



# **PCIe® 3.0 / Post-3.0 Protocol Update**

**Joe Cowan**  
**Hewlett-Packard Company**



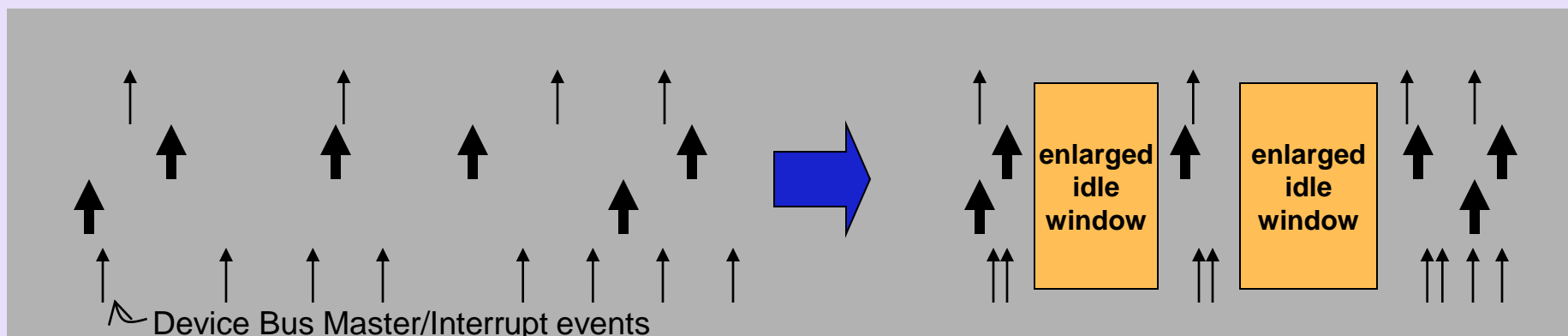
# PCIe 3.0 / Post-3.0 Protocol Update

- ECNs Incorporated into the PCIe 3.0 Base Spec
  - ✓ Optimized Buffer Flush & Fill (OBFF)
  - ✓ ASPM Optionality
  - ✓ End-End TLP Prefix Changes for RCs
  - ✓ Protocol Multiplexing (PMUX)
- ECNs Currently Under Development
  - ✓ Process Address Space ID (PASID)
  - ✓ Lightweight Notification (LN) Protocol
- PCI Code and ID Assignments
  - ✓ Class Code & Capability ID Extraction ECN
  - ✓ PCI Code and ID Assignment Specification
  - ✓ PCI Code & ID Assignment Specification Update ECN

# Optimized Buffer Flush & Fill (OBFF)

# Reducing Platform Power With Optimized Buffer Flush/Fill

- Problem statement: devices do not know power state of central resources
  - ✓ “Asynchronous” device activity prevents optimal power management of memory, CPU, RC internals by idle window fragmentation
  - ✓ Premise: If devices knew when to talk, most could easily optimize their Request patterns
    - Result: System would stay in lower power states for longer periods of time with no impact on overall performance
- Optimized Buffer Flush/Fill (OBFF) – a mechanism for broadcasting PM hint to device



# How to do OBFF?

## Optimal Windows

- **CPU Active** – Platform fully active. Optimal for bus mastering and interrupts
- **OBFF** – Platform memory path available for memory read and writes
- **Idle** – Platform is in low power state

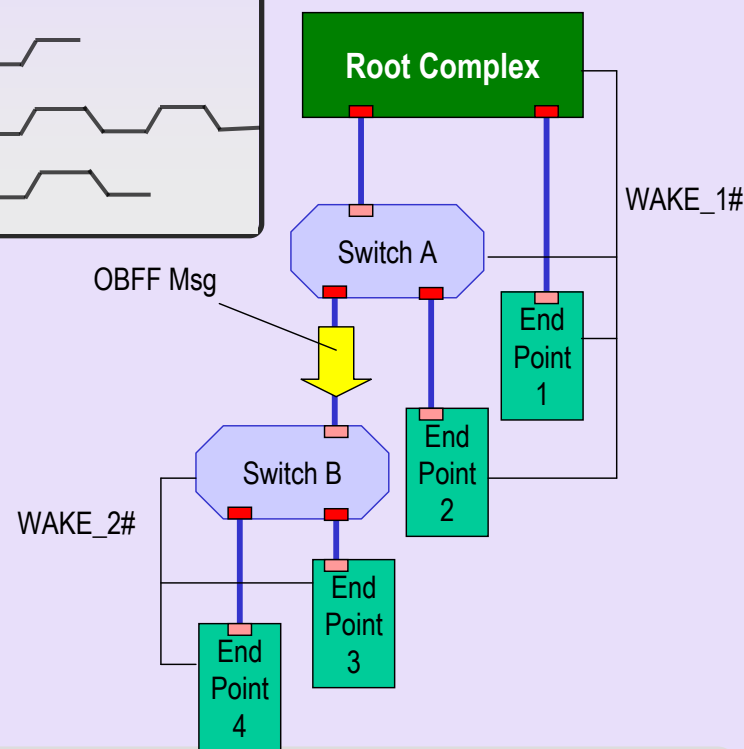
## WAKE# Waveforms

### Transition Event

### WAKE#

Idle → OBFF .....	
Idle → CPU Active .....	
OBFF/CPU Active → Idle .....	
OBFF → CPU Active .....	
CPU Active → OBFF .....	

- Requirements:
  - ✓ Notify all Endpoints of optimal windows with minimal power impact
  - ✓ Keep it Simple – Maximize cost/benefit
- Solution 1: When possible, use WAKE# with expanded meanings
- Solution 2: WAKE# not available – Use PCIe Message



Greatest Potential Improvement When Implemented by All Platform Devices

# ASPM Optionality

# ASPM Optionality

- Permits full matrix of L0s and L1 support for ASPM
  - ✓ Prior to this ECN, all PCIe External Links were required to support ASPM L0s

Table 5-3: Encoding of the ASPM Support Field	
Field	Description
ASPM Support	00b – <del>Reserved</del> <a href="#">No ASPM support</a>
	01b – L0s supported
	10b – <del>Reserved</del> <a href="#">L1 supported</a>
	11b – L0s and L1 supported

- Clarifies that software must not enable L0s in either direction on a given link unless components on both sides of the Link each support L0s
- Defines a new Capability bit ASPM Optionality Compliance, which software can use to help determine whether to:
  - ✓ Enable ASPM and/or
  - ✓ Run ASPM compliance tests

# End-End TLP Prefix Changes for RCs



# End-End TLP Prefix Changes for RCs

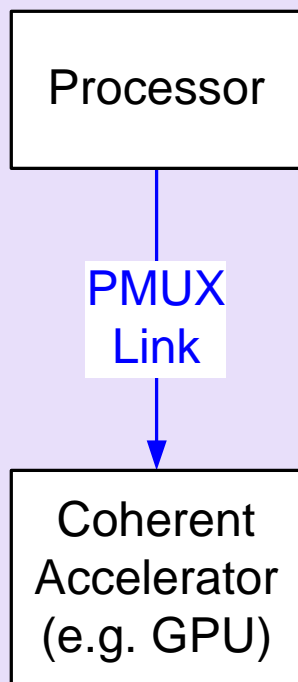
- TLP Prefix was an ECN against the 2.0 Base Spec
  - ✓ Prefixes are one or more DWORDs prepended to a standard TLP Header, in order to carry additional TLP information.
  - ✓ Local TLP Prefixes exist on a single Link.
  - ✓ End-End TLP Prefixes are carried end-to-end from source to destination.
- The End-End TLP Prefix Changes for RCs ECN:
  - ✓ Permits different RPs supporting EE TLP Prefixes to report different values for the Max End-End TLP Prefixes field.
    - Important for multi-component RC implementations, but requires a change in the software programming model
  - ✓ Changes & clarifies error handling requirements for an RP that receives a TLP with more EE TLP Prefixes that it supports.
    - Permits & encourages handling this case as a Non-Fatal Error instead of a Fatal Error

# Protocol Multiplexing (PMUX)

# PMUX Overview

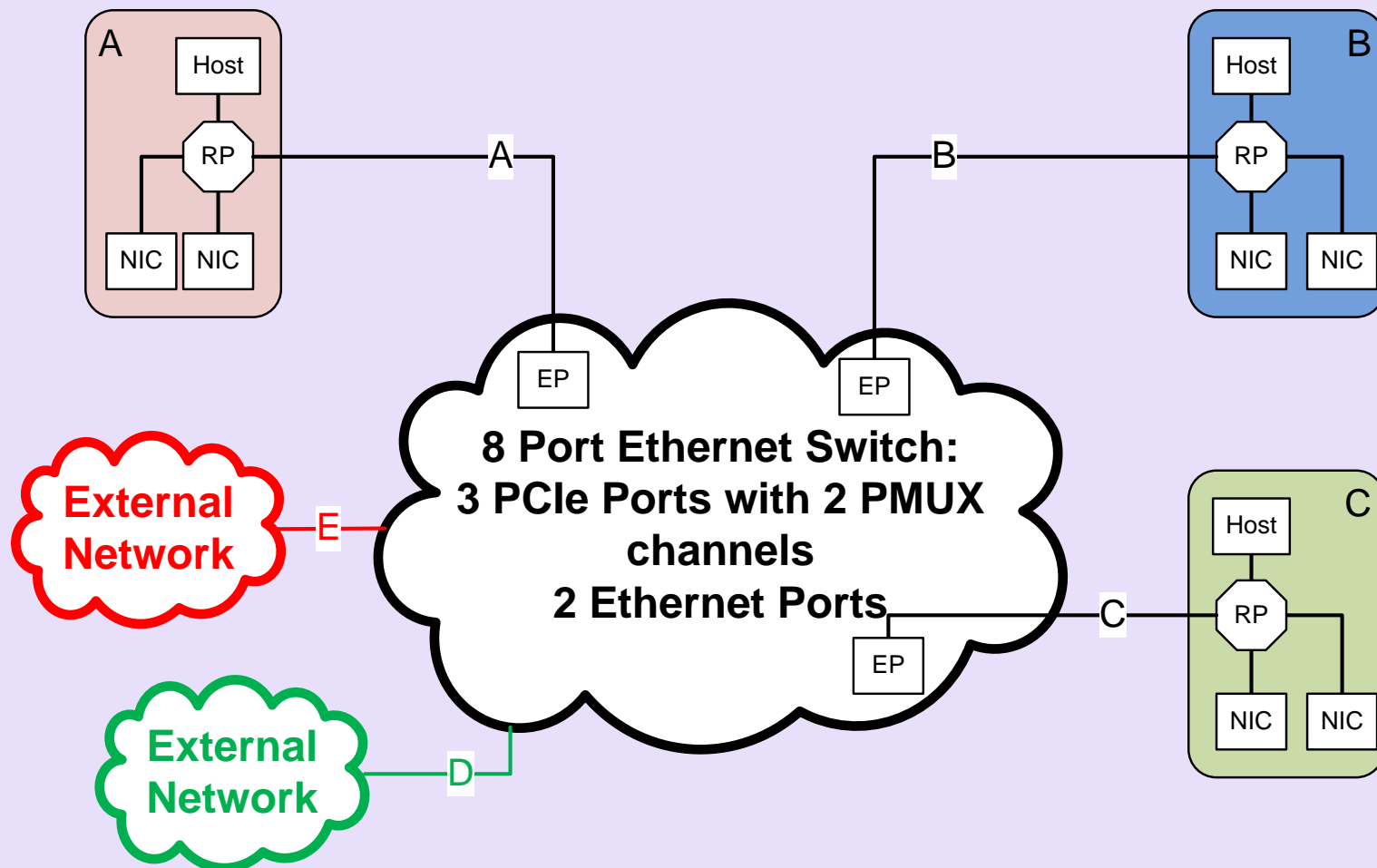
- Added to the *PCI Express 3.0 Base Specification*
  - ✓ “ECN” was against the 0.9 draft of the 3.0 Base Spec
- Supports multiple protocols concurrently on a PCIe Link
  - ✓ Independent protocol streams, independent buffering and flow control resources, independent sequence numbers, etc.
  - ✓ Up to 4 concurrently active PMUX Channels on each Link
- No inherent flow control or packet retransmission
  - ✓ If needed, provided by the protocol within each PMUX Channel
- Link local
  - ✓ Routing is not part of PMUX, but has been considered for possible future work
- Minimal impact to existing PCIe framing protocol
  - ✓ No efficiency impact to PCIe protocol packets

# Example PMUX Usage: Coherent Accelerator

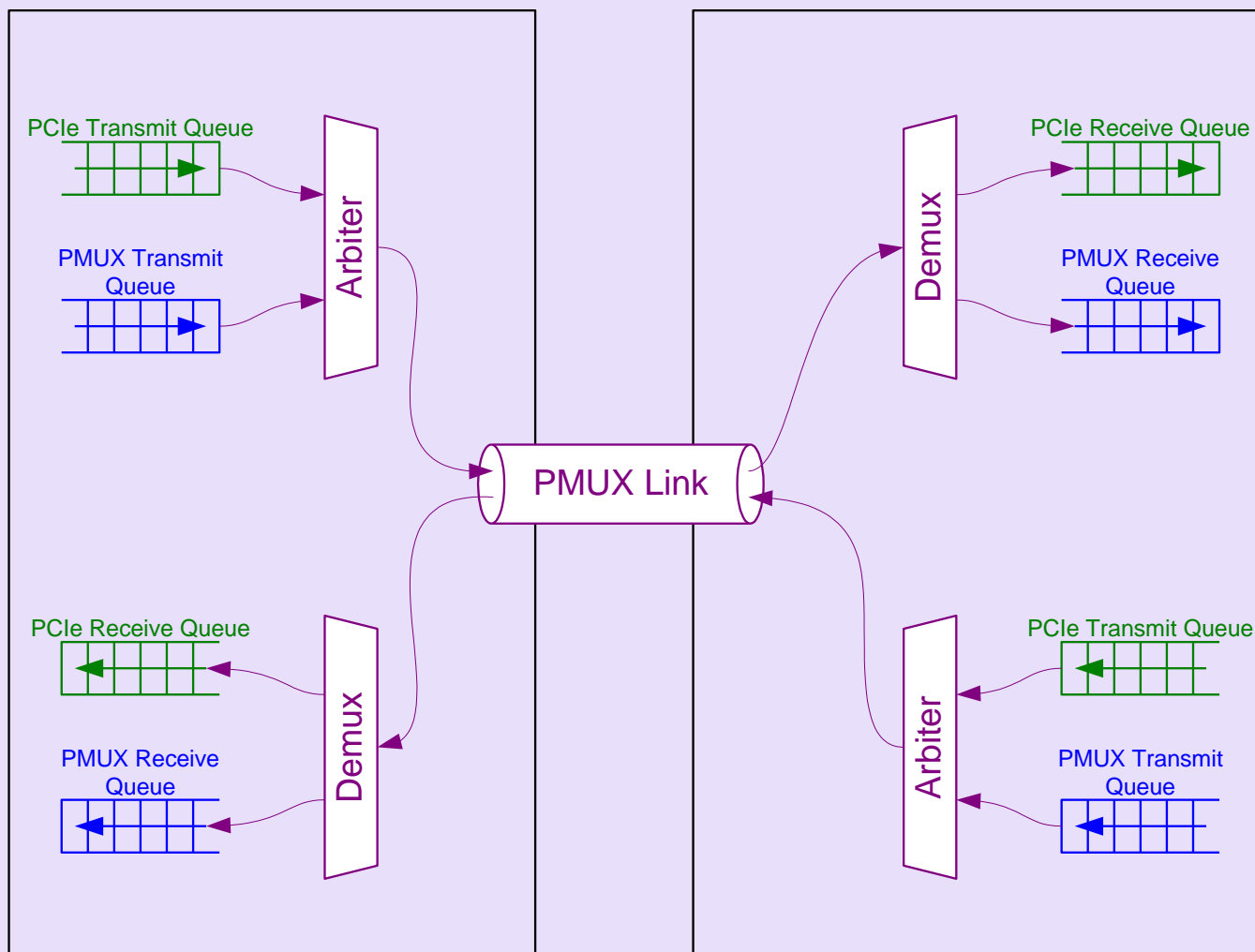


- PMUX link enables a coherent accelerator to participate in the processor's coherency protocol
  - ✓ In addition to PCIe traffic
  - ✓ Same accelerator might use PCIe exclusively on other processor platforms

# Example PMUX Usage: Ethernet Switching



# PMUX Mux/Demux Operation



# Key PMUX Attributes

- Extremely low per-packet overhead compared to PCIe Message encapsulation approaches
  - ✓ 6.5 bytes: PMUX Overhead
  - ✓ 24 bytes: PCIe MsgD Overhead
- Does not impact and is not affected by PCIe flow control & buffering requirements
  - ✓ PCIe flow control gets in the way for some protocols
  - ✓ Deadlock issues with non-tree topologies
  - ✓ PCIe flow control requires at least MPS-sized buffers in each VC
- Does not impact and is not affected by PCIe Ack / Nak
  - ✓ Retransmission not required for some protocols
- Enables low latency routing compared to Message encapsulation
  - ✓ PMUX packets are identified early
  - ✓ Routing decision can be made 17.5 bytes earlier

# PMUX Extended Capability

31	0	Byte Offset
PMUX Extended Capability Header		00h
PMUX Capability		04h
PMUX Control		08h
PMUX Status		0Ch
PMUX Protocol Array [1]		10h
PMUX Protocol Array [2]		14h
...		:
PMUX Protocol Array [62]		104h
PMUX Protocol Array [63]		108h

- PMUX supported if capability is present
- PMUX default is disabled
  - ✓ Software enables
- Variable-length capability
  - ✓ Based on PMUX Protocol Array Size
  - ✓ Each supported PMUX Protocol identified by Authority ID / Protocol ID
- Any given PMUX Port supports up to 63 PMUX Protocols
  - ✓ Up to 4 enabled at a time
  - ✓ Any PMUX Channel can carry any supported Protocol
  - ✓ Protocols selected by PMUX Protocol Array index



# Process Address Space ID (PASID)

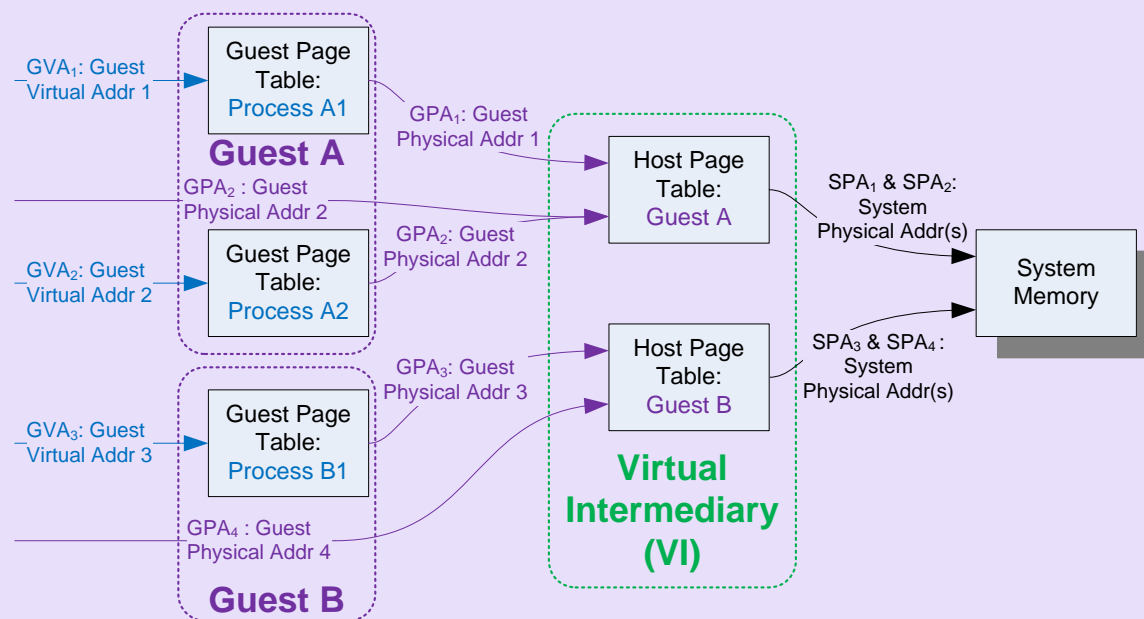
# IOV Background

- Address Translation Services (ATS) supports:
  - ✓ Performance optimization for direct assignment of a Function to a Guest OS running on a Virtual Intermediary (Hypervisor)
- Page Request Interface (PRI) supports:
  - ✓ Functions that can raise a Page Fault
- Single Root-I/O Virtualization (SR-IOV) supports:
  - ✓ Light-weight Functions (Virtual Functions)
  - ✓ Large numbers of Functions (multiple Bus Numbers)

# PASID Overview

- Supports **Direct Assignment** of I/O to a **User Process** running on a Guest OS running on a Virtual Intermediary
  - ✓ Untranslated Memory Requests
  - ✓ Translation Requests
  - ✓ Translation Invalidations
  - ✓ Page Requests
- Supports **Execute Permission**
- Supports **Privileged Mode**

# PASID Address Mapping



Address Translation Cache (either in TA or in Function's ATC)

GVA <sub>1</sub> /A1 → SPA <sub>1</sub>
GVA <sub>2</sub> /A2 → SPA <sub>2</sub>
GPA <sub>2</sub> → SPA <sub>2</sub>
GVA <sub>3</sub> /B1 → SPA <sub>3</sub>
GPA <sub>4</sub> → SPA <sub>4</sub>

Cache Entry Type	Meaning
GVA/PASID → SPA	TA / ATC Entry with PASID
GPA → SPA	TA / ATC Entry without PASID

- 3 User Processes
- 2 Guests
- Cache Example:
  - ✓ 3 GVA Entries
    - GVA<sub>1</sub> / A1 → SPA<sub>1</sub>
    - GVA<sub>2</sub> / A2 → SPA<sub>2</sub>
    - GVA<sub>3</sub> / B1 → SPA<sub>3</sub>
    - Intermediate GPA not cached
  - ✓ 2 GPA Entries
    - GPA<sub>2</sub> → SPA<sub>2</sub>
    - GPA<sub>4</sub> → SPA<sub>4</sub>
- Directly Assigned to User Process
  - ✓ GVA<sub>1</sub> / GVA<sub>2</sub> / GVA<sub>3</sub>
- Directly Assigned to Guest
  - ✓ GPA<sub>2</sub> / GPA<sub>4</sub>
- ... → SPA<sub>2</sub> cached twice
  - ✓ GVA<sub>2</sub> / A2 → SPA<sub>2</sub>
  - ✓ GPA<sub>2</sub> → SPA<sub>2</sub>

# PASID is TWO ECNs

- **Process Address Space ID ECN**  
(PCI Express Base 3.0)
  - ✓ PASID TLP Prefix
  - ✓ Usage on Untranslated Memory Requests
  - ✓ PASID Capability
- **PASID Translation ECN**  
(Address Translation Services 1.1)
  - ✓ Usage on ATS Requests
  - ✓ Usage on ATS Invalidation Requests
  - ✓ Usage on PRI Requests
  - ✓ Usage on PRG Responses
  - ✓ ATS Invalidation rules

# PASID TLP Prefix

- Permitted on:

- ✓ Untranslated Memory Request
  - Including Untranslated AtomicOP Requests
- ✓ ATS Translation Request
- ✓ ATS Invalidation Request
- ✓ Page Request Interface Request (PRI)
- ✓ Page Request Group Response (PRG)

} Base ECN

} ATS ECN

- PASID does not require ATS support

- ✓ Functions can use only Untranslated Memory Requests
- ✓ Without ATS support, pages must be pinned

- PASID does not require PRI support

- ✓ PRI permits paging and late pinning
  - User Process/Guest OS level
  - or
  - Guest OS/Hypervisor level
- ✓ Without PRI support, pages must be pinned

**Complete  
presentation on  
PASID given by  
IOV Workgroup.**

# Lightweight Notification (LN) Protocol

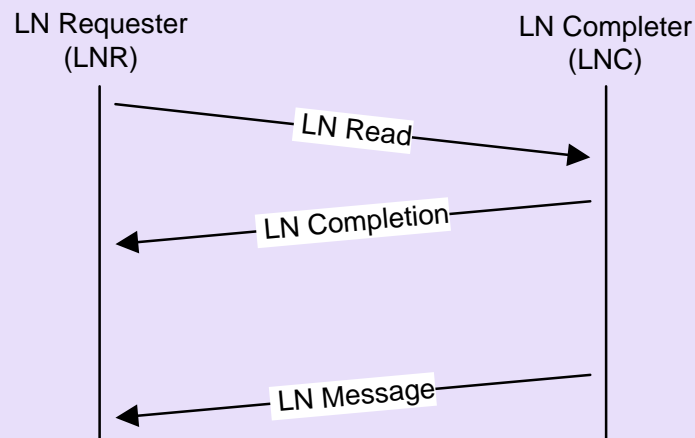
- An optional-normative simple protocol:
  - ✓ A device can *register* one or more cachelines in host memory, and later be notified by a hardware mechanism when any registered cachelines are updated
  - ✓ Architected support for 64-byte & 128-byte cachelines
  - ✓ New LN Requester Capability structure for software to discover and manage LN Requester capabilities in Endpoints
  - ✓ New field in Device Capabilities 2 register to inform software of LN Completer capabilities in the host
- Transactions
  - ✓ LN Reads/Completions/Writes – special forms of Memory Reads/Completions/Writes with registration semantics
  - ✓ LN Messages – SIG-defined Vendor-Specific Messages to convey notifications regarding existing registrations
  - ✓ No changes required for PCIe Switches



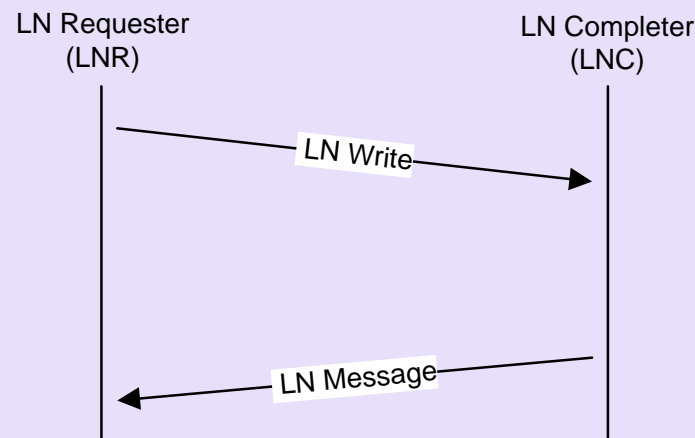
# LN Protocol: Example Benefits

- Reduction of I/O bandwidth consumption & I/O latency:
  - ✓ Device caching can significantly reduce I/O bandwidth consumption and I/O latency for some applications.
  - ✓ Reducing I/O bandwidth consumption also reduces host memory subsystem bandwidth consumption.
- Lightweight signaling:
  - ✓ LN Protocol enables host user-space software to signal a device by updating a cacheline as opposed to performing a PIO operation, which has higher software overhead and synchronization/flow-control issues
- Dynamic device associations:
  - ✓ VM guest drivers communicating with a device via host memory structures enables easier VM guest migration and switching between virtualized and direct I/O for that device.

# LN Protocol: Basic Operation



- LNR reads & registers a cacheline using LN Read
- LNC acknowledges registration & returns data with LN Completion
- Later, LNC notifies LNR with LN Message when cacheline is updated



- LNR writes & registers a cacheline using LN Write
- Later, LNC notifies LNR with LN Message when cacheline is updated

# LN Protocol: Additional Attributes

- Evictions: LN Completer can evict registrations when it runs short on resources.
- LN Message Notification Reasons:
  - ✓ Registered cacheline was updated
  - ✓ Registered cacheline was evicted
  - ✓ All registered cachelines for this LNR were evicted
- Zero-length LN Reads for probing without registering
  - ✓ LN-capable host not required to support LN for all memory regions
  - ✓ LN-capable hosts are required to support LN with 4KB granularity
- Zero-length LN Writes for explicit deregistration
  - ✓ Enables LN Requester to manage its outstanding registrations
- LN Requester Capability Structure provides:
  - ✓ Advertisement of maximum outstanding registrations
  - ✓ Software specified limit for maximum outstanding registrations

# Class Code & Capability ID Extraction ECN

# Class Code & Capability ID Extraction ECN

- An ECN against the *PCI Local Bus Specification*, Rev 3.0
- Extracts the Class Code definitions from Appendix D
- Extracts the Capability ID definitions from Appendix H
- Enables the consolidation of these definitions into a new standalone document that's easier to maintain
- The new document is the *PCI Code and ID Assignment Specification*.
- Consolidating these and other definitions makes them easier to find and manage, and reduces the chance of lost or duplicate assignments.
- New IDs for specifications/ECNs, or new Class Codes will trigger updates to the new specification.

# PCI Code and ID Assignment Specification

# PCI Code and ID Assignment Specification

- Consolidates:
  - ✓ Class Code definitions from the *PCI Local Bus Specification* Appendix D
  - ✓ Capability ID definitions from the *PCI Local Bus Specification* Appendix H
  - ✓ Extended Capability ID definitions from the *PCI Express Base & I/O Virtualization* specifications
- Includes:
  - ✓ New Class Code and Capability ID assignments made since the *PCI Local Bus Specification*, Revision 3.0
  - ✓ Some cleanup of formatting & terminology

# PCI Code & ID Assignment Specification Update ECN



# PCI Code & ID Assignment Specification Update ECN

- A very simple ECN against the new *PCI Code & ID Assignment Specification*
  - ✓ Intentionally chose to separate these “new changes” from the initial version so they could receive proper visibility & review
  - ✓ Wanted the initial version not to include any semantic changes
- Adds a new Programming Interface assignment to Base Class 01h (Storage Controllers) / Sub-Class 08h (Flash Controllers)
  - ✓ 02h – Solid State Storage Controller – Enterprise NVMHCI
- Changes “Flash Controller” references in existing Class Code definitions to “Solid State Storage Controller”
- Adds new “Null Capability” ID for both standard Capabilities and Extended Capabilities
  - ✓ Intended for use by hypervisors that intentionally choose not to expose selected Capabilities in the virtualized Configuration Space for VMs
- Assigns Extended Capability ID 001Ah for Protocol Multiplexing (PMUX)

Thank you for attending the  
PCI-SIG Developers Conference 2011.

For more information please go to  
[www.pcisig.com](http://www.pcisig.com)