PCIe® 2.0 Errata & Protocol Extensions

Mahesh Wagh / Joe Cowan

Intel Corporation / Hewlett-Packard Company

Objective

- PCIe® protocol working group has resolved PCIe 2.0 errata
  - ✓ Provide update on PCIe 2.0 errata, focus on key critical impact errata items
- PCIe protocol working group is actively developing protocol extension ECRs to PCIe 2.0 specification
  - ✓ ECRs available for PCI SIG membership review in Q1-08
  - ✓ Plan is to roll ECRs in to PCIe 3.0 specification
- Objective of this training material is to
  - ✓ Provide an overview of protocol extensions currently under development
  - ✓ Obtain early feedback

PCIe 2.0 Errata

PCIe 2.0 Errata

- Spec errors, clarifications, typos, terminology and formatting grouped in four bins
  - ✓ Bin A: “non-straight-forward changes” – 22 approved as of 2/12
  - ✓ Bin B: “straight-forward changes” – 57 approved as of 2/12
  - ✓ Bin C: “trivial editorial changes” – 23 approved as of 2/12
  - ✓ Bin D: “unclassified” – 11 approved as of 2/12
- As of 2/12, 113 are approved & ~14 errata still being worked
- Formal errata release to PCI-SIG membership planned for end Q1 – mid Q2’08

PCIe 2.0 Errata Highlights

- Ordering Rules Summary Table and Descriptions overhauled to create a more suitable base for AtomicOps & ID-Based Ordering ECNs
  - ✓ Errata **do not** change ordering semantics, but subsequent ECNs **do**
  - ✓ More details in following slides…
- Several misc clarifications on 5.0 GT/s vs 2.5 GT/s operation
- Several misc clarifications on de-emphasis operation
- Several misc clarifications on cross-link operation
- Various fields: Reserved vs “required to be 0”
  - ✓ Generally prefer to keep cases Reserved to avoid squandering bits
  - ✓ A few cases left “required to be 0” in case some Receivers implemented checks, even though such checks were never intended
- Link Capability bits having same value in all Functions of MFD *associated with an Upstream Port*
  - ✓ Link Cap bits in Upstream Port Functions of MFD all refer to same Link
  - ✓ Link Cap bits in Downstream Port Functions of MFD refer to different Links
- ACS Violation bits in AER are optional normative
  - ✓ Not required to be implemented in AER unless ACS is implemented
- Earlier VC Enable bit clarifications for VC Cap struct carried over to same bit in MFVC Cap struct

Ordering Rules Summary Table – Key Problems

- Not all components can distinguish Read Completions from I/O & Configuration Write Completions
- Some row/column labels enumerate individual transactions, causing maintenance problems as new transactions are defined, e.g., AtomicOps
- Associated descriptions are excessively wordy and inconsistent in style

Row Pass Column?		Posted Request	Non-Posted Request		Completion	
		Memory Write or Message Posted Request (Col 2)	Read Request (Col 3)	I/O or Configuration Write Request NPR with Data (Col 4)	Read Completion (Col 5)	I/O or Configuration Write Completion (Col 6)
Posted Request	Memory Write or Message Posted Request (Row A)	a) No b) Y/N	Yes	Yes	a) Y/N b) Yes	a) Y/N b) Yes
Non-Posted Request	Read Request (Row B)	No	Y/N	Y/N	Y/N	Y/N
	I/O or Configuration Write Request NPR with Data (Row C)	No	Y/N	Y/N	Y/N	Y/N
Completion	Read Completion (Row D)	a) No b) Y/N	Yes	Yes	a) Y/N b) No	Y/N
	I/O or Configuration Write Completion (Row E)	Y/N	Yes	Yes	Y/N	Y/N

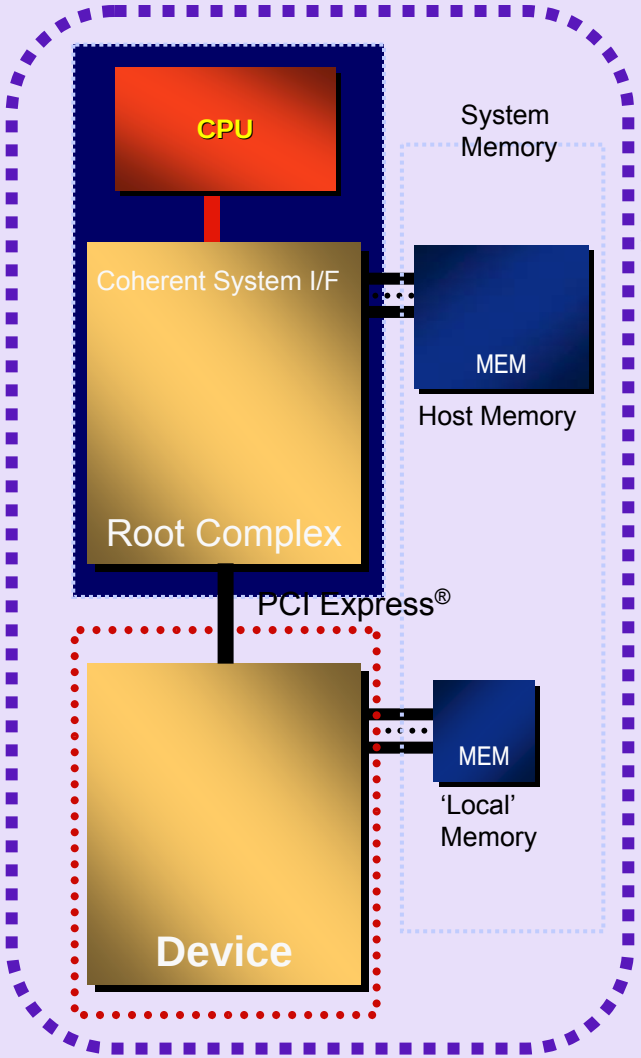
Ordering Rules Summary Table – End Result

Row Pass Column?		Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
			Read Request (Col 3)	NPR with Data (Col 4)	
Posted Request (Row A)		a) No b) Y/N	Yes	Yes	a) Y/N b) Yes
Non-Posted Request	**Read Request** (Row B)	No	Y/N	Y/N	Y/N
	NPR with Data (Row C)	No	Y/N	Y/N	Y/N
Completion (Row D)		a) No b) Y/N	Yes	Yes	a) Y/N b) No

- No longer separate rows/columns for Read Completions vs Config/IO Write Completions
- Row/column labels now consistently based on classes of transactions instead of enumerated individual transactions
- Associated descriptions now more consistently and concisely worded

PCIe Protocol Extensions

PCIe Protocol Extensions



Performance Improvements

- ✓ Opaque Processing Hints – Transaction processing hints to optimize system resources
- ✓ Ordering – Transaction level attribute/hint to optimize ordering within RC and memory subsystem

Software Model Improvements

- ✓ Atomic Operations – Mechanisms to reduce synchronization overhead
- ✓ IO Page Faults– Mechanism to notify device of page faults and request for faulted pages to be made available

Communication Model Enhancements

- ✓ Multicast – Mechanism to transfer common data or command set from one source to multiple recipients

Power Management

- ✓ Dynamic Power Allocation - Support for dynamic power operational modes through standard configuration mechanism
- ✓ Latency Tolerance Reporting
- ✓ Opportunistic Buffer Flush & Fill

Configuration Enhancements

- ✓ Resizable BAR– Mechanism for device to re-negotiate allocated BAR sizes
- ✓ Internal Error Reporting– Extend the AER to report component internal correctable/uncorrectable errors and report multiple errors

Protocol Extensions Summary

Extension	Description	Status
Opaque Processing Hints (OPH)	Request hints to enable optimized processing within host memory/cache hierarchy	Under 30-day Member Review
ID-Based Ordering	New ordering semantics to improve performance	Under 2-week BoD/WG Review
Atomic Operations	Atomic Read-Modify-Write mechanism	Under 30-day Member Review
I/O Page Faults (ECR to ATS Spec)	Extends IO address remapping for page faults	Under development by IOV-WG
Resizable BAR	Mechanism to negotiate BAR size	Under 30-day Member Review
Dynamic Power Allocation (DPA)	Mechanisms to allow dynamic power/performance management for substates of D0 (active state).	Under 30-day Member Review
Multicast	Address-Based Multicast of Posted Request TLPs	Under 30-day Member Review
Internal Error Reporting	Extend AER to report component internal errors (correctable & uncorrectable) and multiple error logs	Under 30-day Member Review
LTR & OBFF	Mechanisms for platform to tune PM and to align device activities	Work-in-Progress
TLP Prefix	Mechanism to extend TLP headers	Work-in-Progress

ID-Based Ordering (IDO)

Review: PCIe Ordering Rules

"No" entries caused by Producer/ Consumer restrictions

Row Pass Column?		Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
			Read Request (Col 3)	NPR with Data (Col 4)	
Posted Request (Row A)		a) No b) Y/N	Yes	Yes	a) Y/N b) Yes
Non-Posted Request	Read Request (Row B)	No	Y/N	Y/N	Y/N
	NPR with Data (Row C)	No	Y/N	Y/N	Y/N
Completion (Row D)		a) No b) Y/N	Yes	Yes	a) Y/N b) No

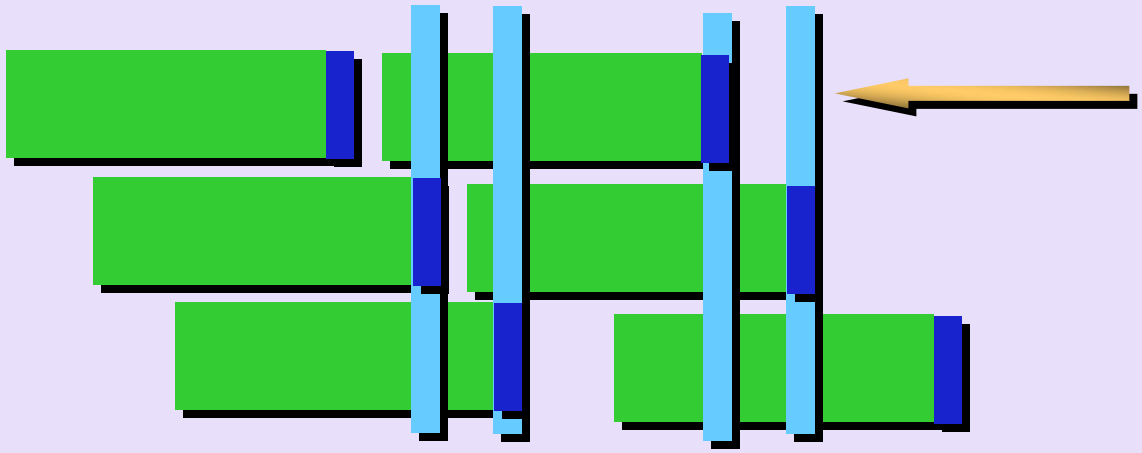
Table is based on new 2.0 errata!

"Yes" entries are required for deadlock avoidance

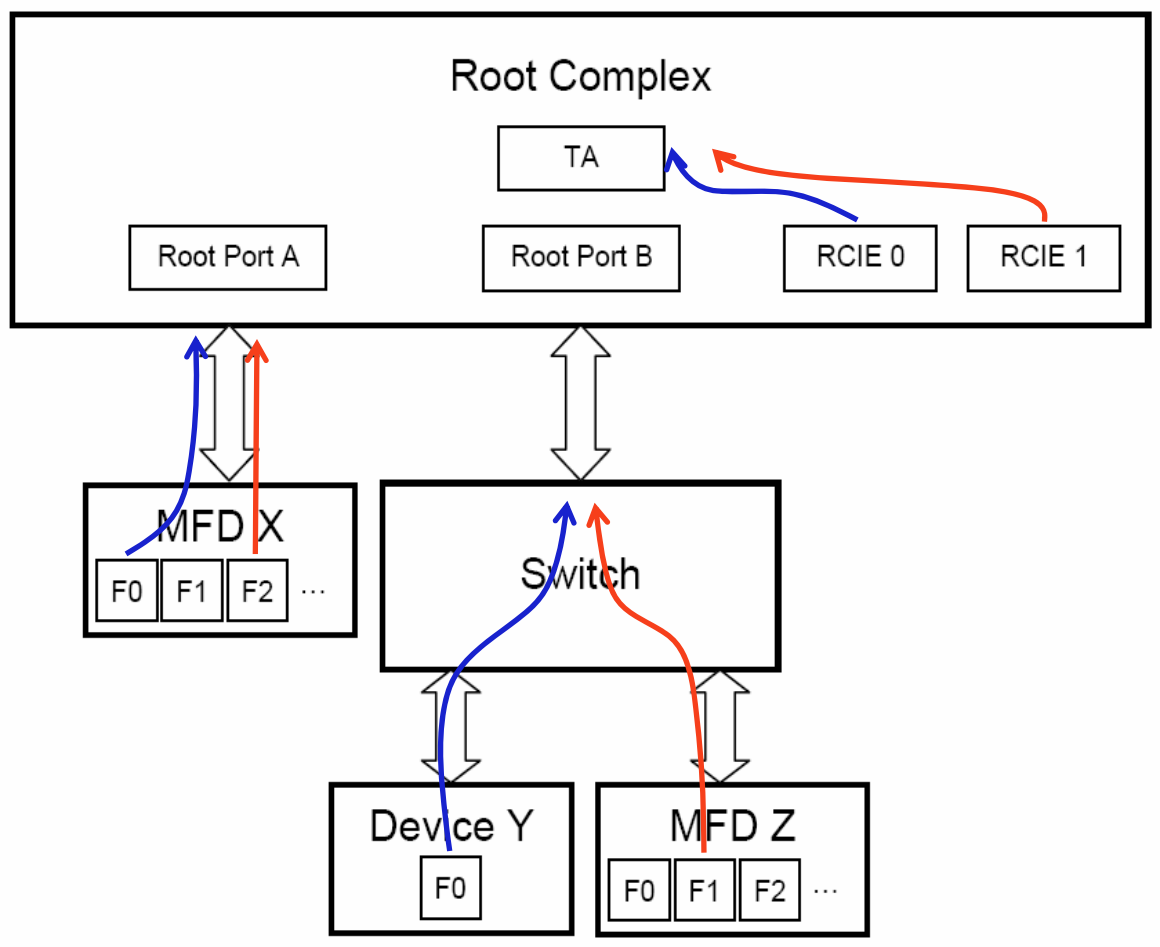
- Maximum theoretical flexibility: All entries are "Y/N"
- Traditional Relaxed Ordering (RO) enables A2 & D2 "Y/N" cases
  - ✓ AtomicOps ECN defines an RO-enabled C2 "Y/N" case
- ID-Based Ordering (IDO) enables A2, B2, C2, & D2 "Y/N" cases

Motivation

- RO works well for single-stream models where a data buffer is written once, consumed, and then recycled
  - ✓ Not OK for buffers that will be written more than once because writes are not guaranteed to complete in order issued
  - ✓ Does not take advantage of the fact that ordering doesn't need to be enforced between unrelated streams
- Conventional Ordering (CO) can cause significant stalls
  - ✓ Observed stalls in the 10's to 100's of ns are seen
  - ✓ Worst case behavior may see such stalls repeatedly for a request stream
- Consider case of NIC or disk controller with multiple streams of writes:



IDO: Perf Optimizations for Unrelated TLP Streams



- TLP Stream: a set of TLPs that all have the same originator
- Optimizations possible for unrelated TLP Streams, notably with:
  - ✓ Multi-Function device (MFD) / Root Port Direct Connect
  - ✓ Switched Environments
  - ✓ Multiple RC Integrated Endpoints (RCIEs)
- IDO permits passing between TLPs in different streams
- Particularly beneficial when a Translation Agent (TA) stalls TLP streams temporarily

IDO Usage

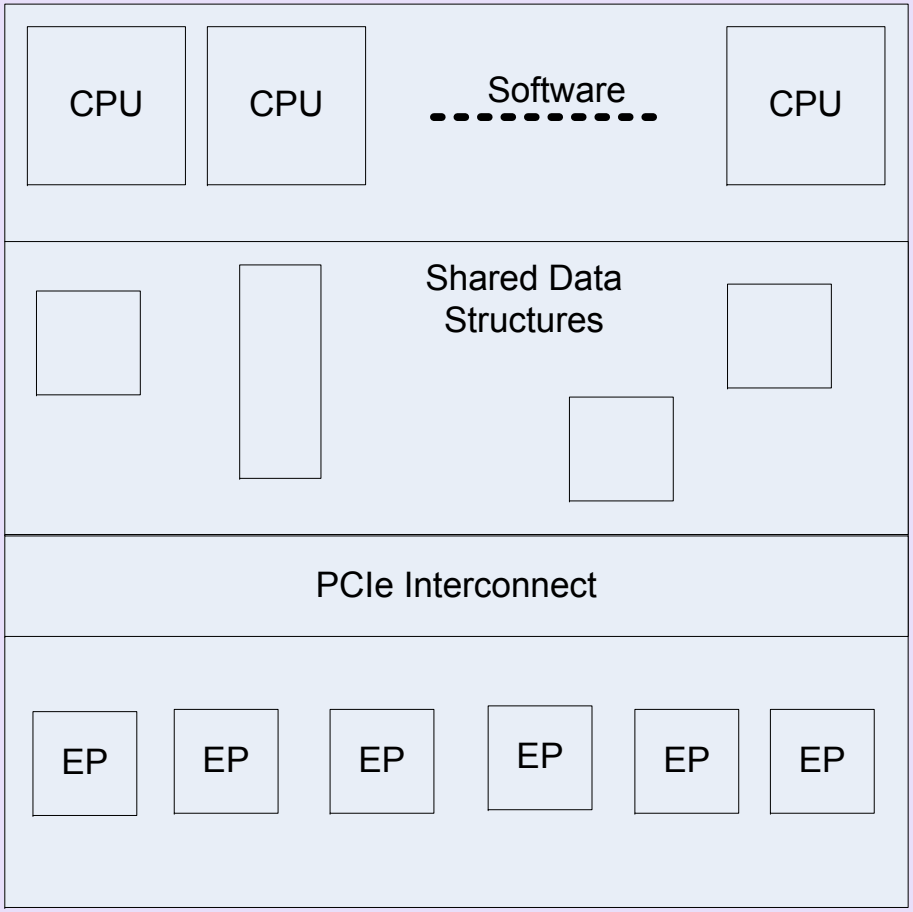
- IDO Use in Conjunction with RO
  - ✓ IDO and RO are orthogonal, though overlapping in some cases
  - ✓ Highly recommended to use both whenever safely possible
  - ✓ RO permits certain TLP passing within same TLP stream, which is never permitted by IDO
  - ✓ For traffic in different TLP streams, IDO permits control traffic to pass any other traffic, and generally it is not safe to Set RO with control traffic
- IDO safe & simple for Endpoints (EPs) to use with 2-party communication
  - ✓ E.g., an EP communicating directly only with the host
  - ✓ "Simple policy" for EP is to set IDO for all applicable TLPs it originates
- IDO is not simple to use with >2-party communication
  - ✓ E.g., 2 EPs doing peer-to-peer & also communicating via host memory
  - ✓ Generally it is not safe for EP to set IDO for all applicable TLPs
  - ✓ Appendix E documents example communication failure case
- No simple high-benefit use cases identified for RPs setting IDO in TLPs they originate
  - ✓ This is not prohibited, but is outside the scope of the spec

Software Control of IDO Use

- Separate IDO Enable bits for Requests and Completions
- By default, EPs and RPs are not enabled to Set IDO
- Envisioned that software drivers for EPs will enable IDO use
  - ✓ Driver should determine if it is safe to enable IDO use; e.g., if EP does >2-party communication
  - ✓ Driver should know the EP's policy for setting IDO when enabled; e.g., if the EP has advanced policies for >2-party communication
- Envisioned that RPs will not be enabled to Set IDO on most platforms
  - ✓ Enabling RPs to Set IDO is not prohibited, but is outside the scope of the spec

Atomic Operations (AtomicOps)

Motivation



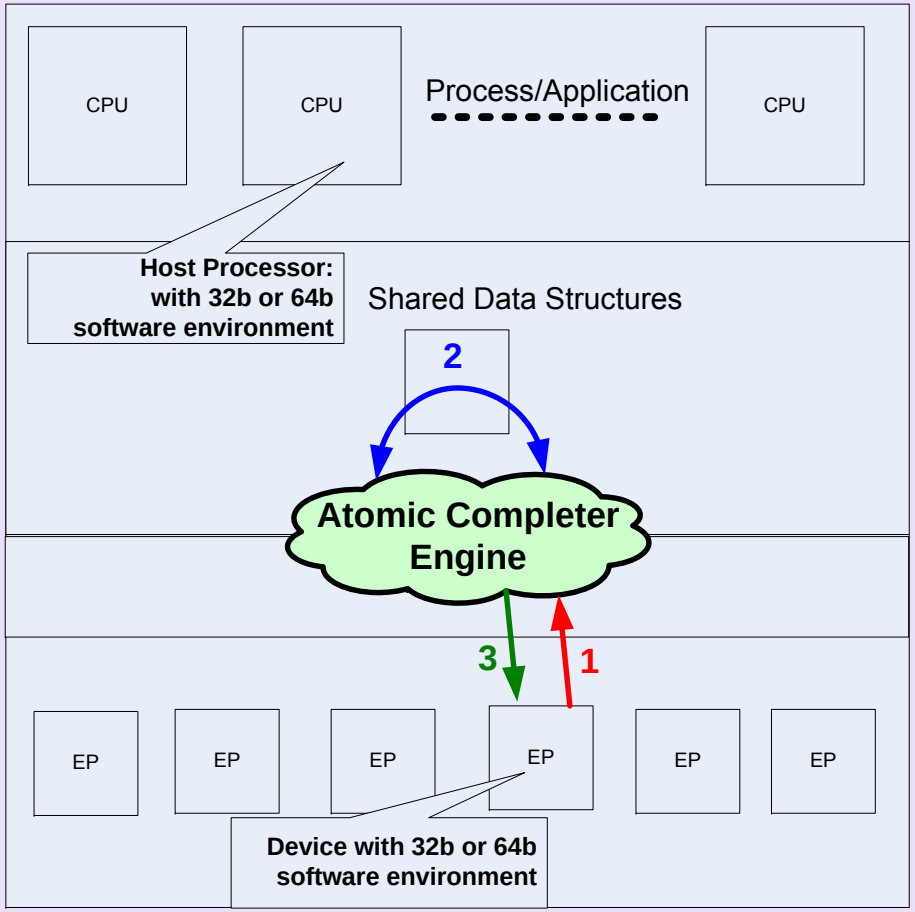
- Atomic transaction support for Host update of main memory exists today
  - ✓ Useful for synchronization without interrupts
  - ✓ Rich library of proven algorithms in this area
- Benefit in extending existing inter-processor primitives for data sharing/synchronization to PCIe interconnect domain
  - ✓ Low overhead critical sections
  - ✓ Non-Blocking algorithms for managing data structures e.g. Task lists
  - ✓ Lock-Free Statistics e.g. counter updates
- Existing application performance can generally be improved by providing synchronization enhancements
  - ✓ Faster packet arrival rates create demand for faster synchronization
- Emerging applications require PCIe synchronization enhancements to support multiple producer – multiple consumer co-ordination
  - ✓ Math, Visualization, Content Processing etc

Atomic Operations

AtomicOp	Description
FetchAdd	Data(Addr) = Data(Addr) + AddData
Swap	Data(Addr) = SwapData
CAS (Compare & Swap)	If (CompareData == Data(Addr)) then Data(Addr) = SwapData

- Each AtomicOp returns initial Data(Addr)
- Operation sizes supported
  - ✓ 32b and 64b operation sizes for FetchAdd and Swap
  - ✓ 32b, 64b, and 128b operation sizes for CAS
- AtomicOp Request address must be naturally aligned to operation size
  - ✓ AtomicOp data access guaranteed to not cross cacheline boundaries

Atomic Operation Mechanism



- AtomicOp Request
  - ✓ Non-Posted Request with Data
  - ✓ FetchAdd, Swap, or CAS
  - ✓ Multiple outstanding AtomicOps supported
  - ✓ Requestor has to allocate space for Completion before issuing each AtomicOp Request
- Atomic Operation Execution
  - ✓ Atomic Completer Engine performs atomic read-modify-write operation
    - Read initial value
    - Compute and write new value
- AtomicOp Completion
  - ✓ Completion returns initial value
  - ✓ "Standard" Completion for Non-Posted Request; e.g. routed by ID, TID links the Completion with its associated Request

*Device to System memory shown in figure for illustration purposes

Atomic Operations Summary

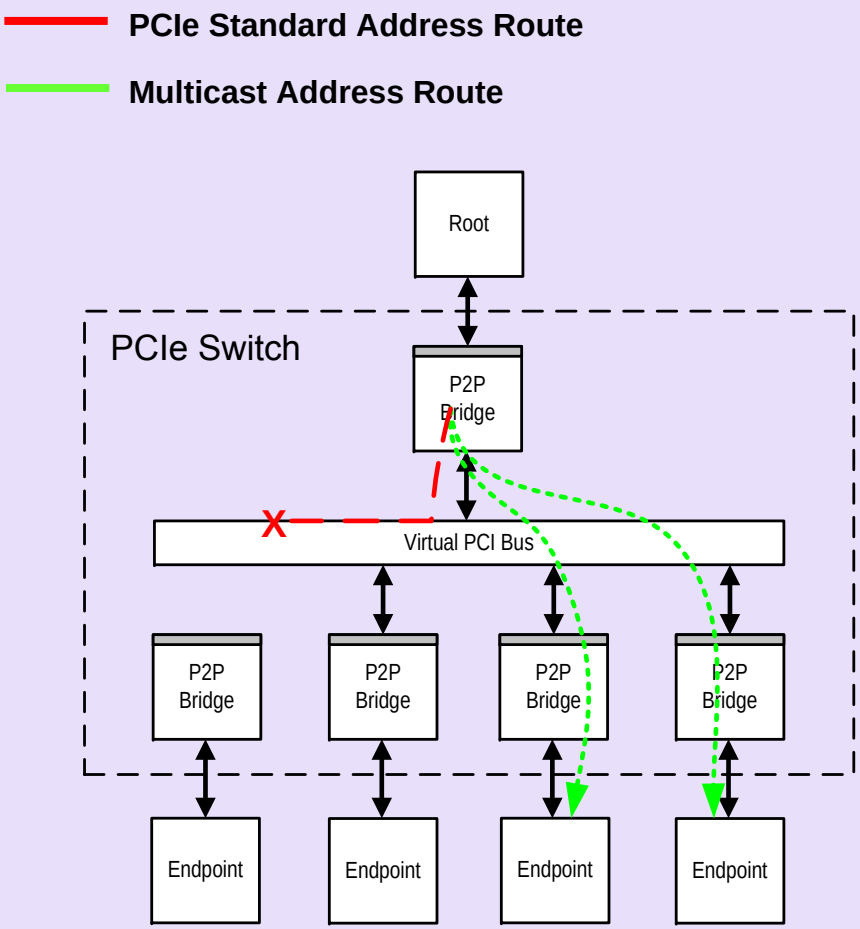
- Atomic Operations provide capability symmetric to and consistent with capabilities supported by most host CPU architectures
  - ✓ Support core synchronization mechanisms (e.g. semaphores) for CPU/device (as for CPU/CPU), without use of bus-lock type mechanism and minimal use of critical sections
- Enable reuse of existing/debugged algorithms for critical sections, data sharing and queuing implementations
  - ✓ Device side re-use of algorithms already proven on hosts
- Atomic Operations supported: FetchAdd, Swap, and CAS
  - ✓ FetchAdd: Lock-Free Statistics
  - ✓ Swap: Sample & Reset
  - ✓ CAS: Essential for implementation of critical sections and non-blocking queuing algorithms
- Relaxed Ordering (RO) Attribute is permitted with AtomicOp Requests
  - ✓ AtomicOp Request with RO Set is permitted to pass Posted Requests
  - ✓ AtomicOp Completion with RO Set is permitted to pass Posted Requests
- Software discovery & control mechanisms (Device Capability 2 & Control 2 regs)
  - ✓ AtomicOp Routing Supported: necessary since current routing elements do not support
  - ✓ Various AtomicOp Completer bits: 3 bits indicate various type/size combos supported
  - ✓ AtomicOp Requester Enable: prevents AtomicOp use without enablement by software
  - ✓ AtomicOp Egress Blocking: prevents forwarding AtomicOp Requests to components that shouldn't receive them; e.g., those that would handle them as Malformed TLPs

Multicast

Motivation

- Many applications that benefit from Multicast
  - ✓ Communications (e.g. route table updates, support of IP Multicast)
  - ✓ Storage (e.g., mirroring, RAID)
  - ✓ Multi-headed graphics
  - ✓ PCIe used in place of backplane Ethernet
- PCIe architecture extended to support address-based Multicast
  - ✓ New Multicast BAR to define Multicast address space
  - ✓ New Multicast Capability structure to configure RCRB/Switches and Endpoints for Multicast address decode and routing
- Supports only Posted, address-routed transactions (e.g., Memory Writes)
  - ✓ Supports both RCs and EPs as both targets and initiators
  - ✓ Compatible with systems employing Address Translation Services (ATS) and Access Control Services (ACS)
  - ✓ Multicast capability permitted at any point in a PCIe hierarchy

Example 1



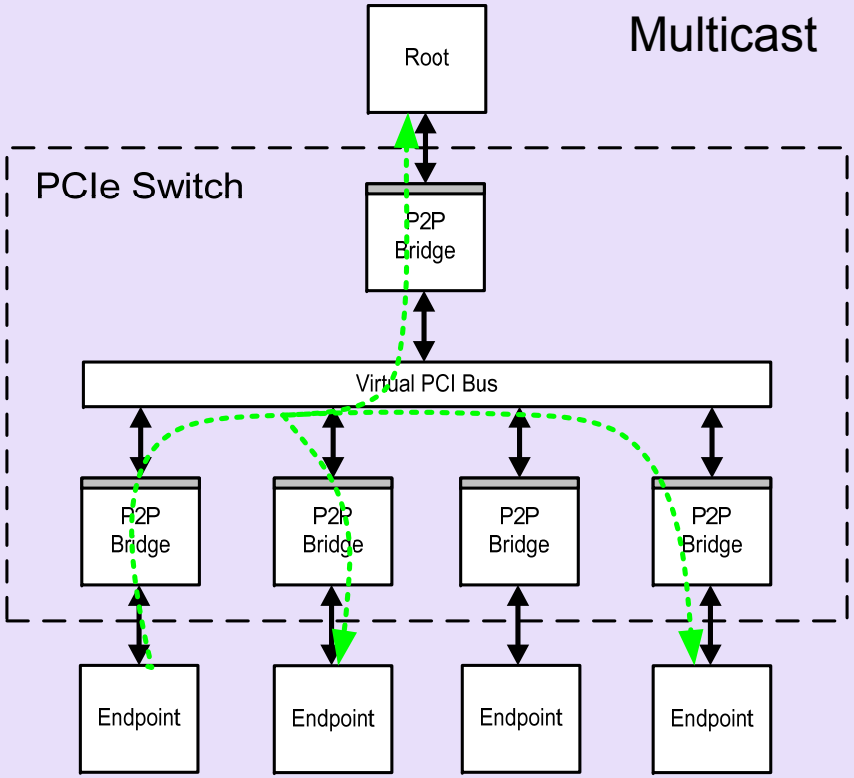
- Request that hits a Multicast address range, is routed unchanged to Ports that are part of the Multicast Group derived from Request address
- PCIe standard address route not used for multicast

Example 2

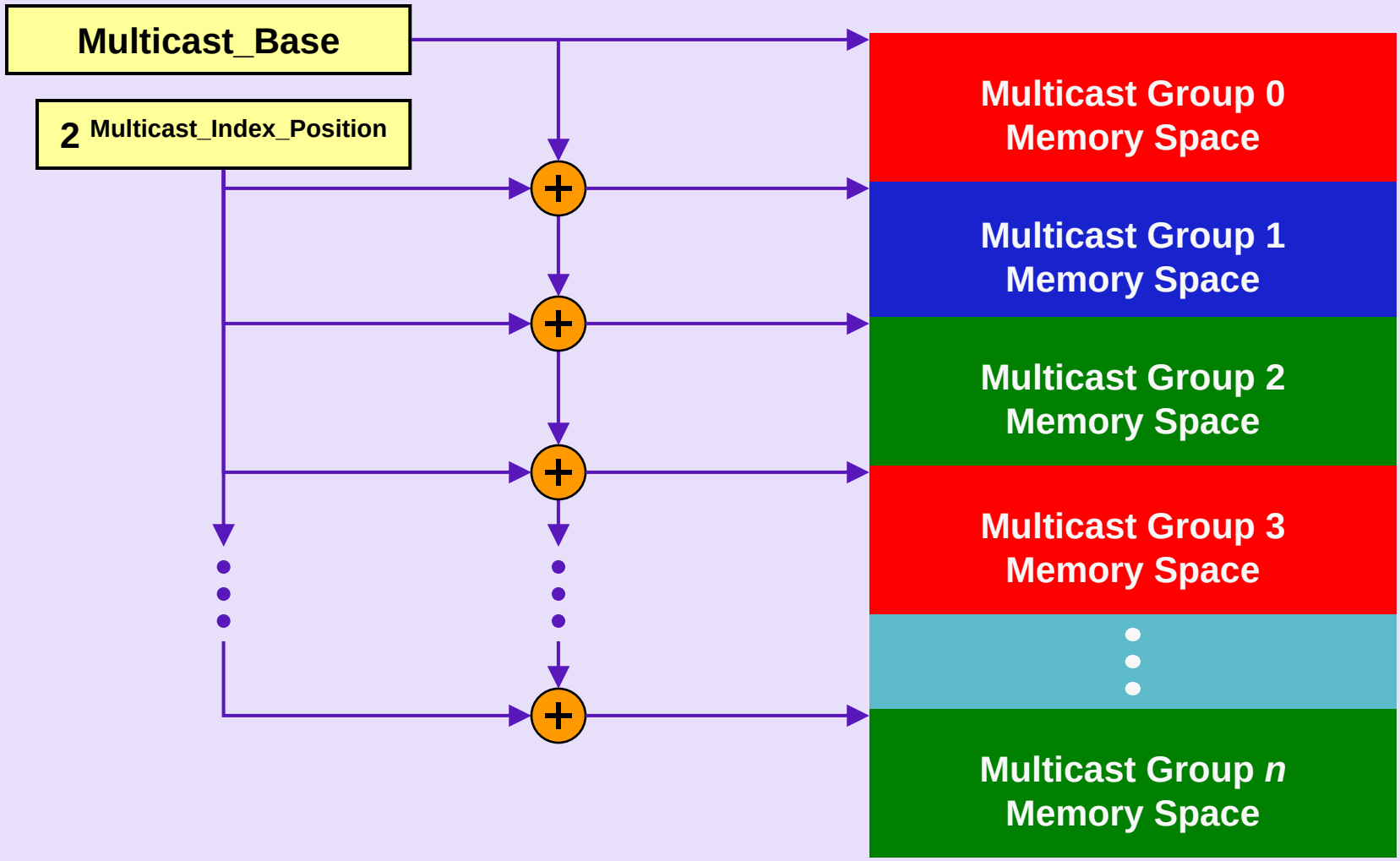
PCIe Standard Address Route

Multicast Address Route

- Address route Upstream– Upstream Port must be part of the forwarding Ports for Multicast



Multicast Memory Space



Multicast Processing

- A TLP has a Multicast Hit if:
  - ✓ MC_Enable is Set, and
  - ✓ The TLP is a Posted Request, and
  - ✓ Address targets a Multicast address range
    - Extract the Multicast Group (MCG) number from the target address
- Perform MC Blocked TLP checking based on the MCG number:
  - ✓ If the Multicast TLP came in an Ingress Port containing a Multicast Capability structure, block the Multicast TLP if:
    - The associated MC_Block_All bit is Set, or
    - The associated MC_Block_Untranslated bit is Set and the address is Untranslated
  - ✓ If the Multicast TLP is being sent by an Endpoint Function containing a Multicast Capability structure, block the Multicast TLP if:
    - The associated MC_Block_All bit is Set, or
    - The associated MC_Block_Untranslated bit is Set and the address is Untranslated
- Perform the Multicast based on the MCG number:
  - ✓ In Switches or Endpoint components, forward the TLP to each Upstream/Downstream Port or Endpoint Function whose associated MC_Receive bit is Set
  - ✓ In RCs, forward the TLP to each Root Port or RC Integrated Endpoint whose associated MC_Receive bit is Set
  - ✓ Do not forward the TLP back to the source Port/Endpoint

Resizable BAR Capability

Motivation

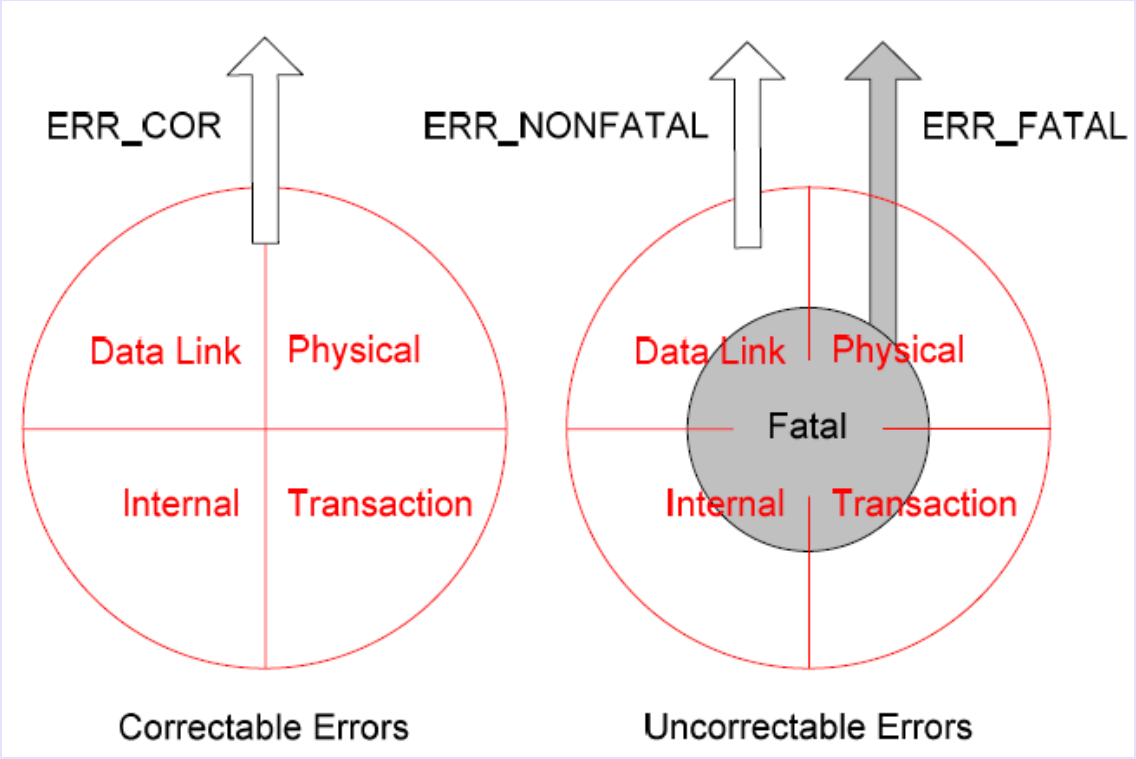
- BAR Space Allocation
  - ✓ Local memory is becoming quite large on some add-in cards
  - ✓ More frequently, all of the requested Memory Space resources are not being allocated for all of the add-in cards
    - Resource is not allocated or Function is forced to report smaller aperture size in order to be allocated
  - ✓ 256 MB seems to be a common limit in 32-bit systems
  - ✓ Current solutions are proprietary and not exposed to all software
- New mechanism allows requested Memory Space resources to be sized appropriately for the system it is in, so that all of the resources can be allocated
- Mechanisms are invoked during enumeration before BAR space allocation

Resizable BAR Capability

- New Resizable BAR Capability structure
  - ✓ Endpoint Functions advertise multiple supported sizes for device Memory Space resources
  - ✓ Controls enable software to select the device's requested BAR sizes
- Hardware utilizes this new capability to report multiple supported sizes for device BARs to resource allocation software
- Resource allocation software will attempt to program the largest acceptable size for each device BAR
  - ✓ System can be configured with optimal resource settings
- Backwards Compatible
  - ✓ Current resource allocation must work for software that does not comprehend this new capability

Internal Error Reporting

Motivation



Error Classification

- New Internal Error classification to report internal errors associated with PCIe interface

Definitions

- Internal Error
  - ✓ An error that occurs within a component and that is not attributable to a packet or event on the PCI Express interface
  - ✓ What is reported as an internal error is implementation specific
- Correctable Internal Error
  - ✓ An internal error that hardware has masked or worked around without any loss of information or improper operation.
  - ✓ Example:
    - An ECC corrected single bit error in an internal packet buffer
- Uncorrectable Internal Error
  - ✓ An internal error that has resulted in improper operation.
  - ✓ Example:
    - A transient memory fault that results in a double bit error in an internal packet buffer that stores the header of a non-ECRC protected TLP.

Internal Error Logging

- Uncorrectable Internal Error
  - ✓ Add Uncorrectable Internal Error bit to the following
    - AER Uncorrectable Error Status Register
    - AER Uncorrectable Error Mask Register
    - AER Uncorrectable Error Severity Register
  - ✓ Optionally log header
- Correctable Internal Errors
  - ✓ Add Correctable Internal Error bit to the following
    - AER Correctable Error Status Register
    - AER Correctable Error Mask Register
  - ✓ Header not logged

Internal Error Reporting

- Switches
  - ✓ New mechanism for PCIe switches to report internal error that can be used by OS/Hypervisor
- Roots
  - ✓ Roots behave like switches in the context of peer-to-peer traffic
  - ✓ An OS/Hypervisor can use the same reporting mechanism and may take the same action when an error can be isolated to a specific Root Port as it does when an error is detected in a Switch
- Endpoints
  - ✓ Use the same mechanism defined for Switches and Roots to report internal errors in Endpoints associated with the PCIe interface
  - ✓ This mechanism is not intended to be used by Endpoints to report errors not associated with the PCIe interface (i.e., application logic). These errors should continue to be handled through proprietary methods employing device specific interrupts.

Multiple Error Handling

- Extend Advanced Error Reporting (AER) capability to allow for multiple error headers to be recorded
  - ✓ New Multiple header log capable and enable bits
  - ✓ Improved error containment and recovery
- Extend AER to define Header Log Overflow indication
  - ✓ New Error overflow bit
  - ✓ Error that requires header log is detected but cannot be logged due to header log resource constraints or Multiple Header Log Enable bit not set and First Error Pointer is valid
- When Multiple Header Log capability is supported and enabled
  - ✓ Error headers are logged in the corresponding order the errors were detected
  - ✓ When the Uncorrectable Status register for which the First Error Pointer corresponds to is cleared the First Error Pointer and the Header Log are updated to point to the next recorded header
  - ✓ When the last recorded header status bit is cleared then the First Error Pointer becomes invalid and value of Header Log is undefined

Opaque Processing Hints (OPH)

OPH Motivation

- Today's PCIe requests are coherent with respect to system memory/caches
  - ✓ Root Complex maintains coherency via "snoop"
- Current communication between device and host does not take full advantage of system capabilities
  - ✓ Use of system cache hierarchy can help reduce access latency
  - ✓ Effective use of system resources (system interconnect and memory)
- Transaction processing hints provide an opportunity to optimize use of system fabric and improve system efficiency
  - ✓ Facilitate Data Residency/Allocation within system cache hierarchy
  - ✓ Minimize memory access latencies
  - ✓ Reduce memory & System interconnect Bandwidth & associated Power consumption

OPH - Usage Models

Data Structures of interest

- Control Data i.e. Descriptors
- Headers for protocol processing
- Payload for data copies

Usage Models

- Efficient processing of network IO in systems with variable access latencies
  - ✓ IO is increasingly becoming faster (E.g. 10G → 40G → 100G …)
- Communications adapters in HPC clusters, Database System Clusters & Computational Accelerators
  - ✓ Multi-node barriers and collective operations are often a key performance limit.
  - ✓ Particularly exchange of lock information; often key barrier to scaling
  - ✓ Delay / overhead per operation limits efficiency in some operations (not all), reducing range of applicability.

OPH - Mechanism

- Mechanism to provide processing hints on per transaction basis for requests that target memory space
  - ✓ Enable system hardware (ex: Root-Complex) to optimize on a per transaction basis
  - ✓ Applicable to Memory Read/Write and Atomic Operations
- OPH mechanism utilizes
  - ✓ 10 bits of existing header fields
  - ✓ 24 bits of New TLP Prefix header for scalability (optional)
- Transaction Processing hints are opaque and implementation specific; used to provide hints for transaction processing
  - ✓ Temporal Re-Use
  - ✓ Enables association of host processing elements with processing of requests from specific devices
  - ✓ Location/Level of a targeted cache (host processor core identifier)
  - ✓ Allocation hints
  - ✓ Not limited to examples provided above

OPH Summary

- Mechanism to make effective use of system fabric and improve system efficiency
  - ✓ Reduce variability in access to system memory
  - ✓ Reduce Memory & System Interconnect BW & Associated Power Consumption
- Ecosystem Impact
  - ✓ Software impact under investigation - minimally may require software support to retrieve hints from system hardware
  - ✓ Endpoints take advantage only as needed → No cost if not used
  - ✓ Root Complex allowed implementation tradeoffs
  - ✓ Minimal impact to Switches
- Architected software discovery, identification and control of capabilities
  - ✓ RC support for processing hints
  - ✓ Endpoint enabling to issue hints

IO Page Faults (ECR to ATS Specification being developed by IOV-WG)

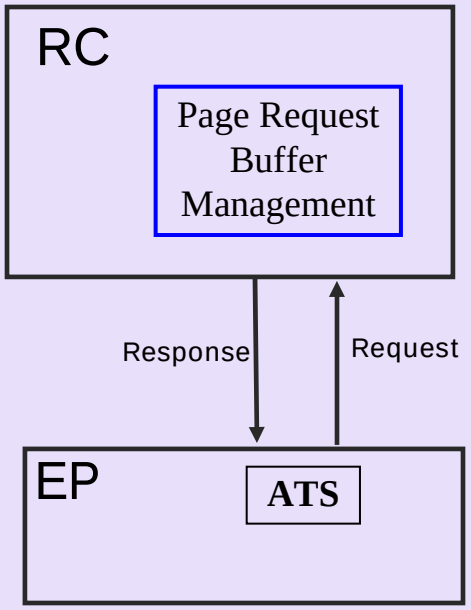
Motivation

- RC Virtualization encourages use of Guest Virtual Addresses by I/O devices
  - ✓ Virtualization increases pressure on memory resources making pinning increasingly expensive
  - ✓ Page fault not desired outcome, but preventing page faults from occurring may be more difficult than building model for supporting the rare occurrence of a page fault
- Acceleration model benefit from user-level (process) virtual address based I/O communication
  - ✓ Maintaining synchronization between “known” addresses in co-processing systems and resident pages of process memory is problematic if pinning is only available model
  - ✓ Again, memory becoming scarce commodity in virtualized systems and minimization of static process footprint is important
  - ✓ Elimination of overhead due to kernel transitions to pin memory
- Hypervisor resource over-commitment important to maintaining overall system performance
  - ✓ Balloon allocators grow and shrink guest physical memory allocations
  - ✓ Swapping of System Image (SI) to disk possible, but without page faulting, only pages not potentially used by I/O can be swapped
    - – Difficult to know for fully virtualized guests.
    - – Become more acute with IOV direct device attachment
  - ✓ Page faults result in better ability to over-commit memory
  - ✓ Yet another instance of desire to minimize static memory footprint in favor of more dynamic footprint

IO Page Fault Mechanism

Basic IO Page Fault Mechanism as follows

- Recognize
  - ✓ Is a valid translation for a page available.
  - ✓ Check IOTLB.
  - ✓ Request translation via Address Translation Services (ATS).
  - ✓ If translation request fails – Request.
- Request
  - ✓ Use the New Page Request interface to load page to RC
- Restart
  - ✓ When page has been loaded, RC notifies I/O device.
  - ✓ I/O device reinitiates 3 REs (Recognize – Request – Restart)



Page Request Interface

- Optional Normative mechanism enables a device to request residence of a specific page or set of pages into the associated system's memory
  - ✓ Devices required to employ a Translation Agent (TA) and/or Address Translation Cache (ATC) and/or Address Translation Services (ATS)
- Page Request Interface is applicable to RC and components that implement Endpoint functions
  - ✓ Completely transparent to routing elements
- Page Request is a PCIe message request initiated by endpoint functions
  - ✓ Requesting device provides address and tag
  - ✓ Allows for group of page requests into tag groups
- After the Page Request is serviced RC sends a tagged Response message back to endpoint function
  - ✓ Notify requesting function about page request load success or resource oversubscription/buffer failure conditions
- Page request interface mechanism is orthogonal to system virtualization interface
  - ✓ RC Page request buffer management implementation tradeoffs allowed

Dynamic Power Allocation (DPA)

Motivation

- Current PCIe provides standard Device & Link-level Power Management
- PCIe 2.0 adds mechanisms for dynamic scaling of Link width/speed
- Device increasingly higher consumers of system power & thermal budget (e.g. gfx)
- No architected mechanism for dynamic control of device thermal/power budgets
- New PCIe power management mechanisms to complement support on-going industry wide efforts to optimize component and platform power management to meet new customer and regulatory operating requirements

DPA

- Extend existing PCIe device PM to provide active (D0) device power management sub-states
  - ✓ Support for up to 32 sub-states per function
  - ✓ Maximum power allocation advertised per sub-state in Watts
  - ✓ Contiguously numbered sub-states 0 to N substate w/o gaps
  - ✓ Each successive sub-state reports maximum power allocation less than or equal to prior sub-state
- Software permitted to dynamically configure a function to operate at any sub-state in any sequence
  - ✓ Function must operate at or below the maximum power allocation corresponding to the selected sub-state
- New DPA capability only for endpoints
  - ✓ DPA capability to enable software to discover and actively manage endpoint function power usage

LTR and OBFF (Developed as separate ECRs) Overview

Reducing Platform Power through Latency Tolerance Reporting

- Problem: Current platform PM policies guesstimate when devices are idle (e.g. w/inactivity timers)
  - ✓ Guessing wrong can cause performance issues, or even HW failures
  - ✓ Worst case: PM disabled to allow functionality at cost to power
  - ✓ Even best case not good – reluctance to power down leaves some PM opportunities on the table
    - Tough balancing act between performance / functionality and power
- Scope of problem is increasing
  - ✓ Power efficiency doesn't just apply to mobile any more
    - Such as Energy Star compliance

Wanted: Mechanism for platform to tune PM based on actual device service requirements

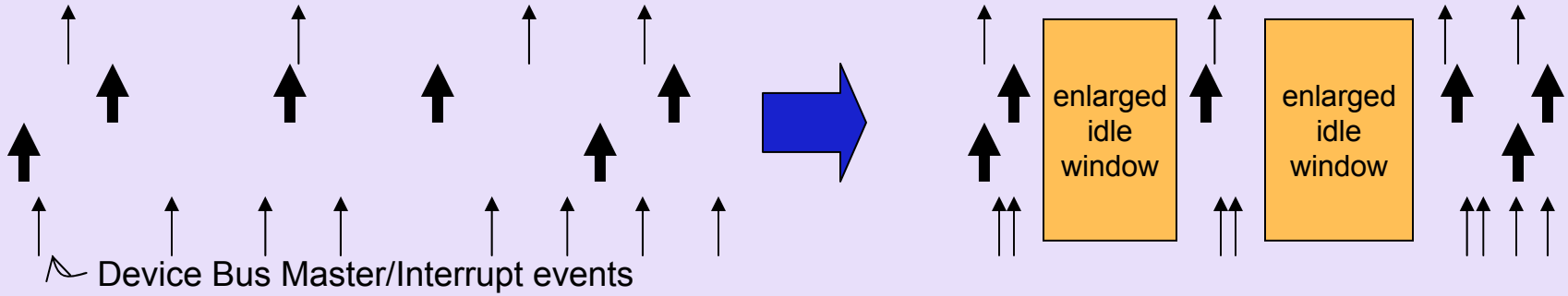
Proposal: Latency Tolerance Reporting (LTR)

- Concept: Msg for device service latency Requirement Reporting
  - ✓ PCIe message sent by Endpoint w/required latency
    - Capability to report both snooped & non-snooped values
  - ✓ "Terminate at Receiver" routing
    - MF devices & Switches coalesce messages from below & send single message up
- Provides Device benefit
  - ✓ Devices can dynamically limit platform PM state as a function of device activity level – avoids performance pitfalls
- Provides Platform benefit
  - ✓ LTR removes guess work from platform PM, enables greater power savings without impact to performance/functionality

LTR enables dynamic power vs. performance tradeoffs at minimal cost impact

Reducing Platform Power through Optimized Buffer Flush/Fill

- Problem Statement: Devices cannot 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?

- 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
  - ✓ WAKE# currently defined for use in L2/L3 - Devices drive, input to platform
  - ✓ Define new semantic for OBFF - Platform central resource drives WAKE# for hint to devices
- Solution 2: WAKE# not available – Use PCIe Message
  - Con: Requires waking links – provide protocol mechanism(s) to avoid needless link wakeups
- Platform Firm/Software to Select 1 or 2 per Link
  - ✓ Switches will translate when needed

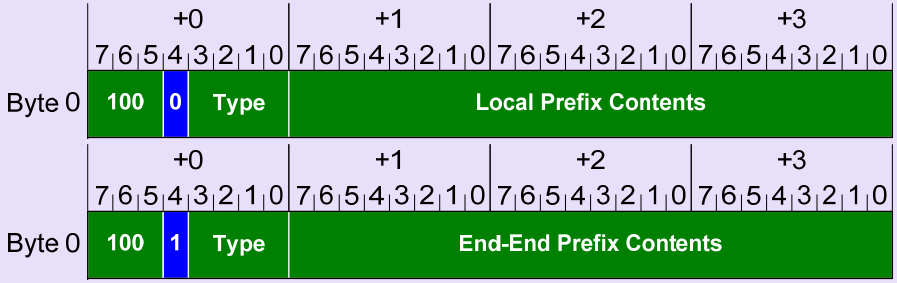
Greatest Potential Improvement When Implemented by All Platform Devices

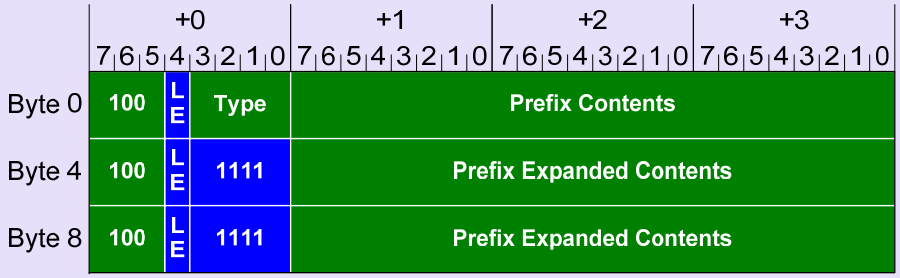
Generic TLP Prefix Header Overview

Motivation

- Emerging usage models require increase in header size to carry new information
  - ✓ Example: Multi-Root IOV, Extended OPH
- TLP Prefix mechanism extends the header sizes by adding Dwords to the front of headers
- Supports Mixed topologies
  - ✓ i.e. fabrics with components that support different sets of prefixes (including none)
- Components can selectively originate Prefixes
  - ✓ e.g. TLPs originating at component X and destined for component A might have a prefix while TLPs originating at component X and destined for component B might not
  - ✓ Application/Software must ensure that the path from requestor to completer supports TLP prefix (of the appropriate type)
    - e.g. OPH sender using Peer-to-Peer must know destination and adjust
- Baseline Error rules continue to apply
  - ✓ e.g. If a Base TLP Header is Malformed, prefixes won't change that

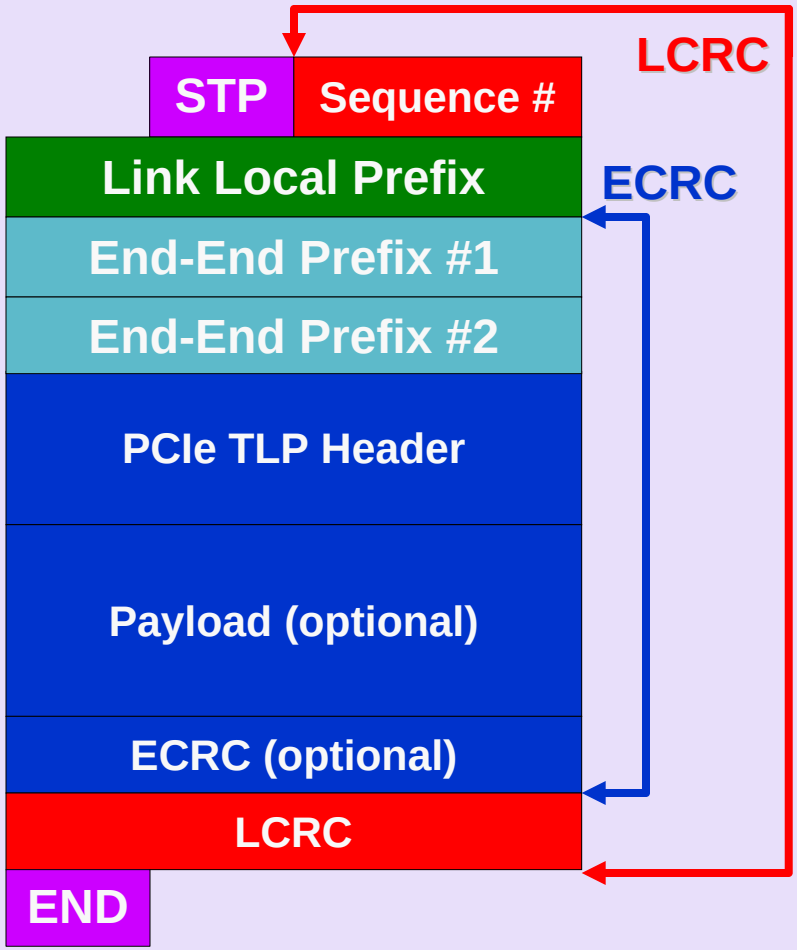
Prefix Encoding





- Base TLP Prefix Size – 1 DW
  - ✓ Appended to TLP headers
- TLP Prefixes can be expanded or stacked
  - ✓ Type encoding of 1111 is used for expansion of previous TLP prefix
  - ✓ End-End TLP prefix has an upper limit of 4 Dwords (16 Bytes)
- Link Local – Where routing elements may process the TLP for routing or other purposes.
  - ✓ Only usable when both ends understand and are enabled to handle link local TLP prefix
  - ✓ ECRC not applicable
- End-End TLP Prefix
  - ✓ Requires support between the requester, completer and routing elements
  - ✓ End-End TLP Prefix not required to but is permitted to be protected by ECRC
    - – If underlying Base TLP is protected by ECRC then End-End TLP Prefix is also protected by ECRC
  - ✓ Upper bound of 4Dwords (16 Bytes) for End-End TLP Prefix
- Note: Bit 4 (LE) indicates Link Local vs. End-End TLP prefi

Stacked Prefix Example:



- Link Local is first
  - ✓ Starts at 0
  - ✓ $Type_{L1}$
- End-End #1 follows Link Local
  - ✓ Starts at 4
  - ✓ $Type_{E1}$
- End-End #2 follows End-End #1
  - ✓ Starts at 8
  - ✓ $Type_{E2}$
- PCIe Header follows End-End #2
  - ✓ Starts at 12
- Switch routes using Link Local and PCIe Header
  - ✓ … and possibly additional Link Local DWORDs
    - if more extension bits needed
  - ✓ Malformed TLP if don't understand
- Switch forwards End-End Prefixes unaltered
  - ✓ End-End Prefixes do not affect routing
  - ✓ Up to 4 DWORDs (16 Bytes) of End-End Prefix
- End-End Prefixes are optional
  - ✓ Different End-End Prefixes sequence can be unordered
    - affects ECRC but does not affect meaning
  - ✓ Repeated End-End Prefix sequence must be ordered
    - e.g. 1st Extended OPH vs. 2nd Extended OPH attribute
    - meaning of this is defined by each End-End Prefix

Call to Action

Call to Action

- Innovate and differentiate your products with PCI Express 2.0 industry standard
- Review and provide feedback on PCIe protocol extensions ECRs
- Contribute to the evolution of PCI Express architecture
- Visit www.pcisig.com for PCI Express specification updates

Thank you for attending the
PCIe Technology Seminar

For more information please go to
www.pcisig.com