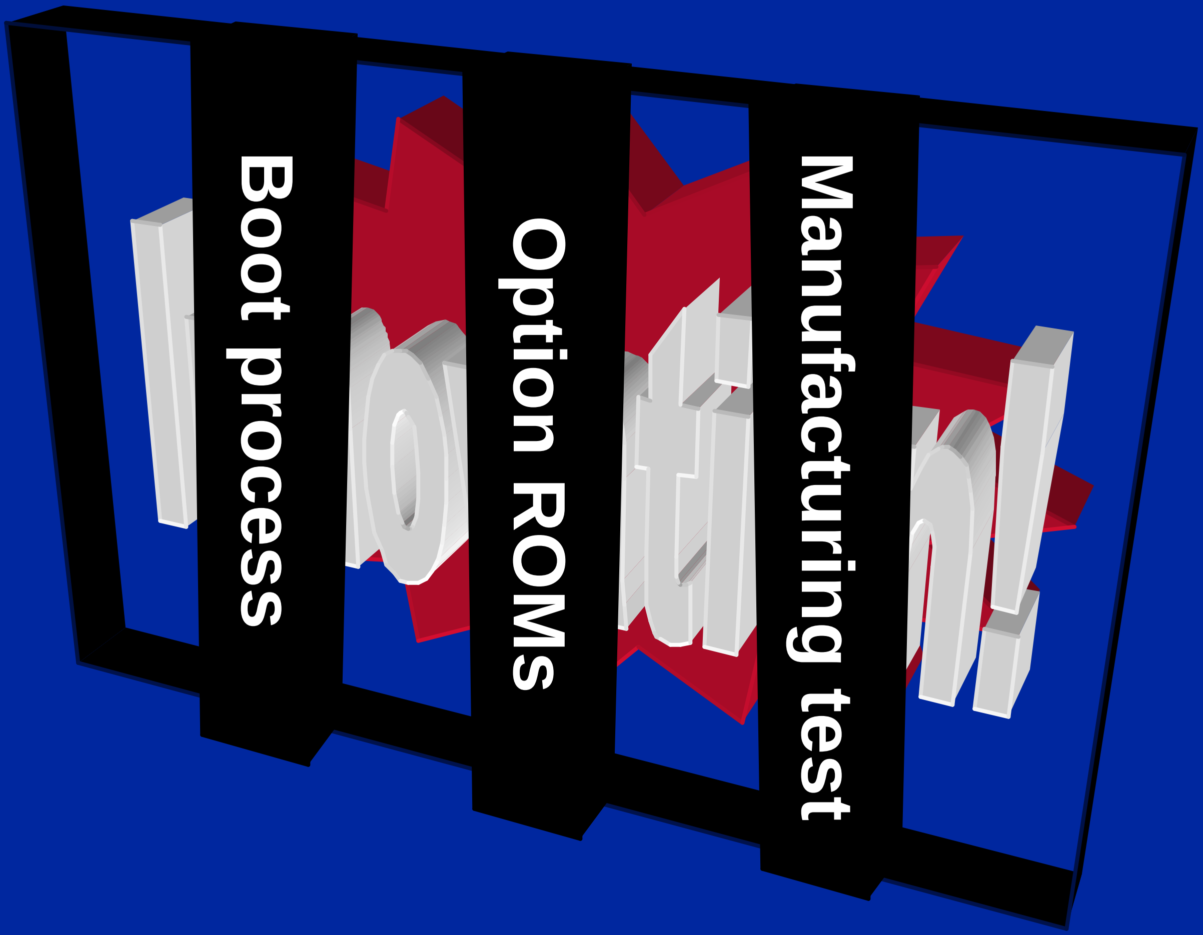
Intel Labs

Agenda

- Why change?
- What is EFI?
- EFI enabling
- EFI sample implementation demo
- Windows NT and EFI – Microsoft
- Implementing EFI – Phoenix Technologies
- Summary

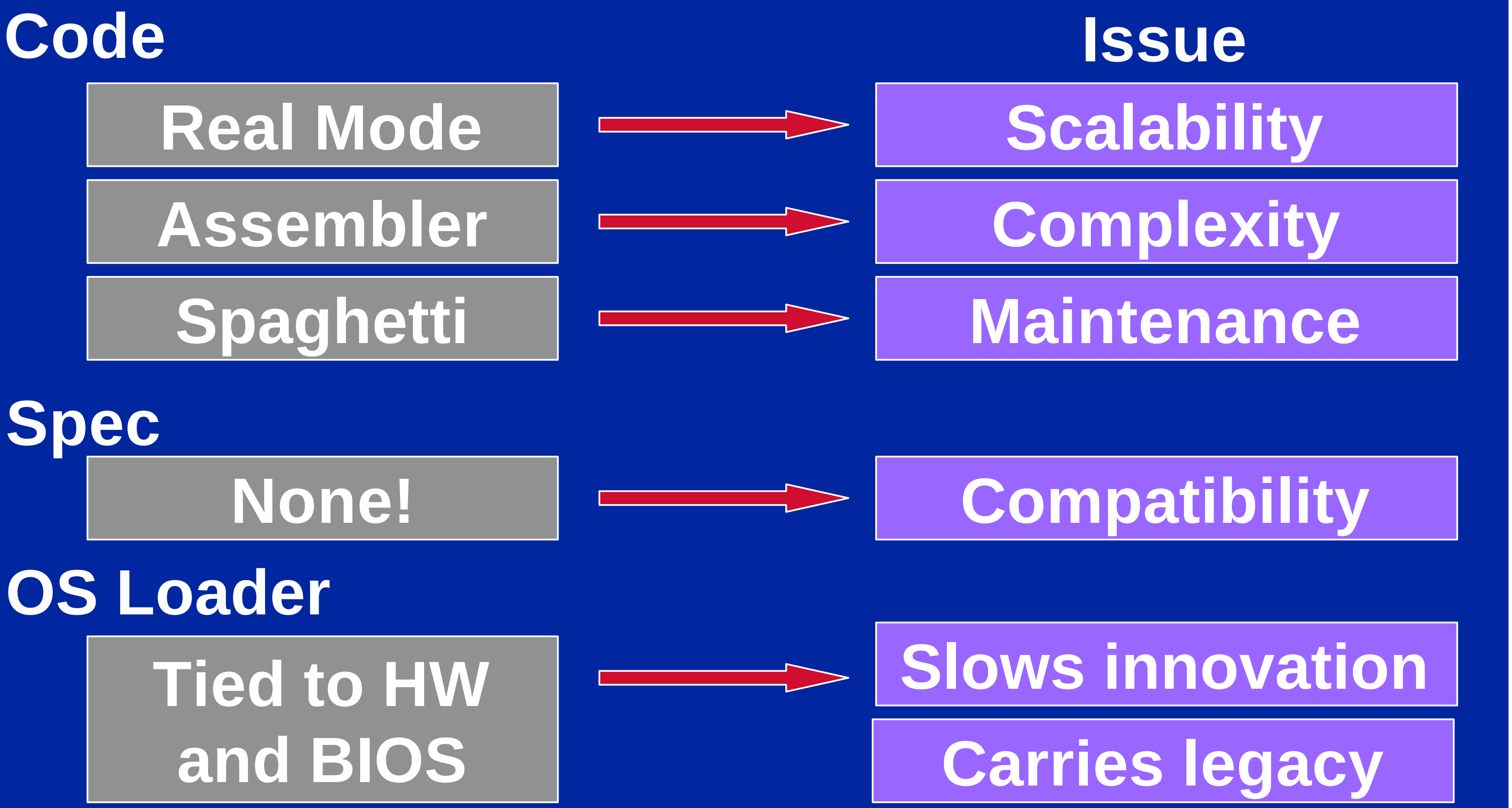
Why Change?

The pre-boot dilemma



Why Change?

Issues with existing boot



Code

Issue

Real Mode → Scalability

Assembler → Complexity

Spaghetti → Maintenance

Spec

None! → Compatibility

OS Loader

Tied to HW and BIOS → Slows innovation

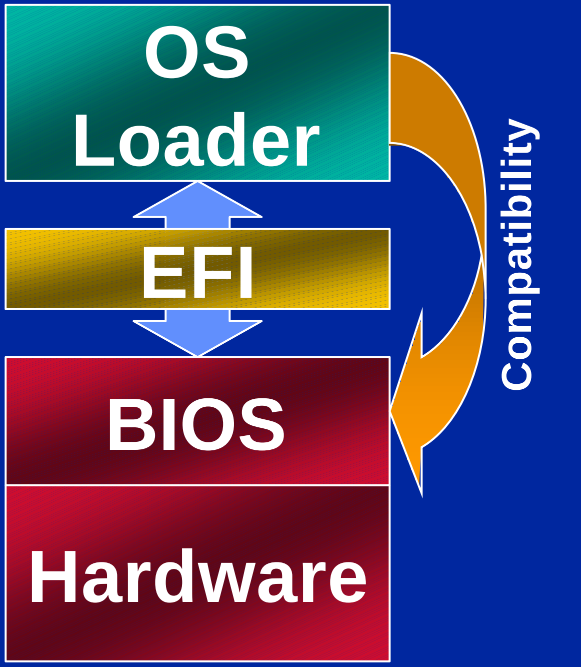
Carries legacy

New Architecture Required

Why Change?

EFI Overview

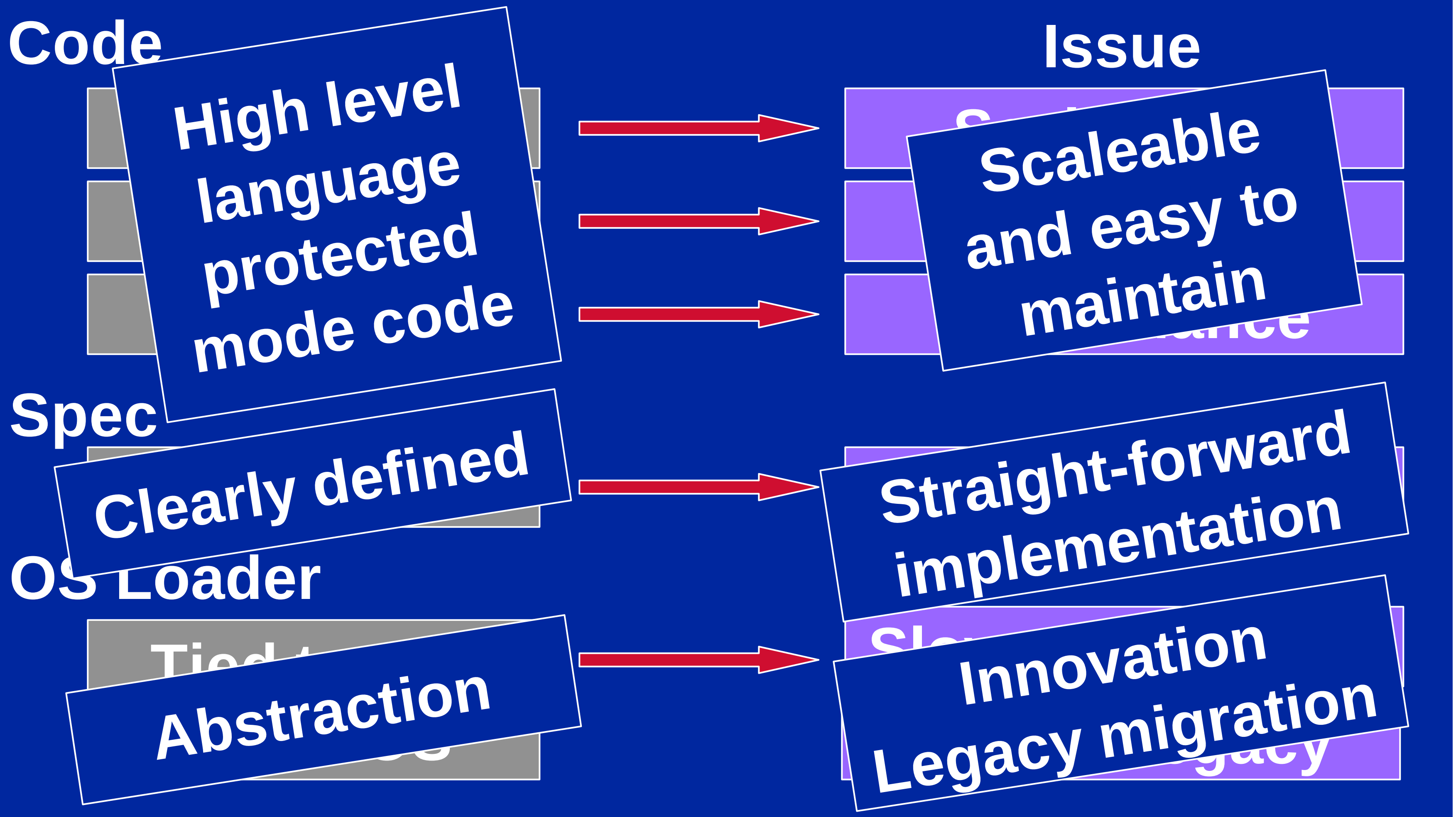
- Interface specification
  - ◆Implementation agnostic
- Abstracts BIOS from OS
  - ◆Decouples development
- Compatible by design
  - ◆Evolution, not revolution
- Modular and extensible
  - ◆OS-Neutral value add
- Complements existing interfaces



Flexible to meet existing and future needs

Why Change?

EFI delivers…..



Code

Issue

High level language protected mode code

Scaleable and easy to maintain

Spec

Clearly defined

Straight-forward implementation

OS Loader

Abstraction

Innovation Legacy migration

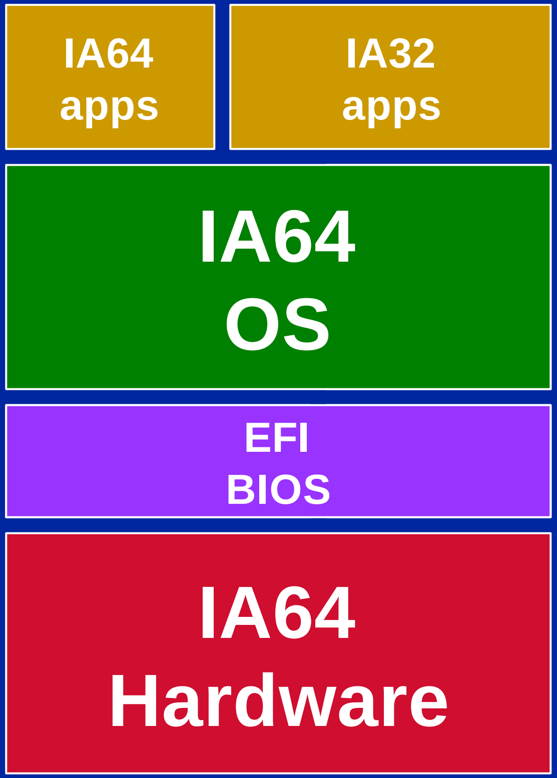
The right solution

Intel Labs

Why Change?

Timing

- IA64 intercept
  - Golden opportunity
    - New operating systems
    - New hardware platform
- Downstream benefits for IA64
  - Legacy migration
  - Scaleability
  - Extensibility
    - Security
    - Manageability
    - Diagnostics



IA64/EFI : the perfect match

intel®

Intel Labs

Why Change?

Breaking away

EFI enables Innovation!

Golden opportunity for change

Agenda

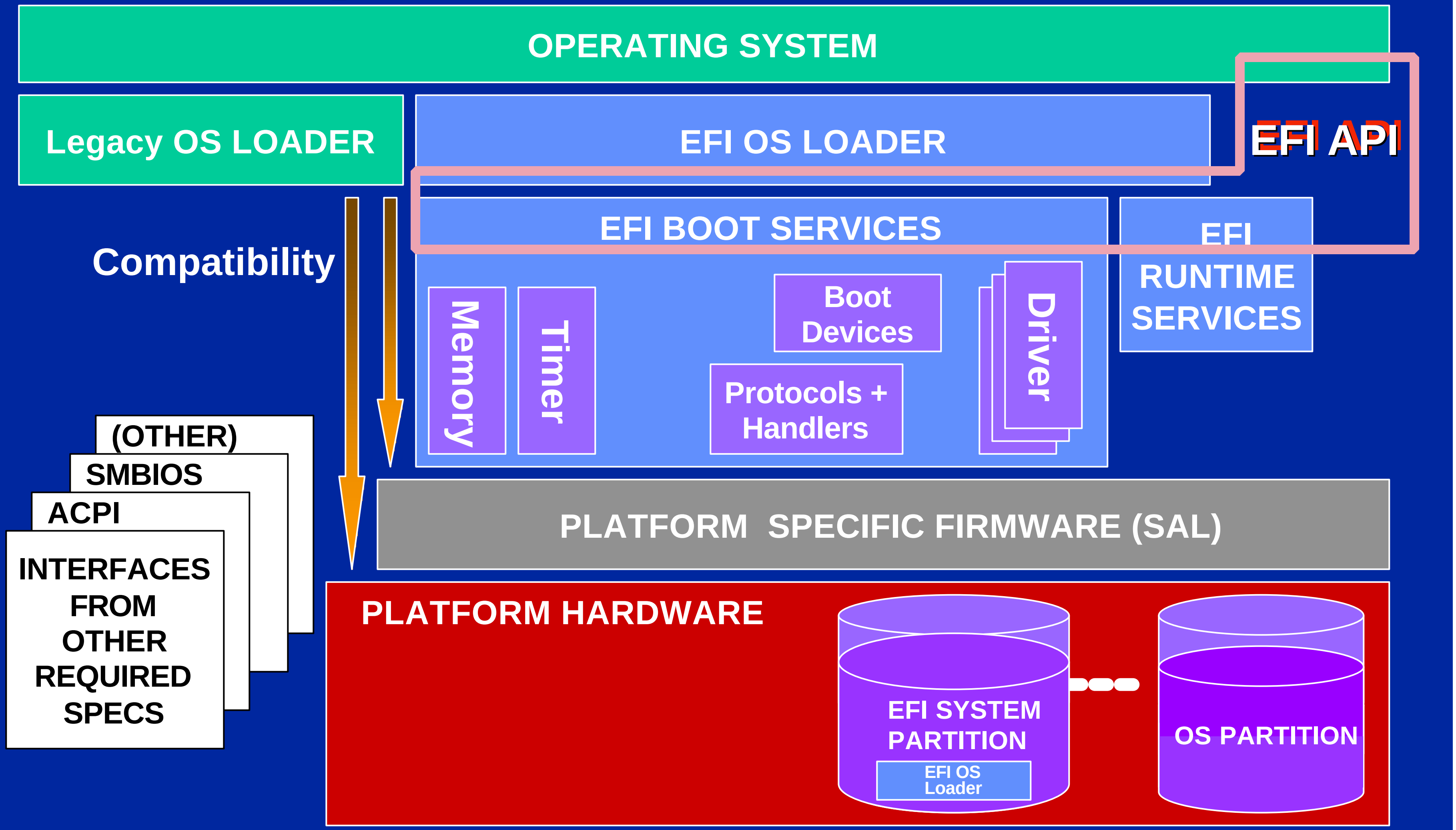
- Why change?

➔ What is EFI?

- Benefits
- Implementation

What is EFI?

Concept



OPERATING SYSTEM

Legacy OS LOADER

EFI OS LOADER

EFI API

Compatibility

EFI BOOT SERVICES

EFI RUNTIME SERVICES

Memory

Timer

Boot Devices

Protocols + Handlers

Driver

(OTHER)

SMBIOS

ACPI

INTERFACES FROM OTHER REQUIRED SPECS

PLATFORM SPECIFIC FIRMWARE (SAL)

PLATFORM HARDWARE

EFI SYSTEM PARTITION

EFI OS Loader

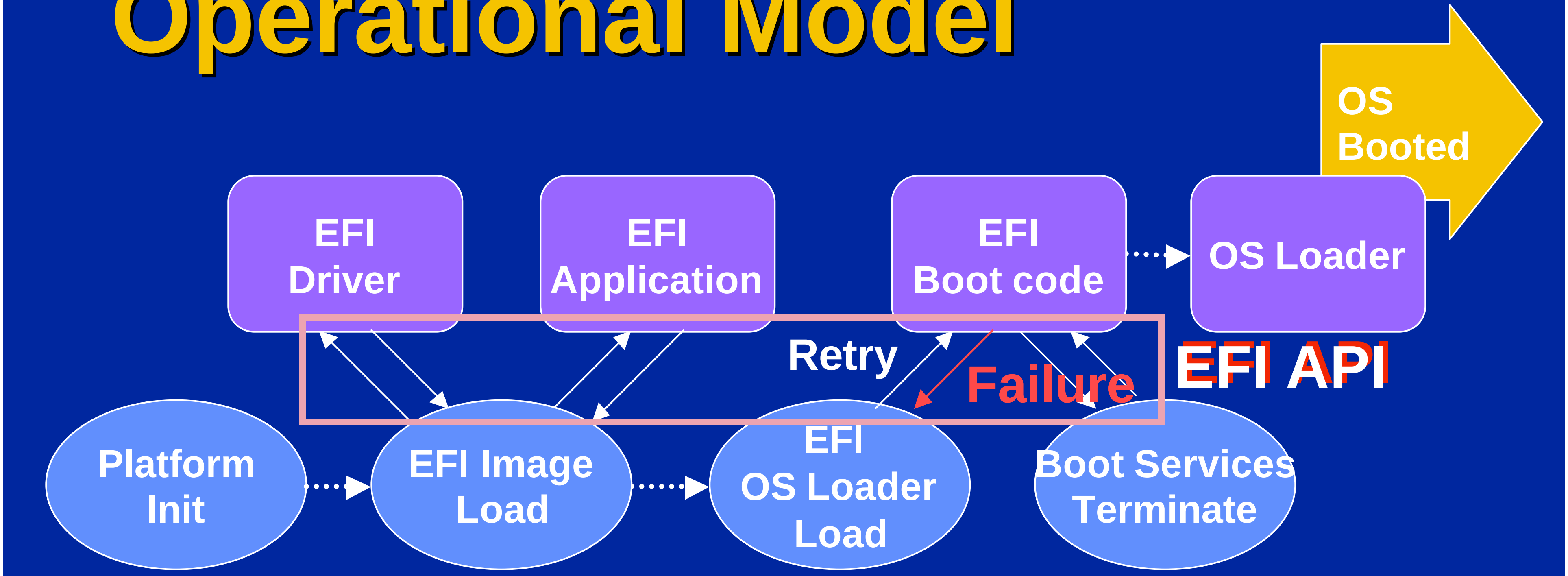
OS PARTITION

intel®

Intel Labs

What is EFI?

Operational Model



OS Booted

EFI Driver

EFI Application

EFI Boot code

OS Loader

Retry

Failure

EFI API

Platform Init

EFI Image Load

EFI OS Loader Load

Boot Services Terminate

Standard firmware platform initialization

Drivers and applications loaded iteratively

Boot from ordered list of EFI OS loaders

Operation handed off to OS loader

→ API specified ⇢ Value add implementation

🟦 Boot Manager 🟪 EFI binaries

What is EFI?

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

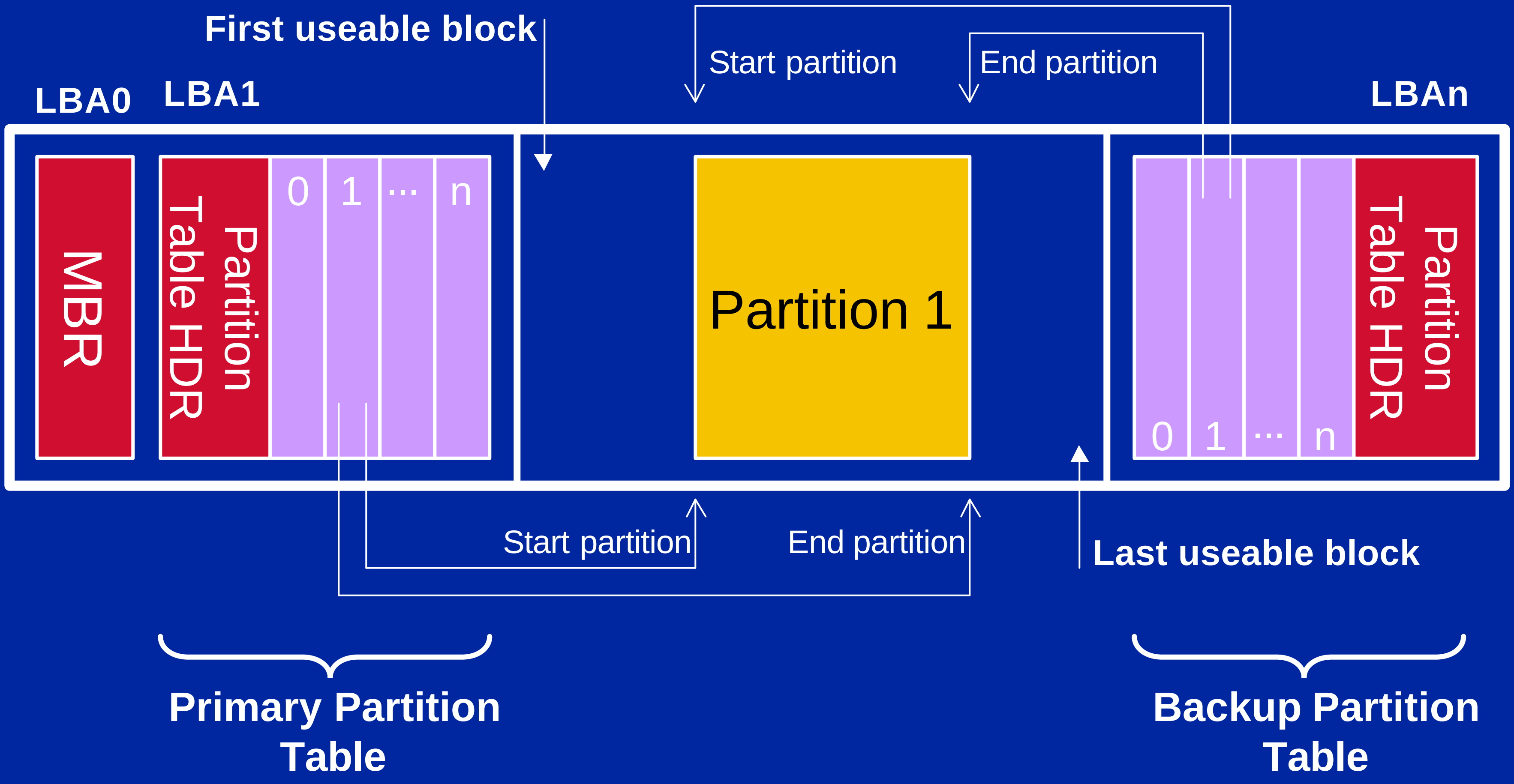
New Partition Structure

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

Designed for flexibility

What is EFI?

New Partition Structure



intel®

Intel Labs

What is EFI?

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

* All trademarks and brands are the property of their respective owners

What is EFI?

Services and Protocols

- Runtime services
- Boot services
- Console services
- Protocols
- GUIDs

What is EFI?: Services and Protocols

Runtime Services

- Boot time and runtime
- Timer, Wakeup alarm
  - Requires processor sync in MP systems
  - Opens path to future legacy migration
- Variables
  - Boot manager handshake
- System reset

Minimal set to meet OSV needs

What is EFI?: Services and Protocols

Boot Services

- Events and notifications
  - polled devices, no interrupts
- Watchdog timer
  - elegant recovery
- Memory allocation
- Handle location
- Image loading
  - drivers, applications, OS loader

Complete, but size efficient

intel®

Intel Labs

What is EFI?: Services and Protocols

Console Services

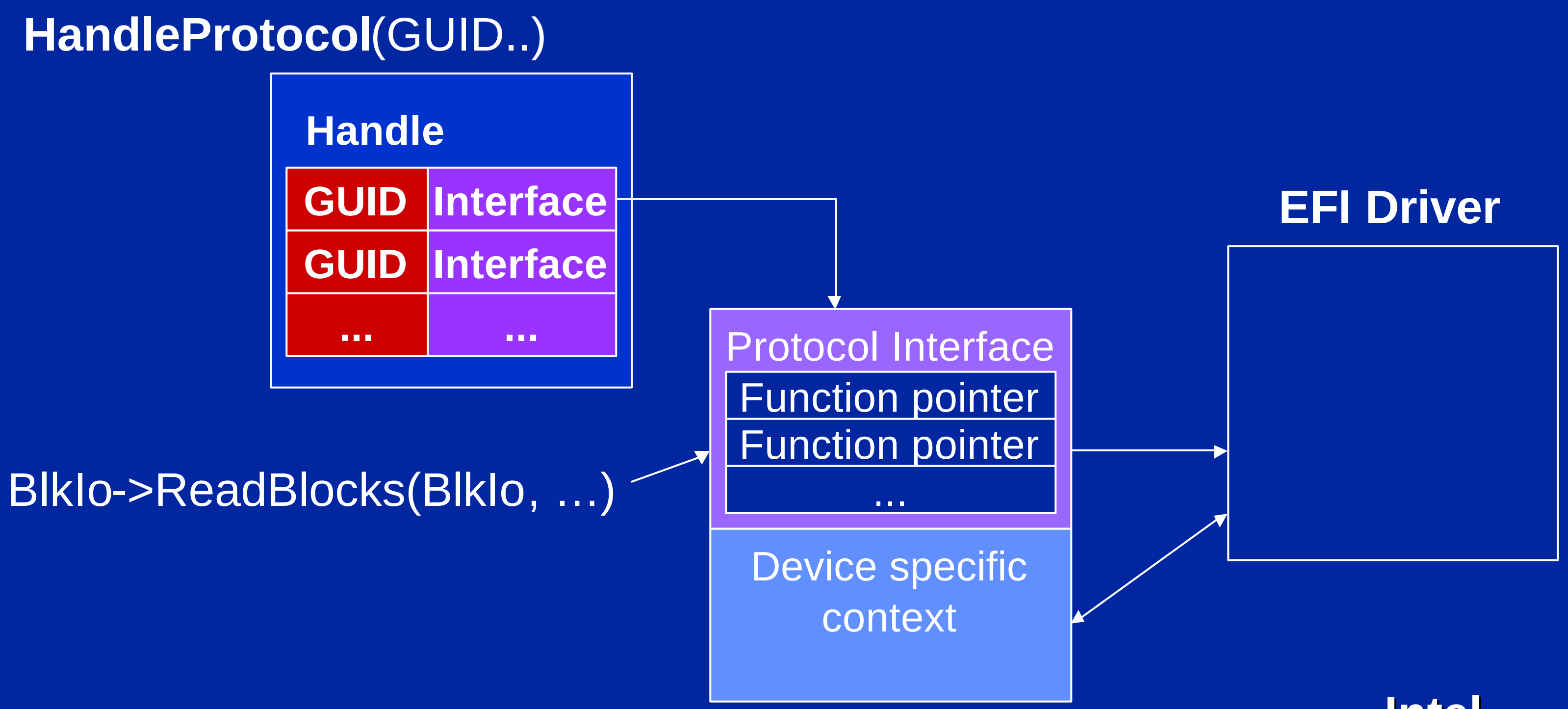
- Abstracted for flexibility
- Support options
  - Local head
    - Character based
    - Graphical (not implemented yet)
  - Remote head
    - Serial link
    - Network

Implementation choices

What is EFI?: Services and Protocols

Protocols

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



HandleProtocol(GUID..)

Handle

GUID	Interface
GUID	Interface
...	...

Protocol Interface

Function pointer

Function pointer

...

Device specific context

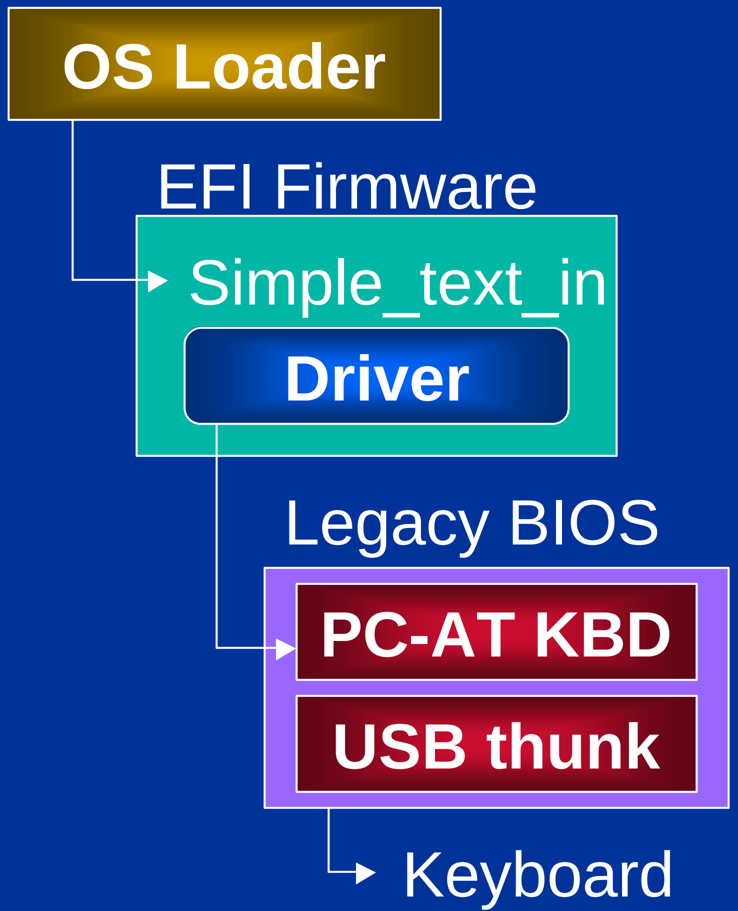
EFI Driver

BlkIo->ReadBlocks(BlkIo, …)

What is EFI?: Services and Protocols

Protocol Example

Initial implementation



OS Loader

EFI Firmware

Simple_text_in

Driver

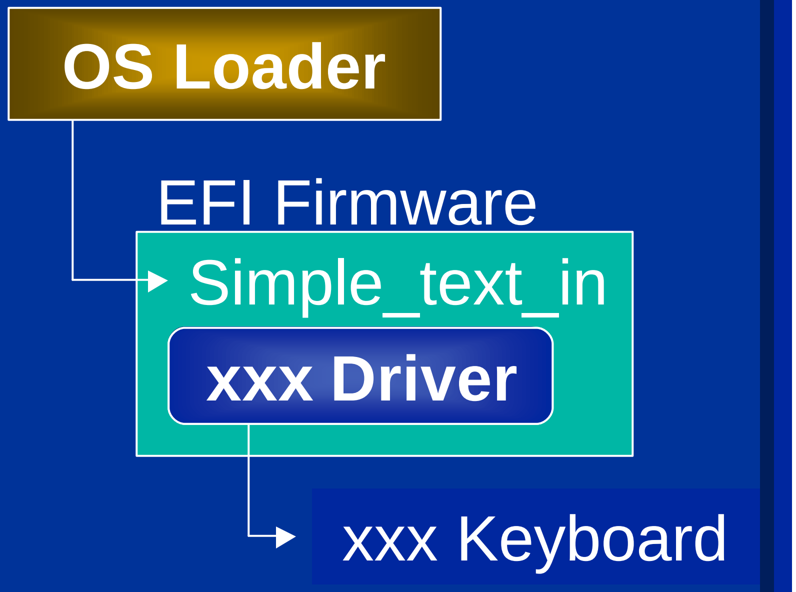
Legacy BIOS

PC-AT KBD

USB thunk

Keyboard

"Legacy Free"



OS Loader

EFI Firmware

Simple_text_in

xxx Driver

xxx Keyboard

Simplified design

What is EFI?

GUID

- "Guaranteed" Unique Identity
  - 128-bit quantity defined by WfM 2.0 spec
- Polices extensibility mechanism
- Allows publishing of new capabilities
  - GUID
  - Interfaces

Safe co-existence of
3rd party extensions

Intel Labs

What is EFI?

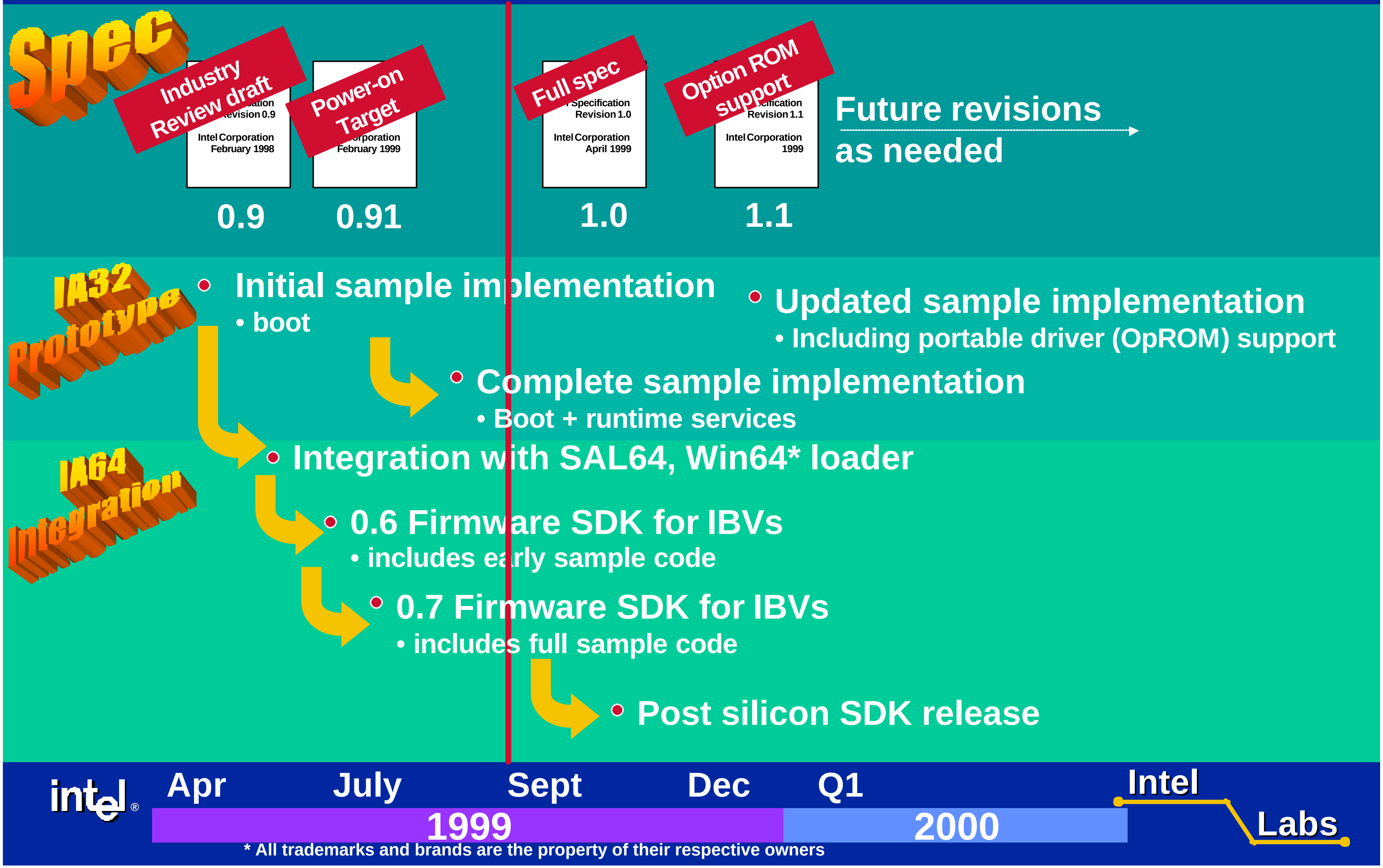
EFI Image Types

- OS Loader
  - EFI application that takes final control
- Application
  - Diagnostics
  - Recovery tools
  - Customer support apps
- Driver
  - Boot support for add-ins
  - Code modules
    - e.g. downloadable workarounds

Differentiation opportunity

Timeline

Roadmap



- Initial sample implementation
  - boot
- Complete sample implementation
  - Boot + runtime services
- Updated sample implementation
  - Including portable driver (OpROM) support
- Integration with SAL64, Win64* loader
- 0.6 Firmware SDK for IBVs
  - includes early sample code
- 0.7 Firmware SDK for IBVs
  - includes full sample code
- Post silicon SDK release

* All trademarks and brands are the property of their respective owners

EFI Enabling

- Industry Intercept on IA-64
  - Intel POR is to use EFI starting at power-on
  - AMI and Phoenix implementing EFI
  - OEMs platforms supporting EFI
  - IA-64 operating systems being developed with EFI
    - IBM/Monterey, Linux, Novell, SCO, Solaris, Windows NT
- IA-32 intercept timing less clear, but:
  - EFI being implemented for embedded systems
  - manufacturing/test infrastructure moving to EFI

Industry momentum

System Design Guides

- EFI is a key component in DIG64
  - enables migration away from legacy
- UNIX Design Guide
  - additional implementation requirements

Foundation for system design

EFI Collaterals

- Complete sample implementation of EFI
  - architecture neutral, IA-32 and IA-64 builds
  - code, build tools and documentation
    - EFI core interface implementation
    - EFI library routines
    - EFI command shell application
    - EFI Developer’s Guide
    - Sample drivers
    - Sample pre-boot applications
- Readily available
  - Simple shrink-wrap license, downloadable code

Low barrier to adoption

Windows and EFI

Pasquale DeMaio

Program Manager

Microsoft

intel®

Intel Labs

Booting the 64 Bit Version of the Windows OS on IA-64

- Boots only via EFI on the IA-64 platform
  - Overall Server Design Guide rules for 64 bit platforms apply http://www.microsoft.com/hwdev/serverdg.htm
  - EFI and ACPI go hand in hand
  - Microsoft contributing specs to the industry
    - EFI FAT32 file system spec
    - PE/COFF image format spec

Software Tools

- Microsoft will be providing disk tools
  - EFI native applications
    - Chkdsk equivalent
    - Format equivalent
    - Fdisk equivalent
  - These tools will be free and you should supply them with your systems

EFI design point

- Keep emergencies in mind while planning your firmware implementations
  - Provide necessary utilities to recover from a disaster
    - Consider remote situation
    - Consider replaced Hard-disk
  - Don’t put critical components on disk

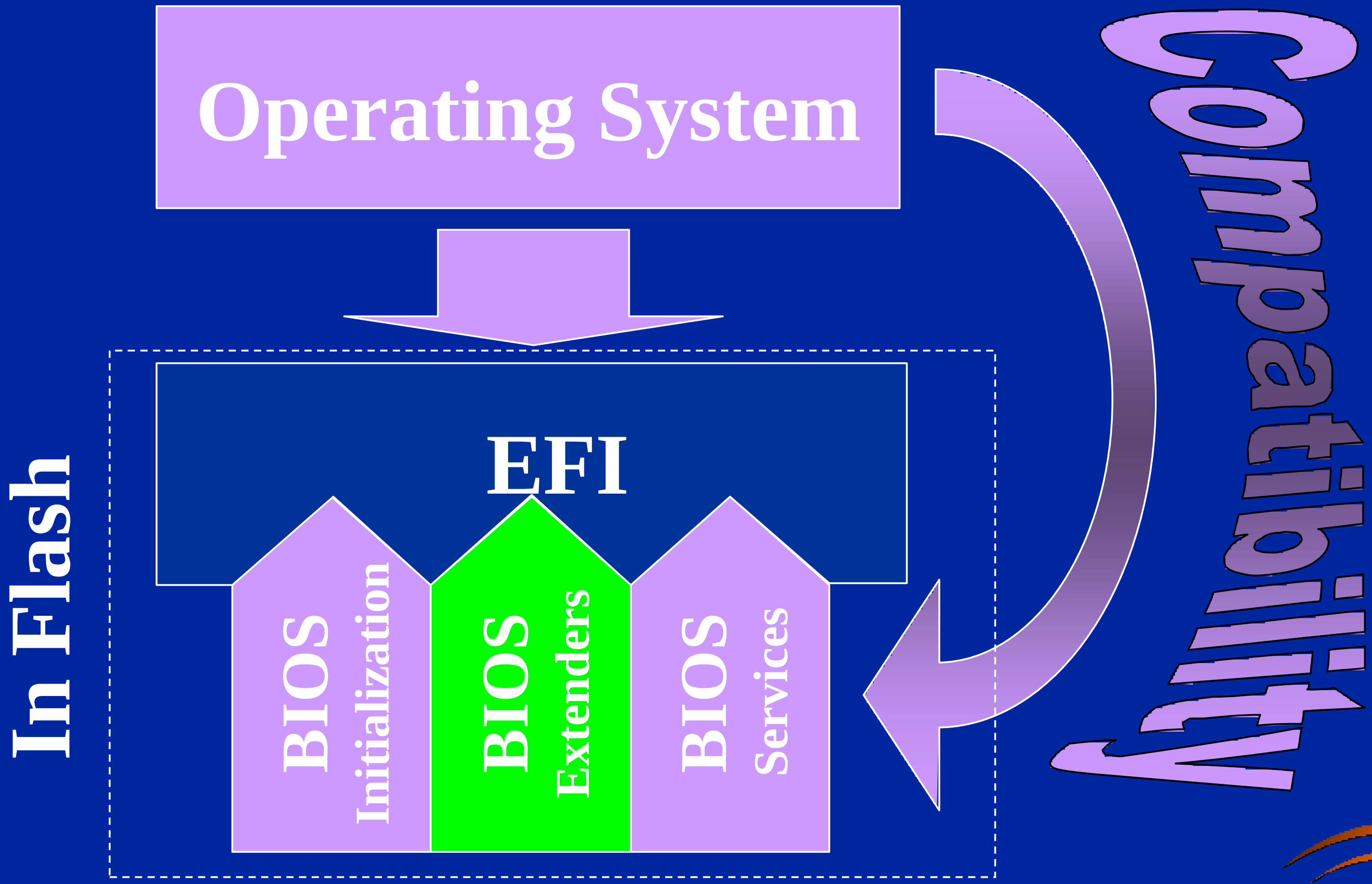
EFI
A BIOS Vendor’s Perspective

Curtis Stevens

Consulting Engineer

Phoenix Technologies

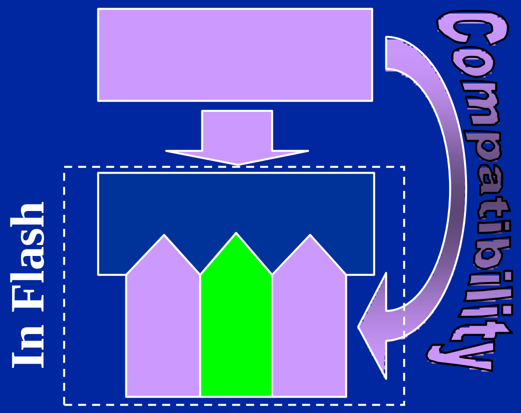
EFI Phase 1



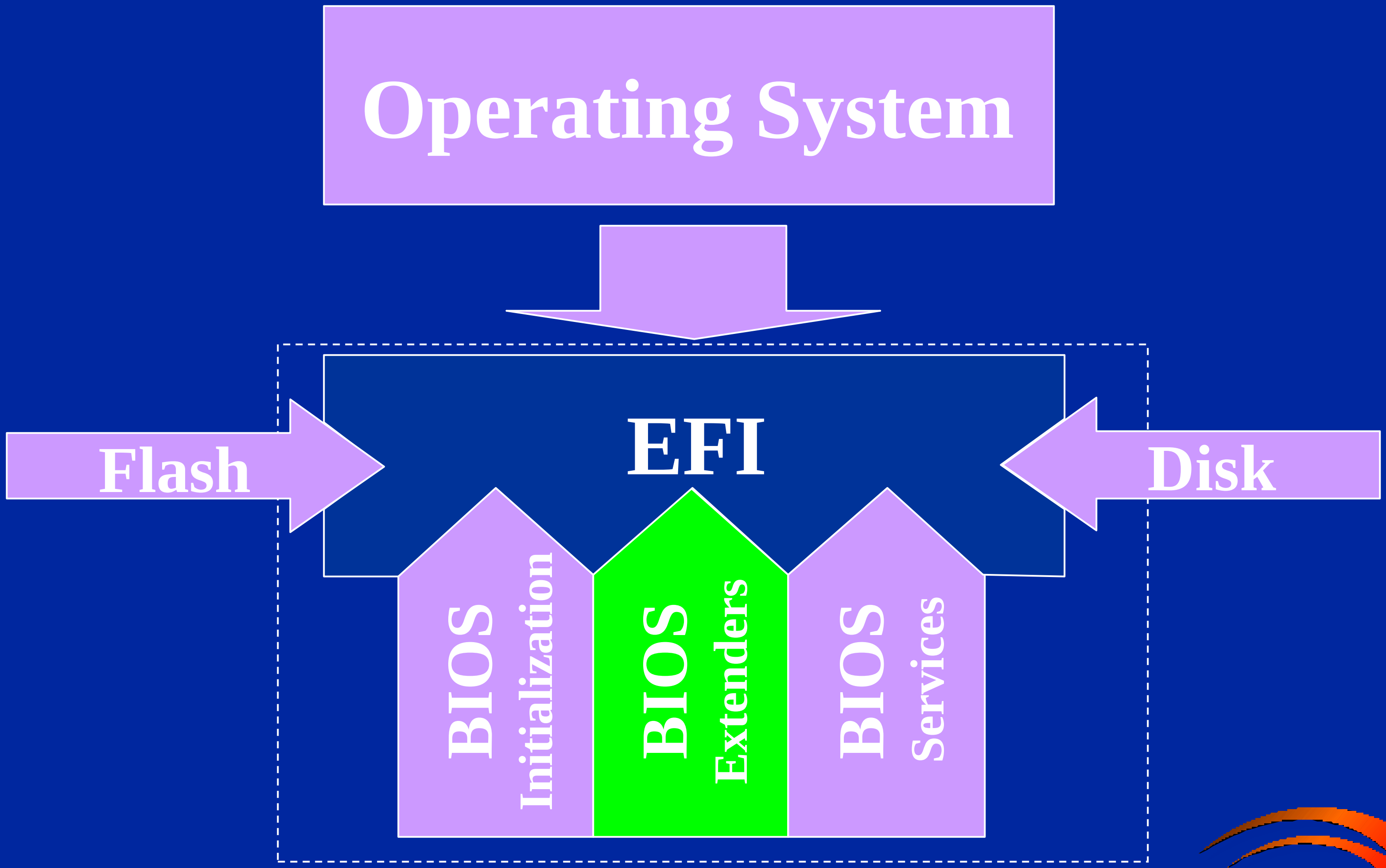
phoenix™ technologies

Building On Existing Firmware

- Peaceful coexistence is key
- Still supports legacy OS
- Enables next generation OS
- Builds on existing specifications
  - ◆EDD 3.0
  - ◆El Torito 2.0
  - ◆ACPI
  - ◆Etc.



Moving Forward



Near Future System Improvements

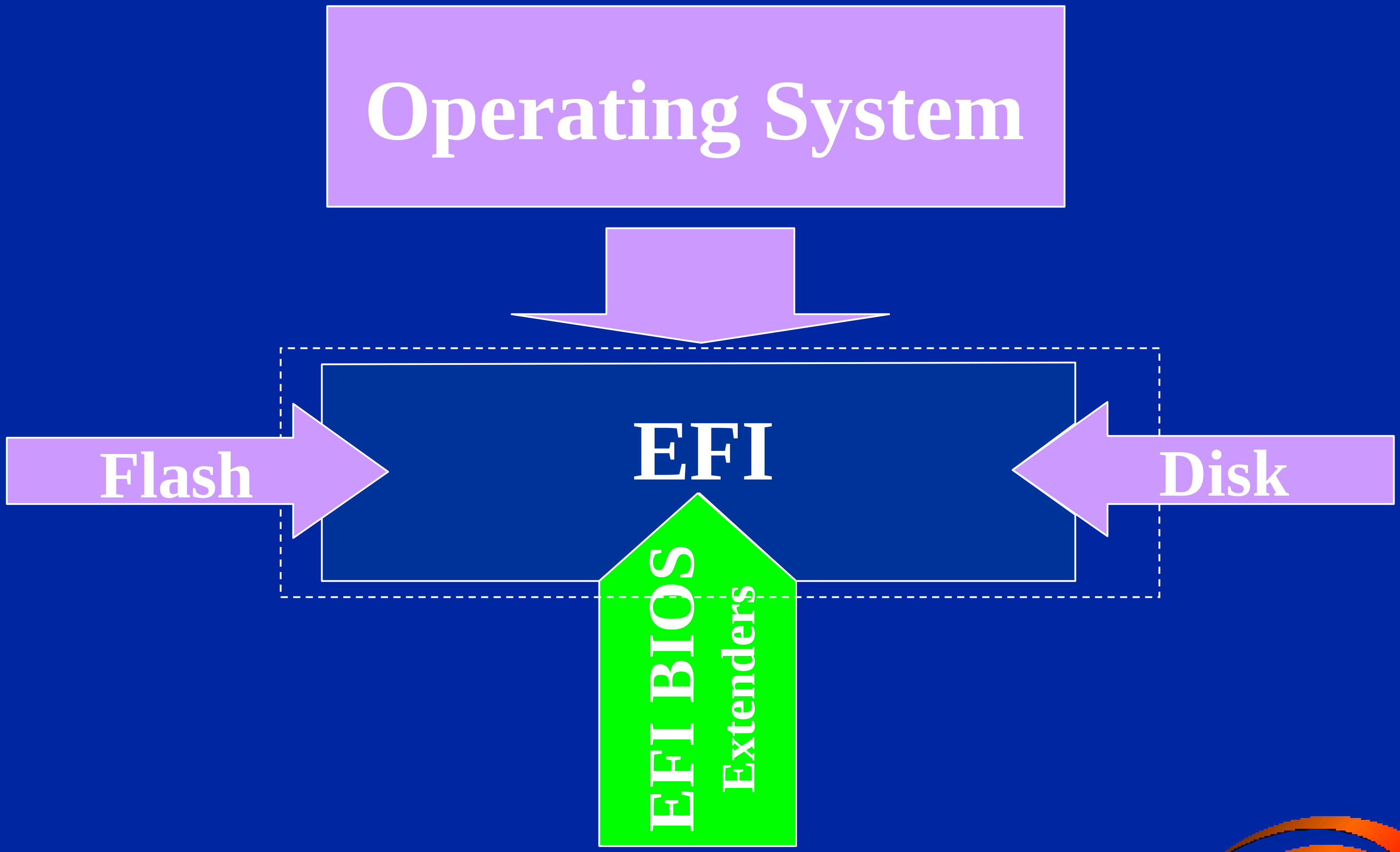
- Pre-OS applications/drivers on disk
  - ◆Non-critical applications & drivers need not take up flash space
  - ◆Well defined environment for system diagnostics
  - ◆Could we see an EFI internet browser?
- Driver updates need not be flashed!
  - ◆Flashing is a dangerous proposition
  - ◆Latest drivers can be placed on the hard drive
    - – The drivers in flash can be disabled

phoenix™ technologies

Future System Improvements

- Option ROMs
  - EFI drivers can now be embedded in a legacy option ROM
  - Once again, peaceful coexistence
  - Work is now being done on a portable option ROM capability
  - Embedded option ROMs do not necessarily need legacy INT structure support
- Boot Disconnect
  - Defines a boot environment shutdown point
  - Positive EFI/BIOS disconnect from all devices

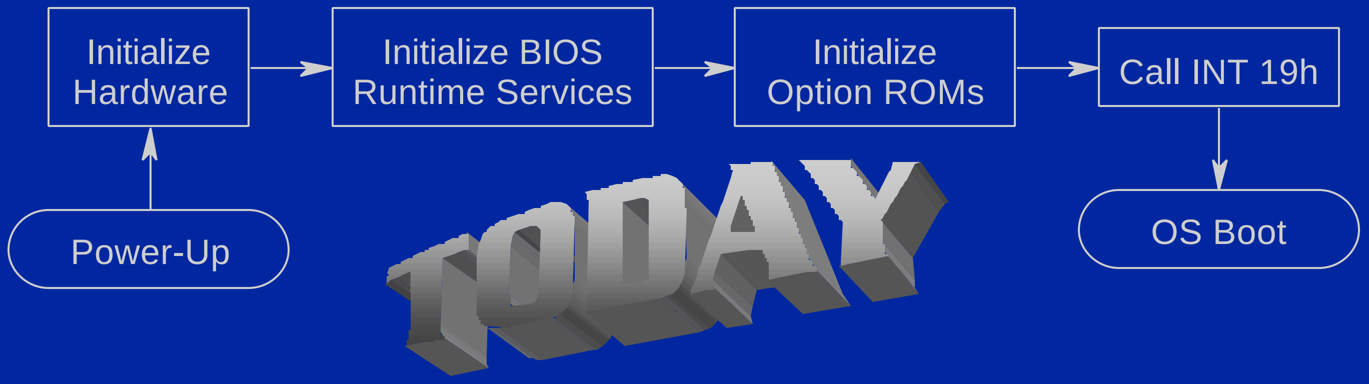
New Possibilities



Dropping Legacy Support

- EFI fully initializes the system using EFI drivers
- Option ROMs drop legacy support
- Drop BIOS constructs like the following
  - Compatibility Region
  - Runtime INT services
  - BIOS Data Area
  - Extended BIOS Data Area

The Boot Process



TODAY

Power-Up

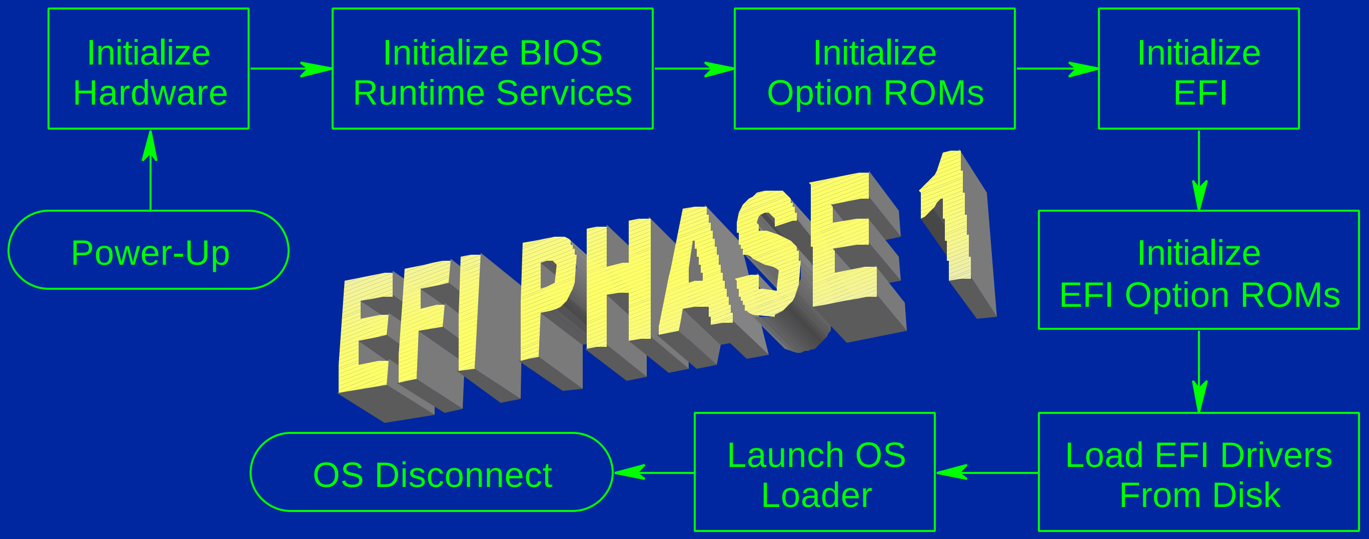
Initialize Hardware

Initialize BIOS Runtime Services

Initialize Option ROMs

Call INT 19h

OS Boot



EFI PHASE 1

Power-Up

Initialize Hardware

Initialize BIOS Runtime Services

Initialize Option ROMs

Initialize EFI

Initialize EFI Option ROMs

Load EFI Drivers From Disk

Launch OS Loader

OS Disconnect

phoenix™ technologies

The Phoenix Experience

With the Intel EFI Sample Code

Intel EFI Code

- Intel EFI Source Code
  - All in C
  - Compiled out of the box
  - Found very few problems
- Used Intel EFI core, drivers, shell, and boot manager
  - Drivers dependent on IA32 INT9, 10, and 13
  - Some SoftSDV limitations
  - More functionality in next version

phoenix™ technologies

Our Implementation

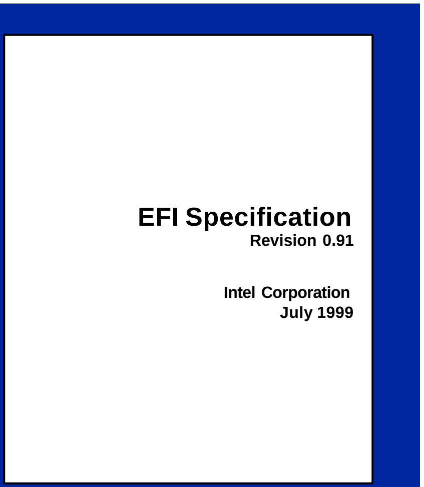
- Phoenix Changes Required
  - New build component - efi.exe
  - Implement PE32+ loader in SAL
  - Call-back procedures to IA32 INTnn
  - Memory Descriptor Entry to EFI memory map entry conversion
  - Replace INT19 with sequence to EFI
  - SAL test procedures now work in EFI environment

phoenix™ technologies

Summary

- Golden opportunity for change
- Flexible solution to meet existing and future needs
- Win, win, win
- Good progress towards industry intercept
- Easy to implement

Call to action



EFI Specification
Revision 0.91

Intel Corporation
July 1999

OEM

OSV

IBV

- Download the spec
  - developer.intel.com
- The only way to boot on IA-64 is with EFI
  - EFI aware operating system loaders
  - EFI conformant platform firmware
  - Pre-boot EFI applications

Intel Labs

EFI on the Web

- EFI Homepage
  - http://developer.intel.com/design/servers/efi/
  - register for EFI mailing list
  - provide feedback on the specification
  - sample implementation and docs
- EFI FAT32 Specification
  - http://www.microsoft.com/hwdev/specs/
- PE/COFF Image Format Specification
  - http://www.microsoft.com/hwdev/specs/