

# UEFI & EDK II TRAINING

EDK II Debugging through UEFI Boot Flow

[tianocore.org](https://tianocore.org)



# LESSON OBJECTIVE

- ★ Debugging commands similar to all debuggers
- ★ Debugging UEFI Platform Initialization Boot Flow

# DEBUGGING COMMANDS

CpuBreakpoint() added to Source



# Source Level Debugging

View call stack Go

Insert CpuBreakpoint()

View and edit local/global variables

Set breakpoint Step into/over routines

Go till View disassembled code

View/edit general purpose register values



# CpuBreakpoint Vs. CpuDeadLoop

## CpuBreakpoint

When using a Software debugger:

- Visual Studio
- GDB (ovmf with qemu)
- Intel® UDK Debugger
- [Windriver\\*](#) Simics
- Debug agent – SourceLevelDebugPkg

## CpuDeadLoop

When using a Hardware debugger:

- In-Target Probe (ITP)
- Intel® SVT DCI cable
- Intel® SVT Closed Chassis Adapter (CCA)
- other 3<sup>rd</sup> Party Hardware (i.e. [Lauterbach](#) w/ JTAG)

The functions `CpuBreakpoint()` and `CpuDeadLoop()` are part of the EDK II Base Libraries and can be compiled with any UEFI or PI Module at any phase of the boot flow (SEC, PEI, DXE, BDS, TSL)



# Special DCI Breakpoint with HW Debugger

## CpuIceBreakpoint

The Intel Architecture has a special op-code for a breakpoint: `int1`  
Better than a `CpuDeadLoop()` since it halts the processor. Better trace information

Downside:

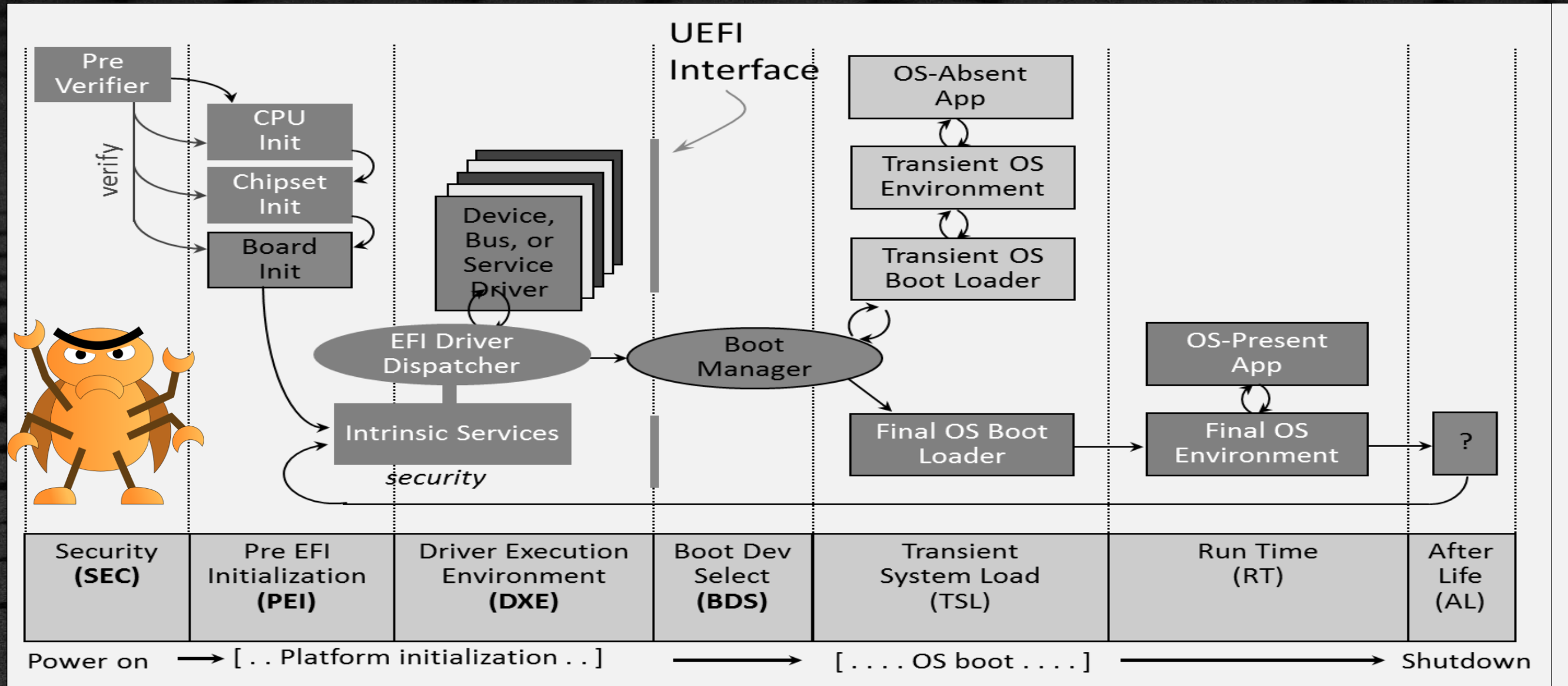
- Requires a Hardware Debugger with DCI capabilities to intercept the `int1` op code
- There is no “C” equivalent – needs to be assembly code

# DEBUGGING THRU BOOT FLOW

Add Breakpoints to the Compiled BIOS / Firmware Source Code

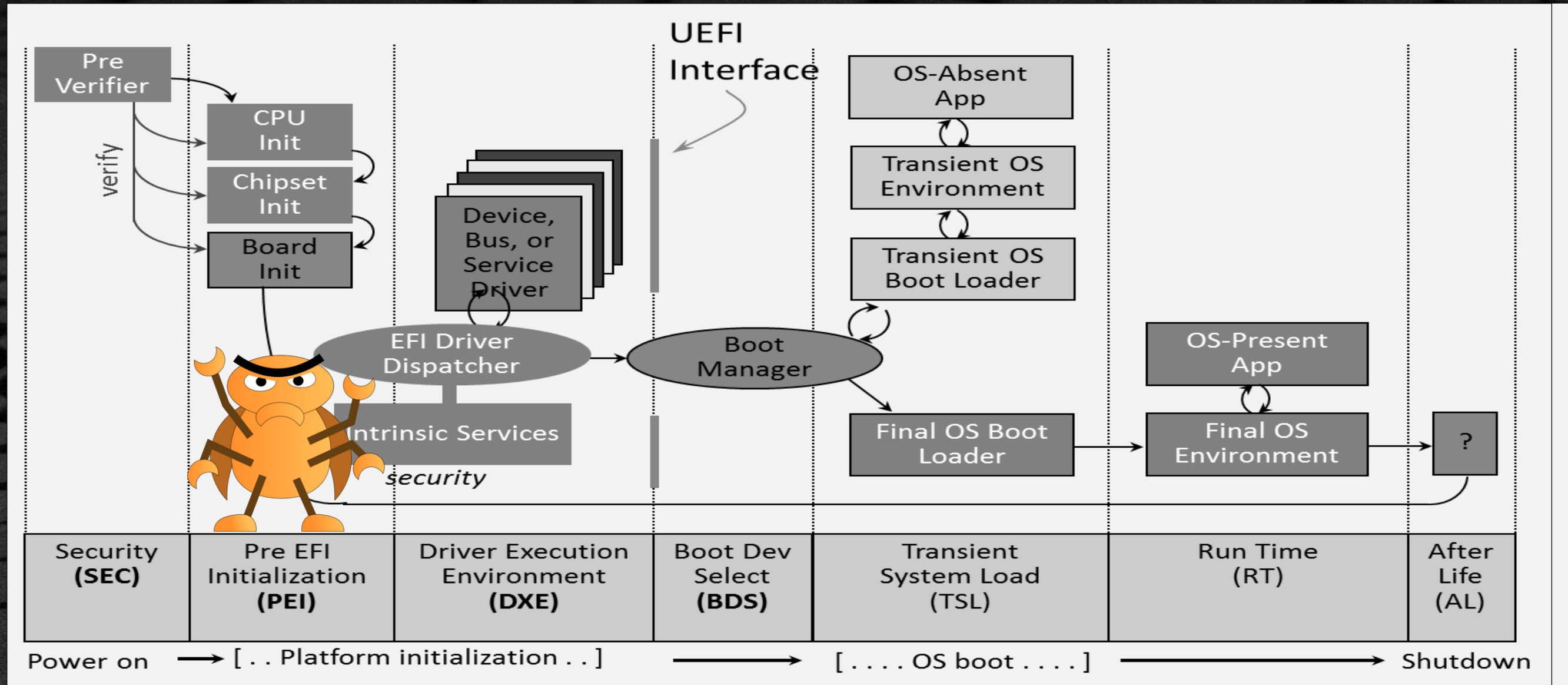


# Debugging the Boot Phases



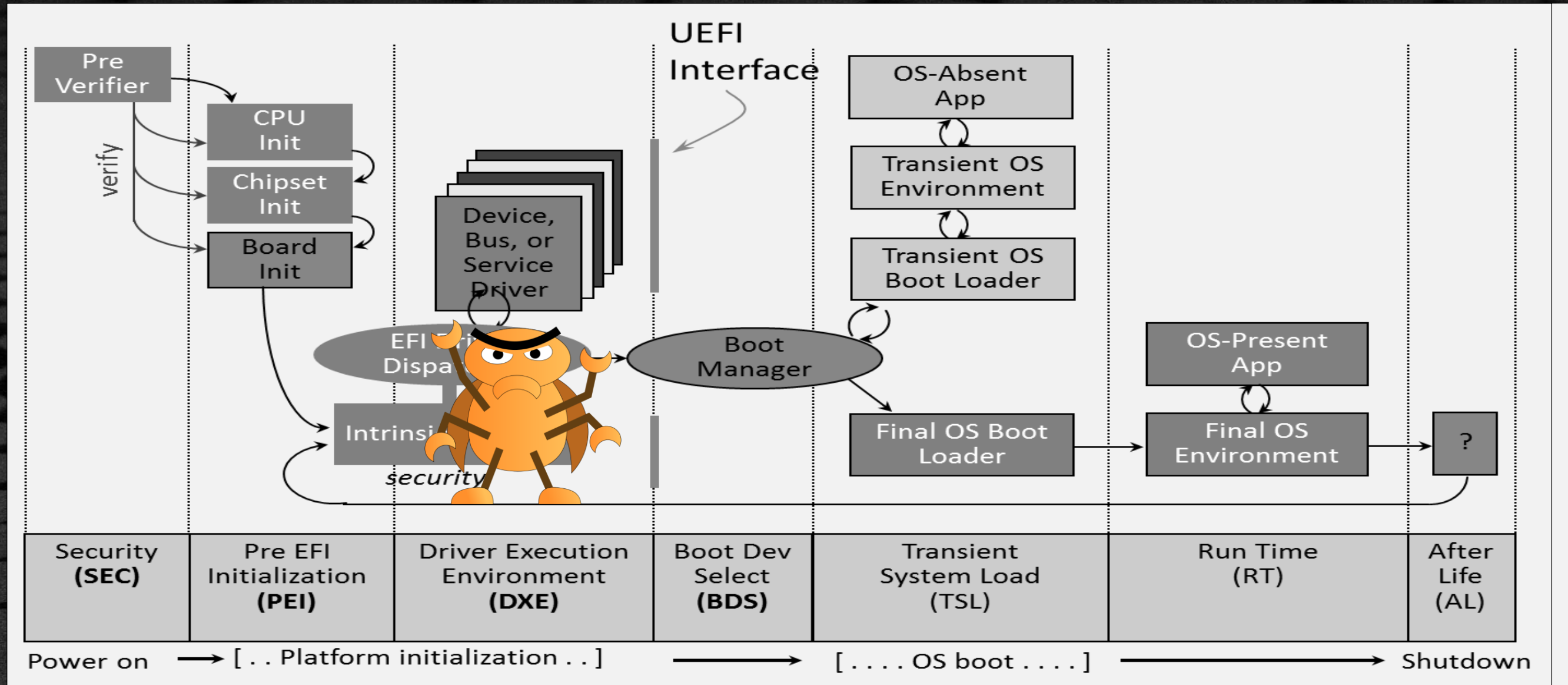


# Debugging the Boot Phases



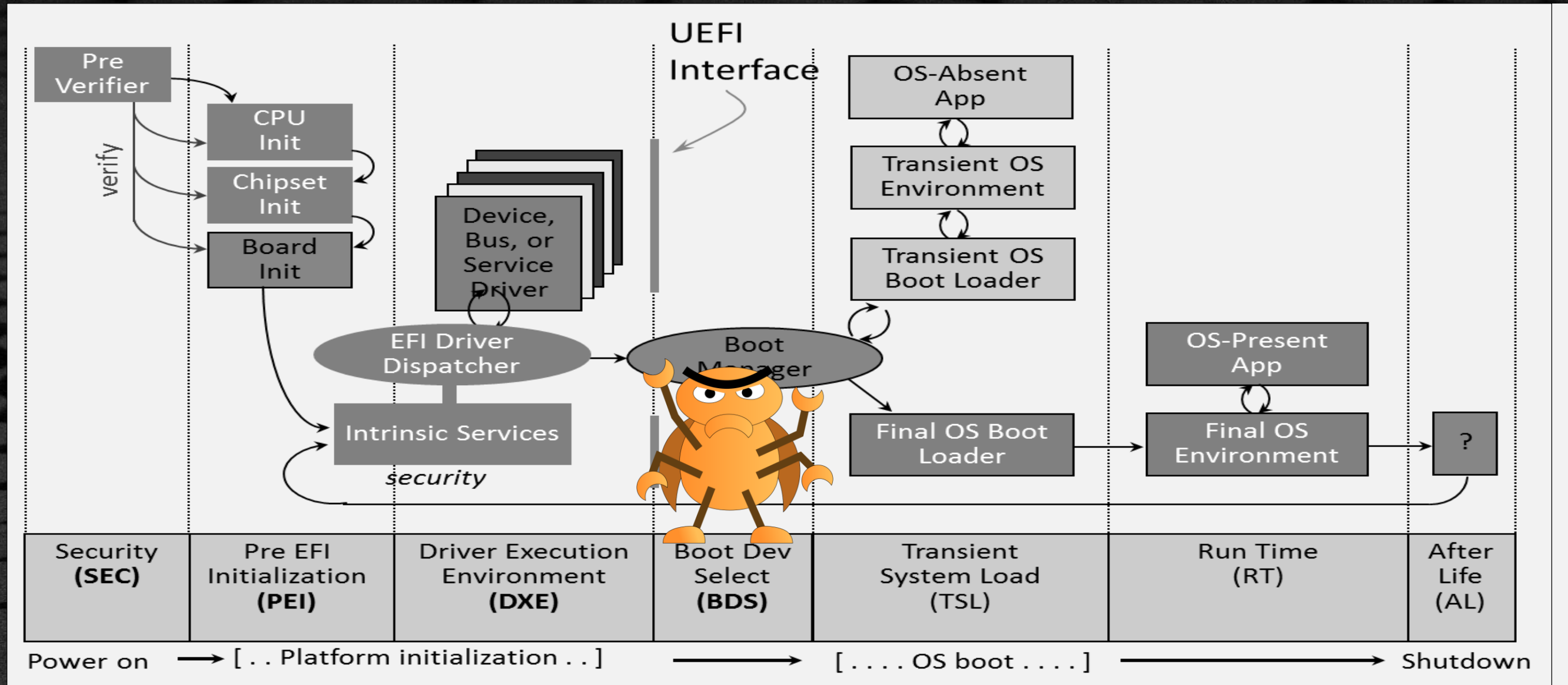


# Debugging the Boot Phases





# Debugging the Boot Phases



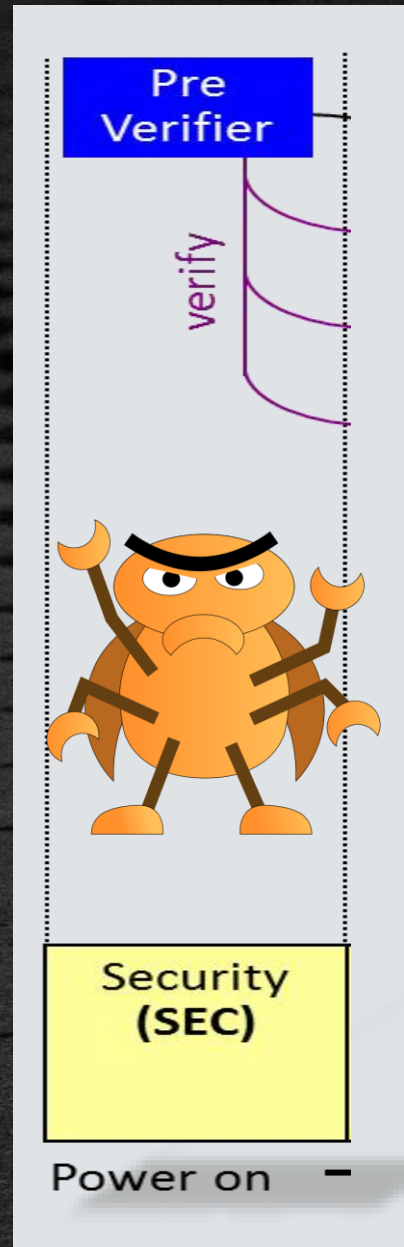


# Debugging the Boot Phases - SEC

## Debugging Sec Phase

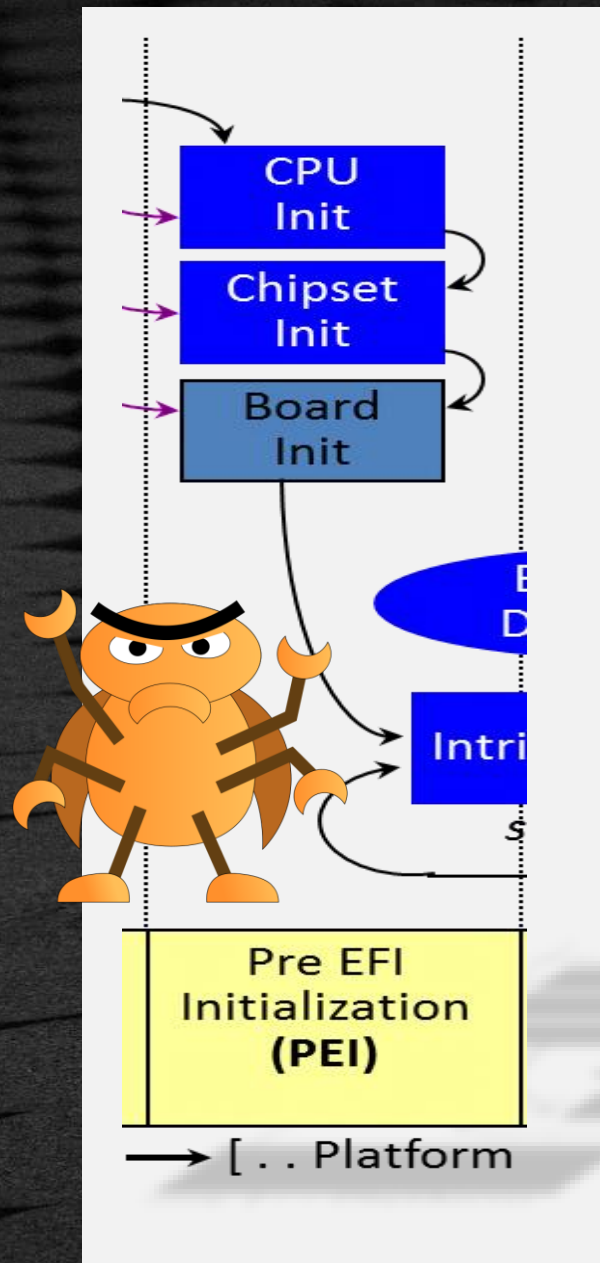
Hardware debugger capable **only**

- Break at the Reset Vector
- Check temporary memory – CAR NEM
- Enable the “C” Code
- Transfer control to PEI





# Debugging the Boot Phases - PEI



- Use debugger prior to PEI Main
- Check proper execution of PEI drivers
- Execute basic chipset & Memory init.
- Check memory availability
- Complete flash accessibility
- Execute recovery driver
- Detect DXE IPL



# PEI Phase: Trace Each PEIM

There is a loop function in :

 [MdeModulePkg/Core/Pei/Dispatcher/Dispatcher.c](#)

Add CpuBreakpoint(); before launching each PEIM

```
VOID
PeiDispatcher (
    IN CONST EFI_SEC_PEI_HAND_OFF  *SecCoreData,
    IN PEI_CORE_INSTANCE            *Private
)
{ // ...
    // Call the PEIM entry point
    //
    PeimEntryPoint = (EFI_PEIM_ENTRY_POINT2)(UINTN)EntryPoint;
    PERF_START (PeimFileHandle, "PEIM", NULL, 0);
    // Add a call to CpuBreakpoint(); approx. line 1004
    CpuBreakpoint();
    PeimEntryPoint(PeimFileHandle, (const EFI_PEI_SERVICES **) &Private->Ps);
}
```



# Check for transition from PEI to DXE

Critical point before calling DXE in:

 [MdeModulePkg/Core/Pei/PeiMain.c](https://github.com/tianocore/MdeModulePkg/Core/Pei/PeiMain.c)

Add CpuBreakpoint(); before entering DxeIpl

```
VOID
EFIAPI
PeiCore (
    IN CONST EFI_SEC_PEI_HAND_OFF          *SecCoreDataPtr,
    IN CONST EFI_PEI_PPI_DESCRIPTOR        *PpiList,
    IN VOID                                *Data
)
{ // ...
    // Enter DxeIpl to load Dxe core.
    //
    DEBUG ((EFI_D_INFO, "DXE IPL Entry\n"));
    // Add a call to CpuBreakpoint(); approx. line 468
    CpuBreakpoint();
    Status = TempPtr.DxeIpl->Entry (
        TempPtr.DxeIpl,
        &PrivateData.Ps,
        PrivateData.HobList
```

# Check for transition from DxeIpl to DXE

Critical point before calling DXE Core in:

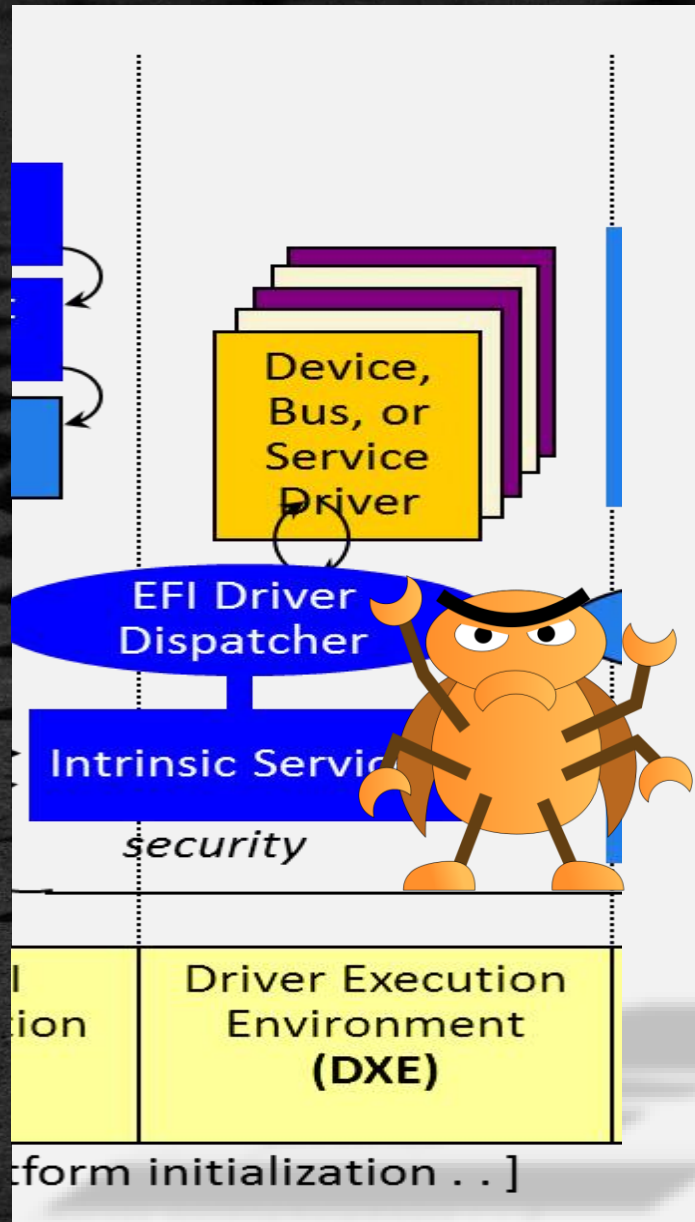
 [MdeModulePkg/Core/DxeIplPeim/DxeLoad.c](#)

Before entering Dxe Core ( Notice also this is a standalone module - DxeIpl.efi)

```
EFI_STATUS
EFIAPI
DxeLoadCore (
    IN CONST EFI_DXE_IPL_PPI *This,
    IN EFI_PEI_SERVICES      **PeiServices,
    IN EFI_PEI_HOB_POINTERS  HobList
)
{ // ...
    // Transfer control to the DXE Core
    // The hand off state is simply a pointer to the HOB list
    //
    // Add a call to CpuBreakpoint(); approx. line 790
    CpuBreakpoint();
    HandOffToDxeCore (DxeCoreEntryPoint, HobList);
    //
    // If we get here, then the DXE Core returned. This is an error
```



# Debugging the Boot Phases - DXE



- Search for cyclic dependency check
- Trace ASSERTs caused during DXE execution
- Debug individual DXE drivers
- Check for architectural protocol failure
- Ensure BDS entry call



# DXE: Trace Each Driver Load

DXE Dispatcher calls to each driver's entry point in:

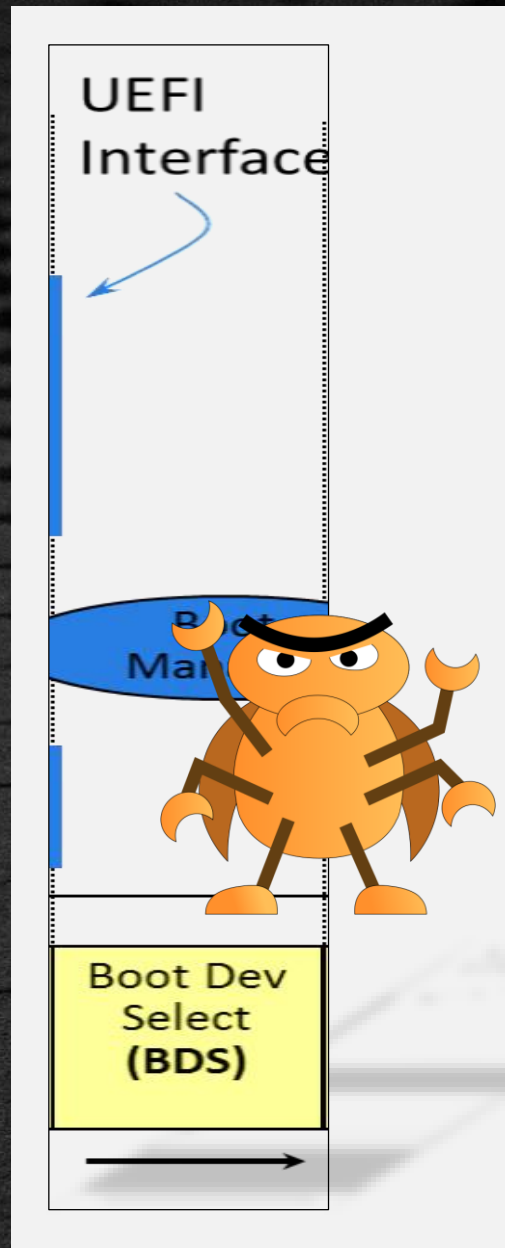
 [MdeModulePkg/Core/Dxe/Image/Image.c](https://github.com/tianocore/MdeModulePkg/Core/Dxe/Image/Image.c)

Break every time a DXE driver is loaded.

```
EFI_STATUS
EFIAPI
CoreStartImage (
    IN EFI_HANDLE  ImageHandle,
    OUT UINTN      *ExitDataSize,
    OUT CHAR16     **ExitData  OPTIONAL
)
{ // ...
    //
    // Call the image's entry point
    //
    Image->Started = TRUE;
    // Add a call to CpuBreakpoint(); approx. line 1673
    CpuBreakpoint();
    Image->Status = Image->EntryPoint (ImageHandle, Image->Info.SystemTable);
}
```



# Debugging the Boot Phases - BDS



- Detect console devices (input and output)
- Check enumeration of all devices' preset
- Detect boot policy
- Ensure BIOS “front page” is loaded



# BDS Phase – Entry Point

DXE call to BDS entry point in:

 [MdeModulePkg/Core/Dxe/DxeMain/DxeMain.c](https://github.com/tianocore/tianocore/blob/master/MdeModulePkg/Core/Dxe/DxeMain/DxeMain.c)

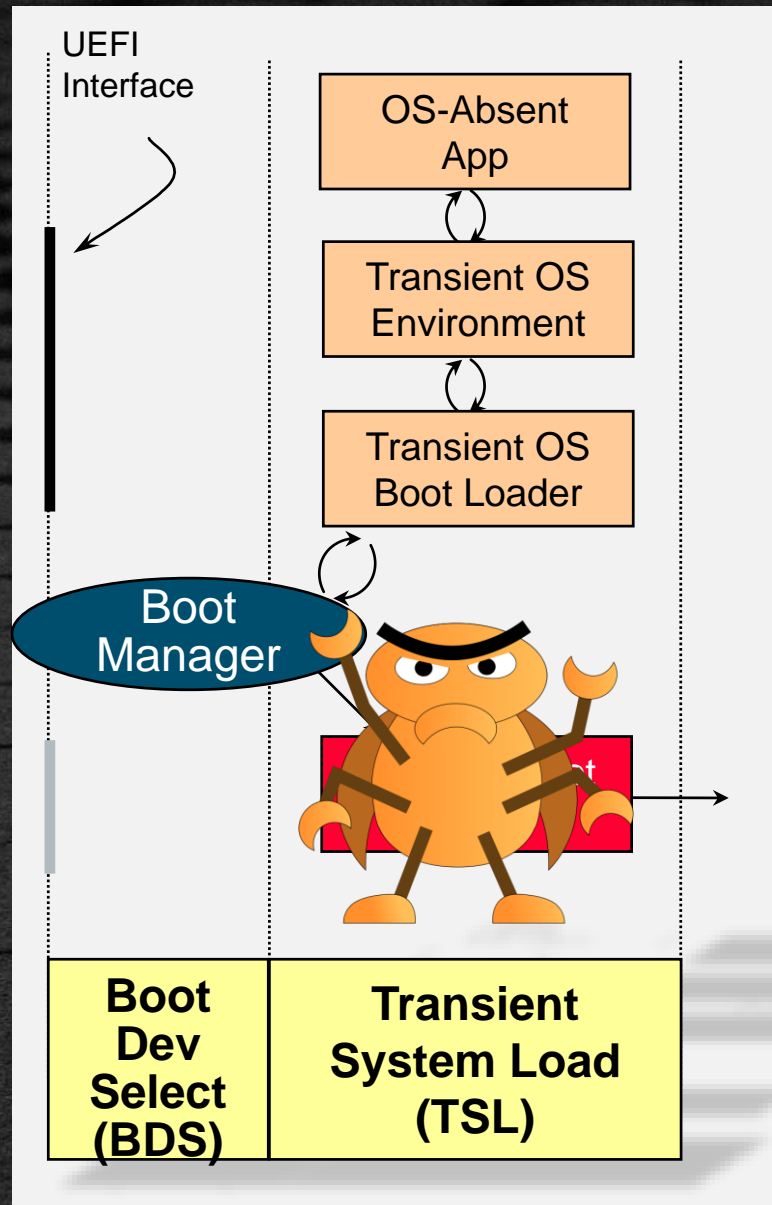
Add CpuBreakpoint(); to break before BDS.

```
VOID
EFIAPI
DxeMain (
    IN VOID *HobStart
)
{ // ...
    // Transfer control to the BDS Architectural Protocol
    //
    // Add a call to CpuBreakpoint(); approx. line 554
    CpuBreakpoint();
    gBds->Entry (gBds);

    //
    // BDS should never return
    //
    ASSERT (FALSE);
    CpuDeadLoop ();
}
```



# Debugging the Boot Phases - Pre-Boot



- “C” source debugging
- UEFI Drivers
  - Init
  - Start
  - Supported
- UEFI Shell Applications
  - Entry point
  - Local variables
- `CpuBreakpoint()`



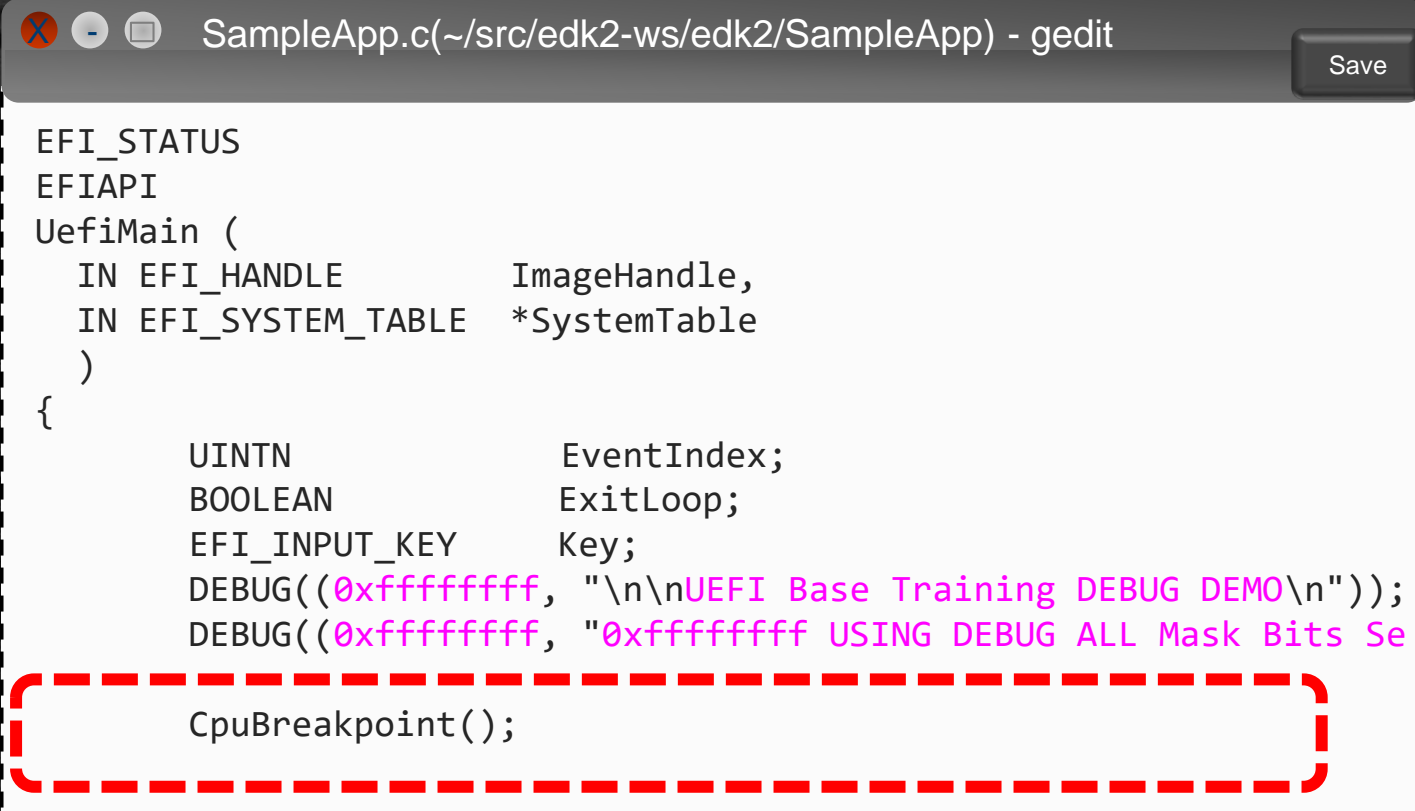
# Debug in Pre-Boot – UEFI Shell Application

Add CpuBreakpoint() to SampleApp.c near the entry point

Add SampleApp.inf to the platform .dsc file

```
bash$ cd <edk2 workspace directory>
bash$ . edksetup.sh
bash$ build -m SampleApp/SampleApp.inf
```

Copy the binary SampleApp.efi to  
USB drive



```
EFI_STATUS
EFIAPI
UefiMain (
    IN EFI_HANDLE        ImageHandle,
    IN EFI_SYSTEM_TABLE  *SystemTable
)
{
    UINTN        EventIndex;
    BOOLEAN      ExitLoop;
    EFI_INPUT_KEY Key;
    DEBUG((0xffffffff, "\n\nUEFI Base Training DEBUG DEMO\n"));
    DEBUG((0xffffffff, "0xffffffff USING DEBUG ALL Mask Bits Se

    CpuBreakpoint();
```



# SUMMARY

- ★ Debugging commands similar to all debuggers
- ★ Debugging UEFI Platform Initialization Boot Flow



# Questions?





# Return to Main Training Page



Return to Training Table of contents for next presentation [link](#)



