I/O Virtualization Architecture Overview

Michael Krause (HP, co-chair)
Renato Recio (IBM, co-chair)

Disclaimer

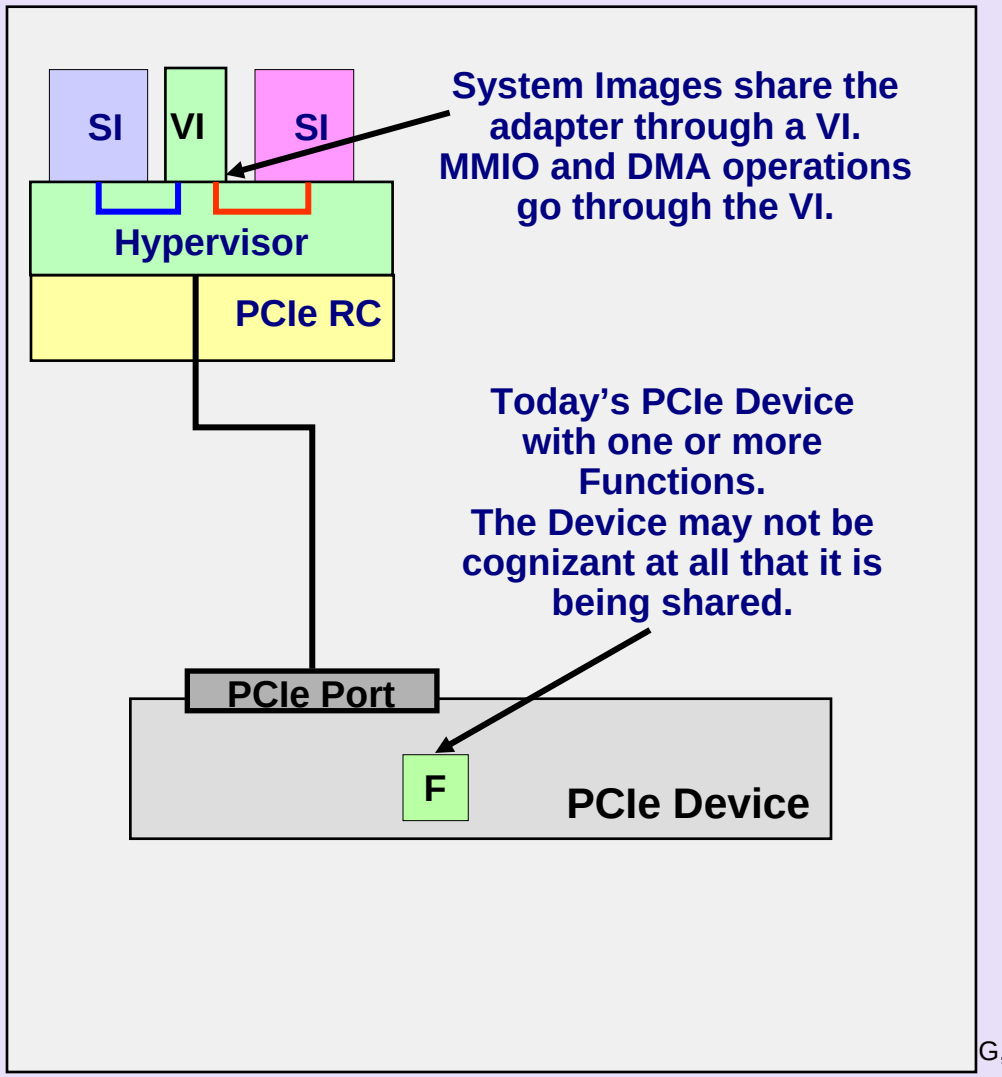
- NOTE: Information in this presentation refers to specifications still in the development process. This presentation reflects the current thinking of the workgroup, but all material is subject to change before the specifications are released.

Agenda

- Virtualization Overview
  - ✓ Terminology
  - ✓ Single-Root (SR) IOV
  - ✓ Multi-Root (MR) IOV
  - ✓ Address Translation Services (ATS)
- Specification Status

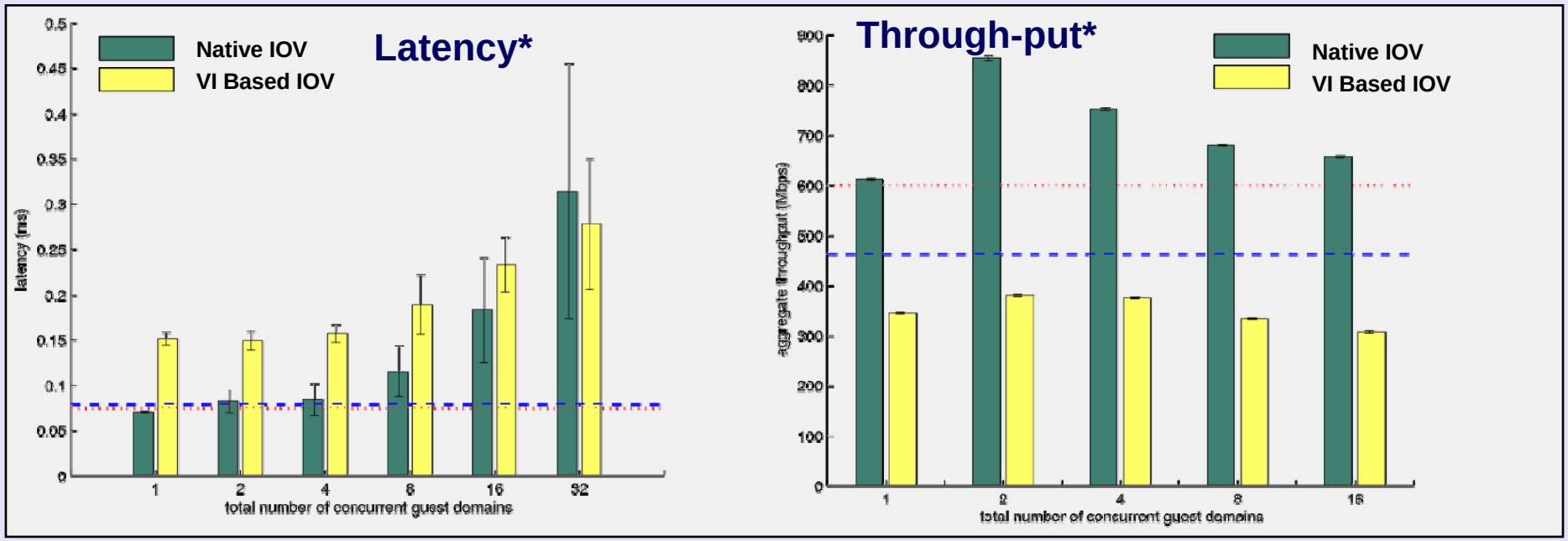
Today's Approach to IOV

VI Based PCI Device Sharing Example



- Virtualization Intermediaries (VI) are used to safely share IO.
- Under this approach:
  - ✓ 1 or more System Images (SI) share the PCI device via a VI.
  - ✓ Virtualization enablers are not needed in either the Root Complex (RC) or PCIe® Device.
  - ✓ The VI is involved in all IO transactions and performs all IO Virtualization Functions.

Performance of Today's IOV

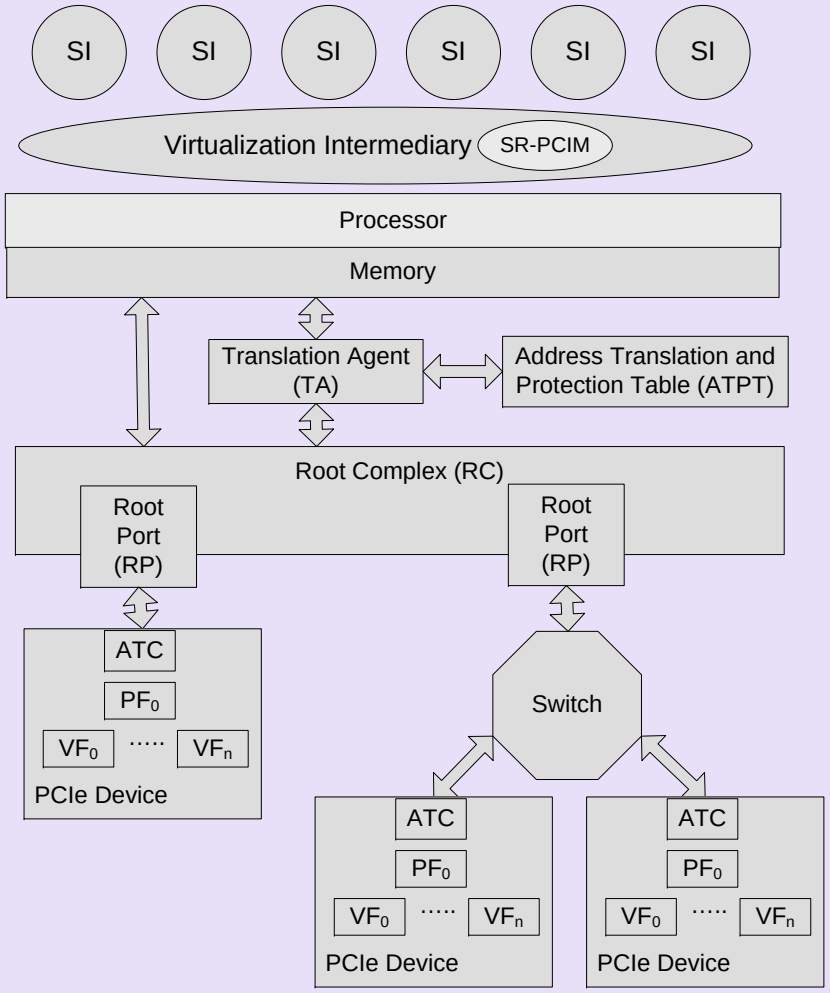


- VI based IOV adds path length on every IO operation.
- Native IOV significantly improves performance
  - ✓ Doubles throughput and reduces latency by up to half.
- Several factors are increasing the virtualization use (e.g. more cores per socket, customer simplification requirements, …).
  - ✓ Making Native IOV even more important in the future.

***Source:** Self-Virtualized I/O: High Performance, Scalable I/O Virtualization in Multi-core Systems; R. Himanshu, I. Ganev, K. Schwan - Georgia Tech and J. Xenidis - IBM

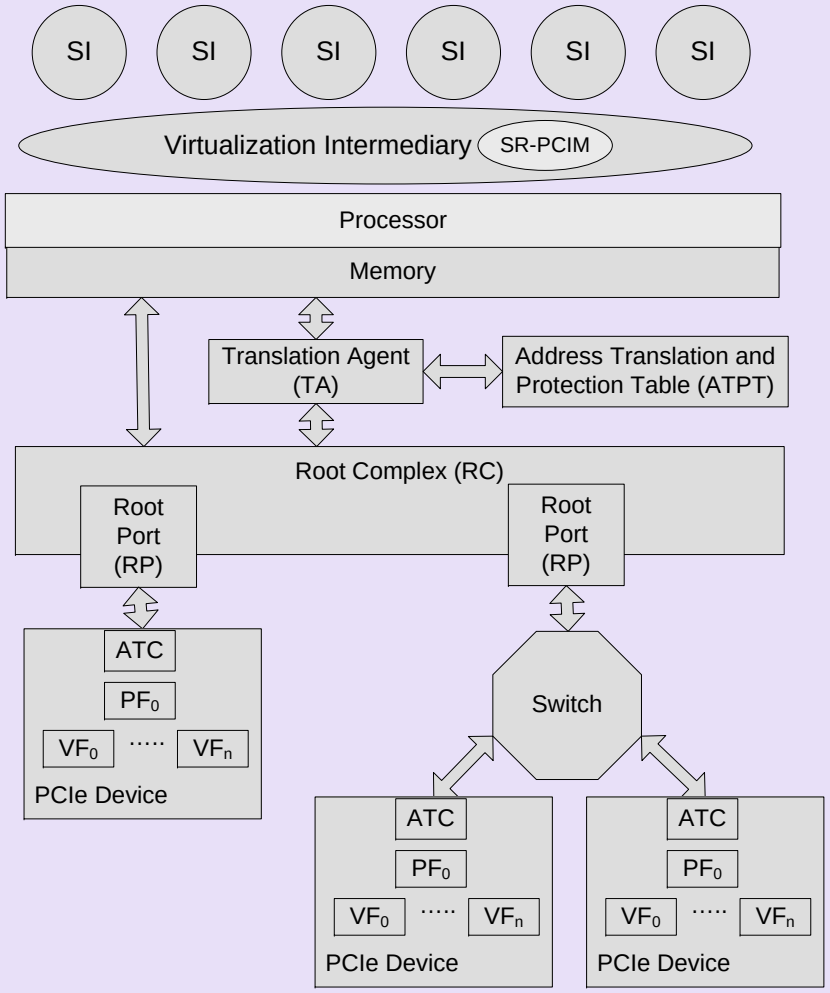
Terminology

Terminology



- System Image (SI)
  - ✓ S/W, e.g. a guest OS, to which virtual and physical devices can be assigned
- Virtualization Intermediary (VI)
  - ✓ Resource management and event handling component
  - ✓ Allocates resources and isolates resources to each SI
- Translation Agent (TA)
  - ✓ Translates PCI Addresses to platform physical addresses
    - May also provide interrupt remapping for MSI / MSI-X interrupts
- ATPT
  - ✓ Contains the set of address translations accessed by a TA to process PCIe requests – DMA Read, DMA Write, or interrupt requests.
  - ✓ Address translations typically managed on a per function identifier (RID) basis

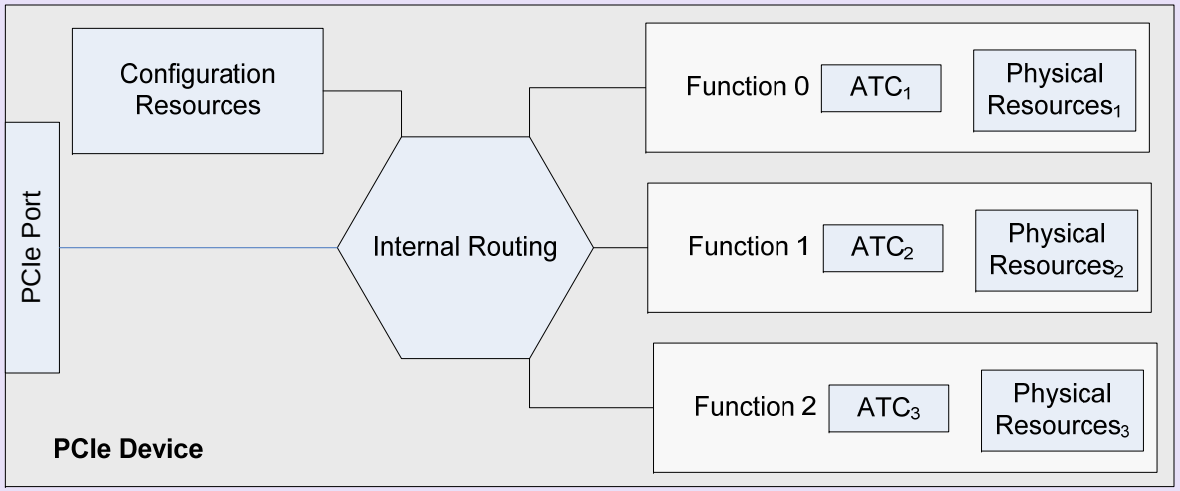
Terminology (cont.)



- PCI Manager (PCIM)
  - ✓ Two flavors:
    - Single Root (SR-PCIM)
    - Multi-Root (MR-PCIM)
  - ✓ Responsible for:
    - Configuration of the SR-IOV and MR-IOV capability
    - Management of Physical Functions (PF) and Virtual Functions (VF)
    - Processing of associated error events and overall device controls such as power management and hot-plug services.
  - ✓ Variety of implementation options
    - Firmware, VI, OS, driver, etc.
    - Outside the scope of the PCI-SIG®

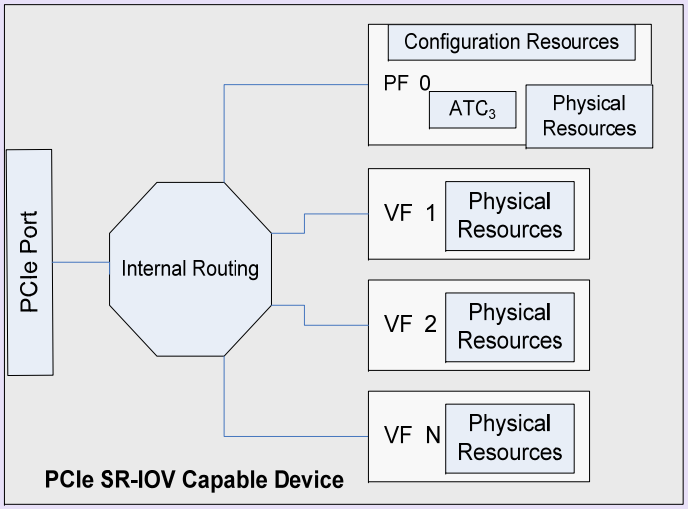
Single Root I/O Virtualization (SR-IOV)

Example Multi-Function I/O Device



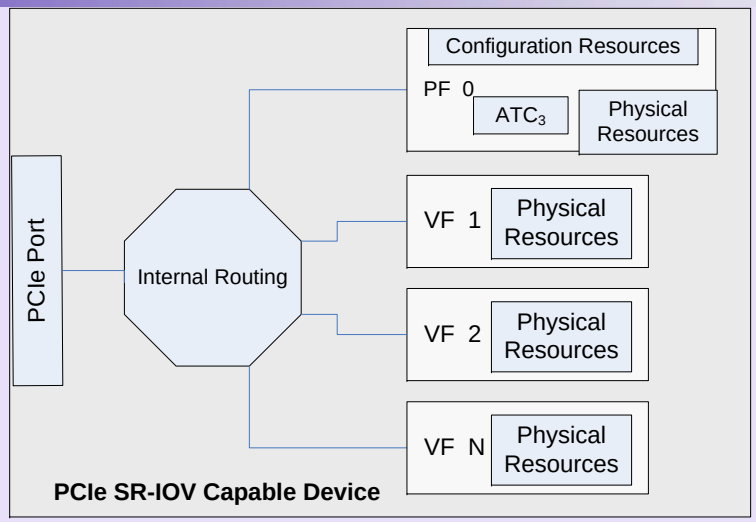
- The PCIe Device shares a common PCIe Link. Per the *PCI Express® Base Specification*, the link and PCIe functionality shared by all Functions is managed through Function 0.
  - ✓ While this figure illustrates only three functions, with the use of the Alternative Routing Identifier (ARI) capability, a PCIe Device can support up to 256 Functions.
  - ✓ All Functions use a single Bus Number captured through the PCI enumeration process.
- In this example, each PCIe Function supports the ATS capability and therefore has an associated ATC to manage ATS obtained translated addresses.
- Each PCIe Function has a set of unique physical resources including a separate configuration space and BAR.
- Each PCIe Function can be assigned to a SI. To prevent one SI from impacting another, all PCIe configuration operations should be intercepted and processed by the VI.

Physical Function (PF)



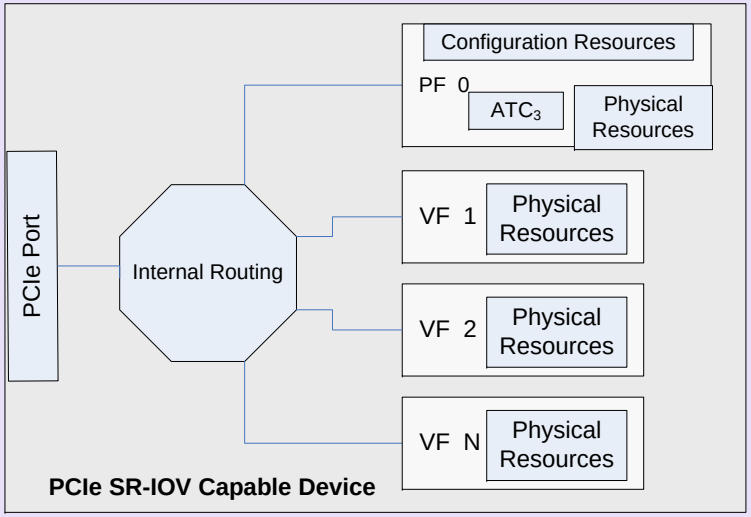
- A PF is a PCIe Function (per the *PCI Express Base Specification*) that supports the SR-IOV capability and is accessible to SR-PCIM, a VI, or a SI.
- Default maximum number of PF is 8. Maximum number of PF with ARI is 256
- A PF can only be associated with the Device's captured Bus Number
- Many potential PF usage models exist
  - Common usage model is to use the PF to bootstrap the device or platform strictly under the control of a VI.
  - Once the SR-IOV capability is configured enabling VF to be assigned to individual SI, the PF takes on a more supervisory role.
    - For example, the PF can be used to manage device-specific functionality such as internal resource allocation to each VF, VF arbitration to shared resources such as the PCIe link or the function-specific link (e.g. a network or storage link), etc. These policy, management, and resource allocation operations are outside the scope of this specification.

Virtual Function (VF)



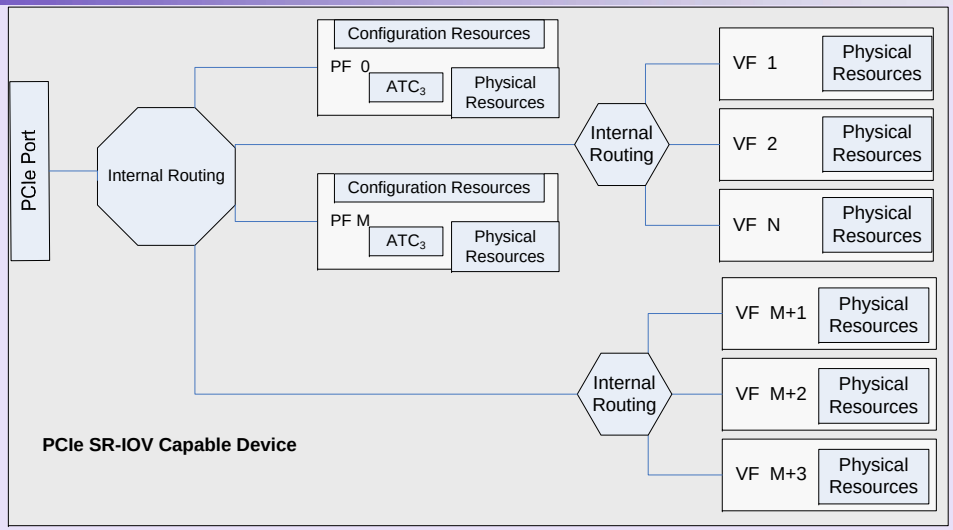
- Virtual Function (VF) – A VF is a “light-weight” PCIe Function that is directly accessible by a SI.
  - ✓ Minimally, resources associated with the main data movement of the function are available to the SI.
  - ✓ A VF can be serially shared by different SI, i.e. a VF can be assigned to one SI and then reset and assigned to another SI.
  - ✓ A VF can be optionally migrated from one PF to another PF. The migration process itself is outside the scope of this specification but is facilitated through configuration controls defined within the specifications.
  - ✓ All VF associated with a PF must be the same device type as the PF, e.g. the same network device type or the same storage device type.

Additional PF and VF Considerations



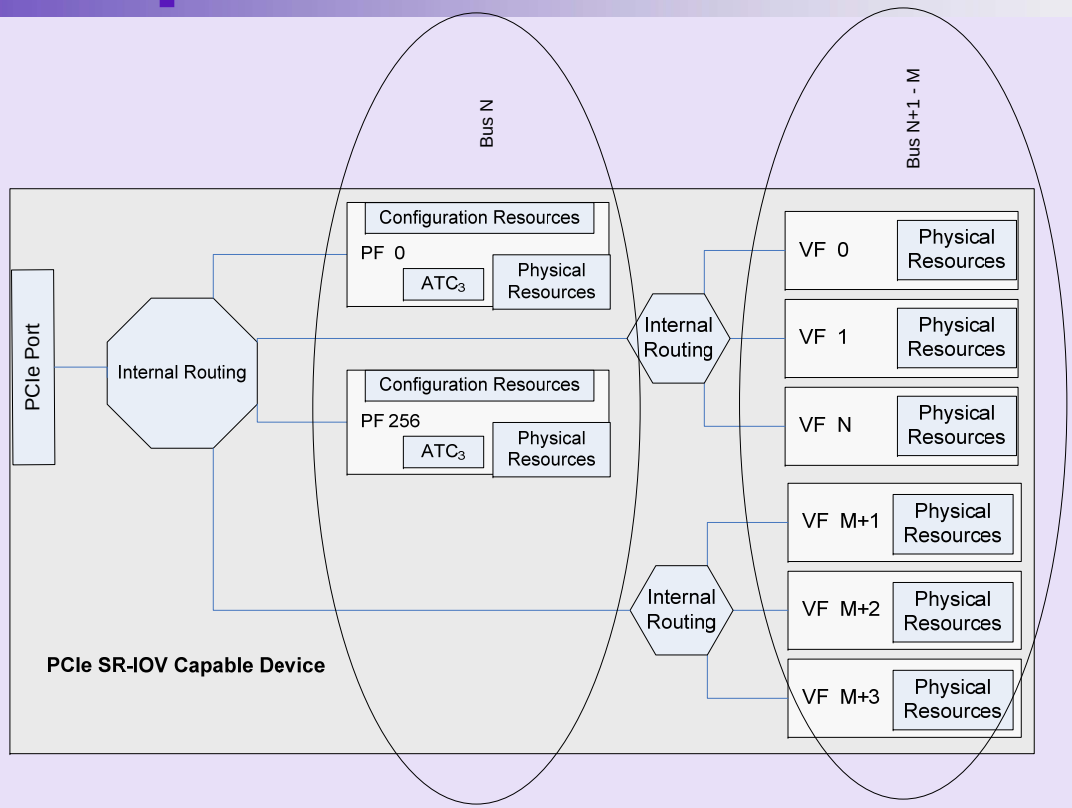
- Each Function, PF, and VF is assigned a unique Requester ID (RID).
  - ✓ A RID is constructed per default mechanism within the *PCI Express Base Specification* or through the mechanism enabled via the ARI capability.
- All PCIe and SR-IOV configuration access is assumed to be through a trusted software component such as a VI or SR-PCIM.
- Each VF contains a non-shared set of physical resources required to deliver function-specific services, e.g. resources such as work queues, data buffers, etc. These resources can be directly accessed by a SI without requiring VI or SR-PCIM intervention.
- One or more VF may be assigned to each SI. Assignment policies are outside the scope of this specification.
- While this example illustrates a single ATC within the PF, the specifications do not preclude an implementation from supporting an ATC per VF within the device.

Multi-PF SR-IOV Device



- Each PF can be assigned zero or more VF. The number of VF per PF is not required to be identical for all PF within the device.
- The ARI capability enables Functions to be assigned to Function Groups and defines how Function Group arbitration can be configured.
  - ✓ PF and VF can be assigned to Function Groups and take advantage of the associated arbitration capabilities. Within each Function Group though, arbitration remains implementation-specific.
- Internal routing between PF and VF is implementation specific.
- For some usage models, all PF may be the same device type, e.g. all PF deliver the same network device or all deliver the same storage device functionality. For other usage models, each PF may be represent a different device type, e.g. in the above figure, one PF might represent a network device while another represents an encryption device.
  - ✓ In situations where there is a usage model dependency between device types, such as for each VF that is a network device type, each SI also requires a VF that is an encryption device type, the SR-IOV capability provides a method to indicate these dependencies.
  - ✓ The policies used to construct these dependencies as well as assign dependent sets of VF to a given SI are outside the scope of the SR-IOV specification.

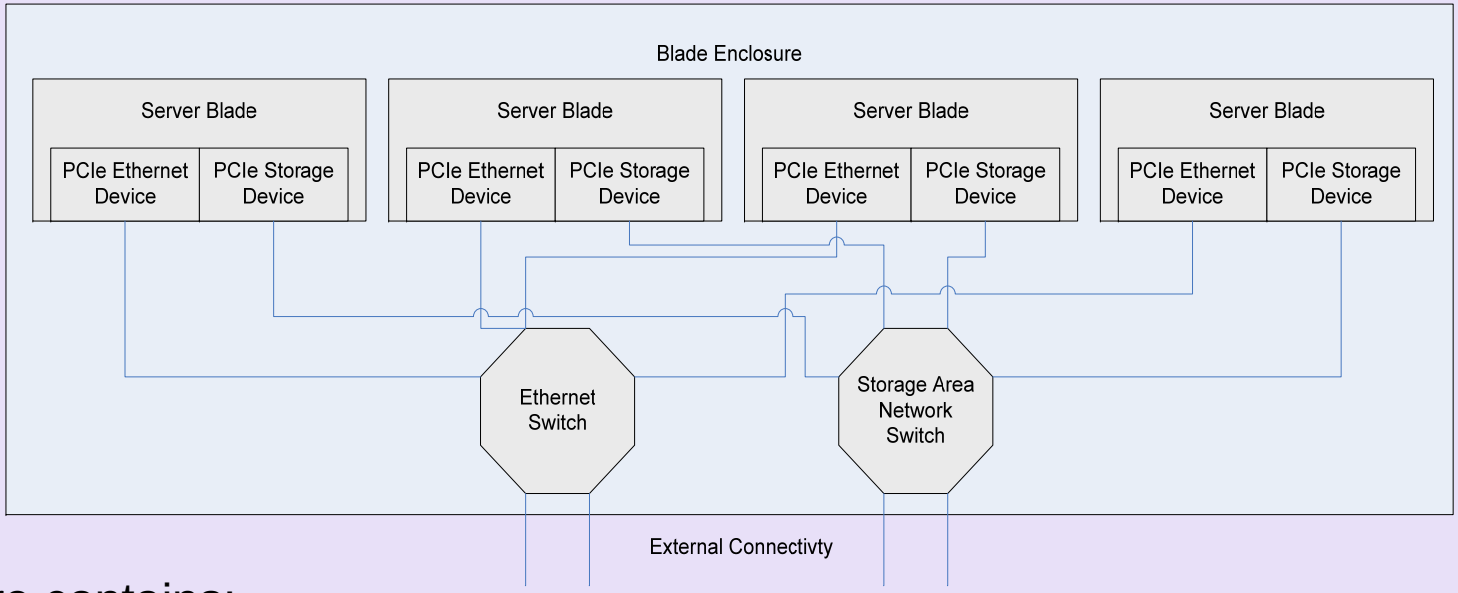
Multiple Bus Number Usage



- A PF can only be associated with the Device's captured Bus Number.
- A VF can be associated with any Bus Number within the Device's Bus Number range – the captured Bus Number plus any additional Bus Numbers configured by software.
  - ✓ Use of multiple Bus Numbers enables a device to support a very large number of VF – up to the size of the RID space minus the bits used to identify intervening busses.
  - ✓ If software does not configure sufficient additional Bus Numbers, then the VF implemented for the additional Bus Numbers may not be visible.

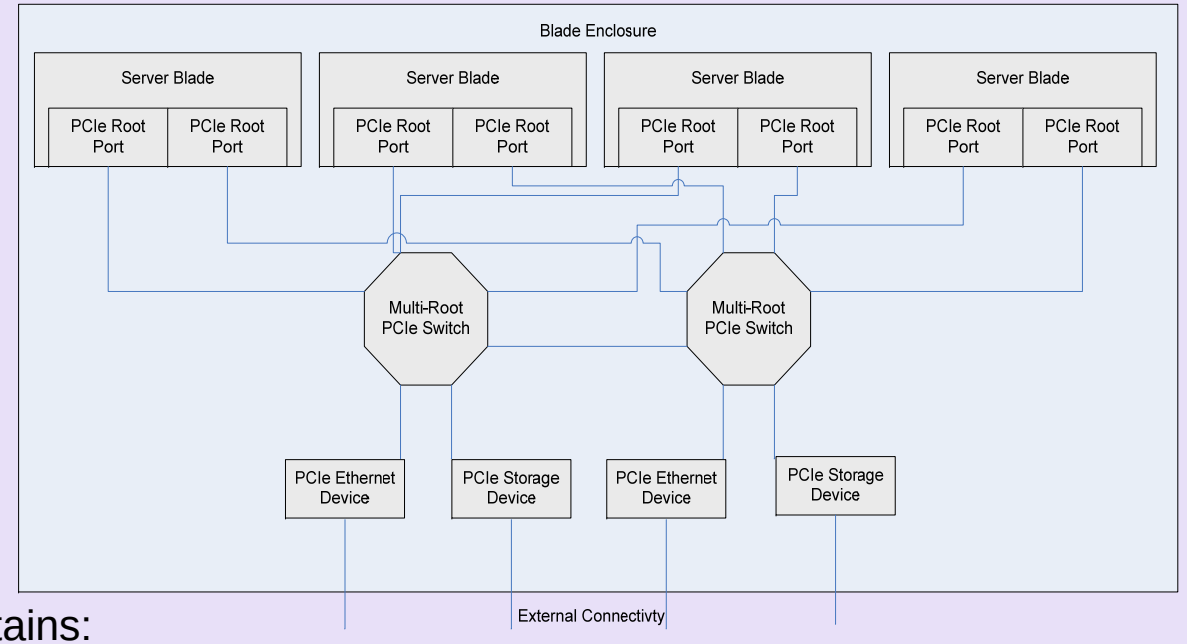
Multi-Root (MR) IOV

Example Blade Enclosure



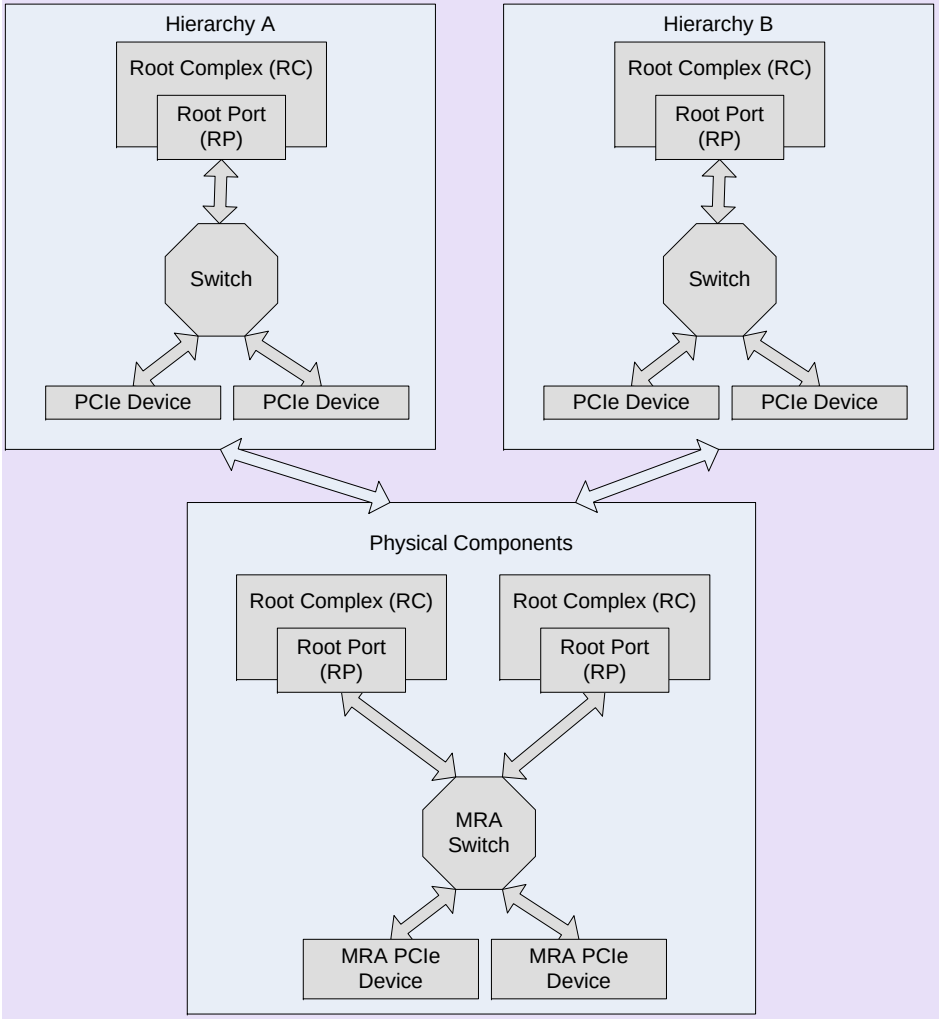
- Enclosure contains:
  - ✓ 4 server blades with 2 PCIe Devices each for a total of 8 Devices
    - – Each server blade contains an Ethernet and a storage device (e.g. Fibre Channel)
    - – Assumes point-to-point connectivity to PCIe RP
  - ✓ Enclosure contains 2 (4 in the case of a High Availability solution) “external” switches
- Each I/O device and switch port is typically provisioned to enable any I/O device to operate at full bandwidth.

MR-IOV Blade Enclosure



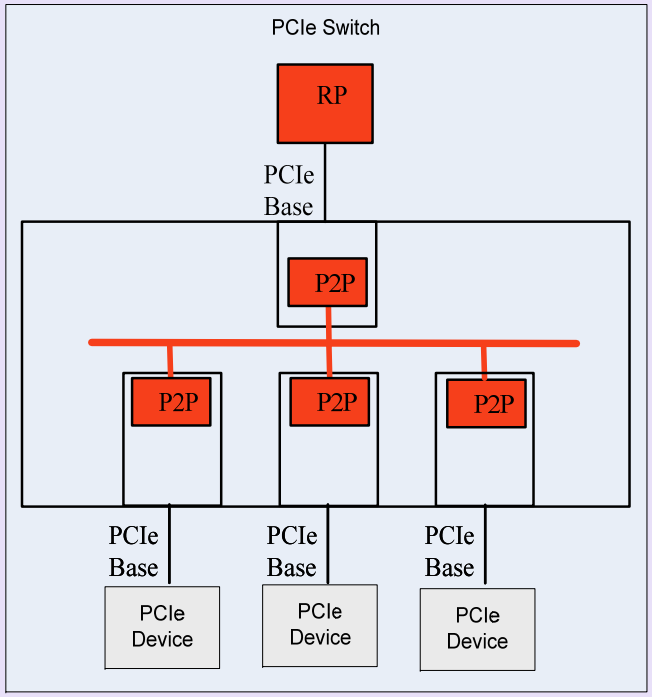
- Enclosure contains:
  - ✓ 4 server blades with no PCIe Devices
  - ✓ 4 Multi-Root Aware (MRA) PCIe Devices – 2 Ethernet and 2 Storage
    - – Can horizontally scale to increase aggregate bandwidth based on workload needs
  - ✓ Enclosure contains 2 MRA PCIe Switches
    - – No external switches required
- I/O Device is shared
  - ✓ Can be dynamically provisioned to meet workload requirements

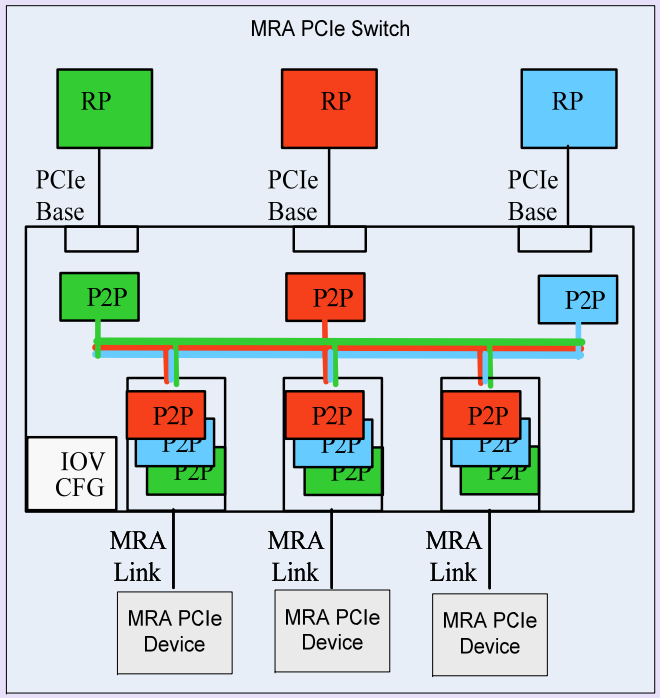
Goal of MR-IOV



- PCI components underneath each RP must be virtualized and logically overlaid on the MRA PCIe Switches and Devices
- The virtualized PCI components are referred to as a Virtual Hierarchy (VH). A VH has the following attributes:
  - ✓ Each VH must contain at least one PCIe Switch.
    - The PCIe Switch will be a virtualized component implemented over of a MRA Switch.
    - The PCIe Switch functionality and semantics are per the *PCI Express Base Specification*.
  - ✓ Each VH may contain any mix of PCIe Devices, MRA PCIe Devices, or PCIe to PCI / PCI-X™ Bridges

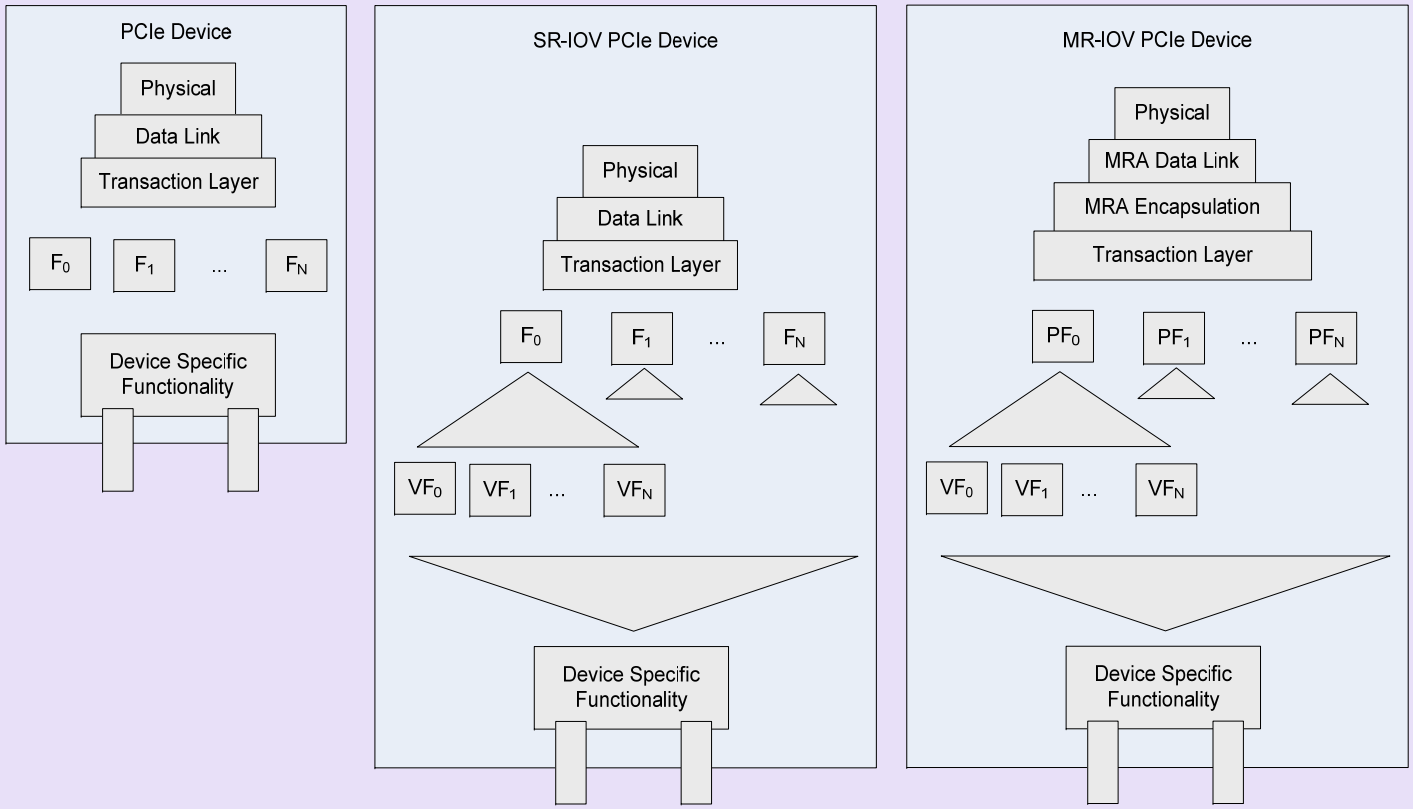
Multi-Root Aware (MRA) Switch



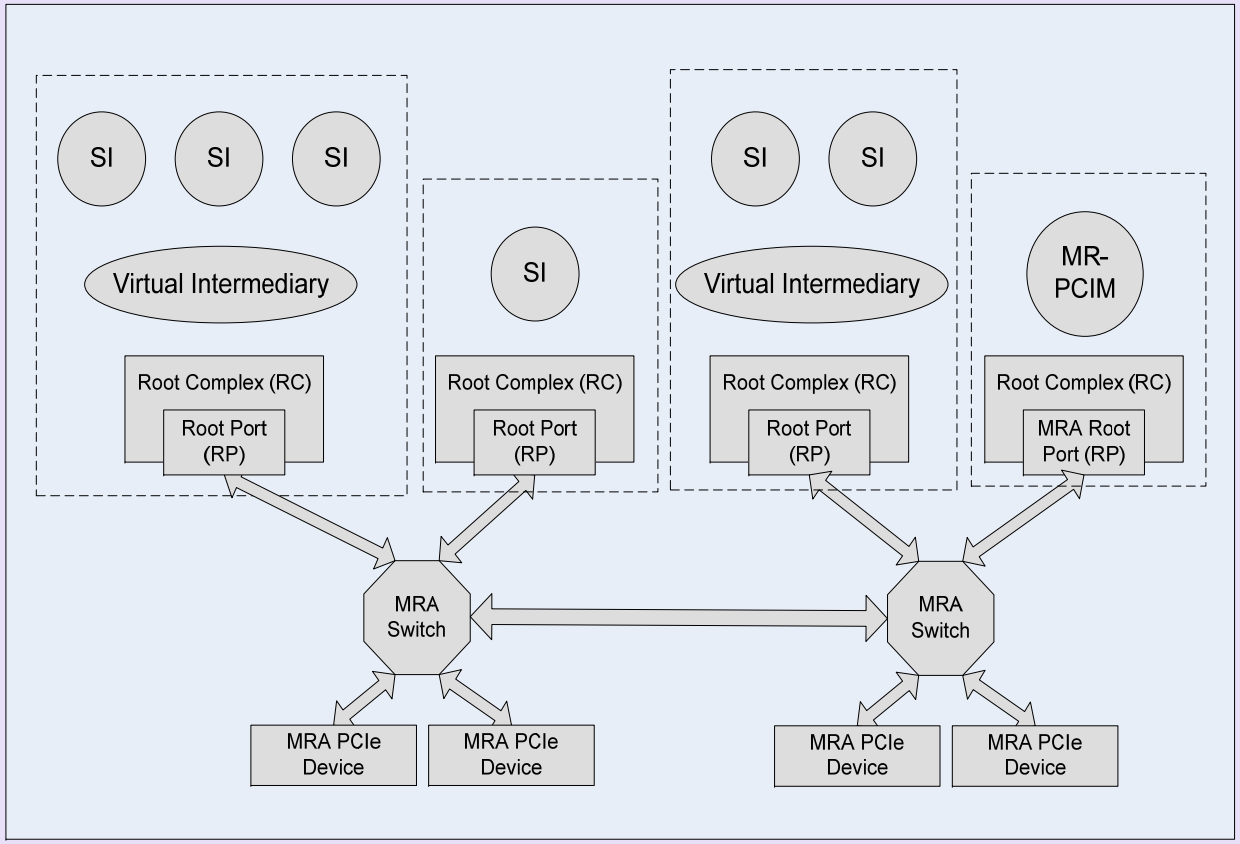


- The MR-IOV topology must contain at least one MRA PCIe Switch.
  - ✓ Multiple MRA PCIe Switches can be provisioned and interconnected in a variety of topologies – tree, fat-tree, star, mesh, etc.
  - ✓ A MRA PCIe Switch typically contains two or more upstream Ports. In a single-stage or a switch at the top of the topology, each upstream Port connects to a RP which acts as the root of the VH.

Comparison of Device Types



MR-PCIM



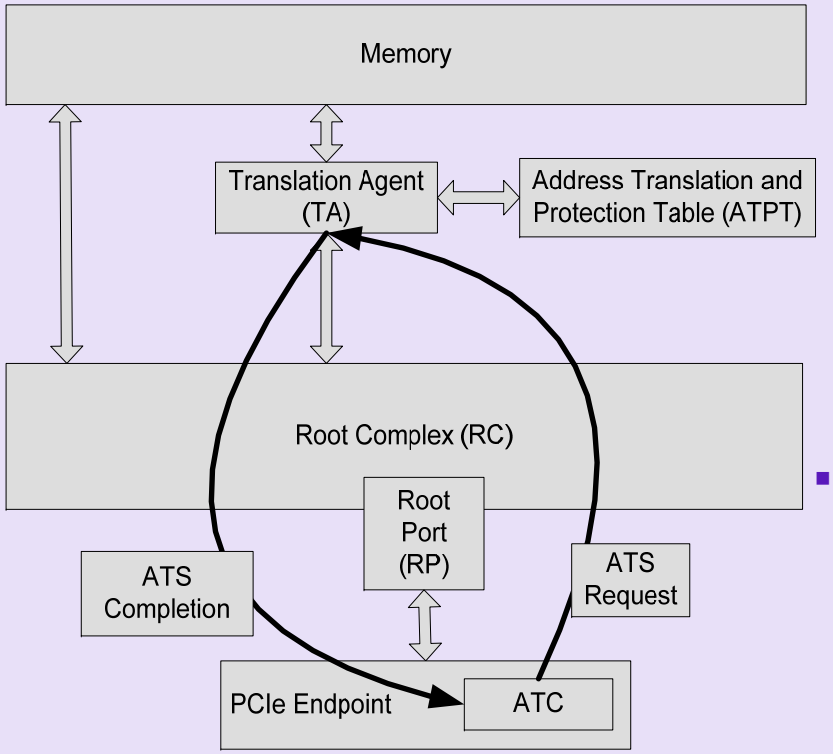
- MR-PCIM
  - ✓ Responsible for MR configuration and event management
    - – Creation VH, MR resource assignment, MR hot-plug, RESET, error handling, etc.
  - ✓ Can be implemented anywhere – above a RC, sideband off switch, etc.

Address Translation Services (ATS)

DMA Address Translation

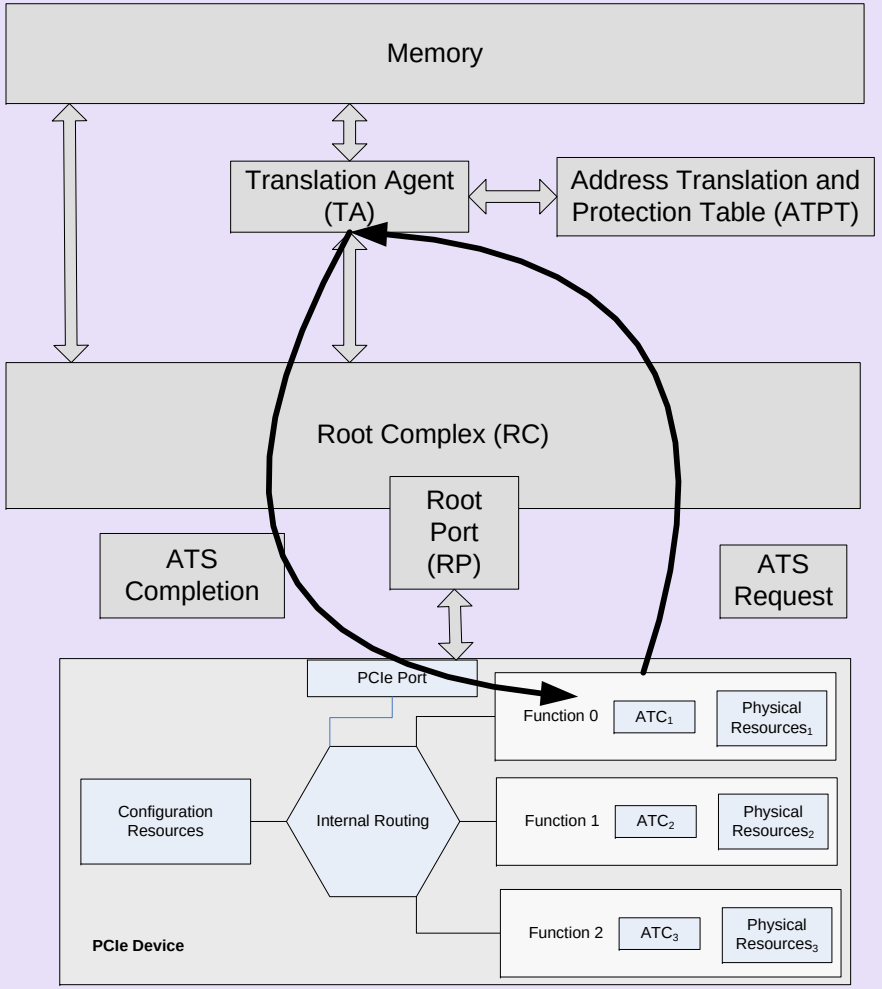
- Translation Agent (TA) performs:
  - ✓ Address translation and an access right validation per Device DMA request
  - ✓ One or more accesses into the ATPT to acquire translation
    - – May need to walk multiple table entries to acquire platform physical address
- Potential Issues with DMA Translation
  - ✓ Increased latency of accesses
    - – Might need one or two accesses to find address of tree associated with a BDF
    - – Might need 3 or 4 accesses to walk the tree
  - ✓ Translation caches (ATC or IOTLB) will be necessary to reduce overhead.
  - ✓ Caches may not provide good behavior if not sized correctly
    - – Only two possibilities for sizing caches: too large or too small
  - ✓ “Untimely” latency may cause issues with isochronous devices

ATS to the “rescue”



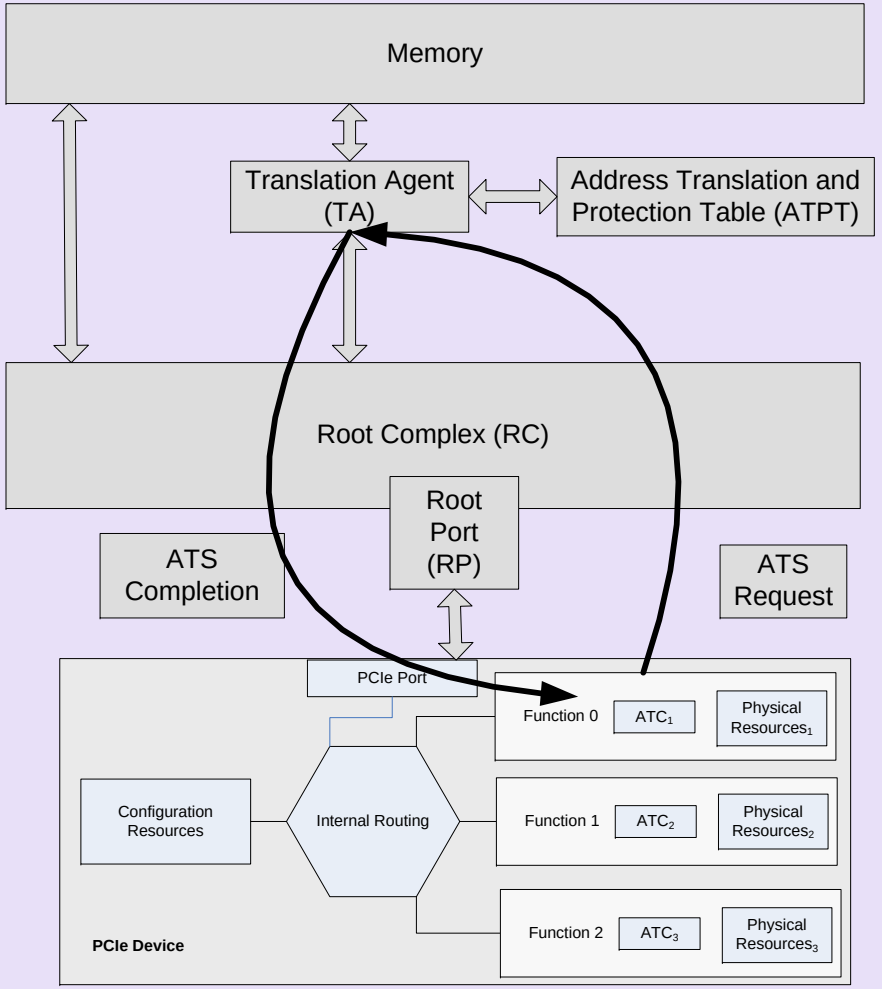
- ATS attempts to mitigate the impact of DMA translation by providing ways for Devices to participate in translation cache management
  - ✓ Device can maintain their own cache of translations – an “Address Translation Cache” (ATC)
  - ✓ TA provides table-walking services to device to avoid excess bus traffic – also means that translation table format is uniform in a system
- Device manages its ATC using its intimate knowledge of future access pattern
  - ✓ Look-ahead for isochronous devices to avoid “untimely” table walk latencies.
  - ✓ High-load devices (graphics) don’t thrash ATC in TA.
  - ✓ Application specific caching in devices – ring buffer
  - ✓ Enable peer-to-peer in virtualized bus

ATS Request



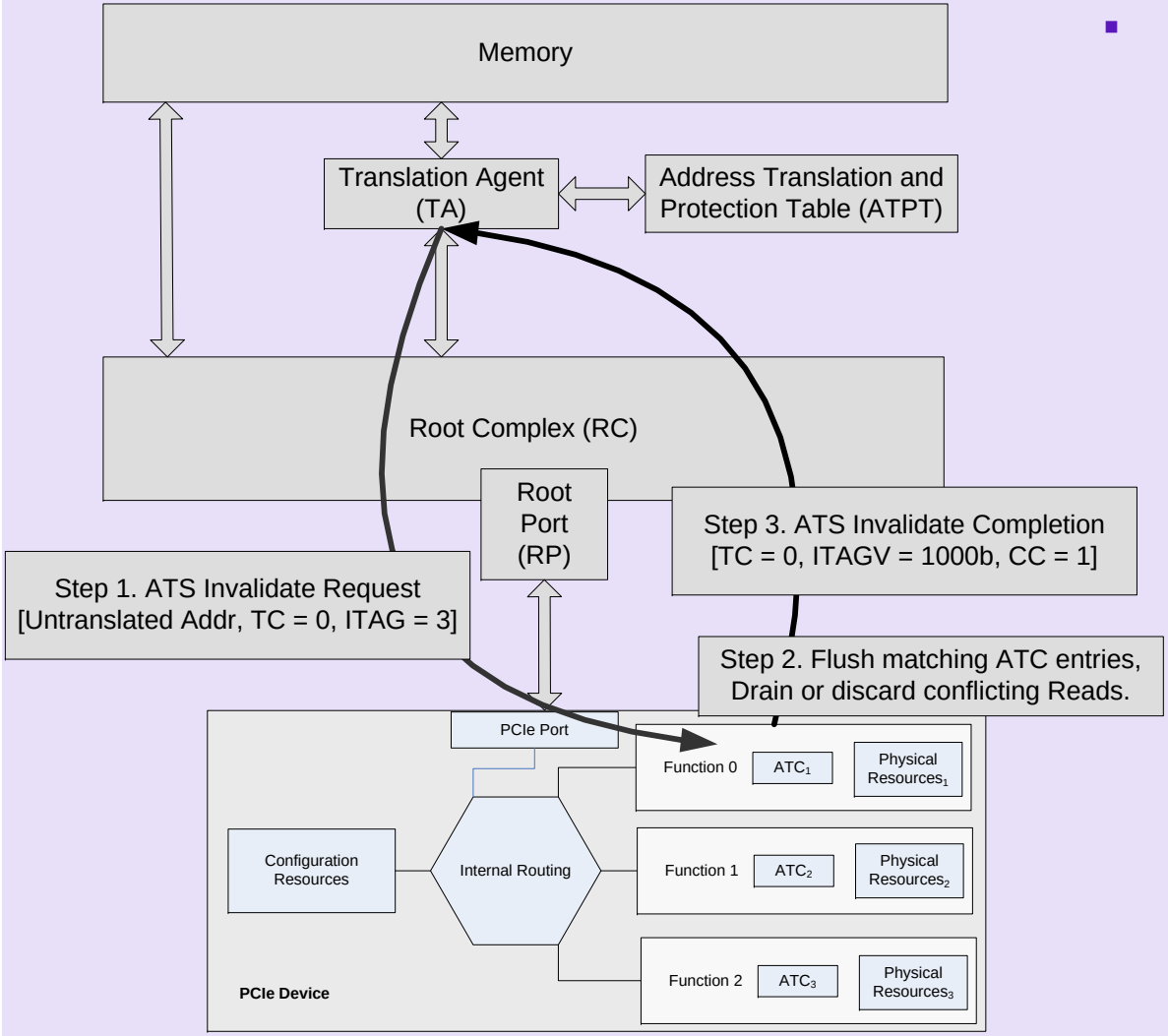
- ATS Request
  - ✓ Used by the Endpoint to request translated address
    - Address can be 32-bit or 64-bit
  - ✓ Requests can flow on any TC
  - ✓ Multiple ATS Requests may be pipelined
- Function Responsibilities
  - ✓ Function implements an ATC
    - ATC is a cache of translated addresses
    - ATC may be implemented independent of IOV support
  - ✓ Function must only populate cache via ATS protocol
  - ✓ Function must not allow ATC to be modified or accessed by software (local or remote) to prevent data leakage
  - ✓ For each ATS Request, there will be a corresponding ATS Completion.

ATS Completion



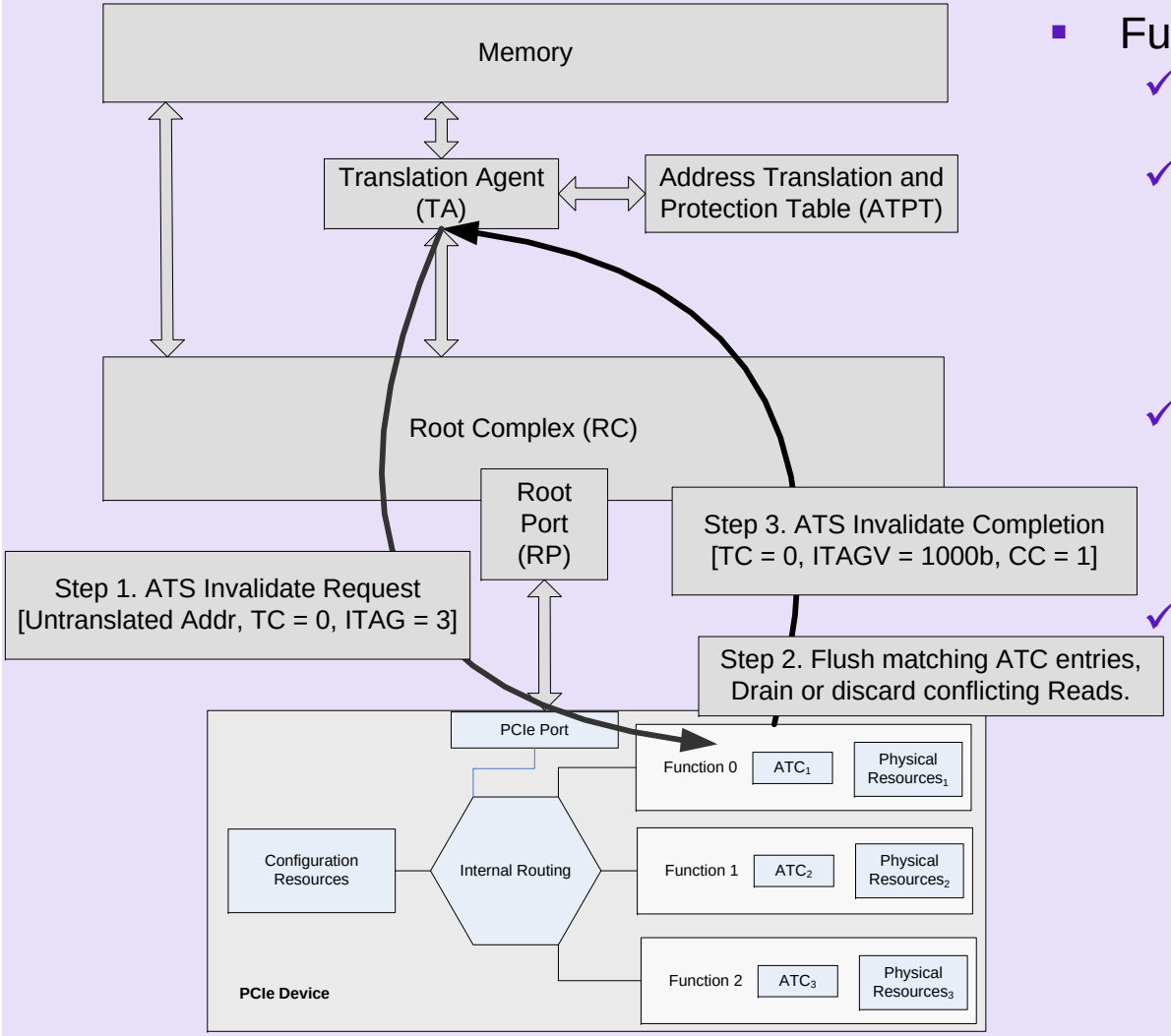
- ATS Completion
  - ✓ Used to return either:
    - A translated address which may cover a range of pages
    - A translation failure
  - ✓ An ATS Completion must be generated for each ATS Request
    - Multiple ATS Completions may be in-flight
    - ATS Completions may be returned in any order relative to the ATS Requests
  - ✓ An ATS Completion must use the same TC as the associated ATS Request
  - ✓ Completions must be sent using the same TC as the ATS Request
  - ✓ ATS Completions return access rights
    - Read-only, Write-only, Read / Write
      - Read = Write = 0 indicates the there is a hole in the translation space
    - A Function can only issue subsequent TLP Requests that match the access rights on the associated address range
- Function responsibilities:
  - ✓ Be capable of sinking all ATS Completions without causing backpressure on the PCIe Link
  - ✓ Be able to discard ATS Completions that are stale
    - If an ATS Invalidation is received prior to the ATS Completion, then the subsequent Completion is stale and should not be used

ATS Invalidation



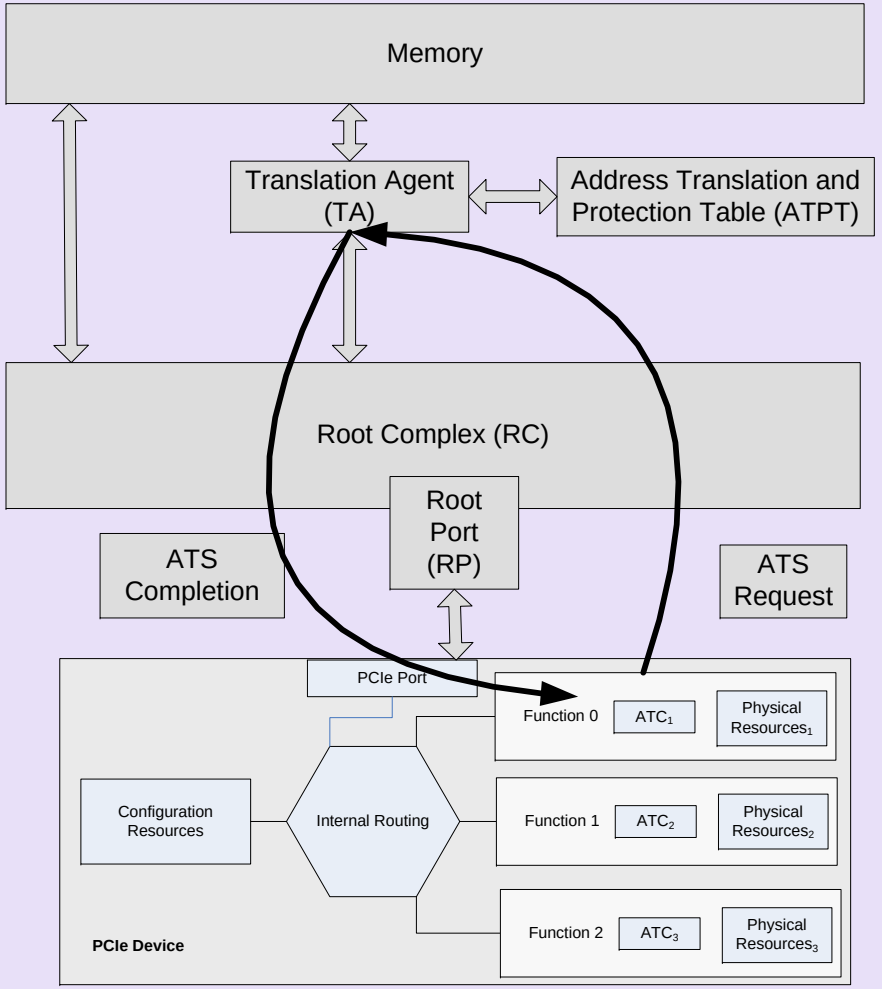
- ATS Invalidation Request
  - ✓ Used to maintain consistency between the ATPT Translation Agent (TA) and the ATC
  - ✓ ATS Invalidation Requests invalidate translations within an ATC
    - A Request may cover a range of address
  - ✓ ATS Invalidation Request may be issued at any time
    - When a translation is changed within the ATPT, the TA issues an ATS Invalidation Request to the impacted ATC
  - ✓ ATS Invalidation Requests may be issued on any TC
    - The TA does not track the TC of prior ATS Request(s)
  - ✓ Each ATS Invalidation Request has an ITAG which uniquely identifies the Request
    - The associated Invalidation Completion returns this ITAG to enable the TA to quickly associate the Completion with the prior Request

ATS Invalidation (cont.)



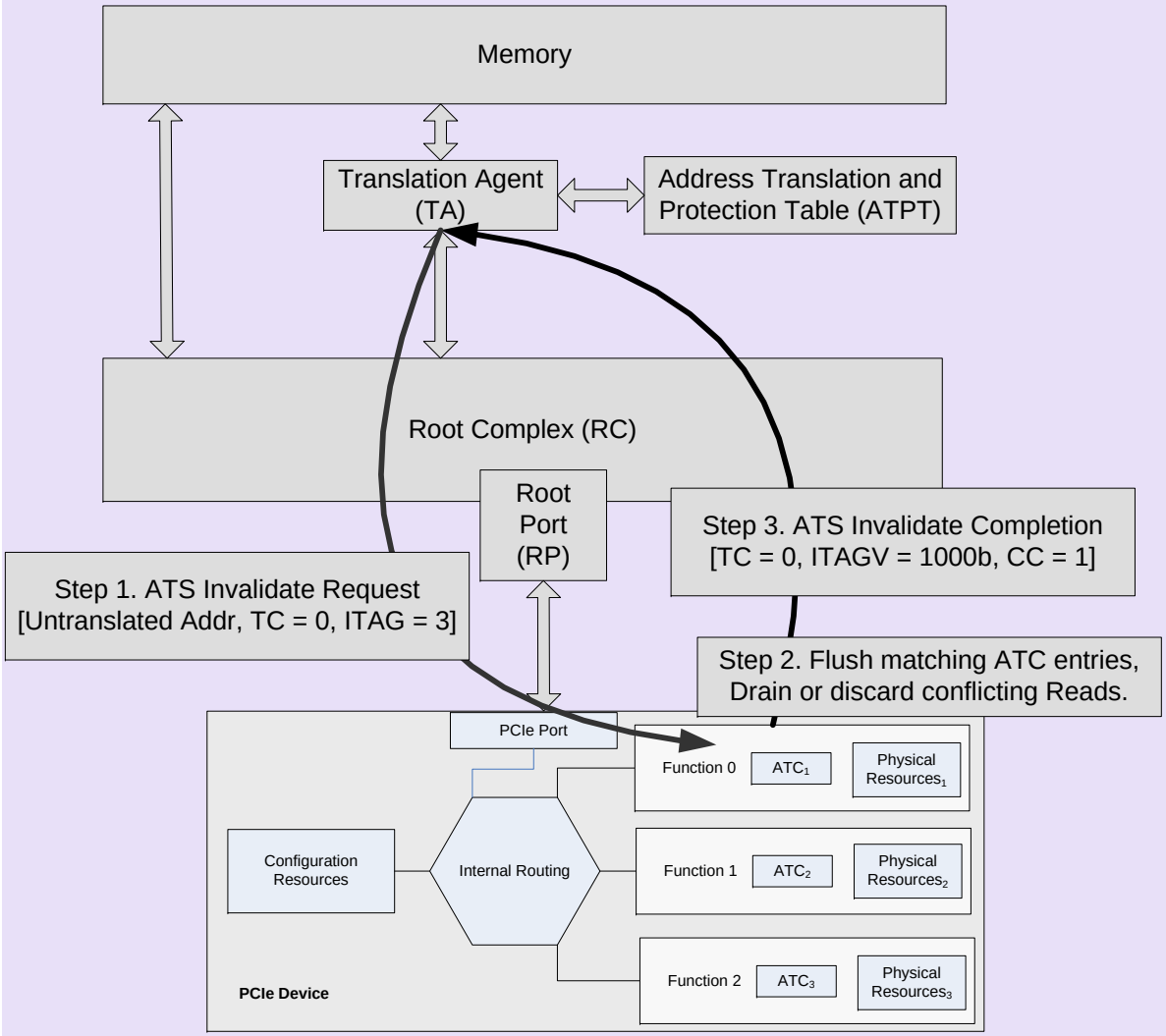
- Function responsibilities:
  - ✓ Must return an ATS Invalidation Completion for each Request
  - ✓ Must not indicate an invalidation has completed until all outstanding Read Requests that reference the associated translated address have been retired.
  - ✓ Must insure that the invalidation completion indication to the RC will arrive at the RC after any previously posted writes that use the 'stale' address.
  - ✓ Is NOT required to immediately flush all pending requests upon receipt of an Invalidate Request. If transactions are in a queue waiting to be sent, it is not necessary for the device to expunge requests from the queue even if those transaction use an address that is being invalidated.

Example ATS Request Flow



1. Endpoint determines address ranges that will benefit from ATS
   1. For example, an Endpoint may want to translate addresses associated with work queues while not for single-use scatter-gather addresses
2. Endpoint issues an ATS Request
3. Translation Agent (TA) examines ATPT to determine if a translation exists
4. If translation exists, issues an ATS Completion with associated translation and access rights
   1. A TA may selectively ignore Requests for a given Function even if a translation exists
5. If translation does not exist, issues an ATS Completion indicating no such translation
   1. Endpoint may still issue DMA TLP but must not set the “Translation bit”
6. Endpoint issues DMA TLP to access associated address range

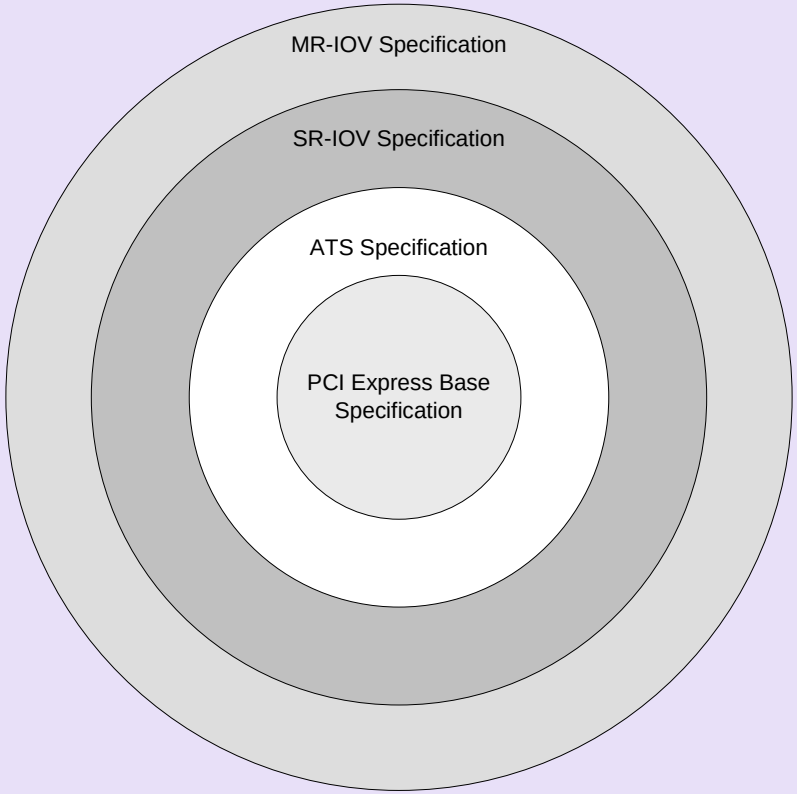
Example ATS Invalidation Flow



- The ATPT requires a translation update
- TA issues ATS Invalidation Request
- Function may flush or complete all pending transactions associated with the effected address range
- Once the Function has cleaned up all state associated with the effected address range, it returns an ATS Invalidation Completion
- TA processes ATS Invalidation Completion and updates ATPT accordingly

Interoperability

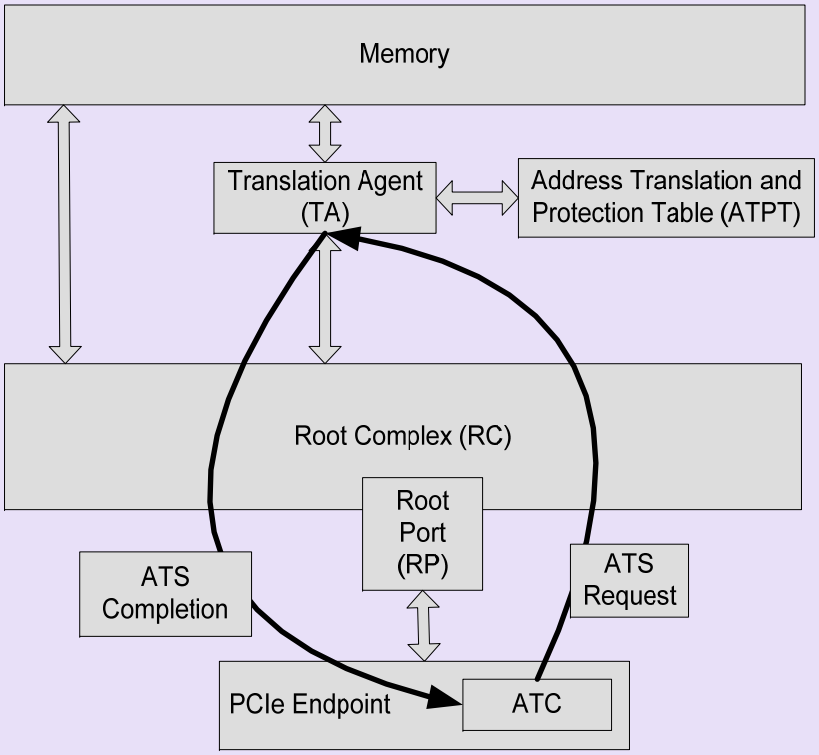
Interoperability Requirements



- At its core, I/O virtualization specifications build upon the *PCI Express Base Specification*. All IOV implementations must be compliant with the *PCI Express Base Specification* version 1.1 or later versions. Where applicable, the IOV specifications note relevant deltas between these versions.
  - ✓ None of the IOV specifications touch the physical layer.
  - ✓ The *Single Root I/O Virtualization Specification* does not touch the data link or transaction layers specified in the *PCI Express Base Specification*.
  - ✓ The *Multi Root I/O Virtualization Specification* does not touch the transaction layer specified in the *PCI Express Base Specification*.
  - ✓ All I/O virtualization capabilities are communicated through new PCIe capabilities implemented in PCI Express extended configuration space.
  - ✓ I/O virtualization specifications have no impact on PCI or PCI-X specifications.
- A hierarchy can be composed of a mix of PCIe and PCI Express to PCI / PCI-X Bridges.
  - A PCI Express to PCI / PCI-X Bridge and PCI / PCI-X Devices can be serially shared by multiple SI.
- The *Address Translation Specification* defines optional functionality applicable to any Function, PF, or VF. ATS can be supported in both SR-IOV and MR-IOV components.
- To implement a SR-IOV Device, the *Single Root I/O Virtualization Specification* requires the Device to be fully compliant with the *PCI Express Base Specification*.
  - ✓ A hierarchy can be composed of a mix of SR IOV and non-SR IOV components. For example, a hierarchy may contain any mix of SR IOV and non-SR IOV Endpoint Devices.
- To implement a MR-IOV Device, the *Multi Root I/O Virtualization Specification* requires the Device to be fully compliant with the *Single Root I/O Virtualization Specification* as well as the *PCI Express Base Specification*.

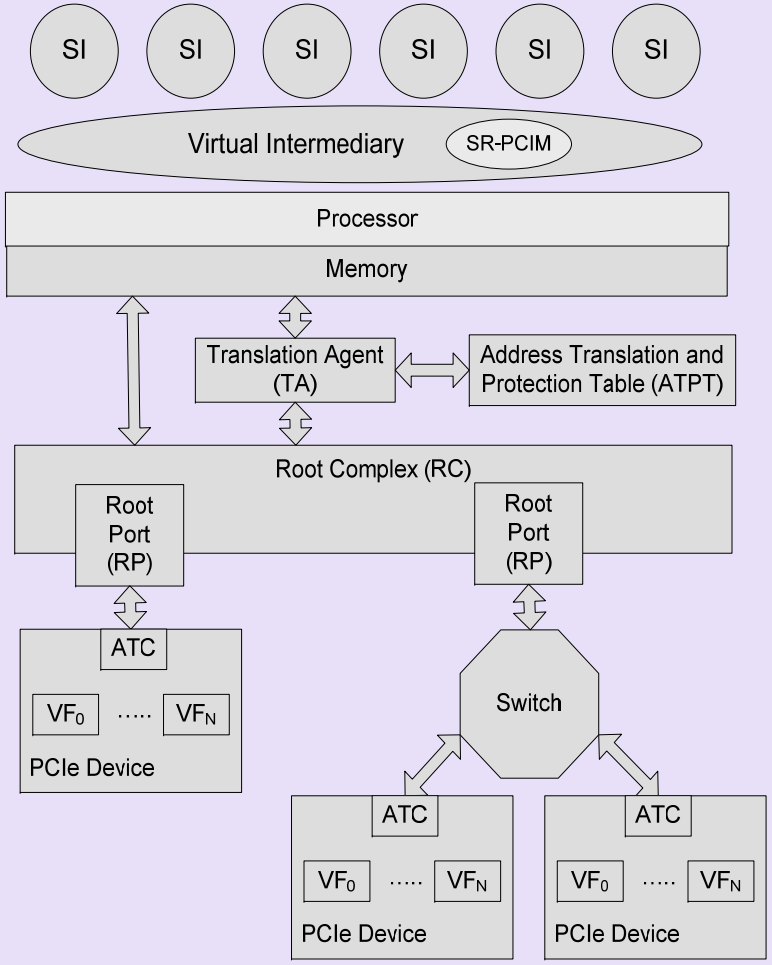
Specification Status and Schedule

ATS Specification Status



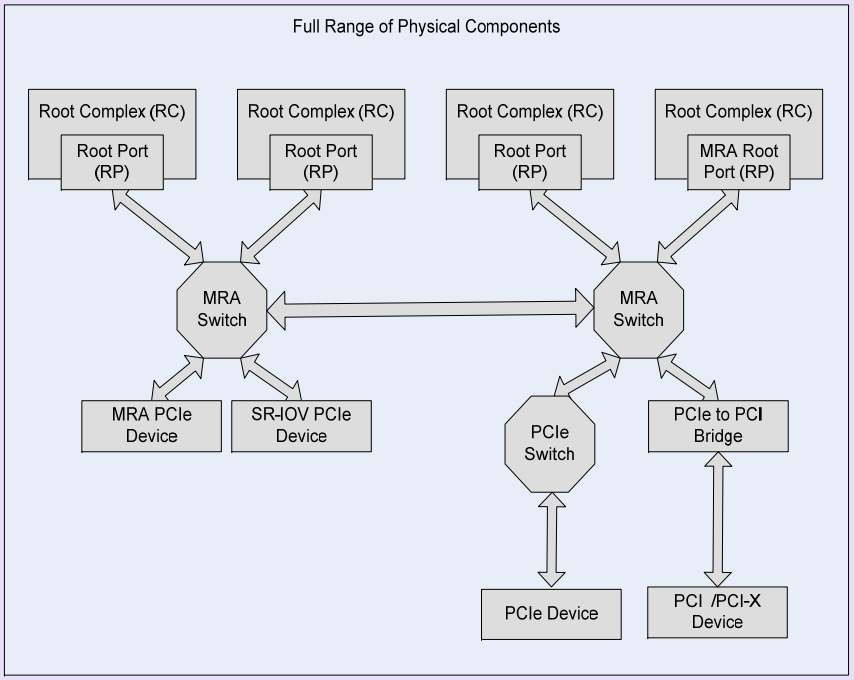
- Version 1.0 March 2007
- PCI-SIG members executing specification in wide range of product offerings

SR-IOV Specification Status



- Per PCI-SIG Specification Development process
- Draft 0.7 completed
- Draft 0.9 May 2007
- Version 1.0 Early 3Q2007

MR-IOV Specification Status



- Per PCI-SIG Specification Development process
- Draft 0.5 completed
- Draft 0.7 May 2007
- Draft 0.9 early 3Q2007
- Version 1.0 late 3Q2007

Questions

Workgroup Members

- AMD
- Broadcom
- Dolphin
- Emulex
- HP
- IBM
- IDT
- Intel
- LSI
- Microsoft
- NextIO
- Neterion
- Nvidia
- PLX
- Qlogic
- Sun
- VMWare

I/O Virtualization Architecture Overview

Michael Krause (HP, co-chair)
Renato Recio (IBM, co-chair)