presented by

ami™

power • manage • secure

Understanding UEFI and PI Architectural Events

UEFI 2021 Virtual Plugfest

Meet the Presenter

Felix Polyudov

Director of Firmware Core Architecture, AMI

What is UEFI Event?

- Event is a UEFI callback-based binary-to-binary communication mechanism
  - Like OS event objects
  - Adopted to UEFI single-threaded environment
  - Facilitated by UEFI Boot Services (UEFI 2.9 spec., ch. 7.1)
- Event Roles
  - Actor: detects underlying condition
  - Reactor(s).
    - Can get notified via callback when even is signaled
    - Can query event status
- PI specification defines simplified event-like callback mechanisms for PEI and MM environments
- PI specification extends list of standard UEFI events

Let’s Dot the i's and Cross the t's

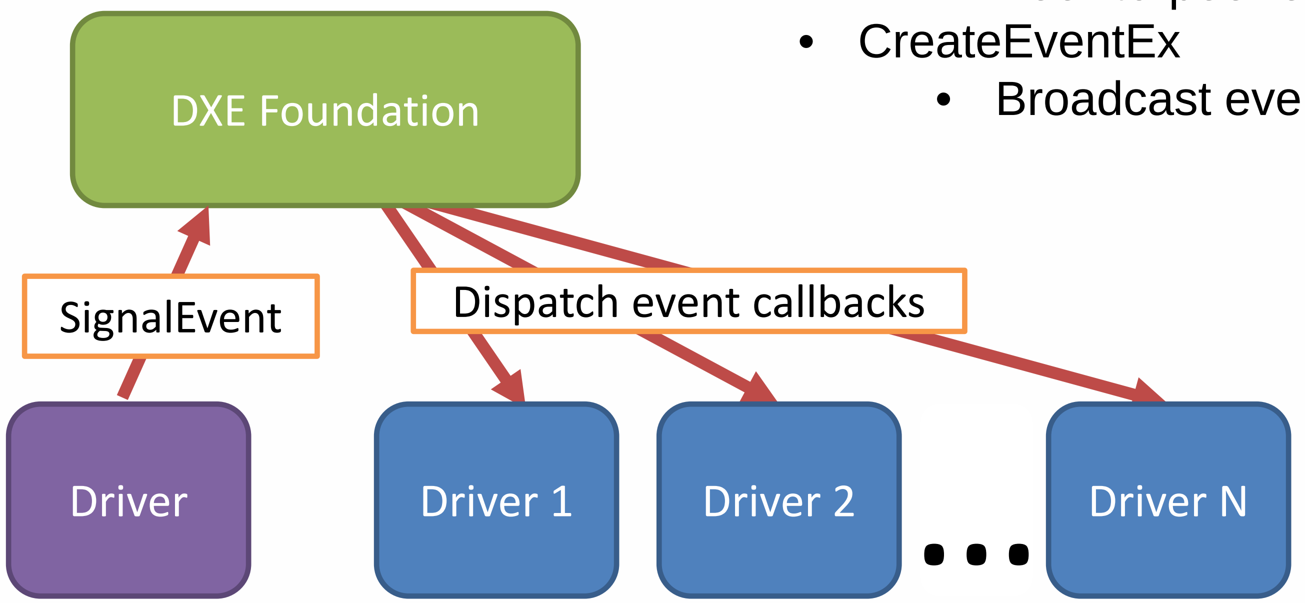
- UEFI vs PI
  - UEFI is a Firmware to OS Interface. There are multiple ways to architect a UEFI solution
  - PI is the mainstream UEFI implementation, but it’s just one of the ways to implement UEFI
  - The presentation covers both domains, but makes it clear which domain is implied
- Events vs UEFI Events
  - Unless noted otherwise, term “event” is used by the presentation in a broad sense referring to all kinds of UEFI and PI callback mechanisms
- MM vs SMM: what’s the difference?
  - Spec view: SMM and MM used by the PI spec interchangeably
    - SMM is an older name that was later replaced with a more architecture-neutral MM. However, SMM is still used here and there.
  - Views on the ground
    - Some people use SMM and MM as a references to IA and ARM MM implementations
    - Some people use SMM to refer to a Traditional MM implementation and MM to refer to a Standalone MM implementation

Functional UEFI Event Classes

- Private events
  - Events used by drivers to implement driver specific logic
- Protocol Specific Events
  - Used for a protocol-specific notifications to protocol consumers
- Timer Events
  - Timed one-shot or periodic callbacks
  - UEFI Polling Mechanism
  - PI 1.7 introduced a PEI timed callback mechanism (Delayed Dispatch PPI)
- Protocol Installation Notifications
  - Private events can be registered with DXE Foundation to get signaled when protocol with the specific GUID is installed
  - PI specification defines a simplified (non-UEFI event based) protocol installation notifications for PEI and MM environments
- System Events (UEFI and PI)
  - System wide special conditions
  - Boot Flow Events (important subclass of the System Events)
    - Reaching certain point in the boot process
    - Some of them are implemented as Protocol Installation Notifications

UEFI Event Notification Types

Notify on Signaling



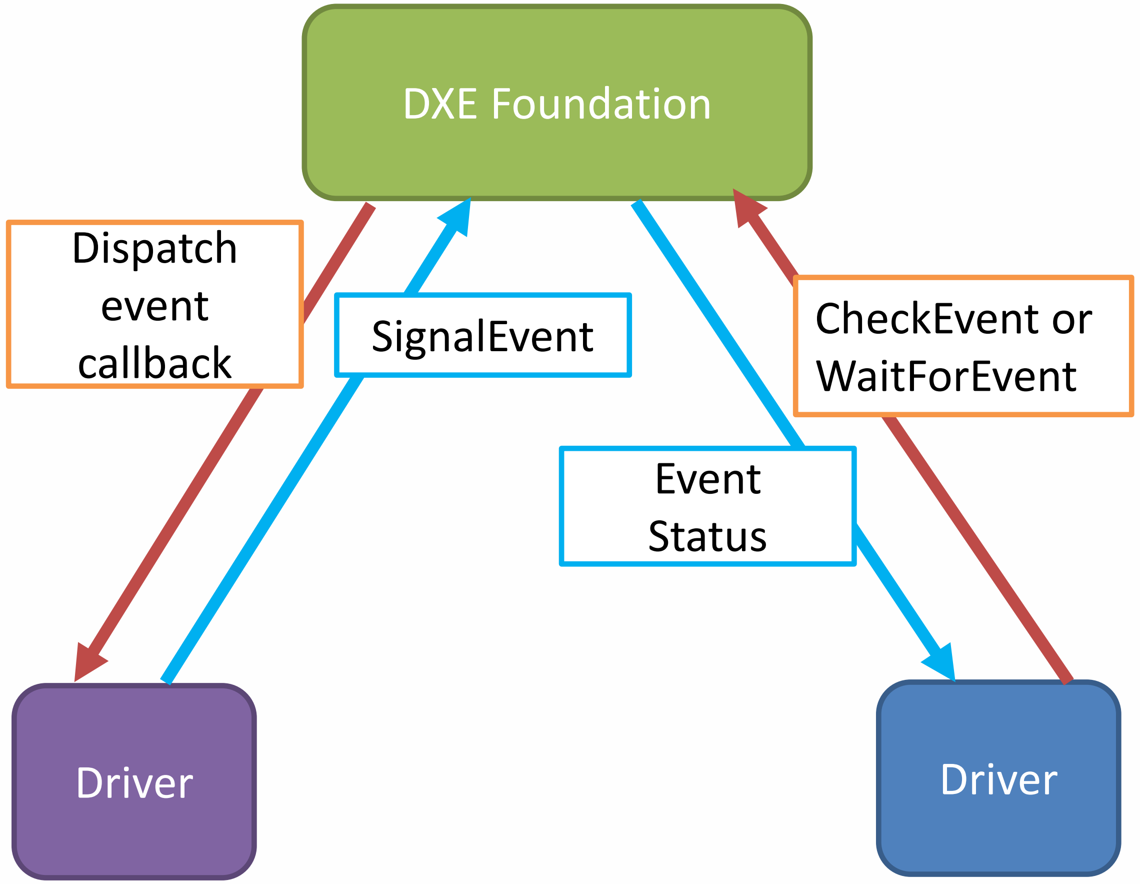
Callbacks are dispatched based on their priority.
Three priority levels (TPLs) are defined:

- Callback (default)
- Notify (elevated)
- High (highest; reserved for use by the firmware)

Events are created with

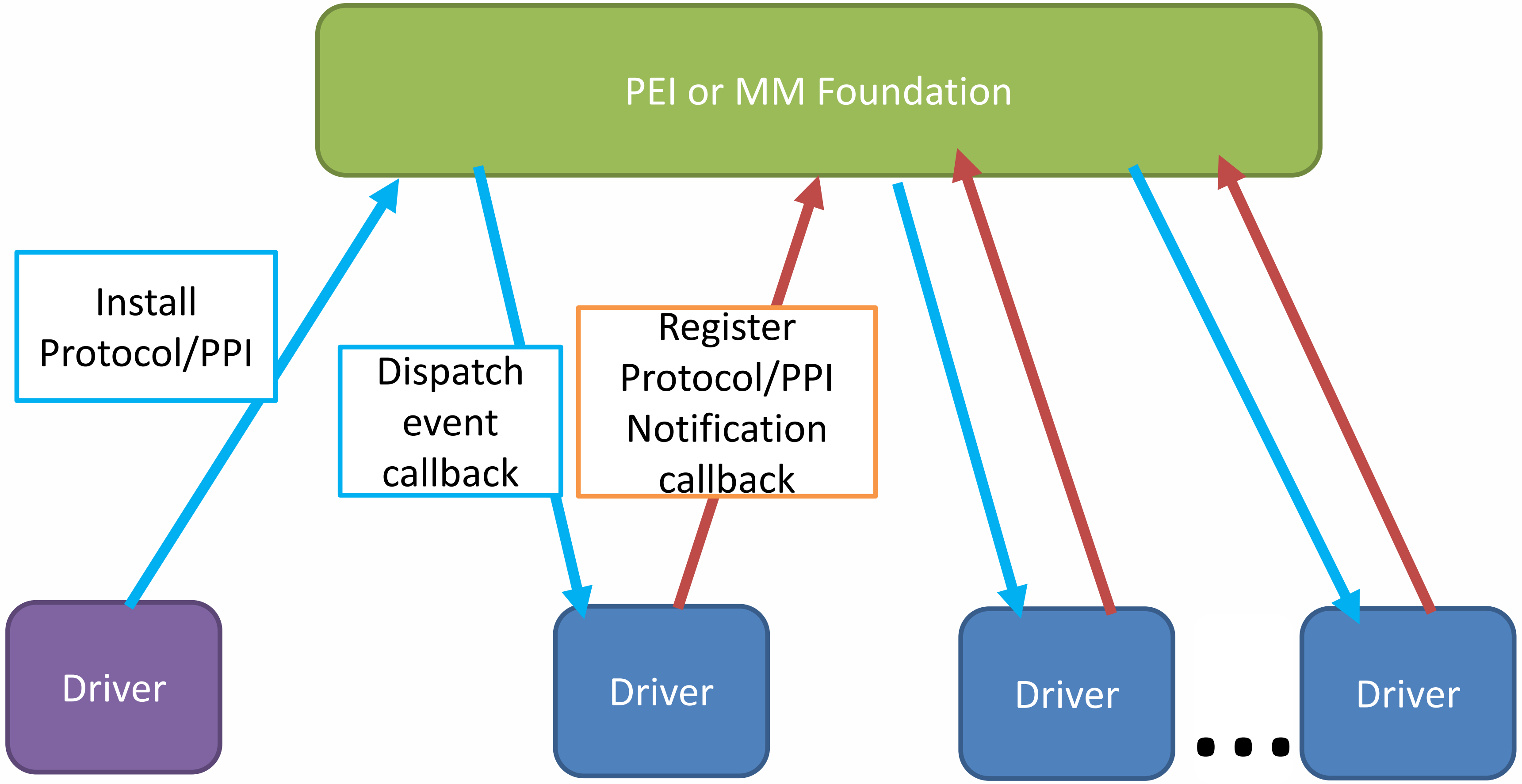
- CreateEvent
  - Peer-to-peer events
- CreateEventEx
  - Broadcast events

Notify on Wait or Check



Legend:
🟪 - Event actor
🟦 - Event reactors

MM and PEI Notification Callbacks



- Events are modeled by Protocols/PPIs with NULL interfaces
- Akin to UEFI Protocol Installation Notifications
- No priority levels*

Legend:

🟪 - Event actor

🟦 - Event reactors

(*) – PEI has an indirect way to introduce two priority levels. See "PEI Notification Types" slide below for details.

Playing Safe with the Events

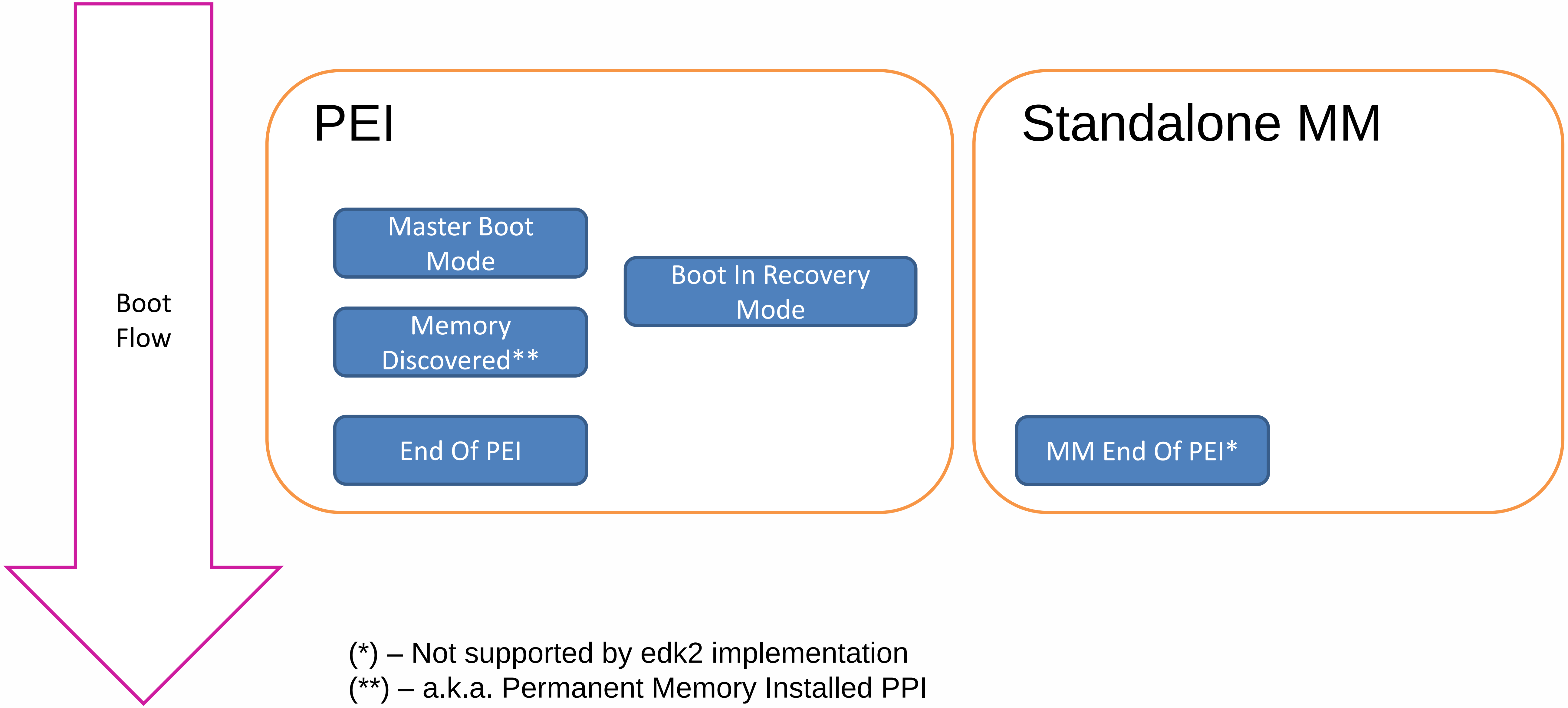
- Your code may be interrupted by the event callbacks
  - Use UEFI TPL API to protect critical portions of the code against reentrancy
- Don’t assume a specific order of callback dispatching
  - UEFI specification does not define execution order of the callbacks with the same TPL
  - PI specification does not define execution order of the callbacks
- Never break TPL restrictions (UEFI spec., ch. 7.1, table 7-3)
  - UEFI specification defines the highest priority level at which each interface can be used
- Use the lowest TPL possible
  - If your event handler is not on TPL Callback, you should know why
- Don’t overburden the system with large number of timer events
  - UEFI specification does not prescribe timer resolution. It is implementation specific.
  - Large number of timer events can reduce system performance.
- Don’t overuse Protocol Installation Notification Callbacks
  - In UEFI drivers prefer driver model over protocol callbacks to deal with the protocols of the managed device
  - In PI code prefer DepEx over protocol callbacks

System Events

- Memory Map Change (*UEFI*)
  - Signaled whenever memory map changes
  - Not fully supported by the edk2 implementation
- Reset System (*UEFI*)
  - Signaled when ResetSystem() is invoked, and the system is about to be reset(only prior to ExitBootServices() invocation).
  - Not supported by the edk2 implementation
- DXE Dispatch (*PI*)
  - Internal pluming used by DXE and MM Foundations
- Boot Flow
  - To be discussed…

Boot Flow Events

PEI Boot Flow Events



(*) – Not supported by edk2 implementation
(**) – a.k.a. Permanent Memory Installed PPI

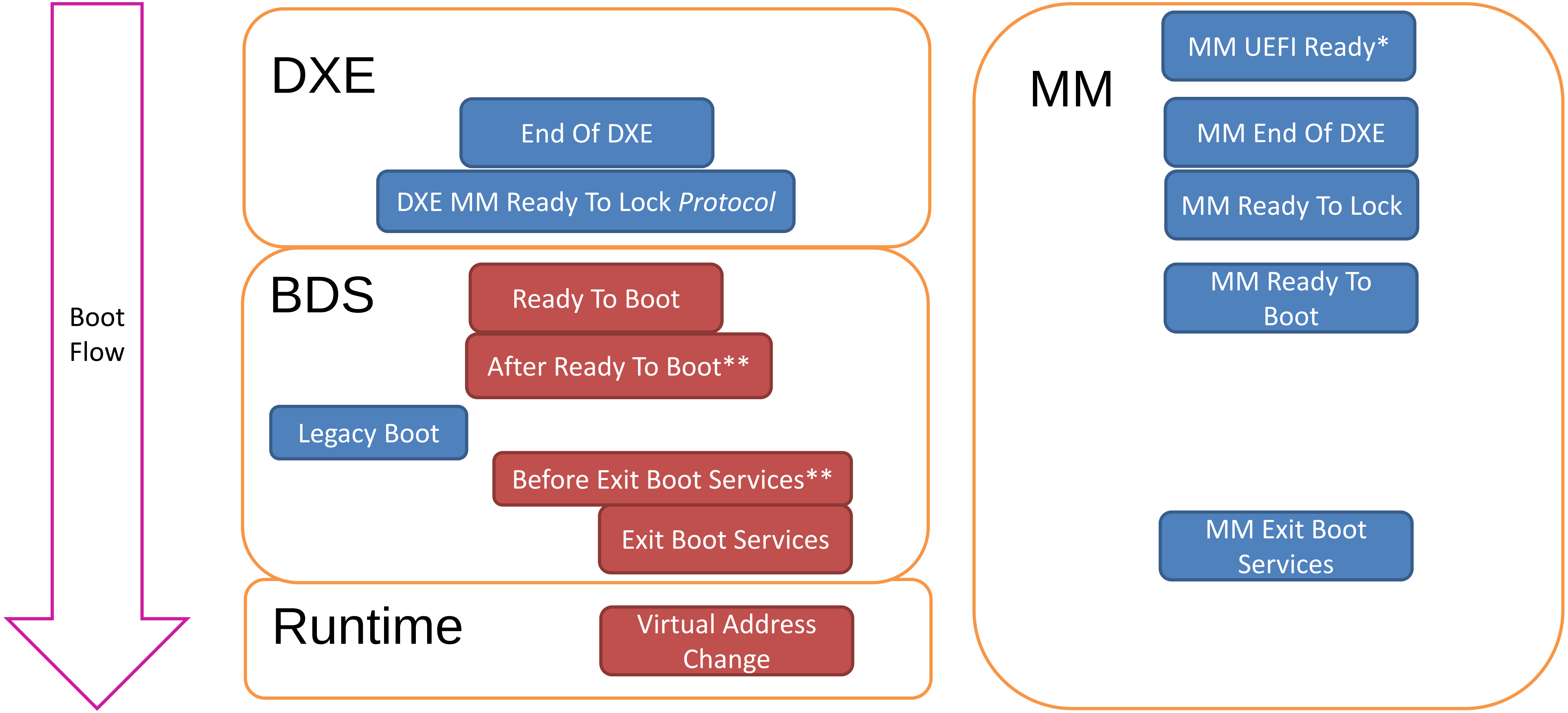
Fun Facts and Things to Note

- Since PEI Boot Flow events are PPI notifications, they can be used as callbacks or as dependencies
- Master Boot Mode PPI is typically installed prior to Memory Discovered PPI, but it’s not required by the PI spec
- Boot in Recovery Mode PPI can be installed at any point in the PEI execution before DxeIpl->Entry()
- According to the spec, if permanent and temporary RAM can co-exist(mainstream scenario on IA platforms), temporary RAM (CAR) should be disabled after Memory Discovered PPI installation; however, edk2 implementation disables CAR before the PPI installation
- Prefer dispatch notifications over callbacks notifications

PEI Notification Types

- PI Specification defines two types of PPI installation notifications(PI 1.7A, vol. 1, ch. 4.2, 7.4):
  - Callback notification
    - Callback functions are called right after the PPI installation (before returning from InstallPpi PEI service)
  - Dispatch notification
    - Invocation of callback functions is deferred until PEIM that installed the PPI returns control back to PEI Foundation
- Dispatch notification type was originally intended to optimize stack usage by reducing number of nested stack frames
  - Thanks to hardware advances, stack overflow is not a typical problem, however, it still occasionally happens. For example, it may happen
    - On special boot paths
      - On S3 resume due to reduced amount of available memory
      - On Recovery due to increased memory usage
    - On feature rich firmware configurations
    - On embedded servers where small core hardware meets server feature set
- Dispatch notifications can be used as a control flow tool
  - Dispatch notifications are guaranteed to by invoked after all the callback notifications has been invoked

DXE,BDS, and RT Boot Flow Events



Boot Flow

DXE

End Of DXE

DXE MM Ready To Lock *Protocol*

BDS

Ready To Boot

After Ready To Boot**

Legacy Boot

Before Exit Boot Services**

Exit Boot Services

Runtime

Virtual Address Change

MM

MM UEFI Ready*

MM End Of DXE

MM Ready To Lock

MM Ready To Boot

MM Exit Boot Services

Legend:

🟥 - Events defined by UEFI spec

🟦 - Event defined by the PI spec

(*) – Not supported by the edk2 implementation

(**) – Introduced in UEFI 2.9

Event Pairs

- UEFI and PI specs define 3 event pairs (events signaled sequentially one after another) to implement smooth transition between the phases
  - First event presents the last chance to access system interfaces and/or to change system configuration before the transition
  - Second event can be used by handlers that facilitate the transition and by handlers that are interested in the finalized pre-transition configuration

UEFI and PI Event Pairs

- End-of-DXE, MM Ready-to-lock (*PI*)
  - End-of-DXE: last chance to use services that are to be disabled and to modify hardware state that is to be locked
  - MM Ready-to-lock: switch hardware into a secure state (e.g., lock SPI writes), disable or harden software interfaces (e.g., stop registration of new MM handlers, lock sensitive UEFI variables)
- Ready-to-Boot, After-Ready-to-Boot (*UEFI*)
  - Ready-to-Boot: last chance to change system configuration before the boot
  - After-Ready-to-Boot: process pre-boot configuration (e.g., finalize SMBIOS and/or ACPI tables, send config data to BKC)
- Before-Exit-Boot-Services, Exit-Boot-Services (*UEFI*)
  - Before-Exit-Boot-Services: last chance to use the boot services
  - Exit-Boot-Services: transition a driver to runtime

Fun Facts and Things to Note

- Don’t take event name literally
  - End-of-DXE is not necessarily the end of DXE phase
    - According to PI spec the event is signaled before “third party extensible modules such as UEFI drivers and UEFI applications are executed”. So, a portion of BDS may run prior to this event
  - MM is typically locked way before MM Ready-to-Lock is signaled
- DXE MM Ready-to-lock is a protocol
  - Unlike the other non-MM boot flow events, this one is implemented as a protocol
- Relative order of peer DXE and MM events is not defined
  - DXE Ready-to-Boot handlers may run before or after MM Ready-to-Boot handler
- MM Ready-to-Boot and MM Exit-Boot-Services are beyond the platform trust boundary
  - MM code should work properly if events are never signaled
  - Any data coming form outside the MM environment should be treated as untrusted
- Ready-to-Boot can happen more than once
- Services called by Exit-Boot-Services callbacks may exhibit a boot time or a runtime behavior
- Services called by Virtual-Address-Change callbacks may not work as intended if service being called has already transitioned to the virtual address memory map

Questions?

More Questions?

Following today’s webinar, join the live, interactive WebEx Q&A for the opportunity to chat with the presenter

Visit this link to attend: https://bit.ly/3aob7O7
Meeting number (access code): 182 688 4062
Meeting password: UEFIForum (83343678 from phones and video systems)

Thanks for attending the UEFI 2021 Virtual Plugfest

For more information on UEFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by

ami™

power • manage • secure