
PCI
SIG®

Single Root Resource Discovery and Allocation

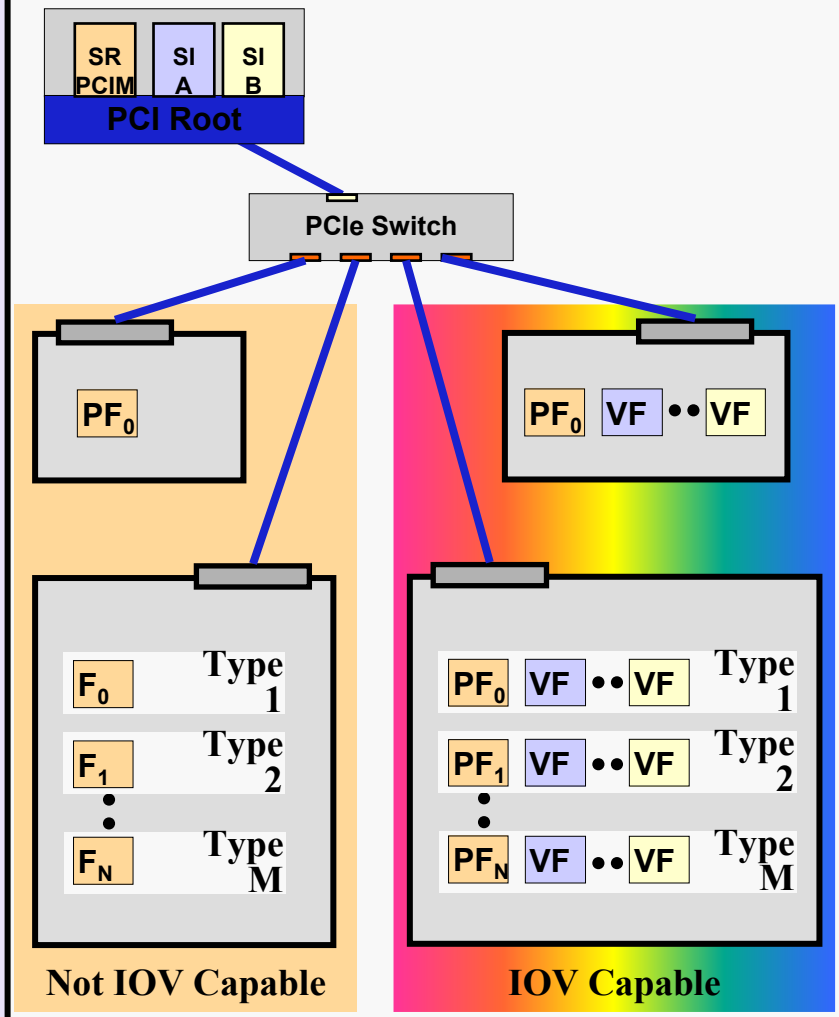
Renato Recio

Contents

- Function types and terms
- Discovering a PCIe Device's SR-IOV Capabilities
- Configuring a PCIe Device's SR-IOV Capabilities
- Discovering the Virtual Functions associated with a configured Physical Function
- Reset and Re-Initialization Mechanisms
- Migration of a Virtual Function across Virtual Hierarchies

Single-Root IOV Function Types and Terms

SR Topology



Terms

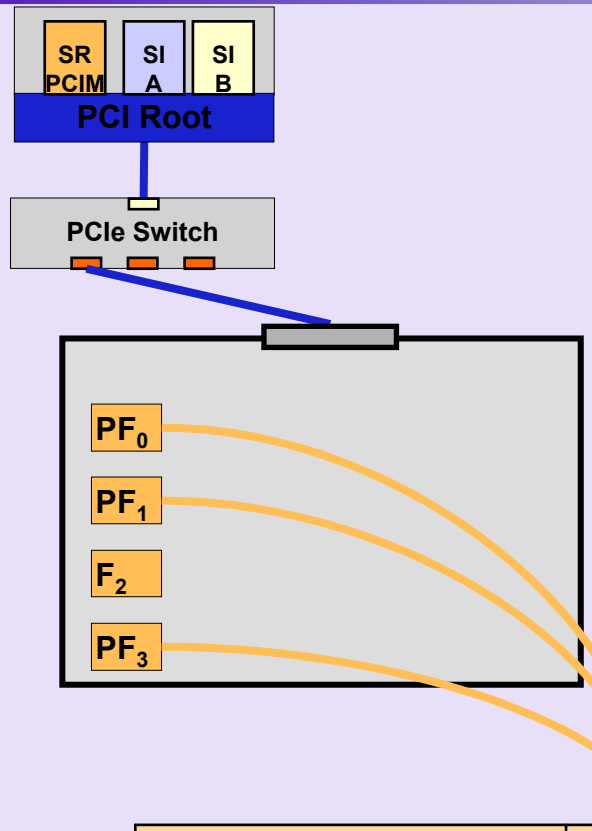
For SR topologies:

Physical Function (PF) is a PCIe Function that supports the SR-IOV capabilities.
A PF is used by SR-PCIM to discover and configure the set of Virtual Functions associated with the PF.

Physical Function 0 (PF_0) is also used to manage Device functions, such as link errors and events.

Virtual Function (VF) is a Function that is associated with a PF.
A VF shares one or more physical resources, such as a link, with other functions (PFs or VFs) on the same Device. A VF supports Native IO Virtualization.

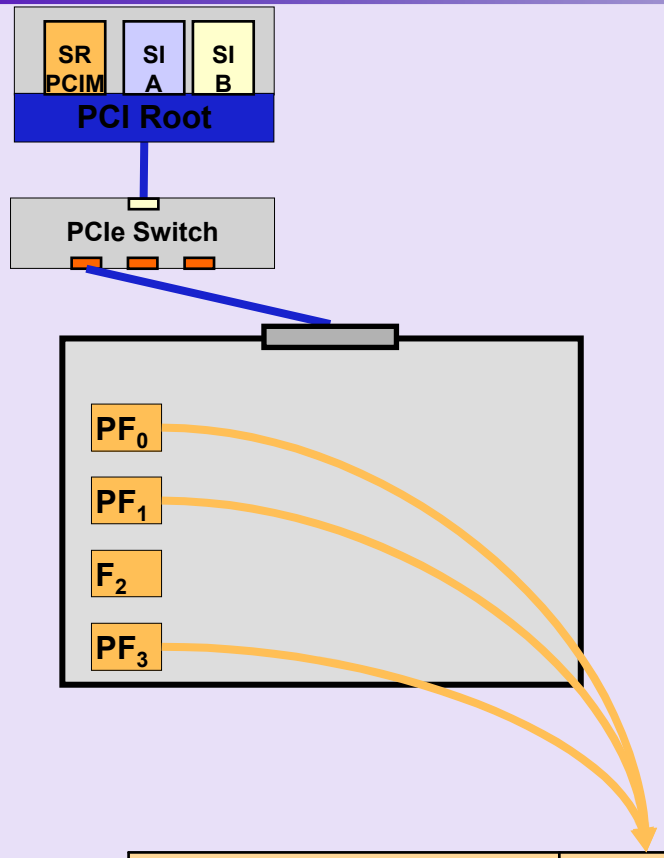
SR-IOV Capability Discovery



- Each PF must have an SR-IOV Extended Capability structure.
- A Device may have a mix of Functions and PFs.
- Each Function and PF consumes one Routing Identifier (RID).
- For a Device that doesn't support ARI, Functions and PFs must be in the first 8 functions.
- For an ARI capable Device the number of Functions and PFs supported is as defined in the base specification.

31 20	19 16	15 0	Offset
Next Capability Pointer	Cap. Vrsn.	Capability ID	00h
SR IOV Capabilities			04h
:			:

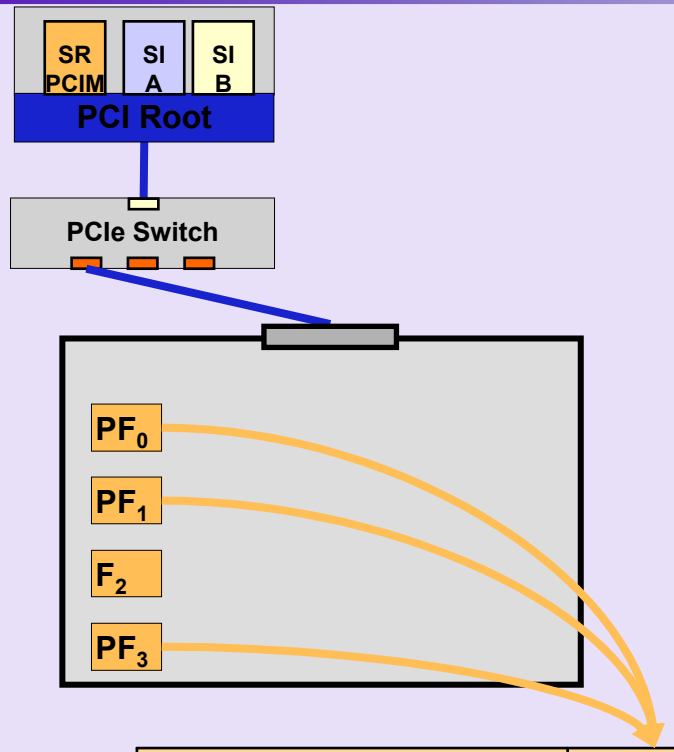
VF Discovery - Part 1



- The MaxVFs and TotalVFs fields are used to discover the maximum number of VFs that can be associated with a PF.
- If the Device does not support VF Migration,
  - ✓ TotalVFs is not applicable.
  - ✓ when MaxVFs is read, the PF must return the number of VFs assigned to the PF.
- If the Device supports VF migration
  - ✓ when TotalVFs is read, the PF must return the number of VFs that **can** be assigned to the PF.
  - ✓ when MaxVFs is read, the PF must return the **initial** number of VFs assigned to the PF.

13	19 16	15 0	Offset
:			:
TotalVFs (RO)		MaxVFs (RO)	0Ch
:			:

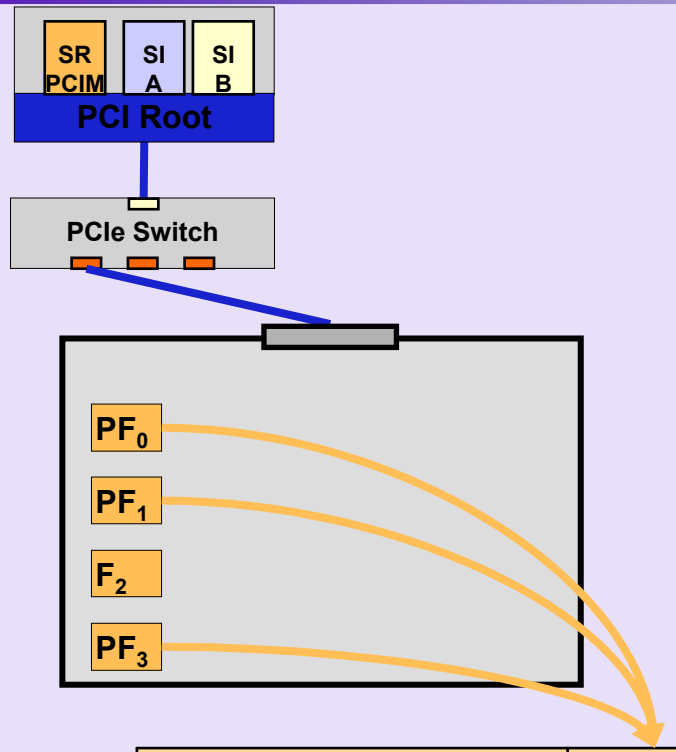
BAR Discovery



- Each PF has its own independent set of BARs in its standard configuration space and an MSE bit for those BARs.
- The VFs share a BAR set *(more later)* and have an MSE bit that controls the memory space of **all** the VFs.
  - ✓ The BAR set that is shared by all the VFs resides in the PF's SR-IOV capabilities

31 20	19 16	15 0	Offset
:			:
VF BAR0 (RW)			20h
:			:
VF BAR5 (RW)			34h
:			:

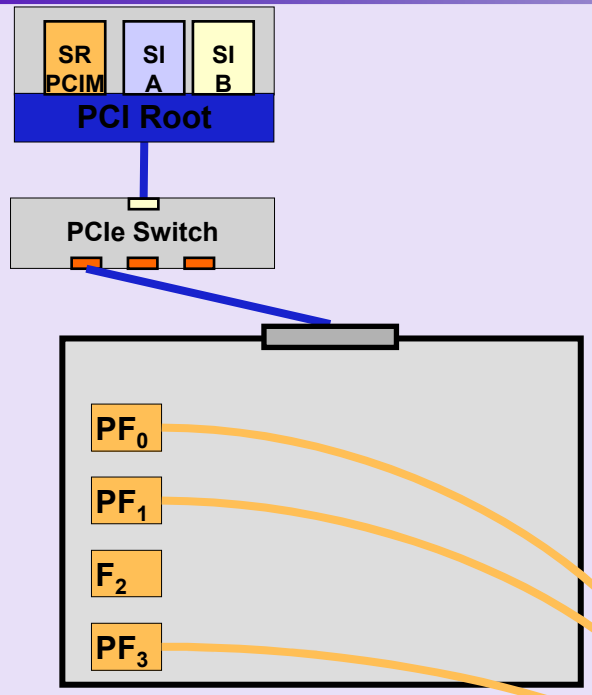
Discovering Supported Page Size



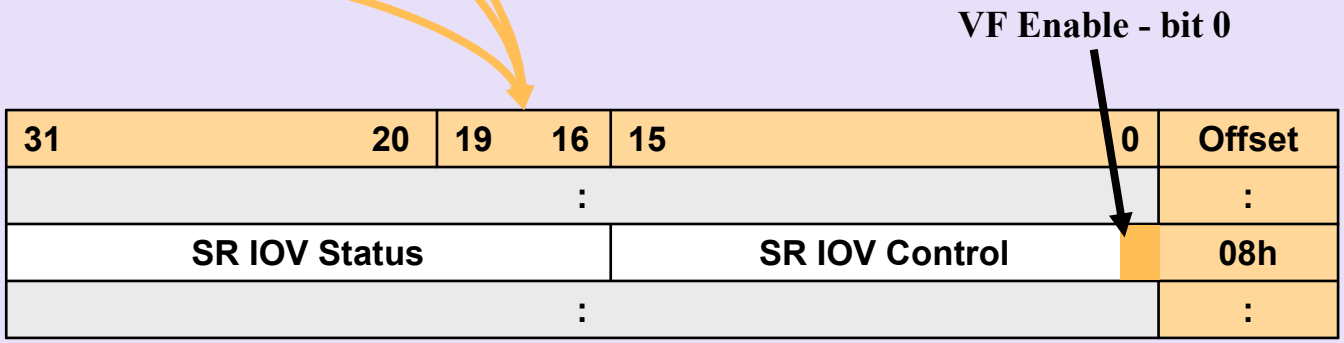
- The Supported Page Sizes field is used to discover the page sizes supported by the PF and its associated VFs.
- When this field is read, the PF must return the page sizes it can support.
- *This field will be used during the IOV configuration phase to align VF BAR apertures on system page boundaries.*

31 20	19 16	15 0	Offset
:			:
Supported Page Sizes (RO)			18h
:			:

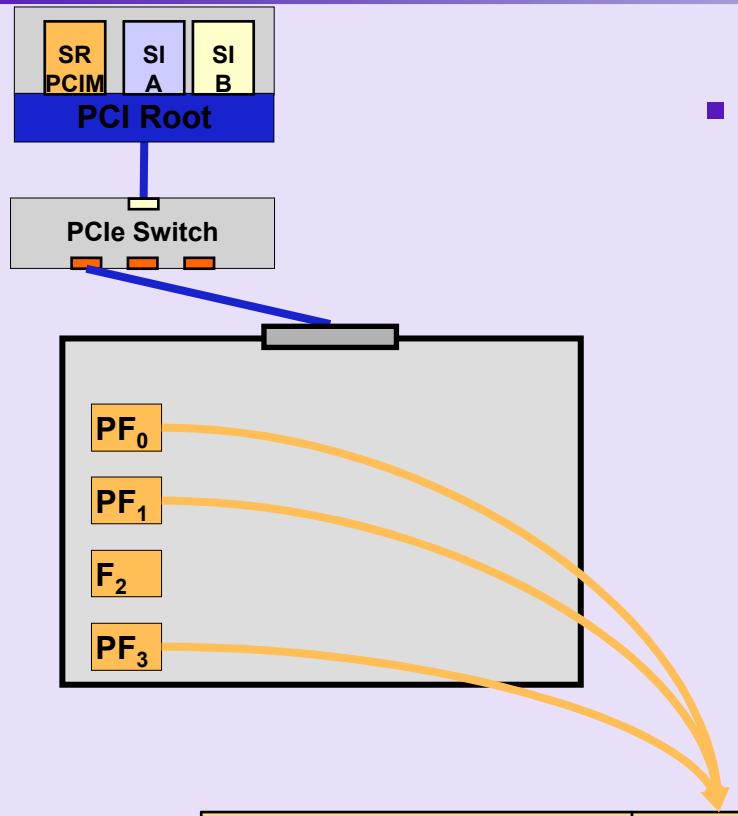
Enabling IOV Capabilities



- The PF's IOV capabilities are enabled by writing a "1" to the "VF Enable" bit of the SR-IOV Control register.
  - ✓ As with the base PCIe Specification, the Device's VFs may return CRS responses to configuration requests after a "1" is written to the VF Enable bit to indicate it is not done reconfiguring its internal resources.



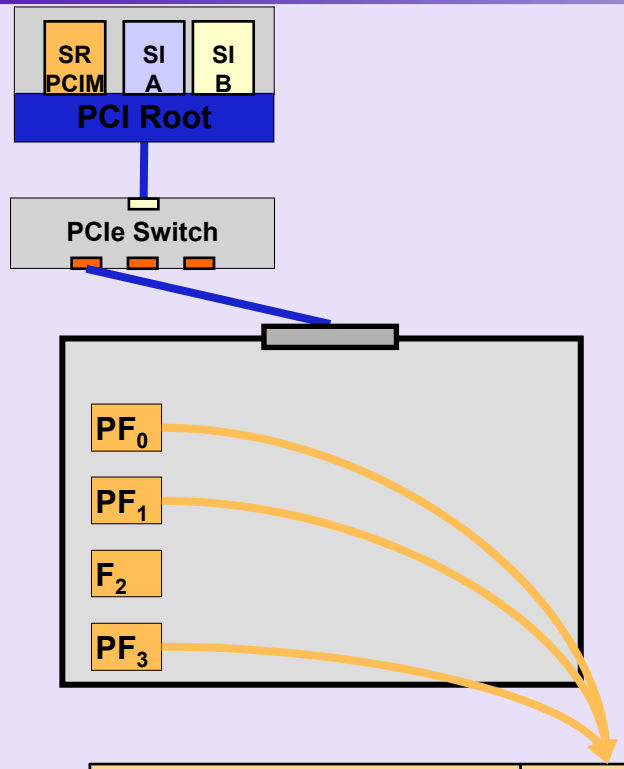
Configuring VFs



- When the PF's IOV capabilities are enabled the PCIe Device must associate Num VFs worth of VFs with the PF.
  - ✓ If the Device does not support VF Migration, then NumVFs must be set to a number between 1 and MaxVFs.
  - ✓ If the Device supports VF Migration, then NumVFs must be set to a number between 1 and Total VFs.

31 20	19 16	15 0	Offset
:			:
Reserved		NumVFs (RW)	10h
:			:

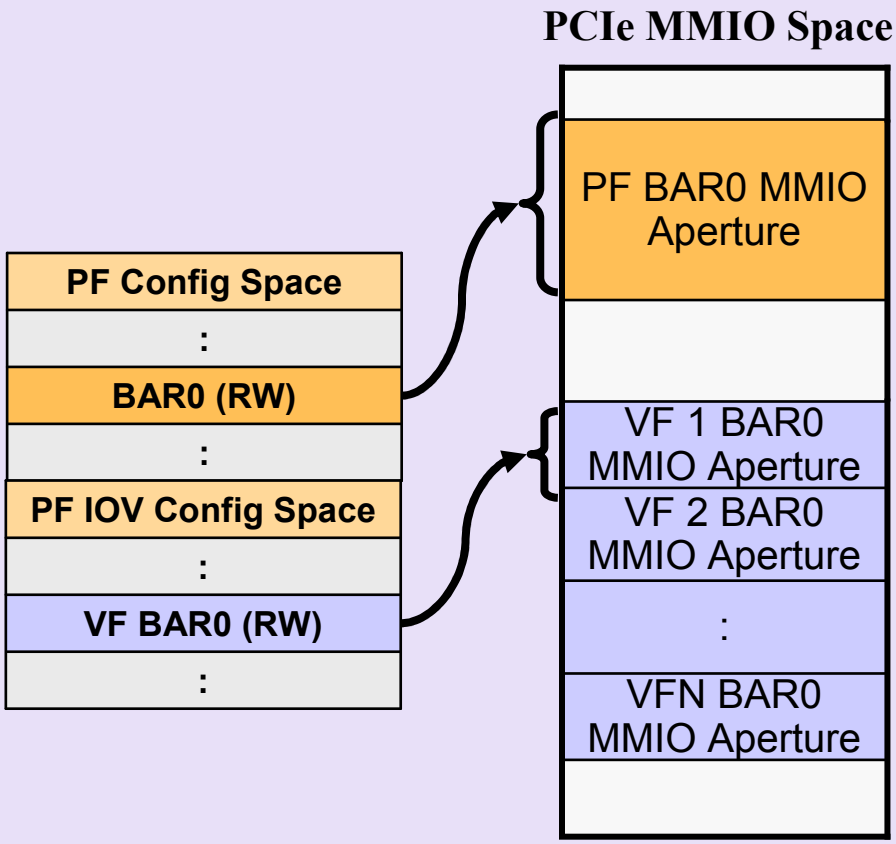
Configuring the VF's BARs



- The System Page Size field defines the page size that will be used by the system.
  - ✓ The System Page Size value must be one of the Supported Page Sizes.
  - ✓ The System Page Size field is used by the Device to align the MMIO aperture defined by each BAR to a system page boundary.
- The behavior of VF BARs is the same as the PCI 3.0 Spec's normal PCI BARs, except that a VF BAR describes the aperture for **multiple VFs**, whereas a PCI BAR describes the aperture for a single function *(more next).*

31 20	19 16	15 0	Offset
:			:
VF BAR0 (RW)			20h
:			:
VF BAR5 (RW)			34h
:			:

PF and VF BAR Semantics

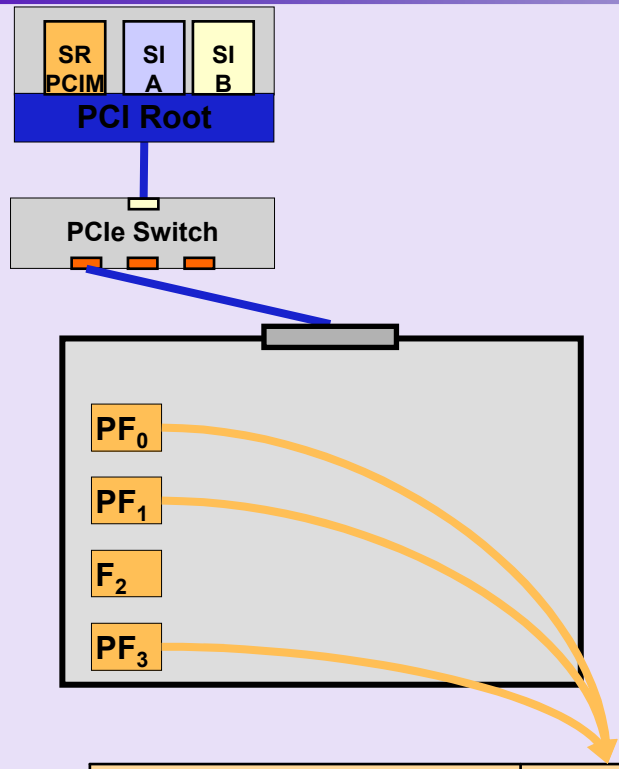


BAR1 VF N SA = BAR1 VF 1 SA + N x (BAR1 VF MA) - 1

where BAR1 VF 1 SA is the address written into VF BAR1 and MA is the memory aperture of VF BAR1.

- The size of the memory aperture required for each VF BAR can be determined by writing all "1"s and then reading the VF BAR.
  - ✓ The results are as described in the base PCI Spec.
- The address written into each VF BAR is used by the Device to set the starting address for that BAR on the **first VF**.
- The differences between VF BARs and PCI 3.0 Spec BARs are:
  - ✓ For each VF BAR, the memory space associated with the second and higher VFs is derived from the starting address of the first VF and the memory space aperture.
  - ✓ The VF BAR's MMIO space is not enabled until VF Enable and VF MSE have been set.

VF Discovery - Part 2



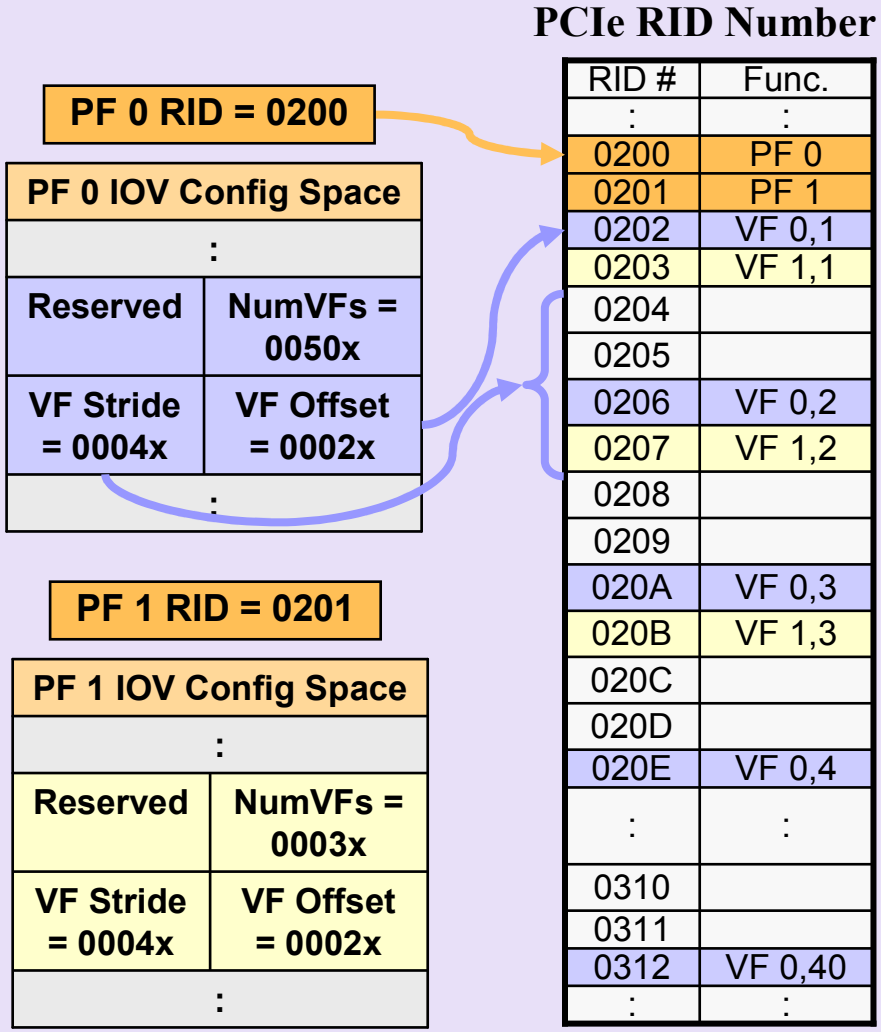
- After the PF's IOV capabilities are enabled, the VFs associated with a PF are discoverable by reading the First VF Offset and VF Stride.
- Following is the equation for determining the RID of VF N:

  VF N = [PF RID + VF Offset + (N-1) * VF Stride] Modulo 2^{16}

  where all arithmetic used is 16 bit unsigned dropping all carries.
- Device defines VF Stride and First VF Offset.
- *The next page describes an example.*

31 20	19 16	15 0	Offset
:			:
VF Stride (RO)		First VF Offset (RO)	14h
:			:

PF and VF RID Semantics



- The RID of any PF or VF must not overlap with the RID of any other PF or VF given any valid setting of NumVFs across all PFs of a device.
- VFs may reside on a different bus number as associated PF.
- As in base, an SR-IOV Device captures bus # from any Type 0 config request.
- If the switch above the Device supports ARI Forwarding bit, RIDs are interpreted as 8 bit Bus # and 8 bit Device #.
- If the switch above the Device doesn't supports ARI, RIDs are interpreted as BDF#.
- Bus # can be used to assign VFs, but all PFs must be located on the first Bus # assigned to a Device.

Reset Mechanisms

Three reset mechanisms are supported:

- Conventional Reset - Resets all PF and VF state.
- FLR that targets a VF - Resets a single VF.
- FLR that targets a PF - Resets a PF and its associated VFs.

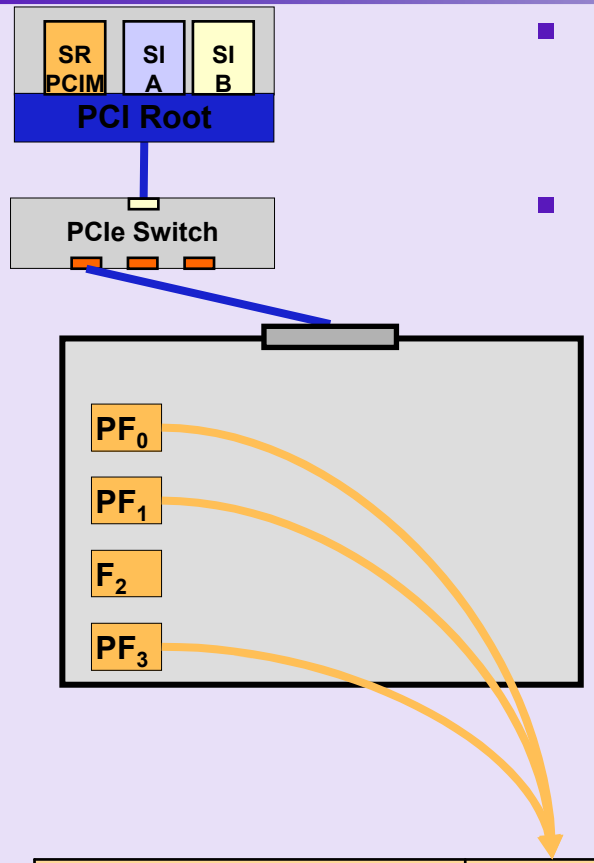
Conventional Reset

- A Fundamental or Hot Reset to an SR-IOV Device shall cause all Functions, PFs, and VFs context to be reset to their original, power-on state.
- If a PF has its SR-IOV capabilities enabled and a Fundamental or Hot Reset is issued to the Device, the Device must reset all PF and VF state:
  - ✓ The PF must disable its SR-IOV capabilities and reverts back to being a PCI Function.
  - ✓ Settable SR-IOV capabilities (e.g. NumVFs) are reset to default values.
  - ✓ MSE and BME are both off.
  - ✓ All BAR values used by the PF and VFs are indeterminate.
  - ✓ All interrupts are disabled.

FLRs to VFs and PFs

- An FLR that targets a VF must be supported.
  - ✓ Software may use FLR to reset a VF.
  - ✓ An FLR that targets a VF must:
    - Not affect the VFs existence (e.g. it still consumes a RID)
    - Not affect any address assigned to it.
      That is, the VF's BAR registers and MSE are unaffected by FLR.
- An FLR that targets a PF must be supported.
  - ✓ Software may use an FLR to reset a PF.
  - ✓ An FLR that targets a PF must:
    - Reset the PF's SR-IOV Extended Capabilities.
    - The VFs are no longer allocated or enabled.

IOV Re-initialization and Reallocation

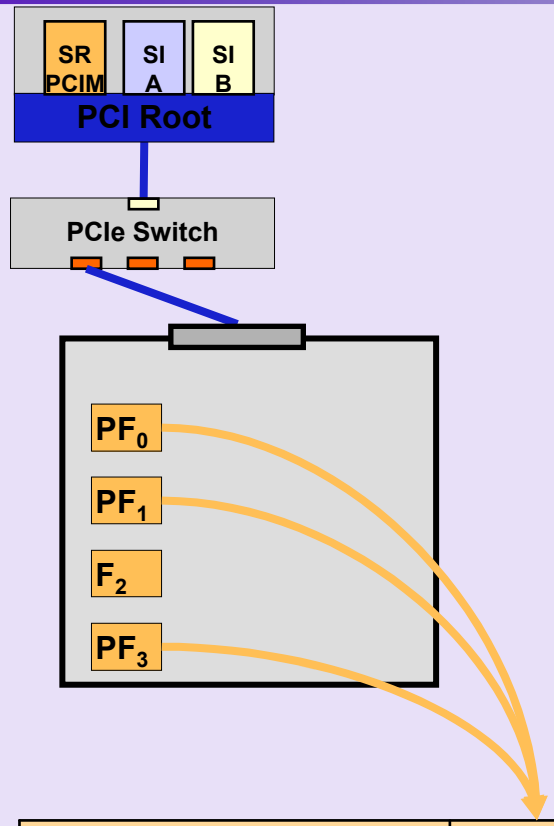


- If a PF has its IOV capabilities enabled, writing a "0" to "VF Enable" shall disable the PF's IOV capabilities.
- After the PF's IOV capabilities are disabled:
  - ✓ The VFs must no longer:
    - Issue PCIe transactions,
    - Exist in the RID space,
    - Exist in the memory mapped address space,
    - Retain any context, or
    - Log any new VF errors.
  - ✓ Any errors already logged via PF error reporting registers, remain logged.

31 20	19 16	15 0	Offset
:			:
SR IOV Status		SR IOV Control	08h
:			:

Writing "0" to VF Enable - bit 0

VF Migration



- VF Migration does not occur in SR systems.
- VF Migration is an optional Device feature defined to support the migration of VFs between Virtual Hierarchies.
  - ✓ If the Device is in an MR topology and migration is enabled, the “VF Migration Capable” bit will be set in the SR-IOV Capabilities Register.
- VF Migration must only occur if it has been enabled through the PF’s SR-IOV capabilities by SR software.
  - ✓ If migration operations are disabled in the PF’s SR-IOV capabilities, VFs are always Active, and the VF State array is undefined.

31 20	19 16	15 0	Offset
:			:
SR IOV Capabilities			04h
SR IOV Status		SR IOV Control	08h
:			:

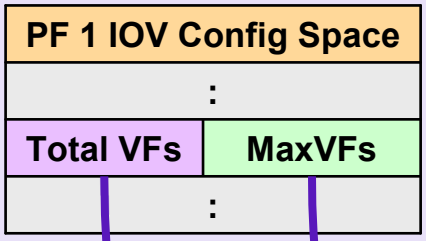
VF Migration Capable - bit 0

VF Migration Enable - bit 1

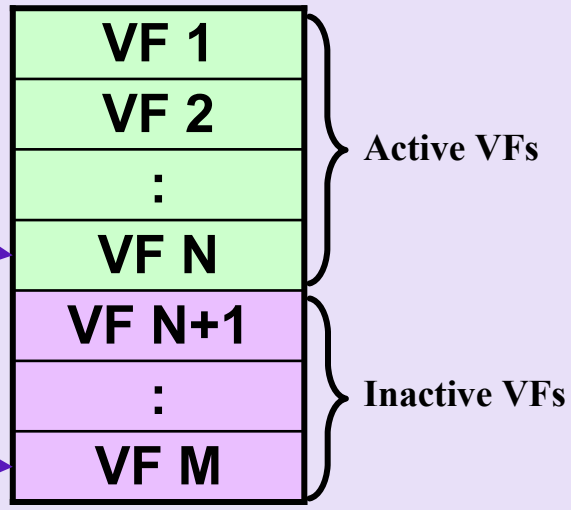
VF Migration Software Requirements

- Software must determine if a VF is *Active* or *Inactive*.
  - ✓ A VF that is active must be available for use by SR.
    - An active VF must have its PCI configuration and memory mapped address space accessible.
  - ✓ An Inactive VFs must not be available for use by SR.
    - An inactive VF consumes PCI configuration and memory mapped address space, but cannot be accessed in either space.
- Software must participate in *Migrate In* operations.
  - ✓ A Migrate In operation is initiated by MR-PCIM and is used to activate an Inactive VF making it usable by SR.
  - ✓ SR software may either Accept or Refuse a Migrate In operation.
- Software must participate in *Migrate Out* operations.
  - ✓ A Migrate Out operation is used to deactivate an Active VF.
  - ✓ This supports the graceful removal of a VF from use by SR.
  - ✓ SR Software may either Accept of Refuse the Migrate Out.
- Software must handle *Forced Remove* operations.
  - ✓ A Force Remove is the ungraceful Deactivation of an Active VF.
  - ✓ This is the VF migration equivalent of a Surprise Down in base PCIe.

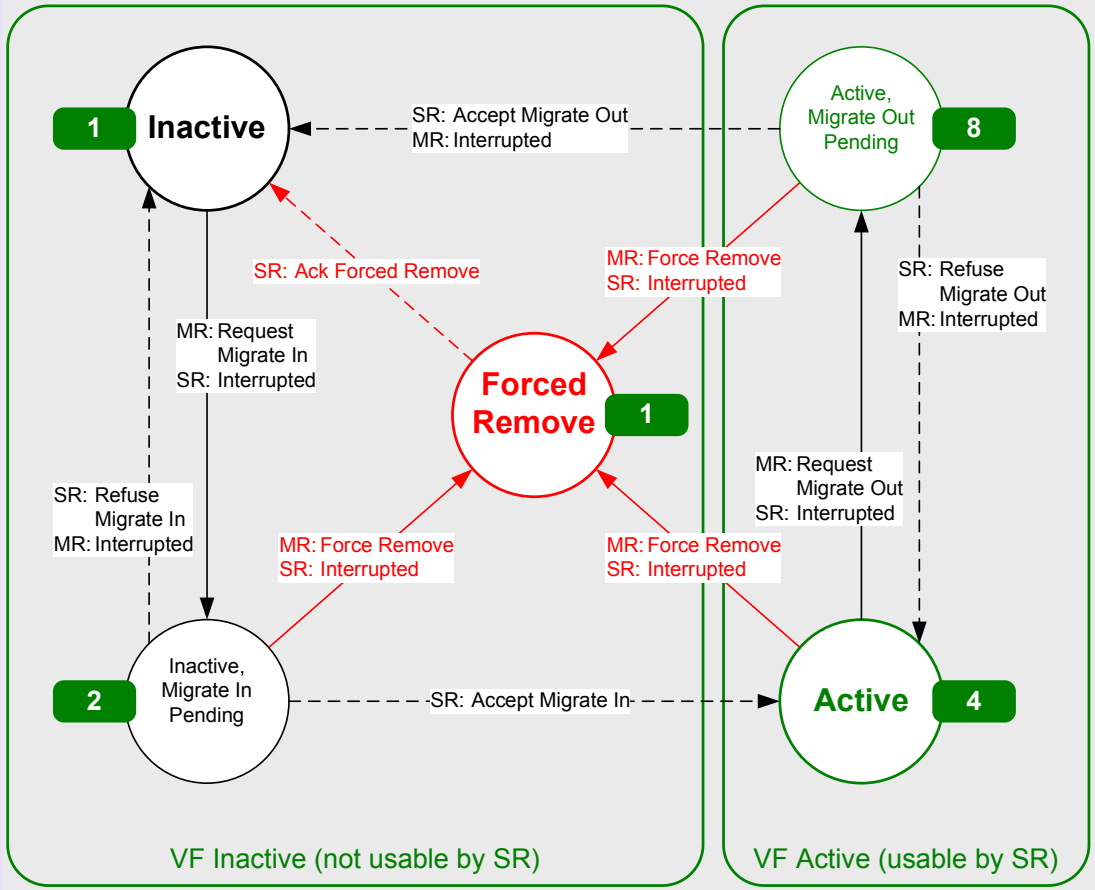
Initial VF Migration State



- If a Device is capable of VF Migration and has enabled SR-IOV VF Migration, the initial VF states are:
  - ✓ VF 1 thru MaxVFs initialize in the Active state.
    - For the example below VF 1 through VF N
  - ✓ VF MaxVFs + 1 through TotalVFs initialize in the Inactive state.
    - For the example below VF N+1 through VF M



VF Migration State Diagram

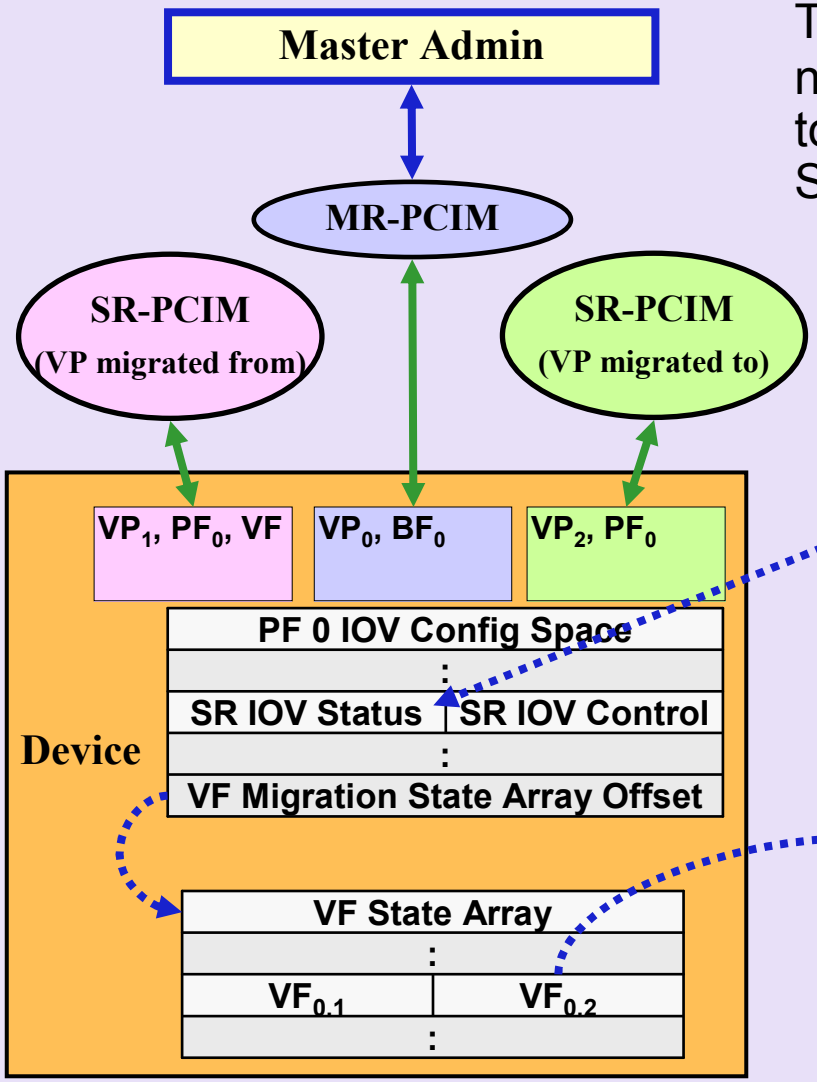


- When SR-IOV capabilities are enabled, the initial VF states are:
  - ✓ Active
  - ✓ Inactive

Current State	Written State	Meaning
2	4	SR Accepts Migrate In
2	1	SR Refuses Migrate In
8	1	SR Completes Migrate Out
8	4	SR Refuses Migrate Out
any	0	No State Transition

- MR-PCIM initiates state transitions indicated by solid lines.
- SR-PCIM initiates state transitions indicated by dashed lines by writing the new state value to the VF State Array.

VF Migration Example



The next two slides will use the following nomenclature for the communications necessary to migrate a VF from one SR-PCIM to another SR-PCIM:

- ↔ HAL with SR-PCIMs and MR-PCIM
- ↔ Dev with SR-PCIMs and MR-PCIM

VF Migration Interrupt Pending (bit 0)

Used to indicate a VF Migration In or Migration Out Request has been issued by MR-PCIM.

SR-PCIM must read the VF State Array (see next 2 slides) to determine which VF(s) are being migrated in or out.

VF State	VF Active	Description
0001	No	**Inactive VF**
0010	No	**VF Migrated In Request Pending**
0100	Yes	**Active VF / PF**
1000	Yes	**VF Migrated Out Request Pending**

VF *Migrate Out* Example

5) VF Migrate Out interrupt to SR-PCIM, SR-PCIM informs SI of Migrate Out. SI accepts Migrate Out.

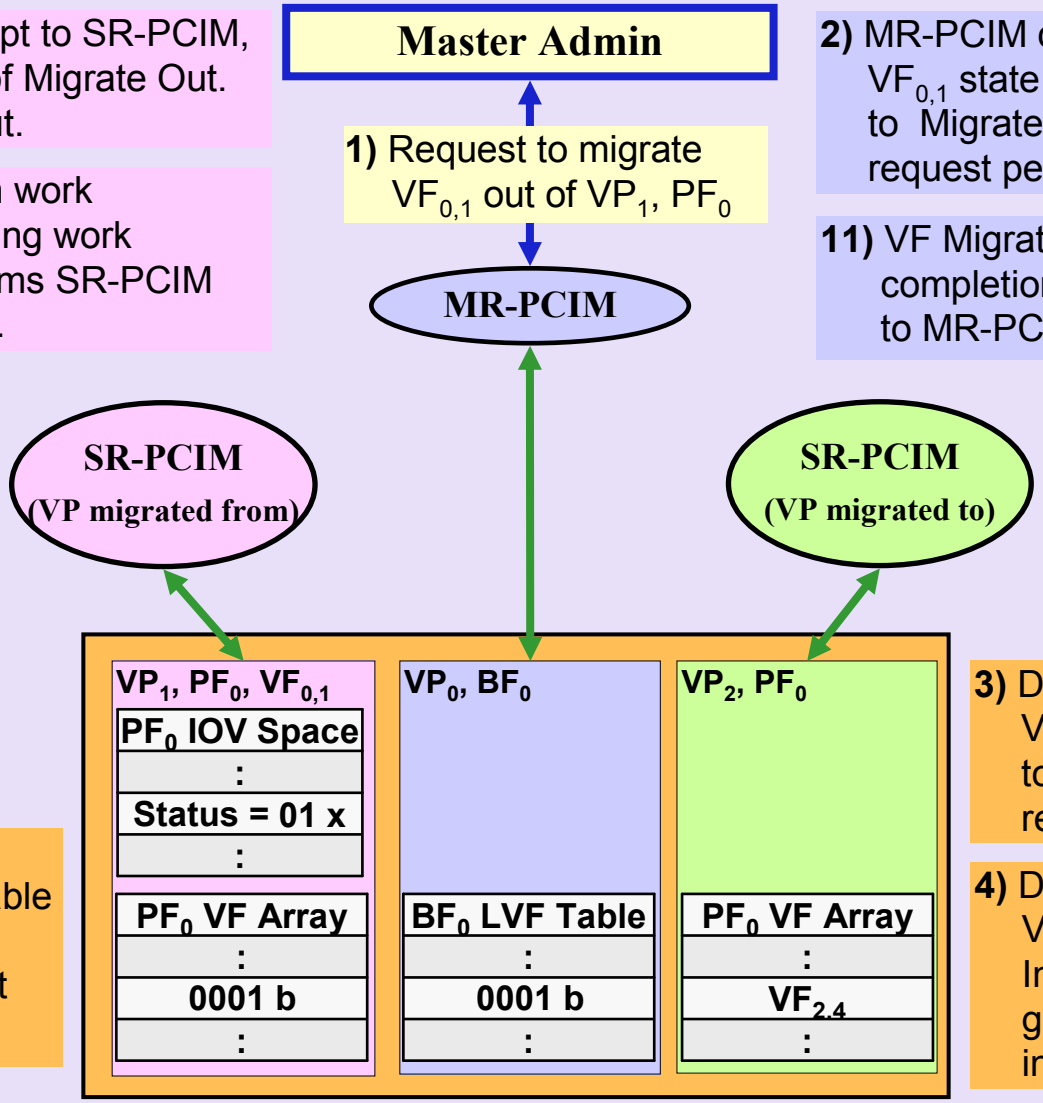
6) SI performs shut down work (e.g. wait for outstanding work to complete) and informs SR-PCIM VF is no longer in use.

7) SR-PCIM issues FLR to reset $VF_{0,1}$.

9) SR-PCIM changes $VF_{0,1}$ state to Inactive VF.

8) FLR Resets VF_0.

10) Device changes $VF_{0,1}$ state in LVF Table and generates VF Migration Interrupt Pending on BF_0.



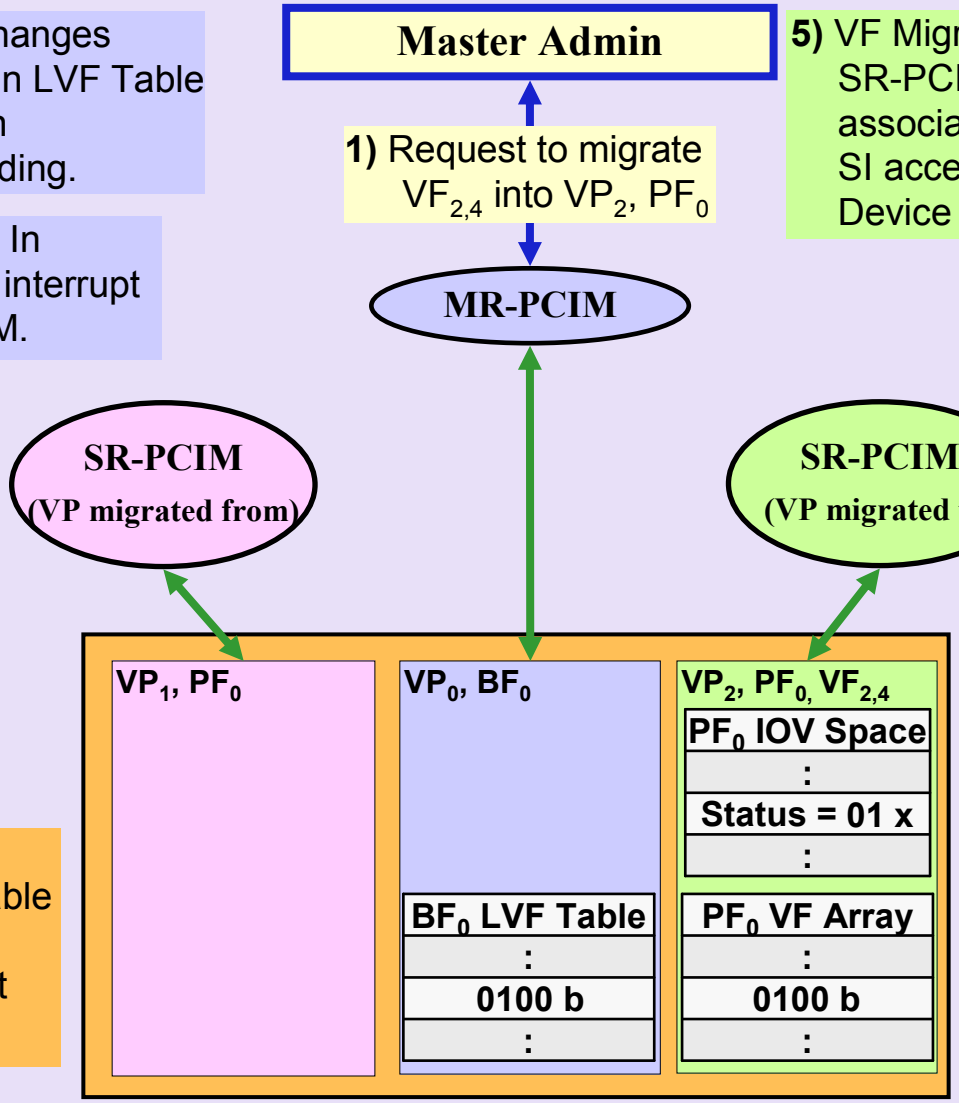
2) MR-PCIM changes $VF_{0,1}$ state in LVF Table to Migrate Out request pending.

11) VF Migrate Out completion interrupt to MR-PCIM.

3) Device changes $VF_{0,1}$ state of VP_1, PF_0 to Migrate Out request pending.

4) Device sets VF Migration Interrupt Pending and generates VF Migrate Out interrupt on VP_1, PF_0.

VF *Migrate In* Example



2) MR-PCIM changes $VF_{2,4}$ state in LVF Table to Migrate In request pending.

11) VF Migrate In completion interrupt to MR-PCIM.

5) VF Migrate In interrupt to SR-PCIM, SR-PCIM informs SI that is to be associated with VF of Migrate In. SI accepts Migrate In and "discovers" Device (through SR-PCIM).

7) SR-PCIM changes $VF_{2,4}$ state to Active VF.

9) SR-PCIM issues FLR to reset $VF_{2,4}$.

8) FLR Resets $VF_{2,4}$.

10) Device changes $VF_{2,4}$ state in LVF Table and generates VF Migration Interrupt Pending on BF_0.

3) Device changes $VF_{2,4}$ state of VP_2, PF_0 to Migrate In request pending.

4) Device sets VF Migration Interrupt Pending and generates VF Migrate In interrupt on VP_2, PF_0.


PCI
SIG®