Non-IA Silicon Support within the Intel® Platform Innovation Framework for the Extensible Firmware Interface

Vincent Zimmer
Staff Software Engineer
Intel Corporation

Michael Kinney
Staff Software Engineer
Intel Corporation

Robert Hart
Sr. Principal Engineer
Insyde Software

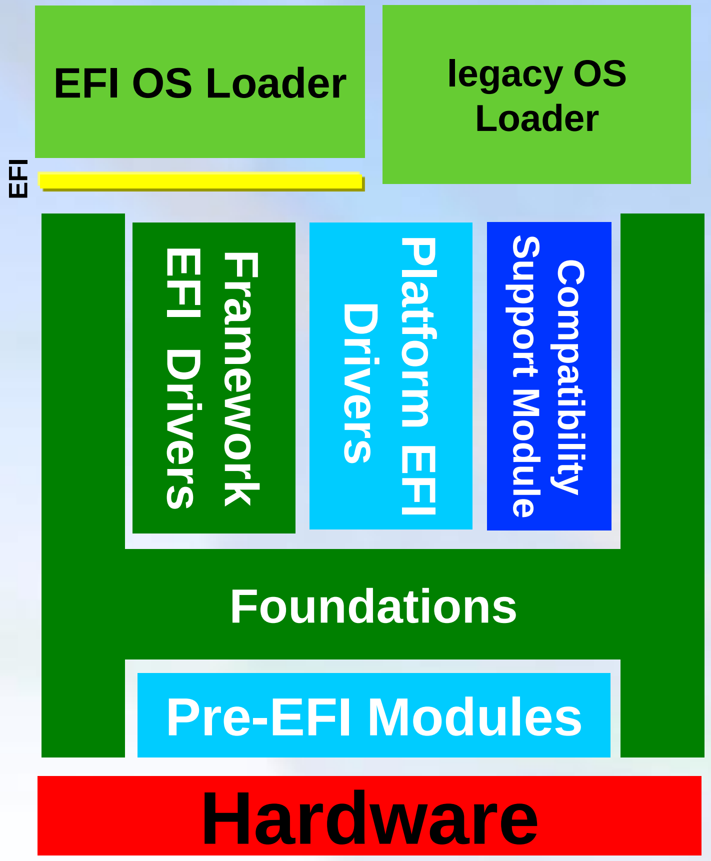
September 18th 2003

Agenda

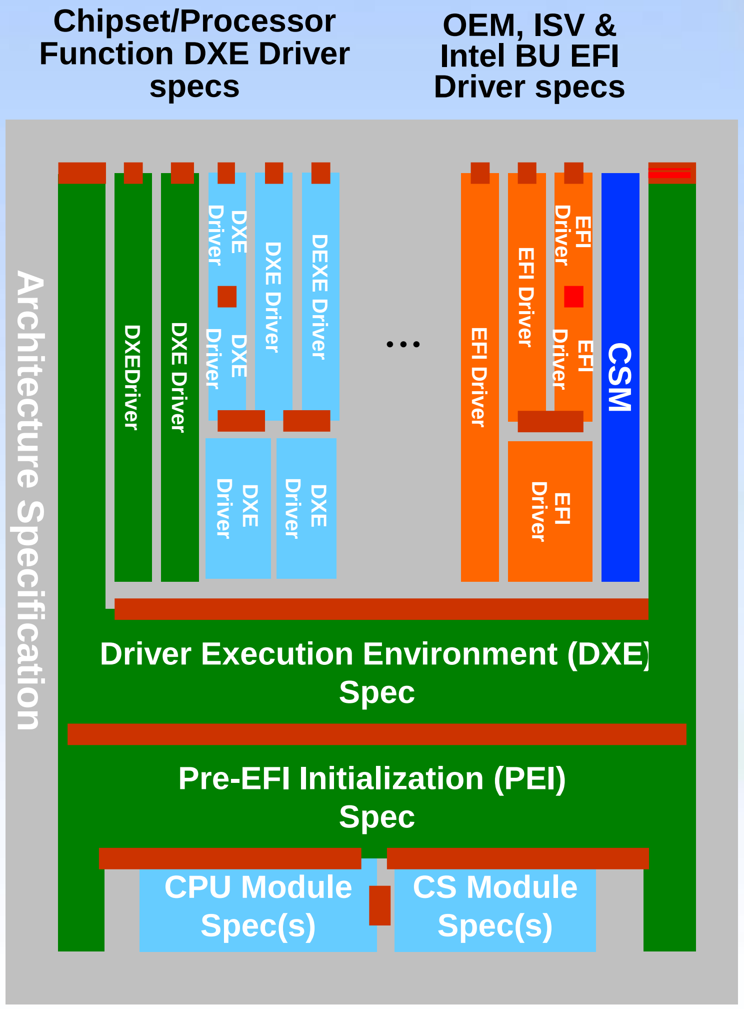
- Framework Overview
- Pre-EFI Initialization Foundation
- Driver Execution Environment Foundation
- Porting the Framework to Non-IA Silicon

What is the Framework?

- Intel Platform Innovation Framework for EFI
- Compatibility Support Module (IA32 only)
- Hardware-specific modules and drivers



Framework Components

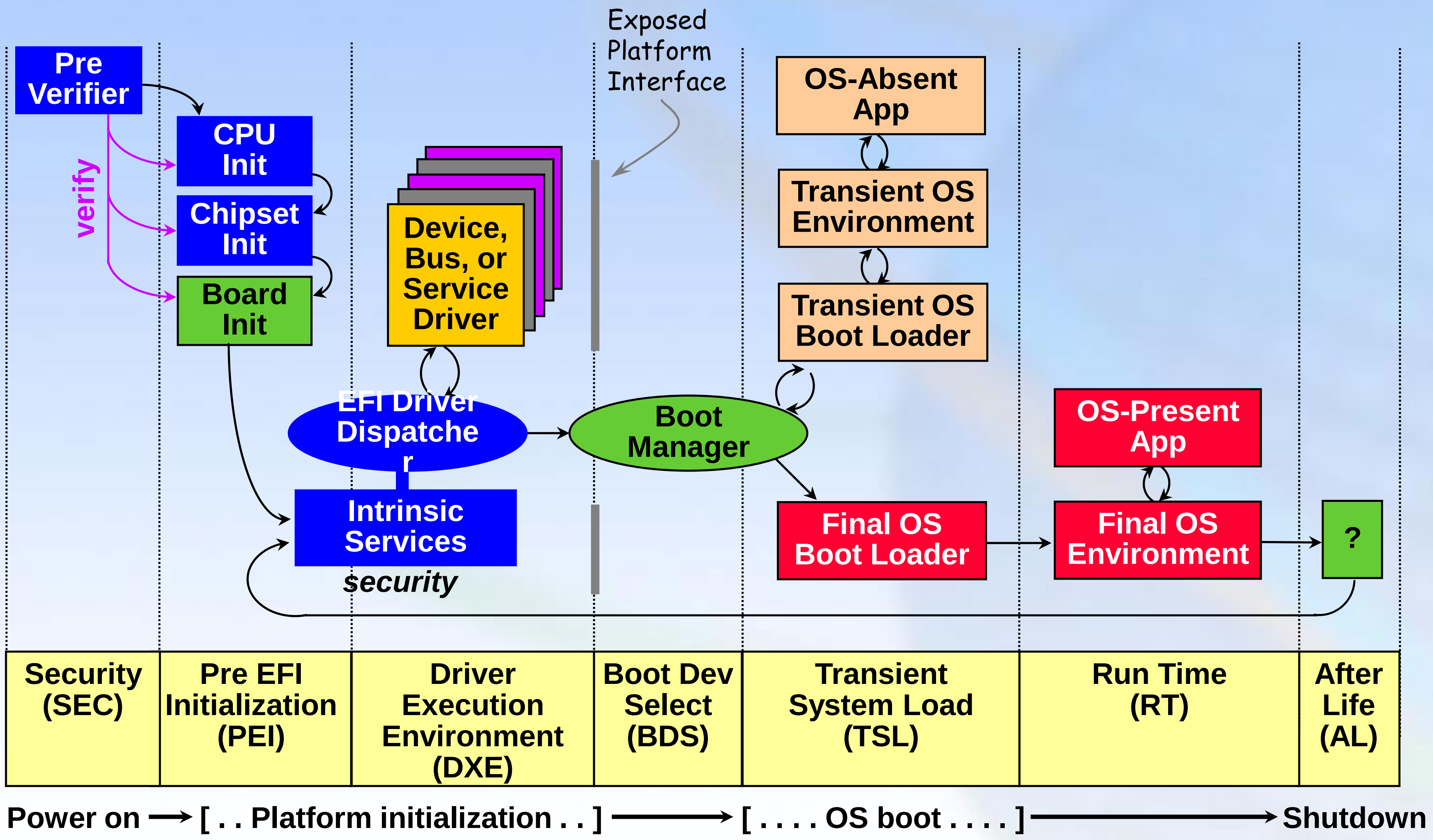

Chipset/Processor Function DXE Driver specs
OEM, ISV & Intel BU EFI Driver specs
Architecture Specification
DXEDriver
DXE Driver
DXE Driver
DXE Driver
DXE Driver
DEXE Driver
DXE Driver
DXE Driver
…
EFI Driver
EFI Driver
EFI Driver
EFI Driver
EFI Driver
CSM
Driver Execution Environment (DXE) Spec
Pre-EFI Initialization (PEI) Spec
CPU Module Spec(s)
CS Module Spec(s)

- PEI is thinnest possible code layer to initialize system
- DXE environment provides generic platform functions to support EFI drivers
- EFI Drivers provide specific platform capabilities and customization
- Interface definitions at multiple levels

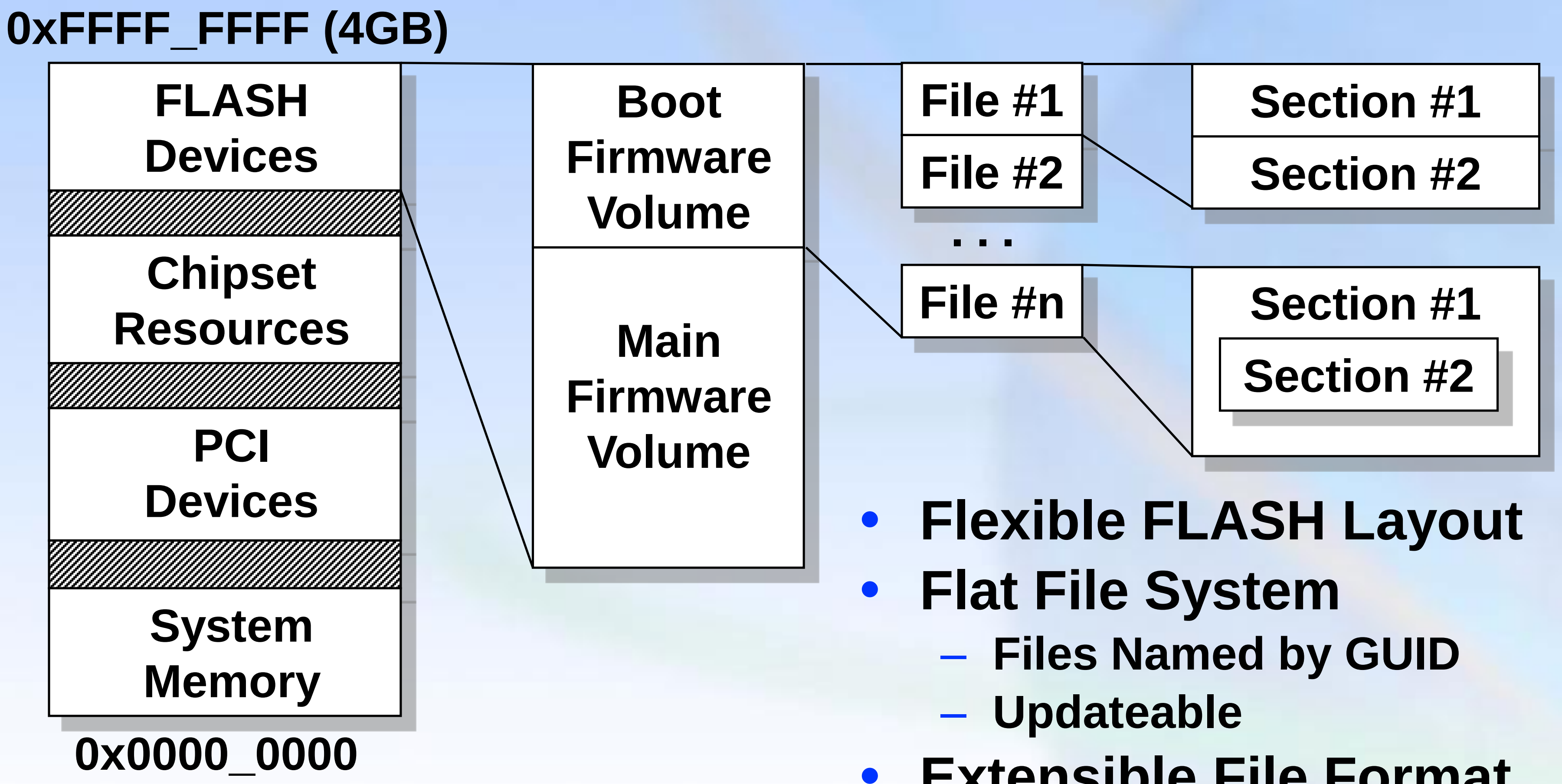
<u>Legend</u>

🟥 API

Framework Operation

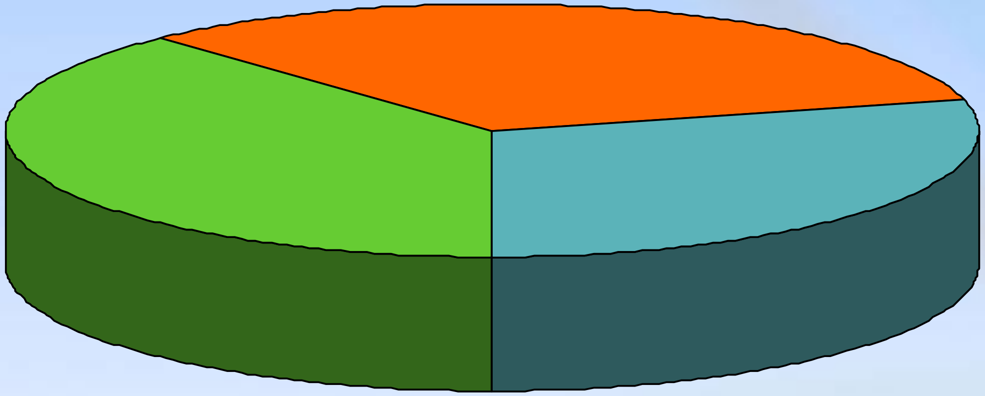


File System for FLASH Devices

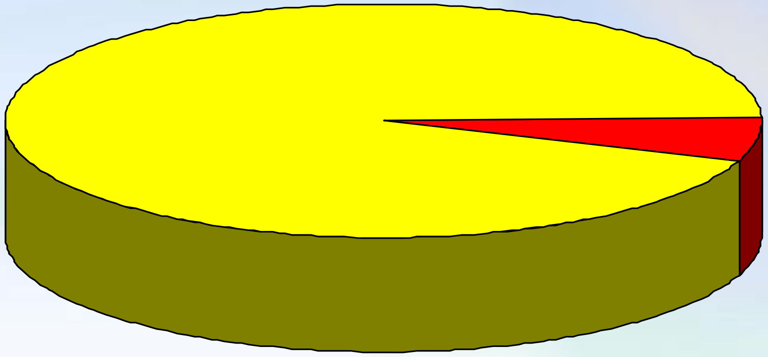


- Flexible FLASH Layout
- Flat File System
  - Files Named by GUID
  - Updateable
- Extensible File Format

Firmware Source Code



- 🟩 Foundation (H)
- 🟧 Common Modules
- 🟦 Platform Specific Modules



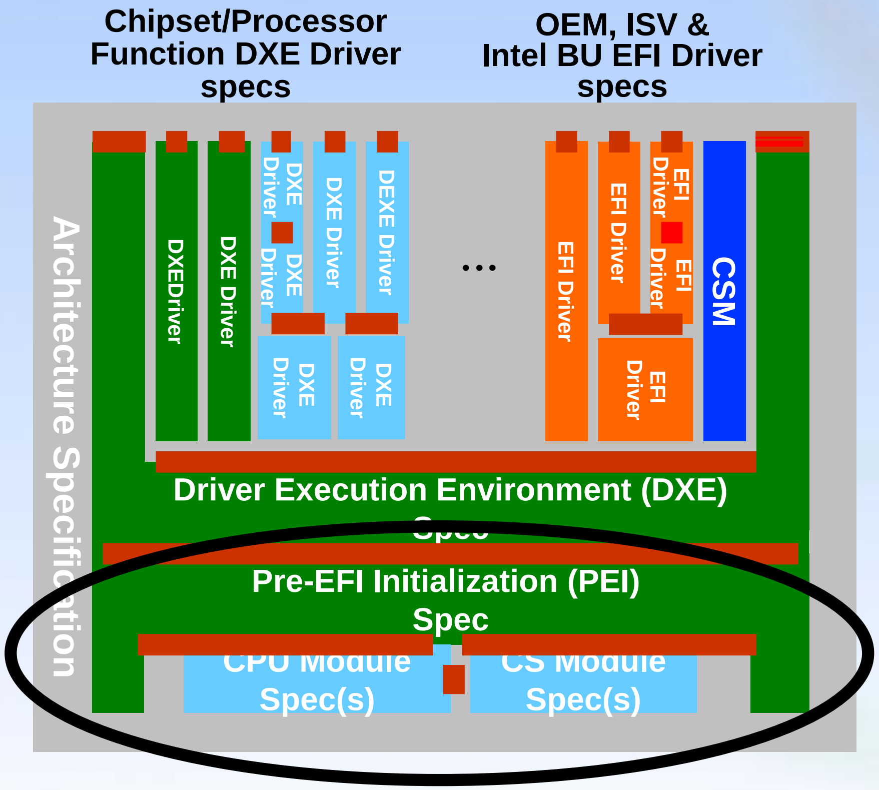
- 🟨 C Code
- 🟥 Assembly

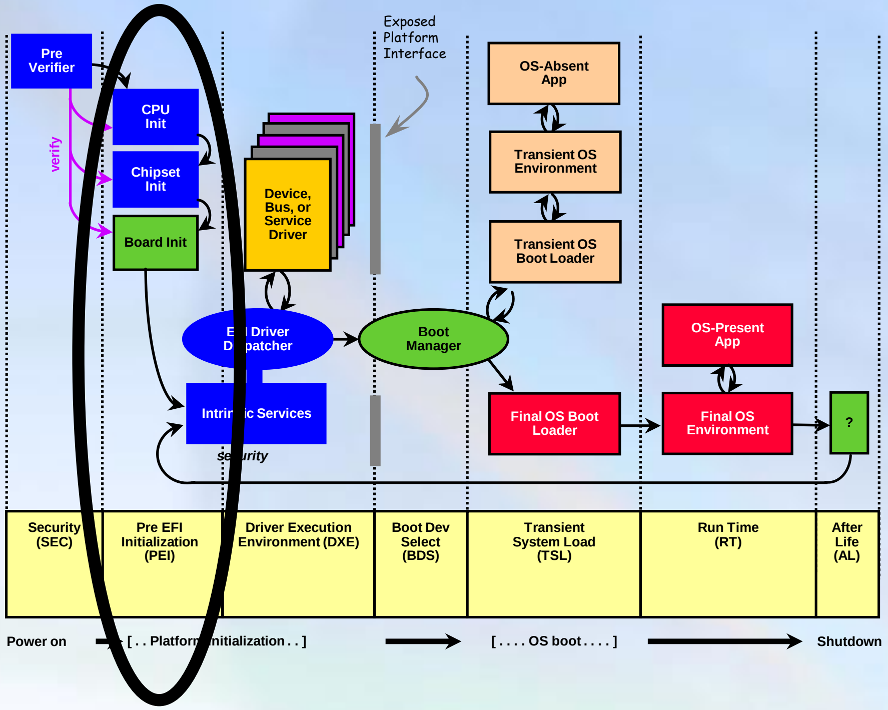
*Data collected from a typical desktop platform

Agenda

- Framework Overview
- **Pre-EFI Initialization Foundation**
- Driver Execution Environment Foundation
- Porting the Framework to Non-IA Silicon

PEI Foundation


Chipset/Processor Function DXE Driver specs
OEM, ISV & Intel BU EFI Driver specs
Architecture Specification
DXEDriver
DXE Driver
DXE Driver
DXE Driver
DXE Driver
DEXE Driver
DXE Driver
DXE Driver
...
EFI Driver
EFI Driver
EFI Driver
EFI Driver
EFI Driver
CSM
Driver Execution Environment (DXE) Spec
Pre-EFI Initialization (PEI) Spec
CPU Module Spec(s)
CS Module Spec(s)

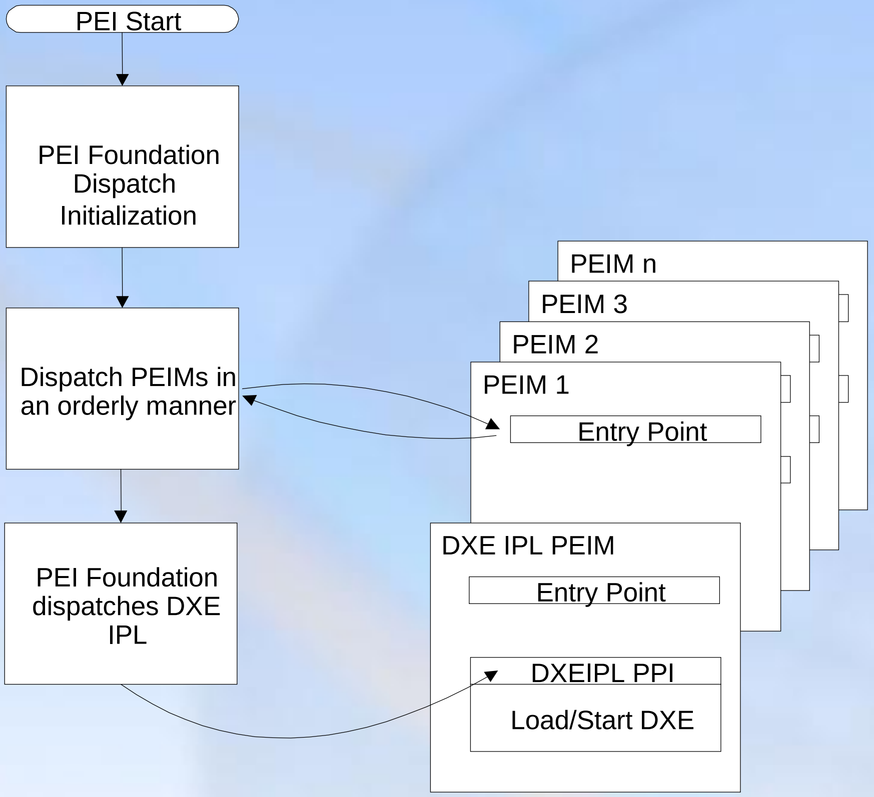


What are the goals of PEI?

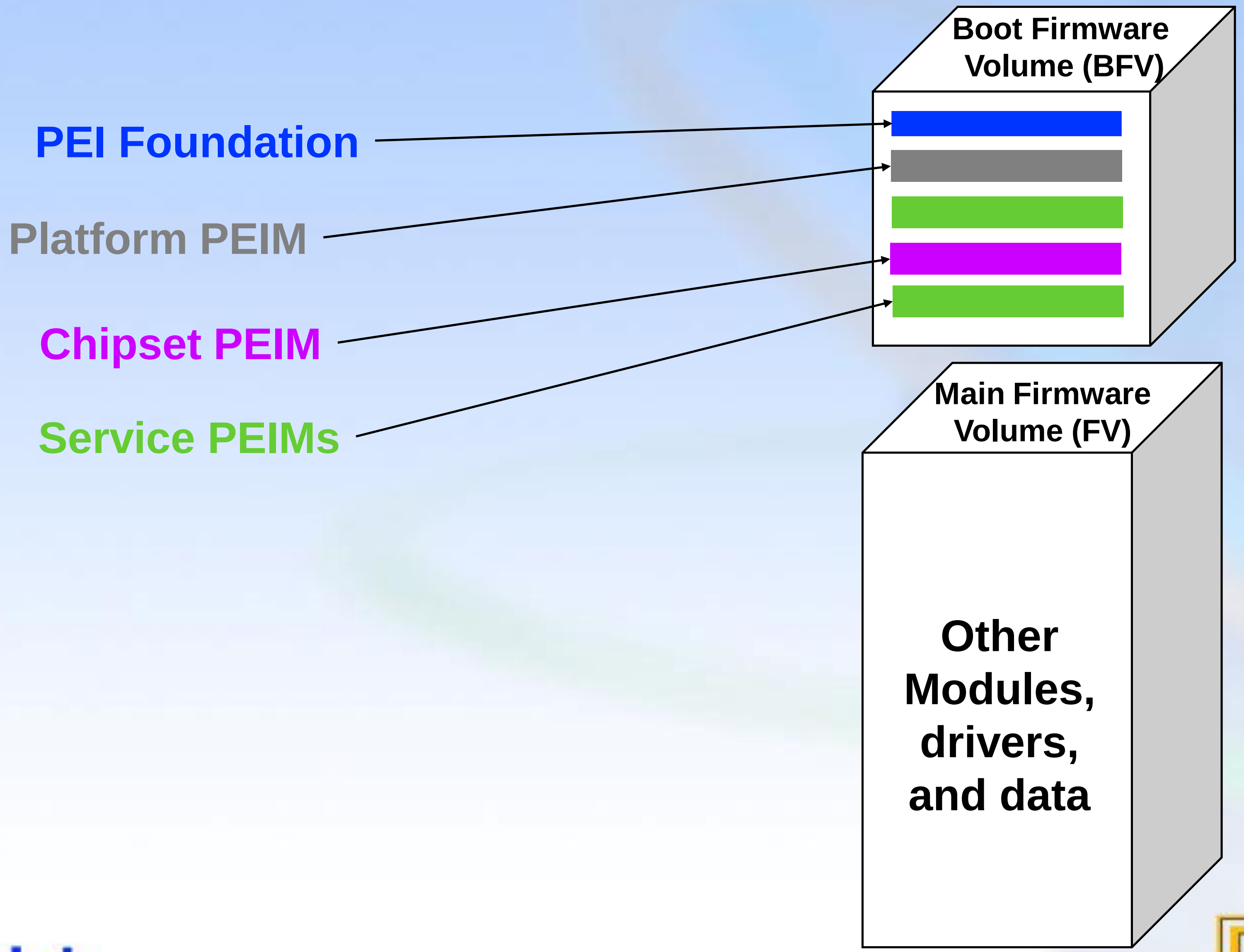
- Discover Boot Mode
- Launch Modules to Initialize Main Memory
- Describe Platform Resources
- Support Various Restarts Events
- Discover & Launch DXE Foundation

What is PEI?

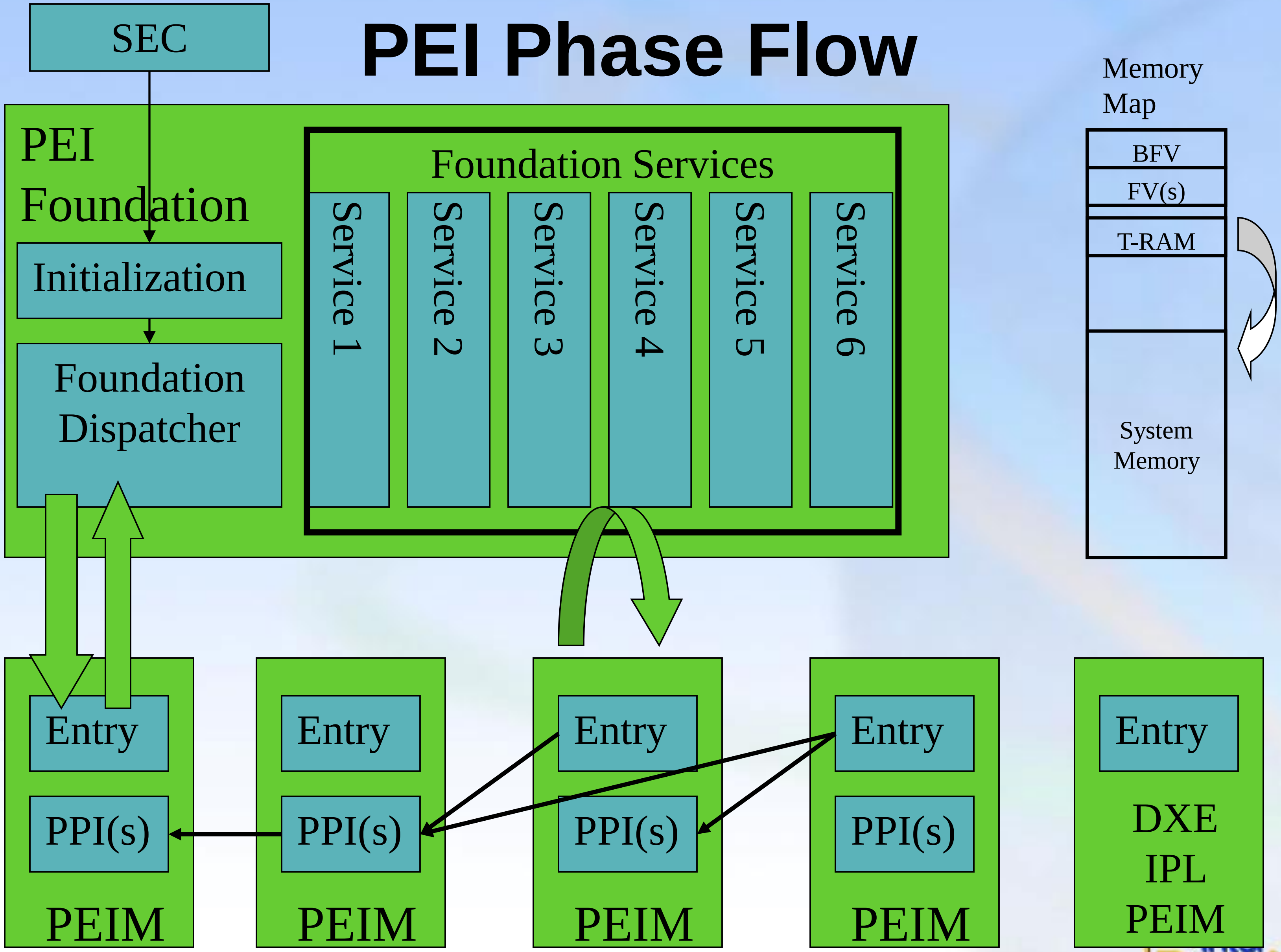
- Single binary for each CPU architecture
  - Flat model execution
- Mostly written in C
- Two main components
  - A dispatcher
    - Locates modules (PEIMs) in Firmware Volume (FV)
    - Execute modules in a predictable useful order
  - PEI services
    - Common functions useful to all PEIMs
- Initialization of main memory
- Ascertains the boot-mode
  - S3, Recovery, normal boot, etc


PEI Start
PEI Foundation Dispatch Initialization
Dispatch PEIMs in an orderly manner
PEI Foundation dispatches DXE IPL
PEIM n
PEIM 3
PEIM 2
PEIM 1
Entry Point
DXE IPL PEIM
Entry Point
DXEIPL PPI
Load/Start DXE

Where is PEI stored?


PEI Foundation
Platform PEIM
Chipset PEIM
Service PEIMs
Boot Firmware Volume (BFV)
Main Firmware Volume (FV)
Other Modules, drivers, and data

PEI Phase Flow



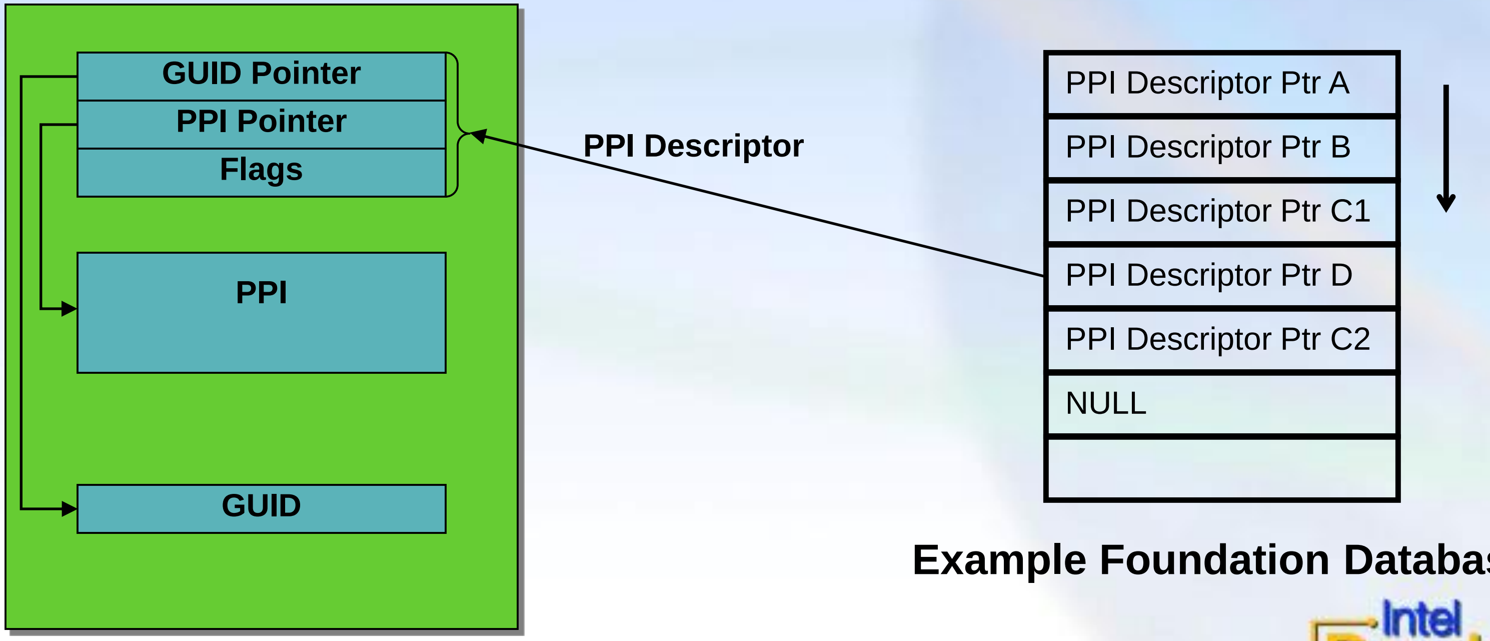
More detail on PEI Dispatcher

Foundation
Dispatcher

- Dependency Expressions
  - Declares the interfaces that a PEI Module Requires
  - PPI's are the dependency
- Dispatcher
  - Evaluates Dependency Expressions
  - Executed PEI Modules

Interface among modules

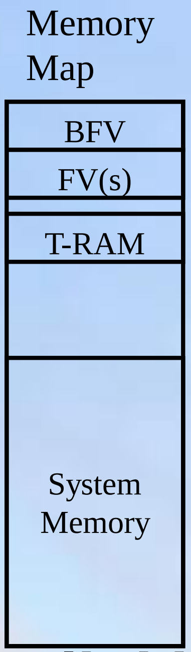
- Declarations in ROM, Described by PPI descriptor
- PEI Foundation Maintains Database PPI of Descriptors
- PPI Database is opaque to a PEIM and contents can be queried or manipulated using Foundation Services.



Example Foundation Database

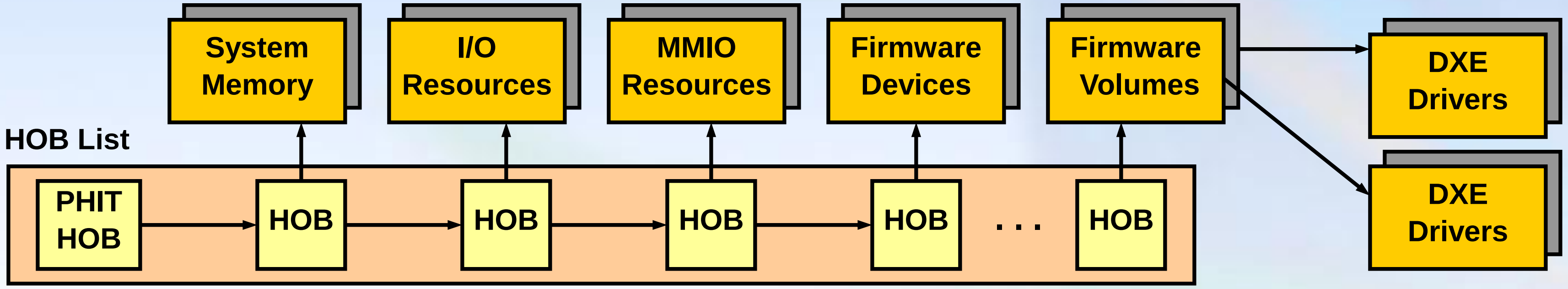
PEI's initial Memory

- Options for Deployment
  - CPU cache
  - SRAM on the platform
  - Other Custom Solution
- No temporary memory option
  - There are other challenges doing binary linking without memory.
  - It's what developer's are used to doing today



Transition from PEI to DXE

- PEI invokes the DXE Foundation
- Resources in HOBS
  - Firmware Volume
  - Memory Map

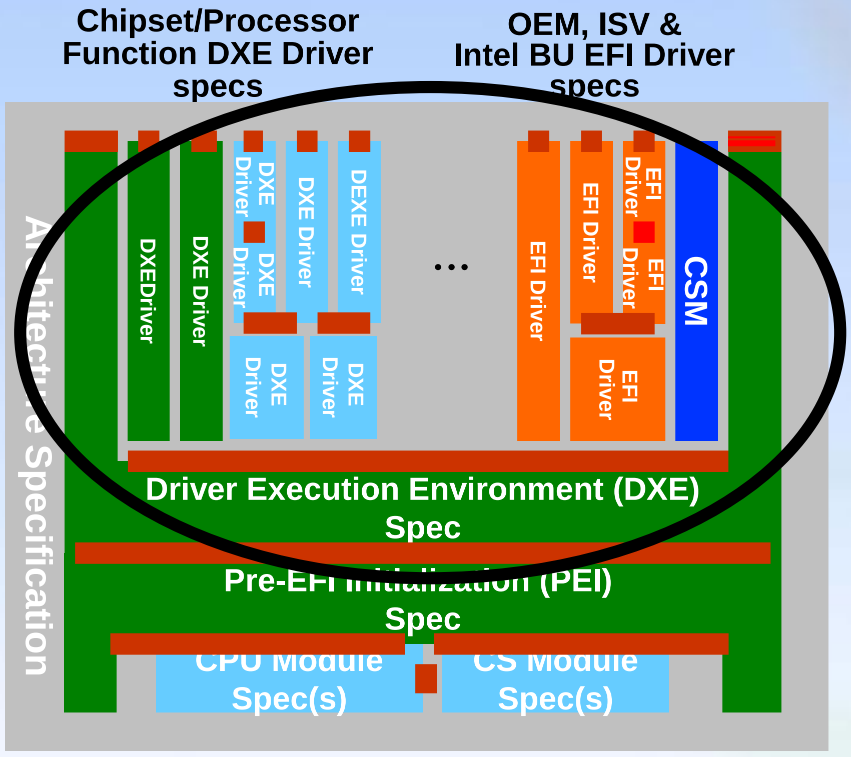


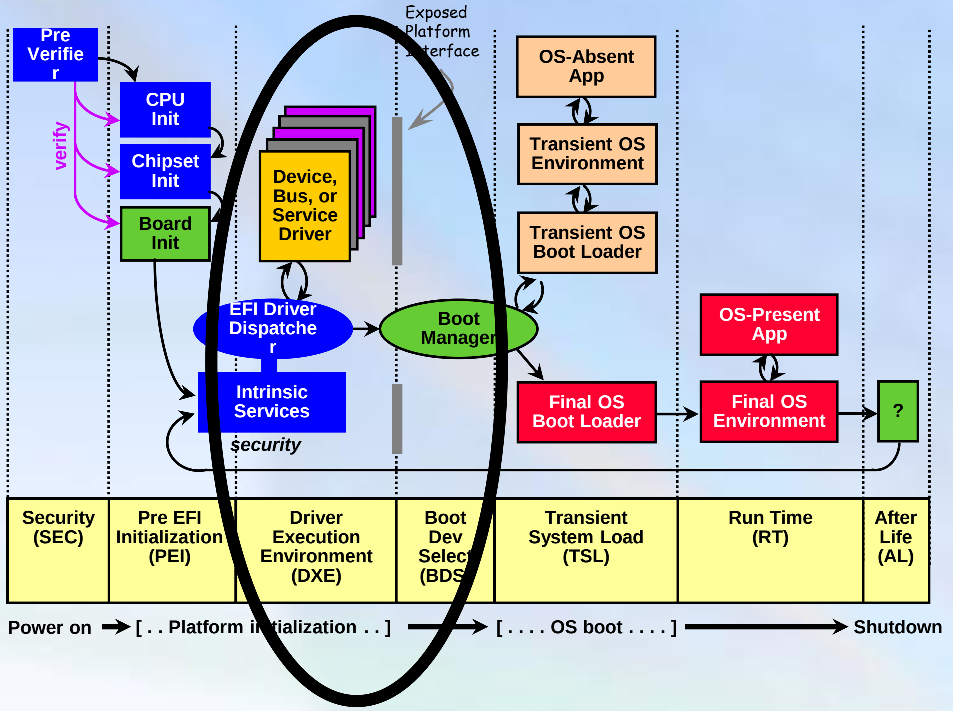
Small as possible. Just initialize memory.

Agenda

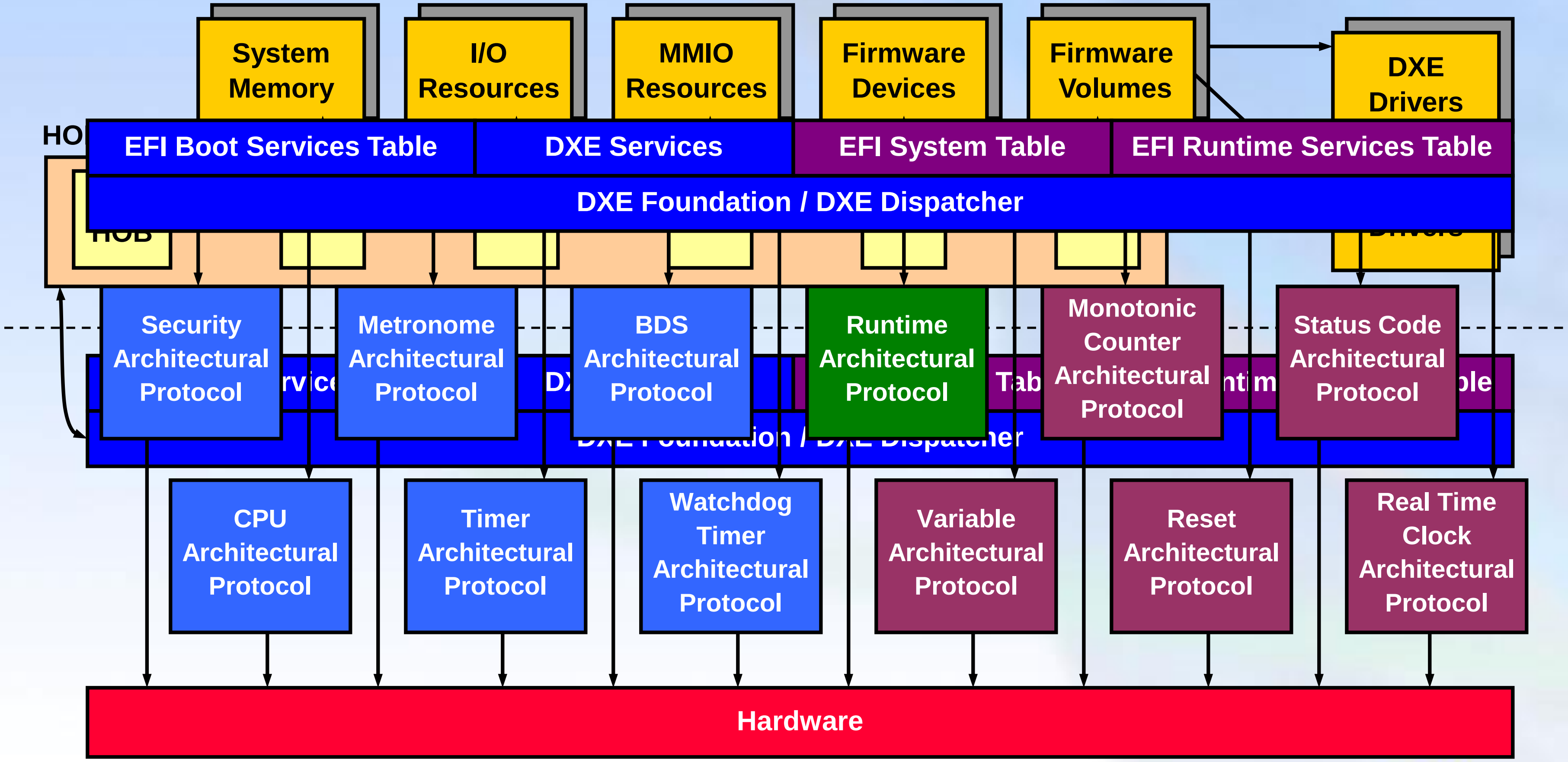
- Framework Overview
- Pre-EFI Initialization Foundation
- **Driver Execution Environment Foundation**
- Porting the Framework to Non-IA Silicon

DXE Foundation Overview

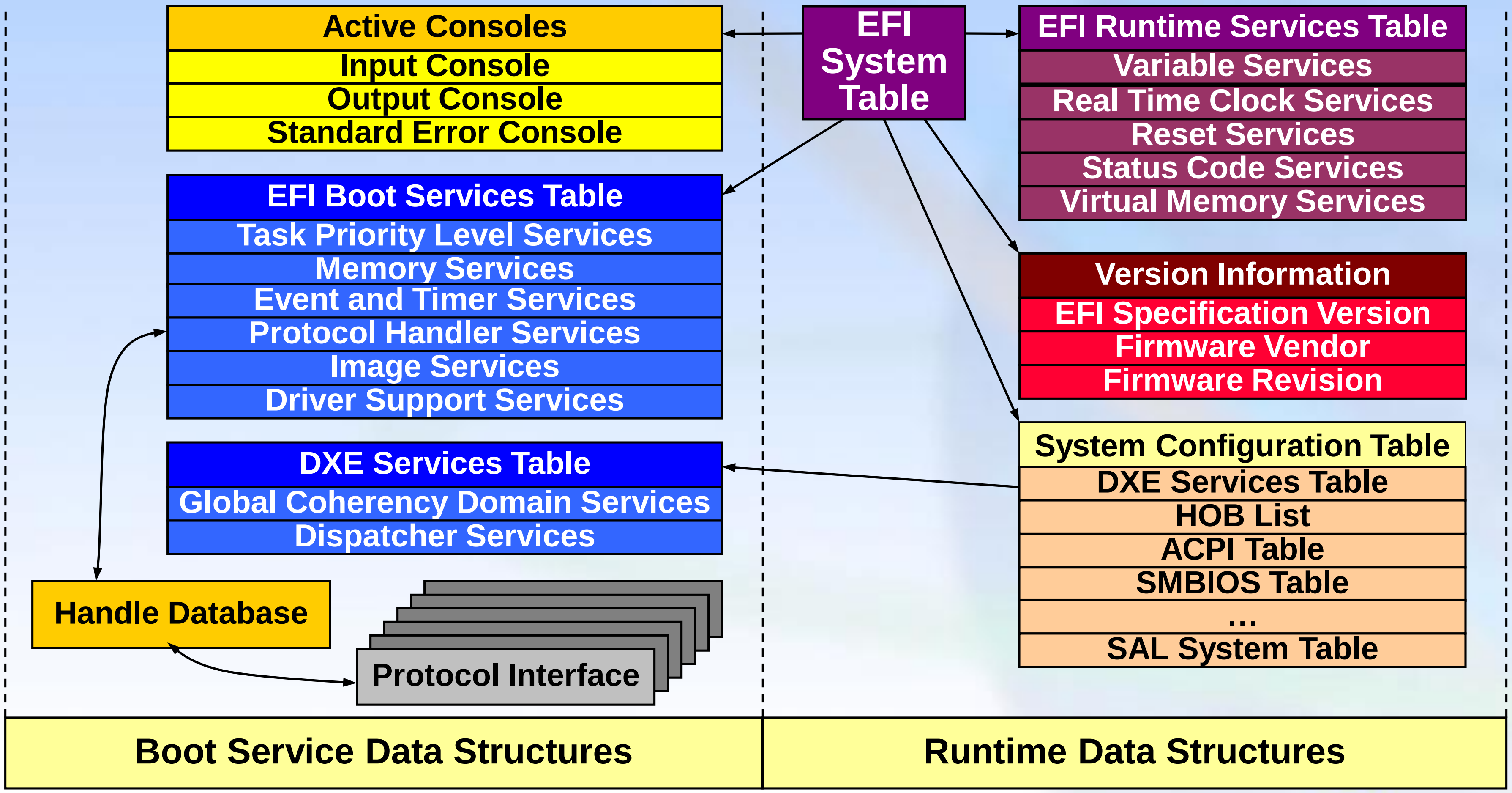

Chipset/Processor Function DXE Driver specs
OEM, ISV & Intel BU EFI Driver specs
Architecture Specification
DXEDriver
DXE Driver
DXE Driver
DXE Driver
DXE Driver
DEXE Driver
DXE Driver
DXE Driver
...
EFI Driver
EFI Driver
EFI Driver
EFI Driver
EFI Driver
CSM
Driver Execution Environment (DXE) Spec
Pre-EFI Initialization (PEI) Spec
CPU Module Spec(s)
CS Module Spec(s)



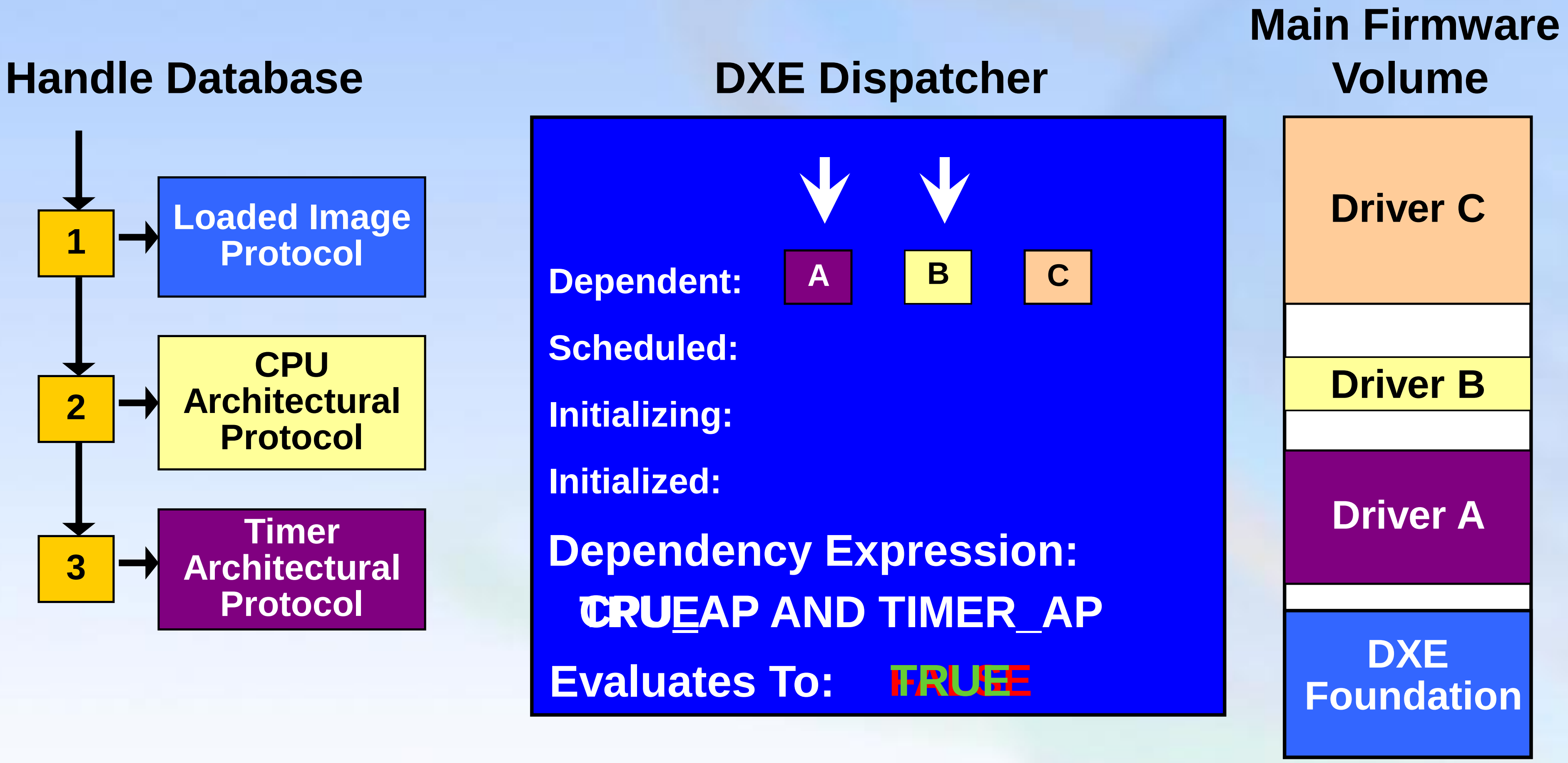
DXE Foundation Components


System Memory
I/O Resources
MMIO Resources
Firmware Devices
Firmware Volumes
DXE Drivers
HOB
EFI Boot Services Table
DXE Services
EFI System Table
EFI Runtime Services Table
DXE Foundation / DXE Dispatcher
Security Architectural Protocol
Metronome Architectural Protocol
BDS Architectural Protocol
Runtime Architectural Protocol
Monotonic Counter Architectural Protocol
Status Code Architectural Protocol
CPU Architectural Protocol
Timer Architectural Protocol
Watchdog Timer Architectural Protocol
Variable Architectural Protocol
Reset Architectural Protocol
Real Time Clock Architectural Protocol
Hardware

Initialization Order



DXE Foundation Dispatcher



Execution Order Determined at Runtime Based on Dependencies

Last Driver Executed in DXE Boot Device Selection (BDS)

- Invoked after DXE Dispatcher is Complete
- Implemented as a Driver
- Connects EFI Drivers as Required
  - Establishes Consoles (Keyboard, Video)
  - Processes EFI Boot Options (Boots OS)
- Tests Memory (Optional)

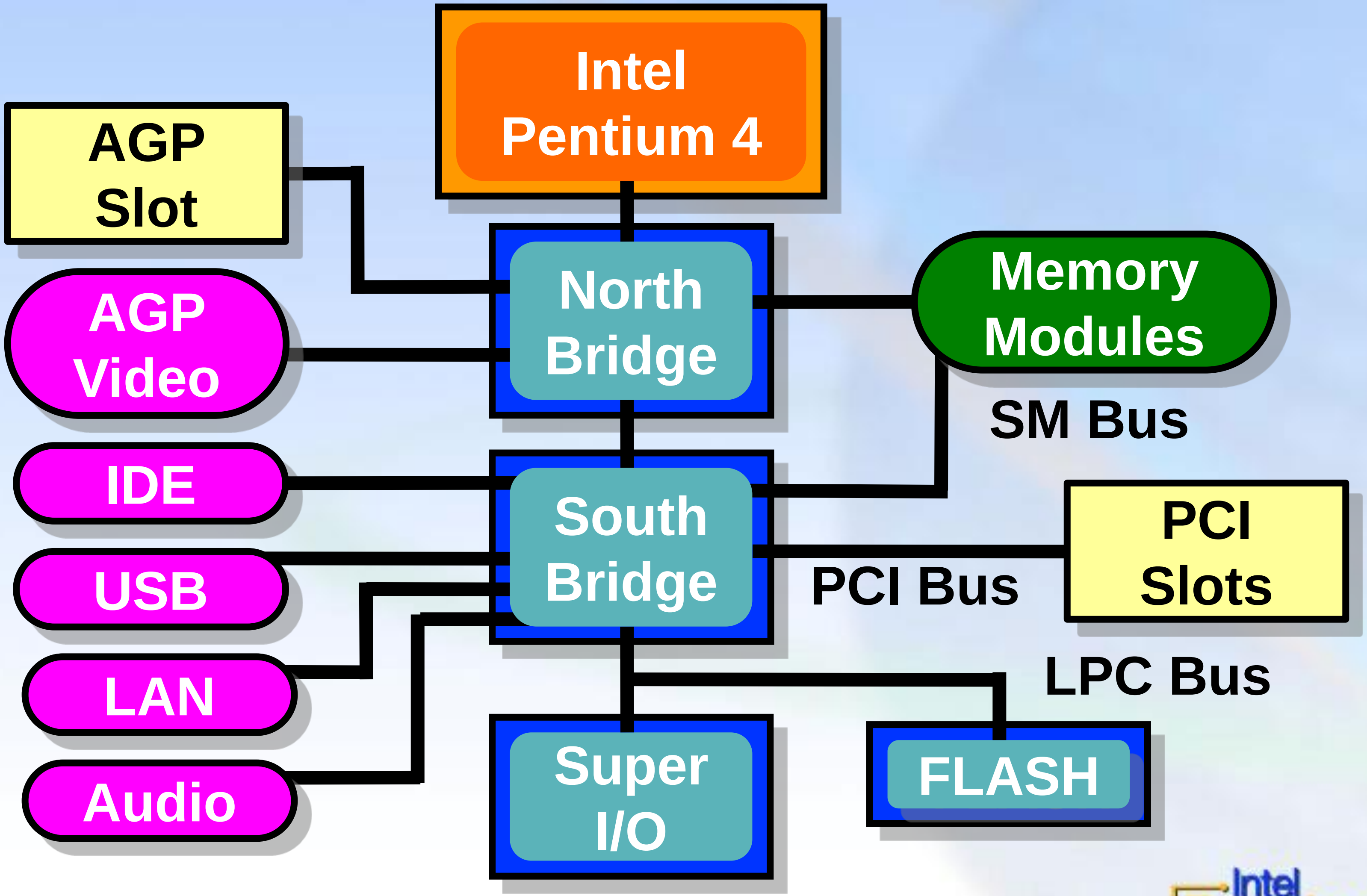
Completes Platform Initialization
Boots Operating System

Agenda

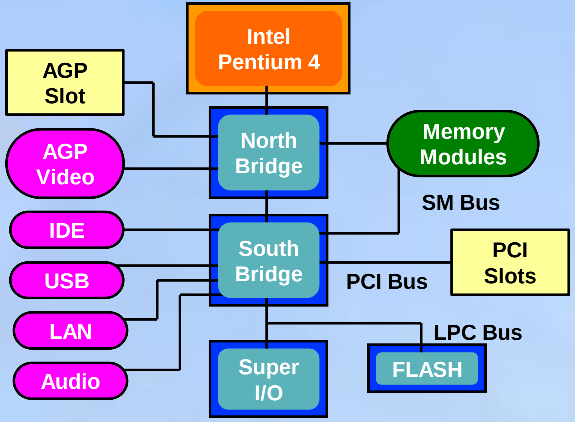
- Framework Overview
- Pre-EFI Initialization Foundation
- Driver Execution Environment Foundation
- **Porting the Framework to Non-IA Silicon**

**Questions on Framework, EFI,
PEI Foundation, or DXE Foundation?**

Desktop System Block Diagram

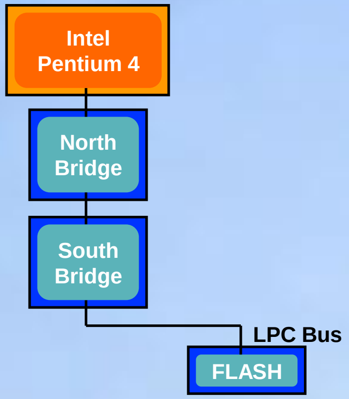


Recommended Porting Strategy



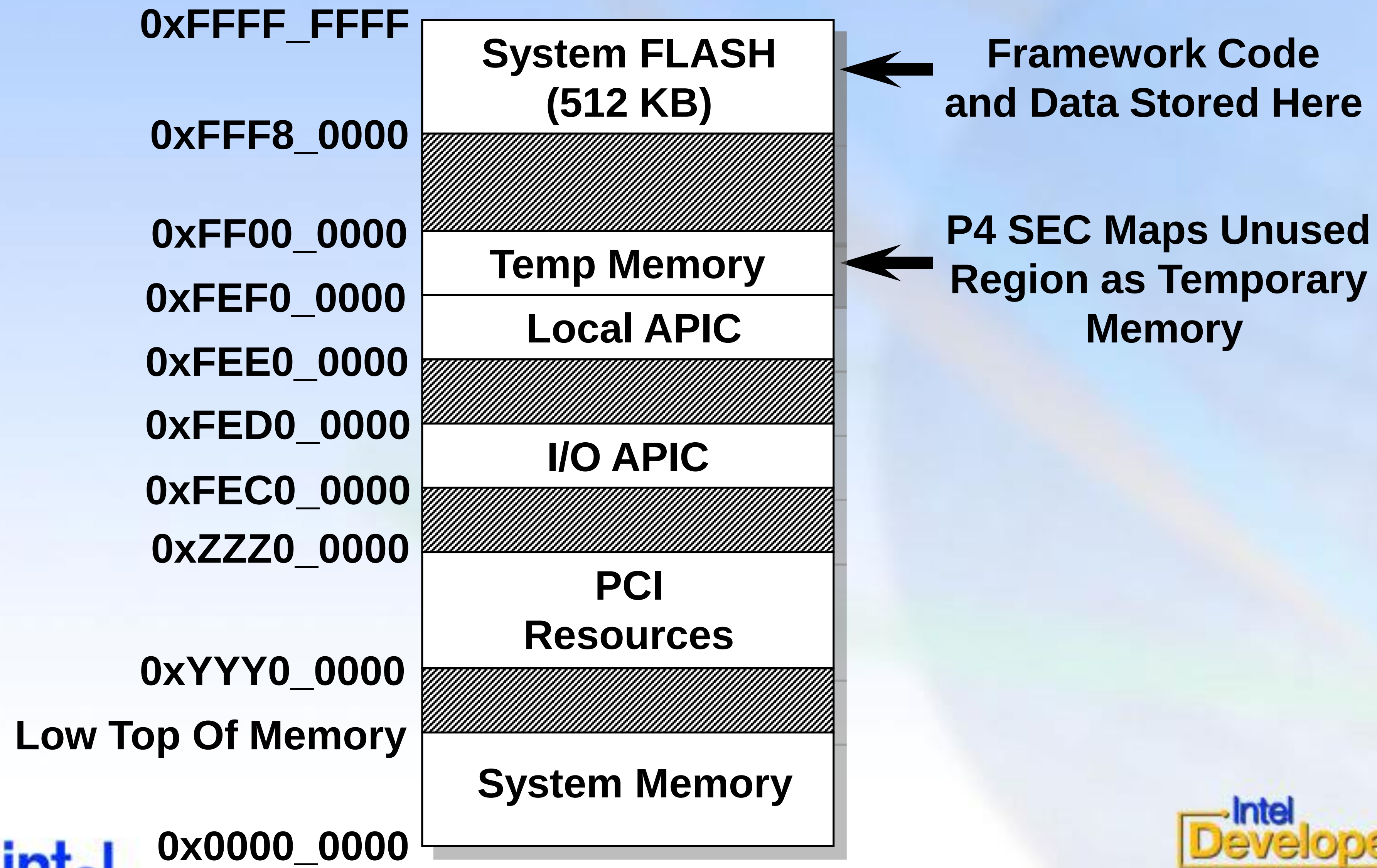
- Do Minimum to Run EFI Shell
  - Use Pentium 4 SEC Code
  - Port PEI Modules to Initialize System Memory
  - Port DXE Architectural Protocol Drivers
  - Port Drivers Required for Console Services
- Incrementally Add Platform Features
  - Add Video, Hard Disks, and Network to DXE
  - Add Compatibility Support Module to DXE
  - Customize Boot Device Selection Driver
  - Add ACPI and SMM Support to DXE
  - Add S3 and Recovery Support to DXE

Reset and SEC

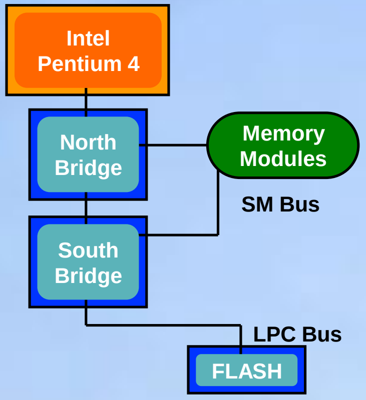


- Reset Fetches Code from FLASH
  - All Instructions Non-Cached
  - Enables Flat Protected Mode Execution
- Pentium 4 SEC Enables Temp Memory
  - All Instructions Remain Non-Cached
  - Data and Stack Cached
  - Enables Execution of C Code
  - Transfers Control to PEI Foundation

Temporary Memory



PEI Modules



Pentium 4 CPU PEIM	Generic	Init and CPU I/O
DXE IPL PEIM	Generic	Starts DXE Foundation
PCI Configuration PEIM	PCAT	Uses I/O 0xCF8, 0xCFC
Stall PEIM	PCAT	Uses 8254 Timer
Status Code PEIM	Platform	Debug Messages
SMBUS PEIM	South Bridge	SMBUS Transactions
Memory Controller PEIMs	North Bridge	Read SPD, Init Memory
Motherboard PEIM	Platform	FLASH Map, Boot Policy

SMBUS PEIM Services

```
typedef
EFI_STATUS
(EFIAPI *PEI_SMBUS_PPI_EXECUTE_OPERATION) (
  IN      EFI_PEI_SERVICE            **PeiServices,
  IN      struct EFI_PEI_SMBUS_PPI   *This,
  IN      EFI_SMBUS_DEVICE_ADDRESS   SlaveAddress,
  IN      EFI_SMBUS_DEVICE_COMMAND   Command,
  IN      EFI_SMBUS_OPERATION        Operation,
  IN      BOOLEAN                    PecCheck,
  IN OUT  UINTN                      *Length,
  IN OUT  VOID                       *Buffer
  );

typedef struct {
  PEI_SMBUS_PPI_EXECUTE_OPERATION  Execute;
  PEI_SMBUS_PPI_ARP_DEVICE         ArpDevice;
} EFI_PEI_SMBUS_PPI;
```

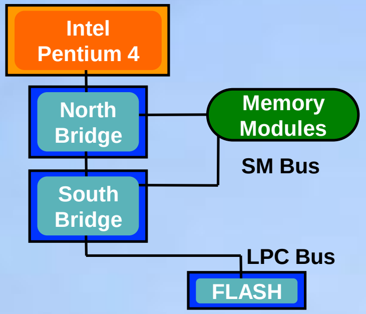
SMBUS Read Buffer Pseudo Code

```
#define SMBUS_R_HD0  0xEFA5
#define SMBUS_R_HBD  0xEFA7

EFI_PEI_SERVICES             *PeiServices;
SMBUS_PRIVATE_DATA           *Private;
UINT8   Index, BlockCount   *Length;
UINT8                        *Buffer;

BlockCount = Private->CpuIo.IoRead8 (
               *PeiServices,Private->CpuIo,SMBUS_R_HD0);
if (*Length < BlockCount) {
  return EFI_BUFFER_TOO_SMALL;
} else {
  for (Index = 0; Index < BlockCount; Index++) {
    Buffer[Index] = Private->CpuIo.IoRead8 (
                      *PeiServices,Private->CpuIo,SMBUS_R_HBD);
  }
}
```

DXE Architectural Protocols



Watchdog	Generic	Uses Timer-based Events
Monotonic Counter	Generic	Uses Variable Services
Runtime	Generic	Platform Independent
CPU	Generic	Pentium 4 DXE Driver
BDS	Generic	Use Sample One for Now
Timer	PCAT	Uses 8254 Timer
Metronome	PCAT	Uses 8254 Timer
Reset	PCAT	I/O 0xCF9
Real Time Clock	PCAT	I/O 0x70-0x71
Security	Platform	Platform Specific Authentication
Status Code	Platform	Debug Messages
Variable	Platform	Depends on FLASH Map

NT Emulation Timer Arch Protocol

```
EFI_STATUS
TimerDriverSetTimerPeriod (
  IN EFI_TIMER_ARCH_PROTOCOL  *This,
  IN UINT64                   TimerPeriod
  )
{
. . .
  gWinNt->EnterCriticalSection (&mNtCriticalSection);
  mTimerPeriod = TimerPeriod;
  mCancelTimerThread = FALSE;
  gWinNt->LeaveCriticalSection (&mNtCriticalSection);
  mNtLastTick = gWinNt->GetTickCount ();
  mNtTimerThreadHandle = gWinNt->CreateThread (
                                   NULL,
                                   0,
                                   NtTimerThread,
                                   &mTimer,
                                   0,
                                   &NtThreadId);
. . .
}
```

XScale Timer Arch Protocol

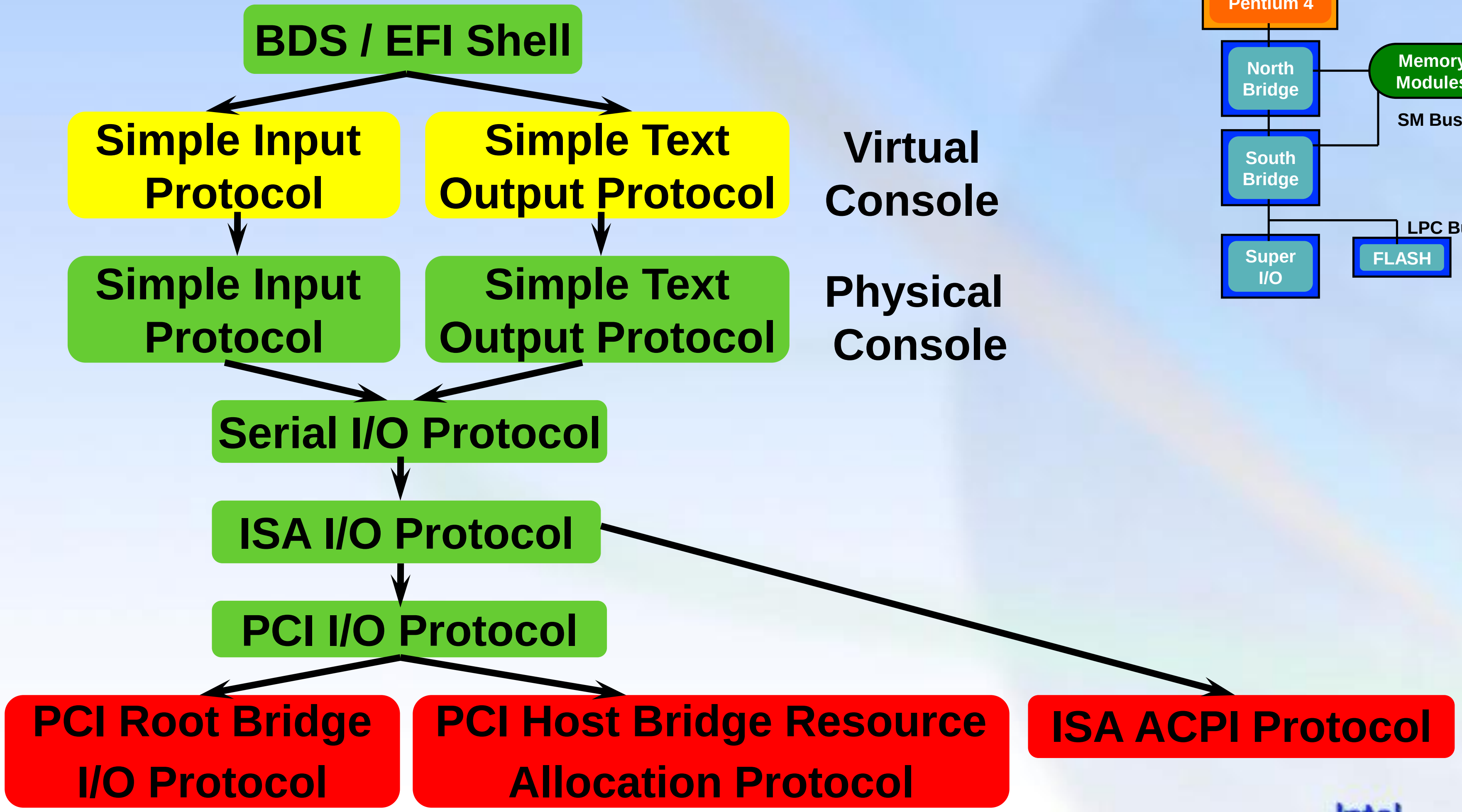
```
EFI_STATUS
TimerDriverSetTimerPeriod (
  IN EFI_TIMER_ARCH_PROTOCOL  *This,
  IN UINT64                   TimerPeriod
  )
{
 UINT64  Count;
 UINT32  Data;
. . .
 Count = DivU64x32 (MultU64x32 (TimerPeriod, OST_CRYSTAL_FREQ) + 5000000,
                   10000000, NULL);
 mCpuIo->Mem.Read  (mCpuIo,EfiWidthUint32,OSCR_BASE_PHYSICAL,1,&Data);
 Data += (UINT32)Count;
 mCpuIo->Mem.Write (mCpuIo,EfiWidthUint32,OSMR0_BASE_PHYSICAL,1,&Data);
 mCpuIo->Mem.Read  (mCpuIo,EfiWidthUint32,OIER_BASE_PHYSICAL,1,&Data);
 Data |= (UINT32)1;
 mCpuIo->Mem.Write (mCpuIo,EfiWidthUint32,OIER_BASE_PHYSICAL,1,&Data);
 mCpuIo->Mem.Read  (mCpuIo,EfiWidthUint32,ICMR_PHYSICAL,1,&Data);
 Data |= (UINT32)(1 << SA_OST0_IRQ_No);
 mCpuIo->Mem.Write (mCpuIo,EfiWidthUint32,ICMR_PHYSICAL,1,&Data);
. . .
}
```

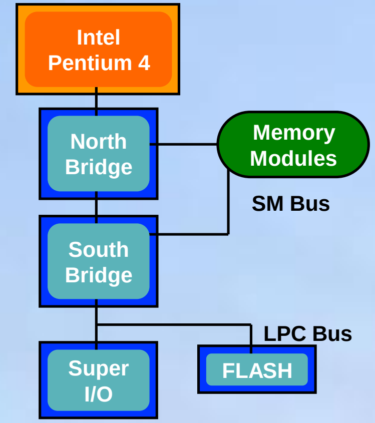
8254 Based Timer Arch Protocol

```
EFI_STATUS
TimerDriverSetTimerPeriod (
  IN EFI_TIMER_ARCH_PROTOCOL  *This,
  IN UINT64                   TimerPeriod
  )
{
 UINT64  Count;
 UINT8   Data;
. . .
 Count = DivU64x32 (MultU64x32(119318, (UINTN) TimerPeriod) + 500000,
                    1000000, NULL);
 Data = 0x36;
 mCpuIo->Io.Write(mCpuIo,EfiCpuIoWidthUint8,TIMER_CONTROL_PORT, 1, &Data);
 mCpuIo->Io.Write(mCpuIo,EfiCpuIoWidthFifoUint8,TIMER0_COUNT_PORT,2,&Count);
 mLegacy8259->EnableIrq (mLegacy8259, Efi8259Irq0, FALSE);
. . .
}
```

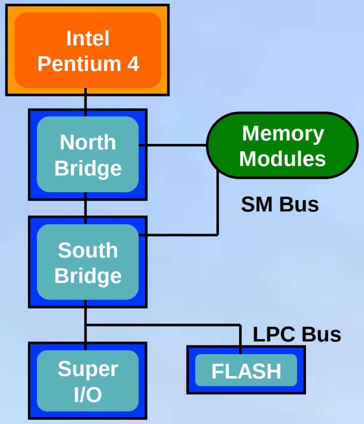
Different Implementations
Same Protocol Interface

Serial Terminal Console Services





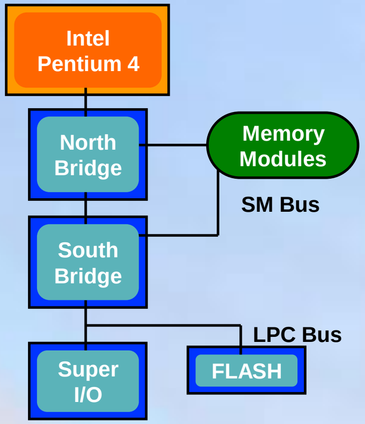
Serial Terminal Console Drivers



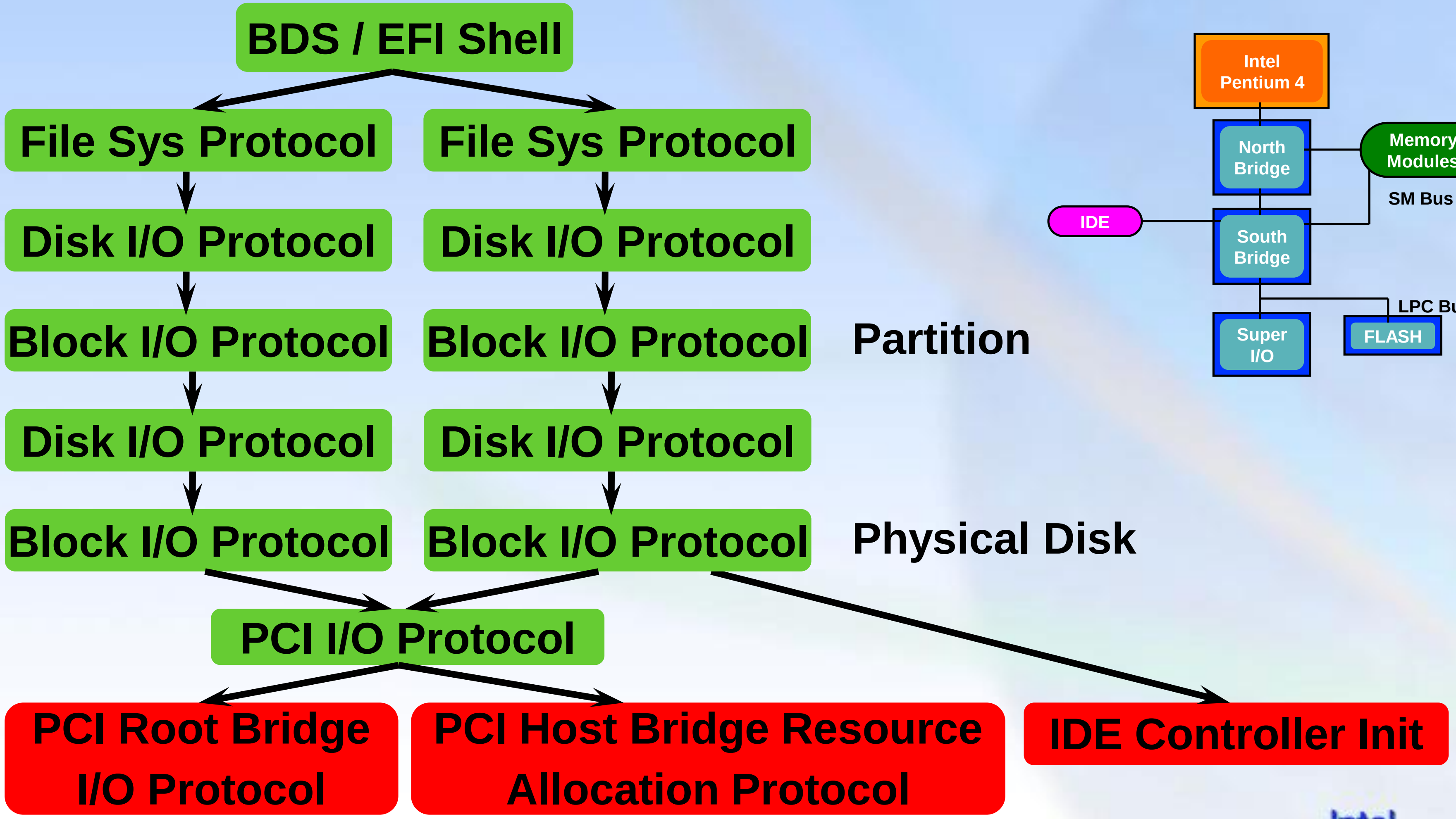
BDS / EFI Shell	Generic	
Console Splitter	Generic	
Terminal	Generic	
ISA Serial	PCAT	
ISA Bus	Generic	
PCI Bus	Generic	
Console Platform	Platform	Platform Specific Policy
PCI Root Bridge	North Bridge	Work with Chipset Vendor
PCI Host Bridge	North Bridge	Work with Chipset Vendor
ISA ACPI	Super I/O	Work with Super I/O Vendor

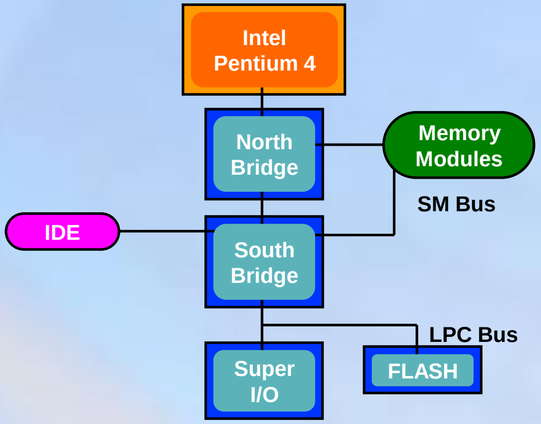
Porting Phase 1 Summary
Do Minimum to Run EFI Shell

Status Code	PEI	Platform
Memory Controller	PEI	North Bridge
SMBUS	PEI	South Bridge
Motherboard	PEI	Platform
Security	DXE	Platform
Status Code	DXE	Platform
Variable	DXE	Platform
Console Platform	DXE	Platform
PCI Root Bridge	DXE	North Bridge
PCI Host Bridge	DXE	North Bridge
ISA ACPI	DXE	Super I/O



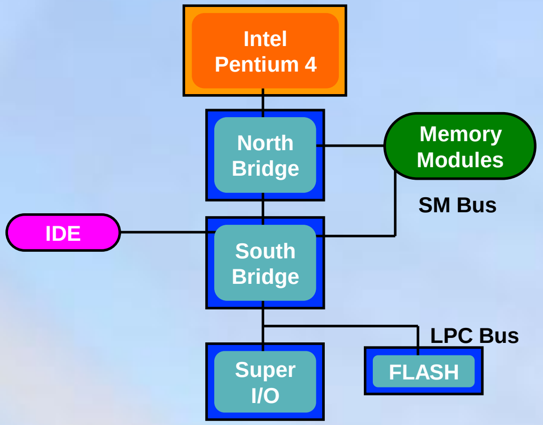
Add Platform Features - IDE





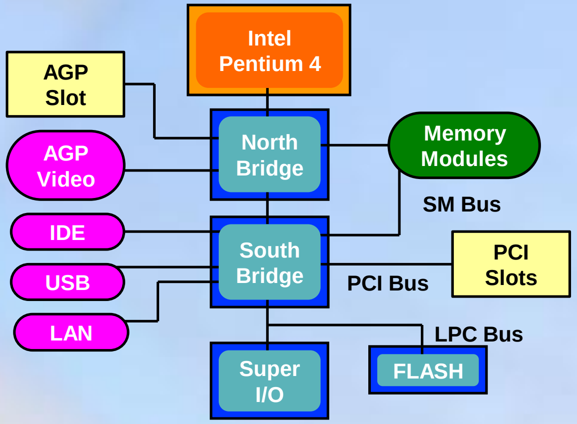
Add Platform Features - IDE

BDS / EFI Shell	Generic	
FAT	Generic	
Partition	Generic	
Disk I/O	Generic	
IDE Bus	PCAT	
PCI Bus	Generic	
PCI Root Bridge	North Bridge	
PCI Host Bridge	North Bridge	
IDE Controller Init	South Bridge	IDE Channel Attributes



Add Platform Features

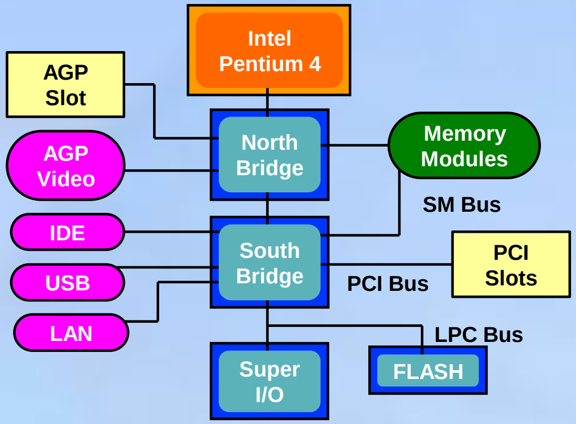
- Integrated USB
  - Work with Chipset Vendor
  - USB Host Controller Protocol
- Integrated Video
  - Work with Chipset Vendor
  - UGA Draw and UGA I/O Protocol
- Integrated LAN
  - Work with Chipset Vendor
  - UNDI Driver
- AGP and PCI Slots
  - Work with IHVs to Provide EFI Drivers



Compatibility Support Module

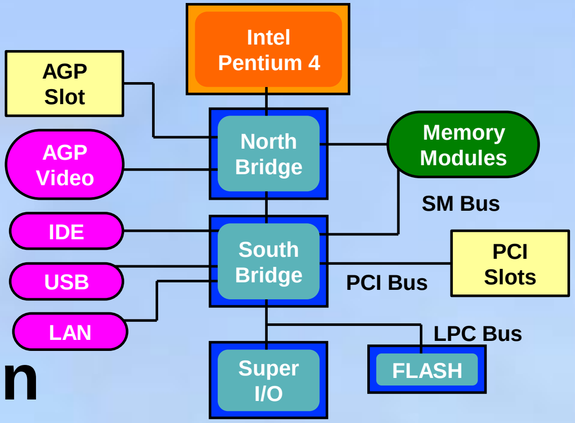
- Implement Chipset Specific Protocols
  - Legacy BIOS Platform
  - Legacy Region
  - Legacy Interrupt
- Enables POST of Legacy Option ROMs
  - Video Controllers
  - Disk Controllers
  - Network Interface Controllers
- Enables Legacy OS Boot
  - ACPI Disabled

FLASH Devices



- Implement EFI_FIRMWARE_VOLUME_BLOCK_PROTOCOL
  - Work FLASH Vendor
  - ReadBlock(), WriteBlock(), EraseBlock()
- Enables Variable Write Services
- Enables Recovery / Capsule Updates
  - Select Recovery Devices (Floppy, CD-ROM)
  - Implement Recovery PEIMs

SMM, ACPI, and S3



- SMM
  - Provides Legacy USB Emulation
  - Enables Chipset Workarounds
- ACPI
  - Platform Specific ACPI Tables and AML Code
  - Enables Booting of ACPI Aware OS
- S3 (If required)
  - Place System in Same State as Last Boot
  - Select Wake Devices / Events
  - Requires PEIMs and DXE Drivers Save Scripts
  - S3 Resume Replays Scripts

Customize Platform With BDS

- Provides User Interface
  - Custom Splash Screen
  - Custom Setup Screens
  - EFI Shell (Optional)
- Evaluates Boot Mode
  - Quiet Boot, Quick Boot, Diagnostics Boot, etc.
- Makes Default Policy Decisions
  - Default Console Selection
  - Default Boot Devices and Default Boot Order

Framework Designed to be Ported

Summary

- Framework is Next Generation Firmware Architecture
- PEI is Thinnest Layer of Code to Initialize Memory
- DXE Completes Platform Initialization and Boots Operating System
- Framework Designed to be Ported

Call To Action

- Download Framework Specifications
  - http://www.intel.com/technology/framework
  - PDF and On-Line Help Formats Available
- Provide Feedback
- Deploy Framework on your Platforms
- Who To Contact for More Information

– OEMs	Contact Participating IBV
– IHVs	Continue to Develop EFI Drivers
– OSVs	Continue to Develop EFI OS Loaders
– Other	Contact Intel

Q & A

Non-IA Silicon Support with the Intel® Platform Innovation Framework for the Extensible Firmware Interface

http://www.intel.com/technology/framework

Session	#	Day	Time	Room
Next Generation EFI 32 OS Loader	S186	Wed	11:00-11:50 AM	C-1/2
Introducing the Intel Platform Innovation Framework for EFI	S11	Wed	2:30-4:20 PM	C-1/2
Using the Wireless LAN to provision and manage mobile devices *	S115	Wed	2:30-3:20 PM	J-3
BIOS compatibility within the Intel Platform Innovation Framework for EFI	S12	Wed	4:30-5:20 PM	C-1/2
Non-Intel Silicon Support within the Intel Platform Innovation Framework for EFI	S13	Thu	10:00–11:50AM	C-1/2
Writing and Debugging EFI Drivers	S14	Thu	2:00-3:50 PM	C-1/2
EFI Specification Evolution	S15	Thu	4:00-4:50 PM	C-1/2

* non-EFI track

Non-IA Silicon Support with the Intel Platform Innovation Framework for the Extensible Firmware Interface

Vincent Zimmer	- Intel Corporation
Michael Kinney	- Intel Corporation
Robert Hart	- Insyde Software

Please remember to turn in your session survey form.

Intel Developer Forum

intel