HYPERVISTA® TECHNOLOGIES

Winning the Race to Bare Metal

~

UEFI Hypervisors

“He who gets there fastest with the mostest wins”

HYPERVISTA® TECHNOLOGIES

Summary

- There is a race to bare metal between black hats and white hats
- UEFI pre-OS capabilities provide the ability to launch tools BEFORE the OS bootloader is called
- Bare metal hypervisor technology is powerful, well understood and maturing
- Commodity support for UEFI 2.0 is emerging
- The combination of bare metal hypervisors and UEFI has great potential and implications for system exploitation and security

HYPERVISTA® TECHNOLOGIES

Our Custom Bare Metal Hypervisor

- Lightweight and pluggable by design
- Contains a custom hypervisor runtime debugger
- Capable of hosting x64 line of Windows Operating Systems
- Implements Intel® VMX (VT-x)

HYPERVISTA® TECHNOLOGIES

UEFI

- **U**nified **E**xtensible **F**irmware **I**nterface

- UEFI describes a programmatic interface between the platform firmware and the OS
- UEFI replaces and extends the functionality of legacy BIOS
- UEFI 2.0 is supported in Microsoft Windows Vista x64 SP1 and Linux distributions
- Commodity motherboards are beginning to support UEFI 2.0
- Some of the more interesting an powerful features of UEFI are the pre-OS capabilities

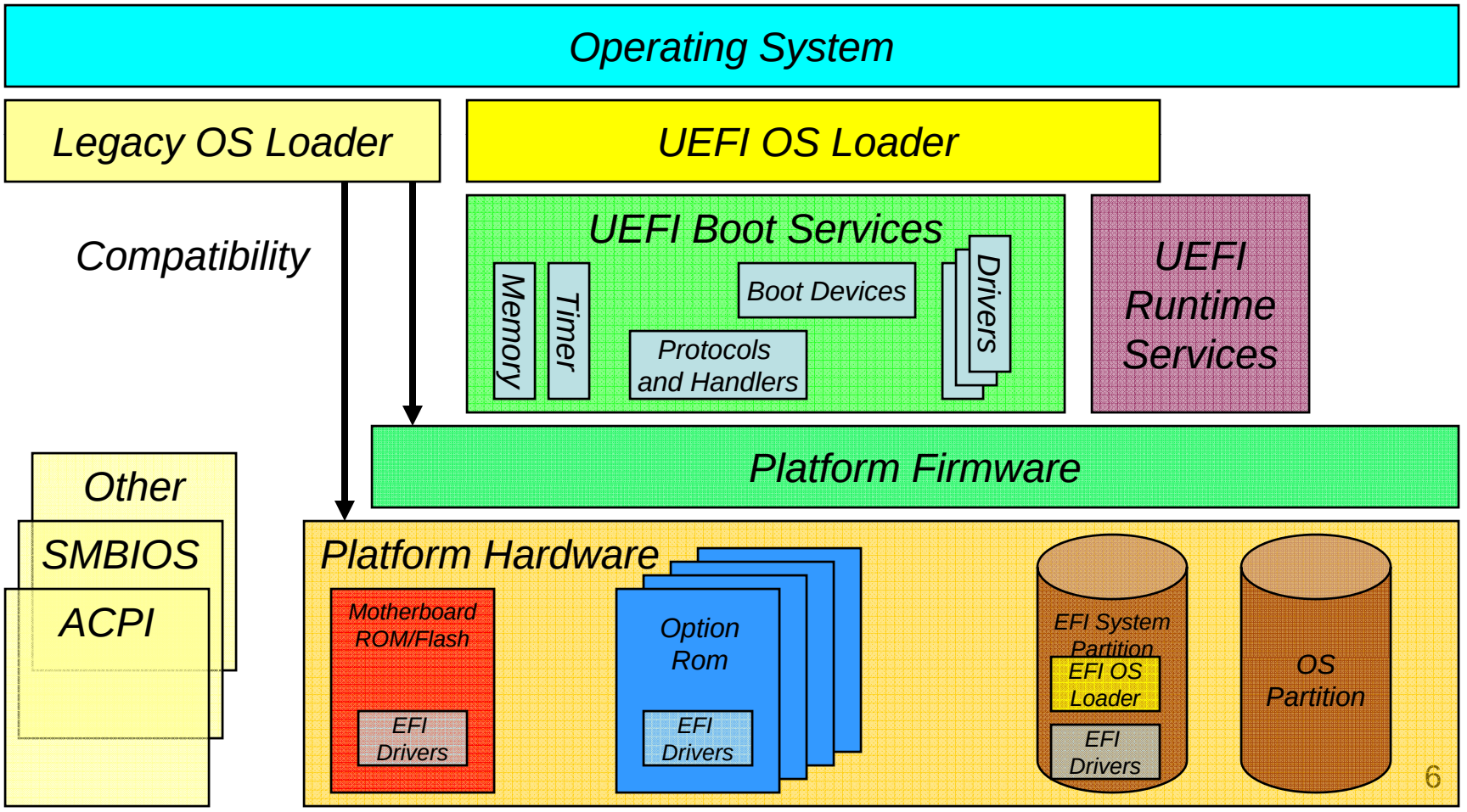
HYPERVISTA® TECHNOLOGIES

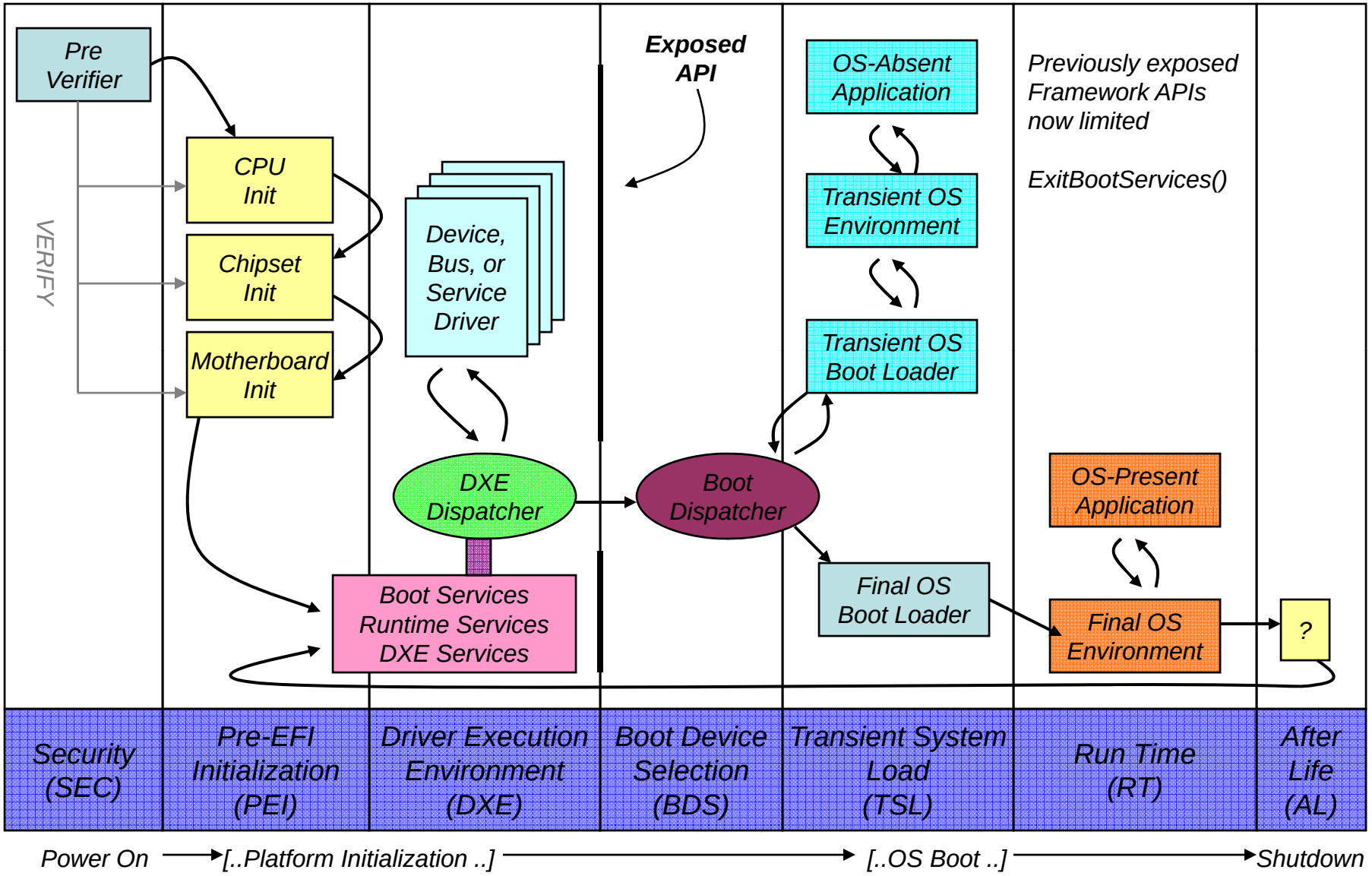
UEFI Goals

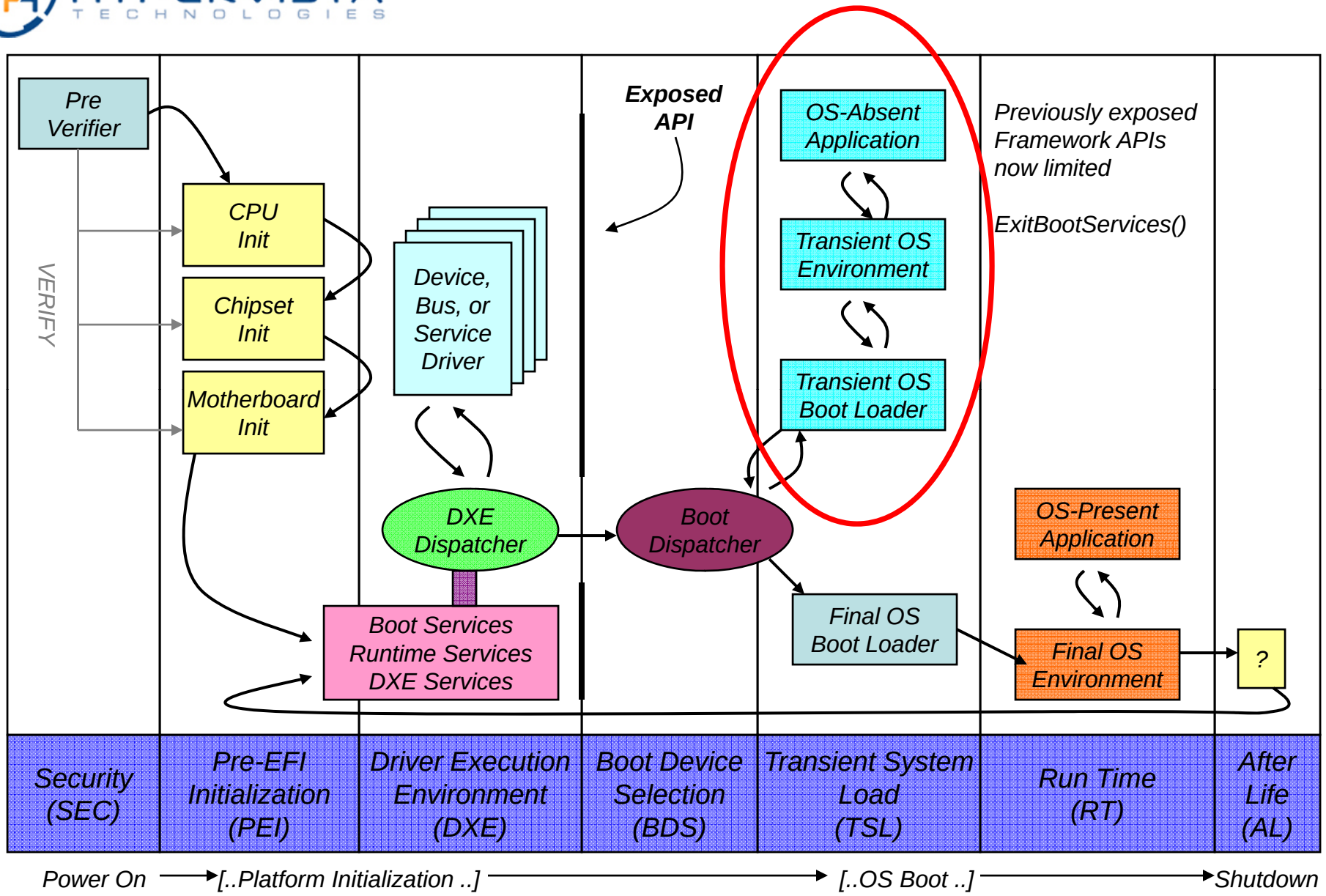
- OS neutrality
- Well defined, clean, and extensible interfaces
- Modularity
- HLL friendly
- Scalability
- Revolutionary Boot Manager
- Kick-Ass pre-OS capabilities

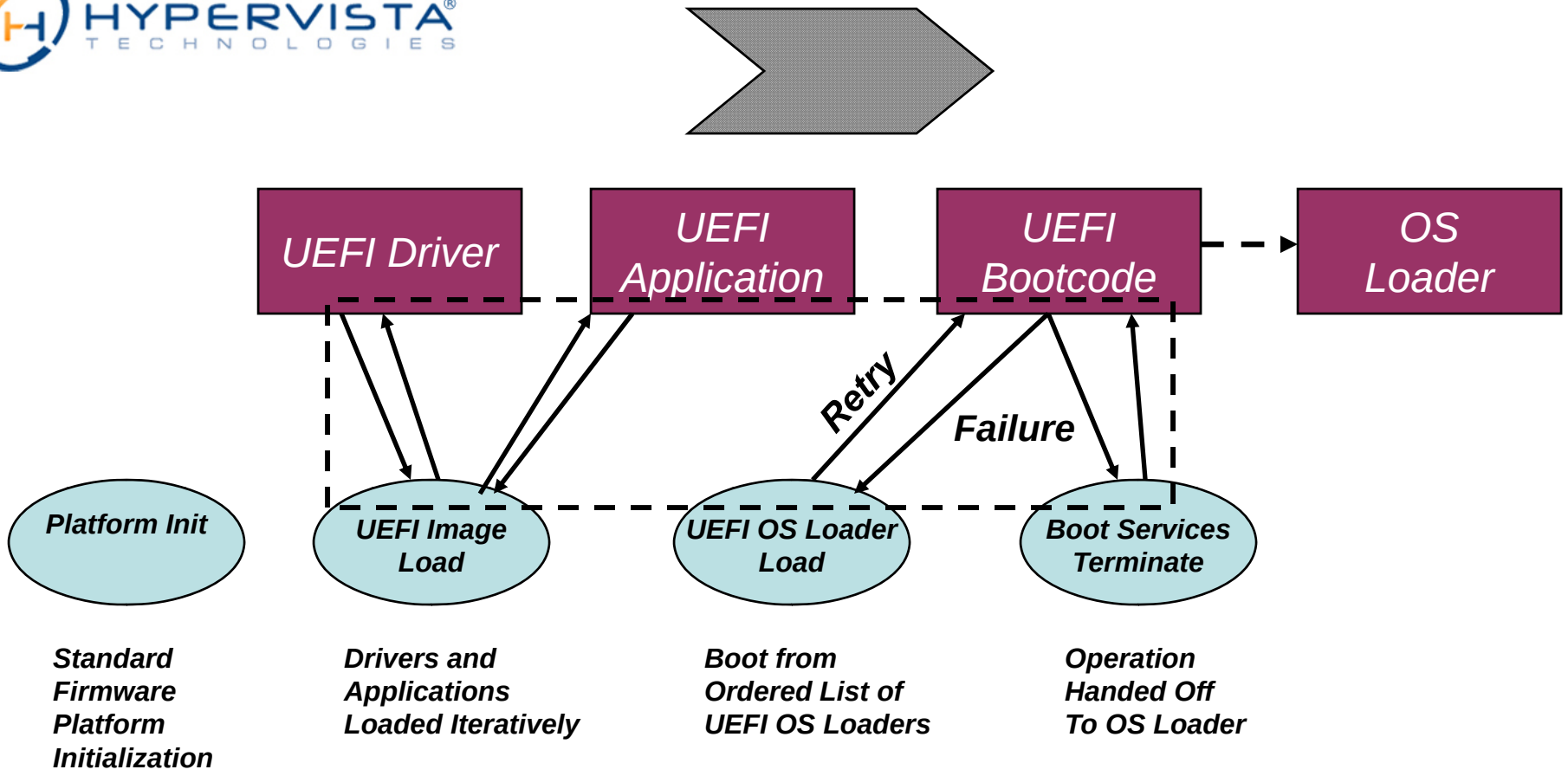
HYPERVISTA® TECHNOLOGIES

UEFI Framework



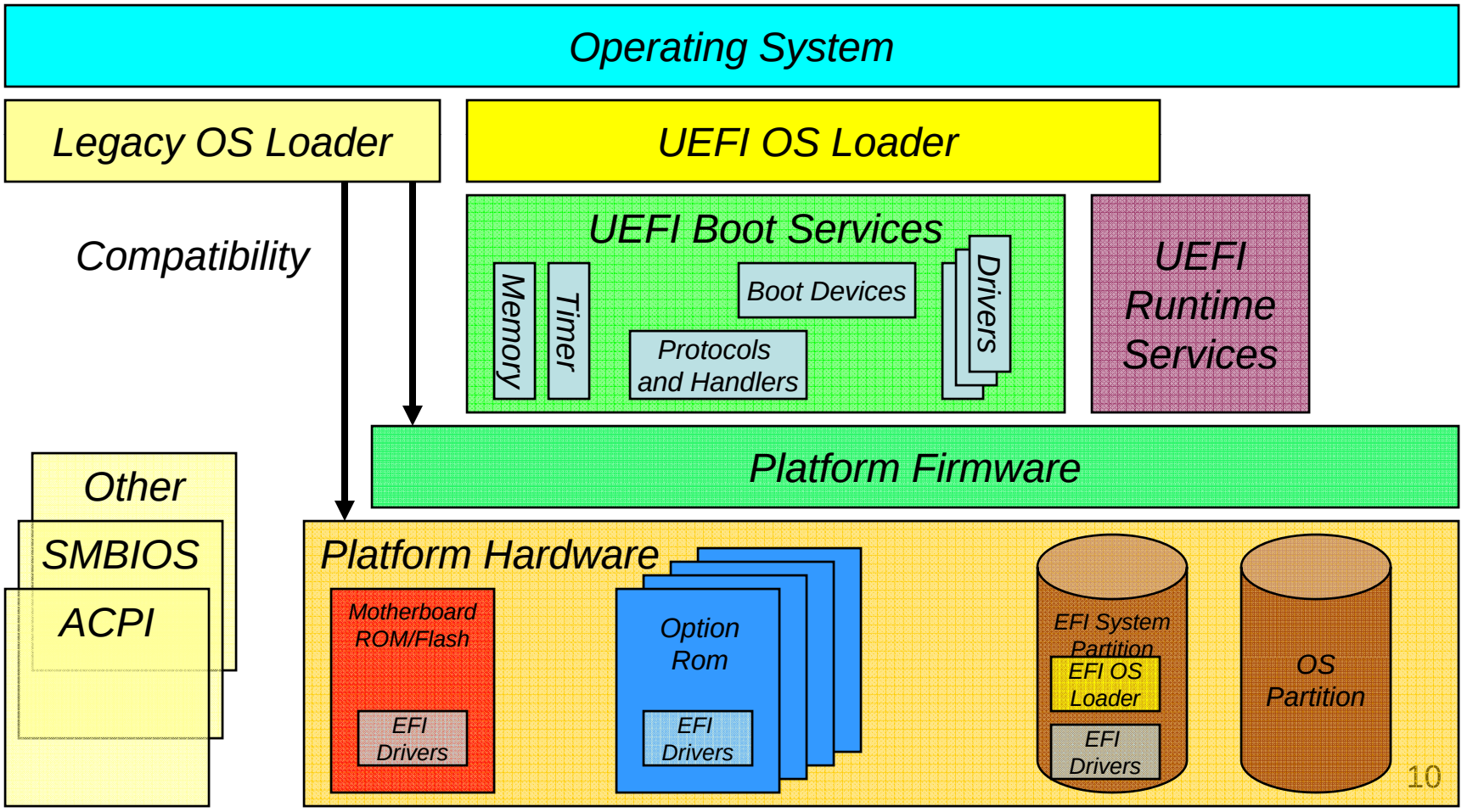

Pre Verifier
VERIFY
CPU Init
Chipset Init
Motherboard Init
Device, Bus, or Service Driver
DXE Dispatcher
Boot Services
Runtime Services
DXE Services
Exposed API
Boot Dispatcher
OS-Absent Application
Transient OS Environment
Transient OS Boot Loader
Final OS Boot Loader
Previously exposed Framework APIs now limited
ExitBootServices()
OS-Present Application
Final OS Environment
?
Security (SEC)
Pre-EFI Initialization (PEI)
Driver Execution Environment (DXE)
Boot Device Selection (BDS)
Transient System Load (TSL)
Run Time (RT)
After Life (AL)
Power On
[..Platform Initialization ..]
[..OS Boot ..]
Shutdown


Pre Verifier
VERIFY
CPU Init
Chipset Init
Motherboard Init
Device, Bus, or Service Driver
DXE Dispatcher
Boot Services
Runtime Services
DXE Services
Exposed API
Boot Dispatcher
OS-Absent Application
Transient OS Environment
Transient OS Boot Loader
Final OS Boot Loader
Previously exposed Framework APIs now limited
ExitBootServices()
OS-Present Application
Final OS Environment
?
Security (SEC)
Pre-EFI Initialization (PEI)
Driver Execution Environment (DXE)
Boot Device Selection (BDS)
Transient System Load (TSL)
Run Time (RT)
After Life (AL)
Power On
[..Platform Initialization ..]
[..OS Boot ..]
Shutdown


UEFI Driver
UEFI Application
UEFI Bootcode
OS Loader
Retry
Failure
Platform Init
UEFI Image Load
UEFI OS Loader Load
Boot Services Terminate
Standard Firmware Platform Initialization
Drivers and Applications Loaded Iteratively
Boot from Ordered List of UEFI OS Loaders
Operation Handed Off To OS Loader
Key
Specified by API
Value Added Implementation
Boot Manager
UEFI Binaries
UEFI API

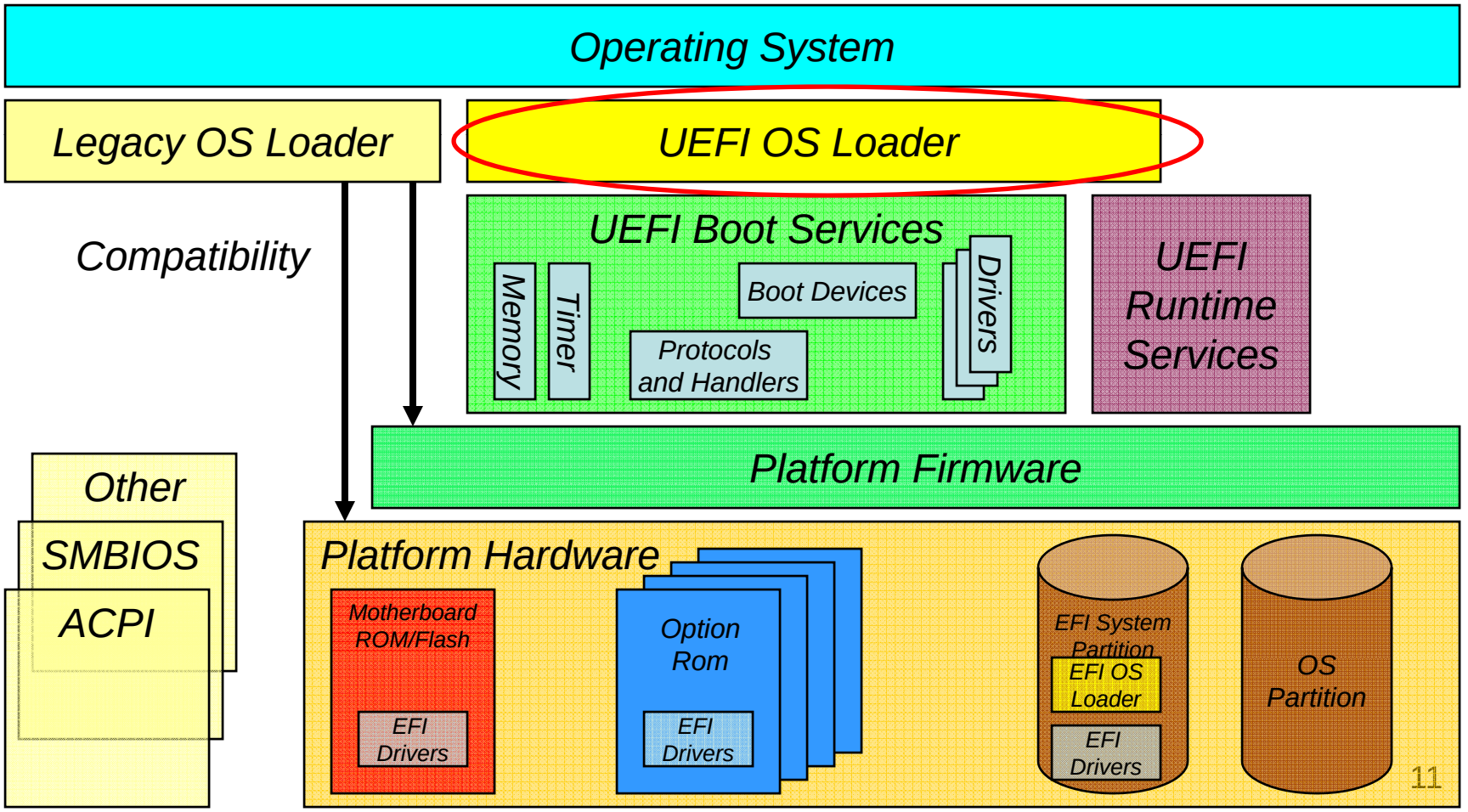
HYPERVISTA® TECHNOLOGIES

UEFI Framework



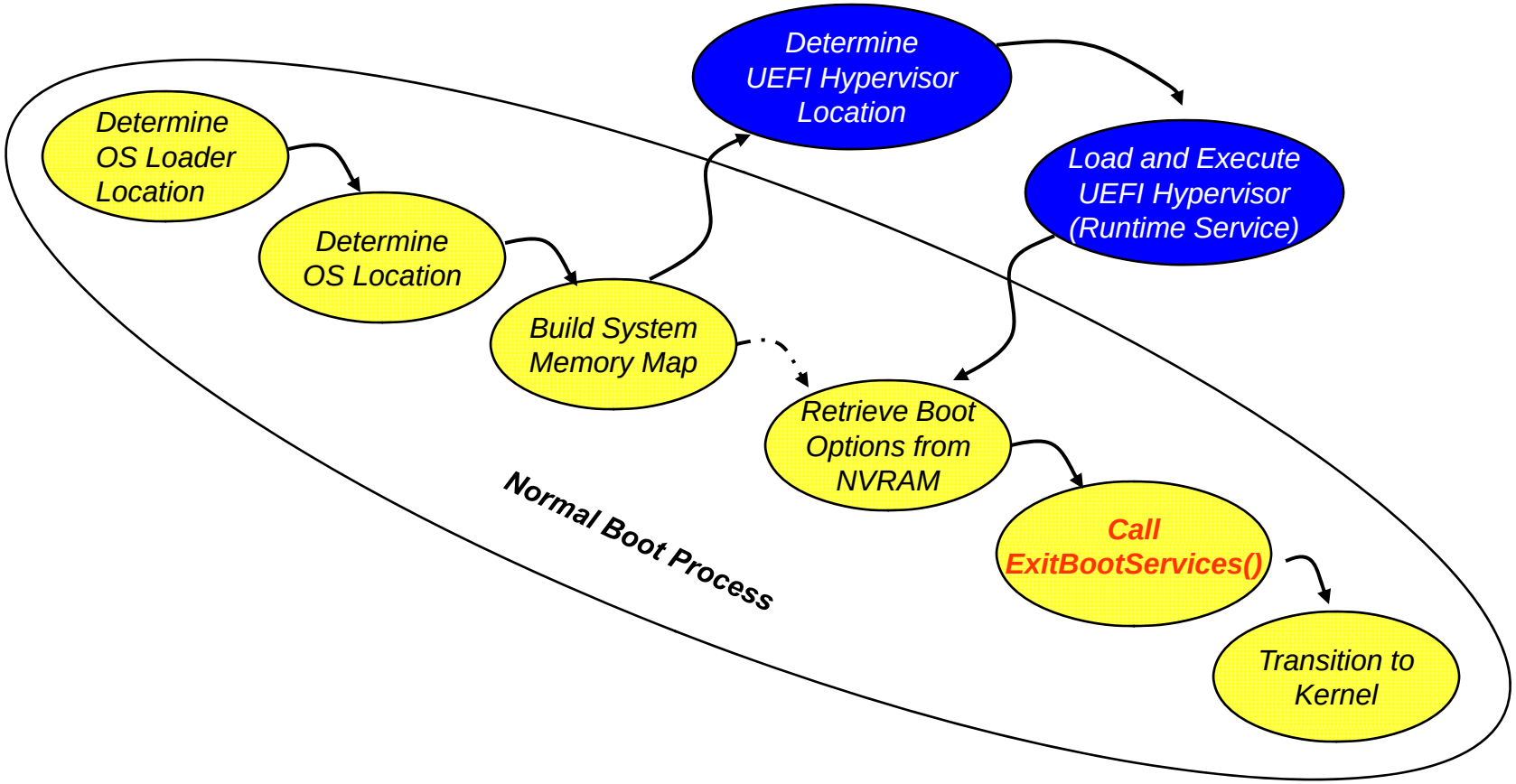
HYPERVISTA® TECHNOLOGIES

UEFI Framework



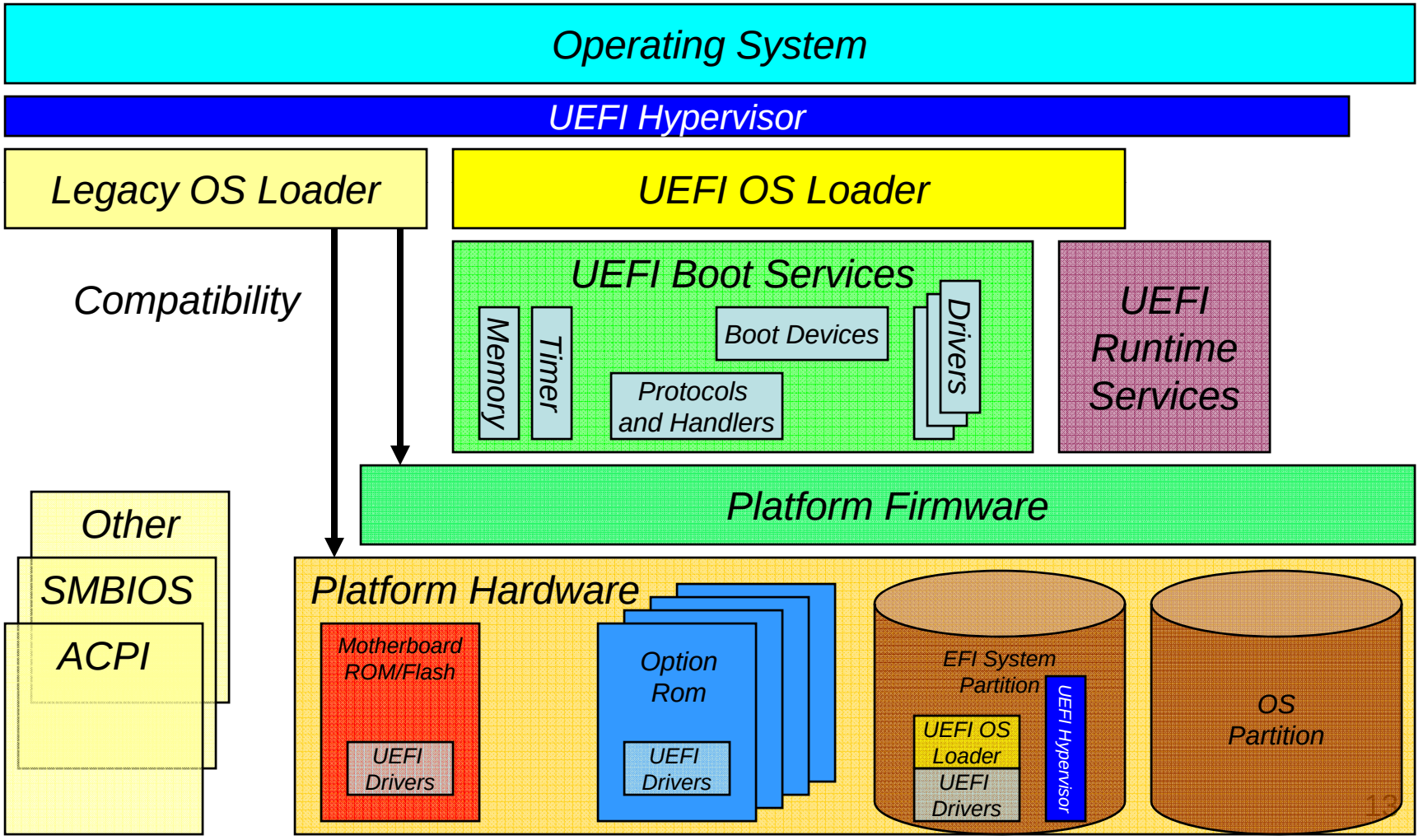
HYPERVISTA® TECHNOLOGIES

OS Loader Flow



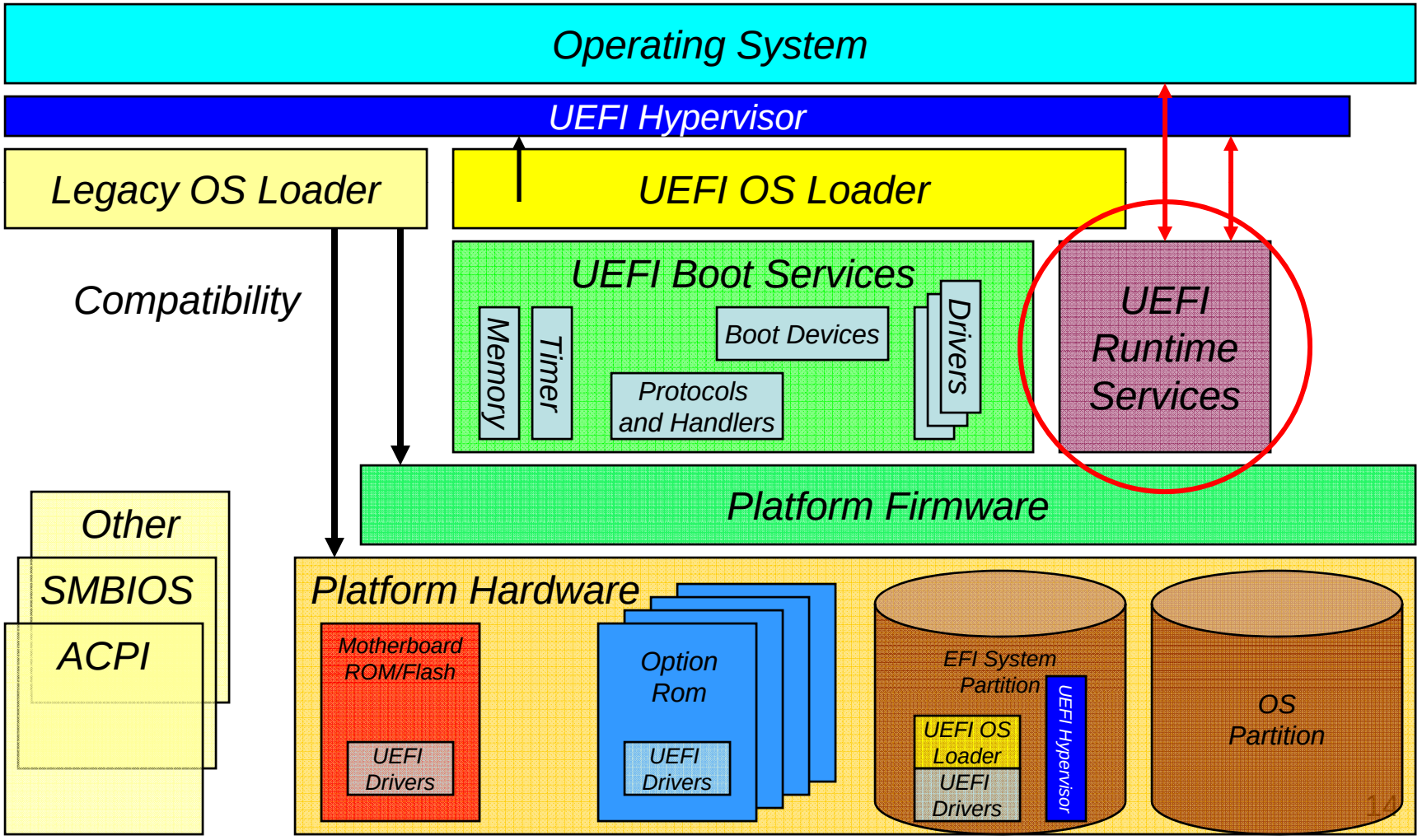
HYPERVISTA® TECHNOLOGIES

UEFI Hypervisor Framework



HYPERVISTA® TECHNOLOGIES

UEFI Hypervisor Framework



HYPERVISTA® TECHNOLOGIES

Key Concepts of the Architecture

- Objects managed by UEFI firmware
- UEFI Images
- Handle Database and protocols
- UEFI System Table
- Events
- Device Paths
- Capsules

HYPERVISTA® TECHNOLOGIES

Key Concepts of the Architecture

- Objects managed by UEFI firmware
- EFI Images
- Handle Database and protocols
- UEFI System Table
- Events
- Device Paths
- Capsules

HYPERVISTA® TECHNOLOGIES

UEFI Image

- A class of files defined by the specification that contain executable code
- UEFI images contain the old familiar PE/COFF header that defines the format of the executable code
  - UEFI uses a subset of the PE32+ image format with a modified image signature
- The header defines processor type AND the image type
  - UEFI Application
  - UEFI Boot Services Driver
  - UEFI Runtime Driver

HYPERVISTA® TECHNOLOGIES

UEFI Image

- A class of files defined by the specification that contain executable code
- UEFI images contain the old familiar PE/COFF header that defines the format of the executable code
  - UEFI uses a subset of the PE32+ image format with a modified image signature
- The header defines processor type AND the image type
  - UEFI Application
  - UEFI Boot Services Driver
  - **UEFI Runtime Driver**

HYPERVISTA® TECHNOLOGIES

Overview of Runtime Services

- Available before, during and *after* the OS is booted; after ExitBootService() is called
- UEFI Runtime Drivers are loaded in memory marked as *EfiRuntimeServiceCode*
- UEFI Runtime data structuresare marked as *EfiRuntimeServiceData*
- UEFI Runtime Drivers coexist with and can be invoked by a UEFI-aware OS

HYPERVISTA® TECHNOLOGIES

UEFI Sample Code

(Note the lack of 16-bit assembly language code!!)

HYPERVISTA® TECHNOLOGIES

```
/*++
Copyright (c) 1998  Intel Corporation
Module Name:
    rtdriver.c
Abstract:
    Test runtime driver
Revision History
--*/
#include "efi.h"
#include "efilib.h“
//
EFI_STATUS
TestRtUnload (
    IN EFI_HANDLE       ImageHandle
    );
//
CHAR16  *RtTestString1 = L"This is string #1";
CHAR16  *RtTestString2 = L"This is string #2";
CHAR16  *RtTestString3 = L"This is string #3";
EFI_GUID RtTestDriverId = { 0xcc2ac9d1, 0x14a9, 0x11d3, 0x8e, 0x77, 0x0, 0xa0,
   0xc9, 0x69, 0x72, 0x3b };
```

HYPERVISTA® TECHNOLOGIES

```
EFI_STATUS
InitializeTestRtDriver (
    IN EFI_HANDLE           ImageHandle,
    IN EFI_SYSTEM_TABLE     *SystemTable
    )
{
    EFI_LOADED_IMAGE        *Image;
    EFI_STATUS              Status;

        // Initialize the Library.

   InitializeLib (ImageHandle, SystemTable);
    Print(L"Test RtDriver loaded\n");

     // Add an unload handler

    Status = BS->HandleProtocol (ImageHandle, &LoadedImageProtocol,
   (VOID*)&Image);
    ASSERT (!EFI_ERROR(Status));
    Image->Unload = TestRtUnload;
```

HYPERVISTA® TECHNOLOGIES

```
// Add a protocol so someone can locate us

Status = LibInstallProtocolInterfaces (&ImageHandle, &RtTestDriverId, NULL,
    NULL);
     ASSERT (!EFI_ERROR(Status));

     // Modify one pointer to verify fixups don't reset it

     Print(L"Address of RtTestString3 is %x\n", RtTestString3);
     Print(L"Address of RtTestString3 pointer is %x\n", &RtTestString3);
     RtTestString3 = RtTestString2;
     return EFI_SUCCESS;
}

EFI_STATUS
TestRtUnload (
     IN EFI_HANDLE       ImageHandle
     )
{
     DEBUG ((D_INIT, "Test RtDriver unload being requested\n"));
     LibUninstallProtocolInterfaces (ImageHandle, &RtTestDriverId, NULL,
    NULL);
     return EFI_SUCCESS;
```

HYPERVISTA® TECHNOLOGIES

UEFI Hypervisor

- Our challenge was to convert our x64 Windows hypervisor device driver to a UEFI Runtime device driver
- We identified the MSI P45 Platinum motherboard as UEFI 2.0 compliant
- We used the Tianocore UEFI 2.0.0.1 SDK

HYPERVISTA® TECHNOLOGIES

DEMO

HYPERVISTA® TECHNOLOGIES

Resources

- UEFI 2.0 Specification – http://www.uefi.org

- "*Beyond BIOS – Implementing the Unified Extensible Firmware Interface with Intel's Framework*" by Zimmerman, Rothman, and Hale - Intel® Press - http://www.intel.com/intelpress/

HYPERVISTA® TECHNOLOGIES

Questions ?

Email: don AT hypervista-tech.com