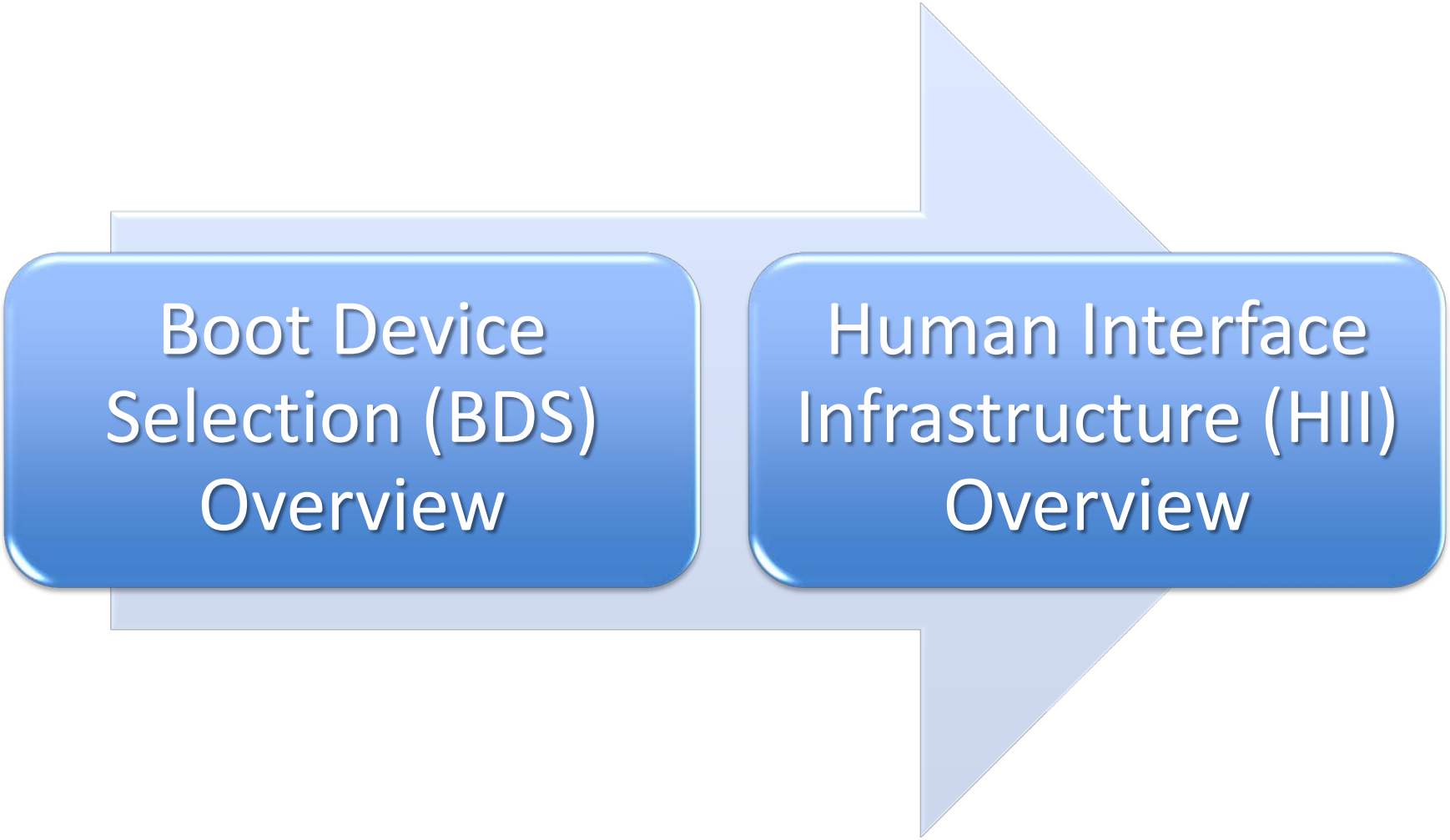
UEFI & EDK II Base Training

Boot Device Selection (BDS) Phase and the Human Interface Infrastructure (HII)

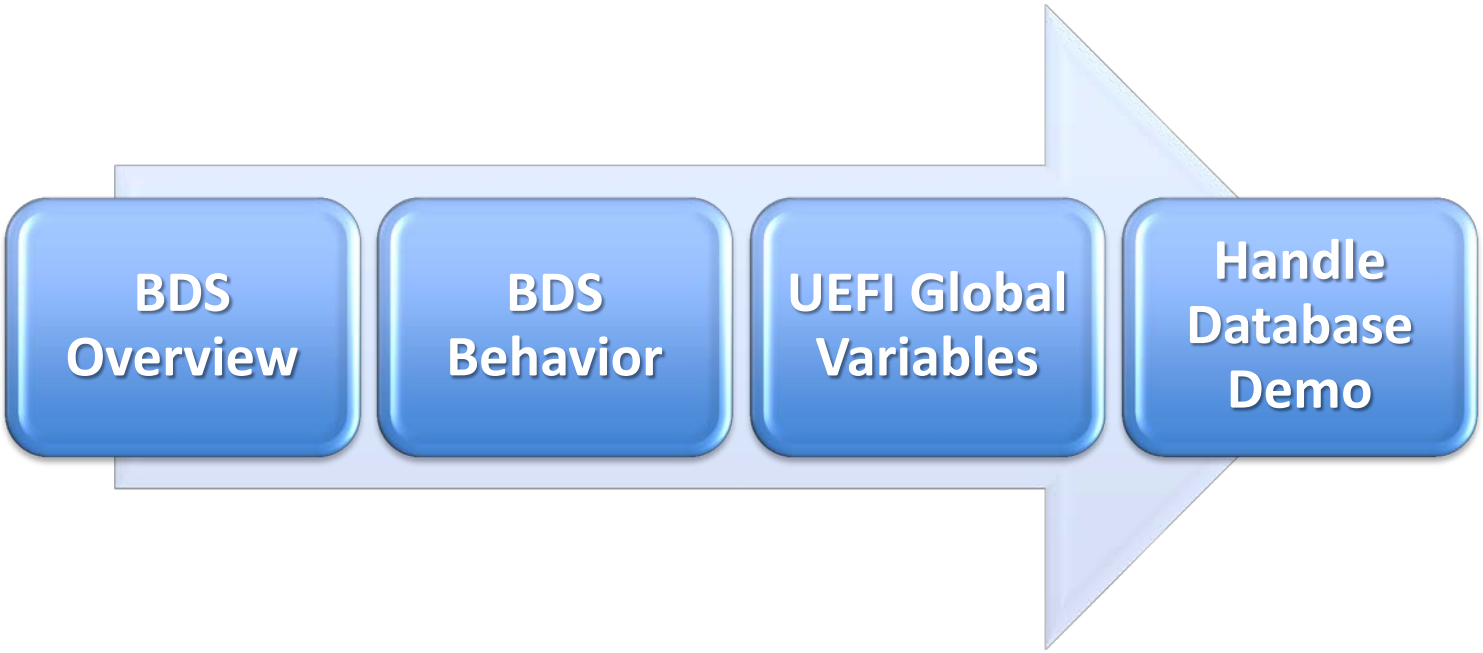
Intel Corporation

Software and Services Group

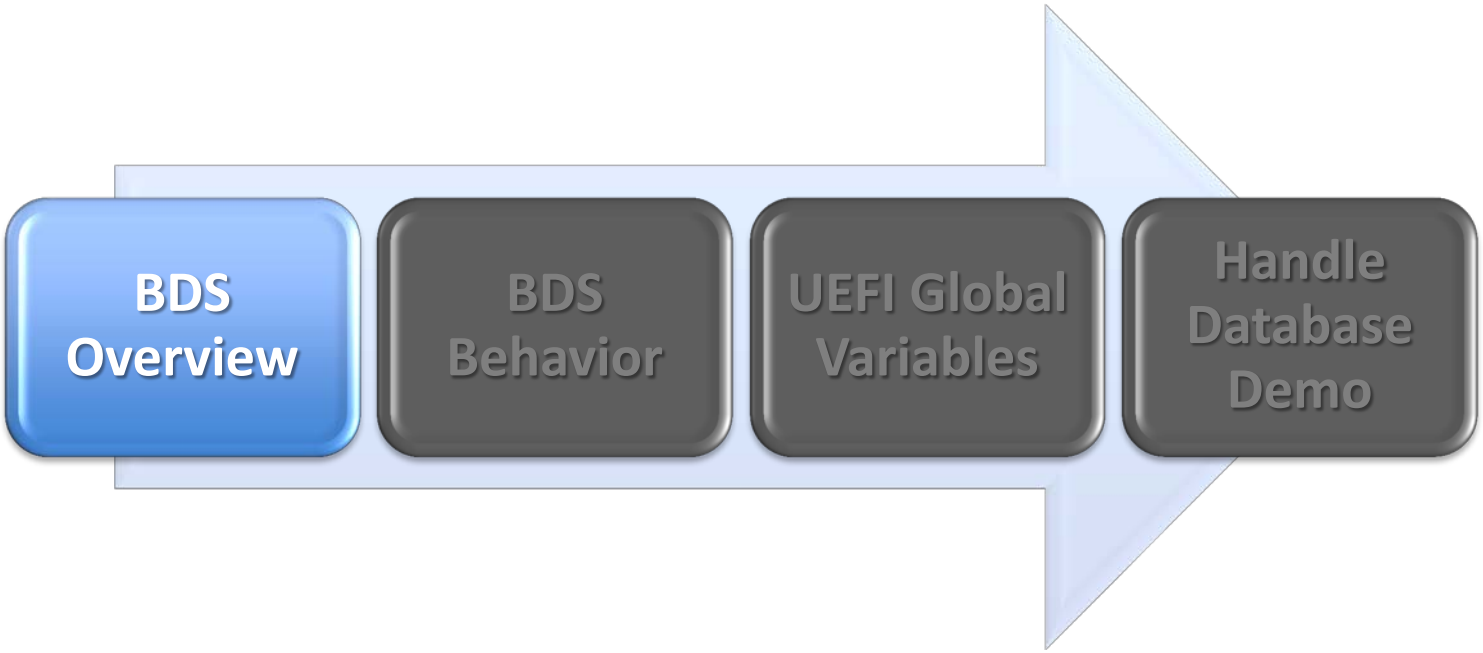
Agenda


Boot Device Selection (BDS) Overview
Human Interface Infrastructure (HII) Overview

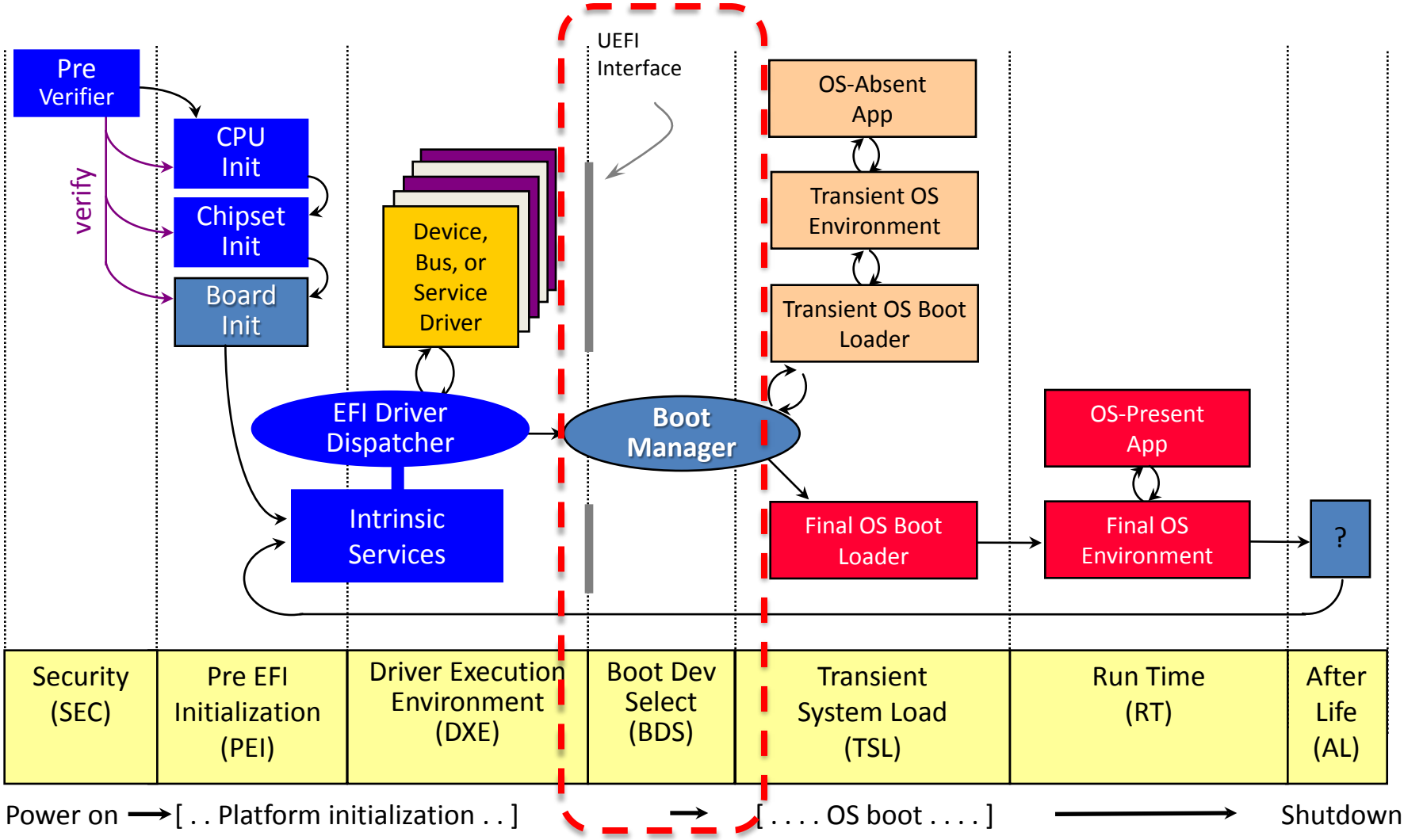
Agenda
- Boot Device Selection



Agenda
- Boot Device Selection



Architecture Execution Flow

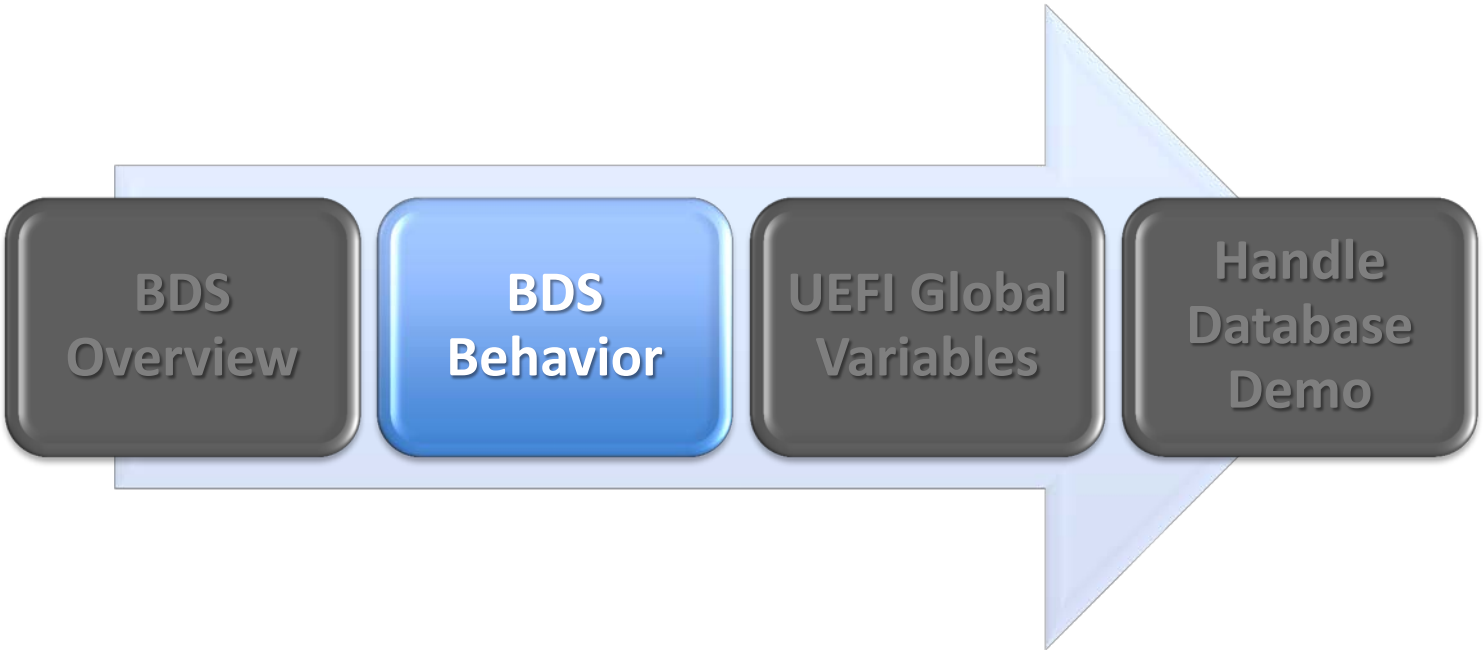


Boot Device Selection

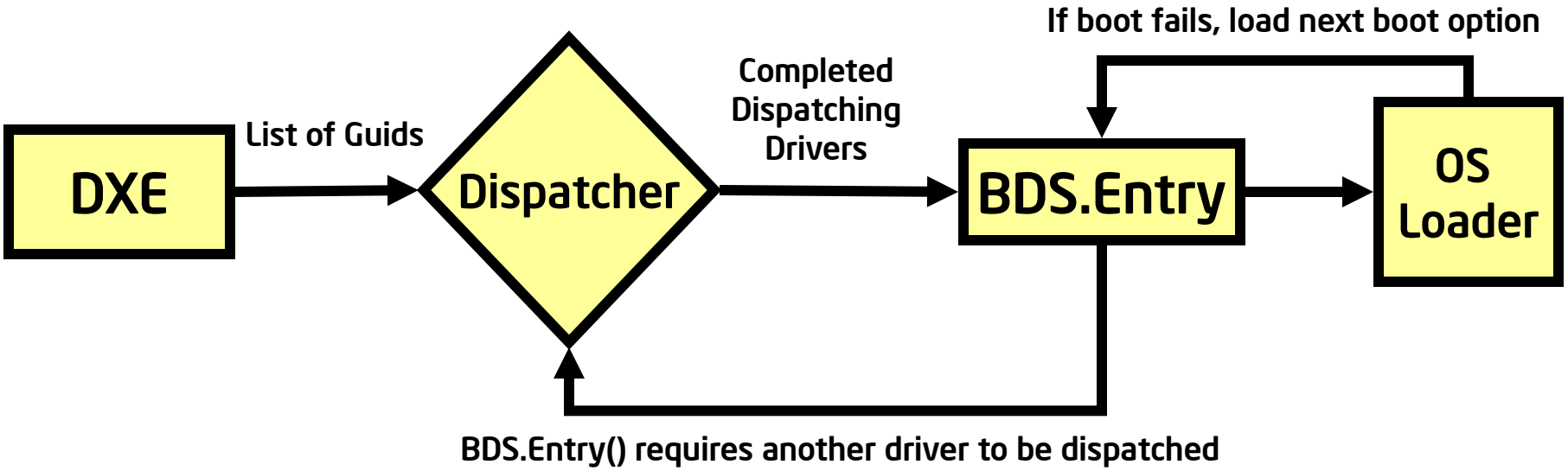
- Policy engine controls how the system will boot
- Takes control from DXE Foundation
- Attempts to pass control to boot target
- Arms watchdog to guard against boot failure
- Iterates list of possible boot targets
  - Drivers and boot targets stored in architectural environment variable lists
  - May need to return to DXE Foundation if more firmware volumes are encountered
- May present user interface and choices
  - Setup, boot list, boot list maintenance, IHV adapter configuration, diagnostics, recovery
  - OEM chooses what to expose and how to meet business requirements for the platform in given market

See § 3 of the UEFI 2.x Spec. (Boot Manager)

Agenda
- Boot Device Selection



DXE-Dispatcher-BDS Flow



BDS Steps

1. Initialize language and string database.
2. Get current boot mode.
   - The boot mode will determine the different policy executed in step 3-7
3. Build device list.
4. Connect devices.
5. Detect console devices.
6. Perform memory / diagnostic tests.
7. Process boot options.

Source Example: DXE Main Calls BDS Code

```
VOID
EFIAPI
DxeMain (
  IN  VOID *HobStart  // Pointer to the beginning of the
                      // HOB List from PEI
  )
{
  …    …    …    …   DXE init and DXE dispatcher  …    …    …

  // Transfer control to the BDS Architectural Protocol
  //
  gBds->Entry (gBds);
  // BDS should never return
  //
  ASSERT (FALSE);
  CpuDeadLoop ();
}
```

EDK I - \Foundation\Core\Dxe\DxeMain\DxeMain.c

EDK II - \MdeModulePkg\Core\Dxe\DxeMain\DxeMain.c

Call - DxeMain()

Source Example: Locate BDS Entry Point

```
/**
Service routine for BdsInstance->Entry().
Devices are connected, the consoles are
initialized, and the boot options are tried.
**/
VOID EFIAPI
BdsEntry (
  IN EFI_BDS_ARCH_PROTOCOL  *This
  )
{  // . . .
// Initialize the global system boot option
  InitializeListHead (&DriverOptionList);
  InitializeListHead (&BootOptionList);
// . . .
// Validate Variable.
  BdsFormalizeEfiGlobalVariable();
// Do the platform init, can be
// customized by OEM/IBV
  PlatformBdsInit ();
// . . .
// Initialize the platform specific string &
//    language
  InitializeStringSupport ();
// . . .
```

```
// Set up the device list based on
// EFI 1.1 variables
  BdsLibBuildOptionFromVar
    (&DriverOptionList,
     L"DriverOrder");
// . . .
// Check if we have the boot next option
  mBootNext = BdsLibGetVariableAndSize (
                L"BootNext",
                &gEfiGlobalVariableGuid,
                &BootNextSize
                );
// Setup some platform policy here
  PlatformBdsPolicyBehavior
    (&DriverOptionList,
      &BootOptionList,
      BdsProcessCapsules, BdsMemoryTest);
// BDS select the boot device to load OS
  BdsBootDeviceSelect ();
// Only assert here, we should never
// return back to DxeCore.
  ASSERT (FALSE);
  return ;
}
```

EDK I - \Sample\Platform\Generic\Dxe\UefiPlatformBds\BdsEntry.c
EDK II - \IntelFrameworkModulePkg\Universal\BdsDxe\BdsEntry.c
Call - BdsEntry()

Source Example: Locate BDS Function

```
/**
// Boot for the boot order specified
// by platform policy.
**/
VOID BdsBootDeviceSelect (
  VOID
  )
{
  EFI_STATUS        Status;
// . . .
  // Got the latest boot option
  //
  InitializeListHead (&BootLists);
  // First check the boot next option
  if (mBootNext != NULL) {
    // Indicate we have the boot next
    //variable,
     BootNextExist = TRUE;
    // Clear the this variable
    gRT->SetVariable (
          L"BootNext",
          &gEfiGlobalVariableGuid,
           . . .
          );
```

```
  // Add the boot next boot option
   UnicodeSPrint (Buffer, sizeof (Buffer),
           L"Boot%04x", *mBootNext);
   BootOption = BdsLibVariableToOption
           (&BootLists, Buffer);
   // If fail to get boot option from
   // variable, just return and do nothing.
   if (BootOption == NULL) {
     return;
   }
   BootOption->BootCurrent = *mBootNext;
}
// Parse the boot order to get boot option
BdsLibBuildOptionFromVar (&BootLists,
      L"BootOrder");
```

EDK II - \IntelFrameworkModulePkg\Universal\BdsDxe\BdsEntry.c

Source Example: Locate BDS Function – Cont.

```
// When we didn't have chance to build boot
// option variables in the first full
// configuration boot, then
// we have no boot options.
// Give the last chance to enumerate the boot
//options.

  if (IsListEmpty (&BootLists)) {
    BdsLibEnumerateAllBootOption
    (&BootLists);
  }
 // Here we make the boot in a loop,
 // every boot success will
 // return to the setup page
  for (;;) {
    // Check the boot option list first
    if (Link == &BootLists) {
      // There are two ways to enter here:
      // 1. There is no active boot option,
      //   give user chance to add new boot
      //   option
      // 2. All the active boot option
      //   processed, and there is no
      //   one is success to boot, then we
      //   back here to allow user
      //   add new active boot option
      // . . .
    }
```

```
  // Get the boot option from the link
  // list
   BootOption = CR (Link,
     BDS_COMMON_OPTION,
     Link, BDS_LOAD_OPTION_SIGNATURE);
  // Check if LOAD_OPTION_ACTIVE,
  . . .
  // All the driver options should have
  // been processed since now boot will be
  // performed.
     Status = BdsLibBootViaBootOption
         (BootOption,
          BootOption->DevicePath,
          &ExitDataSize, &ExitData);
     if (Status != EFI_SUCCESS) {
      // Call platform action is boot fail
      PlatformBdsBootFail (BootOption,
          Status, ExitData, ExitDataSize);
       // Check the next boot option
       Link = Link->ForwardLink;
     } else {
       // Call platform action to
       //indicate the boot success
. . .}
} //end for
```

Example: "BIOS Setup" Menu (PlatformBdsPolicyBehavior)

Boot Option

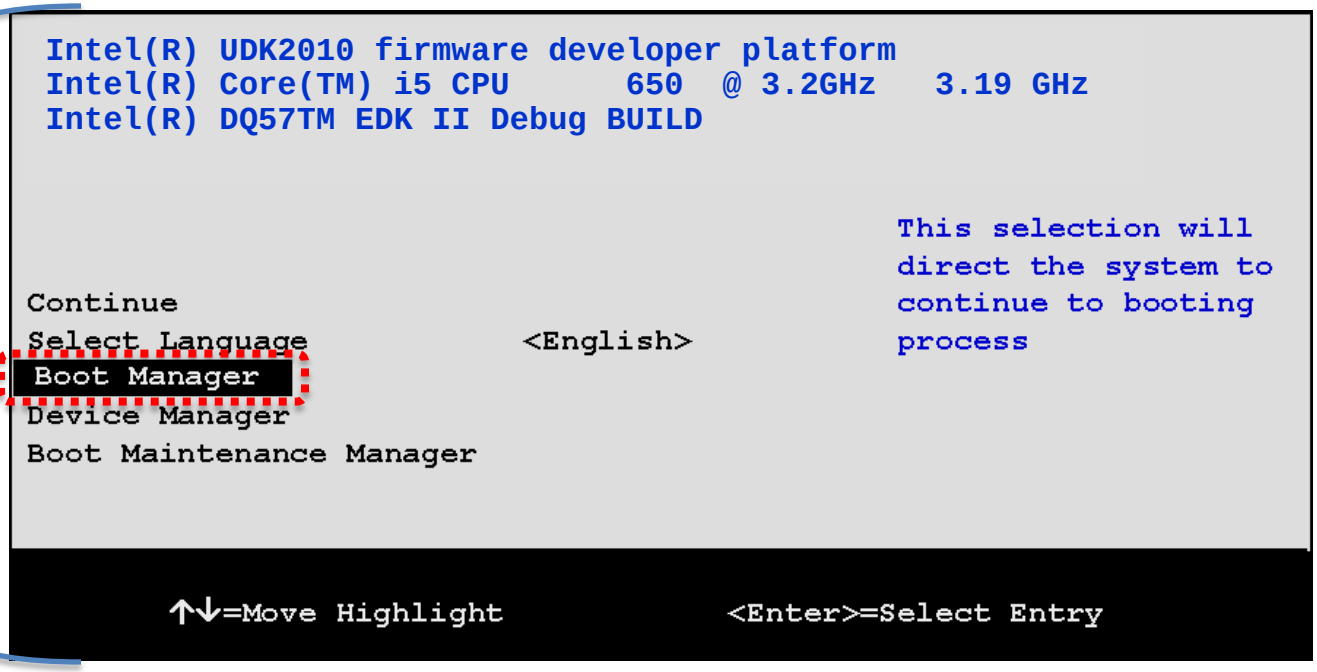
- BDS enumerates all possible boot devices in the system and create their boot option variables
- Current BDS will connect all devices and do this enumeration when user interrupts auto boot
  - Boot Manager & Device Manager
  - Boot Maintenance Manager
- Current BDS has two steps to enumerate the boot option
  - Legacy boot option for legacy boot (CSM)
  - UEFI boot option for UEFI boot

See *BDS Behavior and Suggestions* Document on CD

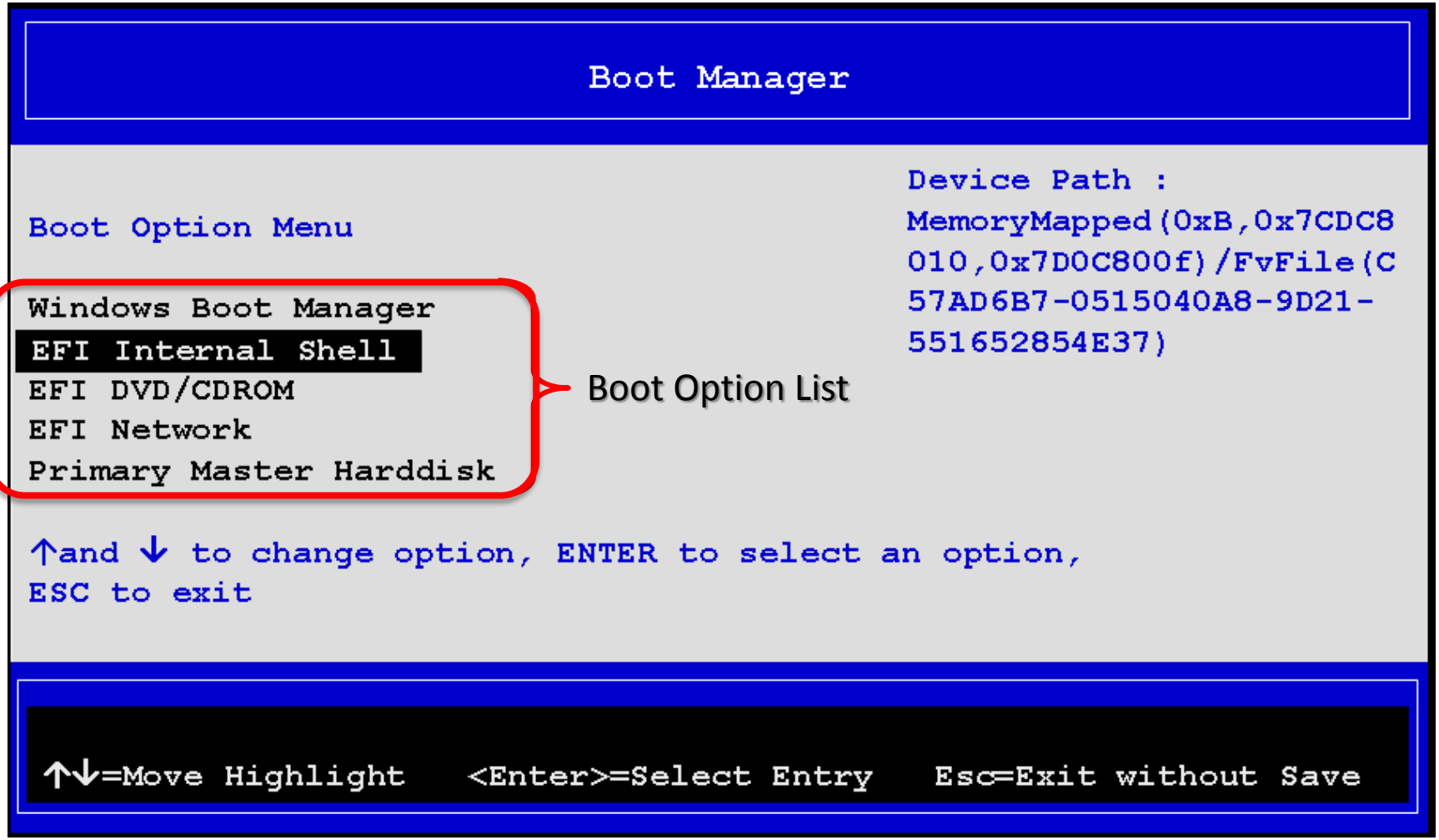
Sample Boot Manager from EDK

- Functionally replaces legacy BIOS Boot Specification (BBS)
- Order of processing load options
  - Driver Order Options - Load any drivers specified in driver option list
  - Check Boot Next Feature - This feature is for operating system setup; so that, on the next boot, this option is selected once and then removed from the list
  - Boot Option List - Options stored in NVRAM with boot maintenance menu

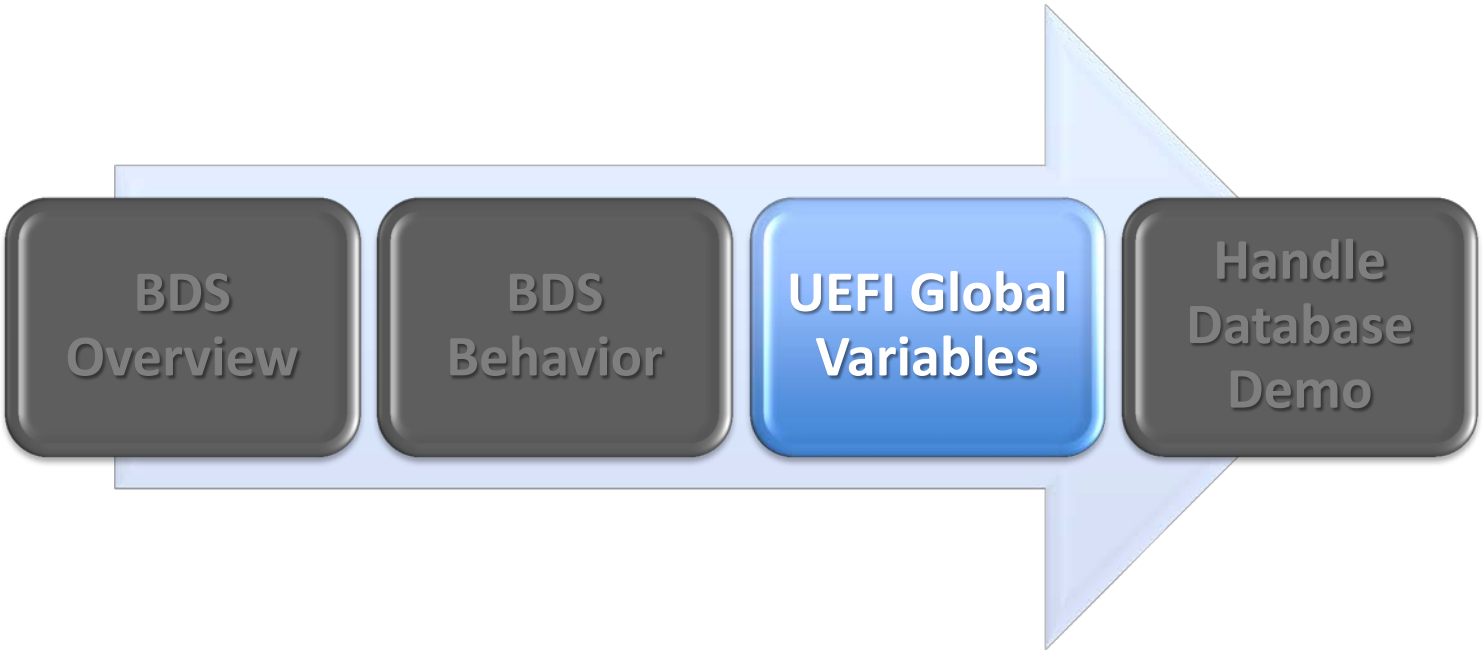
Example: "BIOS Setup" Menu (*PlatformBdsPolicyBehavior*)



Sample Boot Manager from EDK

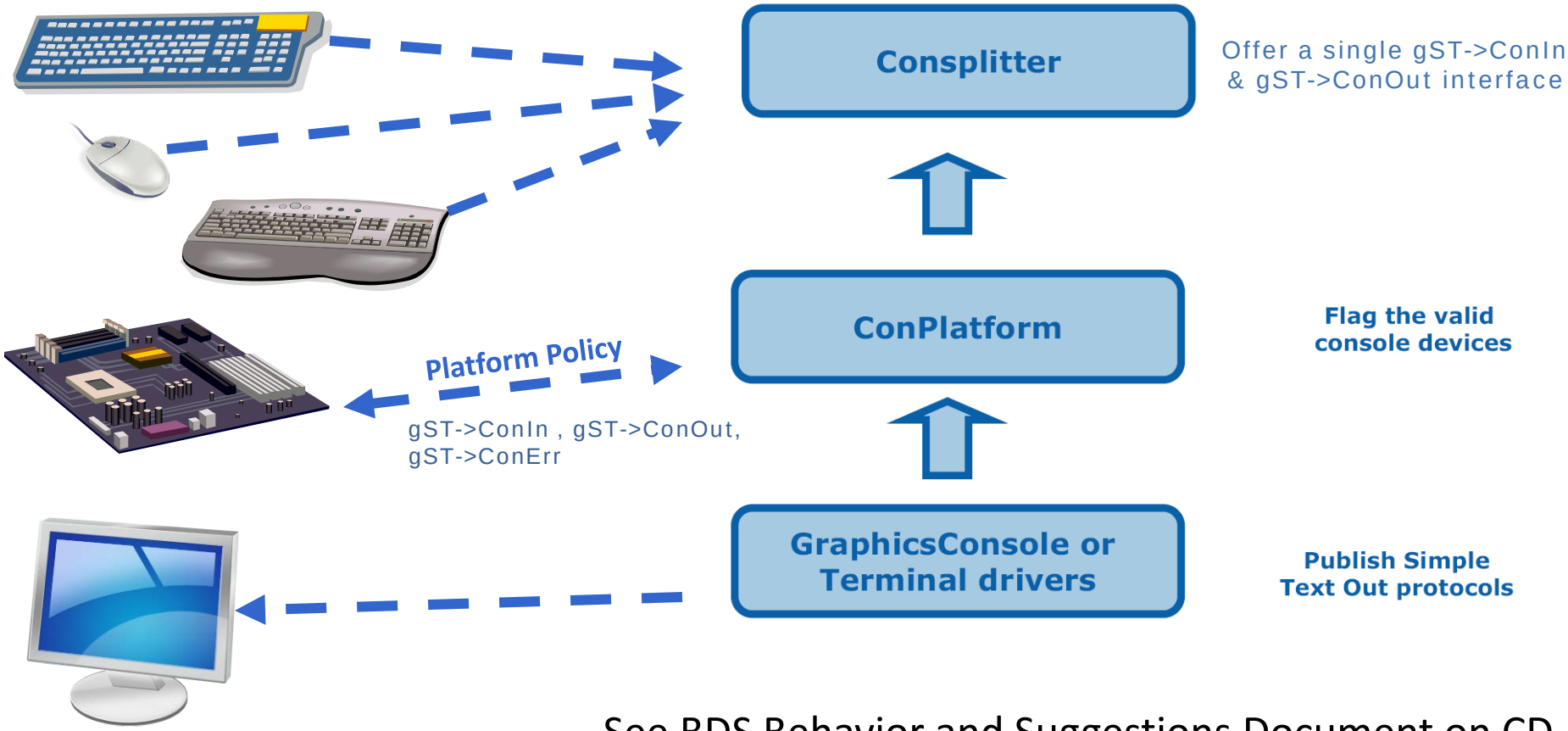


Agenda
- Boot Device Selection



Console Device

Driver stack overview for console output case



See BDS Behavior and Suggestions Document on CD

BDS Policy Input

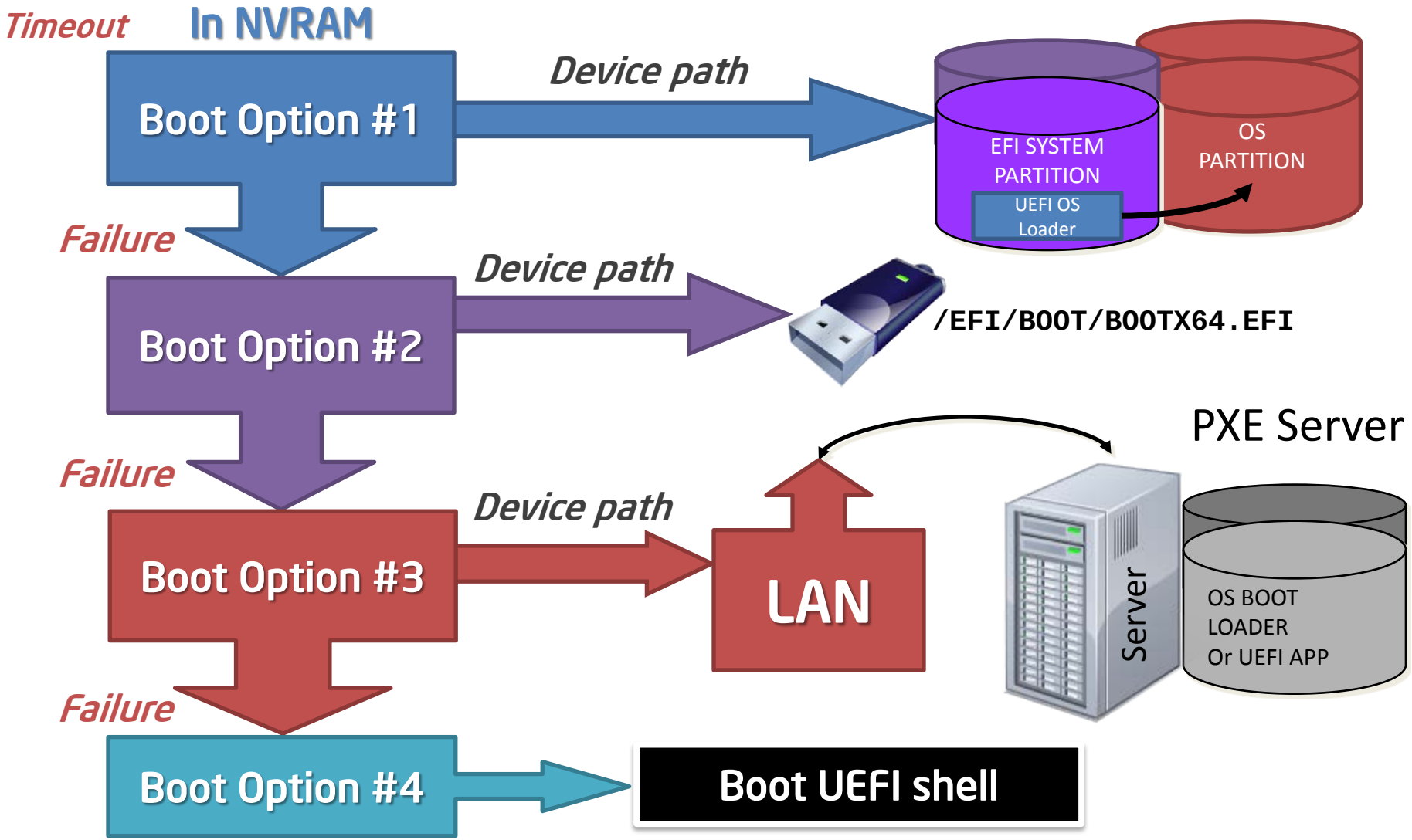
Globally Defined Variables

- **`ConIn`** The device path of the default input device.
- **`ConOut`** The device path of the default output device.
- **`ErrOut`** The device path of the default error output device.

BDS will fill in the corresponding system table entries with the handle of the device that the variables are pointing to.

See § 3.2 of the UEFI 2.x Spec.

Boot Option List Example



BDS Boot Policy

Globally Defined Variables

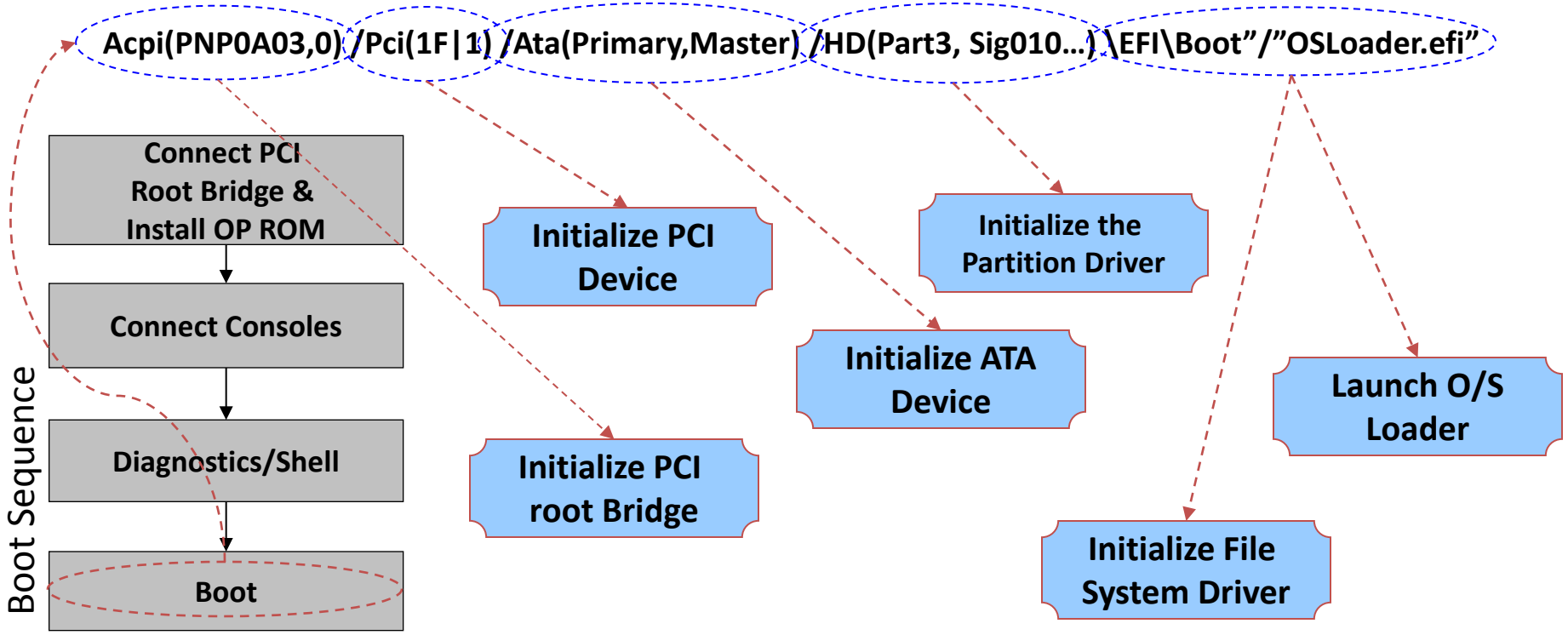
- **BootOrder** A list of unsigned integer values that make up an ordered list of the Boot#### variable. The BootOrder is not an optional variable.

- **BootNext** The Boot option for the next boot only. This option takes precedence over the Boot#### variable.

- **Boot####** A boot load option. BDS will attempt to load the boot driver specified
  - –Boot#### variable contains an EFI_LOAD_OPTION.
  - –example would be an OS loader or Setup

Why use the UEFI Device Path?

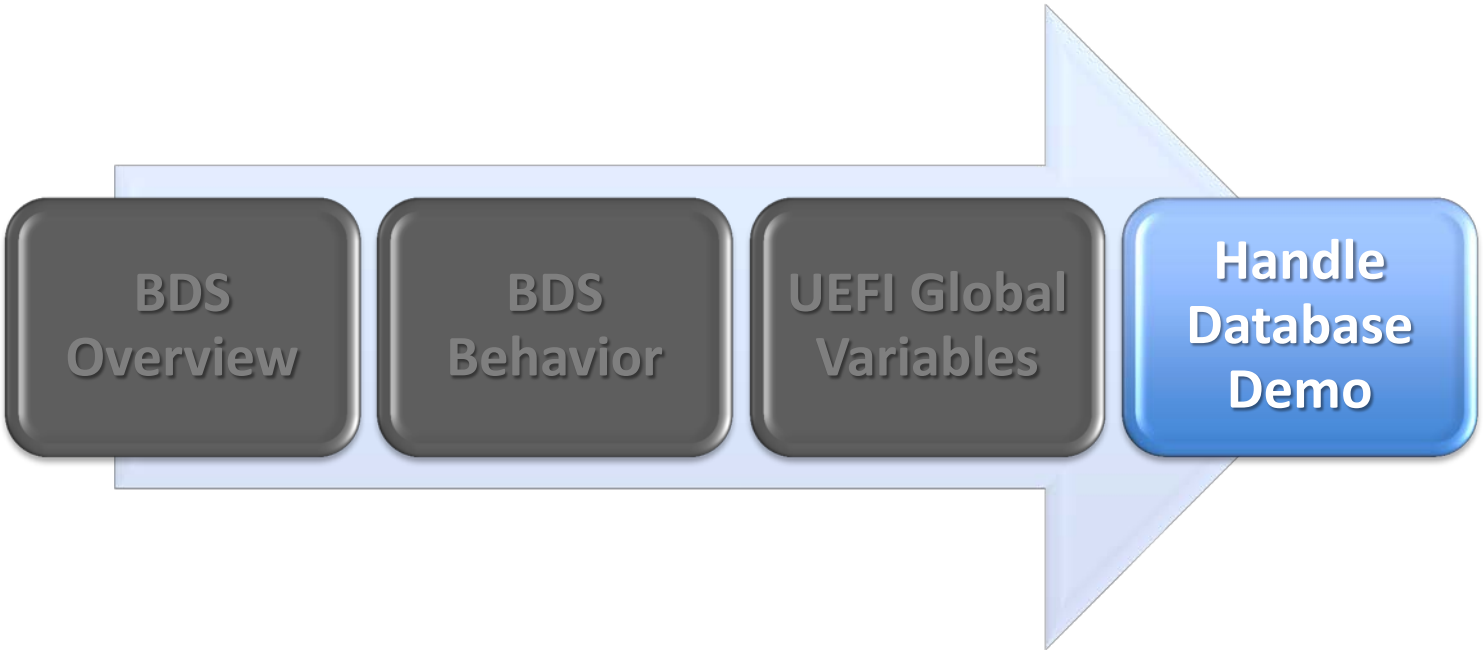
- An UEFI Device Path describes a *boot target*
  - Binary description of the physical location of a specific target



Boot Devices Selection Summary

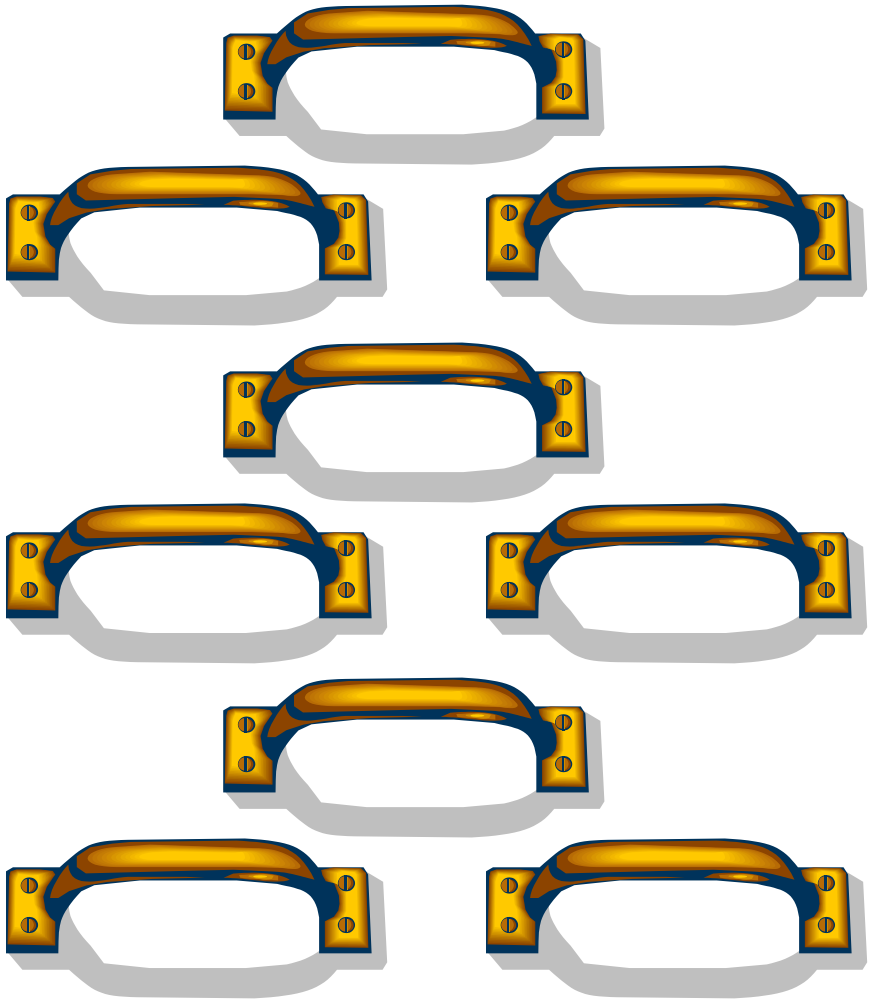
- DXE already put UEFI driver images in memory
  - *Quiescent state*: entry point doesn't touch hardware
- Boot devices described by UEFI Device Path
  - Path list of software visible components, supported with UEFI drivers between host bus and device
- Initializes console devices
  - Various output devices, keyboard and mouse
  - "connect" on UEFI drivers specified by device path
- Initializes boot devices
  - Mass storage or Network / "connect" on required UEFI drivers
- Proceeds to pass control to OS loader
  - Iterating list of possibilities if required

Agenda
- Boot Device Selection

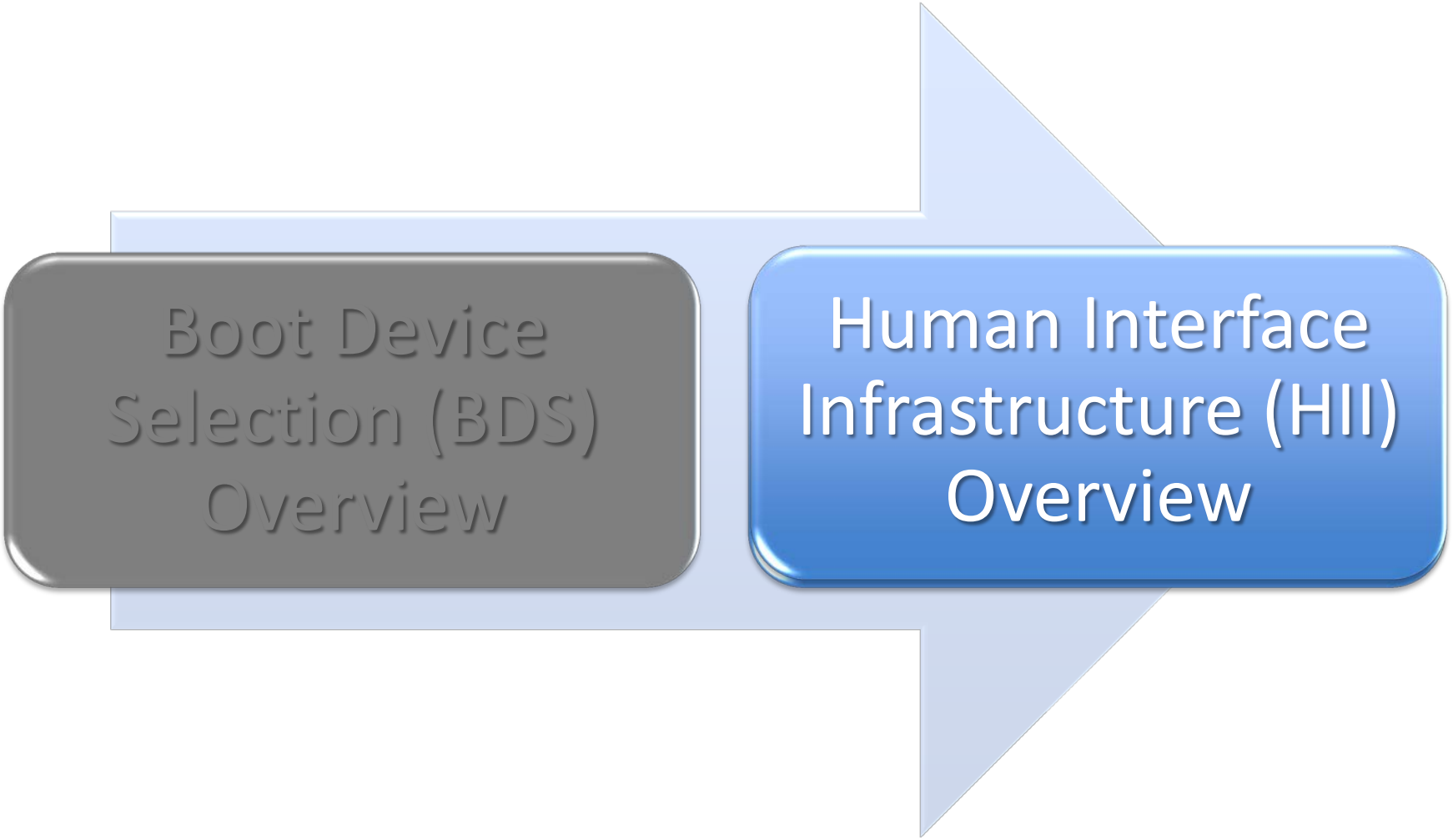


Demo - Handle Database –UEFI Variables

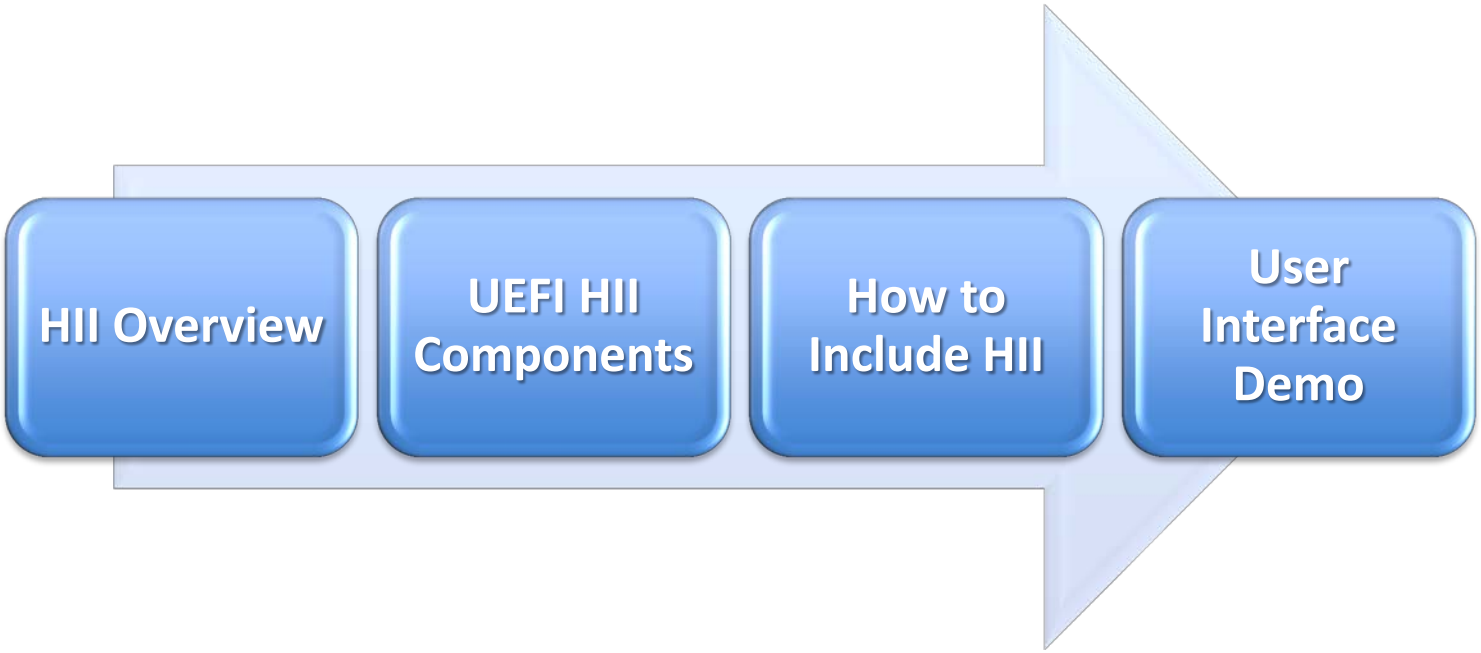
Intel® DQ57TM
Mother Board



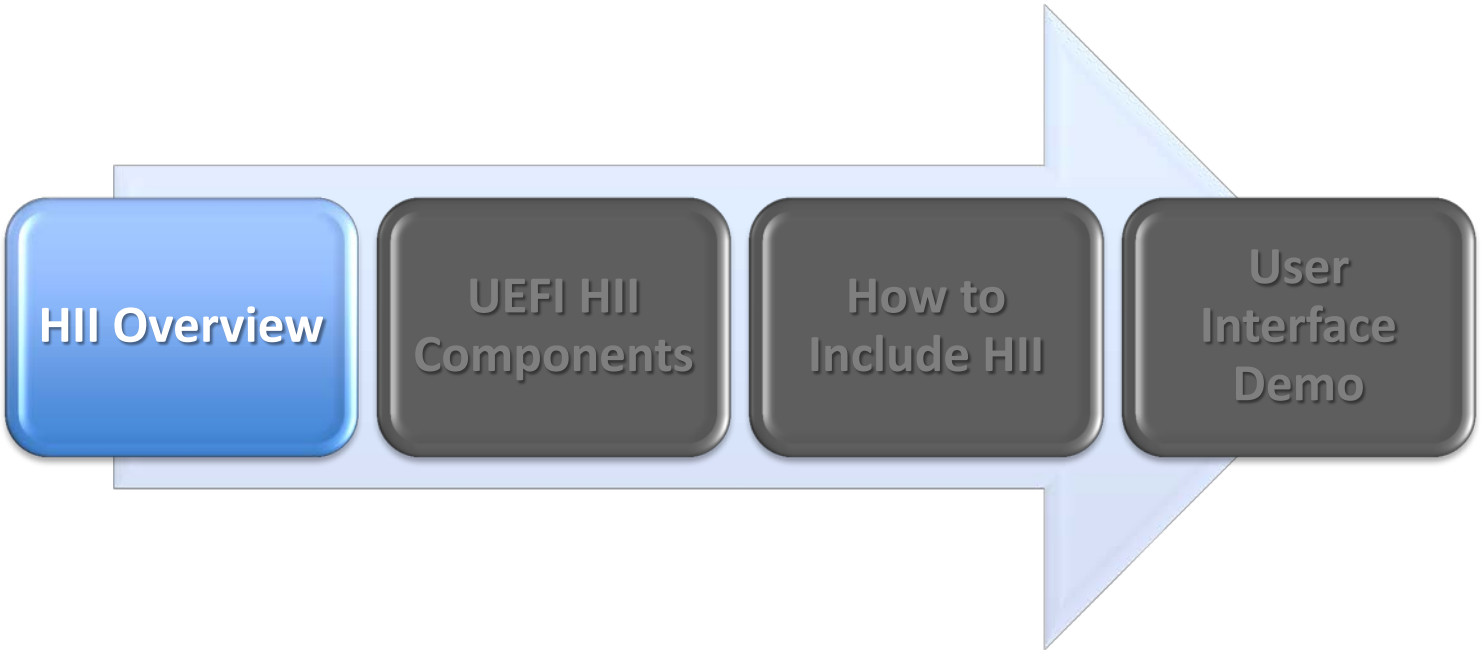
Agenda


Boot Device Selection (BDS) Overview
Human Interface Infrastructure (HII) Overview

Agenda
- Human Interface Infrastructure (HII)



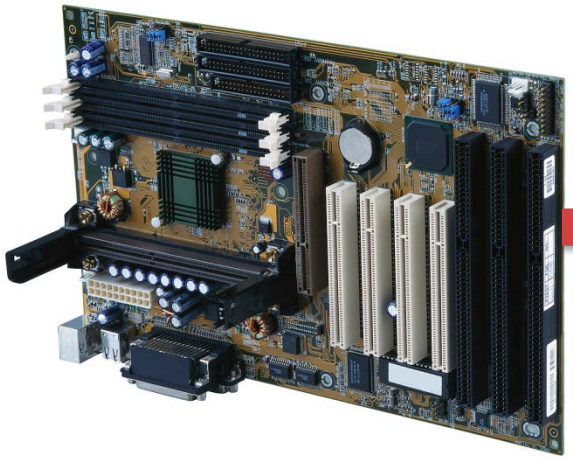
Agenda
- Human Interface Infrastructure (HII)



HII: Key Concepts

- Solve problems from legacy BIOS …
  - Different menus for BIOS setup & OpROM
  - User has problems finding the right menu
  - OEMs need a consistent user interface
- UEFI Human Interface Infrastructure (HII)
  - System firmware has a common setup browser
    - Drivers don't carry their own UI
    - Single point for pre-OS setup interface
  - Firmware & Drivers publish to a "database"

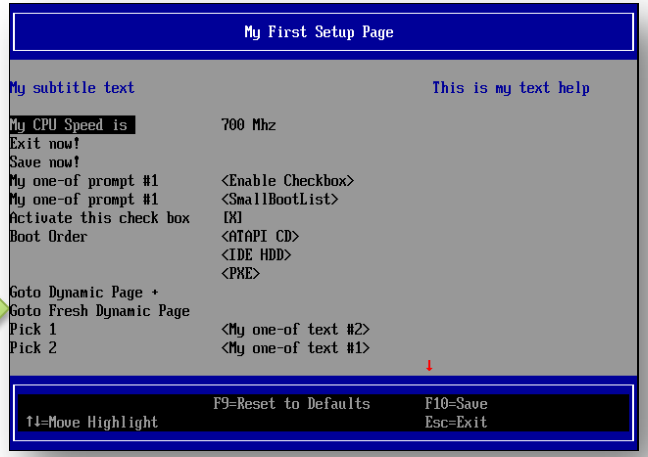
HII: Key Concepts



forms & strings

localization

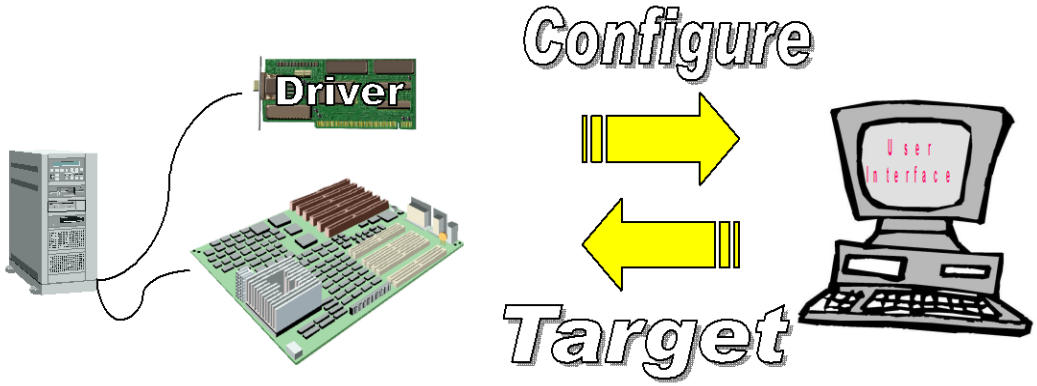
HII

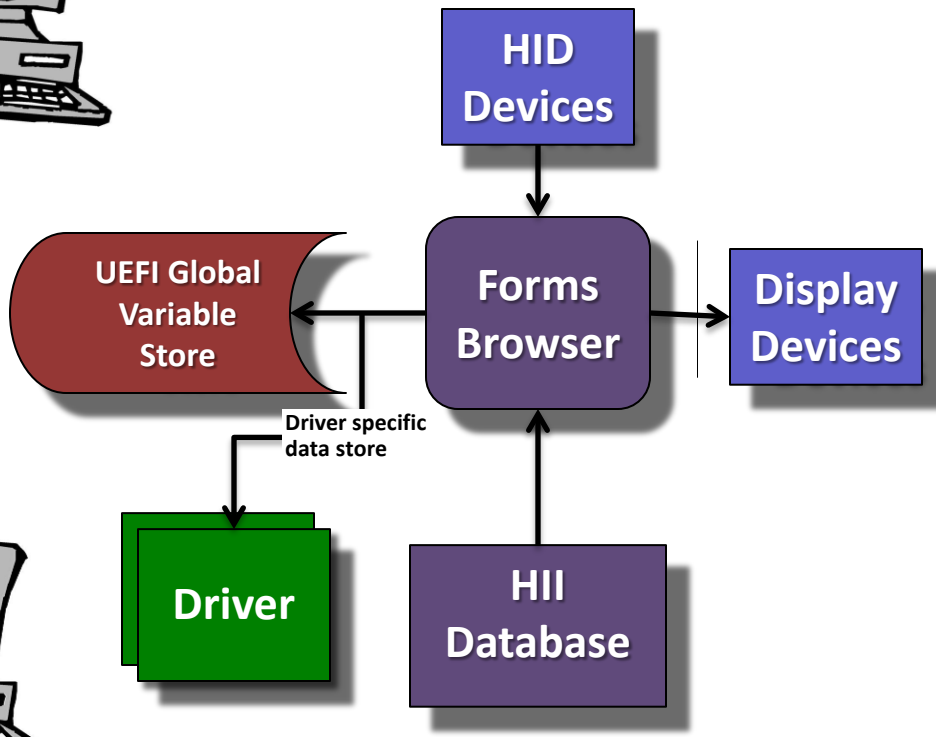


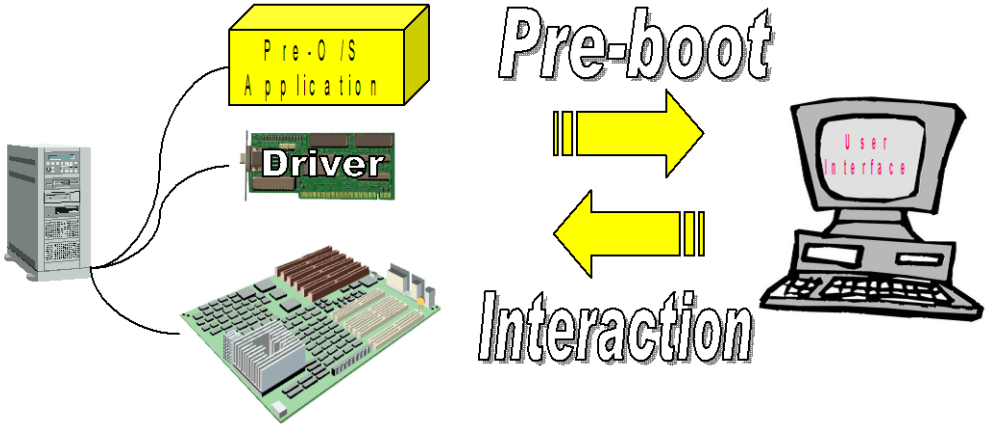
setup browser

input sources

Design Discussions





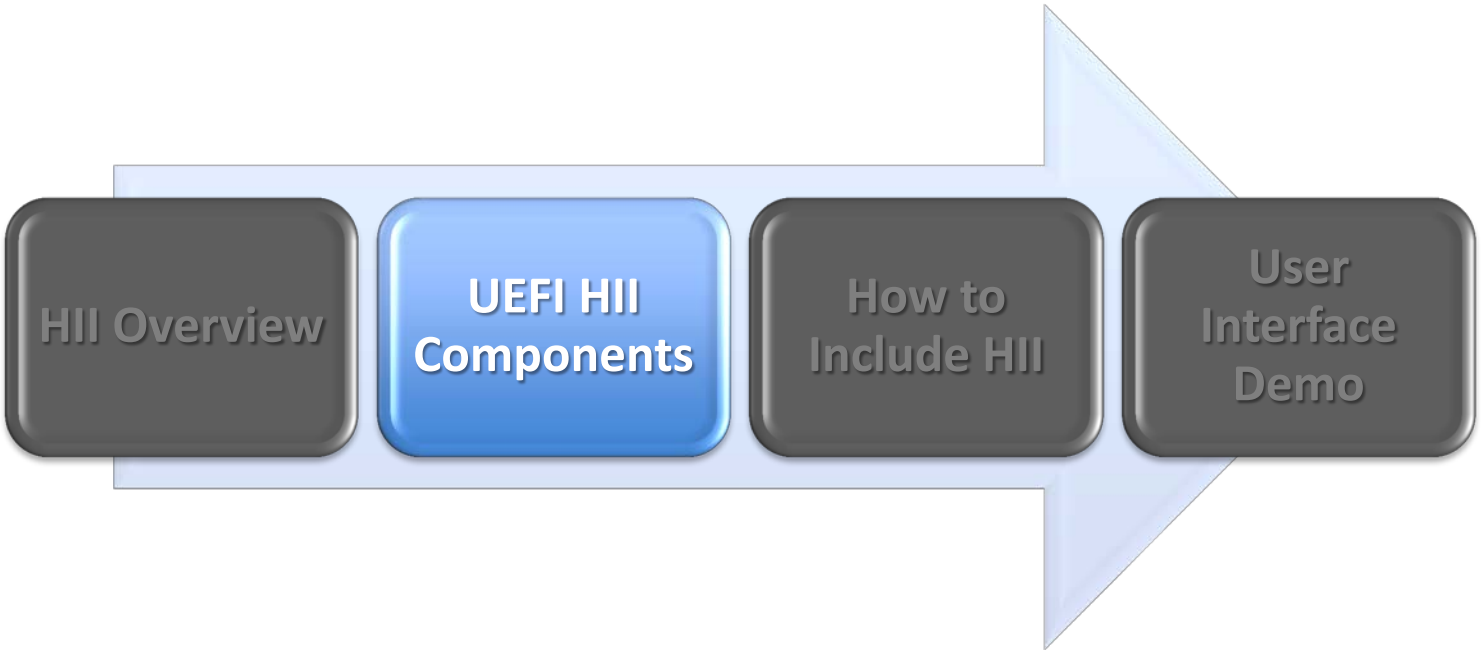


See § 28.2 of the UEFI 2.x Spec.

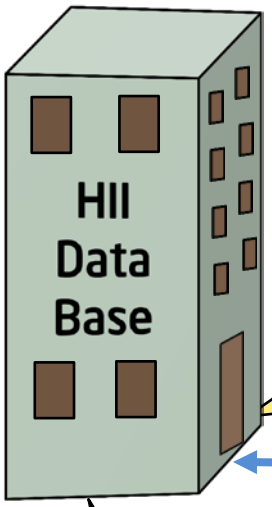
Why Human Interface Infrastructure (HII)

- Single window for full platform configuration
- Avoid multiple hotkeys
- Localization support
- Unified look and feel
- If driver supports any configuration, must be implemented using HII
- Expose content to be seamlessly integrated in platform solutions

Agenda
- Human Interface Infrastructure (HII)



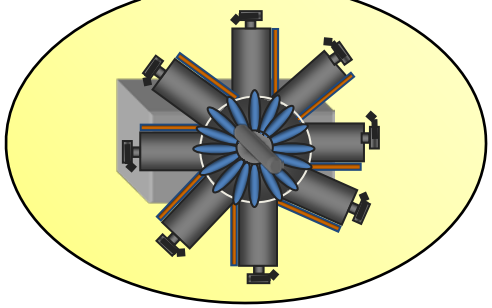
UEFI Specification: HII



Data

Fonts, Strings, Image, Forms
GUID, Keyboard Layout, Device Paths

NVRAM



HII Browser Engine Protocols

- Configuration Routing Protocol
- Configuration Access Protocol
- Form Browser 2 Protocol

HII Protocols

- Font Protocol
- String Protocol
- Image Protocol
- Database Protocol

See § 28-31 of the UEFI 2.3x Spec.

Human Interface Components

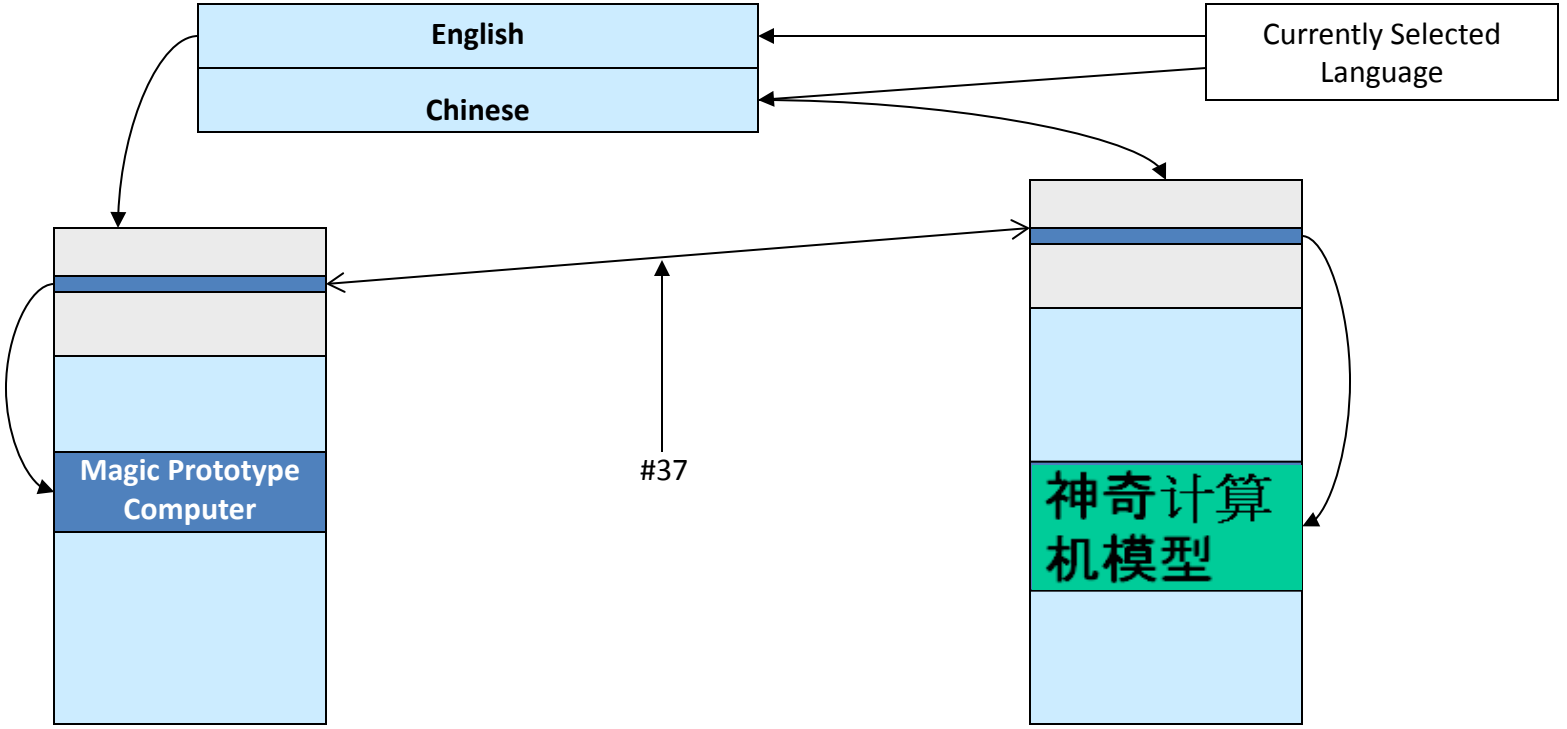
- **Strings** – Unicode representation
- **Fonts** – Bitmap fonts for easier localization
- **Keyboard** – Keyboard Mapping
- **Forms** – Describes UI layout for 'windowing' interfaces
  – An application that uses String and Font support
- **Package** – Self supporting data structure containing fonts, strings, and forms from a driver or set of drivers

Strings

- Strings stored in Unicode
  - Already the text standard in UEFI today
- Localization happens at the string level
  - Caller externs and passes in language independent string token
  - String support determines actual string from token and selected language
  - Usage Model:
    - String library supporting translations
    - Tools to extract strings depending on use by driver
    - Analysis of strings used to extract fonts

Token to String Mapping

- Request: *Print string with token 37*
- Currently selected language is as in UEFI 2.x. This is used to select between language data structures. The structures indicate which language(s) they support.
- The top part of the structure maps from token to string. The bottom part of the structure is the string content.



Font Management

- One Standard Font for UEFI
  - One font database accumulated during boot
- Each Component Provides Its Fonts
  - System provides ASCII and ISO Latin-1
  - Fonts only required for characters in actual strings
    - If the firmware will never print “tractor” in Kanji, discard the bit image
  - Result is a sparse array of characters indexed by the Unicode ‘weight’
- Wide and Narrow glyphs supported

Keyboards

- Support keyboards independent of language
  - Ex: UK English keyboard layout versus US English
  - Adding support of other modifiers (e.g. Alt-GR, Dead-keys, etc)
- Keyboard Layout
  - Allows for a standardized mechanism to describe a keyboard layout and add to system database
  - Allows for switching keyboard layouts

Spanish

English

French

Forms

- Forms are stored in the HII database, along with the strings, fonts and images
- Other applications may use the information within the forms to validate configuration setting values
- The Forms Browser provides a forms-based user interface which understands
  - how to read the contents of the forms
  - interact with the user
  - save the resulting values
- The Forms Browser uses forms data installed by an application or driver during initialization in the HII database

See § 28.2.5 of the UEFI 2.x Spec.

Visual Forms Representation (VFR)

- Language used to describe what a page layout would be in a browser as well as the op-codes and string tokens to display
- Examples of defined VFR opcodes:
  - `FormSet` and `Form` definitions
  - `Subtitle` and other text fields
  - Input: `checkbox`, `numeric`, `string` & `password`
  - Boolean expressions to support errors, suppress, and gray outs: `disableif`, `suppressif` & `grayoutif`

NOTE: VFR is not part of UEFI; this is part of EDK II build tools

Form Example (.vfr file)

BDVar.vfr

```
formset
  guid      = BLANKDRV_FORMSET_GUID,
  title     = STRING_TOKEN(STR_FORM_SET_TITLE),
  help      = STRING_TOKEN(STR_FORM_SET_TITLE_HELP),
  classguid = EFI_HII_PLATFORM_SETUP_FORMSET_GUID,

  varstore BLANKDRV_CONFIGURATION,          // This is the data structure type
    varid = CONFIGURATION_VARSTORE_ID ,     // Optional VarStore ID
    name  = BDMyIfrNVData,                  // Define referenced name in vfr
    guid  = BLANKDRV_FORMSET_GUID;          // GUID of this buffer storage form
// Define a Form (EFI_IFR_FORM)
  form formid = 1,                          // Form ID
  title  = STRING_TOKEN(STR_FORM1_TITLE);   // Form title
    subtitle text = STRING_TOKEN(STR_SUBTITLE_TEXT);

// Define "one of" (EFI_IFR_ONE_OF)
   oneof name = MyOneOf,
      varid   = BDMyIfrNVData.MyBaseAddress,      // Define reference name for Question
      prompt  = STRING_TOKEN(STR_ONE_OF_PROMPT),  // Use "DataStructure.Member" to ref
      help    = STRING_TOKEN(STR_ONE_OF_HELP),
      //
      // Define an option (EFI_IFR_ONE_OF_OPTION)
      option text = STRING_TOKEN(STR_ONE_OF_TEXT1), value = 0x0, flags = 0;
      option text = STRING_TOKEN(STR_ONE_OF_TEXT2), value = 0x1, flags = 0;
      //
      // DEFAULT indicate this option will be marked with EFI_IFR_OPTION_DEFAULT
      option text = STRING_TOKEN(STR_ONE_OF_TEXT3), value = 0x2, flags = DEFAULT;
    endoneof;
```

BlankDrvNVDataStruct.h

```
typedef struct {
    UINT16  MyStringData[40];
    UINT8   MyHexData;
    UINT8   MyBaseAddress;
} BLANKDRV_CONFIGURATION;
```

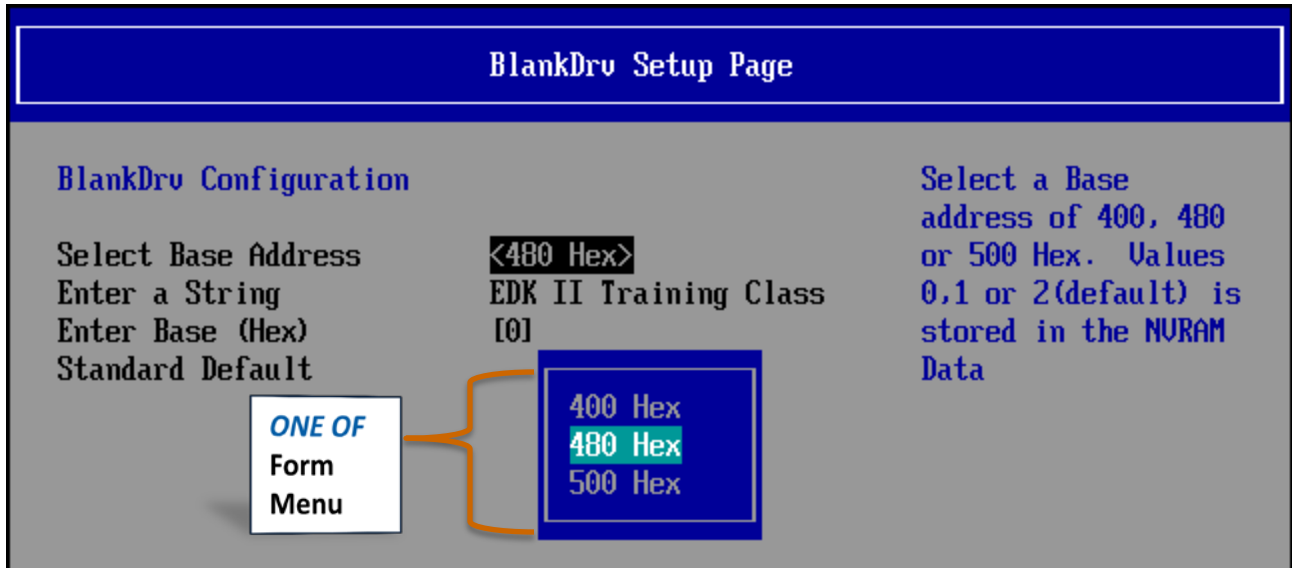
ONE OF Form Menu

See HII Source Code Example: Presentations\Day_2\BDHii\ BlankDrv.c BlankDrv.h BDStrings.uni

Form Example Menu from VFR Example

Driver with Forms menu

FORMSET Menu



Shell Dmpstore to see data kept in NVRAM

```
Shell> dmpstore BDMyIfrNVData
Dump Variable BDMyIfrNVData
Variable NV+BS '5A003BDB-50A1-4568-9170-EBD49E16C47C:BDMyIfrNVData' DataSize = 5
2
  00000000: 45 00 44 00 4B 00 20 00-49 00 49 00 20 00 54 00  *E.D.K. .I.I. .T.*
  00000010: 72 00 61 00 69 00 6E 00-69 00 6E 00 67 00 20 00  *r.a.i.n.i.n.g. .*
  00000020: 43 00 6C 00 61 00 73 00-73 00 00 00 00 00 00 00  *C.l.a.s.s.......*
  00000030: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  *................*
  00000040: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  *................*
  00000050: 00 01                                            *..*
```

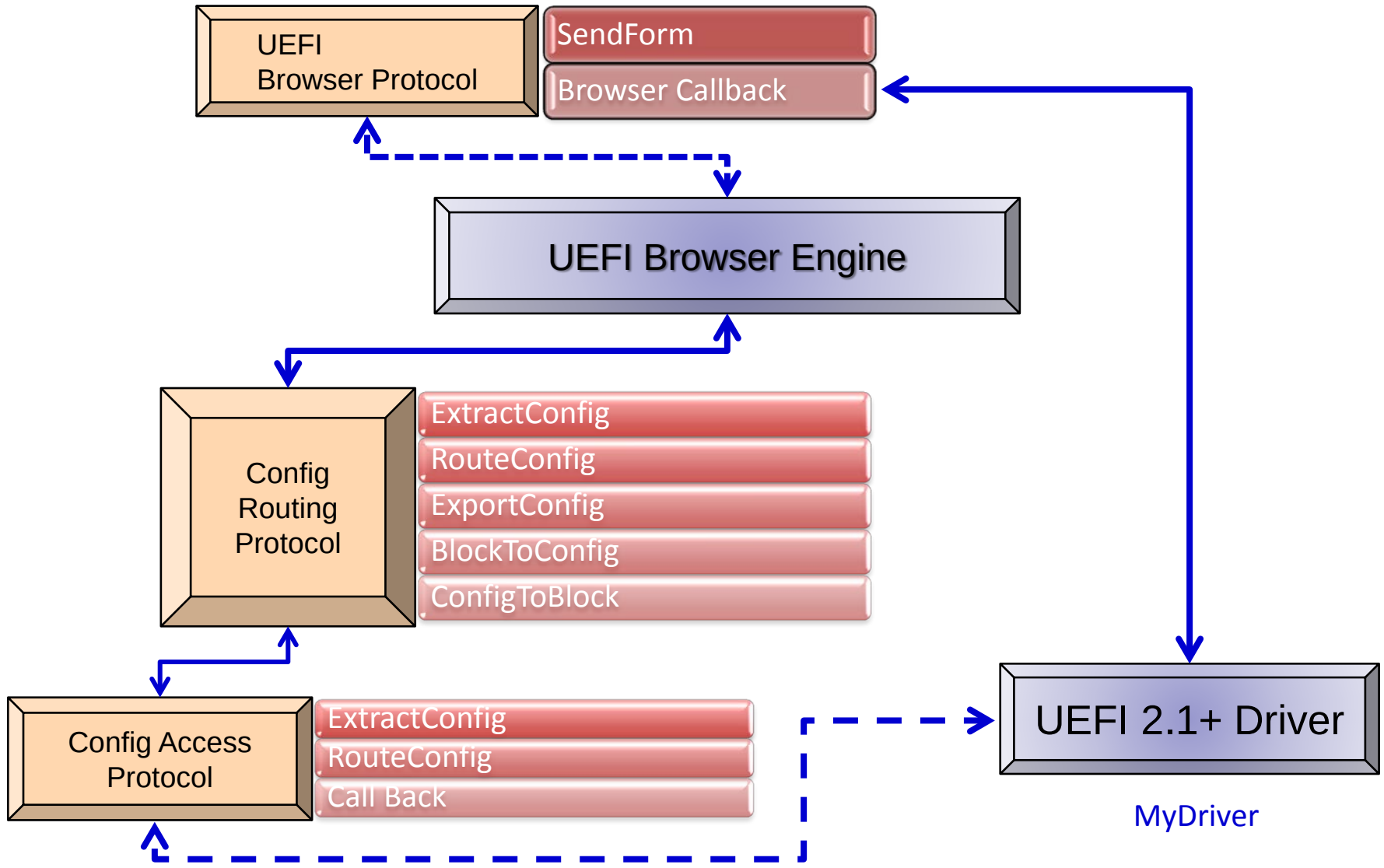
Value from User input Stored in NVRAM

`MyBaseAddress = 480H`

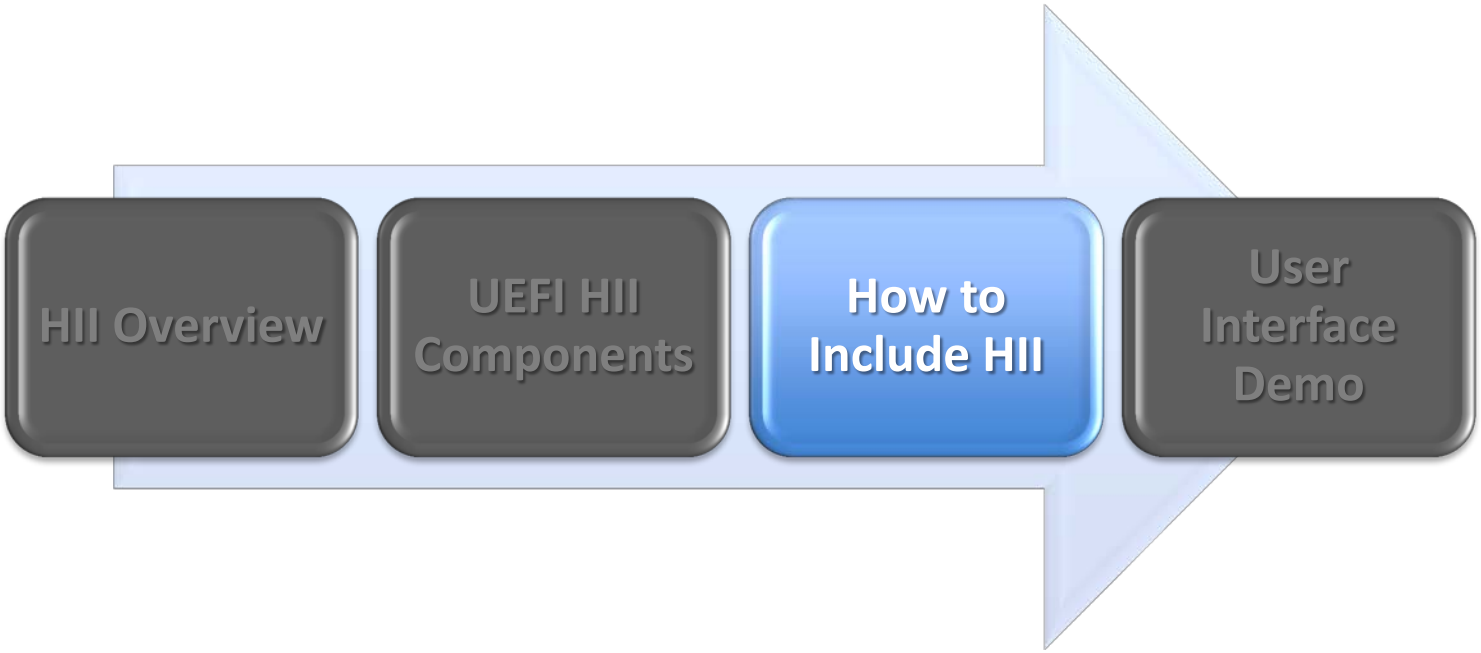
Internal Forms Representation (IFR)

- IFR is defined by the UEFI Specification
  - IFR is "compiled" - created by VFR to IFR tool in EDK II
- Byte encoded operations (much smaller)
- String references abstracted as tokens
- Improved validation, visibility primitives
- A better level of presentation control for firmware
  - Tension between configuration driver and presentation driver over control of presentation format
- Easy to …
  - Interpret for a small setup engine in desktop firmware
  - Translate into another format (XHTML, JavaScript, …)

Form Browser Protocols



Agenda
- Human Interface Infrastructure (HII)



Minimum Files for HII

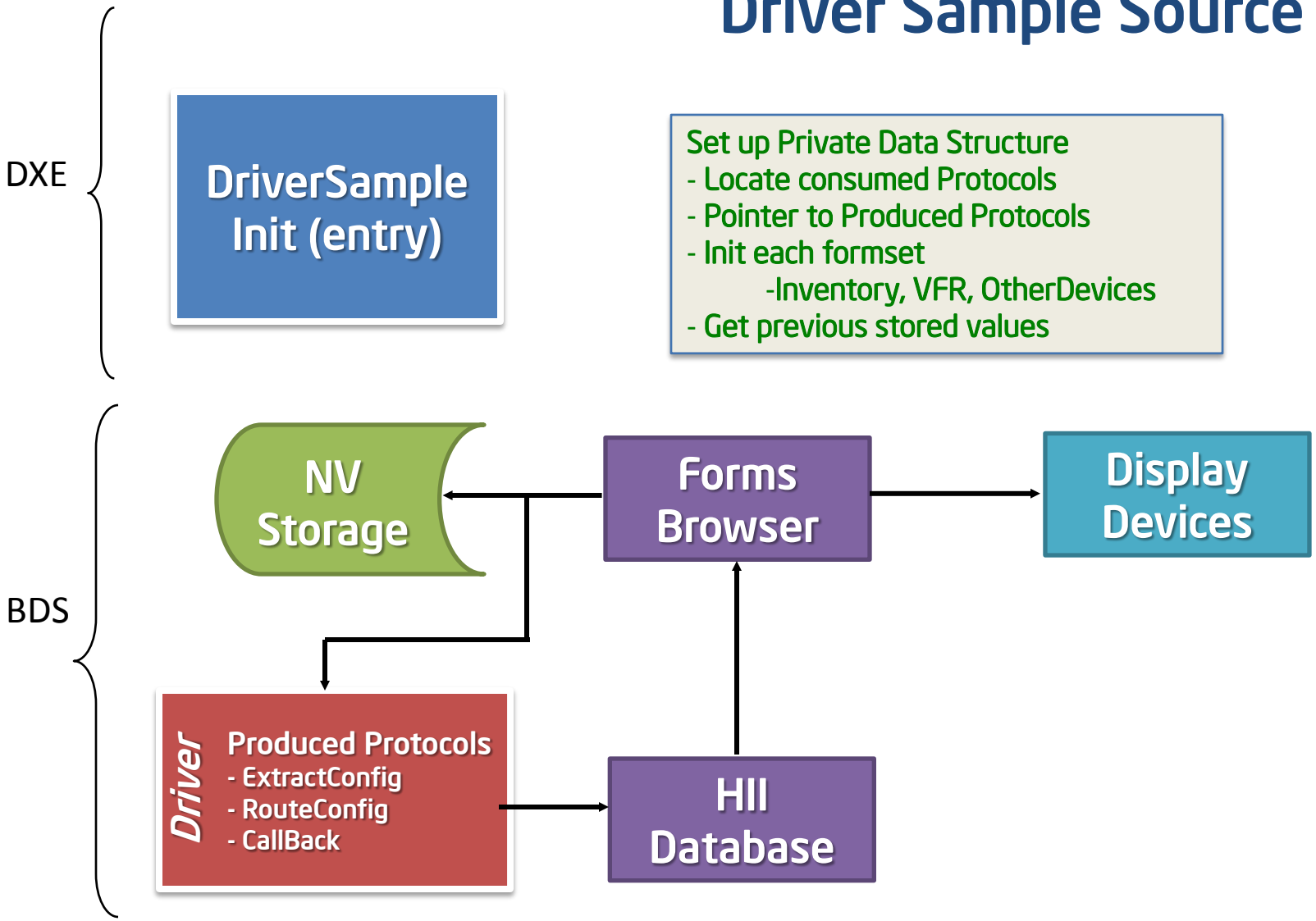
- Driver source file ⟺ .c example *MyDriver.c*
  - Consumes HII protocols
  - Produces EFI_HII_CONFIGURATION_ACCESS_PROTOCOL
  - Publishes Forms Package
- Driver include file ⟺ .h (could be 2 .h files)
  - Defines configuration data
  - Defines private data
- Strings file ⟺ .uni
  - Defines strings in different languages
- Forms file ⟺ .vfr
  - Defines the layout of the screen
- Pre-Make file ⟺ .inf

Driver Sample Code

- UEFI 2.1 Browser example
- Constructed as a developer test to exercise the operations of the infrastructure
  - Sample "does" nothing, aside from interact with the UEFI 2.1+ Protocols and configuration infrastructure
- `DriverSample.c`
  - Consumes protocols
    - `EFI_HII_DATABASE_PROTOCOL`, `EFI_HII_STRING_PROTOCOL`, `EFI_HII_CONFIG_ROUTING_PROTOCOL`, `EFI_FORM_BROWSER2_PROTOCOL`
  - Produces protocol
    - EFI_HII_CONFIGURATION_ACCESS_PROTOCOL
    - *ExtractConfig, RouteConfig and Callback*

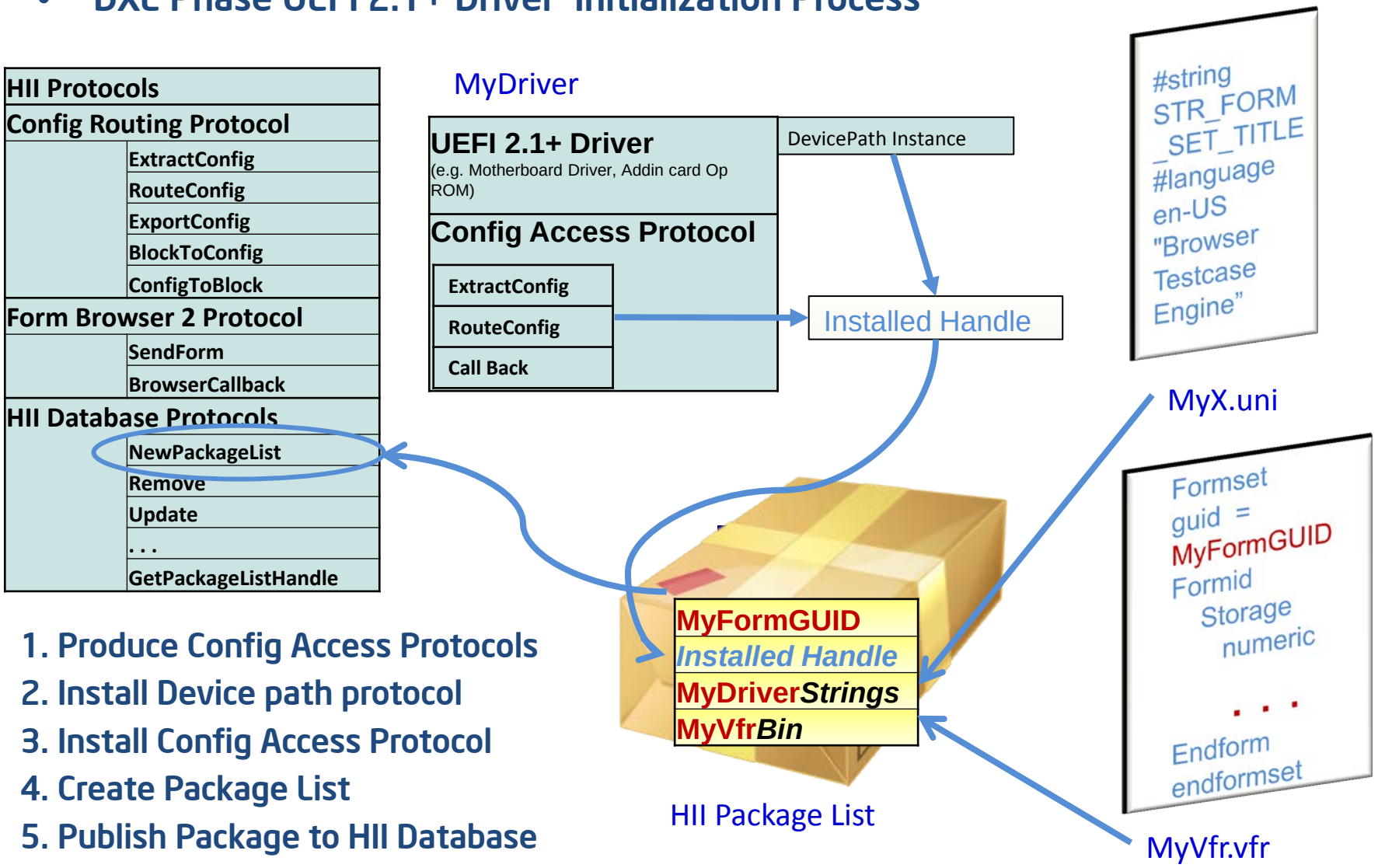
See § 30 of the UEFI 2.x Spec.

Driver Sample Source Code



UEFI 2.1+ Driver Initialization

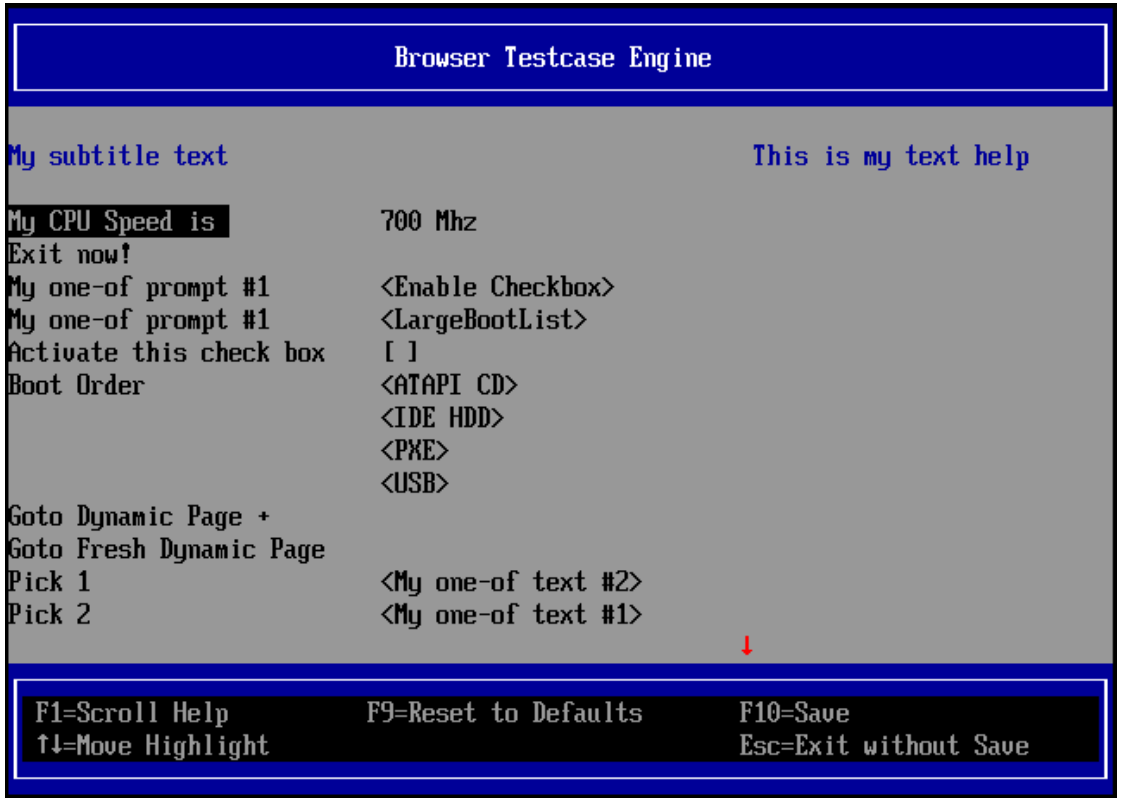
- DXE Phase UEFI 2.1+ Driver Initialization Process



1. Produce Config Access Protocols
2. Install Device path protocol
3. Install Config Access Protocol
4. Create Package List
5. Publish Package to HII Database

UEFI HII Driver Sample

- Main Browser Menu
- Device Manager
- Motherboard Devices
- Browser Test Case Engine



Source Example: UEFI HII Driver Sample

```
// EFI_HII_CONFIGURATION_ACCESS_PROTOCOL
ExtractConfig (  // breaks apart the UNICODE request strings routing them to
                 // the appropriate drivers
   . . .
RouteConfig (    // Breaks apart the UNICODE results strings and returns
                 // configuration information as specified by the request
   . . .
DriverCallback ( // Called from the configuration browser to communicate
                 // activities initiated by a user
   . . .
EFI_STATUS
EFIAPI
DriverSampleInit (                  // Driver entry point
  IN EFI_HANDLE                     ImageHandle,
  IN EFI_SYSTEM_TABLE               *SystemTable
  )
{
   . . .
// Initialize driver private data for produced Protocol
PrivateData->ConfigAccess.ExtractConfig = ExtractConfig;
PrivateData->ConfigAccess.RouteConfig = RouteConfig;
PrivateData->ConfigAccess.Callback = DriverCallback;
   . . .
```

MdeModulePkg\Universal\DriverSampleDxe

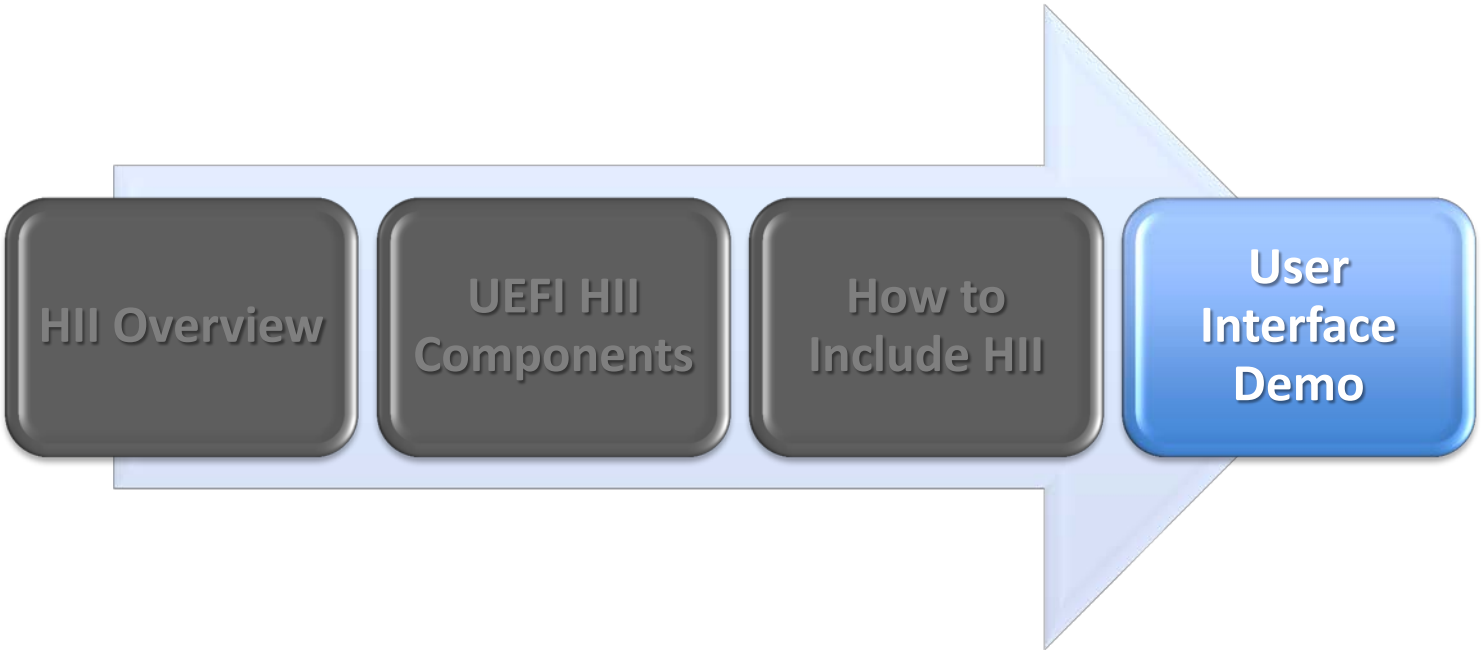
Source Example: UEFI HII Driver Sample (cont.)

```
// Locate HiiDatabase protocol

// Locate HiiString protocol
  Status = gBS->LocateProtocol (&gEfiHiiStringProtocolGuid, NULL, (VOID **) &HiiString);
  PrivateData->HiiString = HiiString;

// Locate Formbrowser2 protocol
  Status = gBS->LocateProtocol (&gEfiFormBrowser2ProtocolGuid, NULL, (VOID **)
                                  &FormBrowser2);
  PrivateData->FormBrowser2 = FormBrowser2;

// Locate ConfigRouting protocol
Status = gBS->LocateProtocol (&gEfiHiiConfigRoutingProtocolGuid, NULL, (VOID **)
                                &HiiConfigRouting);
PrivateData->HiiConfigRouting = HiiConfigRouting;

Status = gBS->InstallMultipleProtocolInterfaces ( // Install Protocol Interfaces
                  &DriverHandle[0],
                  &gEfiDevicePathProtocolGuid,
                  &mHiiVendorDevicePath0,
                  &gEfiHiiConfigAccessProtocolGuid,
                  &PrivateData->ConfigAccess,
                  NULL
                  );
// Publish our HII data
HiiHandle[0] = HiiAddPackages (   // Calls HiiDatabase->NewPackageList Protocol
                   &mFormSetGuid,
                   DriverHandle[0],
                   DriverSampleStrings,
                   VfrBin,
                   NULL         );
```

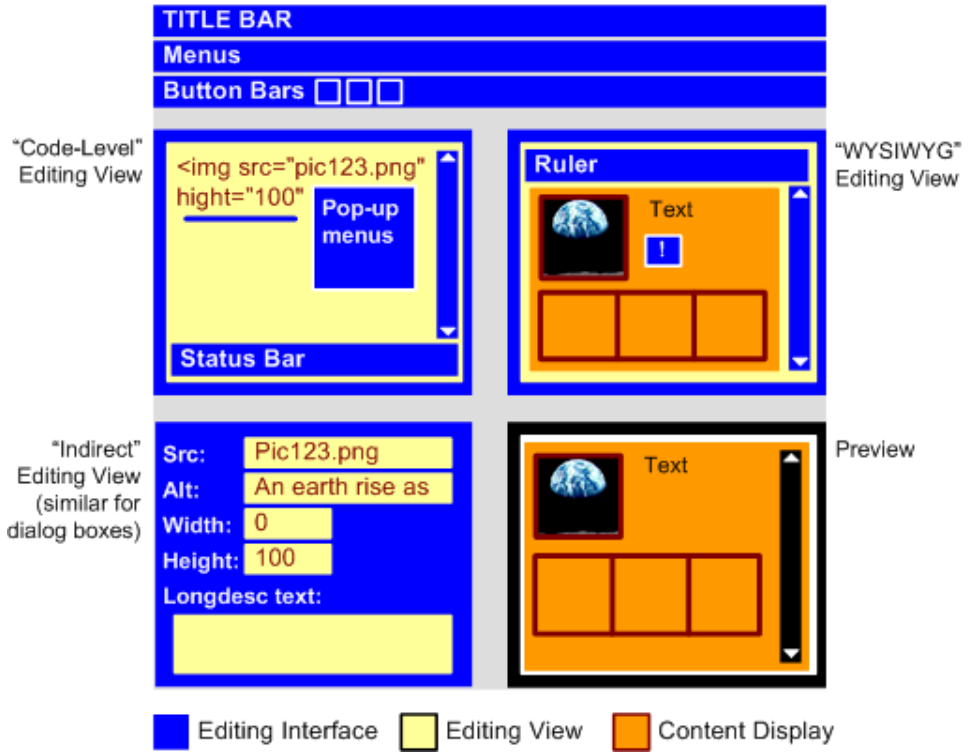
HII: Summary

- Localization designed in from the start
- Localization is independent of display device
- Multi vendor repository for Fonts
- Maps setup easily into Web model
  - Setup replaced with a Markup Language
  - OEM can have unique look and feel
  - Browser defines look and feel
  - IFR maps to XML/HTML plus JavaScript

Agenda
- Human Interface Infrastructure (HII)



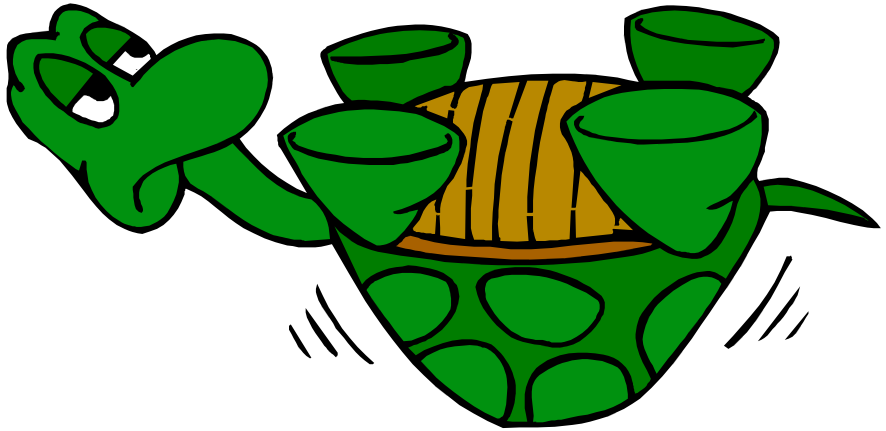
User Interface Demo



Q & A

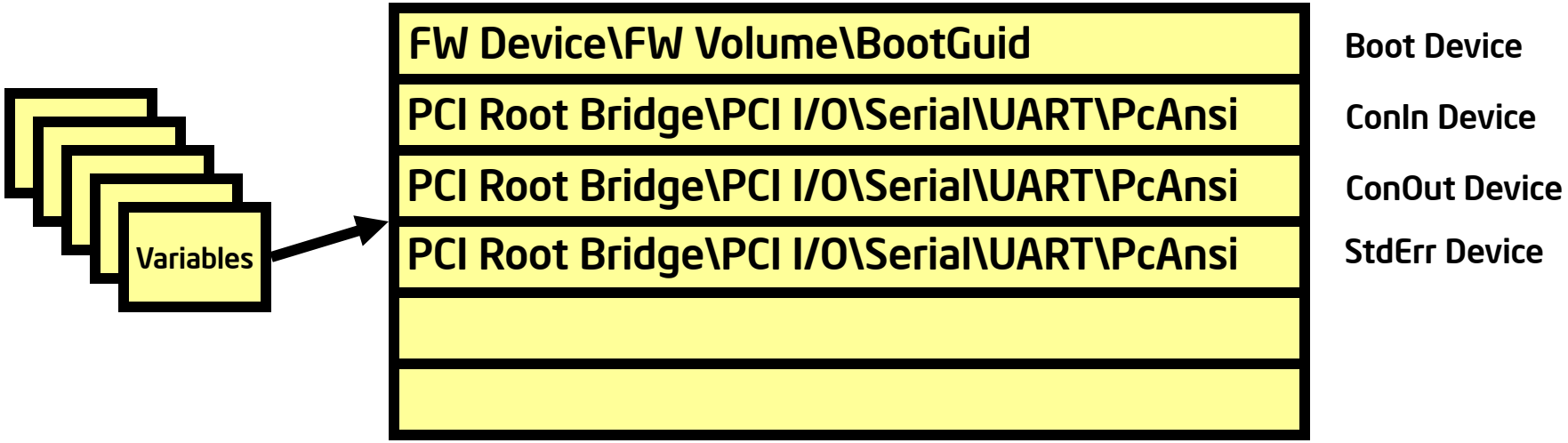

intel®
Software

Back up



BDS Variable and Stack Management

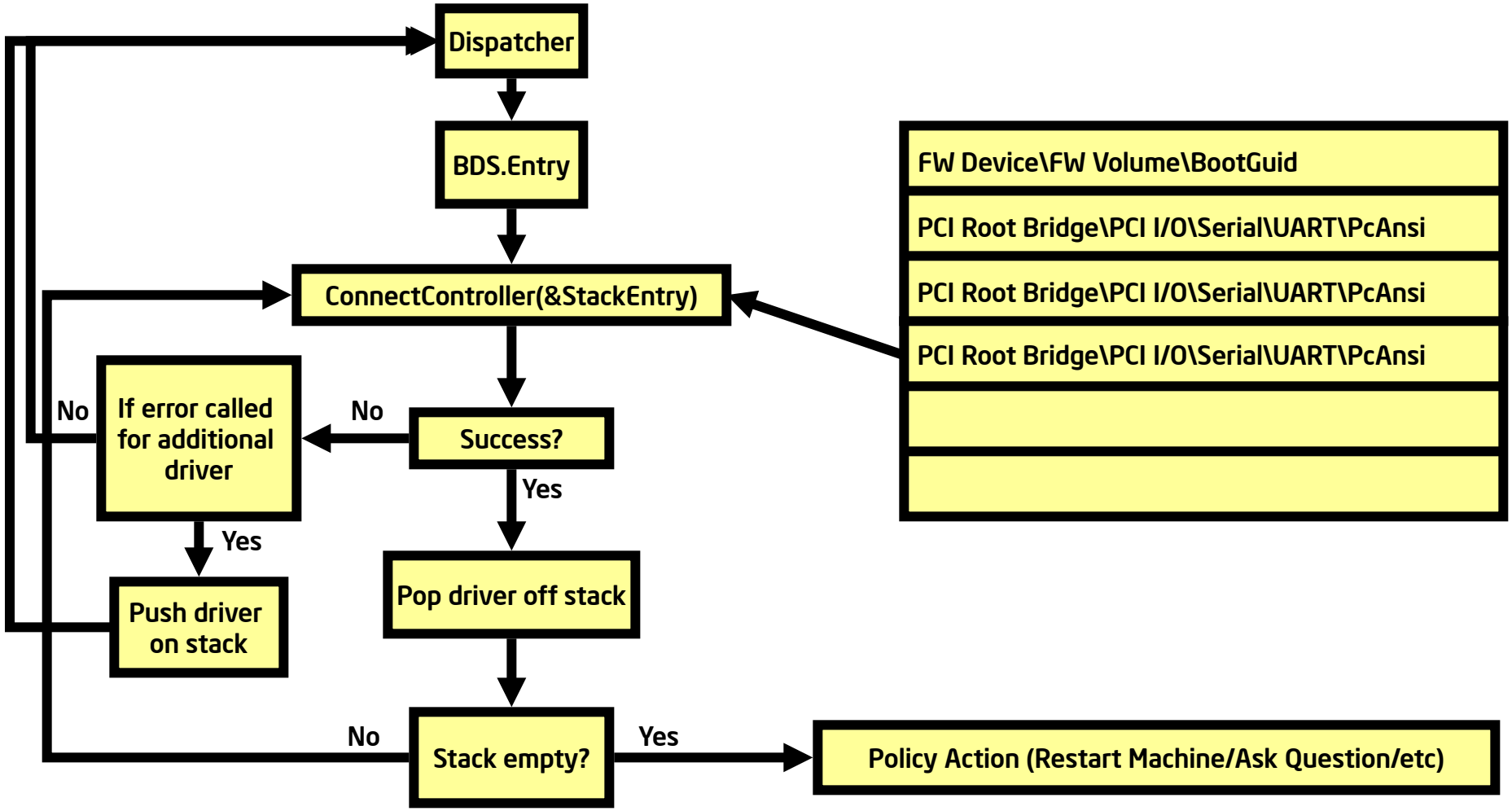
NOTE: The stack is the framework[1] implementation example and NOT architectural.



BDS.Entry will process the global variables. From the device paths a stack will be created on which the BDS will act upon.

[1]Intel® Platform Innovation Framework for UEFI

BDS Variable and Stack Management



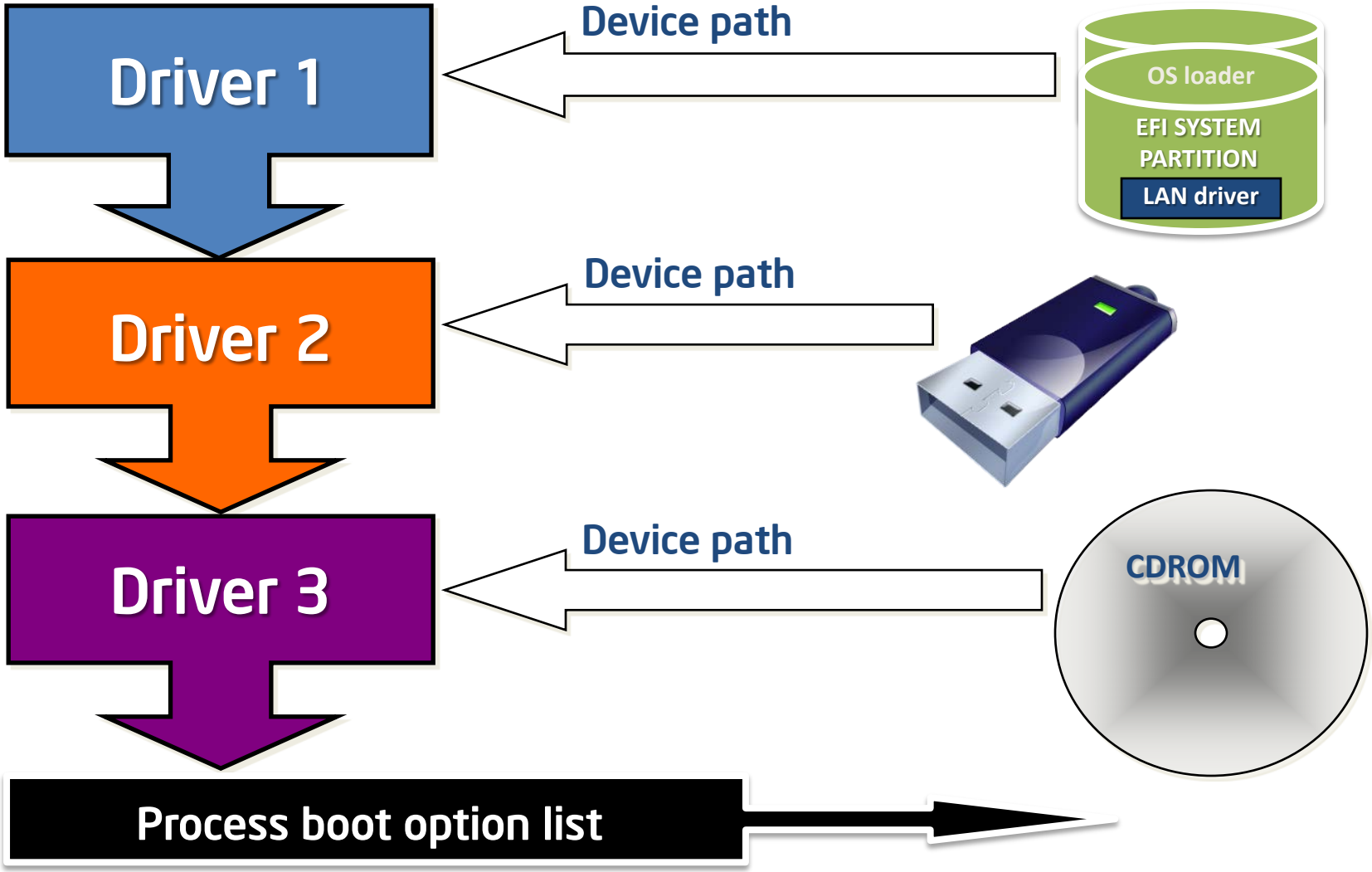
BDS Policy Input

Globally Defined Variables

- **`DriverOrder`** — A list of UINT16's that make up an ordered list of the **`Driver####`** variable.
- **`Driver####`** — A driver load option. BDS will attempt to load the driver specified. Contains an **`EFI_LOAD_OPTION`**. An example would be a PCI Root Bridge, or Serial I/O driver.

Process Driver Option List

NVRAM



Additional considerations for HII

- Localization support
  - OEM might expect support for US, fr-FR, de-DE, es-ES, ja-JP, zh-Hans, etc.
  - EFI_COMPONENT_NAME2_PROTOCOL supports the languages specified
- Provide `EFI_HII_DEFAULT_CLASS_STANDARD` for all configurable items which can be set to default
- Limit call backs that dynamically modifies IFR.
  - Instead, opcodes like grayoutif and suppressif can be used to dynamically change fields from read-only to read/write or dynamically suppress/un-suppress fields
- Provide a title in the HII formset and form

Additional considerations for HII

- Set `EFI_IFR_FLAG_RESET_REQUIRED` for items that require a reboot to take effect. Do not use system reset in `routeconfig` or callbacks.
- HII configuration drivers must implement `EFI_IFR_FORM_SET_OP` and set one of the `ClassGuid[]` to `EFI_HII_PLATFORM_SETUP_FORMSET_GUID` to indicate that the HII formset published by this driver is used for platform configuration.
- Consider Configuration Mapping Support using `UEFI_CONFIG_LANG` "UEFI-X". Please contact your OEM partner for additional information.
- Any settings changes made via HII must take effect when booting in UEFI or in legacy BIOS mode


intel®
Software

Disclaimer

THIS INFORMATION COTNAINED IN THIS DOCUMENT, INCLUDING ANY TEST RESULTS ARE PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT OR BY THE SALE OF INTEL PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel retains the right to make changes to its specifications at any time, without notice.

Recipients of this information remain solely responsible for the design, sale and functionality of their products, including any liability arising from product infringement or product warranty.

Intel may make changes to specifications, product roadmaps and product descriptions at any time, without notice.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2®, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804