

November 15-18, 2010



Santa Clara Marriott
Santa Clara, CA

Platform Management Components Intercommunications (PMCI): An Overview

Hemal V. Shah

Associate Technical Director, Broadcom Corporation

Platform Management Sub-committee Chair, PMCI WG Co-chair,
DMTF

Tom Slaight

Principal Engineer, Intel Corporation
PMCI WG Co-Chair, DMTF

Disclaimer

- The information in this presentation represents a snapshot of work in progress within the DMTF.
- This information is subject to change. The Standard Specifications remain the normative reference for all information.
- For additional information, see the Distributed Management Task Force (DMTF) Web site.

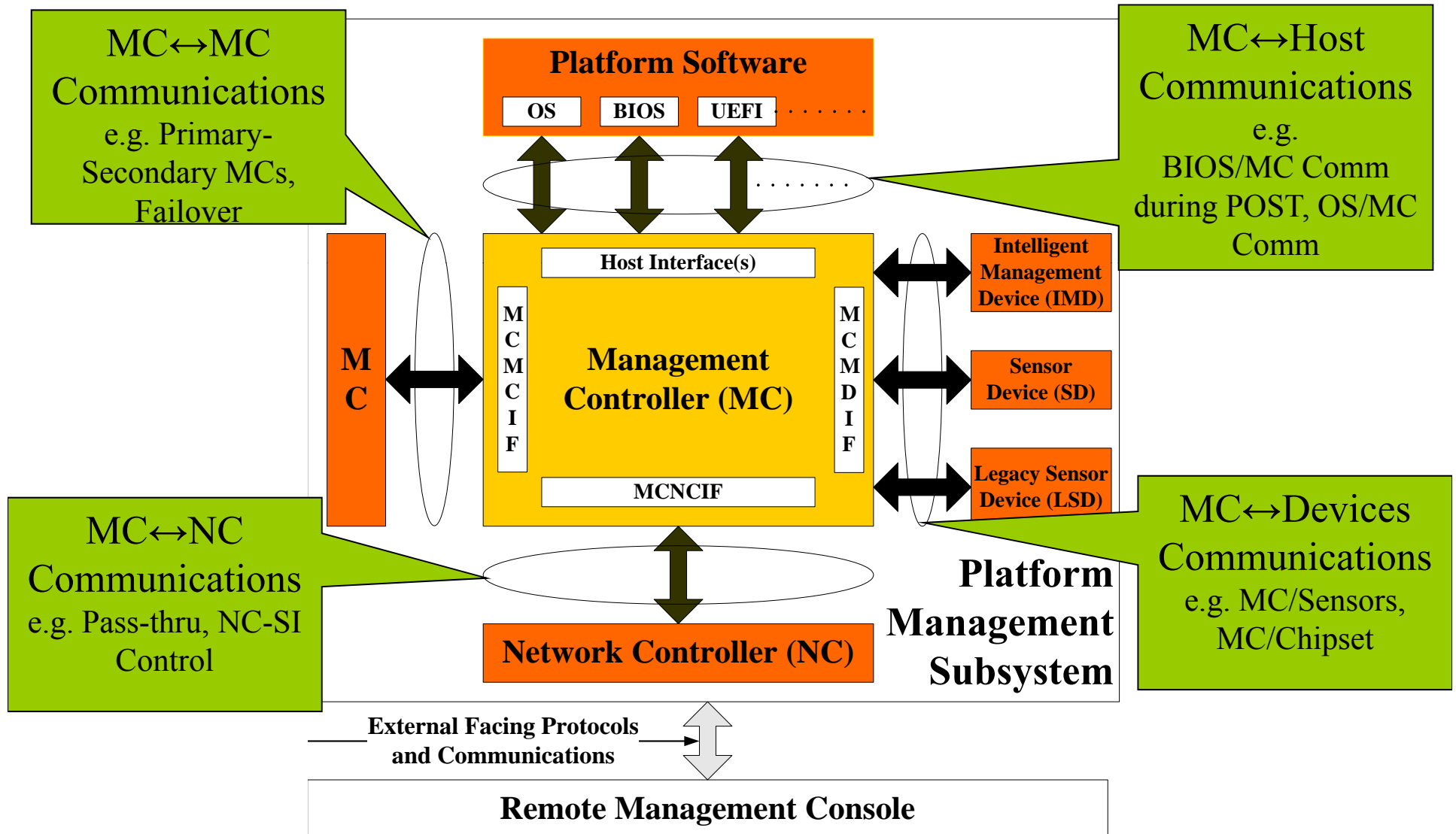




Agenda

- Platform Management Subsystem
- PMCI Overview
- PMCI Protocol Stack
- NC-SI Overview
- MCTP Overview
- PLDM Overview

Platform Management Subsystem



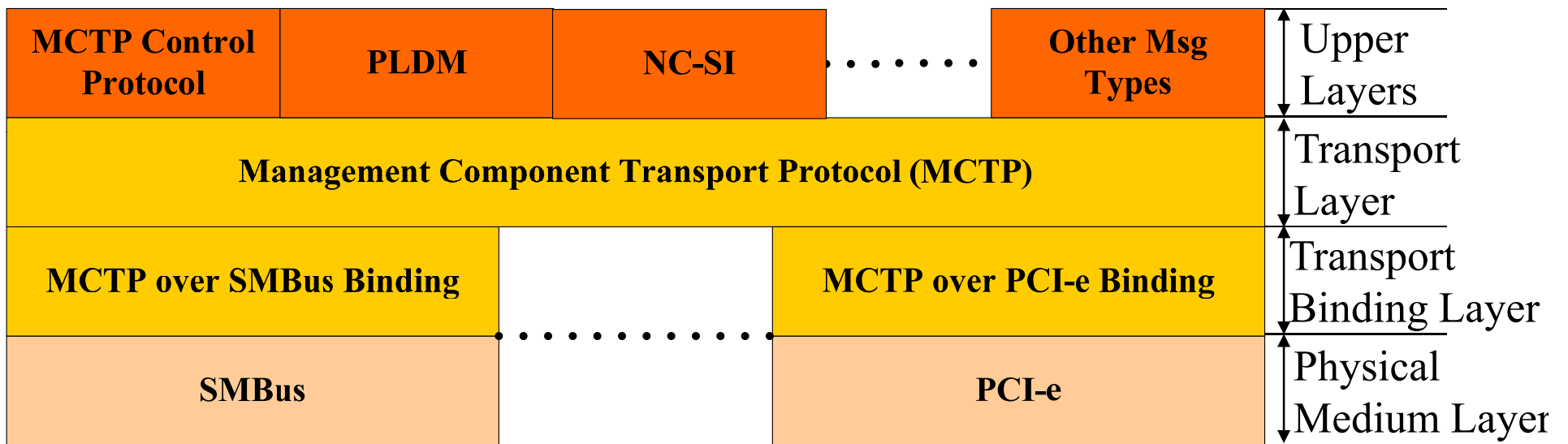


PMCI Working Group

- **Platform Management Component Intercommunications**
 - DMTF Pre-OS WG sub-team formed August 2006
 - 18 Member Companies in workgroup (as of 9/24/2010)
 - Active participants include AMD, Broadcom, IBM, Intel & SMSC
 - Co-chairs: Hemal Shah, Broadcom; Tom Slaight, Intel
- Scope: “Inside the box” communication and functional interfaces between components within the platform management subsystem
 - Mgmt Controller (MC) to Mgmt Controller
 - Mgmt Controller to Intelligent Management Device
 - Mgmt Controller to Network Controller
 - FW / SW to Mgmt Controller
- Builds on and captures learnings from SMBIOS, ASF, & NC-SI
 - Plus leverages SMBus, IPMI, PCIe, and other industry technologies

PMCI technologies and interfaces are complementary to DMTF CIM Profiles and remote access protocols

PMCI Protocol Stack

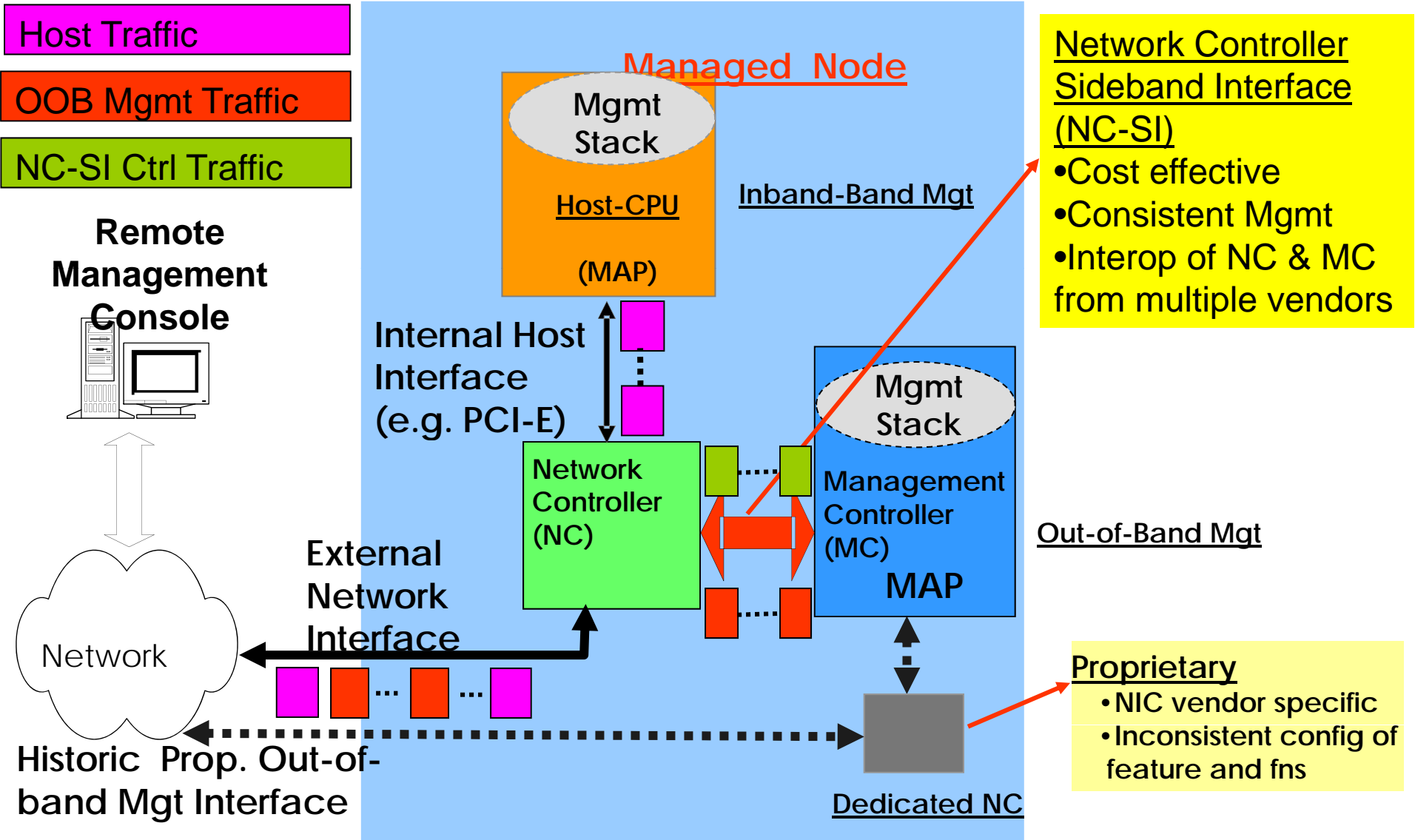




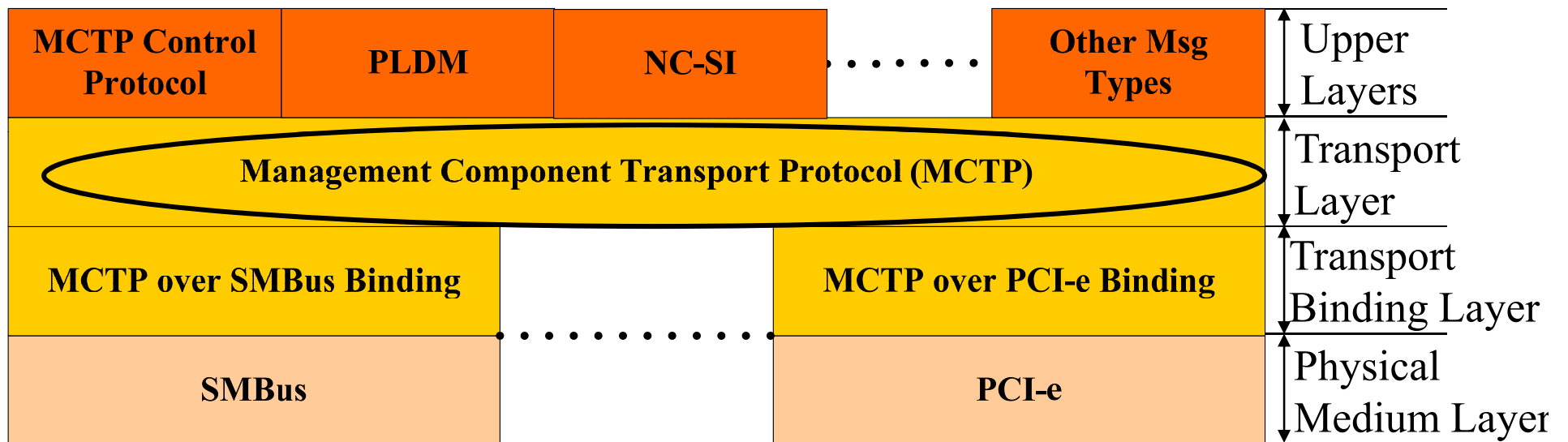
Network Controller-Sideband Interface (NC-SI)

- A common interoperable sideband interface and protocol
 - To transfer management traffic between a management controller (MC) & network controller (NC)
- NC-SI Communications
 - Pass-Thru Management Traffic
 - NC-SI Command/Response Packets
 - Command (Response) sent by MC (NC) to NC (MC)
 - Request/Response Semantics
 - Functions: Control, Configuration, Status, Statistics,...
 - NC-SI Notification Packets
 - Generated and sent by NC to MC
 - Functions: OS/Link Status Change; NC Soft Reset

Out-of-band Management and NC-SI



PMCI Protocol Stack

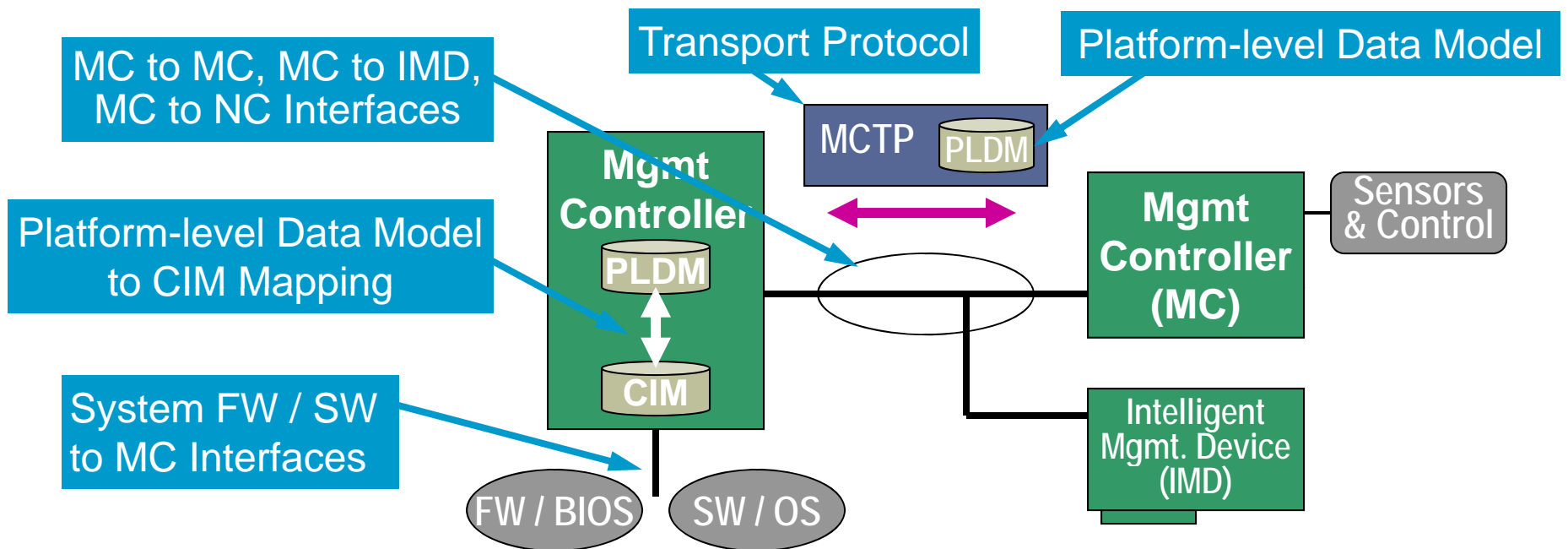




Management Component Transport Protocol (MCTP) Overview

- Base transport for ‘inside the box’ communication
- Carries multiple message types
 - MCTP Control, Platform Level Data Model...
- Suitable for use with multiple media
 - SMBus, PCI-e...
- Suitable to all computer platform types
- Designed to enable low-cost micro-controllers
- Supports logical addressing based on Endpoint IDs
- Provides simple message fragmentation/reassembly
- Built-in Capability discovery
- Supports path transmission unit discovery

MCTP - where it fits



- **Interfaces:** types of interconnects defined for platform management
- **Transport Protocol:** how mgmt data is moved across the interfaces
- **Platform-level Data Model:** how low level platform management hardware inventory, monitoring, and control functions are abstracted and accessed
- **PLDM to CIM Mapping:** how the low-level data model is used under the CIM Profiles

Logical Addressing

MCTP uses Logical Addressing between communication *Endpoints*

- Called the “Endpoint ID” (EID)
 - Similar to an IP Address
- Provides Media-independent addressing
 - Enables devices on different media to intercommunicate without having to deal with physical address format conversions

Bus Owners and Bridges

An MCTP Bus Owner

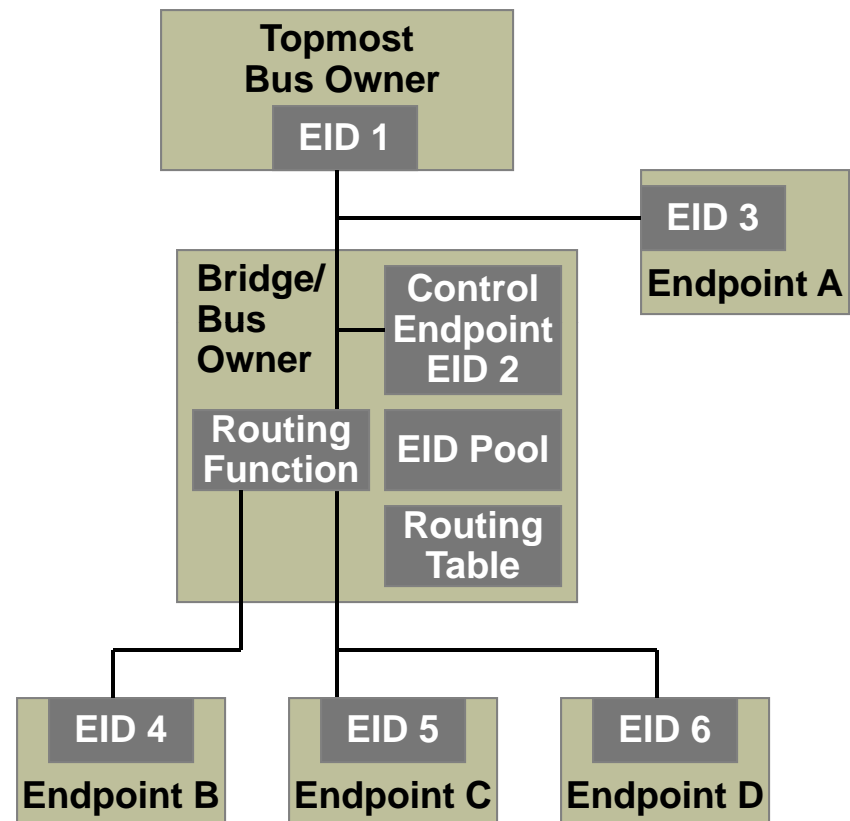
- Assigns EIDs to devices on busses it directly 'owns'
- Allocates EID pools to MCTP Bridges

The "Topmost Bus Owner"

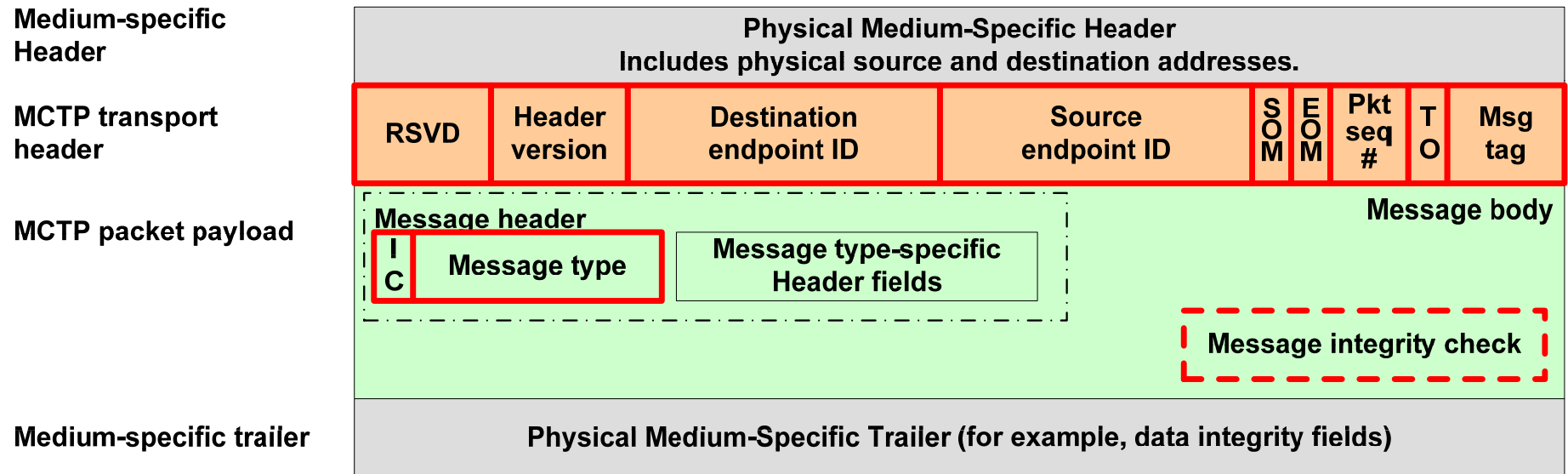
- Is a bus owner that is the source of all EIDs used in an MCTP bus hierarchy

An MCTP Bridge

- Is responsible for routing MCTP packets between two or more busses
- An MCTP Bridge is the bus owner of at least one bus
- Bridges are responsible for assigning EIDs and allocating EID pools to any devices on busses that the Bridge owns.



MCTP Packet Fields





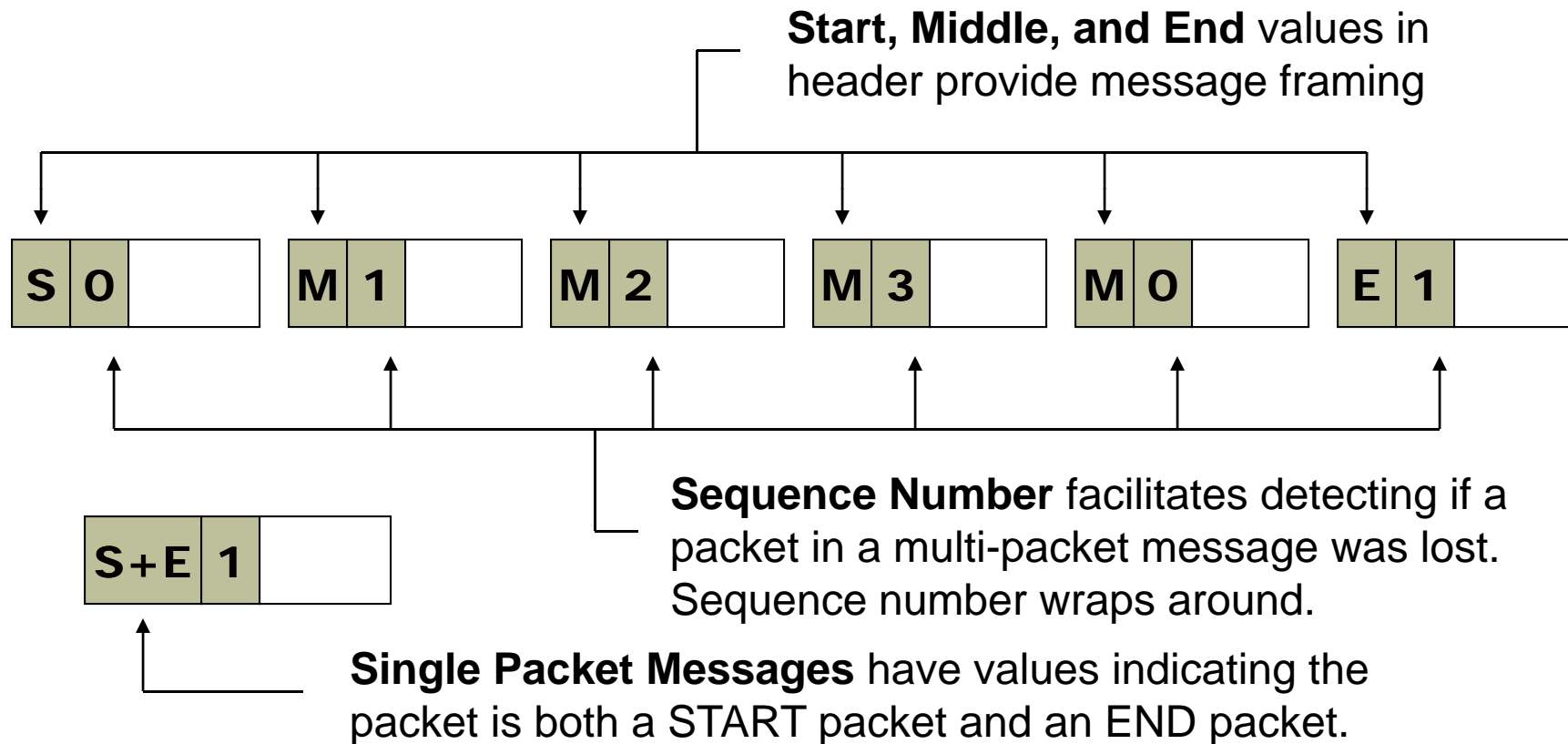
MCTP Common Fields

- **Source Endpoint ID** - Logical Address of the message originator
- **Destination Endpoint ID** - Logical Address of the message target
- **SOM flag** - Identifies first packet in message
- **EOM flag** - Identifies last packet in message
(SOM & EOM = 0 indicates a 'middle' packet)
- **Seq #** - Sequence number. (Used to detect if there are missing packets in a message)
- **Hdr Version** - Identifies MCTP transport packet format and fields for a given medium
- **Msg Tag** - Identifies packets belonging to a particular message
- **TO bit** - "Tag Owner bit" - Identifies whether an endpoint is the originator of the Msg Tag or is using a Msg Tag given to it by another endpoint.
- **Message Type** - Identifies the payload format used for higher level protocols. (only required in the header for the first packet)
- **IC bit** - Integrity Check bit. Indicates whether an optional overall message integrity check field is present.

Message Assembly

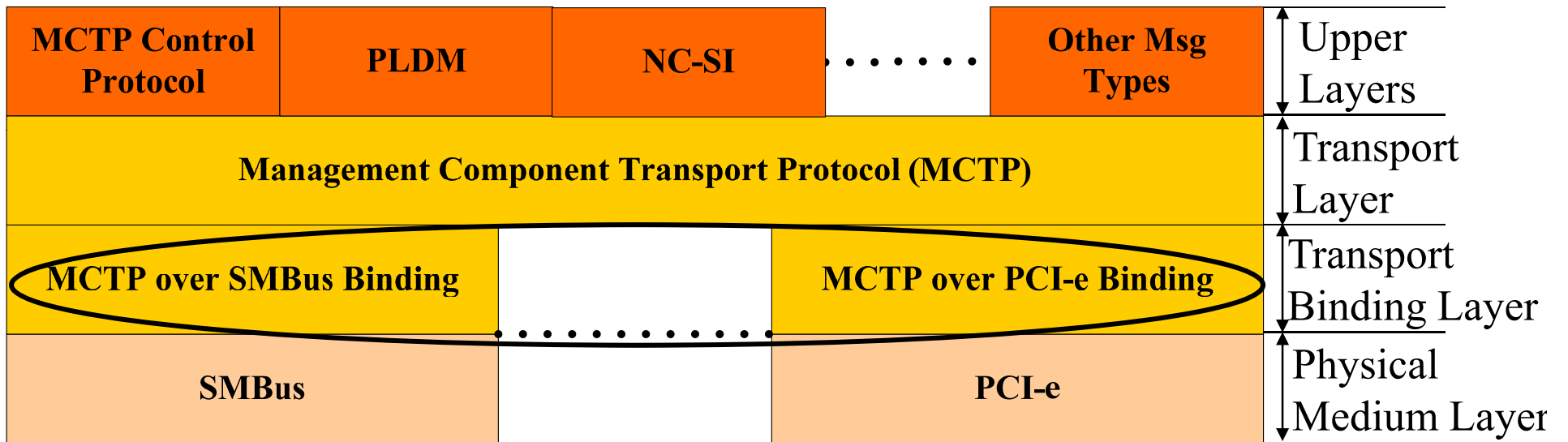
An MCTP Message can be formed of multiple MCTP packets.

- Bits in packet identify start, middle, and end packets for a given message.



The Source Endpoint ID, Msg Tag, and TO bit fields uniquely identify the packets for a given message

PMCI Protocol Stack

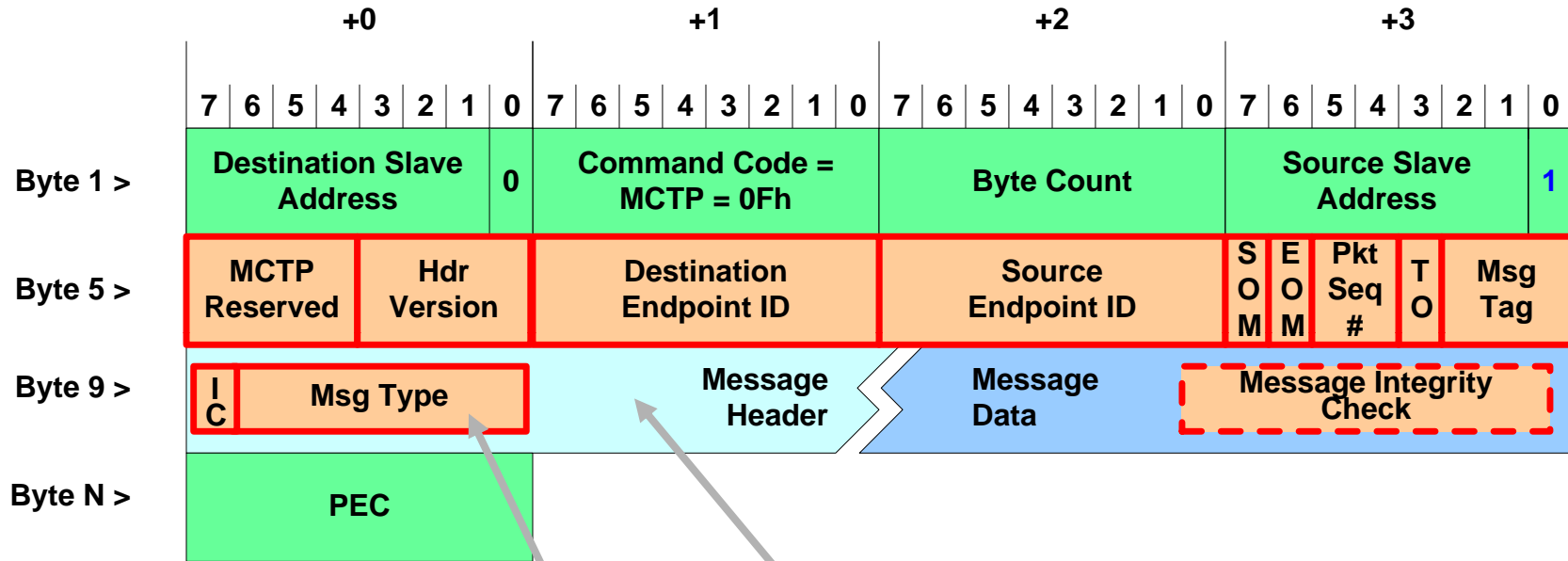




MCTP Transport Bindings

- Bindings for four physical mediums defined
 - SMBus
 - PCIe VDM
 - KCS
 - Serial

MCTP over SMBus Packet Format



MCTP Message Header
(Varies based on Message Type)

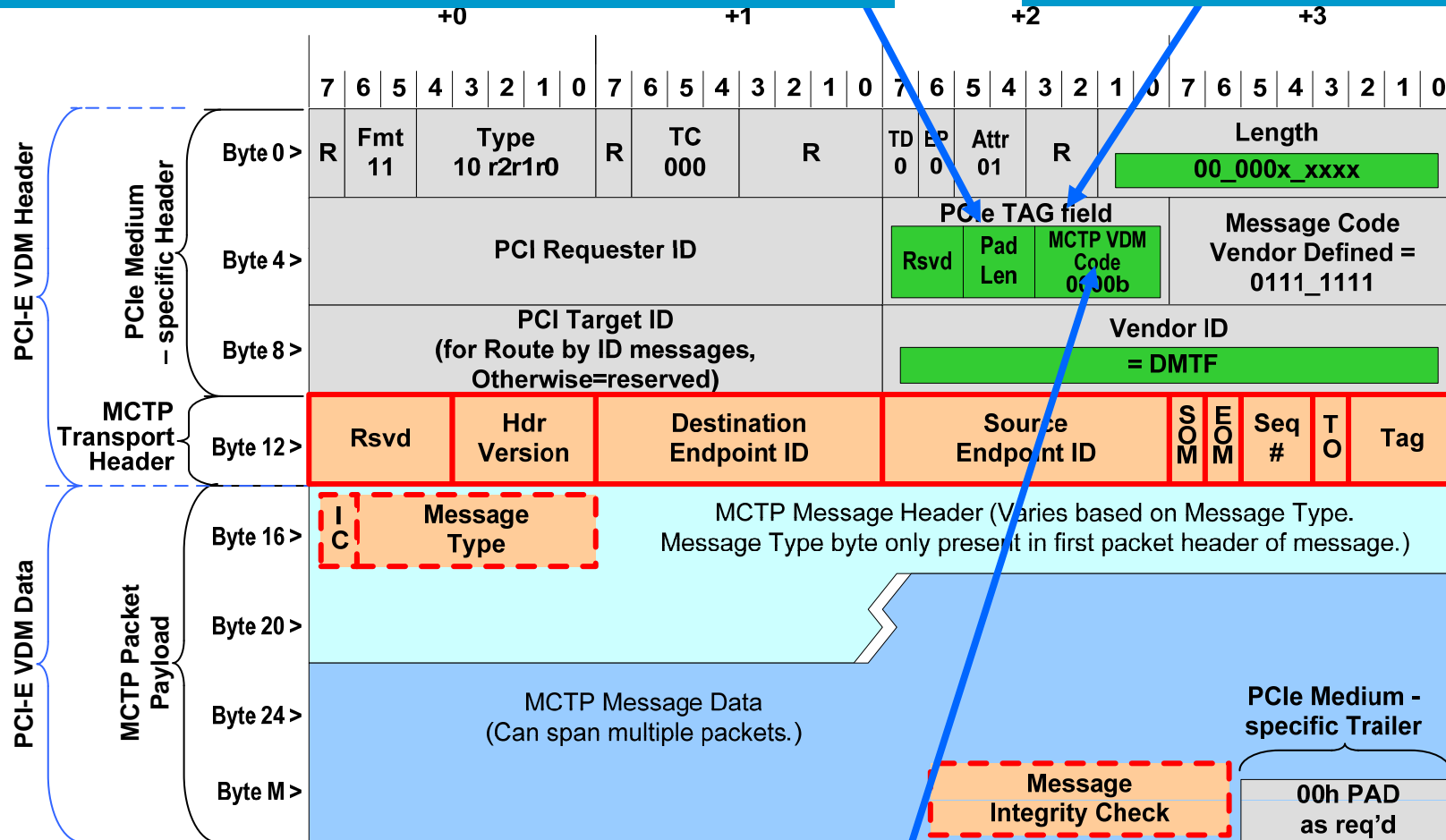
Message Type byte
(only required in first packet header of message.)

= common fields for all MCTP messages

MCTP PCIe VDM Packet Format

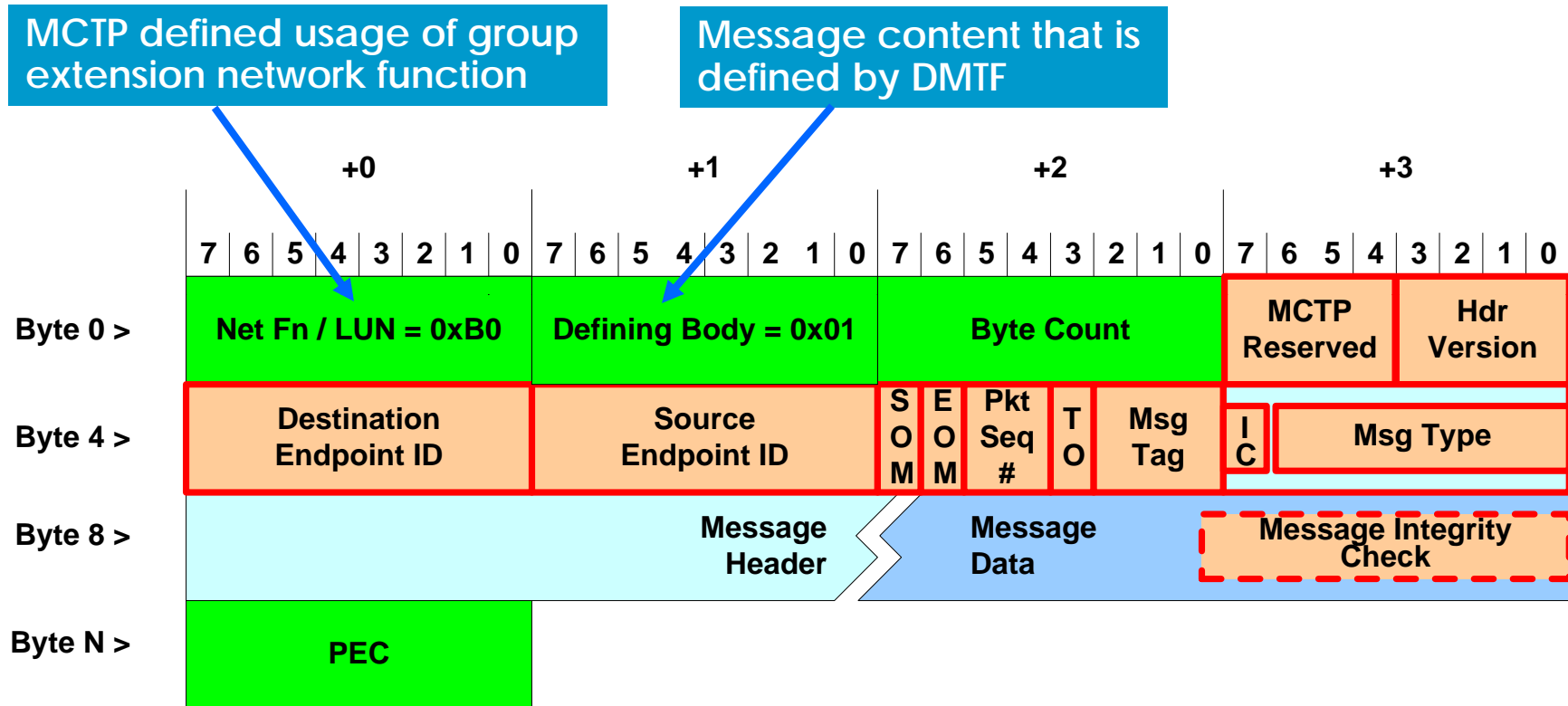
Pad Length - (2-bits) indicates # of pad bytes to get PCIe message DWORD aligned.

MCTP defined usage of PCIe TAG field

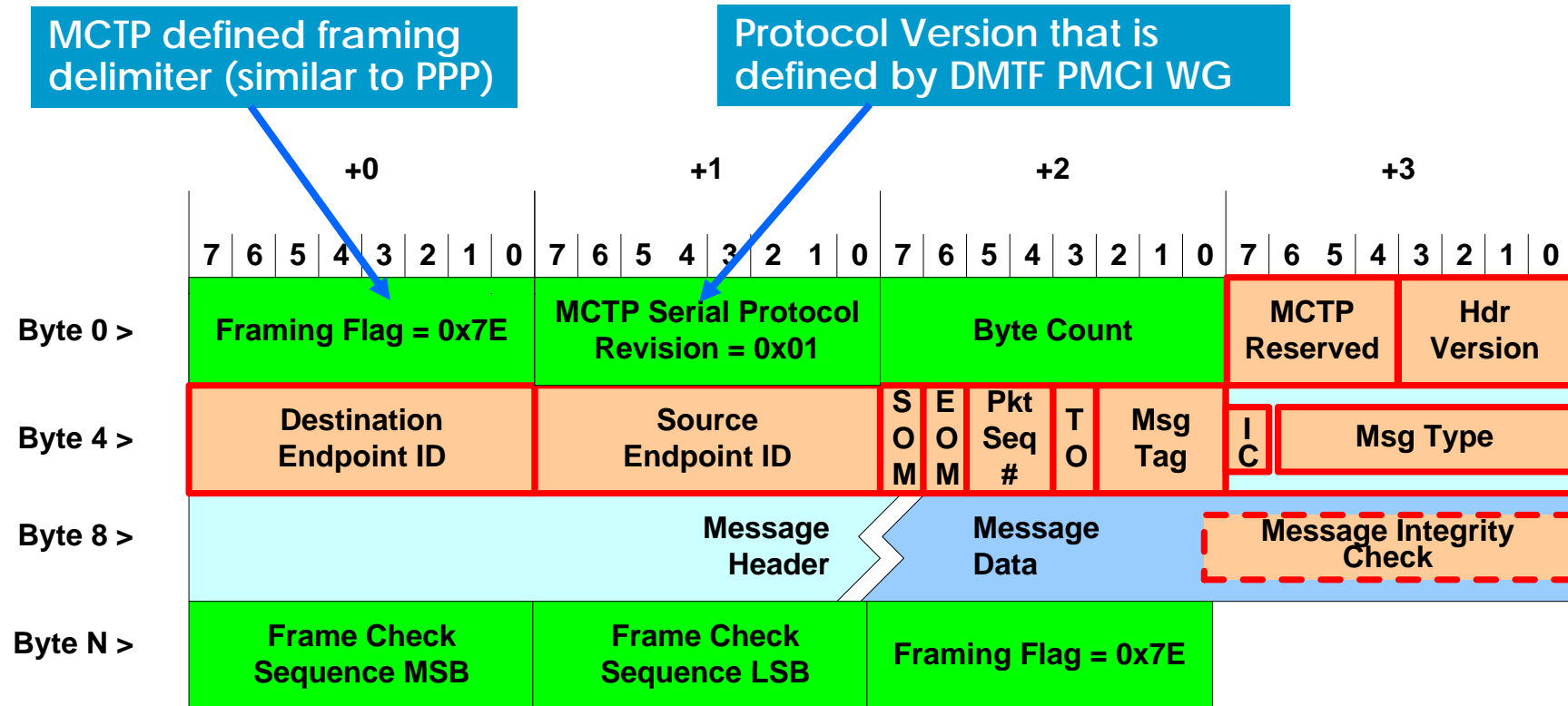


MCTP VDM Code uniquely identifies MCTP VDMs from other possible VDMs that may be defined under the DMTF Vendor ID

MCTP over KCS Packet Format

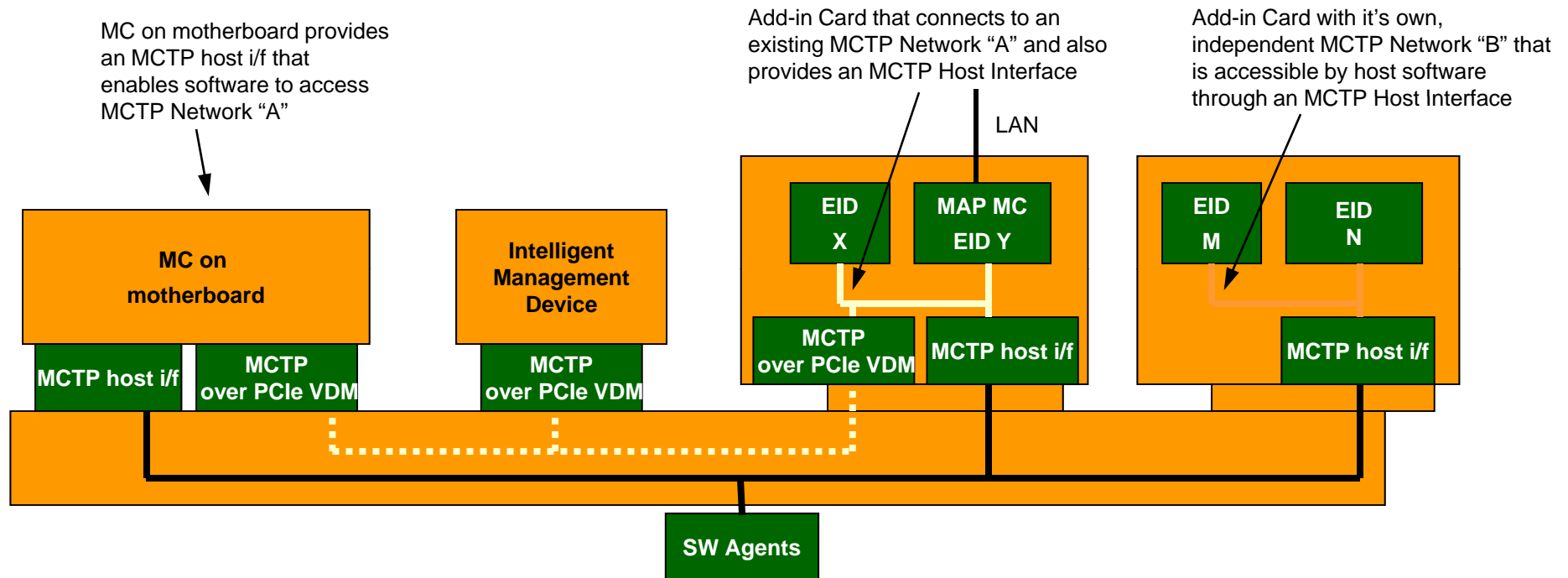


MCTP over Serial Packet Format



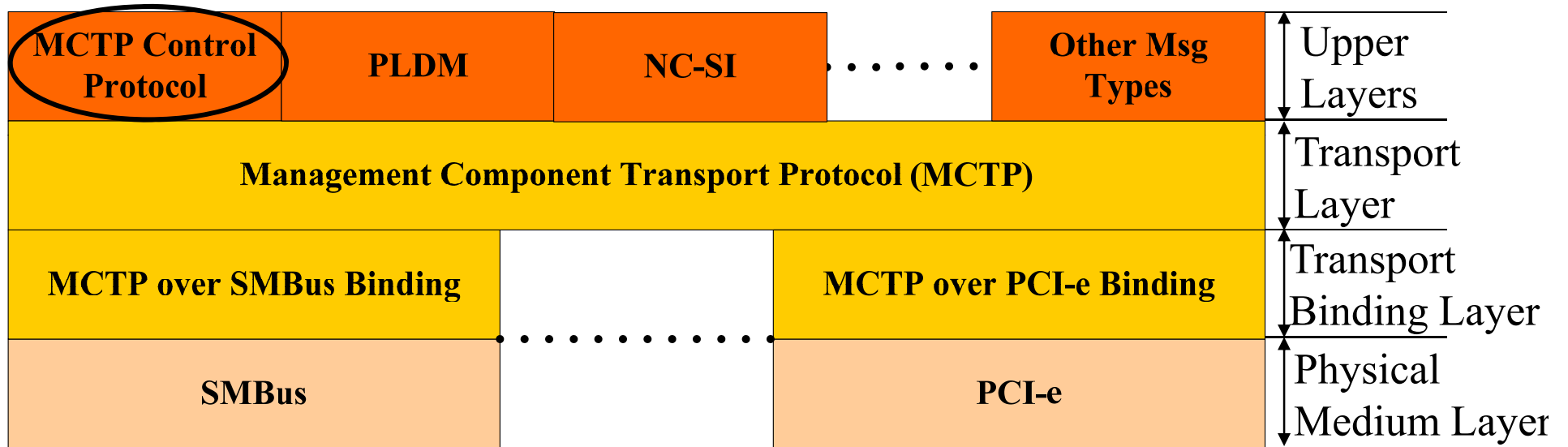
Packet format and bytes shown are prior to escape sequence

MCTP Host Interface



- Used by host software for MCTP communications
- Supports multiple host interfaces
- Supports multiple MCTP networks
- Several mechanisms for host interface discovery
 - SMBIOS, PCI/PCIe, and ACPI

PMCI Protocol Stack





MCTP Control Message Protocol

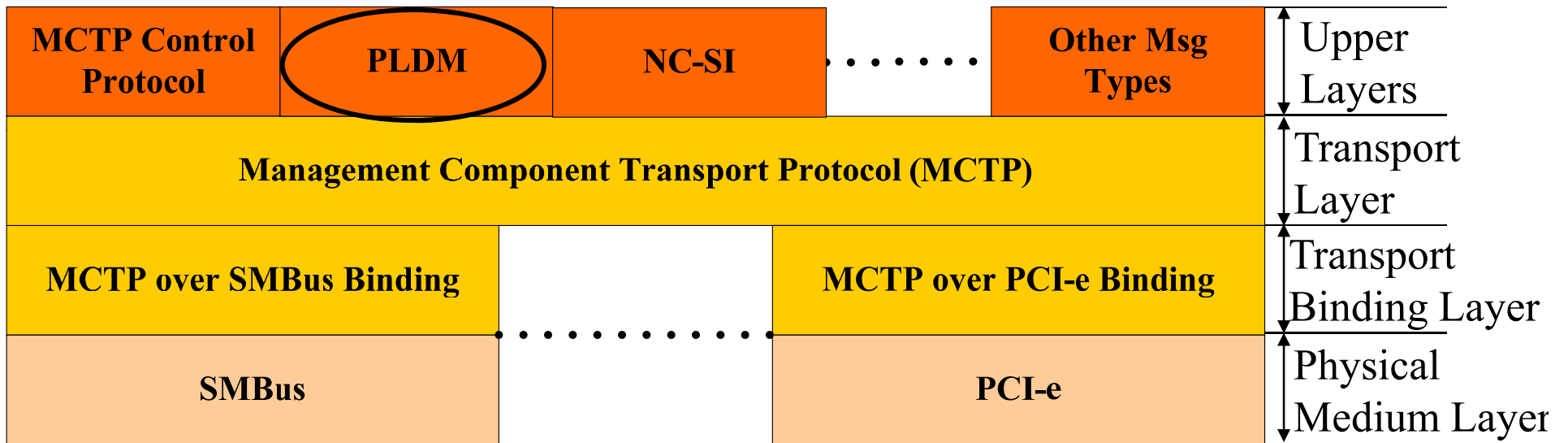
- Used for transfer of MCTP Control Messages
- Built on top of MCTP packet transport specs
- Request / Response protocol with message level retry support
 - Instance ID field supports message retries
- Datagram supported for unacknowledged 'alert' messages
- Control Messages fit in single Baseline MTU sized MCTP packets
 - No need for assembly / disassembly

MCTP Control Messages

MCTP Control Messages are for the initialization and support of MCTP communications, including:

- Endpoint Discovery
Prepare for Endpoint Discovery, Endpoint Discovery, Discovery Notify, Get Endpoint UUID
- EID assignment and EID Pool allocation
Set Endpoint ID, Get Endpoint ID, Allocate Endpoint IDs
- EID to physical address resolution
Resolve Endpoint ID
- Bridge Routing Table initialization and updates
Routing Information Update
- MCTP Network Topology and Path Transmission Unit discovery
Get Routing Table Entries, Query Hop, Get MCTP Network ID
- MCTP Message Type and Vendor Defined Message Support
Get MCTP Version Support, Get Message Type Support, Get Vendor Defined Message Support
- Transport Specific Commands

PMCI Protocol Stack



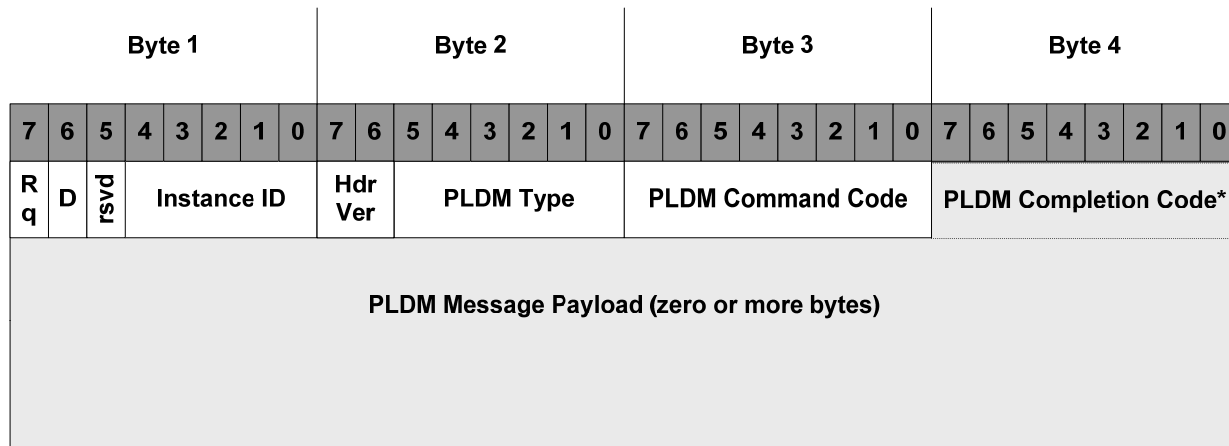


Platform Level Data Model (PLDM) – What is it?

- An effective interface and data model that provides efficient access to
 - Low-level platform inventory, BIOS control and configuration data
 - Platform monitoring and control functions, alerting and event log data...
- Defines data representations and commands that abstract platform mgmt subsystem components
- Provides transport independent Request/Response Style Messaging Model
- Supports a sub-type to distinguish various types of PLDM messages
 - Allow messages to be grouped based on the functions
 - Allows the discovery of the functionality supported
 - Examples:
 - PLDM for SMBIOS data transfer
 - PLDM for BIOS control and configuration
 - PLDM for Platform Monitoring and Control
 - PLDM for FRU
- PLDM over MCTP Binding
 - All the PLDM messages transferable using baseline transmission unit
 - Larger message sizes may be negotiated for specific message types

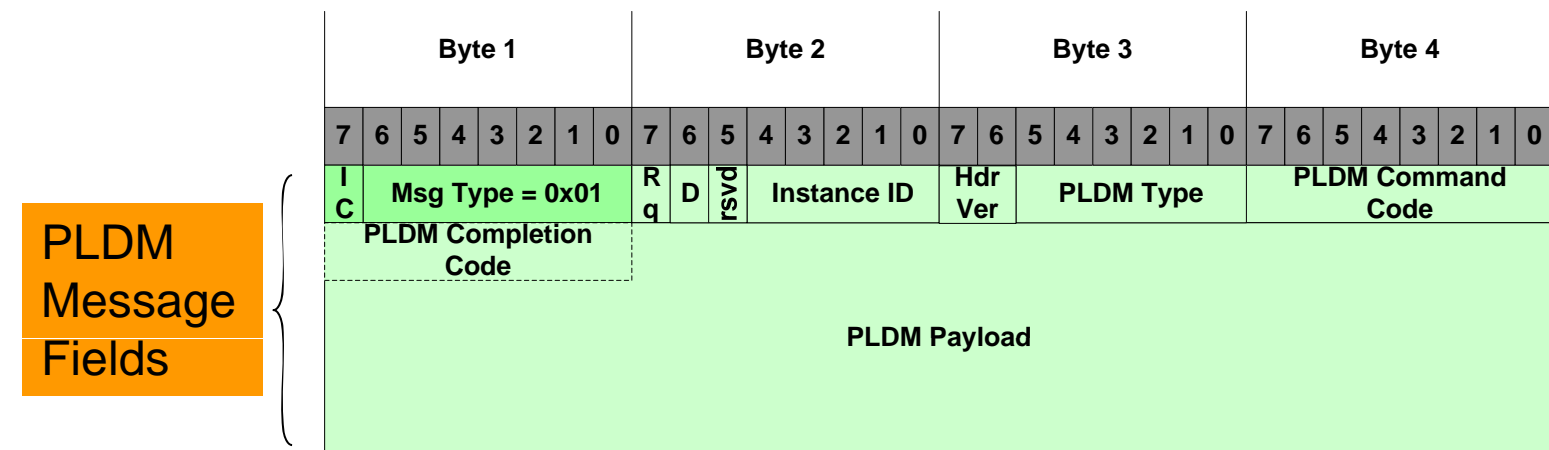
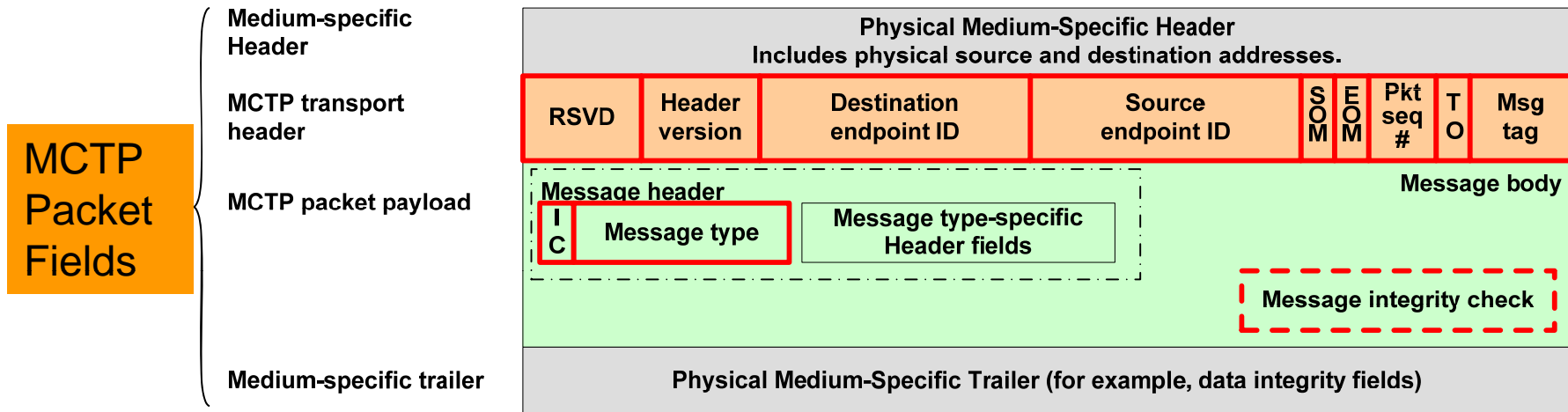


PLDM Base Message Fields



Field Name	Field Size	Description
Rq	1 bit	Request bit.
D	1 bit	Datagram bit.
rsvd	1 bit	Reserved
Instance ID	5 bits	Used to identify the instances of a PLDM request.
Hdr Ver	2 bits	Identifies the header format of PLDM messages.
PLDM Type	6 bits	The PLDM Type field identifies the type of PLDM that is being used.
PLDM Command Code	8 bits	Identifies the type of operation the message (per PLDM type) is requesting.
PLDM Completion Code	8 bits	The PLDM Completion Code field provides the status of the operation
PLDM Message Payload	Variable	Zero or more bytes of PLDM message payload that is specific to a particular payload type, PLDM type, command code, and/or completion code.

PLDM over MCTP Binding – How does PLDM map over MCTP?





Basic PLDM Terminology

- **PLDM command or PLDM Request**
 - A command defined under the PLDM Type that is used for PLDM communications
- **PLDM message**
 - A unit of communication based on the PLDM Type used for PLDM communications
- **PLDM request message**
 - A message that is sent to a PLDM terminus to request a specific PLDM operation
 - A PLDM request message is acked with a corresponding response message
- **PLDM response message**
 - A message that is sent in response to a specific PLDM request message
 - Includes "Completion Code" that indicates whether the response completed normally
- **PLDM Terminus**
 - Identifies a set of resources within the recipient endpoint that is handling a particular PLDM message
 - Terminus ID (TID) is used to identify a PLDM Terminus
 - TID is transport independent



PLDM Types

PLDM Type	PLDM Type Code	Description
PLDM Messaging Control and Discovery	000000b	PLDM Messages used to support communication control and discovery operations for PLDM.
PLDM for SMBIOS	000001b	PLDM Messages used to support SMBIOS data Transfer.
PLDM for Platform Monitoring and Control	000010b	PLDM Messages used to support platform monitoring and control.
PLDM for BIOS Control and Configuration	000011b	PLDM Messages used to support BIOS control and configuration data transfer between the BIOS and the MC.
PLDM for FRU Data	000100b	PLDM Messages used to support FRU data transfer
Reserved	000101b-111110b	
OEM Specific	111111b	Reserved for OEM-specific PLDM Commands



Messaging Control and Discovery Commands

- **SetTID**
 - Used to set the Terminus ID (TID) for a PLDM Terminus
- **GetTID**
 - Used to retrieve the present Terminus ID (TID) setting
- **GetPLDMVersion**
 - Used to retrieve the PLDM base specification versions as well as the PLDM Type specific versions supported for each PLDM Type
- **GetPLDMTypes**
 - Used to discover the PLDM type capabilities supported by the PLDM terminus and get a list of the PLDM types that are supported
- **GetPLDMCommands**
 - Used to discover the PLDM command capabilities supported by the PLDM terminus for a specific PLDM Type and version as a responder



PLDM for SMBIOS Data Transfer – Why we need it?

- SMBIOS provides low-level platform asset inventory info
- A Management Controller
 - Typically provides platform inventory information via a CIM-based external interface
 - May wish to utilize the data in the SMBIOS structure table as a data source for providing platform inventory information
- Providing inventory information in low power states
 - May require maintaining multiple copies of SMBIOS structure table data within a platform, and
 - Keeping the copies consistent between the MC and the SMBIOS table information that is accessed by the system software/BIOS.
- Thus, there is a need for a platform-level data model (PLDM) for SMBIOS data transfer
- PLDM for SMBIOS data transfer can be used between the system firmware (BIOS) and a management controller, and between management controllers



PLDM for SMBIOS Data Transfer Design

- Defines commands to obtain the SMBIOS structure table metadata
- Preserves SMBIOS structure format
- Defines commands to transfer SMBIOS structure data
- Supports both pull/push models
 - Example: BIOS initiating the transfer of its SMBIOS table to a MC
 - Example: A management controller sending read requests to BIOS telling it to provide SMBIOS structure table data
- Defines a data integrity check to protect the SMBIOS structure table data transfer and SMBIOS structure data transfer
- Defines commands to read SMBIOS structure data by type or by handle to enable reading a subset of structures



PLDM Representation of SMBIOS Structure

PLDM Representation of an SMBIOS Structure

Byte	Type	Field
0	uint8	Type as defined in DSP0134
1	uint8	Length (L bytes) as defined in DSP0134
2:3	uint16	Handle as defined in DSP0134
4:L-1	–	The formatted area of the structure
Variable	–	Variable bytes of unformatted area of the structure terminated by double null (0000h) as defined in DSP0134

PLDM Representation of SMBIOSStructureData

Byte	Type	Field
Variable	–	SMBIOS structures (one or more) See Table 1 for the PLDM representation of an SMBIOS structure.
Variable	...	Pad 0 to 3 number of pad bytes. The value stored in each pad byte is 0x00.
	uint32	SMBIOSStructureDataIntegrityChecksum Integrity checksum on the SMBIOS structure data including the pad bytes (if any). It is calculated starting at the first byte of the PLDM representation of SMBIOSStructureData. For this specification, the CRC-32 algorithm with the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ (same as the one used by IEEE 802.3) shall be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first.

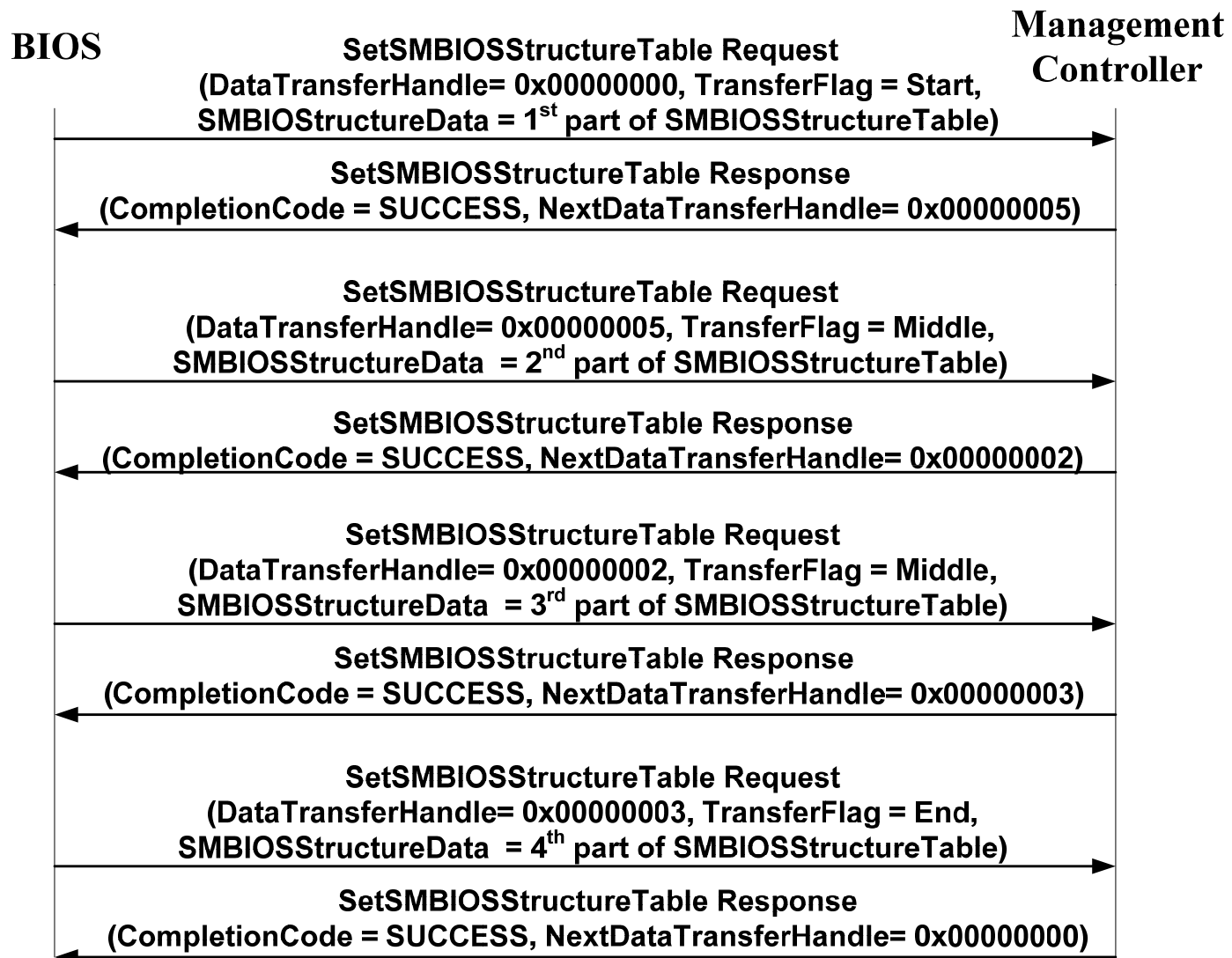


PLDM Commands for SMBIOS Data Transfer

- **GetSMBIOSStructureTableMetadata**
 - Used to get the SMBIOS structure table metadata information
- **SetSMBIOSStructureTableMetadata**
 - Used to set the SMBIOS structure table metadata information
- **GetSMBIOSStructureTable**
 - Used to get the SMBIOS structure table data
 - Data can be transferred using a sequence of one or more command/response messages
- **SetSMBIOSStructureTable**
 - Used to push the SMBIOS structure table data
 - Defined to allow the data to be transferred using a sequence of one or more command/response messages
- **GetSMBIOSStructureByType**
 - Used to get the SMBIOS structures of a specific type
 - Defined to allow the SMBIOS structure data to be transferred using a sequence of one or more command/response messages
- **GetSMBIOSStructureByHandle**
 - Used to get the SMBIOS structure by a specific handle
 - Defined to allow the SMBIOS structure data to be transferred by using a sequence of one or more command/response messages



Multipart SMBIOS Structure Table Transfer Example

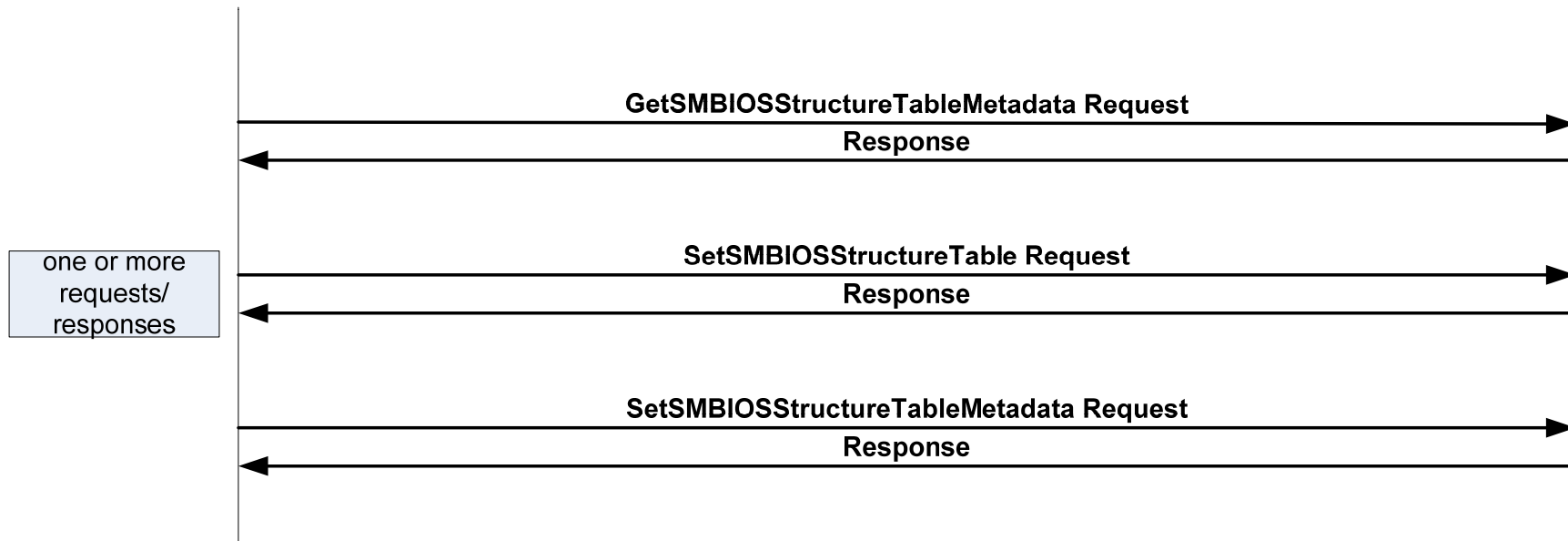




SMBIOS Structure Table Transfer Example

BIOS

**Management
Controller**





PLDM for BIOS Control/Configuration

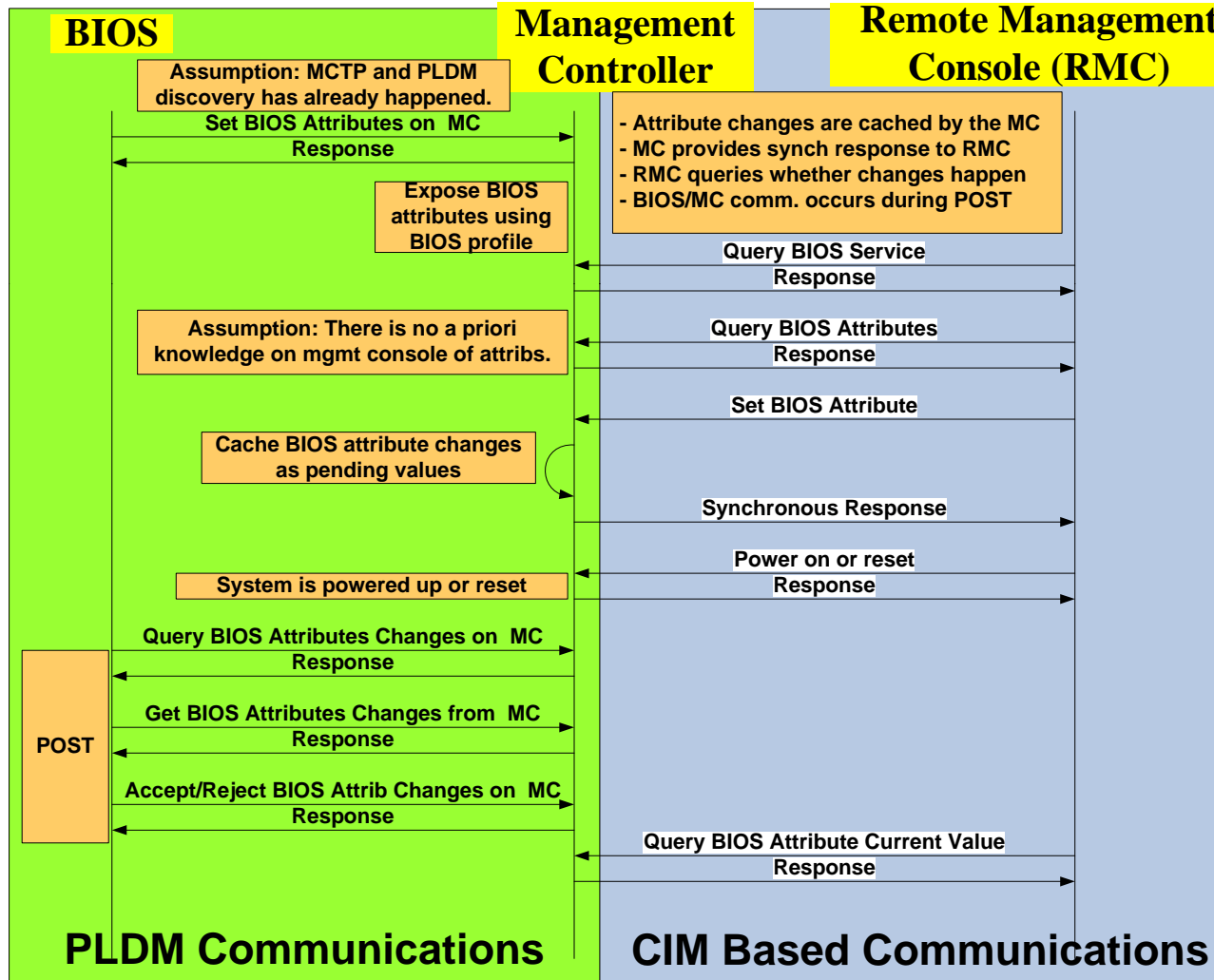
- Complementary to Boot control and BIOS Management CIM profiles
- Provides internal communications model to exchange the BIOS configuration/control data
- Defines the PLDM data structures/messages for communicating
 - BIOS settings
 - BIOS attributes
 - Boot configurations
 - Boot order settings



BIOS Attribute Update Models

- Immediate update model
 - MC acts as a pass-through device
 - When a remote management console updates a BIOS attribute on the MC, the MC immediately updates the BIOS attributes
- Deferred update model
 - BIOS attribute changes are not done immediately
 - BIOS attribute changes made by remote management console are cached as pending changes
 - Pending changes do not take effect until the next time the BIOS runs
 - The next time the BIOS runs, the MC provides the BIOS attribute changes to the BIOS
 - The BIOS processes the update and informs the MC whether it accepted or rejected the changes

BIOS/MC Communication Example





BIOS String, Attribute, and Value Tables

BIOS String Table

BIOS String Handle	BIOS String
Handle 1	String 1
Handle 2	String 2
...	...

- Useful for efficient BIOS attributes data transfer between BIOS/MC
- Not exposed to remote mgmt console
- Provide common mechanism for BIOS/MC to communicate mapping

BIOS Attribute Table

Attribute Handle	Attribute Type	Attribute Name Handle	Type Specific Fields
Attrib 1 Handle	Attrib 1 Type	Attrib1 Name Handle
Attrib 2 Handle	Attrib 2 Type	Attrib2 Name Handle	...
...

BIOS Attribute Value Table

Attribute Handle	Attribute Type	Type Specific Value(s)
Attrib 1 Handle	Attrib 1 Type	...
Attrib 2 Handle	Attrib 2 Type	...
...

An Example of BIOS Tables

BIOS String Table

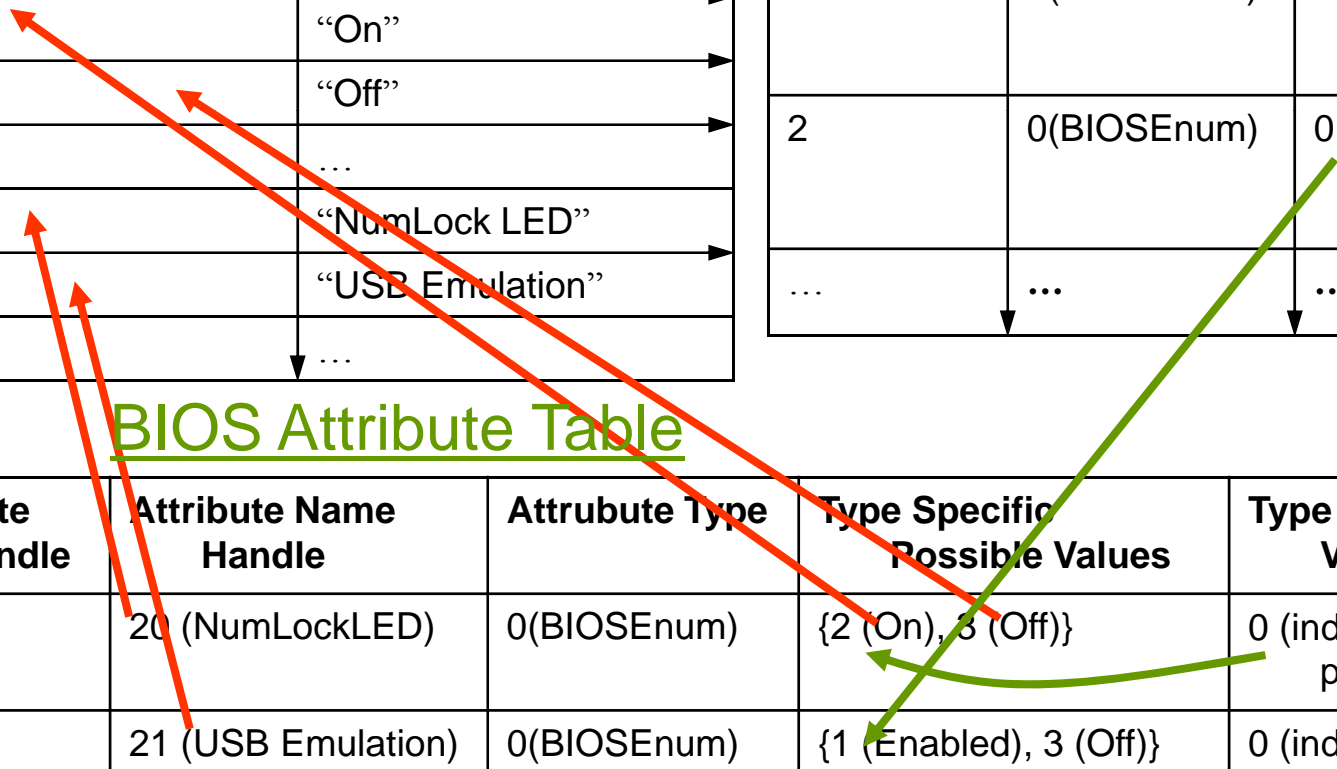
BIOS String Handle	BIOS String
1	“Enabled”
2	“On”
3	“Off”
...	...
20	“NumLock LED”
21	“USB Emulation”
...	...

BIOS Attribute Value Table

Attribute Handle	Attribute Type	Type Specific Current Value(s)
1	0(BIOSEnum)	0 (index into the array of possible values)
2	0(BIOSEnum)	0 (index into the array of the possible values)
...

BIOS Attribute Table

Attribute Handle	Attribute Name Handle	Attribute Type	Type Specific Possible Values	Type Specific Default Value(s)
1	20 (NumLockLED)	0(BIOSEnum)	{2 (On), 3 (Off)}	0 (index into the array of possible values)
2	21 (USB Emulation)	0(BIOSEnum)	{1 (Enabled), 3 (Off)}	0 (index into the array of the possible values)
...



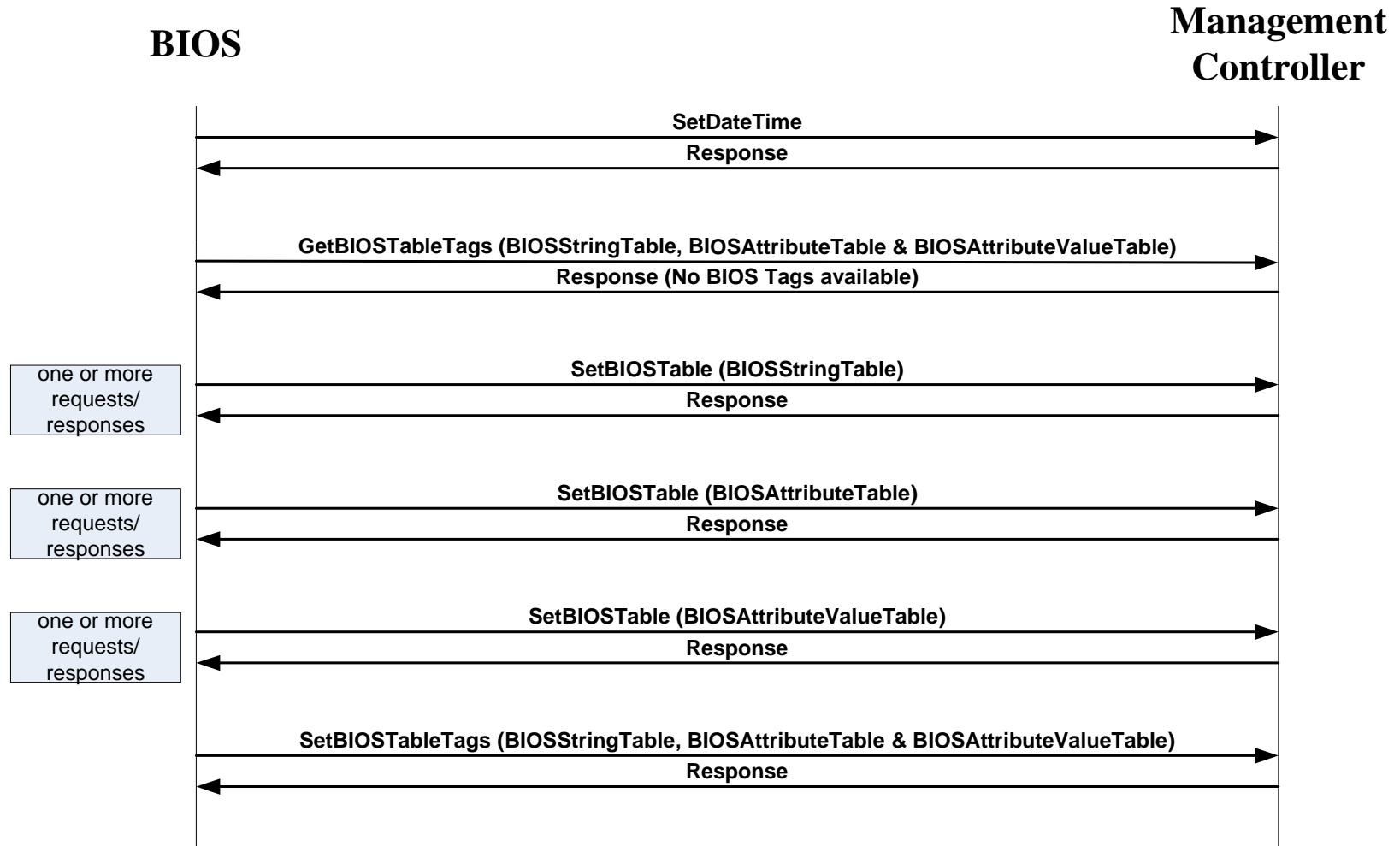


PLDM Commands for BIOS Control/Configuration

- **GetBIOSTable**
- **SetBIOSTable**
- **UpdateBIOSTable**
- **GetBIOSTableTags & SetBIOSTableTags**
- **AcceptBIOSAttributesPendingValues**
- **SetBIOSAttributeCurrentValue**
- **GetBIOSAttributeCurrentValueByHandle**
- **GetBIOSAttributePendingValueByHandle**
- **GetBIOSAttributeCurrentValueByType**
- **GetBIOSAttributePendingValueByType**
- **GetBIOSStringTableStringType**
- **SetBIOSStringTableStringType**

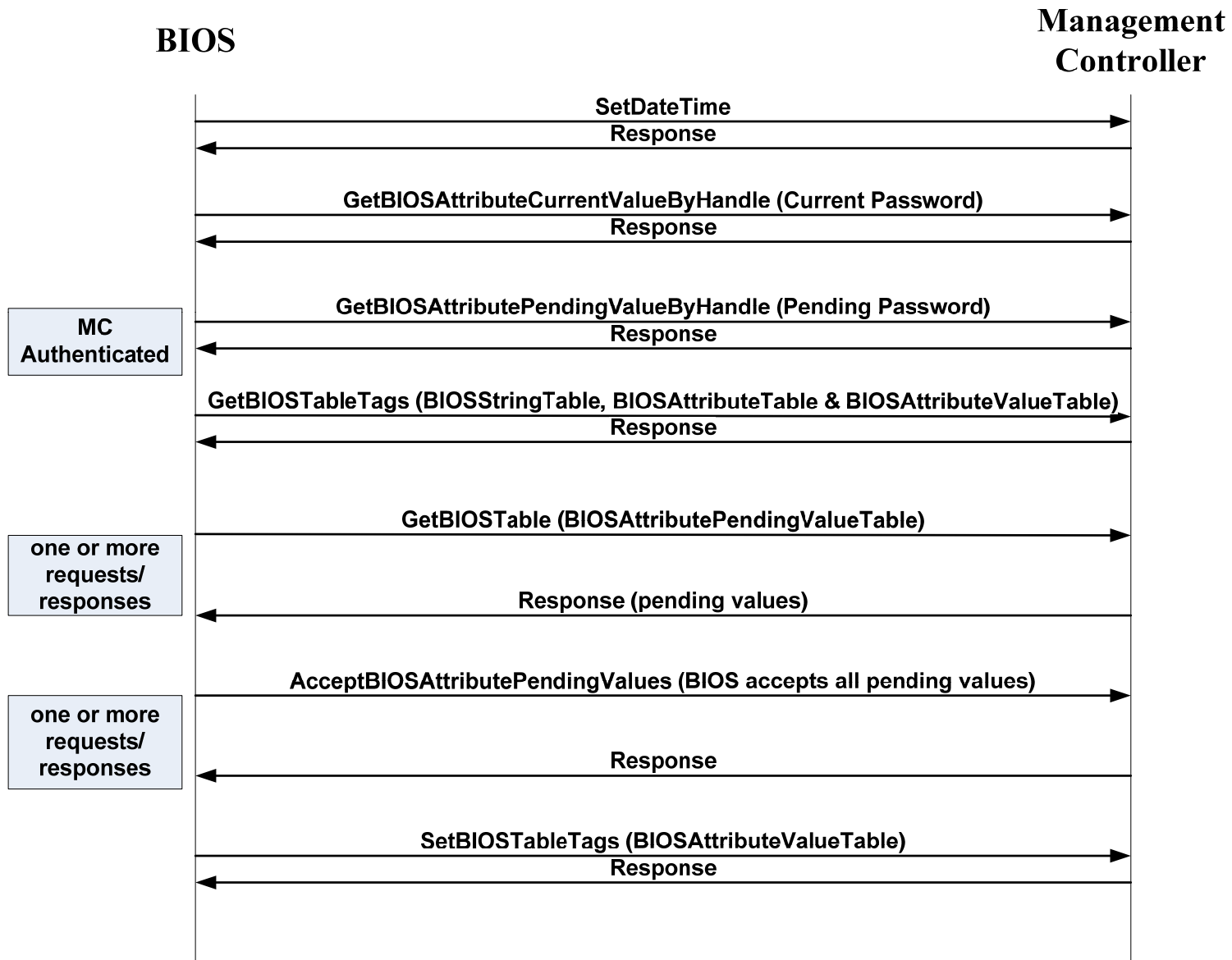


Example of BIOS Table Initialization





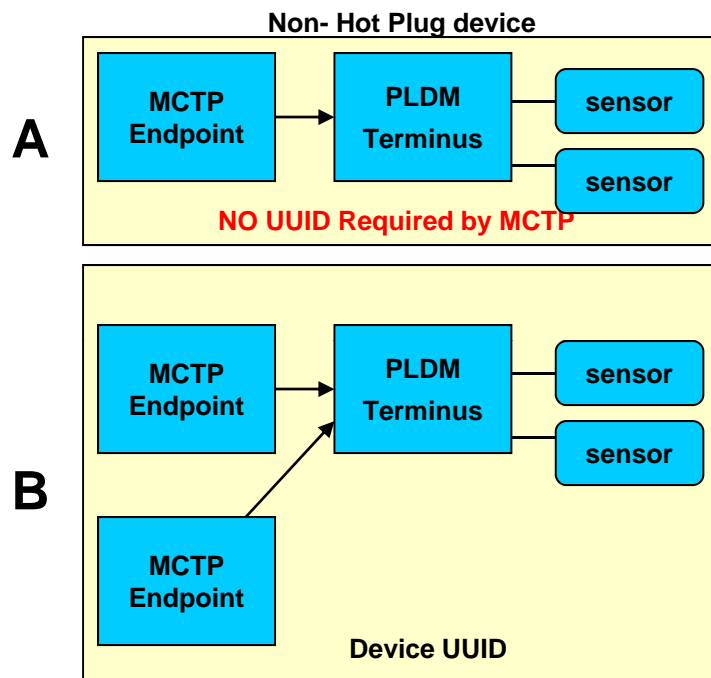
Remote BIOS Setting Changes Acceptance Example



PLDM for Monitoring & Control

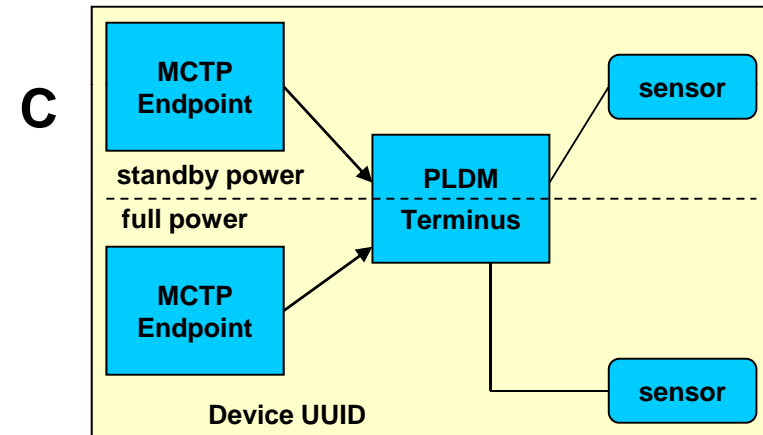
- Provides Platform model for monitoring
 - Sensors
 - Numeric Sensors with Thresholds
 - State Sensors
- Provides Platform model for control
 - via PLDM Effecters
- Provides Platform model for Events
 - defines “Event Messages” for asynchronous reporting of state changes from sensors
 - defines an Event Log for ‘post mortem’ analysis of platform events

PLDM Terminus



Terminus Commands

- SetTID (see DSP0240)
- GetTID (see DSP0240)
- GetTerminusUID



- A PLDM Terminus is the communication terminus of PLDM Messages
 - All Termini interpret PLDM requests and return responses
 - Termini may also generate requests
- Uses an eight-bit “Terminus ID” (TID) to identify which terminus originated a message.
 - Primarily used for identifying which terminus issued an Event Message
- TIDs are assigned to termini during PLDM subsystem initialization
- Hot-plug devices additionally provide a UUID

PLDM Numeric Sensors

- Used to represent a numeric reading
 - Reading is represented as an integer
- Sensor commands do not identify units/scaling
 - MAP performs the conversion
- Numeric sensor thresholds
 - Thresholds is associated with a severity
 - Warning, critical, and fatal
 - Readable and settable thresholds
- Numeric sensor status
 - Present status: Based on comparing reading against thresholds
 - Event status: Based on transitions btwn different monitored states
 - Auto-arm sensors auto update event status based on state transitions
 - Manual-rearm sensors auto update event status on detecting a worsening transition



PLDM State Sensors

- Represent a particular state from a set of state information
 - The set can be small, e.g. on | off, present | absent
 - Or lengthy, e.g. "Other", "Unknown", "Running/Full Power", "Warning", "In Test", "Not Applicable", "Power Off", "Off Line", "Off Duty", "Degraded", "Not Installed", "Install Error", "Power Save - Unknown", "Power Save - Low Power Mode", "Power Save - Standby", "Power Cycle", "Power Save - Warning", "Paused", "Not Ready", "Not Configured", "Quiesced"
- State Sensors can be used to generate platform events
 - Similar to Numeric Sensors for maintaining 'threshold crossing' status and generating threshold crossing events
- Composite state sensors
 - Enable multiple sets of state information to be returned in a single reading
 - Individual state sensors will typically map to an individual instance of CIM_Sensor or a particular CIM object property that returns state information
 - Same number of bytes always returned regardless of which sensors are enabled
 - Individual sensors are indexed under Sensor Access ID number, starting from 0.
- State Sets
 - Common definition of enumerations for states used in PLDM
 - Being designed to map to CIM properties that return platform states



PLDM State Sets and Entities

- **State Sets IDs**

- A definition of enumerated State Sets and Entity types used by PLDM for reporting state and identifying types of managed entities
- Standard State Set IDs defined in DSP0249
- Includes an OEM State Set ID Range

- **PLDM Entity ID Codes**

- A PLDM Entity ID code represents a physical or logical Entity within the PLDM Subsystem
 - E.g. Processor, Fan, Door
- Standard Entity ID Codes defined in DSP0249
- Includes an OEM Entity ID Code Range

Sensor Commands

Numeric Sensor Commands

- **SetSensorEnable**
 - Enable/disable event generation and/or monitoring by sensor
- **GetSensorReading**
 - get numeric value and threshold event state
- **GetSensorThresholds**
 - get present threshold settings
- **SetSensorThresholds**
 - set threshold values
- **RestoreSensorThresholds**
 - restore default thresholds
- **GetSensorHysteresis**
 - get threshold hysteresis settings
- **SetSensorHysteresis**
 - get threshold hysteresis settings
- **InitNumericSensor**
 - set initial numeric reading and event states

State Sensor Commands

- **SetStateSensorEnables**
 - Enable/disable event generation and/or monitoring by sensor
- **GetStateSensorReadings**
 - get present and previous state values
- **InitStateSensor**
 - set initial sensor and event states



Event Logging

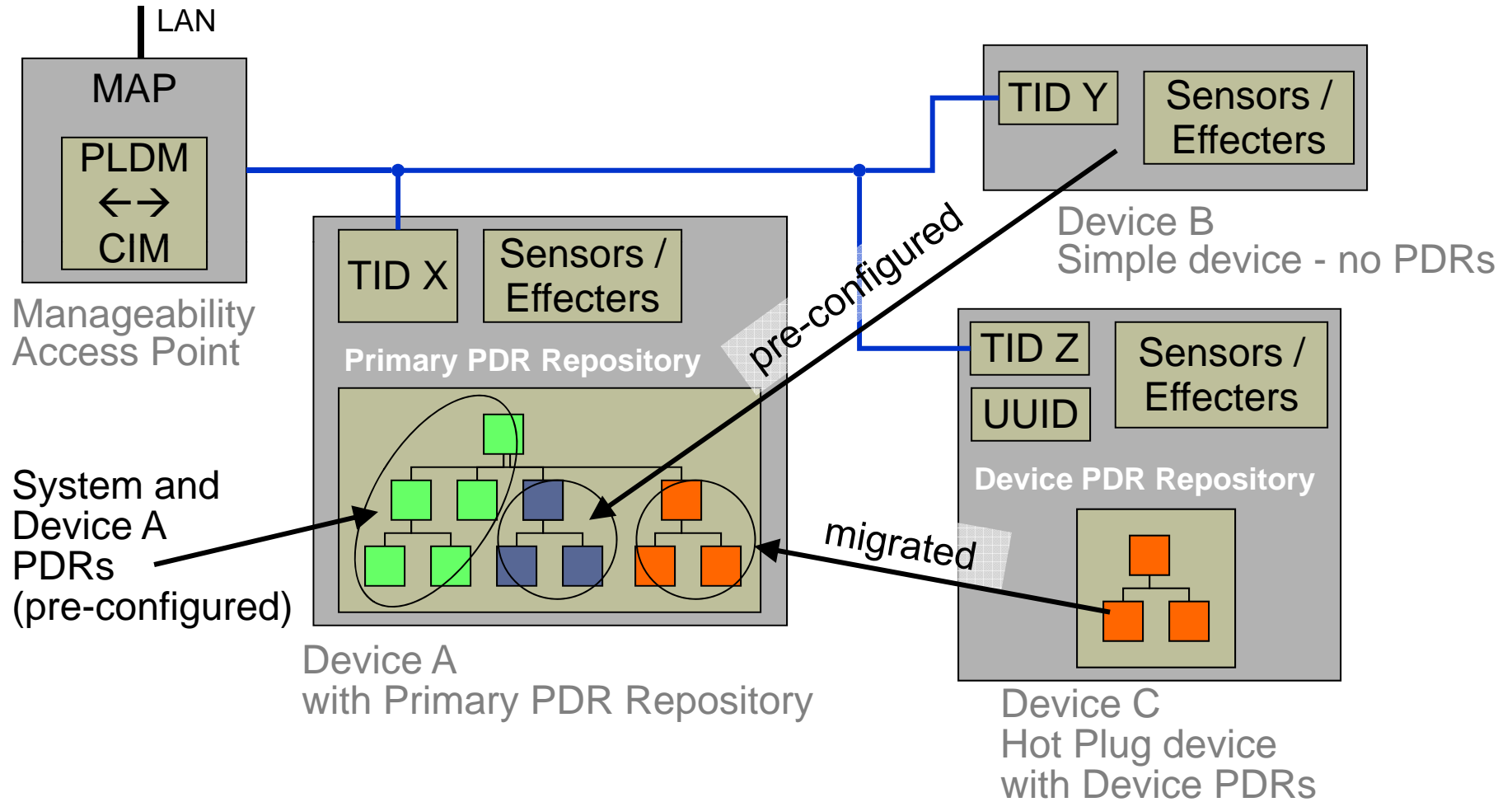
- Provides ability to save PLDM Event Messages in a non-volatile log
 - Event Receiver function internally routes messages to Event Logging function
- Event Logging function:
 - manages log access
 - timestamps entries
 - performs optional automatic log clearing policy
- Event Log commands
 - GetPLDMEventLogInfo
 - EnablePLDMEventLogging
 - ClearPLDMEventLog
 - GetPLDMEventLogTimestamp
 - SetPLDMEventLogTimestamp
 - ReadPLDMEventLog
 - GetPLDMEventLogPolicyInfo
 - SetPLDMEventLogPolicy



Platform Descriptor Records

- PDR = Platform Descriptor Record
- PDRs provides a data structure for delivering collections of semantic and association information for the platform management subsystem
 - Semantic information about Sensors and Effecters
 - E.g. Units / State Set, Resolution, Tolerance, Accuracy, etc.
 - Information about the associations between sensors & monitored entities
 - E.g. that temperature sensor 1 is associated to processor 1 on module 3.
 - Other platform associations and capabilities
 - Interrupt source-to-target associations
 - Sensor / Effector initialization information
 - OEM Units, OEM State Sets
 - etc.

PDR Usage





PLDM for FRU Data

Why we need it?

- FRU data is used to provide low-level asset information
- A Management Controller
 - Typically provides FRU information via a CIM-based external I/F
 - May wish to utilize the data in the low level FRU data structures as a data source for providing FRU information
- Providing FRU information may require
 - Maintaining one or more FRU tables within a platform
 - Keeping the FRU information consistent and current
 - Ability to discover and correlate FRU information
- Thus, there is a need for PLDM for FRU
- PLDM for FRU can be used to communicate FRU information between management subsystem components



PLDM for FRU Data Design

- Defines a structure for a table representation of FRU data records
- Defines commands to access and modify FRU table metadata
- Defines commands to transfer FRU data records
- Supports both pull/push models
 - Example: MC initiating the transfer of FRU records from a device
 - Example: A device updating FRU records on a MC
- Defines a data integrity check to protect the FRU record data transfer
- Defines commands to read FRU data records by record set identifier or record type or record field type



PLDM Representation of FRU Data

PLDM Representation of a FRU Record

Size	Type	Field
2 Bytes	uint16	FRU Record Set Identifier #1
1 Byte	uint8	FRU Record Type #1
1 Byte	uint8	Number of FRU fields
1 Byte	uint8	Encoding Type for FRU fields 0 = Unspecified 1 = strASCII 2 = strUTF8 3 = strUTF16 4 = strUTF16-LE 5 = strUTF16-BE 6-255 = reserved
1 Byte	uint8	FRU Field #1 Type
1 Byte	uint8	FRU Field #1 Length
Up to 255 bytes	Determined by Field Type	FRU Field #1 Value
1 Byte	uint8	FRU Field #2 Type
1 Byte	uint8	FRU Field #2 Length
Up to 255 bytes	Determined by Field Type	FRU Field #2 Value
....
....
1 Byte	uint8	FRU Field #n Type
1 Byte	uint8	FRU Field #n Length
Up to 255 bytes	Determined by Field Type	FRU Field #n Value

FRU Record Types

Record Type	Description
0	Reserved
1	General FRU Record
2 – 253	Reserved
254	OEM FRU Records
255	Reserved

FRU Record Field Types

Field Type Number	Field Type Description	Field Format
0	Reserved	N/A
1	Chassis Type	String
2	Model	String
3	Part Number	String
4	Serial Number	String
5	Manufacturer	String
6	Manufacture Date	Timestamp104
7	Vendor	String
8	Name	String
9	SKU	String
10	Version	String
11	Asset Tag	String
12	Description	String
13	Engineering Change Level	String
14	Other Information	String
254	OEM	String
255	Reserved	N/A



PLDM Commands for FRU Data Transfer

- **GetFRURecordTableMetadata**
 - Used to get the FRU Record table metadata information
- **SetFRURecordTableMetadata**
 - Used to set the FRU Record table metadata information
- **GetFRURecordTable**
 - Used to get the table of FRU records
 - Data can be transferred using one/more command/response messages
- **SetFRURecordTable**
 - Used to push the table of FRU records
 - Data can be transferred using one/more command/response messages
- **GetFRURecordByOption**
 - Used to get the FRU records of a specific record set and/or record type
 - Allows specific fields of FRU records to be transferred
 - Data can be transferred using one/more command/response messages



Summary

- Platform Management Component Intercommunications (PMCI) Working Group Focus
 - Define “Inside the box” communication & functional interfaces between management subsystem components
- PMCI protocols/interfaces are complementary to DMTF CIM Profiles and remote access protocols
- Network Controller Sideband Interface (NC-SI)
 - A sideband interface/protocol to transfer management traffic between management controller/network controller
- Management Component Transport Protocol (MCTP)
 - Flexible, scalable, compact, and low overhead transport optimized for internal communications within a platform
 - Designed to carry multiple message types
 - Provides simplified message routing
 - Supports add-in cards, hot plug devices, and fixed address/simple devices
 - Several transport bindings are defined: SMBus, PCIe VDM, KCS, Serial...
- Platform Level Data Model (PLDM)
 - An interface/data model that provides efficient access to low-level management data/functions within a platform
 - Provides a simple request/response style communication model
 - Messages/data transfer is transport independent
 - Extensible and flexible
 - PLDM for SMBIOS, PLDM for BIOS Control/Configuration, PLDM for Platform Monitoring/Control, PLDM for FRU Data..



PMCI Specifications

DSP#	Rev	Document	Status
DSP0222	1.0	Network Controller Sideband Interface (NC-SI) Specification	DMTF Standard
MCTP SPECS			
DSP0236	1.0 1.1	Management Component Transport Protocol (MCTP) Base Specification	DMTF Standard
DSP0237	1.0	Management Component Transport Protocol SMBus/I2C Transport Binding Specification	DMTF Standard
DSP0238	1.0	Management Component Transport Protocol (MCTP) PCIe VDM Transport Binding Specification	DMTF Standard
DSP0239	1.0 1.1	Management Component Transport Protocol (MCTP) IDs and Codes	DMTF Standard
DSP0253	1.0	MCTP Serial Transport Binding	DMTF Standard
DSP0254	1.0	MCTP KCS Transport Binding	DMTF Standard
DSP0256	1.0	MCTP Host Interface	DMTF Standard
WHITE PAPERS			
DSP2015		Platform Management Component Intercommunications (PMCI) Architecture White Paper	Published
DSP2016		Management Component Transport Protocol (MCTP) Overview White Paper	Published



PMCI Specifications

DSP#	Rev	Document	Status
PLDM SPECIFICATIONS			
DSP0240	1.0	PLDM Base Specification - common formats for PLDM messages, and common terminology and conventions for PLDM specifications	DMTF Standard
DSP0241	1.0	PLDM Over MCTP Binding - how PLDM messages are transported using MCTP	DMTF Standard
DSP0245	1.0	PLDM IDs and Codes - codes and values that are common across the PLDM specifications	DMTF Standard
DSP0246	1.0	PLDM for SMBIOS Data Transfer	DMTF Standard
DSP0247	1.0	PLDM for BIOS Control and Configuration	DMTF Standard
DSP0248	1.0	PLDM for Platform Monitoring and Control	DMTF Standard
DSP0249	1.0	PLDM States Sets Specification - definition of enumerated State Sets and Entity types used by PLDM for reporting state/identifying types of managed entities	DMTF Standard
DSP0248	1.0	PLDM for FRU	In development
DSP0844	1.0	PLDM CIM Mapping specification	To be started



Acknowledgements

Thanks to all the contributors and participants of PMCI Working Group!