

REVERSING SR-IOV FOR FUN AND PROFIT

Adir Abraham

@adirab

January 20th, 2019



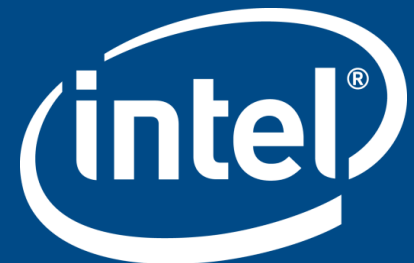
Legal Notices and disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [\[intel.com\]](https://www.intel.com).

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel and the Intel logo, are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.



\$ WHOAMI

- ▶ Name: Adir Abraham
 - ▶ Title: “We Break What We Make” @ **Intel**
 - ▶ Graduated: BSc CS Education – Technion IIT
 - ▶ Graduated: BA Economics – Technion IIT
 - ▶ Certified: CISSP, CySA+, CCSK
 - ▶ Twitter: **@adirab**
 - ▶ LinkedIn: **<https://linkedin.com/in/adirab>**
- 
- A series of white diagonal lines of varying lengths and thicknesses, located in the bottom right corner of the slide, creating a modern, abstract graphic element.

AGENDA

▶ Overview

- ▶ PCI Devices
- ▶ PCIe Devices
- ▶ I/O Virtualization

▶ PCIe Virtualization

- ▶ Motivation
- ▶ Directed I/O
- ▶ PCIe Architecture

▶ SR-IOV

- ▶ Understanding SR-IOV Capability
- ▶ ARI – Alternative Routing ID Interpretation (if time permits)
- ▶ ATS – Address Translation Service (if time permits)

▶ Demo

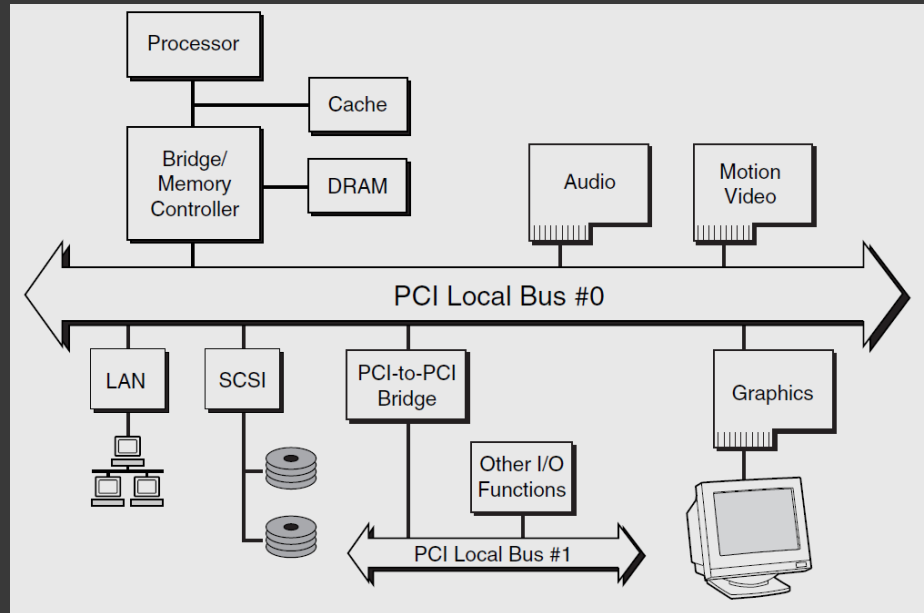
- ▶ Reversing PCIe features
- ▶ Reversing SR-IOV capabilities

▶ Show case (if time permits)

- ▶ Reversing NIC DPI and dataflow

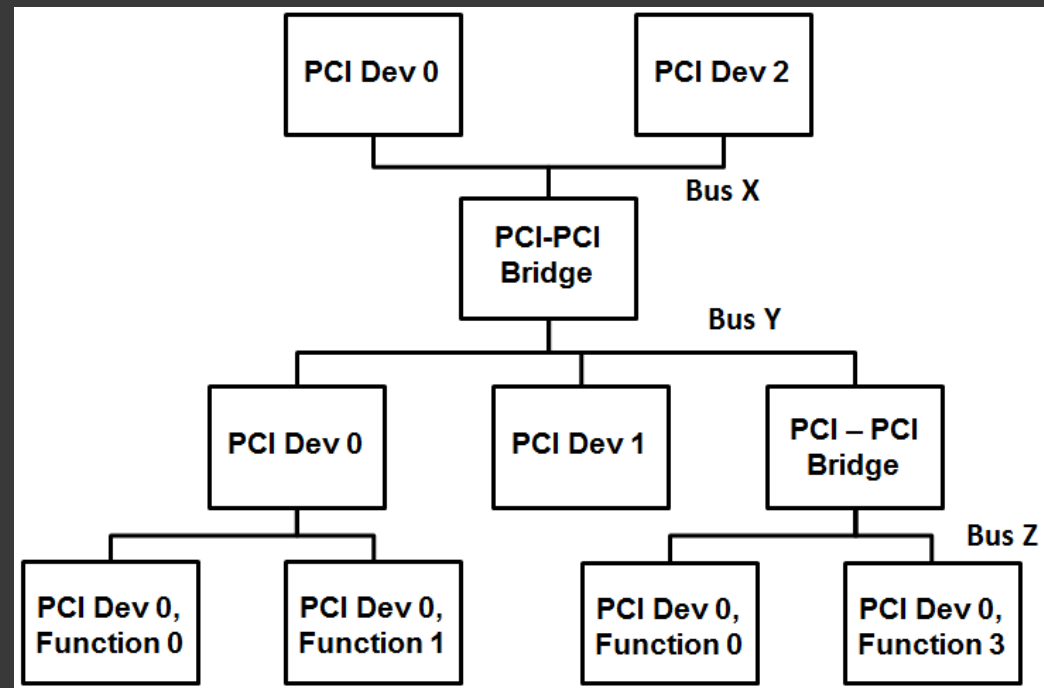
PCI - PERIPHERAL COMPONENT INTERCONNECT

- ▶ It's a **bus protocol** created by Intel around 1992
- ▶ Also known as “Conventional PCI”
- ▶ Purpose is to **attach/interconnect local hardware devices** to a computer, **independent** of any particular processor's native bus.
- ▶ PCI is **integrated into the chipset**, forming a “backbone”
- ▶ The most recent Conventional PCI standard is “PCI Local Bus Specification Rev. 3.0”
- ▶ PCI standards are currently maintained and defined by the PCI-SIG.



GENERIC PCI TOPOLOGY: BUSES, DEVICES, AND FUNCTIONS

- ▶ **Each Bus** can have up to **32 devices** attached
- ▶ **Each Device** can extend up to **8 functions**
- ▶ **Up to 256 Buses** are interconnected by up to **256 bridge interfaces** and are enumerated



WHAT KIND OF PCI(E) DEVICES DO WE HAVE?

CLASS CODES!

Base Class	Meaning
00h	Device was built before Class Code definitions were finalized
01h	Mass storage controller
02h	Network controller
03h	Display controller
04h	Multimedia device
05h	Memory controller
06h	Bridge device
07h	Simple communication controllers
08h	Base system peripherals
09h	Input devices
0Ah	Docking stations
0Bh	Processors
0Ch	Serial bus controllers
0Dh	Wireless controller
0Eh	Intelligent I/O controllers
0Fh	Satellite communication controllers
10h	Encryption/Decryption controllers
11h	Data acquisition and signal processing controllers
12h - FEh	Reserved
FFh	Device does not fit in any defined classes

EXAMPLE – BASE CLASS 03H

Base Class	Sub-Class	Interface	Meaning
03h	00h	0000 0000b	VGA-compatible controller. Memory addresses 0A 0000h through 0B FFFFh. I/O addresses 3B0h to 3BBh and 3C0h to 3DFh and all aliases of these addresses.
		0000 0001b	8514-compatible controller -- 2E8h and its aliases, 2EAh-2EFh
	01h	00h	XGA controller
	02h	00h	3D controller
	80h	00h	Other display controller



EXAMPLE – BASE CLASS 09H

Base Class	Sub-Class	Interface	Meaning
09h	00h	00h	Keyboard controller
	01h	00h	Digitizer (pen)
	02h	00h	Mouse controller
	03h	00h	Scanner controller
	04h	00h	Gameport controller (generic)
		10h	Gameport controller (see below)
	80h	00h	Other input controller



EXAMPLE – BASE CLASS 0DH

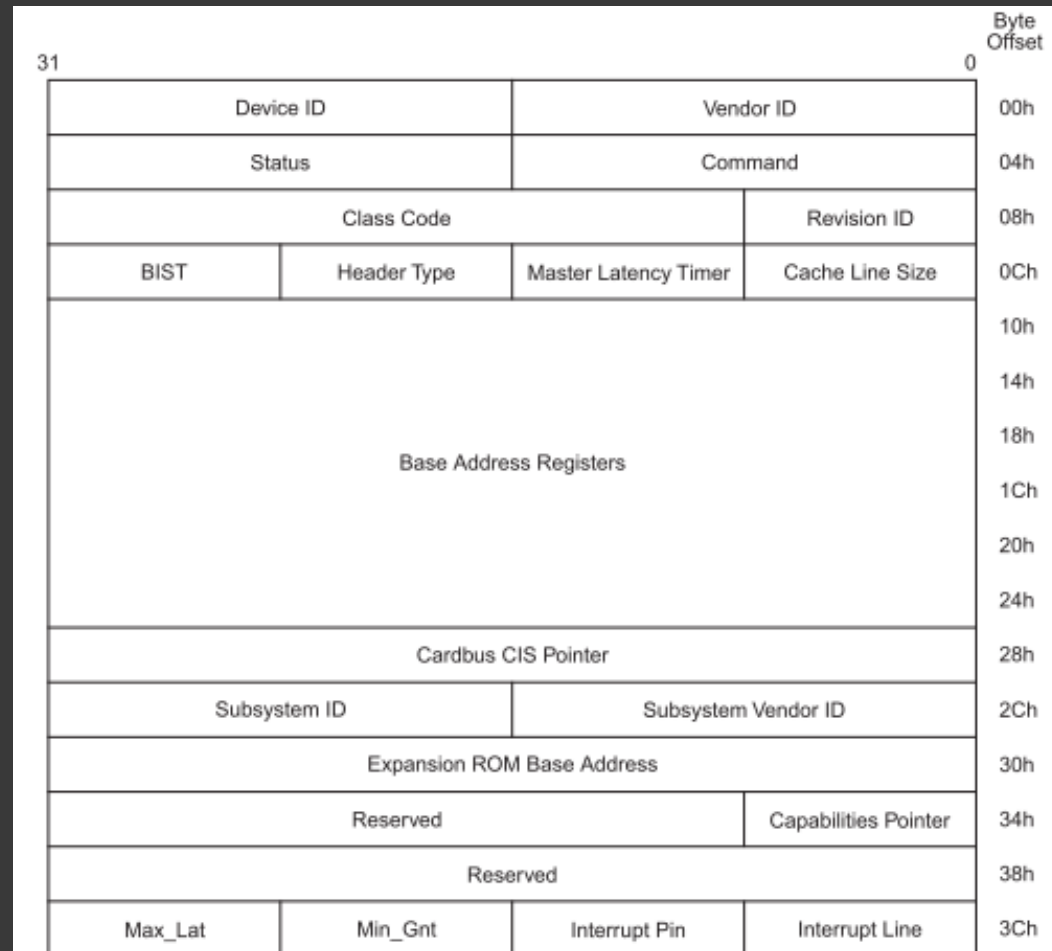
Base Class	Sub-Class	Interface	Meaning
0Dh	00	00h	iRDA compatible controller
	01h	00h	Consumer IR controller
	10h	00h	RF controller
	11h	00h	Bluetooth
	12h	00h	Broadband
	20h	00h	Ethernet (802.11a – 5 GHz)
	21h	00h	Ethernet (802.11b – 2.4 GHz)
	80h	00h	Other type of wireless controller



EACH PCI(E) DEVICE INCLUDES CONFIGURATION SPACE

- “Devices will **allocate resource such as memory** and record the address into its configuration space”

(PCI Local Bus Specification ver.2.3 Chap 6)



OM14316

Figure 3-2: Type 0 Configuration Space Header

PCI ADDRESS SPACES

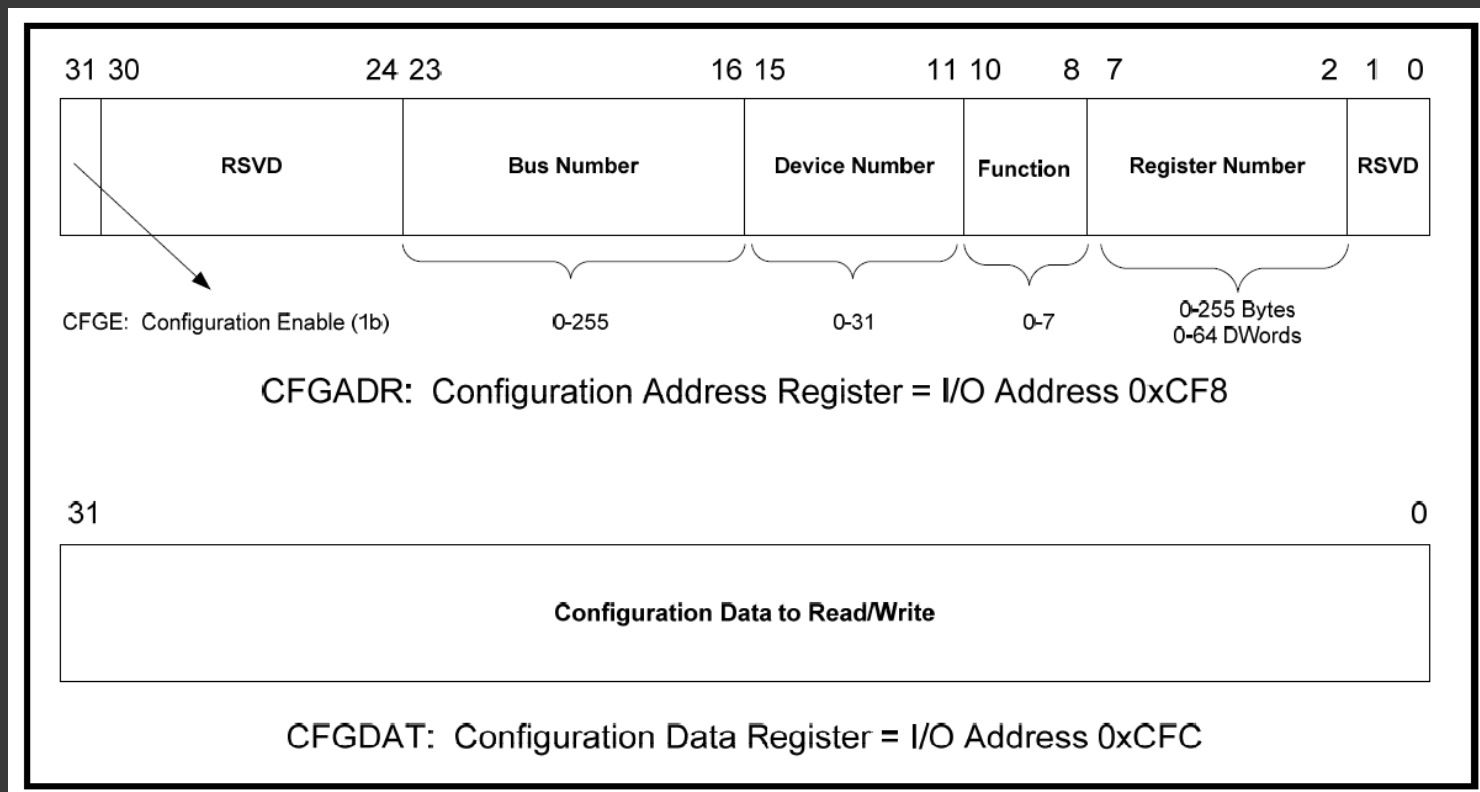
- ▶ PCI implements **three** address spaces:
- ▶ 1) PCI Configuration Space (up to 256 Bytes)
 - ▶ Required/standard. Defined in the specifications. Every PCI device has a configuration space.
- ▶ 2) PCI Memory-mapped space
 - ▶ Optional. Dependent on whether the device manufacturer needs to map system memory to the PCI device
- ▶ 3) PCI I/O-mapped space
 - ▶ Optional. Same as PCI Memory Space

TYPICAL PCI COMMANDS

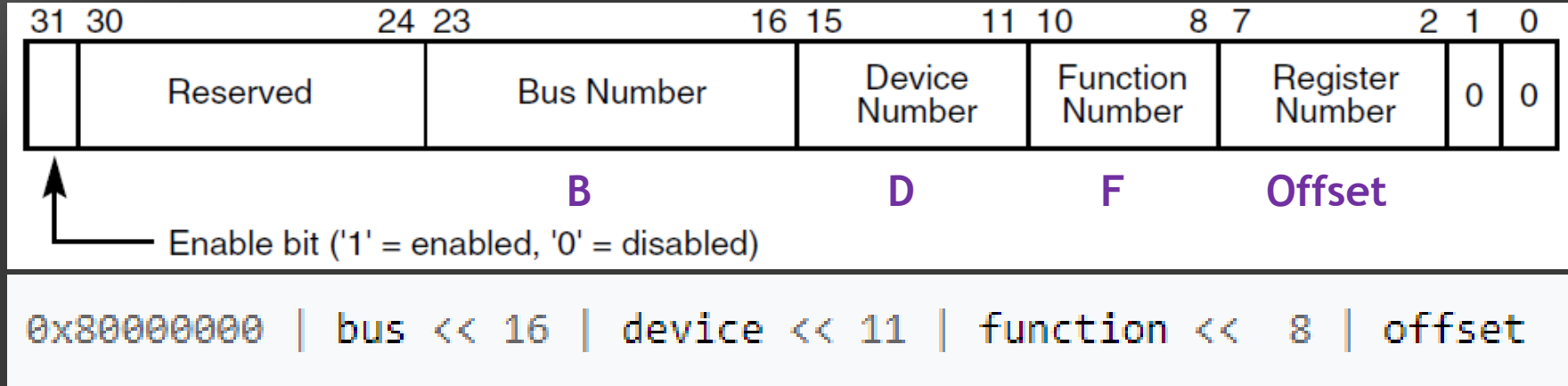
TLP Type	Fmt [2:0] ² (b)	Type [4:0] (b)	Description
MRd	000 001	0 0000	Memory Read Request
MRdLk	000 001	0 0001	Memory Read Request-Locked
MWr	010 011	0 0000	Memory Write Request
IORd	000	0 0010	I/O Read Request
IOWr	010	0 0010	I/O Write Request
CfgRd0	000	0 0100	Configuration Read Type 0
CfgWr0	010	0 0100	Configuration Write Type 0
CfgRd1	000	0 0101	Configuration Read Type 1
CfgWr1	010	0 0101	Configuration Write Type 1

GENERATING CONFIGURATION TRANSACTIONS

- ▶ Two **32-bit I/O locations** are used to generate configuration transactions: **CF8h (CONFIG_ADDRESS)** ; **CFCh (CONFIG_DATA)**
- ▶ Compatible PCI is configured using the **port I/O address/data pair** (CONFIG_ADDRESS, CONFIG_DATA)



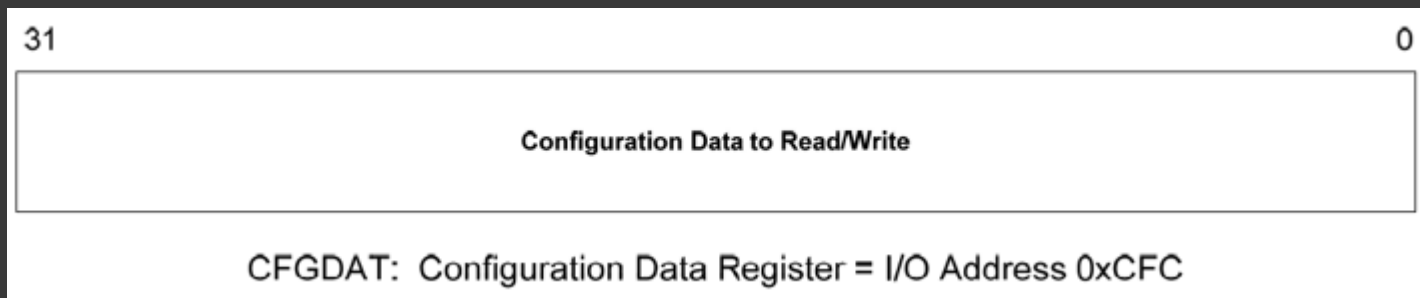
I/O PORT CONFIG_ADDRESS (CF8H)



- ▶ 32 bits (**GIMME BUS 0, DEVICE 31 (0x1F), Function 0, offset 0x88**)
- ▶ Port CF8h
- ▶ Bit 31 when set, all reads and writes to CONFIG_DATA are PCI Configuration transactions
- ▶ Bits 30:24 are read-only and must return 0 when read
- ▶ Bits 23:16 select a specific **Bus** in the system (up to 256 buses)
- ▶ Bits 15:11 specify a **Device** on the given Bus (up to 32 devices)
- ▶ Bits 10:8 Specify the **function** of a device (up to 8 devices)
- ▶ Bits 7:0 Select an **offset** within the Configuration Space
- ▶ Addresses are often given in **B/D/F, Offset** notation

I/O PORT CONFIG_DATA (CFCH)

- ▶ CONFIG_DATA can be accessed in **DWORD**, **WORD**, or **BYTE** configurations
- ▶ Reads and Writes to CONFIG_DATA with **Bit 31** in CONFIG_ADDRESS **set/enabled** results in a PCI Configuration transaction to the device specified in CONFIG_ADDRESS
- ▶ PCI spec says that if Bit 31 is not enabled, then the transaction is forwarded out as Port I/O



REVERSING PCI DEVICE READ REQUESTS

(reloc.PciCf8Read8_167)

0x000034a6

jmp sym.PciCf8Read8; RELOC 32 PciCf8Read8

(fcn) sym.PciCf8Read8 157

sym.PciCf8Read8 (int arg4);

; arg int arg4 @ rcx

0x00002c72 push rdi; RELOC 32

0x00002c73 push rsi

0x00002c74 push rbx

0x00002c75 mov rbx, rcx; RELOC 32 ; arg4

0x00002c78 sub rsp, 0x20; RELOC 64

(reloc.DebugAssertEnabled_125)

0x00002c7c call sym.DebugAssertEnabled; RELOC 32 DebugAssertEnabled

0x00002c81 test al, al; RELOC 64

0x00002c83 je 0x2ca6; RELOC 32

REVERSING PCI DEVICE READ REQUESTS

```
(reloc.SaveAndDisableInterrupts_167)
0x00002ca6      call sym.SaveAndDisableInterrupts; RELOC 32 SaveAndDisableInterrupts
0x00002cab      mov ecx, 0xcf8; RELOC 32
0x00002cb0      mov esi, eax; RELOC 32
(reloc.IORead32_179)
0x00002cb2      call sym.IORead32; RELOC 32 IORead32
0x00002cb7      mov rdx, rbx
0x00002cba      mov ecx, 0xcf8
0x00002cbf      mov edi, eax
0x00002cc1      shr rdx, 4; RELOC 32
0x00002cc5      mov eax, ebx
0x00002cc7      and ebx, 3; RELOC 64
0x00002cca      and eax, 0xfc; RELOC 64
0x00002ccf      and edx, 0xffff00; RELOC 64
0x00002cd5      or  edx, eax
0x00002cd7      or  edx, 0x80000000; RELOC 64
(reloc.IOWrite32_222)
0x00002cdd      call sym.IOWrite32; RELOC 32 IOWrite32
0x00002ce2      lea rcx, [rbx + 0xcfc]; RELOC 64
(reloc.IORead8_234)
0x00002ce9      call sym.IORead8; RELOC 32 IORead8
0x00002cee      mov edx, edi
0x00002cf0      mov ecx, 0xcf8; RELOC 64
0x00002cf5      mov ebx, eax
(reloc.IOWrite32_248)
0x00002cf7      call sym.IOWrite32; RELOC 32 IOWrite32
0x00002cfc      movzx ecx, sil
(reloc.SetInterruptState_1)
0x00002d00      call sym.SetInterruptState; RELOC 32 SetInterruptState
0x00002d05      add rsp, 0x20
0x00002d09      mov eax, ebx
0x00002d0b      pop rbx
0x00002d0c      pop rsi; RELOC 32
0x00002d0d      pop rdi
0x00002d0e      ret
```

REVERSING PCI DEVICE READ REQUESTS

(reloc.SaveAndDisableInterrupts_167)

```
0x00002ca6      call sym.SaveAndDisableInterrupts; RELOC 32 SaveAndDisableInterrupts
0x00002cab      mov  ecx, 0xcf8; RELOC 32
0x00002cb0      mov  esi, eax; RELOC 32
```

(reloc.IORead32_179)

```
0x00002cb2      call sym.IORead32; RELOC 32 IORead32
0x00002cb7      mov  rdx, rbx
0x00002cba      mov  ecx, 0xcf8
0x00002cbf      mov  edi, eax
0x00002cc1      shr  rdx, 4; RELOC 32
0x00002cc5      mov  eax, ebx
0x00002cc7      and  ebx, 3; RELOC 64
0x00002cca      and  eax, 0xfc; RELOC 64
0x00002ccf      and  edx, 0xffff00; RELOC 64
0x00002cd5      or   edx, eax
0x00002cd7      or   edx, 0x80000000; RELOC 64
```

0x80000000 | bus << 16 | device << 11 | function << 8 | offset

(reloc.IOWrite32_222)

```
0x00002cdd      call sym.IOWrite32; RELOC 32 IOWrite32
0x00002ce2      lea  rcx, [rbx + 0xcfc]; RELOC 64
```

(reloc.IORead8_234)

```
0x00002ce9      call sym.IORead8; RELOC 32 IORead8
0x00002cee      mov  edx, edi
0x00002cf0      mov  ecx, 0xcf8; RELOC 64
0x00002cf5      mov  ebx, eax
```

(reloc.IOWrite32_248)

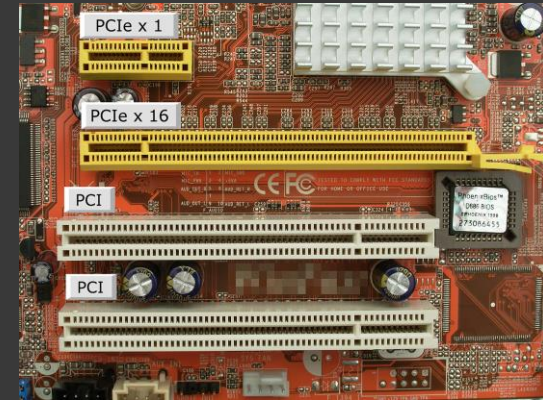
```
0x00002cf7      call sym.IOWrite32; RELOC 32 IOWrite32
0x00002cfc      movzx ecx, sil
```

(reloc.SetInterruptState_1)

```
0x00002d00      call sym.SetInterruptState; RELOC 32 SetInterruptState
0x00002d05      add  rsp, 0x20
0x00002d09      mov  eax, ebx
0x00002d0b      pop  rbx
0x00002d0c      pop  rsi; RELOC 32
0x00002d0d      pop  rdi
0x00002d0e      ret
```

PCI EXPRESS (PCIE)

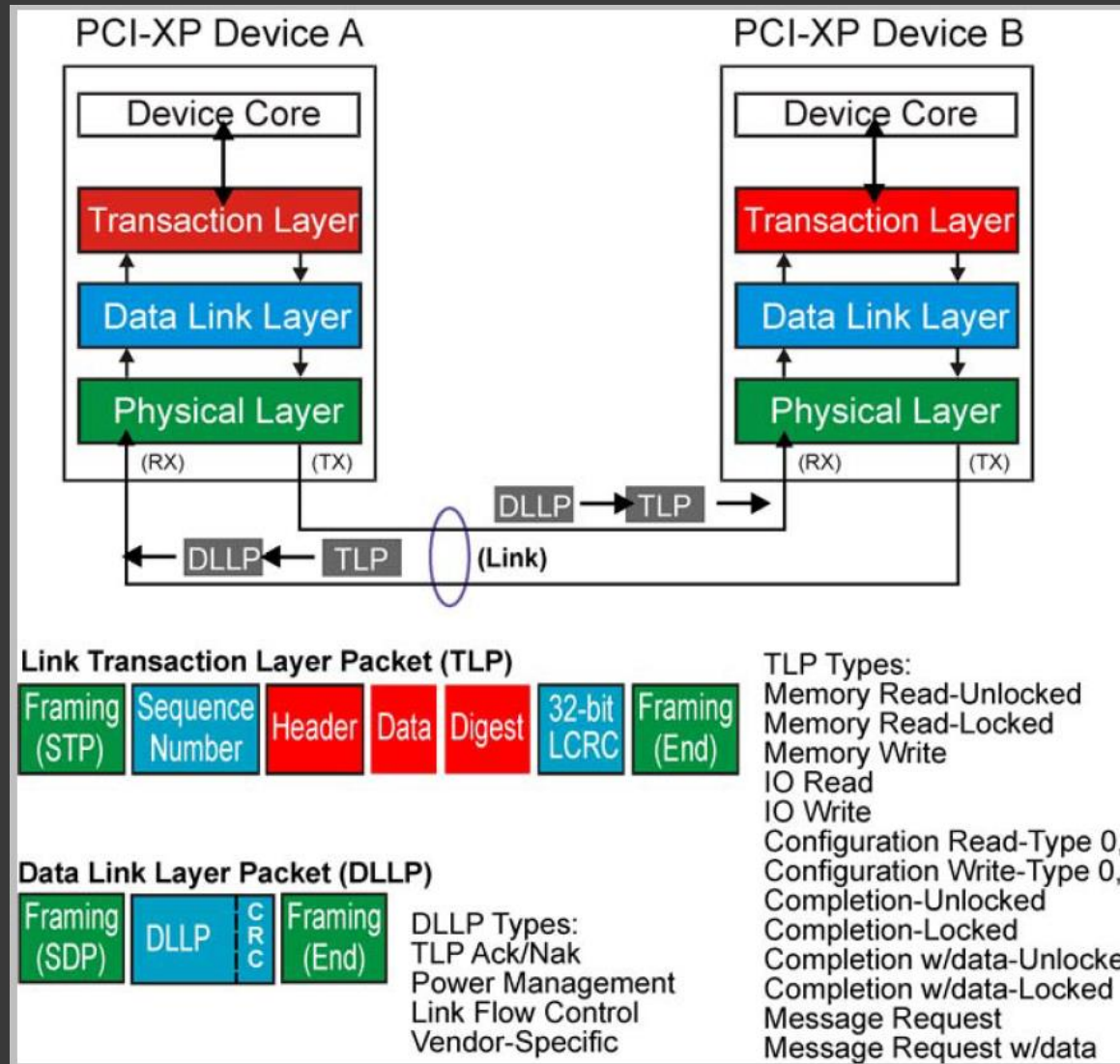
- ▶ Peripheral Component Interconnect **Express**
- ▶ Developed around 2003
- ▶ **Packet-based** transaction protocol
- ▶ Packet data is stripped across **lanes**
- ▶ Peak data throughput **scales** with the overall link width (maximum speed HW capability)
- ▶ Very **different** from PCI at the **hardware** level



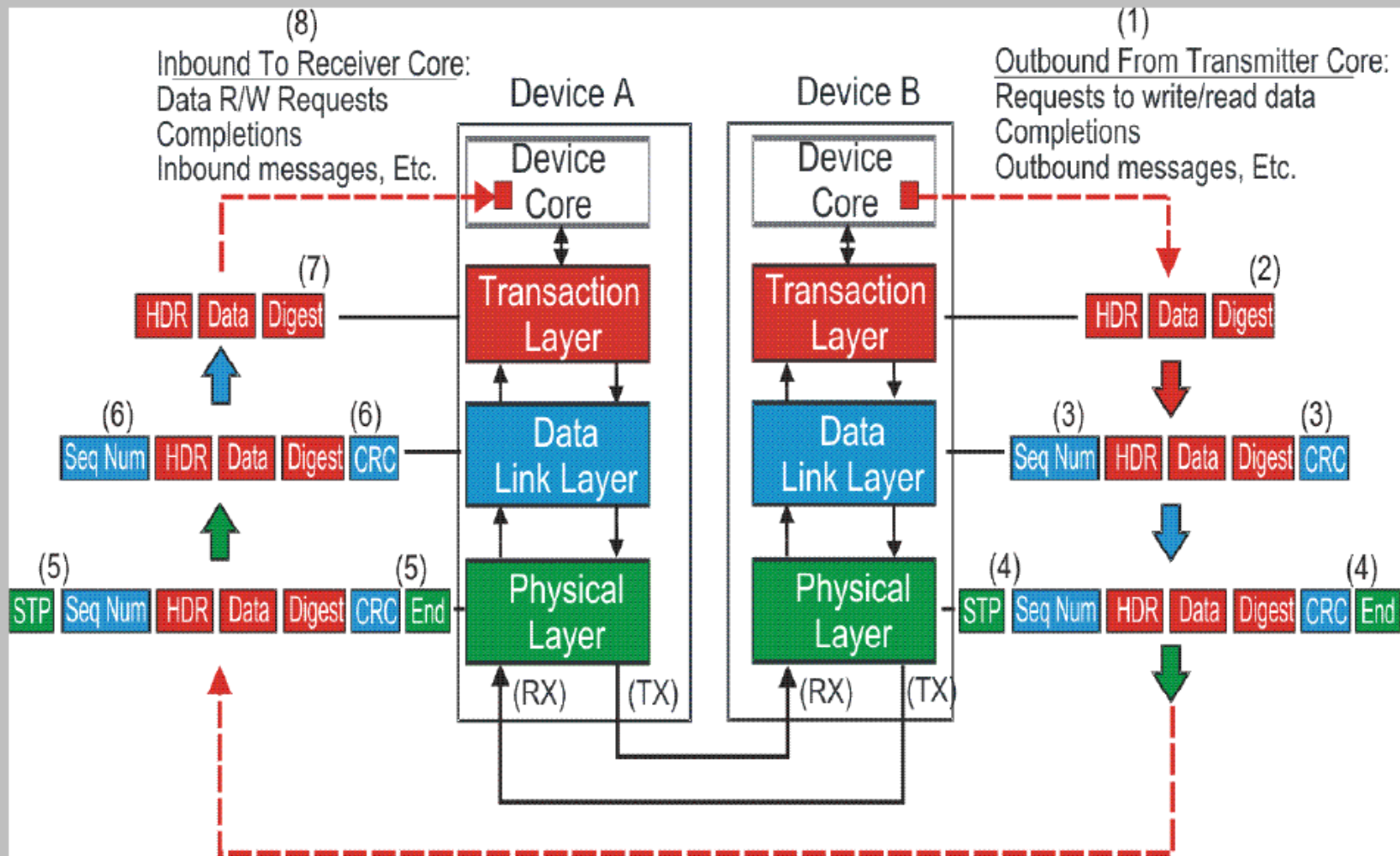
PCI Express link performance

PCI Express version	Introduced	Line code	Transfer rate	Throughput				
				×1	×2	×4	×8	×16
1	2003	8b/10b	2.5 GT/s	250 MB/s	0.50 GB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s
2	2007	8b/10b	5.0 GT/s	500 MB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s	8.0 GB/s
3	2010	128b/130b	8.0 GT/s	984.6 MB/s	1.97 GB/s	3.94 GB/s	7.88 GB/s	15.8 GB/s
4	2017	128b/130b	16.0 GT/s	1969 MB/s	3.94 GB/s	7.88 GB/s	15.75 GB/s	31.5 GB/s
5	2019(*)	128b/130b	32.0 GT/s	3938 MB/s	7.88 GB/s	15.75 GB/s	31.51 GB/s	63.0 GB/s

PCIE DEVICES USE TLP

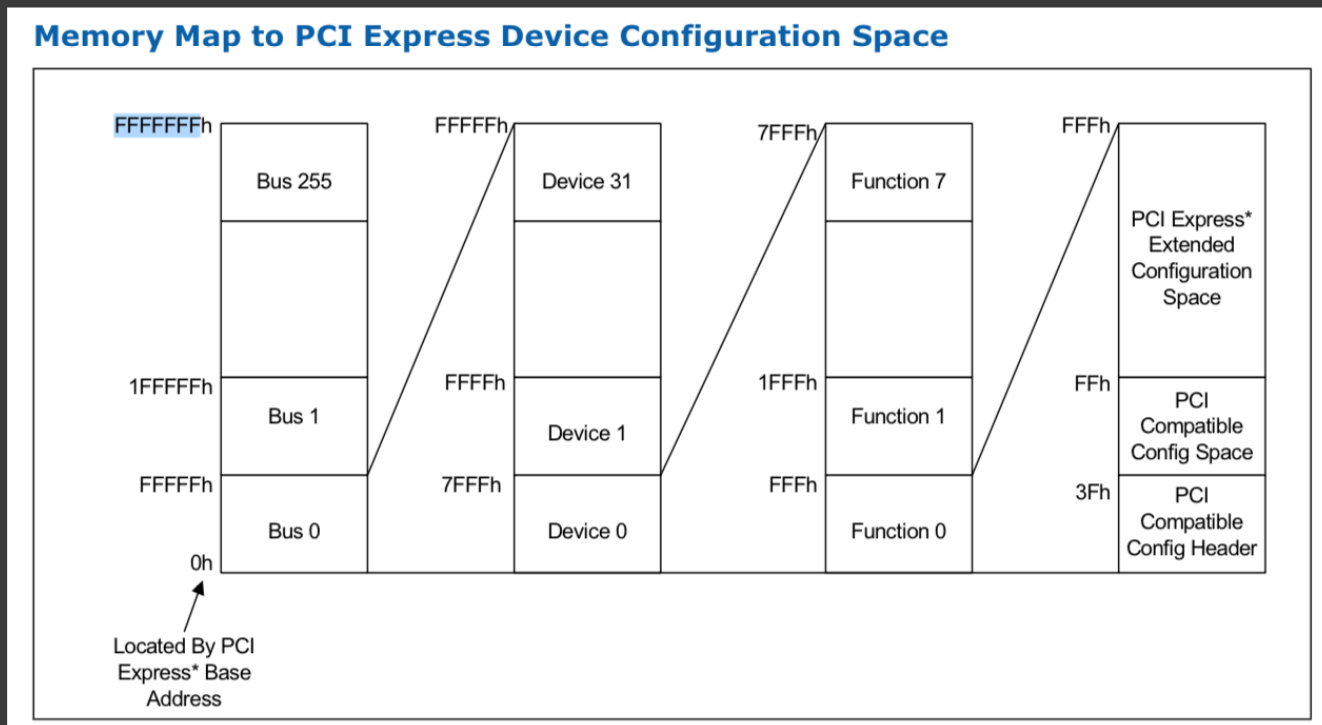


TLP ASSEMBLY/DISASSEMBLY



PCIE AND PCI SIMILARITIES

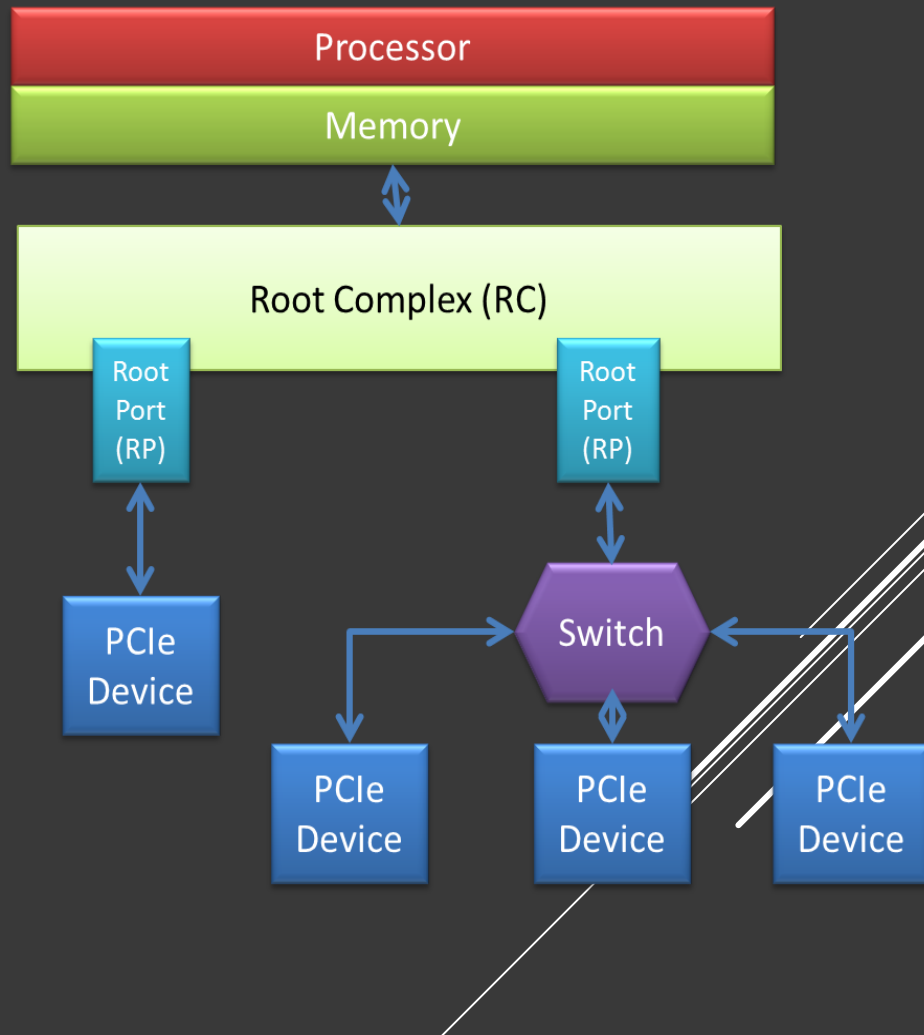
- ▶ For **software configuration** purposes, it is mostly the same
 - ▶ Adds an **extended configuration space** of 4KB, using 4KB of PCIe Extended Capabilities registers
- ▶ Provides **backwards compatibility** for Conventional PCI



PCIE COMPONENTS

► Root Complex

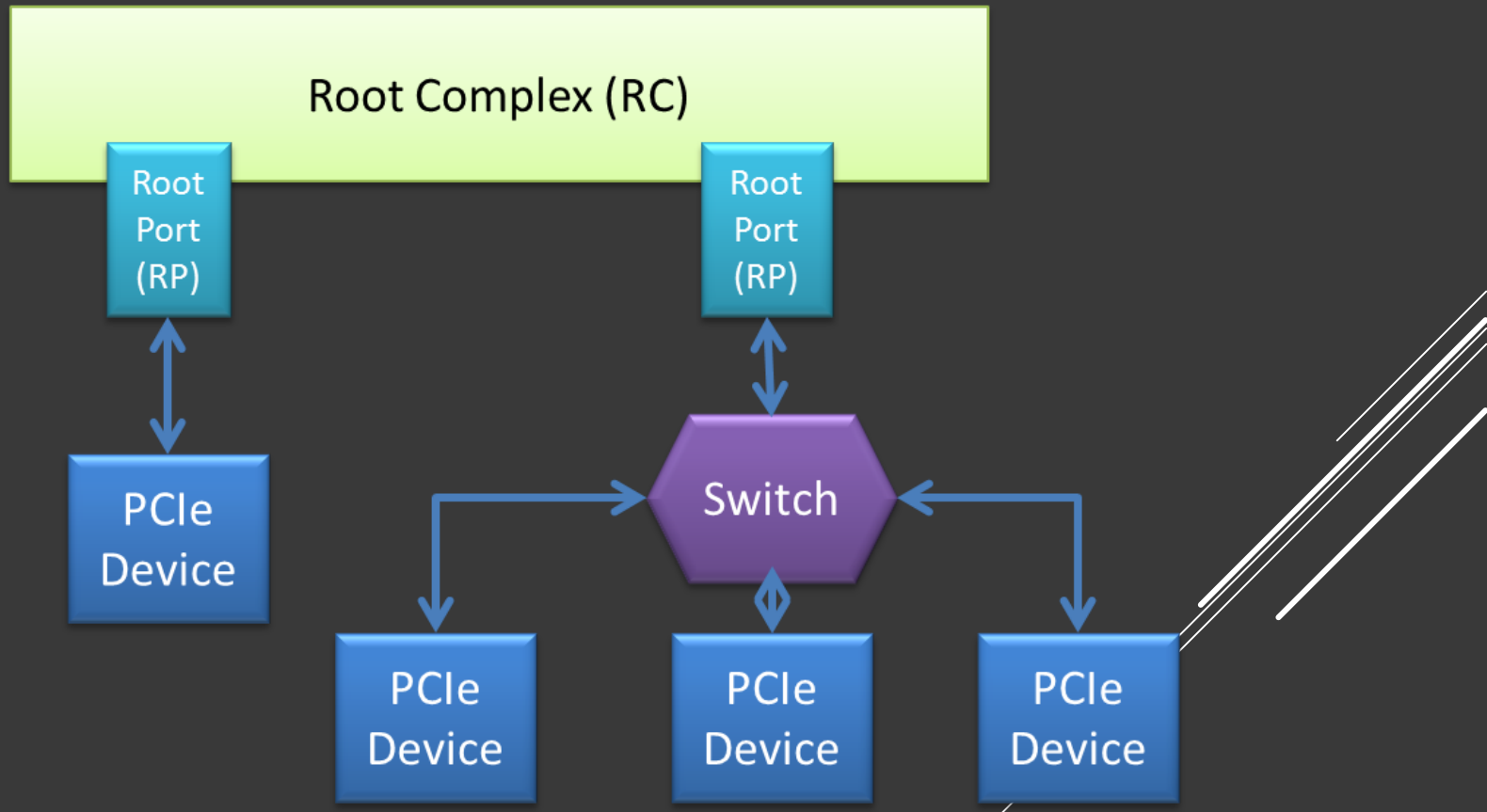
- A root complex connects the processor and memory subsystem to the PCIe switch fabric composed of one or more switch devices
- Similar to a host bridge in a PCI system
 - Generate transaction requests on behalf of the processor, which is interconnected through a local bus.
 - May contain more than one PCIe port and multiple switch devices.



PCIe COMPONENTS

► Root Port (RP)

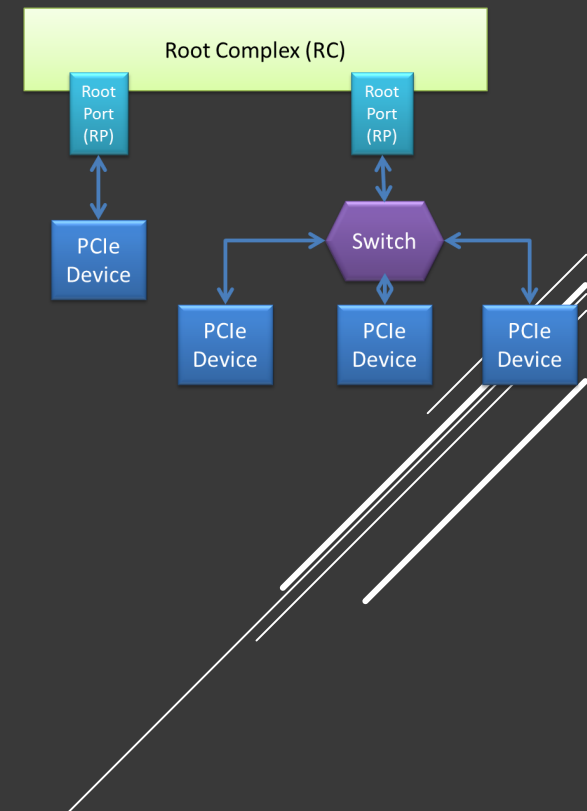
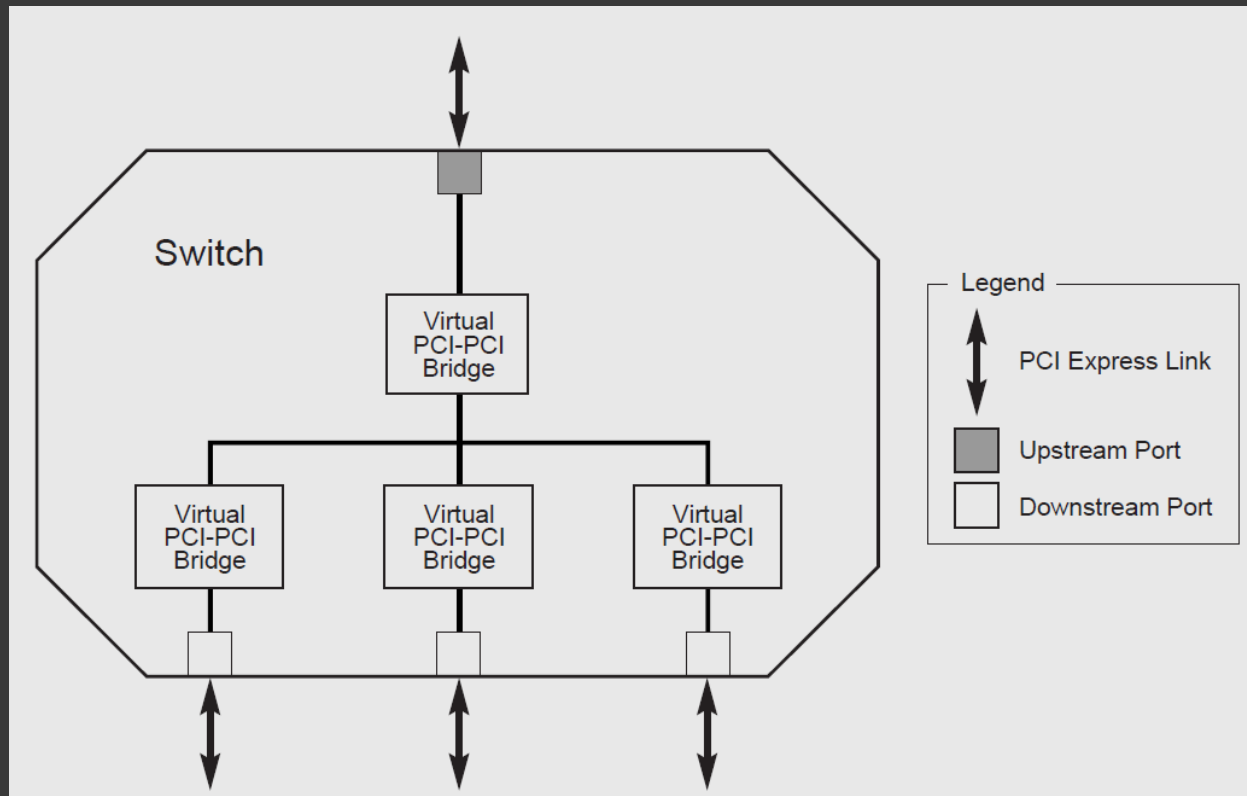
- The portion of the motherboard that contains the host bridge. The host bridge allows the PCIe ports to talk to the rest of the computer



PCIE COMPONENTS

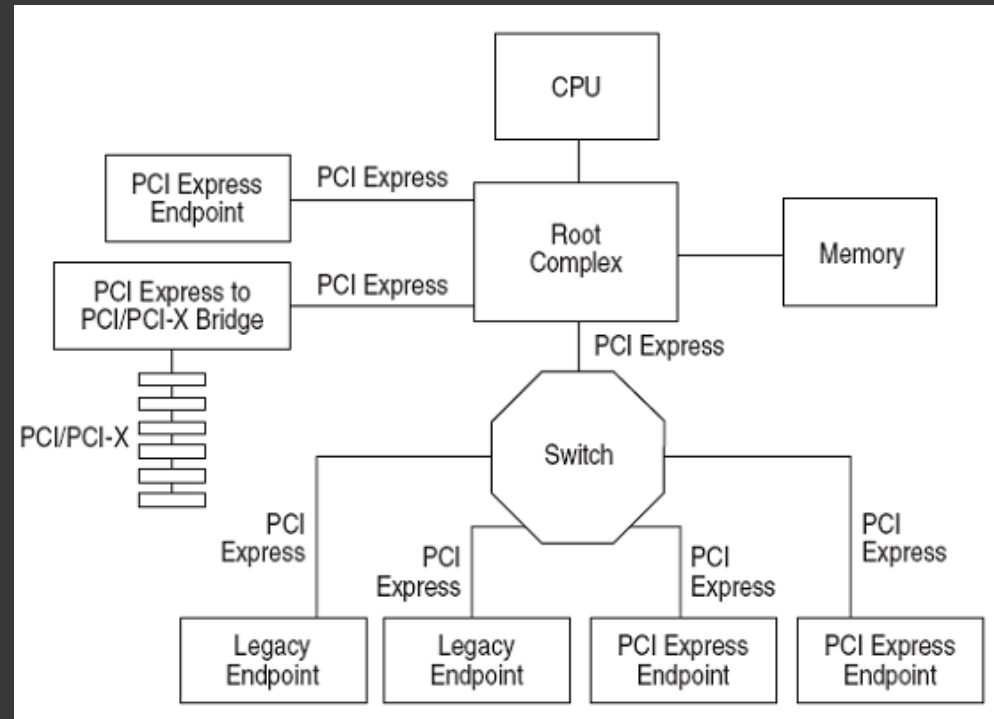
► Switch

- A defined System Element that connects two or more Ports to allow Packets to be routed from one Port to another. To configuration software, a Switch appears as a collection of virtual PCI-to-PCI Bridges.



GENERIC PCIE TOPOLOGY

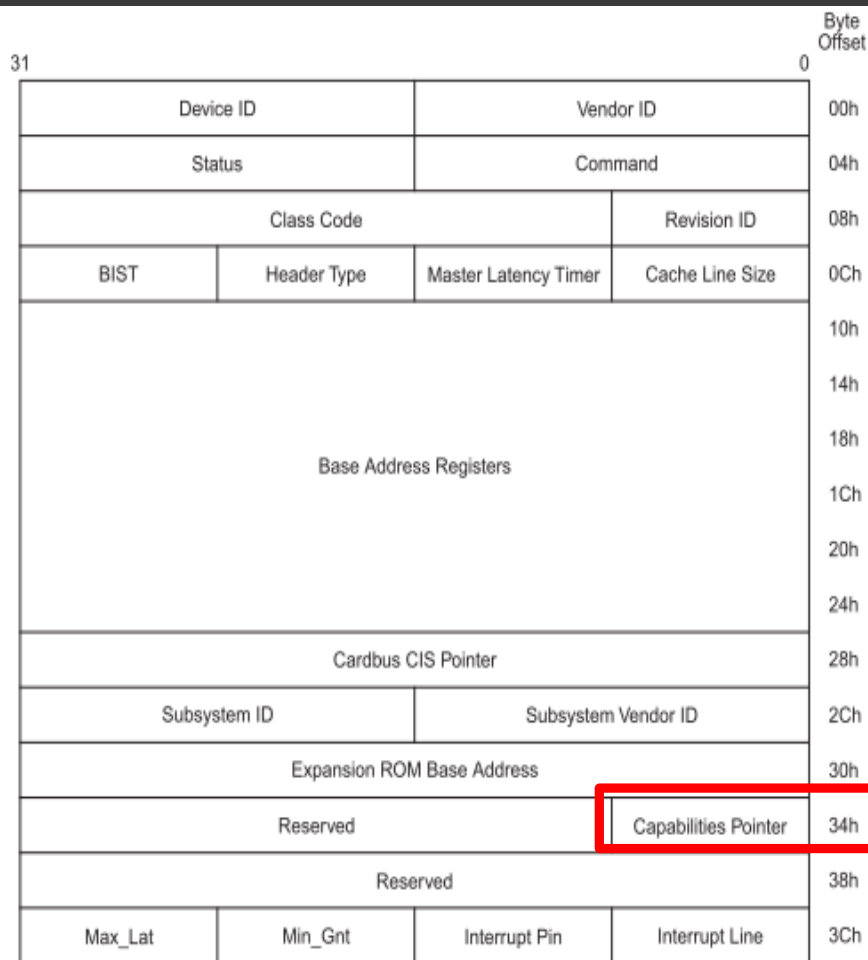
- ▶ The Root Complex connects the processor to the system memory and components
- ▶ Same number of devices supported as PCI
- ▶ Up to **256 PCIe buses**
- ▶ Up to **32 PCIe devices**
- ▶ Up to **8 Functions**
- ▶ Each Function can implement **up to 4 KB** of configuration space



PCI EXPRESS ADDRESS SPACES

- ▶ PCIe implements **four** address spaces:
- ▶ 1) PCIe Configuration Space (up to 4KBytes)
 - ▶ Required/standard. Defined in the specifications. Every PCIe device has its configuration space mapped to memory.
 - ▶ Also provides the first 256 bytes of compatible PCI (memory-mapped and via port IO for backwards compatibility)
- ▶ 2) PCIe Memory-mapped space
 - ▶ Optional. Dependent on whether the device manufacturer needs to map system memory to the PCI device
- ▶ 3) PCIe I/O-mapped space
 - ▶ Optional. Same as PCI Memory Space
- ▶ 4) PCIe Message Space
 - ▶ For low-level protocol messaging/interrupts.

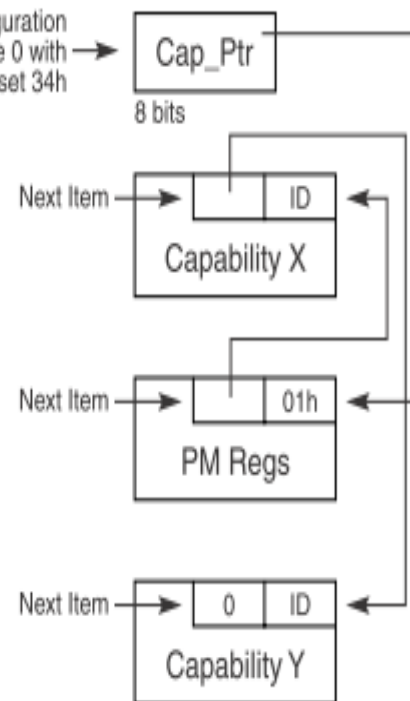
EXTENDED CAPABILITIES



OM14316

Figure 3-2: Type 0 Configuration Space Header

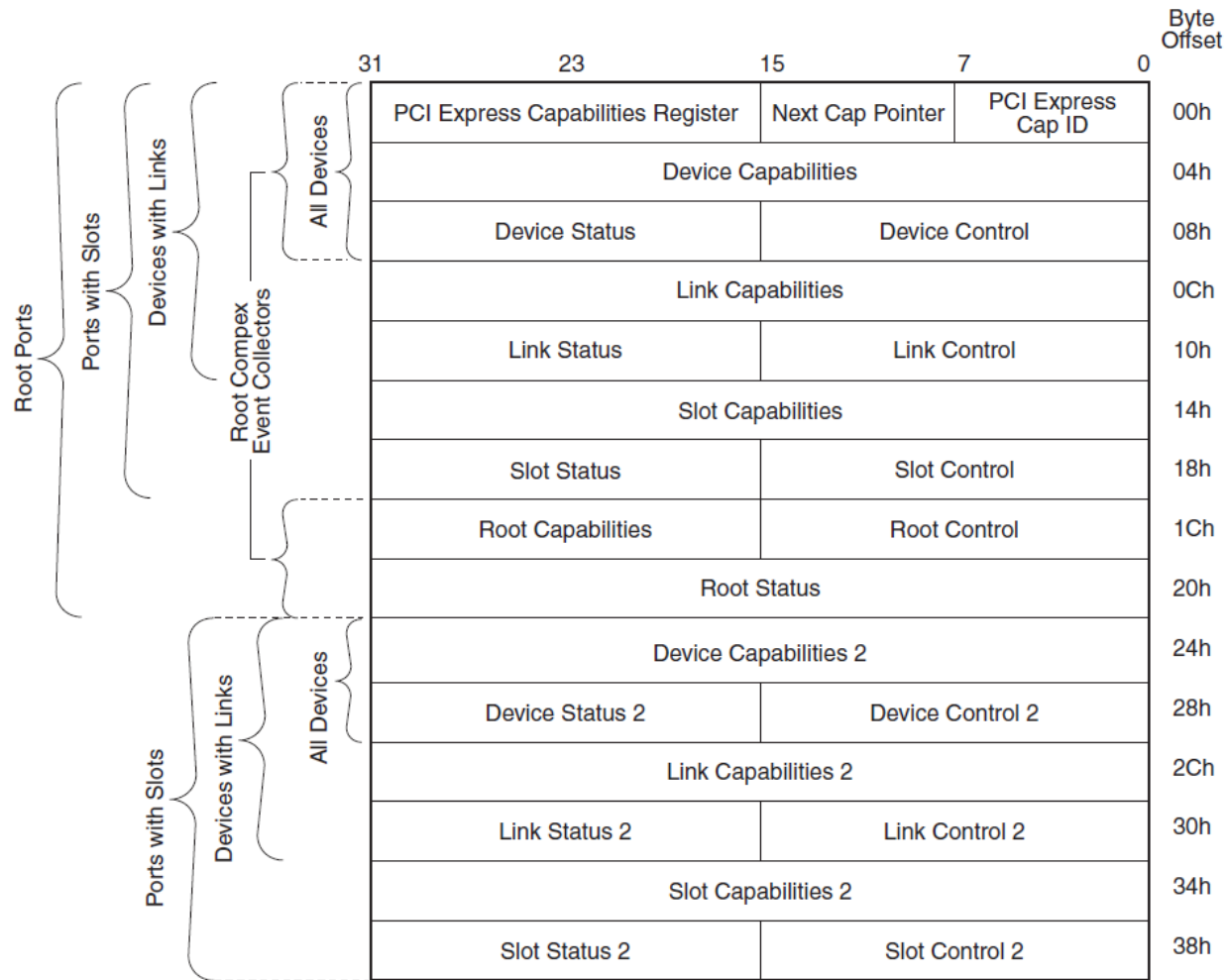
Standard PCI Configuration
Initial Header Type 0 with
Capability pointer at Offset 34h



A-0318

Figure 3-2: Capabilities Linked List

PCI EXPRESS CAPABILITY STRUCTURE

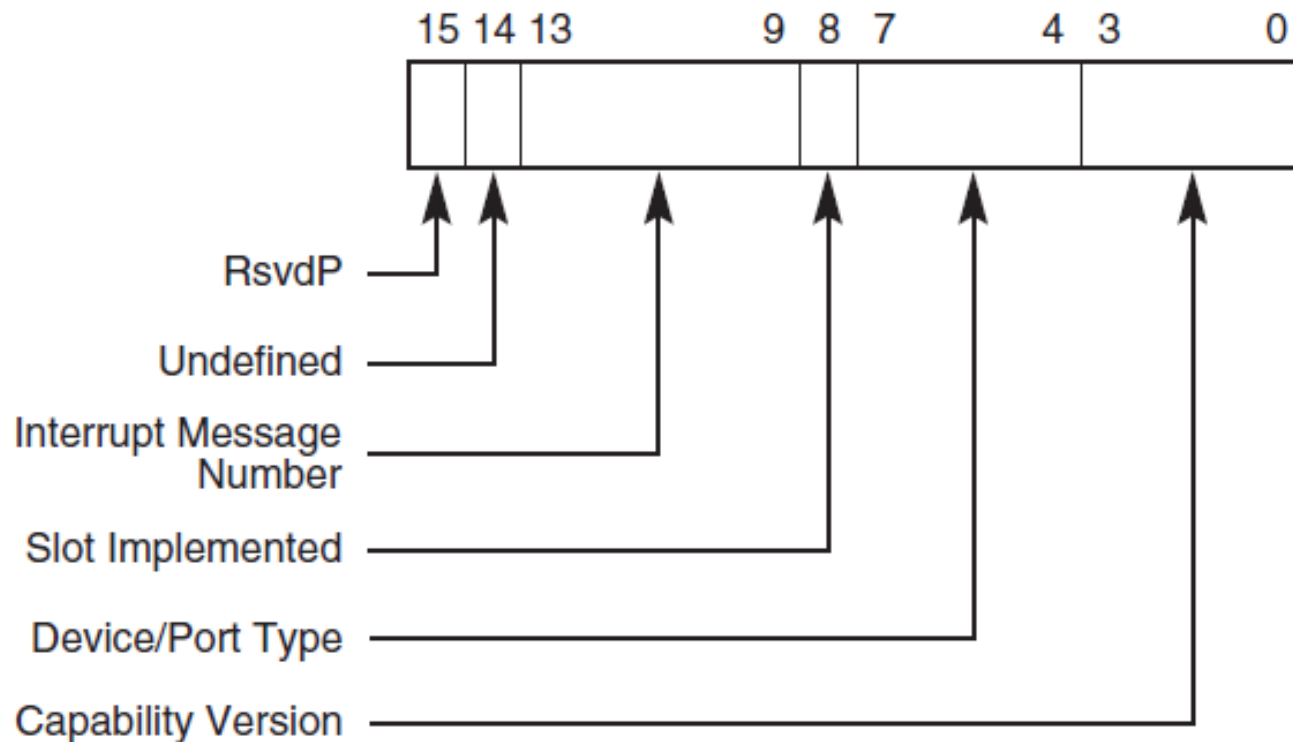


Note: Registers not applicable to a device are RsvdZ.

OM14318B

Figure 3-3: PCI Express Capability Structure

PCI EXPRESS CAPABILITIES REGISTER



OM14502A

Figure 3-5: PCI Express Capabilities Register

PCI EXPRESS DEVICE/PORT TYPE

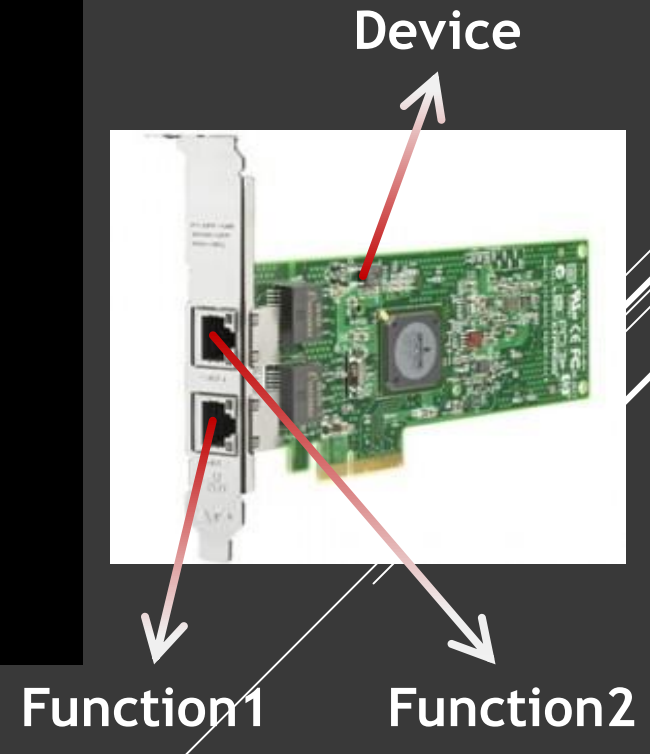
7:4	<p>Device/Port Type – Indicates the specific type of this PCI Express Function. Note that different Functions in a Multi-Function Device can generally be of different types.</p> <p>Defined encodings for Functions that implement a Type 00h PCI Configuration Space header are:</p> <ul style="list-style-type: none">0000b PCI Express Endpoint0001b Legacy PCI Express Endpoint1001b RCiEP1010b Root Complex Event Collector <p>Defined encodings for Functions that implement a Type 01h PCI Configuration Space header are:</p> <ul style="list-style-type: none">0100b Root Port of PCI Express Root Complex0101b Upstream Port of PCI Express Switch0110b Downstream Port of PCI Express Switch0111b PCI Express to PCI/PCI-X Bridge1000b PCI/PCI-X to PCI Express Bridge <p>All other encodings are Reserved.</p> <p>Note that the different Endpoint types have notably different requirements in Section 1.3.2 regarding I/O resources, Extended Configuration Space, and other capabilities.</p>
-----	---

PCIE DEVICE

► Unique PCI Function Address

- **Bus** / **Dev** / **Function**
- Command, lspci -v, can get PCI device information on linux

```
01:00.1 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
Subsystem: Intel Corporation Ethernet Server Adapter I350-T2
Flags: fast devsel, IRQ 17
Memory at f7700000 (32-bit, non-prefetchable) [size=1M]
Memory at f7980000 (32-bit, non-prefetchable) [size=16K]
Capabilities: [40] Power Management version 3
Capabilities: [50] MSI: Enable- Count=1/1 Maskable+ 64bit+
Capabilities: [70] MSI-X: Enable- Count=10 Masked-
Capabilities: [a0] Express Endpoint, MSI 00
Capabilities: [100] Advanced Error Reporting
Capabilities: [140] Device Serial Number a0-36-9f-ff-ff-17-b1-6c
Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
Capabilities: [160] Single Root I/O Virtualization (SR-IOV)
Capabilities: [1a0] Transaction Processing Hints
Capabilities: [1d0] Access Control Services
Kernel modules: igb
```



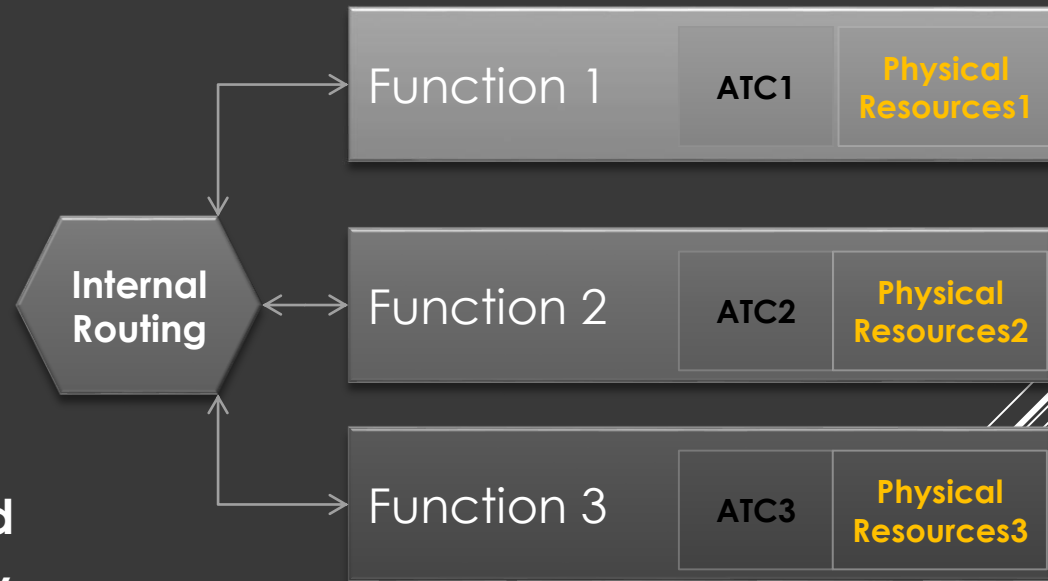
COMPONENTS IN PCIE DEVICE

► Physical Resources


- **Memory** which **allocated** from **physical** memory

► ATC - Address Translation Cache

- A **hardware** stores recently used address translations.
- This term is used instead of TLB buffer
- To differentiate the **TLB used for I/O** from the **TLB used by the CPU**

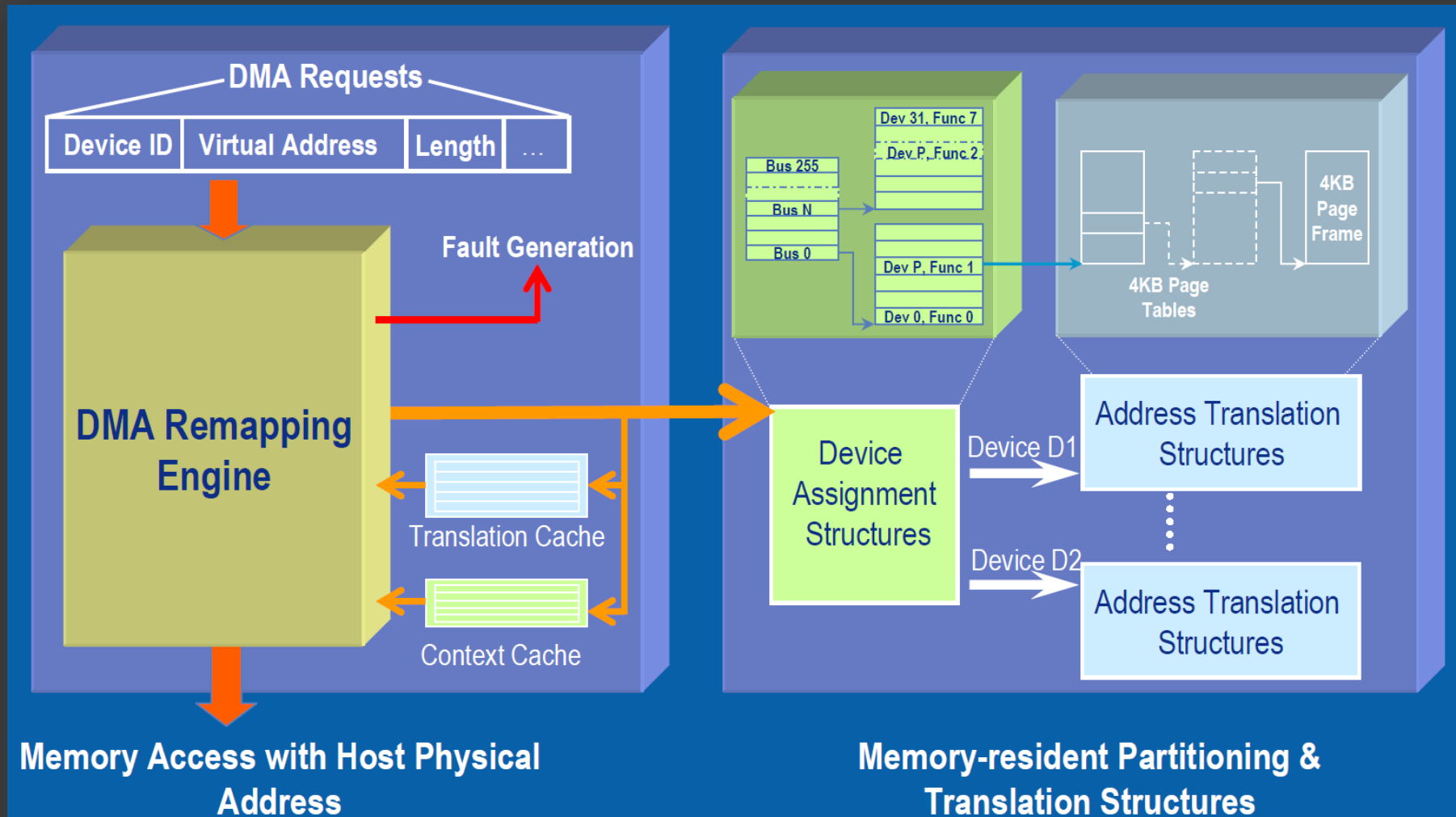


I/O VIRTUALIZATION **CONCEPT & MOTIVATION**

- ▶ **Virtualizing** the I/O **path** between a host and a PCIe device
 - ▶ **Prevent** direct access to physical (memory) resources
 - ▶ **Managing** and **separating** each device's I/O space **using a VMM**
 - ▶ Can apply to anything that uses an adapter in a computer, such as:
 - ▶ Ethernet Network Interface Cards (NICs)
 - ▶ Disk Controllers (including RAID controllers)
 - ▶ Fibre Channel Host Bus Adapters (HBAs)
 - ▶ Graphics/Video cards or co-processors
 - ▶ SSDs mounted on internal cards
- 

I/O VIRTUALIZATION SOLUTIONS

HARDWARE-BASED **DMA-REMAPPING**



IS IOMMU (VT-D) ENABLED ON YOUR SYSTEM?

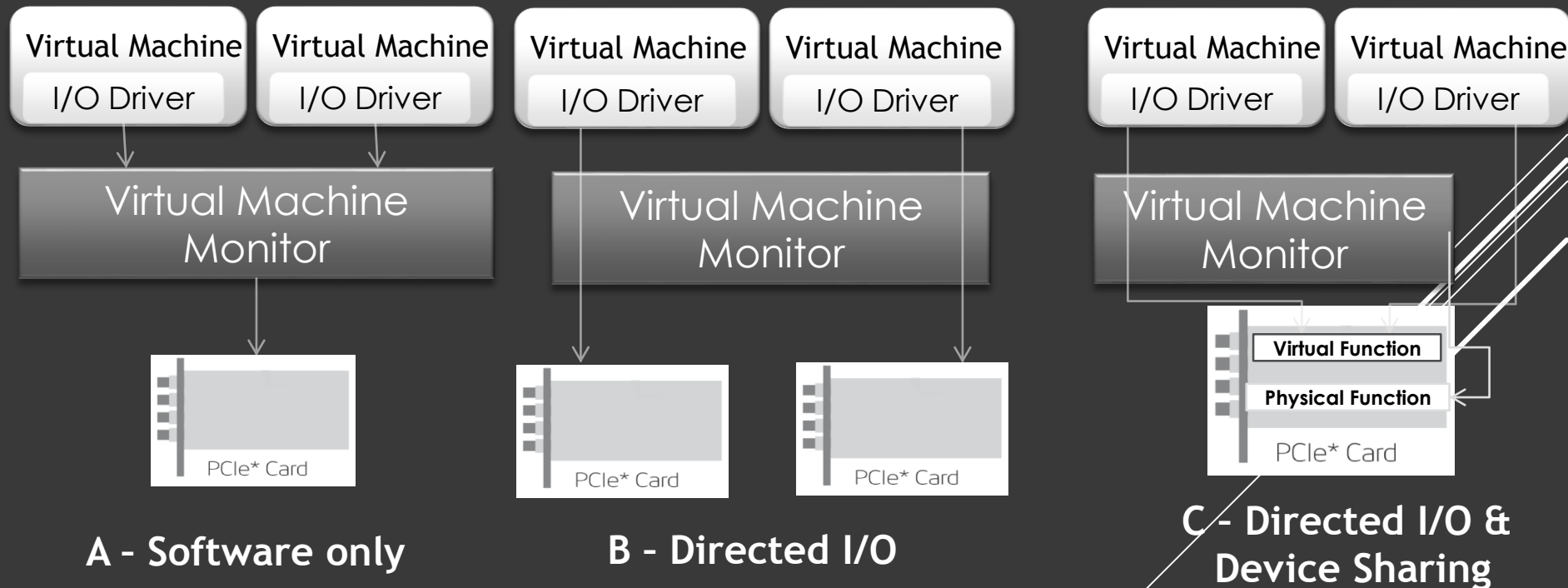
```
$ sudo dmesg | grep -e dmar -e IOMMU
[    0.000000] DMAR: IOMMU enabled
[    0.004000] DMAR: dmar0: reg_base_addr ZZZZZZZZ ver 1:0 cap XXXXXXXXXXXX ecap WWWWWW
[    0.004000] DMAR-IR: IOAPIC id 128 under DRHD base 0xYYYYYYYY IOMMU 0
[    1.192371] DMAR: dmar0: Using Queued invalidation
[    1.526166] AMD IOMMUv2 driver by Joerg Roedel <jroedel@suse.de>
[    1.526167] AMD IOMMUv2 functionality not available on this system
```

IOMMU includes:

- (a) I/O device assignment
- (b) DMA **remapping**
- (c) interrupt **remapping**
- (d) reliability features

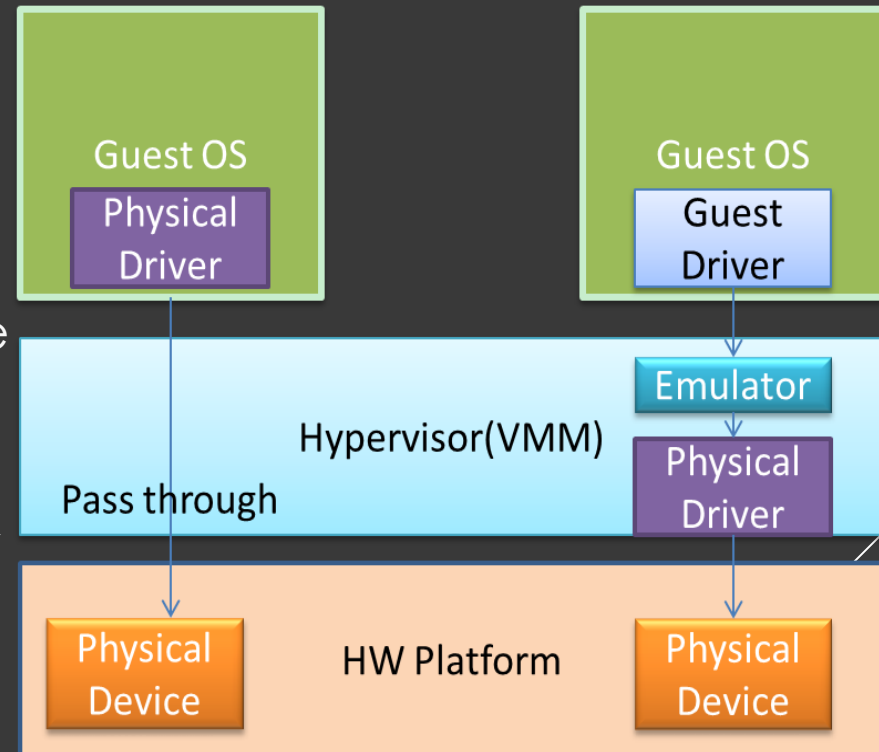
I/O VIRTUALIZATION IMPLEMENTATIONS

- ▶ A - Software only (**emulation**)
- ▶ B - Directed I/O (**enhance performance**)
- ▶ C - Directed I/O and Device Sharing (**resource saving**)



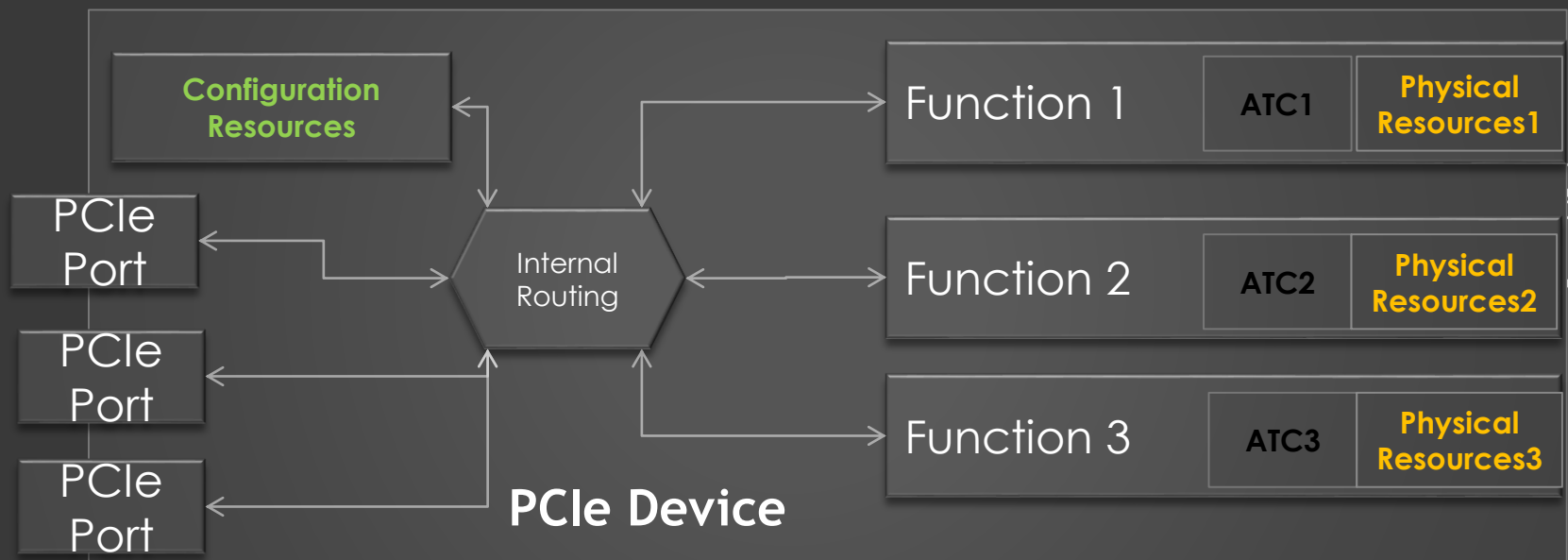
DIRECTED I/O - PASSTHROUGH

- ▶ Software-based sharing adds overhead to each I/O due to emulation layer
 - ▶ This indirection has the additional affect of eliminating the use of hardware acceleration that may be available in the physical device.
- ▶ Directed I/O has added enhancements to facilitate memory translation and ensure protection of memory that enables a device to directly DMA to/from host memory.
 - ▶ Bypass the VMM's I/O emulation layer
 - ▶ Throughput improvement for the VMs

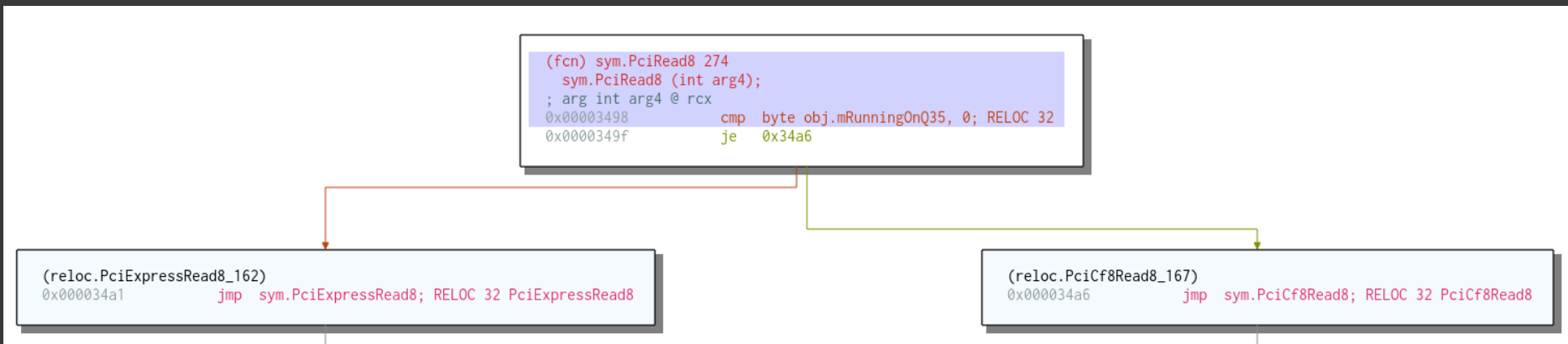


EXAMPLE: MULTI-FUNCTION DEVICE

- ▶ The link and PCIe functionality shared by all functions is managed through **Function 0**
- ▶ All functions use a **single** Bus Number captured through the PCI enumeration process (using classic PCIe features only)
- ▶ For **physically-isolated** functionality capability, each function/**port** must be **assigned** to a different CPU **core**



REVERSING PCI(E) DEVICE DRIVERS – READ REQUEST



READING PCIE DEVICE VS PCI DEVICE



PCI EXPRESS – MEMORY READ EXAMPLE

CHECK IF THE ADDRESS IS 32-BIT OR 64-BIT WIDE

```
(fcn) sym.PciExpressRead8 98
    sym.PciExpressRead8 (int arg4);
; arg int arg4 @ rcx
0x00002943    push rbx
0x00002944    mov  rbx, rcx; RELOC 32                ; arg4
0x00002947    sub  rsp, 0x20; RELOC 64
(reloc.DebugAssertEnabled_76)
0x0000294b    call sym.DebugAssertEnabled; RELOC 32 DebugAssertEnabled
0x00002950    test al, al; RELOC 64
0x00002952    je   0x2975
```

```
0x00002954    test rbx, 0xffffffff00000000; RELOC 64
0x0000295b    je   0x2975; RELOC 32
```

FOR A 32BIT BAR, PREPARE BASE+OFFSET AND PERFORM MMIOREAD8()



(reloc._gPcd_FixedAtBuild_PcdPciExpressBaseAddress_120)

0x00002975 add rbx, qword [obj._gPcd_FixedAtBuild_PcdPciExpressBaseAddress]; RELOC 32 ...

0x0000297c add rsp, 0x20; RELOC 32


0x00002980 mov rcx, rbx; RELOC 64

0x00002983 pop rbx; RELOC 32

(reloc.MmioRead8_133)

0x00002984 jmp sym.MmioRead8; RELOC 32 MmioRead8

RCX IS TRANSFORMED AS A BASE+OFFSET READ CONTENT AND RETURN



```
(fcn) sym.MmioRead8 28
  sym.MmioRead8 (int arg4);
; arg int arg4 @ rcx
0x00002106      push rbx
0x00002107      mov  rbx, rcx                                ; arg4
0x0000210a      sub  rsp, 0x20; RELOC 32
(reloc.MemoryFence_15)
0x0000210e      call sym.MemoryFence; RELOC 32 MemoryFence
0x00002113      mov  bl, byte [rbx]; RELOC 64
(reloc.MemoryFence_22)
0x00002115      call sym.MemoryFence; RELOC 32 MemoryFence
0x0000211a      add  rsp, 0x20; RELOC 32
0x0000211e      mov  eax, ebx
0x00002120      pop  rbx; RELOC 64
0x00002121      ret
```

COMPONENTS OF SR-IOV - **AT**

► **TA** – Translation Agent

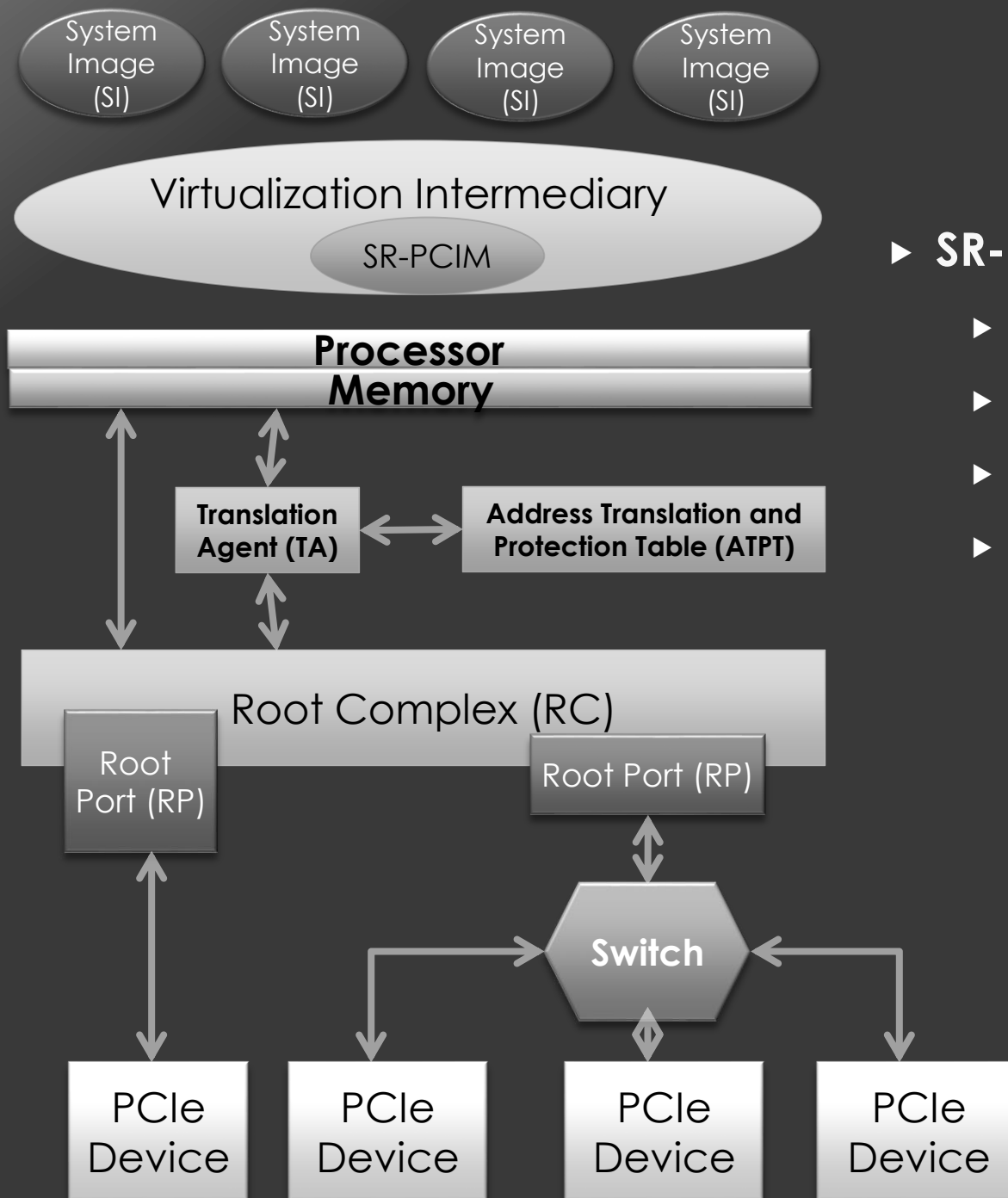


- **Translate address** within a **PCIe transaction** into the associated platform **physical address**.
- Hardware or combination of hardware and software
- A TA may also support to enable a PCIe function to obtain address translations **a priori to DMA access** to the associated memory.

COMPONENTS OF SR-IOV - **ATPT**

- ▶ ATPT – **A**ddress **T**ranslation and **P**rotection **T**able
 - ▶ Contain the set of address translations accessed by a TA to Process PCIe requests
 - ▶ DMA Read/Write
 - ▶ Interrupt requests
- ▶ DMA Read/Write requests are translated through a combination of the **Routing ID** and **the address contained within a PCIe transaction**
- ▶ In PCIe, interrupts are treated as memory write operations.
 - ▶ Though the combination of the Routing ID and the address contained within a PCIe transaction as well



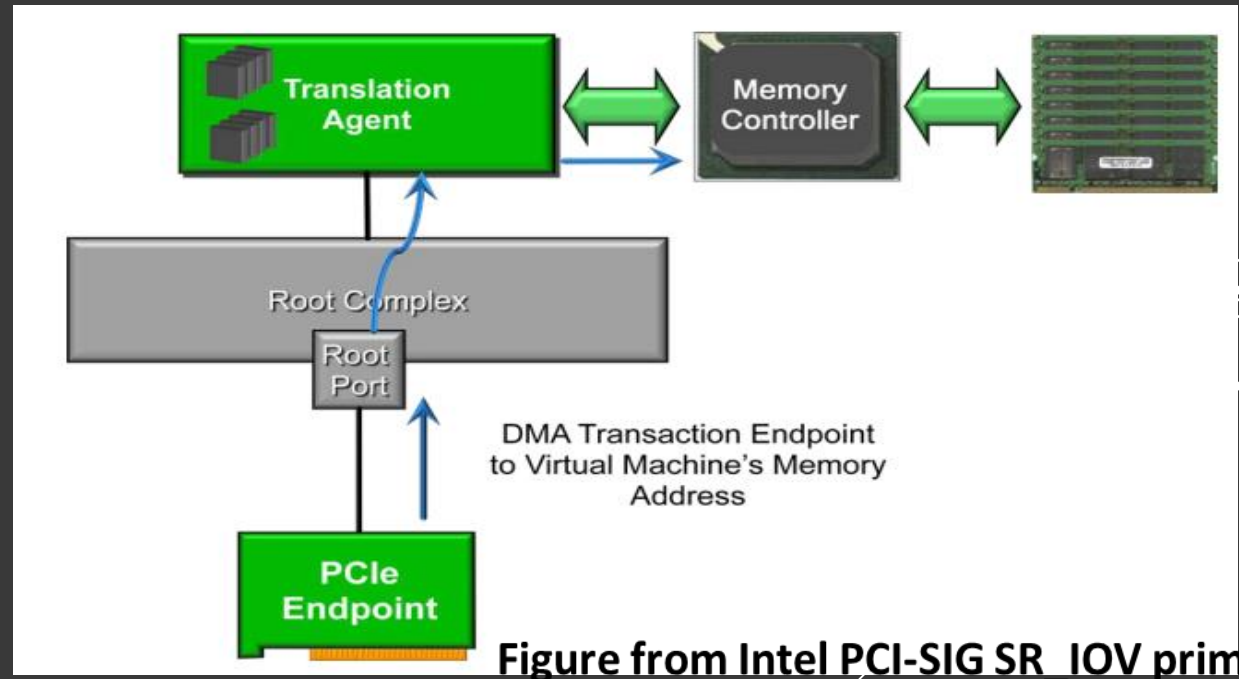


► SR-PCIM

- Configure SR-IOV Capability
- Management of PFs and VFs
- Processing of error events
- Device controls
 - Power management
 - Hot-plug

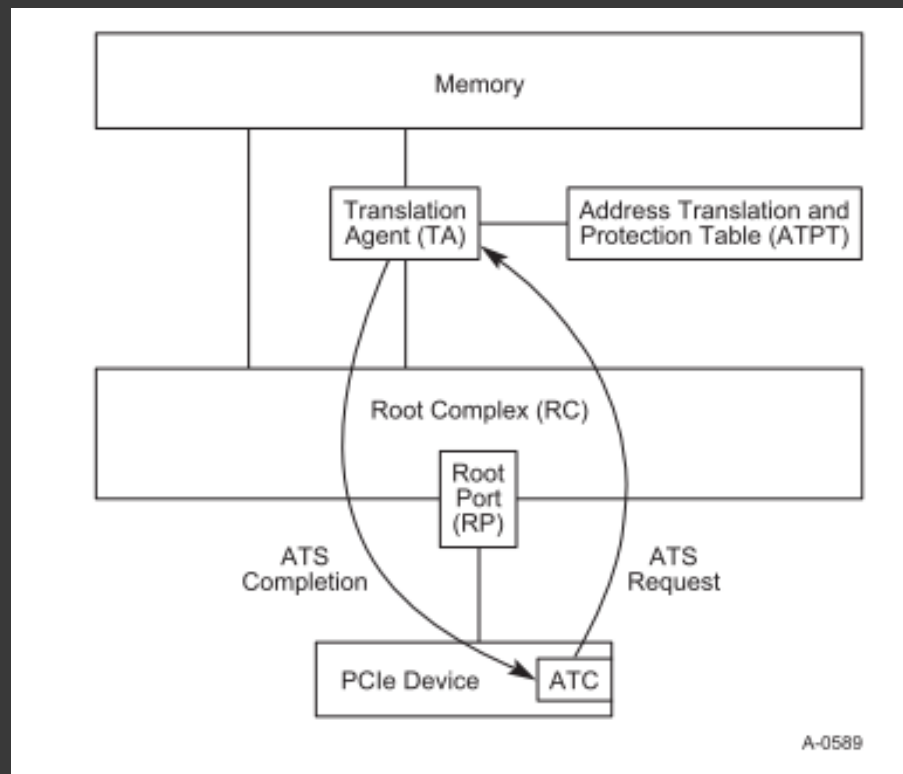
ATS – ADDRESS TRANSLATION SERVICES

- ▶ ATS provides a mechanism allowing a virtual machine to perform **DMA transaction directly to and from a PCIe endpoint**.




ATS – ADDRESS TRANSLATION SERVICES

- ▶ ATS uses a **request-completion** protocol between a Device and a Root Complex (RC)



ATS – ADDRESS TRANSLATION SERVICES

- ▶ When the Function receives the ATS Translation Completion
 - ▶ Either updates its ATC to reflect the translation
 - ▶ Or notes that a translation does not exist.
 - ▶ The Function generates subsequent requests using
 - ▶ Either a translated address
 - ▶ Or an un-translated address based on the results of the Completion.
- 
- A series of several parallel white diagonal lines in the bottom right corner of the slide, extending from the middle of the right edge towards the bottom left.

ARI – ALTERNATIVE ROUTING ID INTERPRETATION

- ▶ Routing ID is used to forward requests to the corresponding PFs and VFs
- ▶ All VFs and PFs must have distinct Routing IDs
- ▶ ARI provides a mechanism to allow single PCIe component to support **up to 256 functions**.
 - ▶ Originally there are **8 functions** at most in a PCIe.

15 ----- 8	7-----3	2-----0
Bus #	Device #	Function #

Table 1. Traditional Routing

15 ----- 8	7 ----- 0
Bus #	Identifier

Table 2. Alternate Routing

Figure from Intel PCI-SIG SR_IOV prim

ARI – ALTERNATIVE ROUTING ID INTERPRETATION

Table 2-1: VF Routing ID Algorithm

VF Number	VF Routing ID
VF 1	$(\text{PF Routing ID} + \text{First VF Offset}) \text{ Modulo } 2^{16}$
VF 2	$(\text{PF Routing ID} + \text{First VF Offset} + \text{VF Stride}) \text{ Modulo } 2^{16}$
VF 3	$(\text{PF Routing ID} + \text{First VF Offset} + 2 * \text{VF Stride}) \text{ Modulo } 2^{16}$
...	...
VF N	$(\text{PF Routing ID} + \text{First VF Offset} + (N-1) * \text{VF Stride}) \text{ Modulo } 2^{16}$
...	...
VF NumVFs (last one)	$(\text{PF Routing ID} + \text{First VF Offset} + (\text{NumVFs}-1) * \text{VF Stride}) \text{ Modulo } 2^{16}$

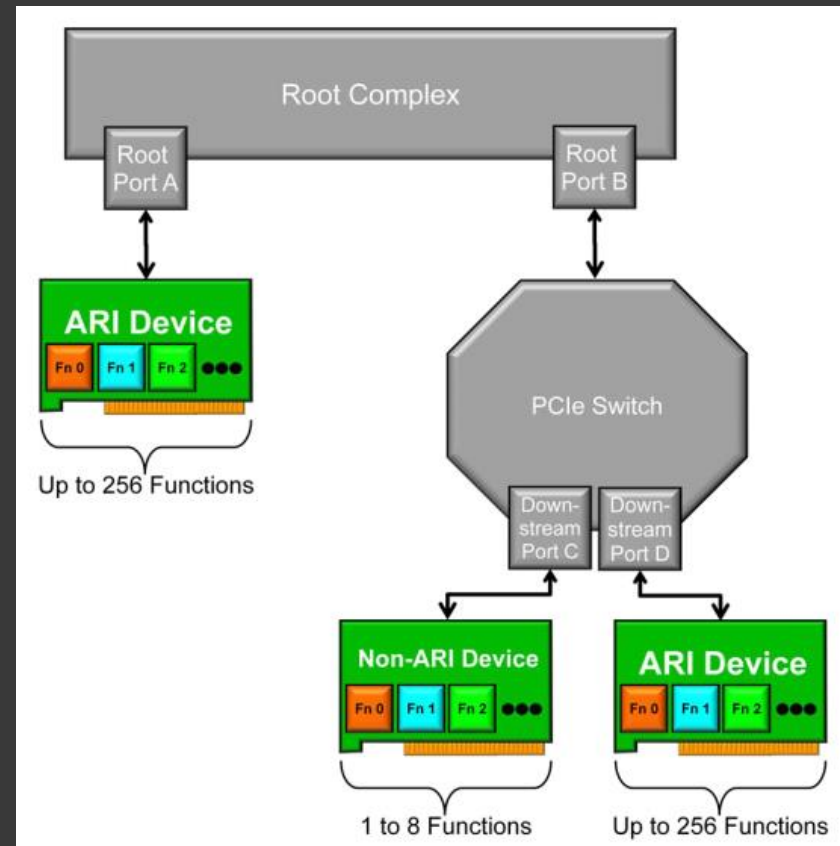


Figure from Intel PCI-SIG SR_IOV prim

Figure from SR-IOV Specification revision 1.1

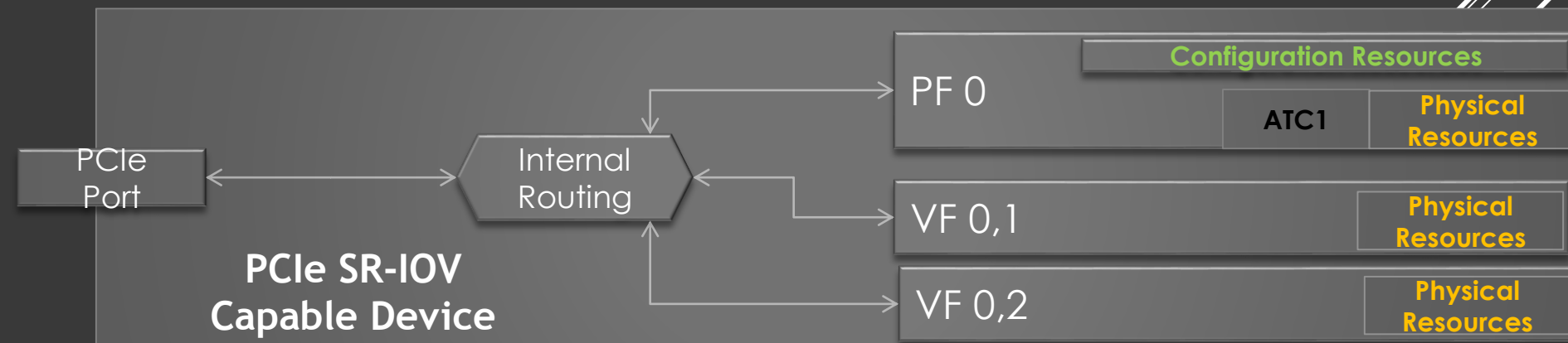
SR-IOV CONFIGURATION SPACE MAPPING

- ❑ Determine desired number of Virtual Functions from *InitialVFs* field
- ❑ Program *NumVFs* field to match
- ❑ Multi-Root adds a further layer where configuration software **first** allocates VFs to Virtual Hierarchies – thus *InitialVFs* may be less than *TotalVFs*

31	24	23	20	19	16	15	0	Byte Offset
Next Capability Offset		Capability Version		PCI Express Extended Capability ID				00h
SR-IOV Capabilities								04h
SR-IOV Status				SR-IOV Control				08h
Total VFs (RO)				Initial VFs (RO)				0Ch
RsvdP		Function Dependency Link (RO)		Num VFs (RW)				10h
VF Stride (RO)				First VF Offset (RO)				14h
VF Device ID (RO)				RsvdP				18h
Supported Page Sizes (RO)								1Ch
System Page Size (RW)								20h
VF BAR0 (RW)								24h
VF BAR1 (RW)								28h
VF BAR2 (RW)								2Ch
VF BAR3 (RW)								30h
VF BAR4 (RW)								34h
VF BAR5 (RW)								38h
VF Migration State Array Offset (RO)								3Ch

PCIE SR-IOV CAPABLE DEVICE

- ▶ **SR-IOV (Single Root I/O Virtualization)**
 - ▶ A technique performs and manages **PCle Virtualization**.
- ▶ **PF – physical Function**
 - ▶ Provide full PCIe functionality, **including** the **SR-IOV capabilities**
 - ▶ Discover the **page sizes** supported by a PF and its associated VF
- ▶ **VF – virtual Function**
 - ▶ A “**light-weight**” PCIe function that is **directly accessible** by a **system image (SI)**
 - ▶ including an **isolated** memory space, a work queue, interrupts and command processing.
 - ▶ Including **data movement extensions** capabilities (HW dependent)
 - ▶ Can be **optionally migrated** from one PF to another PF (using MR-IOV)
 - ▶ Can be serially shared by different SI



SR-IOV EXTENDED CAPABILITIES

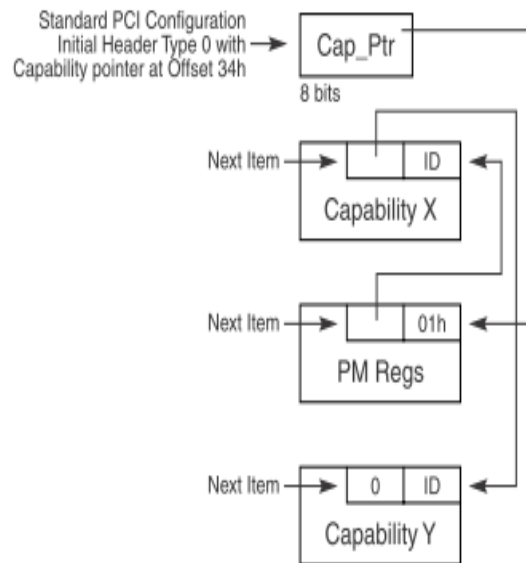


Figure 3-2: Capabilities Linked List

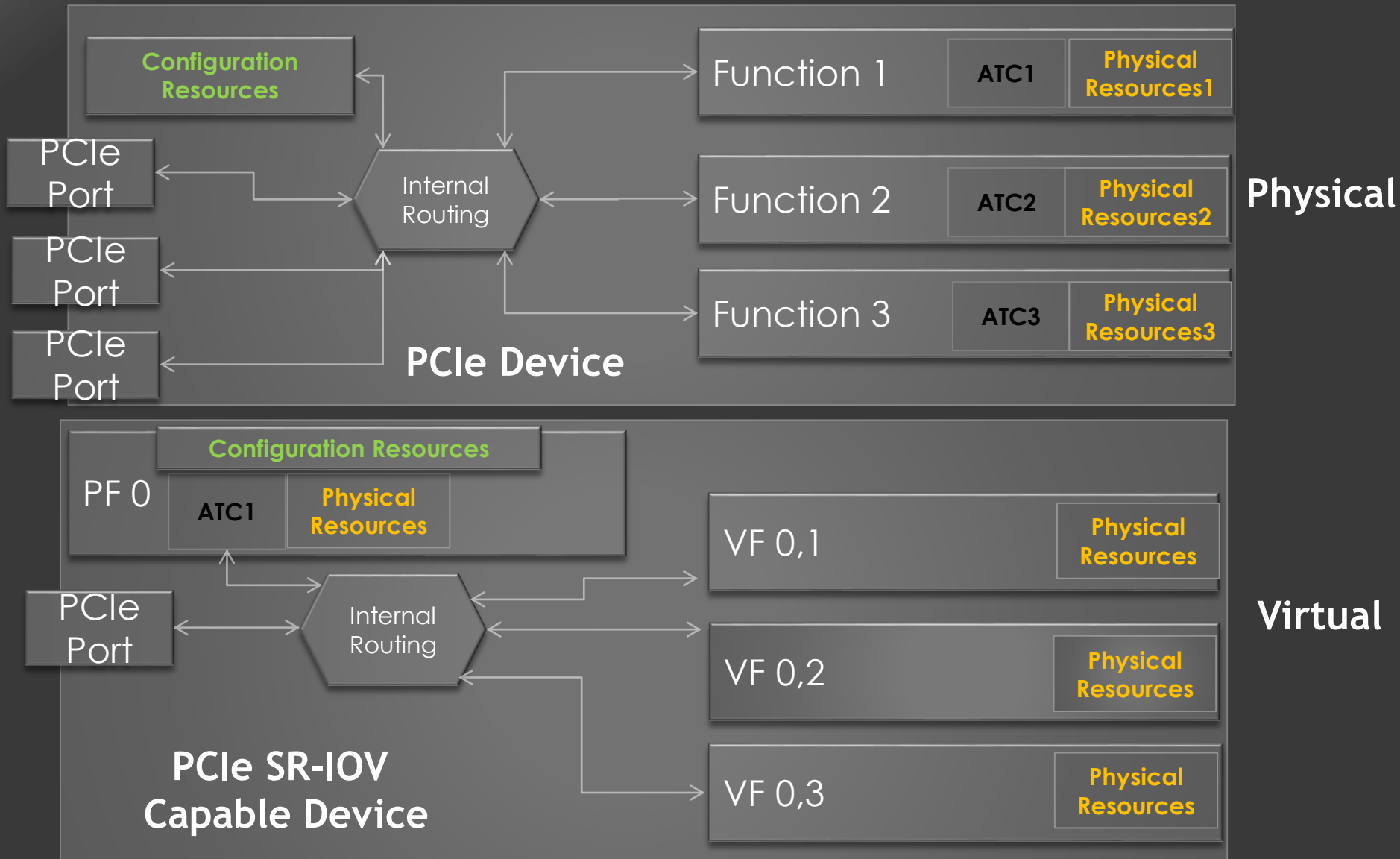
A-0318

31	24	23	20	19	16	15	0	Byte Offset
Next Capability Offset			Capability Version		PCI Express Extended Capability ID			00h
SR-IOV Capabilities								04h
SR-IOV Status				SR-IOV Control				08h
TotalVFs (RO)				InitialVFs (RO)				0Ch
RsvdP		Function Dependency Link (RO)		NumVFs (RW)				10h
VF Stride (RO)				First VF Offset (RO)				14h
VF Device ID (RO)				RsvdP				18h
Supported Page Sizes (RO)								1Ch
System Page Size (RW)								20h
VF BAR0 (RW)								24h
VF BAR1 (RW)								28h
VF BAR2 (RW)								2Ch
VF BAR3 (RW)								30h
VF BAR4 (RW)								34h
VF BAR5 (RW)								38h
VF Migration State Array Offset (RO)								3Ch

A-0634A

Figure 3-1: Single Root I/O Virtualization Extended Capabilities

FUNCTIONS: PHYSICAL VS. VIRTUAL

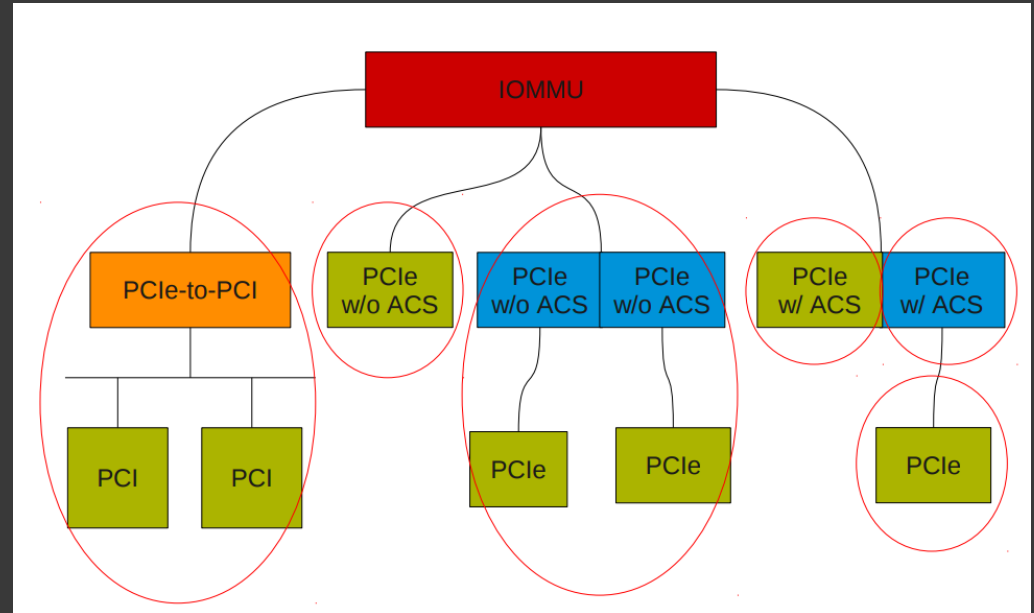


VFIO – USER-SPACE VIRTUAL FUNCTION I/O

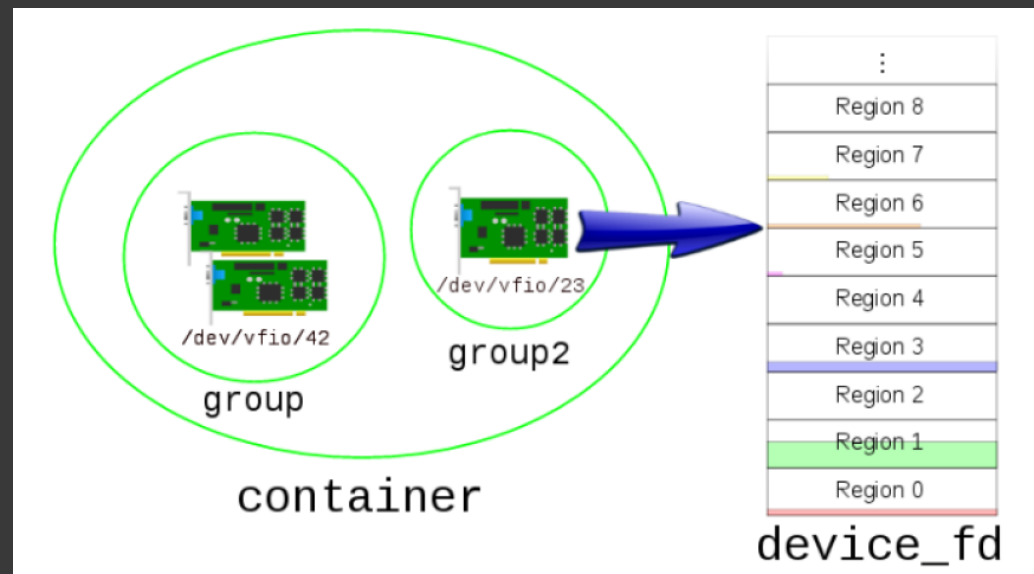
- ▶ A **user level driver** framework for Linux
- ▶ Originally developed by Tom Lyon (Cisco)
- ▶ **IOMMU-based DMA** and interrupt isolation
- ▶ Full devices access (MMIO, I/O port, PCI config)
- ▶ Efficient interrupt mechanisms
- ▶ Modular IOMMU and device backends

HW-BASED ISOLATION **GROUPS** USING IOMMU

- ▶ **IOMMU** is in charge of “**isolation GROUPS**”

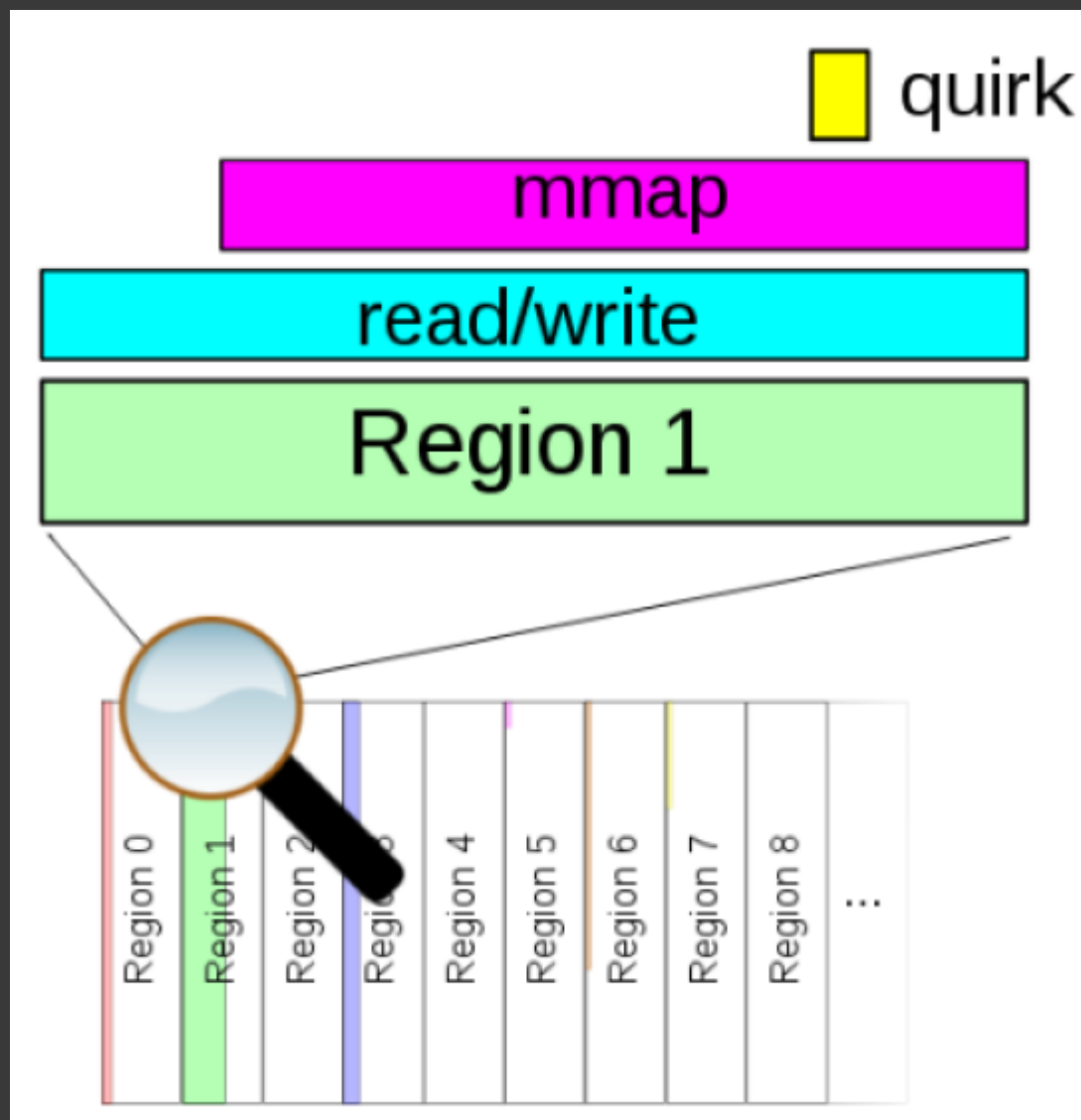


- ▶ VFIO moves **ownership** to the group level

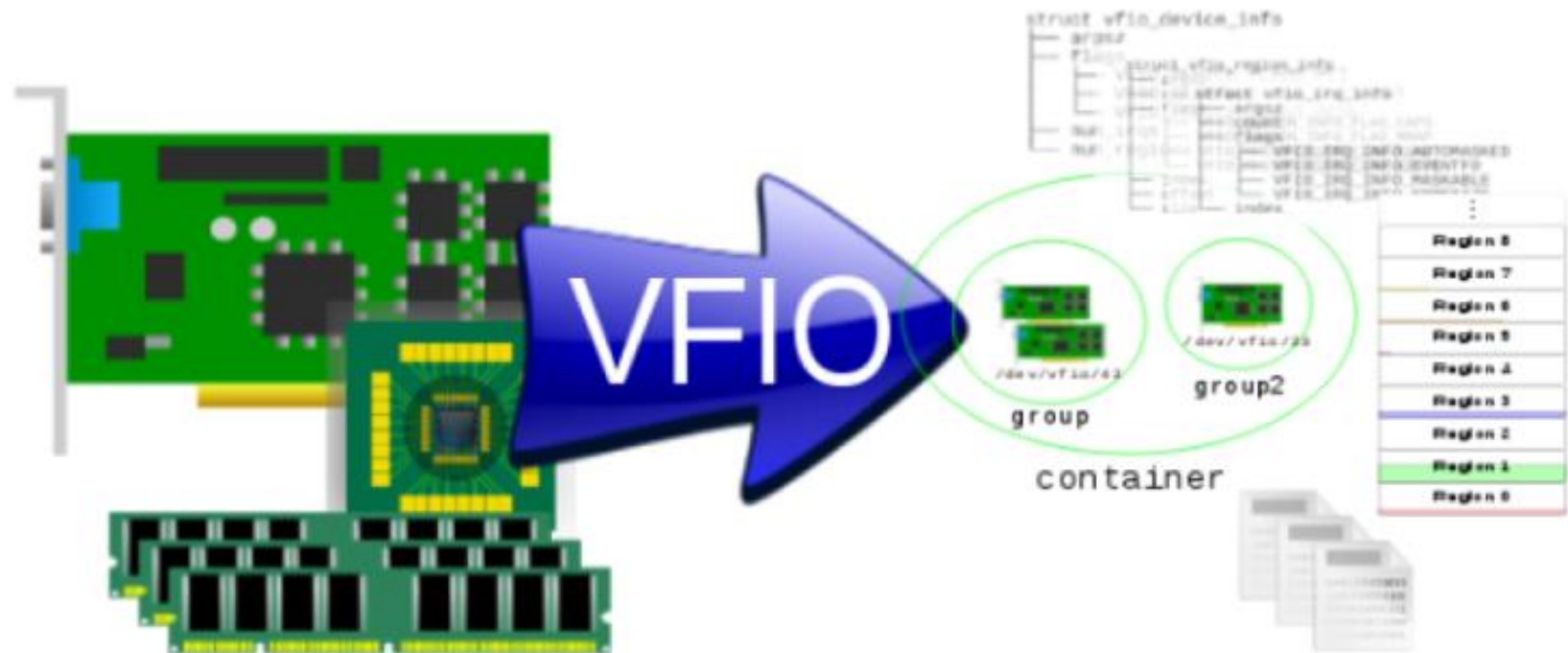


SR-IOV CAPABILITIES IMPLEMENTATION USING VFIO

- ▶ Access to device **file** grants ownership
- ▶ Ownership extends to all devices within the group
- ▶ All accesses **through VFIO**



VFIO IN A NUTSHELL



REVERSE ENGINEERING SR-IOV PROCESS – PROBE VFS OF A PF AND CREATE GROUP

```
(fcn) sym.vfio_pci_probe 333
    sym.vfio_pci_probe (int arg_58h);
; arg int arg_58h @ rbp+0x58
0x080013c0      call 0x80013c5; RELOC 32 __fentry__
0x080013c5      push r14
0x080013c7      push r13
0x080013c9      push r12
0x080013cb      push rbp
0x080013cc      push rbx
0x080013cd      cmp byte [rdi + 0x49], 0
0x080013d1      jne 0x80014ec
```

```
0x080013d7      lea r12, [rdi + 0xa0] ; 160
0x080013de      mov rbx, rdi
0x080013e1      mov rdi, r12
(reloc.vfio_iommu_group_get)
0x080013e4      call 0x80013e9; RELOC 32 vfio_iommu_group_get
0x080013e9      test rax, rax
0x080013ec      mov r14, rax
0x080013ef      je 0x80014ec
```

IF THE **DEVICE** INCLUDES SR-IOV CAPABILITY, TRACE VFS INSIDE A PF AND ALLOCATE MEM

```
0x080013d7      lea r12, [rdi + 0xa0]                ; 160
0x080013de      mov rbx, rdi
0x080013e1      mov rdi, r12
(reloc.vfio_iommu_group_get)
0x080013e4      call 0x80013e9; RELOC 32 vfio_iommu_group_get
0x080013e9      test rax, rax
0x080013ec      mov r14, rax
0x080013ef      je 0x80014ec
```

```
0x080013f5      mov rdi, qword [0x080013fc]           ; [0x080013fc:8]=0x80c0be000...
0x080013fc      mov edx, 0x100                       ; 256
0x08001401      mov esi, 0x14080c0
0x08001406      call 0x800140b; RELOC 32 kmem_cache_alloc_trace
0x0800140b      test rax, rax
0x0800140e      mov rbp, rax
0x08001411      je 0x80014f7
```

```
0x080014ec      mov r13d, 0xfffffffffa
0x080014f2      jmp 0x8001471
```

IF WE FIND A VF, ADD IT TO THE **GROUP** – ALLOCATE RESOURCES ACCORDING TO IOMMU

0x0800140e
0x08001411

mov rbp, rax
je 0x80014f7

```
0x08001417    lea rdi, [rax + 0x60]                ; '' ; 96
0x0800141b    mov qword [rax], rbx
0x0800141e    mov dword [rax + 0x8c], 5
0x08001428    mov rdx, 0; RELOC 32
0x0800142f    mov rsi, 0; RELOC 32
(reloc.__mutex_init)
0x08001436    call 0x800143b; RELOC 32 __mutex_init
0x0800143b    mov dword [arg_58h], 0          ; [0x58:4]=-1 ; 'X' ; 0
0x08001442    mov rdx, rbp
0x08001445    mov rsi, 0; RELOC 32
0x0800144c    mov rdi, r12
(reloc.vfio_add_group_dev)
0x0800144f    call 0x8001454; RELOC 32 vfio_add_group_dev
0x08001454    test eax, eax
0x08001456    mov r13d, eax
0x08001459    jne 0x80014a0
```

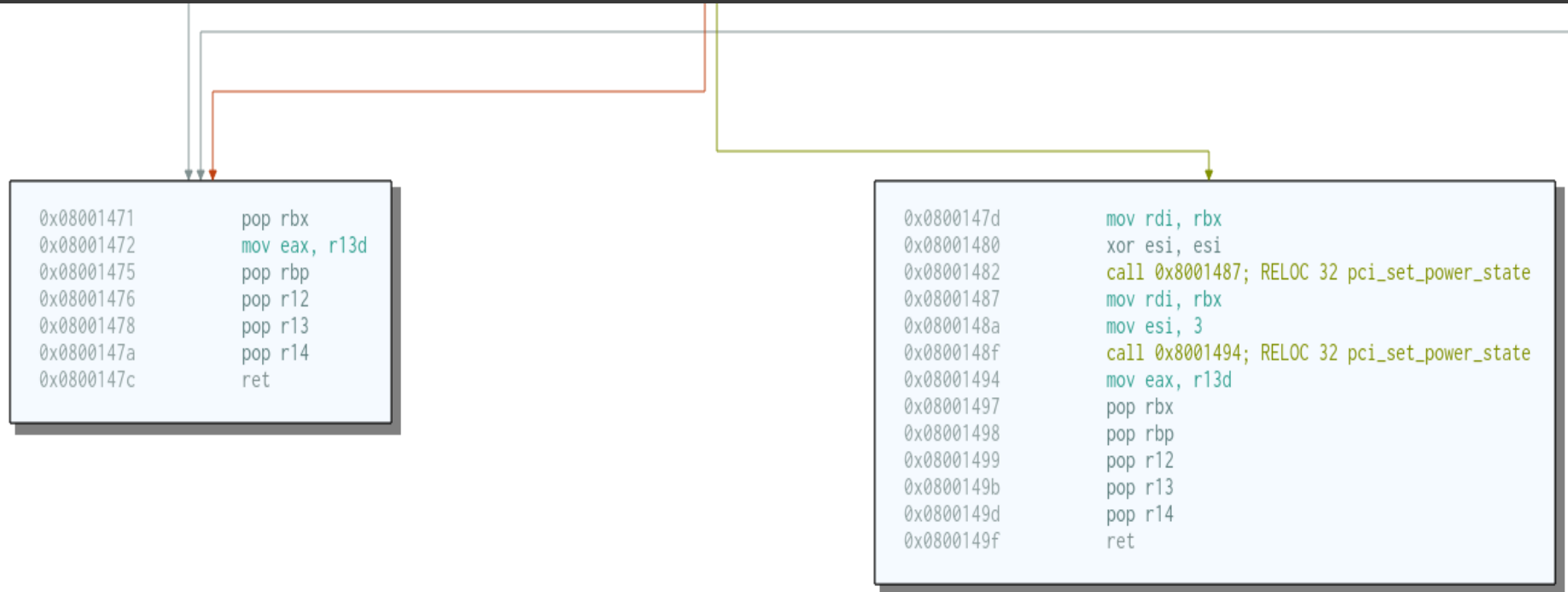
0x080014f7
0x080014fa
0x080014fd
(reloc.vfio_...
0x08001503
0x08001508

CHECK FOR SPECIAL CAPABILITIES (VGA PASSTHROUGH, ETC.) OF THAT VF

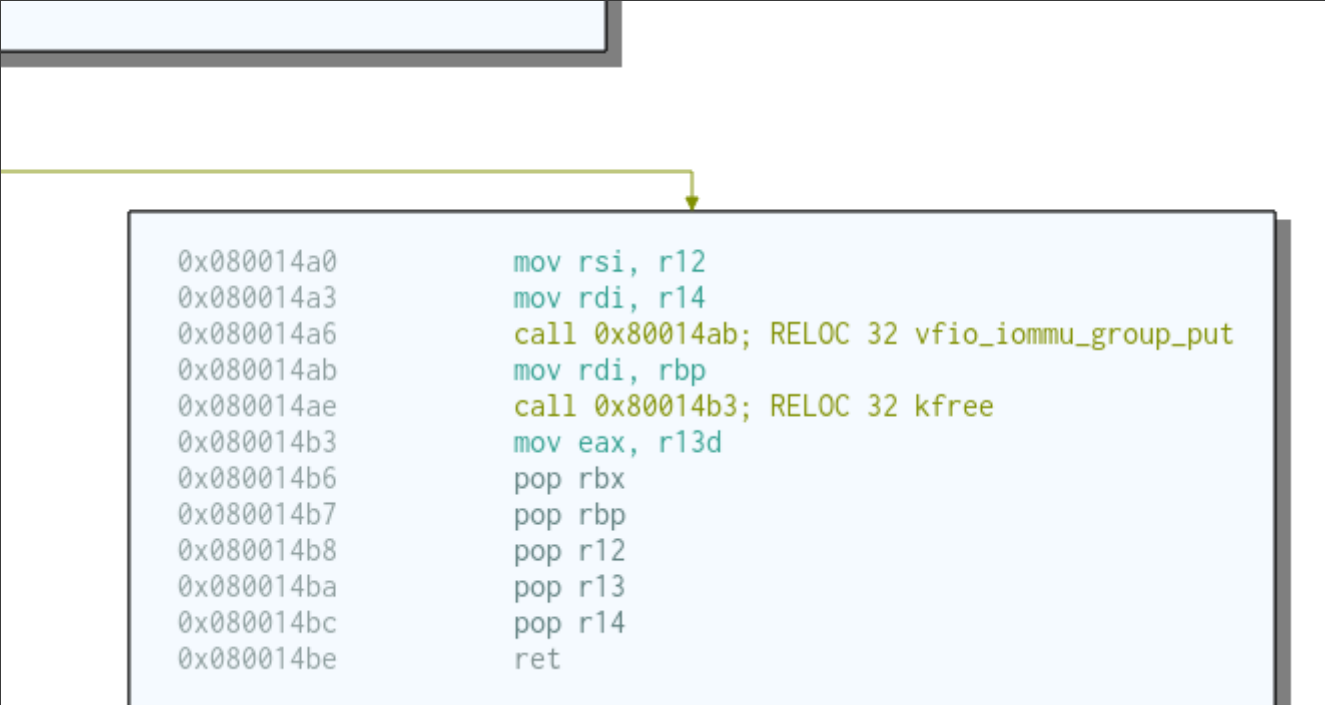
```
0x0800145b    mov eax, dword [rbx + 0x44]                ; [0x44:4]=-1 ; 'D' ; 68
0x0800145e    shr eax, 8
0x08001461    cmp eax, 0x300                            ; 768
0x08001466    je 0x80014bf
```

```
0x080014bf    mov rcx, 0; RELOC 32
0x080014c6    xor edx, edx
0x080014c8    mov rsi, rbp
0x080014cb    mov rdi, rbx
(reloc.vga_client_register)
0x080014ce    call 0x80014d3; RELOC 32 vga_client_register
0x080014d3    xor esi, esi
0x080014d5    mov rdi, rbp
0x080014d8    call sym.vfio_pci_set_vga_decode
0x080014dd    mov rdi, rbx
0x080014e0    mov esi, eax
(reloc.vga_set_legacy_decoding)
0x080014e2    call 0x80014e7; RELOC 32 vga_set_legacy_decoding
0x080014e7    jmp 0x8001468
```

SET POWER STATE (ANOTHER CAPABILITY) AND RETURN TO LOOK FOR MORE VFS



IF WE DON'T FIND ADDITIONAL VFS, CLOSE GROUP ASSIGNMENT AND RETURN GROUP



```
0x080014a0    mov rsi, r12
0x080014a3    mov rdi, r14
0x080014a6    call 0x80014ab; RELOC 32 vfio_iommu_group_put
0x080014ab    mov rdi, rbp
0x080014ae    call 0x80014b3; RELOC 32 kfree
0x080014b3    mov eax, r13d
0x080014b6    pop rbx
0x080014b7    pop rbp
0x080014b8    pop r12
0x080014ba    pop r13
0x080014bc    pop r14
0x080014be    ret
```

PROFIT

- Reversing PCIe SR-IOV is easy – **if** you know what you're looking for
- When you design your PCIe driver – **always** check that its capabilities are **working properly** (look for DMA and unknown mappings)

```
0x0000444d    mov ecx, dword [n]
0x00004453    mov rdx, qword [local_938h]
0x0000445a    mov rsi, qword [local_930h]
0x00004461    mov rax, qword [local_928h]
0x00004468    mov r8d, 0
0x0000446e    mov rdi, rax
0x00004471    call sym.DeviceReadDMA
0x00004476    test eax, eax
0x00004478    jne 0x44a7
```

```
0x0000447a    mov ecx, dword [n]
0x00004480    mov rdx, qword [local_938h]
0x00004487    mov rsi, qword [local_930h]
0x0000448e    mov rax, qword [local_928h]
0x00004495    mov r8d, 0
0x0000449b    mov rdi, rax
0x0000449e    call sym.DeviceReadDMA
0x000044a3    test eax, eax
0x000044a5    je 0x44ae
```

SUM-UP - EXAMPLE

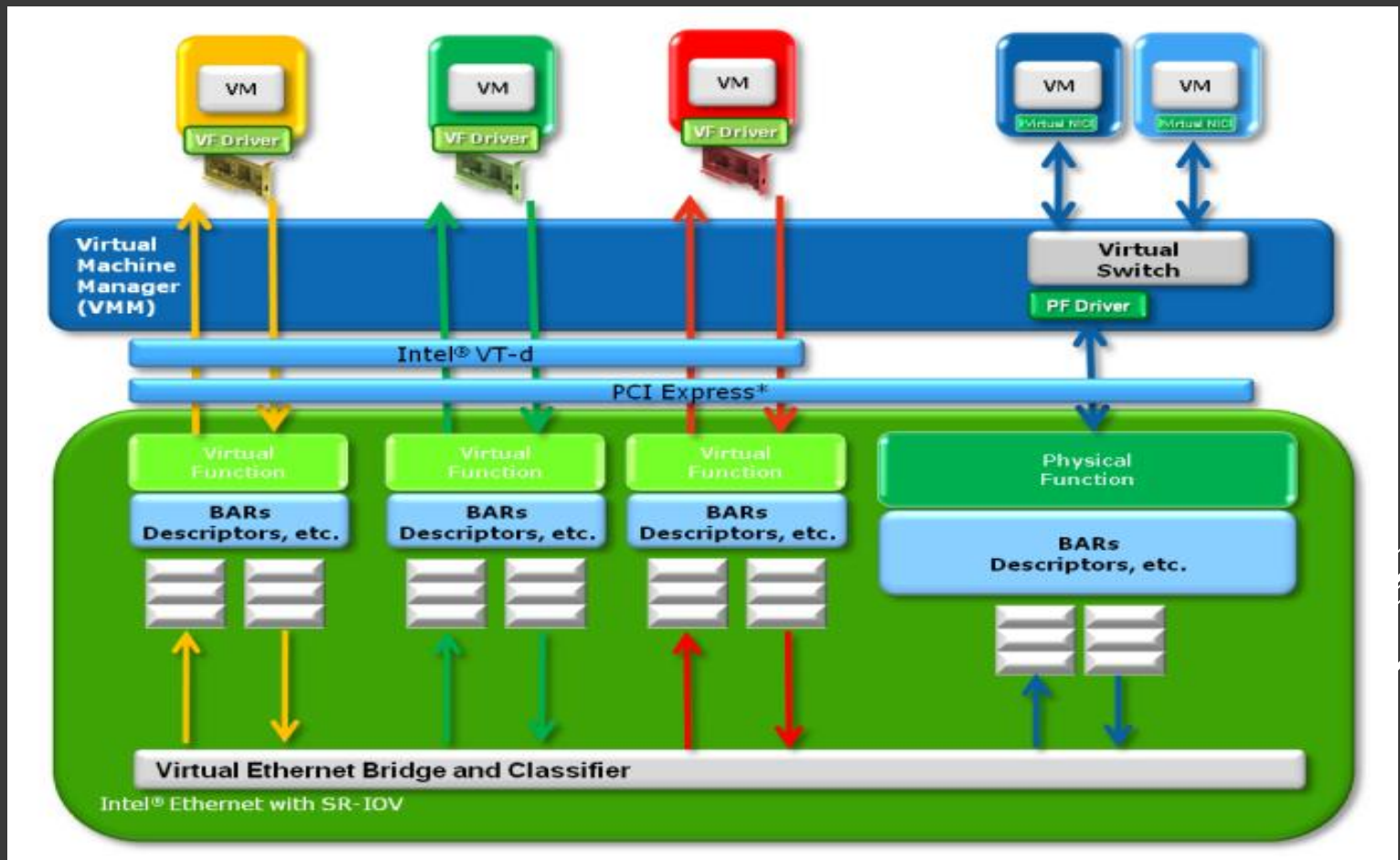
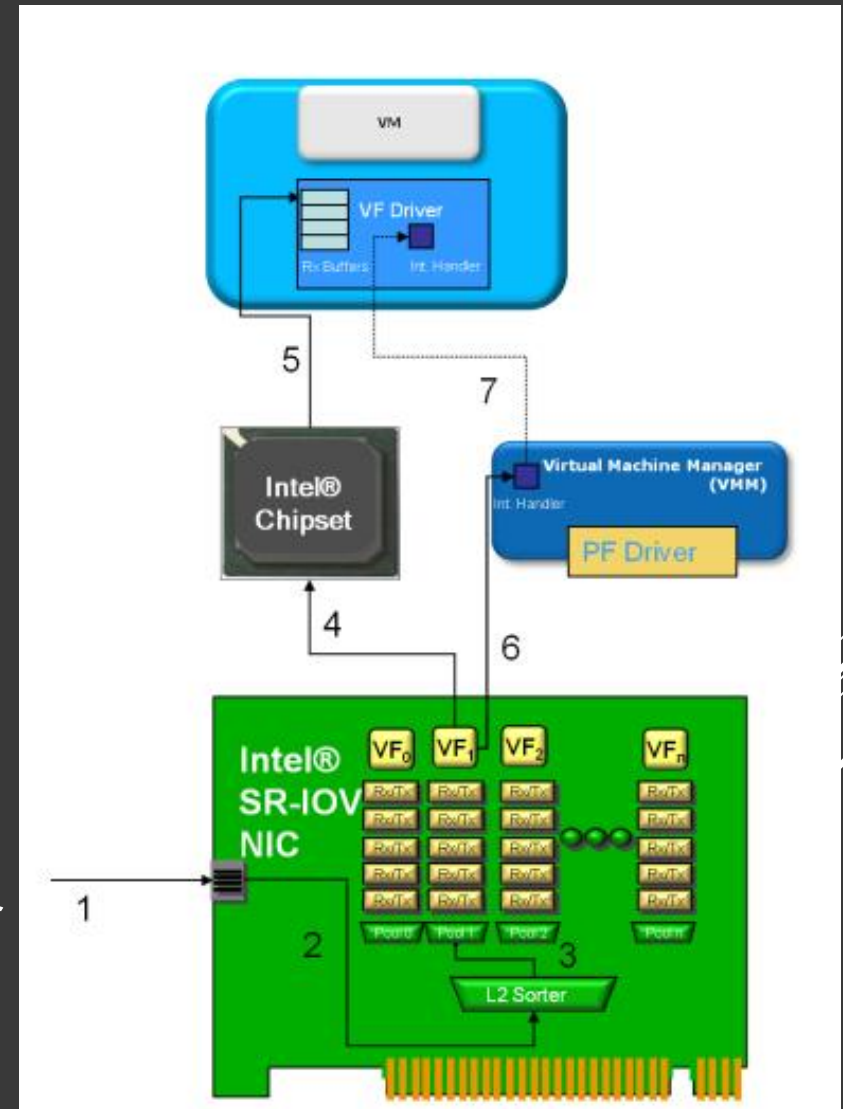


Figure from Inter PCI-SIG SR-IOV Primer

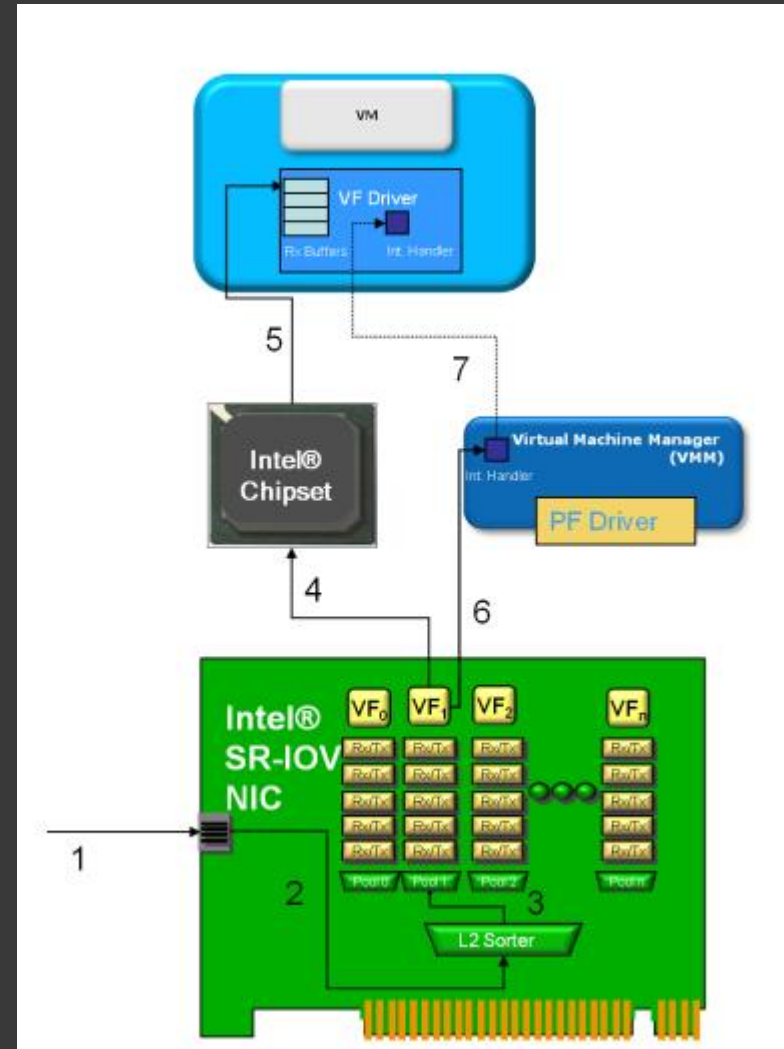
PCIE DATA PATH FOR INCOMING PACKETS

1. The Ethernet packet arrives at the Ethernet NIC
2. The packet is sent to the Layer 2 sorter/switch/classifier
 - This Layer 2 sorter is configured by the Master Driver. When either the MD or the VF Driver configure a **MAC address** or **VLAN**, this Layer 2 sorter is configured.



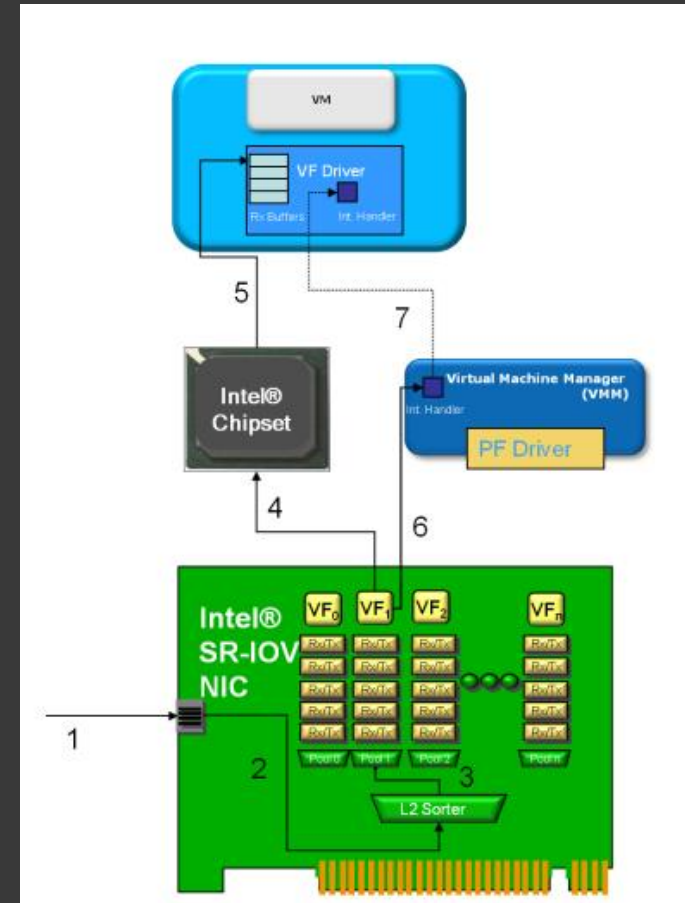
DATA PATH FOR INCOMING PACKETS

3. After being sorted by the Layer 2 Switch, the packet is placed into a receive queue dedicated to the target VF.
4. The DMA operation is initiated. The target memory address for the DMA operation is defined within the descriptors in the VF, which have been configured by the VF driver within the VM.



DATA PATH FOR INCOMING PACKETS

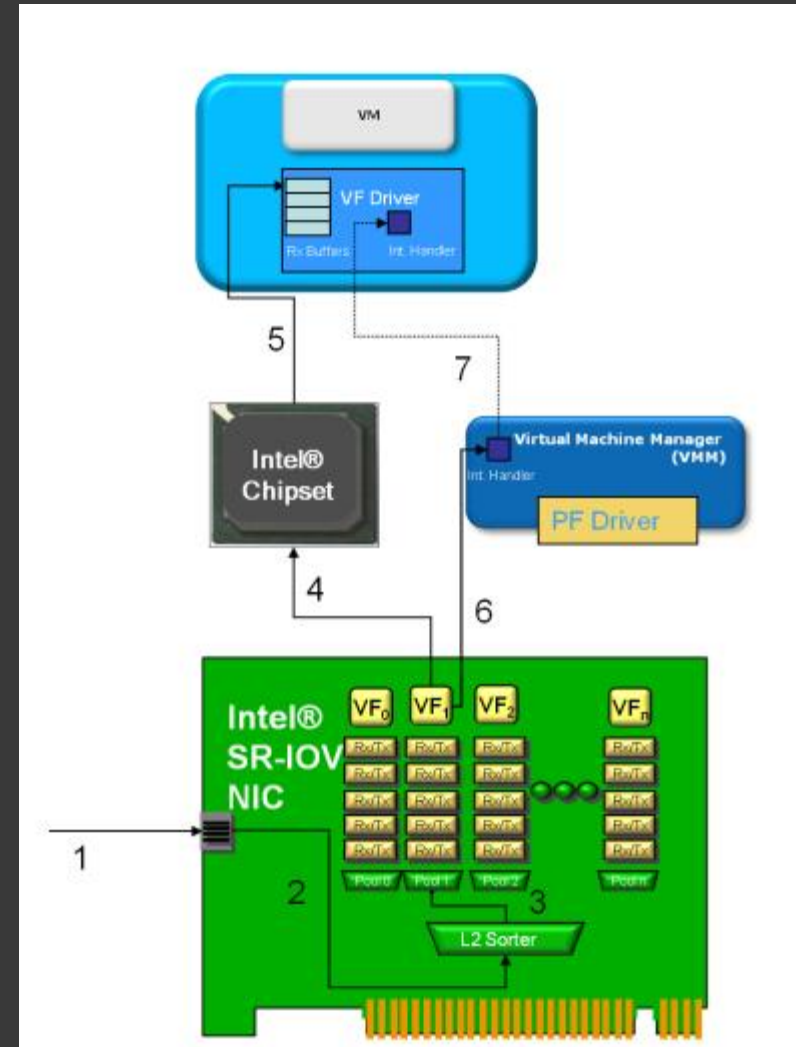
5. The DMA Operation has reached the chipset. Intel VT-d, which has been configured by the **VMM then remaps the target DMA address from a virtual host address to a physical host address.** The DMA operation is completed; the Ethernet packet is now in the memory space of the VM



DATA PATH FOR INCOMING PACKETS

6. The NIC **fires interrupt**, indicating a packet has arrived. This interrupt is handled by the VMM

7. The VMM fires a virtual interrupt to the VM, so that it is informed that the packet has arrived



NOW YOU CAN TOO!

- Find your favorite PCIe device
- Look for its device driver
- Check how the device is enumerated
- Look for its capabilities
- Follow the data flow and...
- **Hack away!**


THANKS!

▶ Adir Abraham

▶ Twitter: [@adirab](#)

▶ LinkedIn: <https://linkedin.com/in/adirab>

REFERENCE

- PCI-SIG
 - PCI Express Base Specification v4.0 Revision 1.0
 - Single Root I/O Virtualization and Sharing Specification Revision 1.1
 - Conventional PCI Local Bus Specification rev. 3.0
 - Address Translation Services Revision 1.1
 - Intel PCI-SIG SR-IOV Primer
 - opensecuritytraining.info
 - VFIO: A User's perspective
- 
- A series of four parallel white lines of varying lengths, slanted diagonally from the bottom right towards the center of the slide.

BACKUP

“VANILLA” PCIE DEVICE

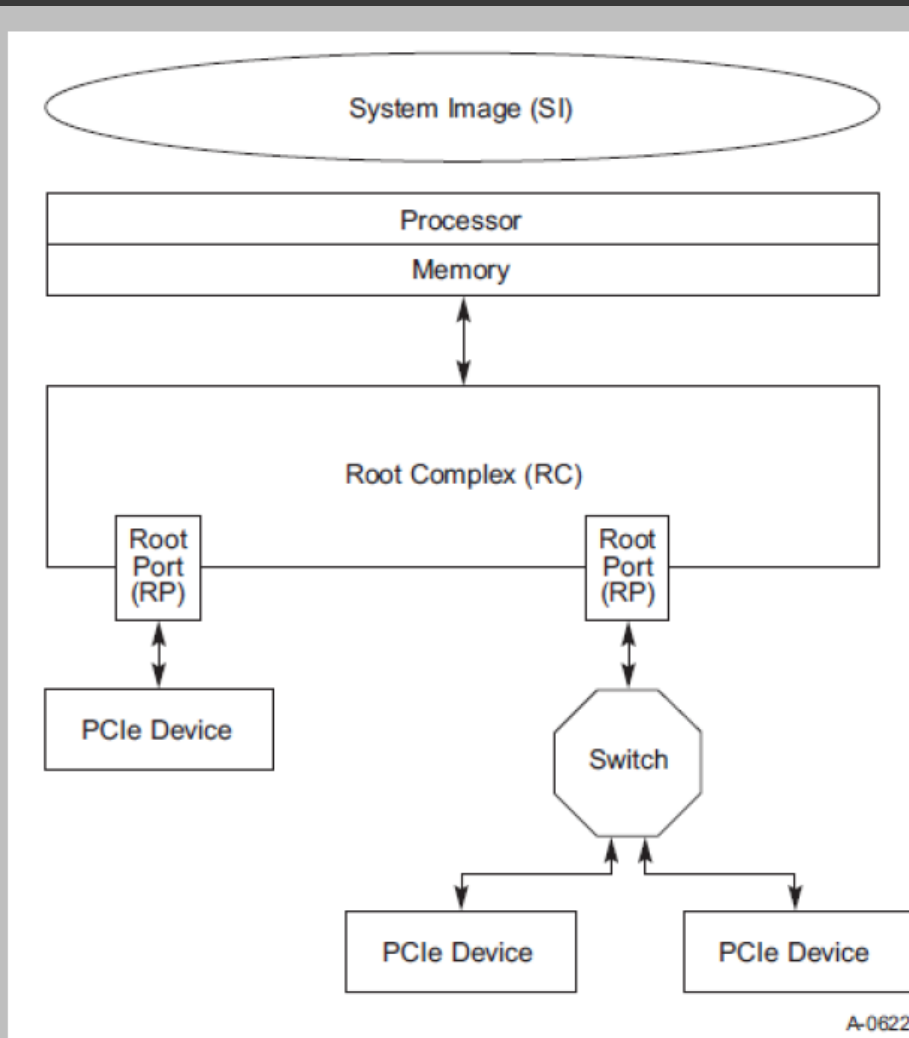
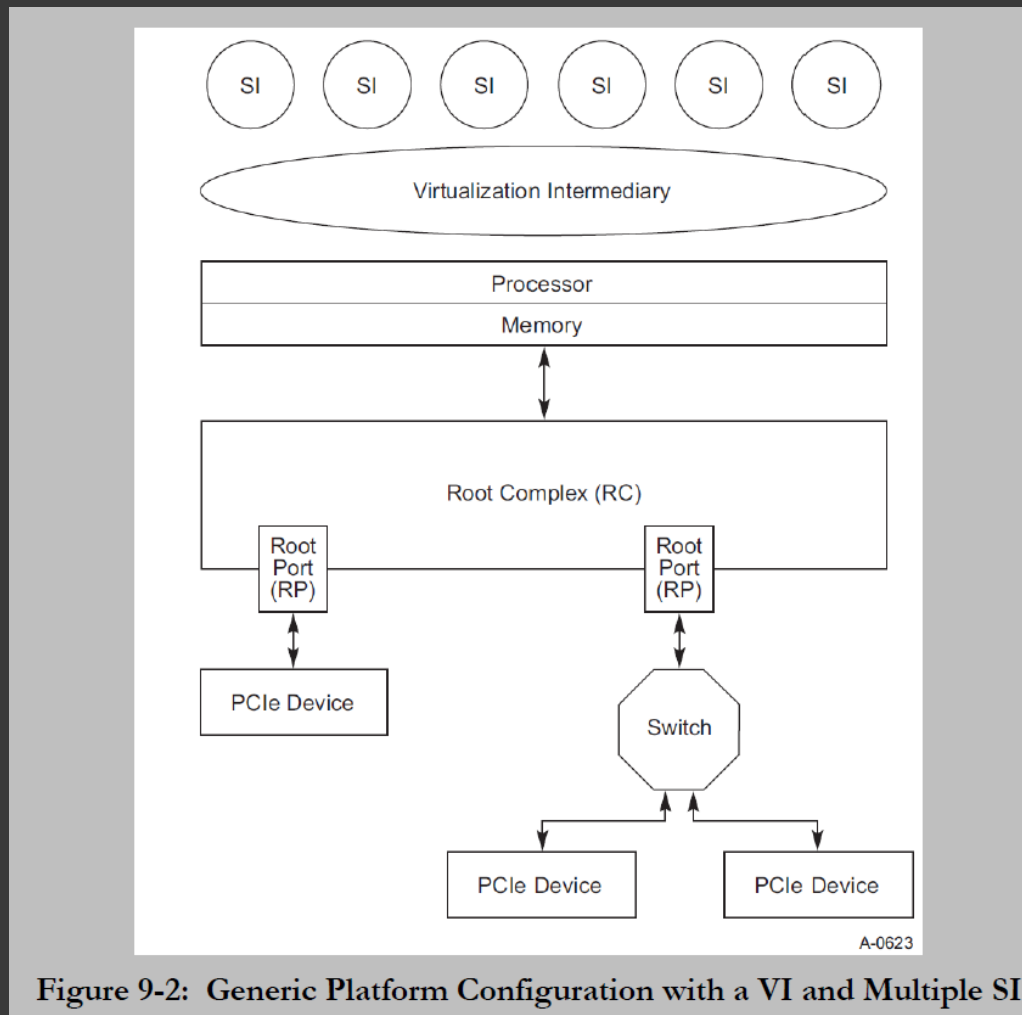
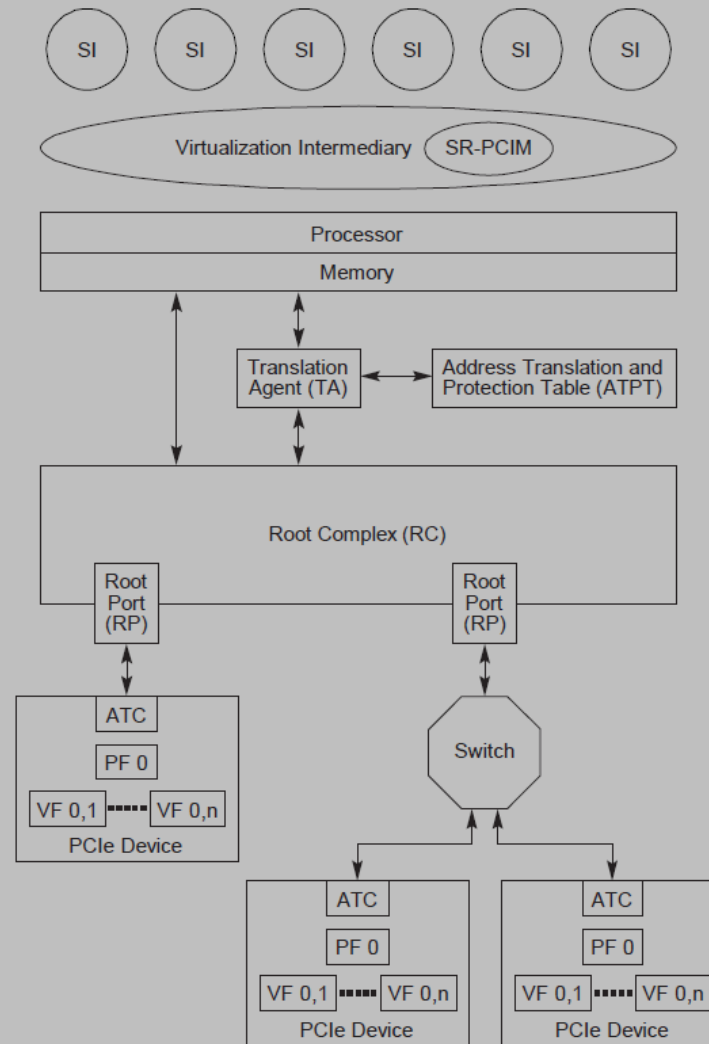


Figure 9-1: Generic Platform Configuration

PCI EXPRESS FABRIC WITH PCIE EXPRESS PHYSICAL FUNCTIONS



PCI EXPRESS WITH SR-IOV ENABLED



A-0624A

Figure 9-3: Generic Platform Configuration with SR-IOV and IOV Enablers

PCI EXPRESS DEVICE WITH MULTIPLE **PHYSICAL** FUNCTIONS

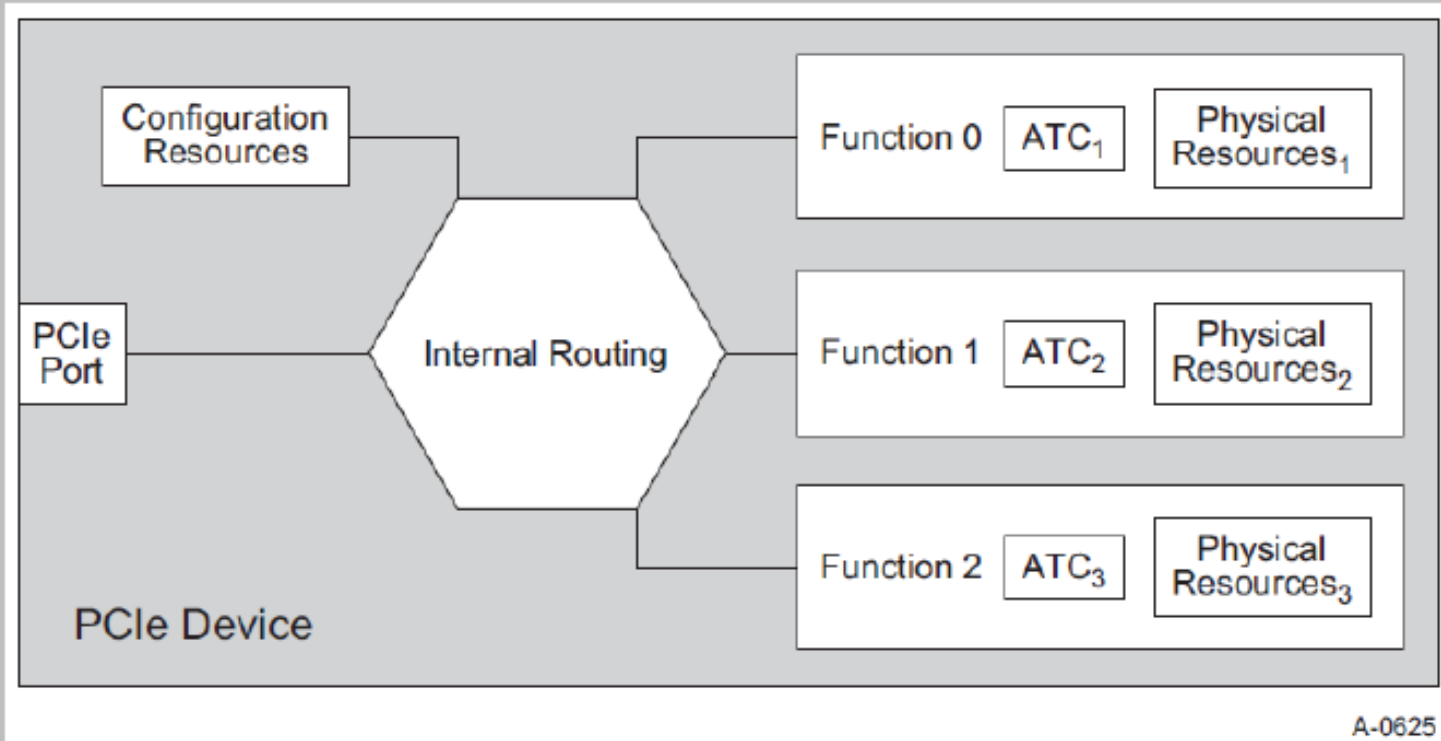
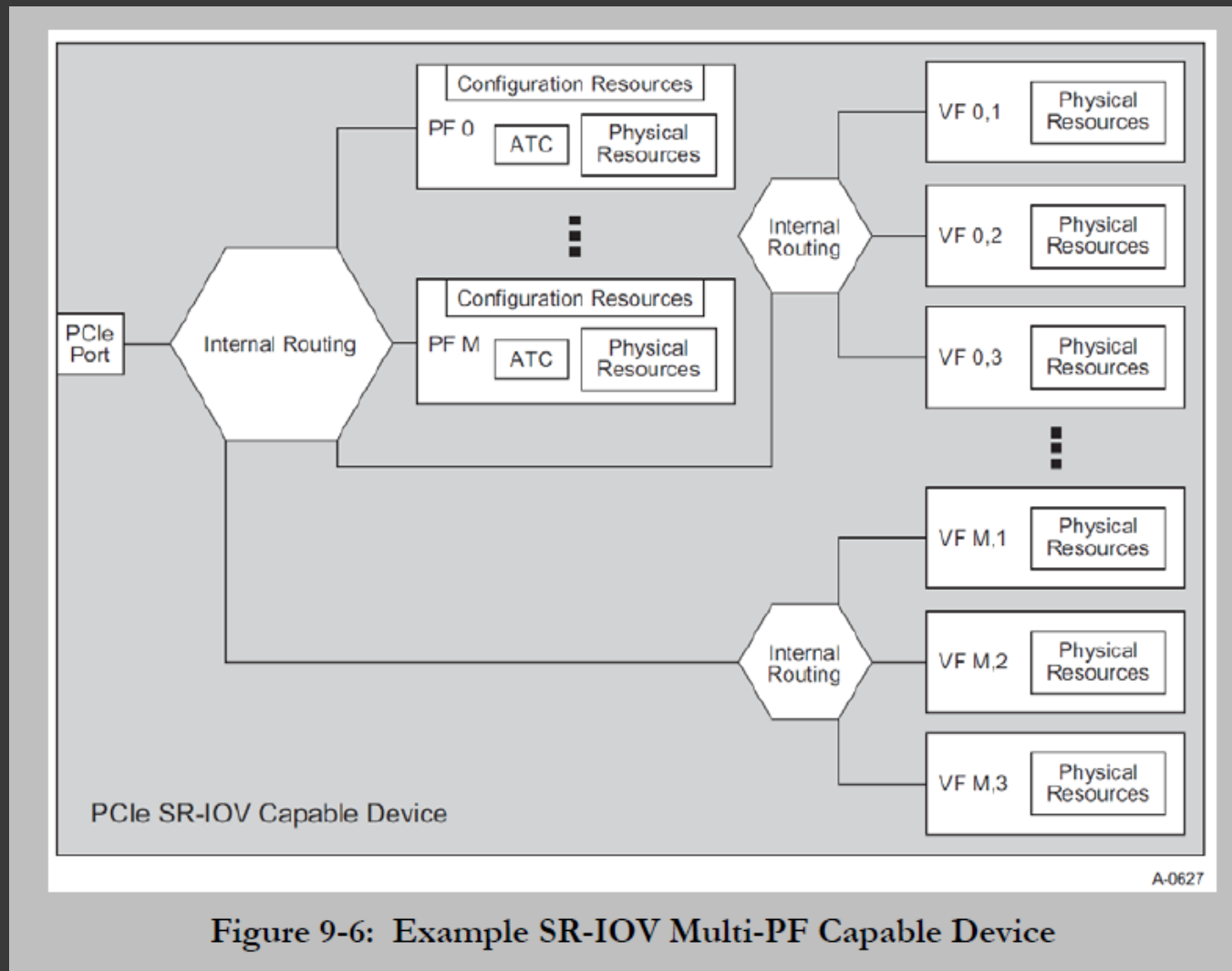


Figure 9-4: Example Multi-Function Device

PCI EXPRESS **SR-IOV** CAPABLE DEVICE WITH **MULTIPLE** PFS (OUR NON-FOCUS DUE TO TIME CONSTRAINTS ☺)



PCI EXPRESS **SR-IOV** CAPABLE DEVICE, WITH **SINGLE** PF (OUR FOCUS FOR SIMPLICITY)

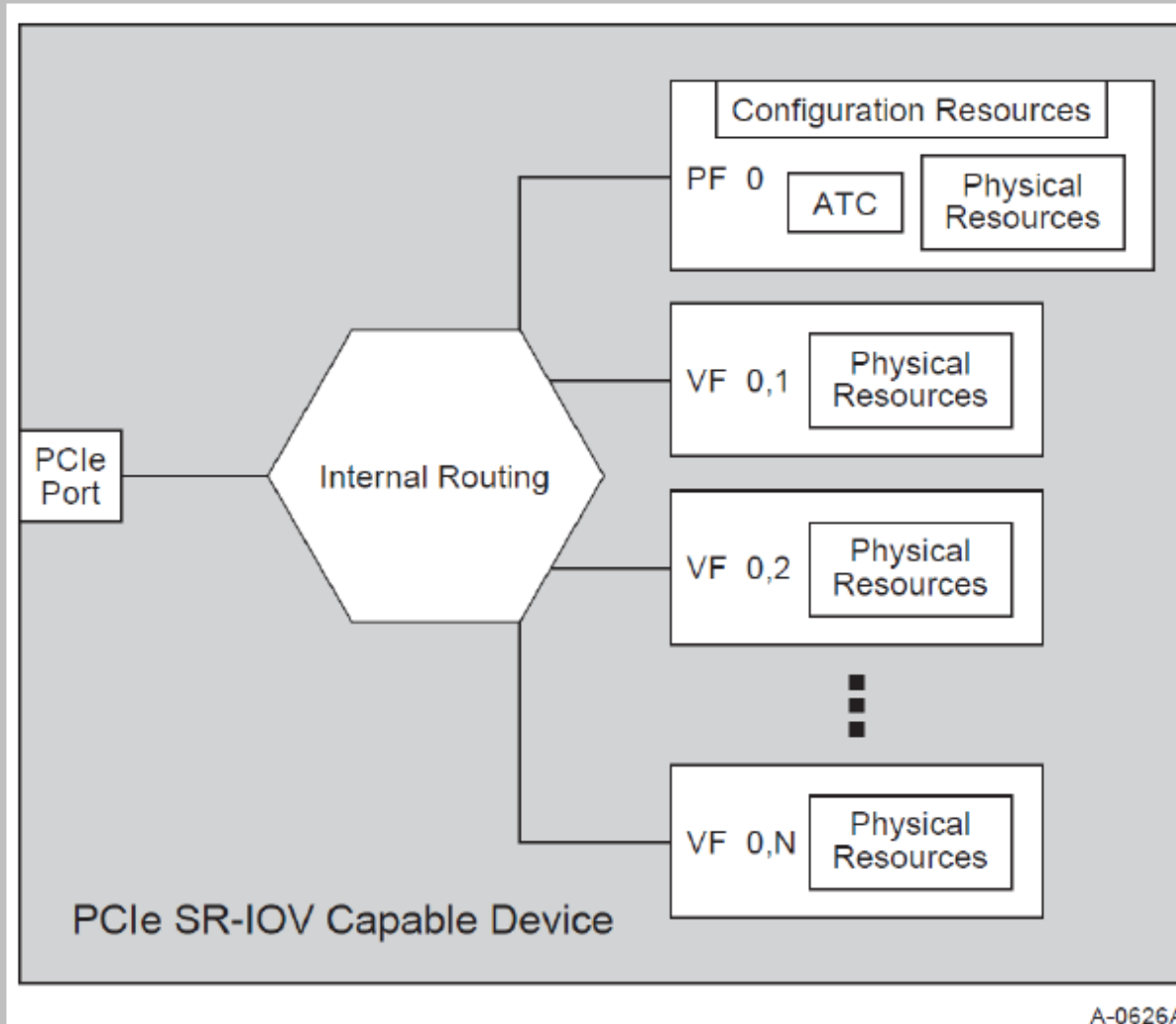


Figure 9-5: Example SR-IOV Single PF Capable Device