EDK II Topology

SMM

All core package code referenced in this document is located in the **GitHub EDK II repository**.

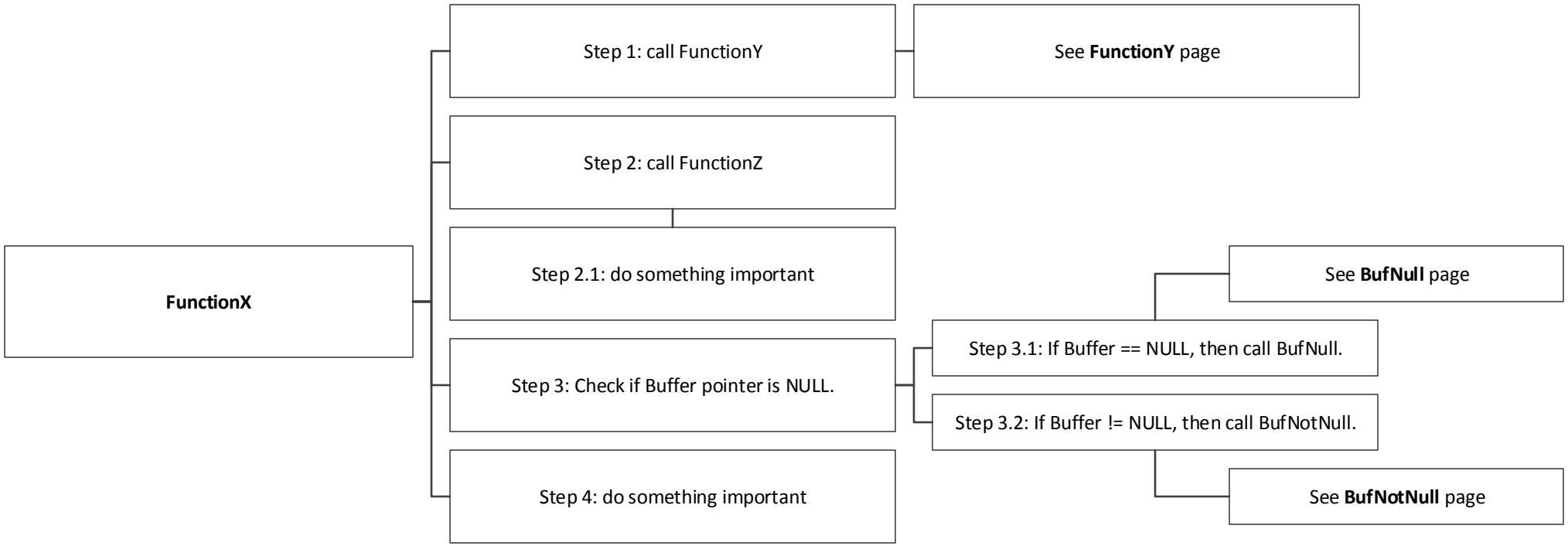
For more in-depth information about EDK II, visit the **Intel® Firmware: Beyond BIOS** page.

Visit TianoCore.org for more **EDK II documentation** and **EDK II projects.**

The function being discussed always starts in the only box on the far left. Boxes represent steps in a function, a branch evaluation in a funtion, or a note to see details on another page about a function being called at a step. Connectors between boxes indicate code flow (who called what) and should be read left-to-right, top-to-bottom. Text in each box will indicate if it's a call, a branch evaluation, or a note. This format was chosen to fit important function details on 1 page.

For the example below, the equivalent C code (note connectors from Step 1-4 to FunctionX):

```
FunctionX() {
        FunctionY();    // see details on FunctionY page
        FunctionZ();    // Step 2; Step 2.1 is in FunctionZ() and is listed because it is important; note the connector between Step 2 and 2.1
        If (Buffer == NULL)     // Step 3
                BufNull();      // Step 3.1
        else
                BufNotNull();   // Step 3.2
        Step 4
}
```



How protocol services are defined so one can find the protocol code to examine it

Summary: if you want to examine the code for a protocol function, you should find the structure definition for the protocol, then find the declaration of the structure, then find the structure member that corresponds with the protocol in the structure definition because they may have different names. EFI_BOOT_SERVICES defines LocateProtocol, mBootServices is of type EFI_BOOT_SERVICES, and the structure member CoreLocateProtocol corresponds with the structure definition LocateProtocol.

MdePkg/Include/UefiSpec.h defines EFI_BOOT_SERVICES structure, and has structure members for protocol services (LocateProtocol, InstallProtocolInterface, etc). **MdeModulePkg/Core/Dxe/DxeMain.c** has a variable mBootServices of type EFI_BOOT_SERVICES. mBootServices sets function pointers for functions such as LocateProtocol to CoreLocateProtocol and InstallMultipleProtocolInterfaces to CoreInstallMultipleProtocolInterfaces. These functions are defined in **MdeModulePkg/Core/Dxe/Handle.c**.

How protocols are loaded from flash into memory

Summary: drivers are loaded from flash into memory by some mechanisms into a linked list during the PEI phase.

MdeModulePkg\Core\Dxe\DxeMain.c has DxeMain function which is called when the DXE Core driver is loaded. MdeModulePkg\Core\Dxe\DxeMain.inf has MODULE_TYPE=DXE_CORE and ENTRY_POINT=DxeMain. The end of DxeMain calls CoreInstallMultipleProtocolInterface with the GUID for the HOB that was populated with drivers from the flash part during PEI. PEI phase calls ReadSection (associated with FvReadFileSection in **Universal/FirmwareVolume/FwVolDxe/FwVol.c**), which eventually gets to a call to LocateProtocol with gEfiDecompressProtocolGuid as a parameter.

Platform actions required for DXE_SMM_DRIVER to initialize SMM

The following is a partial list of items that platform code must perform in a driver with MODULE_TYPE=DXE_SMM_DRIVER

- Allocate SMRAM for SMM Save State of all processors.
- Set SMBASE for all processors.
- Relocate SMBASE for all processors.
- Copy SMI Handler to each processor SMBASE.
- Install SMM Configuration protocol in SMRAM space to provide SMRAM Region data and RegisterSmmEntry function for registering SMM Entry Point.
- Install various protocols: sync CPUs entering SMM, reading SMM Save State, etc.
- Register event notification for SMM Ready To Lock.

See **PI Volume 4 Chapter 5.5 "EFI_SMM_CONFIGURATION_PROTOCOL"** for details.

Platform actions required before SmmIplEntry() is executed

Actions associated with **EFI_SMM_ACCESS2_PROTOCOL**

- Determine memory ranges to report as SMRAM ranges. This is usually done after memory init by building a HOB for memory ranges.
- Install SmmAccess2 protocol to provide functions Open, Close, Lock, GetCapabilities.

See **PI Volume 4 Chapter 5.3 "EFI_SMM_ACCESS2_PROTOCOL"** for details.

Actions associated with **EFI_SMM_CONTROL2_PROTOCOL**

- Install SmmControl2 protocol to provide functions Trigger, Clear.
- Disable pending SMIs.

See **PI Volume 4 Chapter 5.4 "EFI_SMM_CONTROL2_PROTOCOL"** for details.

Architecture

DXE

SMM

EDK II Topology - SMM

gST

gSmst

NonUDK
MRC

Sync SMI

Async SMI

EFI_SMM_ACCESS2_PROTOCOL
Open()
Close()
Lock()
GetCapabilities()

Notifications:
ReadyToLock
EndOfDxe
ReadyToBoot
ExitBootServices
...

UDK
Ia32FamilyCpuPkg/
SmiEntry.asm

NonUDK
Chipset

UDK
MdeModulePkg/Core/
PiSmmCore/
PiSmmCore.c
PiSmmIpl()

ExecuteSmmCore
FromSmram()

UDK
MdeModulePkg/
Core/PiSmmCore/
PiSmmCore.c.
SmmMain()

UDK
MdeModulePkg/Core/
PiSmmCore/
PiSmmCore.c
SmmEntryPoint()

EFI_SMM_CONTROL2_PROTOCOL
Trigger()
Clear()

EFI_SMM_BASE2_PROTOCOL
GetSmstLocation()
InSmm()
...

SmiMange()

EFI_SMM_CONFIGURATION_PROTOCOL
RegisterSmmEntry()
SmramReservedRegions()

UDK
MdeModulePkg/Universal/
Variable/RuntimeDxe/
VariableSmm.c
SmmVariableHandler()

VariableServiceSetVariable()
VariableServiceGetVariable()

Flash

UDK
Ia32FamilyCpuPkg/
PiSmmCpuDxe

UDK
Ia32FamilyCpuPkg/
PiSmmCommunication

DXE Runtime

EFI_SMM_COMMUNICATION_PROTOCOL
Communicate()

SMM Communication Buffer
Comm Header
Variable Comm Header
FunctionId
ReturnStatus
VariableAccess Data
Guid
Dataize
NameSize
Attribures
Name[]
Data[]

OS
Runtime

Variable Services
GetVariable
SetVariable
GetNextVariableName
QueryVariableInfo

UDK
MdeModulePkg/Universal/
Variable/RuntimeDxe/
VariableSmmRuntimeDxe.c

RuntimeServiceSetVariable
RuntimeServiceGetVariable

gRT

KEY

Non UDK module

UDK SMM Driver

UDK DXE Runtime Driver

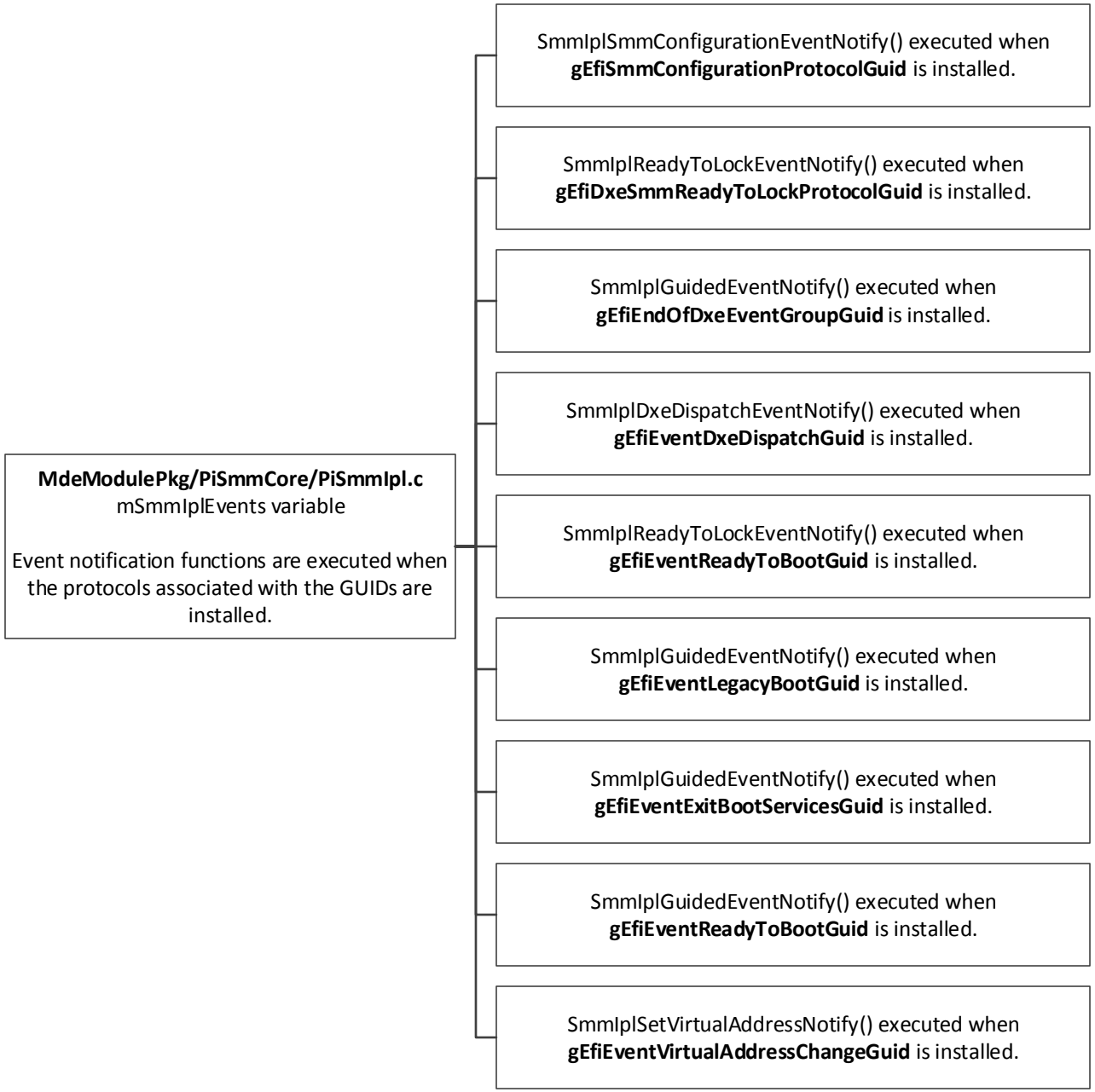
UDK SMM core

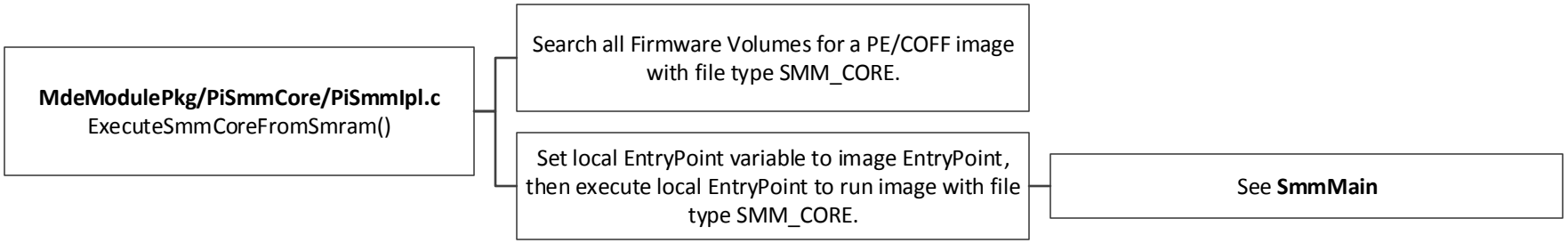
January 2016 Rev 0.5

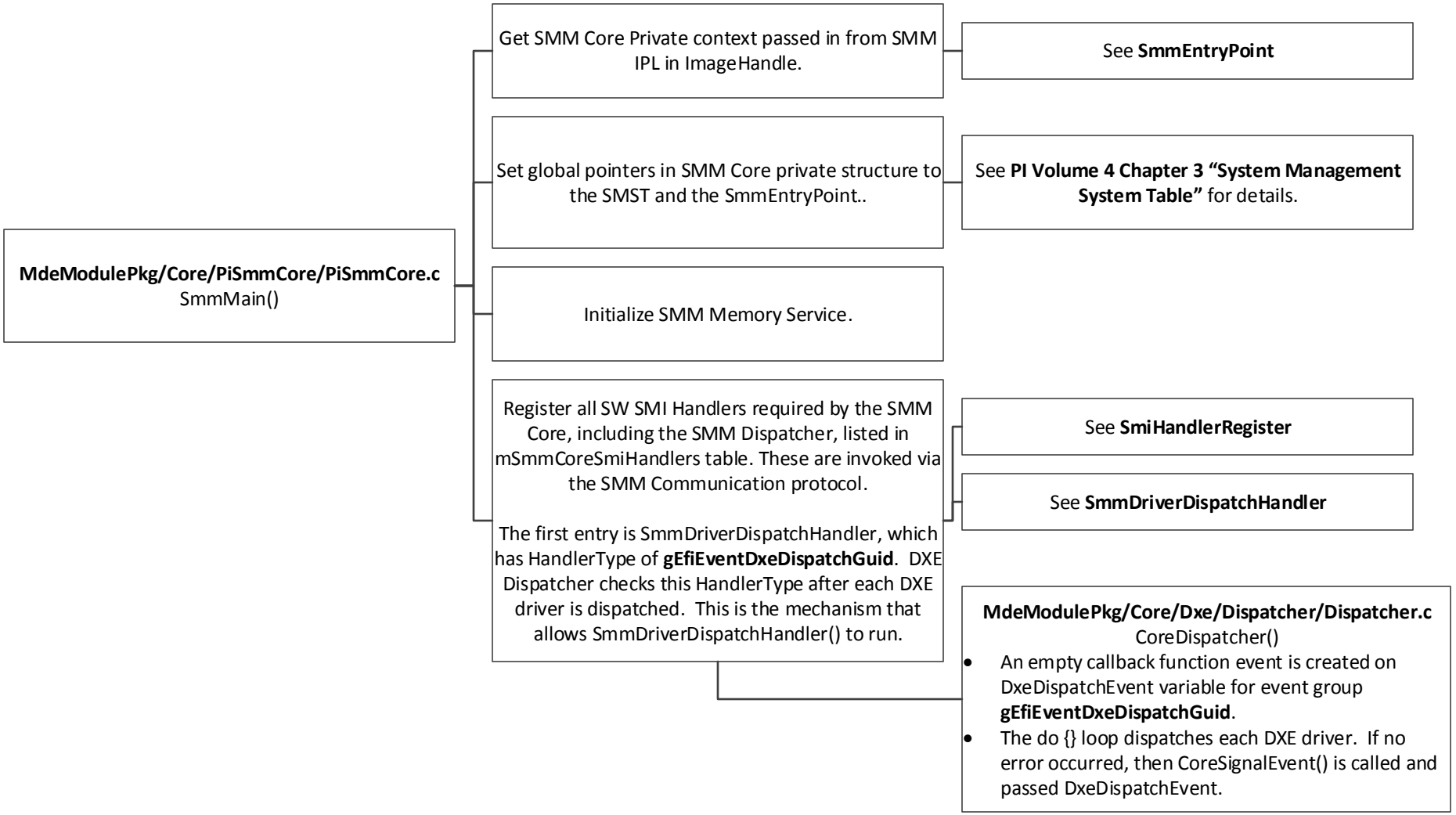
See **Platform** page for steps the platform code must take to initialize SMM.

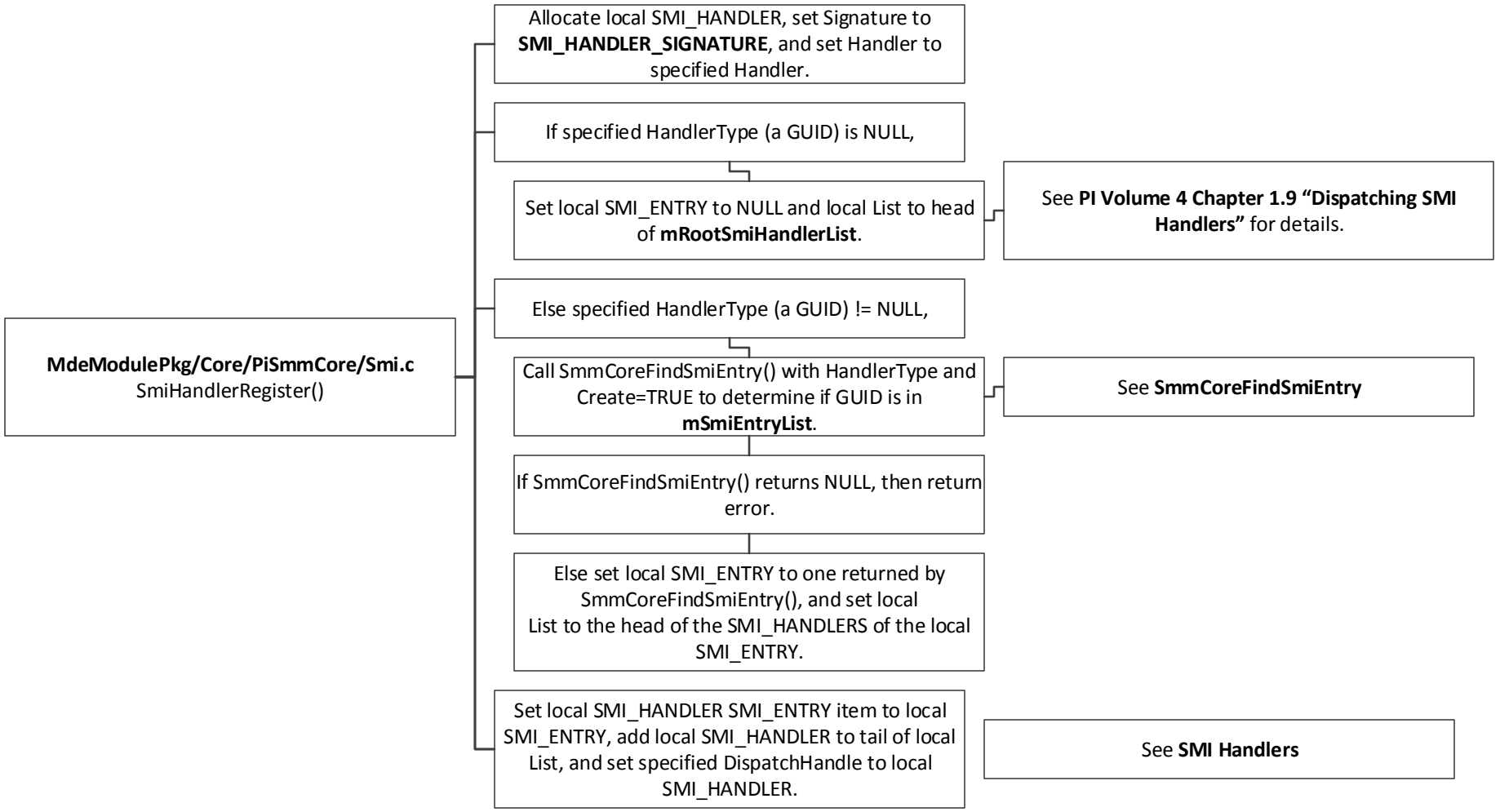
MdeModulePkg/PiSmmCore/PiSmmIpl.c
MdeModulePkg/PiSmmCore/PiSmmIpl.inf
MODULE_TYPE=DXE_RUNTIME_DRIVER
ENTRY_POINT=SmmIplEntry

- Get SmmAccess2 protocol.
- Get SmmControl2 protocol.
- Get SMRAM ranges with GetFullSmramRanges().
  - Get SmmConfiguration protocol and use mSmmAccess to call GetCapabilities().
- Open SMRAM ranges with SmmAccess2 Open().
- Set SMRAM cache type.
- Load SMM Core into SMRAM and execute it from SMRAM.
  - See **SMM Core**
- Install SMM Base2 protocol and SMM Communication protocols.
- Create the set of protocol and event notifications that SMM IPL requires. Registers **mSmmIplEvents** table, which includes **gEfiEventDxeDispatchGuid** event that is associated with SmmIplGuidedEventNotify() which is executed each time DXE Dispatcher is finished; see **MdeModulePkg/Core/Dxe/Dispatcher/Dispatcher.c** CoreDispatcher().
  - See **SMM IPL Events**


MdeModulePkg/PiSmmCore/PiSmmIpl.c
mSmmIplEvents variable
Event notification functions are executed when the protocols associated with the GUIDs are installed.
SmmIplSmmConfigurationEventNotify() executed when gEfiSmmConfigurationProtocolGuid is installed.
SmmIplReadyToLockEventNotify() executed when gEfiDxeSmmReadyToLockProtocolGuid is installed.
SmmIplGuidedEventNotify() executed when gEfiEndOfDxeEventGroupGuid is installed.
SmmIplDxeDispatchEventNotify() executed when gEfiEventDxeDispatchGuid is installed.
SmmIplReadyToLockEventNotify() executed when gEfiEventReadyToBootGuid is installed.
SmmIplGuidedEventNotify() executed when gEfiEventLegacyBootGuid is installed.
SmmIplGuidedEventNotify() executed when gEfiEventExitBootServicesGuid is installed.
SmmIplGuidedEventNotify() executed when gEfiEventReadyToBootGuid is installed.
SmmIplSetVirtualAddressNotify() executed when gEfiEventVirtualAddressChangeGuid is installed.


MdeModulePkg/PiSmmCore/PiSmmIpl.c
ExecuteSmmCoreFromSmram()
Search all Firmware Volumes for a PE/COFF image with file type SMM_CORE.
Set local EntryPoint variable to image EntryPoint, then execute local EntryPoint to run image with file type SMM_CORE.
See **SmmMain**


MdeModulePkg/Core/PiSmmCore/PiSmmCore.c
SmmMain()
Get SMM Core Private context passed in from SMM IPL in ImageHandle.
See SmmEntryPoint
Set global pointers in SMM Core private structure to the SMST and the SmmEntryPoint..
See PI Volume 4 Chapter 3 "System Management System Table" for details.
Initialize SMM Memory Service.
Register all SW SMI Handlers required by the SMM Core, including the SMM Dispatcher, listed in mSmmCoreSmiHandlers table. These are invoked via the SMM Communication protocol.
The first entry is SmmDriverDispatchHandler, which has HandlerType of gEfiEventDxeDispatchGuid. DXE Dispatcher checks this HandlerType after each DXE driver is dispatched. This is the mechanism that allows SmmDriverDispatchHandler() to run.
See SmiHandlerRegister
See SmmDriverDispatchHandler
MdeModulePkg/Core/Dxe/Dispatcher/Dispatcher.c
CoreDispatcher()
An empty callback function event is created on DxeDispatchEvent variable for event group gEfiEventDxeDispatchGuid.
The do {} loop dispatches each DXE driver. If no error occurred, then CoreSignalEvent() is called and passed DxeDispatchEvent.


MdeModulePkg/Core/PiSmmCore/Smi.c
SmiHandlerRegister()
Allocate local SMI_HANDLER, set Signature to SMI_HANDLER_SIGNATURE, and set Handler to specified Handler.
If specified HandlerType (a GUID) is NULL,
Set local SMI_ENTRY to NULL and local List to head of mRootSmiHandlerList.
See PI Volume 4 Chapter 1.9 "Dispatching SMI Handlers" for details.
Else specified HandlerType (a GUID) != NULL,
Call SmmCoreFindSmiEntry() with HandlerType and Create=TRUE to determine if GUID is in mSmiEntryList.
See SmmCoreFindSmiEntry
If SmmCoreFindSmiEntry() returns NULL, then return error.
Else set local SMI_ENTRY to one returned by SmmCoreFindSmiEntry(), and set local List to the head of the SMI_HANDLERS of the local SMI_ENTRY.
Set local SMI_HANDLER SMI_ENTRY item to local SMI_ENTRY, add local SMI_HANDLER to tail of local List, and set specified DispatchHandle to local SMI_HANDLER.
See SMI Handlers

mRootSmiHandlerList

- Double-linked list of SMI_HANDLER items.
- Each SMI_HANDLER item has an SMI_ENTRY pointer that is NULL.
- Registration is done by SmmMain() calls and platform code calls to SmiRegisterHandler() with HandlerType = NULL.

SMI_HANDLER item
Signature = **SMI_HANDLER_SIGNATURE**
SMI_ENTRY pointer = **NULL**
Handler = code entry point

SMI_HANDLER item
Signature = **SMI_HANDLER_SIGNATURE**
SMI_ENTRY pointer = **NULL**
Handler = code entry point

SMI_HANDLER item
Signature = **SMI_HANDLER_SIGNATURE**
SMI_ENTRY pointer = **NULL**
Handler = code entry point

MdeModulePkg/Core/PiSmmCore/Smi.c
mRootSmiHandlerList and **mSmiEntryList**

Lined arrows represent double-linked list Forward and Back pointers.

SMI_HANDLER item
Signature = **SMI_HANDLER_SIGNATURE**
SMI_ENTRY pointer = **NULL**
Handler = code entry point

SMI_HANDLER item
Signature = **SMI_HANDLER_SIGNATURE**
SMI_ENTRY pointer = **NULL**
Handler = code entry point

mSmiEntryList

- Double-linked list of SMI_ENTRY items.
- Each SMI_ENTRY item points to a double-linked list of SMI_HANDLER items.
- Each SMI_HANDLER item points to the SMI_ENTRY item that it is registered with.
- Registration is done by SmmMain() calls and platform code calls to SmiRegisterHandler() with HandlerType != NULL.

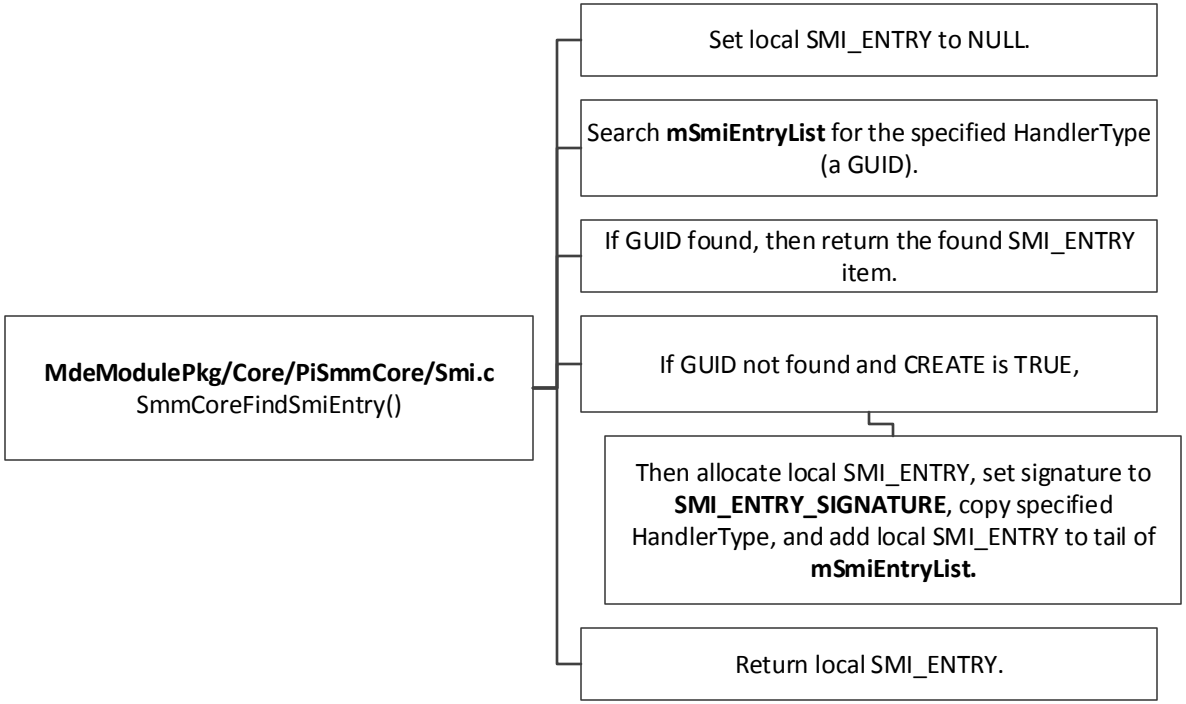
SMI_ENTRY item
Signature = **SMI_ENTRY_SIGNATURE**
SMI_HANDLER pointer != **NULL**
HandlerType = a GUID

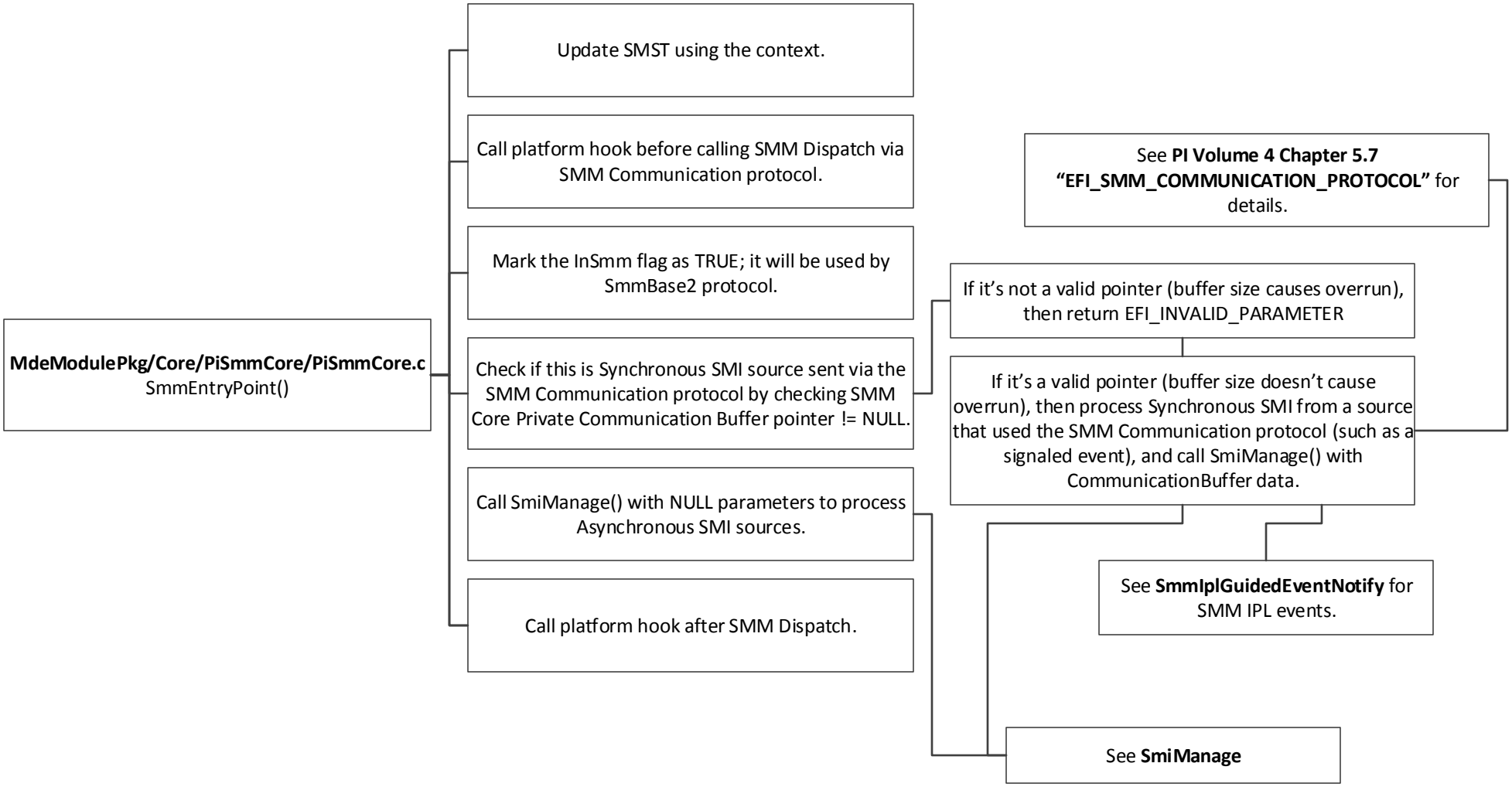
SMI_ENTRY item
Signature = **SMI_ENTRY_SIGNATURE**
SMI_HANDLER pointer != **NULL**
HandlerType = a GUID

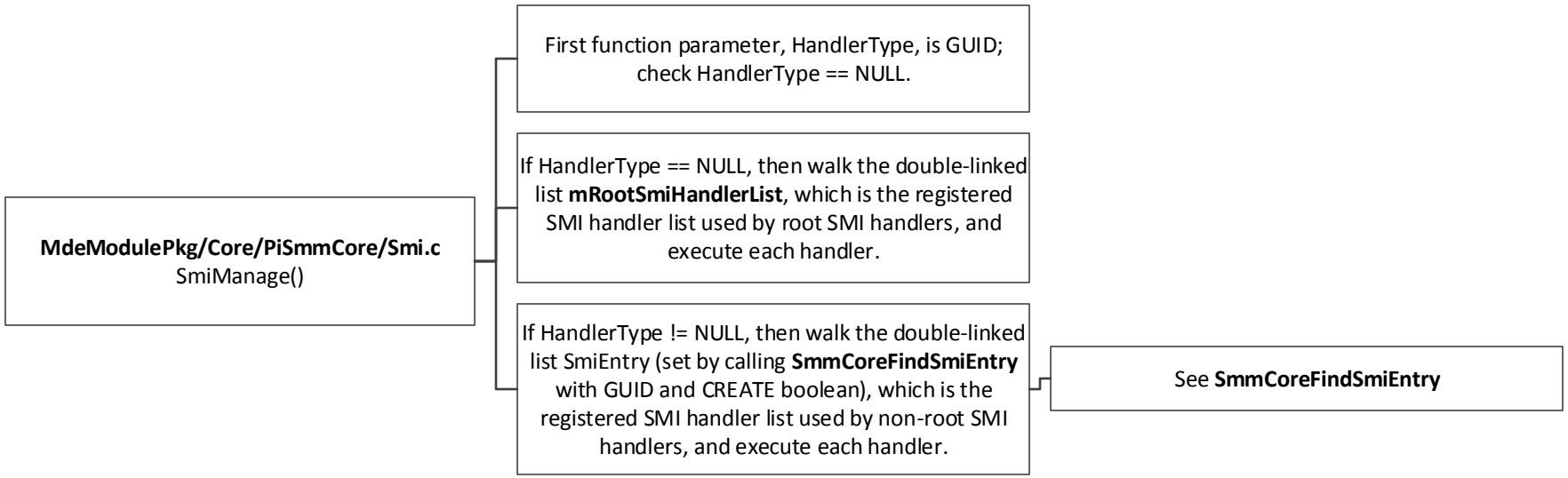
SMI_ENTRY item
Signature = **SMI_ENTRY_SIGNATURE**
SMI_HANDLER pointer != **NULL**
HandlerType = a GUID

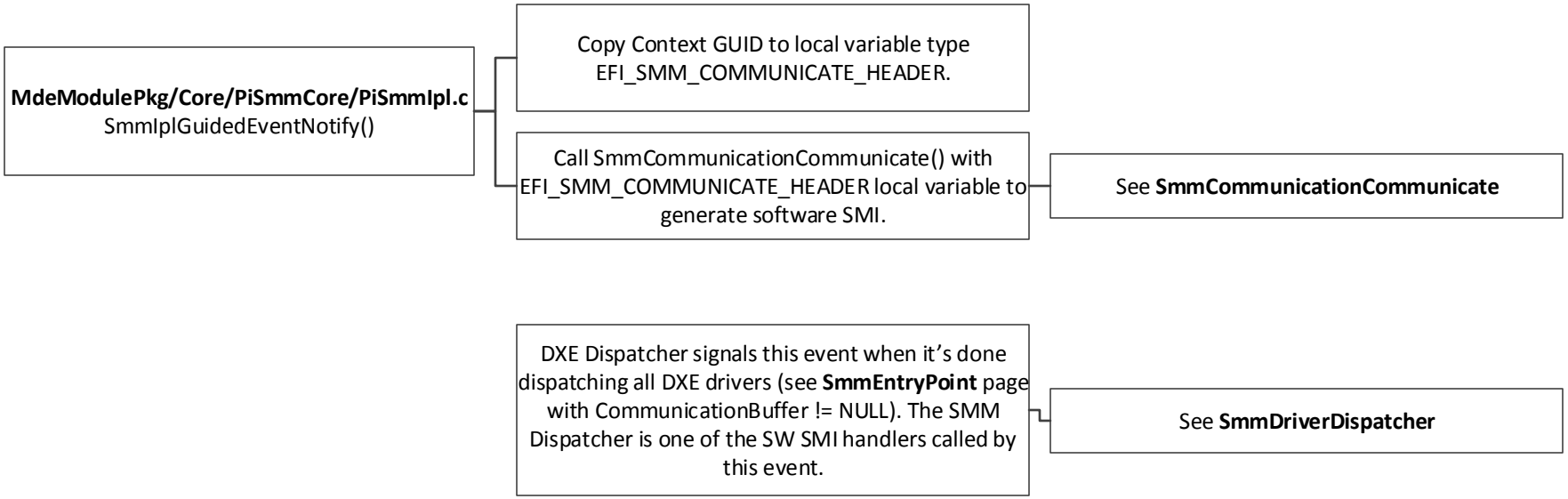
SMI_HANDLER item
Signature = **SMI_HANDLER_SIGNATURE**
SMI_ENTRY pointer = this SMI_ENTRY
Handler = code entry point

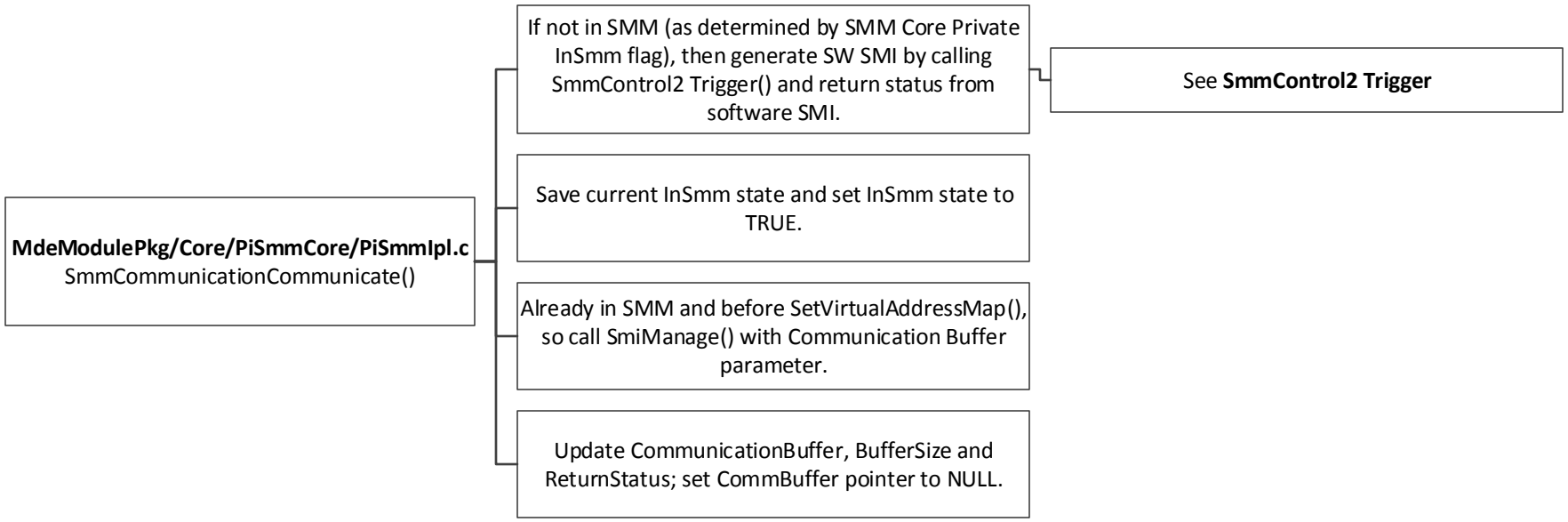
SMI_HANDLER item
Signature = **SMI_HANDLER_SIGNATURE**
SMI_ENTRY pointer = this SMI_ENTRY
Handler = code entry point

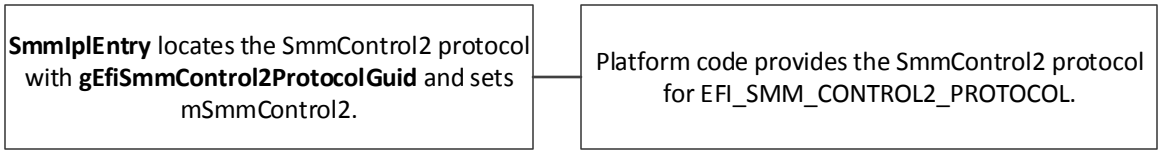

MdeModulePkg/Core/PiSmmCore/Smi.c
SmmCoreFindSmiEntry()
Set local SMI_ENTRY to NULL.
Search mSmiEntryList for the specified HandlerType (a GUID).
If GUID found, then return the found SMI_ENTRY item.
If GUID not found and CREATE is TRUE,
Then allocate local SMI_ENTRY, set signature to SMI_ENTRY_SIGNATURE, copy specified HandlerType, and add local SMI_ENTRY to tail of mSmiEntryList.
Return local SMI_ENTRY.

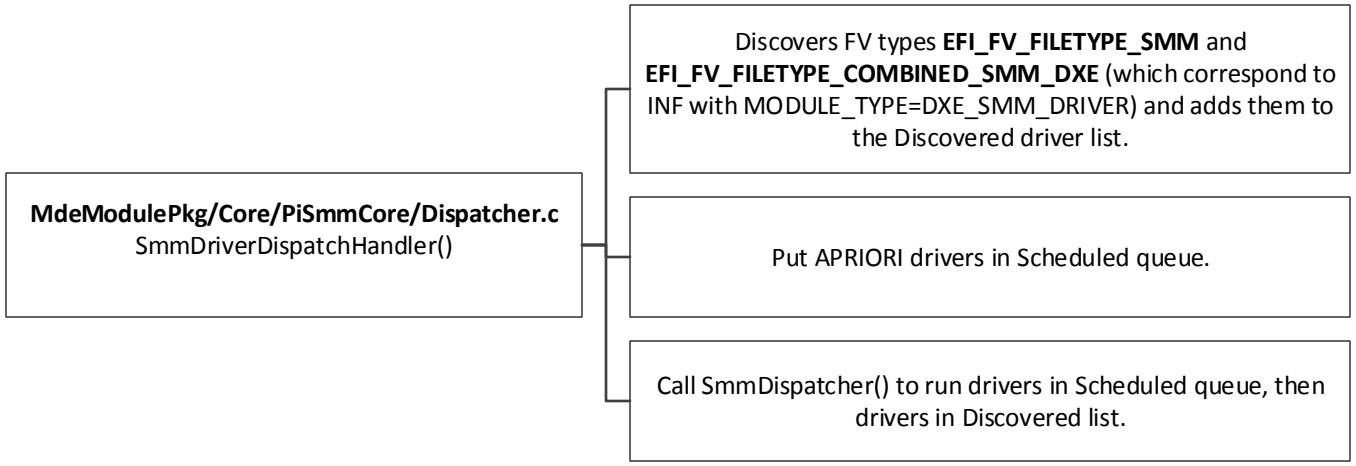

MdeModulePkg/Core/PiSmmCore/PiSmmCore.c
SmmEntryPoint()
Update SMST using the context.
Call platform hook before calling SMM Dispatch via SMM Communication protocol.
Mark the InSmm flag as TRUE; it will be used by SmmBase2 protocol.
Check if this is Synchronous SMI source sent via the SMM Communication protocol by checking SMM Core Private Communication Buffer pointer != NULL.
Call SmiManage() with NULL parameters to process Asynchronous SMI sources.
Call platform hook after SMM Dispatch.
See **PI Volume 4 Chapter 5.7 "EFI_SMM_COMMUNICATION_PROTOCOL"** for details.
If it's not a valid pointer (buffer size causes overrun), then return EFI_INVALID_PARAMETER
If it's a valid pointer (buffer size doesn't cause overrun), then process Synchronous SMI from a source that used the SMM Communication protocol (such as a signaled event), and call SmiManage() with CommunicationBuffer data.
See **SmmIplGuidedEventNotify** for SMM IPL events.
See **SmiManage**


MdeModulePkg/Core/PiSmmCore/Smi.c
SmiManage()
First function parameter, HandlerType, is GUID; check HandlerType == NULL.
If HandlerType == NULL, then walk the double-linked list mRootSmiHandlerList, which is the registered SMI handler list used by root SMI handlers, and execute each handler.
If HandlerType != NULL, then walk the double-linked list SmiEntry (set by calling SmmCoreFindSmiEntry with GUID and CREATE boolean), which is the registered SMI handler list used by non-root SMI handlers, and execute each handler.
See SmmCoreFindSmiEntry


MdeModulePkg/Core/PiSmmCore/PiSmmIpl.c
SmmIplGuidedEventNotify()
Copy Context GUID to local variable type EFI_SMM_COMMUNICATE_HEADER.
Call SmmCommunicationCommunicate() with EFI_SMM_COMMUNICATE_HEADER local variable to generate software SMI.
See **SmmCommunicationCommunicate**
DXE Dispatcher signals this event when it's done dispatching all DXE drivers (see **SmmEntryPoint** page with CommunicationBuffer != NULL). The SMM Dispatcher is one of the SW SMI handlers called by this event.
See **SmmDriverDispatcher**


MdeModulePkg/Core/PiSmmCore/PiSmmIpl.c
SmmCommunicationCommunicate()
If not in SMM (as determined by SMM Core Private InSmm flag), then generate SW SMI by calling SmmControl2 Trigger() and return status from software SMI.
See SmmControl2 Trigger
Save current InSmm state and set InSmm state to TRUE.
Already in SMM and before SetVirtualAddressMap(), so call SmiManage() with Communication Buffer parameter.
Update CommunicationBuffer, BufferSize and ReturnStatus; set CommBuffer pointer to NULL.


SmmIplEntry locates the SmmControl2 protocol with **gEfiSmmControl2ProtocolGuid** and sets mSmmControl2.
Platform code provides the SmmControl2 protocol for EFI_SMM_CONTROL2_PROTOCOL.


MdeModulePkg/Core/PiSmmCore/Dispatcher.c
SmmDriverDispatchHandler()
Discovers FV types EFI_FV_FILETYPE_SMM and EFI_FV_FILETYPE_COMBINED_SMM_DXE (which correspond to INF with MODULE_TYPE=DXE_SMM_DRIVER) and adds them to the Discovered driver list.
Put APRIORI drivers in Scheduled queue.
Call SmmDispatcher() to run drivers in Scheduled queue, then drivers in Discovered list.