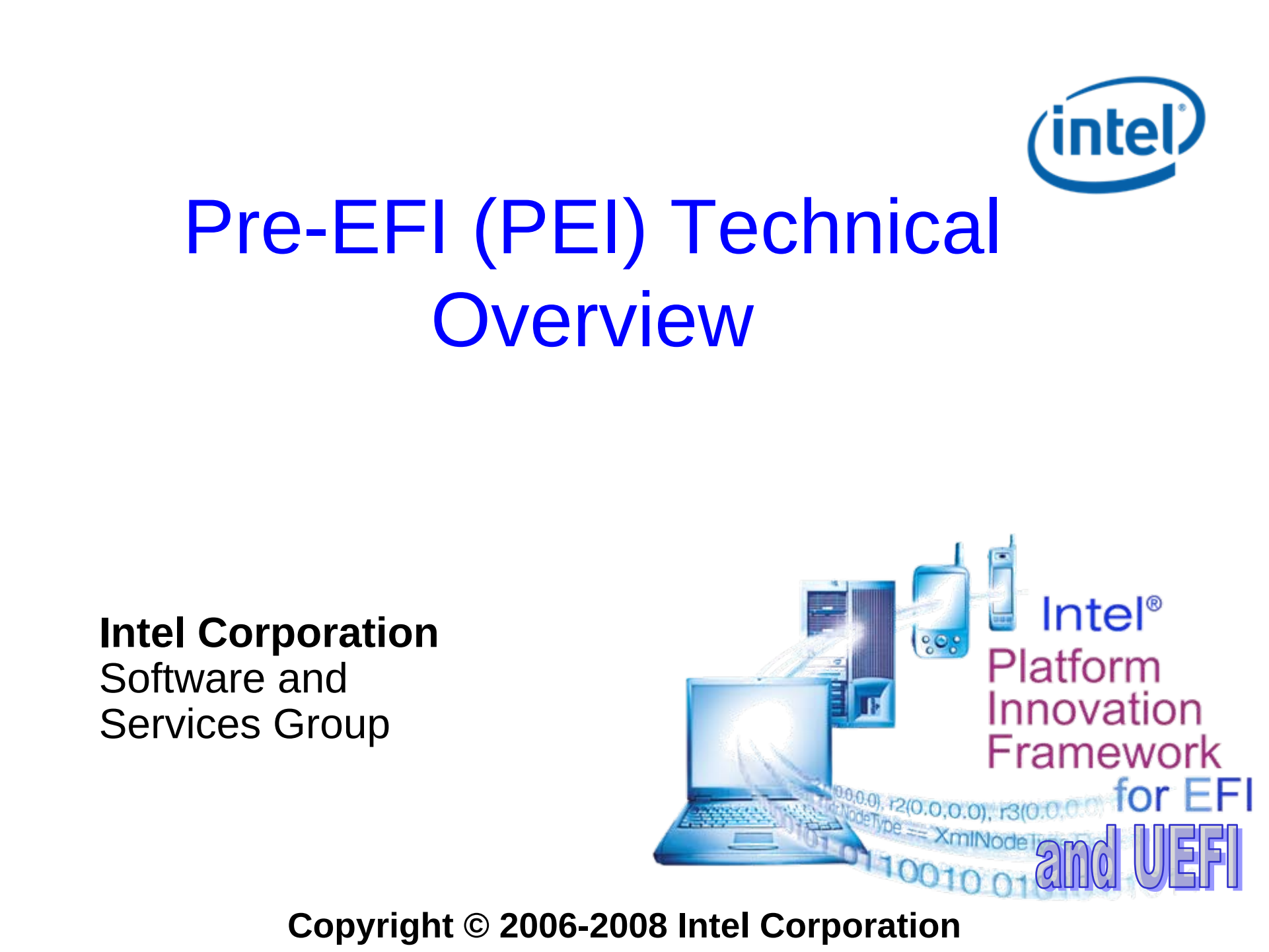


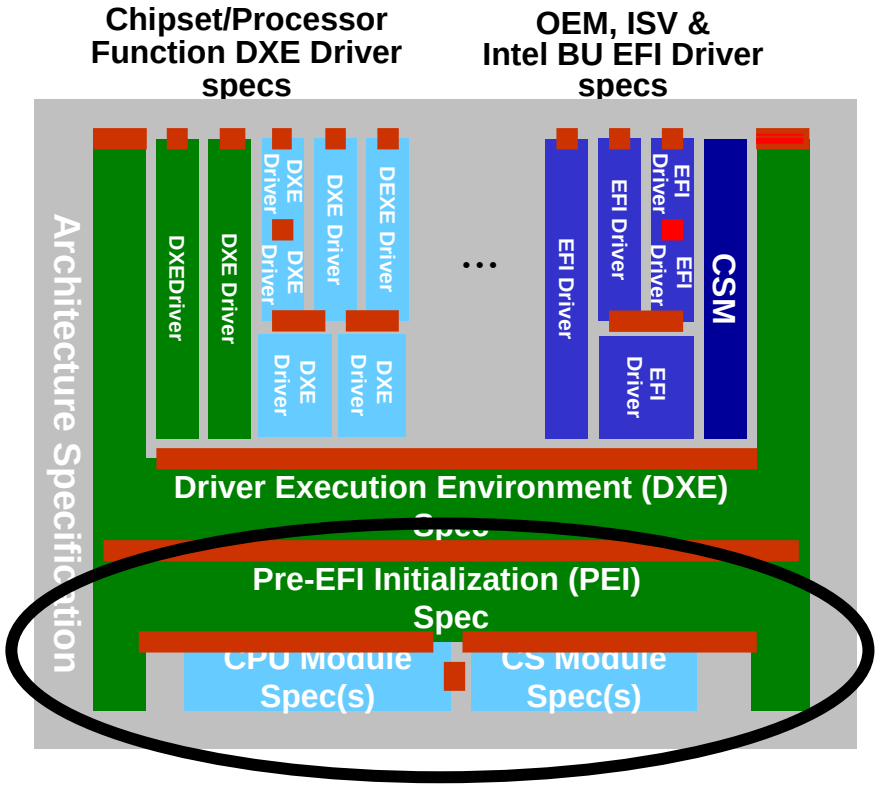
Pre-EFI (PEI) Technical Overview

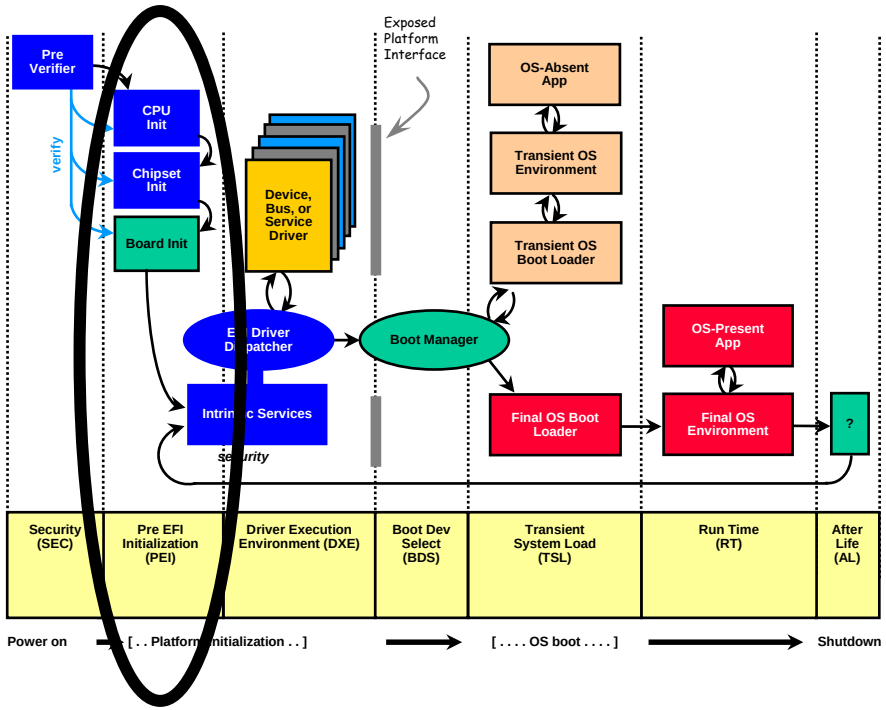
Intel Corporation
Software and
Services Group

Agenda

- PEI Overview
- PEI Core and PEI Services
- PEIM & PPI
- Dispatcher
- PEI Memory Environment
- PEI Hand-off Block (HOB)
- Boot Path
- PEI to DXE Transition

PEI Foundation



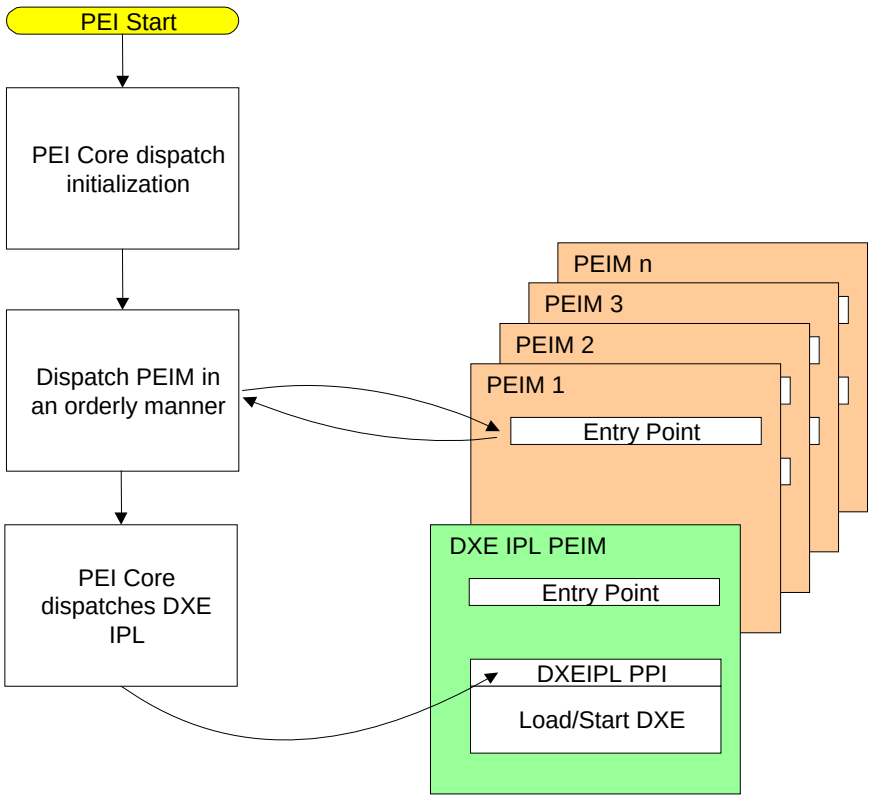


Why PEI

- Writing Modular code without memory is hard
- Legacy code is hand coded to different register rules
  - IBV A uses EBP and IBV B used EBX other registers are preserved by code convention
  - Porting from A to B requires rewriting lots of code!
- Quick Path for Memory Initialization and basic chipset initialization
- Modular Code for S3 and Recovery

What is PEI

- PEI is “Pre-EFI initialization”.
- Consumes reset, INIT, MCA
- Small, tight startup code
  - Startup with transitory memory store for call-stack (I.e., cache)
  - XIP from ROM
- Core locates, validates, and dispatches PEIMs
- Publishes own protocol and call-abstraction w/ PPI
  - Silicon/platform abstractions
- Primary goals
  - Discover boot mode
  - Launch modules that initialize main memory
  - Discovery & launch DXE core- Convey platform info into DXE



PEI Overview

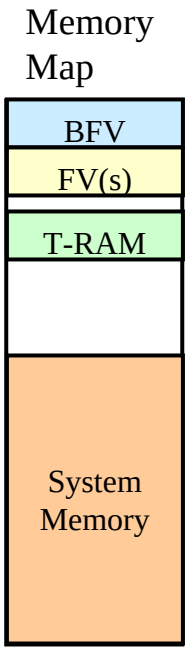
- Function:
  - Discover and initialize some RAM that won't be reconfigured
  - Describes location of FV(s) containing DXE Core & Architecture Protocols
  - Describes other fixed, platform specific resources that only PEI can know about
- Components:
  - Binaries: PEI Core and PEI Modules (PEIMs)
    - Standard header with execute in place code/data, Relocation information, Authentication information.
  - Interfaces: Methods of Inter-PEIM communication
    - Core set of services (PeiServices), PEIM to PEIM Interfaces (PPIs), and simple Notifies (no timer in PEI)
- Environment:
  - Small amount of temporary RAM that may be relocated
  - Executed from ROM

PEI Terminology

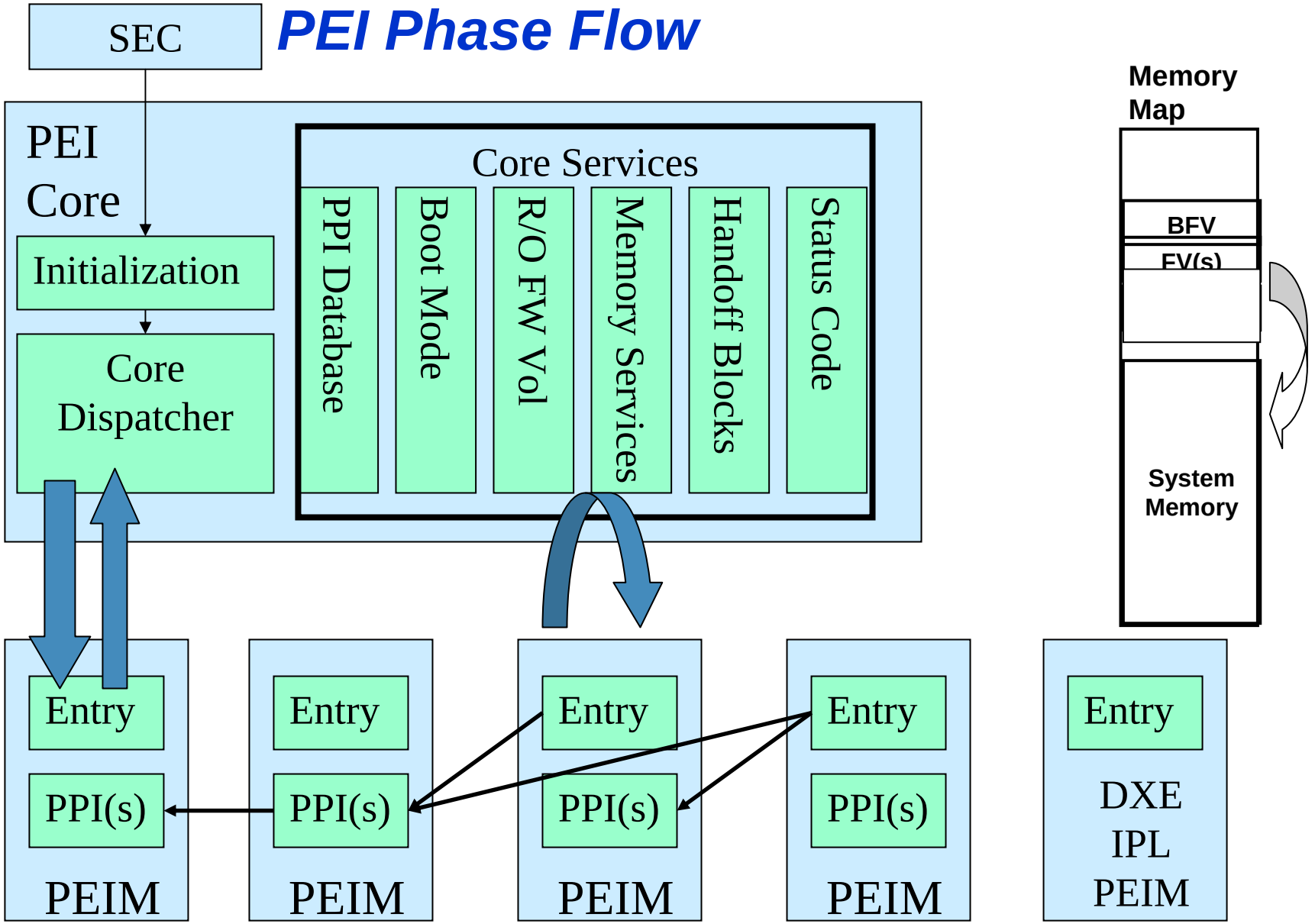
- PEI Core – The main PEI executable binary responsible for dispatching PEIM and provide basic services.
- PEIM – An executable binary that is loaded by the core to do various tasks and initializations.
- PPI – PEIM to PEIM Interface. An interface that allows a PEIM to invoke another PEIM.
- PEI Dispatcher – The part of the PEI core that searches for and executes the PEIM.
- PEI Services – Functions provided by the core visible to all PEIM.

PEI's initial Memory

- Minimum requirement for Framework PEI architecture is a small amount of temporary RAM
  - Minimum amount is dependent upon processor architecture requirements
    - IA32 sample implementation uses 8K
- PEI Temporary RAM requirement is met by architecturally defined mechanisms to allow processor cache to avoid data evictions, allowing the cache to be used as RAM
  - Processor family specific mechanism
    - P4 mechanism is different than PIII
    - IPF abstracts mechanism with a PAL call



PEI Phase Flow



Agenda

- PEI Overview
- PEI Core and PEI Services
- PEIM & PPI
- Dispatcher
- PEI Memory Environment
- PEI Hand-off Block (HOB)
- Boot Path
- PEI to DXE Transition

PEI Core

- Single binary for each CPU architecture
- Resides in Boot Firmware Volume (BFV)
- Well specified and validated
- Mostly written in C
  - Some assembly for performance optimizations
- Two main components
  - A dispatcher
    - Locates modules (PEIMs)
    - Execute modules in a predictable useful order
  - PEI services
    - Common functions useful to all PEIMs

Where the PEI Core code is located

- PEIMAIN is the main core source code module
  - invoked by PeiMain during transition from SEC to PEI
- Location in open source tree:
  - EDK I \Foundation\Core\Pei\PeiMain
  - EDK II \EdkModulePkg\Core\Pei\PeiMain
- Entry point - PeiCore

```
EFI_STATUS
EFIAPI
PeiCore (
  IN EFI_PEI_STARTUP_DESCRIPTOR  *PeiStartupDescriptor, // Information and services provided by SEC phase.
  IN PEI_CORE_INSTANCE           *OldCoreData       //Pointer to old core data that is used to initialize the core's data
areas.
  )
{   //  ...        ...        ...        ...
        InitializePpiServices (&PrivateData.PS, OldCoreData);
    // Call PEIM dispatcher
        PeiDispatcher (PeiStartupDescriptor, &PrivateData, DispatchData);
    //  ...        ...        ...        ...
    // Call to DXE IPL Entry
}
```

PEI Services

PPI Services:	Manages PEIM-to-PEIM Interface (PPIs) to facilitate intermodule calls between PEIMs. Interfaces are installed and tracked on a database maintained in temporary RAM.
Boot Mode Services:	Manages the boot mode (S3, S5, normal boot, diagnostics, etc.) of the system.
HOB Services:	Creates data structures called Hand-Off Blocks (HOBs) that are used to pass information to the next phase of the PI Architecture.
Firmware Volume Services	Walks the Firmware File Systems (FFS) in firmware volumes to find PEIMs and other firmware files in the flash device.
PEI Memory Services:	Provides a collection of memory management services for use both before and after permanent memory has been discovered.
Status Code Services:	Provides common progress and error code reporting services (for example, port 080h or a serial port for simple text output for debug).
Reset Services:	Provides a common means by which to initiate a warm or cold restart of the system.

See § 4.1 PI 1.1 Vol. 1Spec

PEI Services - Functions

PPI Services	InstallPpi () ReInstallPpi () LocatePpi () NotifyPpi ()
Boot Mode Services	GetBootMode () SetBootMode ()
HOB Services	GetHobList () CreateHob ()
Firmware Volume Services	FfsFindNextVolume () FfsFindNextFile () FfsFindSectionData ()
Memory Services	InstallPeiMemory () AllocatePages () AllocatePool () CopyMem () SetMem ()
Status Code Services	PeiReportStatusCode ()
Reset Services	PeiResetSystem ()

Agenda

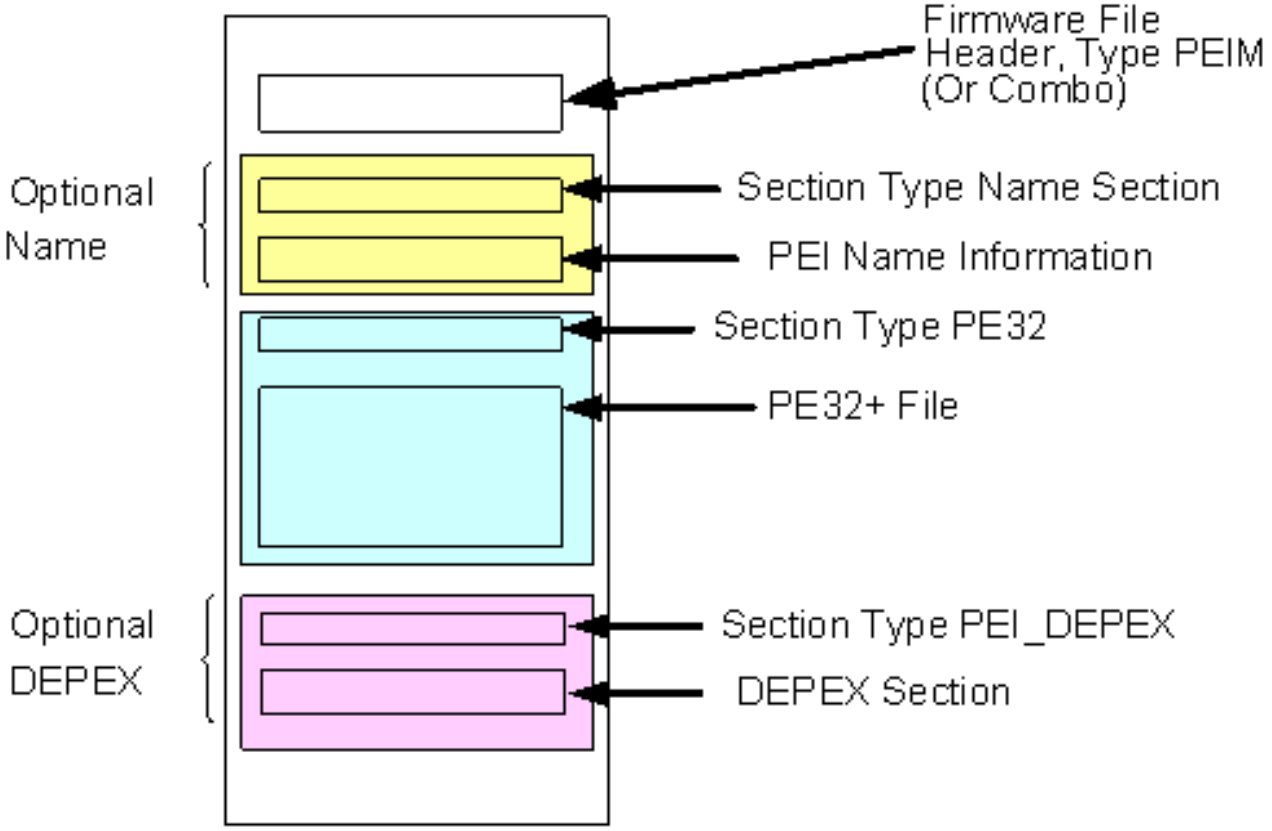
- PEI Overview
- PEI Core and PEI Services
- PEIM & PPI
- Dispatcher
- PEI Memory Environment
- PEI Hand-off Block (HOB)
- Boot Path
- PEI to DXE Transition

PEI Module Concept

- Modules in PEI are called PEI Modules (PEIMs)
  - Executable objects abstracting features
  - Separately built uncompressed binary images
  - PEIMs are contained in files within Firmware Volumes (FVs)
  - PEIMs can reside in multiple FVs as long as mechanisms for searching them is provided.
- PEIMs Are Execute In Place (XIP)
- PEIMs Define Interfaces To Other PEIMs
  - PEIM to PEIM Interface (PPI)
- PEIMs describe the requirements (PPIs) needed to run them
  - Dispatcher ensures requirements are met

PEIM Layout

- Each PEI Module (PEIM) is stored in a file
  - Standard header
  - Execute-in-place code/data section
  - Optional relocation information
  - Authentication information



See § 6.2 PI 1.1 Vol. 1 Spec

PEIM to PEIM Interfaces(PPIs)

PPI Classes:

Architectural PPI - PPI whose GUID is known to the PEI Foundation, and provide a common interface to the PEI Foundation of a service that has a platform-specific implementation, such as ReportStatusCode()

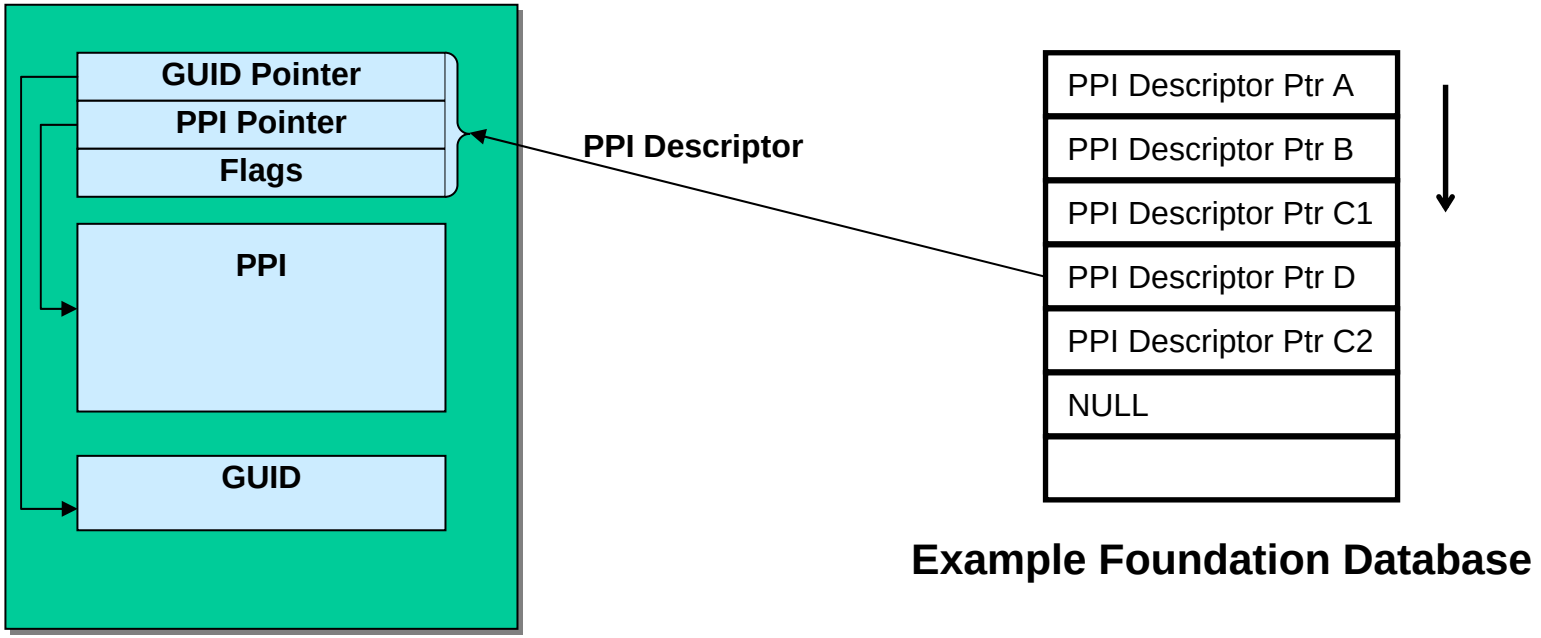
Additional PPIs are PPIs that are important for interoperability but are not depended upon by the PEI Foundation.

The PEI Services that provide access to PPIs are InstallPpi(), ReinstallPpi(), LocatePpi(), and NotifyPpi

See § 6.5 PI 1.1 Vol. 1Spec

PEIM to PEIM Interfaces(PPIs)

- Static declarations in ROM, described by a PPI descriptor
- PEI Foundation maintains database PPI of descriptors
- PPI database can be queried or manipulated using Foundation PEI Services



Example: PEIM Entry Point

- PEIM entry point

```
EFI_STATUS
EFIAPI
PeimEntry (
  IN EFI_FFS_FILE_HEADER   *FfsFileHeader,
  IN EFI_PEI_SERVICES      **PeiServices
  )
```

- PEI Services invoked using the syntax

```
(**PeiServices).InstallPpi (…)
```

PEIM Example

FwVol.c


```
EFI_STATUS
EFIAPI
PeiFvFindNextVolume (
  IN     EFI_PEI_SERVICES       **PeiServices,
  IN     UINTN                      Instance,
  IN OUT EFI_FIRMWARE_VOLUME_HEADER **FwVolHeader
  )
```

Code

Comment

```
*++ Routine Description:   Return the Firmware Volume instance requested
Arguments:
  PeiServices       - The PEI core services table.
  Instance          - Instance of FV to find
  FwVolHeader       - Pointer to contain the data to return
Returns:
  Pointer to the Firmware Volume instance requested
  EFI_INVALID_PARAMETER         - FwVolHeader is NULL
  EFI_SUCCESS                   - Firmware vol instance successfully
++*/
```

PEIM Example - Continued

```
{
. . .
// Locate all instances of FindFV
     Status = (**PeiServices).LocatePpi (
                          PeiServices,
                          &gEfiFindFvPpiGuid,
                          0,
                          NULL,
                          &FindFvPpi
                          );
 if (Status != EFI_SUCCESS) {
      Status = EFI_NOT_FOUND;
    } else {
      Status = FindFvPpi->FindFv (
                          FindFvPpi,
                          PeiServices,
                          &LocalInstance,
                          FwVolHeader
                          );

    }
  }
  return Status;
}
```

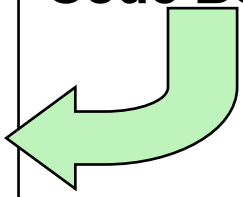
Source code

FwVol.c

Location in open source tree:

- EDK I \Foundation\Core\Pei\FwVol
- EDK II \EdkModulePkg\Core\Pei\FwVol

Code Body



Agenda

- PEI Overview
- PEI Core and PEI Services
- PEIM & PPI
- Dispatcher
- PEI Memory Environment
- PEI Hand-off Block (HOB)
- Boot Path
- PEI to DXE Transition

Dispatcher

- Hands control to the PEIMs in orderly manner.
- PEI code examines each file in all FVs of type PEIM.
- Examines dependency expression to order the execution of the file.
- DEPEX.

Dispatch Model

- The PEI Core uses two mechanisms to decide upon its schedule
  - Are the required interfaces for the PEIM installed?
    - DEPEX algorithm in core/simple BNF to evaluation
  - Is the file authentication state deemed sufficient?
    - OEM-provided algorithm/OEM-provided policy
- Restart the dispatch if PEIM sets Boot Mode to recovery.

Dispatching

- Dependency Expression (DEPEX) in Firmware File Section
- Notation
  - AcXpY = A consumes X and produces Y
- In general, provides Weak Ordering
- Rules of DEPEX can be made to maintain strict ordering, else many PEIM's might fail.

Dispatch Algorithm

- A PEIM is Ready To Run if
  - It Hasn't Run Already in the Same Pass
  - The GUIDs in It's DEPEX Appear In the PPI database describing PEIMs That Have Been Run *or*
  - It's DEPEX is NULL
  - If authentication section exists, it authenticates
- Dispatching Other FV
  - Multiple FV support adds some complexity
    - A PEIM must describe where other FVs are
      - Core uses a architecturally specified PPI (FindFv)
    - When an FV is discovered, it is added to Core algorithm's search order

Where the PEI Dispatcher Code is located

- Location in open source tree:
  - EDK I \Foundation\Core\Pei\Dispatcher\Dispatcher.c
  - EDK II \EdkModulePkg\Core\Pei\Dispatcher\Dispatcher.c
- Entry point – PeiDispatcher()

```
EFI_STATUS
PeiDispatcher (
  IN EFI_PEI_STARTUP_DESCRIPTOR  *PeiStartupDescriptor,     //Pointer in Startup Descriptor
  IN PEI_CORE_INSTANCE           *PrivateData,              //Pointer to the private data passed in
  IN PEI_CORE_DISPATCH_DATA      *DispatchData              //Pointer to Core dispatch data
  )
//  ...       ...       ...       ...
for (;;) {  // check for more FV that will have more PEIMs
    for (;;) { //Check all PEIMs in current FV
      Status = FindNextPeim (   &PrivateData->PS,  DispatchData->CurrentFvAddress,
                &DispatchData->CurrentPeimAddress  );
      //  ...       ...       ...       ...
      //If the PEIM has its dependencies satisfied, and its entry point has been found, so invoke it.
            PERF_START (
              (VOID *) (UINTN) (DispatchData->CurrentPeimAddress),  "PEIM", NULL, 0 );
       //   ...       ...       ...       ...
        DispatchData->CurrentPeim++;
      } // end inner for loop
} // end outer for loop
```

End Conditions of PEI

- A dispatch pass is done when no PEIM is dispatched during a pass through the known PEIMs
- We Have Memory
  - Report Data For Subsequent DXE Phase
  - Use HOB's for the hand-off information
- Done with dispatch: Invoke the DXE Initial Program Load (IPL)

Agenda

- PEI Overview
- PEI Core and PEI Services
- PEIM & PPI
- Dispatcher
- PEI Memory Environment
- PEI Hand-off Block (HOB)
- Boot Path
- PEI to DXE Transition

PEI pre-permanent memory Environment

- Purpose: returns initialized, tested memory
- Will need to discover boot type (path)
  - To properly handle INIT, S3, etc.
- No real system memory yet - processor resources are only context
  - Use processor cache as memory call stack
  - Developer must be aware of constraints of the environment, depending on processor architecture
- Flat, physical memory model
- Inputs: Location in the Firmware Volume. PEI Service Table
- Outputs: Update to state via PEI Service Calls

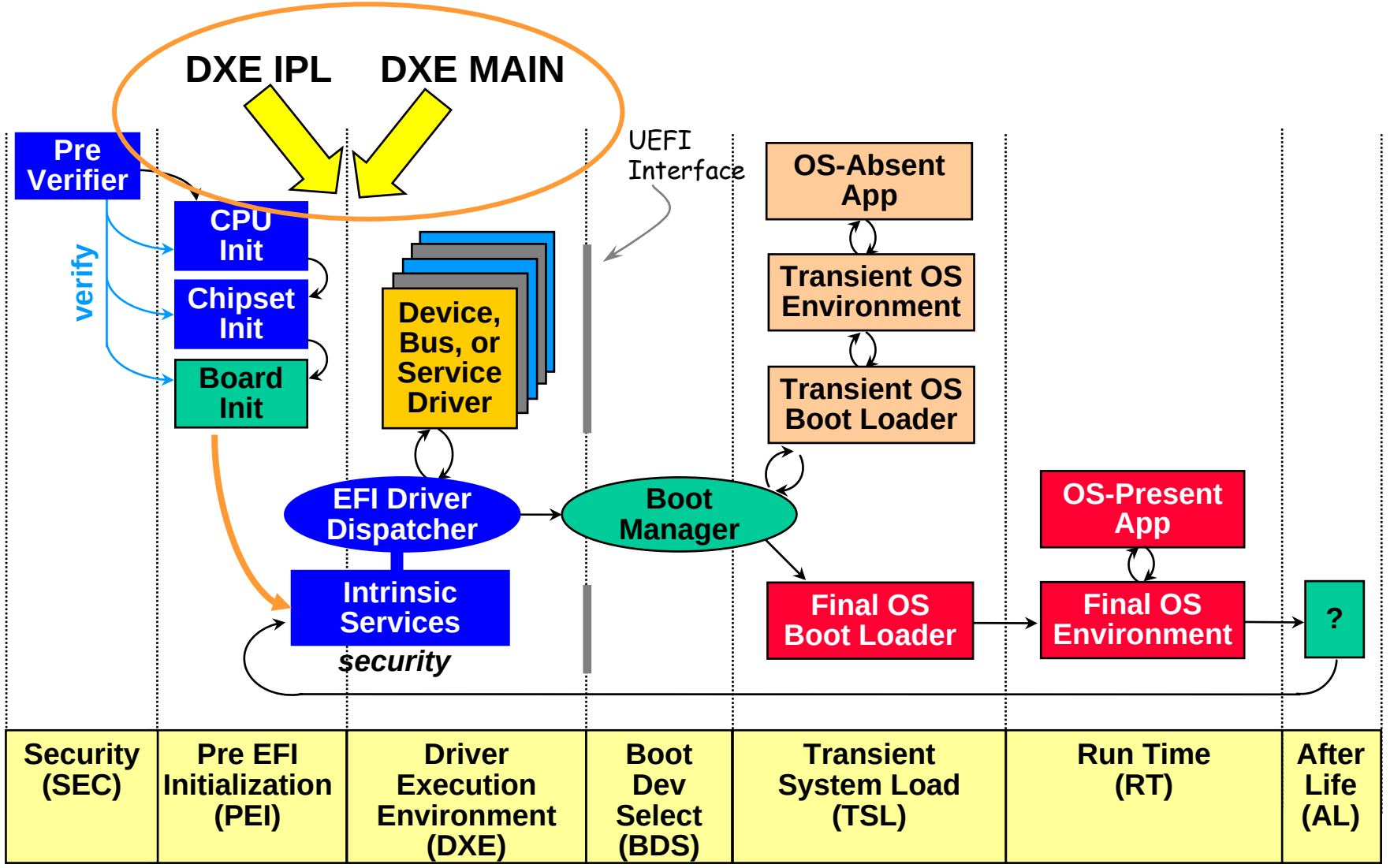
Post-Permanent Memory Environment

- Purpose: Prepare for DXE handoff
- Have memory for a stack and a “HOB List”
- Still XIP code running from FVs
- Still flat, physical memory model
- C-style calling conventions
- Inputs: Pointer to the first HOB
  - PEI Handoff Information Table (PHIT)

Agenda

- PEI Overview
- PEI Core and PEI Services
- PEIM & PPI
- Dispatcher
- PEI Memory Environment
- PEI Hand-off Block (HOB)
- Boot Path
- PEI to DXE Transition

Framework Architecture Execution



PEI Handoff Blocks

- HOBs bridge the gap between PEI and DXE
  - PEI collects the state in HOBs
- All data contained within a block (cell)
- Block is self-describing (type, length)
- Allocated in a heap (of Blocks)
  - No de-allocation
- DXE will maintain a “snap-shot” of HOBs when it starts.
  - Pointers in HOBs describe physical things that can not be moved…..FVs, physical memory properties
  - Or memory that is allocated by PEIMs for…. AP stacks, MP spinlocks
- All HOBs contained w/in PEI memory

Hand Off Blocks

- HOBs – a series of data structures in memory, created during PEI, that describe platform features, configuration, or data. HOBs are produced during PEI, and read-only during DXE (consumer).
- PEI must build HOB:
  - PEI Handoff Information Table (PHIT)
  - Resource Descriptor for physical system memory
  - Memory Allocation HOB - BSP Stack

See § 4.5 PI 1.1 Vol. 3 Spec

How HOBs Fit In To PEI & DXE

- PEI aggregates state in HOBs
- HOBs describe physical memory, physical I/O, allocated memory, allocated I/O
- HOBs are similar consepts to GCD in DXE
- GUIDed HOB allows private information to be conveyed into DXE Driver from associated PEIM.

HOB Types

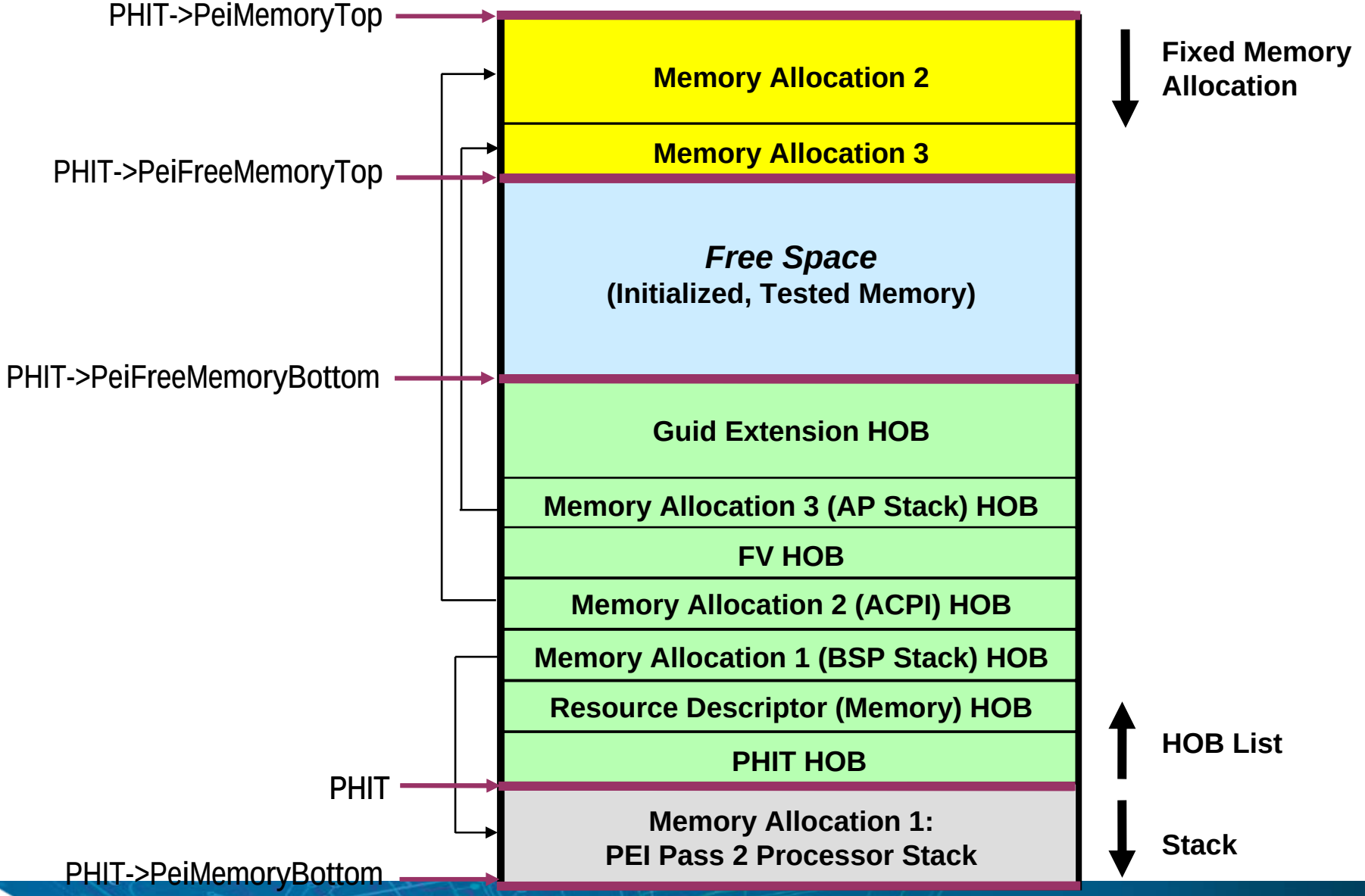
- PEI Handoff Information Table HOB (PHIT)
  - Describes PEI memory environment
  - Describes boot mode
- Resource Descriptor Hob to describe Physical Memory Descriptor
  - Describes physical system memory ranges
- Memory Allocation HOB
  - Describes memory ranges allocated by PEI
  - Describes stack allocation, IPF BSP store, etc.
- PEIM Specific (Named by GUID) HOB
  - Can pass information to DXE Drivers by GUID

See § 5 PI 1.1 Vol. 3 Spec

PEI System Memory Usage

- Architecturally defined PEI phase memory map
  - Goal is to keep simple allocation mechanism
- Memory Allocation HOB
  - Uses EFI_MEMORY_TYPE to type the memory
    - Runtime Memory: APStacks, ACPI / S3 Save / Restore memory, etc.
    - Boot Services: modules - DXE Core
- No architectural requirements on PEI system memory location & size
  - Implementation recommendation to put near physical top of memory (below 4GB for IA32)
  - Minimum size provided by Platform PEIM

Producer Phase Memory Map



PEI Hand Off to DXE

After the PEI Core dispatches all PEIMs, it transfers control to DXE

- Invoke DXE Initial Program Load (IPL) PPI to discover and dispatch the DXE Foundation and components
- The DXE IPL PPI passes the HOB list from PEI to the DXE Foundation

Agenda

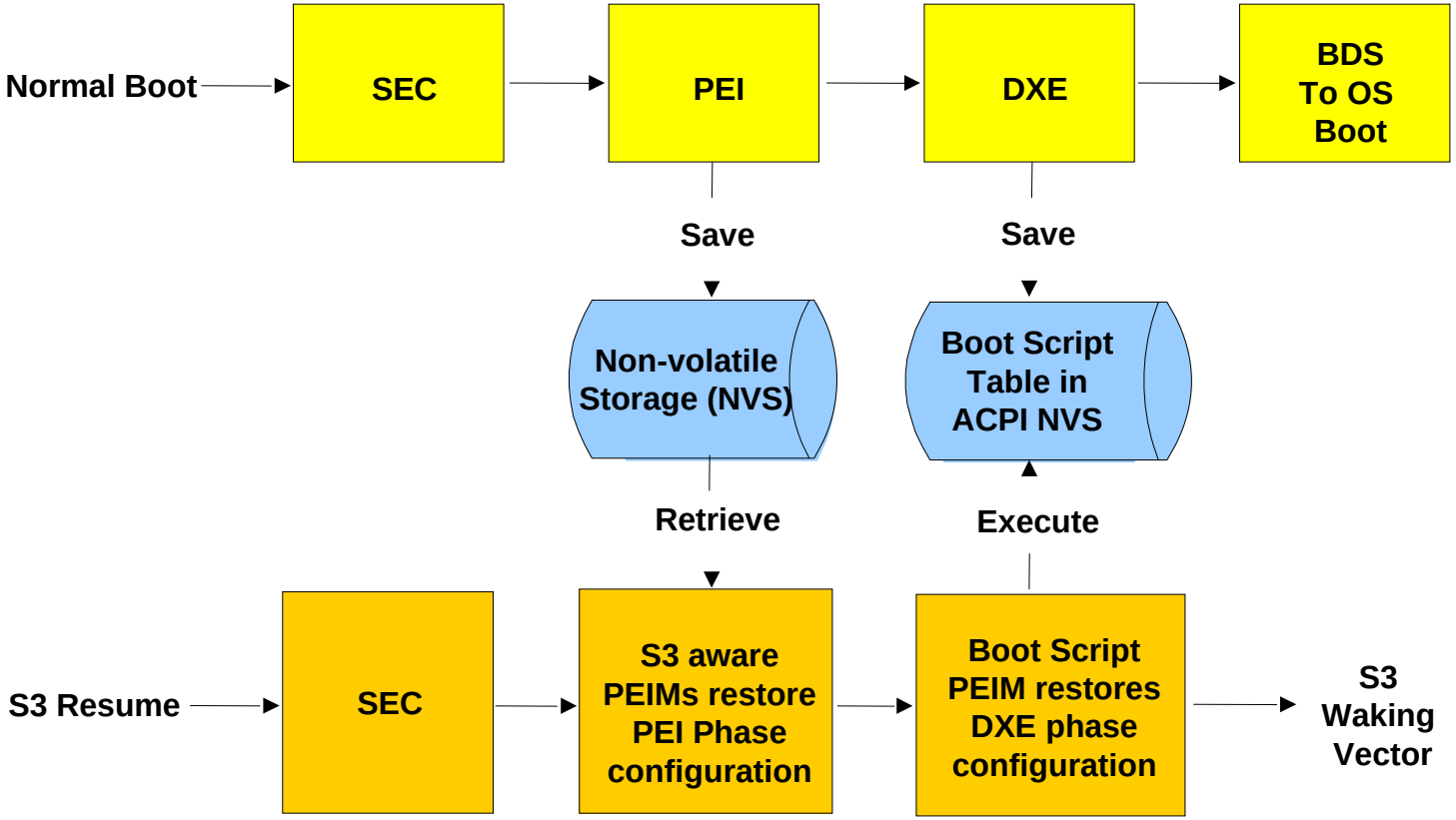
- PEI Overview
- PEI Core and PEI Services
- PEIM & PPI
- Dispatcher
- PEI Memory Environment
- PEI Hand-off Block (HOB)
- Boot Path
- PEI to DXE Transition

Boot Paths

- Boot Path is accessible to Each PEIM via boot service
  - Default Normal Mode
  - Includes e. g. S3, Recovery, Update
  - Each PEIM Must Do “The Right Thing”
  - Can Increase Priority of Boot Path By Returning Status Back From Call
  - “Force Recovery”

•See back up for Sleep state assumptions

S3 Boot Path



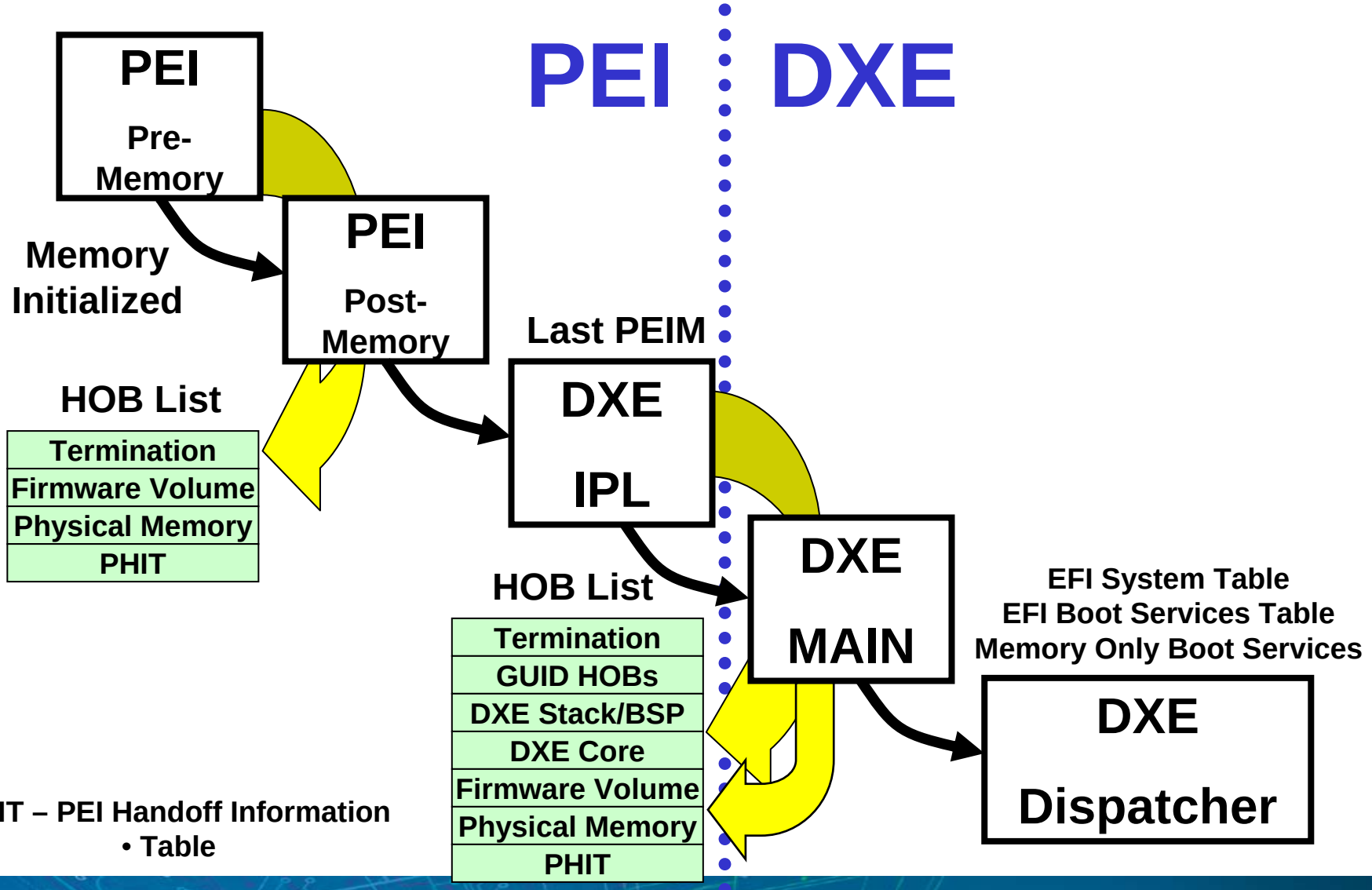
Recovery Support

- Loads a FV containing DXE from recovery media\ (net/disk/CD…..)
- Recovery may be initiated by any PEIM
  - A PEIM that read a recovery jumper
  - The PEIM sets the Boot Mode to “Recovery” via core service SetBootMode()
- Recovery may be initiated by the PEI Core
  - If a PEIM doesn’t validate, or
  - If PEI Core completes dispatch of all known modules and doesn’t get enough system memory to start DXE
- Recovery PEIMs are put in a fault tolerant block of the FV

Agenda

- PEI Overview
- PEI Core and PEI Services
- PEIM & PPI
- Dispatcher
- PEI Memory Environment
- PEI Hand-off Block (HOB)
- Boot Path
- PEI to DXE Transition

PEI to DXE Transition



•PHIT – PEI Handoff Information
• Table

DXE IPL

- No hard coded addresses allowed
- Find Largest Physical Memory HOB
  - Ideally this should be near Top Of Memory(TOM)
- Allocate DXE Stack from Top of Memory
  - 128KB for Stack (IA-32/IPF)
  - 16 KB for BSP (IPF Only)
  - Build HOB that describes DXE Stack
- Search FVs from HOB List for DXE Core
- Load DXE Core into Memory (PE/COFF)
  - Build HOB that describes DXE Core
- Switch Stacks and Handoff to DXE Core

Where the PEI call to DXE IPL Code is located

- Location in open source tree:
  - EDK I \Foundation\Core\Pei\PeiMain\PeiMain.c
  - EDK II \EdkModulePkg\Core\Pei\PeiMain\PeiMain.c
- **Call**: DxeIpl->Entry()

```
        EFI_STATUS
        EFIAPI
        PeiCore (
          IN EFI_PEI_STARTUP_DESCRIPTOR  *PeiStartupDescriptor,
          IN PEI_CORE_INSTANCE           *OldCoreData
          )
        {
        //      ...       ...       ...       ...
  DEBUG ((EFI_D_INFO, "DXE IPL Entry\n"));
  Status = TempPtr.DxeIpl->Entry ( TempPtr.DxeIpl, &PrivateData.PS,  PrivateData.HobList );
  ASSERT_EFI_ERROR (Status);  // IF code gets here then NO DXE
  return EFI_NOT_FOUND;
}
```

Where the PEI Transition Code is located

- Location in open source tree:
  - EDK I \Sample\Universal\DxeIpl\Pei\DxeLoad.c
  - EDK II \MdeModulePkg\Core\DxeIplPeim\DxeLoad.c
- **Call**: DxeLoadCore (inside the call DxeIpl->Entry())
  - EDK I - SwitchStacks Function call
  - EDK II - HandOffToDxeCore Function call

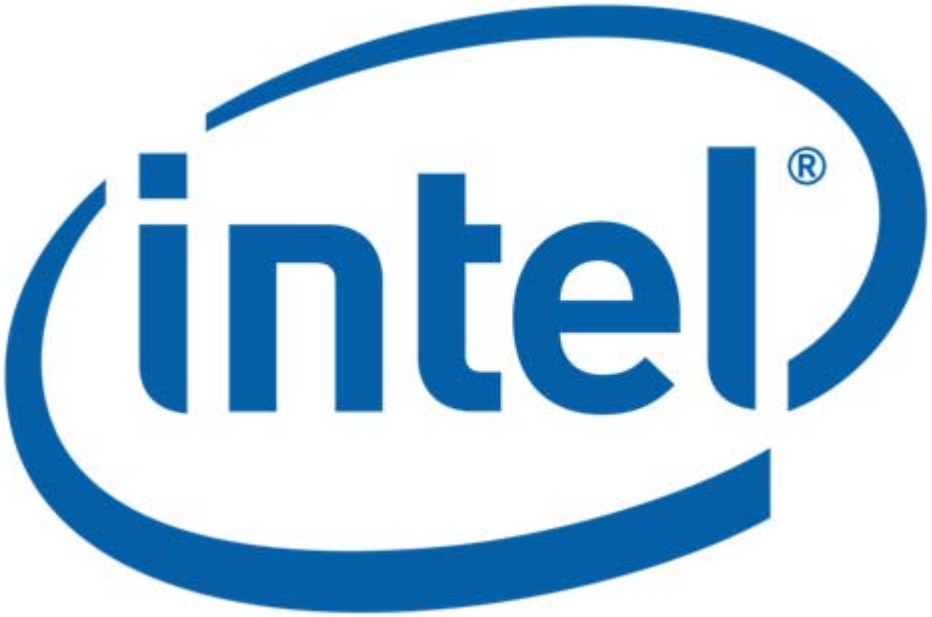
```
{ //  ------ EDK I  ------------------
 SwitchStacks (
    (VOID *) (UINTN) DxeCoreEntryPoint,
    (UINTN) (HobList.Raw),
    (VOID *) (UINTN) TopOfStack,
    (VOID *) (UINTN) BspStore
    );
}
```

```
{ //  ------ EDK II  ------------------
  // Transfer control to the DXE Core
  // The handoff state is simply a pointer to
  //     the HOB list

  HandOffToDxeCore (DxeCoreEntryPoint, HobList);

}
```

PEI Summary

- Modular Code without memory
- Easier Silicon Initialization
  - Reference code would just work
- Code from multiple vendors can coexist
- Modularity makes porting easier
- Complex Chipset initialization in C
  - Code easier to maintain
- Standard way to do S3 and Recovery

Q & A



Back up

PEI Architecture Goals

- Provide framework to do basic system initialization
  - Do the minimum configuration necessary to get to DXE …. Memory initialization is the main task
- Modular
  - Allow OEMs / IBVs / IHVs to supply separately built modules
- Interoperable
  - Discoverable interfaces, defined resource usage & calling conventions
- Updateable
  - Modules can be separately updated or added
  - Support for “Field Replaceable Units”
    - I.e. Flash on a processor / memory controller module

PEI Core (Continued)

- Minimal Input From Startup Code (SEC)
  - PEI Core validated
  - A verification interface
  - A small amount of temporary use RAM
- The PEI Core uses the stack to maintain private data kept visible to PEIMs thru PEI Services
- The PEI Core must switch stacks when real system memory is found
  - Must migrate private data and any pointers there-in
  - Installs PPI notification to all PEIMs that real memory has been found
- The PEI Core terminates when system memory has been found and all PEIMs have been dispatched.
  - Invokes DXE IPL PPI as last action.

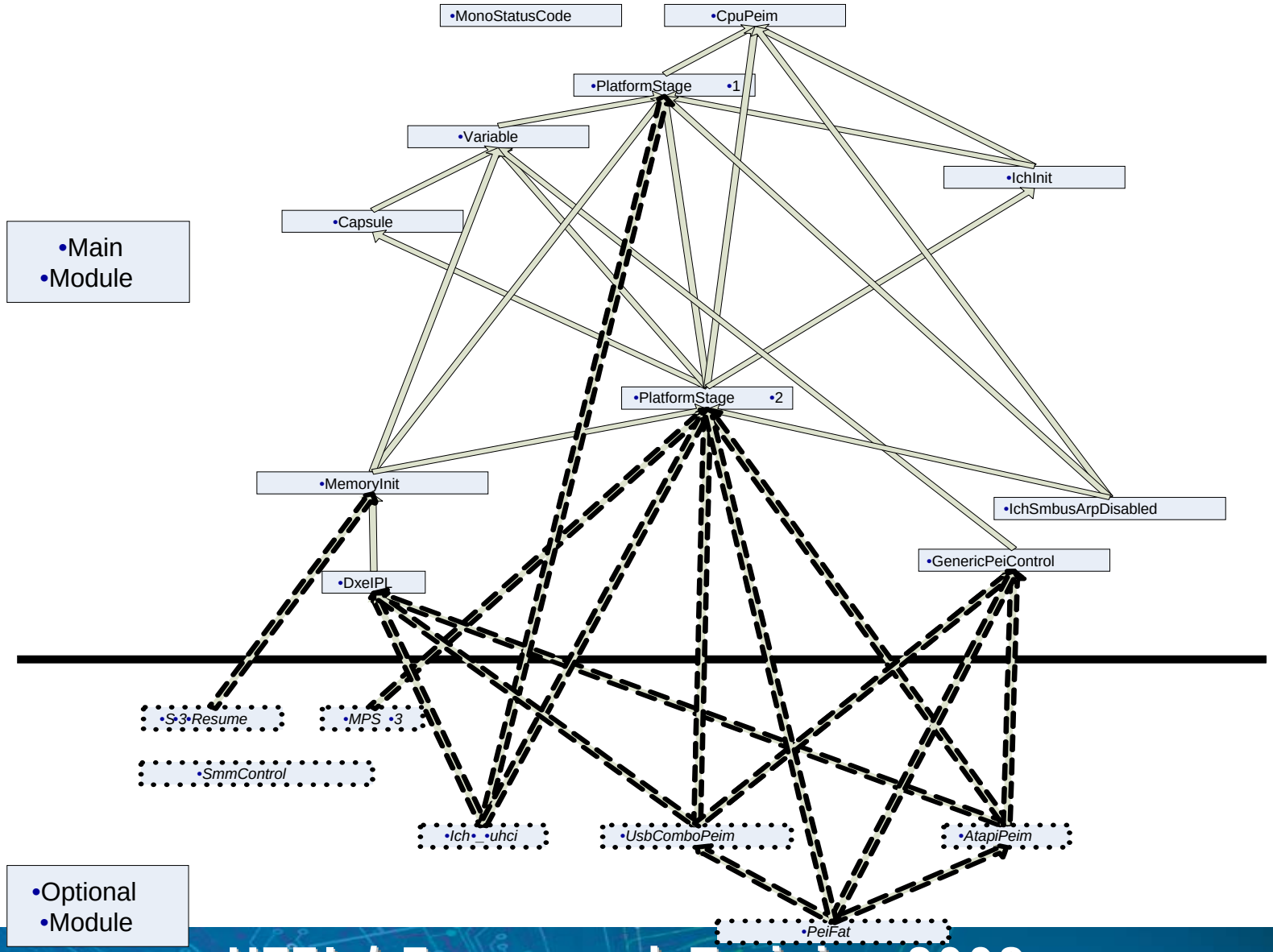
Notifications

- Provides a simple call back mechanism for PEIMs
  - Occurs on installation of a PPI
  - Helps resolve complex dependency problems
  - Can use a Null PPI to signal another PEIM
  - Registered with PEI Core with a Notify Descriptor
- Type 1 - Dispatch Level
  - Allows a PEIM to be re-started after another PEIM installs some PPIs and exits
  - Has available stack space just like a normal PEIM entry
- Type 2 - Call Back Level
  - Allow a PEIM to be called back immediately upon a PPI installation.
  - Notify function is called back on current stack

Architectural PPIs

- The architectural PPIs are interfaces that have standard definition
  - The PEIMs to publish these are IBV/Platform Builder…
- Standardization is to ensure that the PEI Core can have interoperability behavior across class of systems
- These PPIs include the following
  - Find FV
  - PEI Status Code
  - Memory discovered
  - Security
  - Decompression
  - PE32 Load Image

PEI Module Dependency



Sleep State Assumptions

System State	Description	Assumption
R0	Cold Boot	Cannot assume that the previously stored configuration data is valid.
R1	Warm Boot	May assume that the previously stored configuration data is valid.
S3	ACPI Save to RAM Resume	The previously stored configuration data is valid and RAM is valid. RAM configuration must be restored from nonvolatile storage (NVS) before RAM may be used. The firmware may only modify previously reserved RAM.
S4, S5	Save to Disk Resume, "Soft Off"	S4 and S5 are identical from a PEIM's point of view.
Boot on Flash Update		This boot mode can be either an INIT, S3, or other means by which to restart the machine.