PCIe® 3.0 Preview

Jasmin Ajanovic
PCI-SIG® Protocol WG & Bridge WG Chairman
Intel Corporation

Objective

- PCIe® Protocol Work Group is actively developing protocol extension ECRs to PCIe 2.0 specification
  - ✓ A set of approved errata and ECNs to PCIe 2.0 specification available
  - ✓ Approved Errata and ECNs consolidated into PCIe 2.1 specification
- PCIe Protocol and Electrical Work Groups are actively developing the technology to deliver doubling of bandwidth over PCIe 2.0 specification
  - ✓ 8GT/s signaling technology developed by Electrical Working Group
  - ✓ New Encoding scheme developed by Protocol Working Group
  - ✓ All errata and ECNs to be incorporated into PCIe 3.0 Base
- Objective of this training material is to:
  - ✓ Provide an overview of PCIe 3.0 technology currently under development
  - ✓ Obtain early and intermediate feedback

PCIe 3.0 Phy Logical Layer

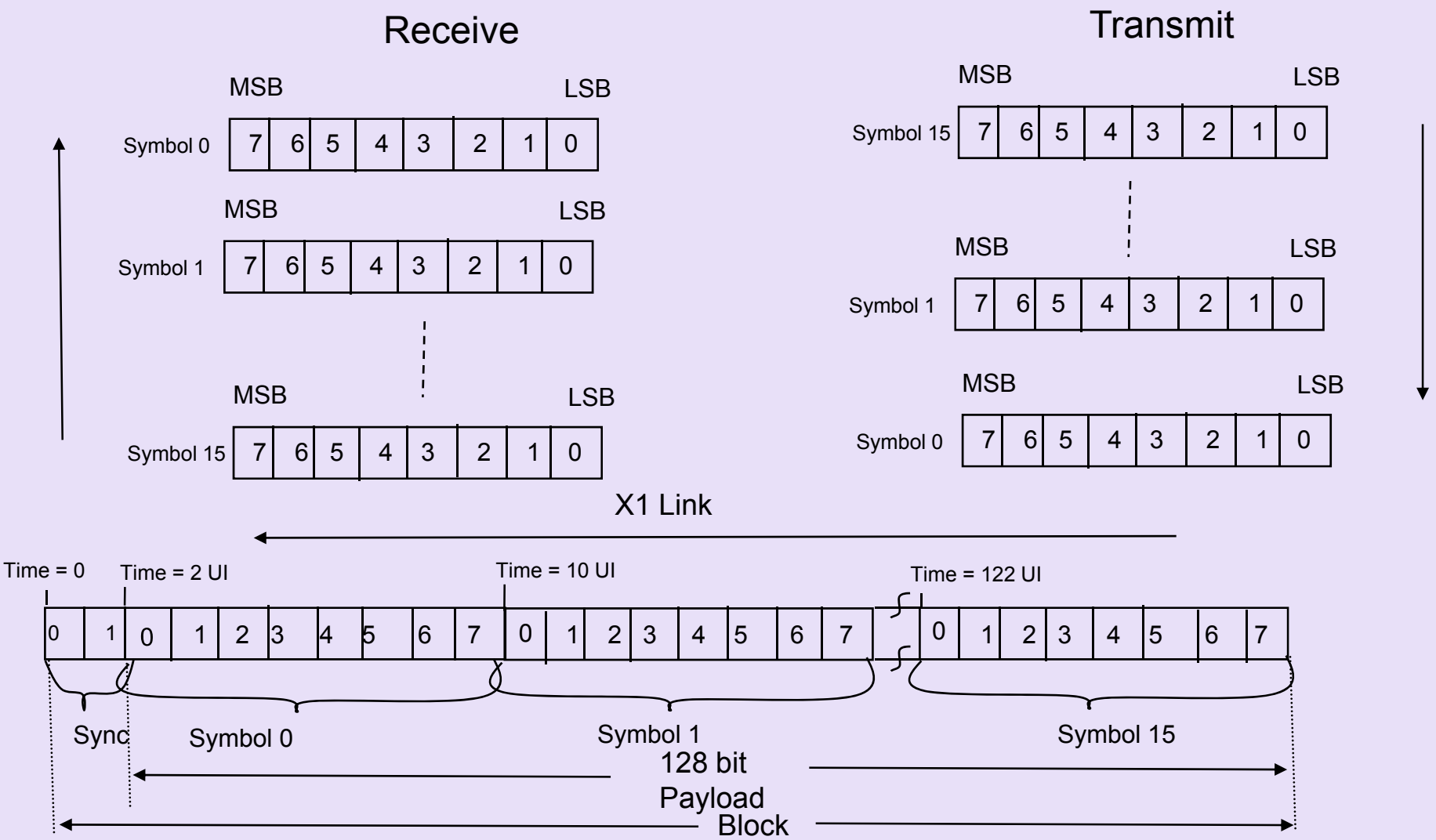
Problem Statement

- PCIe® 3.0 data rate decision: 8 GT/s
  - ✓ High Volume Manufacturing channel for client/ servers
    - Same channels and length for backwards compatibility assuming worst-case
  - ✓ Low power and ease of design
    - Avoid using complicated receiver equalization, etc.
- Requirement: Double Bandwidth from PCIe 2.0
  - ✓ PCIe 1.0a data rate: 2.5 GT/s
  - ✓ PCIe 2.0 data rate: 5 GT/s
    - Doubled the bandwidth from PCIe 1.x to PCIe 2.0 by doubling the data rate
  - ✓ Data rate gives us a 60% boost in bandwidth
  - ✓ Rest will come from Encoding
    - Replace 8b/10b encoding with a scrambling-only encoding scheme when operating at PCIe 3.0 data rate
- Double B/W: Encoding efficiency improvement of 1.25 X data rate improvement of 1.6 yields 2X improvement in bandwidth
- Challenge: 8b/10b encoded the 2^8 data patterns and 12 K-codes

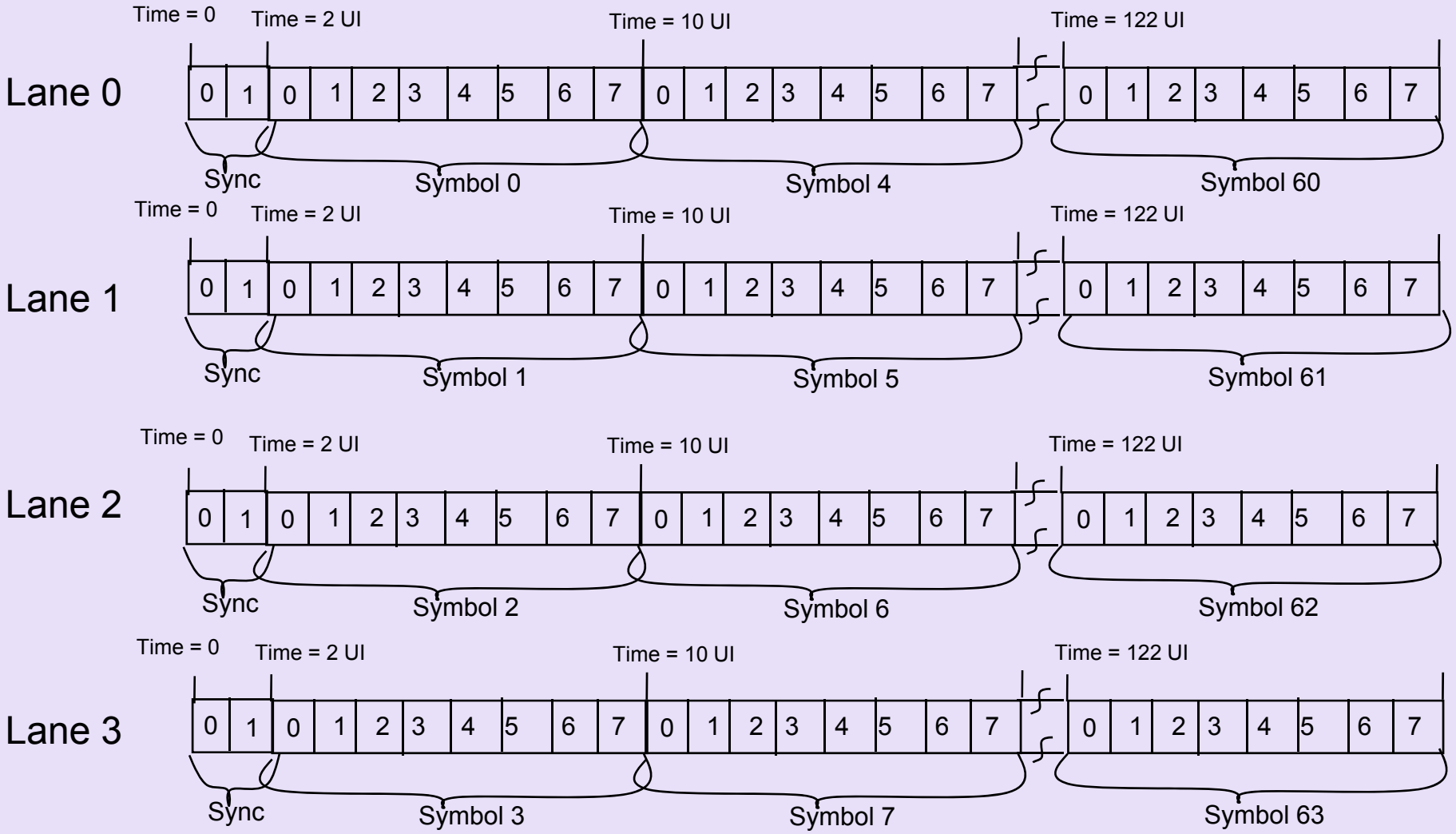
Overall Scheme

- Current direction is to use two levels of encapsulation
  - ✓ Lane level encoding (e.g., 128/130 code) on individual lanes
  - ✓ Physical layer packetization to identify “packet” boundaries
- Lane Level Encoding
  - ✓ Mostly 128/130 bit code. The per-Lane code is called a Block
    - Two types of Blocks: Data Block (TLP, DLLP, LIDL) and Ordered Set Blocks
    - SKP OS blocks Variable in length (may not be 130 bits)
  - ✓ 2 bit Sync Header followed by payload (mostly 128 bit)
  - ✓ Sync header not scrambled
    - 10b (0b followed by 1b in the wire) used for Data Blocks
    - 01b (1b followed by 0b in the wire) used for Ordered Set Blocks
  - ✓ Block lock
    - EIEOS Ordered Set substitutes COM used for Symbol lock in 8b/10b
- Phy Layer packetization to identify packet boundaries. Packet types:
  - ✓ Link Level (TLP or DLLP or LIDL)
  - ✓ Lane Level (Ordered Sets)
- Scrambling only (no 8b/10b) to provide edge density
  - ✓ Additive scrambling on a per-lane basis
  - ✓ Degree 23 polynomial for LFSR with different taps for 8 adjacent lanes (or different seeds for same tap)
  - ✓ Electrical Idle Exit Ordered Set resets scrambler (Recovery/ Config)

Mapping of bits on a x1 Link



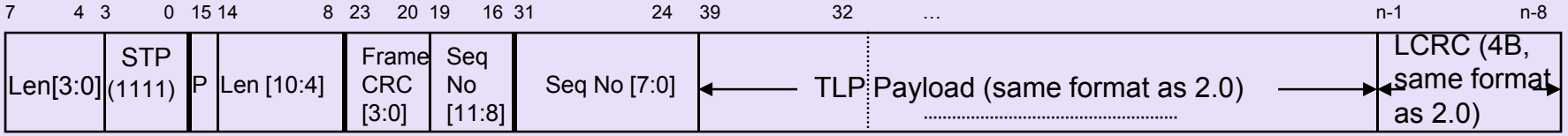
Mapping of bits on a x4 Link



Physical Layer Encapsulation

- First Symbol (scrambled) indicates packet type:
  - ✓ 00000000 is Logical IDL
    - All subsequent lanes in same Symbol-time should be LIDL
    - Receivers check for all 0s (after descrambling) in LIDL
    - PAD functionality merged with LIDL
  - ✓ 1111xxxx is STP
    - Subsequent 11 bits (link wide) define the length
  - ✓ 00001111 is SDP
    - 2nd Symbol also gets a fixed encoding
  - ✓ 00000011 is EDB
    - EDB packet is 4 Symbols; each with the same value 00000011

P-Layer Encapsulation: TLP

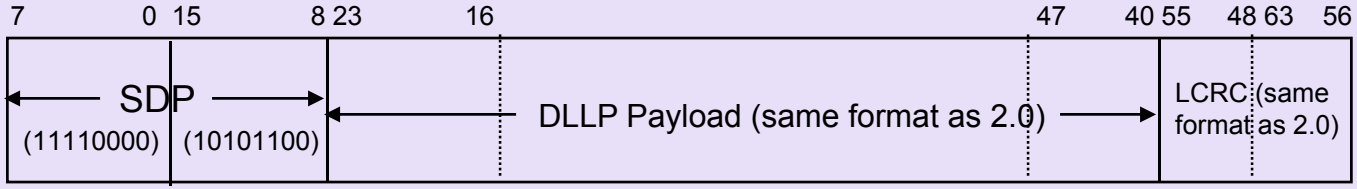


[Len[10:0]: length of the TLP in DWs, Frame CRC[4:0]: Check Bits covering Length[0:10], P: Frame Parity, No END]

- Length known from the first 3 Symbols
  - ✓ First 4 bits are 1111 (bit[0:3] = 4'b1111)
  - ✓ Bits 4:14 has the length of the TLP (valid values: 5 to 1031)*
  - ✓ Bits 15 and 20:23 is check bits to cover the TLP Length field
    - – Primitive Polynomial (X4 + X + 1) protects 15 bit field
      - Provides double bit flip detection guarantee (length 11 bits + CRC 4 bits)
    - – Odd parity covers the 15 bits (length 11 bits + CRC 4 bits)
      - Guaranteed detection of triple bit errors (over 16 bits)
- Sequence Number occupies bits 16:19 and 24:31
- TLP payload is from the 4th Symbol position (same as 2.0)
- No explicit END. Need to check first Symbol after TLP for implicit END vs an explicit EDB => Ensures triple bit flip detection
- All Symbols are (de)scrambled

*Note: Valid values for a TLP Prefix is 5 to ~ 1039 (Max value depends on type of TLP Prefix)

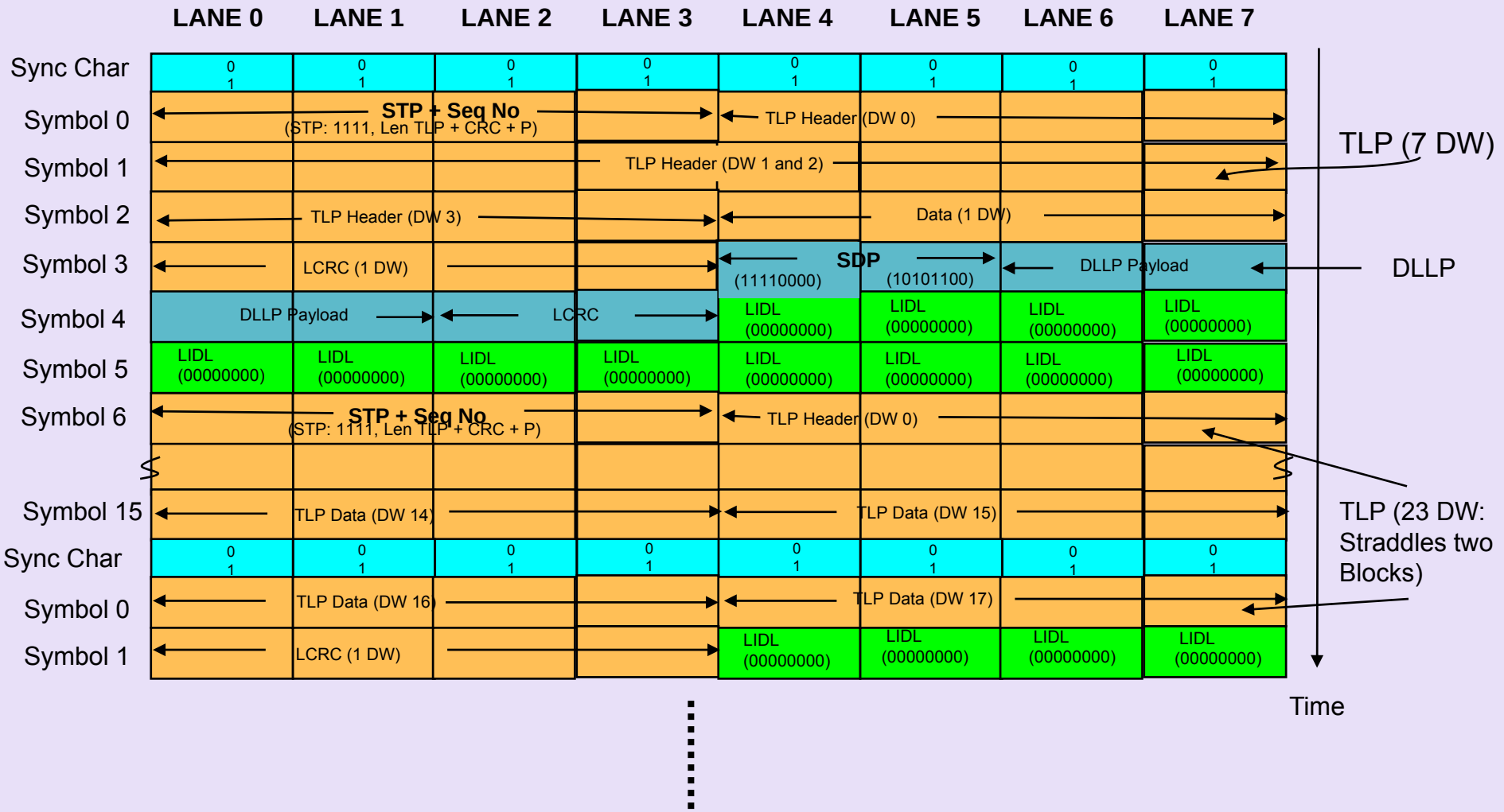
P-Layer Encapsulation: DLLP



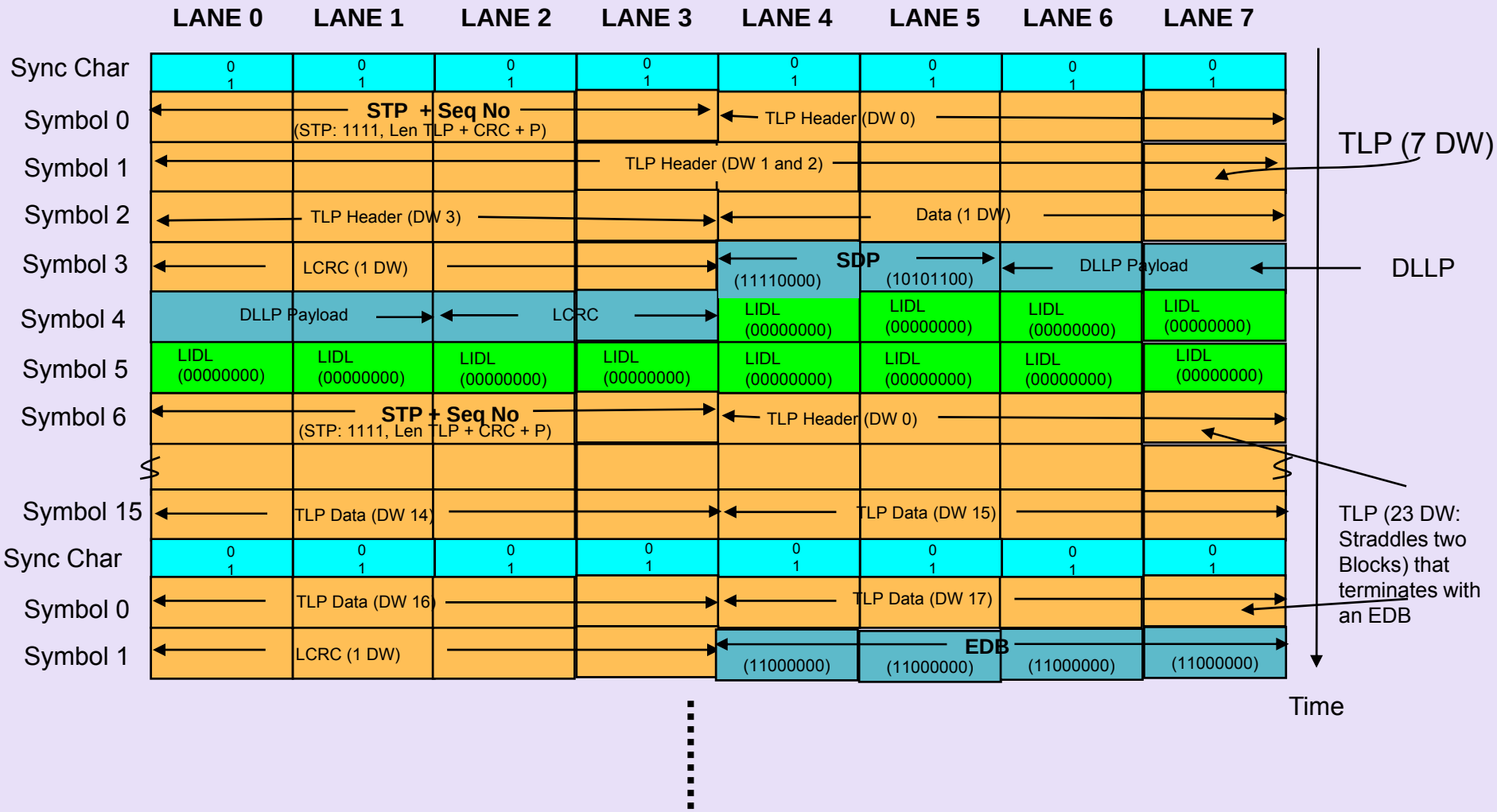
(DLLP Layout)

- Preserve DLLP layout of 2.0 spec
- First Symbol is F0h
- Second Symbol is ACh
- Next 4 Symbols (2 through 5) are the DLLP layout
- Next 2 Symbols (6 and 7): LCRC (identical to 2.0)
- No explicit END
- All Symbols are (de)scrambled

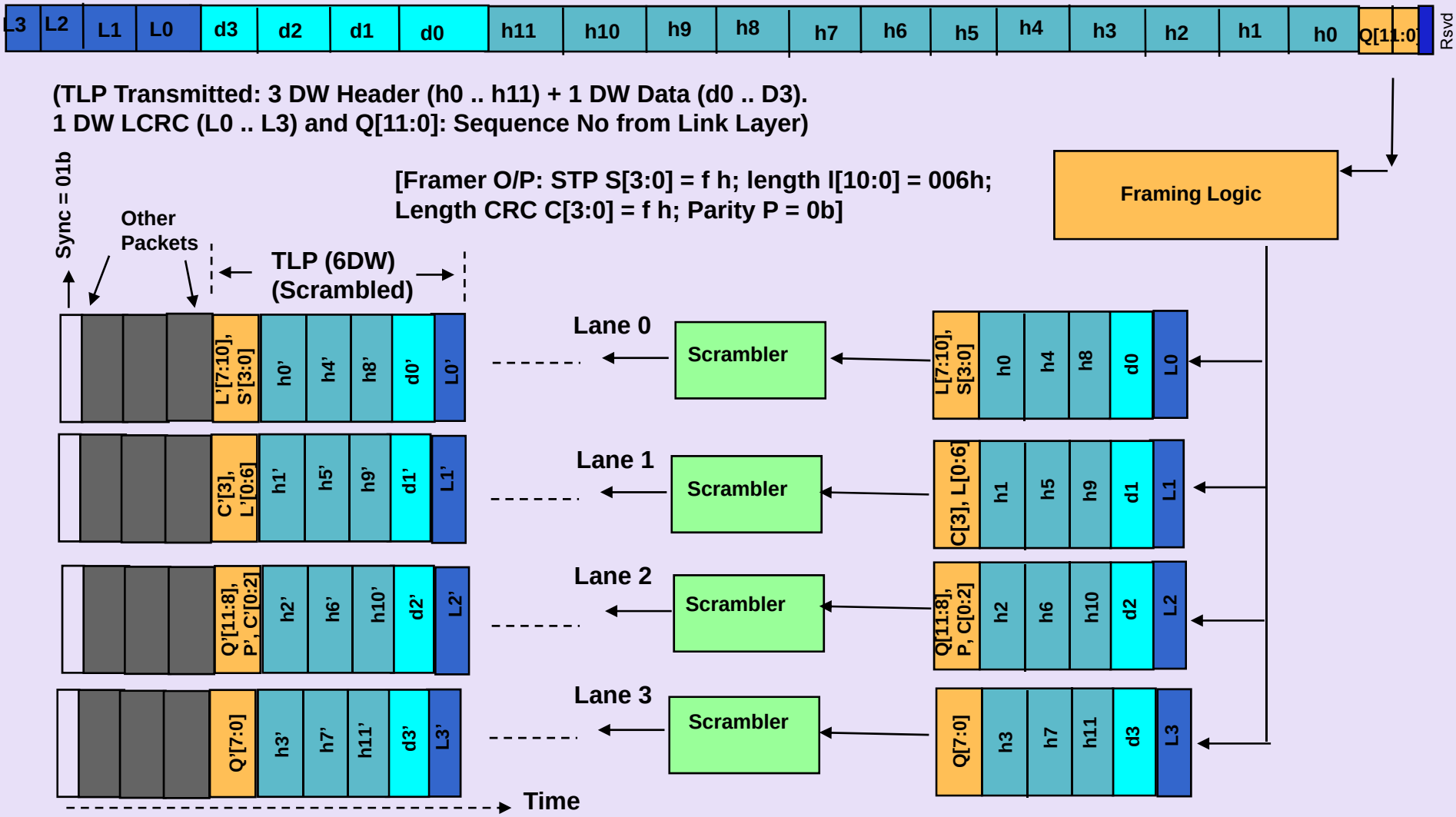
Ex: TLP/ DLLP/ IDLs in x8



Ex: TLP/ DLLP/ IDLs in x8



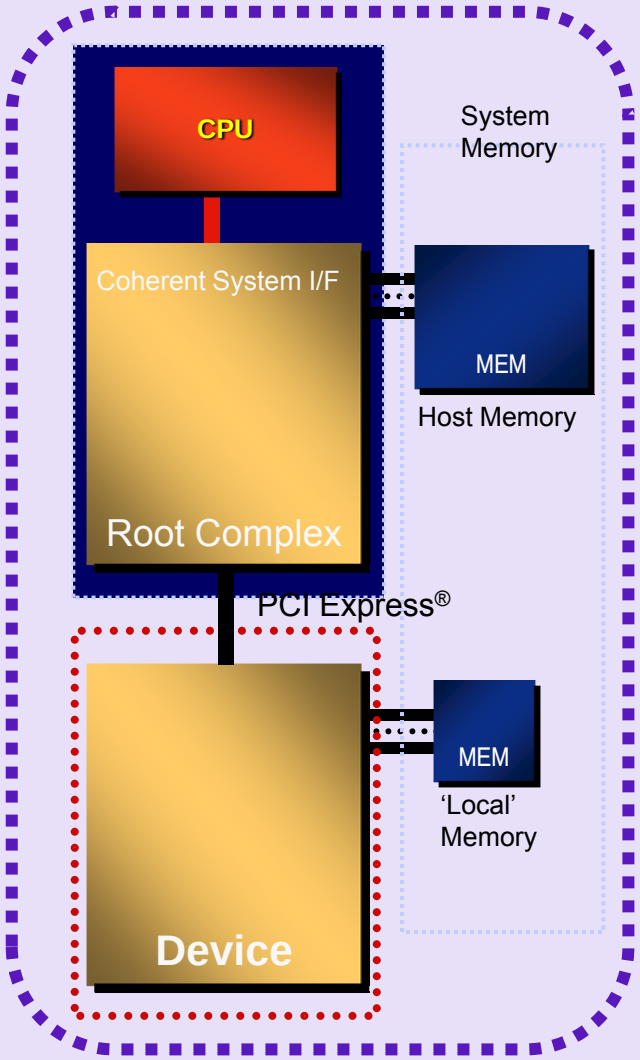
TLP Transmission in a X4 Link



Error Detection and Recovery

- Framing error is detected by the physical layer
  - ✓ The first byte of a packet is not one of the allowed sets (e.g., TLP, DLLP, LIDL)
  - ✓ Sync character is not 01 or 10
  - ✓ Same sync character not present in all lanes after deskew
  - ✓ CRC error in the length field of a TLP
  - ✓ Ordered set not one of the allowed encodings or not all lanes sending the same ordered set after deskew (if applicable)
  - ✓ 10 sync header received after 01 sync header without a marker packet in the 01 sync header OR received a marker packet in the 01 sync header and the subsequent sync header in any lane not 10
- Any framing error requires directing LTSSM to Recovery
  - ✓ Stop processing any received TLP/ DLLP after error until we get through Recovery
  - ✓ Block lock acquired with EIEOS
  - ✓ Scrambler reset with each EIEOS
- Error Detection Guarantees
  - ✓ Triple bit flip detection within each TLP/ DLLP/ IDL/ OS

PCIe Protocol Extensions



- Performance Improvements
  - ✓ **TLP Processing Hints** – hints to optimize system resources and performance
  - ✓ **TLP Prefix** – mech to extend TLP headers for TLP Processing Hints, MR-IOV, and future extensions
  - ✓ **ID-Based Ordering** – Transaction-level attribute/hint to optimize ordering within RC and memory subsystem
  - ✓ **Extended Tag Enable Default** – permits default for Extended Tag Enable bit to be Function-specific
- Software Model Improvements
  - ✓ **Atomic Operations** – new atomic transactions to reduce synchronization overhead
  - ✓ ***Page Request Interface*** *– mech in ATS 1.1 for a device to request faulted pages to be made available (not covered)*
- Communication Model Enhancements
  - ✓ **Multicast** – mechanism to transfer common data or commands sent from one source to multiple recipients
- Power Management
  - ✓ **Dynamic Power Allocation** – support for dynamic power operational modes through standard configuration mech
  - ✓ **Latency Tolerance Reporting** – Endpoints report service latency requirements for improved platform power mgmt
  - ✓ **Optimized Buffer Flush/Fill** – Mechs for devices to align DMA activity for improved platform power mgmt
- Configuration Enhancements
  - ✓ **Resizable BAR**– Mechanism to support BAR size negotiation
  - ✓ **Internal Error Reporting**– Extend AER to report component internal errors and record multiple error logs

Protocol Extensions

Protocol Extensions Summary

Extension	Description	Status (as of 02/2009)
Atomic Operations (AtomicOps)	Atomic Read-Modify-Write mechanism	PCIe 2.1 spec
Internal Error Reporting	Extend AER to report component internal errors and record multiple error logs	PCIe 2.1 spec
Resizable BAR	Mechanism to support BAR size negotiation	PCIe 2.1 spec
Multicast	Address-Based Multicast of Posted Request TLPs	PCIe 2.1 spec
ID-Based Ordering (IDO)	New type of relaxed ordering semantics to improve performance	PCIe 2.1 spec
Dynamic Power Allocation (DPA)	Dynamic power mgmt for substates of D0 (active state)	PCIe 2.1 spec
Latency Tolerance Reporting (LTR)	Endpoints report service latency requirements, enabling improved platform power mgmt	PCIe 2.1 spec
Extended Tag Enable Default	Permits default for Extended Tag Enable bit to be Function-specific instead of 0b	PCIe 2.1 spec
TLP Processing Hints (TPH)	Hints for optimized TLP processing within host memory/cache hierarchy	PCIe 2.1 spec
Optimized Buffer Flush/Fill (OBFF)	Mechanisms for devices to align DMA activity for improved platform power mgmt	ECN pending membership review
TLP Prefix	Mechanism to extend TLP headers	PCIe 2.1 spec

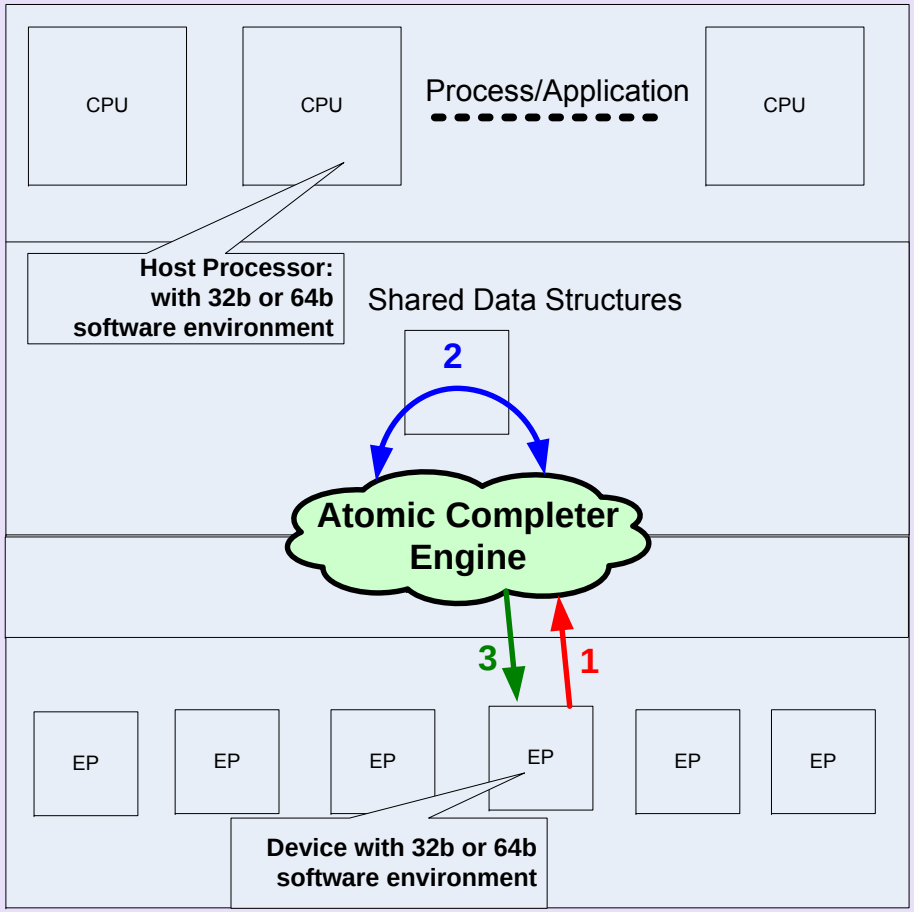
Atomic Operations (AtomicOps)

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

AtomicOps Mechanism



1. 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
2. Atomic Operation Execution
   - ✓ Atomic Completer Engine performs atomic read-modify-write operation
     - Read initial value
     - Compute and write new value
3. AtomicOp Completion
   - ✓ Completion returns initial value
   - ✓ "Standard" Completion for Non-Posted Request

*Device to System memory shown in figure for illustration purposes

AtomicOps 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 mechs (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

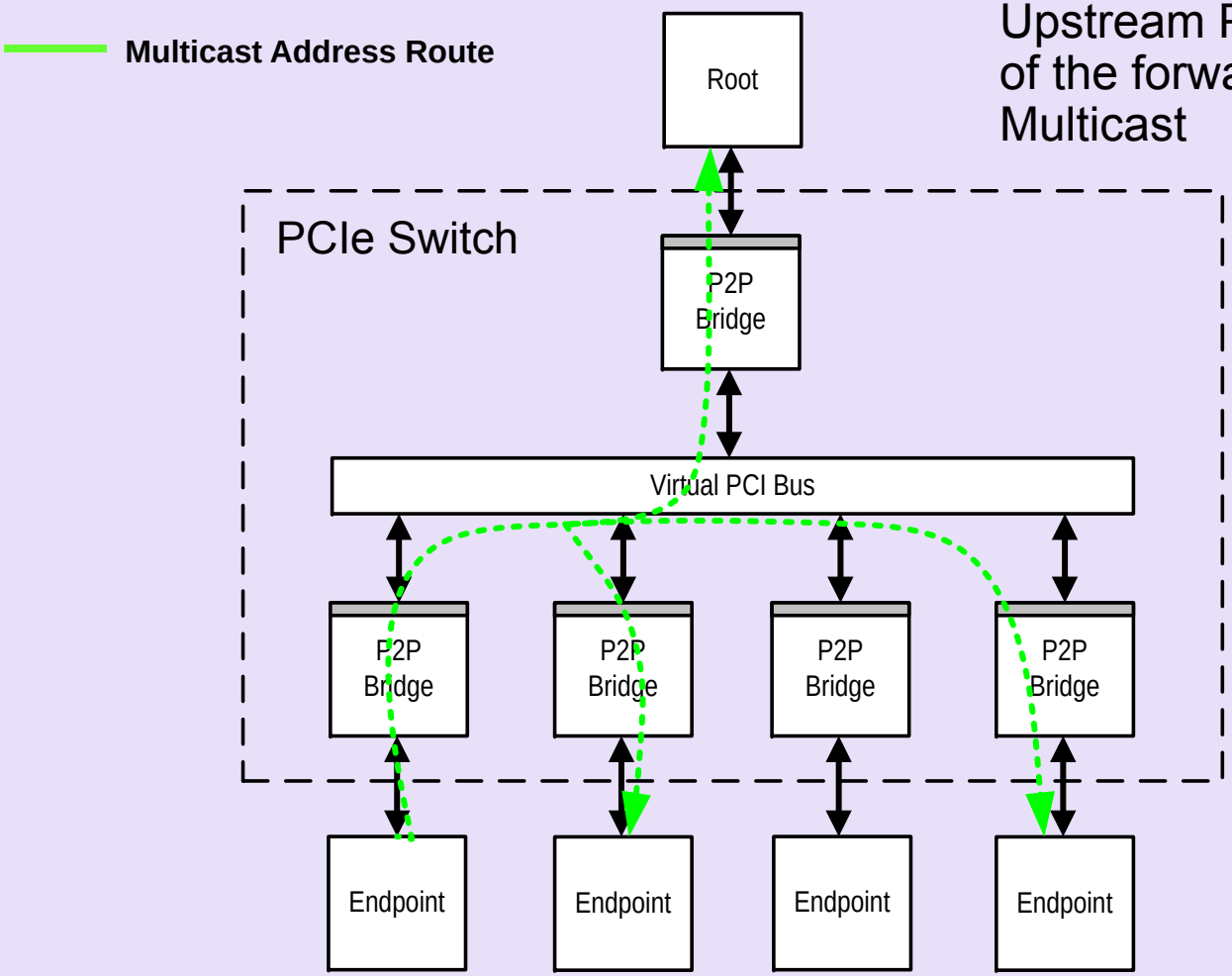
Multicast

Multicast Motivation & Mechanism Basics

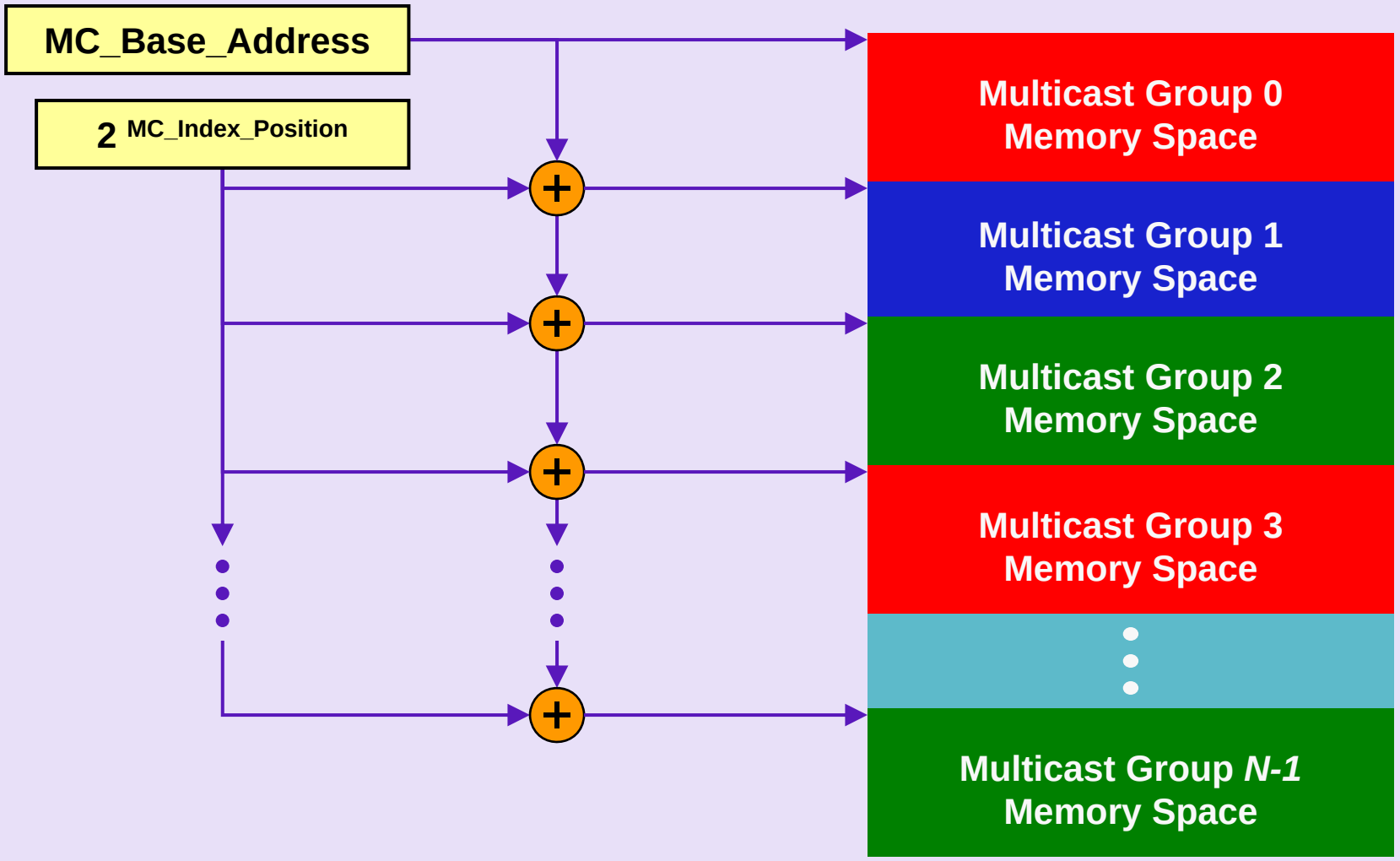
- Several key applications benefit from Multicast
  - ✓ Communications backplane (e.g. route table updates, support of IP Multicast)
  - ✓ Storage (e.g., mirroring, RAID)
  - ✓ Multi-headed graphics
- PCIe architecture extended to support address-based Multicast
  - ✓ New Multicast BAR to define Multicast address space
  - ✓ New Multicast Capability structure to configure routing elements and Endpoints for Multicast address decode and routing
  - ✓ New Multicast Overlay mechanism in Egress Ports allow Endpoints to receive Multicast TLPs without requiring Endpoint Multicast Capability structure
- 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

Multicast Example

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



Multicast Memory Space



Multicast TLP Blocking

- Permits a routing element Ingress Port to block incoming MC TLPs, or an Endpoint Function to block outgoing MC TLPs
- For MC TLPs, extract the Multicast Group (MCG) number from the target address and perform MC Blocked TLP checking based on the MCG number:
  - ✓ If the Multicast TLP came in an Ingress Port of a routing element 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

Multicast Overlay Mech

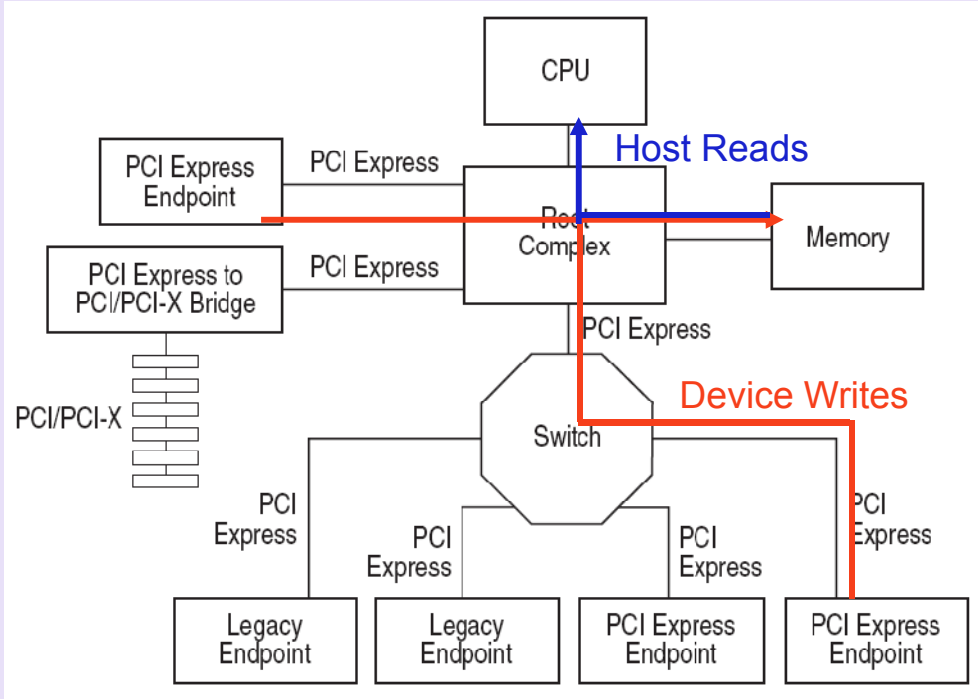
- Mechanism allows an Egress Port to remap the address of an outgoing MC TLP, effectively overlaying Multicast Memory Space on top of a target unicast address range
  - ✓ In a Downstream Port, this enables the MC TLP to target (unicast) Memory Space accepted by an Endpoint that lacks a Multicast Capability structure
  - ✓ At a Switch Upstream Port, Multicast space can be remapped onto a Memory Space range associated with Host Memory
- Due to the address being modified, ECRC (if present) must either be stripped or regenerated
  - ✓ ECRC rules for Multicast Overlay are as shown in table below

MC_Overlay Enabled	TLP has ECRC	ECRC Regeneration Supported	Action if ECRC Check Passes	Action if ECRC Check Fails
No	x	x	Forward TLP unmodified	
Yes	No	x	Forward modified TLP	
Yes	Yes	No	Forward modified TLP with ECRC stripped	
Yes	Yes	Yes	Forward modified TLP with regenerated ECRC	Forward modified TLP with inverted regenerated ECRC

TLP Processing Hints (TPH)

TPH 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)
- TLP 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



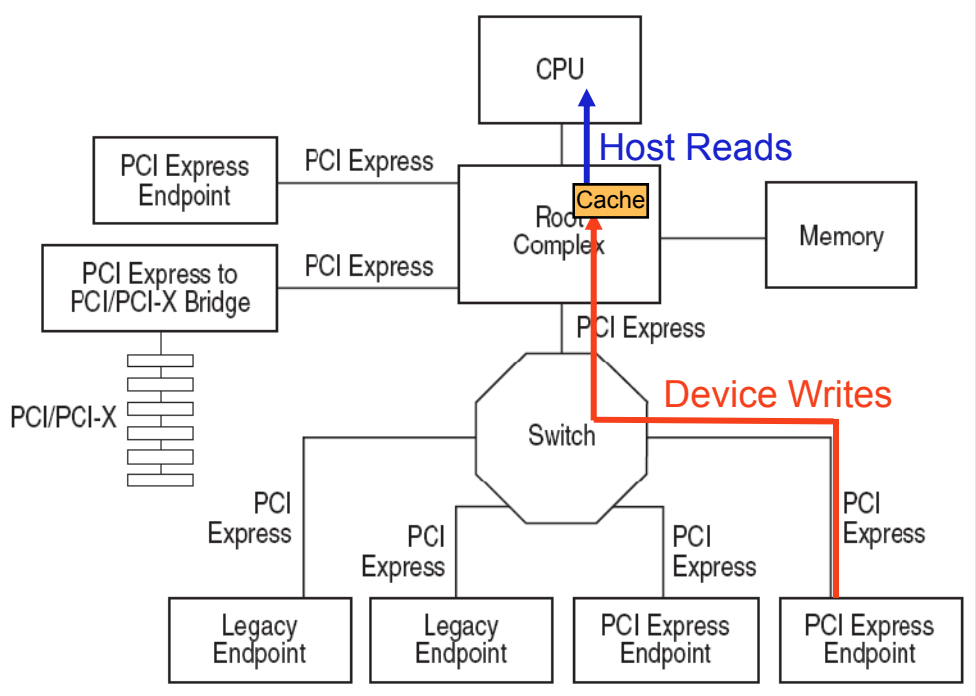
TPH Usage Models

- **Data Structures of interest**
  - ✓ Control Data; e.g., Descriptors
  - ✓ Headers for protocol processing
  - ✓ Payload for data copies

- **Usage Models**
  - ✓ Efficient processing of network I/O in systems with variable access latencies
    - I/O 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.

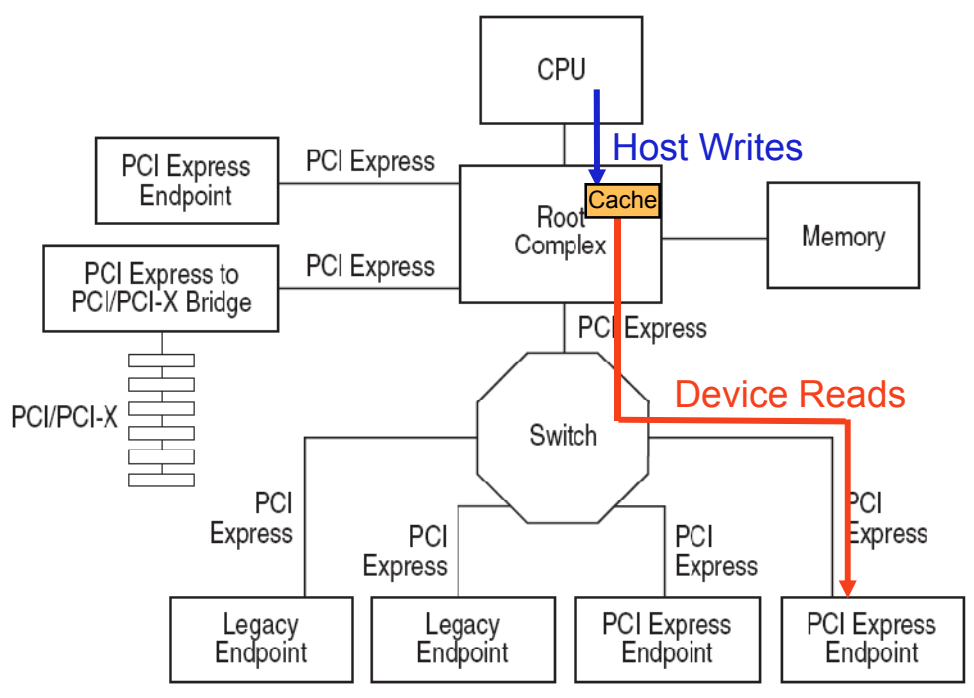
Request Patterns
Detail

- Device Writes Host Reads (DWHR)
  - ✓ Push data to host agent
    - Reduce average overhead for access by host agent
  - ✓ Utilize opportunistic allocation policy
    - i.e. Update on line allocated and dirty
- DWHR (prioritized)
  - ✓ Push data to host agent
    - Reduce overhead for access by host agent
    - Minimize latency to single traversal through system fabric
  - ✓ Preferential allocation policy
    - i.e. Allocate/update on request
  - ✓ Prioritize retention of line
    - i.e. bias LRU to take advantage of temporal locality
  - ✓ Use for critical communication structures, flags, etc.



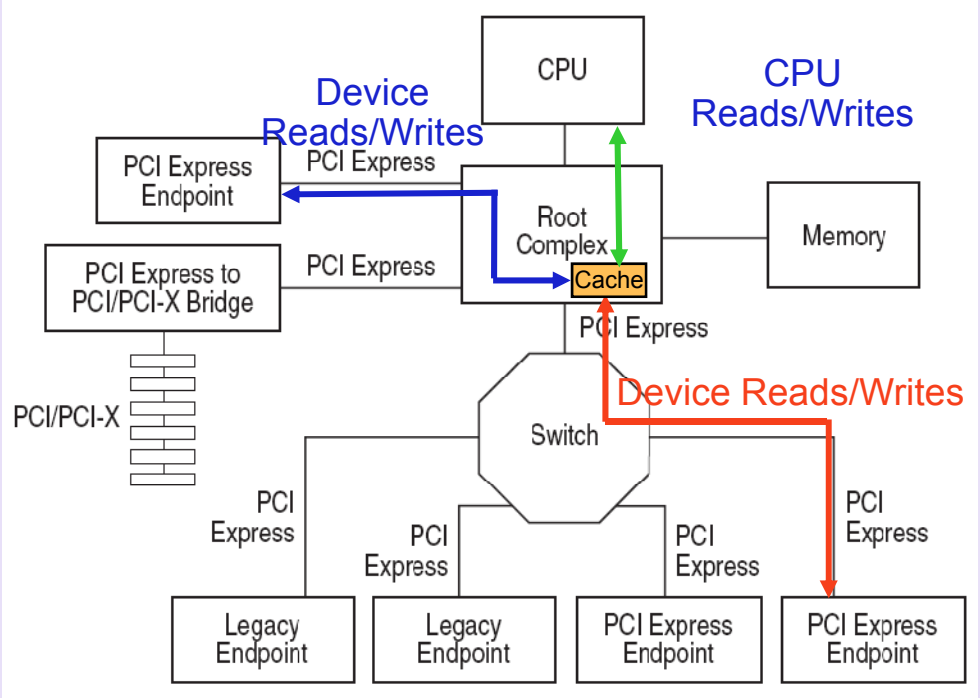
Request Patterns
Detail (cont)

- Host Writes Device Reads (HWDR)
  - ✓ Pull data from host agent
    - Minimize latency to single traversal through system fabric
- HWDR (prioritized)
  - ✓ Pull data from host agent
    - Minimize latency to single traversal through system fabric
  - ✓ Prioritize retention of line
    - Take advantage of temporal locality
  - ✓ Use for critical communication structures, flags, etc.

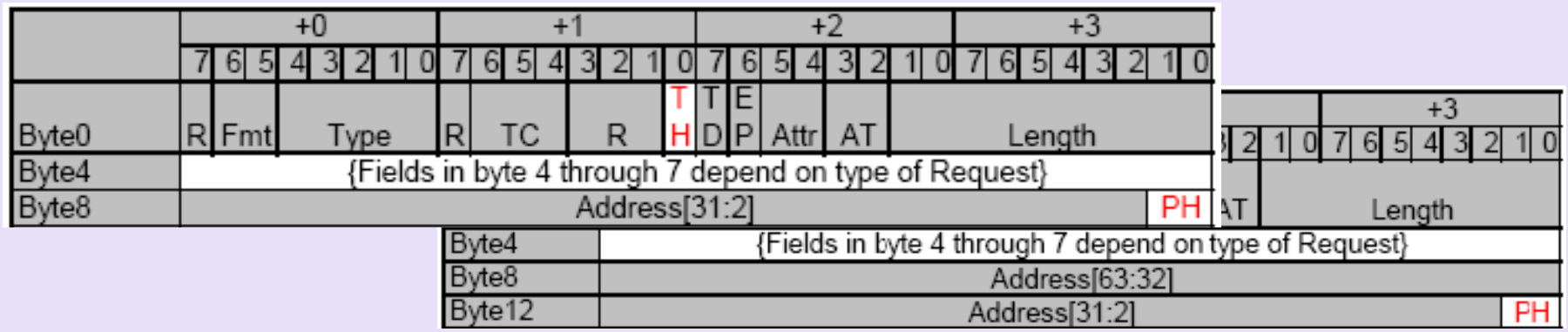


Request Patterns
Detail (cont)

- Device Reads/Writes, Device Reads/Writes (D*D*)
  - ✓ Mem-free state save
    - Reduced latency, improve number of supported contexts
    - Reduce required device resident memory
  - ✓ Utilized system caches for device “read mostly” data structures.
  - ✓ Steering tag is undefined
- Bi-directional shared structure access
  - ✓ Tag key data structures for symmetric frequent access from both Device and Host
  - ✓ System places line in optimal cache in path between Device and Host Agent
  - ✓ Example: Link list structures with insertions and deletions from both Device and Host



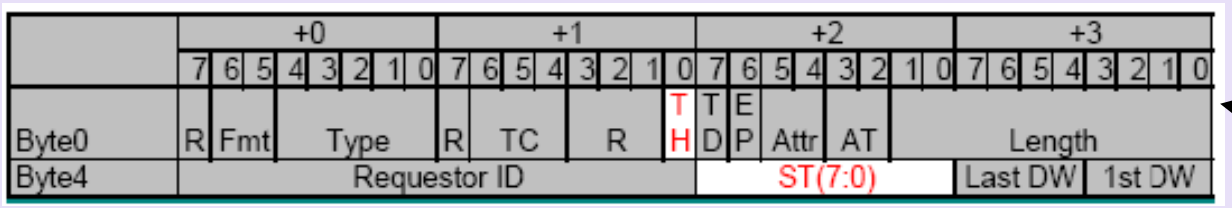
TPH Mechanism



- Mechanism to provide processing hints on per TLP basis for Requests that target Memory Space
  - ✓ Enable system hardware (ex: Root-Complex) to optimize on a per TLP basis
  - ✓ Applicable to Memory Read/Write and Atomic Operations

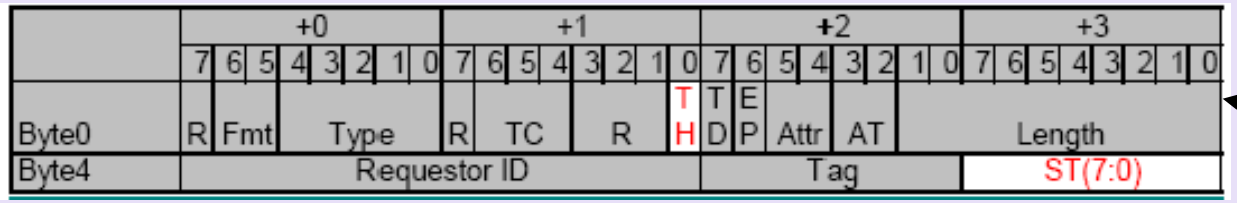
PH[1:0]	Processing Hint	Usage Model
00	Bi-directional data structure	Bi-Directional data structure
01	Requestor	D*D*
10	Target	DWHR HWDR
11	Target with Priority	DWHR (Prioritized) HWDR (Prioritized)

Steering Tag (ST)



	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte0	R	Fmt		Type					R		TC			R		TH	TD	EP	Attr		AT		Length									
Byte4	Requestor ID																ST(7:0)								Last DW				1st DW			

Memory Write TLP



	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte0	R	Fmt		Type					R		TC			R		TH	TD	EP	Attr		AT		Length									
Byte4	Requestor ID																Tag								ST(7:0)							

Memory Read or AtomicOperation TLPs

- ST: 8 bits defined in header to carry System specific Steering Tag values
  - ✓ Use of Steering Tags is optional – 'No preference' value used to indicate no steering tag preference
  - ✓ Architected Steering Table for software to program system specific steering tag values

TPH 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 & power consumption
- Ecosystem Impact
  - ✓ Software impact is 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 can make implementation tradeoffs
  - ✓ Minimal impact to Switches
- Architected software discovery, identification, and control of capabilities
  - ✓ RC support for processing hints
  - ✓ Endpoint enabling to issue hints

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
- Devices are increasingly higher consumers of system power & thermal budget
- 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-1 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

Latency Tolerance Reporting (LTR)

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

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

Latency Tolerance Reporting (LTR)

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte0	R	Fmt		Type					R	TC			Reserved				TD	EP	Attr		R				Length							
Byte4	Requester ID																Tag								Message Code							
Byte8	Reserved																Reserved															
Byte12	No-Snoop Latency																Snoop Latency															

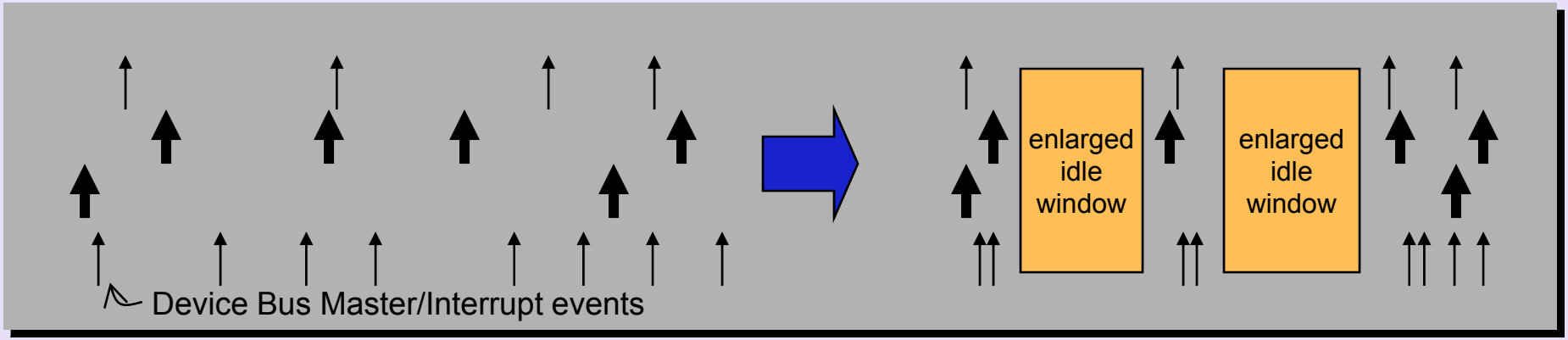
- Concept: Msg for device service latency reporting
  - ✓ PCIe Message sent by Endpoint with required service latency
    - Capability to report both snooped & non-snooped values
  - ✓ “Terminate at Receiver” routing
    - multi-Function devices & Switches coalesce Messages from below & send single Message 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

Optimized Buffer Flush & Fill

Reducing Platform Power through 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?

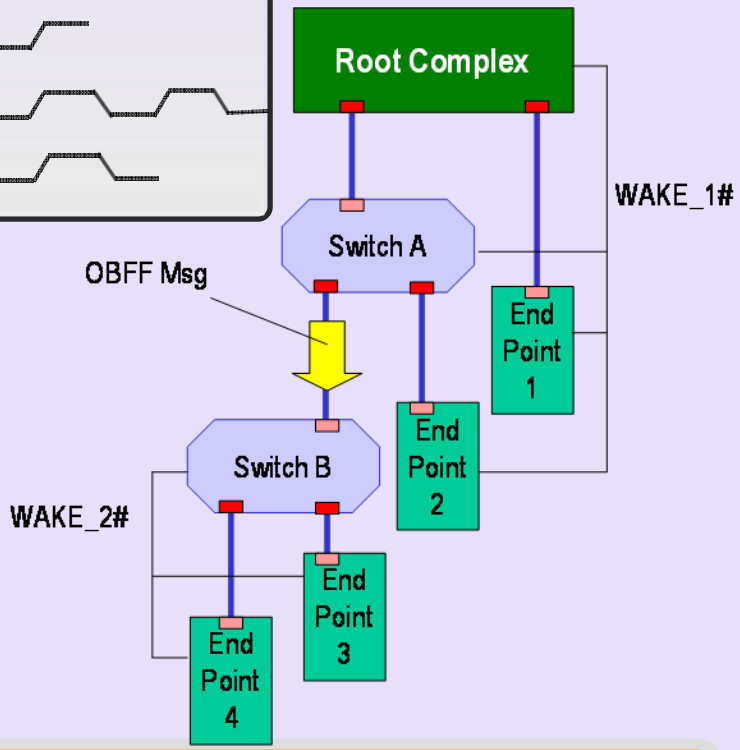
Optimal Windows

- **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

Call to Action

Call to Action

- Innovate and differentiate your products with PCI Express 3.0 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 PCI-SIG Developers Conference Europe 2009

For more information please go to www.pcisig.com

Backup Material

PCIe 2.0 Errata

PCIe 2.0 Errata Released July 2008

- Spec errors, clarifications, typos, terminology and formatting grouped into three bins
  - ✓ Bin A: “non-straight-forward changes” – 57
  - ✓ Bin B: “straight-forward changes” – 35
  - ✓ Bin C: “trivial editorial changes” – 38
- 130 total

PCIe 2.0 Errata Highlights

- Ordering Rules Summary Table and Descriptions overhauled to create a more suitable base for AtomicOps & ID-Based Ordering ECRs
  - ✓ Errata **do not** change ordering semantics, but subsequent ECRs **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

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

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

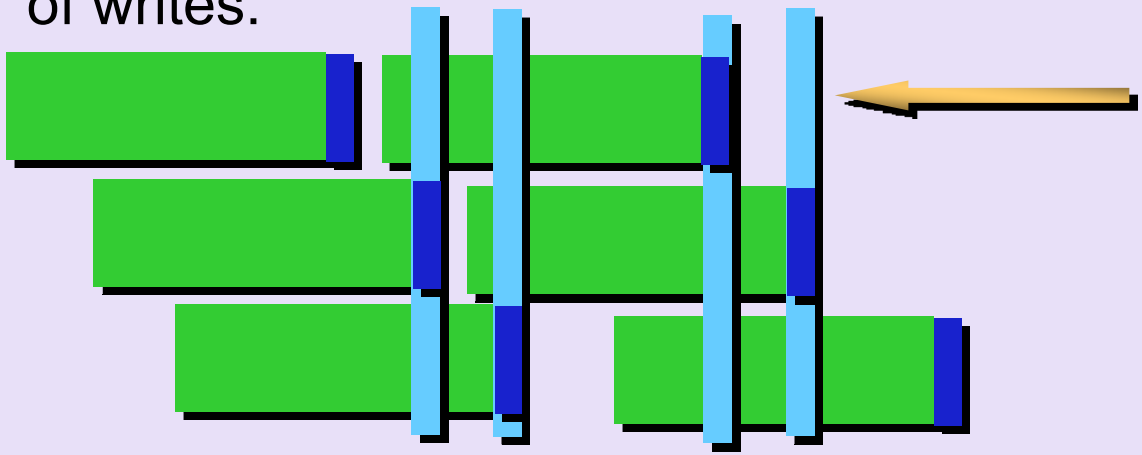
Table is based on new 2.0 errata!

"Yes" entries are required for deadlock avoidance

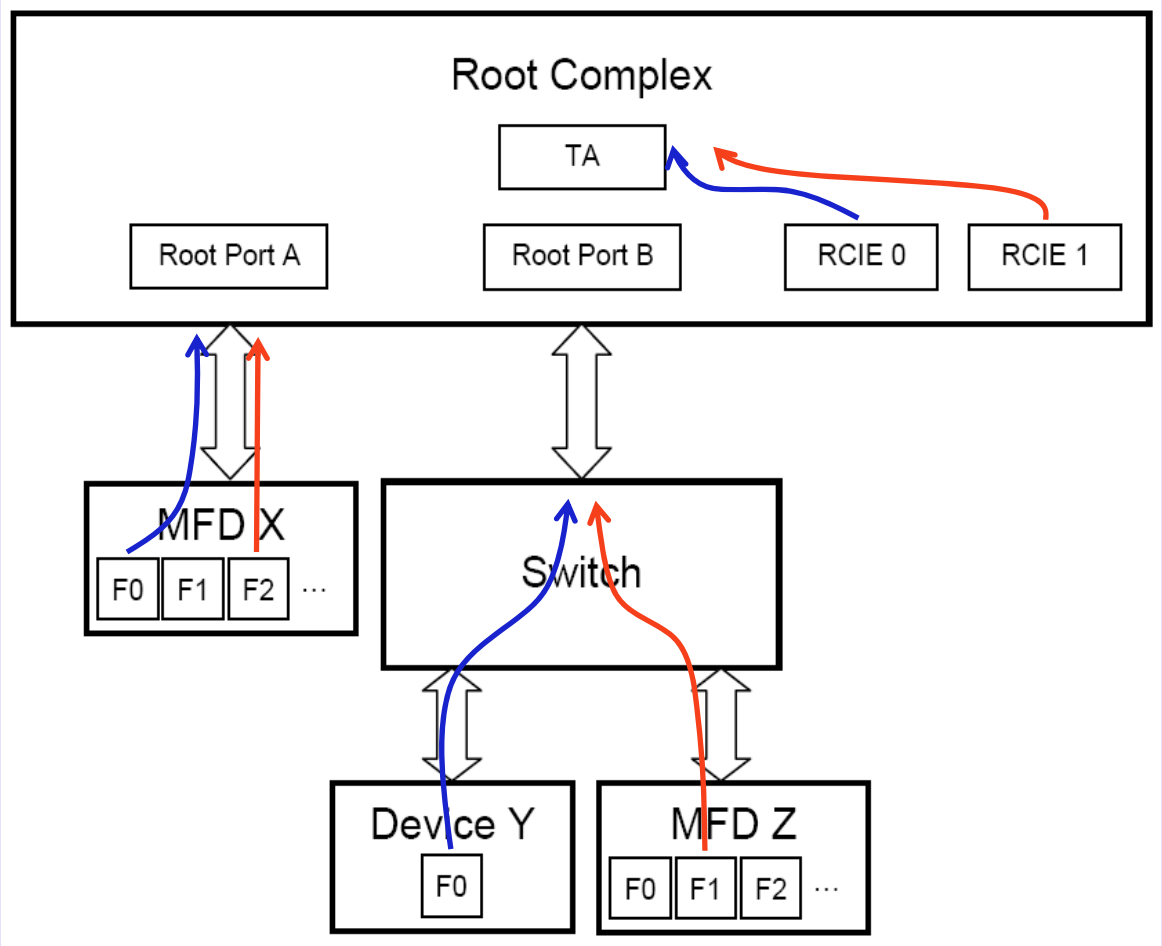
- Maximum theoretical flexibility: All entries are "Y/N"
- Traditional Relaxed Ordering (RO) enables A2 & D2 "Y/N" cases
  - ✓ AtomicOps ECR defines an RO-enabled C2 "Y/N" case
- ID-Based Ordering (IDO) enables A2, B2, C2, & D2 "Y/N" cases

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

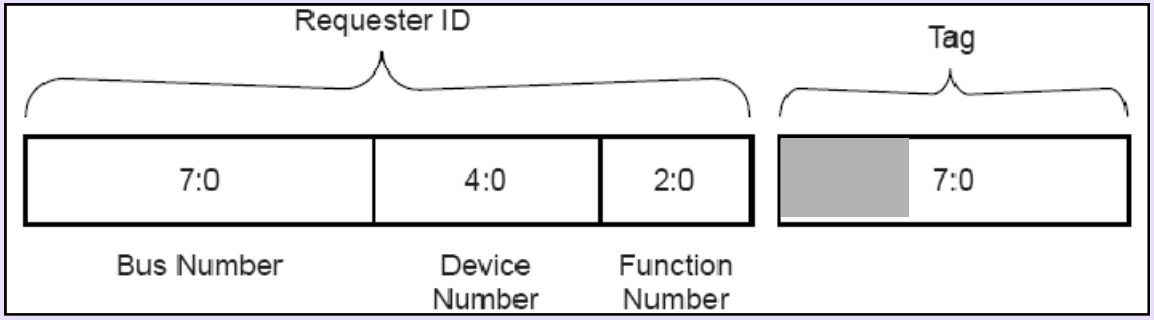
- IDO Use in Conjunction with RO
  - ✓ IDO and RO are orthogonal, though overlapping in some cases
  - ✓ Highly recommended to use both whenever safely possible
  - ✓ RO permits certain TLP passing within same TLP stream, which is never permitted by IDO
  - ✓ For traffic in different TLP streams, IDO 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

Extended Tag Enable Default

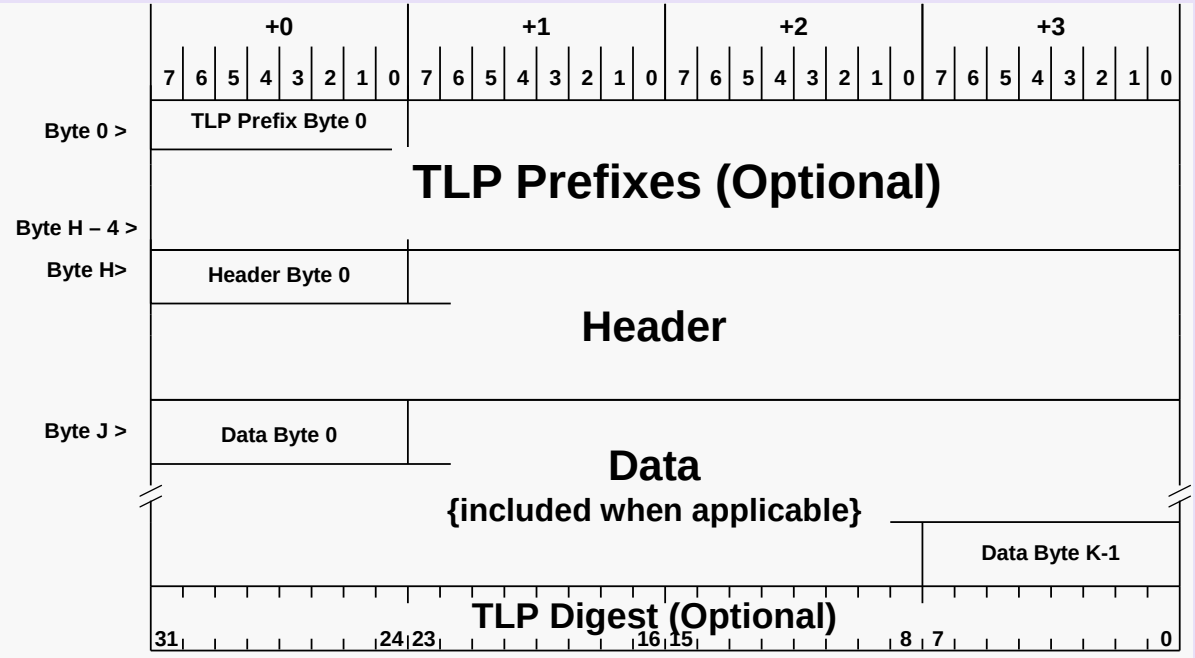
Motivation



- The obligatory 32 tags provided by PCIe per Function are not sufficient to meet the throughput and requirements of emerging applications.
- Extended tags allow up to 256 concurrent requests, but such capability is not enabled by default.
  - ✓ Extended Tag Enable control field default value is 0b (disabled)
- ECR permits a Extended Tag usage to be enabled by default
  - ✓ Extended Tag Enable control field default value is 1b (enabled)
- No Interoperability or Operational issues expected
  - ✓ Mandatory for PCIe Receivers to support receipt of requests that use Extended Tag field
  - ✓ Mandatory for PCIe to PCI-X bridge to support receipt of requests that use Extended Tag Field
- Retains software control of Extended Tag Enable

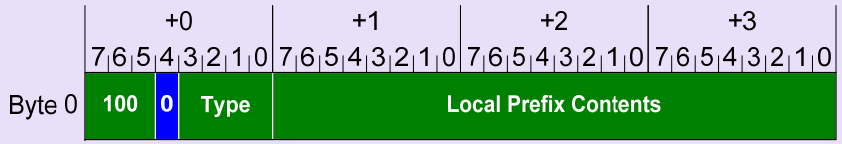
TLP Prefix

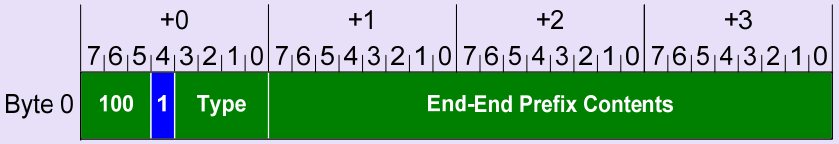
Motivation



- Emerging usage models require increase in header size to carry new information
  - ✓ Example: Multi-Root IOV, Extended TPH
- TLP Prefix mechanism extends the header sizes by adding DWORDs to the front of headers

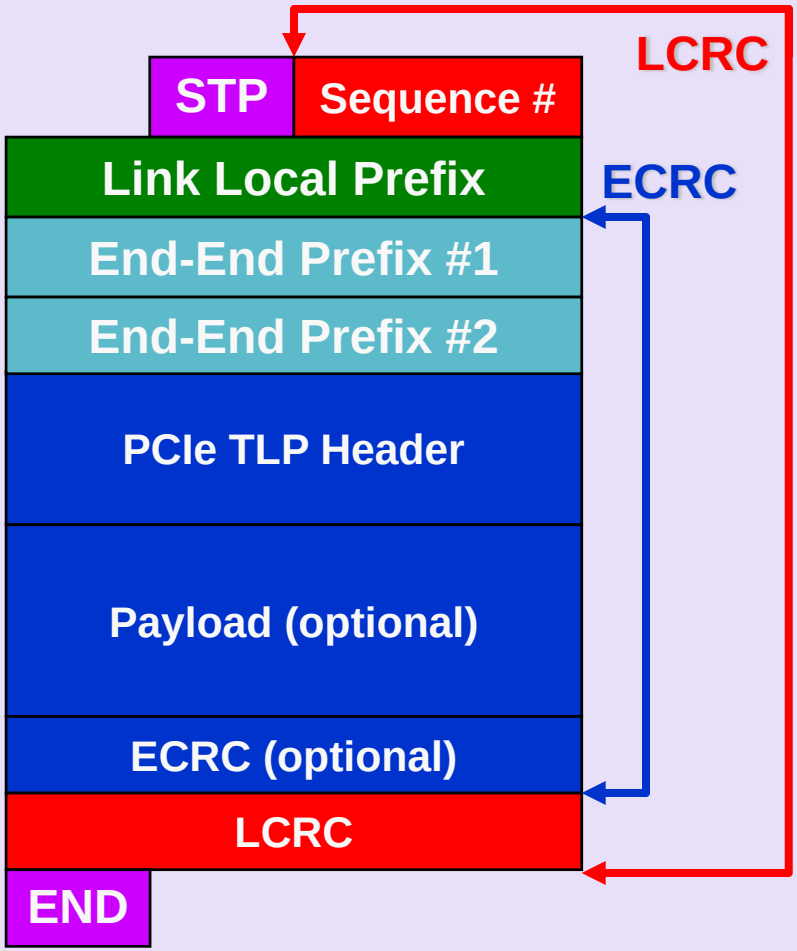
Prefix Encoding





- Base TLP Prefix Size – 1 DW
  - ✓ Appended to TLP headers
- TLP Prefixes can stacked or repeated
  - ✓ More than one TLP Prefix supported
- 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
- Fmt field grows to 3 bits
  - ✓ New error behavior defined
  - ✓ Undefined Fmt and/or Type values results in Malformed TLP
  - ✓ “Extended Fmt Field Supported” capability bit indicates support for 3 bit Fmt
    - – Support is recommended for all components (independent of Prefix support)

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 are unordered
    - affects ECRC but does not affect meaning
  - ✓ Repeated End-End Prefix sequence must be ordered
    - e.g. 1st Extended TPH vs. 2nd Extended TPH attribute
    - meaning of this is defined by each End-End Prefix

Internal Error Reporting

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
- Corrected 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.

AER 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
- Corrected Internal Errors
  - ✓ Add Corrected 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 Recording

- 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